Architectural Aspects of Deriving Performance Guarantees: Timing Analysis and Timing Predictability

Reinhard Wilhelm, Jan Reineke

Timing Analysis and Timing Predictability: Architectural Dependences

Reinhard Wilhelm and Jan Reineke

Department of Computer Science
Saarland University
Saarbrücken, Germany

ISCA 2010, Tutorial on:
Architectural Aspects of Deriving Performance Guarantees:
Timing Analysis and Timing Predictability
June 20, 2010

Hard Real-Time Systems

Safety critical applications:

- Avionics, automotive, train industries, manufacturing control

Sideairbag in car, Reaction in <10 mSec

Crankcraft-synchronous tasks,

Reaction in <45 μSec

Embedded controllers must finish their tasks *within given time bounds*.

Developers would like to know the *Worst-Case Execution Time (WCET)* to give a guarantee

Hard Real-Time Systems

- Embedded controllers are expected to finish their tasks reliably within time bounds
- Task scheduling must be performed
- Essential: upper bound on the execution times of all tasks statically known
- Commonly called Worst-Case Execution Time (WCET)
- Analogously, Best-Case Execution Time (BCET)

Static Timing Analysis
the Application Domain

The Problem:

- Given
  1. a software to produce some reaction,
  2. a hardware platform, on which to execute the software,
  3. required reaction time.
- Derive:
  - a guarantee for timeliness.

What does the execution time depends on?

- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)

- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)

What does the execution time depends on?

- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  - Caused by caches, pipelines, speculation, etc.

- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)

What does the execution time depends on?

- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  - Caused by caches, pipelines, speculation, etc.
  - Explosion of the space of inputs **and** initial states
    $\Rightarrow$ all exhaustive approaches infeasible
- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)

What does the execution time depends on?

- the input—this has always been so and will remain so,
- the initial state of the platform—this is (relatively new)
  - Caused by caches, pipelines, speculation, etc.
  - Explosion of the space of inputs **and** initial states
    $\Rightarrow$ all exhaustive approaches infeasible
- interferences from the environment—this depends on whether the system design admits it (preemptive scheduling, interrupts, multi-core)
  - External interferences as seen from analyzed task

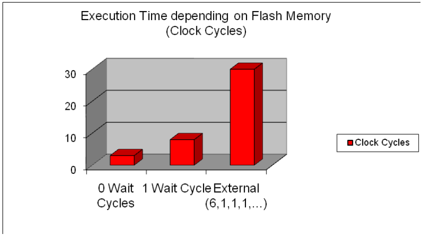
Access Time

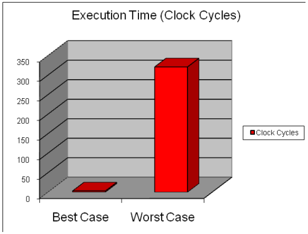
SAARLAND UNIVERSITY
COMPUTER SCIENCE

x=a+b; →

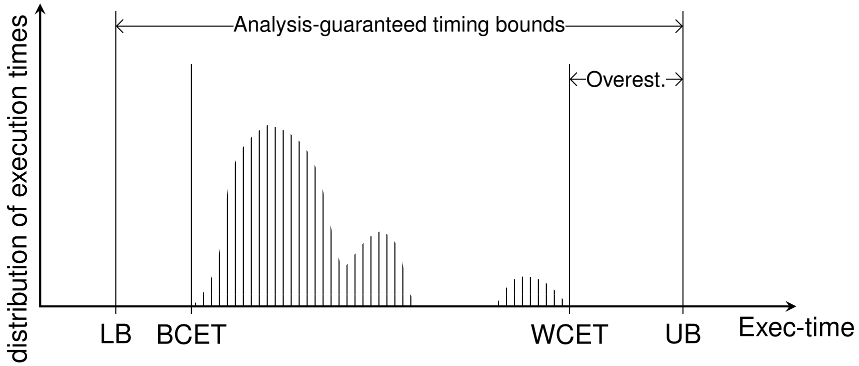
```
LOAD    r2, _a
LOAD    r1, _b
ADD     r3,r2,r1
```

MPC5xx   PPC755



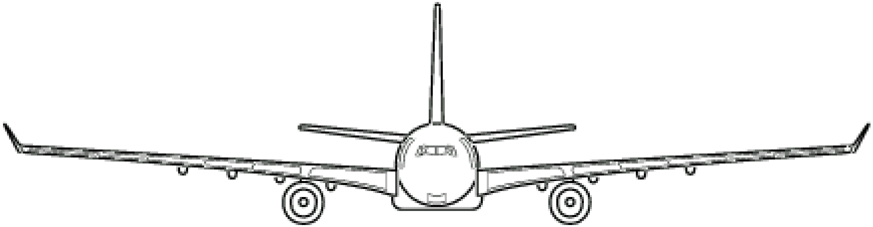


Timing Analysis


Analysis-guaranteed timing bounds
Overest.
distribution of execution times
LB
BCET
WCET
UB
Exec-time

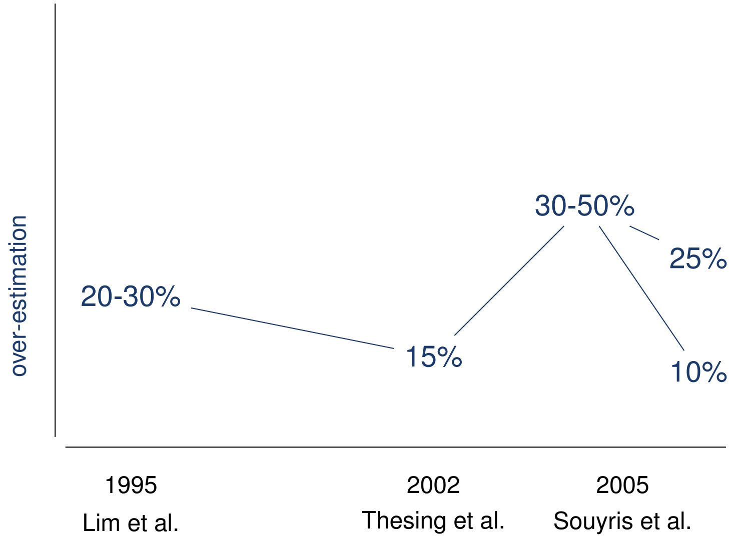
aiT WCET Analyzer

IST Project DAEDALUS final review report: "*The AbsInt tool is probably the best kind in the world and it is justified to consider this result as a breakthrough.*"

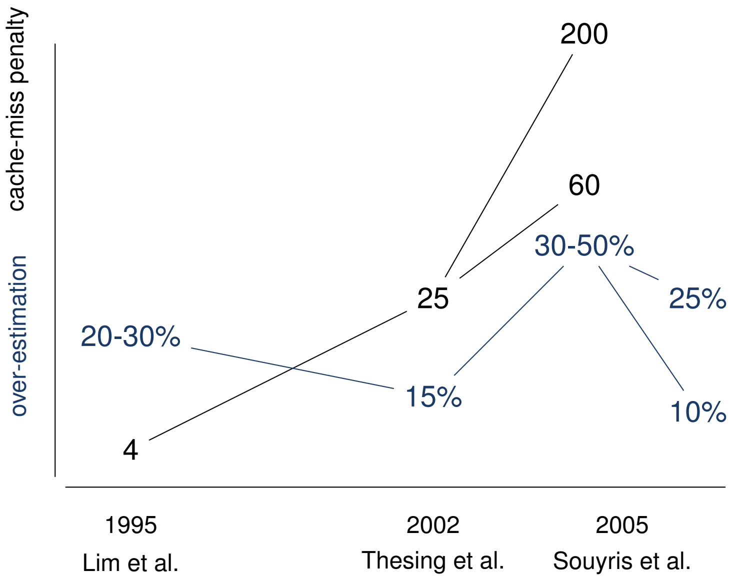


Several time-critical subsystems of the airbus A380 have been certified using aiT; aiT is the only validated tool for these applications.

Tremendous Progress during the 15 past Years

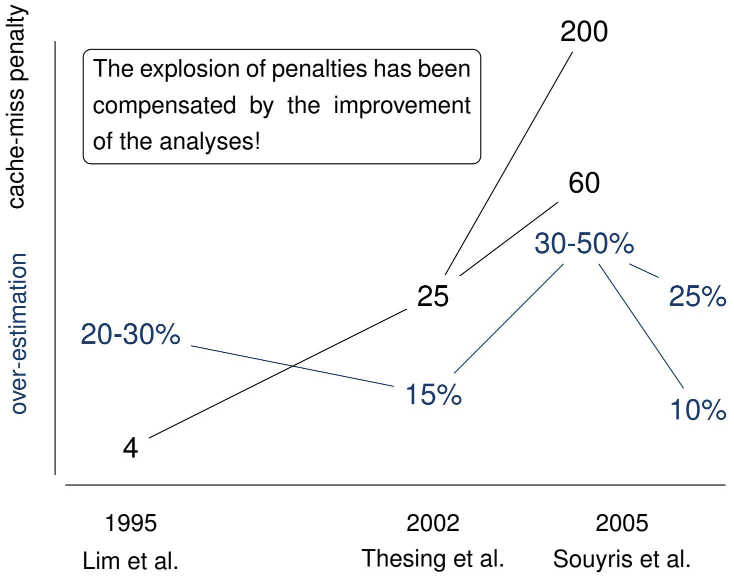


Tremendous Progress during the 15 past Years

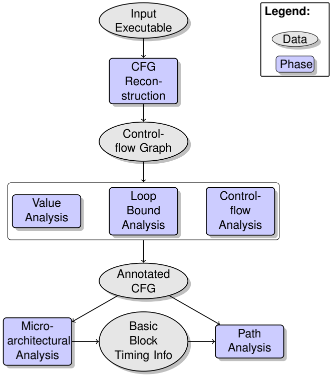


Tremendous Progress during the 15 past Years

The explosion of penalties has been compensated by the improvement of the analyses!



Static Timing Analysis



Predictability

- Predictability: not a boolean property
- Some performance-enhancing features, like certain
  - Caches
  - Pipelines

  are analyzable, others are not...
- Explore trade-offs between (worst-case)predictability, (average-case) performance, and cost
- Goal:
  Design architectures with high worst-case performance which can be precisely and efficiently determined

Predictable Multi-Core Architecture

- Predictable cores are a prerequisite for predictable multi-cores
- General problem: *Sharing* of resources
  - Main memory, caches
  - Busses
  - I/O
  - Flash memory

Outline

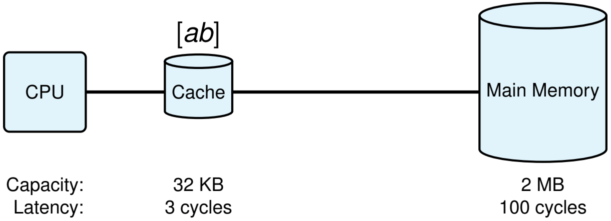
Outline

1. Timing analysis and predictability of caches
   - Caches
   - Cache Analysis for Least-Recently-Used
   - Beyond Least-Recently-Used
     - Predictability Metrics
     - Sensitivity – Caches and Measurement-Based Timing Analysis
2. Predictability of architectures
   - Pipeline Analysis
   - Timing anomalies
   - Classification of architectures
3. Extension to multi-core
   - Why multi-core?
   - Predictable multi-core
   - Analysis of current multi-core

Outline

Caches

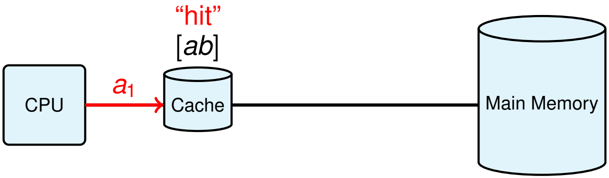
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

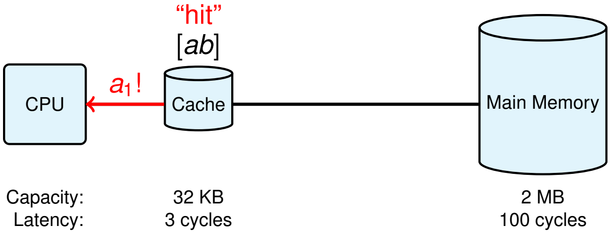


Capacity:	32 KB	2 MB
Latency:	3 cycles	100 cycles

- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

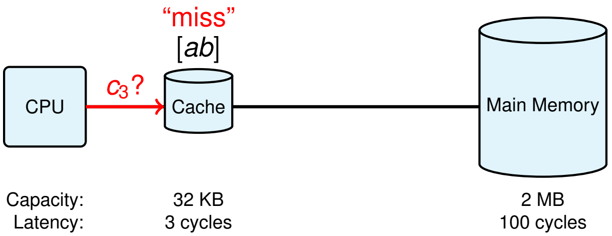
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

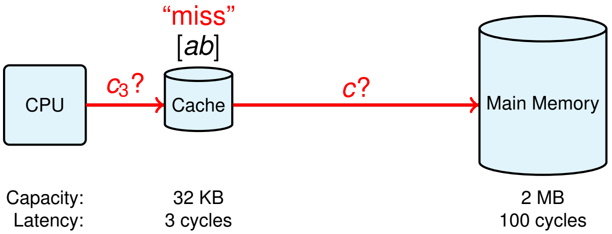
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

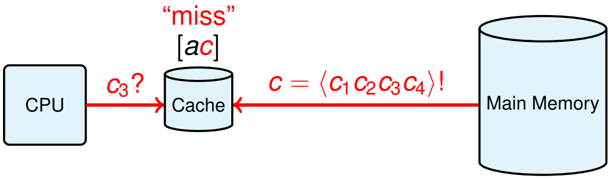
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

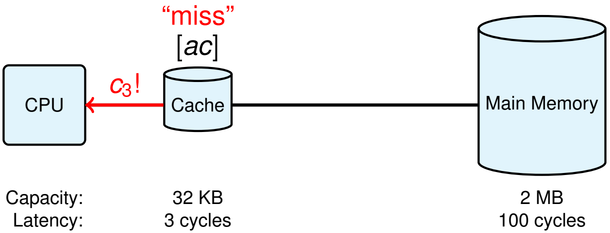


	Cache	Main Memory
Capacity:	32 KB	2 MB
Latency:	3 cycles	100 cycles

- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

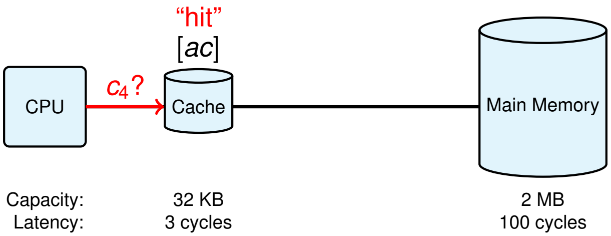
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

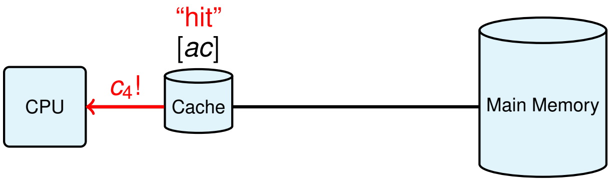
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

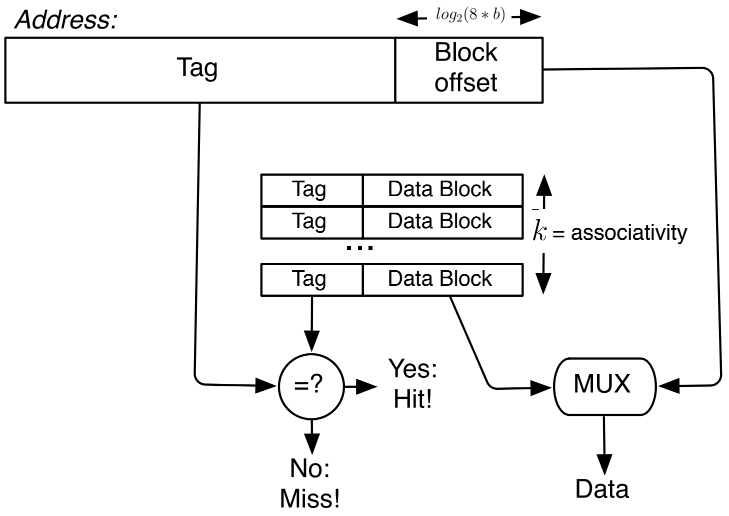
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



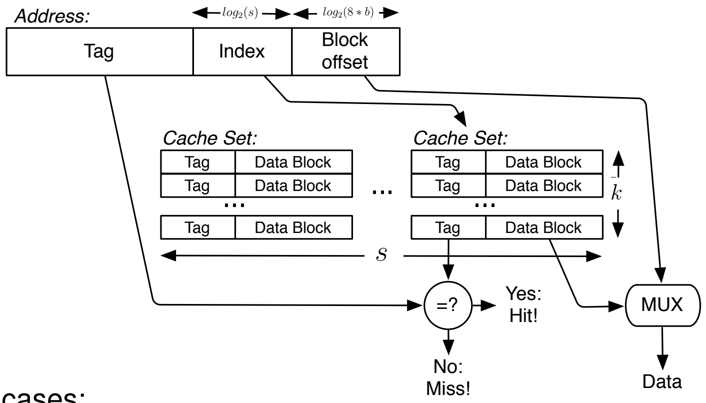
Capacity:	32 KB	2 MB
Latency:	3 cycles	100 cycles

- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Fully-Associative Caches



Set-Associative Caches



Special cases:

- direct-mapped cache: only one line per cache set
- fully-associative cache: only one cache set

Cache Replacement Policies

- Least-Recently-Used (LRU) used in
  INTEL PENTIUM I and MIPS 24K/34K
- First-In First-Out (FIFO or Round-Robin) used in
  MOTOROLA POWERPC 56X, INTEL XSCALE, ARM9, ARM11
- Pseudo-LRU (PLRU) used in
  INTEL PENTIUM II-IV and POWERPC 75X
- Most-Recently-Used (MRU) as described in literature

Each cache set is treated independently:
$\longrightarrow$ Set-associative caches are compositions of fully-associative caches.

Outline

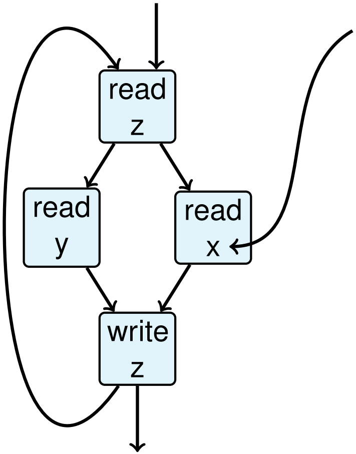
Cache Analysis

Two types of cache analyses:

1. Local guarantees: classification of individual accesses
   - May-Analysis $\longrightarrow$ Overapproximates cache contents
   - Must-Analysis $\longrightarrow$ Underapproximates cache contents
2. Global guarantees: bounds on cache hits/misses

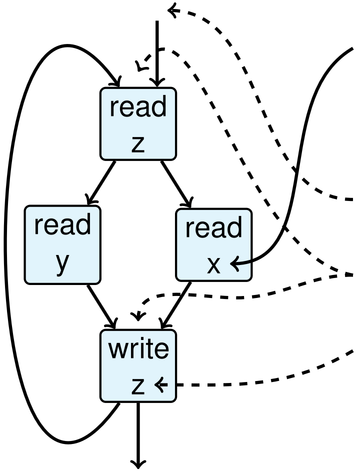
- Cache analyses almost exclusively for LRU
- In practice: FIFO, PLRU, ...

Challenges for Cache Analysis



Always a cache hit/always a miss?

Challenges for Cache Analysis

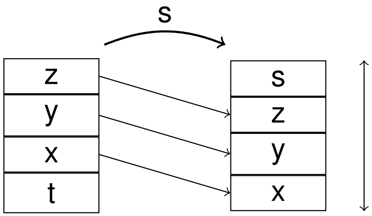


Always a cache hit/always a miss?

1. Initial cache contents unknown.
2. Different paths lead to these points.
3. Cannot resolve address of *z*.

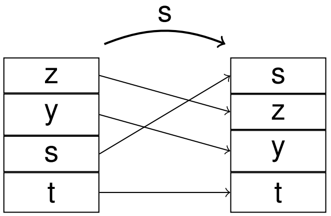
Least-Recently-Used (LRU): Concrete Behavior

“Cache Miss”:



LRU has notion of age

“Cache Hit”:

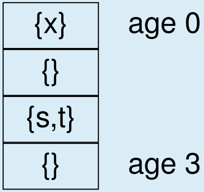


LRU: Must-Analysis: Abstract Domain

- Used to predict *cache hits*.
- Maintains *upper bounds on ages* of memory blocks.
- Upper bound $\leq$ associativity $\longrightarrow$ memory block definitely cached.

Example

Abstract state:



... and its interpretation:

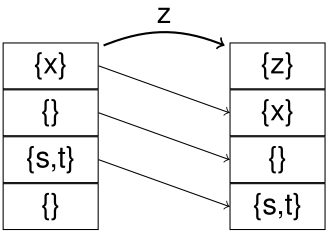
Describes the set of all concrete cache states in which *x*, *s*, and *t* occur,

- *x* with an age of 0,
- *s* and *t* with an age not older than 2.

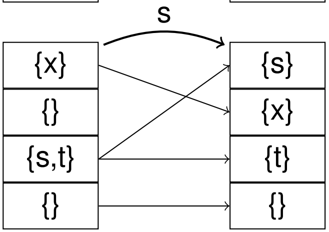
$$\gamma([\{x\},\{\},\{s,t\},\{\}]) = \{[x,s,t,a],[x,t,s,a],[x,s,t,b],\ldots\}$$

LRU: Must-Analysis: Update

"Potential Cache Miss":



"Definite Cache Hit":



Why does *t* not age in the second case?

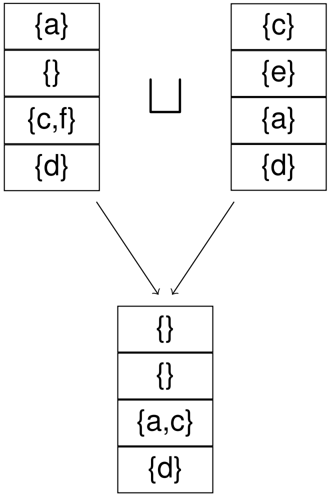
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



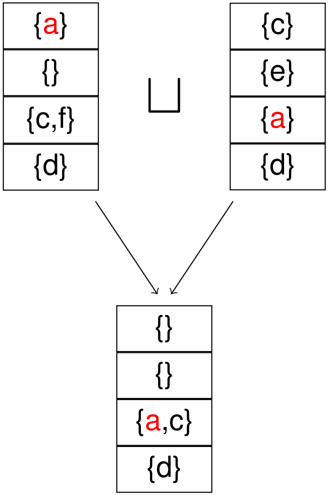
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



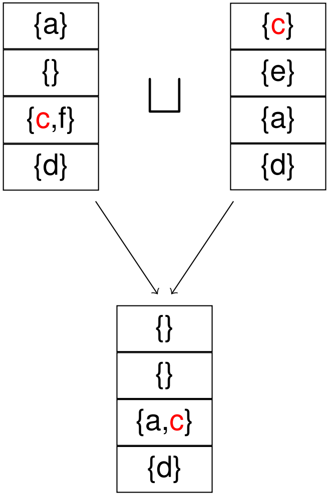
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



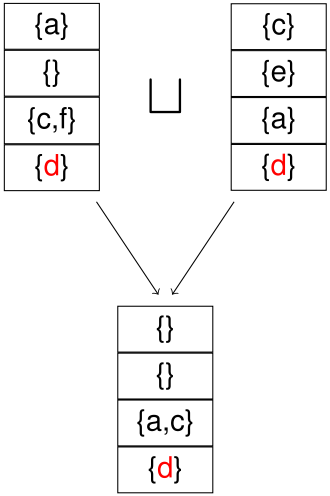
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



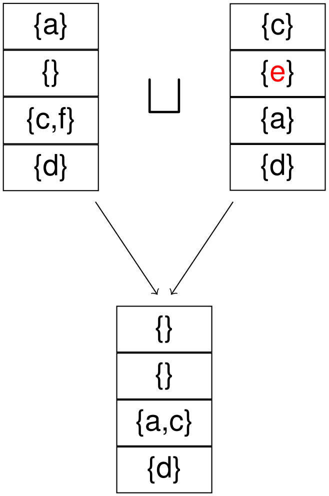
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Intersection + Maximal Age”



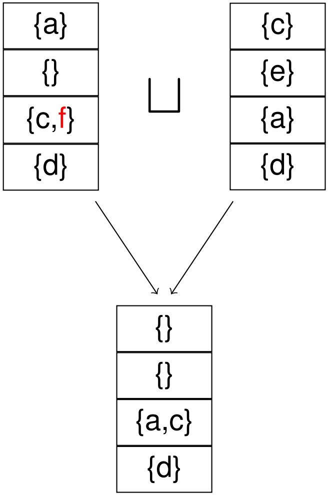
LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Intersection + Maximal Age”



LRU: Must-Analysis: Join

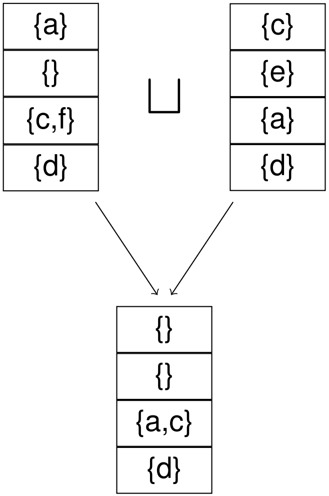
Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Intersection + Maximal Age”

How many memory blocks can be in the must-cache?



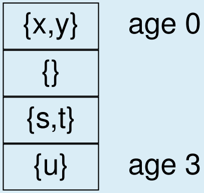
LRU: May-Analysis: Abstract Domain

- Used to predict *cache misses*.
- Maintains *lower bounds on ages* of memory blocks.
- Lower bound $\geq$ associativity
  $\longrightarrow$ memory block definitely *not* cached.

Example

. . . and its interpretation:

Abstract state:



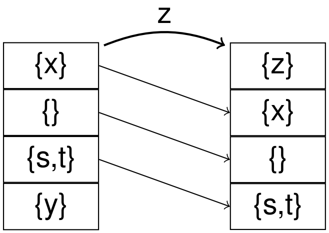
Describes the set of all concrete cache states in which no memory blocks except *x*, *y*, *s*, *t*, and *u* occur,

- *x* and *y* with an age of at least 0,
- *s* and *t* with an age of at least 2,
- *u* with an age of at least 3.

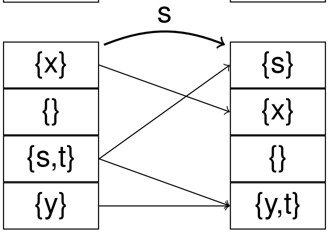
$$\gamma([\{x, y\}, \{\}, \{s, t\}, \{u\}]) = \{[x, y, s, t], [y, x, s, t], [x, y, s, u], \ldots\}$$

LRU: May-Analysis: Update

"Definite Cache Miss":



"Potential Cache Hit":



Why does *t* age in the second case?

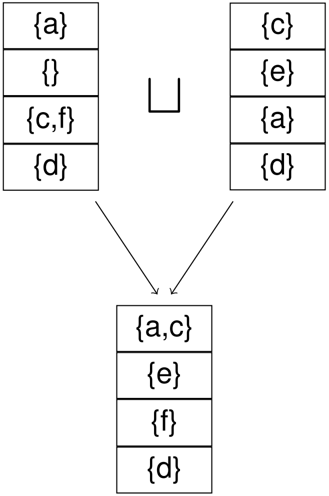
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Union + Minimal Age”



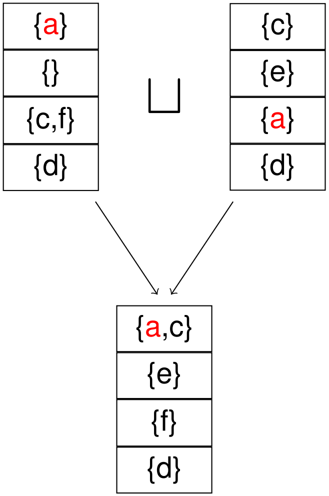
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"



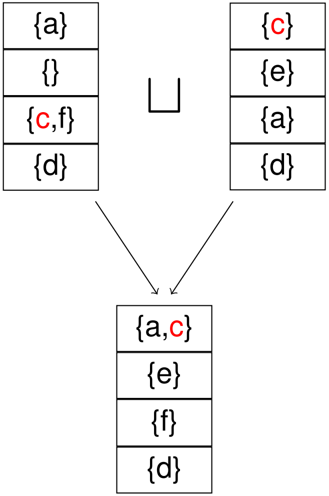
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Union + Minimal Age”



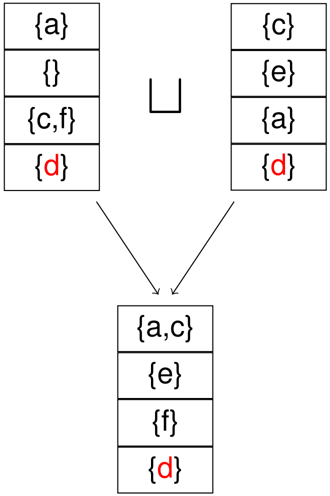
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

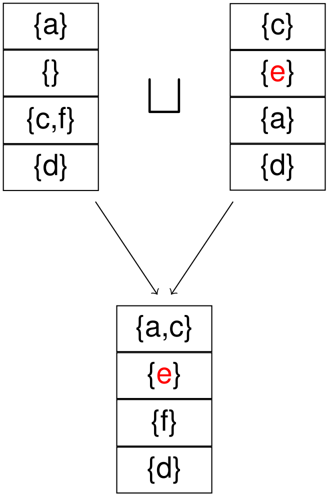
- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"



LRU: May-Analysis: Join

Need to combine information where control-flow merges.



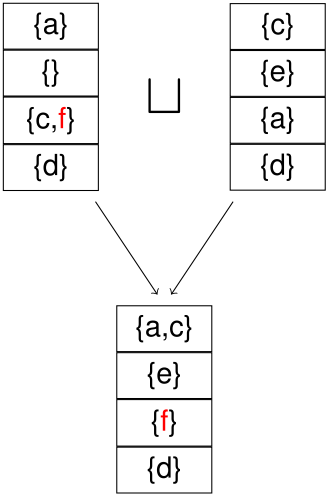
Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"

LRU: May-Analysis: Join

Need to combine information where control-flow merges.



Join should be conservative:

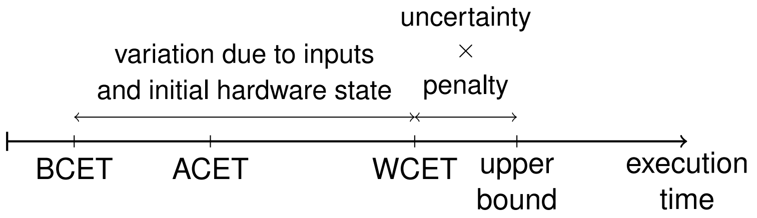
- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Union + Minimal Age”

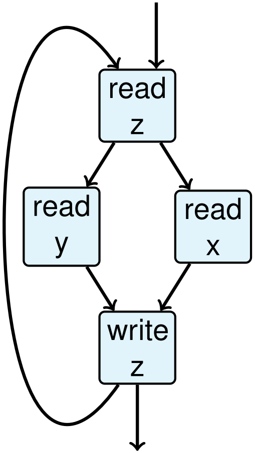
Outline

Uncertainty in WCET Analysis

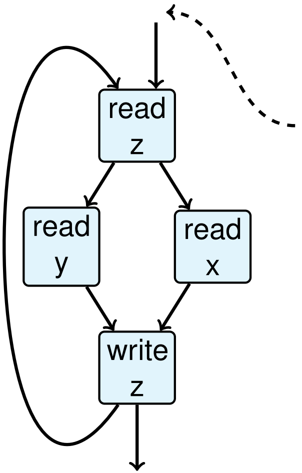
- Amount of uncertainty determines precision of WCET analysis
- Uncertainty in cache analysis depends on replacement policy



Uncertainty in Cache Analysis

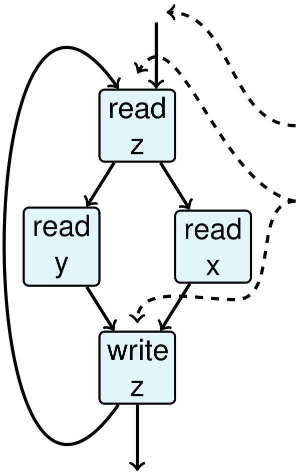


Uncertainty in Cache Analysis



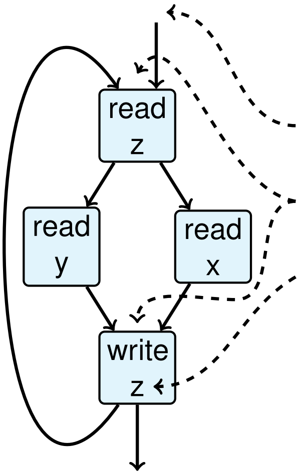
1. Initial cache contents unknown.

Uncertainty in Cache Analysis



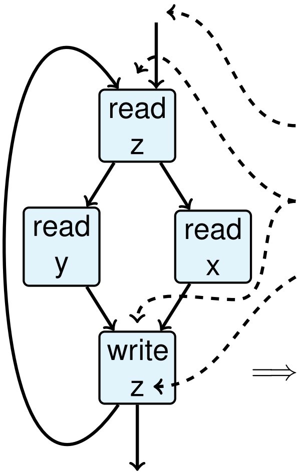
1. Initial cache contents unknown.
2. Need to combine information.

Uncertainty in Cache Analysis



1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of *z*.

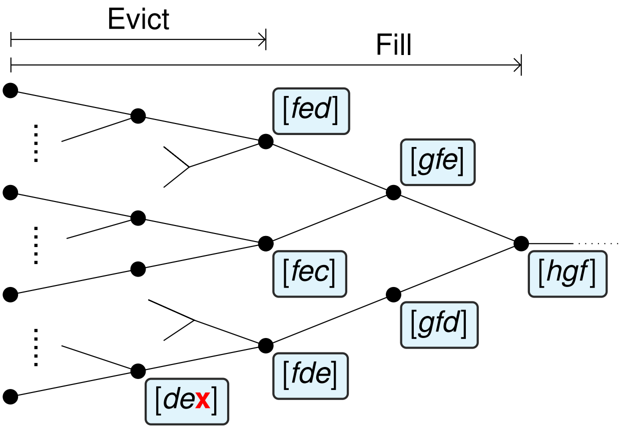
Uncertainty in Cache Analysis



1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of *z*.

$\Longrightarrow$ Amount of uncertainty determined by ability to recover information

Predictability Metrics



Sequence: $\langle$a, . . . , e, f, g, h$\rangle$

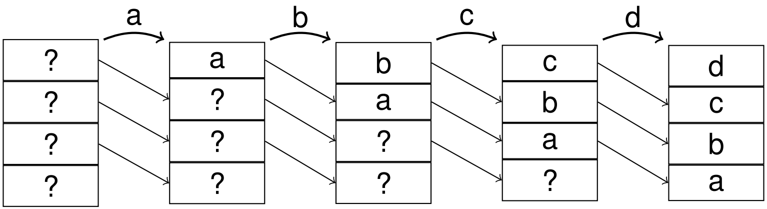
Meaning of Metrics

- Evict
  - Number of accesses to obtain *any may*-information.
  - I.e. when can an analysis predict any cache misses?
- Fill
  - Number of accesses to complete *may*- and *must*-information.
  - I.e. when can an analysis predict each access?

$\longrightarrow$ Evict and Fill bound the precision of *any* static cache analysis.
Can thus serve as a benchmark for analyses.

Evaluation of Least-Recently-Used

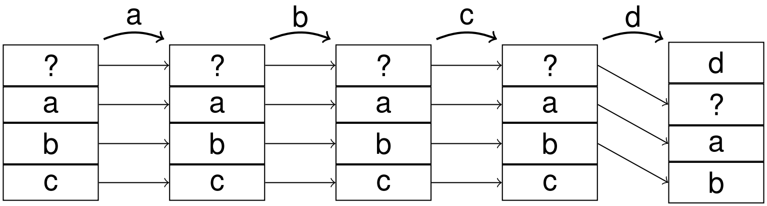
- LRU “forgets” about past quickly:
  - cares about most-recent access to each block only
  - order of previous accesses irrelevant



- In the example: Evict $=$ Fill $= 4$
- In general: Evict$(k) =$ Fill$(k) = k$, where k is the associativity of the cache

Evaluation of First-In First-Out (sketch)

- Like LRU in the miss-case
- But: “Ignores” hits



- In the worst-case $k-1$ hits and k misses: (k = associativity)
  $\longrightarrow \text{Evict}(k) = 2k-1$
- Another k accesses to obtain complete knowledge:
  $\longrightarrow \text{Fill}(k) = 3k-1$

Evaluation of Policies

Policy	Evict(k)	Fill(k)	Evict(8)	Fill(8)
LRU	k	k	8	8
FIFO	$2k-1$	$3k-1$	15	23
MRU	$2k-2$	$\infty/3k-4$	14	$\infty/20$
PLRU	$\frac{k}{2}\log_2 k+1$	$\frac{k}{2}\log_2 k+k-1$	13	19

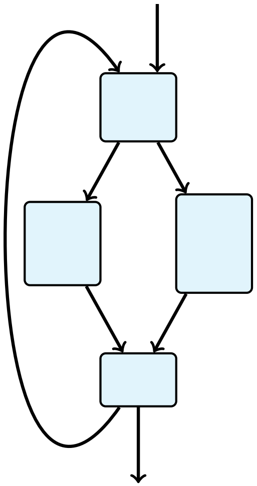
- LRU is optimal w.r.t. metrics.
- Other policies are much less predictable.

$\longrightarrow$ Use LRU if predictability is a concern.

- How to obtain *may*- and *must*-information within the given limits for other policies?

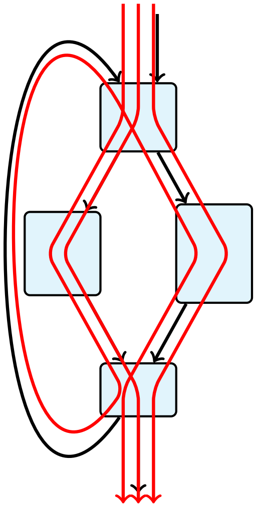
Measurement-Based Timing Analysis

- Run program on a number of inputs and initial states.
- Combine measurements for basic blocks to obtain WCET estimation.
- Sensitivity Analysis demonstrates this approach may be dramatically wrong.

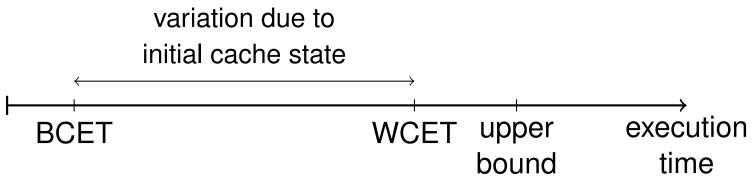


Measurement-Based Timing Analysis

- Run program on a number of inputs and initial states.
- Combine measurements for basic blocks to obtain WCET estimation.
- Sensitivity Analysis demonstrates this approach may be dramatically wrong.



Influence of Initial Cache State



Definition (Miss sensitivity)

Policy **P** is (k, c)-miss-sensitive if

$$m_{\mathbf{P}}(q, s) \leq k \cdot m_{\mathbf{P}}(q', s) + c$$

for all access sequences $s \in M^*$ and cache-set states $q, q' \in C^{\mathbf{P}}$.

Sensitivity Results

Policy	2	3	4	5	6	7	8
LRU	$1,2$	$1,3$	$1,4$	$1,5$	$1,6$	$1,7$	$1,8$
FIFO	$2,2$	$3,3$	$4,4$	$5,5$	$6,6$	$7,7$	$8,8$
PLRU	$1,2$	–	∞	–	–	–	∞
MRU	$1,2$	$3,4$	$5,6$	$7,8$	MEM	MEM	MEM

- LRU is optimal. Performance varies in the least possible way.
- For FIFO, PLRU, and MRU the number of misses may vary strongly.
- Case study based on simple model of execution time by Hennessy and Patterson (2003):
  WCET may be 3 times higher than a measured execution time for 4-way FIFO.

Caches: Summary

Cache Analysis for Least-Recently-Used

…efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

…requires context-sensitivity for precision.

Caches: Summary

Cache Analysis for Least-Recently-Used

. . . efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

. . . requires context-sensitivity for precision.

Predictability Metrics

. . . quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Caches: Summary

Cache Analysis for Least-Recently-Used

…efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

…requires context-sensitivity for precision.

Predictability Metrics

…quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Sensitivity Analysis

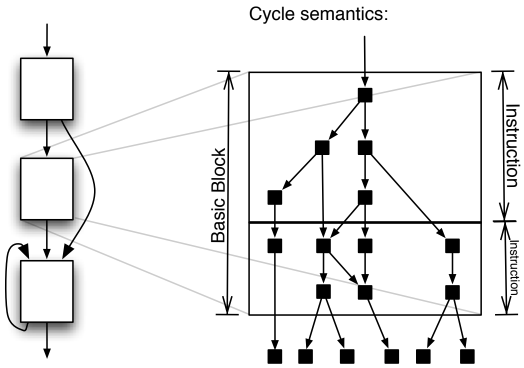
…determines the influence of initial state on cache performance.

Outline

Outline

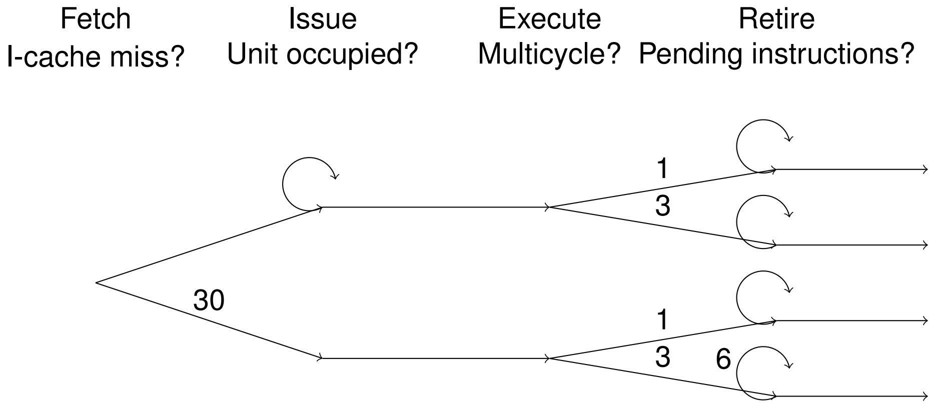
Pipeline analysis

Cyclewise evolution of processor model



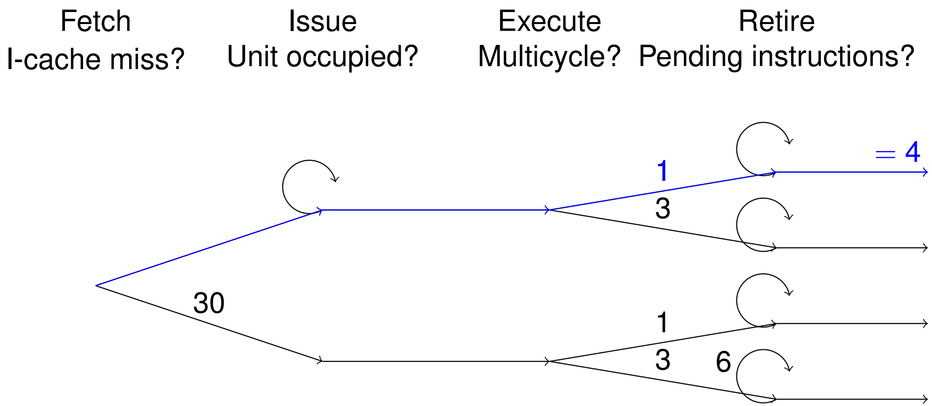
(Concrete) Instruction Execution

instruction MUL



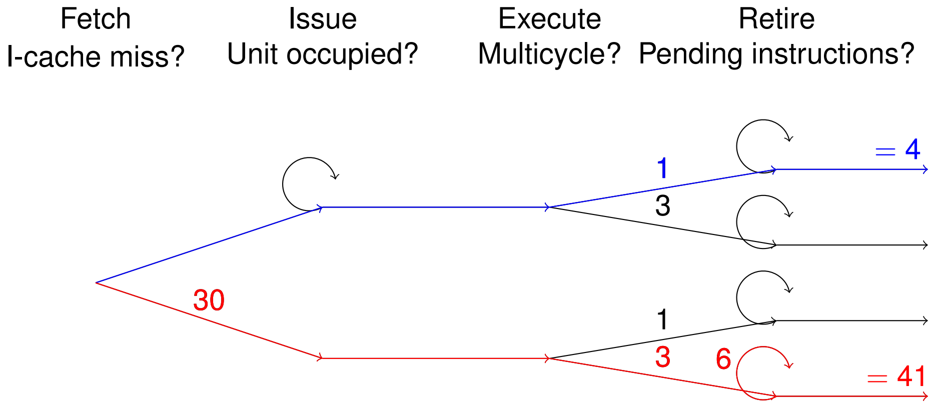
(Concrete) Instruction Execution

instruction MUL



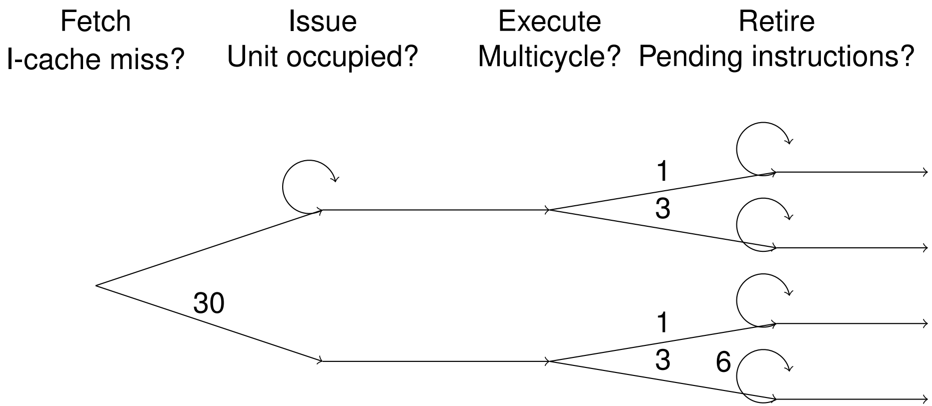
(Concrete) Instruction Execution

instruction MUL



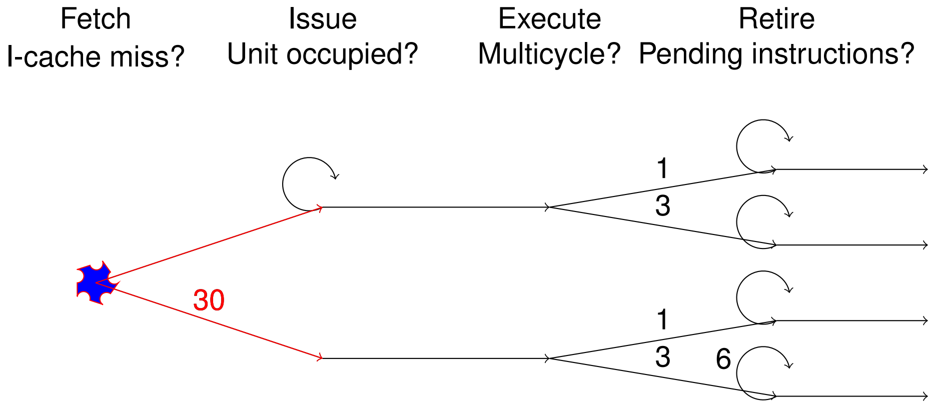
(Abstract) Instruction Execution

instruction MUL



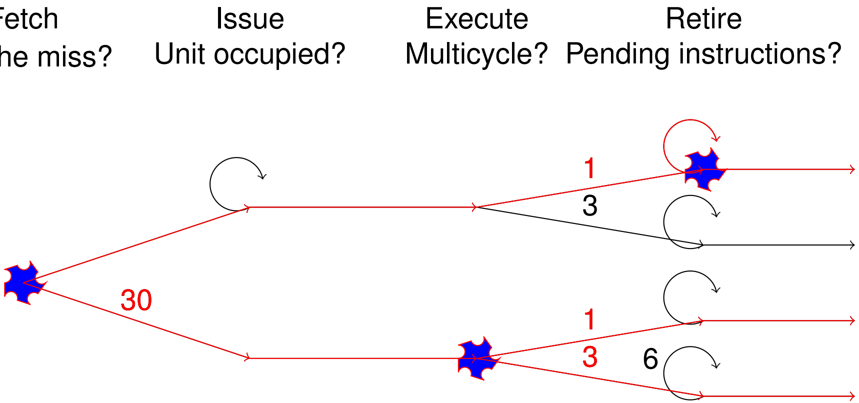
(Abstract) Instruction Execution

instruction MUL



(Abstract) Instruction Execution

instruction MUL



Characteristics of Pipeline analysis

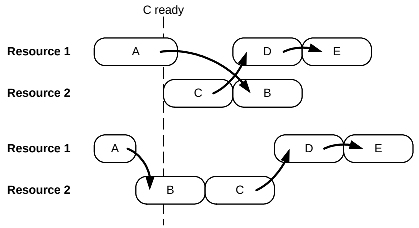
- Abstract Domain of Pipeline Analysis
  - Power set domain
    - Elements: sets of states of a state machine
  - Join: set union
- Pipeline Analysis
  - Manipulate sets of states of a state machine
  - Store sets of states to detect fixpoint
  - Forward state traversal
  - Exhaustively explore non-deterministic choice

Outline

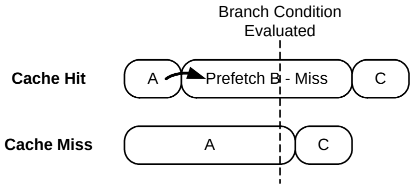
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Timing anomalies

- When local worst-case does not lead to the global worst-case



Scheduling anomaly.



Speculation anomaly.

Domino effects

- One event triggers another one which triggers another one...
- Unbounded effect of a timing accident

$\Rightarrow$ Always analyse all cases

$\Rightarrow$ Particularly, there doesn't exist an upper-bound on a delay is not representing the worst-case:

  $\rightarrow$ All possible delays to access a shared ressource

Outline

Classification of architectures

- *Timing compositional*
  - No timing anomalies
  - e. g., ARM7
- *Compositional with bounded effects*
  - Timing anomalies but no domino effects
  - e. g., TriCore (probably)
- *Non-compositional architectures*
  - Timing anomalies, domino effects
  - e. g., PPC 755

from Wilhelm et al.: Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems, IEEE TCAD, July 2009

Outline

Outline

Context

- Applications
  - AUTOSAR
  - IMA
- Behavior of the system
  - Compositionality:
    Composition of components
  - Composability:
    Behavior of each components should not change by the composition

Outline

Predictability of Multi-Core Architectures

PROMPT

Minimise sharing in multi-processor architectures:

- Interferences might be huge (bus contention, cache pollution)
- Huge overestimation when analysis is possible
  - Set of tasks that might be executed in parallel
  - Cache contents

Predictability of Multi-Core Architectures
PROMPT

Minimise sharing in multi-processor architectures:

- Interferences might be huge (bus contention, cache pollution)
- Huge overestimation when analysis is possible
  - Set of tasks that might be executed in parallel
  - Cache contents

PROMPT (PRedictability Of Multi-Processor Timing)

- Start with a generic, parameterisable architecture with predictable (fully timing compositional) cores
- Instantiate architecture for given set of applications, based on their resource requirements

Predictability of Multi-Core Architectures

Design Principles

- Simplification of individual components
- Elimination of interferences on shared resources:
  - Wherever it is not absolutely needed
  - Private resources for private uses
  - Shared resource for global state

Predictability of Multi-Core Architectures

Design Principles

- Simplification of individual components
- Elimination of interferences on shared resources:
  - Wherever it is not absolutely needed
  - Private resources for private uses
  - Shared resource for global state

$\Rightarrow$ Delays for accesses to the shared global state
  - Determination of delays, or
  - Cumulative analyses of WCET, bus arbiter and scheduling

Predictability of Multi-Core Architectures

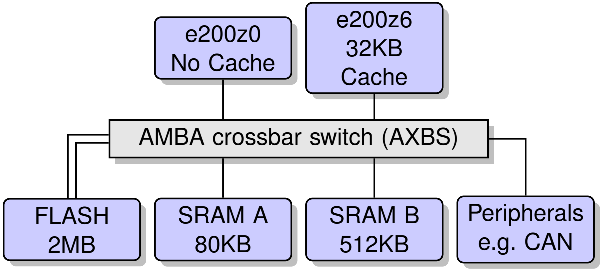
Design guidelines

single	Fully timing compositional architectures ▸ delay bounded by a constant: access to shared resources, preemptions
single	Disjoint instruction and data caches
single	Caches with LRU
multi	A shared bus protocol with bounded access delay
multi	Private caches
multi	Private memories, or, only share the lonely resources

Outline

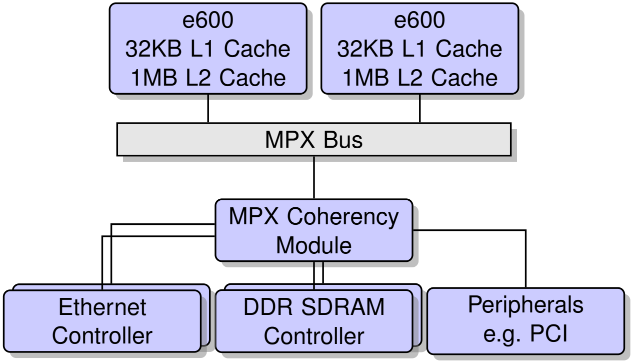
Smart configuration of existing multi-core

MPC5668G - An automotive processor



Smart configuration of existing multi-core

MPC8641D - An avionics processor



Summary

- Static timing analysis
  - Efficiency and precision
  - Strongly depends on the architecture
- Caches
  - predictabiliy and sensitivity metrics
  - LRU is the most predictable policy
- Timing analysis of multi-core
  - Hard but possible
  - Predictable multi-core: less complexity and more precise results
- Recommendations for the design of multi-core
  - Predictable single-core
  - Sharing only if needed

References

C. Ferdinand et al.: Cache Behavior Prediction by Abstract Interpretation. Science of Computer Programming 35(2): 163-189 (1999)
C. Ferdinand et al.: Reliable and Precise WCET Determination of a Real-Life Processor, EMSOFT 2001
R. Heckmann et al.: The Influence of Processor Architecture on the Design and the Results of WCET Tools, IEEE Proc. on Real-Time Systems, July 2003
St. Thesing et al.: An Abstract Interpretation-based Timing Validation of Hard Real-Time Avionics Software, IPDS 2003
L. Thiele, R. Wilhelm: Design for Timing Predictability, Real-Time Systems, Dec. 2004
R. Wilhelm: Determination of Execution Time Bounds, Embedded Systems Handbook, CRC Press, 2005
St. Thesing: Modeling a System Controller for Timing Analysis, EMSOFT 2006
J. Reineke et al.: Predictability of Cache Replacement Policies, Real-Time Systems, Springer, 2007
R. Wilhelm et al.:The Determination of Worst-Case Execution Times - Overview of the Methods and Survey of Tools. ACM Transactions on Embedded Computing Systems (TECS) 7(3), 2008.
R.Wilhelm et al.: Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems, IEEE TCAD, July 2009
R. Wilhelm et al.: Designing Predictable Multicore Architectures for Avionics and Automotive Systems, RePP Workshop, Grenoble, Oct. 2009

Tutorial ISCA 2010

Architectural Aspects of Deriving Performance Guarantees

Part II
Timing analysis of parallel and distributed embedded systems

© Lothar Thiele

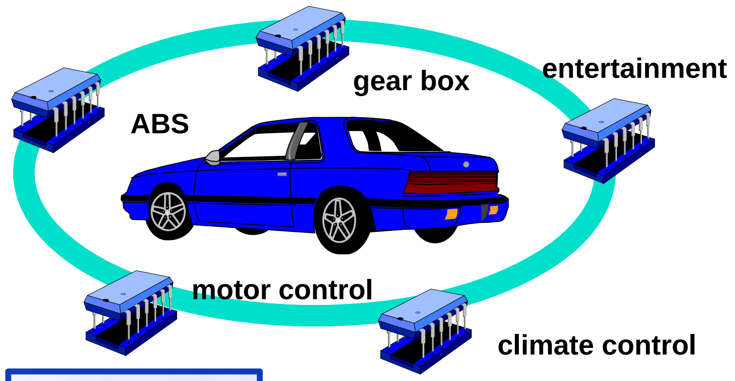
Contents

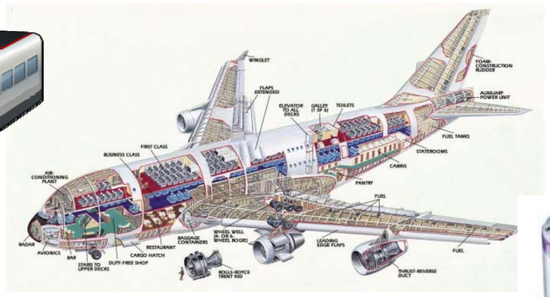
- ***Drivers***
- Compositional Analysis
  - Overview
  - Real-Time Calculus
  - Artificial Example
- Architectural Interactions
  - Shared Resources in Multicore Systems
  - Compilation for Multiprocessors
- Challenges

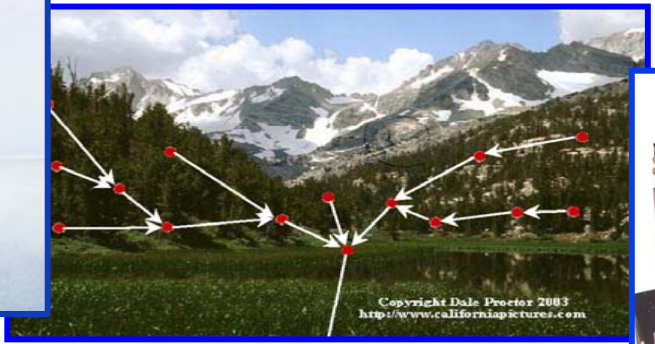
Drivers

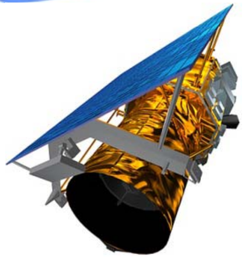
Embedded Systems

Information processing system that is physically embedded within a larger system

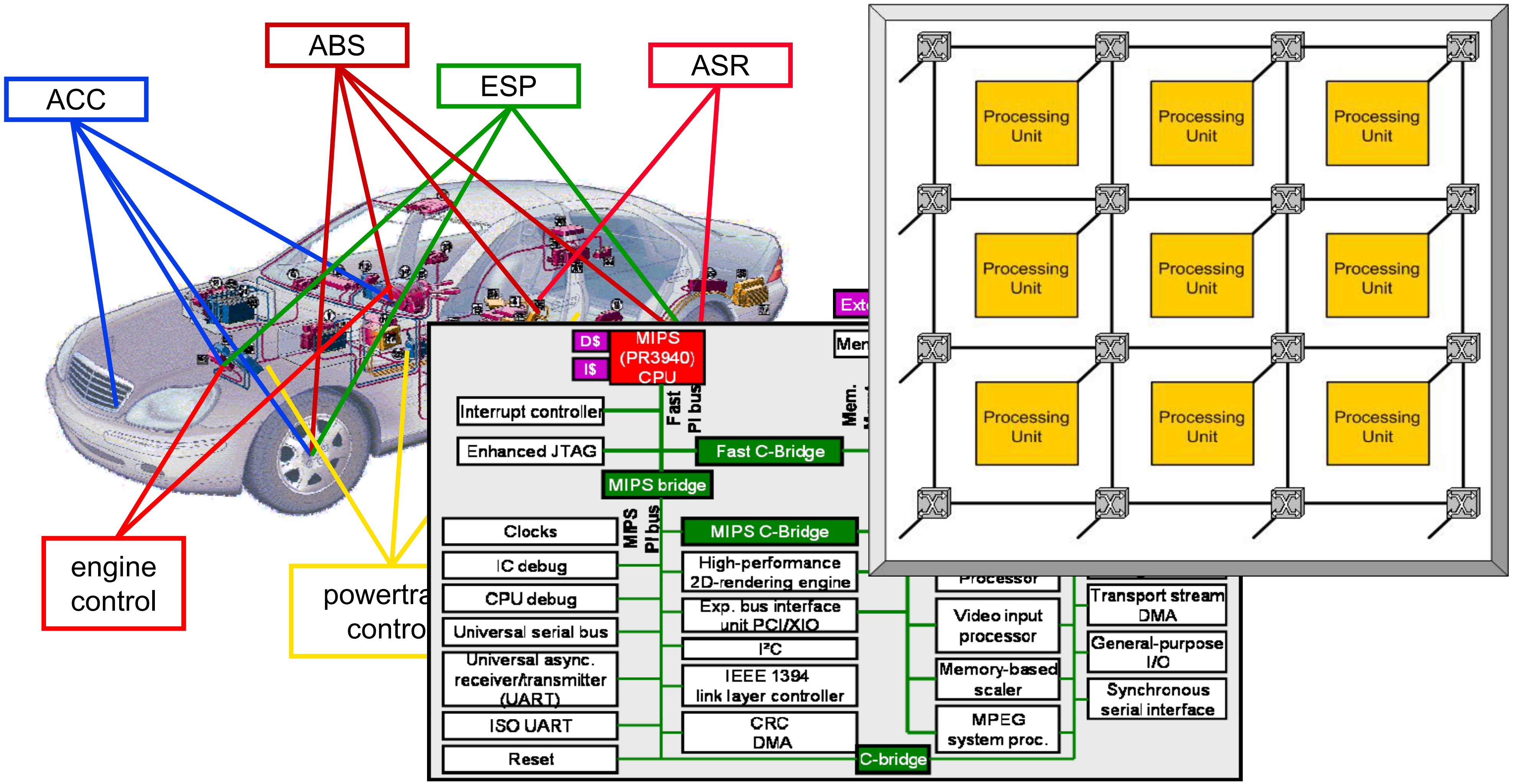





Copyright Dale Proctor 2003
http://www.californiapictures.com

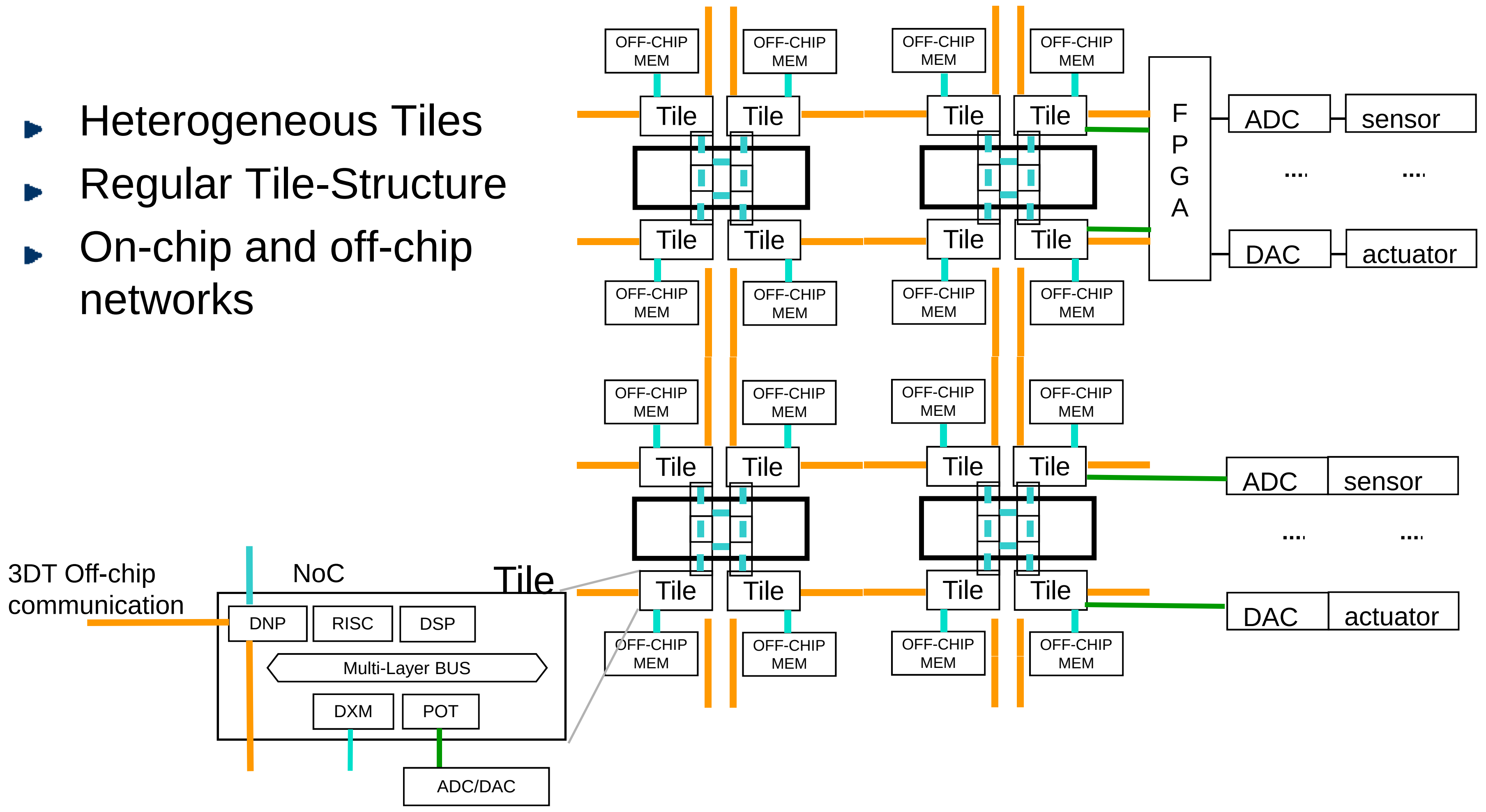


Target Platforms



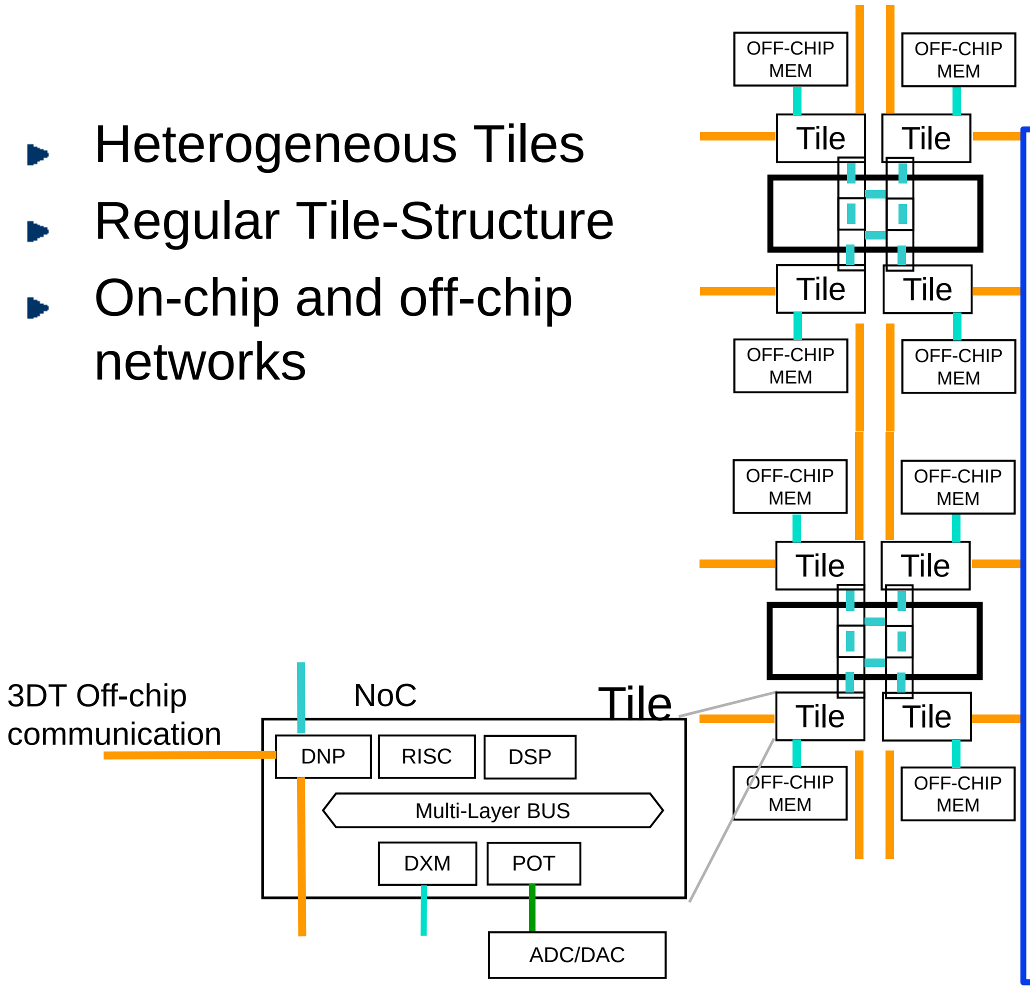
A Sample HW Architecture (EU-SHAPES)

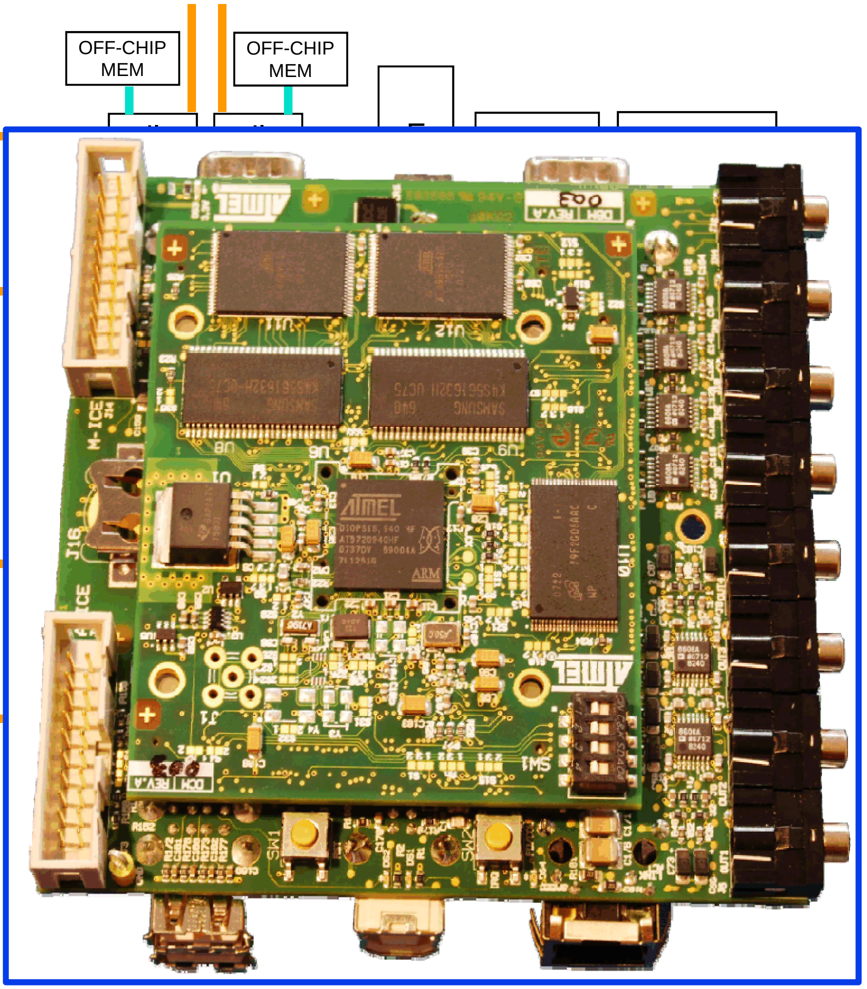
- Heterogeneous Tiles
- Regular Tile-Structure
- On-chip and off-chip networks



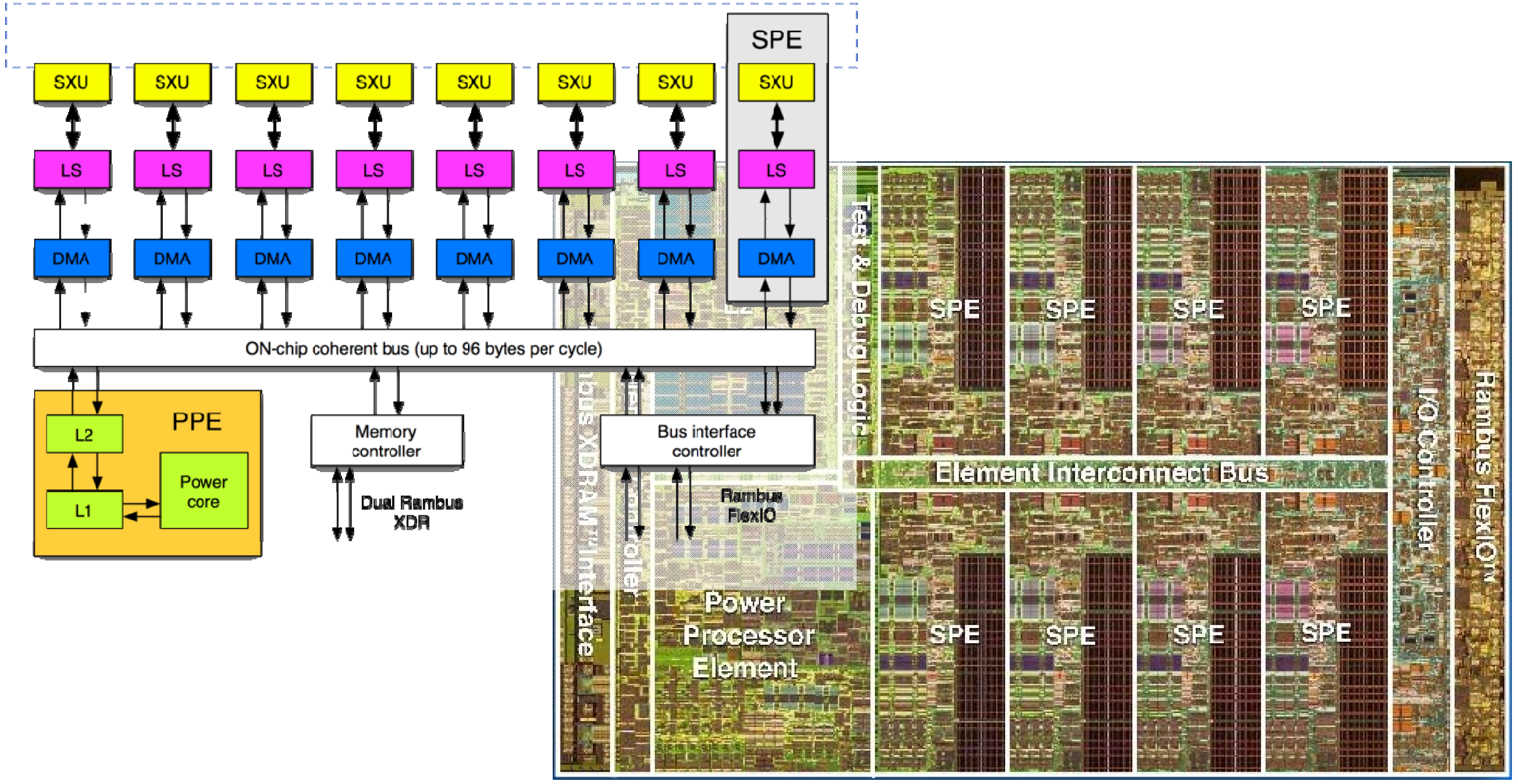
A Sample HW Architecture (EU-SHAPES)

- Heterogeneous Tiles
- Regular Tile-Structure
- On-chip and off-chip networks

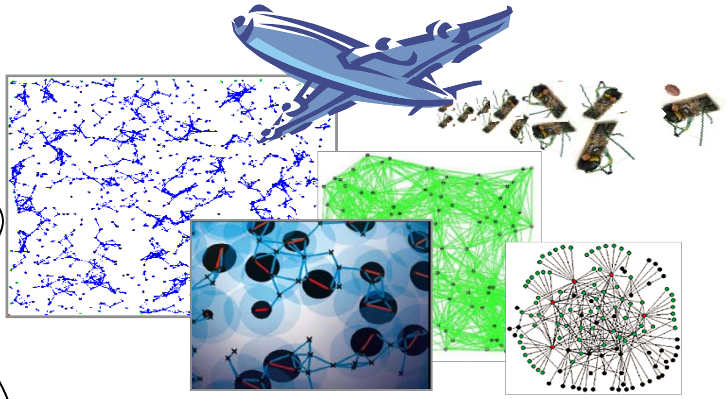




IBM Cell Processor



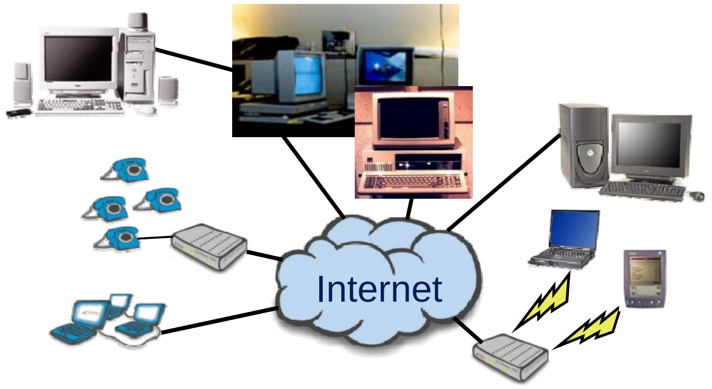
Big Picture



Centralized Systems

Networked Systems

Large-scale Distributed Systems



New Applications and System Paradigms

Contents

Compositional Analysis

- Overview -

Analysis and Design

Embedded System =

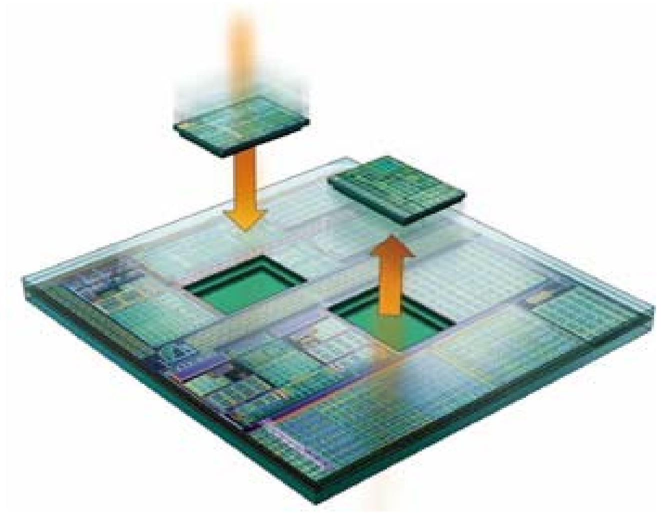
Computation + Communication + Resource Interaction

Analysis:

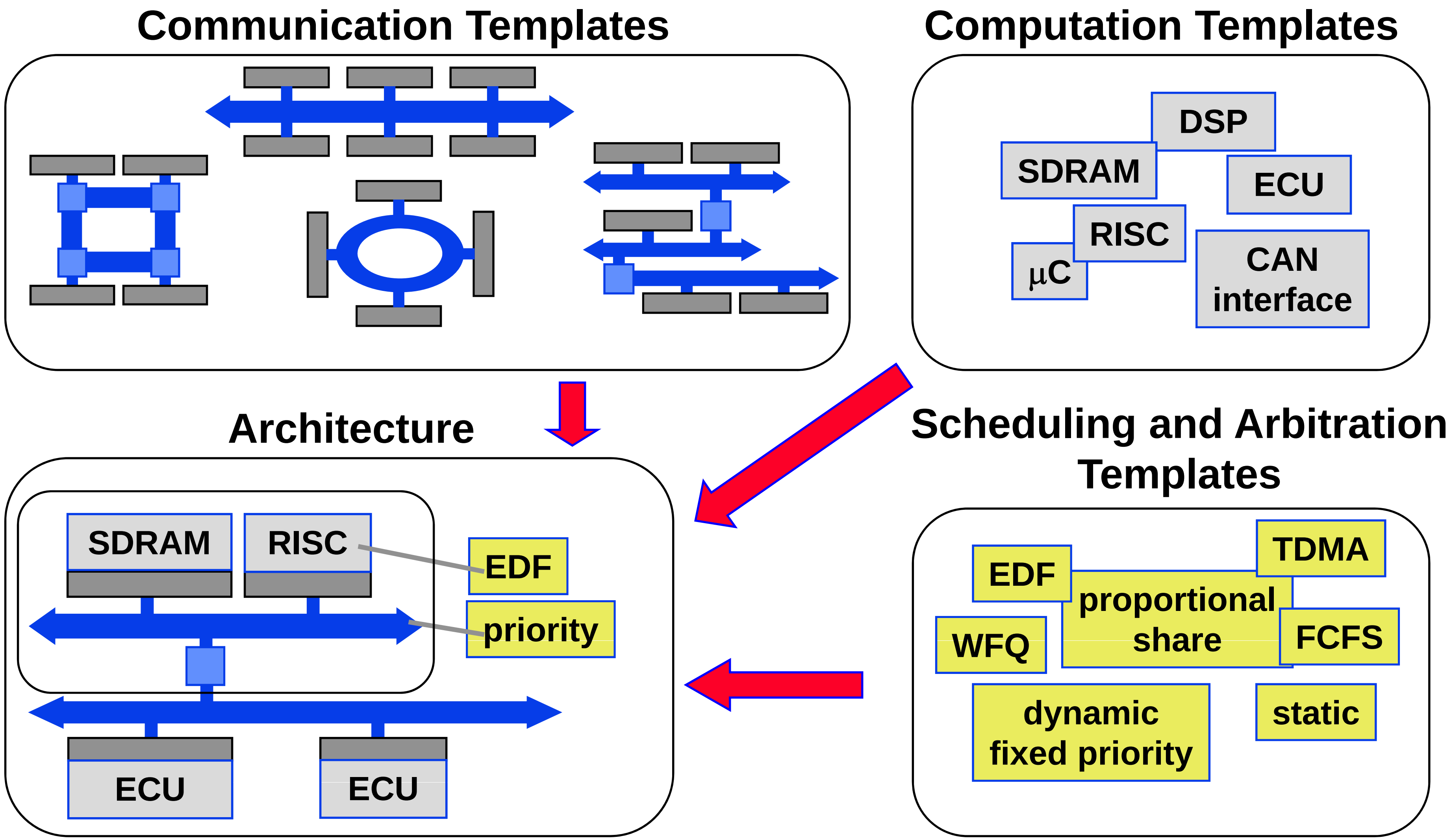
Infer system properties from subsystem properties.

Design:

Build a system from subsystems while meeting requirements.



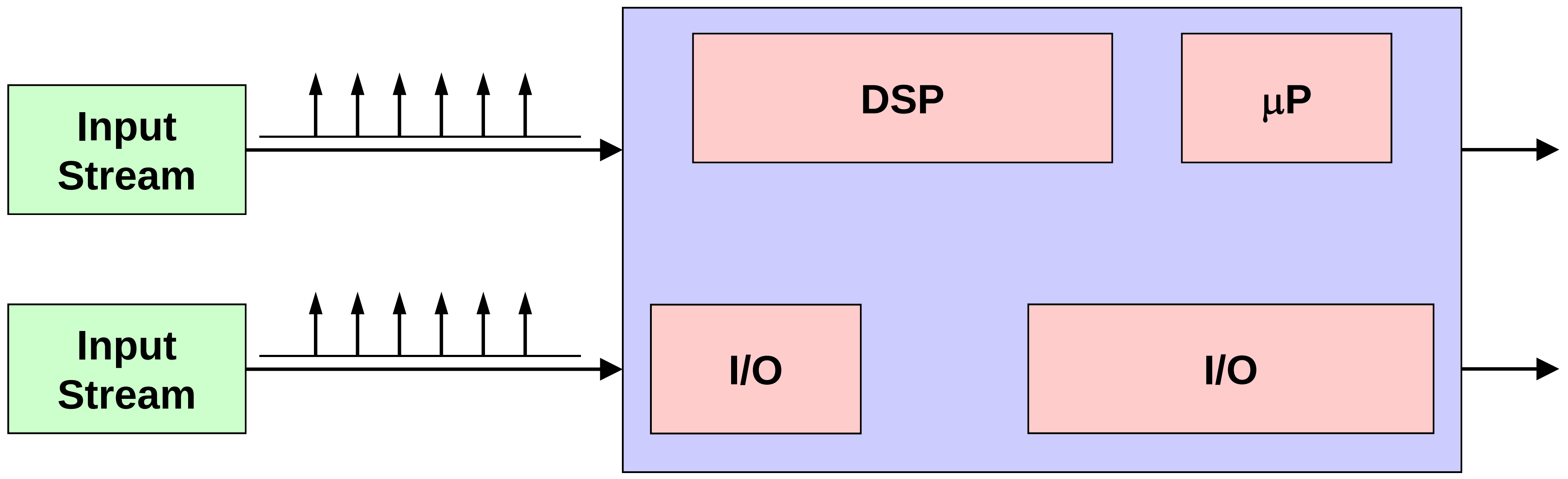
System Composition



Why Performance Analysis ?

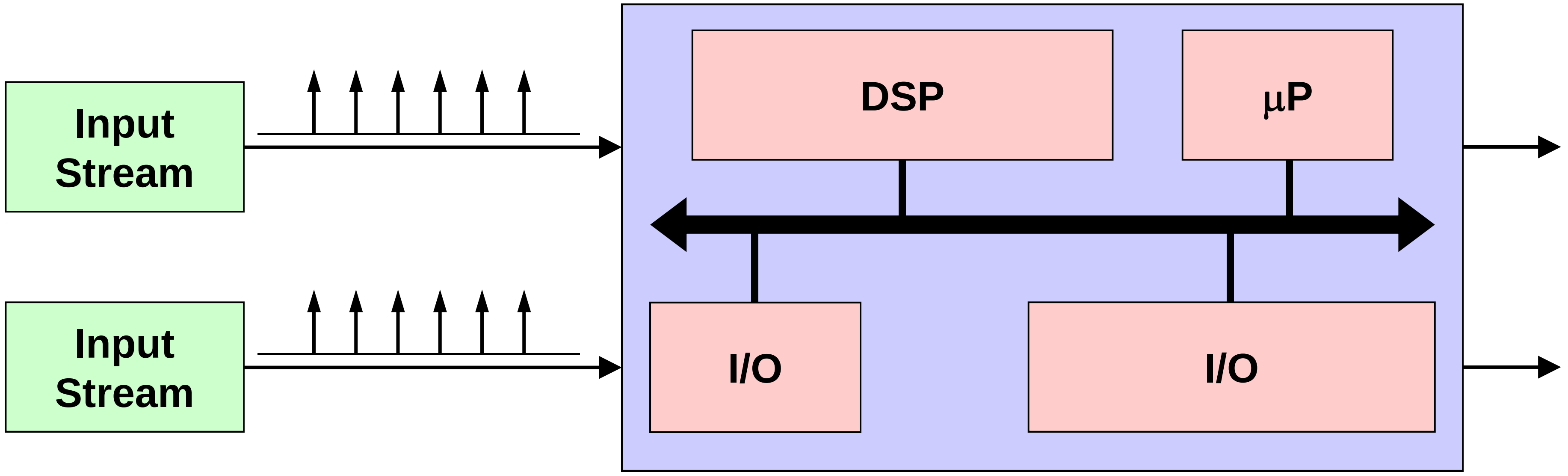
- Prerequisite for ***design space exploration (design decisions and optimization)***
  - part of the feedback cycle
  - get inside into design characteristics and bottlenecks
  - support early design decisions
- Design ***validation***
  - verify system properties
  - used at various design stages from early design until final implementation

Distributed Embedded System



Computational Resources ...

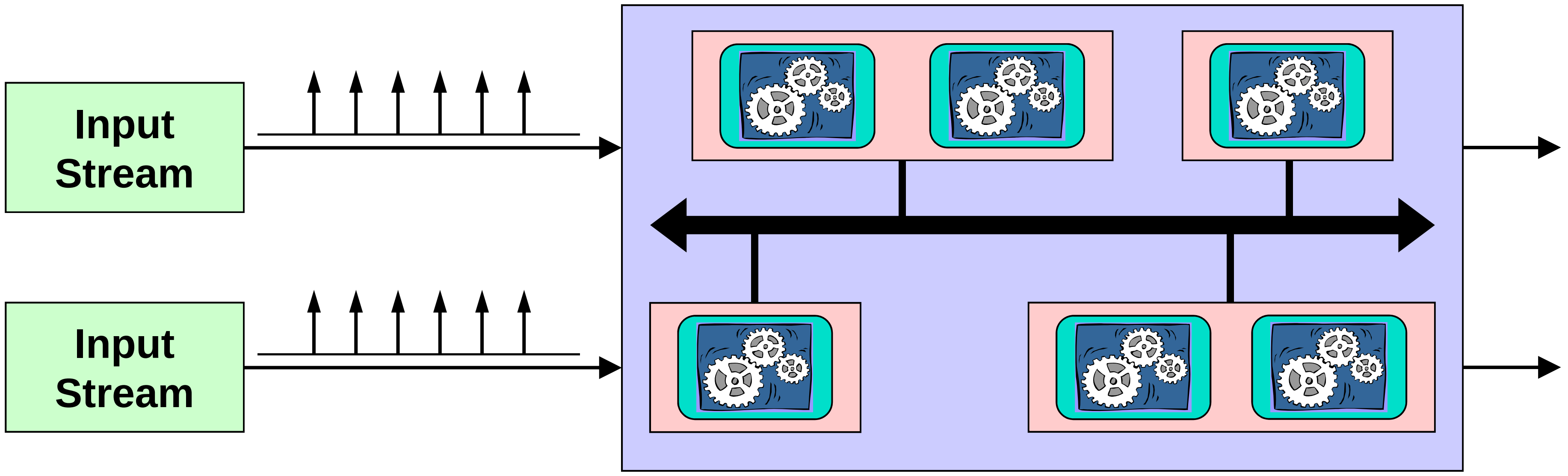
Distributed Embedded System



Computational Resources ...

... Communication Resources ...

Distributed Embedded System



Computational Resources ...

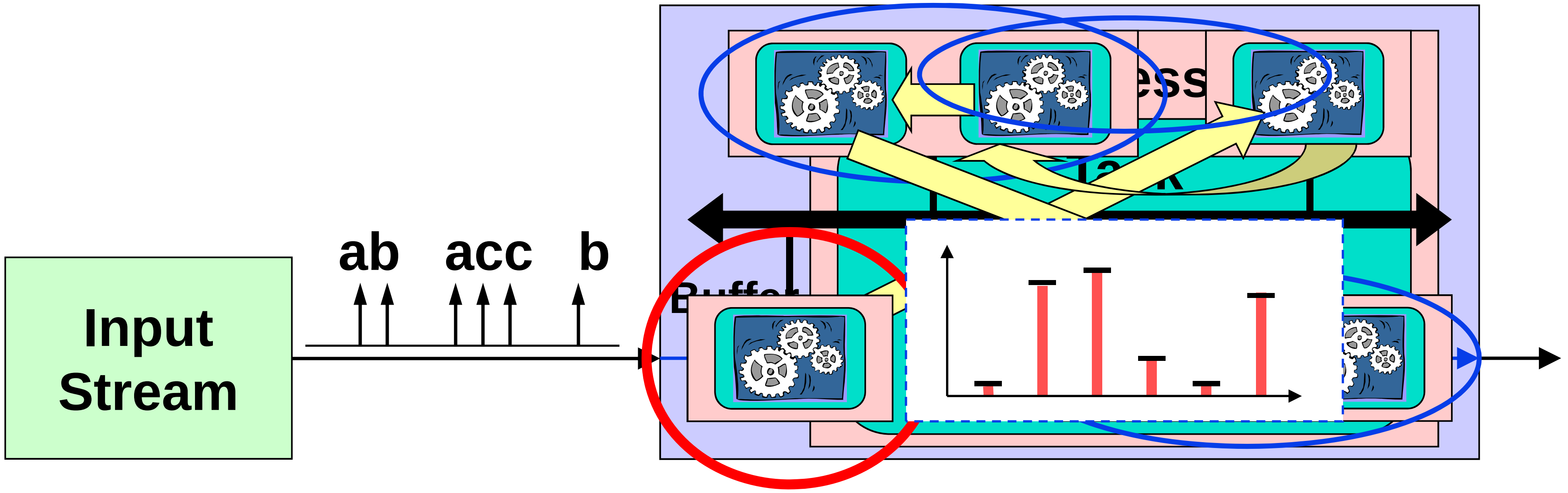
... Communication Resources ...

... Tasks

Why Is Evaluation Difficult ?

- ***Non-determinism:***
  - uncertain system environment, e.g. load scenarios
  - (non-deterministic) computations in processing nodes
- ***Interference:***
  - *sharing* exclusive resources (scheduling and arbitration)
  - interaction between *resource types*: exclusive (computation, communication) and shared (energy)
- ***Long-term dependencies***
  - *resource feedback*: internal data streams interact on exclusive resources which in turn change stream characteristics

Difficulties



Task Communication

Task Scheduling

Complex Input:

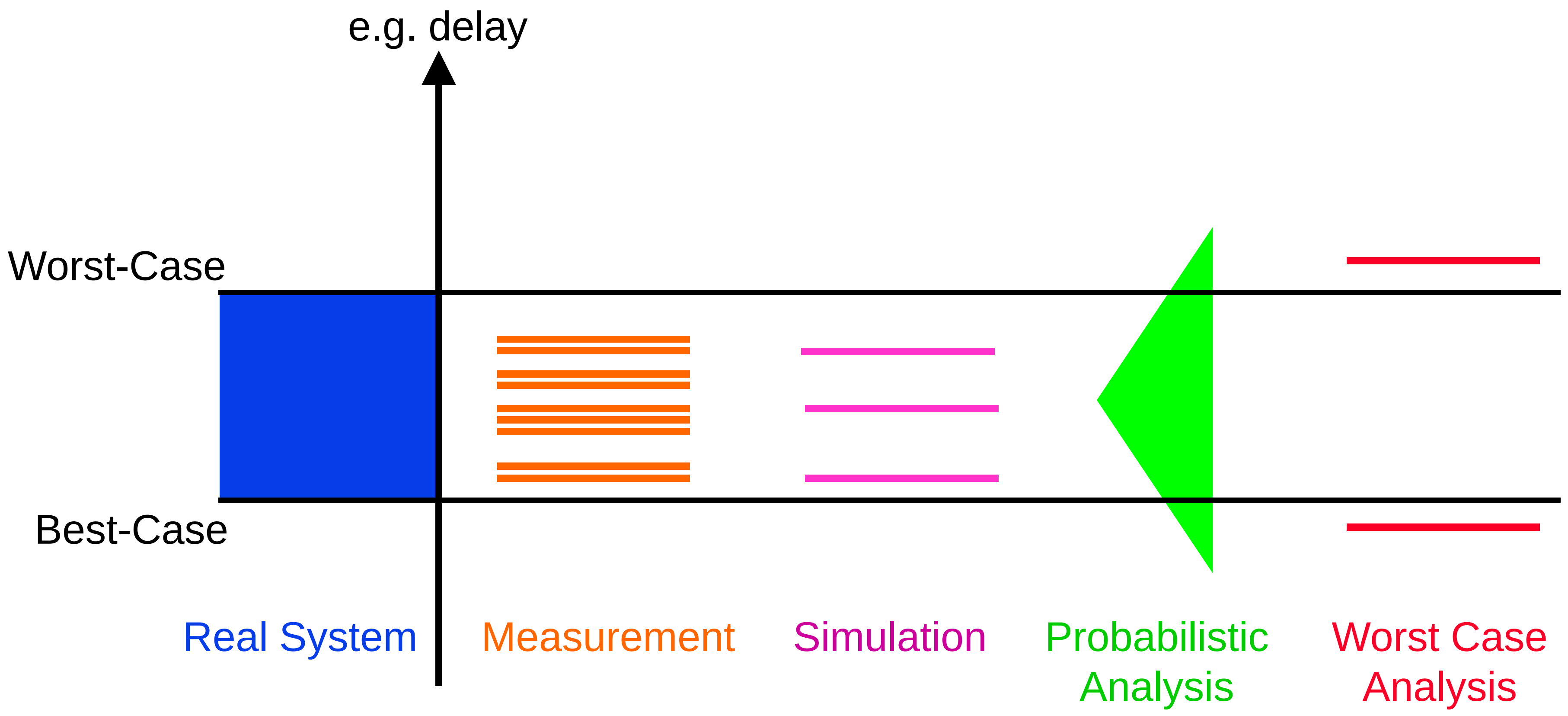
- Timing (jitter, bursts, ...)
- Different Event Types

Variable Resource Availability

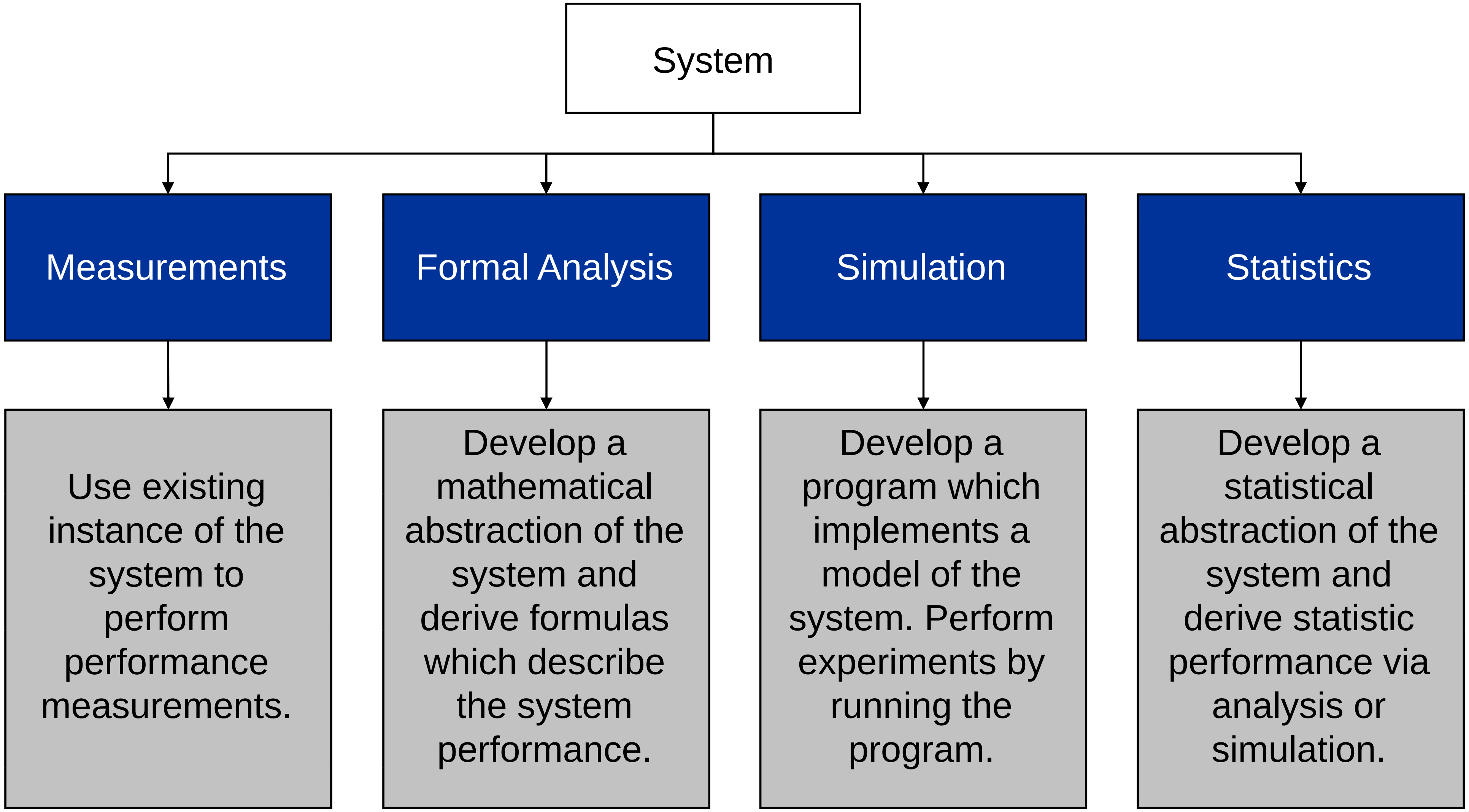
Variable Execution Demand

- Input (different event types)
- Internal State (Program, Cache, ...)

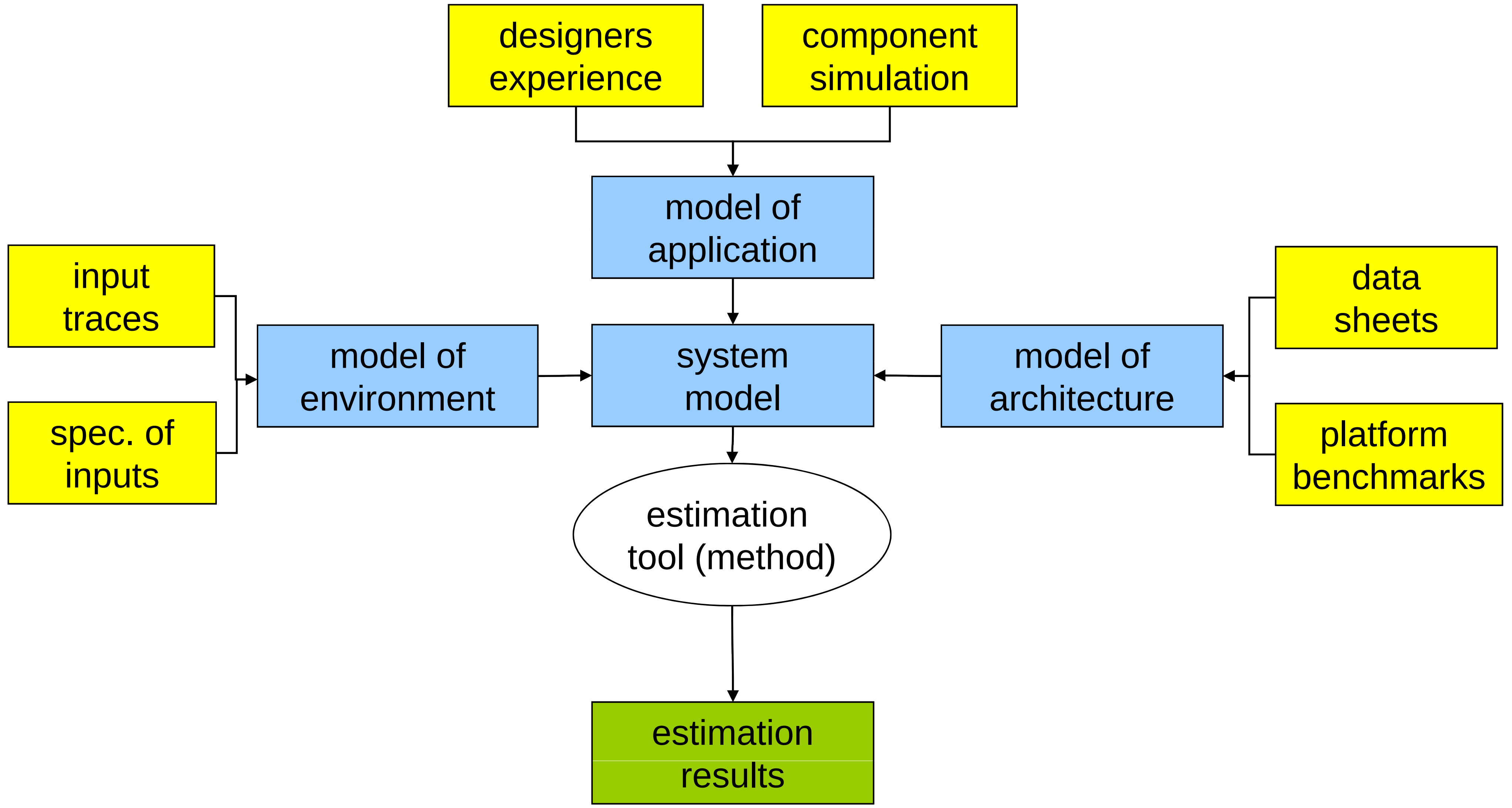
System-Level Evaluation Methods



Overview


System
Measurements
Formal Analysis
Simulation
Statistics
Use existing instance of the system to perform performance measurements.
Develop a mathematical abstraction of the system and derive formulas which describe the system performance.
Develop a program which implements a model of the system. Perform experiments by running the program.
Develop a statistical abstraction of the system and derive statistic performance via analysis or simulation.

Performance Estimation Methods



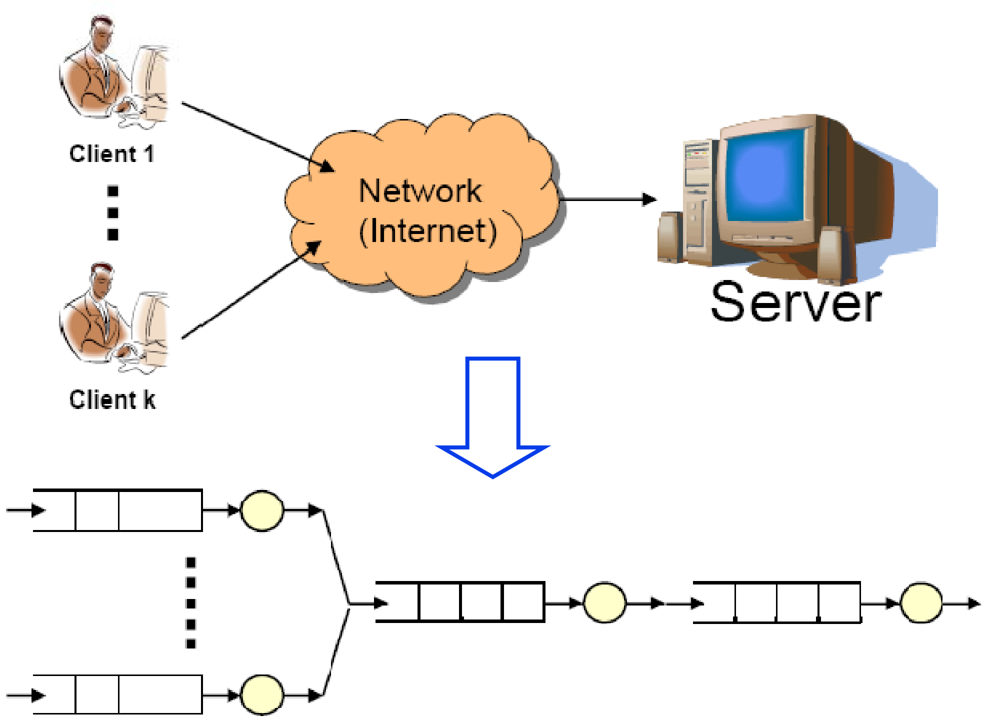
Formal Analysis - Dynamic Models

- Combination between
  - **Static models** possibly extended by non-determinism in run-time and event processing
  - **Dynamic models** for describing e.g. resource sharing mechanisms (scheduling and arbitration).

- Existing approaches
  - ***Classical real-time scheduling*** theory
  - ***Stochastic queuing theory*** (statistical bounds)
  - ***Non-deterministic queuing theory (***worst case/best case behavior)

Example - Queuing Systems

- ***Example:*** clients request some service from a server over a network.

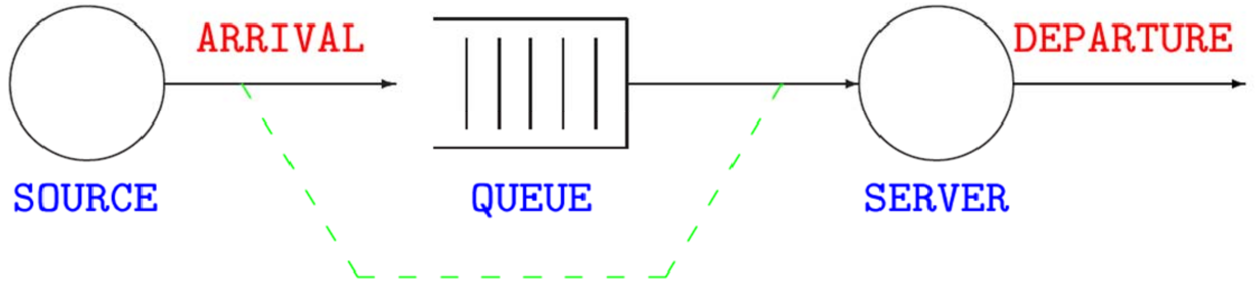


Stochastic Models - Queuing Systems

- A ***queuing system*** is described by
  - Arrival rate
  - Service mechanism
  - Queuing discipline

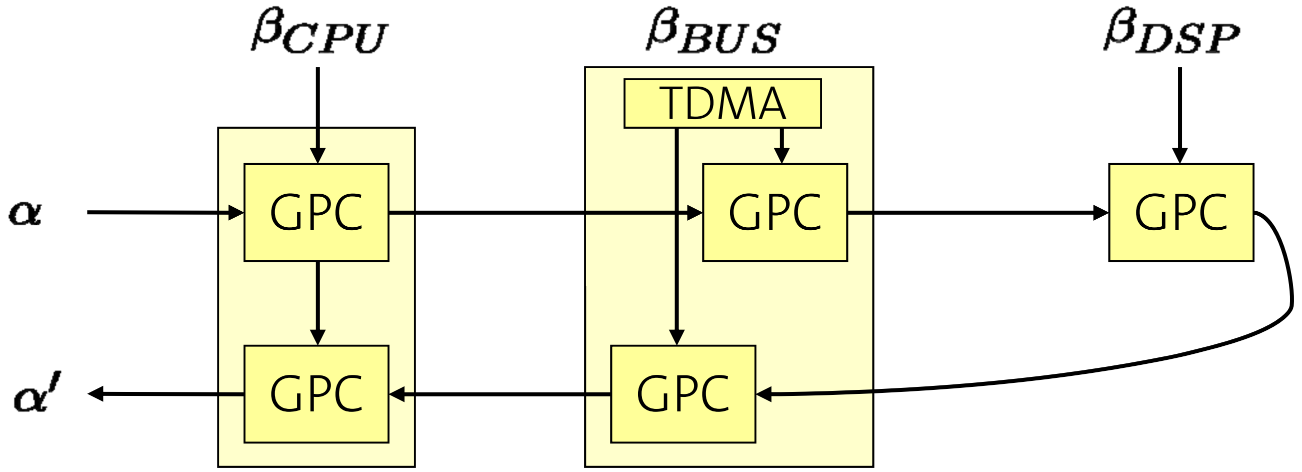
- ***Performance measures***
  - average delay in queue
  - time-average number of customers in queue.
  - proportion of time server is busy

The classical M/M/1 queuing system:
(M = Markovian (exp.) distribution)



Nondeterministic Models - Queuing Systems

- A ***queuing system*** is described by
  - Arrival function (bounds on arrival times)
  - Service functions (bounds on server behavior)
  - Resource interaction
- ***Performance measures***
  - worst case delay in queue
  - worst-case number of customers in queue.
  - worst-case and best-case end-to-end delay in the system



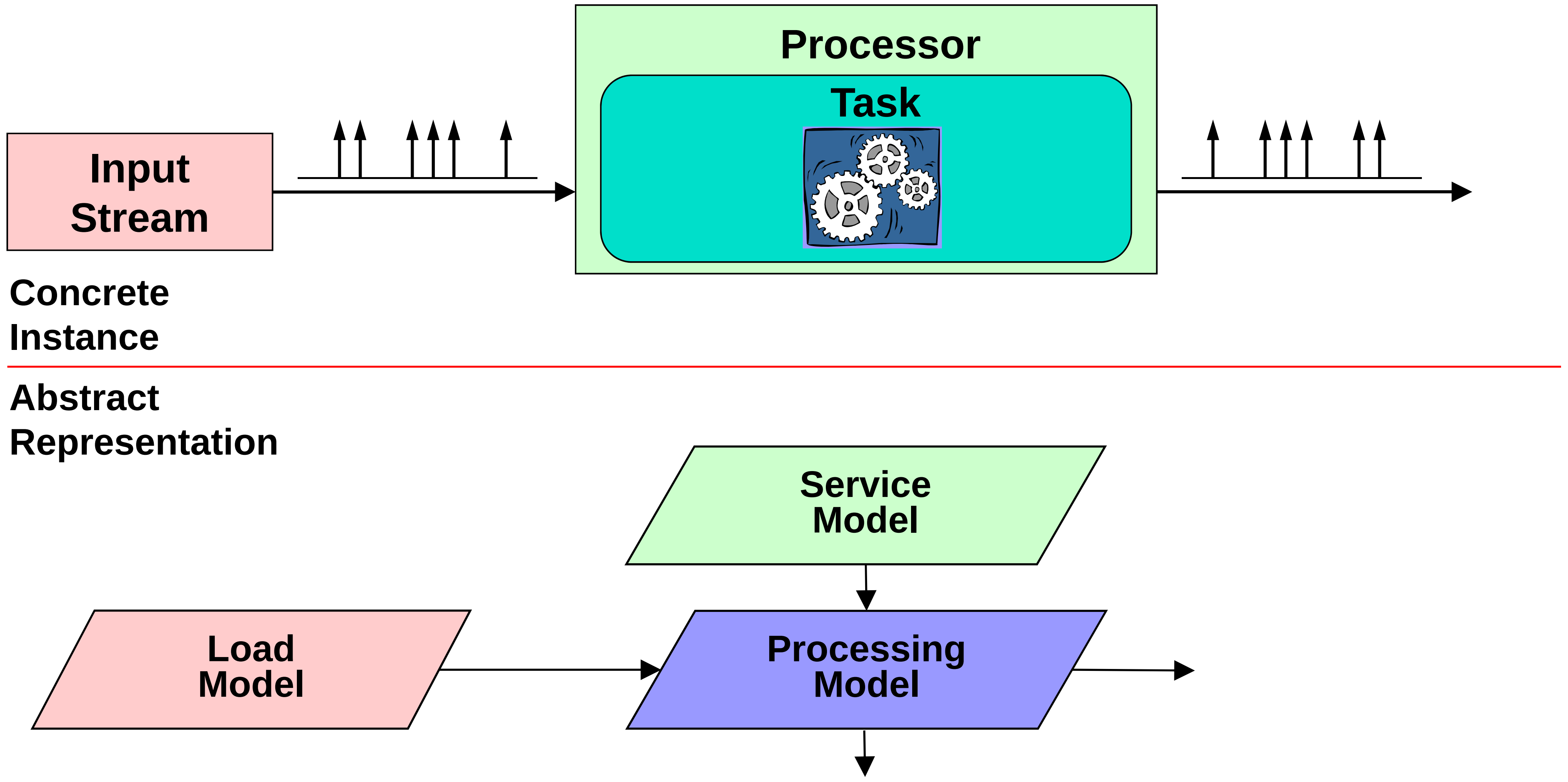
Compositional Analysis

- Real-Time Calculus -

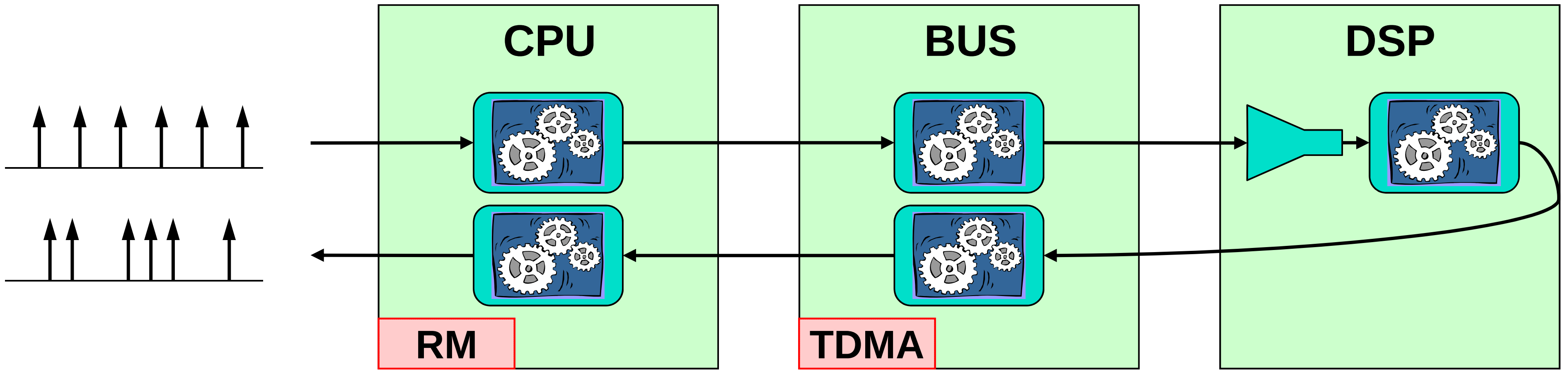
Network/Real-time Calculus Methods

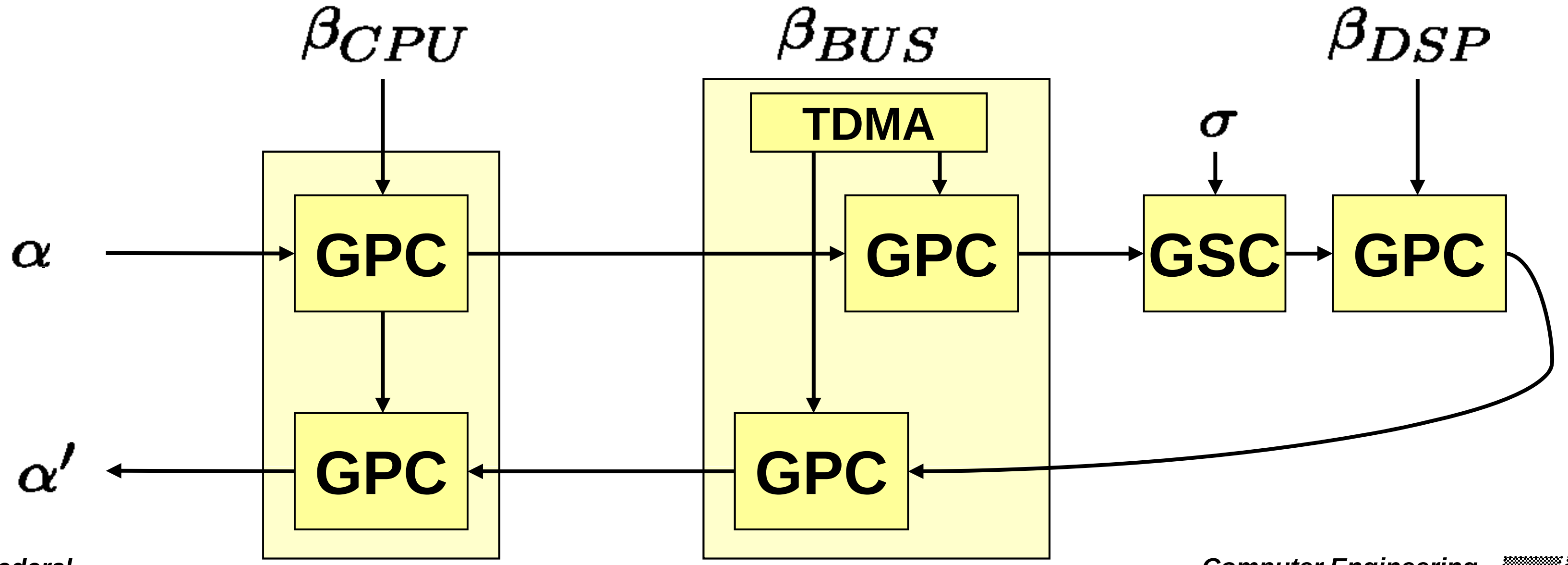
- ***Advantages***
  - More powerful abstraction than “classical” real-time analysis
  - Resources are first-class citizens of the method
  - *Allows composition in terms of (a) tasks, (b) streams, (c) resources, (d) sharing strategies.*
- ***Disadvantages***
  - Needs some effort to understand and implement
  - Extension to new arbitration schemes not always simple

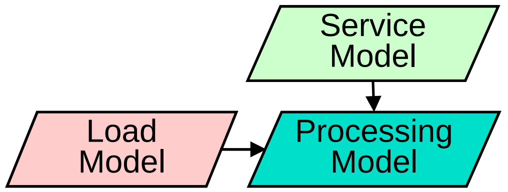
Abstract Models for Performance Analysis



Modular System Composition



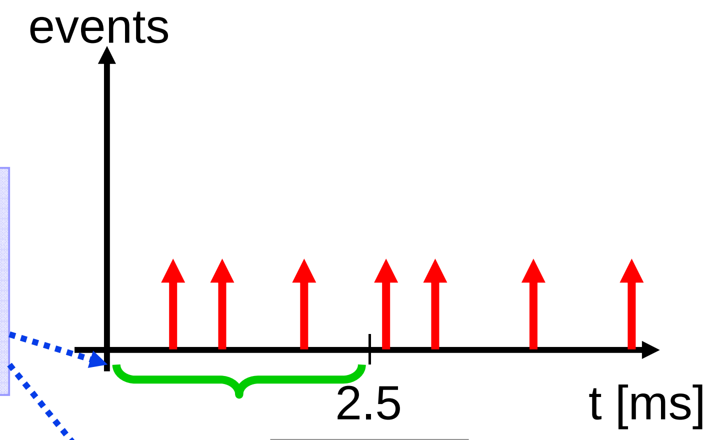




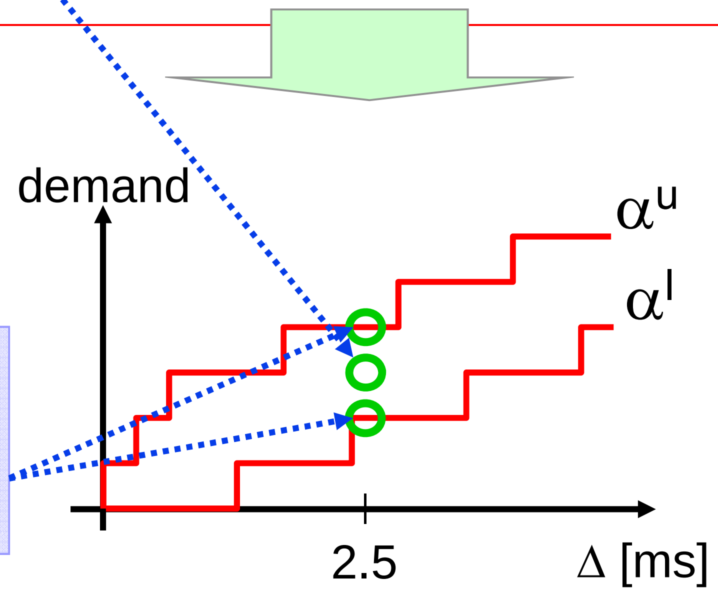
Load Model (Environment)

Event Stream

number of events in
in t=[0 .. 2.5] ms



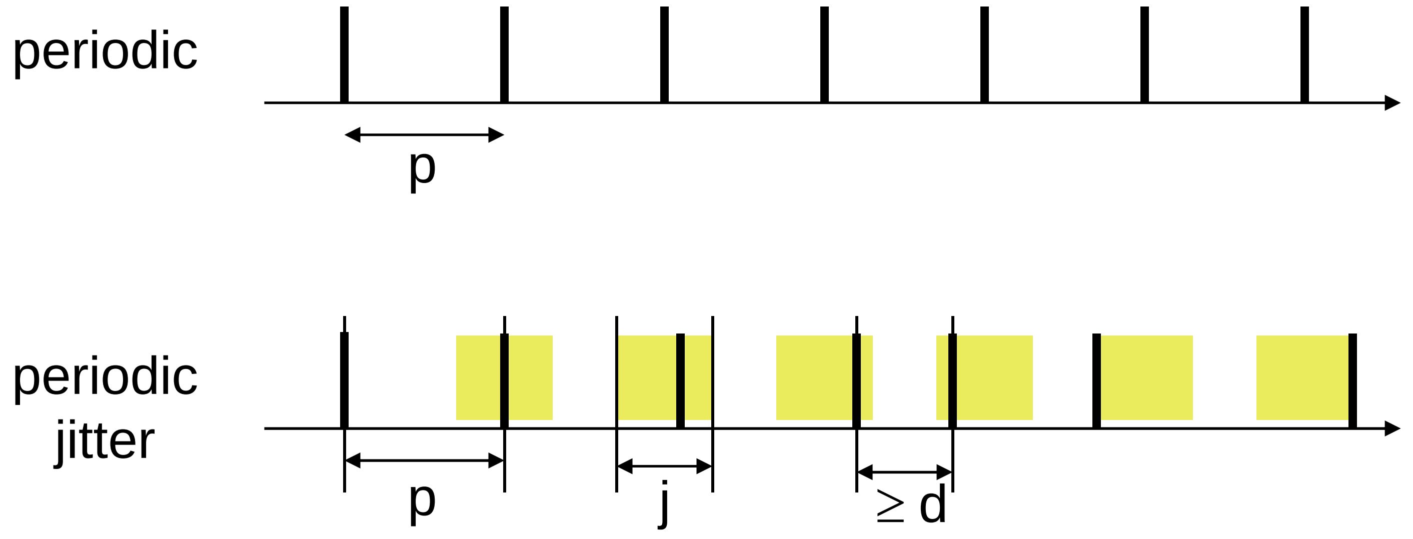
Arrival Curve α



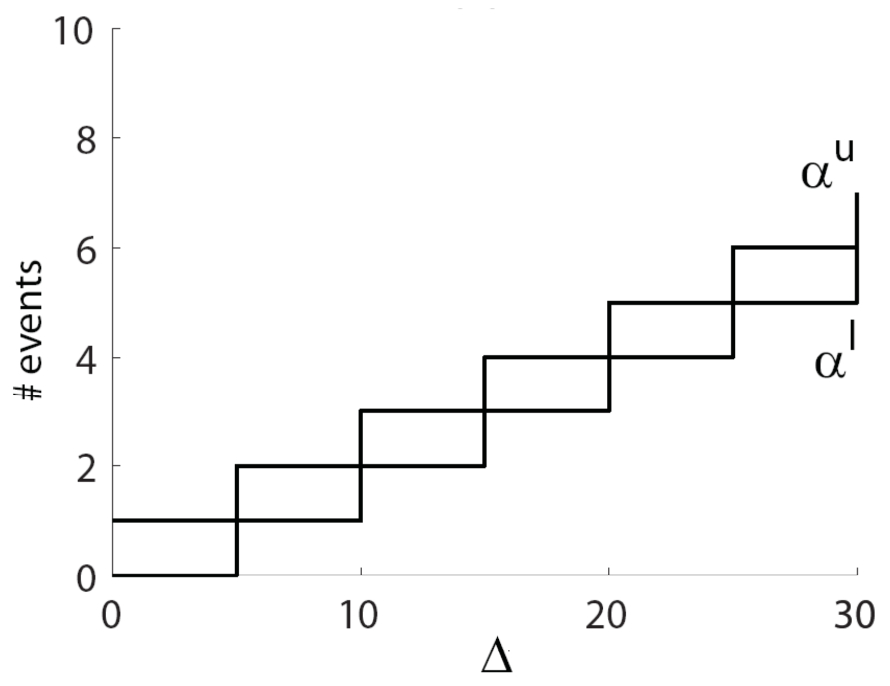
maximum / minimum
arriving demand in *any*
interval of length 2.5 ms

Example 1: Periodic with Jitter

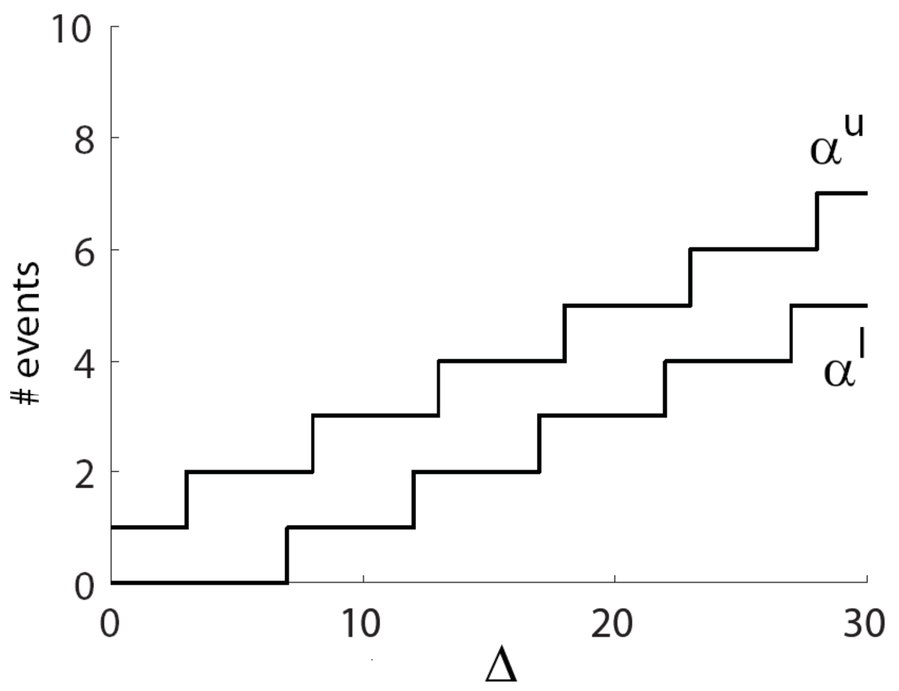
- A ***common event pattern*** that is used in literature can be specified by the parameter triple (*p, j, d*), where *p* denotes the period, *j* the jitter, and *d* the minimum inter-arrival distance of events in the modeled stream.



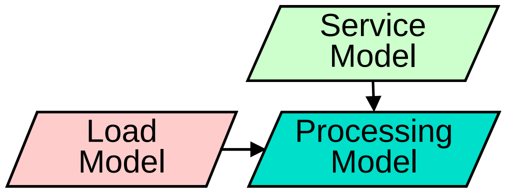
Example 1: Periodic with Jitter



periodic



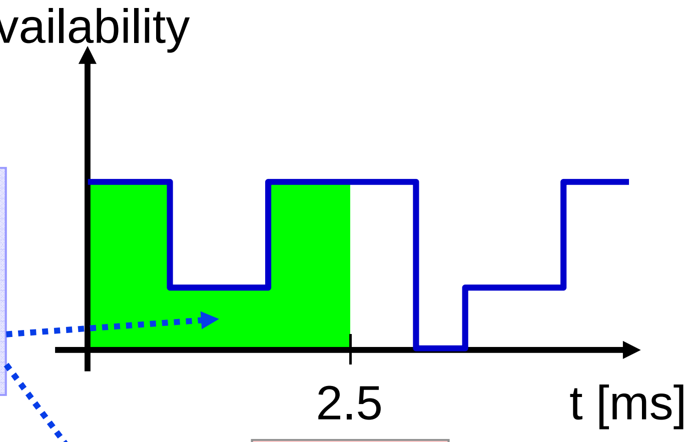
periodic with jitter



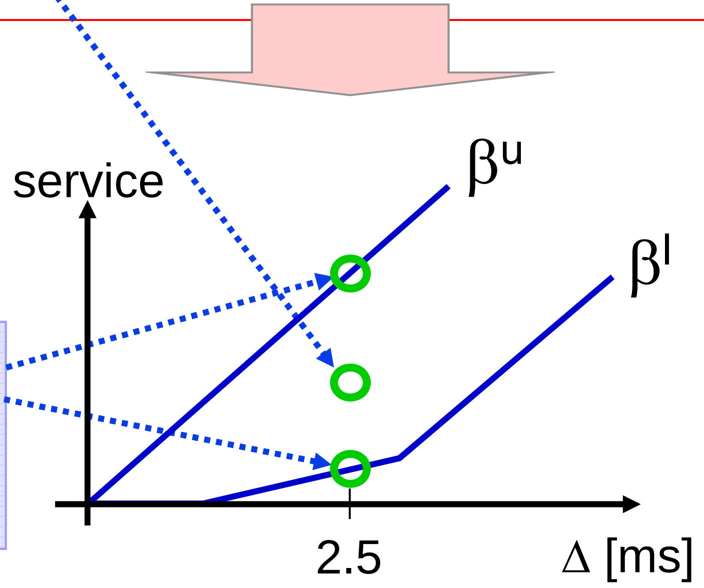
Service Model (Resources)

Resource Availability

available service
in t=[0 .. 2.5] ms



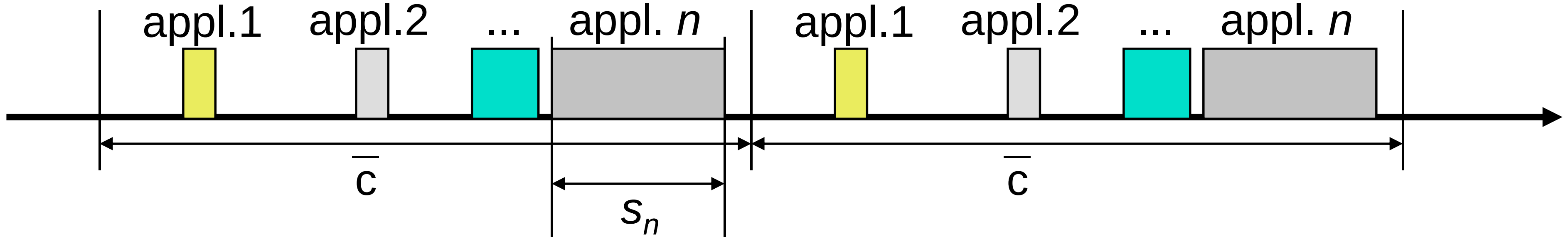
Service Curves [β^l, β^u]



maximum/minimum
available service in *any*
interval of length 2.5 ms

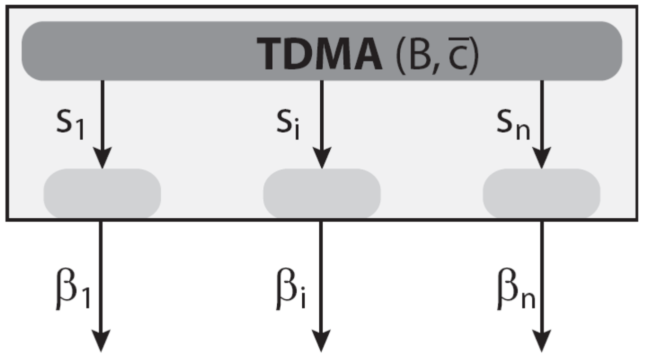
Example 2: TDMA Resource

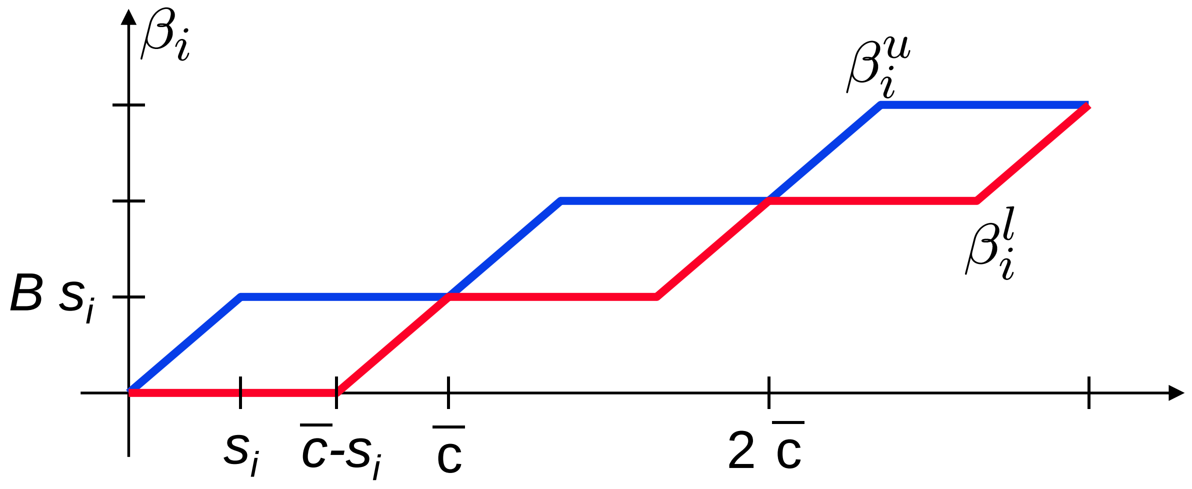
- Consider a real-time system consisting of *n* ***applications*** that are executed on a resource with bandwidth *B* that controls resource access using a ***TDMA policy***.
- Analogously, we could consider a distributed system with *n* ***communicating nodes***, that communicate via a shared bus with bandwidth *B*, with a bus arbitrator that implements a TDMA policy.
- ***TDMA policy***: In every TDMA cycle of length $\bar{c}$, one single resource slot of length s_i is assigned to application *i*.



Example 2: TDMA Resource

- ***Service curves*** available to the applications / node *i:*

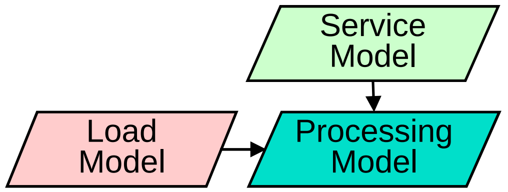


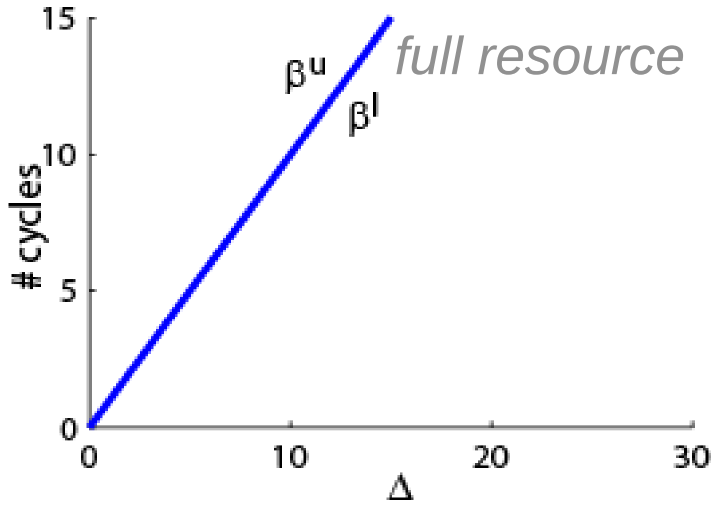


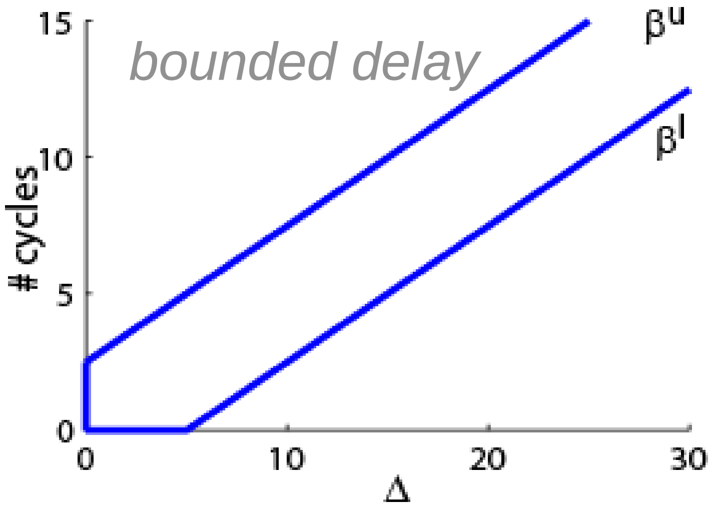
$$\beta_i^l(\Delta) = B \max\{\left\lfloor \frac{\Delta}{\bar{c}} \right\rfloor s_i, \Delta - \left\lceil \frac{\Delta}{\bar{c}} \right\rceil (\bar{c} - s_i)\}$$

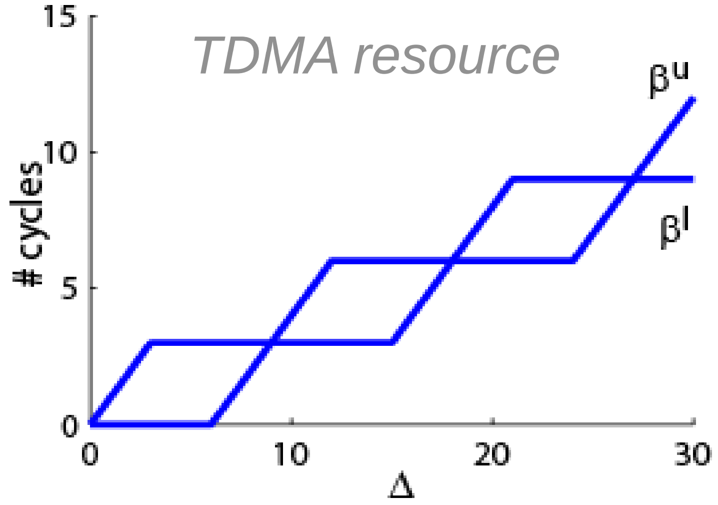
$$\beta_i^u(\Delta) = B \min\{\left\lceil \frac{\Delta}{\bar{c}} \right\rceil s_i, \Delta - \left\lfloor \frac{\Delta}{\bar{c}} \right\rfloor (\bar{c} - s_i)\}$$

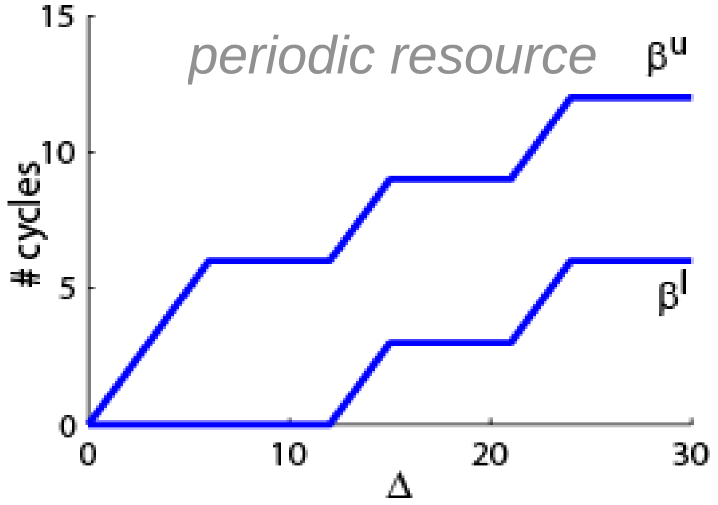
Service Model - Examples



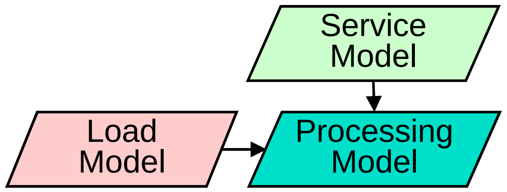






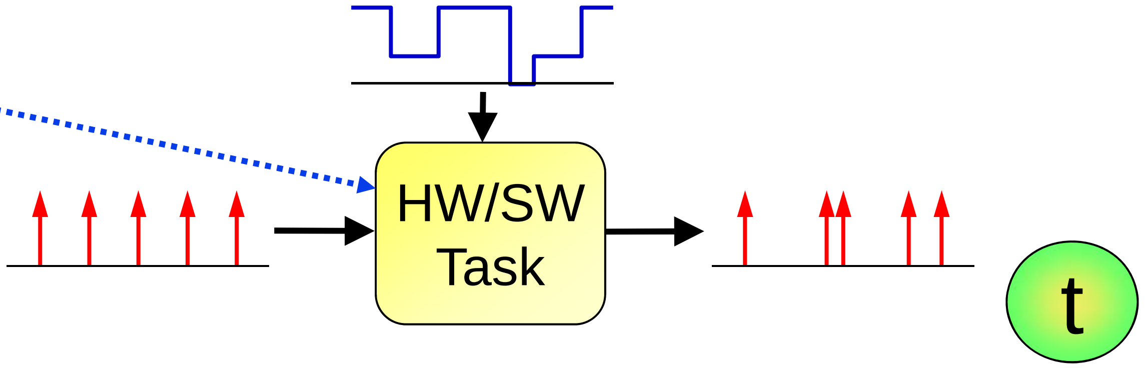


Processing Model (HW/SW)



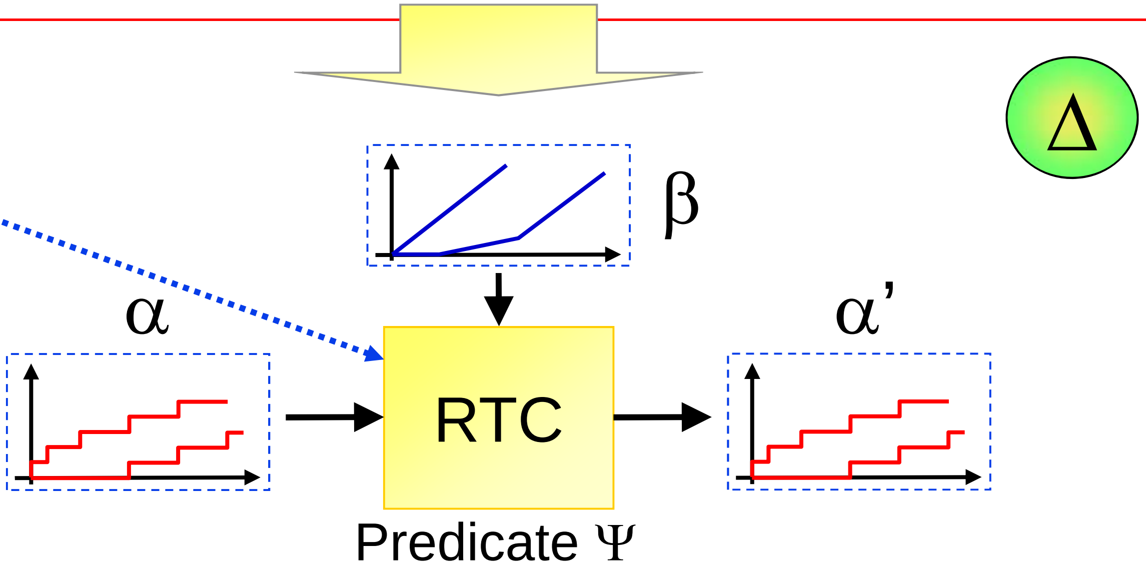
HW/SW Components

Processing semantics and functionality of HW/SW tasks



Abstract Components

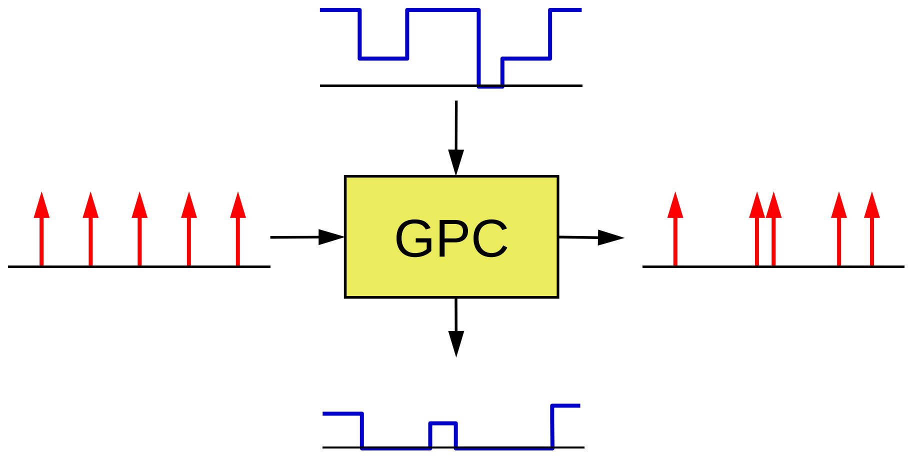
$$\alpha'(\Delta) = f(\alpha, \beta)$$



Foundation

- Real-Time Calculus can be regarded as a ***worst-case/best-case variant of classical queuing theory***. It is a formal method for the analysis of distributed real-time embedded systems.
- ***Related Work***:
  - *Min-Plus Algebra*: F. Baccelli, G. Cohen, G. J. Olster, and J. P. Quadrat, Synchronization and Linearity --- An Algebra for Discrete Event Systems, Wiley, New York, 1992.
  - *Network Calculus*: J.-Y. Le Boudec and P. Thiran, Network Calculus - A Theory of Deterministic Queuing Systems for the Internet, Lecture Notes in Computer Science, vol. 2050, Springer Verlag, 2001.
  - *Adversarial Queuing Theory* [Andrews, Borodin, Kleinberg, Leighton, … 1996]

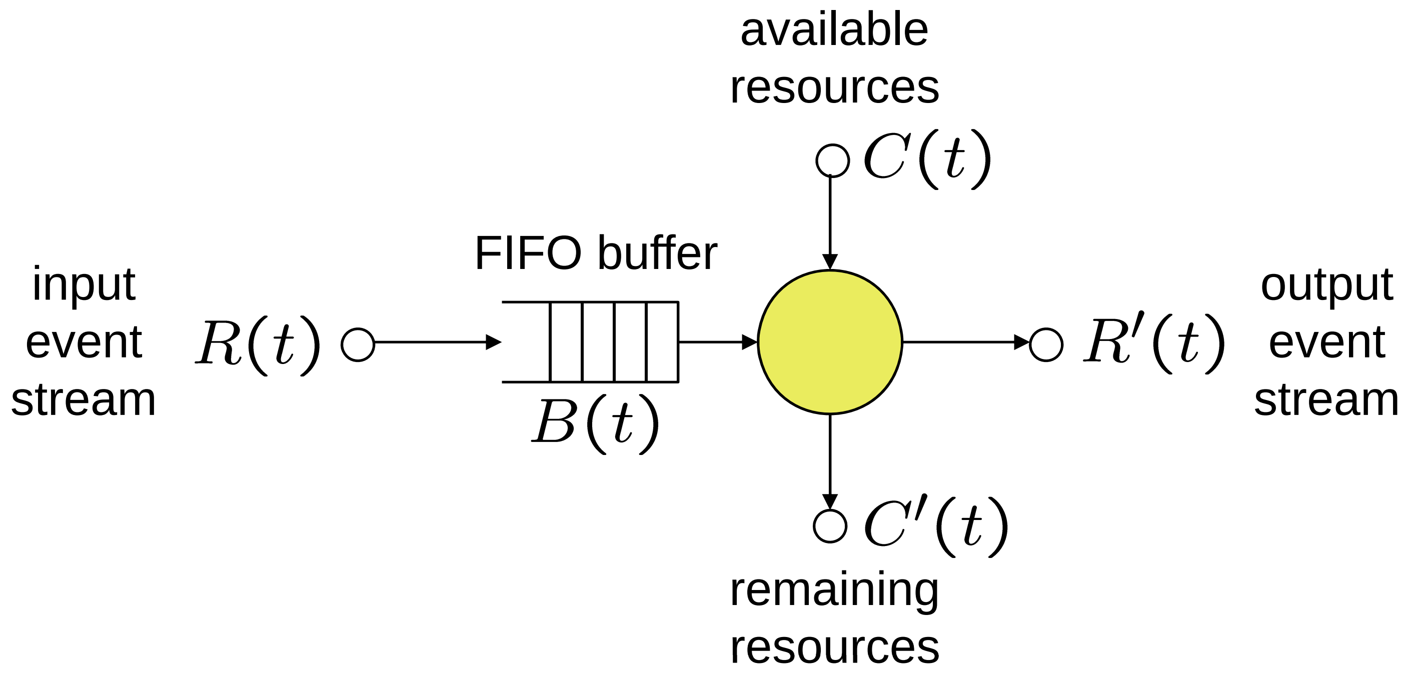
Greedy Processing Component



Behavioral Description

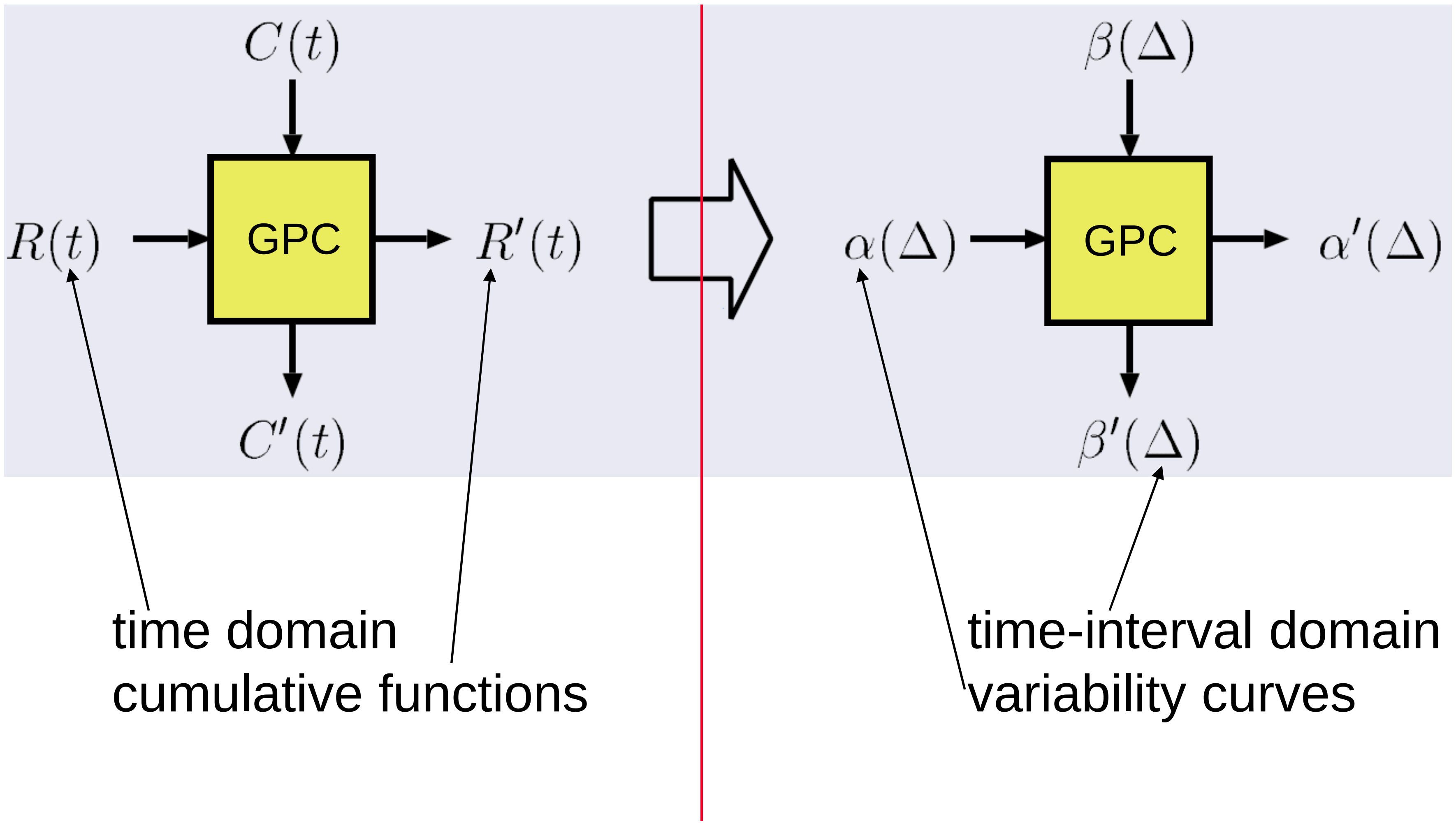
- Component is triggered by incoming events.
- A fully preemptable task is instantiated at every event arrival to process the incoming event.
- Active tasks are processed in a greedy fashion in FIFO order.
- Processing is restricted by the availability of resources.

Greedy Processing Component (GPC)



- ***Examples:***
  - computation (event – task instance, resource – computing resource [tasks/second])
  - communication (event – data packet, resource – bandwidth [packets/second])

Abstraction



time domain
cumulative functions

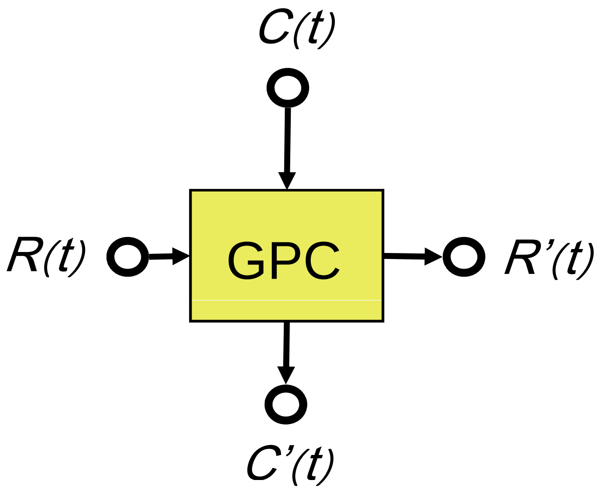
time-interval domain
variability curves

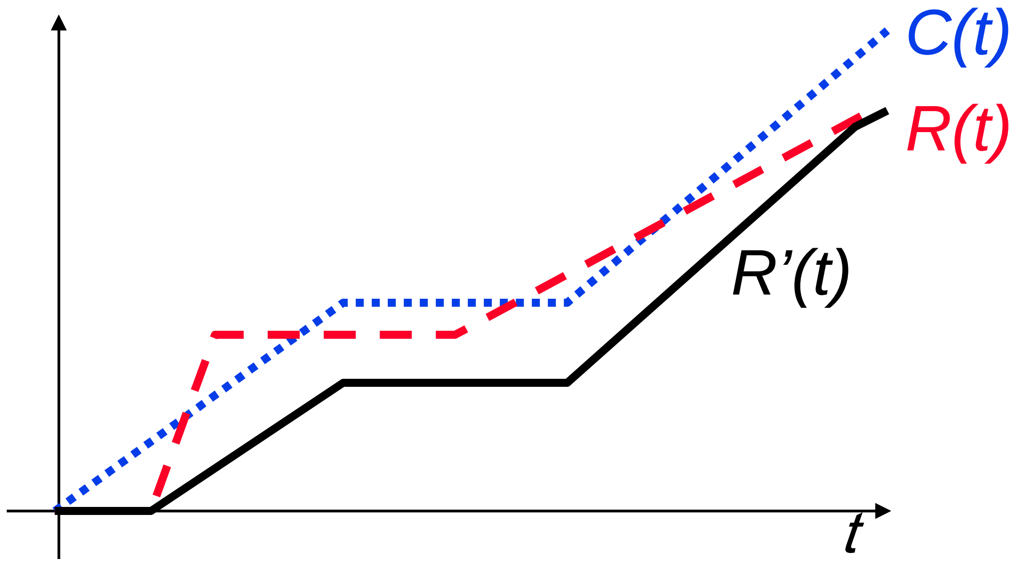
Greedy Processing Component (GPC)

If the resource and event streams describe available and requested units of processing or communication, then

$$\left.\begin{aligned} C(t) &= C'(t) + R'(t) \\ B(t) &= R(t) - R'(t) \end{aligned}\right\} \text{Conservation Laws}$$

$$R'(t) = \inf_{0 \le u \le t} \{R(u) + C(t) - C(u)\}$$

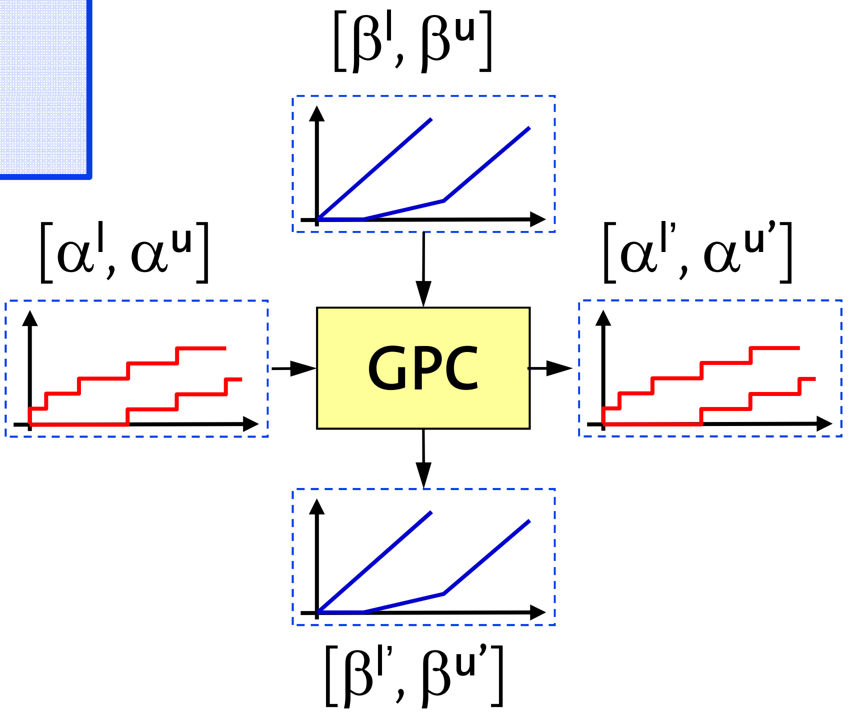




Tight Bounds

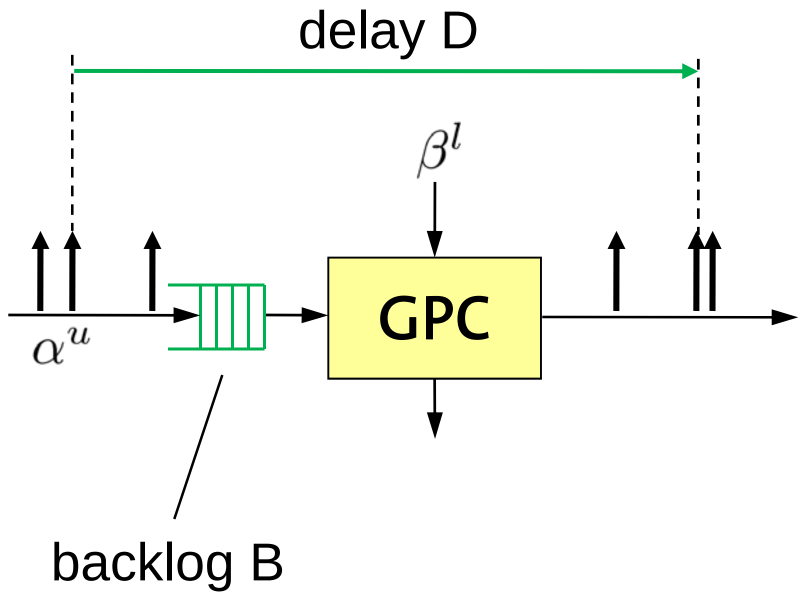
The greedy processing component transforms the variability curves as follows:

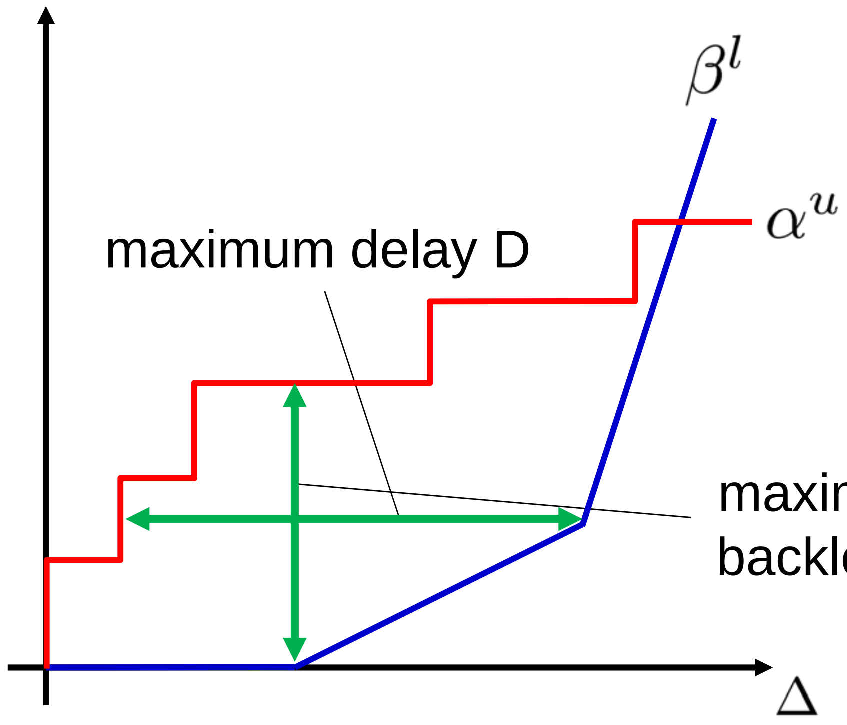
$$\alpha^{u\prime} = [(\alpha^u \otimes \beta^u) \oslash \beta^l] \wedge \beta^u$$
$$\alpha^{l\prime} = [(\alpha^l \oslash \beta^u) \otimes \beta^l] \wedge \beta^l$$
$$\beta^{u\prime} = (\beta^u - \alpha^l)\overline{\oslash}0$$
$$\beta^{l\prime} = (\beta^l - \alpha^u)\overline{\otimes}0$$



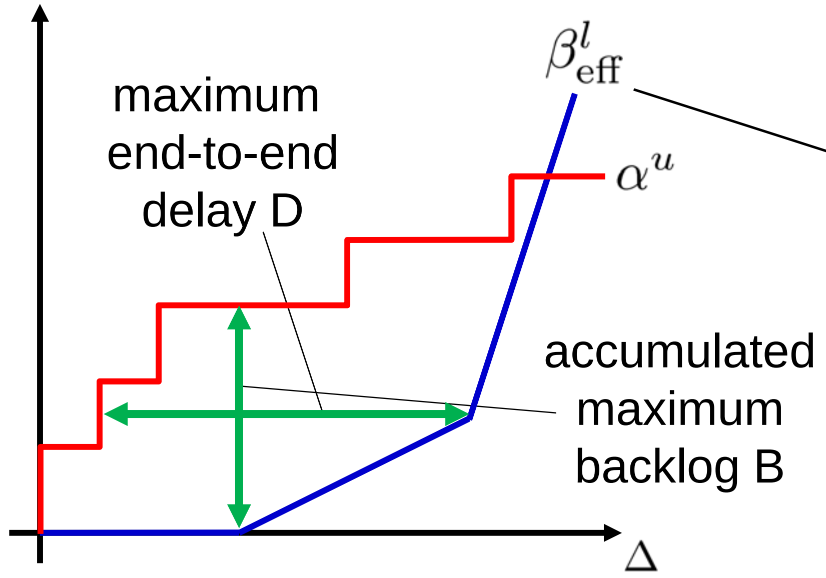
$$(f \otimes g)(t) = \inf_{0 \le u \le t} \{f(t-u) + g(u)\}$$
$$(f \oslash g)(t) = \sup_{u \ge 0} \{f(t+u) - g(u)\}$$
$$(f \overline{\otimes} g)(t) = \sup_{0 \le u \le t} \{f(t-u) + g(u)\}$$
$$(f \overline{\oslash} g)(t) = \inf_{u \ge 0} \{f(t+u) - g(u)\}$$

Delay and Backlog

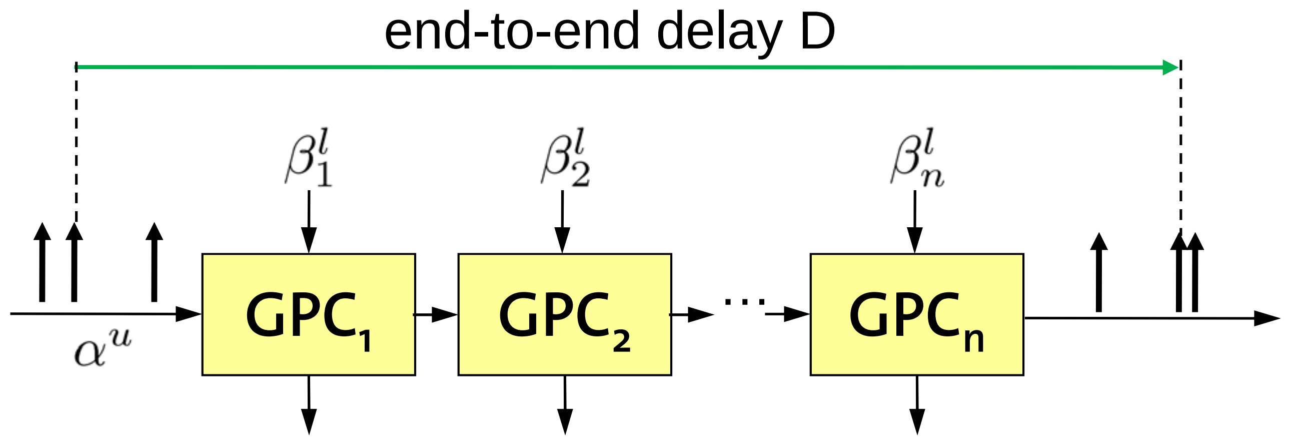




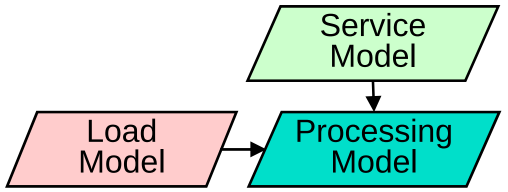
Celebrated Result on Delay and Backlog

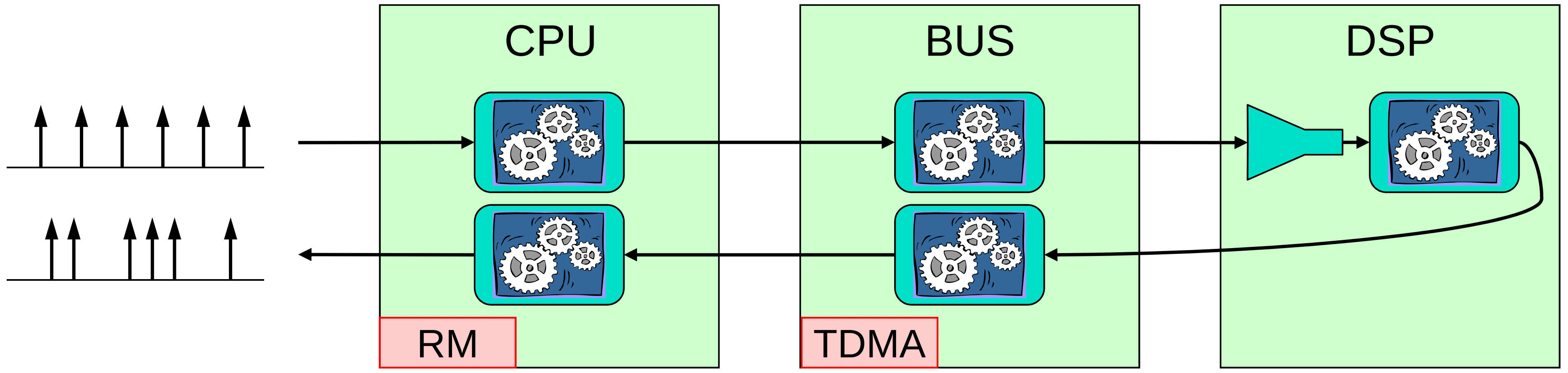


$$\beta^l_{\text{eff}} = \bigotimes_{i=1}^{n} \beta^l_i$$



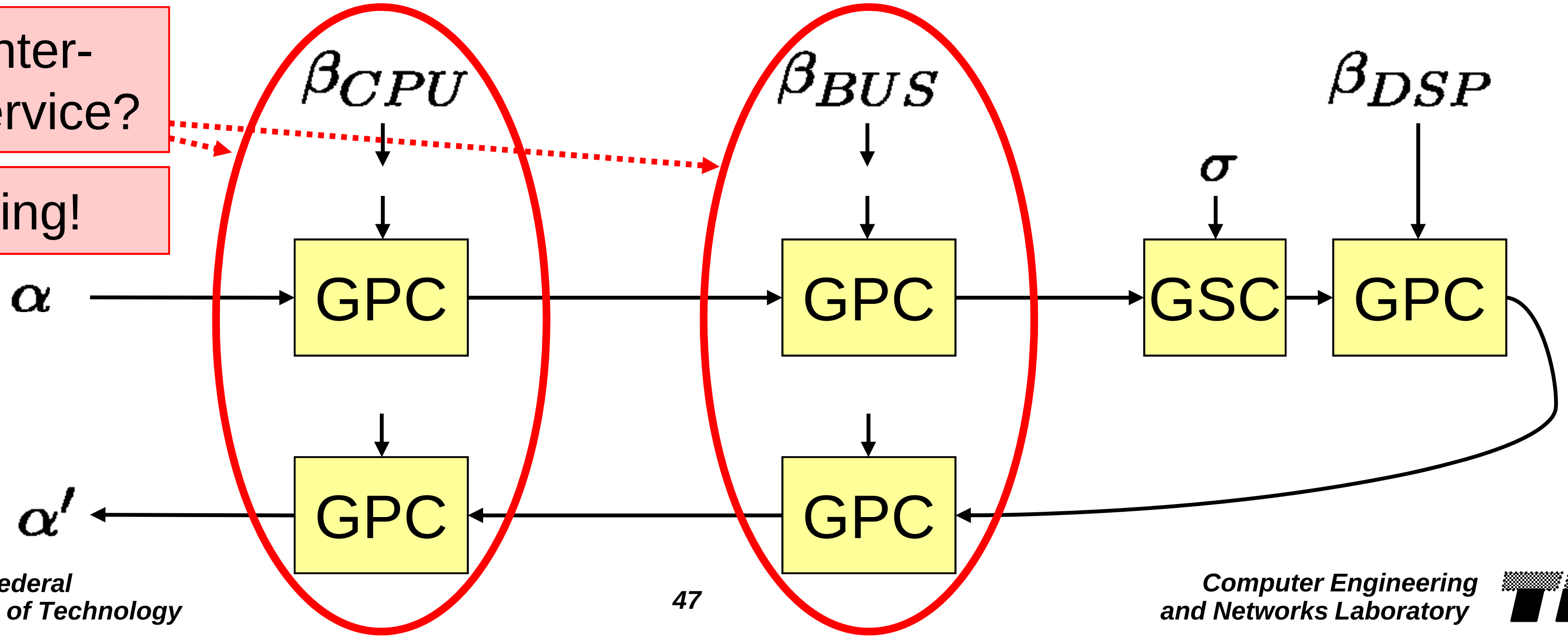
System Composition

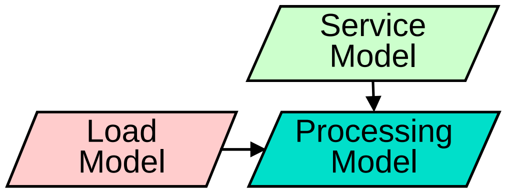




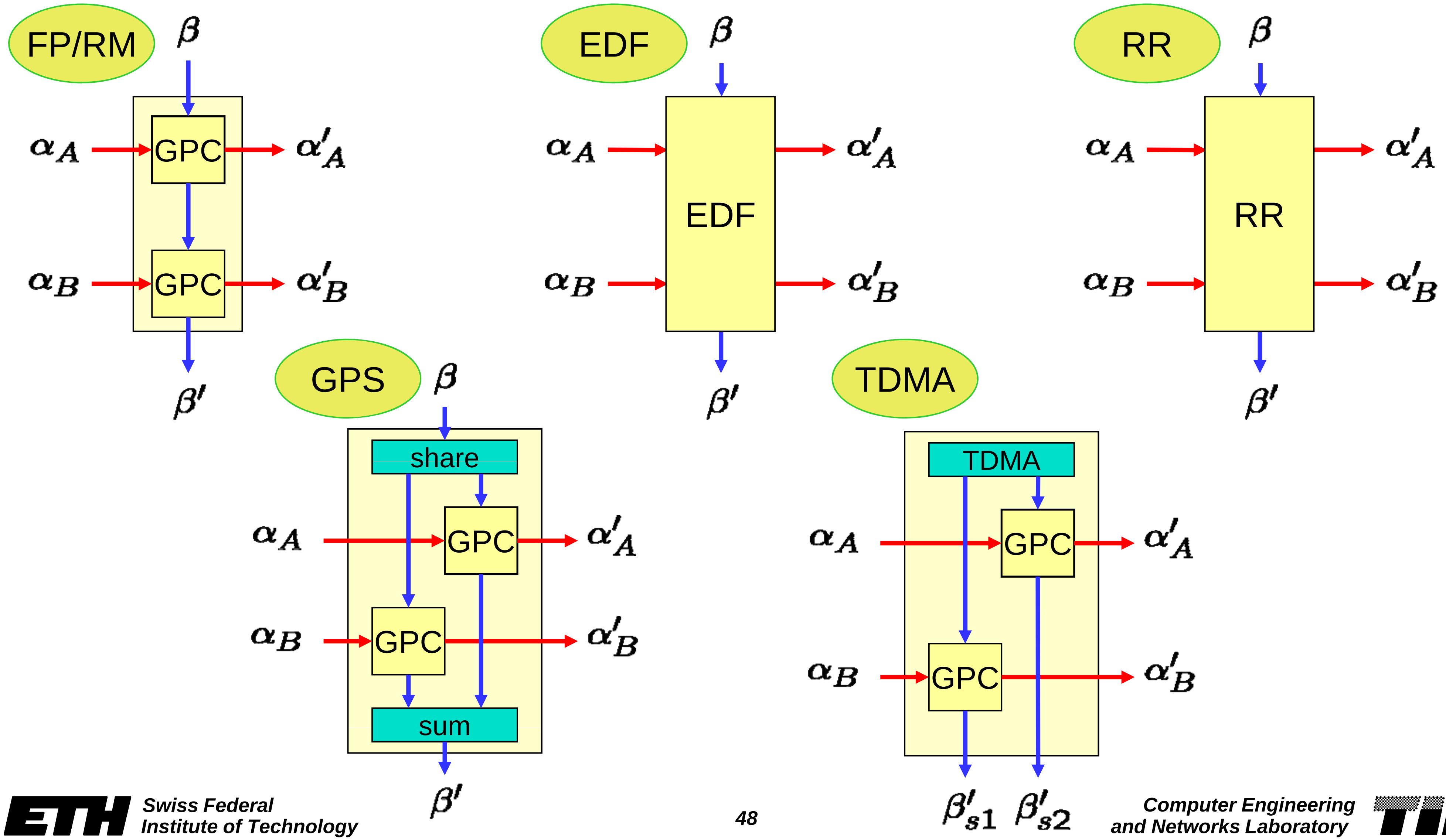
How to inter-connect service?

Scheduling!

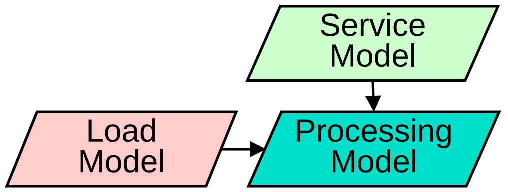


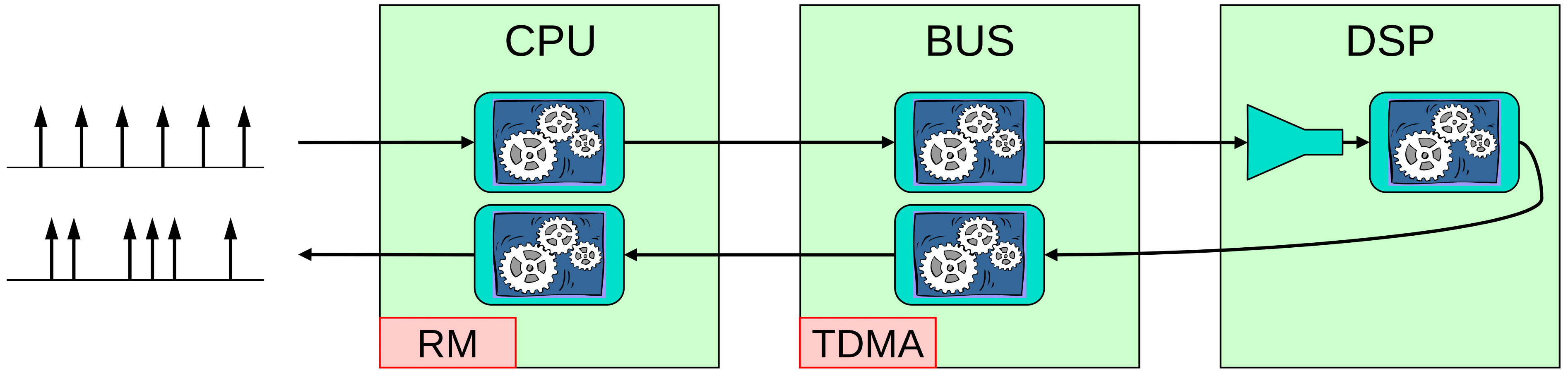


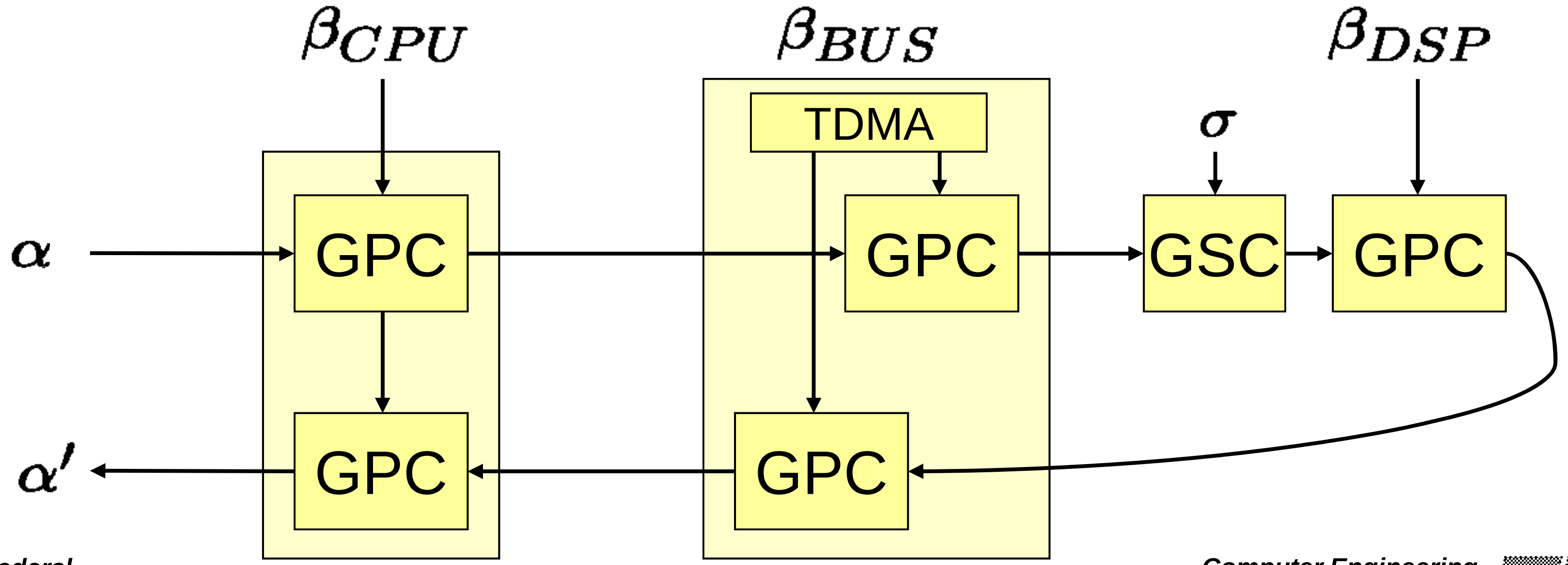
Scheduling and Arbitration



Complete System Composition







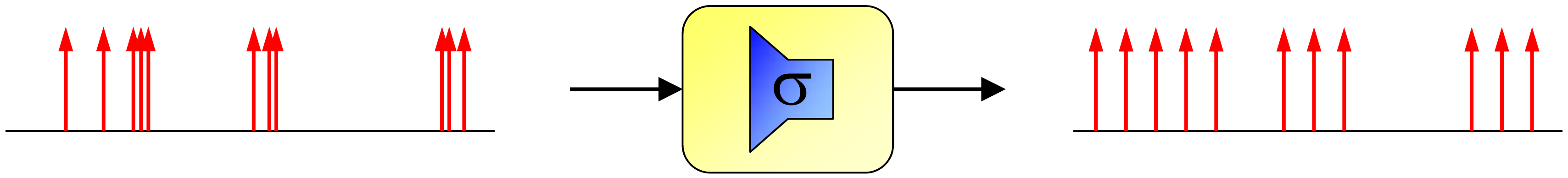
Compositional Analysis

- Artificial Examples-

Greedy Traffic Shaper

- Access Shaper
  - delays access requests such that the resulting access pattern conforms to a given specification

- Greedy Access Shaper
  - no access request gets delayed any longer than necessary

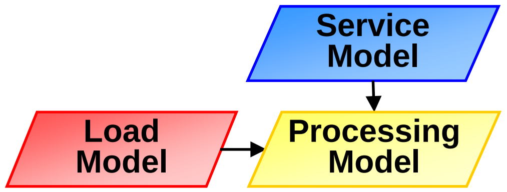


Why Access Shaping?

- Internal Re-Shaping
  - Reduces global buffer requirements
  - Reduces end-to-end delays

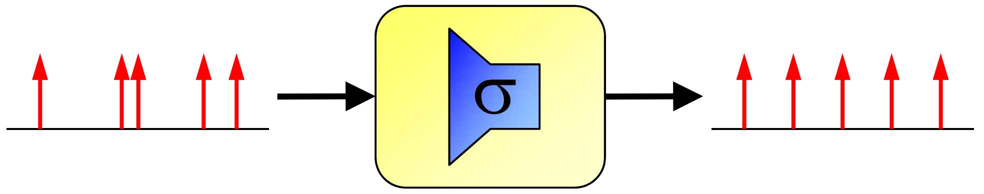
- External Input-Shaping
  - Ensures specification conformant system inputs

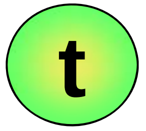
How to model and analyze greedy shapers?



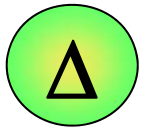
Modeling of Greedy Shapers

Greedy Shaper





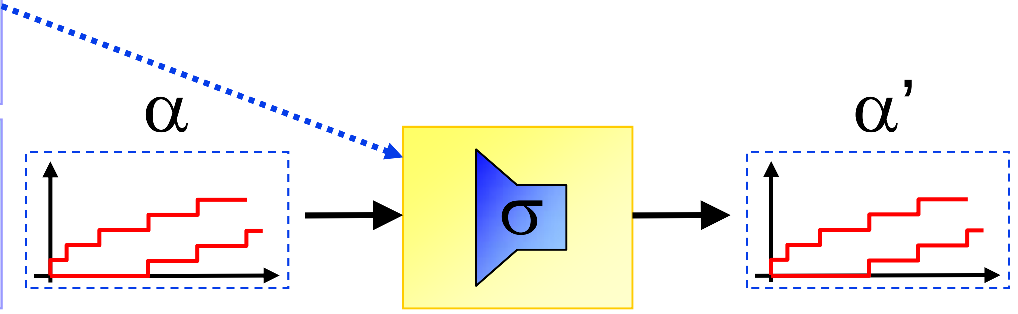
Abstract Greedy Shaper



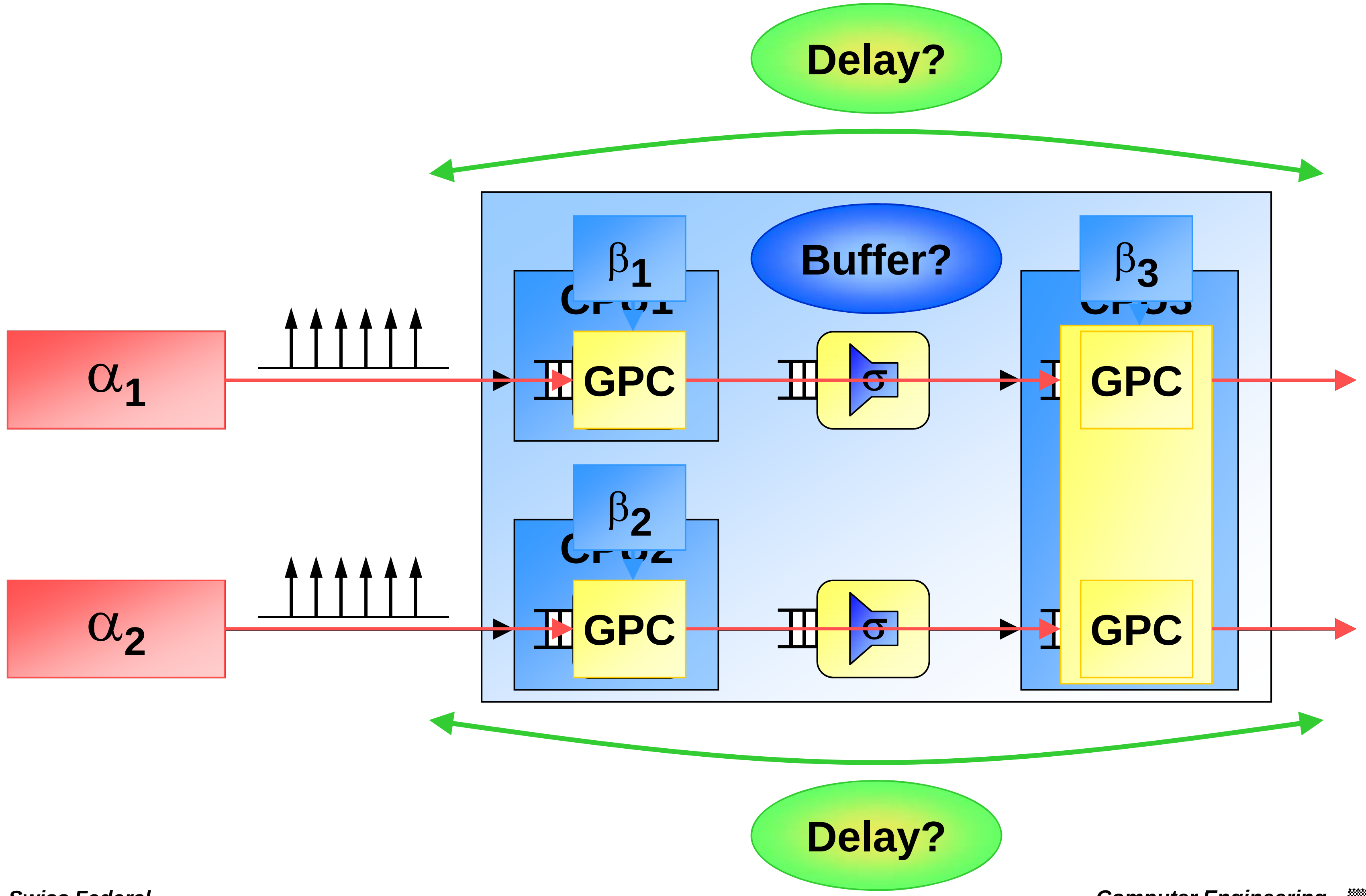
$$\alpha'(\Delta) = f(\alpha, \sigma)$$

$$\alpha'^u = \alpha^u \otimes \sigma$$
$$\alpha'^l = \alpha^l \otimes (\sigma \overline{\oslash} \sigma)$$

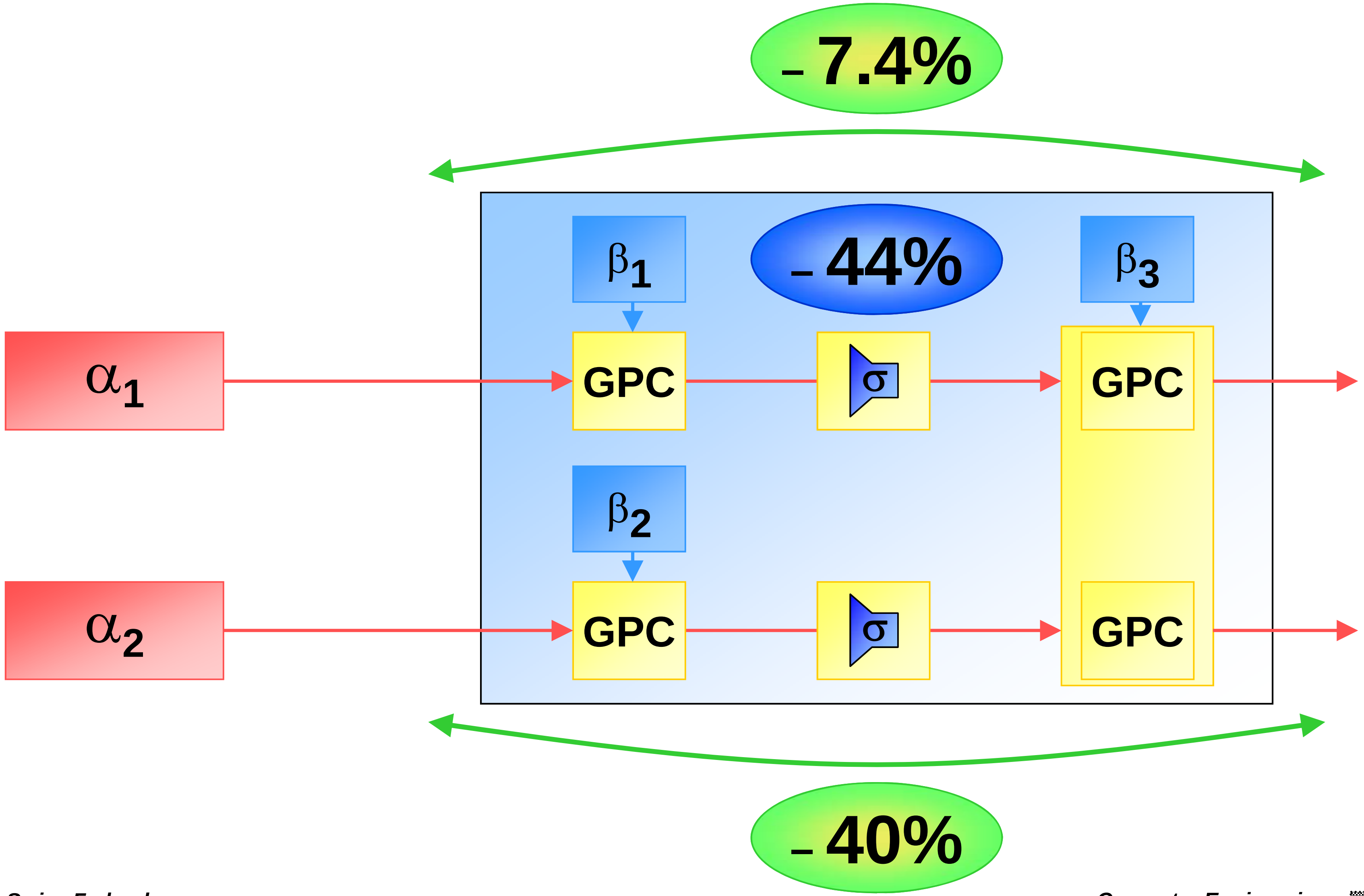
$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\}$$
$$(f \overline{\oslash} g)(\Delta) = \inf_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}$$



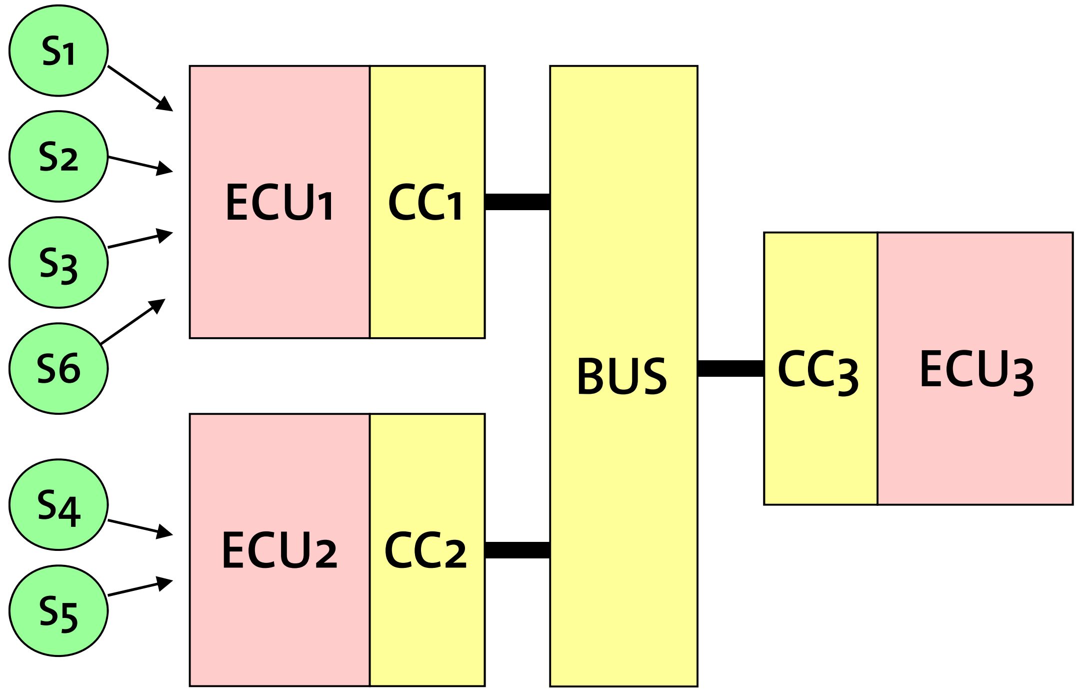
Internal Re-Shaping



Internal Re-Shaping



Case Study



6 Real-Time Input Streams
- with jitter
- with bursts
- deadline > period

3 ECU's with own CC's

13 Tasks & 7 Messages
- with different WCED

2 Scheduling Policies
- Earliest Deadline First (ECU's)
- Fixed Priority (ECU's & CC's)

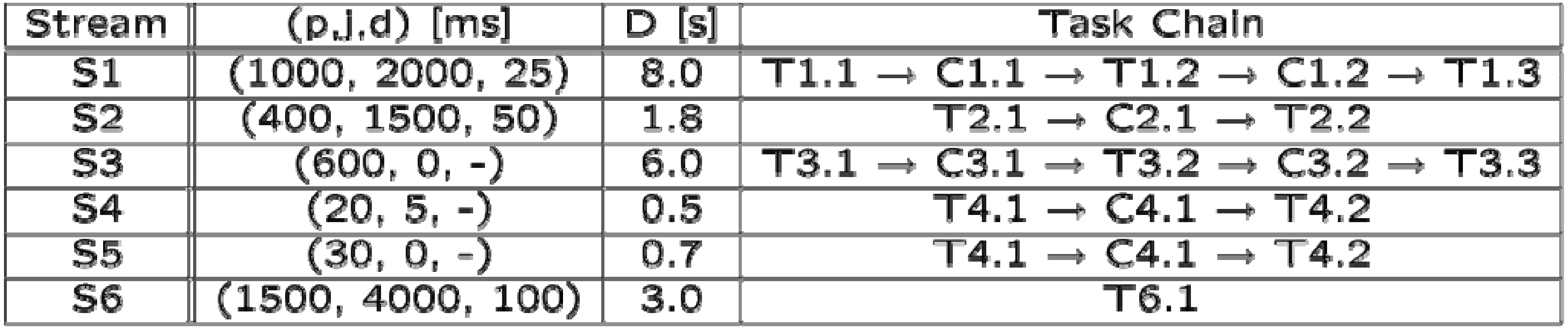
Hierarchical Scheduling
- Static & Dynamic Polling Servers

Bus with TDMA
- 4 time slots with different lengths
  (#1,#3 for CC1, #2 for CC3, #4 for CC3)

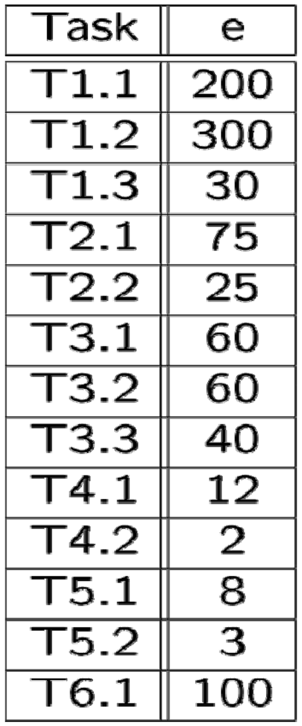
Total Utilization:

- ECU1	59 %
- ECU2	87 %
- ECU3	67 %
- BUS	56 %

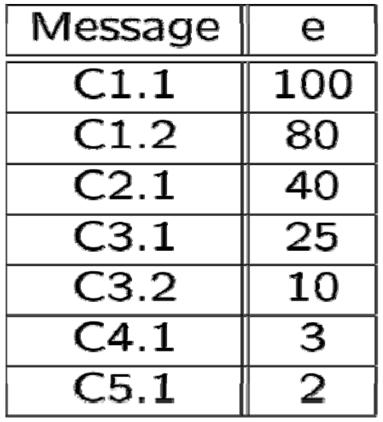
Specification Data



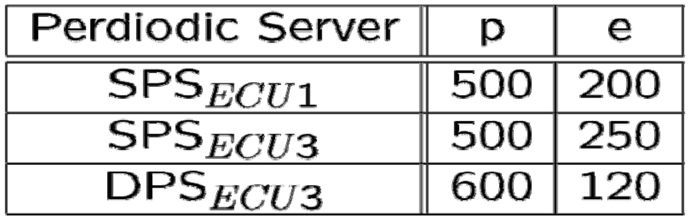
Stream	(p,j,d) [ms]	D [s]	Task Chain
S1	(1000, 2000, 25)	8.0	T1.1 → C1.1 → T1.2 → C1.2 → T1.3
S2	(400, 1500, 50)	1.8	T2.1 → C2.1 → T2.2
S3	(600, 0, -)	6.0	T3.1 → C3.1 → T3.2 → C3.2 → T3.3
S4	(20, 5, -)	0.5	T4.1 → C4.1 → T4.2
S5	(30, 0, -)	0.7	T4.1 → C4.1 → T4.2
S6	(1500, 4000, 100)	3.0	T6.1



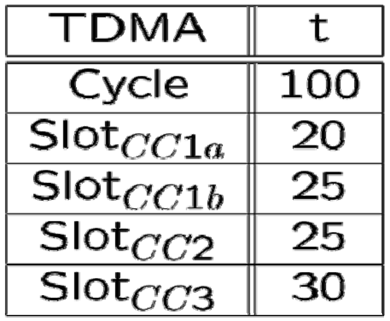
Task	e
T1.1	200
T1.2	300
T1.3	30
T2.1	75
T2.2	25
T3.1	60
T3.2	60
T3.3	40
T4.1	12
T4.2	2
T5.1	8
T5.2	3
T6.1	100



Message	e
C1.1	100
C1.2	80
C2.1	40
C3.1	25
C3.2	10
C4.1	3
C5.1	2

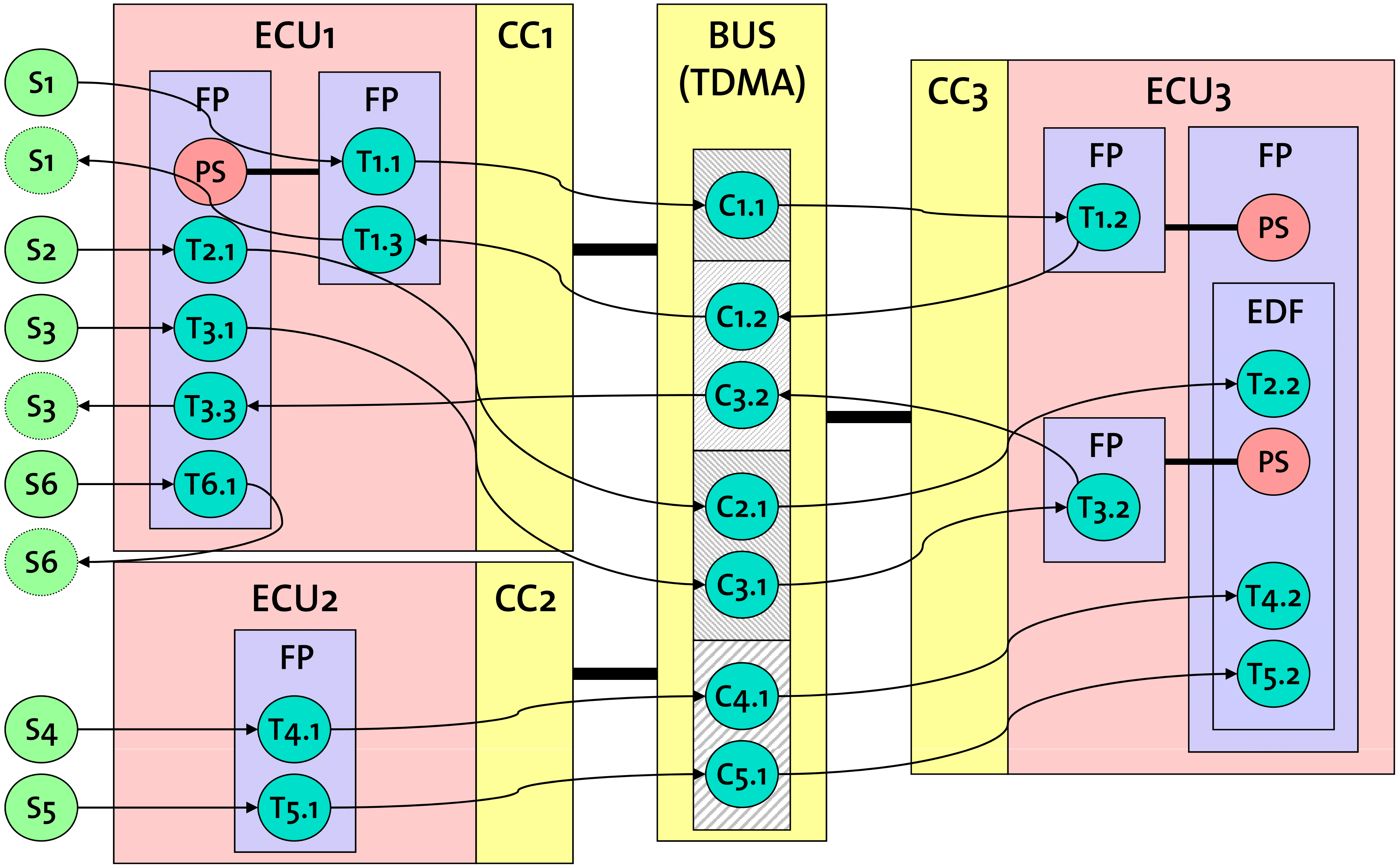


Perdiodic Server	p	e
SPS_{ECU1}	500	200
SPS_{ECU3}	500	250
DPS_{ECU3}	600	120

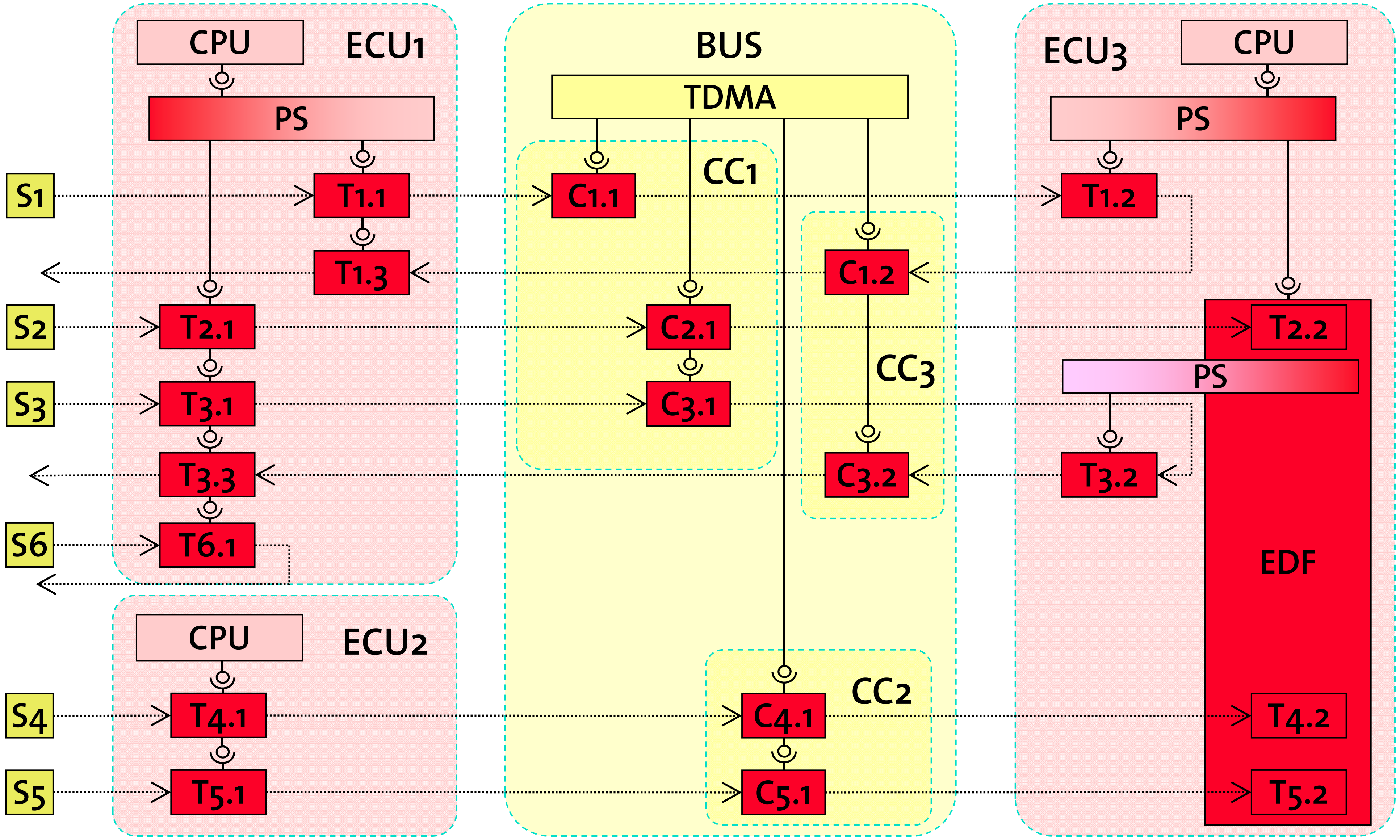


TDMA	t
Cycle	100
Slot_{CC1a}	20
Slot_{CC1b}	25
Slot_{CC2}	25
Slot_{CC3}	30

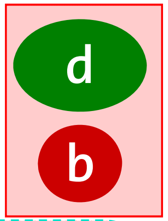
The Distributed Embedded System...

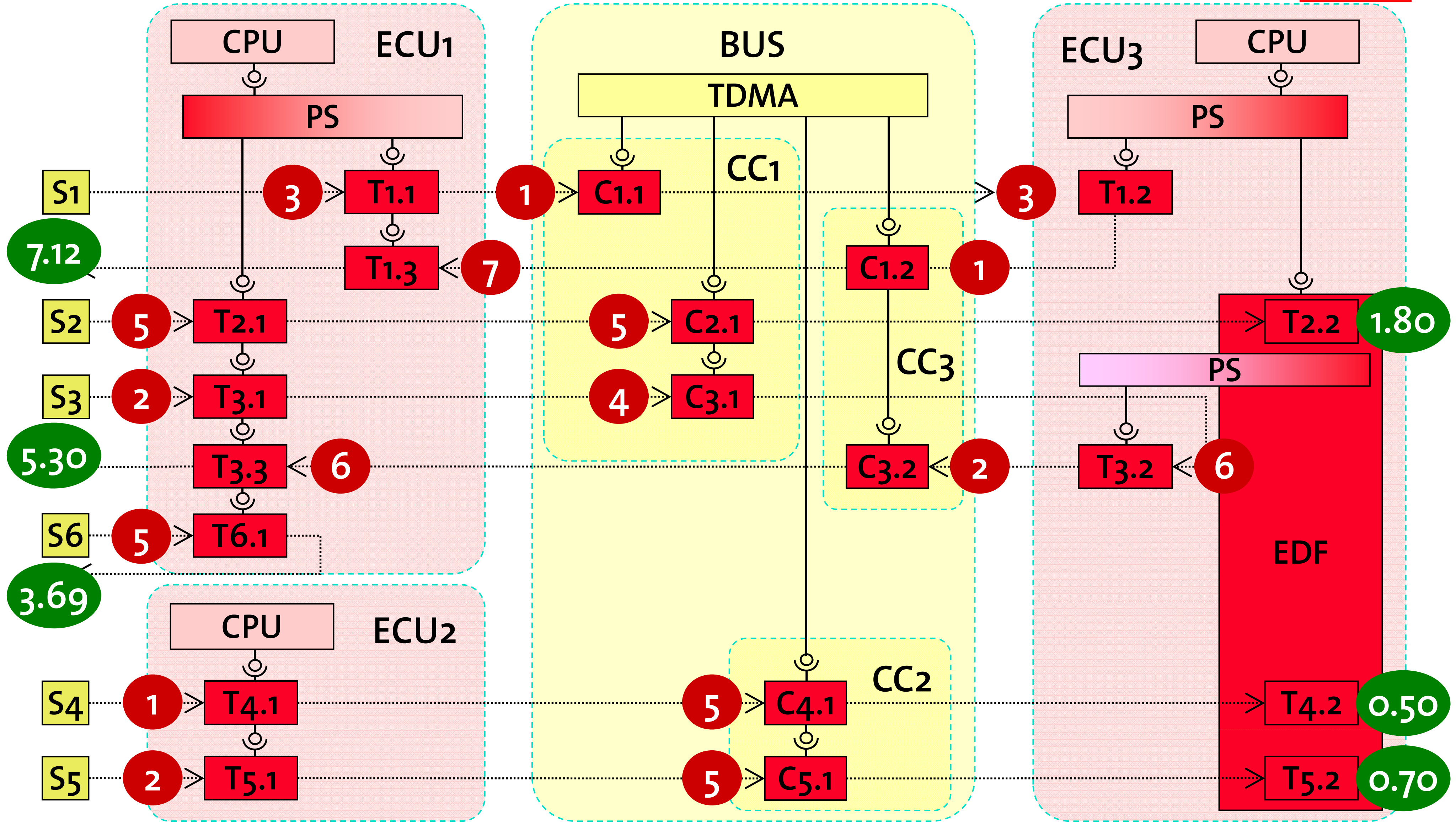


... and its MPA Model

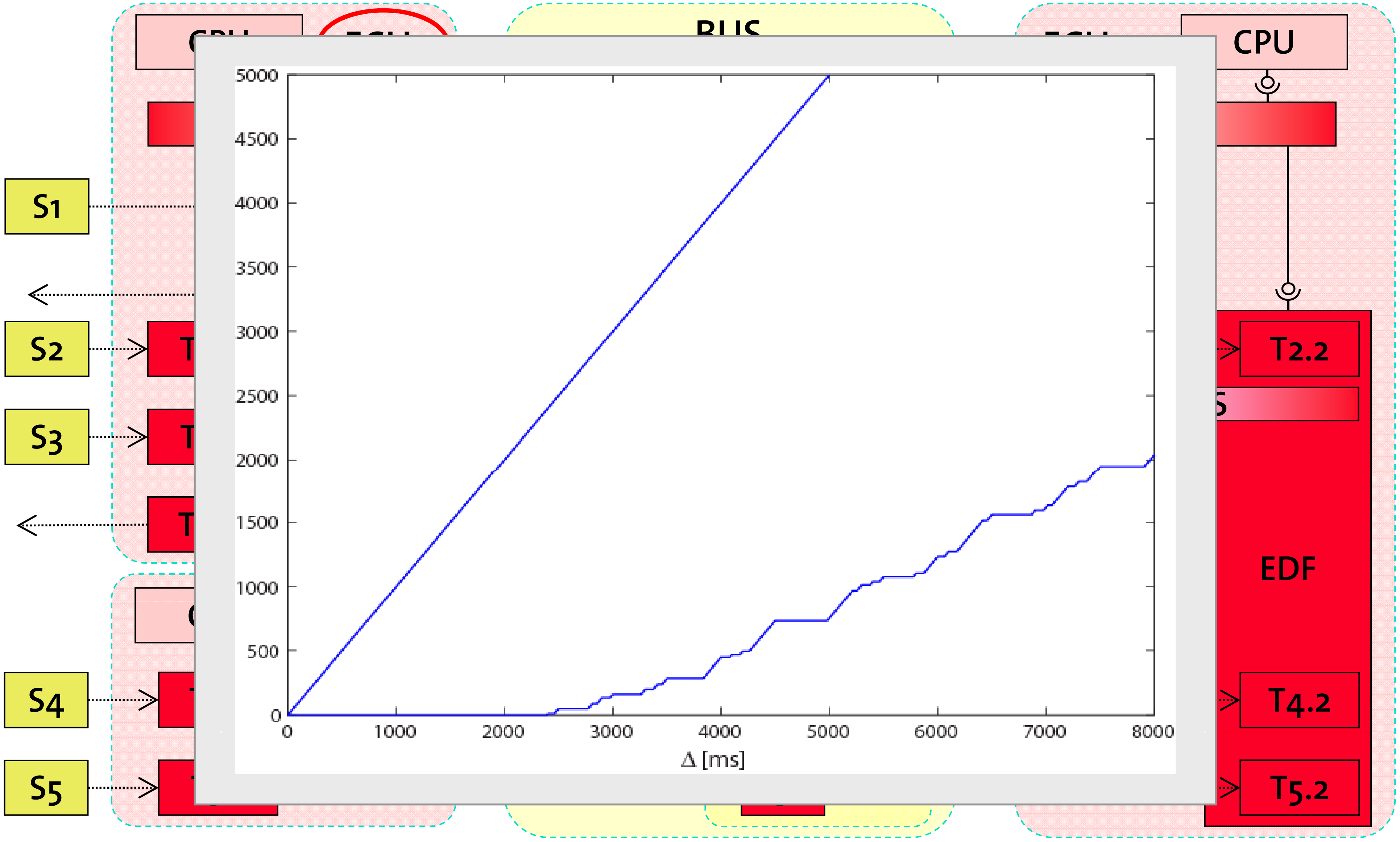


Buffer & Delay Guarantees

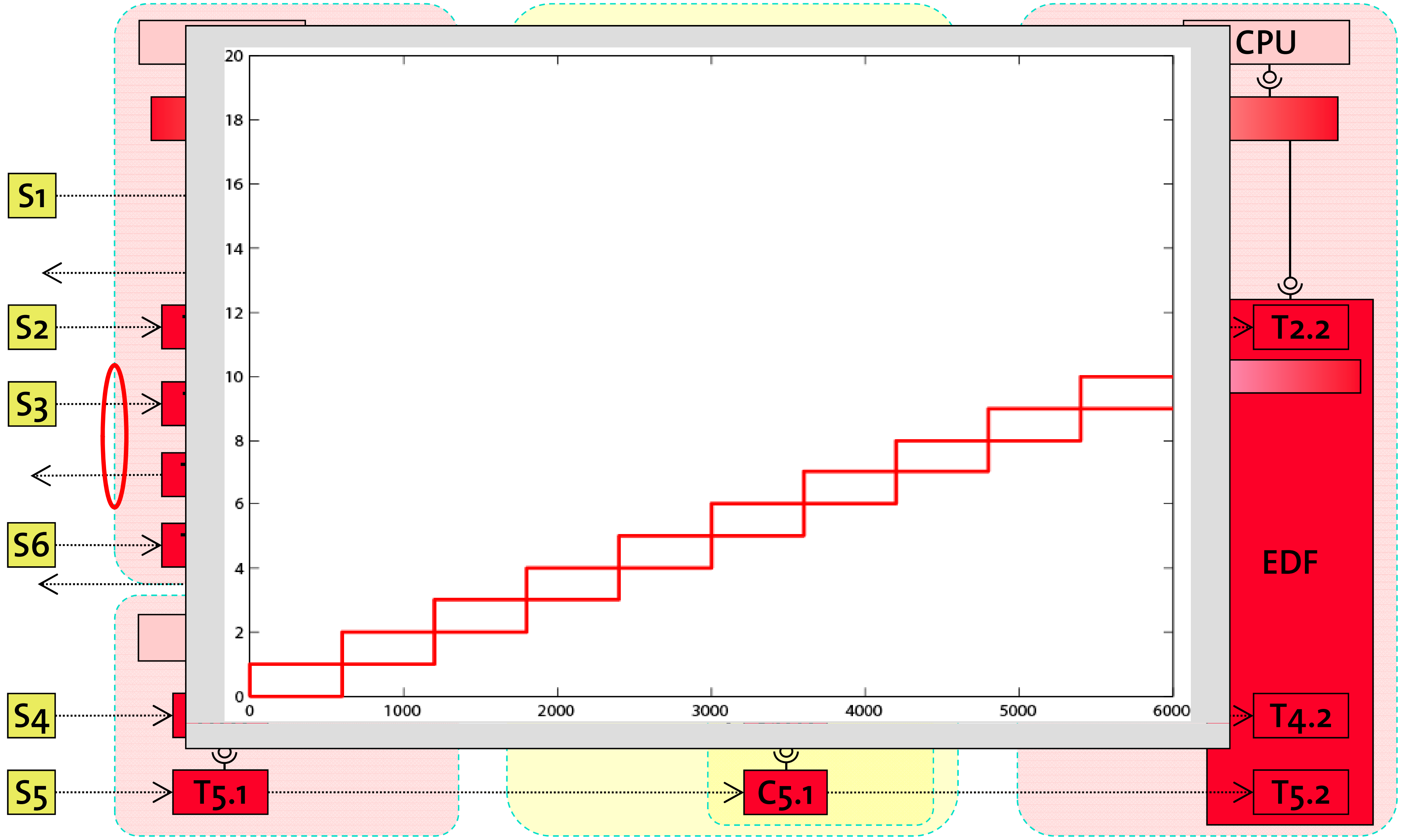




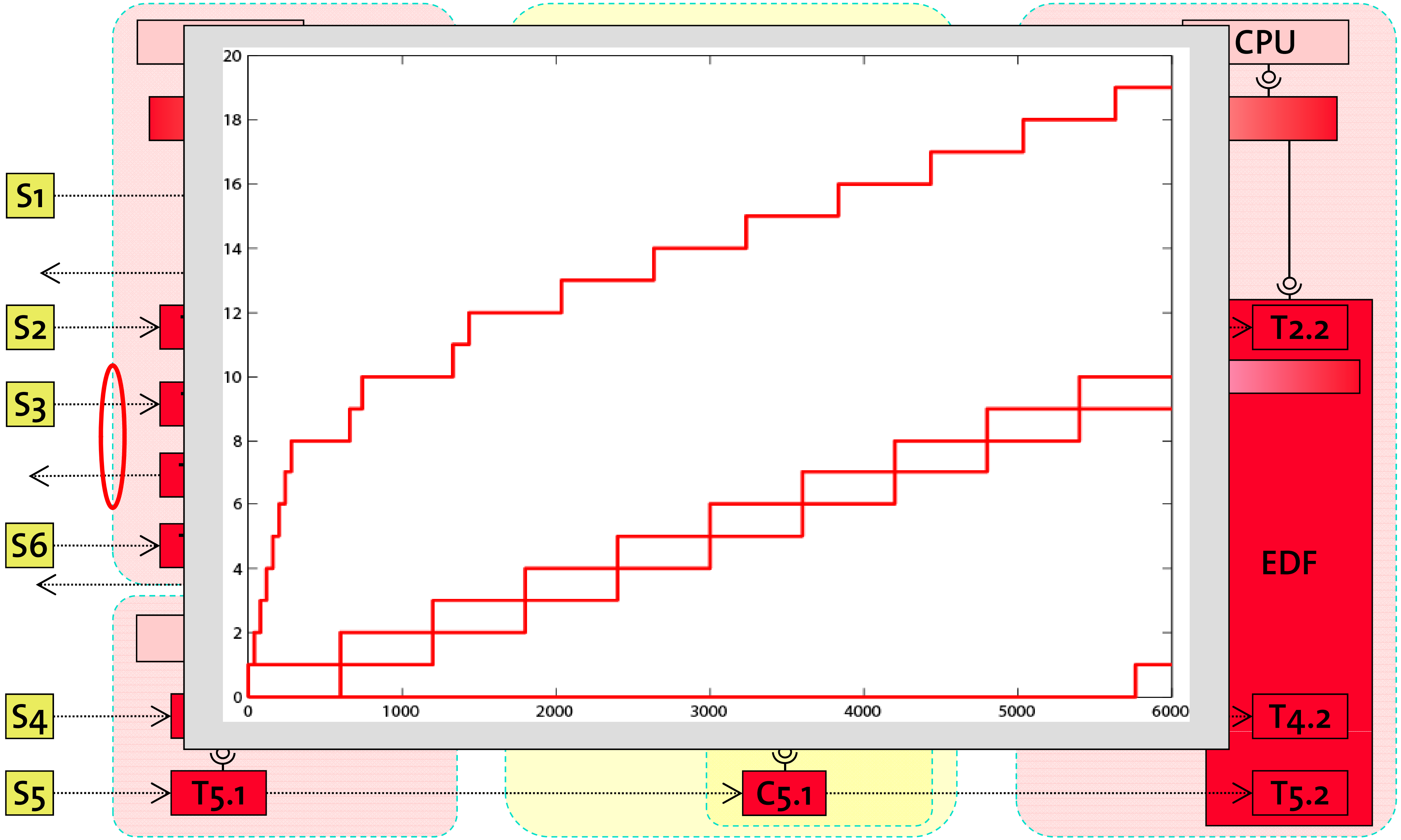
Available & Remaining Service of ECU1



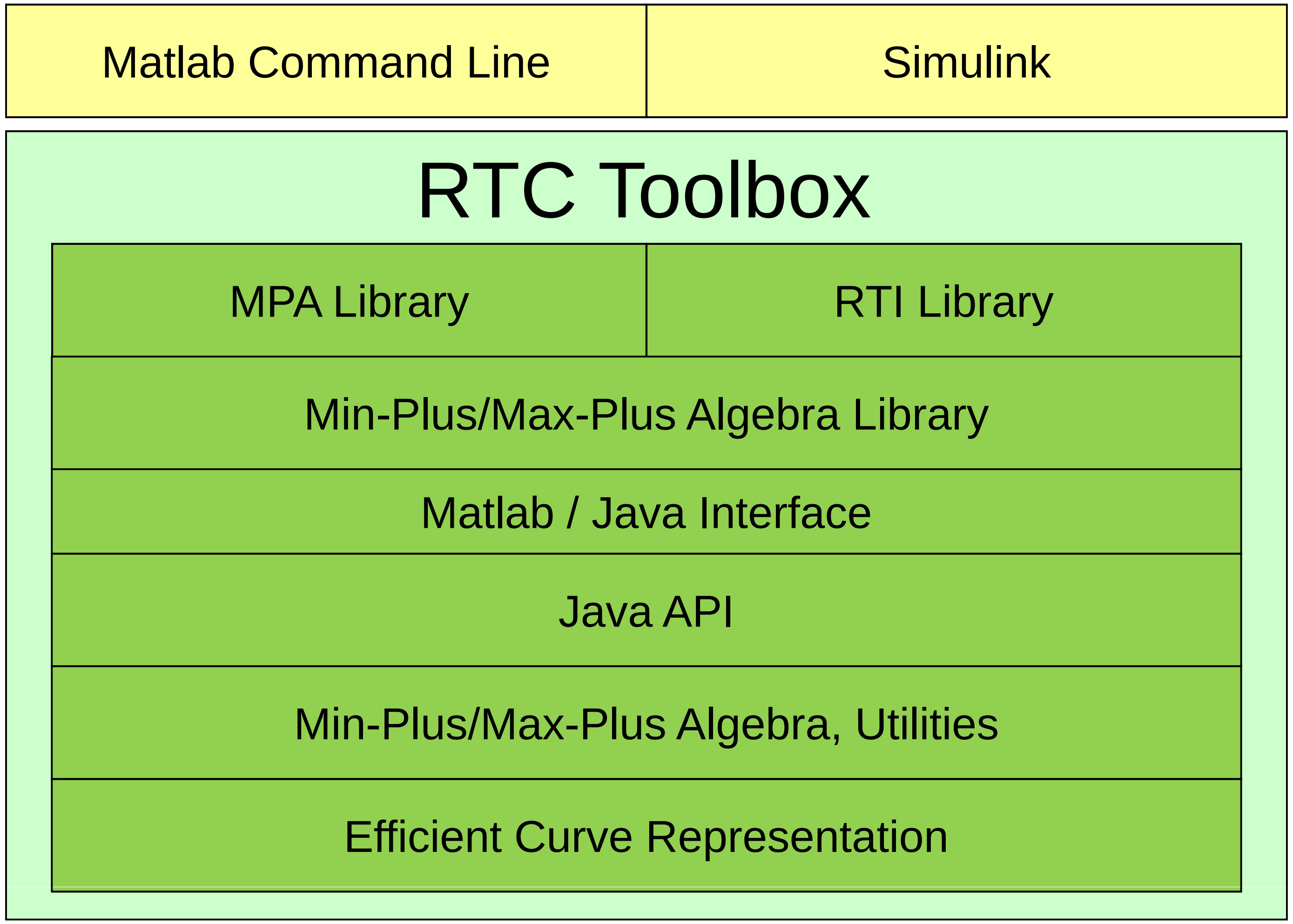
Input of Stream 3



Output of Stream 3



RTC Toolbox (www.mpa.ethz.ch/rtctoolbox)


Matlab Command Line
Simulink
RTC Toolbox
MPA Library
RTI Library
Min-Plus/Max-Plus Algebra Library
Matlab / Java Interface
Java API
Min-Plus/Max-Plus Algebra, Utilities
Efficient Curve Representation

Contents

Architectural Interactions

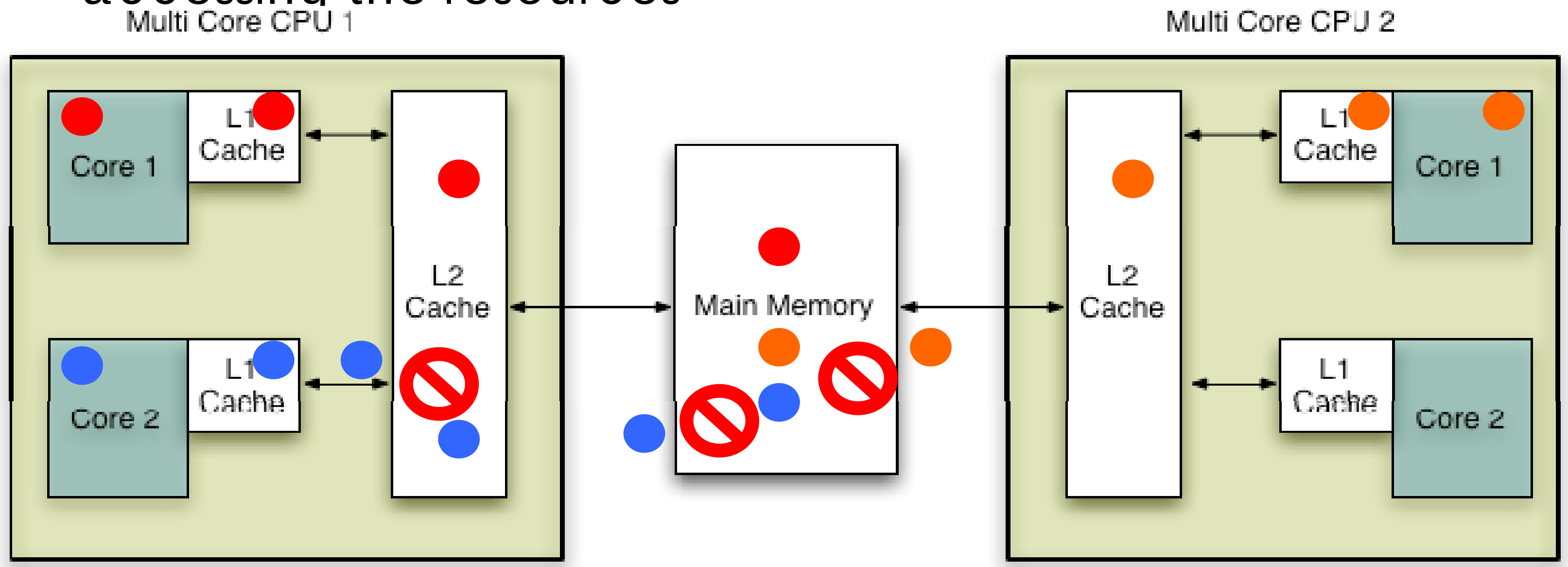
- Shared Resources in Multicore Systems -

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Motivation

Interferences:
CPU1/Core2 blocked by **CPU1/Core1** on L2 Cache
CPU2/Core1 blocked by **CPU1/Core1** on Main Memory
CPU1/Core2 blocked by **CPU2/Core1** on Main Memory

- COTS Systems use shared resources (Memory, Bus)
- Multiple entities competing for shared resources
  - waiting for other entities to release the resource
  - accessing the resources



[illegible] Main Memory request served

Main Memo[illegible]

[illegible] 2 stall

Motivation (2)

Multi-Core Architecture with shared resource

- shared memory, communication peripherals, I/O peripherals

Stalling due to Interference

- Depends on structure of tasks on the cores
- Depends on blocking vs. non-blocking execution semantics
- Depends on arbitration policy on the shared resource
    - static access, for example TDMA
    - dynamic access, for example round robin, FCFS, priority driven

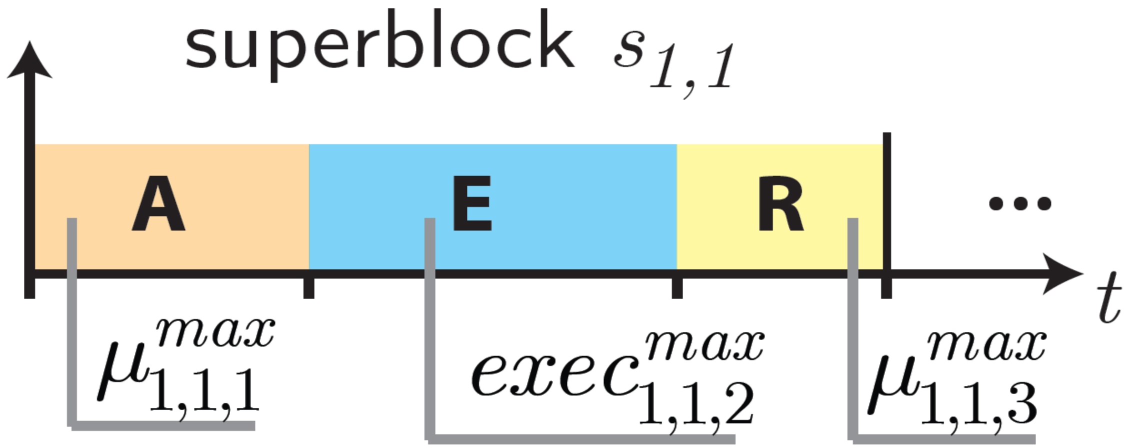
Related Work

- Schliecker et al. [CODES 2006, CODES 2008, DATE 2010]
  - Event models specify tasks interference in time windows
  - tasks active time increases by number of interferences
  - Iterative approach to compute WCET

- Rosen et al. [RTSS 2007]
  - static analysis delivers feasible execution traces
  - a given TDMA schedule the WCET is computed
  - efficient TDMA schedules are obtained using EA

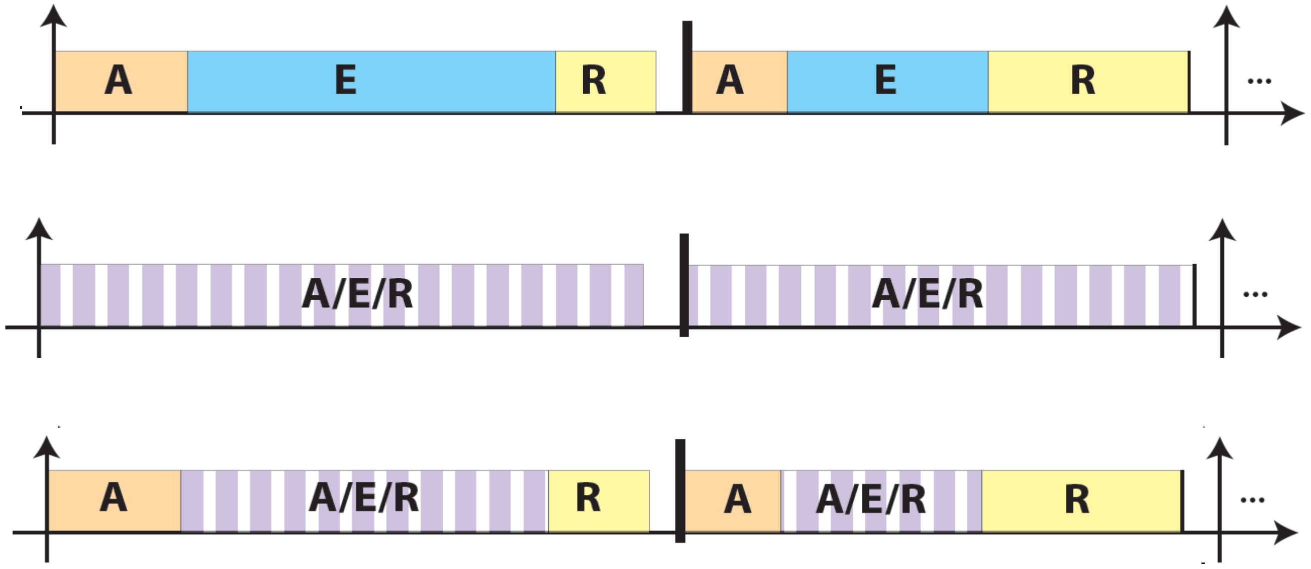
Task / Superblock Model (1)

- Tasks are structured as sequences of superblocks
  - fixed order of execution
  - upper bounds on execution and communication demands
- Dedicated phases for resource access and computation
  - phases have different amount of access requests
  - structure increases predictability (in terms of WCRT)
  - model motivated by industrial applications in the automotive industry



Task / Superblock Model (2)

- 3 Models to specify resource accesses:
  - Dedicated Model
  - General Model
  - Hybrid Model

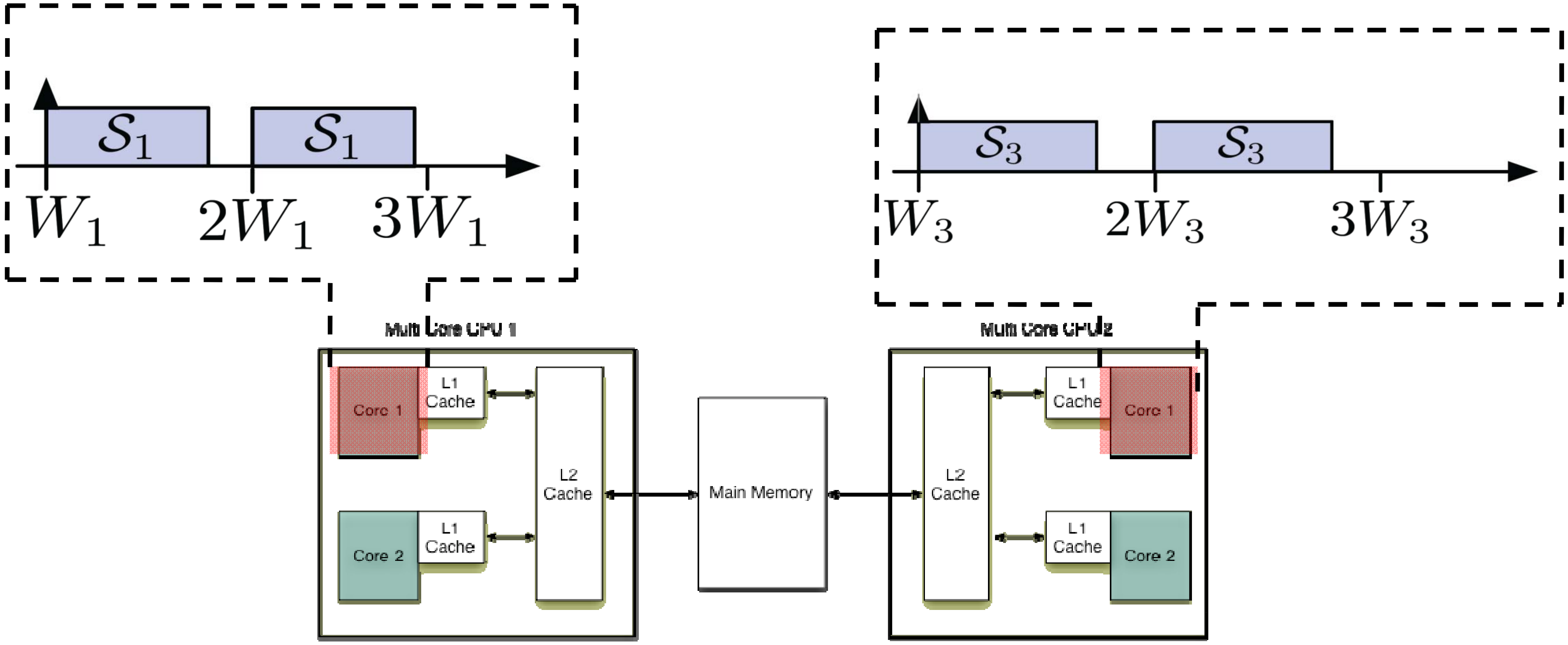


- 2 Models to execute superblocks:
  - Sequential
  - Time-triggered (superblocks, phases)

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

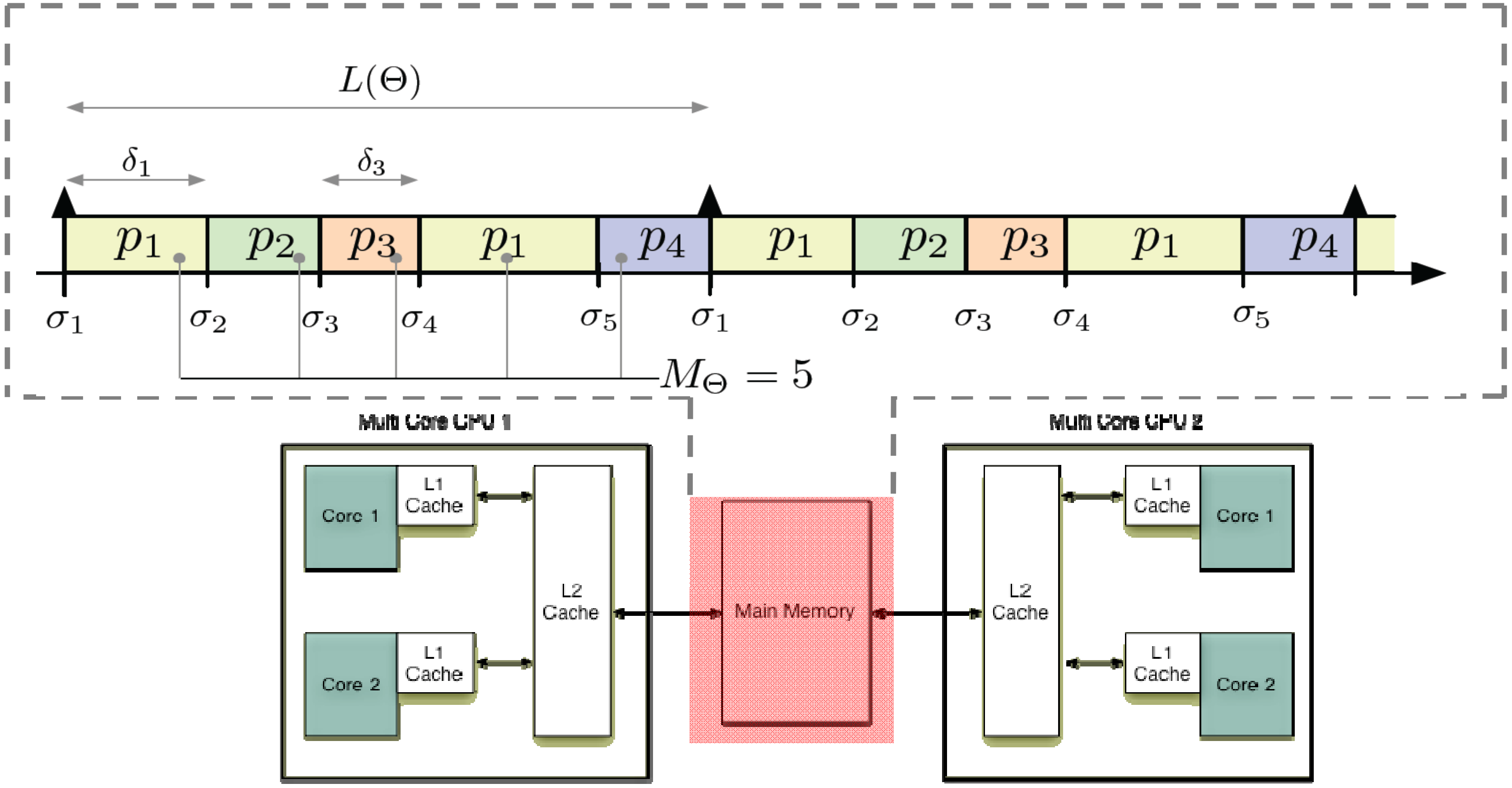
Computer Engineering and
Networks Laboratory

Static execution on the processing element



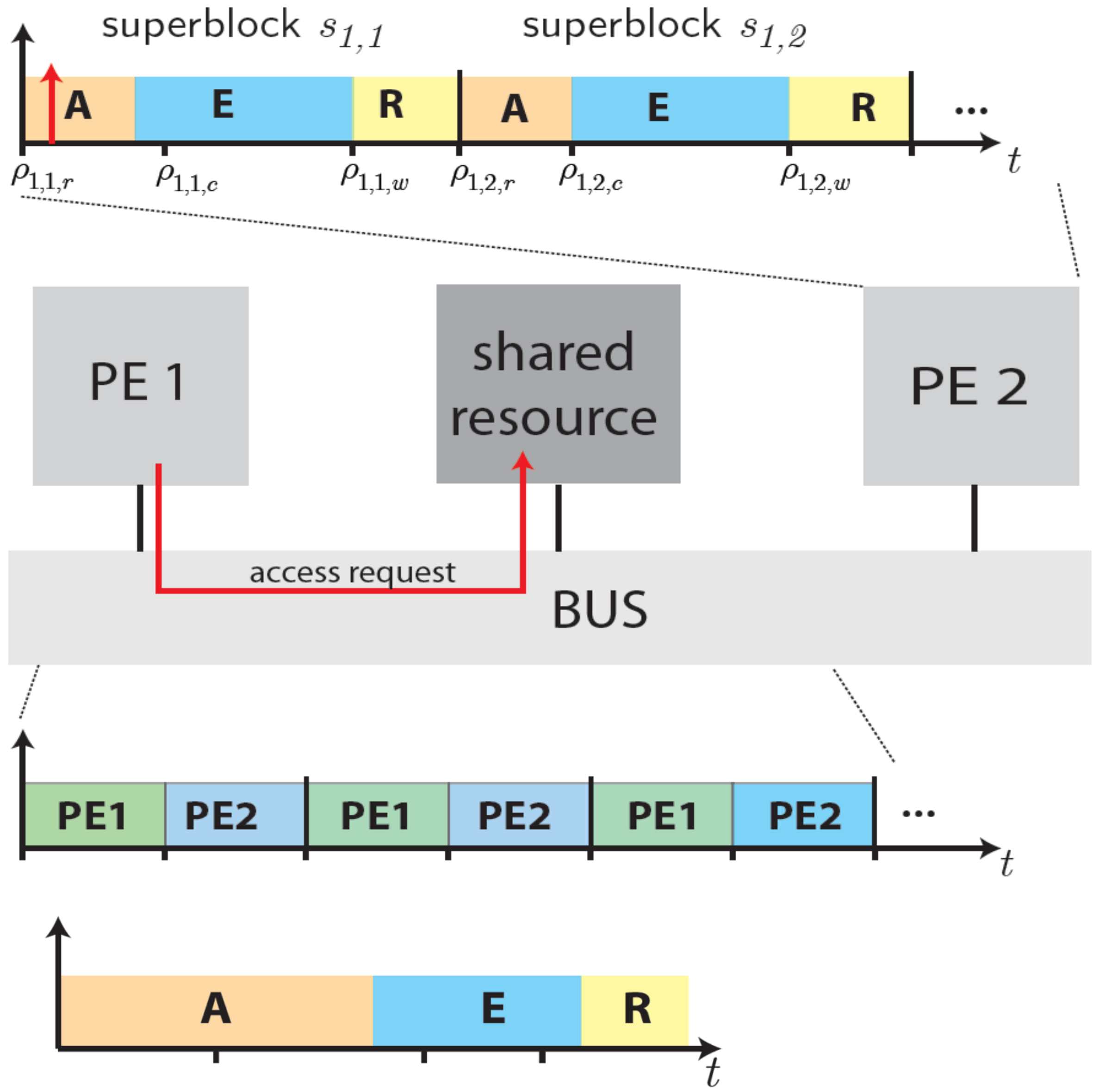
TDMA on the shared resource

Independence between tasks **single source of interference**


$L(\Theta)$
δ_1
δ_3
p_1 p_2 p_3 p_1 p_4 p_1 p_2 p_3 p_1 p_4
σ_1 σ_2 σ_3 σ_4 σ_5 σ_1 σ_2 σ_3 σ_4 σ_5
$M_\Theta = 5$
Multi Core CPU 1
Core 1
L1 Cache
Core 2
L1 Cache
L2 Cache
Main Memory
Multi Core CPU 2
L2 Cache
L1 Cache
Core 1
L1 Cache
Core 2

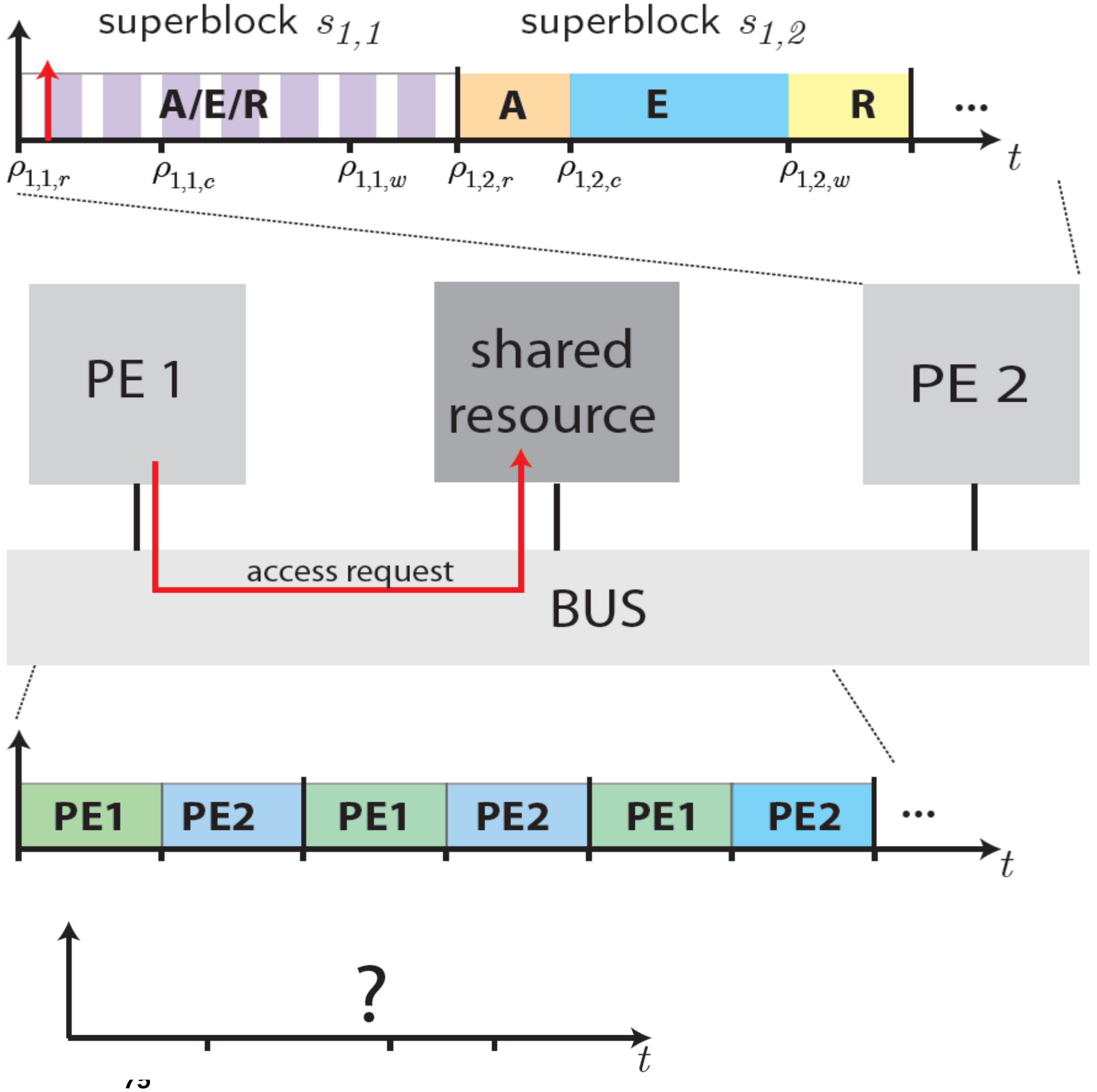
Static Arbitration (1)

- Generate worst-case trace
- Read/Write access whenever slot is active
- Execution is performed immediately
- Assume PE2 grants access:



Static Arbitration (2)

- Generate worst-case trace
- Where to place the access requests ?
- Algorithm for WCCT by maximizing stalling
- Assume PE2 grants access:



Analysis for static arbitration - Summary

- analysis is complex
  - makes use of arrival and service curves (real-time calculus)
  - has been extended to dynamic resource sharing as well
- analysis handles dedicated and general phases
  - sequential and time-triggered execution
- analysis of mixed models possible by composition
  - superblocks can be specified using different models
- Time complexity
  - Dedicated phase: $O(M_{\mathrm{Q}})$
  - General phase: $O(M_{\mathrm{Q}} \log exec^{max})$

Resource Access Models (1)

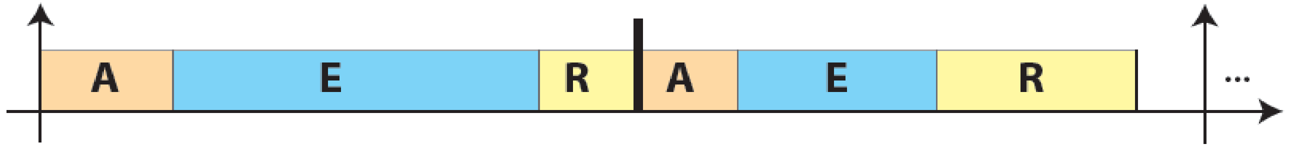
- Influence of different access models on schedulability
- Influence of the execution model on predictability (equivalent WCRT)

- Intuition:
  - Separation of resource access and computation increases predictability
  - Everything time-triggered increases predictability

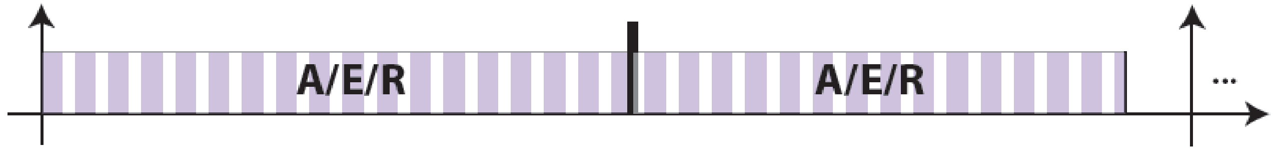
?

Resource Access Models (2) - Reminder

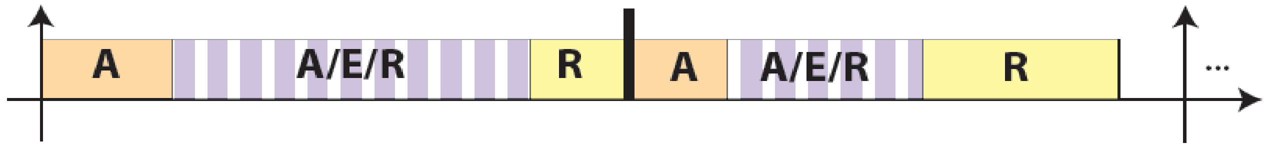
- 3 Models to specify resource accesses:
  - Dedicated Model



  - General Model

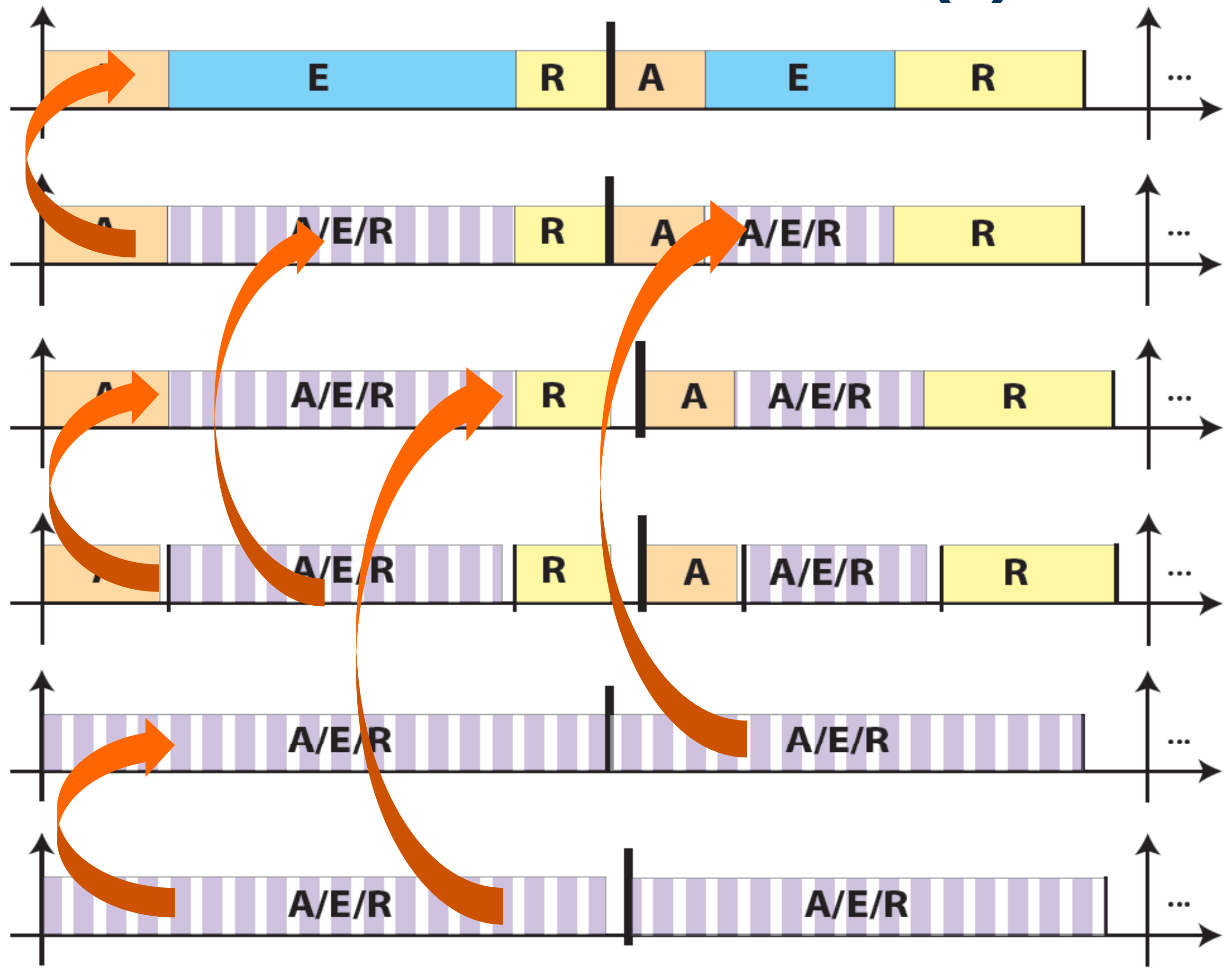


  - Hybrid Model



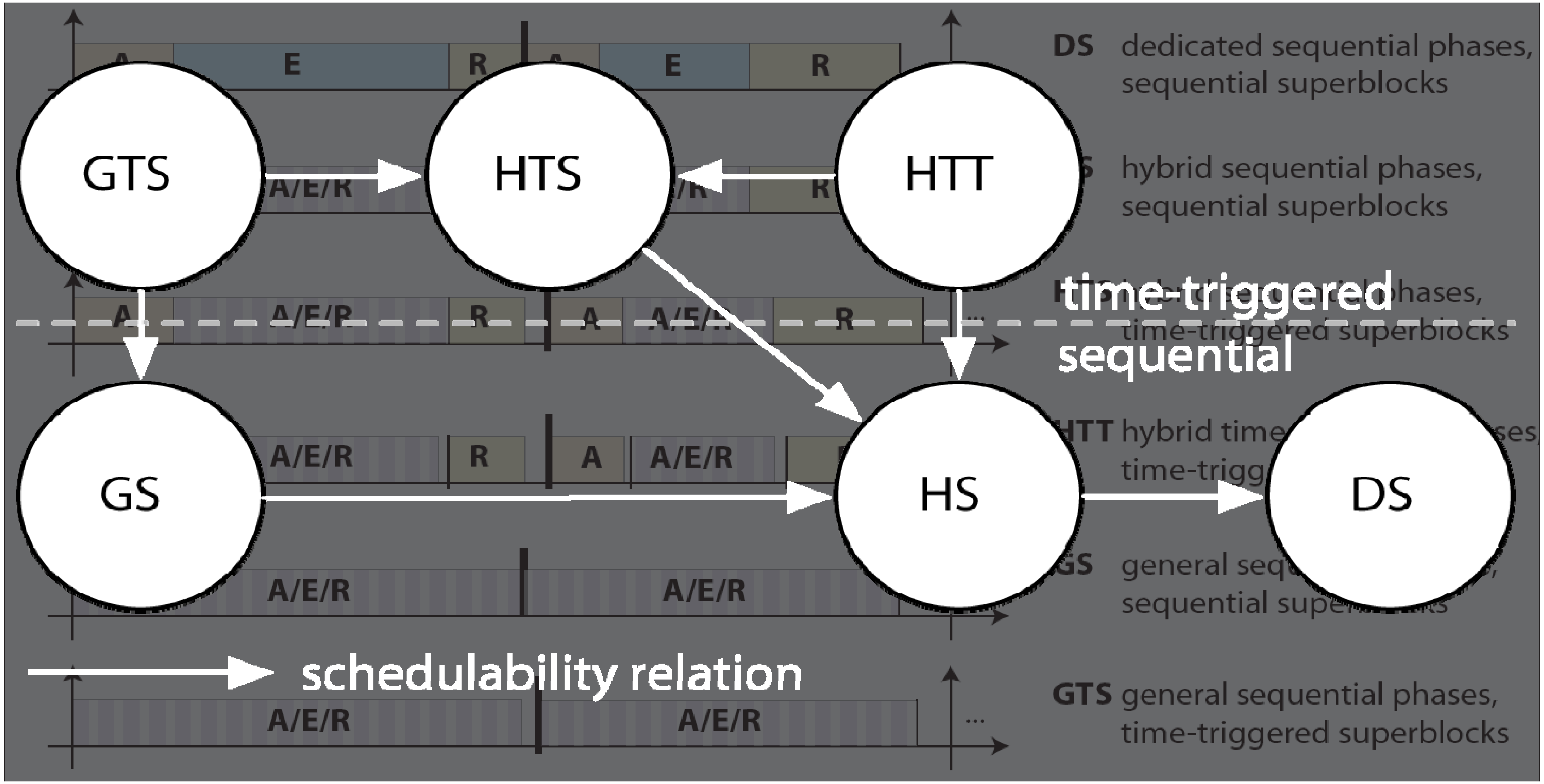
- 2 Models to execute superblocks:
  - Sequential
  - Time-triggered

Resource Access Models (3)

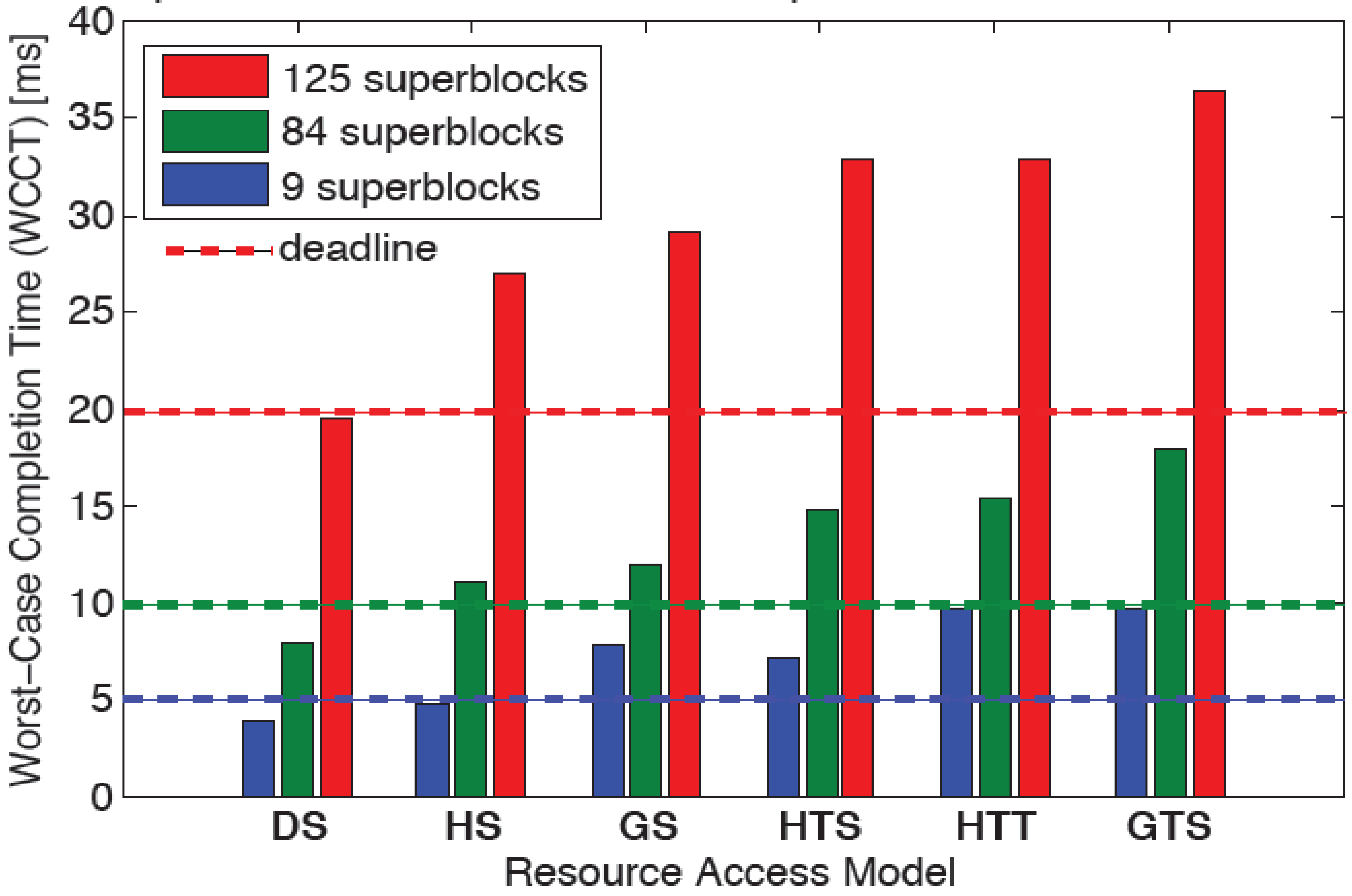


DS	dedicated sequential phases, sequential superblocks
HS	hybrid sequential phases, sequential superblocks
HTS	hybrid sequential phases, time-triggered superblocks
HTT	hybrid time-triggered phases time-triggered superblocks
GS	general sequential phases, sequential superblocks
GTS	general sequential phases, time-triggered superblocks

Schedulability between Models


GTS
HTS
HTT
GS
HS
DS
time-triggered
sequential
schedulability relation
DS dedicated sequential phases, sequential superblocks
HS hybrid sequential phases, sequential superblocks
HTT hybrid time-triggered phases, time-triggered superblocks
GS general sequential phases, sequential superblocks
GTS general sequential phases, time-triggered superblocks

Experimental Results for 3 sets of superblocks and different models



Comparisons of Access Models

- Intuition:
  - Separation of resource access and computation increases predictability ✓
  - Everything time-triggered increases predictability 🚫

?

- Excessive time-triggering may degrade performance
- No advantage in terms of predictability
- Model DS is model choice for resource sharing systems
  - Separate Memory Access and Computation

Conclusion

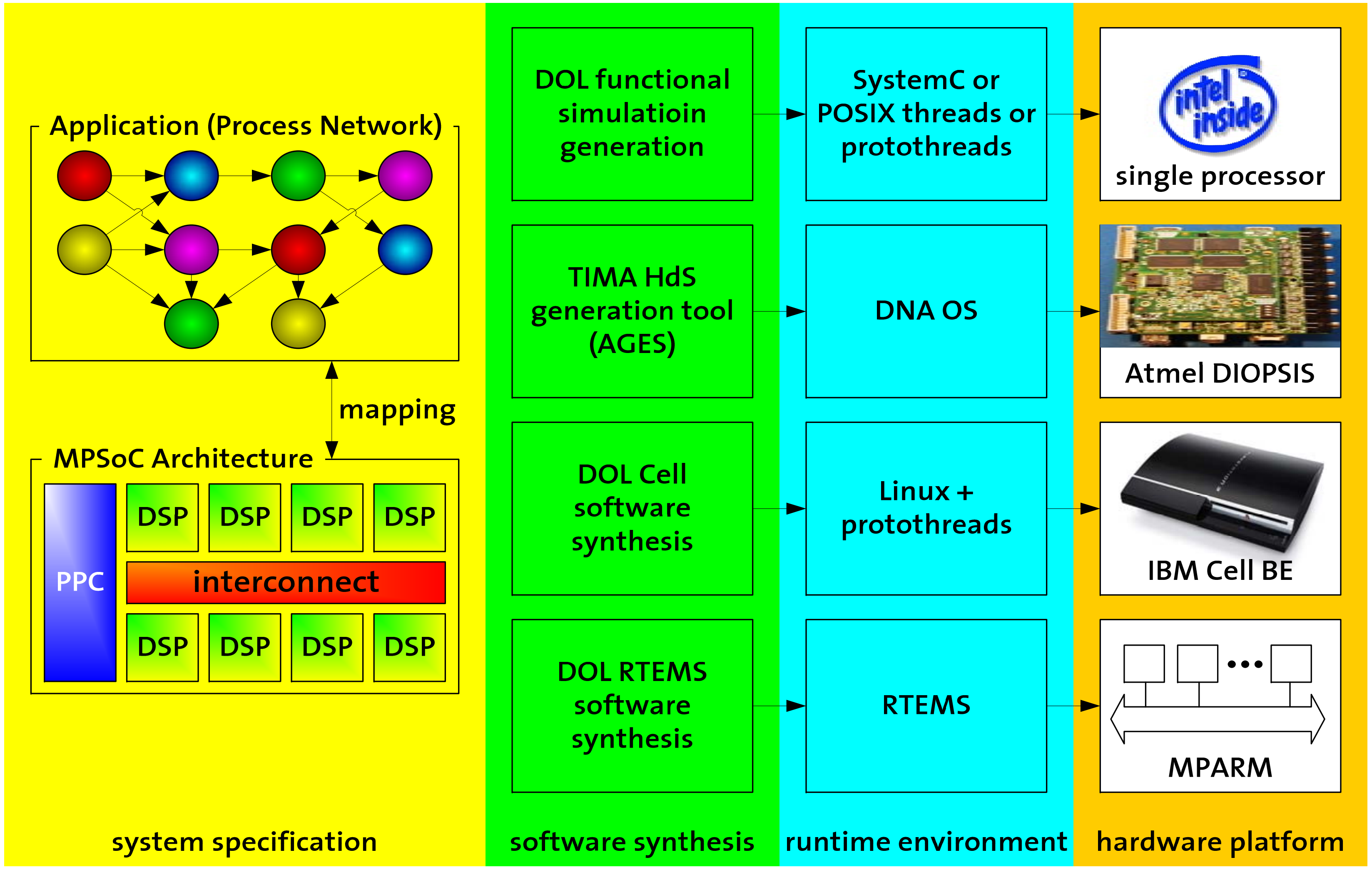
- Resource Sharing in Multi-Core Systems is an important issue in terms of
  - Analyzability
  - Predictability
  - Efficiency
- Static arbitration policies
  - Elimination of Interference
  - Tight bounds on WCCT can be derived
- Excessive time-triggering is counter productive

Even for simple models:
Resource Sharing is a hard Problem

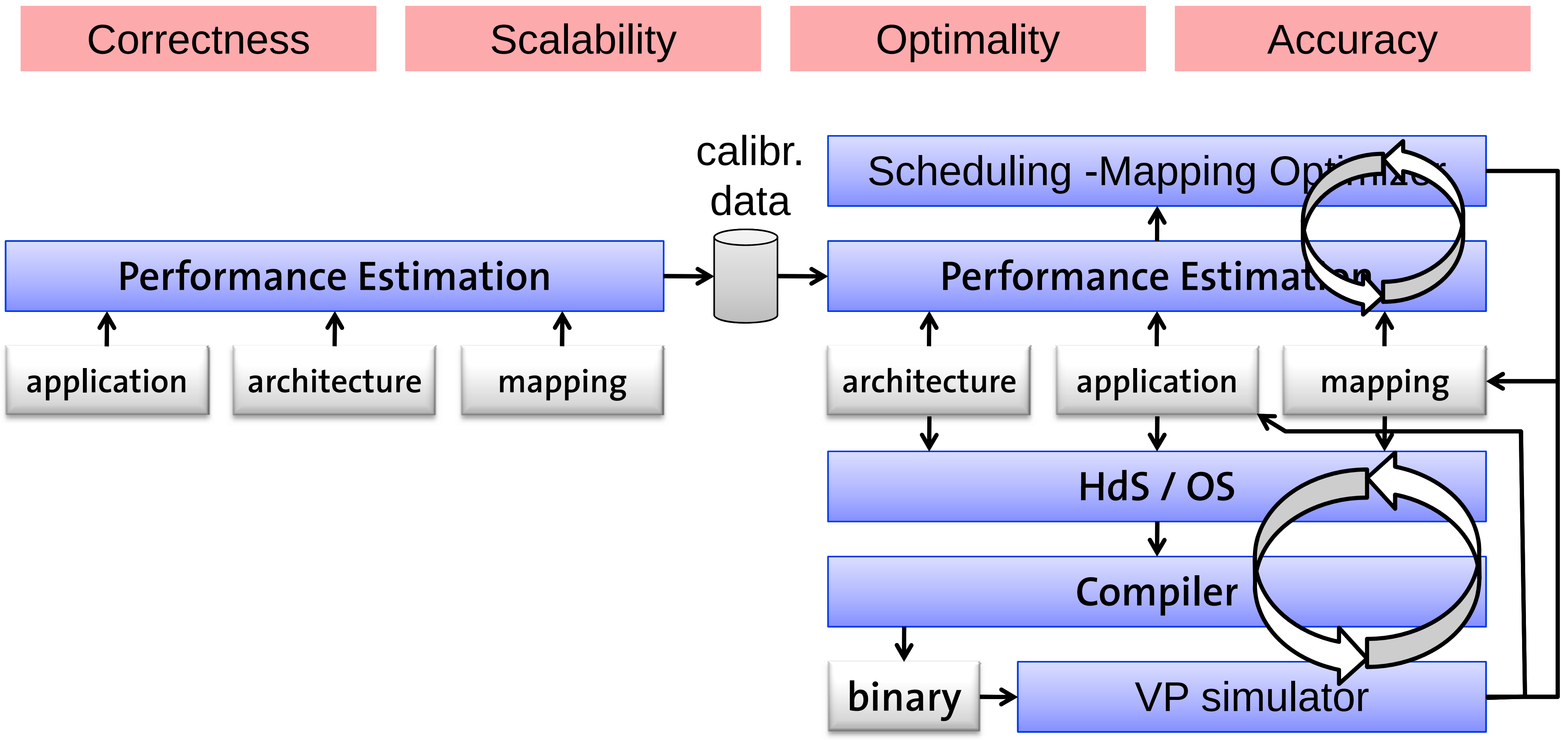
Architectural Interactions

- Compilation for Multiprocessors-

Versatile MPSoC Software Design Flow



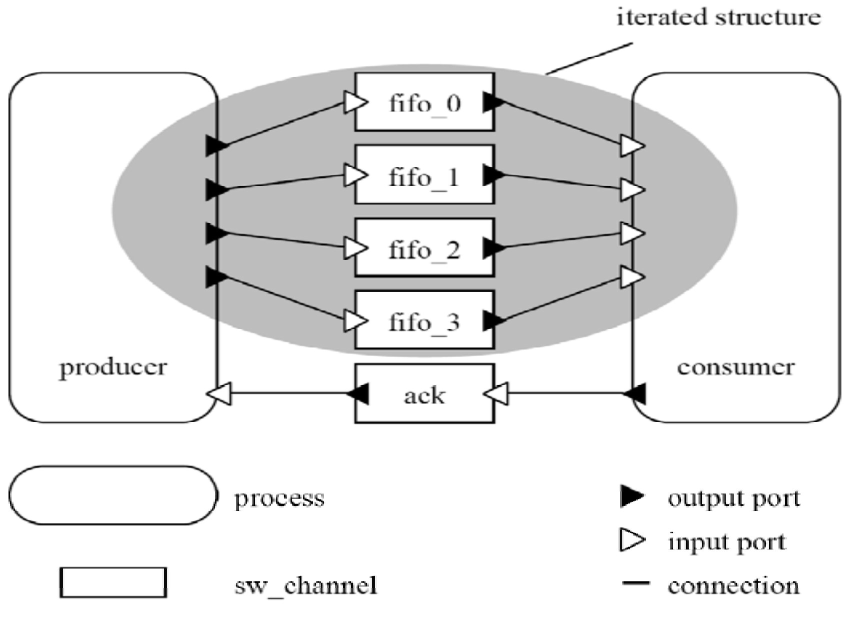
DOL Design Flow



Application Specification

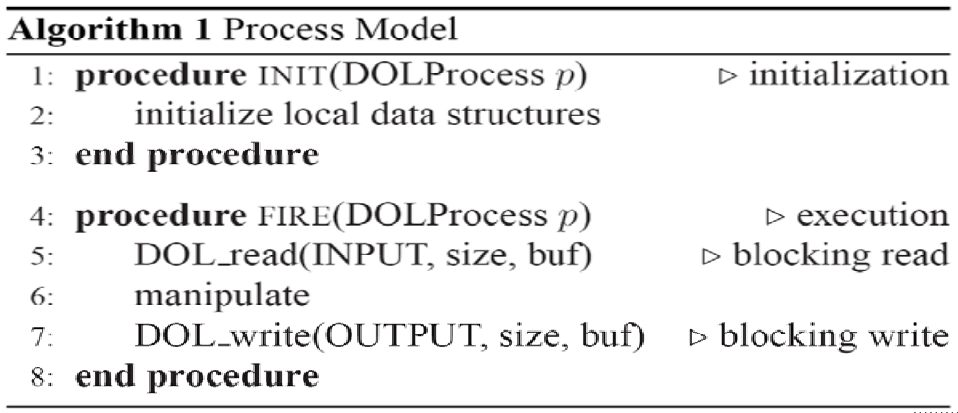
Structure

- **Process Network**
  - Processes
  - SW channels (FIFO behavior)
- **Iterators**
  - Scalability for processes, SW channels, entire structures



Functional specification

- **Language:** C/C++
- **API:** DOL primitives

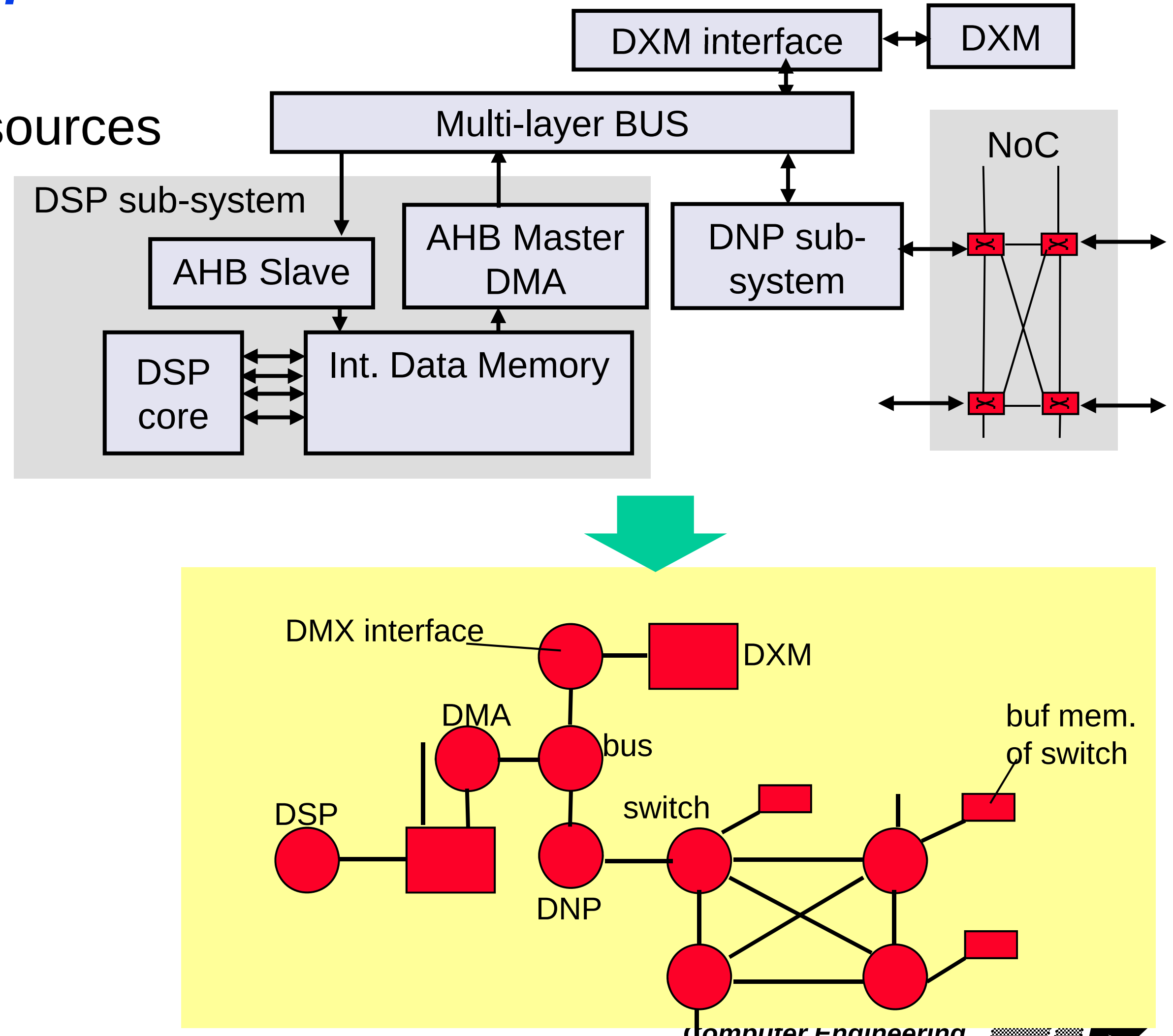


Algorithm 1 Process Model

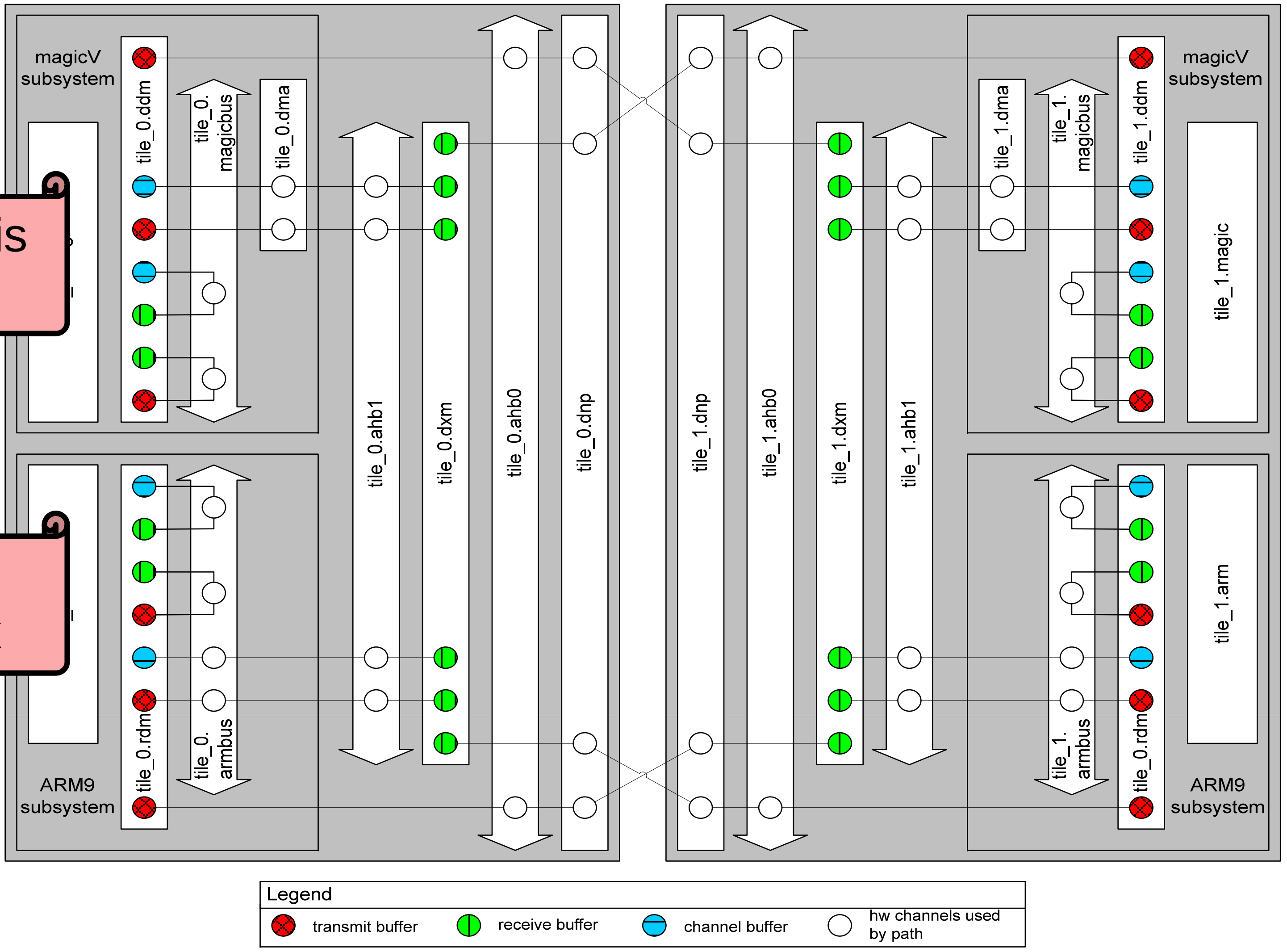
```
1: procedure INIT(DOLProcess p)                      ▷ initialization
2:     initialize local data structures
3: end procedure

4: procedure FIRE(DOLProcess p)                      ▷ execution
5:     DOL_read(INPUT, size, buf)                    ▷ blocking read
6:     manipulate
7:     DOL_write(OUTPUT, size, buf)                  ▷ blocking write
8: end procedure
```

Target Platform Abstraction (1)

- ***Topology modeled by a graph***
  - two node types:
    - execution and comm. resources
    - storage resources
- ***Execution resources***
  - RISCs, DSPs, …
- ***Communication resources***
  - buses, switches, links, I/Os
- ***Storage resources***
  - RAMs, HW FIFOs, …



Target Platform Abstraction (2)

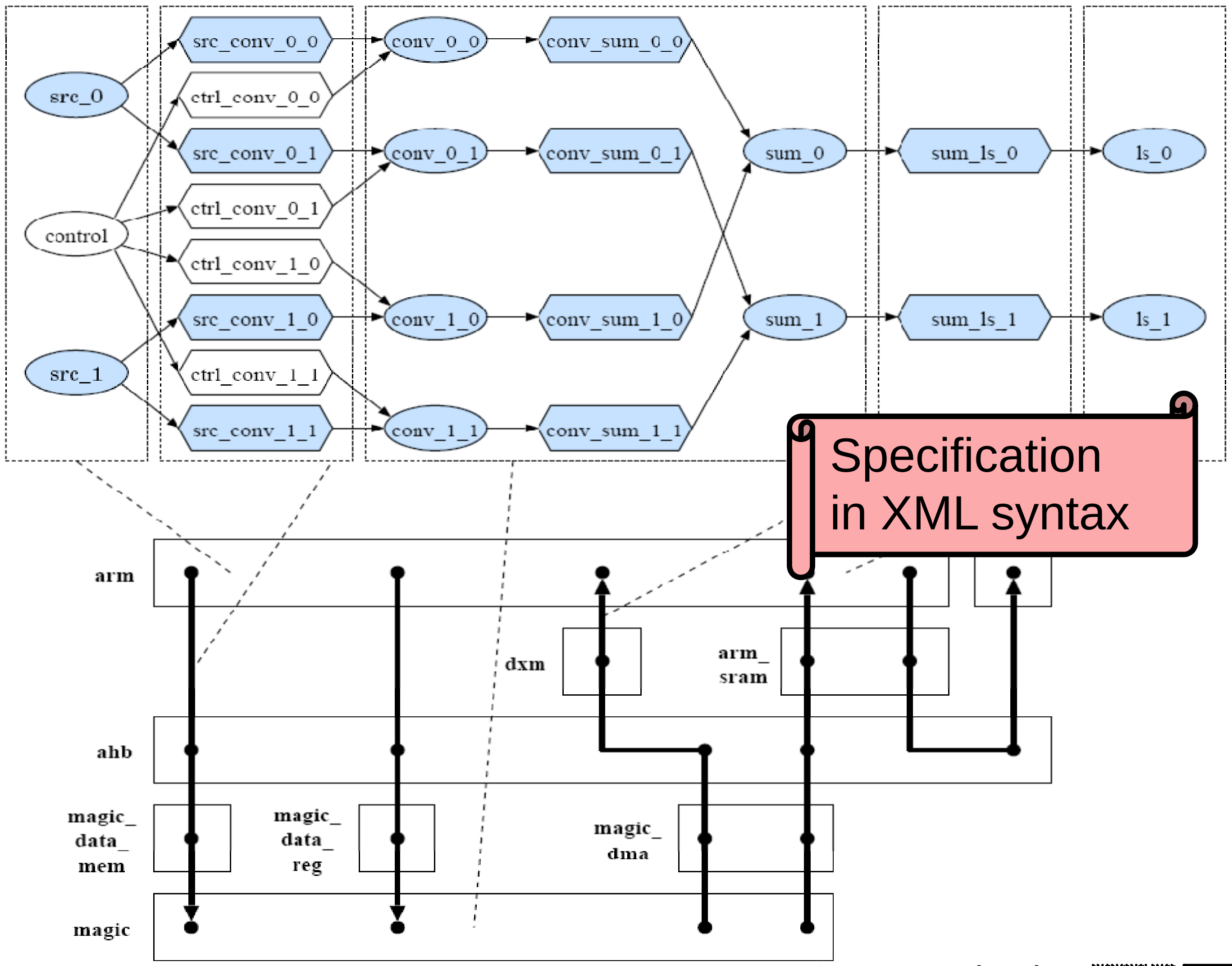


ATMEL Diopsis Platform

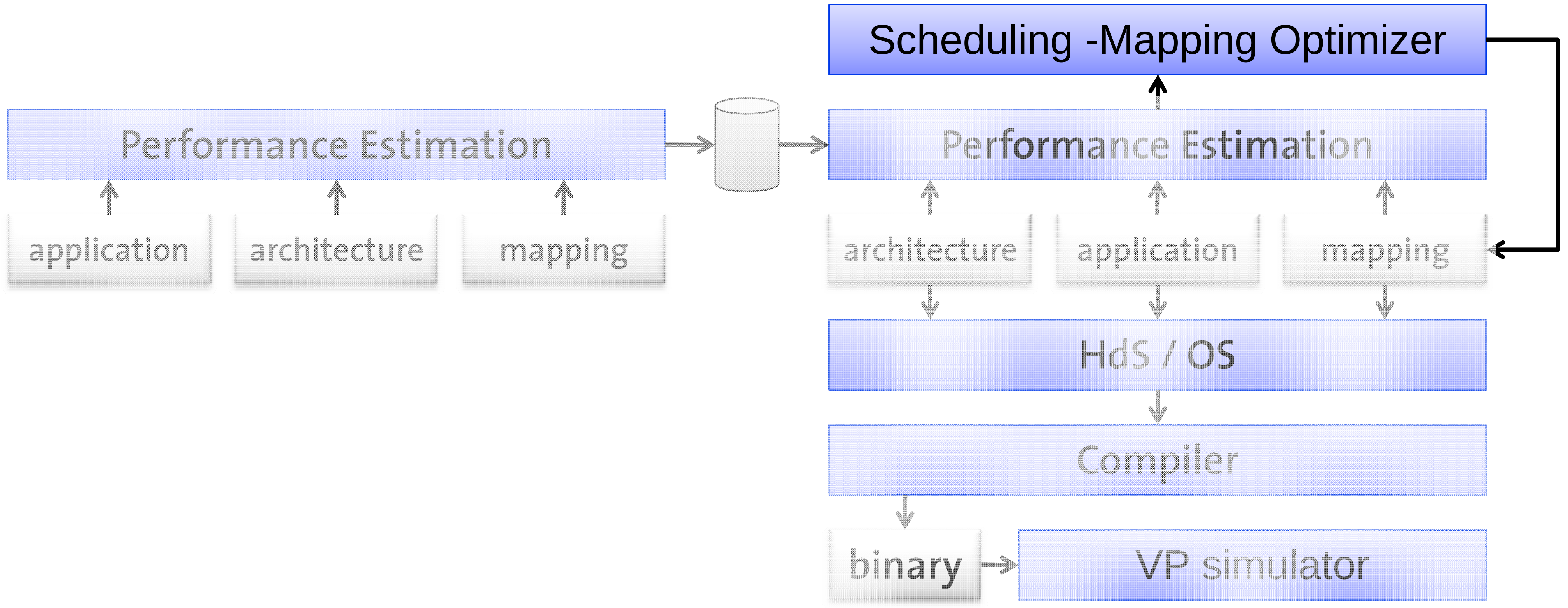
Specification in XML syntax

Mapping Specification

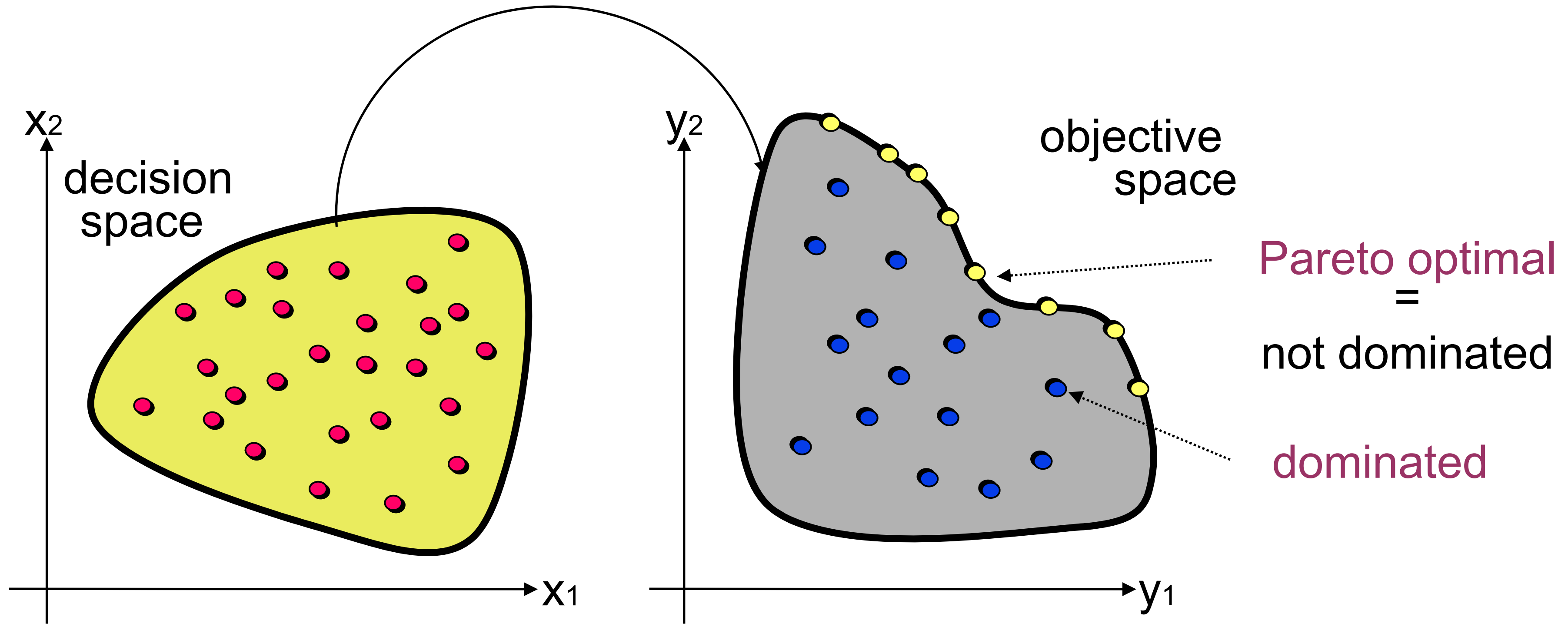
- **Binding**
  - Processes to execution resources
  - SW channels to read/write paths
- **Scheduling**
  - Processors
  - Communication
- **Constraints**
  - For Hardware-dependent Software (HdS) generation



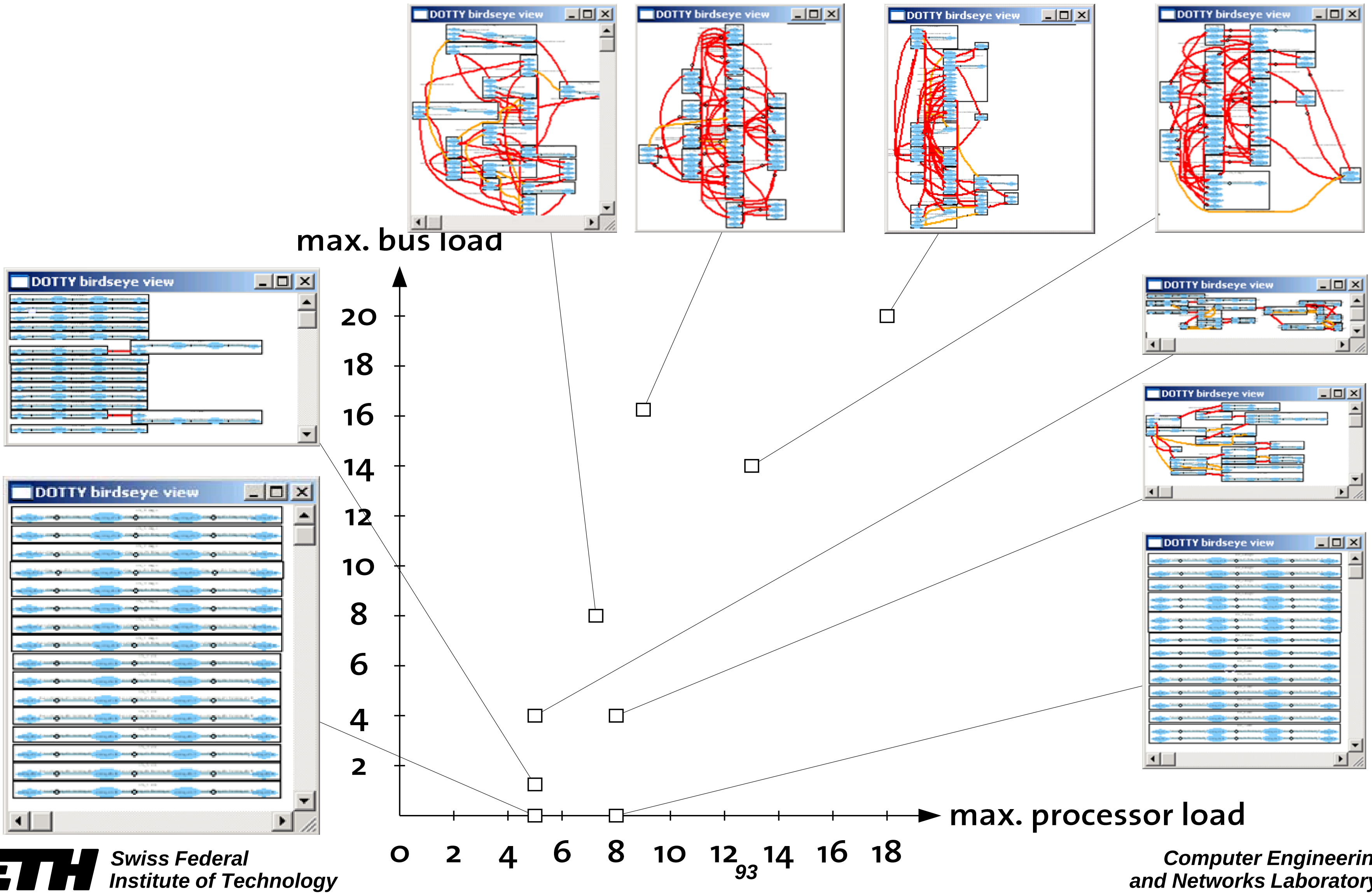
DOL Design Flow



Multiobjective Optimization



Design Space Exploration



Design Space Exploration

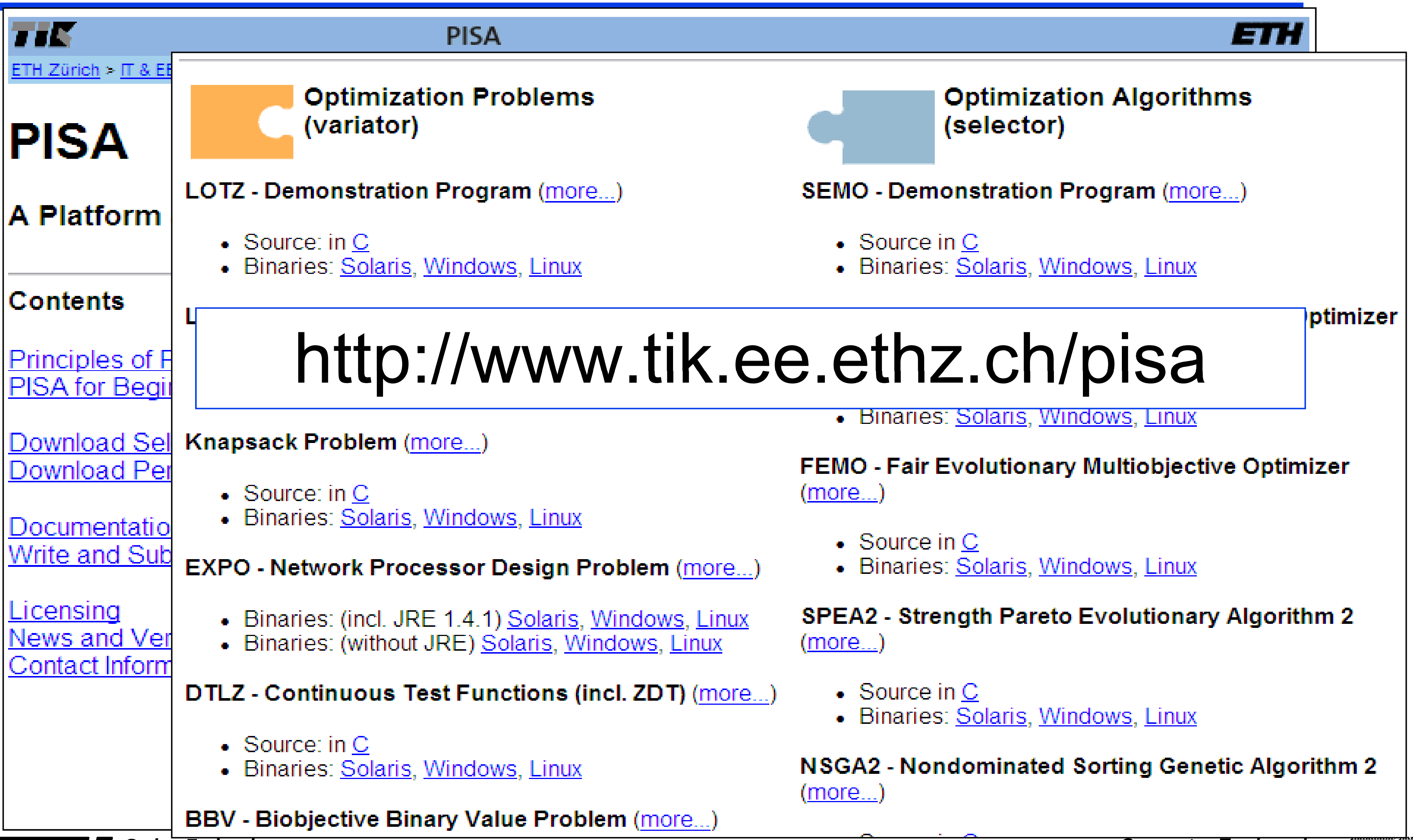
- **Example for ATMEL Multitile Platform**:
  - 64 processes, 16 processors, optimal mapping known
  - 32 processes execute efficiently on ARM, 32 efficiently on mAgic
  - different interconnection structures between processes
  - $16^{64} \approx 1.15 \cdot 10^{77}$ possible mappings (including symmetric ones)
  - Evaluation of 10.000 mappings

	naive	evolutionary algorithm	optimum
64 indep. tasks	7/0	4/0	4/0
16 4-stage pipelines	9/10	4/0	4/0
64-stage pipeline	9/14	4/4	4/2

max. processor load

max. interconnect load

PISA Website



PISA

ETH Zürich > IT & EE

PISA

A Platform

Contents

Principles of P
PISA for Begi

Download Sel
Download Per

Documentatio
Write and Sub

Licensing
News and Ver
Contact Inform

Optimization Problems (variator)

LOTZ - Demonstration Program (more...)

- Source: in C
- Binaries: Solaris, Windows, Linux

L

Knapsack Problem (more...)

- Source: in C
- Binaries: Solaris, Windows, Linux

EXPO - Network Processor Design Problem (more...)

- Binaries: (incl. JRE 1.4.1) Solaris, Windows, Linux
- Binaries: (without JRE) Solaris, Windows, Linux

DTLZ - Continuous Test Functions (incl. ZDT) (more...)

- Source: in C
- Binaries: Solaris, Windows, Linux

BBV - Biobjective Binary Value Problem (more...)

Optimization Algorithms (selector)

SEMO - Demonstration Program (more...)

- Source in C
- Binaries: Solaris, Windows, Linux

ptimizer

- Binaries: Solaris, Windows, Linux

FEMO - Fair Evolutionary Multiobjective Optimizer (more...)

- Source in C
- Binaries: Solaris, Windows, Linux

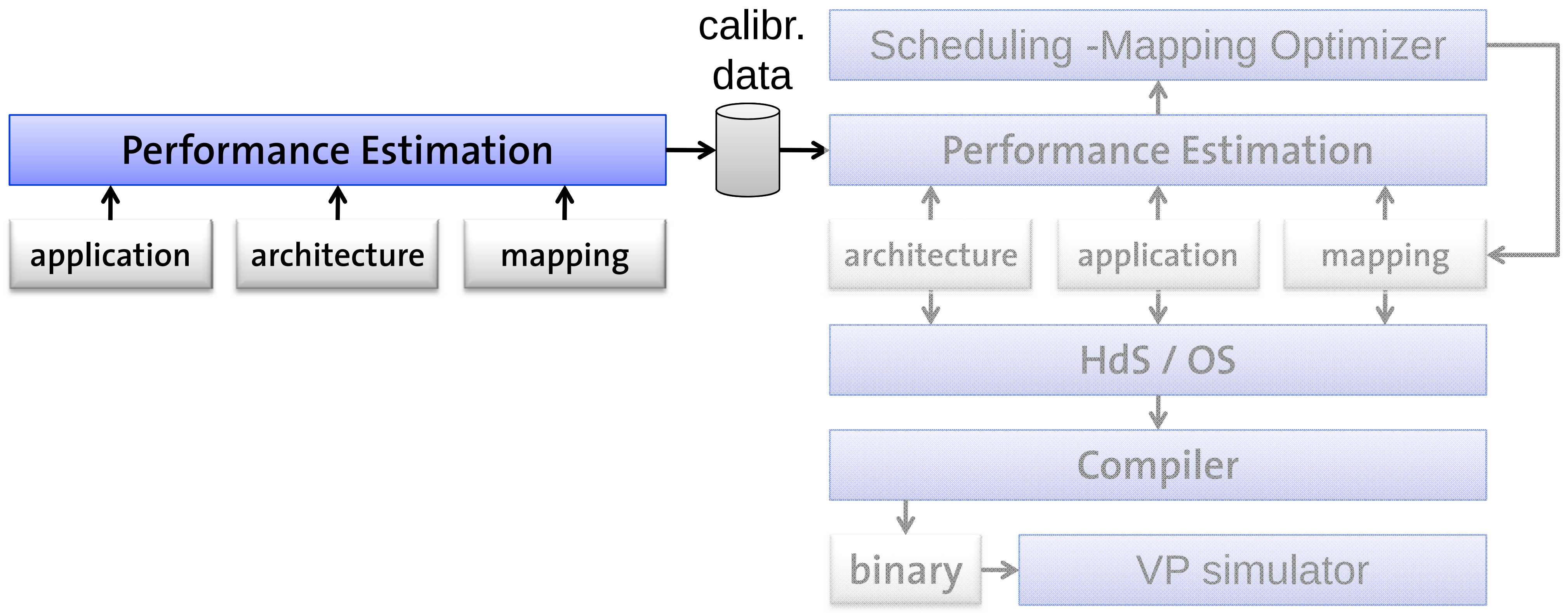
SPEA2 - Strength Pareto Evolutionary Algorithm 2 (more...)

- Source in C
- Binaries: Solaris, Windows, Linux

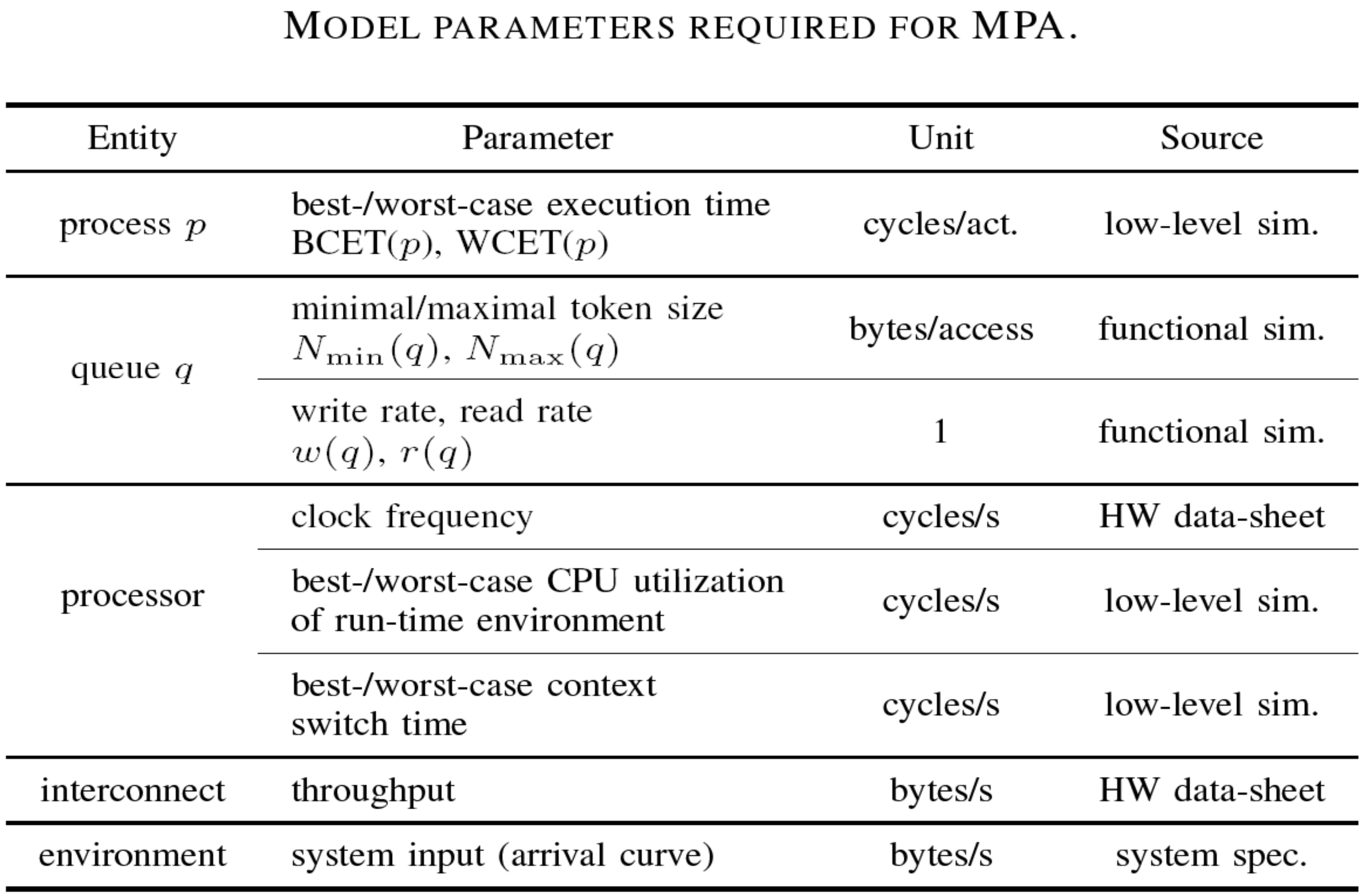
NSGA2 - Nondominated Sorting Genetic Algorithm 2 (more...)

http://www.tik.ee.ethz.ch/pisa

DOL Design Flow



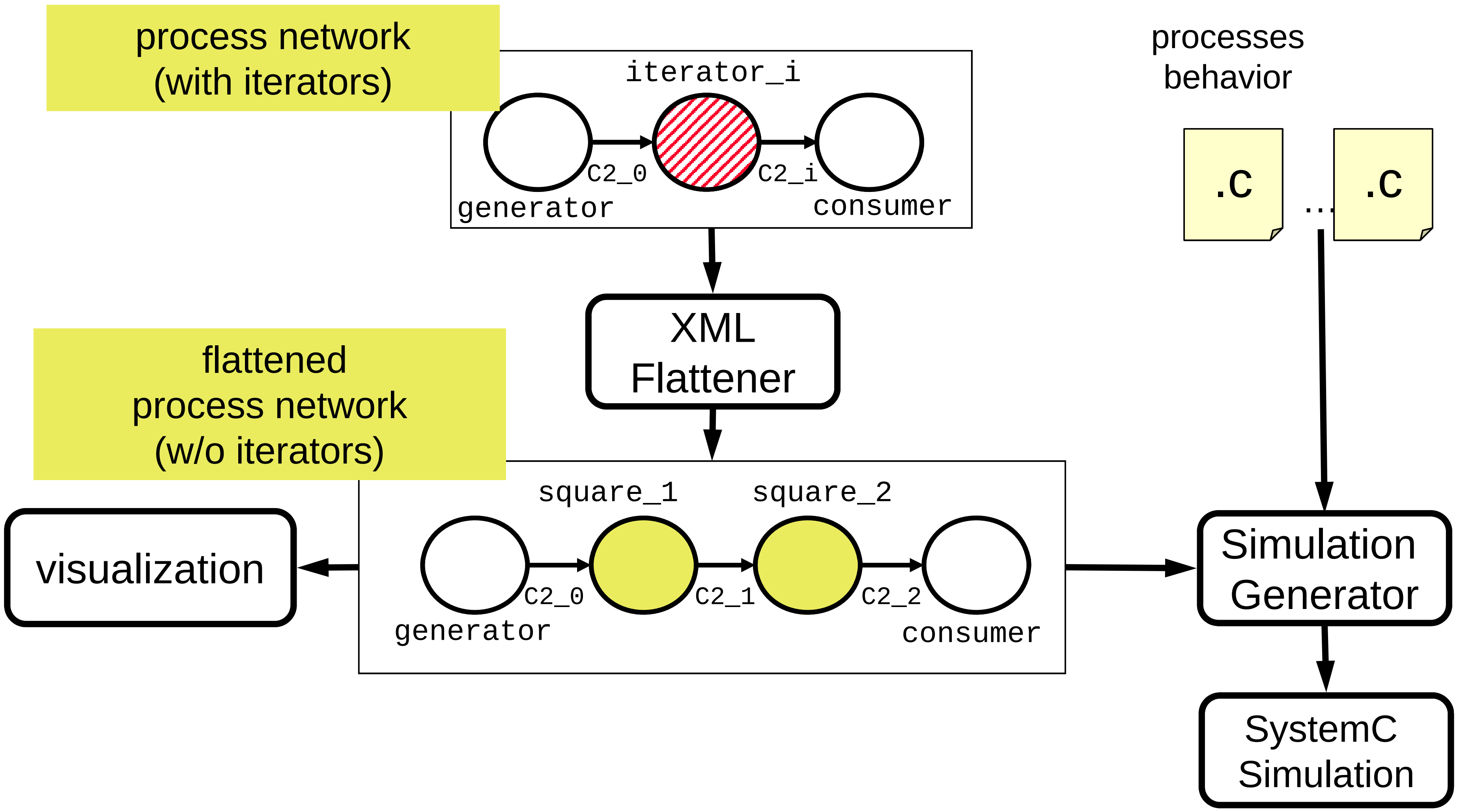
Multitile Calibration



MODEL PARAMETERS REQUIRED FOR MPA.

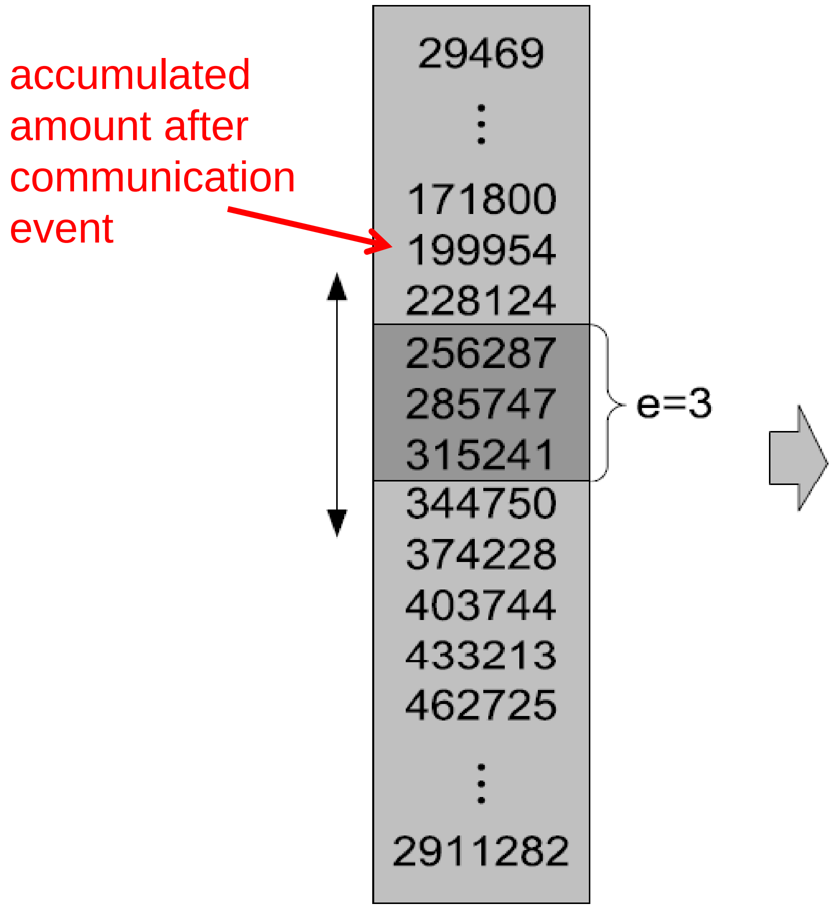
Entity	Parameter	Unit	Source
process p	best-/worst-case execution time BCET(p), WCET(p)	cycles/act.	low-level sim.
queue q	minimal/maximal token size $N_{\min}(q)$, $N_{\max}(q)$	bytes/access	functional sim.
	write rate, read rate $w(q)$, $r(q)$	1	functional sim.
processor	clock frequency	cycles/s	HW data-sheet
	best-/worst-case CPU utilization of run-time environment	cycles/s	low-level sim.
	best-/worst-case context switch time	cycles/s	low-level sim.
interconnect	throughput	bytes/s	HW data-sheet
environment	system input (arrival curve)	bytes/s	system spec.

Functional Simulation

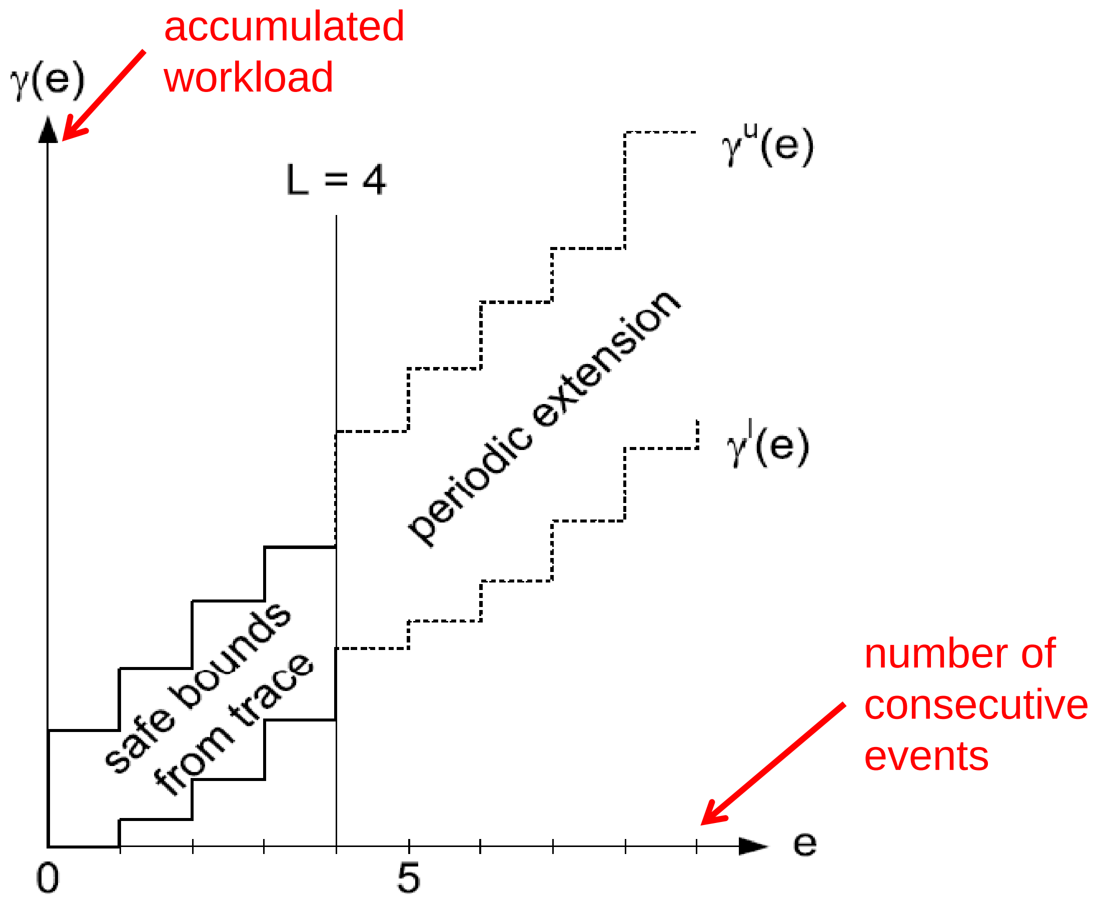


Workload Extraction

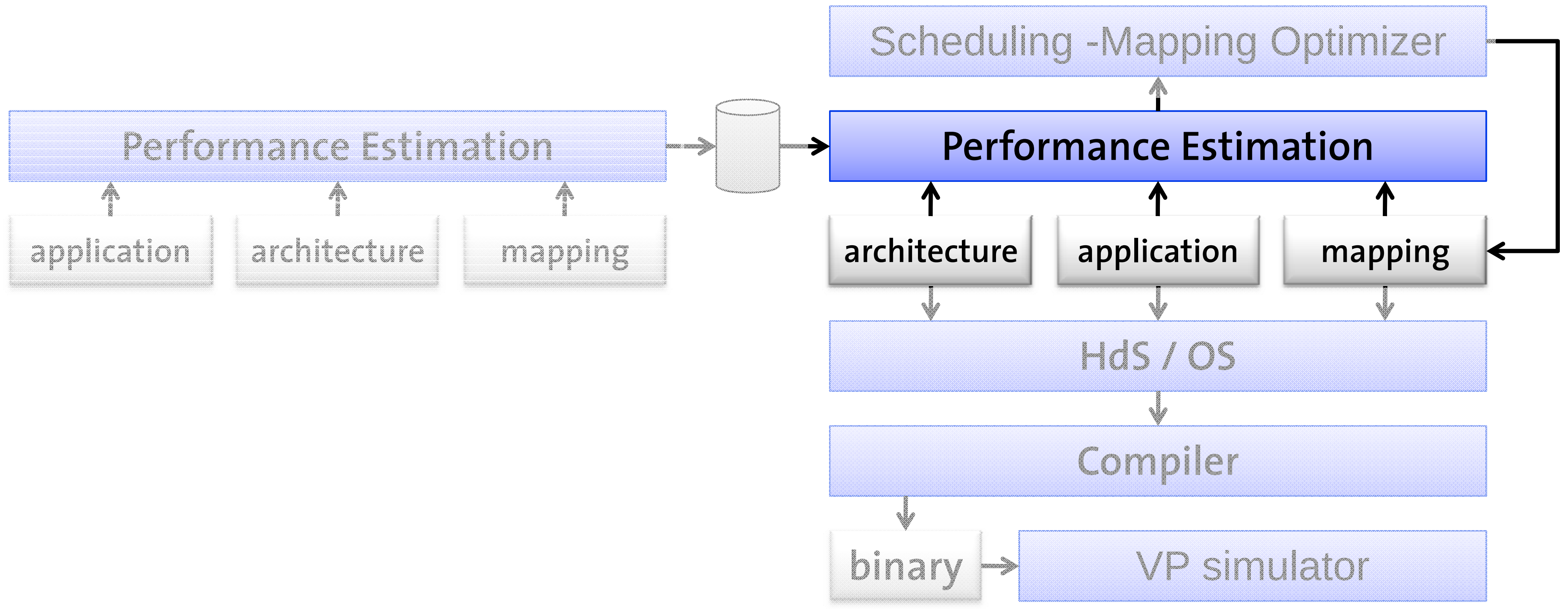
functional simulation



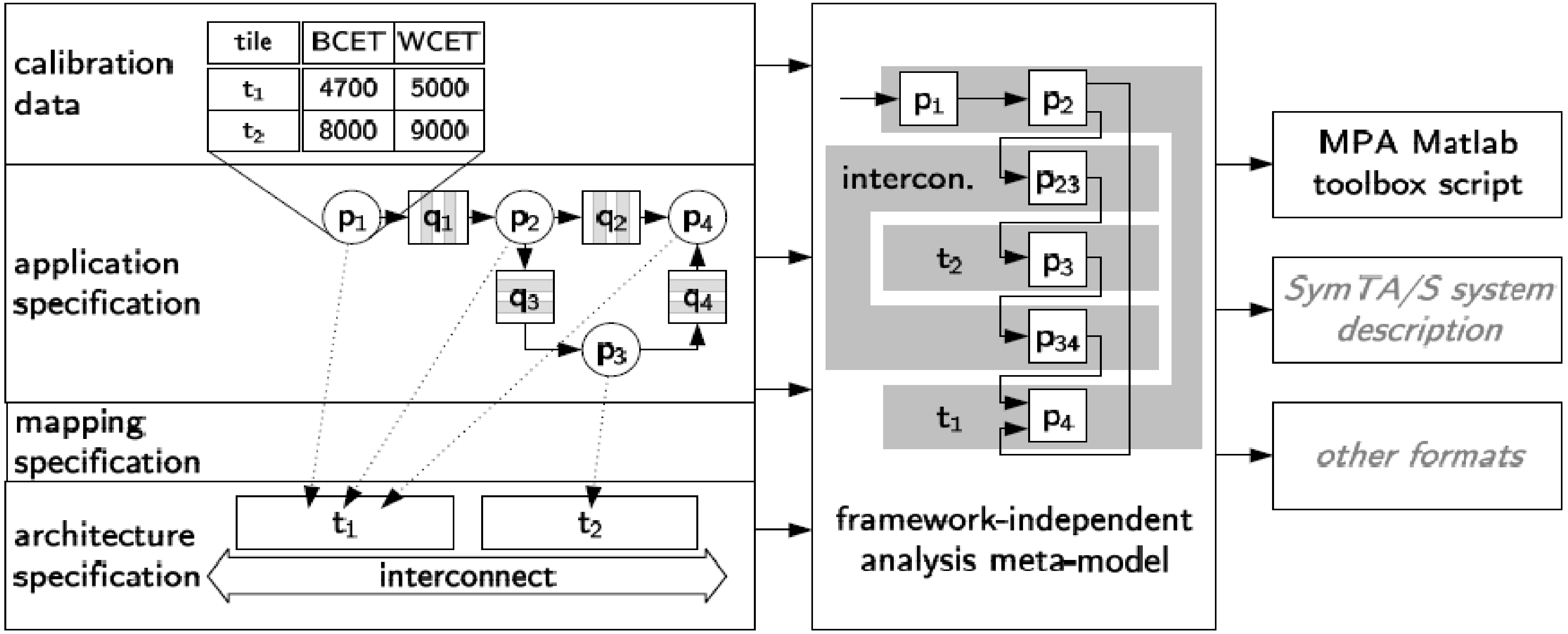
workload bounds



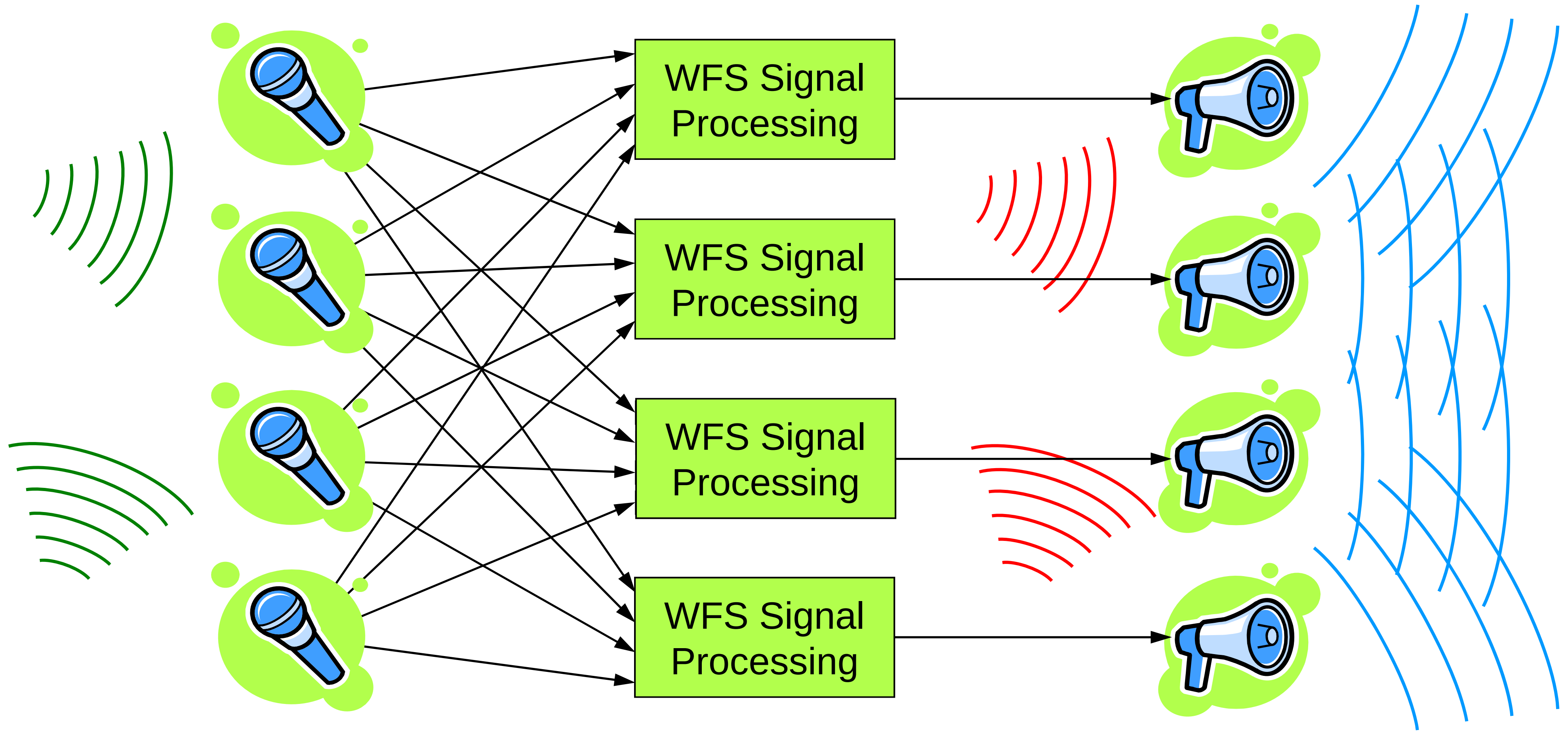
DOL Design Flow



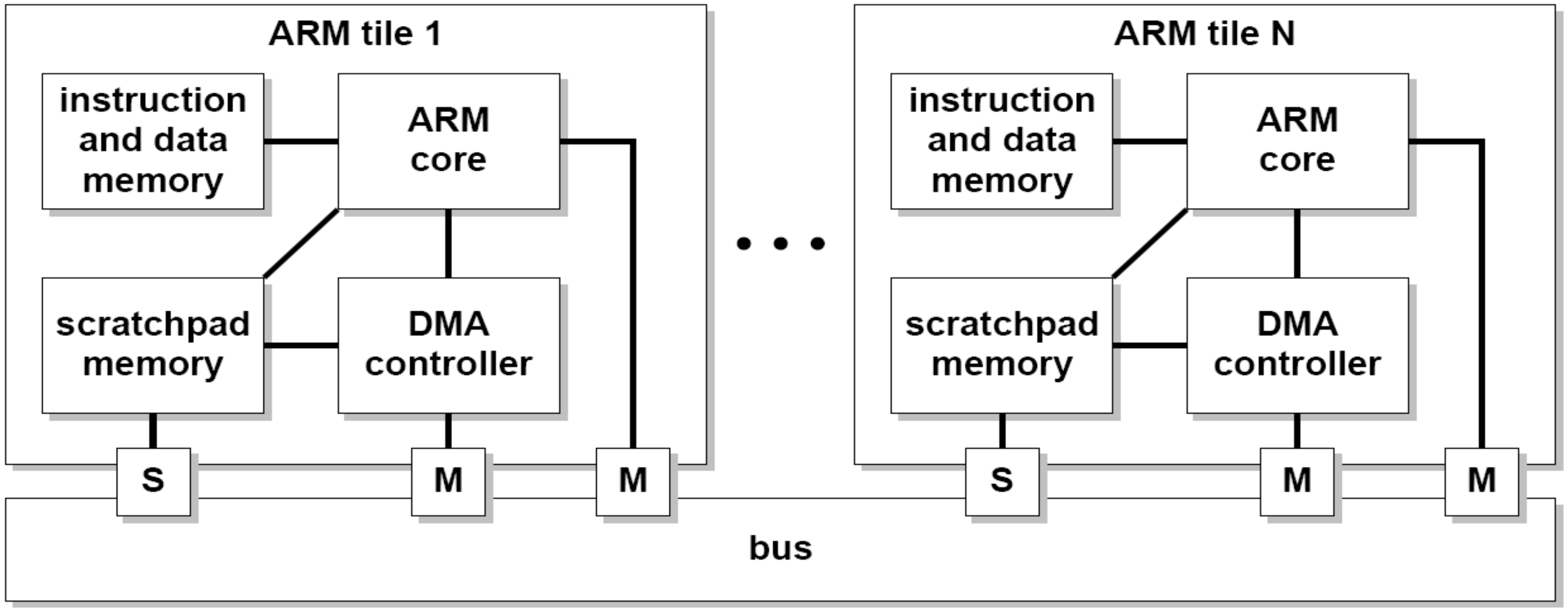
Integration


calibration data
tile
BCET
WCET
t1
4700
5000
t2
8000
9000
application specification
p1
q1
p2
q2
p4
q3
q4
p3
mapping specification
architecture specification
t1
t2
interconnect
p1
p2
intercon.
p23
t2
p3
p34
t1
p4
framework-independent analysis meta-model
MPA Matlab toolbox script
SymTA/S system description
other formats

Example: Wave Field Synthesis



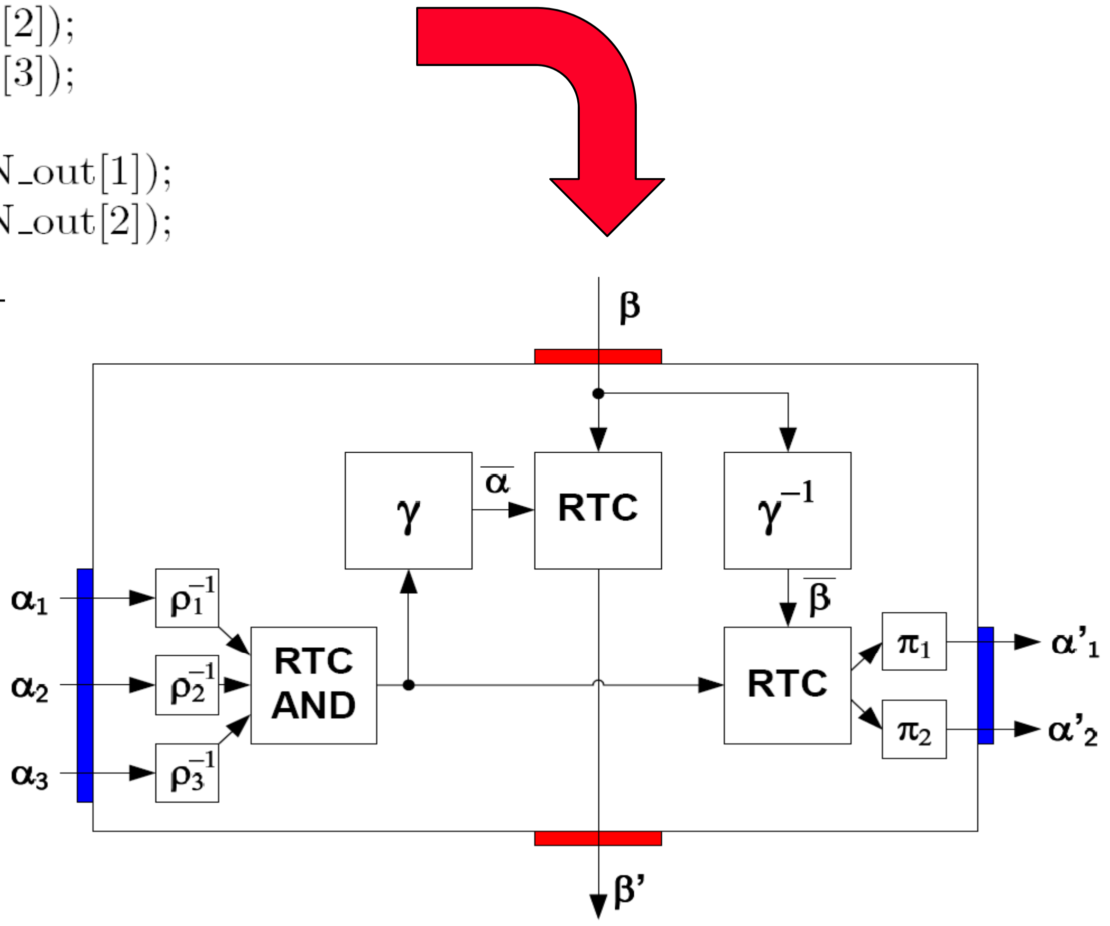
Example: Architecture Template



Example: Application Modeling

Algorithm 1 Example of a process with multiple inputs and outputs.

```
1: function FIRE(DOLProcess *p)
2:     DOL_read(input[1], buffer_in[1], N_in[1]);
3:     DOL_read(input[2], buffer_in[2], N_in[2]);
4:     DOL_read(input[3], buffer_in[3], N_in[3]);
5:     execute;
6:     DOL_write(output[1], buffer_out[1], N_out[1]);
7:     DOL_write(output[2], buffer_out[2], N_out[2]);
8: end function
```

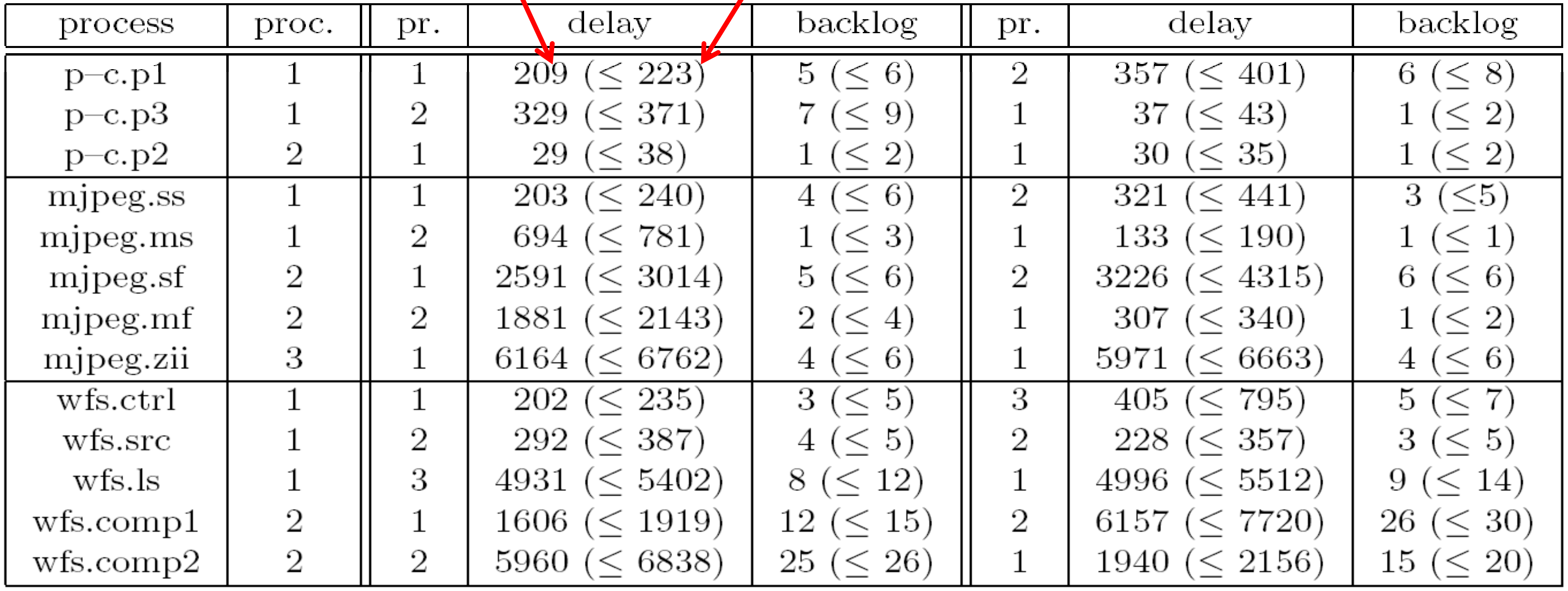


Example: Compilation Times

step		duration		
		P–C	MJPEG	WFS
model calibration (one-time effort)	functional simulation generation	22 s	42 s	35 s
	functional simulation	0.2 s	3.6 s	2.4 s
	synthesis (generation of binary)	2 s	4 s	3 s
	simulation on MPARM	23 s	13550 s	740 s
	log-file analysis and back-annotation	1 s	12 s	3 s
model generation		1 s	1 s	1 s
performance analysis based on generated model		0.2 s	2.5 s	1.4 s

Example: Accuracy

observed estimated



process	proc.	pr.	delay	backlog	pr.	delay	backlog
p–c.p1	1	1	209 ($\leq$ 223)	5 ($\leq$ 6)	2	357 ($\leq$ 401)	6 ($\leq$ 8)
p–c.p3	1	2	329 ($\leq$ 371)	7 ($\leq$ 9)	1	37 ($\leq$ 43)	1 ($\leq$ 2)
p–c.p2	2	1	29 ($\leq$ 38)	1 ($\leq$ 2)	1	30 ($\leq$ 35)	1 ($\leq$ 2)
mjpeg.ss	1	1	203 ($\leq$ 240)	4 ($\leq$ 6)	2	321 ($\leq$ 441)	3 ($\leq$5)
mjpeg.ms	1	2	694 ($\leq$ 781)	1 ($\leq$ 3)	1	133 ($\leq$ 190)	1 ($\leq$ 1)
mjpeg.sf	2	1	2591 ($\leq$ 3014)	5 ($\leq$ 6)	2	3226 ($\leq$ 4315)	6 ($\leq$ 6)
mjpeg.mf	2	2	1881 ($\leq$ 2143)	2 ($\leq$ 4)	1	307 ($\leq$ 340)	1 ($\leq$ 2)
mjpeg.zii	3	1	6164 ($\leq$ 6762)	4 ($\leq$ 6)	1	5971 ($\leq$ 6663)	4 ($\leq$ 6)
wfs.ctrl	1	1	202 ($\leq$ 235)	3 ($\leq$ 5)	3	405 ($\leq$ 795)	5 ($\leq$ 7)
wfs.src	1	2	292 ($\leq$ 387)	4 ($\leq$ 5)	2	228 ($\leq$ 357)	3 ($\leq$ 5)
wfs.ls	1	3	4931 ($\leq$ 5402)	8 ($\leq$ 12)	1	4996 ($\leq$ 5512)	9 ($\leq$ 14)
wfs.comp1	2	1	1606 ($\leq$ 1919)	12 ($\leq$ 15)	2	6157 ($\leq$ 7720)	26 ($\leq$ 30)
wfs.comp2	2	2	5960 ($\leq$ 6838)	25 ($\leq$ 26)	1	1940 ($\leq$ 2156)	15 ($\leq$ 20)

Contents

Challenges

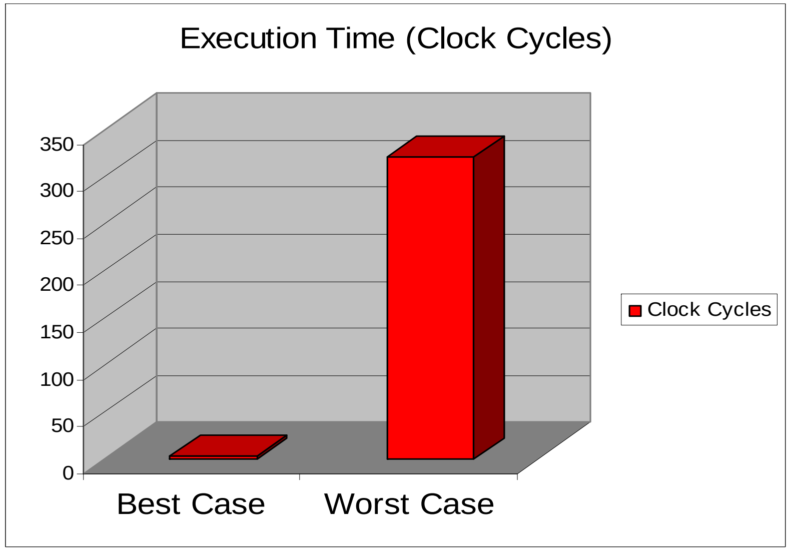
- Abstractions -

WCET

x = a + b;

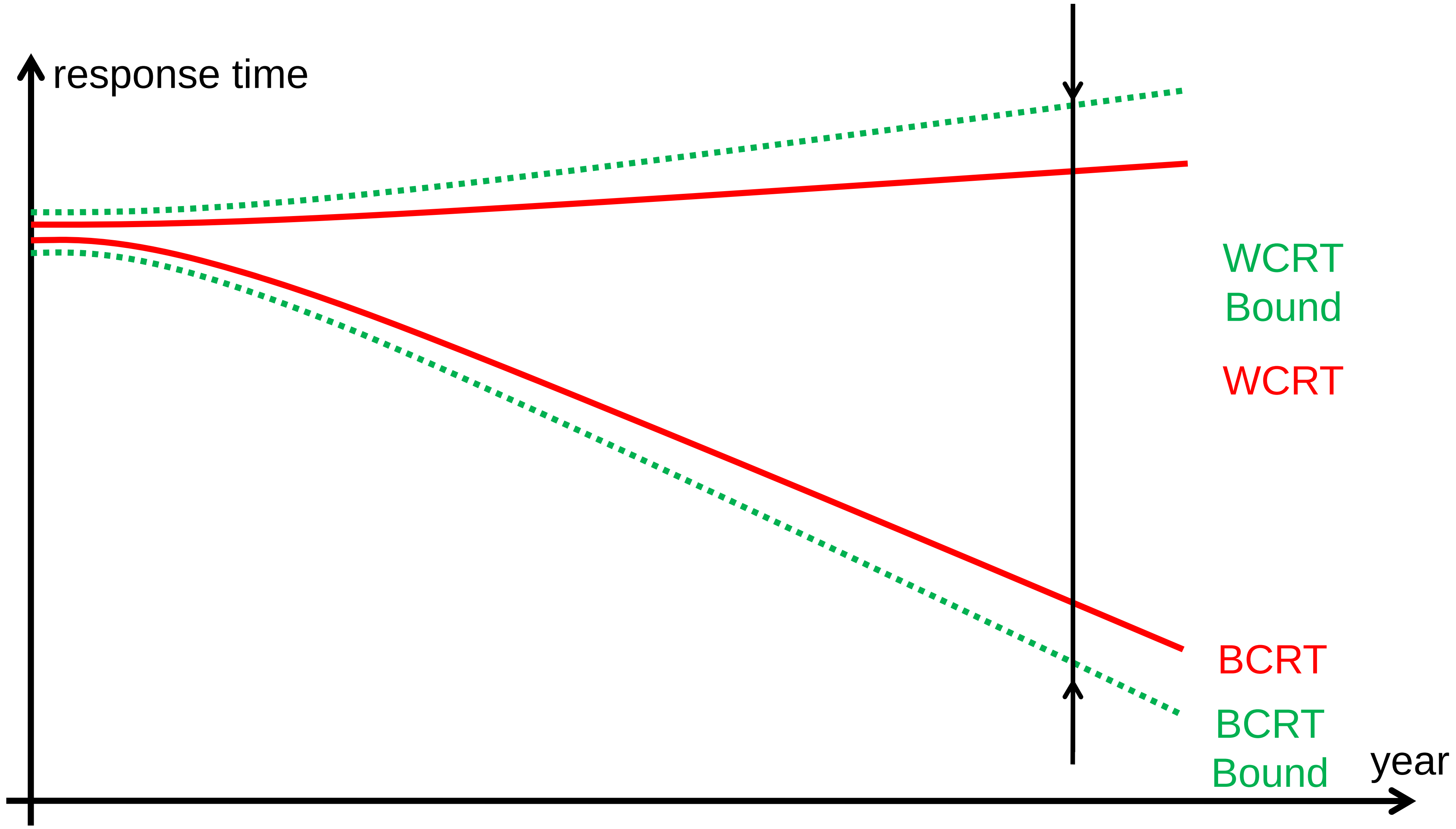
```
LOAD      r2,  a
LOAD      r1, _b
ADD       r3,r2,r1
```

PPC 755

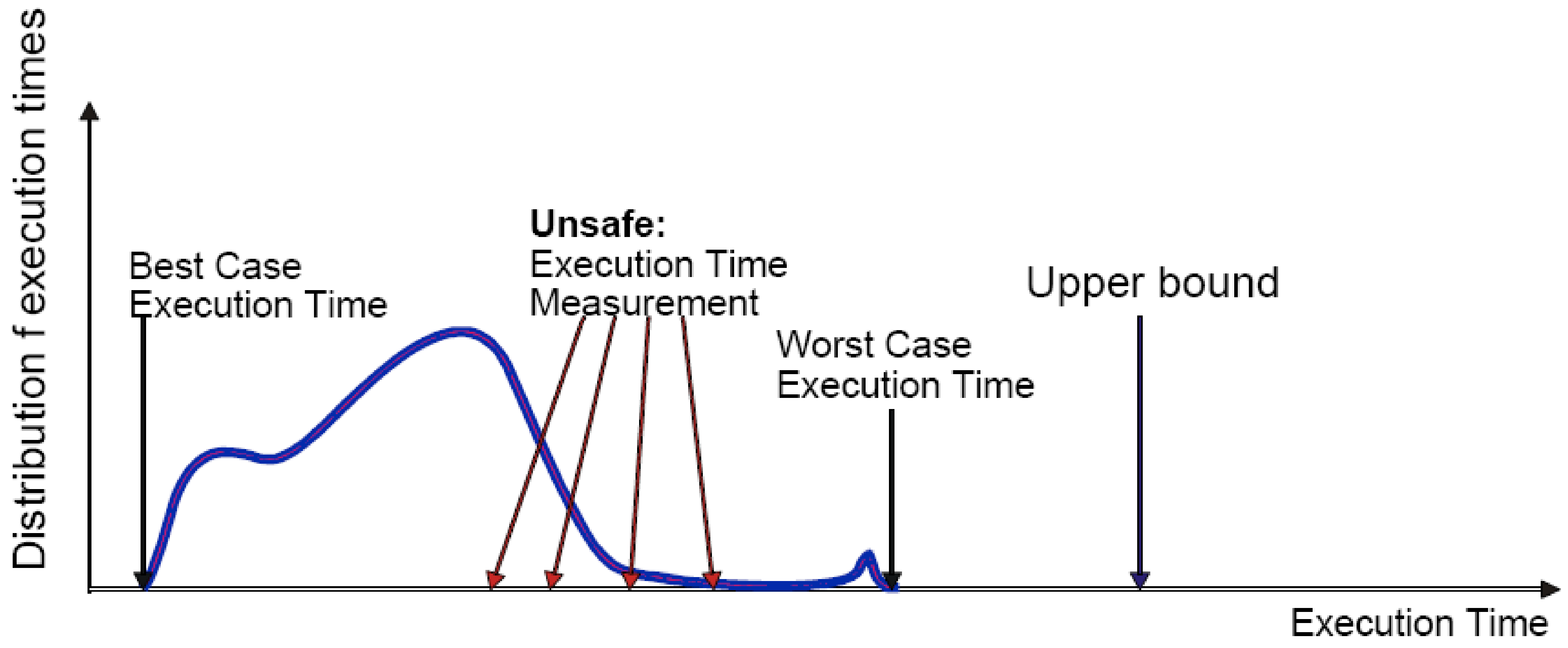


© Reinhard Wilhelm

(Timing) Predictability

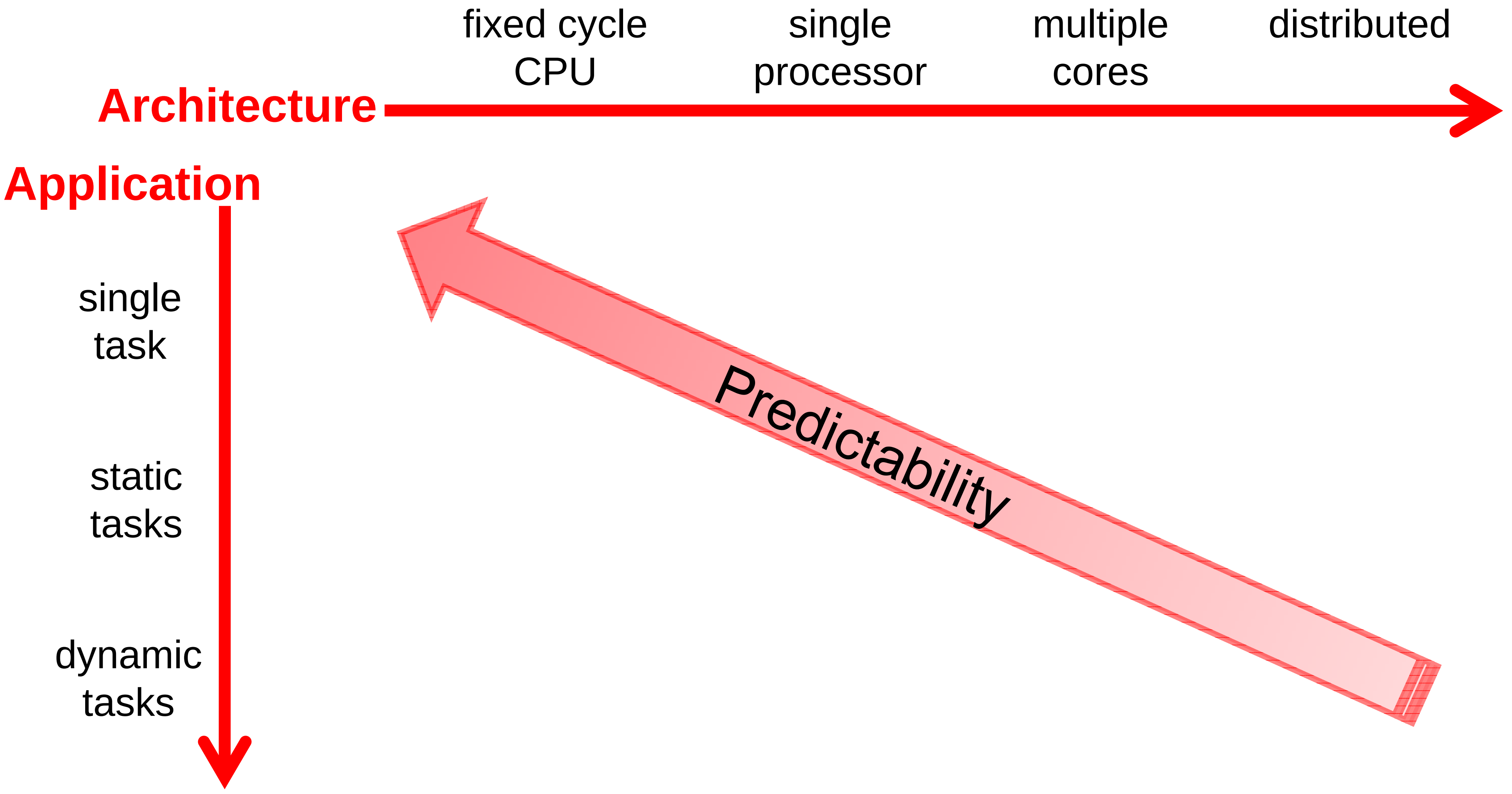


WCET

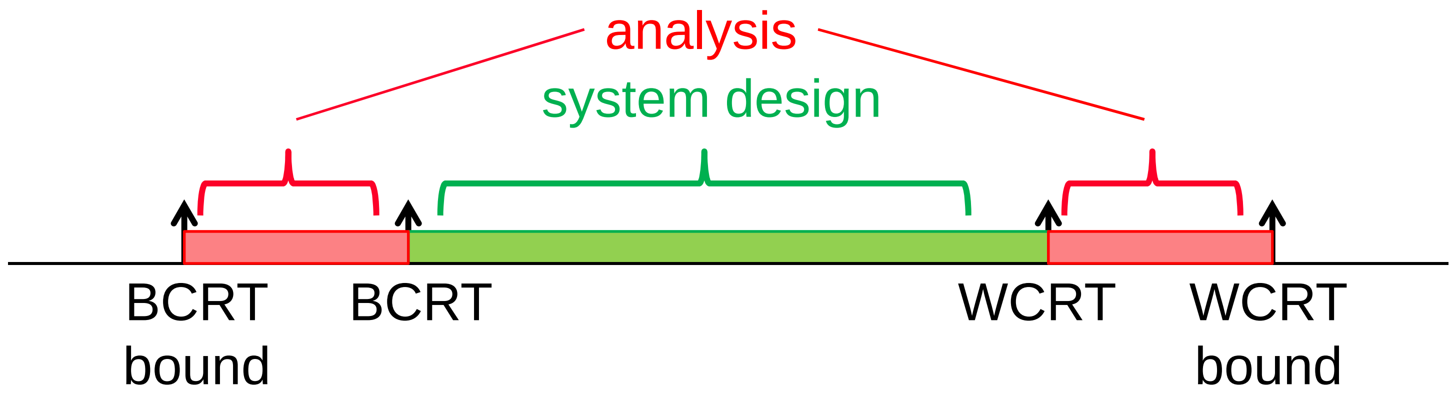


© Reinhard Wilhelm

Application and Architecture

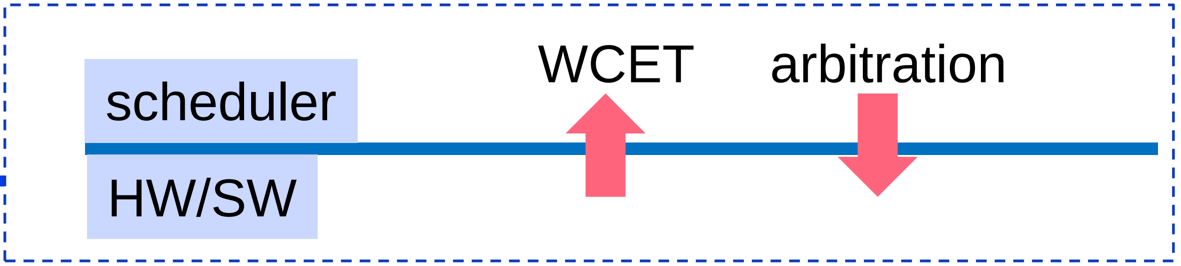


Classification of Predictability Loss



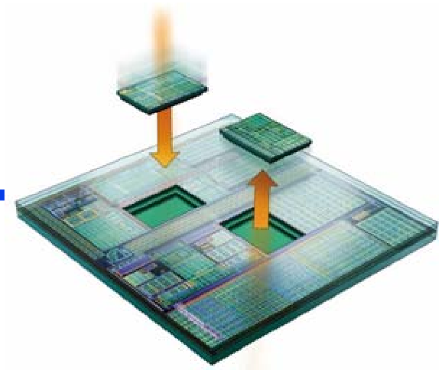
- *Analysis Loss:*
  - Construct system that can be easily analyzed
  - Use appropriate abstractions (models and methods)
- *System Design Loss:*
  - Decrease interference, long-range dependencies
  - Increase robustness of components
  - Use appropriate interfaces

Interfaces



- A task is (classically) characterized by its WCET.
  - May be useful in case of simple processors, but we have long-range state-dependent uni-processor behavior (pipelines, caches, speculation).
  - In case of multi-processors, we have additional interferences on the communication system which heavily influences WCET. We also may have intra-task parallelism.
  - WCET can no longer be considered as a useful interface between these abstraction layers.
- What about the other interfaces ?
  - Is the classical ISA (using instructions that abstract away time) still appropriate?

Acknowledgement



- ***Co-workers:***
  Jian-Jia Chen, Iuliana Bacivarov, Kai Lampka, Wolfgang Haid, Simon Perathoner, Nikolay Stoimenov, Kai Huang, Andreas Schranzhofer, Marco Caccamo, Rodolfo Pellizzoni
- ***Funding:***
  EU-SHAPES, EU-PREDATOR, EU-COMBEST, EU-ARTISTDESIGN, EU-EURETILE, EU-PRO3D, IBM, Siemens, NCCR-MICS, KTI

- ***Further Information:***
  - http://www.tik.ee.ethz.ch/~thiele
  - http://www.tec.ethz.ch