Design and Analysis of Time-Critical Systems

Response-time Analysis with a Focus on Shared Resources

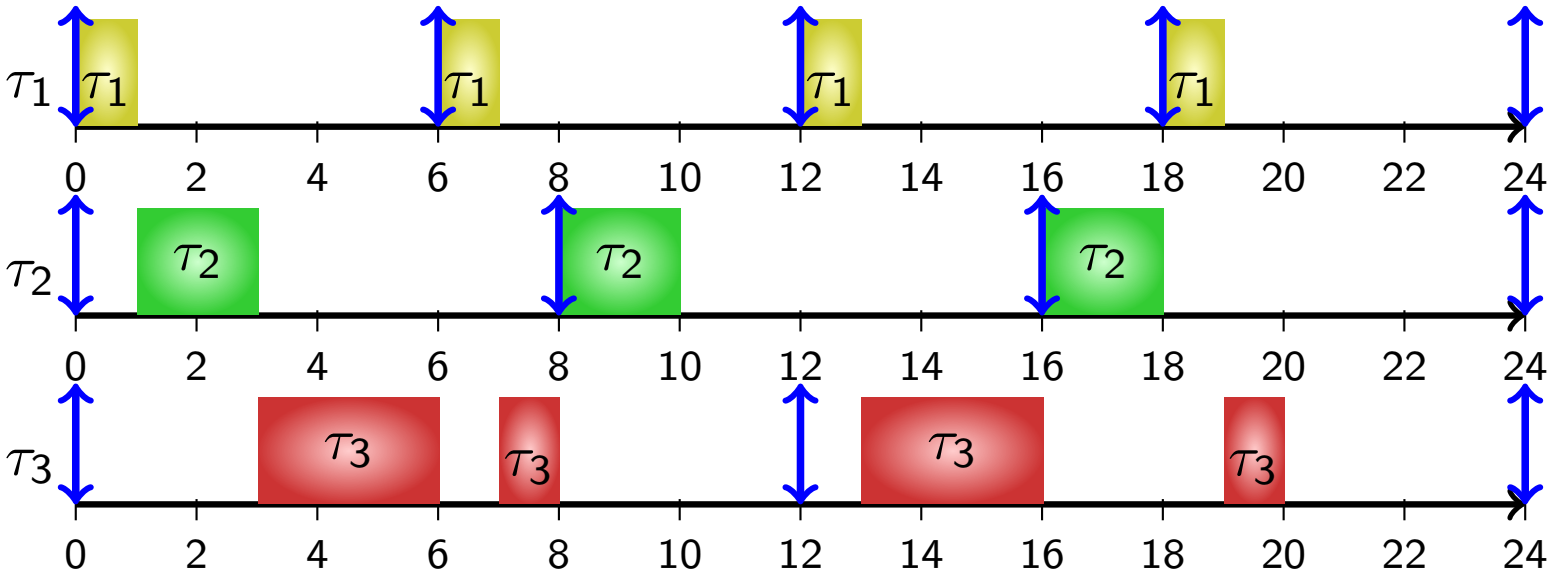
Jan Reineke @ SAARLAND UNIVERSITY
COMPUTER SCIENCE

ACACES Summer School 2017

Fiuggi, Italy

Fixed-Priority Preemptive Scheduling

1. Prior to execution: fix static priorities of tasks
2. During execution:
   always schedule highest-priority ready task

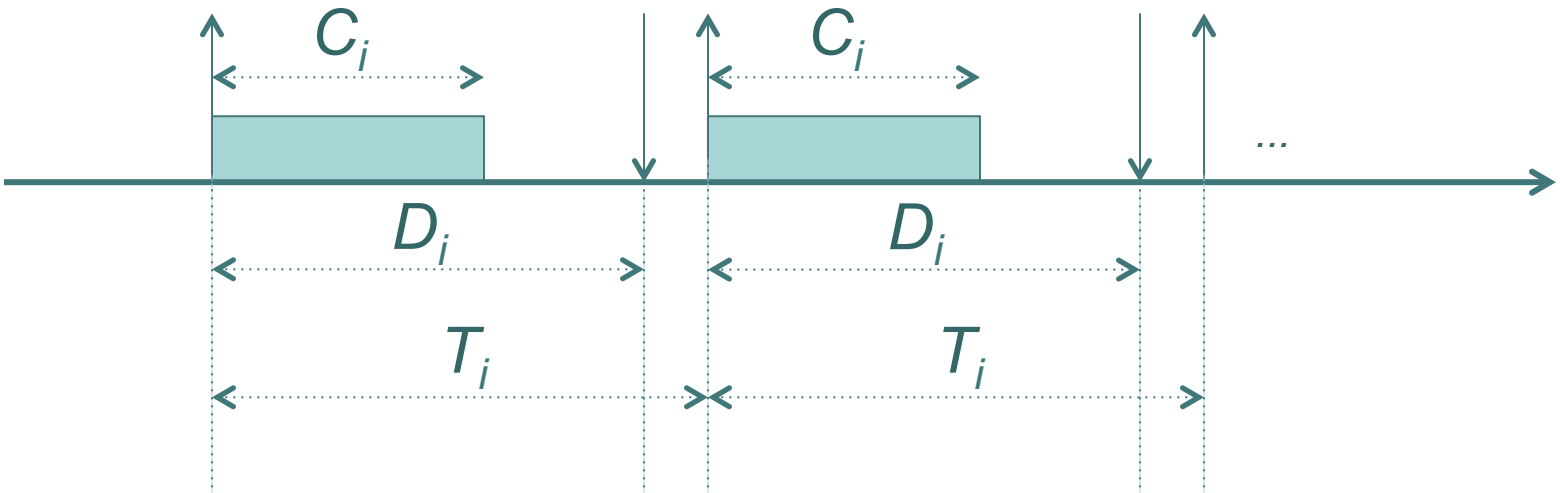


Convention: smaller index ⇔ higher priority

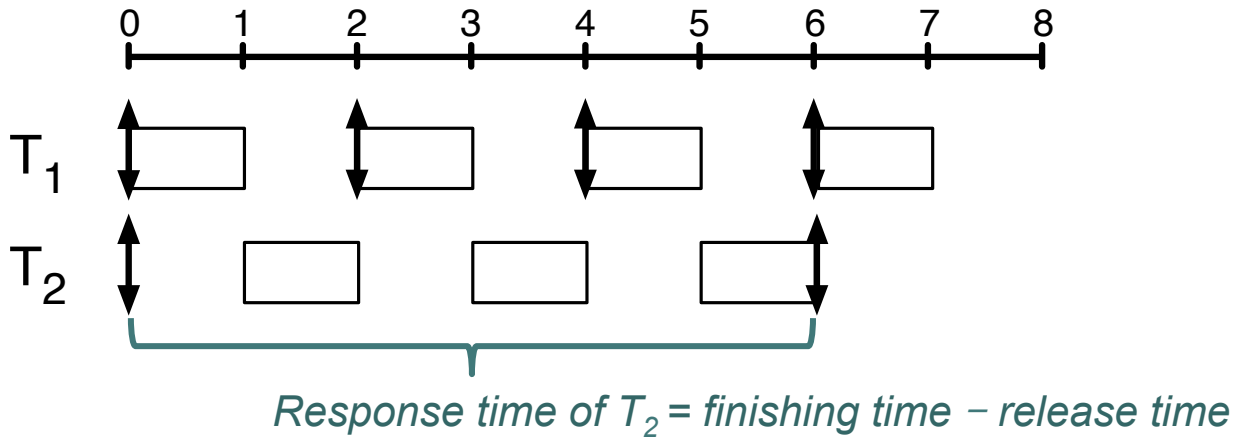
A Standard Recurrent Task Model

Tasks are characterized by 3 parameters:

1. C_i, (a bound on) its worst-case execution time,
2. T_i, its period,
3. D_i, its relative deadline, with $D_i \leq T_i$.



Timing Analysis for Fixed-Priority Preemptive Scheduling



Traditional division of work:

1. Worst-case Execution Time (WCET) Analysis
   determines bounds on the execution time of a task when run in isolation
2. Response-time Analysis
   determines bounds on tasks' response times given WCET bounds, accounting for interference due to scheduling decisions

Response-Time Analysis for Fixed-Priority Scheduling in a Nutshell

The time-demand function $W_i(t)$ of task τ_i is defined as follows:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

- C_i: *WCET of task τ_i*
- $\sum_{j=1}^{i-1}$: *Sum over interference from higher-priority tasks*
- $\left\lceil \frac{t}{T_j} \right\rceil$: *How often can task τ_j be released in an interval of length t?*
- C_j: *WCET of task τ_j*

Theorem *(Fixed-priority schedulability):*
A system T of periodic tasks is schedulable on one processor by fixed-priority preemptive scheduling if it holds that:

$$\forall \tau_i \in T : \exists t, 0 < t \leq D_i : W_i(t) \leq t$$

Response-Time Analysis for Fixed-Priority Scheduling in a Nutshell

Want to determine whether there is a t, s.t. $W_i(t) \leq t \leq D_i$.

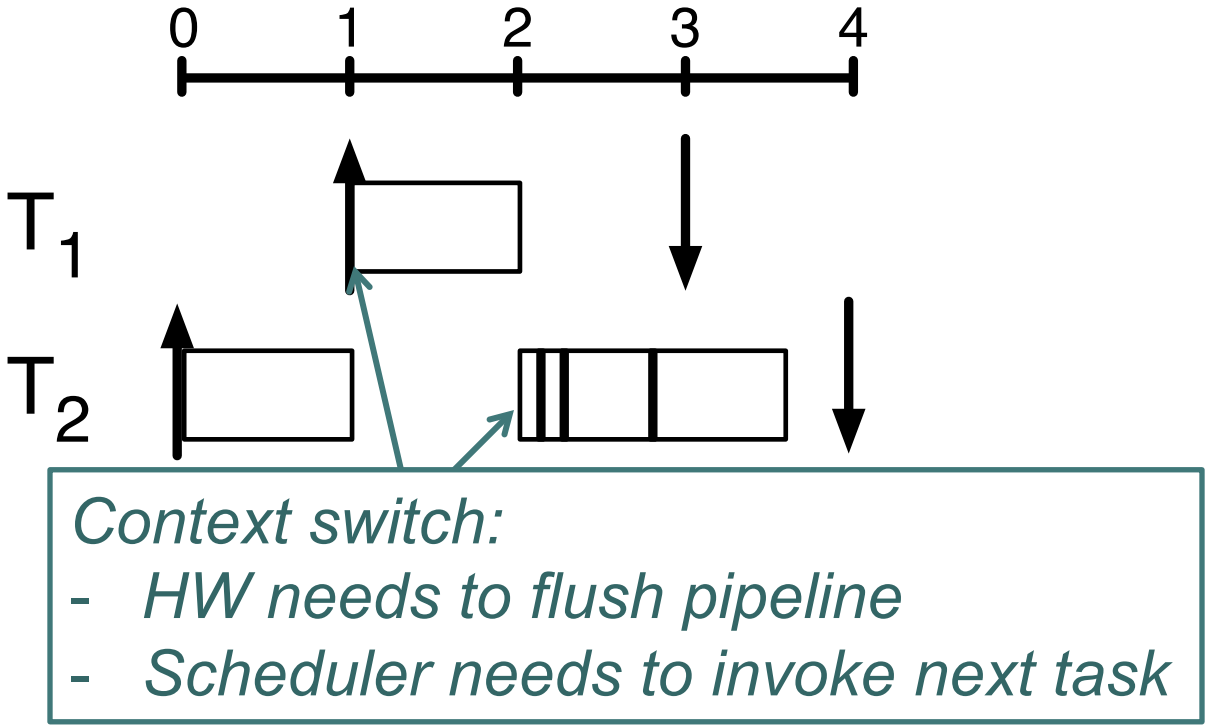
Compute smallest such t (if there is any) by fixed-point iteration:

$$
\begin{aligned}
&n := 0 \\
&R_i^0 := C_i \\
&\textbf{do} \\
&\quad n := n + 1 \\
&\quad R_i^n := C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{n-1}}{T_j} \right\rceil C_j \\
&\textbf{while } R_i^n < D_i \text{ and } R_i^{n-1} < R_i^n
\end{aligned}
$$

A set of tasks is schedulable if $R_i^n < D_i$ for every task i.

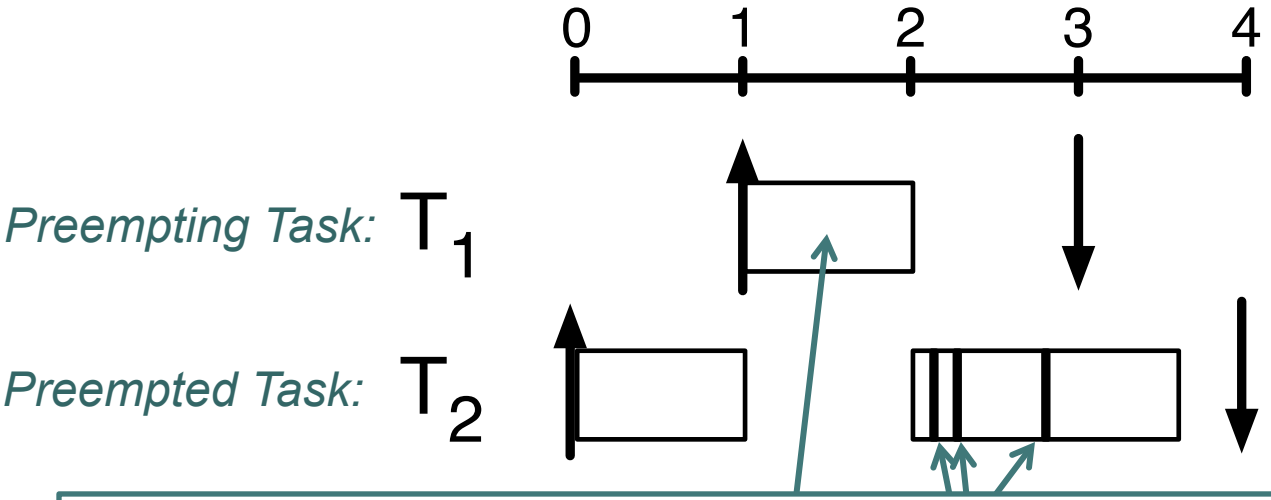
Unrealistic Assumptions of the Response-time Analysis

1. **Context switches are free**
2. Tasks execution times are not affected by preemptions



Unrealistic Assumptions of the Response-time Analysis

1. Context switches are free
2. **Tasks execution times are not affected by preemptions**



Overhead referred to as Cache-Related Preemption Delay (CRPD)

1. *Preempting task T_1 evicts some of T_2's cache contents*
2. *Preempted task T_2 suffers additional cache misses*

How to Account for Preemption Overheads

Alternatives:

1. Incorporate all possible overheads in WCET bound

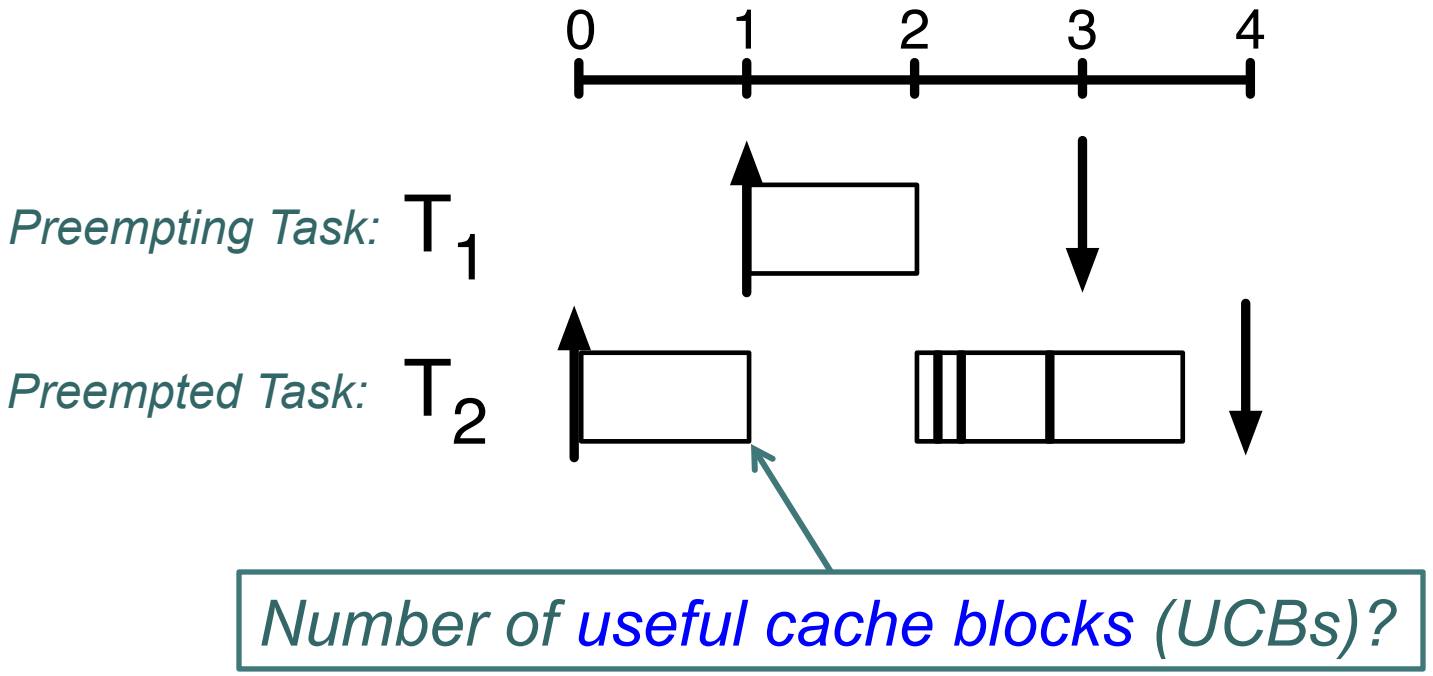
   - Would have to assume cache misses on every memory access → extremely pessimistic

2. Extend interface between WCET and Response-time Analysis

   - Capture memory-access behavior of tasks to bound CRPD during reponse-time analysis

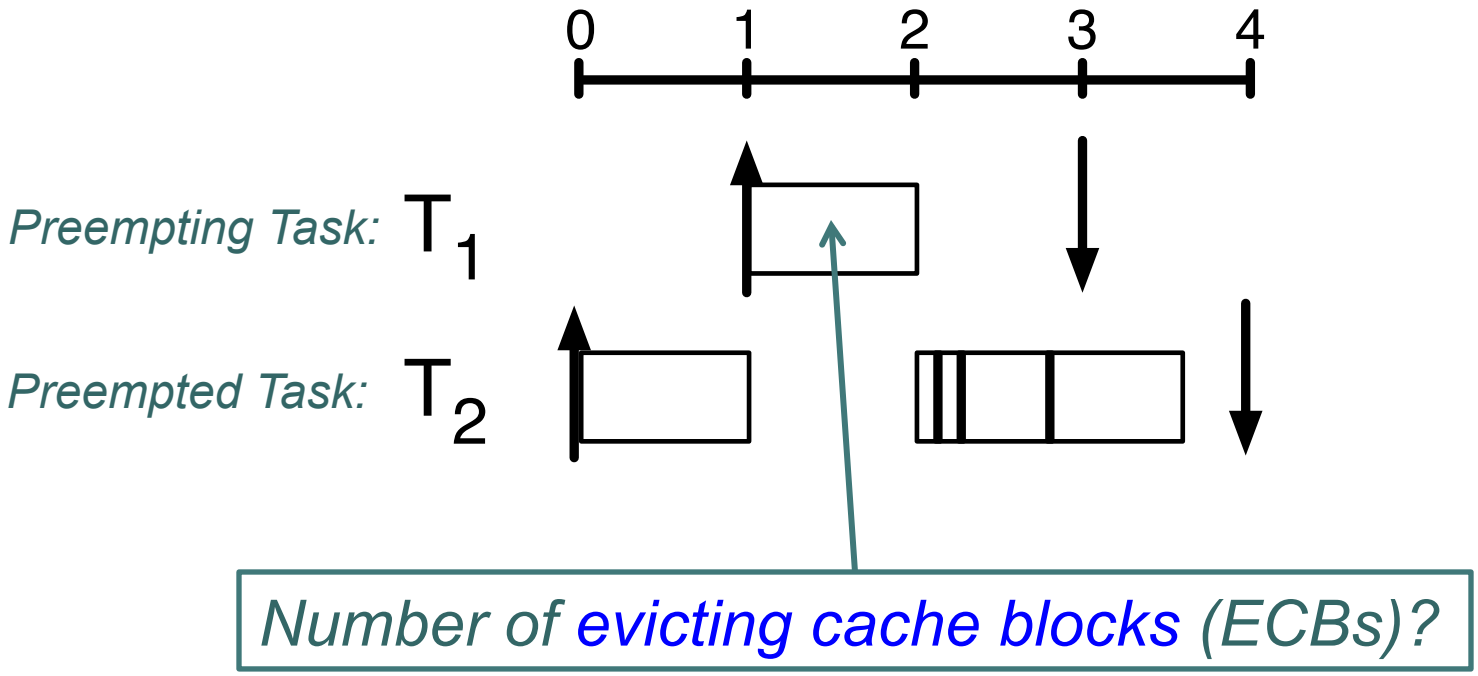
How to Characterize Memory-access Behavior of Tasks?

Approach 1: Consider preempted task



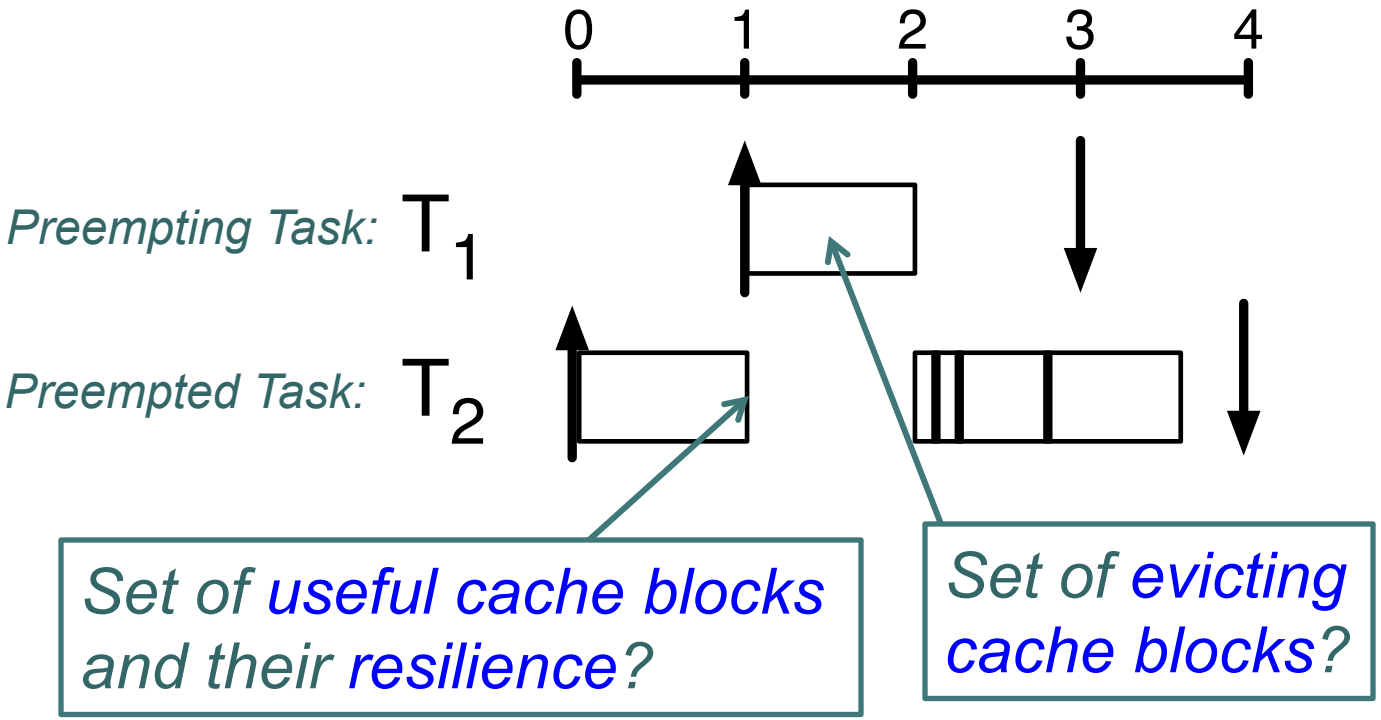
How to Characterize Memory-access Behavior of Tasks?

Approach 2: Consider preempting task



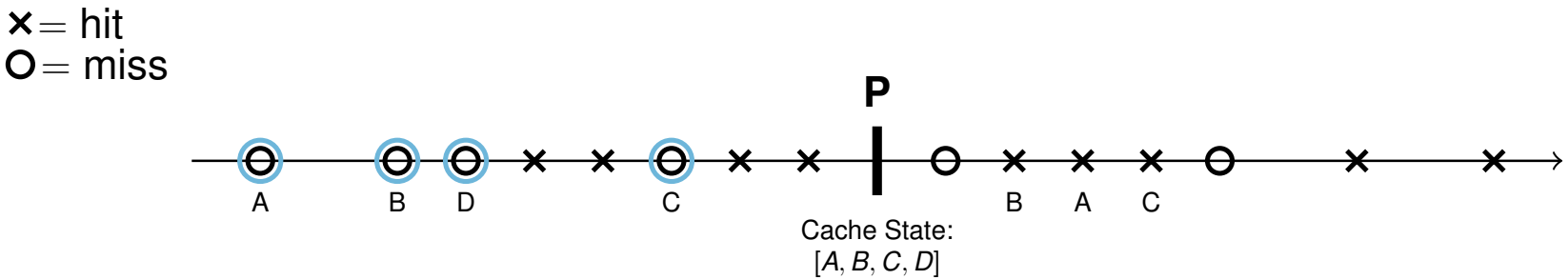
How to Characterize Memory-access Behavior of Tasks?

Approach 3: Combine both using Resilience



Approach 1: Analysis of the Preempted Task

Definition *(Useful cache block, Lee et al. 1996):*
A memory block *m* at program point *P* is called a useful cache block, if
1. ***m* may be cached at *P***, *and*
2. *m* may be reused at program point *Q* that may be reached from *P* with no eviction of *m* on this path.

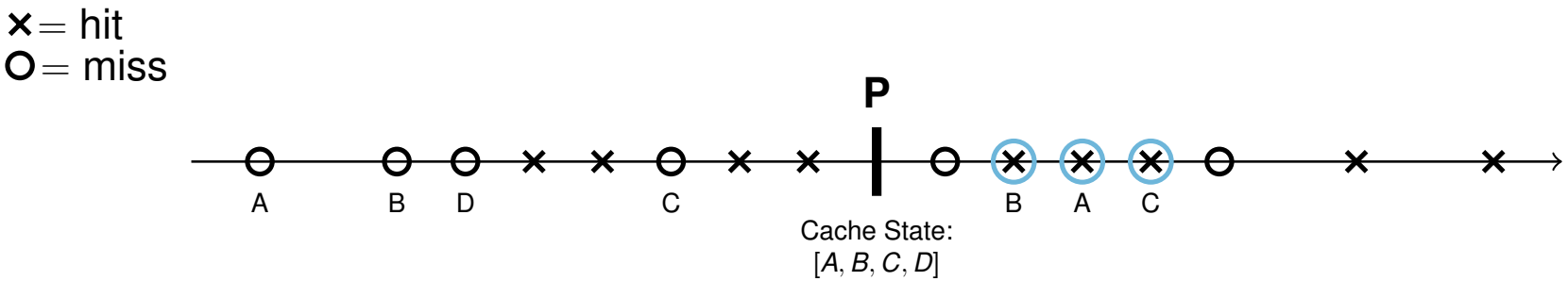


$$\text{UCB} \subseteq \{A, B, C, D\}$$

Approach 1: Analysis of the Preempted Task

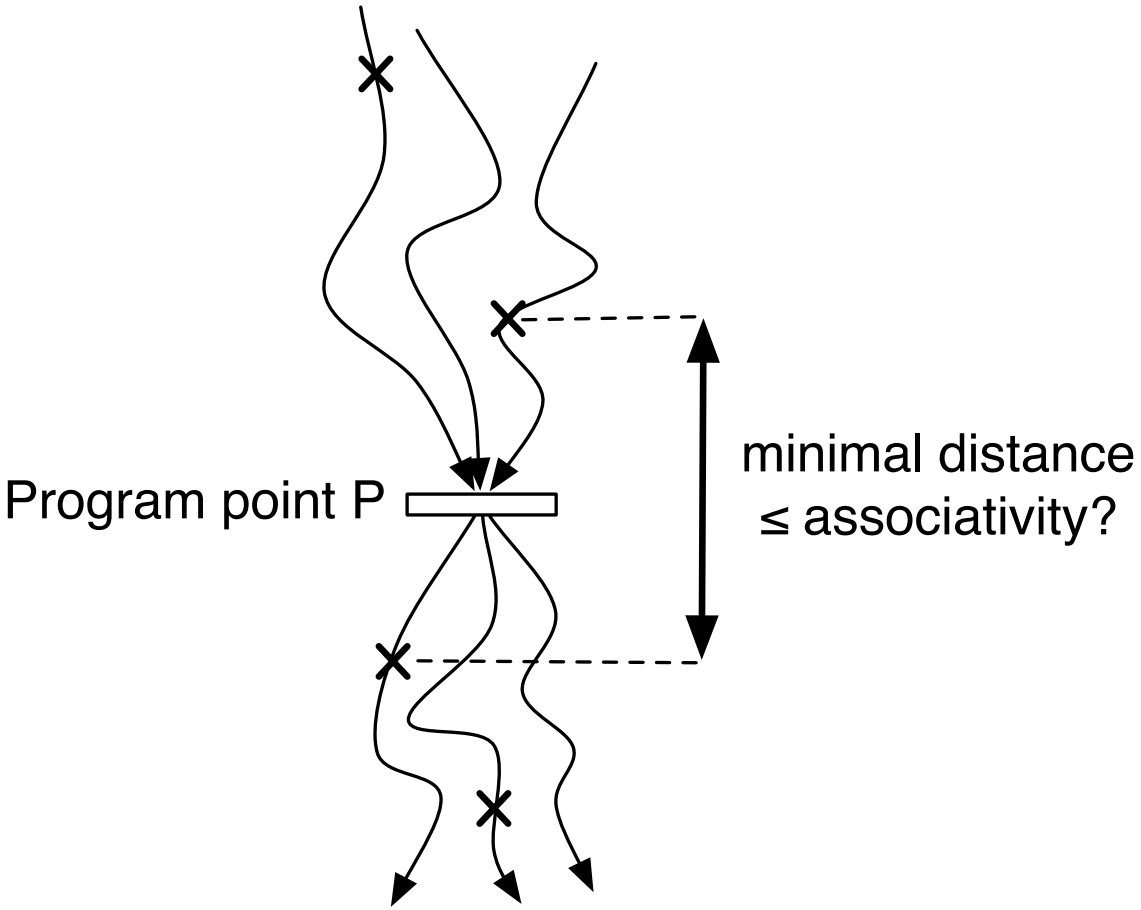
Definition *(Useful cache block, Lee et al. 1996):*
A memory block *m* at program point *P* is called a useful cache block, if

1. *m* may be cached at *P, and*
2. ***m* may be reused at program point *Q* that may be reached from *P* with no eviction of *m* on this path.**



$$\text{UCB} = \{A, B, C\}$$

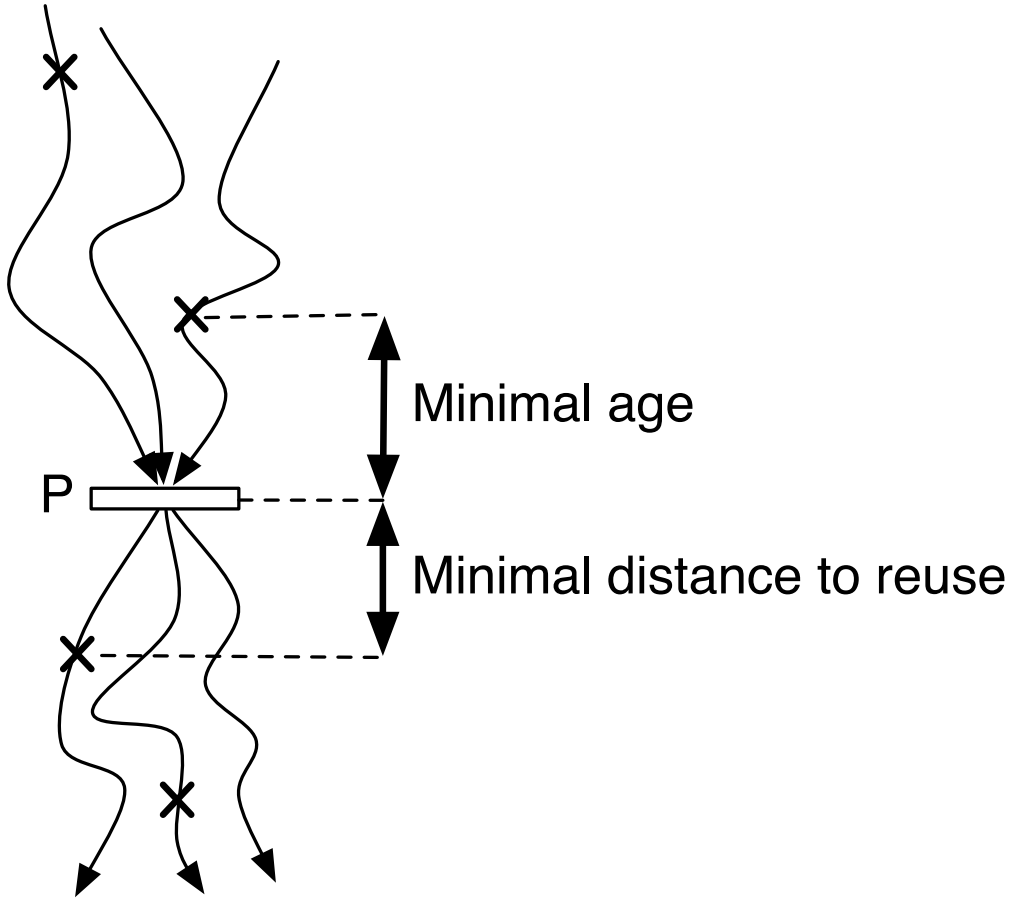
Useful Cache Block (UCB) Analysis



Useful Cache Block (UCB) Analysis

What may be cached?

Forward May-Analysis!



What may be reused?

Backward May-Analysis!

Bounding the CRPD using UCBs for Fully-associative Caches

- CRPD bound at program point P:

$$CRPD_{UCB}^{LRU}(P) = BRT \cdot \min(\left|UCB(P)\right|, associativity)$$

where BRT is the “block reload time”.

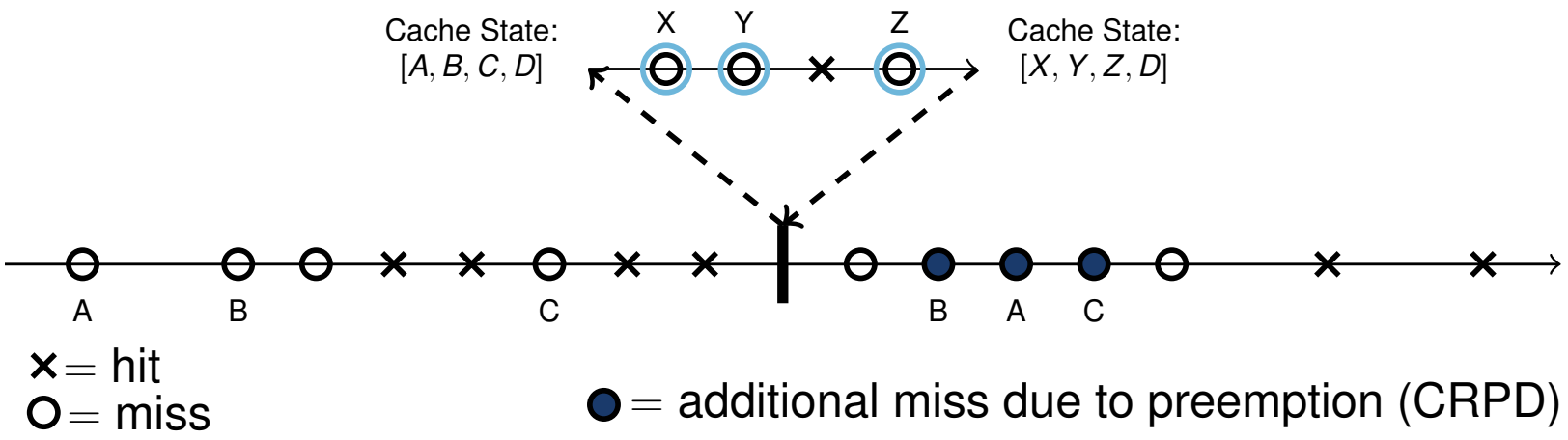
- CRPD bound independent of program point:

$$CRPD_{UCB}^{LRU} = \max_{P} CRPD_{UCB}^{LRU}(P)$$

For set-associative caches:
sum up bounds for all cache sets

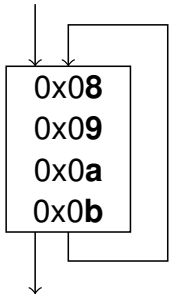
Approach 2: Analysis of the Preempting Task

Definition *(Evicting cache block, Tomiyama and Dutt 2000):*
A memory block *of the preempting task is called an evicting cache block, if it may be accessed during the execution of the preempting task.*



$$CRPD_{ECB}^{LRU} \stackrel{?}{=} BRT \cdot \min(|ECB|, k)$$ **No!**

CRPD Computation for LRU using ECBs
Pitfall



ECBs $= \{\mathbf{e}\}$

$[b, a, 9, 8] \xrightarrow{8} [8, b, a, 9] \xrightarrow{9} [9, 8, b, a] \xrightarrow{a} [a, 9, 8, b] \xrightarrow{b} [b, a, 9, 8]$ 0 misses

$[\mathbf{e}, b, a, 9] \xrightarrow{8^*} [8, \mathbf{e}, b, a] \xrightarrow{9^*} [9, 8, \mathbf{e}, b] \xrightarrow{a^*} [a, 9, 8, \mathbf{e}] \xrightarrow{b^*} [b, a, 9, 8]$ 4 misses

|UCB| = 4

|ECB| = 1

associativity = 4

number of additional misses = 4

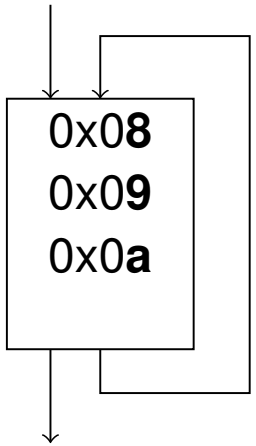
CRPD Computation for LRU using ECBs
Sound but Imprecise

- ECB analysis only to determine whether the set if used at all by the preempting task or not:

$$CRPD_{ECB}^{LRU} = \begin{cases} 0 & ECB = \emptyset \\ BRT \cdot k & otherwise \end{cases}$$

Cannot do better without knowledge of preempted task.

Approach 3: Analysis of the Preempted and the Preempting Task



ECBs $= \{\mathbf{e}\}$

Without preemption:

$$[a, 9, 8, 7] \xrightarrow{8} [8, a, 9, 7] \xrightarrow{9} [9, 8, a, 7] \xrightarrow{a} [a, 9, 8, 7]$$

With preemption:

$$[\mathbf{e}, a, 9, 8] \xrightarrow{8} [8, \mathbf{e}, b, a] \xrightarrow{9} [9, 8, \mathbf{e}, b] \xrightarrow{a} [a, 9, 8, \mathbf{e}]$$

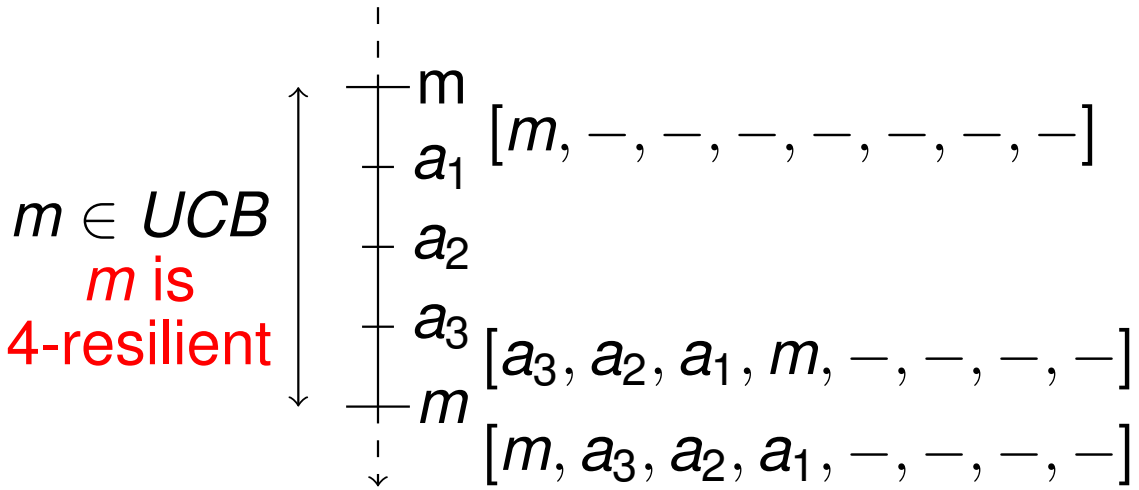
Some of the UCBs are guaranteed to *remain useful* under preemption!

Why?

- Minimal number of ECBs to evict a UCB is 2, but |ECB| = 1.
- A single ECB is not sufficient to evict any of the UCBs.

Combining UCBs and ECBs: *Notion of Resilience*

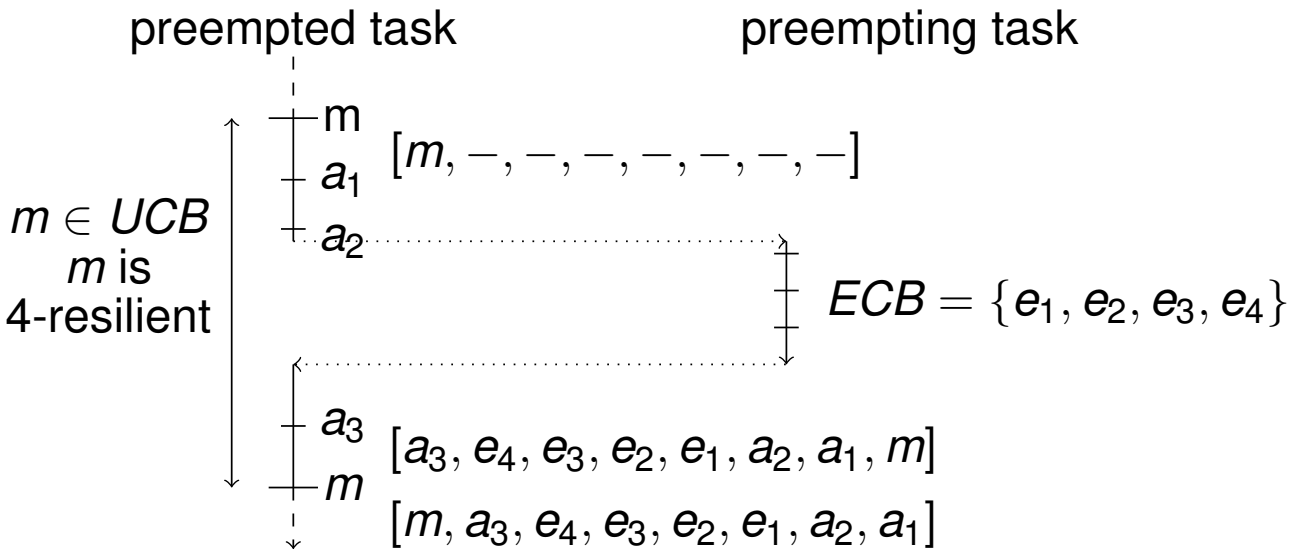
For a given block m determine the maximum number of ECBs that it will "survive":



Definition of Resilience

Definition *(Resilience):*
A memory block m is called l-resilient at program point *P*, if all possible next accesses to m

- that would be hits without preemption,
- would still be hits in case of a preemption at P with l accesses.

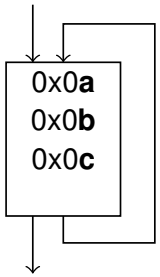


Bounding the CRPD using Resilience

Can exclude all useful cache blocks from the CRPD bound that are guaranteed to survive the preemption:

$$CRPD \leq BRT \times \left| \overbrace{\underbrace{UCB}_{\text{useful}} \setminus \underbrace{\{m \mid m \text{ is } |ECB|\text{-resilient}\}}_{\text{remain useful}}}^{\text{blocks contributing to CRPD}} \right|$$

Bounding the CRPD using Resilience Example



ECBs $= \{\mathbf{e}\}$

$[c, b, a, x] \xrightarrow{a} [a, c, b, x] \xrightarrow{b} [b, a, c, x] \xrightarrow{c} [c, b, a, x]$ no misses

$[\mathbf{e}, c, b, a] \xrightarrow{a} [a, \mathbf{e}, c, b] \xrightarrow{b} [b, a, \mathbf{e}, c] \xrightarrow{c} [c, b, a, \mathbf{e}]$ no misses

- |ECB| = 1
- a, b, and c are 1-resilient
- CRPD = BRT*|UCB \ {m | m is |ECB|-resilient}|
  = BRT*|{a,b,c} \ {a,b,c}|
  = 0

How to Incorporate CRPD Bounds in Response-time Analysis?

- Have so far seen how to bound CRPD for a single preemption.
- *Reminder*:

The time-demand function $W_i(t)$ of task τ_i is defined as follows:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

- C_i: *WCET of task τ_i*
- $\sum_{j=1}^{i-1}$: *Sum over interference from higher-priority tasks*
- $\left\lceil \frac{t}{T_j} \right\rceil$: *How often can task τ_j be released in an interval of length t?*
- C_j: *WCET of task τ_j*

How to Incorporate CRPD Bounds in Response-time Analysis?

Introduce additional term $\gamma_{i,j}$ in the time-demand function:

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil (C_j + \gamma_{i,j}).$$

Used to bound the CRPD

Fixed-point iteration can be appropriately adapted.

Questions:

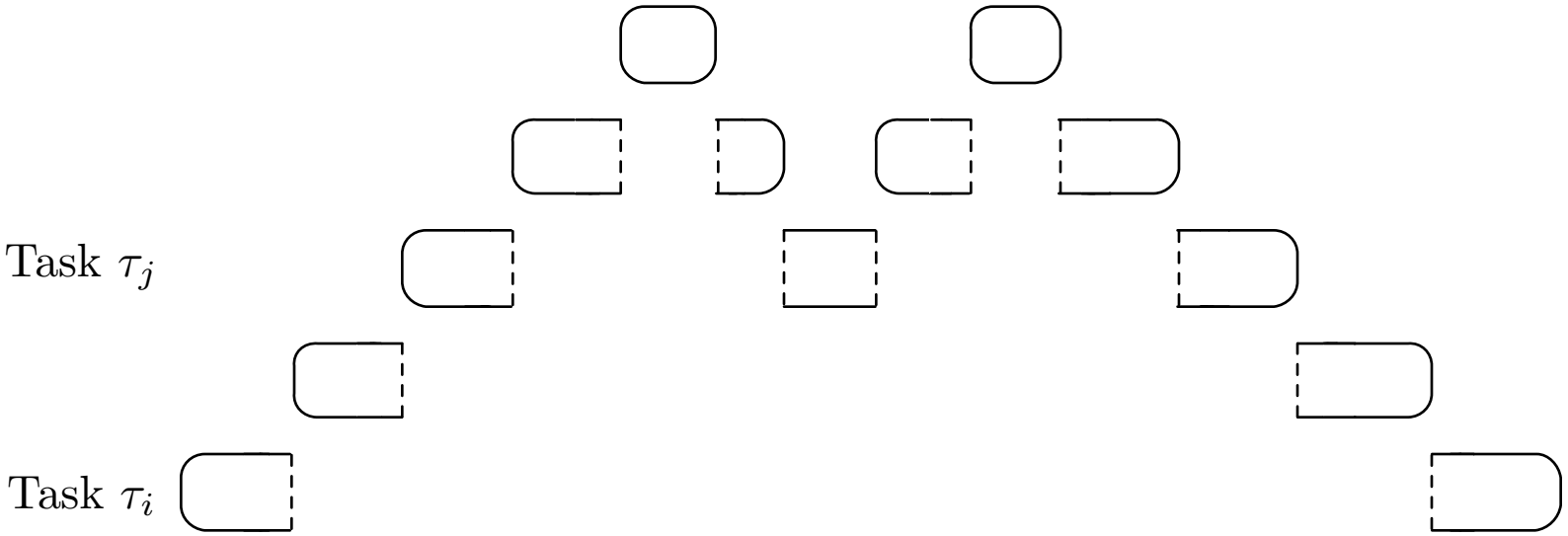
1. *What is the meaning of $\gamma_{i,j}$?*
2. *Which value should it take?*

Interpretation of $\gamma_{i,j}$

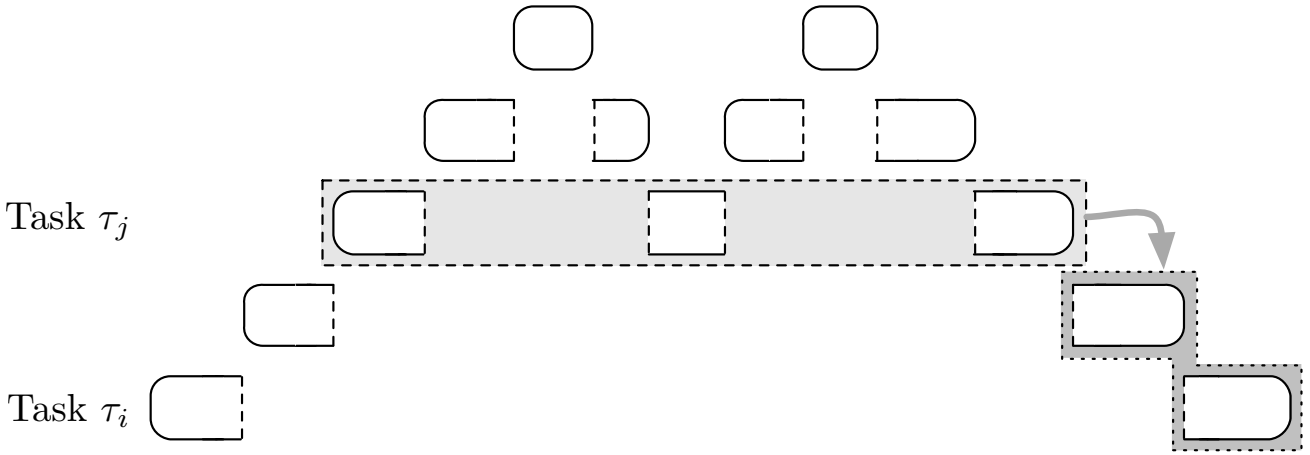
Without nested preemptions:

$\gamma_{i,j}$ = CRPD due to one preemption of task i by task j

But what if there are nested preemptions?

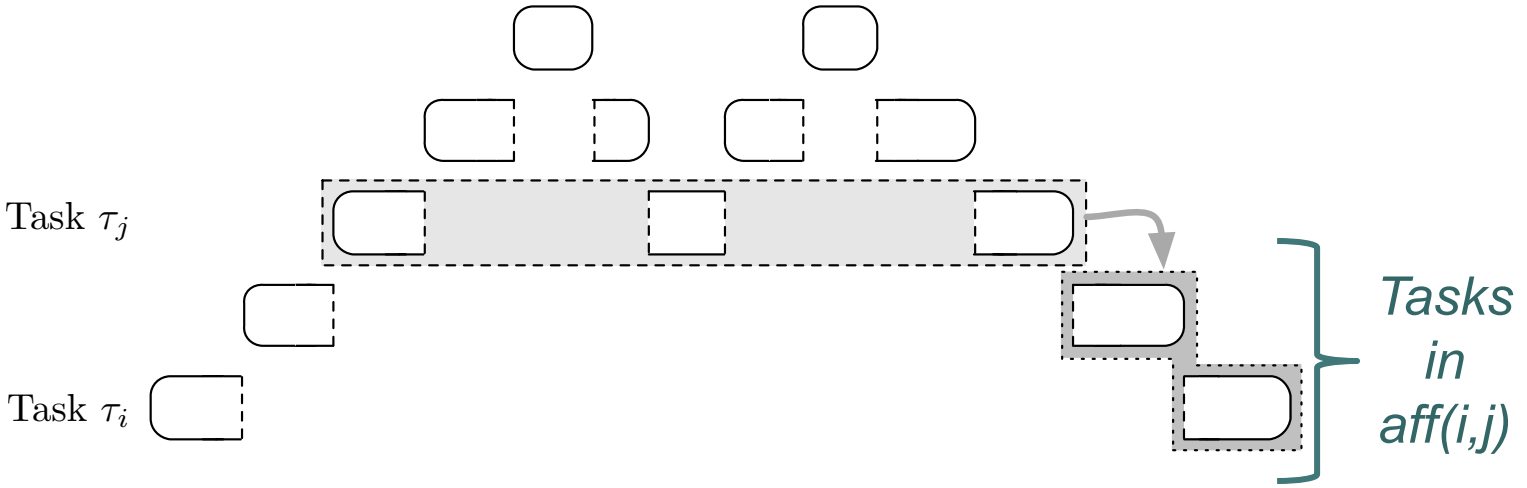


First Interpretation of $\gamma_{i,j}$: "Effect of Preempting Task"



Under this interpretation, $\gamma_{i,j}$ needs to bound the effect of *j*'s execution on all tasks of lower priority up to task *i*.

First Interpretation of $\gamma_{i,j}$: "Effect of Preempting Task"

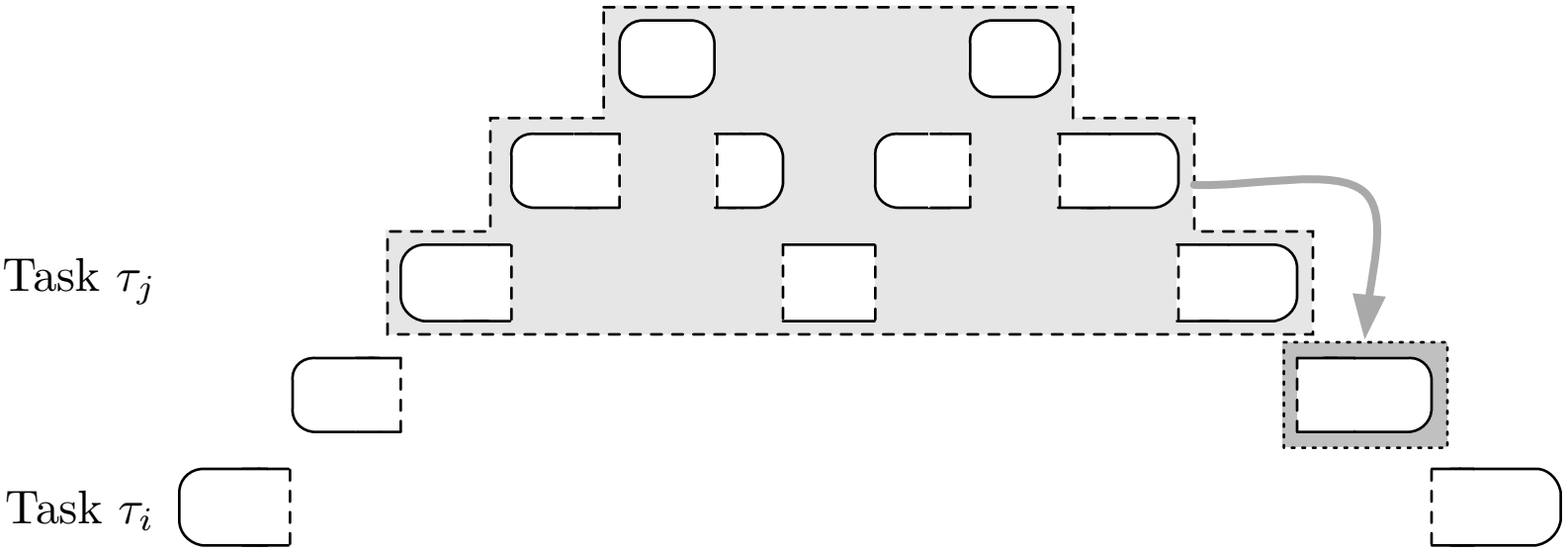


For *direct-mapped caches* there are three such approaches:

1. "ECB-Only": $\gamma_{i,j} = BRT \cdot \left| ECB_j \right|$
2. "UCB-Union": $\gamma_{i,j} = BRT \cdot \left| \bigcup_{k \in aff(i,j)} UCB_k \right|$
3. Combination of the above:

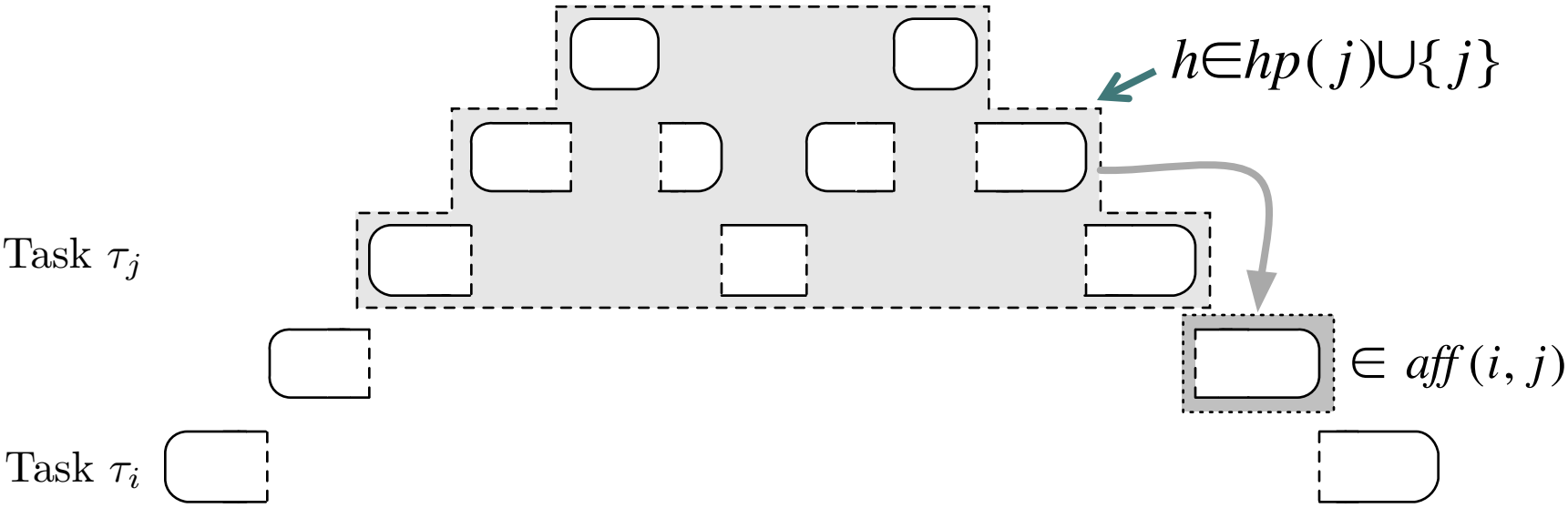
$$\gamma_{i,j} = BRT \cdot \left| \bigcup_{k \in aff(i,j)} UCB_k \bigcap ECB_j \right|$$

Second Interpretation of $\gamma_{i,j}$: "Effect on Immediately Preempted Task"



Under this interpretation, $\gamma_{i,j}$ needs to bound the effect of *j*'s execution and that of the execution of tasks preempting *j* itself on the task that *j* immediately preempted.

Second Interpretation of $\gamma_{i,j}$: "Effect on Immediately Preempted Task"



For *direct-mapped caches* there are three such approaches:

1. "UCB-Only": $\gamma_{i,j} = BRT \cdot \max_{k \in aff(i,j)} |UCB_k|$
2. "ECB-Union": $\gamma_{i,j} = BRT \cdot \left| \bigcup_{h \in hp(j) \cup \{j\}} ECB_h \right|$
3. Combination of the above:

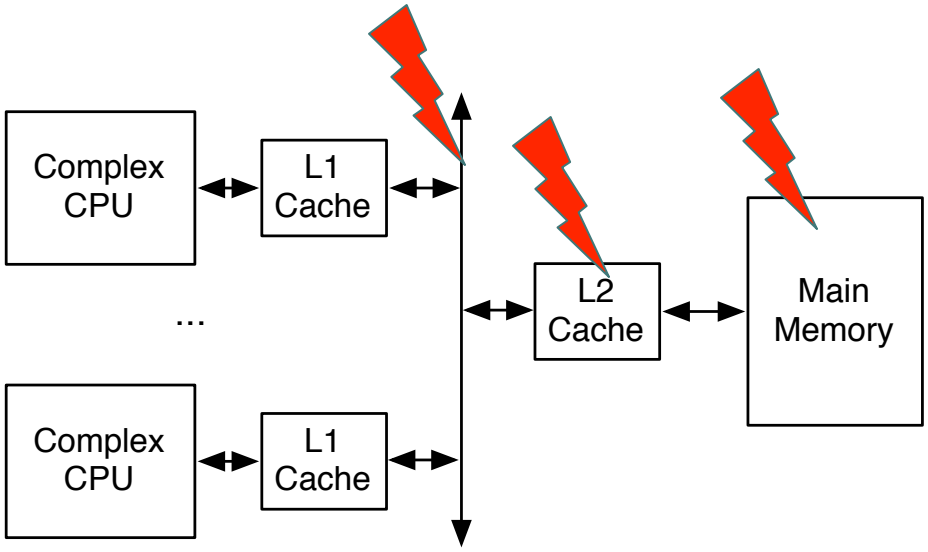
$$\gamma_{i,j} = BRT \cdot \max_{k \in aff(i,j)} \left| UCB_k \cap \bigcup_{h \in hp(j) \cup \{j\}} ECB_h \right|$$

Summary – Preemptive Scheduling

- Preemptive scheduling is desirable, but not free due to shared resources, in particular caches:
  - → Need to bound the CRPD
  - → Requires extension of interface between WCET analysis and response-time analysis

Multi-Core Timing Analysis

Execution time depends strongly on execution context due to interference on shared resources



“Standard Approach” for Timing Analysis

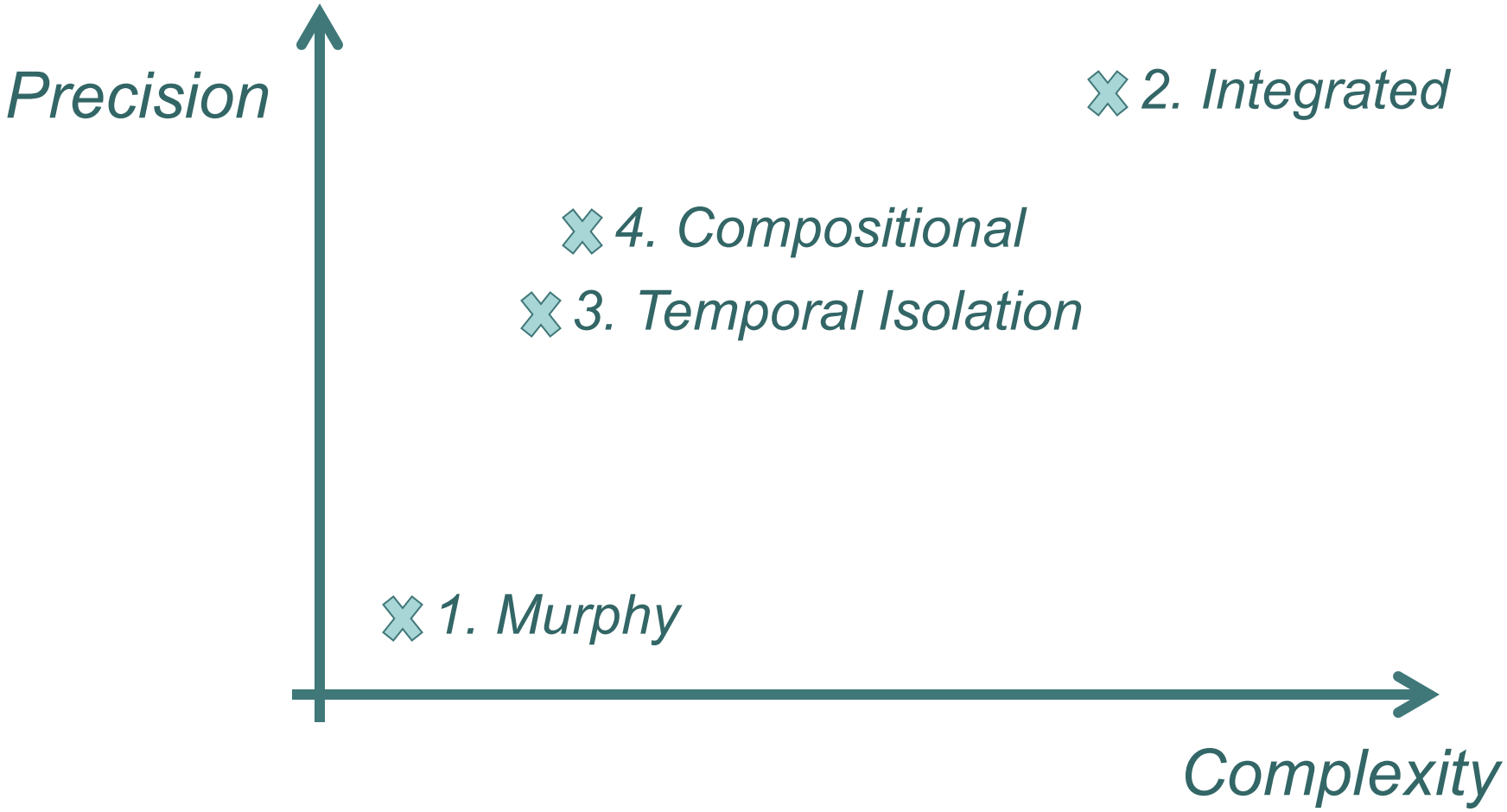
Two-phase approach:

1. Determine WCET (worst-case execution time) bounds for each task on platform
2. Perform response-time analysis

Simple interface between WCET analysis and response-time analysis: WCET bounds

Still adequate in case of multi cores?

Four Approaches to Timing Analysis for Multi- and Many-Cores



1. Murphy Approach

Maintain standard two-phase approach:

1. Determine context-independent WCET bound
2. Perform response-time analysis

Radojkovic et al. (ACM TACO, 2012) on Intel Atom and Intel Core 2 Quad:

up to 14x slow-down due to interference on shared L2 cache and memory controller

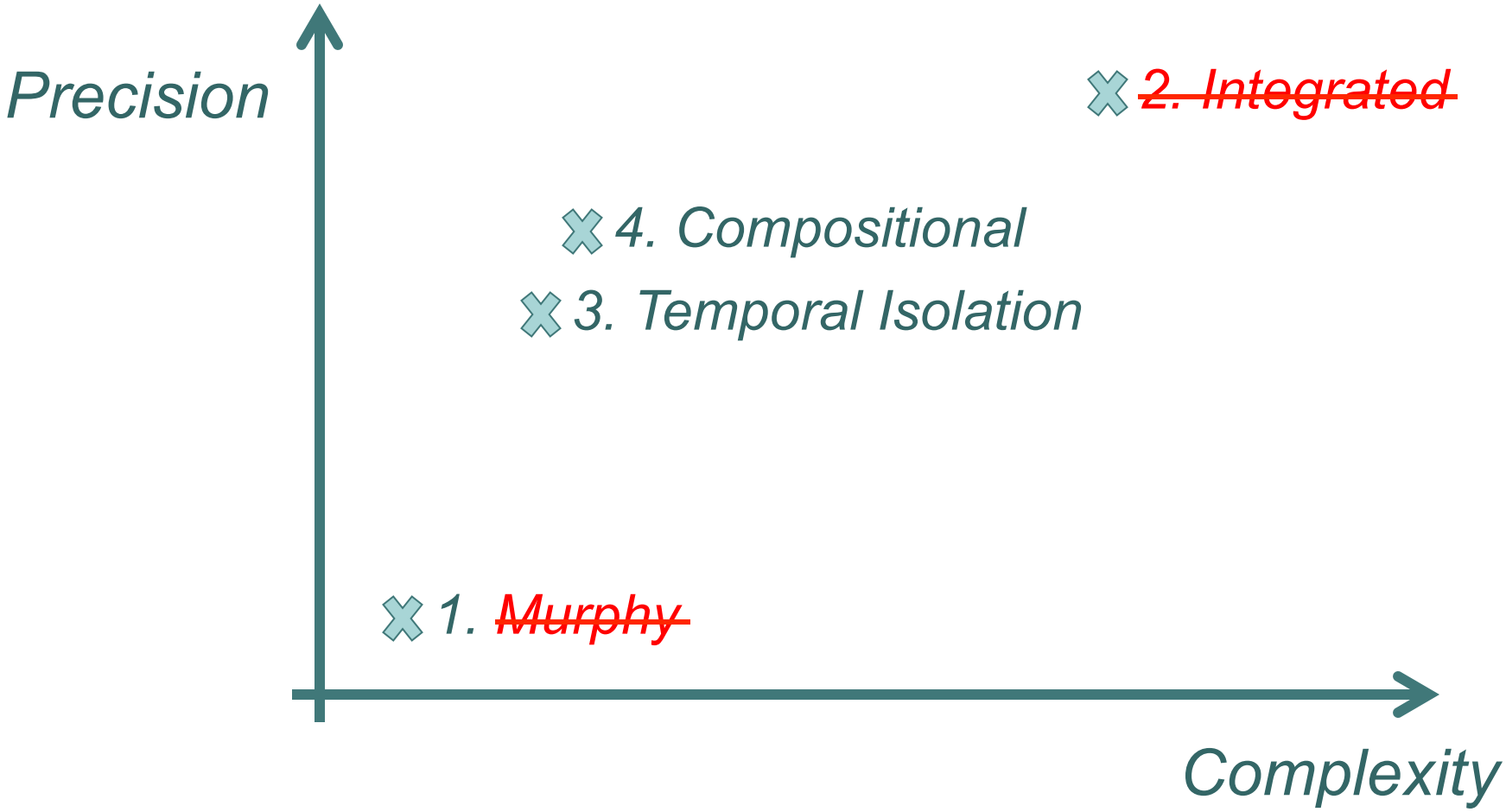
→ *Results will be extremely pessimistic*

2. Integrated Analysis Approach

Analyze entire task set at once in a combined WCET and response-time analysis

→ Infeasible even for the analysis of two co-running tasks

Four Approaches to Timing Analysis for Multi- and Many-Cores



3. Isolation Approach

Isolate tasks running on different cores by partitioning shared resources in

- space: caches, DRAM banks, and
- time: bus, network-on-chip.

Seemingly re-enables “single-core interface” between WCET and schedulability analysis.

Problem solved?

3. Isolation Approach

Challenge:

Determine partitioning of resources that ensures schedulability.

Approach:

Partitioning-aware WCET bound:

Mapping from partitioning parameters to WCET bounds [Reineke/Doerfert, RTAS 2014]

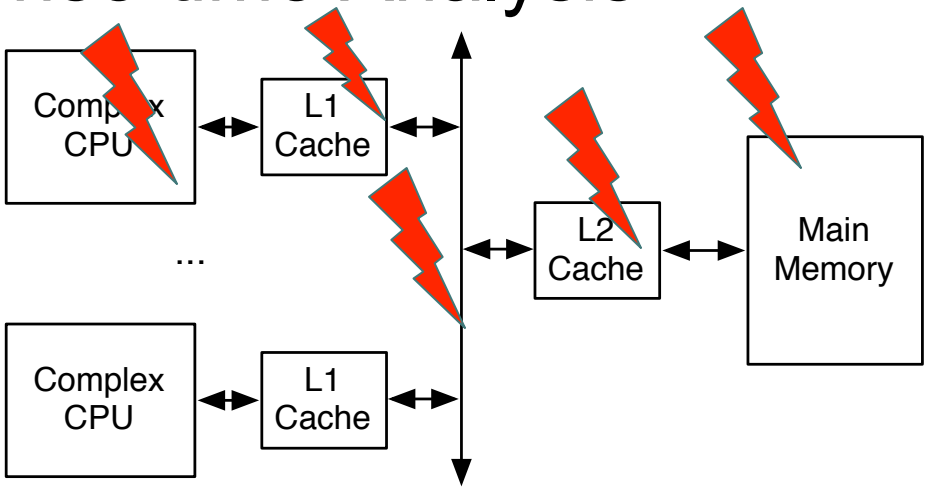
Open problem:
What is a good “partitioning-aware” scheduling policy?

4. Compositional Approach

1. “WCET Analysis”/“Low-level Analysis”: for each task:
   a) Compute WCET bound assuming no interference
   b) Compute maximal interference generated by task on each shared resource

2. Perform extended response-time analysis

4. Compositional Approach: Response-time Analysis



Response time of a task =

Corresponds to „regular“ response-time analysis:

- Execution time in isolation
- + Interference on its Core
- + Interference on Caches
- + Interference on Bus
- + Interference on Memory

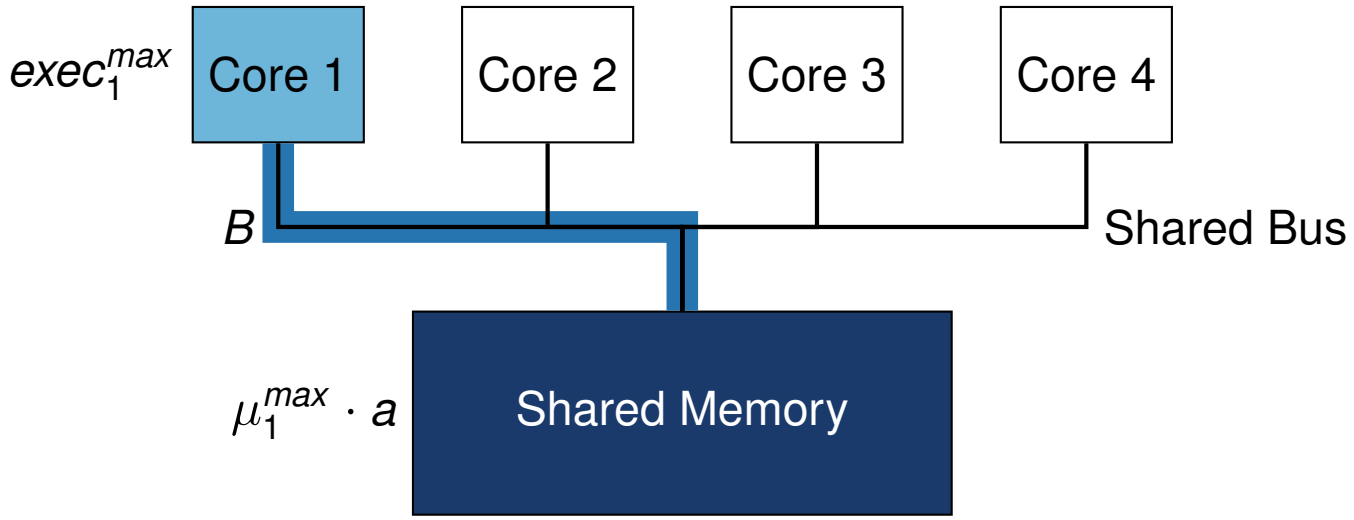
4. Compositional Approach Challenges

What are good interference characterizations?

→ Want precision and analysis efficiency

Approaches usually rely on timing compositionality.

Timing Compositionality: By Example



Timing Compositionality =
Ability to simply sum up timing contributions by different components

Implicitly or explicitly assumed by (almost) all approaches to timing analysis for multi cores and cache-related preemption delays (CRPD).

Conclusions

Multicores require rethinking interfaces between WCET analysis and response-time analysis

Promising Approaches:

- Temporal isolation:
  - Smart partitioning of resources required
- Compositional analysis:
  - Timing-compositionality assumption

Literature: Cache-related Preemption Delay

- Lee et al.: *Analysis of cache-related preemption delay in fixed-priority preemptive scheduling*. IEEE Trans Comput 47(6):700–713, 1998
- Tomiyama, Dutt: *Program path analysis to bound cache-related preemption delay in preemptive real-time systems*. In: Proceedings CODES, 2000
- Burguiere, Reineke, Altmeyer: *Cache Related Preemption Delay for Set-Associative Caches*, In: Proceedings WCET, 2009
- Altmeyer, Maiza, Reineke: *Resilience analysis: tightening the CRPD bound for set-associative caches*, In: Proceedings LCTES, 2010
- Altmeyer, Davis, Maiza: *Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems*, Real-Time Systems, 2012
- Reineke, Altmeyer, Grund, Hahn, Maiza; *Selfish-LRU: Preemption-Aware Caching for Predictability and Performance*, In: Proceedings RTAS, 2014

Literature: Interference Effects on Multi Cores

- Radojkovic et al.: On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments, ACM TACO, 2012.
- Abel et al.: Impact of Resource Sharing on Performance and Performance Prediction: A Survey. In: Proceedings CONCUR, 2013

Literature: Multi-core Timing Analysis

- Rosen, Andrei, Eles, Peng: Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In: Proceedings RTSS, 2007
- Pellizzoni, Schranzhofer, Chen, Caccamo, Thiele: Worst case delay analysis for memory interference in multicore systems, In: Proceedings DATE, 2010
- Schranzhofer, Pellizzoni, Chen, Thiele, Caccamo: Timing analysis for resource access interference on adaptive resource arbiters, In: RTAS, 2011
- Reineke and Doerfert: Architecture-parametric timing analysis, In: Proceedings RTAS, 2014
- Altmeyer, Davis, Indrusiak, Maiza, Nelis, Reineke: A generic and compositional framework for multicore response time analysis, In: Proceedings RTNS, 2015
- Huang, Chen, Reineke: MIRROR: symmetric timing analysis for real-time tasks on multicore platforms with shared resources, In: Proceedings DAC, 2016