

2IMN20 - Real-Time Systems

Multiprocessor systems



Lecturer: Dr. Mitra Nasri

Assistant professor

IRIS Cluster

m.nasri@tue.nl

Sources

Sources of some of the slides used in this lecture:



Geoffrey Nelissen (TU Eindhoven)



Giorgio Buttazzo (Scuola Superior Sant'Anna, Pisa):
<http://retis.sssup.it/~giorgio/CBSD.html>

Thanks for sharing.

Overview of the lecture

- **Why multiprocessors?**
- **Implications for real-time systems (on WCET)**
 - Lack of timing predictability due to cache, bus, and DRAM memory accesses
- **Modeling computation**
- **Modeling platforms (type of multicore platforms)**
- **Multicore real-time **scheduling** and **schedulability analysis****
 - Partitioned and global scheduling
 - Utilization bounds



WCET = worst-case execution time

Why multiprocessors? More computational capabilities



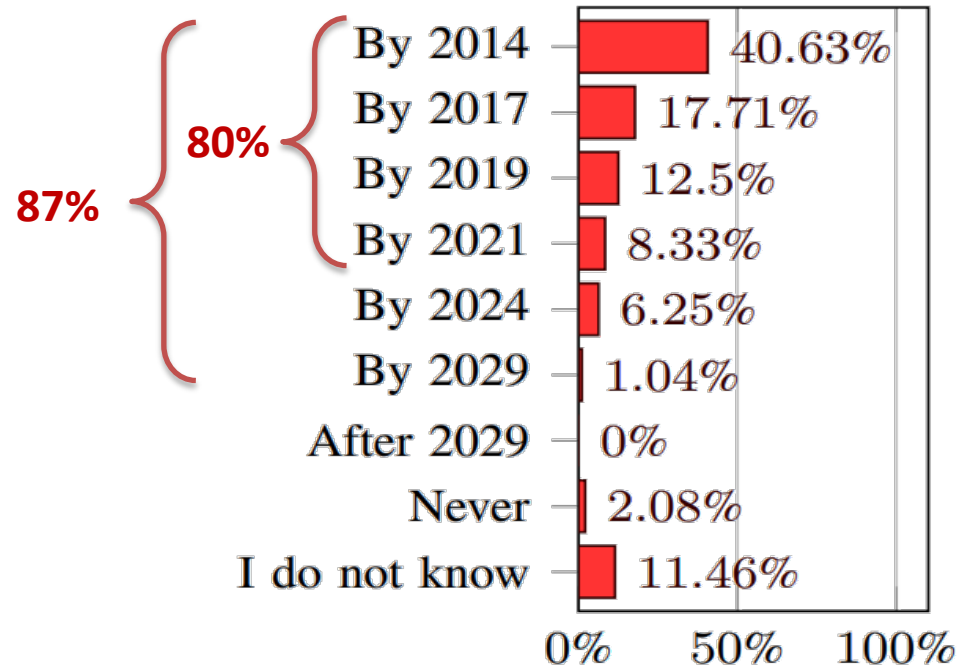
What **percentage** of real-time systems (being) developed by industry use **multicore platforms**?

A: 20%

B: 40%

C: 60%

D: 80%



Our empirical study in 2020 on 120 industry practitioners:

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "A Comprehensive Survey of Industry Practice in Real-Time Systems," Real-Time Systems Journal (RTS), Springer, 2021. [[paper](#) | [data](#) | [companion page](#) | [presentation video](#) (25 min)]

What proportion of real-time systems will **never stop using** single core platforms?

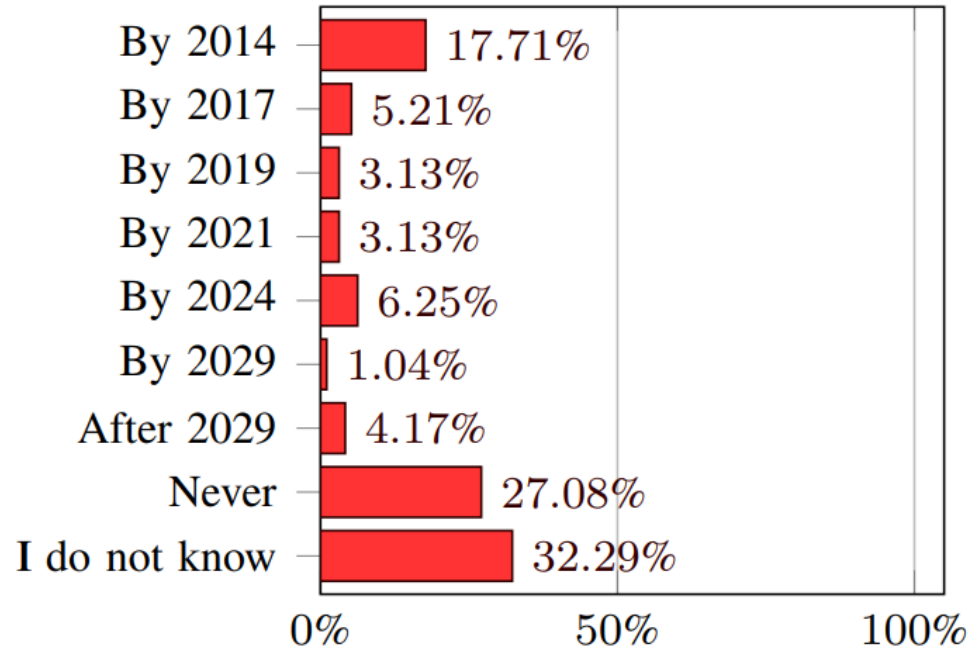
A: 0%

B: 7%

C: 17%

D: 27%

E: 37%



Our empirical study in 2020 on 120 industry practitioners:

Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis, "A Comprehensive Survey of Industry Practice in Real-Time Systems," Real-Time Systems Journal (RTS), Springer, 2021. [[paper](#) | [data](#) | [companion page](#) | [presentation video](#) (25 min)]

To get a good picture of your future job market, carefully go through this survey paper



Benny Akesson



Mitra Nasri



Geoffrey Nelissen



Sebastian Altmeyer



Rob Davis



Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, Robert I. Davis,
"A Comprehensive Survey of Industry Practice in Real-Time Systems,"
Real-Time Systems Journal (RTS), Springer, 2021.

The paper (open access):

<https://link.springer.com/article/10.1007/s11241-021-09376-1>

Data:

<https://uvaauas.figshare.com/articles/dataset/DataSetReal-TimeSystemsSurvey/13117709>

A presentation video (25 min):

https://www.akesson.nl/files/videos/akesson20-rtss_video.mp4

Video of our interviews with 4 panelists (Rolls-Royce, Volkswagen, Microsoft, TNO):

https://www.akesson.nl/files/videos/akesson20-rtss_panel.mp4

The weblog (contains videos that explain the results and trends and other stuff):

<https://akesson.nl/2020/12/18/an-empirical-survey-based-study-into-industry-practice-in-real-time-systems-rtss-2020/>

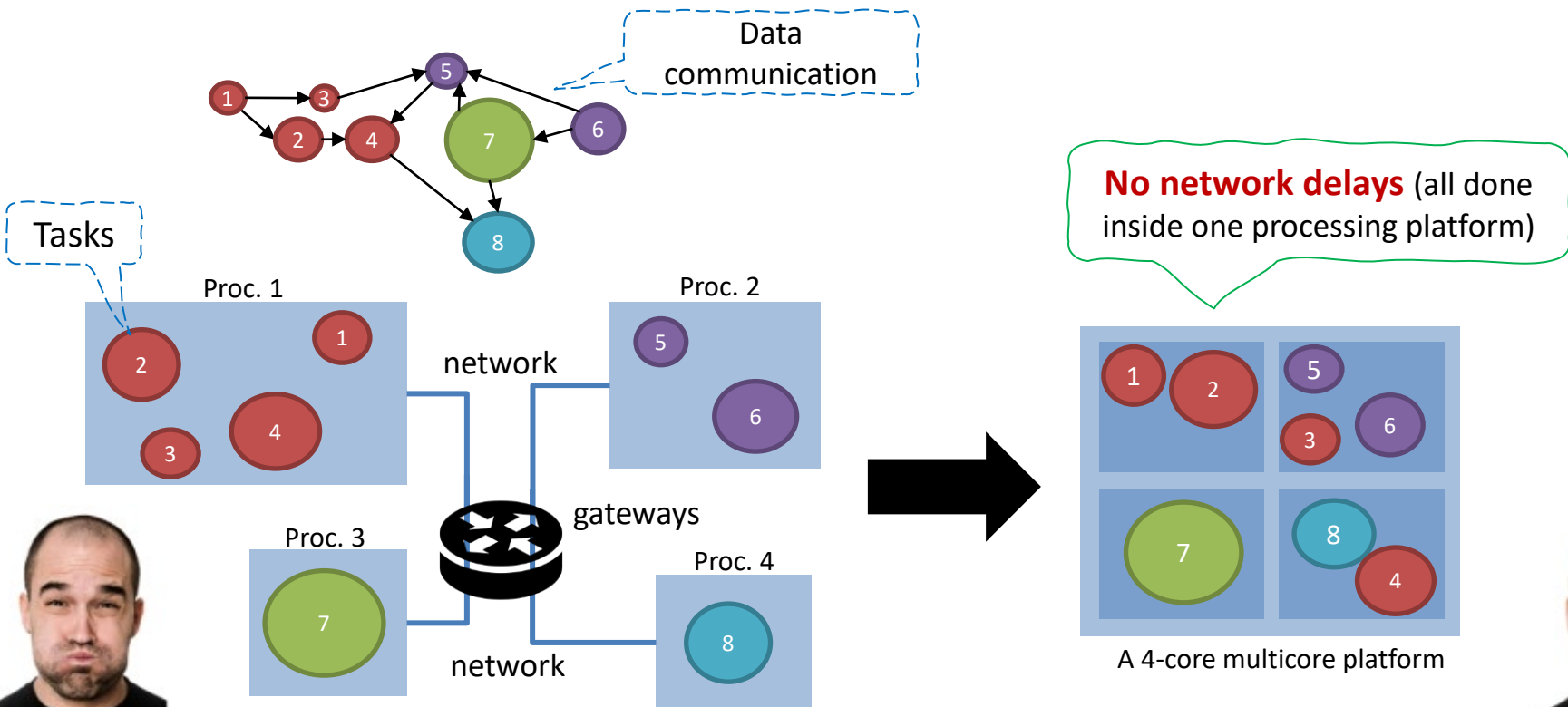
How did traditional real-time systems adopt multicore platforms?

- By **consolidating** old applications into one platform
- By developing new applications that can **parallelize** the computation among the cores

Switching to multicore systems:

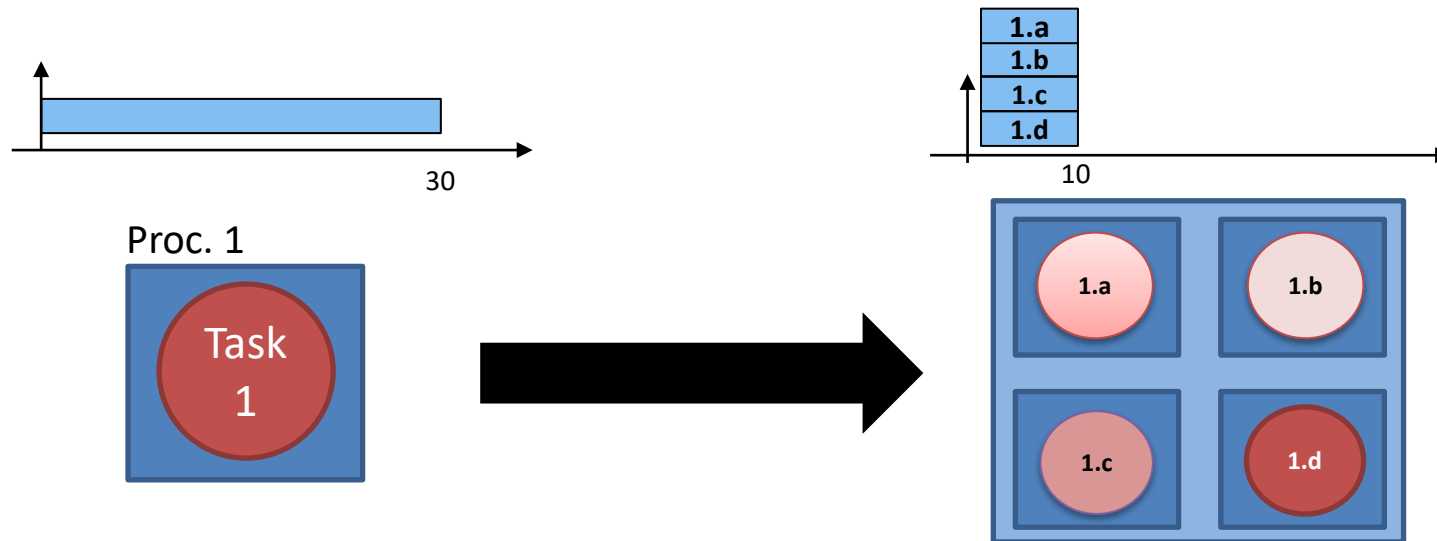
Consolidation

- **Integrate functionalities of multiple processors** into a single multicore platform
 - Example from automotive industry: ECU consolidation
 - It reduces the hardware cost, wiring, and communication delays



Switching to multicore systems: parallel programming

- By **parallelizing the application code**, each code segment can run on a different core in parallel, hence, tasks will have shorter execution times



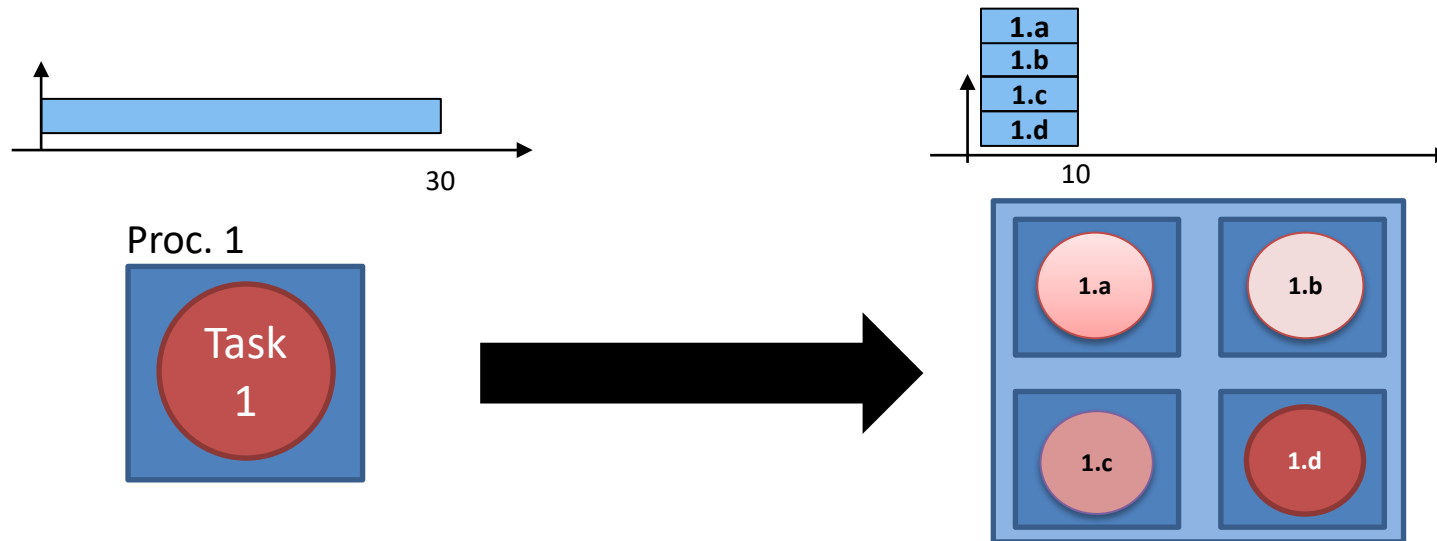
Main challenges:

How to **split the code into parallel segments** that can be executed simultaneously?
What to do with data or control flow dependencies?

How to **allocate segments to different cores**?

Switching to multicore systems: parallel programming

- By **parallelizing the application code**, each code segment can run on a different core in parallel, hence, tasks will have shorter execution times



Parallelizing legacy code implies tremendous costs and efforts due to:

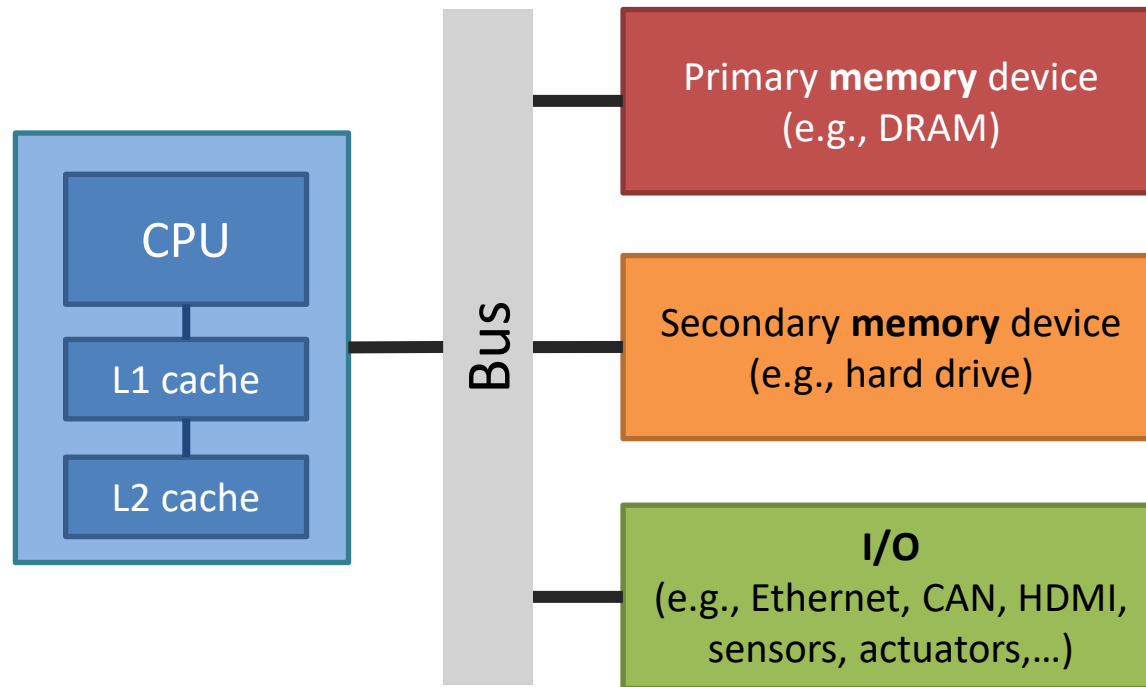
- updating the toolchain
- re-designing the application
- re-writing the source code

- writing new documentation
- re-testing the system
- re-certifying the system

Multicore platforms

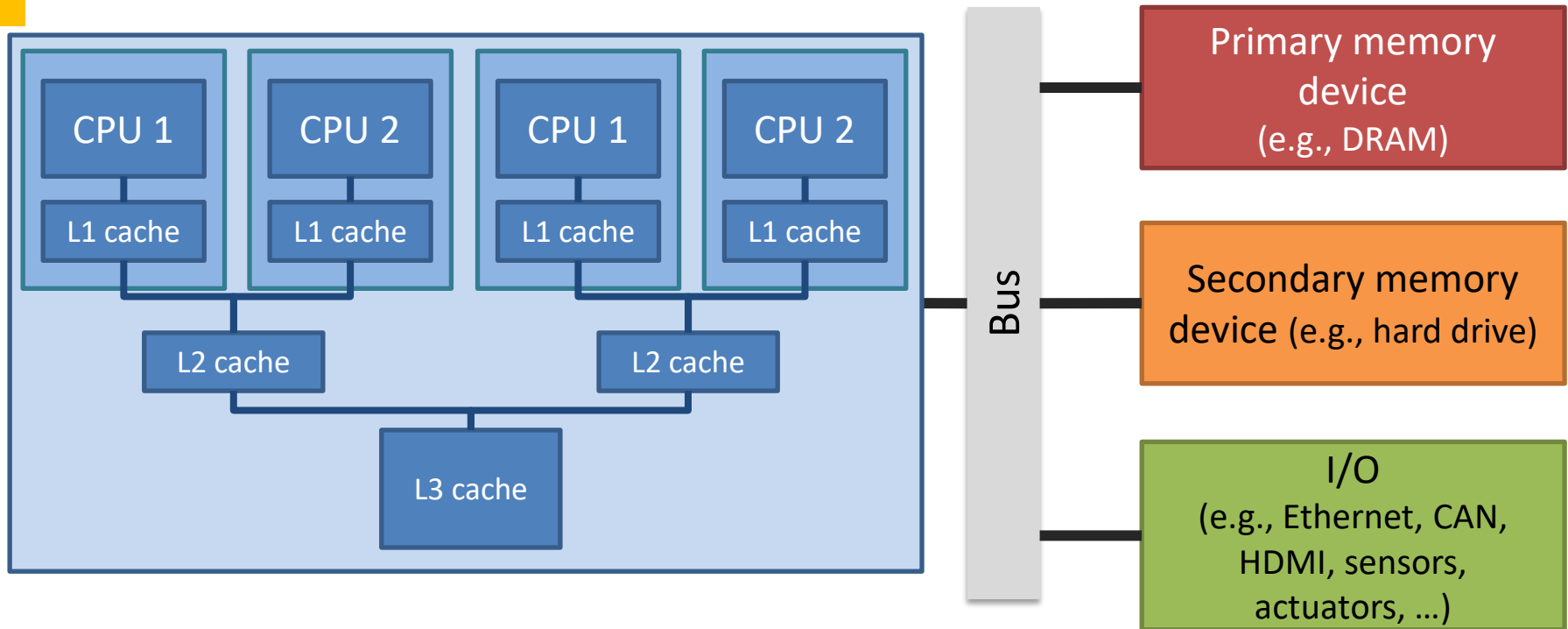
(implications for real-time systems)

Switching to multicore: implications



In **single core** systems, tasks are sequentially executed
➔ **access to shared hardware resources is serialized**

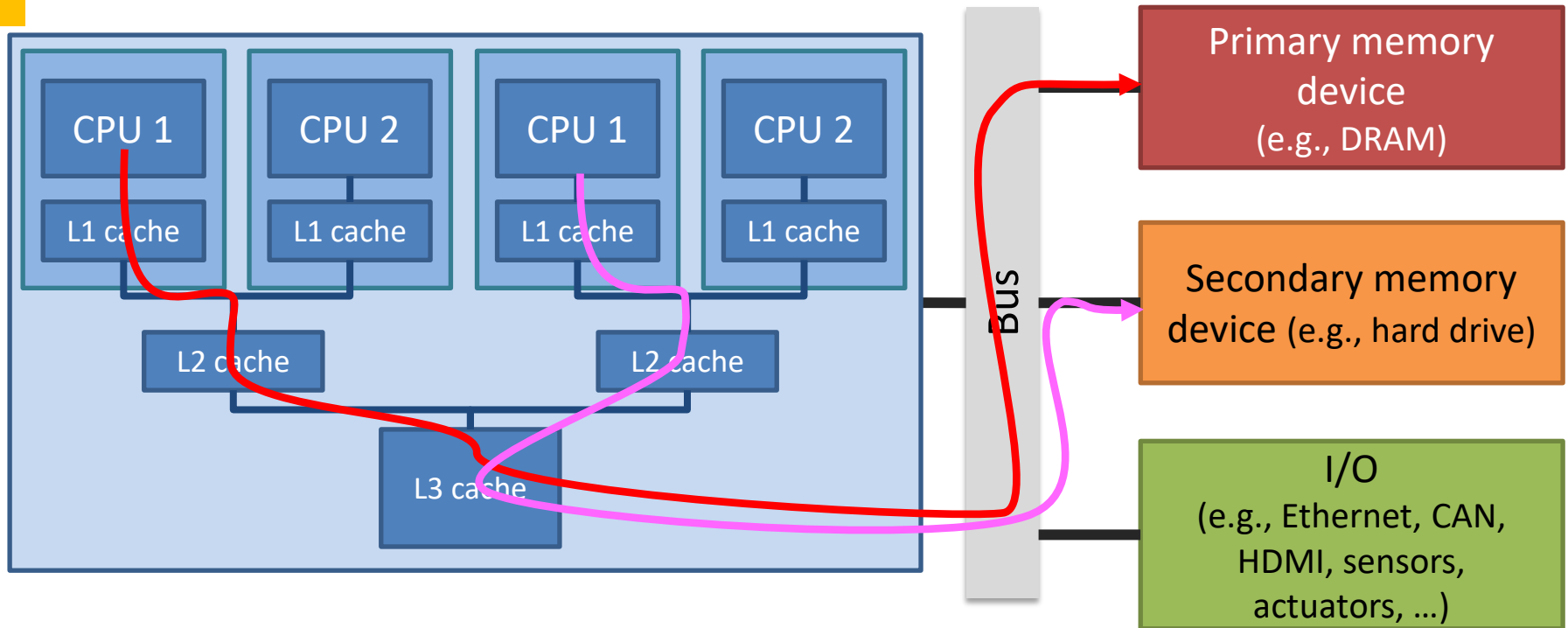
Switching to multicore: implications



In **multicore core** systems tasks may execute in parallel

➔ **Tasks compete for shared hardware resources**

Switching to multicore: implications

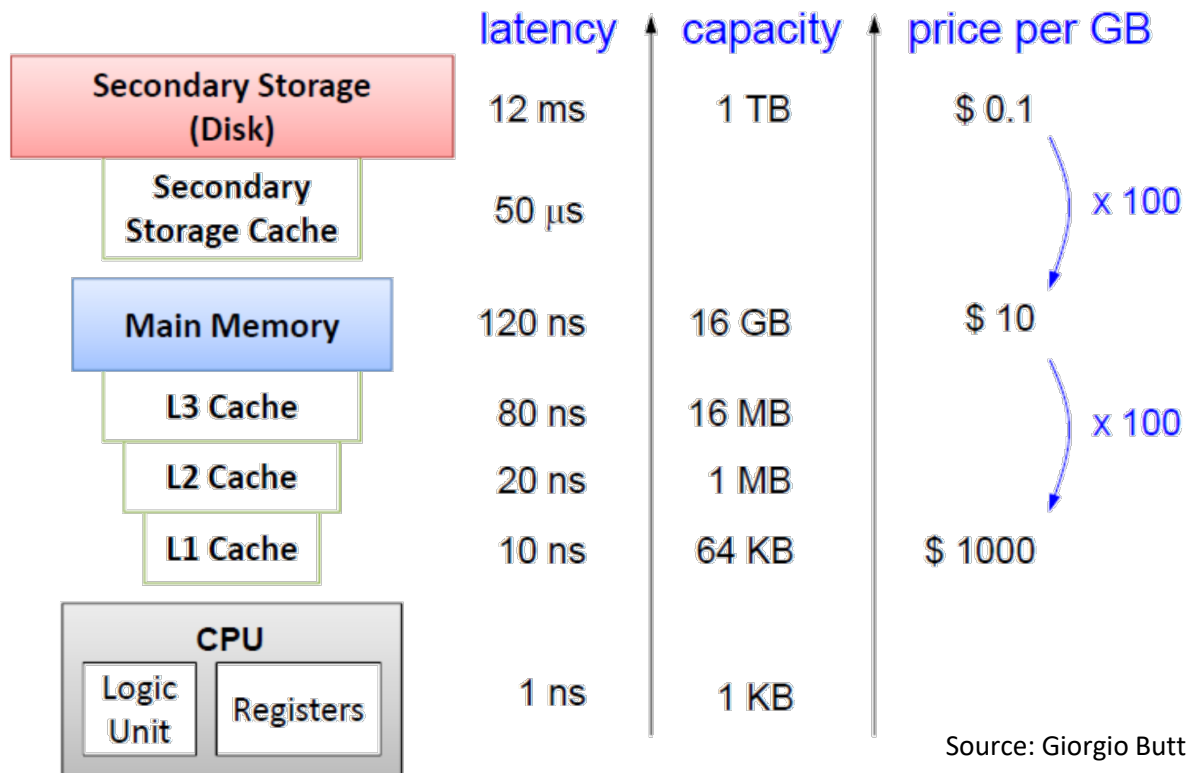


In **multicore core** systems tasks may execute in parallel

➔ **Tasks compete for shared hardware resources**

WCET in multicore varies a lot

The impact of cache memory on execution time variations

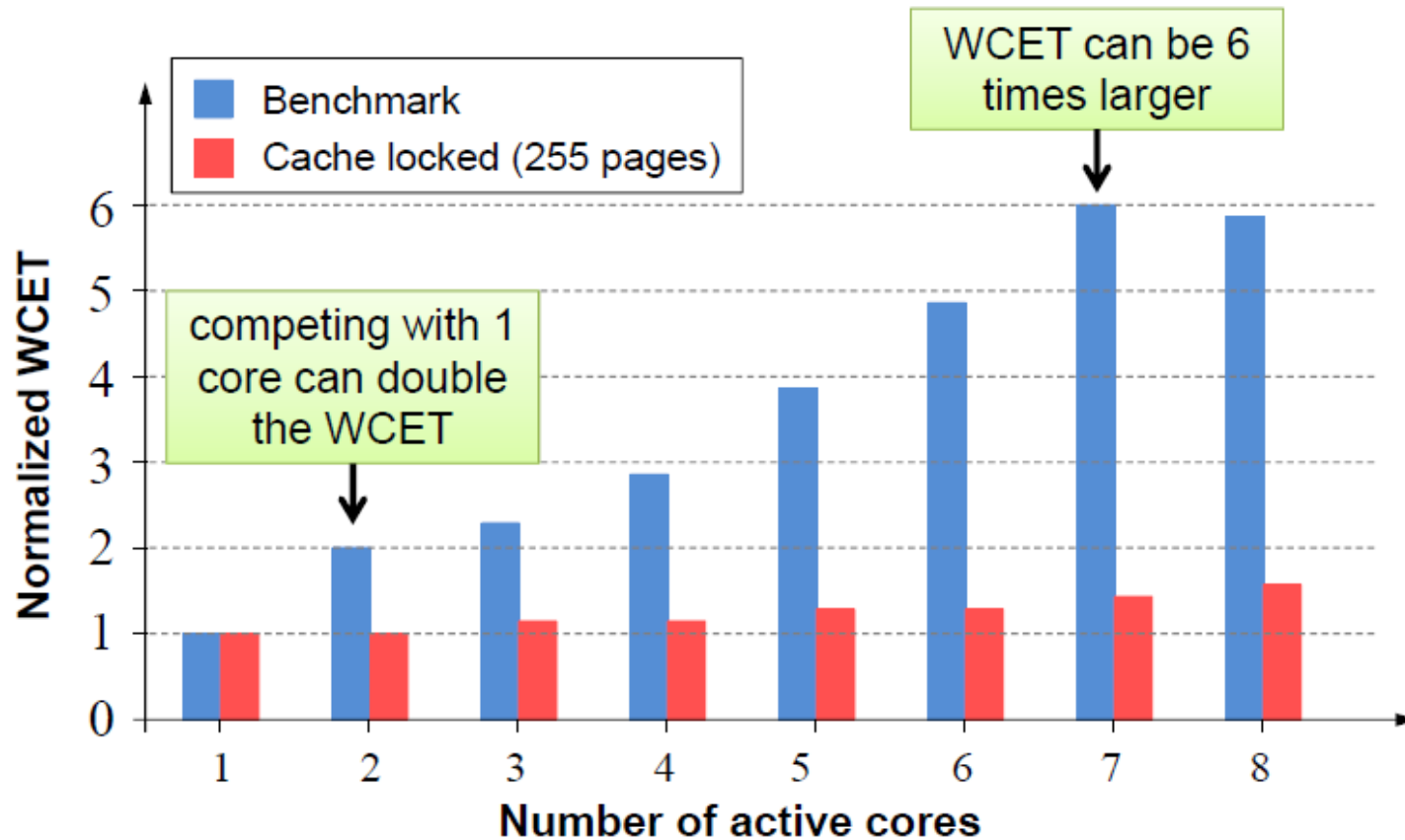


- To **accelerate execution**, the processors **load** *'frequently used instructions and data'* **closer to the CPU**
- Since **capacity is limited**, different **tasks compete** for the same memory space

WCET in multicore varies a lot

The impact of cache memory on execution time variations

Test by Lockheed Martin Space Systems on an 8-cores platform

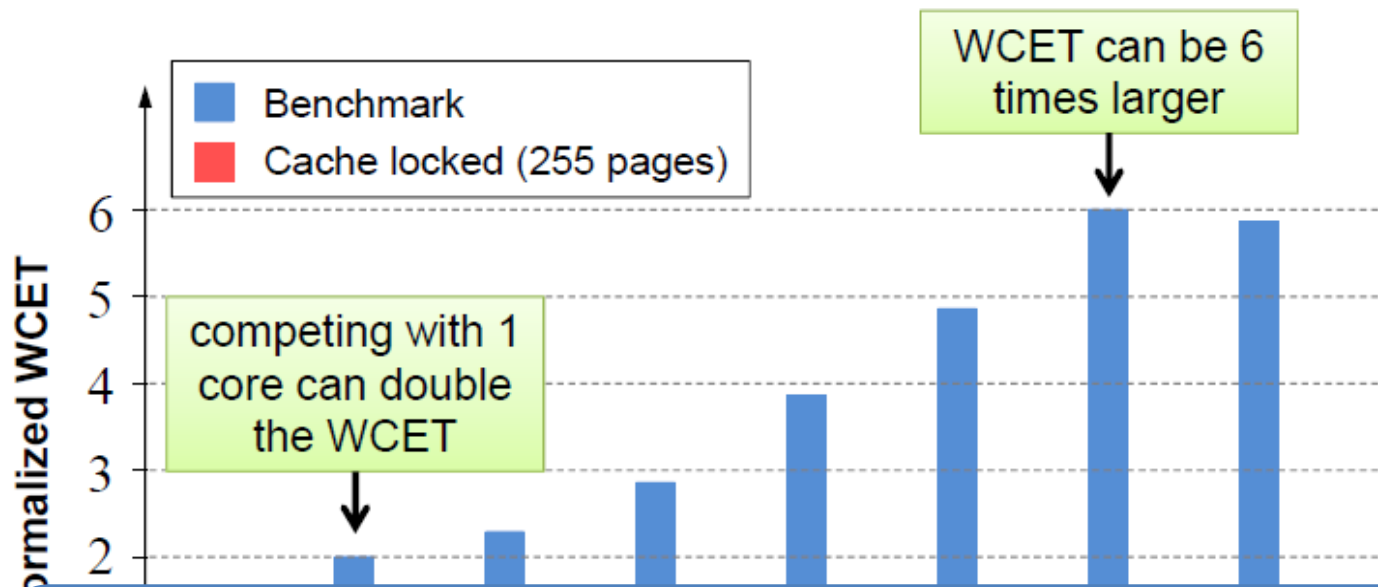


Source: Giorgio Buttazzo

WCET in multicore varies a lot

The impact of cache memory on execution time variations

Test by Lockheed Martin Space Systems on a 8-cores platform

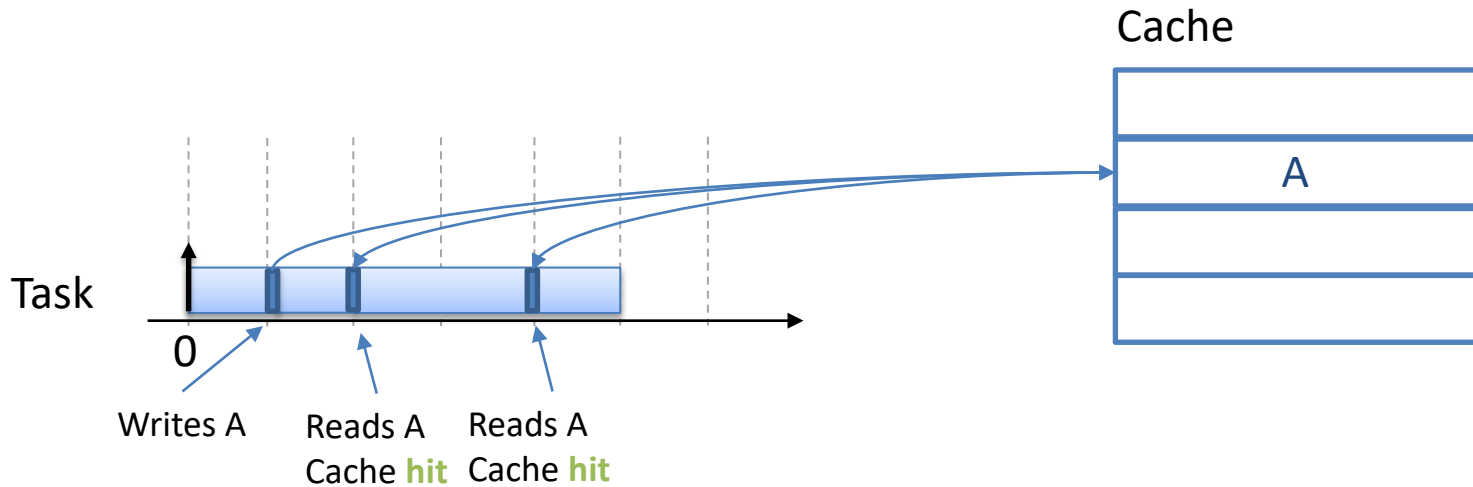


The **main assumption of real-time systems** that the **WCET is known**, does not hold anymore for multicore systems

WCET in multicore varies a lot

The impact of cache memory on execution time variations

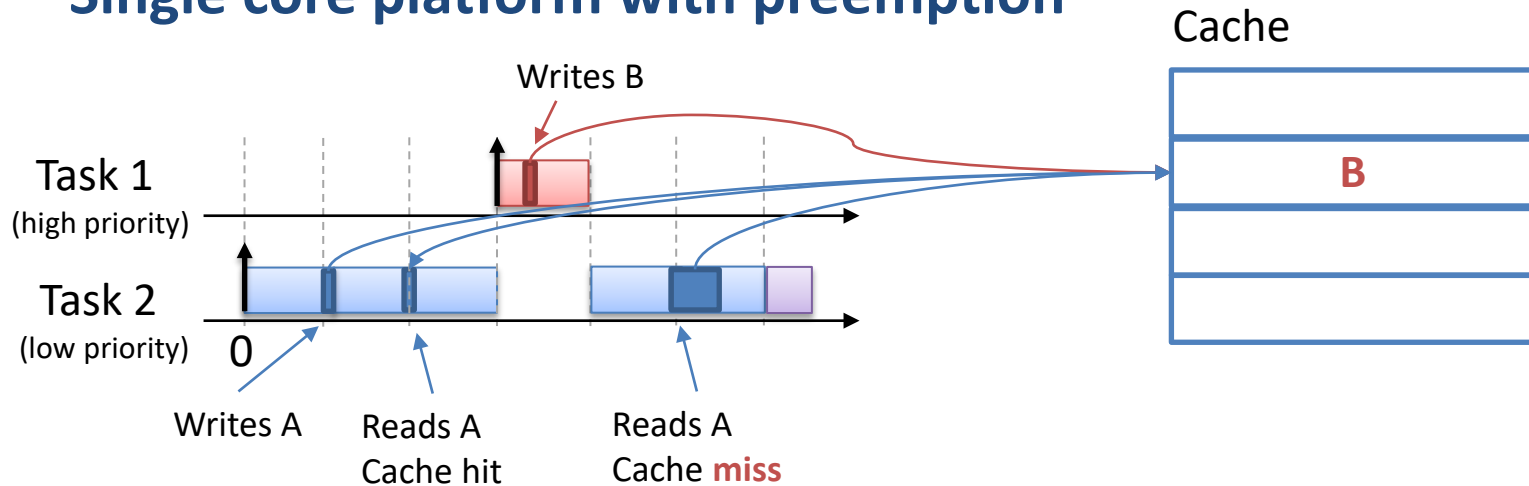
Single core platform (execution in isolation)



WCET in multicore varies a lot

The impact of cache memory on execution time variations

Single core platform with preemption

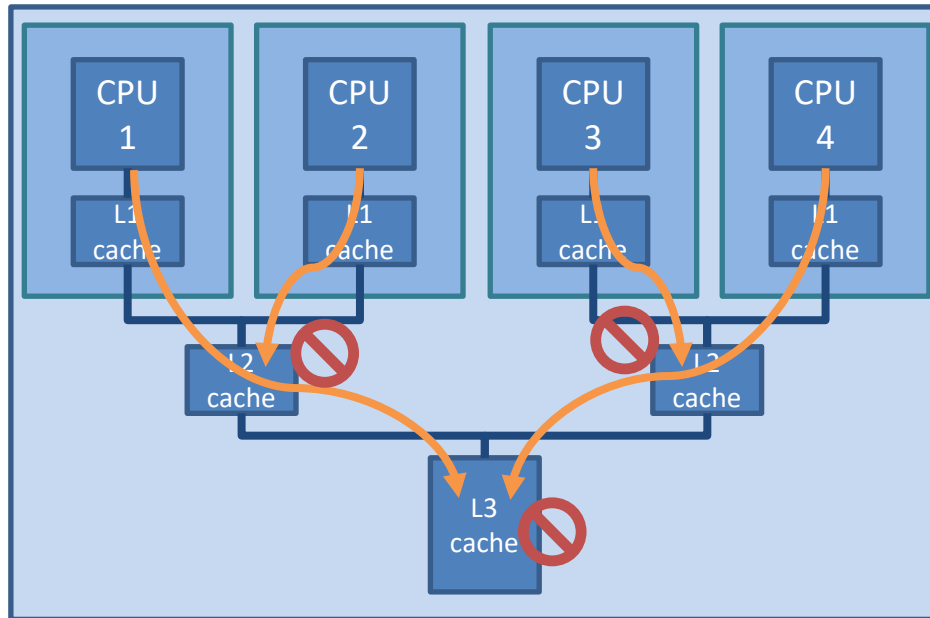


In **single core** systems **cache related preemption delays (CRPD)** increase the WCET of each task

WCET in multicore varies a lot!

Impact of cache memory on multicore platforms

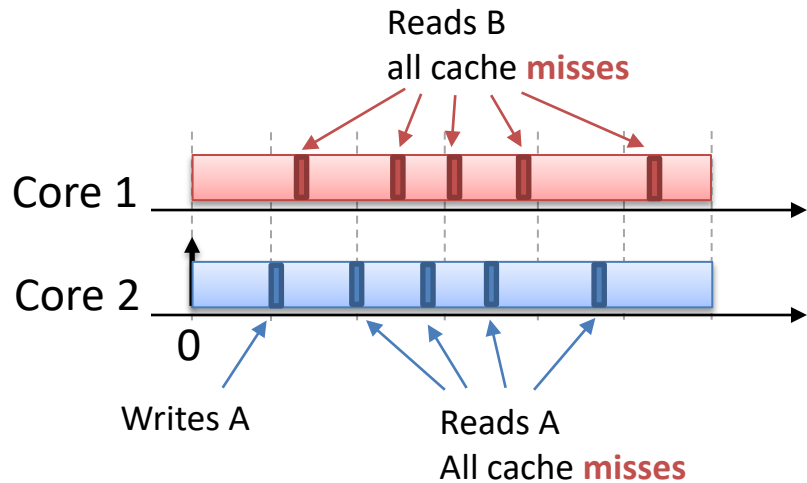
Multicore platform



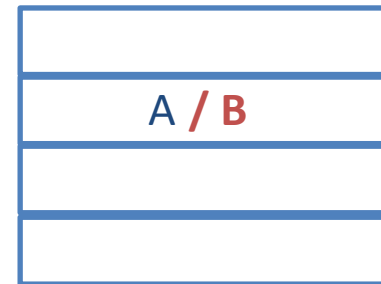
Cores compete to access/use shared resource (shared caches here)

WCET in multicore varies a lot!

Impact of cache memory on multicore platforms



Share L2 Cache

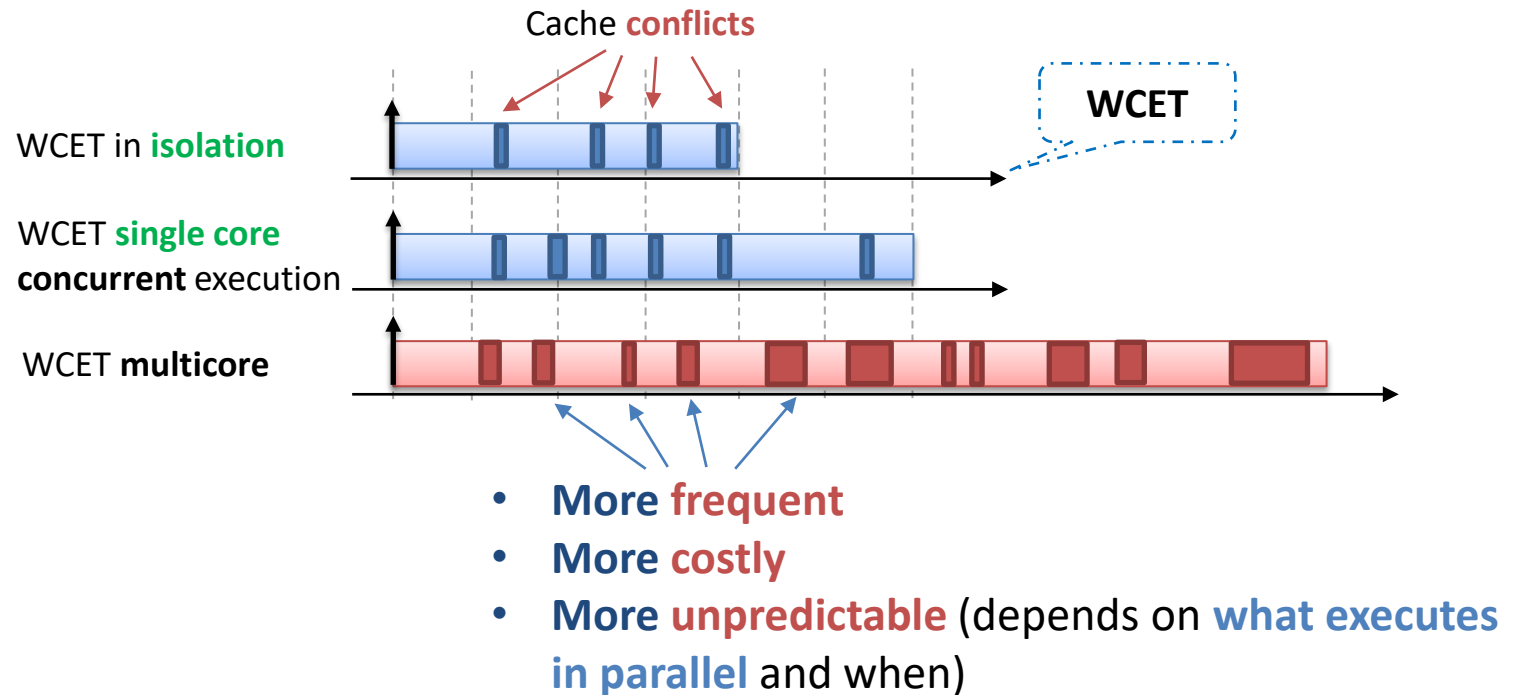


In **multicore** systems tasks executing in parallel can **trash** each others **cache content**

→ Can cause significant WCET increase

WCET in multicore varies a lot!

Impact of cache memory on multicore platforms



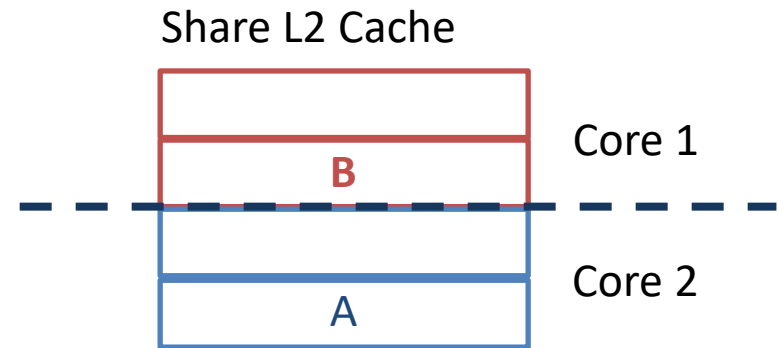
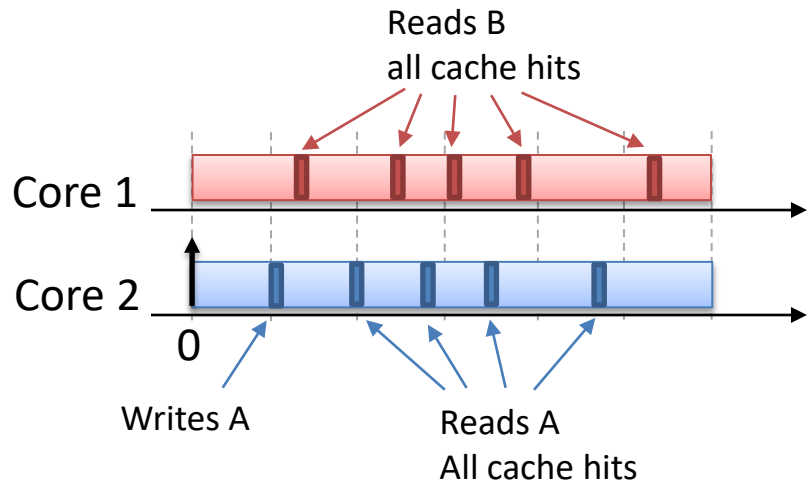
In **multicore** systems tasks executing in parallel can **trash** each others **cache content**

➔ **Can cause significant WCET increase**

WCET in multicore varies a lot!

Impact of cache memory on multicore platforms

A solution: Partition caches between cores



Reduces conflicts → **less variability** on the WCET

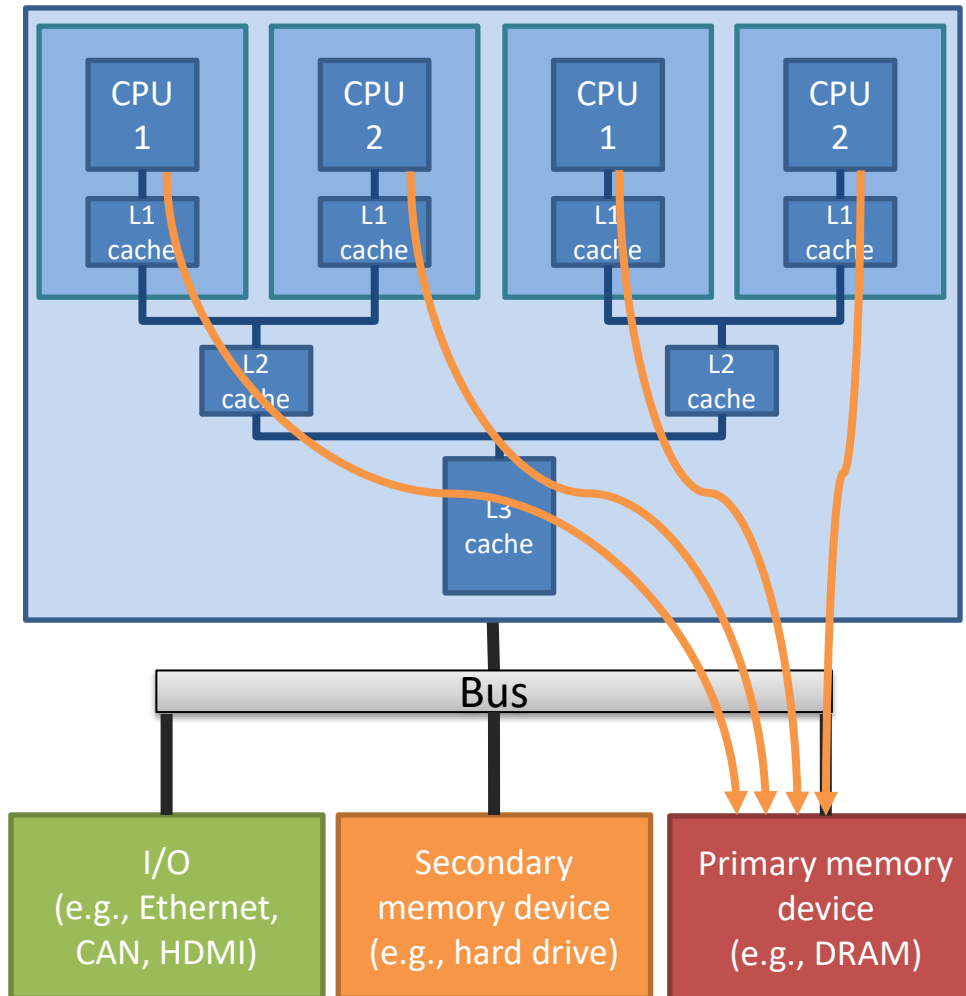
But **reduced cache size** per core → **increased WCET**

Other considerations?

Interference on memory banks, I/Os, and more

WCET in multicore varies a lot!

Memory bank conflicts

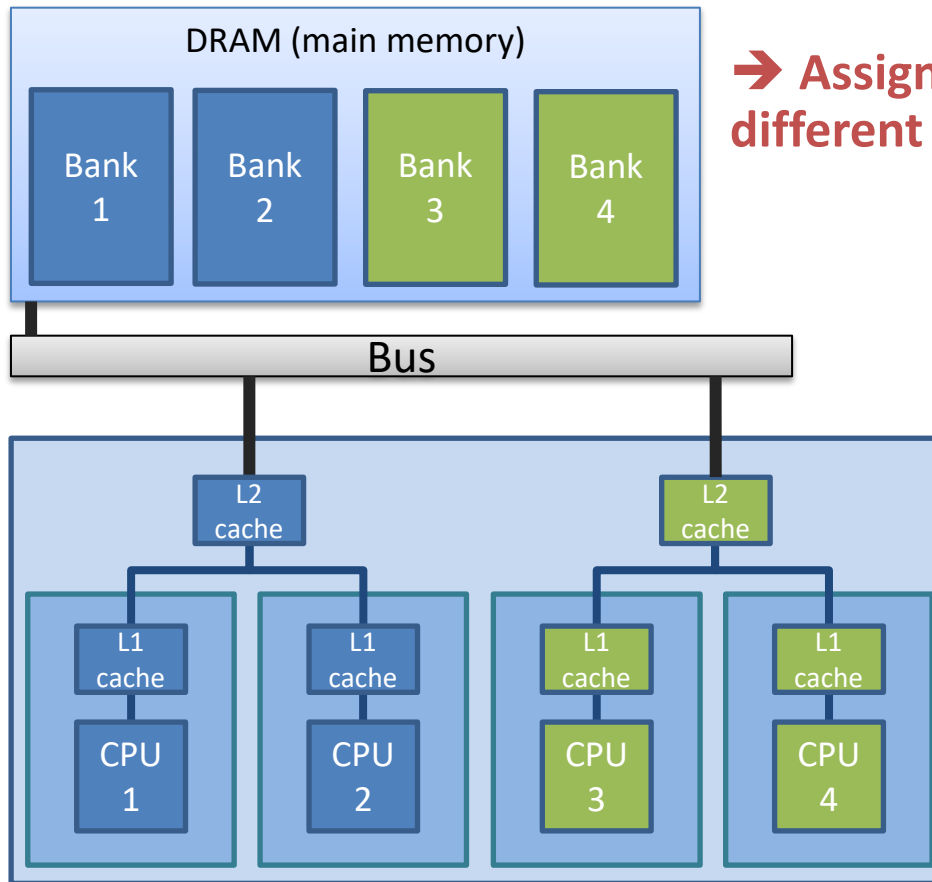


If **all cores** try to **access** the **main memory** at the same time, it may **overload** the **memory controller** and generate long delays for serving memory requests

WCET in multicore varies a lot!

A solution to reduce memory access conflicts

To reduce memory access conflicts, the DRAM is divided into **banks** that **can be accessed in parallel** without blocking

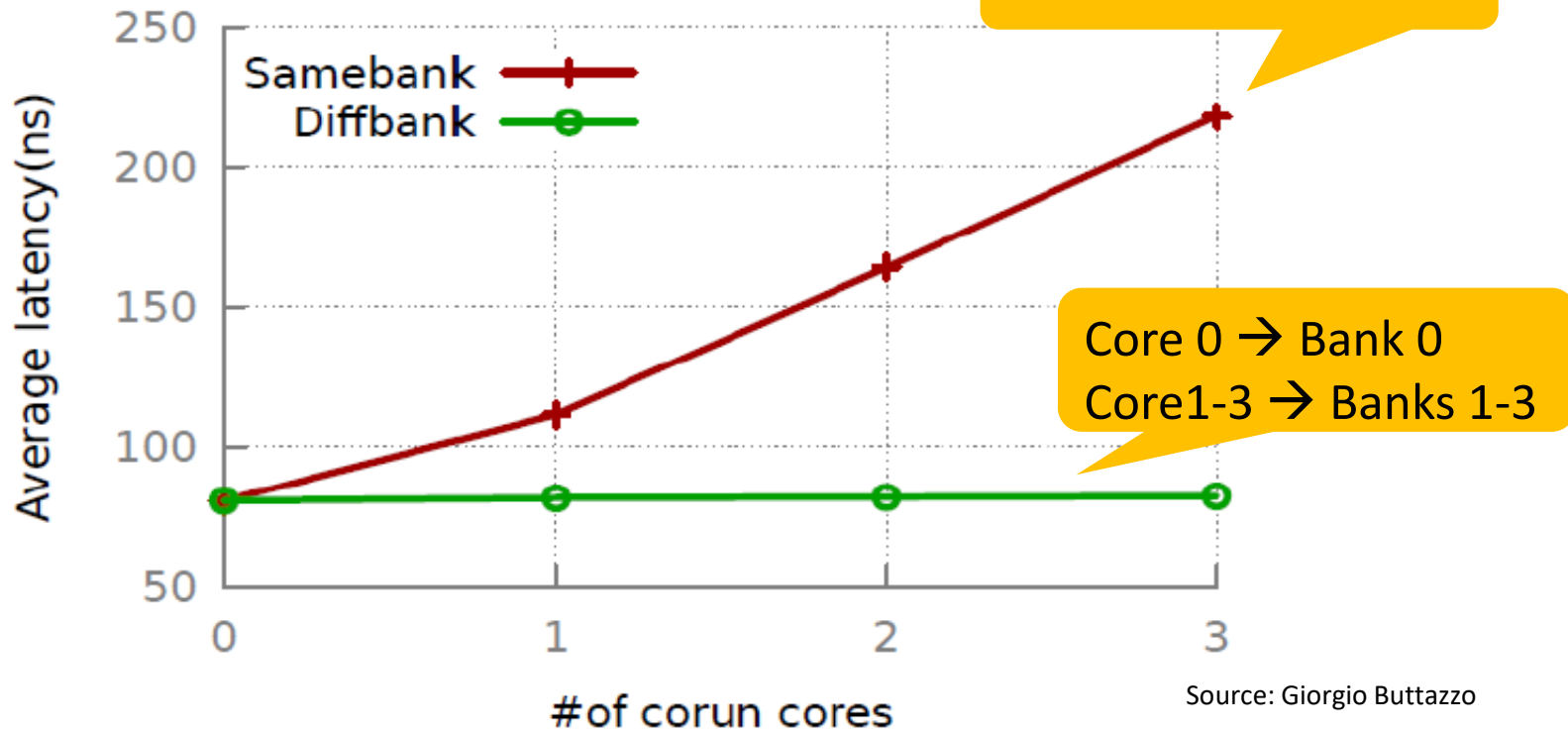


→ Assign different memory banks to different cores to avoid inter-core interference

WCET in multicore varies a lot!

A solution to reduce memory access conflicts

Test on Intel-Xeon



Reduces conflicts on memory accesses → **less variability** on the WCET

But also **reduces intra-task memory access parallelism** → **increased WCET**

WCET in multicore systems

Summarizing

- In **multicore** platforms, the WCET of tasks **varies a lot** due to **contention** to access **shared hardware resources** (e.g., caches, main memory controller, bus arbiter, I/Os)
- It can cause **significant increase in WCET**
- Execution time variations are **non-deterministic**
- There are **ways to improve time determinism** (e.g., cache partitioning, memory bandwidth regulation and DRAM banks partitioning between cores) but at the cost of **worse average-case performances**

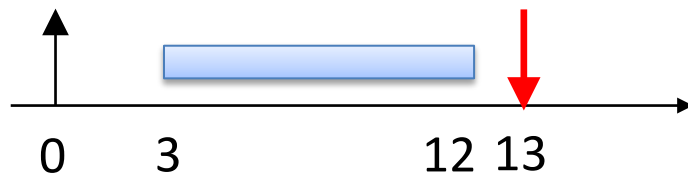
WCET in multicore systems

For simplicity, in the rest of this lecture, we assume that **a sound upper-bound on the WCET exists and is known**

Modeling computation on multiprocessor platforms

Modeling tasks

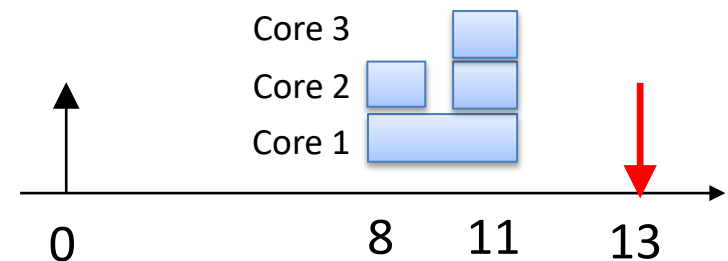
Sequential task



Different tasks may be executed at the same time, but **each task executes on at most one core at a time.**

v.s.

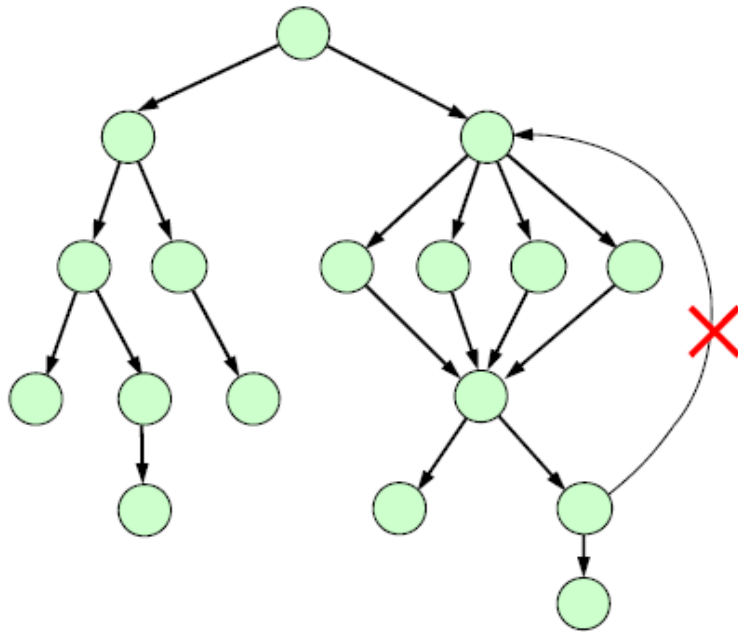
Parallel task



Different execution segments of the **same task may execute in parallel on different cores** at the same time.

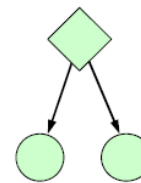
Modeling parallel tasks

Representing a parallel code requires more complex structures like a graph (usually a **directed-acyclic graph**, a.k.a. **DAG**):

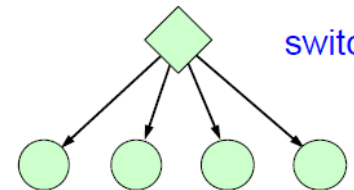


In a DAG this connection is forbidden

DAGs can be **conditional**
(e.g., by an **if-then-else** block)

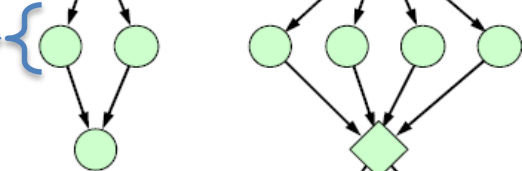


if-then



switch

Both must be executed



Only one of the two execute



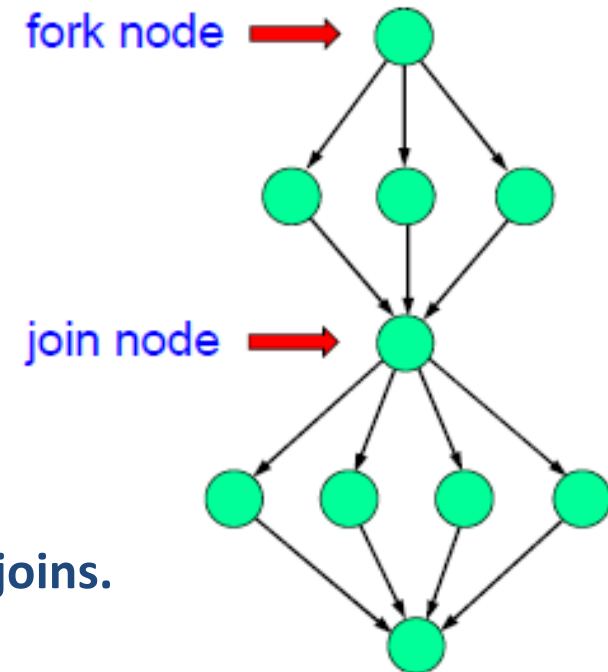
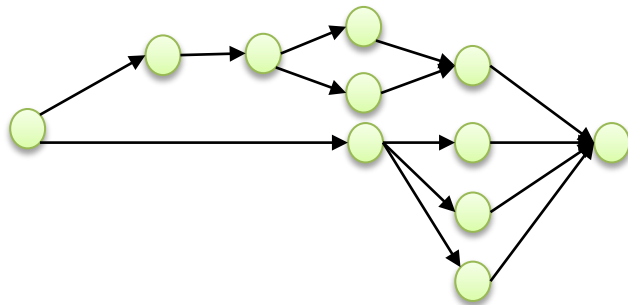
OR nodes represent conditional statements



Computation nodes represent a piece of code that must be sequentially executed

Structured parallelism

- Fork-Join Graphs (a special type of DAGs)
 - After a **fork node**, all **immediate successors must be executed** (the order does not matter).
 - A **join node** is executed **only after** all immediate predecessors are completed.
- Nested fork-join model can have nested forks and joins.

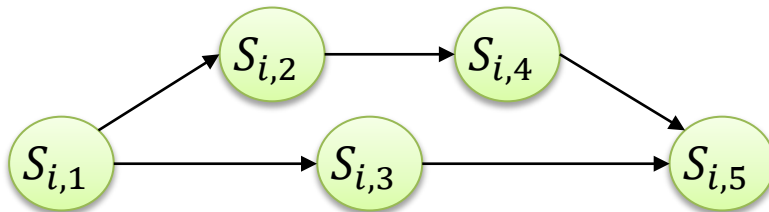


Modeling computation

Precedence constraints

Intra-task

Precedence constraint
between segments of a task

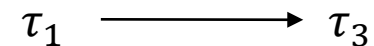


All execution nodes share the **same timing constraints** (e.g., period and end-to-end deadline)

v.s.

Inter-task

Precedence constraint
between tasks



Each task may have to respect **its own timing properties and/or constraints**.
Usually used to model inter-tasks data dependencies.

Modeling computation

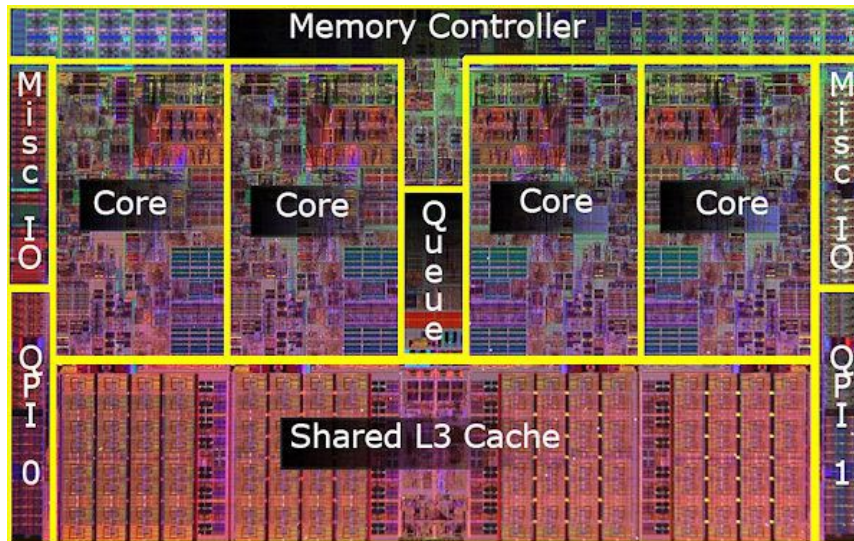
For simplicity, in the rest of this lecture, we assume that
all tasks are sequential
and do **not** have **precedence constraints**

Platform models

Types of multicore systems

Identical cores

Core i7



Source: Intel

All cores are identical

→ Tasks' **execution times** are the same on all cores

Types of multicore systems

Heterogeneous platforms

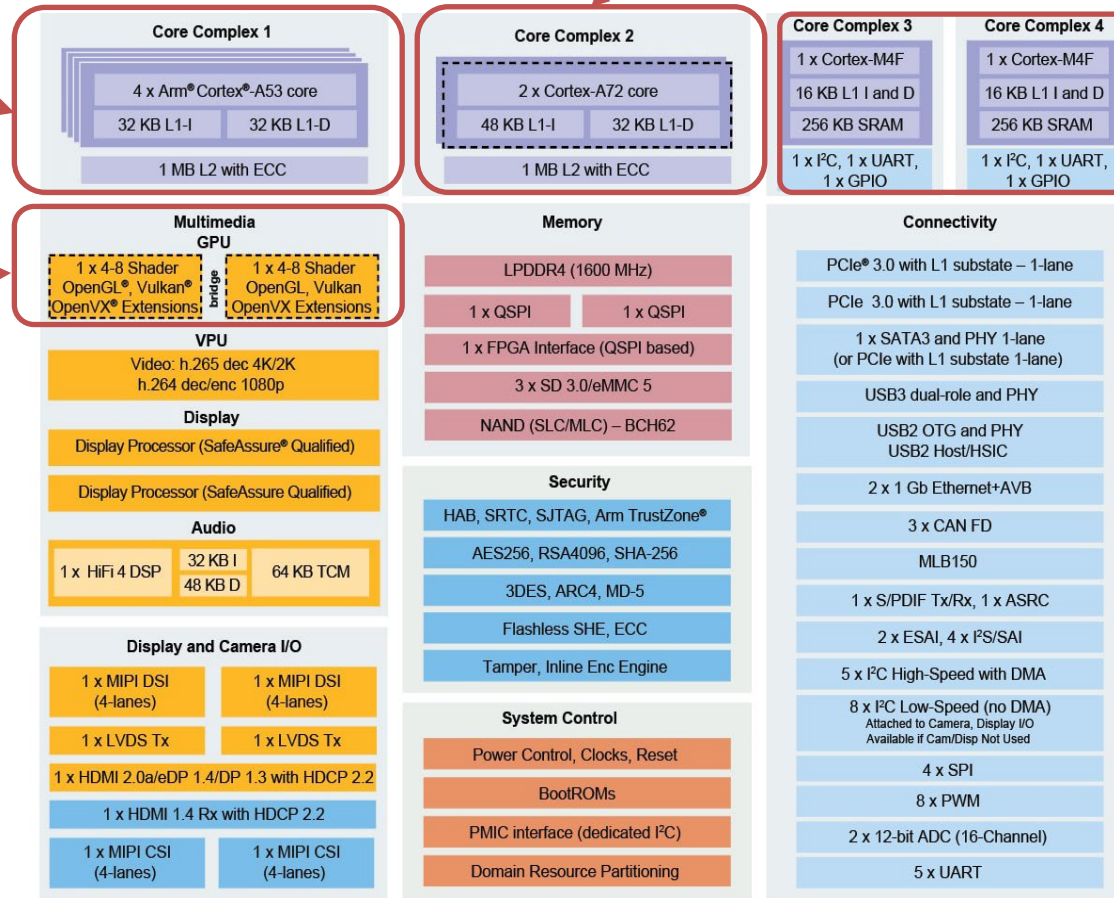
NXP i.MX8

2 big ARM cores

4 small ARM cores

2 GPUs

2 cores for IO management

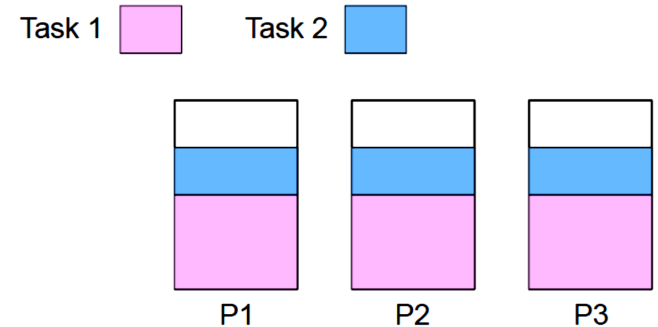


Source: NXP

Types of multicore systems

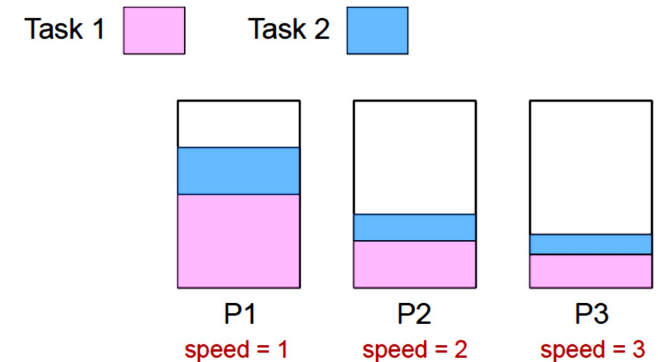
- **Identical**

Processors are of the same type and have the same speed.
Each task has the same WCET on each processor.



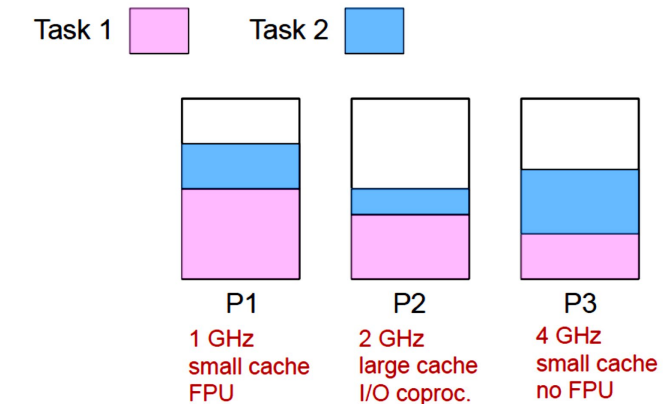
- **Uniform**

Processors are of the same type but may have different speeds. Task WCETs are smaller on faster processors.



- **Unrelated**

Processors can be of different types. The WCET of a task depends on the processor type and the task itself.

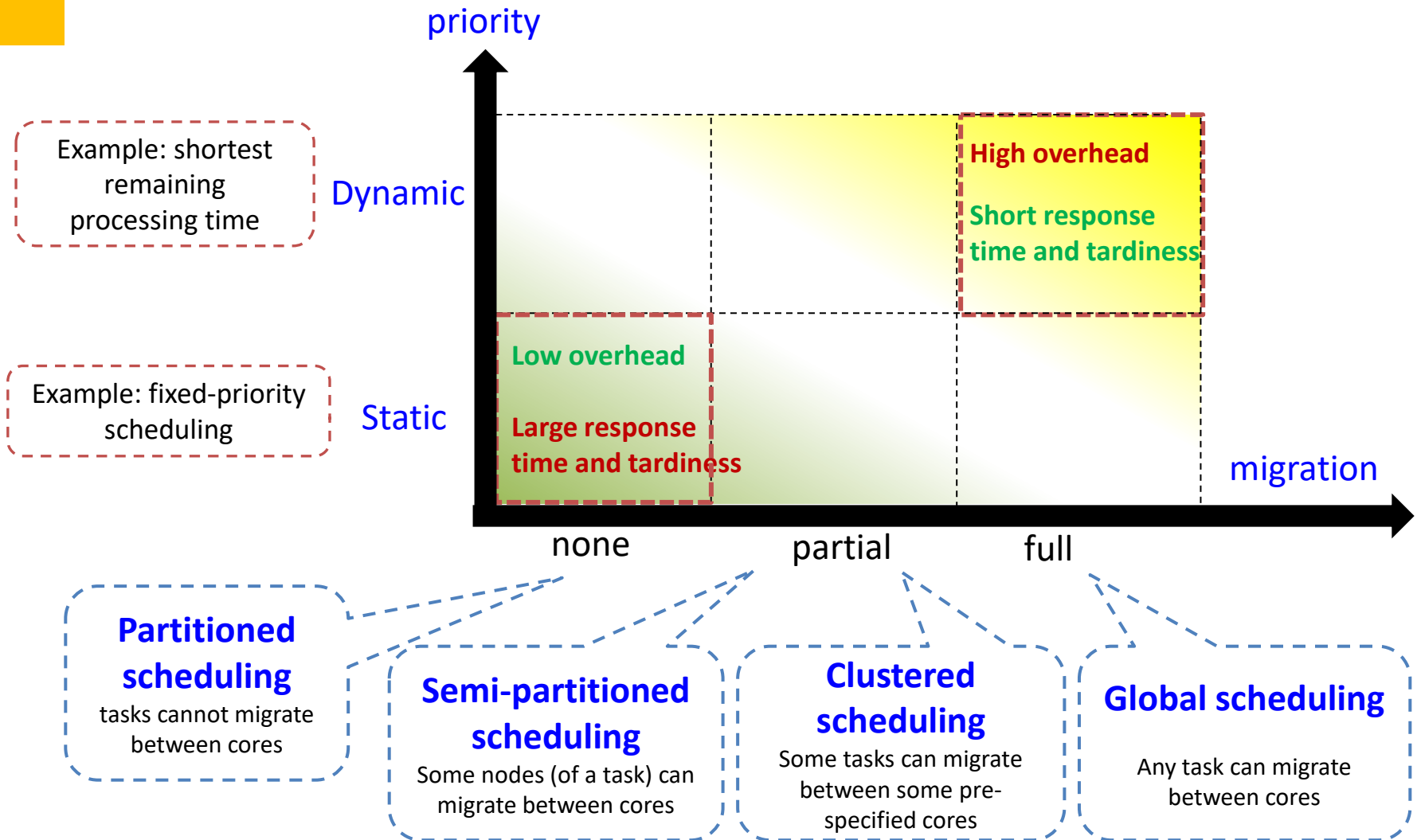


Types of multicore systems

For simplicity, in the rest of this lecture, we assume that
all cores are identical

Multicore real-time scheduling schemes

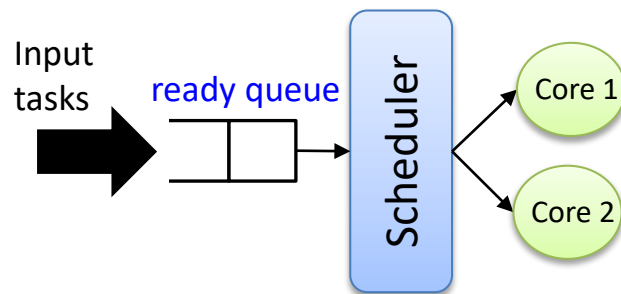
Classification of multiprocessor scheduling algorithms



Global scheduling

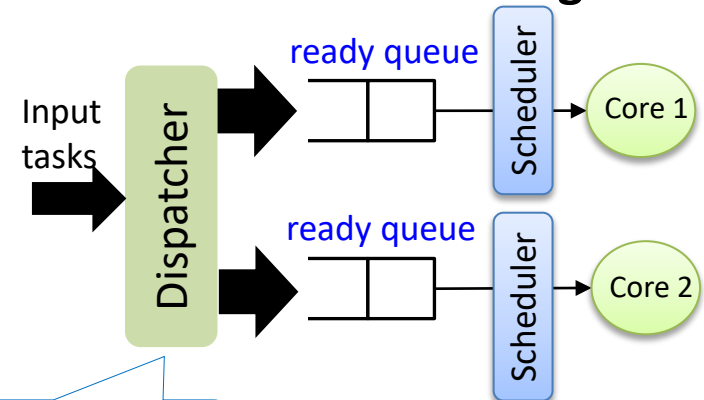
- The system manages a **single queue** of ready tasks
- The processor (to execute the task) is determined at **runtime**
- During execution, a task can **migrate** to another processor

Global scheduling



Use one ready queue for all cores

Partitioned scheduling



Dispatches the task to the right queue according to its predefined core assignment.

Schedules the ready tasks according to a predefined scheduling policy (e.g., EDF, RM, ...)

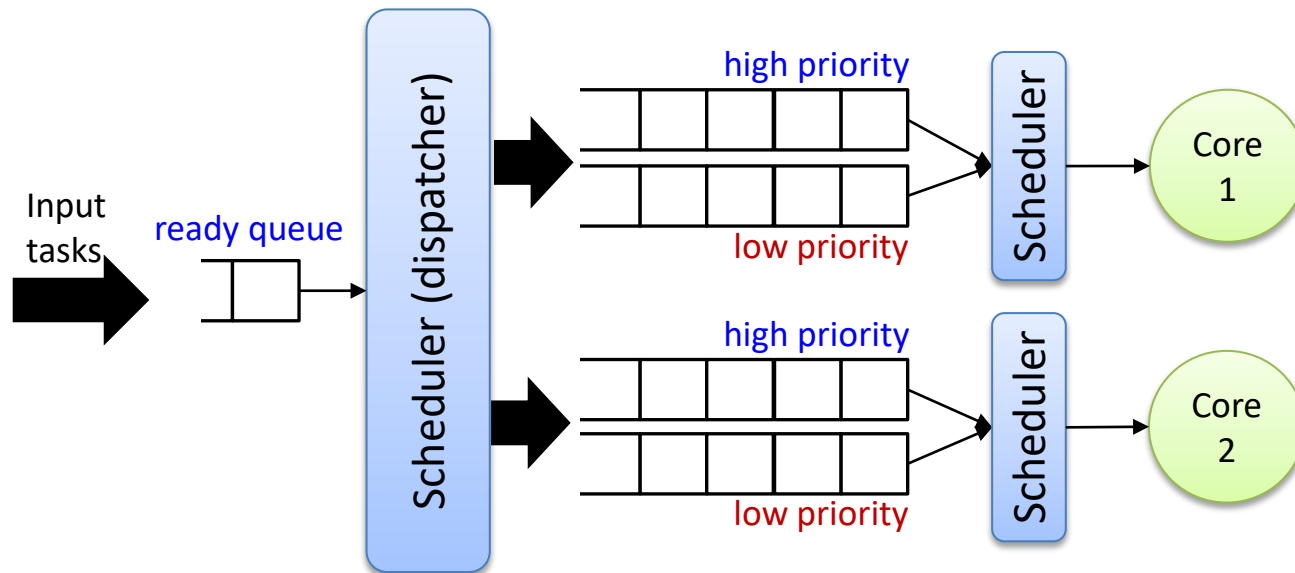
Partitioned scheduling uses separate queues for the cores.

Which approach is better for load balancing?

Which approach has a lower overhead?

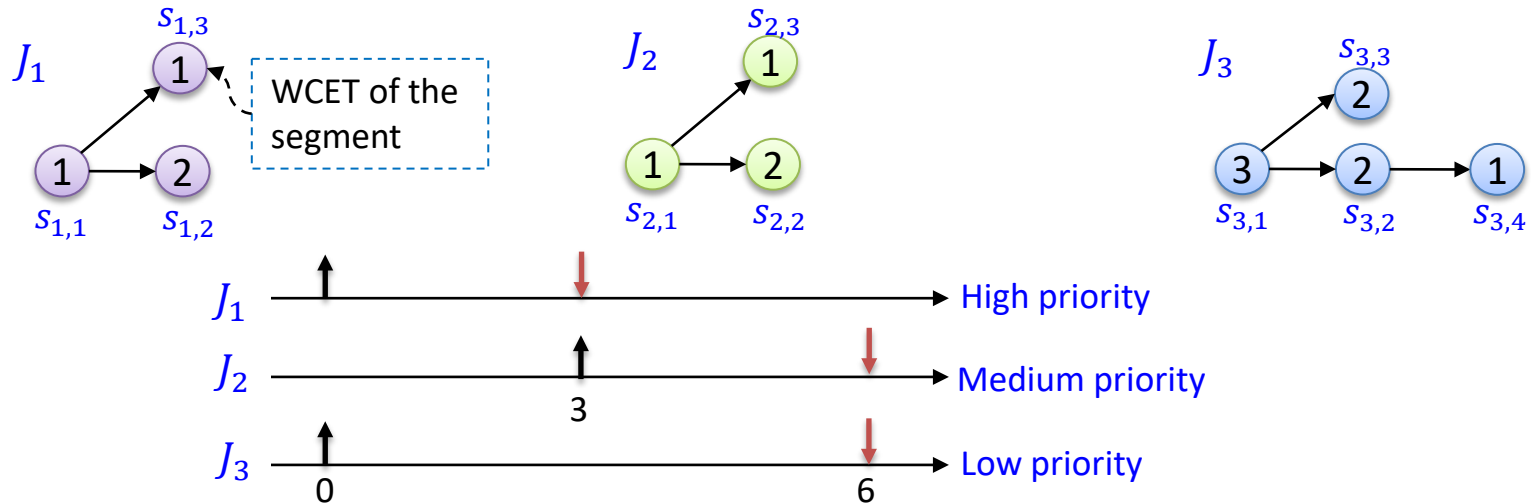
Global scheduling with multi-level queues (hierarchical)

- Use a set of priority queues for each processor
- In each priority queue, tasks are served by FCFS (FIFO)

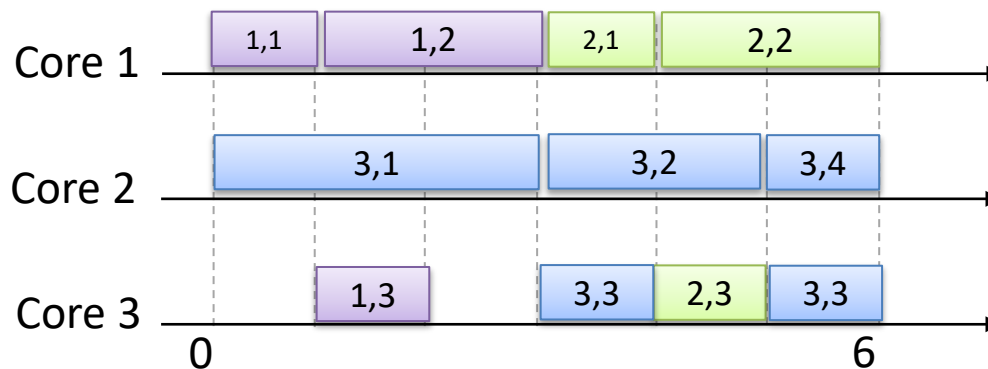


One of the first two-level schedulers was Mesos (Hindman et al., 2011), developed at the University of California (Berkeley), and is now hosted in the Apache Software Foundation. Mesos was a foundation base for other Cluster systems such as Twitter's Aurora (Aurora, 2018) and Marathon (Mesosphere, 2018).

Example: global scheduling of DAGs

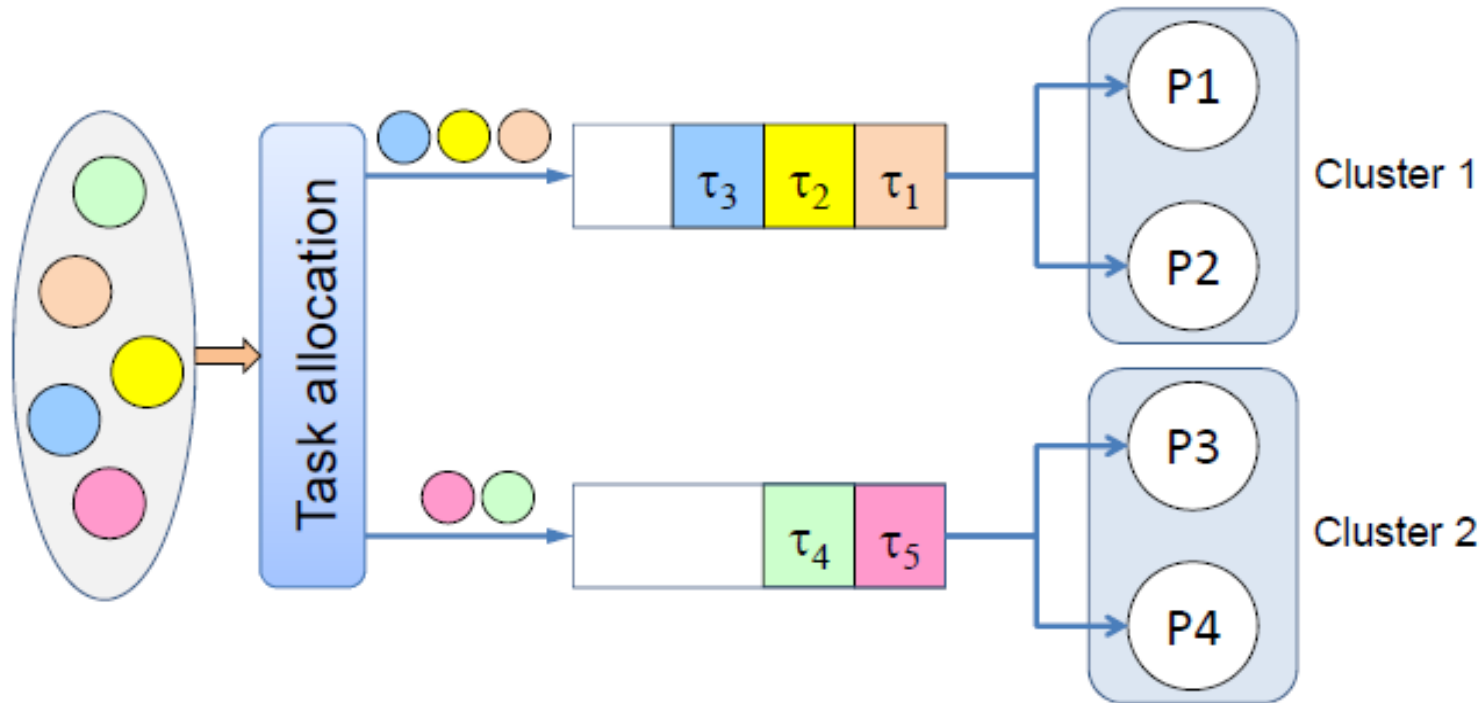


Schedule these jobs using a global single-queue preemptive fixed-priority scheduling algorithm on 3 identical cores:



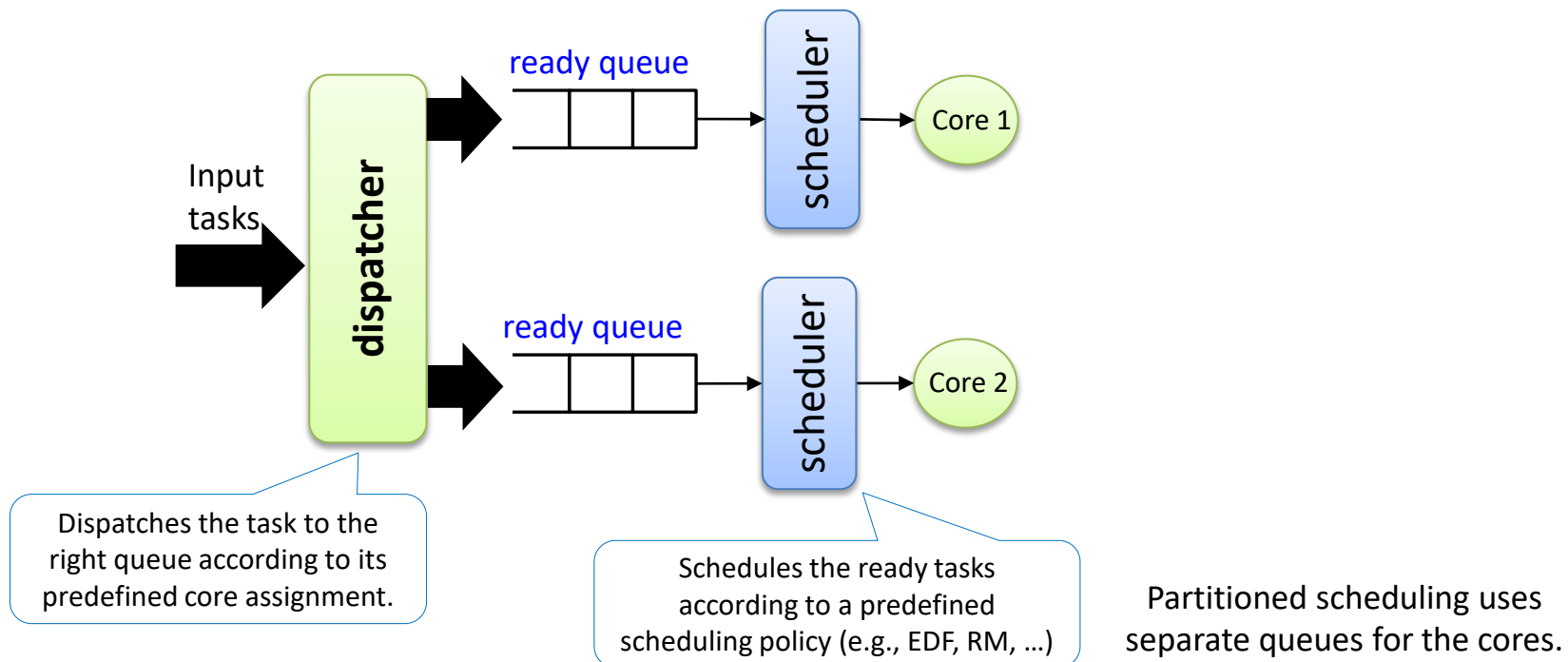
Clustered scheduling

- A task can only migrate within a predefined subset of processors (cluster).



Partitioned scheduling

- Each processor manages its **own ready queue**
- The processor for each task type is **determined offline**
- The **processor cannot be changed at runtime**



Partitioned Scheduling

The scheduling problem reduces to:

**Bin-packing
problem**

NP-hard in the
strong sense



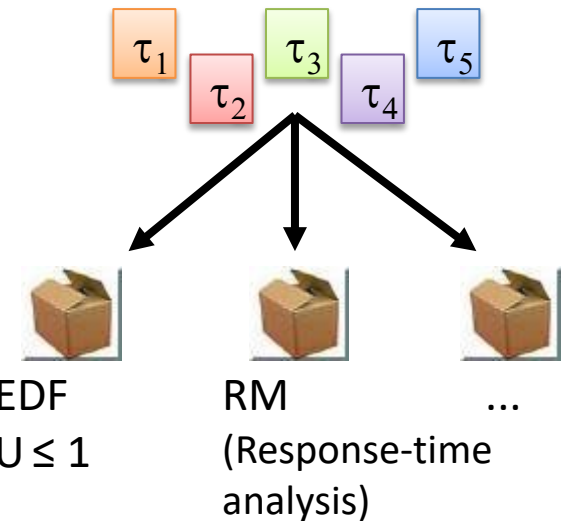
Various heuristics used:

FirstFit, NextFit, BestFit, FFDU, BFDD, etc.

+

**Uniprocessor
scheduling
problem**

Well understood



Possible partitioning choices

1. Partition by information-sharing requirements
 2. Partition by functionality
 3. Use the least possible number of processors or run at the lowest possible frequency
 - Depends on considerations like fault tolerance, power consumption, temperature, etc.
 4. Partition to increase schedulability
- } **These approaches might not be good for schedulability**

Classic partitioning algorithms

Partitioning problem:

Given a set of tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ and a multiprocessor platform with **m processors**,
find an **assignment** from tasks to processors such that each task is assigned to one and only one processor

It is a bin packing problem

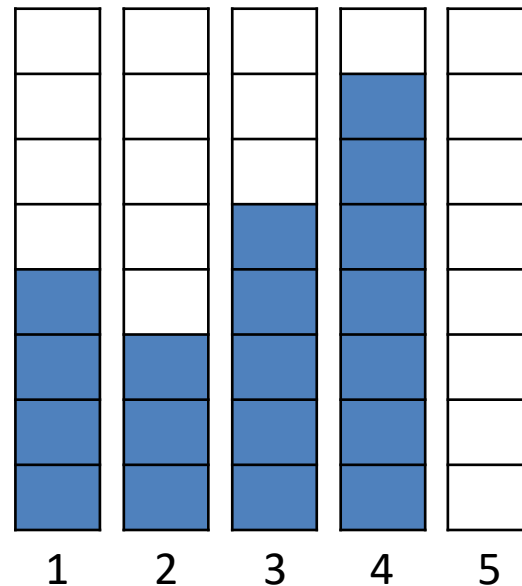
Classic solutions:

1. Decide how you want to **sort the tasks** (decreasing, increasing, or random) and according to which **criteria** (e.g., task utilization, task density, task deadline, or period)
2. Decide what is the **fitness evaluation** method (how will you reject an assignment)
3. Use any of the following fitting policies to assign tasks to processors:

- **First fit (FF)**
- **Best fit (BF)**
- **Worst fit (WF)**
- **Random fit (RF)**
- **Next fit (NF)**

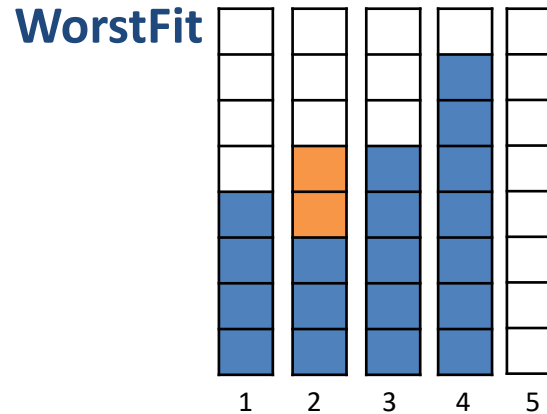
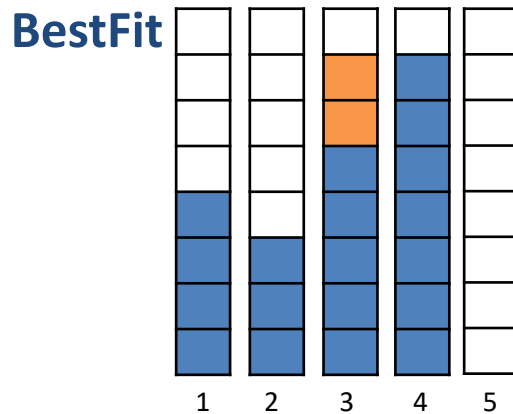
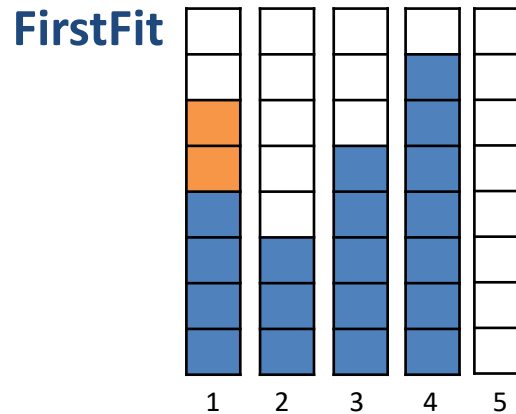
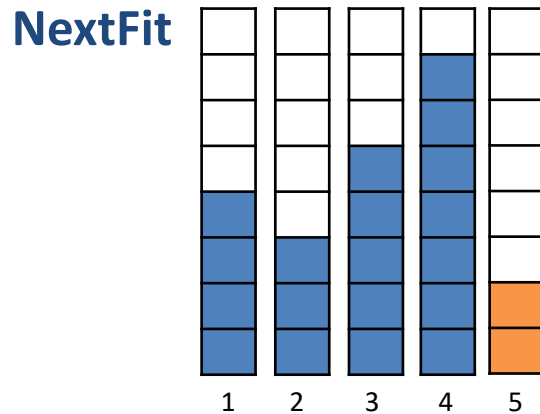
Comparison

- Suppose the current situation is represented in blue, the **latest item was put in bin 4**, and a new item of **size 2 arrives**. What is the resulting bin-packing using First fit (FF), Best fit (BF), Worst fit (WF), and Next fit (NF).



Comparison

- Suppose the current situation is represented in blue, the **latest item was put in bin 4**, and a new item of **size 2 arrives**. What is the resulting bin-packing using First fit (FF), Best fit (BF), Worst fit (WF), and Next fit (NF).



Observations

The performance of each algorithm strongly depends on the input sequence

however:

- **NextFit** has a **poor performance** since it does not exploit the empty space in the previous bins
- **FirstFit** improves the performance by exploiting the empty space available in all the used bins.
- **BestFit** tends to **fill the used bins** as much as possible.
- **WorstFit** tends to **balance the load** among the used bins.

Multicore real-time schedulability tests

Assumptions:

- Identical multicore systems
- WCET of each task is a sound upper bound on the actual execution time of the task for any possible co-running task in any execution scenario.
- Sequential tasks
- No precedence or data dependencies between tasks

Utilization bound for partitioned EDF (with first-fit policy)

The lower bound on the utilization of a task set that is **not schedulable** by **partitioned EDF with FirstFit partitioning** on a multicore platform with m identical cores is

$$U^{EDF+FF} \leq \frac{1}{2} \cdot (m + 1)$$

Example:

$$U_1 = 0.3, U_2 = 0.9, U_3 = 0.7$$
$$m = 3$$

Test if $U < 0.5(3 + 1)$:

It is schedulable because $1.9 \leq 2$

Example:

$$U_1 = 0.3, U_2 = 0.9, U_3 = 0.7, U_4 = 0.5$$
$$m = 3$$

$U = 2.4$ so the test rejects the task set.

However, it is **schedulable** because the partitions will be:

- Core1: task 2
- Core2: task 3
- Core3: task 1 and task 4

Utilization bound for partitioned EDF (with first-fit policy)

Show that the utilization bound for EDF+FF cannot be larger than $0.5(m+1)$.

- We make a counter example that has ϵ larger utilization than $0.5(m+1)$ and is NOT schedulable by EDF+FF.
- In our example, we have
 m processors, $n = m + 1$ tasks

τ_i	C_i	T_i	D_i	U_i
1	$1+\epsilon$	2	2	~ 0.5
2	$1+\epsilon$	2	2	~ 0.5
...	$1+\epsilon$	2	2	~ 0.5
m	$1+\epsilon$	2	2	~ 0.5
$m + 1$	$1+\epsilon$	2	2	~ 0.5

$$U^{tot} = (0.5 + \epsilon) \cdot (m + 1)$$

At least 2 tasks must be assigned to the same processor. However,

$$\frac{1+\epsilon}{2} + \frac{1+\epsilon}{2} > 1$$

→ Not schedulable

Utilization bound for partitioned EDF (with first-fit policy)

A refined bound:

If $n > \beta \cdot m$, then the task set is schedulable by U^{EDF+FF} if

$$U^{EDF+FF} \leq \frac{\beta \cdot m + 1}{\beta + 1}$$

Where U_{max} is the maximum task utilization among all tasks, and $\beta = \left\lfloor \frac{1}{U_{max}} \right\rfloor$ is the maximum number of tasks with utilization U_{max} that fit into one processor.

Example:

$$U_1 = 0.3, U_2 = 0.5, U_3 = 0.5, U_4 = 0.3$$

$$m = 2, n = 4$$

$$U_{max} = 0.5$$

$$\beta = \lfloor 1/0.5 \rfloor = 2$$

It is schedulable because $1.6 \leq \frac{2 \times 2 + 1}{2 + 1}$

Utilization bound of global EDF

The lower bound on the utilization of a task set that is not schedulable by **global EDF scheduling** on a multiprocessor system with m cores is **1**.

Namely: regardless of the number of cores in the system, we may not be able to find a feasible schedule for the tasks even if the utilization is just about 1.

Utilization bound of global scheduling

The Dhall's effect

Example:

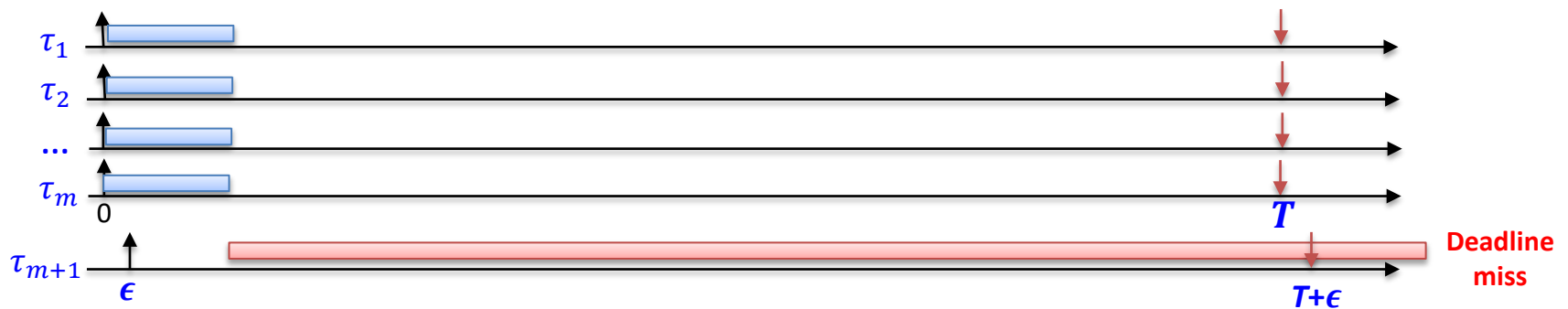
m processors, $n = m + 1$ tasks

$$T \rightarrow \infty \Rightarrow U \rightarrow 1$$

m light tasks,
 $U \sim 0$

1 heavy task $U \sim 1$

τ_i	C_i	T_i	D_i	ϕ_i	U_i
1	1	T	T	0	~ 0
2	1	T	T	0	~ 0
...	1	T	T	0	~ 0
m	1	T	T	0	~ 0
$m + 1$	T	T	T	ϵ	~ 1

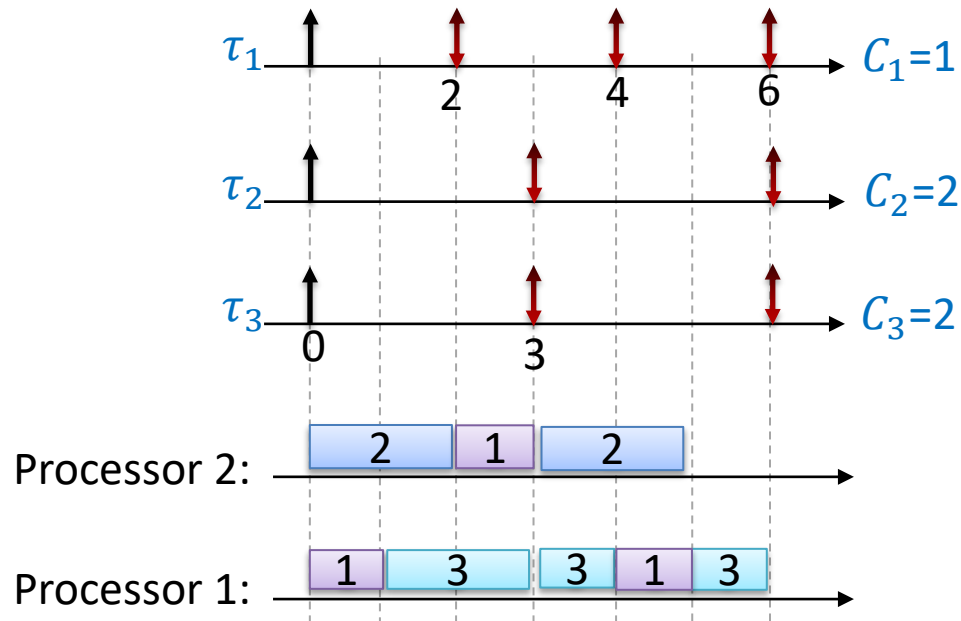


This task set is feasible by partitioned scheduling
or by a non-work-conserving scheduler

Global v.s. partitioned

- There are task sets that are **schedulable only with a global** scheduler
- Example:** $\tau_1 = (C_1 = 1, T_1 = 2)$; $\tau_2 = (2, 3)$; $\tau_3 = (2, 3)$; $Prio_1 > Prio_2 > Prio_3$

$$U^{tot} = \frac{11}{6} = 1.8$$



Schedulable with global FP scheduling

It is impossible to find a feasible schedule using partitioned scheduling:

- We cannot schedule either of the two tasks together on one core since:





$$\frac{1}{2} + \frac{2}{3} > 1$$

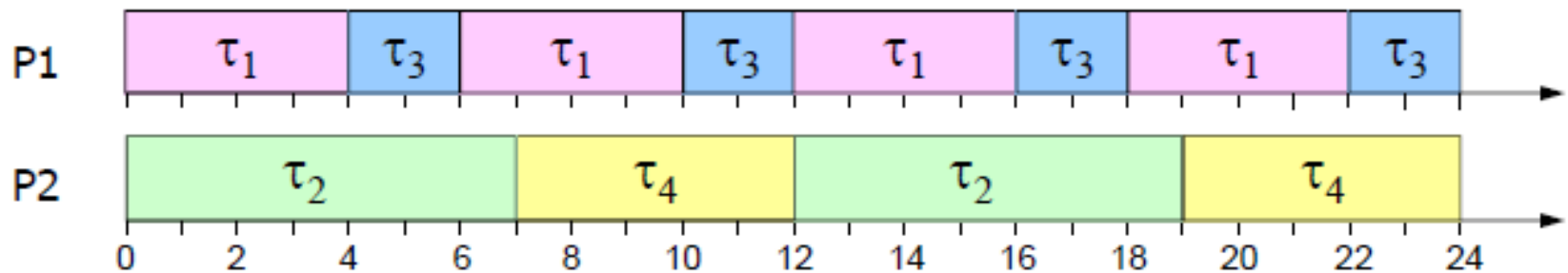
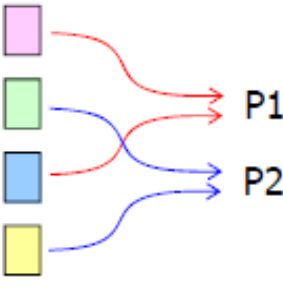
and

$$\frac{2}{3} + \frac{2}{3} > 1$$

Global v.s. partitioned

- There are task sets that are **schedulable only with a partitioned** scheduler
- Example for 2 cores** (assume that each core is scheduled with EDF)

τ_i	C_i	T_i	U_i	
1	4	6	0.6	
2	7	12	0.58	
3	4	12	0.33	
4	10	24	0.41	



The task set is not schedulable by global-EDF and all $4! = 24$ global priority assignments (for global FP) lead to deadline miss.

Global v.s. partitioned

Global and partitioned scheduling are incomparable.

That is, some task sets are schedulable with partitioned but not with global scheduling, and others are schedulable with global but not with partitioned scheduling

Global v.s. partitioned

- ✓ More efficient parallel execution
 - ✓ Automatic load balancing
 - ✓ Lower avg. response time
 - ✓ Easier re-scheduling to react to dynamic loads, selective shutdown, etc.
 - ✓ More efficient reclaiming and overload management
 - × Migration costs
 - × Scheduling overheads
 - × Inter-core synchronization
 - × Weak analysis framework
- ✓ Supported by the industry (e.g., AUTOSAR)
 - ✓ No migrations
 - ✓ Isolation between cores
 - ✓ Mature scheduling framework
 - ✓ Low scheduling overhead (no need to access a global ready queue)
 - × Cannot exploit unused capacity
 - × Dynamically rescheduling not convenient
 - × NP-hard allocation problem

Conclusion

Take-away messages

- **Parallelism** is the new way of **exploiting** the “always increasing” number of transistors in processor chips
- **Multi-core** platforms **introduce** many sources of **non-determinism** for the timing of tasks
- There **exist means to improve timing determinism** but at the cost of worse average-case execution times
- **Global and partitioned** scheduling are **not comparable**
- **Global scheduling** uses more **efficiently platform resources** and allows for **more responsive** systems
- The **theoretical framework** of global scheduling is **weak**, and its **implementation** generates more **overheads**
- **Partitioned scheduling** is **well accepted in industry**



QUIZ TIME

Quiz wrap-up

What is the maximum possible utilization for a "feasible task set" on 5 cores?

- 1
- 2
- 3
- 4
- 5
- 6



Here is a task set with $U = 5$ that is feasible on 5 cores:

- All tasks have WCET = 10 and period 10, and therefore their utilization is 1.
- Each task is assigned to a core.
- So, all cores have utilization of 1.
- The total utilization of this feasible task set is 5.

Quiz wrap-up

What is the utilization bound of global EDF on 8 cores?

- 1
- 2
- 4
- 5
- 6
- 8



Regardless of the number of cores, the minimum schedulable utilization for global EDF is 1 (see Dhall effect).

Quiz wrap-up

Is a task set of implicit deadline sporadic tasks with total utilization $U=4$ schedulable with partitioned EDF on 8 cores?

- yes ✓
- no
- maybe

Yes because the utilization of the task set is smaller than the utilization bound from the following sufficient test

$$U \leq \frac{1}{2} \cdot (m + 1)$$

Quiz wrap-up

Since the utilization bound of global EDF is less than that of partitioned EDF for $m > 1$, partitioned EDF dominates global EDF.

- yes
- no 
- maybe

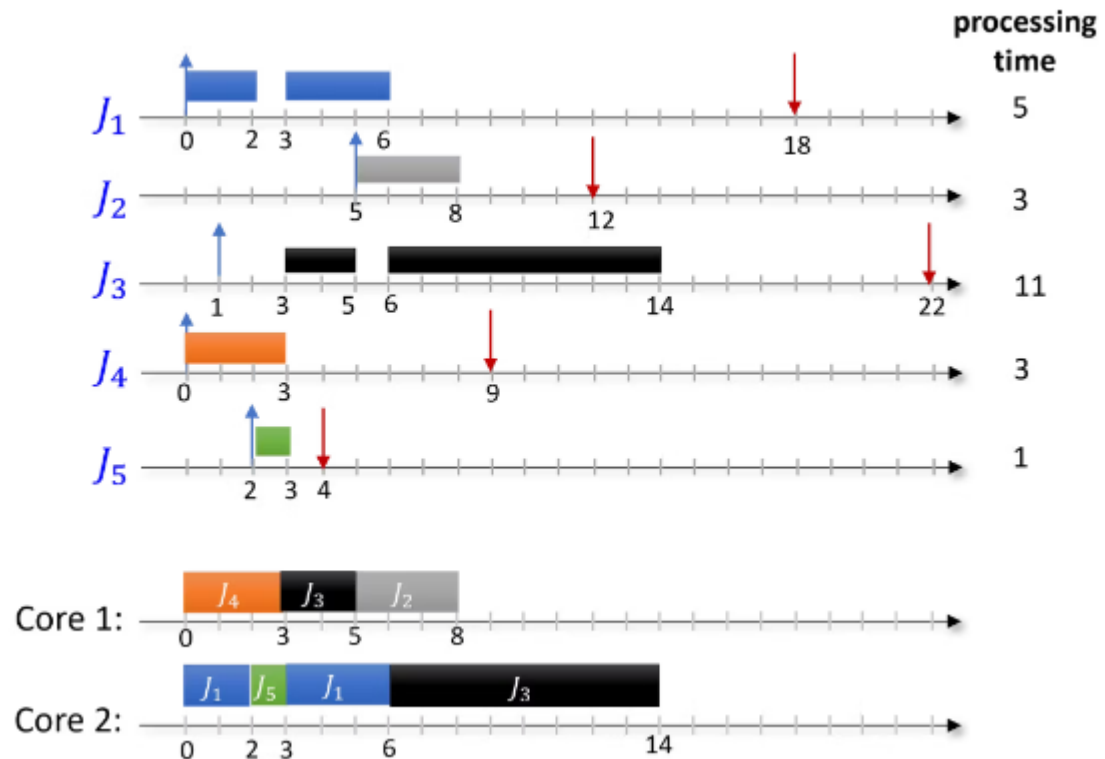
No. As shown in the slides, neither partitioned EDF nor global EDF dominate each other.

Quiz wrap-up

Consider 5 jobs scheduled with global EDF on 2 cores. Which job is preempted by J5?

See the schedule

- None
- J1 ✓
- J2
- J3
- J4
- J5

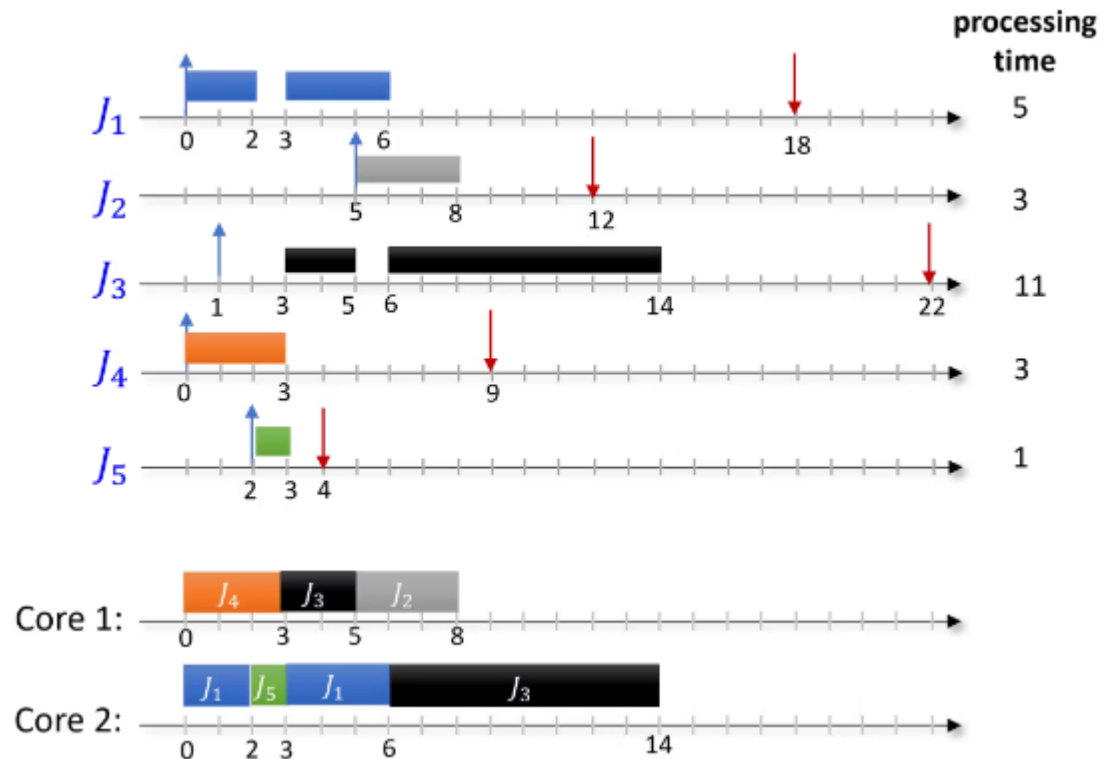


Quiz wrap-up

Consider 5 jobs scheduled with global EDF on 2 cores. Which job migrates?

See the schedule

- None
- J1
- J2
- J3 ✓
- J4
- J5



Self-study slides (included in the exam)

WCET in multicore varies a lot!

Impact of memory bank controllers



Edvard Munch: The Scream

(Paper from 2019)

Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention

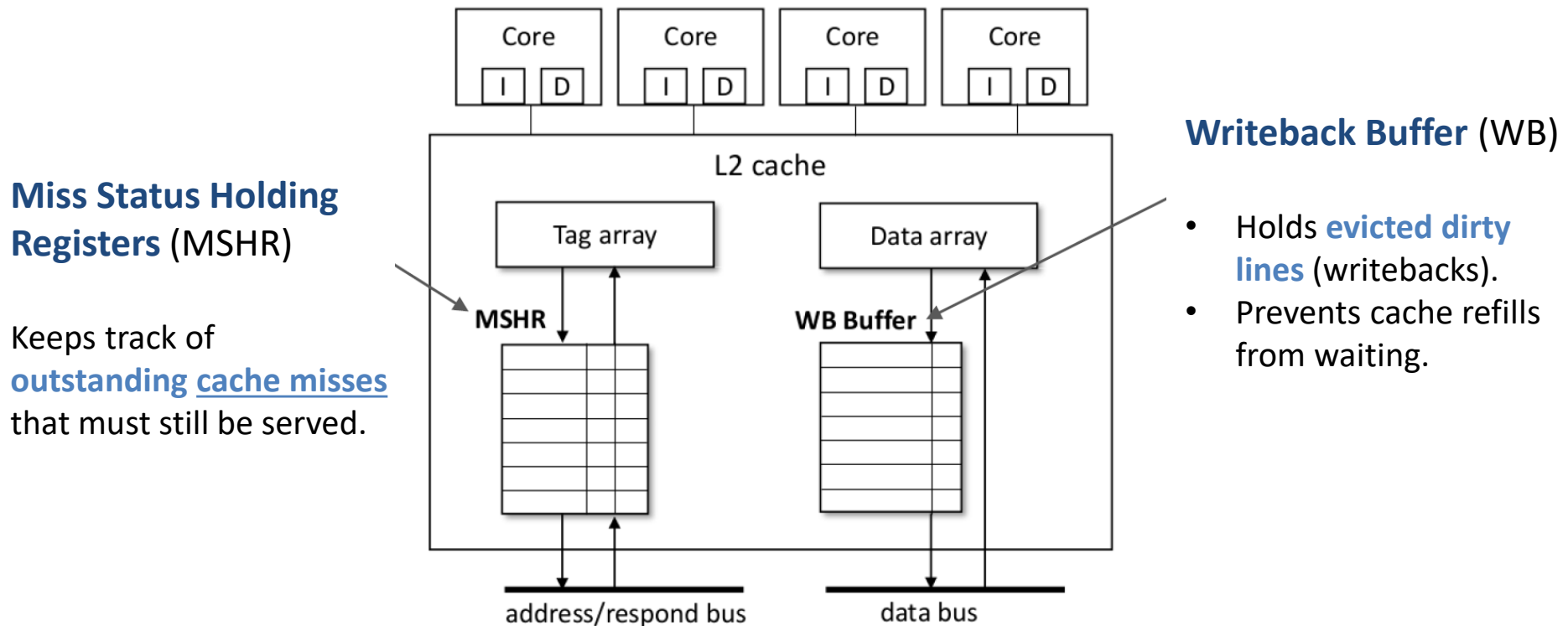
Source: Michael Garrett Bechtel and Heechul Yun. **Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention**. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

Slides: <http://www.ittc.ku.edu/~heechul/papers/cachedos-rtas2019-slides.pdf>

Paper: <http://www.ittc.ku.edu/~heechul/papers/cachedos-rtas2019-camera.pdf>

Non-blocking cache

Allow for multiple concurrent cache accesses and therefore greatly improve performance.



If either structure is full cache does not accept any new request until there is free entries in both the MSHR and the Writeback buffer

Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

“Blocking shared caches”: a timing attack!

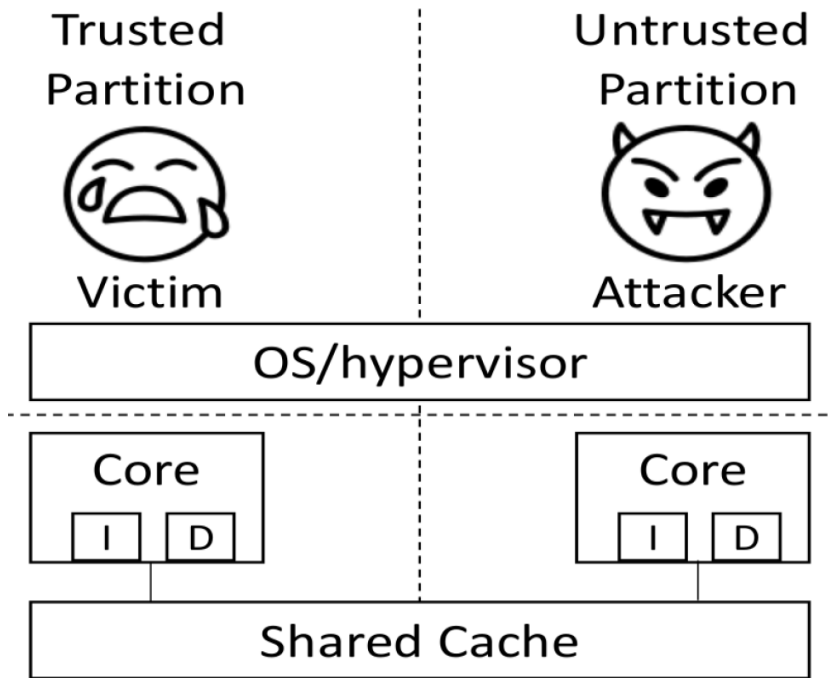
- Blocking shared caches affects all cores
 - No cores can access the cache.
 - Can significantly affect application timings.
- The cache unblocks when MSHRs and Writeback buffer have free entries.
- Unblocking can **take a long time** (since it needs memory access).
 - Can be **maliciously used by attackers**.



Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

Threat model



Assumptions

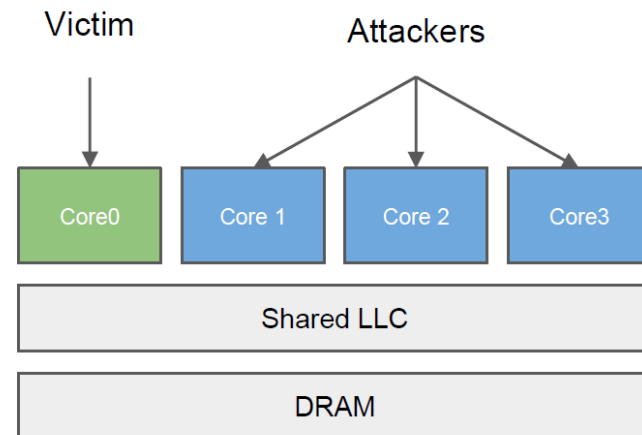
- Attackers can't directly affect the victim.
 - Because of core and memory isolation.
- Attackers can't run privileged code.
- System has a shared cache.

This holds for many systems!

Design of the experiment

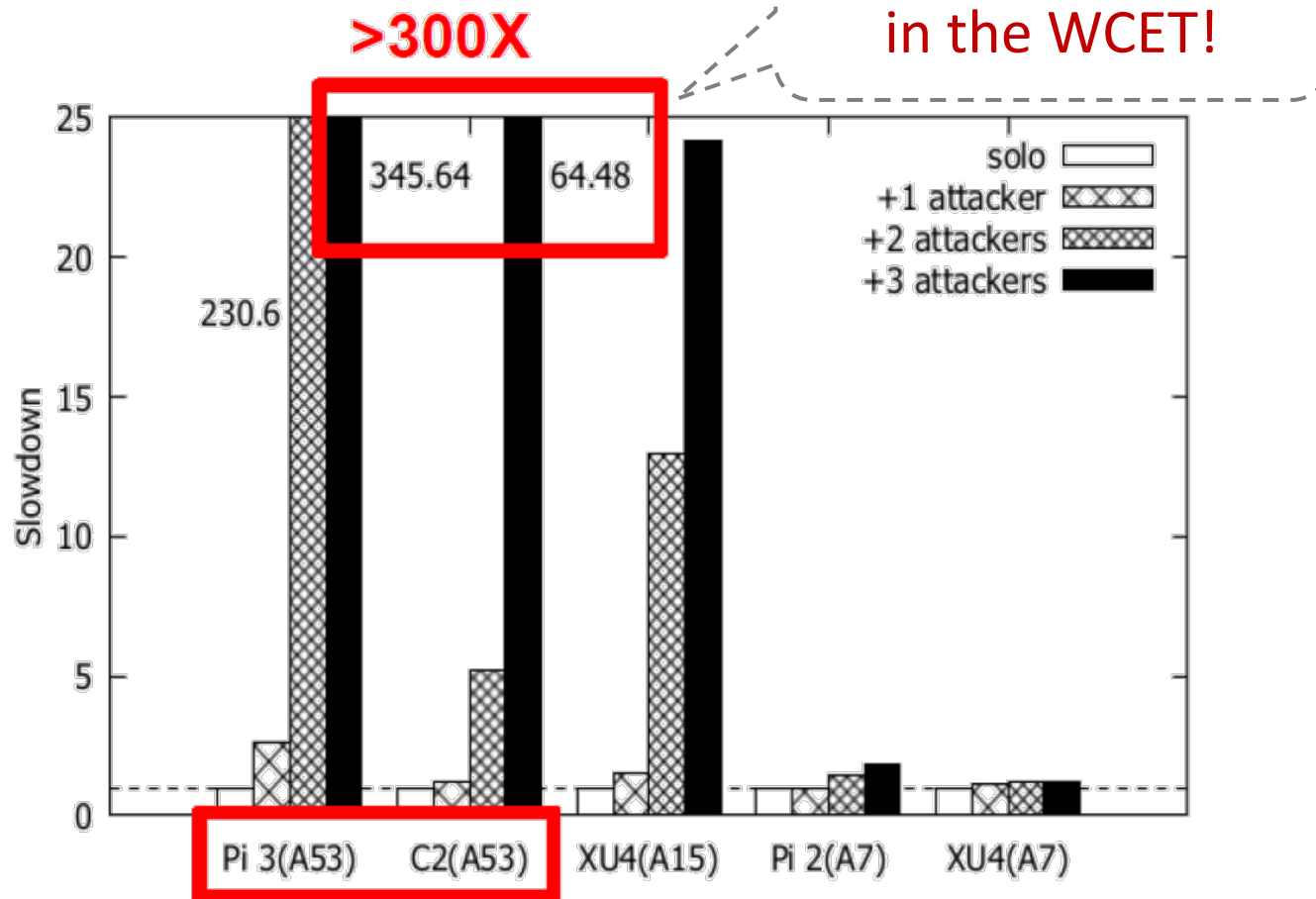
```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
    ptr[i] = 0xff;
}
```

Write Attacker (BwWrite)



Platform	Raspberry Pi 3	Odroid C2	Raspberry Pi 2	Odroid XU4	
SoC	BCM2837	AmlogicS905	BCM2836	Exynos5422	
CPU	4x Cortex-A53	4x Cortex-A53	4x Cortex-A7	4x Cortex-A7	4x Cortex-A15
	in-order	in-order	in-order	in-order	out-of-order
	1.2GHz	1.5GHz	900MHz	1.4GHz	2.0GHz
Private Cache	32/32KB	32/32KB	32/32KB	32/32KB	32/32KB
Shared Cache	512KB (16-way)	512KB (16-way)	512KB (16-way)	512KB (16-way)	2MB (16-way)
Memory	1GB LPDDR2	2GB DDR3	1GB LPDDR2	2GB LPDDR3	
(Peak BW)	(8.5GB/s)	(12.8GB/s)	(8.5GB/s)	(14.9GB/s)	

Results



Michael Garrett Bechtel and Heechul Yun. Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.