Property-Based Timing Analysis and Optimization for Complex Cyber-Physical Real-Time Systems

Jian-Jia Chen

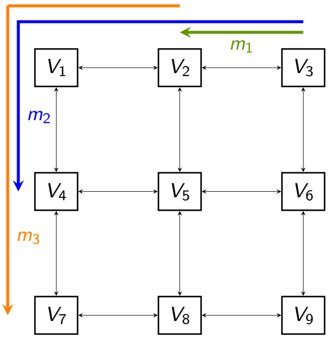
TU Dortmund

13.06.2023

Outline

Contents

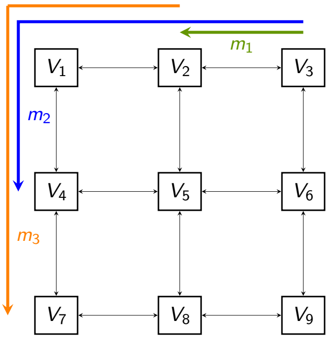
An Example of Classical Timing Analysis Flow



3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- Suppose to analyze lowest-priority message m_3
- Imagine that the path of m_3 is equivalent to a uniprocessor

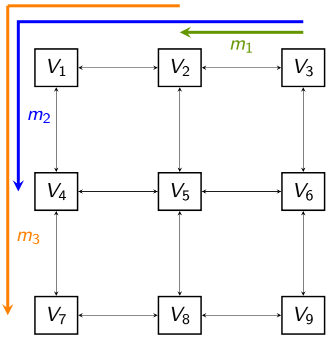
An Example of Classical Timing Analysis Flow



3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- Suppose to analyze lowest-priority message m_3
- Imagine that the path of m_3 is equivalent to a uniprocessor
    - 1994/1997: only consider m_2 (unsafe)

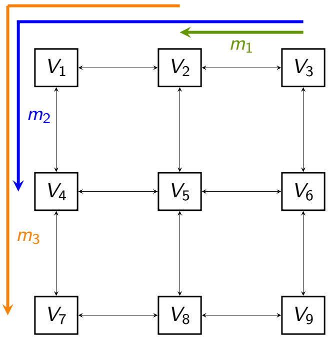
An Example of Classical Timing Analysis Flow



3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- Suppose to analyze lowest-priority message m_3
- Imagine that the path of m_3 is equivalent to a uniprocessor
  - 1994/1997: only consider m_2 (unsafe)
  - 1998-2015: m_1 pushes m_2 which introduces release jitters (unsafe)

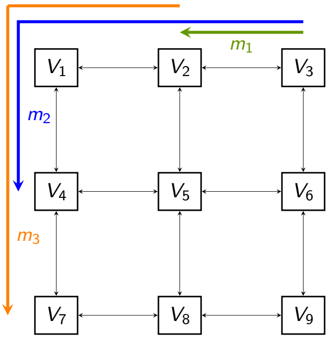
An Example of Classical Timing Analysis Flow



3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- Suppose to analyze lowest-priority message m_3
- Imagine that the path of m_3 is equivalent to a uniprocessor
  - 1994/1997: only consider m_2 (unsafe)
  - 1998-2015: m_1 pushes m_2 which introduces release jitters (unsafe)
  - 2016/2018: m_2 may interfere with m_3 more than its best transmission time (??)

An Example of Classical Timing Analysis Flow

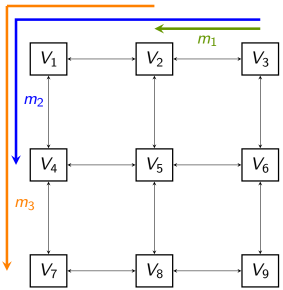


3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- Suppose to analyze lowest-priority message m_3
- Imagine that the path of m_3 is equivalent to a uniprocessor
  - 1994/1997: only consider m_2 (unsafe)
  - 1998-2015: m_1 pushes m_2 which introduces release jitters (unsafe)
  - 2016/2018: m_2 may interfere with m_3 more than its best transmission time (??)
- At least 8 papers introduced unsafe analyses

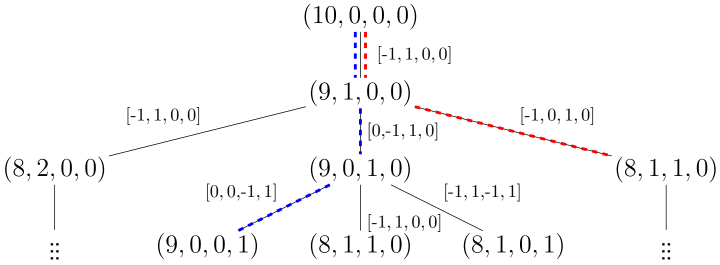
Timing Analysis Flow: Multiple Resources

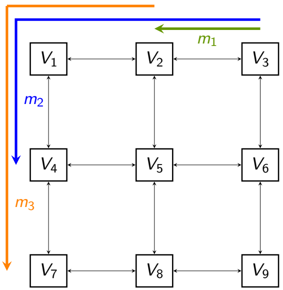
- Imagine that the path of m_3 is equivalent to a uniprocessor
- Is the imagination correct?



Timing Analysis Flow: Multiple Resources

- Imagine that the path of m_3 is equivalent to a uniprocessor
- Is the imagination correct?





Ueter, ..., Chen, ... in RTCSA. 2020

Motivation of PropRT: Property-Based Real-Time Analyses

- Ad-hoc solutions for one dedicated problem
  - **originally not** formulated with the goal of general applicability
  - **later applied** to deal with a wide range of problems
  - **later found to be misused** since the assumptions are not met

Motivation of PropRT: Property-Based Real-Time Analyses

- Ad-hoc solutions for one dedicated problem
  - **originally not** formulated with the goal of general applicability
  - **later applied** to deal with a wide range of problems
  - **later found to be misused** since the assumptions are not met
- An Example: critical instant theorem by Liu and Layland in JACM 1973
  - Original statement: *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks*
  - No formal statement regarding its applicability

European Research Council
Established by the European Commission

Motivation of PropRT: Property-Based Real-Time Analyses

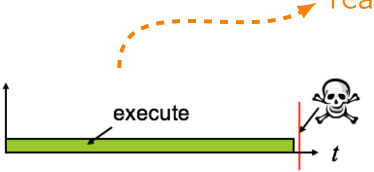
- Ad-hoc solutions for one dedicated problem
  - **originally not** formulated with the goal of general applicability
  - **later applied** to deal with a wide range of problems
  - **later found to be misused** since the assumptions are not met
- An Example: critical instant theorem by Liu and Layland in JACM 1973
  - Original statement: *A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks*
  - No formal statement regarding its applicability
- Misuse of critical instant theorem in the first four analyses of previous example

European Research Council
Established by the European Commission

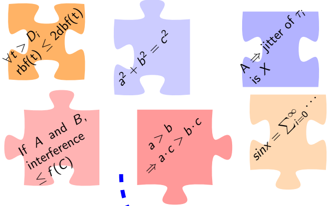
Property-Based Modulable Timing Analysis and Optimization

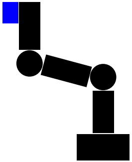
Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems

Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems

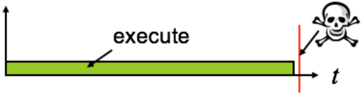


Property-Based Modulable Timing Analysis and Optimization

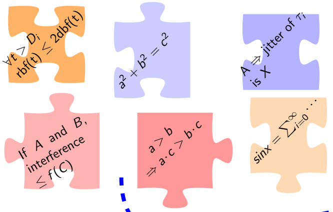


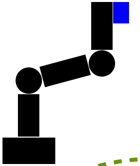


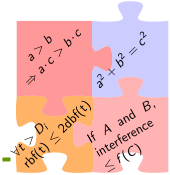
Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems



Property-Based Modulable Timing Analysis and Optimization

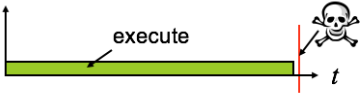




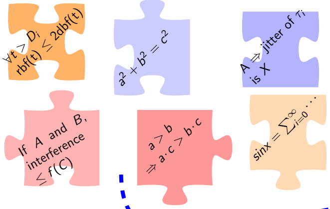


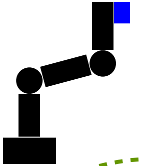
when Z holds,
the speedup factor is x

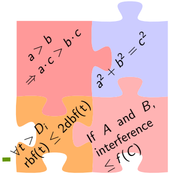
Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems



Property-Based Modulable Timing Analysis and Optimization

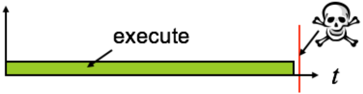




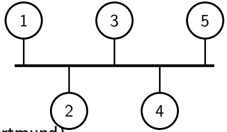


when Z holds,
the speedup factor is x

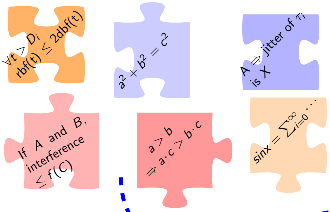
Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems

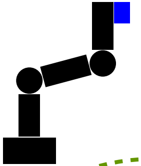


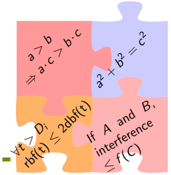
Thread A	Thread B
mutex.lock();	mutex.lock();
.....	.....



Property-Based Modulable Timing Analysis and Optimization



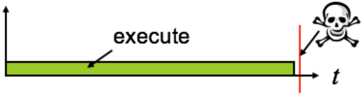


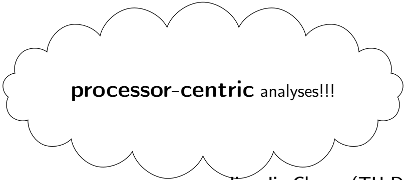


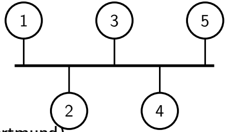
when Z holds,
the speedup factor is x

Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems

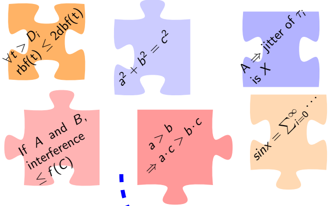
Thread A	Thread B
mutex.lock();	mutex.lock();
.....	.....

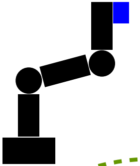


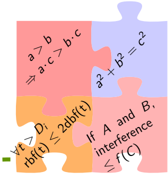




Property-Based Modulable Timing Analysis and Optimization



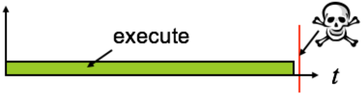


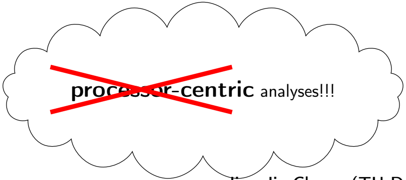


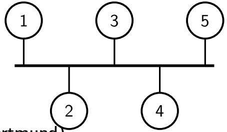
when Z holds,
the speedup factor is x

Radically new
property-based and modulable
foundation for complex
real-time cyber-physical systems

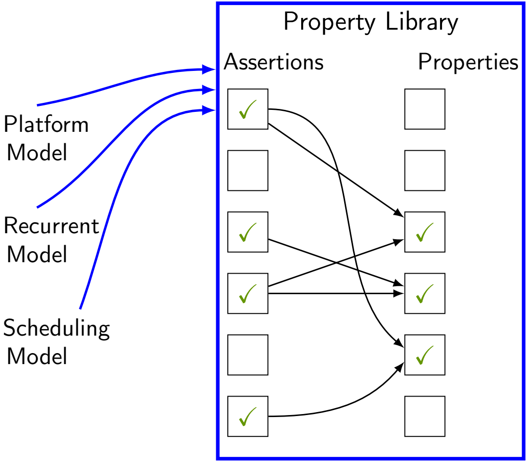
Thread A	Thread B
mutex.lock();	mutex.lock();
.....	.....



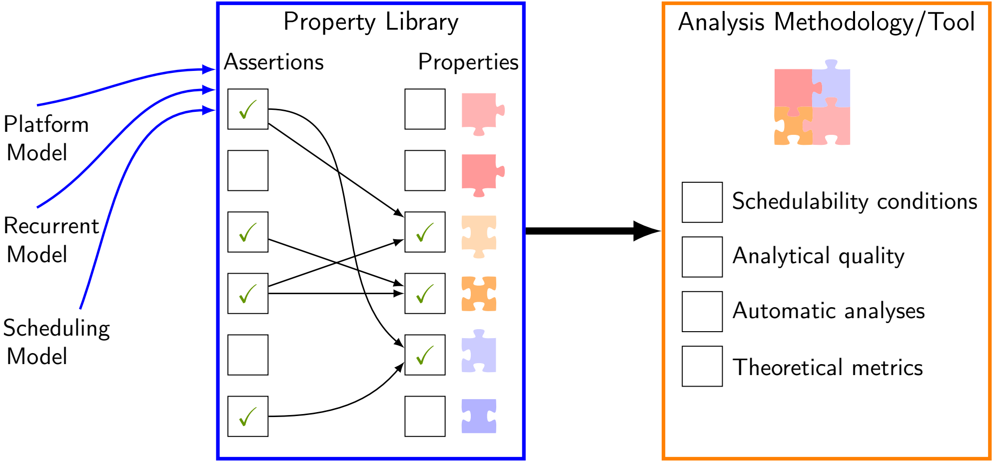




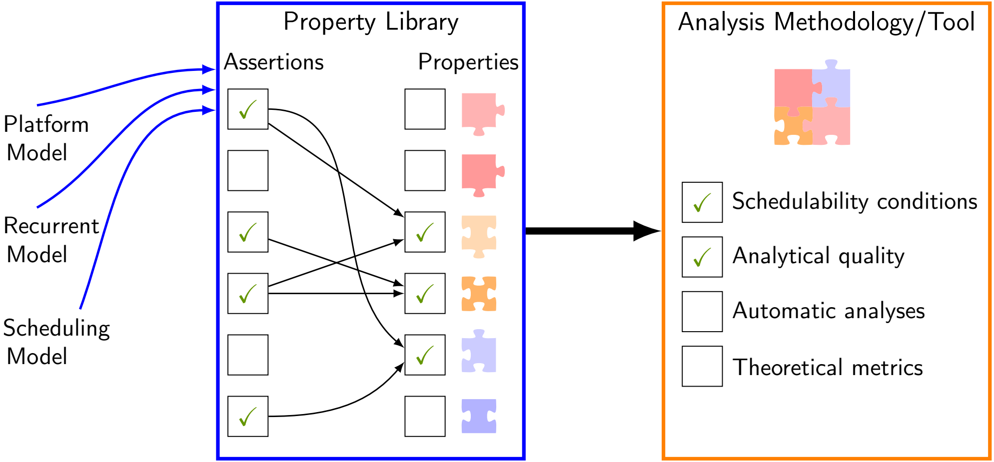
Library of Properties



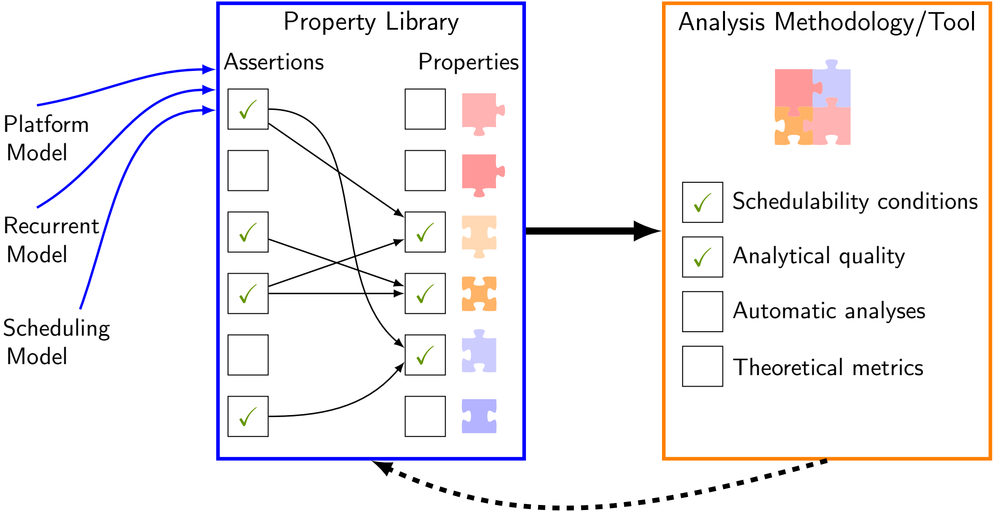
Library of Properties



Library of Properties



Library of Properties

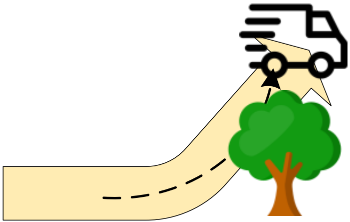


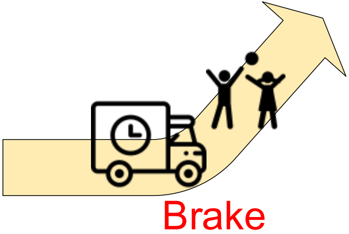
Goals of PropRT

- *Formal properties* to modularly compose real-time embedded systems
- *Methodologies* for generating/verifying properties and modular compositions
- *Predictable interplay* of computation, communication, and synchronization for complex real-time embedded systems

Goals of PropRT

- *Formal properties* to modularly compose real-time embedded systems
- *Methodologies* for generating/verifying properties and modular compositions
- *Predictable interplay* of computation, communication, and synchronization for complex real-time embedded systems

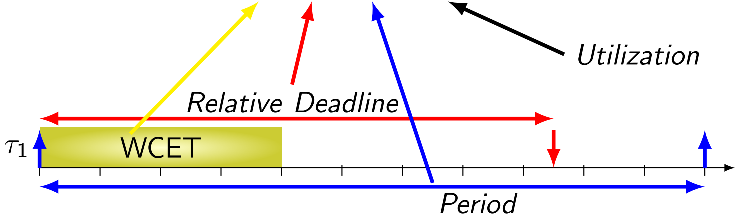




Contents

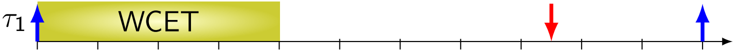
Periodic Task System Model

$$\tau_i(C_i,\ D_i,\ T_i),\ U_i = \frac{C_i}{T_i}$$



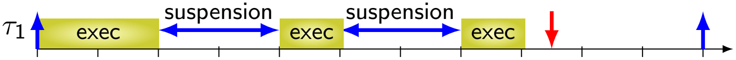
Periodic Task System Model

$$\tau_i(C_i,\ D_i,\ T_i),\ U_i = \frac{C_i}{T_i}$$



Periodic Task System Model

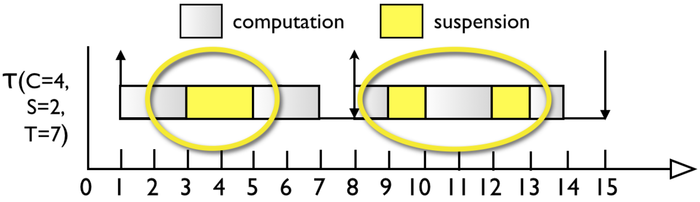
$$\tau_i(C_i,\ D_i,\ T_i),\ U_i = \frac{C_i}{T_i}$$



Suspension time: S_i

Possible Self Suspensions

Implicit-deadline sporadic suspending task: τ(C, S, T)



Jobs may alternate between computation and suspension phases without any restriction on how they interleave

- 1-Segmented self-suspension: 2 computation segments separated by a suspension interval
- Segmented self-suspension: f computation segments separated by $f-1$ suspension intervals
- Dynamic self-suspension: the suspension pattern is unknown and can be arbitrary

Reasons for Suspension: Computation Offloading

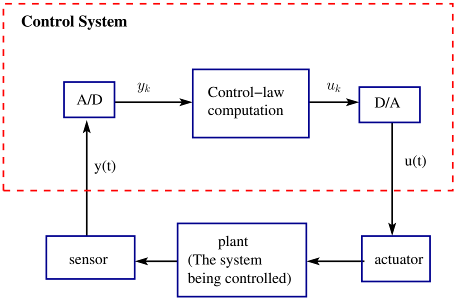
Pseudo-code for this system

set timer to interrupt periodically with period T;

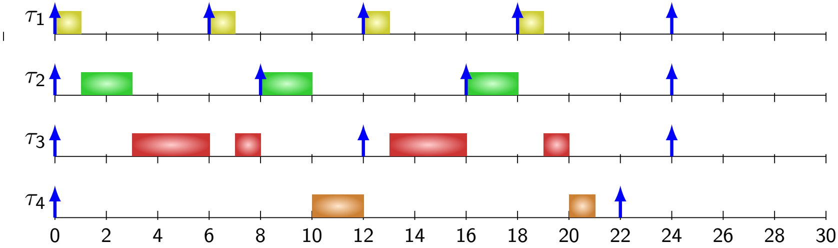
at each timer interrupt
do

- perform analog-to-digital conversion to get y;
- compute control output u by using accelerators;
- output u and do digital-to-analog conversion;

od



The Golden Critical Instant Theorem (without Suspension)



- Release the higher-priority tasks at the same time as task (i.e., τ_k) under analysis
- The following jobs of a higher-priority task should be released then by following the period constraint

$$\exists t | 0 < t \leq D_k \;\; s.t. \;\; C_k + \sum_{\tau_j \in higher\_priority(\tau_k)} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$

Suspension Induces Jitter under Fixed-Priority

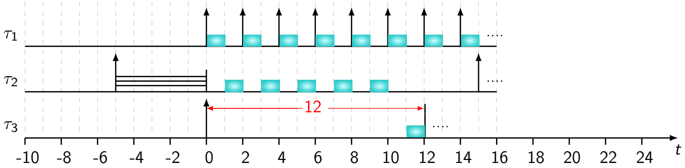
Schedulability test of task τ_k:

$$\exists t | 0 < t \leq D_k \quad \text{s.t.} \quad C_k \qquad + \sum_{\tau_j \in higher\_priority(\tau_k)} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$

Suspension Induces Jitter under Fixed-Priority

Schedulability test of task τ_k:

$$\exists t | 0 < t \leq D_k \quad \text{s.t.} \quad C_k + S_k + \sum_{\tau_j \in higher\_priority(\tau_k)} \left\lceil \frac{t + S_j}{T_j} \right\rceil C_j \leq t.$$

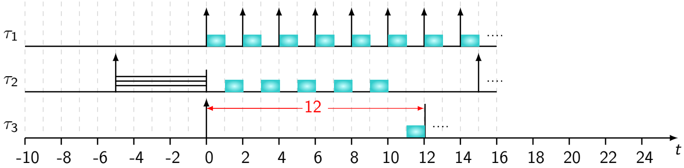


τ_i	C_i	S_i	T_i
τ_1	1	0	2
τ_2	5	5	20
τ_3	1	0	∞

Suspension Induces Jitter under Fixed-Priority

Schedulability test of task τ_k:

$$\exists t | 0 < t \leq D_k \quad \text{s.t.} \quad C_k + S_k + \sum_{\tau_j \in higher\_priority(\tau_k)} \left\lceil \frac{t + S_j}{T_j} \right\rceil C_j \leq t.$$



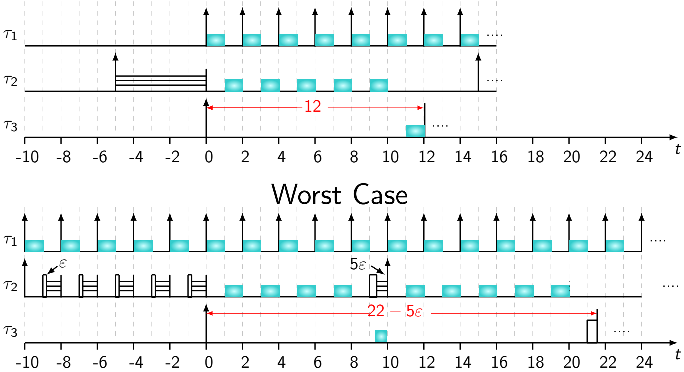
τ_i	C_i	S_i	T_i
τ_1	1	0	2
τ_2	5	5	20
τ_3	1	0	∞

Suspension Induces Jitter under Fixed-Priority

Schedulability test of task τ_k:

$$\exists t | 0 < t \leq D_k \quad \text{s.t.} \quad C_k + S_k + \sum_{\tau_j \in higher\_priority(\tau_k)} \left\lceil \frac{t + S_j}{T_j} \right\rceil C_j \leq t.$$

τ_i	C_i	S_i	T_i
τ_1	1	0	2
τ_2	5	5	20
τ_3	1	0	∞



Literature of Self-Suspension (before 2013)

- Simplest method: convert suspension into computation in the analysis

Literature of Self-Suspension (before 2013)

- Simplest method: convert suspension into computation in the analysis
- Ridouard et al. in RTSS 2004
  - Uniprocessor scheduling of self-suspending tasks is NP-hard in the strong sense

Literature of Self-Suspension (before 2013)

- Simplest method: convert suspension into computation in the analysis
- Ridouard et al. in RTSS 2004
  - Uniprocessor scheduling of self-suspending tasks is NP-hard in the strong sense
- Bletsas and Audsley in RTAS 2004 (flawed), ECRTS 2004 (flawed), RTCSA 2005 (flawed)
- Lakshmanan et al. in RTAS 2010 (flawed)
- Kim et al. in RTSS 2013 (flawed)
- Ding et al. 2009 (flawed)
- Meng RTCSA 1994 (flawed)
- Kim et al. in RTCSA 1995 (flawed)
- Rajkumar IBM report 1991 (inconclusive)

My Contribution for Systems with Self Suspensions

- A summary of misconceptions in the literature (Real-Time Systems Journal 2019)
- 15+ technical papers
  - Publications detailing how to safely analyze the timing properties for different self-suspension models
  - Publications with different methods and models for improving the scheduling quality
  - Computational complexity analysis

My Contribution for Systems with Self Suspensions

- A summary of misconceptions in the literature (Real-Time Systems Journal 2019)
- 15+ technical papers
  - Publications detailing how to safely analyze the timing properties for different self-suspension models
  - Publications with different methods and models for improving the scheduling quality
  - Computational complexity analysis
- Extension for resource-centric analyses
  - Sharing of memory or bus (DAC 2016) and GPUs (RTAS 2018)
  - NoC schedulability analysis (RTCSA 2020)
  - Response time analysis for deferrable servers (ECRTS 2022)

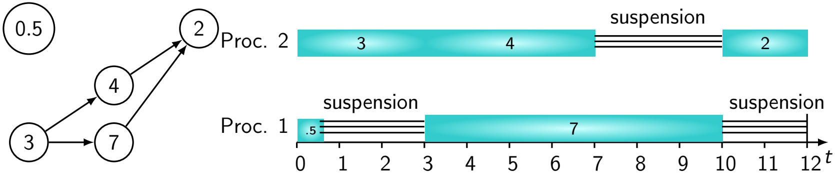
My Contribution for Systems with Self Suspensions

- A summary of misconceptions in the literature (Real-Time Systems Journal 2019)
- 15+ technical papers
  - Publications detailing how to safely analyze the timing properties for different self-suspension models
  - Publications with different methods and models for improving the scheduling quality
  - Computational complexity analysis
- Extension for resource-centric analyses
  - Sharing of memory or bus (DAC 2016) and GPUs (RTAS 2018)
  - NoC schedulability analysis (RTCSA 2020)
  - Response time analysis for deferrable servers (ECRTS 2022)
- Extension for DAG scheduling and analysis
  - Gang scheduling (ECRTS 2021)
  - Type-aware scheduling (IEEE TC 2023)

My Contribution for Systems with Self Suspensions

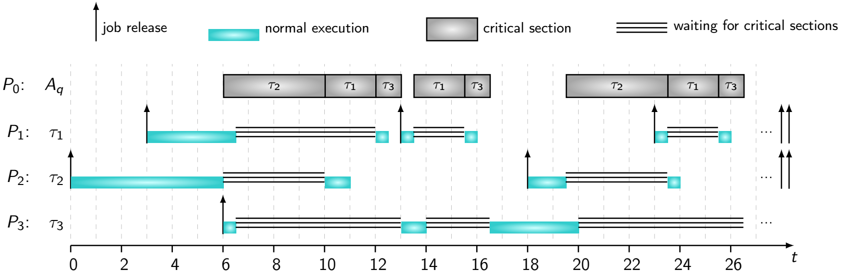
- A summary of misconceptions in the literature (Real-Time Systems Journal 2019)
- 15+ technical papers
  - Publications detailing how to safely analyze the timing properties for different self-suspension models
  - Publications with different methods and models for improving the scheduling quality
  - Computational complexity analysis
- Extension for resource-centric analyses
  - Sharing of memory or bus (DAC 2016) and GPUs (RTAS 2018)
  - NoC schedulability analysis (RTCSA 2020)
  - Response time analysis for deferrable servers (ECRTS 2022)
- Extension for DAG scheduling and analysis
  - Gang scheduling (ECRTS 2021)
  - Type-aware scheduling (IEEE TC 2023)
- Extension in multiprocessor locking protocols
  - Resource-oriented partitioned scheduling (RTSS)
  - Dependency-graph approach (RTSS 2018, RTAS 2019)

Reasons for Suspension: DAG Structure



- A task may be parallelized such that it can be executed simultaneously on some processors to perform independent computation
- To this end, we can use a *directed acyclic graph (DAG)* to model the dependency of the subtasks in a sporadic task
- Each vertex in the DAG represents a subtask

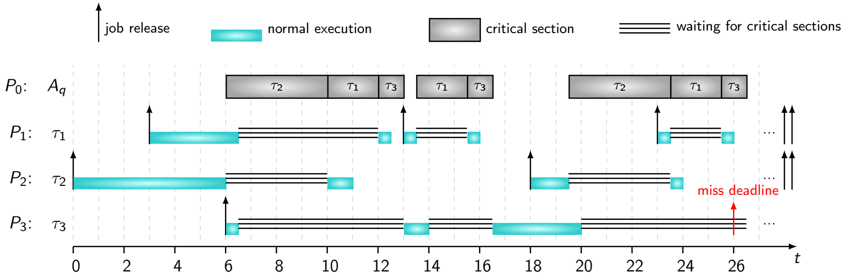
Reasons for Self-Suspensions: Locking Protocols



- Distributed PCP in the above example
- Semaphores in multiprocessor systems: remote blocking due to mutual exclusion

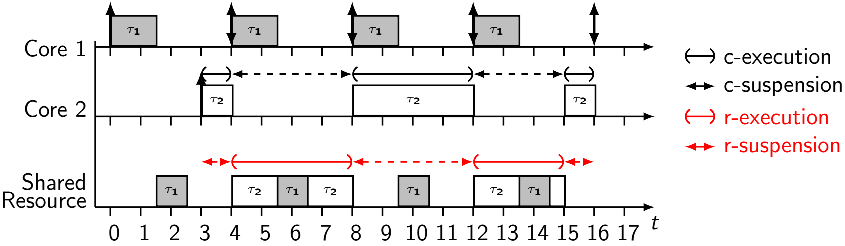
Does Spinning Avoid Self-Suspension?

τ_i	$Proc(\tau_i)$	C_i	$T_i\ (= D_i)$	N_k	L_i
τ_1	$Proc_1$	6	10	1	2
τ_2	$Proc_2$	11	18	1	4
τ_3	$Proc_3$	8	20	3	1



A job of task τ_3: run 0.5 time unit on $Proc_3$, critical section 1 time unit, run 1 time unit on $Proc_3$, access the critical section for 1 time unit, run 3.5 time units on $Proc_3$, and access the critical section for 1 time unit

Reasons for Self-Suspensions: Physical Resource Sharing

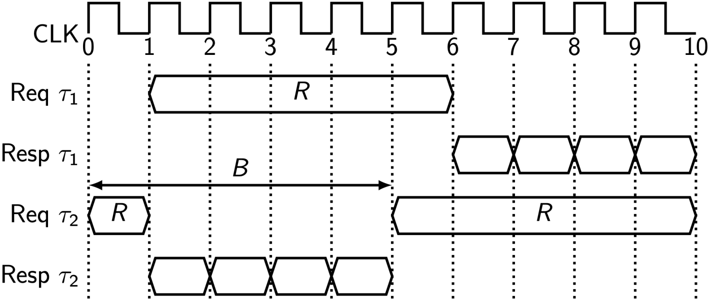


- Multiple cores may share a bus
- The contention on the bus can be considered as a suspension problem (with respect to the bus access)

Contents

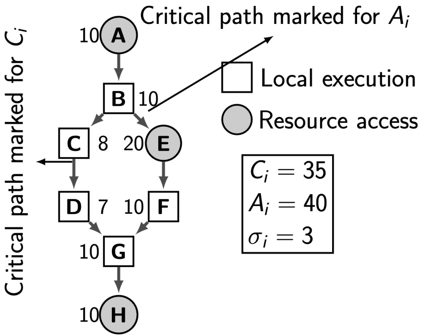
Platform Model

- Multicore with a share resource
- For example, atomic (non-split-transaction) bus
  - Bus sits idle while memory processes the request and sends the response
- Fixed-priority arbitration



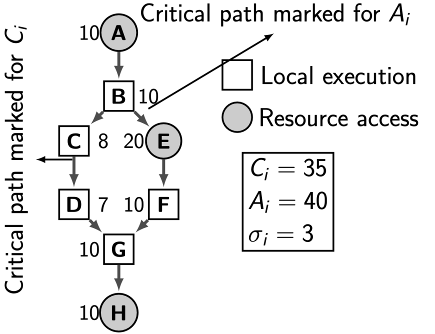
Task and Scheduling Model

- Resource access task τ_i $(C_i, A_i, T_i, D_i, \sigma_i)$
  - C_i: upper bound on local computation
  - A_i: upper bound on resource accesses
  - T_i: minimum inter-arrival time
  - D_i: relative deadline ($D_i \leq T_i$)
  - σ_i: the maximum number segments of consecutive resource accesses
- Path analysis
- Fixed-priority scheduling (we use deadline-monotinic scheduling)



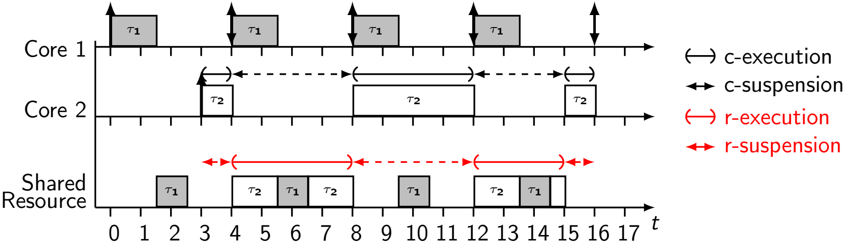
Task and Scheduling Model

- Resource access task τ_i $(C_i, A_i, T_i, D_i, \sigma_i)$
  - C_i: upper bound on local computation
  - A_i: upper bound on resource accesses
  - T_i: minimum inter-arrival time
  - D_i: relative deadline ($D_i \leq T_i$)
  - σ_i: the maximum number segments of consecutive resource accesses
- Path analysis
- Fixed-priority scheduling (we use deadline-monotinic scheduling)



Assume compositional properties: 75 is a safe upper bound.

Key Observations: Symmetric Property



- From the core perspectives for τ_2
  - accessing or waiting: [3,4), [8,12), [15, 16)
  - suspension: [4,8), [12, 15)
- From the shared resource perspectives for τ_2
  - executing or waiting: [4,8), [12, 15)
  - suspension:[3,4), [8,12), [15, 16)

Schedulability Test for Task τ_k

- WCRT is upper bounded by the minimum $t | 0 < t \leq D_k$

$$(C_k + exec\_core(t)) + (A_k + exec\_resource(t)) \leq t$$

Schedulability Test for Task τ_k

- WCRT is upper bounded by the minimum $t | 0 < t \leq D_k$

$$\left(C_k + \sum_{\tau_i \in hp(\tau_k, c)} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i \right) + \sigma_k B + \left(A_k + \sum_{\tau_i \in hp(\tau_k, r)} \left\lceil \frac{t + T_i}{T_i} \right\rceil A_i \right) \leq t$$

- σ_k B: the maximum blocking time by the lower priority tasks on the shared resource
- hp(τ_k, c): higher-priority tasks thanτ_k on the same core
- hp(τ_k, r): higher-priority tasks than τ_k on shared resource

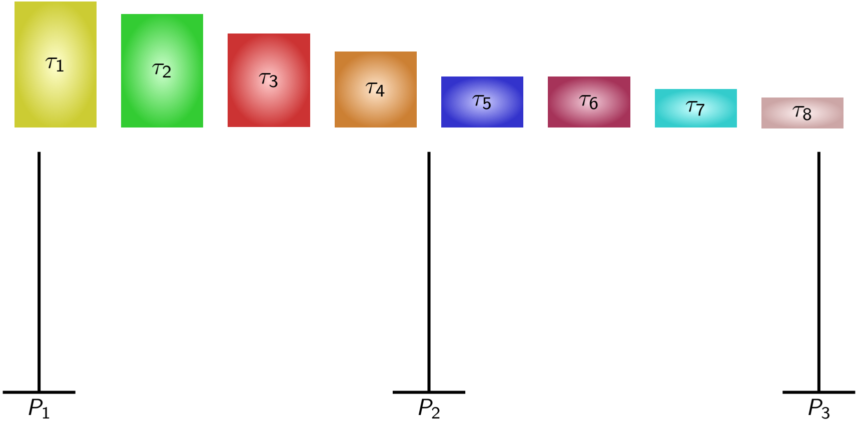
Schedulability Test for Task τ_k

- WCRT is upper bounded by the minimum $t | 0 < t \leq D_k$

$$\left(C_k + \sum_{\tau_i \in hp(\tau_k, c)} \left\lceil \frac{t + T_i}{T_i} \right\rceil C_i \right) + \sigma_k B + \left(A_k + \sum_{\tau_i \in hp(\tau_k, r)} \left\lceil \frac{t + T_i}{T_i} \right\rceil A_i \right) \leq t$$

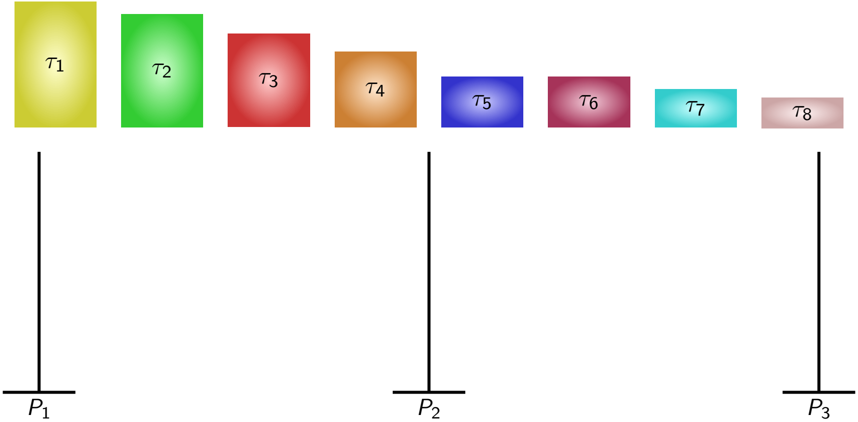
- σ_k B: the maximum blocking time by the lower priority tasks on the shared resource
- hp(τ_k, c): higher-priority tasks thanτ_k on the same core
- hp(τ_k, r): higher-priority tasks than τ_k on shared resource
- Pessimism of the above response time analysis: number of resource access segments was not exploited
- In our paper, we explain how to calculate and utilize the information σ_k in a symmetric and more precise manner

Task Assignment (Partition)



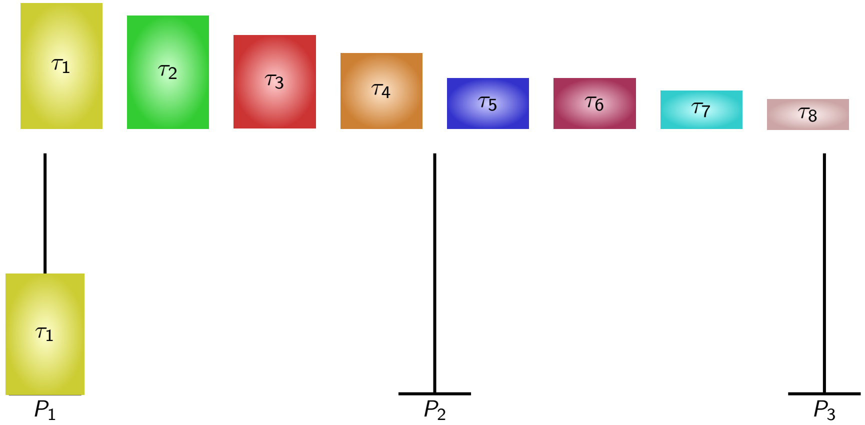
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



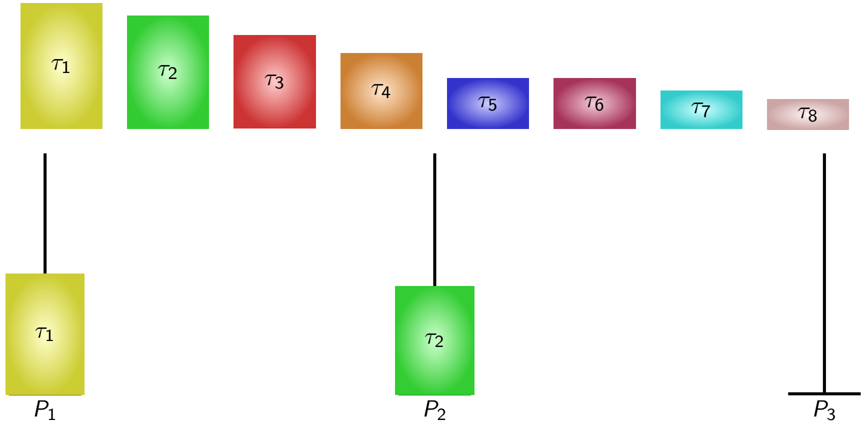
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



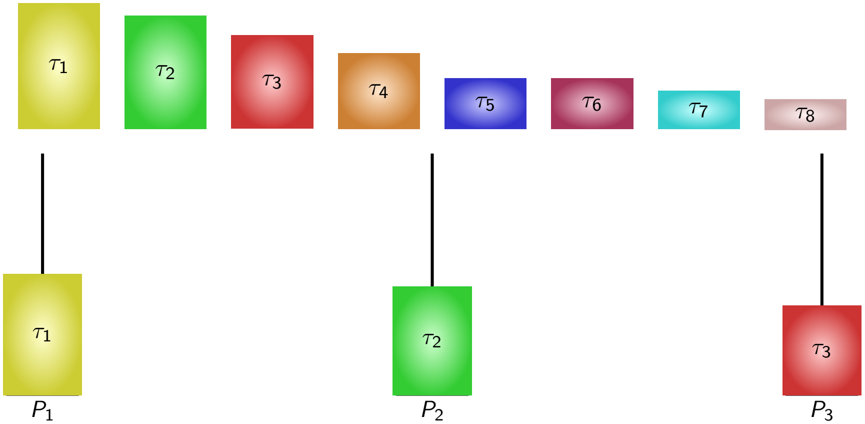
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



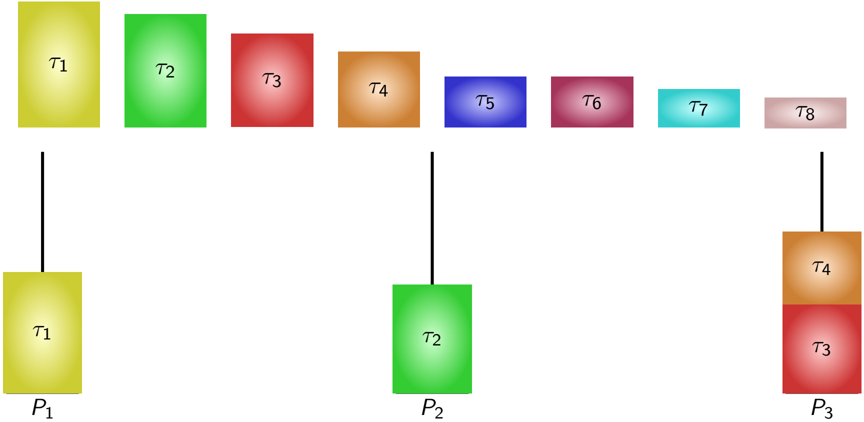
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



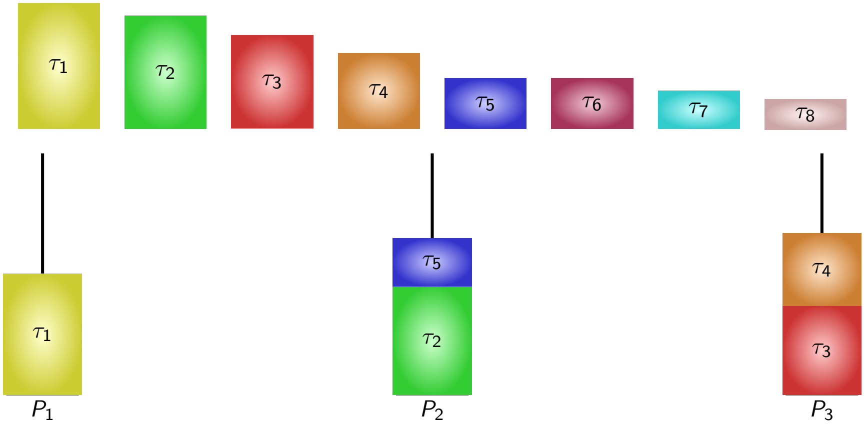
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



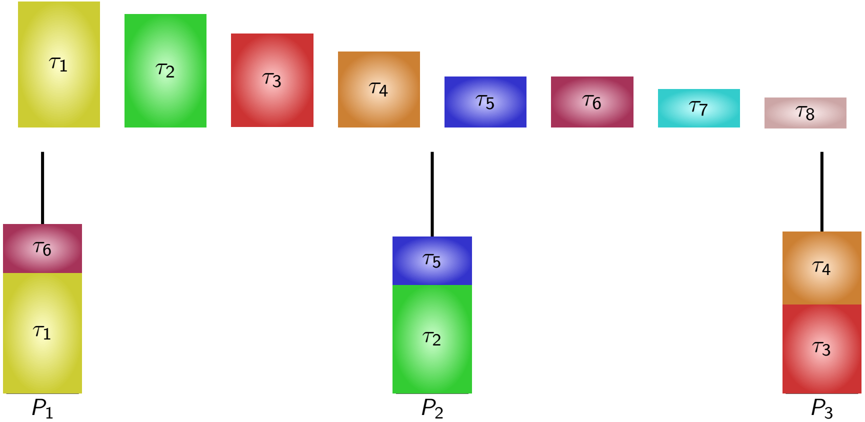
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



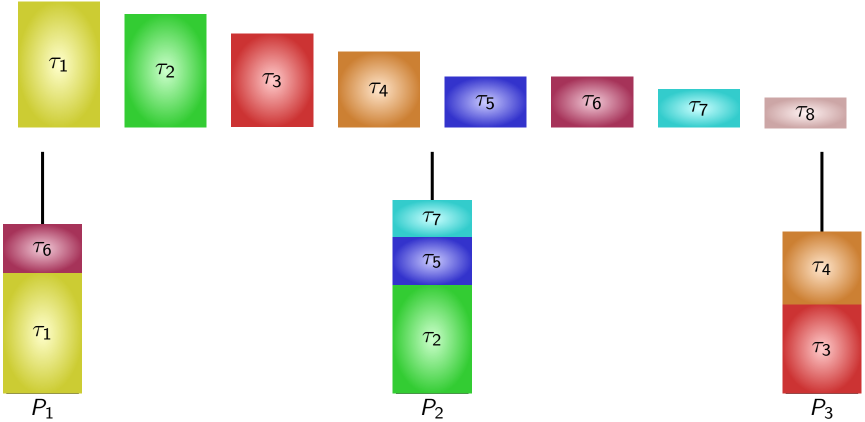
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



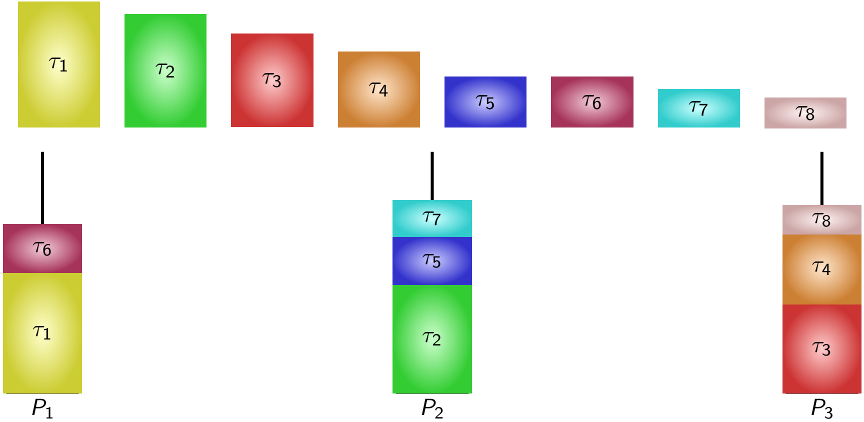
- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

Task Assignment (Partition)



- Schedulability tests are based on the previous slide.
- Fitting can be First-Fit (FF), Worst-Fit (WF), Best-Fit (BF)

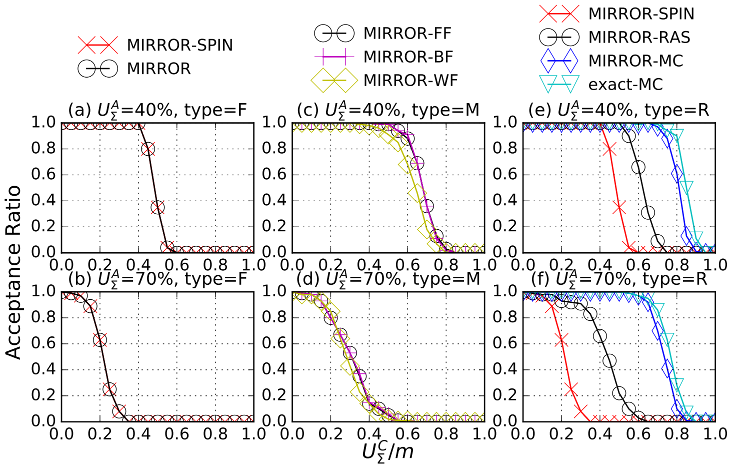
Experiments

- Configuration
  - 4-core platform (m=4)
  - 20 tasks
  - Periods [10-1000ms]
  - Each utilization level:100 task sets
- Comparison:
  - Exact-MC (Bonifaci et al. in RTNS 2015): do memory access and then do execution
  - MIRROR-SPIN (This resembles the test from Altmeyer et al. in RTNS 2015)
- Evaluation Metrics:
  - The acceptance ratio of a level: the number of task sets that are schedulable by the test divided by the number of task sets.

Experiments

Resource access segments σ_i:

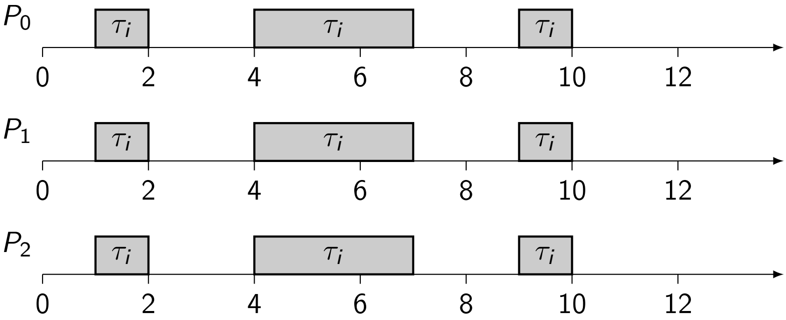
- 1 (rare access, type=R),
- 2 (moderate access, type=M),
- 10 (frequent access, type=F).



Contents

Gang Scheduling

A set of threads is grouped together into a *gang* s.t. they must be *co-scheduled at the same time*

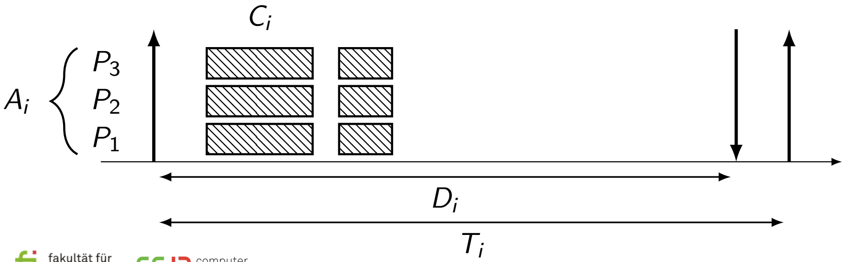


Gang Task Model

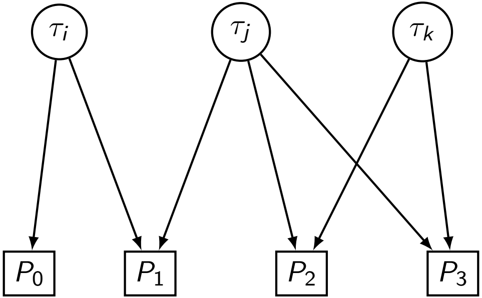
Definition

[A *sporadic constrained-deadline gang task* τ_i]

- WCET: C_i
- Gang size: E_i
- Relative deadline: $D_i \leq T_i$
- Minimal inter-arrival time: T_i

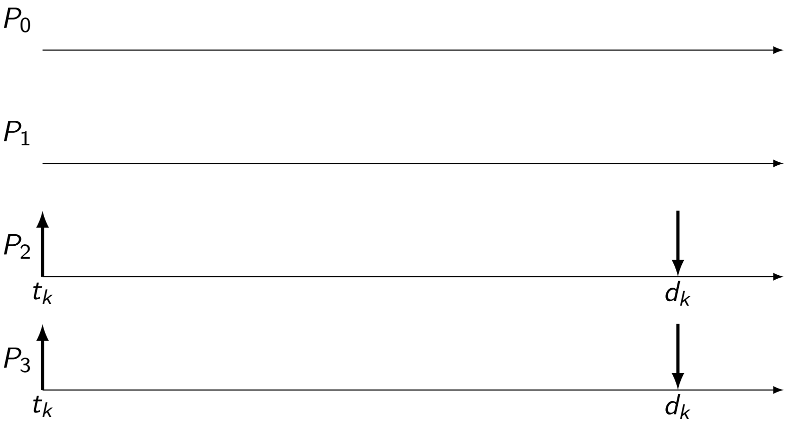


Exemplary Stationary Gang Assignment

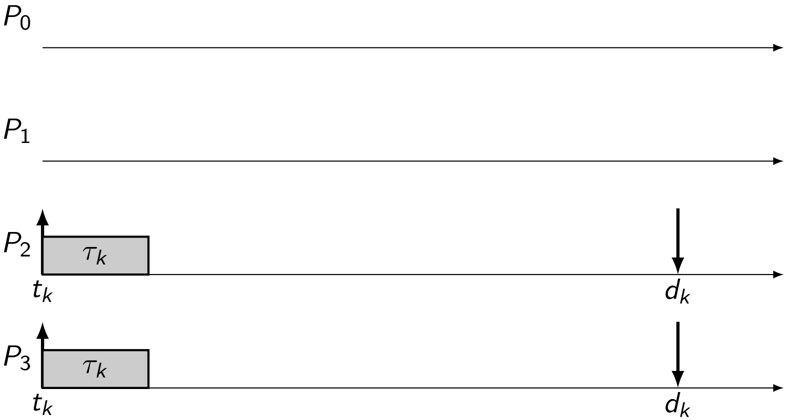


Fixed-Priority Stationary Gang Schedule

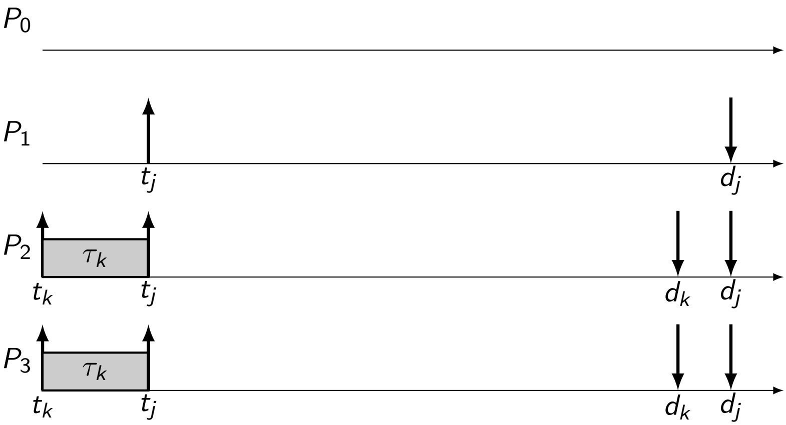
Fixed-Priority Stationary Gang Schedule



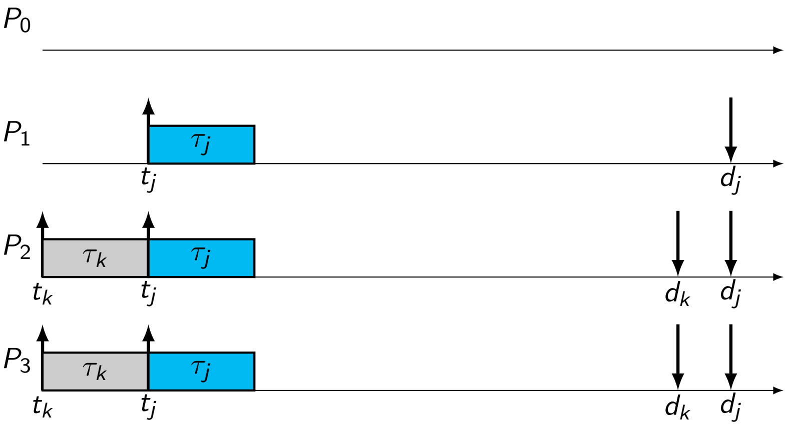
Fixed-Priority Stationary Gang Schedule



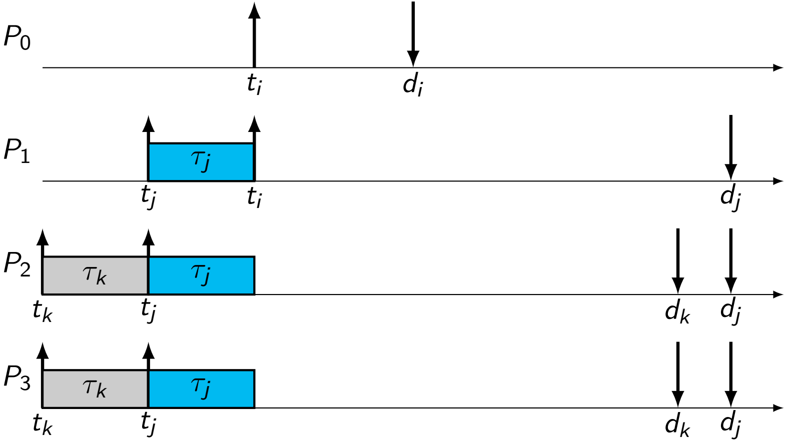
Fixed-Priority Stationary Gang Schedule



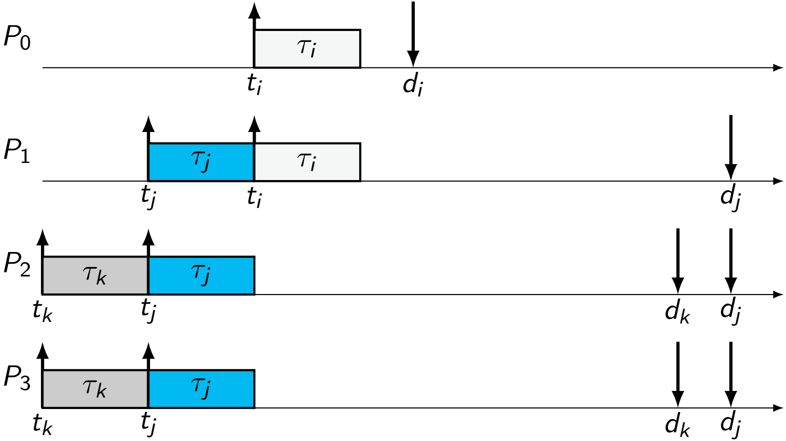
Fixed-Priority Stationary Gang Schedule



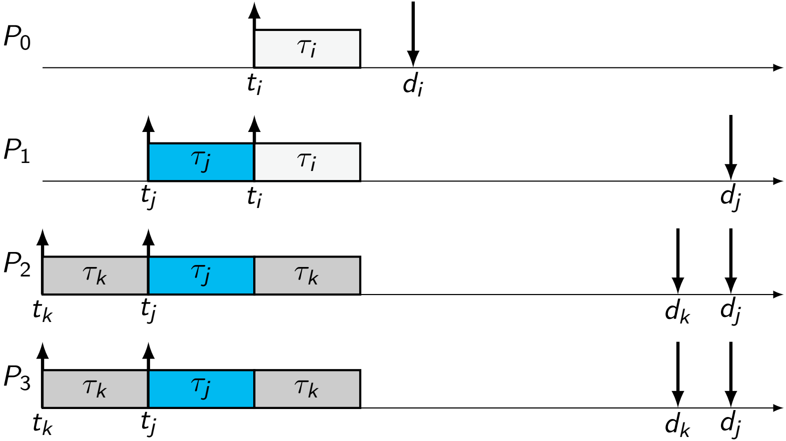
Fixed-Priority Stationary Gang Schedule



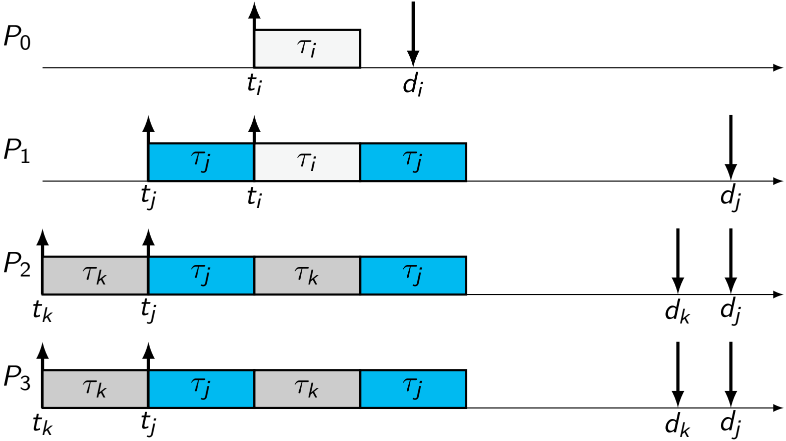
Fixed-Priority Stationary Gang Schedule



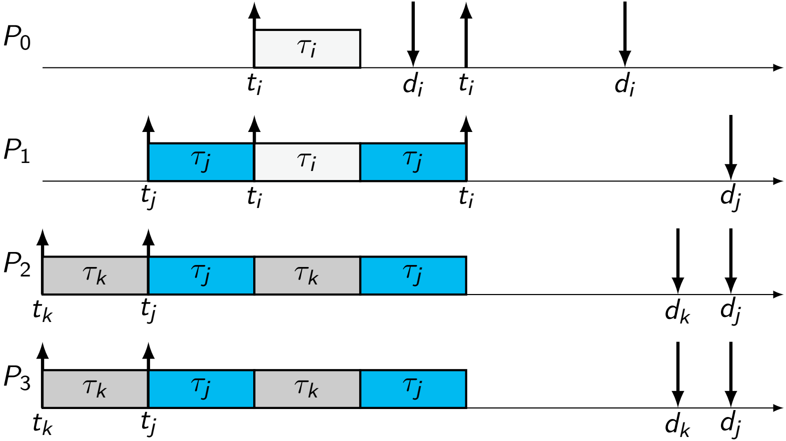
Fixed-Priority Stationary Gang Schedule



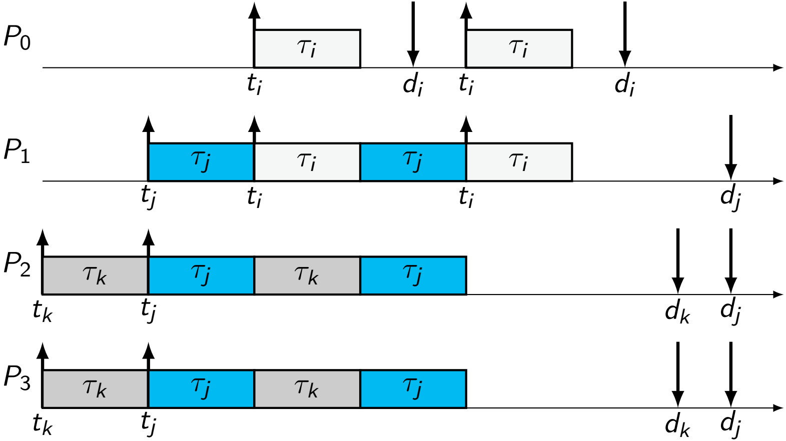
Fixed-Priority Stationary Gang Schedule



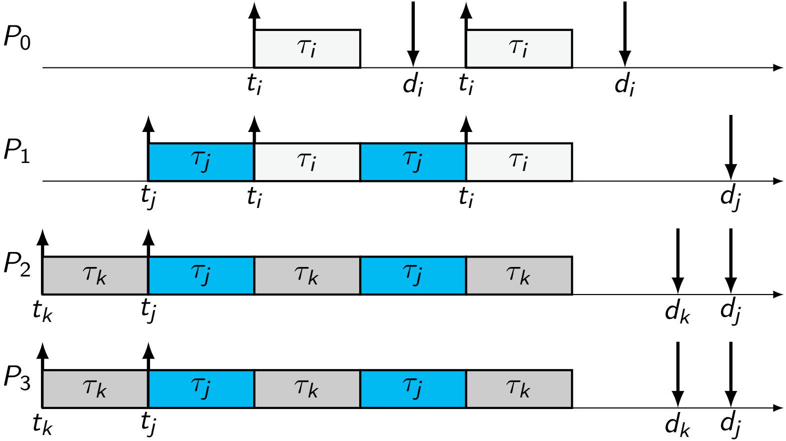
Fixed-Priority Stationary Gang Schedule



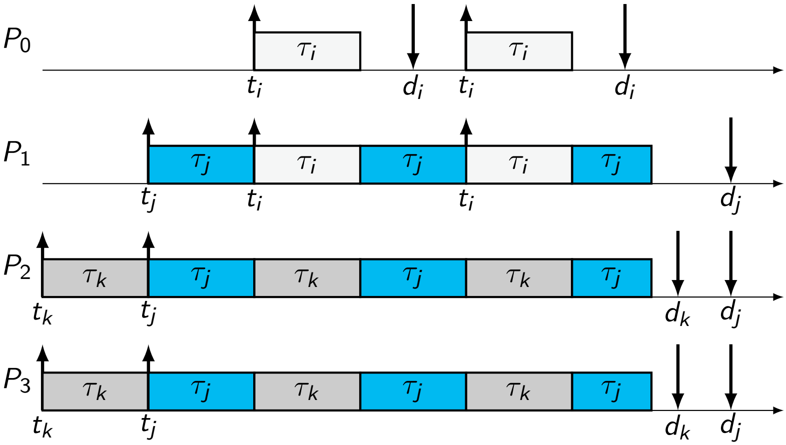
Fixed-Priority Stationary Gang Schedule



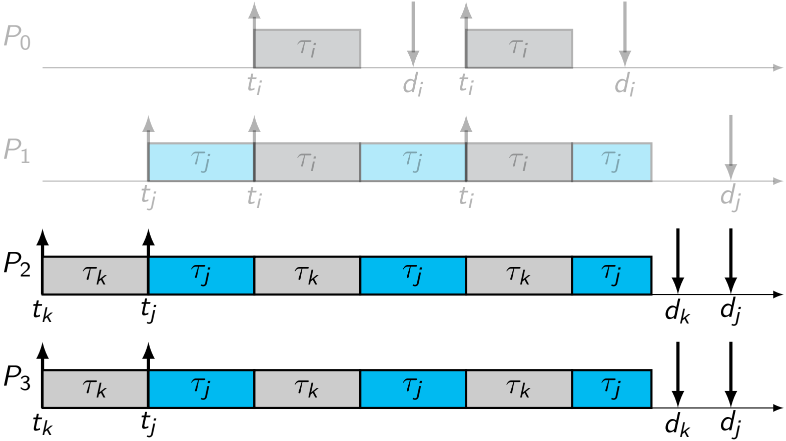
Fixed-Priority Stationary Gang Schedule



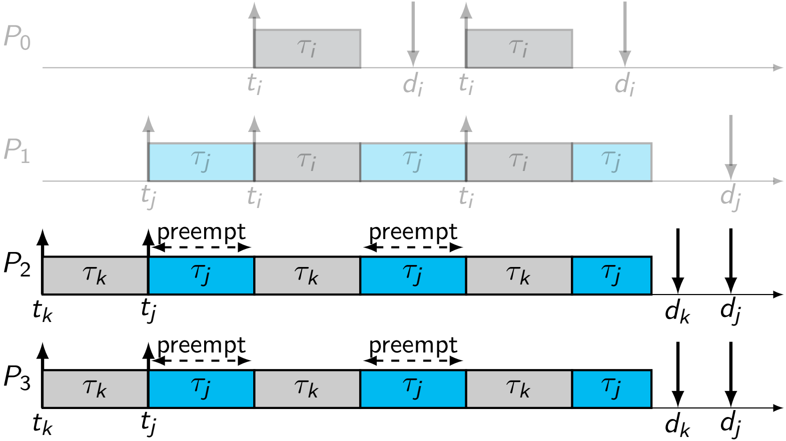
Fixed-Priority Stationary Gang Schedule



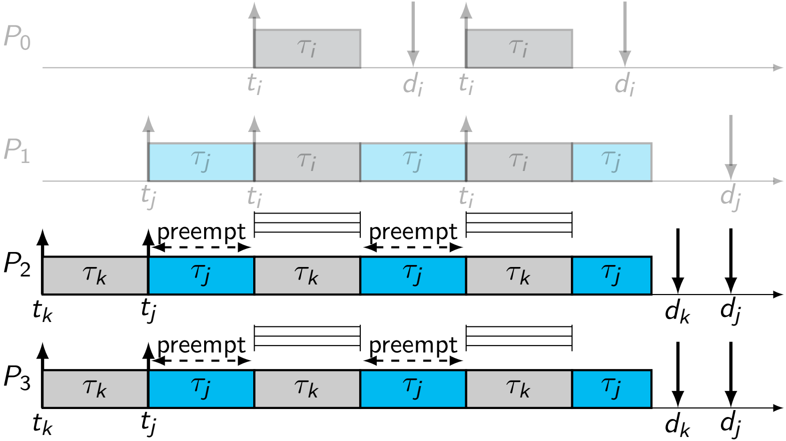
Response-Time Analysis



Response-Time Analysis



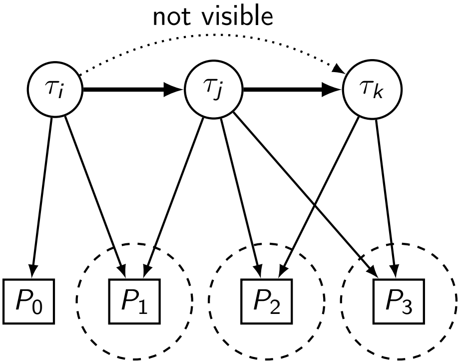
Response-Time Analysis



Exemplary Stationary Gang Assignment

Definition

[Self-Suspension] Higher-priority tasks that **do not interfere** with the job under analysis **may cause self-suspension** like behaviour of interfering tasks



Transformation and Schedulability Analysis

For each task in priority order **do**:

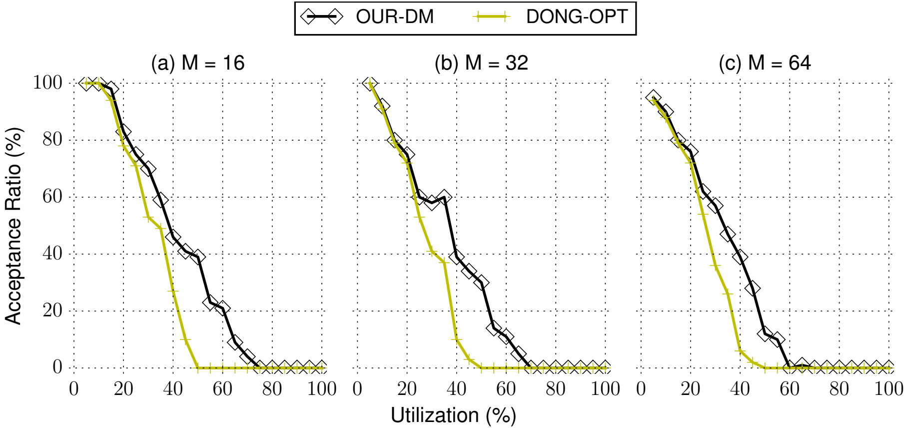
1. Transform higher-priority task set
2. Analyze worst-case response-time based on uniprocessor self-suspension analysis

Definition

[Transformation] Let a sporadic gang task τ_i be transformed to the corresponding self-suspending task $(C_i, D_i, T_i, S_{i,k})$ with the same C_i, D_i, and T_i as for τ_i, where

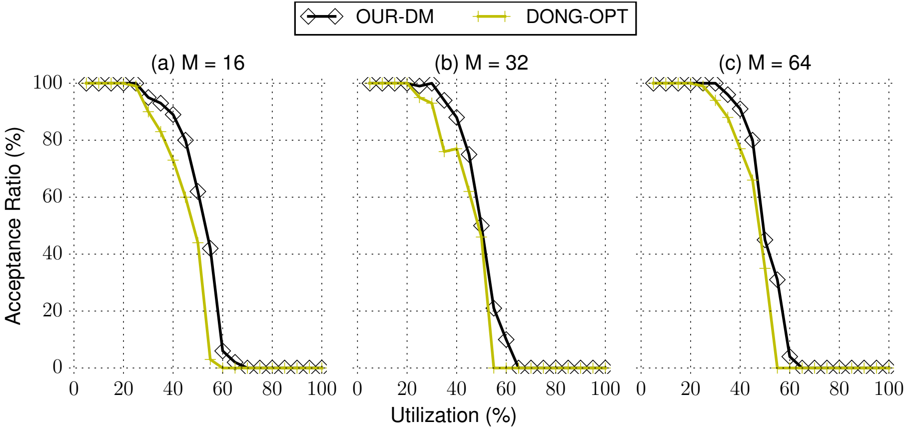
$$\begin{cases} S_{i,k} = \min\left\{R_i - C_i, \sum_{\tau_j \in V_{i,k}} \left(1 + \left\lceil \frac{R_i}{T_j} \right\rceil\right) \cdot C_j\right\} & \text{if it has suspension behaviour} \\ S_{i,k} = 0 & \text{otherwise} \end{cases}$$

Evaluation: Gang Size [1, M/4]



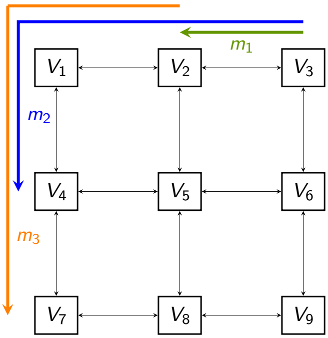
- OUR-DM: Ueter et al. ECRTS 2021
- Dong-OPT: Dong and Liu, RTSS 2017

Evaluation: Gang Size [M/8, M/2]



- OUR-DM: Ueter et al. ECRTS 2021
- Dong-OPT: Dong and Liu, RTSS 2017

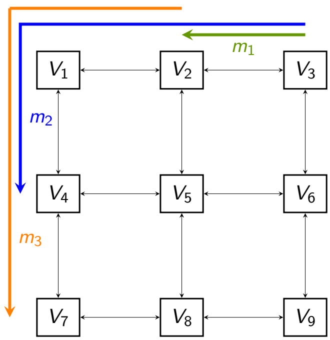
Real-Time Networks on Chip (Revisited)



3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- We can map this to gang scheduling, using each link as a processor

Real-Time Networks on Chip (Revisited)



3 periodic messages
fixed path, preemptive
priority: $m_1 > m_2 > m_3$

- We can map this to gang scheduling, using each link as a processor
  - Assuming that the switching is reserved for one stream completely along the path
  - Ueter et al. RTCSA 2020

Contents

Probabilistic Reasoning

Worst Case Response Time Exceedance Probability (WCRTEP)

The WCRTEP of task τ_k is an upper bound on the probability that the response time of a job of τ_k is greater than t, i.e.,

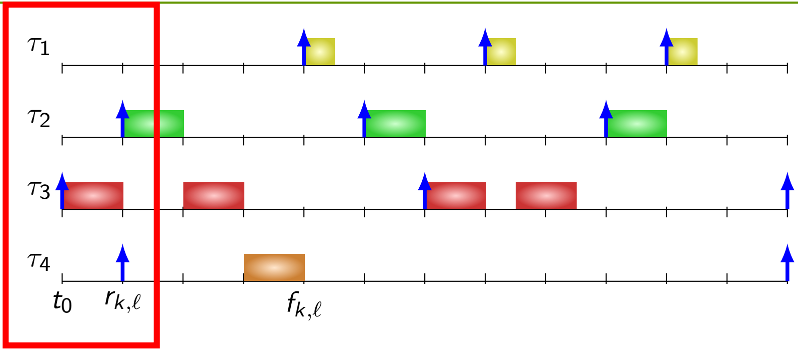
$$\sup_{j \in \mathbb{N}} \{\mathbb{P}(R_{k,j} > t)\}, \tag{1}$$

Worst Case Deadline Failure Probability (WCDFP)

The WCDFP of task τ_k is an upper bound on the probability that a job of τ_k misses its relative deadline D_k:

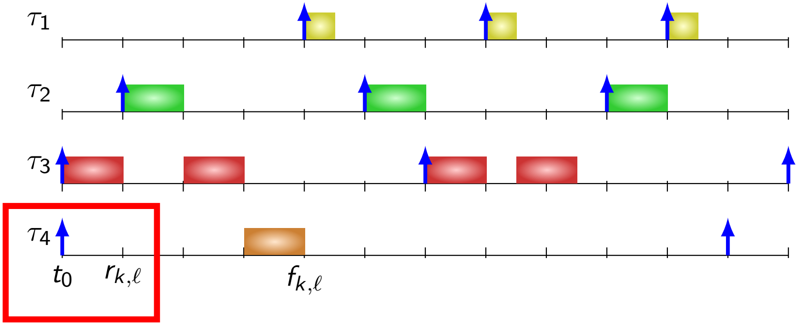
$$\sup_{j \in \mathbb{N}} \{\mathbb{P}(R_{k,j} > D_k)\} \tag{2}$$

Critical Instant



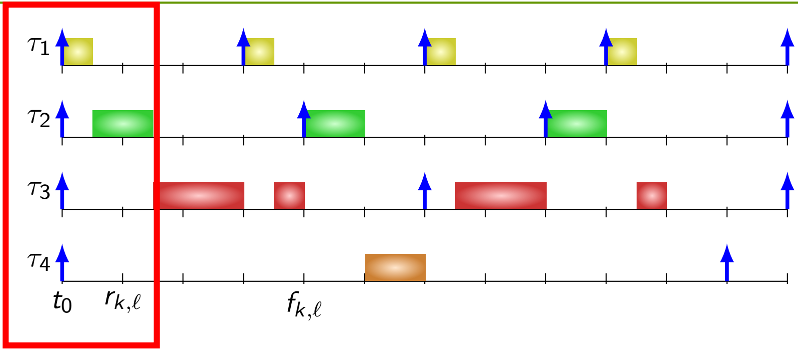
- 1) Interval extension

Critical Instant



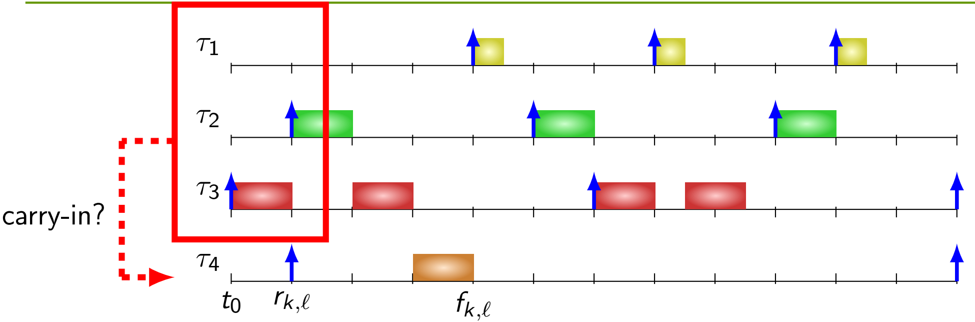
- 1) Interval extension 2) release time modification

Critical Instant



- 1) Interval extension 2) release time modification 3) simultaneous release

Critical Instant for Probabilistic Setup? Refuted

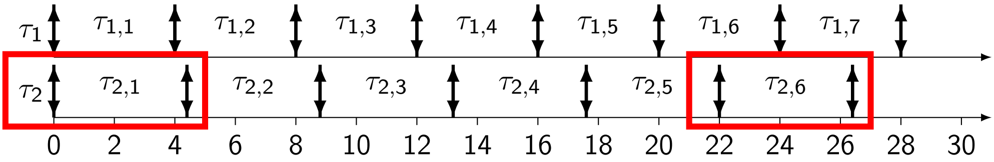


- 1) Interval extension is not deterministic: a probabilistic distribution function!
- Detailed in Chen et al. RTSS 2022

Counterexample: $\mathbb{P}(R_{2,6} > t)$ is higher than $\mathbb{P}(R_{2,1} > t)$

Periodic task τ_1 and τ_2, simultaneously released at time 0, for all $j \in \mathbb{N}$:

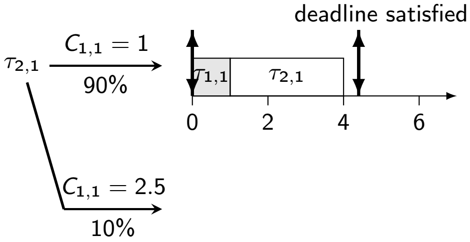
- $T_1 = 4$, $\mathbb{P}(C_{1,j} = 1) = 0.9$, $\mathbb{P}(C_{1,j} = 2.5) = 0.1$
- $T_2 = 4.4$, $\mathbb{P}(C_{1,j} = 3) = 1.0$



Counterexample: $\mathbb{P}(R_{2,6} > t)$ is higher than $\mathbb{P}(R_{2,1} > t)$

Consider t = 4.4, we obtain:

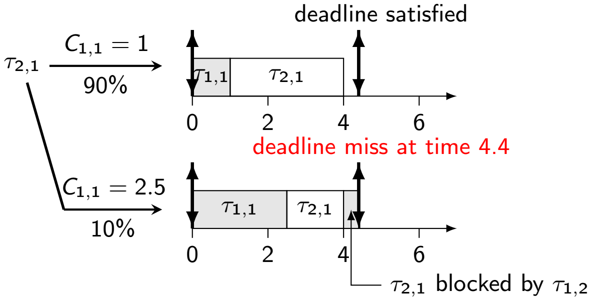
$\mathbb{P}(R_{2,1} > 4.4) = \mathbb{P}(C_{1,1} = 2.5) = \textcolor{red}{0.1}$



Counterexample: $\mathbb{P}(R_{2,6} > t)$ is higher than $\mathbb{P}(R_{2,1} > t)$

Consider t = 4.4, we obtain:

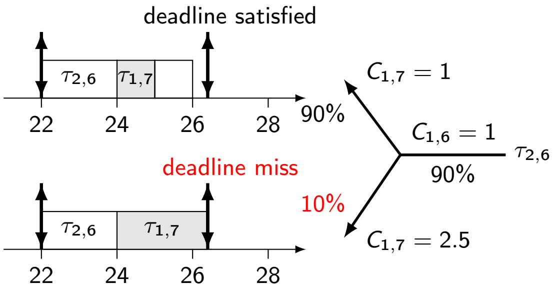
$\mathbb{P}(R_{2,1} > 4.4) = \mathbb{P}(C_{1,1} = 2.5) = \mathbf{0.1}$



Counterexample: $\mathbb{P}(R_{2,6} > t)$ is higher than $\mathbb{P}(R_{2,1} > t)$

Consider t = 4.4, we obtain:

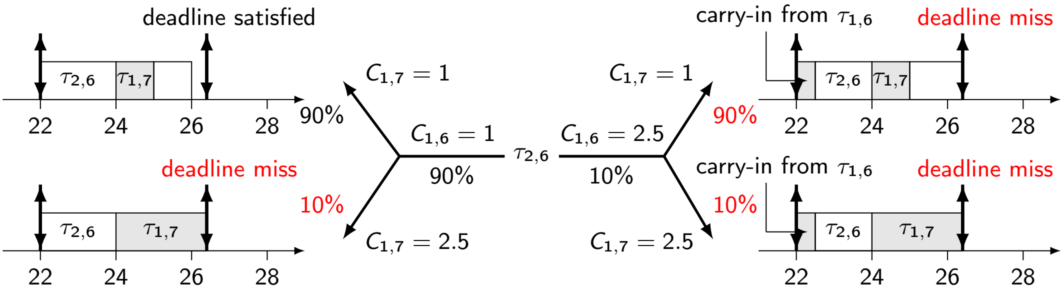
$\mathbb{P}(R_{2,6} > 4.4) = \mathbb{P}(C_{1,6} = 1) \cdot \mathbb{P}(C_{1,7} = 2.5)$



Counterexample: $\mathbb{P}(R_{2,6} > t)$ is higher than $\mathbb{P}(R_{2,1} > t)$

Consider t = 4.4, we obtain:

$\mathbb{P}(R_{2,6} > 4.4) = \mathbb{P}(C_{1,6} = 1) \cdot \mathbb{P}(C_{1,7} = 2.5) + \mathbb{P}(C_{1,6} = 2.5) = 0.9 \cdot 0.1 + 0.1 = $ **0.19**



Higher priority tasks like τ_1 may still provide carry-in!

Contents

Conclusion

- To effectively analyze real-time properties in multicore systems, we need
  - *formal properties* to modularly compose real-time embedded systems
  - *methodologies* for generating/verifying properties and modular compositions
  - *predictable interplay* of complex real-time embedded systems

Conclusion

- To effectively analyze real-time properties in multicore systems, we need
  - *formal properties* to modularly compose real-time embedded systems
  - *methodologies* for generating/verifying properties and modular compositions
  - *predictable interplay* of complex real-time embedded systems
- Classical computation-centric view is limited and may be prone to error
- The focus should be shifted to
  - communication,
  - synchronization, and
  - parallelization.

Conclusion

- To effectively analyze real-time properties in multicore systems, we need
  - *formal properties* to modularly compose real-time embedded systems
  - *methodologies* for generating/verifying properties and modular compositions
  - *predictable interplay* of complex real-time embedded systems
- Classical computation-centric view is limited and may be prone to error
- The focus should be shifted to
  - communication,
  - synchronization, and
  - parallelization.

It is the worst of time and also the best of time.