Introduction to Multiprocessor Real-Time Systems

WS 2012/2013

Max Planck Institute for Software Systems

Björn Brandenburg
Real-Time Systems Group

Three Kinds of Multiprocessors

	Proc. 1	Proc. 2	Proc. 3
Identical	2 GHz / FPU	2 GHz / FPU	2 GHz / FPU
Uniform Heterogeneous	2 GHz / FPU	1 GHz / FPU	500 MHz / FPU
Unrelated Heterogeneous	1 GHz / FPU	3 GHz / large cache	500 MHz / I/O coproc.

identical:
- ➡ all processors have equal **speed** and **capabilities**

uniform heterogeneous (or homogenous):
- ➡ all processors have equal **capabilities**
- ➡ but **different speeds**

unrelated heterogenous:
- ➡ no regular relation assumed
- ➡ tasks may not be able to execute on all processors

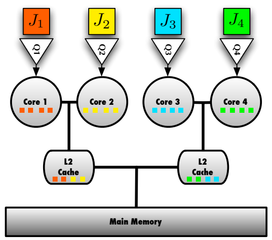
We consider only identical multiprocessors in this class.

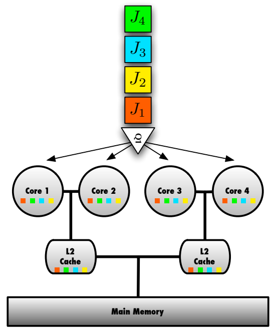
What makes multiprocessor scheduling hard?

*"Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that **a task can use only one processor** even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors."* [emphasis added]

LIU, C. L. (1969). Scheduling algorithms for multiprocessors in a hard real-time environment. In JPL Space Programs Summary, vol. 37-60. JPL, Pasadena, CA, 28–31.

Scheduling Approaches





Partitioned Scheduling
- ➡ task **statically** assigned to cores
- ➡ One ready queue **per core**
- ➡ uniprocessor scheduler on each core

Global Scheduling
- ➡ jobs **migrate** freely
- ➡ All cores serve **shared** ready queue
- ➡ requires new schedulability analysis

Global Scheduling — Dhall Effect

Uniprocessor Utilization Bounds

- EDF = *1*
- Rate-Monotonic (RM) = *ln 2*

Question: What are the utilization bounds on a multiprocessor?

- Notation: ***m*** is the number of processors
- Intuition: would like to **fully utilize** all processors!

Guesses?

- Global EDF = ?
- Global RM = ?

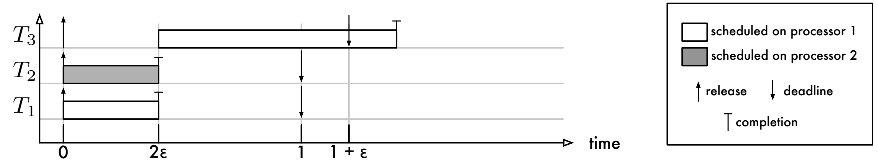
Dhall, S. and Liu, C. (1978). On a real-time scheduling problem. Operations Research, 26(1):127– 140.

Dhall Effect — Illustration

A Difficult Task Set

- ***m + 1*** tasks
- First ***m*** tasks — (***T_i*** for ***1 ≤ i ≤ m***):
  - Period ***= 1***
  - WCET: ***2ε***
- Last task ***T_{m+1}***
  - Period **=** ***1 + ε***
  - WCET **=** ***1***

Total utilization?



Dhall Effect — Implications

Utilization Bounds

- For *ε → 0*, the **utilization bound approaches 1**.
- Adding processors makes no difference!

Global vs. Partitioned Scheduling

- Partitioned scheduling is easier to implement.
- Dhall Effect shows limitation of global EDF and RM scheduling.
- Researchers lost interest in global scheduling for ~25 years.

Since late 1990ies…

- It's a limitation of EDF and RM, not global scheduling in general.
- Much recent work on global scheduling.

Partitioned Scheduling

Reduction to *m* uniprocessor problems

- Assign each task **statically** to one processor
- Use uniprocessor scheduler on each core
  - Either fixed-priority (**P-FP**) scheduling or EDF (**P-EDF**)

Find task mapping such that

- No processor is **over-utilized**
- Each partition is **schedulable**
  - trivial for implicit deadlines & EDF

Connection to Bin Packing

Bin packing decision problem

*Given **a number of bins B**, a bin capacity **V**, and a set of **n** items **$x_1,\ldots,x_n$** with sizes **$a_1,\ldots,a_n$**, does there exist a packing of **$x_1,\ldots,x_n$** that fits into **B** bins?*

Bin packing optimization problem

*Given a bin capacity **V** and a set of **n** items **$x_1,\ldots,x_n$** with sizes **$a_1,\ldots,a_n$**, assign each item to a bin such that the **number of bins** is minimized.*

Bin-Packing Reduction

Bin packing decision problem

*Given **a number of bins B**, a bin capacity **V**, and a set of **n** items **$x_1,\ldots,x_n$** with sizes **$a_1,\ldots,a_n$**, does there exist a packing of **$x_1,\ldots,x_n$** that fits into **B** bins?*

1) Normalize sizes $a_1,\ldots,a_n$ and capacity V
- assume **unit-speed** processors

2) Create an implicit-deadline sporadic task T_i for each item x_i
- with utilization $u_i = a_i / V$
- Pick period arbitrarily, scale WCET appropriately

3) Is the resulting task set feasible under P-EDF on B processors?
- Hence, finding a valid partitioning is NP-hard.

Upper Utilization Bound

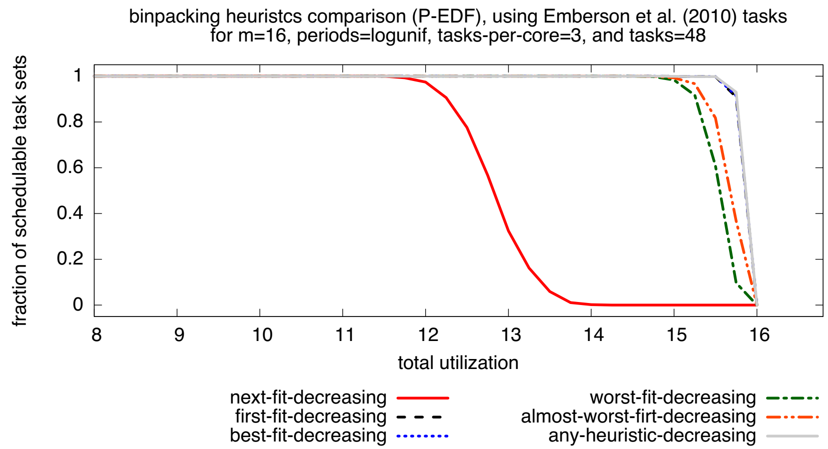
<u>Theorem</u>: there exist task sets with utilizations arbitrarily close to $(m+1)/2$ that cannot be partitioned.

Andersson, B., Baruah, S., and Jonsson, J. (2001). Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 193–202.

A difficult-to-partition task set
- $m + 1$ tasks
- For each T_i for $1 \leq i \leq m + 1$:
  - Period $= 2$
  - WCET: $1 + \varepsilon$
  - Utilization: $(1 + \varepsilon) / 2$

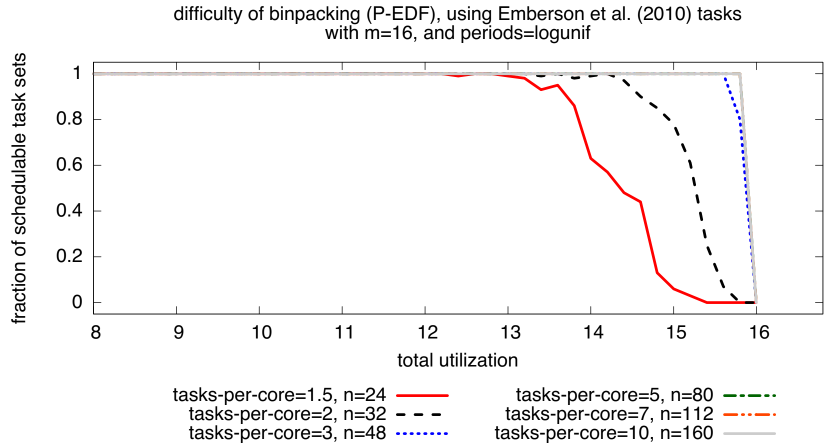
Partitioning not possible
- Any two tasks together over-utilize a single processor by ε!
- Total utilization = $(m + 1) \cdot (1 + \varepsilon) / 2$

Partitioning in Practice (I)



Bottom line: heuristics work well most of the time (for independent tasks).

Partitioning in Practice (II)



Bottom line: larger number of tasks ➔ easier to partition.

Improving Upon Partitioning

Worst-Case Loss

- Partitioning may cause almost up to **50% utilization loss**!
- For **pathological task sets**, the system is half-idle!
- It gets much more difficult for non-independent task sets
  - Locks, precedence, etc.

Can't we do better?

- Can we achieve a utilization bound of ***m***?
- Avoid **offline** assignment phase?
- Global scheduling…

Global Scheduling

General Approach

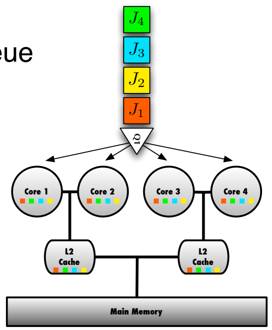
- At **each point in time**, assign **each job** a priority
- At any point in time, schedule the ***m*** highest-priority jobs

Implementation

- Conceptually a globally shared ready queue
- Actual implementation can differ
- **efficient & correct: ongoing research**

Challenges

- migrations require coordination
- cache affinity
- lock contention
- e.g., see Linux



Classification of Scheduling Policies

Task-Level Fixed-Priority (FP) Scheduler (static priorities)

- Each **task** is assigned a fixed priority
- All jobs (of a task) have the same priority
- Example: Rate-Monotonic Scheduling

Job-Level Fixed-Priority (JLFP) Scheduler (dynamic priorities)

- The priority of each task **changes over time**.
- The priority of a job does **not** change.
- Example: EDF

Job-Level Dynamic-Priority (JLDP) Scheduler

- No restrictions.
- The priority of each job changes over time.
- Priorities are a function of **time**, **job identity**, and **system state**.

Unknown Critical Instant

Critical Instant

- Job release time such that **response time is maximized**.
- **Exists** unless system is over-loaded.

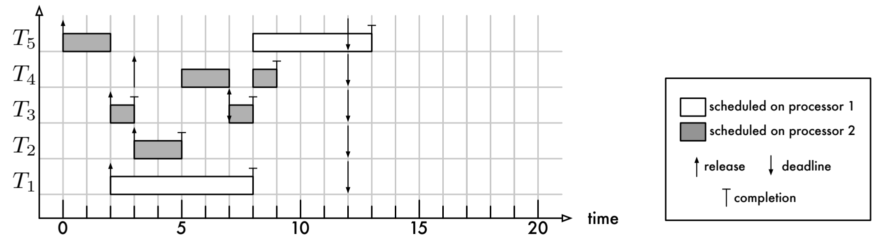
Uniprocessor

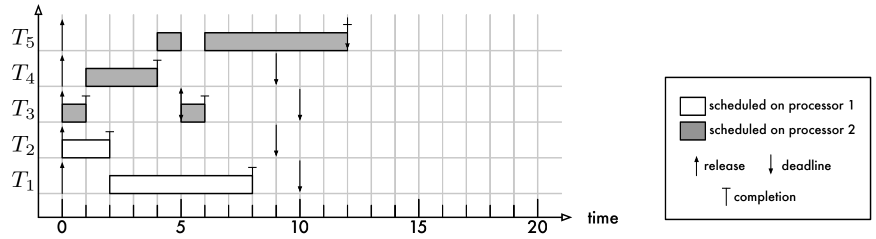
- Liu & Layland: synchronous release sequence yields worst-case response-times
  - synchronous: all tasks release a job at time 0
  - *assuming constrained deadlines and no deadline misses*

Multiprocessors

- **No general critical instant is known!**
- It is **not** necessarily the synchronous release sequence.
- A G-EDF example…

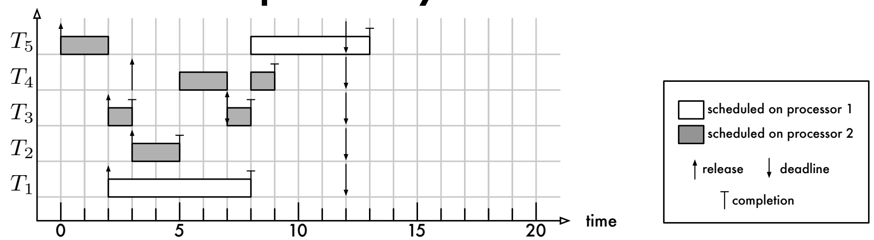
Unknown Critical Instant





The synchronous release sequence is not always the worst case!

Non-Optimality of Global EDF



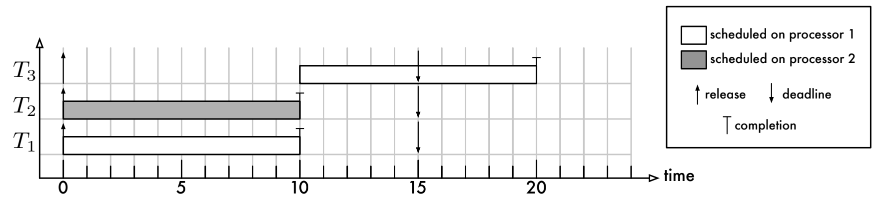
Uniprocessor

- EDF is optimal

Multiprocessor

- G-EDF is not optimal (w.r.t. meeting deadlines)
- Key problem: **sequentiality** of tasks
  - Two processors available for T_5, but it can only use one.

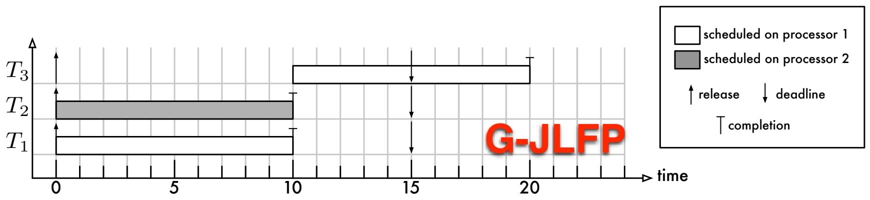
Non-Optimality of G-JLFP Scheduling

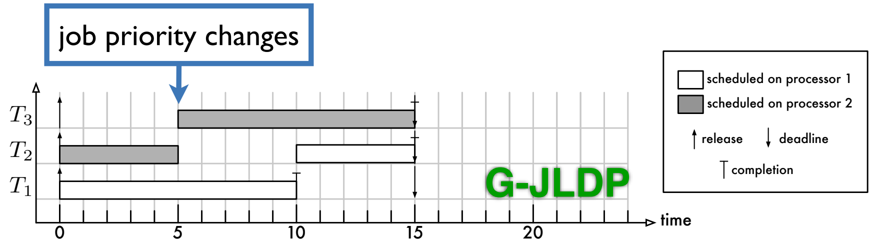


Any Job-Level Fixed-Priority Scheduling Policy is not optimal

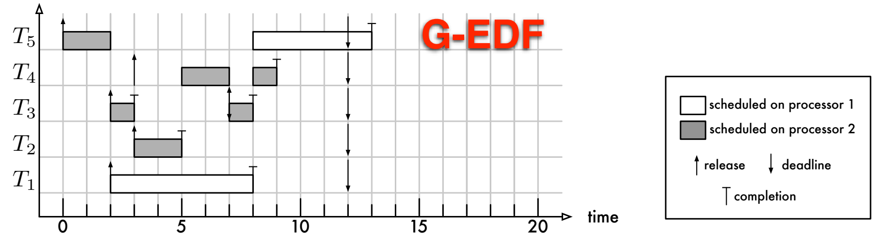
- Example: two processors, three tasks
  - Period 15, WCET = 10
  - synchronous release at time 0
- One of the three jobs is **scheduled last** under **any** **JLFP** policy
  - Deadline miss inevitable!

Global JLDP Example





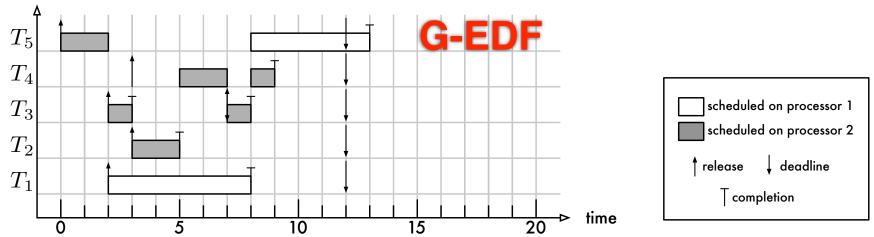
Optimal Multiprocessor Scheduling

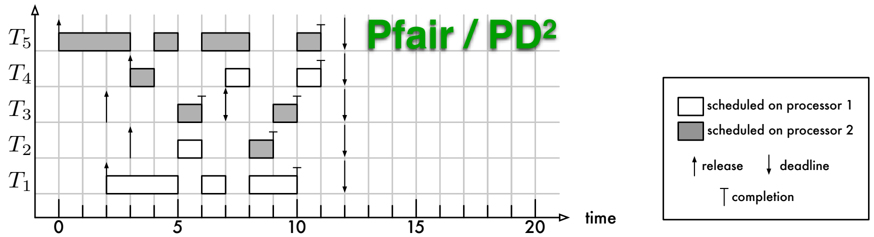


G-EDF is a JLFP Policy

- Can (pseudo-)deadlines be used to schedule correctly?
- **Yes**, but deadlines alone are not enough.
  - Need to break jobs into "smaller pieces".
  - Need appropriate **tie-breaking rules**.
- PD^2

Optimal Multiprocessor Scheduling





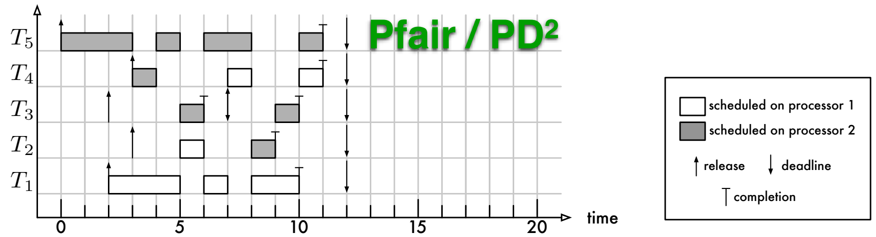
Optimal Multiprocessor Scheduling

Pfair

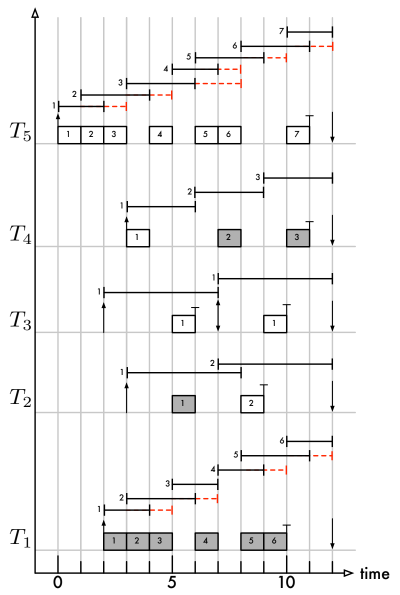
- Notion of "fair share of processor"
- If a schedule is **pfair**, then no **implicit** deadline will be missed.

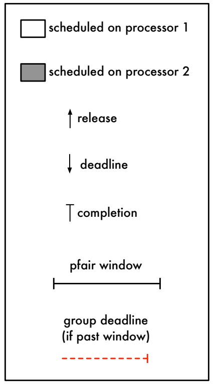
PD^2

- Constructs a **pfair** schedule.
- Splits jobs into **unit-sized** **subtasks**
  - Each subtask has its own **deadline**
- Uses two deadline tie-breaking rules



PD^2 Illustration





Optimal Online Scheduling of Sporadic Tasks with Arbitrary Deadlines

Is it possible to extend **Pfair/PD²** to support **arbitrary deadlines**?

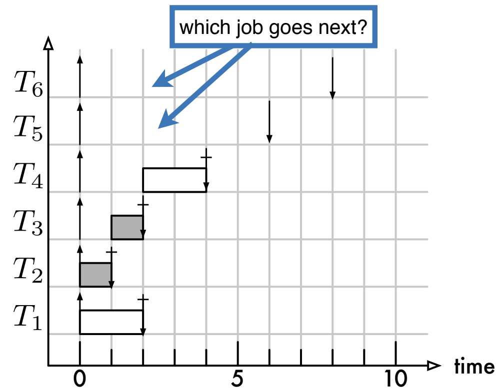
Optimal Online Scheduling of Sporadic Tasks with Arbitrary Deadlines

Theorem: there does not exist an **online** scheduler that **optimally** schedules sporadic tasks with constrained deadlines.

Fisher, Goossens, Baruah (2010), Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real-Time Systems*, volume 45, pp 26-71.

Non-Existence of Optimal Online Schedulers for General Sporadic Tasks

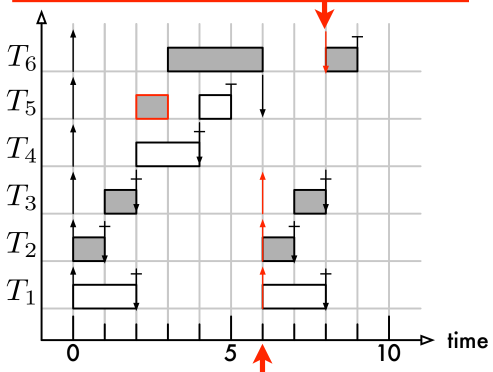
Task	WCET	Deadline	Period
T_1	2	2	5
T_2	1	1	5
T_3	1	2	6
T_4	2	4	100
T_5	2	6	100
T_6	4	8	100



Non-Existence of Optimal Online Schedulers for General Sporadic Tasks

Task	WCET	Deadline	Period
T_1	2	2	5
T_2	1	1	5
T_3	1	2	6
T_4	2	4	100
T_5	2	6	100
T_6	4	8	100

If T_5 goes first, then T_6 can miss its deadline.

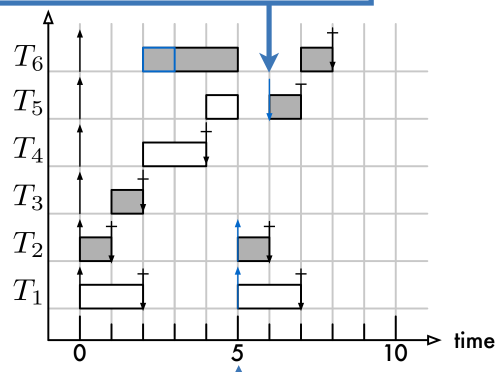


New jobs at **time 6**.

Non-Existence of Optimal Online Schedulers for General Sporadic Tasks

Task	WCET	Deadline	Period
T_1	2	2	5
T_2	1	1	5
T_3	1	2	6
T_4	2	4	100
T_5	2	6	100
T_6	4	8	100

If T_6 goes first, then T_5 can miss its deadline.

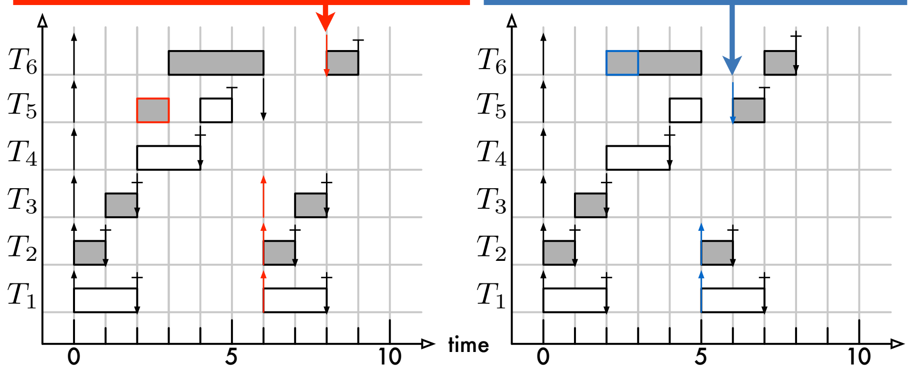


New jobs at **time 5**.

Non-Existence of Optimal Online Schedulers for General Sporadic Tasks

If T_5 goes first, then T_6 can miss its deadline.

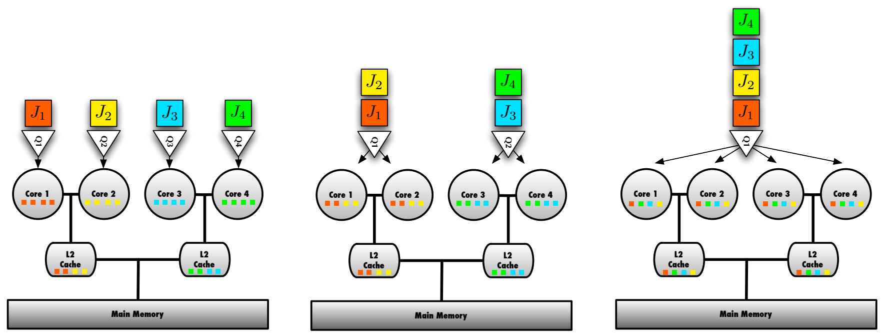
If T_6 goes first, then T_5 can miss its deadline.



The task set is **feasible**, but correct decision requires **knowledge of future arrivals**!

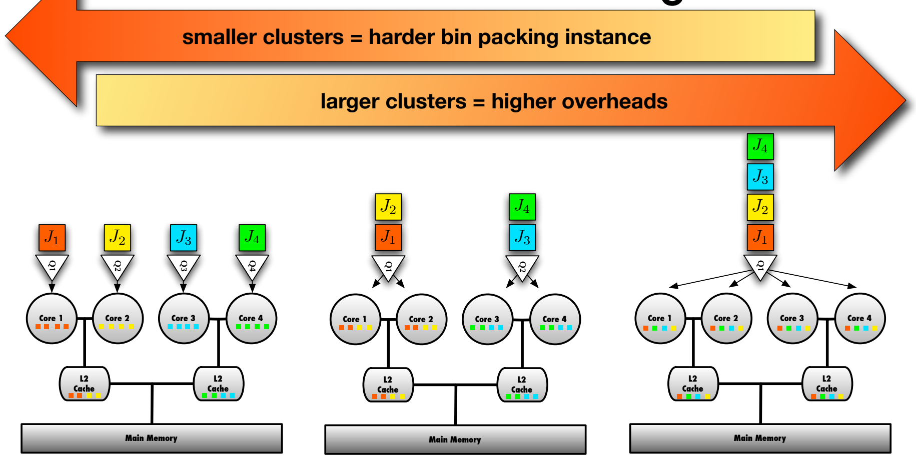
Clustered Scheduling

A hybrid / generalization of global and partitioned scheduling.



partitioned scheduling — clustered scheduling — global scheduling

Clustered Scheduling



partitioned scheduling   clustered scheduling   global scheduling

Semi-Partitioned Scheduling

another generalization partitioned scheduling

Partition first
- **Assign each task statically** to a processor if possible
- Keep track which tasks could not be assigned (if any)
- Details vary according to specific **semi-partitioned** algorithm

Split remaining tasks across multiple processors
- **Split** each **unassigned task** into multiple “portions” or “chunks”
- **Distribute** portions/chunks among multiple processors
  - E.g., split each job into **subjobs with precedence constraints**
  - Alternatively, do not migrate jobs, but **vary a task’s processor assignment** over time (soft real-time)

Summary

Approaches
- Partitioned
- **Global**
- Hybrid
  - Clustered
  - Semi-Partitioned

Priorities
- Task-Level Fixed Priority
- Job-Level Fixed Priority
- **Job-Level Dynamic Priority**

Optimal Online Scheduling
- **Implicit deadlines**: requires global job-level dynamic priority scheduler
- Constrained deadlines: does not exist
- Arbitrary deadlines: does not exist