

Research Review 2021

Using All Processor Cores While Being Confident about Timing

November 2021

Bjorn Andersson

Copyright 2021 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.

DM21-0810

Complex, cyber-physical DoD systems depend on correct timing—any timing failure could be disastrous. What's more, while these systems drive demand for use of multicore processors, concern about timing has led to disabling all processor cores except one—limiting system capability.

We aim to develop a solution to overcome this obstacle.

DoD Systems Interact with Their Physical Environment



DoD Systems Include Software



DoD Systems Include Software That Interacts with the Physical Environment



DoD Systems Include Software That Has Real-Time Requirements



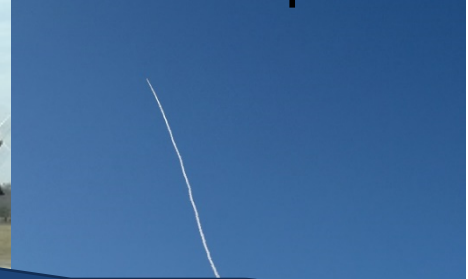
Satisfying Real-Time Requirements Is a Challenge for the DoD in General



Satisfying Real-Time Requirements Is Challenging for Upgrading the Blackhawk UH-60 Helicopter



Satisfying Real-Time Requirements Is Challenging for Upgrading the Blackhawk UH-60 Helicopter

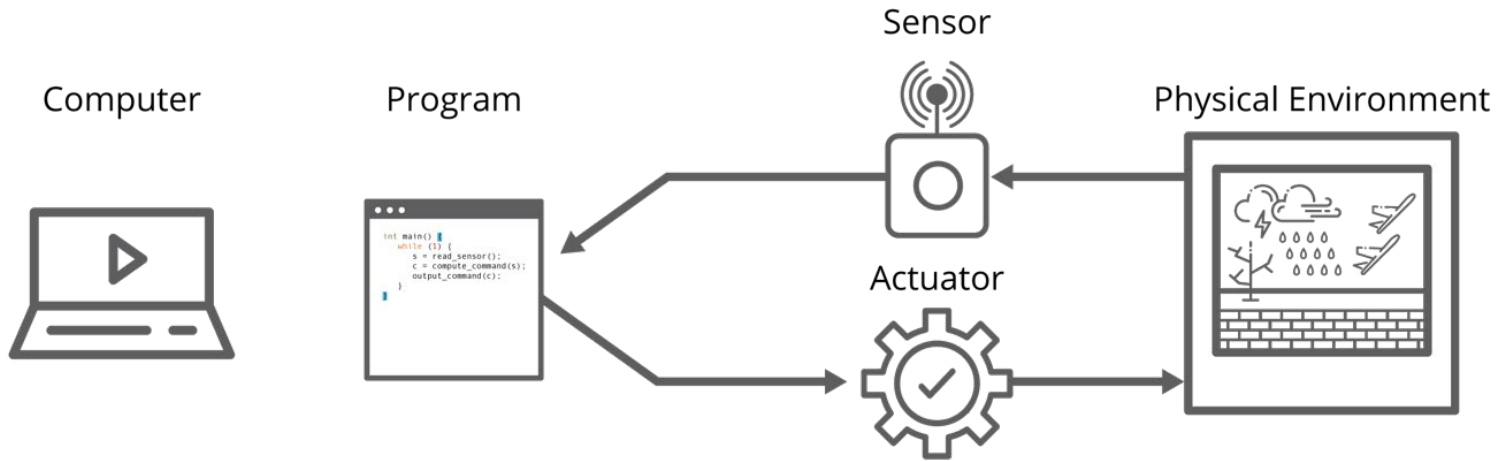


“The trick there, when you’re processing flight critical information, it has to be a deterministic environment, meaning we know exactly where a piece of data is going to be exactly when we need to — no room for error,” Langhout says. “On a multi-core processor there’s a lot of sharing going on across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

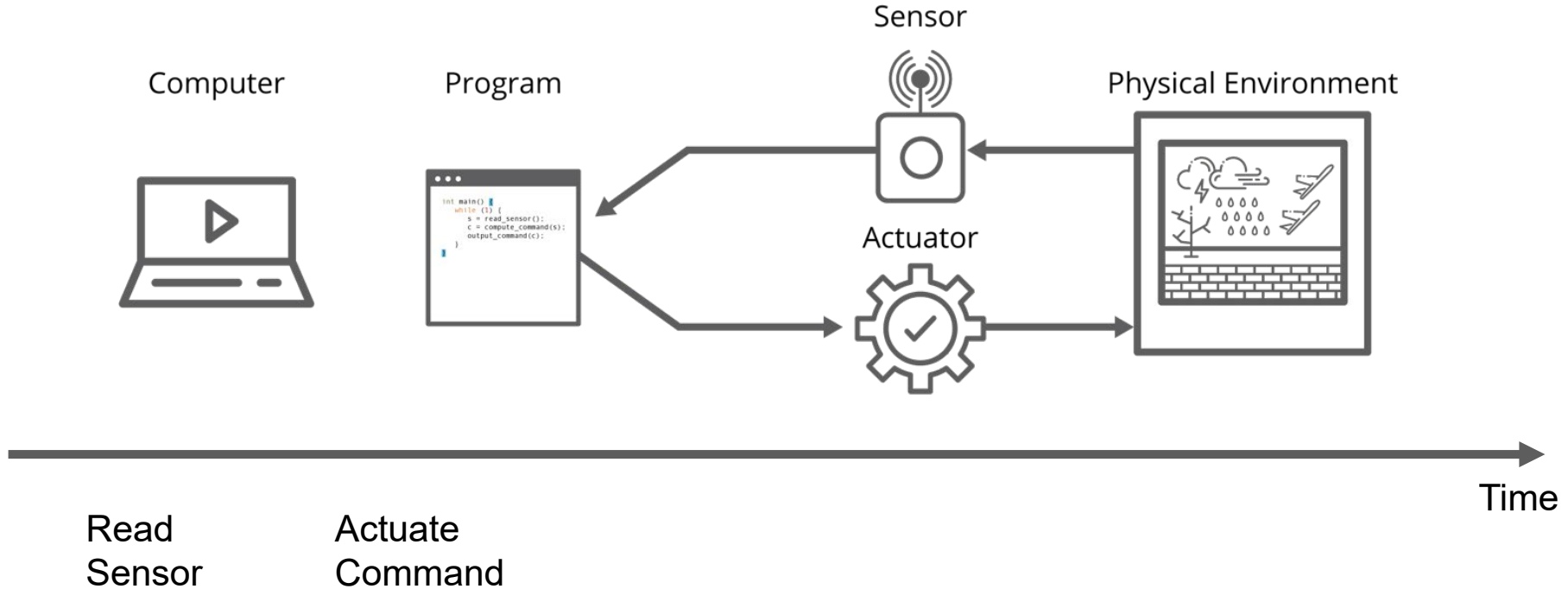
- Jeff Langhout, Acting Director, U.S. Army Aviation and Missile Research Development and Engineering Center (AMRDEC)

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/>.

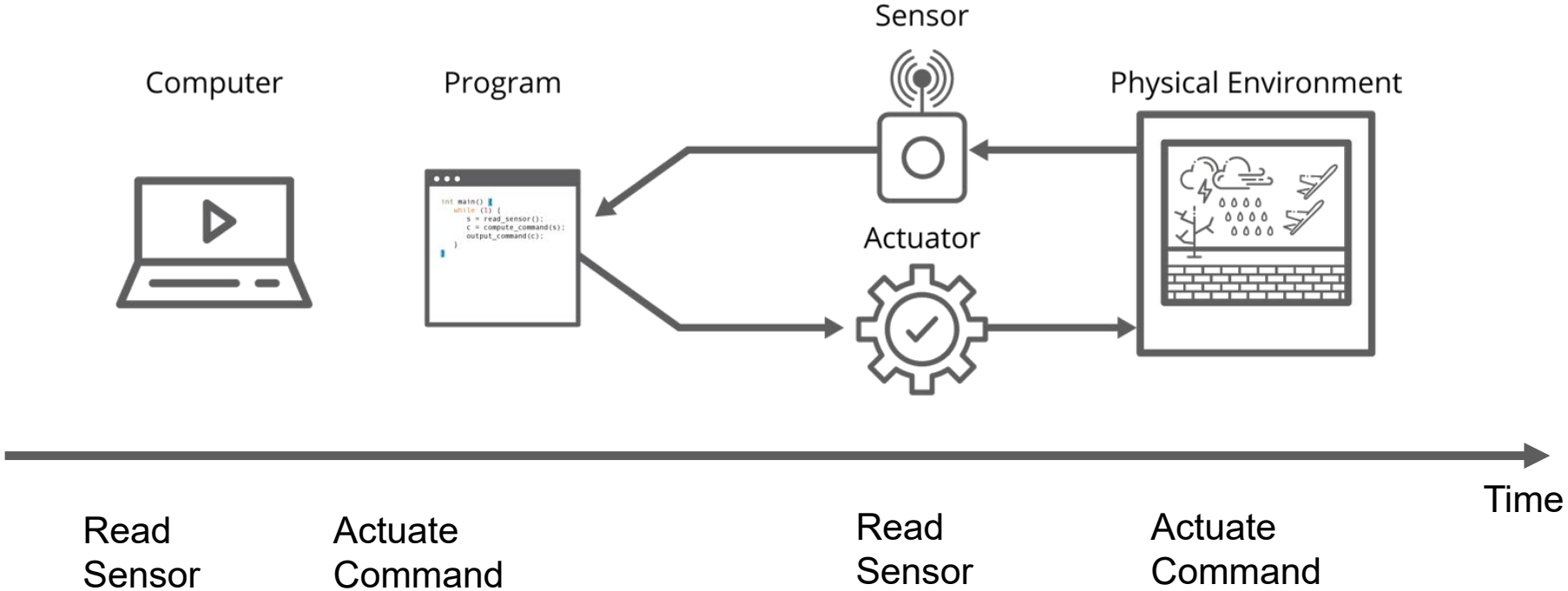
Commonality of DoD Systems



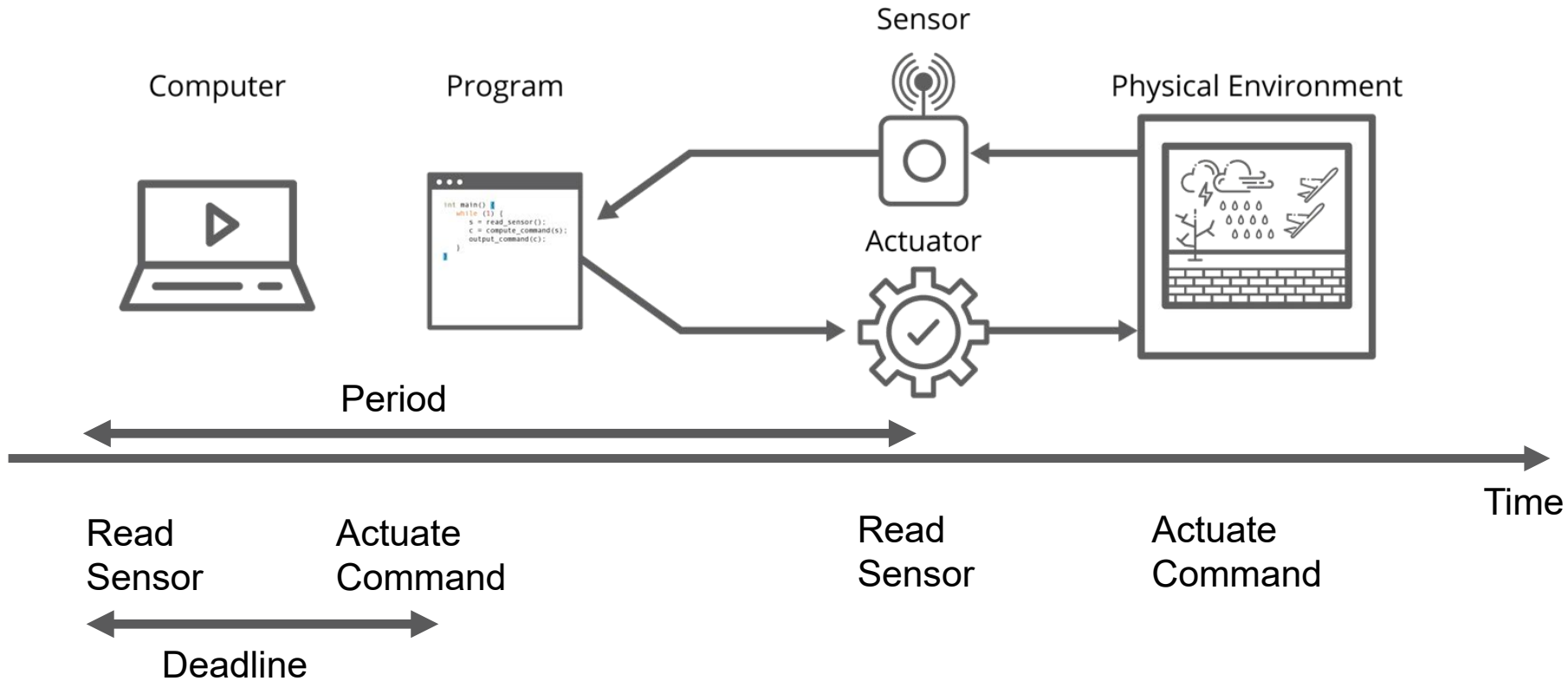
Commonality of DoD Systems



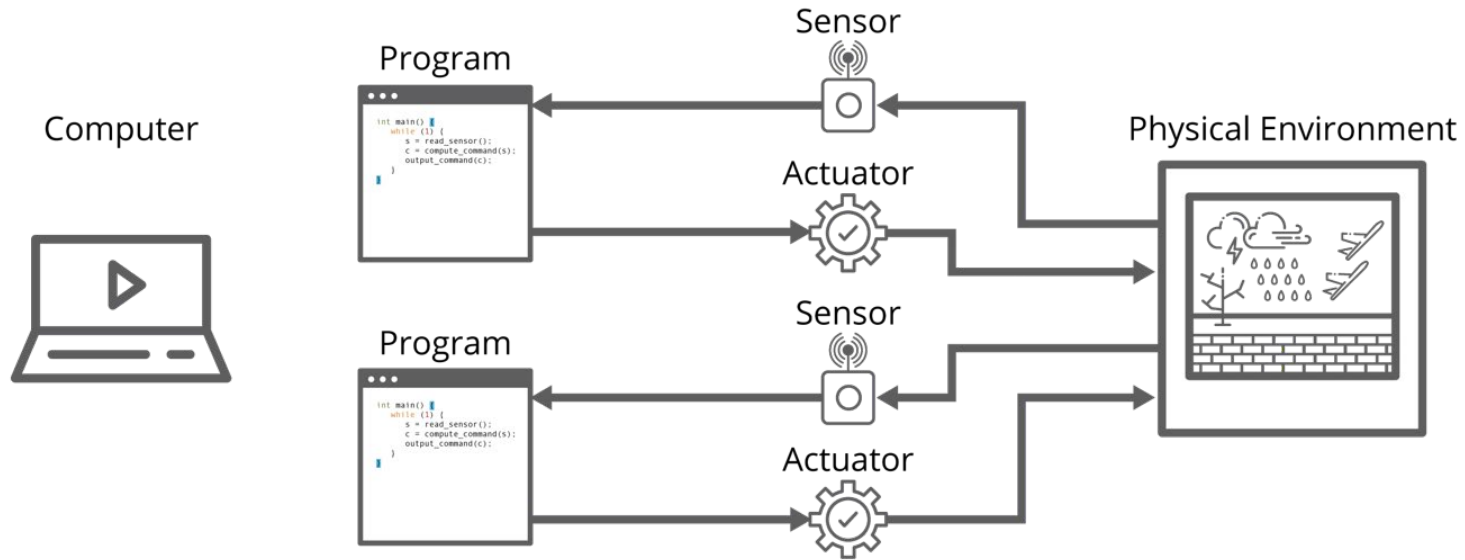
Commonality of DoD Systems



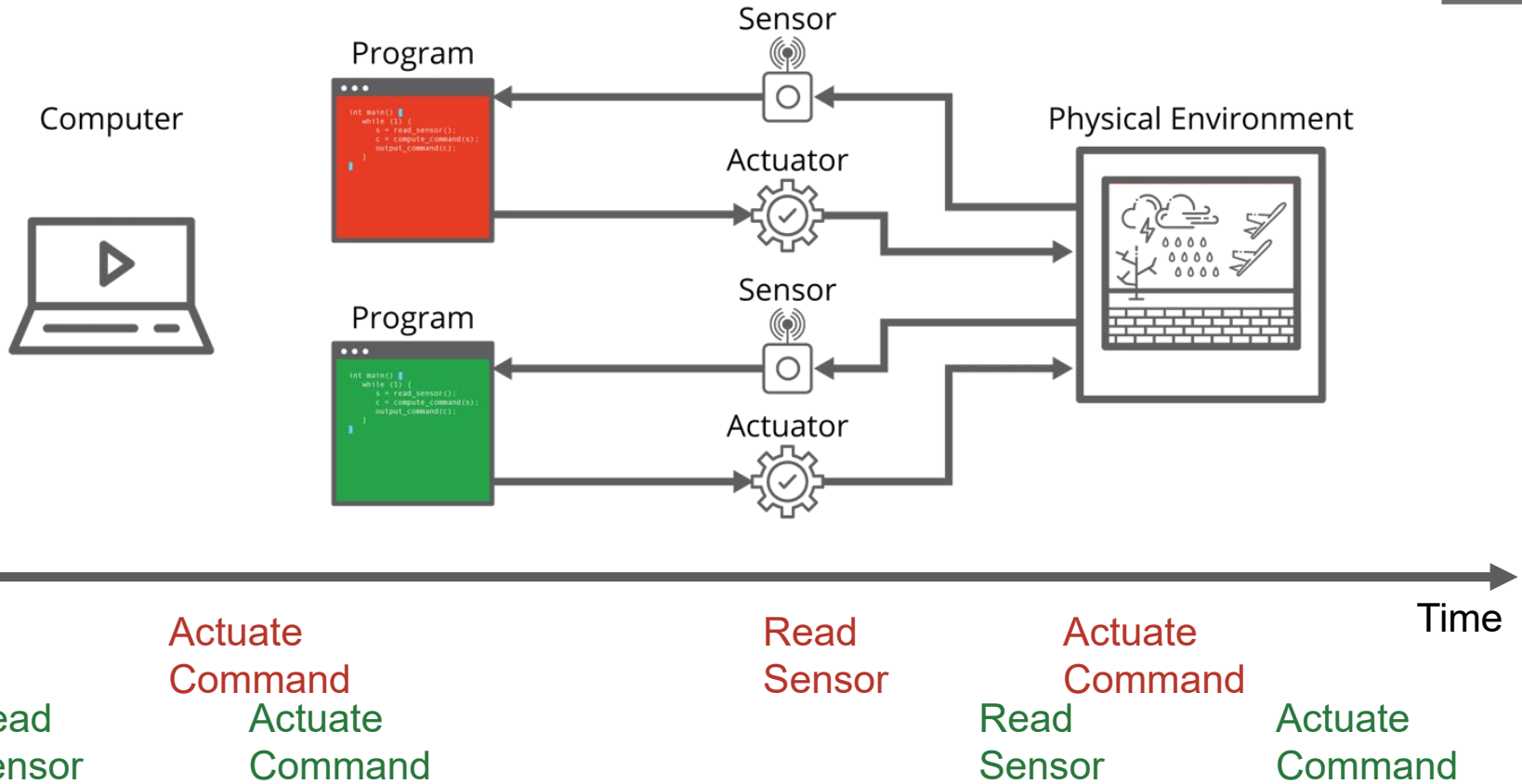
Commonality of DoD Systems



Commonality of DoD Systems



Commonality of DoD Systems



Why is Satisfying Real-Time Requirements Challenging?


What Causes Delay of Software?

What Causes Delay of Software?



Time

What Causes Delay of Software?



Time when one thread
in the software system arrives

Deadline

Time

What Causes Delay of Software?

Thread executes
one path



Time when one thread
in the software system arrives

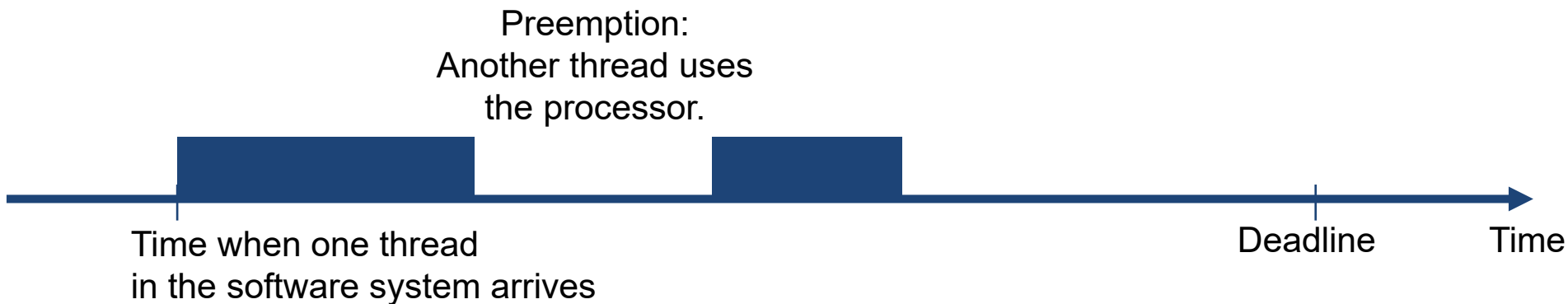
Deadline

Time

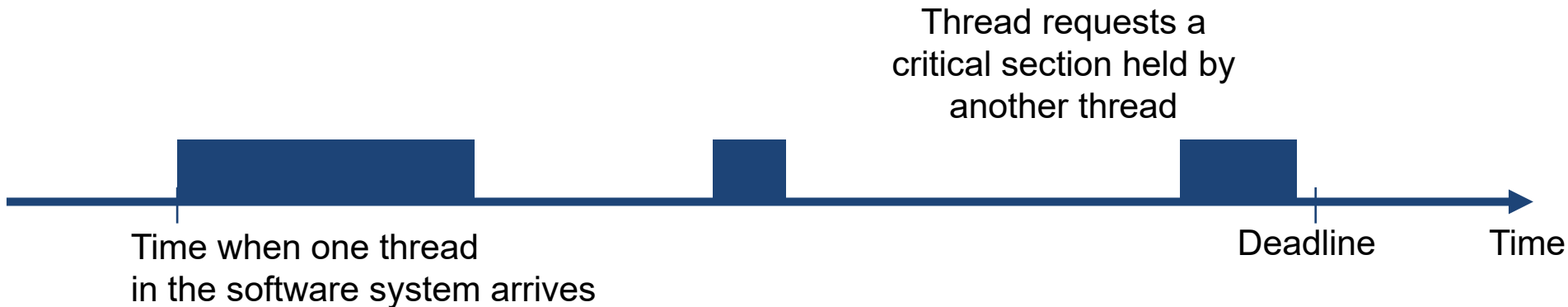
What Causes Delay of Software?



What Causes Delay of Software?



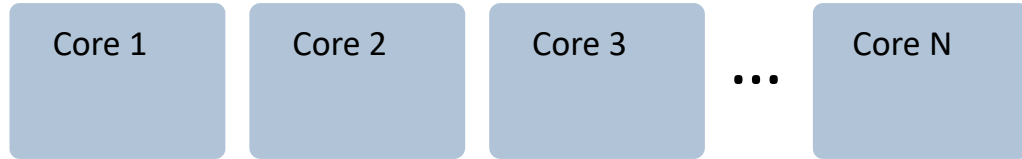
What Causes Delay of Software?



Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

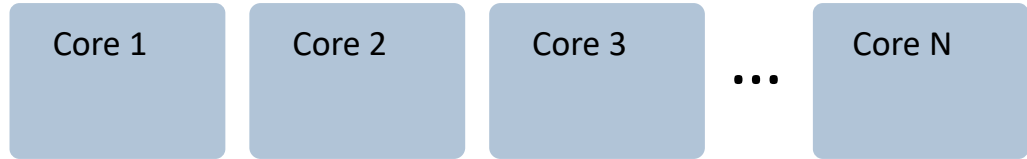
- *All computers are multicores.*



Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

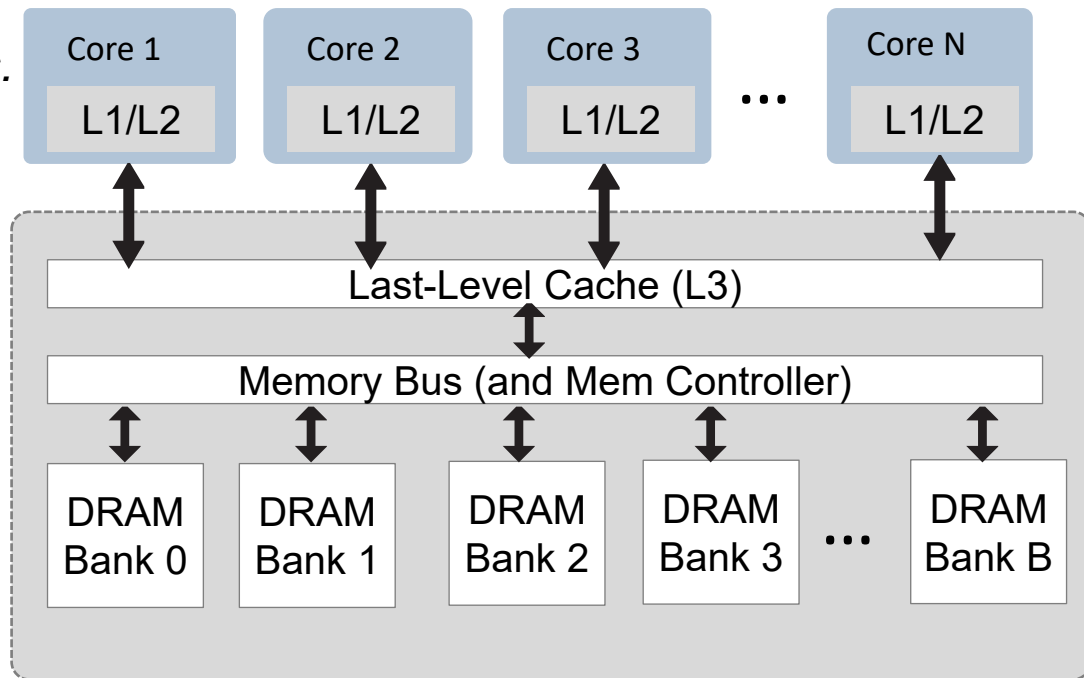
- *All computers are multicores.*
- *Most chip makers do not offer single core.*

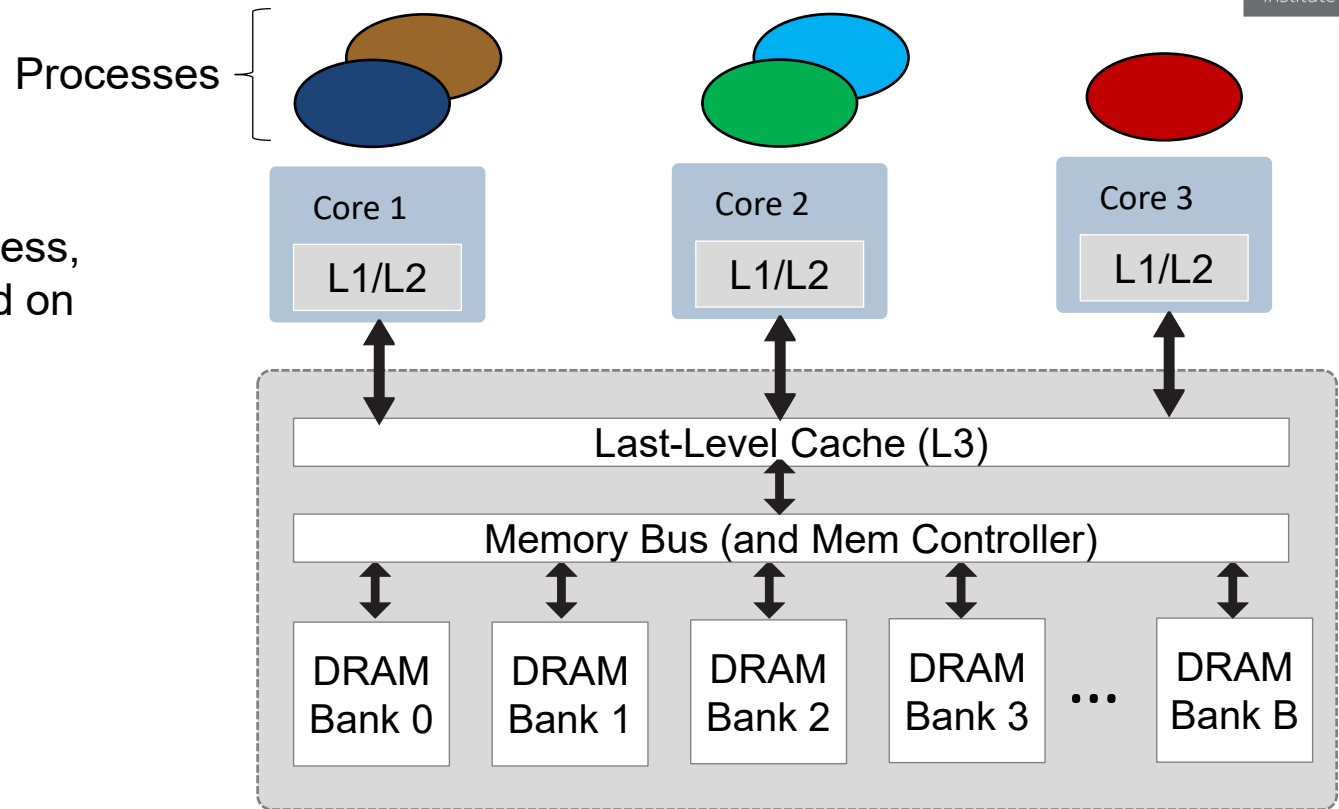


Real-Time Requirements of Software Executing on a Multicore Processor

Hardware Trends

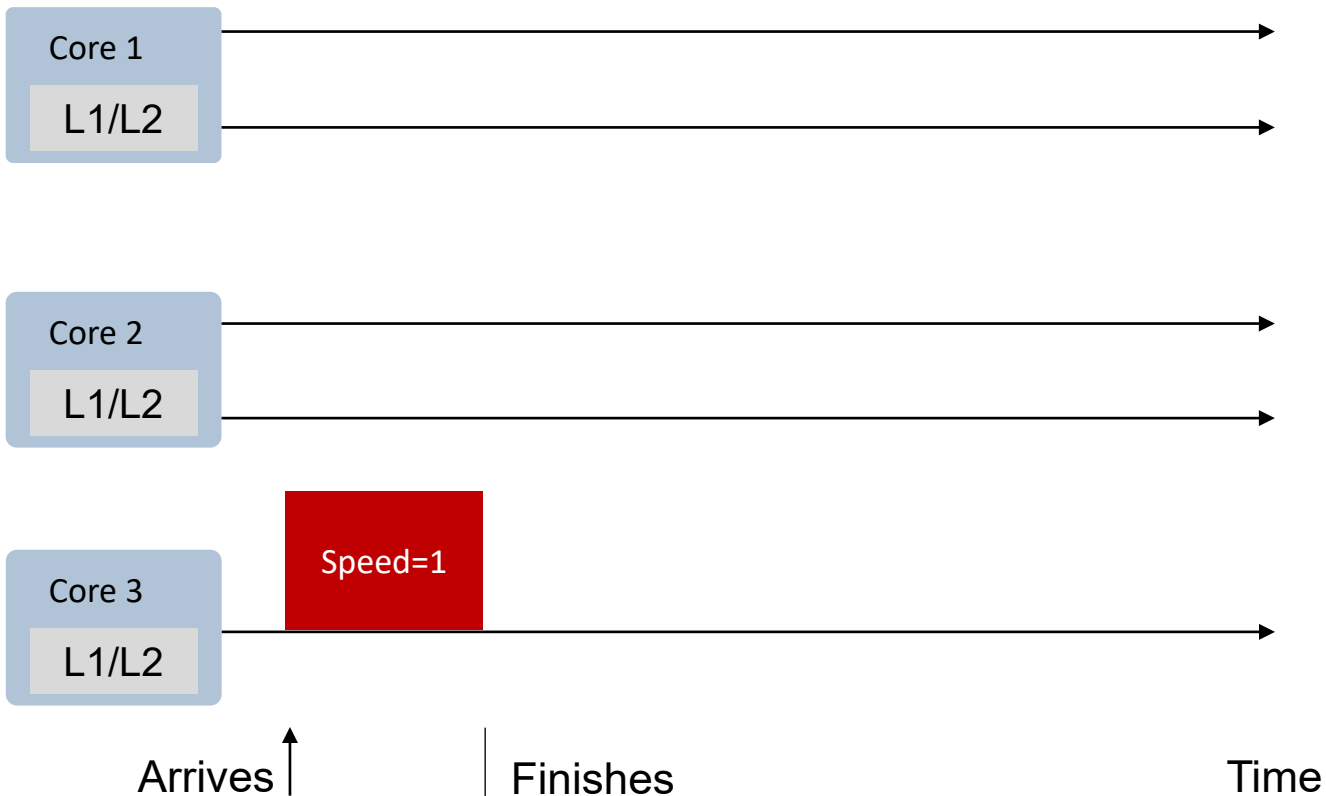
- *All computers are multicores.*
- *Most chip makers do not offer single core.*
- *Most multicores have shared memory.*



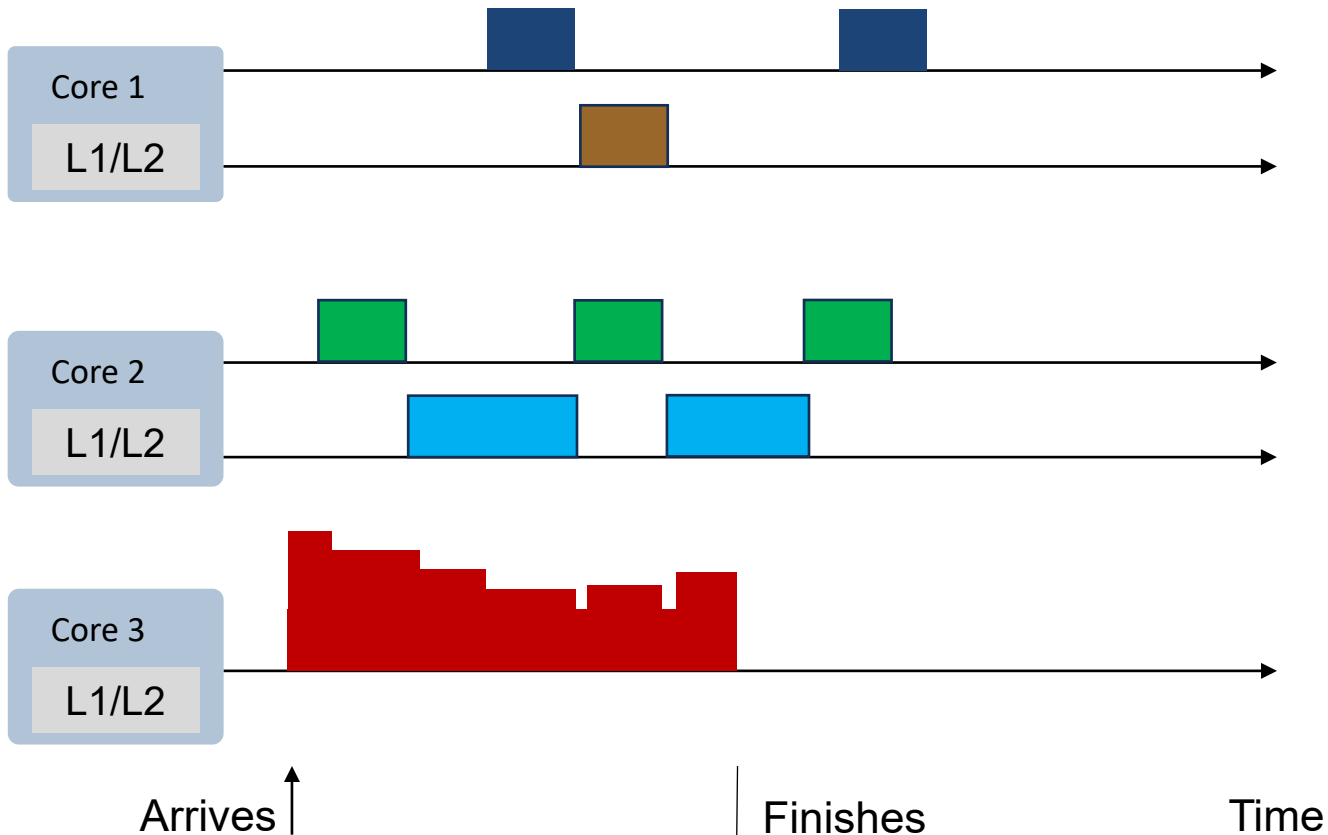


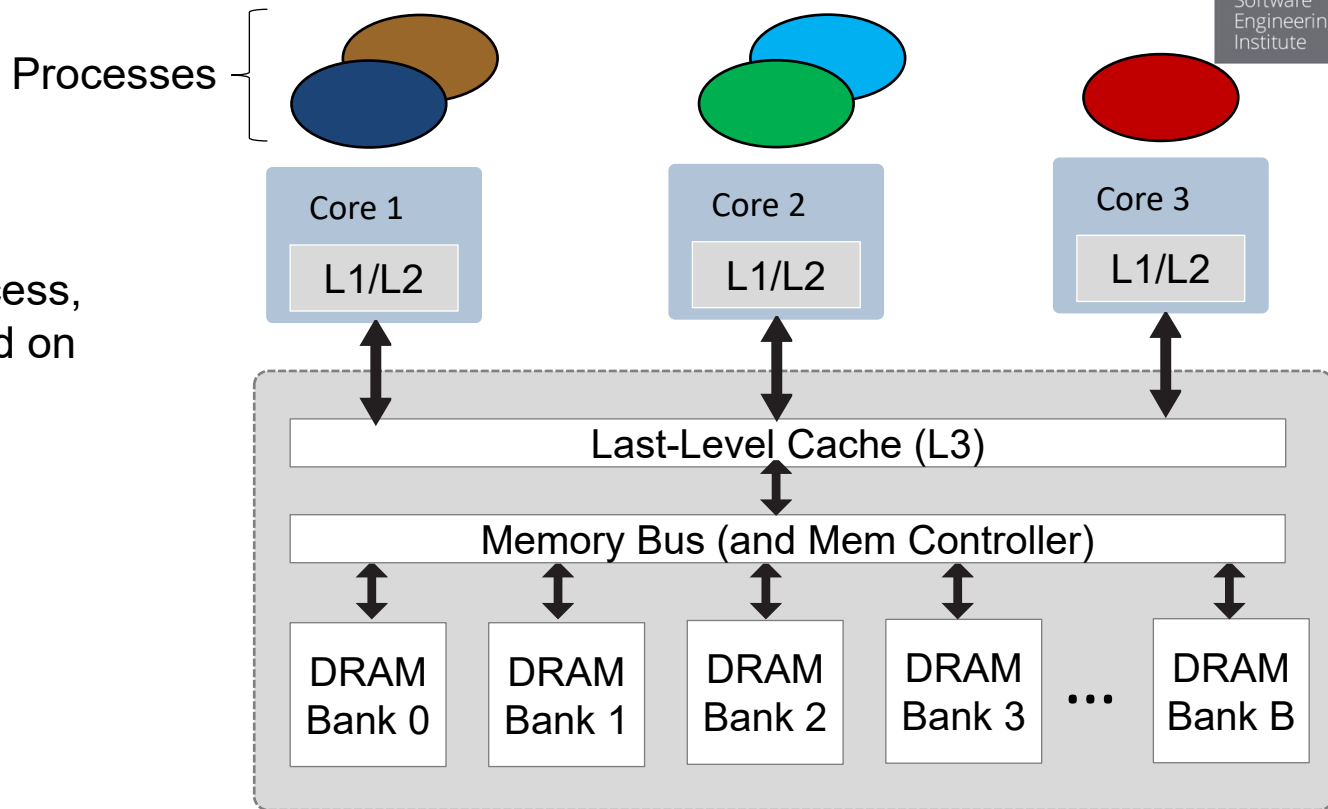
Problem: For each process, compute an upper bound on its response time.

How Co-Runners Impact Speed of Execution



How Co-Runners Impact Speed of Execution



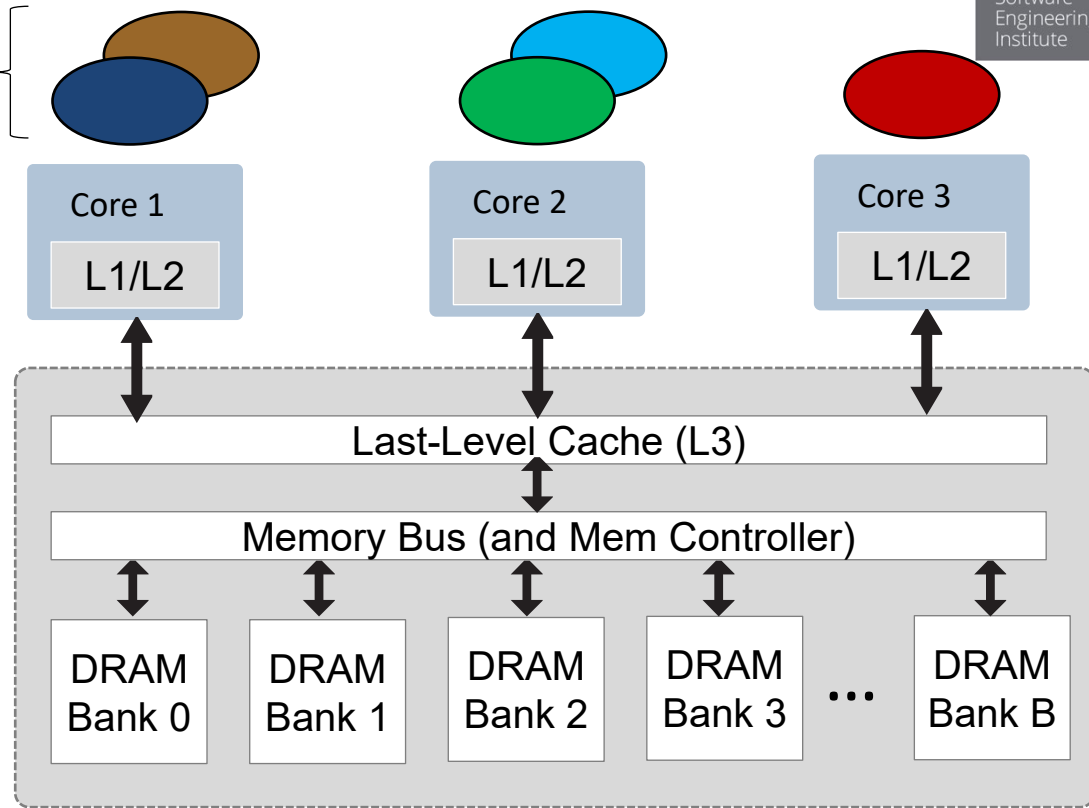


Problem: For each process, compute an upper bound on its response time.

Issues

- Shared hardware resources impact timing.

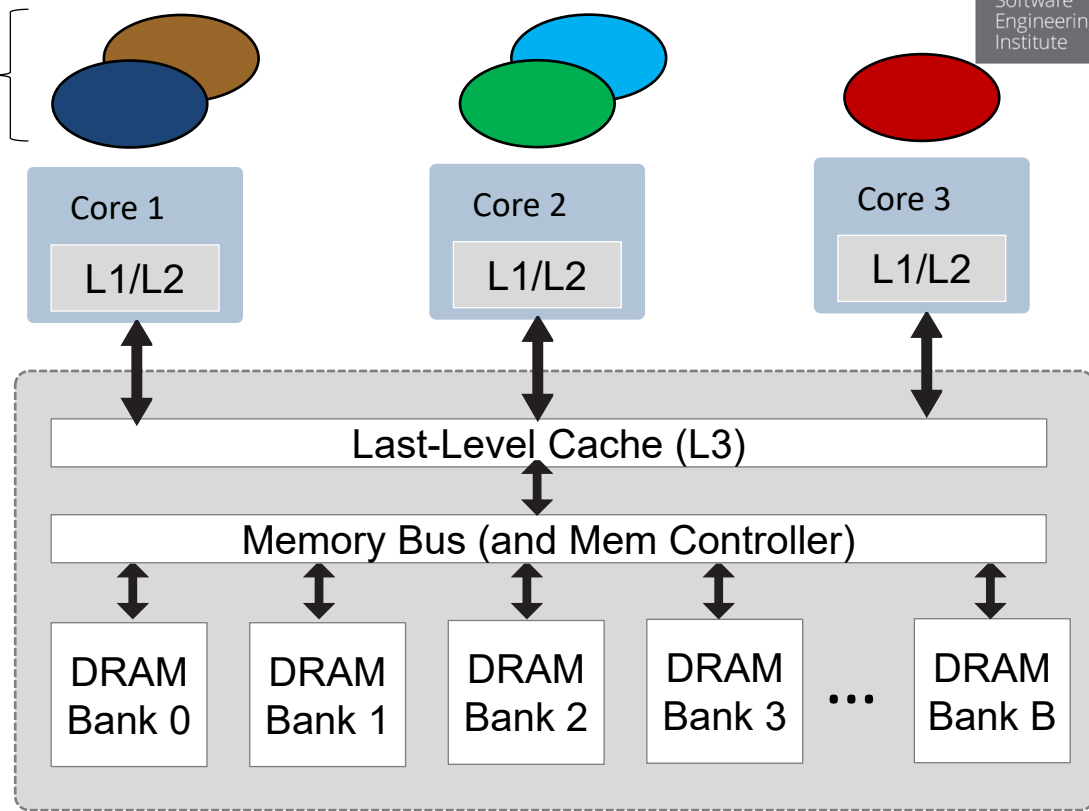
Processes



Issues

- Shared hardware resources impact timing.
- 103 times slowdown has been observed [Yun15].

Processes

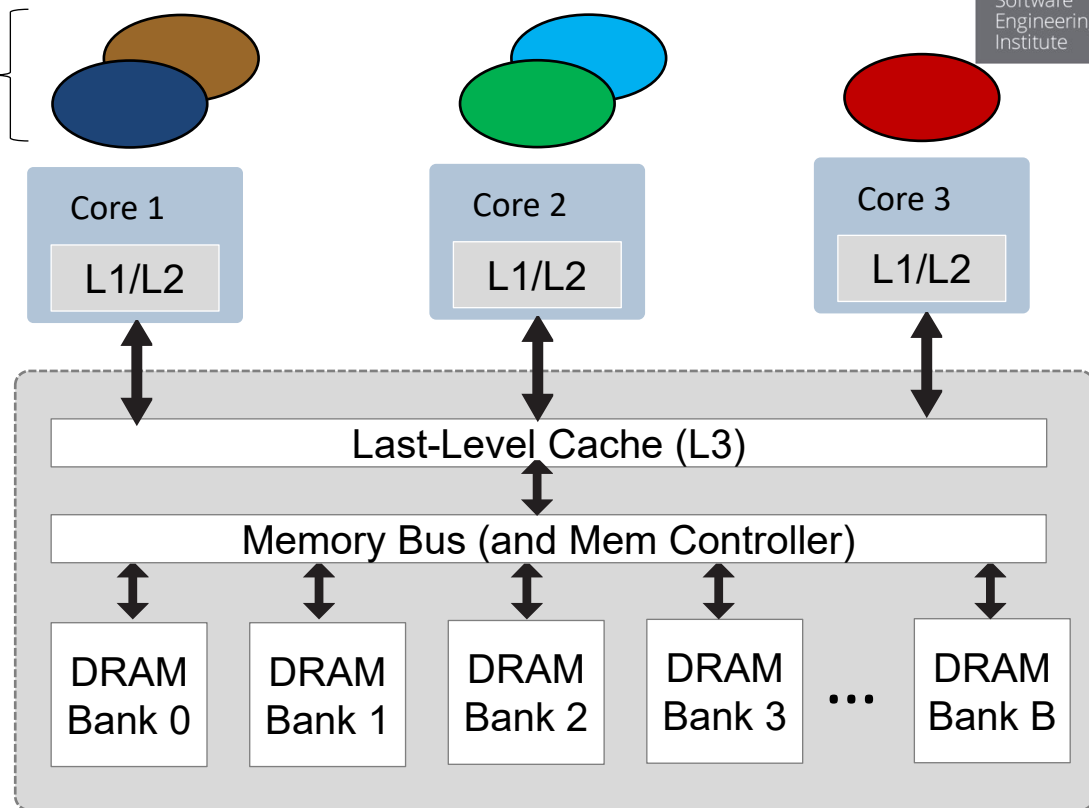


[Yun15] H. Yun and P. K. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," OSPERT, 2015.

Issues

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

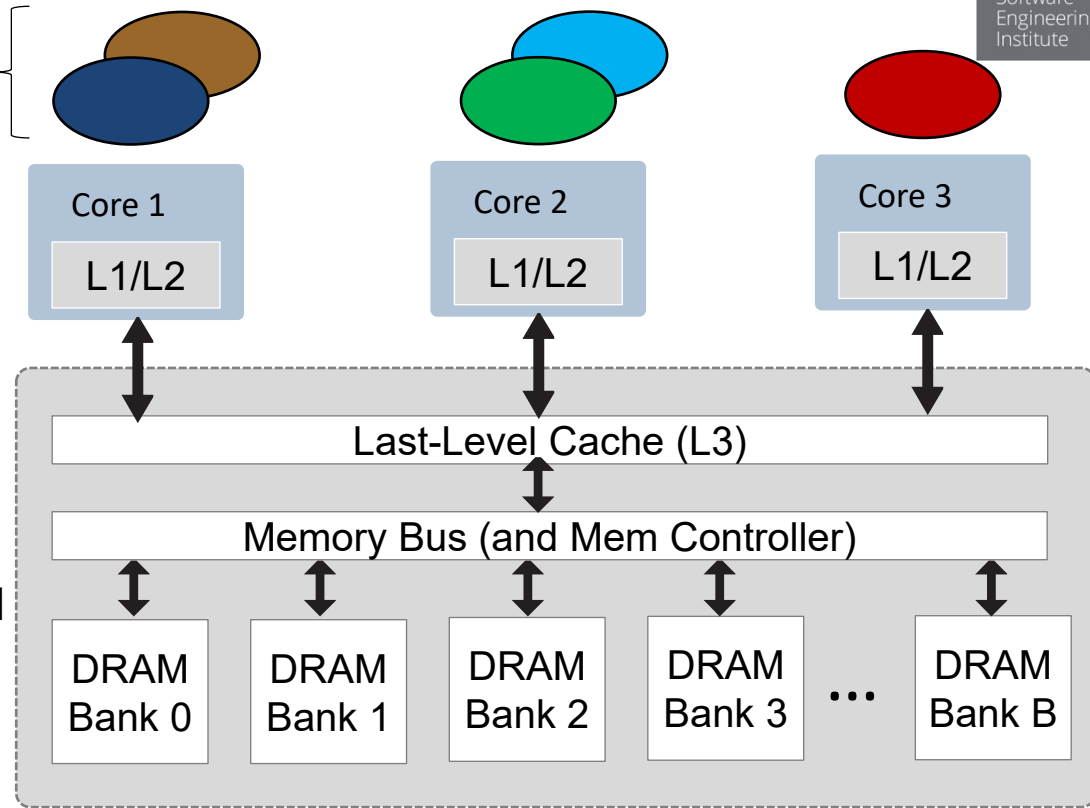
Processes



Issues

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

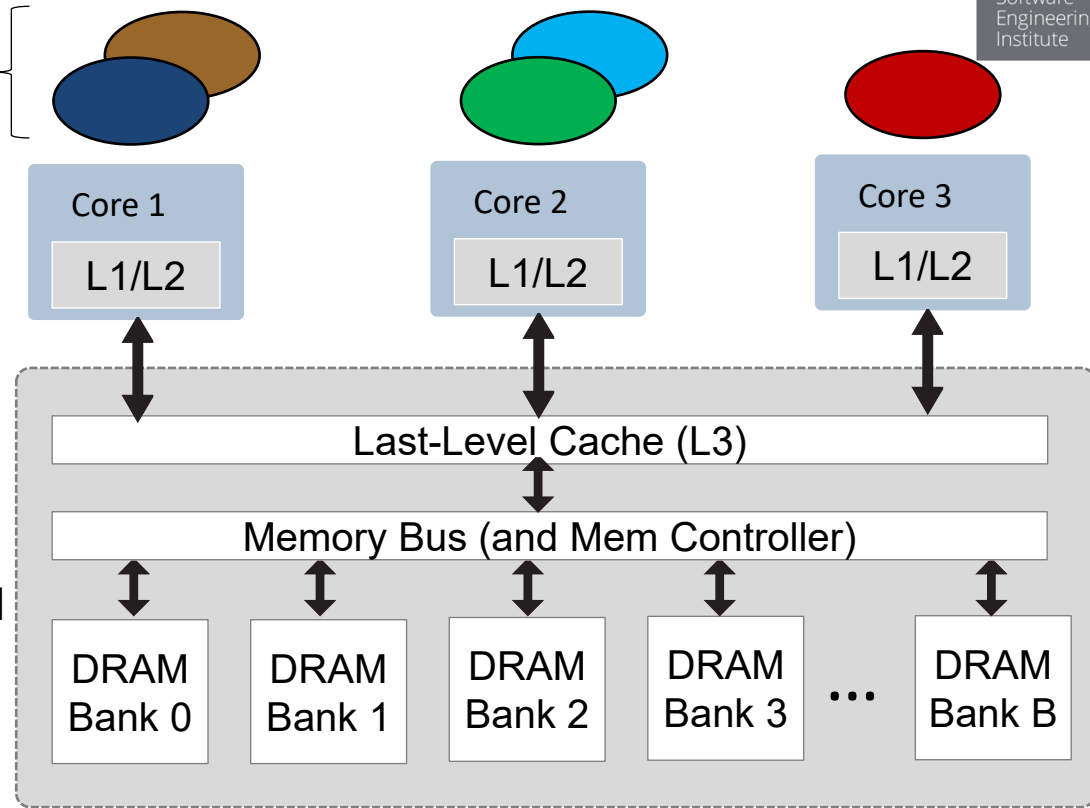
Processes



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

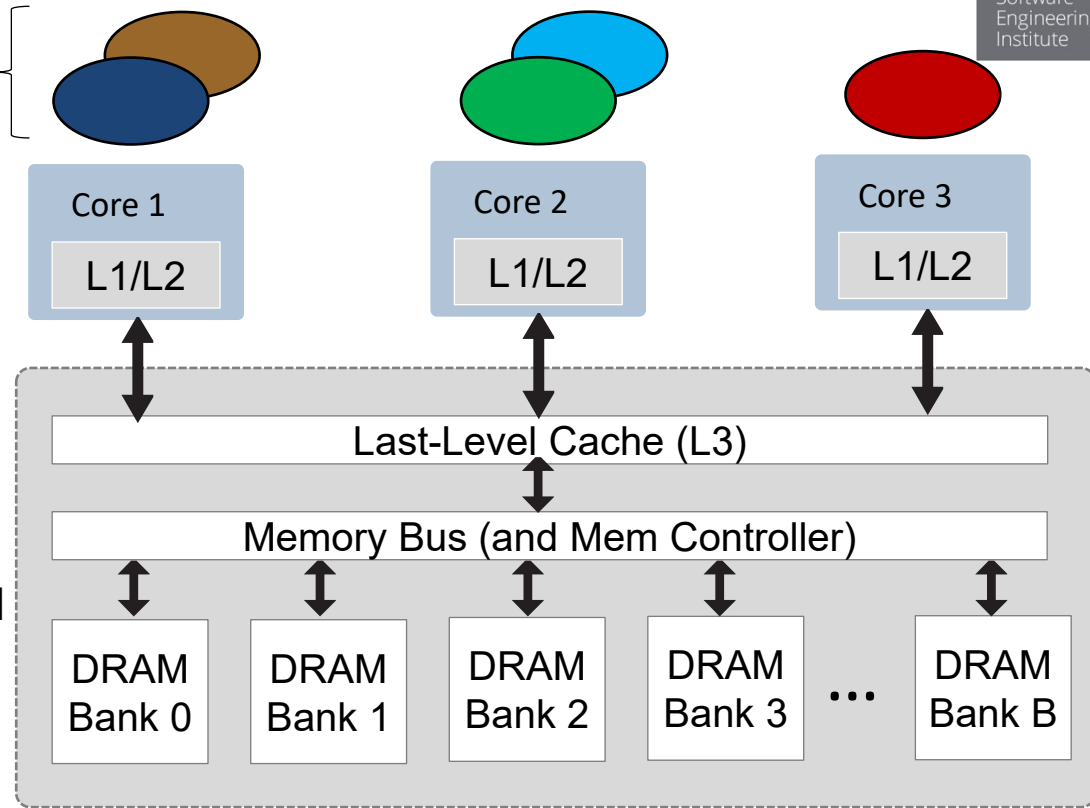
Processes



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



We need a new method to compute response times of processes.

Overview of Our Solution

Does it hold that for all scenarios, for each task, all its deadlines are met?

yes/no/
undecided →

Overview of Our Solution

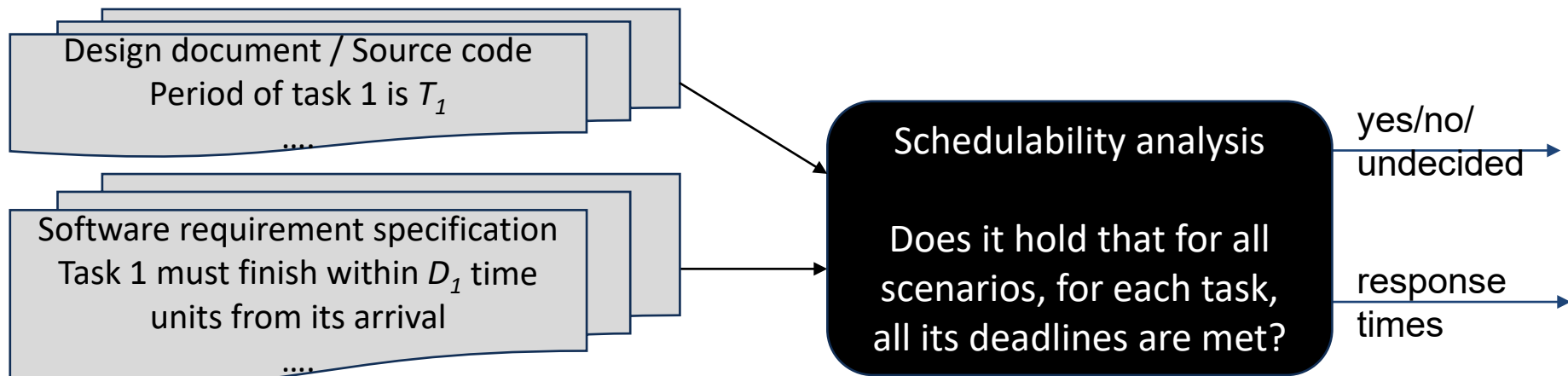
Schedulability analysis

yes/no/
undecided →

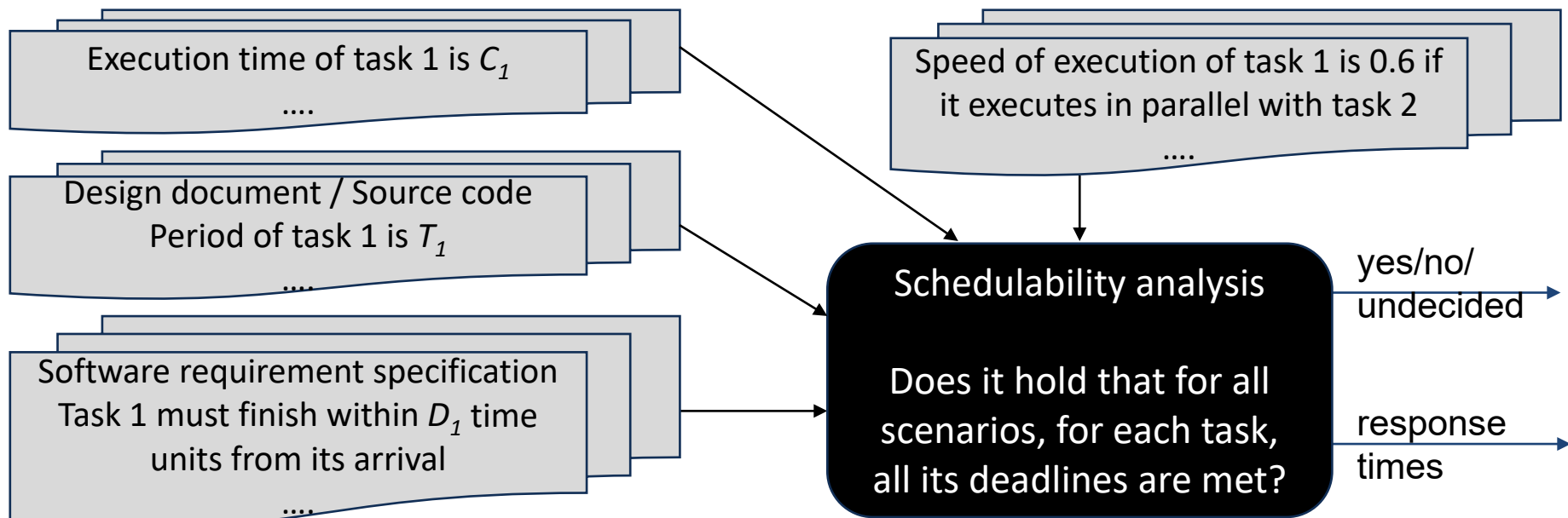
Does it hold that for all
scenarios, for each task,
all its deadlines are met?

response
times →

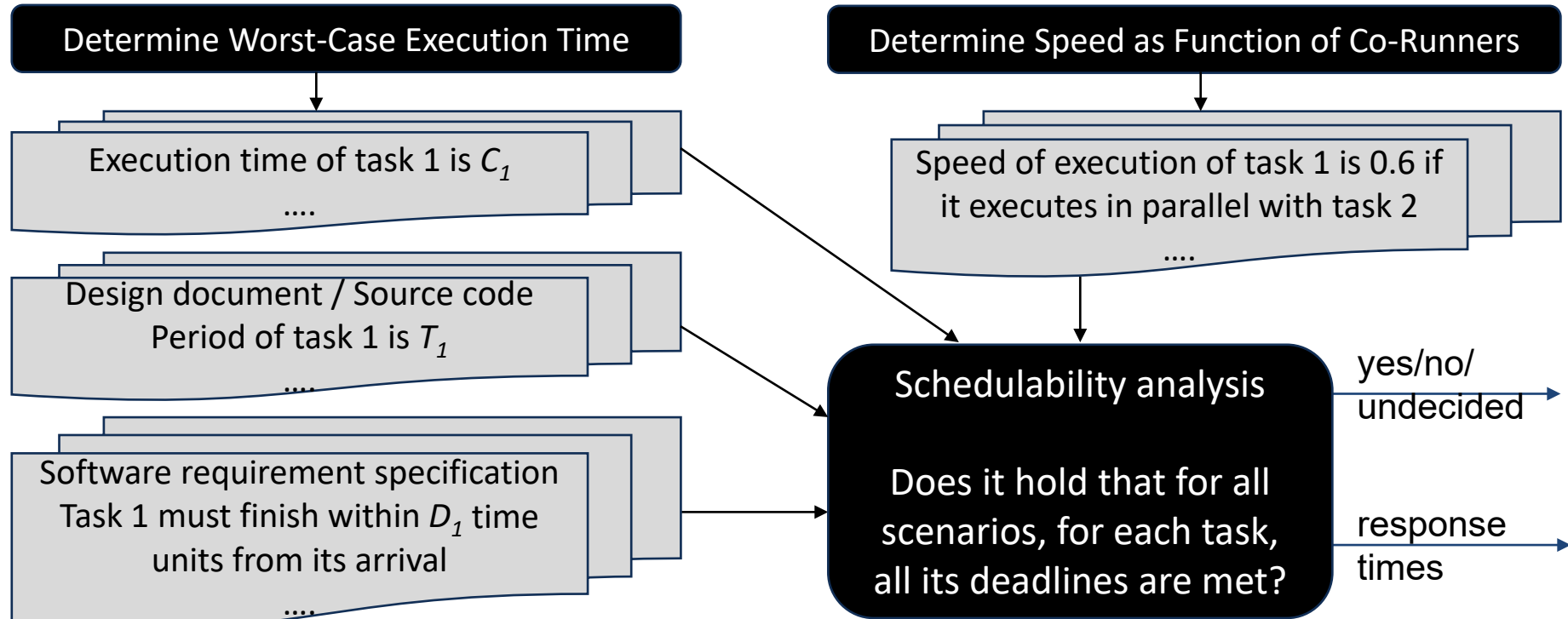
Overview of Our Solution



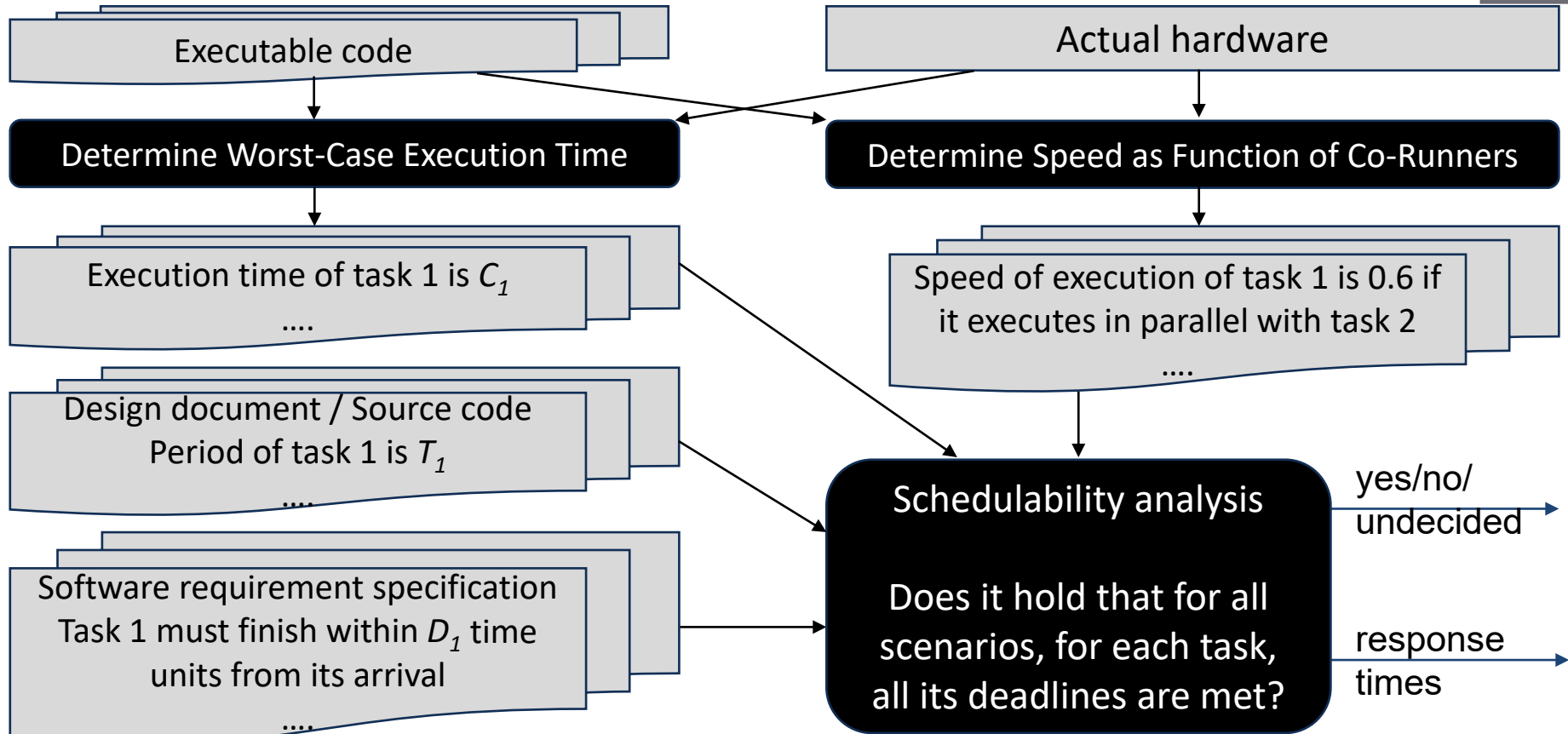
Overview of Our Solution



Overview of Our Solution



Overview of Our Solution



Our Tools Perform These Tasks

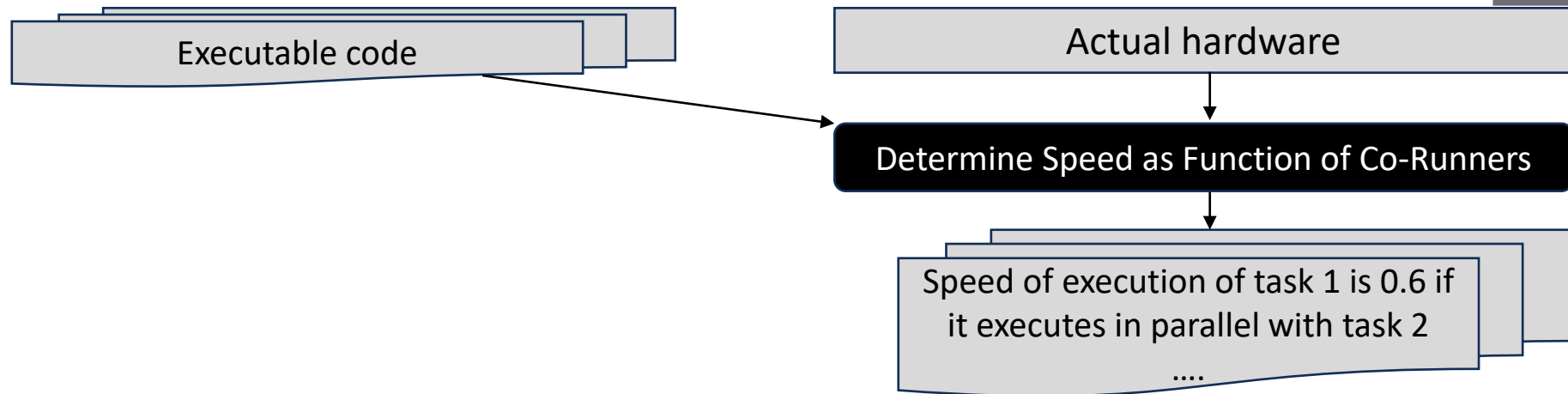
Determine Worst-Case Execution Time

Determine Speed as Function of Co-Runners

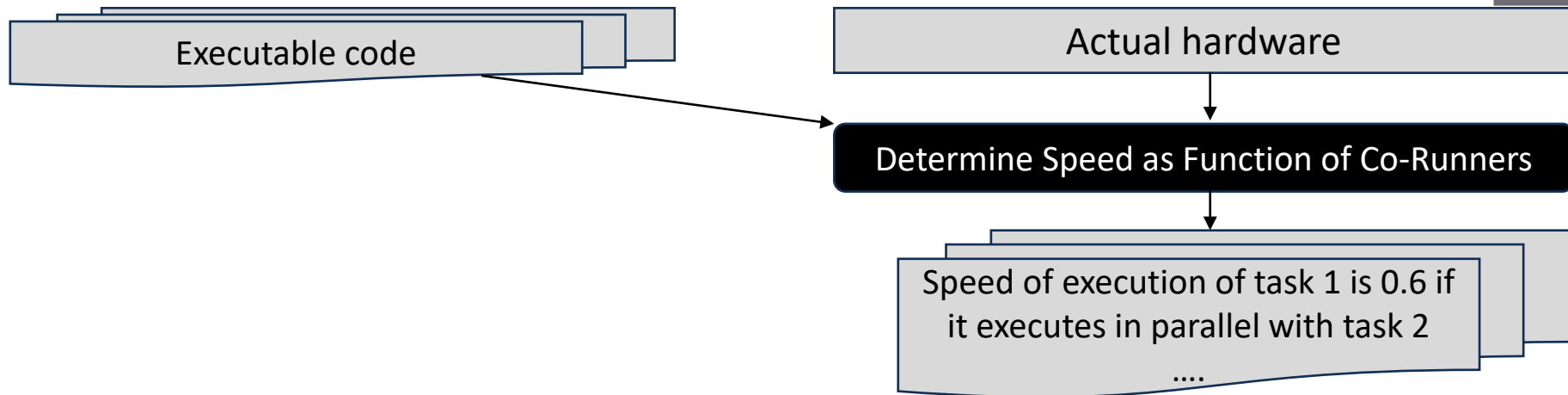
Schedulability analysis

Does it hold that for all scenarios, for each task, all its deadlines are met?

Our Abstraction Speed as Function of Co-Runners Is New



Our Abstraction Speed as Function of Co-Runners Allows undoc hw



A Look at Our Tools - 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				
Segment 1								1]], 1.0], [[[5, 1]], 1.0], [[[5, 2]], 1.0]]
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

Main Idea

Fixed-point iteration
where each iteration
solves a linear program

A Look at Our Tools - 2

The screenshot shows the Eclipse IDE interface for a project named 'runtime-osate2 - myprojectwithtaskset/mypakwithtaskset.aadl - OSATE2'. The AADL Navigator on the left shows a tree structure with 'instances' expanded to 'mypakwithtaskset_mysystem_imp_Instance.aadl'. The main editor displays AADL code for 'mypakwithtaskset.aadl', including thread implementations for 'mythread1', 'mythread2', and 'mythread3'. A 'Message' dialog box is overlaid on the editor, displaying the following text:

```

Message
Taskset is schedulable
Upper bounds on the response times of task are as follows:
For task 1: 6.875526837243904
For task 2: 61.749311415110206
For task 3: 43.685004903641804
  
```

The status bar at the bottom indicates 'myprojectwithtaskset/instances/mypakwithtaskset_mysystem_imp_Instance.aadl2' and 'Schedulability Analysis ...ution Times'.

A Look at Our Tools - 3

```

ba@ba-desktop: ~/ga_find_wcet
File Edit View Search Terminal Help
ba@ba-desktop:~/ga_find_wcet$ more myconf
0
262144
ba@ba-desktop:~/ga_find_wcet$ ./ga_find_wcet myconf ./bubblesort
0.306359
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < ascending_integers.dat

real    0m0.002s
user    0m0.001s
sys     0m0.001s
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < descending_integers.dat

real    0m0.275s
user    0m0.275s
sys     0m0.000s
ba@ba-desktop:~/ga_find_wcet$ time ./bubblesort < worstcaseinput_GA_inputfile.dat

real    0m0.308s
user    0m0.308s
sys     0m0.000s
ba@ba-desktop:~/ga_find_wcet$ █

```

Main Idea

Use genetic algorithms

Fitness = execution time

Gene = input to program

Maximize fitness

=

Maximize execution time

A Look at Our Tools - 4

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. It also implements other schedulability tests.

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	Executable code (filename)	Input size (bits)
Task 1	1.5	1.5	1	3	1					
Segment 1						0.102672	0.5	[[[[[3, 1]], 0.9275886057079874], [[4, 1]], 0.867521186977716]]	memintensive2	60000
Segment 2										
Segment 3										
Task 2	2.0	2.0	1	2	1					
Segment 1						0.009247	0.5	[[[[[3, 1]], 0.9380198823290727], [[4, 1]], 0.84370437956204]]	bubblesort	60000
Segment 2										
Segment 3										
Task 3	2.0	2.0	1	3	2					
Segment 1						0.09486	0.5	[[[[[1, 1]], 0.9261410788381742], [[2, 1]], 0.77591918530939]]	memintensive2	60000
Segment 2										
Segment 3										
Task 4	2.0	2.0	1	2	2					
Segment 1						0.009334	0.5	[[[[[1, 1]], 0.9375251104861391], [[2, 1]], 0.83758076094759]]	bubblesort	60000
Segment 2										
Segment 3										
Task 5	2.25	2.25	2	1	2					
Segment 1						0.018271	1.0	[[[[[1, 1]], 0.9265682843957603], [[2, 1]], 0.83478777356421]]	bubblesort	85000
Segment 2						0.004043	0.5	[[[[[1, 1]], 0.9972866304884064], [[2, 1]], 0.84369782971619]]	bubblesort	38000
Segment 3										
Task 6										
Segment 1										
Segment 2										
Segment 3										
Task 7										
Segment 1										
Segment 2										
Segment 3										
Task 8										
Segment 1										
Segment 2										
Segment 3										
Task 9										
Segment 1										

Load taskset from file taskset.txt Save taskset to file taskset.txt Do schedulability analysis Options... Do experiments... Obtain parameters empirically

Main Idea

Use genetic algorithms

Fitness = slowdown

Gene = input to program

Maximize fitness

=

Maximize slowdown

Publications

Andersson, B.; Kim, H.; de Niz, D.; Klein, M.; Rajkumar, R.; & Lehoczky, J. *Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times*. *ACM Transactions on Embedded Computing Systems*. Volume 17. Number 3. Article 71. May 2018.

Andersson, B.; de Niz, D.; & Klein, M. *The Multicore Challenge in Assured Autonomy*. Presented at AUVSI Exponential. August 2021.

Kim, H.; de Niz, D.; Andersson, B.; Klein, M.; & Lehoczky, J. *Addressing Multi-Core Timing Interference using Co-Runner Locking*. To be presented at IEEE Real-Time Systems Symposium. December 2021.

Petrucci, V.; Andersson, B.; Massa, E.; & Lima, G. *Heterogeneous Quasi-Partitioned Scheduling*. To be presented at IEEE Real-Time Systems Symposium. December 2021.

Tools

<http://www.andrew.cmu.edu/user/banderss/projects.html>

Transition success

Tools have been used in internal seminars at Army CCDC Aviation & Missile Center (AvMC)

Team Members



Bjorn Andersson
Principal Researcher
Software Solutions Division
Software Engineering Institute



Dionisio Di Niz
Technical Director, Assuring
CyberPhysical Systems
Software Solutions Division
Software Engineering Institute



Mark Klein
Principal Technical Advisor
Software Solutions Division
Software Engineering Institute



John Lehoczky
Thomas Lord University
Professor of Statistics
Carnegie Mellon University



Hyoseung Kim
Associate Professor
Department of Electrical and
Computer Engineering
University of California,
Riverside



Bill Anderson
Senior Member of the
Technical Staff (retired)
Software Solutions Division
Software Engineering Institute



Anton Dimov Hristozov
Software Engineer
Software Solutions Division
Software Engineering Institute

Contact Information

Point of Contact

Bjorn Andersson, Ph.D.

Principal Researcher

Telephone: +1 412.268.9243

Email: info@sei.cmu.edu

Thanks!