



# Multi-Core Systems

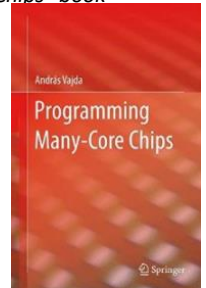


**Sara Afshar**

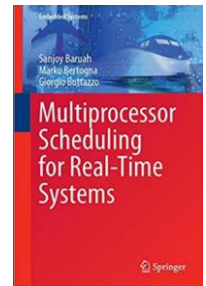
# Materials

➤ *Extra reading Materials:*

- *A Survey of Hard Real-Time Scheduling for Multiprocessor Systems*, ACM Computing Surveys, by Robert I. Davis and Alan Burns, University of York
- *Multi-core and Many-core Processor Architectures*, Chapter 2 of “Programming Many-Core Chips” book by Andras Vajda



- *Multiprocessor Scheduling for Real-Time Systems*, Sanjoy Baruah Marko Bertogna Giorgio Buttazzo





# *Multiprocessor Architecture*

# Multi-core Processors

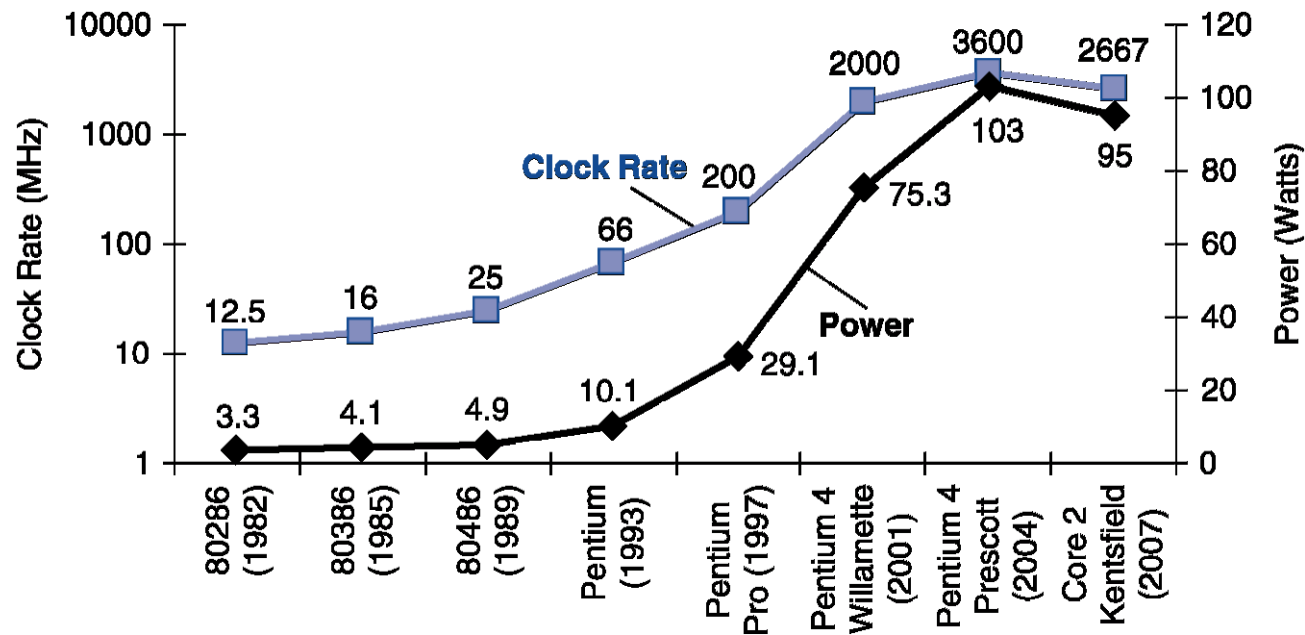
- *Integration of multiple processor cores on a single chip*
- *Single computing component with two or more independent actual processing units*
- *Better power consumption*
- *Better performance*



# Why Multi-core?

## Power Wall Problem

- Increase processor clock rate by adding transistors to small chips
- Increase power dissipation beyond inexpensive cooling techniques



# Multi-Core Architecture

- *Homogenous or heterogeneous cores*
  - *Homogenous: only identical cores*
  - *Heterogeneous: different cores in ISA, functionality and performance*
    - Cell BE architecture by IBM
    - Sony and Toshiba: gaming devices and computers targeting high performance computing

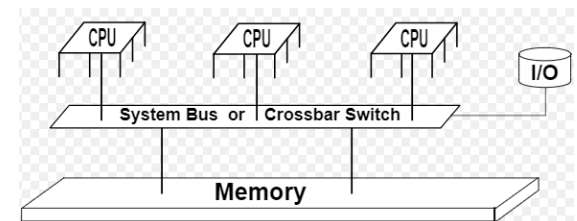
**IBM**

**SONY  
TOSHIBA**



# Multi-Core Architecture

- *Multiple cores on a chip require inter-core communication mechanism*
  - *Shared memory*
  - *Common bus*
  - *Broadcast communication medium*
  - *Maintain a cash-coherent memory*
    - *Cash coherent: if all processors have a consistent view of each memory location at any point in time*
    - *MESI a cache coherence protocol deployed in Intel processors*
- *Do not scale up to high number of cores (~250 cores) → message passing*





# Multi-Core Architecture

- *Commonly used cache coherence protocol MESI protocol deployed in Intel processors*
  - *Modified—A block in this state is the only valid copy of the block. The memory does not hold valid information and no other cache may have a valid copy. The core that owns this block can write to it without notifying any other core.*
  - *Exclusive—The first core to read in a block from memory will load it into the Exclusive state. This means that if the same core later modifies the block, it can silently be upgraded to the modified state without notifying anyone else. This is beneficial for the vast majority of data which is not shared between threads or processes. If a block is in the exclusive state we know that*
    - *the memory has an up-to-date copy*
    - *there are no other cached copies in the system*



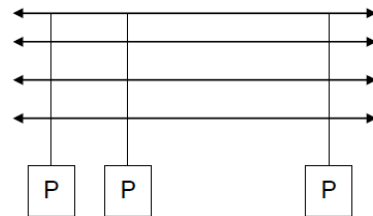


# Multi-Core Architecture

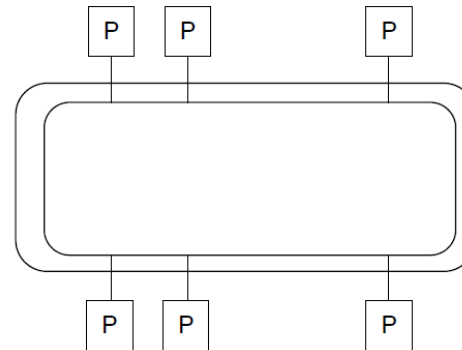
- *Shared—As soon as a second core is reading the same block, it will be loaded to the cache of that core and will be marked Shared in all caches*
- *Invalid—As soon as one of the copies is modified by one of the cores, all other copies will be marked invalid and will need to be refreshed at the next access*

# Multi-Core Architecture

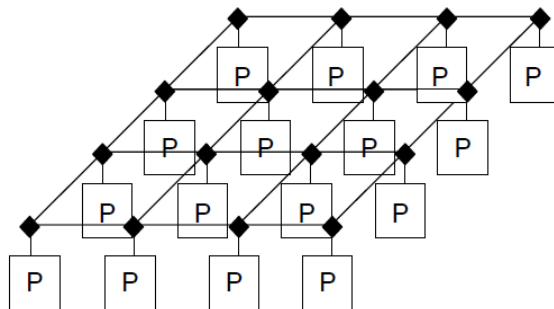
- *Shared bus problems in high number of cores*
  - *Latency*
  - *bandwidth*
- *Types of on-chip interconnect*



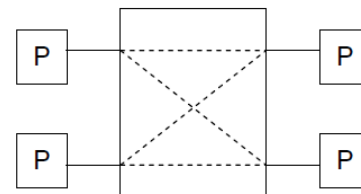
Bus interconnect



Ring interconnect



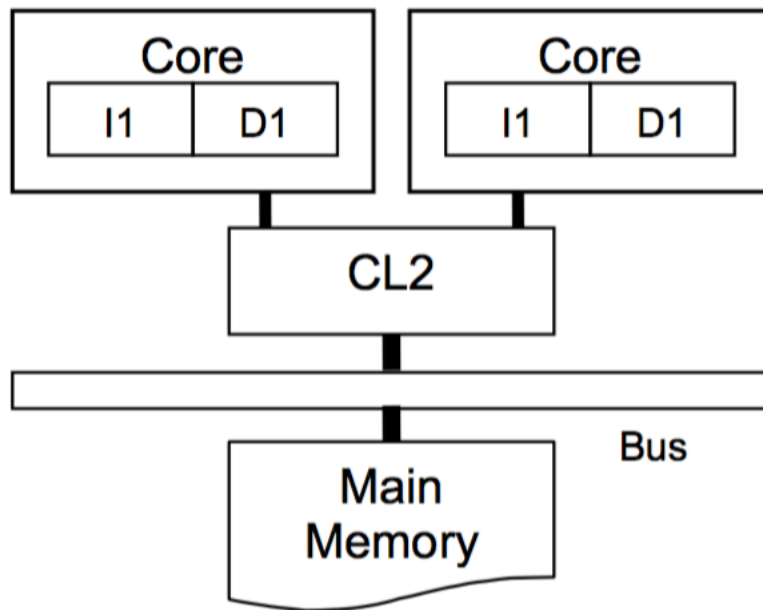
Mesh interconnect



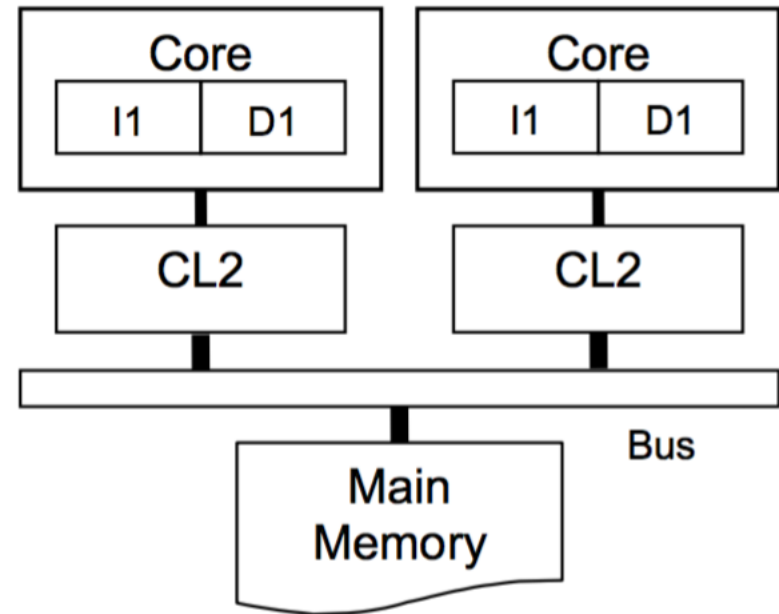
Crossbar interconnect

# General Processor Architectures

- *Cores may or may not share cash*
  - *Local cash memory: not to flood the bus with memory and I/O traffic*
  - *Typically one or two levels between the core and the bus*

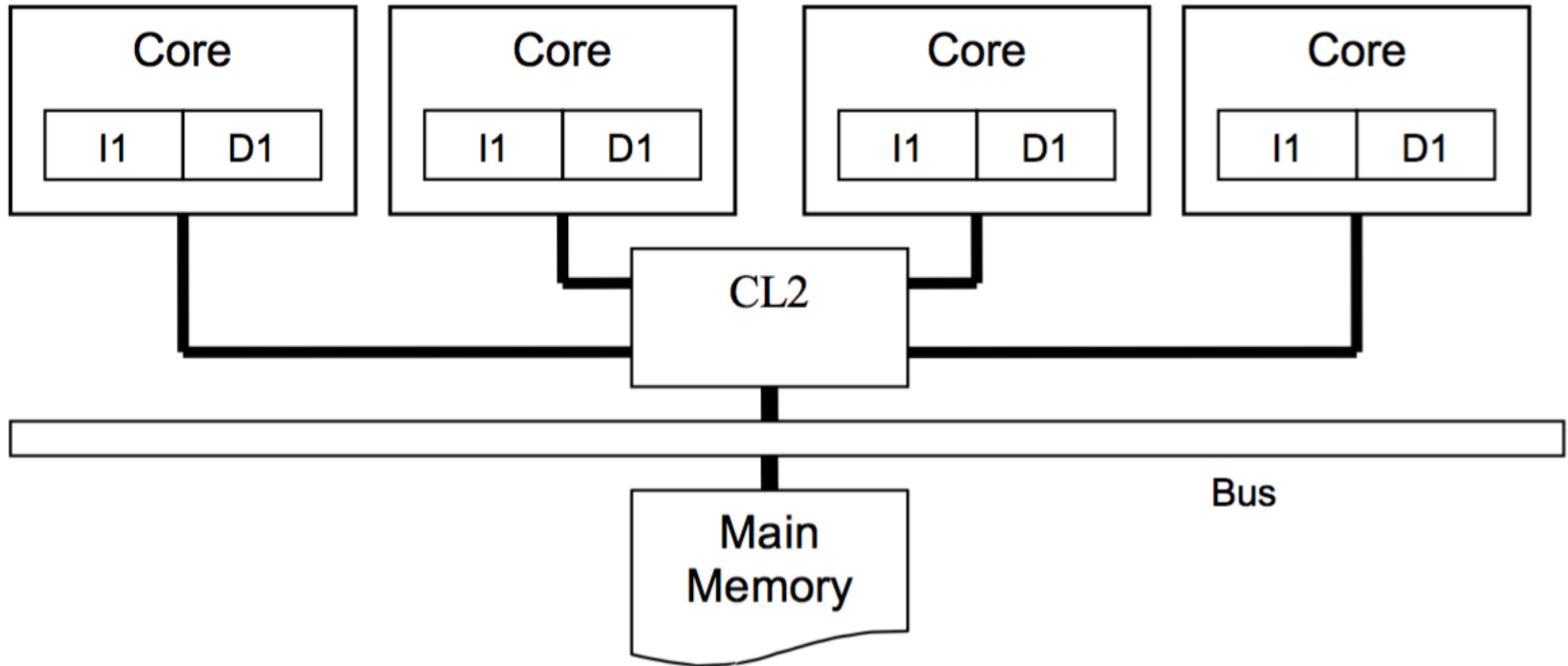


(a) Shared CL2 (Intel - Xeon)

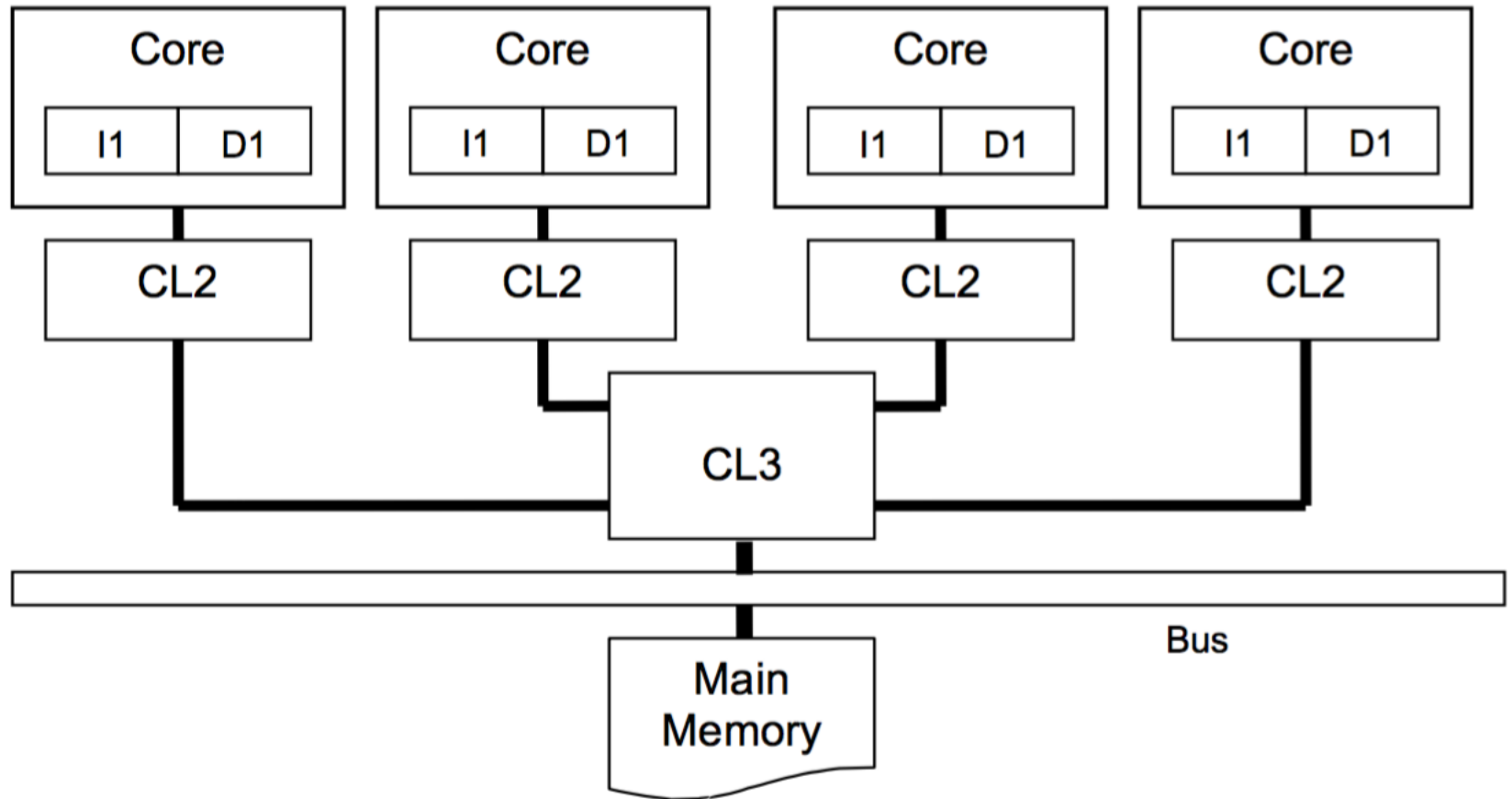


(b) Dedicated CL2 (AMD - Athlon)

# Intel – Kentsfield XE Architecture

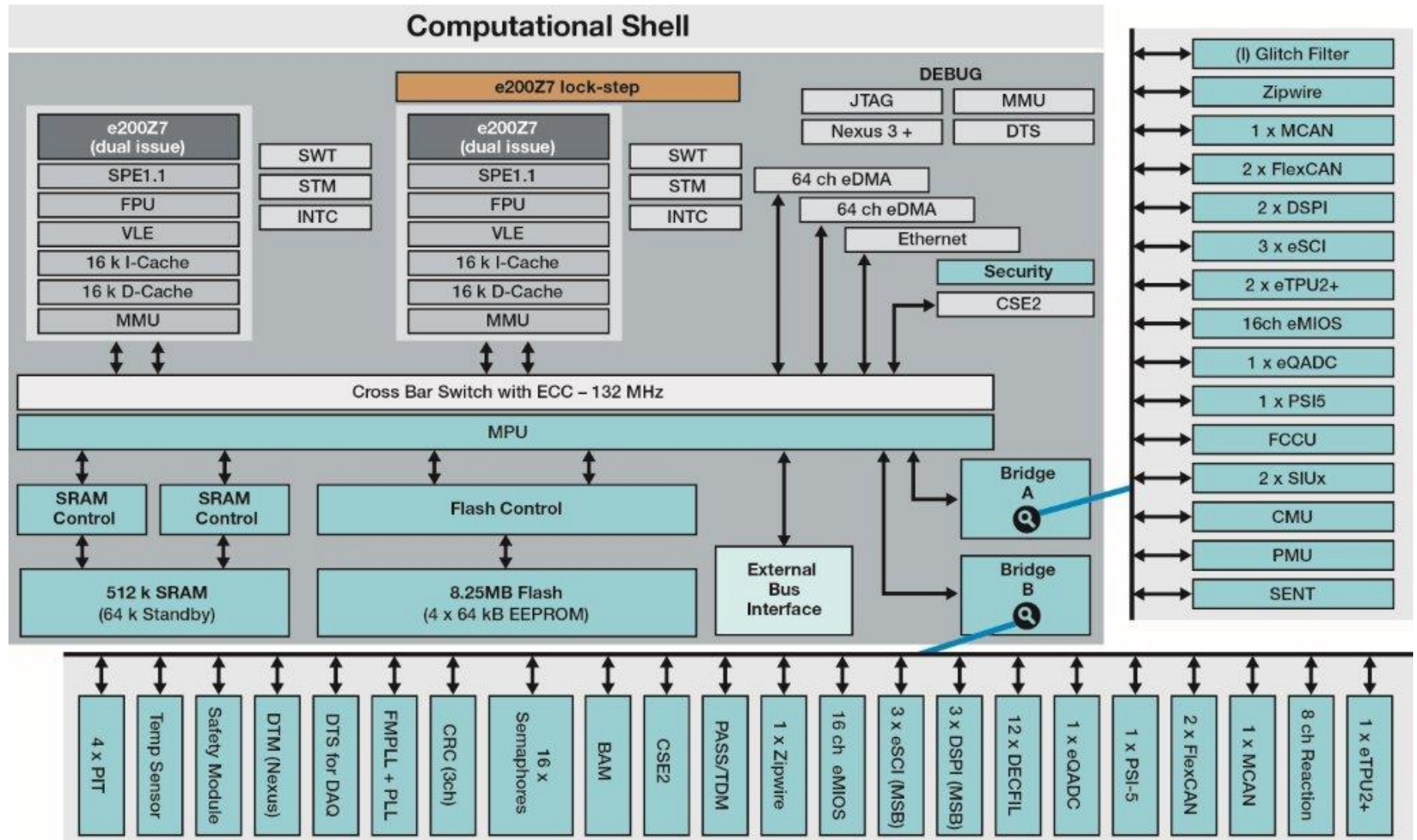


# AMD - Opteron Architecture

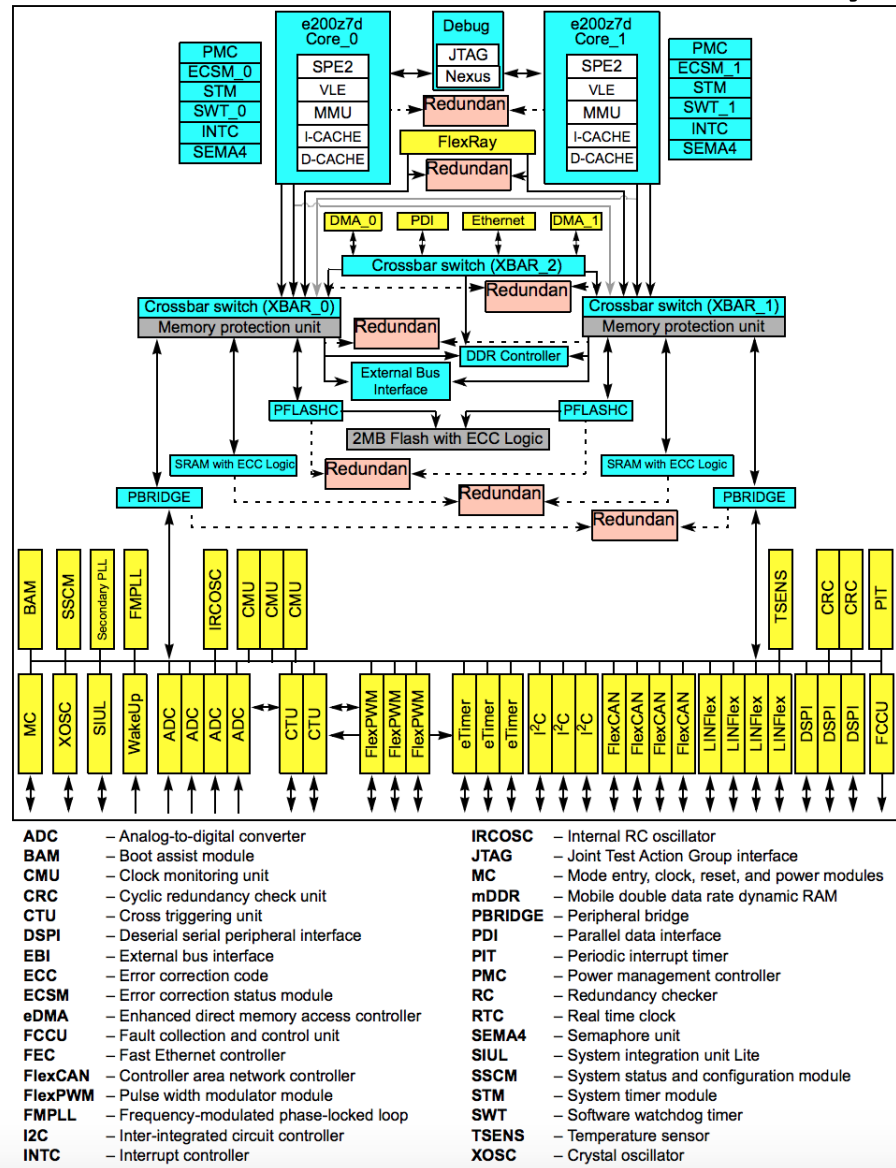


A. Asaduzzaman, Cache Optimization for Real-Time Embedded Systems, Doctoral thesis, Faculty of Engineering and Computer Science, Florida Atlantic University, USA, 2012.

# MPC5777c (no shared caches, but a shared system bus)

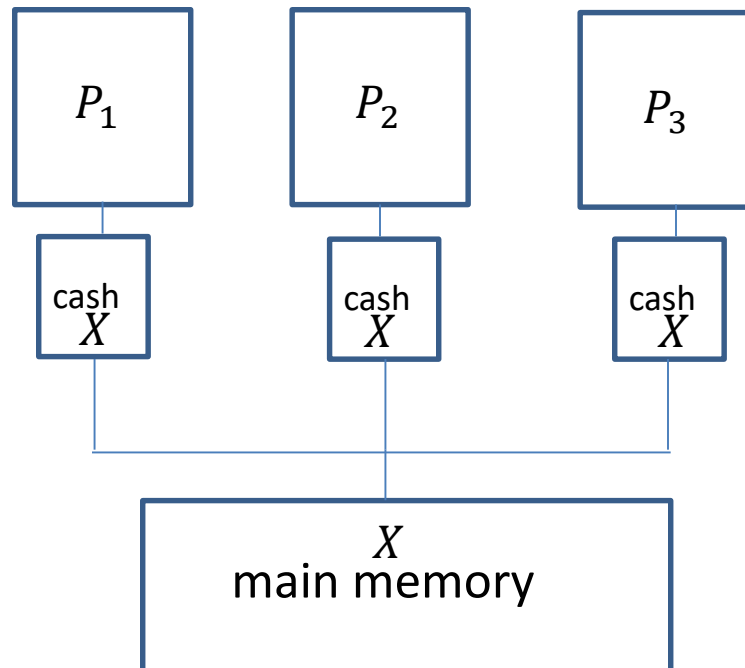


# MPC5675K (No shared caches, but a shared system bus)



# Cash Coherent

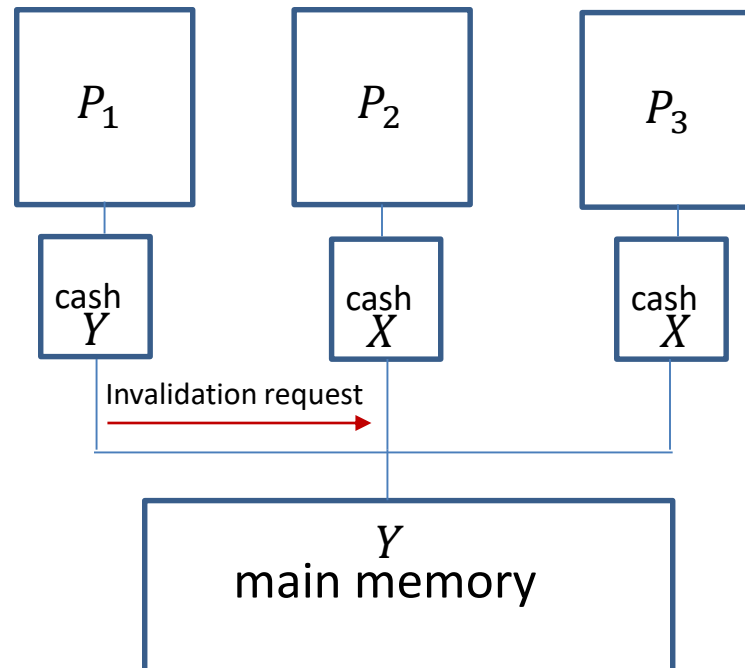
- *Private caches*
  - *Data should be consistent*
  - *Invalidation with snooping*
    - *Invalidation: If a core writes to a memory place all copies in other caches are invalidated*
    - *Snooping: all cores continuously “snoop” (monitor) the bus*





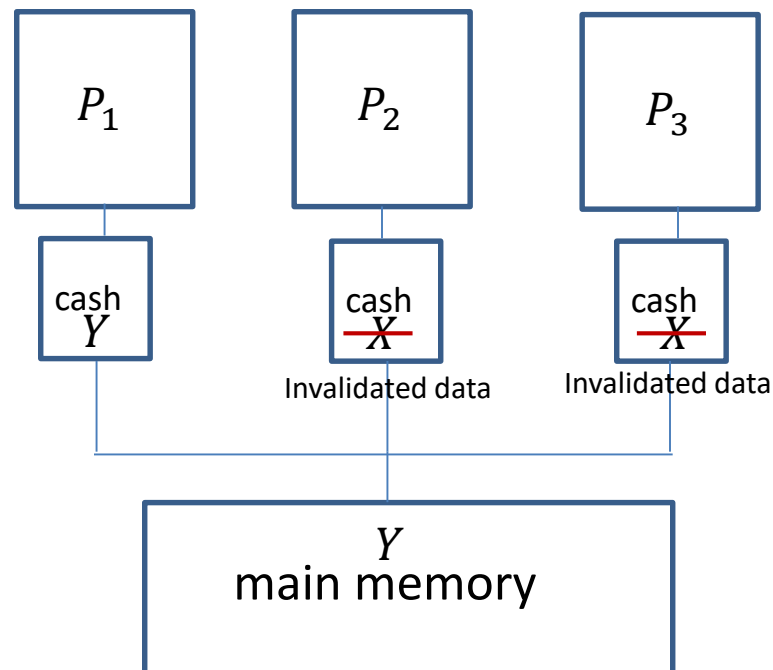
# Cash Coherent

- *Private caches*
  - *Data should be consistent*
  - *Invalidation with snooping*
    - *Invalidation: If a core writes to a memory place all copies in other caches are invalidated*
    - *Snooping: all cores continuously “snoop” (monitor) the bus*



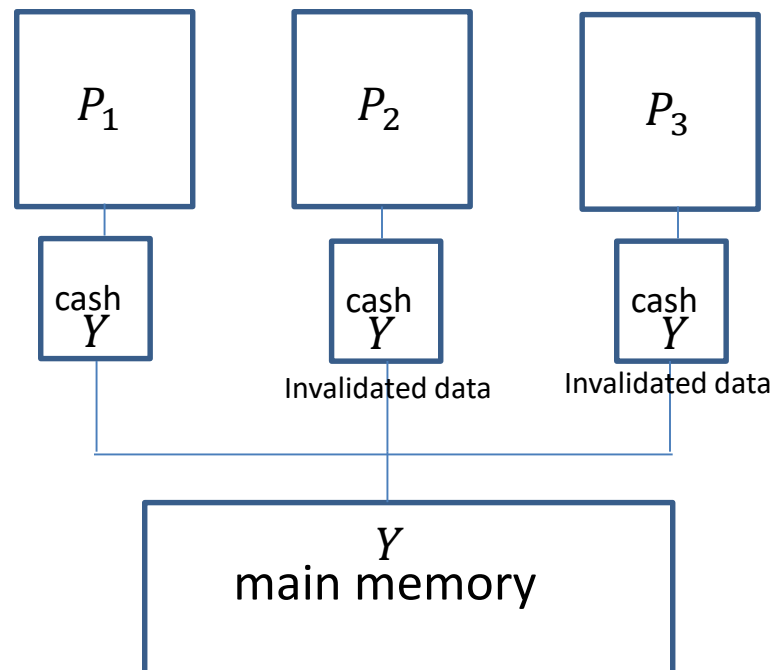
# Cash Coherent

- *Private caches*
  - *Data should be consistent*
  - *Invalidation with snooping*
    - *Invalidation: If a core writes to a memory place all copies in other caches are invalidated*
    - *Snooping: all cores continuously “snoop” (monitor) the bus*



# Cash Coherent

- *Private caches*
  - *Data should be consistent*
  - *Invalidation with snooping*
    - *Invalidation: If a core writes to a memory place all copies in other caches are invalidated*
    - *Snooping: all cores continuously “snoop” (monitor) the bus*





# *Multiprocessor Scheduling*

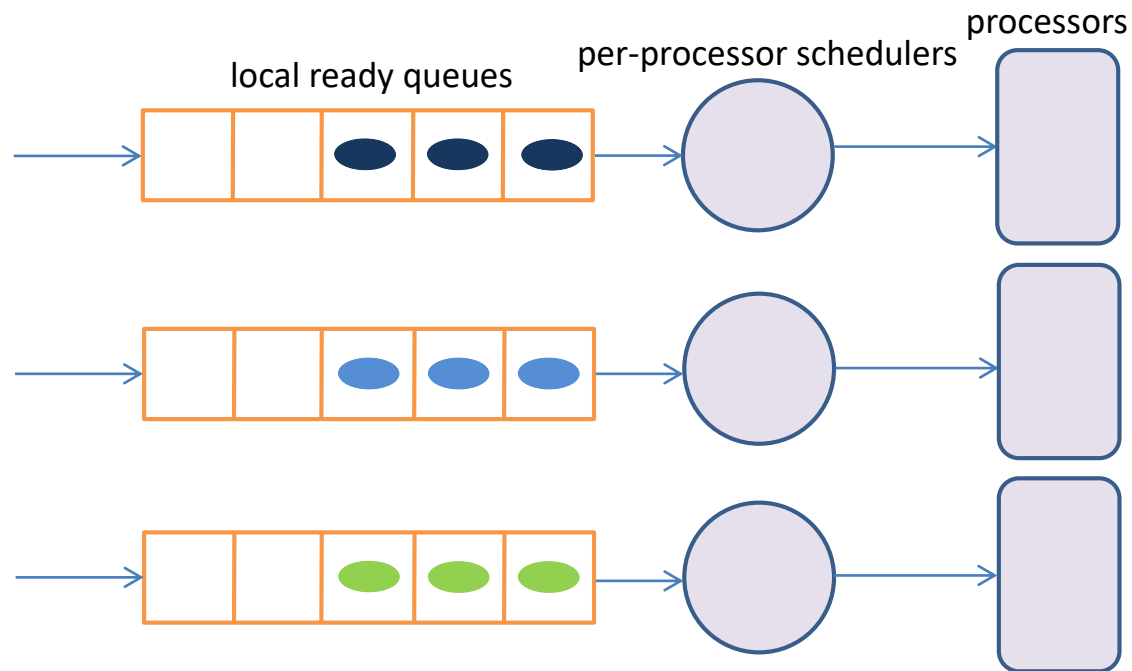


# Scheduling

- *Scheduler: operating systems (OS)*
- *Scheduling tasks:*
  - *Uniprocessor: which task should run when*
  - *Multiprocessor: which task should run when and where*

# Partitioned Scheduling

- *Each core has its own scheduler and ready queue*
- *Tasks are pre-assigned to cores during design time*
- *Tasks only execute on their pre-assigned cores*



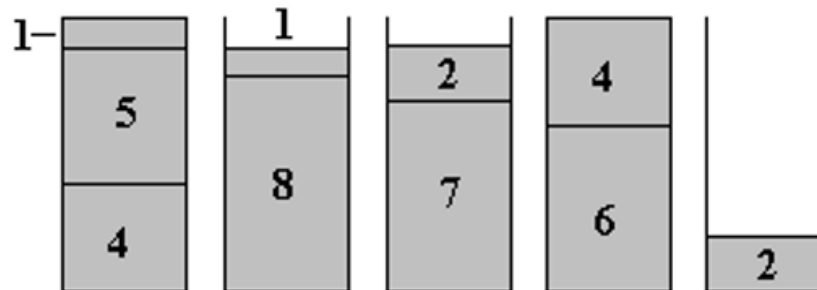


# Partitioned Scheduling

- *Advantages:*
  - *Higher degree of predictability*
  - *Trivial implementation*
  - *Low run-time overheads*
  - *Re-use of uniprocessor scheduling*
  - *Support of commercial real-time Oses: VxWorks, QNX, LynxOS, ThreadX*
  - *Availability in industrial Oses and standards*
    - *Erika Enterprise, POSIX, AUTOSAR*
- *Disadvantages:*
  - *Allocation of tasks to cores a bin-packing problem*
    - *NP-hard in the strong sense*
    - *Finding an optimal solution cannot be done in a polynomial time*
    - *Heuristics*
  - *Fragmentation of processor utilization*

# Allocation Heuristics

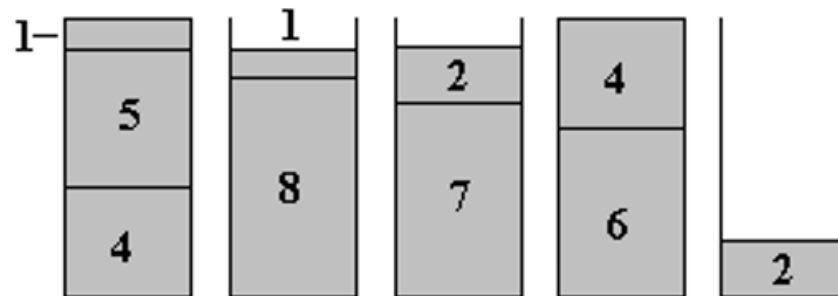
- *First Fit (FF):*
  - *Tasks in the order of arrival*
  - *Next task into the first core that it fits into. If does not fit in any core add a new core*
  - *Example.  $S = \{4, 8, 5, 1, 7, 6, 1, 4, 2, 2\}$  and bins of size 10*
  - *Fills cores*





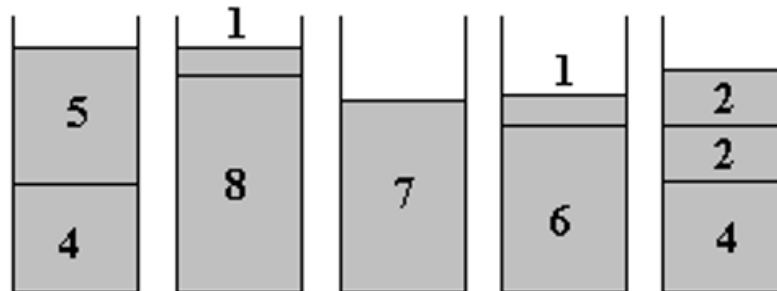
# Allocation Heuristics

- *Best Fit (BF):*
  - *Tasks in the order of arrival*
  - *Next task into the core that leaves the least place. If does not fit in any core add a new core*
  - *Example.  $S = \{4, 8, 5, 1, 7, 6, 1, 4, 2, 2\}$  and bins of size 10*
  - *Fills cores*



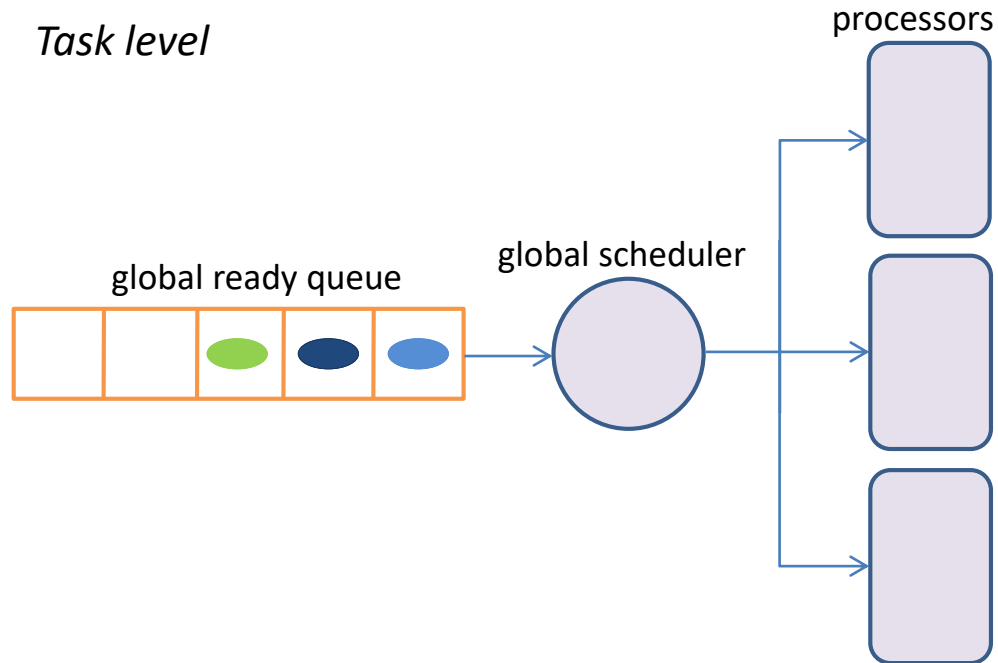
# Allocation Heuristics

- *Worst Fit (WF):*
  - *Tasks in the order of arrival*
  - *Next task into the core that leaves the most place. If does not fit in any core add a new core*
  - *Example.  $S = \{4, 8, 5, 1, 7, 6, 1, 4, 2, 2\}$  and bins of size 10*
  - *Balance load*



# Global Scheduling

- *One global scheduler and unique ready queue*
- *Tasks are assigned dynamically to available cores during run-time*
- *Tasks can migrate:*
  - *Job level*
  - *Task level*





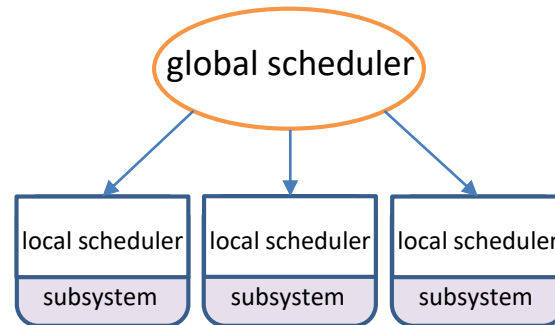
# Global Scheduling

- *Advantages:*
  - *Preferable in adaptive and open systems since assigns tasks dynamically, hence avoids complex mapping problem*
    - *Adaptive systems: task requirements change during run-time*
    - *Open systems: tasks may be added to or removed from the system dynamically*
  - *Less context switch, preemptions*
    - *Tasks get preempted only when there is no idle processor*
- *Disadvantages:*
  - *Migration overhead is expensive*
  - *RM and EDF are not optimal anymore for multiprocessors*



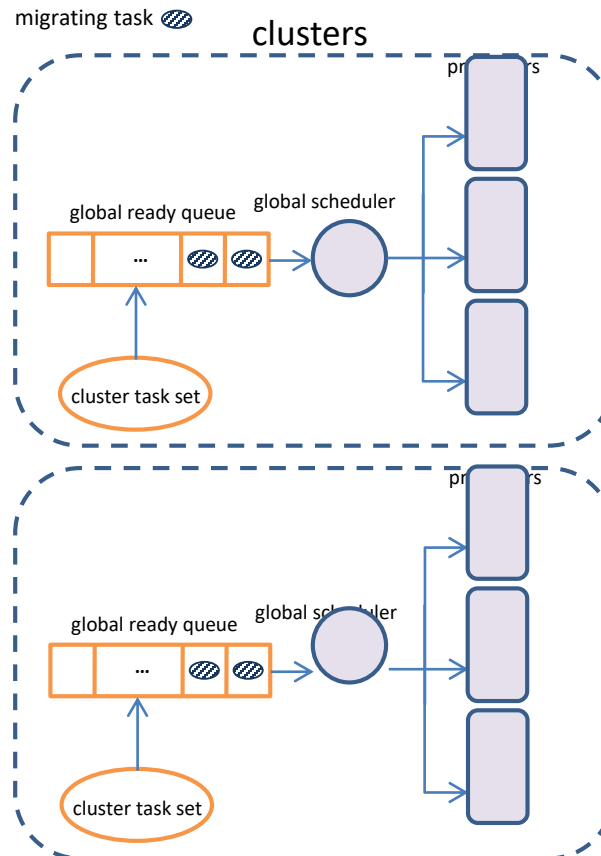
# Hierarchical Scheduling

- *Schedule tasks in hierarchy manner*
- *Typically servers are used to provide processor bandwidth for sub-systems*
  - *Server provide timing isolation*



# Virtual Scheduling



- *Cluster of processors*
- *Tasks are allocated to clusters*
- *Global scheduling within a cluster*

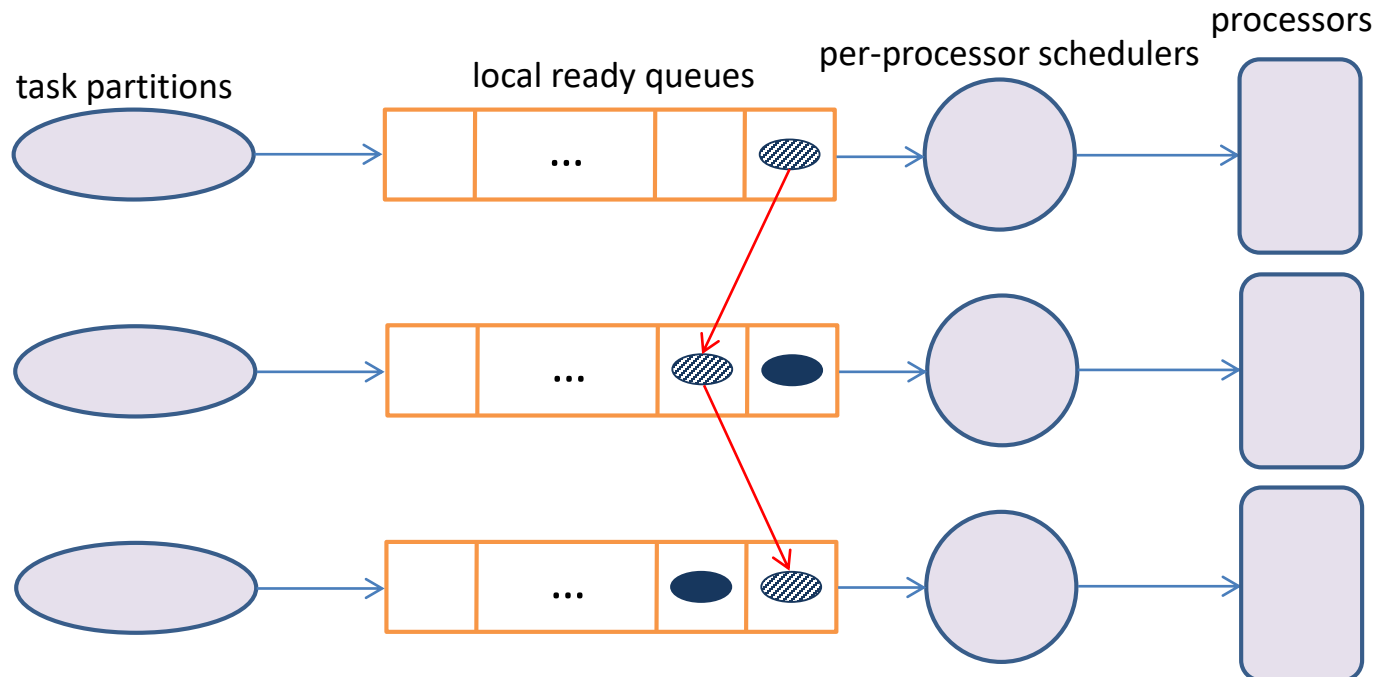


# Hybrid Scheduling

➤ *Semi-partitioned scheduling*

➤ *Synchronization Deferrable Servers (SDS) framework*

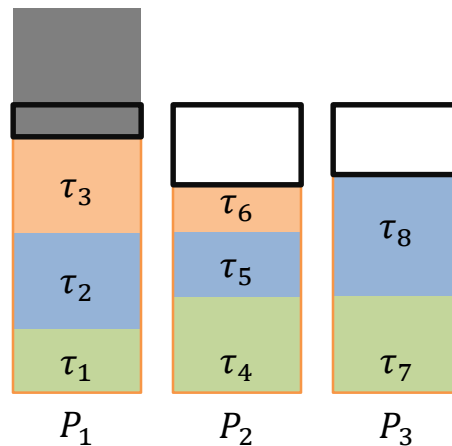
migrating task   
partitioned task 





# Semi-Partitioned Scheduling

- *Partitioned tasks: non-split tasks*
- *Migrating tasks: split tasks*

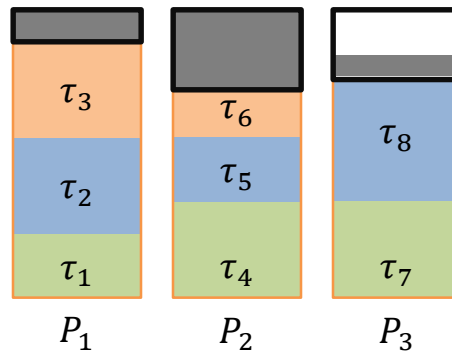






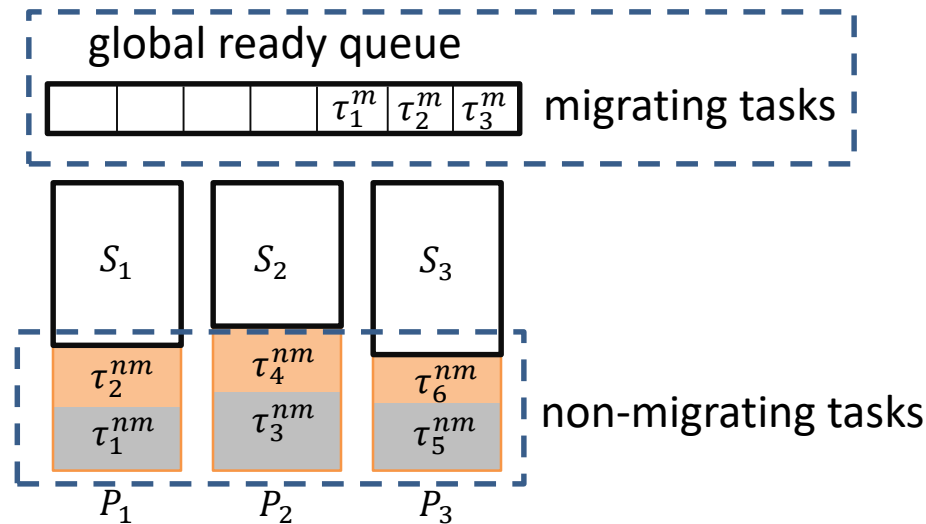
# Semi-Partitioned Scheduling

- *Partitioned tasks: non-split tasks*
- *Migrating tasks: split tasks*



# SDS

- *SDS by Zhu et al., RTSS'11*
- *Main critical application*
  - *Partitioned scheduling*
- *Additional non-critical applications*
  - *Global scheduling within servers; utilize the remained capacity on each core*
  - *Resource reservation techniques: servers  $\rightarrow$  temporal isolation*





# Scheduling Definitions/Concepts

- *Schedulability:*
  - $\tau_i$  is schedulable given a scheduling algorithm if:  $R_i \leq D_i$
  - A task set is schedulable under a scheduling algorithm if all of its tasks are schedulable
- *Sufficient schedulability test: if all task sets deemed schedulable using the test are in fact schedulable*
- *Necessary schedulability test: if all task sets deemed unschedulable using the test are in fact unschedulable*
- *Exact schedulability test: both sufficient and necessary*



# Scheduling Definitions/Concepts

## ➤ *Feasibility:*

- *A task set is feasible if there exists some scheduling algorithm that can schedule all possible sequences of jobs generated by the task set*
  - *E.g.,  $u_{sum} \leq m$  ( $u_{sum}$ : task set utilization,  $m$ : number of cores)*
  - *Necessary and sufficient for implicit deadline periodic task sets*
  - *Necessary for constrained and arbitrary deadline task sets*

## ➤ *Optimality:*

- *A scheduling algorithm is optimal if it can schedule all of the task sets that can be scheduled by any other algorithm, i.e. all the feasible task sets*



# Scheduling Definitions/Concepts

## ➤ Comparability

- *Dominance: Algorithm A dominates algorithm B, if all task sets schedulable by B are also schedulable by A, and task sets exist that are schedulable by A, but not by B.*
- *Incomparable: Algorithm A and B are incomparable if task sets exist that are schedulable by A, but not by B and vice-versa*
- *Equivalent: Algorithm A and B are equivalent if all the task sets schedulable by B are also schedulable by A and vice-versa*

## ➤ Predictability

- *A scheduling algorithm is predictable if the response times of jobs cannot be increased by decreases in their execution times, with all other parameters remaining constant*



# Scheduling Definitions/Concepts

- *Sustainability:*
  - *A scheduling algorithm/test is sustainable if and only if changes below does not make a schedulable task set unschedulable*
    - *Decreasing execution times*
    - *Increasing periods or inter-arrival times*
    - *Increasing deadlines*
- *PFP (partitioned fixed priority) and PEDF (partitioned earliest deadline first) are sustainable*
- *GFP (global FP) and GEDF (global EDF) are not sustainable*



# Scheduling Definitions/Concepts

- *Anomalies*
  - *When a change in a task set parameter results in counter-intuitive effect on schedulability*
    - *Period anomaly: increase in period*
    - *Execution time anomaly: decrease in execution time*
    - *Critical instant effect: not known under GFP as is under PFP/uniprocessor scheduling*
- *As an example, unsustainable scheduling tests by increasing period:*
  - *GFP for periodic task sets*
  - *Global optimal scheduling (full migration, dynamic priorities) for periodic task sets*



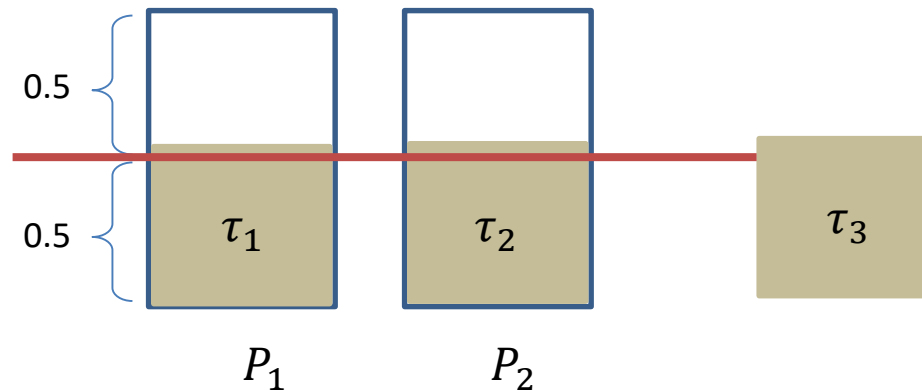
# Performance Metrics

- *Schedulability tests:*
  - *Utilization bounds*
  - *Approximation Ratio*
  - *Resource Augmentation or Speedup factor*
  - *Empirical measures, e.g., the percentage of schedulable task sets*



# Utilization Bounds

- *Schedulability test for implicit-deadline task sets*
- *Sufficient but not necessary*
- *Worst-case utilization bound  $U_A$  for a scheduling algorithm  $A$  : minimum utilization of any implicit deadline task set schedulable by  $A$* 
  - *I.e., there are no implicit deadline task sets with  $u_{sum} \leq U_A$  that are unschedulable by  $A$* 
    - $u_{sum}$ : task set utilization
    - E.g.,  $U_A$  for any partitioning algorithm is  $U_A = (m+1)/2$





# Approximation Ratio

- *Used for comparing scheduling performance of algorithm  $A$  with an optimal algorithm*
- *Increased number of cores for maintaining schedulability under  $A$*
- $\mathfrak{R}_A = \lim_{M_O \rightarrow \infty} (\max_{\forall \tau} (\frac{M_A}{M_O}))$ 
  - $M_O$ : *number of processors required by an optimal algorithm*
  - $M_A$ : *the number required by algorithm  $A$*
  - $\tau$  *given task set*
- $\mathfrak{R}_A \geq 1$



# Resource Augmentation

## (Speedup Factor)

- *Used for comparing the performance of a scheduling algorithm  $A$  with an optimal algorithm*
- *Increased processor speed for maintaining schedulability under  $A$*
- $f_A = \max_{\forall m, \tau}(f(\tau))$ :
  - *Minimum factor to increase speed of all  $m$  processors such that all feasible task sets on  $m$  processors of speed 1 become schedulable by algorithm  $A$*
- $f_A \geq 1$



# Empirical Measures

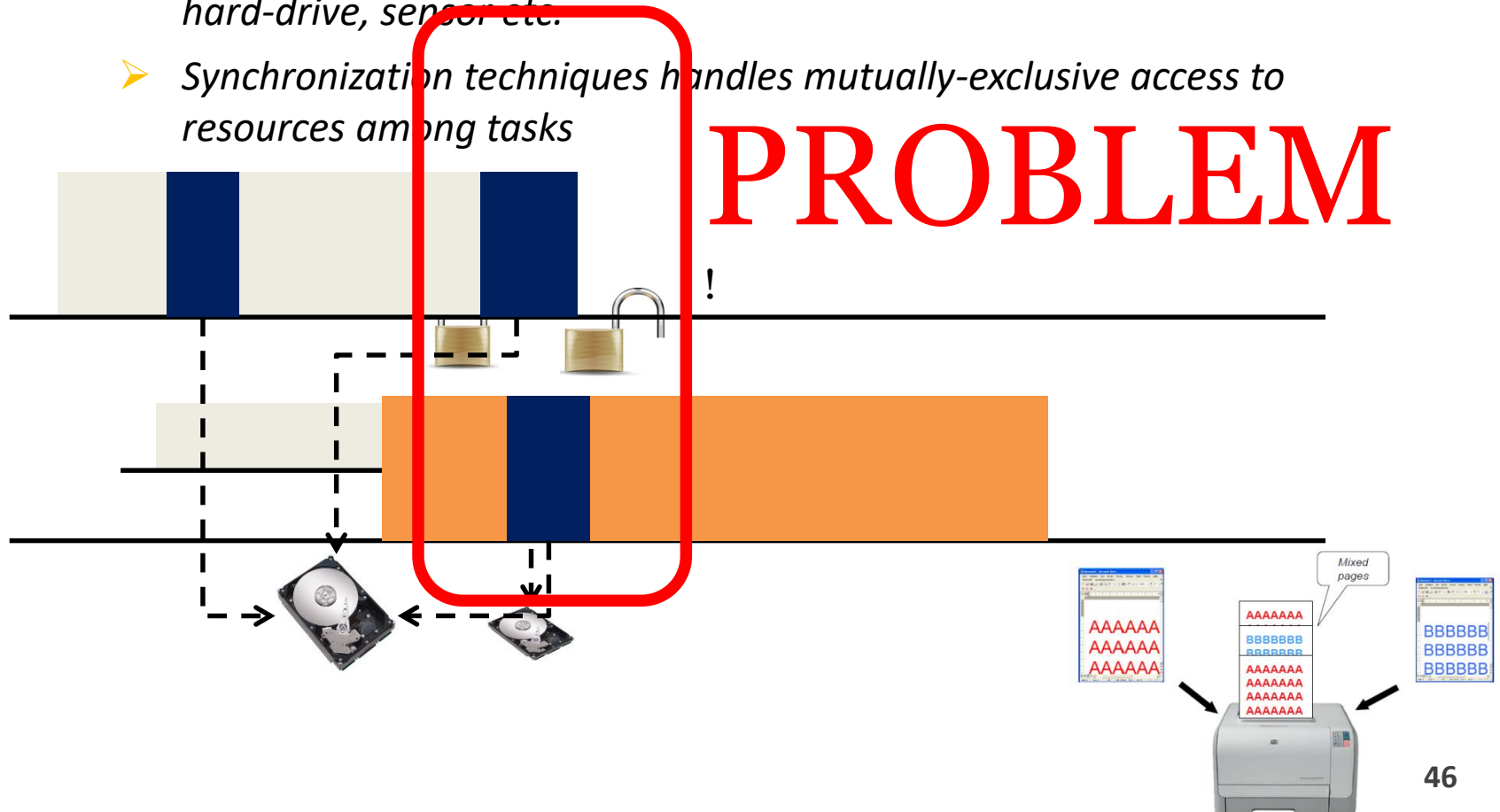
- *Comparative scheduling performance and analysis of algorithms*
  - *Number of randomly generated task sets deemed schedulable*
- *Scheduling algorithms performances compared by varying task set parameters:*
  - *Number of tasks*
  - *Number of processors*
  - *Task set utilization*
  - *Task periods*
  - *Task deadlines*
  - *Task utilization*
- *Lacks the involvement of*
  - *Scheduling decision overhead*
  - *Context switches*
  - *migrations*



# *Multiprocessor Resource Sharing*

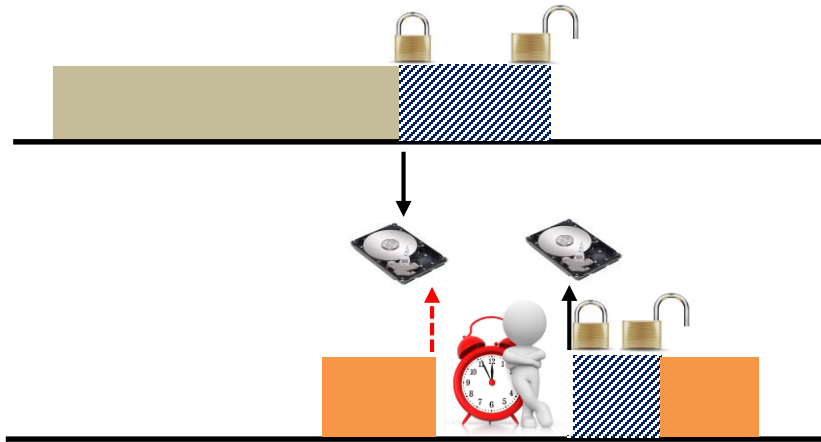
# Resource Sharing

- Tasks may use hardware/software components such as a database, hard-drive, sensor etc.
- Synchronization techniques handles mutually-exclusive access to resources among tasks



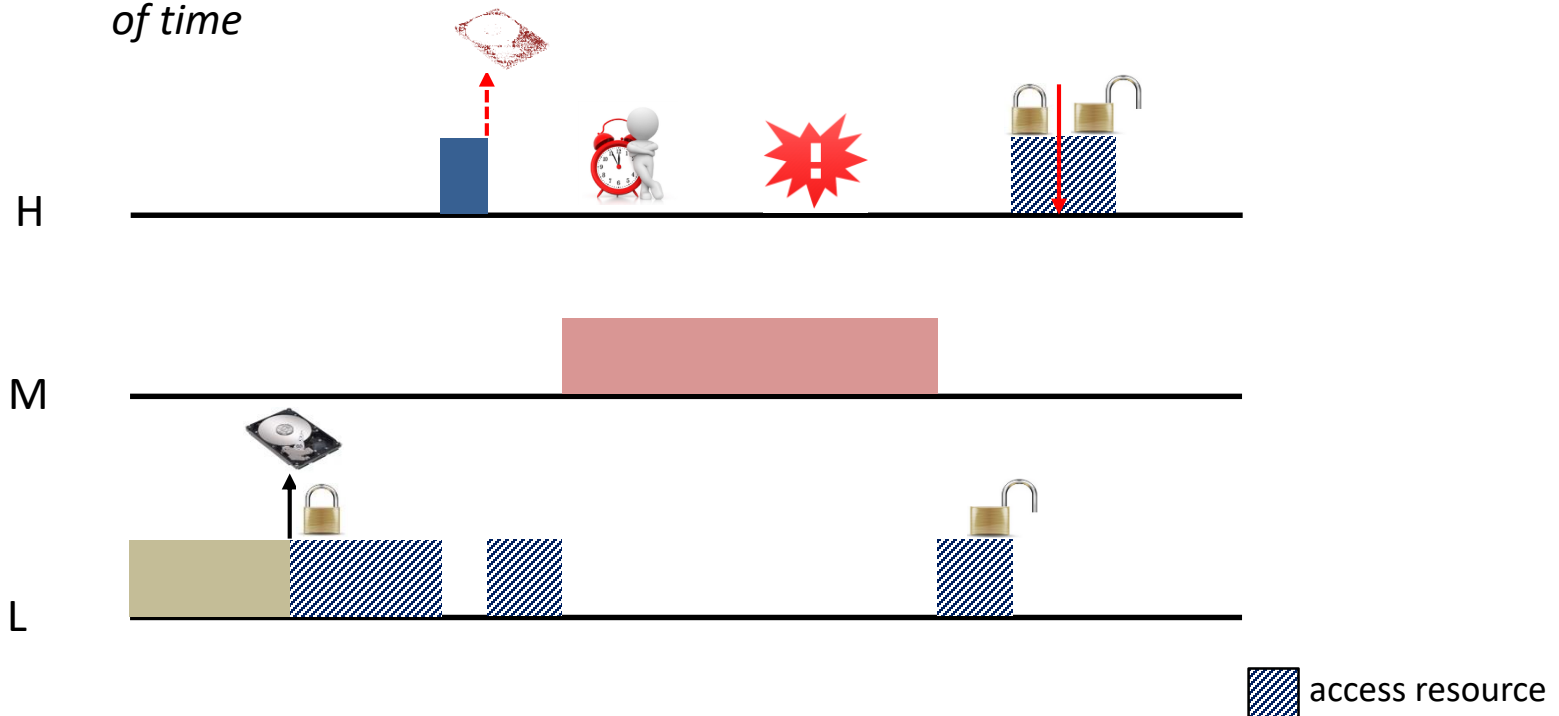
# Resource Sharing Protocols

- *Tasks may experience delay due to resource sharing*
- *Blocking delays should be bound for real-time systems*



# Blocking in Uniprocessor

- *Blocking can endanger system correctness*
- **Priority inversion**: high priority task is forced to wait for a lower priority task for an unbounded amount of time







# Resource Sharing Synchronization Protocols

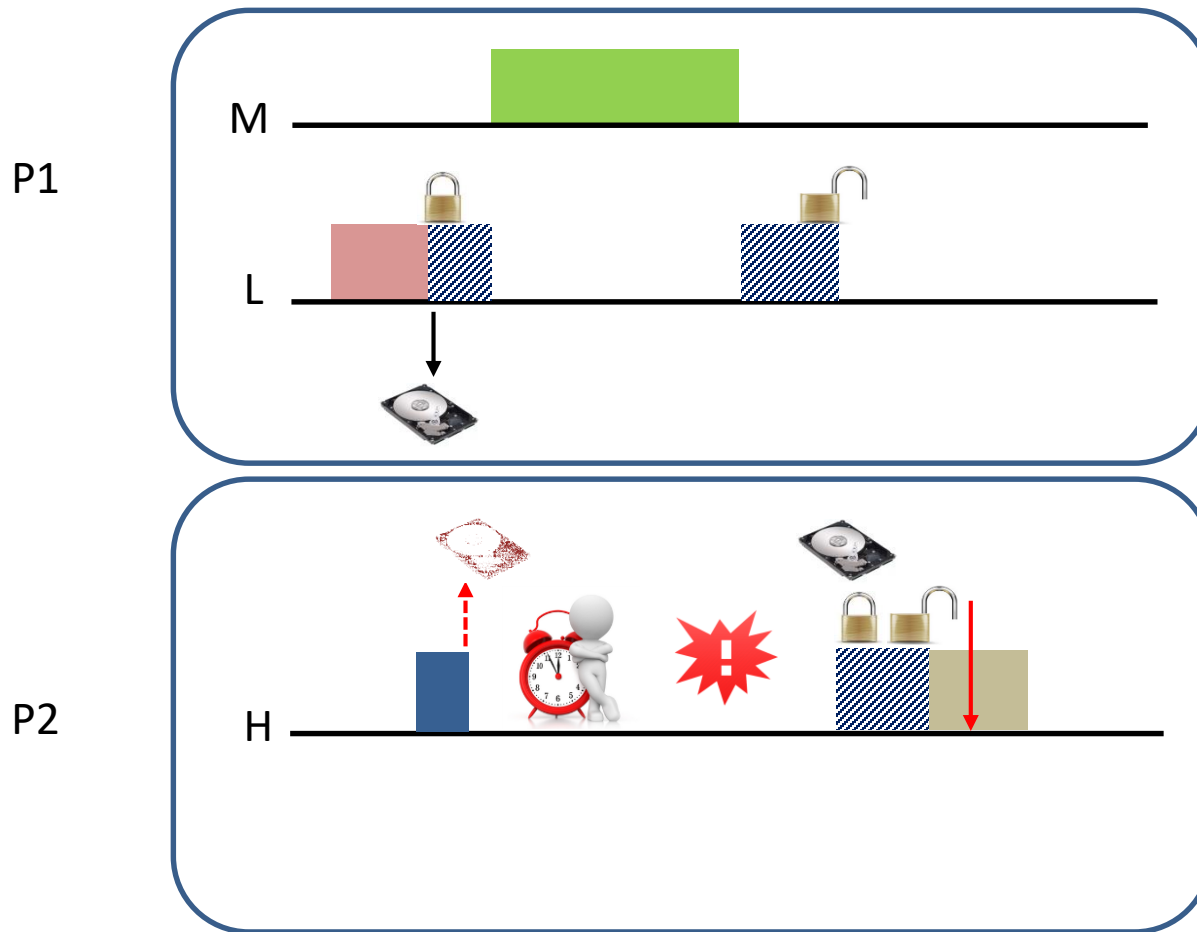
- *Mutually-exclusive resource sharing*
- *Bound waiting times of tasks to meet their deadlines*
  - *Maximum delay due to resource sharing → system schedulability can be checked*



# Multiprocessor Resource Sharing

- *Local resources are only used by tasks allocated to the same core*
  - *Uniprocessor resource sharing protocols*
    - *SRP, PCP, IPCP*
- *Global resources are used by tasks allocated to different cores*
  - *Multiprocessor synchronization protocols*

# Remote Blocking in Multiprocessor



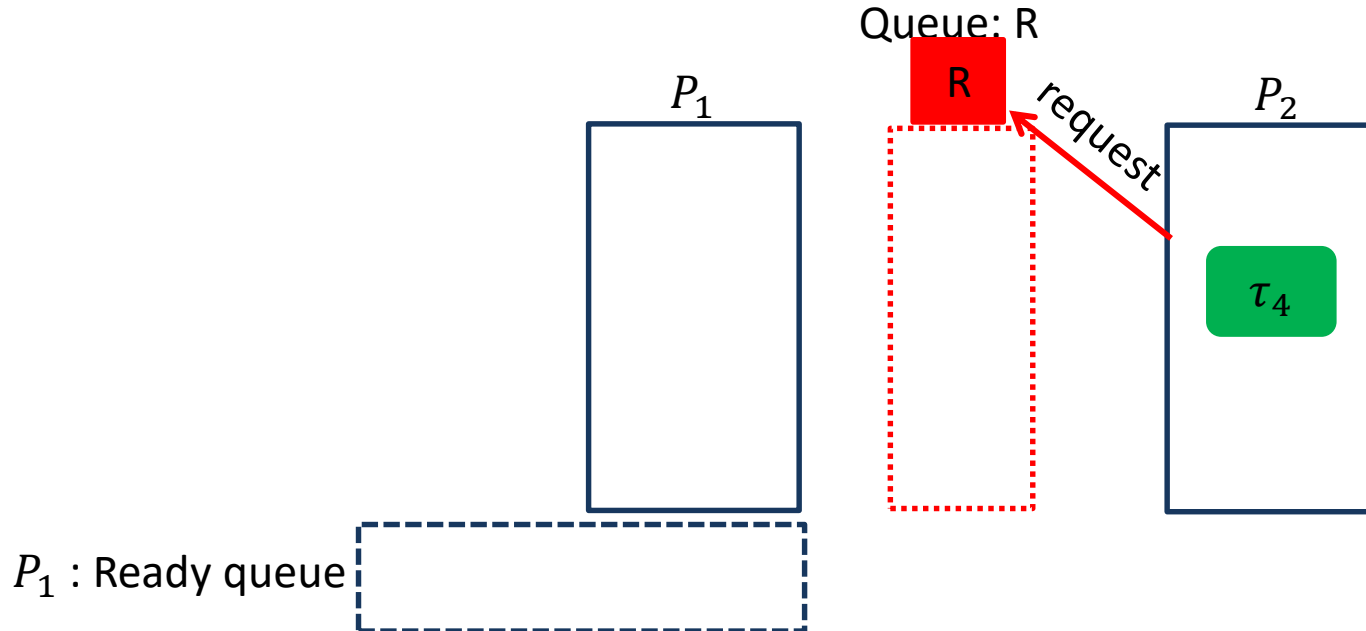


# Multiprocessor Resource Sharing Synchronization Protocols

- *Multiprocessor synchronization protocols*
  - *Spin-based: performs a non-preemptive busy wait loop*
  - *Suspension-based: suspends and releases the processor*

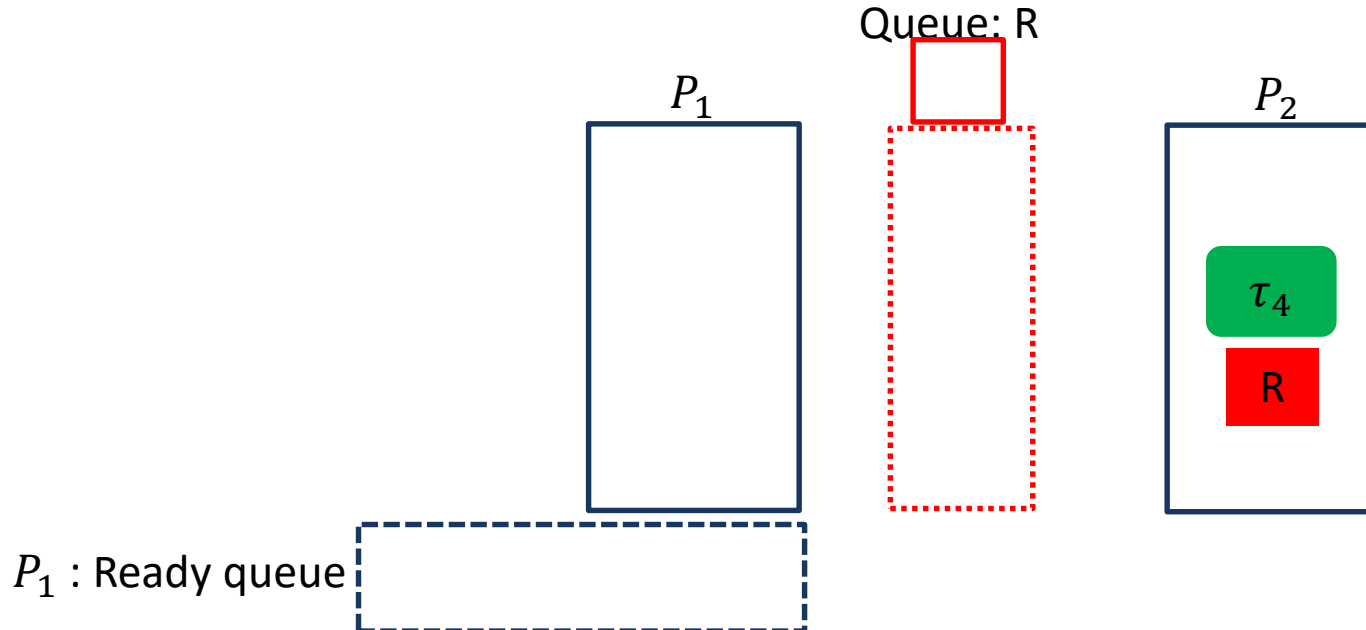
# Spin-Based Protocol

$\tau_4$  requests shared resource  $R$



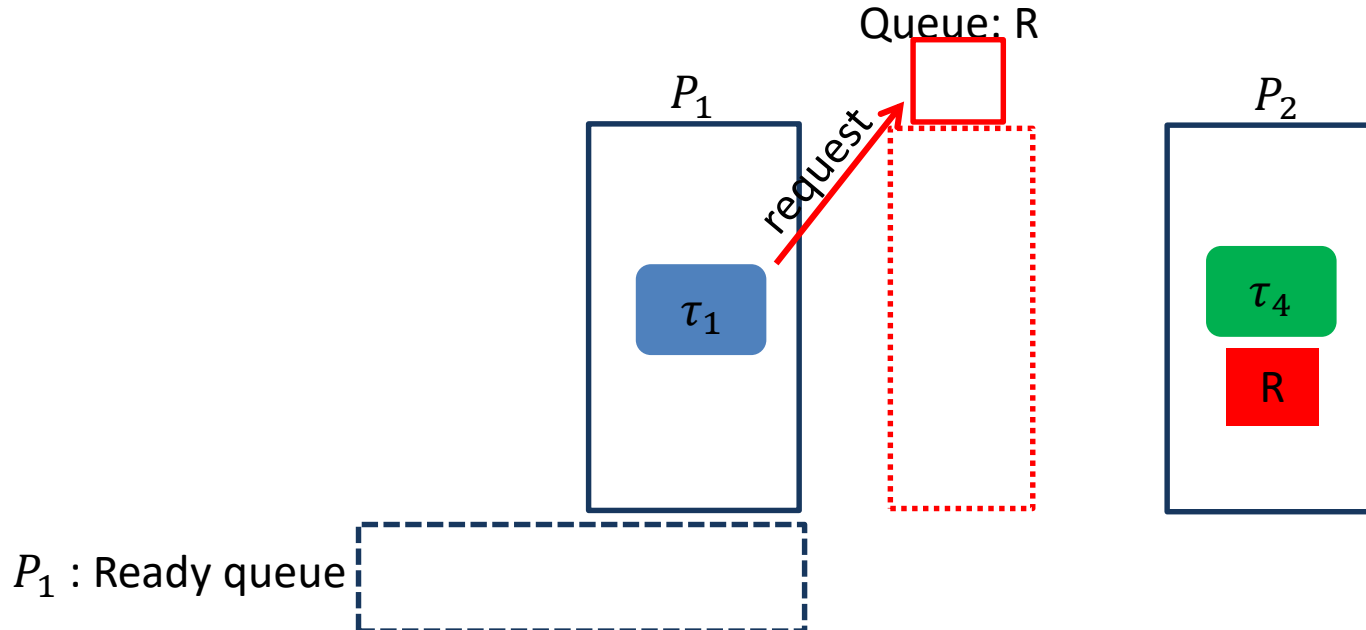
# Spin-Based Protocol

$\tau_4$  locks shared resource  $R$  since the resource is free



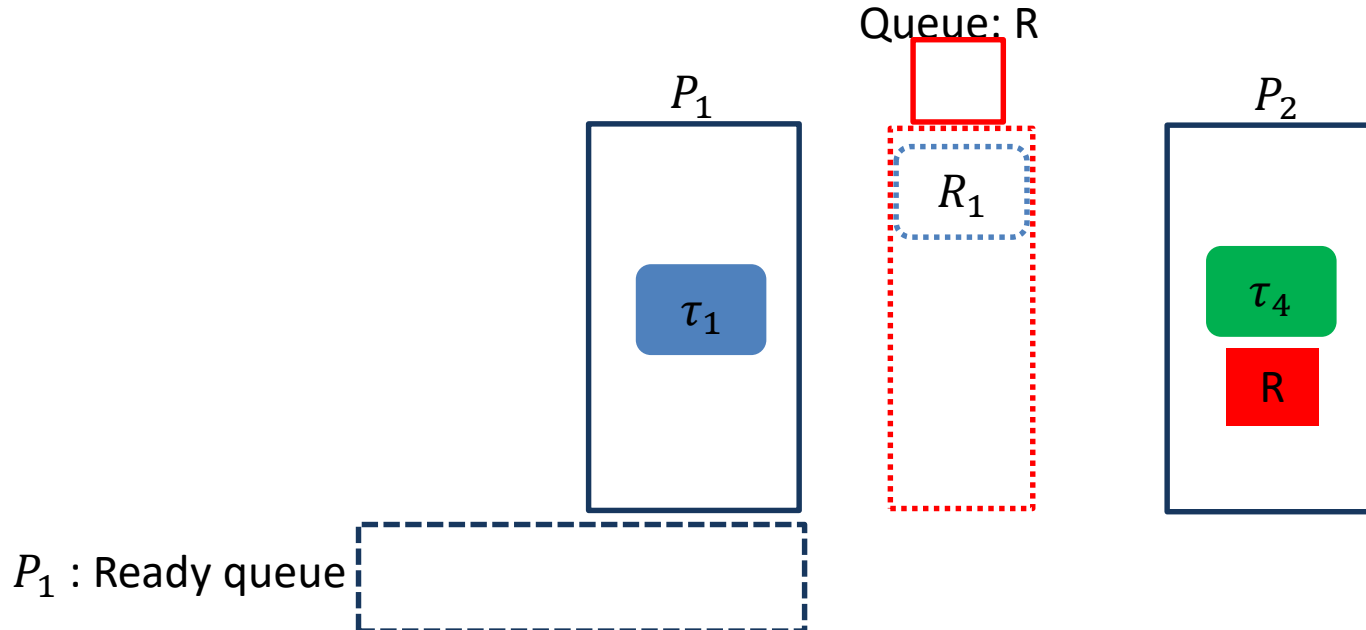
# Spin-Based Protocol

$\tau_1$  requests shared resource  $R$  from another processor



# Spin-Based Protocol

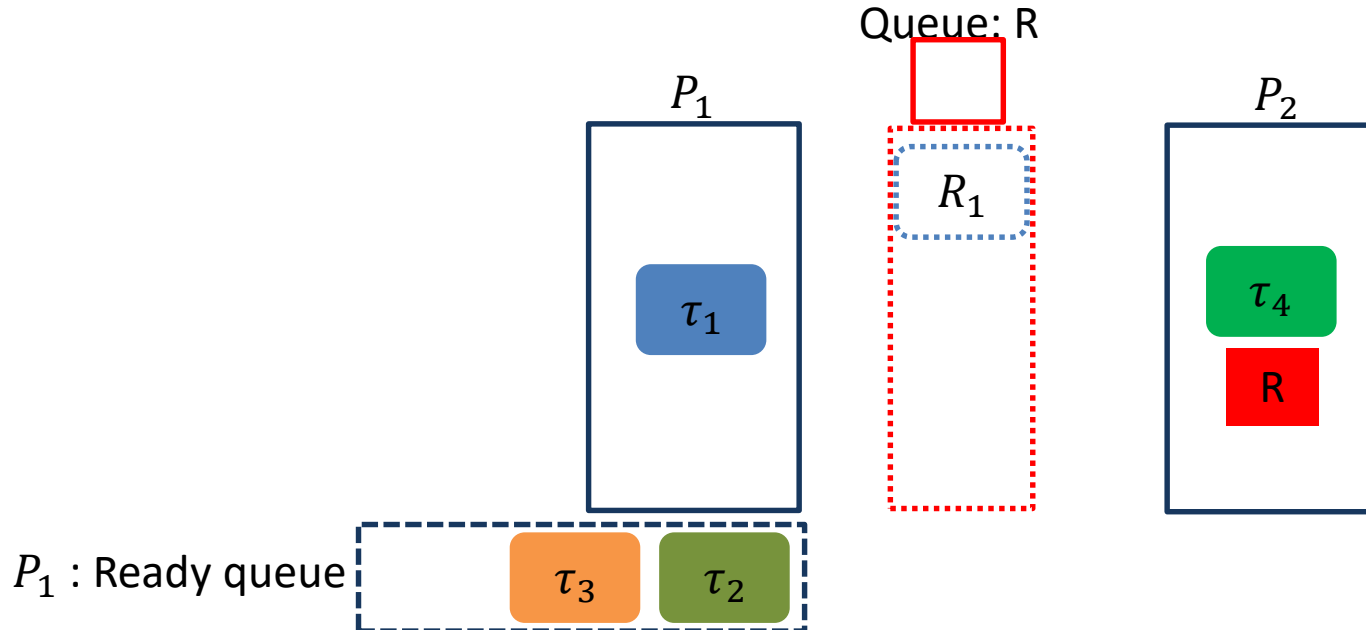
$\tau_1$  gets blocked on shared resource  $R$  since the resource is already locked and it puts its request in the  $R$ 's waiting queue





# Spin-Based Protocol

- $\tau_1$  disallows execution of any other local ready task on processor  $P_1$  since it is non-preemptively spinning



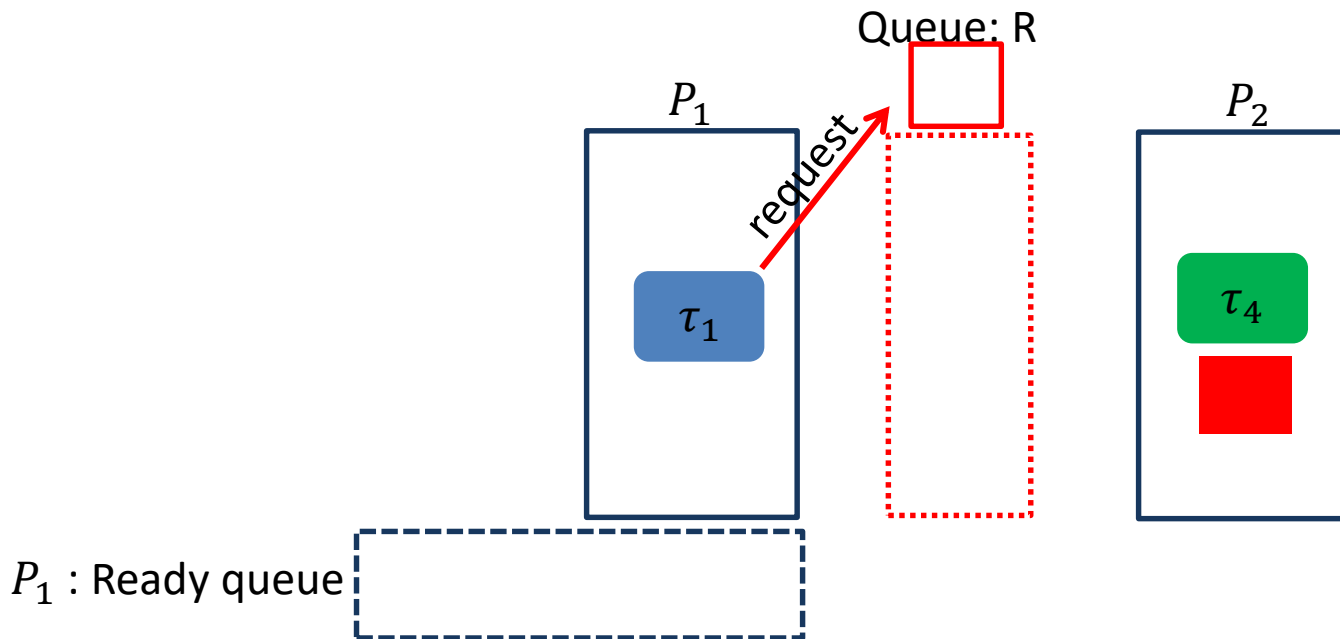


# Spin-Based Protocol

- *Advantage:*
  - *Task gets immediate access when the resource is granted*
  - *Prevents additional resource requests*
- *Disadvantage:*
  - *Waste processor cycles*
  - *Incurring extra blocking to higher priority tasks and extra interference to lower priority tasks*
- *MSRP and FMLP for short resources*

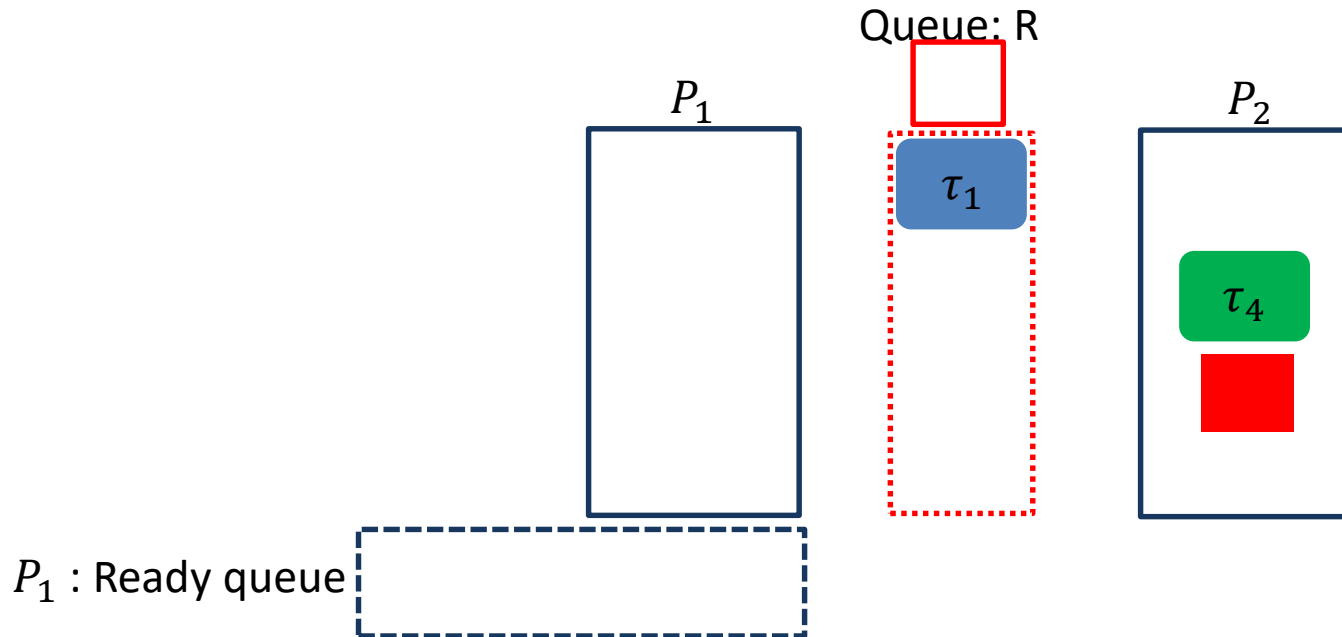
# Suspend-Based Protocol

- In a suspension-based protocol under the same example scenario  $\tau_1$  suspends and releases the processor when it gets blocked on resource  $R$  that is already locked by another task and stays in the waiting queue of  $R$



# Suspend-Based Protocol

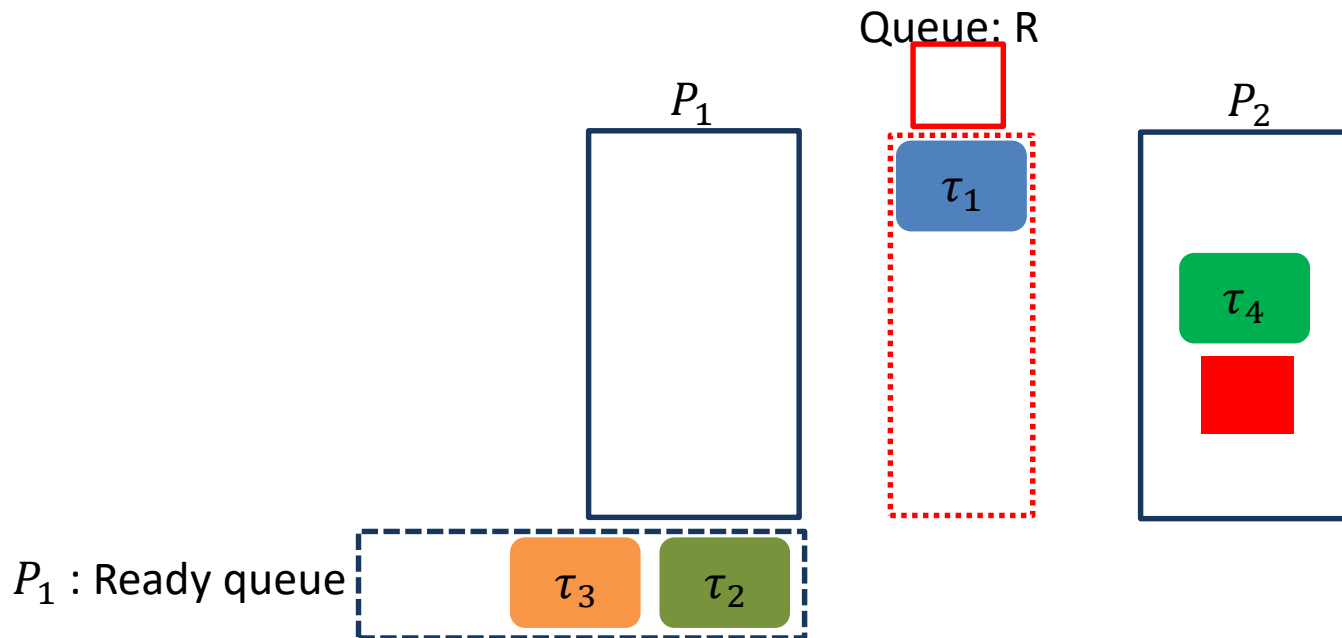
- In a suspension-based protocol under the same example scenario  $\tau_1$  suspends and releases the processor when it gets blocked on resource  $R$  that is already locked by another task and stays in the waiting queue of  $R$





# Suspend-Based Protocol

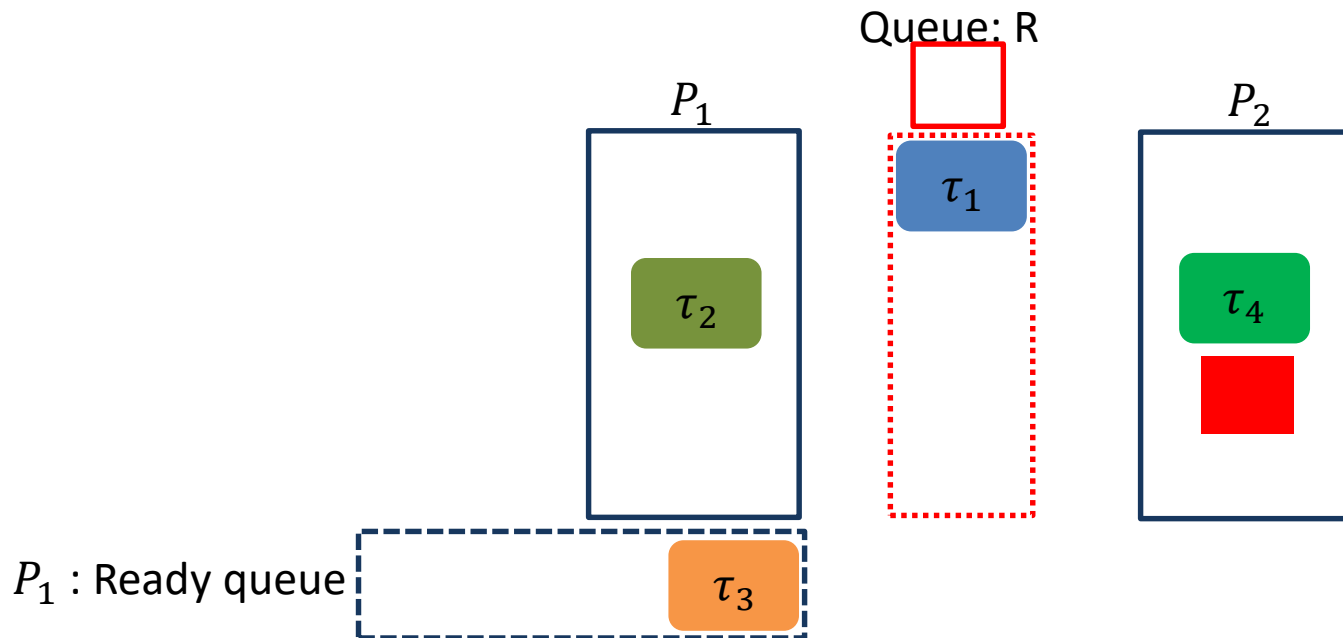
- Hence it allows execution of any other local ready tasks that arrive on the core such as  $\tau_2$  and  $\tau_3$





# Suspend-Based Protocol

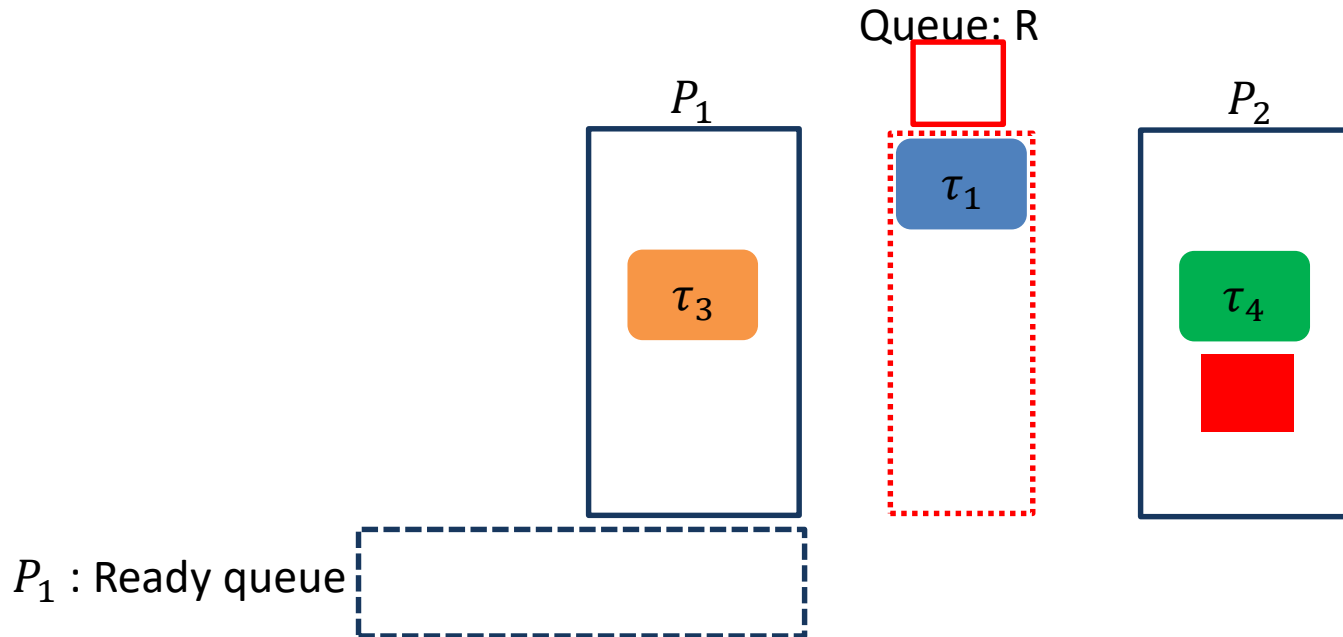
- Hence it allows execution of any other local ready tasks that arrive on the core such as  $\tau_2$  and  $\tau_3$





# Suspend-Based Protocol

- Hence it allows execution of any other local ready tasks that arrive on the core such as  $\tau_2$  and  $\tau_3$





# Suspend-Based Protocol

- *Advantage:*
  - *Processor cycle is not wasted*
  - *No extra blocking to higher or lower priority tasks*
- *Disadvantage:*
  - *Tasks may not get immediate access when the resource is granted*
  - *Allows additional resource requests*
- FMLP for long resources and MSOS





# Main Characteristics

- *Multiprocessor resource sharing characterized by:*
  - *Behavior upon remotely blocking*
    - *Non-preemptive Spinning: non-preemptive busy-wait loop*
    - *Suspension*
    - *Preemptive spinning*
  - *Queue type*
    - *FIFO-ordered*
    - *Priority-ordered*
    - *Unordered*
  - *Queueing policy upon preemption during spinning*
    - *Skipping: preempted task is skipped in the queue upon locking the resource*
    - *Dequeueing: preempted task is dequeued and requeued after resumption*
    - *Classical policy, preempted task stays in the queue in its position*



# Spin-Based Protocols

- *Multiprocessor Stack Resource Policy (MSRP)*
  - *Non-preemptive spin-based protocol*
  - *Extension of SRP*
  - *Partitioned-EDF*
  - *FIFO-based queues*
- *Multiprocessor Bandwidth Inheritance (M-BWI) protocol*
  - *Non-preemptive spin-based protocol*
  - *Extension of BWI: CBS servers + PIP*
  - *Existence of both soft and hard real-time tasks*
  - *Global scheduling*
  - *Suitable for open systems*
  - *FIFO-based queues*



# Spin-Based Protocols

- *Multiprocessor resource sharing Protocol (MrsP)*
  - *Preemptive spin-based protocol*
  - *Variant of PCP*
  - *Partitioned-FP*
  - *Helping/donation mechanism: spinning task can execute critical section of a remote preempted lock holder task*
  - *FIFO-based queues*



# Suspend-Based Protocols

- *Distributed Priority Ceiling Protocol (DPCP)*
  - *For distributed systems*
  - *Message passing*
  - *Global critical sections execute on special cores*
  - *Partitioned-FP (RM)*
  - *Priority-based queues*
- *Multiprocessor Priority Ceiling Protocol (MPCP)*
  - *Extension of PCP*
  - *Partitioned-FP (RM)*
  - *Priority-based queues*



# Suspend-Based Protocols

- *Flexible Multiprocessor Locking Protocol (FMLP)*
  - *Suspension-based for long resources*
  - *Spin-based for short resources*
  - *Global-EDF, Partitioned-EDF, PD<sup>2</sup>*
  - *FIFO-based queue*
- *Parallel PCP*
  - *Extension of PIP*
  - *Global-FP*



# Flexible Spin-Lock Model

- *Spin: spins on highest priority level* → (HP)
- *Suspend-based: spins on lowest priority level* → (LP)
- *Flexible spin-lock model (FSLM)*





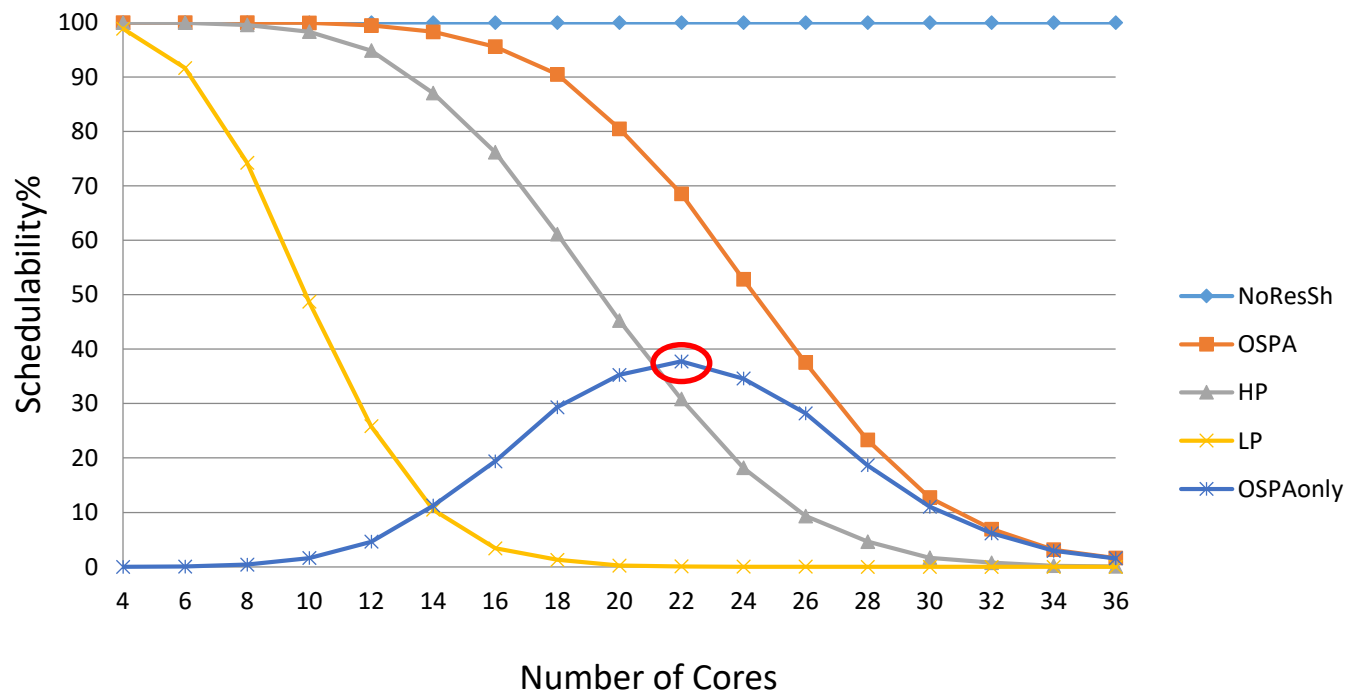
# Flexible Spin-Lock Model

- *Per core*
- *Per task*
- *Per resource*
- *Per request*
- *Combination of above*



# Spin-locks from FSLM

- *OSPA (Optimal Spin-Lock Priority Assignment) algorithm: per core*
  - *Improves HP, LP up to 40%*







# Open Issues

- *Processor utilization limits*
- *Ineffective schedulability tests*
- *Consideration of overheads*
- *Limited task models*
- *Limited resource sharing protocols*



# Related Areas of Research

- *Worst-case Execution Time (WCET) analysis*
- *Network / bus scheduling*
- *Memory architectures*
- *Uniform and heterogeneous processors*
- *Operating Systems*
- *Power consumption and dissipation*
- *Scheduling tasks with soft real-time constraints*
- *Non-real-time issues such as load balancing*



**THANK YOU!**