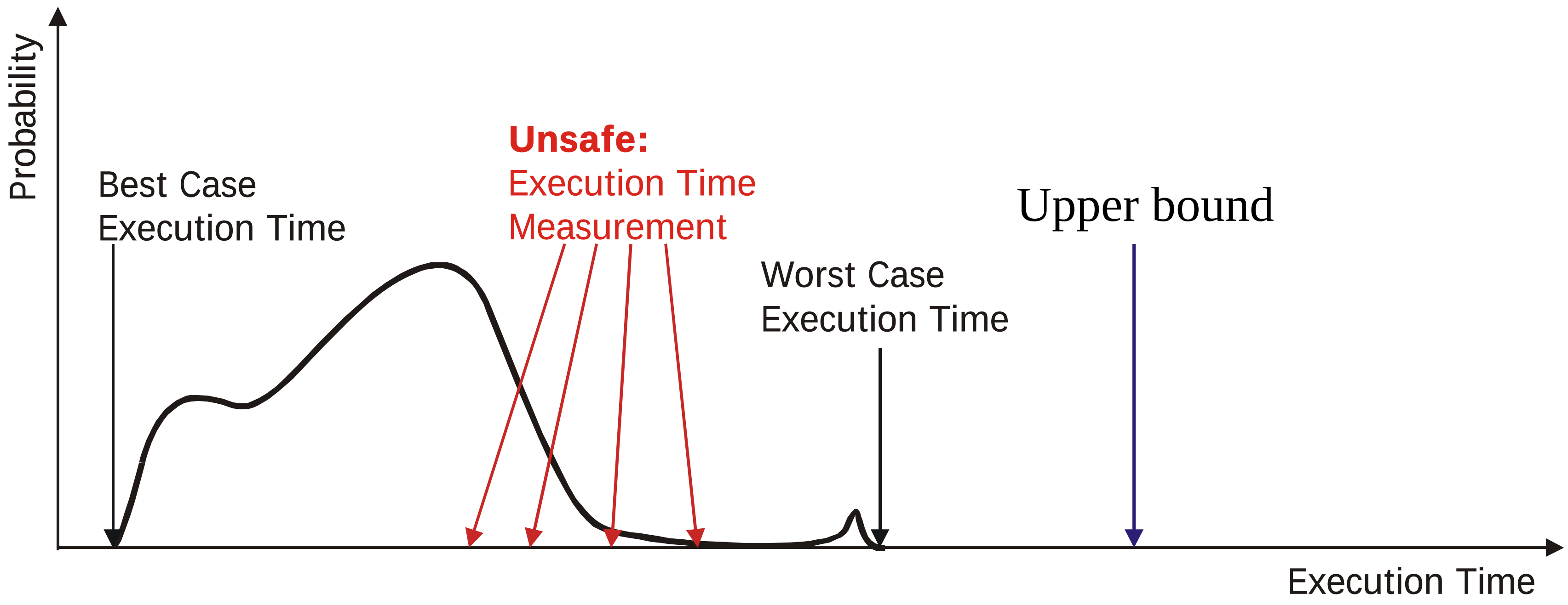
Embedded Systems 23

Measurement vs. Analysis

REVIEW



- typically huge variations in ET depending on input, cache effects,…
- cannot be covered within product development time
- rules of thumb add safety margins: pessimistic? optimistic?

Path Analysis

by Integer Linear Programming (ILP)

REVIEW

- Execution time of a program =

$$\sum_{\text{Basic\_Block } b} \text{Execution\_Time}(b) \times \text{Execution\_Count}(b)$$

- ILP solver maximizes this function to determine the WCET
- Program structure described by linear constraints
  - automatically created from CFG structure
  - needs info about loop/recursion bounds
  - additional linear constraints may be added to exclude infeasible paths (contradictory conditions,…)

REVIEW

Timing Analysis

1. **For each instruction, determine possible ET in context:**
   - Determine possible processor behavior at instruction
   - Exclude timing accidents when context renders them impossible
   - Determine instruction WCET and BCET based on this
2. **Accumulate across basic blocks**
   - Determines safe bounds for WCET and/or BCET for basic blocks (with contextual info. inherited)
3. **Worst-case Path Determination**
   - Maps cost-annotated (WCET/BCET) control flow graph to an integer linear program
   - Determines paths with extremal (max./min.) cost
   - Thus determines WCET / BCET of complete task

Abstract Interpretation

REVIEW

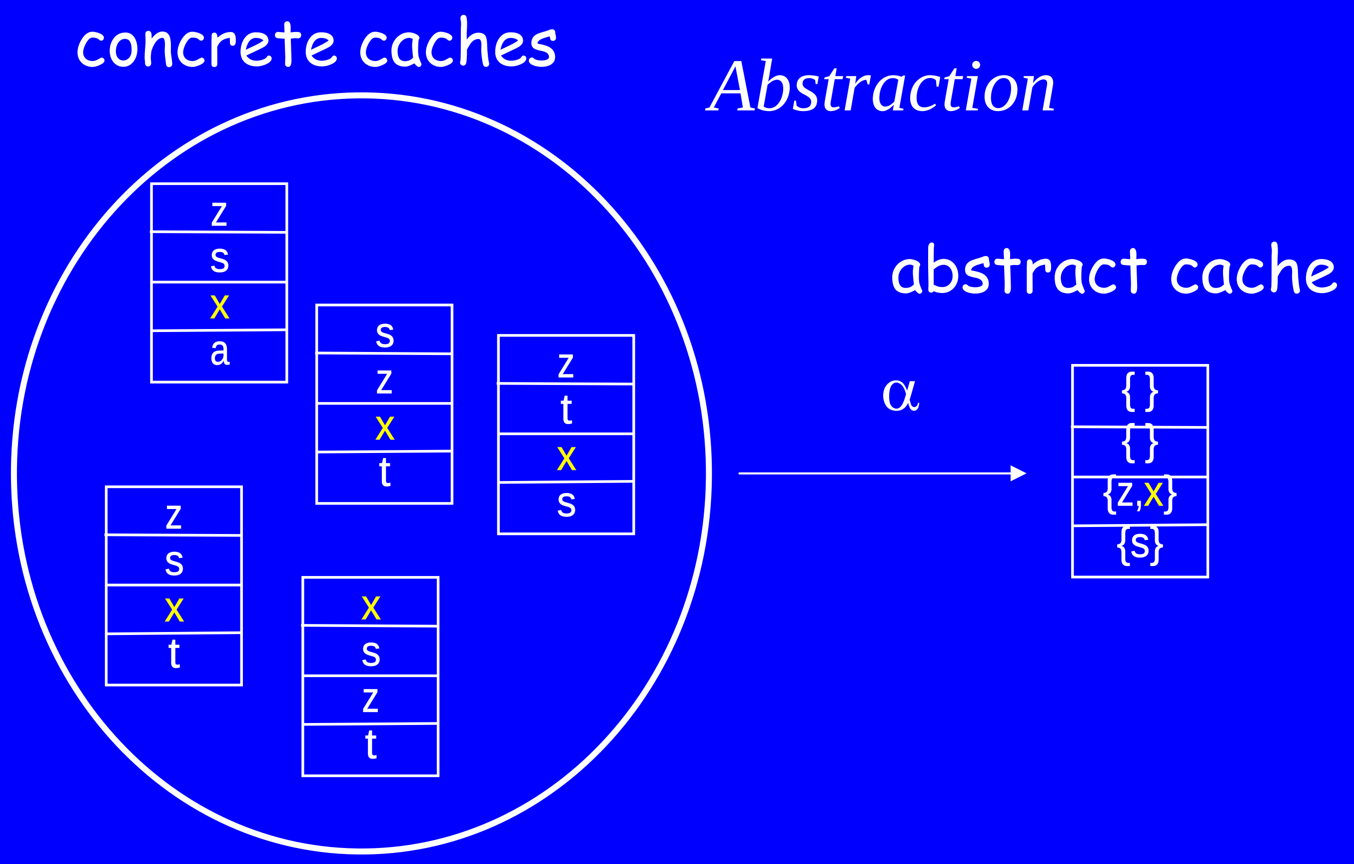
- Semantics-based method for static program analysis
- Basic idea: Perform the program's computations using abstract values in place of the concrete values

- **Abstract domain** = complete semilattice related to concrete domain by **abstraction** and **concretization** functions
- **Abstract transfer functions** for each statement type = abstract versions of their semantics
- **Join function**: combining abstract values from different control-flow paths (lub on lattice)

Abstract Domain: Must Cache

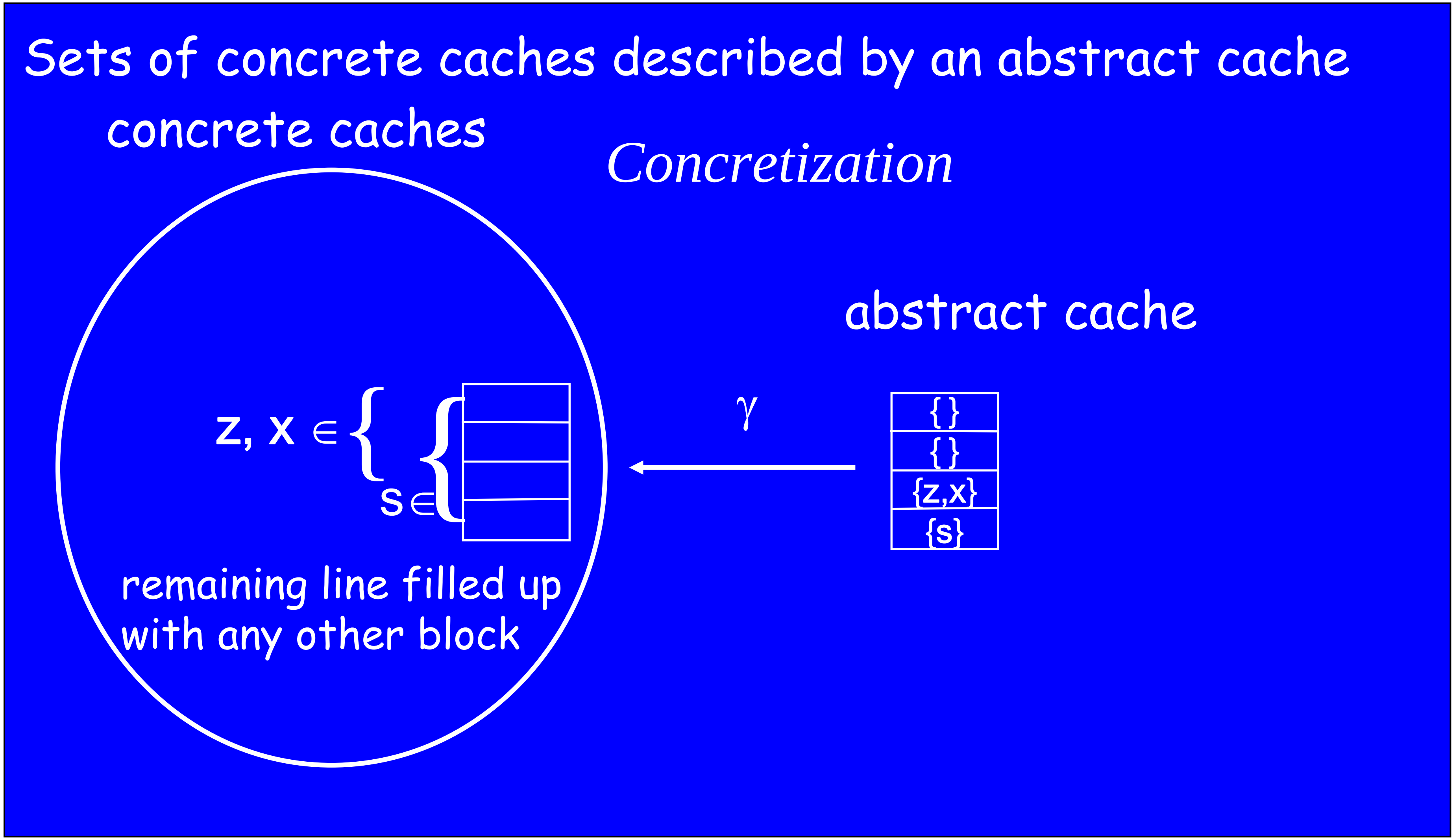
REVIEW

Representing sets of concrete caches by their description



Abstract Domain: Must Cache

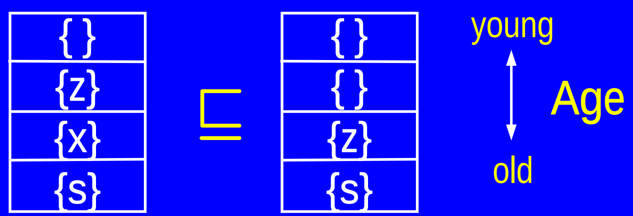
REVIEW



Lattice for Must Cache

REVIEW

- Set A of elements
- Information order $\sqsubseteq$
- Join operator $\sqcup$
- Top element $\top$
- Bottom element $\perp$



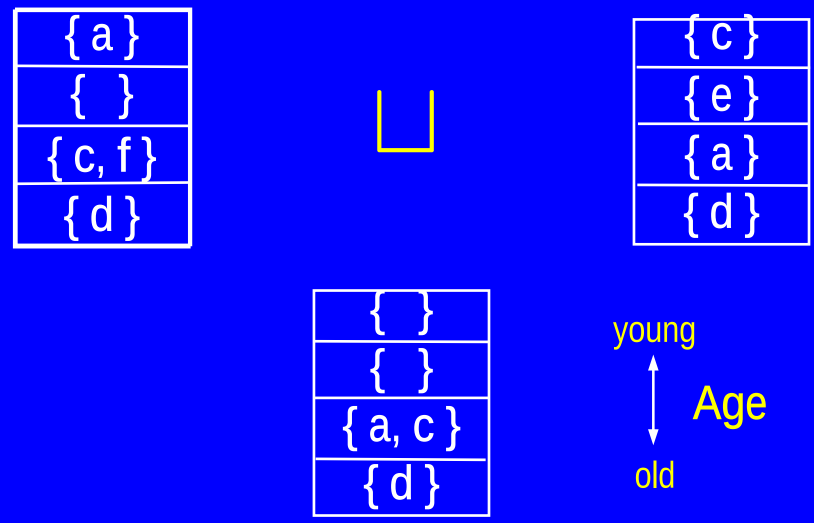
Better precision:

more elements in the cache or with younger age.

NB. The more precise abstract cache represents less concrete cache states!

Lattice: Must Cache

- Set A of elements
- Information order ⊑
- Join operator ⊔
- Top element ⊤
- Bottom element ⊥

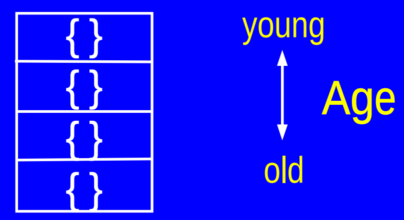


Form the intersection and associate the elements with the maximum of their ages

Lattice: Must Cache

REVIEW

- Set A of elements
- Information order $\sqsubseteq$
- Join operator $\sqcup$
- Top element $\top$
- Bottom element $\bot$



No information:

All caches possible

Lattice: Must Cache

REVIEW

- Set A of elements
- Information order $\sqsubseteq$
- Join operator $\sqcup$
- Top element $\top$
- Bottom element $\bot$

Dedicated unique bottom element representing the empty set of caches

Galois connection – Relating Semantic Domains

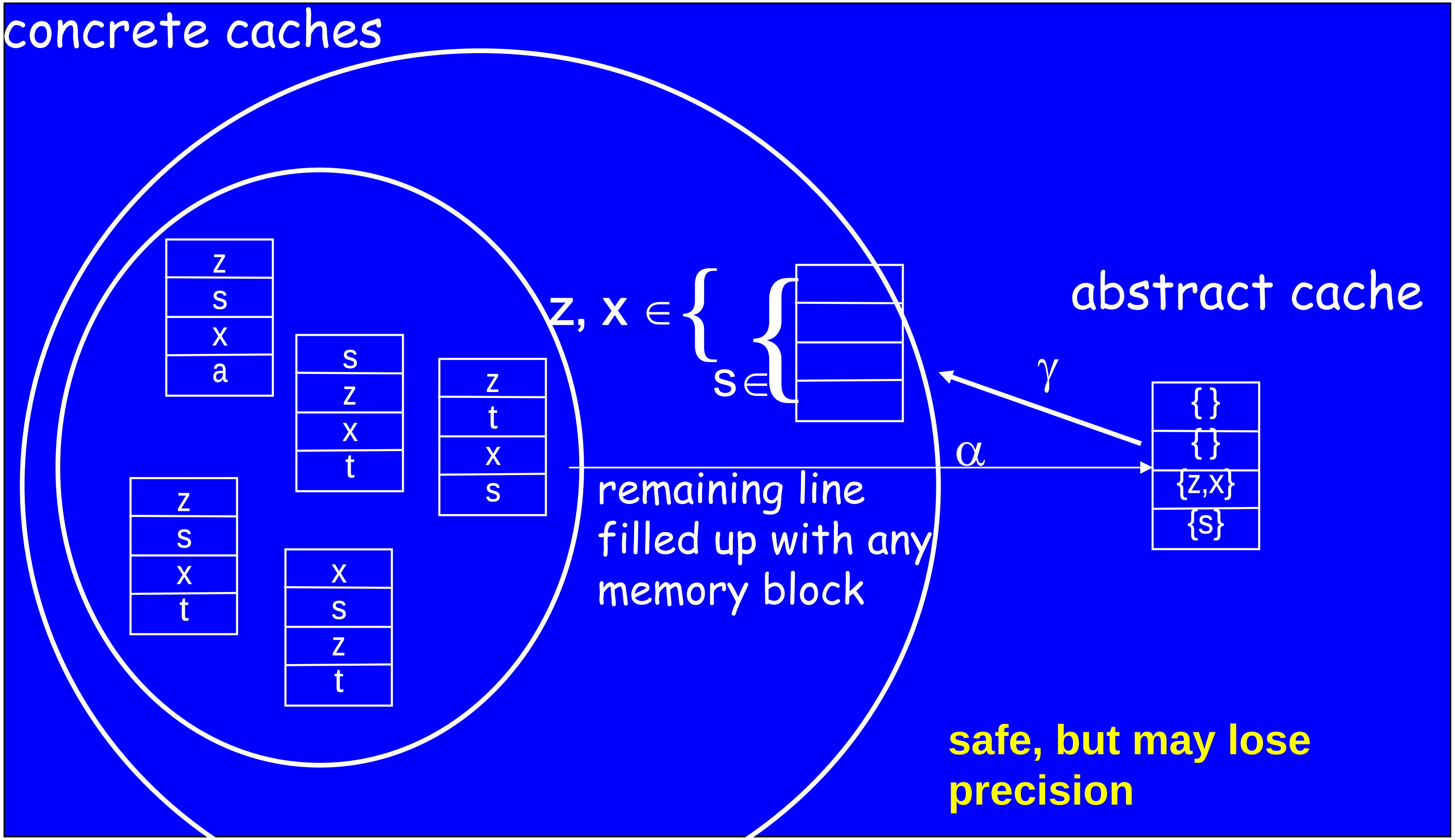
- Lattices C, A
- two monotone functions α and γ
- Abstraction: α: C $\rightarrow$ A
- Concretization γ: A $\rightarrow$ C
- (α,γ) is a Galois connection
  if and only if

$$\gamma \bullet \alpha \sqsupseteq_{\mathcal{C}} \mathrm{id}_{C} \quad \text{and} \quad \alpha \bullet \gamma \sqsubseteq_{\mathcal{A}} \mathrm{id}_{A}$$

Switching safely between concrete and abstract domains, possibly losing precision

Abstract Domain Must Cache

$\gamma \bullet \alpha \sqsupseteq_{\mathcal{C}}$ **id**$_{\mathbf{C}}$



Result of the Cache Analysis

REVIEW

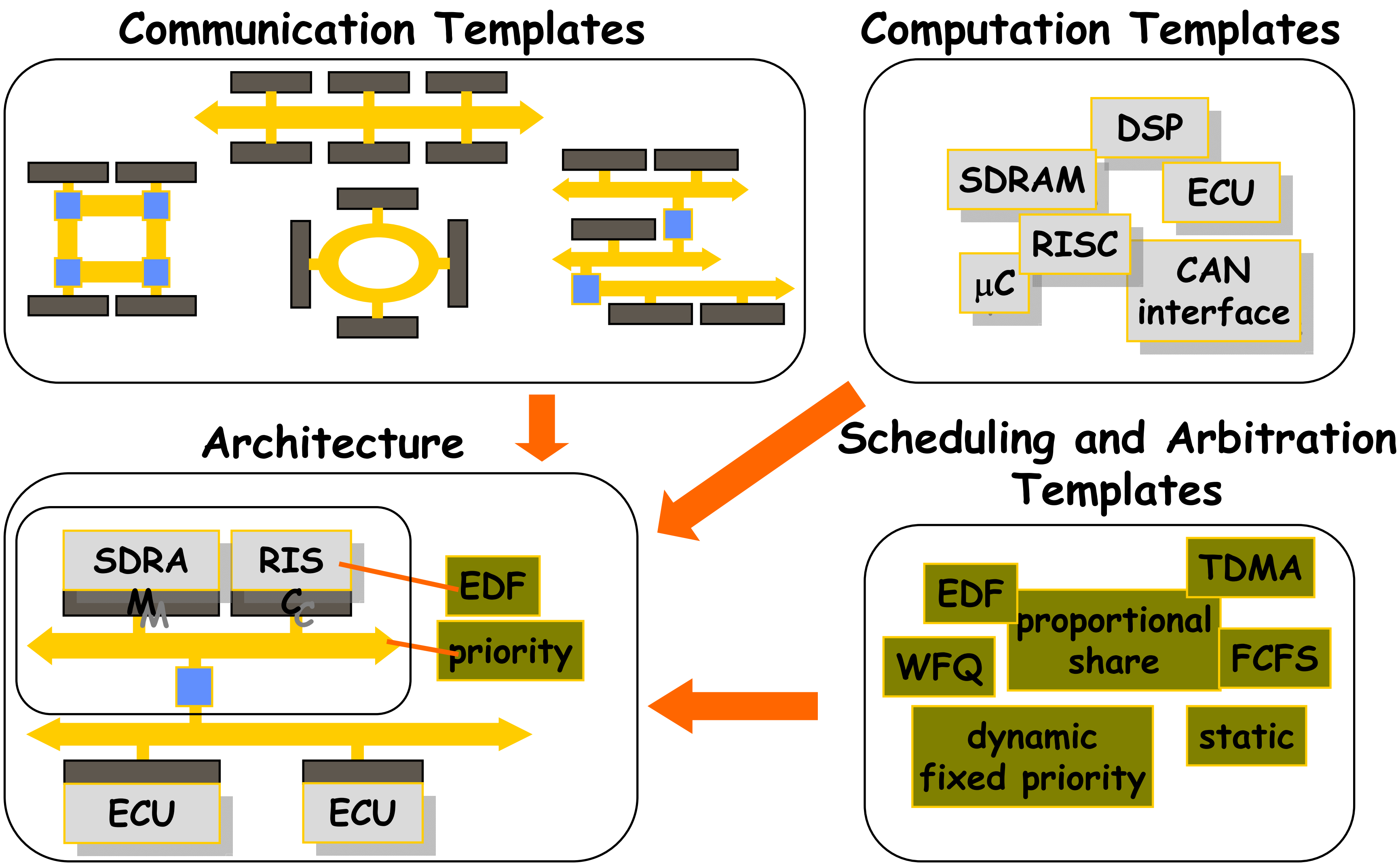
Categorization of memory references

Category	Abb.	Meaning
always hit	**ah**	The memory reference will always result in a cache hit.
always miss	**am**	The memory reference will always result in a cache miss.
not classified	**nc**	The memory reference could neither be classified as **ah** nor **am**.

WCET: ah improves bound, nc and am count as pot. miss
BCET: am tightens bound, nc and ah count as potent. hit

Realtime Calculus

System Composition



A Four-Step Approach

1. *Abstraction*: Build abstract models for “first class citizens”

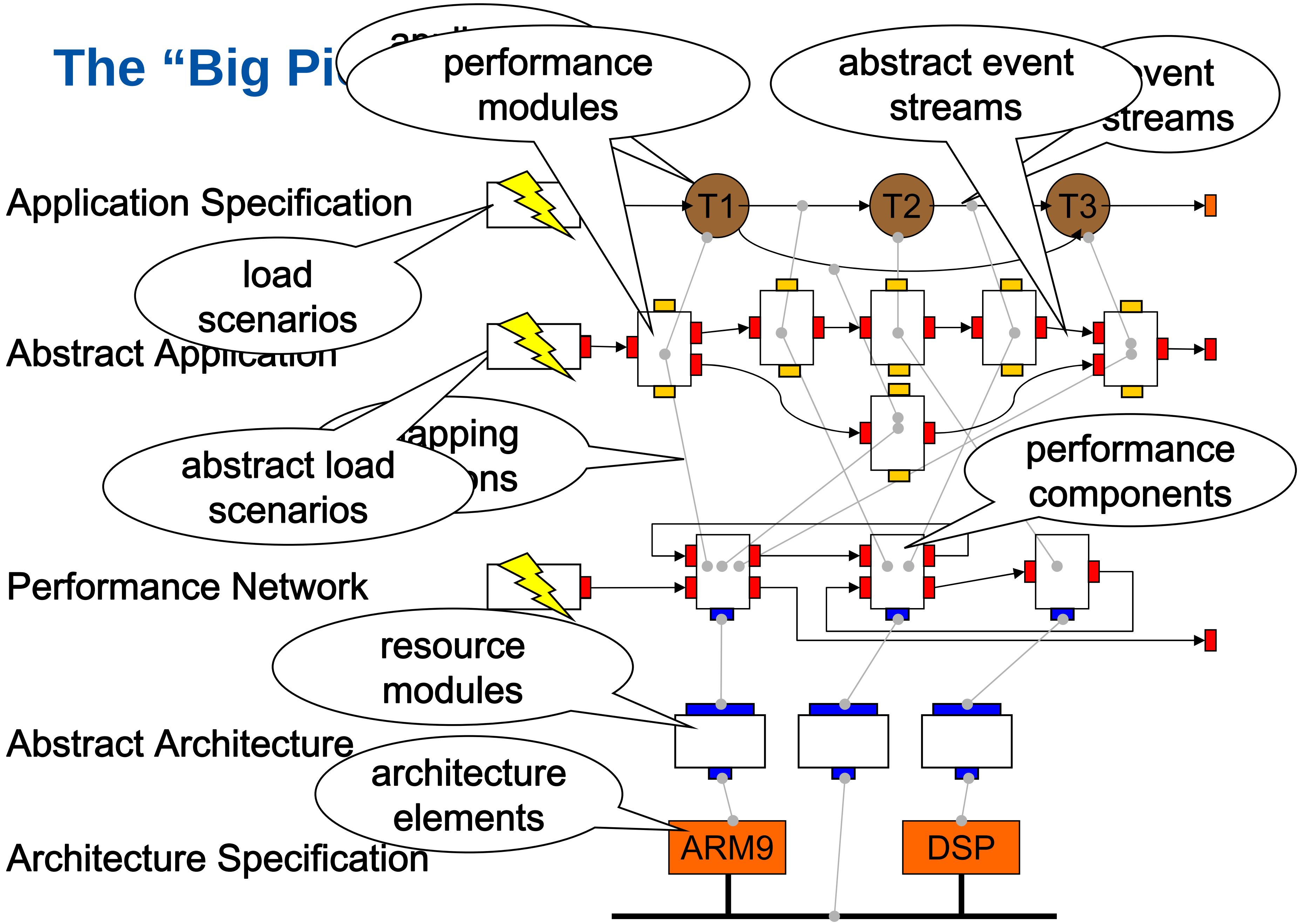
event streams	->	abstract event streams
architecture elements	->	resource modules
application processes	->	performance modules

2. *Performance Components*: Combine performance modules using resource sharing information

3. *Performance Network*: Combine all models to a network that represents the performance aspects

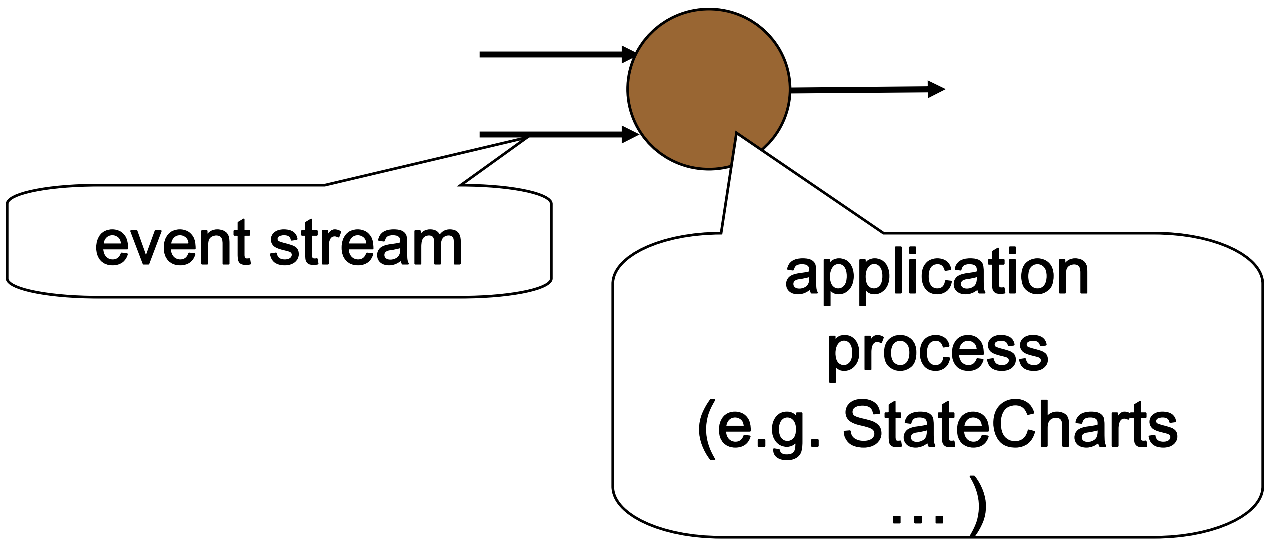
4. *Analysis*

The “Big Pi


Application Specification
Abstract Application
Performance Network
Abstract Architecture
Architecture Specification
performance modules
abstract event streams
load scenarios
abstract load scenarios
performance components
resource modules
architecture elements
T1
T2
T3
ARM9
DSP

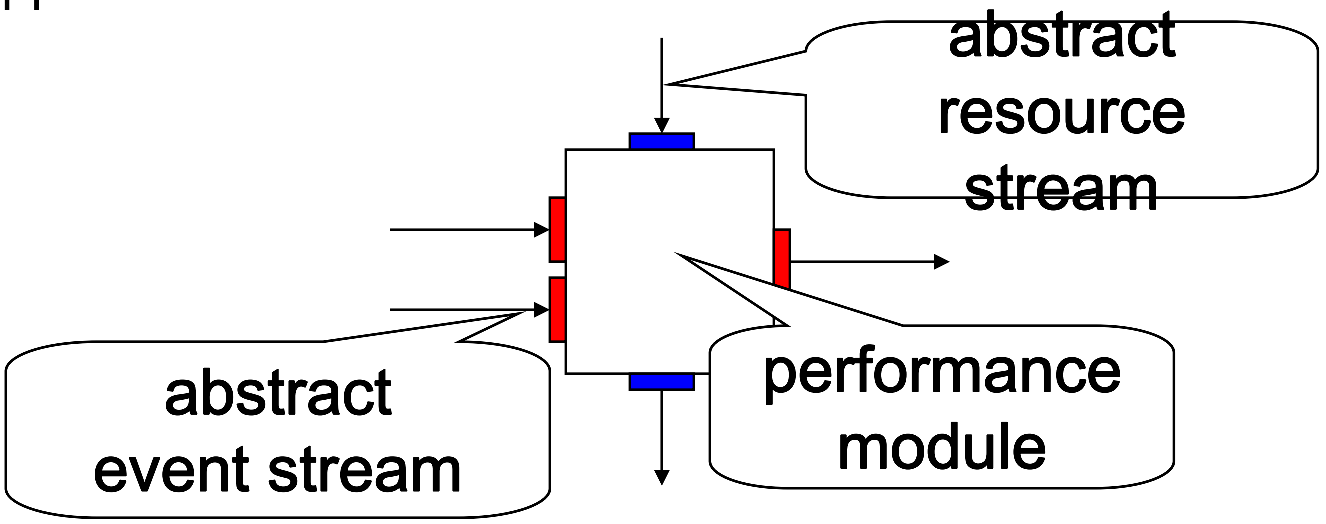
Step 1: Abstract Application Model

From a functional model…

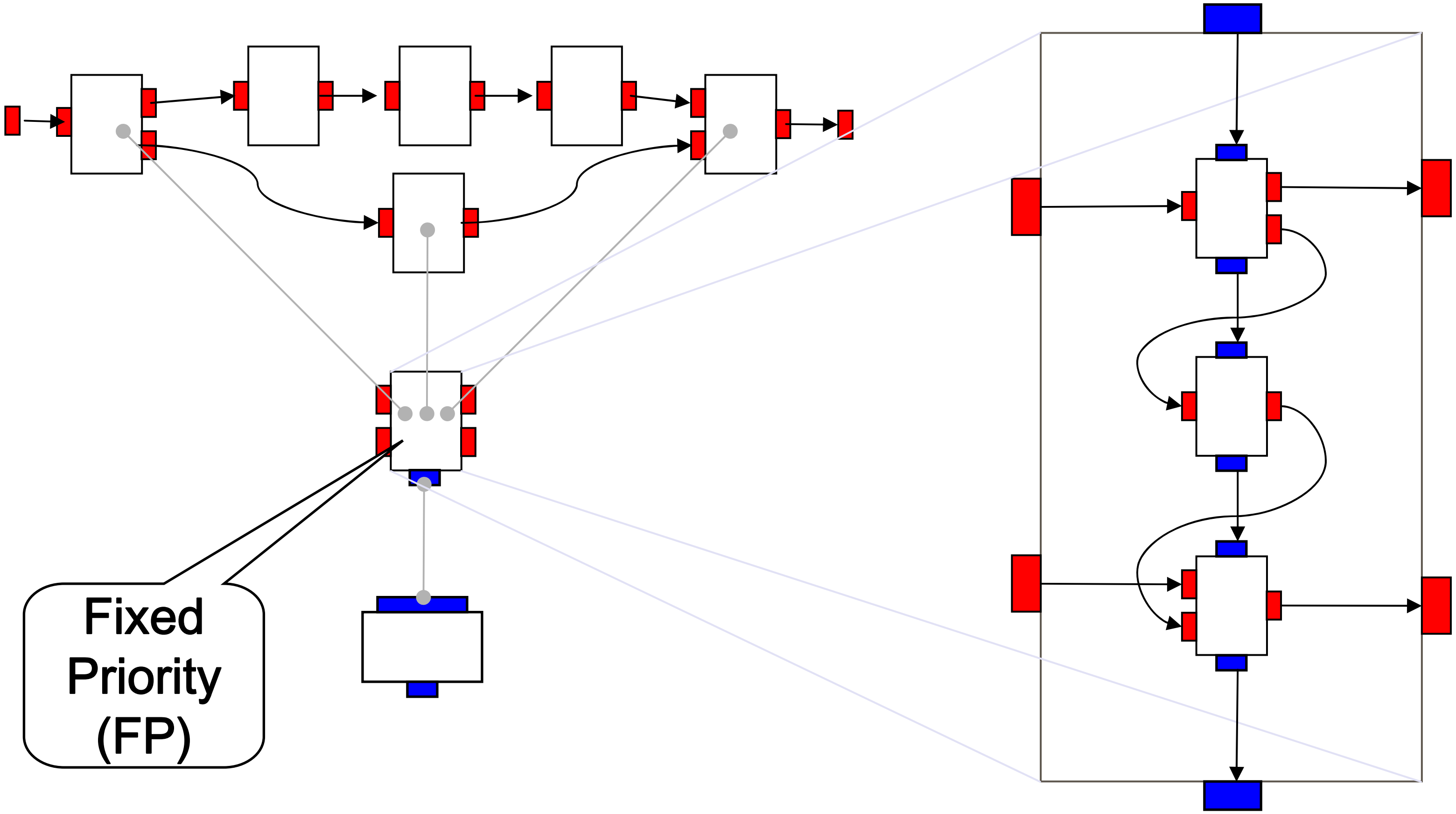


Step 1: Abstract Functional Units

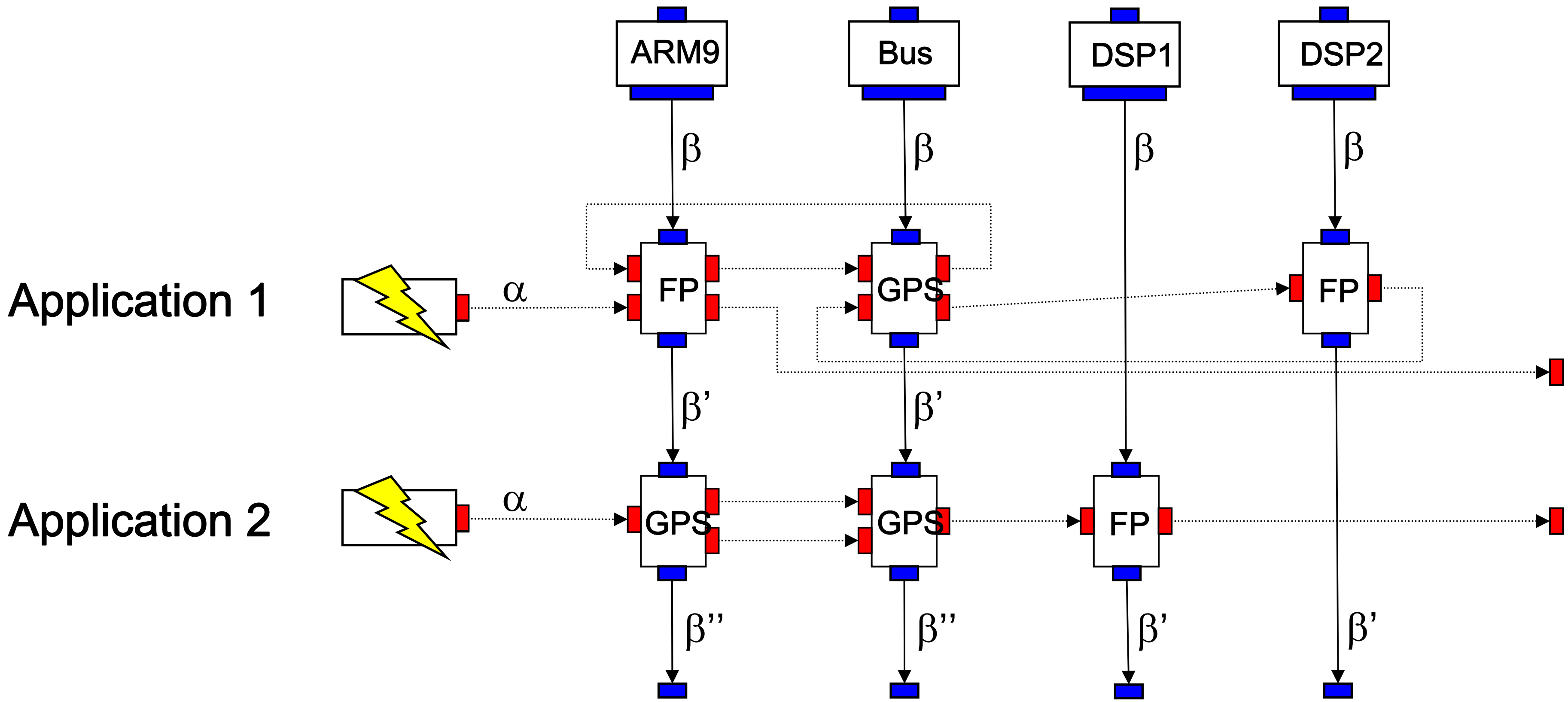
… to an abstract application model



Step 2: Build Performance Components

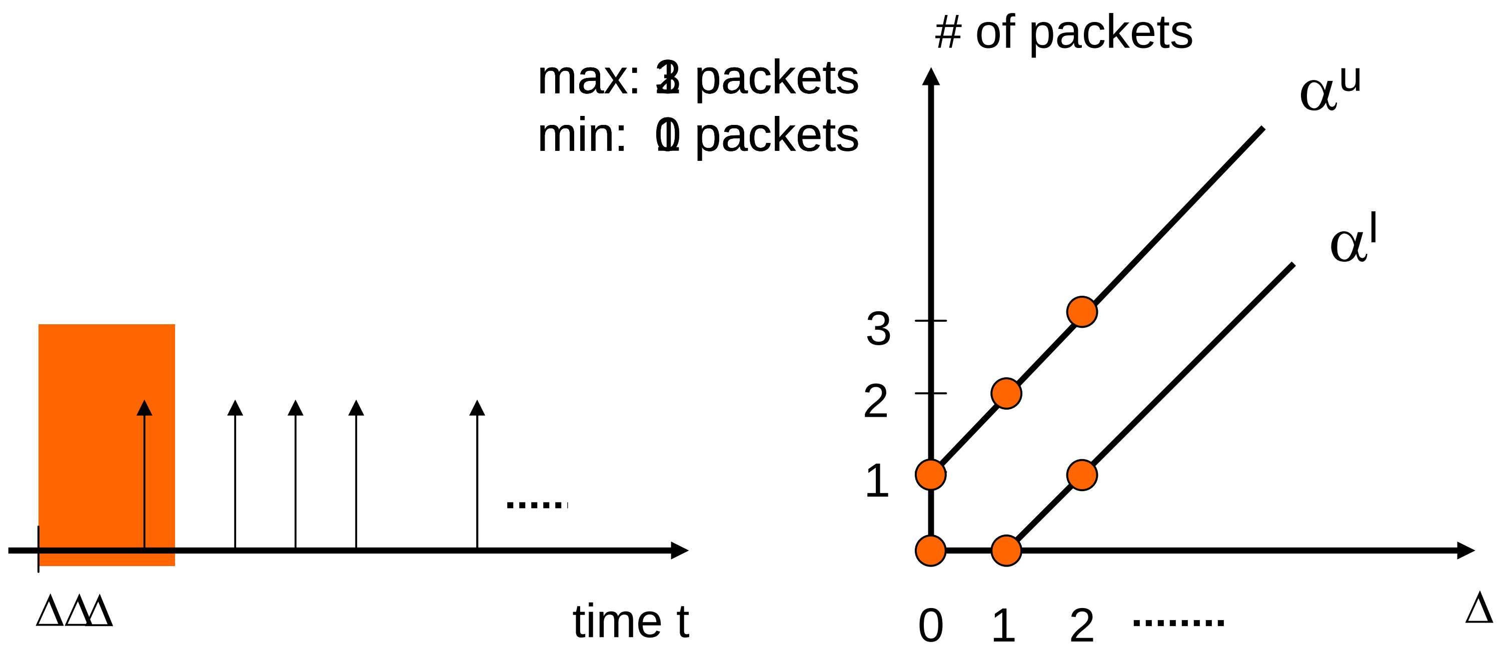


Step 3 and 4: Compose and Analyze



Event & Resource Models

- Use arrival curves to capture packet streams:

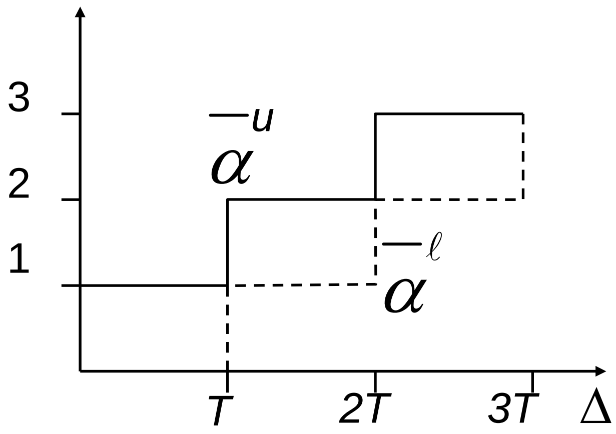


Arrival curves

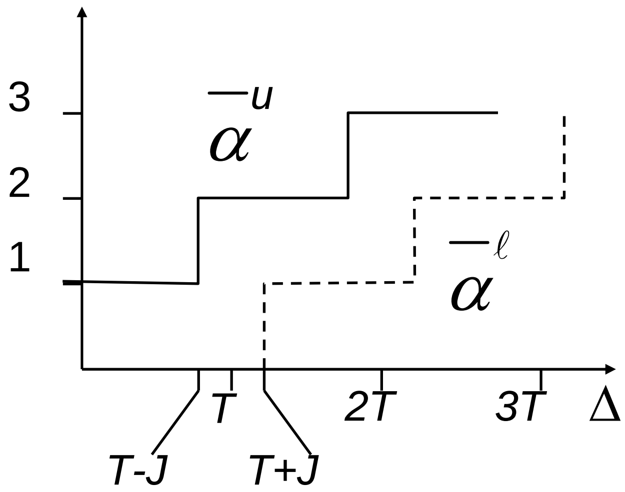
Arrival curves describe the maximum and minimum number of events arriving in some time interval Δ

Examples:

periodic event stream



periodic event stream with jitter



Arrival curves

Definition: Let $R(t)$ denote the number of events that arrive on an event stream in the time interval $[0,t)$. Then the following holds:

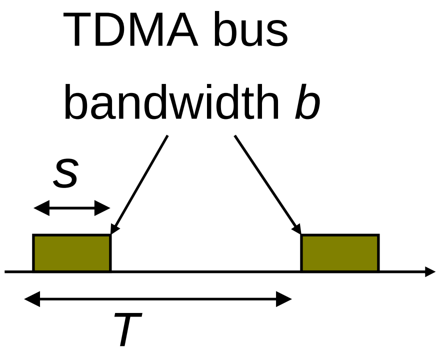
$$\overline{\alpha}^{l}(t-s) \leq R(t) - R(s) \leq \overline{\alpha}^{u}(t-s), \forall s < t$$

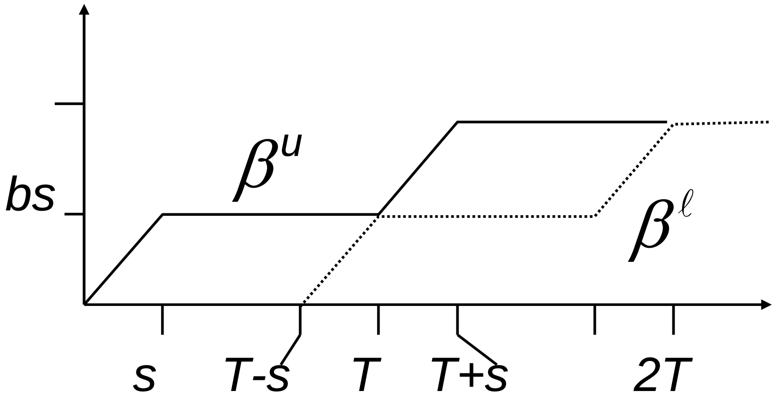
$$\overline{\alpha}^{l}(0) = \overline{\alpha}^{u}(0).$$

Service curves

Service curves β^u resp. β^ℓ describe the maximum and minimum service capacity available in some time interval Δ

Example:





Service curves

Definition: Let C(*t*) denote the number of communication or processing cycles available from a resouce of the time interval [0,*t*).
Then the following holds:

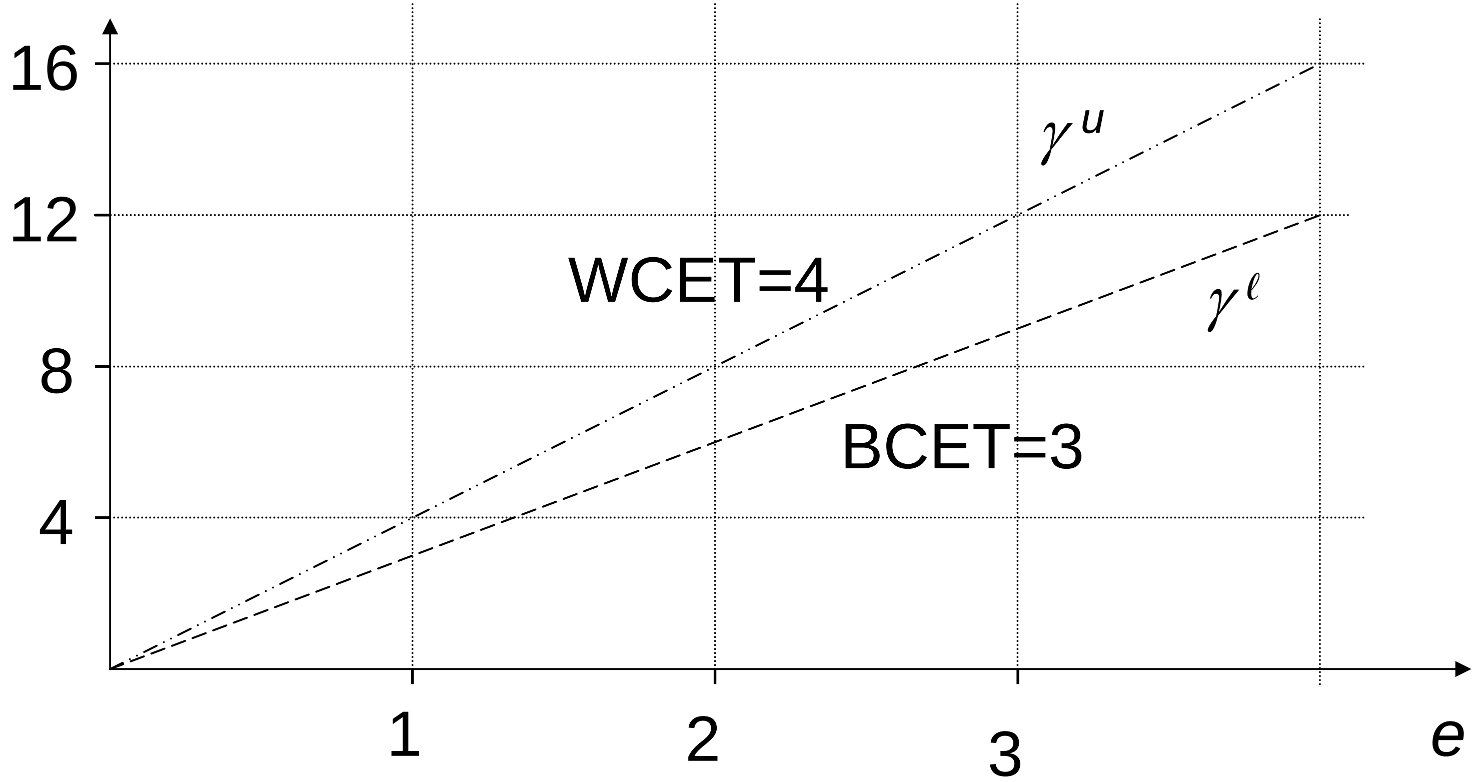
$$\overline{\beta}^{l}(t-s) \leq C(t) - C(s) \leq \overline{\beta}^{u}(t-s), \forall s < t$$

$$\overline{\beta}^{l}(0) = \overline{\beta}^{u}(0).$$

Workload characterization

γ^u resp. γ^ℓ describe the maximum and minimum service capacity required as a function of the number *e* of events

Example



Workload required for incoming stream

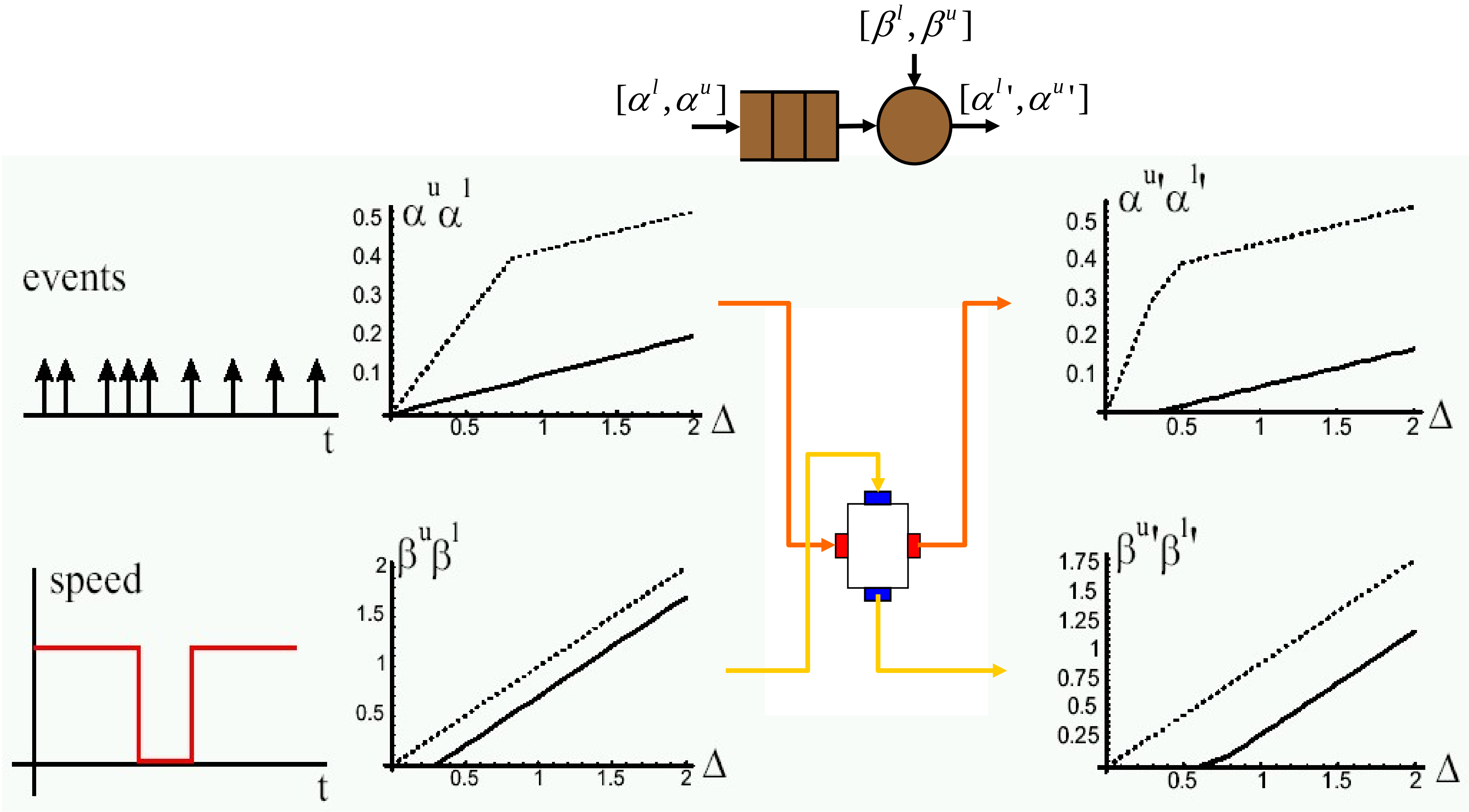
- Incoming workload

$$\alpha^{u}(\Delta) = \gamma^{u}\left(\overline{\alpha^{u}}(\Delta)\right) \qquad \alpha^{\ell}(\Delta) = \gamma^{\ell}\left(\overline{\alpha^{\ell}}(\Delta)\right)$$

- Upper and lower bounds on the number of events

$$\overline{\beta}^{u}(\Delta) = \gamma^{-1}\left(\beta^{u}(\Delta)\right) \qquad \overline{\beta}^{\ell}(\Delta) = \gamma^{-1}\left(\beta^{\ell}(\Delta)\right)$$

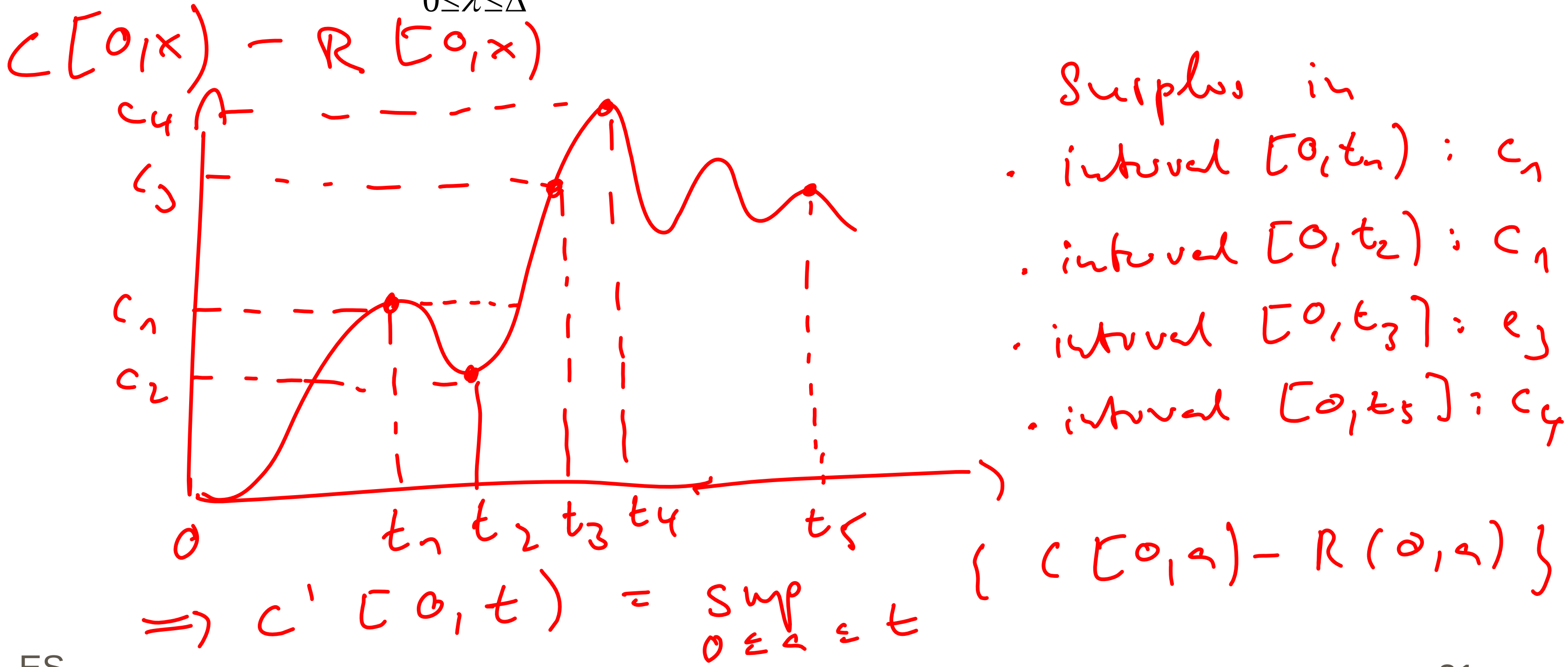
Transformation of Curves by Modules



Performance modules

Theorem: Given an event stream described by the arrival curves α^u, α^l, and a resource described by the service curves β^u, β^l, then the resulting service is bounded by

$$\beta^{l}(\Delta) = \sup_{0 \le \lambda \le \Delta} \{\beta^{l}(\lambda) - \alpha^{u}(\lambda)\}, \forall 0 \le \Delta$$



$$C'[s,t) = C'[0,t) - C'[0,s)$$

$$C'[s,t) = \sup_{0 \le a \le t} \{ C[0,a) - R[0,a) \}$$

$$- \sup_{0 \le b \le s} \{ C[0,b) - R[0,b) \}$$

$$s \le t \Rightarrow b \le a$$

$$= \inf_{0 \le b \le s} \{ \sup_{0 \le a \le t} (C[0,a) - C[0,b) - (R[0,a) - R[0,b)) \} \}$$

$$= \inf_{0 \le b \le s} \{ \sup_{0 \le a-b \le t-b} \{ C[b,a) - R[b,a) \} \}$$

$$\ge \inf_{0 \le b \le s} \{ \sup_{0 \le \lambda \le t-b} \{ \beta^L(\lambda) - \alpha^u(\lambda) \} \}$$

$$\ge \sup_{0 \le \lambda \le t-s} \{ \beta^L(\lambda) - \alpha^u(\lambda) \} \}$$

Performance Modules

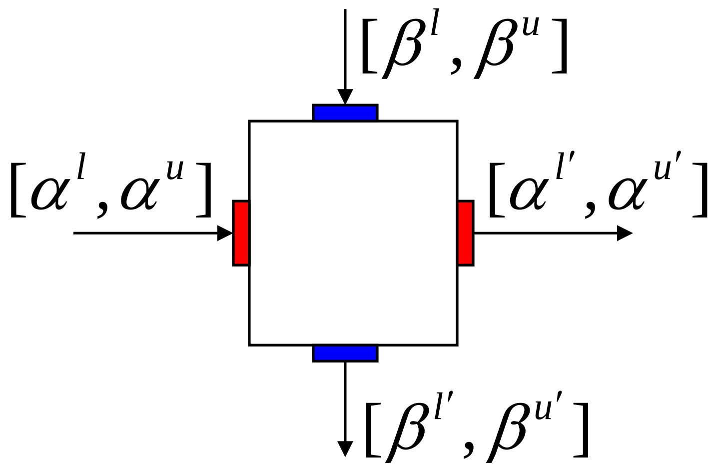
$$v(\Delta) \wedge w(\Delta) = \min\{v(\Delta), w(\Delta)\}$$

$$v \underline{\oplus} w(\Delta) = \inf_{0 \le \lambda \le \Delta} \{v(\lambda) + w(\Delta - \lambda)\}$$

$$v \underline{\otimes} w(\Delta) = \inf_{0 \le \lambda} \{v(\Delta + \lambda) - w(\lambda)\}$$

$$v \overline{\oplus} w(\Delta) = \sup_{0 \le \lambda \le \Delta} \{v(\lambda) + w(\Delta - \lambda)\}$$

$$v \overline{\otimes} w(\Delta) = \sup_{0 \le \lambda} \{v(\Delta + \lambda) - w(\lambda)\}$$



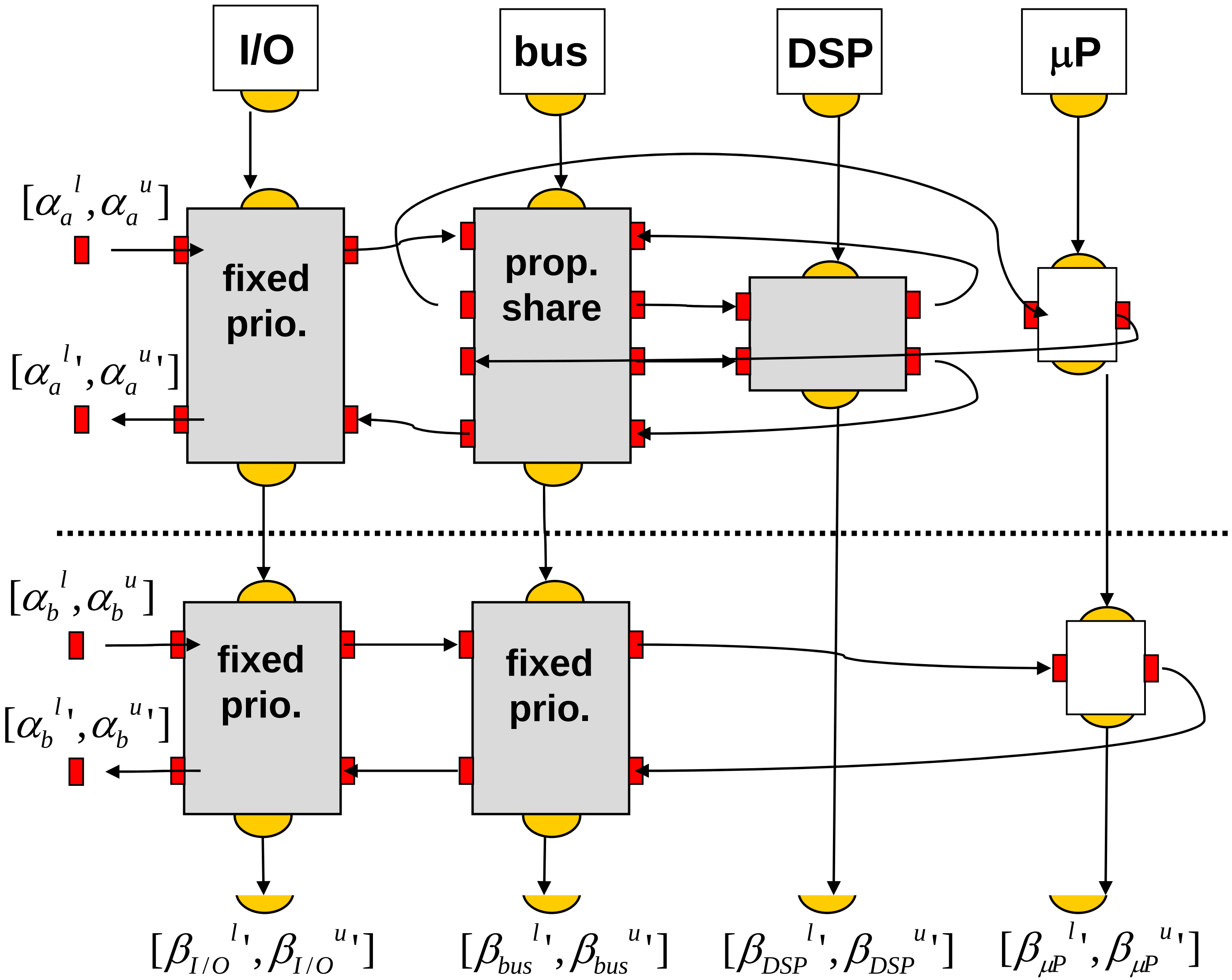
$$\alpha^{u'} = [(\alpha^u \underline{\oplus} \beta^u) \overline{\otimes} \beta^l] \wedge \beta^u$$

$$\alpha^{l'} = [(\alpha^l \overline{\otimes} \beta^u) \underline{\oplus} \beta^l] \wedge \beta^l$$

$$\beta^{u'} = (\beta^u - \alpha^l) \underline{\otimes} 0$$

$$\beta^{l'} = (\beta^l - \alpha^u) \overline{\oplus} 0$$

Compose and Analyze

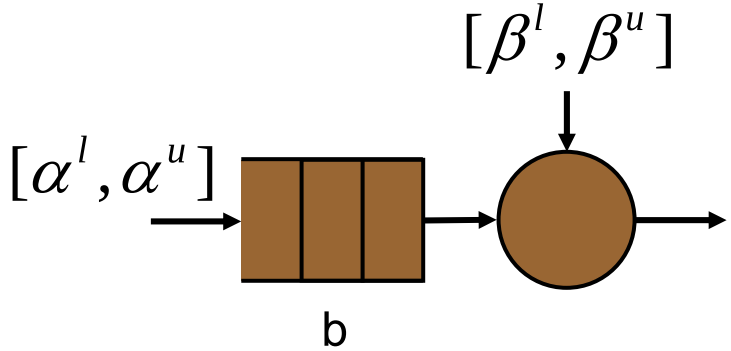


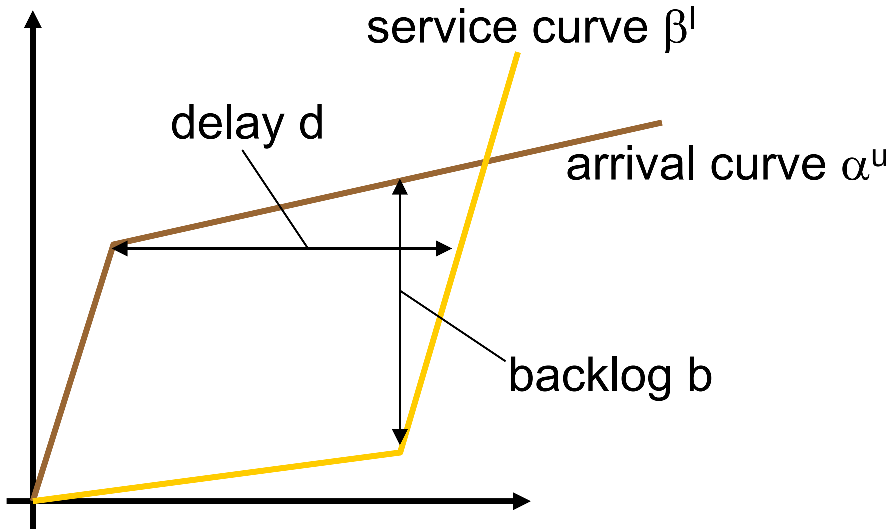
Compose and Analyze

Delay and Memory

$$d(t) = \inf\{\tau \geq 0 : R(t) \leq R'(t+\tau)\} \leq \sup_{u \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(u) \leq \beta^l(u+\tau)\} \right\}$$

$$b(t) = R(t) - R'(t) \leq \sup_{u \geq 0} \{\alpha^u(u) - \beta^l(u)\}$$

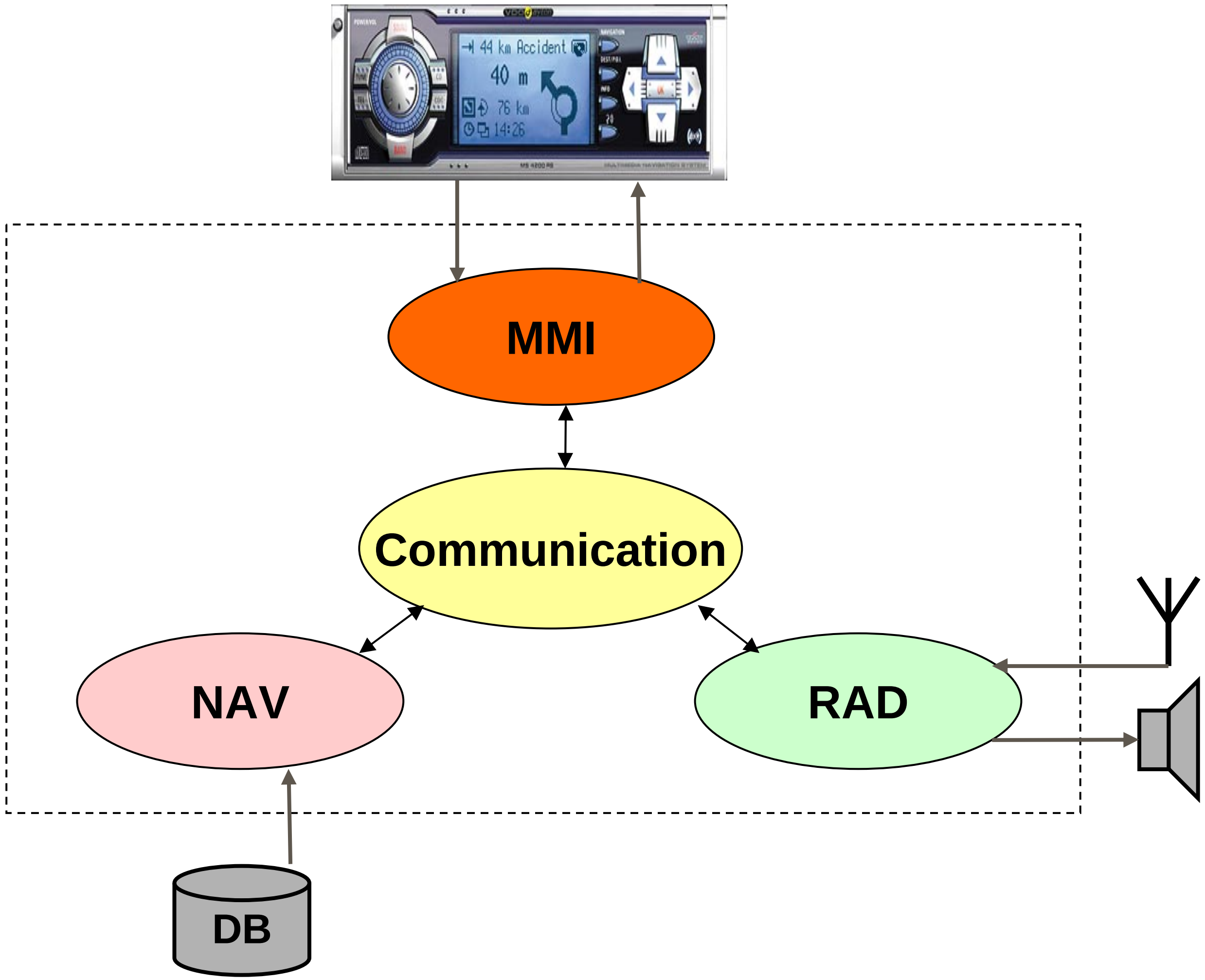




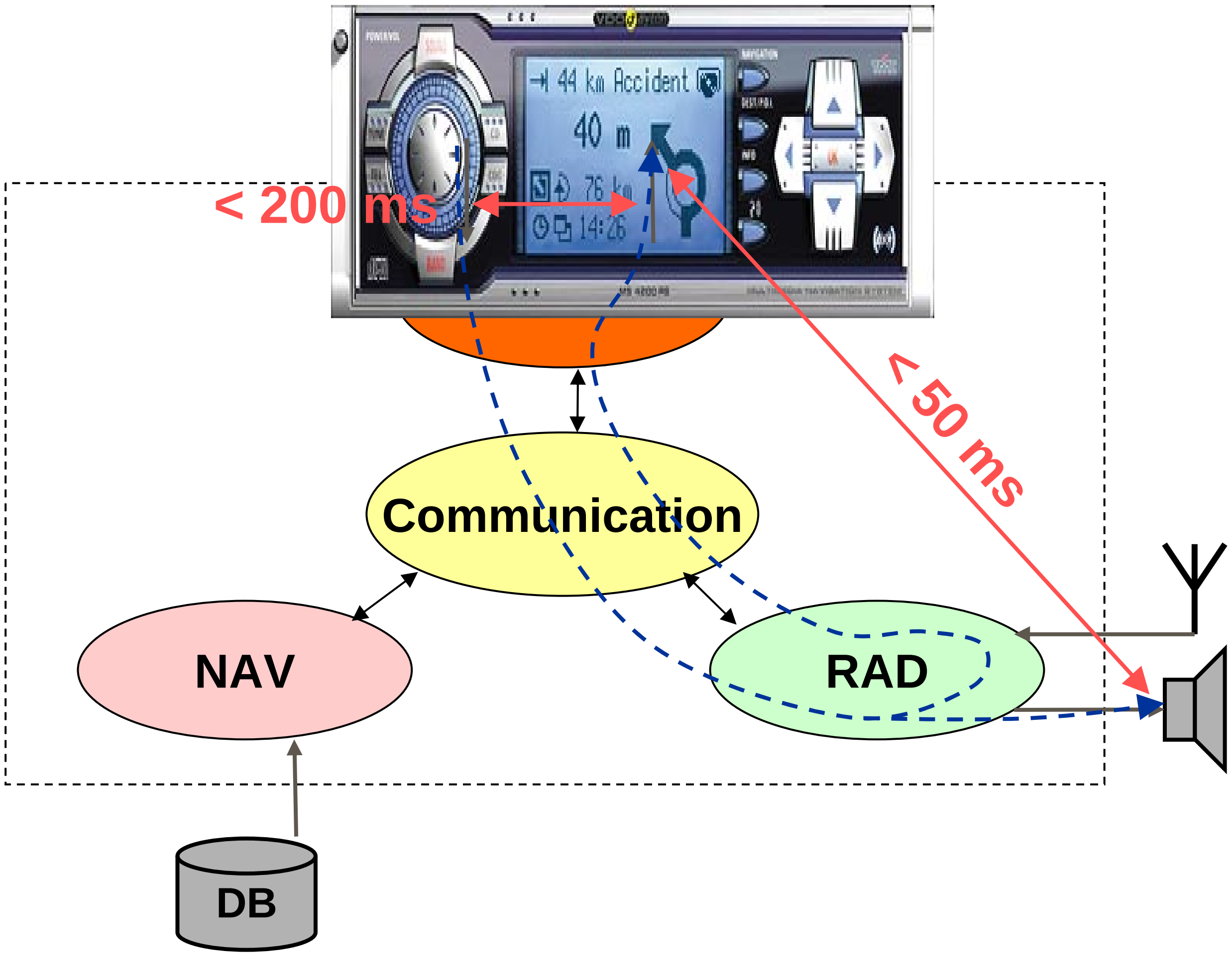
Application: In-Car Navigation System

- Car radio with navigation system
- User interface needs to be responsive
- Traffic messages (TMC) must be processed in a timely way
- Several applications may execute concurrently

System Overview

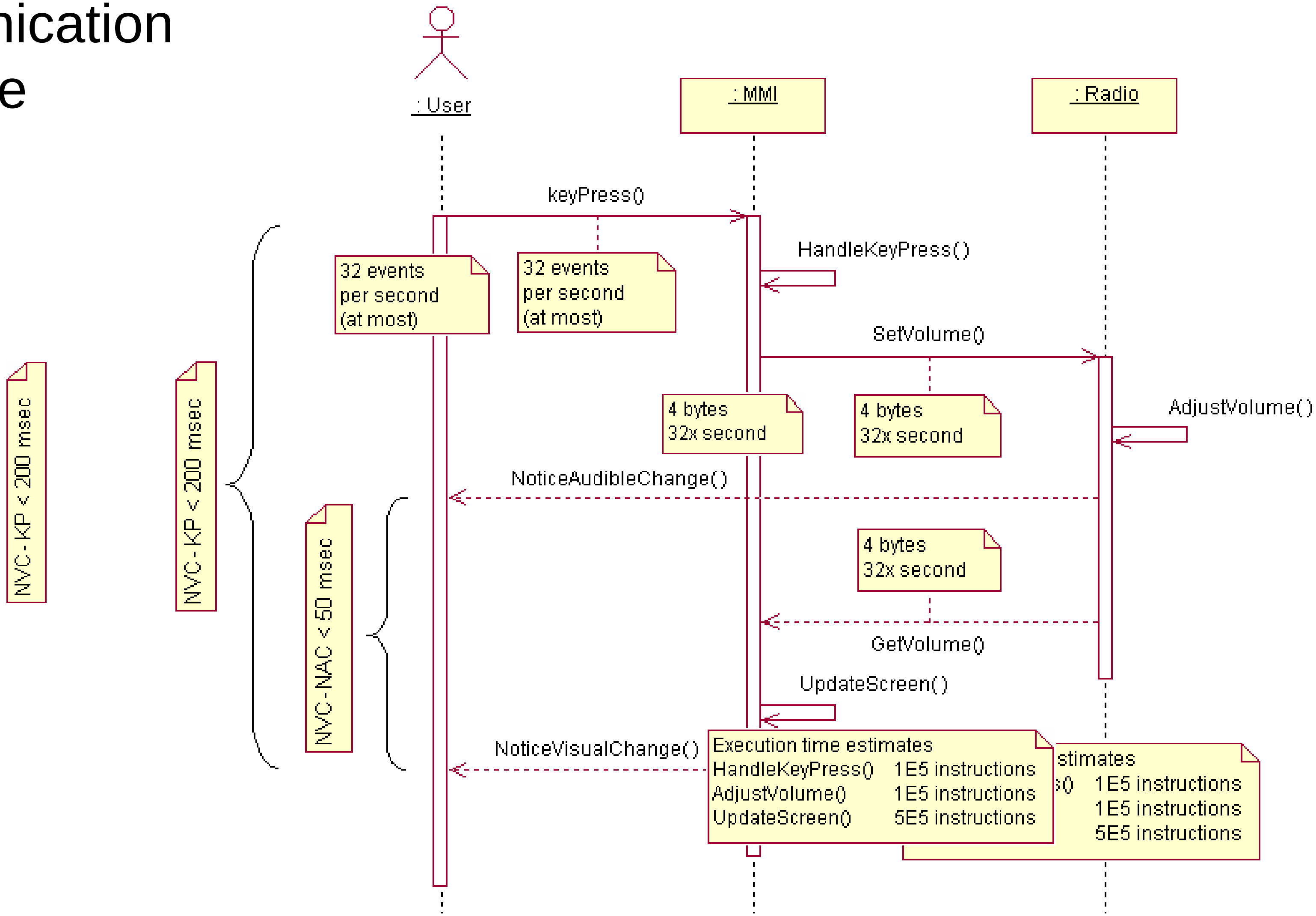


Use case 1: Change Audio Volume

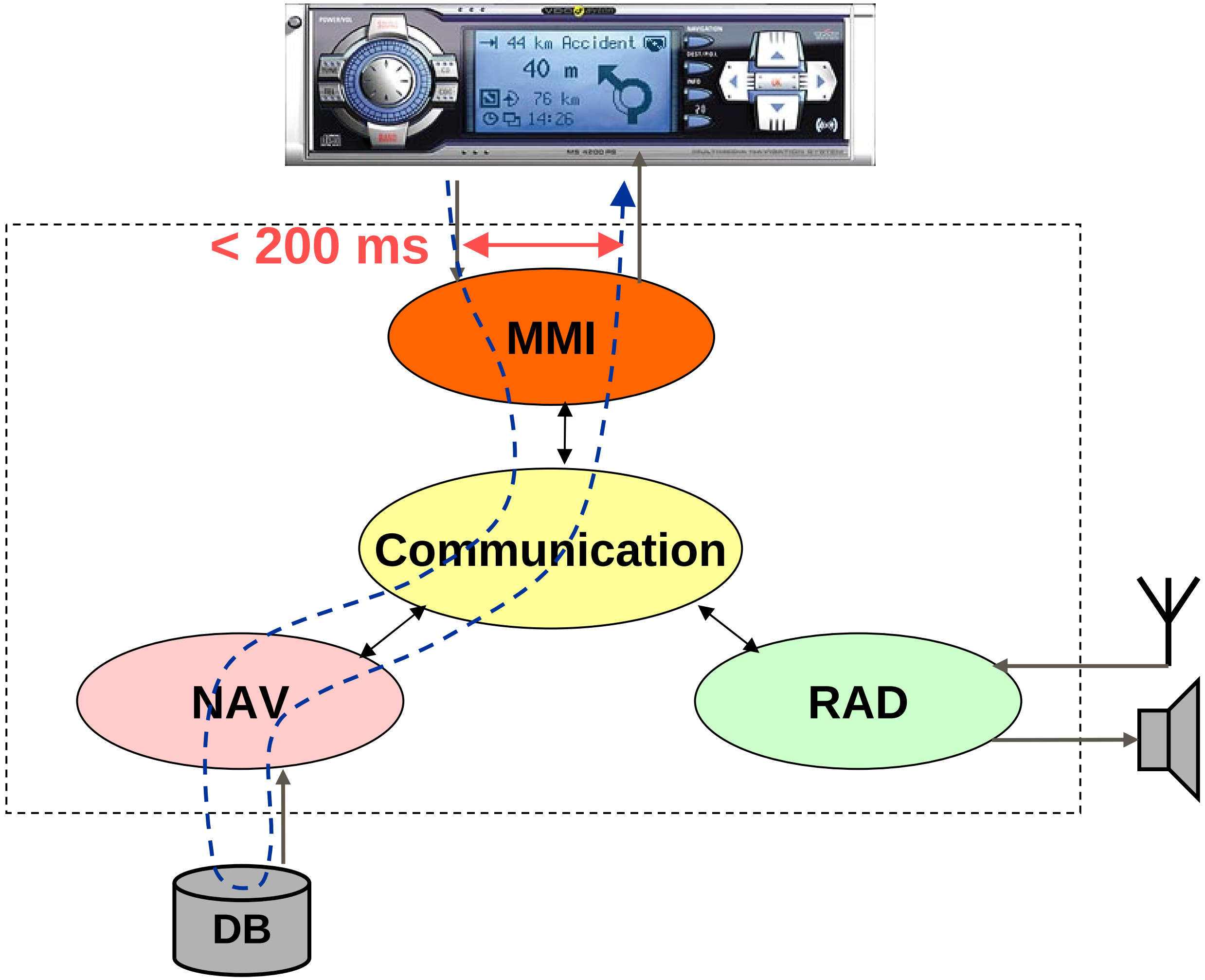


Use case 1: Change Audio Volume

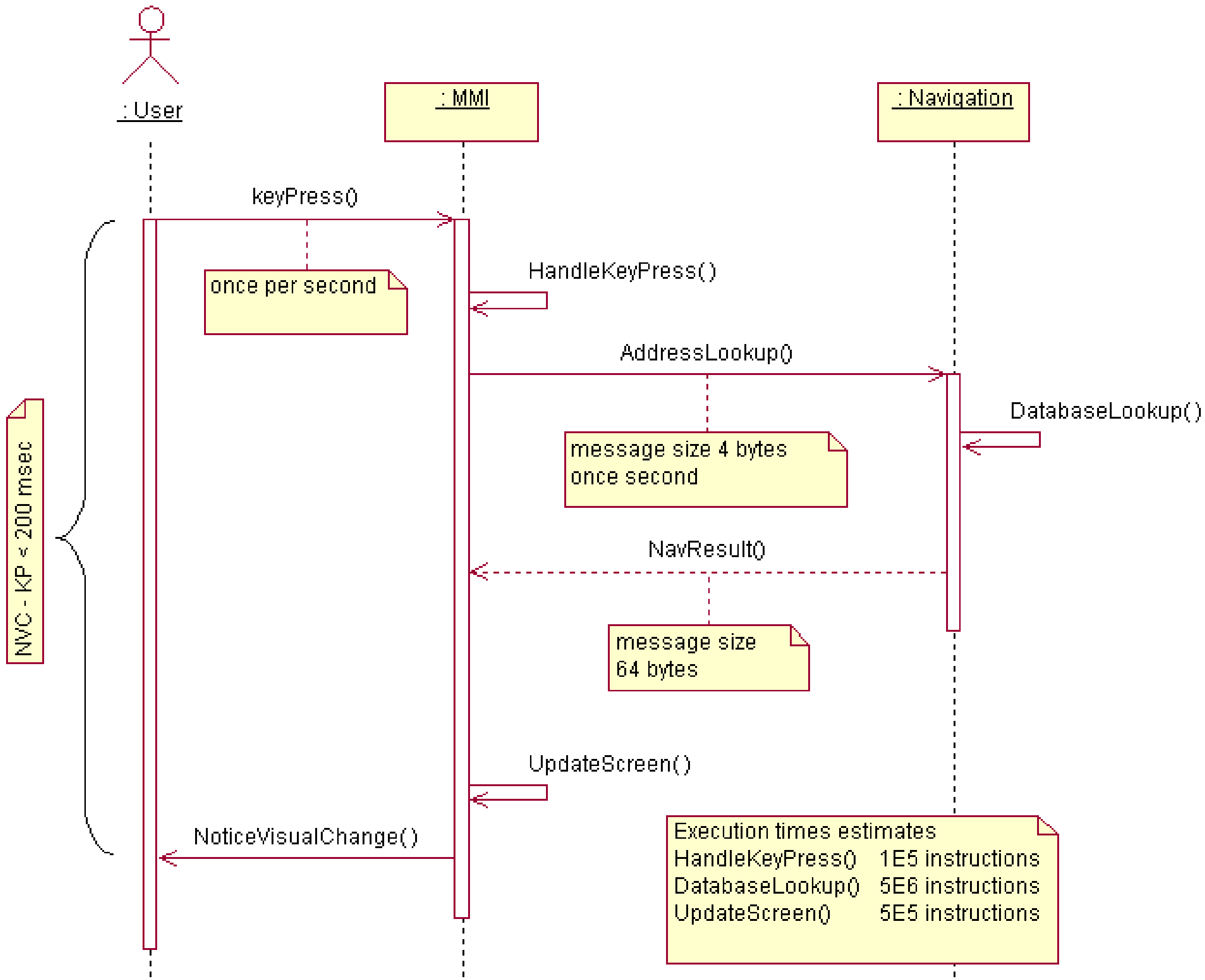
Communication
Resource
Demand



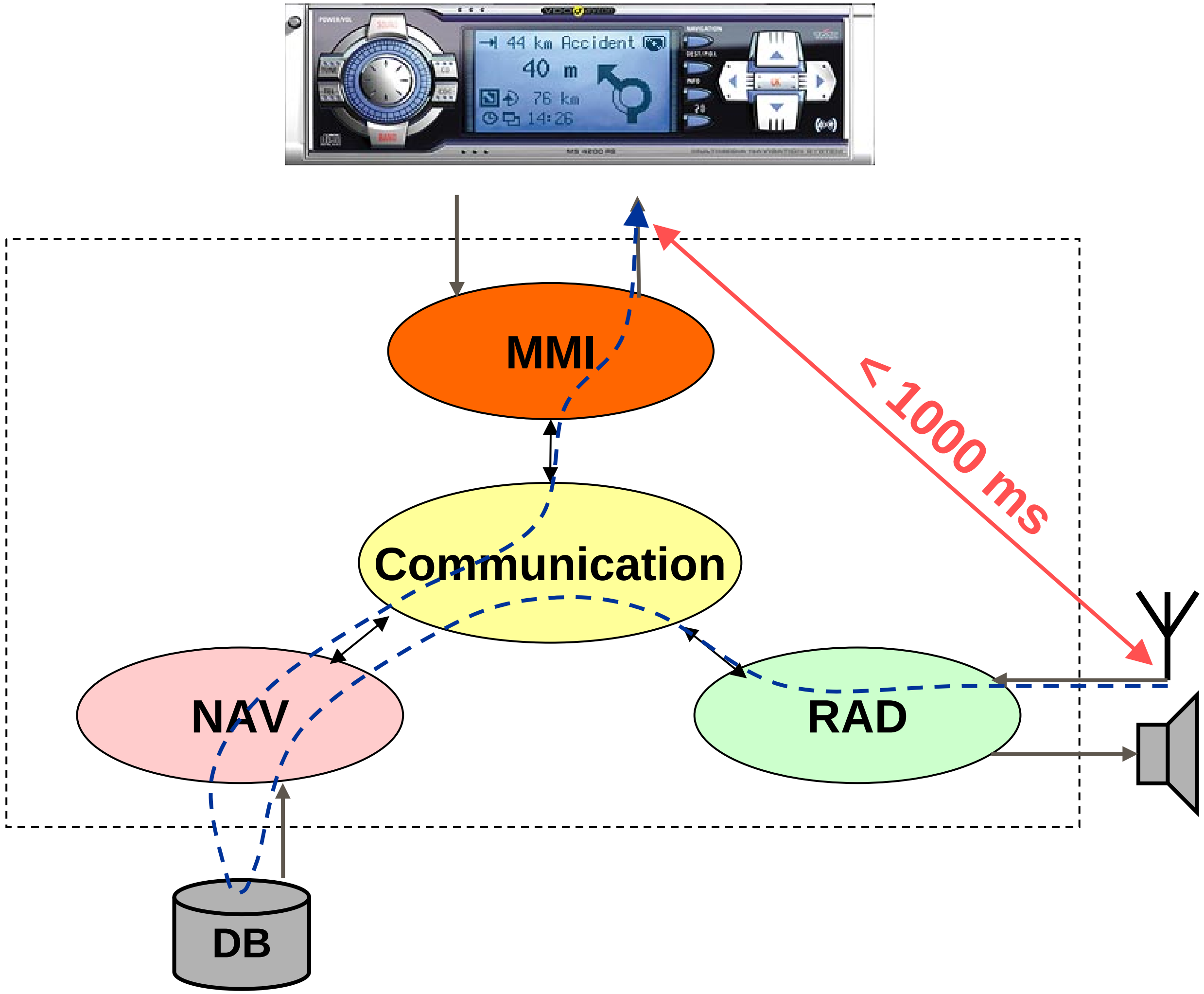
Use case 2: Lookup Destination Address



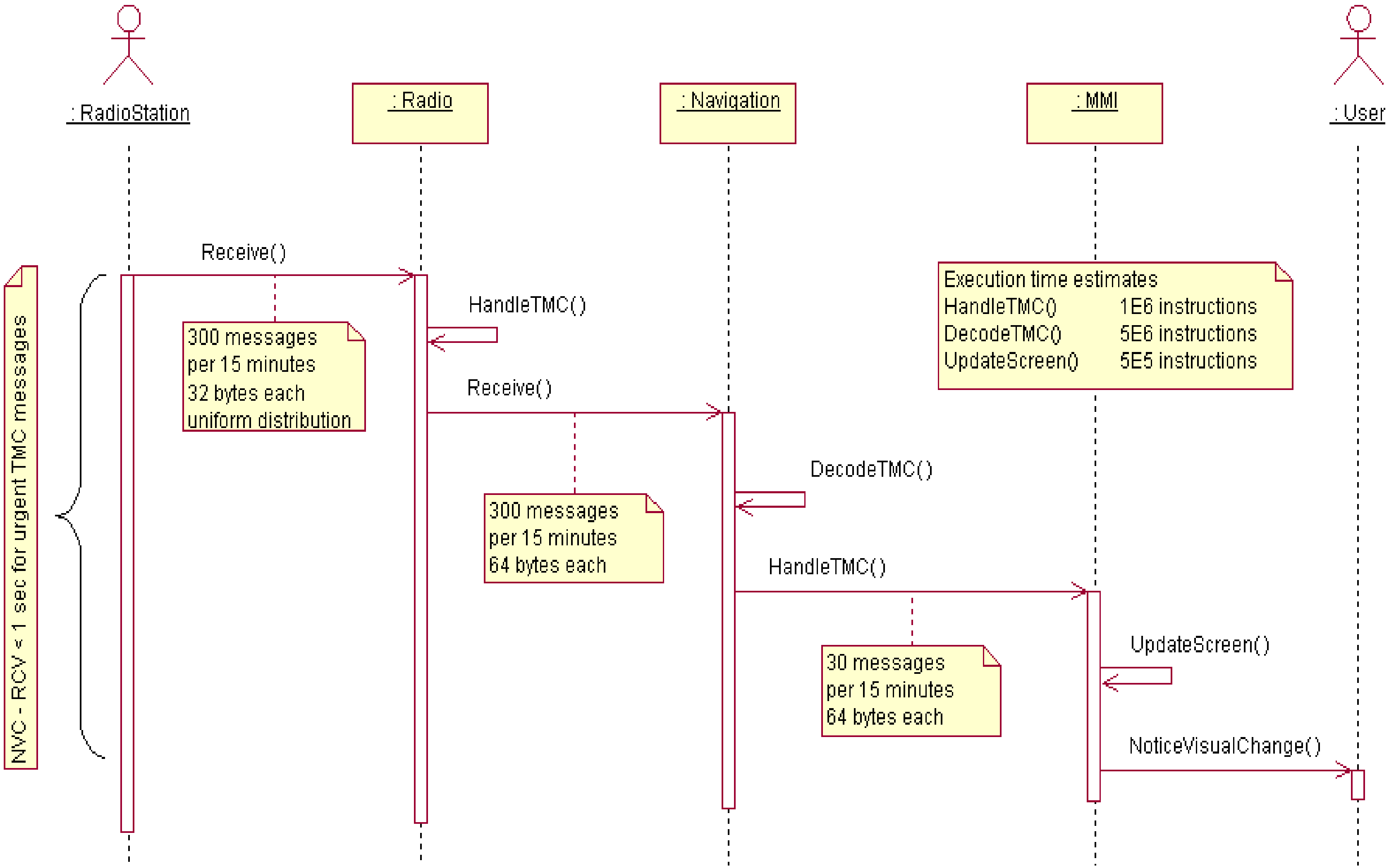
Use case 2: Lookup Destination Address



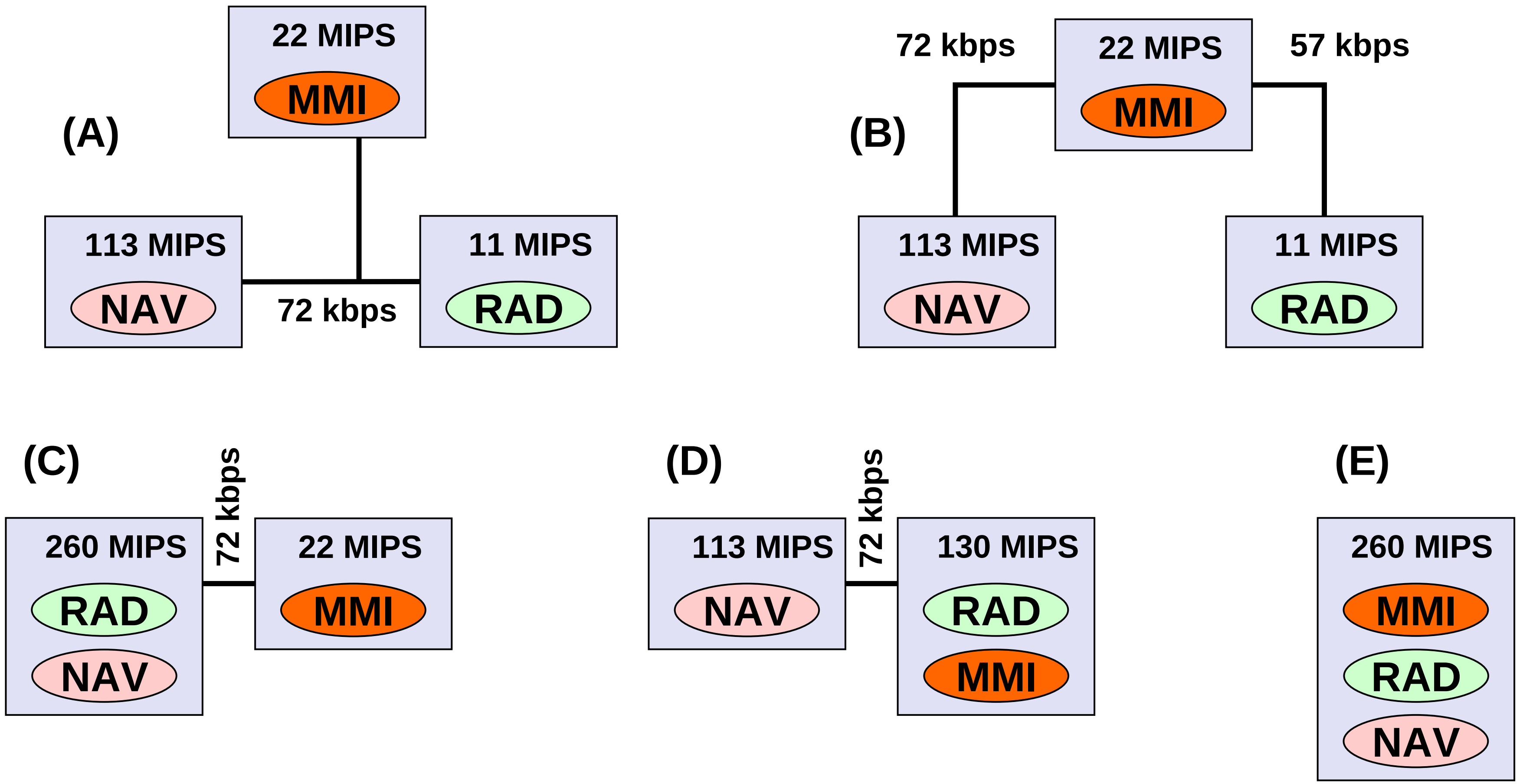
Use case 3: Receive TMC Messages



Use case 3: Receive TMC Messages



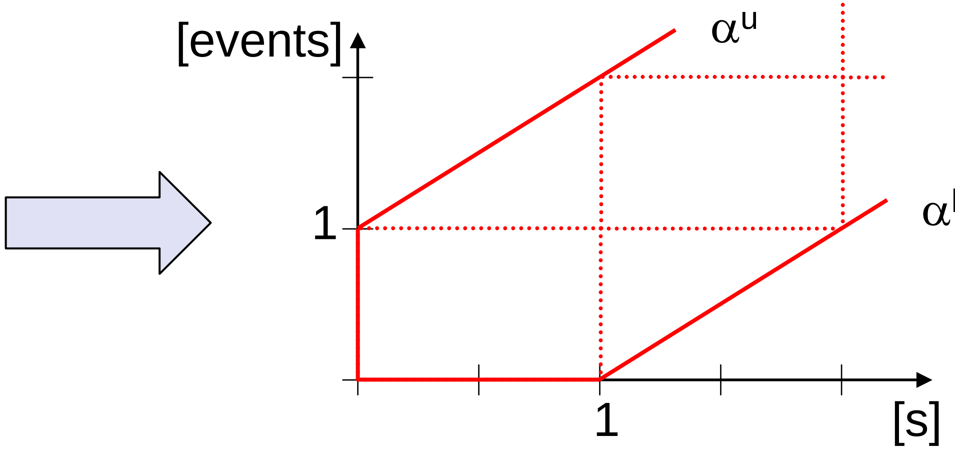
Proposed Architecture Alternatives



Step 1: Environment (Event Steams)

- Event Stream Model

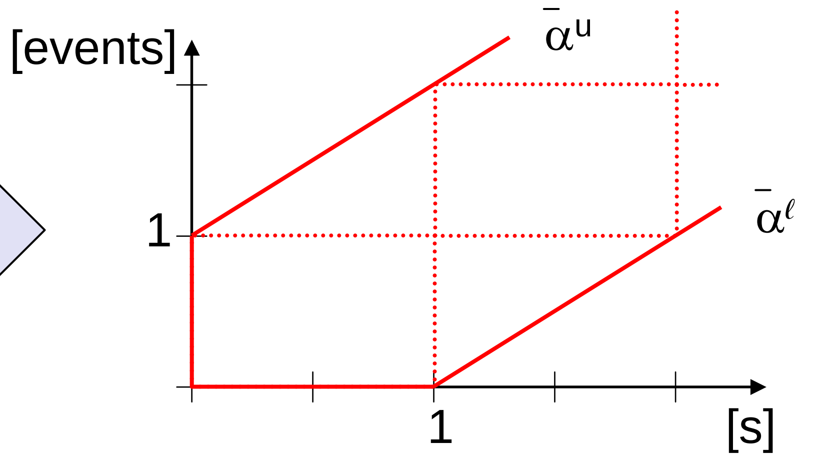
  e.g. Address Lookup
  (1 event / sec)



Step 1: Architectural Elements

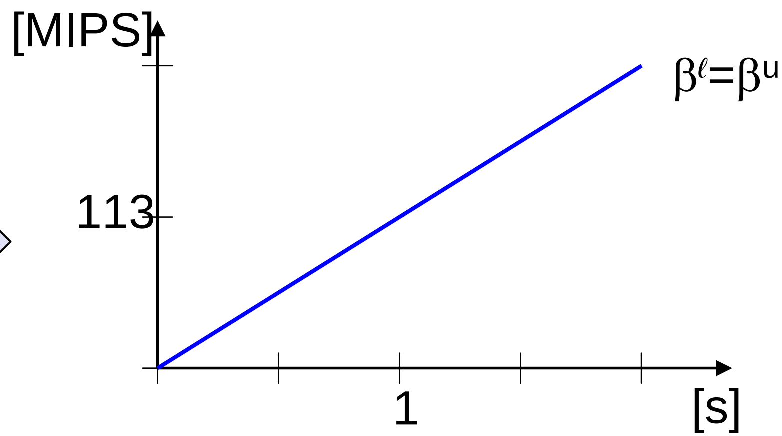
- Event Stream Model

e.g. Address Lookup
(1 event / sec)



- Resource Model

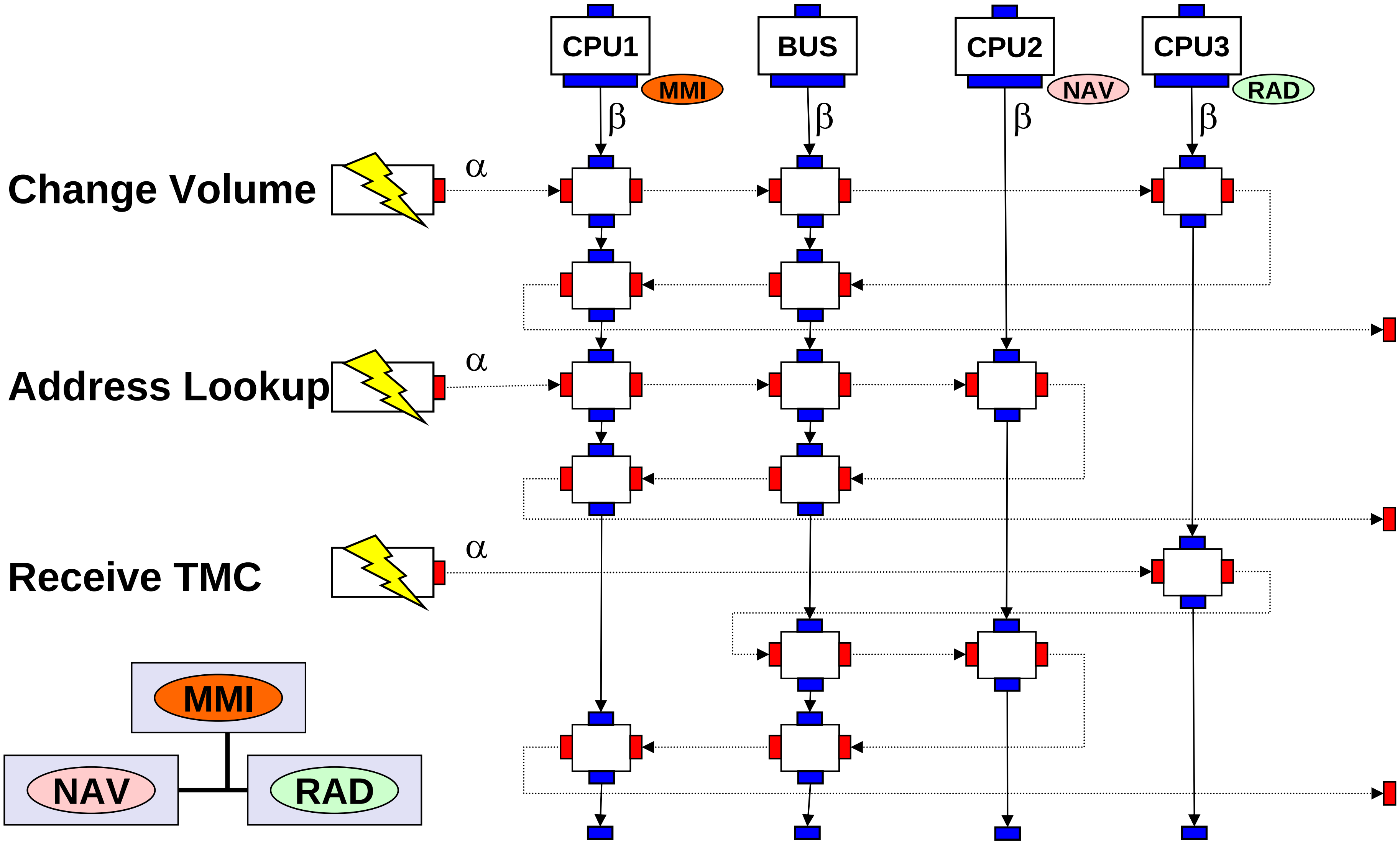
e.g. unloaded RISC CPU
(113 MIPS)



Step 2: Mapping / Scheduling

- Rate Monotonic Scheduling
(Pre-emptive fixed priority scheduling):
  - Priority 1: Change Volume (p=1/32 s)
  - Priority 2: Address Lookup (p=1 s)
  - Priority 3: Receive TMC (p=6 s)

Step 2: Performance Model

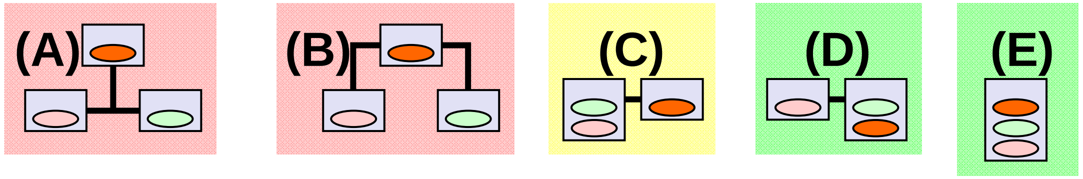


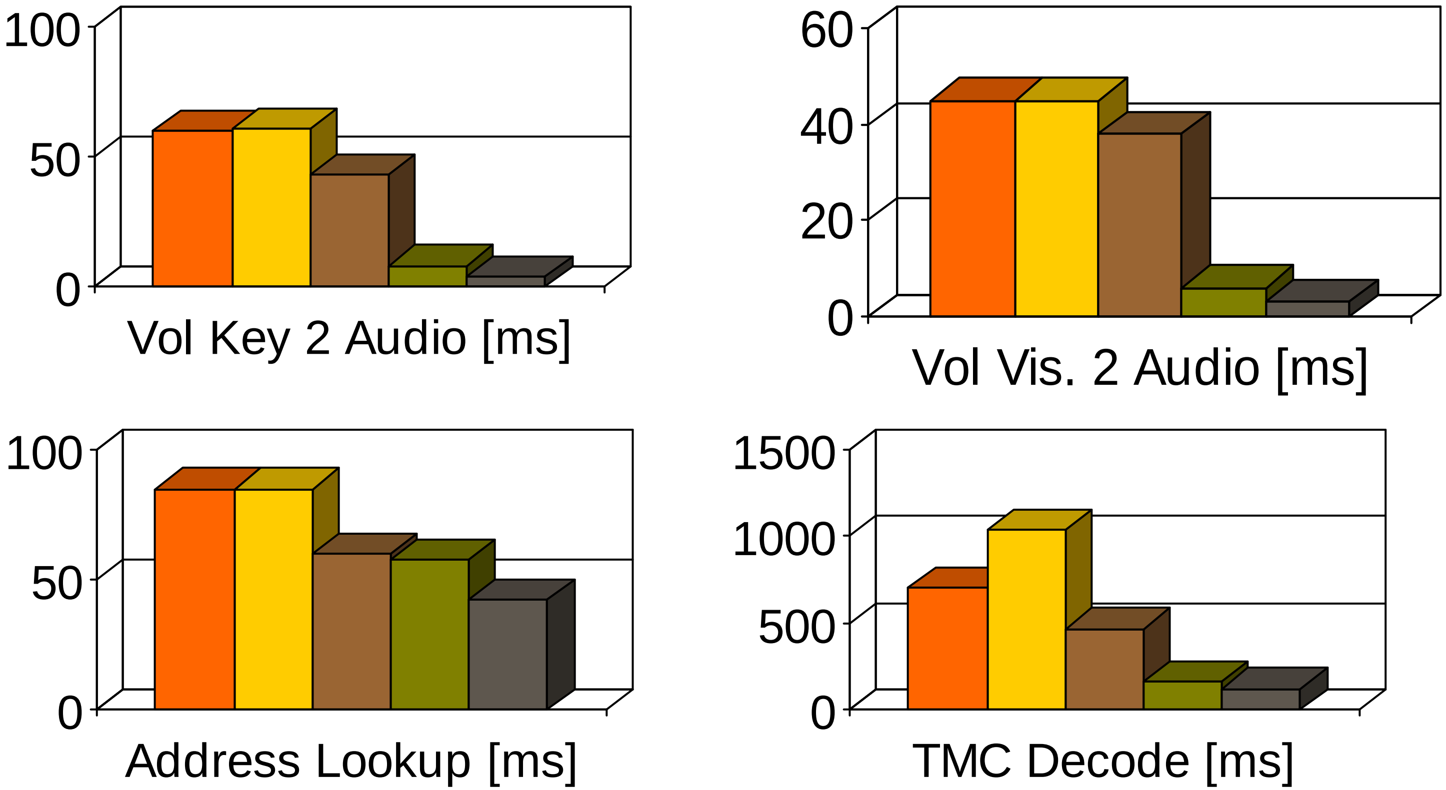
Analysis – Design Question 1

How do the proposed system architectures compare in respect to end-to-end delays?

Analysis – Design Question 1

- End-to-end delays:

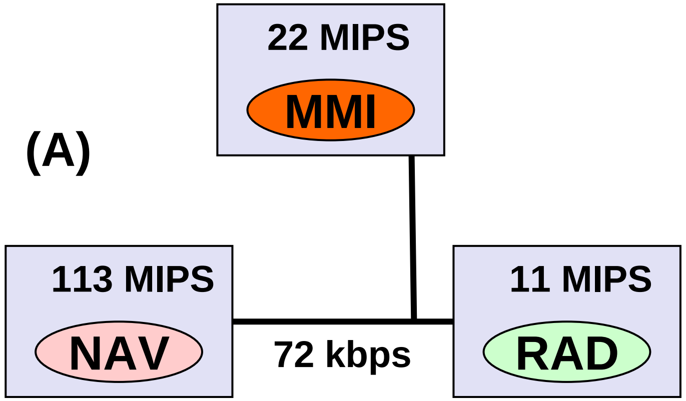




Analysis – Design Question 2

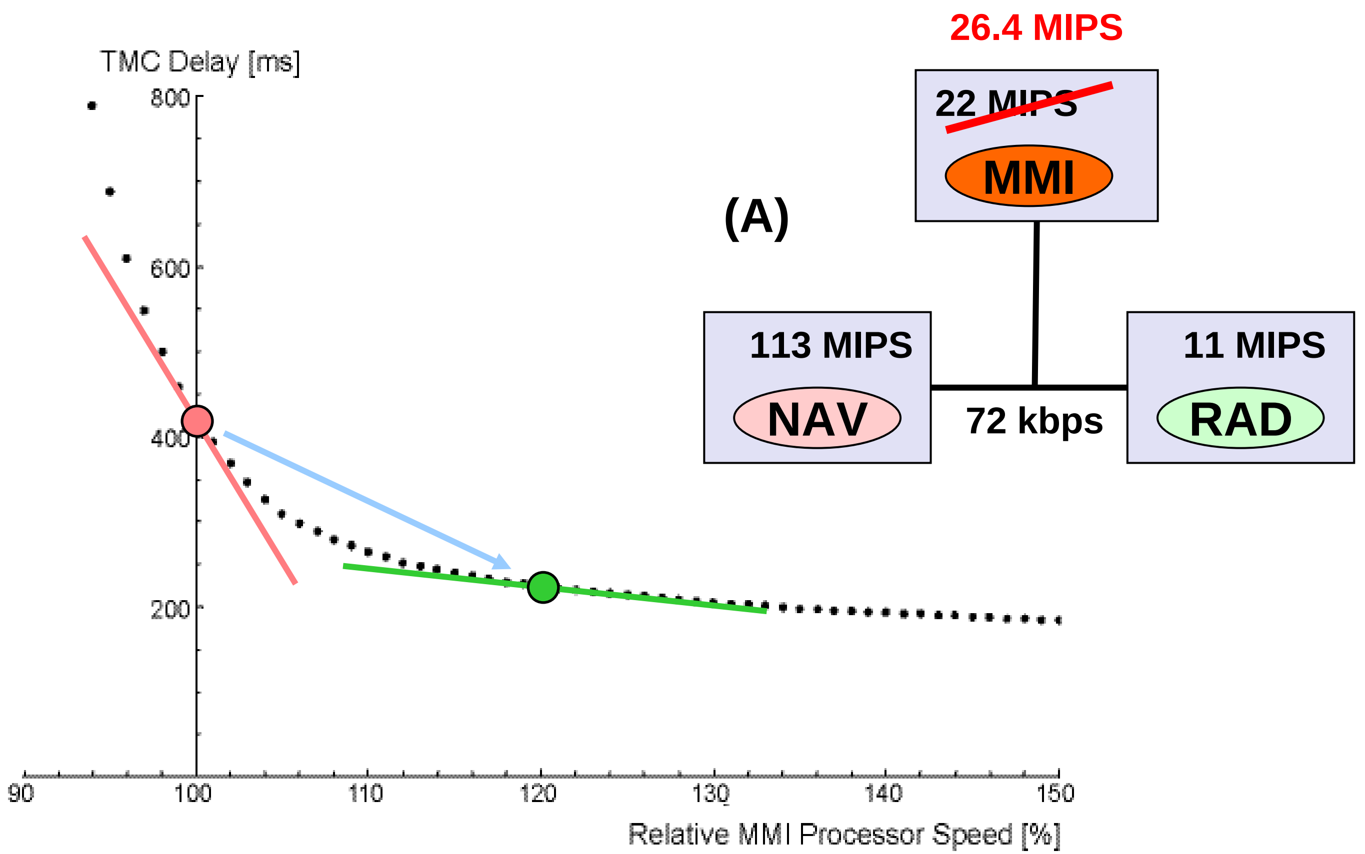
How robust is architecture A?

Where is the bottleneck of this architecture?



Analysis – Design Question 2

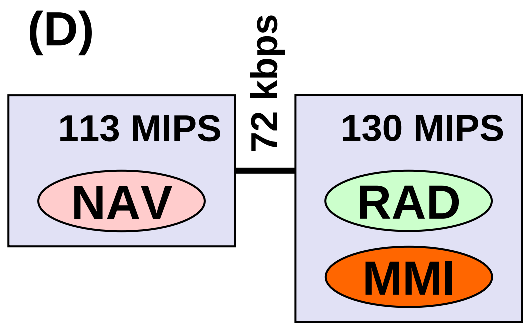
- TMC delay vs. MMI processor speed:



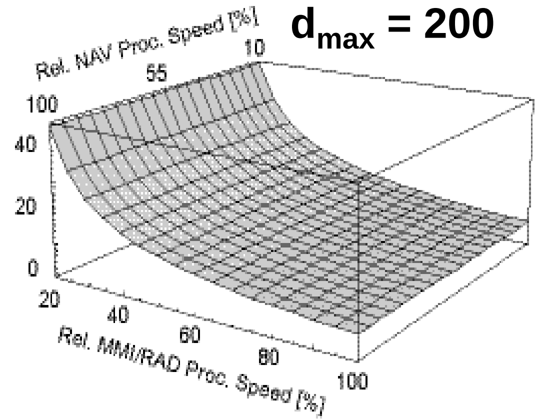
Analysis – Design Question 3

Architecture D is chosen for further investigation.

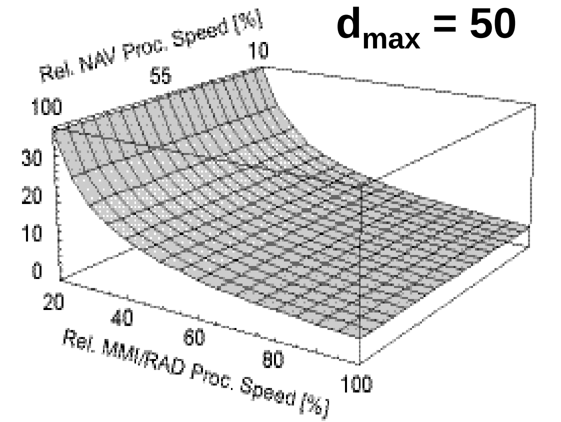
How should the processors be dimensioned?



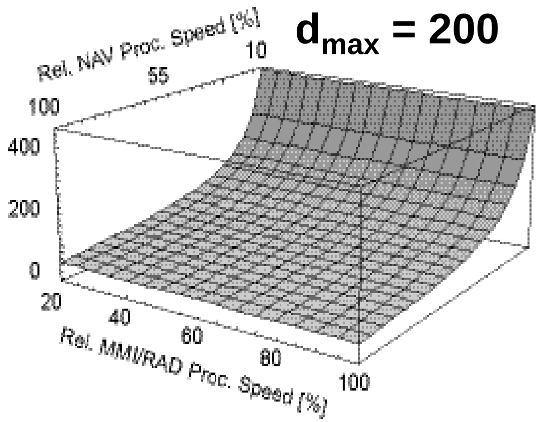
Analysis – Design Question 3



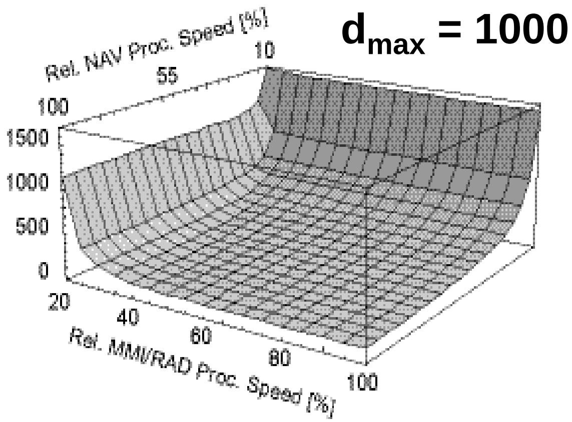
Vol K2V Delay [ms]



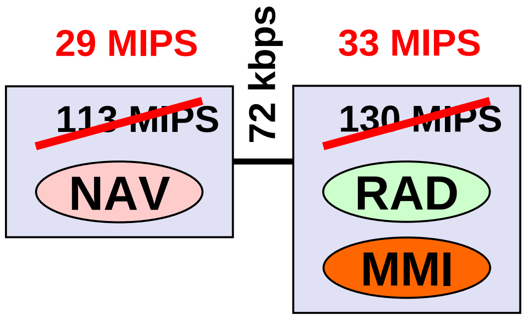
Vol A2V Delay [ms]



Addr Delay [ms]



TMC Delay [ms]



Conclusions – Realtime Calculus

- Easy to construct models
- Evaluation speed is fast and linear to model complexity (~ 1s per evaluation)
- Needs little information to construct early models (Fits early design cycle very well)
- Results conservative (may underestimate performance)