

Real-Time Systems

Hard Real-Time Multiprocessor Scheduling

Marcus Völz

Outline

- Introduction
- Terminology, Notation and Assumptions
- Anomalies + Impossibility Results
- Partitioned Scheduling (no migration)
- Global Scheduling (task- / job-level migration)
 - G-FP
 - G-EDF
- Optimal MP Scheduling
- MP – Resource Access Protocols
- Open Research Issues

Lessons Learned From UP

- **Liu-Layland Criterion for fixed task priority algorithms:**
 - can schedule any workload up to a utilization of $U_{RMS}(n) = n\sqrt{2} - 1 \leq 0.693$
 - scheduling of workloads with higher utilization not guaranteed
- **fixed job priority algorithms are optimal: (e.g., EDF)**
- **there are optimal greedy algorithms**
 - with a single measure characterizing the “importance” of a job (e.g., time to deadline, laxity, ...)
- **all preemptive FTP, FJP algorithms are predictable**
 - response times cannot increase when decreasing execution times
- **all preemptive FTP algorithms and EDF are sustainable**
 - no period / deadline anomalies
- **simultaneous release is critical instant**
- **response times depend on set but not on order of high-prio. tasks**

Taxonomy of Multiprocessor Scheduling

Two problems to solve:

- Priority Problem: When to run a given job of the workload?
 - fixed task priority (e.g., RMS, ...)
 - fixed job priority (e.g., EDF, ...)
 - dynamic job priority (e.g., LST, ...)
- Allocation Problem: Where to run this job?
 - no migration
 - task-level migration (no migration of running jobs)
 - job-level migration (migrate also running jobs)

partitioned

global

Taxonomy of Multiprocessor Scheduling

Two problems to solve:

- Priority Problem: When to run a given job of the workload?
 - fixed task priorities
 - fixed job priorities
 - dynamic job priorities
- Allocation Problem: Where to run this job?
 - no migration
 - task-level migration
 - job-level migration

Preemption Costs – UP / MP

Migration Costs – MP

partitioned

global

Preemption Costs

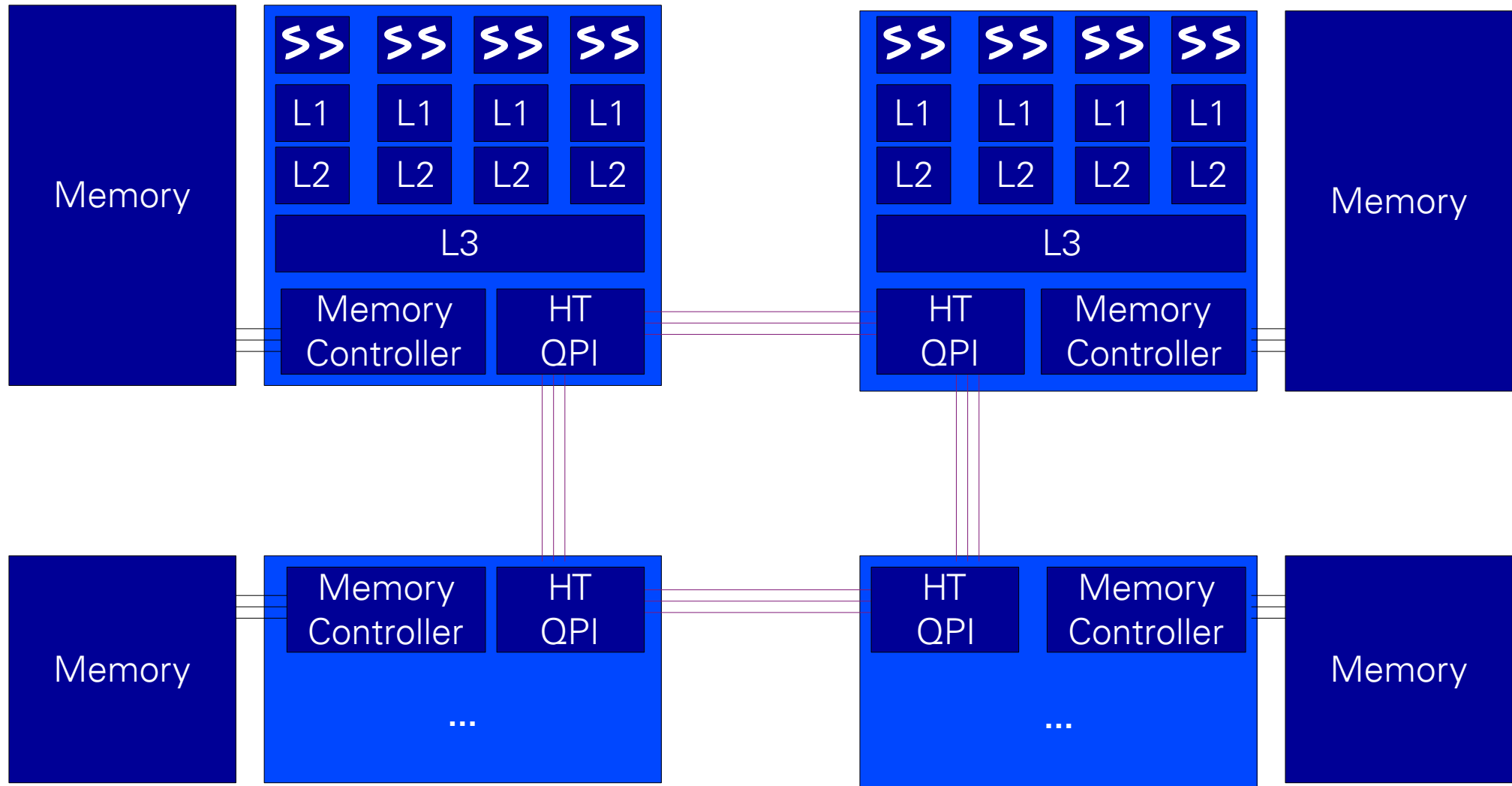
- direct costs:
 - timer / device interrupt
 - save register state
 - manipulate ready list
 - UP: no synchronization required
 - load register state of next thread
- indirect costs
 - cache evictions between two consecutive runs
 - TLB refills after evictions / shutdown

Migration Costs

- job-level migration
 - migration of running job implies preemption at source CPU
- task-level migration
 - job is already preempted
- direct costs
 - manipulate remote / global ready list
 - synchronization
 - fetch register state
- indirect costs
 - fetch active cache working set from remote cache
 - load remaining data from remote memory

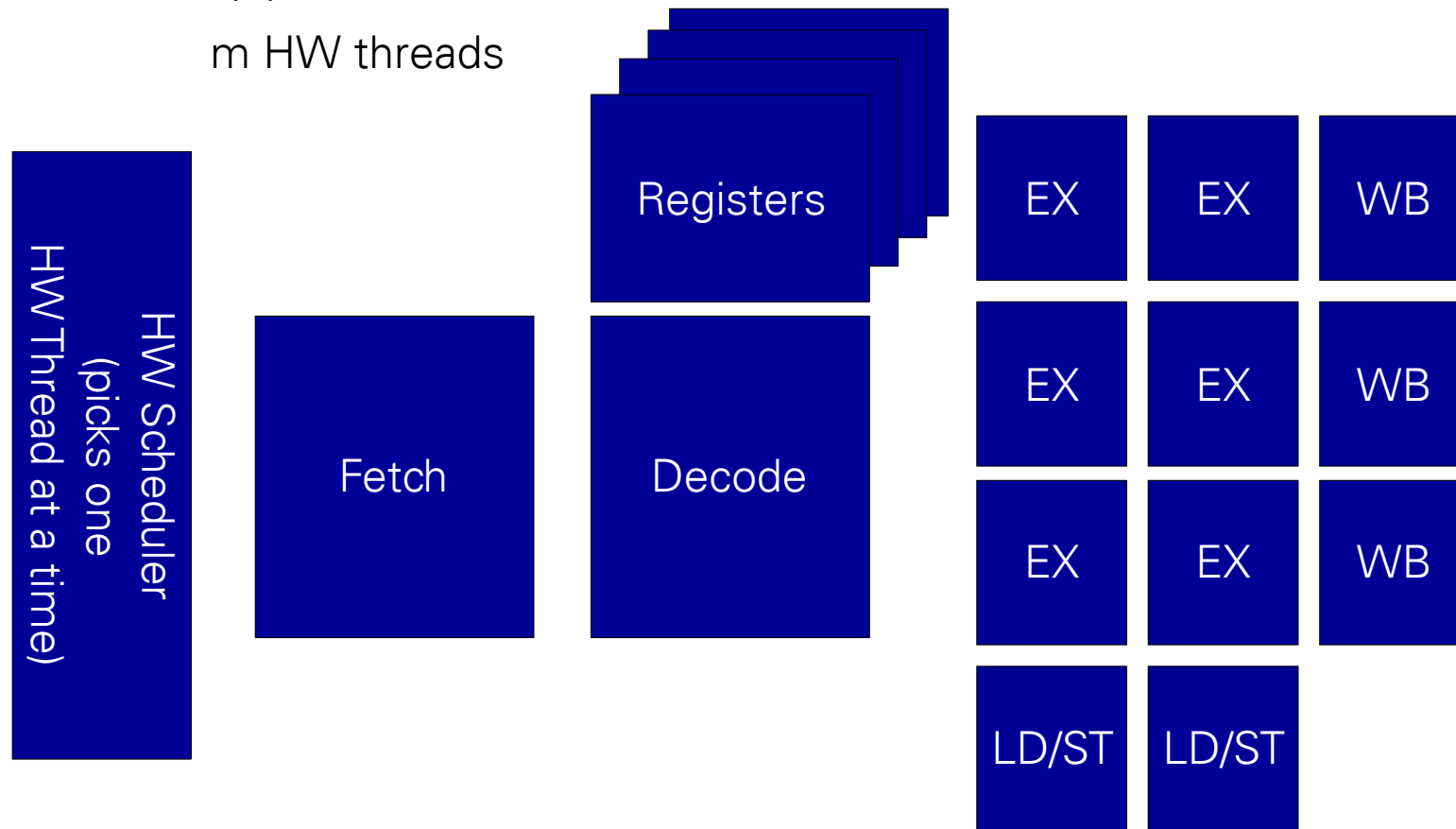
Multiprocessor Architectures

- AMD Opteron / Intel Core Duo: SMT + multi core + ccNuma



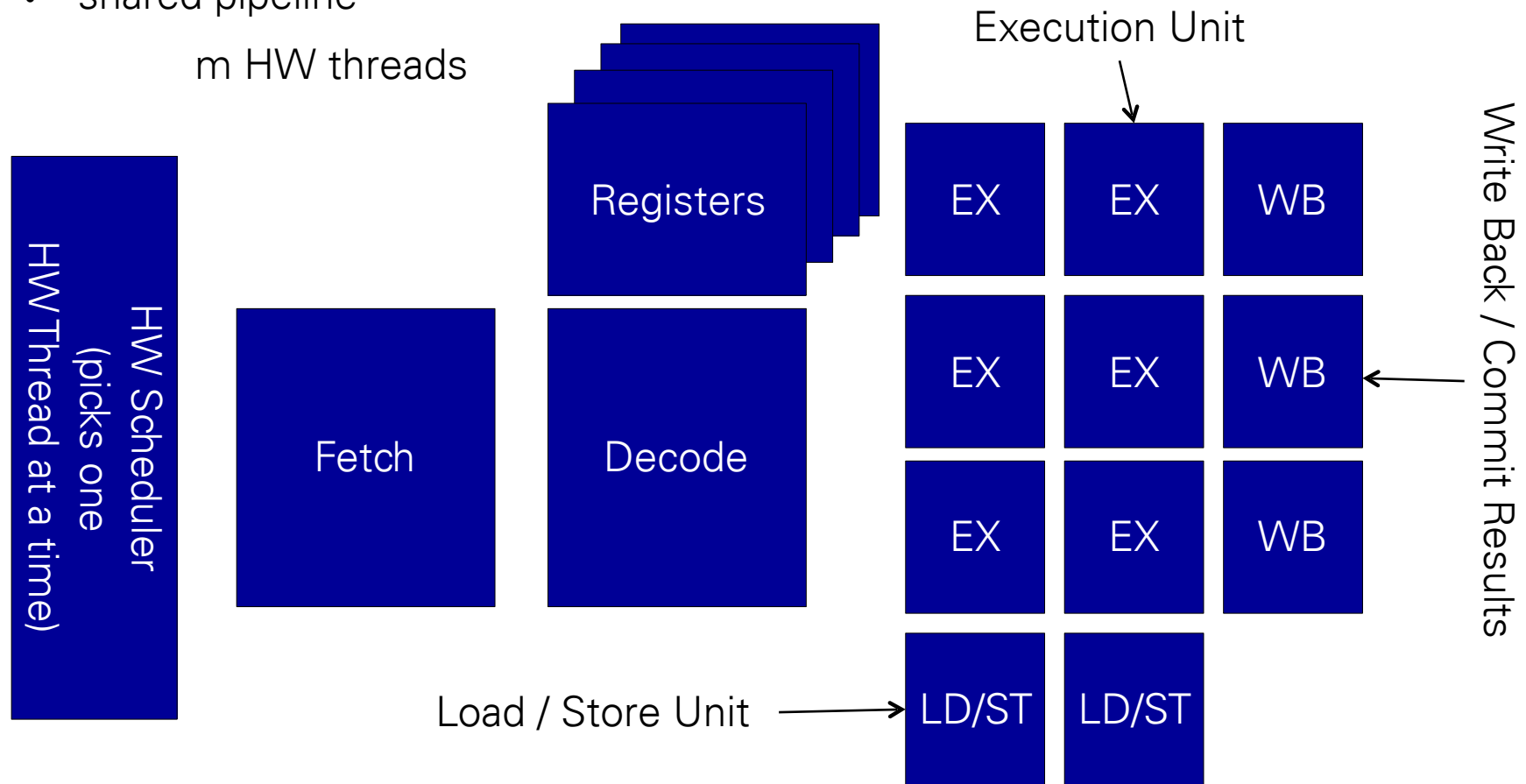
Multiprocessor Architectures

- Symmetric Multi-Threaded (SMT) Processors
 - m hardware threads
 - shared pipeline



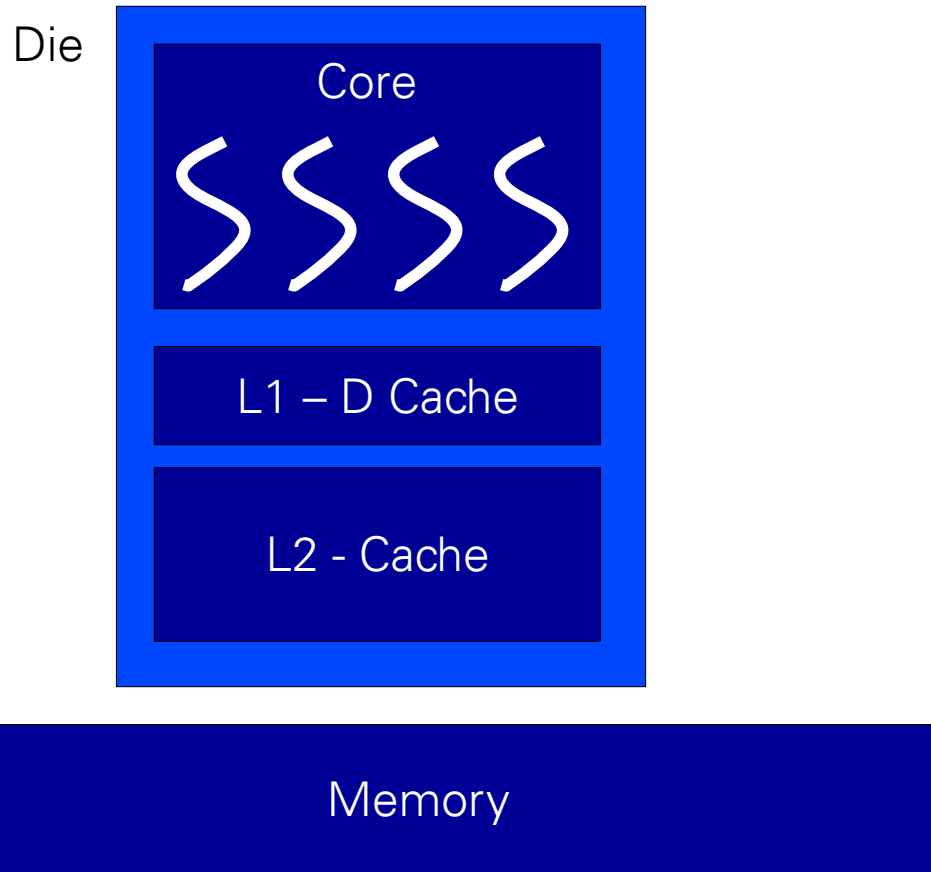
Multiprocessor Architectures

- Symmetric Multi-Threaded (SMT) Processors
 - m hardware threads
 - shared pipeline



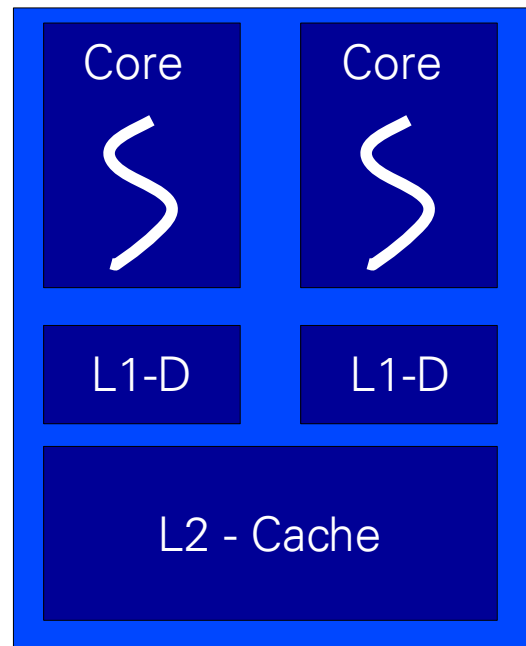
Multiprocessor Architectures

- Symmetric Multi-Threaded (SMT) Processors
 - operating system multiplexes n SW threads on m HW threads
 - caches + pipeline is shared \Rightarrow no indirect migration costs



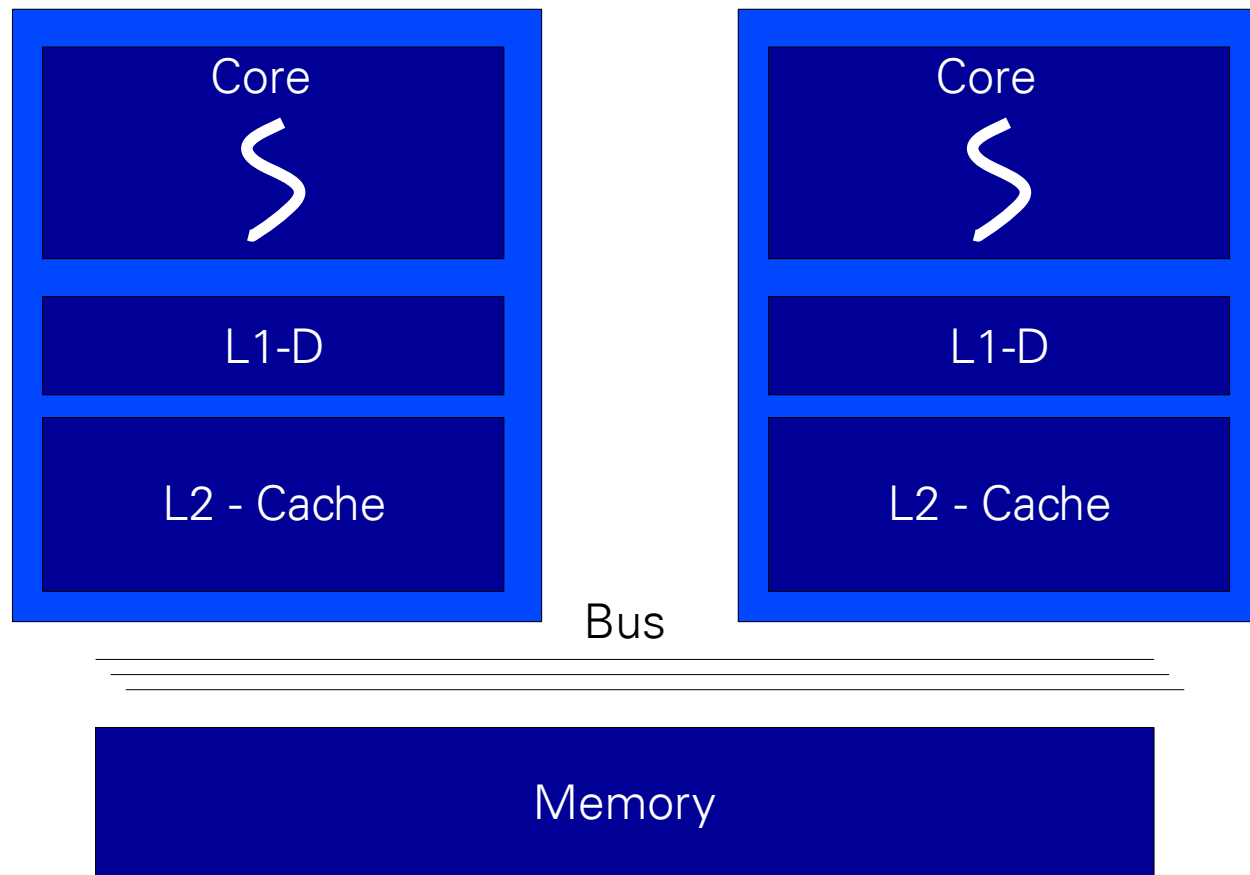
Multiprocessor Architectures

- Multi-Core Processors
 - operating system multiplexes n SW threads on m cores
 - timing of last level cache dominates migration costs



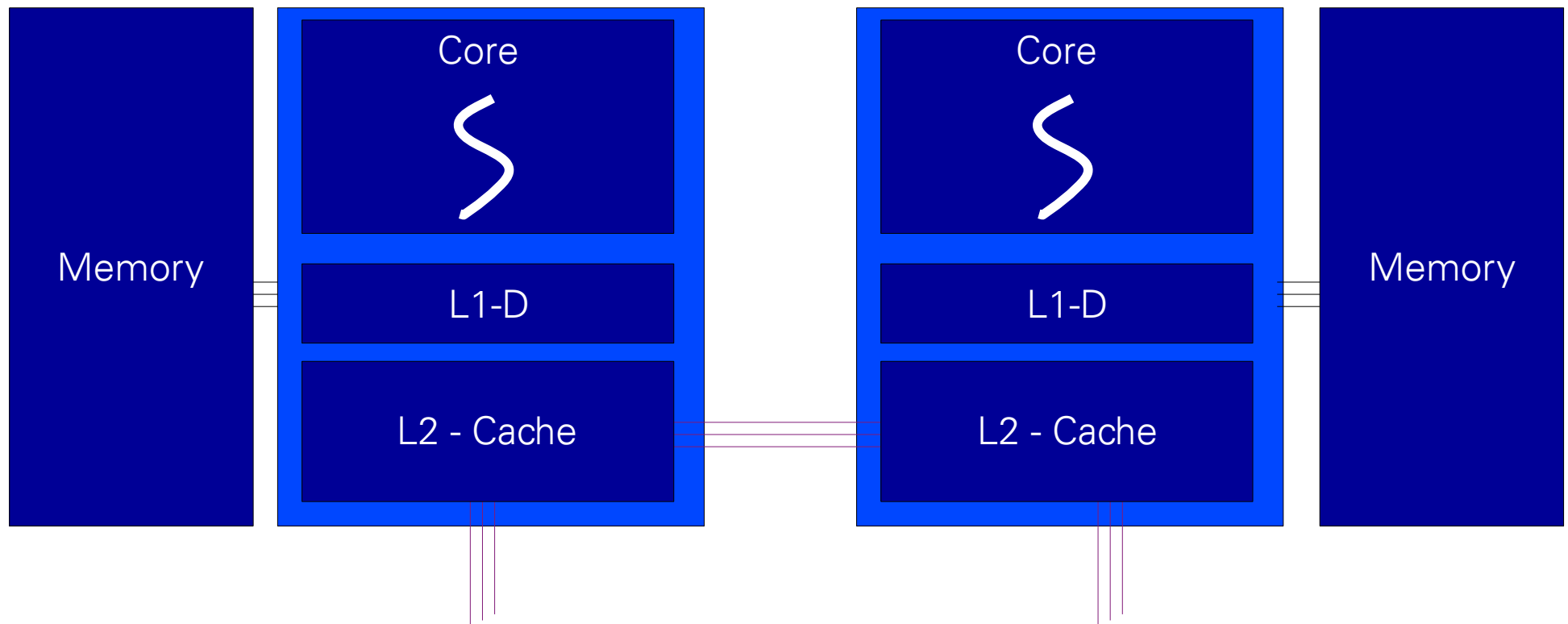
Multiprocessor Architectures

- Symmetric Multiprocessors
 - operating system multiplexes n SW threads on m dies
 - timing of interconnect dominates migration costs



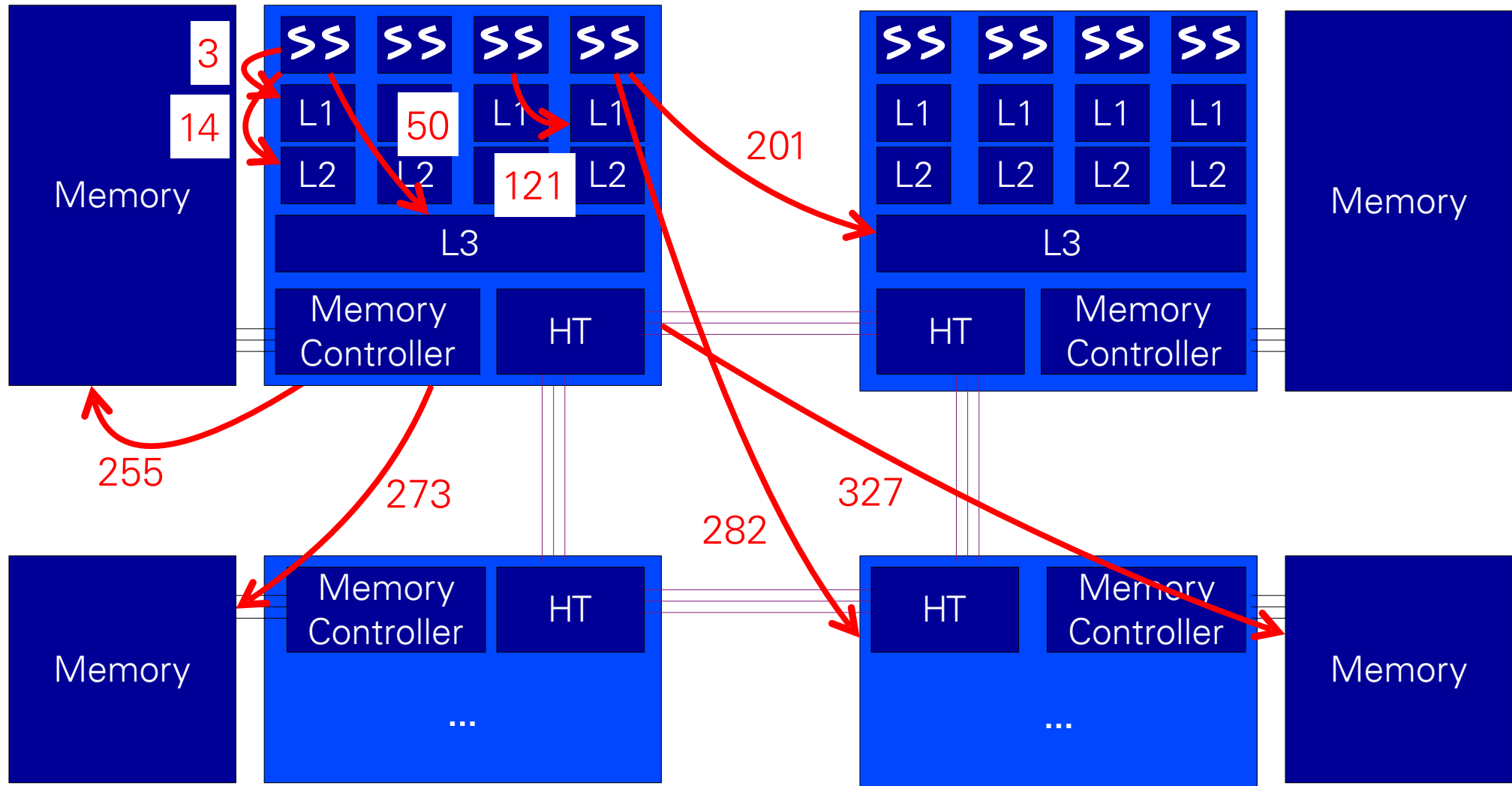
Multiprocessor Architectures

- (cache coherent) NUMA
 - like SMP
 - non-uniform memory access: fetch from remote memory



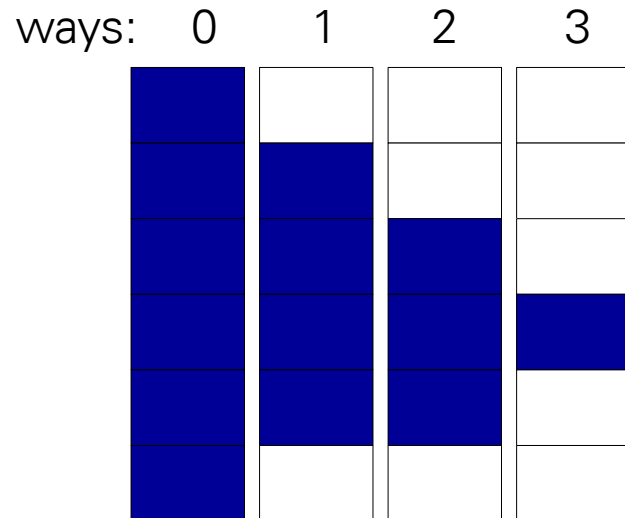
Multiprocessor Architectures

- AMD Opteron [Corey: OSDI '08]



Migration Costs

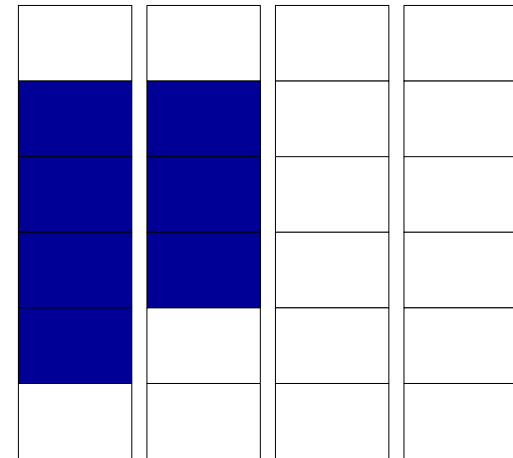
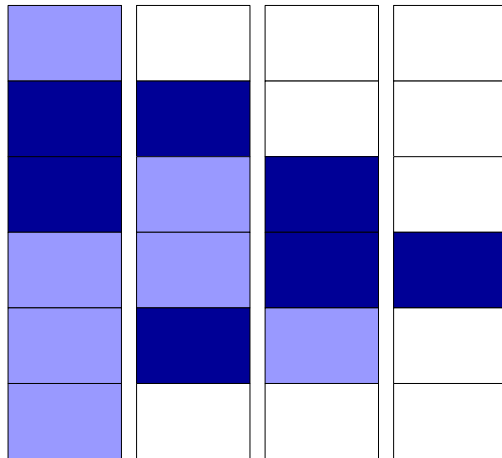
- Active Cache Working Set
 - cachelines a thread would access again if it would run
 - varies over time
 - ages out after preemption



Migration Costs

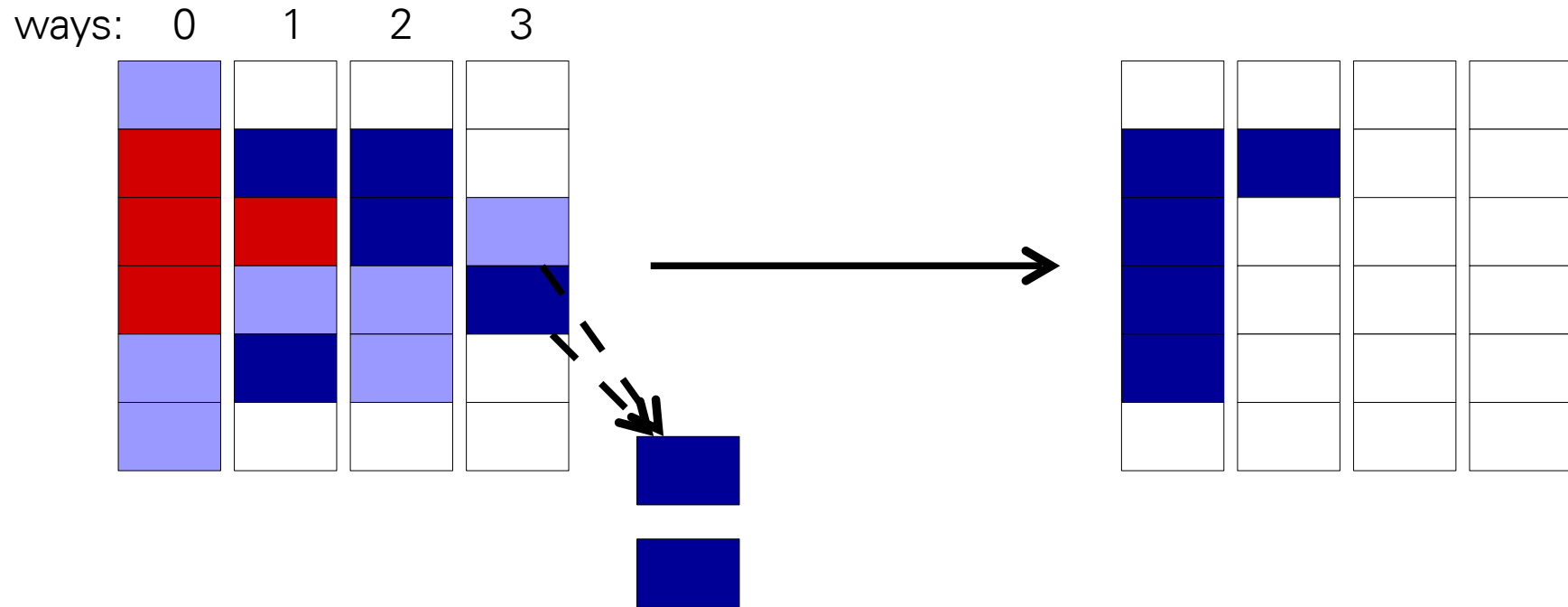
- Active Cache Working Set
 - cachelines a thread would access again if it would run
 - varies over time
 - ages out after preemption

ways: 0 1 2 3



Migration Costs

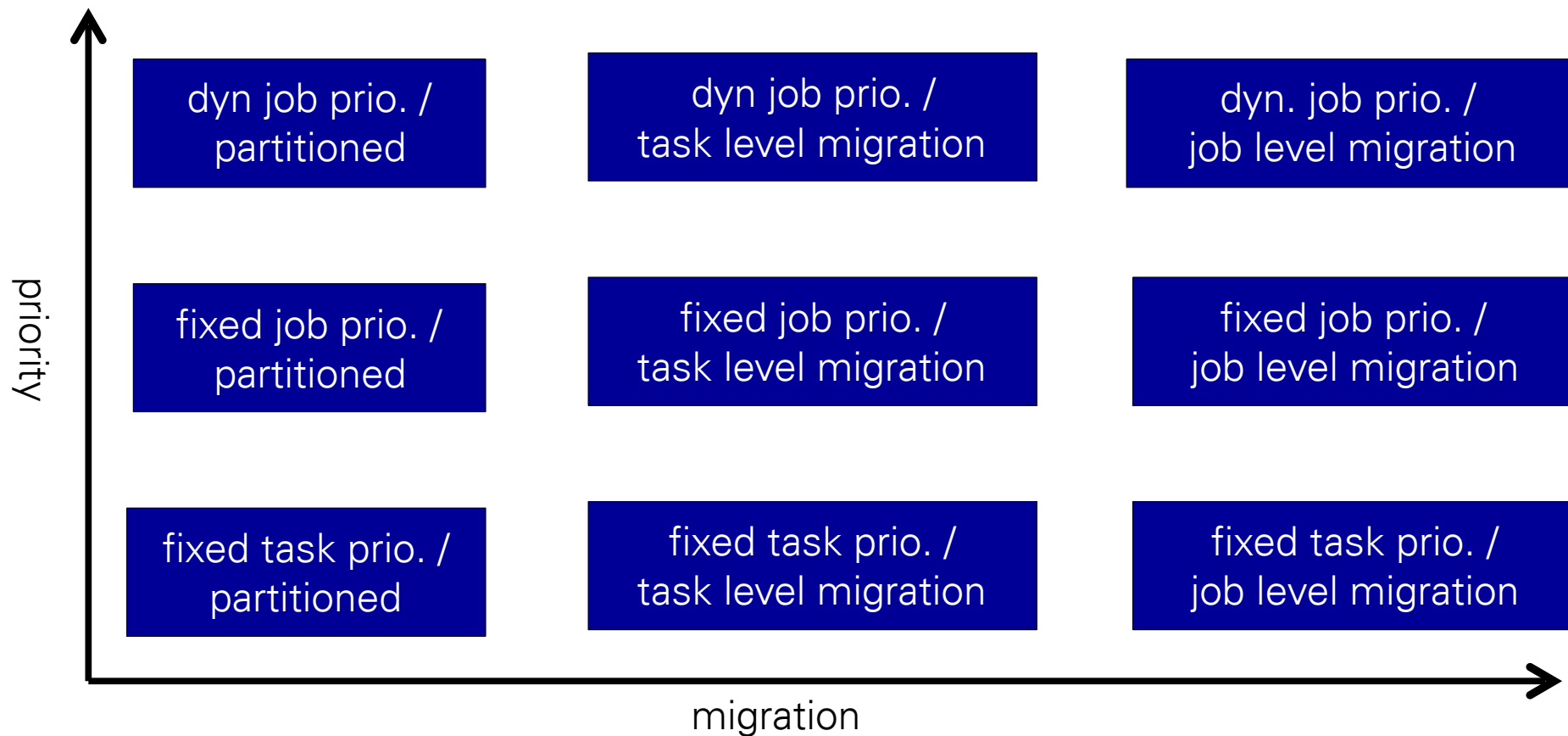
- Active Cache Working Set
 - cachelines a thread would access again if it would run
 - varies over time
 - ages out after preemption



Migration Costs

- Summary
 - migration costs are highly architecture dependent
 - non-trivial to predict
 - may cause a significant delay when a thread resumes execution
- Assumption for the remainder of this lecture:
 - zero preemption and migration costs / attributed to WCET

Design Space of MP Scheduling



Design Space of MP Scheduling

Partitioned

dyn job prio. /
partitioned

fixed job prio. /
partitioned

fixed task prio. /
partitioned

Global

dyn job prio. /
task level migration

dyn. job prio. /
job level migration

fixed job prio. /
task level migration

fixed job prio. /
job level migration

fixed task prio. /
task level migration

fixed task prio. /
job level migration

Design Space of MP Scheduling

Relative ordering between classes of scheduling algorithms

dyn. job prio. / job level migration

dyn job prio. /
task level migration

fixed job prio. /
partitioned

dyn job prio. /
partitioned

fixed job prio. /
task level migration

fixed job prio. /
job level migration

fixed task prio. /
partitioned

fixed task prio. /
task level migration

fixed task prio. /
job level migration

Later in this lecture

Outline

- Introduction
- Terminology, Notation and Assumptions
- Anomalies + Impossibility Results
- Partitioned Scheduling
- Global (Task-Lvl migration) Scheduling
 - G-FTP (e.g., G-RMS)
 - G-EDF
- Optimal MP Scheduling
- MP – Resource Access Protocols
- Open Research Issues

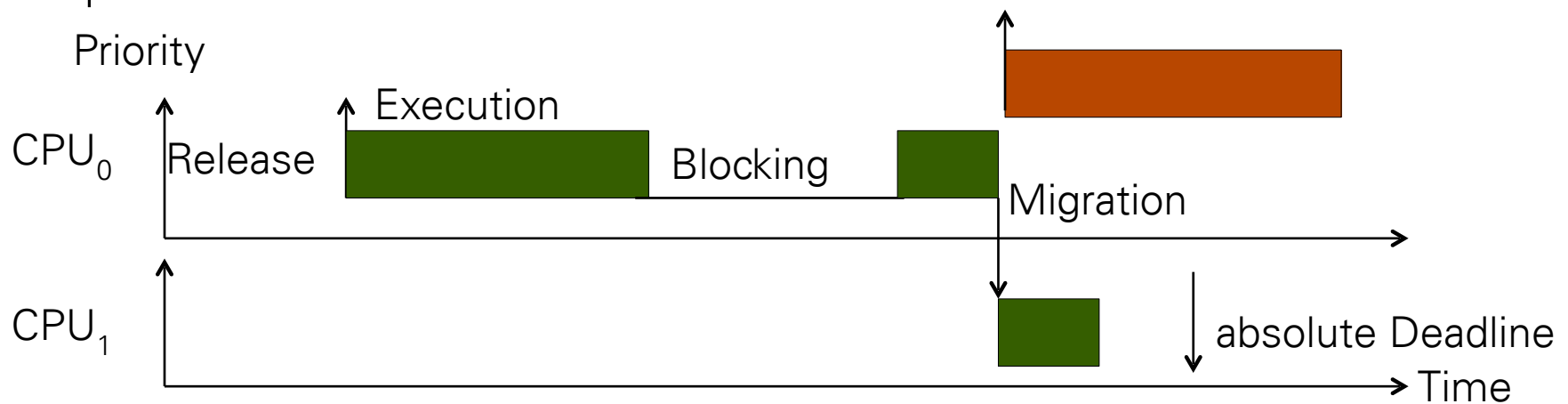
Terminology, Notation and Assumptions

- Periodic Tasks
 - Task $\tau_i = (P_i, D_i, C_i)$ $P_i = \text{const.}$
- Sporadic Tasks
 - $P_i = \text{Minimal Interrelease Time}$
- Deadlines
 - implicit deadline: $D_i = P_i$ (relative deadline = period)
 - constrained: $D_i \leq P_i$ (relative deadline < period)
 - arbitrary (deadline may be after period end)
- Utilization
- Density

$$U_i = \frac{C_i}{P_i}$$
$$\partial_i = \frac{C_i}{\min(D_i, P_i)}$$

Terminology, Notation and Assumptions

- Assumptions for the remainder of this lecture
 - independent tasks
 - fully preemptible / migratable (negligible costs)
 - unlimited number of priorities
 - tasks are single threaded: a job can utilize only 1 CPU at a time
 - jobs do not block (shared resources later in this lecture)
- pictures show schedules for 2 CPUs

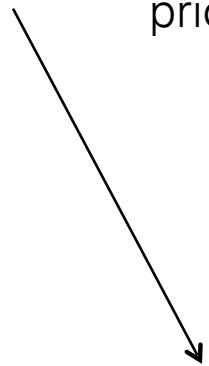


Terminology: P-DMS / G-RMS / G-EDF

- Scheduling Algorithms:

Deadline Monotonic Scheduling:

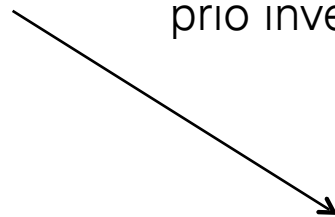
prio inverse proportional to deadline



- P-DMS

Rate Monotonic Scheduling:

prio inverse proportional to period



G-RMS / G-EDF

Earliest Deadline First:

job prio. inverse proportional to deadline



Terminology: P-DMS / G-RMS / G-EDF

- Scheduling Algorithms:

Deadline Monotonic Scheduling:

prio inverse proportional to deadline

Rate Monotonic Scheduling:

prio inverse proportional to period

Earliest Deadline First:

job prio. inverse proportional to deadline

- P-DMS

G-RMS / G-EDF

Global:

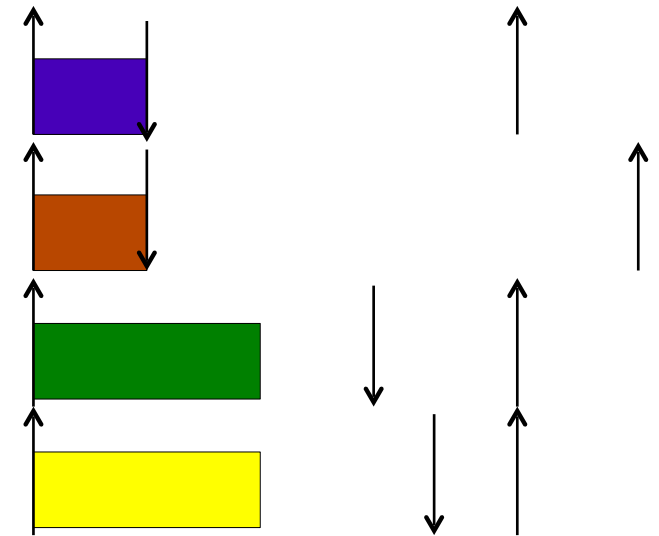
- threads may migrate to other CPUs
- scheduler picks thread from global ready queue
- accesses to ready queue must be synchronized

Partitioned:

- assign threads to processors
- scheduler picks threads from local (per CPU) ready queue
- no synchronization overhead for accessing the ready queue

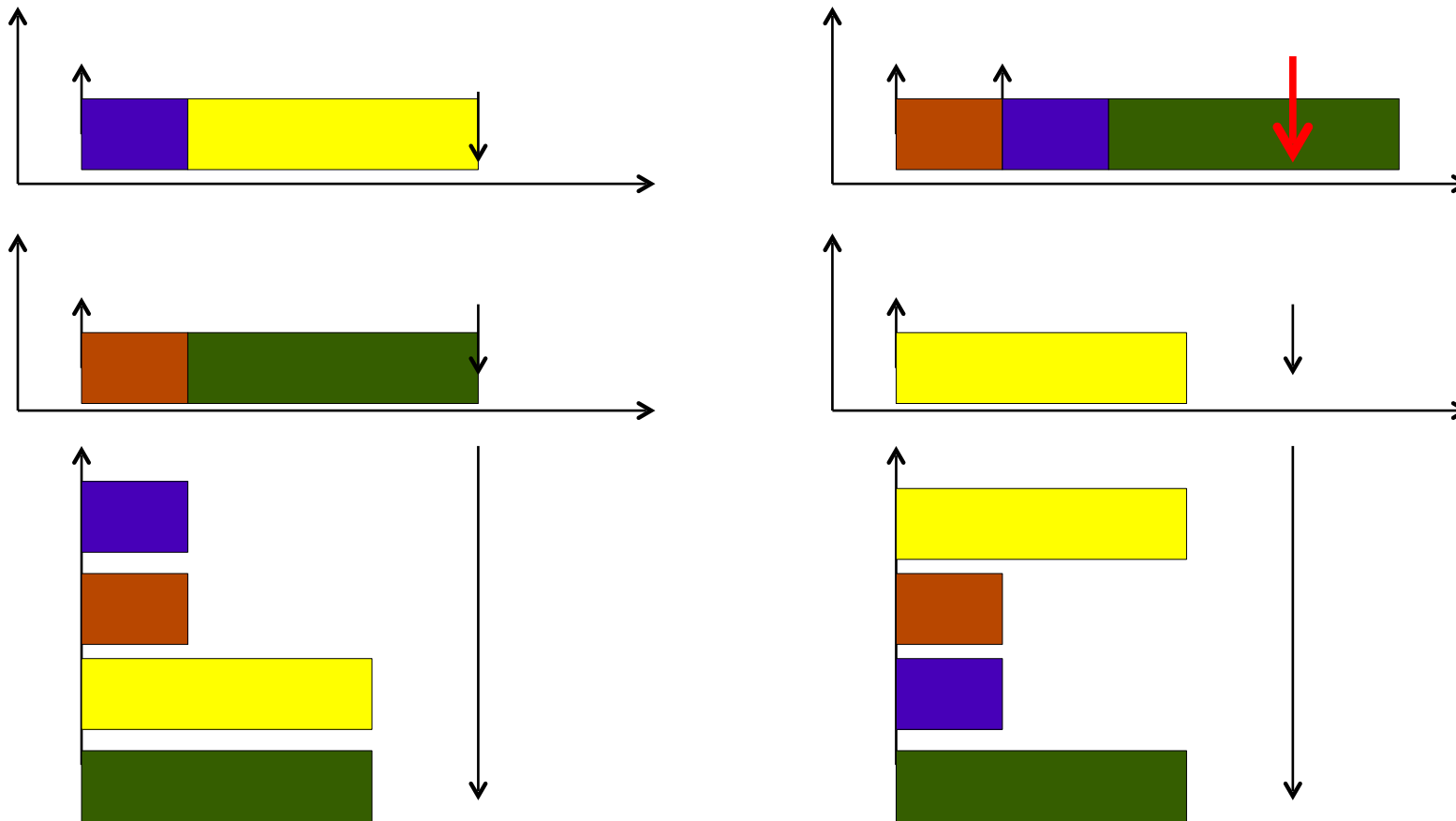
Anomalies

- Simultaneous Release is not Critical Instance [Lauzac '98]
 - longer response time in second period



Anomalies

- Response time (of green) depends not only on set of higher prioritized tasks but also on their relative priority ordering



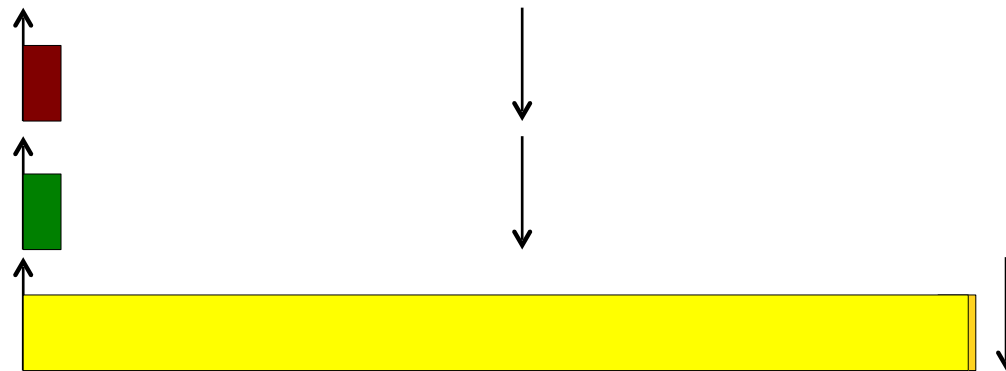
Sustainability [Baruah '06]

- A schedulable workload remains schedulable if we
 - decrease the execution time of a task (predictability)
 - otherwise, WCET won't work as admission criterion
 - increase the minimal interrelease time (period) of a task
 - otherwise, more frequent recurrence is no safe approximation
 - increase the relative deadline of a task
 - otherwise, earlier deadline is no safe approximation
- G-FTP + G-EDF are not sustainable if $\#CPUs > 1$
- all preemptive FJP / FTP algorithms are predictable


Fixed Job Priority Fixed Task Priority

Dhall Effect

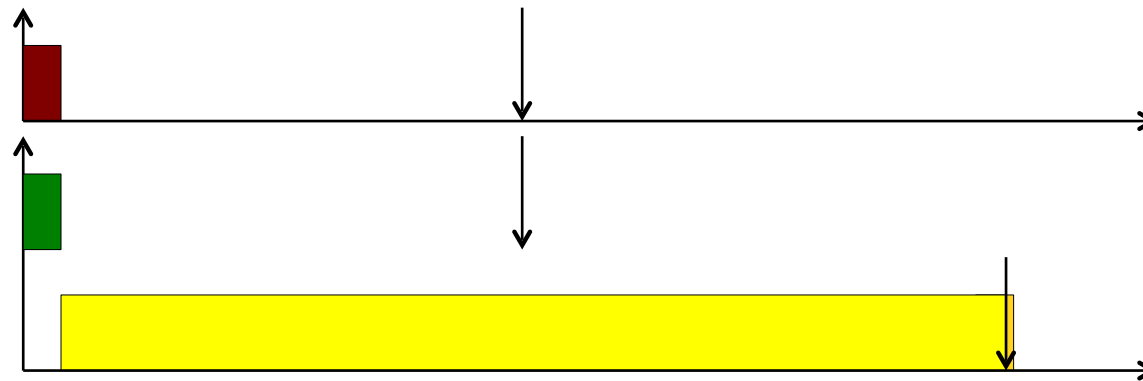
- The utilization bound of Global EDF is as low as $U_{\text{EDF}} = 1 + \epsilon$
 - m tasks with short periods and infinitesimal low U_i (e.g., $U_i = \epsilon$)
 - 1 task with larger period and U_j close to 1 (e.g., $U_j > (2 - \epsilon) / 2$)



- Dhall Effect does not manifest if $U_i < 41\%$

Dhall Effect

- The utilization bound of Global EDF is as low as $U_{\text{EDF}} = 1 + \epsilon$
 - m tasks with short periods and infinitesimal low U_i (e.g., $U_i = \epsilon$)
 - 1 task with larger period and U_j close to 1 (e.g., $U_j > (2 - \epsilon) / 2$)



- Dhall Effect does not manifest if $U_i < 41\%$

some Impossibility Results

- [Hong '88]
 - No optimal online MP scheduling algorithm for arbitrary collections of jobs, unless all jobs have the same relative deadline.
- [Dertouzos '89]
 - Even if execution times are known precisely, clairvoyance for job arrivals is necessary for optimality.
- [Fisher '07]
 - No optimal online algorithm for sporadic tasksets with constrained or arbitrary deadlines.

Partitioned Scheduling

Partitioned

dyn job prio. /
partitioned

dyn job prio. /
task level migration

dyn. job prio. /
job level migration

fixed job prio. /
partitioned

fixed job prio. /
task level migration

fixed job prio. /
job level migration

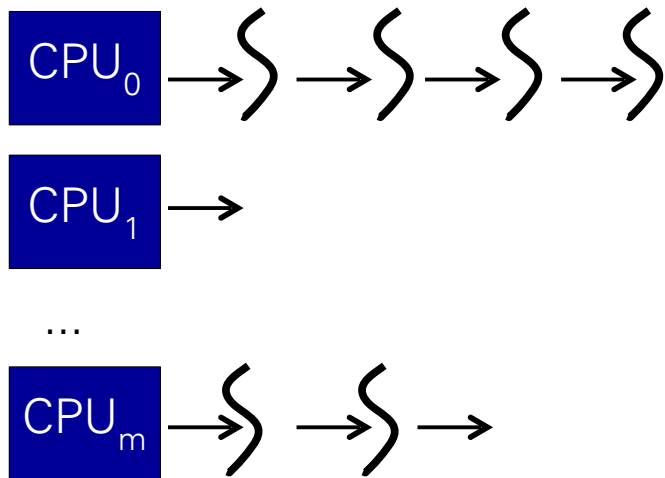
fixed task prio. /
partitioned

fixed task prio. /
task level migration

fixed task prio. /
job level migration

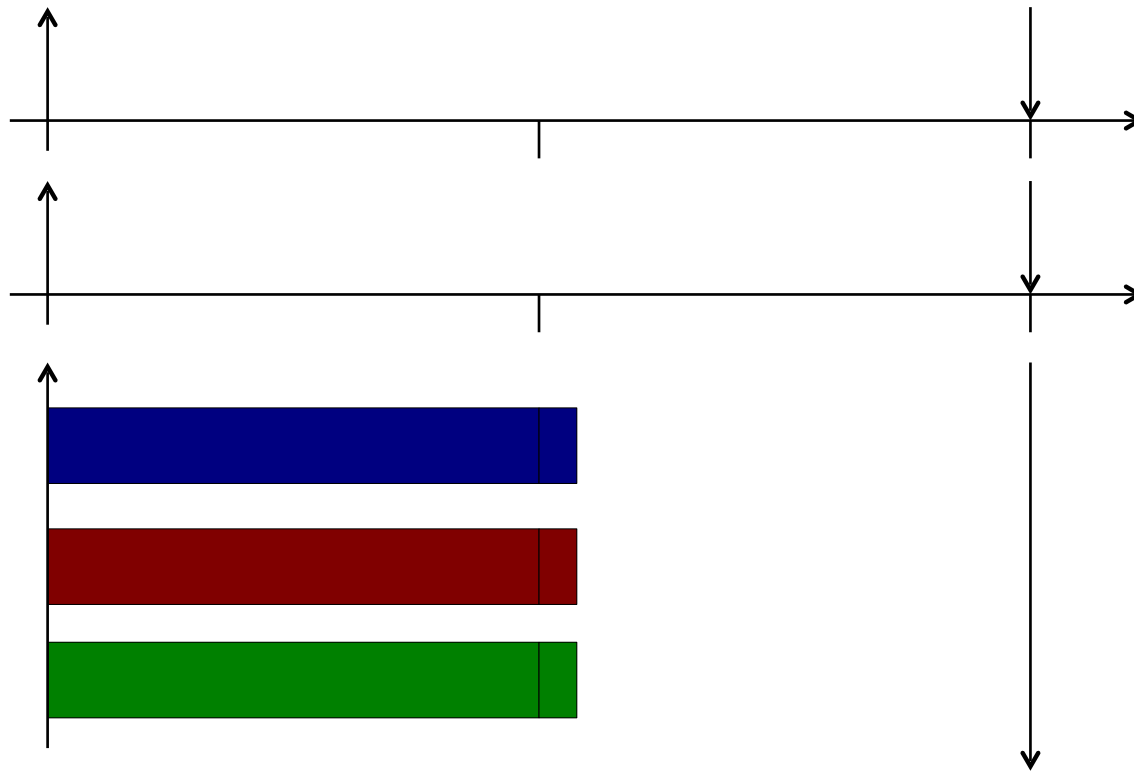
Partitioned Scheduling

- Split workload by allocating tasks to CPUs
- Run allocated task with UP scheduling algorithm
 - reap benefit of well known UP results
 - optimal task allocation is NP complete:
 - pack n tasks with density d_i on m CPUs with capacity $d_{\max} = 1$
 - Bin-packing



Partitioned Scheduling

- Utilization bound for implicit deadline workloads [Anderson '01]

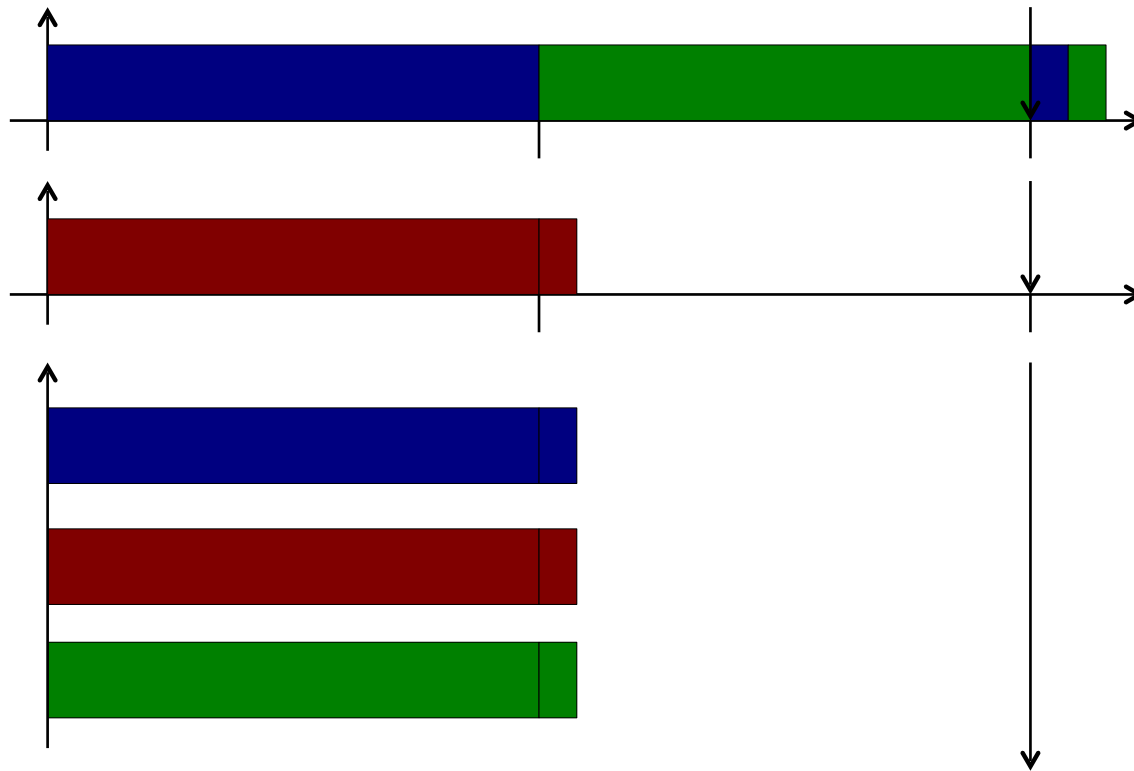


$$U_{opt} = \frac{m + 1}{2}$$

No partitioning scheduling algorithm can produce a feasible schedule of $m+1$ tasks with execution time $1+\epsilon$ and period of 2 on m processors

Partitioned Scheduling

- Utilization bound for implicit deadline workloads [Anderson '01]

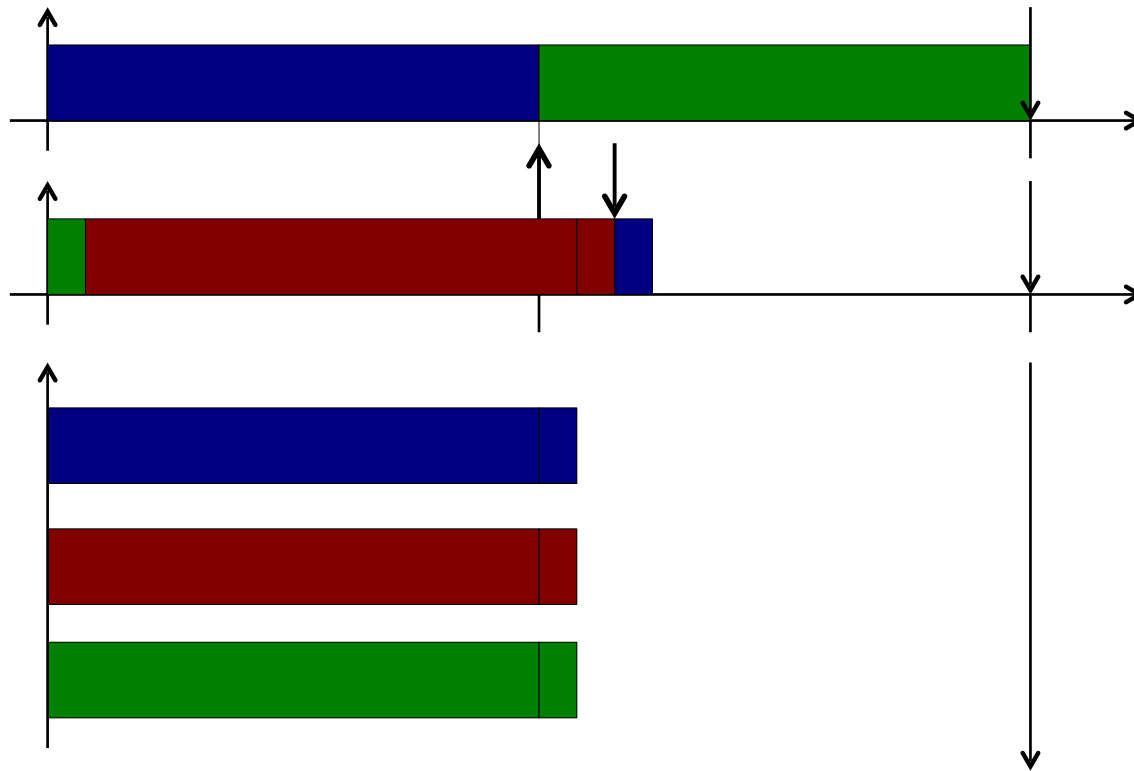


$$U_{opt} = \frac{m+1}{2}$$

No partitioning scheduling algorithm can produce a feasible schedule of $m+1$ tasks with execution time $1+\epsilon$ and period of 2 on m processors

Partitioned Scheduling

- Utilization bound for implicit deadline workloads [Anderson '01]



$$U_{opt} = \frac{m+1}{2}$$






No partitioning scheduling algorithm can produce a feasible schedule of $m+1$ tasks with execution time $1+\epsilon$ and period of 2 on m processors

Easy if blue and green can migrate to CPU_2

PDMS-HPTS-DS [Lakshmanan '09]

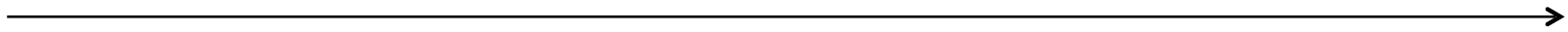
- Can we improve on Anderson's Utilization Bound?
 - by allowing a few jobs to migrate
- **PDMS** Partitioned Deadline Monotonic Scheduling
- **HPTS** Split Highest Priority Task
- **DS** Allocate according to Highest Density First

PDMS-HPTS-DS

$\tau_1 = (4,3,1)$		↓	↑	$u_1 = 0.25 \quad \delta_1 = 1/3 = 0.33$
$\tau_2 = (6,2,2)$			↑	$u_2 = 0.33 \quad \delta_2 = 1$
$\tau_3 = (4,4,1)$		↓		$u_3 = 0.25 \quad \delta_3 = 1/4 = 0.25$
$\tau_4 = (6,4,2)$		↓	↑	$u_4 = 0.33 \quad \delta_4 = 1/2 = 0.5$
$\tau_5 = (6,5,1)$		↓	↑	$u_5 = 0.16 \quad \delta_5 = 1/5 = 0.2$
				<hr/>
				$u_{\text{sum}} = 1.33 \Rightarrow u_{\text{sum}} / 2 = 0.66\%$

CPU₀

CPU₁



PDMS-HPTS-DS

$\tau_1 = (4,3,1)$  $u_1 = 0.25 \quad \delta_1 = 1/3 = 0.33$

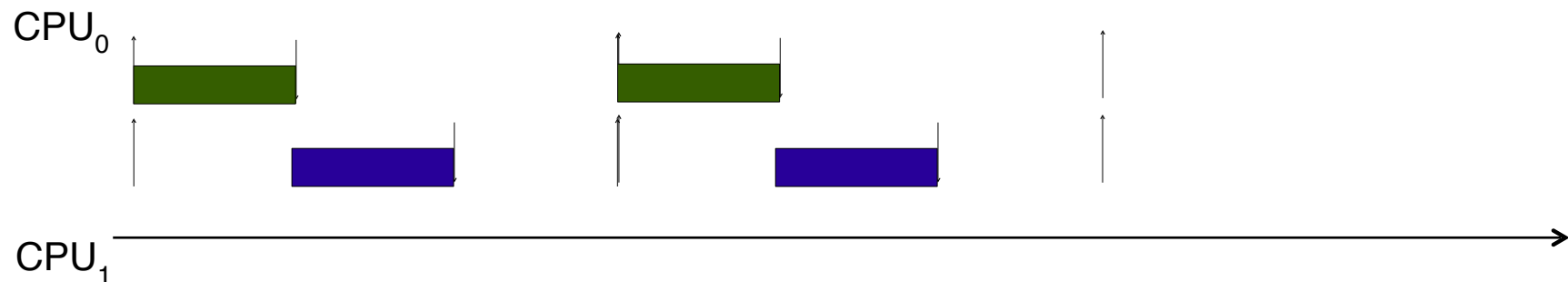
$\tau_2 = (6,2,2)$ $u_2 = 0.33 \quad \delta_2 = 1$

$\tau_3 = (4,4,1)$  $u_3 = 0.25 \quad \delta_3 = 1/4 = 0.25$

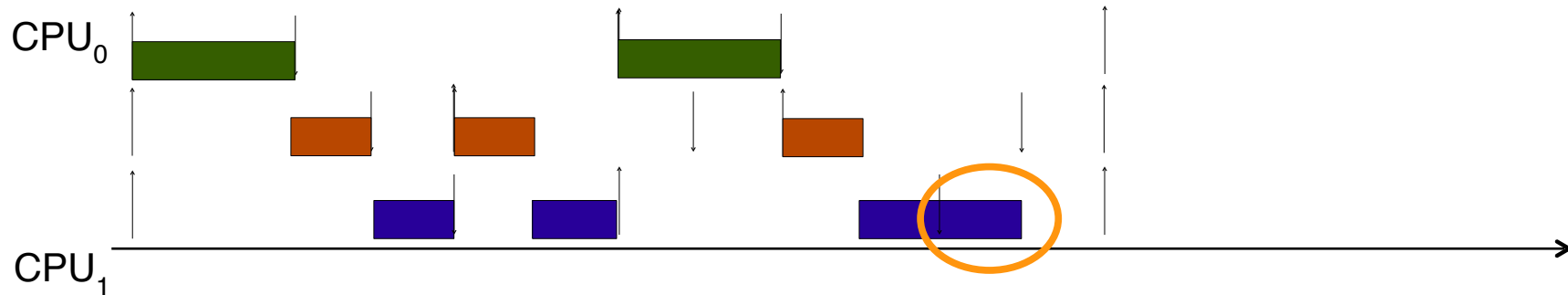
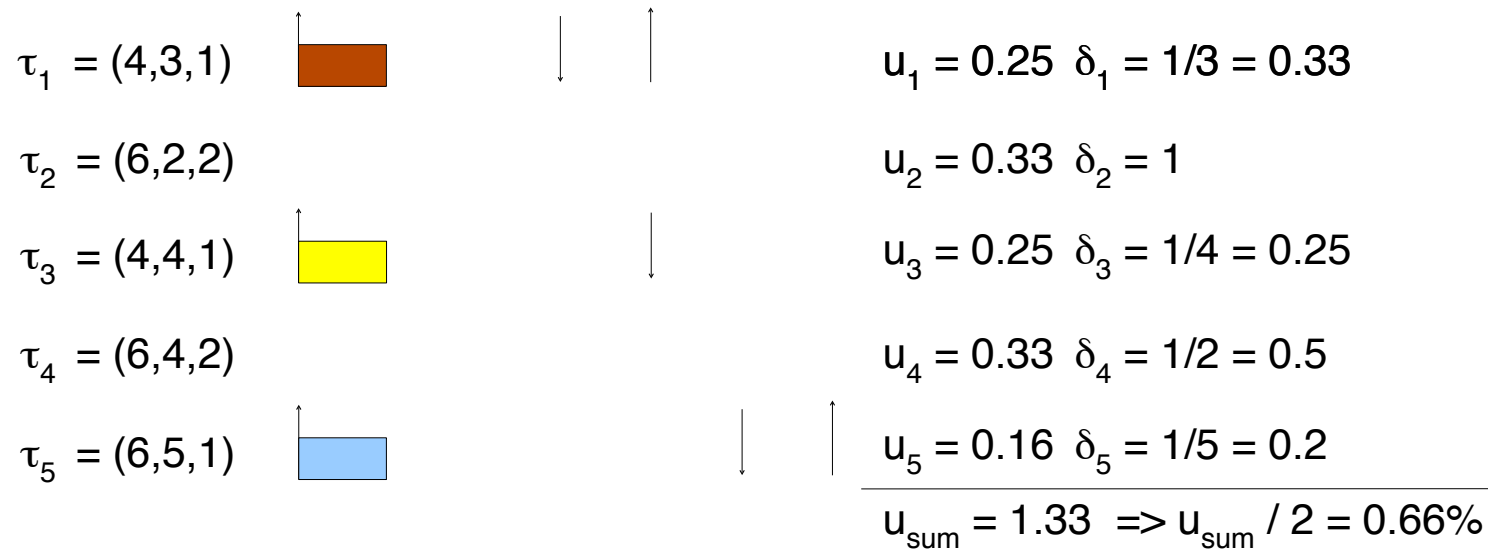
$\tau_4 = (6,4,2)$ $u_4 = 0.33 \quad \delta_4 = 1/2 = 0.5$

$\tau_5 = (6,5,1)$  $u_5 = 0.16 \quad \delta_5 = 1/5 = 0.2$

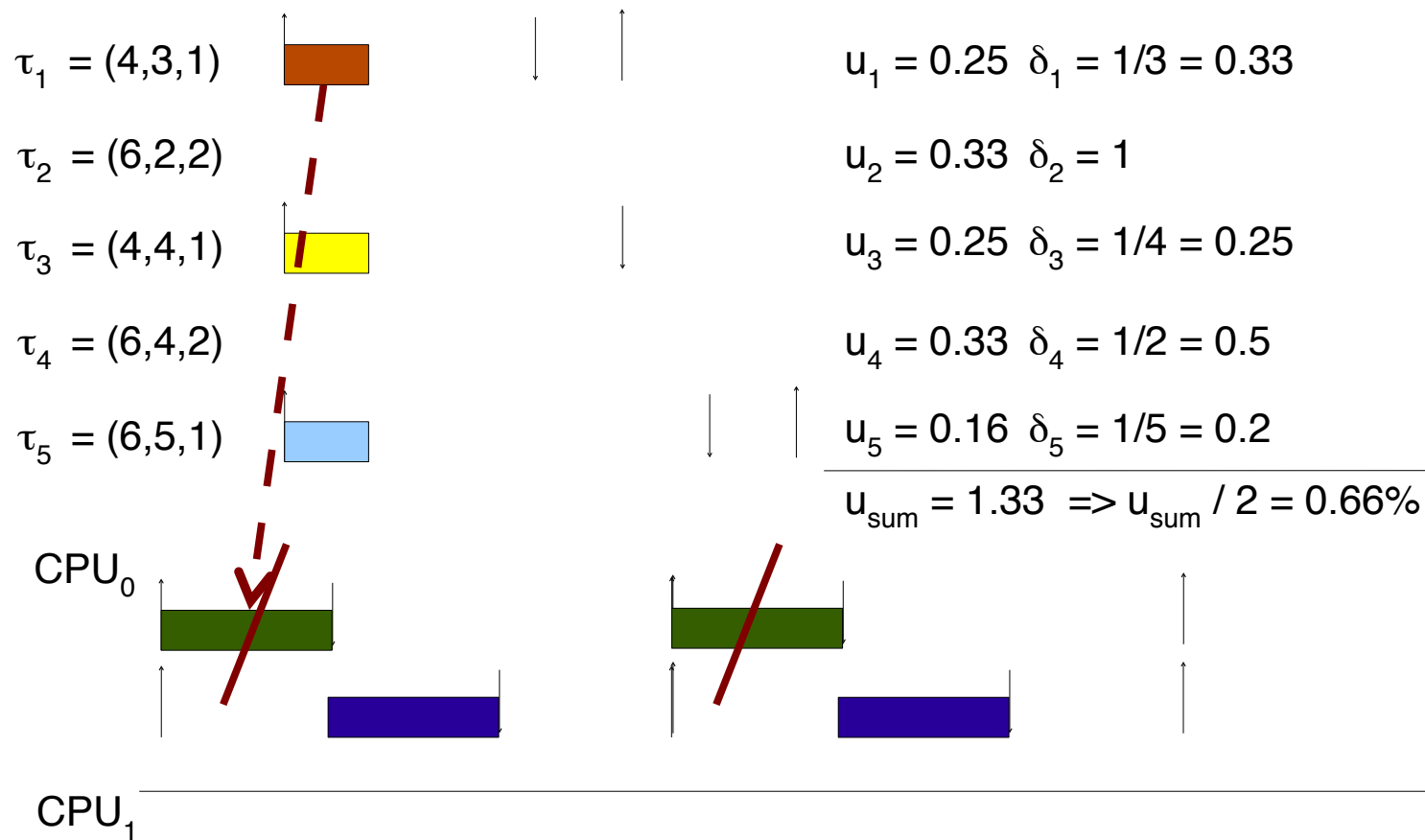
$$u_{\text{sum}} = 1.33 \Rightarrow u_{\text{sum}} / 2 = 0.66\%$$



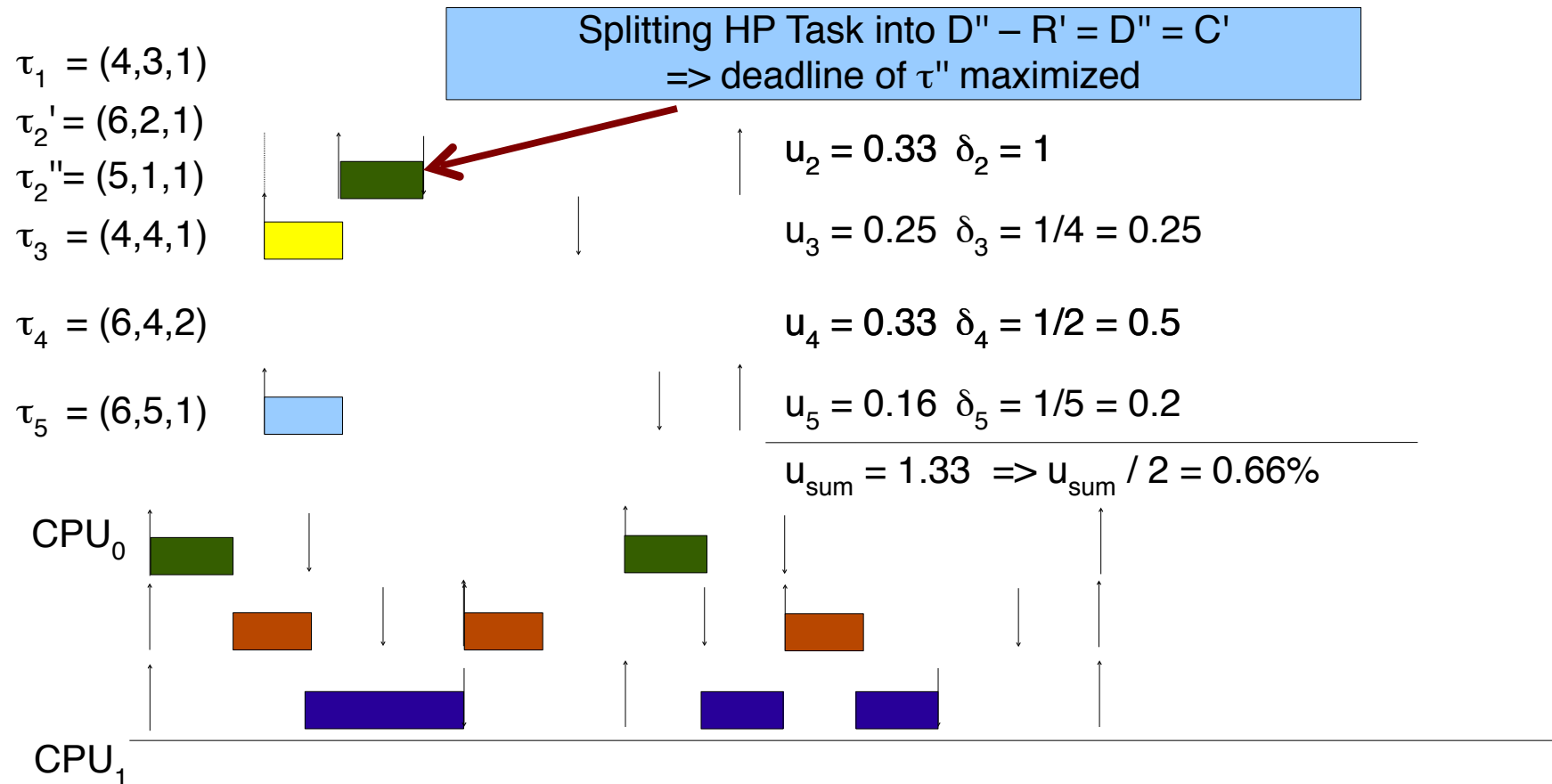
PDMS-HPTS-DS



PDMS-HPTS-DS



PDMS-HPTS-DS



PDMS-HPTS-DS

$$\tau_1 = (4, 3, 1)$$

$$\tau_2' = (6, 2, 1)$$

$$\tau_2'' = (5, 1, 1)$$

$$\tau_3 = (4, 4, 1)$$

$$\tau_4 = (6, 4, 2)$$

$$\tau_5 = (6, 5, 1)$$

$$u_1 = 0.25 \quad \delta_1 = 1/3 = 0.33$$

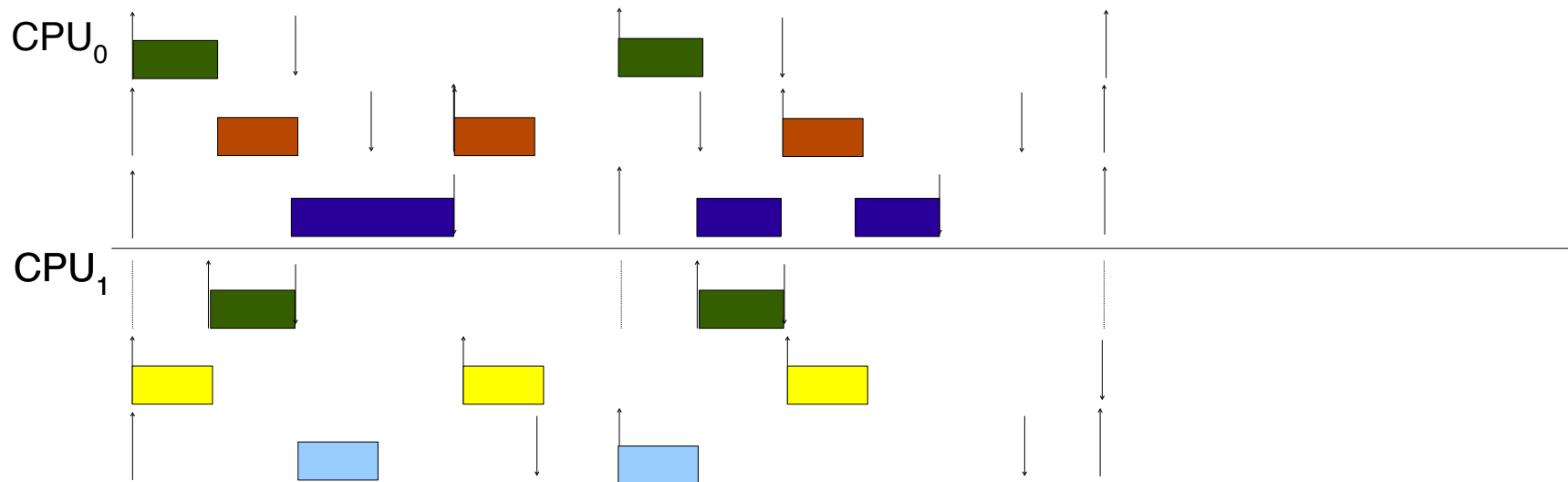
$$u_2 = 0.33 \quad \delta_2 = 1$$

$$u_3 = 0.25 \quad \delta_3 = 1/4 = 0.25$$

$$u_4 = 0.33 \quad \delta_4 = 1/2 = 0.5$$

$$u_5 = 0.16 \quad \delta_5 = 1/5 = 0.2$$

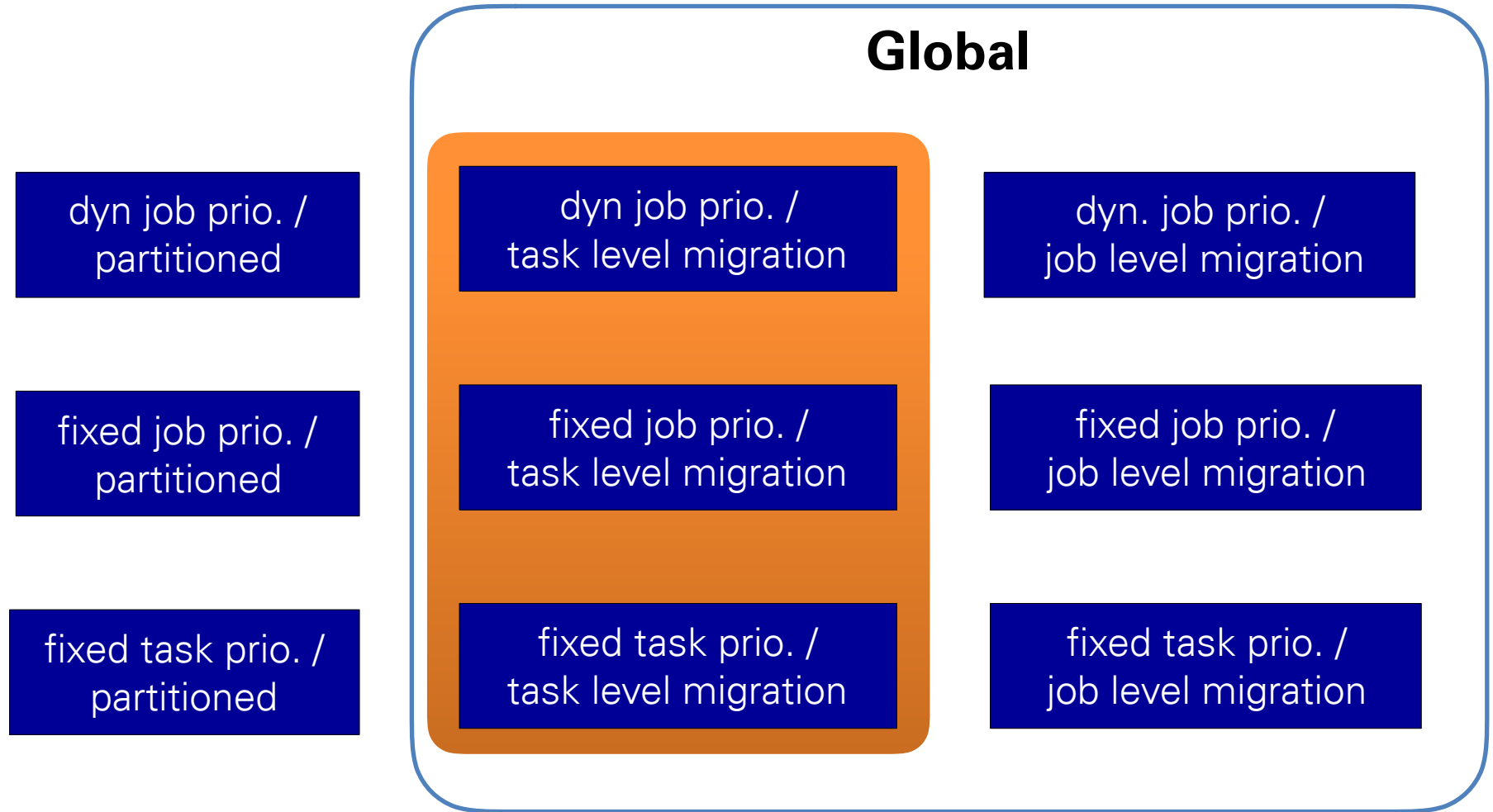
$$u_{\text{sum}} = 1.33 \Rightarrow u_{\text{sum}} / 2 = 0.66\%$$



PDMS-HPTS-DS

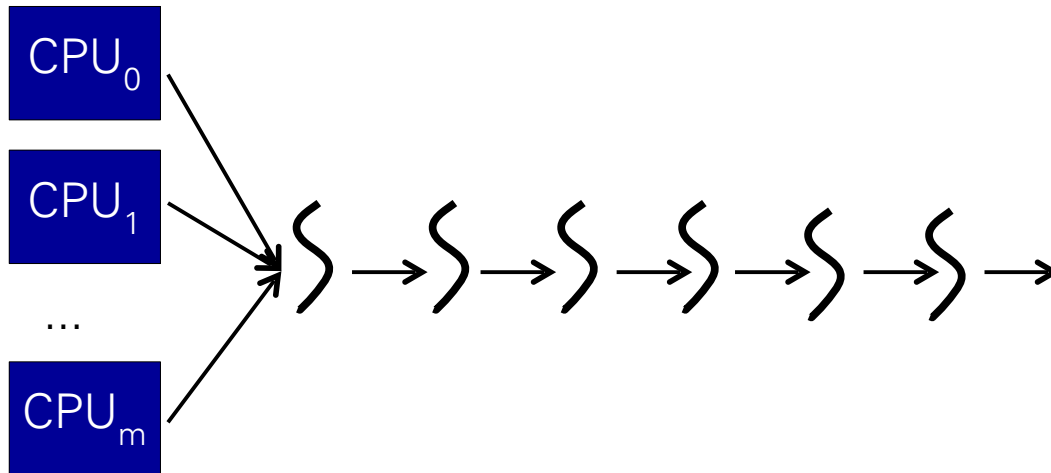
- $U_{\text{PDMS-HPTS-DS}} = 69.3\%$ if all tasks have a utilization $U_i < 41\%$

Global Scheduling



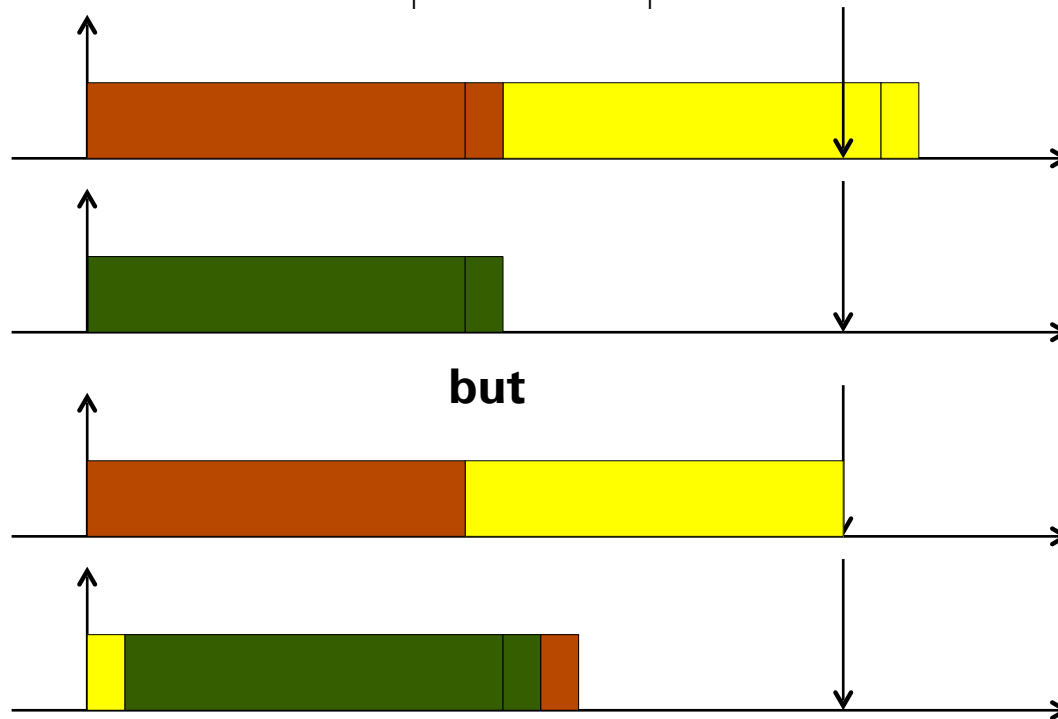
Global Scheduling

- Always pick the ready jobs of the m most “important” tasks
- A task may migrate
 - When a new job of a task is released it may receive a different CPU; once started, a job is no longer migrated
- No need for allocation / load balancing
 - Load balancing is automatic



Global Scheduling – Utilization Bound

- Utilization bound for global fixed-job priority algorithms
 - on m CPUs, G-FJP algorithms cannot schedule $m+1$ tasks with $C_i = 1 + e$, $P_i = 2$ ($e \rightarrow 0$)



- $$U_{\text{OPT}} = \lim_{e \rightarrow 0} \frac{(m+1)(1+e)}{2} = \frac{m+1}{2}$$

Global Scheduling - Job Level Migration

dyn job prio. /
partitioned

dyn job prio. /
task level migration

dyn. job prio. /
job level migration

fixed job prio. /
partitioned

fixed job prio. /
task level migration

fixed job prio. /
job level migration

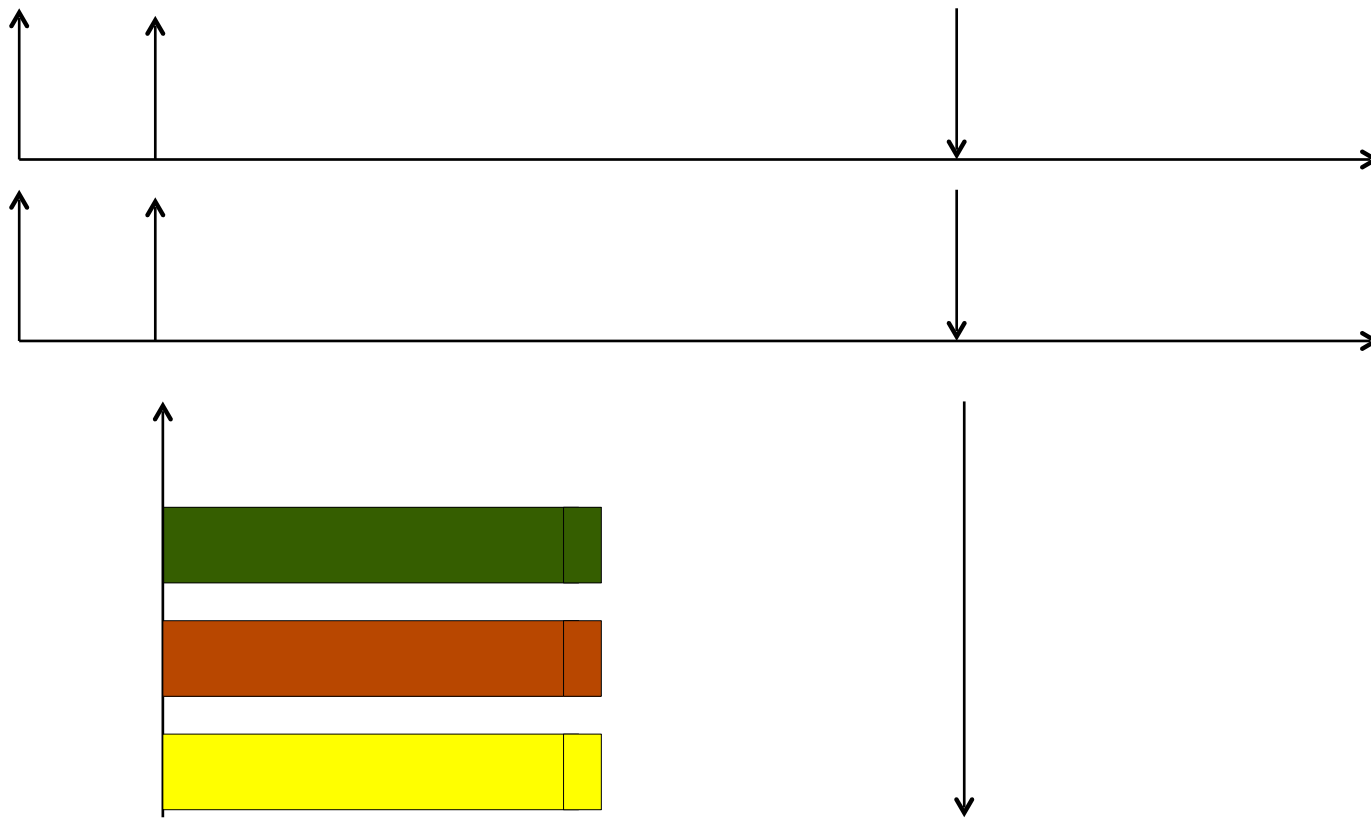
fixed task prio. /
partitioned

fixed task prio. /
task level migration

fixed task prio. /
job level migration

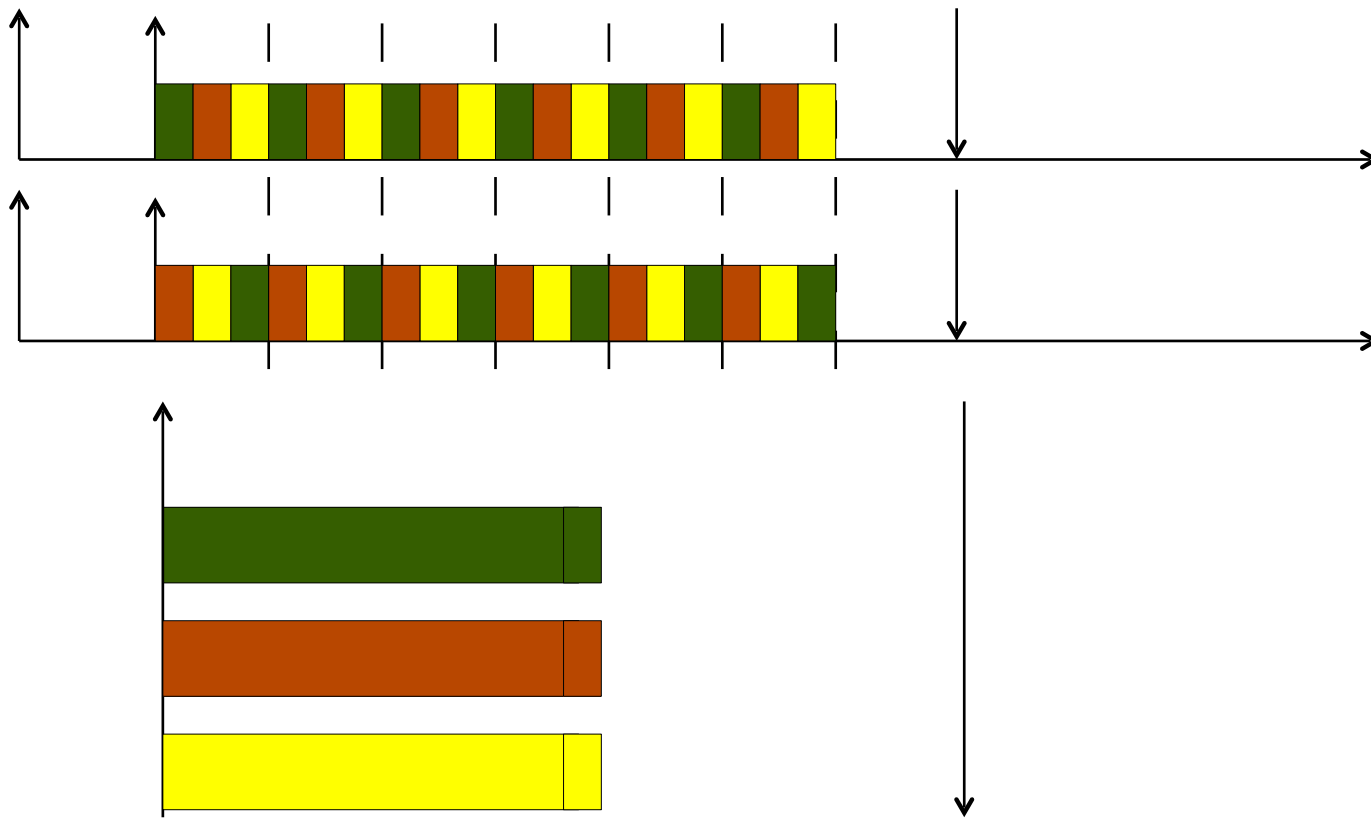
PFAIR [Baruah '96]

- Divide timeline into equal length quanta
- At each quanta of length t , allocate tasks to processors such that the accumulated processor time is either $\lfloor tu_i \rfloor$ or $\lceil tu_i \rceil$



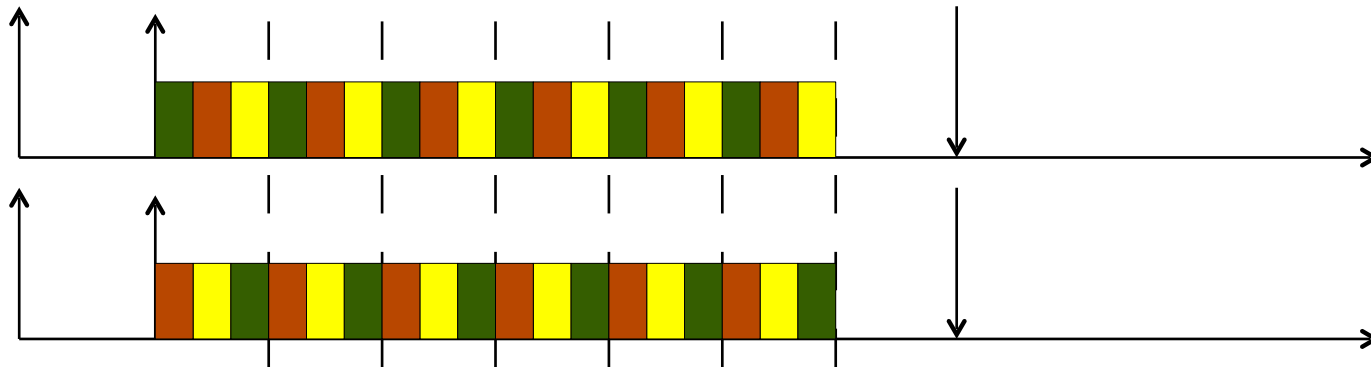
PFAIR [Baruah '96]

- Divide timeline into equal length quanta
- At each quanta of length t , allocate tasks to processors such that the accumulated processor time is either $\lceil tu_i \rceil$ or $\lfloor tu_i \rfloor$



PFAIR [Baruah '96]

- Divide timeline into equal length quanta
- At each quanta of length t , allocate tasks to processors such that the accumulated processor time is either $\lceil tu_i \rceil$ or $\lfloor tu_i \rfloor$



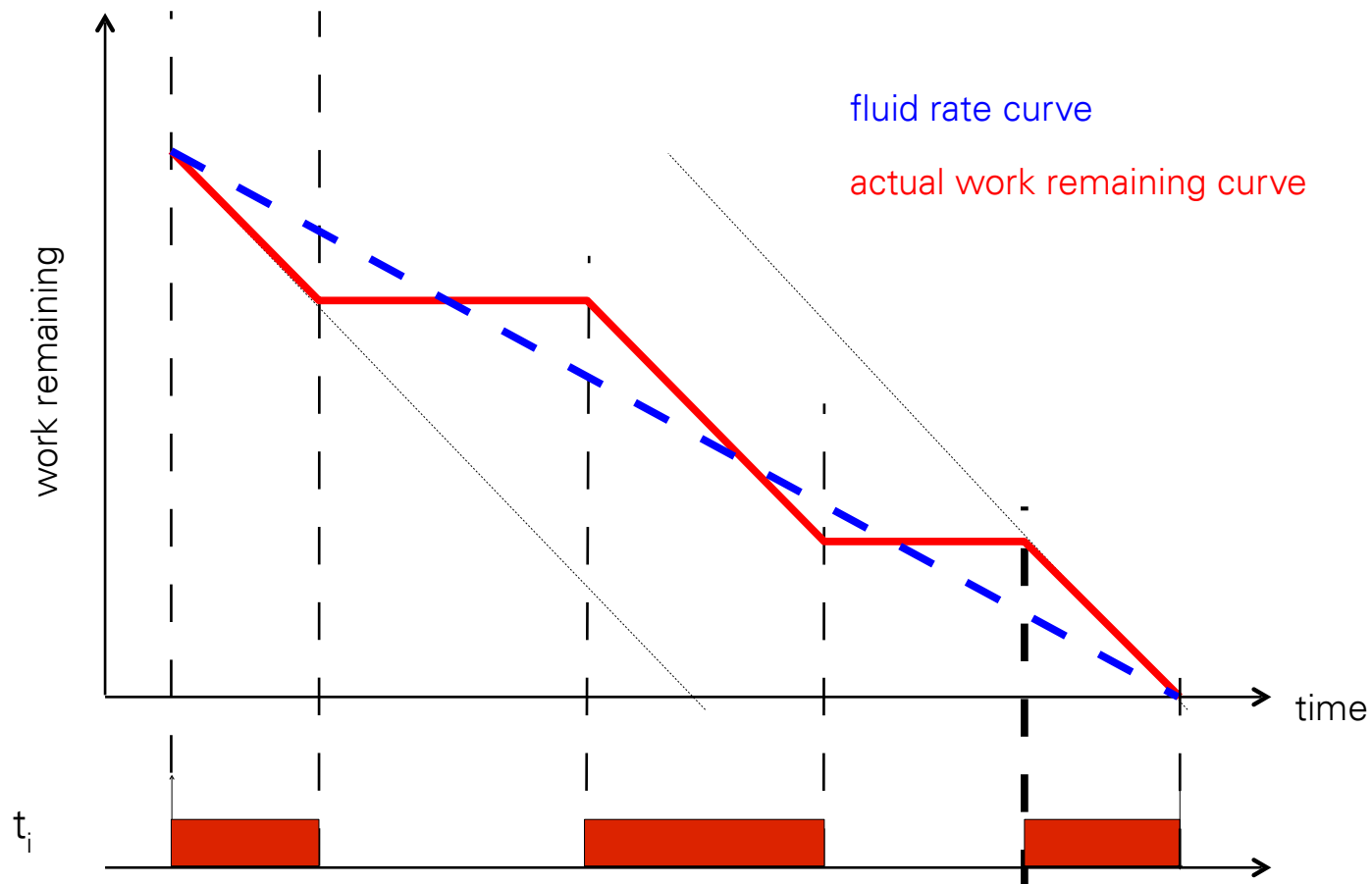
- PFAIR is optimal for periodic implicit deadline tasksets: $U_{OPT} = m$
- Very high preemption and migration costs

DP-Fair [Brandt '10]

- DP-Fair
 - Optimal scheduler for periodic implicit deadline tasksets with a minimal number of preemptions / migrations?
 - Recall [Hong '88]:
 - No optimal MP scheduling algorithm for arbitrary tasksets if not all tasks have the same relative deadline
 - deadline partitioning
 - any task's deadline becomes a deadline for all tasks
 - always run zero laxity jobs
 - laxity = time to deadline – remaining execution time
 - zero laxity => job may miss its deadline if it is not run
 - jobs that twine themselves around the *fluid rate curve* are somehow in good shape

DP-Fair [Brandt '10]

- Fluid Rate Curve



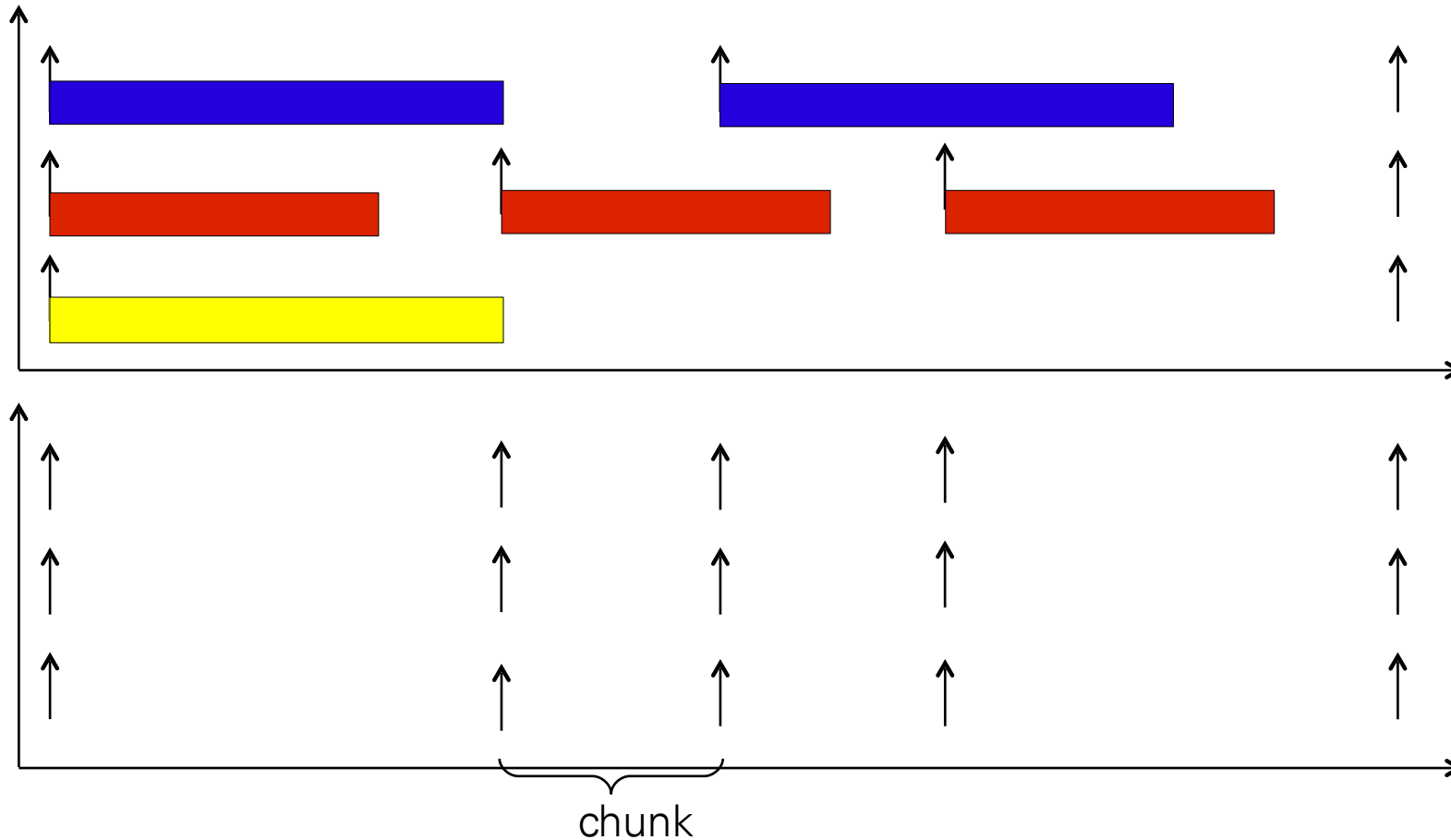
zero laxity event: no more time to run others

DP-Fair [Brandt '10]

- DP-Fair:
 - a family of optimal, deadline partitioning scheduling algorithms
 - Split timeline into chunks according to job deadlines
 - Allocate work to a chunk proportional to U_i
 - local execution time: $C_{i,j}^l = (t_{j+1} - t_j) U_i$
 - Rule 1: always run a job with zero local laxity
 - jobs with remaining local execution time = time to end of chunk
 - Rule 2: never run a job with no remaining local work
 - Rule 3: split up idle time proportional to length of chunk
 - allocate at most $(m - U_{\text{sum}}) (t_{j+1} - t_j)$ idle time to chunk j

DP-Fair [Brandt '10]

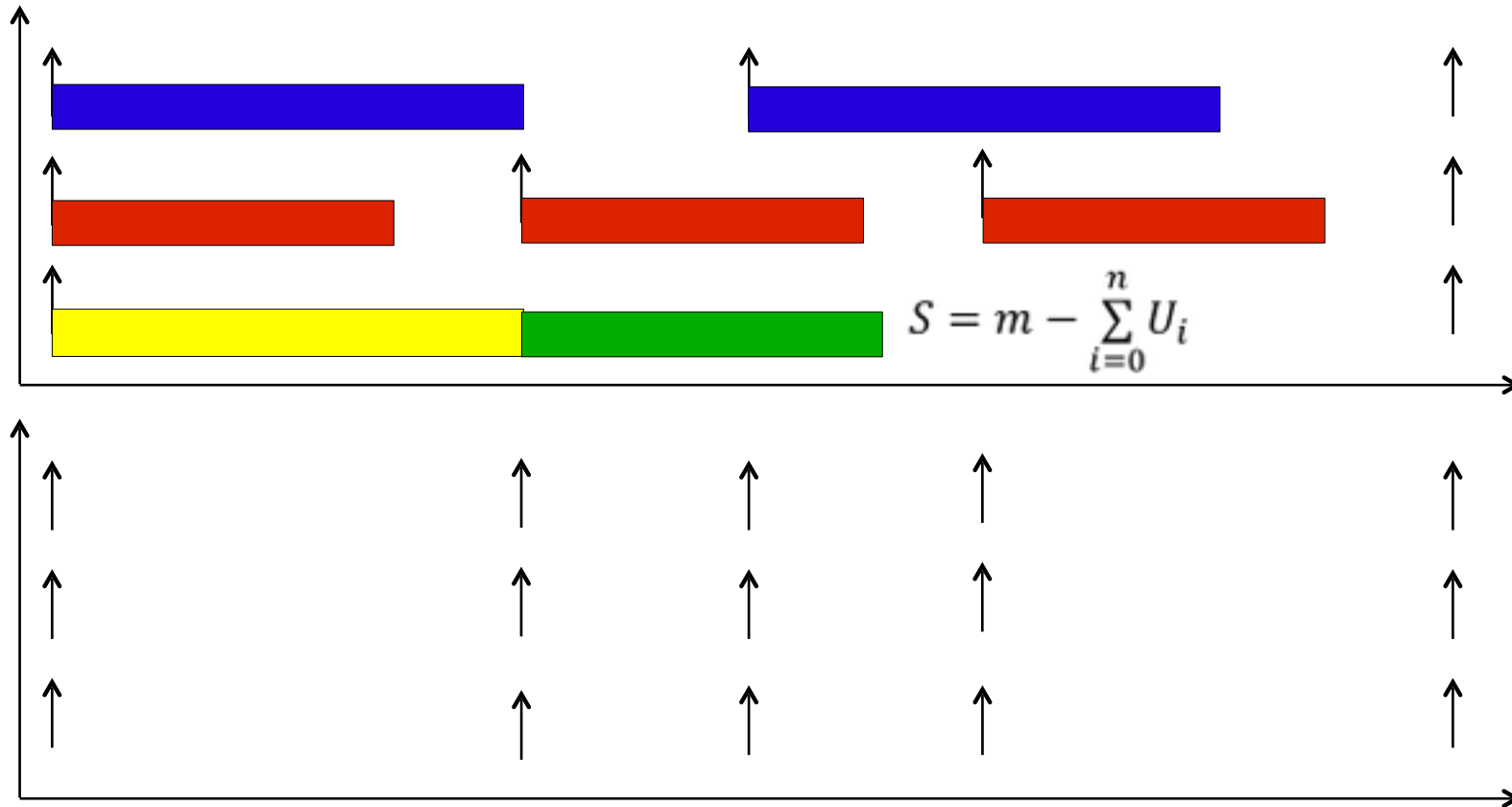
Deadline partitioning



Introduce additional releases / deadlines for all jobs whenever there is such an event for one job in the original schedule => chunks

DP-Fair [Brandt '10]

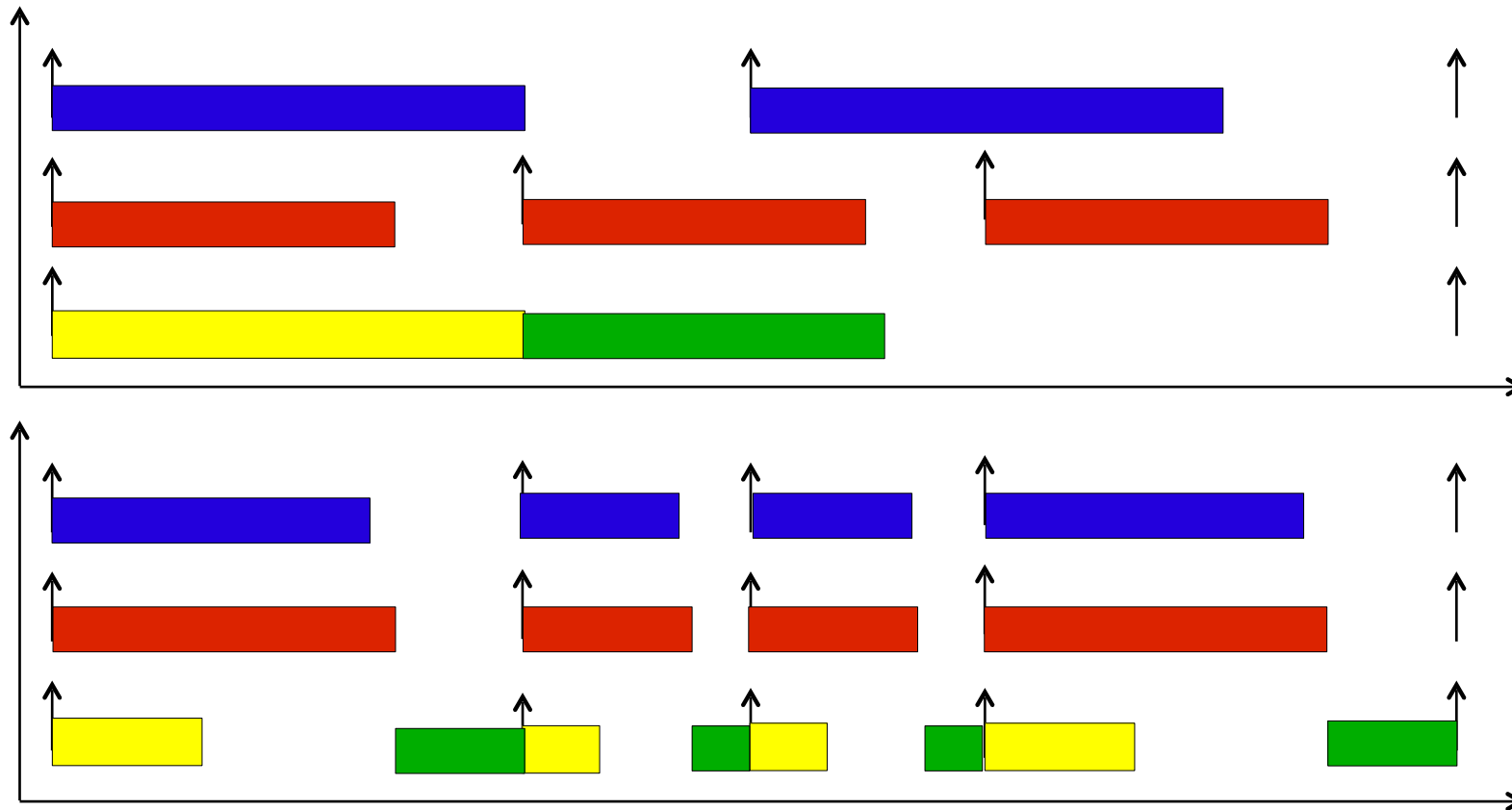
Idle time



Treat idle time as just another job to schedule.

DP-Fair [Brandt '10]

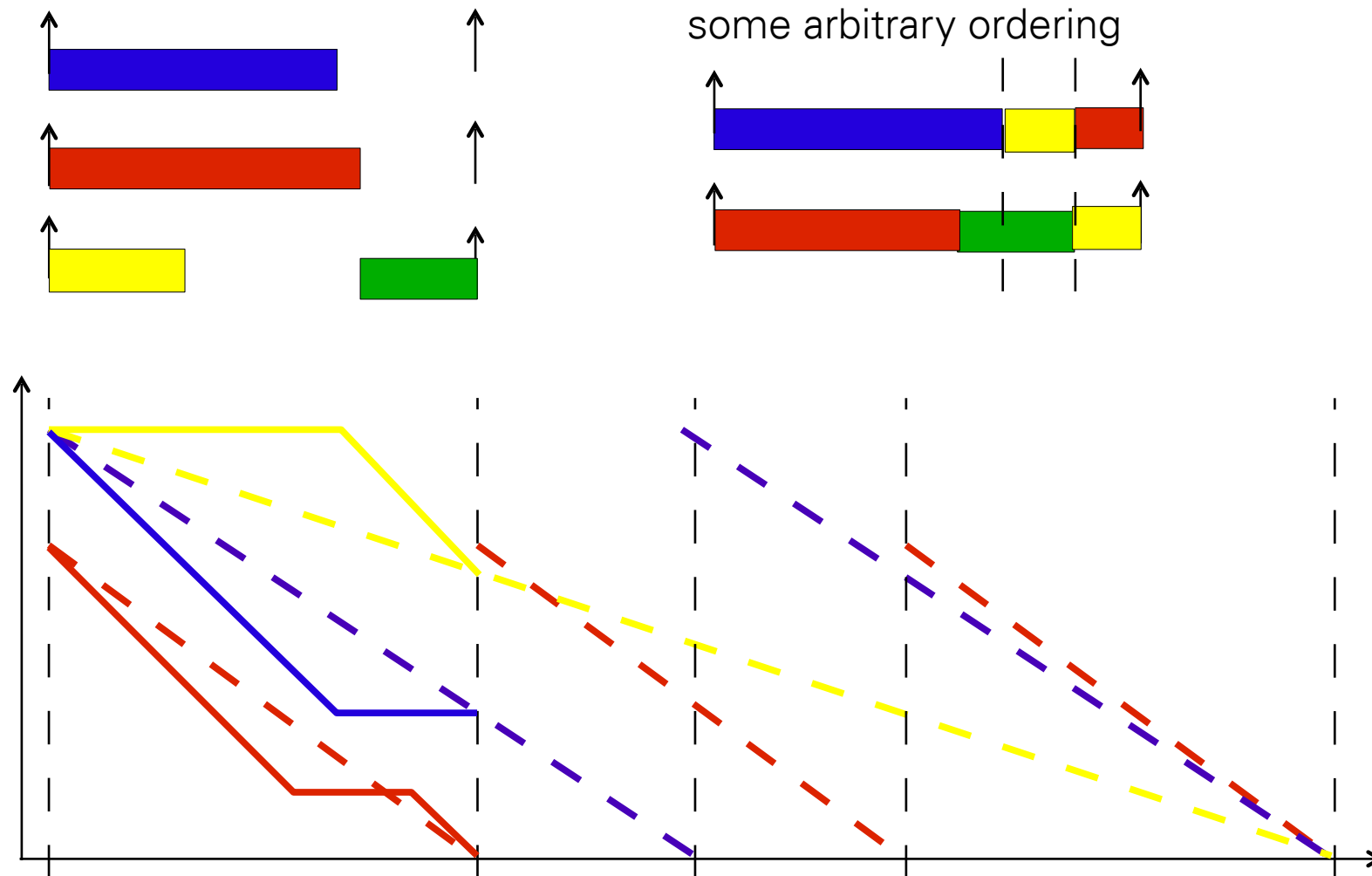
Allocate work proportional to U_i



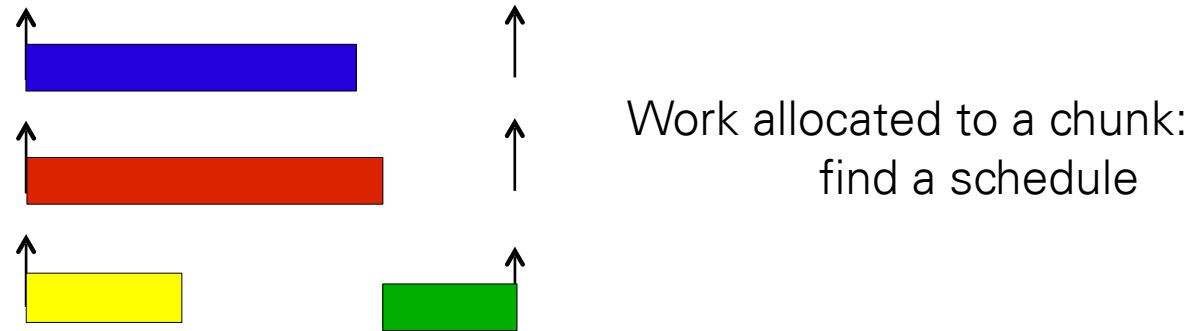
Allocate execution (and idle) time of a job proportionally to its utilization
=> amount of time that this job must run in a given chunk

DP-Fair [Brandt '10]

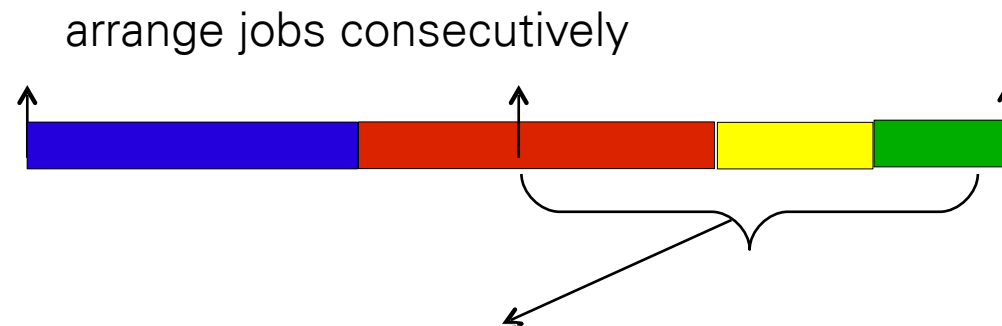
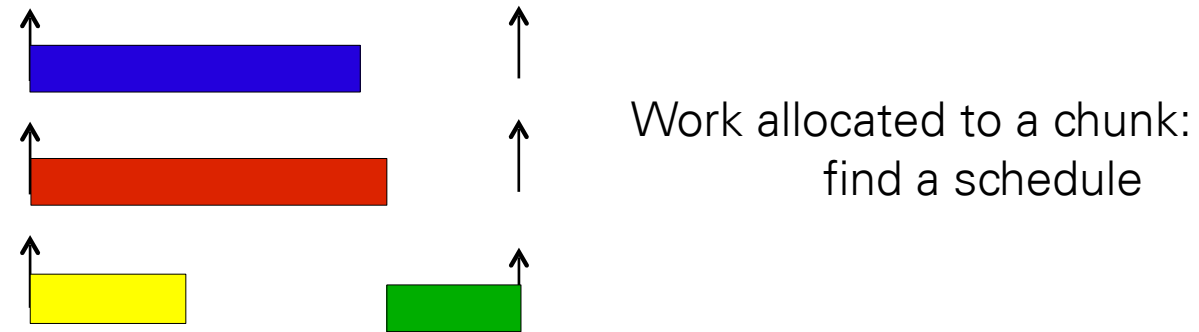
Jobs hit their fluid rate curve at the end of each chunk



DP-Wrap – a DP-Fair Scheduler

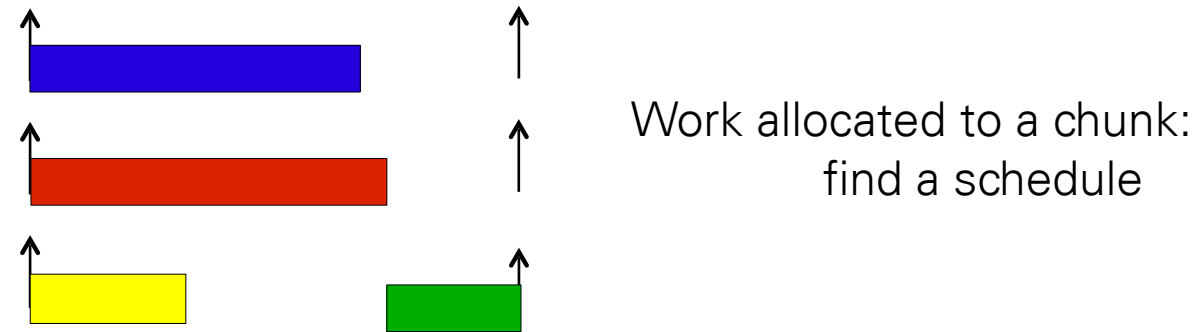


DP-Wrap – a DP-Fair Scheduler

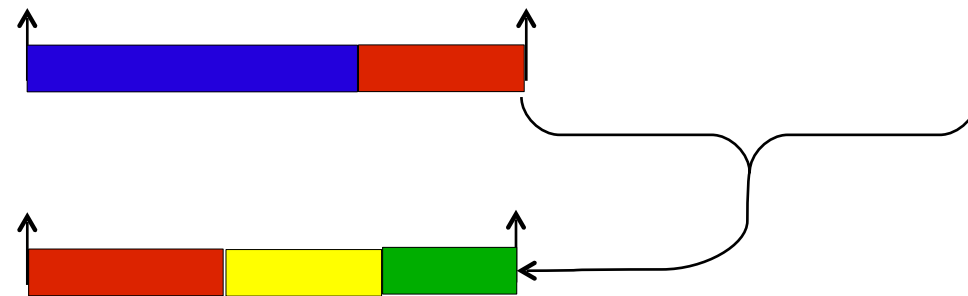


wrap around to obtain schedule for 2nd CPU, ...

DP-Wrap – a DP-Fair Scheduler



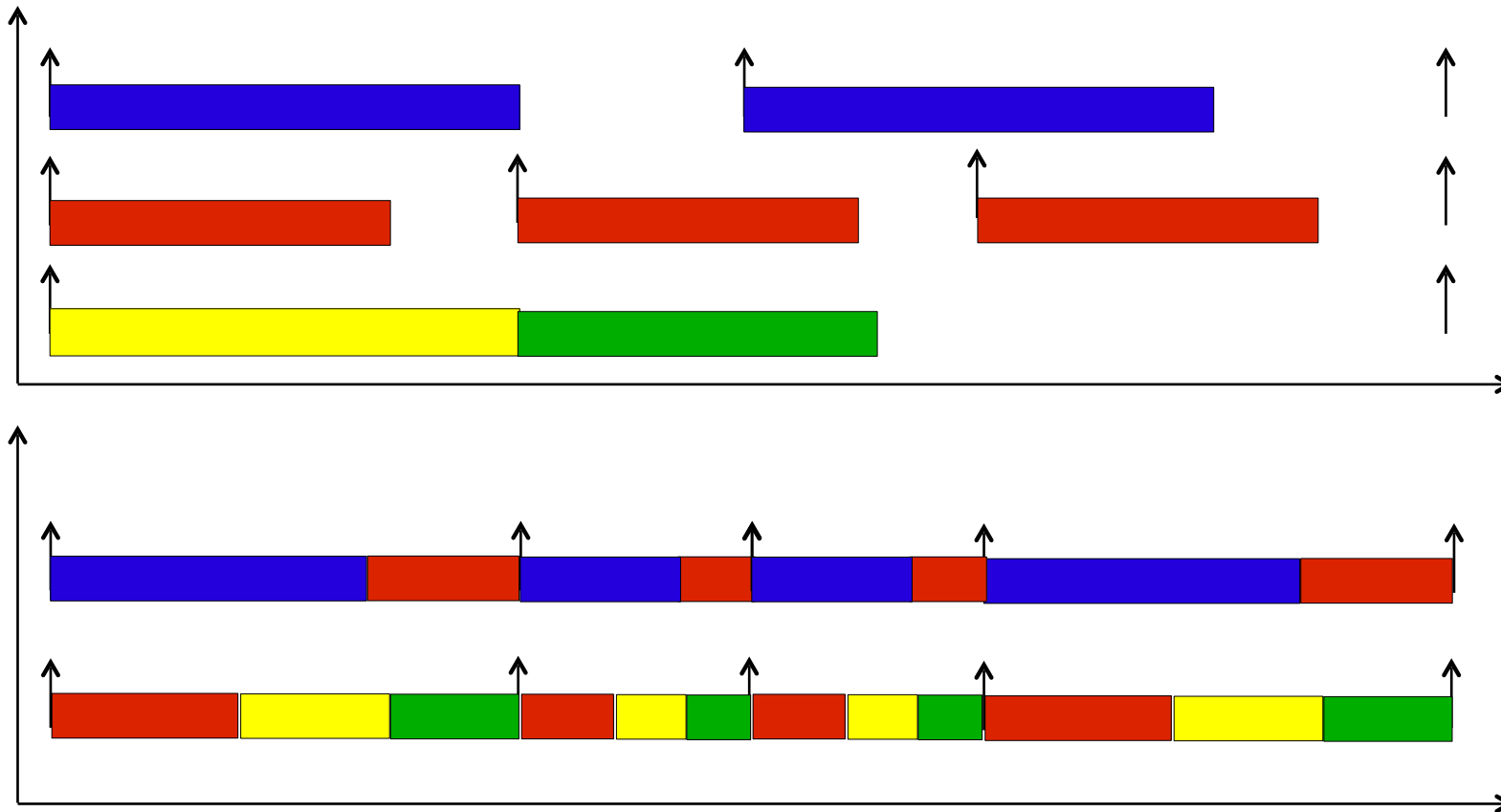
arrange jobs consecutively



wrap around to obtain schedule for 2nd CPU, ...

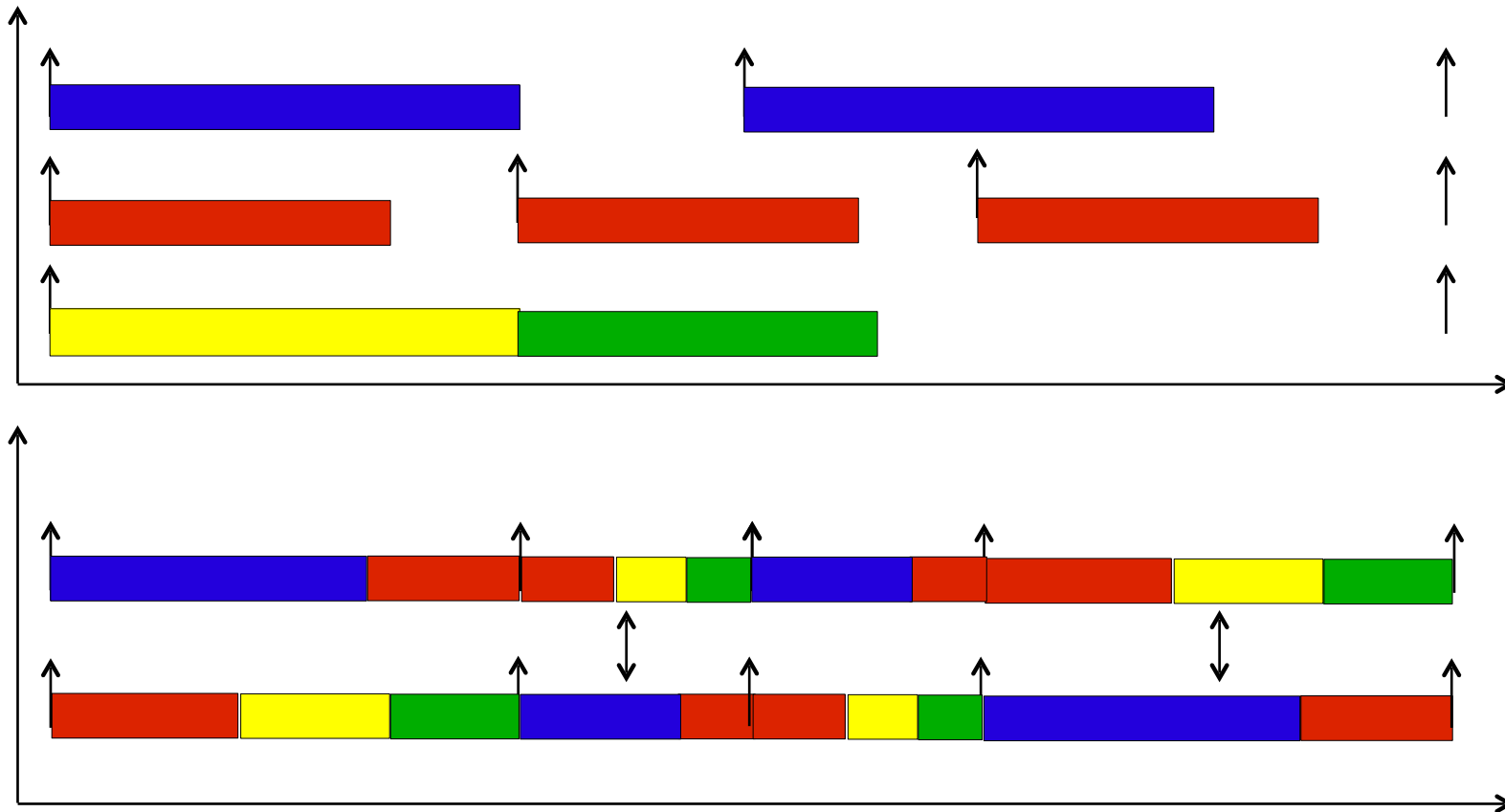
$m - 1$ migrations per chunk
 $n - 1$ context switches per chunk

DP-Wrap – a DP-Fair Scheduler



Unnecessary migration of red task at chunk boundaries
=> mirror processor assignment of every second chunk

DP-Wrap (mirrored)



Design Space of MP Scheduling

dyn job prio. /
partitioned

dyn job prio. /
task level migration

dyn. job prio. /
job level migration

fixed job prio. /
partitioned

fixed job prio. /
task level migration

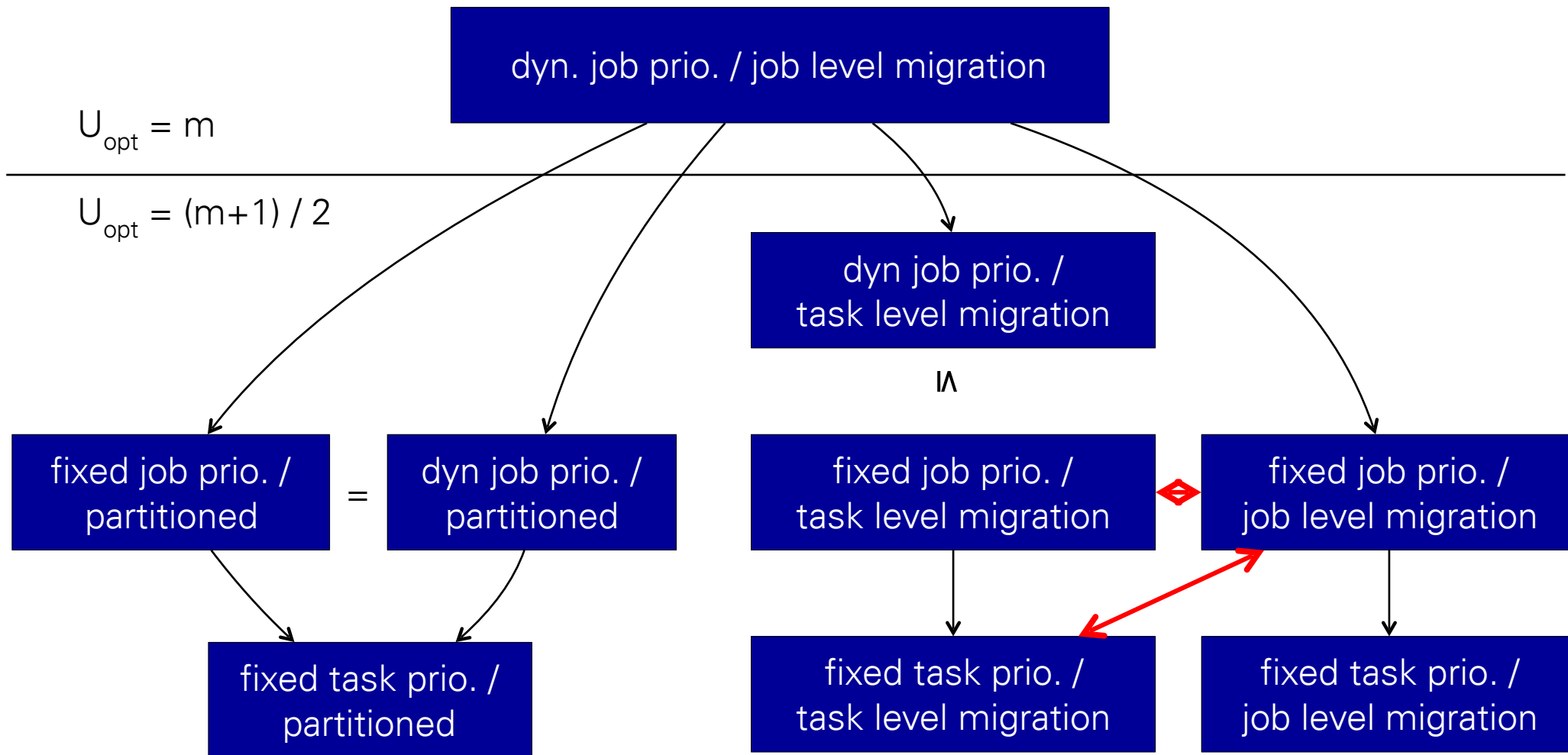
fixed job prio. /
job level migration

fixed task prio. /
partitioned

fixed task prio. /
task level migration

fixed task prio. /
job level migration

Design Space of MP Scheduling



$A \rightarrow B \Rightarrow A$ can schedule any taskset that B can schedule and more

$A \leftrightarrow B \Rightarrow$ dominance is not yet known

Outline

- Introduction
- Terminology, Notation and Assumptions
- Anomalies + Impossibility Results
- Partitioned Scheduling
- Global (Task-Lvl migration) Scheduling
 - G-FTP (e.g., G-RMS)
 - G-EDF
- Optimal MP Scheduling
- MP – Resource Access Protocols
- Open Research Issues

MP Resource Access Protocols

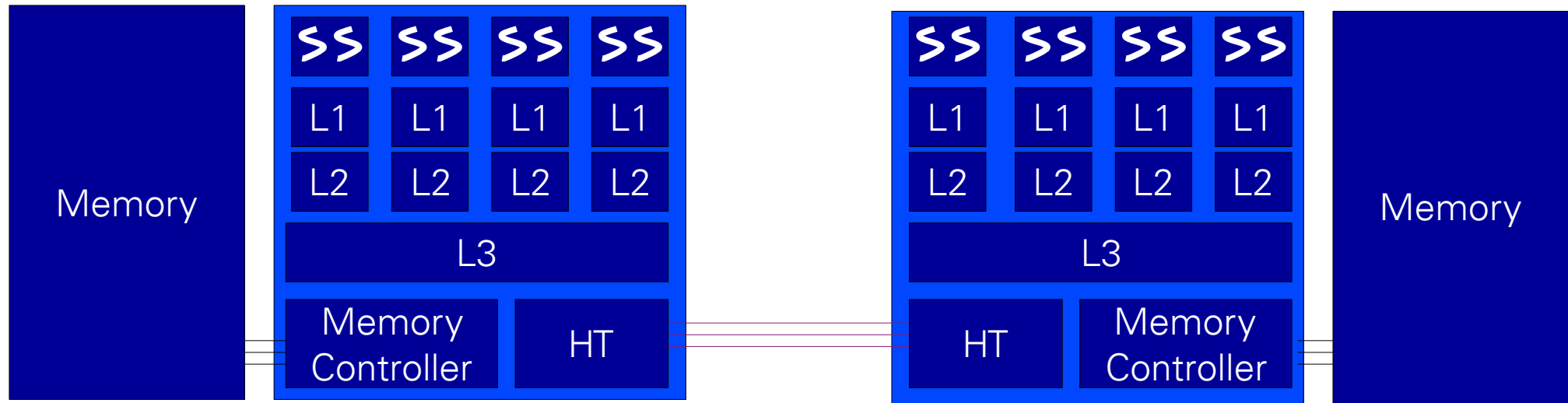
- UP-BPCP in a nutshell
 - resource holders inherits priority from blocked threads
 - resource granted if $\text{prio}(t_j) > \hat{S}$
 - only the resource holder, which holds a resource with $\hat{R}_i = \hat{S}$ receives additional resources
- UP-SRP (actually UP-CPP) in a nutshell
 - resource holder runs at max \hat{R}_i of held resources
=> only higher prioritized threads may run (acquire resources)

MP Resource Access Protocols

- UP:
 - Basic Priority Ceiling Protocol BPCP
 - Stack Resource Protocol SRP (Ceiling Priority Protocol CPP)
 - bounded priority inversion: $|CS|$
 - BPCP does not influence unrelated threads
- General Idea:
 - run UP protocol on every CPU of MP system
- Ceiling Priority i of Resource R_i : $\hat{R}_i = \max \text{prio}(t_j)$
 - here, priorities have a global meaning
- System Ceiling $\hat{S} = \max \hat{R}_i$ of held resources
- Synchronization Processor: CPU on which R_i is executed

Locking for Clustered Scheduling

- [Brandenburg '11]:
 - clustered scheduling: global within the cluster; partitioned in between



- Idea:
 - Every task helps out resource holders for a bounded time
 - Only the n-highest prioritized threads may acquire resources

Outline

- Introduction
- Terminology, Notation and Assumptions
- Anomalies + Impossibility Results
- Partitioned Scheduling
- Global (Task-Lvl migration) Scheduling
 - G-FTP (e.g., G-RMS)
 - G-EDF
- Optimal MP Scheduling
- MP – Resource Access Protocols
- Open Research Issues

Open Issues [Burns '09]

- Limited processor Utilization
 - minimally dynamic algorithms
 - novel partitioning approaches
 - increase the guaranteed processing capability; overheads
- Ineffective Schedulability Tests (in particular, sporadic workloads)
 - large gap between feasibility / infeasibility tests
 - identify finite collection of worst-case job arrival sequences
- Overheads
 - migration costs; run queue manipulations; context switching
 - algorithms that permit intra-cluster migration; task-level migr.
- **Task Models**
 - intra-task parallelism, runtime integration
 - heterogeneous resources, Turbo Boost, GPUs

References

- **[Burns '09]**
R. Davis, A. Burns, "A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems", 2009
- [Colette]
S. Collette, L. Cucu, J. Goossens, "Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism", Int. Conf. On Real-Time and Network Systems, 2007
- [Edmonds]
J. Edmonds, K. Pruhs, "Scalably scheduling processes with arbitrary speedup curves", Symp. On Discrete Algorithms, 2009
- [Brandt '10]
G. Levin, S. Funk, C. Sadowski, I. Pye, S. Brandt, "DP-Fair: A Simple Model for Understanding Optimal Multiprocessor Scheduling", 2010
- [Hong '88]
K. Hong, J. Leung, "On-line scheduling of real-time tasks", RTSS, 1988
- [Anderson '01]
B. Anderson, S. Baruah, J. Jonsson, "Static-priority scheduling on multiprocessors", RTSS, 2001
- [Lakshmanan '09]
K. Lakshmanan, R. Rajkumar, J. Lehoczky, "Partitioned Fixed-Priority Preemptive Scheduling for Multi-Core Processors", ECRTS, 2009

References

- [Fisher '07]
N. Fisher, "The multiprocessor real-time scheduling of general task systems", PhD. Thesis, University of North Carolina at Chapel Hill, 2007
- [Dertouzos '89]
M. Dertouzos, A. Mok, "Multiprocessor scheduling in a hard real-time environment", IEEE Trans. on Software Engineering, 1989
- [Dhall]
S. Dhall, C. Liu, "On a Real-Time Scheduling Problem", Operations Research, 1978
- [Baruah '06]
S. Baruah, A. Burns, "Sustainable Scheduling Analysis", RTSS, 2006
- [Corey '08]
S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, Z. Zhang, "Corey: An Operating System for Many Cores", OSDI 2008
- [Carpenter '04]
J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms", Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 2004
- [Whitehead]
J. Leung, J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks", 1982
- [Brandenburg '08]
B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, J. Anderson, "Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin", RTAS, 2008

References

- [Gai '03]
P. Gai, M. Natale, G. Lipari, A. Ferrari, C. Gabellini, P. Marceca, "A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform", RTAS, 2003
- [Block]
A. Block, H. Leontyev, B. Brandenburg, J. Anderson, "A Flexible Real-Time Locking Protocol for Multiprocessors", RTCSA, 2007
- [MSRP]
P. Gai, G. Lipari, M. Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip", RTSS, 2001
- [MPCP]
R. Rajkumar, L. Sha, J. Lehoczky, "Real-time synchronization protocols for multiprocessors", RTSS, 1988
- [Lauzac '98]
S. Lauzac, R. Melhem, D. Mosse, "Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor", EuroMicro, 1998
- [Baruah '96]
S. Baruah, N. Cohen, G. Plaxton, D. Varvel, "Proportionate progress: A notion of fairness in resource allocation", Algorithmica 15, 1996
- [Brandenburg'11]
B. Brandenburg, J. Anderson, "Real-Time Resource-Sharing under Clustered Scheduling: Mutex, Reader-Writer, and k-Exclusion Locks", ESORICS 2011