Design and Analysis of Real-Time Systems

Predictability and Predictable Microarchitectures

Jan Reineke

Advanced Lecture, Summer 2013

Notion of Predictability

Oxford Dictionary:

- predictable = adjective, able to be predicted
- to predict = verb, state that a specified event will happen in the future

Fuzzy term in the WCET community.

May refer to the ability to predict:

- the WCET precisely,
- the execution time precisely,
- the WCET efficiently.

Notion of Predictability

Oxford Dictionary:

- predictable = adjective, able to be predicted
- to predict = verb, state that a specified event will happen in the future

Fuzzy term in the WCET community.

May refer to the ability to predict:

- the WCET precisely,
- the execution time precisely,
- the WCET efficiently.

How are these related?

Ability to predict the WCET precisely

In theory we can precisely "predict" (rather: determine) the WCET of most systems:

- enumerate all inputs
- enumerate all initial states of microarchitecture
- enumerate all possible environments

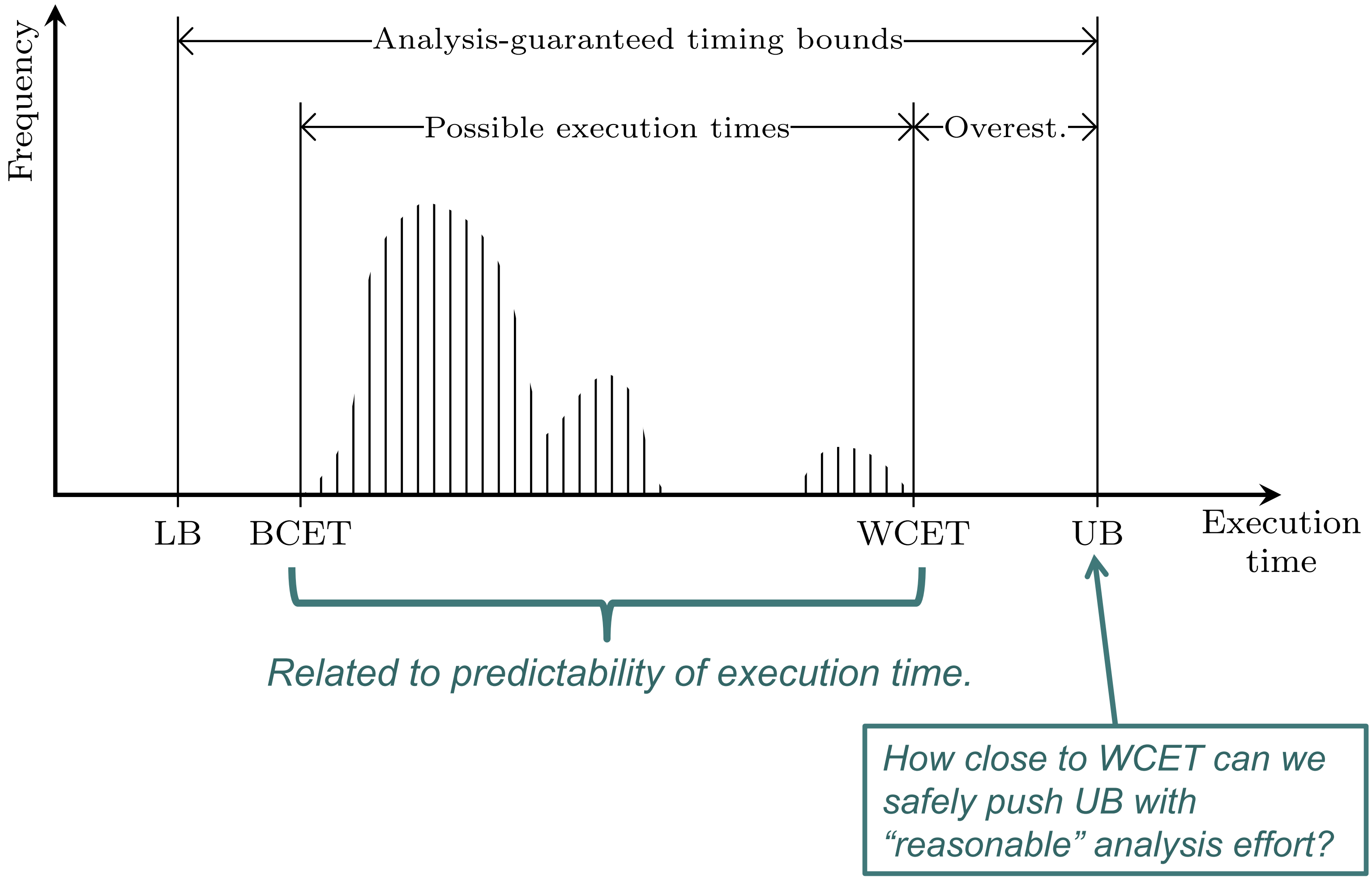
However, this is of course not feasible in practice.

→ Predictability of WCET is not the "right goal"

Contrast with ability to predict execution time:

→ Related to variability in execution times

Variability of Execution Times



Notion of Predictability

Fuzzy term in the WCET community.

May refer to the ability to predict:

- ~~the WCET precisely~~,
- the execution time precisely
  → execution-time predictability
- the WCET efficiently
  → analyzability

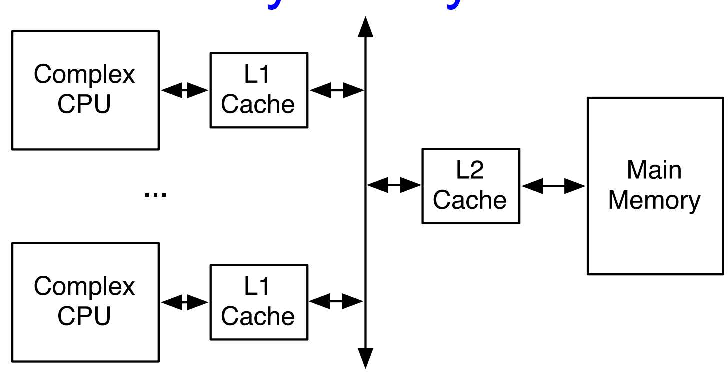
Challenges to Timing Predictability

Uncertainty about

- program inputs,
- initial state of microarchitecture, and
- activity in environment (e.g. other cores in multi-core), resulting in interference

→ introduces variability in execution times,
 thus decreases execution-time predictability.

→ introduces non-determinism in analysis,
 thus decreases analyzability.



Two Ways to Increase Predictability

1. Reduce uncertainty.
2. Reduce influence of uncertainty on
   a. Variability of execution times, and/or
   b. Analysis efficiency.

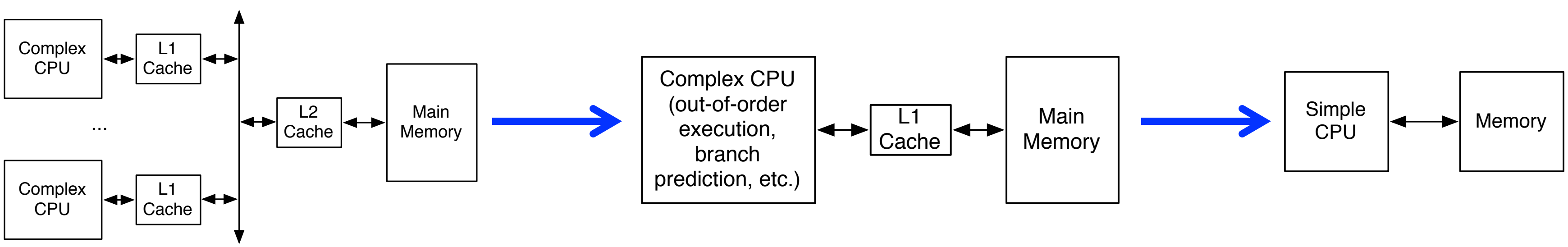
1. Reduce Uncertainty

- Reduce number of program inputs?

  Difficult…

- Reduce number of micro-architectural states:

  E.g. eliminate branch predictor, cache, out-of-order execution…

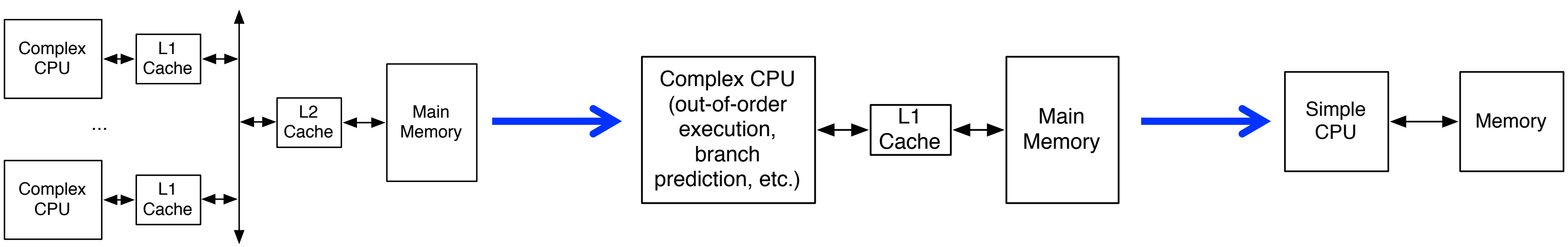


1. Reduce Uncertainty

- Reduce number of program inputs?

  Difficult…
- Reduce number of micro-architectural states:

  E.g. eliminate branch predictor, cache, out-of-order execution…



If done naively: Reverses many micro-architectural developments…
→ Decreases performance…
Key question: How to reduce uncertainty without sacrificing performance?

2.a) Reducing Influence of Uncertainty on Variability of Execution Times

If a source of uncertainty has no influence on execution times, it is irrelevant for timing analysis.

Example: Temporal Isolation

Temporal Isolation

- Temporal isolation between cores =

timing of program on one core is independent of activity on other cores

- Formally:

$$T(P_1, \langle p_1, c_1, p_2, c_2 \rangle) = T_{isolated}(P_1, \langle p_1, c_2 \rangle)$$

- Can be exploited in WCET analysis:

$$\begin{aligned} WCET(P_1) &= \max_{p_1, c_1, p_2, c_2} T(P_1, \langle p_1, c_1, p_2, c_2 \rangle) \\ &= \max_{p_1, c_1} T_{isolated}(P_1, \langle p_1, c_1 \rangle) \end{aligned}$$

Temporal Isolation
How to achieve it?

- Partition resources in space and/or time
  - Resource appears like a slower and/or smaller private resource to each client
- Examples:
  - Time-division multiple access (TDMA) arbitration in shared busses
  - Partitioned shared caches

- Why not simply provide private resources then?

2.b) Reducing Influence of Uncertainty on Analysis Efficiency

Does non-determinism have to be a problem for analyzability?

→ Timing Anomalies

→ Domino Effects

→ Lack of Timing Compositionality

- Eliminate Timing Anomalies,

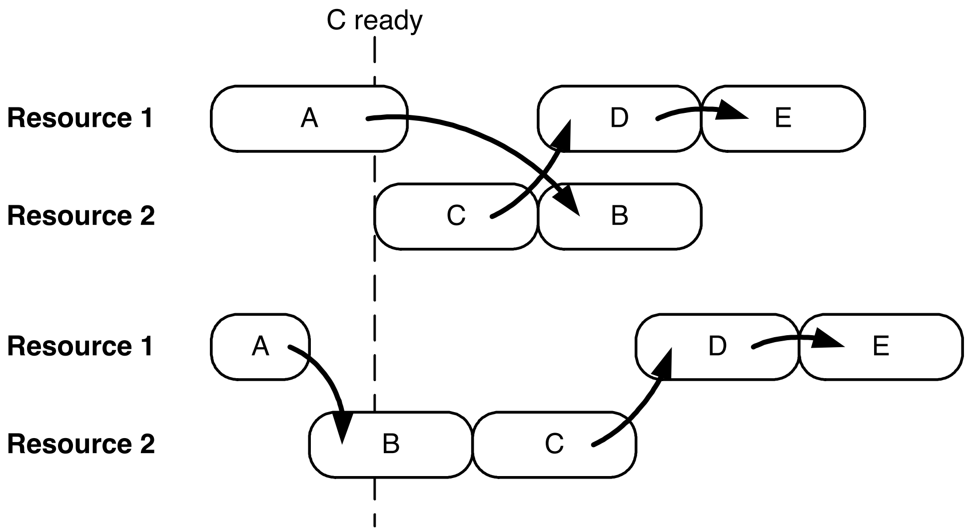
  e.g. stall pipeline on cache miss and use LRU.
- Eliminate Domino Effects

  e.g. use LRU rather than FIFO.

Timing Anomalies

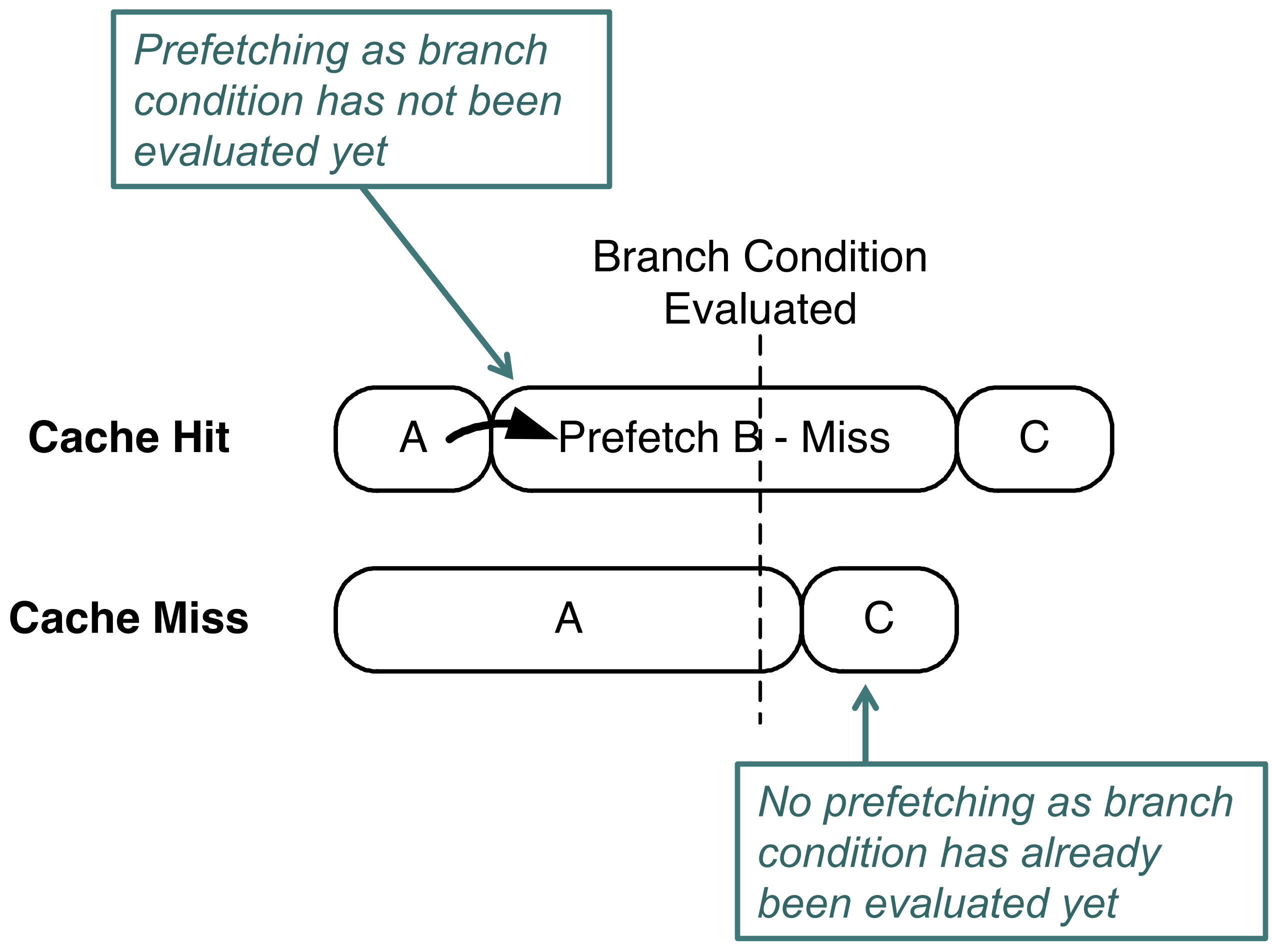
Timing Anomaly = Counterintuitive scenario in which the "local worst case" does not imply the "global worst case".

Example: Scheduling Anomaly

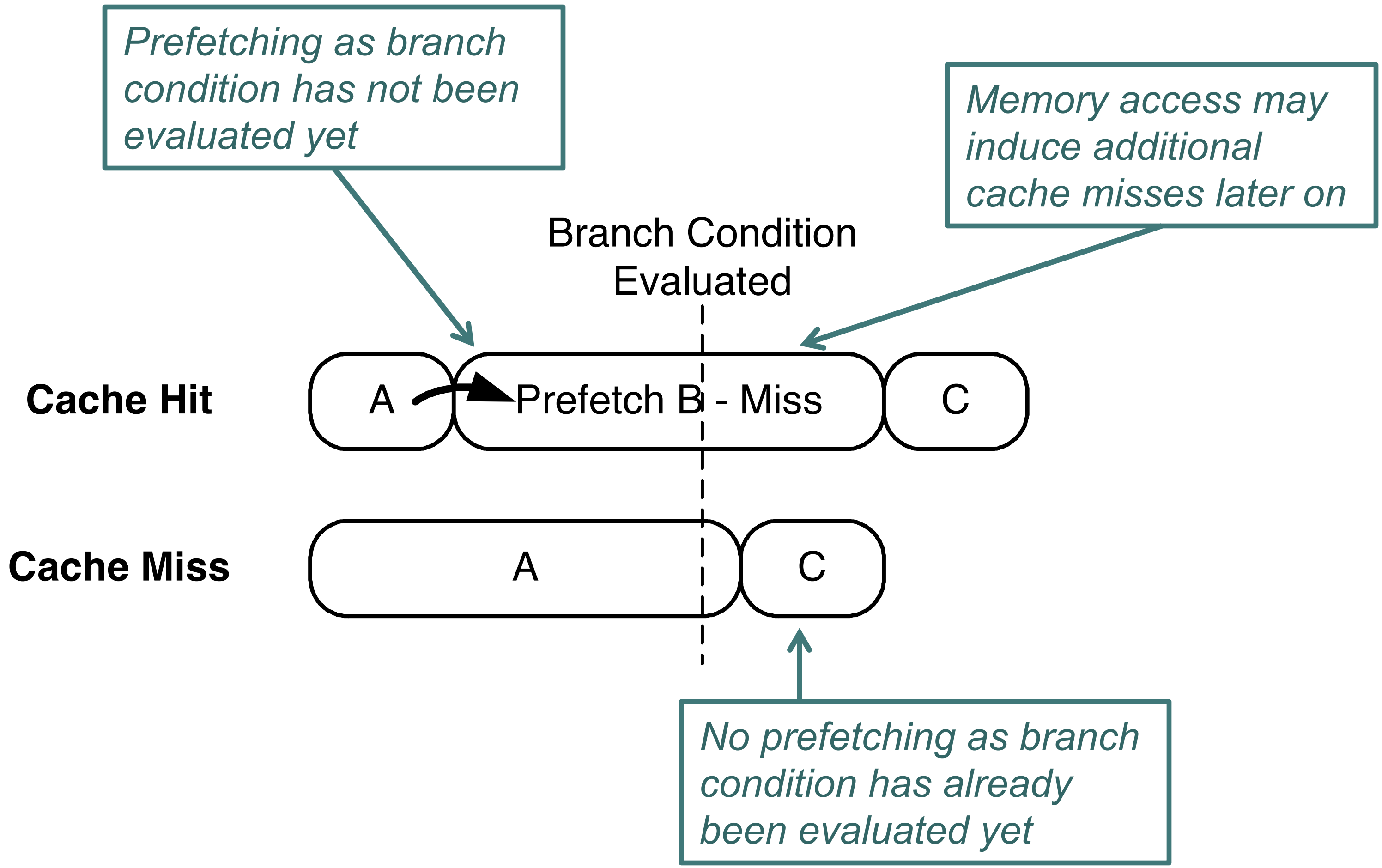


Recommended literature: *Bounds on multiprocessing timing anomalies*
RL Graham - SIAM Journal on Applied Mathematics, 1969 – SIAM
(http://epubs.siam.org/doi/abs/10.1137/0117039)

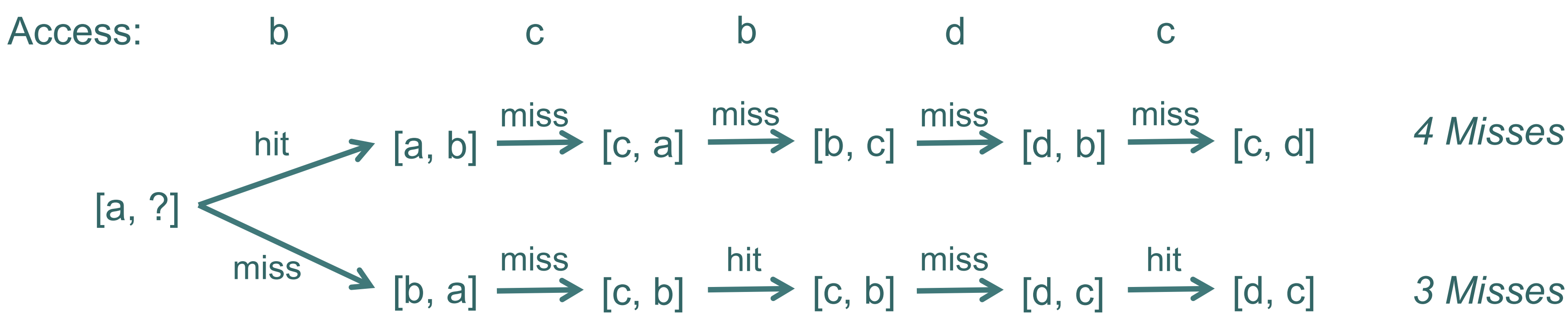
Timing Anomalies
Example: Speculation Anomaly



Timing Anomalies
Example: Speculation Anomaly

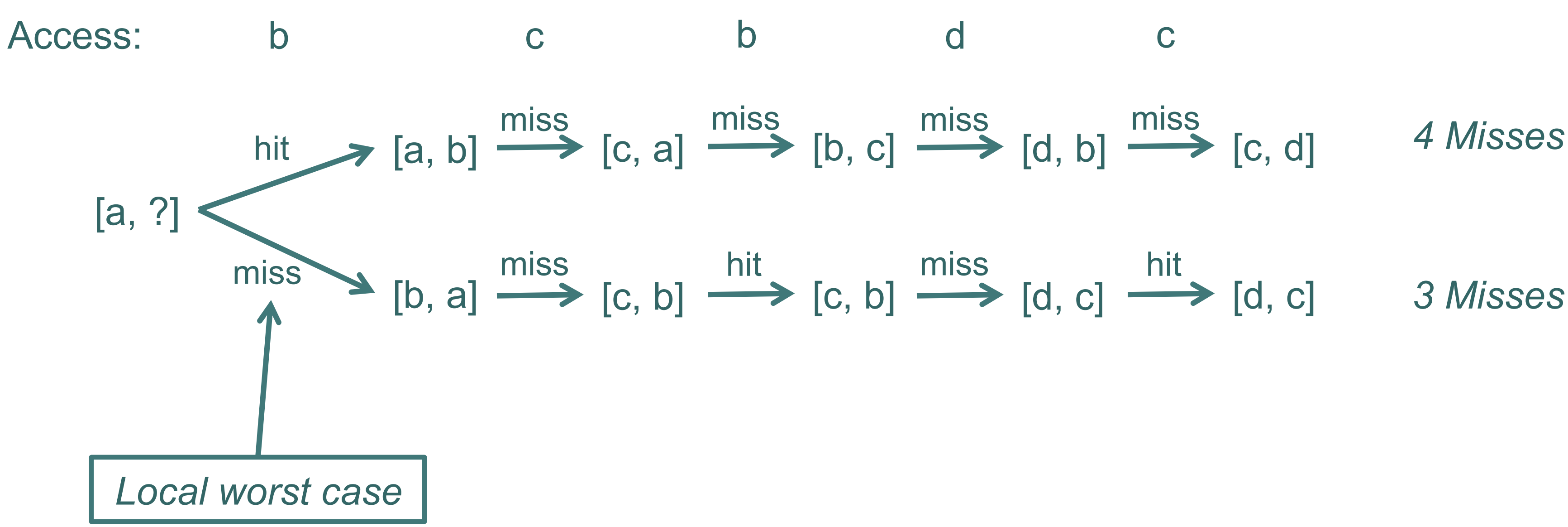


Timing Anomalies
Example: Cache Timing Anomaly of FIFO



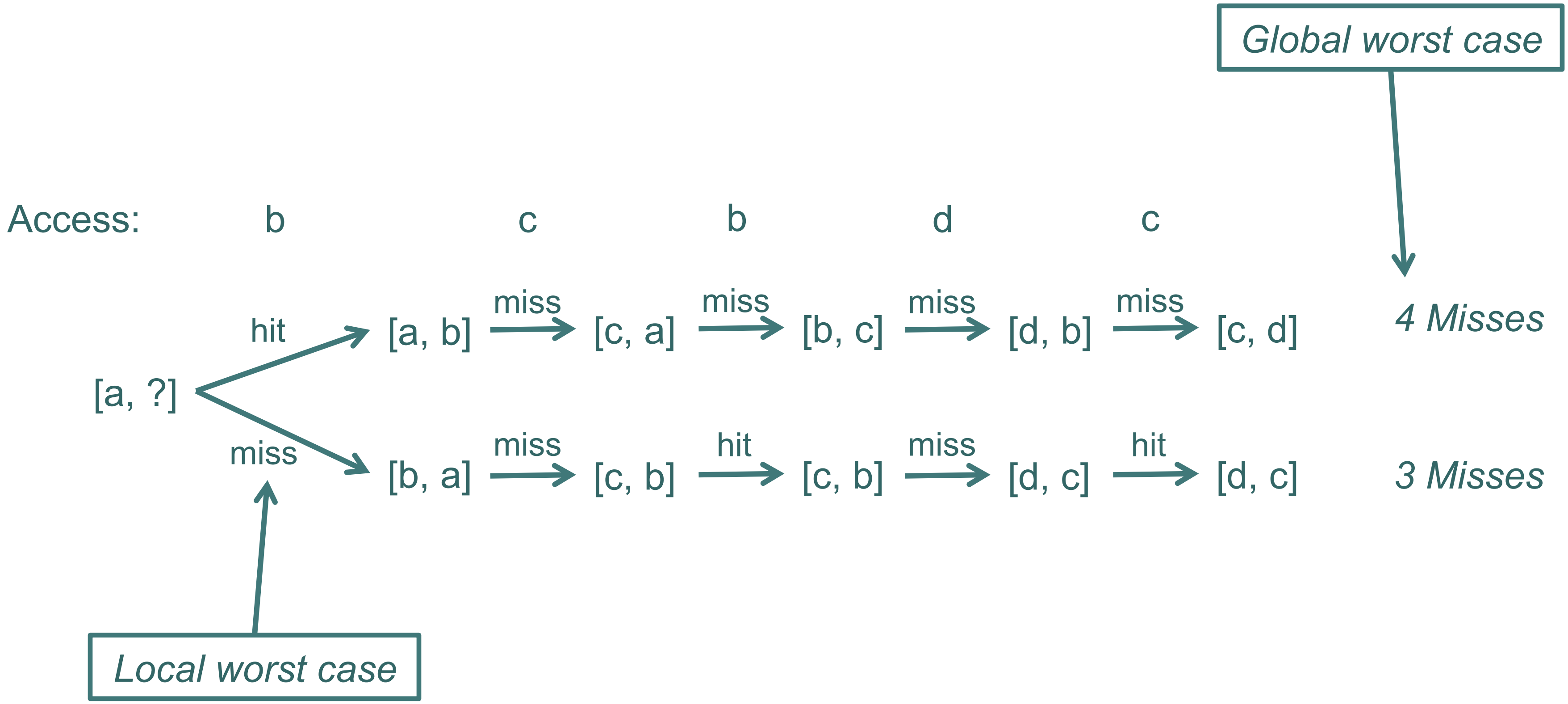
Similar examples exist for PLRU and MRU.
Impossible for LRU.

Timing Anomalies
Example: Cache Timing Anomaly of FIFO



Similar examples exist for PLRU and MRU.
Impossible for LRU.

Timing Anomalies
Example: Cache Timing Anomaly of FIFO

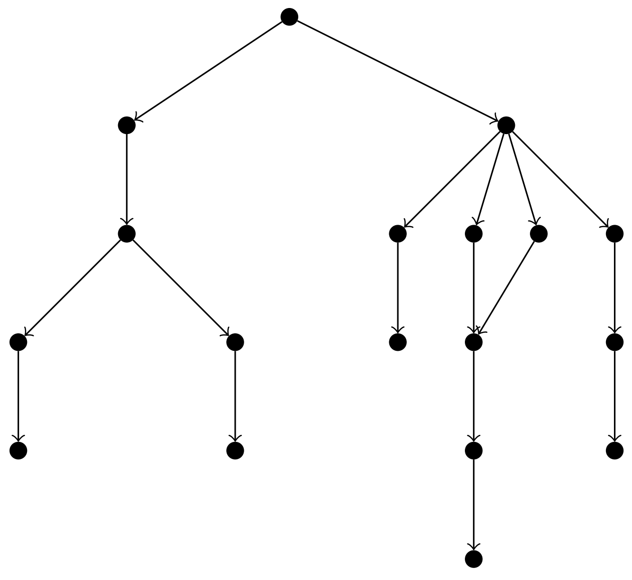


Similar examples exist for PLRU and MRU.
Impossible for LRU.

Timing Anomalies
Consequences for Timing Analysis

In the presence of timing anomalies, a timing analysis cannot make decisions “locally”: it needs to consider all cases.

→ May yield “State explosion problem”



Timing Anomalies
Open Analysis and Design Challenges

- How to determine whether a given timing model exhibits timing anomalies?
- How to construct processors without timing anomalies?
  - Caches: LRU replacement
  - No speculation
  - Other aspects: “halt” everything upon every “timing accident” → possibly very inefficient
- How to construct conservative timing model without timing anomalies?
  - Can we e.g. add a “safety margin” to the local worst case?

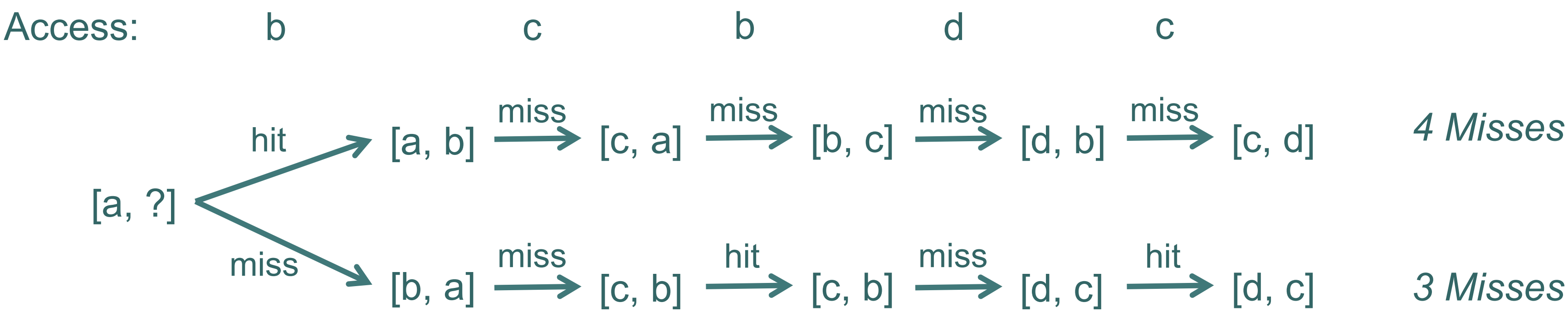
Domino Effects

- Intuitively:

  domino effect = “unbounded” timing anomaly

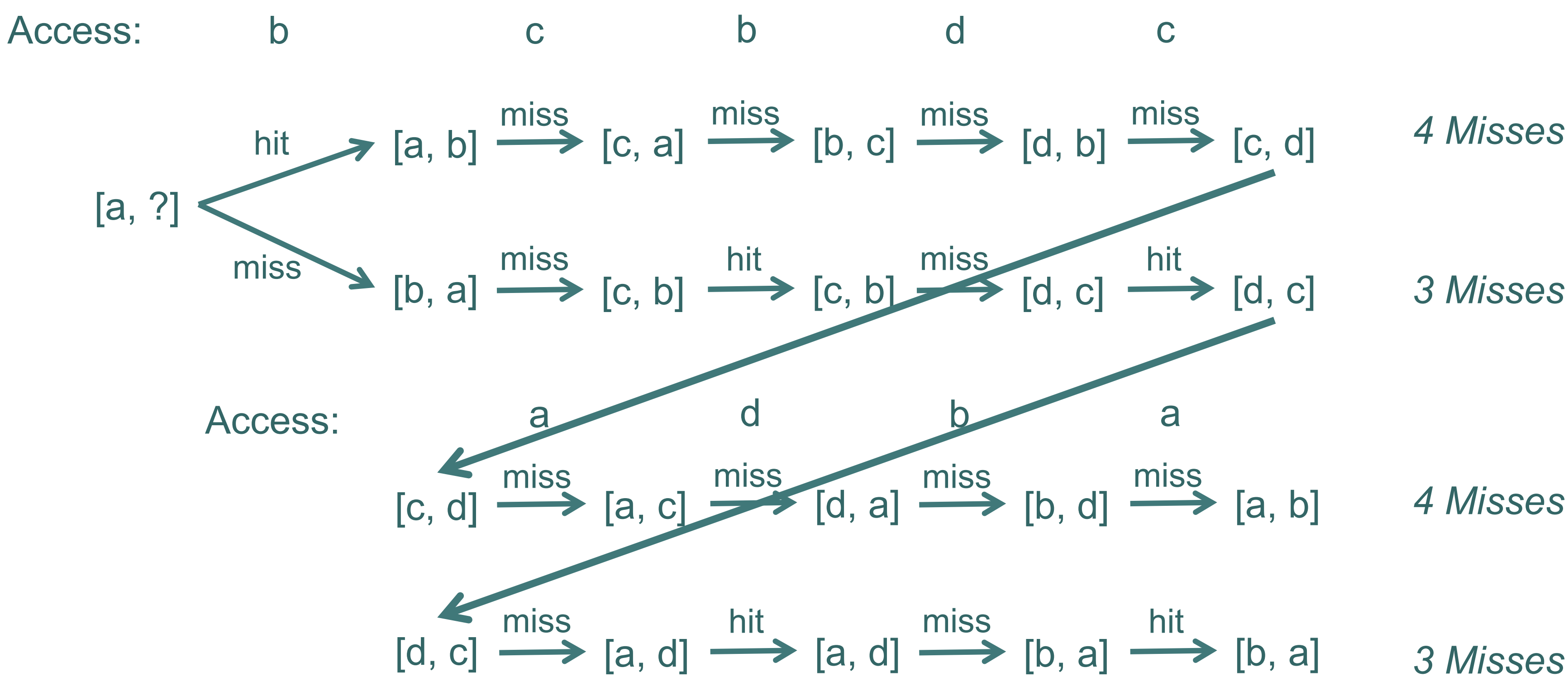
- Examples:
  - Pipeline (e.g. PowerPC 755)
  - Caches (FIFO, PLRU, MRU, …)

Domino Effects
Example: Cache Domino Effect of FIFO



Similar examples exist for PLRU and MRU.
Impossible for LRU.

Domino Effects
Example: Cache Domino Effect of FIFO



Similar examples exist for PLRU and MRU.
Impossible for LRU.

Domino Effects
Example: Cache Domino Effect of FIFO

Access: b c b d c

$$[a, ?] \xrightarrow{\text{hit}} [a, b] \xrightarrow{\text{miss}} [c, a] \xrightarrow{\text{miss}} [b, c] \xrightarrow{\text{miss}} [d, b] \xrightarrow{\text{miss}} [c, d] \quad \textit{4 Misses}$$

$$[a, ?] \xrightarrow{\text{miss}} [b, a] \xrightarrow{\text{miss}} [c, b] \xrightarrow{\text{hit}} [c, b] \xrightarrow{\text{miss}} [d, c] \xrightarrow{\text{hit}} [d, c] \quad \textit{3 Misses}$$

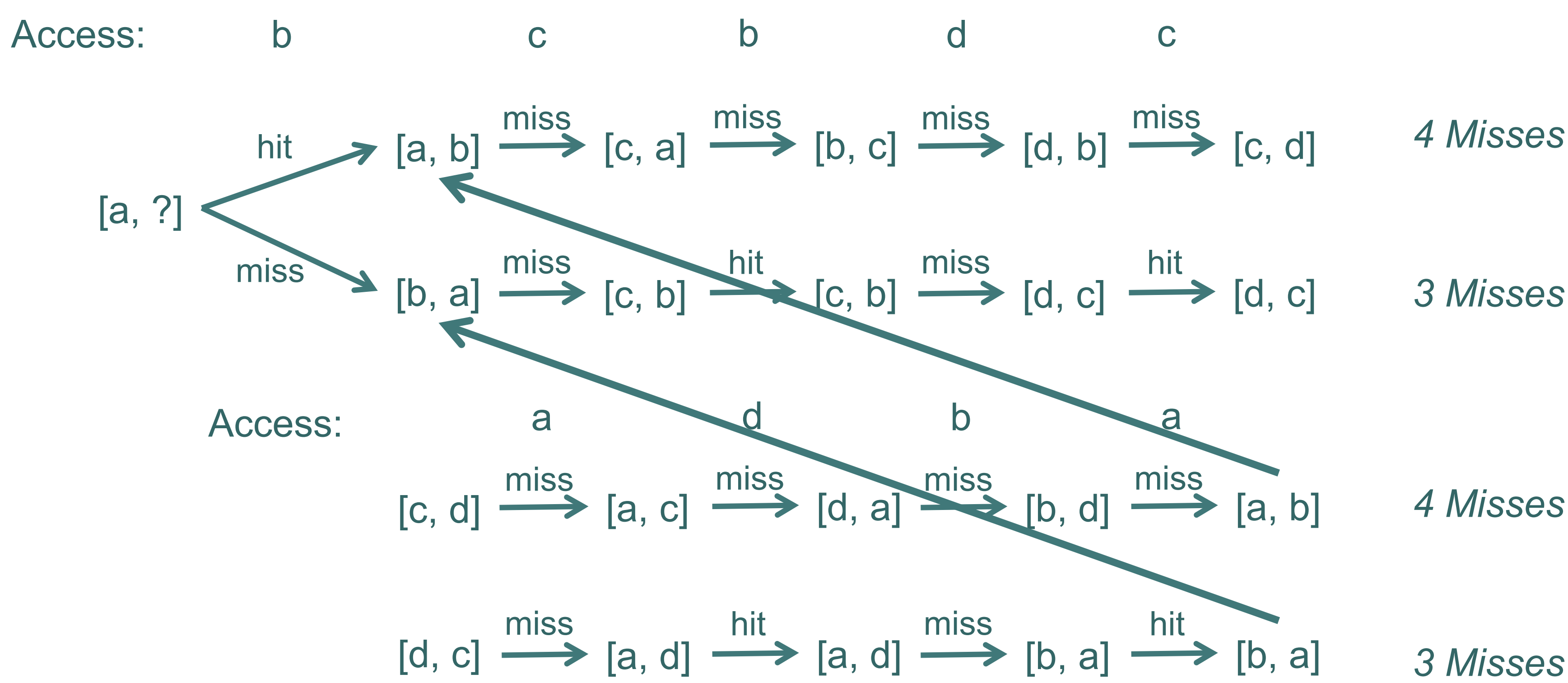
Access: a d b a

$$[c, d] \xrightarrow{\text{miss}} [a, c] \xrightarrow{\text{miss}} [d, a] \xrightarrow{\text{miss}} [b, d] \xrightarrow{\text{miss}} [a, b] \quad \textit{4 Misses}$$

$$[d, c] \xrightarrow{\text{miss}} [a, d] \xrightarrow{\text{hit}} [a, d] \xrightarrow{\text{miss}} [b, a] \xrightarrow{\text{hit}} [b, a] \quad \textit{3 Misses}$$

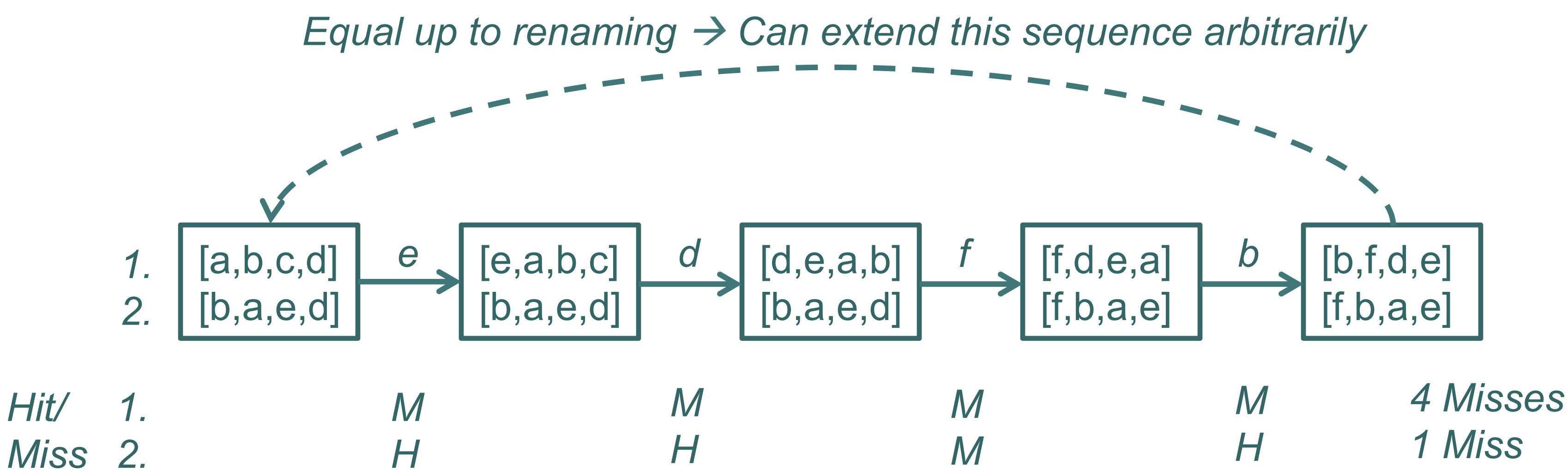
Similar examples exist for PLRU and MRU.
Impossible for LRU.

Domino Effects
Example: Cache Domino Effect of FIFO



Similar examples exist for PLRU and MRU.
Impossible for LRU.

Domino Effects
Example: FIFO Cache



Similar examples exist for PLRU and MRU.
Impossible for LRU.

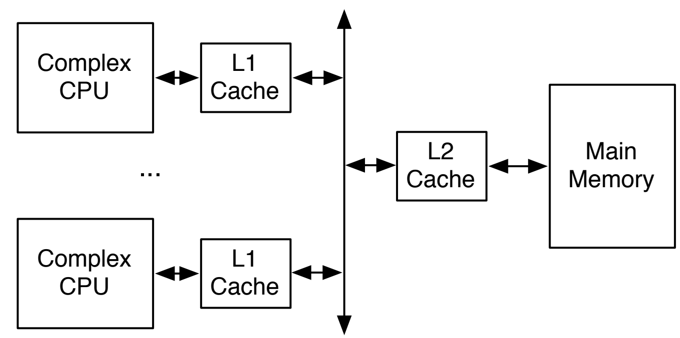
Domino Effects
Open Analysis and Design Challenges

Exactly as with timing anomalies:

- How to determine whether a given timing model exhibits domino effects?
- How to construct processors without domino effects?
- How to construct conservative timing model without domino effects?

Timing Compositionality Motivation

- Some timing accidents are hard or even impossible to statically exclude at any particular program point:
  - Interference on a shared bus: depends on behavior of tasks executed on other cores
  - Interference on a cache in preemptively scheduled systems
  - DRAM refreshes
- But it may be possible to make cumulative statements about the number of these accidents



Timing Compositionality
Intuitive Meaning

- Timing of a program can be decomposed into contributions by different “components”, e.g.
  - Pipeline
  - Cache non-preempted
  - Cache-related preemption delay
  - Bus interference
  - DRAM refreshes
  - …
- Example, decomposition into pipeline and cache

$$T_{pipeline,\ cache}(P, \langle p, c\rangle) = T_{pipeline}(P, \langle p\rangle) \oplus T_{cache}(P, \langle c\rangle)$$

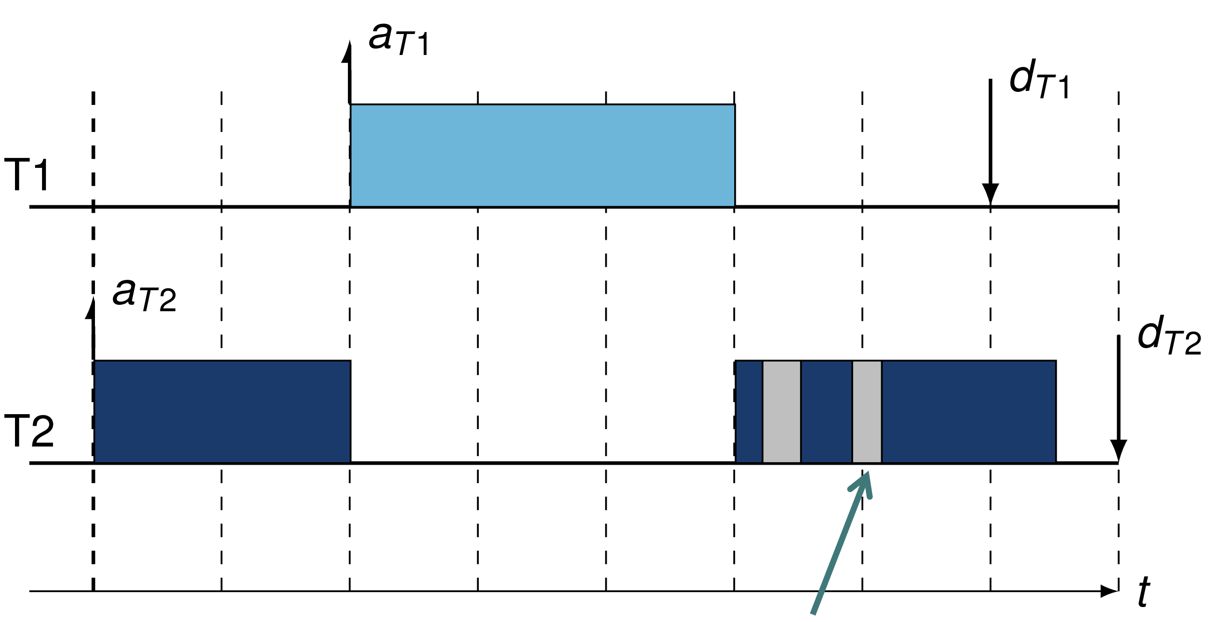
Timing Compositionality
Application in Timing Analysis

Then, the components (here: pipeline and cache) can also be analyzed separately:

$$\begin{aligned} WCET_{pipeline,\ cache}(P) &= \max_{p,c} T_{pipeline,\ cache}(P, \langle p, c \rangle) \\ &\leq \max_{p} T_{pipeline}(P, \langle p \rangle) \oplus \max_{c} T_{cache}(P, \langle c \rangle) \\ &= WCET_{pipeline}(P) + WCET_{cache}(P) \end{aligned}$$

Timing Compositionality
Example: “Cache-aware” Response-Time Analysis

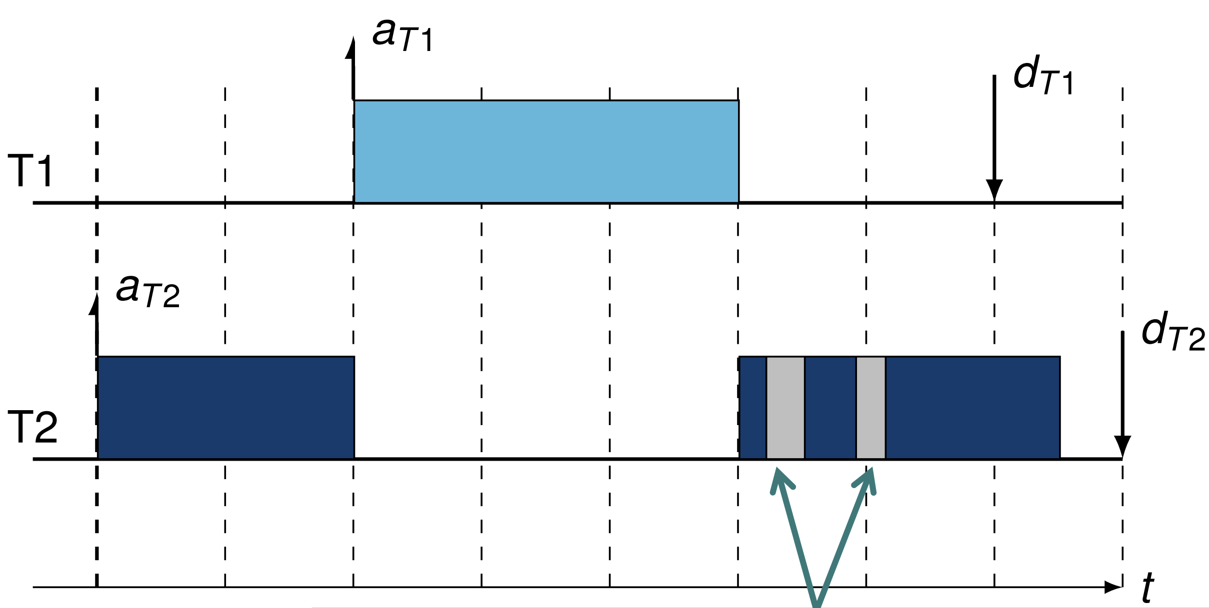
In preemptive scheduling, preempting tasks may “disturb” the cache contents of preempted tasks:



Timing Compositionality

Example: “Cache-aware” Response-Time Analysis

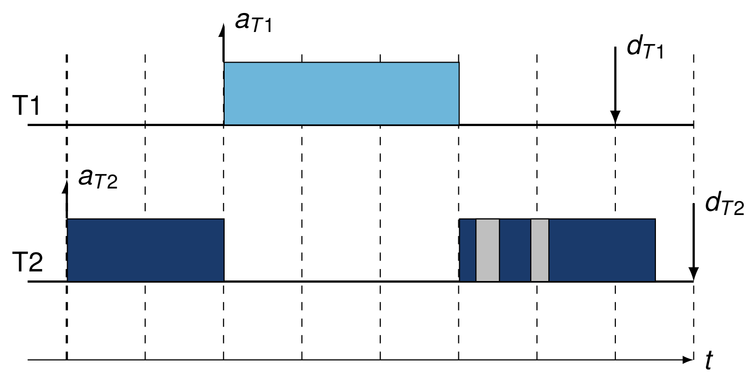
In preemptive scheduling, preempting tasks may “disturb” the cache contents of preempted tasks:



Additional misses due to preemption, referred to as the Cache-Related Preemption Delay (CRPD).

Timing Compositionality

Example: “Cache-aware” Response-Time Analysis



Timing decomposition:

- WCET of T1 without preemptions: C1
- WCET of T2 without preemptions: C2
- Additional cost of T1 preempting T2:

$$CRPD_{1,2} = BRT * \#additional\ misses$$

→ Response time of T2:

$$R2 \leq C2 + \#preemptions \times (C1 + CRPD_{1,2})$$

Timing Compositionality
Open Analysis and Design Challenges

- How to check whether a given decomposition of a timing model is valid?
- How to compute bounds on the cost of individual events, such as cache misses (BRT in previous example) or bus stalls?
- How to build microarchitecture in a way that permits a sound and precise decomposition of its timing?

Summary:
Approaches to Increase Predictability

Reduce size by simplifying microarchitecture, e.g. eliminate cache, branch prediction, etc.

Reduce influence of uncertainty on executions, e.g. by temporal isolation

Decouple analysis efficiency from number of executions:
- *Eliminate timing anomalies and domino effects,*
- *Achieve timing compositionality*

e.g. by LRU replacement, stalling pipeline upon cache miss

Uncertainty about Inputs → *Possible Executions* → *Analysis Efficiency*

Program inputs
Initial state of microarchitecture
Tasks on other cores

Summary

- (Fuzzy) notions of timing predictability
- Important related notions:
  - Timing anomalies
  - Domino effects
  - Timing compositionality
  - Temporal isolation
- Two ways of increasing predictability:
  1. Reduce uncertainty
  2. Reduce influence of uncertainty