Carnegie Mellon University
Software Engineering Institute

Resilient Scalable Verification of Cyber-Physical Systems

March 2022

Dionisio (Dio) de Niz

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

DM22-0225

Resilient Scalable Verification of Cyber-Physical Systems

Carnegie Mellon University Software Engineering Institute

Scalable Verification

- Larger Systems
- Rapidly Evolving
- Critical Kinetic Consequences
- In presence of cyber-attacks

New Technology: Multicore

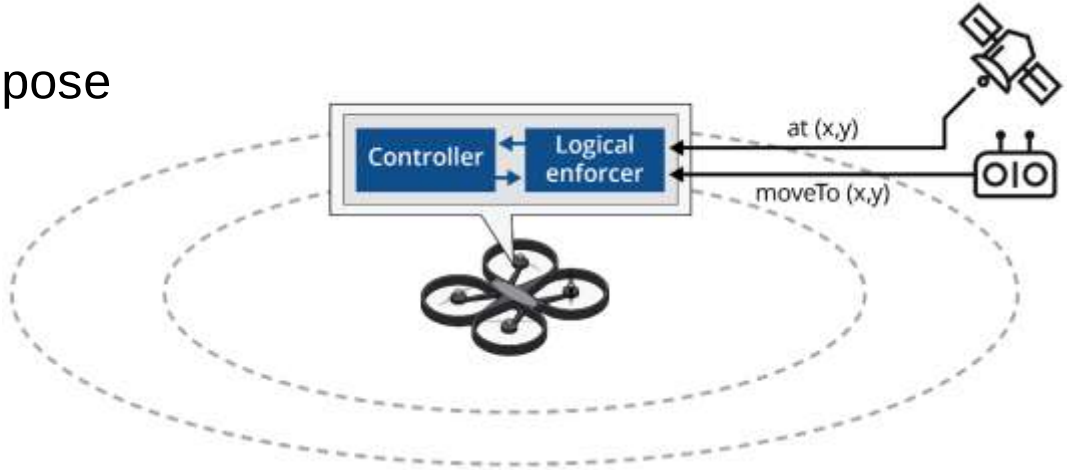
- Born for general-purpose computing
- Critical: Need to ensure they work in the worst-case

Preventing Costly Integration Errors with Early Analysis

- Early Design Models & Analysis

Scalable Enforcement-Based Verification

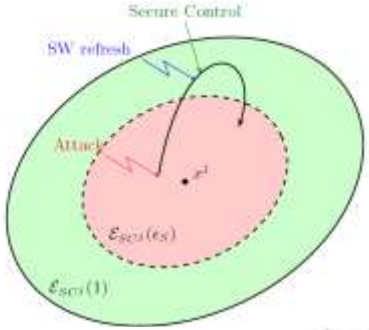
- Leave Most Code **Unverified**
- Add **simpler (verifiable)** runtime enforcer to make algorithms predictable
- Formally: specify, verify, and compose multiple enforcers
  - Logic: replaces unsafe values
  - Timing: at right time
  - Physics: verified physical effects
- Enforcer protection against failures/attacks
- Resilient to failures / dynamic environment



Verifying Physics (Control Theory)

Recoverable Set: $\varepsilon_{SCj}(1)$

Safety Set: $\varepsilon_{SCj}(\epsilon_s) \triangleq \epsilon_s\, \varepsilon_{SCj}(1)$

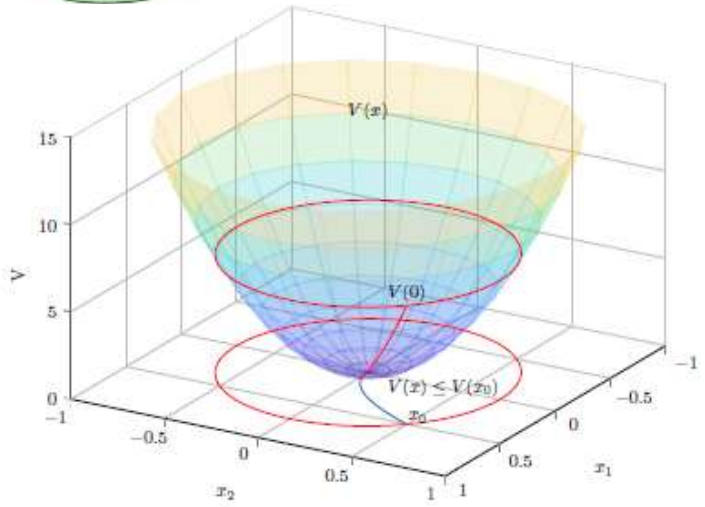


Controlled System: $\dot{x} = f_\varphi(x) \triangleq f(x, \varphi(x))$
Lyapunov Function: $V_\varphi : \mathbb{R}^n \to \mathbb{R}$, $\mathcal{N}_{V_\varphi}(x_{eq}) \subseteq \mathcal{N}_\varphi(x_{eq})$,
$V_\phi(x_{eq}) = 0$ and $\forall x \in \mathcal{N}_{V_\varphi}(x_{eq}) - \{x_{eq}\}$: (i) $V_\varphi(x) > 0$,

$$\dot{V}_\varphi(x) = \frac{\partial V}{\partial x} \cdot f_\varphi(x) < 0$$

Lyapunov level set:For $\epsilon > 0$,

$$\mathcal{E}_\varphi(\epsilon) = \{x \in \mathcal{N}_{V_\varphi}(x_{eq}) | V_\varphi(x) \leq \epsilon\}. \qquad \epsilon \leq 1$$

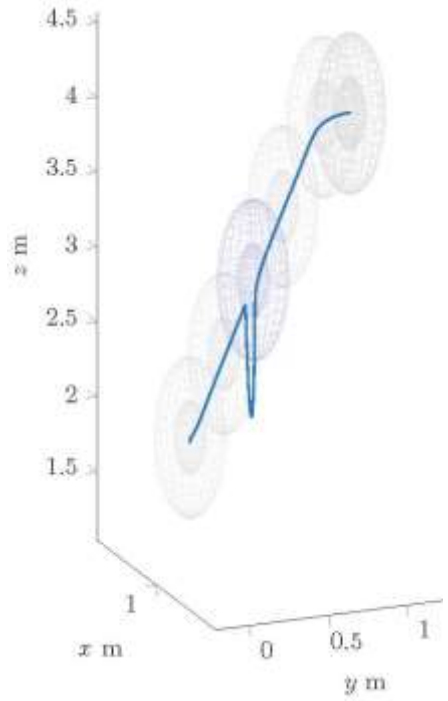


R. Romagnoli, B. H. Krogh, B. Sinopoli. Design of Softw are Rejuvenation for CPS Security Using Invariant Sets. ACC 2019

Analysis of Mission Progress Enforcing Unsafe Behavior

6 DOF $\Rightarrow$ 12 state variables

$$\ddot{p}_x = -\cos\phi \sin\theta \frac{F}{m}$$
$$\ddot{p}_y = \sin\phi \frac{F}{m}$$
$$\ddot{p}_z = g - \cos\phi\cos\theta \frac{F}{m}$$
$$\ddot{\phi} = \frac{1}{J_x}\tau_\phi$$
$$\ddot{\theta} = \frac{1}{J_y}\tau_\theta$$
$$\ddot{\psi} = \frac{1}{J_z}\tau_\psi.$$

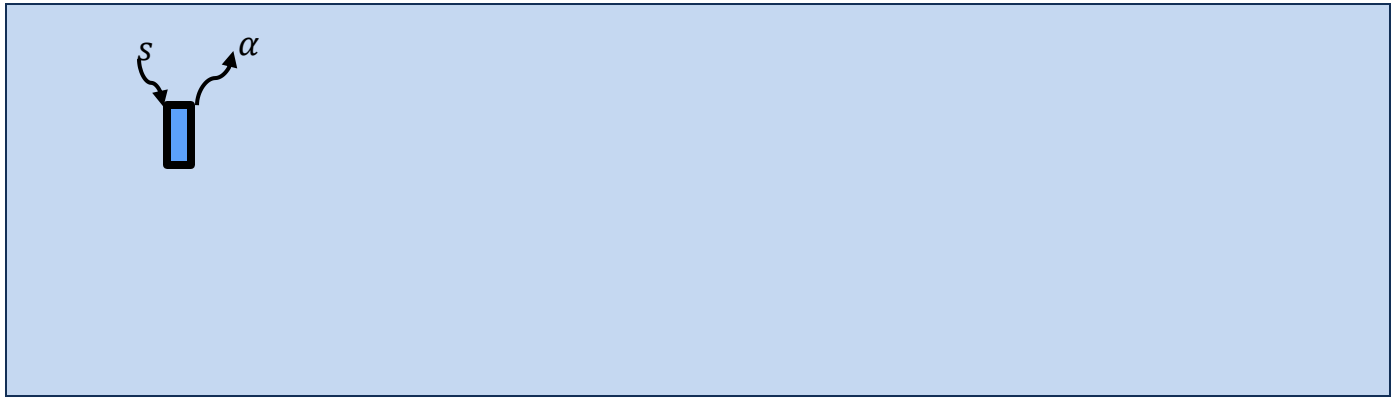


Linear design:

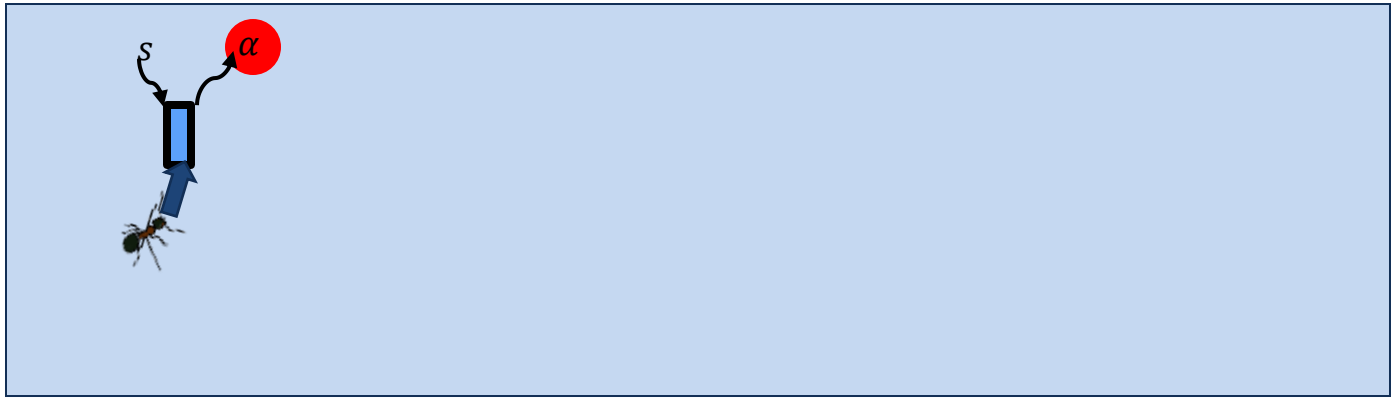
- linearize at equilibrium
- assume full state available
- LQ state feedback design
- reference points = equilibrium states

Drone Experiment

Enforcing Unverified Components

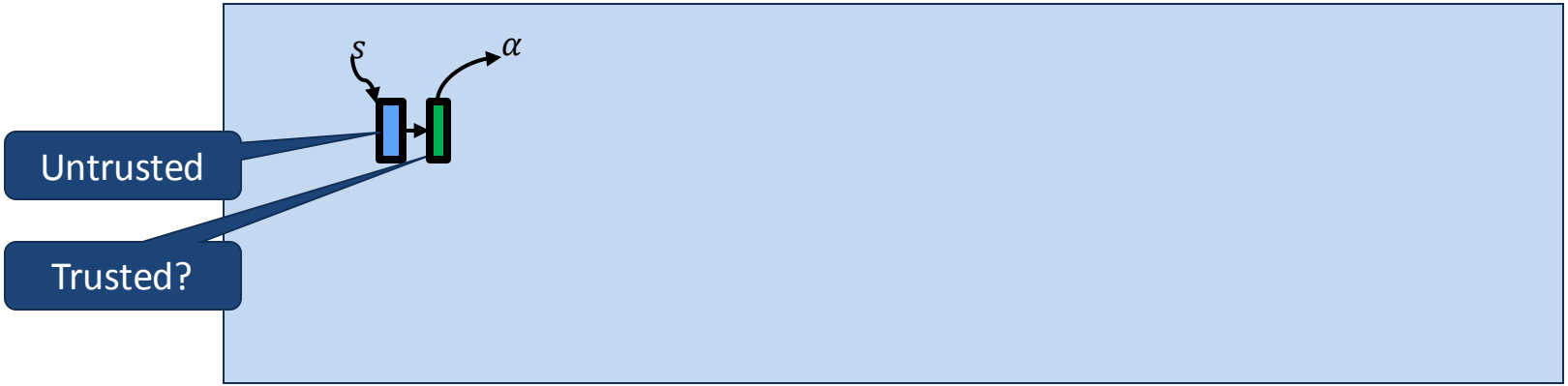


Enforcing Unverified Components

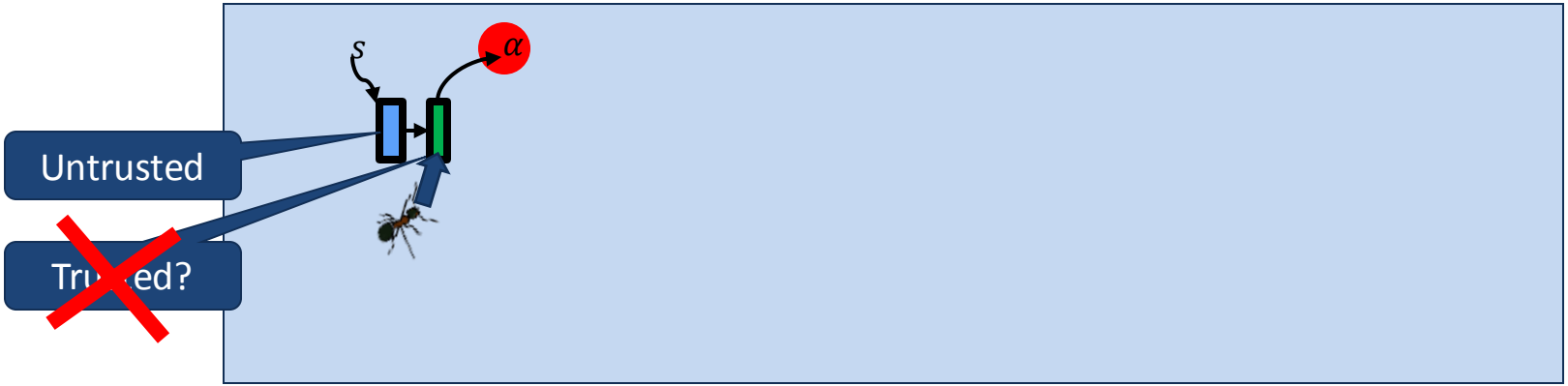


Ant illustration by Jan Gillbank, license by Creative Commons Attribution 3.0 Unported

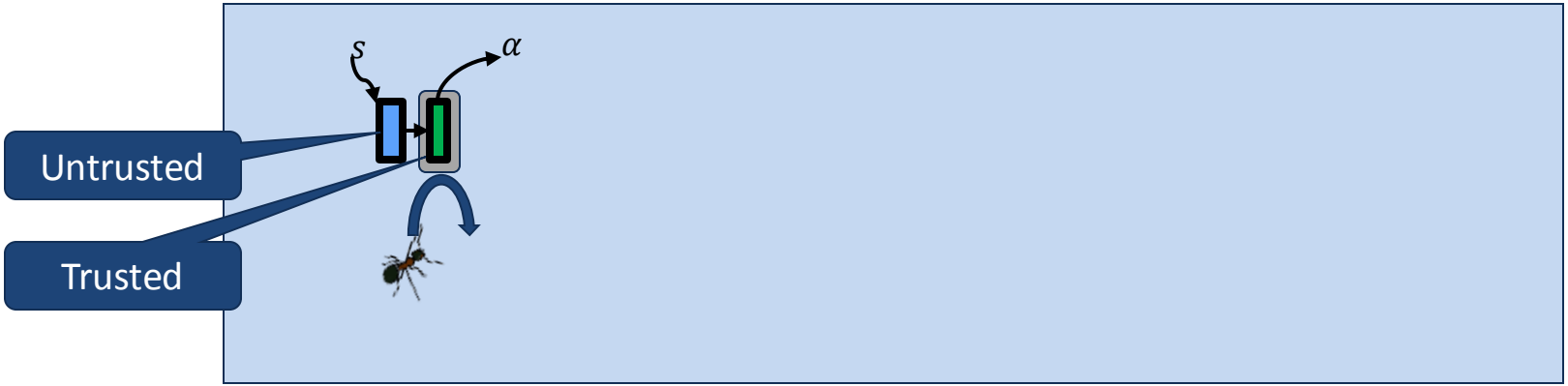
Enforcing Unverified Components



But enforcer can be corrupted (bug or cyber attack)

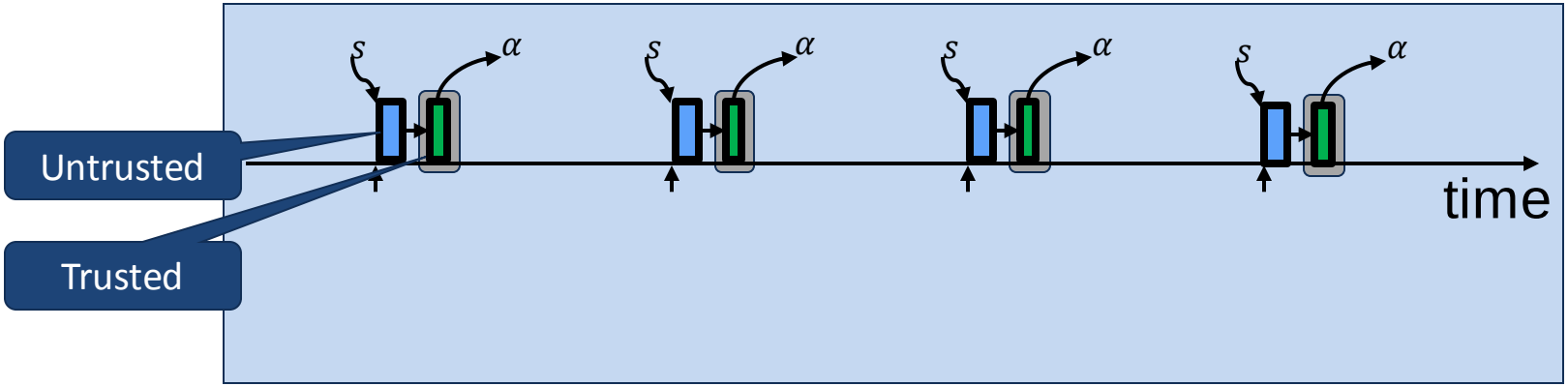


Add Memory Protection

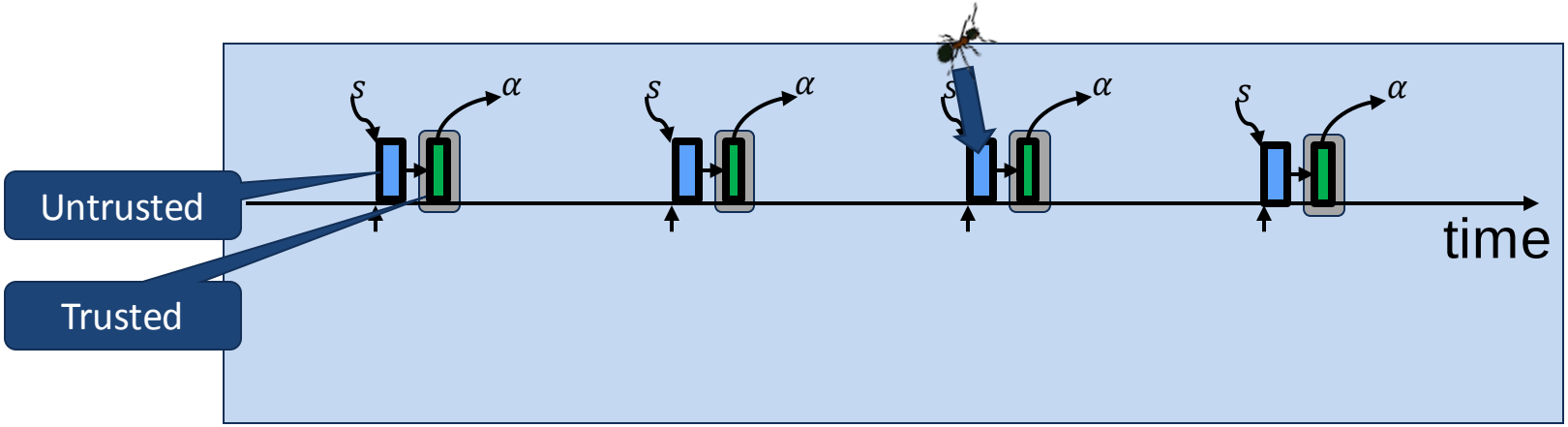


Trusted = Verified & Protected

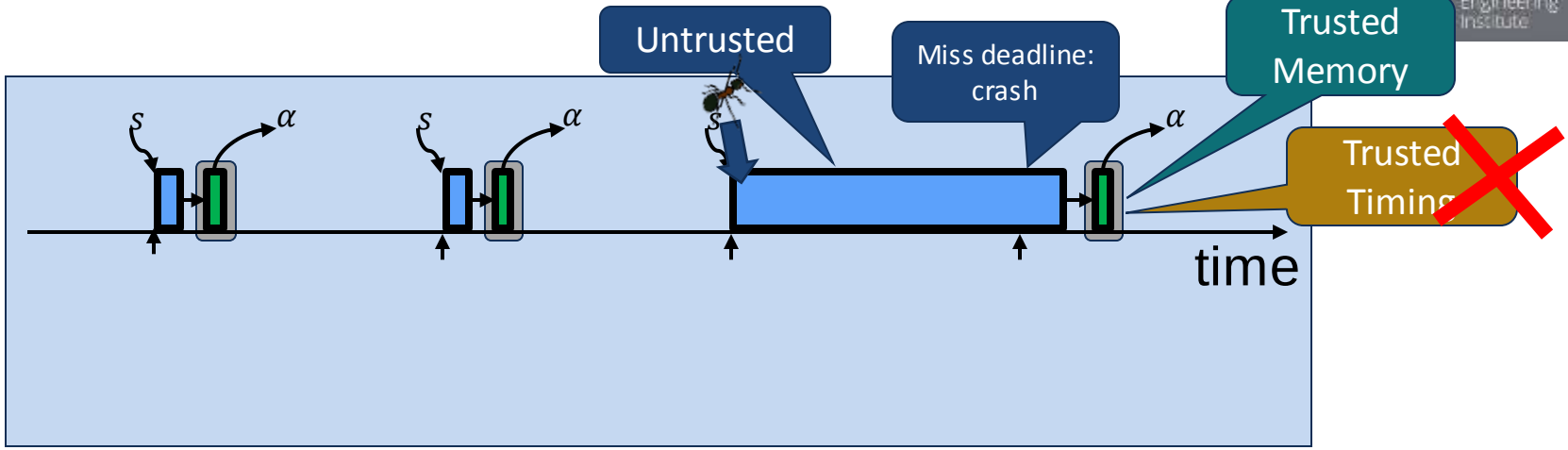
Periodic Execution Must Finish by Deadline



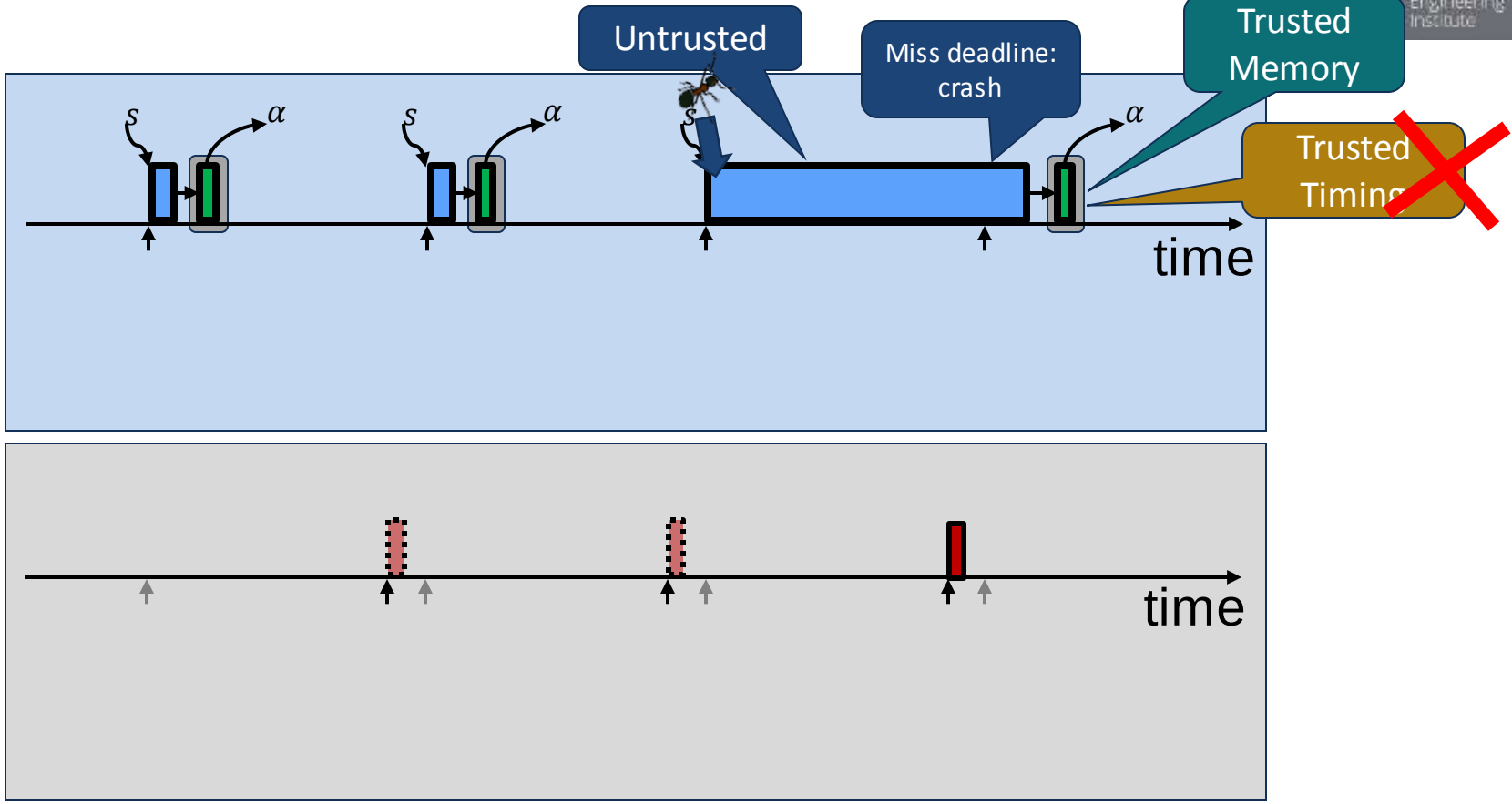
Periodic Execution Must Finish by Deadline



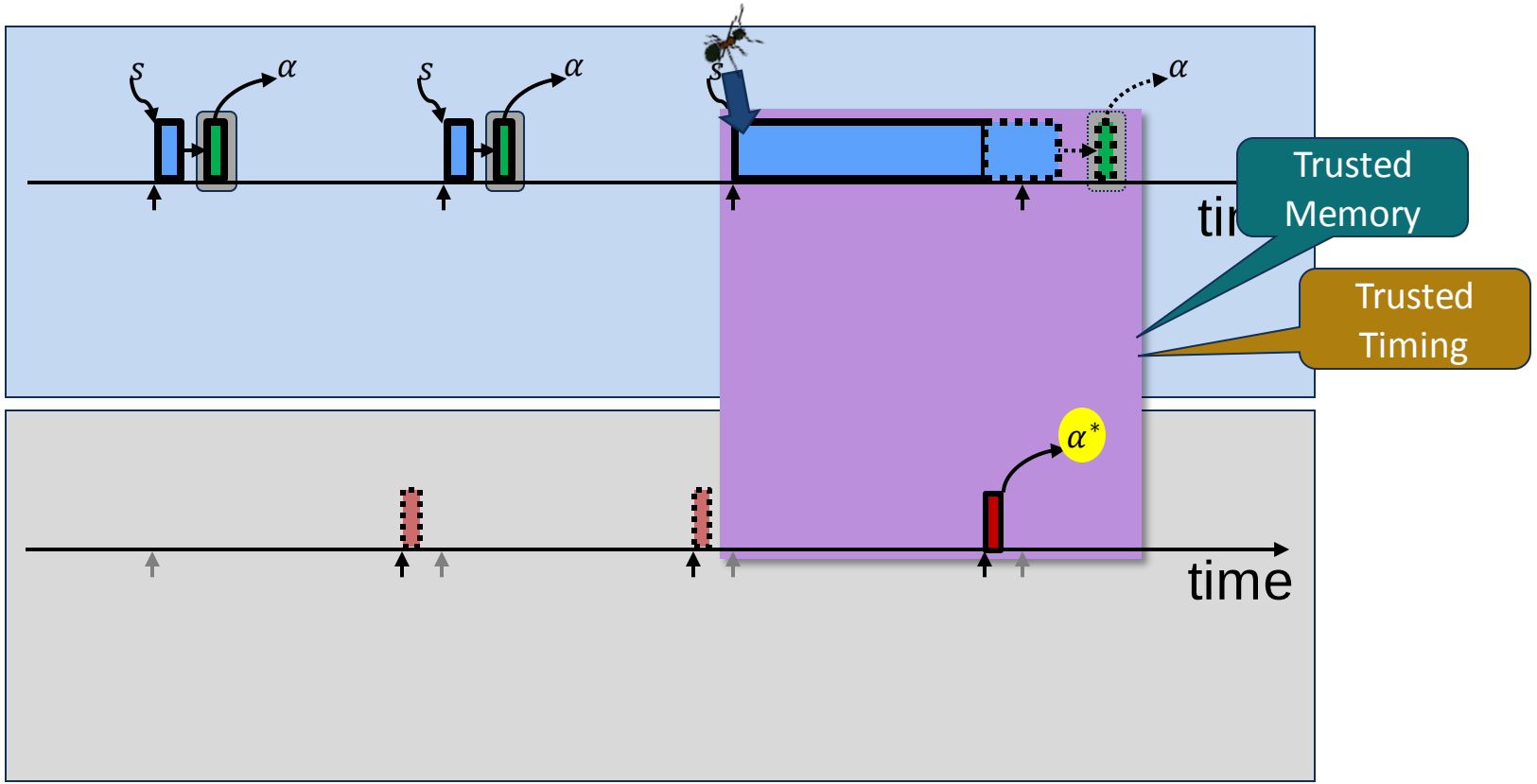
Periodic Execution Finish by Deadline



Periodic Execution Finish by Deadline



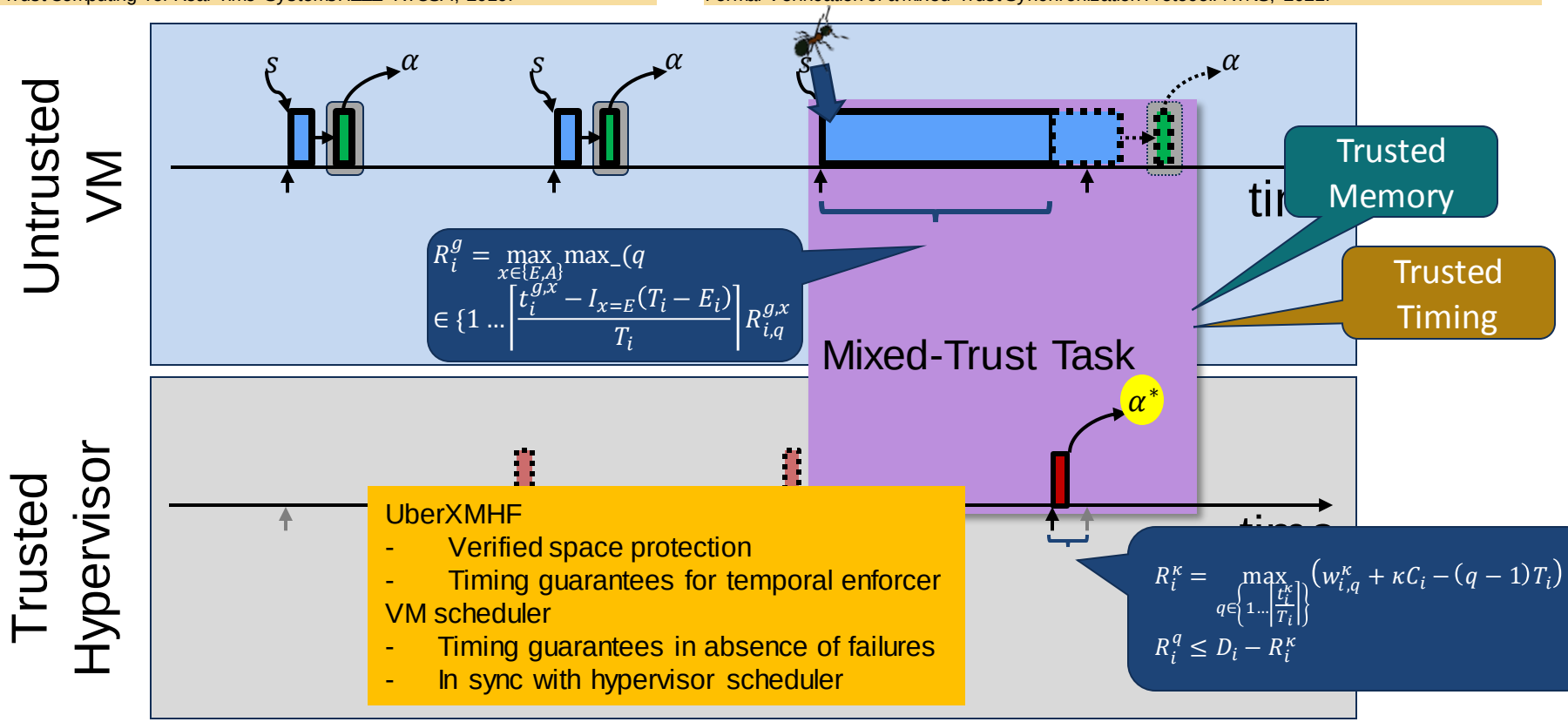
Periodic Execution Finish by Deadline



Real-Time Mixed-Trust Computation

D. de Niz, B. Andersson, M. Klein, J. Lehoczky, A. Vasudevan, H. Kim, & G. Moreno. Mixed-Trust Computing for Real-Time Systems. IEEE RTCSA, 2019.

R.Martins, M. McCall, D. de Niz, A. Vasudevan, B. Andersson, M. Klein, J. Lehoczky, and H. Kim. Formal Verification of a Mixed-Trust Synchronization Protocol. RTNS, 2021.

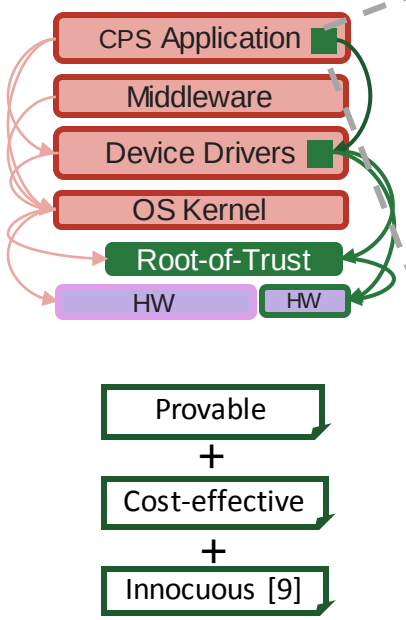


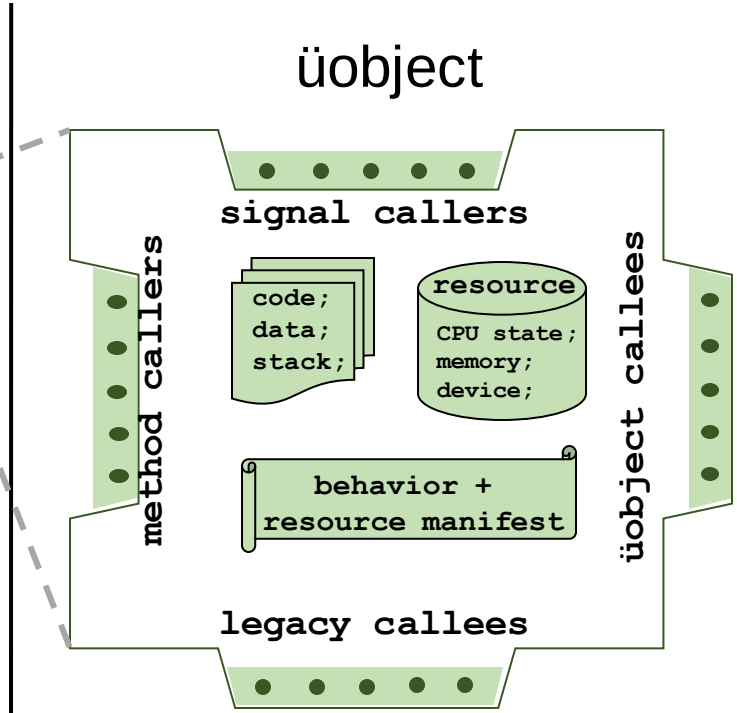
$$R_i^g = \max_{x \in \{E,A\}} \max\_(q \in \{1 \ldots \left\lceil \frac{t_i^{g,x} - I_{x=E}(T_i - E_i)}{T_i} \right\rceil R_{i,q}^{g,x}$$

$$R_i^\kappa = \max_{q \in \left\{1 \ldots \left\lceil \frac{L_i^\kappa}{T_i} \right\rceil\right\}} \left(w_{i,q}^\kappa + \kappa C_i - (q-1)T_i\right)$$

$$R_i^q \leq D_i - R_i^\kappa$$

Verified Protection at Hypervisor, Kernel, Application





- Singleton object guarding exclusive indivisible system resource
- Principled entry, interruption, legacy code invocations and üobject invocations
  - execution trace respecting program control-flow enables use of state-of-the-art program verification tools
  - facilitate AG reasoning and composition
- Call-return Interfacing
  - Handle various CHIC programming idioms
- Resource Interface Confinement
  - Resource protection and access control
  - Support Shared memory concurrency -> multi-threaded execution and reasoning

A. Vasudevan, P. Maniatis, R.Martins, S. Chaki. Practical, Provable, End-to-End Guarantees at the Edge. USENIX Workshop on Hot Topics in Edge Computing 2020

M. McCormack, A. Vasudevan, G. Liu, V. Sekar
Formalizing an Architectural Model of a Trustworthy Edge IoT Security Gateway
RTCSA 2021

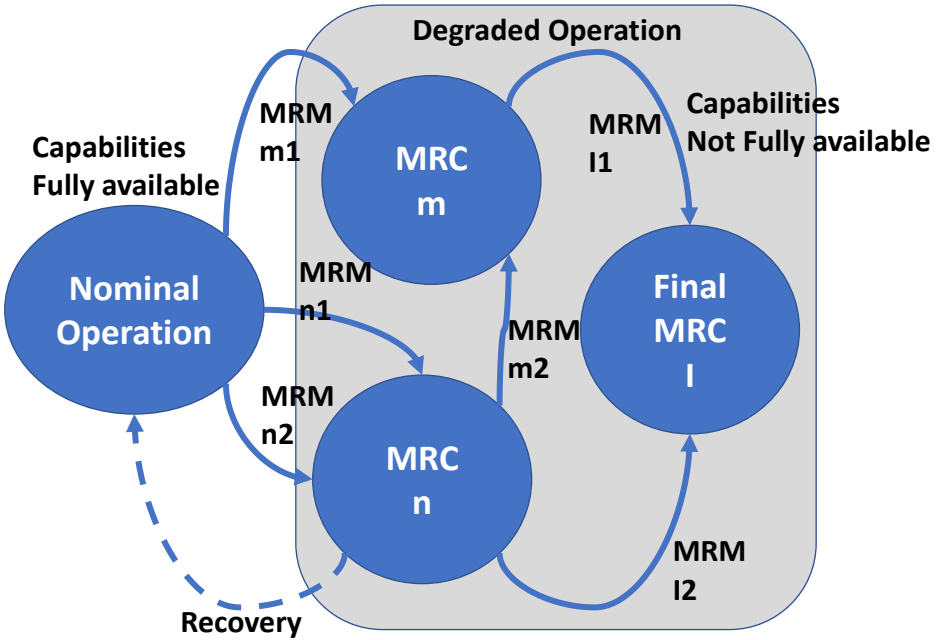
Resilient Real-Time Mixed-Trust

CPS need to adapt to failures/environment

- Daimler: Safety First for Automate Driving[1]
  - Built to preserve safety across failures
    - Minimum Risk Maneuver (MRM) to transition to degraded Mode or Minimal Risk Condition (MRC)

Resilient Real-Time Mixed Trust

- Add enforcer to preserve safety
- Use enforcer to execute MRM



[1]Daimler et al. Safety First for Automated Driving.
https://www.daimler.com/documents/innovation/other/safety-first-for-automated-driving.pdf, 2019

Collision Avoidance Enforcer Example

LIDAR

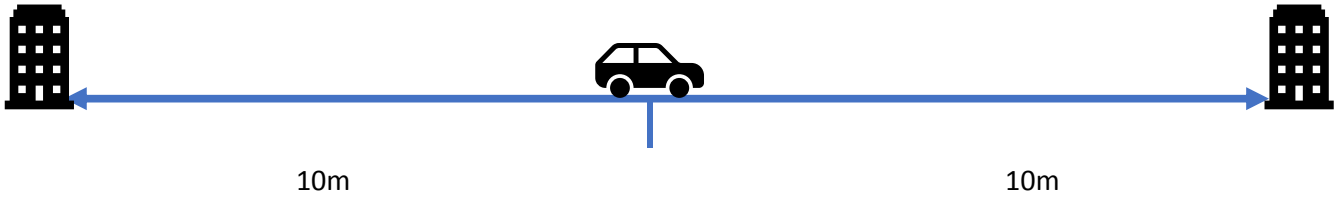
- Detection distance $20\ m$
- Max braking: $-10\ \frac{m}{s^2}$
- Max speed: $20\ \frac{m}{s}$

SONAR

- Detection distance $5\ m$
- Max braking: $-10\ m/s^2$
- Max speed: $10\ \frac{m}{s}$

LIDAR Failure Transitioning Enforcer

- Upon failure: start braking at $-10\ \frac{m}{s^2}$
- Once speed < $10\ \frac{m}{s}$ Transition to SONAR enforcer



Resilient Real-Time Mixed-Trust Digraph Model / Scheduling

D. de Niz, B. Andersson, H. Kim, M. Klein, and J. Lehoczky.
Resilient Mixed-Trust Scheduling. IEEE RTSS 2021.

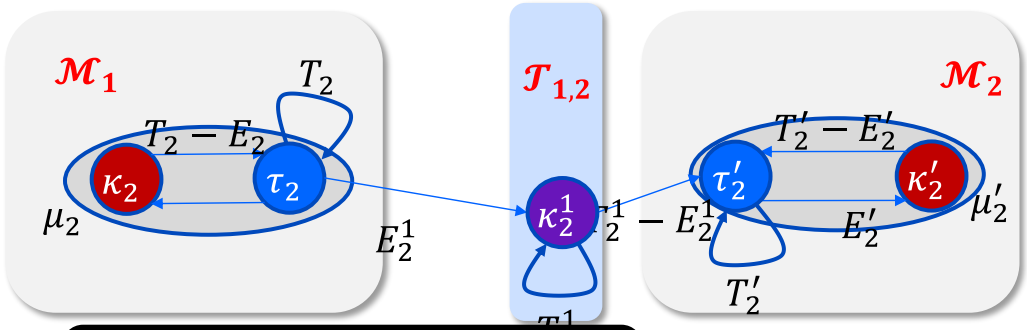
Task: graph $G_i = (V_i, E_i)$

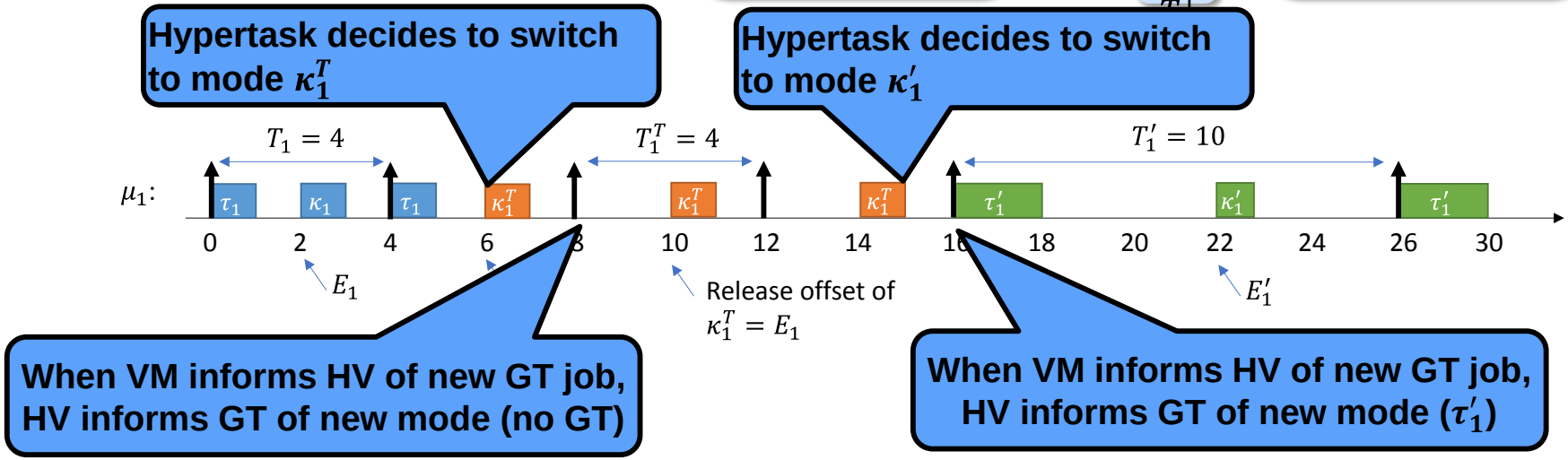
$V_i = \{v_{i,1}, v_{i,2}, \dots\}$

$E_i = \{e_{i,1}, e_{i,2}, \dots\}$

$v_{i,1} = (\tau_{i,1}, \kappa_{i,1}), \dots$

$e_{i,1} = (v_{i,1}, v_{i,2}), \dots$

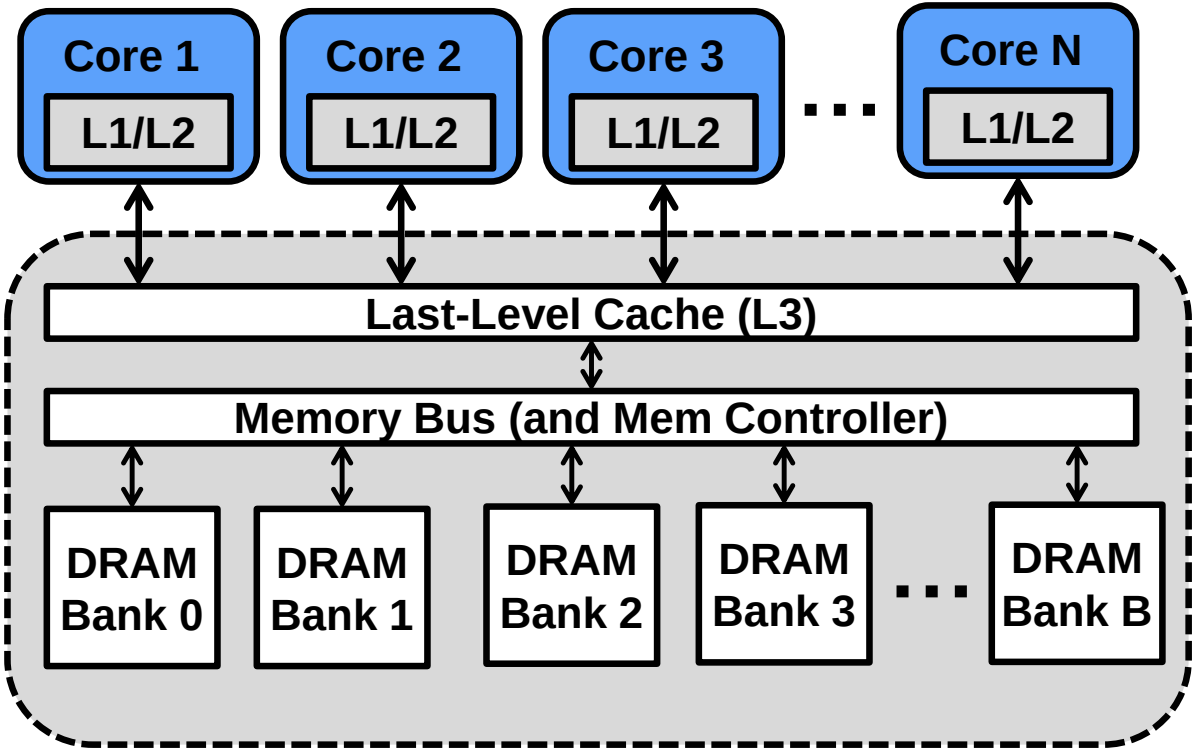




Carnegie Mellon University
Software Engineering Institute

Multicore

Multicore Processors: A Shared Hardware Problem



Shared Hardware Delays: Shared Cache

Core 1

Core 2

Shared Cache

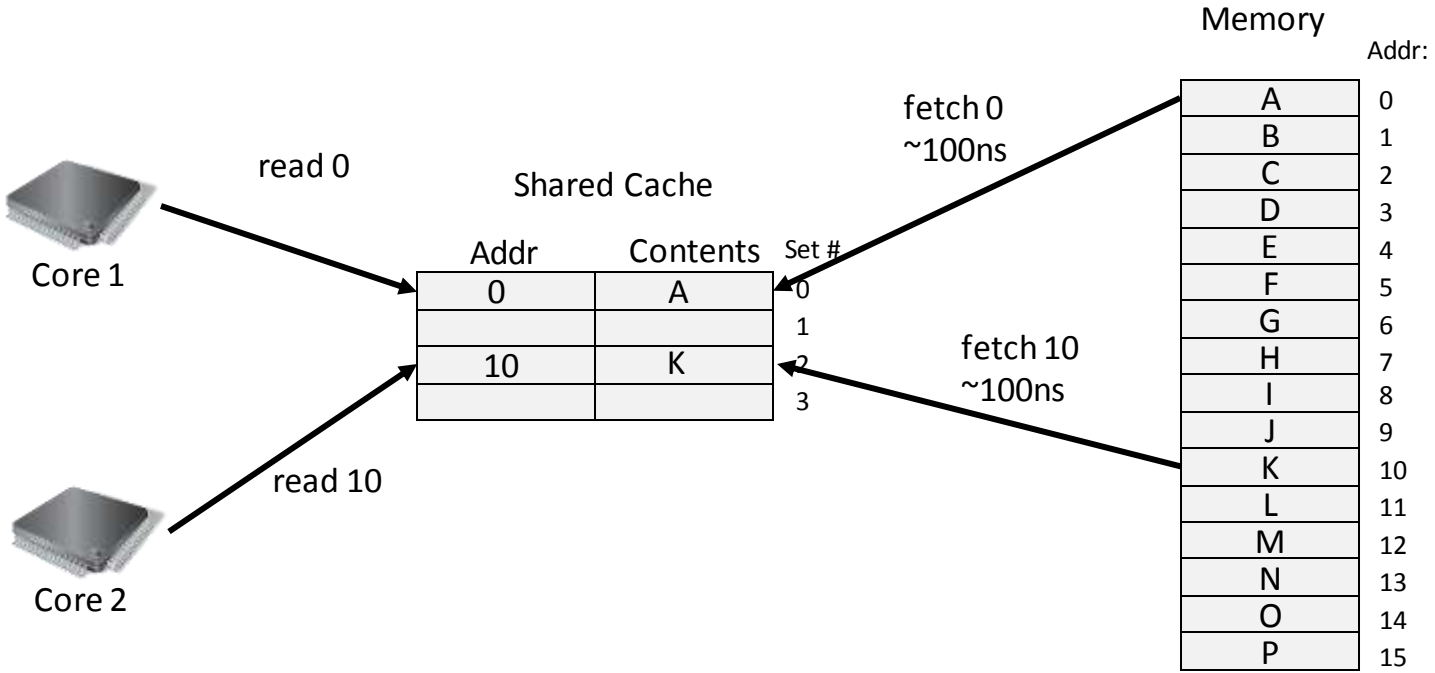
Addr	Contents	Set #
		0
		1
		2
		3

Memory

	Addr:
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11
M	12
N	13
O	14
P	15

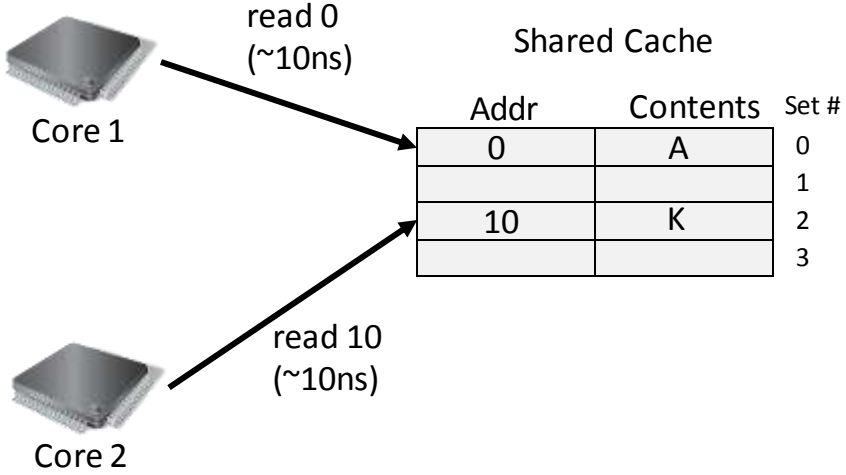
Icons credit: http://www.doublejdesign.co.uk

Shared Hardware Delays: Shared Cache



Icons credit: http://www.doublejdesign.co.uk

Shared Hardware Delays: Shared Cache

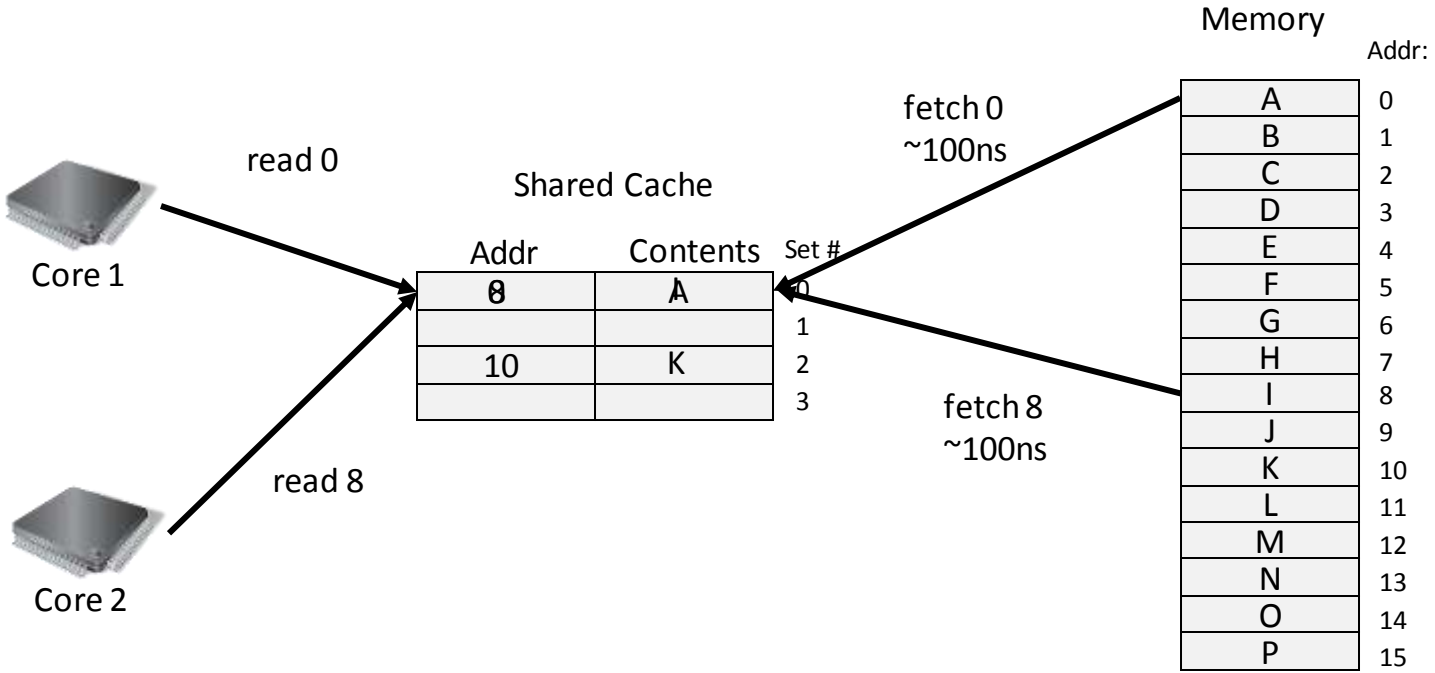


Memory

	Addr:
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11
M	12
N	13
O	14
P	15

Icons credit: http://www.doublejdesign.co.uk

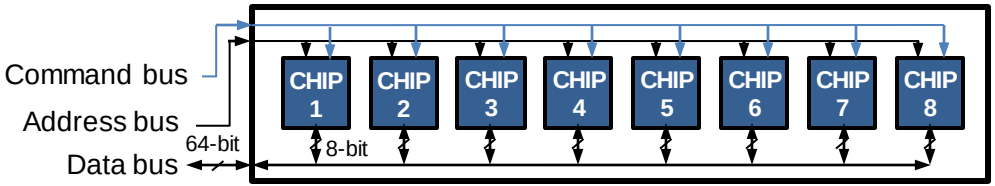
Shared Hardware Delays: Shared Cache



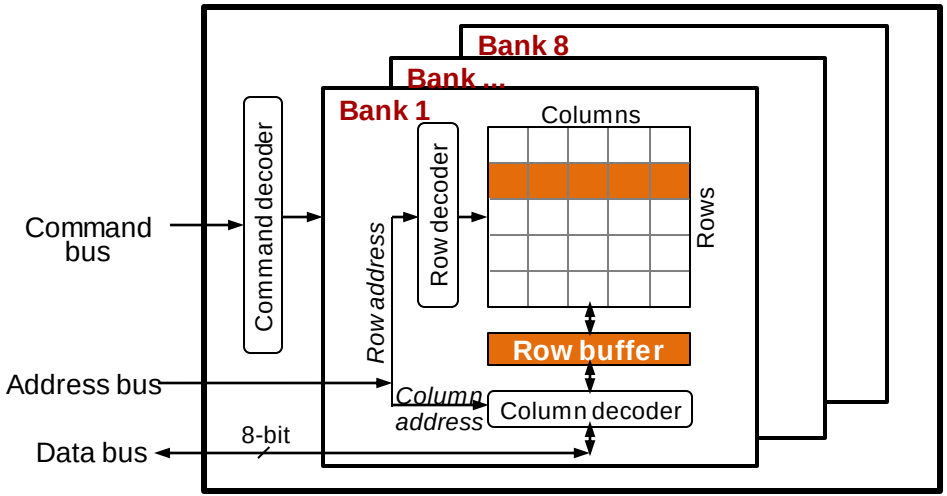
Icons credit: http://www.doublejdesign.co.uk

DRAM Memory Organization

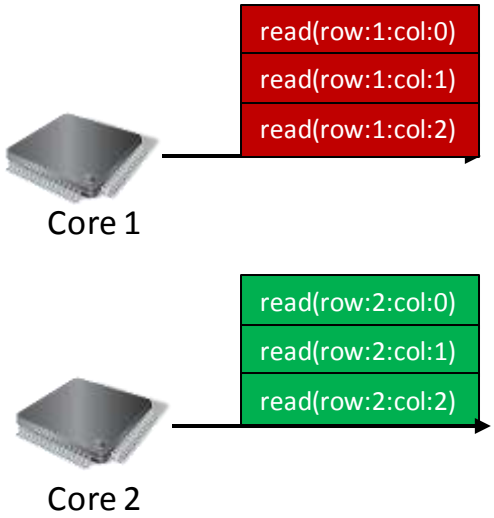
DRAM Rank

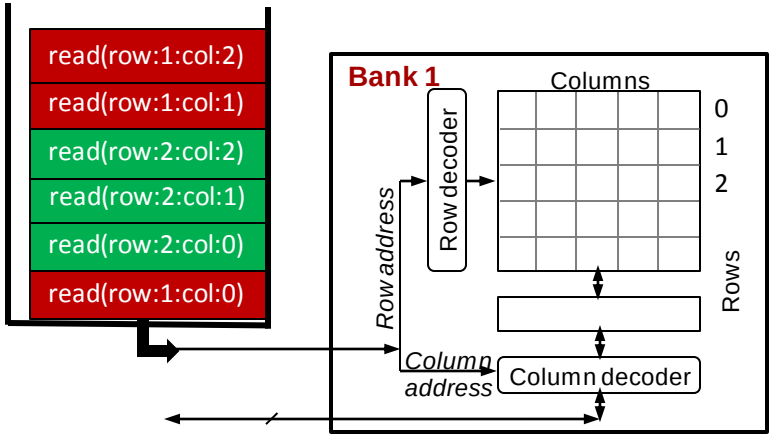


DRAM Chip



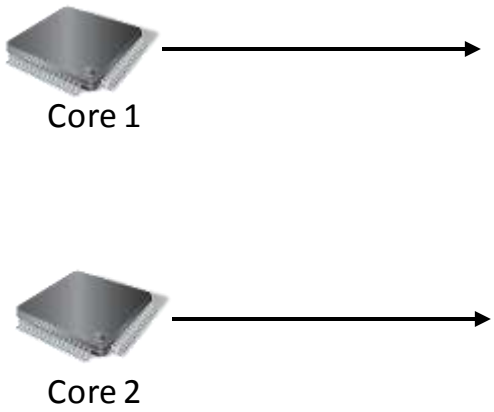
Memory Banks Delays



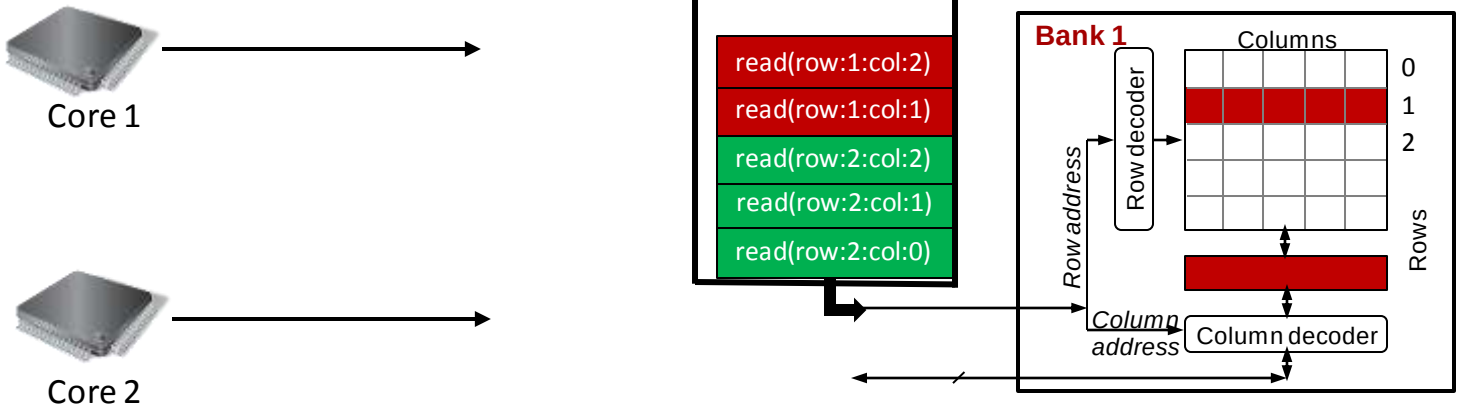


Icons credit: http://www.doublejdesign.co.uk

Memory Banks Delays

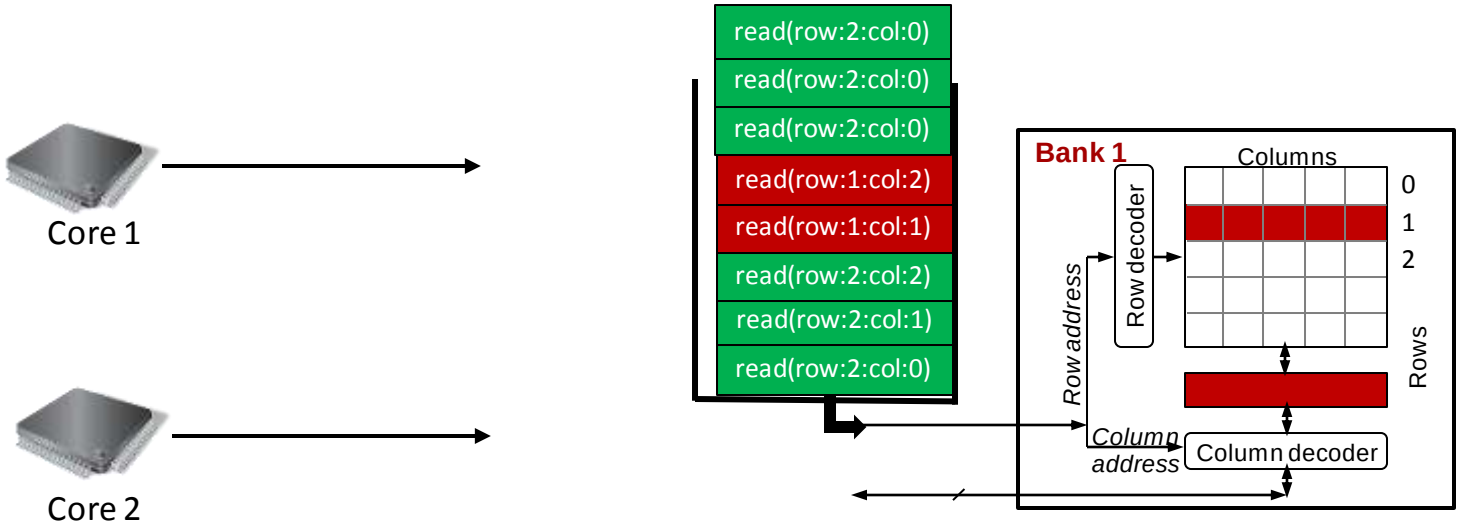


Memory Banks Delays



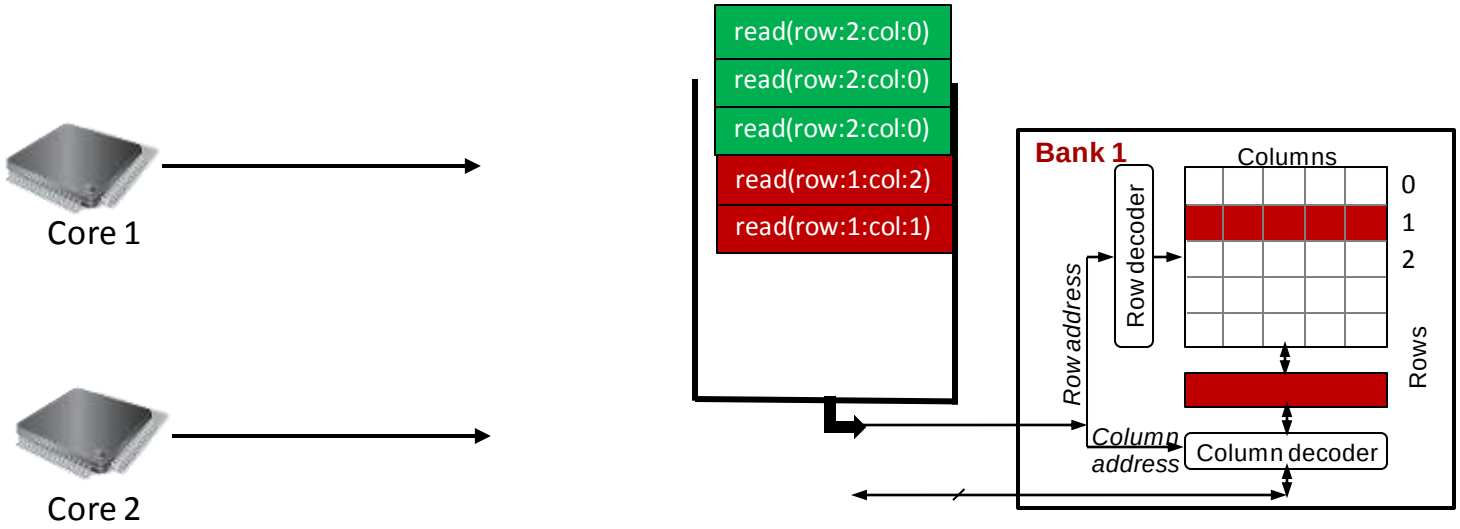
Icons credit: http://www.doublejdesign.co.uk

Memory Banks Delays



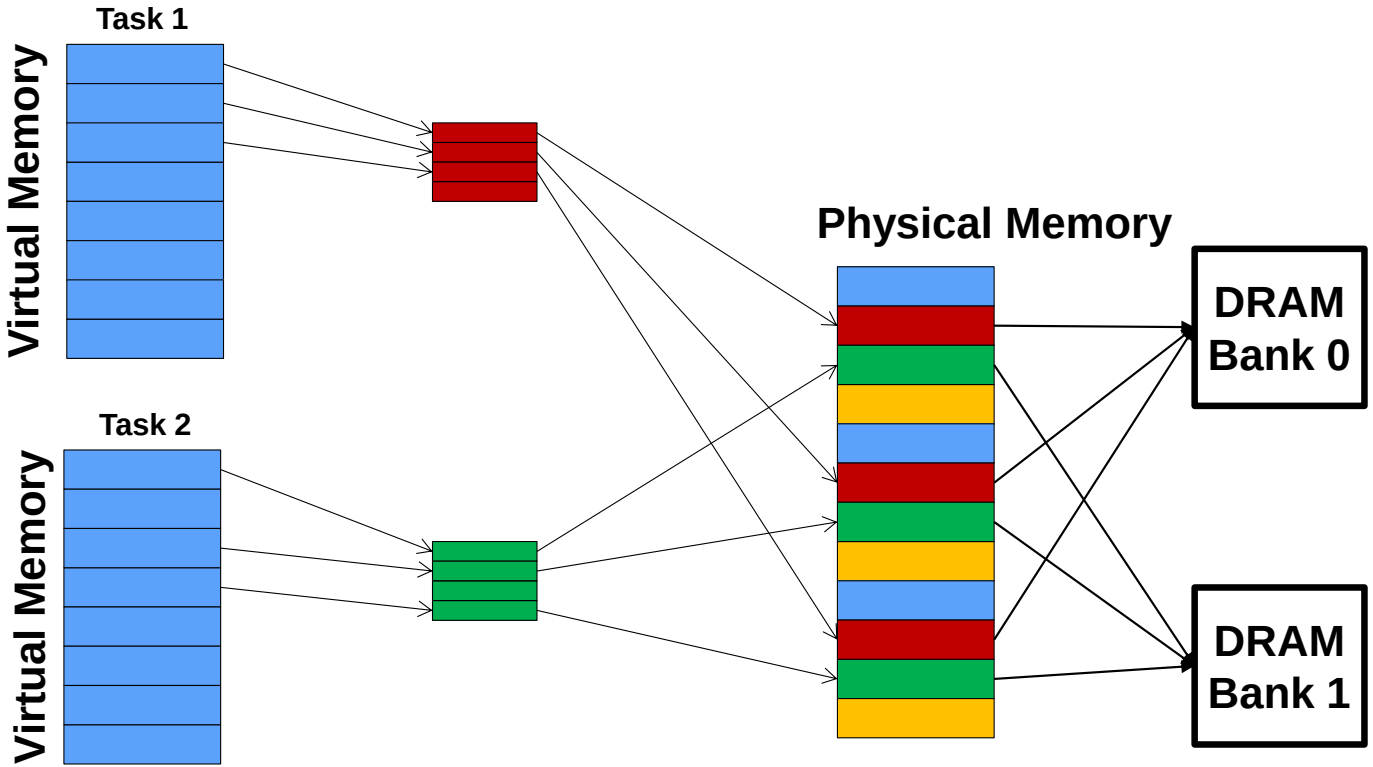
Icons credit: http://www.doublejdesign.co.uk

Memory Banks Delays

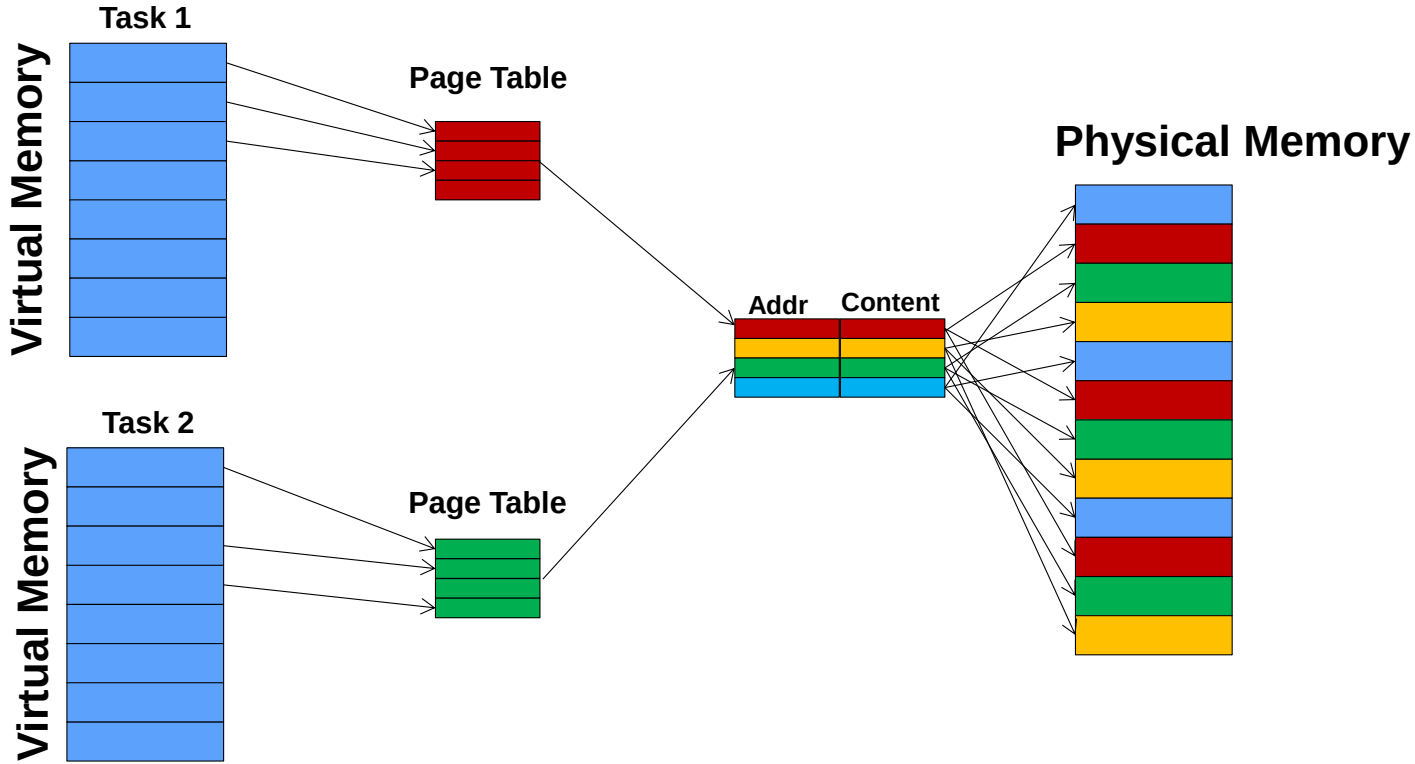


Icons credit: http://www.doublejdesign.co.uk

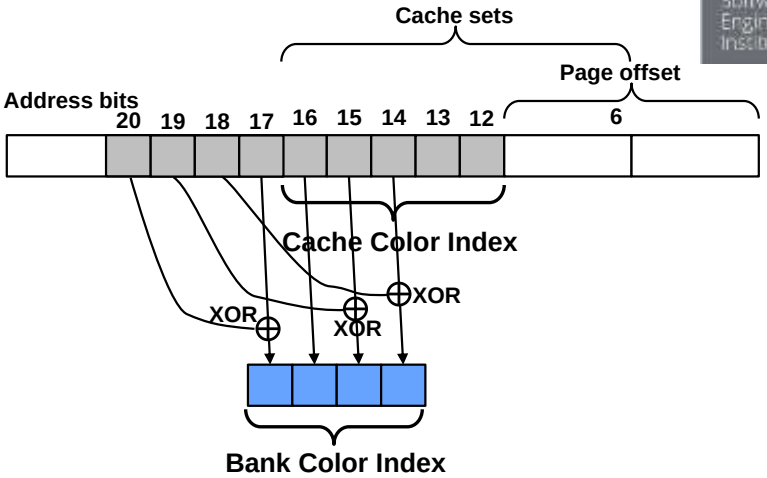
Use Virtual Memory to Map Physical Memory of Different Banks



And Different Cache Sets



Coordinated Partition Allocations



Address bit for banks may conflict with cache set bits

- **Different cache colors may share bank color**

Solution

- **Model conflicts and allocate colors to avoid them**
- **Assign: cache colors, bank colors, and cores**

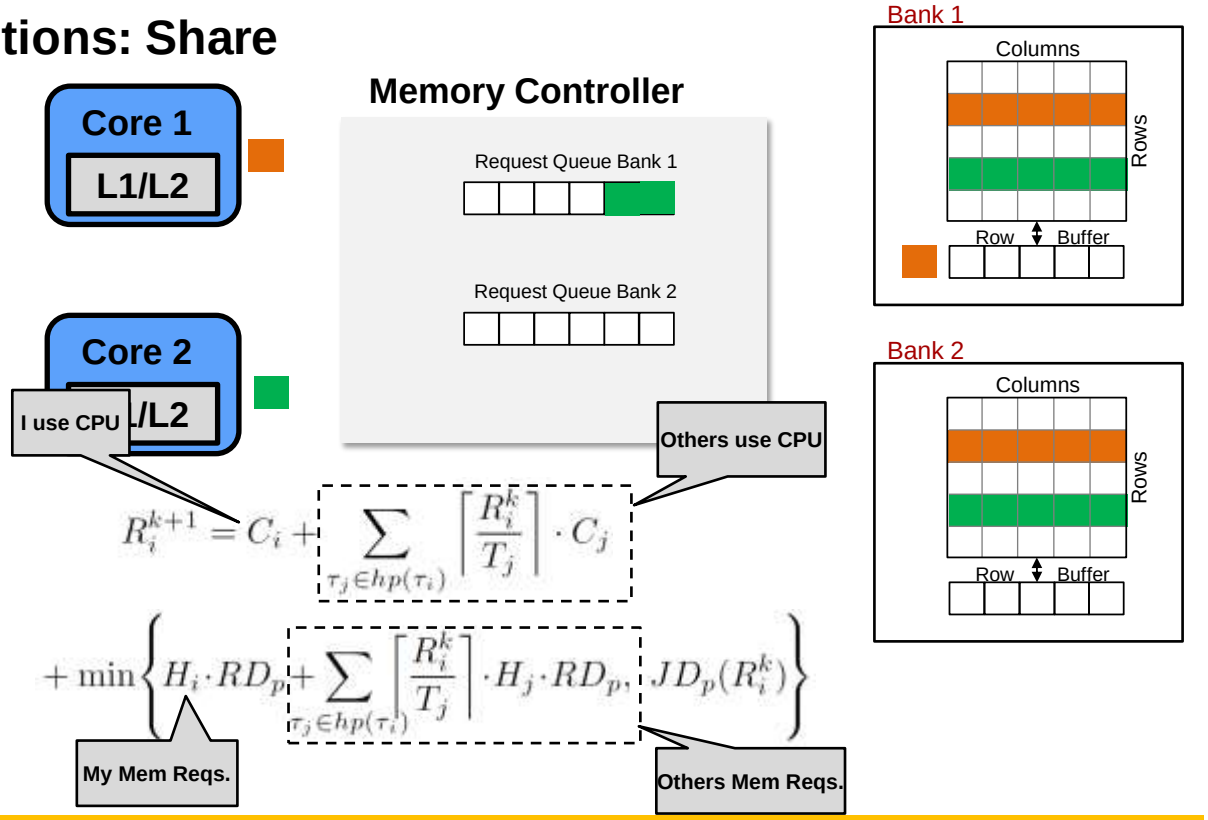
N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar. "Coordinated Bank and Cache Coloring for Temporal Protection of Memory Access". ICESS 2013

Sharing Partitions

H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. "Bounding and Reducing Memory Interference in COTS-Based Multi-Core Systems". Journal of Real-Time Systems. 2016

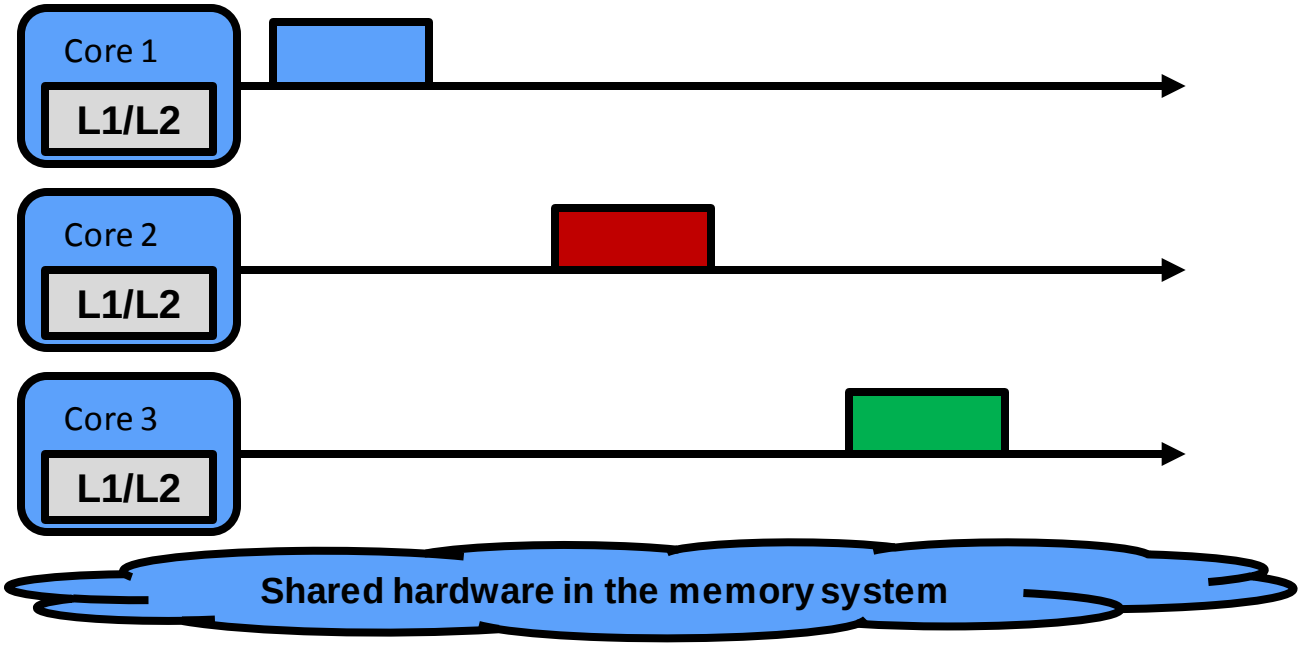
When not enough partitions: Share

But Predictably



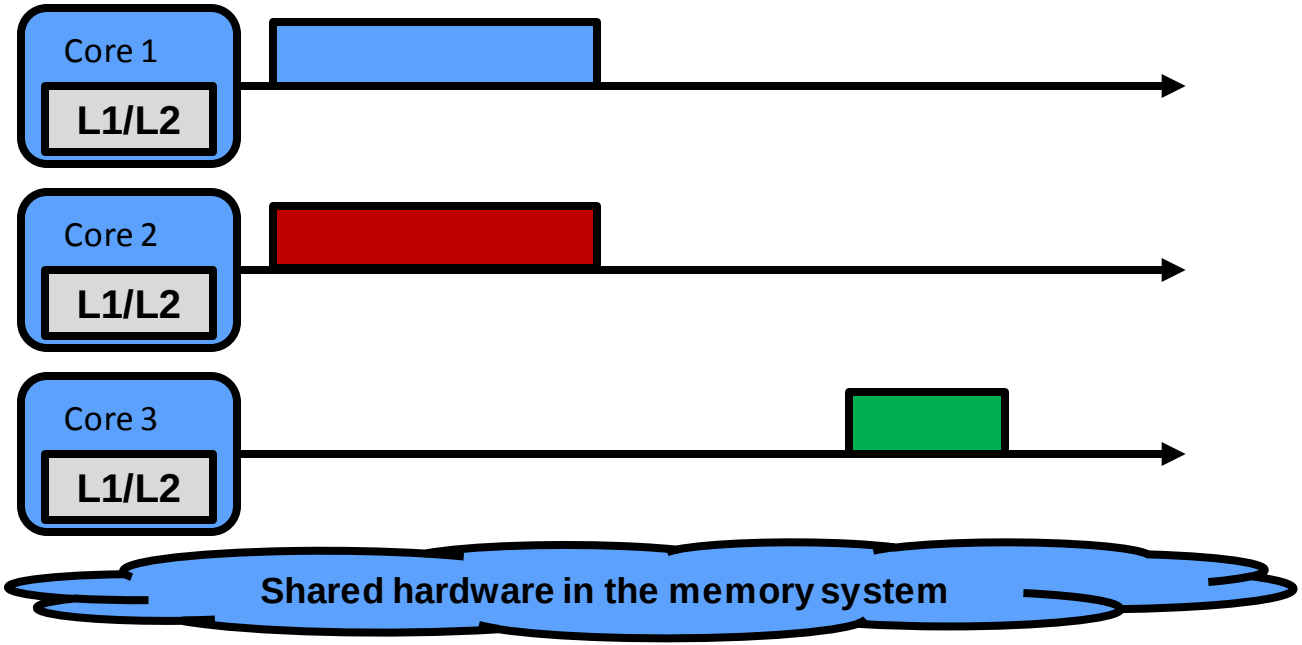
Challenge: Need Processor Documentation (not always public)

No Documentation: Co-Runner Analysis



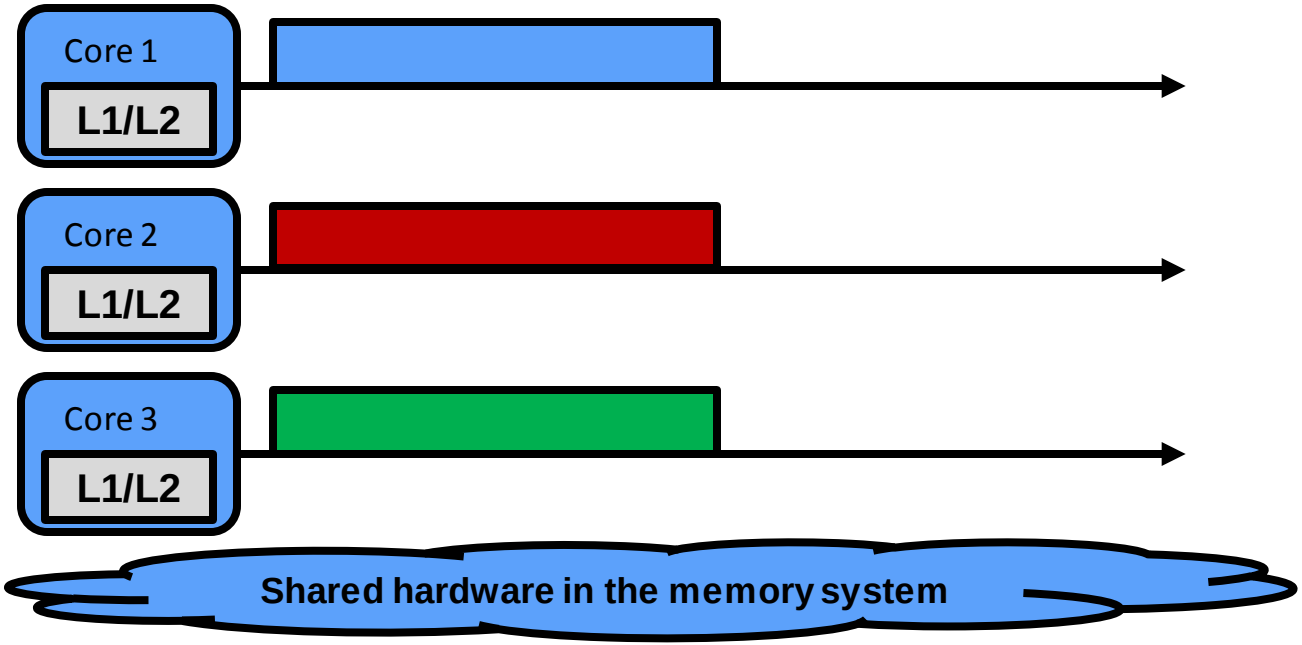
The blue, red, and green tasks execute at different times ⇒ no slowdown

No Documentation: Co-Runner Analysis



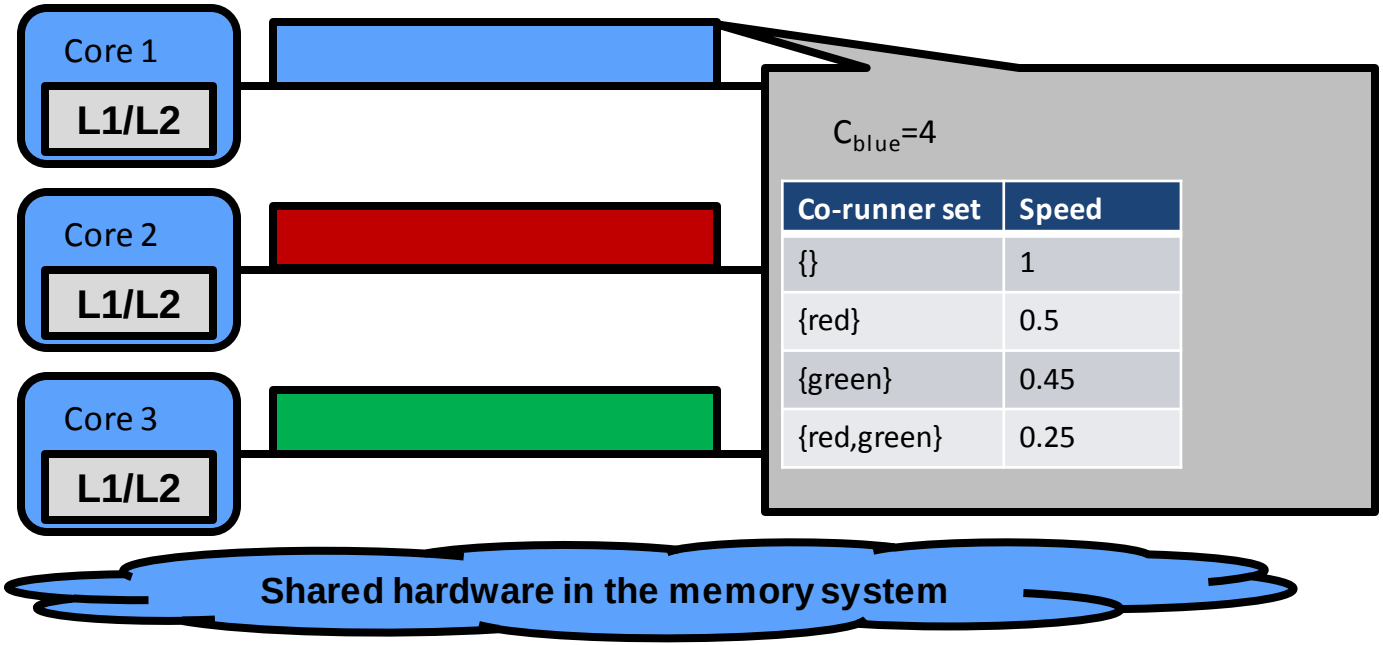
The blue and red tasks execute at
the same time ⇒ slowdown ⇒ increased execution time of blue and red.

No Documentation: Co-Runner Analysis



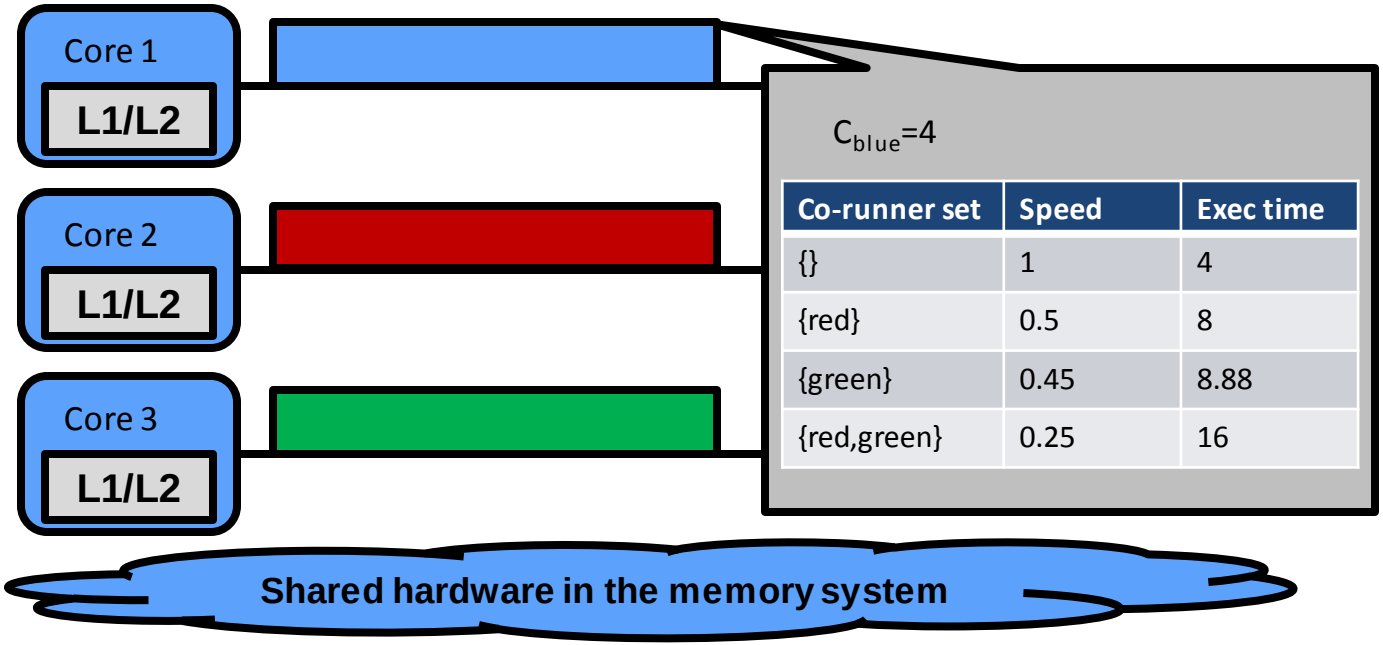
The blue, red, and green tasks execute at
the same time ⇒ slowdown ⇒ increased execution time of all tasks.

No Documentation: Co-Runner Analysis



The blue, red, and green tasks execute at
the same time ⇒ slowdown ⇒ increased execution time of all tasks.

No Documentation: Co-Runner Analysis



C_{blue}=4

Co-runner set	Speed	Exec time
{}	1	4
{red}	0.5	8
{green}	0.45	8.88
{red,green}	0.25	16

The blue, red, and green tasks execute at
the same time ⇒ slowdown ⇒ increased execution time of all tasks.

Co-runner Schedulability

B. Andersson, H. Kim, D. de Niz, M. Klein, R. Rajkumar, and J. Lehoczky. Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times ACM Transactions on Embedded Computing Systems. 2018.

How does schedulability analysis work?

Define reqlp(τ,Π,*i*,*t*) as the optimal value of the objective function of the following:

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

subject to

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

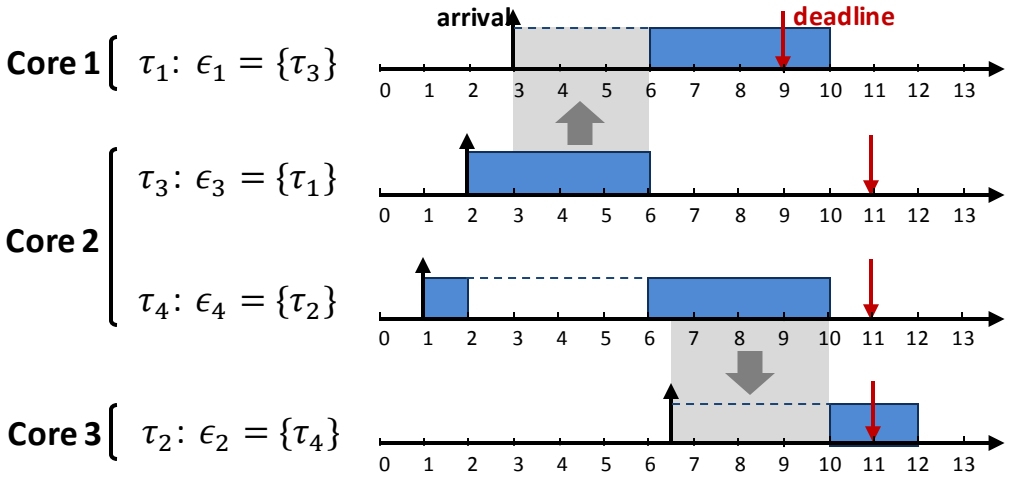
$$\forall i' \in \text{top}(\tau,\Pi,\text{i}), \forall v_{i'}^{k'} \in V_{i'}, \sum_{\substack{s \in S(\tau,\Pi,i',k') \wedge \\ (\exists \langle i'',k''\rangle \text{ s.t. } (i'' \in \text{hep}(\tau,\Pi,\text{i})) \wedge (\langle i'',k''\rangle \in s))}} \text{pw}_{i',s\setminus\{\langle i',k'\rangle\}}^{k'}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

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

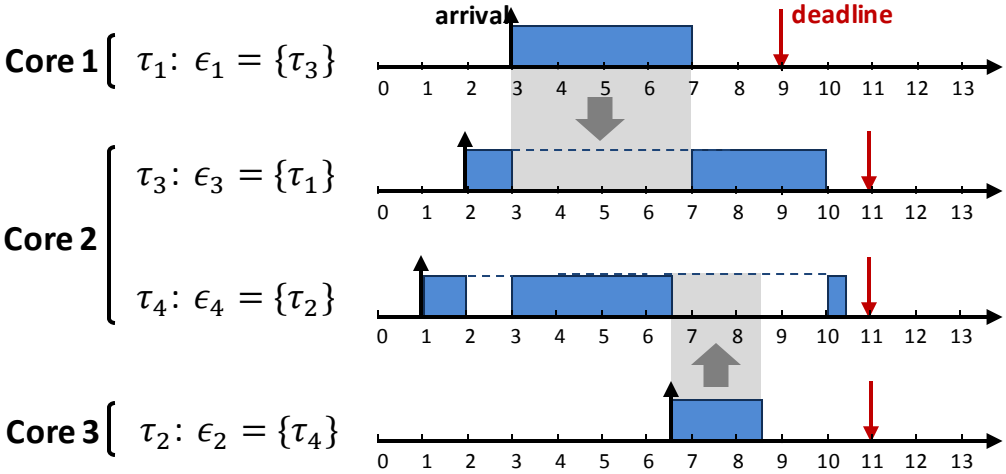
Schedulability analysis (timing verification) is done as follows:

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

Inter-Core Interference Affect Tasks Differently



Co-Runner Locker: Preempt Tasks That Lead to Deadlines Misses



$$R_i = C_i \cdot \theta_i + \sum_{\tau_j \in hpp(\tau_i)} \left\lceil \frac{R_i + I_j(C_j \cdot \theta_j)}{T_j} \right\rceil C_j \cdot \theta_j + \sum_{\tau_k \in \epsilon_i \wedge \tau_k \in hp(\tau_i)} \left\lceil \frac{R_i + I_k(C_k \cdot \theta_k)}{T_k} \right\rceil C_k \cdot \theta_k$$

where

$$I_j(x) = \begin{cases} \max(R_j - x, 0) & , \exists \tau_y : \tau_y \in \epsilon_j \wedge \tau_y \in hp(\tau_j) \\ 0 & , otherwise \end{cases}$$

New Schedulability Test

Find tasks that must not run together to prevent deadline misses

Carnegie Mellon University
Software Engineering Institute

Early Analysis: Reducing Integration Cost

RAH-66 COMANCHE SOFTWARE REWORK & INTEGRATION COSTS

Photo Credit: Boeing-Sikorsky

Two major software (SW) rebuilds occurred during development indicating significant integration issues

- **1st increment: 75% of SW replaced**
- **2nd increment: 50% of SW replaced**

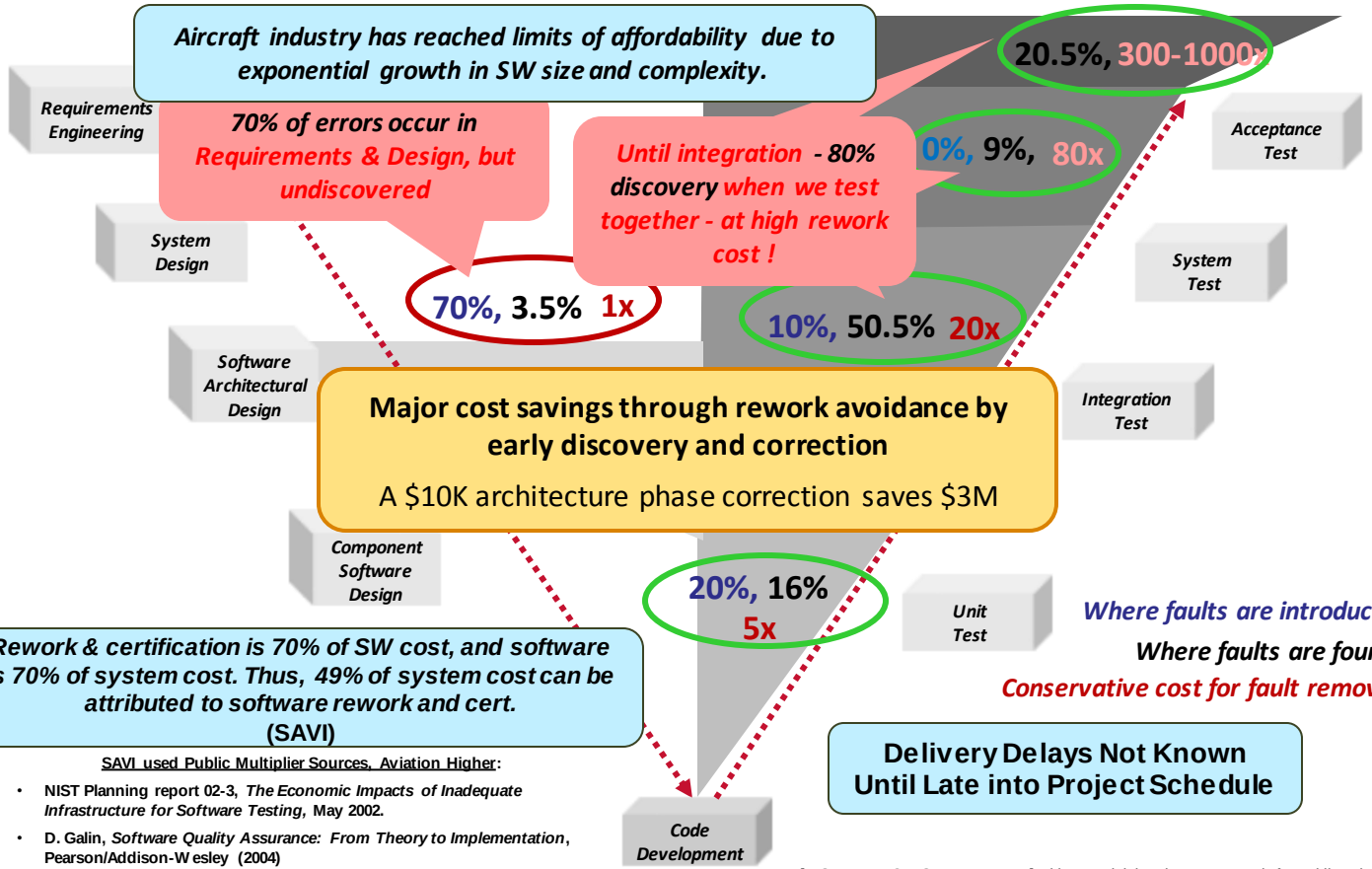
- ***In 1983, the Army planned to buy 5,023 vehicles at $12.1 million/copy.***
- ***Test schedule delays and increasing development costs scaled down the planned buy to 650 aircraft at $58.9 million/copy.***
- ***Most testing involved integration of the complete Mission Equipment Package, which incorporated a radar, infrared, and image-intensified television sensors for night flying and target acquisition.***
- ***Technical challenges remained in software development, integration of mission equipment, radar and infrared signatures, and radar perf.***
- ***The first flight had been originally planned to take place during August 1995, but was delayed by a number of structural and software problems that had been encountered.***
- ***Key program elements, including development and integration of certain software capabilities, failed to foster confidence with Army overseers; several capabilities were viewed as having been unproven and risky.***
- ***The anticipated consumption of up to 40% of the aviation budget by the Comanche alone for a number of years was considered to be extreme.***

References:

- *http://www.defense-aerospace.com/articles-view/release/3/32273/pentagon-hit-over-comanche-failings-(jan.-23).html*
- *https://en.wikipedia.org/wiki/Boeing%E2%80%93Sikorsky_RAH-66_Comanche#cite_note-26*
- *https://en.wikipedia.org/wiki/Boeing%E2%80%93Sikorsky_RAH-66_Comanche#cite_note-Eden_p139-9)*

Comanche costs were expected to consume up to 40% of US Army Aviation budget resulting in cancellation. Integration and software rework were significant cost contributors.

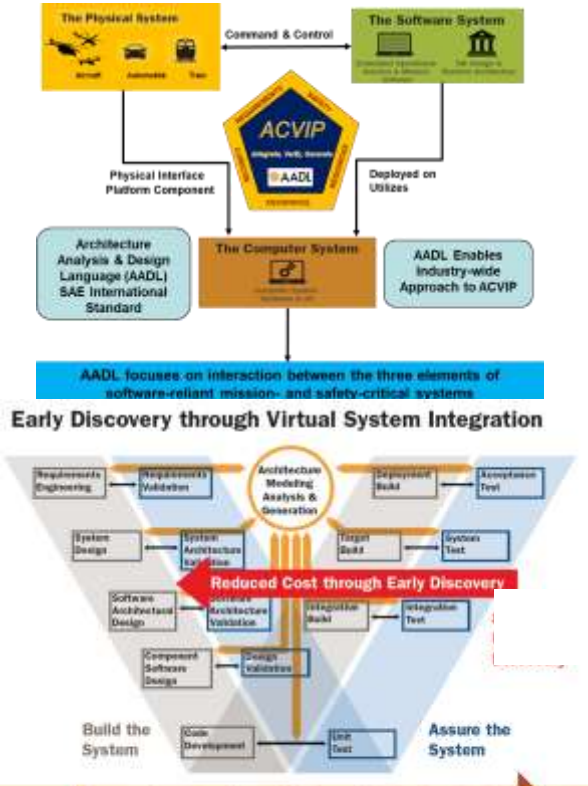
Underlying Cause – Interaction revealed Late Large SOFTWARE REWORK



SAVI used Public Multiplier Sources, Aviation Higher:

- NIST Planning report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, May 2002.
- D. Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson/Addison-Wesley (2004)
- B.W. Boehm, *Software Engineering Economics*, Prentice Hall (1981)

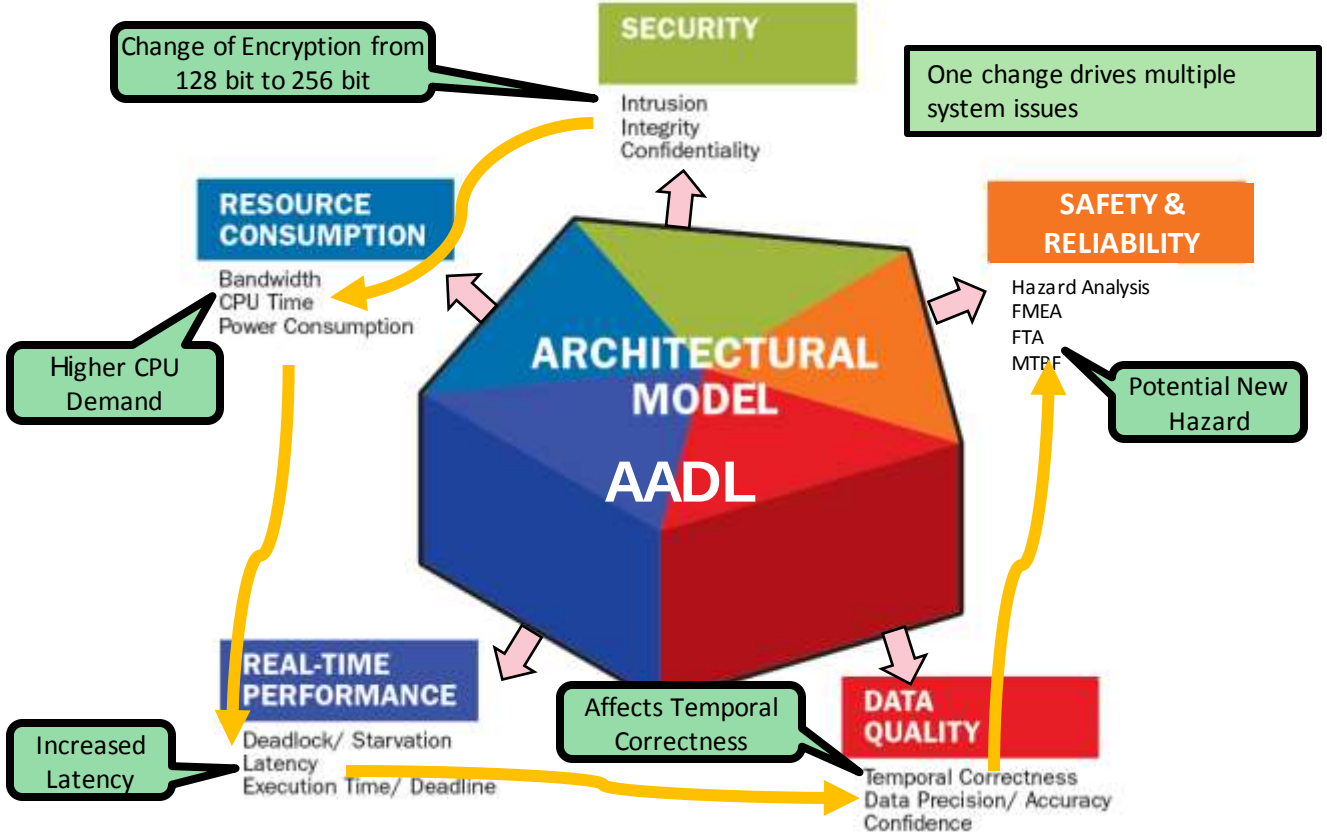
ARCHITECTURE CENTRIC VIRTUAL INTEGRATION Practice (ACVIP)



- **Leverages** research from the AVSI **SAVI** consortium which used virtual integration find issues early in commercial aviation systems
- **Utilizes architecture models** to perform **virtual integration focusing on** software-intensive parts of real-time safety- and security-critical **computing systems** to **identify issues early before integration.**
- **Important enabler for MOSA** to provide a standard analyzable and processable architecture description for embedded systems.
- **Process** (from ACVIP Modeling & Analysis Handbook)
  1) **Develop ACVIP Management Plan**
  2) **Establish Model Structure**
  3) **Define Model Content Needed for Analysis**
  4) **Incrementally Execute Analyses, Resolve**
  5) **Build System in Conformance to Models**
  6) **Support Certification and Readiness Reviews**
- **Supports architecture-based incremental and compositional modeling & analysis** of computing system properties
- Analytical results + automated build support **increasing**

Virtual Integration of Software, Hardware, and System supporting verification, airworthiness, safety and cybersecurity certification

ANALYSIS OF SYSTEM PROPERTIES VIA ARCHITECTURE MODEL



Concluding Remarks

Carnegie Mellon University Software Engineering Institute

Enforcement-Based Verification for Scalable Verification of CPS

- **Minimize what is verified**
- **Monitor and enforce safety properties with verified enforcers**
  - Logic, Time, Physics
- **Verify and Protect Enforcers**

Predictable Real-Time Execution in Multicore Processors

- **Shared Hardware and Partitioning Mechanisms**
- **Coordinated Configuration conflicting partitions to meet deadlines**
- **Shared Partitions**
- **Undocumented Hardware**

Early Analysis to Prevent Integration Cost (Virtual Integration)

- **Virtual Integration with Analysis of Architectural Models**
- **Models that can be refined down to implementation (e.g., code generation)**