Model-based design of real-time systems

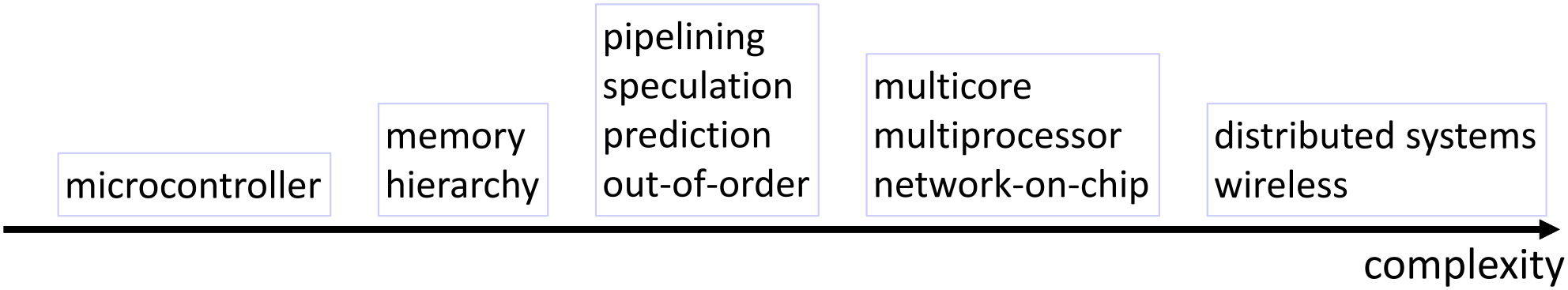
Contents

- Why?
- Interference
- Task Level
- Task Graph Level
- Outlook and Summary

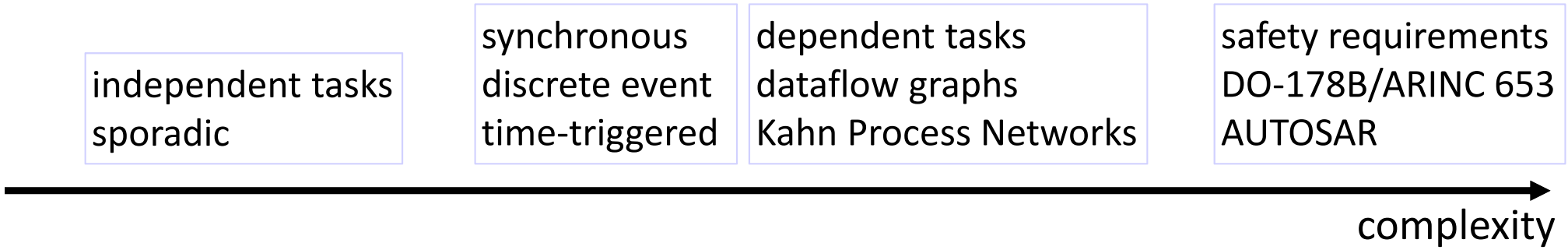
Contents

- ***Why?***
- Interference
- Task Level
- Task Graph Level
- Outlook and Summary

Platform

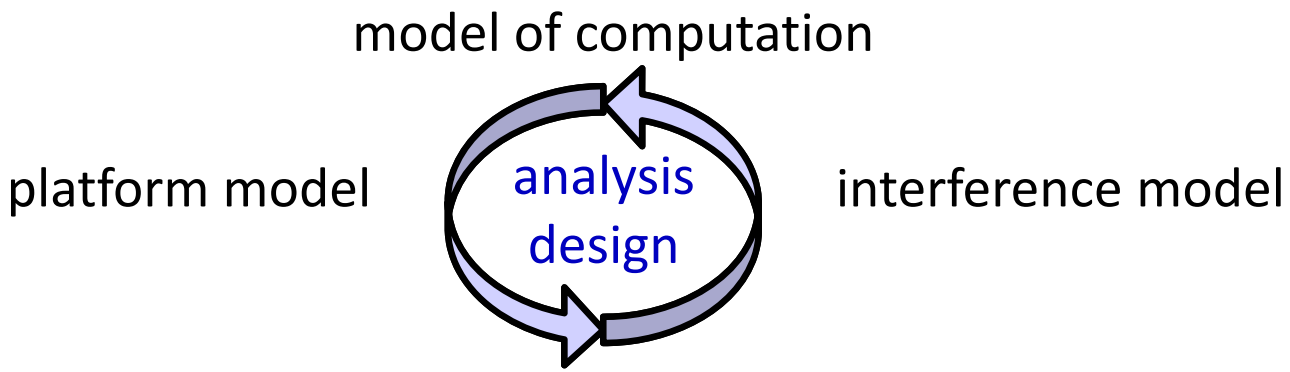


Application



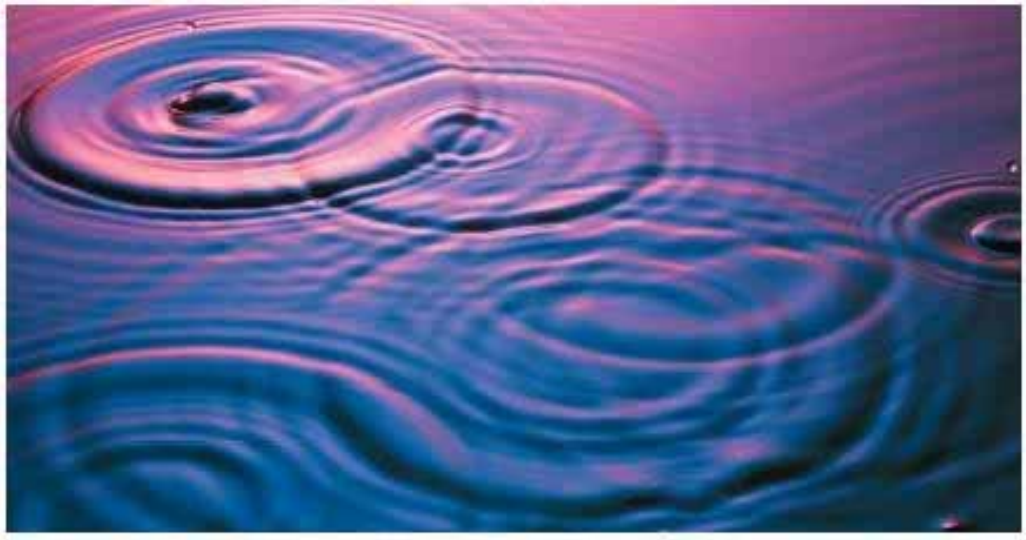
- What are essential application and platform characteristics?
  - What are the dominating interference mechanisms?
  - What are useful abstraction models?
  - What are corresponding analysis methods?

- What are essential application and platform characteristics?
  - What are the dominating interference mechanisms?
  - What are useful abstraction models?
  - What are corresponding analysis methods?



Contents

- Why?
- ***Interference***
- Task Level
- Task Graph Level
- Outlook and Summary



Scope

Interference:
The execution of a task influences that of some other task.

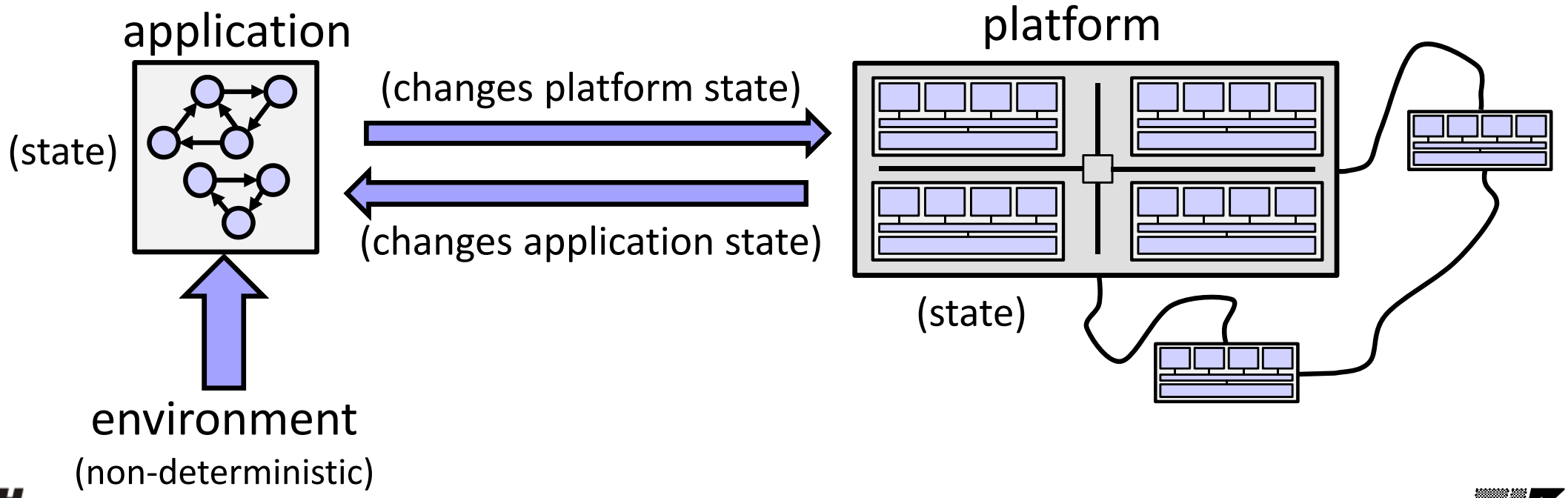
- Interference mechanisms
  - **Time**: sharing resources in time, e.g. computation and communication
  - **Space**: sharing resources in “space”, e.g. memory
  - **Temperature**: heating up each other
  - **Energy**: sharing the energy source

Why do we care?

- Interference often
  - endagers safety → isolation
  - decreases efficiency and schedulability → performance
  - complicates analysis → guarantees

Why do we care?

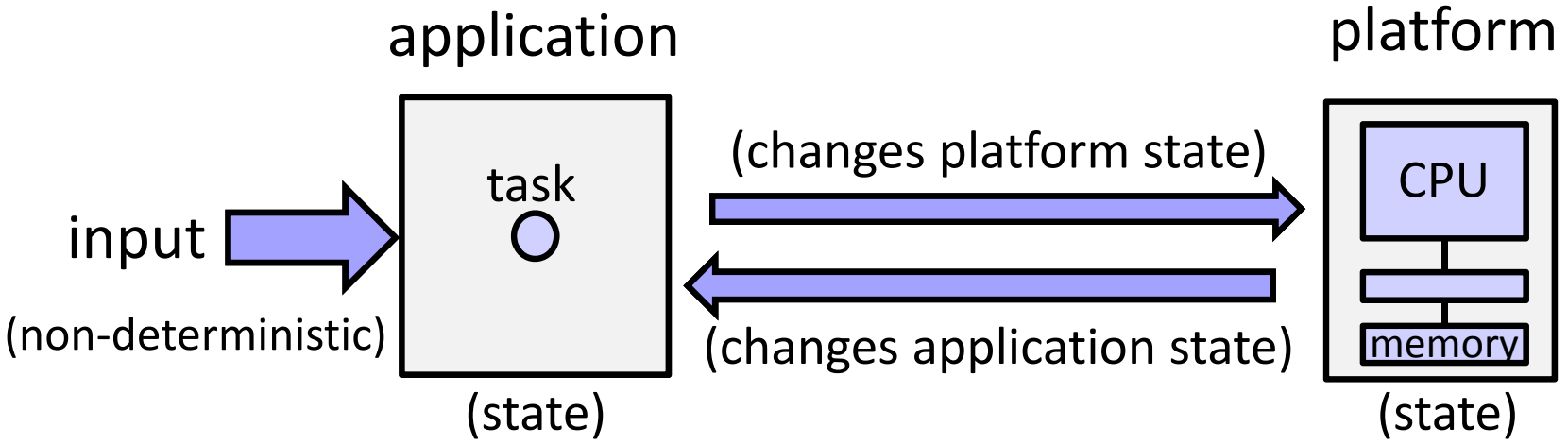
- Interference often
  - reduces safety → isolation
  - decreases efficiency and schedulability → performance
  - complicates analysis → guarantees



Contents

- Why?
- Interference
- ***Task Level***
  - ***single task on single processor***
  - several tasks on single processor
  - single tasks on parallel processors
- Task Graph Level
- Outlook and Summary

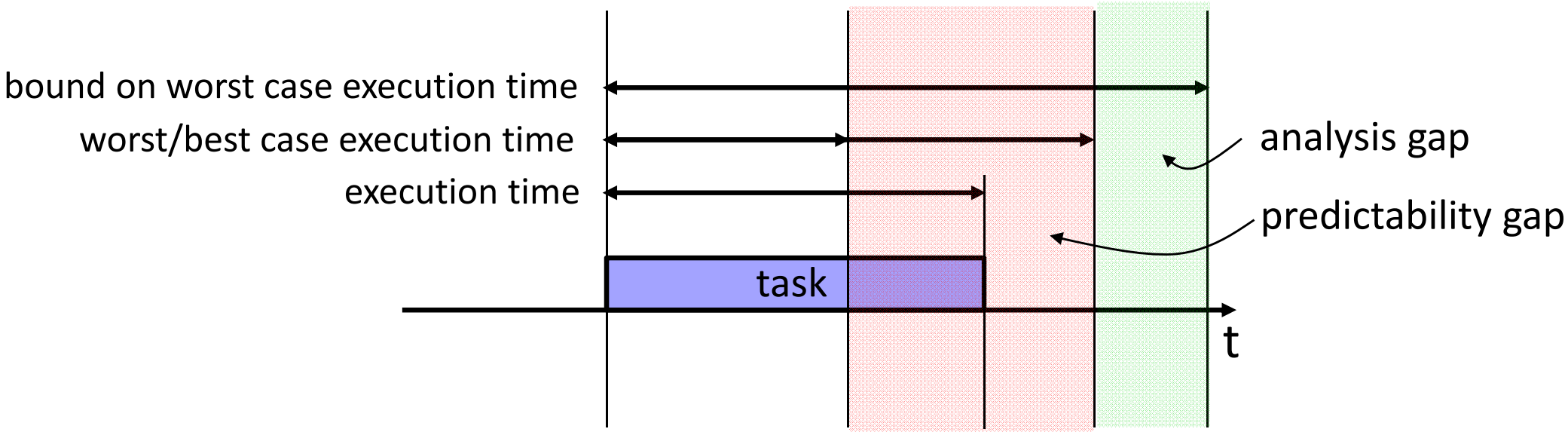
Single Task – Single Processor



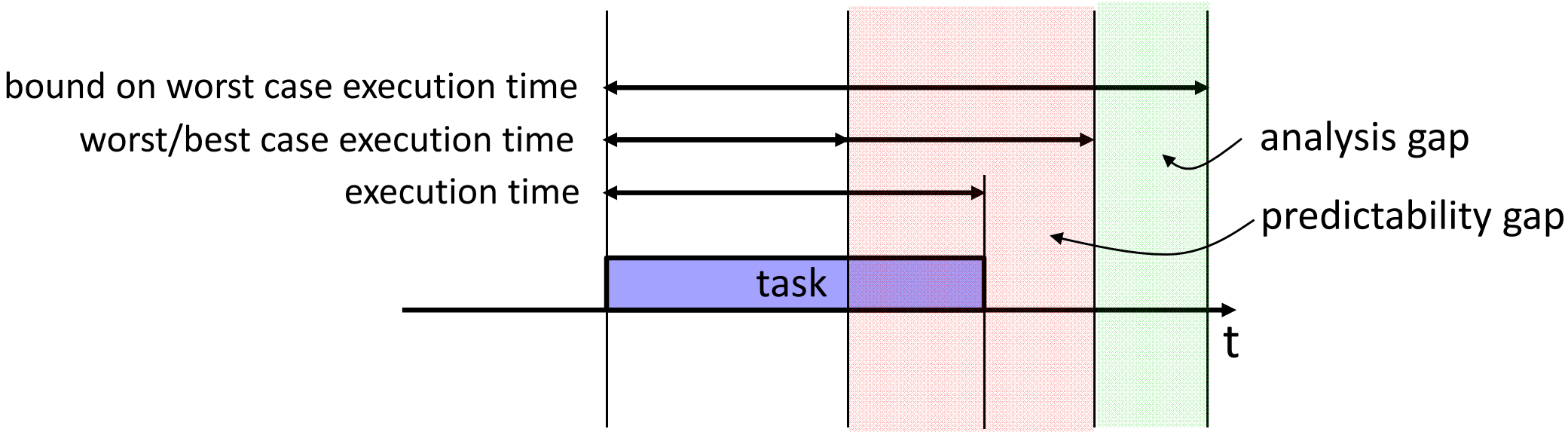
- Program path
- Program counter
- Local memory
- …

- Register values
- Cache state
- Pipeline state
- Branch prediction
- …

Single Task – Single Processor



Single Task – Single Processor



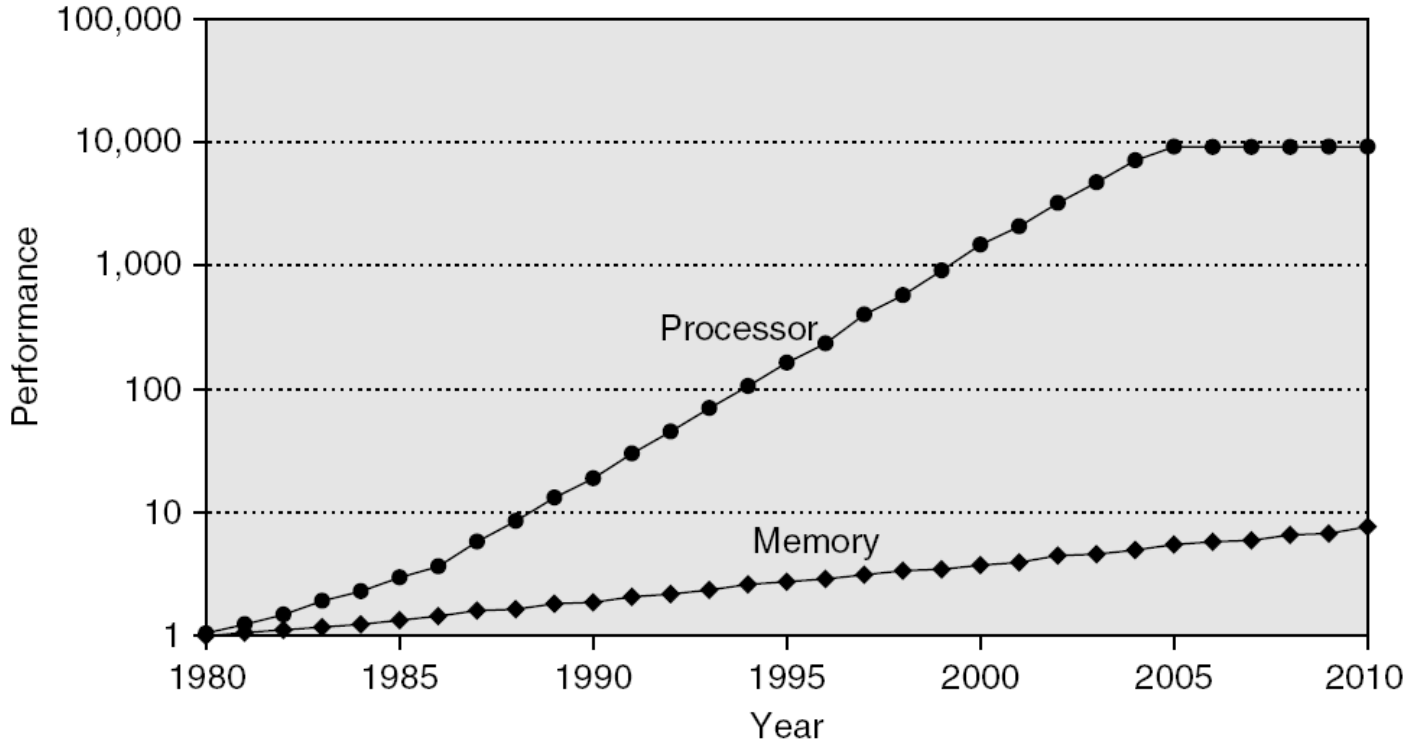
gaps increase due to:

- increasing penalty
- large state space
- long dependencies
- timing anomalies

Penalty

- *Example:*

  Single core memory vs. processor performance



©Hennessy/Patterson

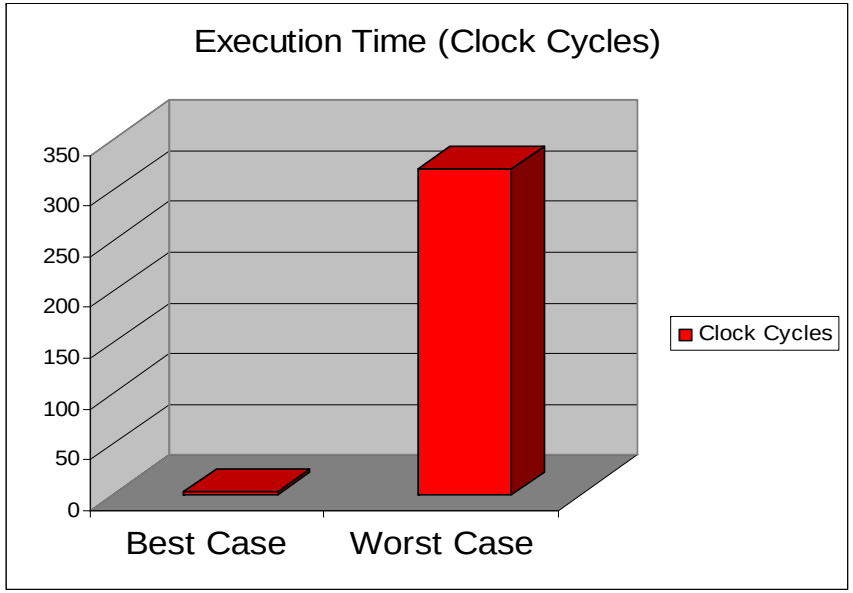
Is it an important effect?

- Penalty:

x = a + b;

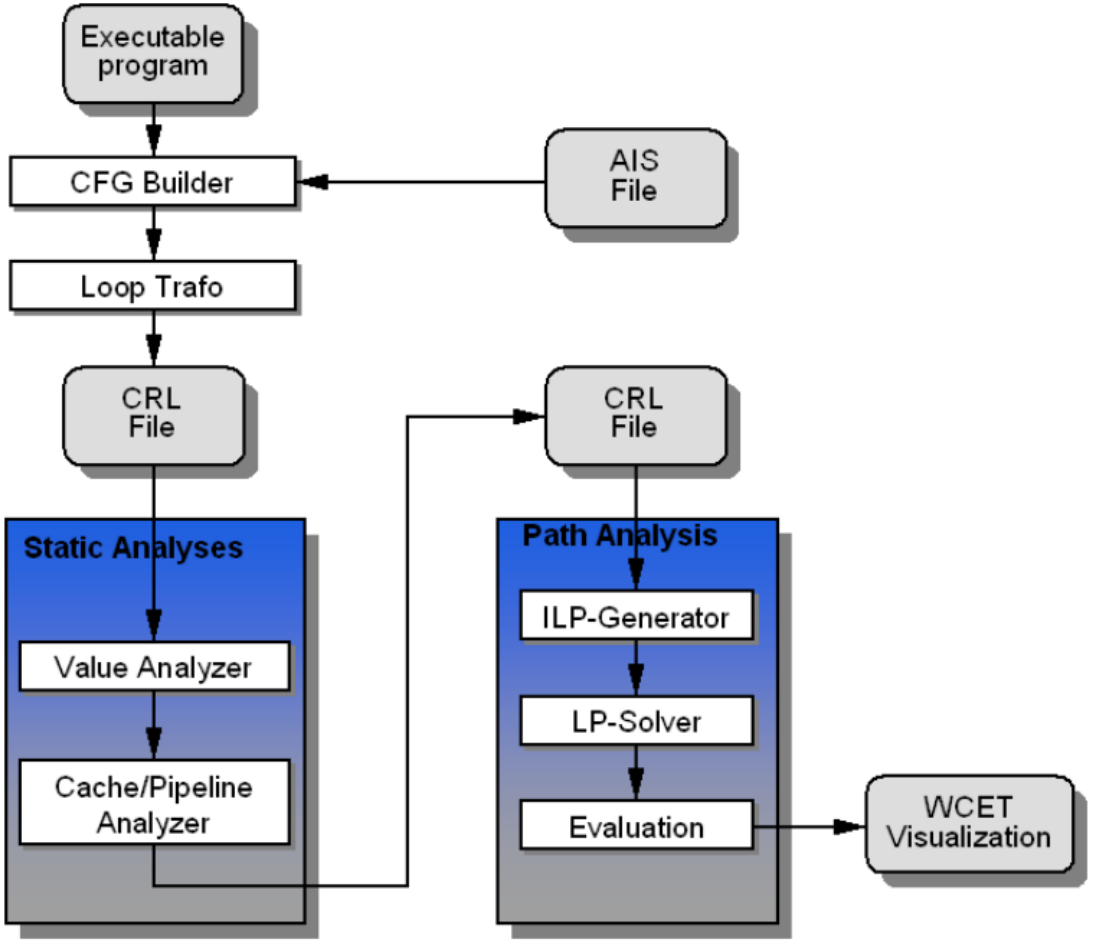
```
LOAD    r2, _a
LOAD    r1, _b
ADD     r3,r2,r1
```

PPC 755



Counter Measures

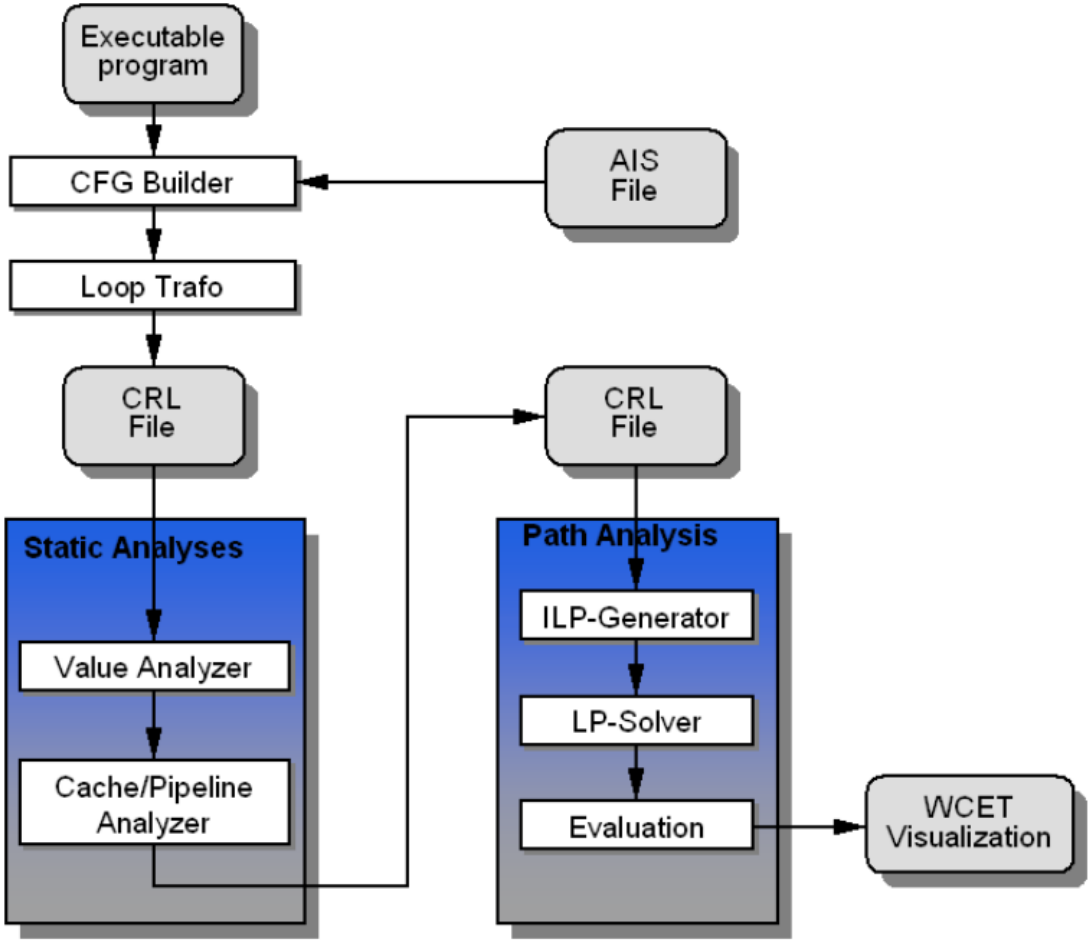
- analysis tools



© R. Wilhelm et al. , Absint/AiT

Counter Measures

- analysis tools

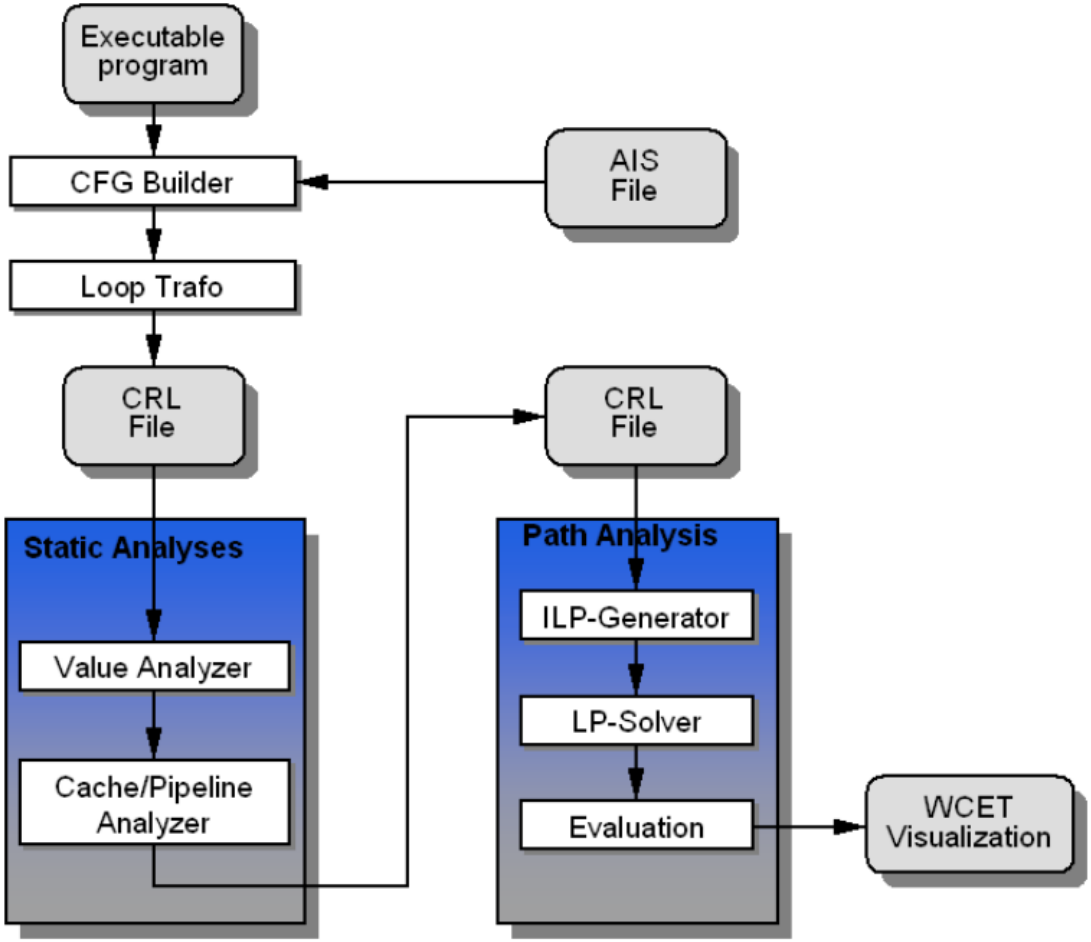


© R. Wilhelm et al. , Absint/AiT

- predictable architectures
  - reduce interference
  - reduce state
  - reduce timing anomalies
- improve analysis
- new reclaiming strategies (mixed criticality)
- reduce safety level

Counter Measures

- analysis tools



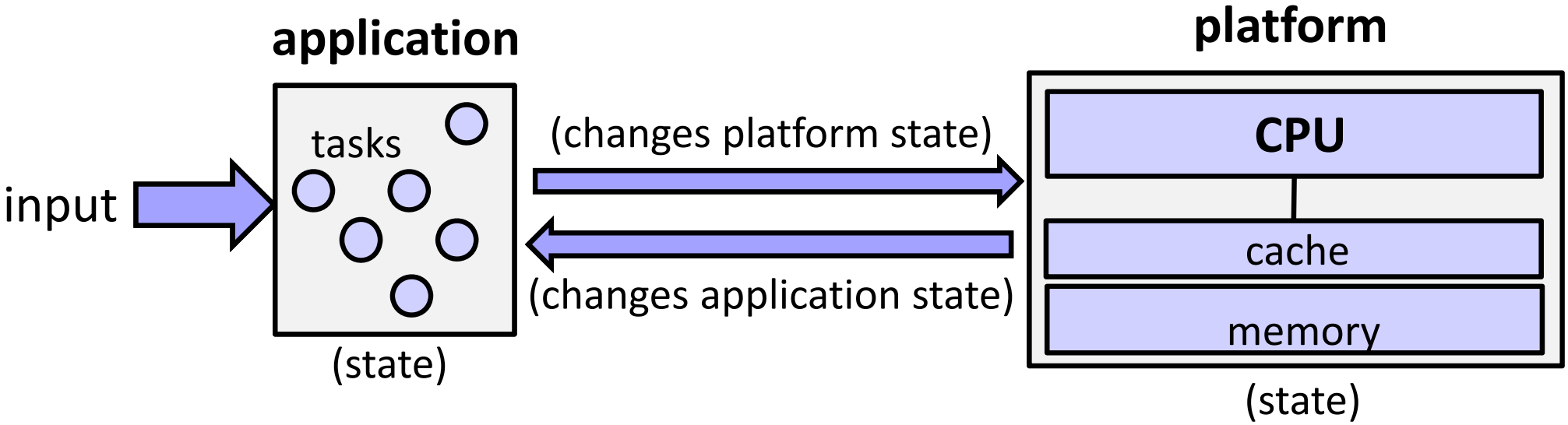
© R. Wilhelm et al. , Absint/AiT

- predictable architectures
  - reduce interference
  - reduce state
  - reduce timing anomalies
- improve analysis
- new reclaiming strategies (mixed criticality)
- reduce safety level

give up

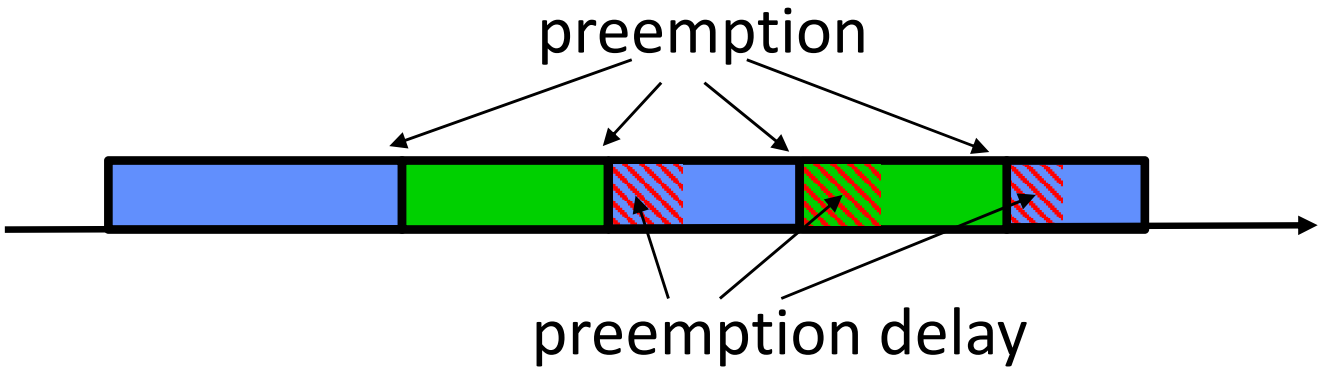
Contents

- Why?
- Interference
- ***Task Level***
  - single task on single processor
  - ***several tasks on single processor***
  - single tasks on parallel processors
- Task Graph Level
- Outlook and Summary


application
platform
tasks
input
(changes platform state)
CPU
cache
memory
(changes application state)
(state)
(state)

Interference Mechanism: Memory

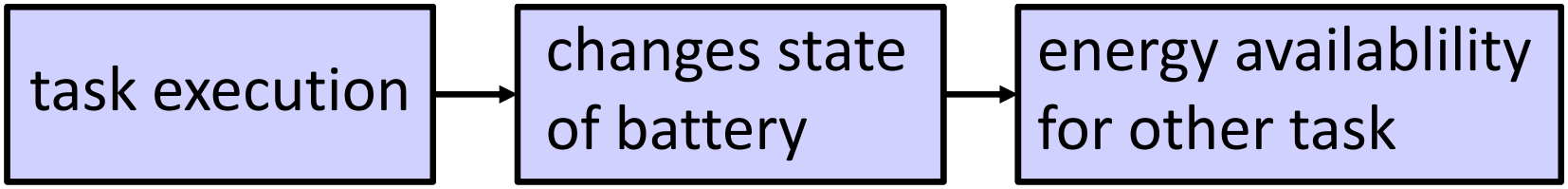
- **We only look at additional interferences (besides computation resource).**
- ***Cache:***
  - Cache-related preemption delay: impact of preemption on the cache content [Chang-Gun Lee et al. 1996].



  - Preemption points in order to reduce penalty and non-determinism [Mitra et al. 2003, Ramaprasad et al. 2006].

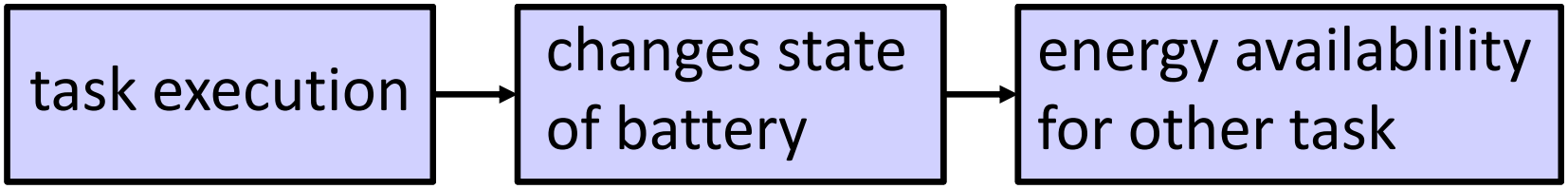
Interference Mechanism: Energy

- ***Energy Harvesting:***
  - **Interference: The execution of a task influences that of some other task.**



Interference Mechanism: Energy

- ***Energy Harvesting:***
  - **Interference: The execution of a task influences that of some other task.**



- *Typical question:*
  - Given a task model and a bound on the harvested energy.
  - Determine an ***optimal on-line scheduling*** algorithm:

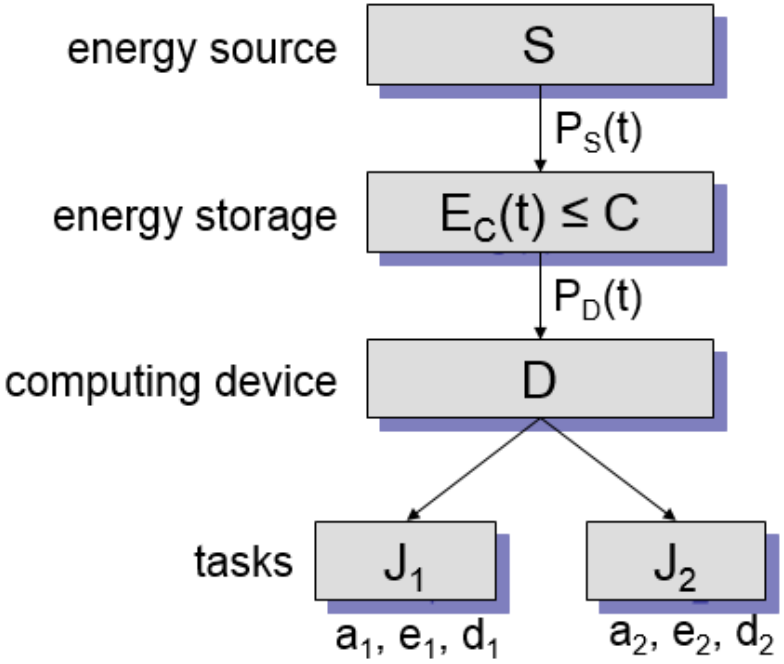
    If the task set is schedulable, it determines a feasible schedule.
  - Construct an ***admittance test***:

    Determine, whether a set of event streams with a given characteristic is schedulable.

[Moser et al. 2006]

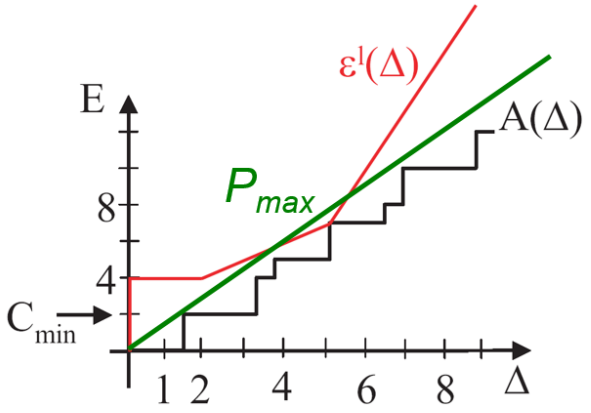
Interference Mechanism: Energy

- Example of a task model [Moser et al. 2006]:



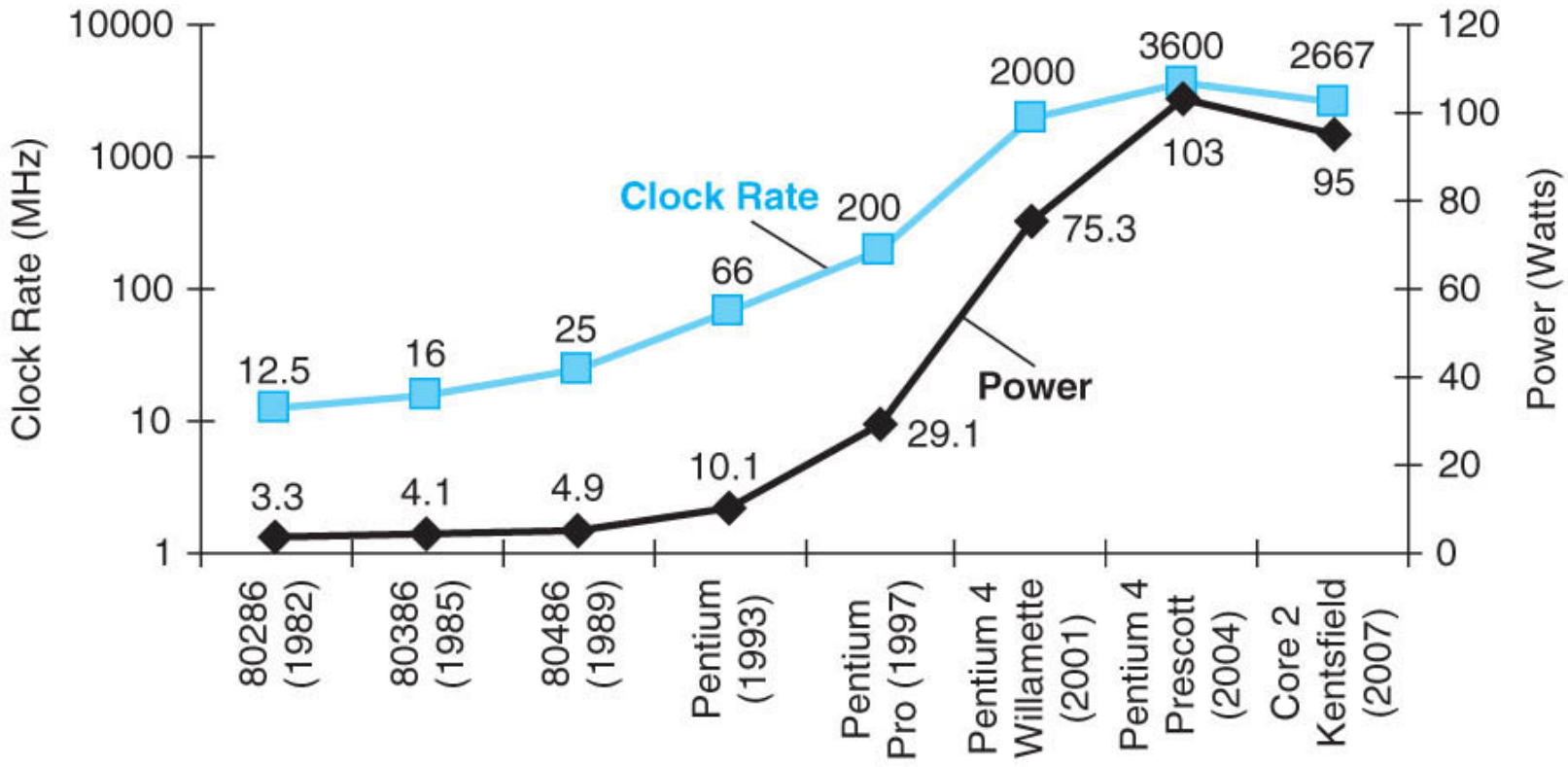
- Lazy scheduling: A task set is schedulable iff

$$\forall \Delta : \quad \sum_{i \in \mathbf{I}} e_i \alpha_i (\Delta - d_i) \leq \min\{\epsilon^l(\Delta) + C, P_{max}\Delta\}$$



Interference Mechanism: Temperature

Clock Speed and Power Consumption for Intel x86 Processors



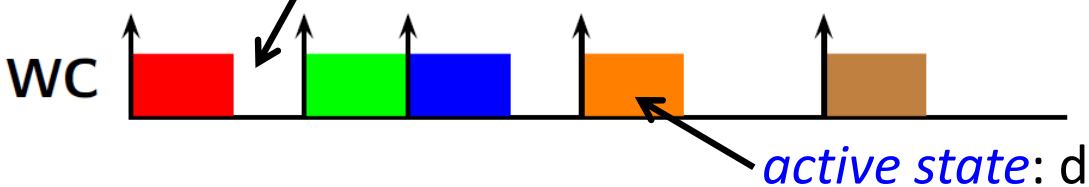
[Hennessy, Patterson: HW Interface]

Interference Mechanism: Temperature

- What is the *most critical event sequence* in terms of maximal chip temperature?
- How can we simultaneously guarantee *timing and temperature*?
- What are possible *scheduling techniques* with low overhead (no temperature sensors, simple control)?

Interference Mechanisms: Temperature

idle state: static (leakage) power

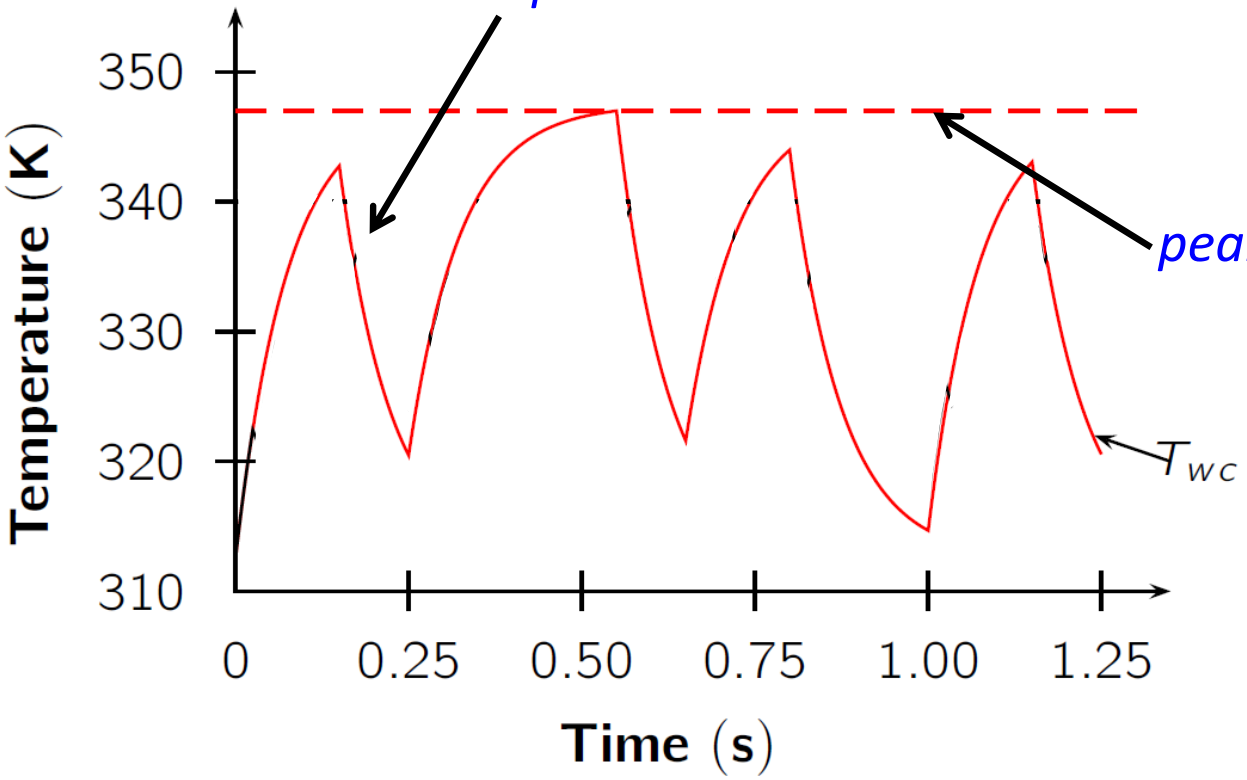


$$P = \rho T + \omega$$

active state: dynamic and static (leakage) power

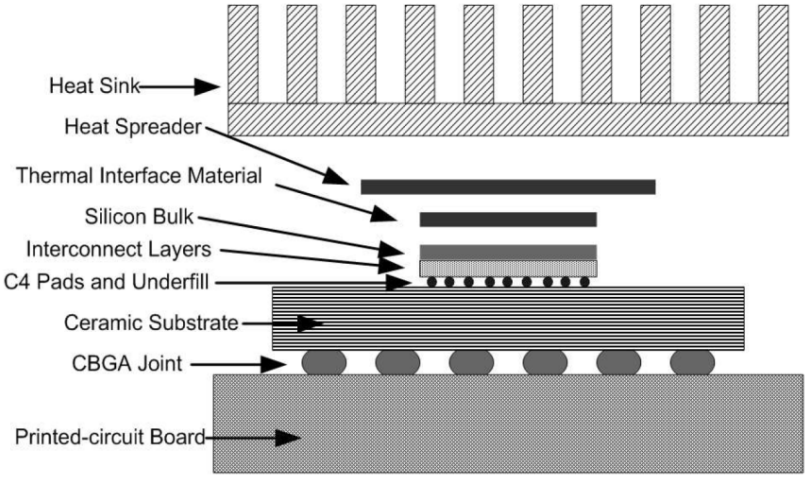
temperature decrease: based on linear thermal model

$$C\frac{dT}{dt} = -G(T - T_{amb}) + P$$

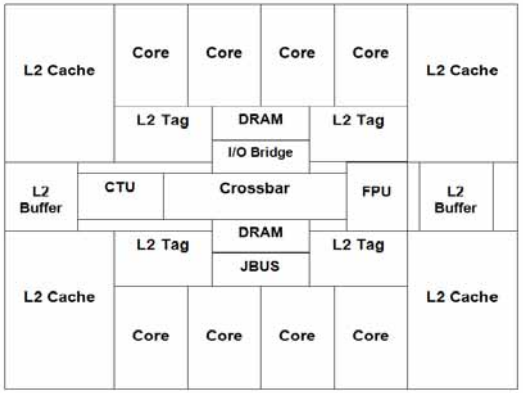


Reality is more complex …

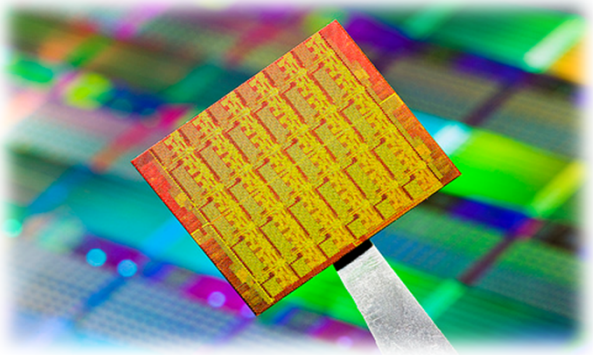
- A computer architecture is not a single power source (gates, arithmetic units, cache, interfaces, …)
- Packages are complex physical structures
- Future architectures are multi-core and multi-processors



[Wei Huang: Hotspot/PhD]



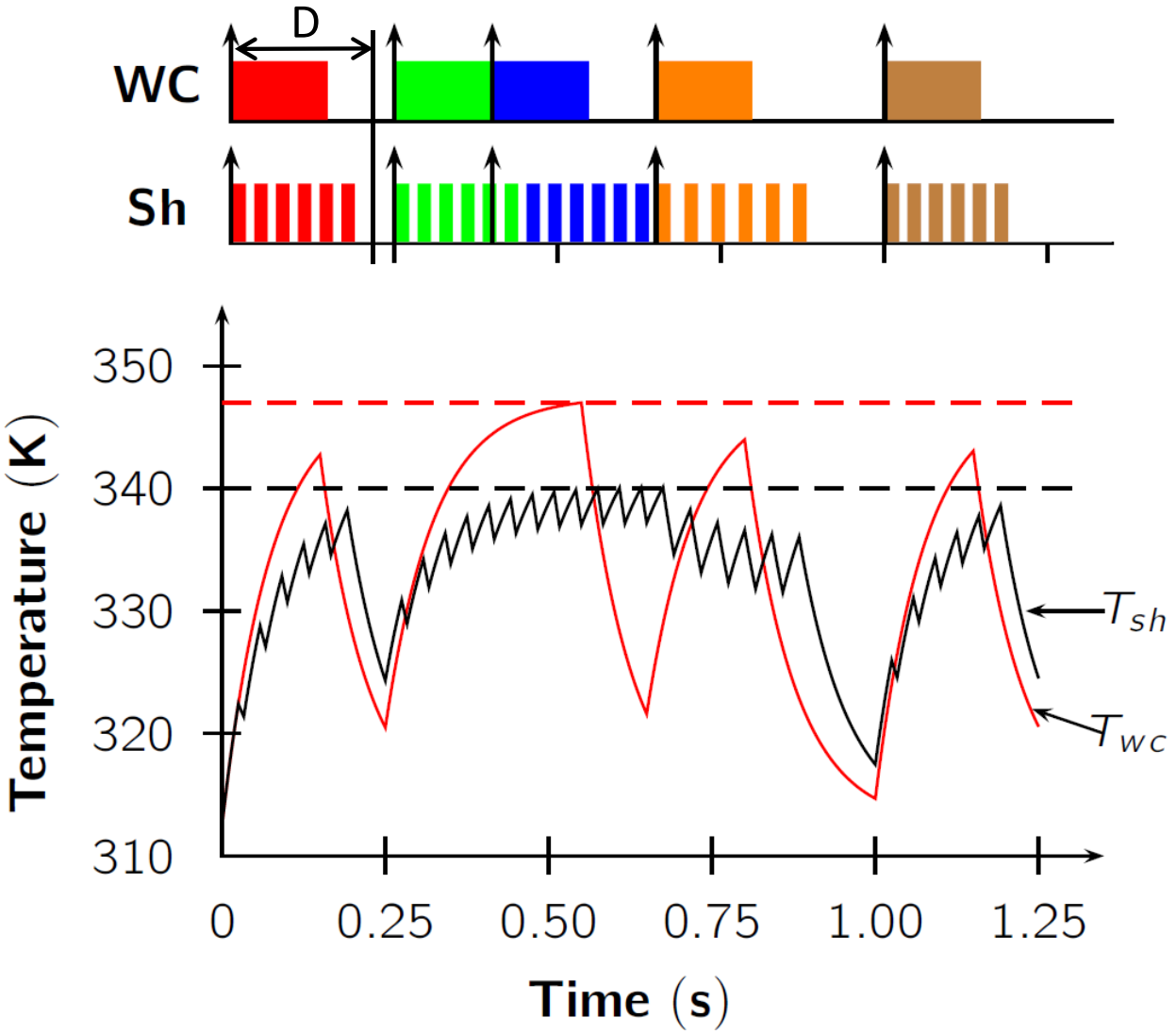
[Zanini: Niagara core]



[Intel SCC]

Example: Guarantee Timing and Temperature

deadline : to be satisfied

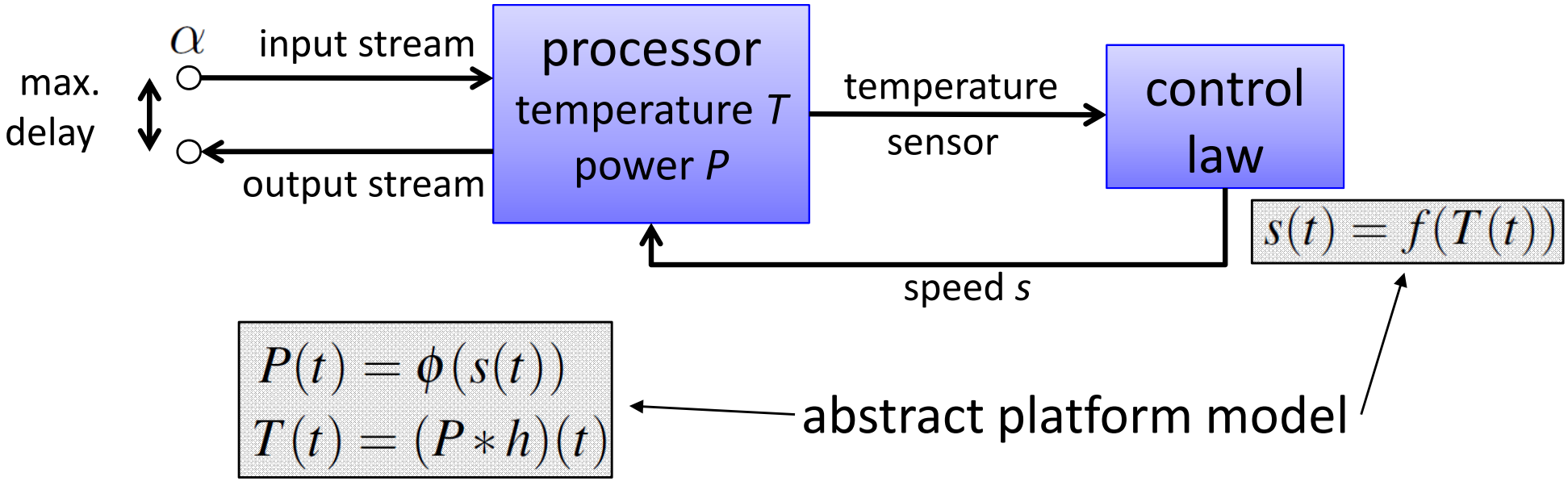


How to choose an optimal active-idle schedule?

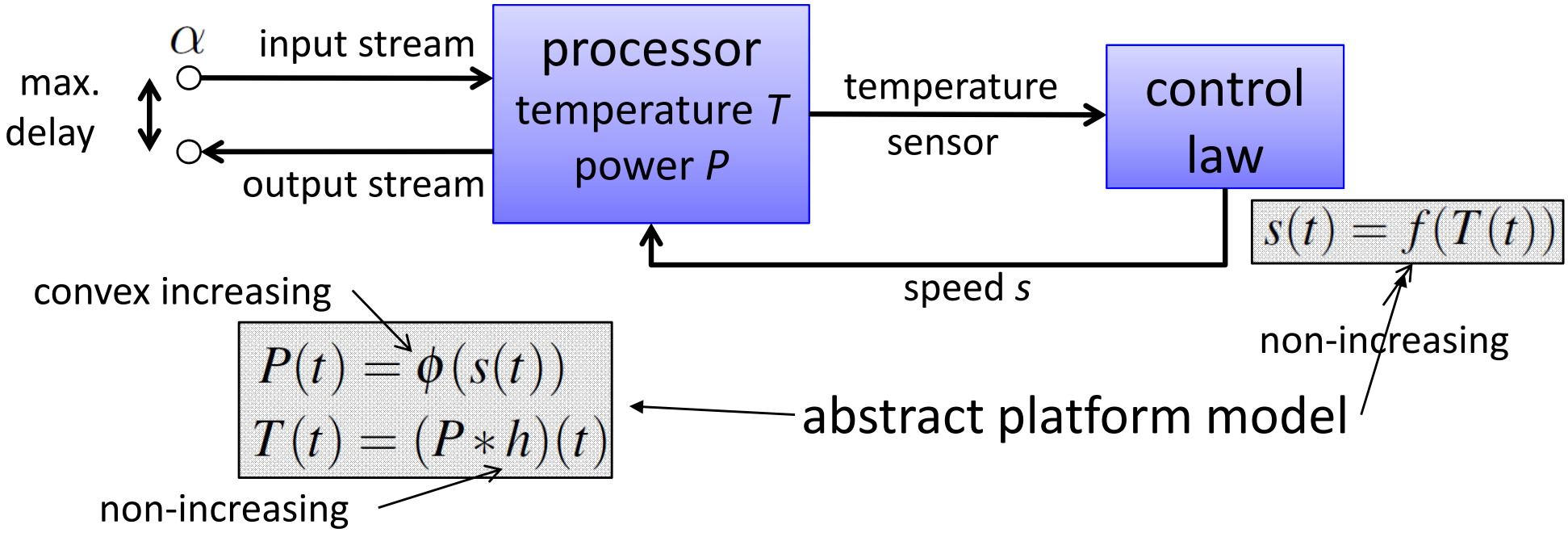
How to dynamically adapt the schedule (non-deterministic task arrival times)?

[Kumar et al. 2010-2014]

Example: Closed Loop Control



Example: Closed Loop Control



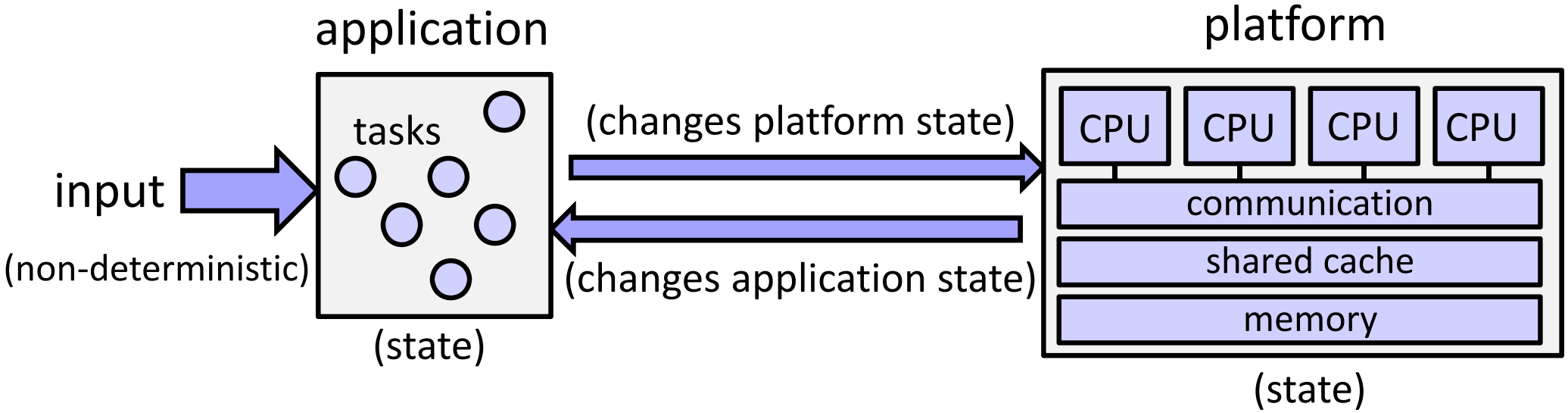
[Kumar et al. 2010-2014]

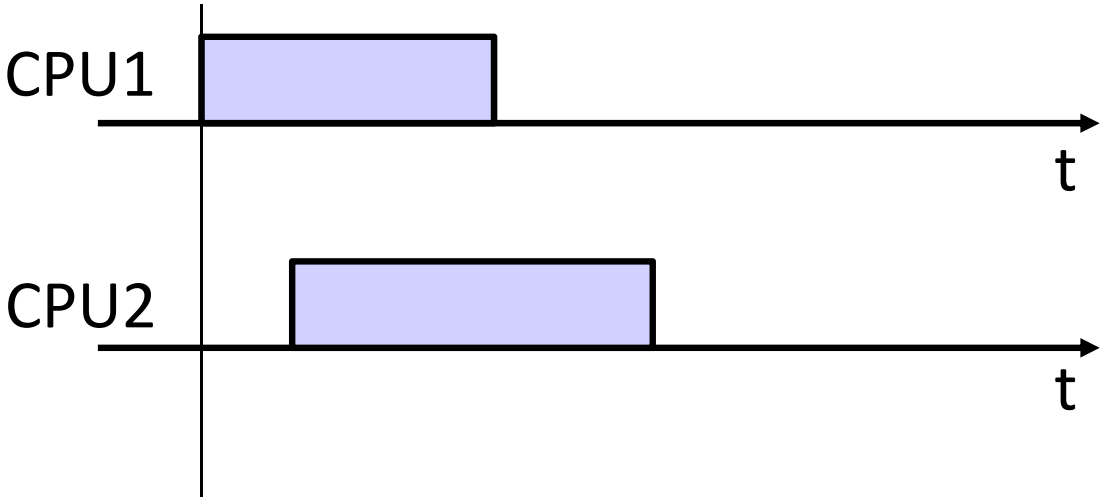
- Determine the *critical instance* w.r.t. timing and temperature based on the input arrival curve α.

Contents

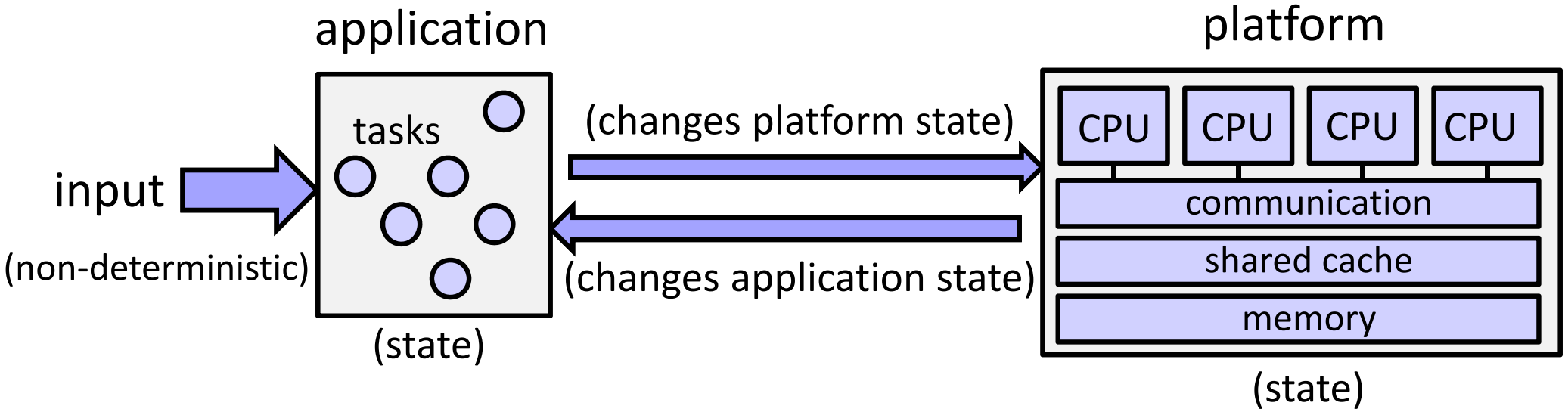
- Why?
- Interference
- ***Task Level***
  - single task on single processor
  - several tasks on single processor
  - ***single tasks on parallel processors***
- Task Graph Level
- Outlook and Summary

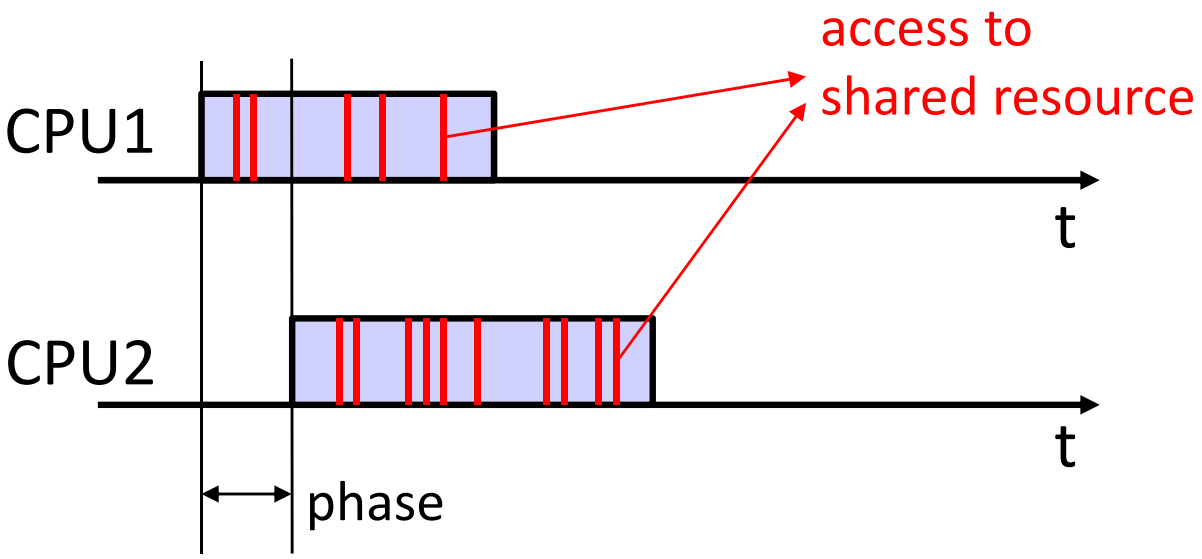
Single Tasks on Multiple Processors





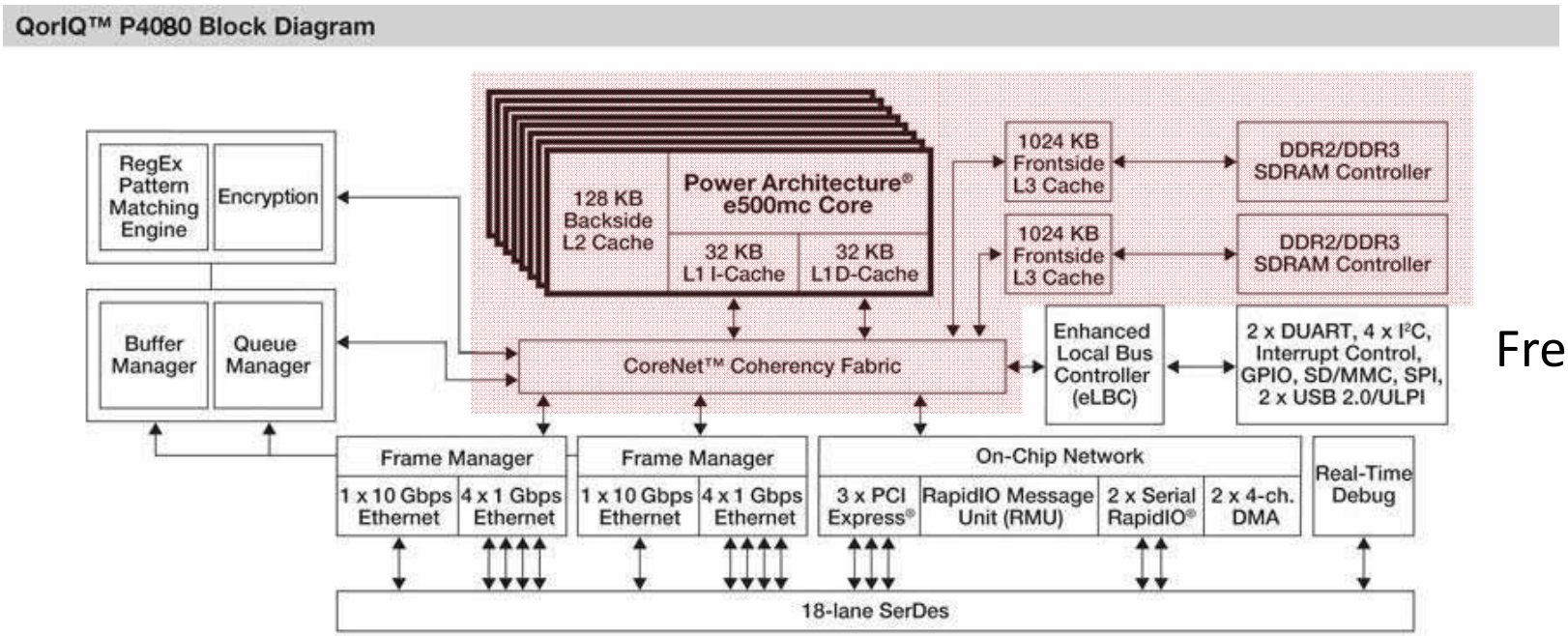
Single Tasks on Multiple Processors



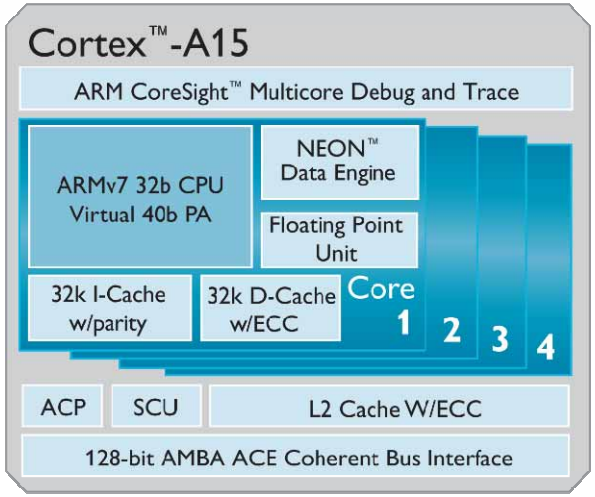


- Shared bus/network
- Shared cache
- Cache eviction
- Shared memory
- I/O
- …

Sample Architectures (1)



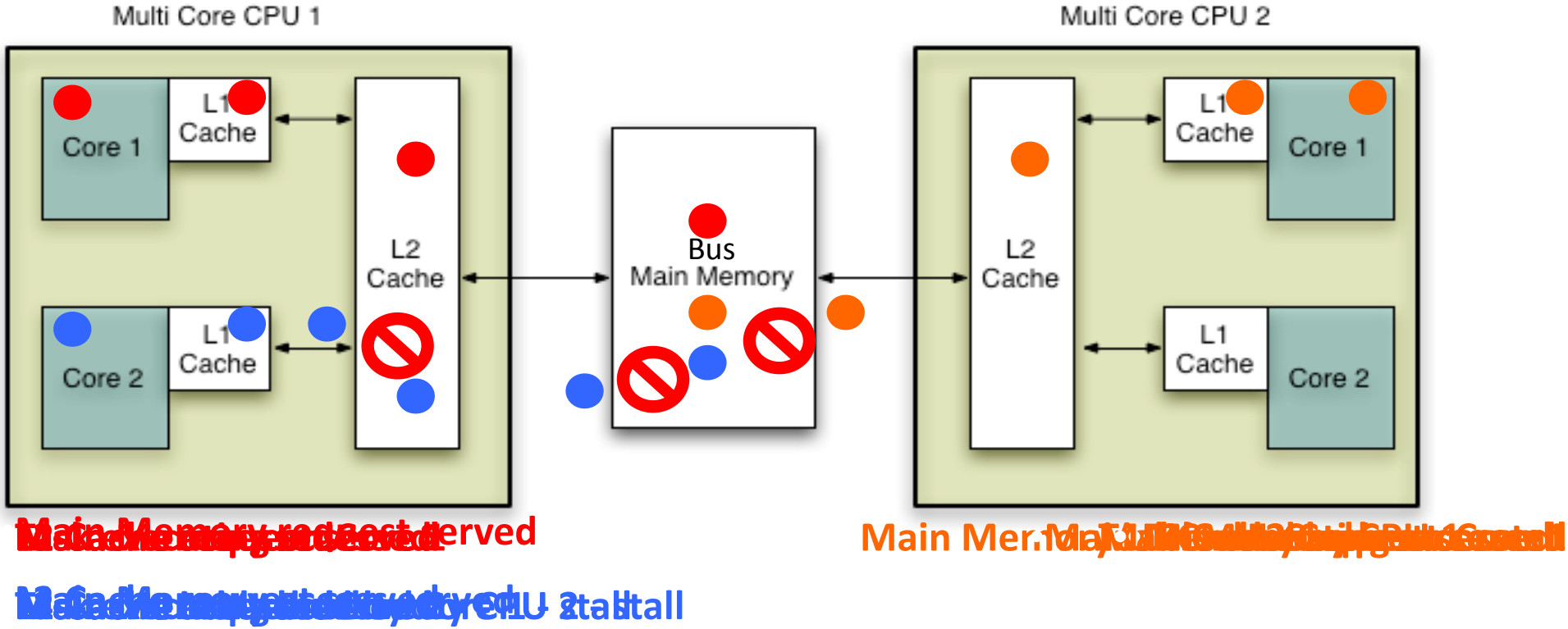
Freescale P4080



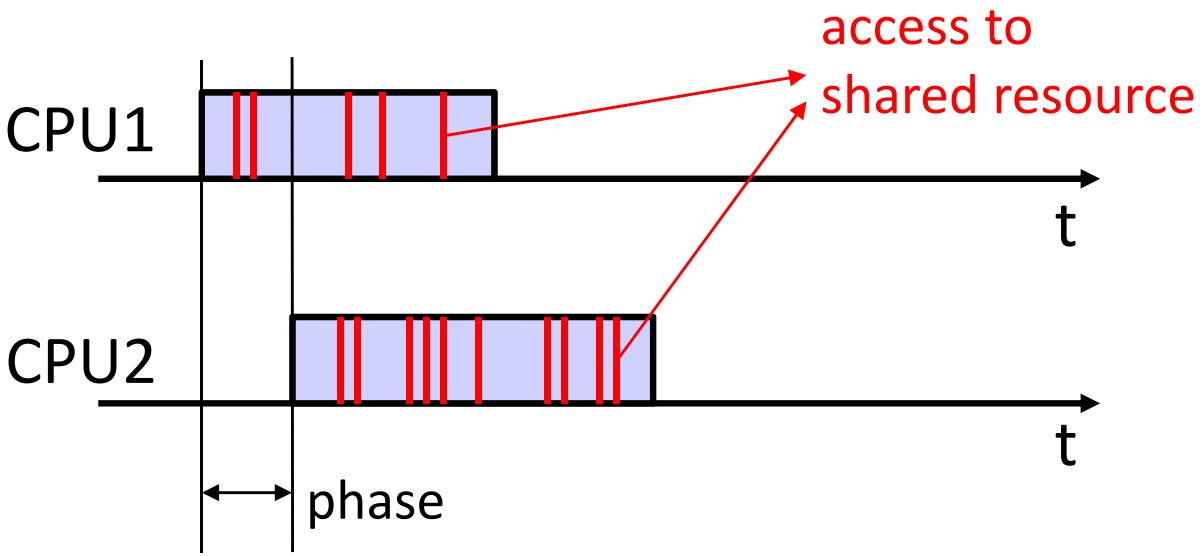
ARM/Cortex Ax

Interference

- Multicore systems use shared resources (Memory, Bus)
- Multiple entities compete for shared resources
  - waiting for other entities to release the resource
  - access the resource



Challenges



- High penalty:
  - eviction on shared caches
  - bus/network arbitration
- High uncertainty:
  - phase between cores [synchronization, dynamic scheduling]
  - access pattern to shared resource [program paths, microarchitecture]
  - accessed memory locations [program state]

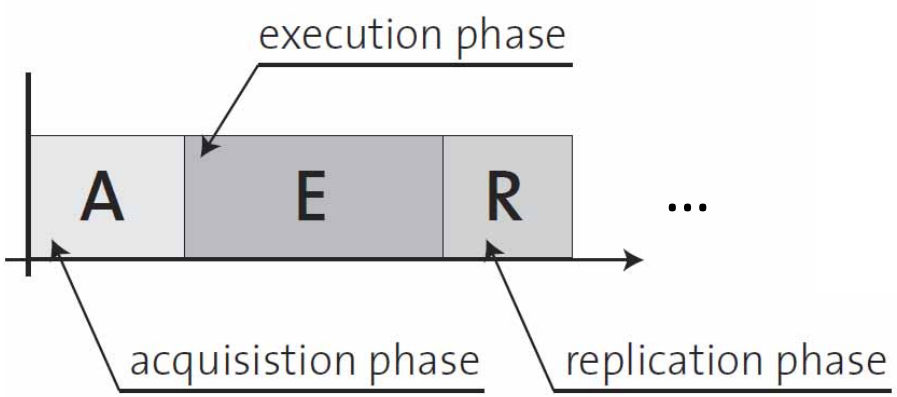
Reducing Predictability and Analysis Gaps

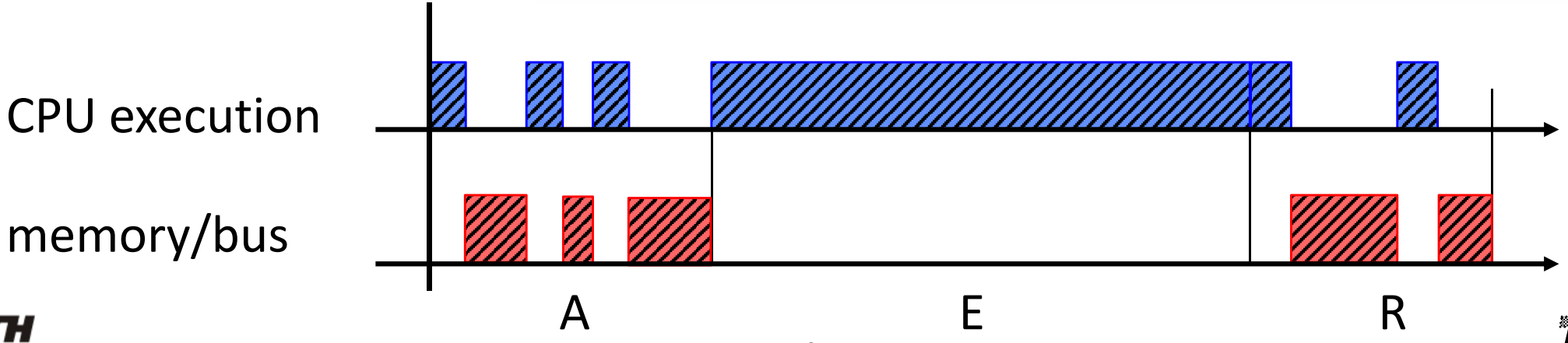
Task model and resource-aware scheduling

Isolation in time and space

Task / Superblock Model (1)

- Tasks are structured as a sequences of phases:
  - a task may contain many phases
  - model motivated by current practice in automotive and avionics





Task / Superblock Model (2)

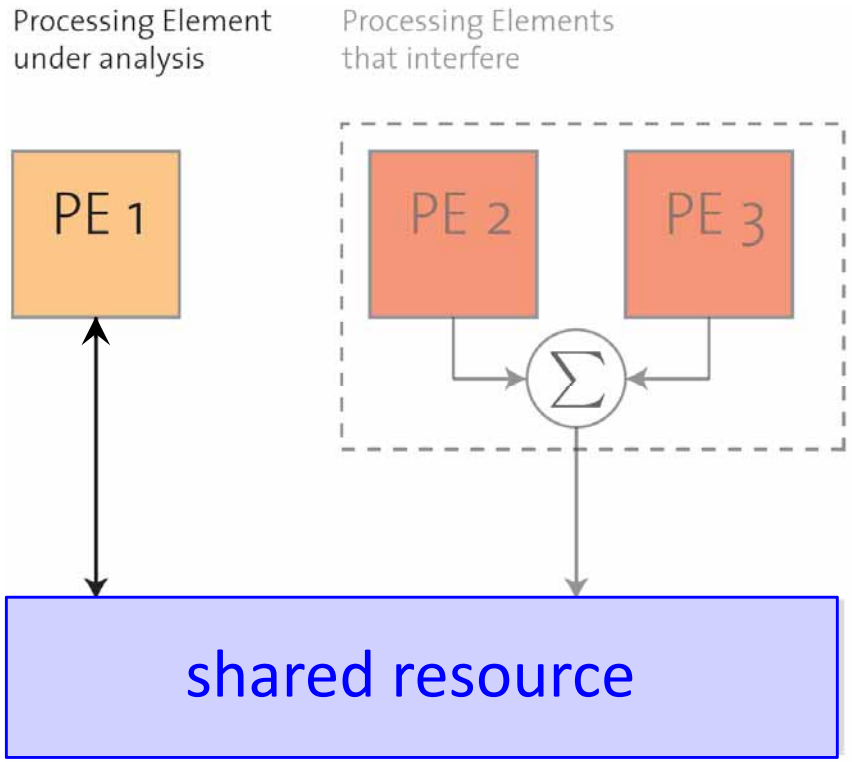
- *Advantages:*
  - Decreasing non-determinism → reduced predictability gap
  - Increased analysis accuracy → reduced analysis gap
  - Has been combined with various task and bus scheduling methods

    [G. Yao, R. Pellizzoni, S. Bak, E. Betti, and M. Caccamo: RTS Journal 2012; A. Schranzhofer et al.: 2009-2011]
  - Analysis methods based on classical real-time analysis, real-time calculus and timed automata.

- *Disadvantages:*
  - Code refactoring necessary
  - Memory access analysis necessary

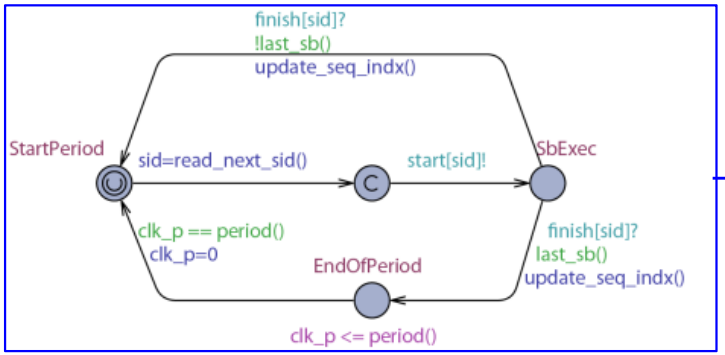
Dynamic Arbitration: Mixing Abstractions

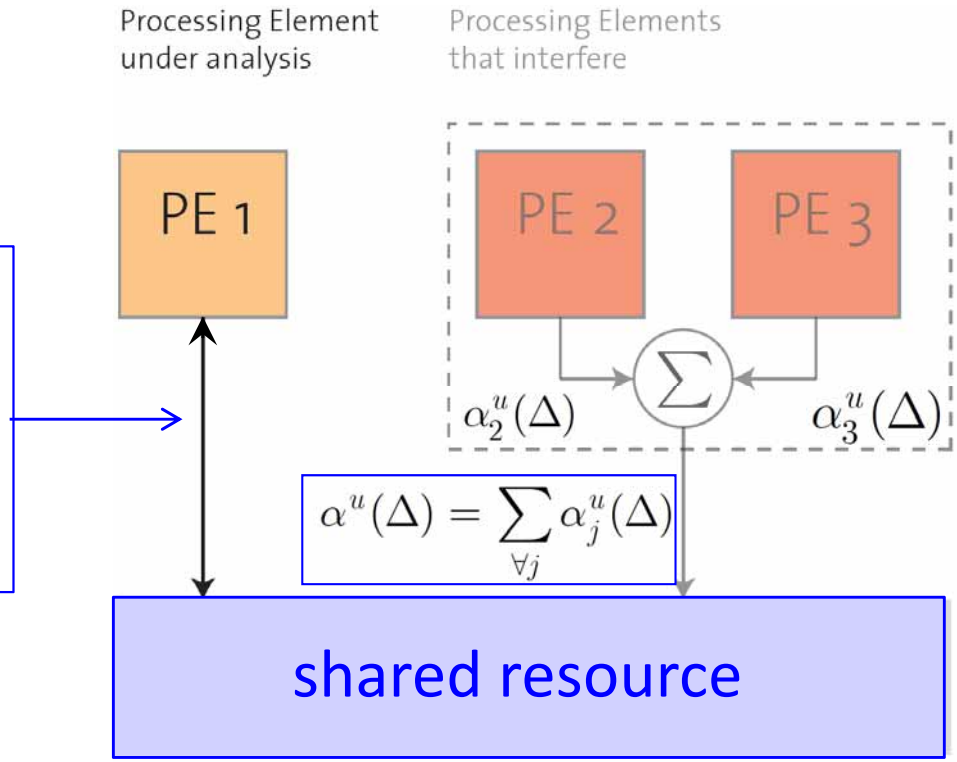
[Giannopolou et al. : EMSOFT 2012]



Dynamic Arbitration: Mixing Abstractions

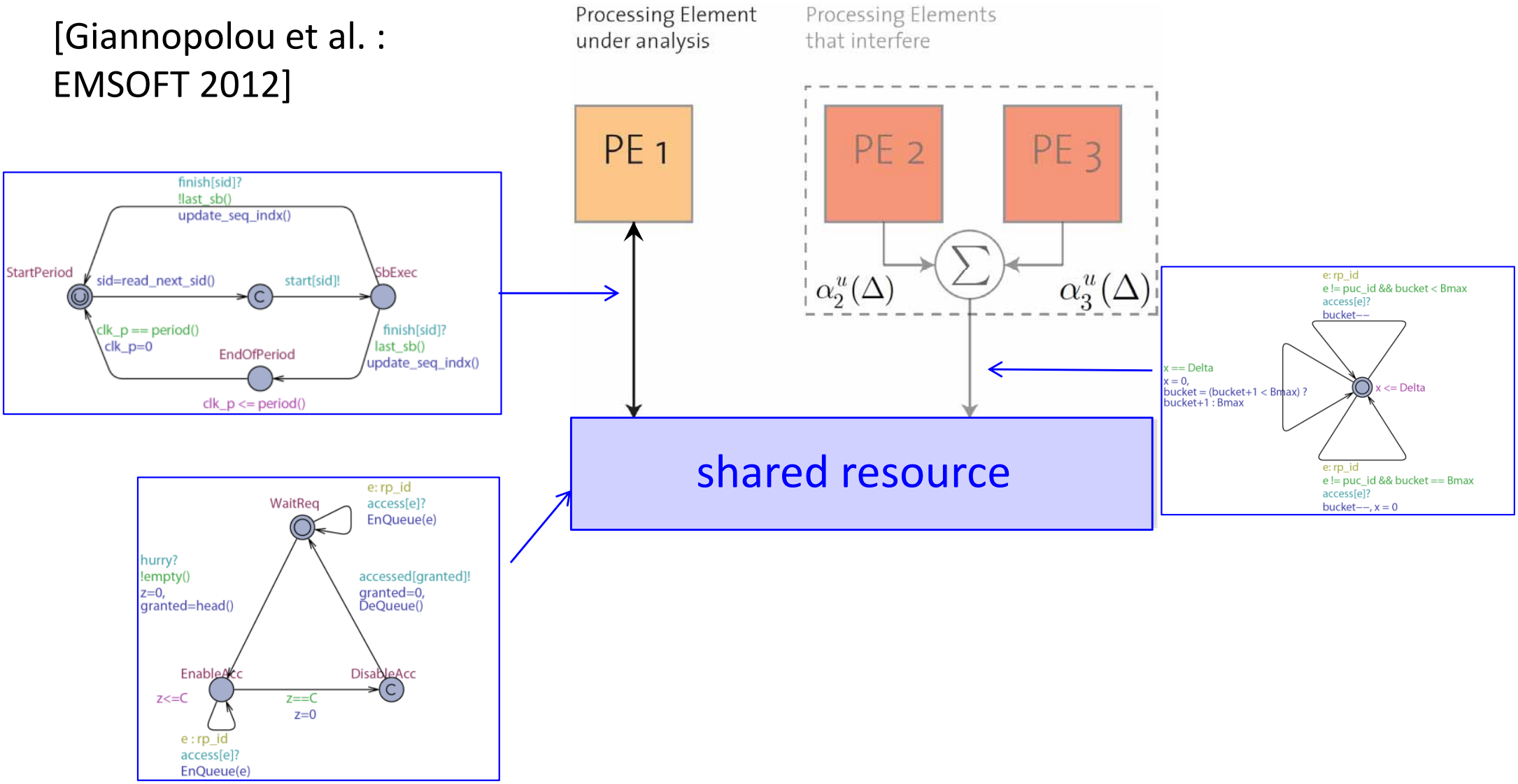
[Giannopolou et al. : EMSOFT 2012]





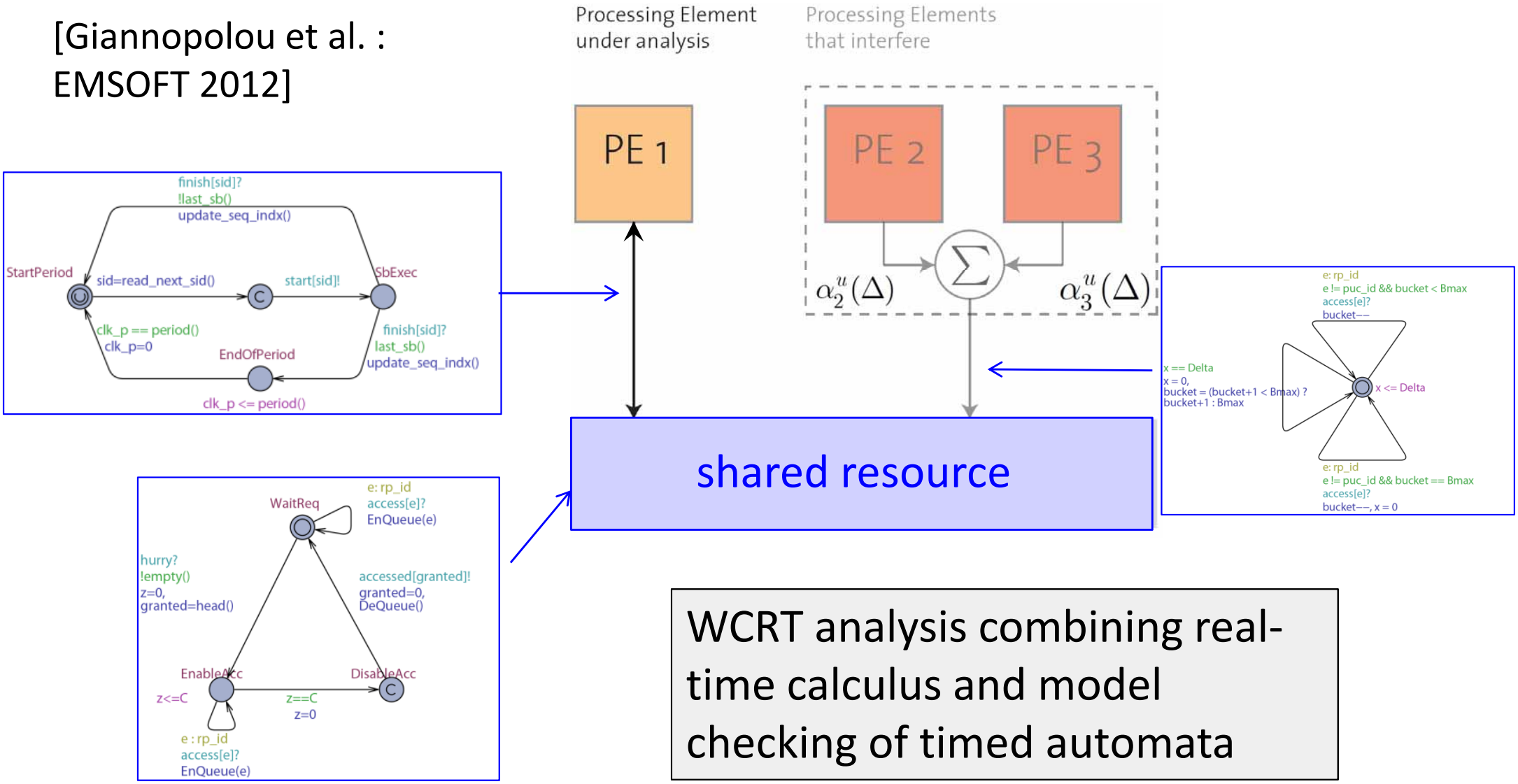
Dynamic Arbitration: Mixing Abstractions

[Giannopolou et al. : EMSOFT 2012]



Dynamic Arbitration: Mixing Abstractions

[Giannopolou et al. : EMSOFT 2012]



WCRT analysis combining real-time calculus and model checking of timed automata

Dynamic Arbitration: Mixing Abstractions

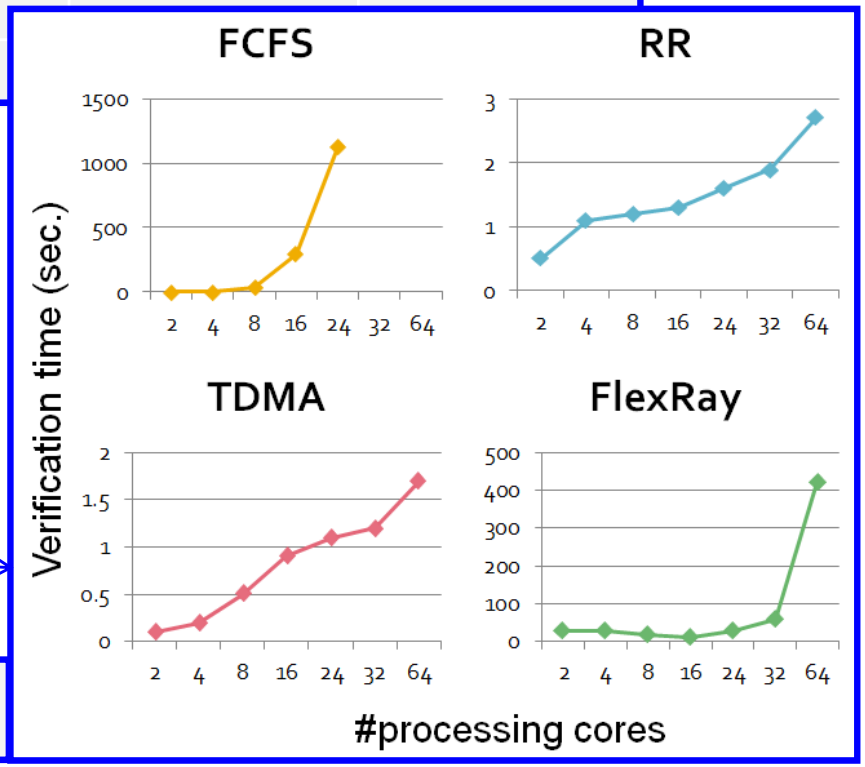
accuracy

Benchmark	Improvement w.r.t. analytic approach (FCFS)	Improvement w.r.t. analytic approach (RR)	Avg. verification time (FCFS)	Avg. verification time (RR)
canrdr01	21.6%	21.6%		
cacheb01	0.6%	0.8%		
tblook01	26.8%	26.8%	102.4 sec	58.7 sec
a2time01	11.3%	11.3%		
rspeed01	0.6%	0.8%		

6 benchmarks from EEMBC 1.1 benchmark suite (automotive) on 5 cores

selected application from automotive industry

scalability



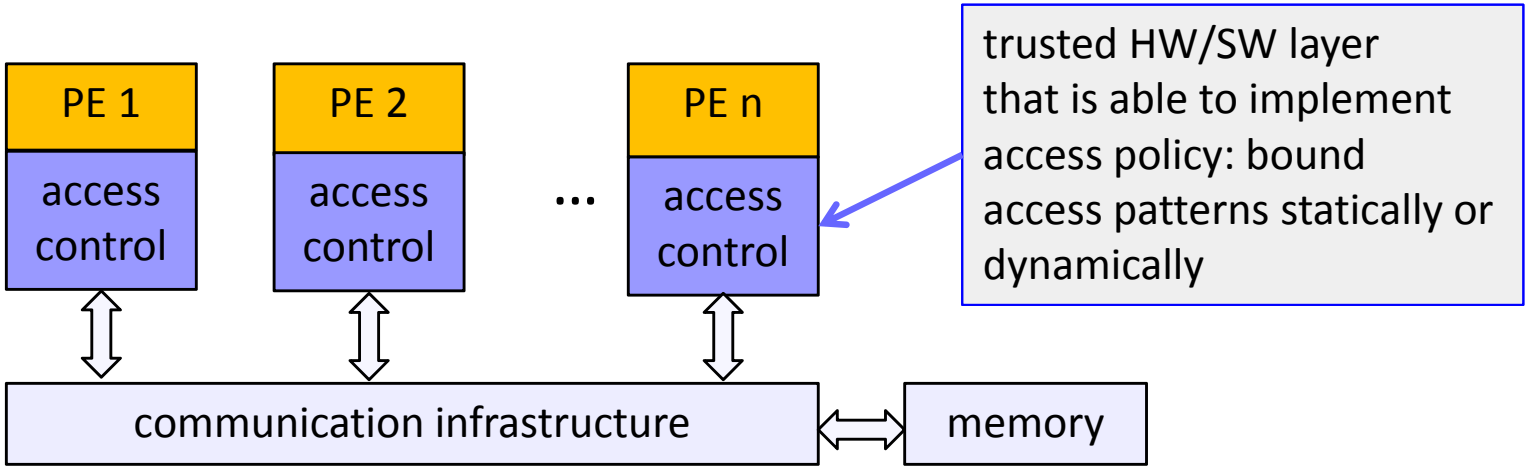
Reducing Predictability and Analysis Gaps

Task model and resource-aware scheduling

Isolation in time and space

Isolation in Time: Access Control

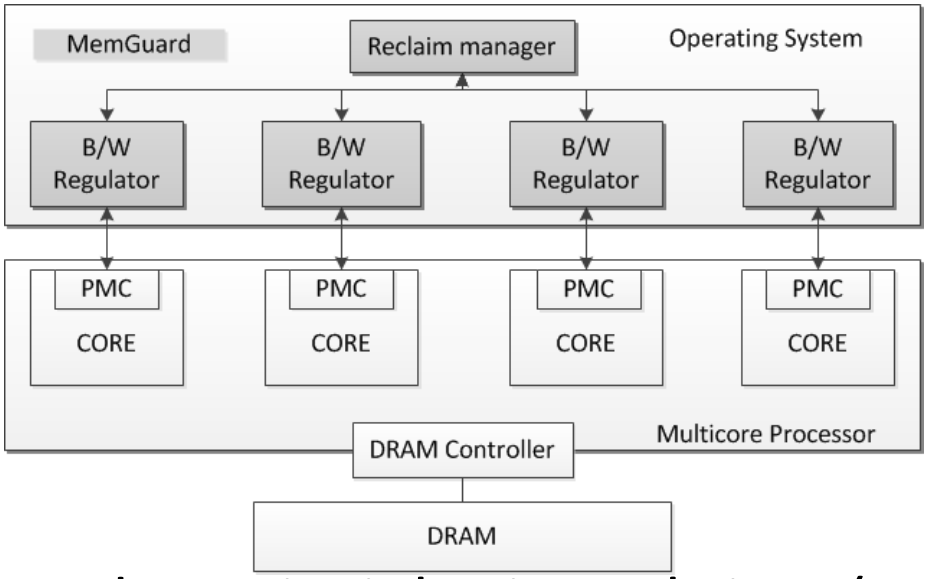
- ***Basic Ideas***
  - communication scheduling, e.g. [Diemer NOCS 2010], TMDA, ….
  - limit interference by providing guarantees on the access pattern of cores and/or tasks

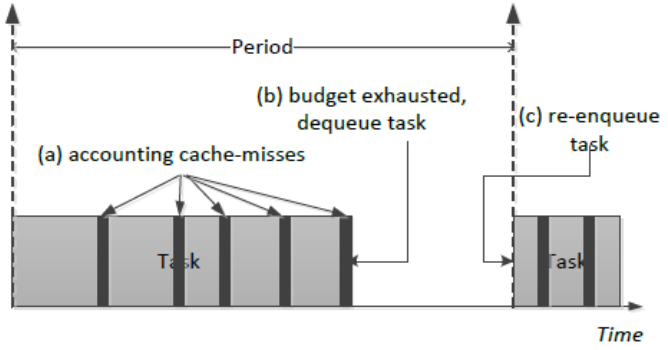


  - *hardware:* clock modulation [A. Herdrich et al. ICS 2009], memory controller [B. Akeson et al. CODES/ISSS 2007, RTCSA 2008]
  - *software:* influence OS scheduling [F. Bellosa SOSP 1997], [H. Yun et al. ECRTS 2012, RTAS 2013]

Isolation in Time: Access Control

- *Example*: Software-based approach [H. Yun et al. ECRTS 2012, RTAS 2013]
  - monitor memory traffic (L1 cache misses), dequeue tasks if (static or dynamic) budget is exceeded:

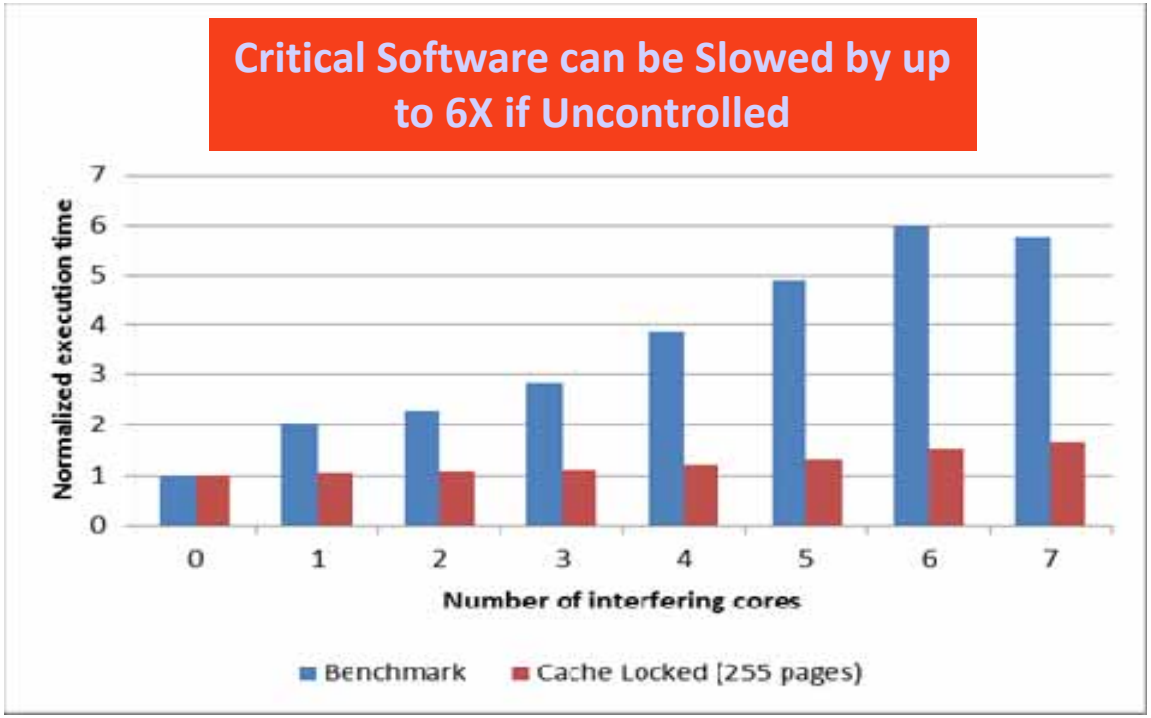




- Related to principles in real-time (communication) systems:
  - traffic shaping in communication systems
  - real-time servers (CBS, DS, DBS) which integrate shaping and scheduling

Does it Matter?

- For example, LM Space Systems tests indicated that competing with only one other core can mean a 6-fold increase in execution time (blue bars).
- Combination of isolation in time and space can control increases in measured worst case execution times (red).



Source: Lockheed Space Systems HWIL Testbed

Isolation in Space: Scratchpads and Caches

- *Private Scratchpad:*
  - Memory access is under software / compiler control
- *Private cache:*
  - Often not the case: majority of multicore platforms have shared cache
- *Cache partitioning:*
  - Way-partitioning:
    - Easy to apply, but inflexible
    - Reducing number of ways per core can greatly increase cache conflicts
  - Colored Lockdown [Mancuso et al.: RTAS 2013]:
    - the OS decides the physical memory mapping via virtual memory pages
    - use cache locking instructions to lock “hot” pages of real-time critical task

Predictable Hardware Architectures

- There are task interferences between
  - temporal partitions on a single core (e.g. cache, pipeline, dynamic execution, MMU, execution units) and
  - spatial partitions on several cores (e.g. shared resources, communication and interconnect, main memory, caches, I/O)

- *Goal:* provide hardware mechanisms to reduce or bound inter-task interference in order to
  - improve timing composability
  - enable accurate and efficient WCRT analysis

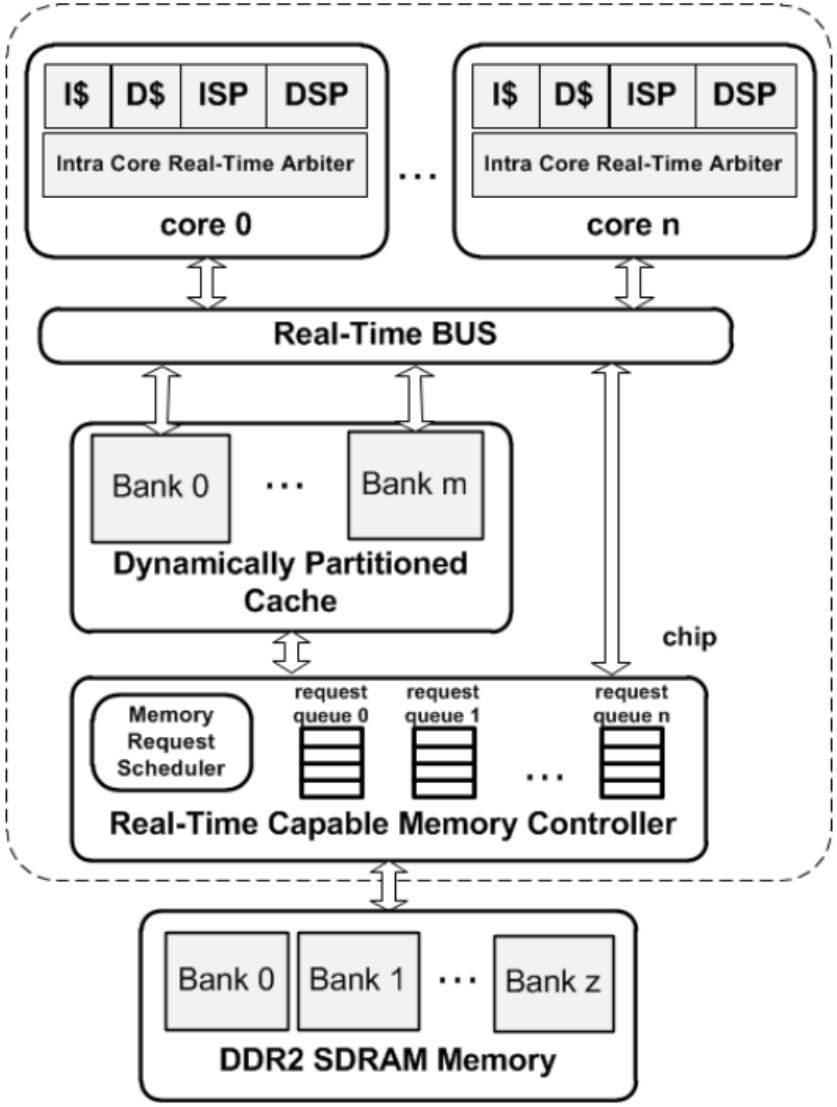
Predictable Hardware Architectures

Typical mechanisms
[e.g. Merasa IEEE Micro 2010]:

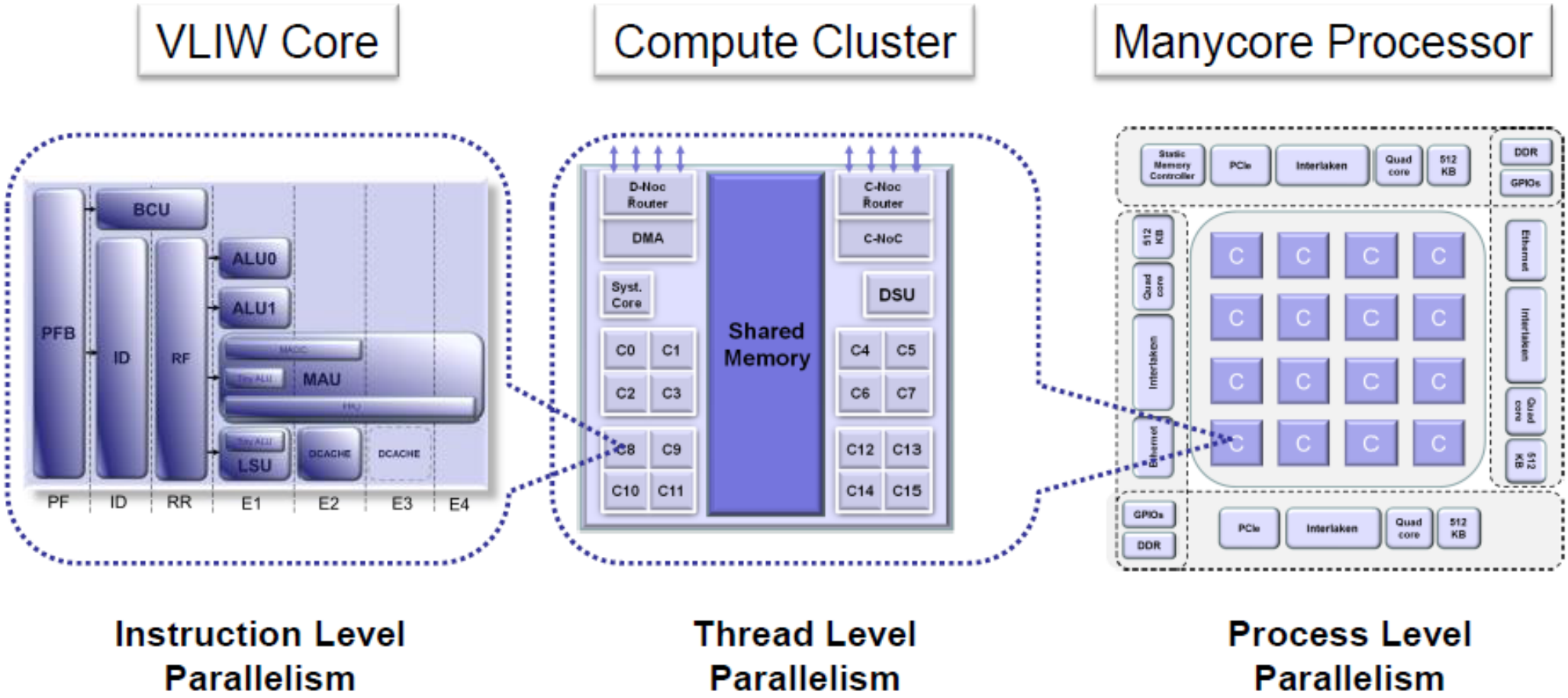
- reduce non-determinism in single cores (in-order piplines)
- software and hardware-managed scratchpad memories
- split-phase memory access
- real-time bus, cross-bar architectures
- partitioned memory architecture, use of memory banks

Other approaches:

- (Flex)PRET [EA Lee et al.: 2008-2014]



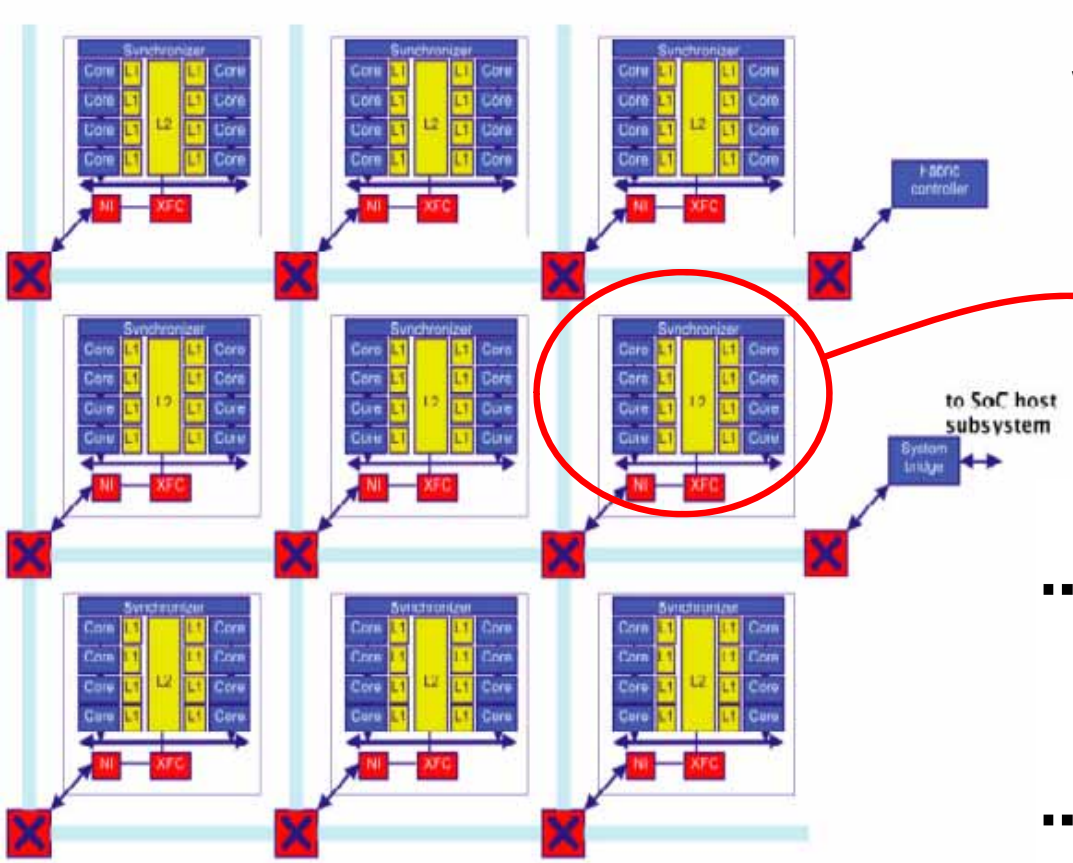
Sample Architectures (2)

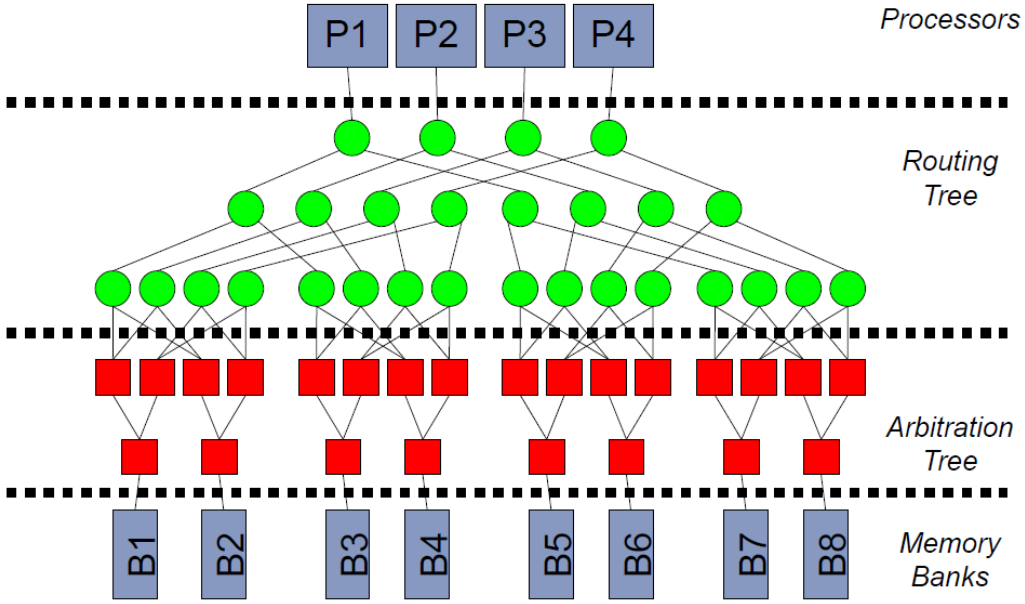


Kalray, RTAS 2014

Sample Architectures (3)

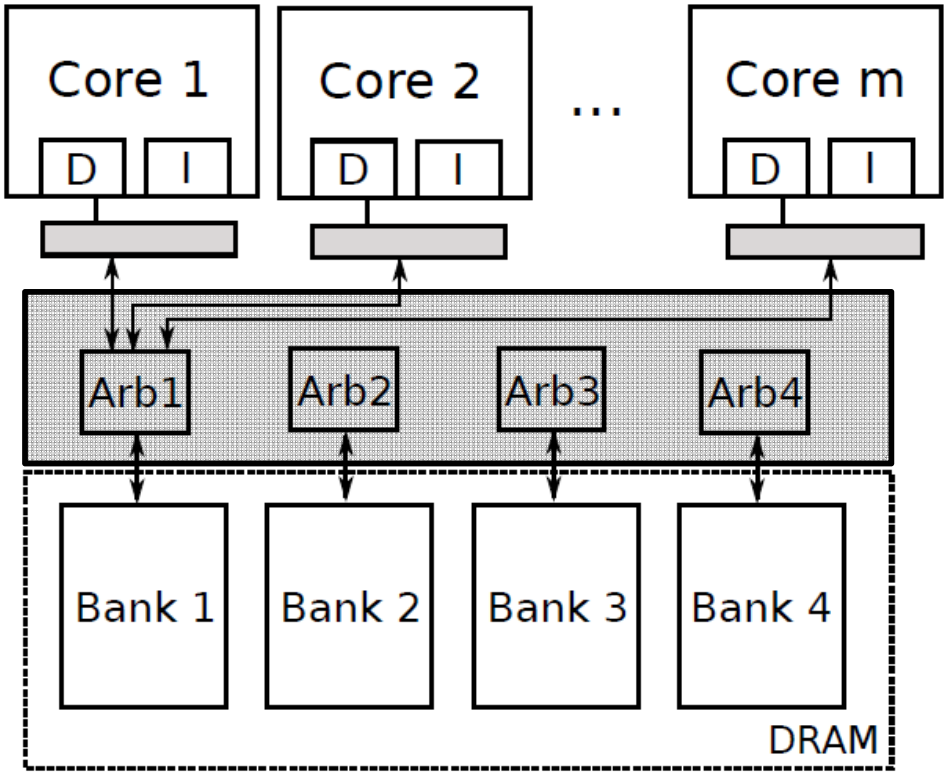
P2012 – STHorm (ST Microelectronics)
www.vampa.epfl.ch





© Luca Benini
Artist Summer School 2012

Platform Model



cores have access to private cache (or none)
cores do not show timing anomalies
(block until memory access granted)

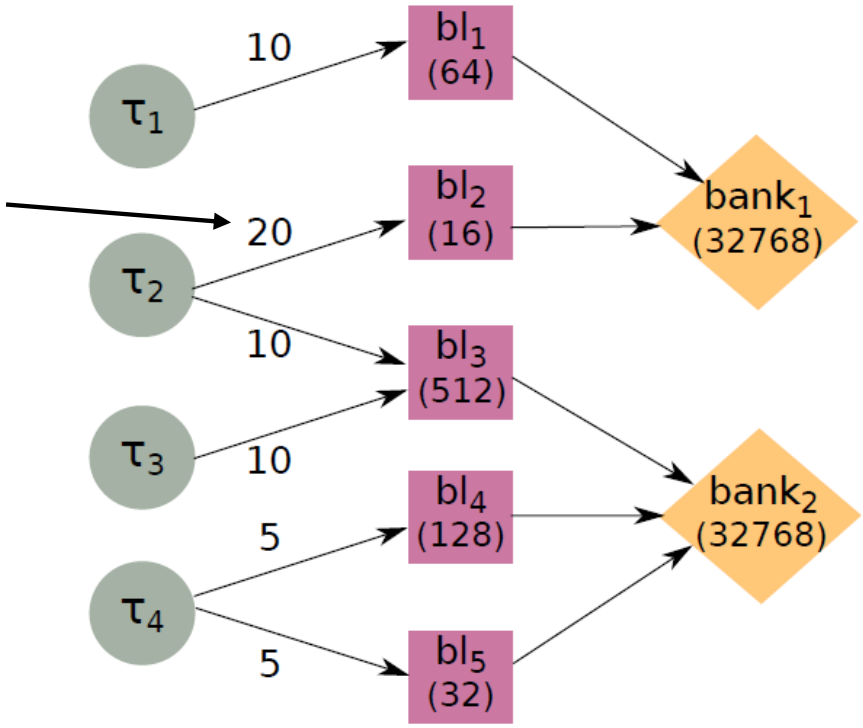
crossbar between caches and banks
with various arbitration policies
(FCFS, RR, FP, TDMA, …)

This platform models fits well current embedded multicore architectures like MPPA from Kalray, STHorm from ST

Refined System Model - Shared Memory

- Model partitioned memory access
  - Memory interference graph

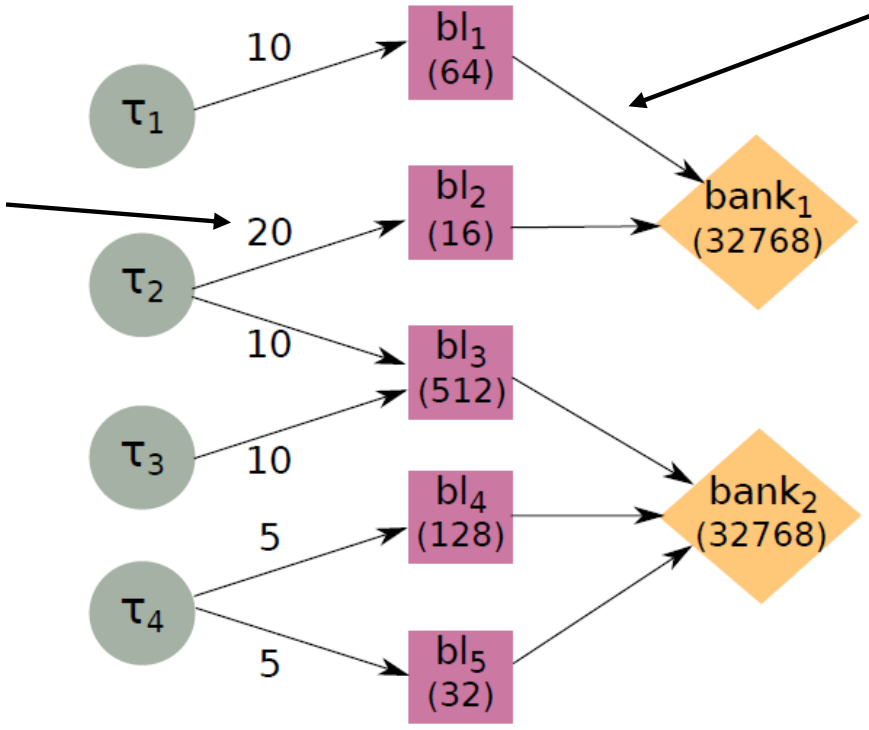
determined through task-level memory analysis



Refined System Model - Shared Memory

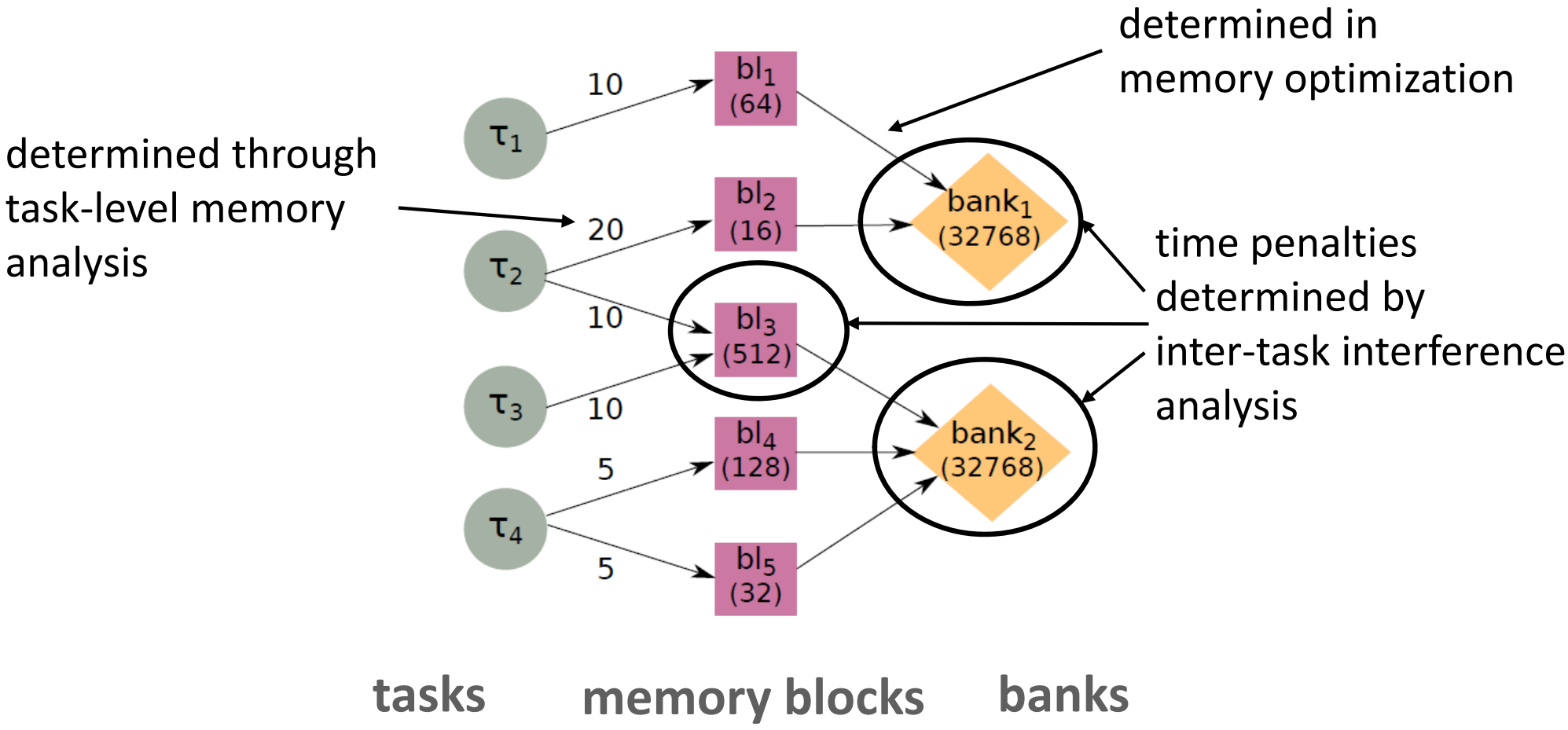
- Model partitioned memory access
  - Memory interference graph

determined through task-level memory analysis



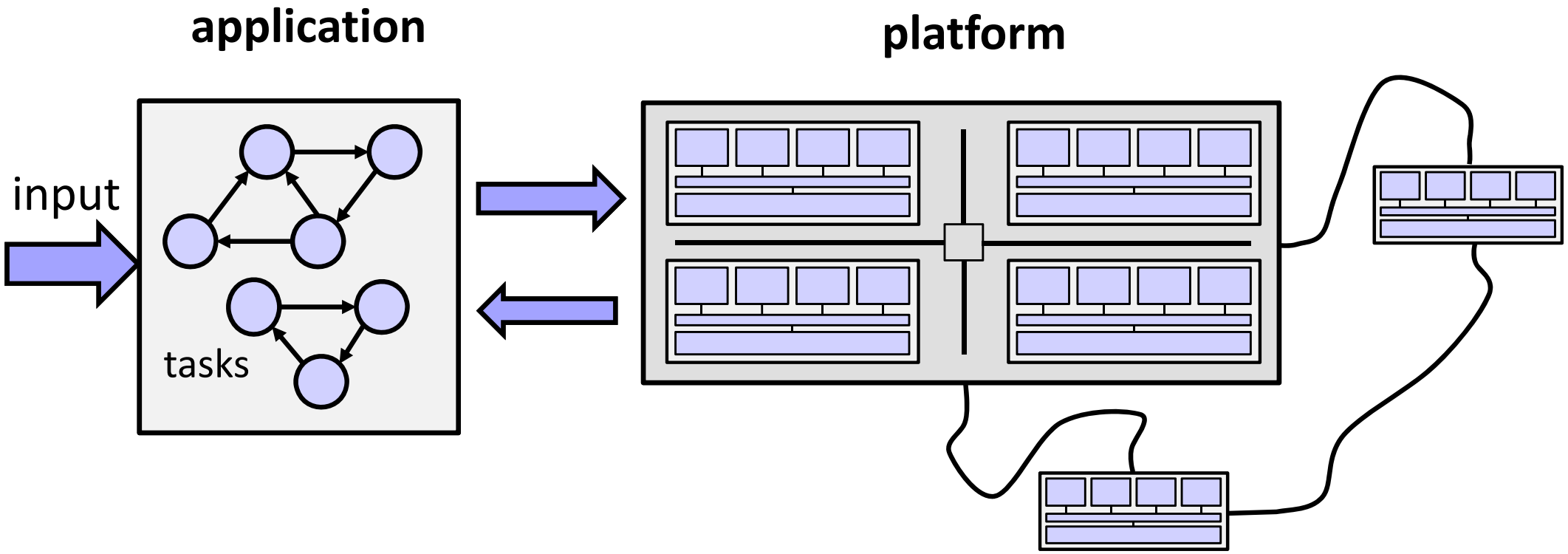
Refined System Model - Shared Memory

- Model partitioned memory access
  - Memory interference graph

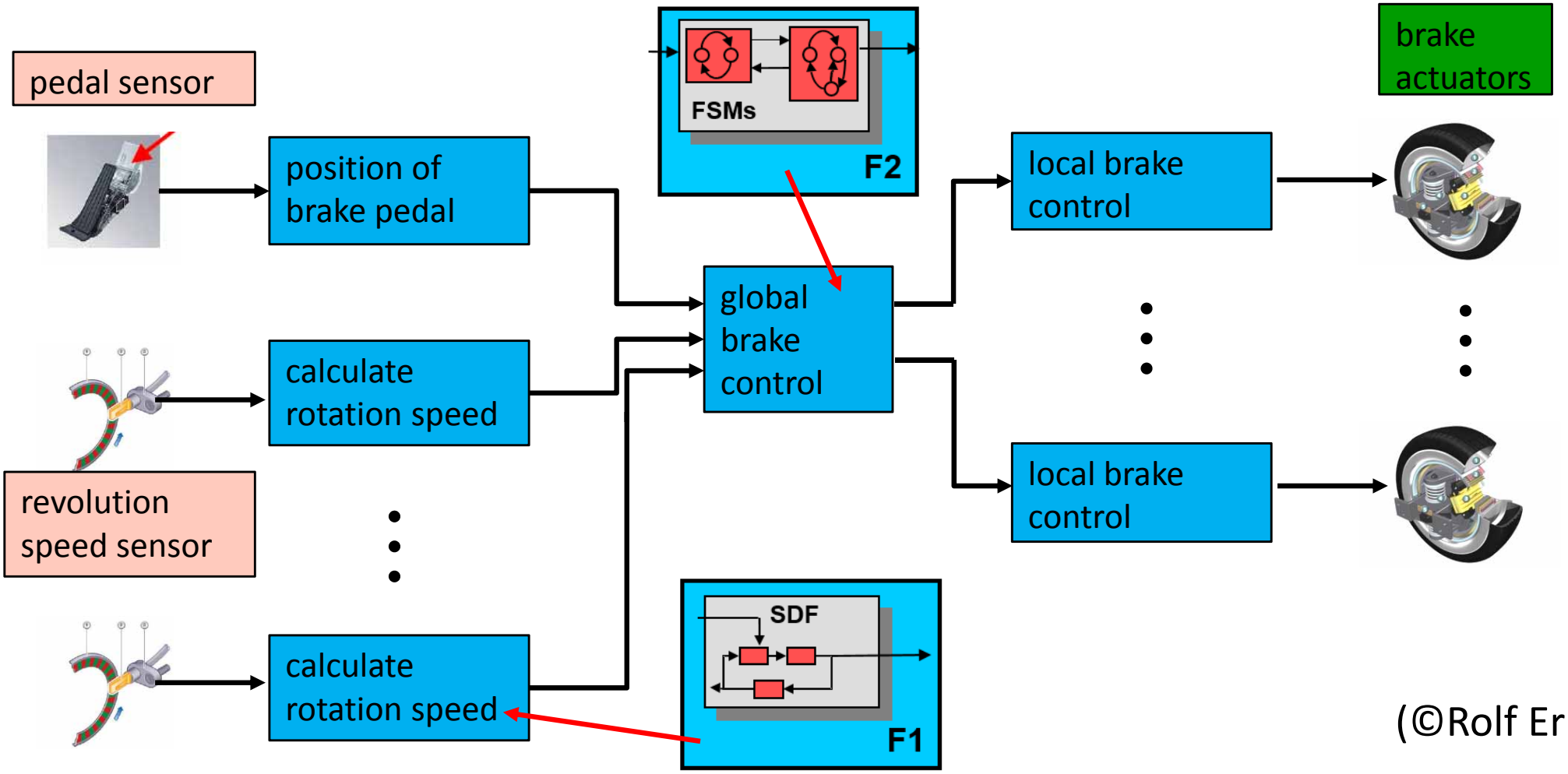


Contents

- Why?
- Interference
- Task Level
  - single task on single processor
  - several tasks on single processor
  - single tasks on parallel processors
- ***Task Graph Level***
- Outlook and Summary

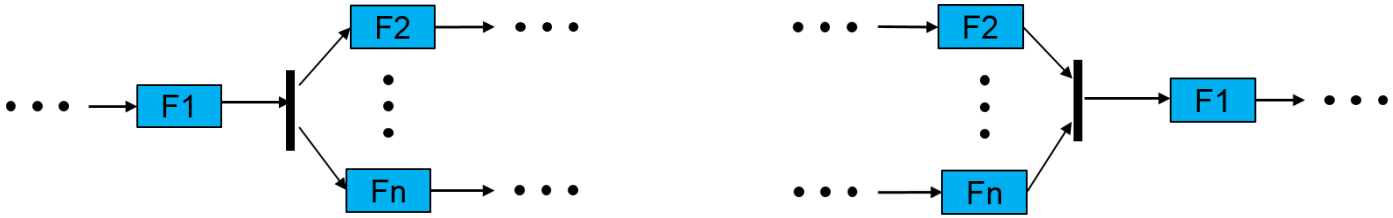

application
platform
input
tasks

Example Automotive



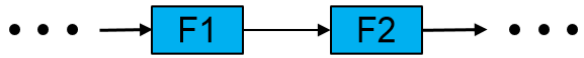
Interaction Mechanisms between Functions

- *Synchronization and Communication Mechanisms*
  - Fork/Join [parallel] and Branch/Merge [sequential]



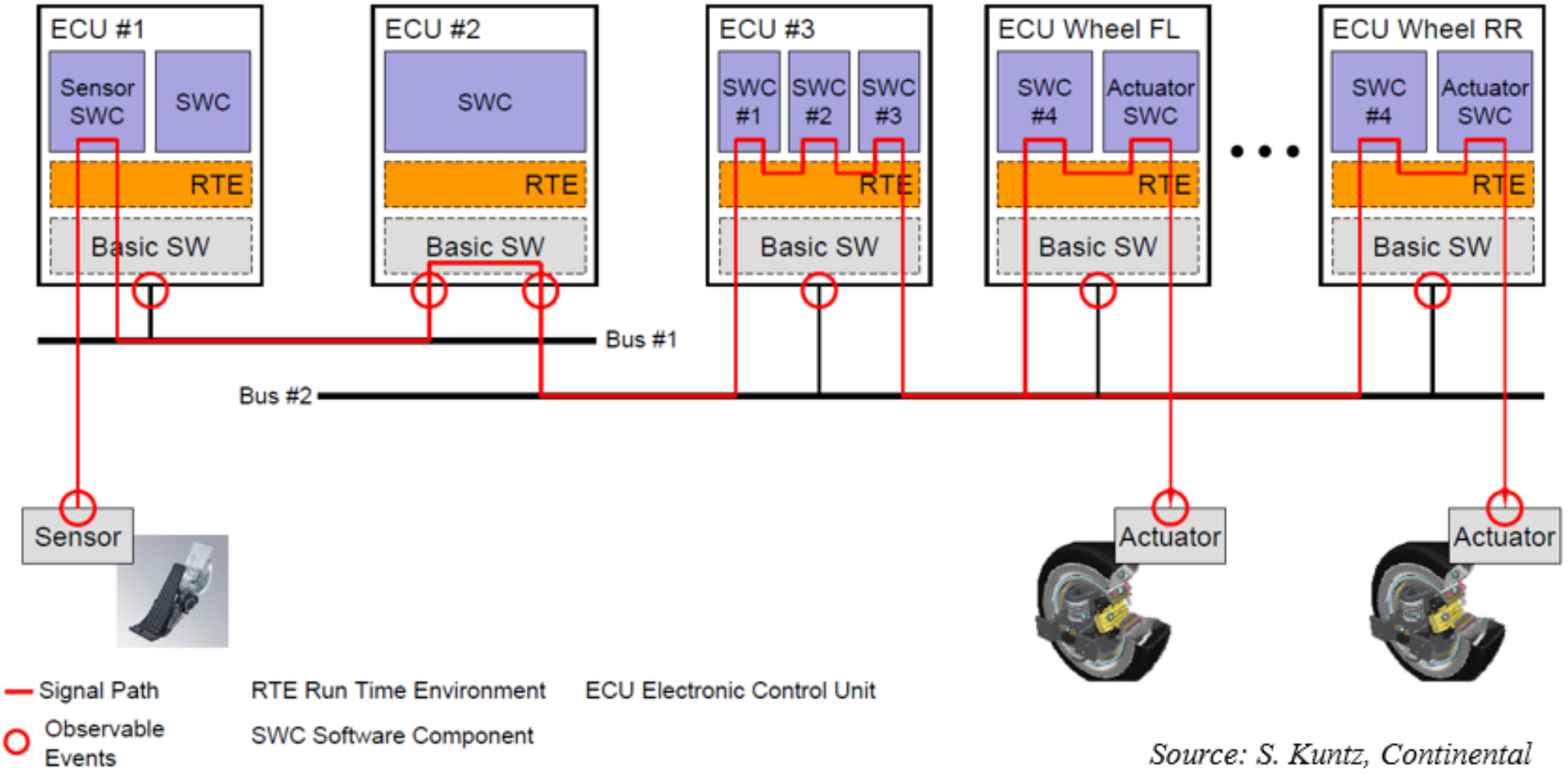
  - Publish/Subscribe and Remote Procedure Call

  - Function chain



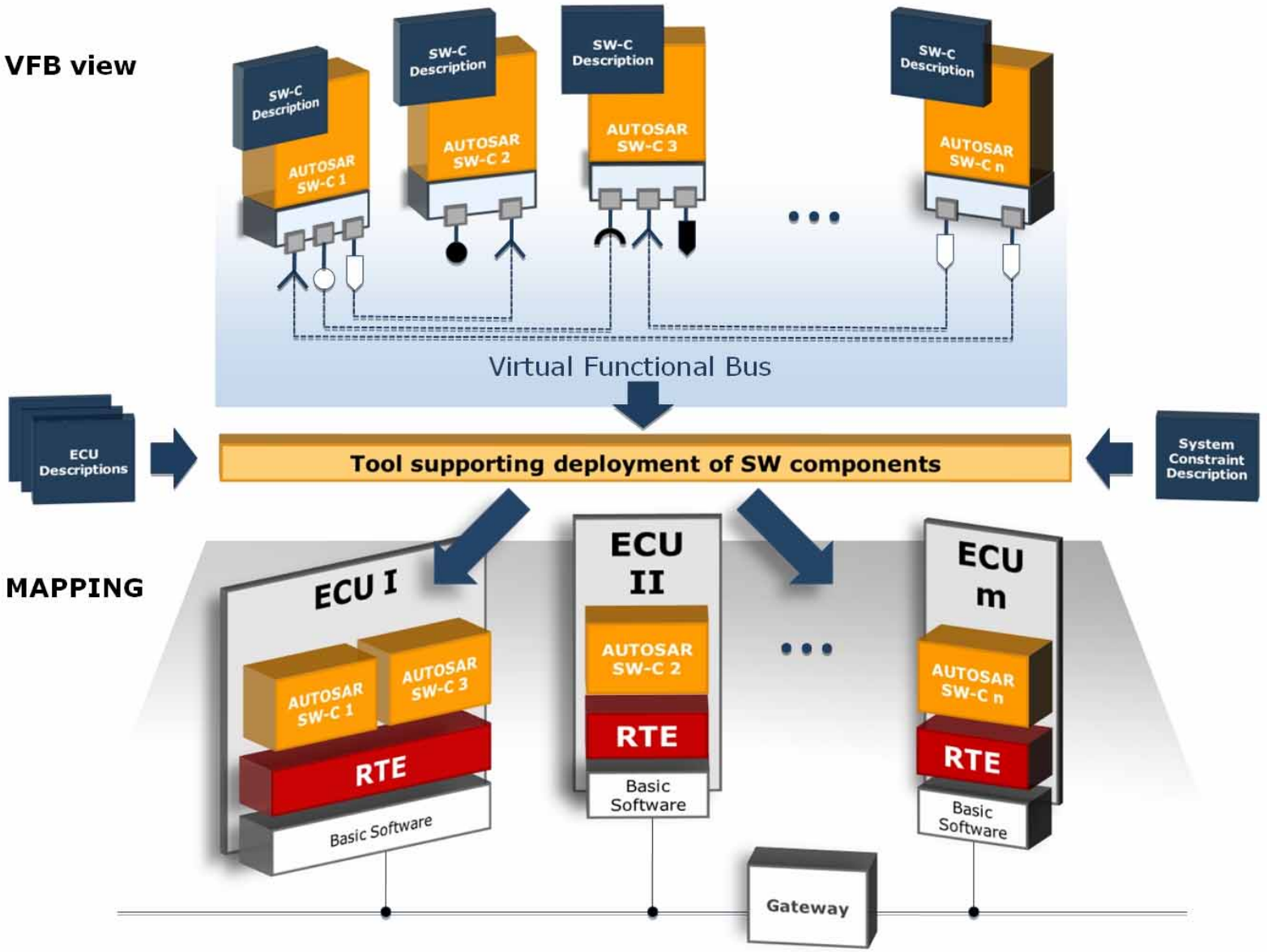
(©Rolf Ernst)

Example Distributed Architecture



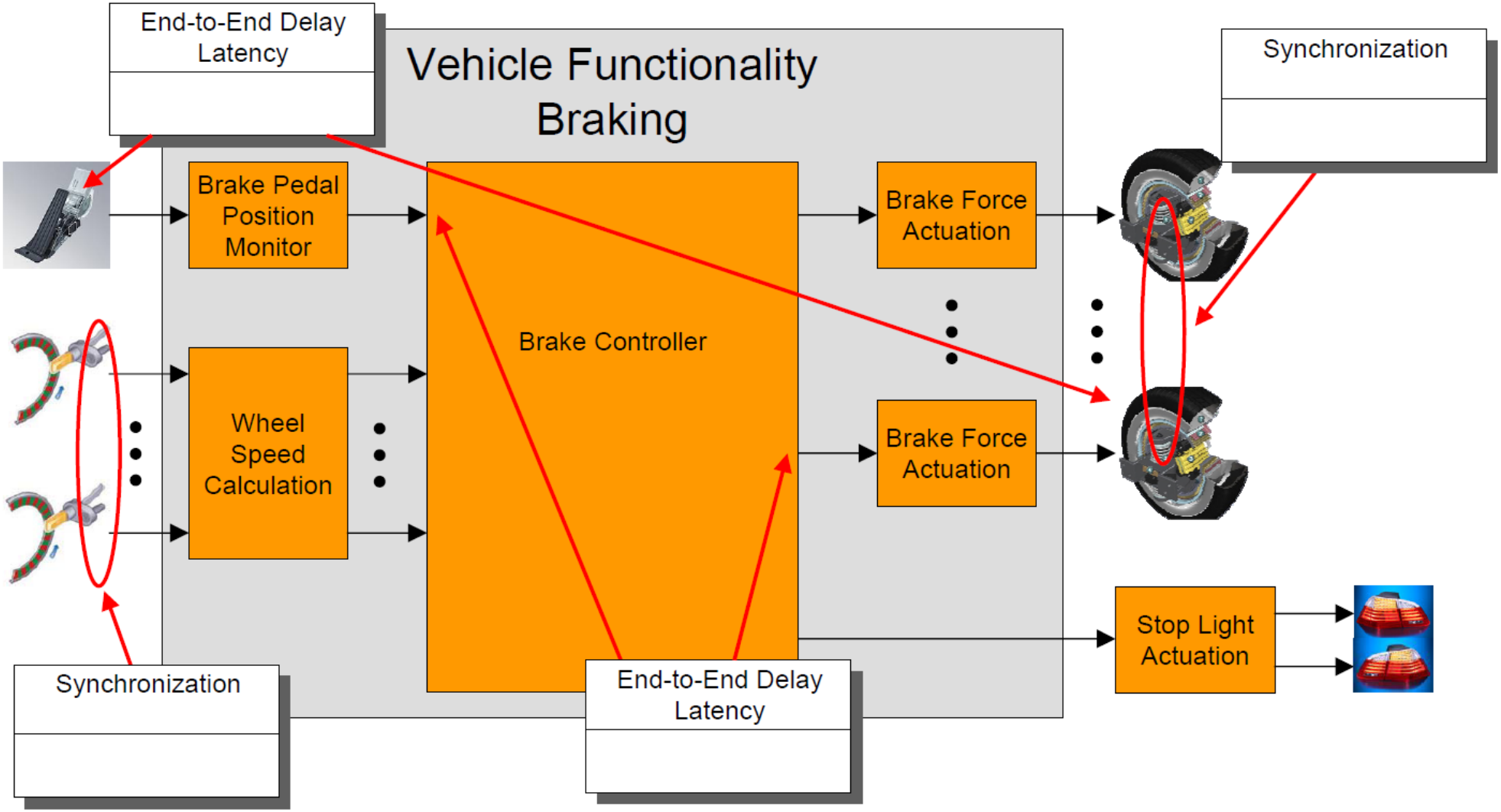
Source: S. Kuntz, Continental

Example Autosar Mapping



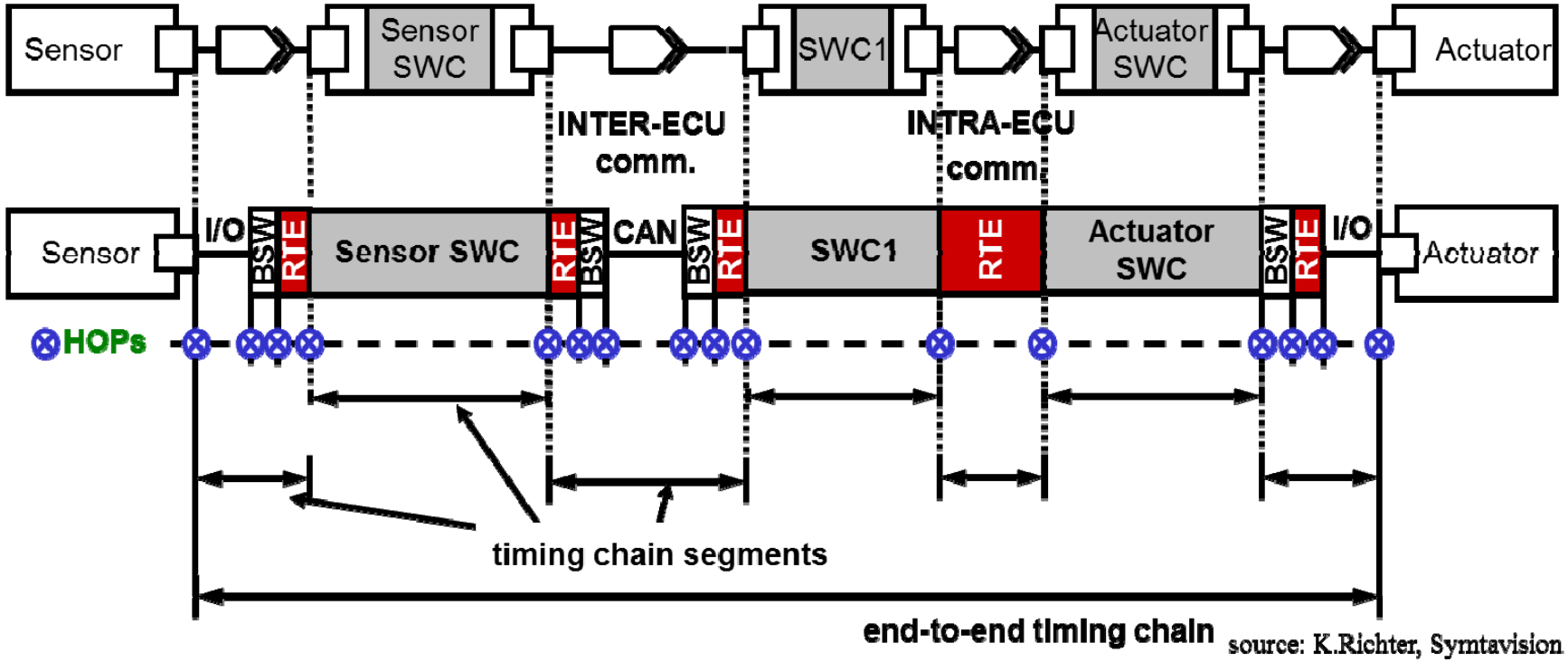
©autosar.org

End-to-end Properties



Summary Challenges

- Applications are often specified as heterogeneous combinations of models of computation (component models, synchronization and communication) with different abstractions.
- End-to-end properties need to be guaranteed:



Key Techniques

Compositionality: Inferring global properties of composite components from the properties of constituent components.

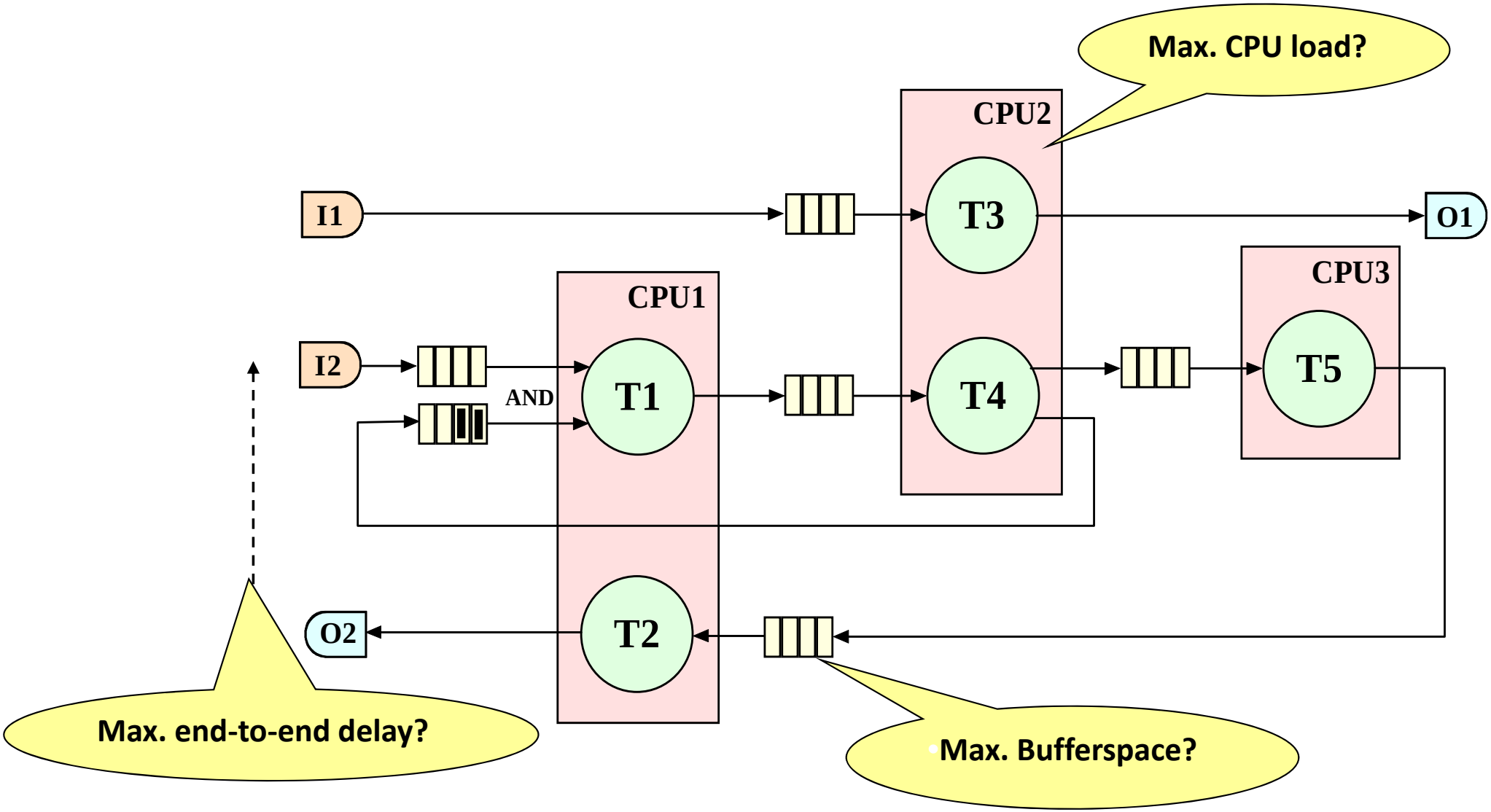
Example: Compositional timing analysis of distributed systems.

Composability: Ensuring that essential properties of components are preserved when they are used to build composite components.

Example: Resource isolation, partitioned design principles.

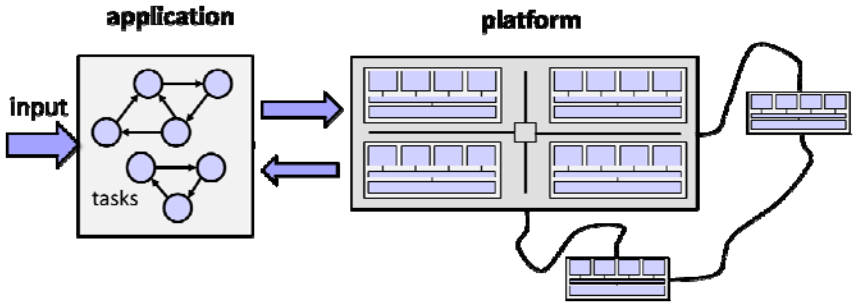
Compositional Analysis Methods

System Level Performance Analysis



Formal analysis methods

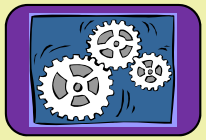
Distributed system



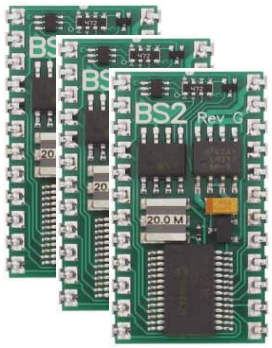
Abstraction

$$r_i = C_i + \sum_{\forall j \in hp(i)} \lceil \frac{r_i}{T_j} \rceil C_j$$

Analysis method

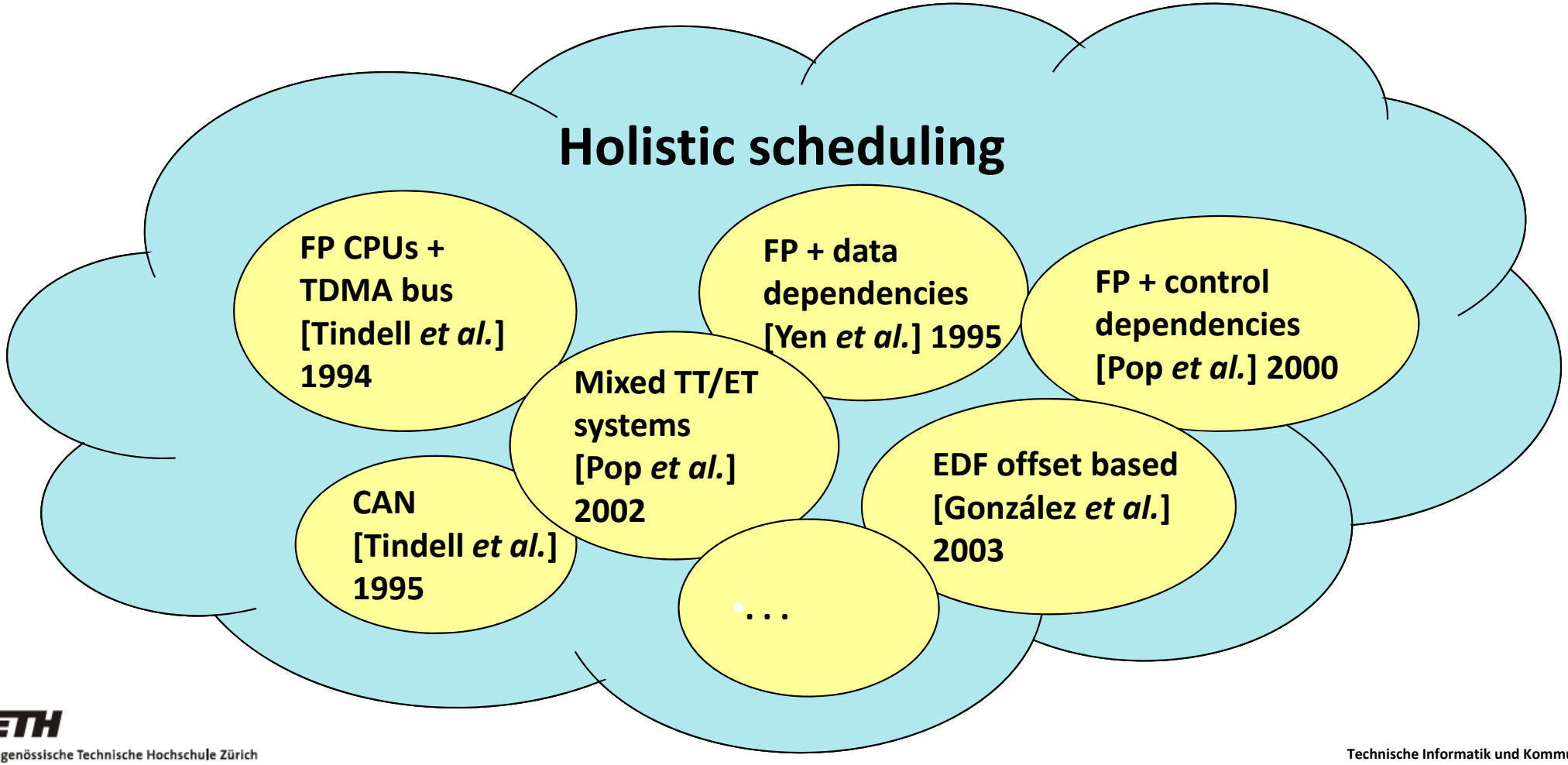


Performance values



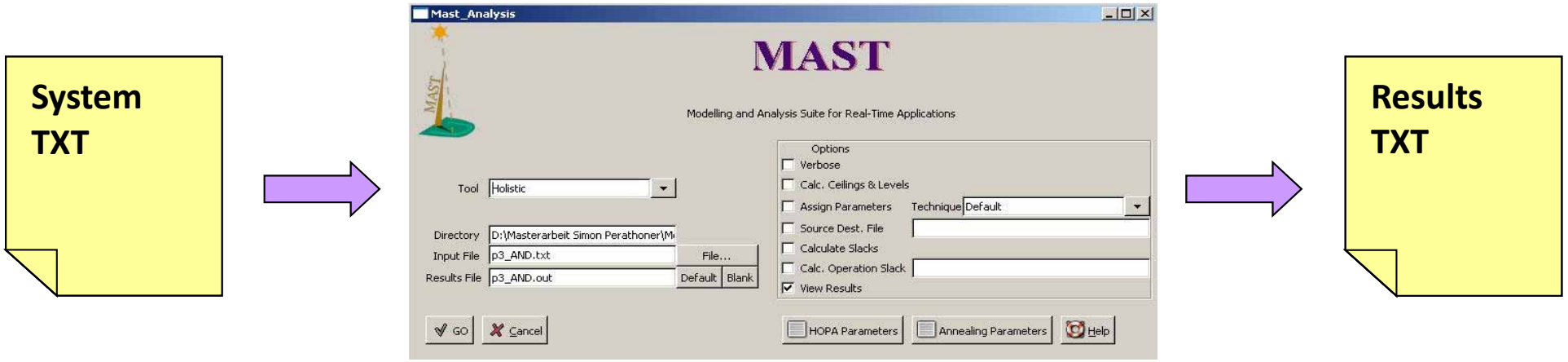
Abstraction 1 - Holistic Scheduling

Basic concept: extend concepts of classical scheduling theory to distributed systems

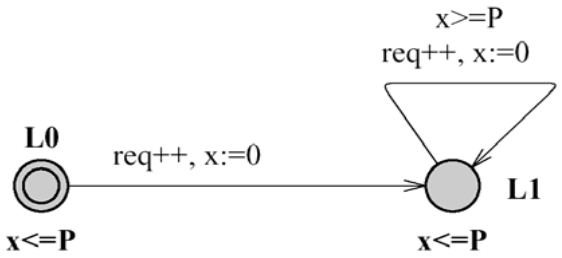


Holistic Scheduling – MAST tool

MAST - The Modeling and Analysis Suite for Real-Time Applications [González Harbour *et al.*]



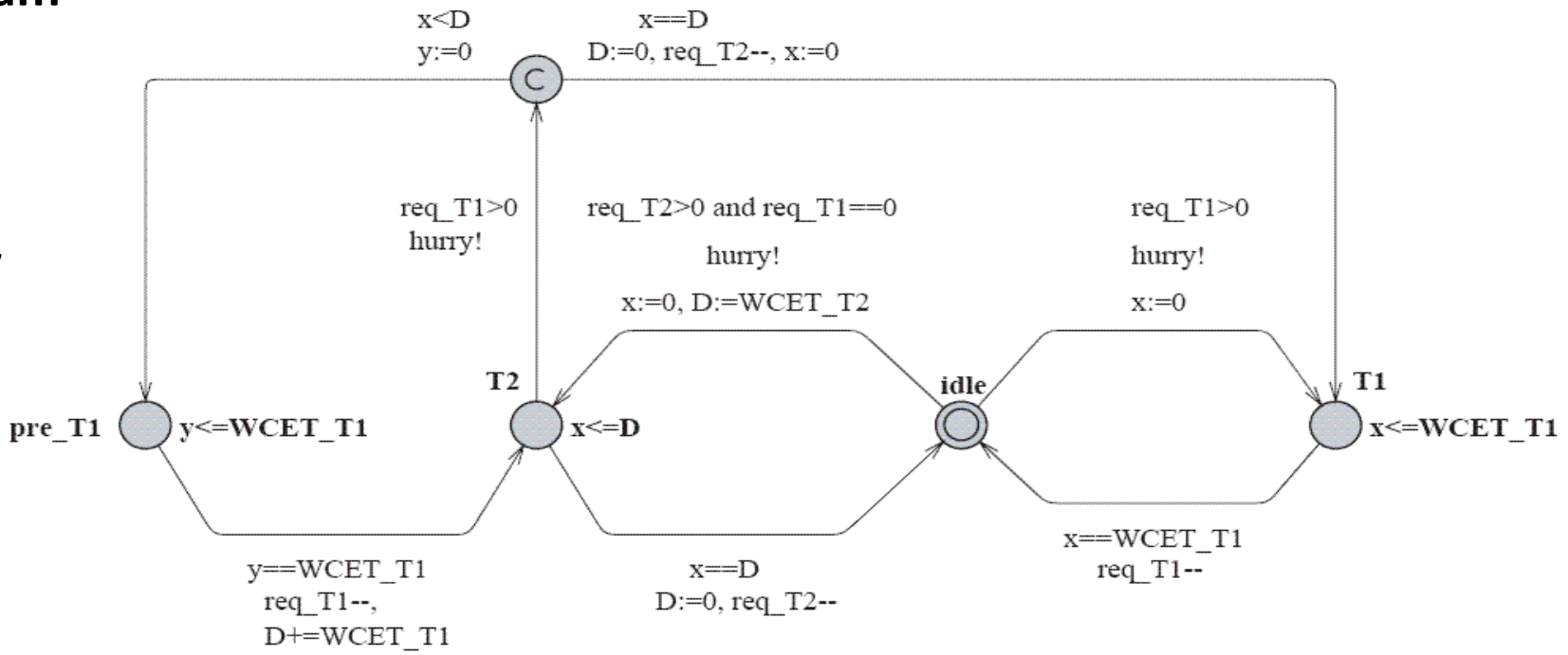
Abstraction 1 – Timed Automata



Verification of performance properties by model checking (e.g. UPPAAL)

periodic stream

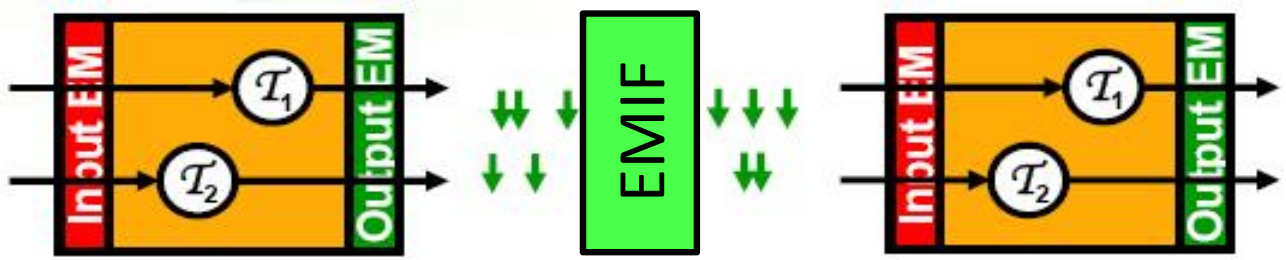
fixed priority scheduling



Abstraction 2 – The SymTA/S Approach

Basic concept: **Application of classical scheduling techniques at resource level and propagation of results to next component**

Problem: **The local analysis techniques require the input event streams to fit given standard event models**

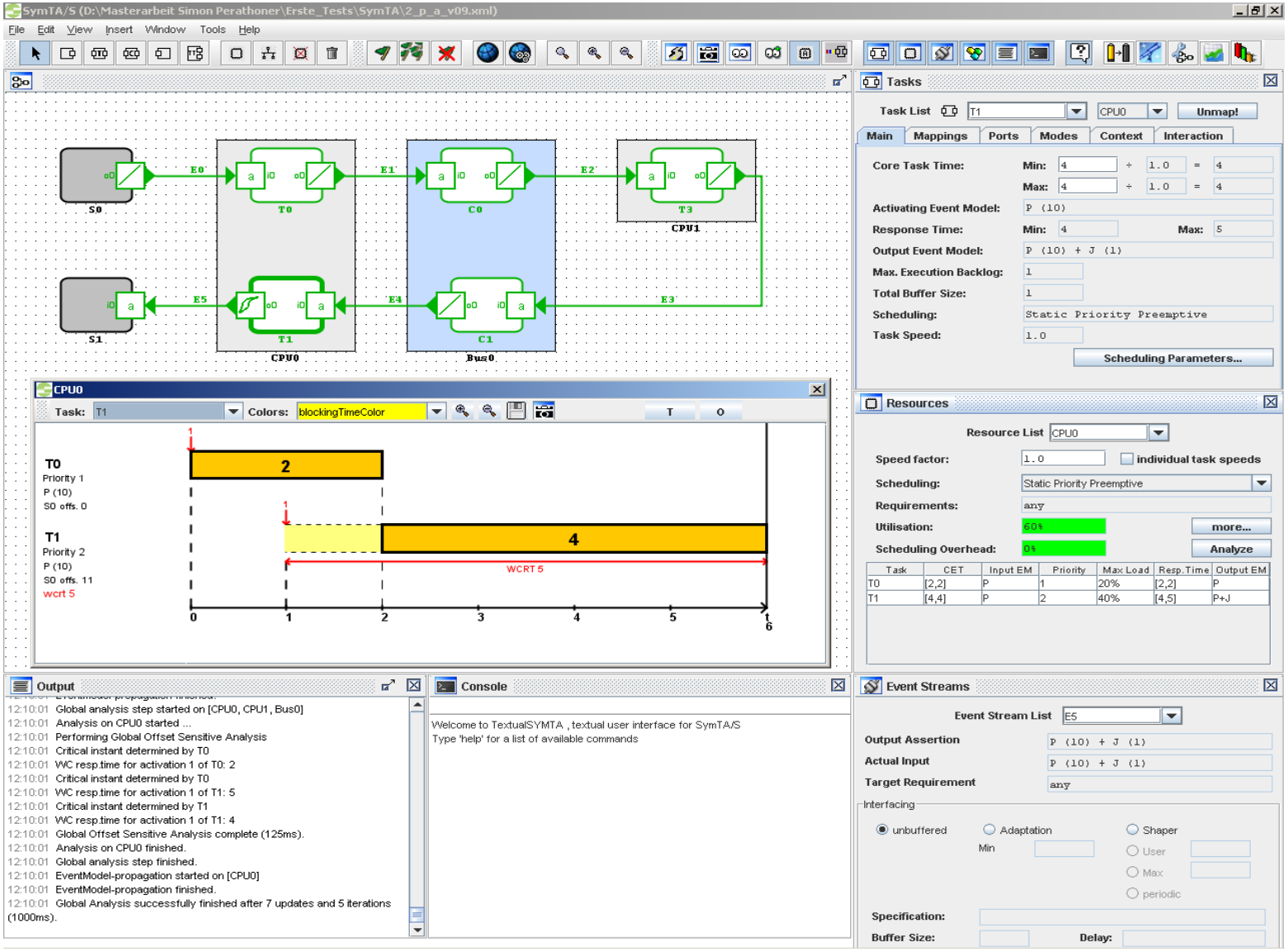


Solution: **Use appropriate interfaces**

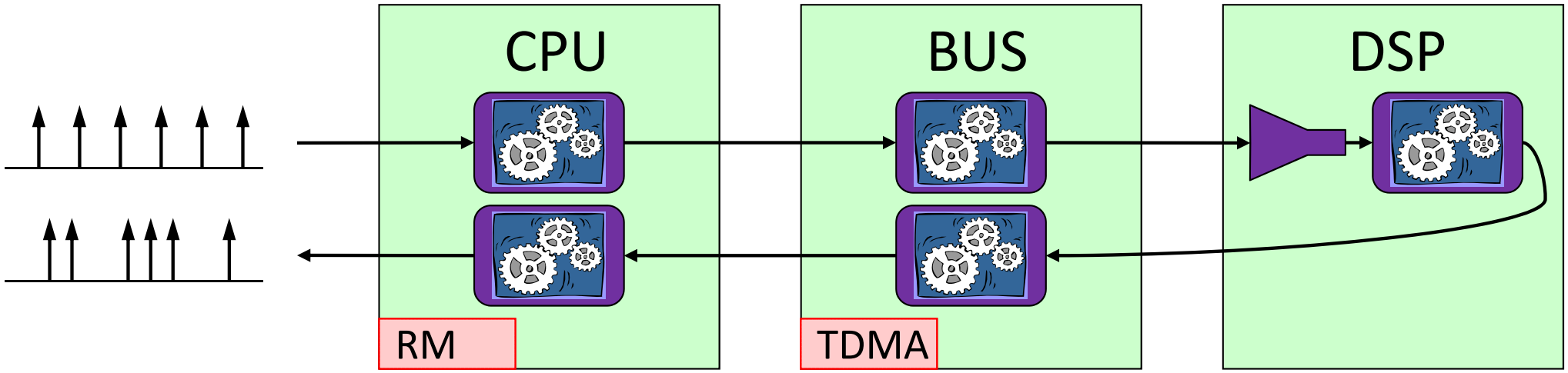
[Ernst et al.]

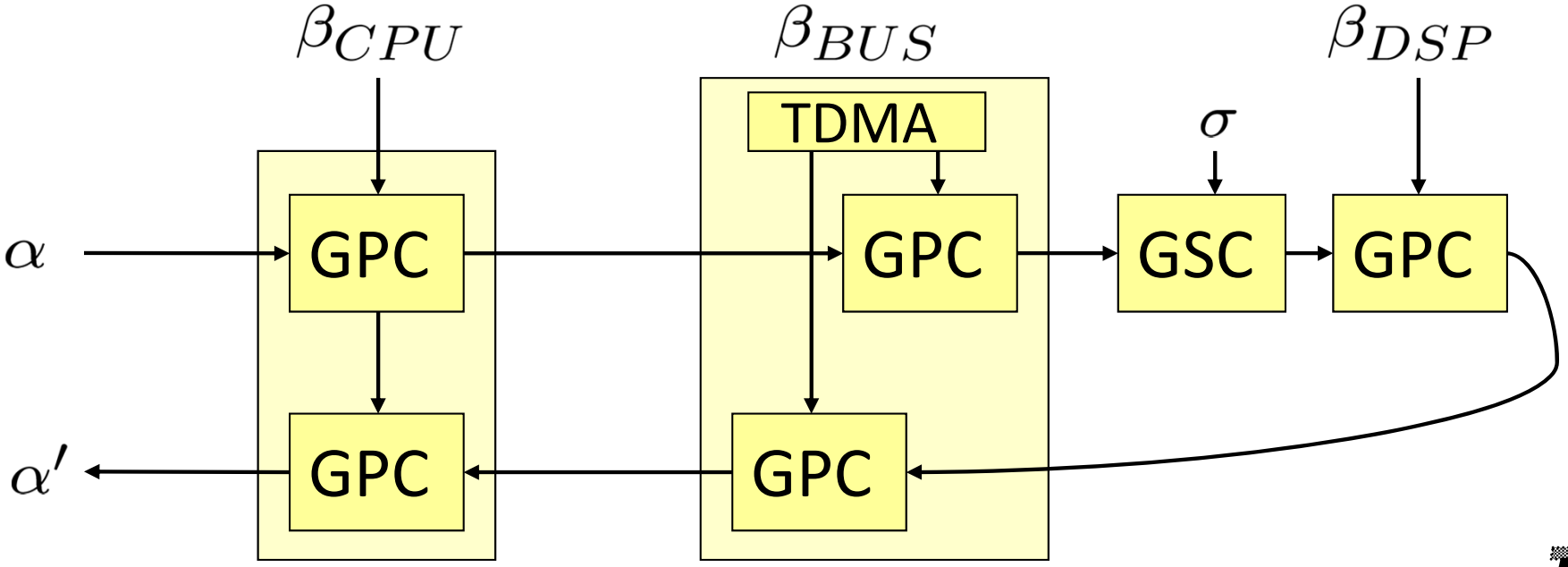
SymTA/S – Tool

SYMTA VISION

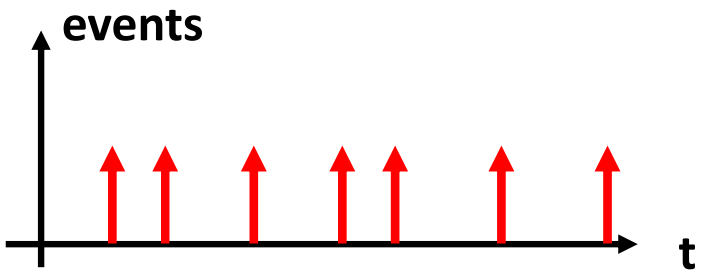


System Composition in Real-Time Calculus

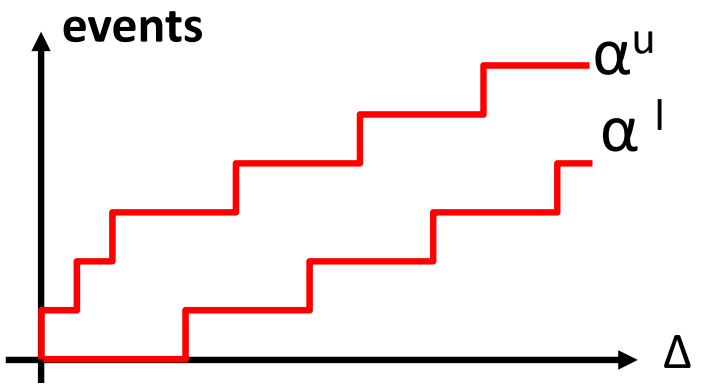




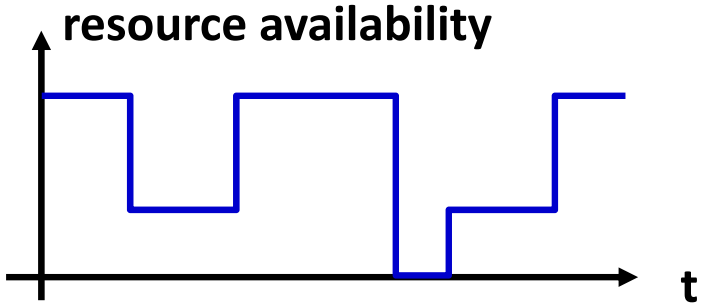
Abstraction 3 – Real-Time Calculus (RTC)



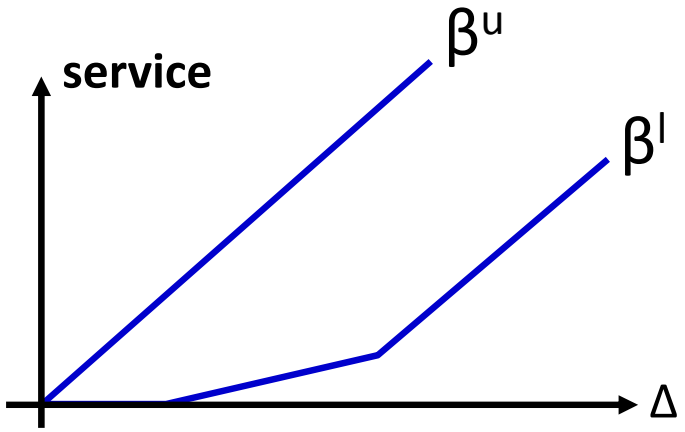
Load model



Arrival curves

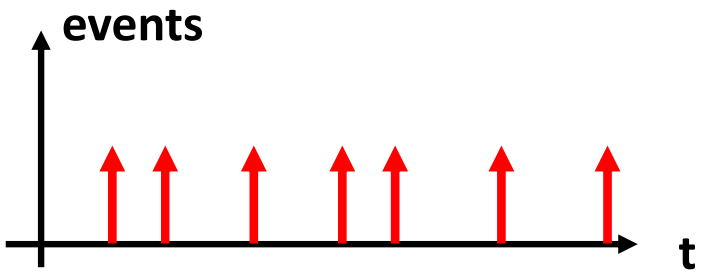


Service model

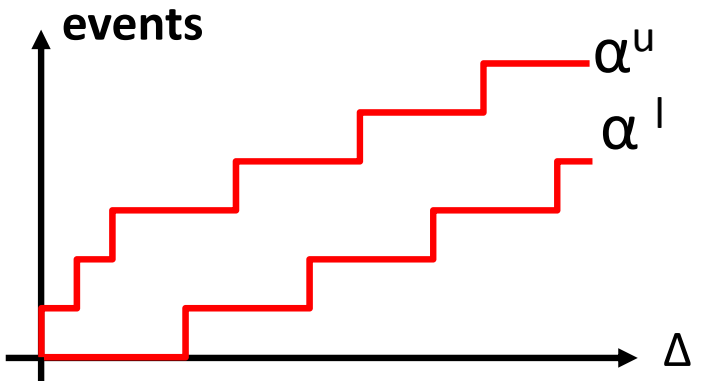


Service curves

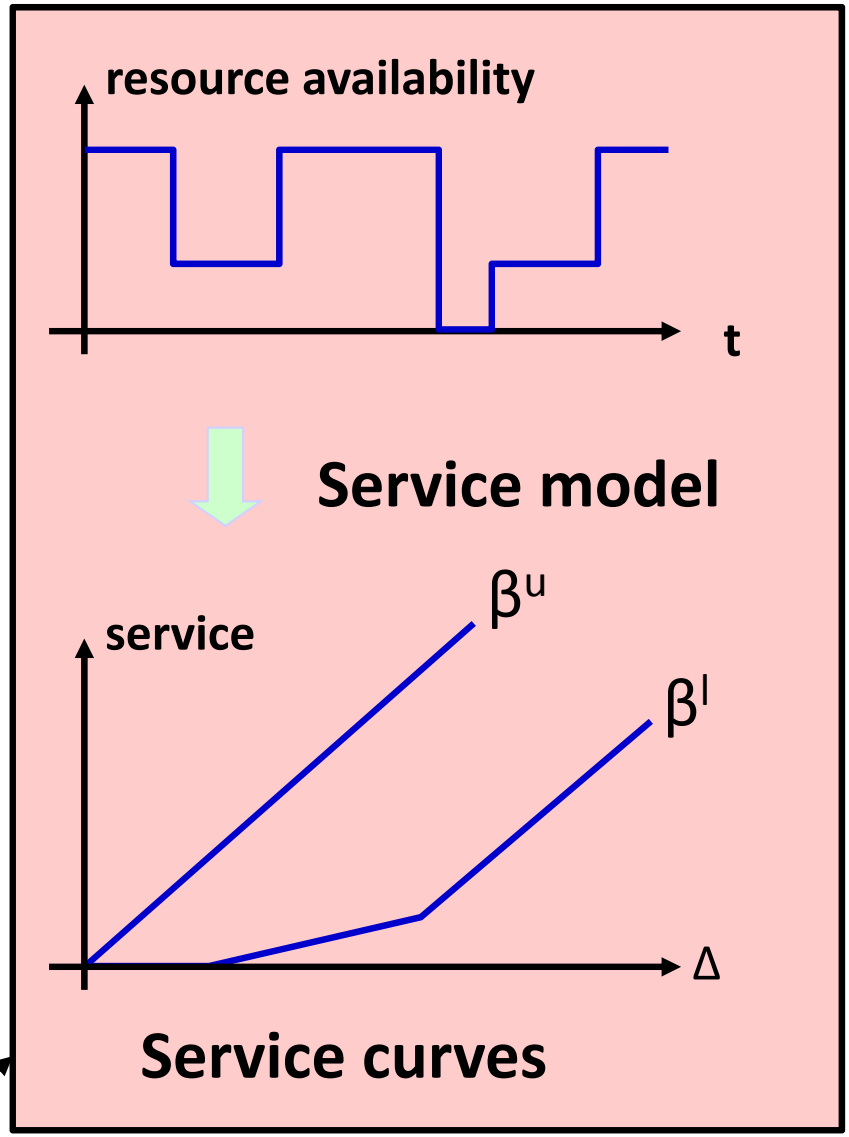
Abstraction 3 – Real-Time Calculus (RTC)



Load model

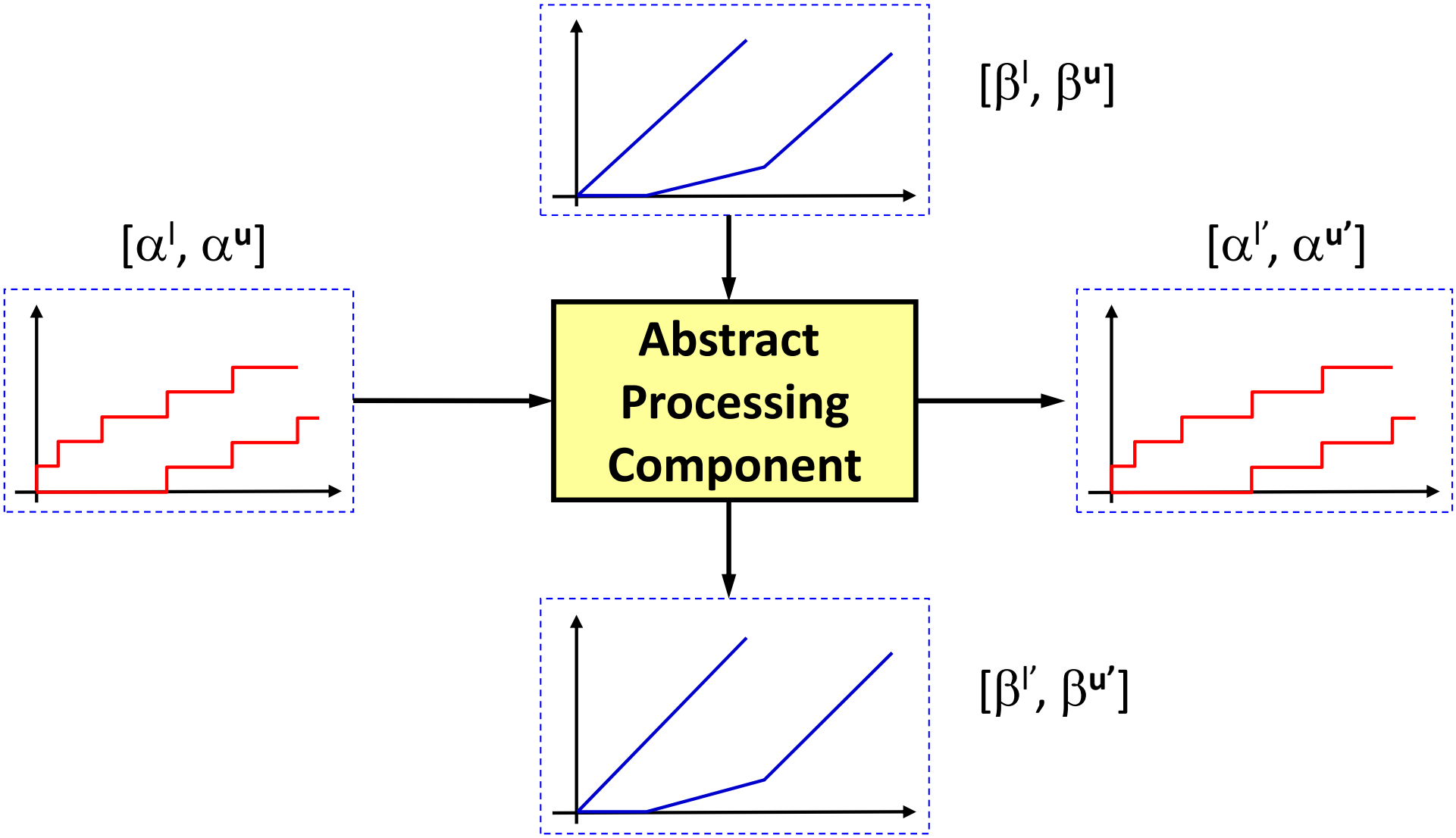


Arrival curves



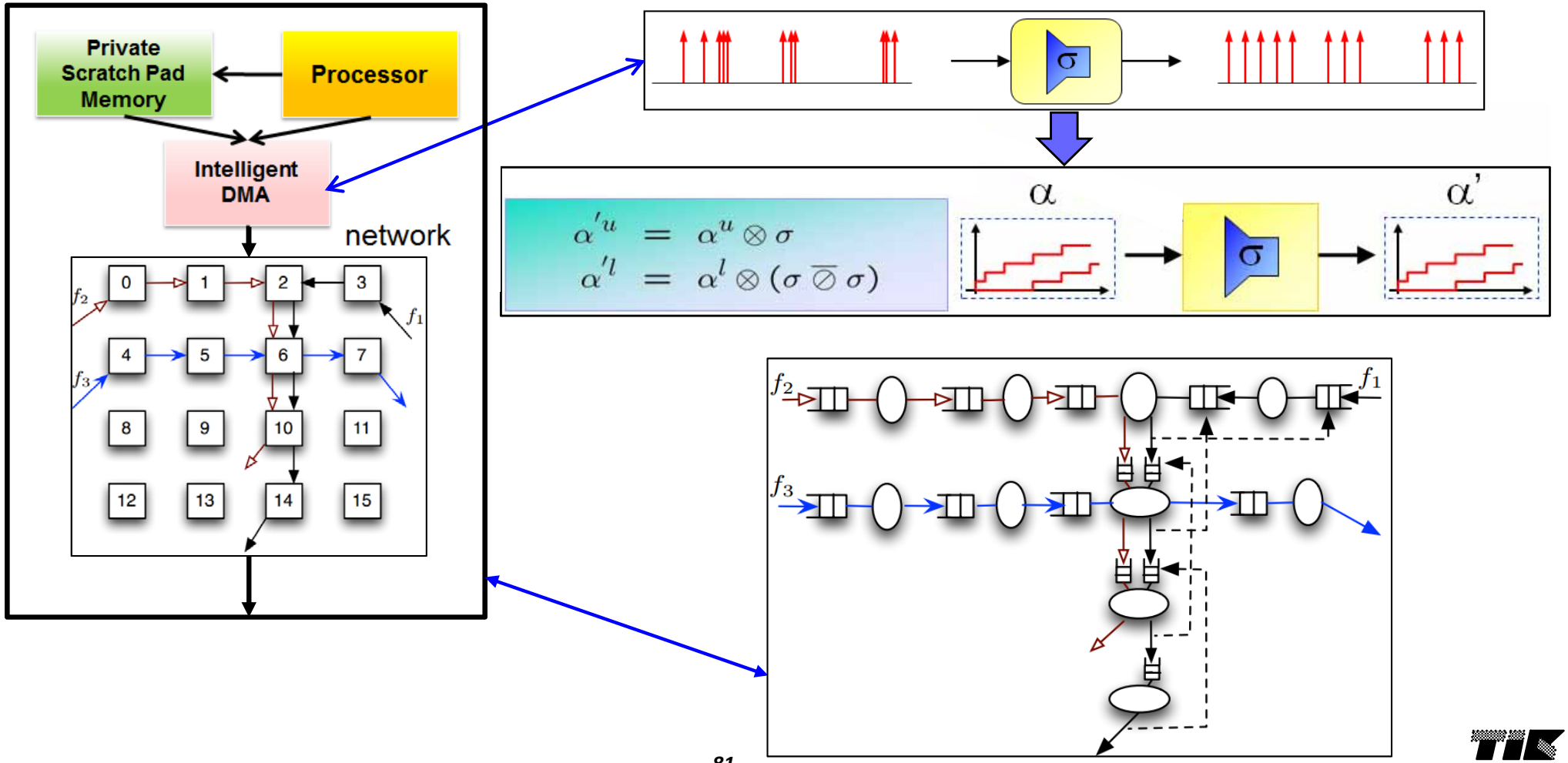
first class citizen of analysis

Abstraction 3 – Real-Time Calculus



Example: Network on Chip

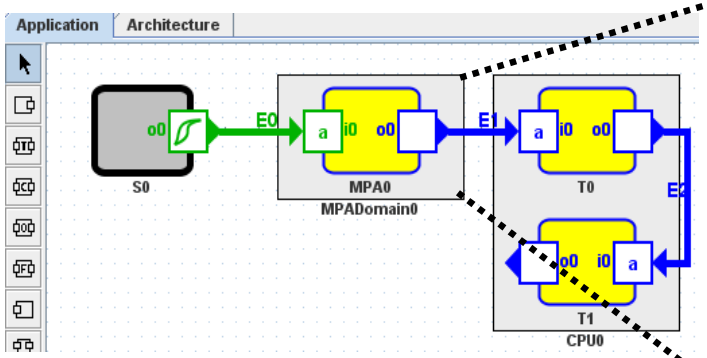
- Composition in the analyis of communication networks by using network-calculus [Cruz] and real-time calculus:



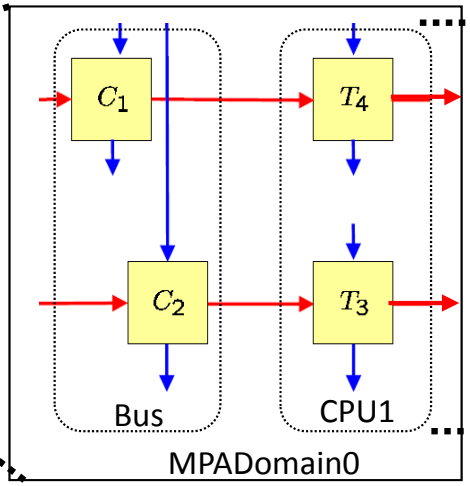
Composing Abstractions

Global Picture

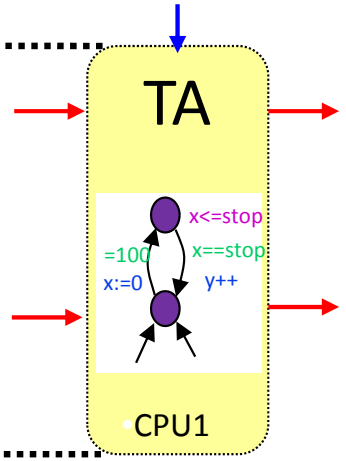
SymTA/S – Holistic Analysis



MPA-RTC Toolbox



Parametric Analysis



Analytical Model
pjd models (but not only)

Analytical Model
time interval domain

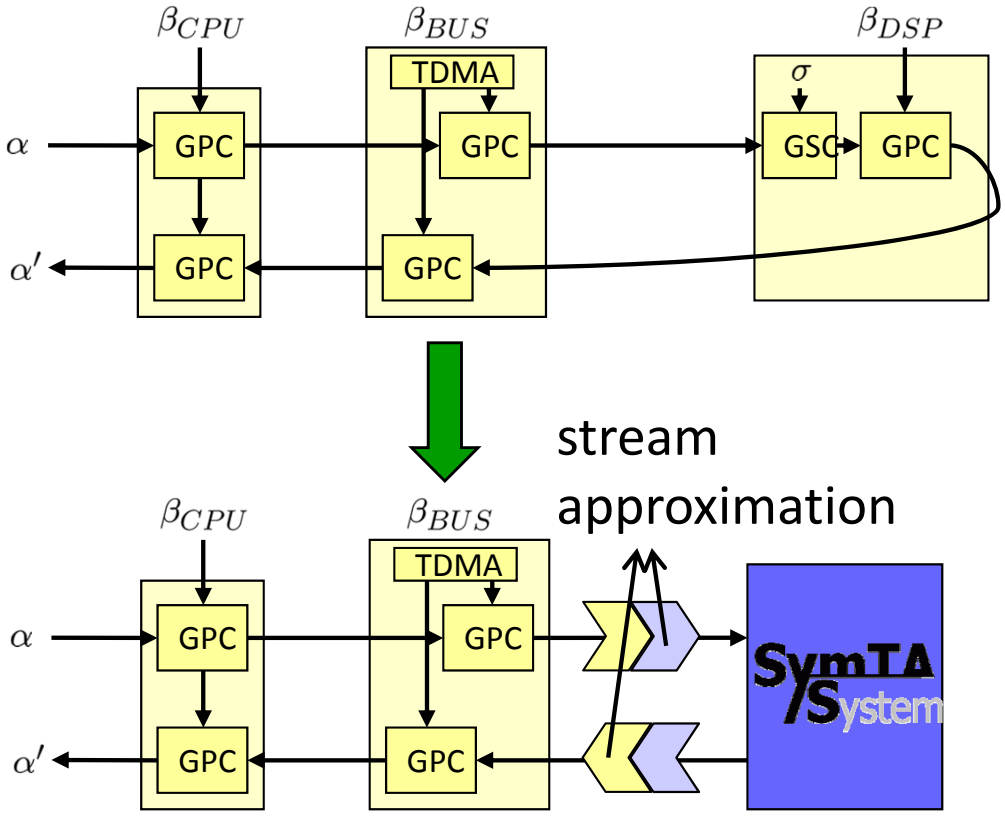
State-based Model
timed automata

[Perathoner et al., 2011]

Benefits

- incorporate advanced anaysis methods; adapt analysis method to component characteristics
- reduce analysis complexity

Composition by Stream Approximation



Replace RTC component by

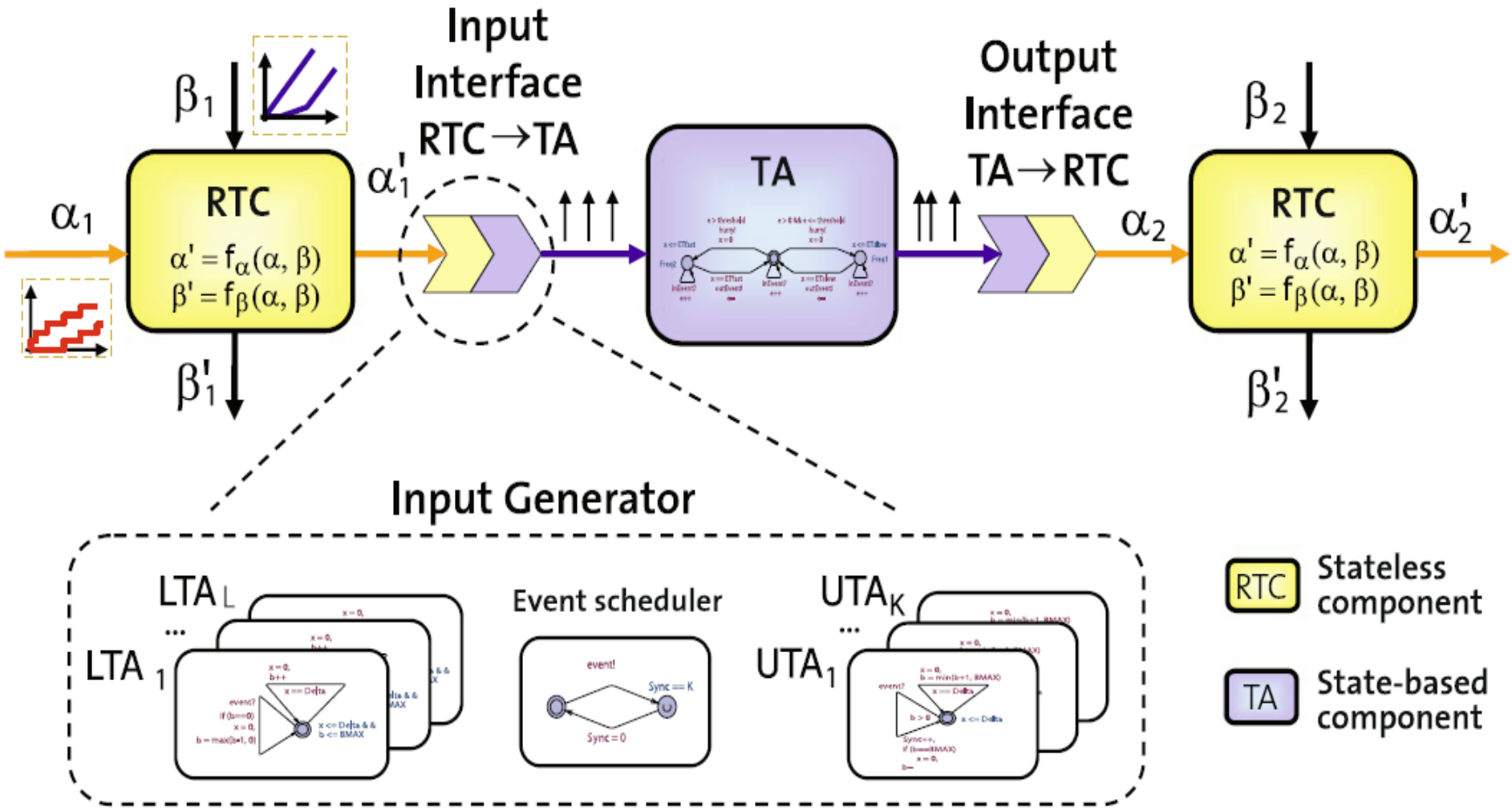
- classical single-resource real-time analysis
- holistic scheduling analysis (MAST, *Gonzales*, Tindell, Yen, *Eles*, …)
- Symta/S compositional analysis (*Ernst*)
- timed automata

[Perathoner et al., 2011]

Benefits

- incorporate advanced anaysis methods; adapt analysis method to component characteristics
- reduce analysis complexity

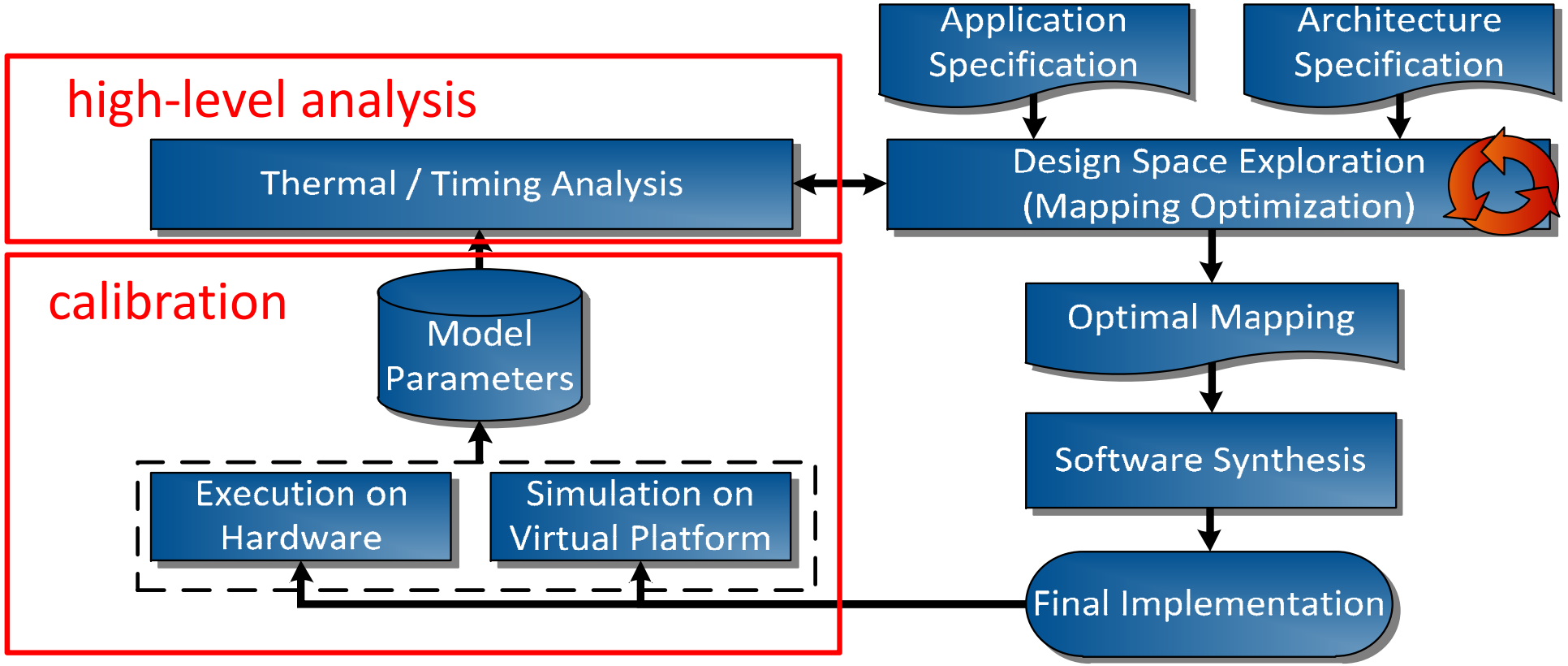
Composition by Domain Transformation



[Lampka et al., 2011-2013]

Integration in Mapping Optimization

- Distributed Application Layer (DAL) [ETH Zurich]
- Automatically generate an optimized mapping of applications onto distributed platforms



Contents

- Why?
- Interference
- Task Level
  - single task on single processor
  - several tasks on single processor
  - single tasks on parallel processors
- Task Graph Level
- ***Outlook and Summary***

Key Techniques

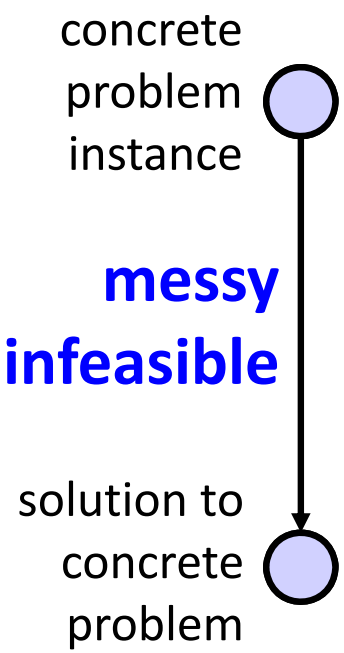
Compositionality: Inferring global properties of composite components from the properties of constituent components.

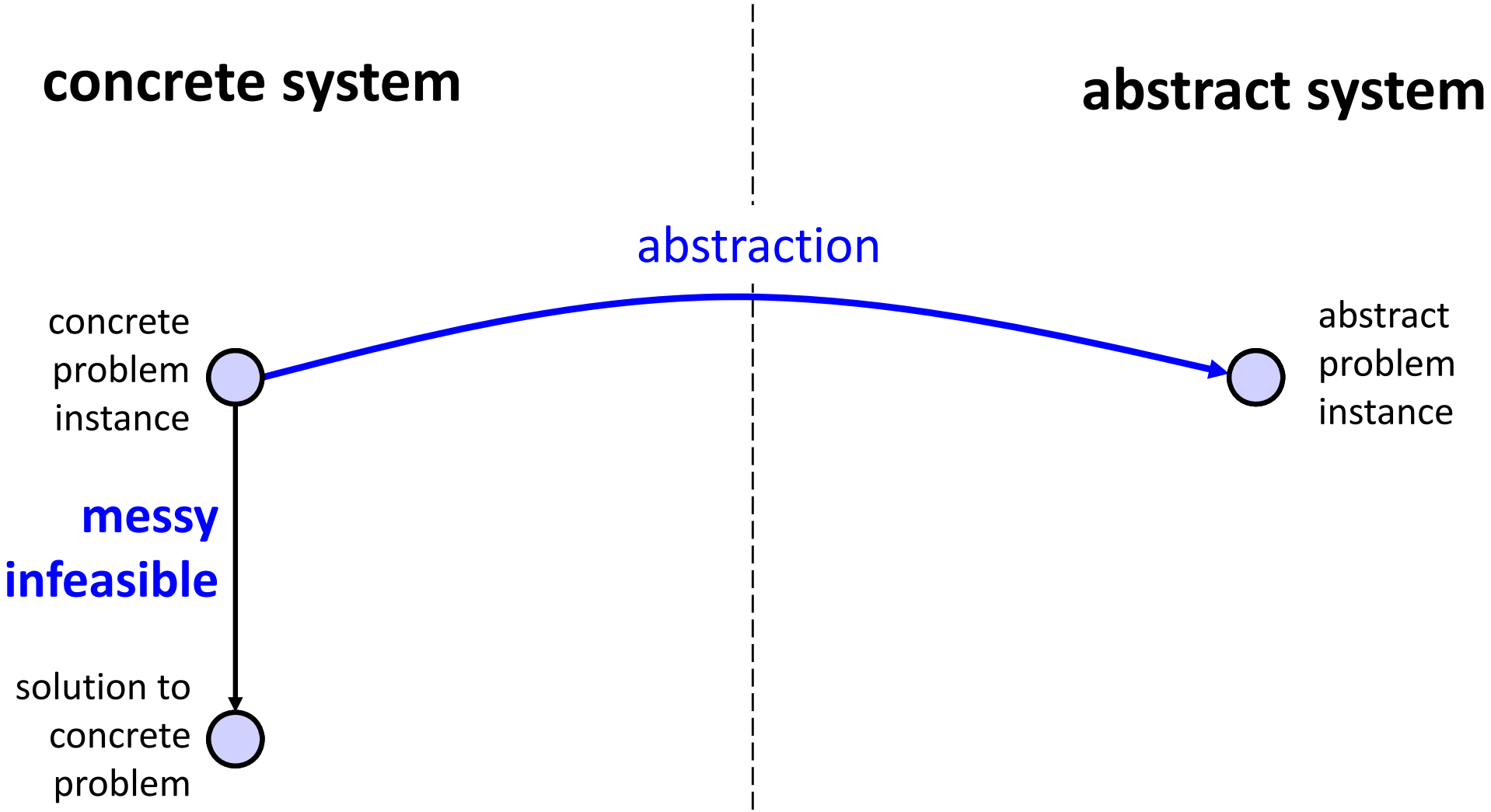
Example: Compositional timing analysis of distributed systems.

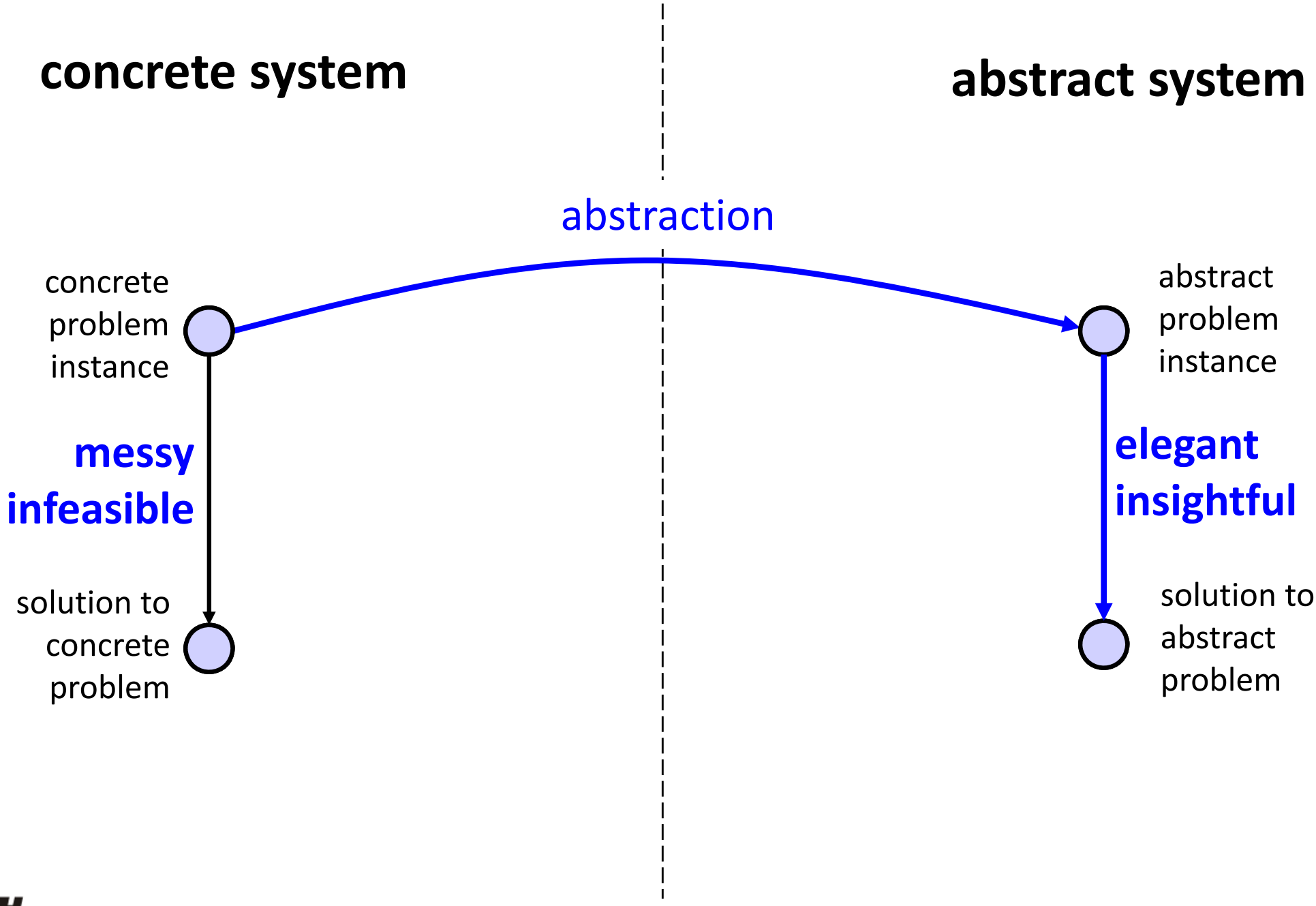
Composability: Ensuring that essential properties of components are preserved when they are used to build composite components.

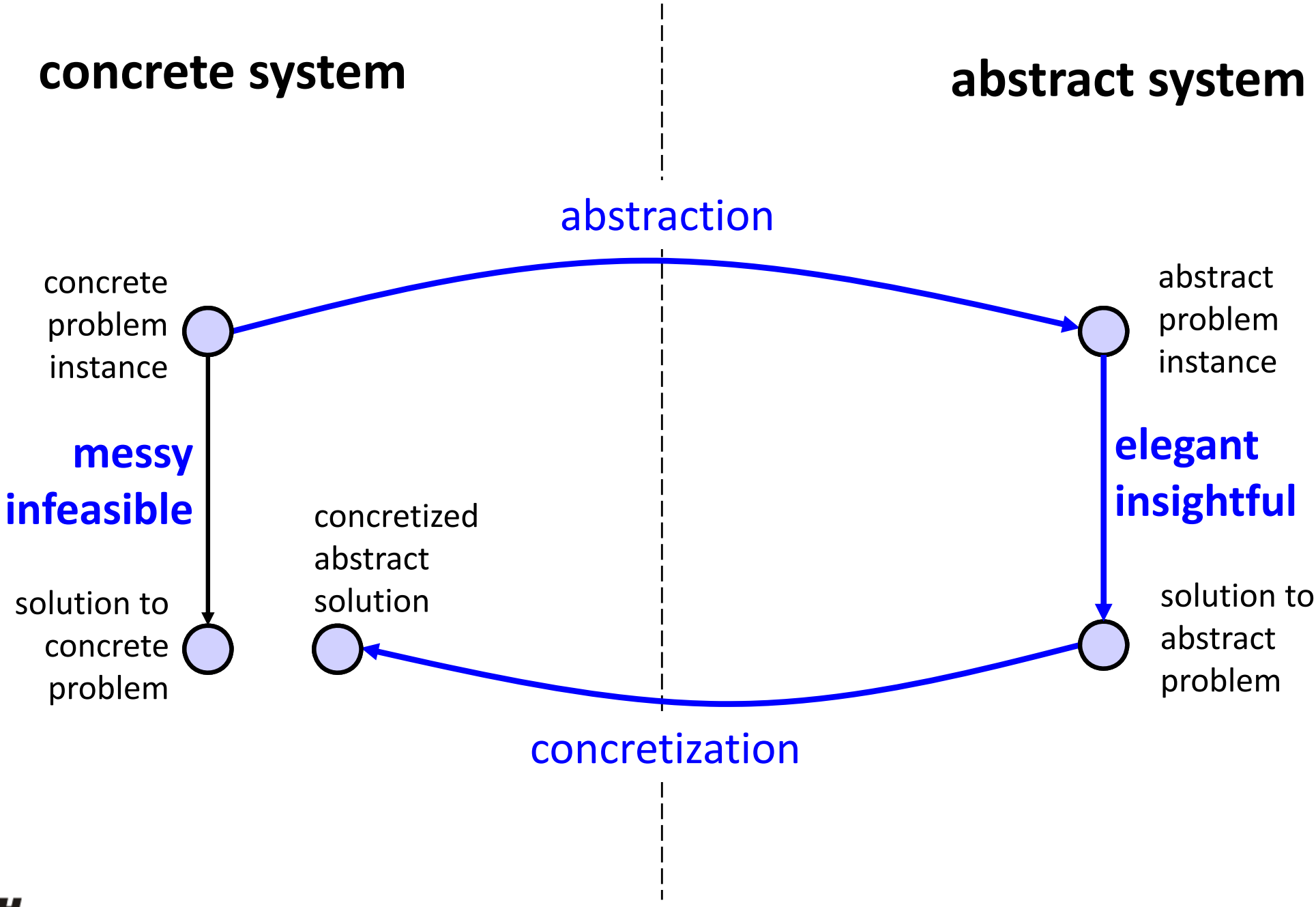
Example: Resource isolation, partitioned design principles.

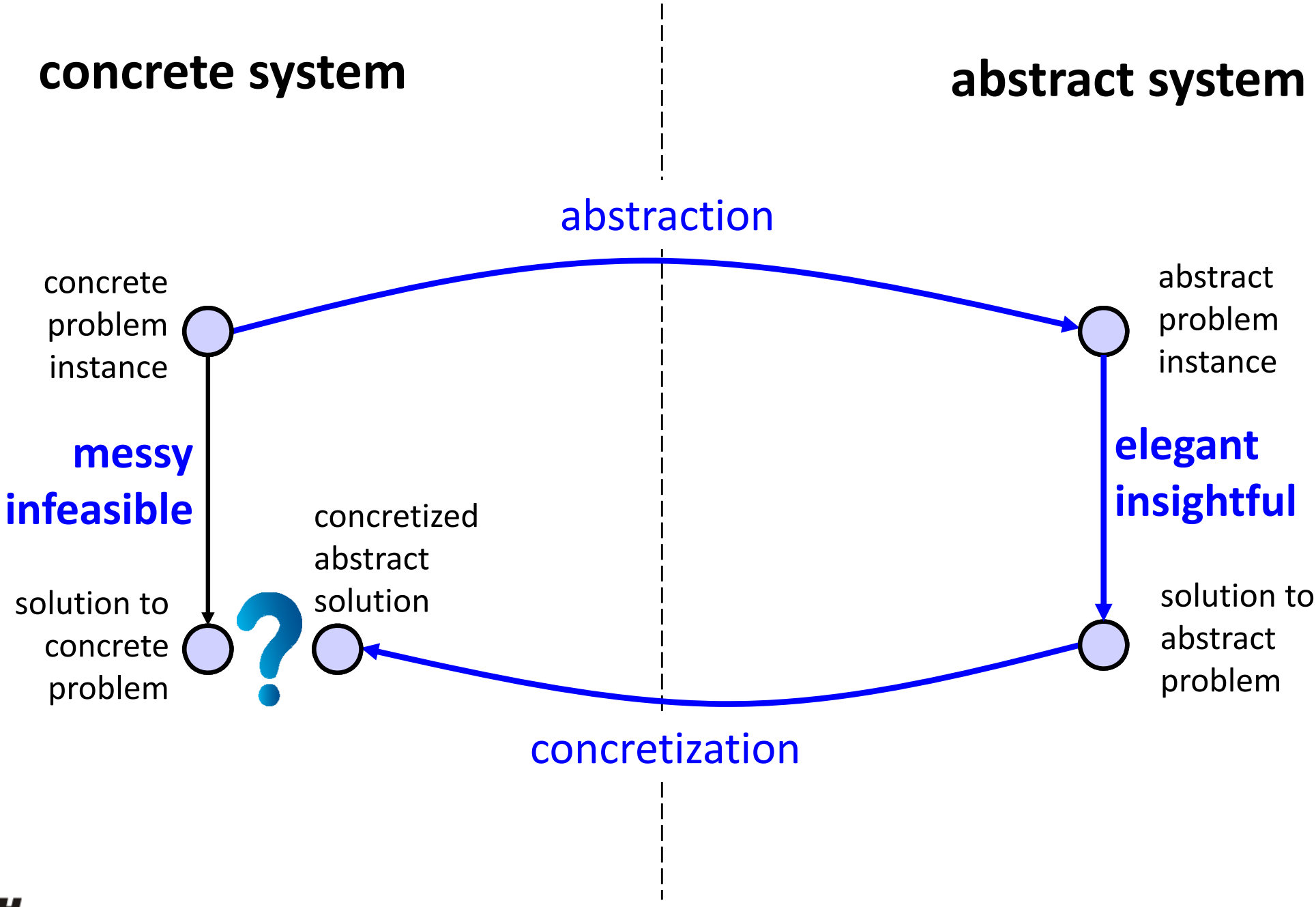
concrete system




concrete system
abstract system
abstraction
concrete problem instance
abstract problem instance
messy infeasible
solution to concrete problem


concrete system
abstract system
abstraction
concrete problem instance
messy infeasible
solution to concrete problem
abstract problem instance
elegant insightful
solution to abstract problem


concrete system
abstract system
abstraction
concrete problem instance
abstract problem instance
messy infeasible
elegant insightful
concretized abstract solution
solution to concrete problem
solution to abstract problem
concretization


concrete system
abstract system
abstraction
concrete problem instance
abstract problem instance
messy infeasible
elegant insightful
concretized abstract solution
solution to concrete problem
solution to abstract problem
concretization