THE MULTICORE CHALLENGE IN ASSURED AUTONOMY

Bjorn Andersson, Dionisio de Niz, and Mark Klein

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM21-0246

Why Troops Don't Trust Drones

17 drone disasters that show why the FAA hates drones

White House Drone Crash Described as a U.S. Worker's Drunken Lark

Boeing 737 Max: Software patches can only do so much

Systems architects, engineers, and management can all learn from the history of the development of this complex aircraft.

RQ-4B GLOBAL HAWK ACCIDENT INVESTIGATION RELEASED

Drone Crash in Iran Reveals Secret U.S. Surveillance Effort

Robotic surgery linked to 144 deaths in the US

Death by robot: the new mechanised danger in our changing world

As the use of autonomous machines increases in society, so too has the chance of robot-related fatalities

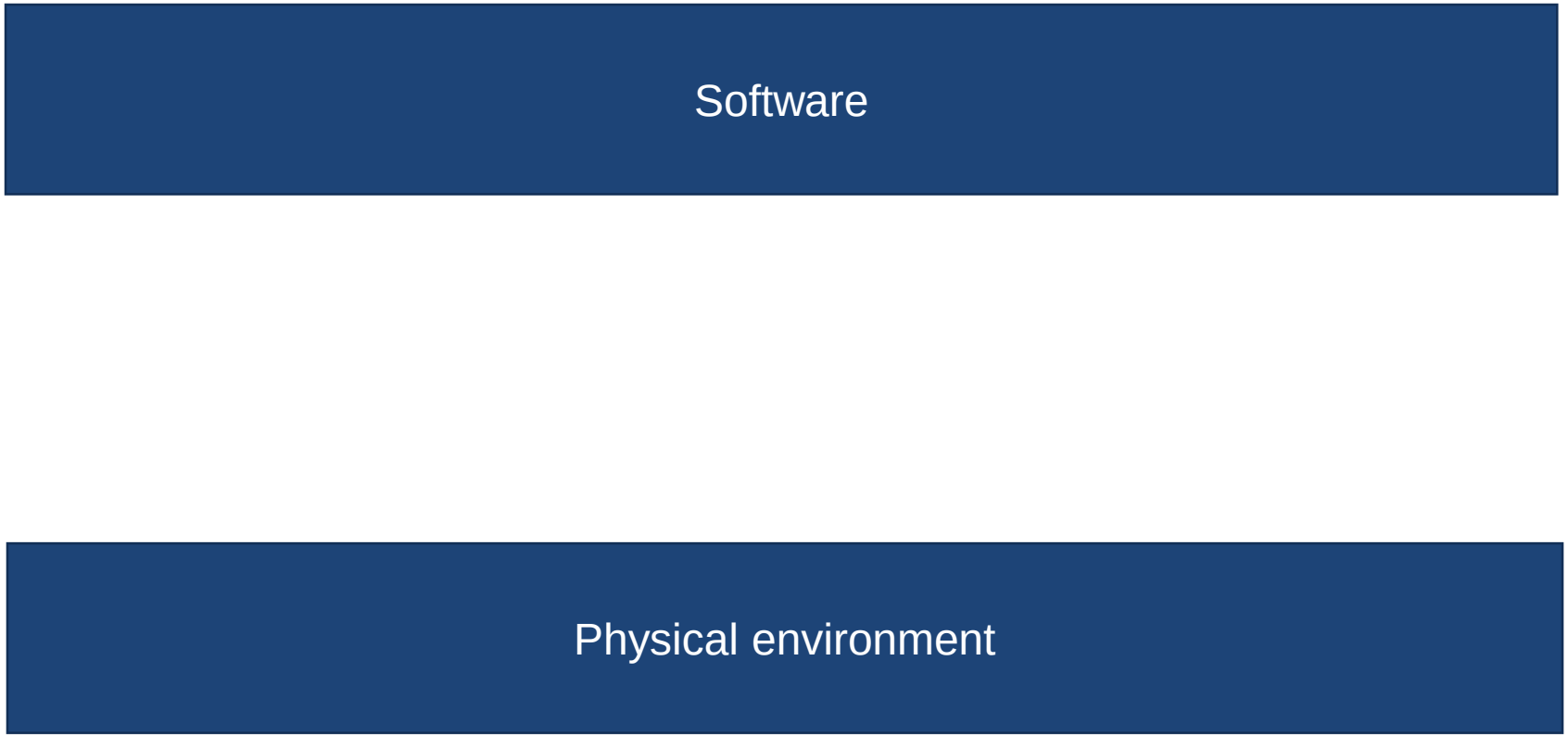
Faulty pacemakers 'killing 2,000 a year': Third of unexpected deaths among patients thought to be caused by malfunctions

ROBOT CANNON KILLS 9, WOUNDS 14

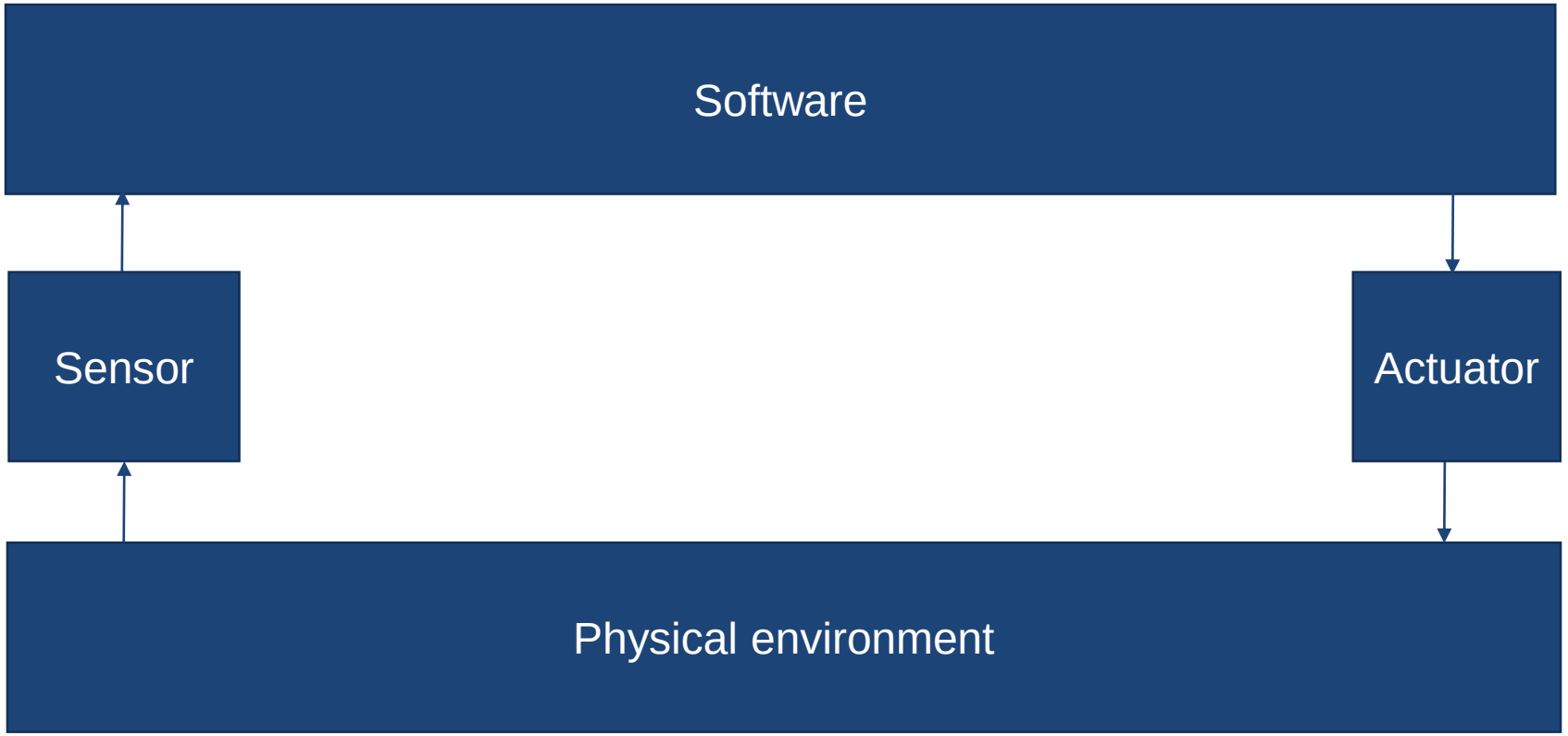
Robot 'goes rogue and kills woman on Michigan car parts production line'

The value that autonomous systems generate to society is limited by their lack of safety.

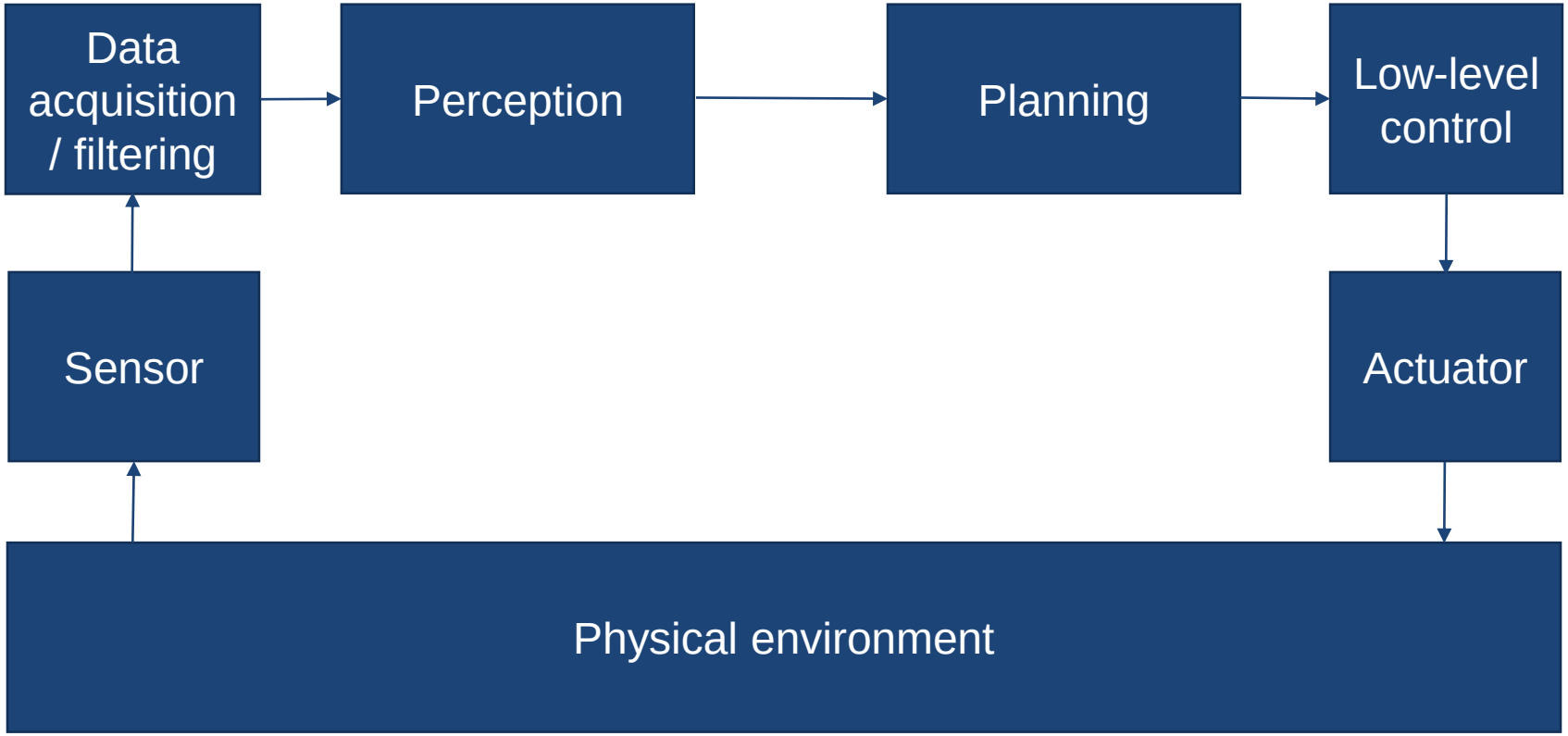
Autonomous Systems



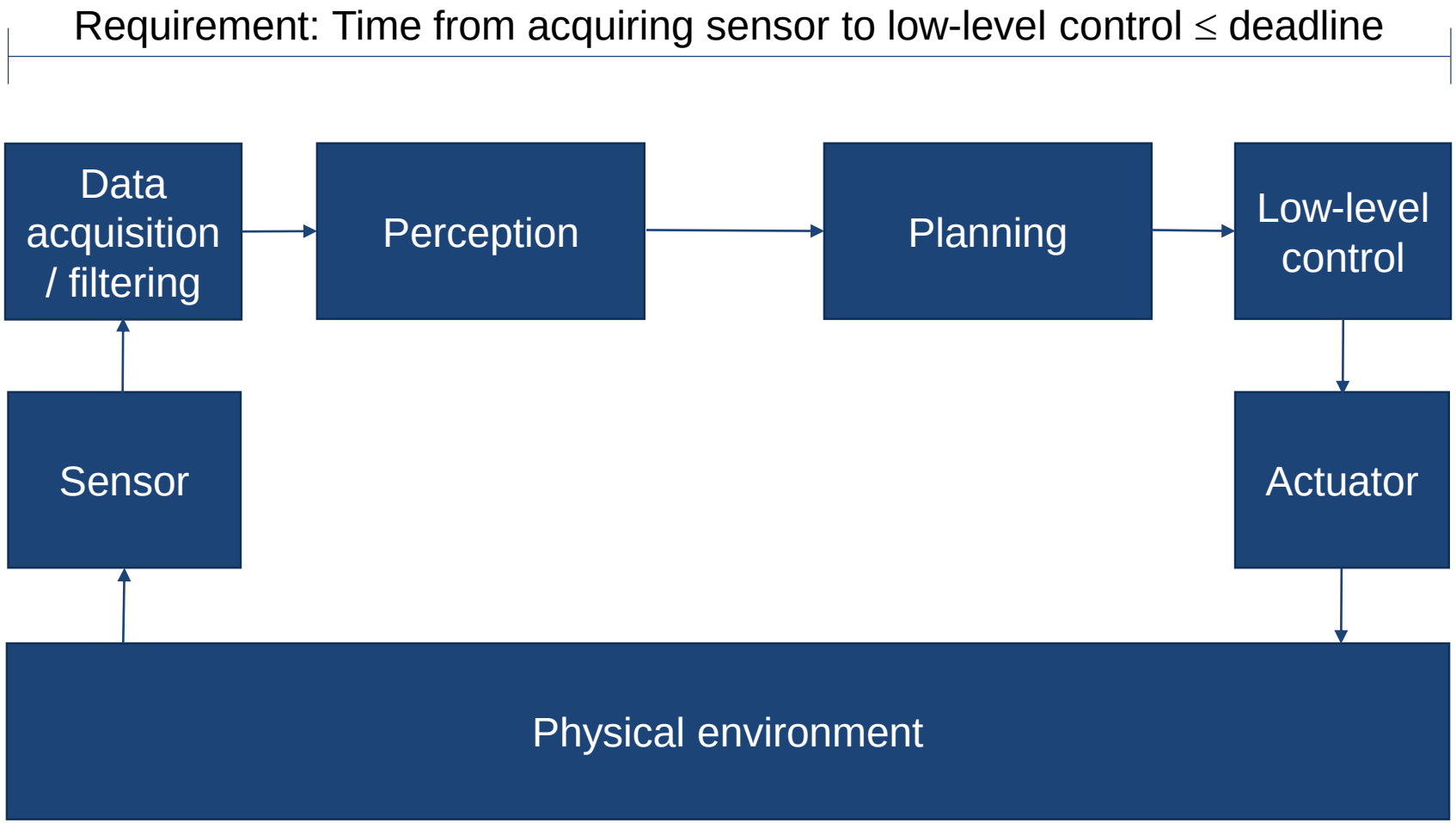
Autonomous Systems



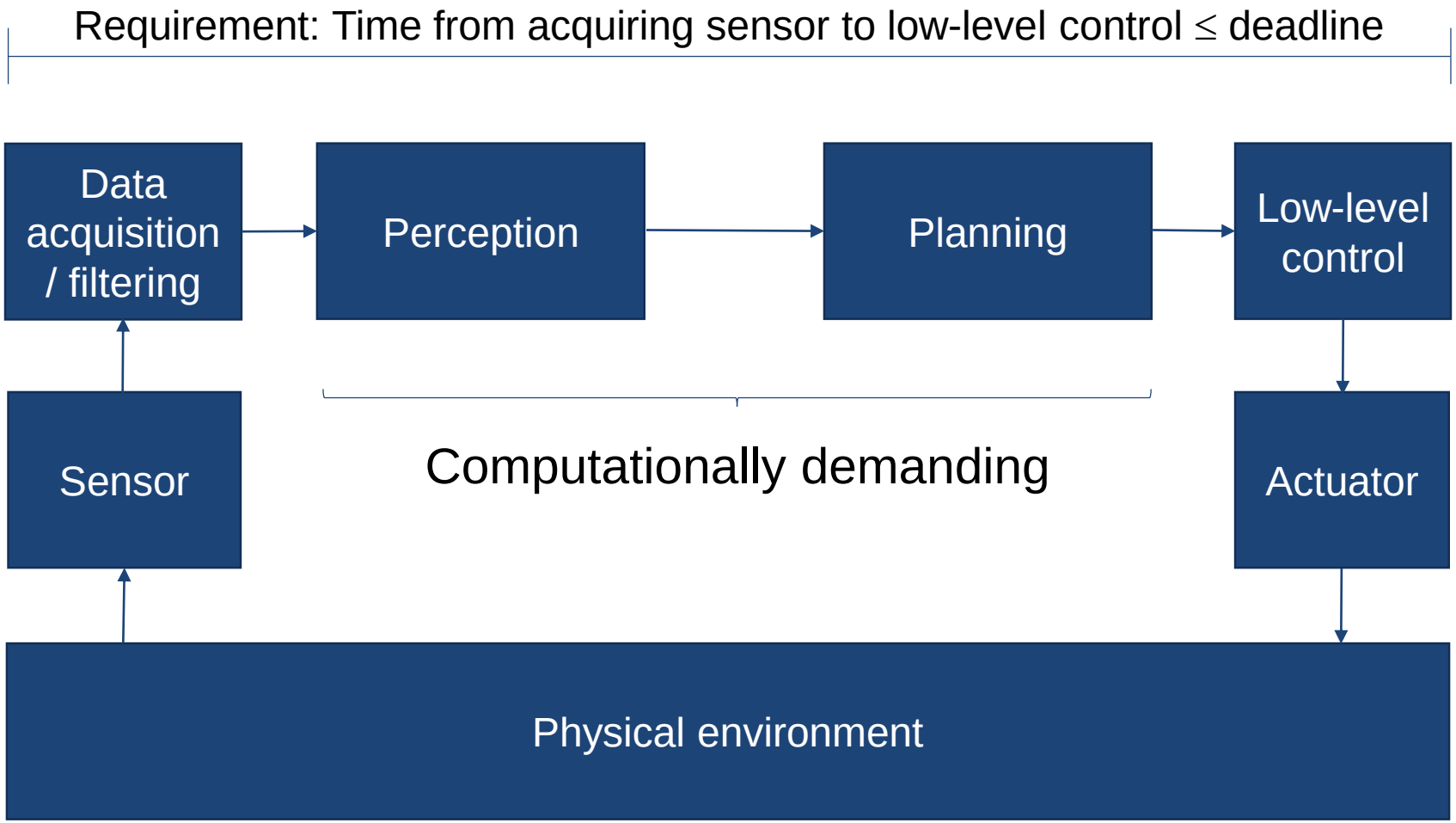
Autonomous Systems



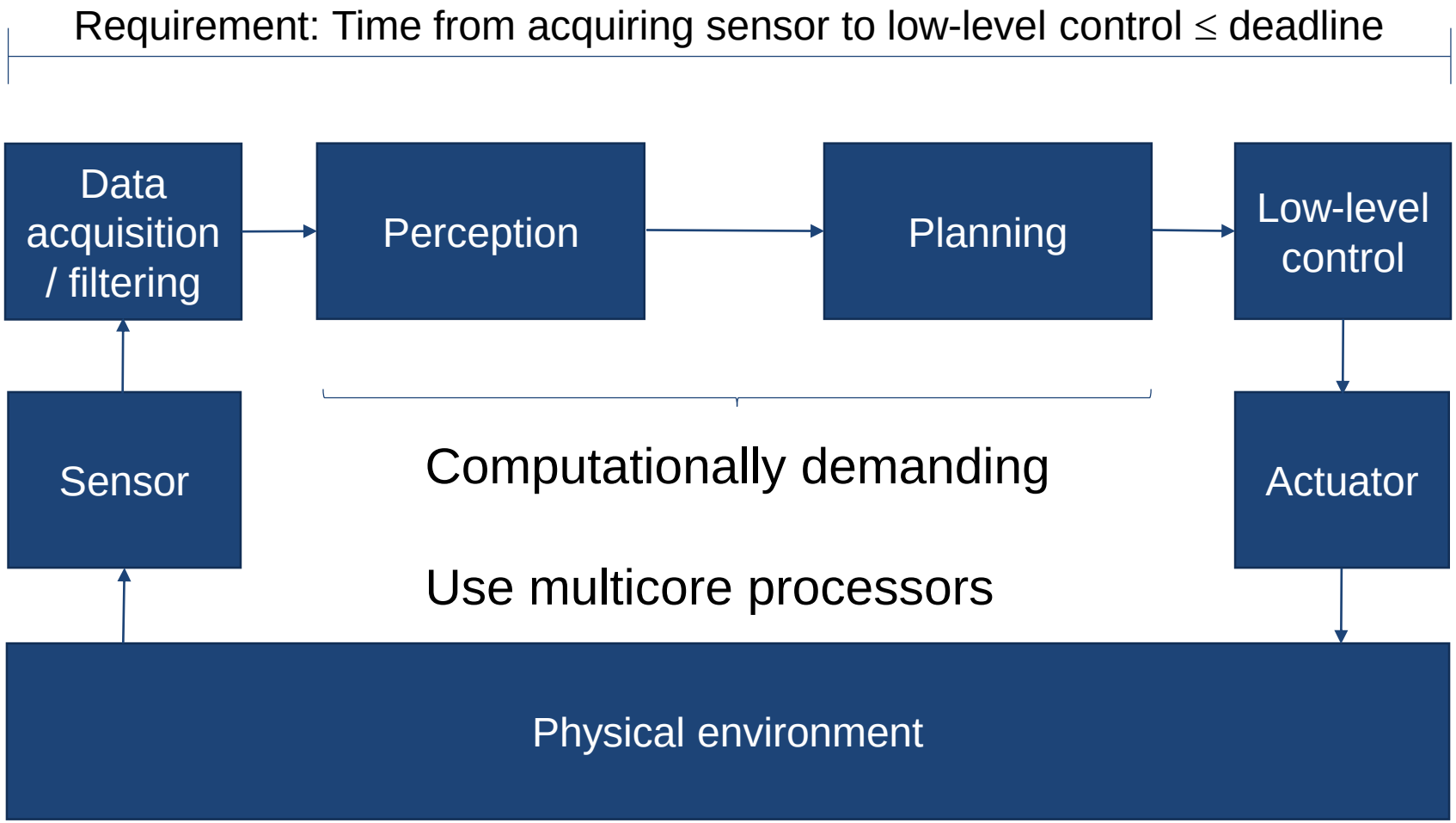
Autonomous Systems



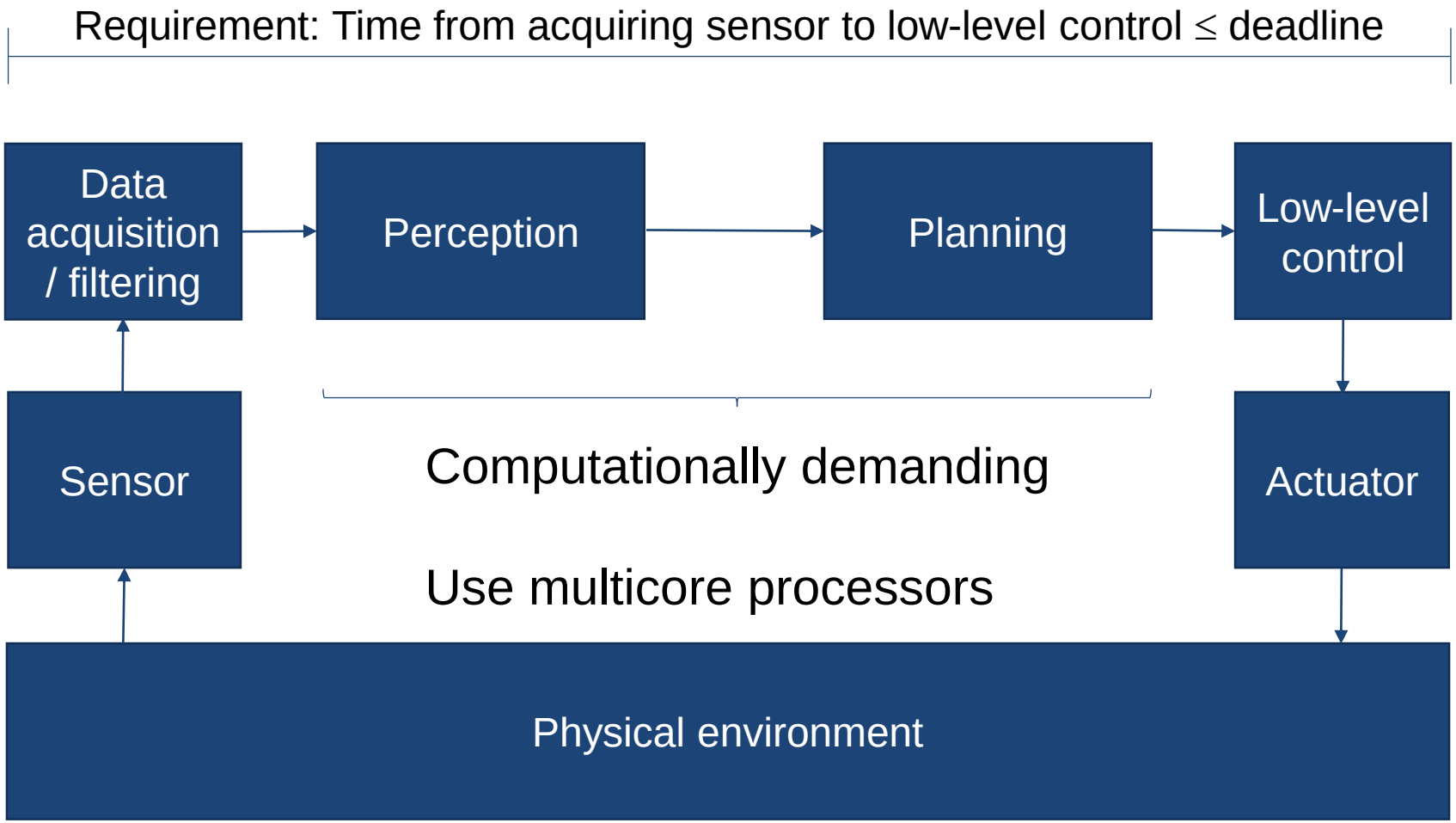
Autonomous Systems



Autonomous Systems

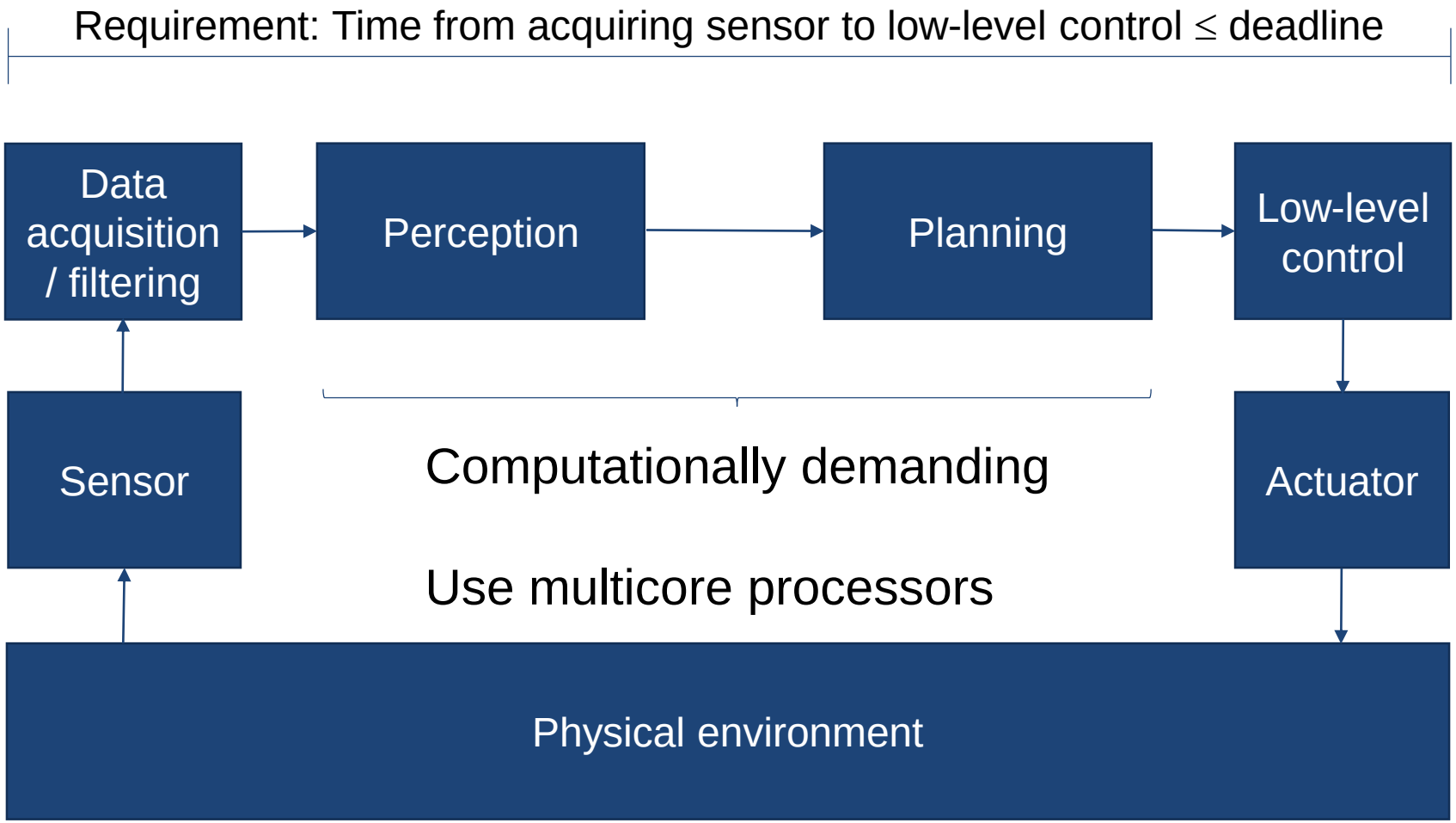


Autonomous Systems



How to satisfy real-time requirements of software executing on multicore processors?

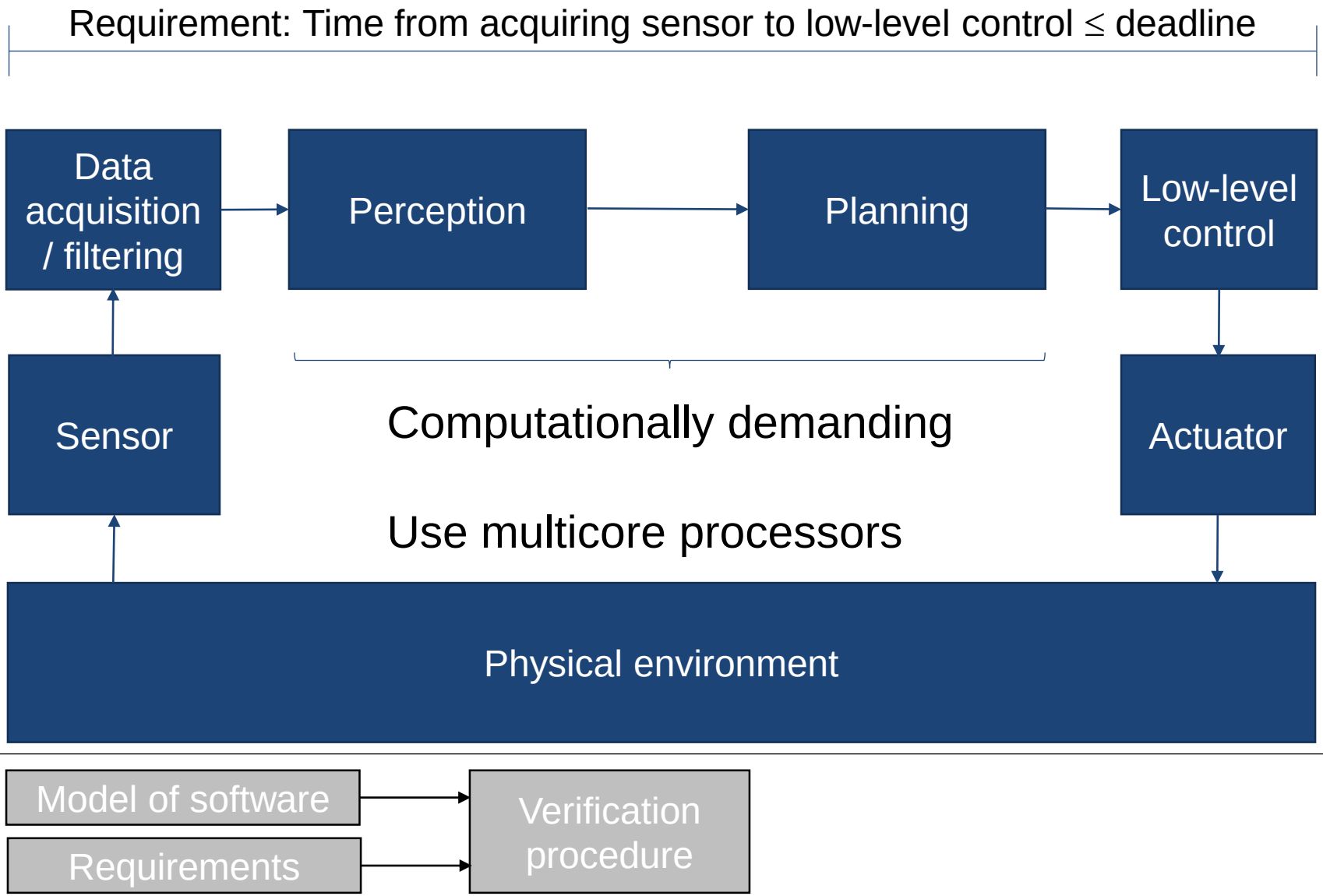
Autonomous Systems



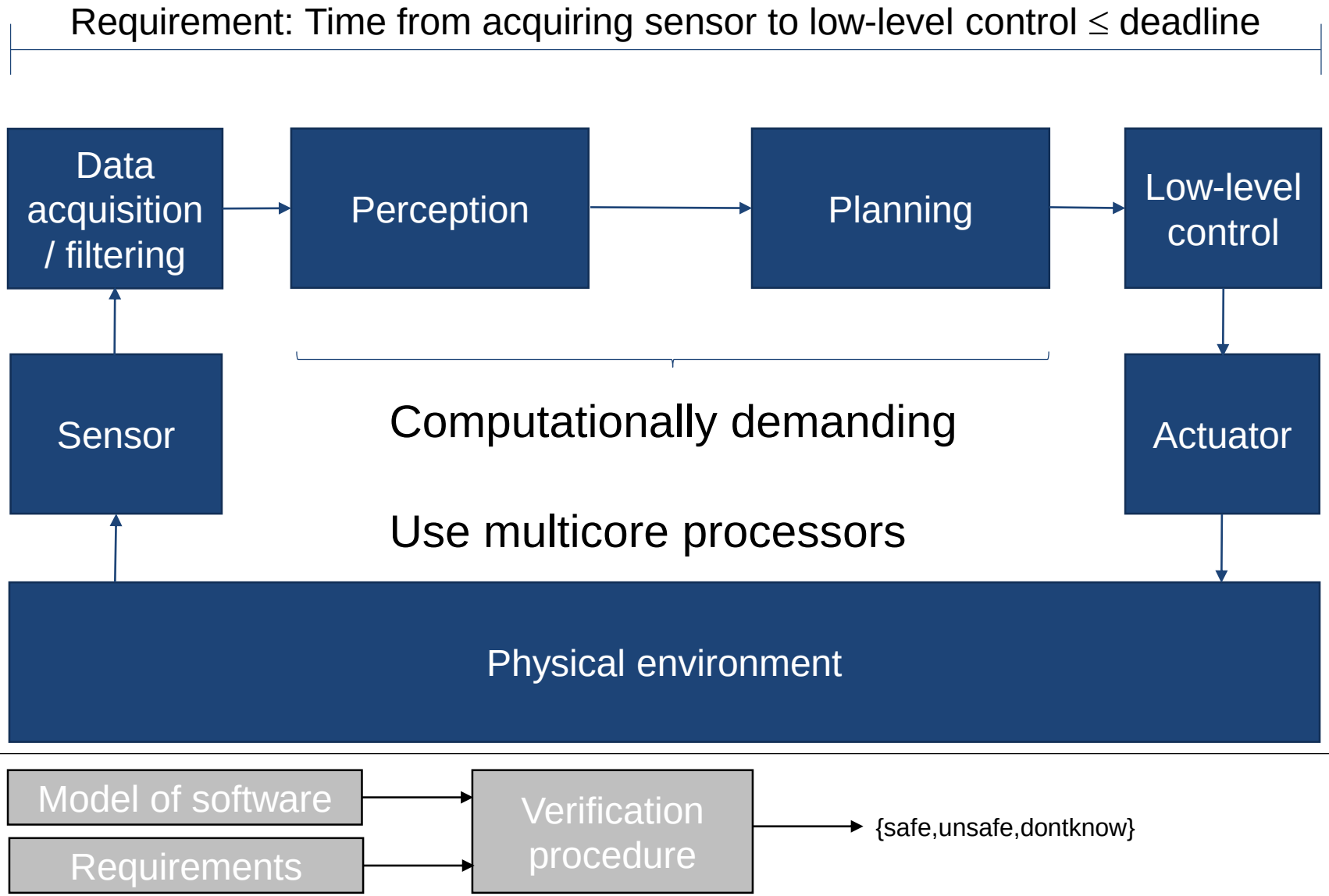
Model of software

Requirements

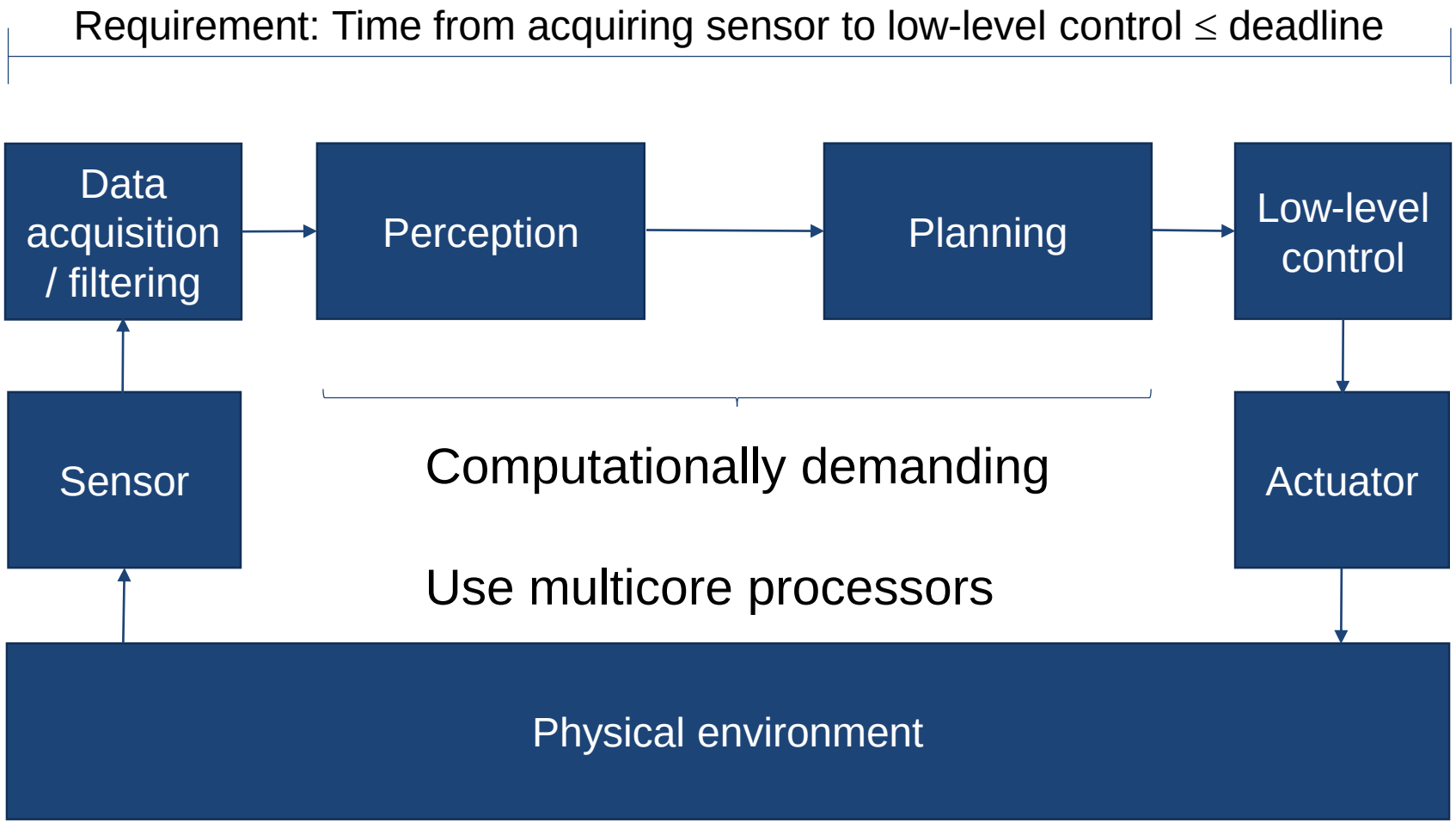
Autonomous Systems



Autonomous Systems



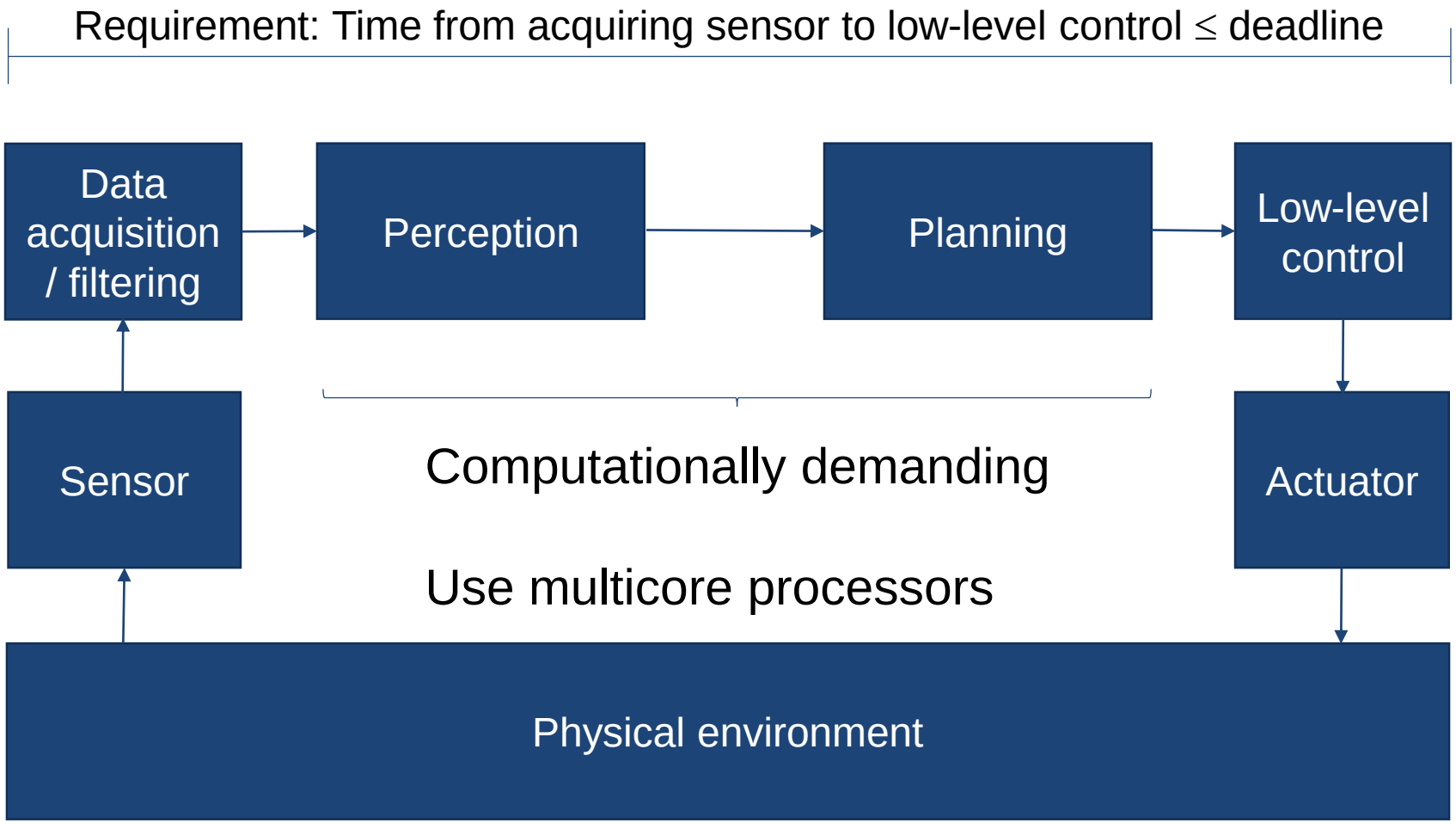
Autonomous Systems



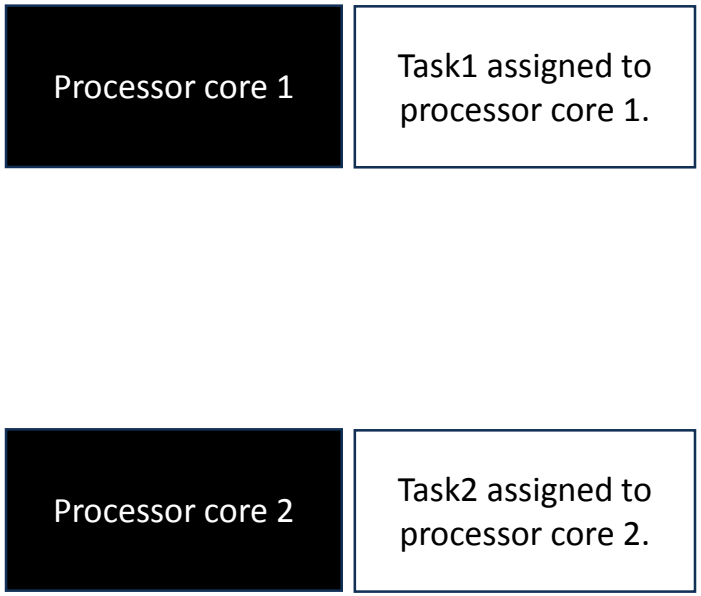
How to choose a model software?
How to design a timing verification procedure?

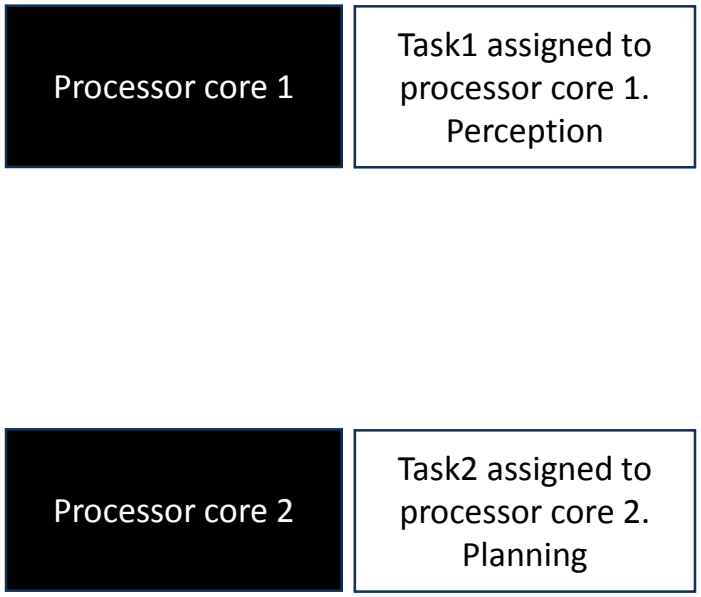
Challenges: Undocumented hardware
Timing verification ≠ logical verification

Autonomous Systems

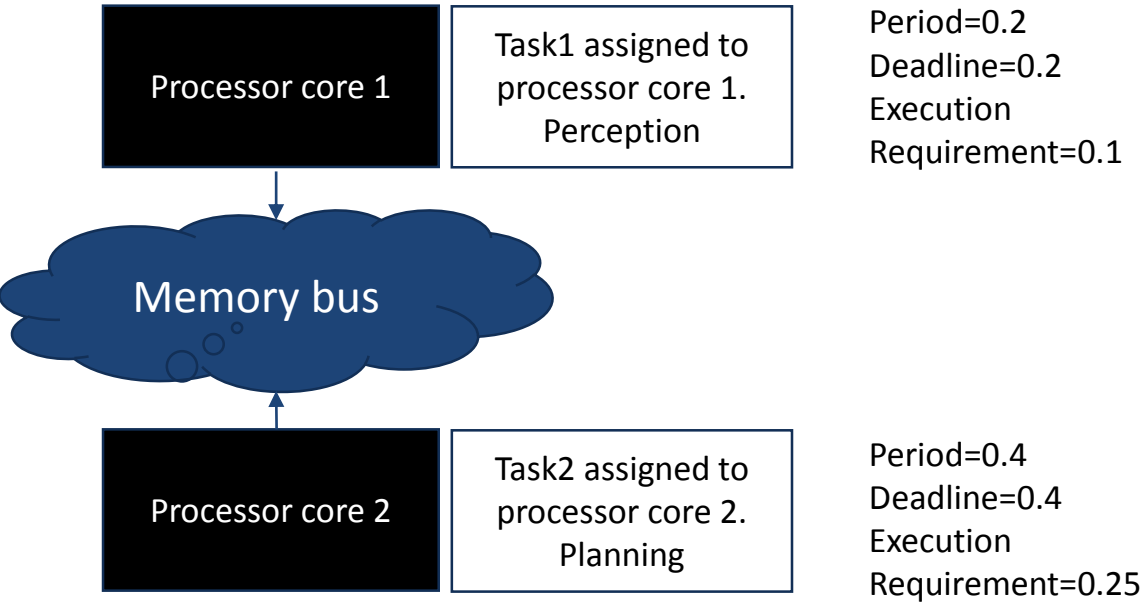


Let us discuss modelling of software: Tasks and Processors.



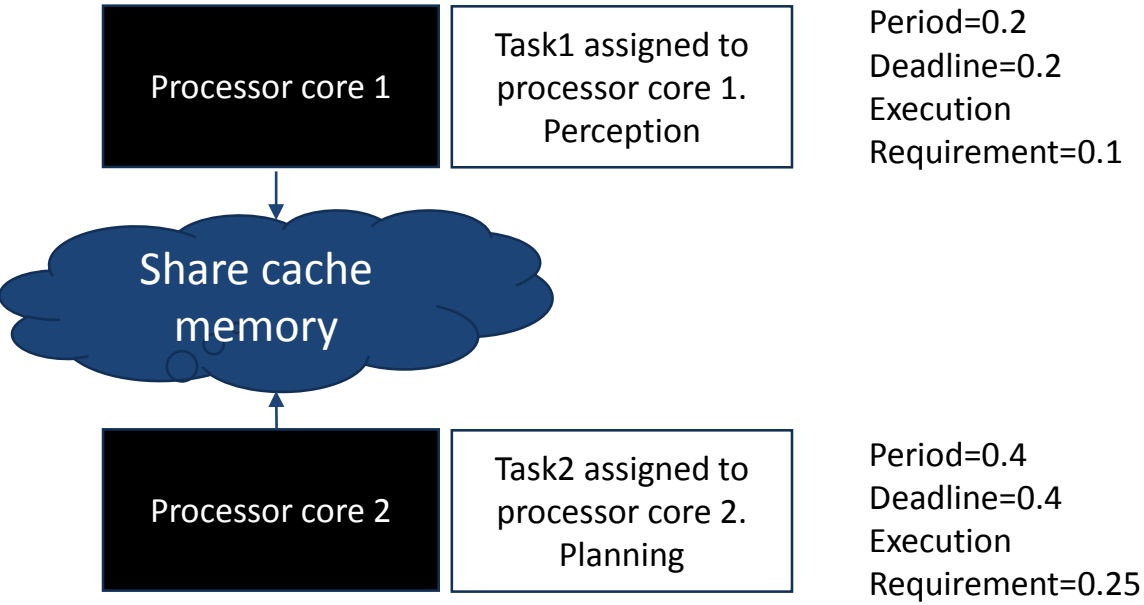


Processor core 1	Task1 assigned to processor core 1. Perception	Period=0.2 Deadline=0.2 Execution Requirement=0.1
Processor core 2	Task2 assigned to processor core 2. Planning	Period=0.4 Deadline=0.4 Execution Requirement=0.25



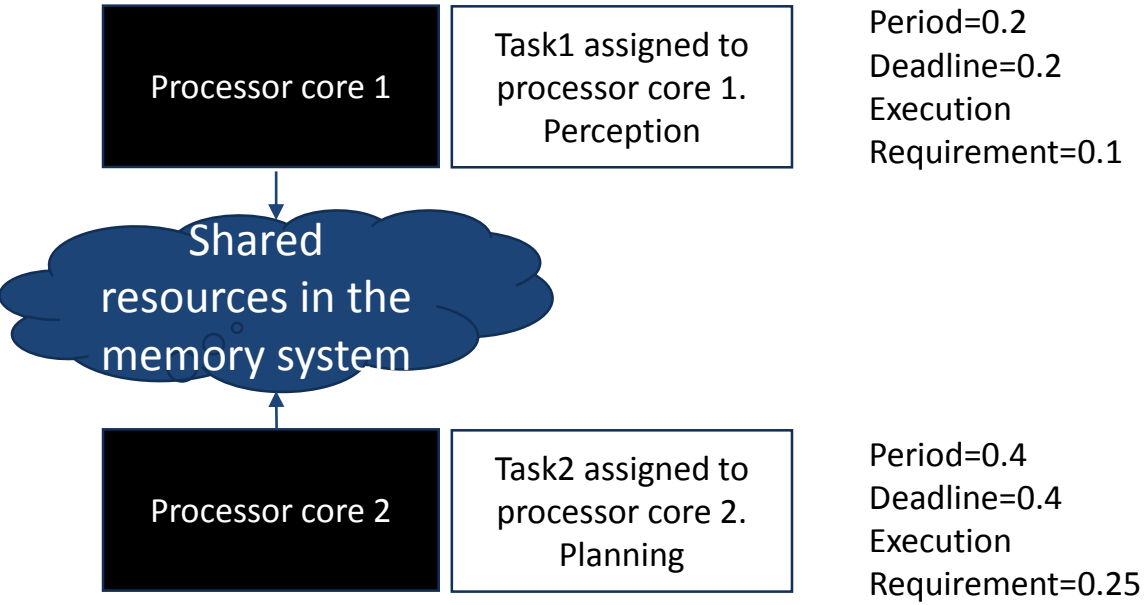
Period=0.2
Deadline=0.2
Execution Requirement=0.1

Period=0.4
Deadline=0.4
Execution Requirement=0.25



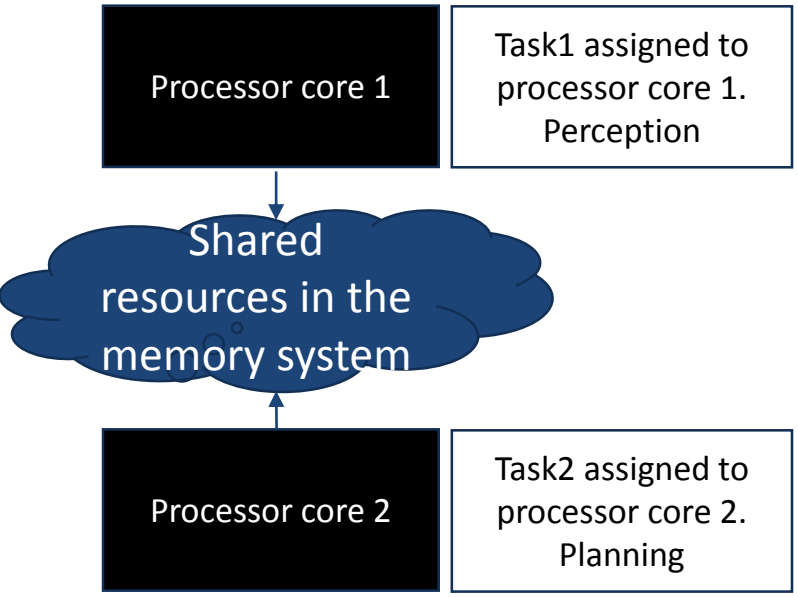
Period=0.2
Deadline=0.2
Execution Requirement=0.1

Period=0.4
Deadline=0.4
Execution Requirement=0.25



Period=0.2
Deadline=0.2
Execution Requirement=0.1

Period=0.4
Deadline=0.4
Execution Requirement=0.25

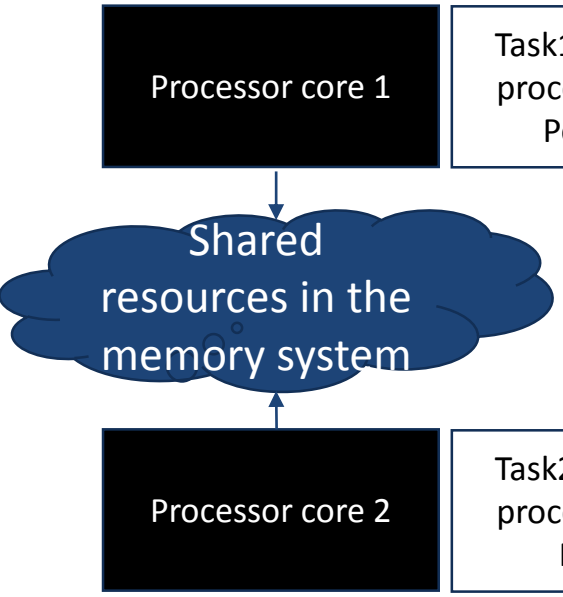


Period=0.2
Deadline=0.2
Execution Requirement=0.1

Execution speed=1 when no co-runner
Execution speed=0.8 when Task2 is co-runner.

Period=0.4
Deadline=0.4
Execution Requirement=0.25

Execution speed=1 when no co-runner
Execution speed=0.5 when Task1 is co-runner.



Task1 assigned to processor core 1. Perception

Period=0.2
Deadline=0.2
Execution Requirement=0.1

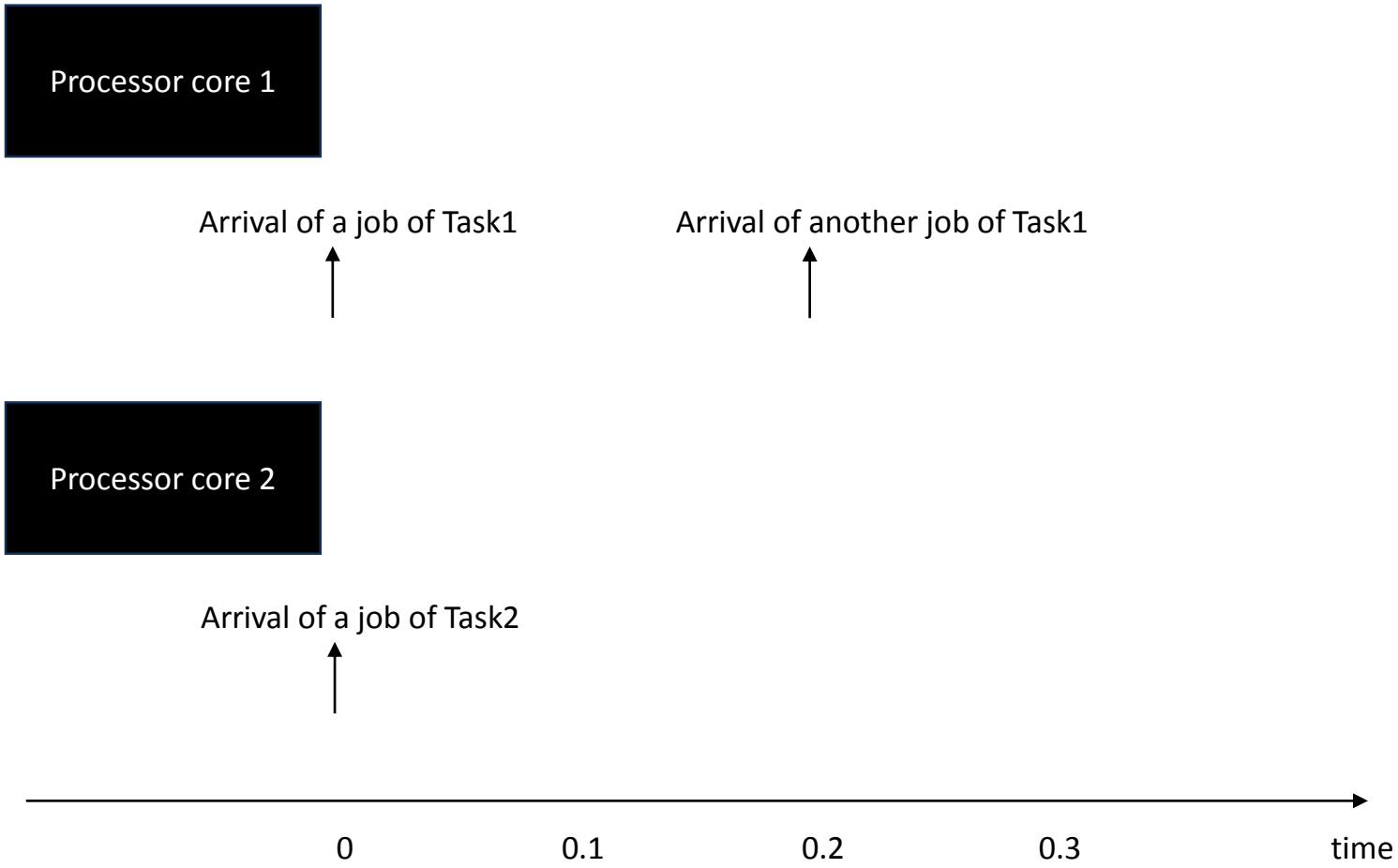
Execution speed=1 when no co-runner
Execution speed=0.8 when Task2 is co-runner.

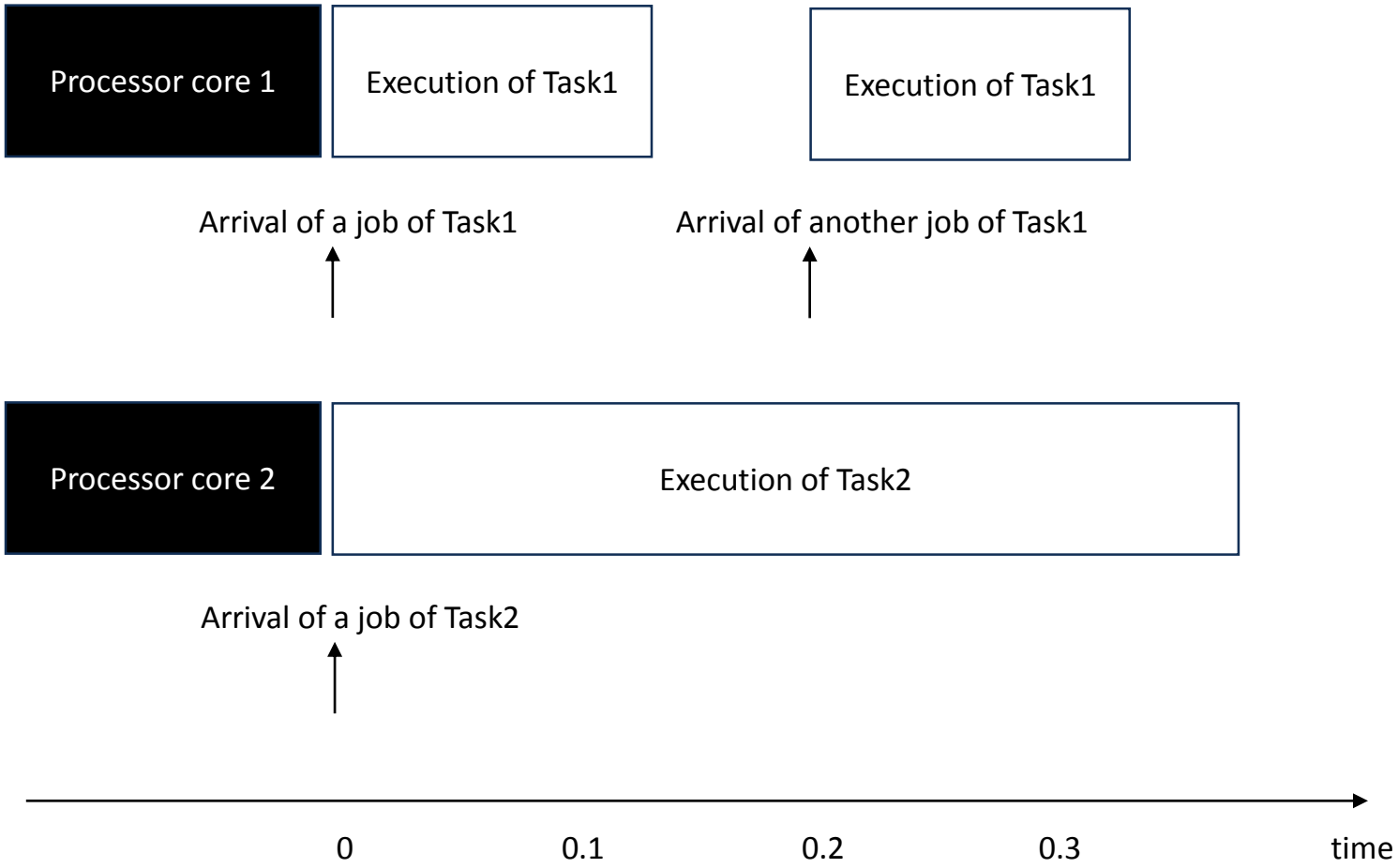
Task2 assigned to processor core 2. Planning

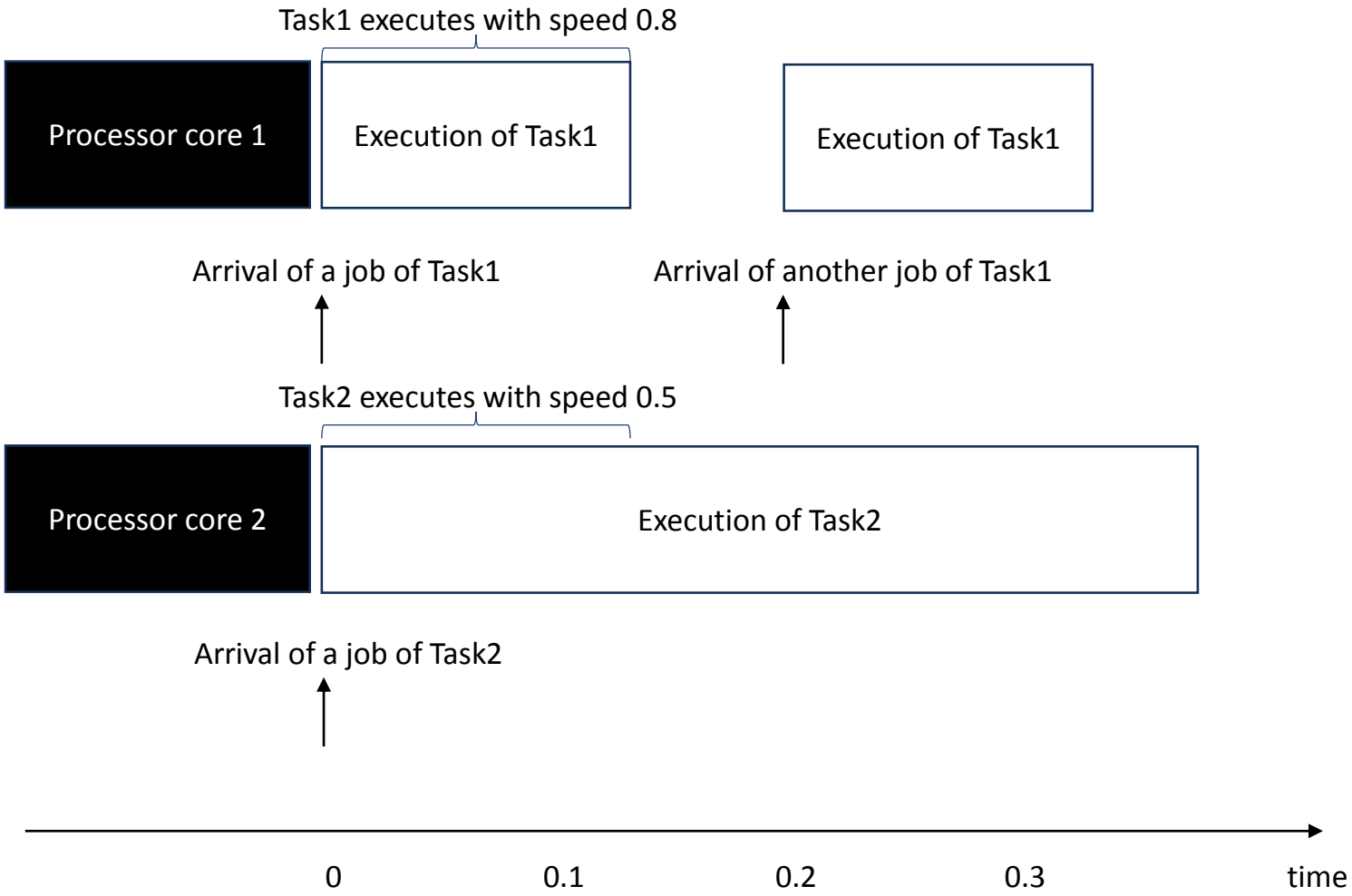
Period=0.4
Deadline=0.4
Execution Requirement=0.25

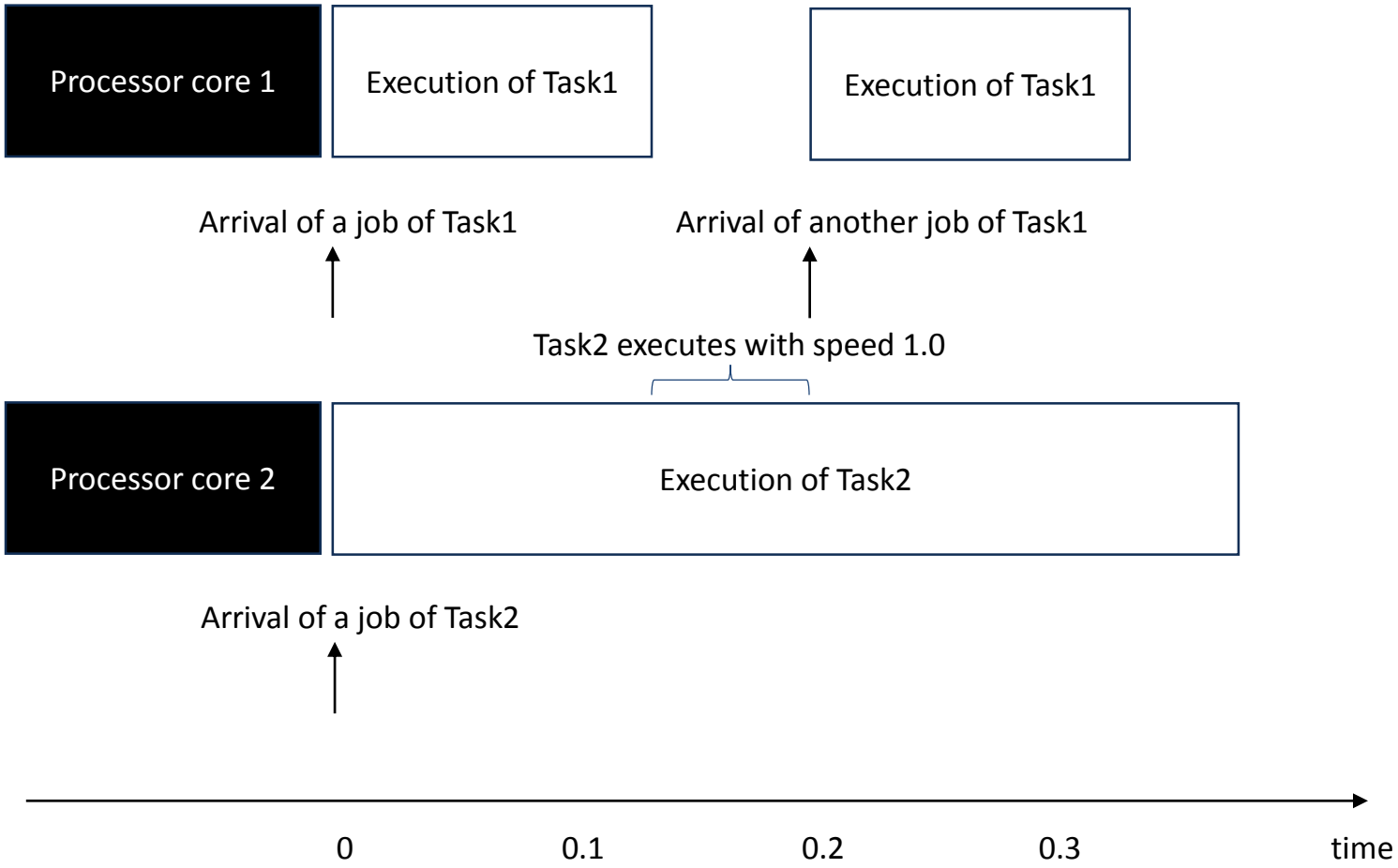
Execution speed=1 when no co-runner
Execution speed=0.5 when Task1 is co-runner.

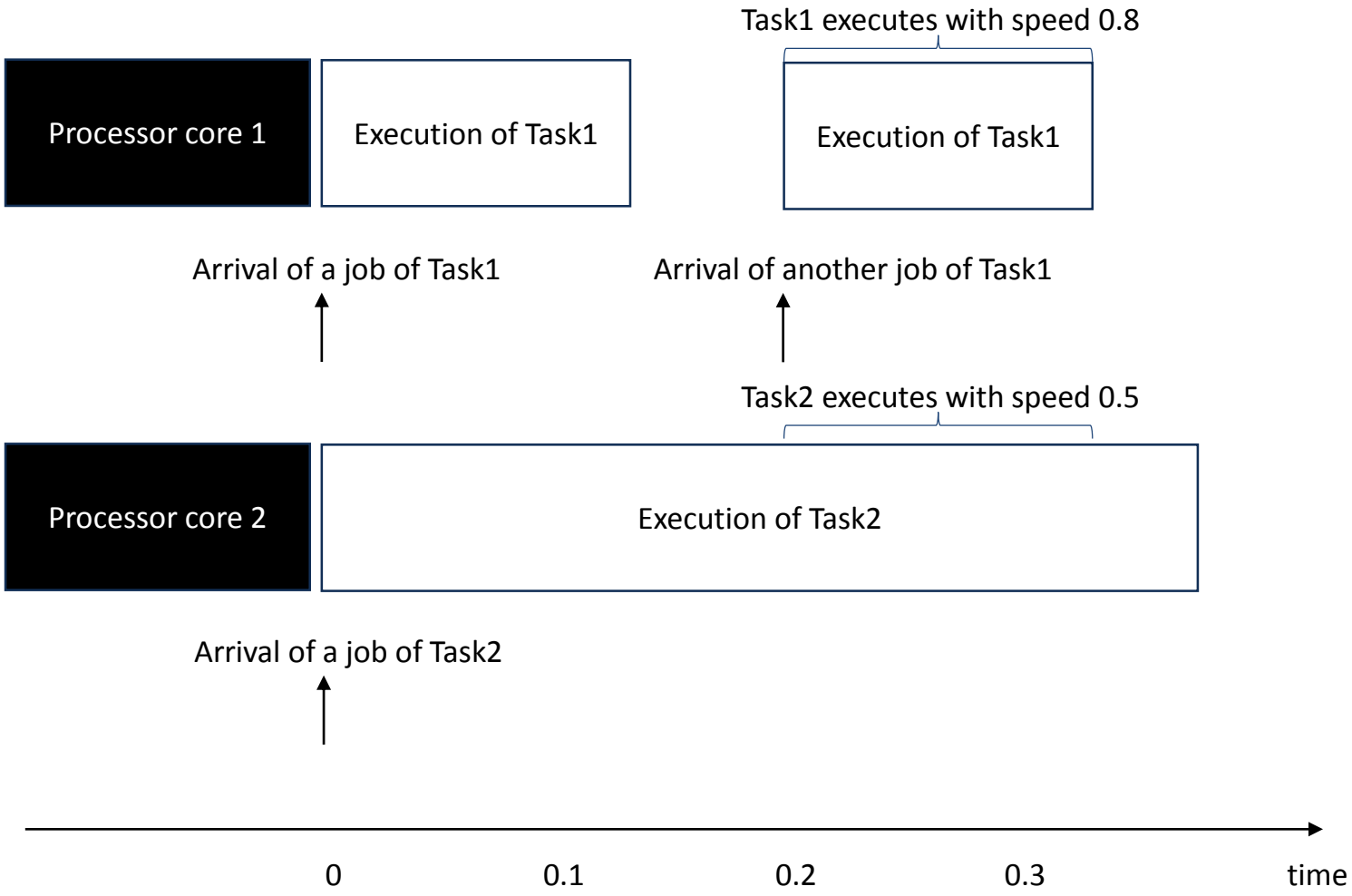
Task	Function performed	Assigned to processor core	Period (T)	Deadline (D)	Execution Requirement (C)	Lower-bound of speed as function of co-runner
Task1	Perception	1	0.2	0.2	0.1	$pw_{1,\varnothing}$ =1
						$pw_{1,\{2\}}$ =0.8
Task2	Planning	2	0.4	0.4	0.25	$pw_{2,\varnothing}$ =1
						$pw_{2,\{1\}}$ =0.5

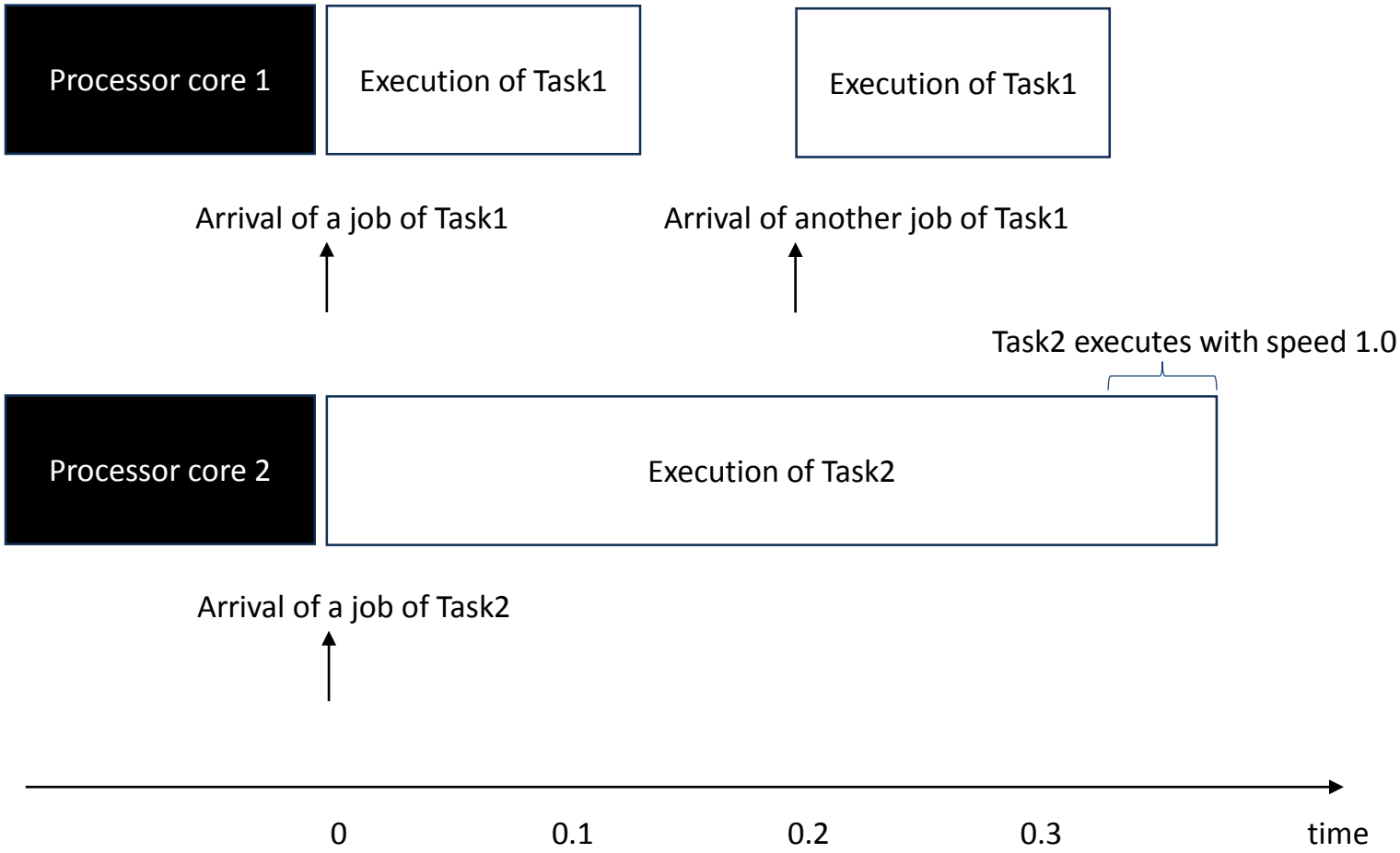


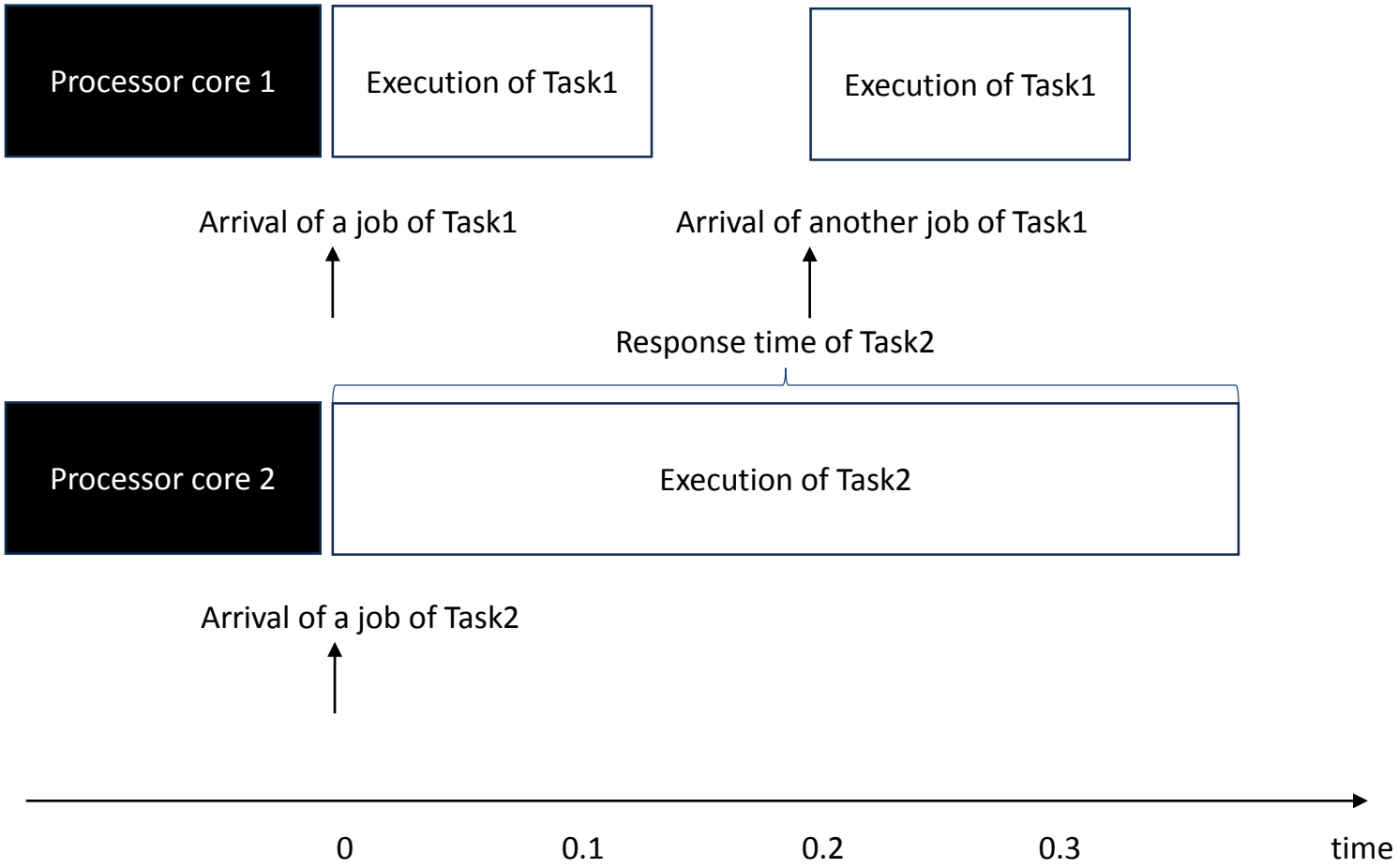


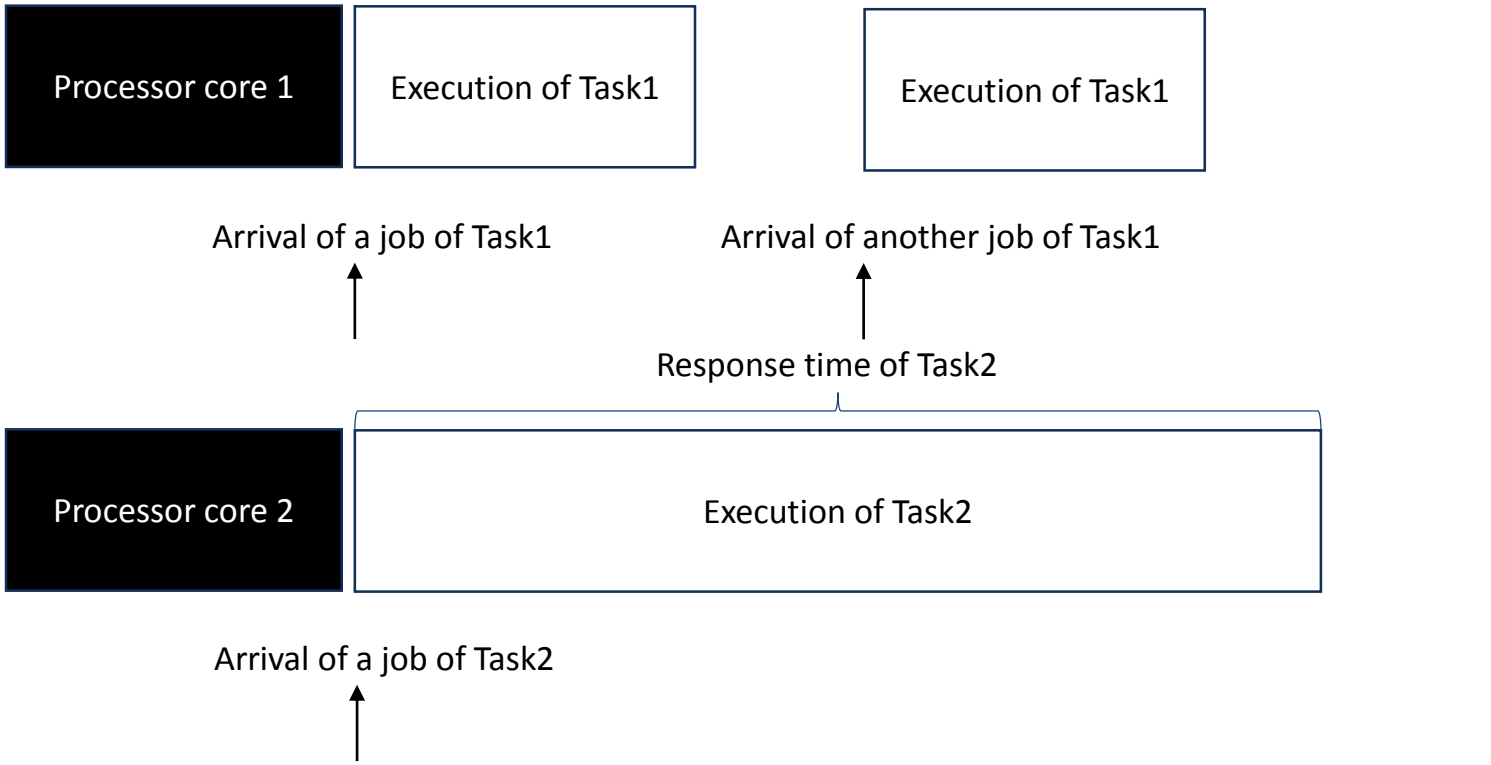




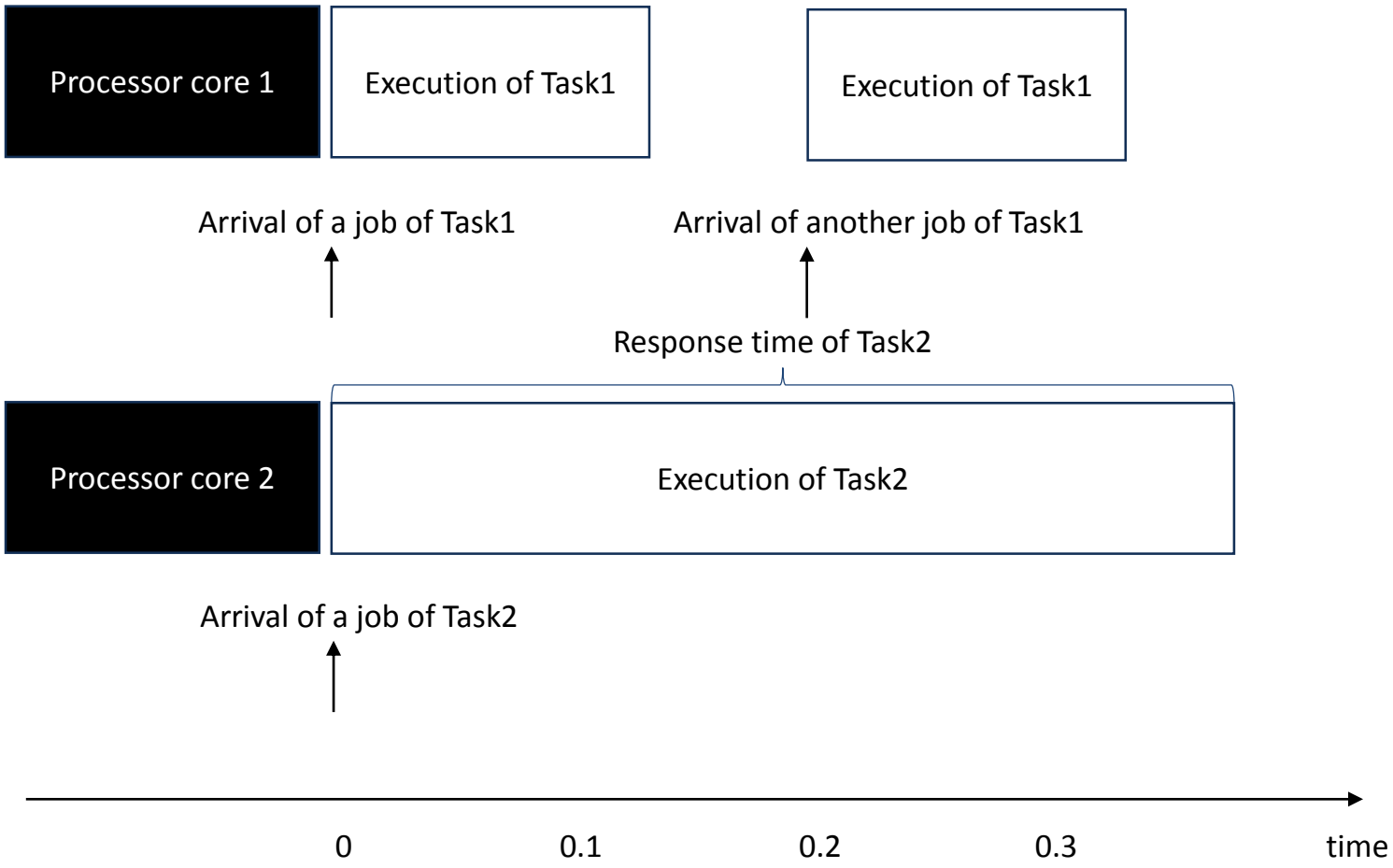






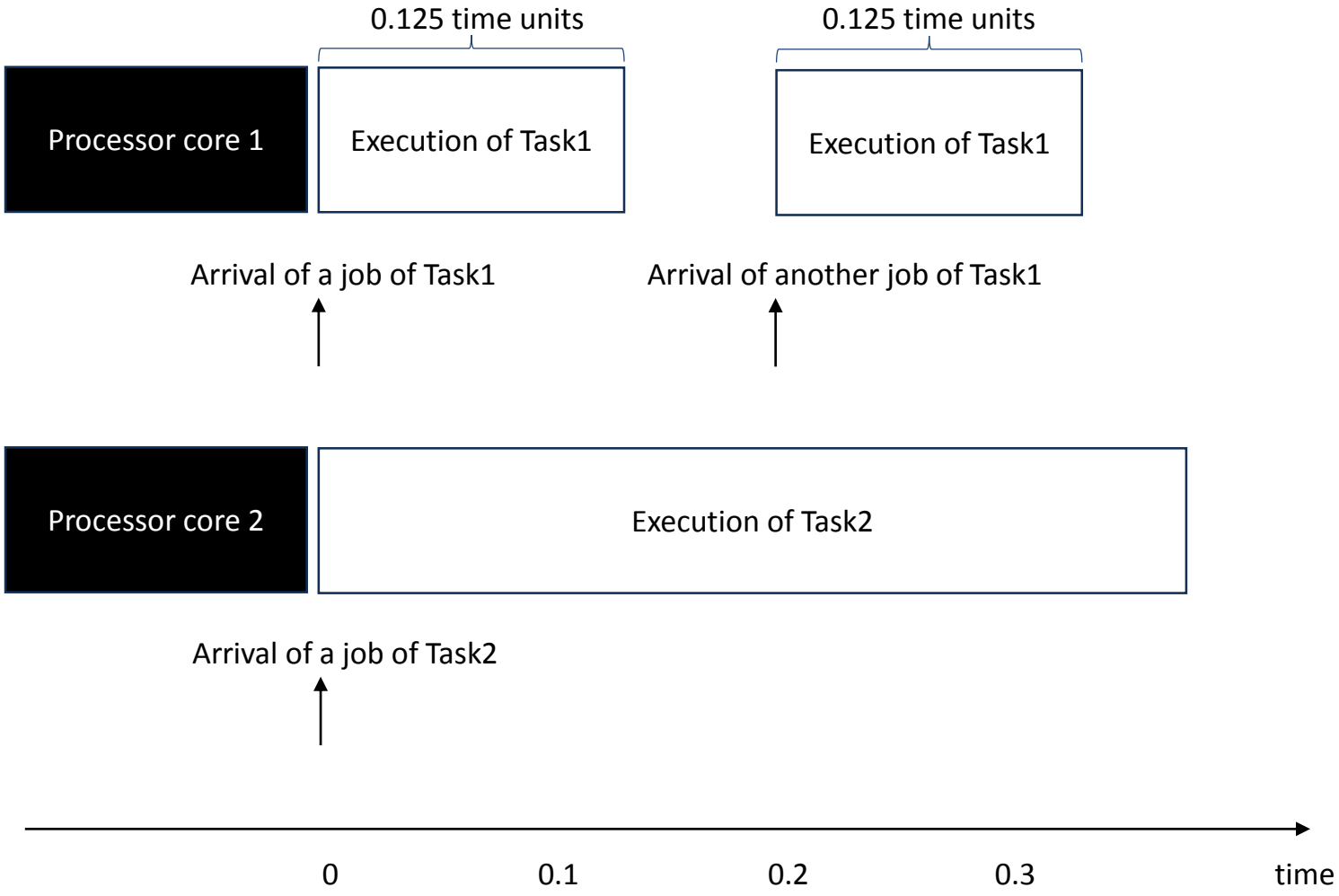


How can we compute the response time of Task2?



Let dur_s denote the cumulative time that tasks in set *s* executes and no other tasks execute.

How can we compute the response time of Task2?

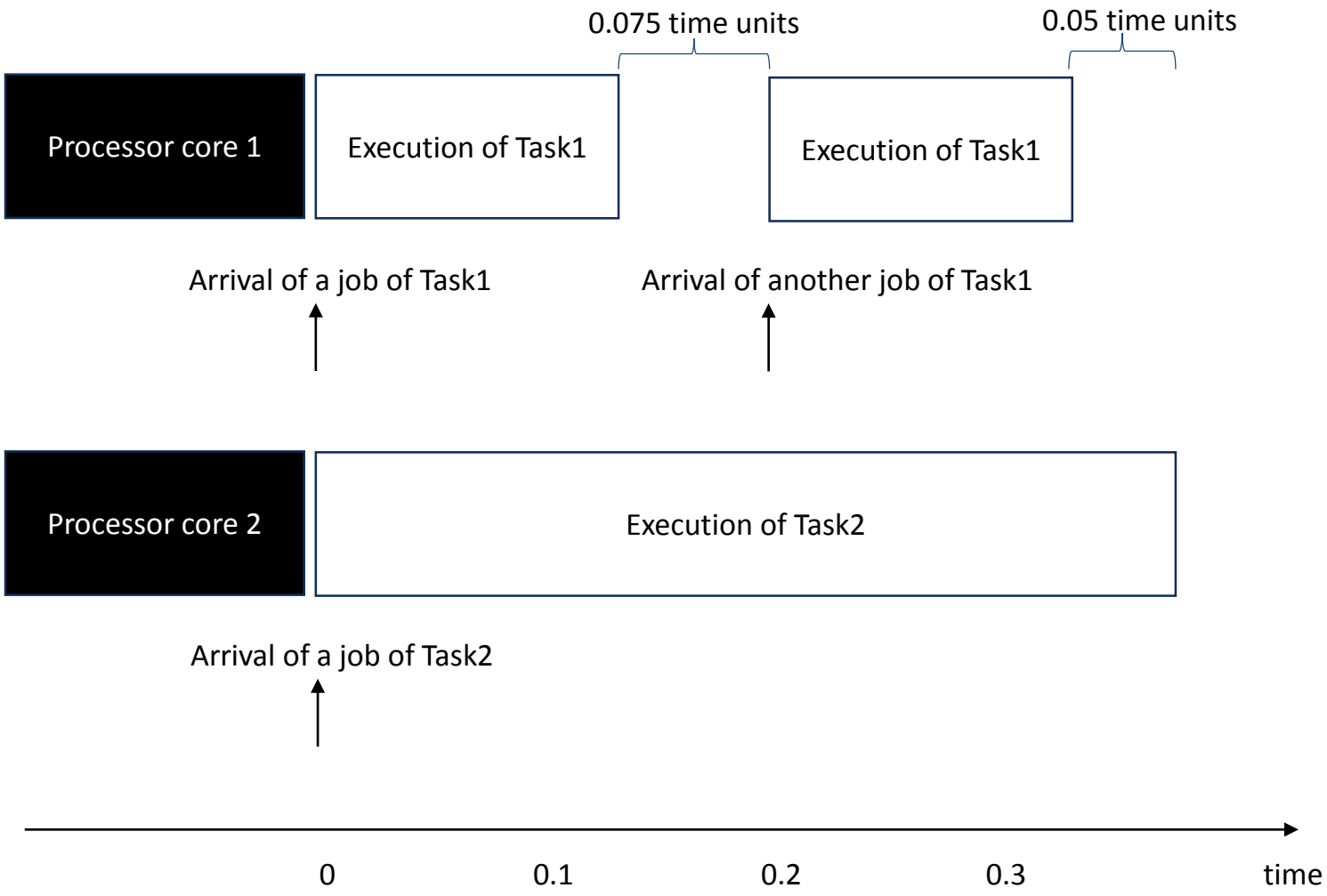


Let dur_s denote the cumulative time that tasks in set *s* executes and no other tasks execute.

0.125+0.125=0.25

$dur_{\{Task1,Task2\}}$ =0.25

How can we compute the response time of Task2?



Let dur_s denote the cumulative time that tasks in set *s* executes and no other tasks execute.

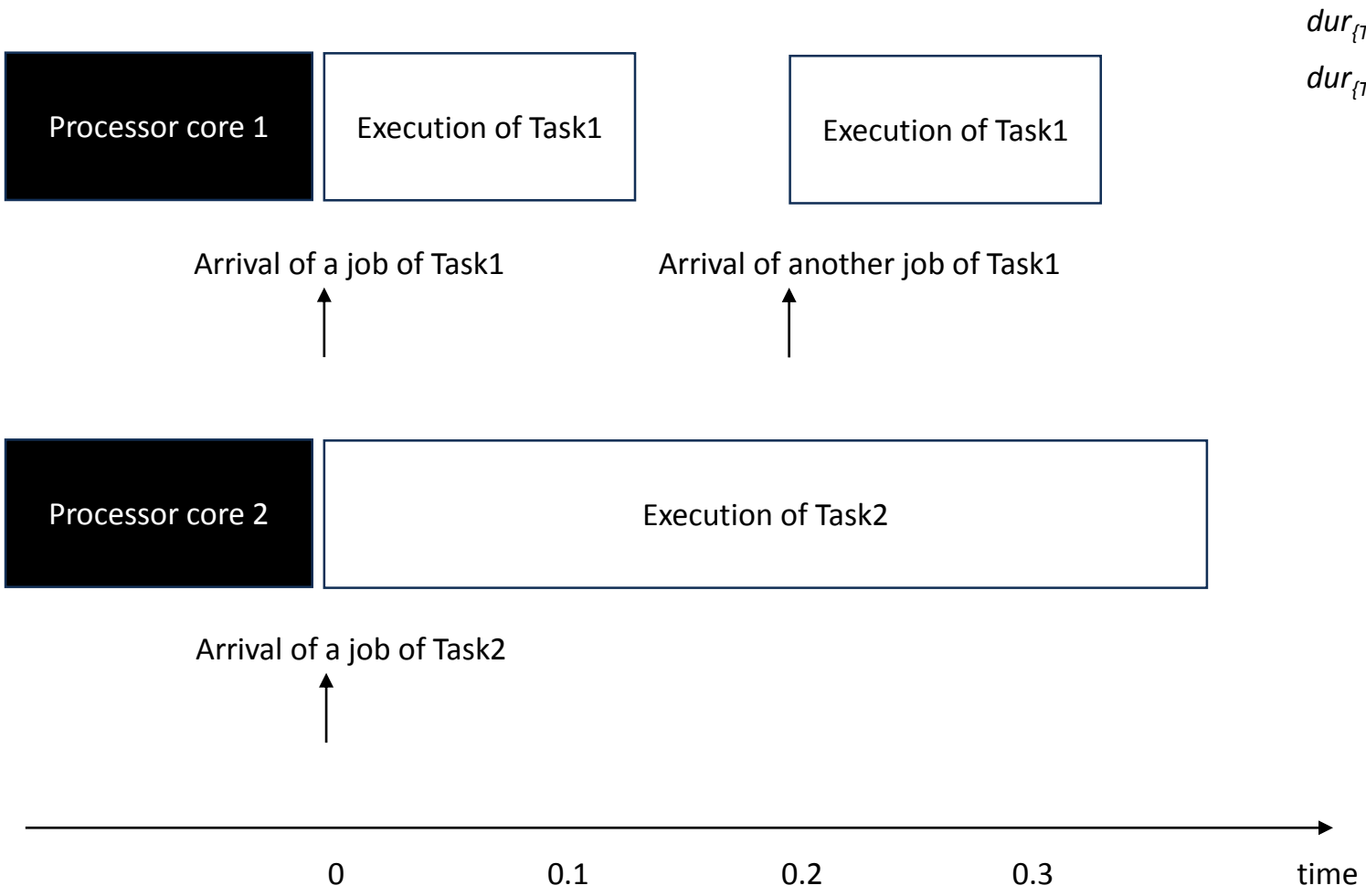
0.125+0.125=0.25

$dur_{\{Task1,Task2\}}$ =0.25

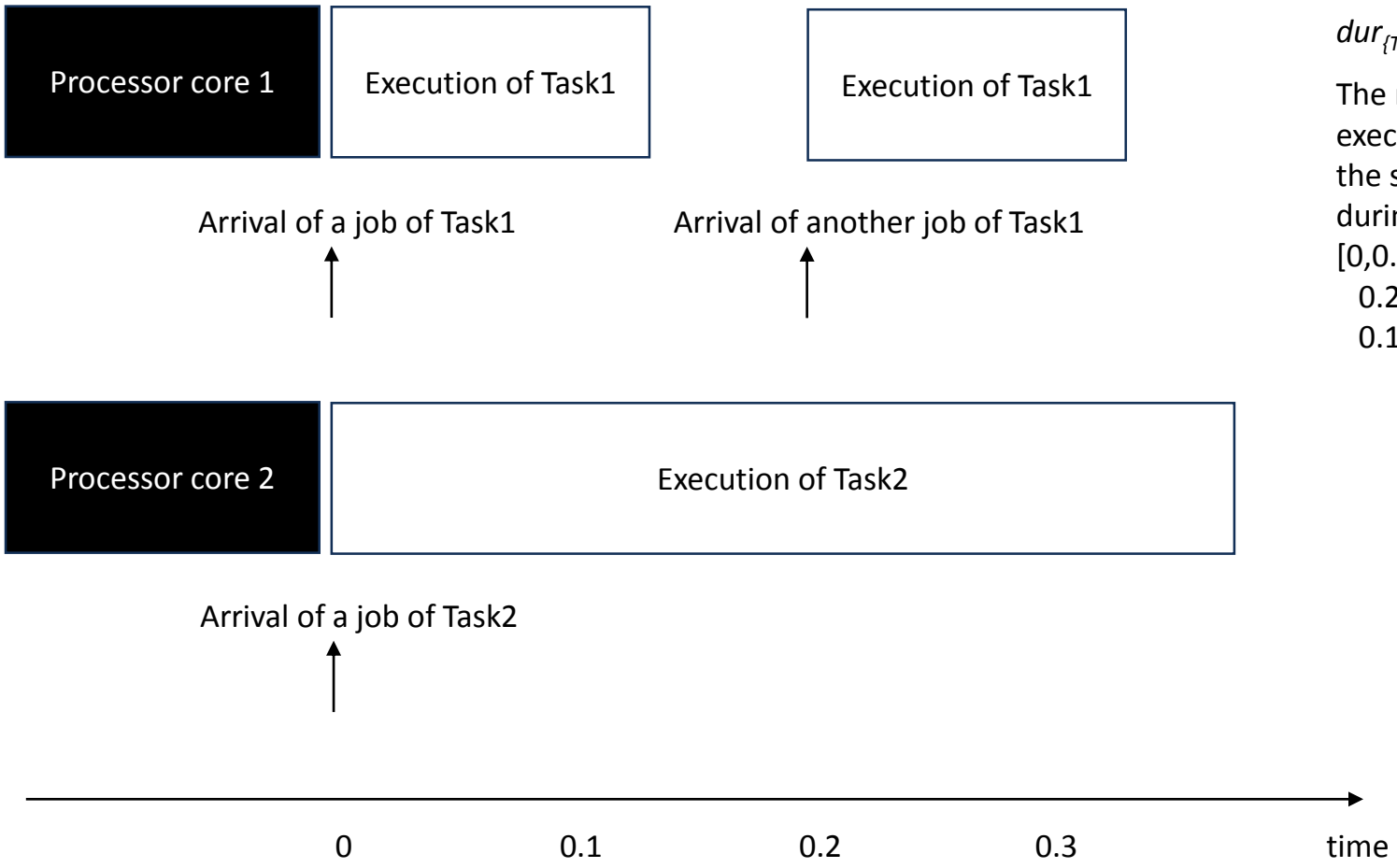
0.075+0.05=0.125

$dur_{\{Task2\}}$ =0.125

How can we compute the response time of Task2?



How can we compute the response time of Task2?



$dur_{\{Task1,Task2\}}$ =0.25

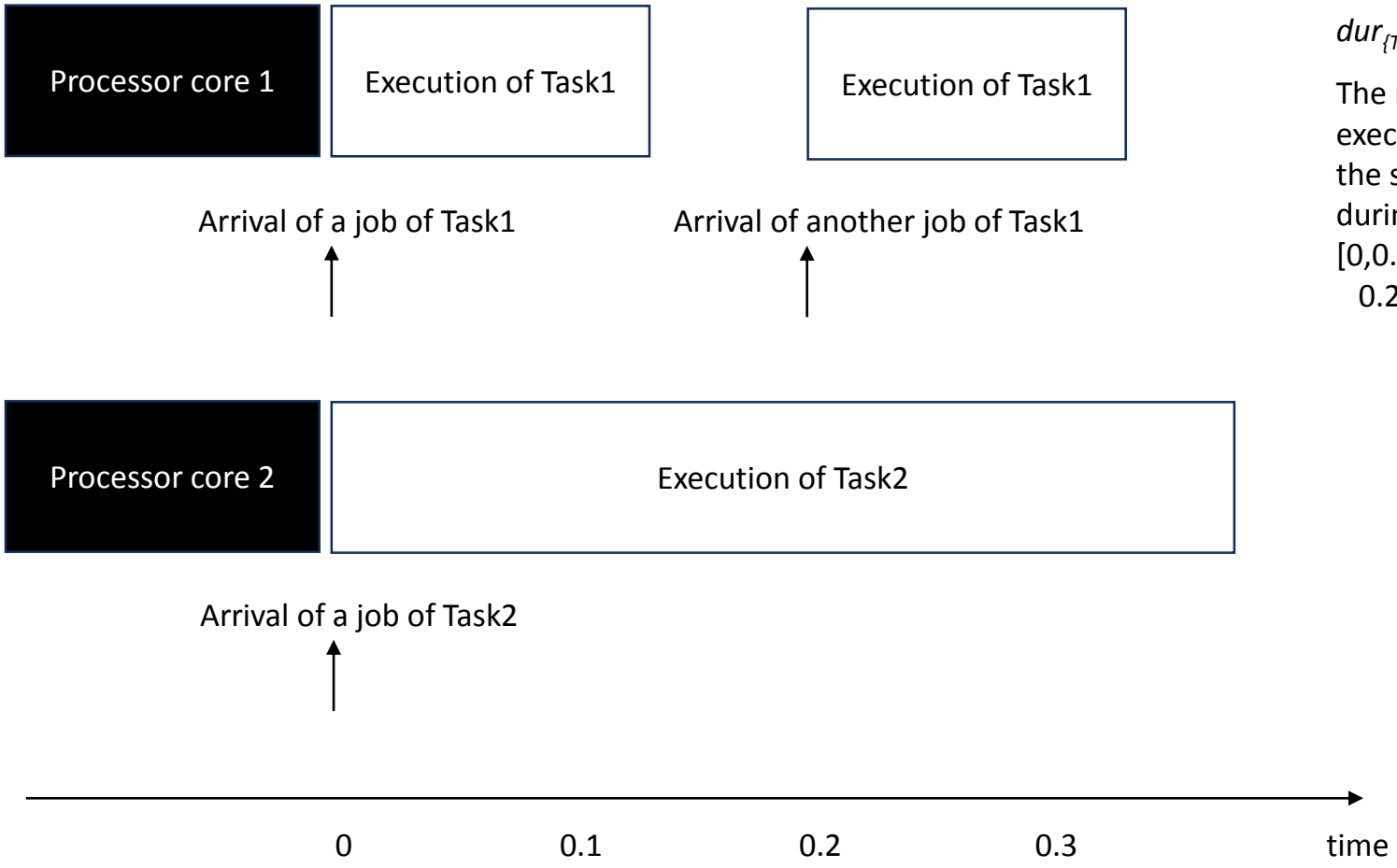
$dur_{\{Task2\}}$ =0.125

The number of units of execution performed by the single job of Task2 during the time interval [0,0.375] is

0.25*0.5+

0.125*1

How can we compute the response time of Task2?

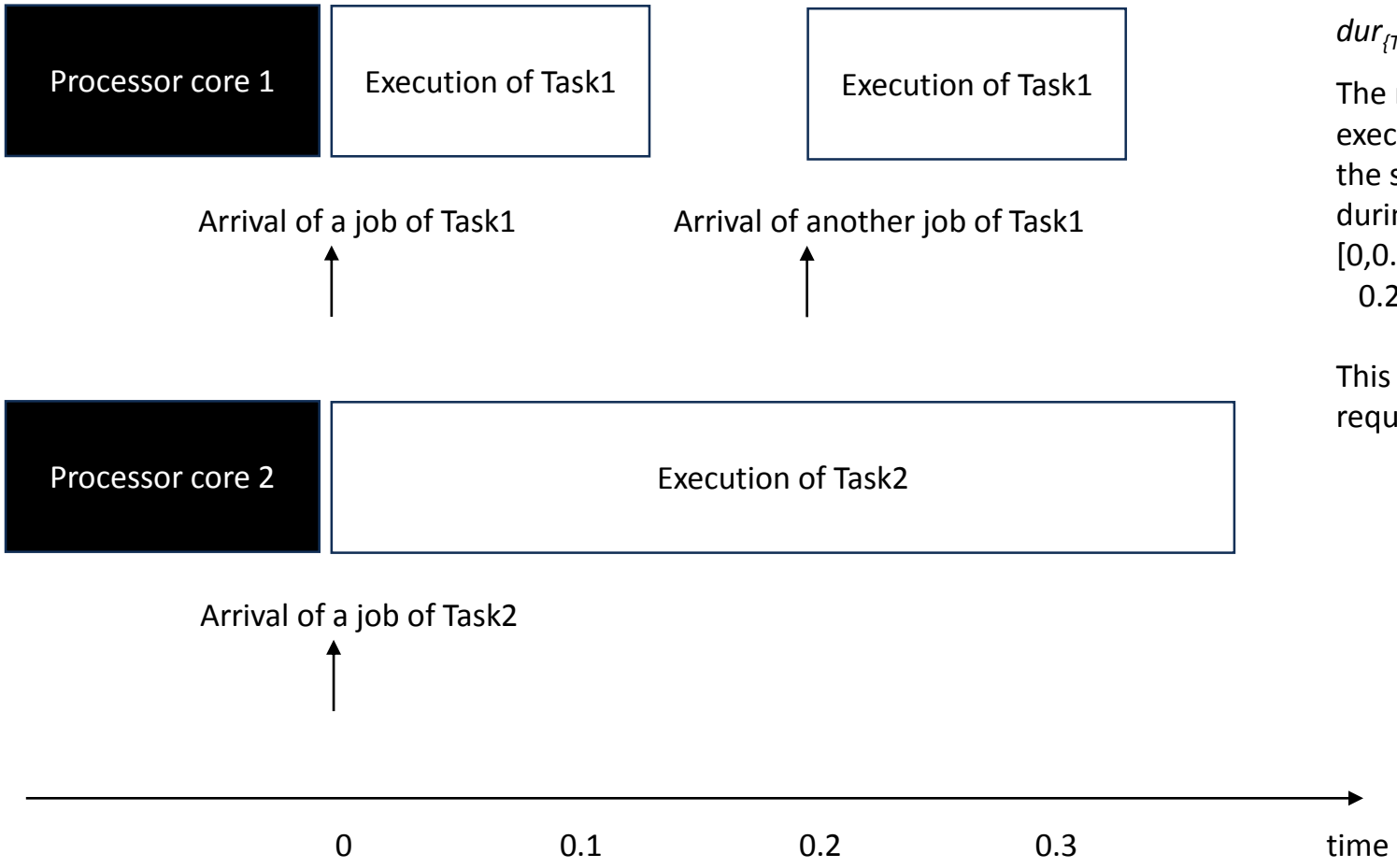


$dur_{\{Task1,Task2\}}$ =0.25

$dur_{\{Task2\}}$ =0.125

The number of units of execution performed by the single job of Task2 during the time interval [0,0.375] is
0.25

How can we compute the response time of Task2?



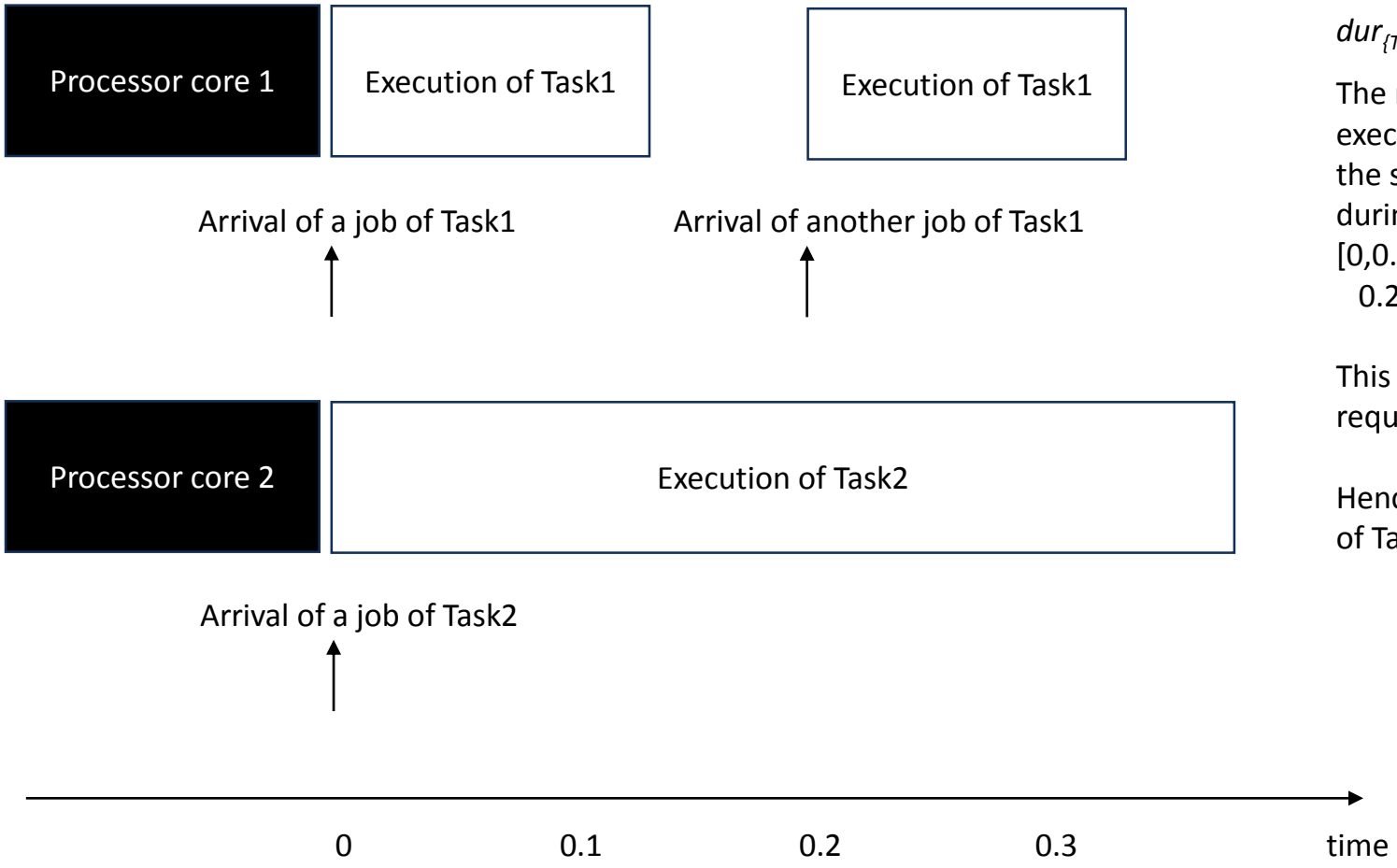
$dur_{\{Task1,Task2\}}$ =0.25

$dur_{\{Task2\}}$ =0.125

The number of units of execution performed by the single job of Task2 during the time interval [0,0.375] is
0.25

This is the execution requirement of Task2.

How can we compute the response time of Task2?



How can we compute the response time of Task2?

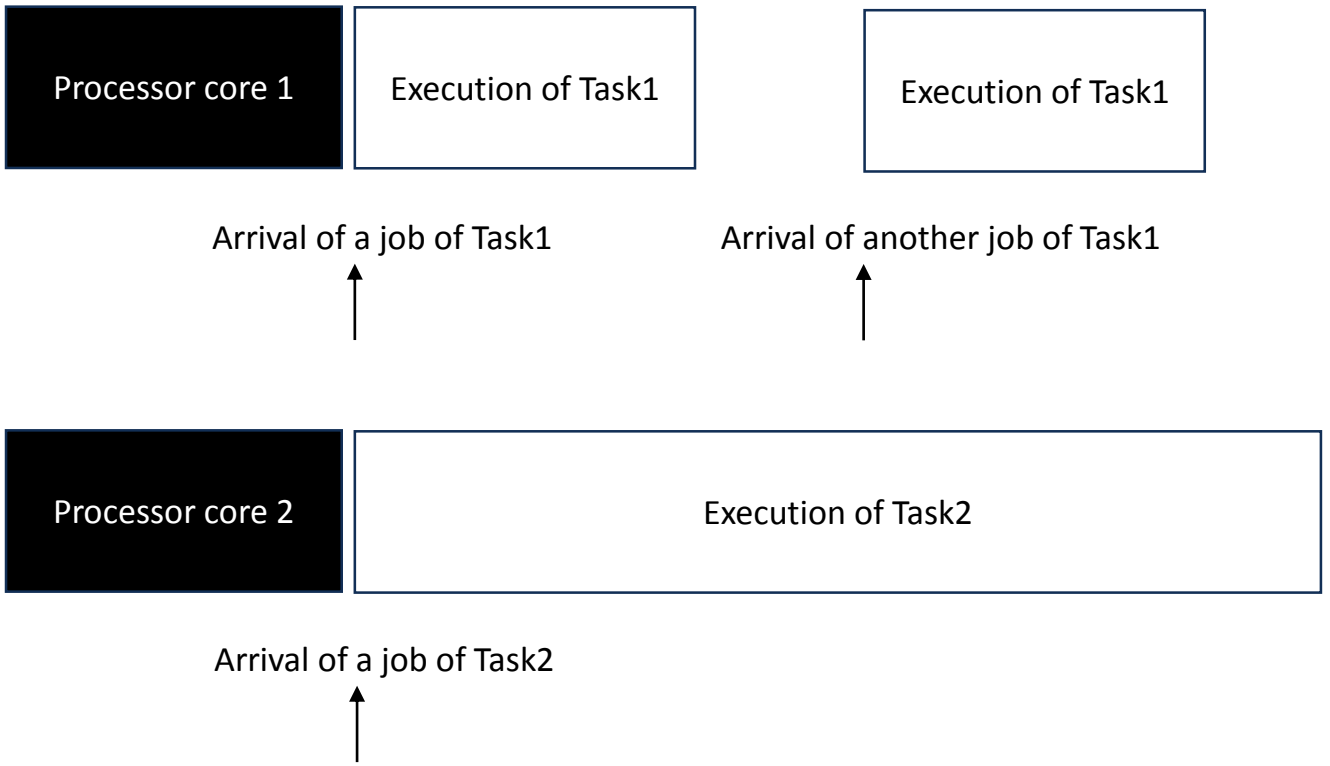
$dur_{\{Task1,Task2\}}$ =0.25

$dur_{\{Task2\}}$ =0.125

The number of units of execution performed by the single job of Task2 during the time interval [0,0.375] is
0.25

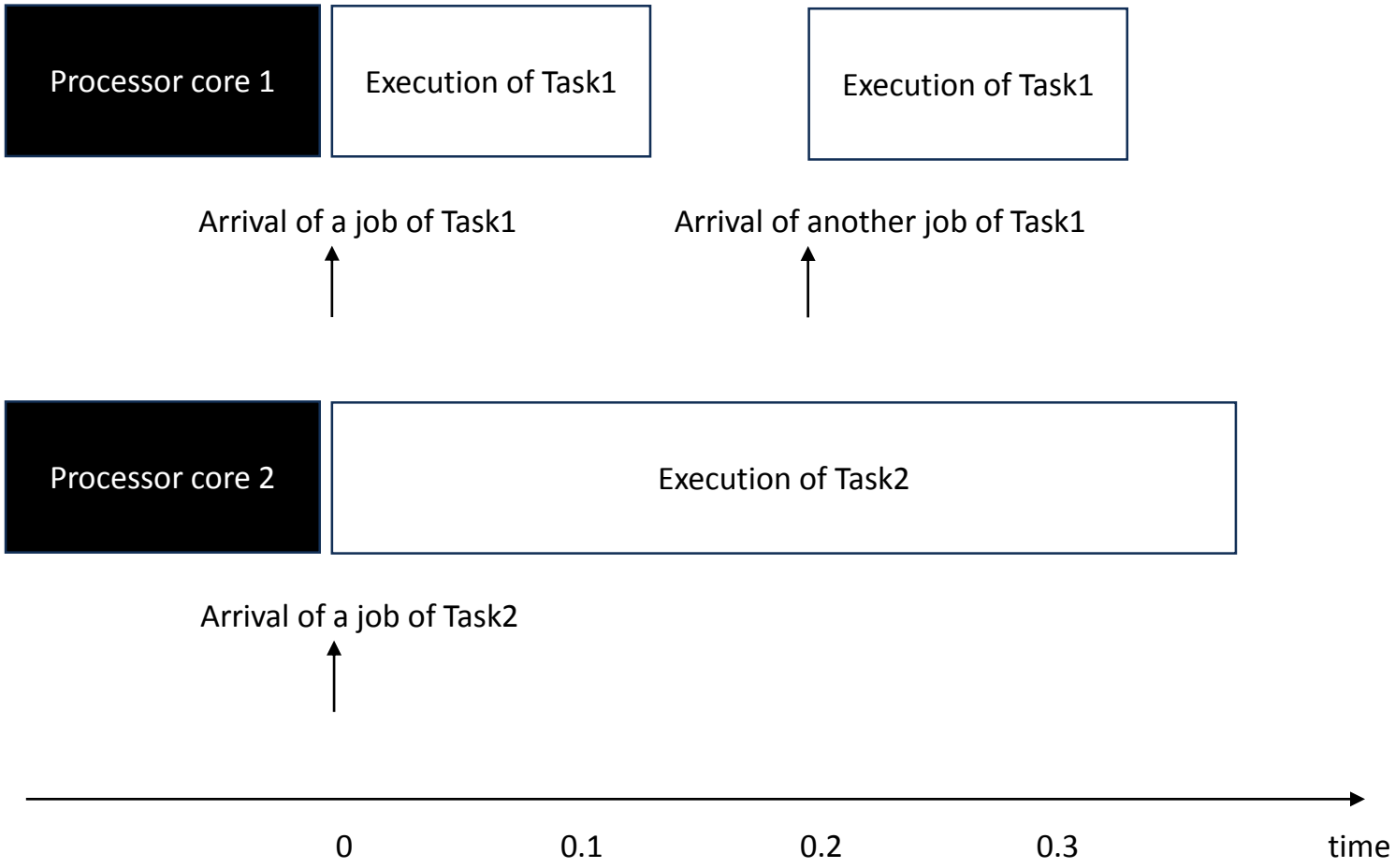
This is the execution requirement of Task2.

Hence, the response time of Task2 is 0.375.

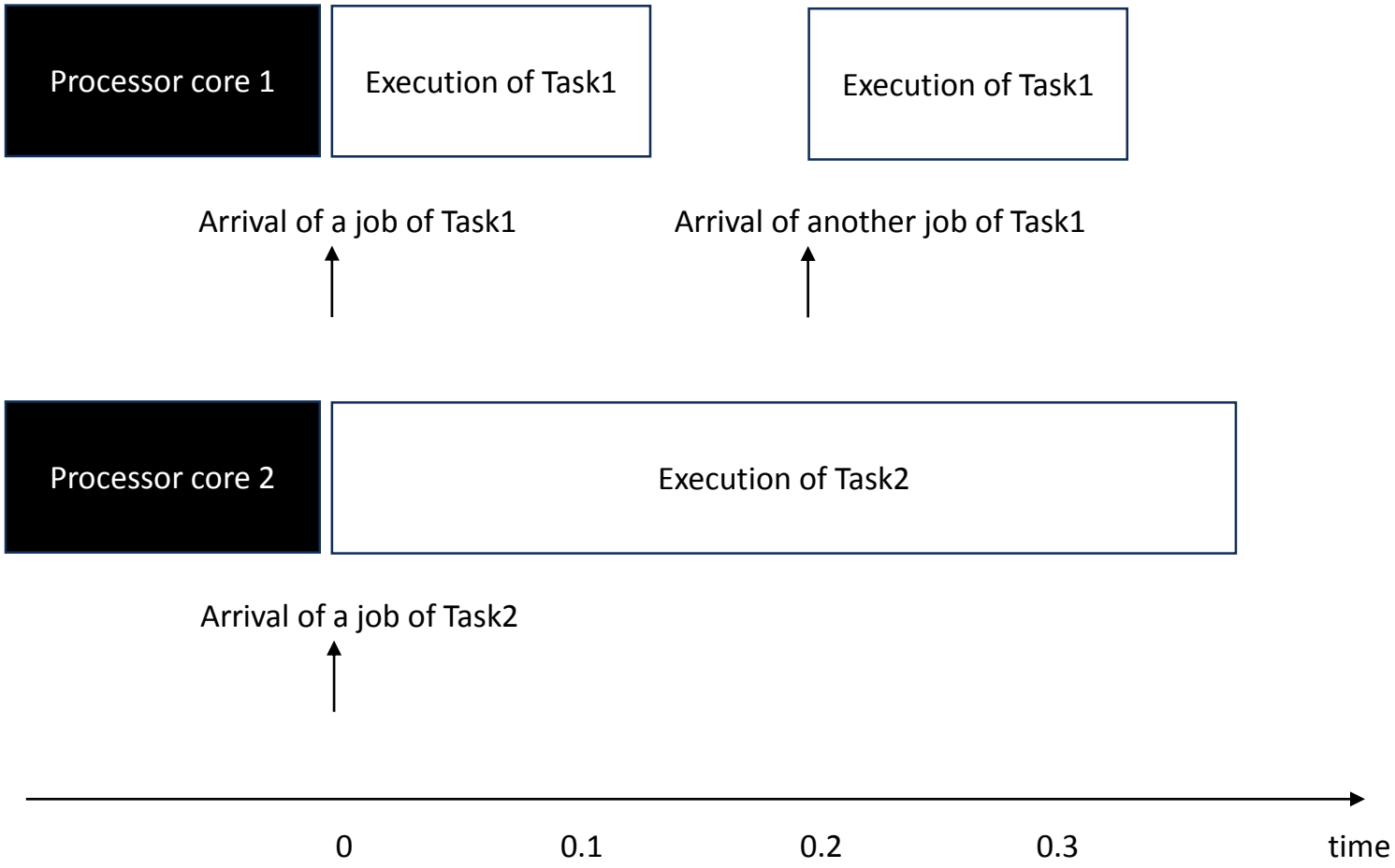


How can we compute the response time of Task2?

In general:
Find R_2 such that:
when solving the following optimization problem:
maximize
$dur_{\{Task1,Task2\}} + dur_{\{Task2\}}$
subject to
$dur_{\{Task1\}} * 1 + dur_{\{Task1,Task2\}} * 0.8 \leq \lceil R_2/0.2 \rceil * 0.1$
$dur_{\{Task2\}} * 1 + dur_{\{Task1,Task2\}} * 0.5 \leq 0.25$
where
$dur_{\{Task1,Task2\}}$, $dur_{\{Task1\}}$, and $dur_{\{Task2\}}$ are non-negative real numbers
the value of the objective function equals R_2.

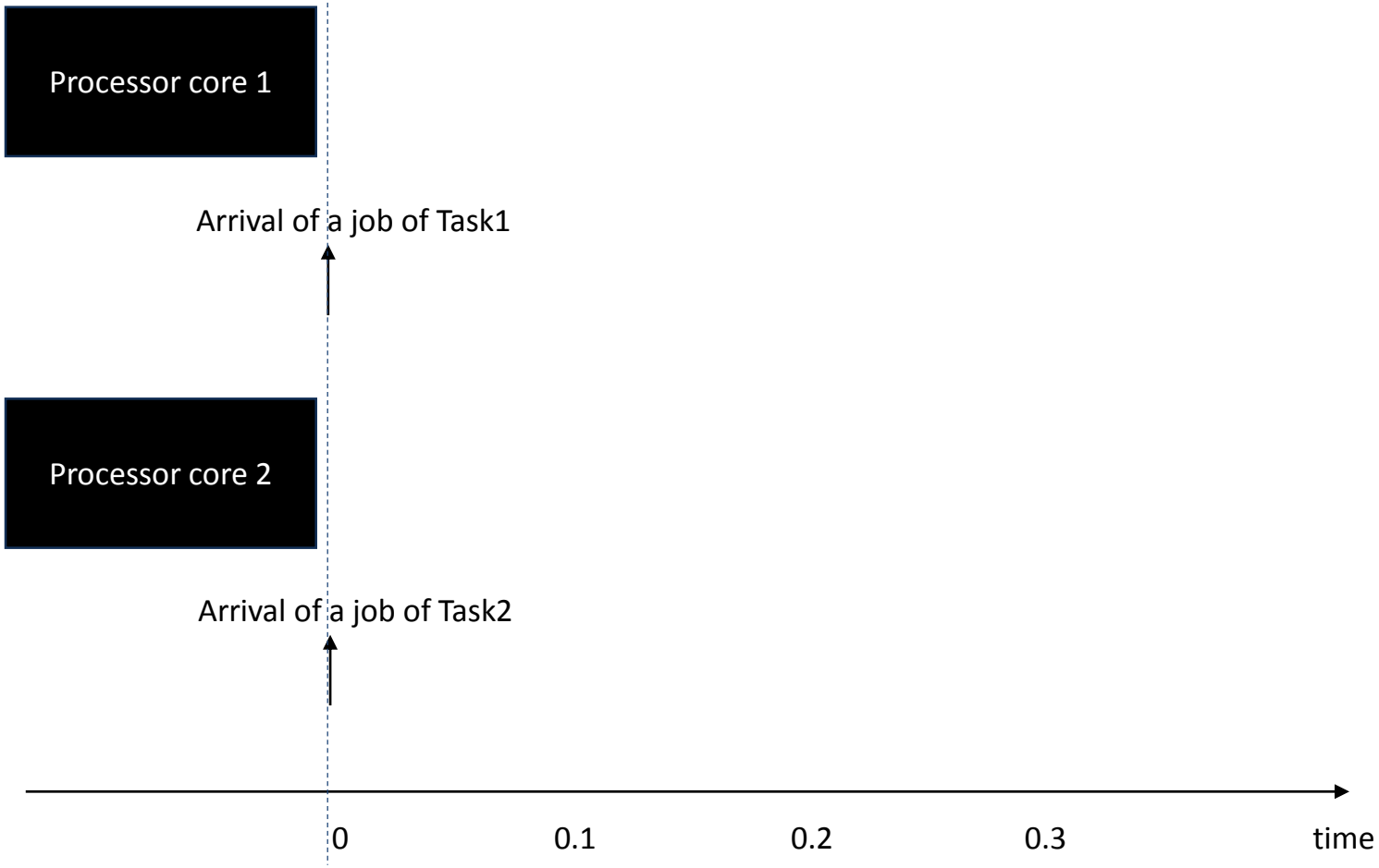


How do we know that the behavior of tasks at run-time respects the model?



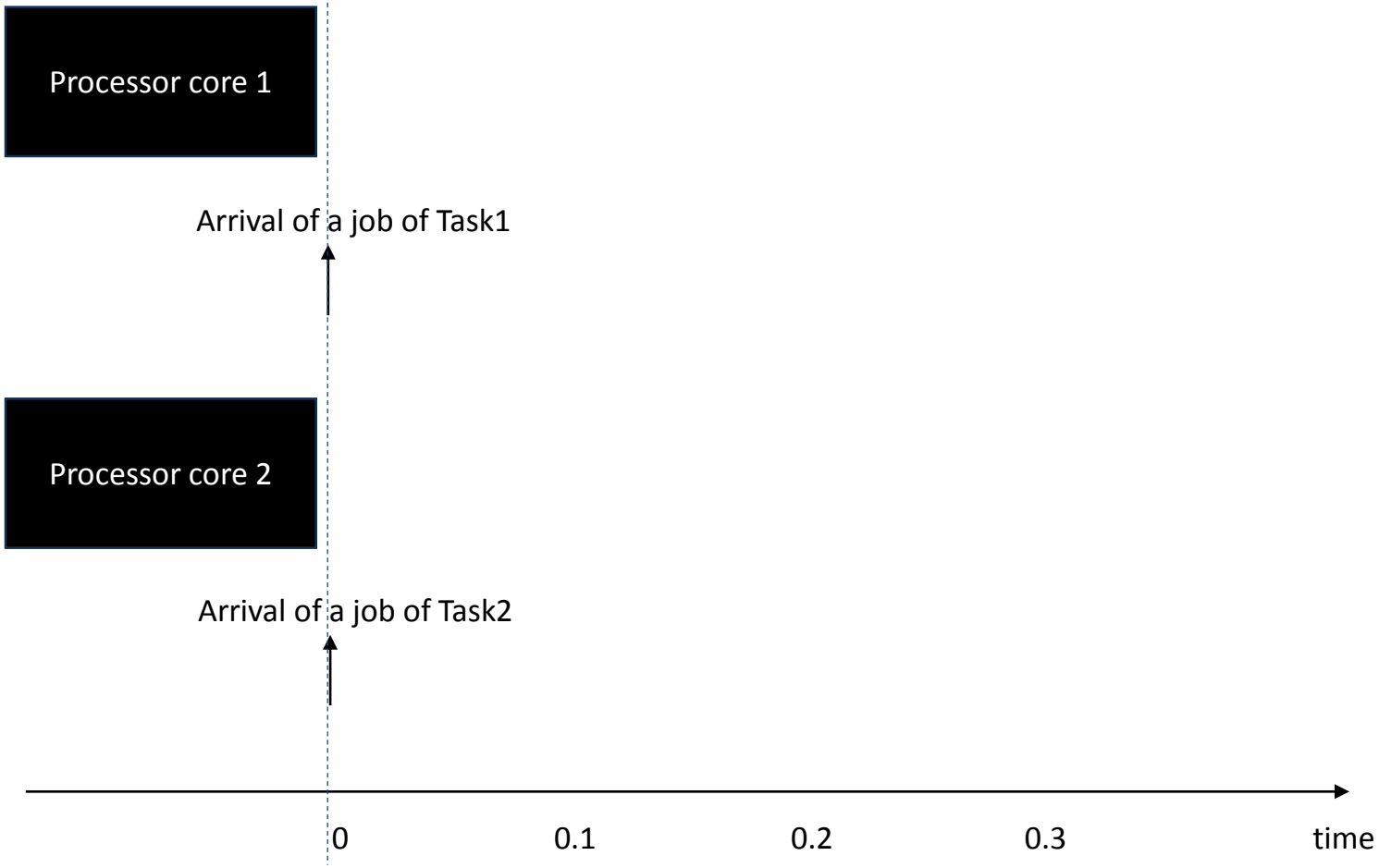
How do we know that the behavior of tasks at run-time respects the model? We need to monitor tasks are run-time.

We need to decide whether Task2 should be stopped?



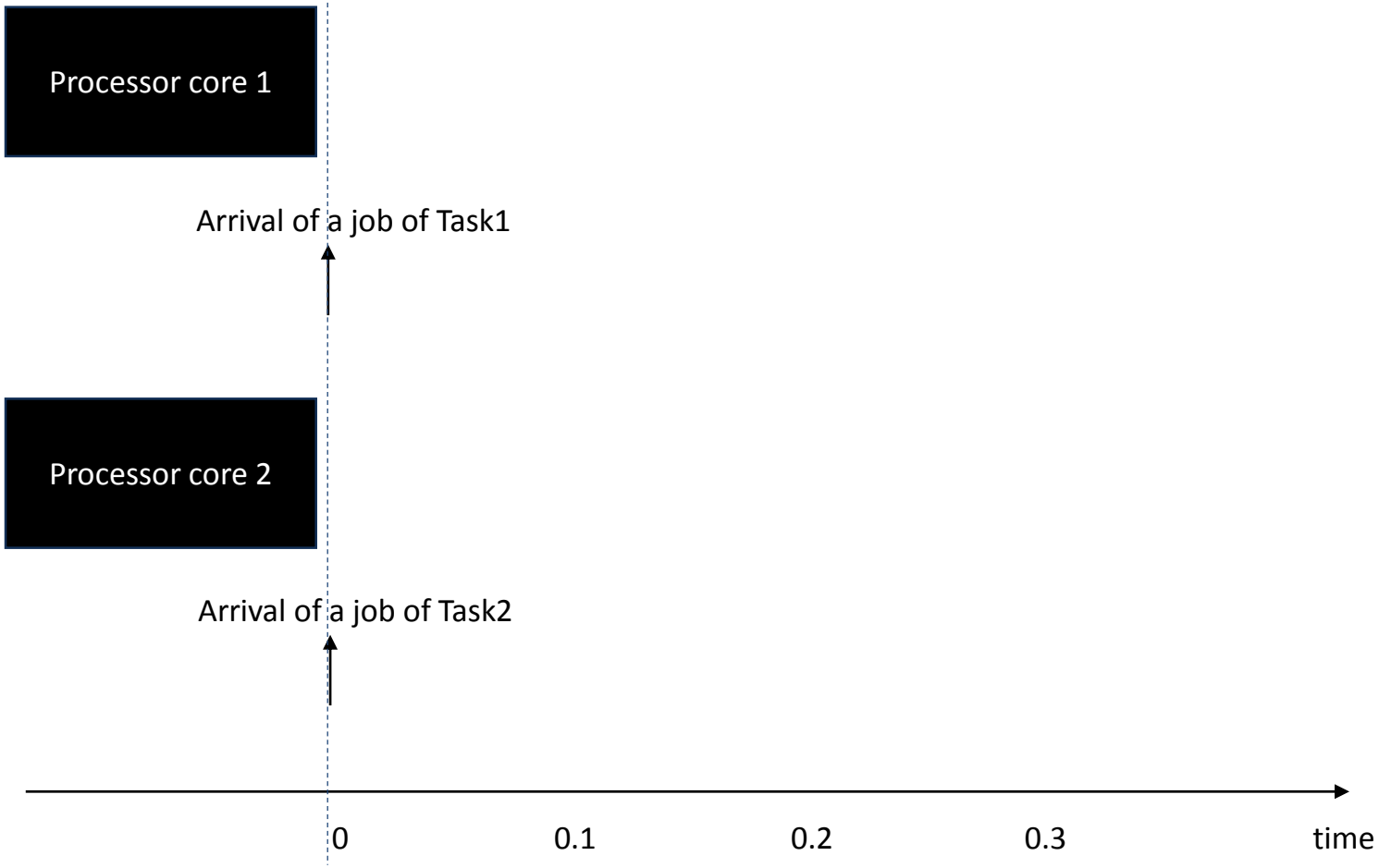
How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Task2 should not be allowed to execute so that it has performed more than C_2 units of execution.



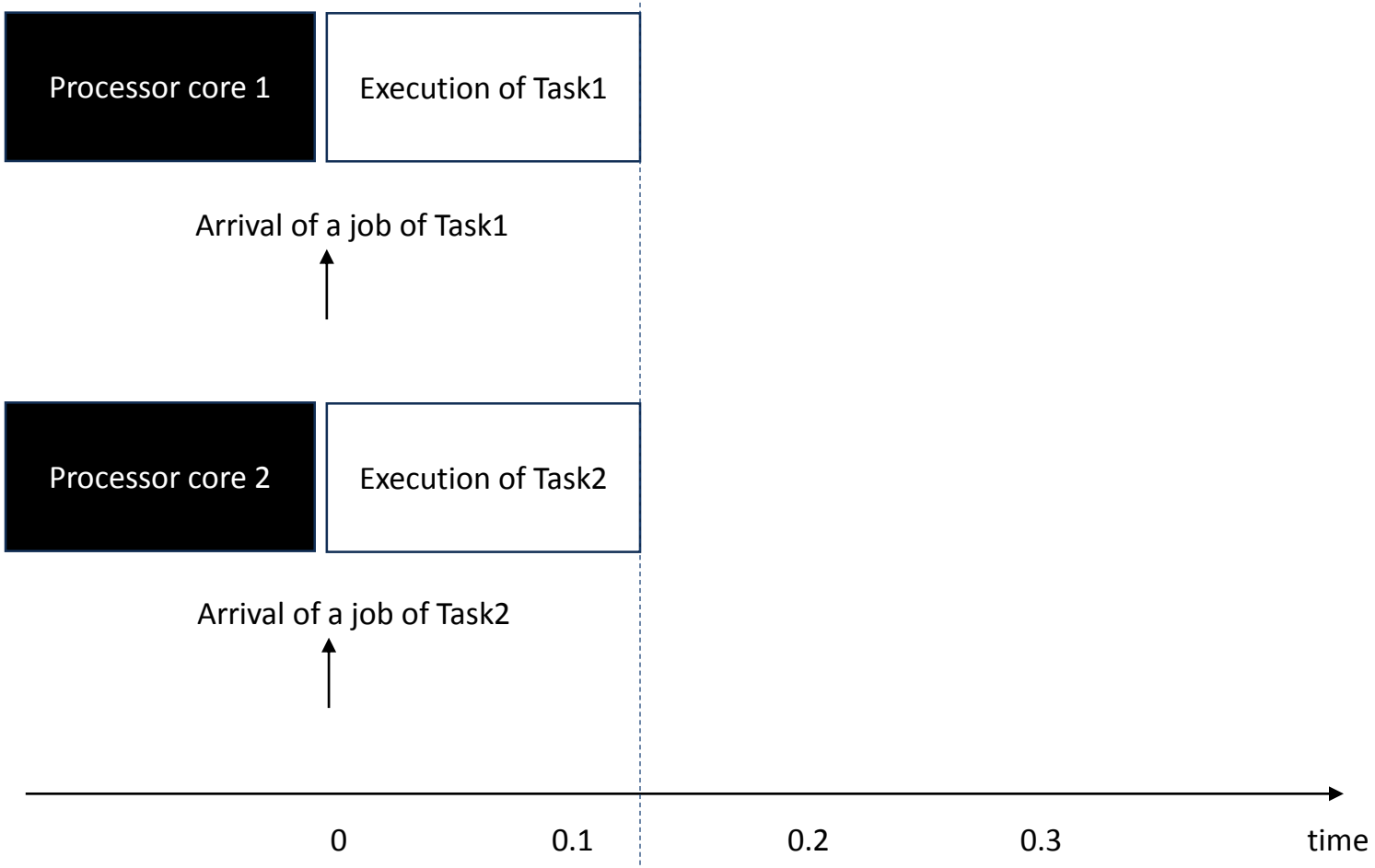
How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Task2 should not be allowed to execute so that it has performed more than 0.25 units of execution.



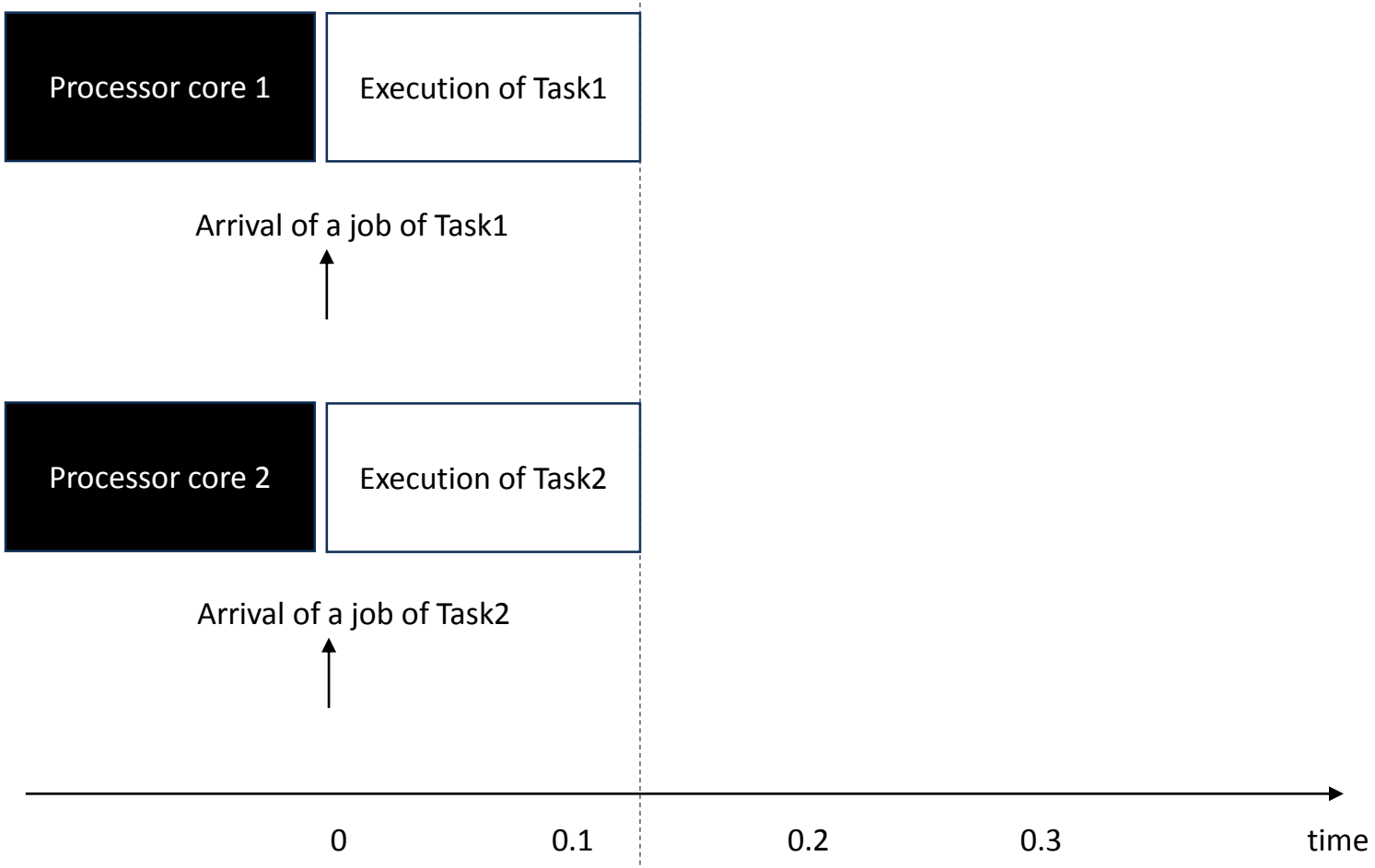
How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Here, Task2 has performed at least 0.125*0.5 units of execution.



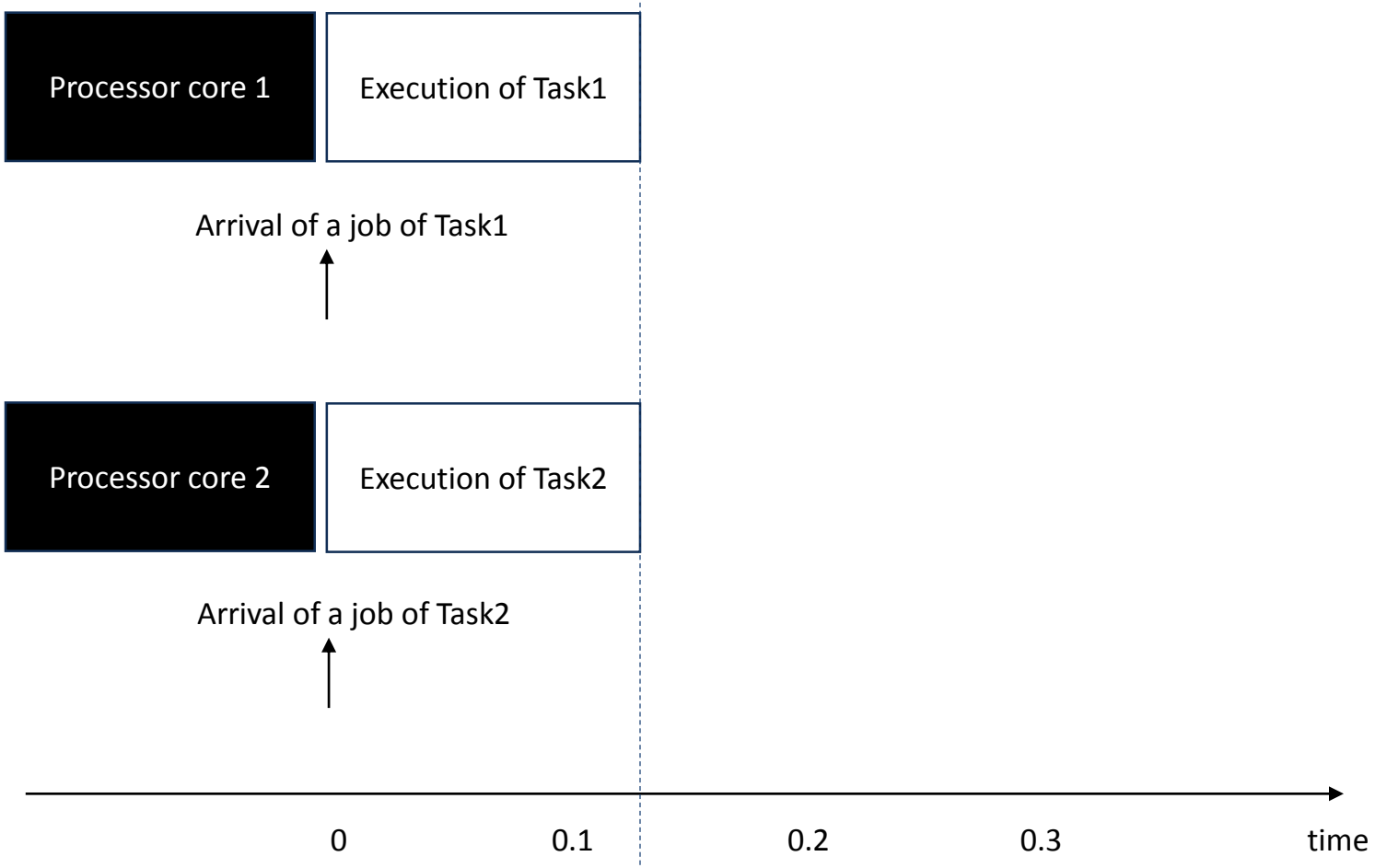
How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Here, Task2 has performed at least 0.125*0.5 units of execution.
Hence, after this time, Task2 should not be allowed to execute so that it has performed more than 0.25-0.125*0.5 units of execution.



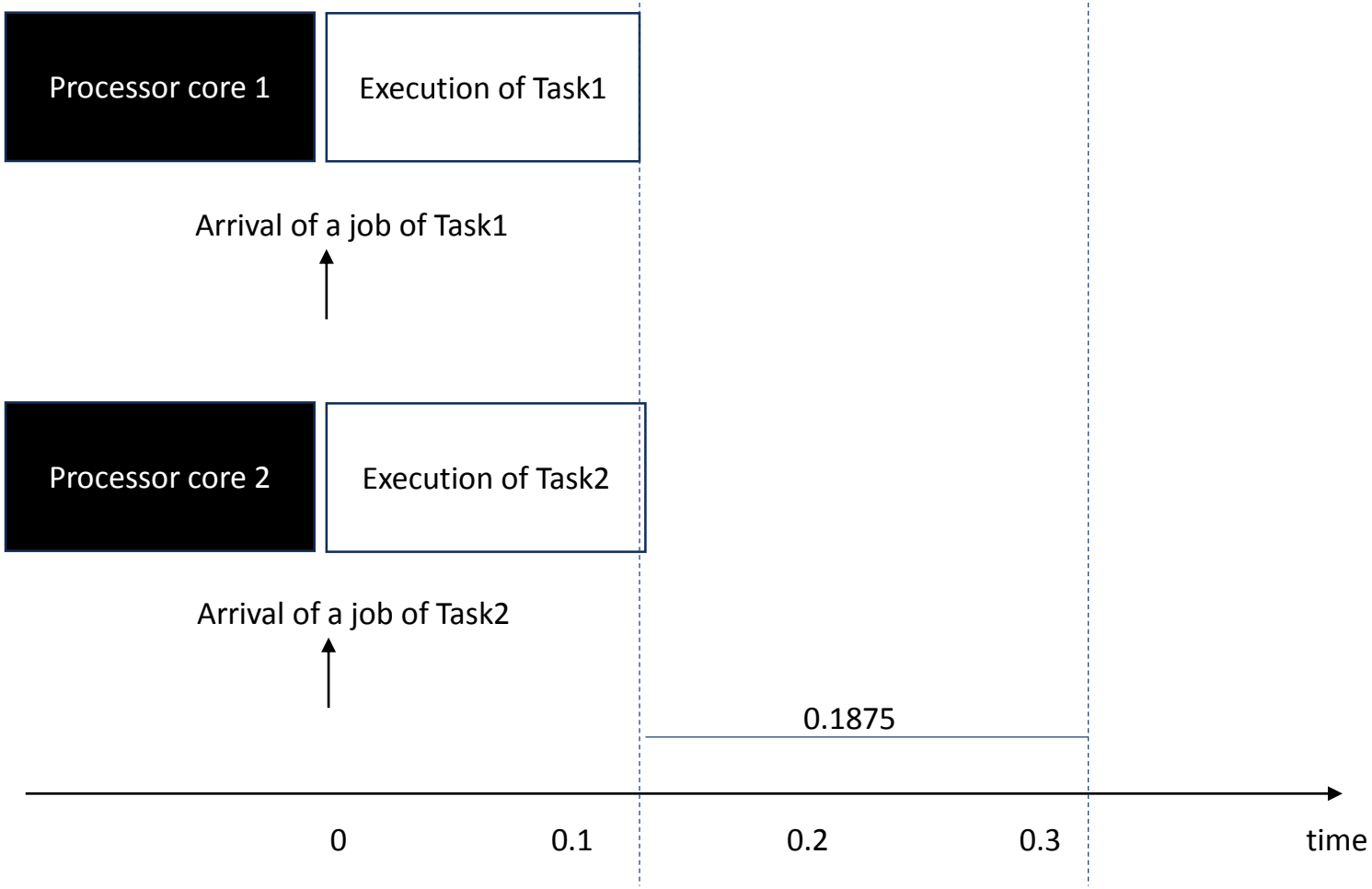
How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Here, Task2 has performed at least 0.125*0.5 units of execution.
Hence, after this time, Task2 should not be allowed to execute so that it has performed more than 0.1875 units of execution.



How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

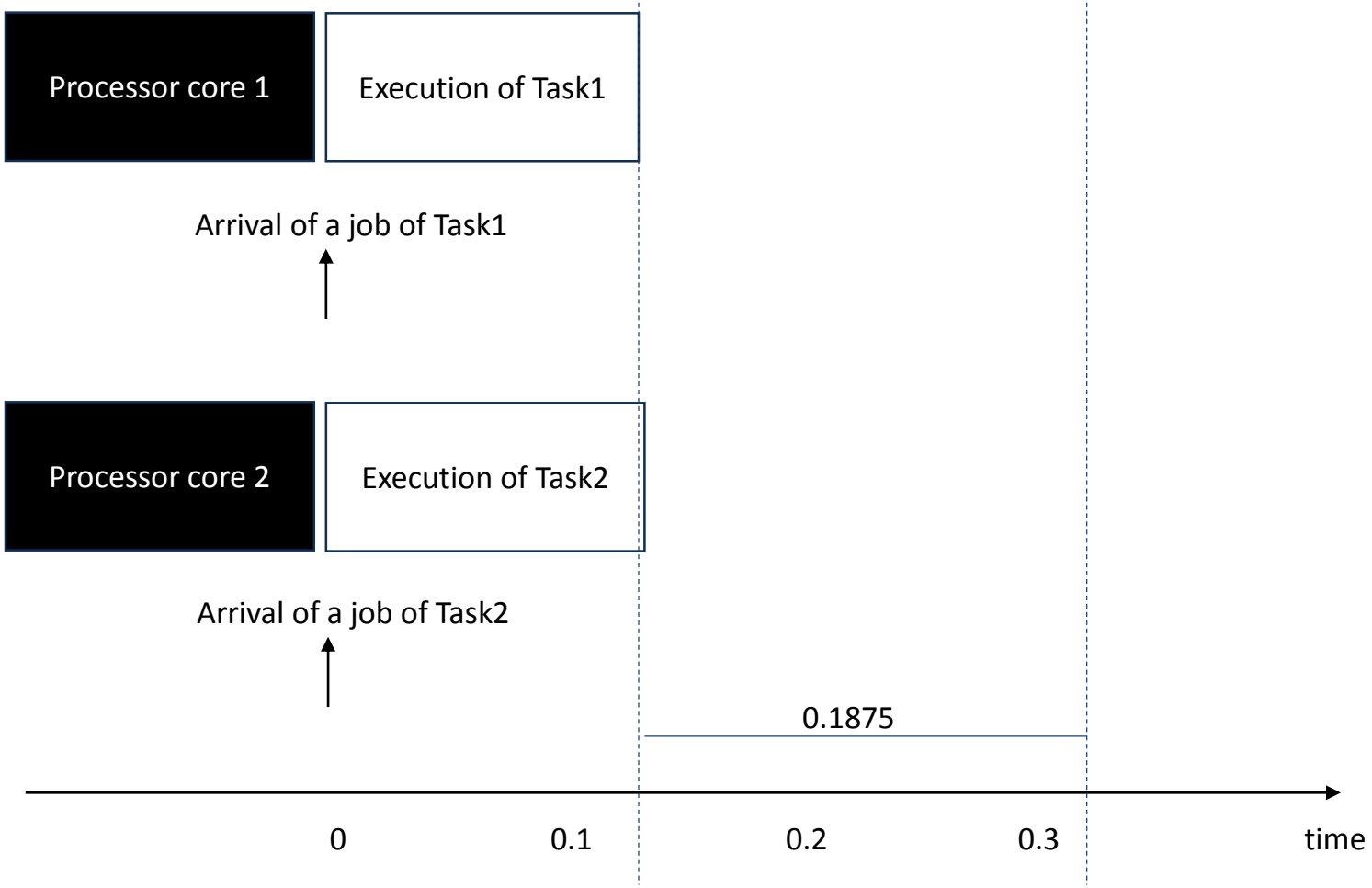
Here, Task2 has performed at least 0.125*0.5 units of execution.
Setup a timer to expire 0.1875 time units in the future.



How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Here, Task2 has performed at least 0.125*0.5 units of execution.
Setup a timer to expire 0.1875 time units in the future.

Timer expires
Check if Task2 still executes;
if so, stop Tasks2.

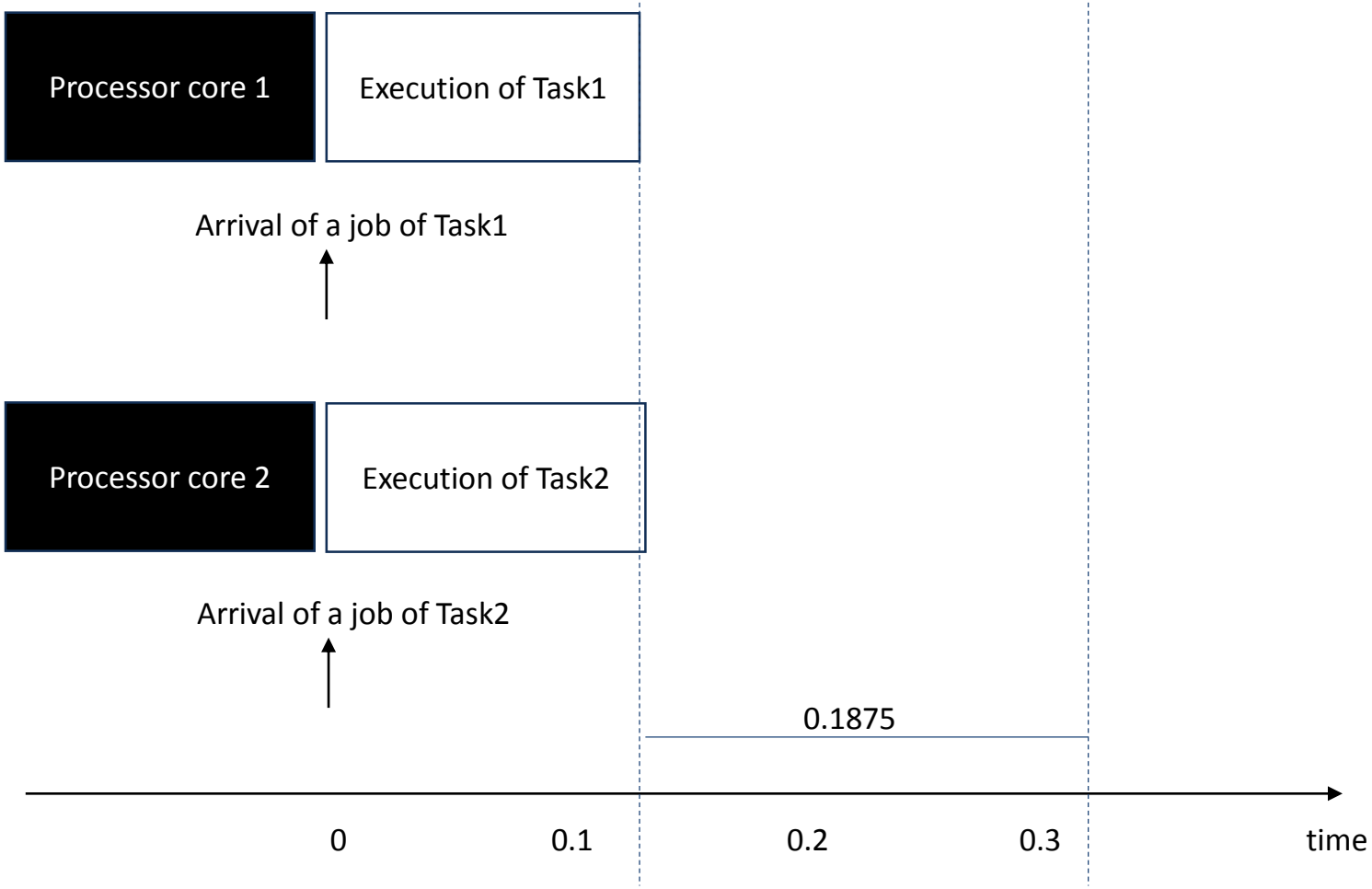


How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Here, Task2 has performed at least 0.125*0.5 units of execution.
Setup a timer to expire 0.1875 time units in the future.

See paper for more details.

Timer expires
Check if Task2 still executes;
if so, stop Tasks2.



How do we know that the behavior of tasks at run-time respects the model? We need to decide if a task has violated the parameters of the model.

Conclusions

Timing verification of software executing on undocumented multicore can be achieved through an abstraction that models the effect of shared hardware resources.

Run-time monitoring of execution time is possible by monitoring a lower bound on the cumulative number of units of execution that a task has performed.