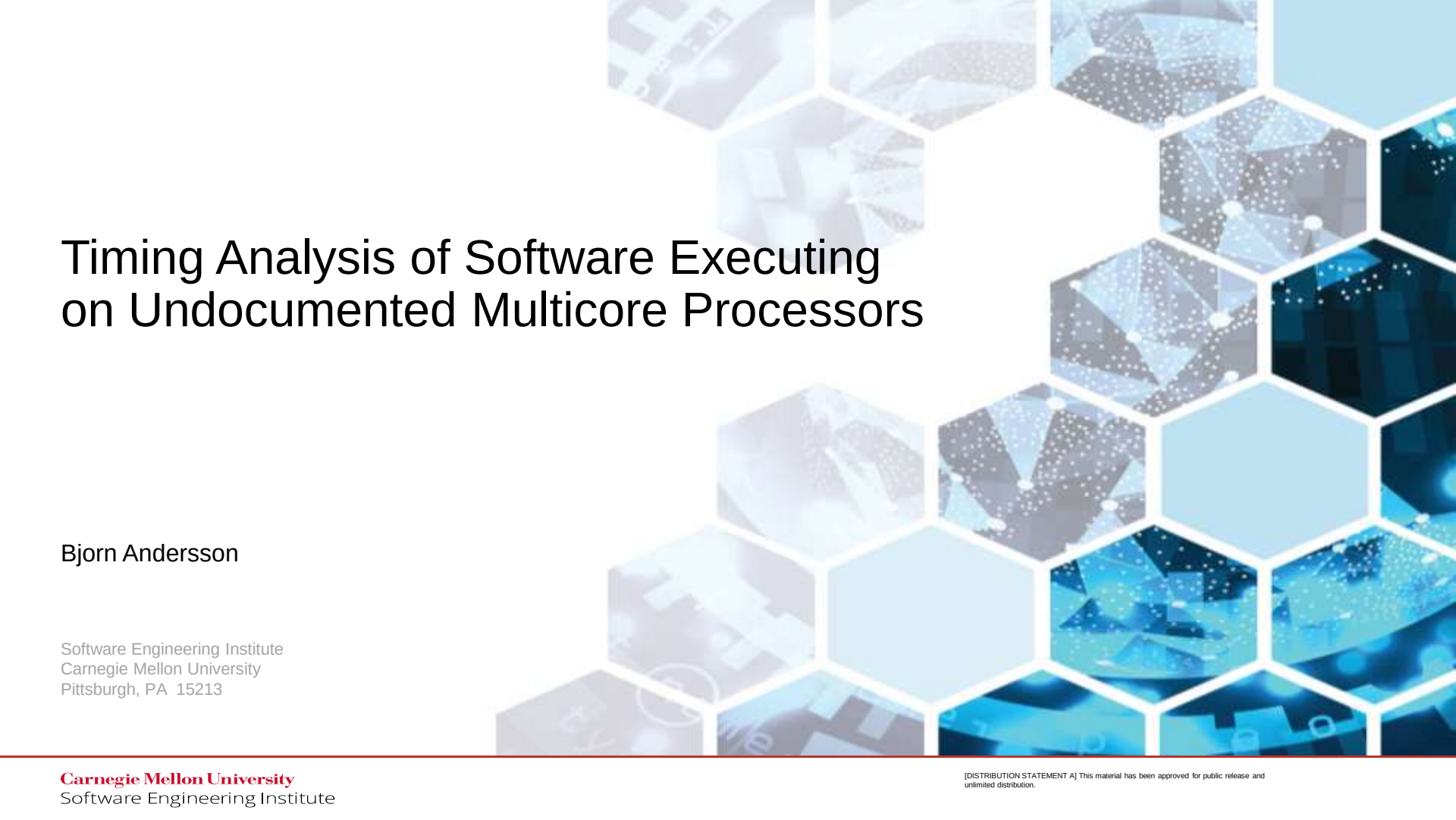




Timing Analysis of Software Executing on Undocumented Multicore Processors

Bjorn Andersson

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Copyright 2020 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM20-0755

Systems Interact with Their Physical Environment



Systems Include Software



Systems Include Software That Interacts with the Physical Environment



Systems Include Software That Has Real-Time Requirements



Satisfying Real-Time Requirements is a Challenge for These Systems in General



Satisfying Real-Time Requirements is a Challenge for ...



“The trick there, when you’re processing flight critical information, it has to be a deterministic environment, meaning we know exactly where a piece of data is going to be exactly when we need to — no room for error,” Langhout says. “On a multi-core processor there’s a lot of sharing going on across the cores, so right now we’re not able to do that.”

- Jeff Langhout, Acting Director, U.S. Army Aviation and Missile Research
Development and Engineering Center (AMRDEC)

Source: “Army still working on multi-core processor for UH-60V,” May 2017, Available at
<https://www.flightglobal.com/news/articles/army-still-working-on-multi-core-processor-for-uh-6-436895/>.

Satisfying Real-Time Requirements is a Challenge for ...



“A majority of avionics today are running on single-core processors, or multicore processors with all but one core disabled.”

“The root of the problem is shared resources, which most of the time creates some kind of interference.”

Source: “It’s time: Avionics need to move to multicore processors,” January 2018, Available at <http://www.intelligent-aerospace.com/articles/2018/01/it-s-time-avionics-needs-to-move-to-multicore-processors.html>.

Satisfying Real-Time Requirements is a Challenge for ...



“In safety-critical domains such as avionics, the multicore “predictability problem” is currently dealt with by turning off all but one core if highly-critical system components exist.”

Source: C. J. Kenna et al., “Making Shared Caches More Predictable on Multicore Platforms,” RTSS’2012.

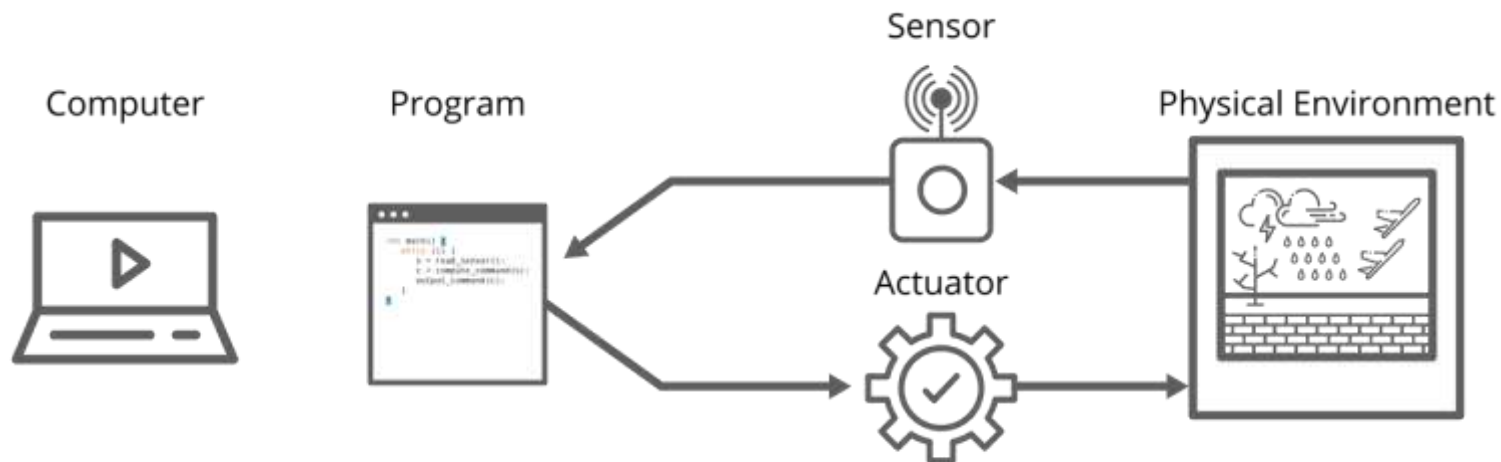
Satisfying Real-Time Requirements is a Challenge for ...



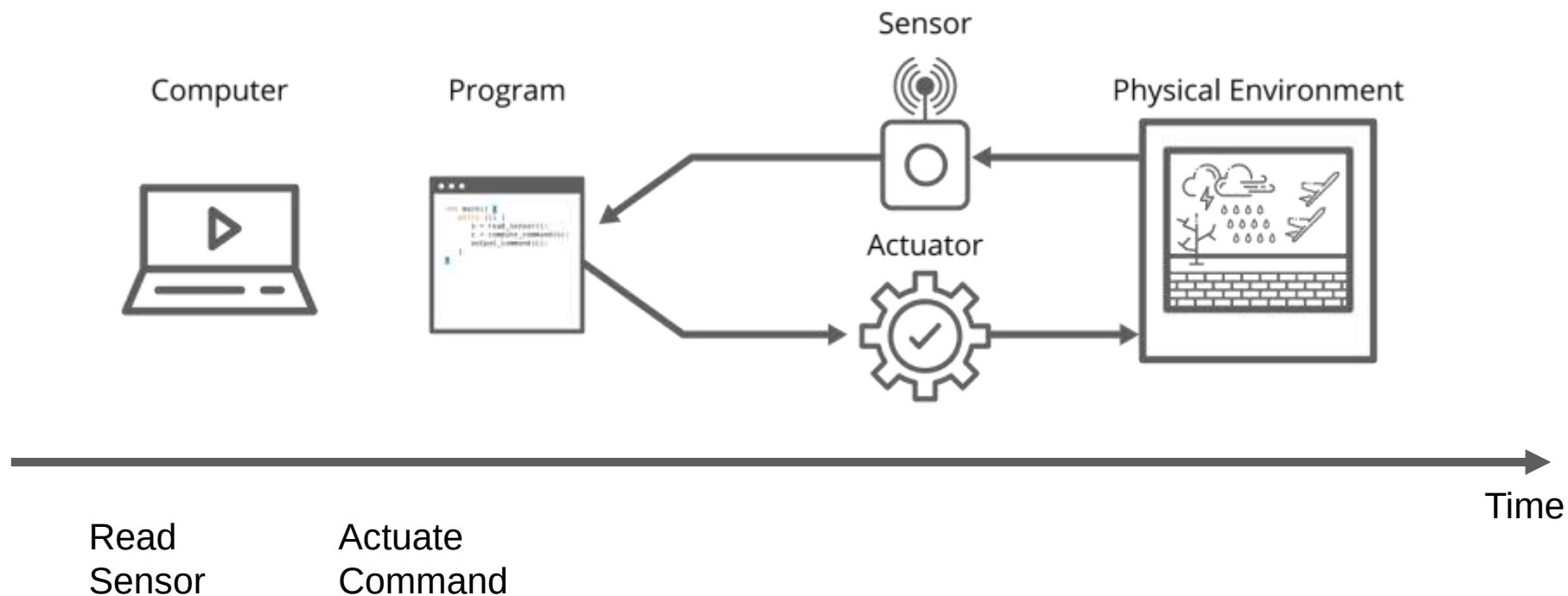
“Currently, avionics manufacturers resolve the multicore “predictability problem” by turning off all but one core if highly critical system components exist.”

Source: B. C. Ward et al., “Making Shared Caches More Predictable on Multicore Platforms,” ECRTS’2013.

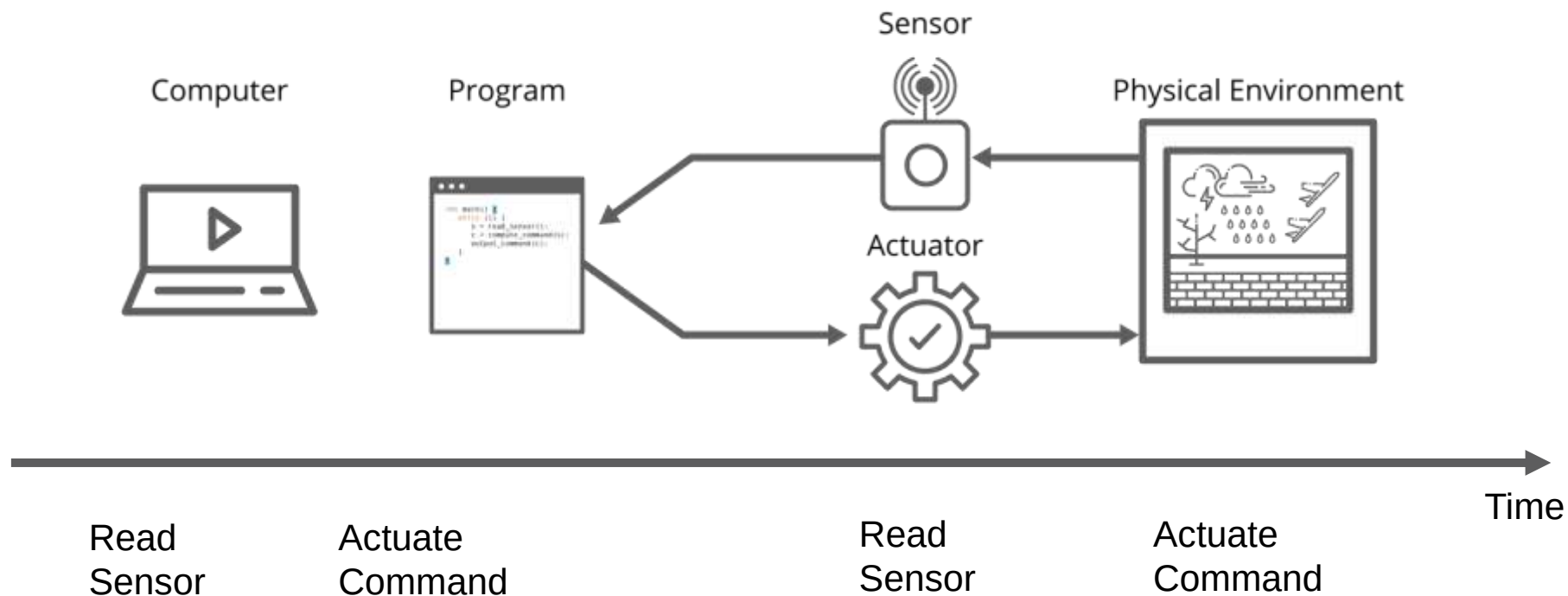
Commonality of these Systems



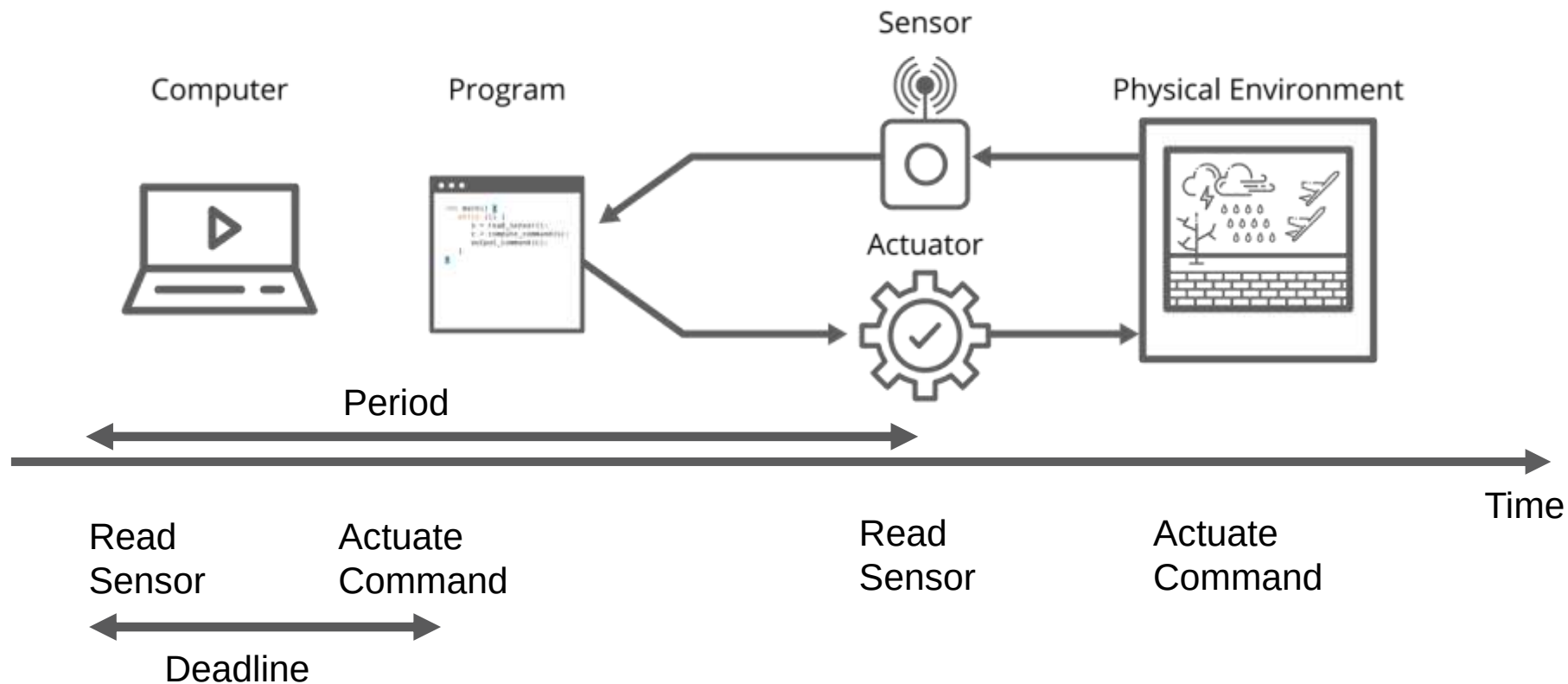
Commonality of these Systems



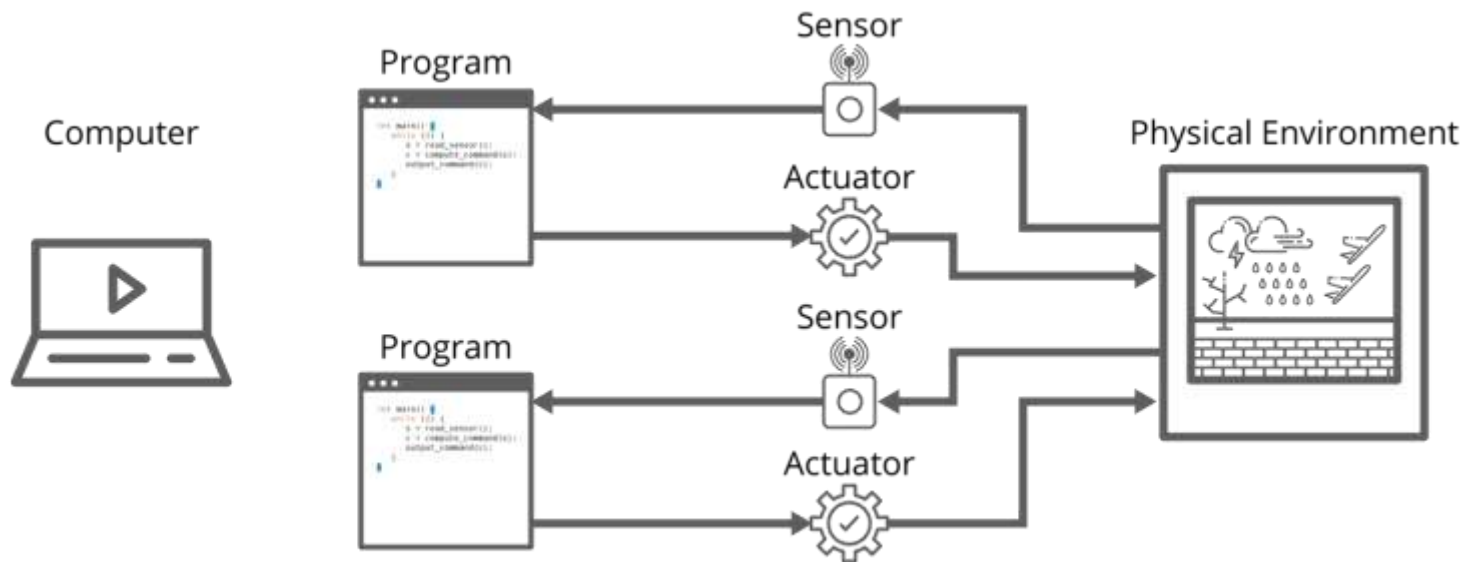
Commonality of these Systems



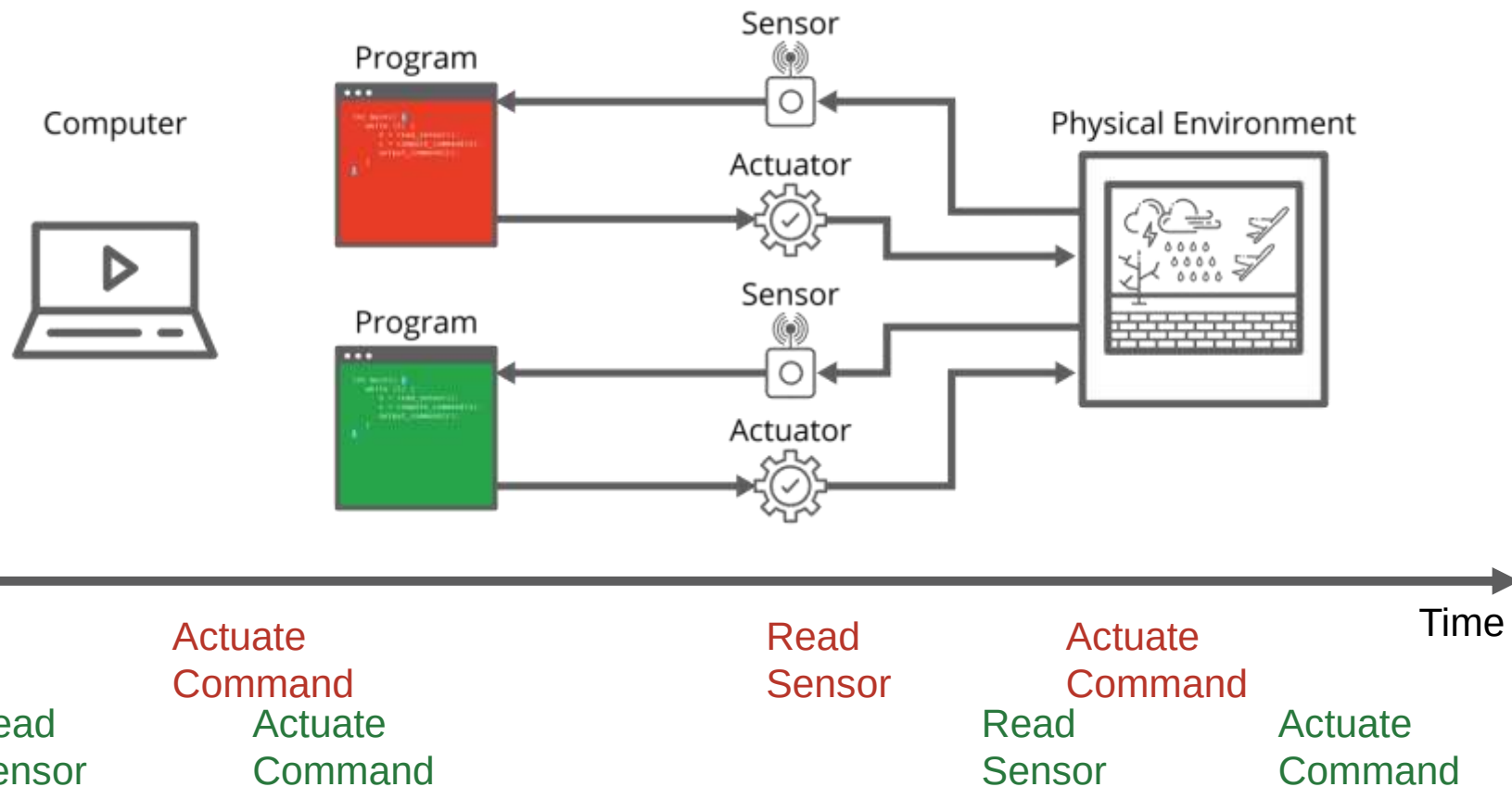
Commonality of these Systems



Commonality of these Systems



Commonality of these Systems



What Makes it Challenging to Satisfy Real-Time Requirements?

What Causes Delay of Software?

What Causes Delay of Software?



Time

What Causes Delay of Software?



What Causes Delay of Software?

Thread executes
one path



Time when one thread
in the software system arrives

Deadline Time

What Causes Delay of Software?

Thread executes another path

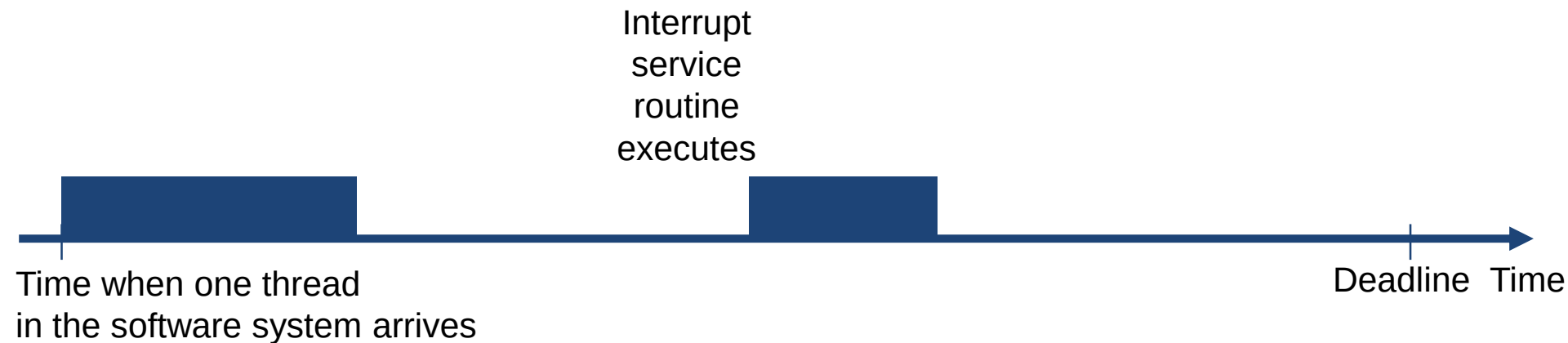


What Causes Delay of Software?

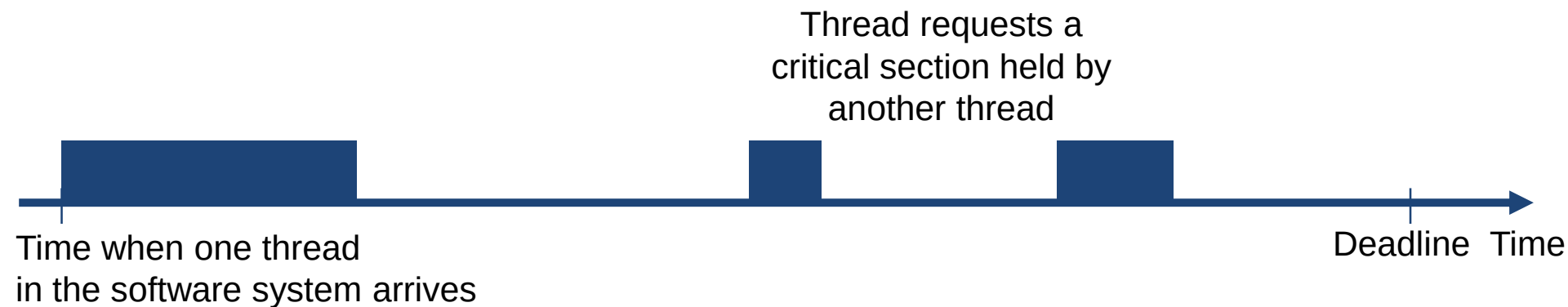
Preemption:
Another thread uses
the processor.



What Causes Delay of Software?



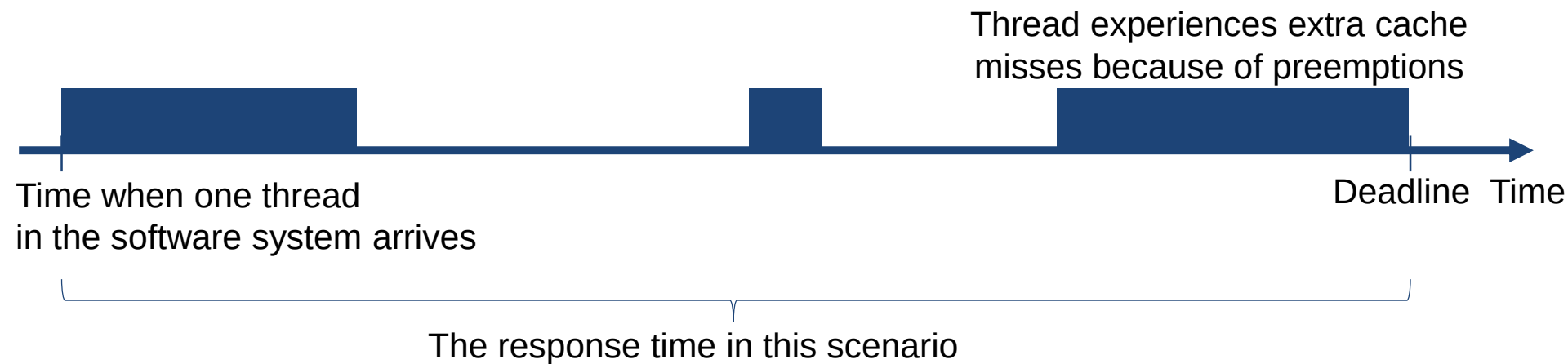
What Causes Delay of Software?



What Causes Delay of Software?

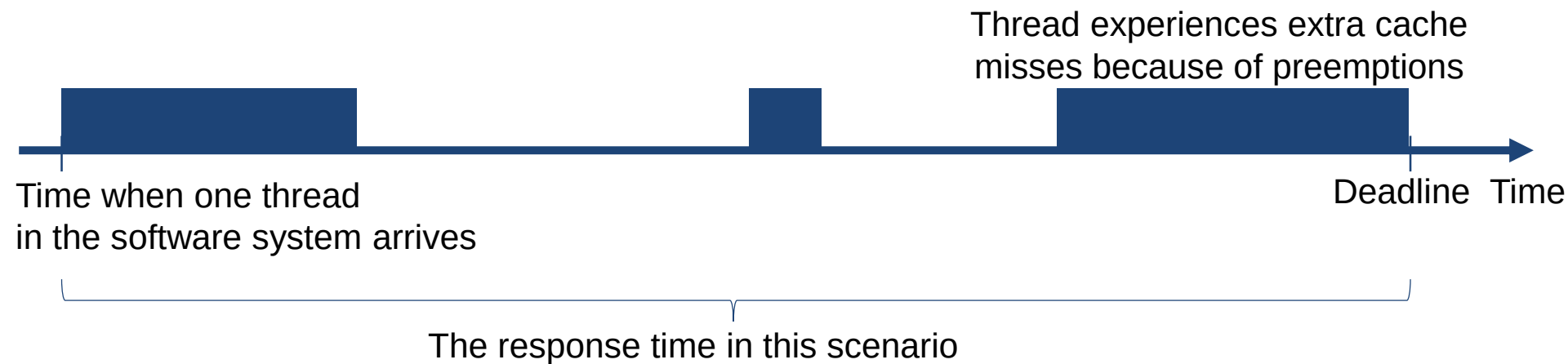


What Causes Delay of Software?



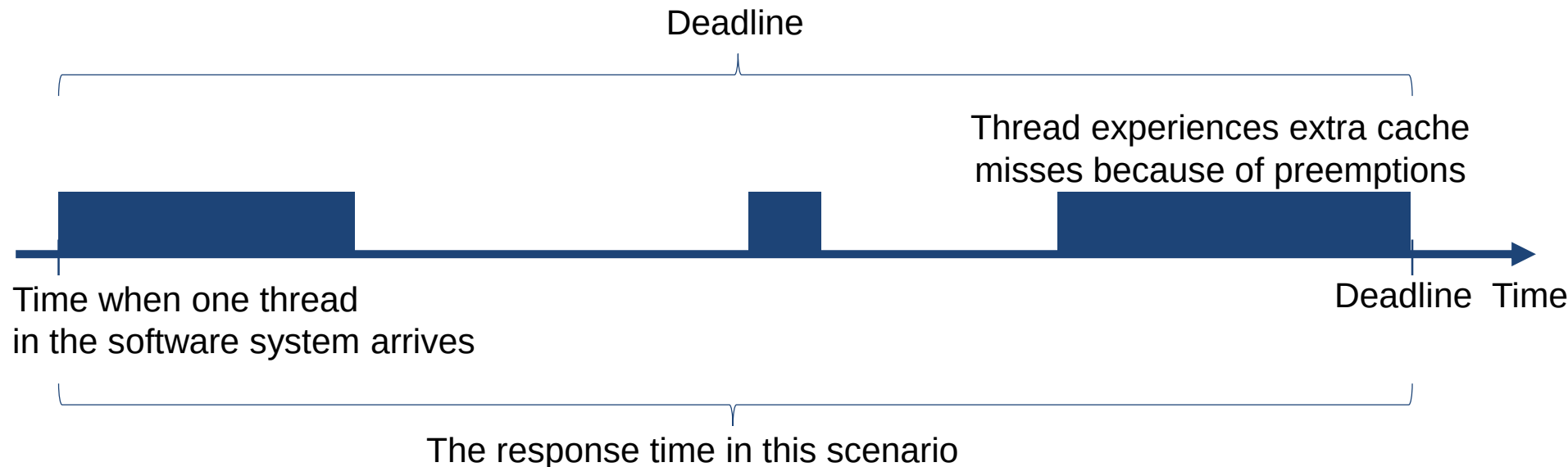
What Causes Delay of Software?

How large can the response time be?



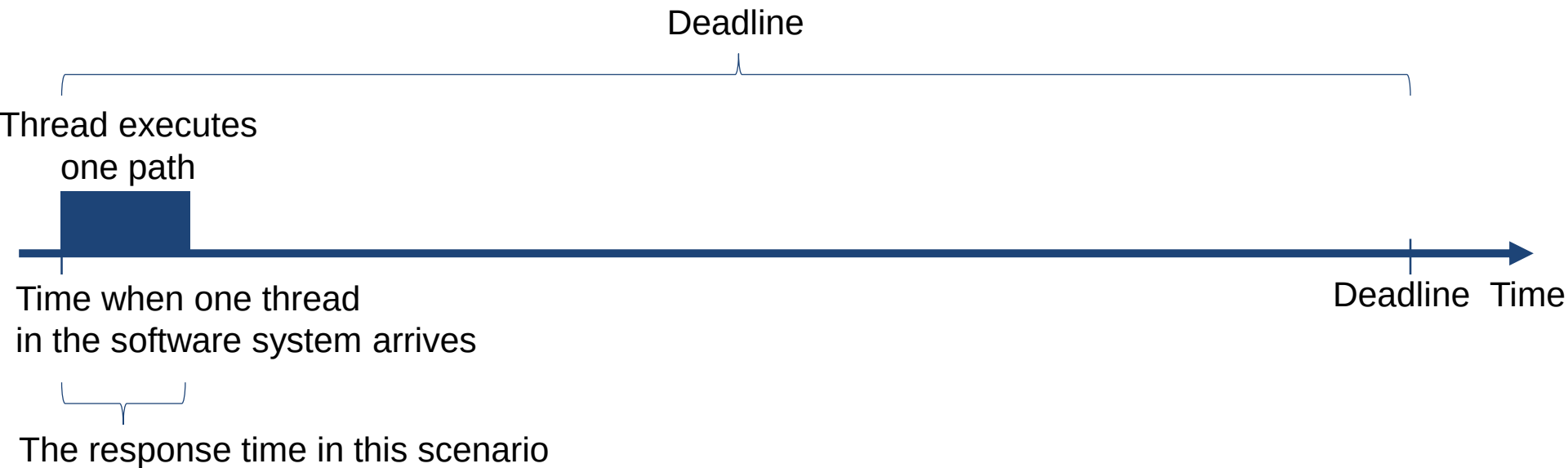
What Causes Delay of Software?

Does it hold for all scenarios that the response time is at most the deadline?



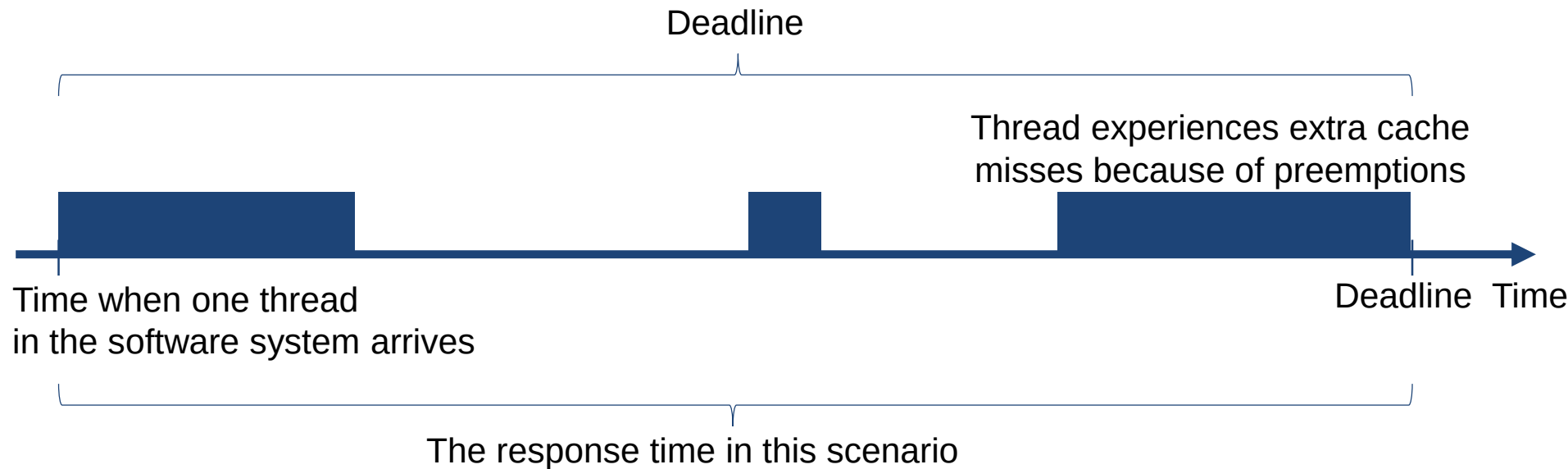
What Causes Delay of Software?

Does it hold for all scenarios that the response time is at most the deadline?



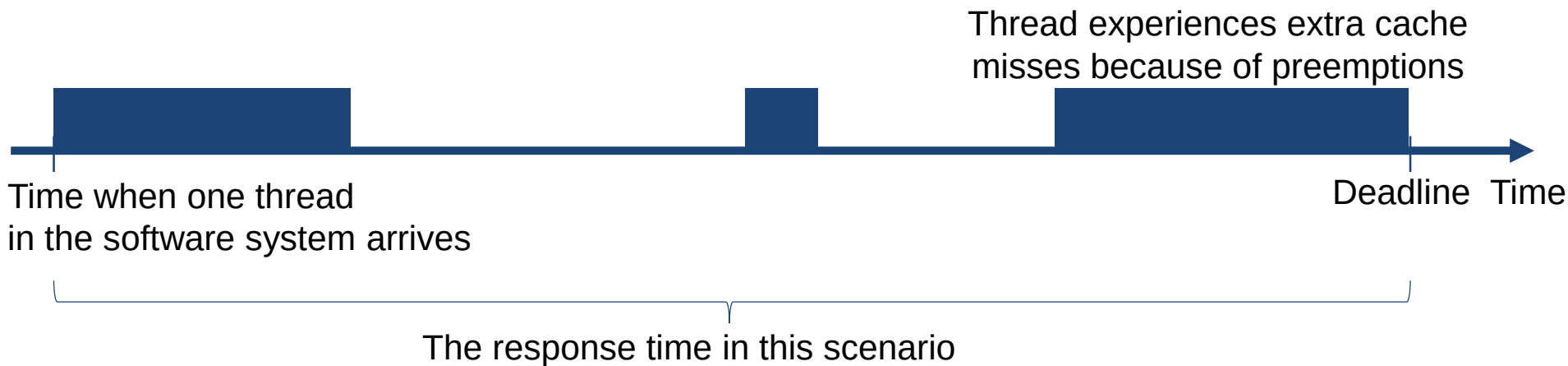
What Causes Delay of Software?

Does it hold for all scenarios that the response time is at most the deadline?



What Causes Delay of Software?

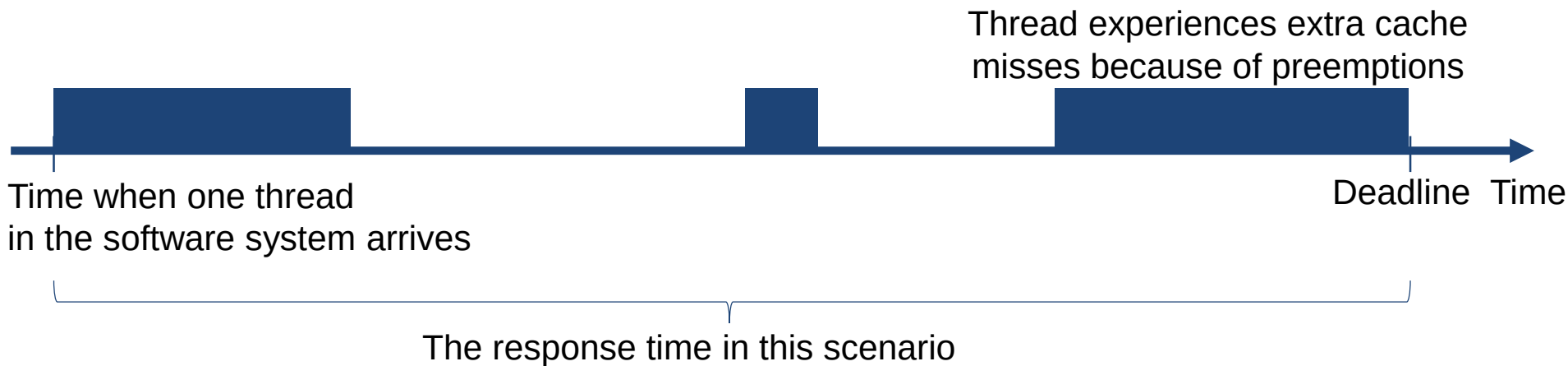
Does it hold for all threads,
that for all arrivals of the thread,
that for all scenarios,
that the finishing time is at most the deadline?



What Causes Delay of Software?

Does it hold for all threads,
that for all arrivals of the thread,
that for all scenarios,
that the finishing time is at most the deadline?

If “yes,” we say the set of threads is *schedulable*.
Otherwise, the set of threads is *unschedulable*.



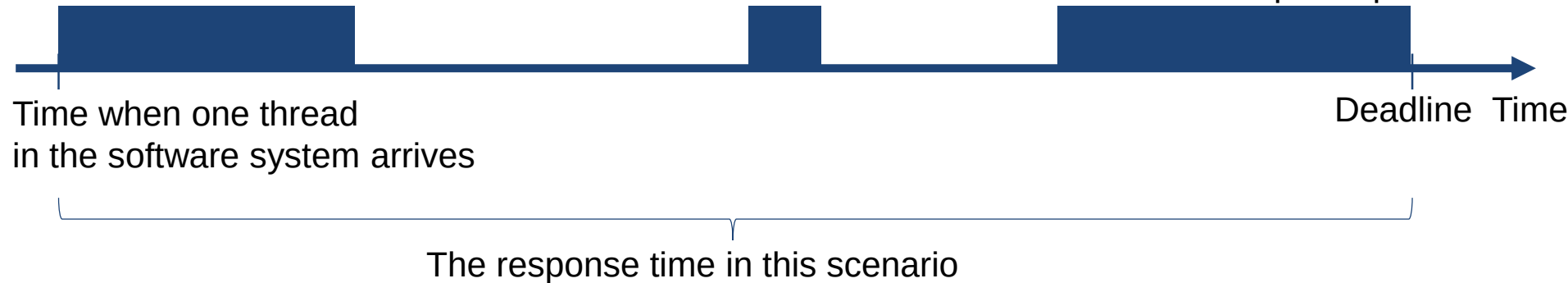
What Causes Delay of Software?

Does it hold for all threads,
that for all arrivals of the thread,
that for all scenarios,
that the finishing time is at most the deadline?

If “yes,” we say the set of threads is *schedulable*.
Otherwise, the set of threads is *unschedulable*.

The process of determining whether a set of threads is schedulable is called *schedulability analysis*.

Thread experiences extra cache misses because of preemptions



Conclusions so far

Many systems interact with the physical world

This interaction requires correct timing

Correct timing depends on whether the delay of the software is at most a certain bound

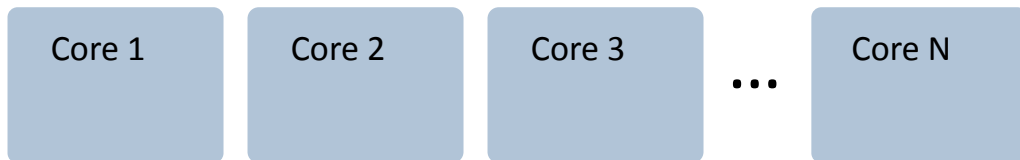
There are many causes of the delay of software (even on a computer with a single core)

Many systems today disable all processor cores except one in order to be confident about timing

Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

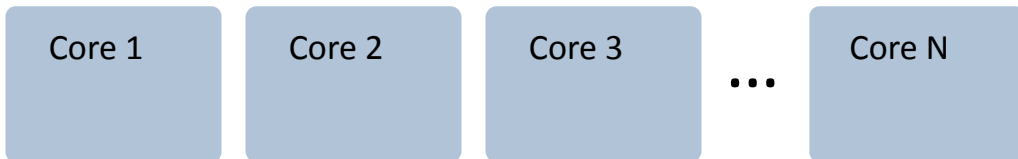
- *All computers are multicores.*



Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

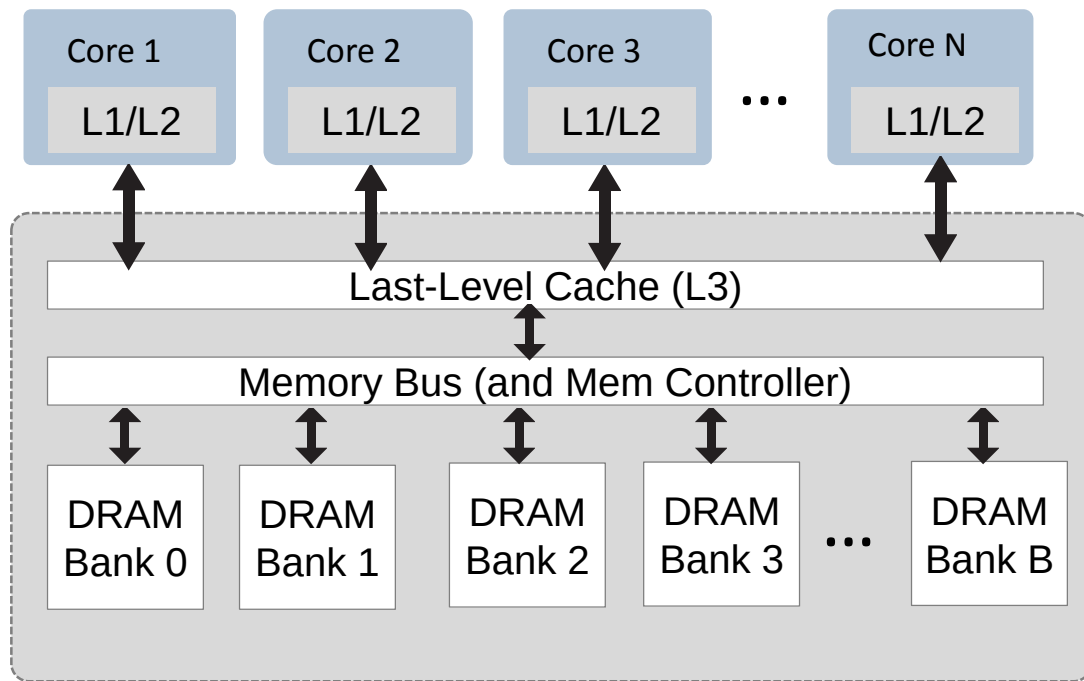
- *All computers are multicores.*
- *Most chip makers do not offer single core.*



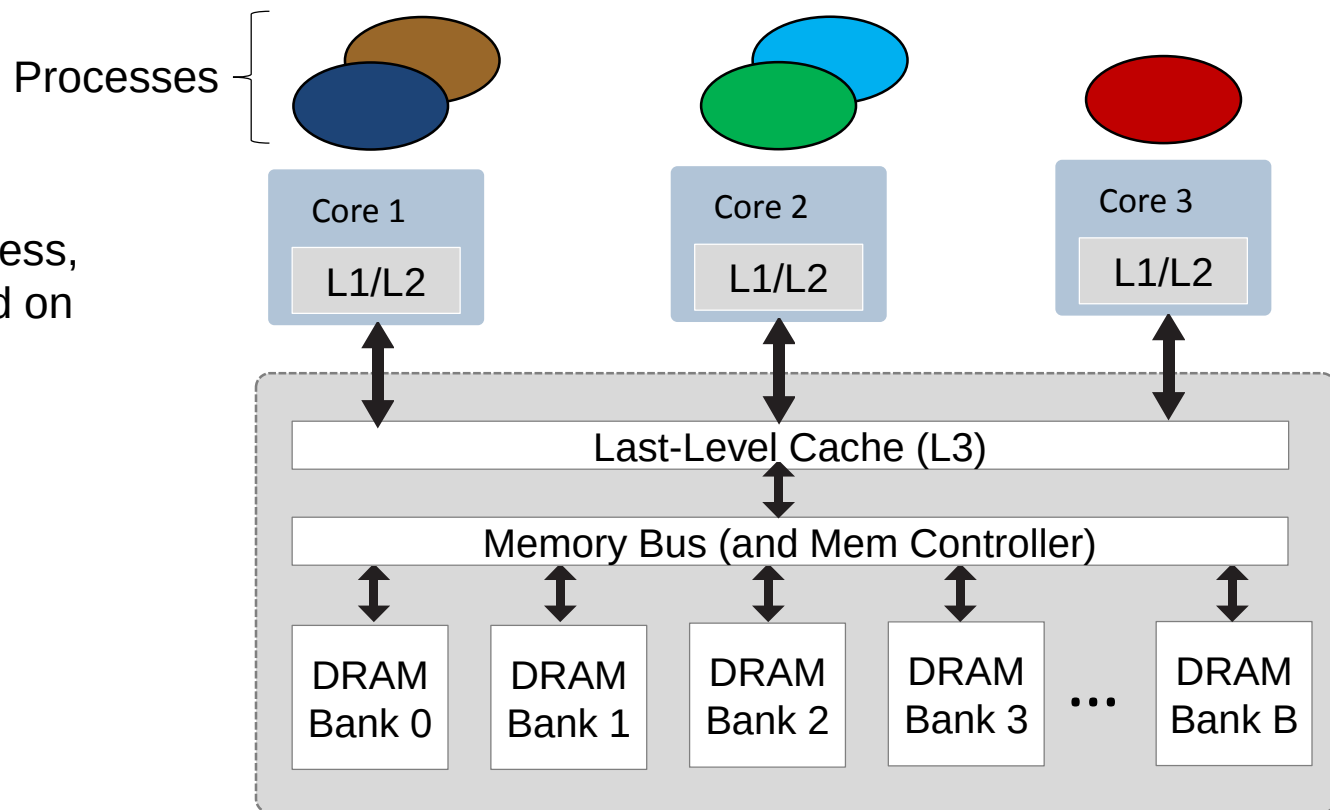
Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

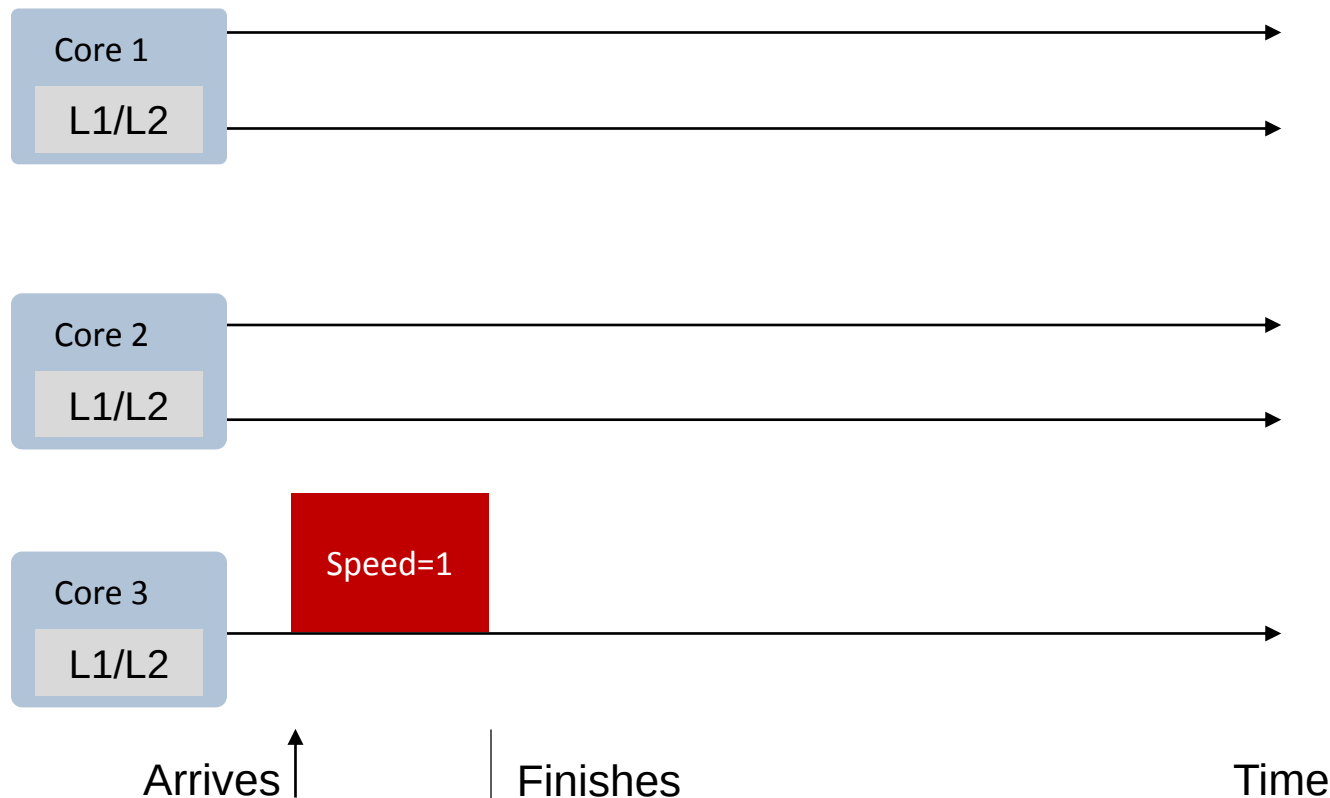
- *All computers are multicores.*
- *Most chip makers do not offer single core.*
- *Most multicores have shared memory.*



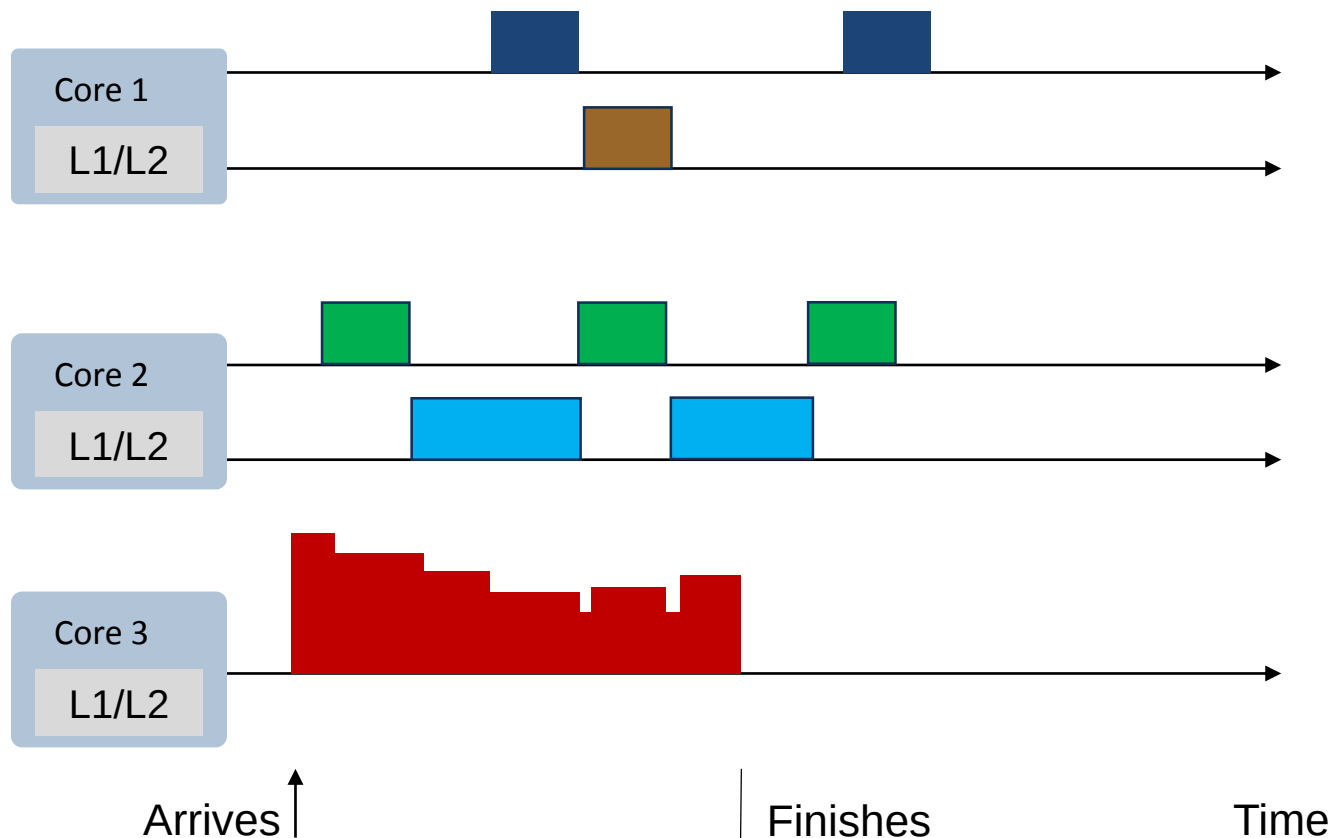
Problem: For each process, compute an upper bound on its response time.



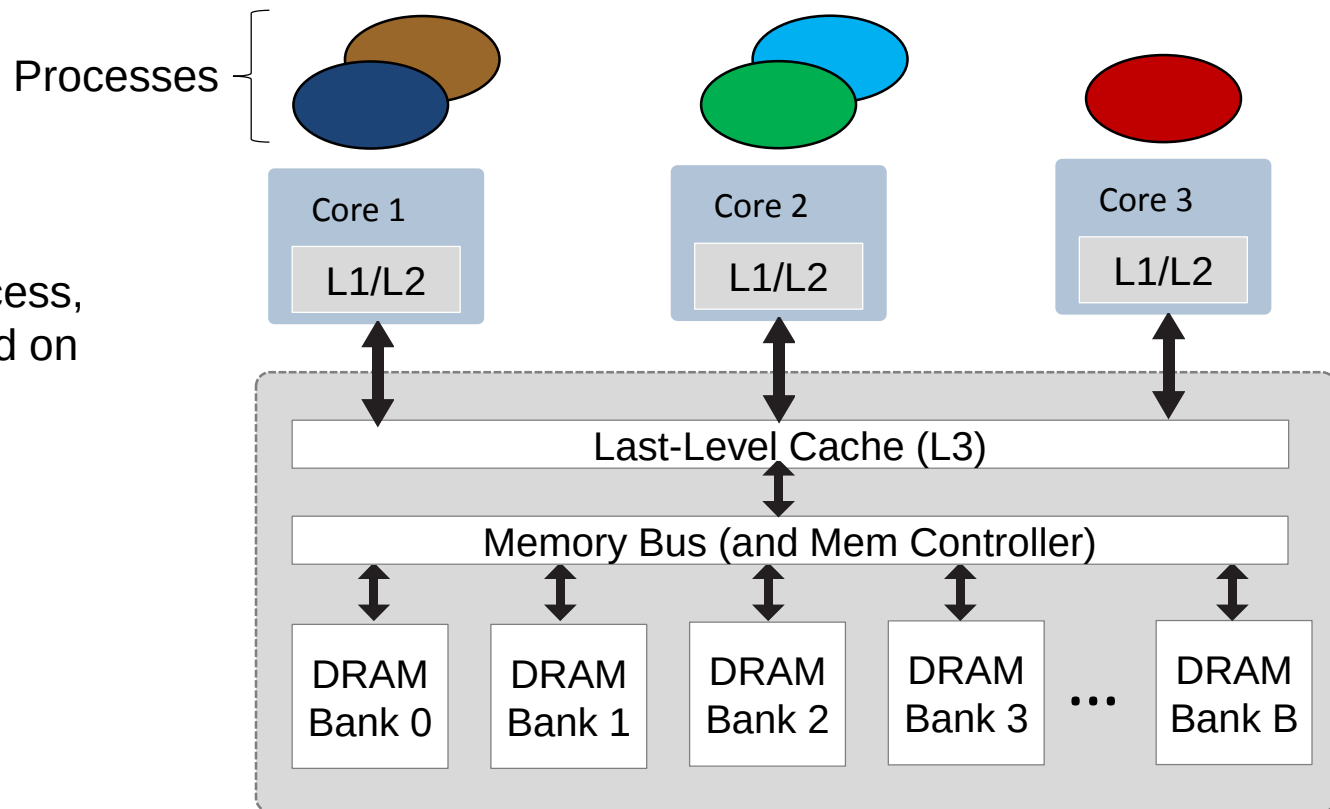
How Co-Runners Impact Speed of Execution



How Co-Runners Impact Speed of Execution



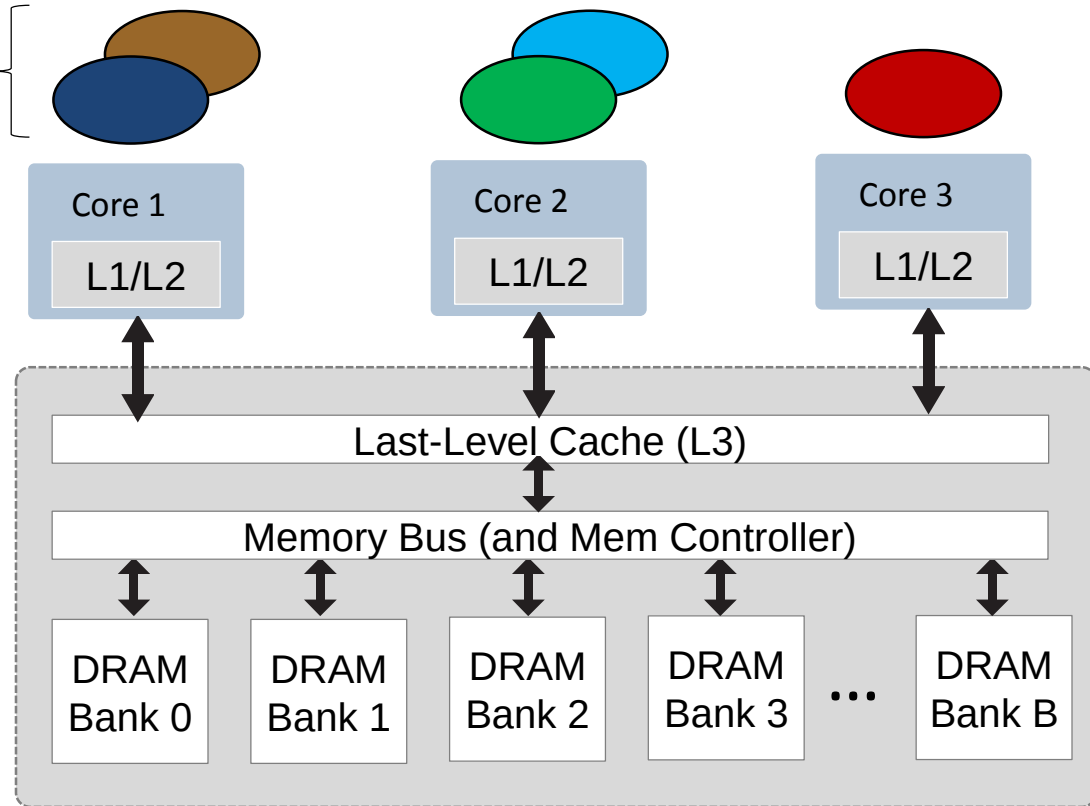
Problem: For each process, compute an upper bound on its response time.



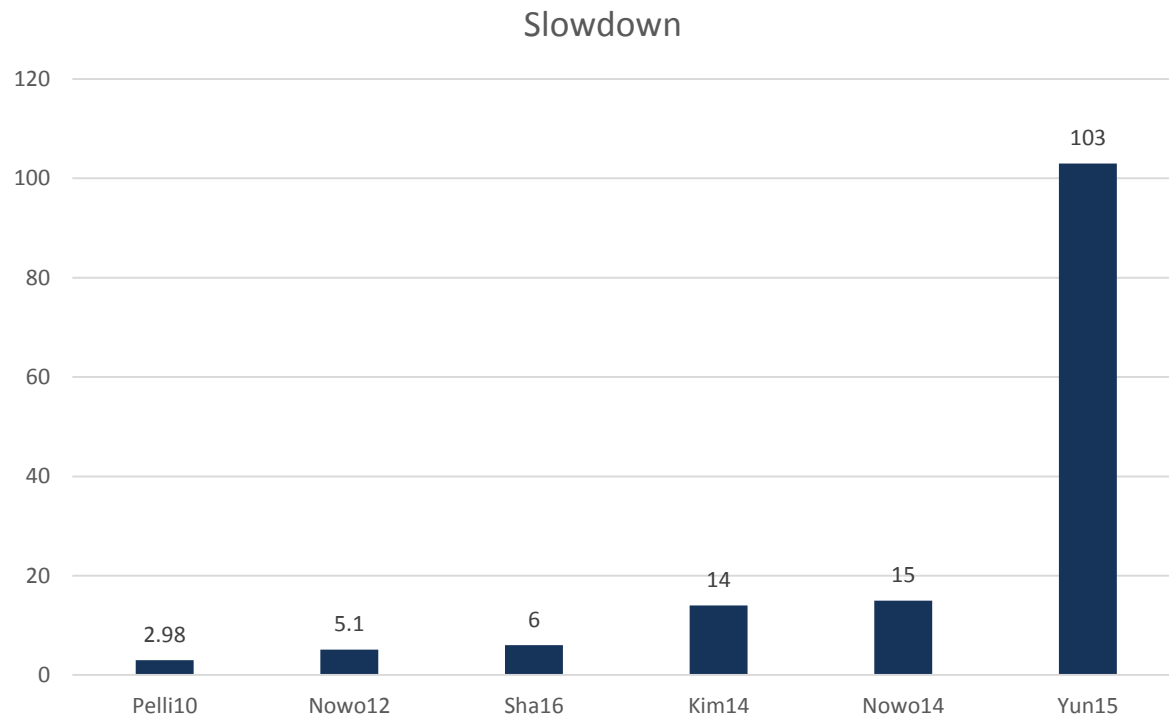
Issues

- *Shared hardware resources impact timing.*

Processes



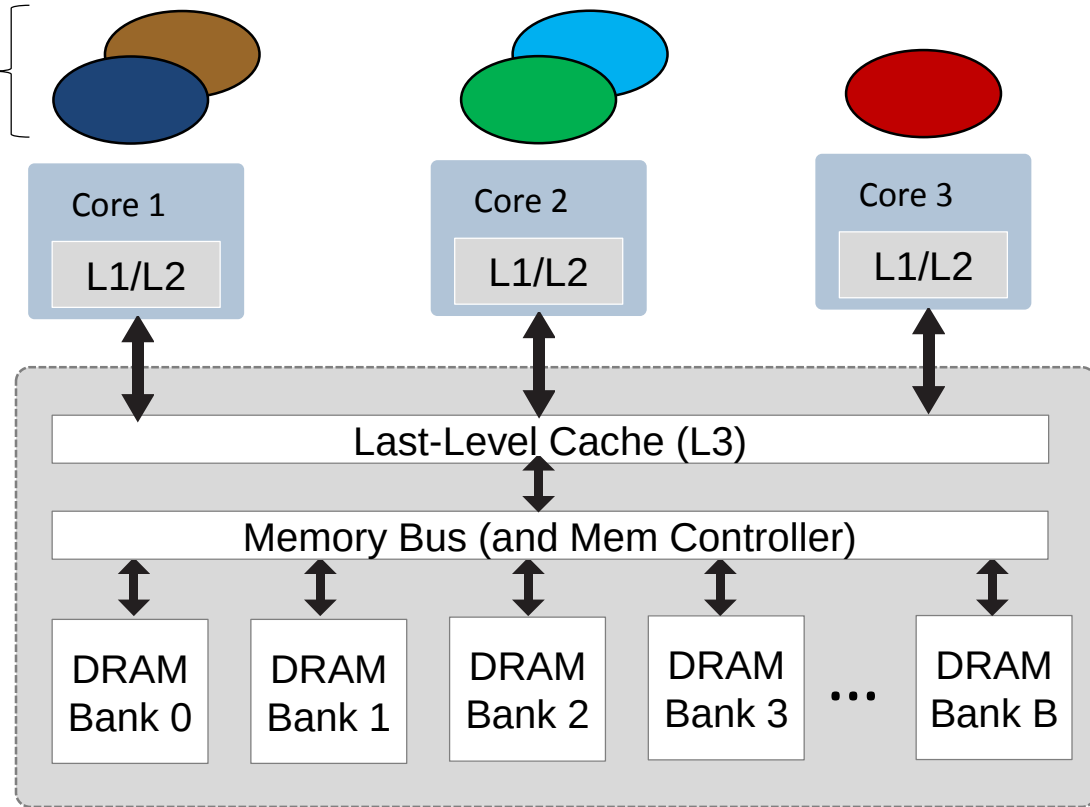
How Bad?



Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes

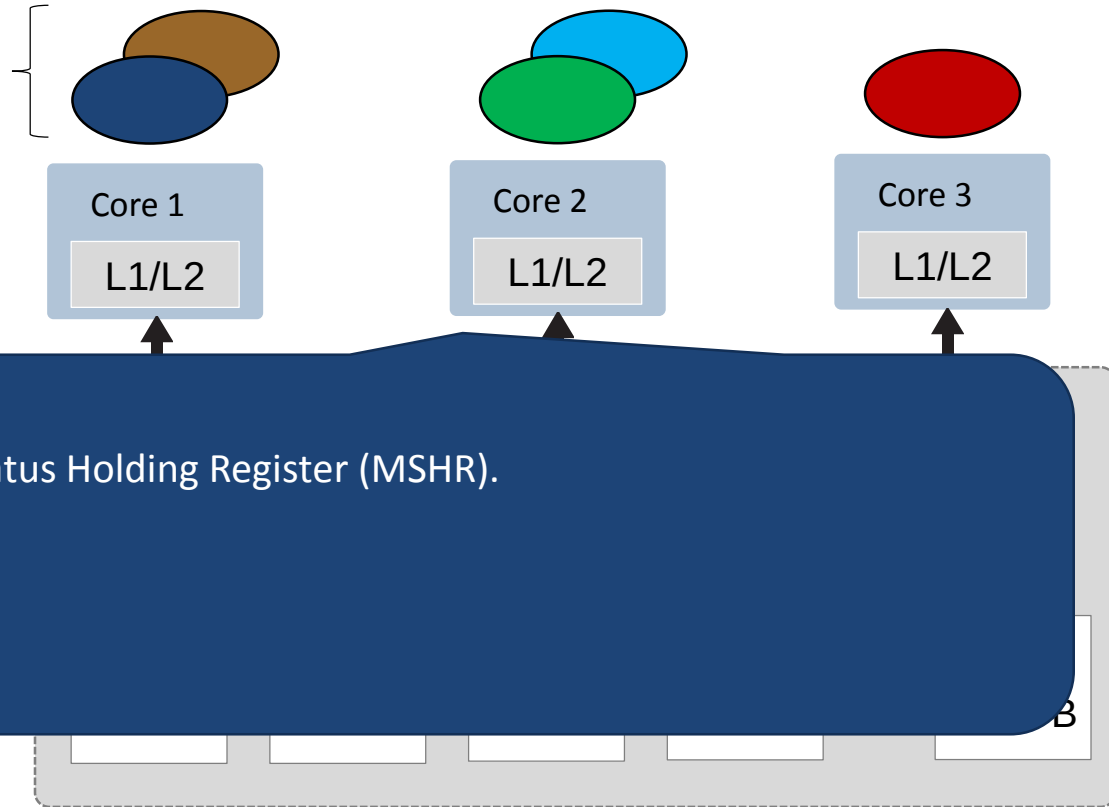


*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes

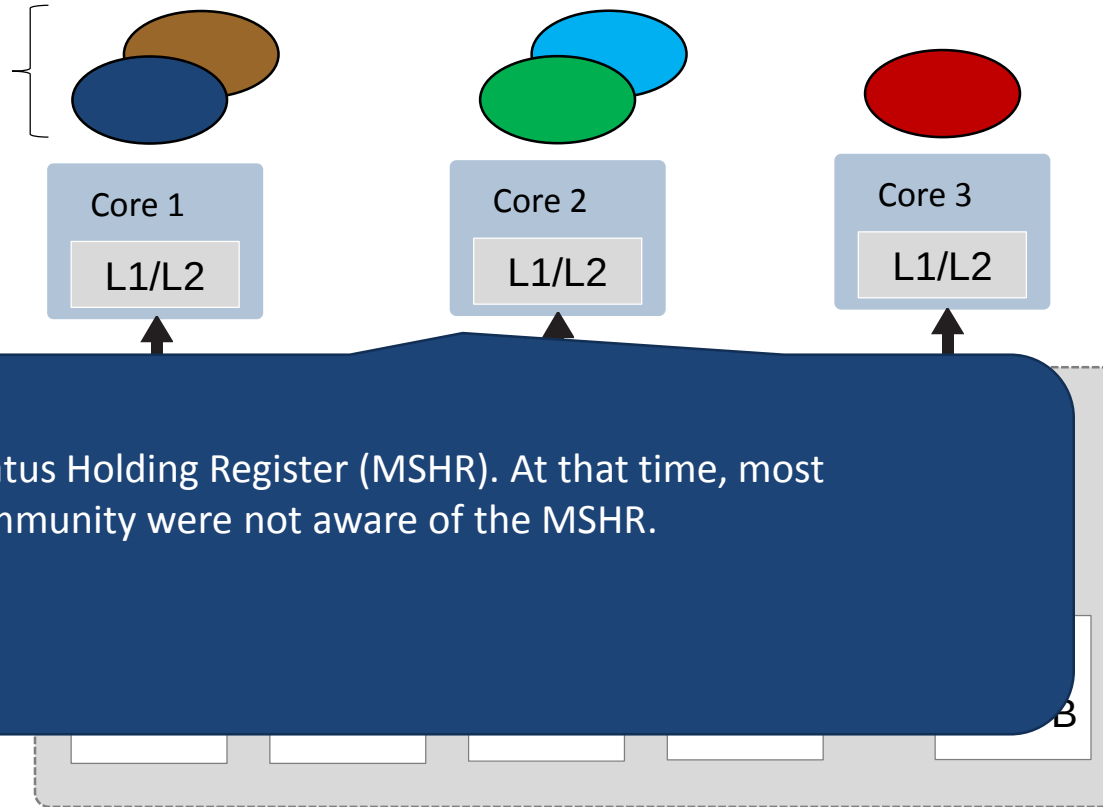


*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes



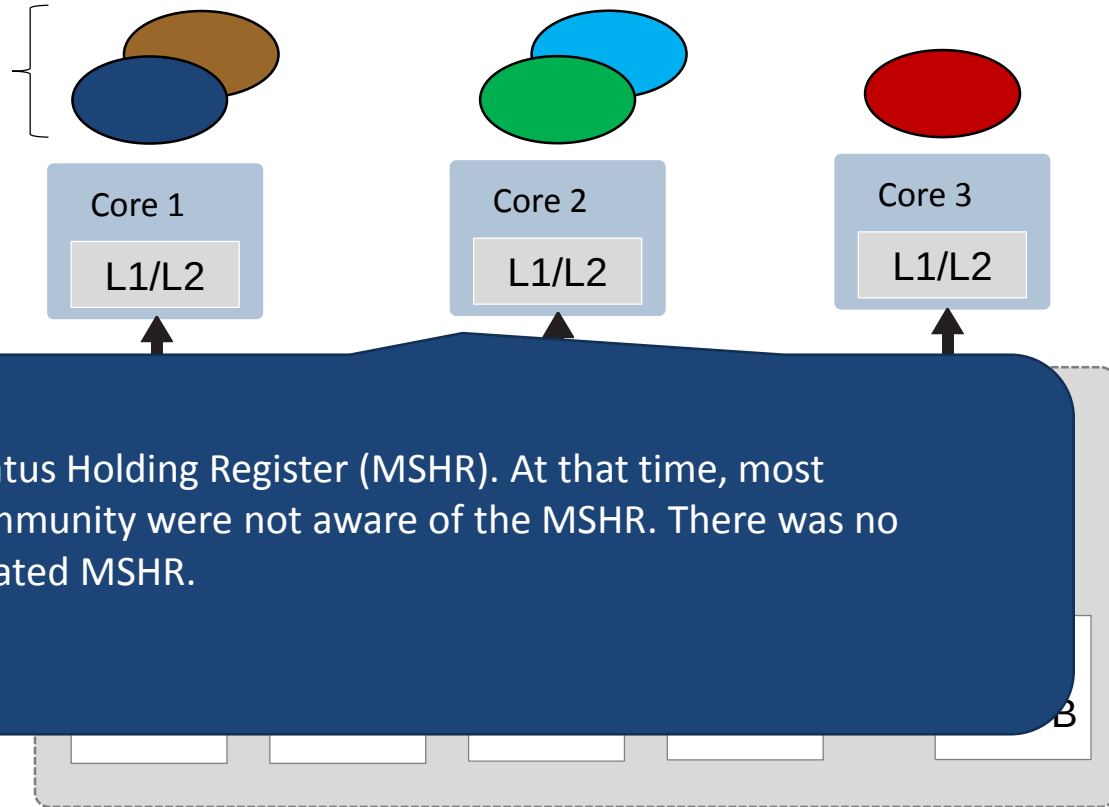
This slowdown is caused by Miss-Status Holding Register (MSHR). At that time, most researchers in real-time systems community were not aware of the MSHR.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes



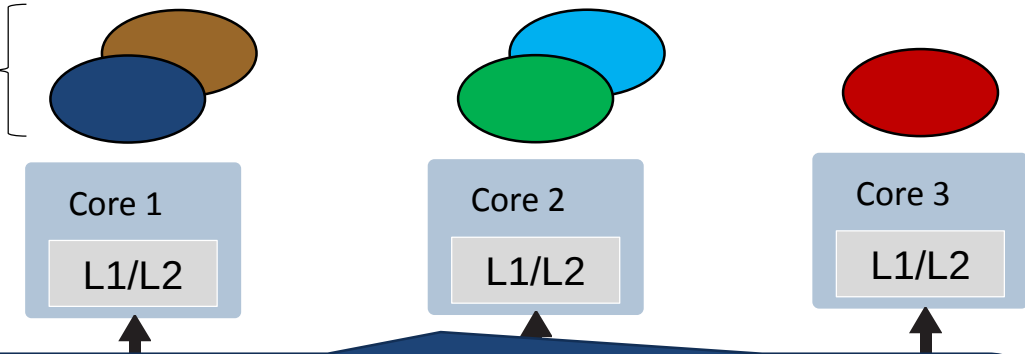
This slowdown is caused by Miss-Status Holding Register (MSHR). At that time, most researchers in real-time systems community were not aware of the MSHR. There was no schedulability analysis that incorporated MSHR.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes



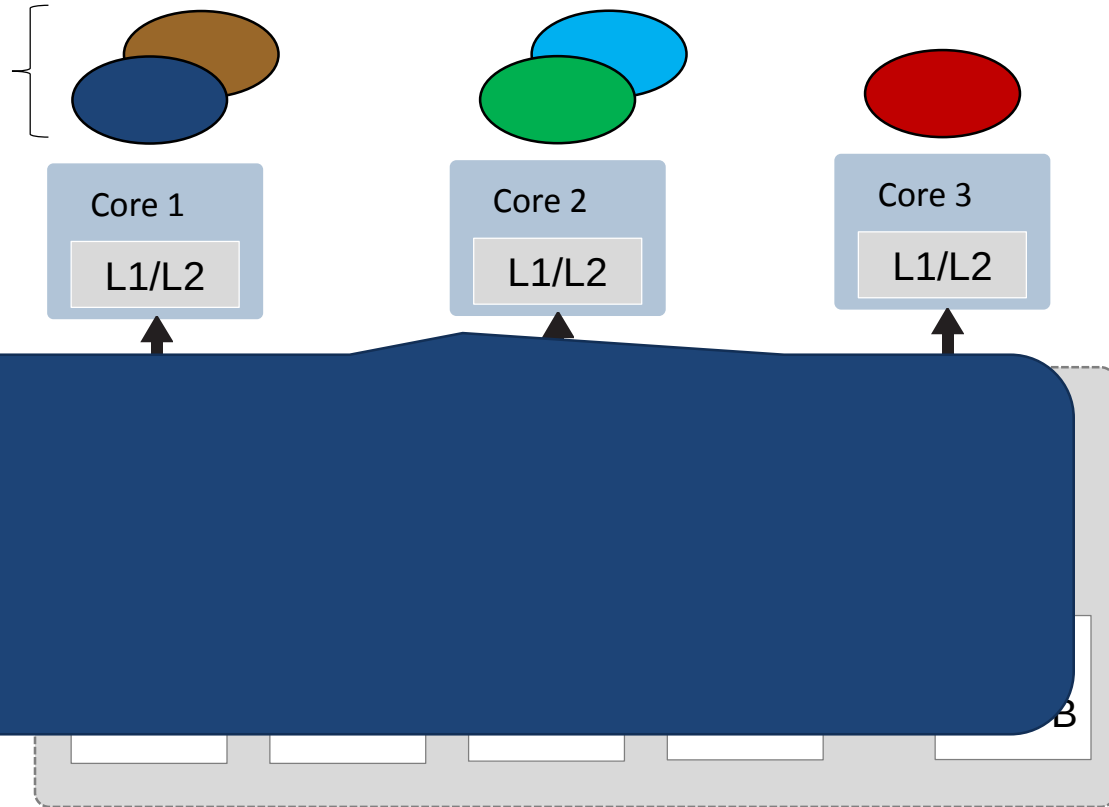
This slowdown is caused by Miss-Status Holding Register (MSHR). At that time, most researchers in real-time systems community were not aware of the MSHR. There was no schedulability analysis that incorporated MSHR. Even today, there is no schedulability analysis that incorporates the timing effects of MSHR.

*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes

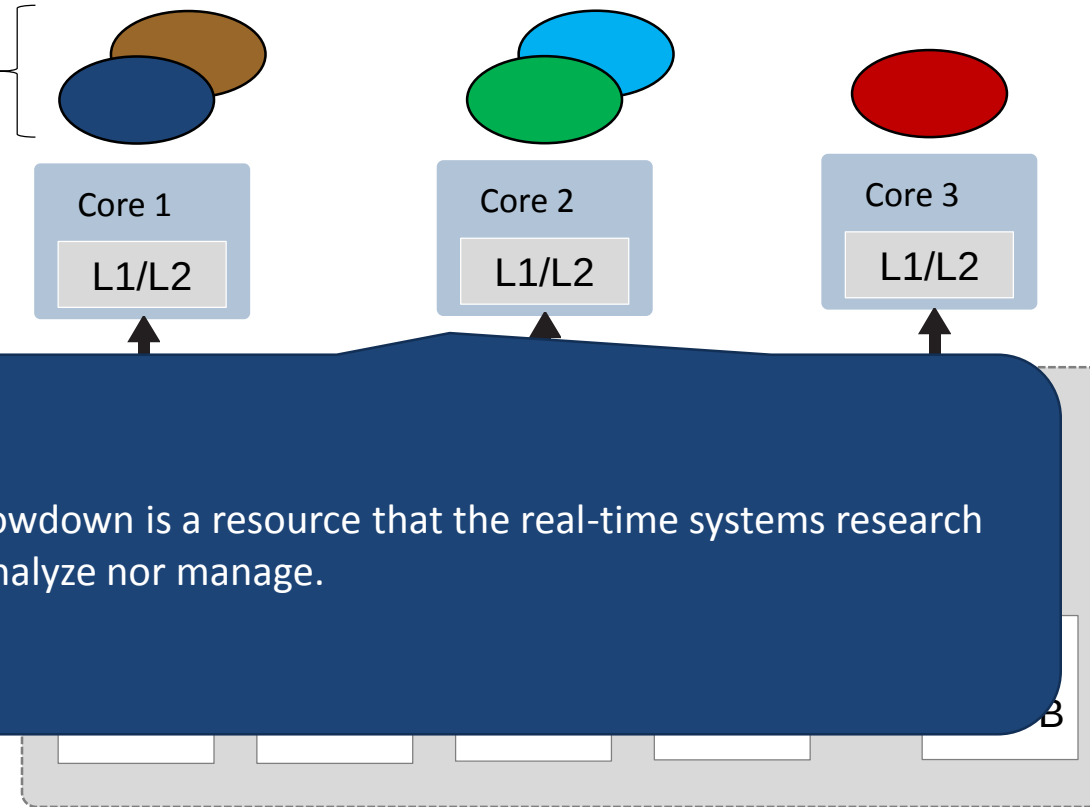


*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed.**

Processes

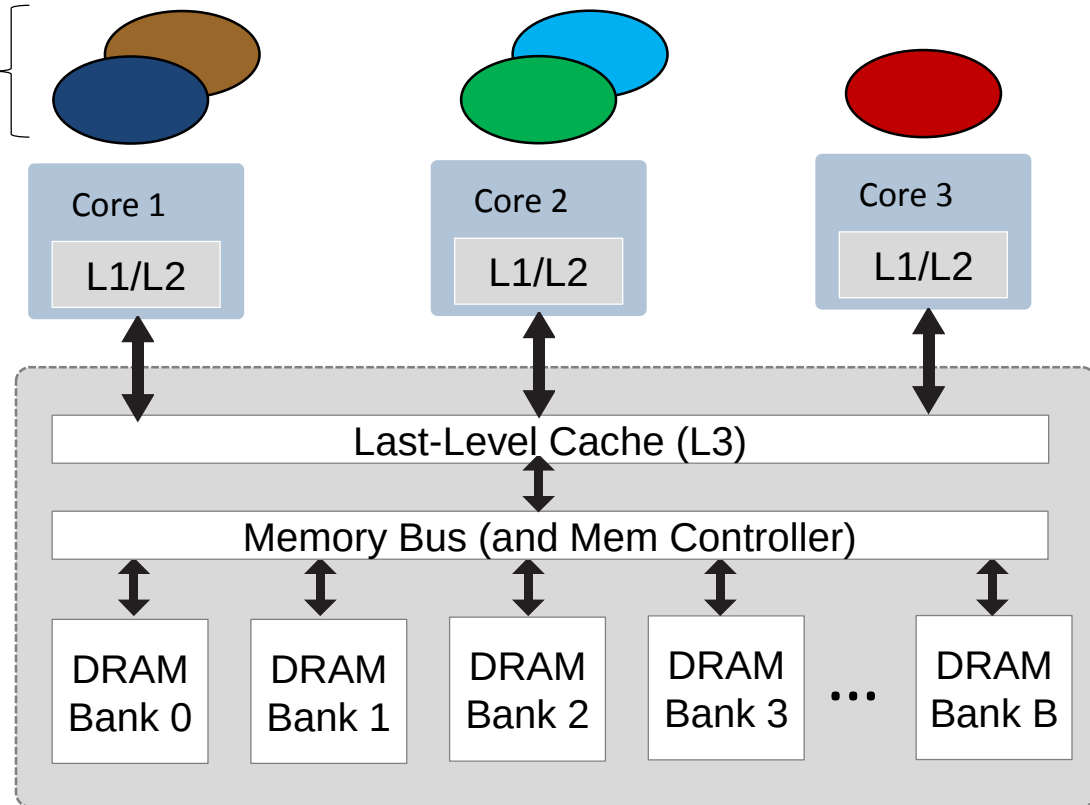


*H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed [Yun15].*
- *Current methods cannot deal with undocumented resources.*

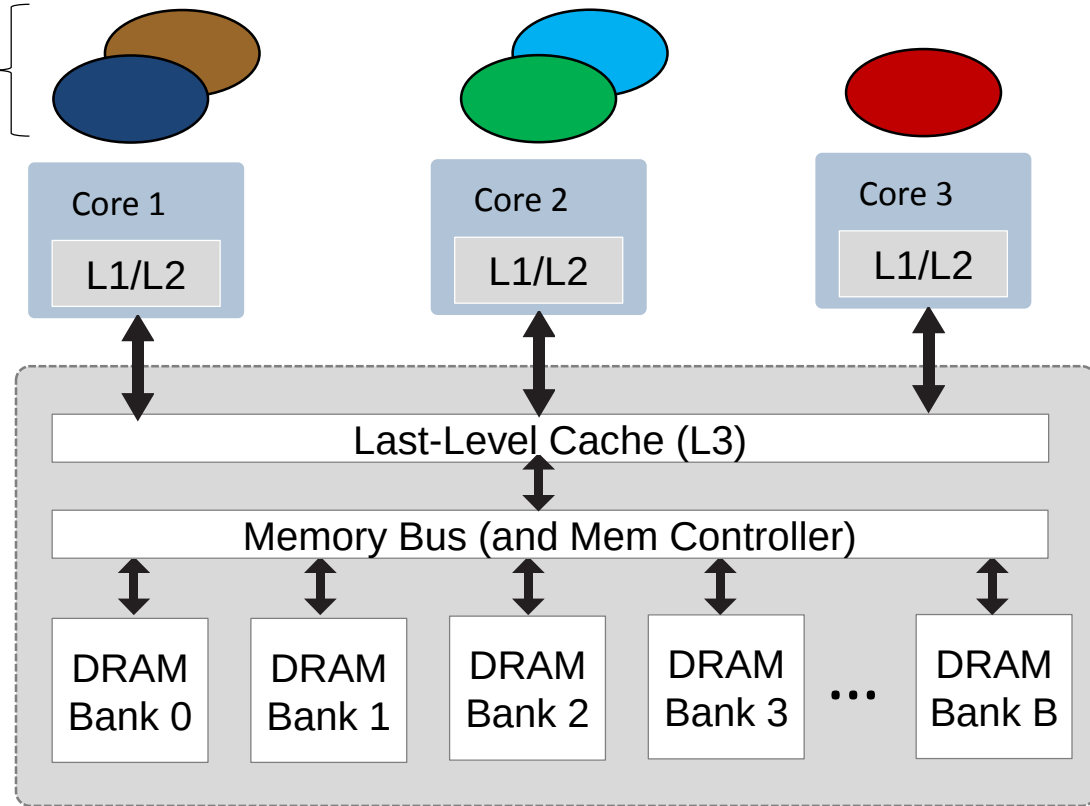
Processes



Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed [Yun15].*
- *Current methods cannot deal with undocumented resources.*
- *Even for the case that resources are documented, current methods can only analyze/manage a small set of them.*

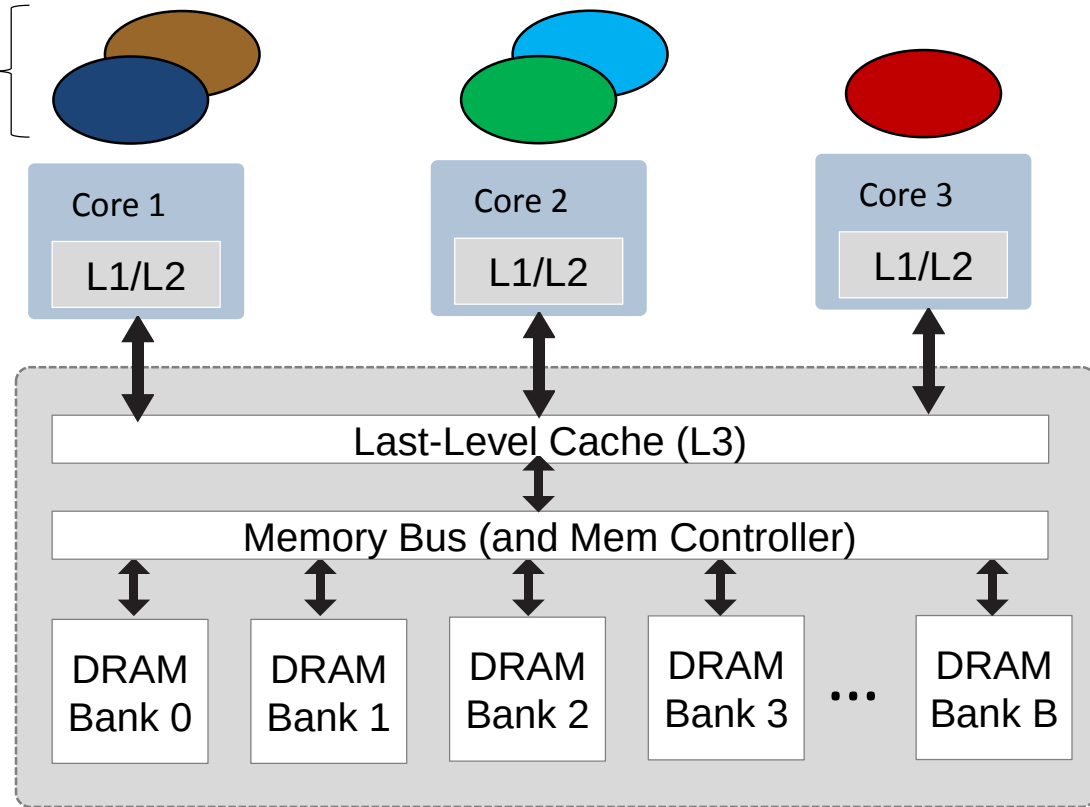
Processes



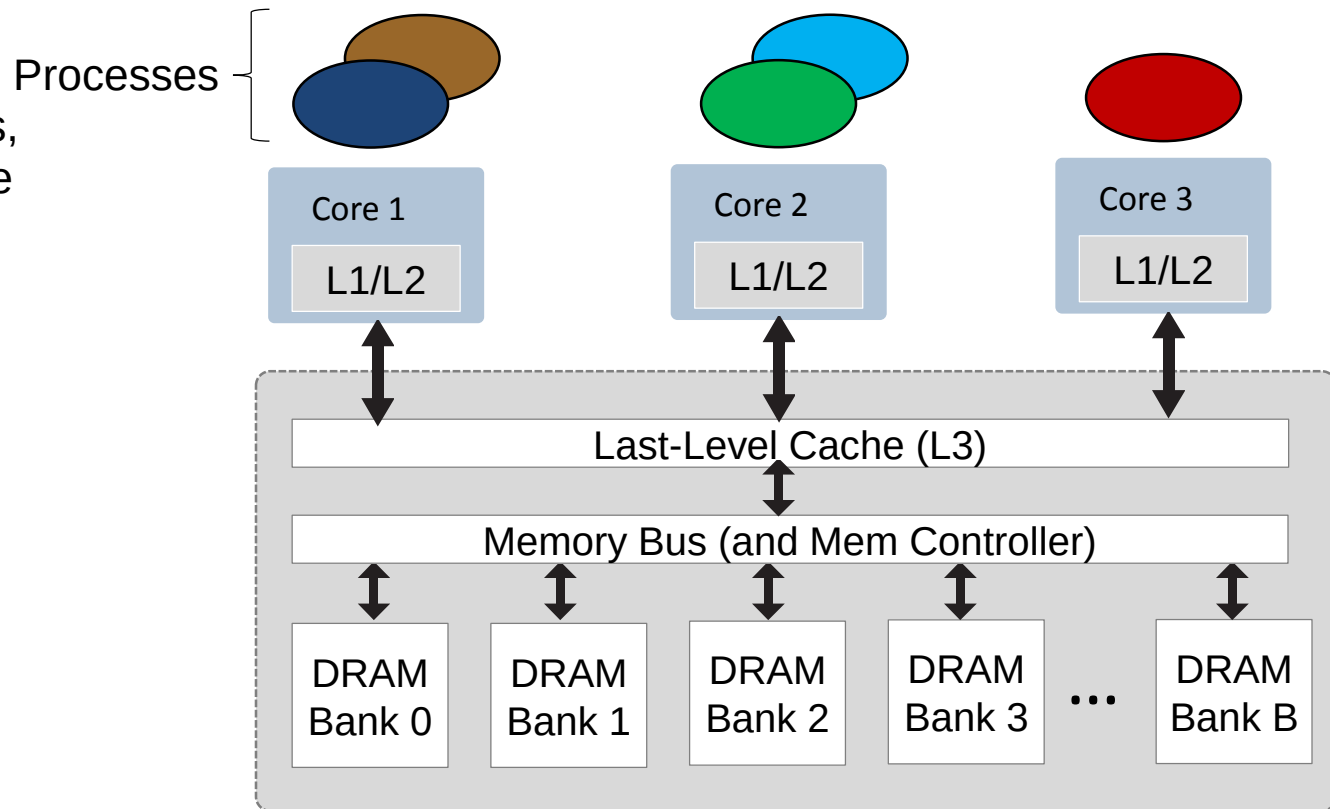
Issues

- *Shared hardware resources impact timing.*
- *103 times slowdown has been observed [Yun15].*
- *Current methods cannot deal with undocumented resources.*
- *Even when resources are documented, current methods can only analyze/manage a small set of them.*
- *The problem is getting worse:*
 - * *Slowdown increasing*
 - * *More undocumented h/w*

Processes

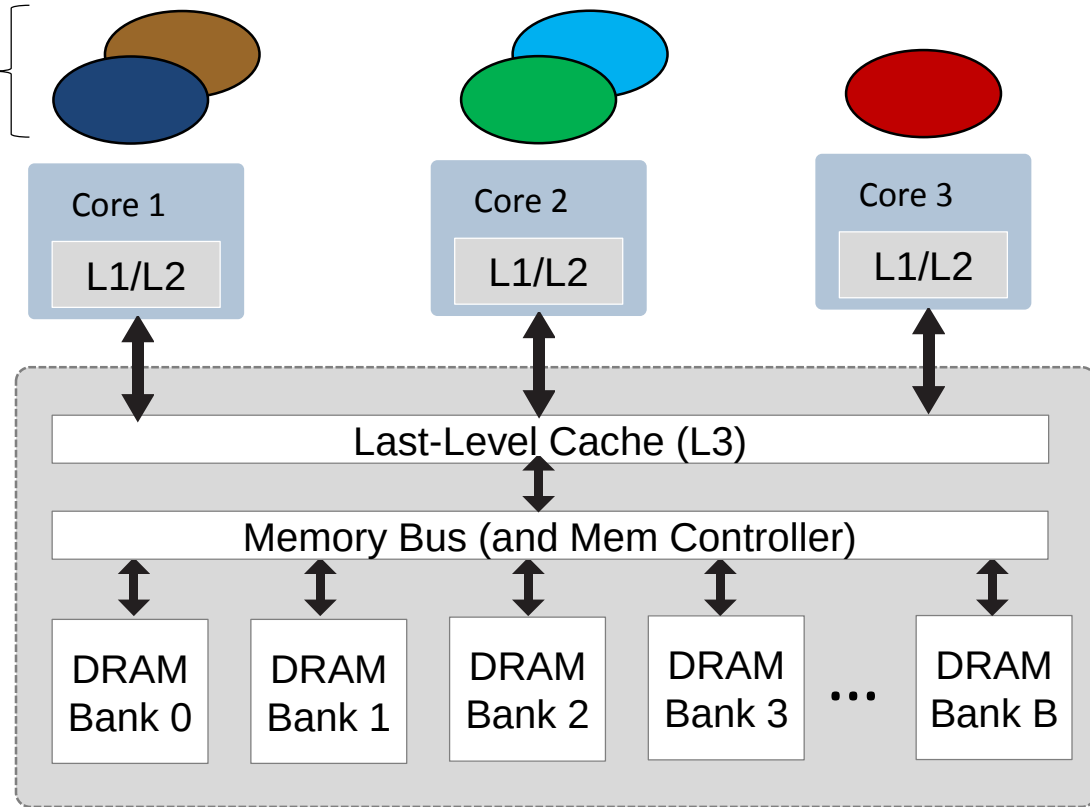


Problem: For each process,
compute its response time

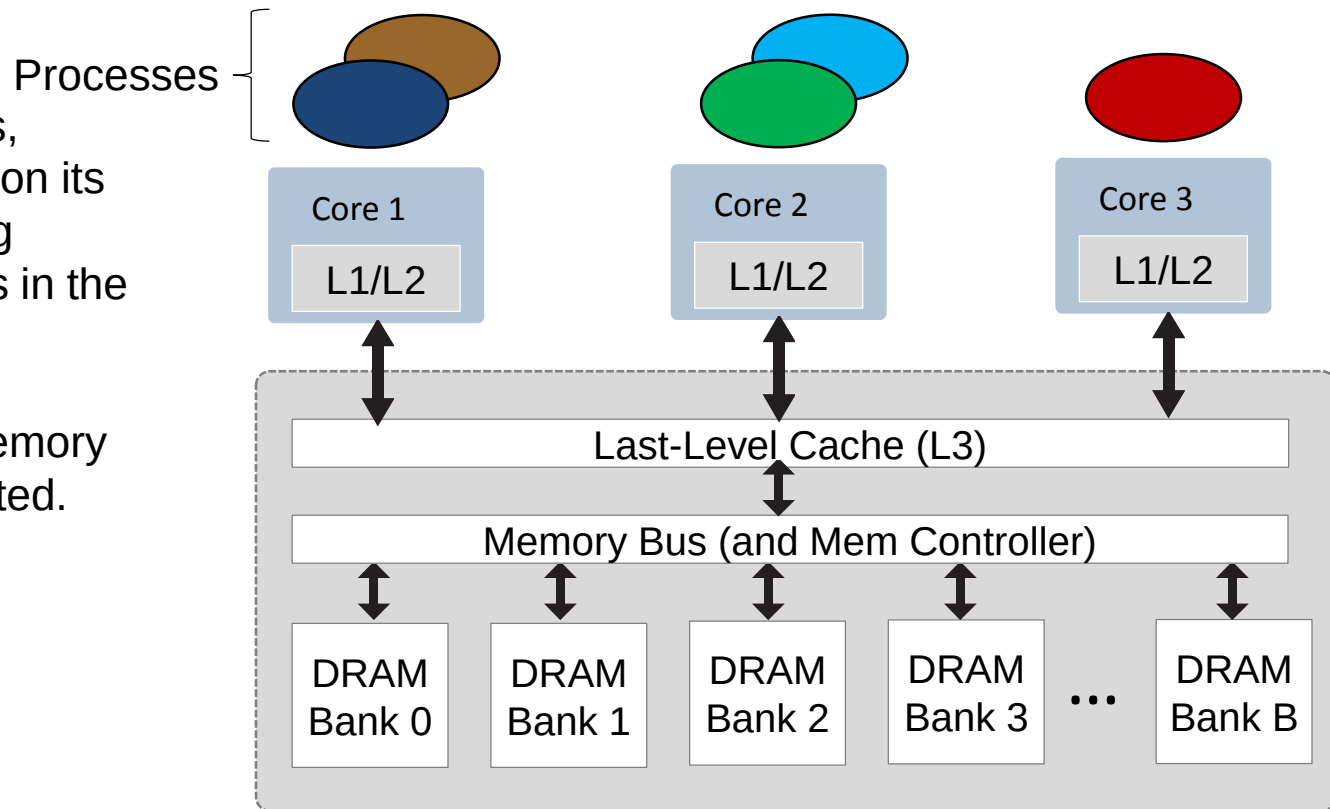


Processes

Problem: For each process, compute an upper bound on its response time

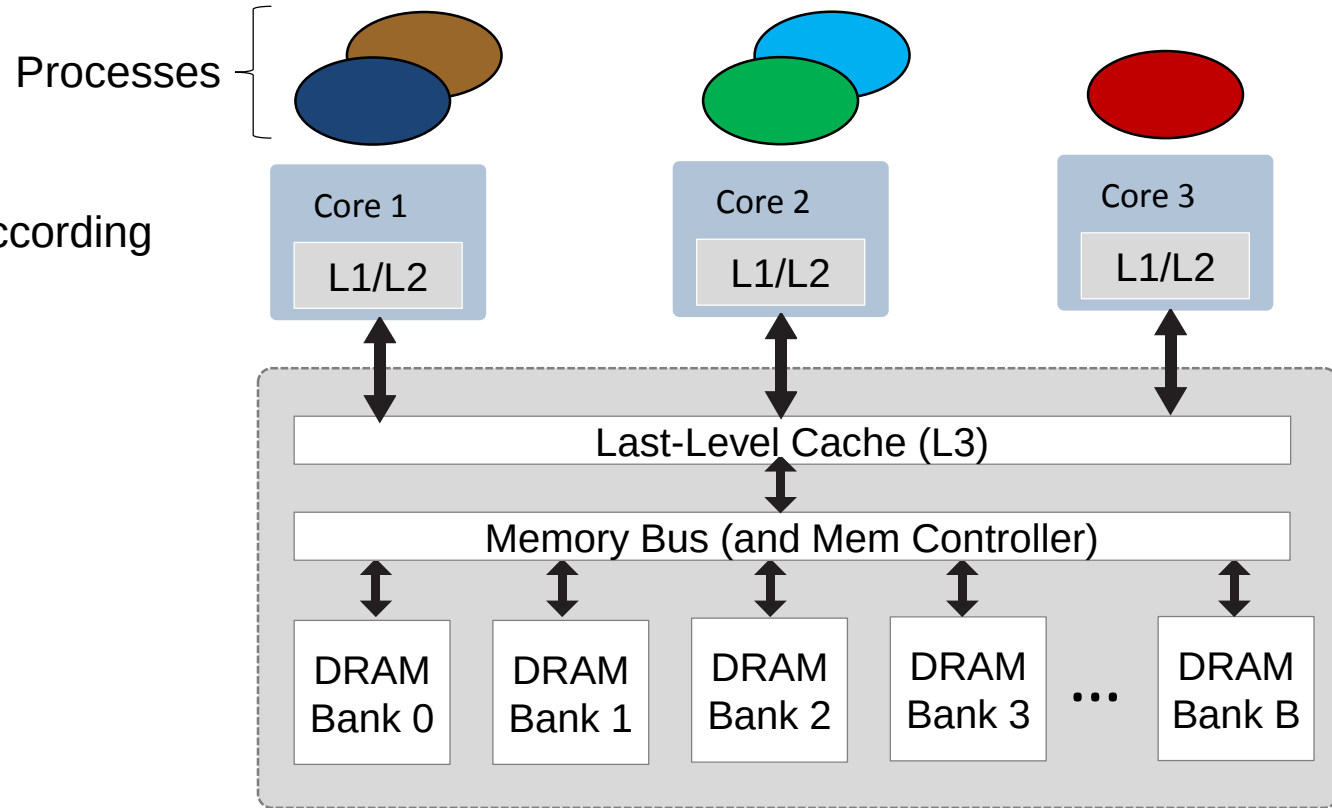


Problem: For each process,
compute an upper bound on its
response time considering
contention for resources in the
memory system
and
that resources in the memory
system are undocumented.



Documented resources

- DRAM memory timing tends to be specified according to JDEC standard.

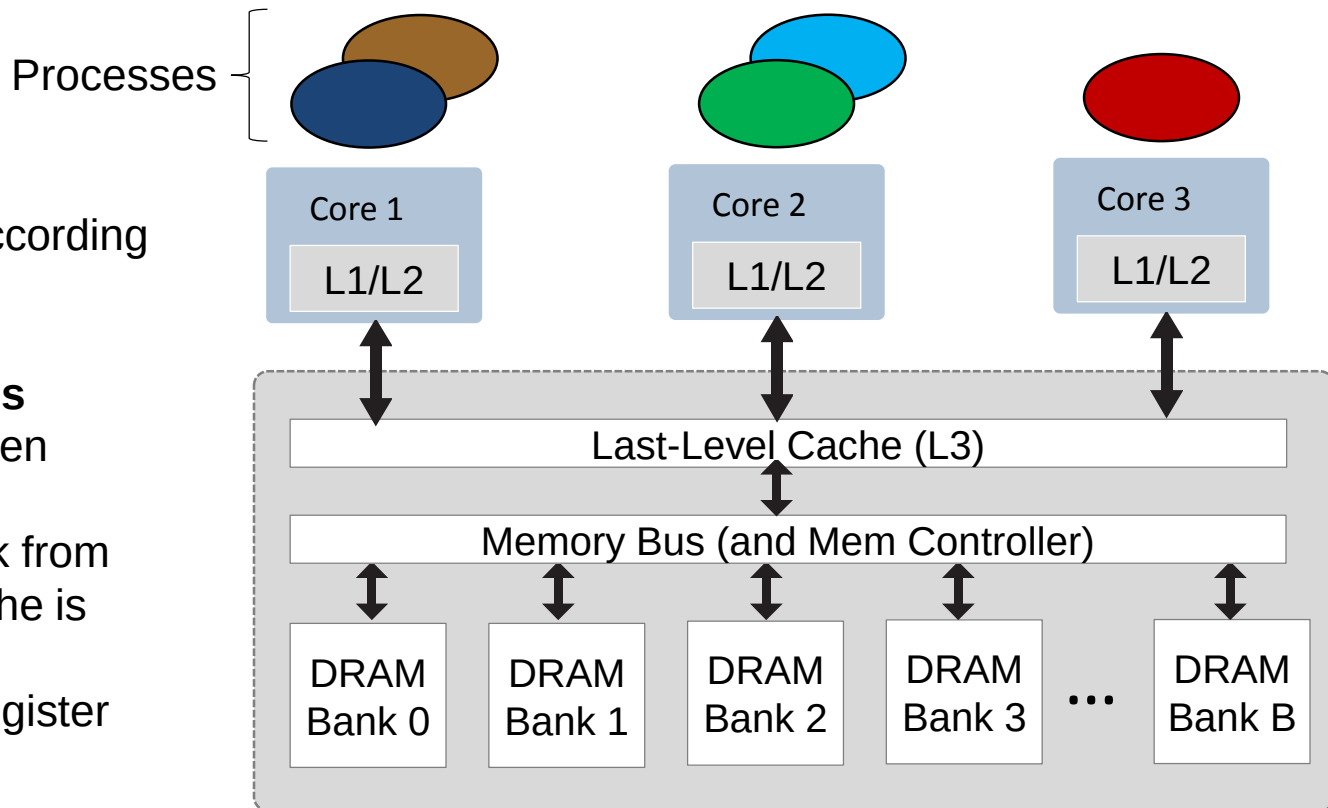


Documented resources

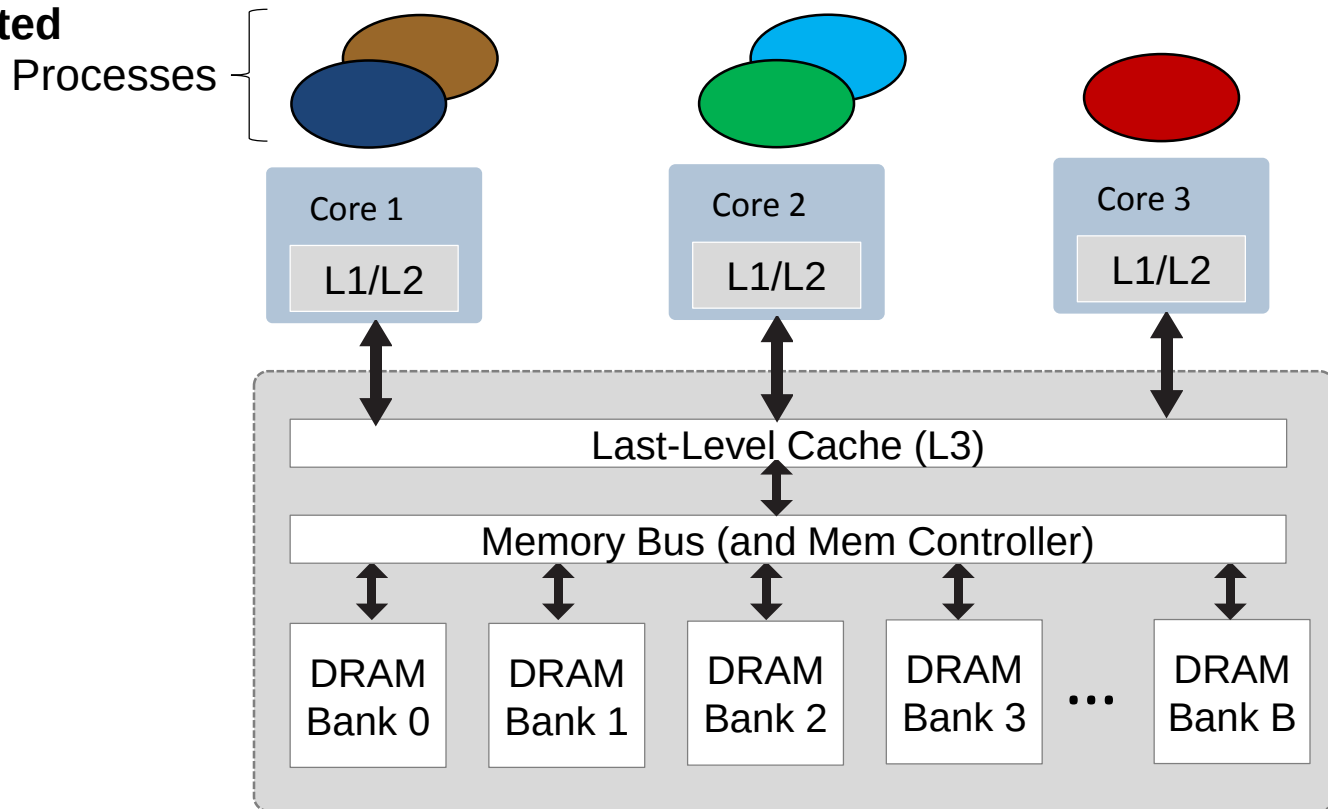
- DRAM memory timing tends to be specified according to JDEC standard.

Undocumented resources

- Memory controller is often undocumented.
- Interconnection network from cores to Last-Level cache is often undocumented.
- Miss-Status Holding Register is often undocumented.



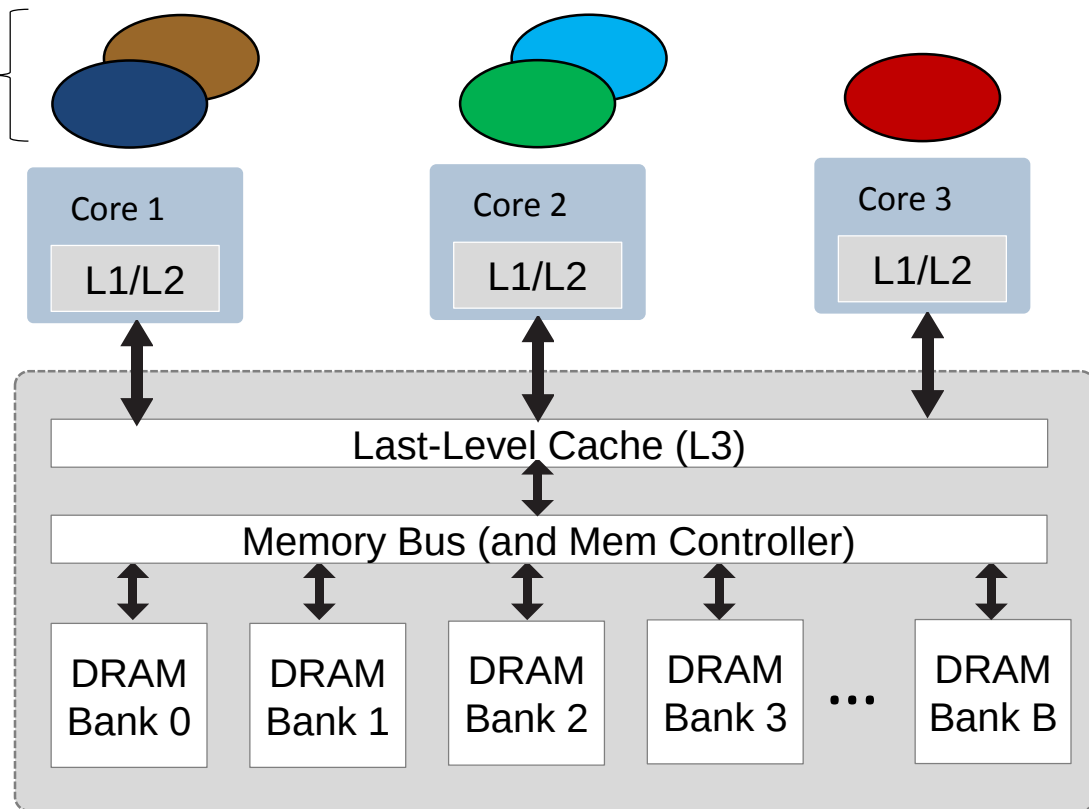
Different reasons for treating resources as undocumented



Different reasons for treating resources as undocumented

- Resources are undocumented

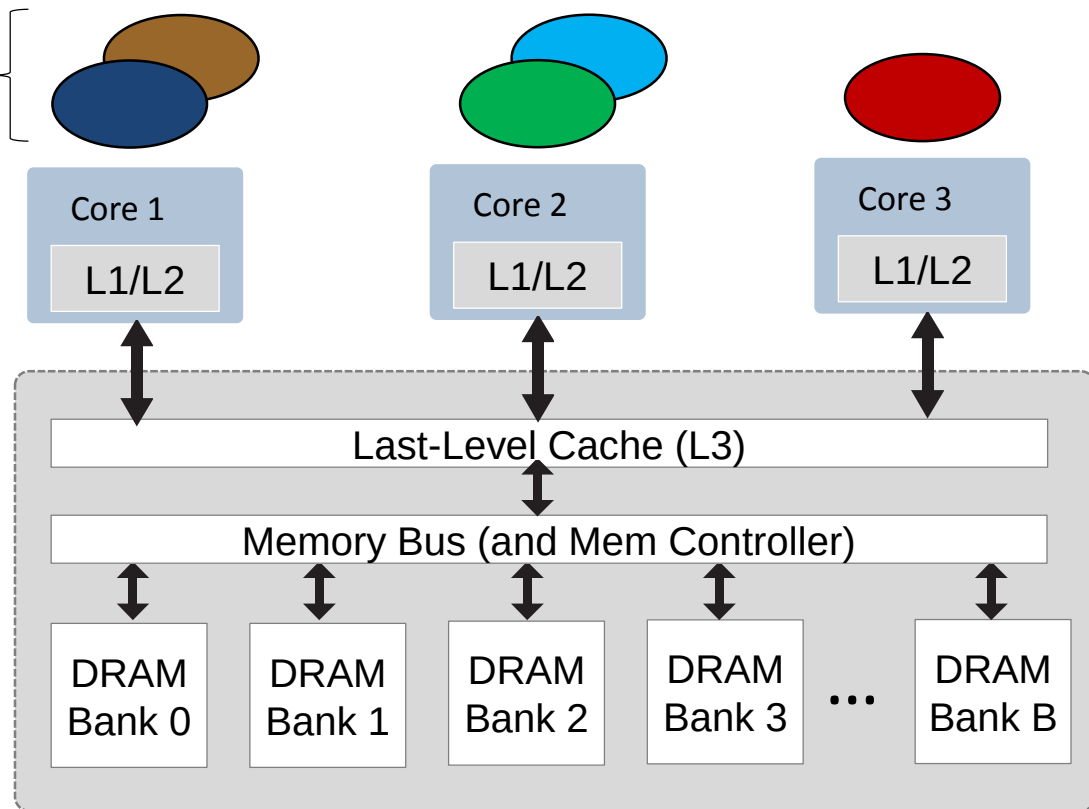
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect

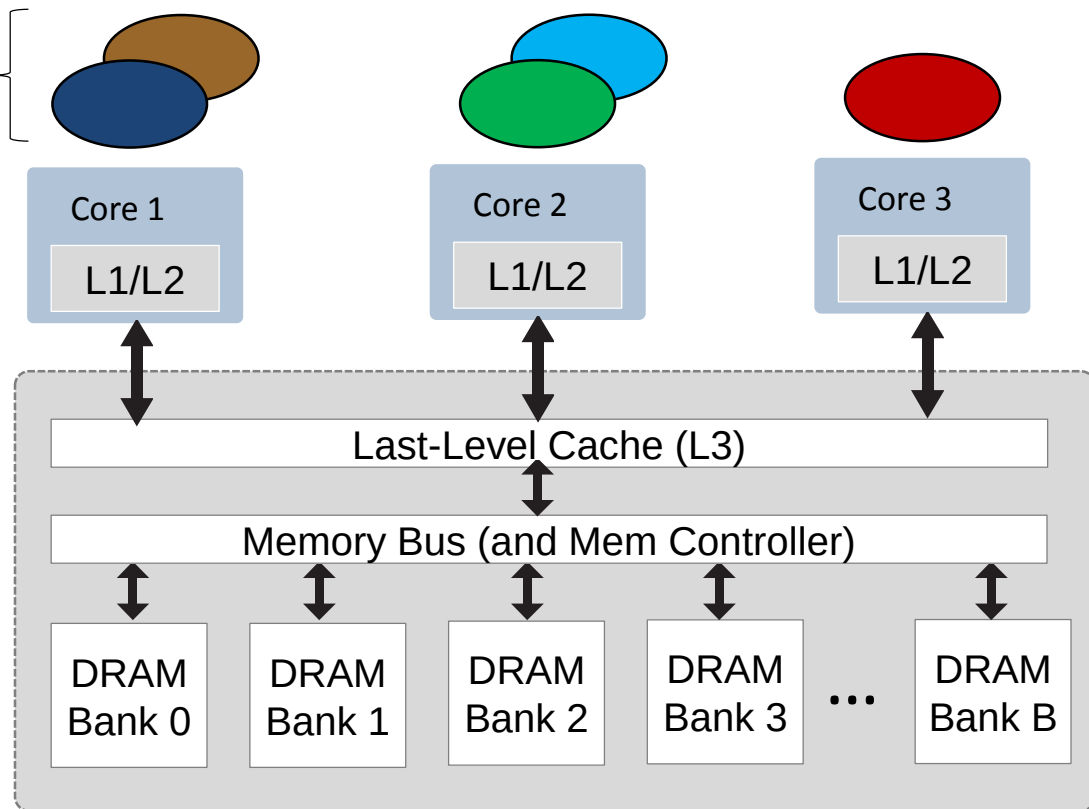
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
 - Heule:PLDI16:
50 out of 1795 x86 instructions have incorrect documentation

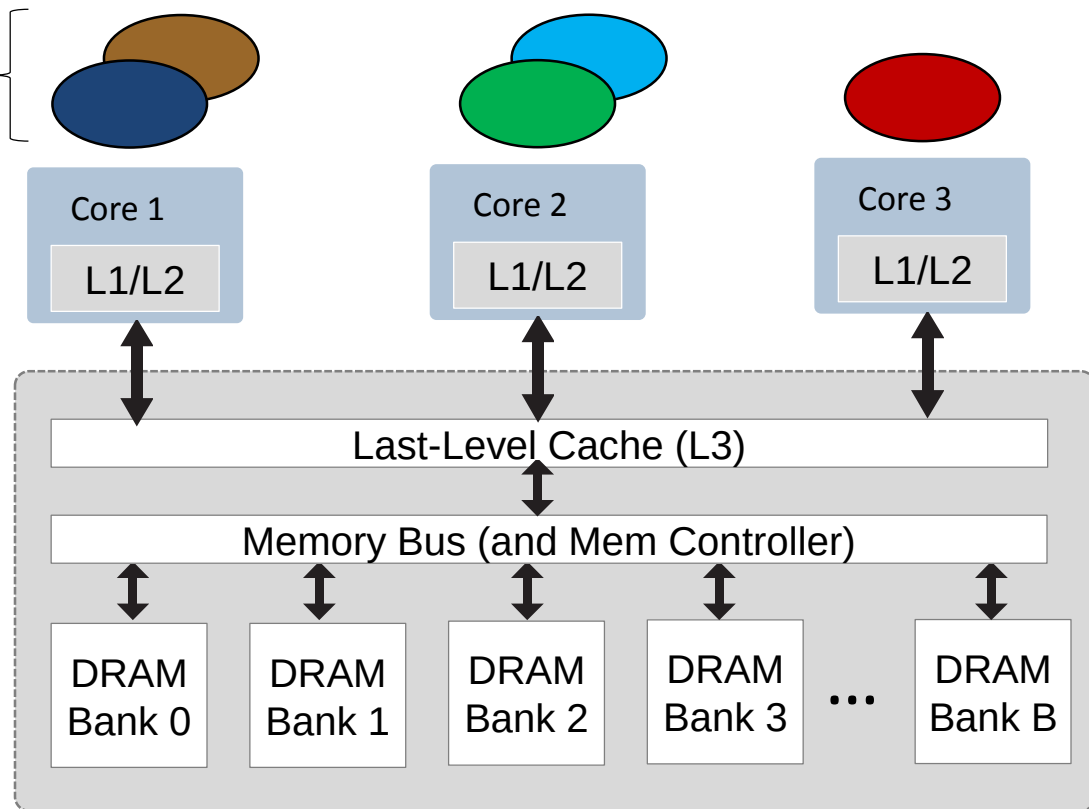
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
 - Heule:PLDI16:
50 out of 1795 x86 instructions have incorrect documentation
 - Dasgupta:PLDI19:
Found incorrect documentation of instructions for x86.

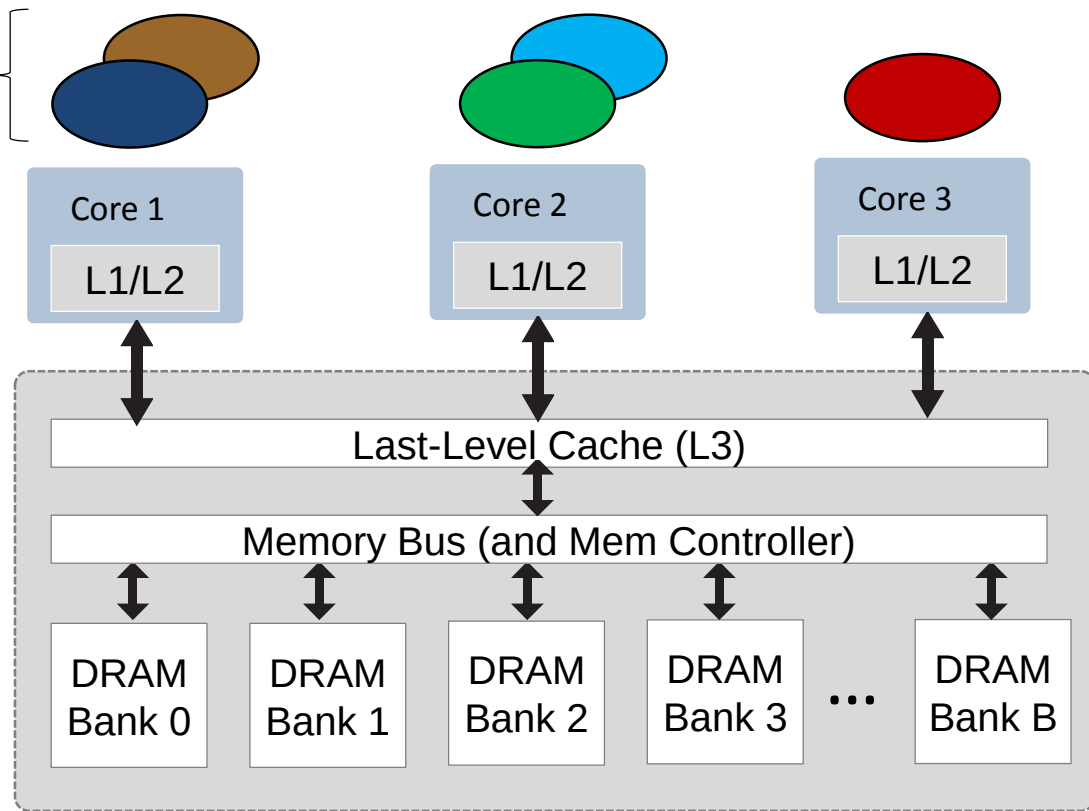
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
 - Heule:PLDI16:
50 out of 1795 x86 instructions have incorrect documentation
 - Dasgupta:PLDI19:
Found incorrect documentation of instructions for x86.
 - Fog19:
There are discrepancies between measured latencies and latencies in data sheets.

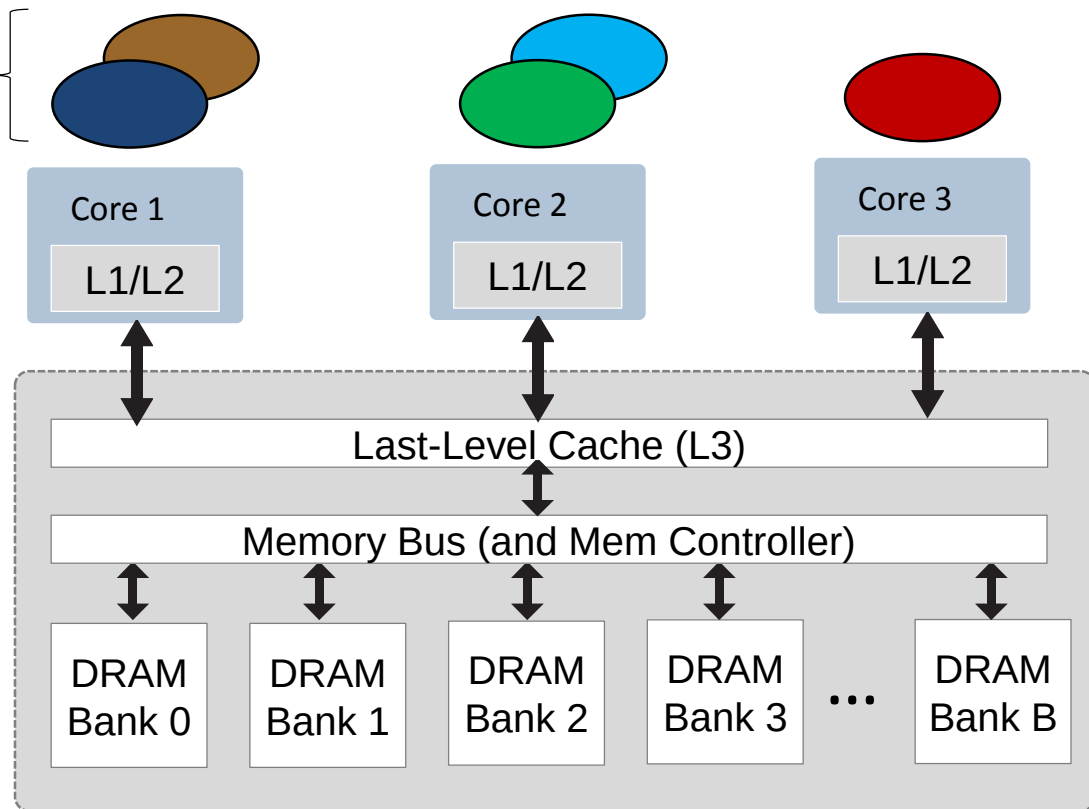
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
- Resources are documented but one believes the documentation to be incorrect

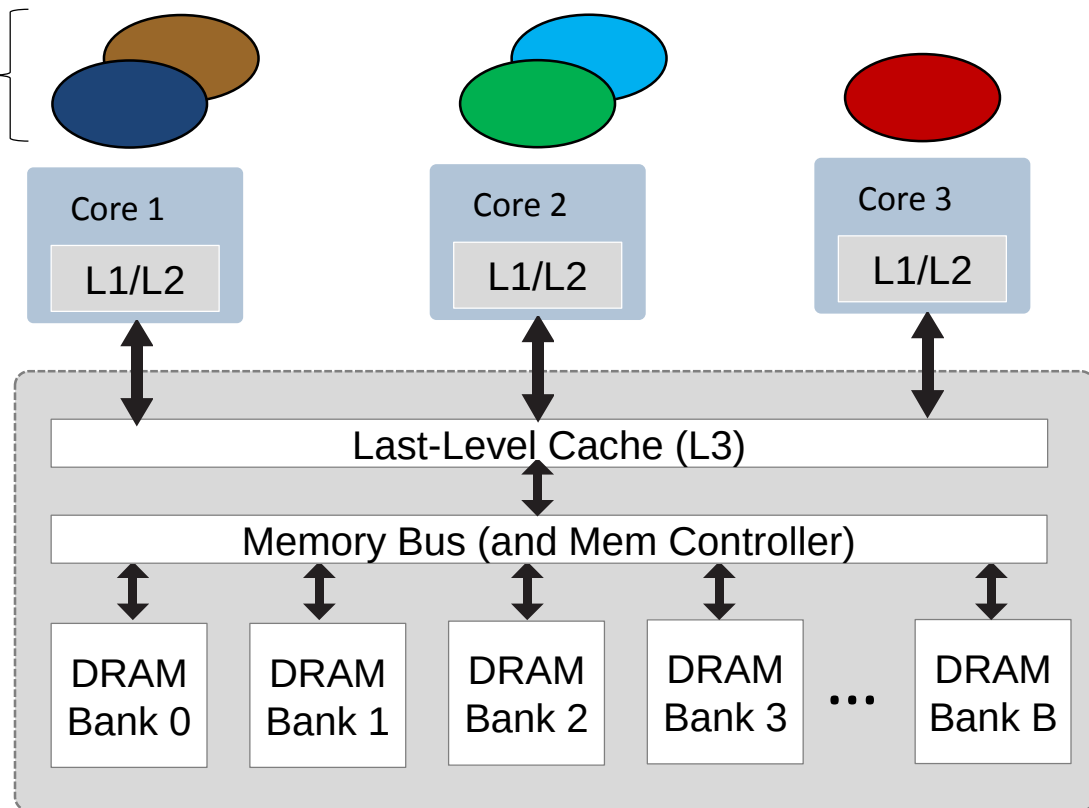
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
- Resources are documented but one believes the documentation to be incorrect
- Resources are documented and one believe documentation to be correct but it is laborious to create a timing model for schedulability analysis

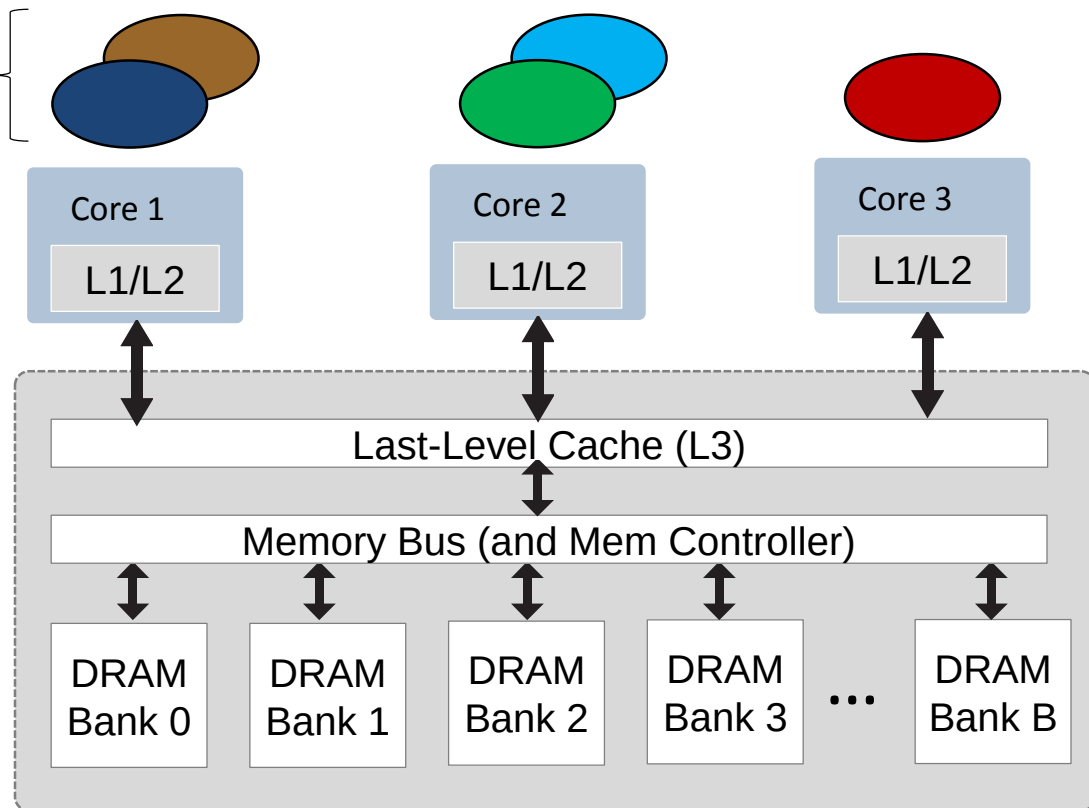
Processes



Different reasons for treating resources as undocumented

- Resources are undocumented
- Resources are documented but documentation is incorrect
- Resources are documented but one believes the documentation to be incorrect
- Resources are documented and one believe documentation to be correct but it is laborious to create a timing model for schedulability analysis (and it needs to be changed when one buys a new chip anyway)

Processes



The consequences of analyzing timing of software executing on multicores using documentation

The consequences of analyzing timing of software executing on multicores using documentation

time



The consequences of analyzing timing of software executing on multicores using documentation

time



Project
starts:
We
want
to
develop
product
X

The consequences of analyzing timing of software executing on multicores using documentation

time



Project starts:
We want to develop product X

Buy multi-core processor Y

The consequences of analyzing timing of software executing on multicores using documentation

time



Project starts: We want to develop product X	Buy multi-core processor Y	Read data sheets of multicore processor Y
---	----------------------------	---

The consequences of analyzing timing of software executing on multicores using documentation

time



Project starts: We want to develop product X	Buy multi-core processor Y	Read data sheets of multicore processor Y	Create model of hardware and develop schedulability analysis equations
---	----------------------------	---	--

The consequences of analyzing timing of software executing on multicores using documentation

time



Project starts: We want to develop product X	Buy multi-core processor Y	Read data sheets of multicore processor Y	Create model of hardware and develop schedulability analysis equations	Deliver product
---	----------------------------	---	--	-----------------

The consequences of analyzing timing of software executing on multicores using documentation

Voice of
the customer:
We need
product X
to use
the new
multicore
processor Y'

time

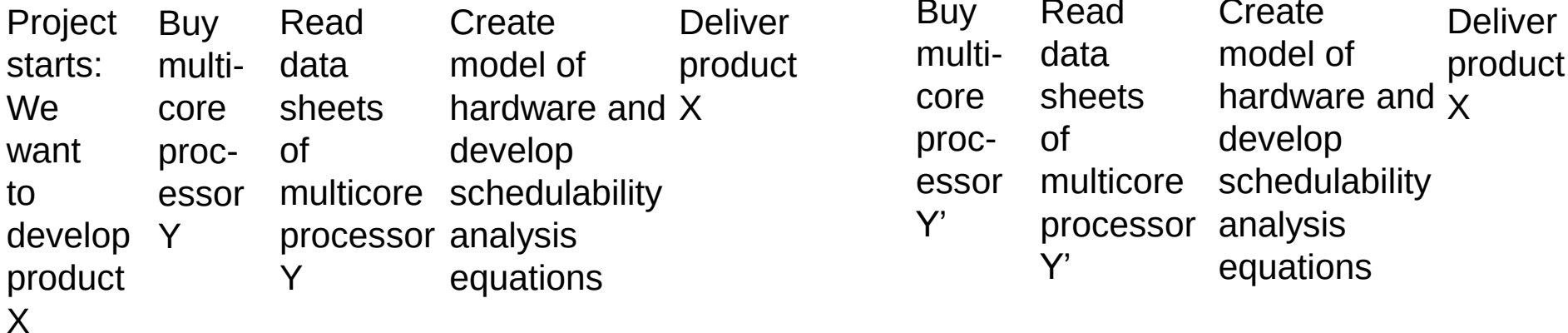


Project starts: We want to develop product X	Buy multi- core proc- essor Y	Read data sheets of multicore processor Y	Create model of hardware and X develop schedulability analysis equations	Deliver product
---	--	---	--	--------------------

The consequences of analyzing timing of software executing on multicores using documentation

Voice of
the customer:
We need
product X
to use
the new
multicore
processor Y'

time

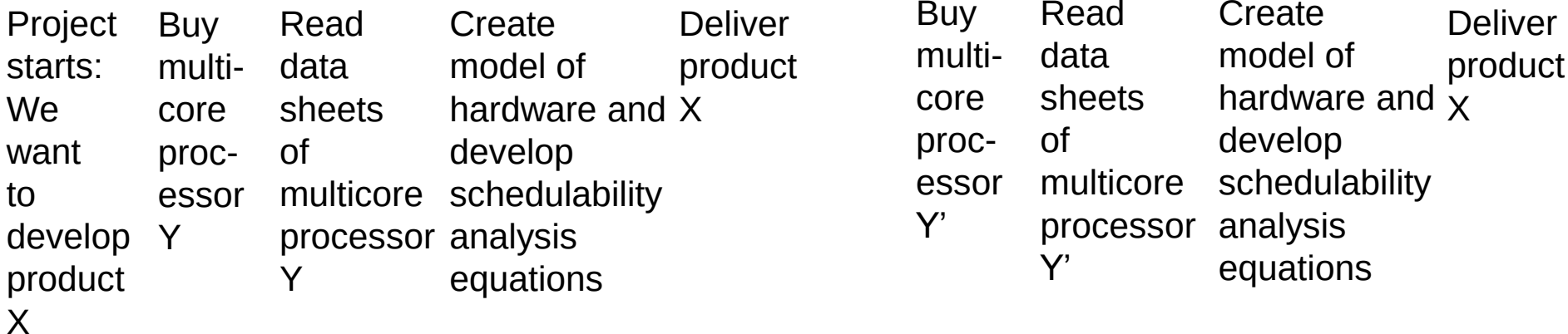


The consequences of analyzing timing of software executing on multicores using documentation

This has to be repeated for each hardware upgrade.
Takes a long time.
Requires PhD level skills.

Voice of the customer:
We need product X to use the new multicore processor Y'

time



The consequences of analyzing timing of software executing on multicores using documentation

New multicore
processor Y
becomes
available

time



The consequences of analyzing timing of software executing on multicores using documentation

New multicore
processor Y
becomes
available

time

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y

The consequences of analyzing timing of software executing on multicores using documentation

New multicore
processor Y
becomes
available

New multicore
processor Y'
becomes
available

time

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y

The consequences of analyzing timing of software executing on multicores using documentation

New multicore
processor Y
becomes
available

New multicore
processor Y'
becomes
available

time



Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y'

The consequences of analyzing timing of software executing on multicores using documentation

New multicore
processor Y
becomes
available

New multicore
processor Y'
becomes
available

New multicore
processor Y''
becomes
available

time

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y'

The consequences of analyzing timing of software executing on multicores using documentation

New multicore
processor Y
becomes
available

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y

New multicore
processor Y'
becomes
available

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y'

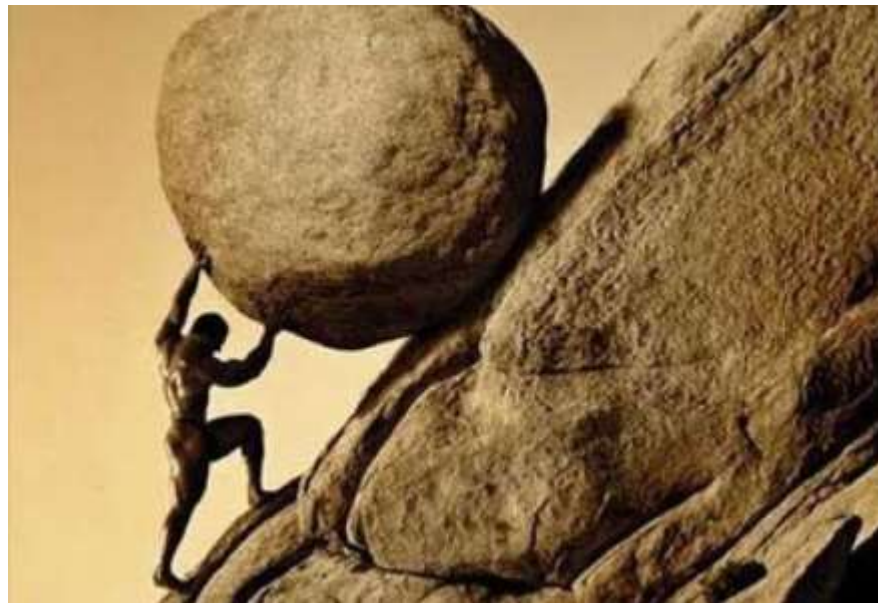
New multicore
processor Y''
becomes
available

Academic
researcher
create
model of
hardware and
develop
schedulability
analysis
equations for Y''

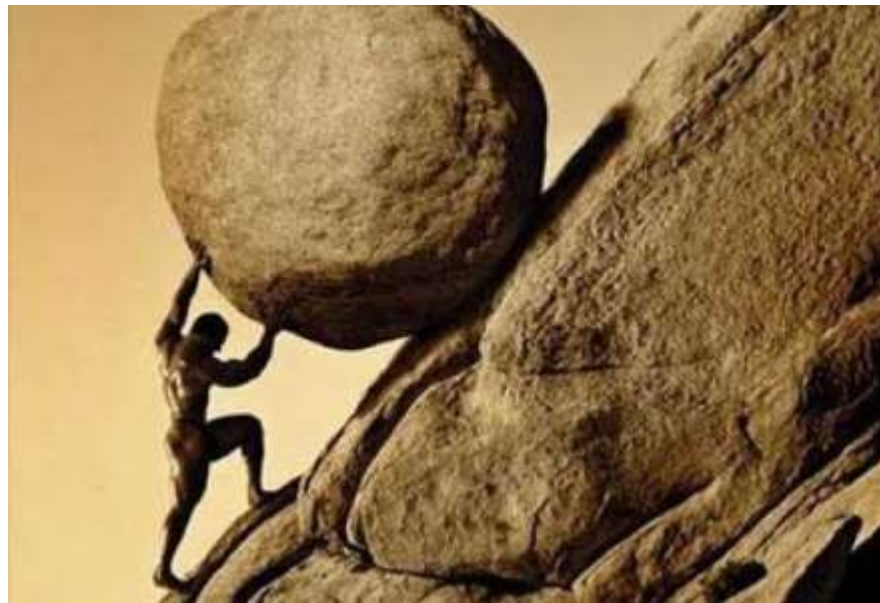
time



The consequences of analyzing timing of software executing on multicores using documentation

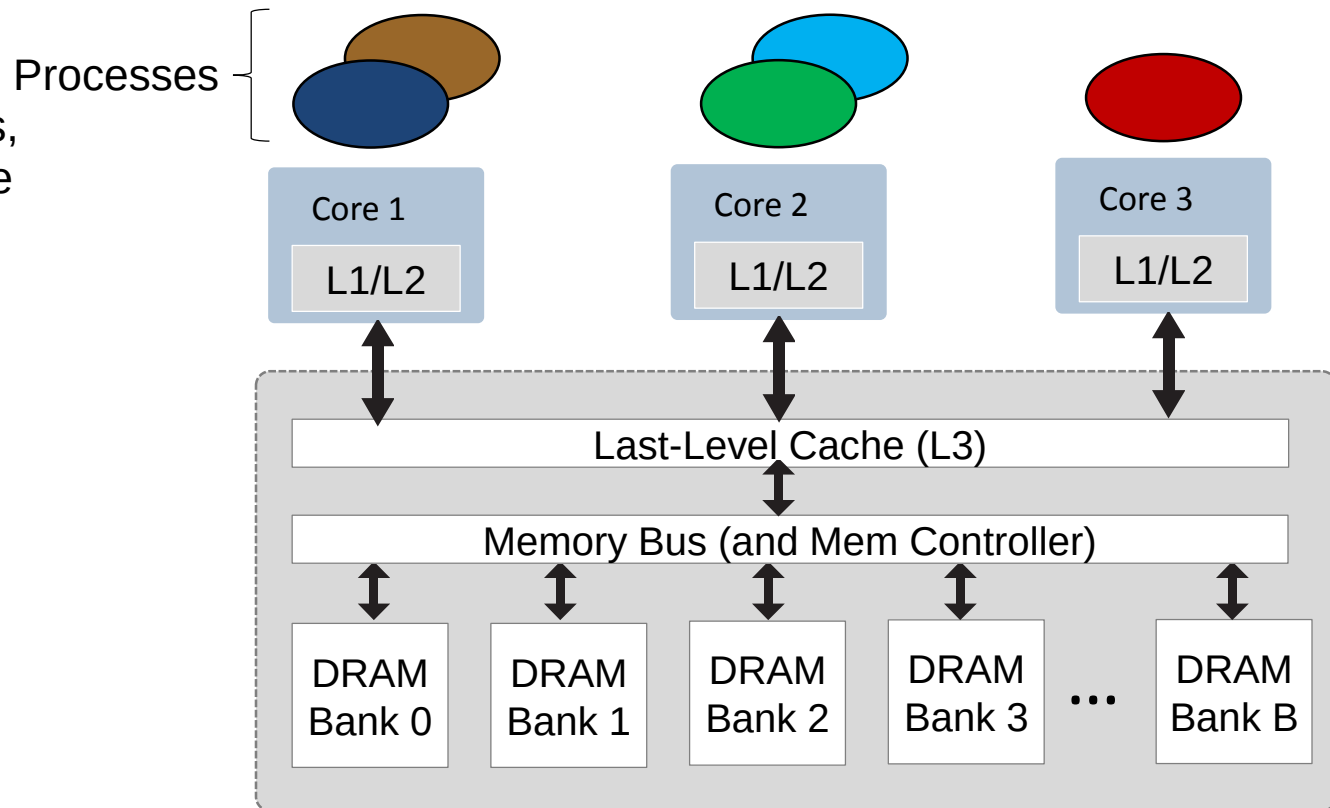


The consequences of analyzing timing of software executing on multicores using documentation



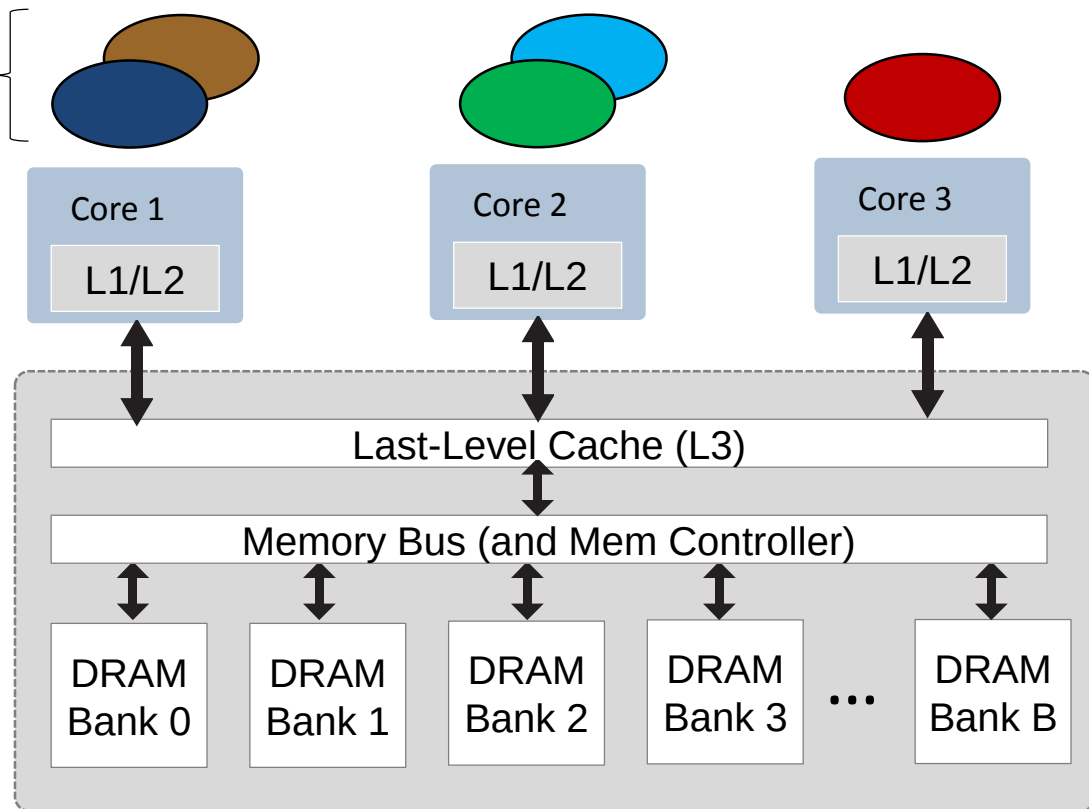
Treating multicore processors as undocumented hardware has the potential to create a model that can be used for processors in the future even for processors that we currently do not know about.

Problem: For each process,
compute its response time



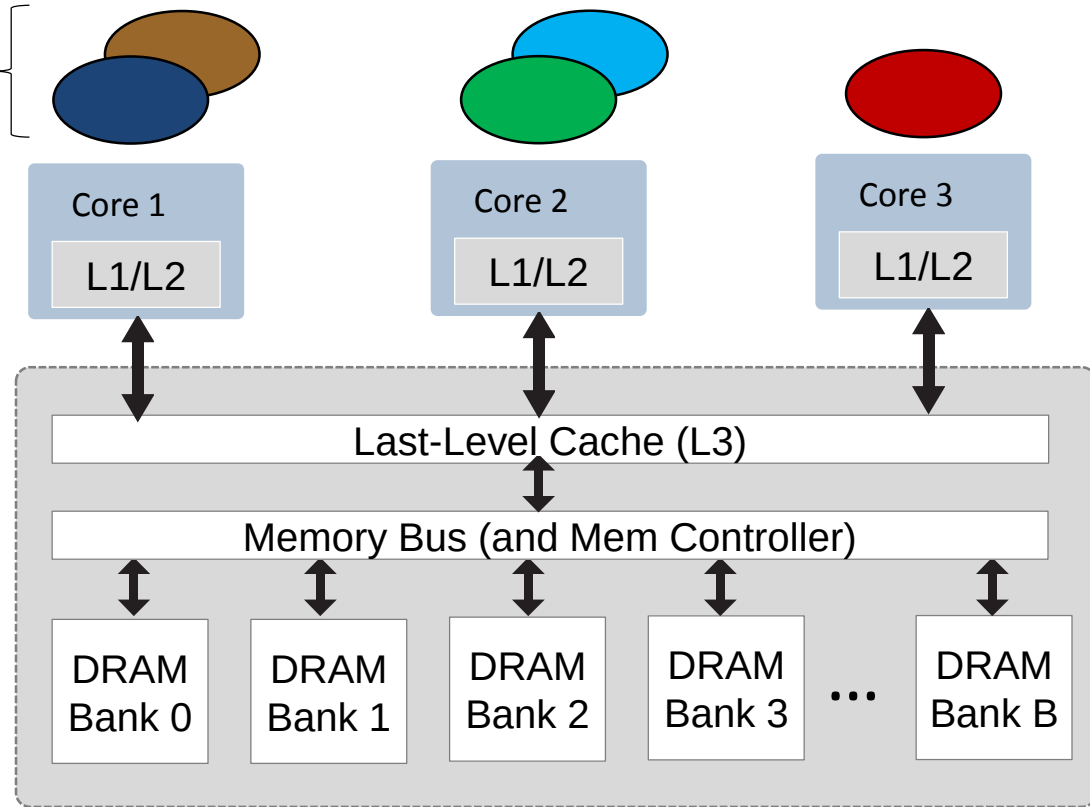
Problem: For each process,
compute an upper bound on its
response time

Processes



Processes

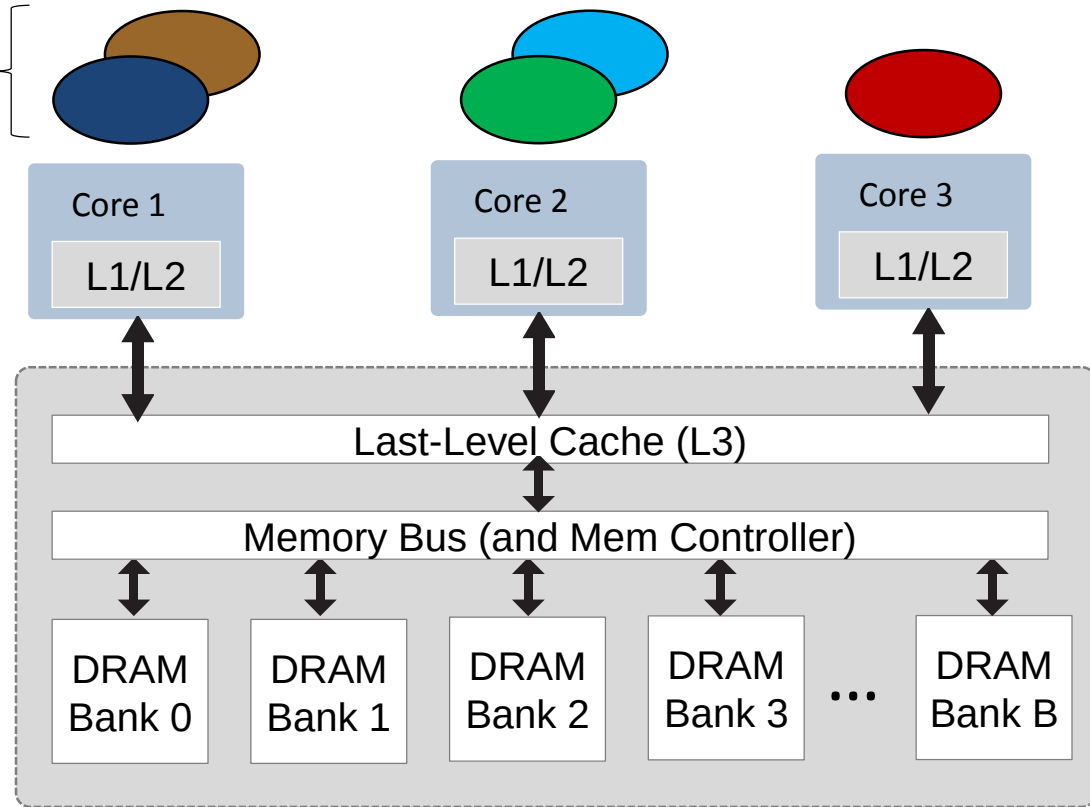
Problem: For each process, compute an upper bound on its response time considering contention for resources in the memory system and that resources in the memory system are undocumented.



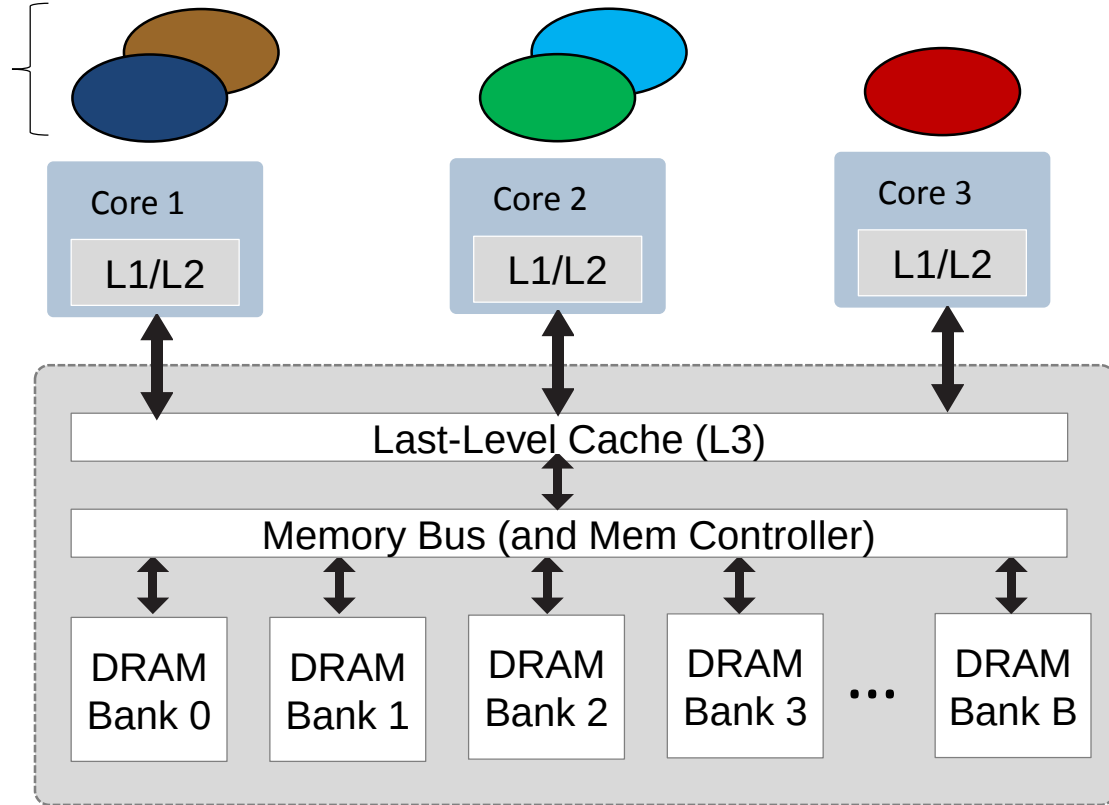
Processes

Problem: For each process, compute an upper bound on its response time considering contention for resources in the memory system and that resources in the memory system are undocumented.

Q: How can we analyze timing of a system when we do not know how the system works?



Processes

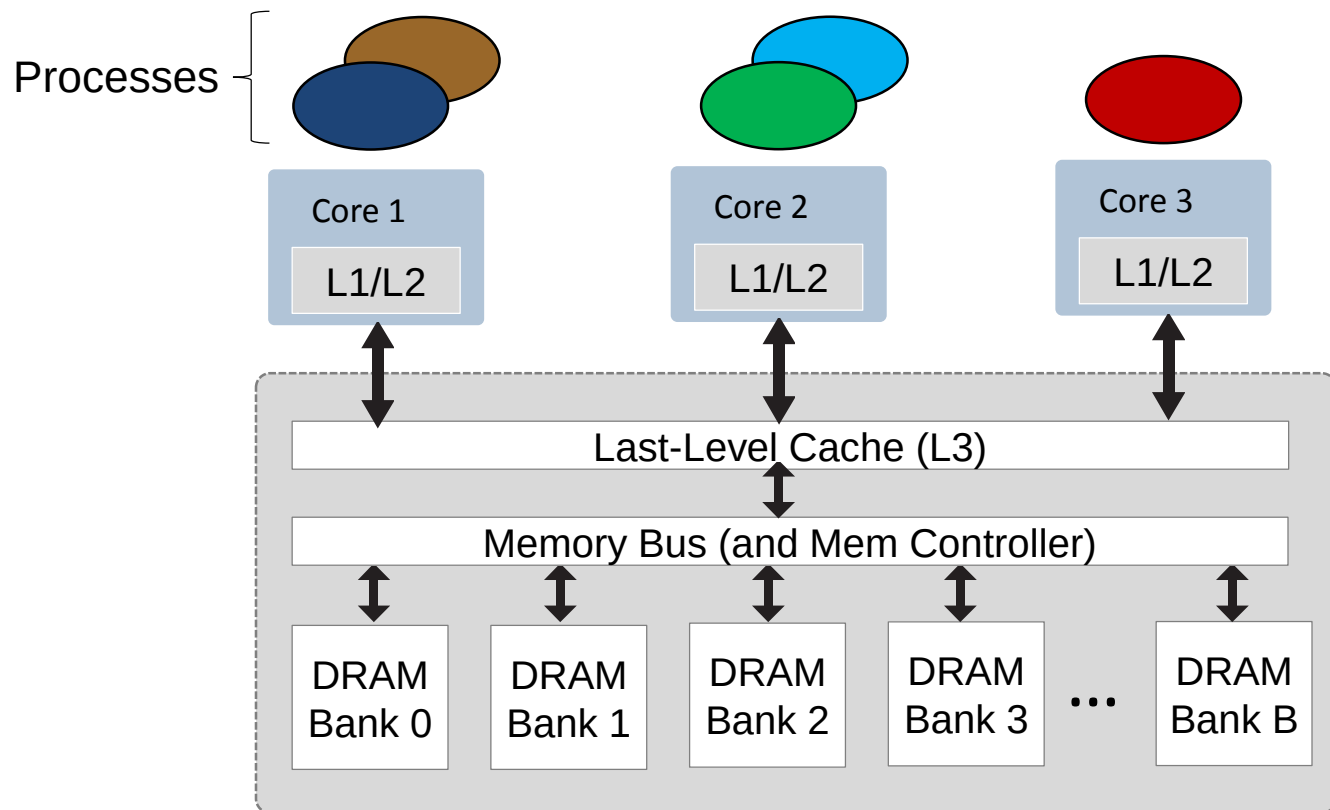


Problem: For each process, compute an upper bound on its response time considering contention for resources in the memory system and that resources in the memory system are undocumented.

Q: How can we analyze timing of a system when we do not know how the system works?

A: Create abstraction that describes effect of undocumented h/w.

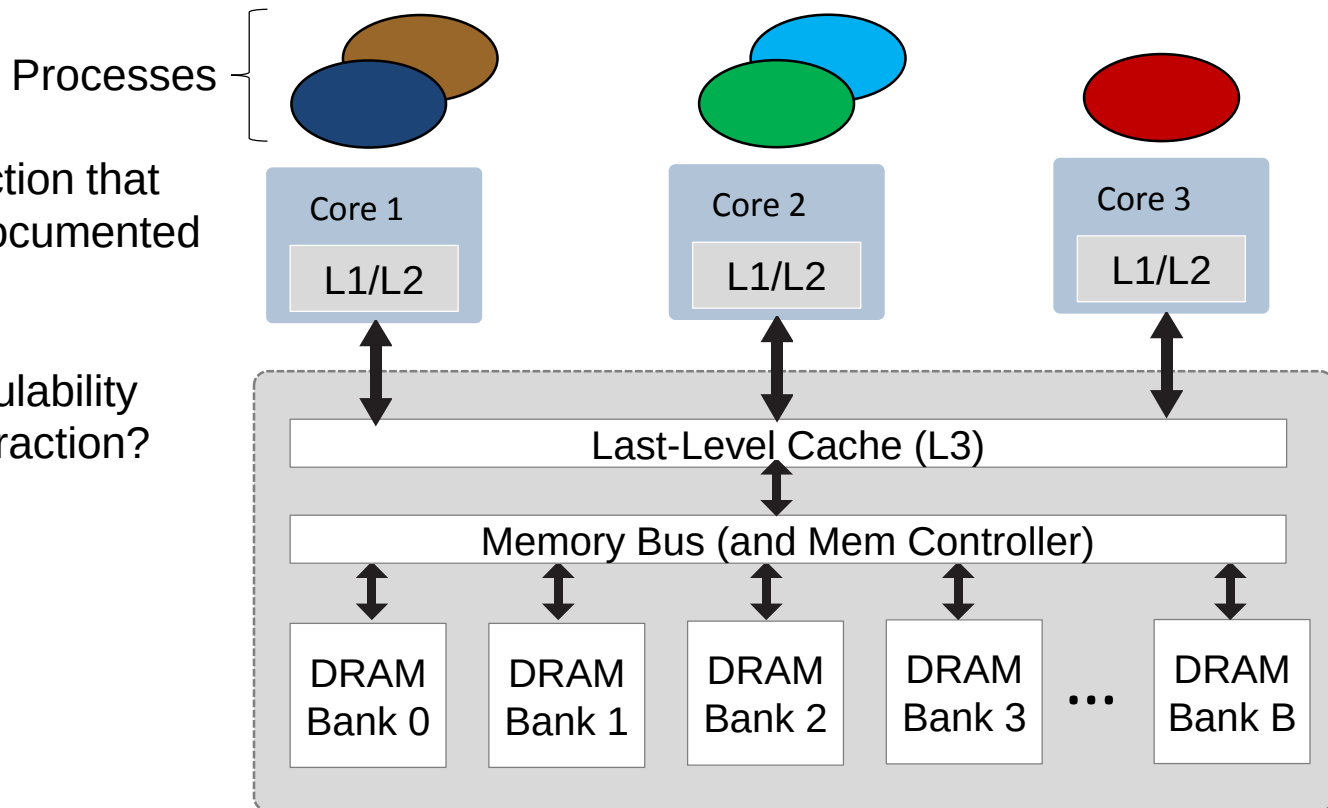
Big Research Questions



Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?



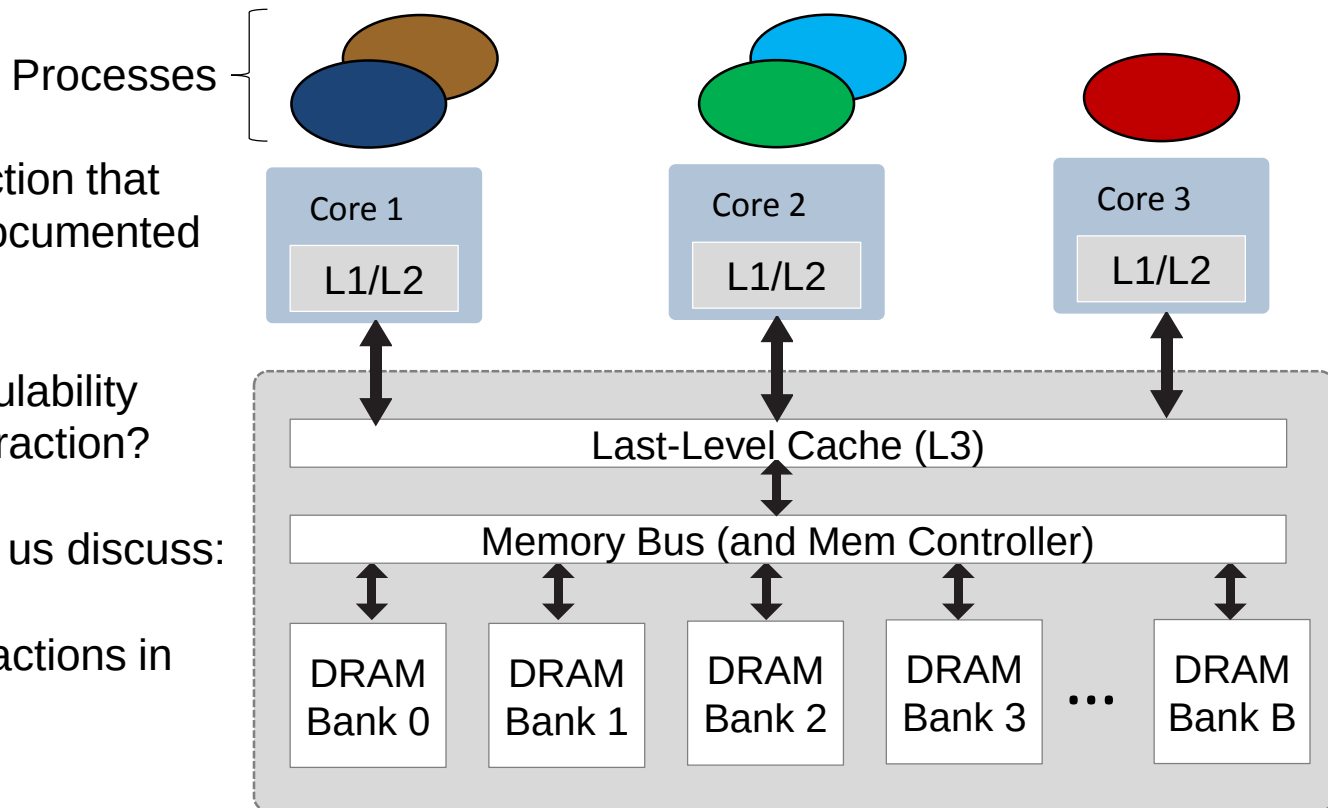
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Before discussing them, let us discuss:

1. Other approaches
2. General ideas for abstractions in other disciplines



Other approaches

Other approaches

Ignore timing
requirements

Other approaches

Ignore timing
requirements

Ignore timing
aspects of
memory system

Other approaches

Ignore timing
requirements

Ignore timing
aspects of
memory system

Disable all
processor cores
except one

Other approaches

Ignore timing
requirements

Ignore timing
aspects of
memory system

Disable all
processor cores
except one

Use simple
processor and
run heavy
computations on
GPU

Other approaches

Ignore timing
requirements

Ignore timing
aspects of
memory system

Disable all
processor cores
except one

Use simple
processor and
run heavy
computations on
GPU

Build your own
processor (ASIC)

Other approaches

Ignore timing
requirements

Ignore timing
aspects of
memory system

Disable all
processor cores
except one

Use simple
processor and
run heavy
computations on
GPU

Build your own
processor (ASIC)

Build your own
processor (FPGA)

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis

Model some resources and create analysis

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis

Model some resources and create analysis

Software-based mechanism to improve predictability/isolation

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Build your own processor (FPGA)

Use well-documented hardware (e.g., RISC-V) and model all resources and create analysis

Model some resources and create analysis

Software-based mechanism to improve predictability/isolation

Reorganize program so that at most one program accesses memory at a time

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Pro: Easy
Con: Potentially unsafe

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Pro: Easy
Con: Potentially unsafe

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Pro: Easy

Con: Lose lots of processing capacity

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Con: Analyzing timing of a single “task” executing on a GPU is still hard.

Other approaches

Ignore timing requirements

Ignore timing aspects of memory system

Disable all processor cores except one

Use simple processor and run heavy computations on GPU

Build your own processor (ASIC)

Con: High fixed cost.

Other approaches

Con: Low clock frequency

Build your own
processor (FPGA)

Use well-
documented
hardware (e.g.,
RISC-V) and
model all
resources and
create analysis

Model some
resources and
create analysis

Software-based
mechanism to
improve
predictability/iso
lation

Reorganize
program so that
at most one
program
accesses
memory at a
time

Other approaches

Con: Limits #suppliers. Laborious.

Build your own
processor (FPGA)

Use well-
documented
hardware (e.g.,
RISC-V) and
model all
resources and
create analysis

Model some
resources and
create analysis

Software-based
mechanism to
improve
predictability/iso
lation

Reorganize
program so that
at most one
program
accesses
memory at a
time

Other approaches

Con: Laborious. There may be many resources for which documentation does not exist.

Build your own
processor (FPGA)

Use well-
documented
hardware (e.g.,
RISC-V) and
model all
resources and
create analysis

Model some
resources and
create analysis

Software-based
mechanism to
improve
predictability/iso
lation

Reorganize
program so that
at most one
program
accesses
memory at a
time

Other approaches

Examples: Cache coloring, Cache locking, Bank coloring, MemGuard, TLB coloring.

Con: Requires changes to operating system. Only work for some resources; there are many resources for which there is no isolation mechanism.

Build your own
processor (FPGA)

Use well-
documented
hardware (e.g.,
RISC-V) and
model all
resources and
create analysis

Model some
resources and
create analysis

Software-based
mechanism to
improve
predictability/iso
lation

Reorganize
program so that
at most one
program
accesses
memory at a
time

Other approaches

Examples: PREM, Linkoping@RTSS07.

Con: Laborious. Requires a local memory that is large enough to store working set. Difficult to prove that no memory accesses occurs in certain phases.

Build your own
processor (FPGA)

Use well-
documented
hardware (e.g.,
RISC-V) and
model all
resources and
create analysis

Model some
resources and
create analysis

Software-based
mechanism to
improve
predictability/iso
lation

Reorganize
program so that
at most one
program
accesses
memory at a
time

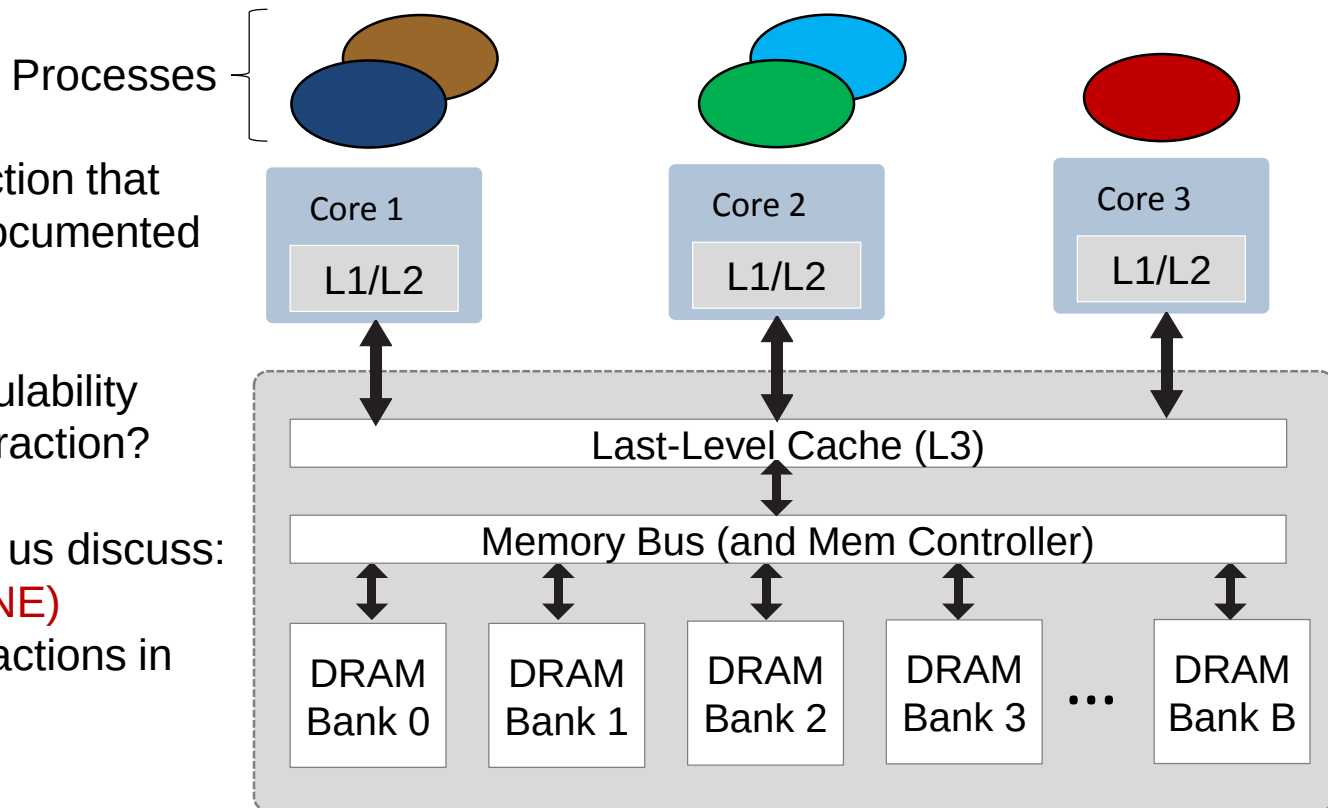
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

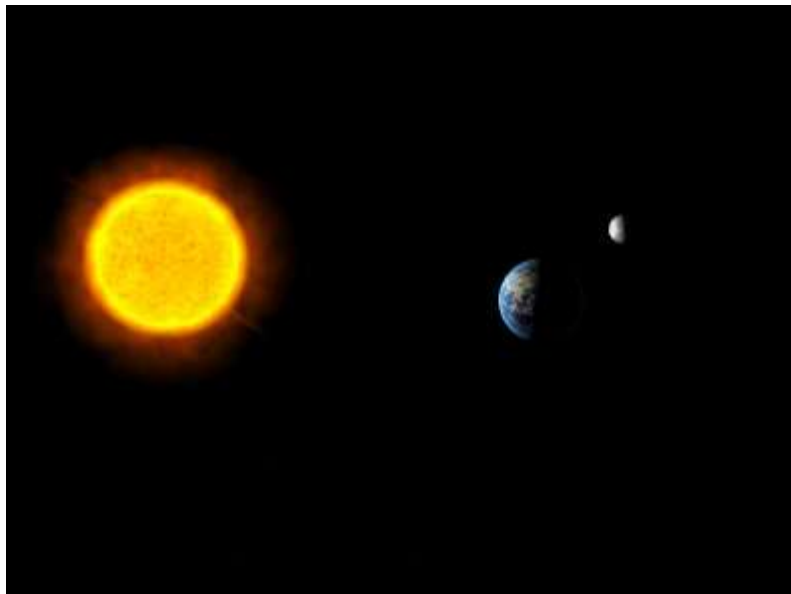
Before discussing them, let us discuss:

1. Other approaches (**DONE**)
2. General ideas for abstractions in other disciplines

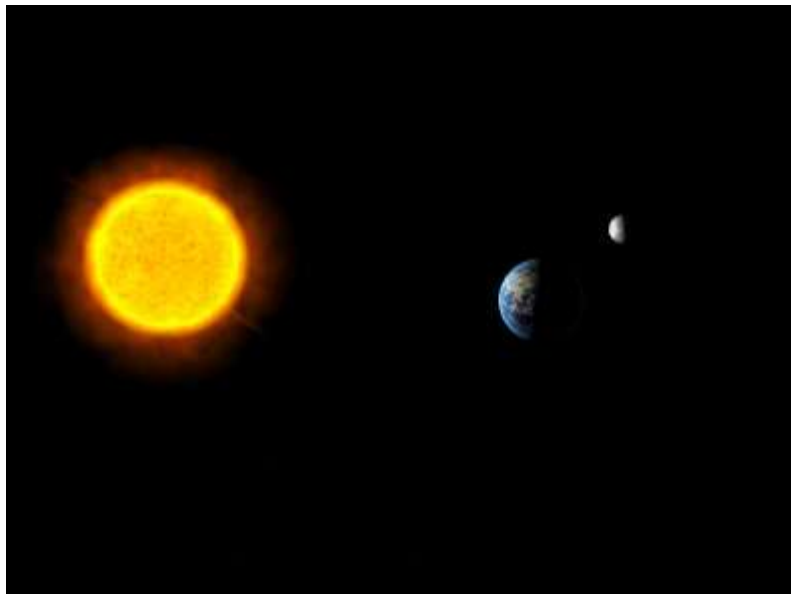


General ideas for abstractions in other disciplines

General ideas for abstractions in other disciplines

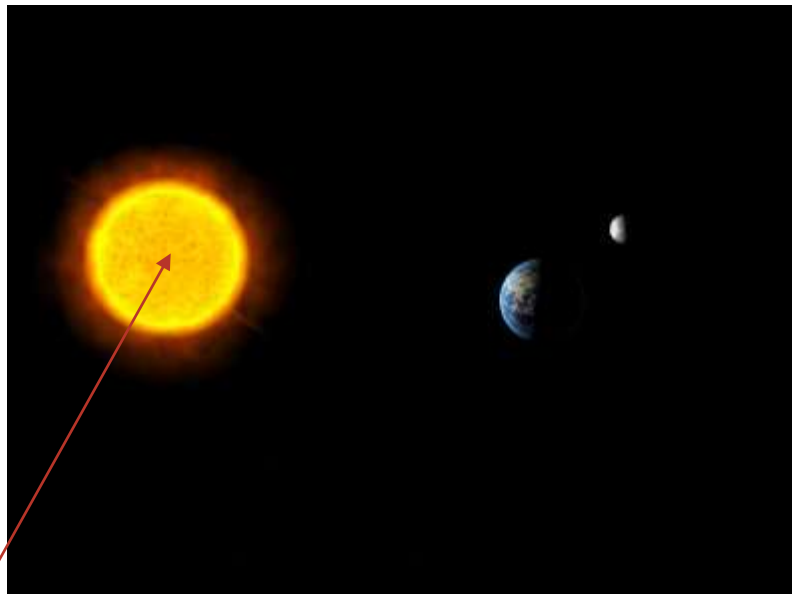


General ideas for abstractions in other disciplines



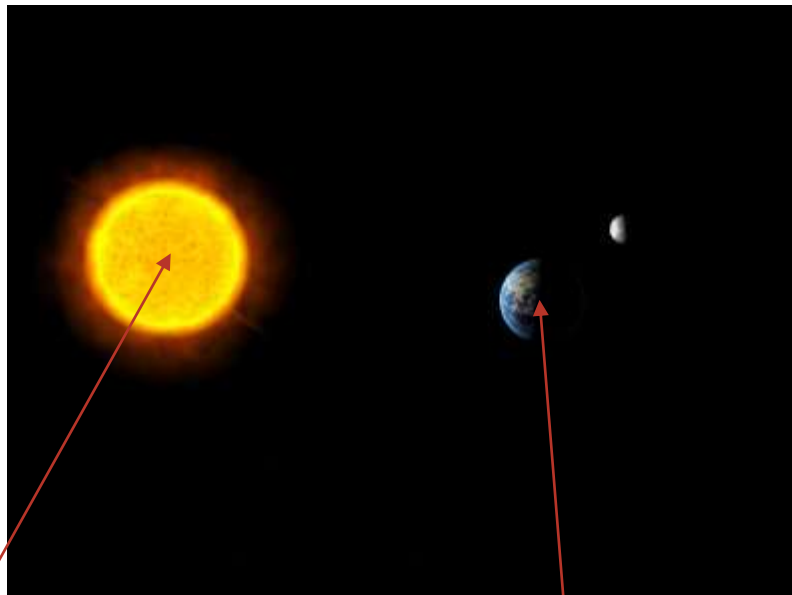
Not drawn to scale

General ideas for abstractions in other disciplines



This body has many atoms

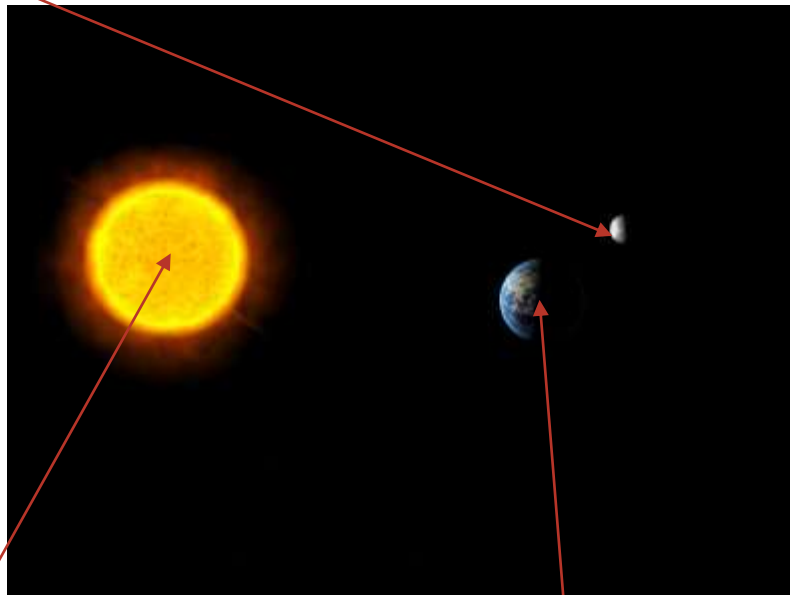
General ideas for abstractions in other disciplines



This body has many atoms This body has many atoms

General ideas for abstractions in other disciplines

This body also has many atoms

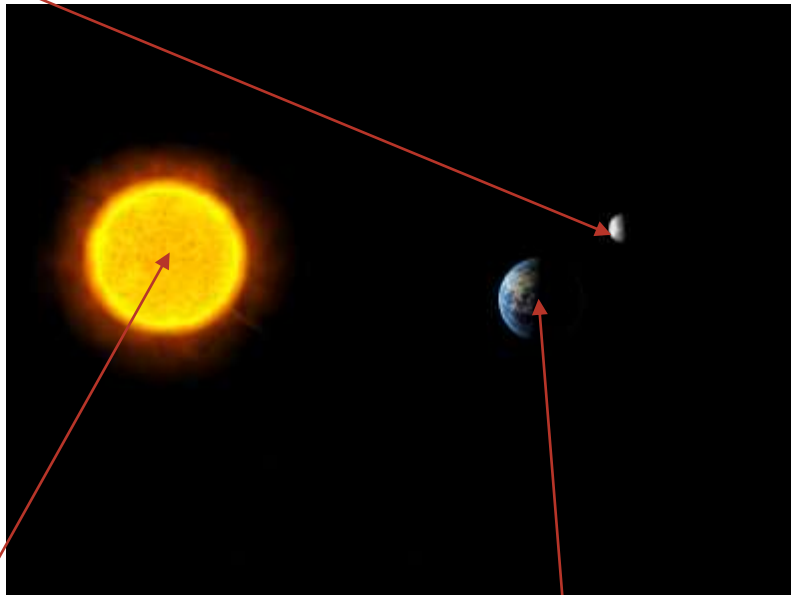


This body has many atoms

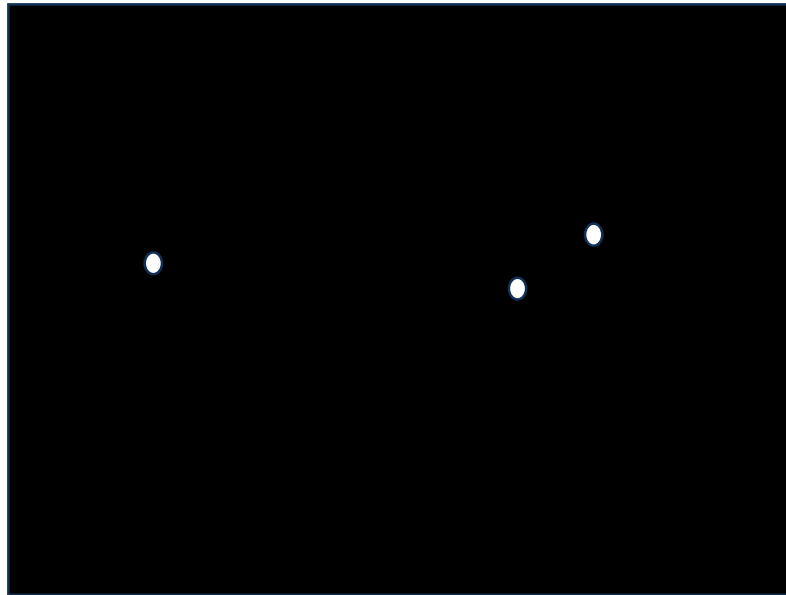
This body has many atoms

General ideas for abstractions in other disciplines

This body also has many atoms



We describe each body as a single point mass

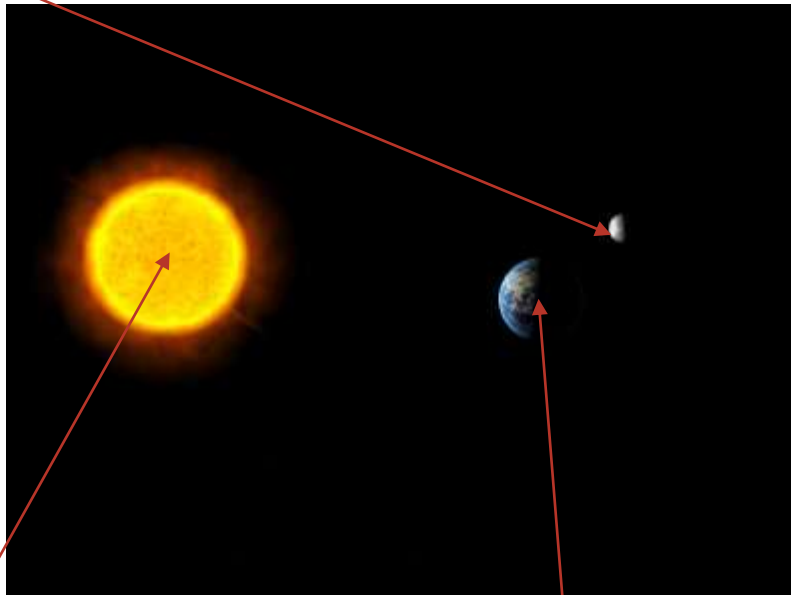


This body has many atoms

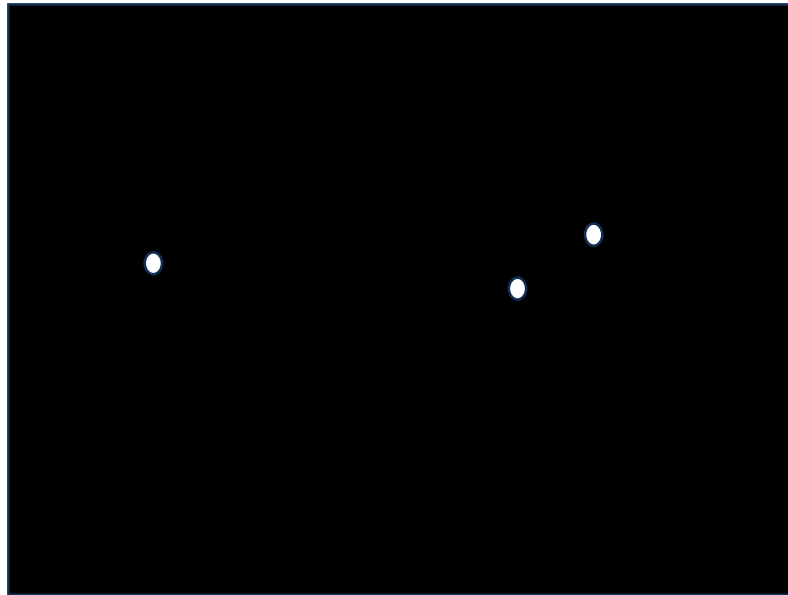
This body has many atoms

General ideas for abstractions in other disciplines

This body also has many atoms



We describe each body as a single point mass

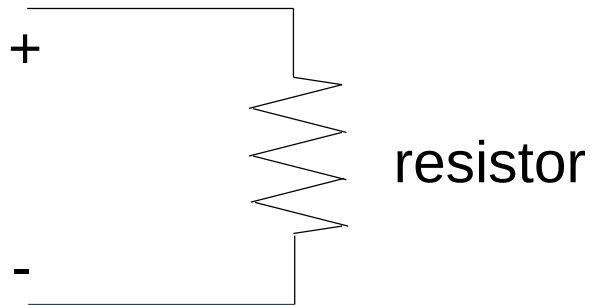


This body has many atoms

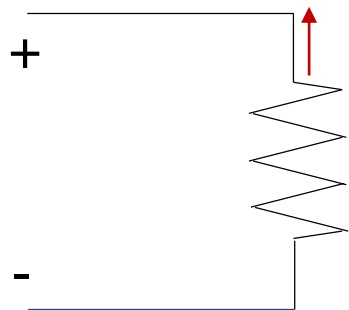
This body has many atoms

Describe each body with only as many parameters that we need for the analysis that we want to do.

General ideas for abstractions in other disciplines

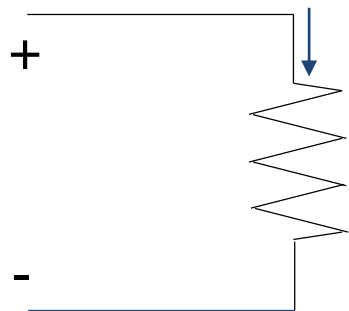


General ideas for abstractions in other disciplines



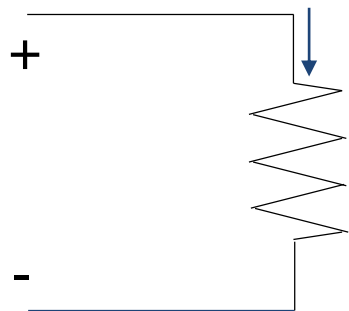
How many electrons per time unit are flowing **here**?

General ideas for abstractions in other disciplines



How much current is flowing **here**?

General ideas for abstractions in other disciplines



How much current is flowing **here**?

Ask questions about the aggregate that you care about.

General ideas for abstractions in other disciplines



For **each** of these 10^{24} water molecules,
how fast does this water molecule move?

General ideas for abstractions in other disciplines



What is the water temperature?

Ask questions about the aggregate that you care about.

General ideas for abstractions in other disciplines



What is the water temperature?

If possible: Describe a system with quantities that you can measure.

General ideas for abstractions in other disciplines

1. Describe a system with parts
2. Describe each part with an abstraction
3. Obtain specific values of the abstraction (e.g., using measurements) for each part
4. Ask questions: calculate the answer to the questions

General ideas for abstractions

1. Describe a system with parts
2. Describe each part with an abstraction
3. Obtain specific values of the abstraction (e.g., using measurements) for each part
4. Ask questions: calculate the answer to the questions

General ideas for abstractions

1. Describe a system with parts
A software system comprises a set of tasks.
2. Describe each part with an abstraction
Each task is described with a T (period), D (deadline), and C (execution time).
3. Obtain specific values of the abstraction (e.g., using measurements) for each part
Obtain T from source code. Obtain D from software requirement specification.
Obtain C by measuring the execution of the program; then add margin.
(or run WCET tool)
4. Ask questions: calculate the answer to the questions
Will all deadlines be met if tasks are scheduled by Rate-Monotonic on a single processor?

General ideas for abstractions

1. Describe a system with parts
A software system comprises a set of tasks.
2. Describe each part with an abstraction
Each task is described with a T (period), D (deadline), and C (execution time).
Describe the **effect** of the memory system on execution speed of a task.
3. Obtain specific values of the abstraction (e.g., using measurements) for each part
Obtain T from source code. Obtain D from software requirement specification.
Obtain C by measuring the execution of the program; then add margin.
(or run WCET tool)
Obtain parameters (e.g., using measurements)
4. Ask questions: calculate the answer to the questions
Will all deadlines be met if tasks are scheduled by Rate-Monotonic on a single processor?

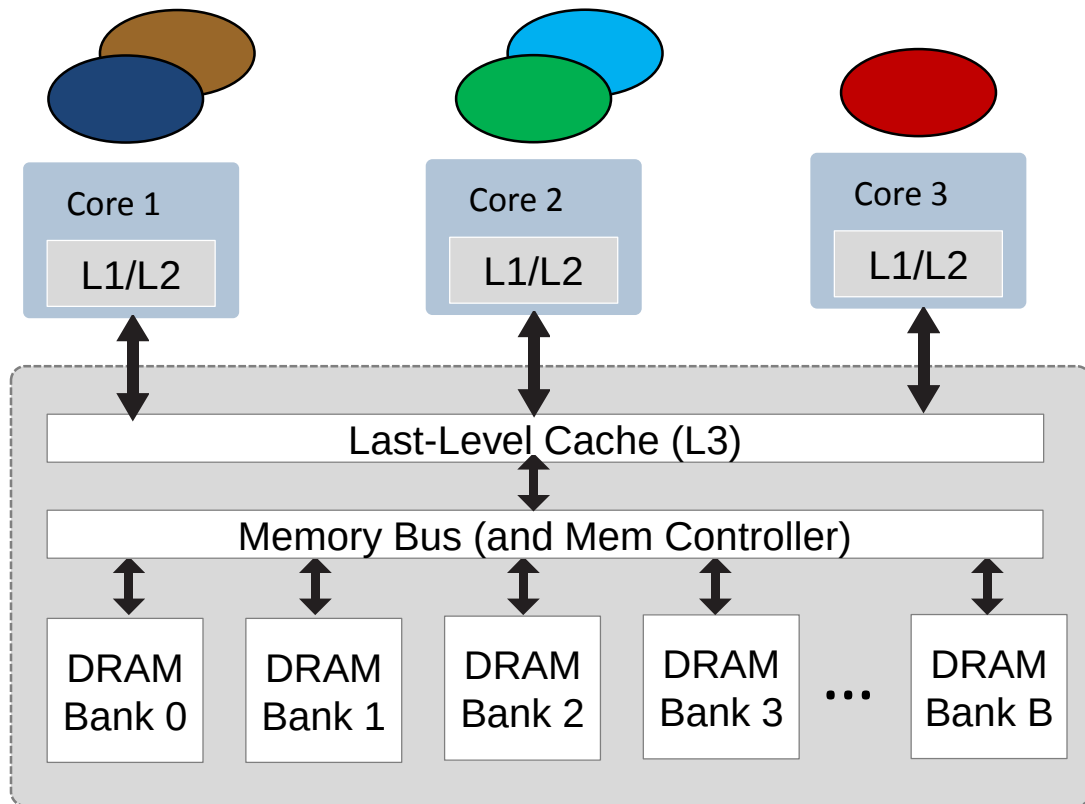
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Before discussing them, let us discuss:

1. Other approaches (**DONE**)
2. General ideas for abstractions in other disciplines (**DONE**)

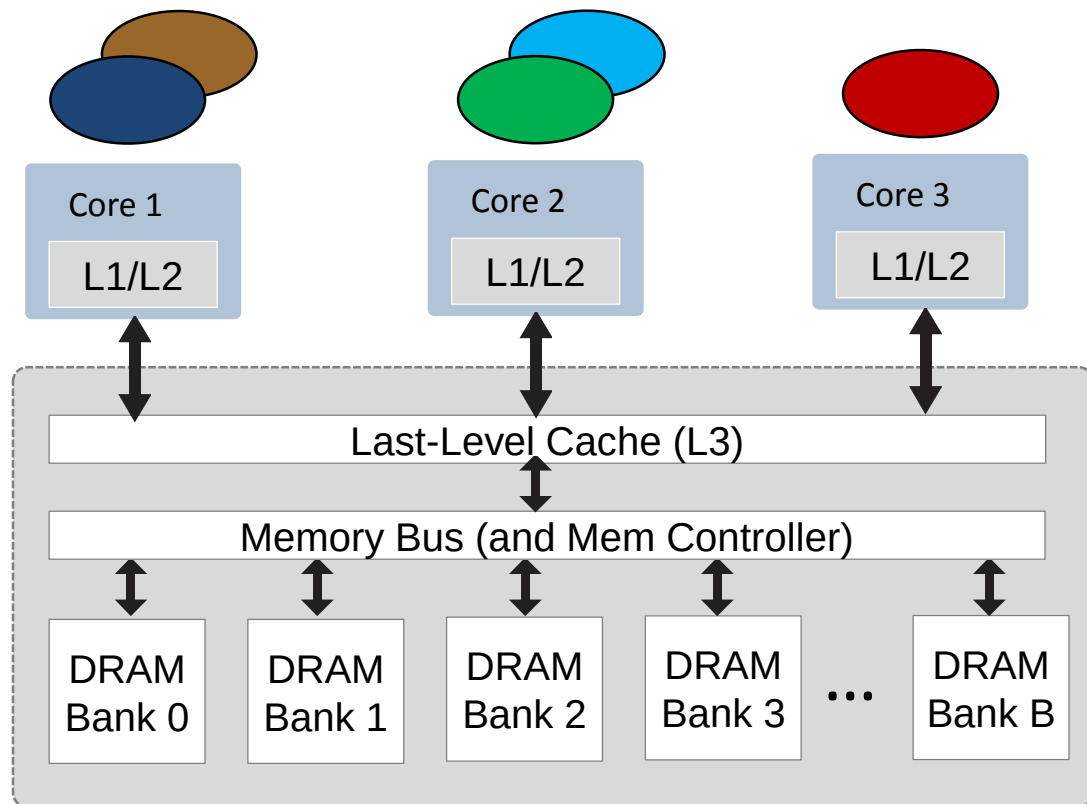


Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Let us discuss Q1.

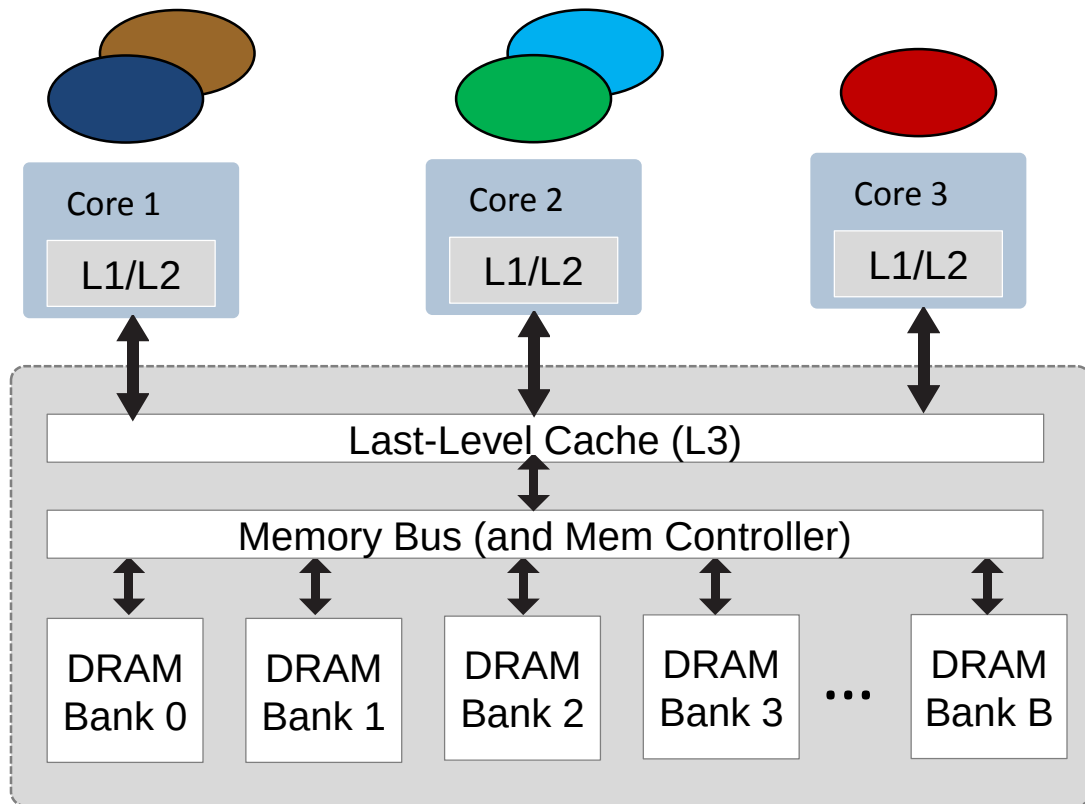


Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

What are the requirements of a good abstraction?



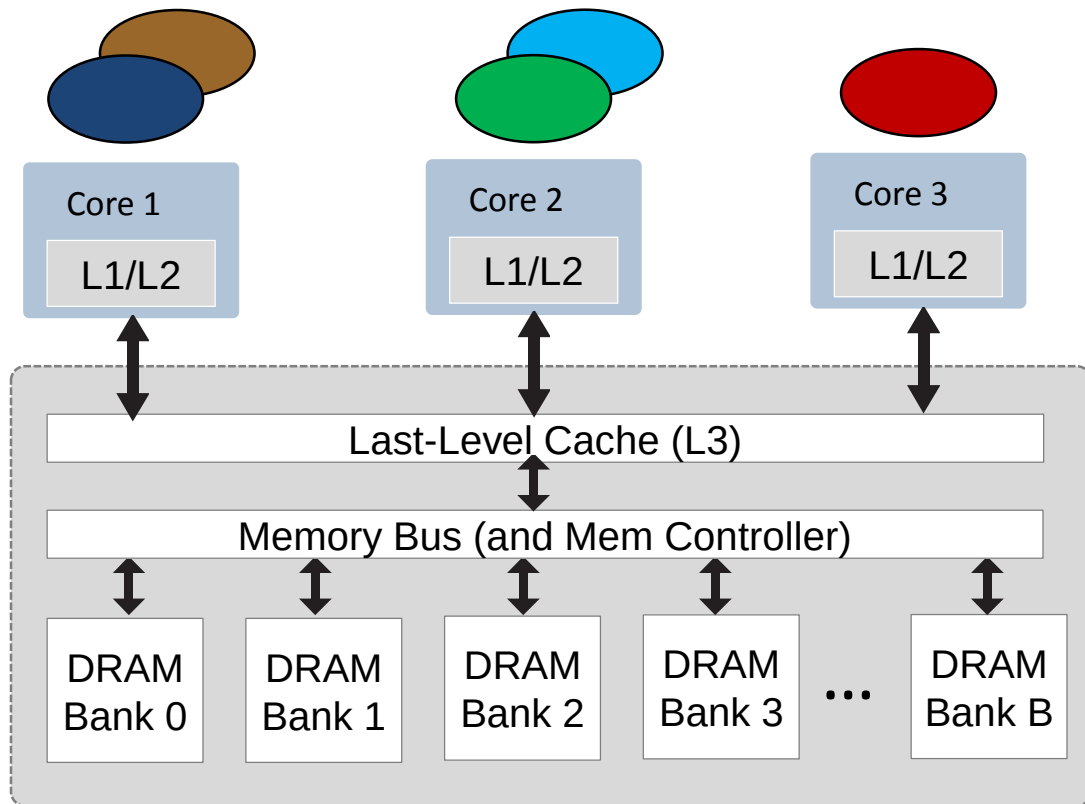
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

What are the requirements of a good abstraction?

1. It should be as small as possible (few numbers; few bits)



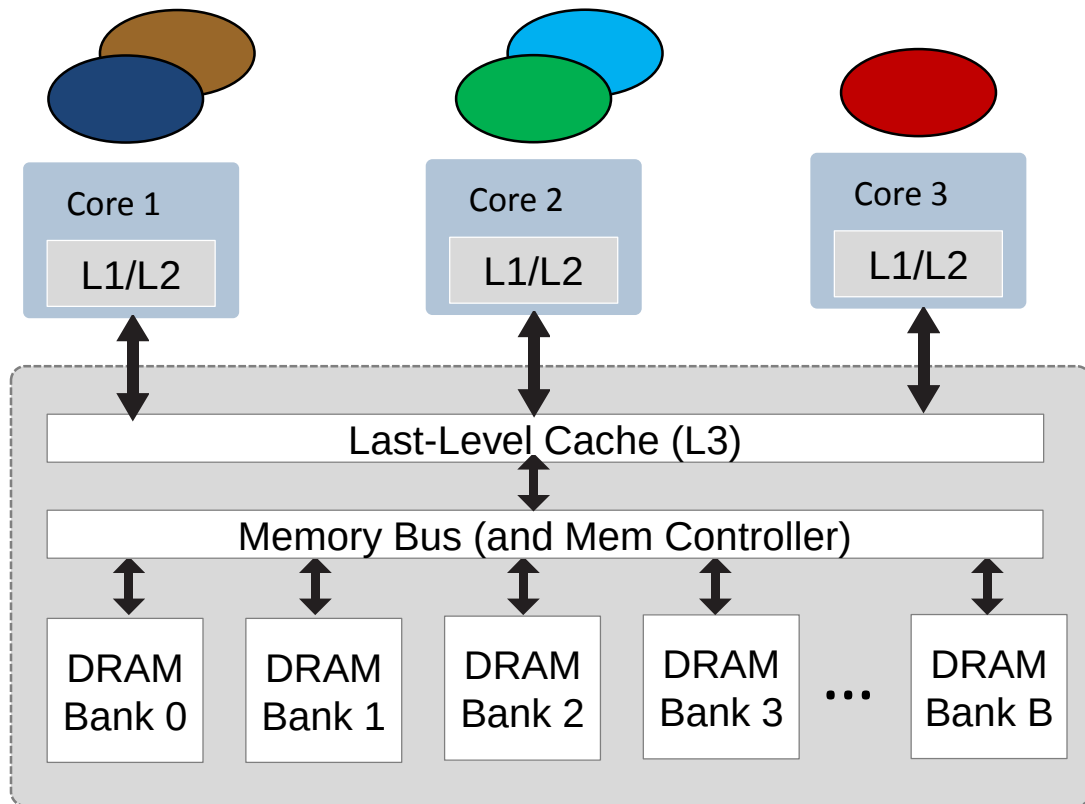
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

What are the requirements of a good abstraction?

1. It should be as small as possible (few numbers; few bits)
2. It should allow us to do prediction/analysis that we care about



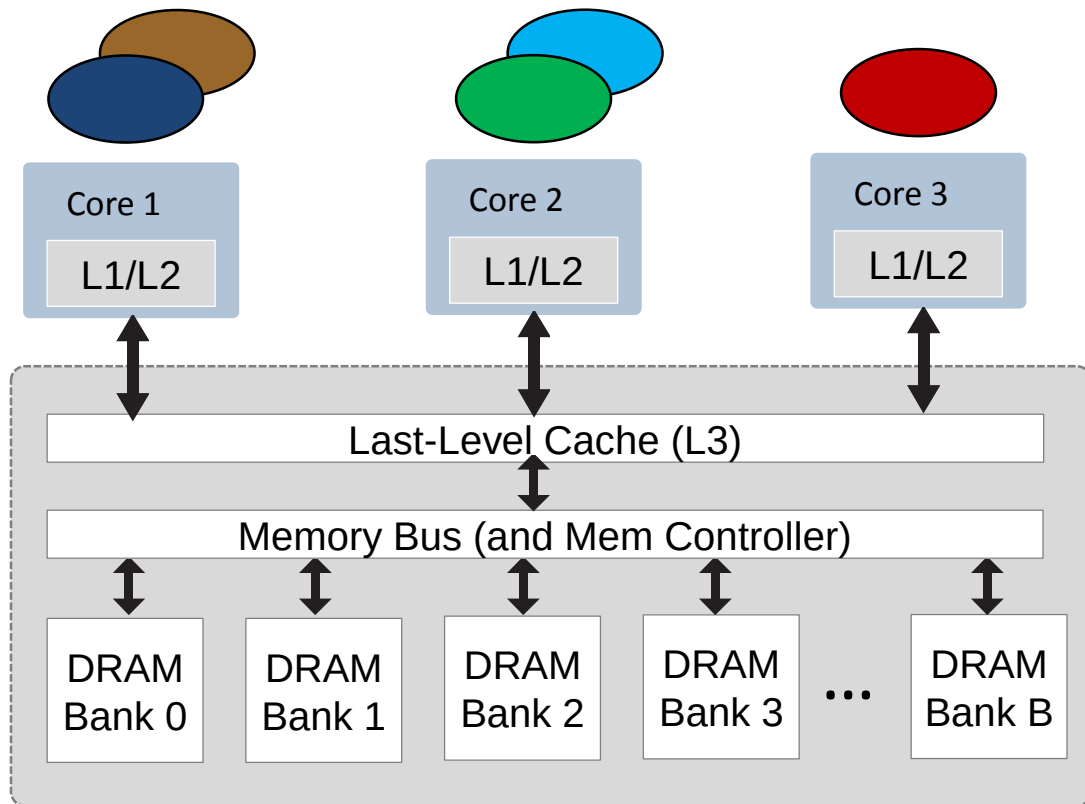
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

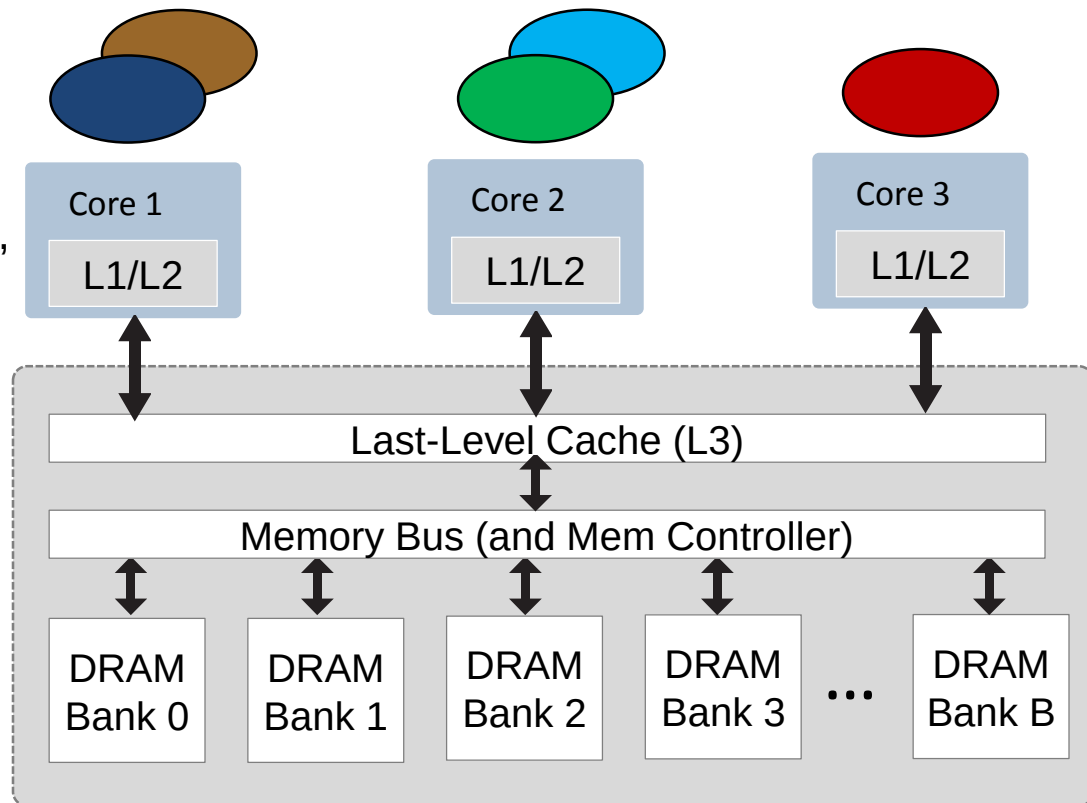
What are the requirements of a good abstraction?

1. It should be as small as possible (few numbers; few bits)
2. It should allow us to do prediction/analysis that we care about
3. Given a system, it should be possible to find the abstraction (through measurements or lower-level analysis)



Considerations in creating an abstraction for multicore

Let $speed(i,t)$ denote the speed of execution of task i at time t . Let $cor(i,t)$ denote the set of tasks, other than task i , that executes at time t .

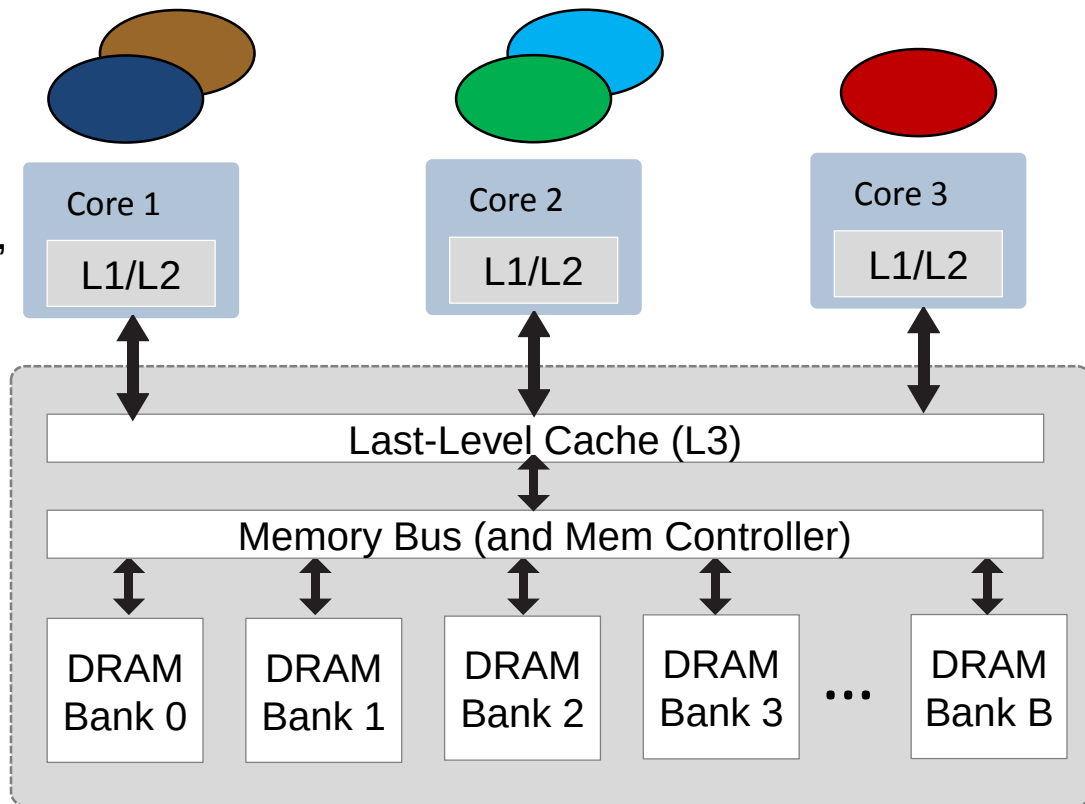


Considerations in creating an abstraction for multicore

Let $speed(i, t)$ denote the speed of execution of task i at time t . Let $cor(i, t)$ denote the set of tasks, other than task i , that executes at time t .

One abstraction could be:

$$\frac{1}{1 + |cor(i, t)|} \leq speed(i, t) \leq 1$$



Considerations in creating an abstraction for multicore

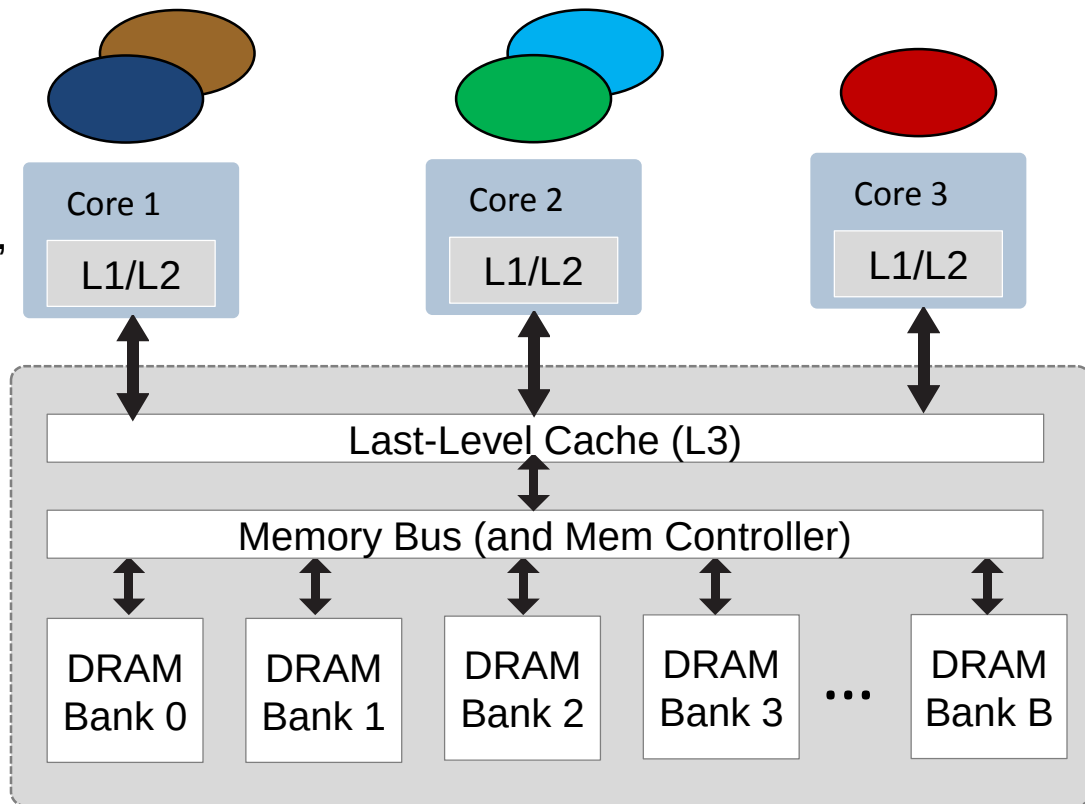
Let $speed(i, t)$ denote the speed of execution of task i at time t . Let $cor(i, t)$ denote the set of tasks, other than task i , that executes at time t .

One abstraction could be:

$$\frac{1}{1 + |cor(i, t)|} \leq speed(i, t) \leq 1$$

Drawback of abstraction:

- Does not reflect that different co-runners can have different effect on task i .



Considerations in creating an abstraction for multicore

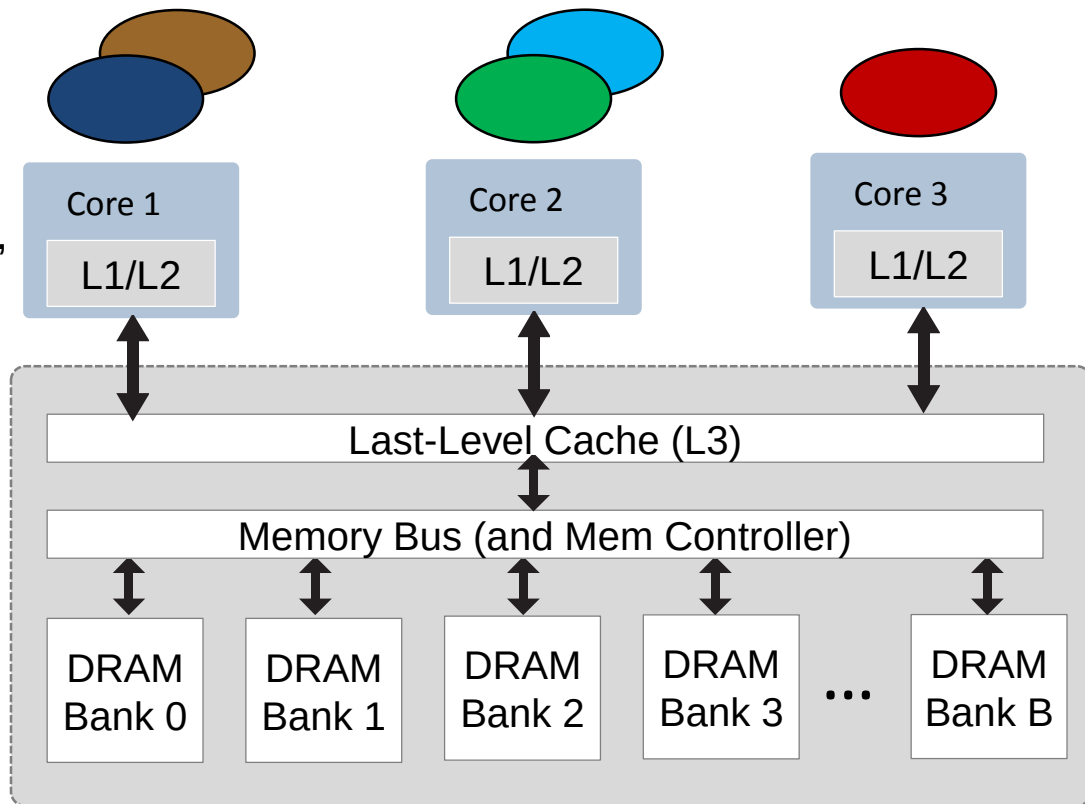
Let $speed(i, t)$ denote the speed of execution of task i at time t . Let $cor(i, t)$ denote the set of tasks, other than task i , that executes at time t .

One abstraction could be:

$$1 - k \times |cor(i, t)| \leq speed(i, t) \leq 1$$

Drawback of abstraction:

- Does not reflect that different co-runners can have different effect on task i .



Considerations in creating an abstraction for multicore

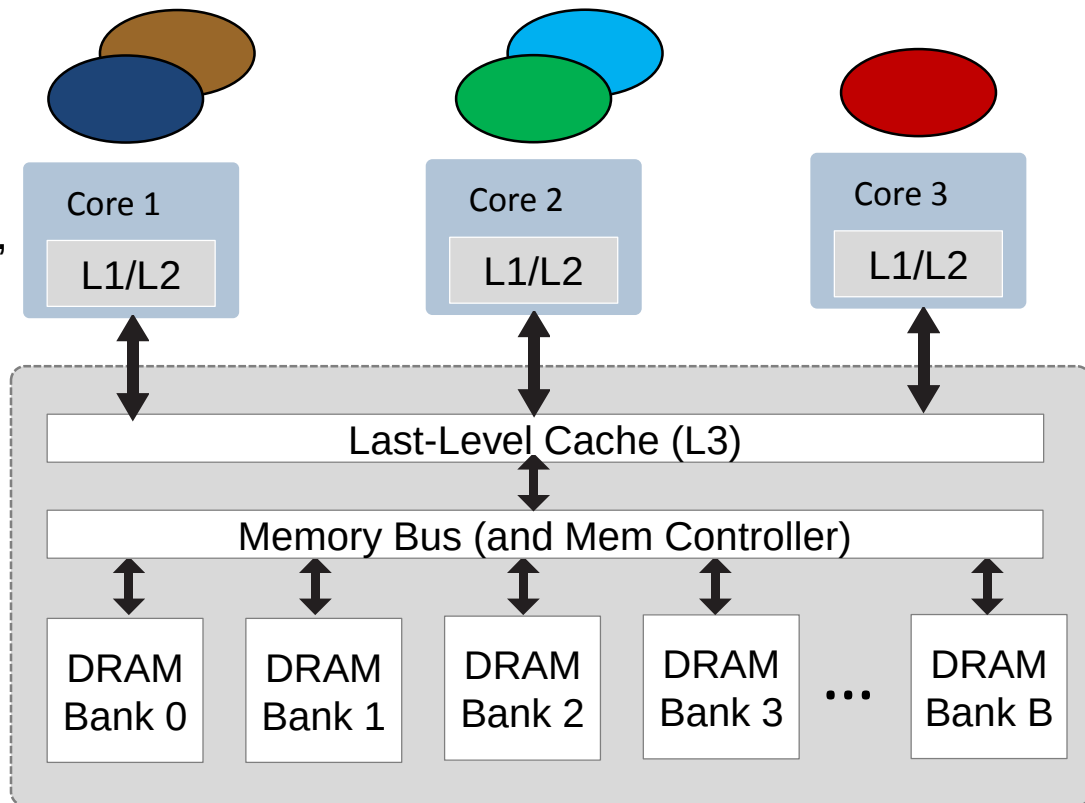
Let $speed(i, t)$ denote the speed of execution of task i at time t . Let $cor(i, t)$ denote the set of tasks, other than task i , that executes at time t .

One abstraction could be:

$$\frac{w_i}{1 + \sum_{j|j \in cor(i, t)} w_{i, j}} \leq speed(i, t) \leq 1$$

Drawback of abstraction:

- In reality, the speed can be super-additive or sub-additive (not reflected in the above).



Considerations in creating an abstraction for multicore

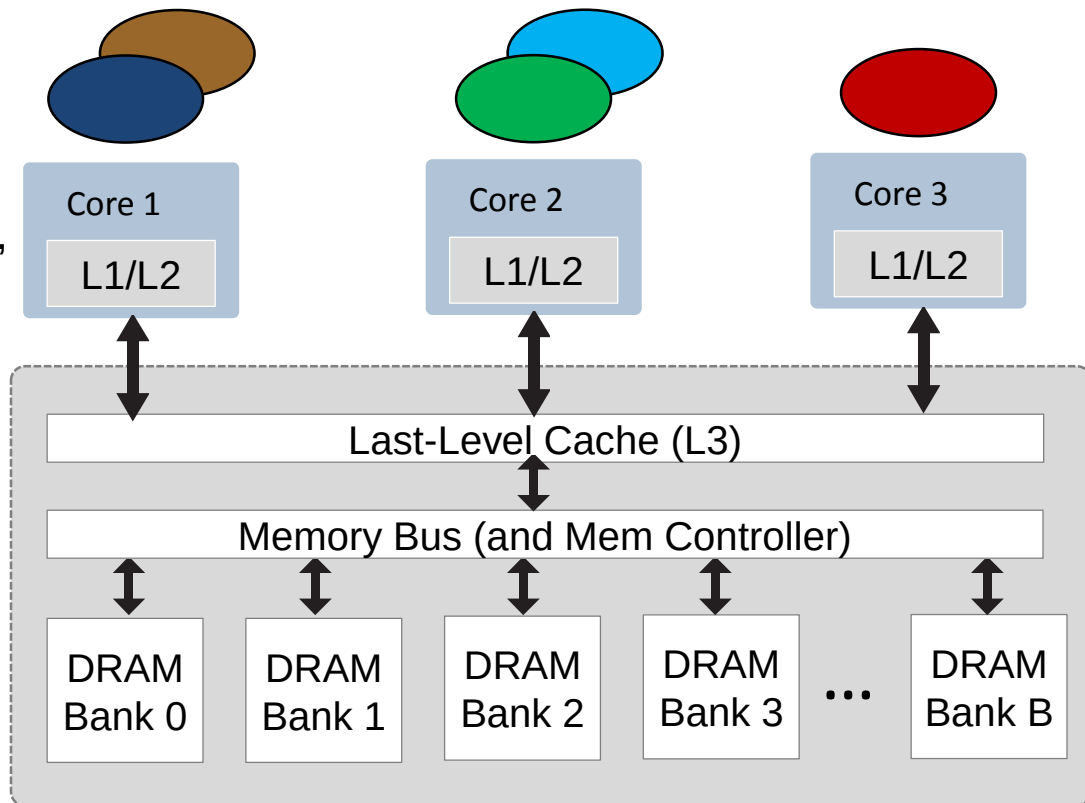
Let $speed(i, t)$ denote the speed of execution of task i at time t . Let $cor(i, t)$ denote the set of tasks, other than task i , that executes at time t .

One abstraction could be:

$$\frac{w_i}{1 + \prod_{j|j \in cor(i, t)} w_{i,j}} \leq speed(i, t) \leq 1$$

Drawback of abstraction:

- In reality, the speed can be super-additive or sub-additive (not reflected in the above).



Considerations in creating an abstraction for multicore

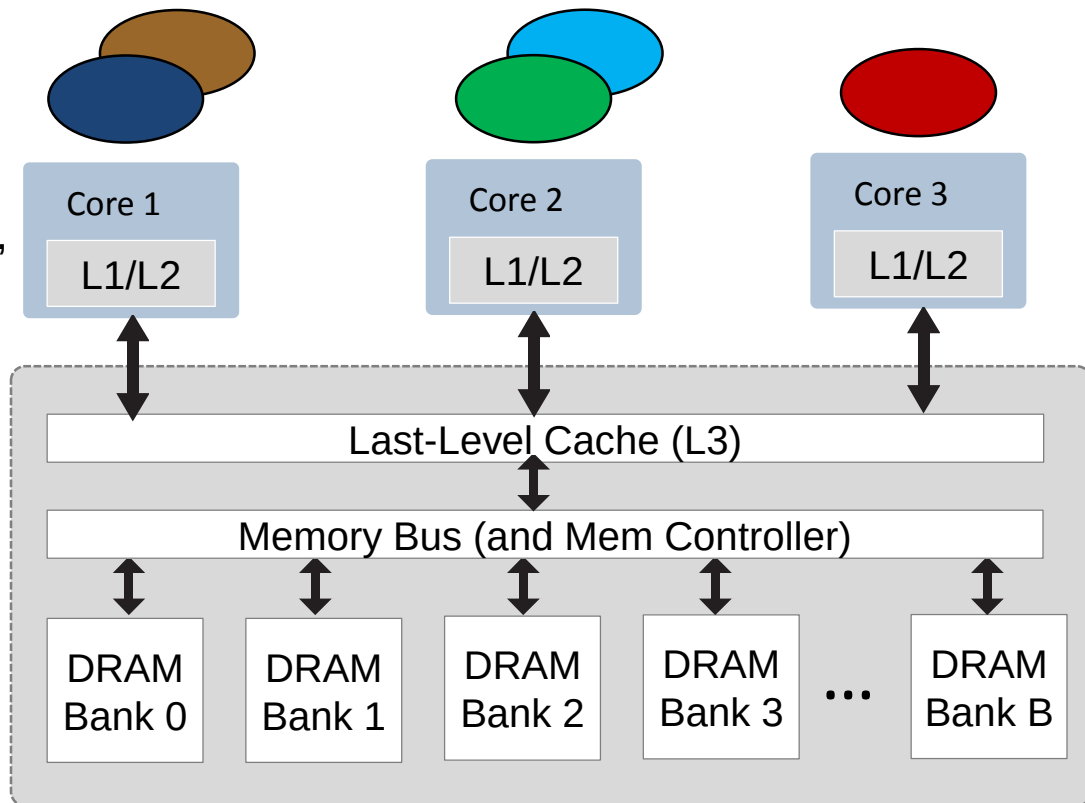
Let $speed(i, t)$ denote the speed of execution of task i at time t . Let $cor(i, t)$ denote the set of tasks, other than task i , that executes at time t .

One abstraction could be:

$$\frac{w_i}{1 + e^{\sum_{j|j \in cor(i, t)} w_{i, j}}} \leq speed(i, t) \leq 1$$

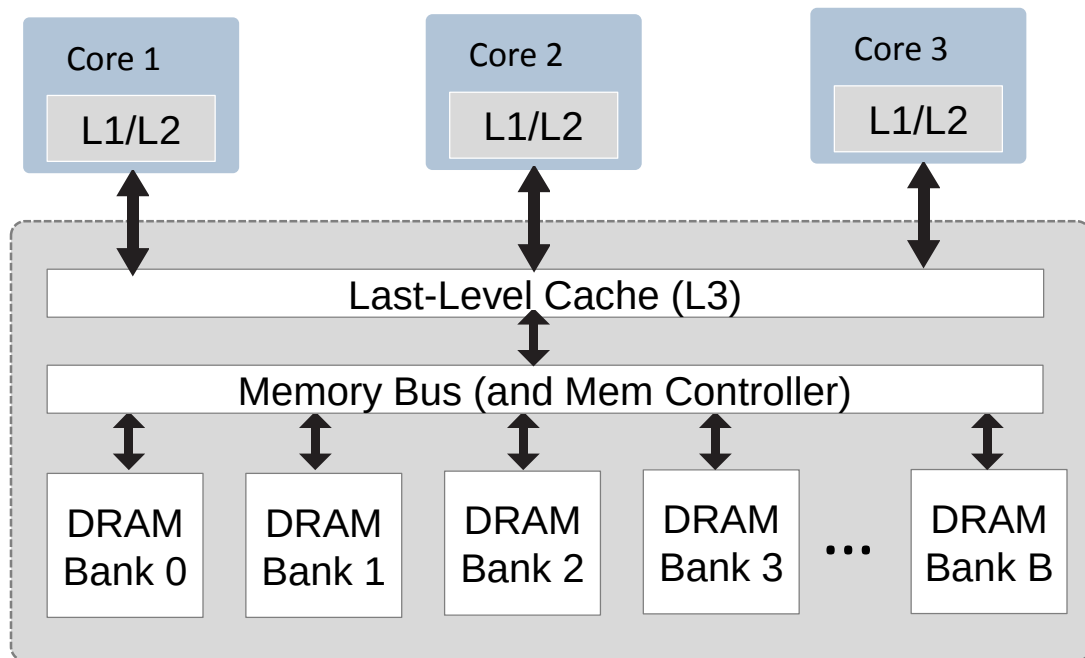
Drawback of abstraction:

- In reality, the speed can be super-additive or sub-additive (not reflected in the above).



Considerations in creating an abstraction for multicore

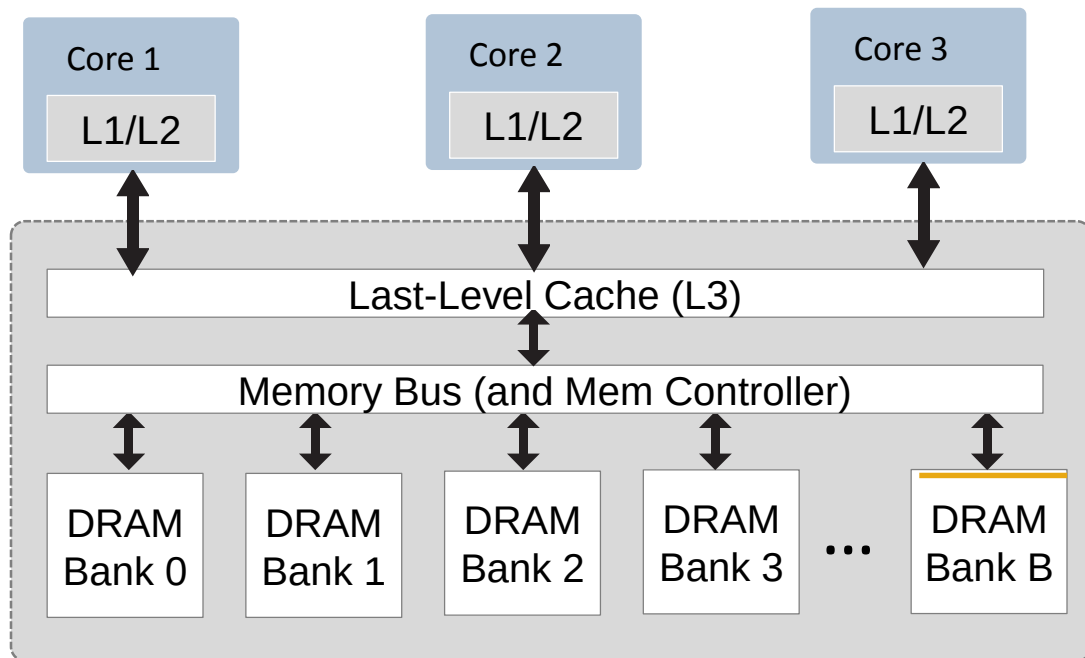
Why is speed sub-additive?



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Consider DRAM bank B and its row buffer.



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

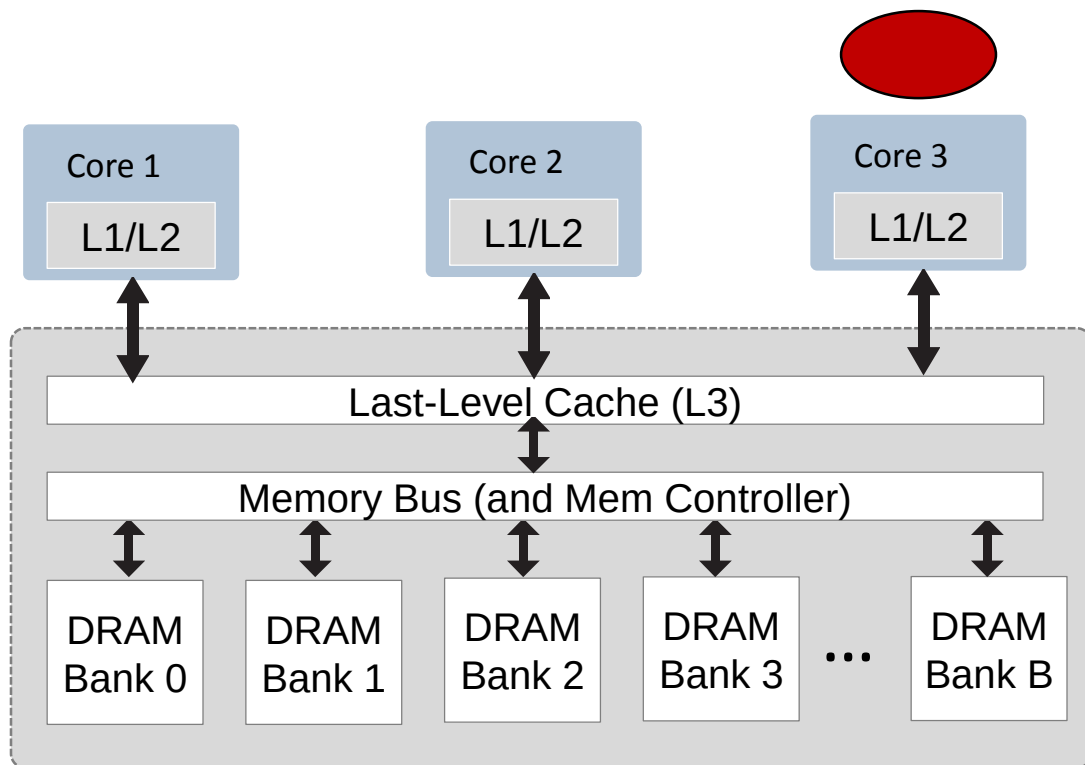
Example:

Red task (victim):

1. load address x to register y
2. load address x' to register z

x and x' are in the same row in
DRAM bank B'

But x and x' are in different columns
(different addresses)



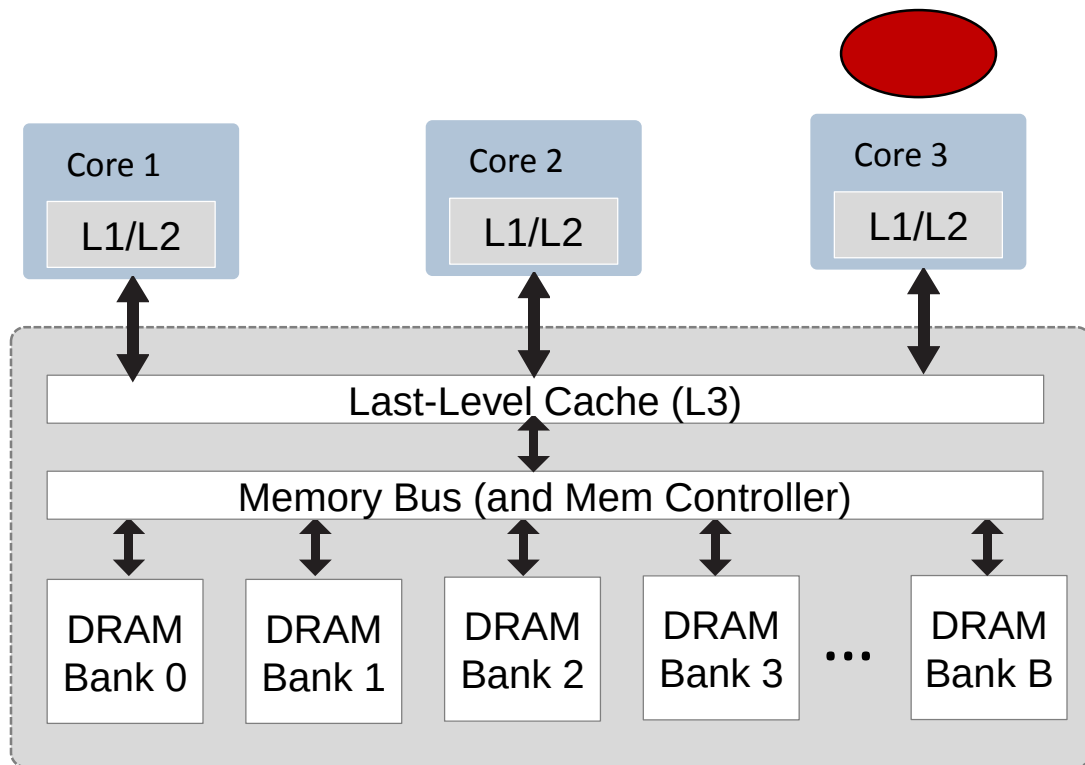
Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

1. load address x to register y
2. load address x' to register z



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

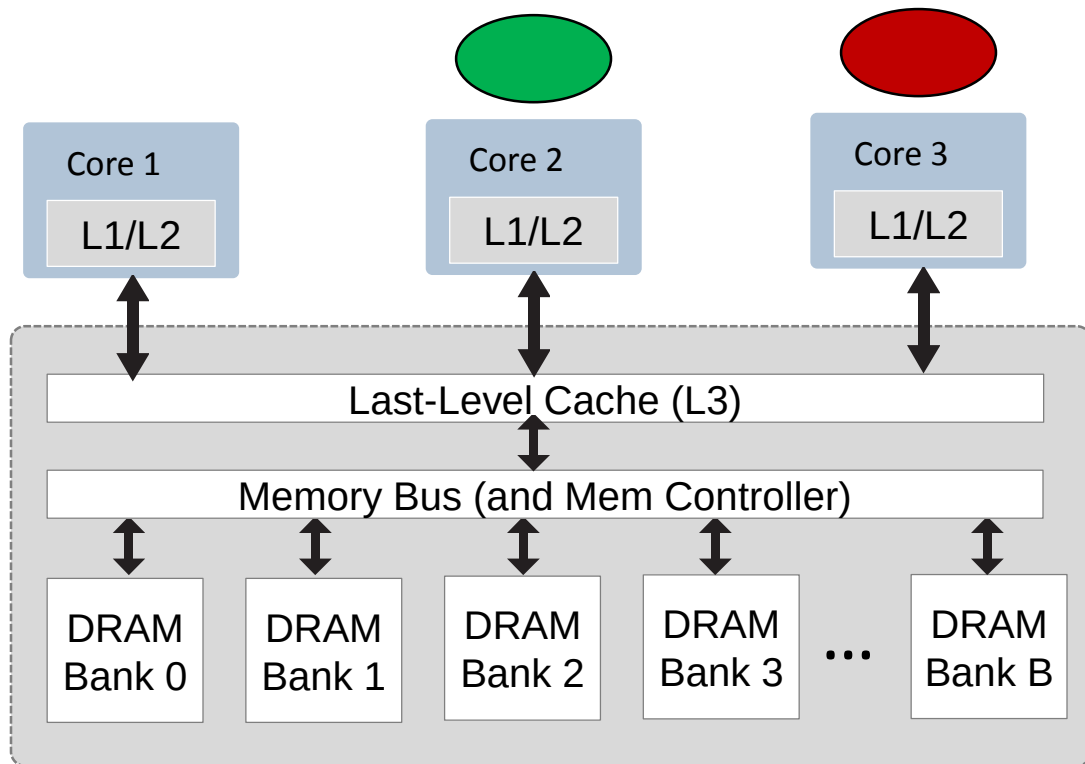
Example:

Red task:

1. load address x to register y
2. load address x' to register z

Green task:

1. load address a (where a is in same bank as x but in different row)



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

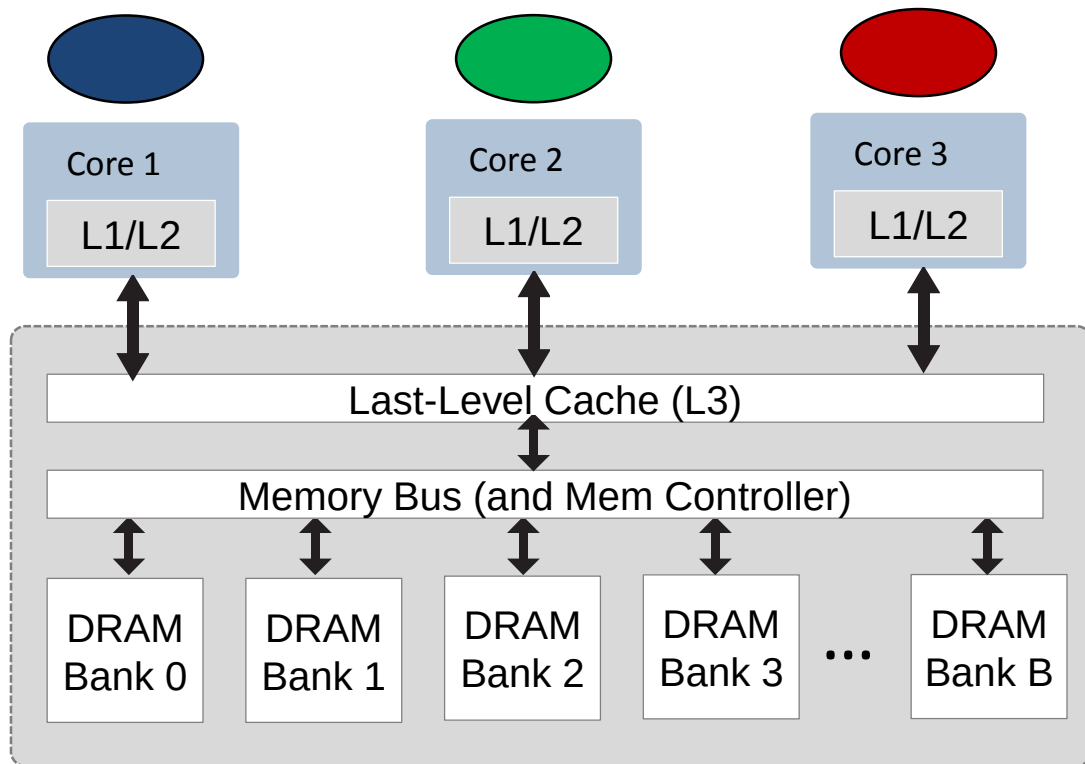
1. load address x to register y
2. load address x' to register z

Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

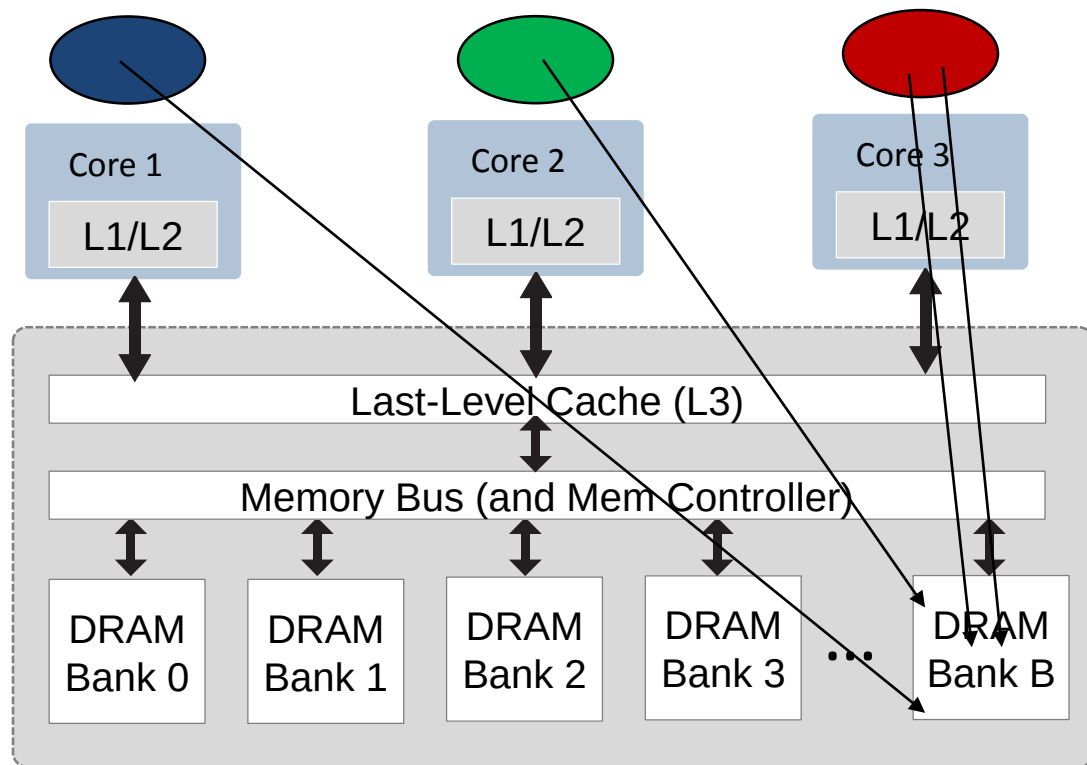
1. load address x to register y
2. load address x' to register z

Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

1. load address x to register y
2. load address x' to register z

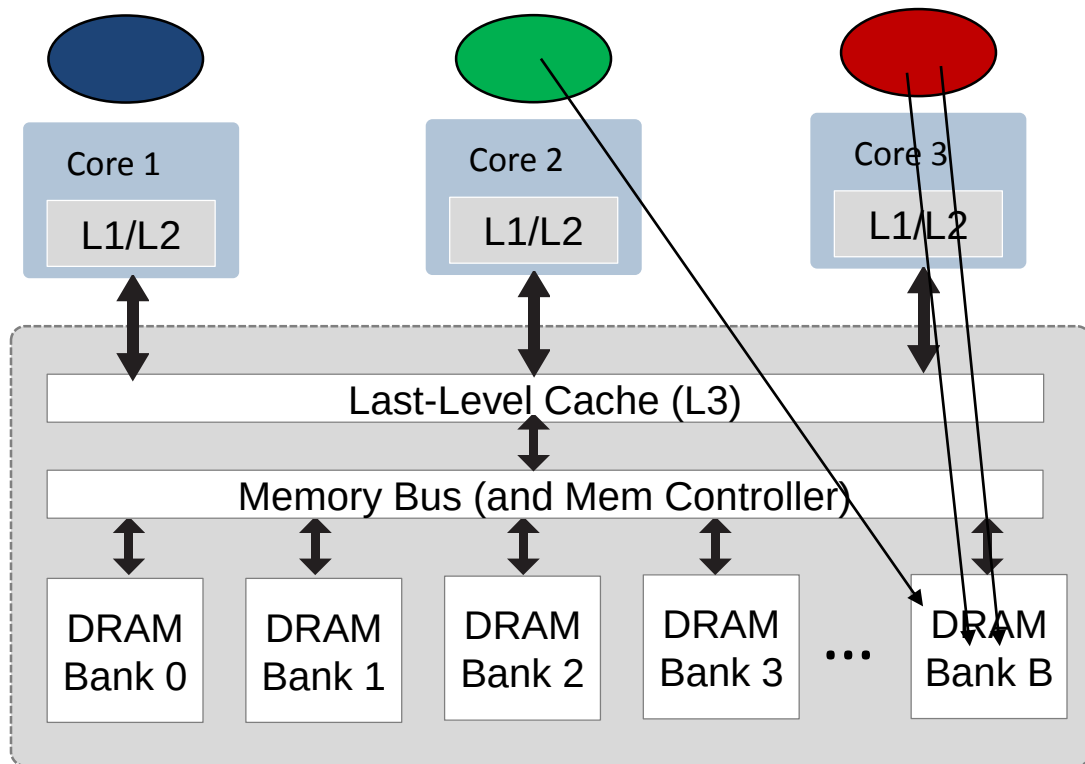
Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)

Observation: Green can evict Red's data in the row buffer.



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

1. load address x to register y
2. load address x' to register z

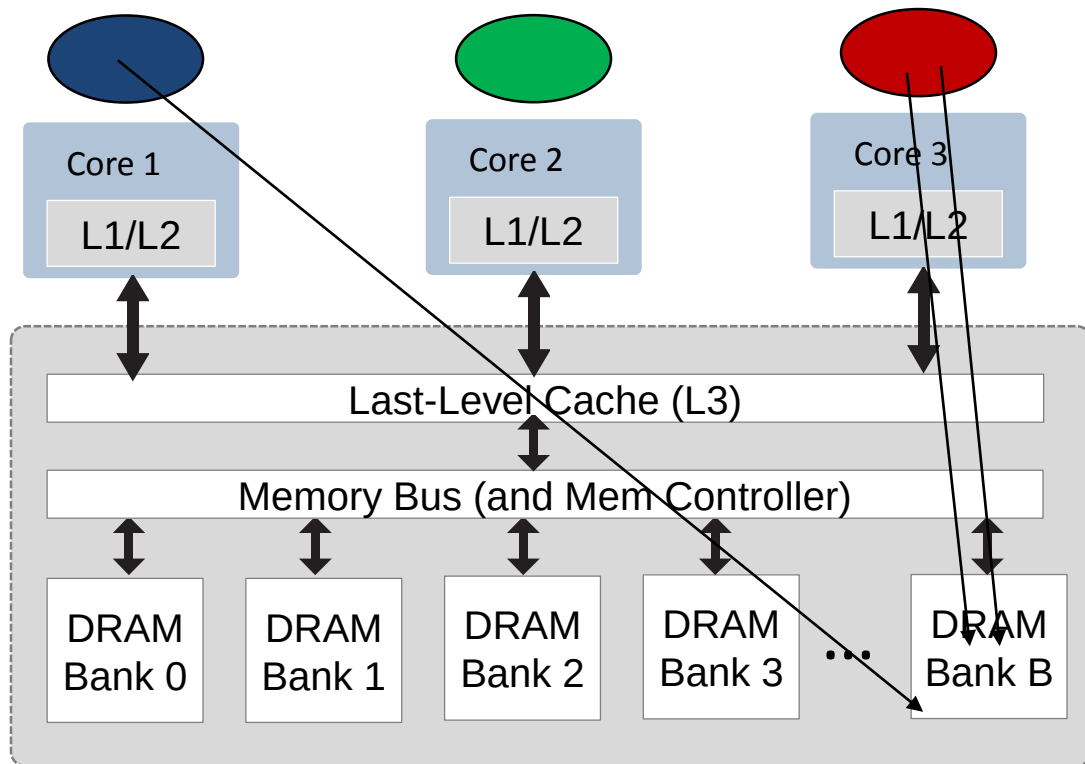
Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)

Observation: Blue can evict Red's data in the row buffer.



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

1. load address x to register y
2. load address x' to register z

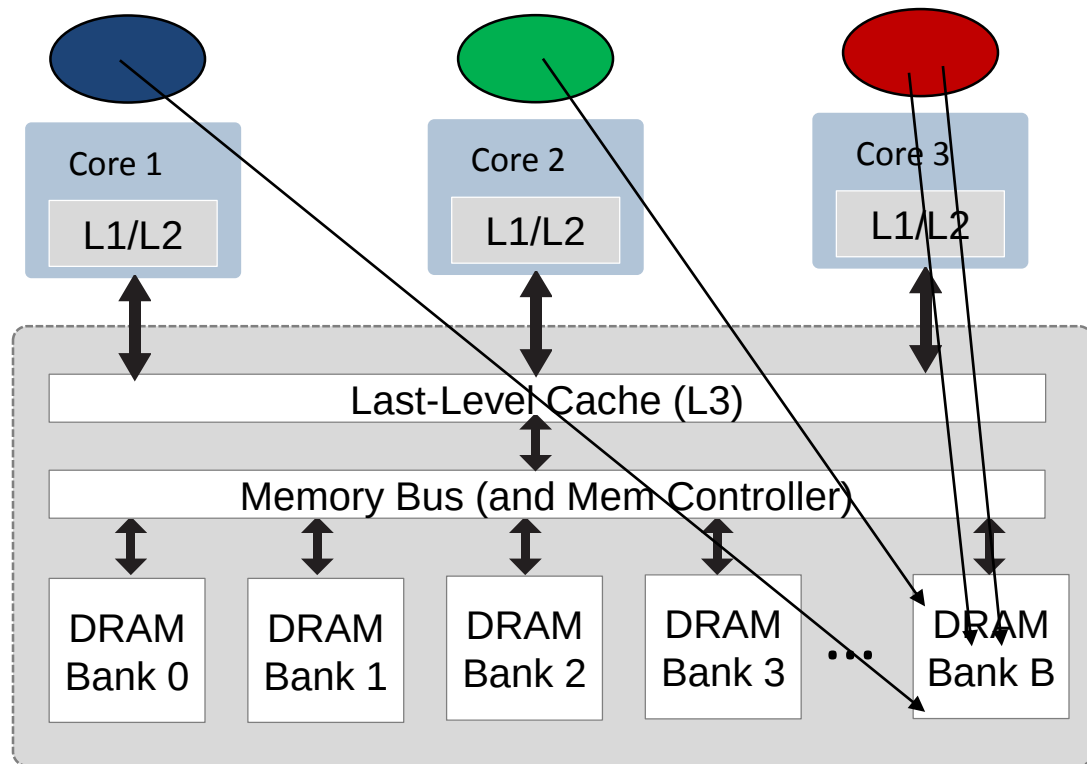
Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)

Observation: If Blue has evicted Red's data in the row buffer, then Green cannot evict it more.



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

1. load address x to register y
2. load address x' to register z

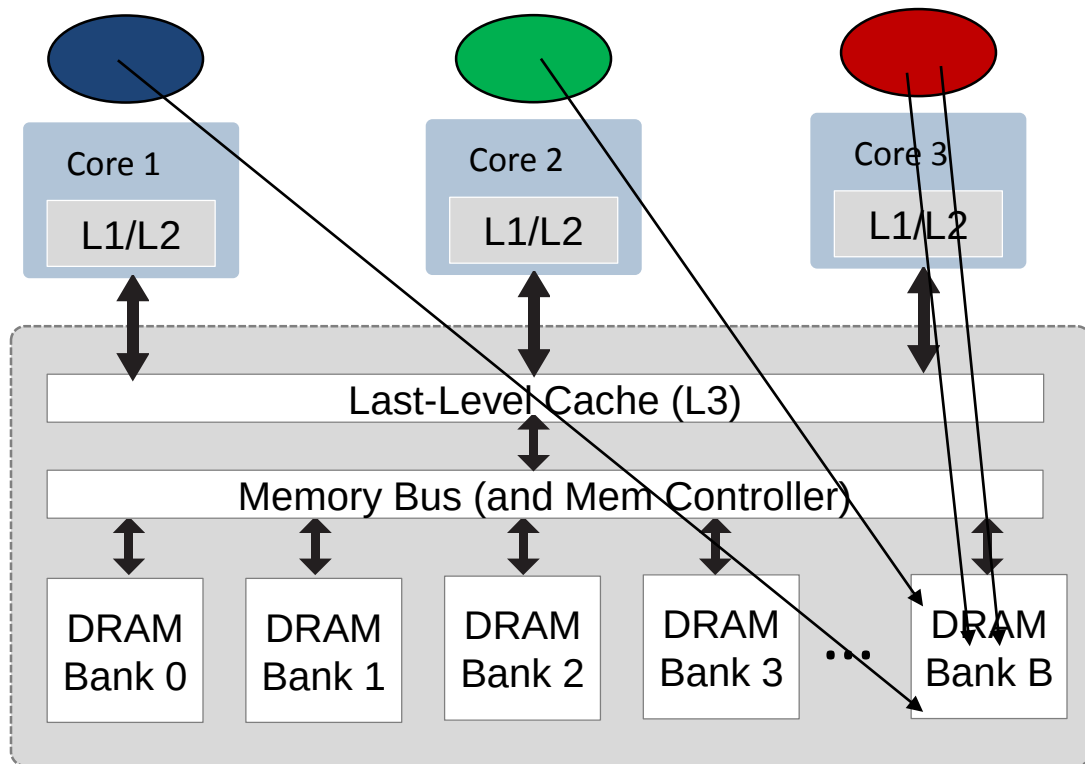
Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)

Observation: If Green has evicted Red's data in the row buffer, then Blue cannot evict it more.



Considerations in creating an abstraction for multicore

Why is speed sub-additive?

Example:

Red task:

1. load address x to register y
2. load address x' to register z

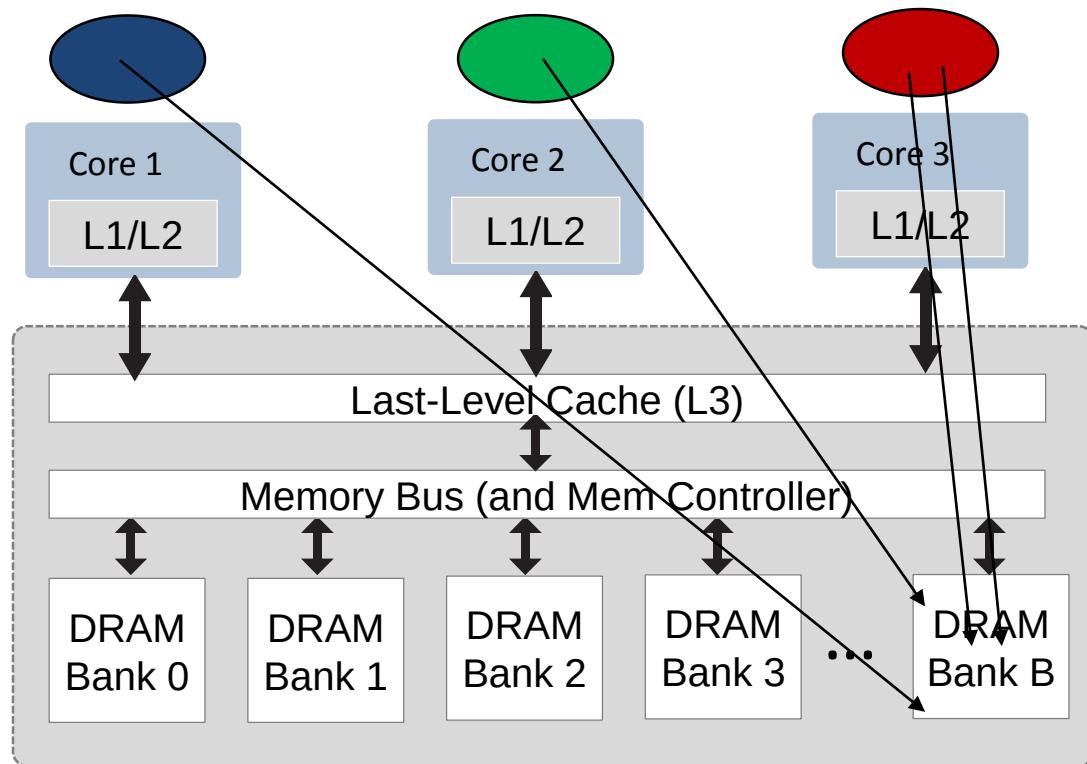
Green task:

1. load address a (where a is in same bank as x but in different row)

Blue task:

1. load address b (where b is in same bank as x but in different row)

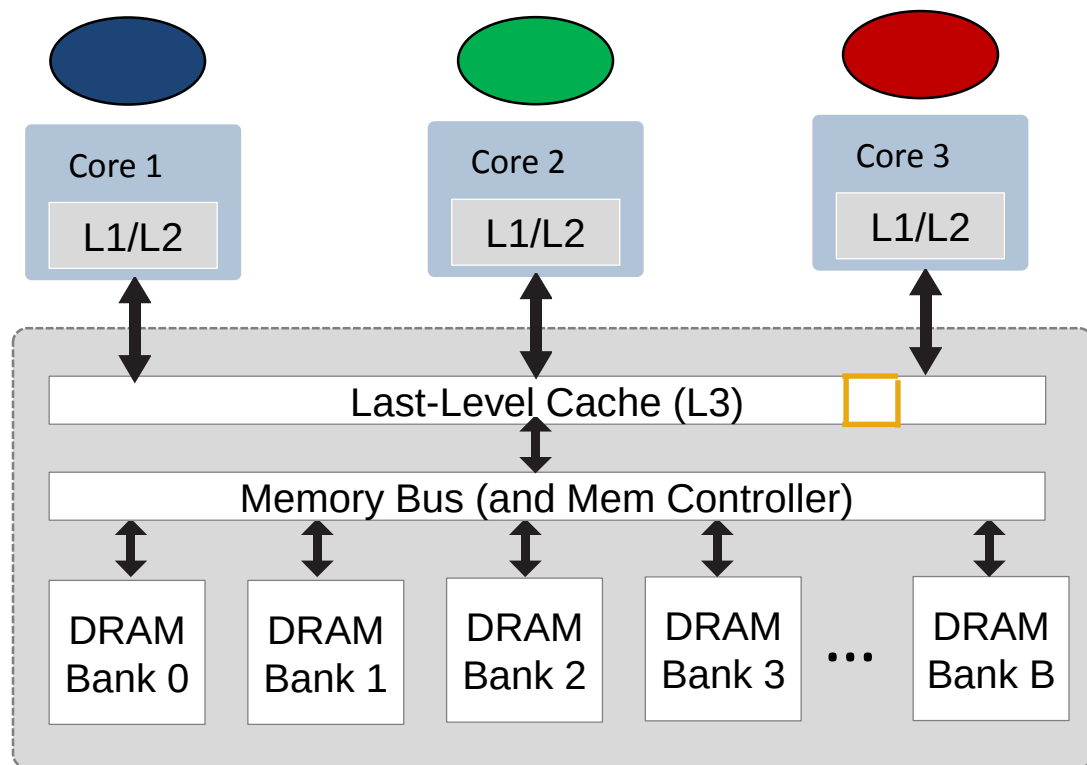
Red experiences a slowdown due to Blue or Green but the slowdown is not greater by both.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Consider Last-Level Cache (L3).
Assume associativity = 2.
Consider one specific cache set.



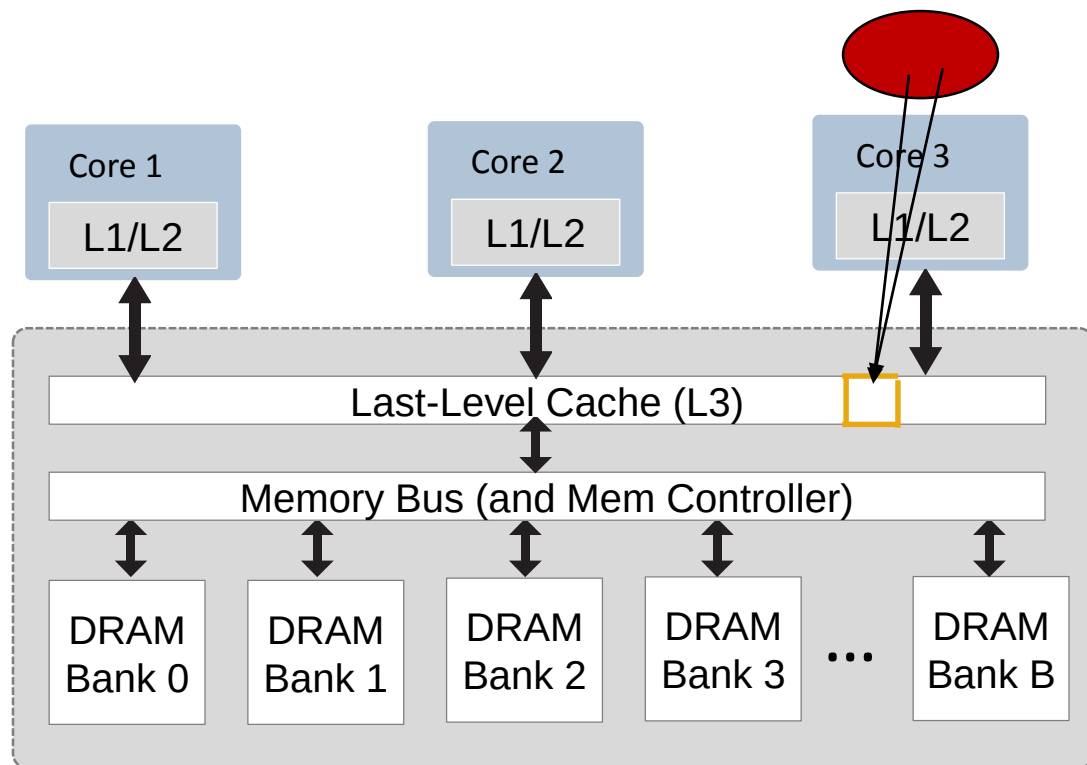
Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z



Considerations in creating an abstraction for multicore

Why is speed super-additive?

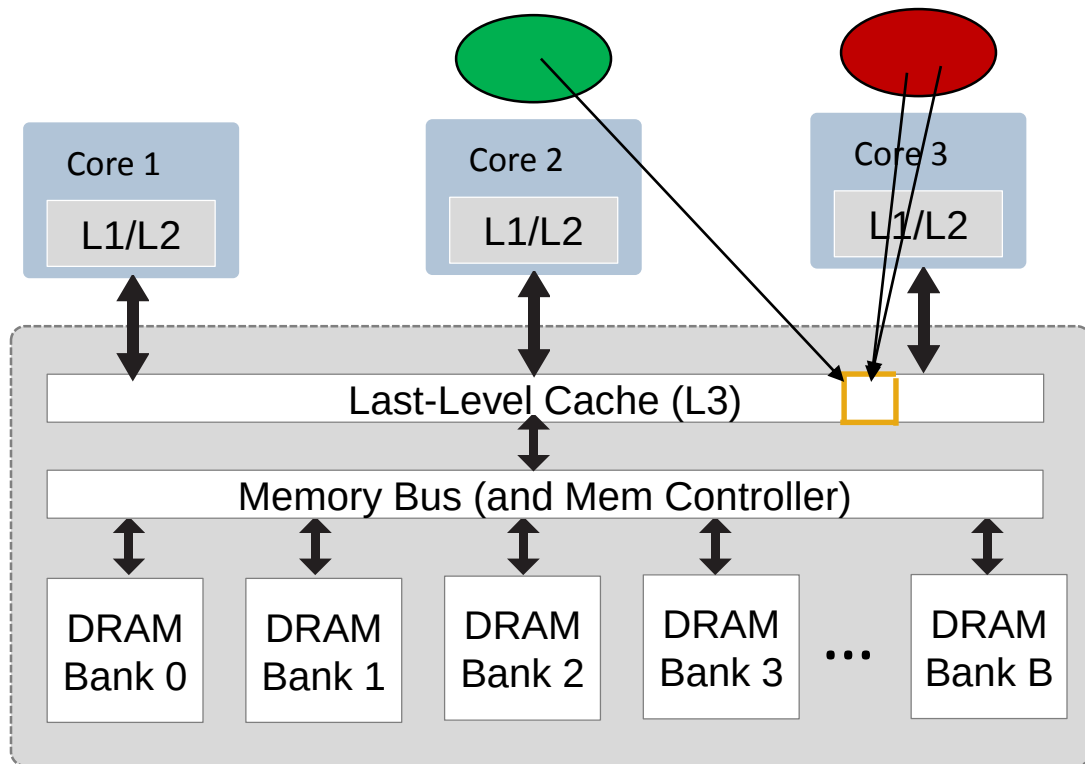
Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

Green task:

1. load address a (where a is in same cache set x)



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

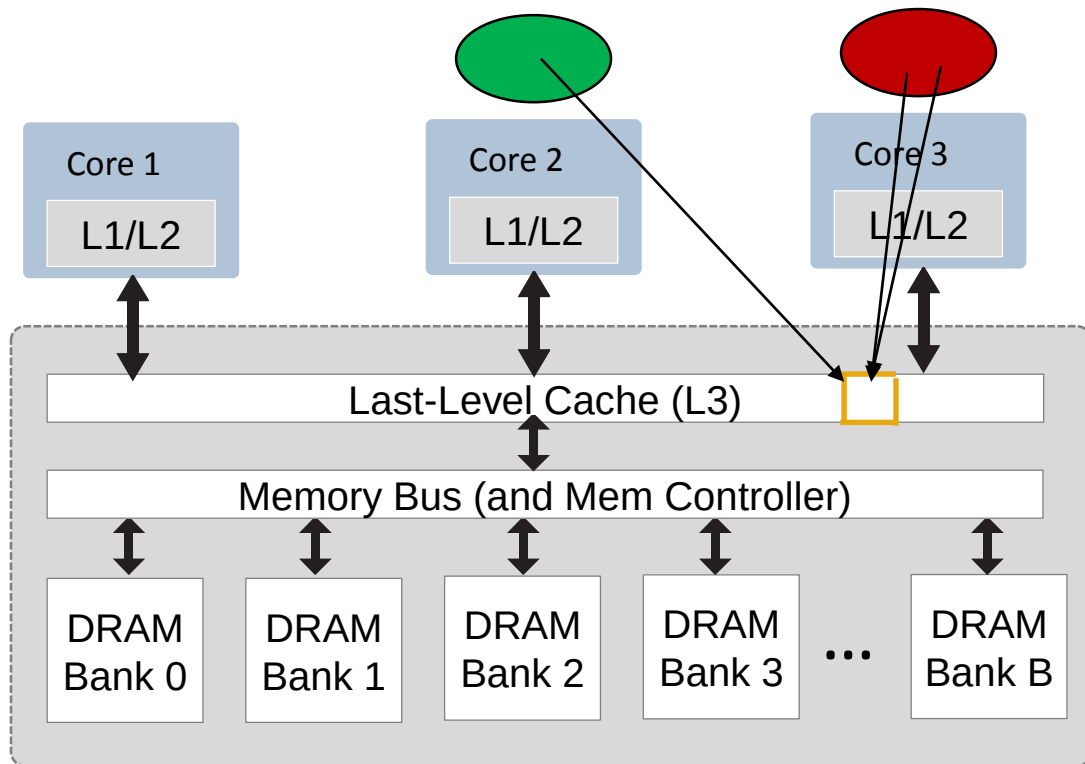
Red task (victim):

1. load address x to register y
2. load address x to register z

Green task:

1. load address a (where a is in same cache set x)

Observation: Green's data and Red's data fit in the cache. No capacity miss.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

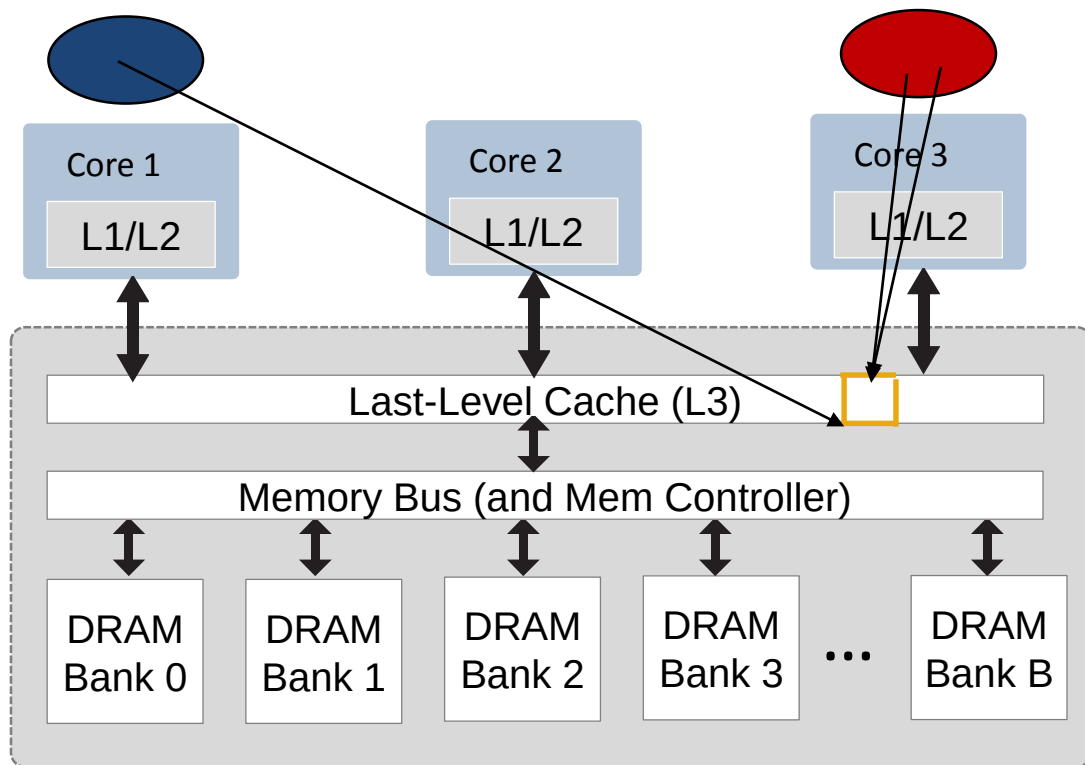
Red task (victim):

1. load address x to register y
2. load address x to register z

Blue task:

1. load address b (where b is in same cache set x)

Observation: Blue's data and Red's data fit in the cache. No capacity miss.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

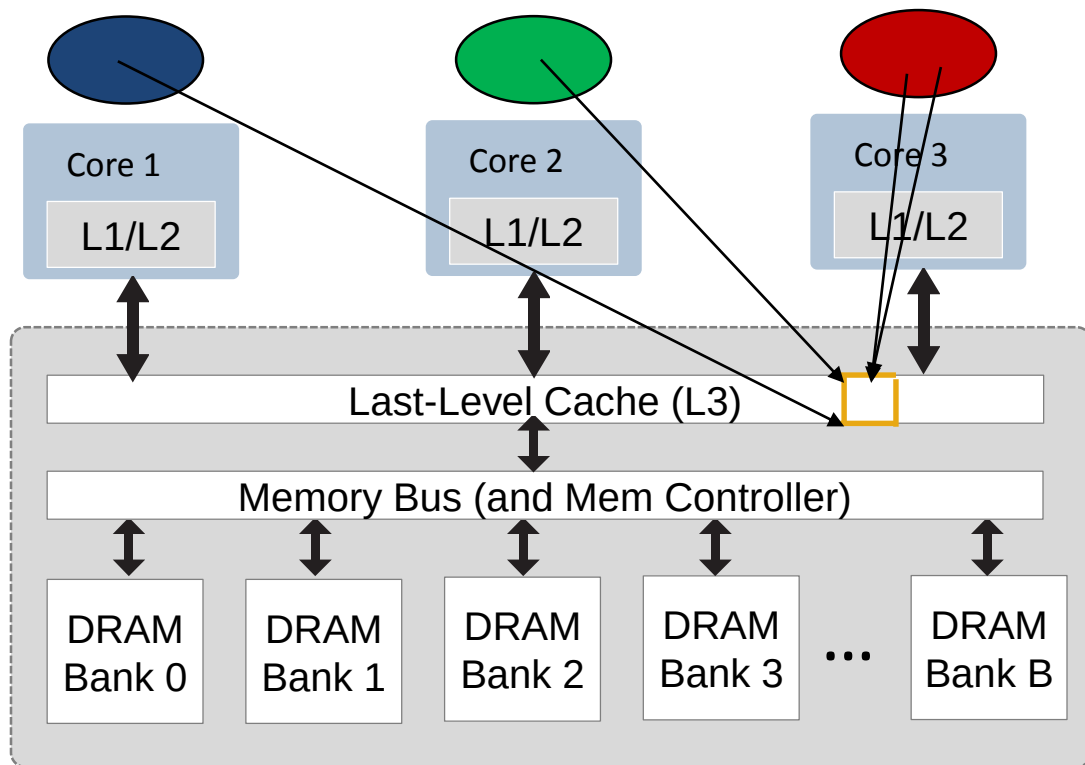
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

Observation: Blue's data, Green's data, and Red's data do not fit in the cache. Capacity miss.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

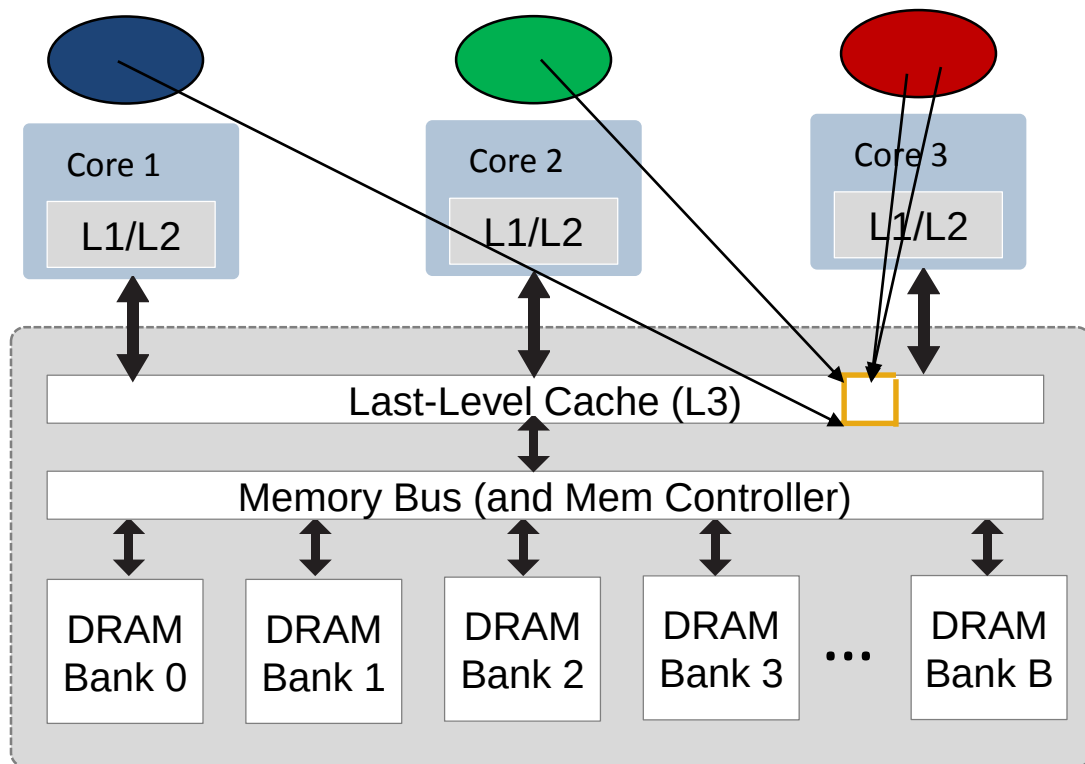
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

Observation: Green alone does not cause slowdown of Red.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

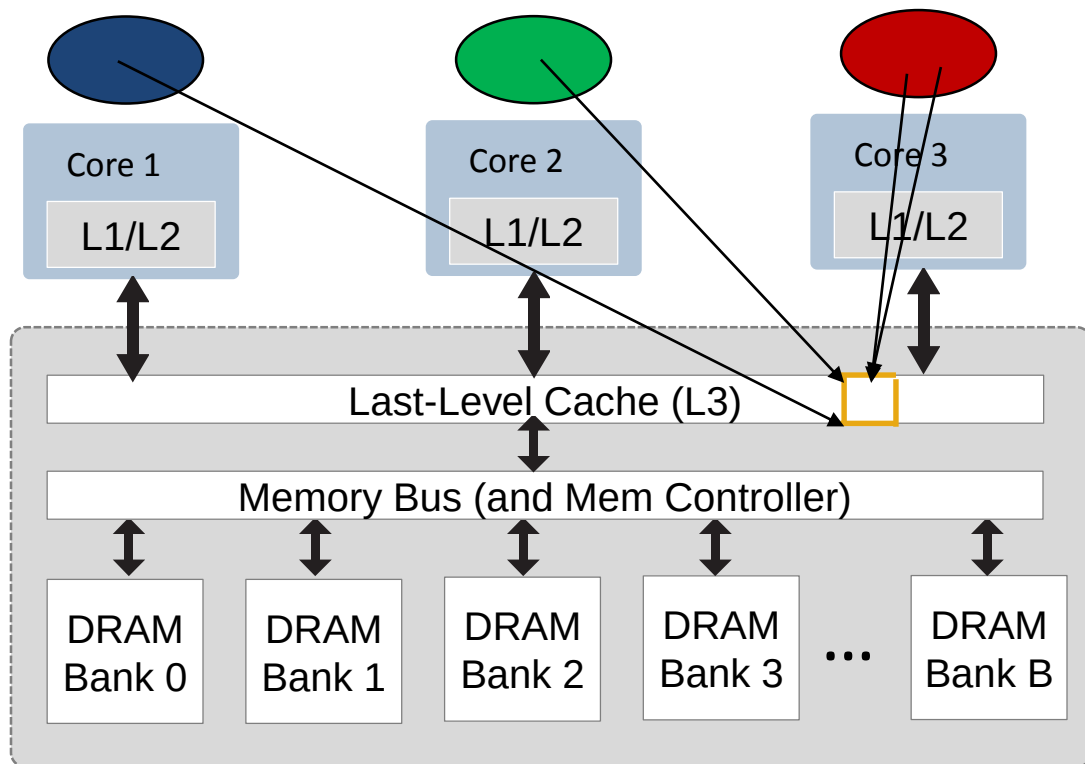
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

Observation: Blue alone does not cause slowdown of Red.



Considerations in creating an abstraction for multicore

Why is speed super-additive?

Example:

Red task (victim):

1. load address x to register y
2. load address x to register z

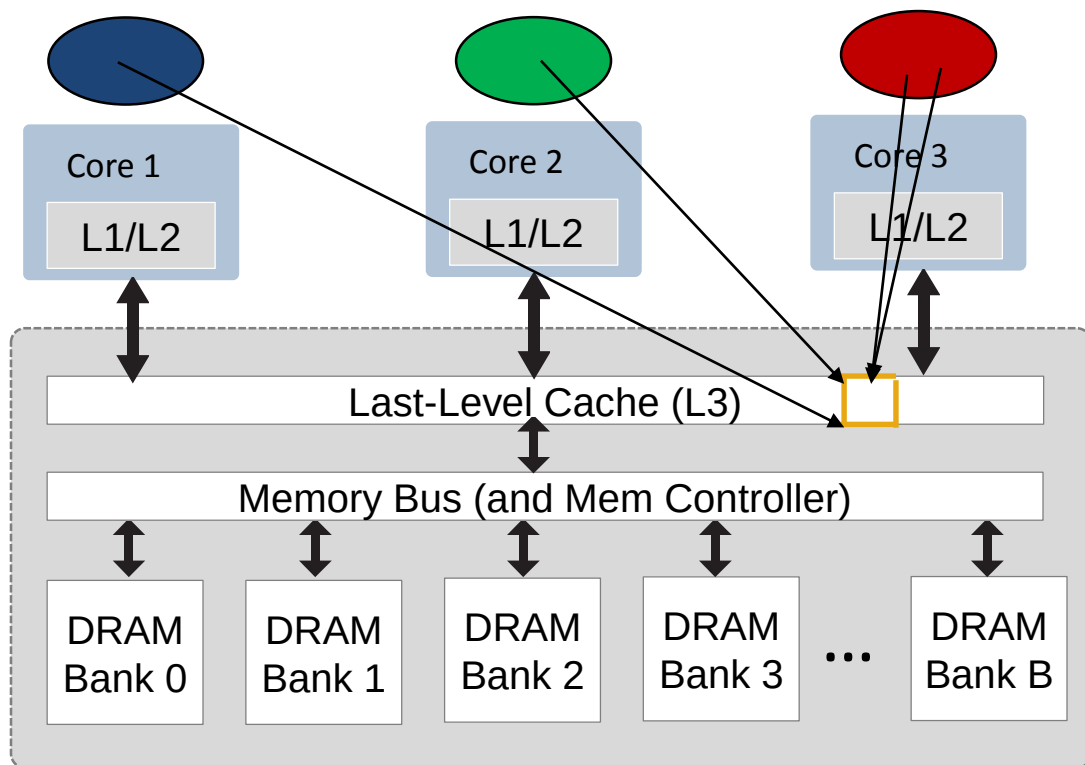
Green task:

1. load address a (where a is in same cache set x)

Blue task:

1. load address b (where b is in same cache set x)

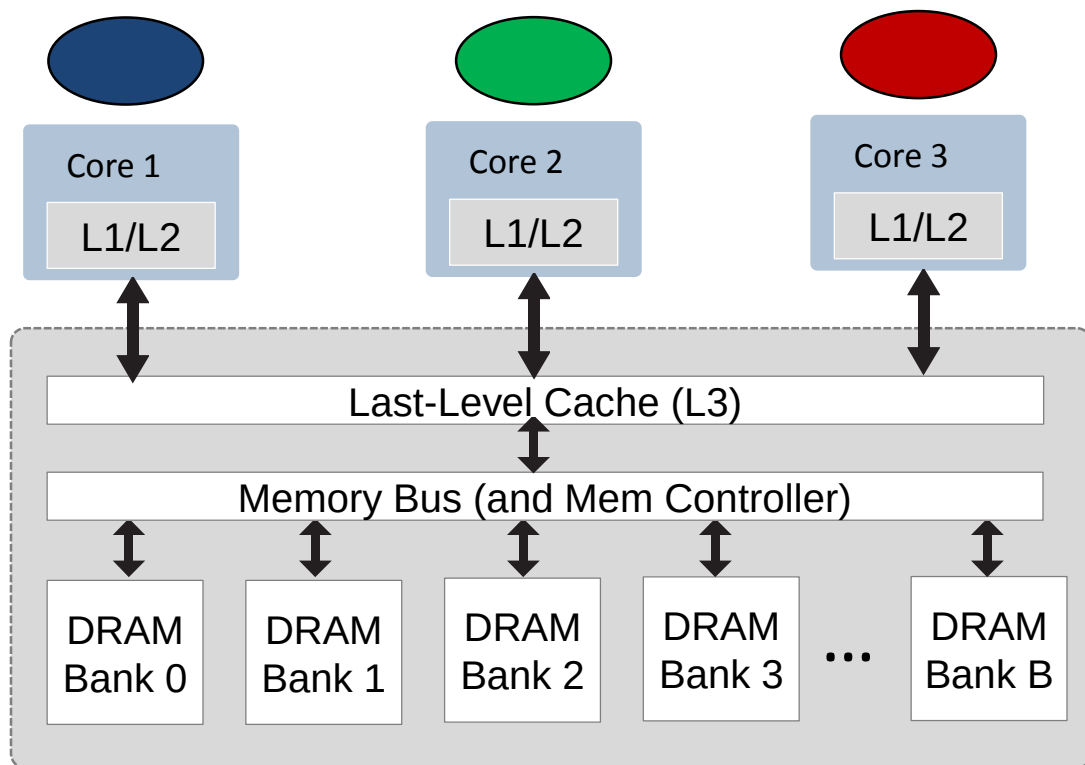
Observation: But Green and Blue cause slowdown of Red.



Considerations in creating an abstraction for multicore

Speed of execution of a task as a function of co-runners can be either

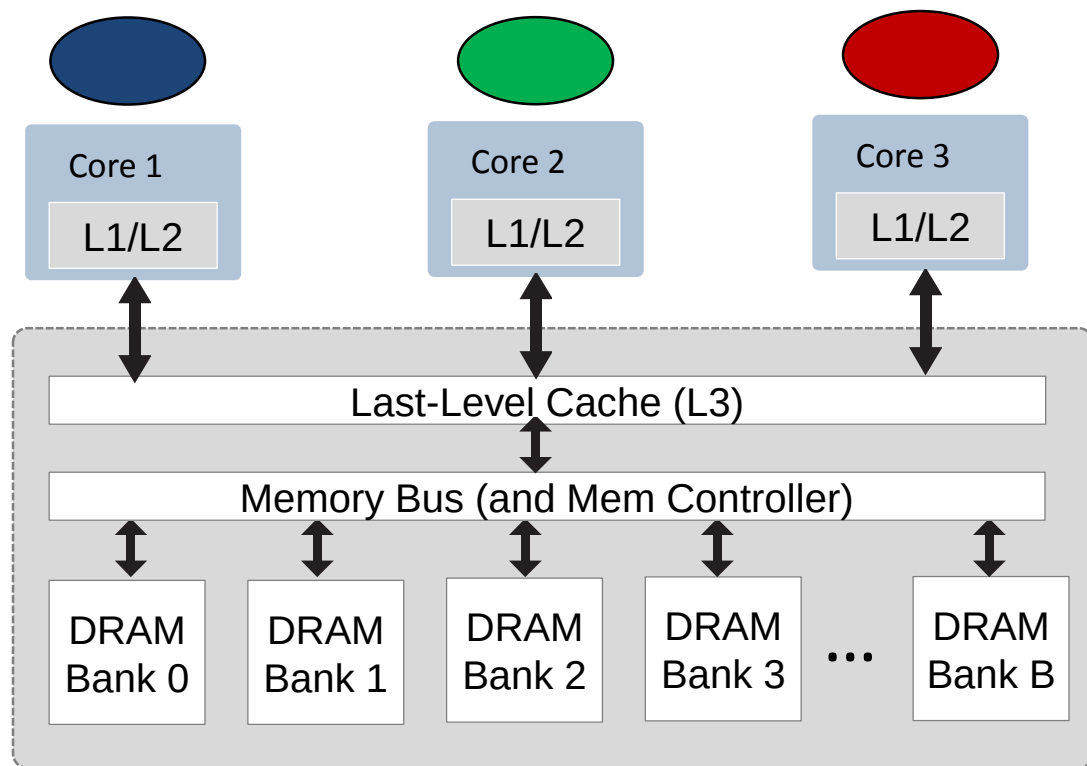
- (i) Additive
- (ii) Sub-additive
- (iii) Super-additive



Considerations in creating an abstraction for multicore

Speed of execution of a task as a function of co-runners can be either

- (i) Additive
- (ii) Sub-additive
- (iii) Super-additive

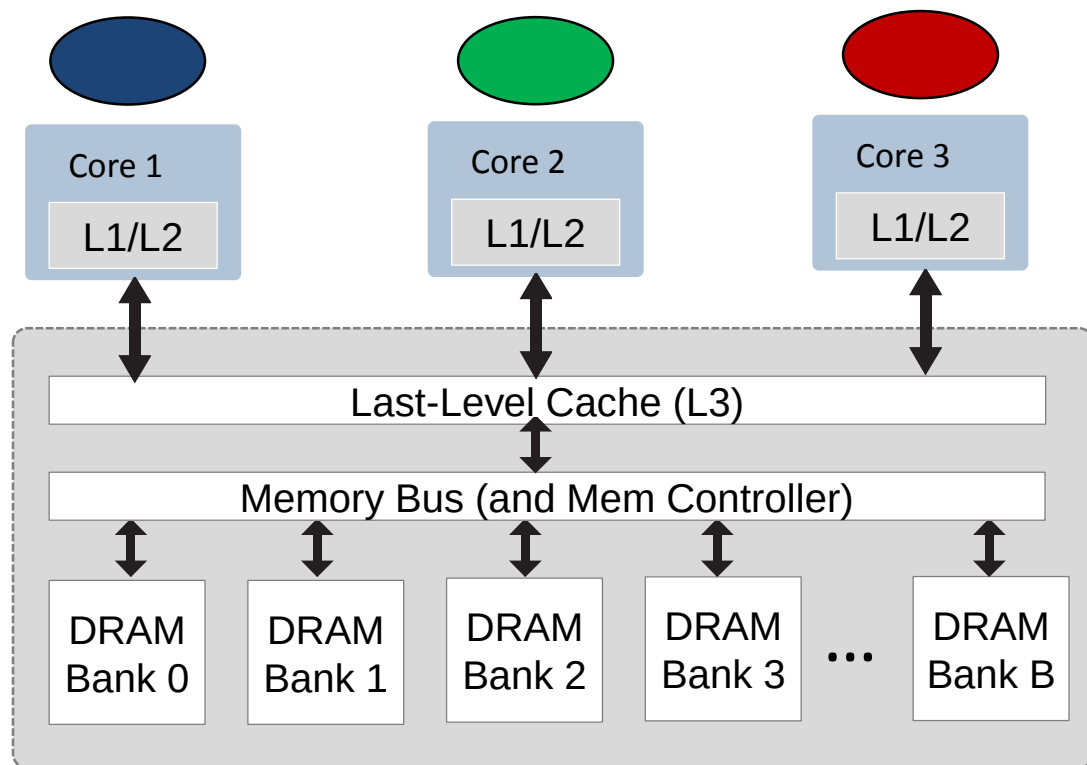


If speed is not additive, how can we describe speed as a function of co-runners?

Considerations in creating an abstraction for multicore

Speed of execution of a task as a function of co-runners can be either

- (i) Additive
- (ii) Sub-additive
- (iii) Super-additive

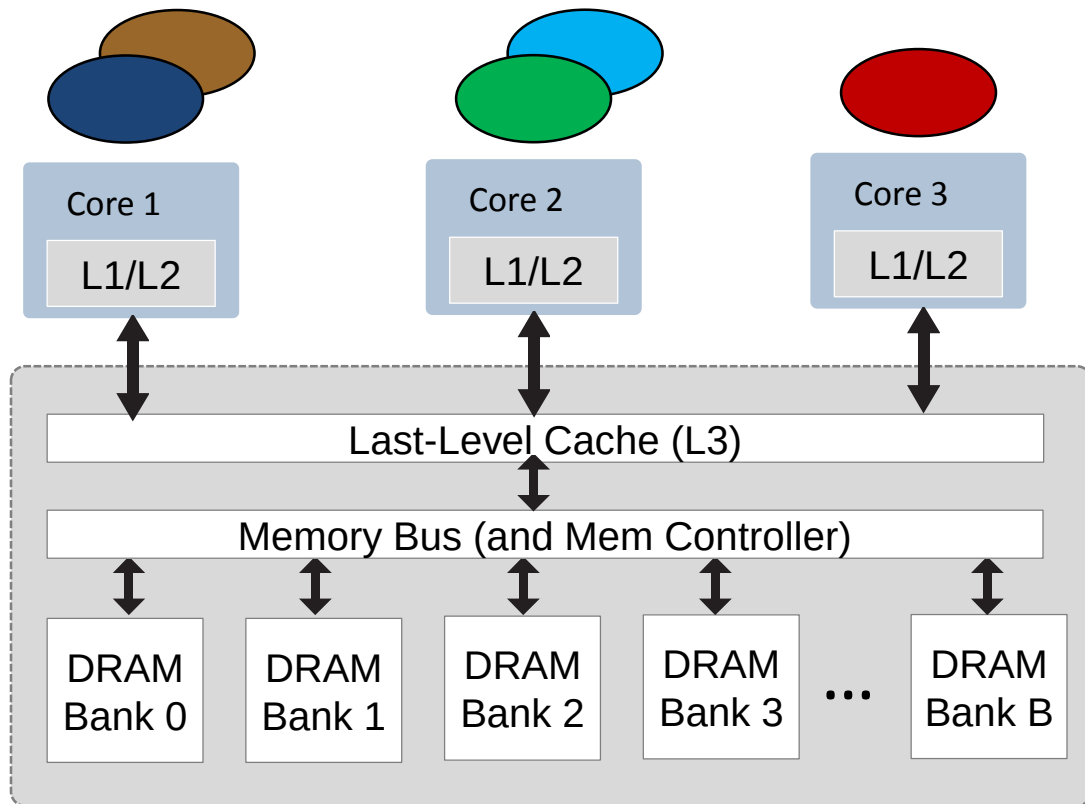


For each task: Enumerate the set of possible co-runner set.

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
 for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$



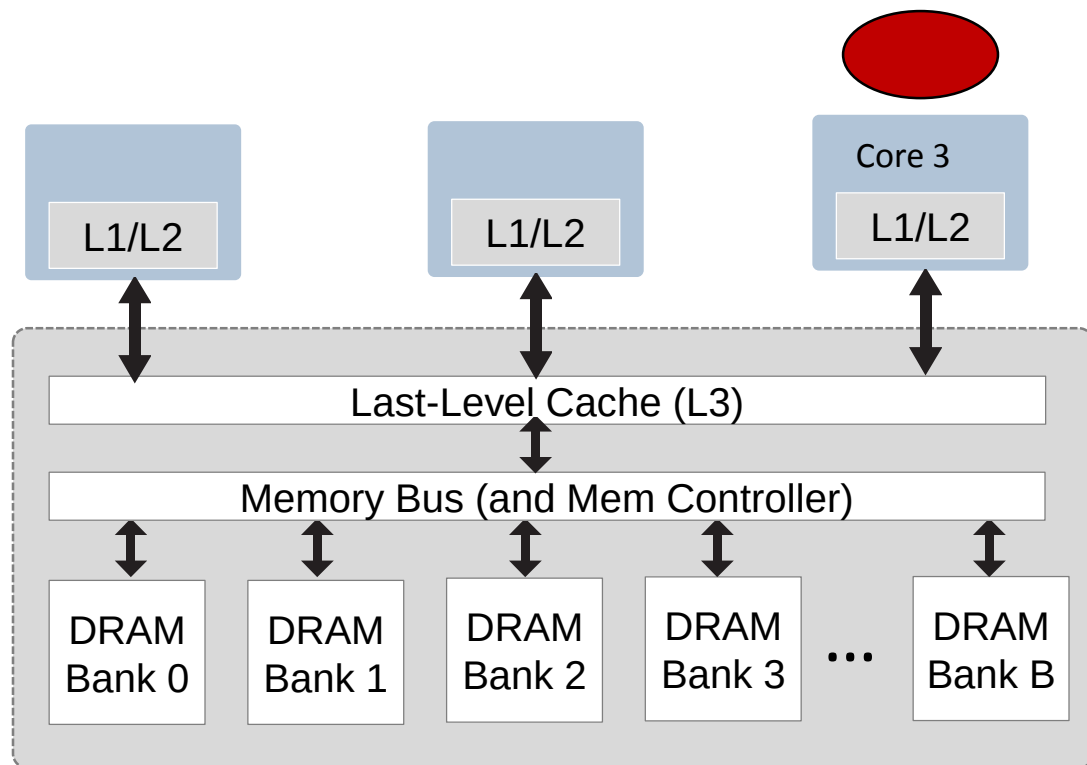
For each task: Enumerate the set of possible co-runner set.

Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$$C_{\text{red}}=4$$

Co-runner set	Speed
{}	1

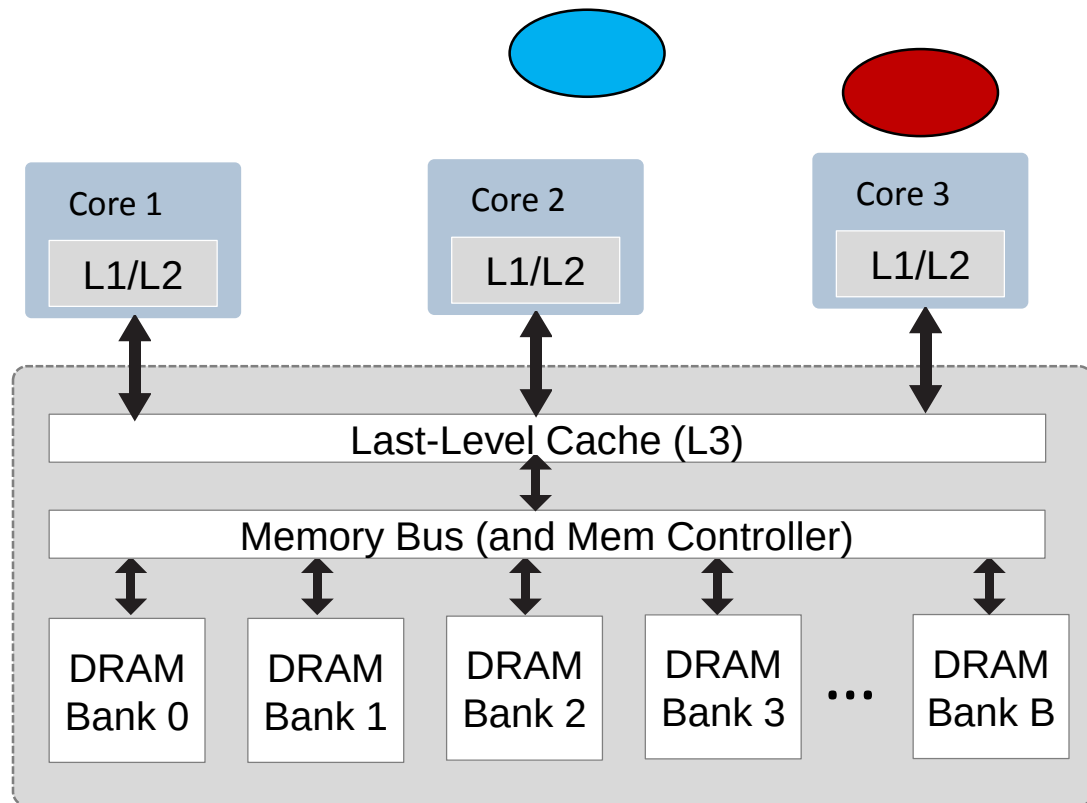


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$$C_{\text{red}}=4$$

Co-runner set	Speed
{}	1
{cyan}	0.5

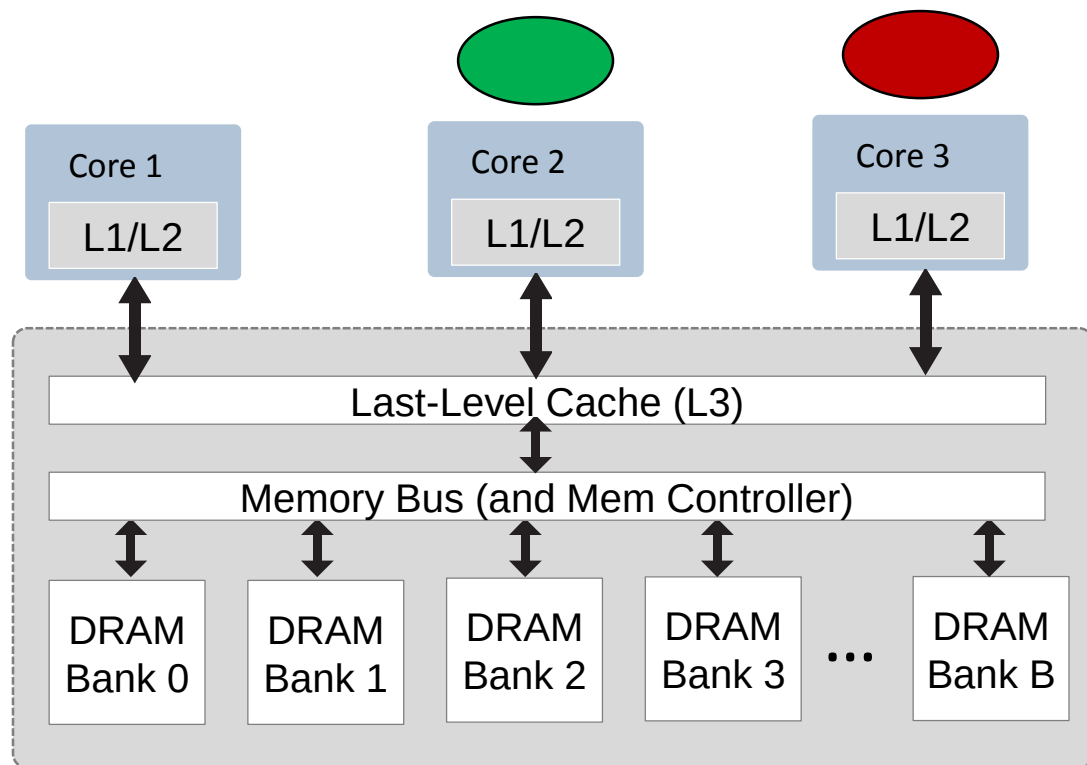


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$$C_{\text{red}}=4$$

Co-runner set	Speed
{}	1
{green}	0.45

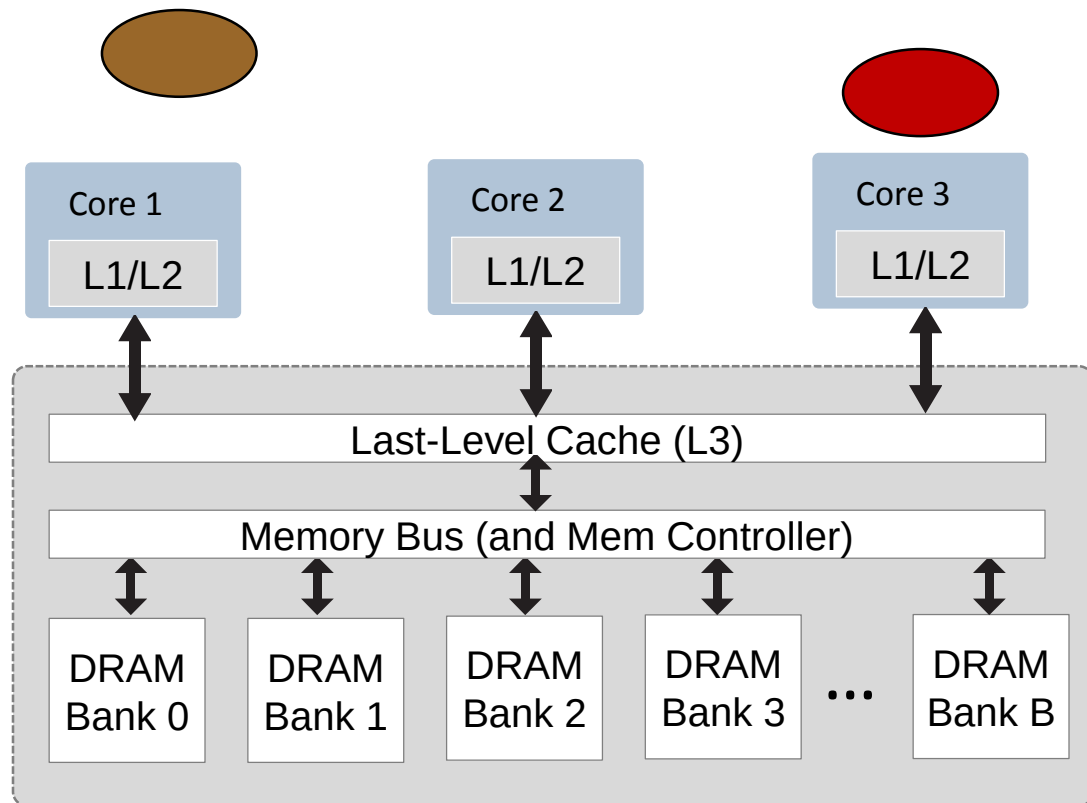


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$$C_{\text{red}}=4$$

Co-runner set	Speed
{}	1
{brown}	0.45

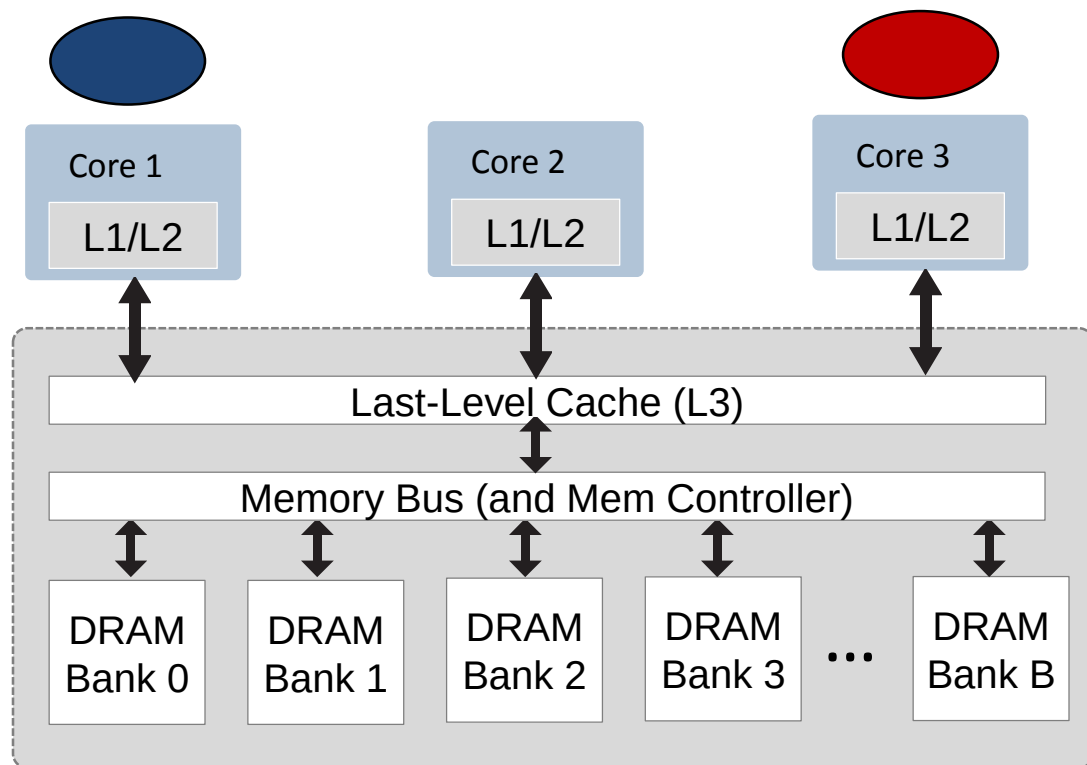


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$C_{red}=4$

Co-runner set	Speed
{}	1
{blue}	0.25

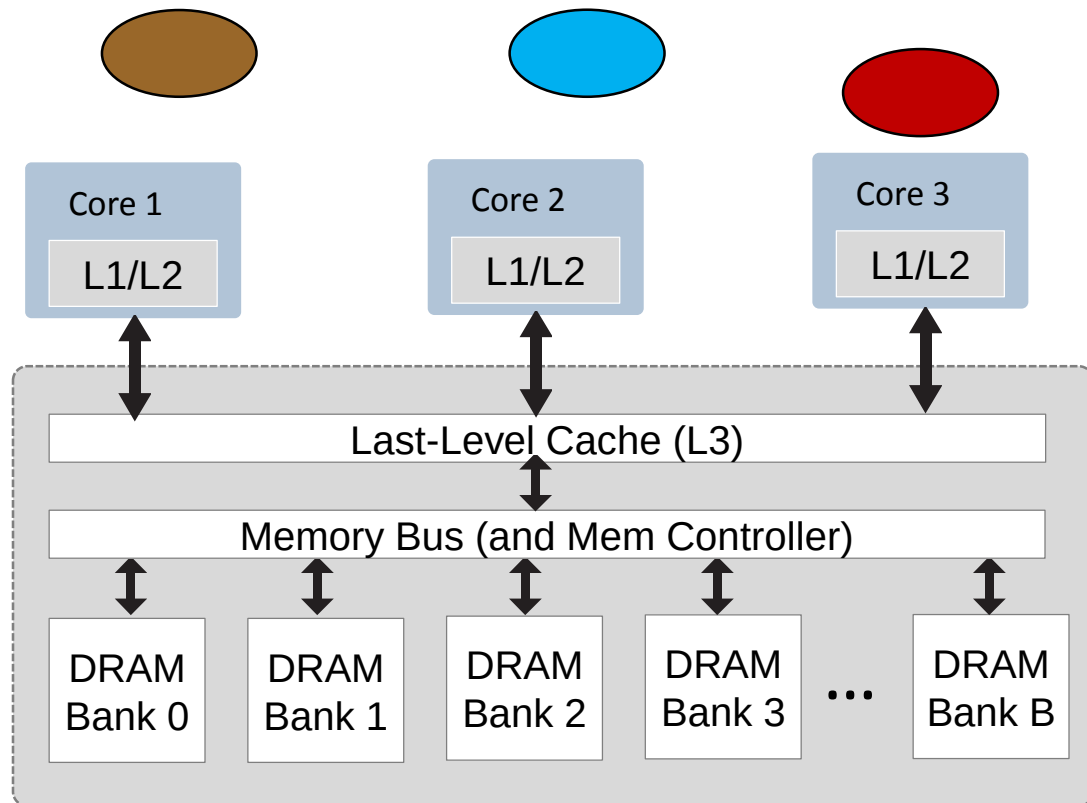


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$$C_{\text{red}}=4$$

Co-runner set	Speed
{}	1
{cyan, brown}	0.18

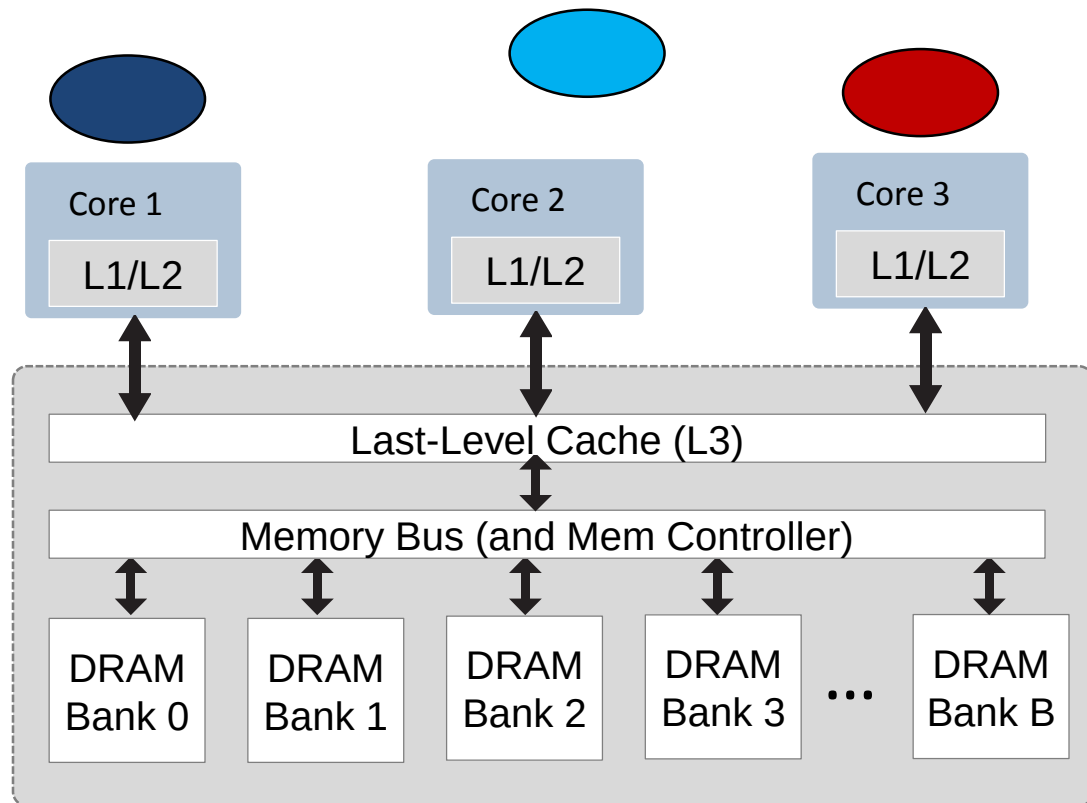


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$C_{red}=4$

Co-runner set	Speed
{}	1
{cyan, blue}	0.12

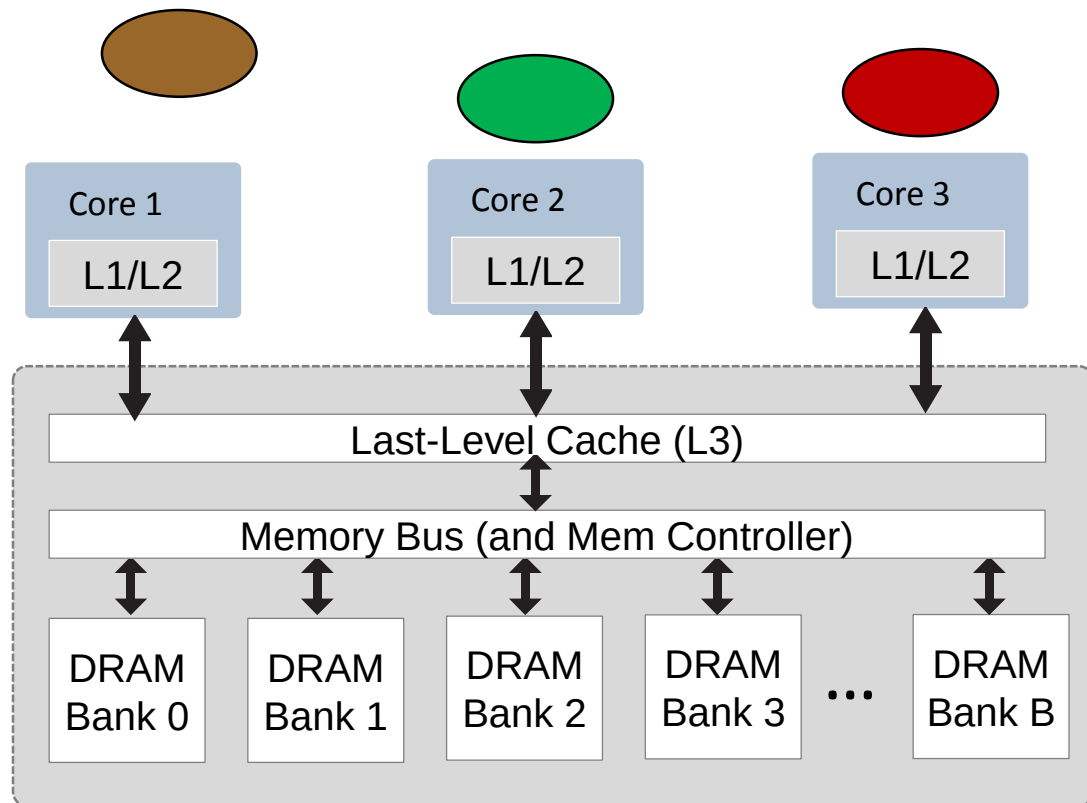


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$C_{red}=4$

Co-runner set	Speed
{}	1
{green,brown}	0.13

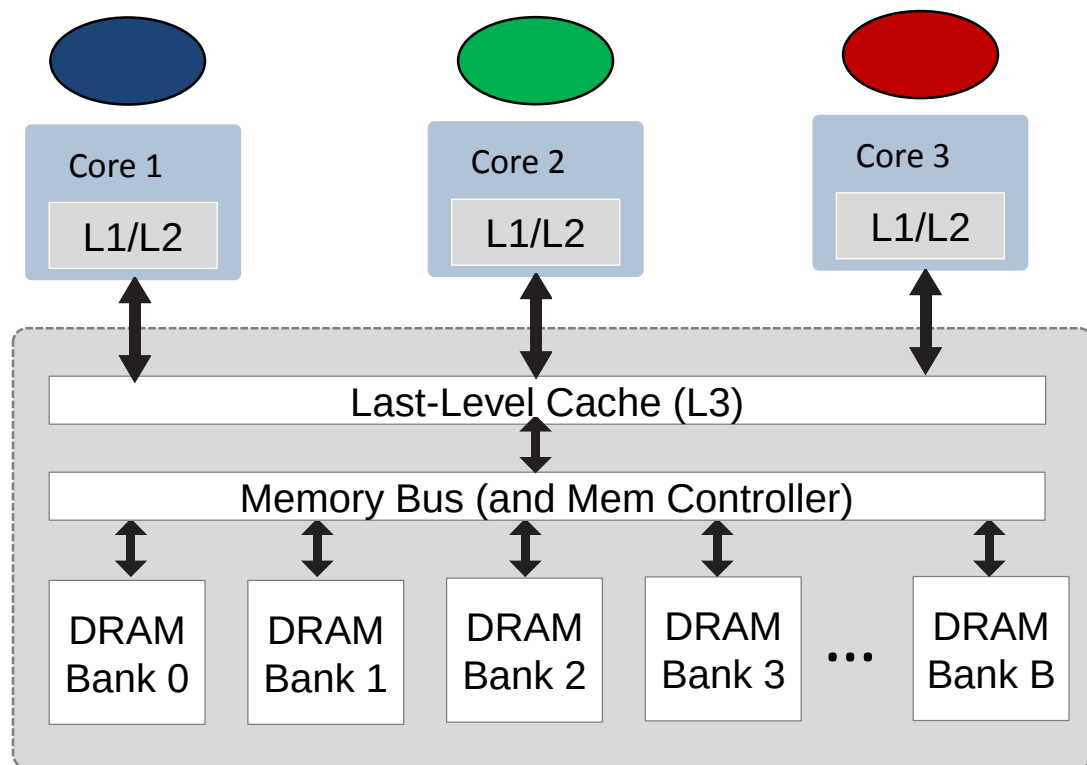


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$C_{red}=4$

Co-runner set	Speed
{}	1
{green, blue}	0.19

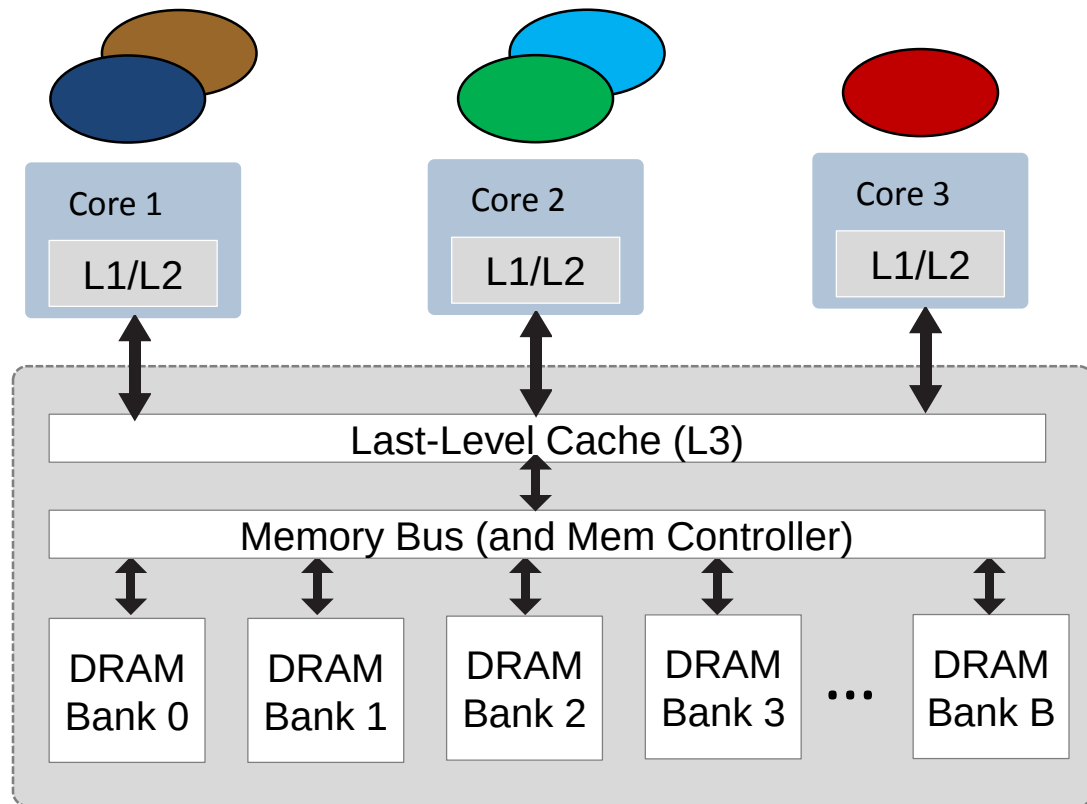


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$C_{\text{red}}=4$

Co-runner set	Speed
{}	1
{cyan}	0.5
{green}	0.45
{brown}	0.45
{blue}	0.25
{cyan, brown}	0.18
{cyan, blue}	0.12
{green, brown}	0.13
{green, blue}	0.19

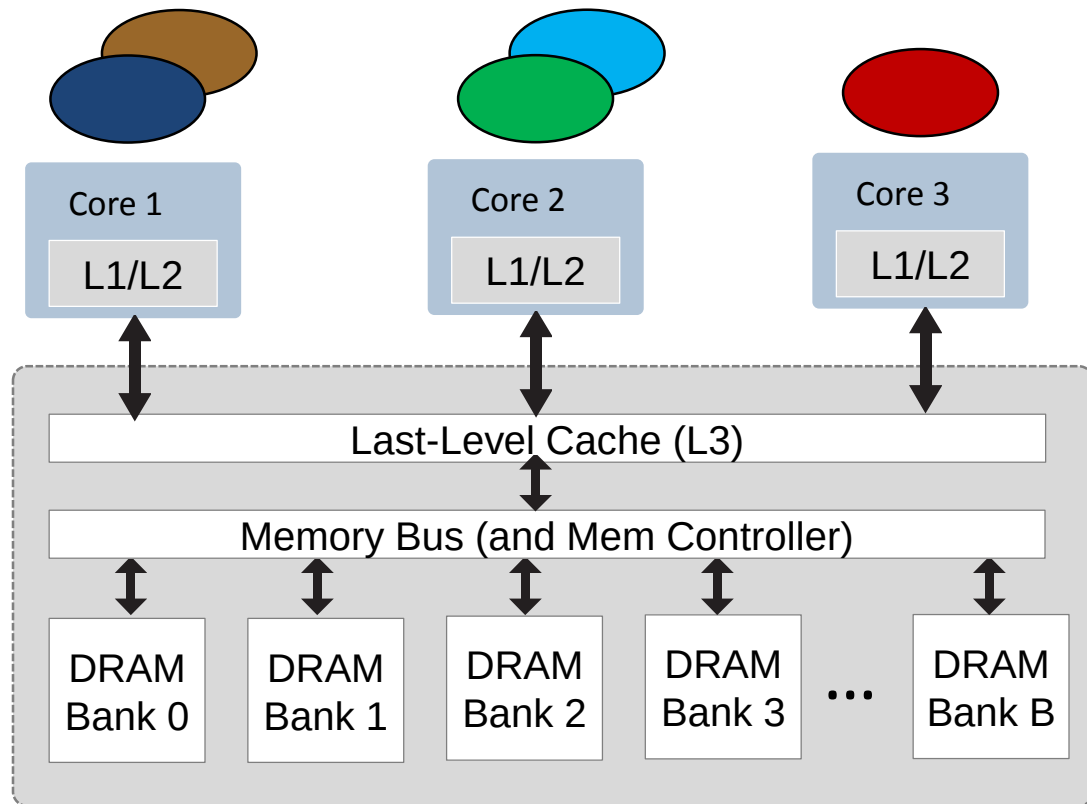


Considerations in creating an abstraction for multicore

Describing resource consumption of Red task

$$C_{\text{red}}=4$$

Co-runner set	Speed
{}	1
{cyan}	0.5
{green}	0.45
{brown}	0.45
{blue}	0.25
{cyan, brown}	0.18
{cyan, blue}	0.12
{green, brown}	0.13
{green, blue}	0.19

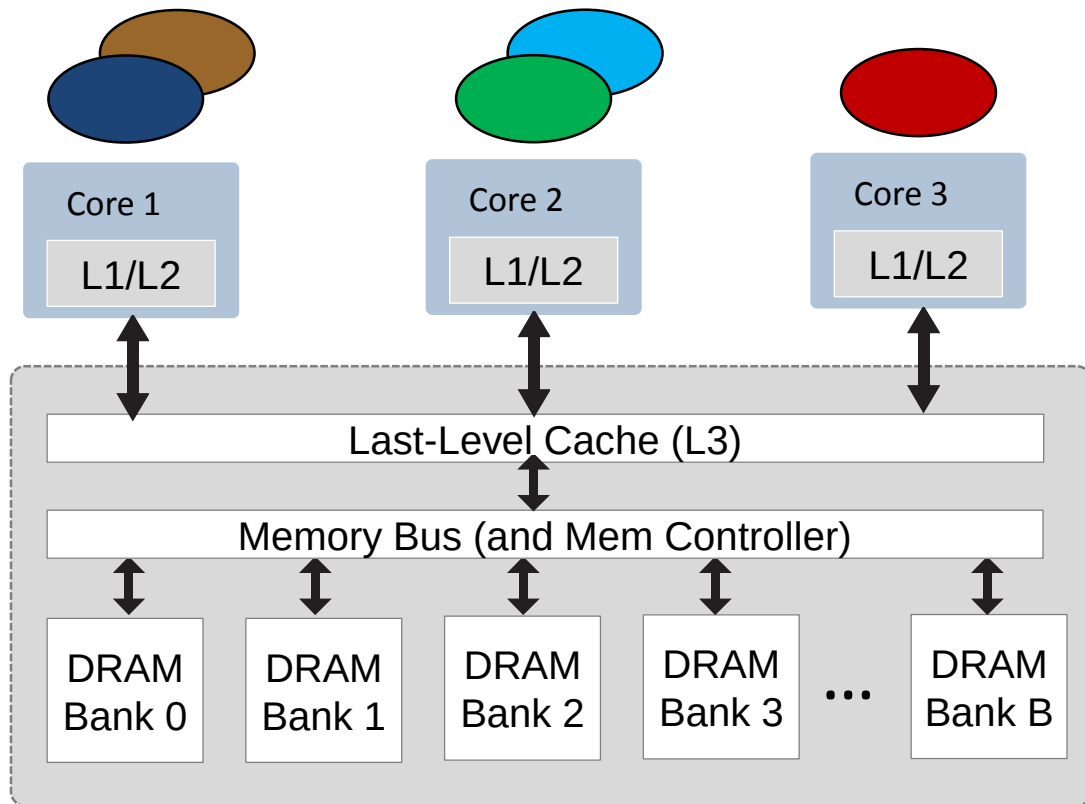


Describe resource consumption of other tasks analogously.

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$



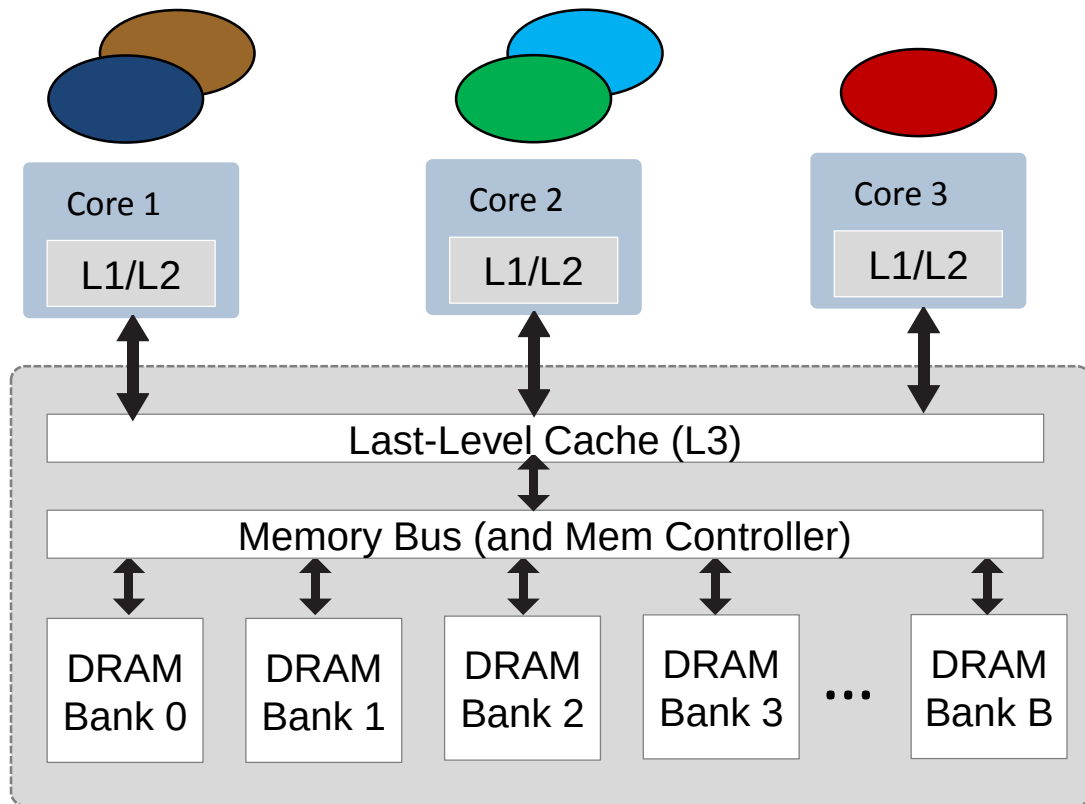
For each task: Enumerate the set of possible co-runner set.

Isn't that exponential?
Couldn't this be bad?

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
 for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$

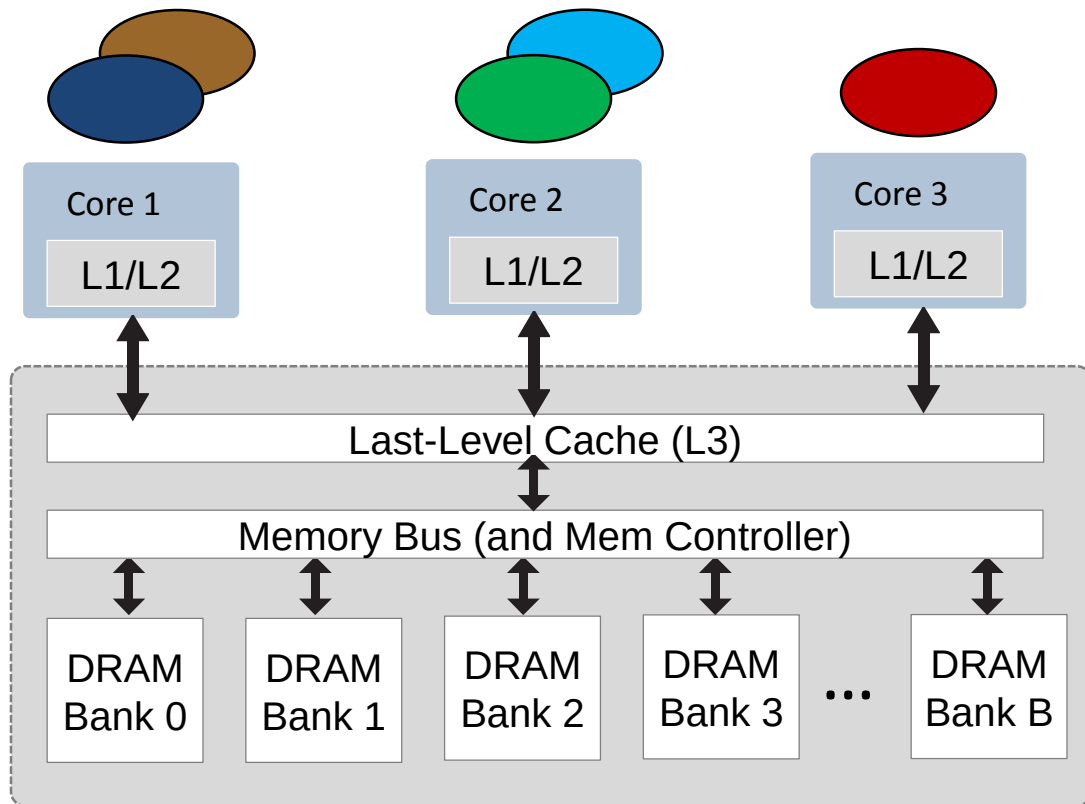


Given a task i : #co-runner sets of task $i \leq \left(\frac{ntasks - 1}{nprocessors - 1} + 1 \right)^{nprocessors - 1}$

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
 for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$



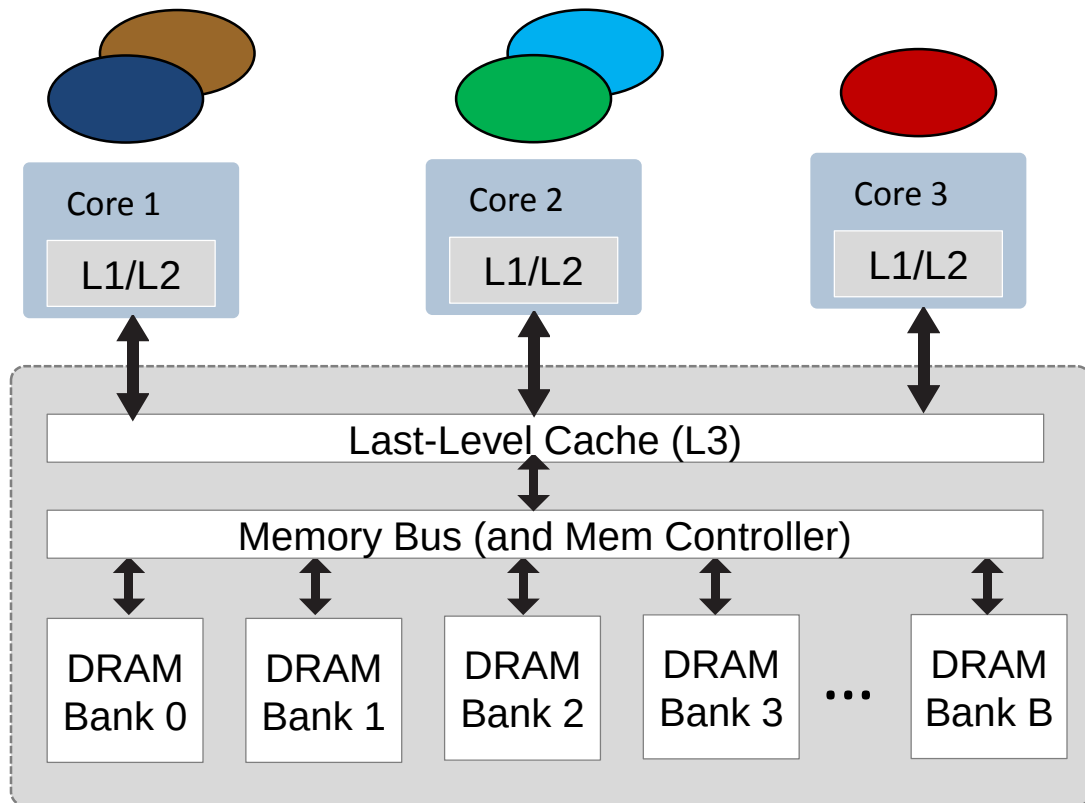
Given a task i : #co-runner sets of task $i \leq \text{polynomial}$

If number of processors is fixed.

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$



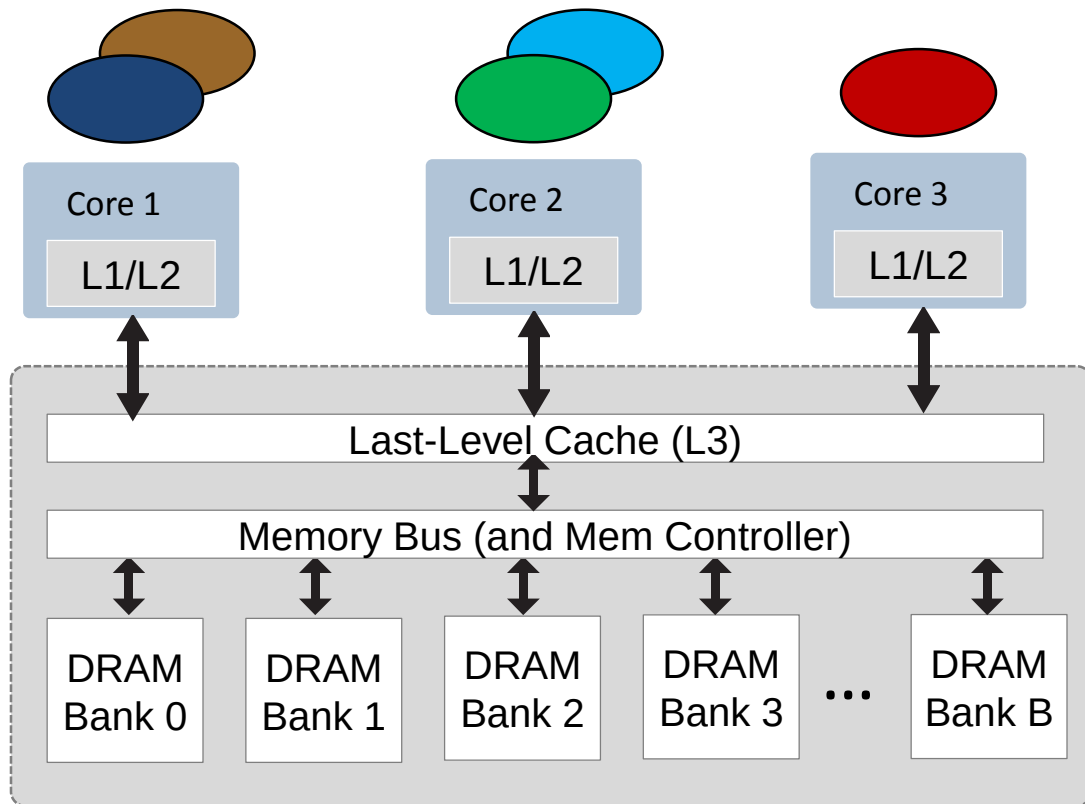
Given a task i : #co-runner sets of task $i \leq n_{tasks}$

If number of processors = 2.

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$



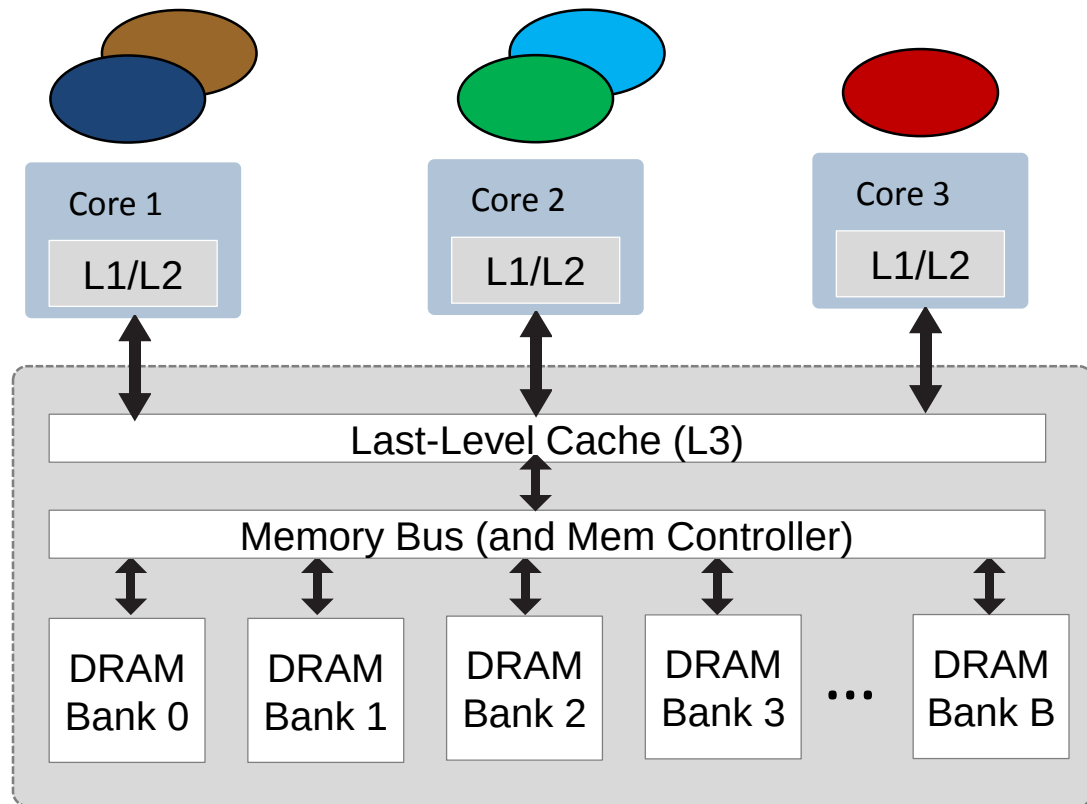
Given a task i : #co-runner sets of task $i \leq \left(\frac{ntasks - 1}{2} + 1\right)^2$

If number of processors = 3.

Considerations in creating an abstraction for multicore

Main idea:

For each task i ,
for each co-runner set of task i do
 $lowerbound speed_i^{cor} \leq speed(i, t) \leq 1$

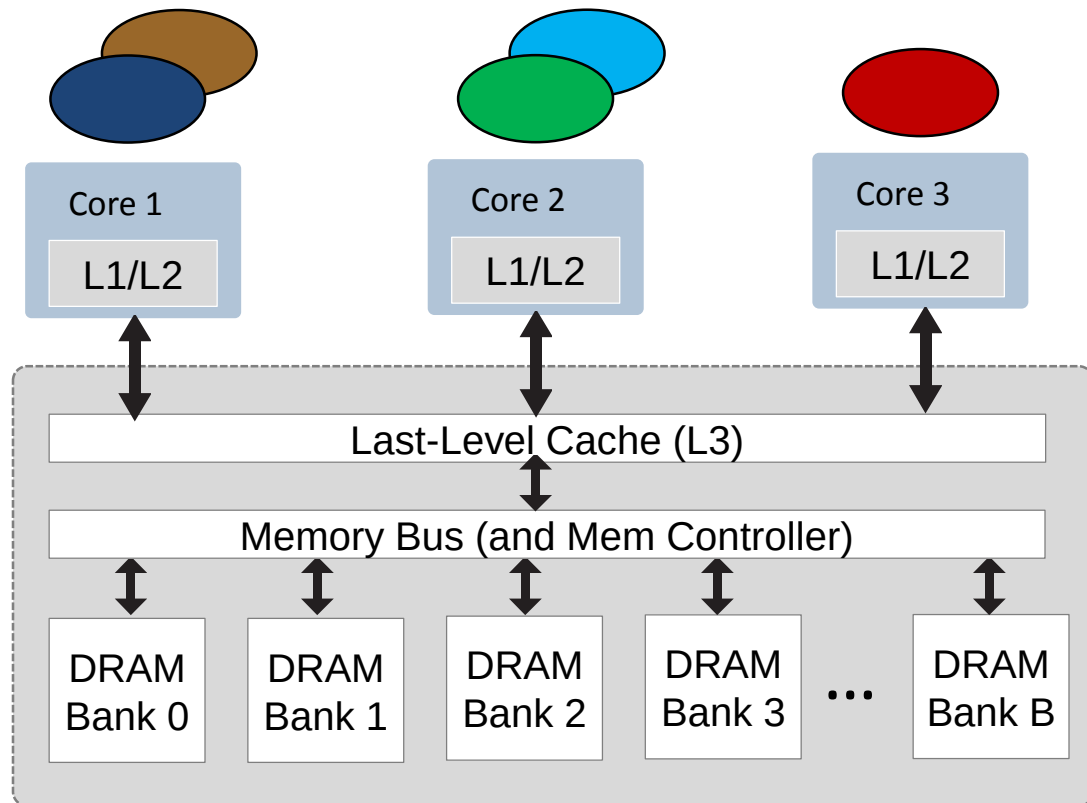


Given a task i : #co-runner sets of task $i \leq \left(\frac{ntasks - 1}{3} + 1\right)^3$

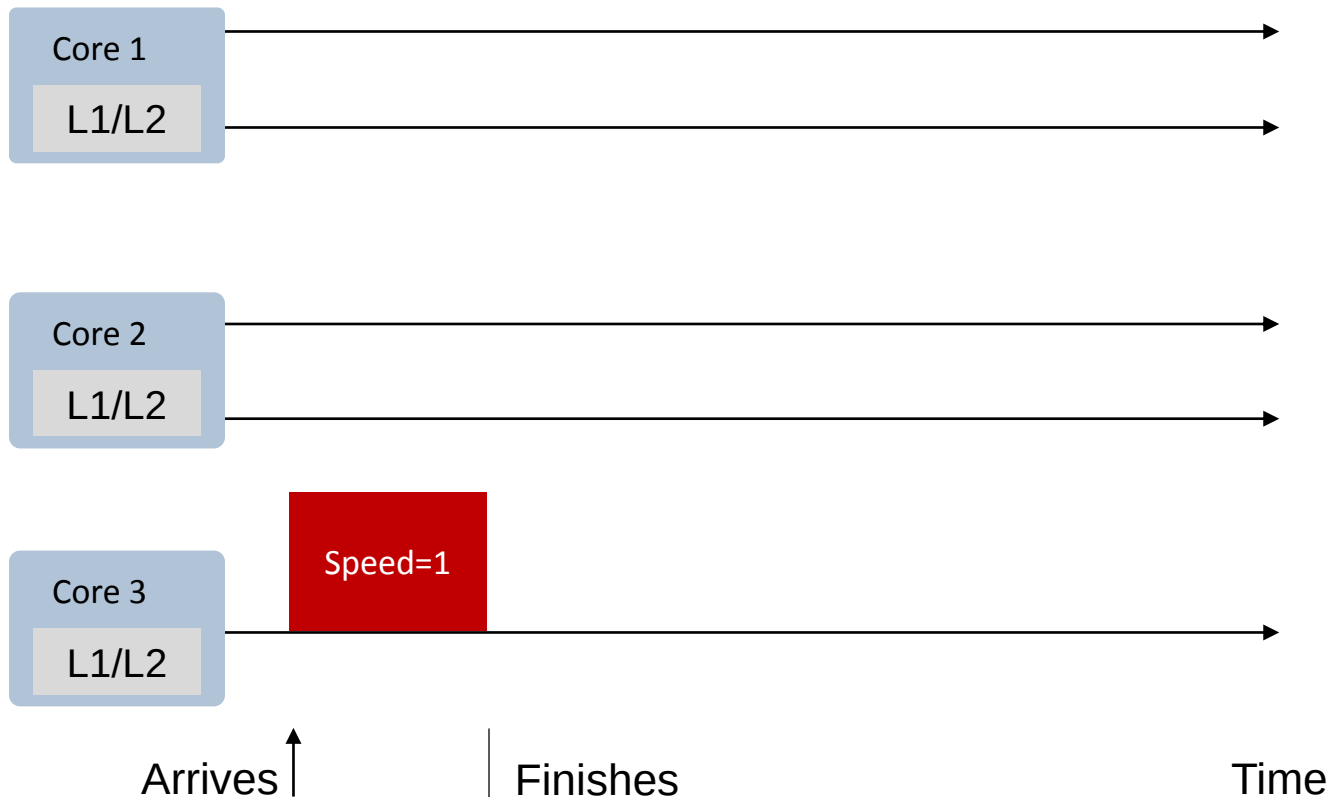
If number of processors = 4.

Considerations in creating an abstraction for multicore

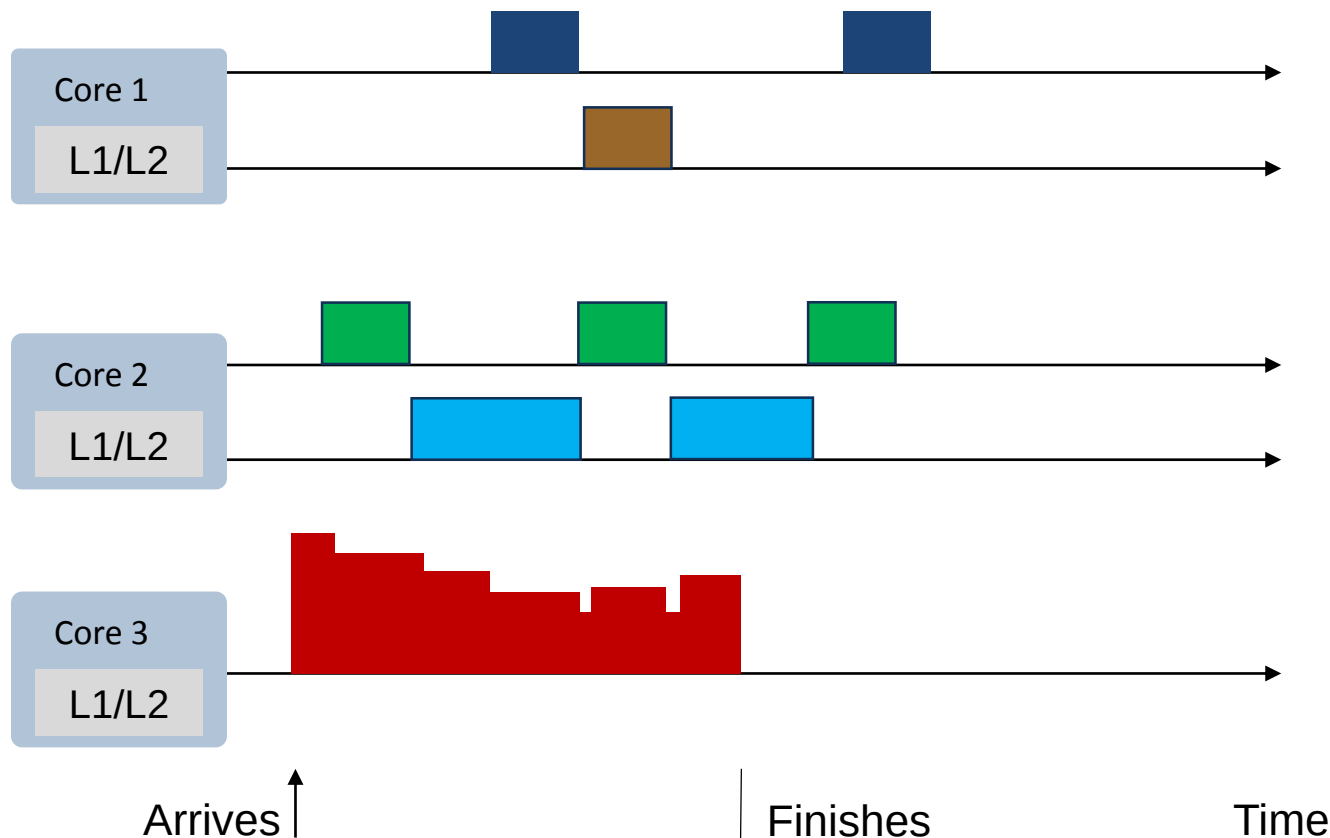
Given a task i : A job of task i can experience different co-runner set at different times.



How Co-Runners Impact Speed of Execution

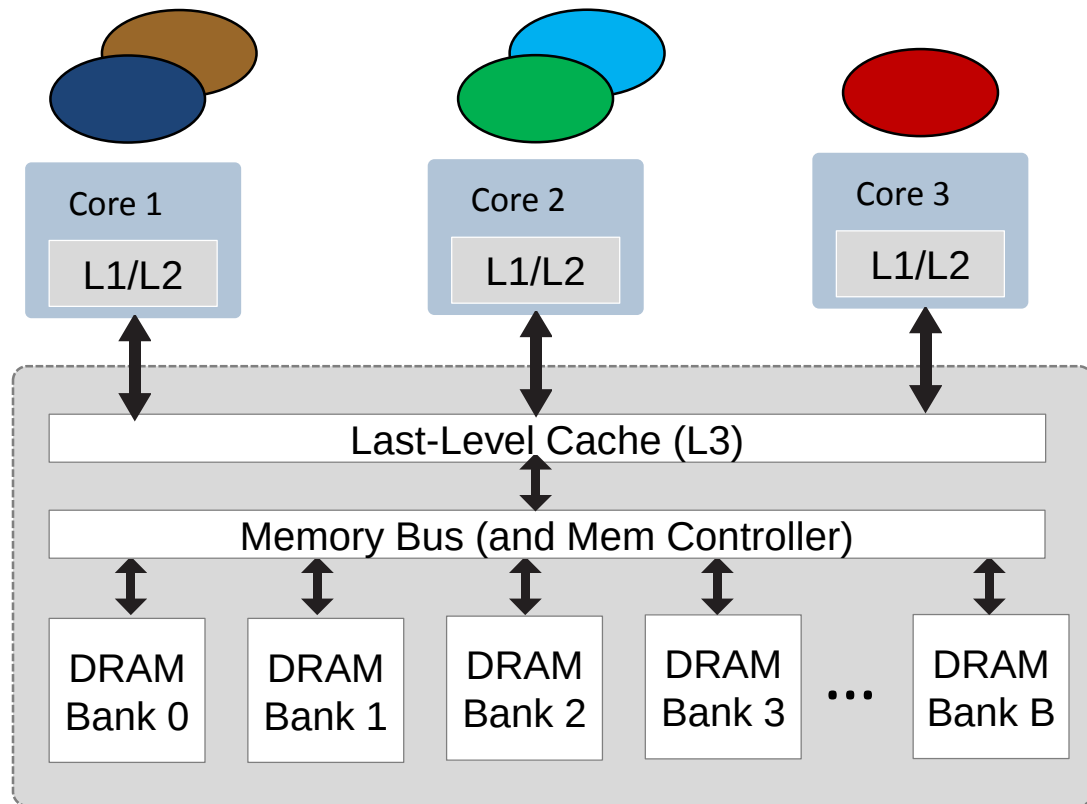


How Co-Runners Impact Speed of Execution



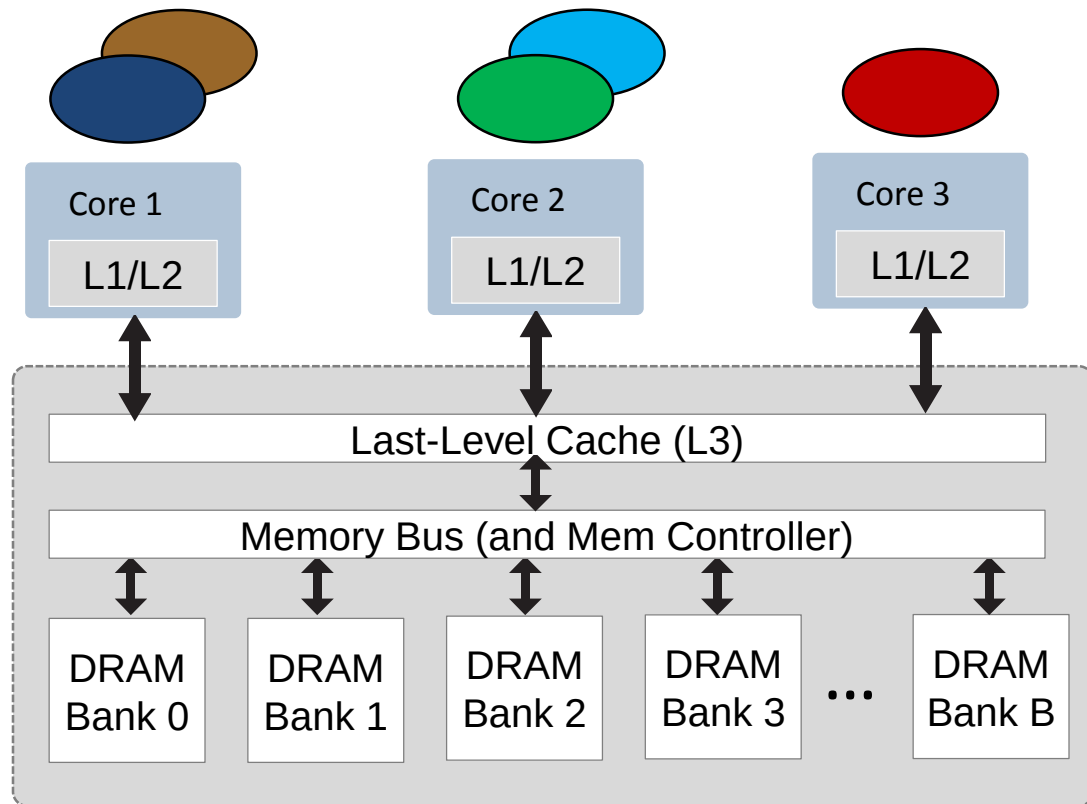
Considerations in creating an abstraction for multicore

Given a task i : A job of task i can experience different co-runner set at different times.

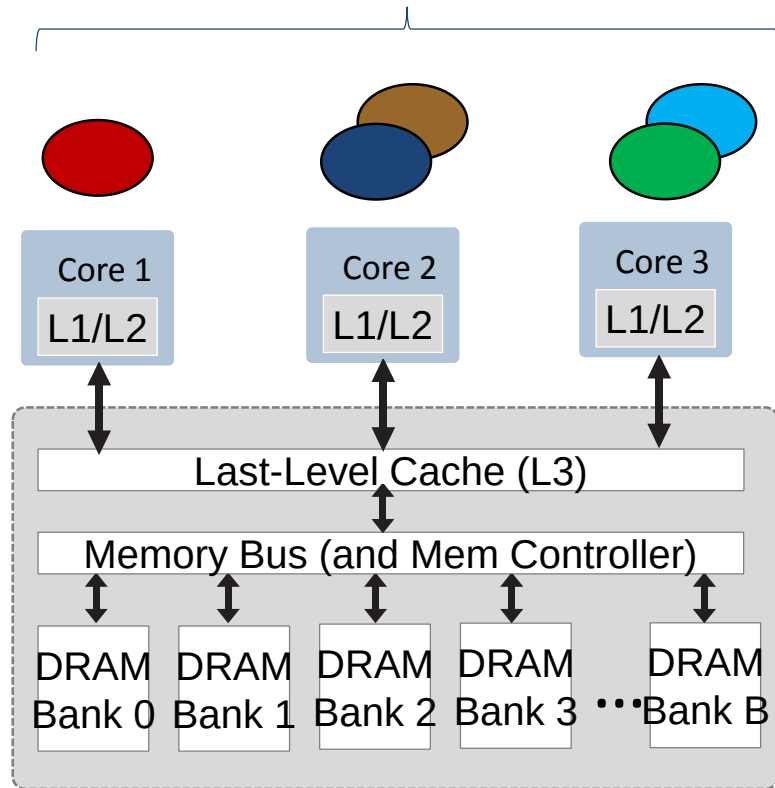


Considerations in creating an abstraction for multicore

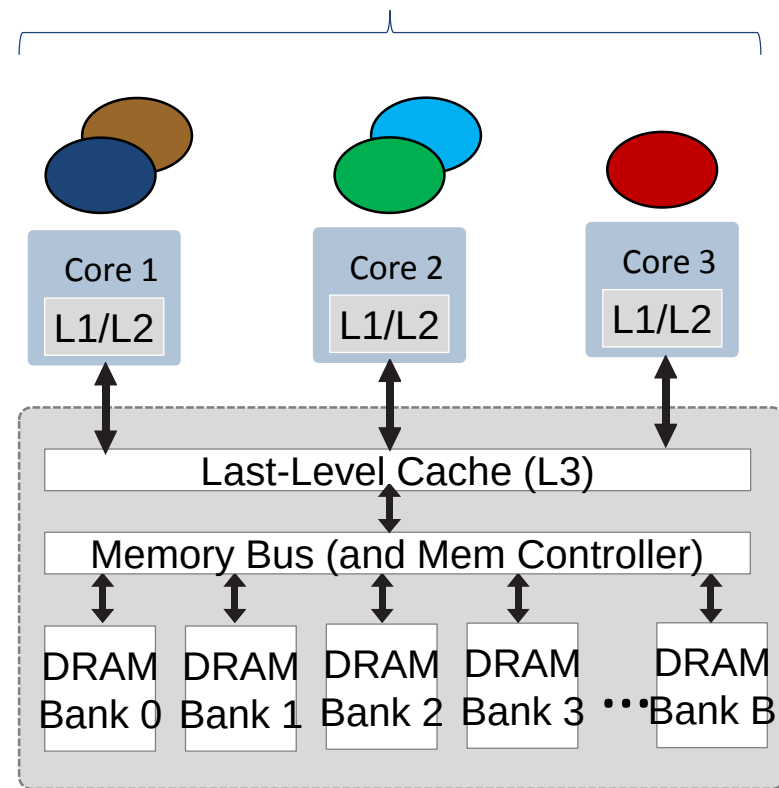
In some computer systems, there are some resources (e.g., memory bus) where the arbitration depends on the processor id of the requestor.



unschedulable

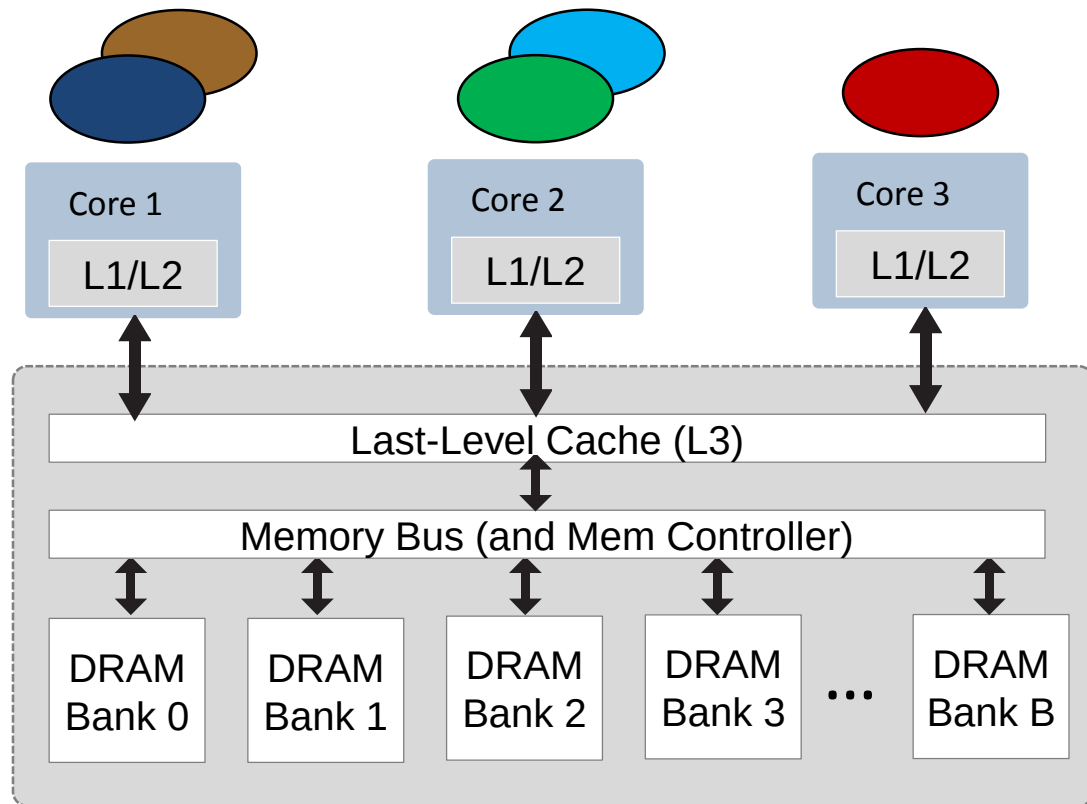


schedulable



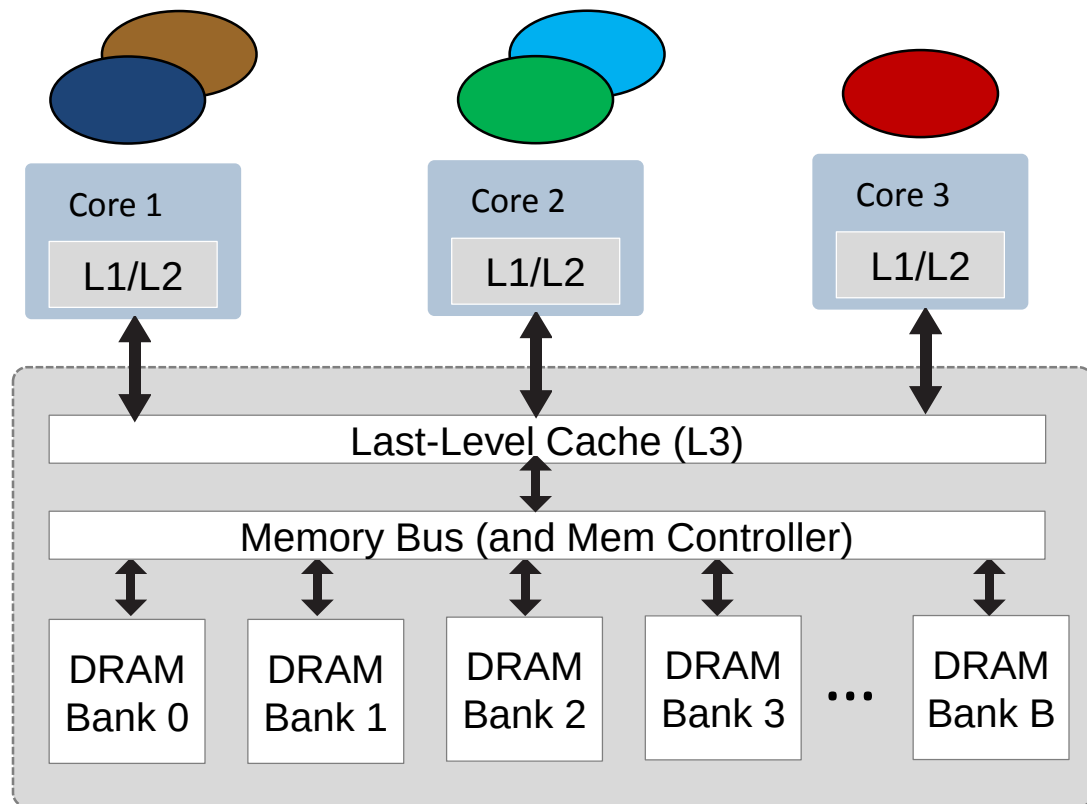
Considerations in creating an abstraction for multicore

In some computer systems, there are some resources (e.g., memory bus) where the arbitration depends on the processor id of the requestor.



Considerations in creating an abstraction for multicore

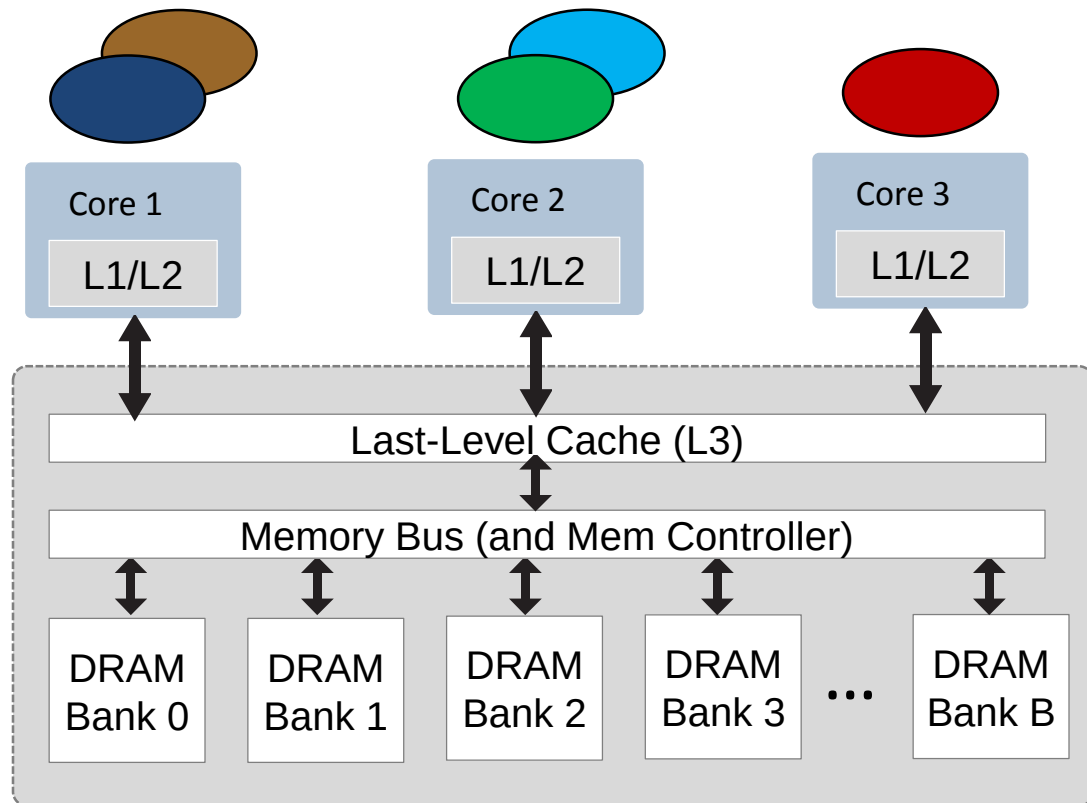
The program behavior may change over time.



Considerations in creating an abstraction for multicore

The program behavior may change over time.

```
while (1) {
    s = wait_until_next_sample()
    update_datastructures(s)
    a = compute_actuation_command()
    actuate_command(a)
}
```

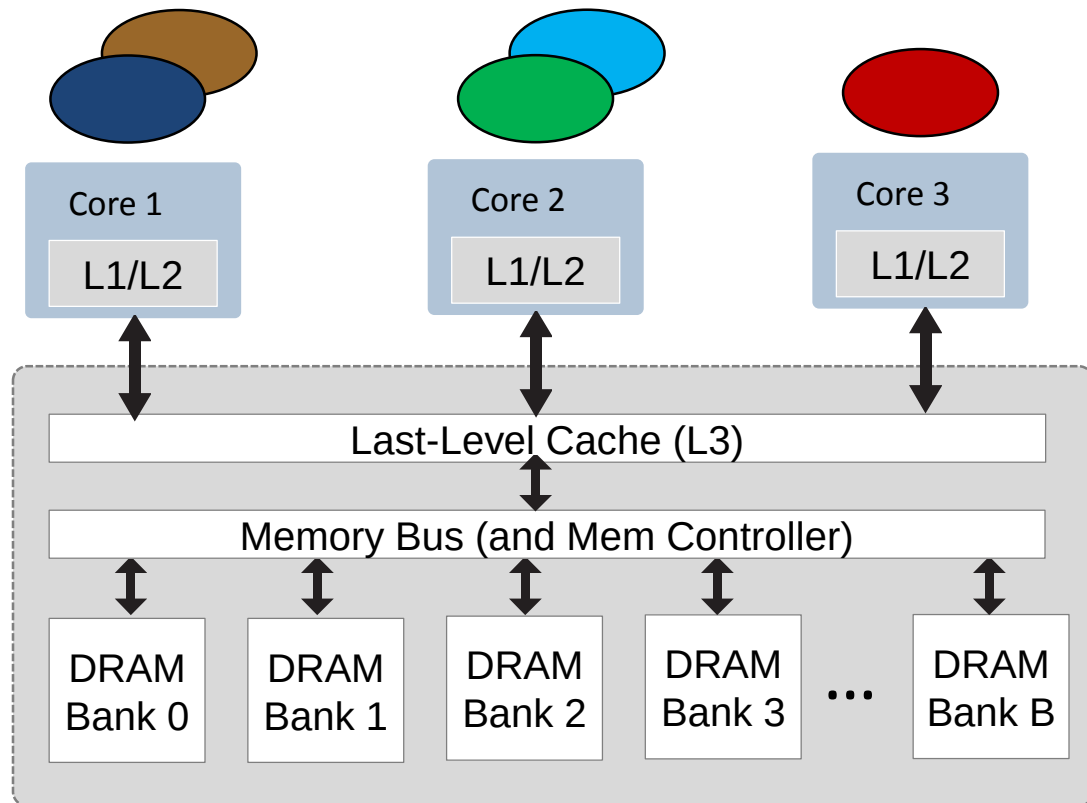


Considerations in creating an abstraction for multicore

The program behavior may change over time.

```
while (1) {
    s = wait_until_next_sample()
    → update_datastructures(s)
    a = compute_actuation_command()
    actuate_command(a)
}
```

The program behavior here



Considerations in creating an abstraction for multicore

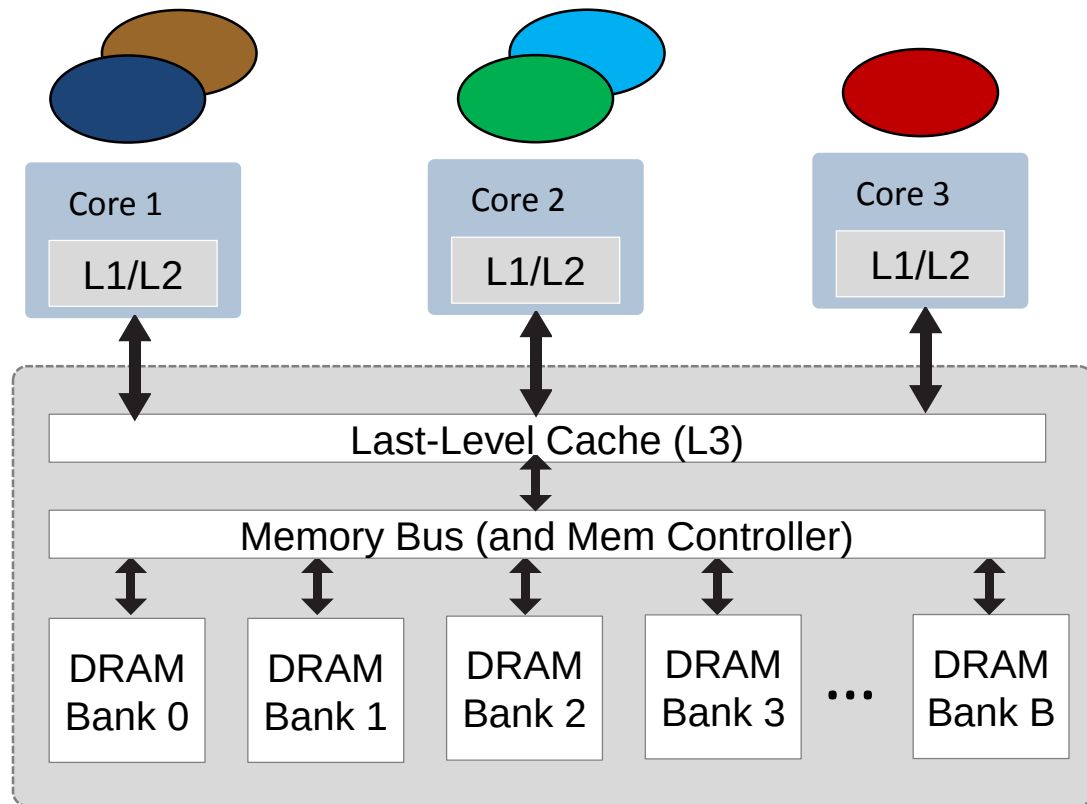
The program behavior may change over time.

```
while (1) {
    s = wait_until_next_sample()
    → update_datastructures(s)
    a = compute_actuation_command() ←
    actuate_command(a)
}
```

The program
behavior here

is different from

the program
behavior here.

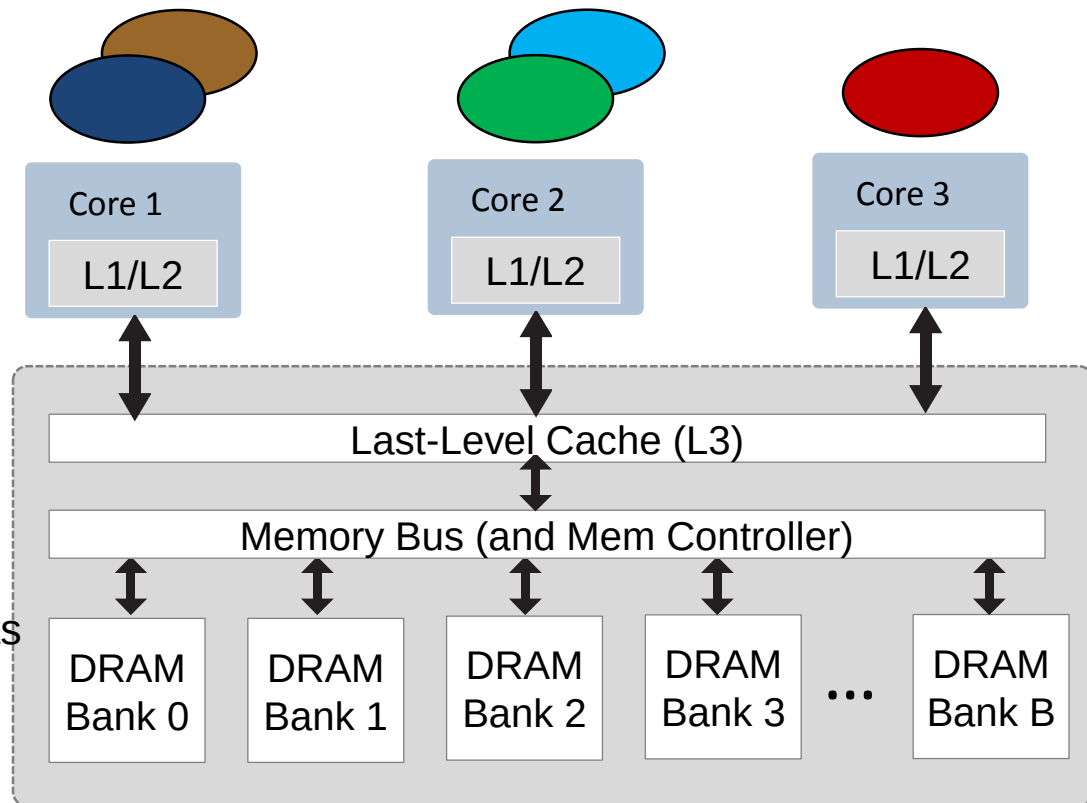


Considerations in creating an abstraction for multicore

The program behavior may change over time.

```
while (1) {
    s = wait_until_next_sample()
    update_datastructures(s)
    a = compute_actuation_command()
    actuate_command(a)
}
```

Describe a task as a sequence of segments where each segment may have different description of lower bound on speed as a function of co-runners.



Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & V_5 &= \{v_5^1, v_5^2\} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Taskset example

5 Task

$$|\tau| = 5$$

$$\text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{array}{llll} T_1 & = 1.500 & D_1 & = 1.500 & V_1 = \{v_1^1\} & \text{prio}_1 = 3 & \text{proc}_1 = 1 \\ C_1^1 & = 0.250 & \text{pd}_1^1 & = 0.500 & \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_2 & = 2.000 & D_2 & = 2.000 & V_2 = \{v_2^1\} & \text{prio}_2 = 2 & \text{proc}_2 = 1 \\ C_2^1 & = 0.250 & \text{pd}_2^1 & = 0.500 & \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_3 & = 2.000 & D_3 & = 2.000 & V_3 = \{v_3^1\} & \text{prio}_3 = 3 & \text{proc}_3 = 2 \\ C_3^1 & = 0.250 & \text{pd}_3^1 & = 0.500 & \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{array}$$

$$\begin{array}{llll} T_4 & = 2.000 & D_4 & = 2.000 & V_4 = \{v_4^1\} & \text{prio}_4 = 2 & \text{proc}_4 = 2 \\ C_4^1 & = 0.250 & \text{pd}_4^1 & = 0.500 & \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_5 & = 2.250 & D_5 & = 2.250 & V_5 = \{v_5^1, v_5^2\} & \text{prio}_5 = 1 & \text{proc}_5 = 2 \\ C_5^1 & = 0.500 & \text{pd}_5^1 & = 1.000 & \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 & = 0.125 & \text{pd}_5^2 & = 0.500 & \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

Taskset example

2 processors

$$|\tau| = 5$$

$$\text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{array}{llll} T_1 & = 1.500 & D_1 & = 1.500 & V_1 = \{v_1^1\} & \text{prio}_1 = 3 & \text{proc}_1 = 1 \\ C_1^1 & = 0.250 & \text{pd}_1^1 & = 0.500 & \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_2 & = 2.000 & D_2 & = 2.000 & V_2 = \{v_2^1\} & \text{prio}_2 = 2 & \text{proc}_2 = 1 \\ C_2^1 & = 0.250 & \text{pd}_2^1 & = 0.500 & \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_3 & = 2.000 & D_3 & = 2.000 & V_3 = \{v_3^1\} & \text{prio}_3 = 3 & \text{proc}_3 = 2 \\ C_3^1 & = 0.250 & \text{pd}_3^1 & = 0.500 & \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{array}$$

$$\begin{array}{llll} T_4 & = 2.000 & D_4 & = 2.000 & V_4 = \{v_4^1\} & \text{prio}_4 = 2 & \text{proc}_4 = 2 \\ C_4^1 & = 0.250 & \text{pd}_4^1 & = 0.500 & \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_5 & = 2.250 & D_5 & = 2.250 & V_5 = \{v_5^1, v_5^2\} & \text{prio}_5 = 1 & \text{proc}_5 = 2 \\ C_5^1 & = 0.500 & \text{pd}_5^1 & = 1.000 & \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 & = 0.125 & \text{pd}_5^2 & = 0.500 & \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{array}{llll} T_1 & = 1.500 & D_1 & = 1.500 & V_1 = \{v_1^1\} & \text{prio}_1 = 3 & \text{proc}_1 = 1 \\ C_1^1 & = 0.250 & \text{pd}_1^1 & = 0.500 & \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_2 & = 2.000 & D_2 & = 2.000 & V_2 = \{v_2^1\} & \text{prio}_2 = 2 & \text{proc}_2 = 1 \\ C_2^1 & = 0.250 & \text{pd}_2^1 & = 0.500 & \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_3 & = 2.000 & D_3 & = 2.000 & V_3 = \{v_3^1\} & \text{prio}_3 = 3 & \text{proc}_3 = 2 \\ C_3^1 & = 0.250 & \text{pd}_3^1 & = 0.500 & \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{array}$$

$$\begin{array}{llll} T_4 & = 2.000 & D_4 & = 2.000 & V_4 = \{v_4^1\} & \text{prio}_4 = 2 & \text{proc}_4 = 2 \\ C_4^1 & = 0.250 & \text{pd}_4^1 & = 0.500 & \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_5 & = 2.250 & D_5 & = 2.250 & V_5 = \{v_5^1, v_5^2\} & \text{prio}_5 = 1 & \text{proc}_5 = 2 \\ C_5^1 & = 0.500 & \text{pd}_5^1 & = 1.000 & \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 & = 0.125 & \text{pd}_5^2 & = 0.500 & \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

Task 1

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & V_5 &= \{v_5^1, v_5^2\} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Task 2

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

Task 3

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & V_5 &= \{v_5^1, v_5^2\} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & V_5 &= \{v_5^1, v_5^2\} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Task 4

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & V_5 &= \{v_5^1, v_5^2\} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Task 5

Taskset example

Minimum inter-arrival times of tasks

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\boxed{T_1 = 1.500} \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$

$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$\boxed{T_2 = 2.000} \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$

$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$\boxed{T_3 = 2.000} \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$

$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$\boxed{T_4 = 2.000} \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$

$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$\boxed{T_5 = 2.250} \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$

$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$

$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

Deadlines of tasks

$$T_1 = 1.500 \quad \boxed{D_1 = 1.500} \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$

$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_2 = 2.000 \quad \boxed{D_2 = 2.000} \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$

$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad \boxed{D_3 = 2.000} \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$

$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad \boxed{D_4 = 2.000} \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$

$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad \boxed{D_5 = 2.250} \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$

$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$

$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

Sets of segments for each task

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & V_5 &= \{v_5^1, v_5^2\} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$T_1 = 1.500 \quad D_1 = 1.500 \quad \boxed{V_1 = \{v_1^1\}} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$

$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

Task 1 has 1 segment.

$$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$

$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$

$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$

$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$

$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$

$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$\begin{aligned} T_1 &= 1.500 & D_1 &= 1.500 & V_1 &= \{v_1^1\} & \text{prio}_1 &= 3 & \text{proc}_1 &= 1 \\ C_1^1 &= 0.250 & \text{pd}_1^1 &= 0.500 & \text{CO}_1^1 &= \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_2 &= 2.000 & D_2 &= 2.000 & V_2 &= \{v_2^1\} & \text{prio}_2 &= 2 & \text{proc}_2 &= 1 \\ C_2^1 &= 0.250 & \text{pd}_2^1 &= 0.500 & \text{CO}_2^1 &= \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_3 &= 2.000 & D_3 &= 2.000 & V_3 &= \{v_3^1\} & \text{prio}_3 &= 3 & \text{proc}_3 &= 2 \\ C_3^1 &= 0.250 & \text{pd}_3^1 &= 0.500 & \text{CO}_3^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{aligned}$$

$$\begin{aligned} T_4 &= 2.000 & D_4 &= 2.000 & V_4 &= \{v_4^1\} & \text{prio}_4 &= 2 & \text{proc}_4 &= 2 \\ C_4^1 &= 0.250 & \text{pd}_4^1 &= 0.500 & \text{CO}_4^1 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

$$\begin{aligned} T_5 &= 2.250 & D_5 &= 2.250 & \boxed{V_5 = \{v_5^1, v_5^2\}} & \text{prio}_5 &= 1 & \text{proc}_5 &= 2 \\ C_5^1 &= 0.500 & \text{pd}_5^1 &= 1.000 & \text{CO}_5^1 &= \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 &= 0.125 & \text{pd}_5^2 &= 0.500 & \text{CO}_5^2 &= \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{aligned}$$

Task 5 has 2 segments.

Taskset example

$$|\tau| = 5$$

$$\text{procs}(\Pi) = \{P_1, P_2\}$$

Priorities for each task

$$\begin{array}{llll} T_1 & = 1.500 & D_1 & = 1.500 & V_1 = \{v_1^1\} & \boxed{\text{prio}_1 = 3} & \text{proc}_1 = 1 \\ C_1^1 & = 0.250 & \text{pd}_1^1 & = 0.500 & \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_2 & = 2.000 & D_2 & = 2.000 & V_2 = \{v_2^1\} & \boxed{\text{prio}_2 = 2} & \text{proc}_2 = 1 \\ C_2^1 & = 0.250 & \text{pd}_2^1 & = 0.500 & \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_3 & = 2.000 & D_3 & = 2.000 & V_3 = \{v_3^1\} & \boxed{\text{prio}_3 = 3} & \text{proc}_3 = 2 \\ C_3^1 & = 0.250 & \text{pd}_3^1 & = 0.500 & \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{array}$$

$$\begin{array}{llll} T_4 & = 2.000 & D_4 & = 2.000 & V_4 = \{v_4^1\} & \boxed{\text{prio}_4 = 2} & \text{proc}_4 = 2 \\ C_4^1 & = 0.250 & \text{pd}_4^1 & = 0.500 & \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_5 & = 2.250 & D_5 & = 2.250 & V_5 = \{v_5^1, v_5^2\} & \boxed{\text{prio}_5 = 1} & \text{proc}_5 = 2 \\ C_5^1 & = 0.500 & \text{pd}_5^1 & = 1.000 & \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 & = 0.125 & \text{pd}_5^2 & = 0.500 & \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

Taskset example

$|\tau| = 5$ $\text{procs}(\Pi) = \{P_1, P_2\}$ Processor to each task is assigned

$$\begin{array}{llll} T_1 & = 1.500 & D_1 & = 1.500 & V_1 = \{v_1^1\} & \text{prio}_1 = 3 & \boxed{\text{proc}_1 = 1} \\ C_1^1 & = 0.250 & \text{pd}_1^1 & = 0.500 & \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_2 & = 2.000 & D_2 & = 2.000 & V_2 = \{v_2^1\} & \text{prio}_2 = 2 & \boxed{\text{proc}_2 = 1} \\ C_2^1 & = 0.250 & \text{pd}_2^1 & = 0.500 & \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_3 & = 2.000 & D_3 & = 2.000 & V_3 = \{v_3^1\} & \text{prio}_3 = 3 & \boxed{\text{proc}_3 = 2} \\ C_3^1 & = 0.250 & \text{pd}_3^1 & = 0.500 & \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{array}$$

$$\begin{array}{llll} T_4 & = 2.000 & D_4 & = 2.000 & V_4 = \{v_4^1\} & \text{prio}_4 = 2 & \boxed{\text{proc}_4 = 2} \\ C_4^1 & = 0.250 & \text{pd}_4^1 & = 0.500 & \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_5 & = 2.250 & D_5 & = 2.250 & V_5 = \{v_5^1, v_5^2\} & \text{prio}_5 = 1 & \boxed{\text{proc}_5 = 2} \\ C_5^1 & = 0.500 & \text{pd}_5^1 & = 1.000 & \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ C_5^2 & = 0.125 & \text{pd}_5^2 & = 0.500 & \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

Execution requirement for each segment of each task

$$\begin{array}{llll} T_1 = 1.500 & D_1 = 1.500 & V_1 = \{v_1^1\} & \text{prio}_1 = 3 \quad \text{proc}_1 = 1 \\ \boxed{C_1^1 = 0.250} & \text{pd}_1^1 = 0.500 & \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_2 = 2.000 & D_2 = 2.000 & V_2 = \{v_2^1\} & \text{prio}_2 = 2 \quad \text{proc}_2 = 1 \\ \boxed{C_2^1 = 0.250} & \text{pd}_2^1 = 0.500 & \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_3 = 2.000 & D_3 = 2.000 & V_3 = \{v_3^1\} & \text{prio}_3 = 3 \quad \text{proc}_3 = 2 \\ \boxed{C_3^1 = 0.250} & \text{pd}_3^1 = 0.500 & \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\} \end{array}$$

$$\begin{array}{llll} T_4 = 2.000 & D_4 = 2.000 & V_4 = \{v_4^1\} & \text{prio}_4 = 2 \quad \text{proc}_4 = 2 \\ \boxed{C_4^1 = 0.250} & \text{pd}_4^1 = 0.500 & \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

$$\begin{array}{llll} T_5 = 2.250 & D_5 = 2.250 & V_5 = \{v_5^1, v_5^2\} & \text{prio}_5 = 1 \quad \text{proc}_5 = 2 \\ \boxed{C_5^1 = 0.500} & \text{pd}_5^1 = 1.000 & \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\} \\ \boxed{C_5^2 = 0.125} & \text{pd}_5^2 = 0.500 & \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\} \end{array}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

Speed as function of co-runners

$$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$

$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$

$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$

$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$

$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$

$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$

$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

Taskset example

$$|\tau| = 5 \quad \text{procs}(\Pi) = \{P_1, P_2\}$$

$$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$

$$C_1^1 = 0.250 \quad \text{pd}_1^1 = 0.500 \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$

$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$

$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$

$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$

$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$

$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

If segment 1 of task 1 executes in parallel with segment 1 of task 4, then Segment 1 of task 1 executes with speed 0.5.

Taskset example

If segment 1 of task 1 executes in parallel with a segmentset for which information in CO is not available, then segment 1 of task 1 executes with speed 0.5.

$$|\tau| = 5$$

$$\text{procs}(\Pi) = \{P_1, P_2\}$$

$$T_1 = 1.500 \quad D_1 = 1.500 \quad V_1 = \{v_1^1\} \quad \text{prio}_1 = 3 \quad \text{proc}_1 = 1$$

$$C_1^1 = 0.250 \quad \boxed{\text{pd}_1^1 = 0.500} \quad \text{CO}_1^1 = \{(\{v_3^1\}, 1.0), (\{v_4^1\}, 0.5), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_2 = 2.000 \quad D_2 = 2.000 \quad V_2 = \{v_2^1\} \quad \text{prio}_2 = 2 \quad \text{proc}_2 = 1$$

$$C_2^1 = 0.250 \quad \text{pd}_2^1 = 0.500 \quad \text{CO}_2^1 = \{(\{v_3^1\}, 0.5), (\{v_4^1\}, 1.0), (\{v_5^1\}, 1.0), (\{v_5^2\}, 1.0)\}$$

$$T_3 = 2.000 \quad D_3 = 2.000 \quad V_3 = \{v_3^1\} \quad \text{prio}_3 = 3 \quad \text{proc}_3 = 2$$

$$C_3^1 = 0.250 \quad \text{pd}_3^1 = 0.500 \quad \text{CO}_3^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 0.5)\}$$

$$T_4 = 2.000 \quad D_4 = 2.000 \quad V_4 = \{v_4^1\} \quad \text{prio}_4 = 2 \quad \text{proc}_4 = 2$$

$$C_4^1 = 0.250 \quad \text{pd}_4^1 = 0.500 \quad \text{CO}_4^1 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

$$T_5 = 2.250 \quad D_5 = 2.250 \quad V_5 = \{v_5^1, v_5^2\} \quad \text{prio}_5 = 1 \quad \text{proc}_5 = 2$$

$$C_5^1 = 0.500 \quad \text{pd}_5^1 = 1.000 \quad \text{CO}_5^1 = \{(\{v_1^1\}, 1.0), (\{v_2^1\}, 1.0)\}$$

$$C_5^2 = 0.125 \quad \text{pd}_5^2 = 0.500 \quad \text{CO}_5^2 = \{(\{v_1^1\}, 0.5), (\{v_2^1\}, 1.0)\}$$

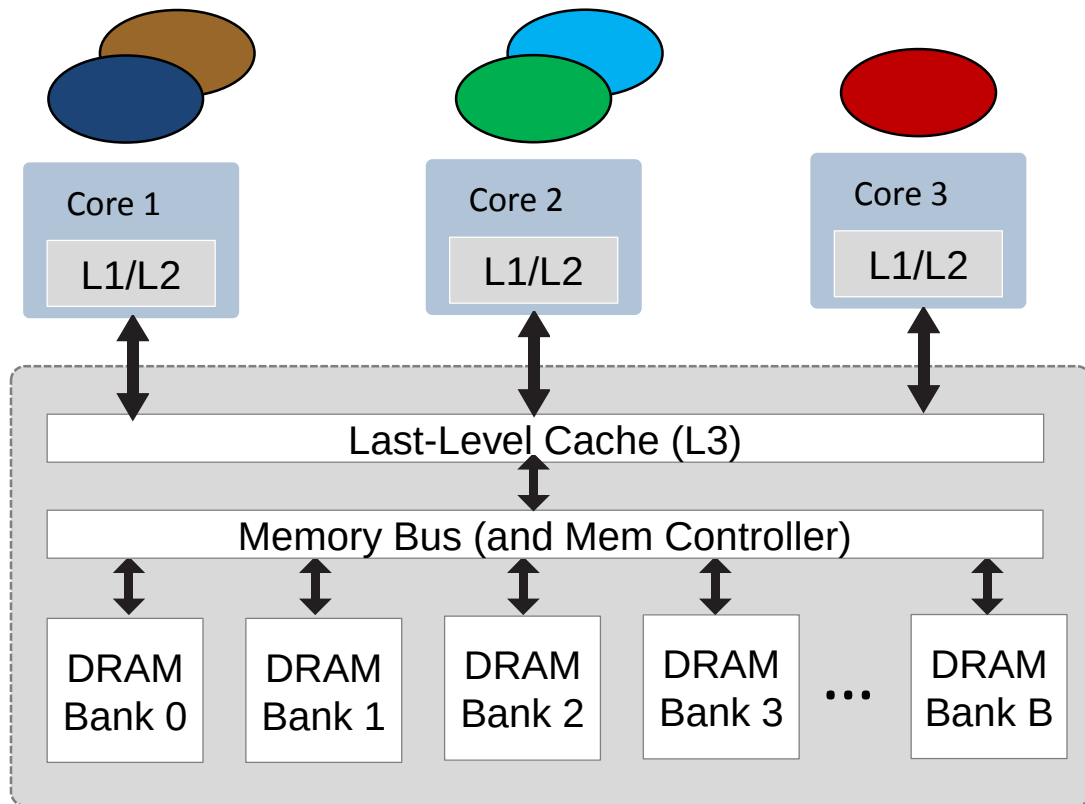
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware? (DONE)

Q2: How to create a schedulability analysis that uses this abstraction?

Before discussing them, let us discuss:

1. Other approaches (DONE)
2. General ideas for abstractions in other disciplines (DONE)

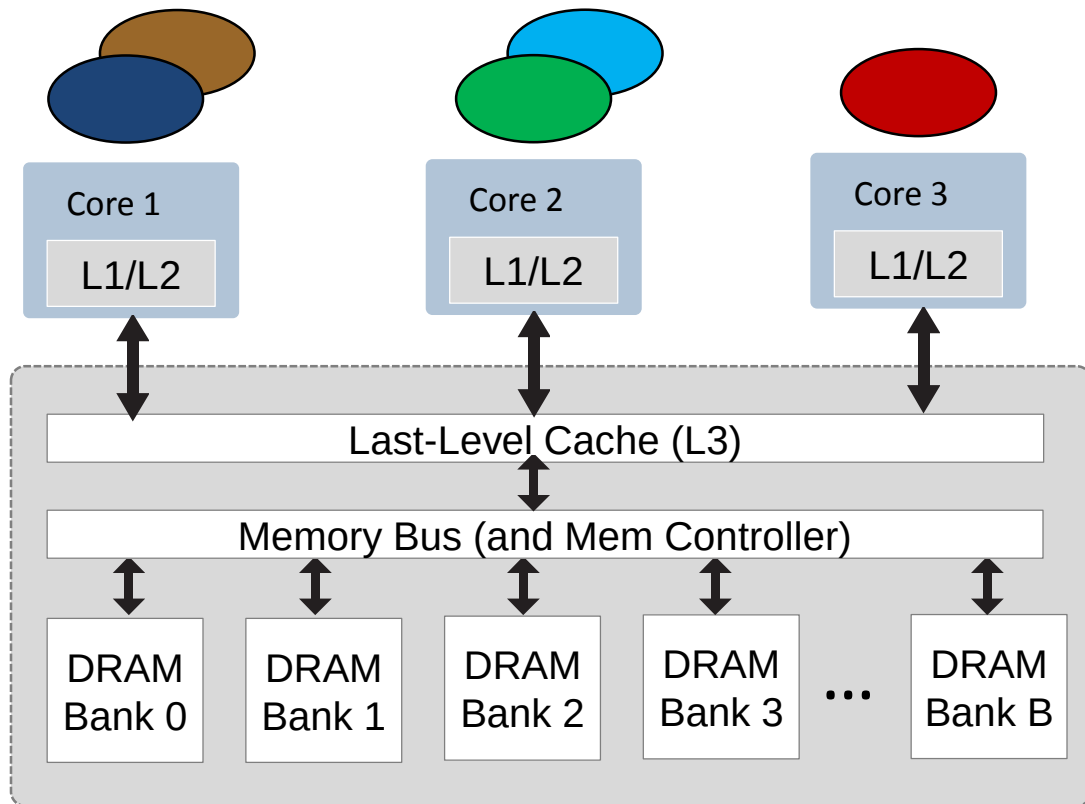


Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

Q2: How to create a schedulability analysis that uses this abstraction?

Let us discuss Q2.

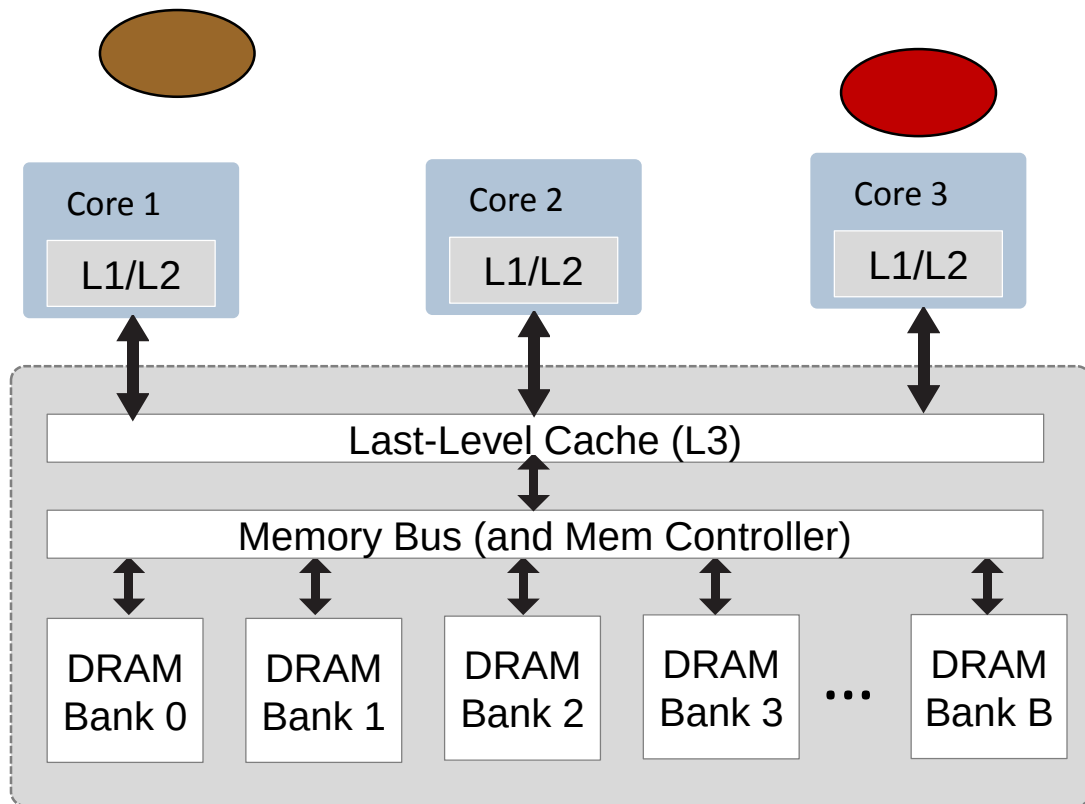


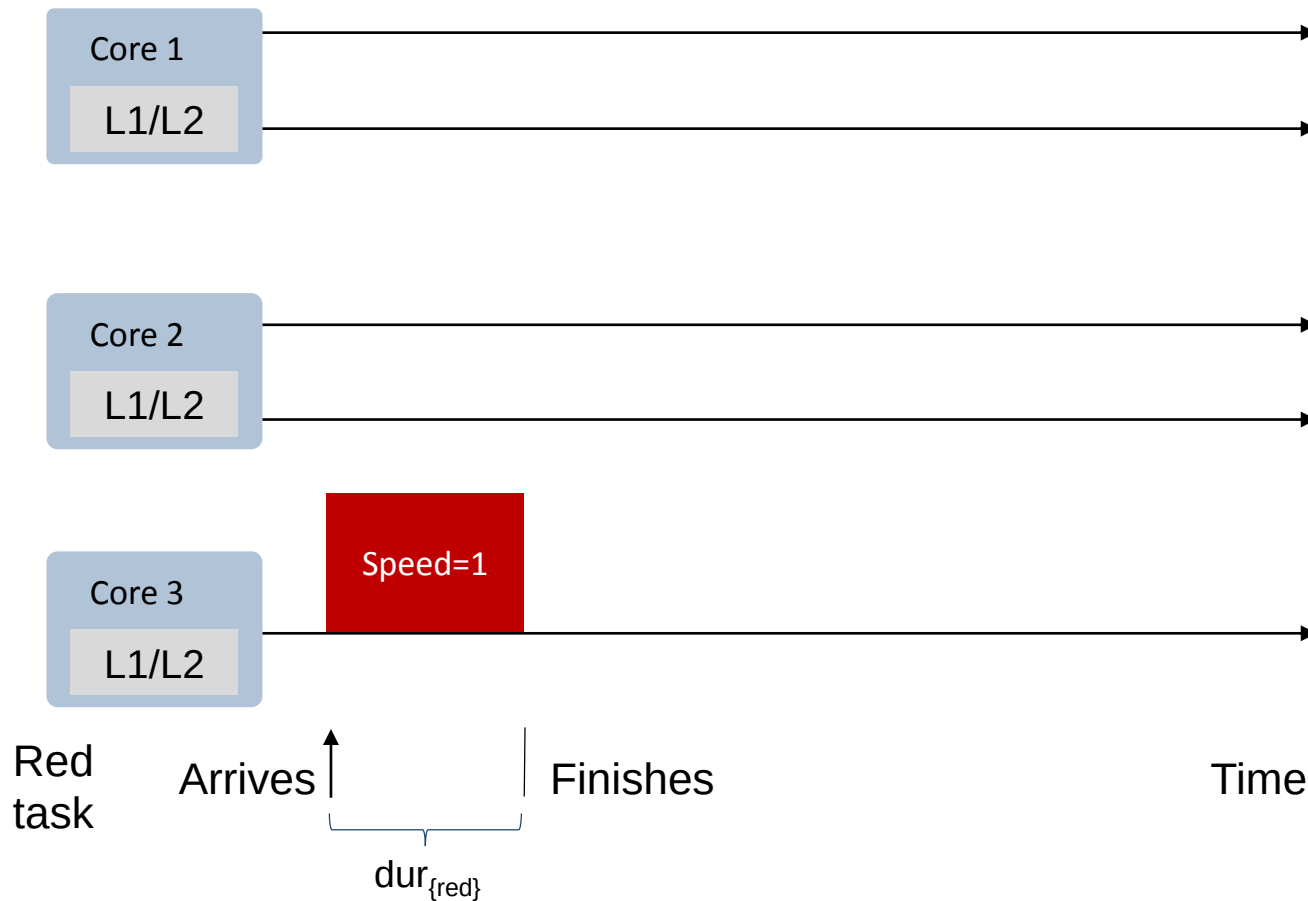
Big Research Questions

Q1: What is a good abstraction that describes the effect of undocumented hardware?

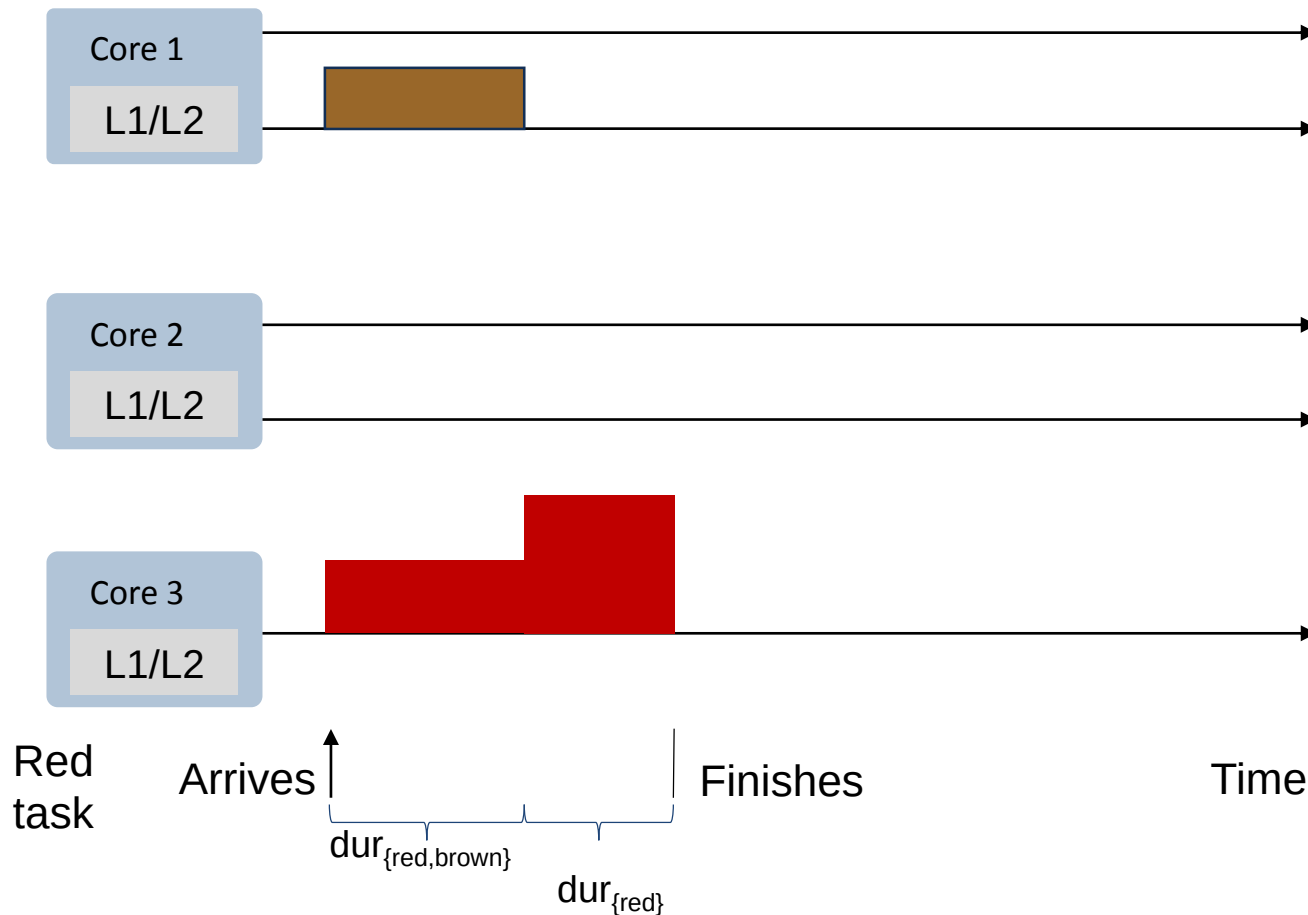
Q2: How to create a schedulability analysis that uses this abstraction?

In order to simplify our discussion initially, let us consider a taskset with only Red and Brown task. Also, let us consider that the minimum inter-arrival time of each of these tasks is infinity (i.e., generates just a single job).



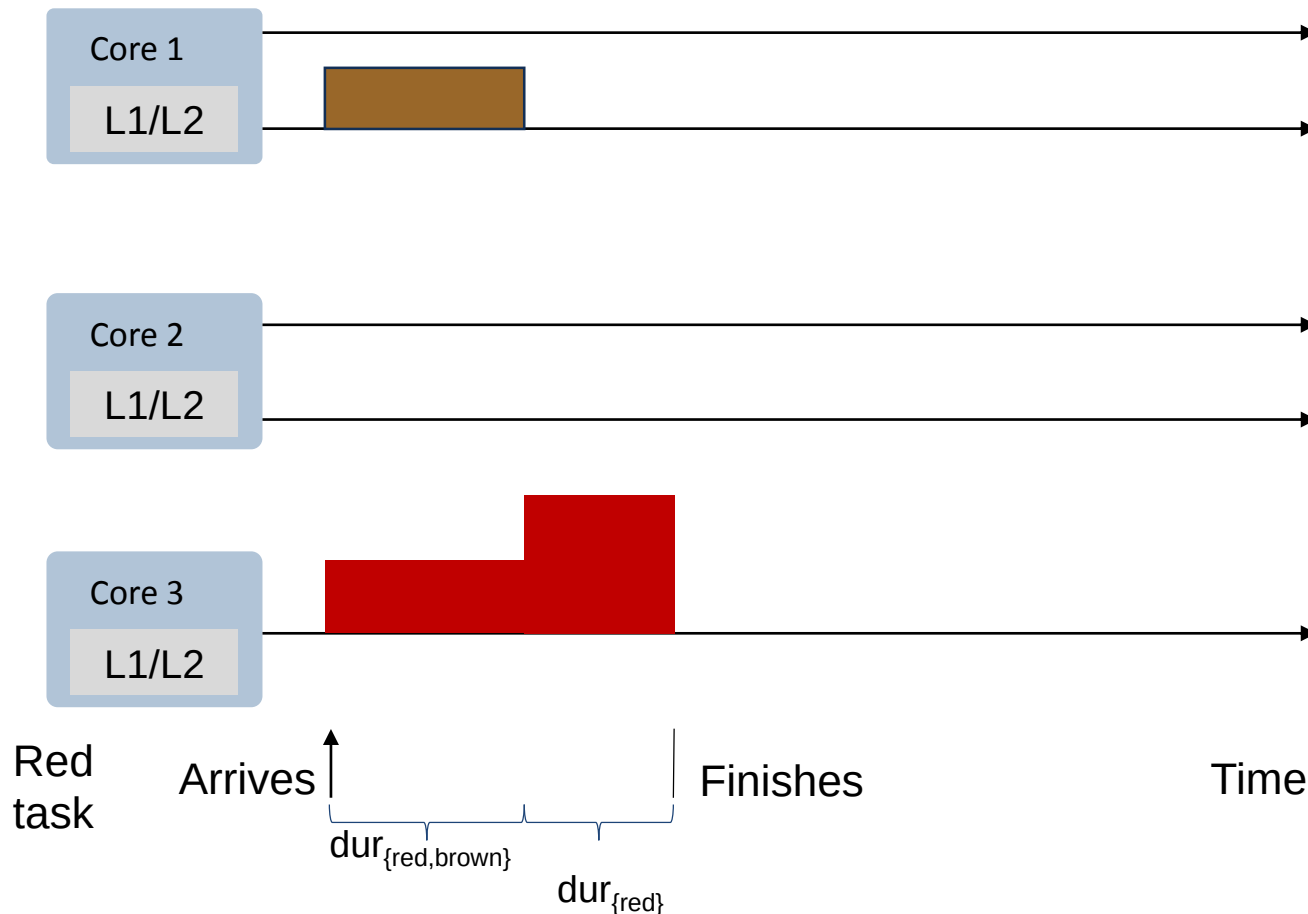


Let $\text{dur}_{\{\text{red}\}}$ denote the cumulative duration that the set of tasks that execute is the set $\{\text{red}\}$.



Let $dur_{\{red\}}$ denote the cumulative duration that the set of tasks that execute is the set $\{red\}$.

Let $dur_{\{red,brown\}}$ denote the cumulative duration that the set of tasks that execute is the set $\{red,brown\}$.



Maximize

$$dur_{\{red,brown\}} +$$

$$dur_{\{red\}}$$

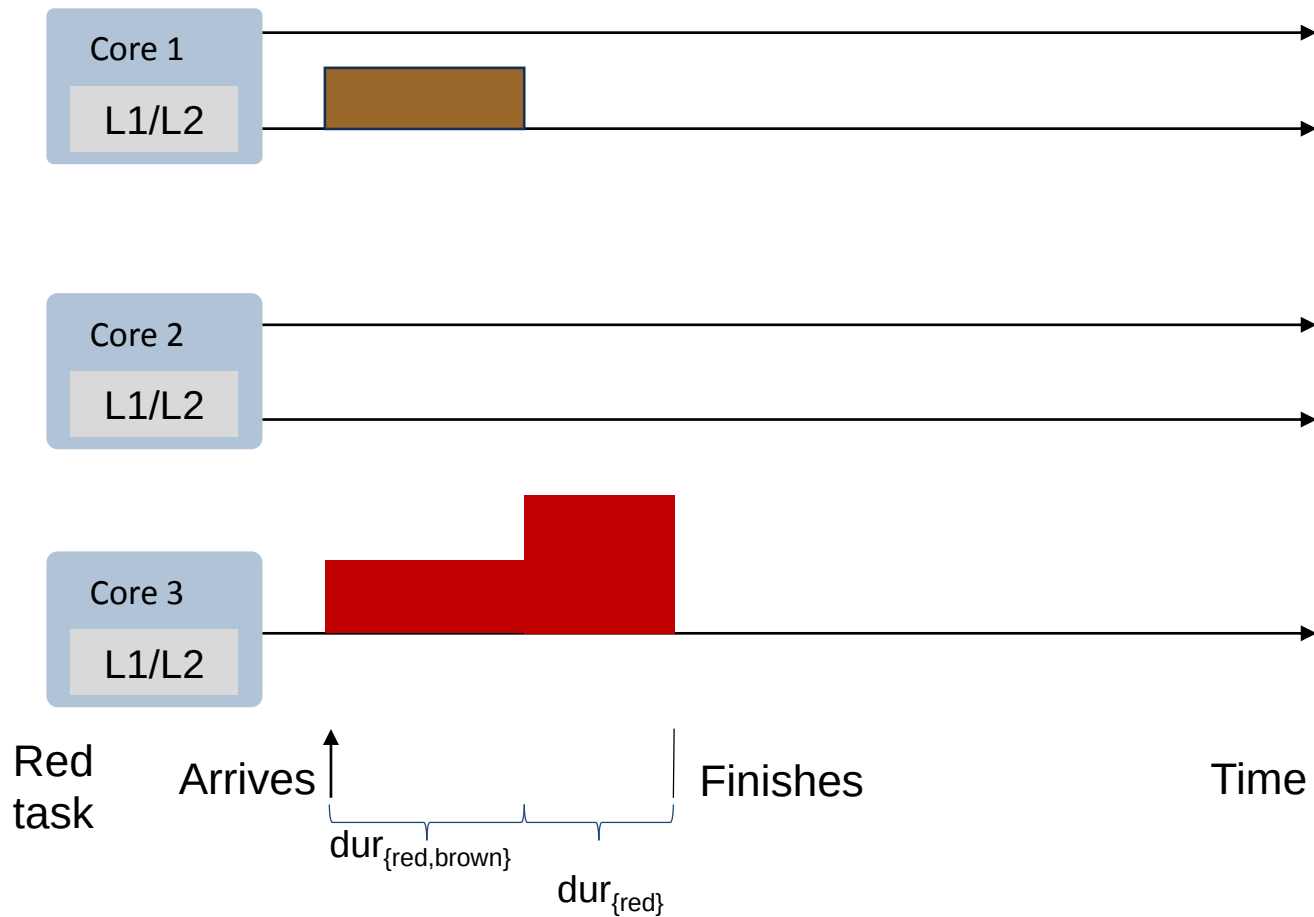
subject to

$$0.45 * dur_{\{red,brown\}} +$$

$$1.00 * dur_{\{red\}} \leq C_{red}$$

$$0.6 * dur_{\{red,brown\}} +$$

$$1.0 * dur_{\{brown\}} \leq C_{brown}$$



Maximize

$$dur_{\{red,brown\}} +$$

$$dur_{\{red\}}$$

subject to

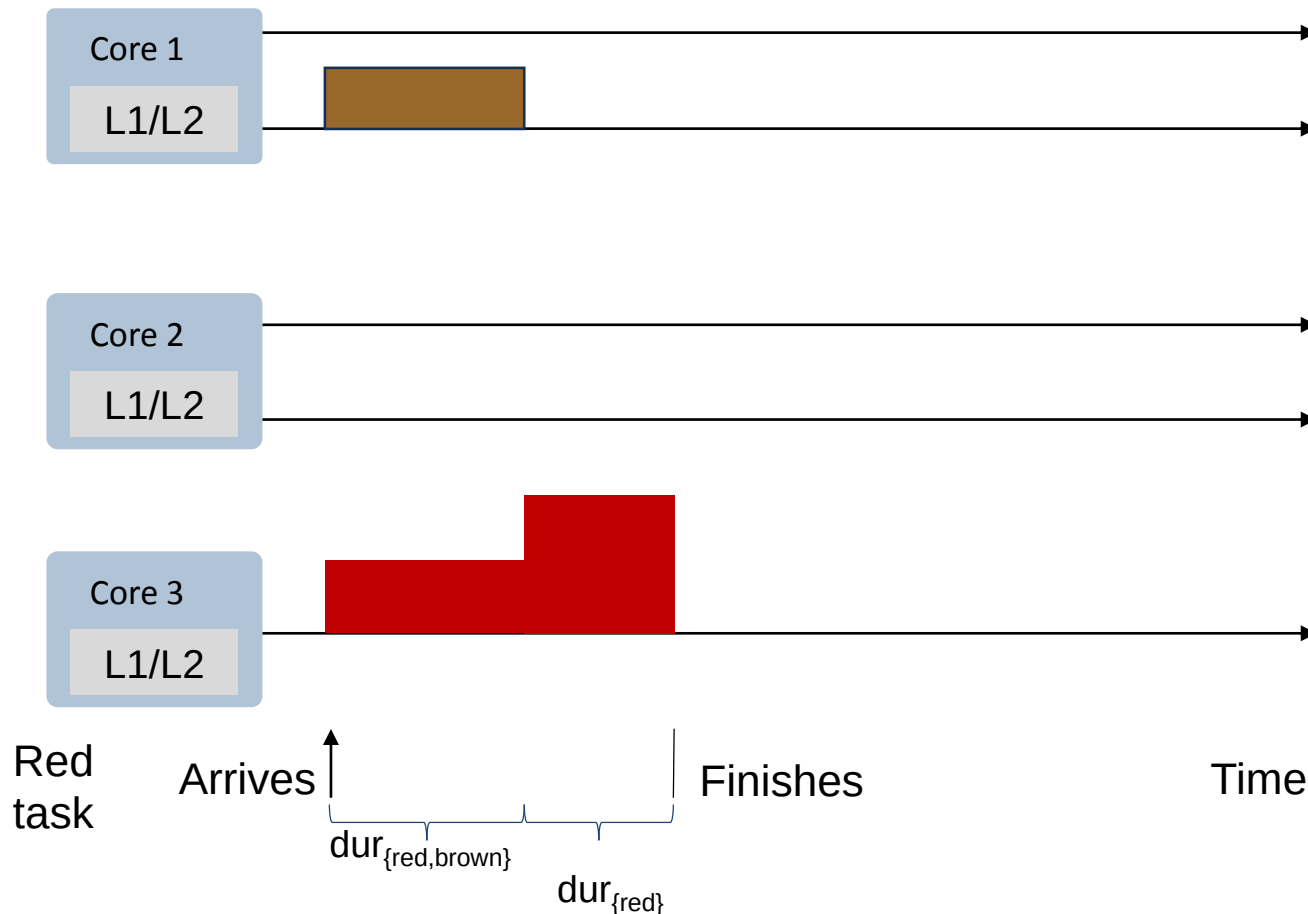
$$0.45 * dur_{\{red,brown\}} +$$

$$1.00 * dur_{\{red\}} \leq C_{red}$$

$$0.6 * dur_{\{red,brown\}} +$$

$$1.0 * dur_{\{brown\}} \leq C_{brown}$$

Lower bound on
the speed of
execution of Red
when it executes
in parallel with
Brown.



Maximize

$$dur_{\{red,brown\}} +$$

$$dur_{\{red\}}$$

subject to

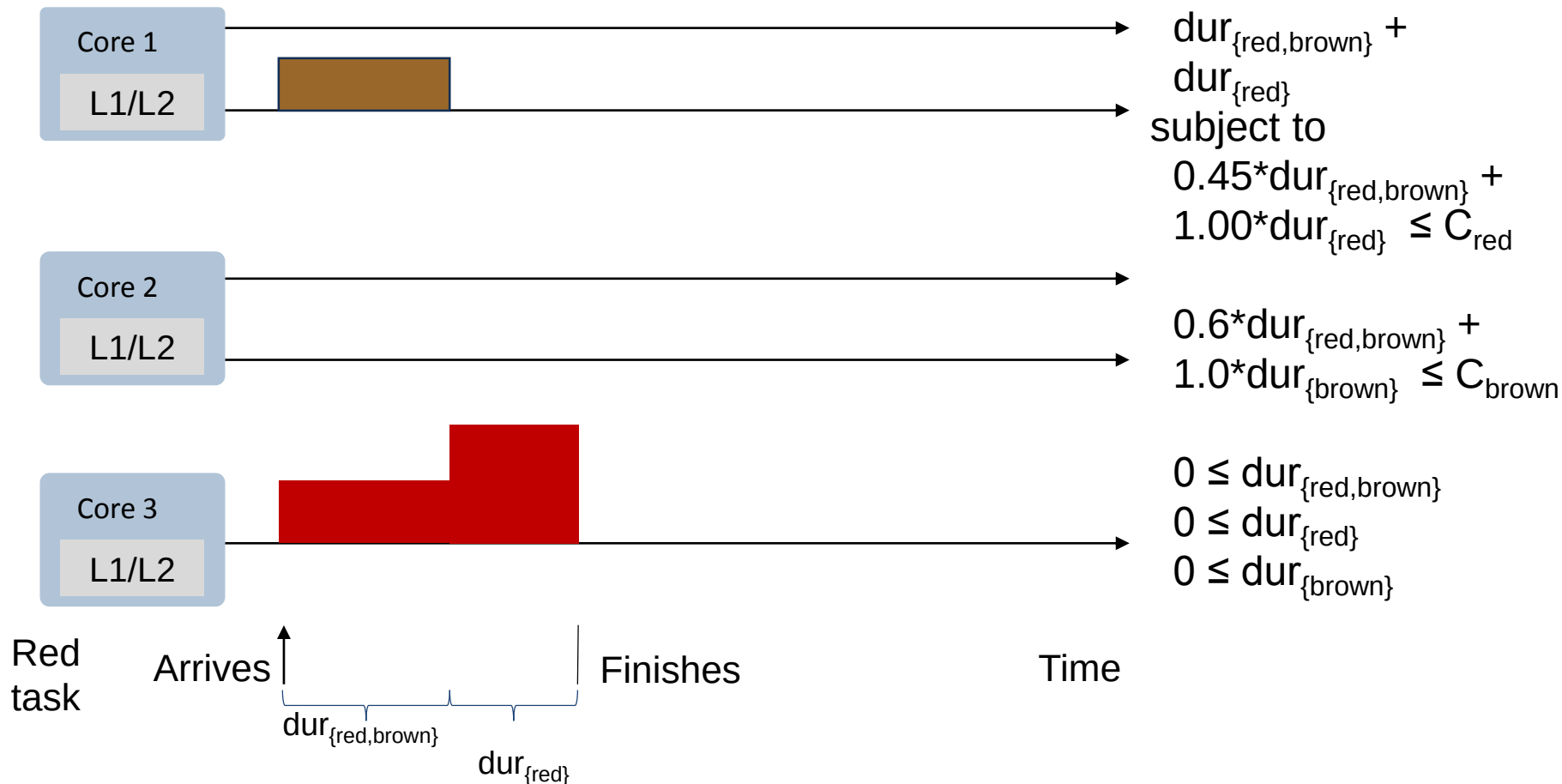
$$0.45 \cdot dur_{\{red,brown\}} +$$

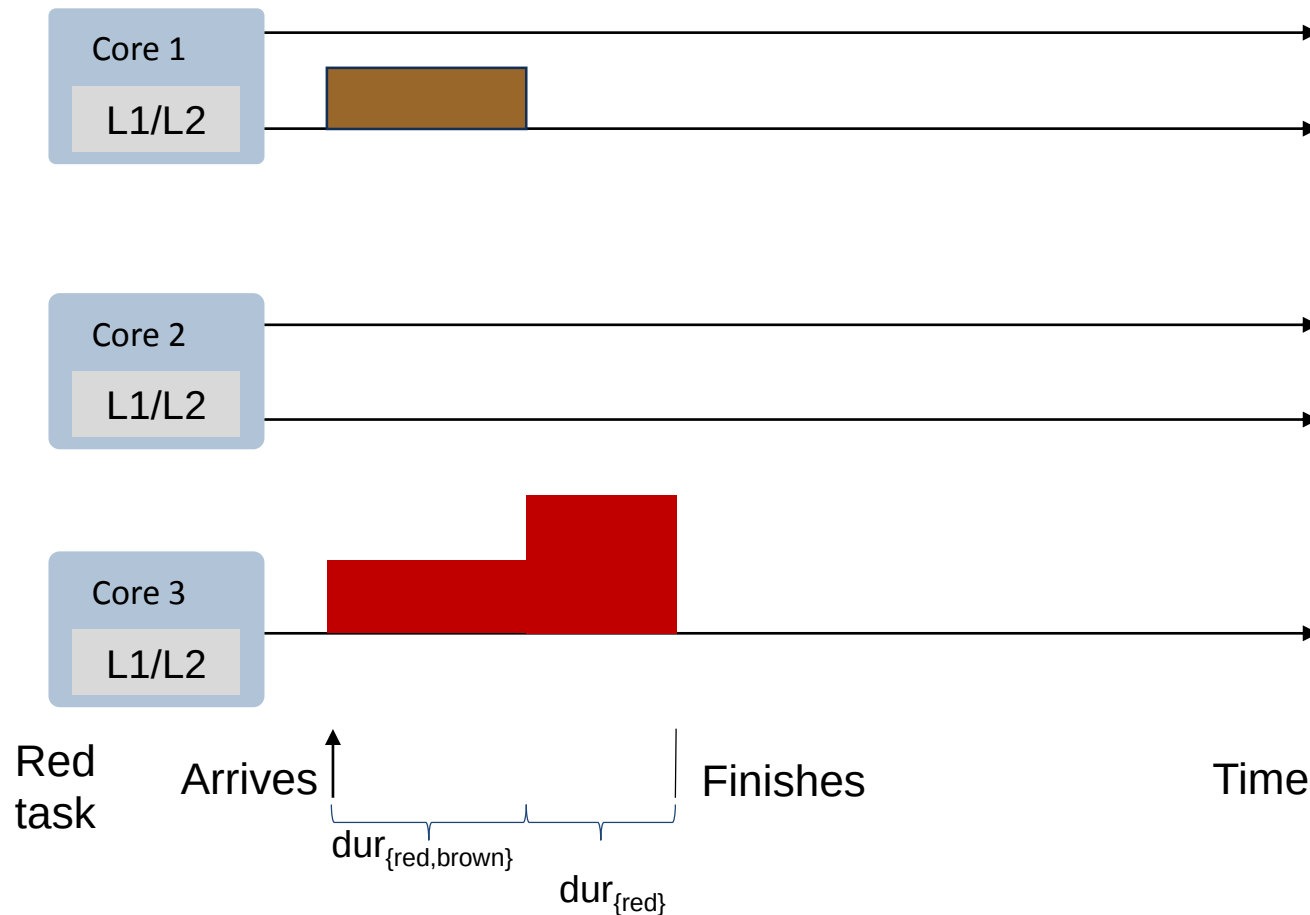
$$1.00 \cdot dur_{\{red\}} \leq C_{red}$$

$$0.6 \cdot dur_{\{red,brown\}} +$$

$$1.0 \cdot dur_{\{brown\}} \leq C_{brown}$$

Lower bound on
the speed of
execution of
Brown
when it executes
in parallel with
Red.





Maximize

$$dur_{\{red,brown\}} +$$

$$dur_{\{red\}}$$

subject to

$$0.45 \cdot dur_{\{red,brown\}} +$$

$$1.00 \cdot dur_{\{red\}} \leq C_{red}$$

$$0.6 \cdot dur_{\{red,brown\}} +$$

$$1.0 \cdot dur_{\{brown\}} \leq C_{brown}$$

$$0 \leq dur_{\{red,brown\}}$$

$$0 \leq dur_{\{red\}}$$

$$0 \leq dur_{\{brown\}}$$

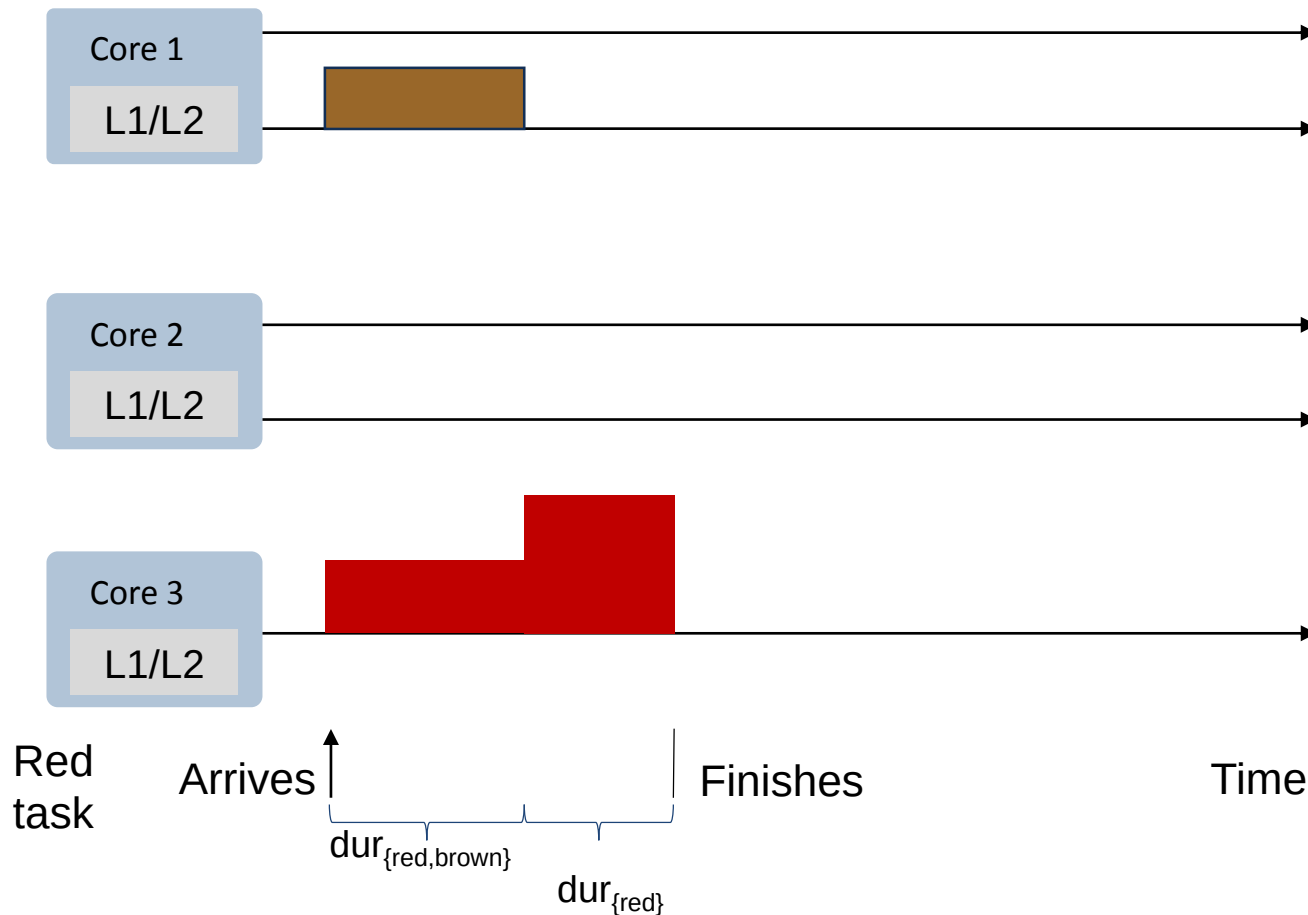
Solving this

optimization problem

yields an upper bound

on the response time

of Red.



Maximize

$$dur_{\{red,brown\}} +$$

$$dur_{\{red\}}$$

subject to

$$0.45 * dur_{\{red,brown\}} +$$

$$1.00 * dur_{\{red\}} \leq C_{red}$$

$$0.6 * dur_{\{red,brown\}} +$$

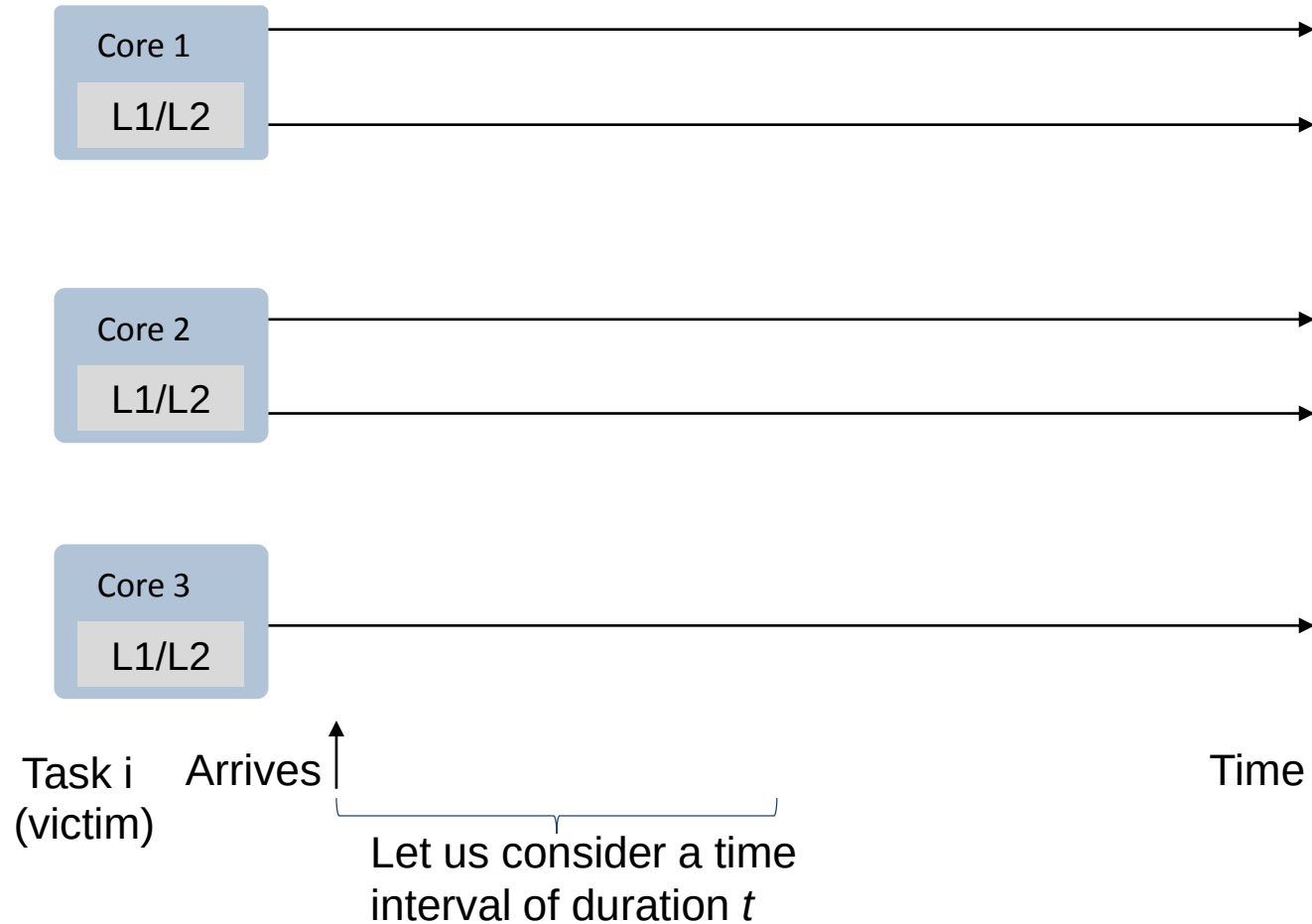
$$1.0 * dur_{\{brown\}} \leq C_{brown}$$

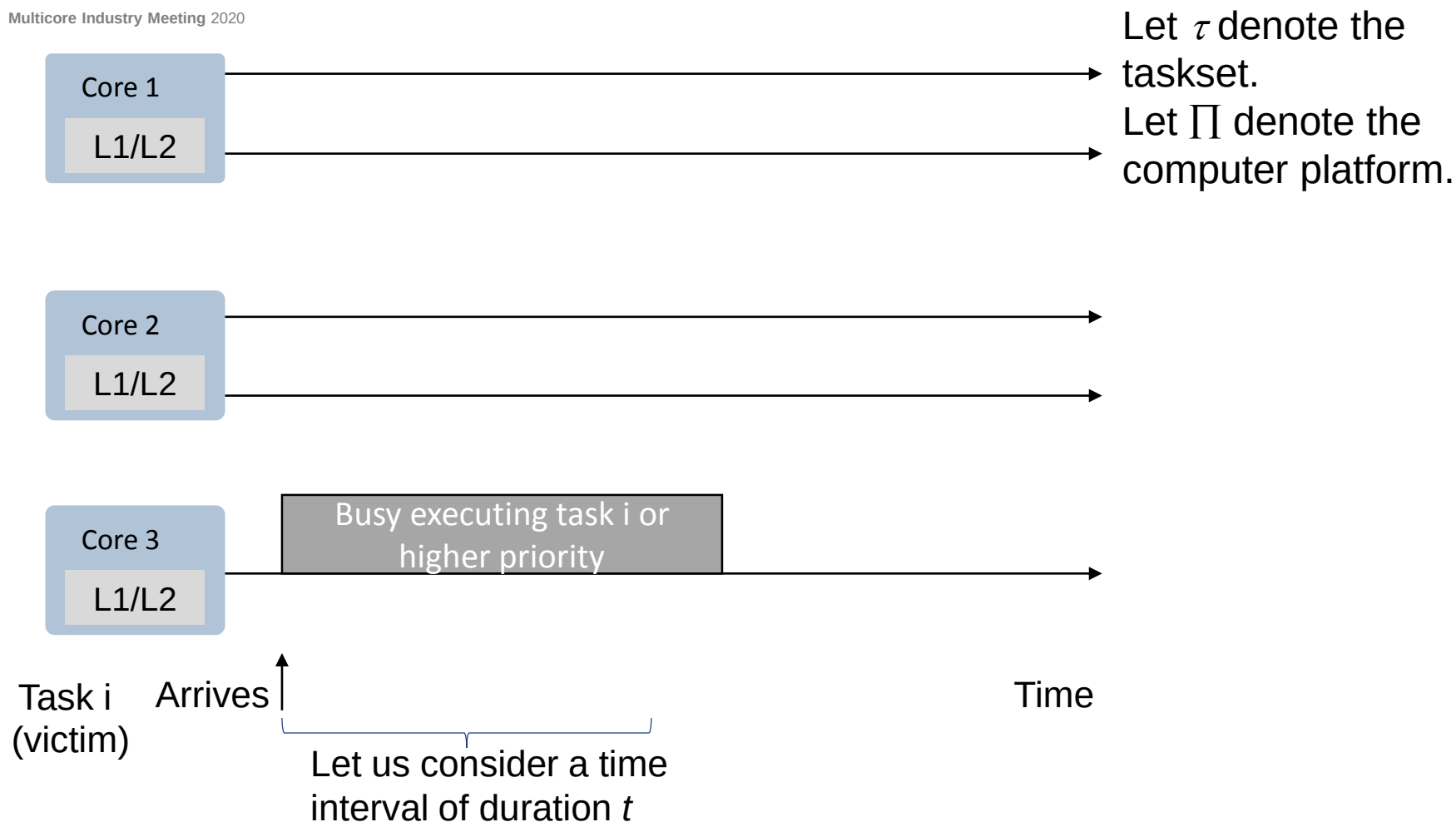
$$0 \leq dur_{\{red,brown\}}$$

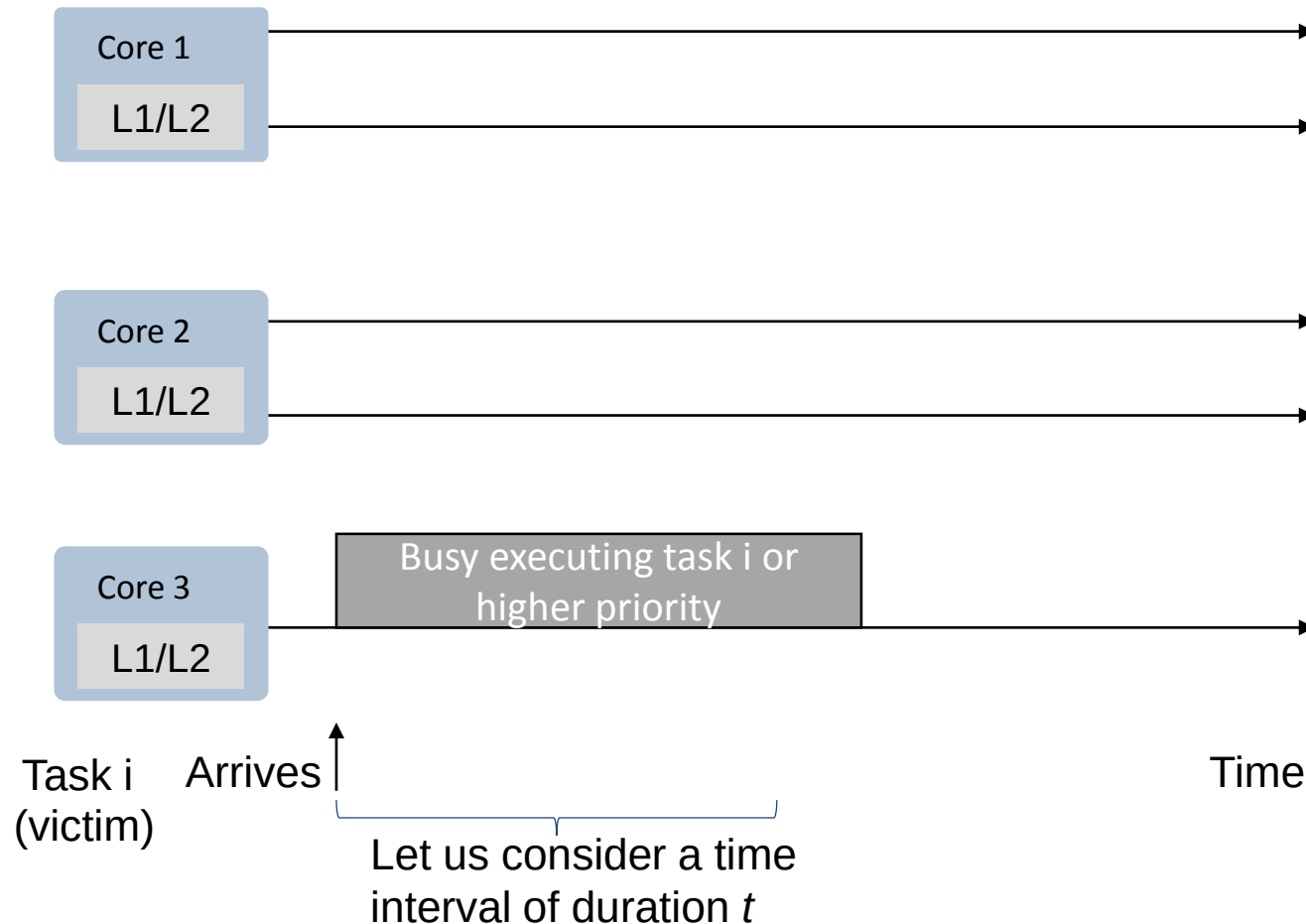
$$0 \leq dur_{\{red\}}$$

$$0 \leq dur_{\{brown\}}$$

Let us formulate this in general.

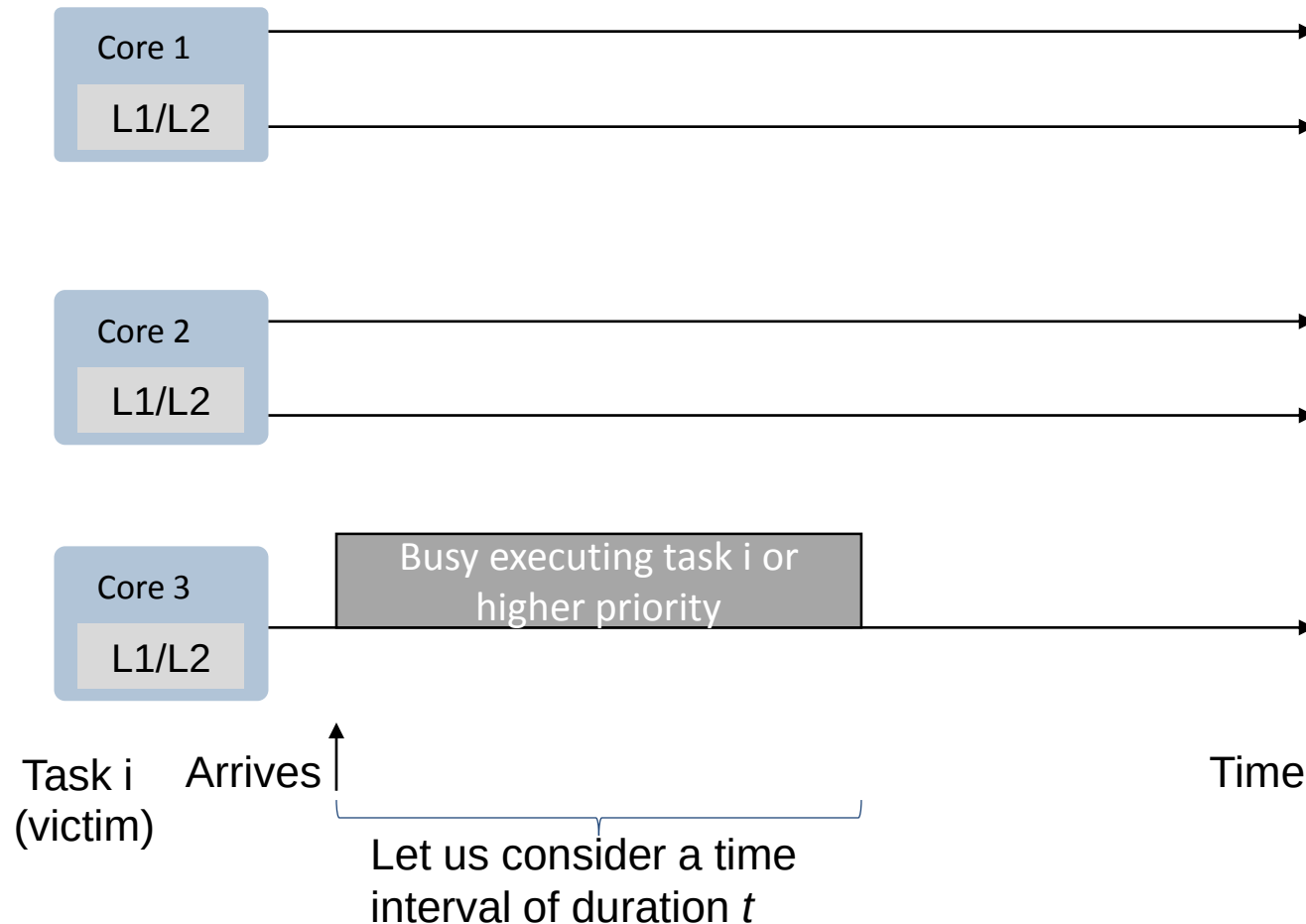






Let τ denote the taskset.
Let Π denote the computer platform.

Given task i , find a value of t such that an upper bound on the duration of time during which the processor (Processor 3) is busy executing task i or higher priority tasks is equal to t .



Let τ denote the taskset.
Let Π denote the computer platform.

Given task i , find a value of t such that an upper bound on the duration of time during which the processor (Processor 3) is busy executing task i or higher priority tasks is equal to t .

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \setminus \{\langle i', k' \rangle\} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \setminus \{\langle i', k' \rangle\} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\begin{aligned} & \text{maximize} && \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s \end{aligned}$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \setminus \{\langle i', k' \rangle\} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \setminus \{\langle i', k' \rangle\} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

For a given taskset, for a given t , this is a linear program.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \times du_s \leq \left\lceil \frac{t}{T_{i'}} \right\rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

For a given taskset, for a given t , this is a linear program because these become constants.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\begin{aligned} & \text{maximize} && \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s \end{aligned}$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \setminus \{\langle i', k' \rangle\} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \setminus \{\langle i', k' \rangle\} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

A linear program can be solved in polynomial time.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\begin{aligned} & \text{maximize} && \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s \end{aligned}$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

Given τ, Π, i, t ,
evaluating $\text{reqlp}(\tau, \Pi, i, t)$
can be done in
polynomial time.

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\begin{aligned}
 & \text{maximize} && \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s \\
 & \text{subject to} && \\
 & \forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \times \{ \langle i', k' \rangle \} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'} \\
 & \forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t) \\
 & \forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}
 \end{aligned}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

Evaluating this can be done in pseudo-polynomial time.

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\begin{aligned}
 & \text{maximize} && \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s \\
 & \text{subject to} && \\
 & \forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s \setminus \{\langle i', k' \rangle\}}^{k'} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'} \\
 & \forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s \setminus \{\langle i', k' \rangle\}}^{k'} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t) \\
 & \forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}
 \end{aligned}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

This is a generalization of the classic response-time analysis for Rate-Monotonic.

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\begin{aligned} & \text{maximize} && \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s \end{aligned}$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s}^{k'} \times \{ \langle i', k' \rangle \} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s}^{k'} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

For details (about this theorem and other results about the model), see

B. Andersson et al., "Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times," ACM TECS, 2018.

Define $\text{reqlp}(\tau, \Pi, i, t)$ as the optimal value of the objective function of the following:

$$\text{maximize} \quad \sum_{i' \in \text{hep}(\tau, \Pi, i)} \sum_{v_{i'}^{k'} \in V_{i'}} \sum_{s \in S(\tau, \Pi, i', k')} du_s$$

subject to

$$\forall i' \in \text{hep}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k')} \text{pw}_{i', s \setminus \{(i', k')\}}^{k'} \times du_s \leq \lceil \frac{t}{T_{i'}} \rceil \times C_{i'}^{k'}$$

$$\forall i' \in \text{top}(\tau, \Pi, i), \forall v_{i'}^{k'} \in V_{i'}, \sum_{s \in S(\tau, \Pi, i', k') \wedge (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s))} \text{pw}_{i', s \setminus \{(i', k')\}}^{k'} \times du_s \leq \text{xUB}(\tau, \Pi, i', k', t)$$

$$\forall s \in S(\tau, \Pi) \text{ s.t. } (\exists \langle i'', k'' \rangle \text{ s.t. } (i'' \in \text{hep}(\tau, \Pi, i)) \wedge (\langle i'', k'' \rangle \in s)), du_s \in \mathbb{R}_{\geq 0}$$

Schedulability analysis (timing verification) is done as follows:

THEOREM 4.2. $(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t)) \Rightarrow \tau$ is schedulable on Π

We have a schedulability analysis for tasks with co-runner dependent execution times.

Comparing this result to classic Rate-Monotonic Analysis

Work	Model (Execution Time Depends on Corunners)	Time Complexity of Exact Schedulability Analysis	Formulation of Schedulability Test	Time Complexity of Schedulability Test
Previous work	No	$\leq$ Pseudo- polynomial	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i],$ $\sum_{j \in \text{hep}(\tau, \Pi, i)} \lceil \frac{t}{T_j} \rceil C_j \leq t))$ $\Leftrightarrow$ schedulable	Pseudo-polynomial
This article	Yes	Co-NP-hard in the strong sense	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i],$ $\text{reqlp}(\tau, \Pi, i, t) \leq t))$ $\Rightarrow$ schedulable	Pseudo-polynomial if the number of processors in Π is fixed

This result generalizes classic Rate-Monotonic Analysis to undocumented multicore

Work	Model (Execution Time Depends on Corunners)	Time Complexity of Exact Schedulability Analysis	Formulation of Schedulability Test	Time Complexity of Schedulability Test
Previous work	No	$\leq$ Pseudo- polynomial	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \sum_{j \in \text{hep}(\tau, \Pi, i)} \lceil \frac{t}{T_j} \rceil C_j \leq t))$ $\Leftrightarrow$ schedulable	Pseudo-polynomial
This article	Yes	Co-NP-hard in the strong sense	$(\forall \tau_i \in \tau, (\exists t \in [0, D_i], \text{reqlp}(\tau, \Pi, i, t) \leq t))$ $\Rightarrow$ schedulable	Pseudo-polynomial if the number of processors in Π is fixed

This looks very
theoretical. Does this
really work in reality?



I understand that for each task, the set of co-runner sets is polynomial in the number of tasks assuming that the number of processors is fixed.



But I still wonder how
one can obtain the
lower bound of speed
of execution of one task
given co-runners.





A measurement-based approach:

5.2 Obtaining Taskset Parameters for a Given Software System

Every scheduling theory that provides 100% guarantees on timing relies on a model. Traditional scheduling theory relies on knowledge of an upper bound on the execution time of a program, and therefore, the research community has developed methods for finding an upper bound on the execution requirement of a program when the program runs in isolation on a single-core processor (Wilhelm et al. 2006). One can distinguish between (1) *static methods* that take the program as input and compute an upper bound without running the program, (2) *hybrid methods* that measure execution times of parts of a program and then use static methods to compute an upper bound for the entire program, and (3) *dynamic methods* where a software practitioner uses domain knowledge to identify a set of worst-case inputs/initial states of the program and then measures the execution time of the program for these inputs/initial states; sometimes the program is run multiple times with variations of inputs/initial state and sometimes a safety margin is added based on engineering judgment. The dynamic method is in common use in industry today.

The model used in this article, however, also requires that a task/segment is described with a lower bound on its speed as a function of consumers. Unfortunately, the current state of the art does not offer any method for obtaining such a lower bound while considering all the complexities of modern memory systems. Therefore, in order to use our scheduling theory (in practice and also in our case study/validation in Section 5.3), we need to develop a method for obtaining such a lower bound. We will do so by using method (3) mentioned above but extending it so that it also provides a lower bound on its speed as a function of consumers. The new method is shown below as pseudo-code:

- (1) Let SS be an integer indicating the sample size used in our experimental method. Assign a value to SS (for example $SS := 100$).
- (2) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do
 - (a) Run a job of task τ_i on the computer platform s.t. no other tasks execute on this platform.
 - (b) Repeat the measurement SS times and let CM_i^k denote the set of these measurements (execution times from measurements of a task executed in isolation).
- (3) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do $C_i^k := \max_{v_i^k \in V_i} x$.
- (4) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do
 - (a) Find an upper bound on the number of memory accesses performed by a segment of a job of task τ_i . (This can be obtained using performance monitoring counters.) Run these measurements SS times. Let H_i^k denote the largest measurement.
 - (b) Find an upper bound on the time for a single memory access assuming that we do not know the consumers. Let MA denote this (e.g., MA could be 500 nanoseconds).
 - (c) $po_i^k := \max_{v_i^k \in V_i} \frac{H_i^k}{2 \cdot n_i^k \cdot MA}$.
- (5) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, do $CO_i^k := \emptyset$.
- (6) For each task $\tau_i \in \tau$, for each $v_i^k \in V_i$, for each $s \in S(\tau, \Pi, I, k)$, do
 - (a) $co := s \setminus \{(i, k)\}$.
 - (b) For those segments in co , keep running all of them continuously (i.e., if segment v_j^l is in co , then when v_i^k has finished execution, let this segment v_j^l start execution again immediately).
 - (c) While the segments in (b) execute continuously, execute a single job of segment v_i^k of task τ_i and measure the execution time of v_i^k . Repeat this measurement SS times and let $CM_{i,co}^k$ denote the set of measurements (execution times from measurements of a task executed with co-runners).

ACM Transactions on Embedded Computing Systems, Vol. 15, No. 3, Article 71. Publication date: May 2016.

OK, I buy that you can obtain these lower bounds on the speed of execution through measurements. But I wonder how trustworthy these numbers are. Have you done any validation?



OK, I buy that you can obtain these lower bounds on the speed of execution through measurements. But I wonder how trustworthy these numbers are. Have you done any validation?



You could do a case study. Choose a set of tasks and use your measurement-based approach to obtain speed as a function of co-runners. Now, you have a taskset. Then run your schedulability analysis on this taskset.

OK, I buy that you can obtain these lower bounds on the speed of execution through measurements. But I wonder how trustworthy these numbers are. Have you done any validation?



Then run the system and measure response times of each task. If you observe one case where the observed response time $>$ calculated upper bound on response time, then you have falsified your theory; otherwise, you have corroborated your theory.



5.3 Validation of Model and Schedulability Analysis

We want to compare the computed upper bound on the response time of a task against the measured response times of jobs of this task in order to find out if there is any case in practice where the measured response time of this task exceeds the upper bound on the response time of this task. If so, the schedulability analysis would be unsafe. To get insight into this question, we pursue a case study with a specific taskset and a real embedded platform (ARM Cortex-A9 quad-core processor). To avoid cache-related preemption delay, we used the software cache partitioning implementation of Linus/BK [Kim et al. 2013]² and assigned a private cache partition to each task.

We constructed a taskset as follows. First, we created six synthetic programs with different intensities of memory accesses on the target platform. We call each such program a *building block*. There is no shared variable between building blocks. We characterize the execution time and coroutine interference using the method in the previous section with $SS = 100$. The coroutine description is complete; that is, the building block is characterized for each possible set of coroutines.

Then we create tasks from these building blocks as follows: Task τ_1 consists of executing building block 1. Hence, we model task τ_1 as having a single segment. Task τ_2 consists of executing building block 2 and then executing building block 3. Hence, we model task τ_2 as having two segments. Task τ_3 consists of executing building block 4. Hence, we model task τ_3 as having a single segment. Task τ_4 consists of executing building block 5. Hence, we model task τ_4 as having a single segment. Task τ_5 consists of executing building block 6. Hence, we model task τ_5 as having a single segment. Task τ_6 consists of executing building block 2 and then executing building block 6. Hence, we model task τ_6 as having two segments.

We assign tasks to processors and assign priorities to tasks and assign T and D parameters and obtain the execution time and coroutine interference parameters from the building blocks. This gives us the taskset specified in Table 5. The CO parameters are not shown because they consume lots of space; these parameters are available at the end of the source-code file of the tool that performs schedulability testing.

Our tool yields the following upper bounds on response times of task: 0.113 for τ_1 , 0.554 for τ_2 , 0.132 for τ_3 , 0.151 for τ_4 , 0.146 for τ_5 , and 0.407 for τ_6 (units in seconds).

We run the system for 1,000 seconds and measure the response times of jobs. For task τ_6 , we record the maximum response time of a job of this task and let r_6 denote it. From the experiment, we obtain $r_1 = 0.107$, $r_2 = 0.483$, $r_3 = 0.527$, $r_4 = 0.137$, $r_5 = 0.142$, $r_6 = 0.365$ (units in seconds). With these figures, we obtain that the response time overestimates the observed response times by less than 15%. From this experiment, we see that (1) no deadline was missed and (2) the observed response times were close to the computed upper bound.

From B. Andersson et al., ``Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times," ACM TECS, 2018:

With these figures, we obtain that the response time overestimates the observed response times by less than 15%. From this experiment, we see that (1) no deadline was missed and (3) the observed response times were close to the computed upper bound.

Conclusion: It is possible to analyze timing of software executing on undocumented multicore.

Thanks!

Backup slides

A Look at Our Analysis Tool - 1

Schedulability analysis of tasks with co-runner dependent execution times

This program implements the schedulability test in B. Andersson et al., "Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times," ACM TECS, 2018.

Number of tasks 5 Number of processors 2

	Minimum inter-arrival time	Deadline	Number of segments	Priority	Processor	Execution requirement	Default speed	Co-runner specification
Task 1	1.5	1.5	1	3	1			
Segment 1						0.25	0.5	[[[3, 1]], 1.0], [[4, 1]], 0.5], [[5, 1]], 1.0], [[5, 2]], 1.0]]
Segment 2								
Segment 3								
Task 2	2.0	2.0	1	2	1			
Segment 1						0.25	0.5	[[[3, 1]], 0.5], [[4, 1]], 1.0], [[5, 1]], 1.0], [[5, 2]], 1.0]]
Segment 2								
Segment 3								
Task 3	2.0	2.0	1	3	2			
Segment 1						0.25	0.5	[[[1, 1]], 1.0], [[2, 1]], 0.5]]
Segment 2								
Segment 3								
Task 4	2.0	2.0	1	2	2			
Segment 1						0.25	0.5	[[[1, 1]], 0.5], [[2, 1]], 1.0]]
Segment 2								
Segment 3								
Task 5	2.25	2.25	2	1	2			
Segment 1						0.5	1.0	[[[1, 1]], 1.0], [[2, 1]], 1.0]]
Segment 2						0.125	0.5	[[[1, 1]], 0.5], [[2, 1]], 1.0]]
Segment 3								

Do schedulability analysis

A Look at Our Analysis Tool - 2

Schedulability analysis of tasks with co-runner dependent execution times

This program implements the schedulability test in B. Andersson et al., "Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times," ACM TECS, 2018.

Number of tasks: 5 Number of processors: 2

	Minimum inter-arrival time	Deadline	Number of segments	Priority	Processor	Execution requirement	Default speed	Co-runner specification
Task 1	1.5	1.5	1	3	1			
Segment 1						0.25	0.5	[[[3, 1]], 1.0], [[4, 1]], 0.5], [[5, 1]], 1.0], [[5, 2]], 1.0]]
Segment 2								
Segment 3								
Task 2	2.0	2.0	1	2				1]], 1.0], [[5, 1]], 1.0], [[5, 2]], 1.0]]
Segment 1								
Segment 2								
Segment 3								
Task 3	2.0	2.0	1	3				1]], 0.5]]
Segment 1								
Segment 2								
Segment 3								
Task 4	2.0	2.0	1	2				1]], 1.0]]
Segment 1								
Segment 2								
Segment 3								
Task 5	2.25	2.25	2	1	2			
Segment 1						0.5	1.0	[[[1, 1]], 1.0], [[2, 1]], 1.0]]
Segment 2						0.125	0.5	[[[1, 1]], 0.5], [[2, 1]], 1.0]]
Segment 3								

Do schedulability analysis

Taskset is schedulable

Upper bounds on the response times of task are as follows:

For task 1: 0.5

For task 2: 1.0

For task 3: 0.5

For task 4: 1.0

For task 5: 1.75

OK

References

[Kirk89] D. Kirk, "SMART (Strategic Memory Allocation for Real-Time) Cache Design," RTSS, 1989.

Main idea: The hardware is designed so that a cache is composed of M partitions and one shared pool. There is also a hardware unit called mapping function which translates each memory access (based on memory address and user id and other info) to a decision on whether the memory access should operate on the shared pool or one of the partitions (and if so, which partition). A task can be assigned more than one partition. The decision on how to allocate partitions to tasks is performed with the idea of maximizing the marginal reduction in the utilization of the taskset.

[Wolfe94] A. Wolfe, "Software-Based Cache Partitioning for Real-Time Applications," International Workshop on Responsive Computer Systems, 1997.

Main idea: Use the virtual-to-physical address translation to make sure that for different processes, the physical addresses generated map to different cache sets (and hence avoid cache eviction).

[Mueller95] F. Mueller, "Compiler Support for Software-Based Cache Partitioning," ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems, 1995.

Main idea: Use the idea in [Wolfe94] but let the compiler do the cache coloring.

[Liedtke97] J. Liedtke, H. Hartig, and M. Hohmuth, "OS-controlled cache predictability for real-time systems," RTAS, 1997.

Main idea: Similar to [Wolfe94] but with OS perspective.

[Belloso97] F. Belloso, "Process Cruise Control: Throttling Memory Access in a Soft Real-Time Environment," Technical Report, University of Erlangen-Nürnberg, 1997.

Main idea: If a given process performs more accesses to the memory bus than it is allowed, then the process is slowed down (by having the TLB miss handler executing NOP instructions).

[Schönberg03] S. Schönberg, "Impact of PCI-bus load on applications in a PC architecture," RTSS, 2003.

Main idea: Compute the slowdown (from DMA accesses causing memory bus accesses which contend with the program's accesses on the memory bus) of the execution of a program

[Edwards07] S. Edwards and E. Lee, "The Case for Precision Timed (PRET) Machine," DAC, 2007.

Main idea: Hw and sw abstractions need to change to be time predictable; e.g., cache should be replaced with scratchpad.

[Rosén07] J. Rosén, A. Andrei, P. Eles, and Z. Peng, "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-On-Chip," RTSS'07.

Main idea: Create a TDMA bus schedule according to the needs of a program (both message passing and cache misses).

[Pellizzoni07] R. Pellizzoni and M. Caccamo, "Toward the Predictable Integration of Real-Time COTS based Systems," RTSS'07.

Main idea: Find a bound on the number of cache misses of a program and a bound on the number of front-side bus accesses from I/O device and compute additional execution time of program. Round-robin bus. Also, perform policing of I/O device.

References

[Steffens08] L. Steffens, M. Agarwal, and P. Wolf, “Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach,” ECRTS, 2008.

Main idea: Analyze cumulative delays of cache misses (low latency streams) using network calculus and also consider message passing. Configure enforcement. Simulation-based approach to obtain cumulative delays of cache misses.

[Schliecker08] S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, R. Ernst, “Reliable Performance Analysis of a Multicore Multithreaded System-on-Chip,” CODES+ISSS, 2008.

Main idea: Compute cumulative delay of memory accesses considering contention on the memory bus. Assume work-conserving memory bus but except from that, make no assumption on arbitration.

[Pellizzoni08] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha, “Coscheduling of CPU and I/O Transactions in COTS-based Embedded Systems,” RTSS, 2008.

Main idea: Extension of [Pellizzoni07]. Intel Core2. Implementing the policer.

[Bui08] B. Bui, M. Caccamo, L. Sha, and J. Martinez, “Impact of Cache Partitioning on Multi-Tasking Real-Time Embedded Systems,” RTCSA, 2008.

Main idea: Use genetic programming to decide how many cache colors a task should have.

[Bourgade 08] R. Bourgade, C. Ballabriga, H. Cassè, C. Rochange, and P. Sainrat, “Accurate analysis of memory latencies for WCET estimation,” RTNS, 2008.

Main idea: DRAM memories are organized as banks with one row buffer for each bank. If a memory access has a memory address such that for the bank that holds that data, its row contains the data to be accessed, then the memory latency is small; otherwise it is large. This paper considers this effect in WCET analysis.

References

[Andersson09] B. Andersson, A. Easwaran, and J. Lee, "Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-Based Multicore Systems," RTSS-WIP, 2009.

Main idea: Model the memory bus of COTS multicore as work-conserving (cache misses). Obtain model from traces.

[Paolieri09] M. Paolieri, E. Quiones, F. Cazorla, G. Bernat, and M. Valero, "Hardware Support for WCET Analysis of Hard Real-Time Multicore Systems," ISCA, 2009.

Main idea: Create hardware that makes timing predictable. Use TDMA bus and h/w cache partitioning. Implement a WCET computation mode (which ensures that that time a memory operation takes is equal to its maximum).

[Kinnan09] L. Kinnan, "Use of multicore processors in avionics systems and its potential impact on implementation and certification," DASC, 2009.

Main idea: General discussion on the topic. Mentions the importance of service history. Mentions that cache coherency protocols can operate much faster in multicores than in multiprocessors on separate chips. Mentions that contention/eviction on a shared L2 cache is particularly severe if two tasks on different processor cores run the same software synchronized (this might be an issue if a multicore is used to achieve fault-tolerance). Also points out that certification requires transparency of hardware but chip makers typically do not want to disclose details. Points out that processor cores within a multicore share clock signals and power signals and hence are less fault tolerant than multiprocessors implemented with multiple chips.

References

[Pellizzoni10] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst Case Delay Analysis of Memory Interference in Multicore Systems," DATE, 2010.

Main idea: Compute upper bounds on extra execution of a task due to bus contention. Assume TDMA scheduling of tasks. Assume different types of bus arbitration (RR,FCS,priority).

[Schranzhofer10] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, "Worst-Case Response Time Analysis of Resource Access Models in Multi-Core Systems," DAC, 2010.

Main idea: Extend [Pellizzoni10] with new models for accessing shared hardware resources; one of them is "dedicated phases" which only allows implicit-communication in the beginning and end of a superblock. Use TDMA bus.

[Pellizzoni10] R. Pellizzoni and M. Caccamo, "Impact of Peripheral-Processor Interference on WCET Analysis of Real-Time Embedded Systems," IEEE Transactions on Computers, 2010.

Main idea: Extend [Pellizzoni07] to a journal article.

[Chattopadhyay10] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling Shared Cache and Bus in Multi-cores for Timing Analysis," SCOPES, 2010.

Main idea: Analyze shared cache and memory bus jointly. Assume TDMA bus and use abstract interpretation in cache analysis. Consider an application comprises multiple tasks with potentially precedence constraints between these tasks. Non-preemptive partitioned scheduling.

[Fuchsen10] R. Fuchsen and R. Winterheim, "How to address certification for multi-core based IMA platforms: current status and potential solutions," DASC, 2010.

Main idea: Measure slowdown of execution because of sharing resources in the memory system.

[Lv10] M. Lv, W. Yi, N. Guan, and G. Yu, "Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software," RTSS, 2010.

Main idea: Describe a program with a control flow graph (CFG) and use abstract interpretation to classify memory accesses in each basic block and then formulate a timed automaton for each task with each basic block being a sequence of locations and then analysis bus contention delay with a Timed-Automata model checker (Uppaal).

References

[Dasari11] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and Jinkyu Lee, "Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus," TrustCom, 2011.

Main idea: Compute worst-case response times of tasks making no assumption on the arbitration policy for the memory bus except assuming that the memory bus is work-conserving.

[Rosén11] J. Rosén, C. F. Neikter, P. Eles, Z. Peng, P. Burgio, and L. Benini, "Bus Access Design for Combined Worst and Average Case Execution Time Optimization of Predictable Real-Time Applications on Multiprocessor Systems-On-Chip," RTAS'11.

Main idea: Similar to [Rosén07].

[Yoon11] M.-K. Yoon, J.-E. Kim, L. Sha, "Optimizing Tunable WCET with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems," RTSS, 2011.

Main idea: Use special hardware that provides predictable WCET. Consider a TDMA memory bus so that the total utilization of the taskset is minimized (e.g., a task with small period and/or many memory accesses should receive more slots in the TDMA schedule).

[Chattopadhyay11] S. Chattopadhyay and A. Roychoudhury "Scalable and Precise Refinement of Cache Timing Analysis via Model Checking," RTSS 2011.

Main idea: Extend WCET and CRPD analysis to use model checking for better precision.

[Radojkovic11] P. Radojkovic, S. Girbal, A. Grasset, E. Quinones, S. Yehia, and F. J. Cazorla, "On the evaluation of the Impact of Shared Resources in Multithreaded COTS Processors in Time-Critical Environments," ACM Transactions on Architecture and Code Optimization, 2011.

Main idea: Create stressing-benchmarks that stress different types of shared resources (e.g. instruction fetch stage in pipeline, later stages in pipeline, L1 cache, L2 cache, memory bandwidth) and find experimentally how much the execution of a pair of stressing-benchmarks is slowed down when executing in parallel. Also, experimentally find slowdown when an application executes in parallel with one of the stressing benchmarks.

[Herter11] J. Herter, P. Backes, F. Hauptenthal, and J. Reineke, "CAMA: A Predictable Cache-Aware Memory Allocator," ECRTS, 2011.

Main idea: Memory allocator where a task specifies not only the size of requested memory block but also the cache color it the requested memory block. This provides more information to WCET analysis. The allocator is implemented by having one list of free memory blocks per cache color.

References

[Nowotsch12] J. Nowotsch and M. Paulitsch, “Leveraging multi-core computing architectures in avionics,” EDCC, 2012.

Main idea: Provide a test approach that models the worst-case behavior for the case of concurrent network and memory usage by multiple applications.

[Mancuso13] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, “Real-Time Cache Management Framework for Multicore Architectures,” RTAS, 2013.

Main idea: Use profiling of memory accesses of programs and use it to guide cache allocation. Also, combine page coloring with cache locking (use page coloring to map frequently accessed pages to certain cache sets and then lock cache blocks of those cache sets).

[Ward13] B. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, “Making Shared Caches More Predictable on Multicore Platforms,” ECRTS, 2013.

Main idea: Use cache coloring and treat cache sets as a shared resource; that is, a task must clock cache sets before starting to execute; then it can release.

[Wu13] Z. Wu, Y. Krish, and R. Pellizzoni, “Worst-Case Analysis of DRAM Latency in Multi-Requestor Systems,” RTSS, 2013.

Main idea: Model the time it takes for a memory operation to be performed considering DRAM timing parameters. Then use this to compute upper bounds on cumulative delay that a program can experience.

[Suzuki13] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, “Coordinated Bank and Cache Coloring for Temporal Protection of Memory Accesses,” ICESS, 2013.

Main idea: Setup the virtual-to-physical translation so that different tasks access different cache sets and different memory banks. This provides cache and memory bank isolation.

References

[Kim14] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, R. Rajkumar, “Bounding Memory Interference Delay in COTS-based Multi-Core Systems,” RTAS, 2014.

Main idea: Model the time it takes for a memory operation to be performed considering DRAM timing parameters. Then use this to compute upper bounds on response times. Assume that a task τ_i performs at most H_i memory accesses. This work differs from [Wu13] in that (i) schedulability analysis is performed (not just compute cumulative latency) and (ii) memory bank sharing is allowed.

[Lampka14] K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. Wu, N. Stoimenov, “A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets,” Real-Time Systems, 2014.

Main idea: Use PREM (that is a program is divided into three parts, fetch data, compute, and write-back result) and partitioned non-preemptive scheduling. Consider the software as consisting of superblocks; a superblock has upper and lower bound on execution time and memory accesses. For each processor core, find a function that is an upper bound on the number of memory accesses in a time interval of duration t . For a processor core under analysis (denoted p), describe the upper bound of the number of memory accesses from other processor cores and let us timed automaton represent the events that memory accesses are generated; this timed automaton must respect the upper bound as mentioned. Then model the bus arbitrator as a timed automaton. And model a superblock as a timed automaton as well. Then state the query that for all possible execution, the response time is at most certain bound. Do binary search on this upper bound. This gives us upper bound on the response time. The paper shows that almost tight bounds can be computed.

[Ye14] Y. Ye, R. West, Z Cheng, and Y. Li, “COLORIS: A Dynamic Cache Partitioning System Using Page Color,” PACT, 2014.

Main idea: Use cache partitioning implemented in software (using the virtual-to-physical translation mechanism) and change the partitioning at run-time (in order to support more tasks and so support changes in the memory footprint).

[Nowotsch14] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, “Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement,” ECRTS, 2014.

Main idea: Use static scheduling (TDMA) to schedule tasks. Assume a round-robin bus. Compute the execution times of tasks.

References

[Yun15a] Heechul Yun,, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha, "Memory Bandwidth Management for Efficient Performance Isolation in Multi-core Platforms," IEEE Transactions on Computers, 2015.

Main idea: Perform policing on the memory bus. The available bandwidth is time-varying because some memory operations are fast (e.g., row hit) and others are slow (e.g., row miss). For soft real-time: reclaim unused memory bandwidth; for hard real-time: disable the reclamation. The sum of bandwidth should be kept below a certain threshold (e.g., 1.2GBps); this is typically much smaller than peak bandwidth (6.4GBps in the system considered in the article).

[Graciolo15] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Frölich, and R. Pellizzoni, "A Survey on Cache Management Mechanisms for Real-Time Embedded Systems," ACM Computing Surveys, 2015.

[Yun15b] H. Yun, R. Pellizzoni, and P. K. Valsan, "Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems," ECRTS, 2015.

Main idea: Modify [Kim14] so that the model the analysis is based on allows read-prioritization and multiple outstanding memory requests.

[Yun15c] H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Processors," OSPERT, 2015.

Main idea: Evaluate the impact of co-runners on execution times. Do this evaluation on three platforms: ARM7, ARM15, and Intel Nehalem.

Find that in some cases the execution time can increase 103 times. Even with cache partitioning, the execution time can increase 14times; this is because of the Miss Status Holding Register (MSHR).

[Panchamukhi15] S.A. Panchamukhi and F. Mueller, "Providing Task Isolation via TLB Coloring," RTAS, 2015.

Main idea: Use the compiler/linker to allocate code and data of each task so that when the tasks run, TLB entries of one task does not evict TLB entries of another task.

References

[Li16] Y. Li, B. Akesson, K. Lampka, and K. Goossens, “Modeling and Verification of Dynamic Command Scheduling for Real-Time Memory Controllers,” RTAS, 2016.

Main idea: Model many of the details of the memory controller (timing specifications by JEDEC) as a timed automaton. Then describe a network of timed automata and compute the worst-case response time of a task.

[Sha16] L. Sha, M. Caccamo, R. Mancuso, J.-E. Kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. B. Kegley, D. Perlman, G. Arundale, and R. Bradford, “Real-Time Computing on Multicore Processors,” Computer, 2016.

Main idea: A framework single-core equivalence (SCE) involving (i) cache locking, (ii) bank coloring, and (iii) memory guard (policing the memory accesses). The memory guard makes the execution time of one task independent of the memory bus contention of other task but it comes at the cost of low memory bandwidth (1Gbps). SCE uses an I/O partition. SCE assumes that the h/w supports cache locking and performance monitoring counters. With SCE, the execution time of a task can increase by approximately 50% (see Figure 5) for 8 cores.

[Kim16] N. Kim, B. C. Ward, M. Chisholm, C.-Y. Fu, J.H. Anderson, and F.D. Smith, “Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning,” RTAS, 2016.

Main idea: Use isolation mechanisms for high-criticality tasks and let low-criticality tasks share resources.

[Kim16] N. Kim, B. C. Ward, M. Chisholm, C.-Y. Fu, J.H. Anderson, and F.D. Smith, “Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning,” RTAS, 2016.

Main idea: Use isolation mechanisms for high-criticality tasks and let low-criticality tasks share resources.

[CAST32A] Certification Authorities Software Team (CAST), Position Paper, CAST-32A, Multi-core Processors, COMPLETED November 2016 (Rev 0), Available at https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/

[Sha16b] L. Sha, M. Caccamo, G. Shelton, M. Nuessen, J. P. Smith, D. Miller, R. Bradford, R. Kegley, D. Perlman, J. Preston, J. W. Wlad, M. Storr, D. DeNiz, S. Chaki, M. Klein, B. Andersson, I. Bate, A. Burns, S. Palin, S. Bak, D. Kingston, M. Clark, T. Kim, and E. Pak, “Position Paper on Minimal Multicore Avionics Certification Guidance,” August 4, 2016.