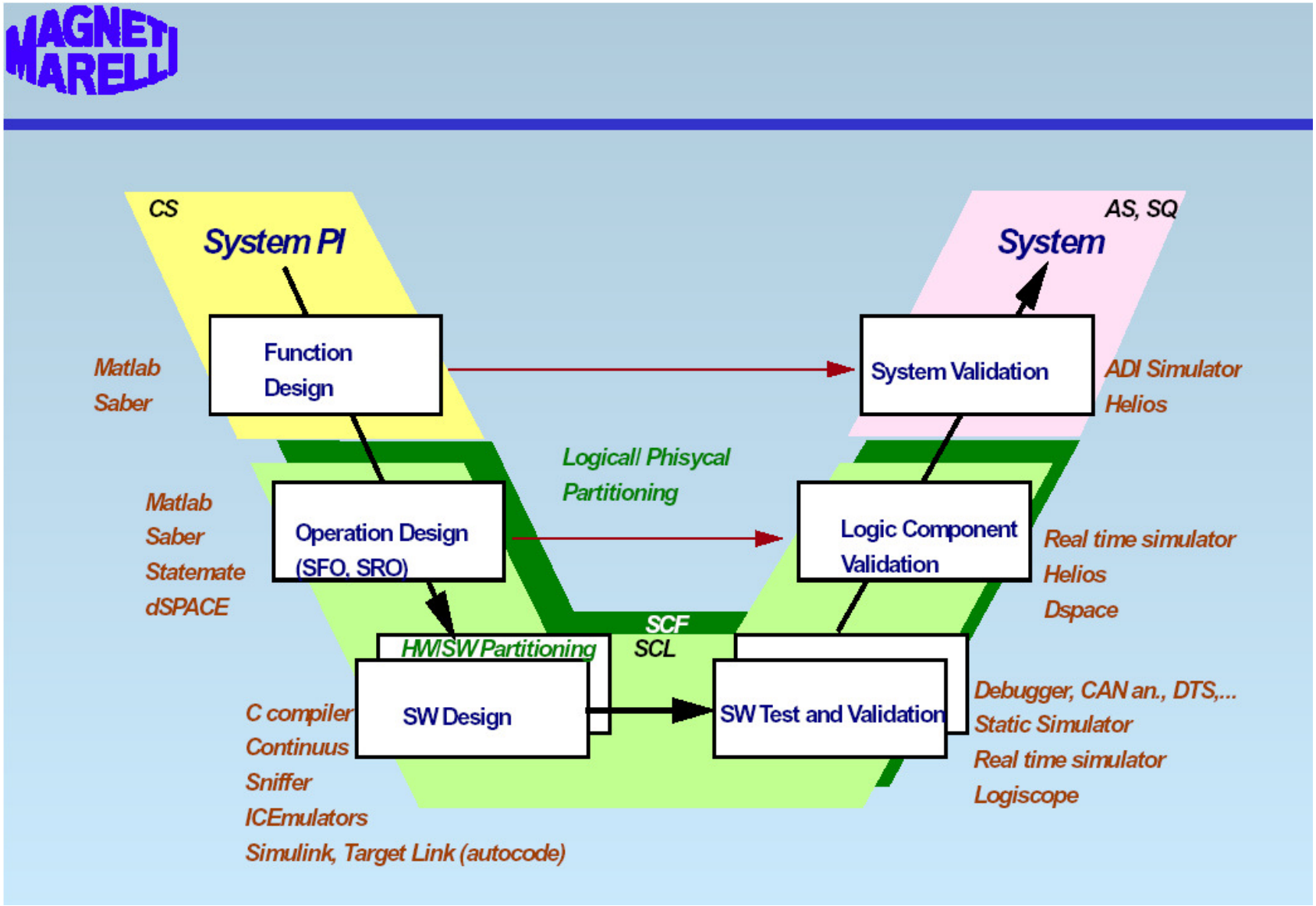
Models, tasks, RT operating systems and schedulability

Marco Di Natale
Associate Professor, Scuola S. Anna - Italy, UTRC Visiting Fellow

A development cycle



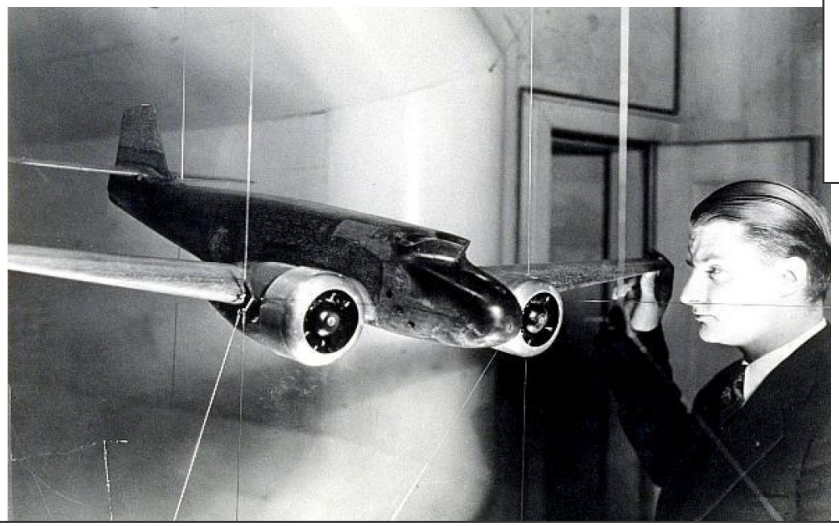
Model-based design

On August 19, 1418, a competition was announced in Florence, where the city's magnificent new cathedral, Santa Maria del Fiore, had been under construction for more than a century

Whoever desires to make any model or design for the vaulting of the main Dome of the Cathedral under construction by the Opera del Duomo-for armature, scaffolding or other thing, or any lifting device pertaining to the construction and perfection of said cupola or vault shall do so before the end of the month of September. If the model be used he shall be entitled to a payment of 200 gold Florins.

Competition between architects was an old and honored custom. Patrons had been making architects compete against one another for their commissions since at least 448 B.C., when the Council of Athens held a public competition for the war memorial it planned to build on the Acropolis. Under these circumstances, it was normal practice for architects to produce models as a means of convincing patrons or panels of judges of the virtues of their particular designs.

Model-based design



Engineering has made use of models since its very early days

Filippo Brunelleschi's design for the dome of the cathedral of Santa Maria del Fiore in Florence remains one of the most towering achievements of Renaissance architecture. Completed in 1436, the dome remains a remarkable feat of design and engineering. Its span of more than 140 feet exceeds St Paul's in London and St Peter's in Rome, and even outdoes the Capitol in Washington, D.C., making it the largest dome ever constructed using bricks and mortar. When work on the dome began in 1420 Brunelleschi was virtually unknown. Sixteen years later the dome was built, and its architect was a superstar.

Model-based design flow

- Typical flow, updated in V-shape or iterative fashion or V-shape plus iterative ….
- The four tenets on the right are fundamental to model-based design
- Of course, you must select a modeling language that allows to do everything in the most natural and easy way …

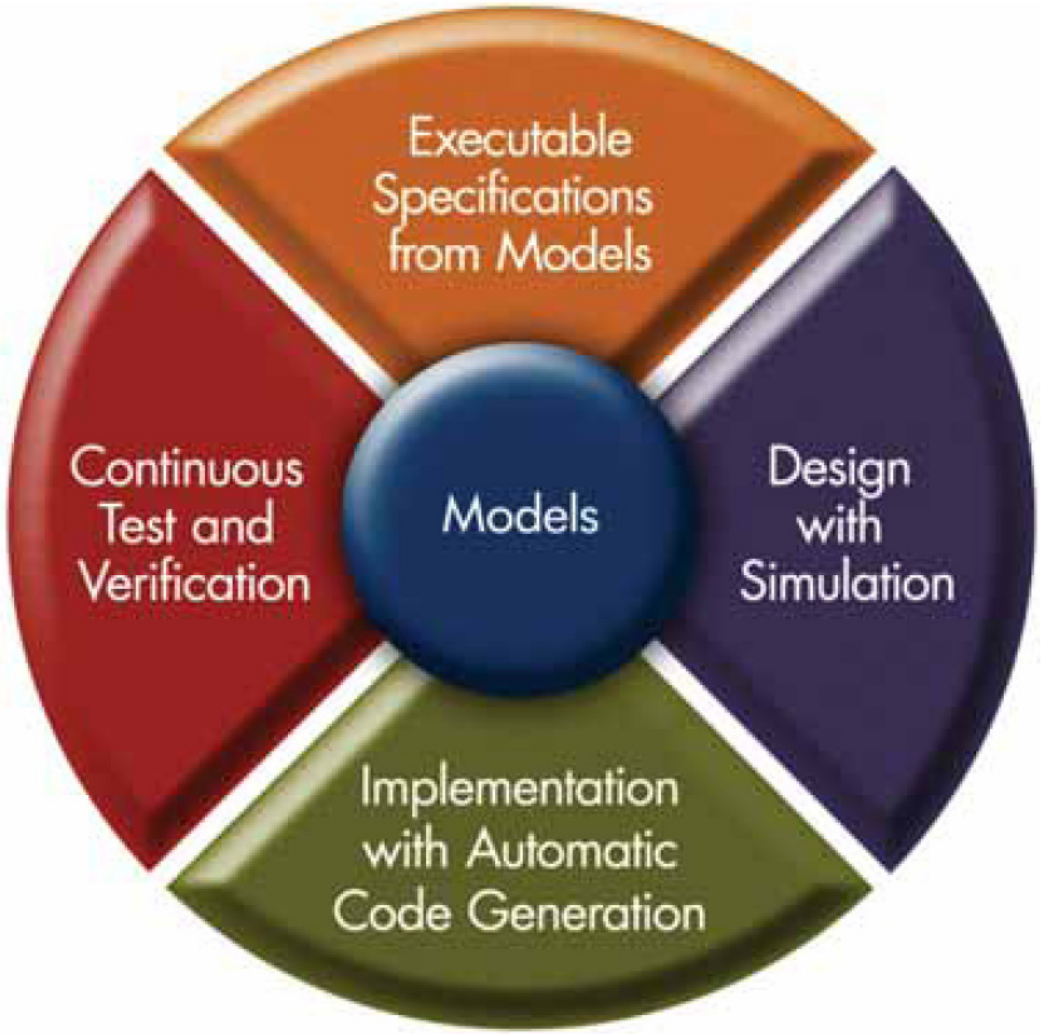
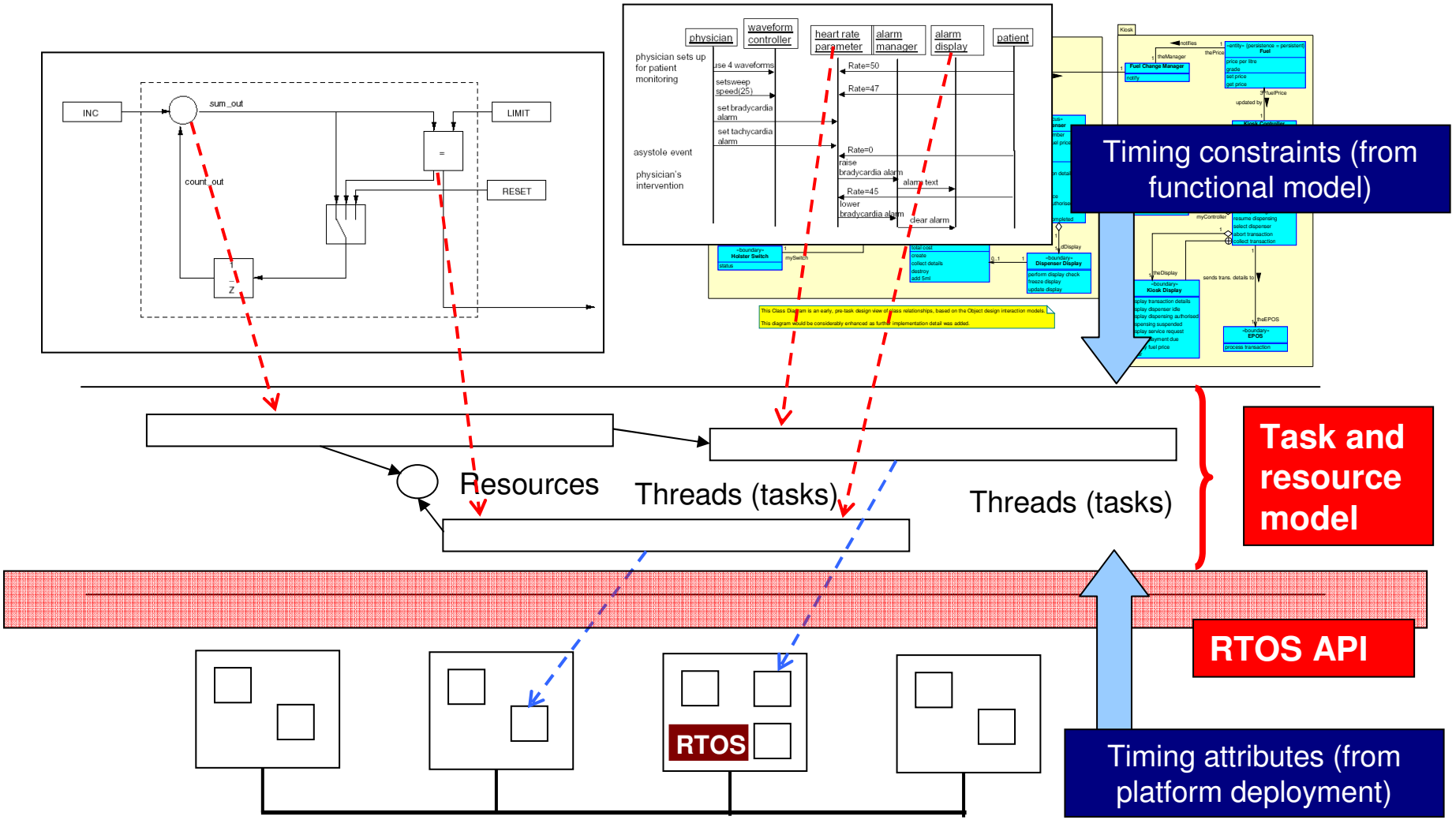


Figure 1 – Elements of model-based design

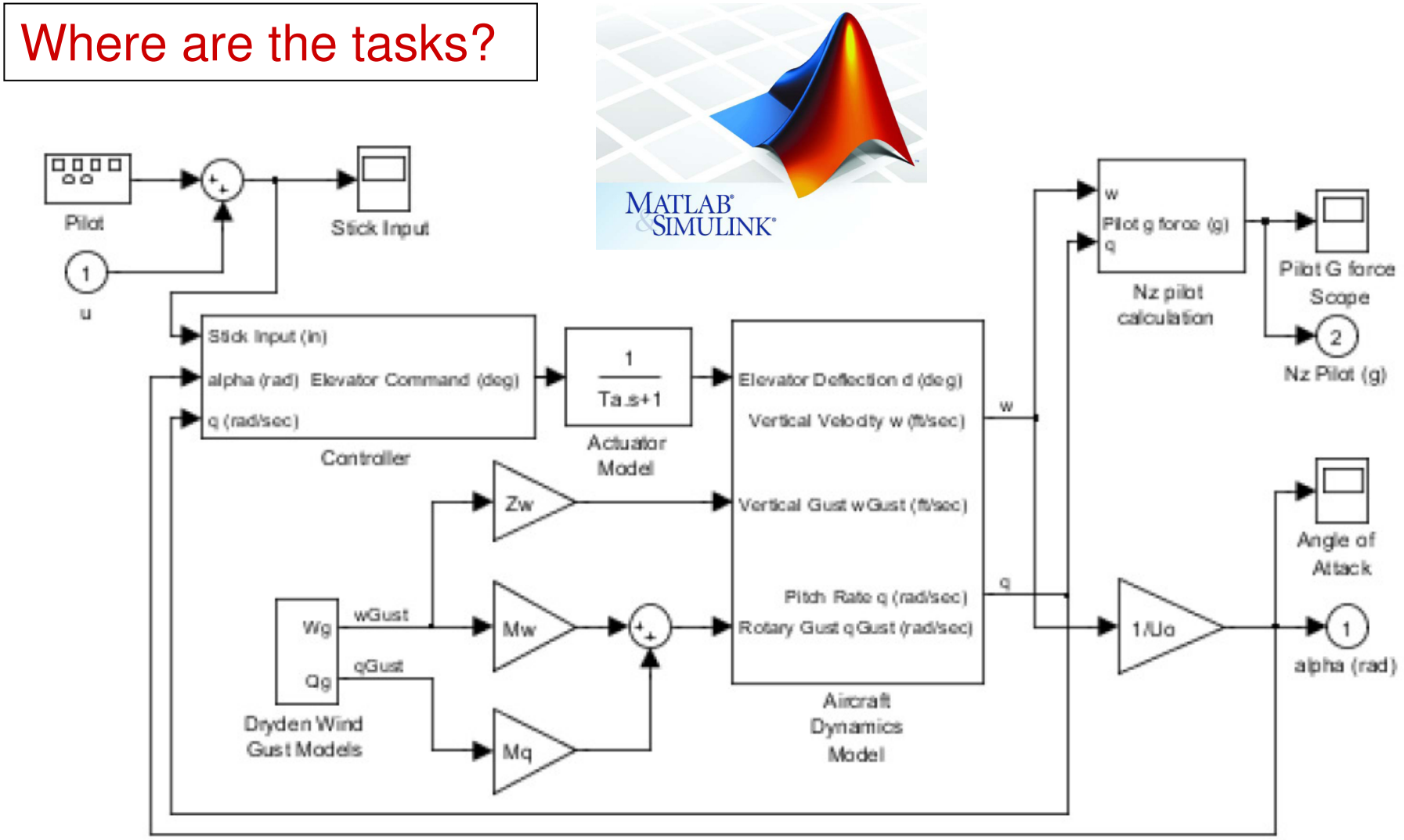
RTS and Platform-Based Design

- Design (continued): matching the logical design into the SW architecture design

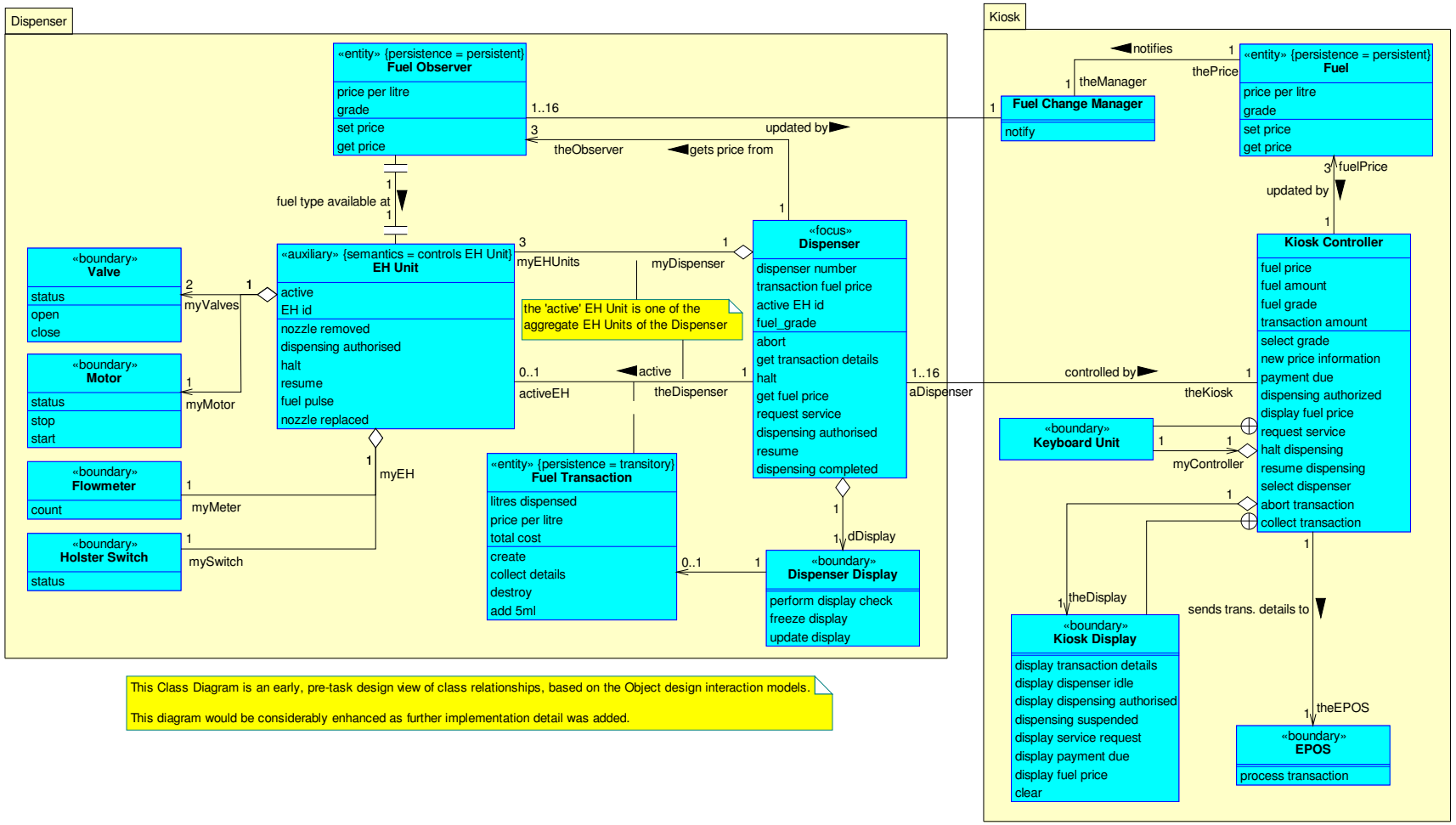


Models and implementation: Simulink

Where are the tasks?

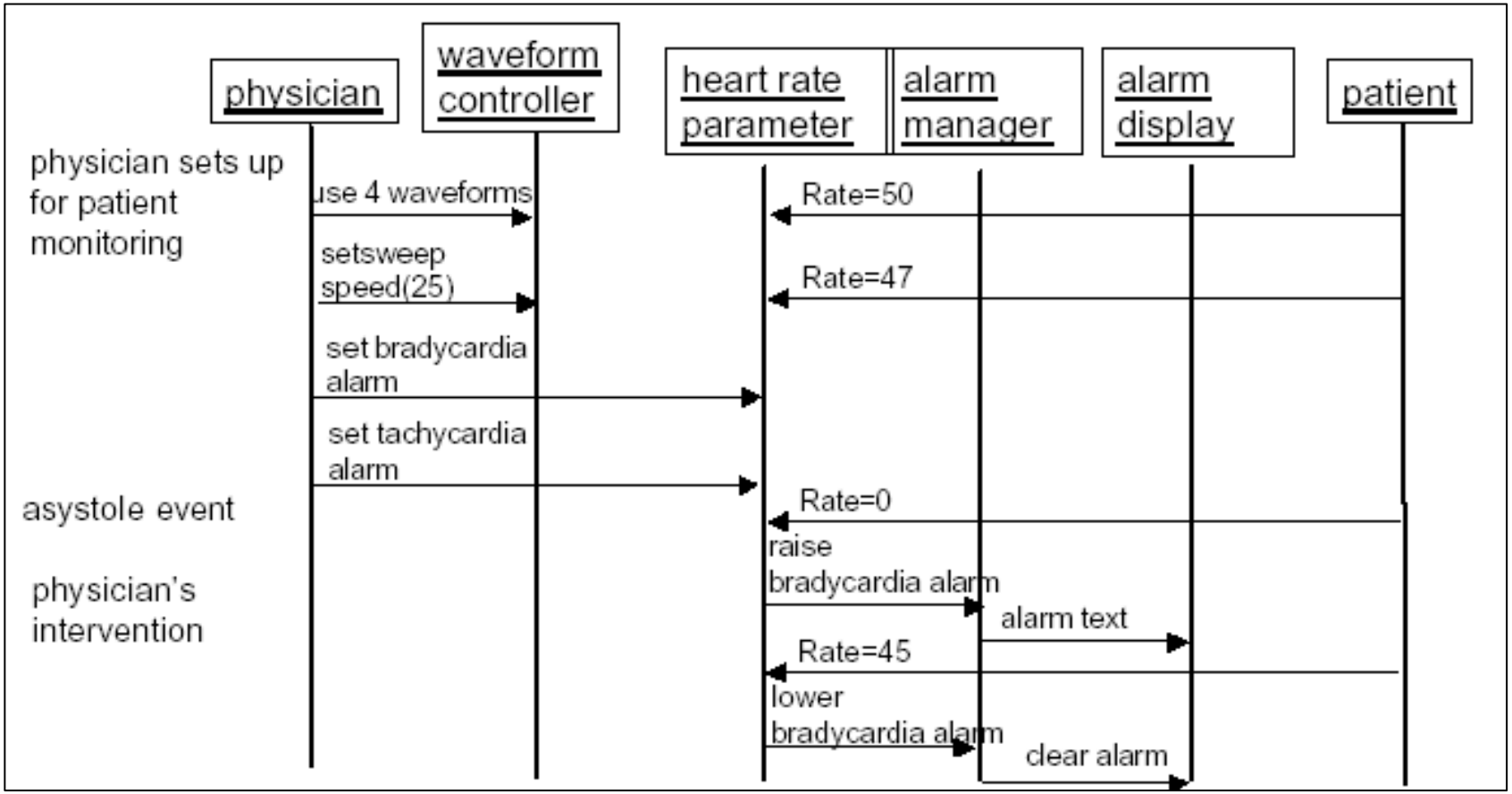


Models and implementation: UML

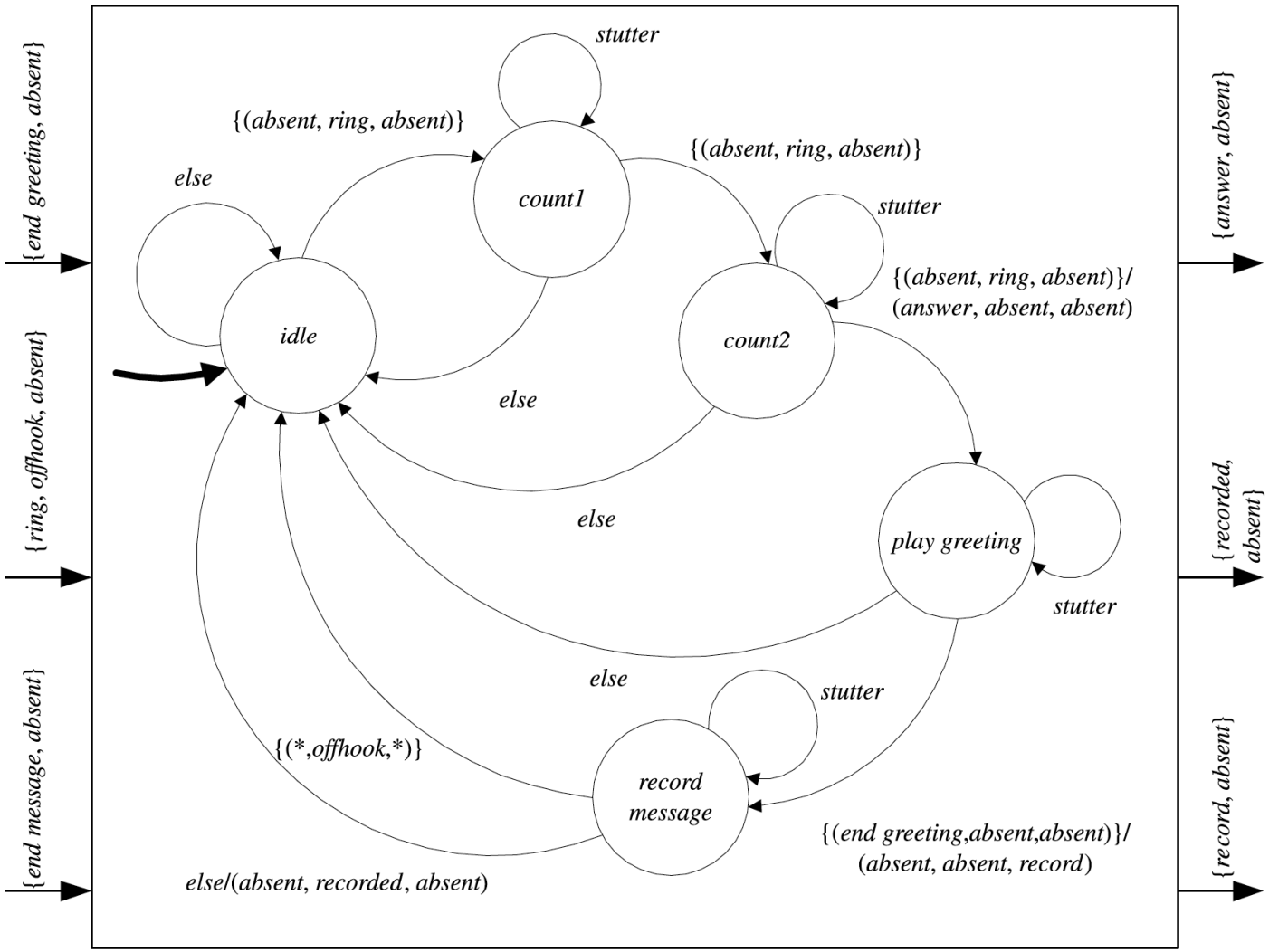


Where are the tasks?

Models and implementation: UML



Models and implementation: FSM



NOTE: *stutter* = {(*absent*, *absent*, *absent*)}

Model-based design: a functional view

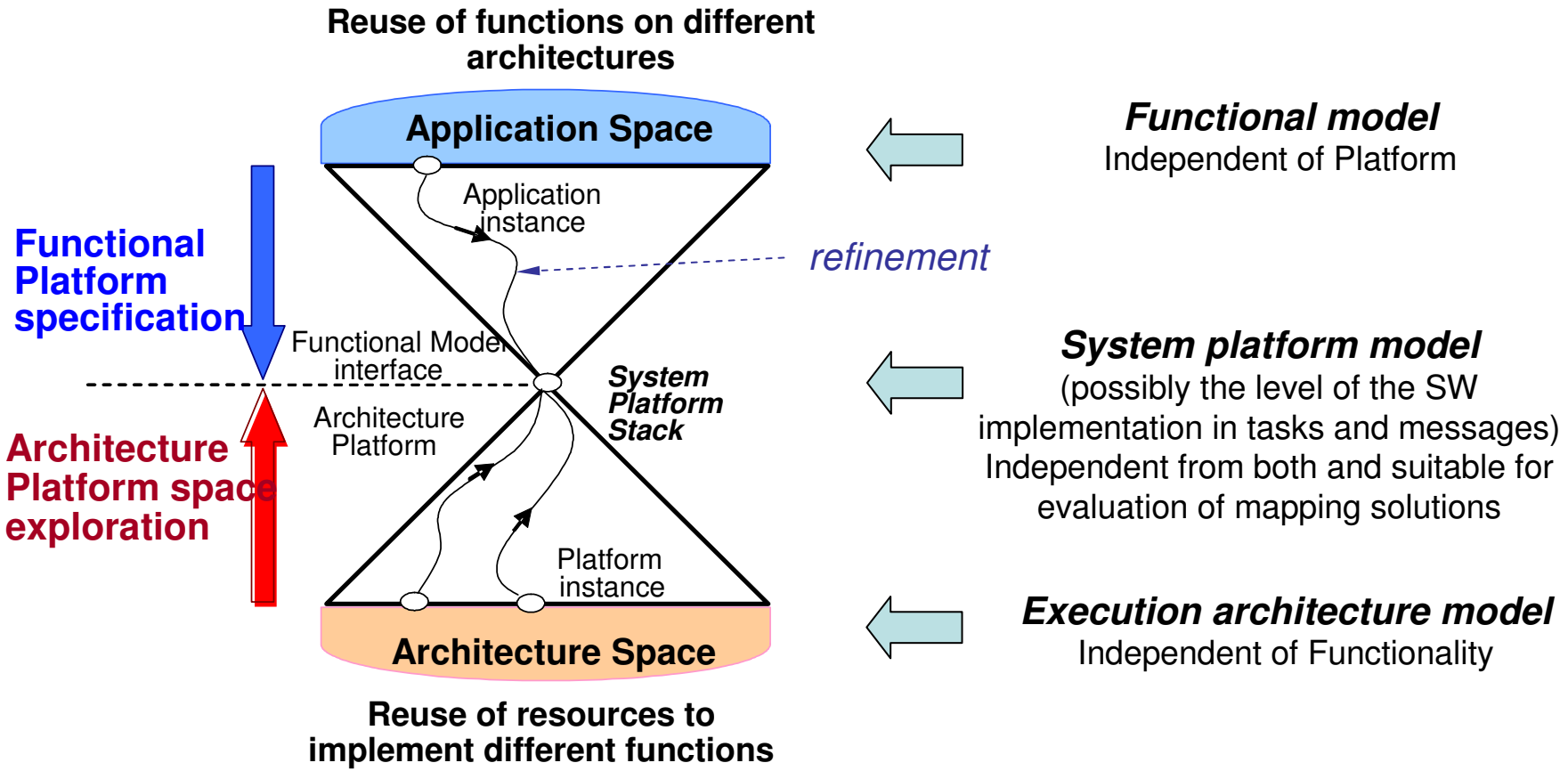
- Advantages of model-based design
  - Possibility of advance verification of correctness of (control) algorithms
- Possible approaches
  1. ***The model is developed considering the implementation and the platform limitations***
  - include from the start considerations about the implementation (tasking model and HW)
    - PROS (apparent)
      - use knowledge about the platform to steer the design towards a feasible solution (in reality, this is often a trial-and-error manual process)
    - CONS (true)
      - the model depends on the platform (updates/changes on the platform create opportunities or more often issues that need to be solved by changing the model)
      - Analysis is more difficult, absence of layers makes isolating errors and causes of errors more difficult
      - the process is rarely guided by sound theory (how good is the platform selection and mapping solution?)
      - Added elements (Rate-transition blocks) introduce delays
  2. ***The model is developed as a "pure functional" model according to a formally defined semantics, irrespective of the possible implementation***
  - The model is then refined and matched to a possible implementation platform. Analysis tools check feasibility of an implementation that refines the functional semantics and suggest options when no implementation is feasible (more …)

Model-based design: a functional view

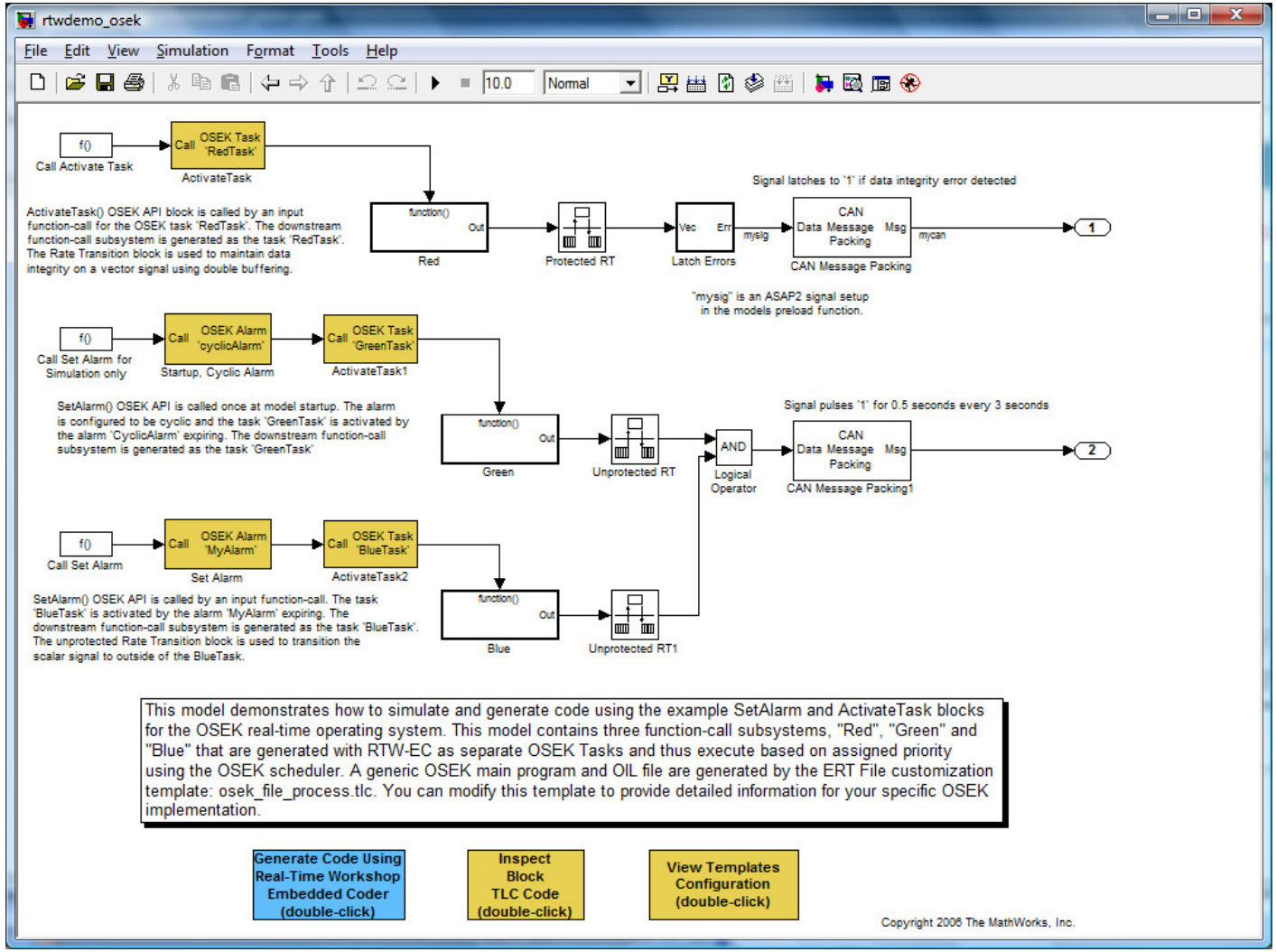
- Advantages of model-based design starting from a purely functional model
  - Possibility of advance verification of correctness of (control) algorithms
  - Irrespective of implementation
  - This allows an easier retargeting of the function to a different platform if and when needed
    - The functional design does not depend on the platform
  - The verification of the functional design can be perfomed by domain experts (control engineers) without knowledge of SW or HW implementation issues
- Necessary assets to leverage these advantages …
  - Capability of defining rules for the correct refinement of a functional model into an implementation model on a given platform
  - Capability of supporting design iterations to understand the tradeoffs and the changes that are required when a given functional model cannot be refined (mapped) on a given platform

Model-based development flow

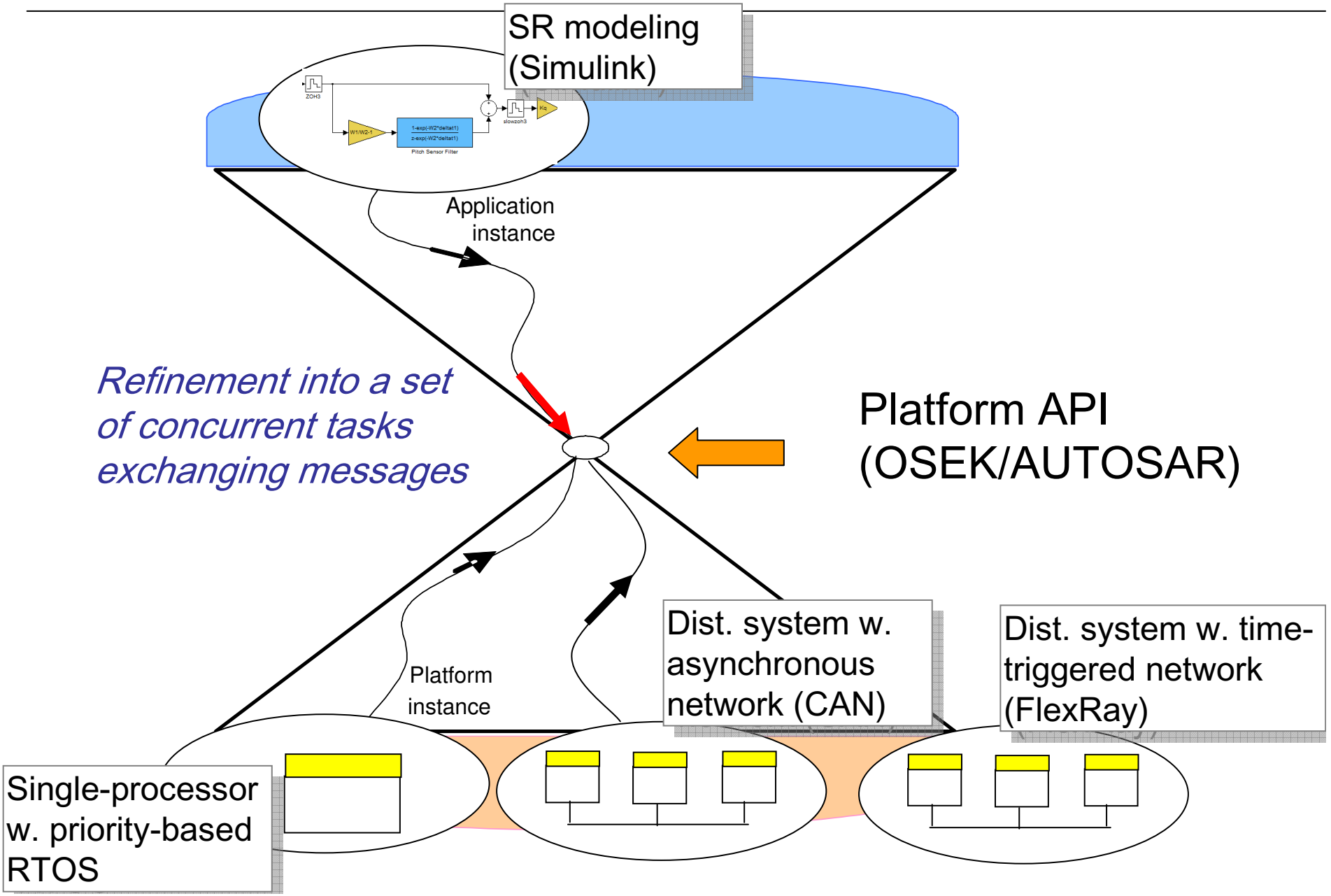
- Platform-based design



Platform-dependent modeling: an example



PBD and RTOS/platform

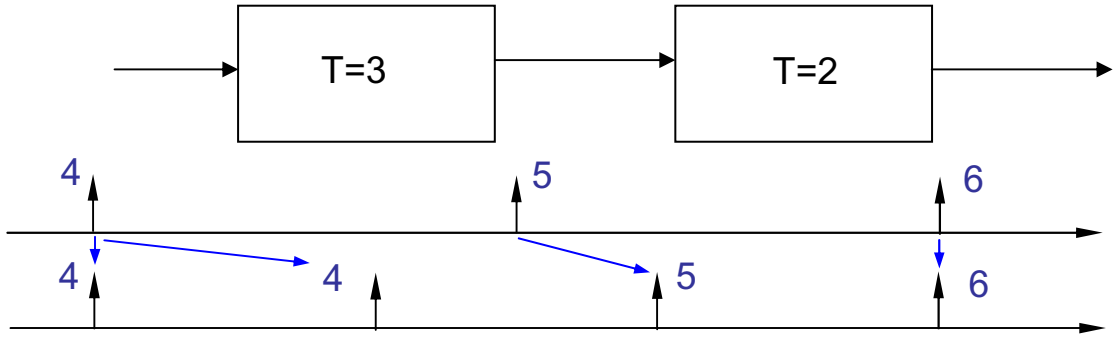


Choosing a functional representation

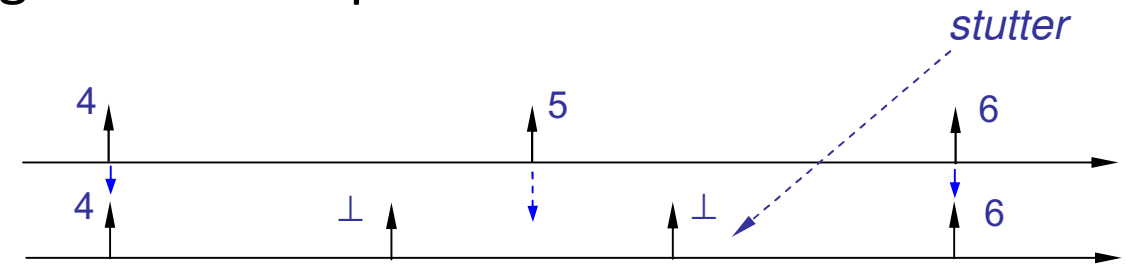
- Synchronous reactive modeling
- Purely functional implies “zero-time” execution or logical time (no notion of platform or computation time)
  - The output update and state update functions are computed immediately at the time the block is triggered/activated
  - ***Rather than “zero time”, a more accurate definition is:***
  - ***“the system response or reaction is guaranteed to be completed before the next system event”.***
  - The only significant references to time are the sampling times (or trigger events) of blocks
  - Also, the partial order in the execution of blocks because of feedthrough behavior must be considered
- Options:
  - Signals are persistent (Simulink)
  - Signals are not persistent

Semantics options

- Signals are persistent (Simulink)



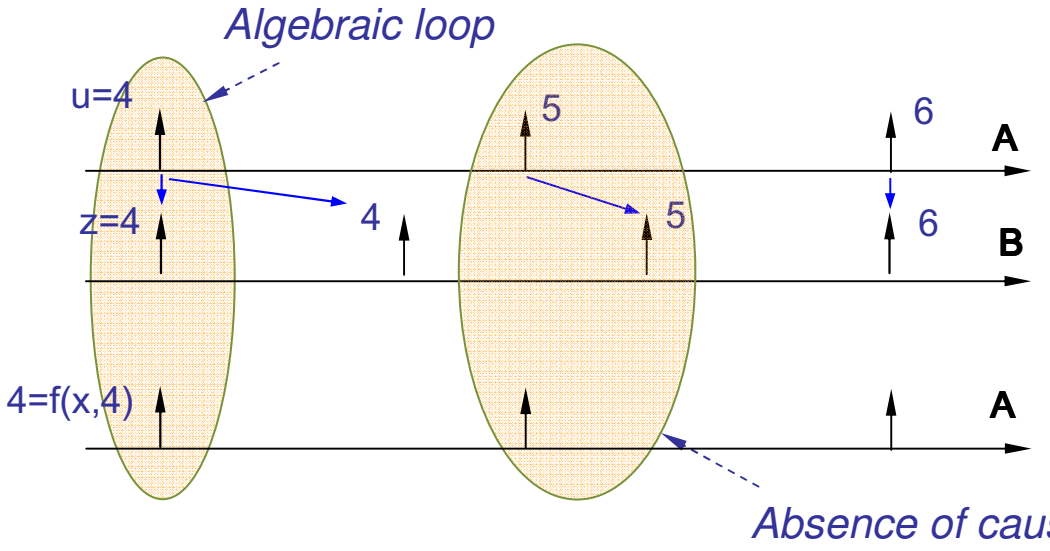
- Signals are not persistent

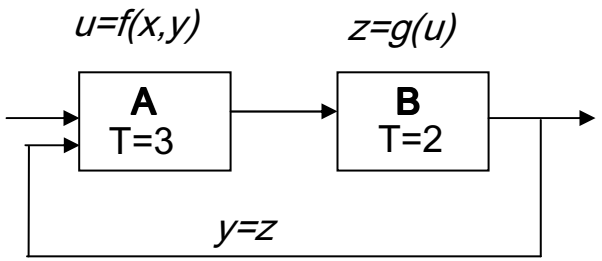


- Algebraic loops (causal loops without delays) result in a fixed point and lack of compositionality

Semantics and Compositionality

- Semantics problem: systems compositions do not behave according to the semantics of the components
  - The problem is typical of SR semantics when there are causal cycles: existence of a fixed point solution cannot be guaranteed (i.e. the system may be ill-defined)
  - When multirate blocks are in a causal loop the composition is always not feasible

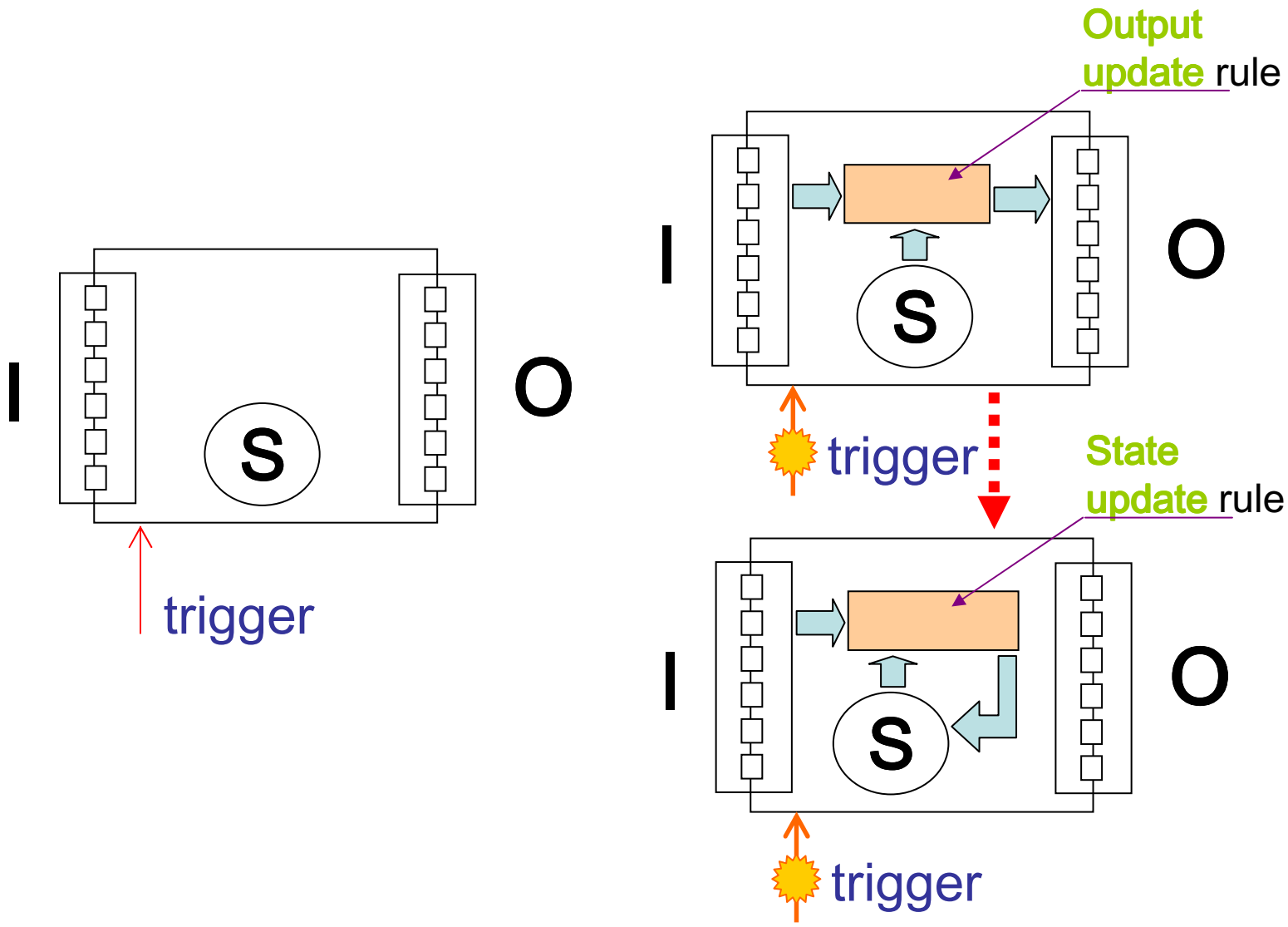




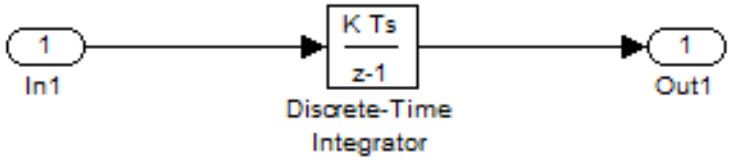
Outline

- Functional vs. Execution model
- Semantics options
- Preserving semantics in refinements
  - Verifying that the synchronous reaction assumption holds with respect to the actual (finite) computation times
  - The behavior of the simulation (of the functional model –i.e. without RT blocks-) must be the same as the run-time behavior
    - Communication behavior must be the same
    - *Outputs are produced before the following event (i.e. The system is not sensitive to whatever happens in between events)*
- Tradeoffs in task implementations
  - Multitask Model implementation by Real-Time Workshop and rate transition (RT) blocks
  - Scheduling trade-offs (schedulability vs. added delays)
- References

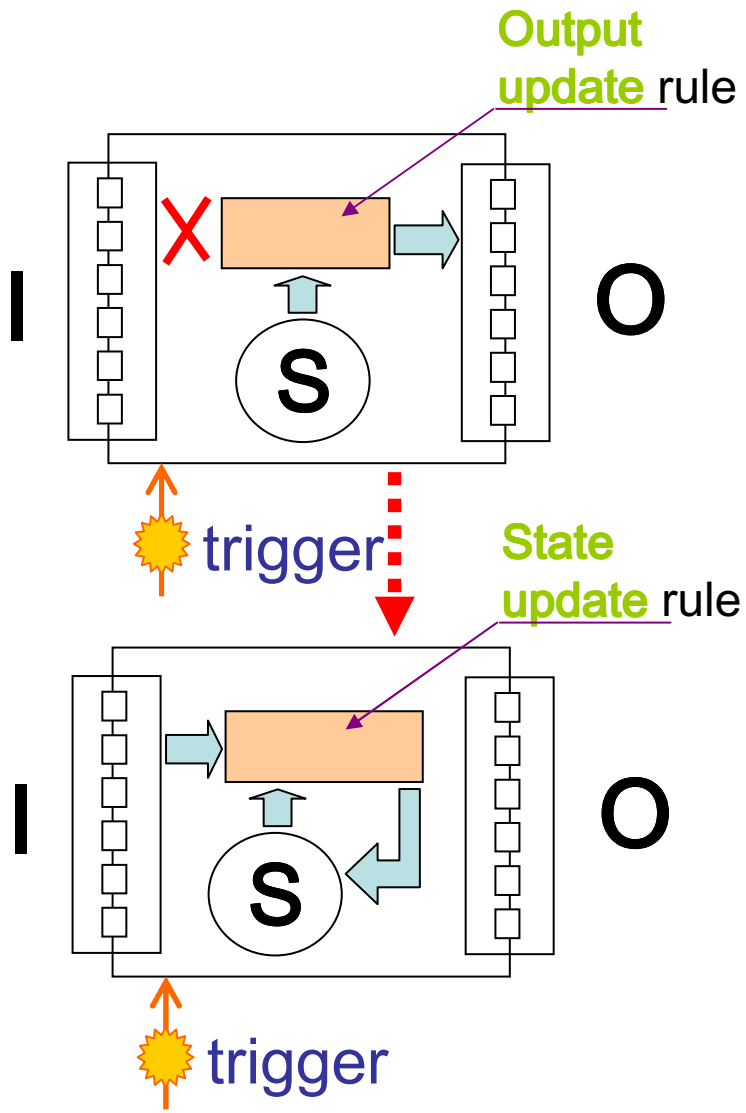
Simulink models (SR)



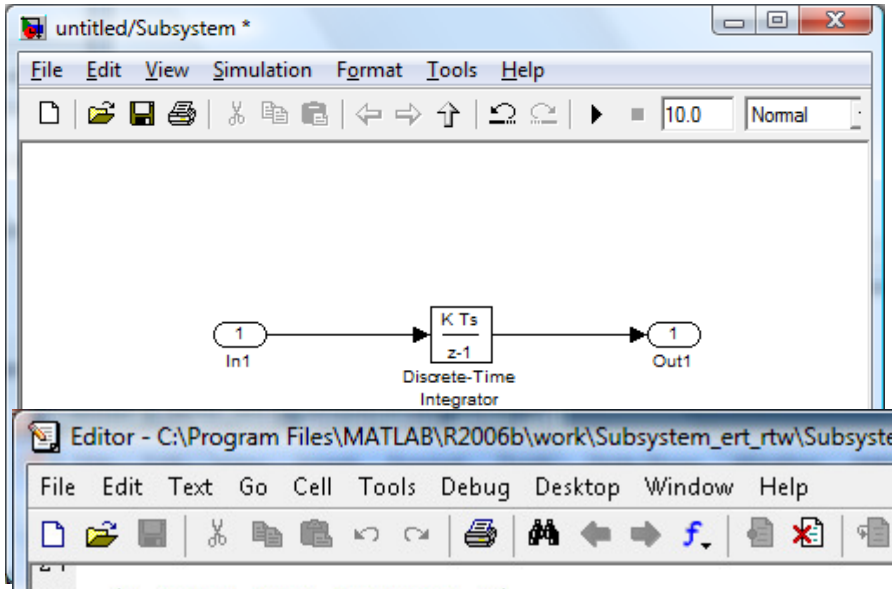
Simulink models (not feedthrough)



Integrator (output does not depend on input but only on state)



Example of generated code

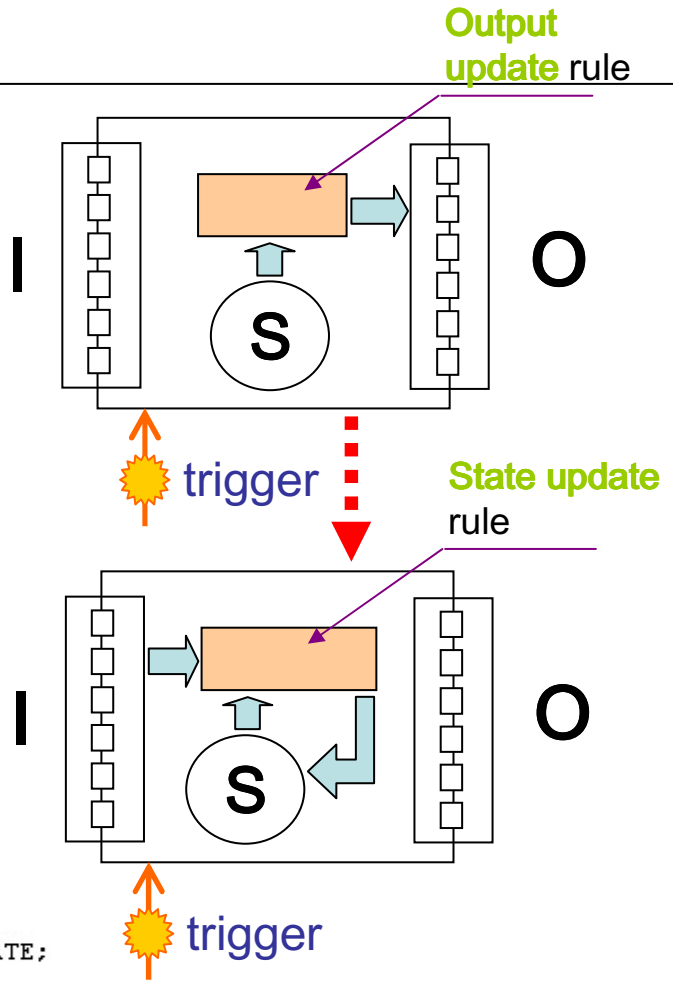


Editor - C:\Program Files\MATLAB\R2006b\work\Subsystem_ert_rtw\Subsystem.c

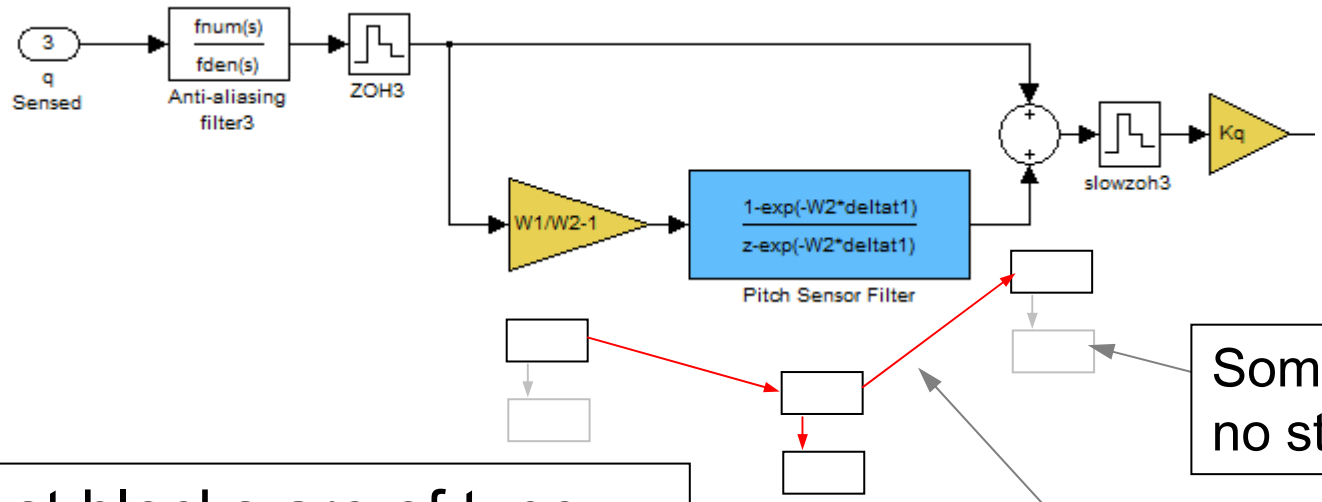
File Edit Text Go Cell Tools Debug Desktop Window Help

```
28  /* Model step function */
29  void Subsystem_step(void)
30  {
31    /* Outport: '<Root>/Out1' incorporates:
32     *  DiscreteIntegrator: '<S1>/Discrete-Time Integrator'
33     */
34    Subsystem_Y.Out1 = Subsystem_DWork.DiscreteTimeIntegrator_DSTATE;
35
36    /* Update for DiscreteIntegrator: '<S1>/Discrete-Time Integrator' */
37    Subsystem_DWork.DiscreteTimeIntegrator_DSTATE =
38      Subsystem_P.DiscreteTimeIntegrator_gai * Subsystem_U.In1 +
39      Subsystem_DWork.DiscreteTimeIntegrator_DSTATE;
40  }
```

C / CPP source or header file Ln 1 Col 1 OVR



Simulink models (feedthrough)



Most blocks are of type feedthrough (output does depend on input)

This implies a precedence constraint in the computation of the block output functions

Some blocks have no state

Dependencies among outputs

Simulation of models

- Simulation of Multirate models
  - order all blocks based upon their topological dependencies
  - The RTW tool (meant for a single processor implementation) generates a total order based on the partial order imposed by the feedthrough semantics
  - *In reality, there are many such total orders that satisfy the dependencies!*
    - *Other choices are possible*
    - *In multiprocessor implementations this can be leveraged to optimize the implementation*
  - Then, for simulation, virtual time is initialized at zero
  - The simulator scans the precedence list in order and execute all the blocks for which the value of the virtual time is an integer multiple of the period of their inputs
  - Simulated execution means computing the block output and then computing the new state

From Models to implementation

- Simulink case

elist

Purpose

List simulation methods in the order in which they are executed during a simulation

Syntax

```
elist m:mid [tid:TID]
elist <gcs | s:sid> [mth] [tid:TID]
elist <gcb | sid:bid> [mth] [tid:TID]
```

Description

`elist m:mid` lists the methods invoked by the system or nonvirtual subsystem method corresponding to the method id `mid` (see the `where` command for information on method IDs), e.g.,

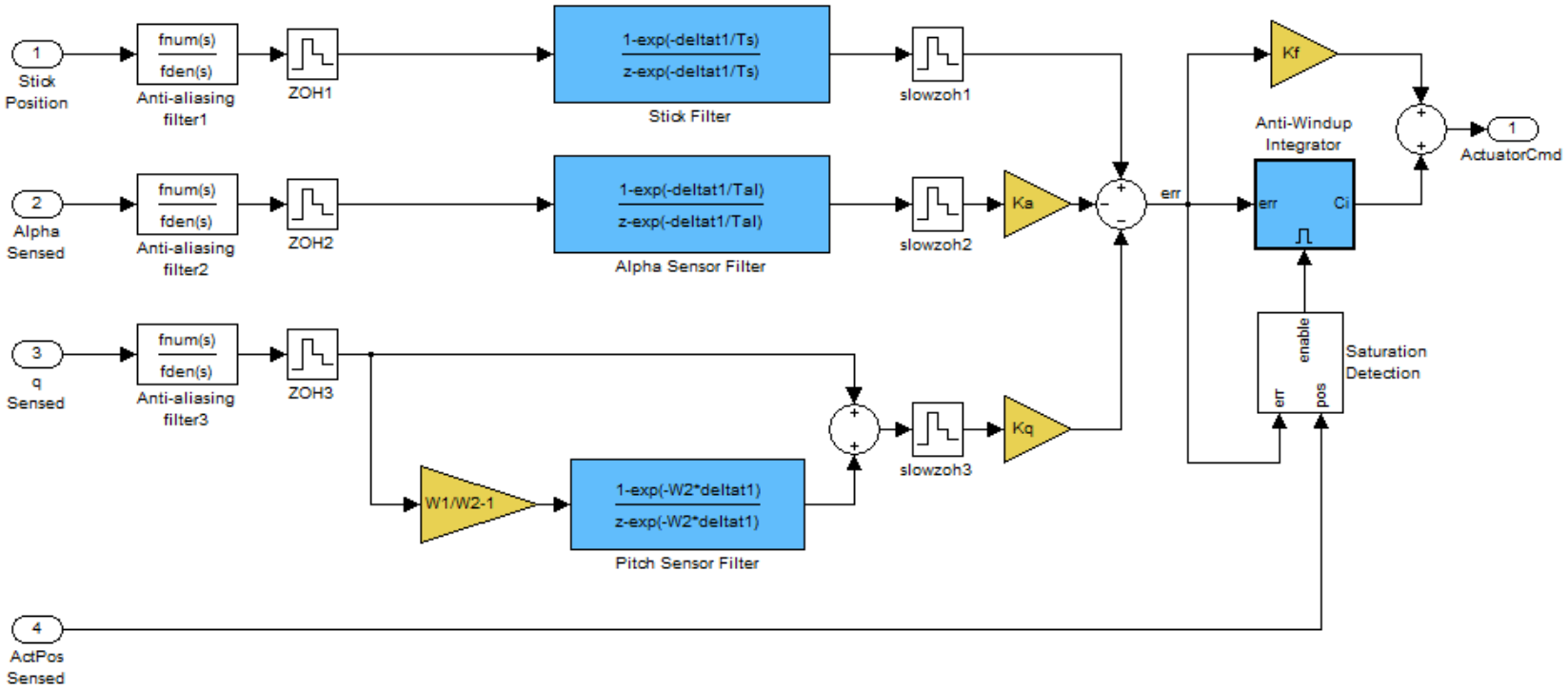
```
sldebug @19): elist n:19

RootSystem.Outputs 'vdp' [tid=0] :
  0:0 Integrator.Outputs 'x1' [tid=0]
  0:1 Outport.Outputs 'Out1' [tid=0]
  0:2 Integrator.Outputs 'x2' [tid=0]
        ...
```

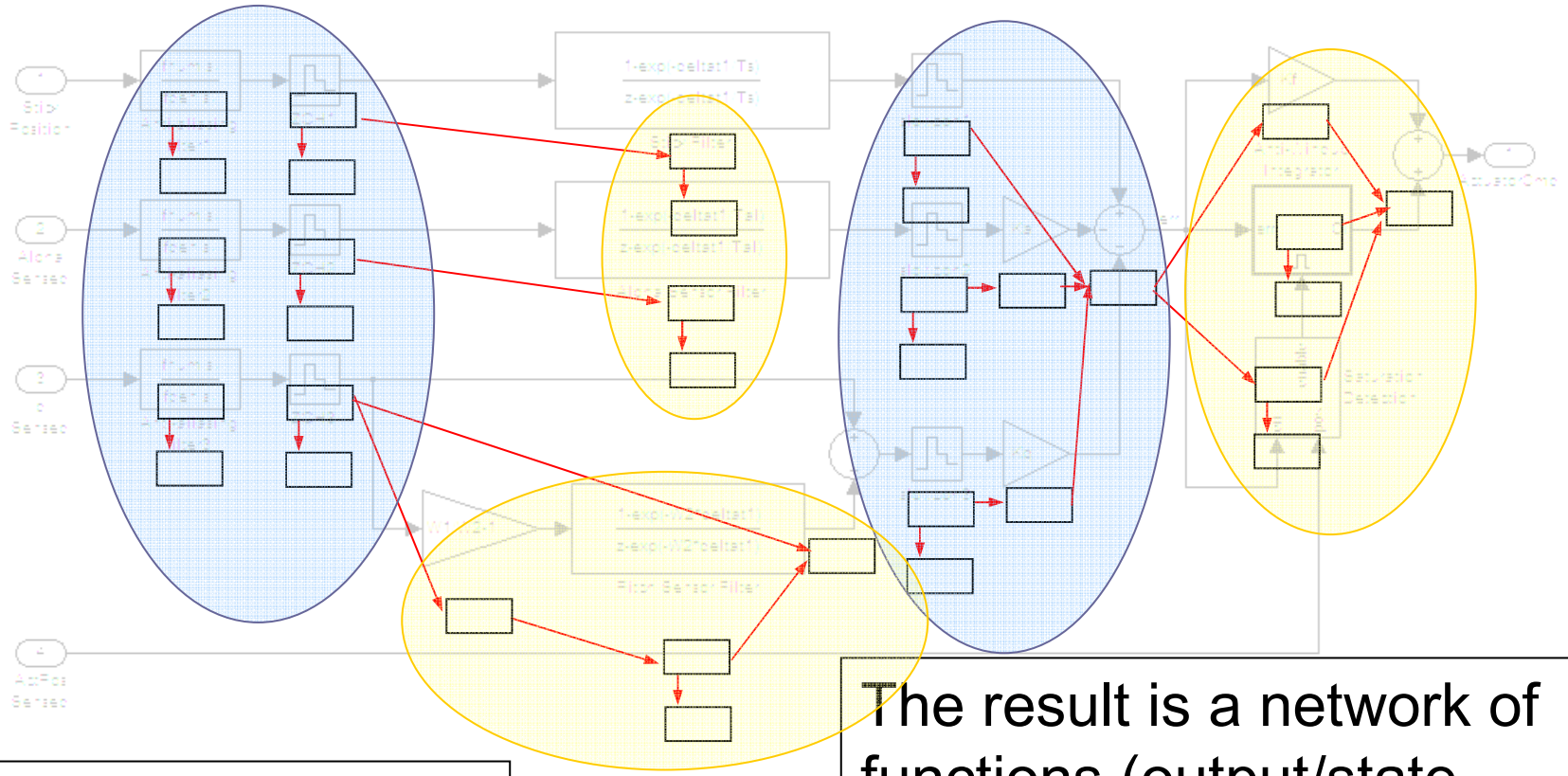
Calling method

Block id | Method | Block | Task id

Simulink models



Simulink models

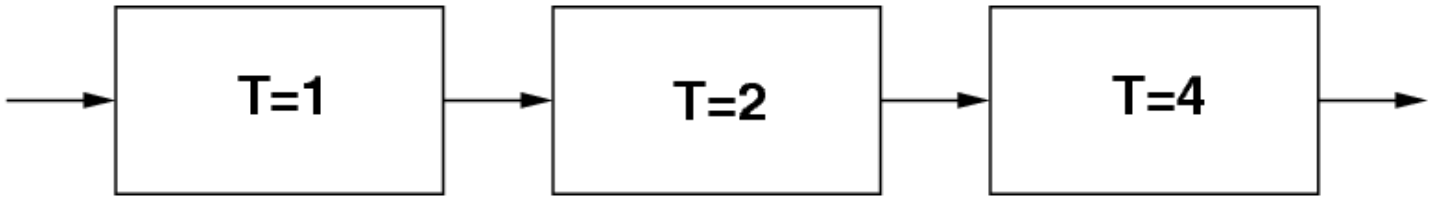


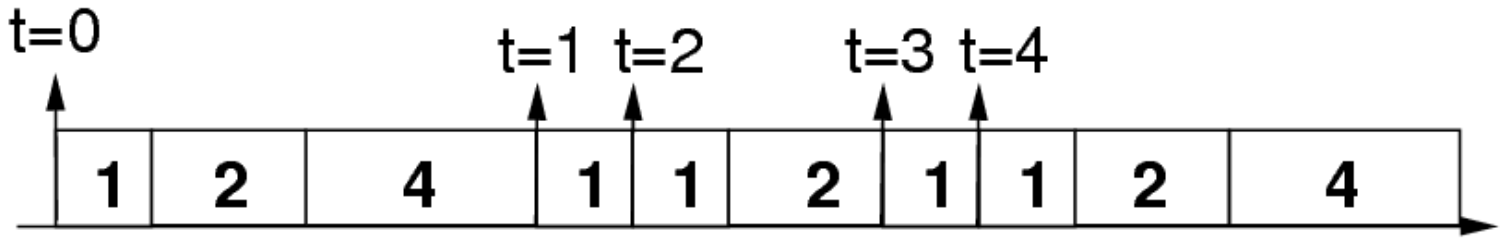
Each blockset is characterized by an execution rate

The result is a network of functions (output/state update) with a set of partial orders

Simulation of mutirate models

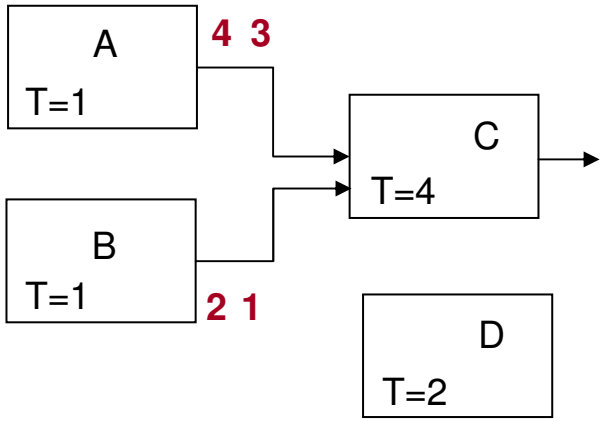
- Simulation of multirate models: an example
  - Simulation runs in virtual time. The virtual clock is updated at each step

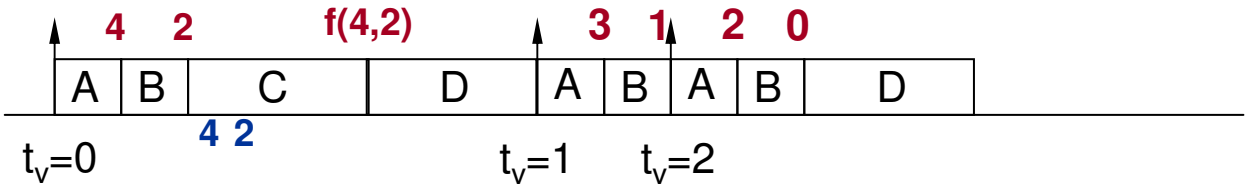




Motivation: Model-based devel. issues

- The implementation of a SR model should preserve its semantics so to retain the validation and verification results. The implementation can use
  - Single task executing at the base rate of the system
  - A set of concurrent tasks, with typically one task for each execution rate, and possibly more.





Simulation: logical execution and communication time

From Models to implementation

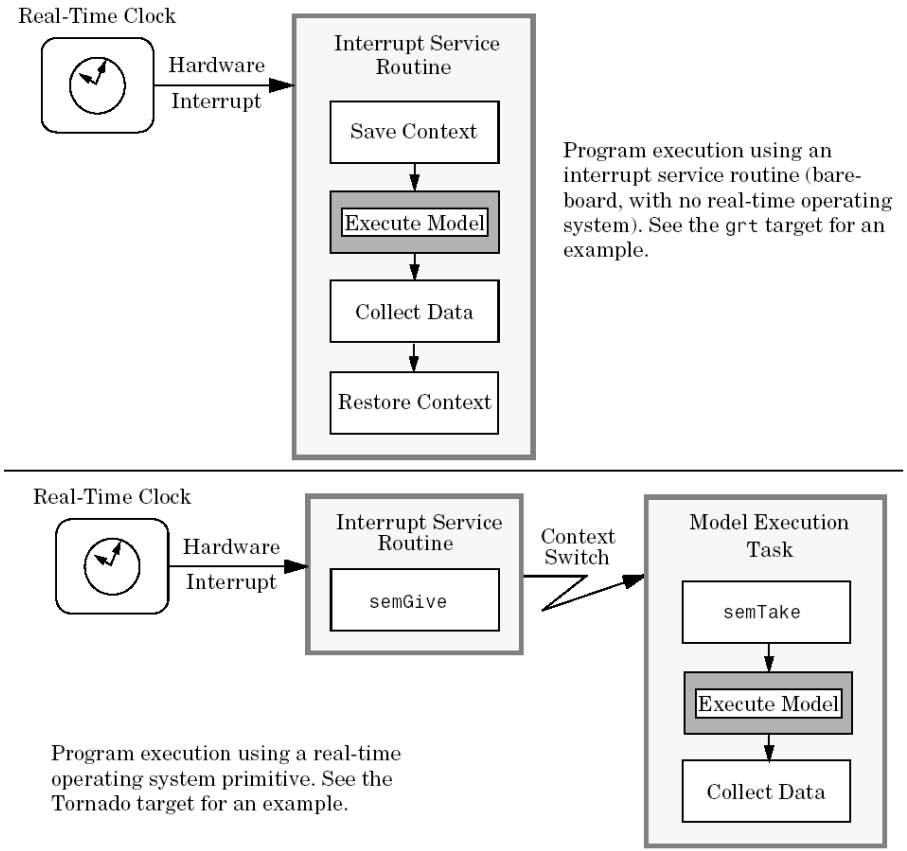
- Simulink case (single task implementation)

Table 2-3: Permitted Solver Modes for Real-Time Workshop Embedded Coder Targeted Models

Mode	Single-Rate	Multi-Rate
`SingleTasking`	Allowed	Allowed
`MultiTasking`	Disallowed	Allowed
`Auto`	Allowed (defaults to `SingleTasking`)	Allowed (defaults to `MultiTasking`)

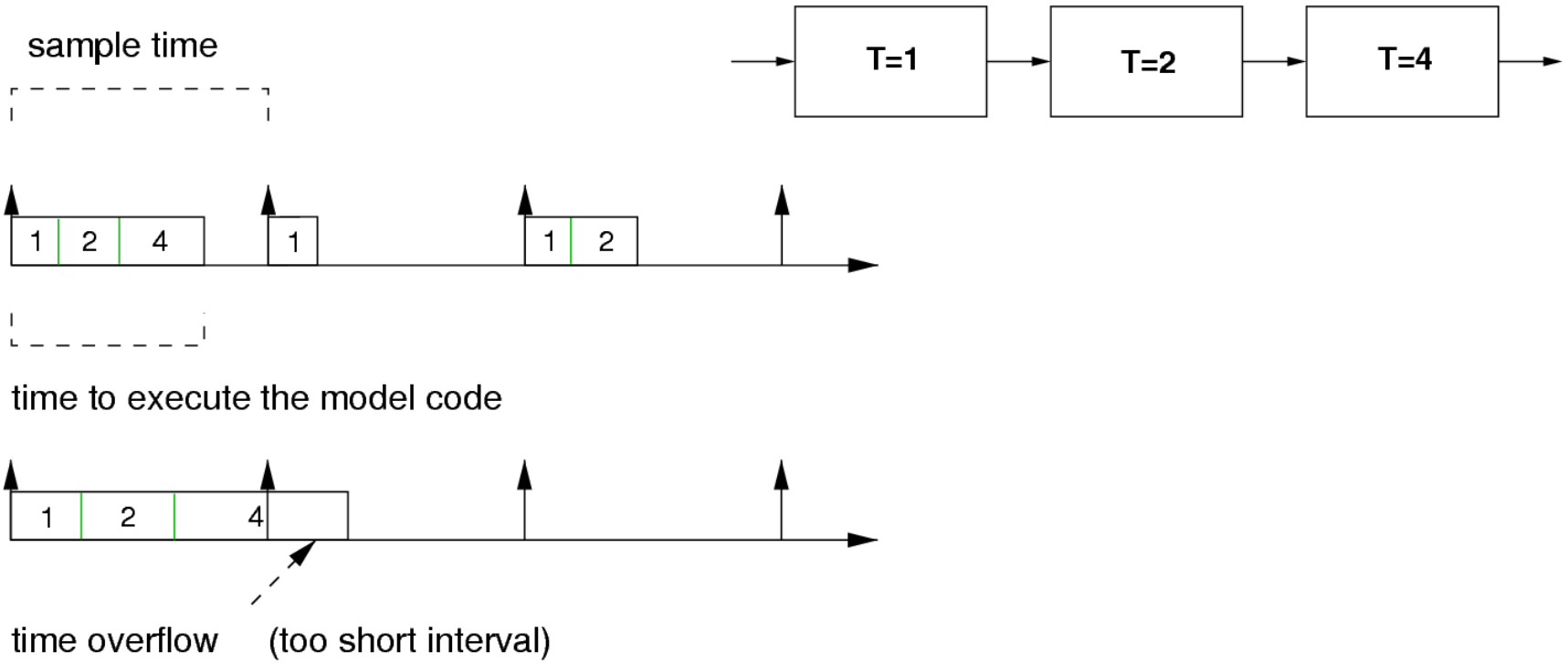
From Models to implementation

- Simulink case (single task implementation)



Implementation of models

- Implementation runs in real-time (code implementing the blocks behavior has finite execution time)
- Generation of code: Singletask implementation



From Models to implementation

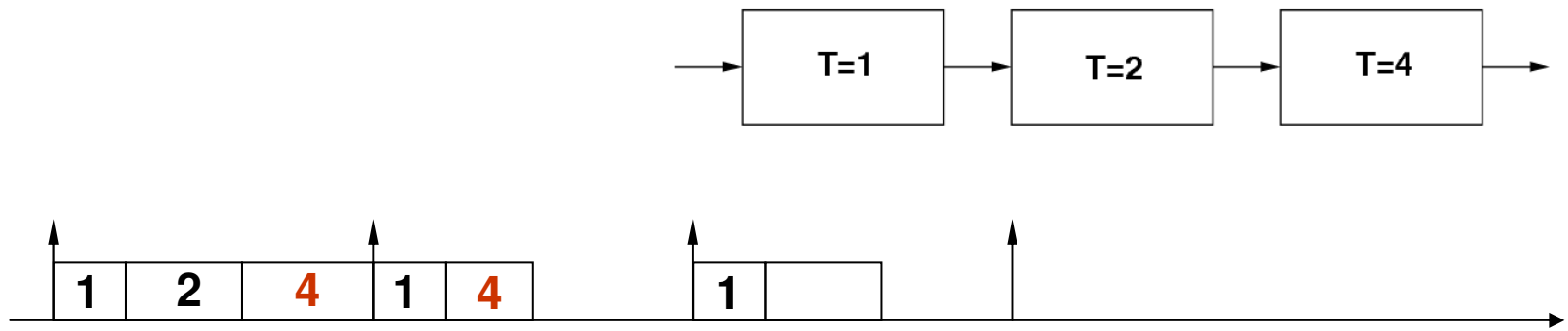
- Simulink case (single task implementation)

```
rt_OneStep()
{
  Check for interrupt overflow or other error
  Enable "rt_OneStep" (timer) interrupt
  ModelStep-- Time step combines output,logging,update
}
```

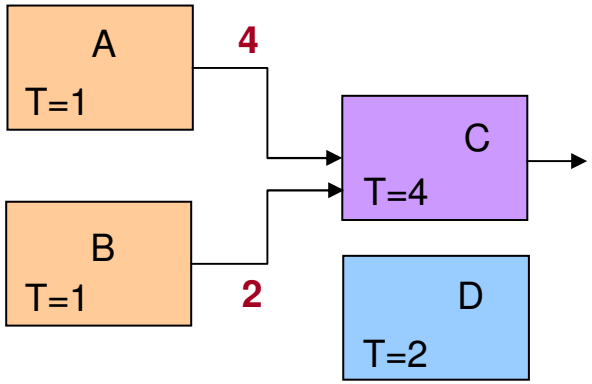
Single-rate `rt_OneStep` is designed to execute *model*_step within a single clock period. To enforce this timing constraint, rt_OneStep maintains and checks a timer overrun flag.

Generation of code: multitask mode

- The RTW code generator assigns each block a task identifier (tid) based on its sample rate.
- The blocks with the fastest sample rates are executed by the task with the highest priority, the next slowest blocks are executed by a task with the next lower priority, and so on (Rate Monotonic)

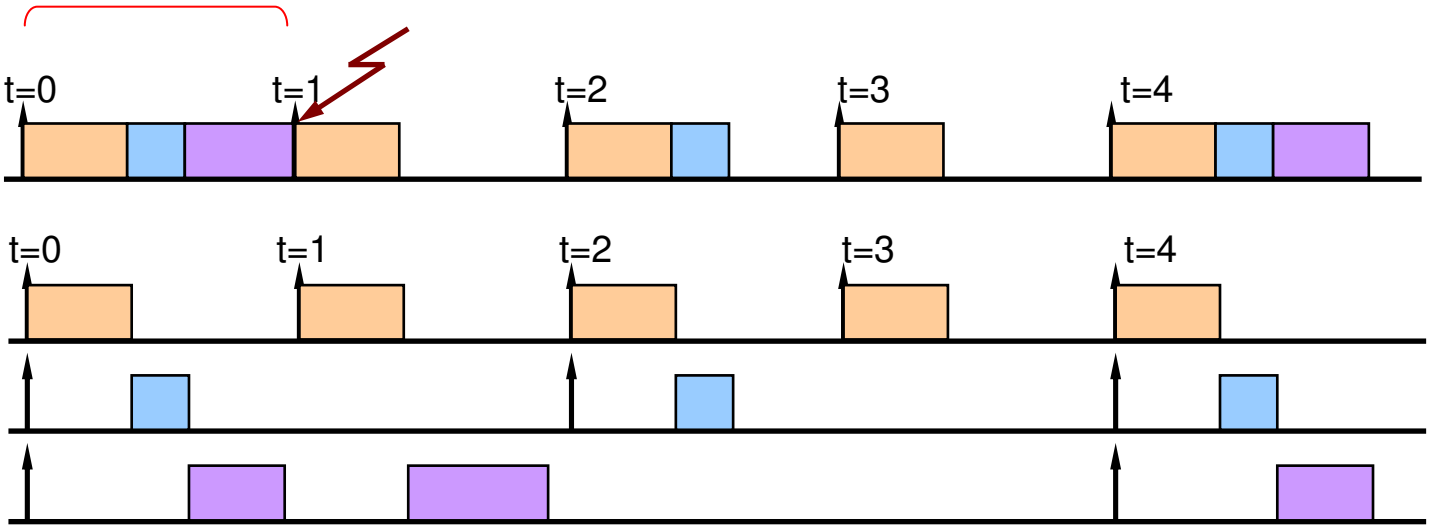


Model implementation: single task

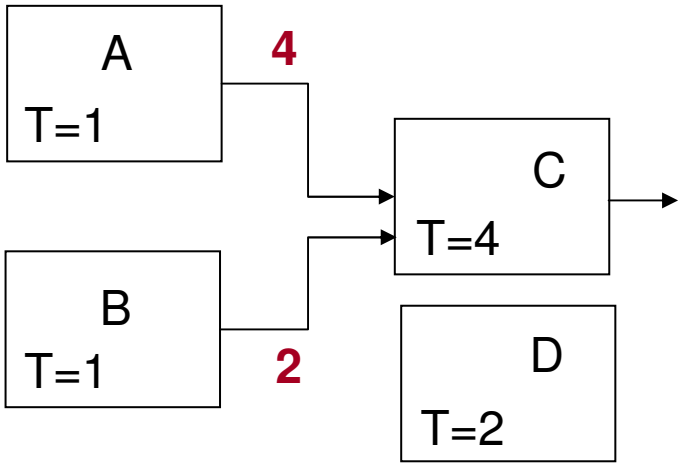


Easy but possibly inefficient

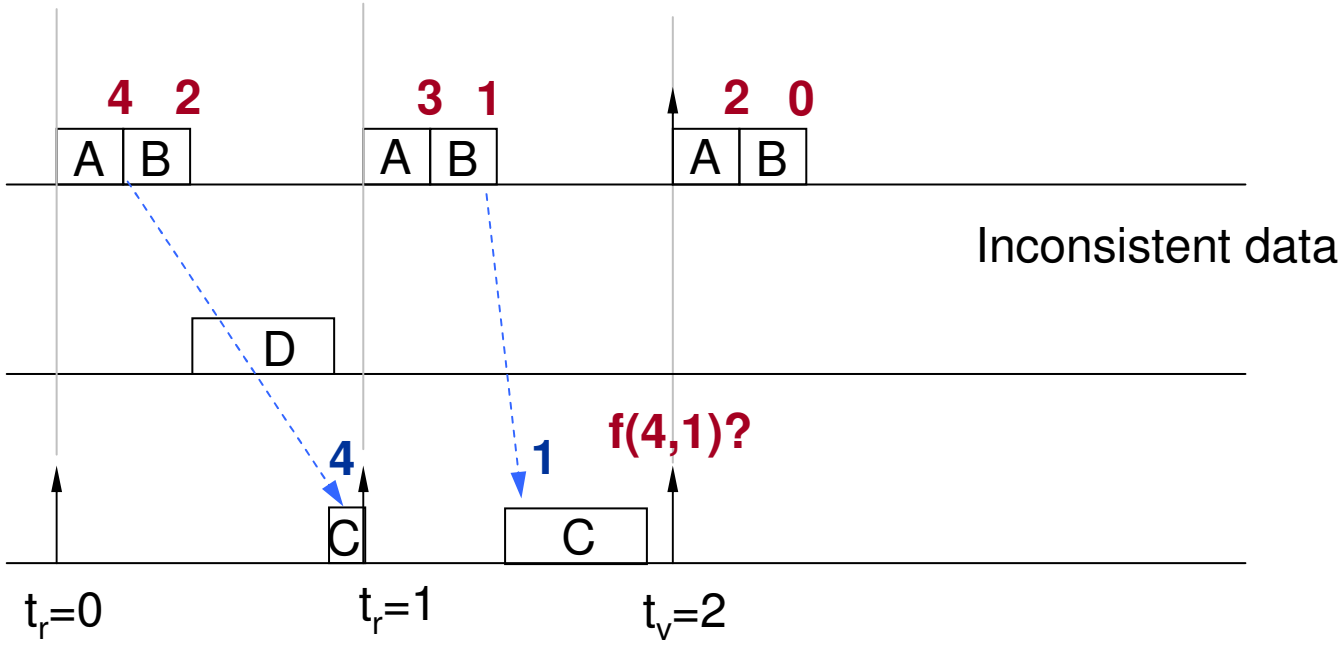
System base cycle =
time to execute the longest system reaction



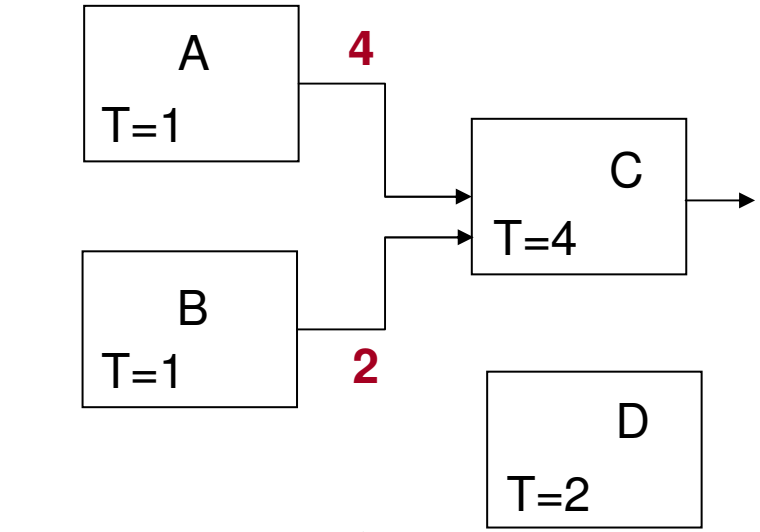
Model implementation: multi-task



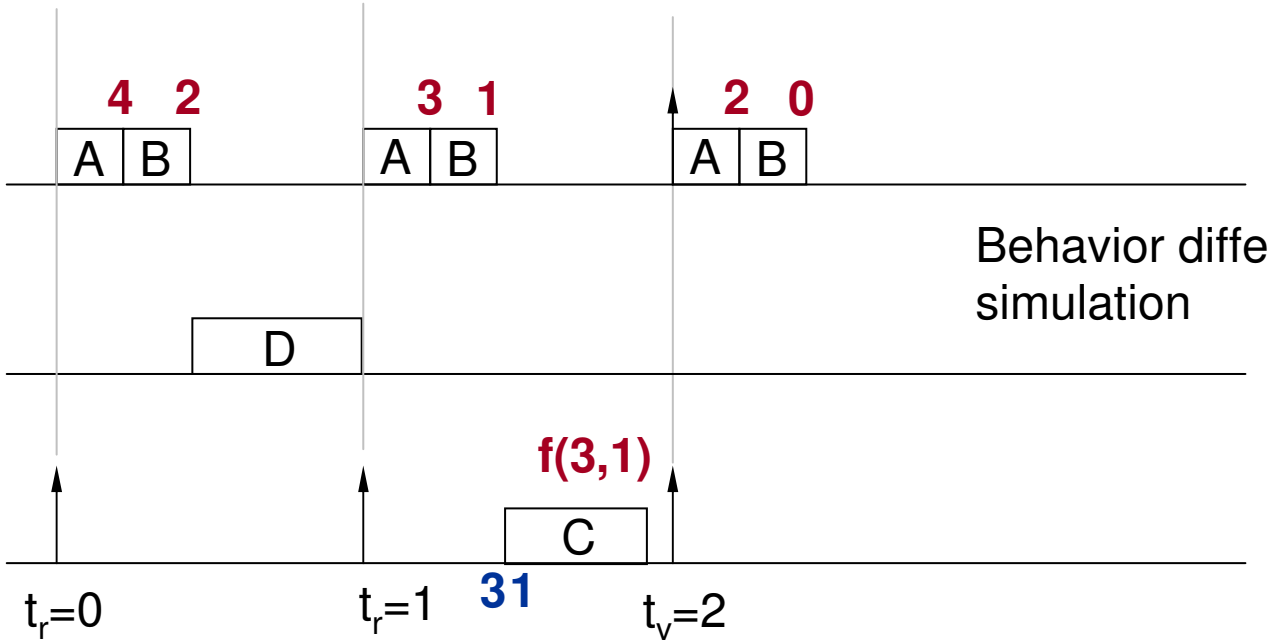
Real-time execution: finite execution time and possible preemption



Model implementation: multi-task



Real-time execution: lack of time determinism (because of preemption)



Behavior different from simulation

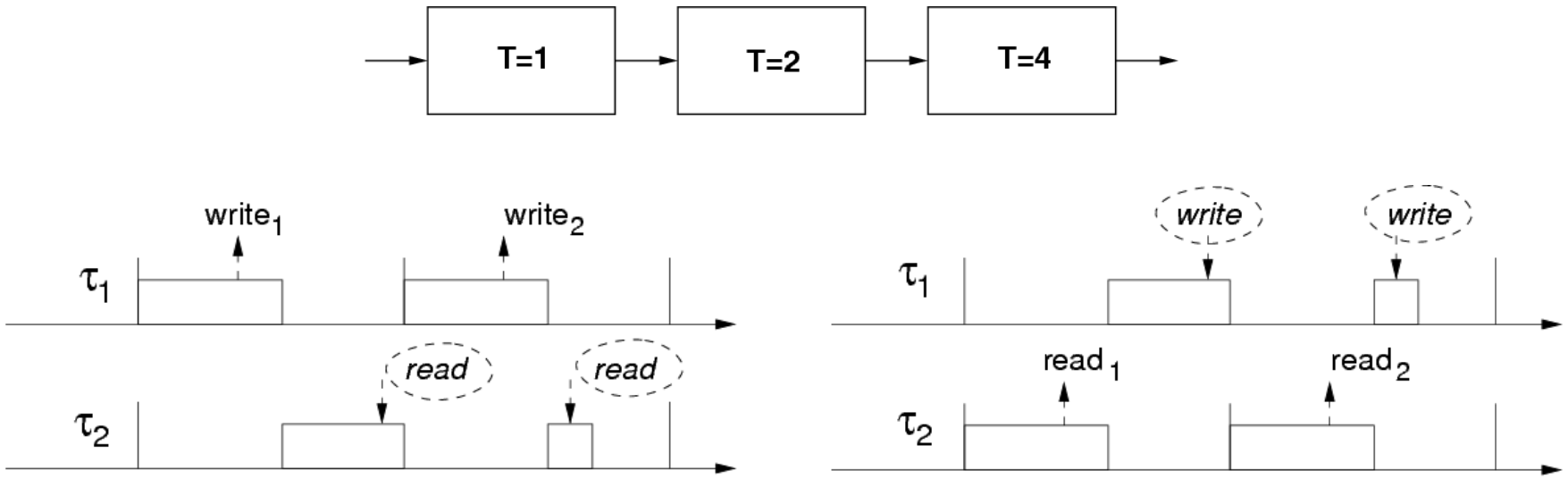
From Models to implementation

- Multitask implementation

```
rt_OneStep()
{
  Check for base-rate interrupt overflow
  Enable "rt_OneStep" interrupt
  Determine which rates need to run this time step
  ModelStep(tid=0) --base-rate time step
  For i=1:NumTasks -- iterate over sub-rate tasks
    Check for sub-rate interrupt overflow
    If (sub-rate task i is scheduled)
      ModelStep(tid=i) --sub-rate time step
    EndIf
  EndFor
}
```

Nondeterminism in time and value

- However, this can lead to the violation of the zero-execution time semantics of the model (without delays) and even to inconsistent state of the communication buffer in the case of
  - low rate (priority) blocks driving high rate (priority) blocks.
  - high rate (priority) blocks driving low rate (priority) blocks.



Adding determinism: RT blocks

- Solution: Rate Transition blocks
  - added buffer space and added latency/delay
  - relax the scheduling problem by allowing to drop the feedthrough precedence constraint
- The mechanism can only be implemented if the rates of the blocks are harmonic (one multiple of the other)
  - Otherwise, it is possible to make a transition to the gcd of the blocks' periods, at the price of additional space and delay

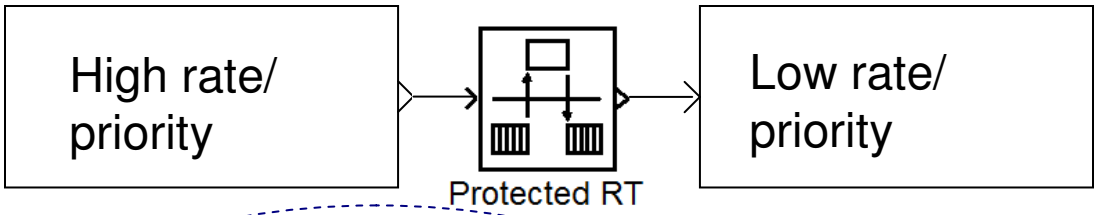
RT blocks: High rate/priority to low rate/priority

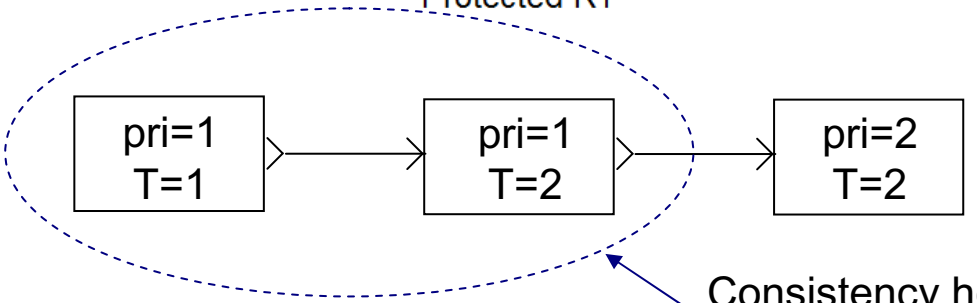
COST

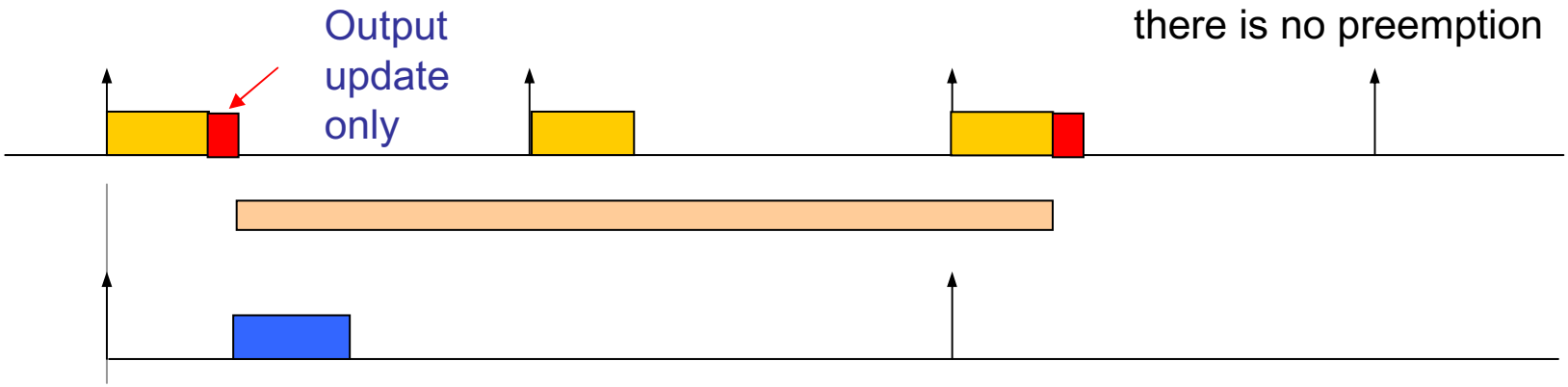
space: 1 additional set of variables for each link

time: overhead of RT implement.

performance: none







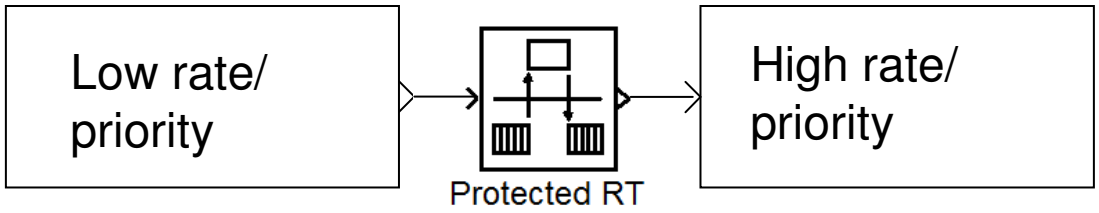
RT blocks: Low rate/priority to high rate/priority

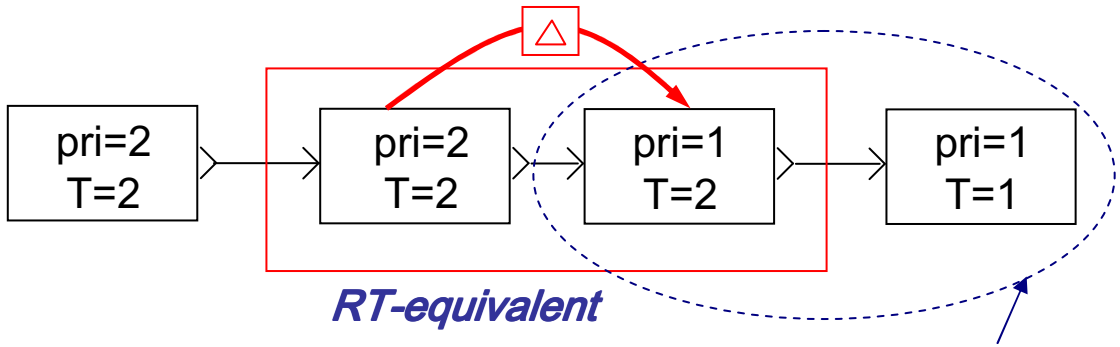
COST

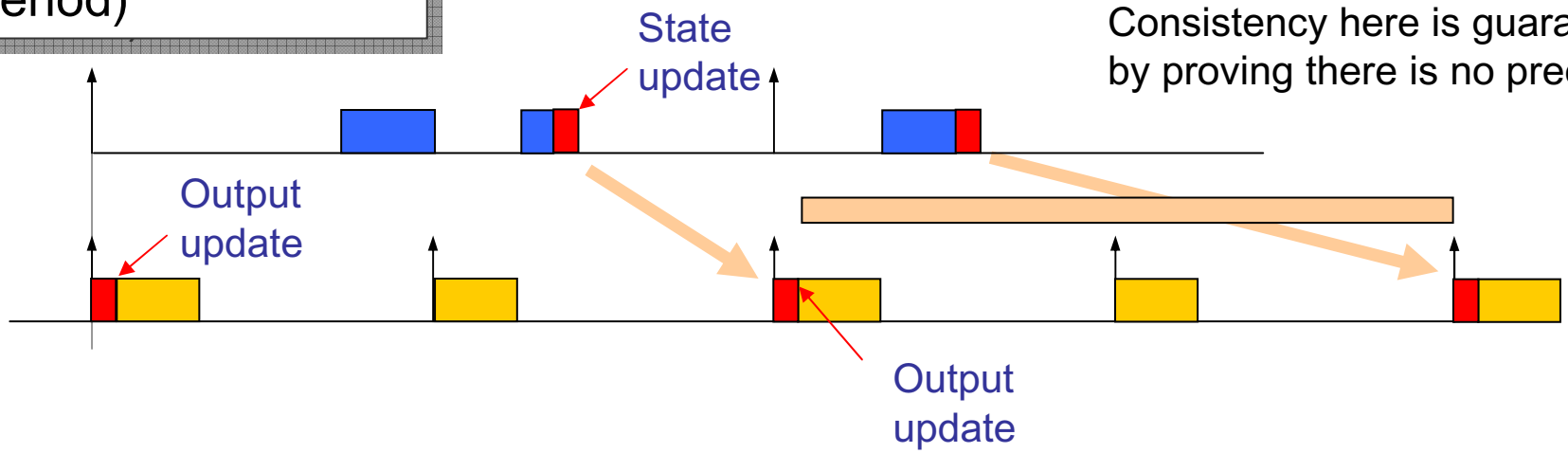
space: 2 additional set of variables for each link

time: overhead of RT implement.

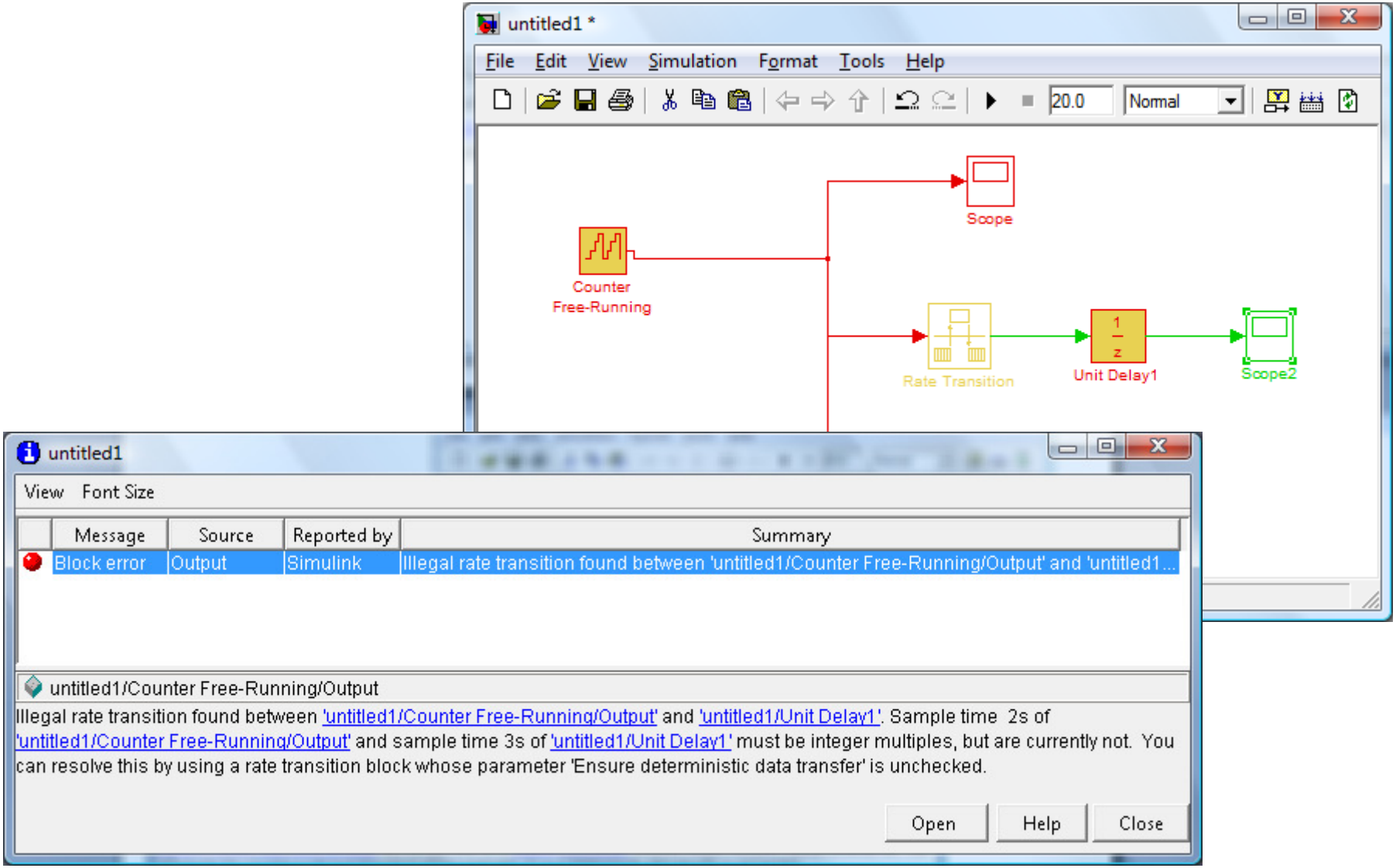
performance: 1-unit delay (low rate period)







Limitations in the use of RT blocks (1)



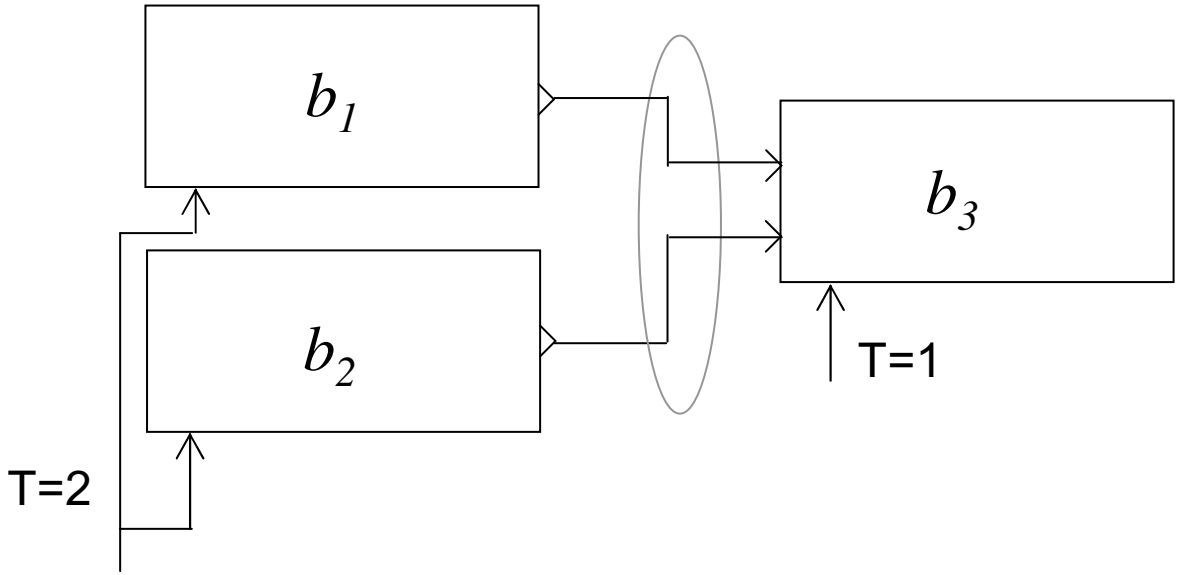
Tradeoffs and design cycles

- RT blocks are **not** a functional entity
  - ***but an implementation device***
- RT Blocks are only required
  - because of the selection of the RM scheduling policy
    in slow to fast transitions
  - because of the possibility of preemption
    in both cases
- In both cases, time determinism (of communication) is obtained at the price of additional memory
- In the case of slow to fast transitions, the RT block also adds a delay equal to the period of the slowest block
  - This is only because of the Rate monotonic scheduling
  - Added delays decrease the performance of controls

Consistency issues

- **Consistency issues in the 1-1 communication between blocks with different rates may happen:**
  - When blocks are executed in concurrent tasks (activated at different rates or by asynchronous events)
  - When a reader may preempt a writer while updating the communication variables (reader with higher priority than writer)
  - When the writer can preempt the reader while it is reading the communication variables (writer with higher priority).
  - *Necessary condition for data inconsistency is the possibility of preemption reader→writer or writer→reader*
- Also, we may want to enforce time determinism (flow preservation)

Consistency issues



- **Also, a relaxed form of time determinism may be required**
  - Input coherency: when inputs are coming from multiple blocks, we want to read inputs produced by instances activated by the same event

Guaranteeing data consistency

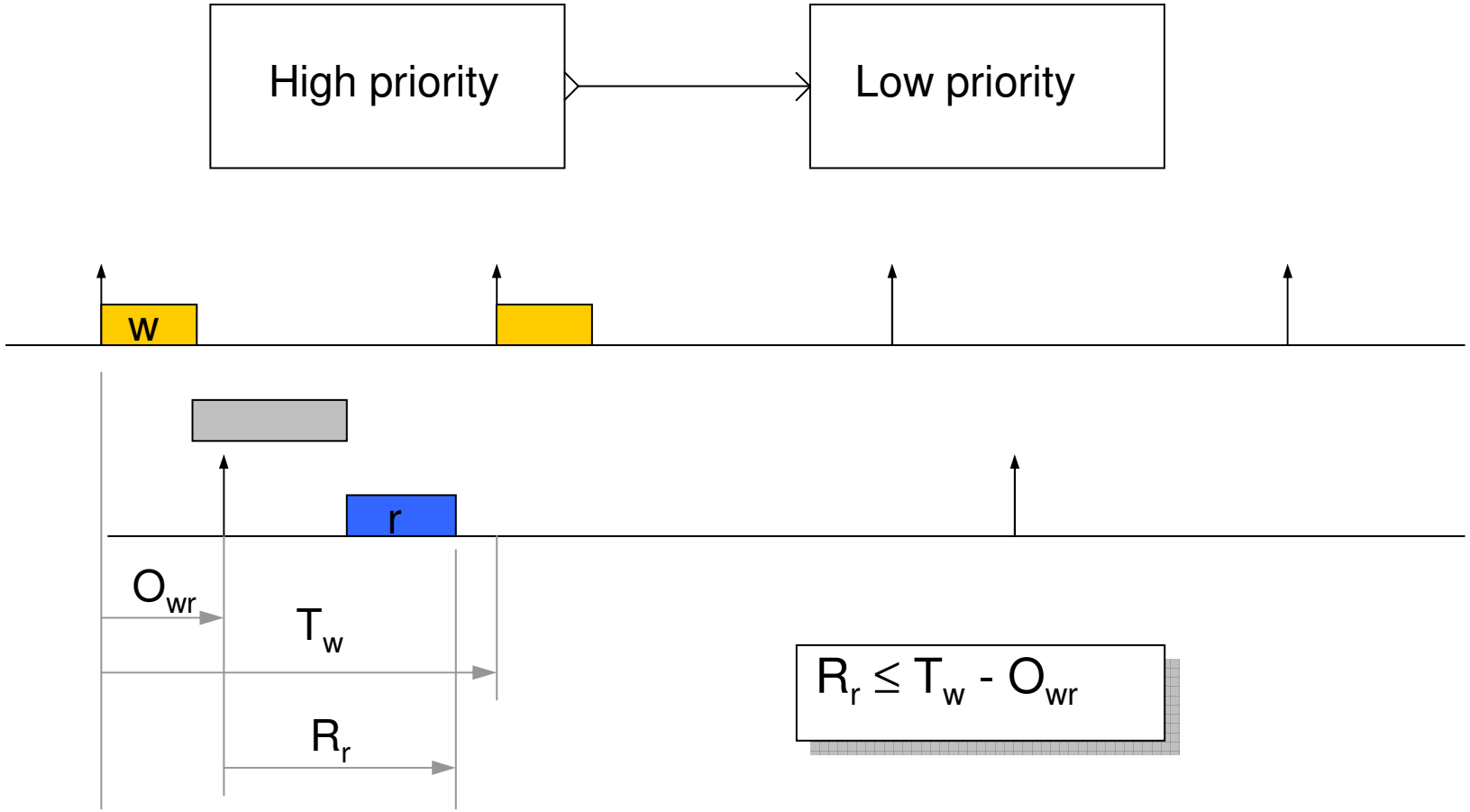
- Demonstrate impossibility of preemption between readers and writers
  - Appropriate scheduling of blocks into tasks, priority assignment, activation offsets and using worst-case response time analysis
- Avoid preemption between readers and writers
  - Disabling preemption among tasks (blocks) (RES_SCHEDULER in OSEK)
- Allow preemption and protect communication variables
  - Protect all the critical sections by
    - Disabling interrupts
    - Using (immediate) priority ceiling (semaphores/OSEK resources)
  - *Problem: need to protect each use of a communication variable. Advantage (does not require extra buffer memory, but only the additional memory of the protection mechanism)*
  - Lock-free/Wait-free communication: multiple buffers with protected copy instructions:
    - Typically w. interrupt disabling or kernel-level code
  - *Problem: requires additional buffer memory (How much?). Advantage: it is possible to cluster the write/read operations at the end/beginning of a task, with limited change to existing code.*
- *The best policy may be a mix of all the previous, depending on the timing contraints of the application and on the communication configuration.*

Demonstrating impossibility of preemption

- Assign priorities and offsets and use timing analysis to guarantee absence of preemption
- Input data:
  - Mapping of functional blocks into tasks
  - Order of functional blocks inside tasks
  - Worst-case execution time of blocks (tasks)
  - Priorities assigned to tasks
  - Task periods
  - (relative) Offset in the activation of periodic tasks (o_{wr} = minimum offset between writer and reader activations, O_{wr} maximum offset between the activations)
- Computed data
  - Worst case response time of tasks/blocks (considering interferences and preemptions) R_r for the writer R_w for the reader
- Two cases:
  - Priority writer > priority reader
  - Priority reader > priority writer

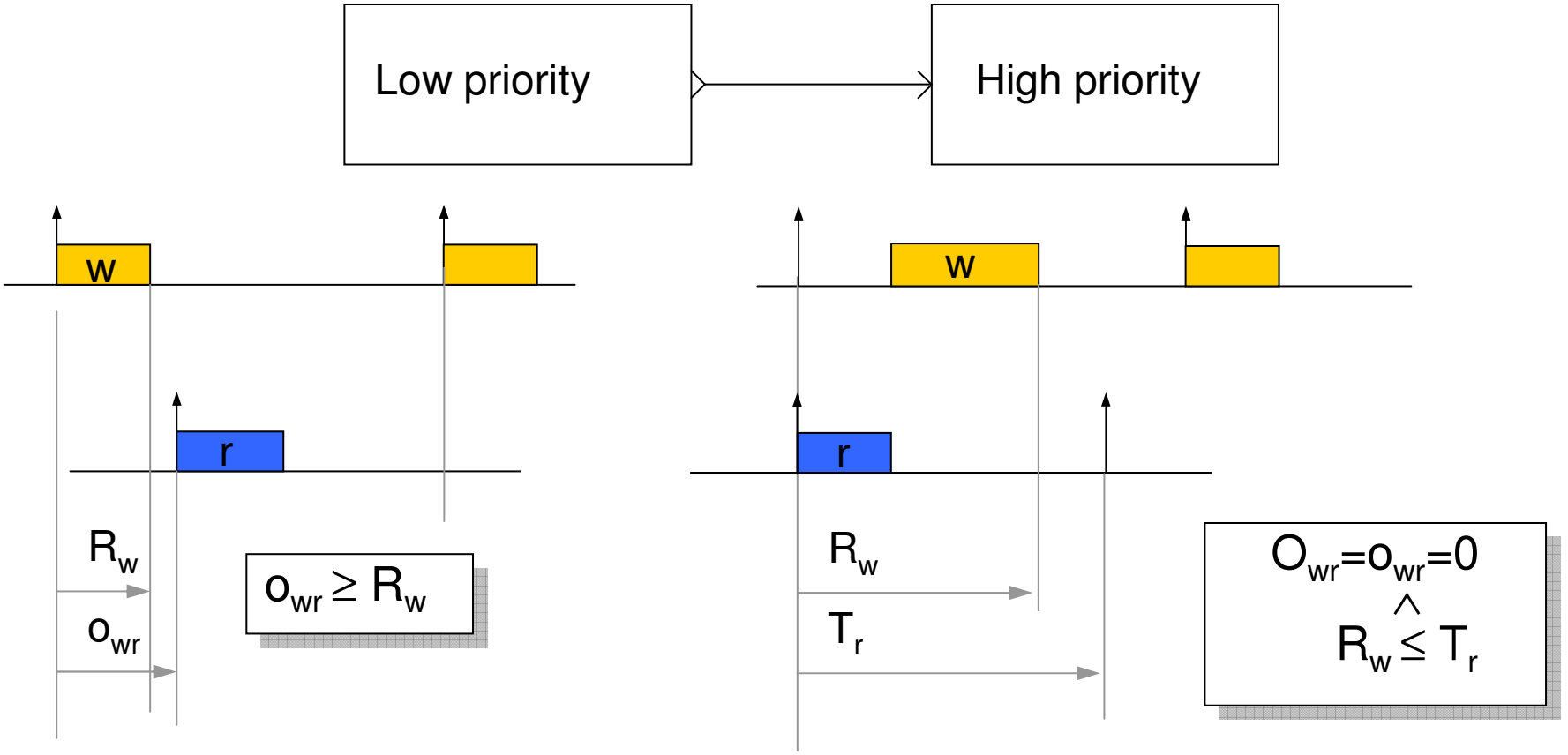
Absence of preemption/High to low priority

- Condition for avoiding preemption writer→reader (no assumptions about relative rates of reader/writer)



Absence of preemption/Low to high priority

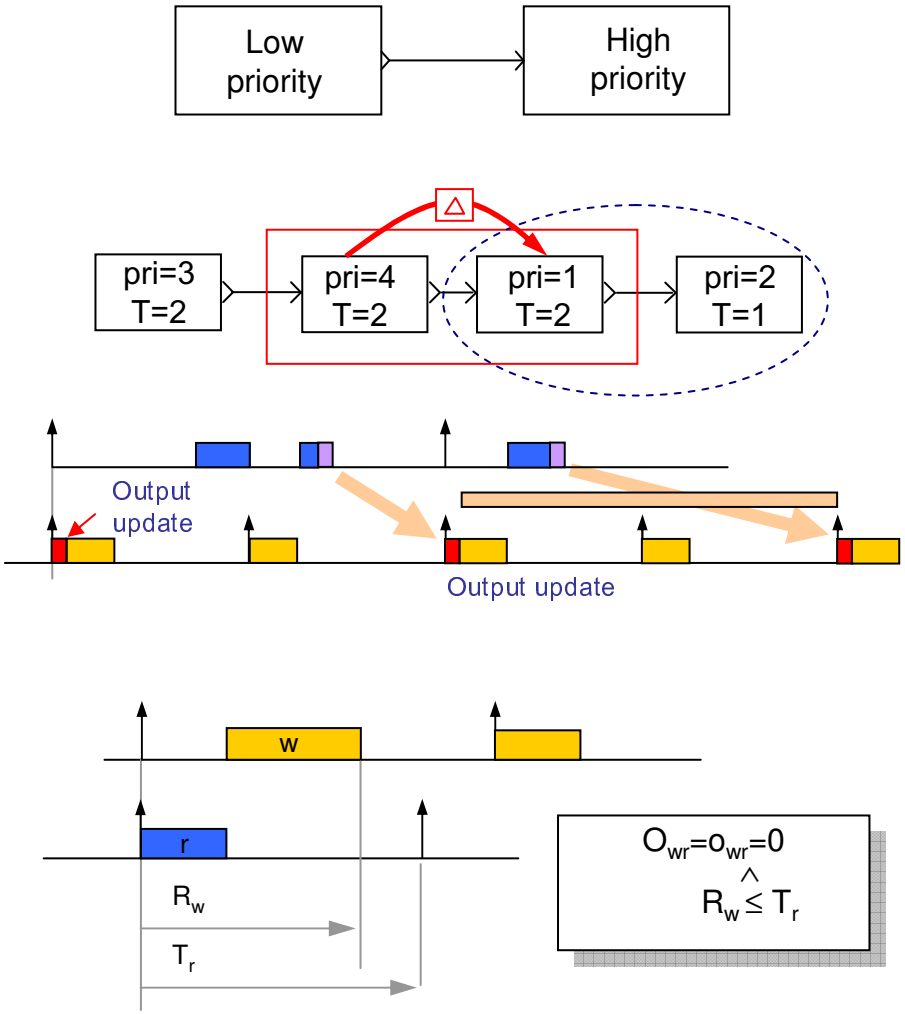
- Condition guaranteeing absence of preemption or reader to writer (reader→writer)



Both conditions are unlikely in practice

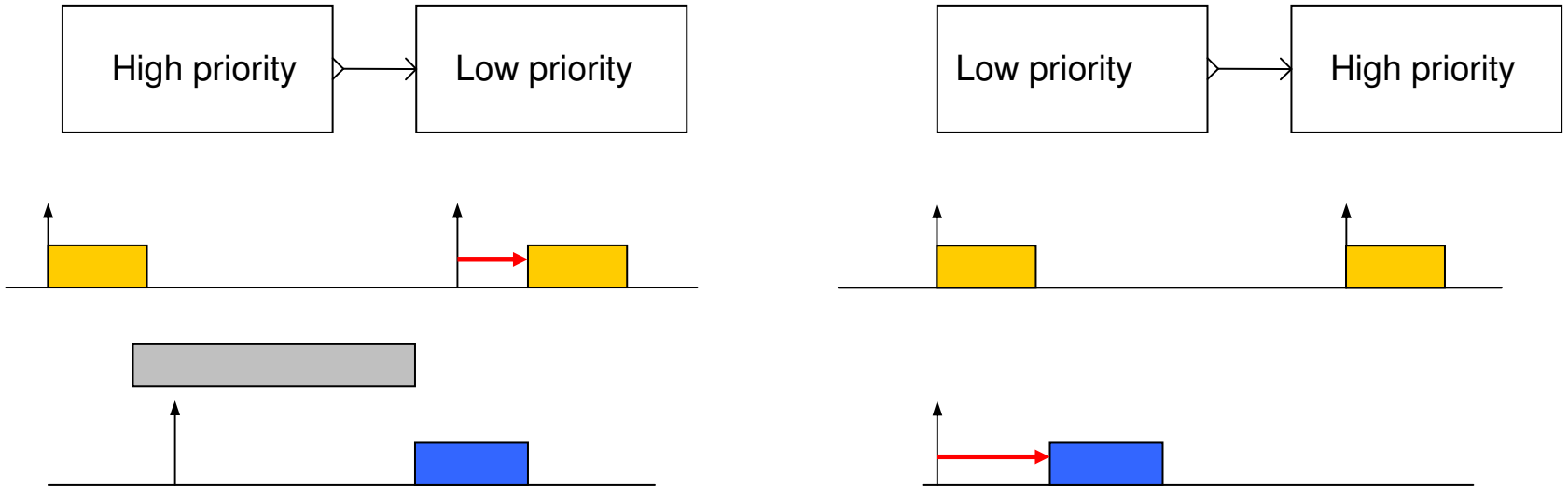
Absence of preemption/Low to high priority

- These conditions are ultimately used by the Rate Transition block mechanisms !!



Avoiding preemption

- Disabling preemption

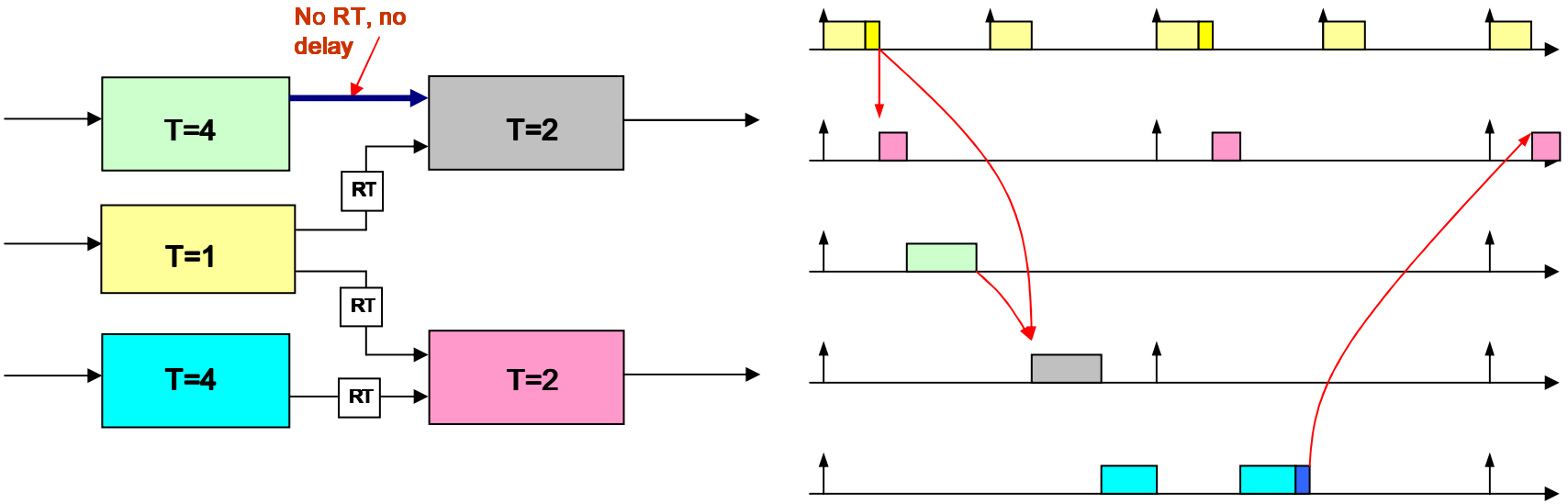


The response time of the high priority block/task is affected, need to check real-time properties

Design/Scheduling trade-offs

However ...

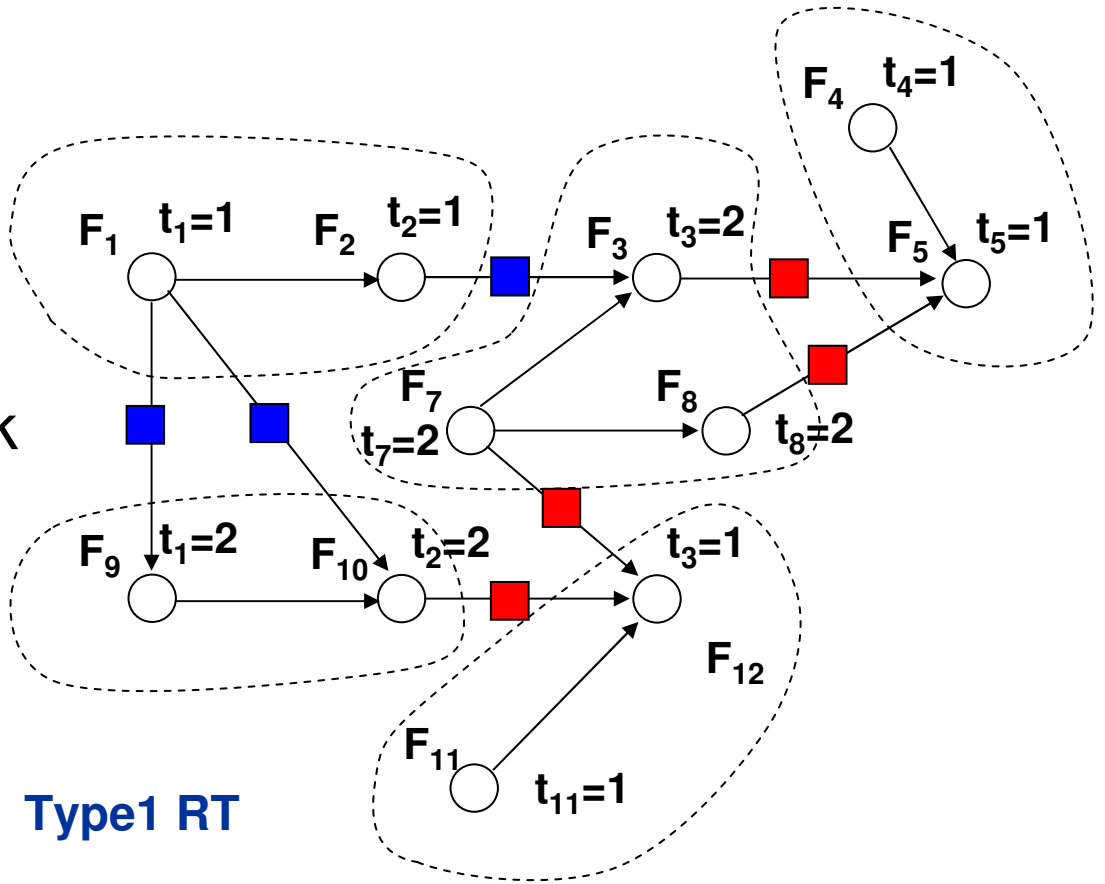
- if the communication is fast-to-slow and the slow block completes before the next instance of the fast writer, the RT block is not required
- if the communication is from slow to fast, it is possible to selectively preserve the precedence order (giving higher priority to the slow block) at the expense of schedulability
  - Two tasks at the same rate, one high priority, the other low priority



An approach

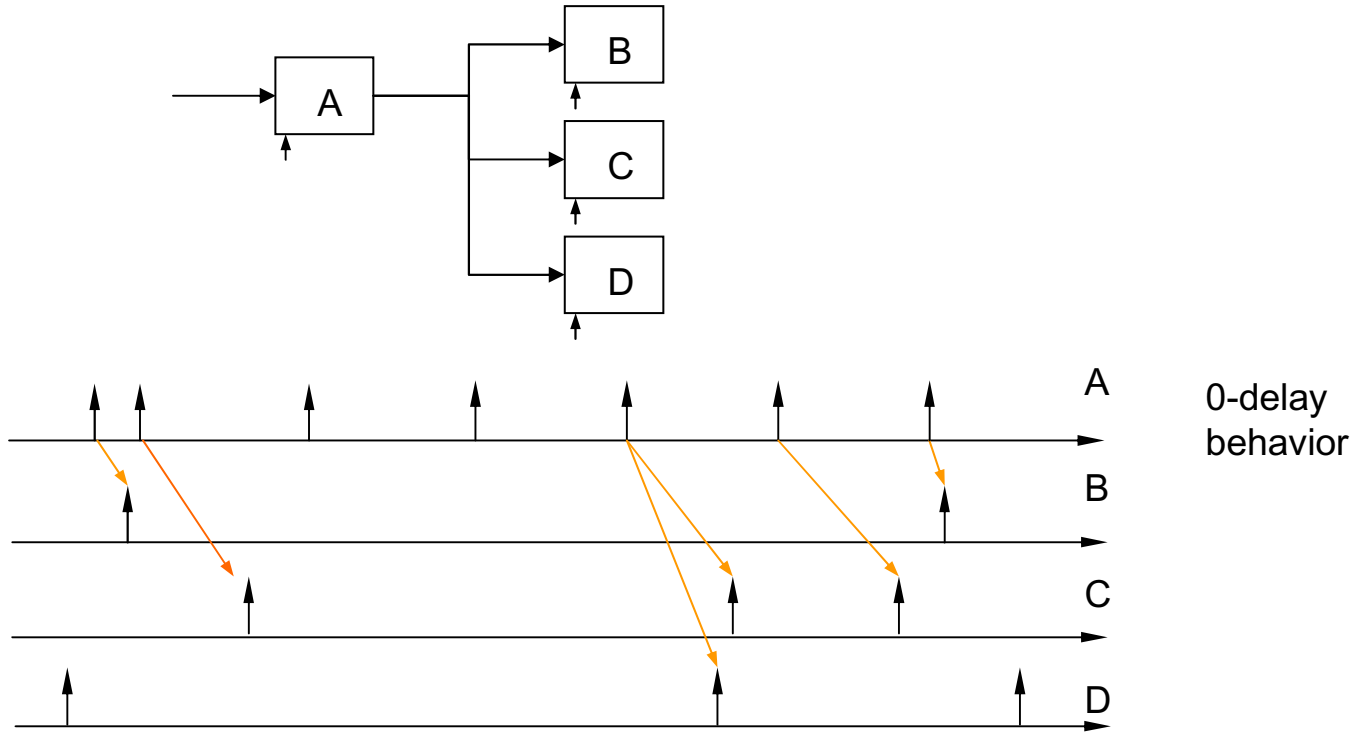
Required steps

- Definition of the network of functional blocks with feedthrough dependencies
- Definition of the synchronous sets
- Priority assignment and mapping into tasks
- Definition of the block order inside tasks



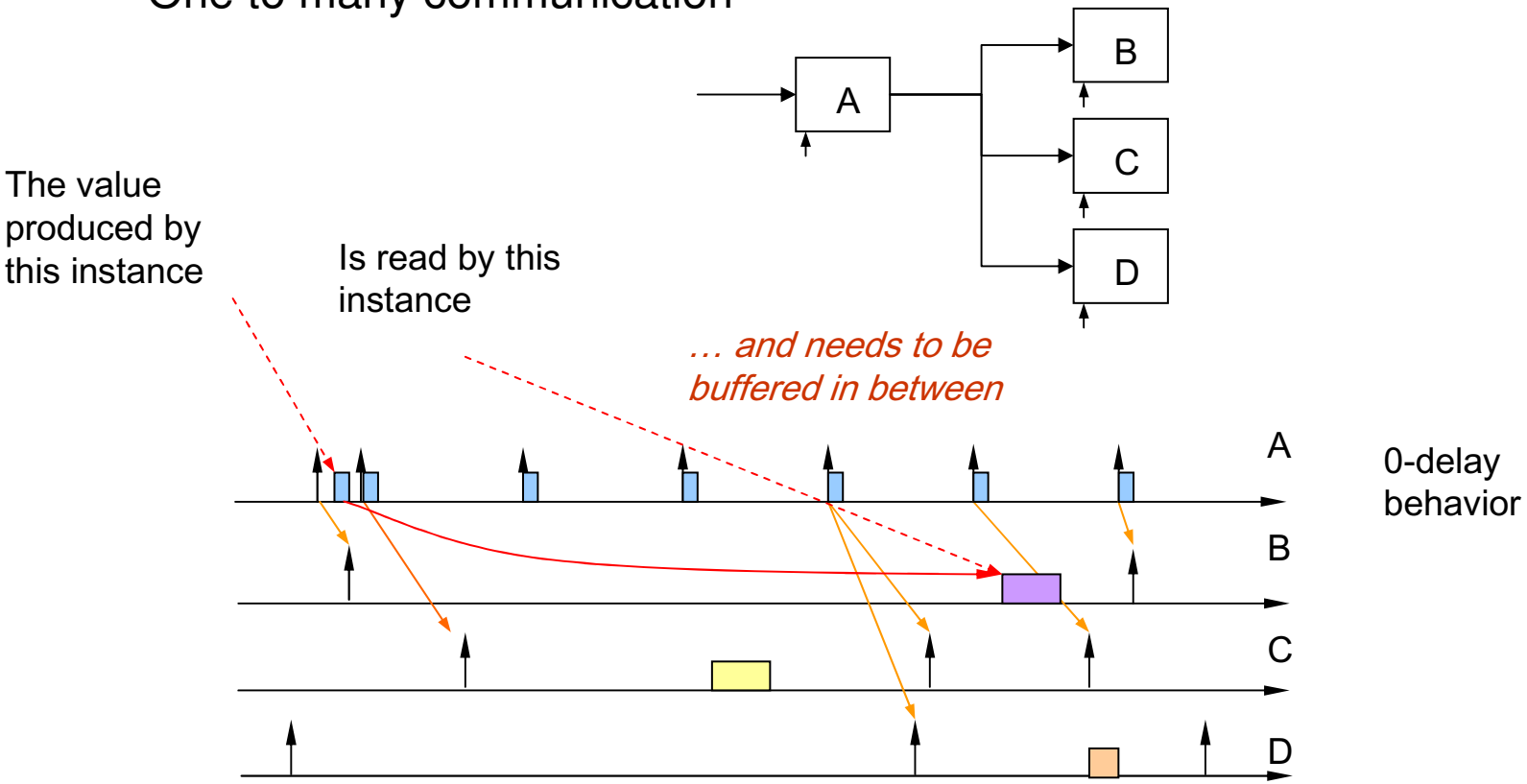
Preserving streams

- What buffering mechanisms are needed for the general case ?
  - Event-driven activation
  - One-to-many communication

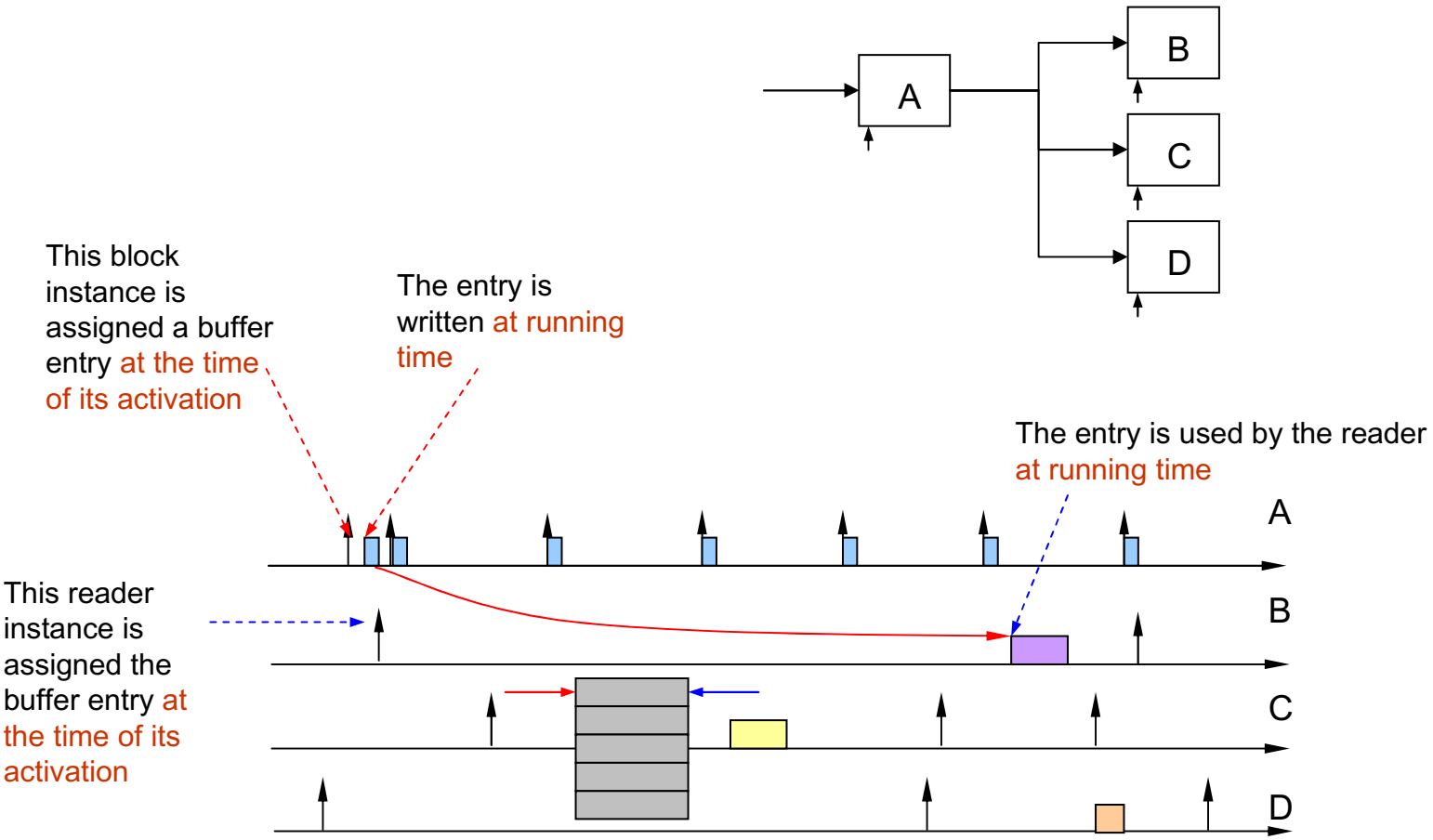


Preserving streams

- What buffering mechanisms are needed for the general case ?
  - Stream preservation (requirement)
  - Event-driven activation
  - One to many communication

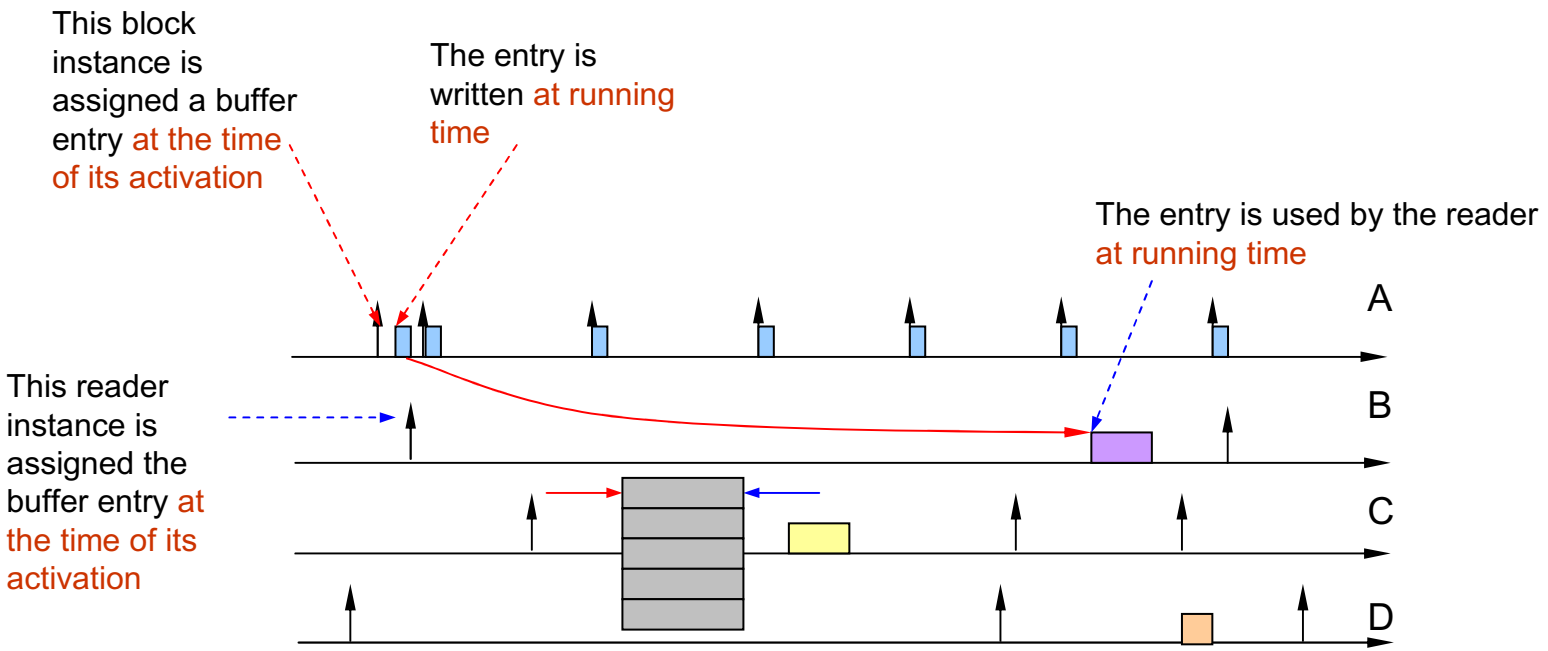


Preserving streams



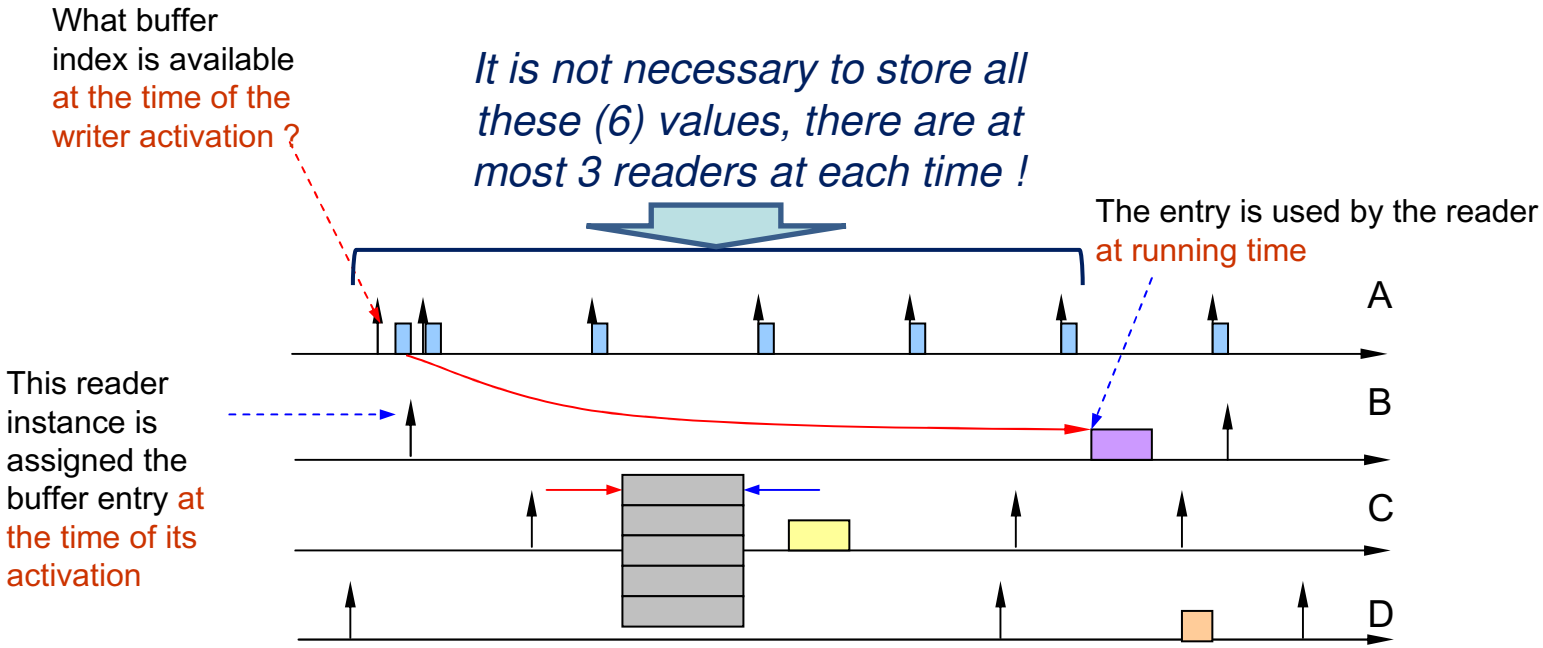
Preserving streams

- The time the buffer index is assigned (activation of the block) may differ significantly from the time when the index is actually used (at running time) because of scheduling delays
  - Support from the OS is needed for assigning indexes at block activation times



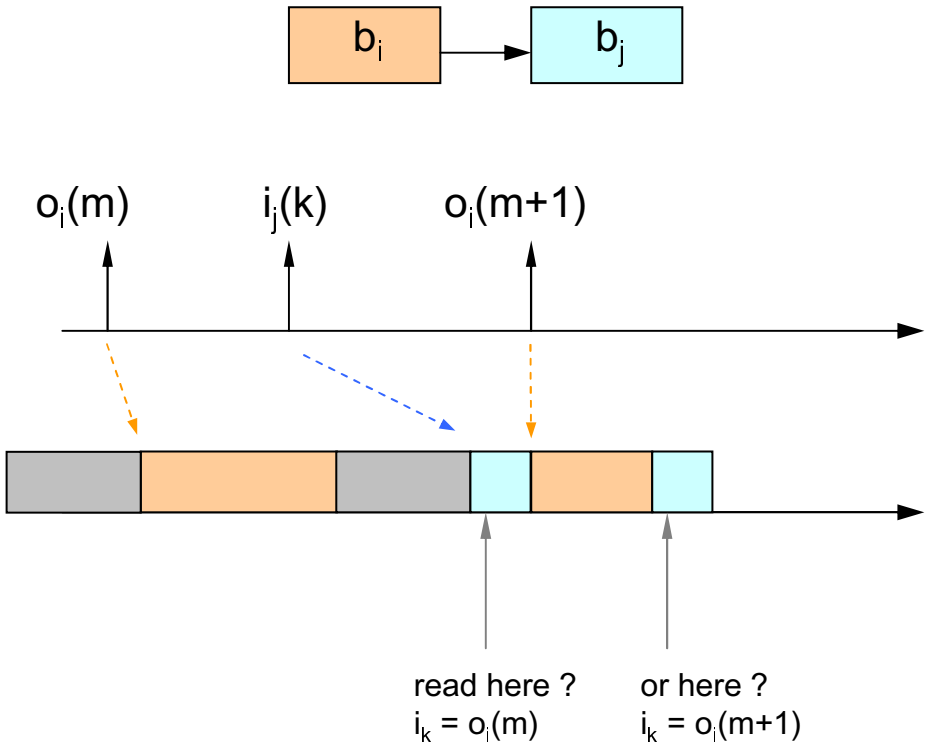
Preserving streams

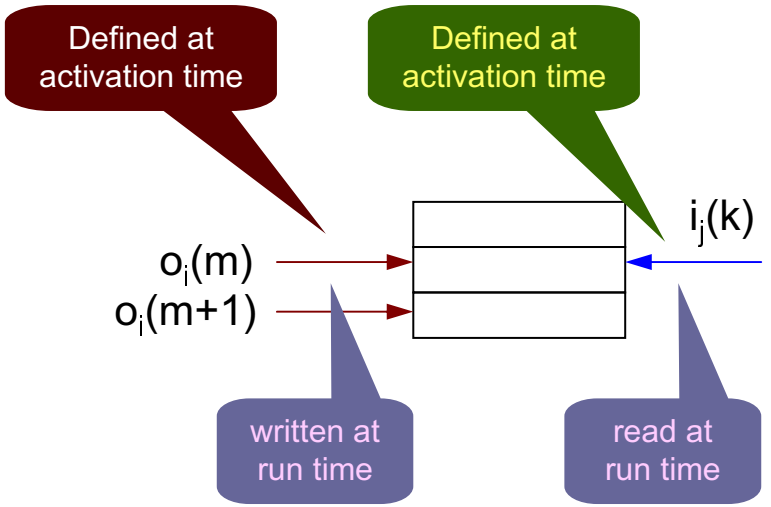
- Many issues
  - Defining efficient mechanisms for assigning indexes to the writers and the readers (if they are executed at kernel level)
  - Sizing the communication buffers (given the system characteristics, how many buffers are needed?)



Model implementation: multi-task

- Efficient but issues with data integrity and time determinism





Q1: How many buffers you need?

Q2: How do you define the index to be used (at activation time) and you pass to the runtime instance ?

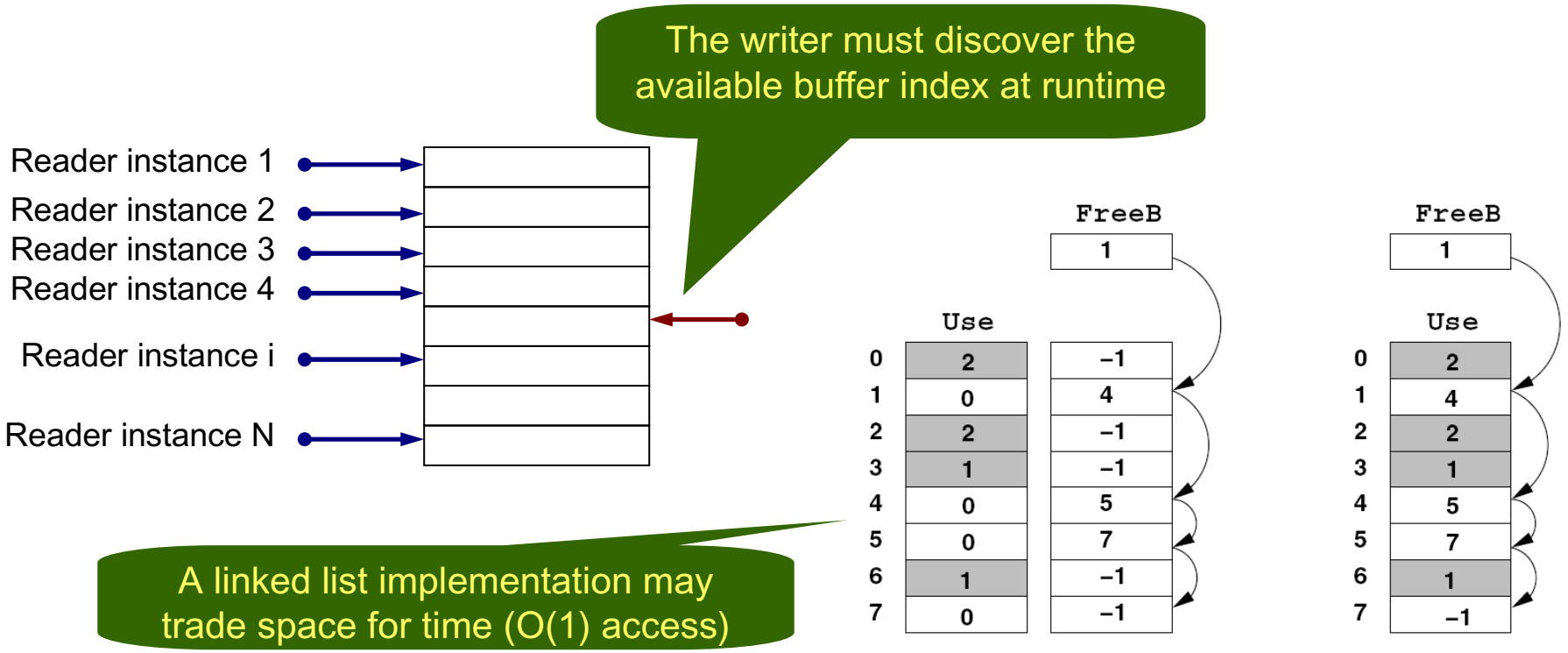
Buffer sizing methods

Two main methods

- preventing concurrent accesses by computing an *upper bound for the maximum number of buffers that can be used at any given time by reader tasks*. This number depends on the *maximum number of reader instances that can be active at any time*.
- *Temporal concurrency control*. The size of the buffer can be computed by *upper bounding the number of times the writer can produce new values, while a given data item is considered valid by at least one reader*.

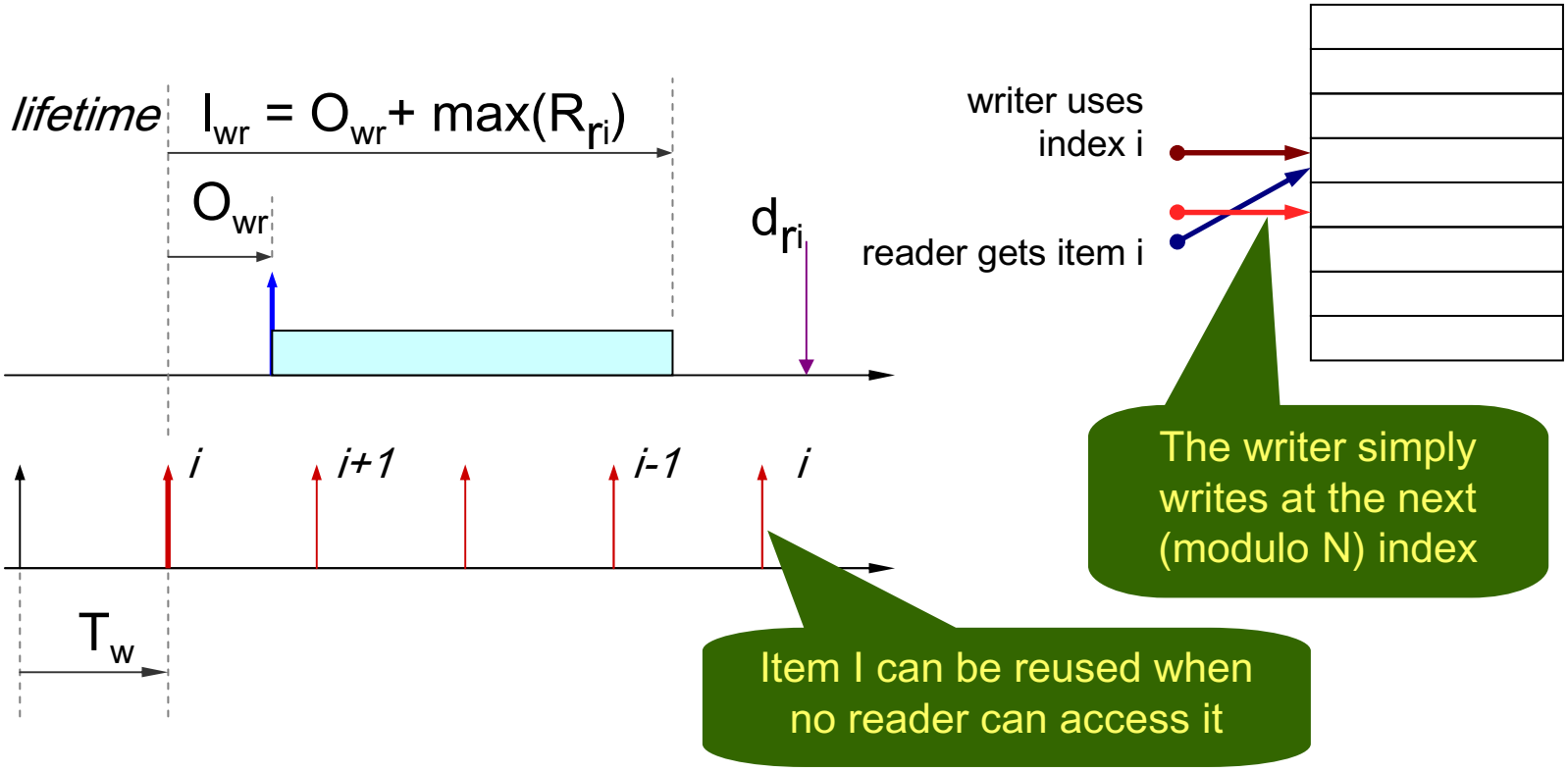
Bounding the *maximum number of reader instances*

- the size is equal to the maximum number N of reader task instances that can be active at any time (the number of reader tasks if d≤T), plus two more buffers: one for the latest written data and one for use by the writer [Chen97] (no additional information is available, and no delays on the links).



Temporal concurrency control

- Based on the concept of datum lifetime. The writer must not overwrite a buffer until the datum stored in it is still valid for some reader.



Combination

- A combination of the temporal concurrency control and the bounded number of readers approaches can be used to obtain a tighter sizing of the buffer.
- Reader tasks are partitioned into two groups: fast and slow readers. The buffer bound for the fast readers leverages the lifetime-based bound of temporal concurrency control, and the size bound for the slow ones leverages information on the maximum number of reader instances that can be active at any time. Overall, the space requirements are reduced.

Combination

- Readers of τ_{wi} are sorted by increasing lifetime ($l_i \leq l_{i+1}$). The bound

$$NB_{w_i,j} = \left\lceil \frac{l_j}{T_w} \right\rceil$$

- Applies to readers with lifetime ≤ l_j (fast readers).
- Once j is chosen, the bound is

Buffer shared among fast readers

$$NB_{w_i} = \left\lceil \frac{l_j}{T_w} \right\rceil + \sum_{i=j+1}^{NR_{w_i}} \min \left\{ \left\lceil \frac{R_{r_i}}{T_{r_i}} \right\rceil \right\}$$

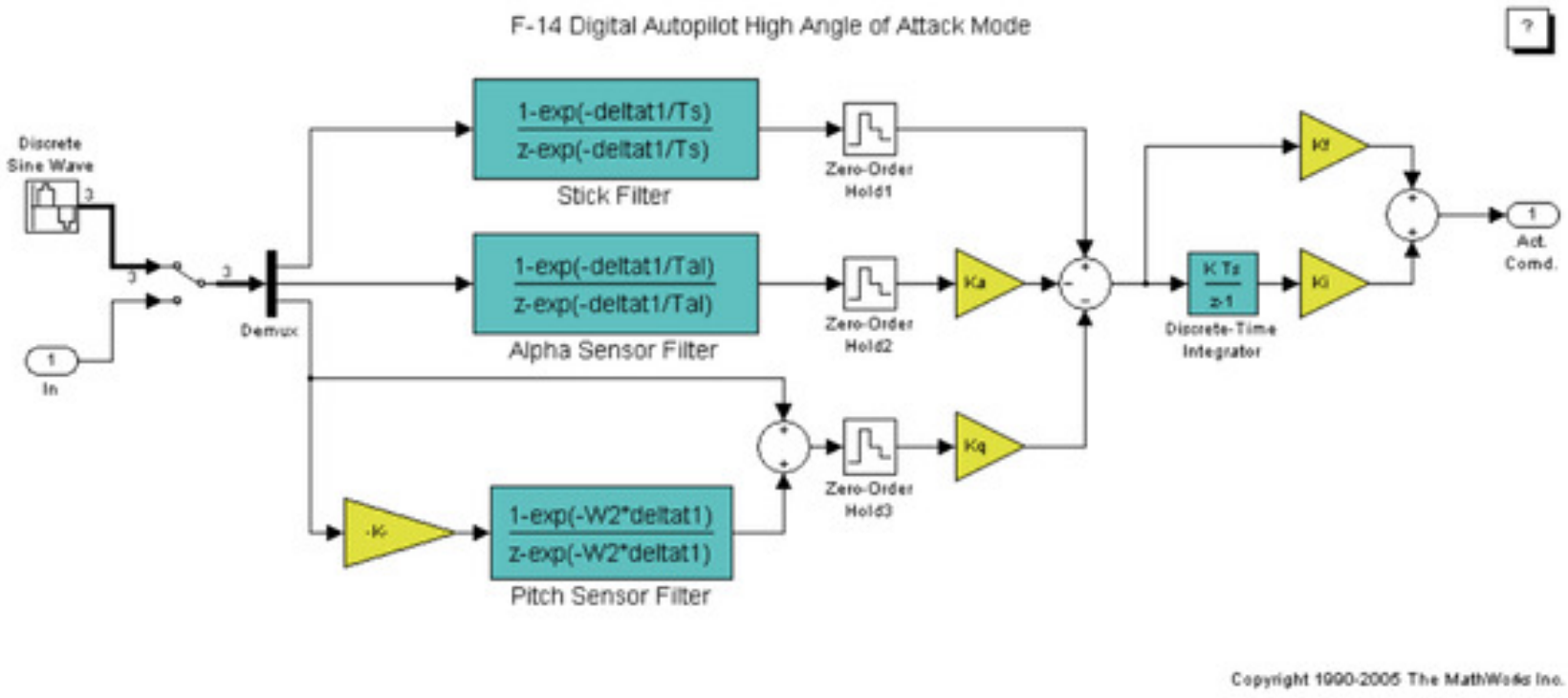
based on the number of reader instances inside the lifetime

$$j \in 0..NR_{w_i} |$$

$$\min \left\{ \left\lceil \frac{l_j}{T_w} \right\rceil + \sum_{i=j+1}^{NR_{w_i}} \min \left\{ \left\lceil \frac{R_{r_i}}{T_{r_i}} \right\rceil \right\} + \max_{i=j+1}^{NR_{w_i}} delay[i] \right\}$$

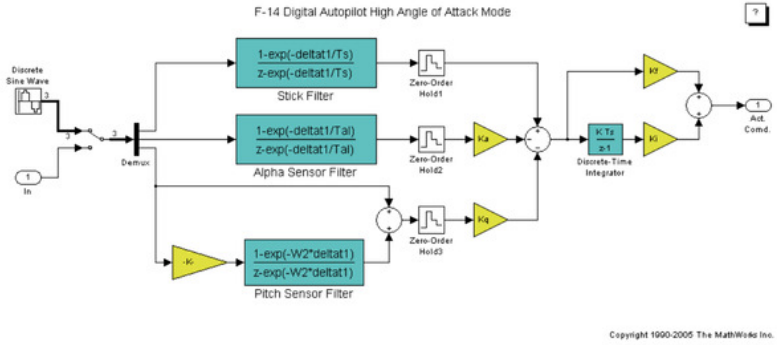
Modeling Real-time systems

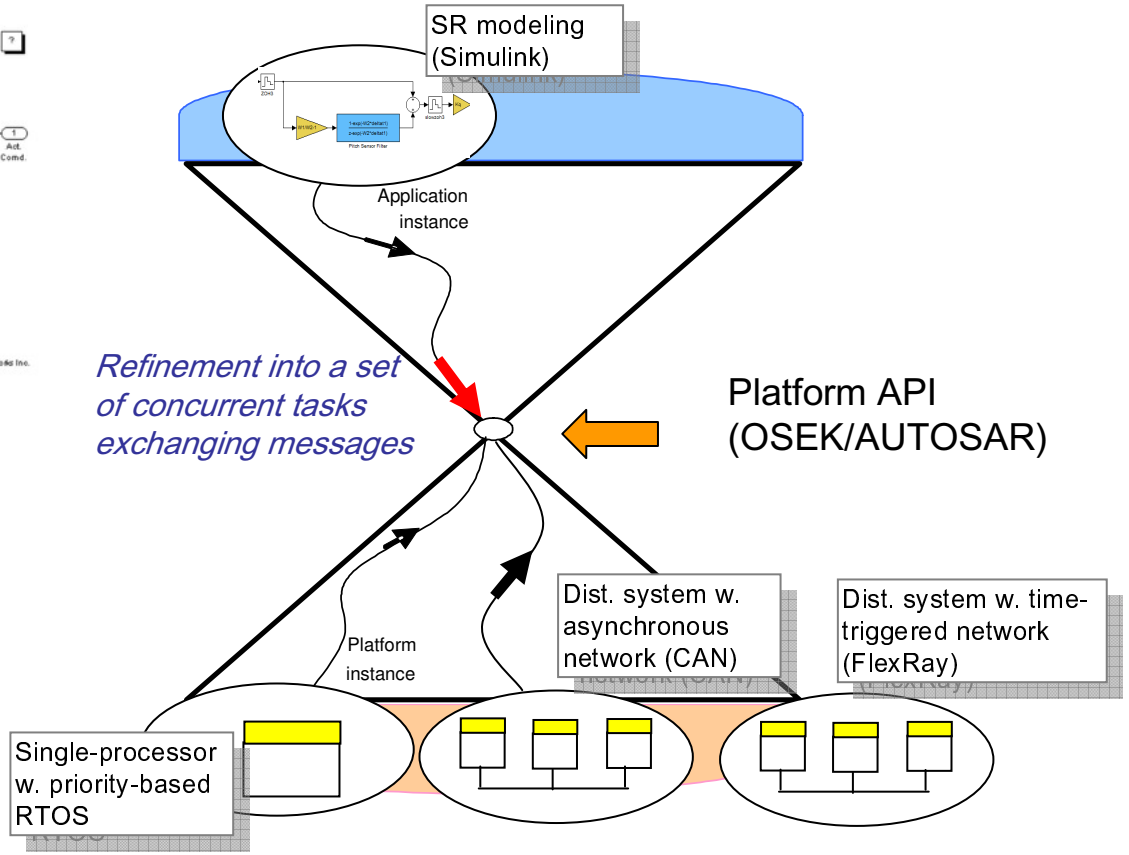
- What type of timing constraints are in a Simulink diagram?



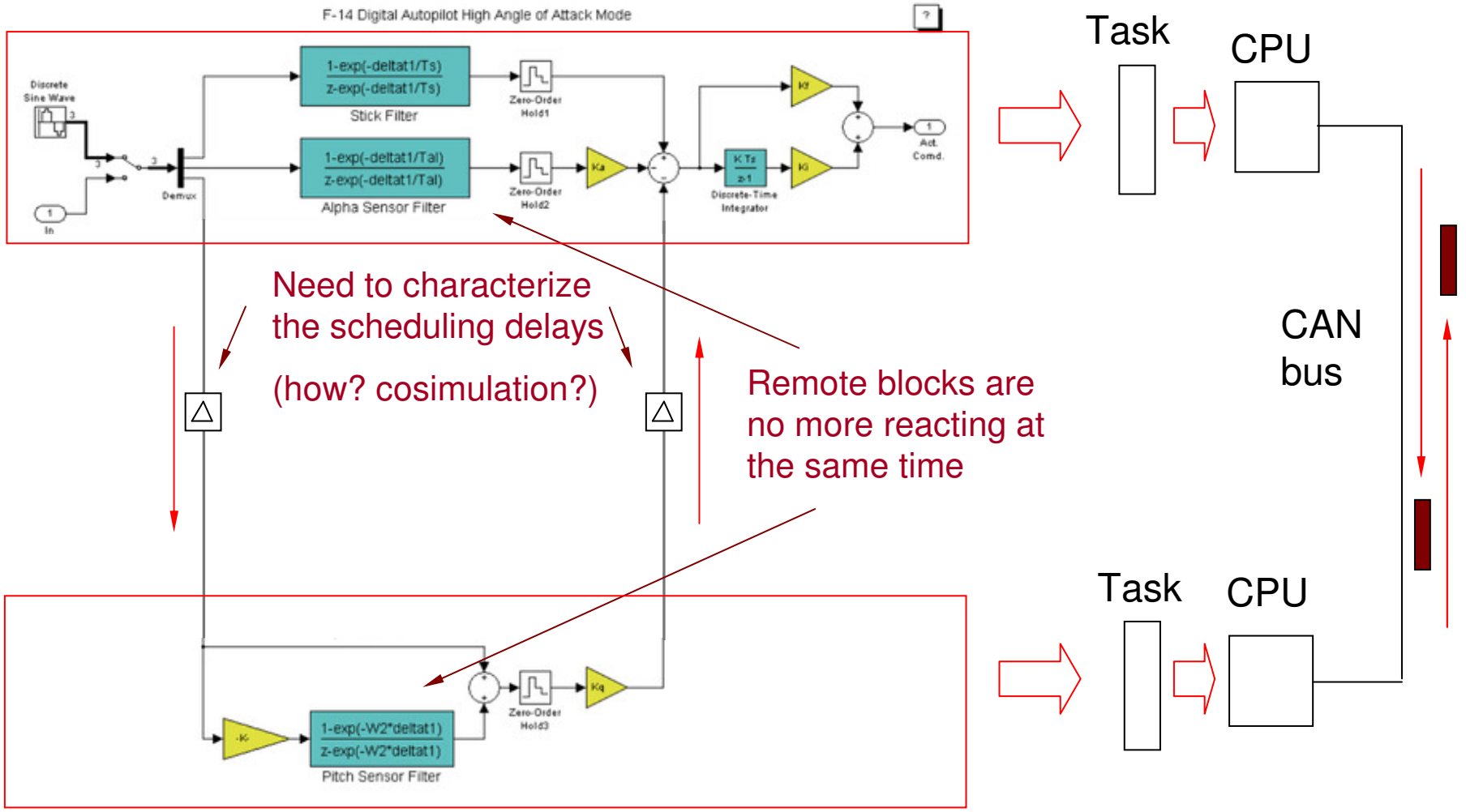
Modeling Distributed Real-time systems

- Where is the task model, the implementation relation and the deployment model?

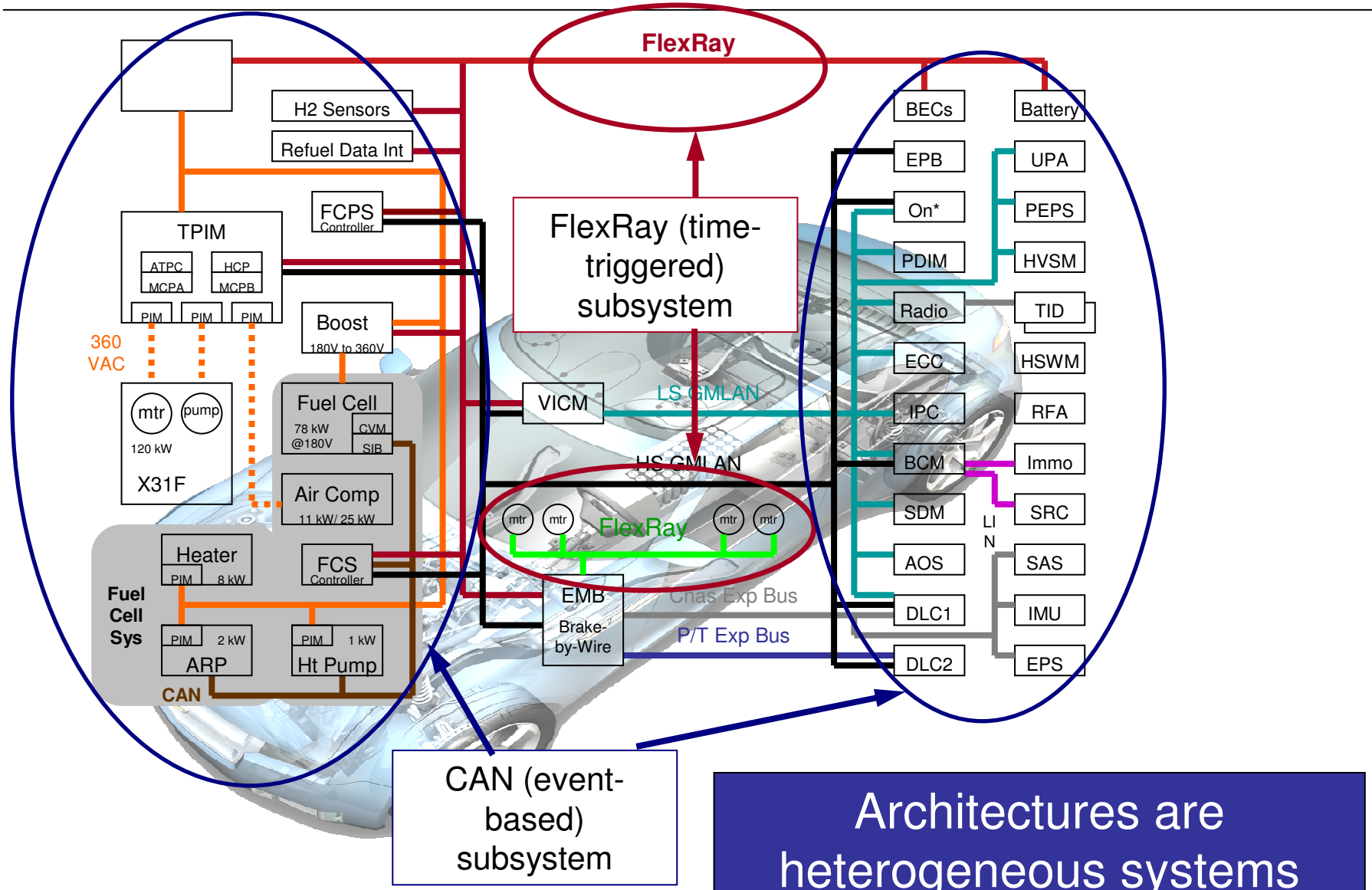




Distributed implementation of models



Heterogeneous Network topology

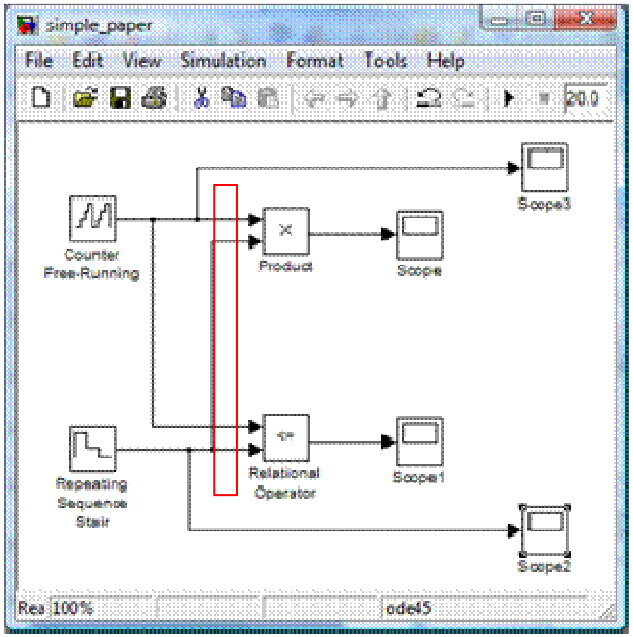


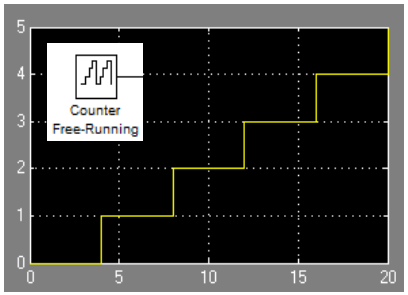
Delays from network

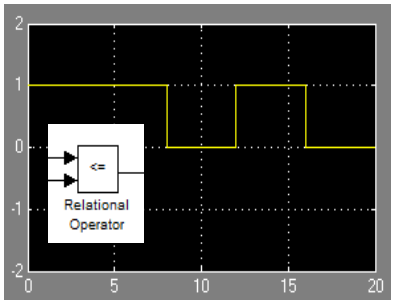
A very simple model with oversampling ….

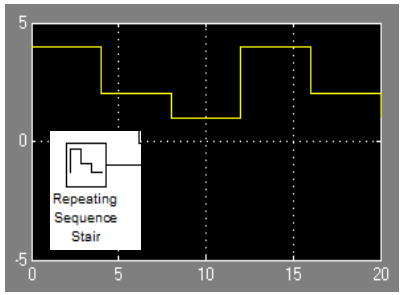
Imagine the data streams between source blocks and the multiplier/comparator are exchanged over a network.

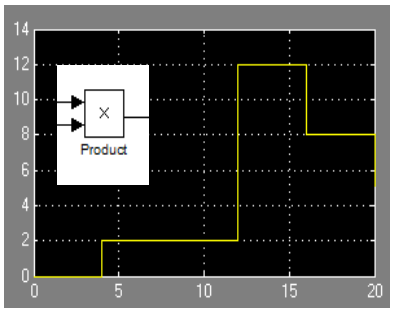
These are the results seen by the control engineer at design time





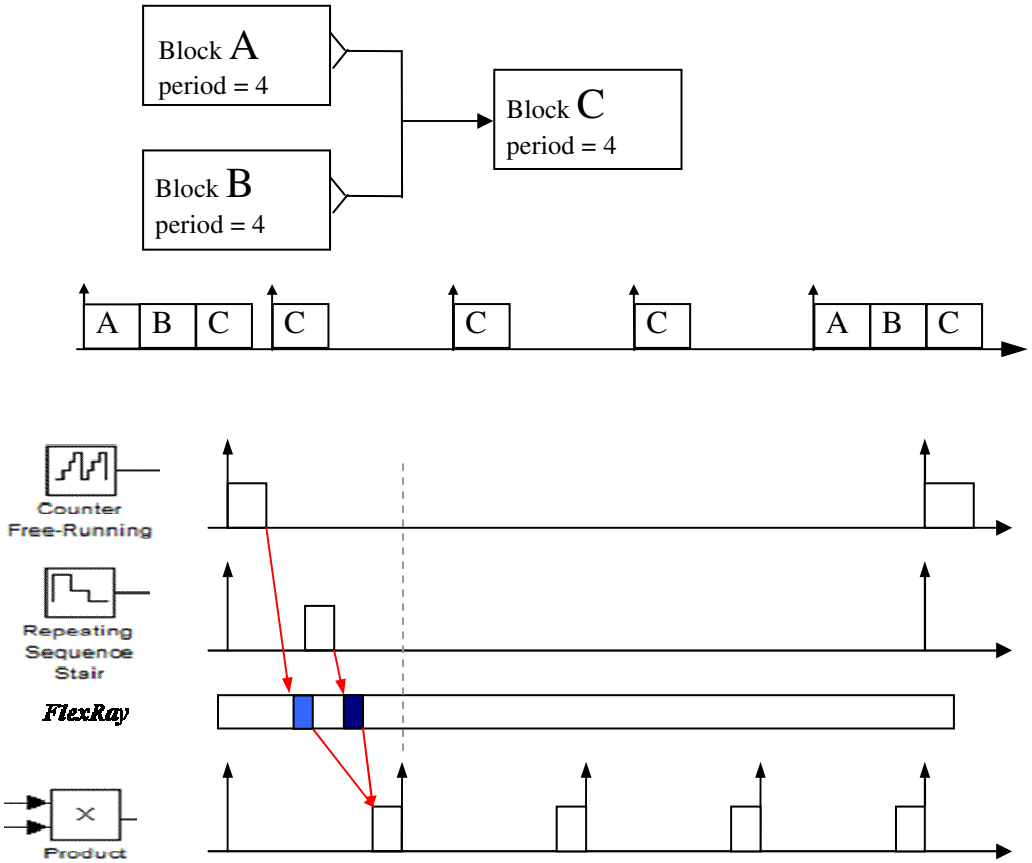






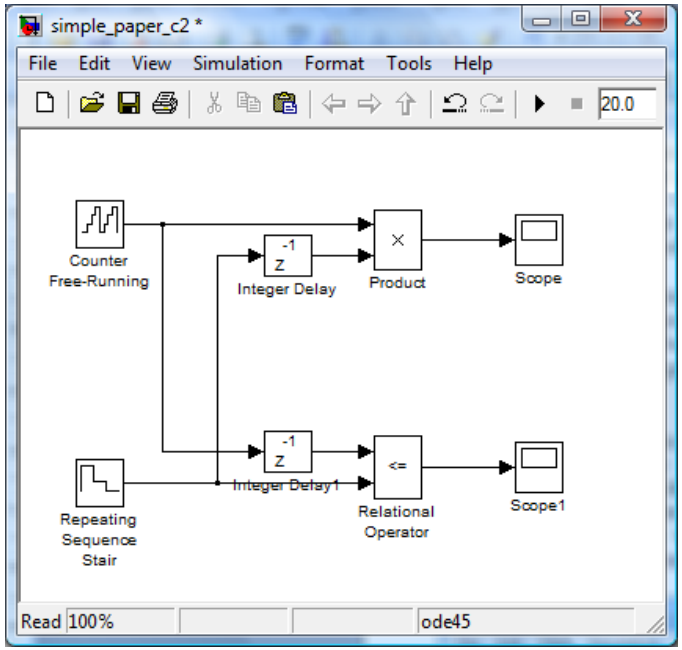
Delays from network

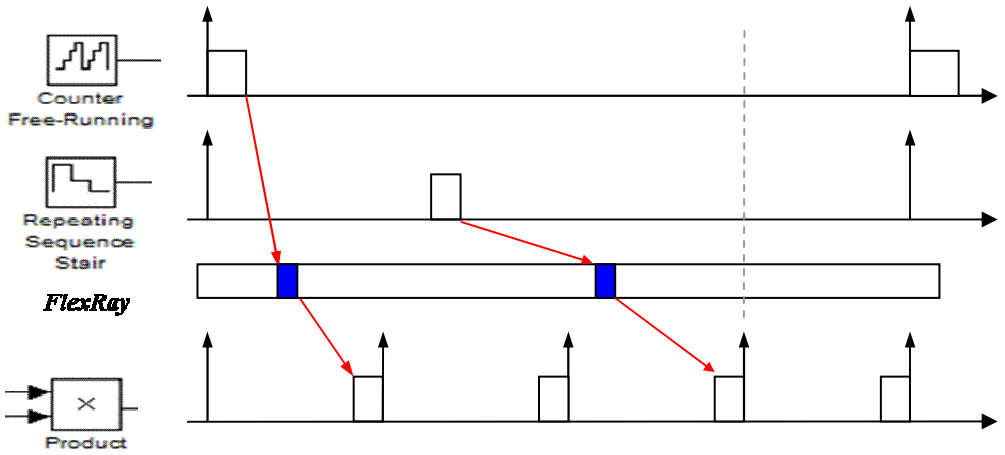
An example of the trade-offs between additional functional delays and scheduling feasibility



Delays from network

Designers may be tempted to ease the scheduling problem by choosing the instance of the receiving task/block





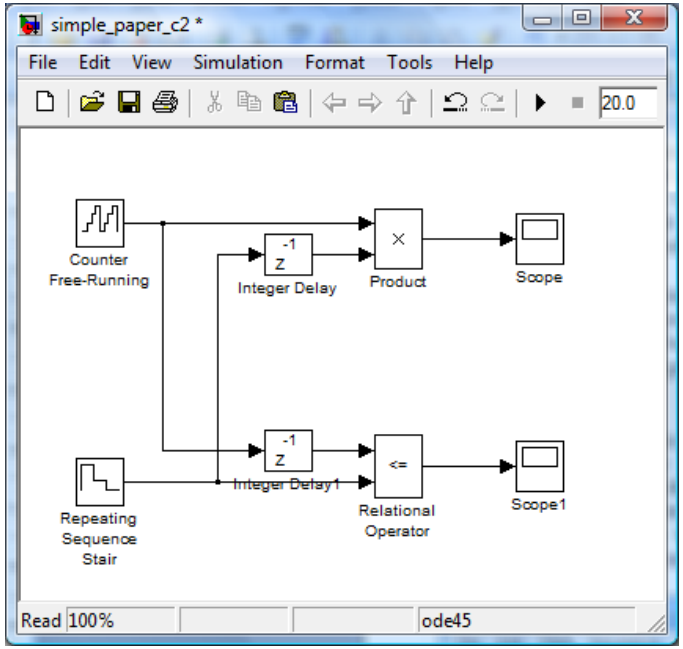
Delays from network

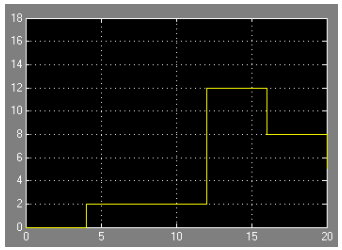
Unfortunately, by doing so, the behavior is different from the one simulated with 0-delay

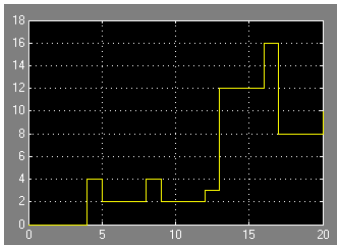
Are the designers/developers fully aware of these issues ?

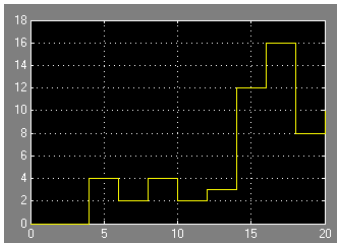
How can we help them ?

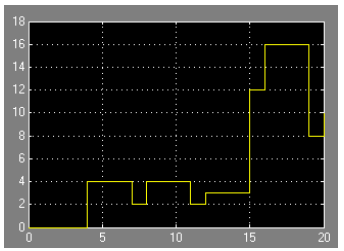
(Task and message design and scheduling are in the background)







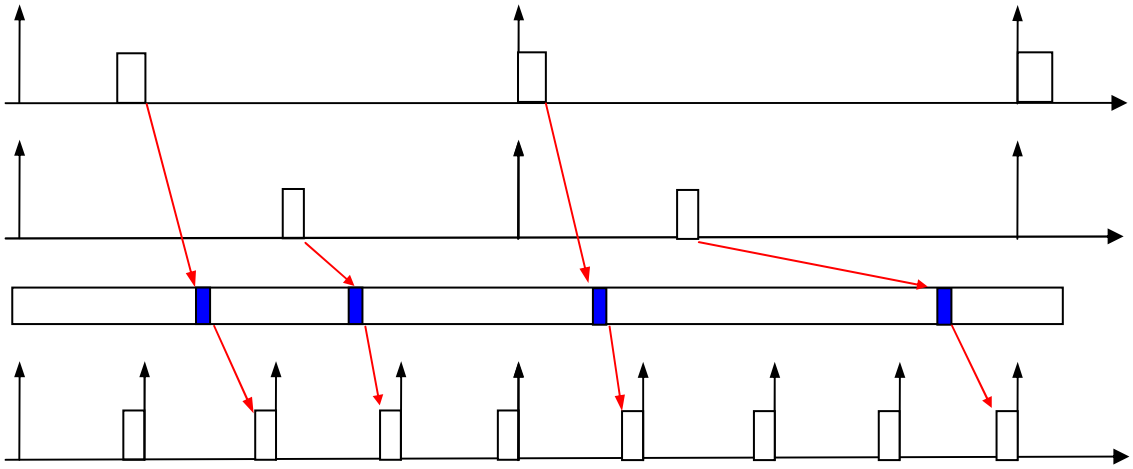




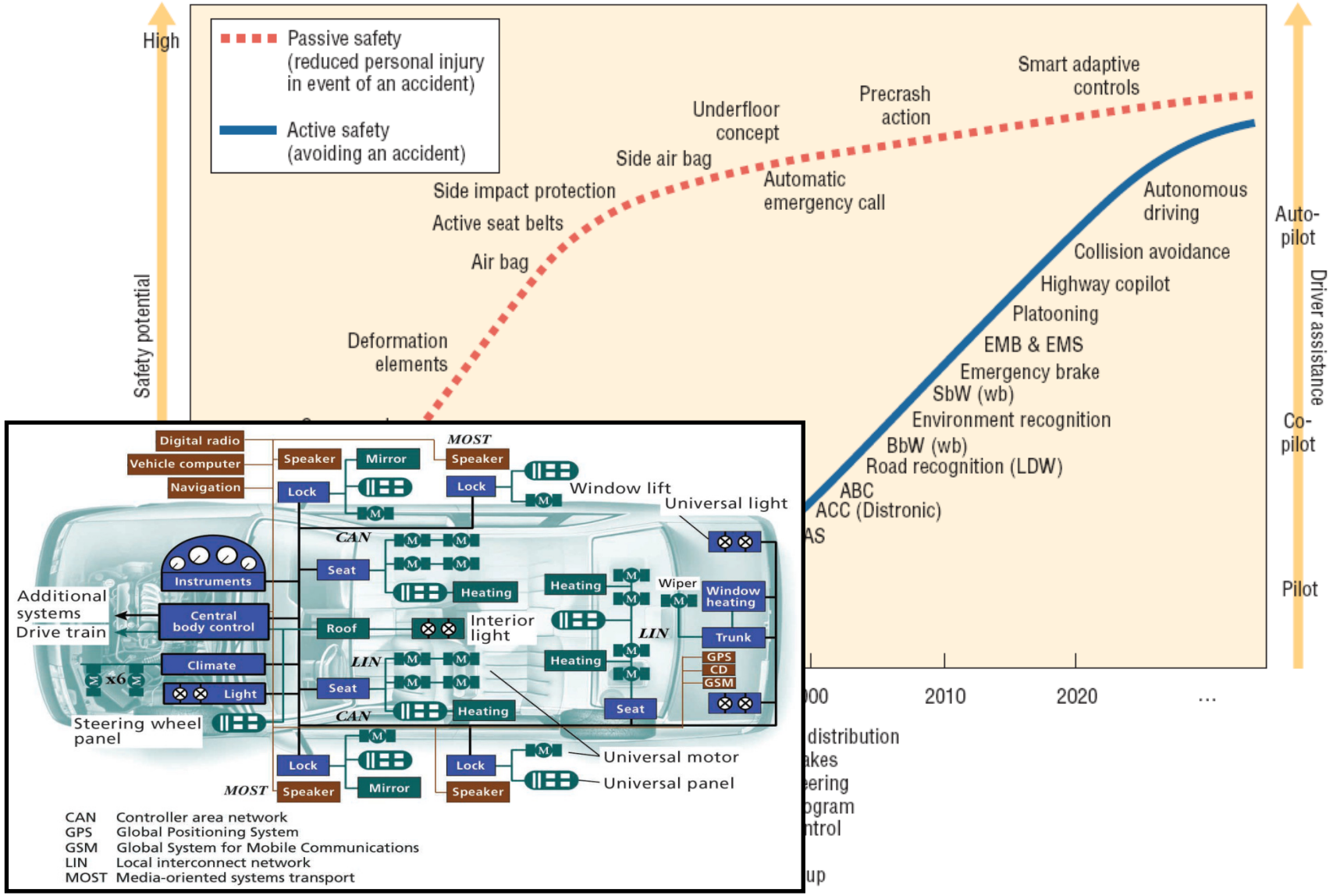
Delays from network

Unfortunately, solutions like this are possible

(not to mention issues with low-level communication levels /drivers and custom code)

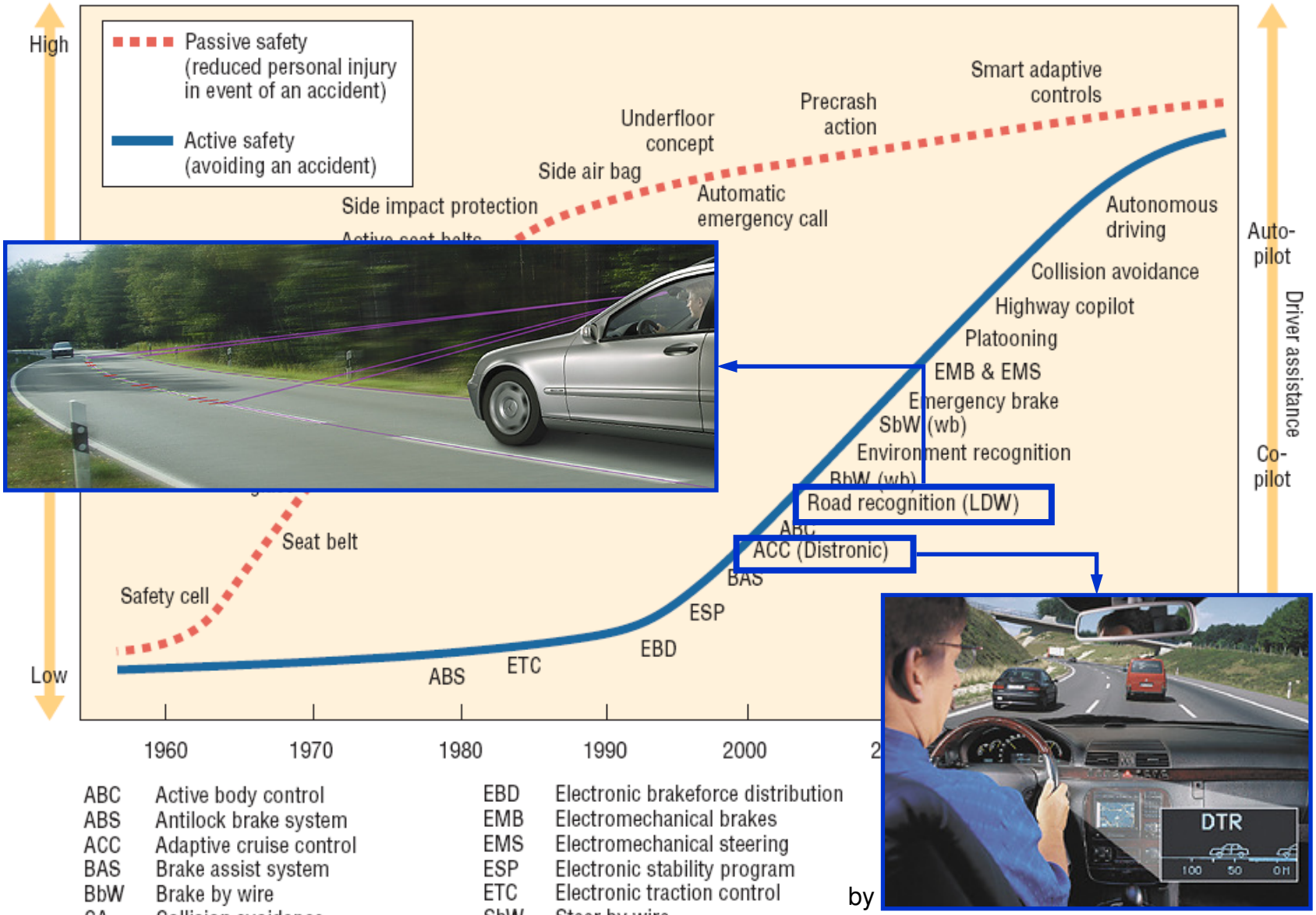


Architecture optimization vs features
- Active and Passive Safety



by Leen and Effernan – IEEE Computer

Active and Passive Safety



ABC	Active body control	EBD	Electronic brakeforce distribution
ABS	Antilock brake system	EMB	Electromechanical brakes
ACC	Adaptive cruise control	EMS	Electromechanical steering
BAS	Brake assist system	ESP	Electronic stability program
BbW	Brake by wire	ETC	Electronic traction control
CA	Collision avoidance	SbW	Steer by wire
DbW	Drive by wire	(wb)	with mechanical backup

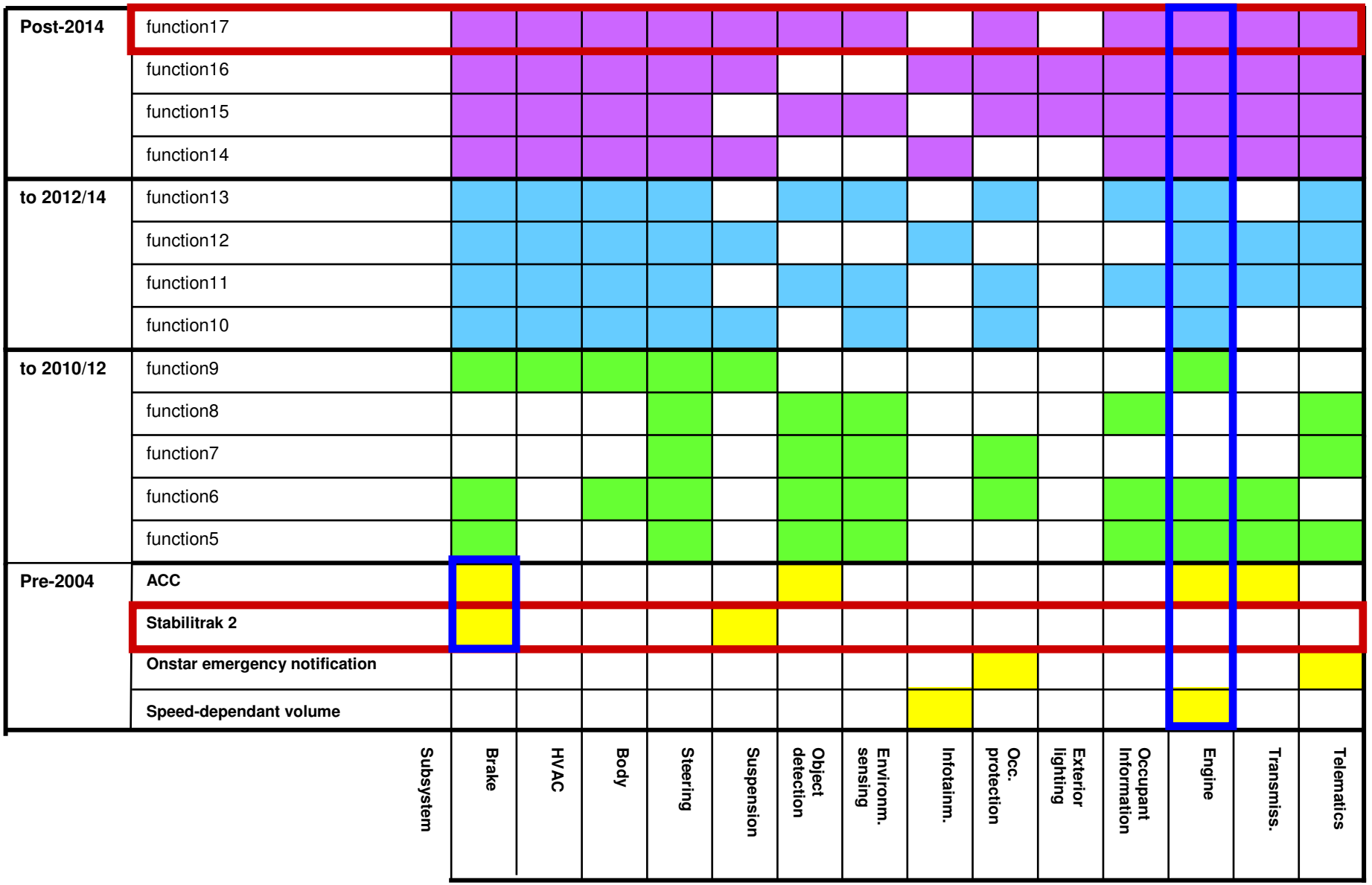
ACC (from Continental web site)

- **Adaptive Cruise Control (ACC) – Chassis Electronics Combined with Safety Aspects**

As with conventional cruise control, the driver specifies the desired velocity - ACC consistently maintains this desired speed.

In addition, the driver can enter the desired distance to a vehicle driving in front.
If the vehicle now approaches a car travelling more slowly in the same lane, ACC will recognize the diminishing distance and reduce the speed through intervention in the motor management and by braking with a maximum of 0.2 to 0.3 g until the preselected distance is reached. If the lane is clear again, ACC will accelerate to the previously selected desired tempo.

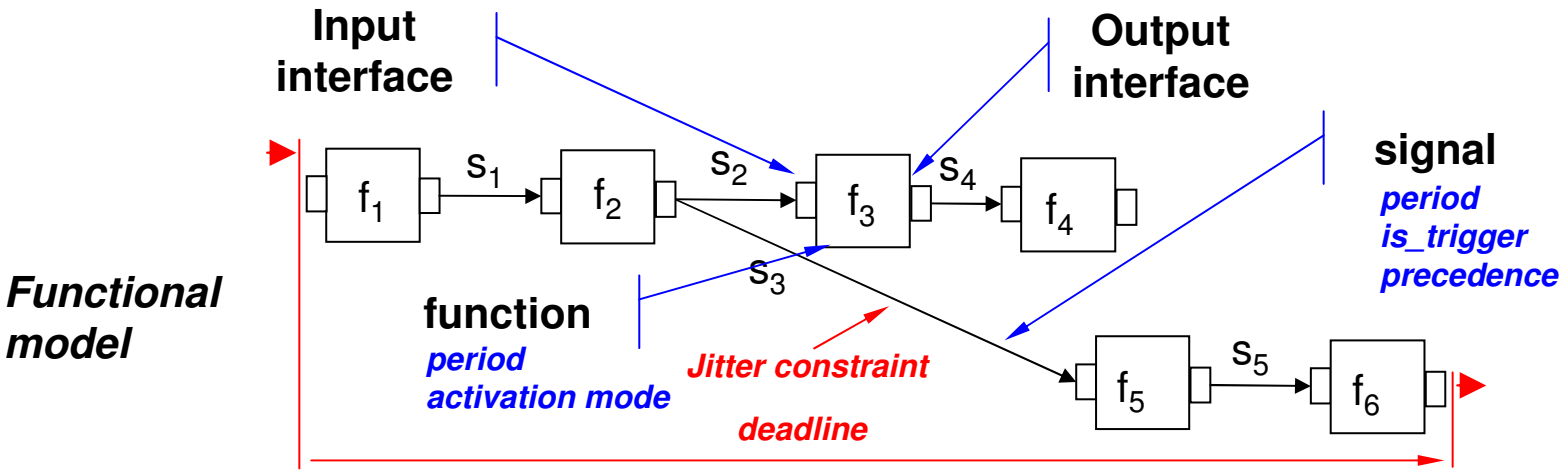
Evolution of Integrated Functions



Automotive architecture trends

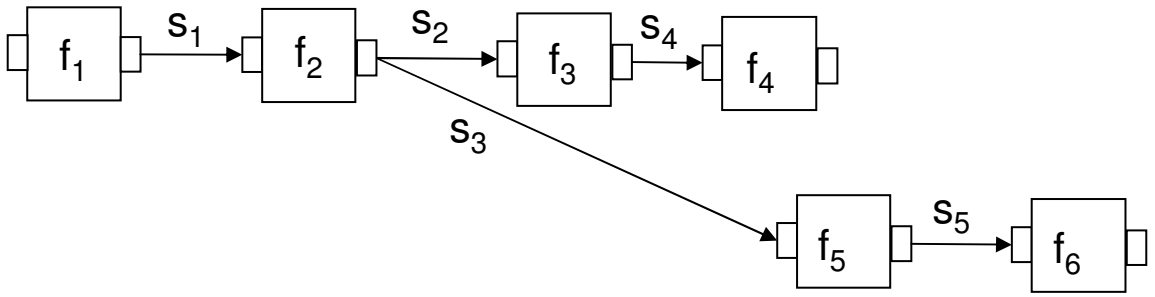
- An increasing number of functions will be distributed on a decreasing number of ECUs and enabled through an increasing number of smart sensors and actuators
  - today: > 5 buses and > 30 ECUs
- 90% of innovation in cars for the foreseeable future will be enabled through the Electronic Vehicle Architecture
- Transition from single-ECU Black-box based development processes to a system-level engineering process
  - System-level methodologies for quantitative exploration and selection,
  - From Hardware Emulation to Model Based Verification of the System
- Architectures need to be defined years ahead of production time, with incomplete information about (future) features
- Multiple non-functional requirements can be defined

Functional model

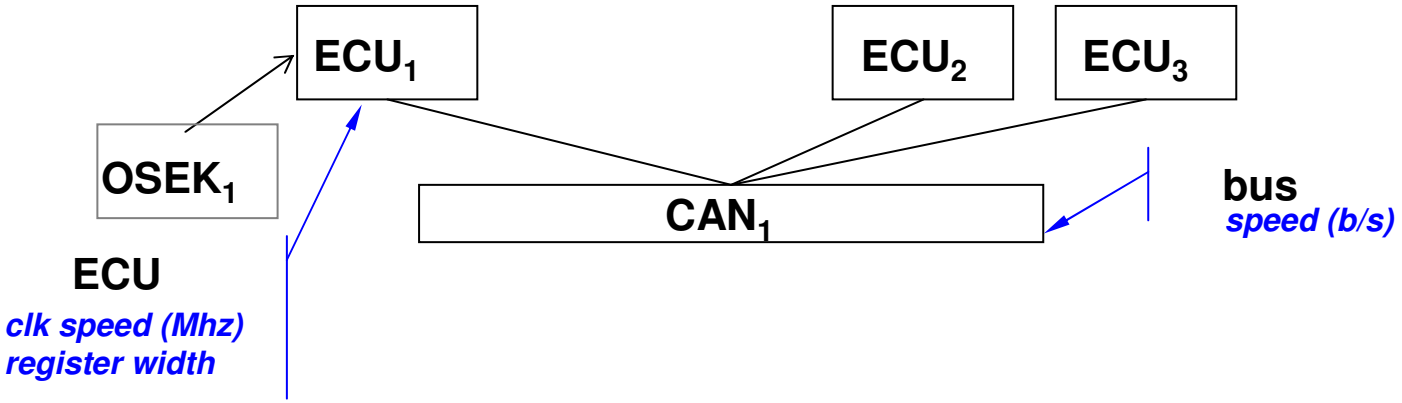


Architecture model

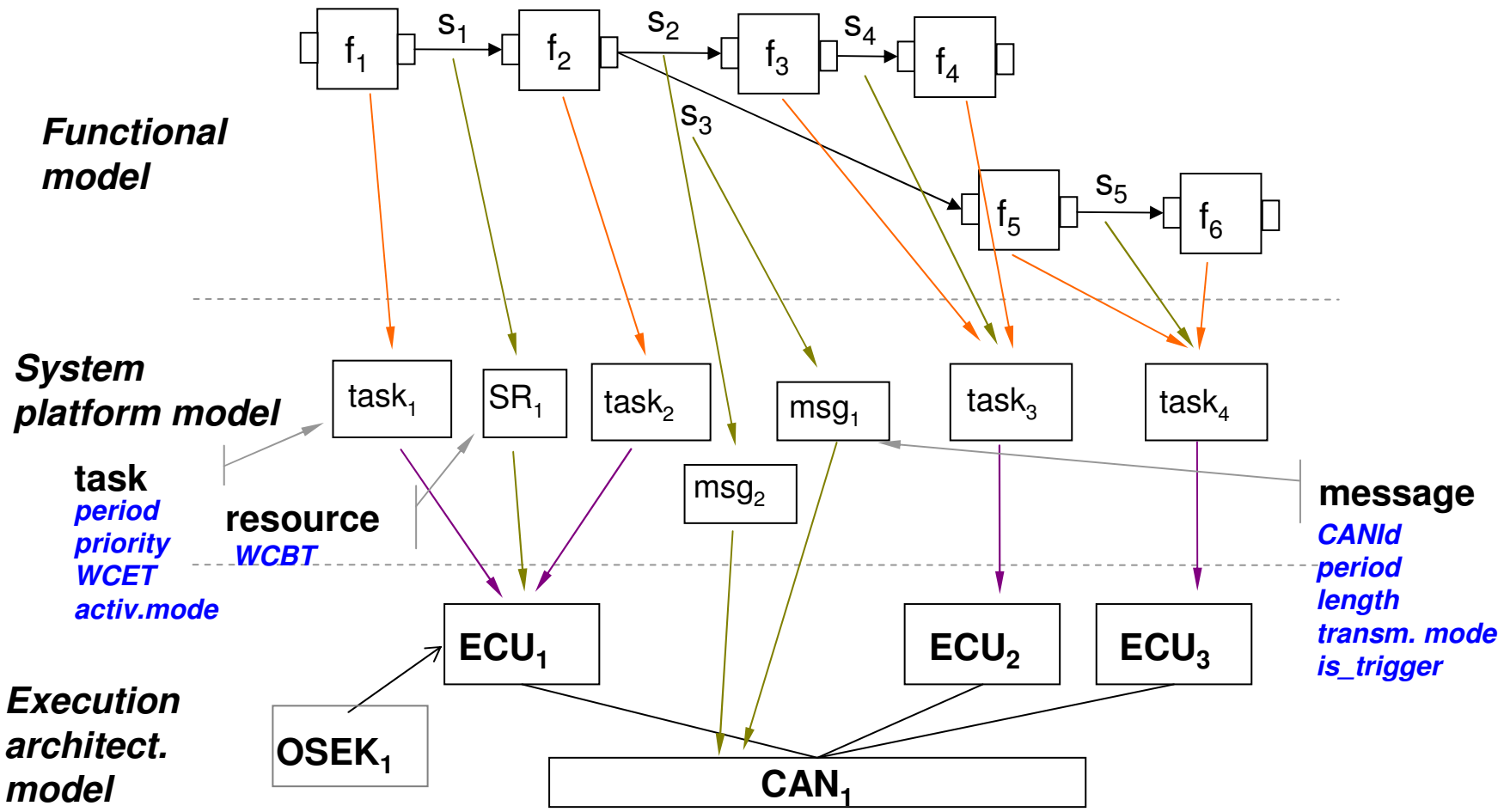
Functional model



Execution architect. model



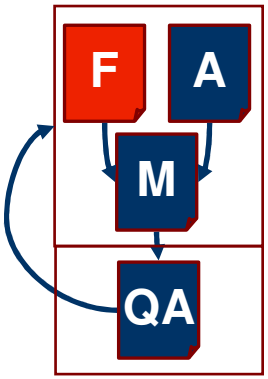
Deployment model

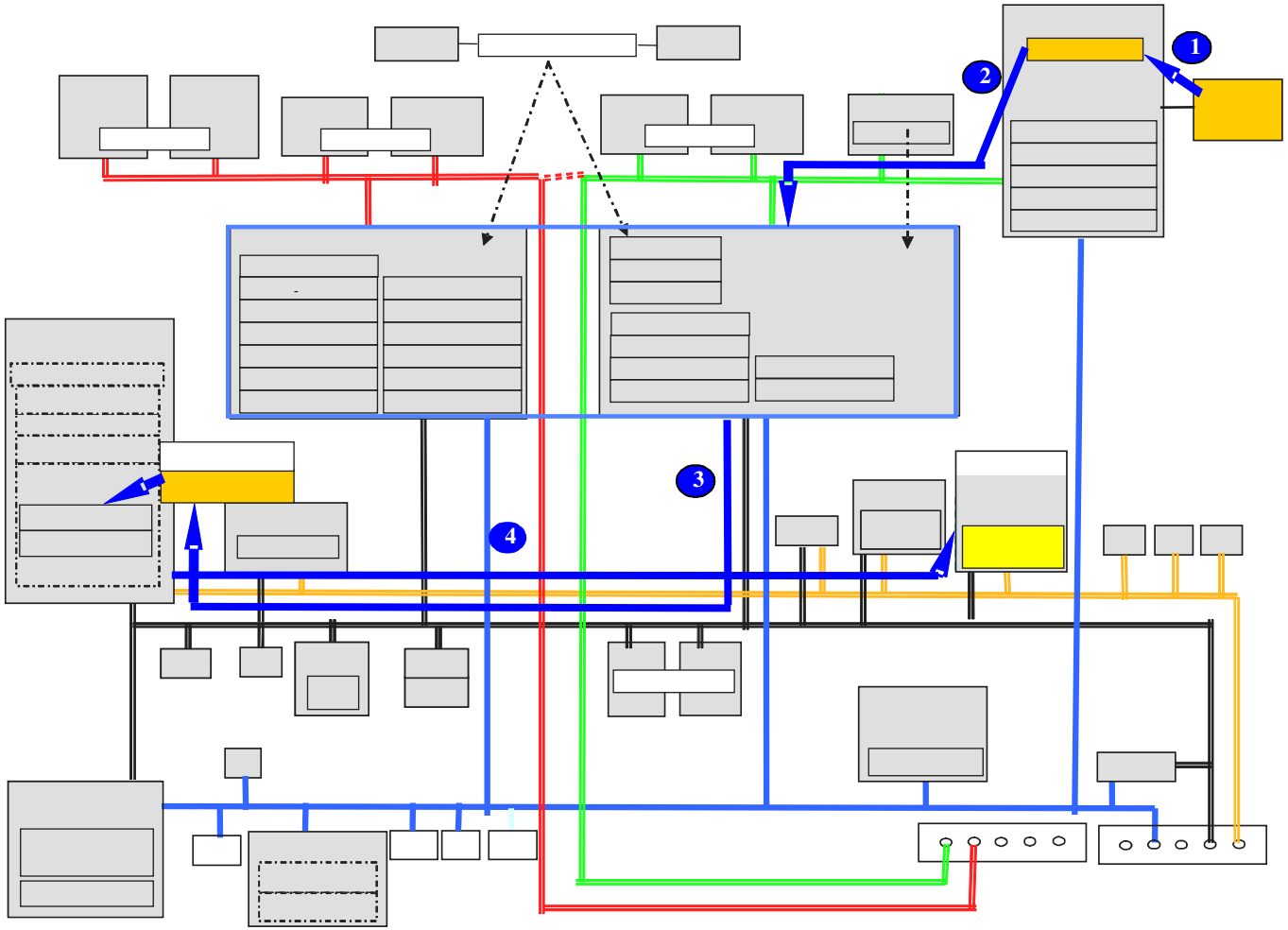


Deployment: An example

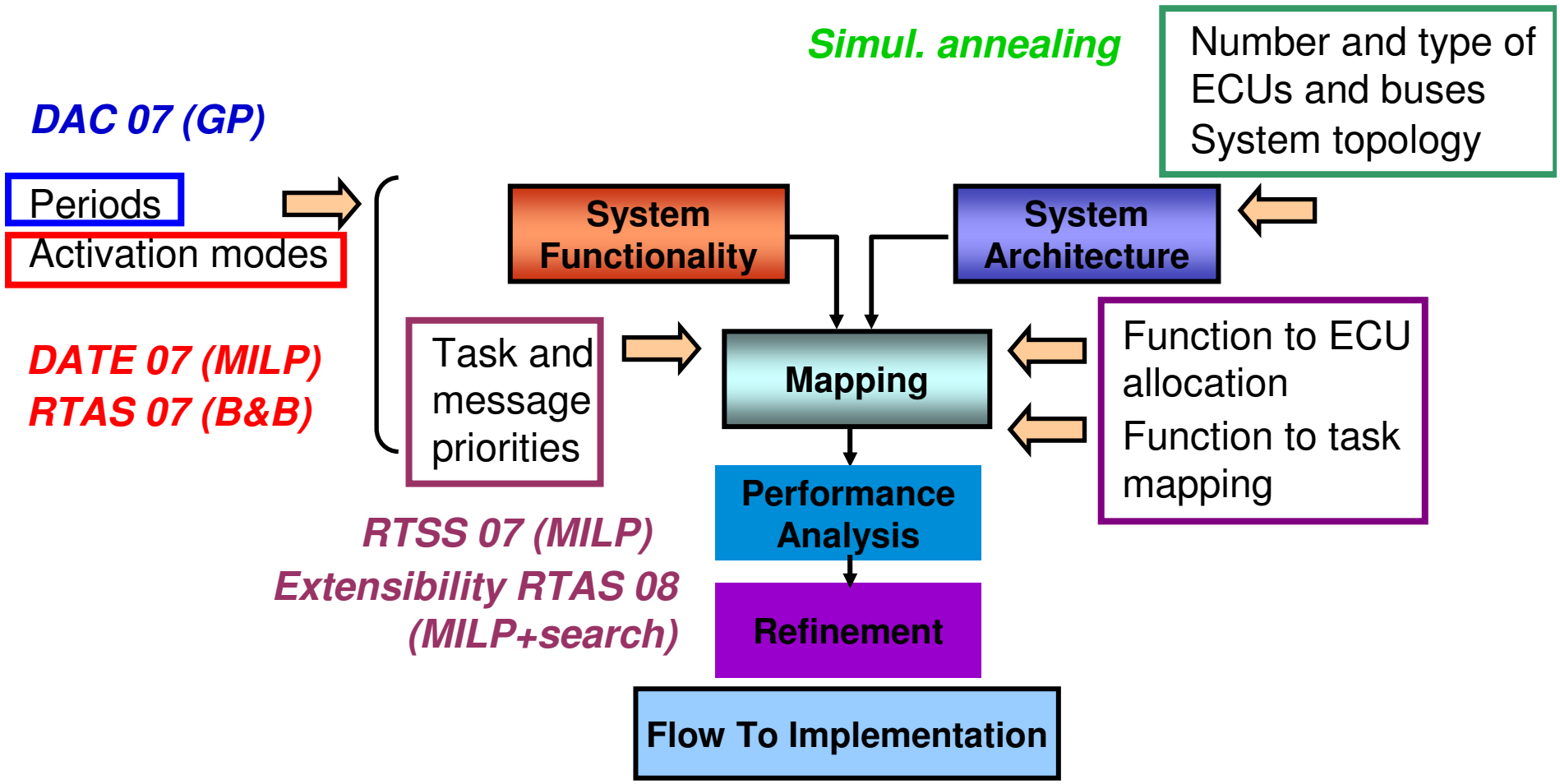
End-to-end latencies

ECU and bus utilizations





Back to architecture synthesis



Approach: Mathematical Programming

- Why Mathematical Programming?
- (compared with search, genetic programming or SA …)
  - Simplicity
    - Problem represented with:
      - Set of decision variables
      - Constraints
      - Objective function
    - "automatically" handles cross dependency among selection choices
  - Easier coding of multi-objective optimization
  - Standardized approach
    - Well established technique
    - Sound theory, methods
    - Availability of commercial solvers (in essence, search engines)
  - How good is your solution?
    - Provides safe estimate of optimal solution
    - Provides intermediate solutions of increasing quality
- Challenge:
  - Capture the problem and obtain efficient runtimes

(Example) Problem Formulation

Objective

Minimization of (average case) end-to-end latencies

Subject to

- **Constraints on end-to-end latencies**
- Constraints on messages size
- Constraints on utilization
- **Constraints on message and task deadlines**
- **Semantics preservation constraints**

Design objectives (optimization variables)

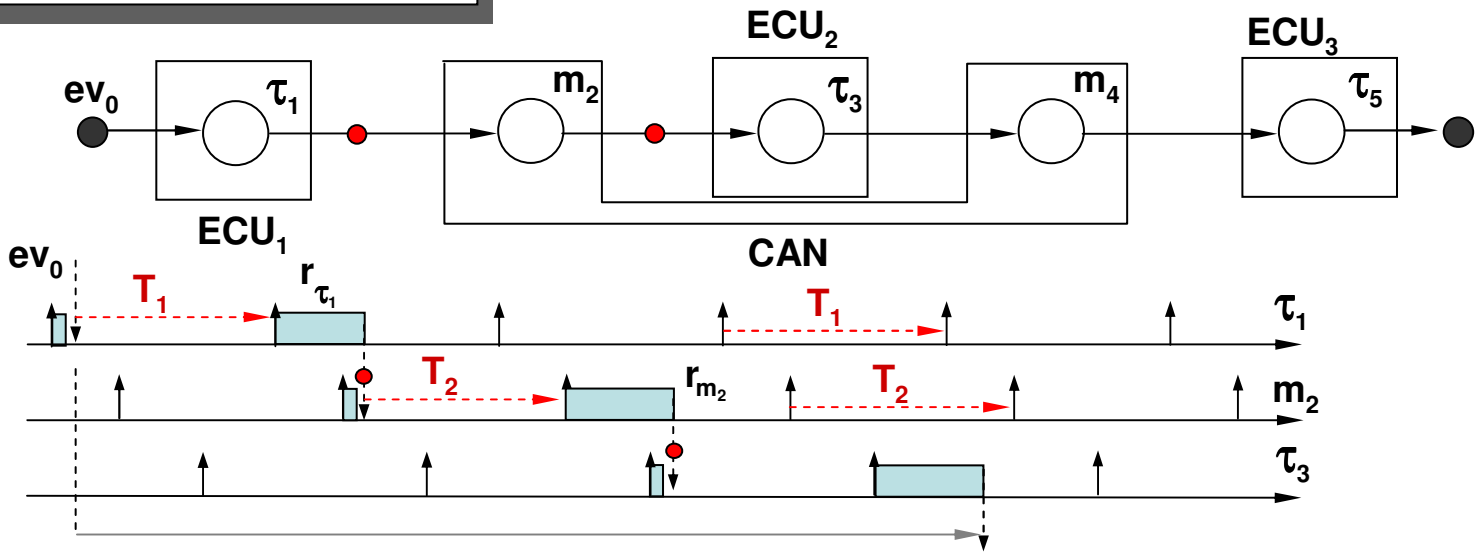
- Placement of tasks onto the CPUs
- Packing of signals to messages
- Assignment of priorities to tasks and messages
- Definition of activation modes/synchronization model
- Period optimization

Periodic Activation Model

High latency, but allows decoupling the scheduling problem

End-to-end latency analysis

Periodic asynchronous activation model



$$l_{(i,j)} = \sum_{k:o_k \in P(i,j)} (T_k + r_k)$$

where (approx.)

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

Worst Case Response Times

Tasks: $$r_i = c_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{t_j} \right\rceil c_j \qquad \forall o_i \in \mathcal{T}$$

Messages: $$r_i = c_i + b_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i - c_i}{t_j} \right\rceil c_j \quad \forall o_i \in \mathcal{M}$$

- Resource utilization
  - Fraction of time the resource (ECU or bus) spends processing its objects (tasks or messages)
- Utilization bounds less than 100%
  - To allow for future extensibility

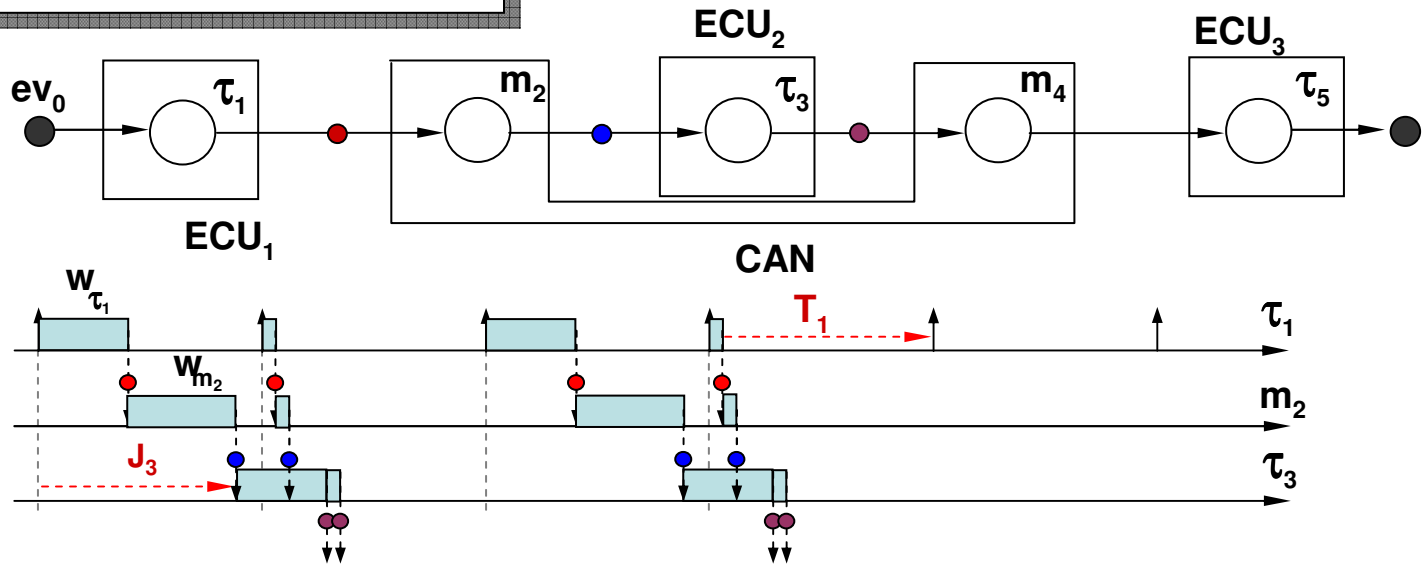
$$\left(\sum_{i:o_i \to R_j} \frac{c_i}{t_i} \right) \leq u_j \quad \forall R_j \in \mathbf{R}$$

Event-based Activation Model

Lower latency for high priority paths, jitter increases along the path

End-to-end latency analysis

Data-driven precedence constrained activation model



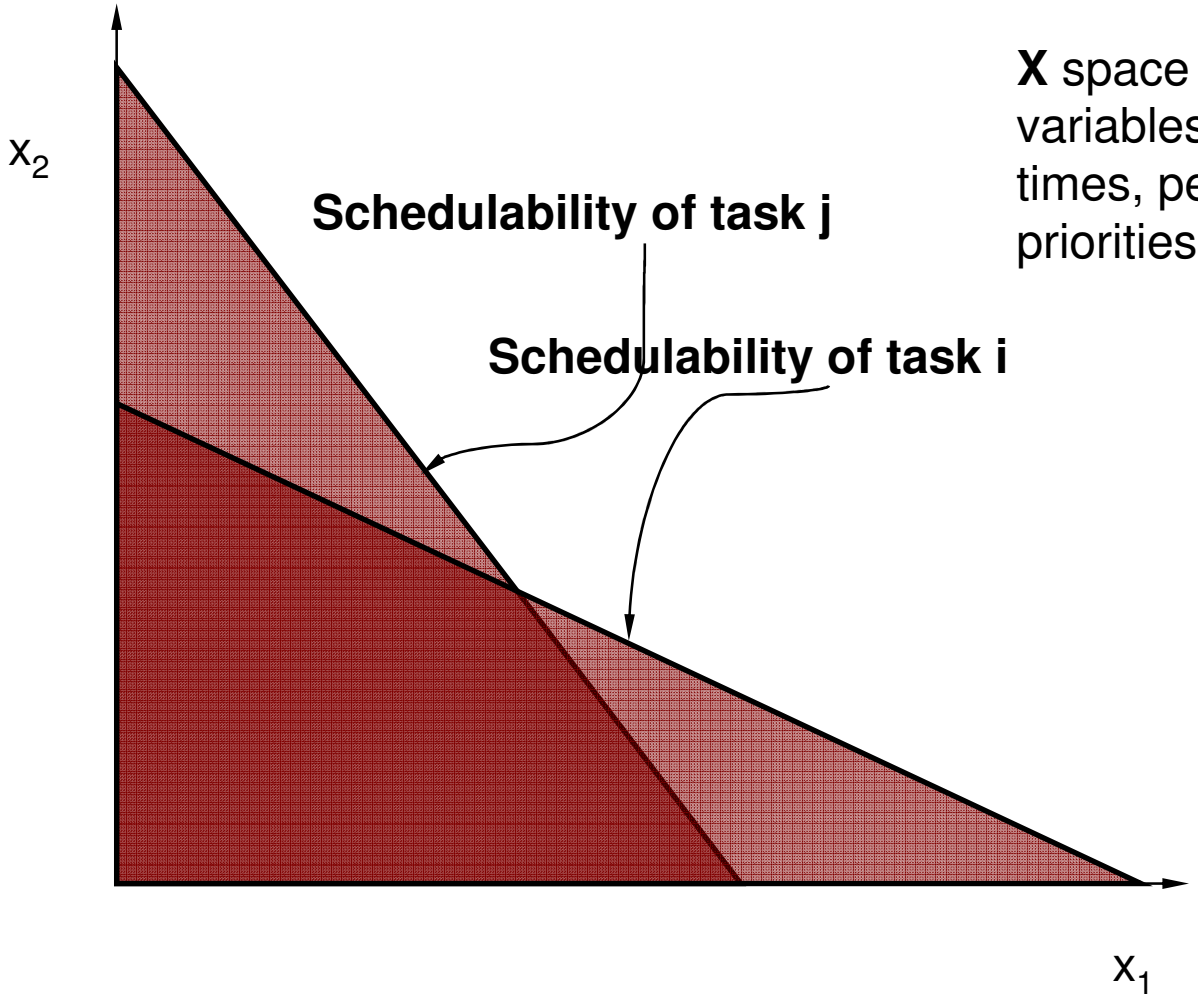
$$l_{(i,j)} = \sum_{k:o_k \in P(i,j)} w_k$$

where (approx.)

$$w_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

Design Process and Requirements

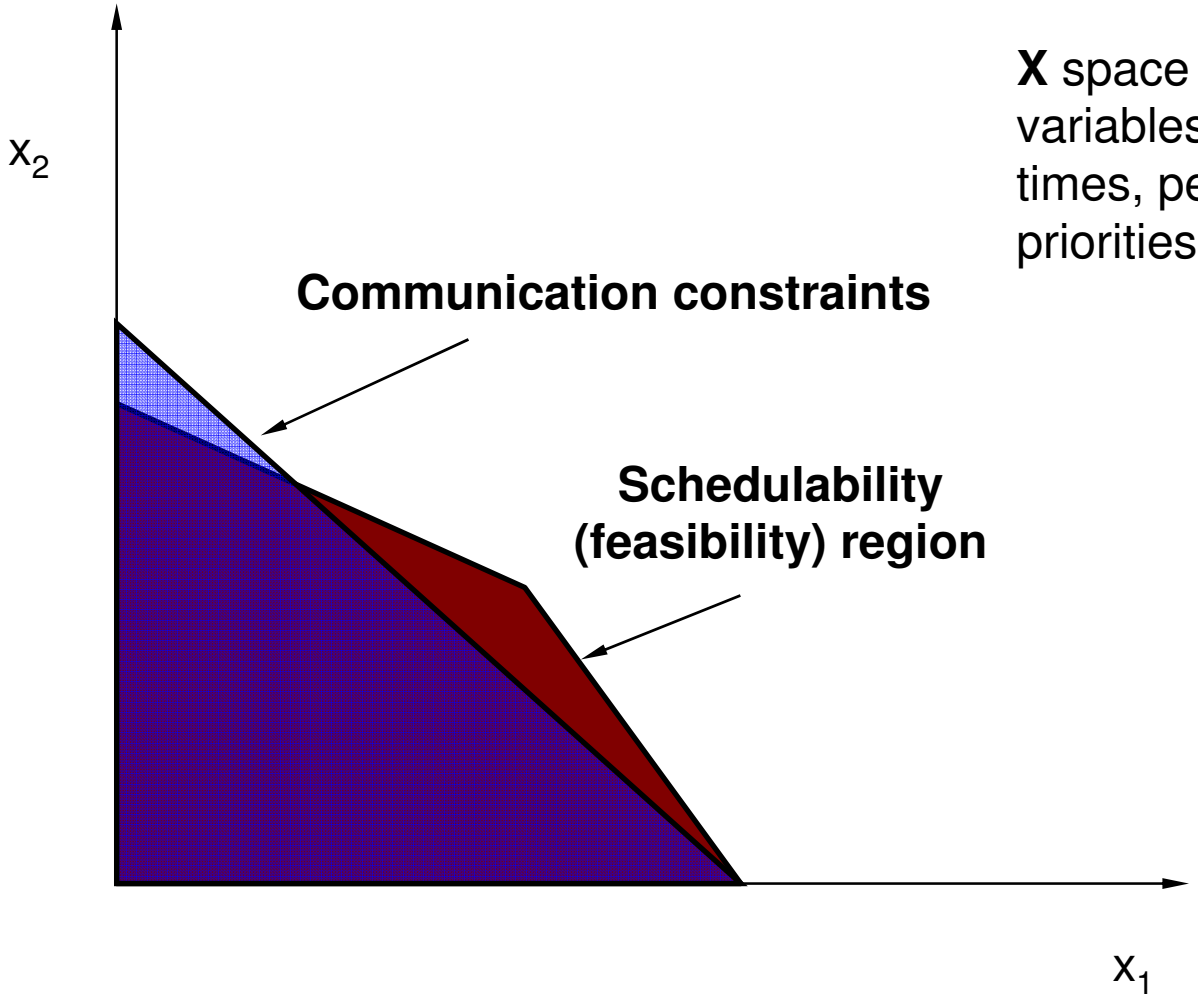
- Design optimization



X space of design optimization variables, such as computation times, periods, placement, priorities …

Design Process and Requirements

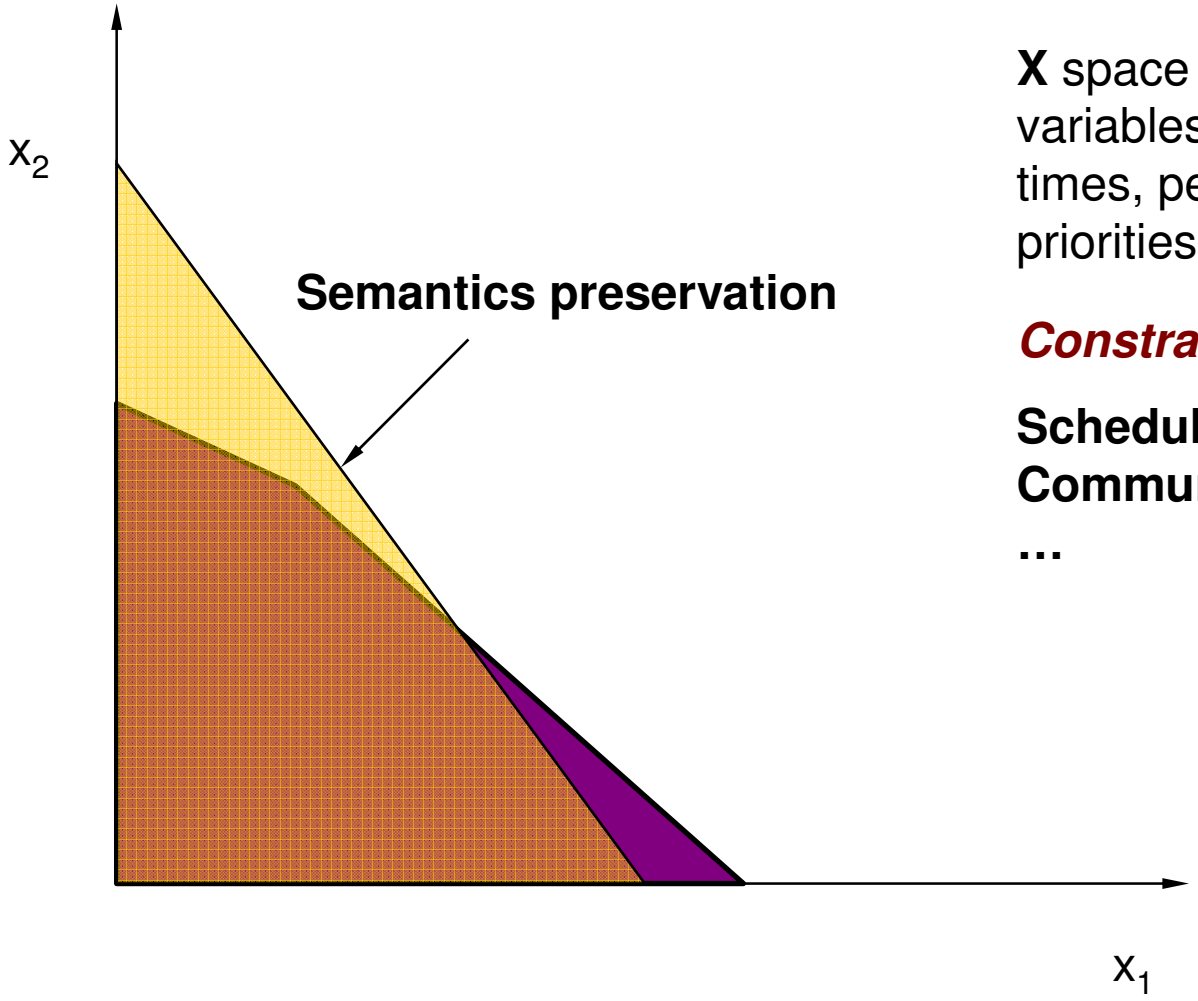
- Design optimization



X space of design optimization variables, such as computation times, periods, placement, priorities …

Design Process and Requirements

- Design optimization



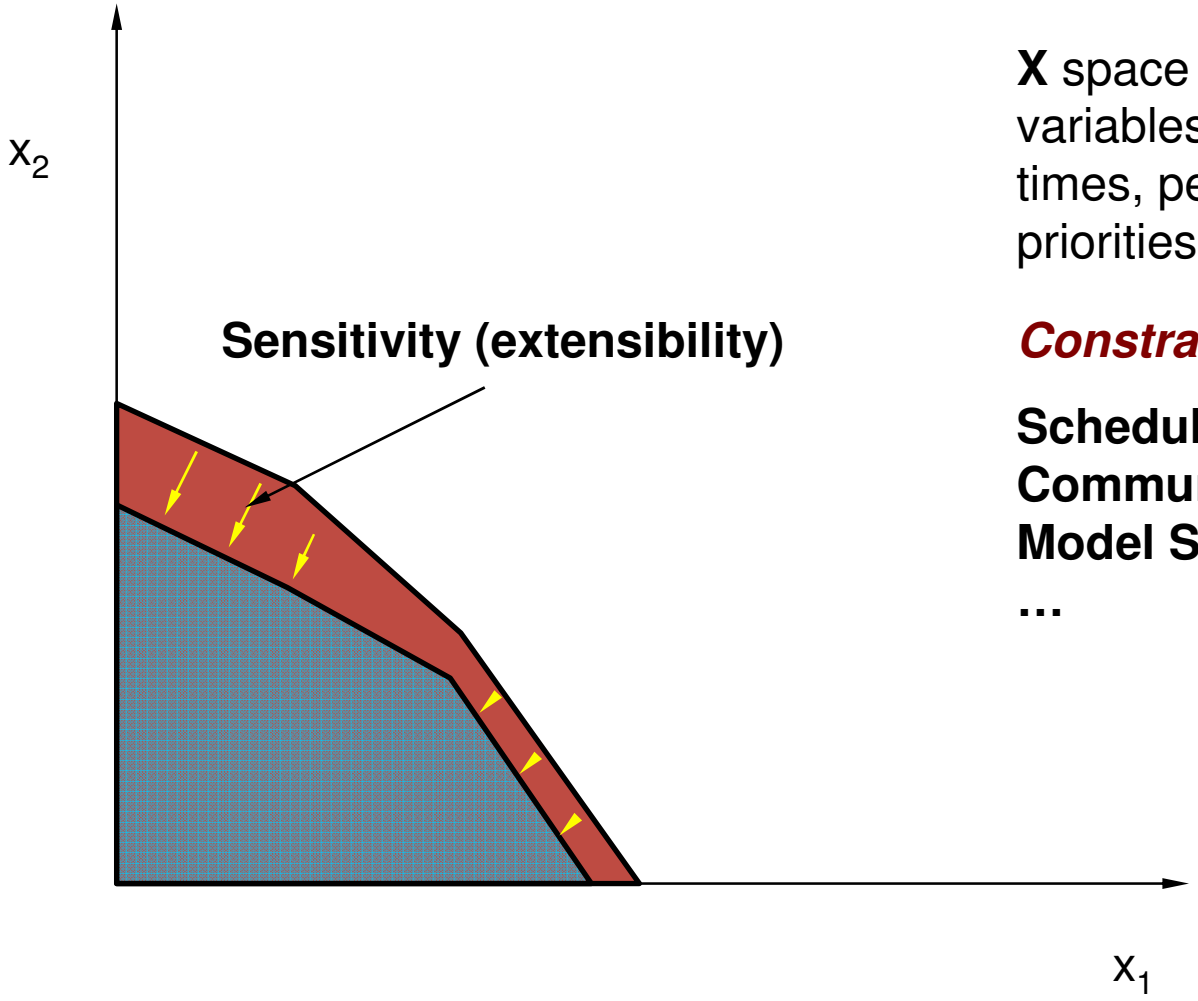
X space of design optimization variables, such as computation times, periods, placement, priorities …

Constraints

Schedulability
Communication
…

Design Process and Requirements

- Design optimization



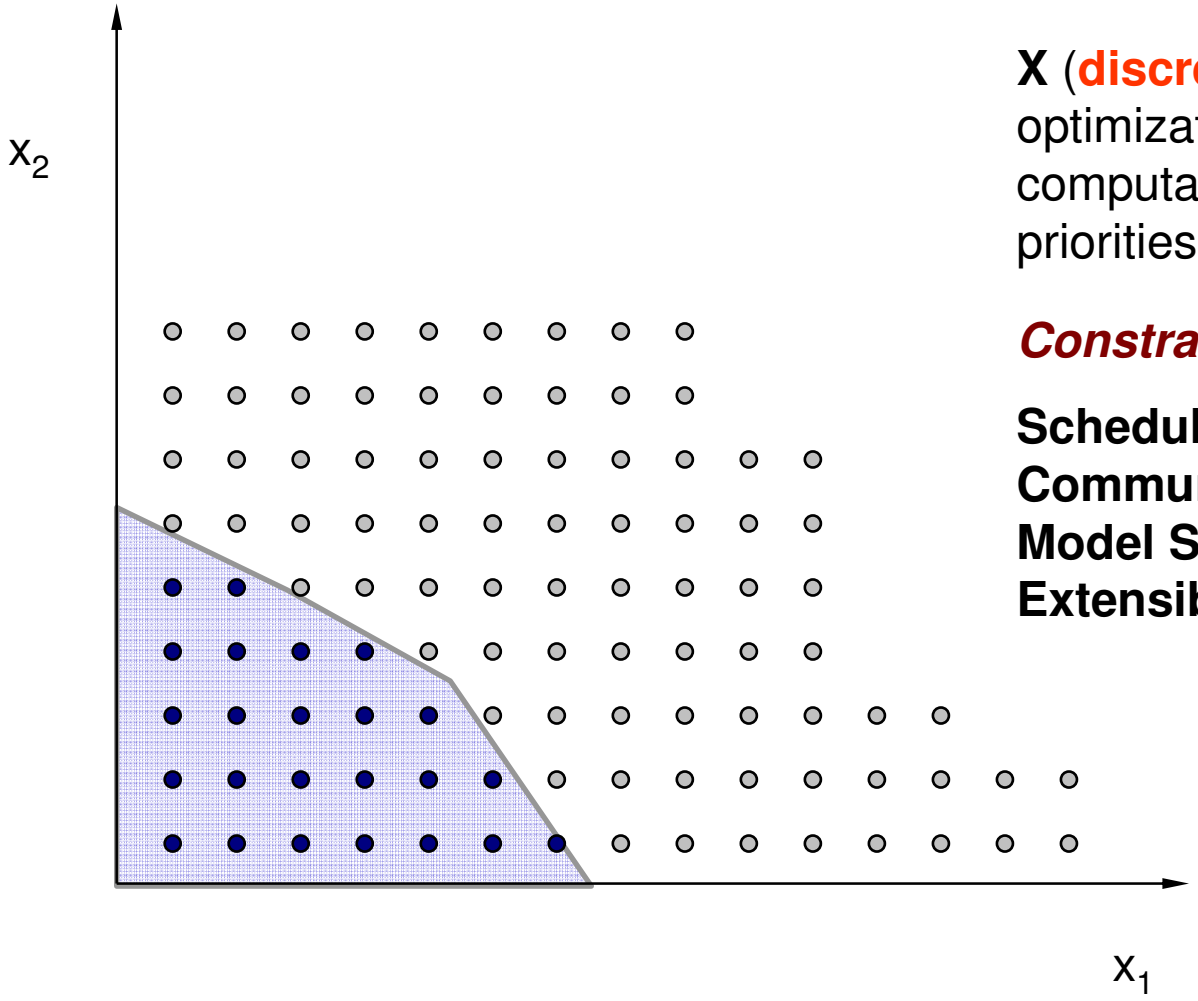
X space of design optimization variables, such as computation times, periods, placement, priorities …

Constraints

Schedulability
Communication
Model Semantics preservation
…

Design Process and Requirements

- Design optimization



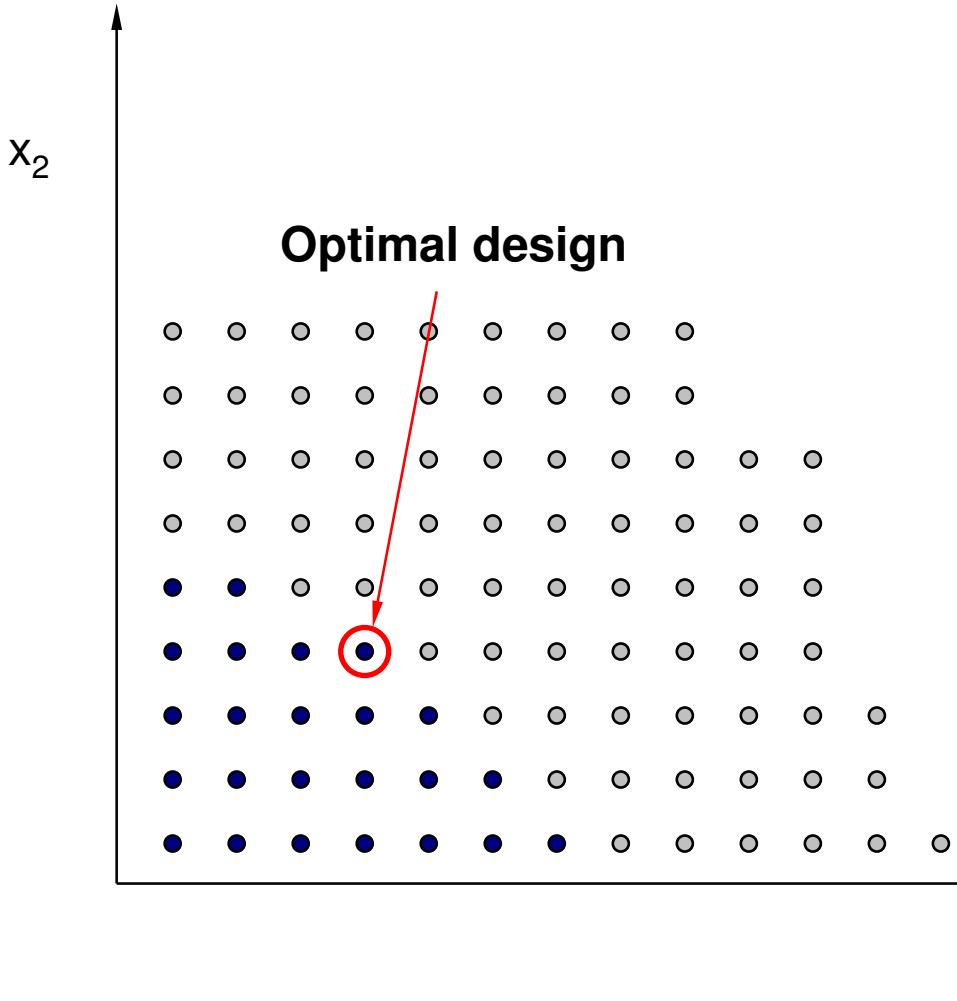
X (**discrete**) space of design optimization variables, such as computation times, placement, priorities, periods …

Constraints

Schedulability
Communication
Model Semantics preservation
Extensibility

Design Process and Requirements

- Design optimization



X (discrete) space of design optimization variables, such as computation times, periods …

Constraints

Schedulability
Communication
Model Semantics preservation
Extensibility

Metrics

Control related

(Example) Problem Formulation

Objective

Minimization of (average case) end-to-end latencies

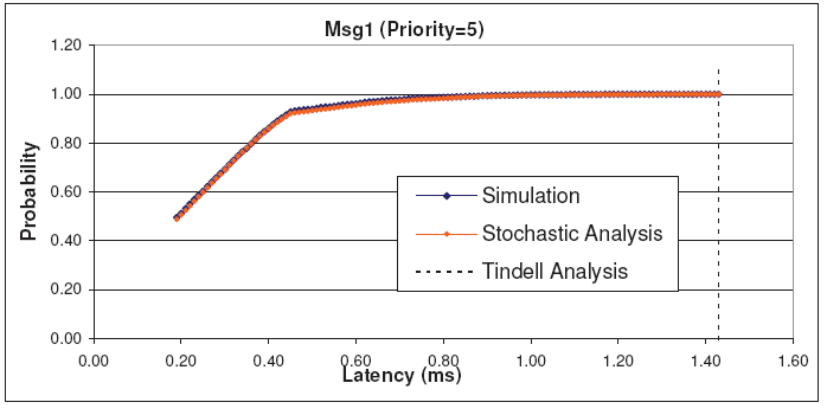
Subject to

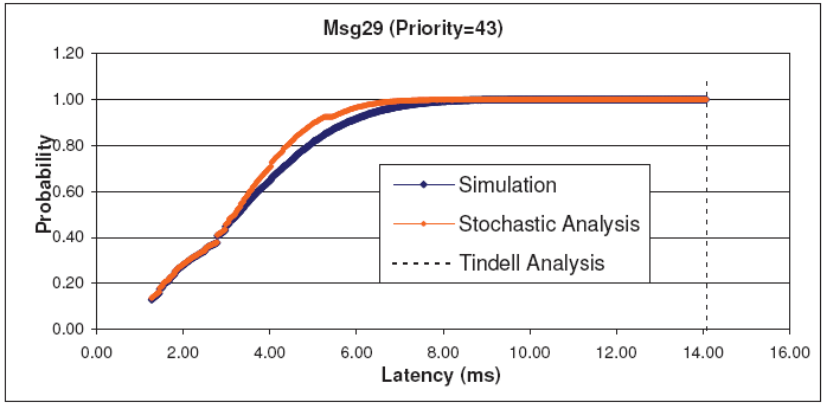
- Constraints on end-to-end latencies
- Constraints on messages size
- Constraints on utilization
- Constraints on message and task deadlines
- Semantics preservation constraints

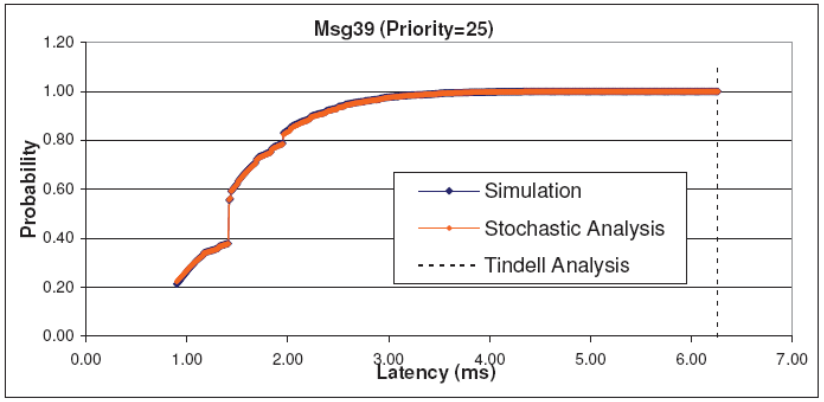
Design objectives (optimization variables)

- Placement of tasks onto the CPUs
- Packing of signals to messages
- Assignment of priorities to tasks and messages
- Definition of activation modes/synchronization model
- Period optimization

Stochastic analysis







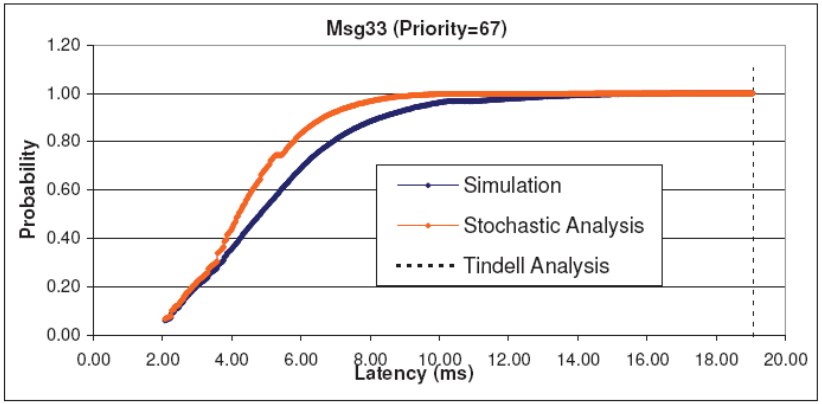


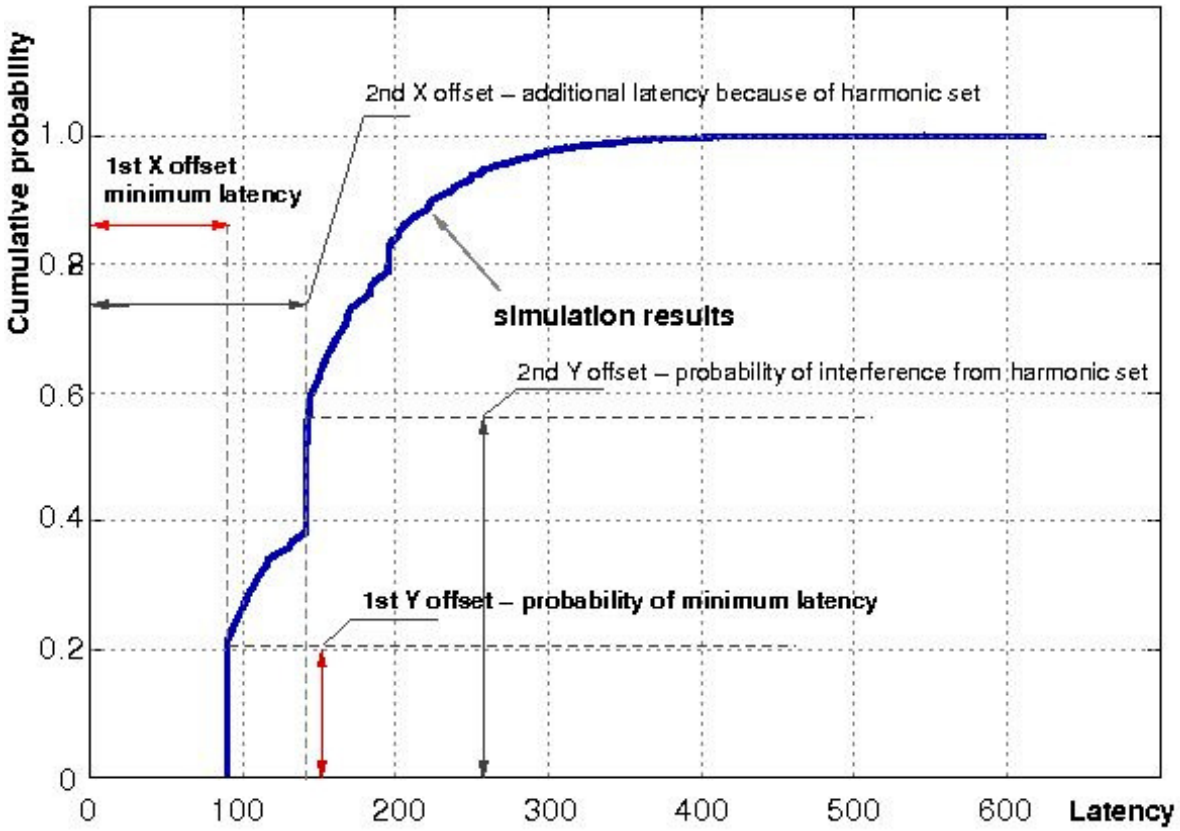
Figure 5. Latency *cdf*s of two high priority representative messages in the test set

Figure 6. Latency *cdf*s of two low priority representative messages in the test set

62 msg set (subset of chassis bus). Low priority msg – Distributions of latencies

Statistical analysis of CAN msgs

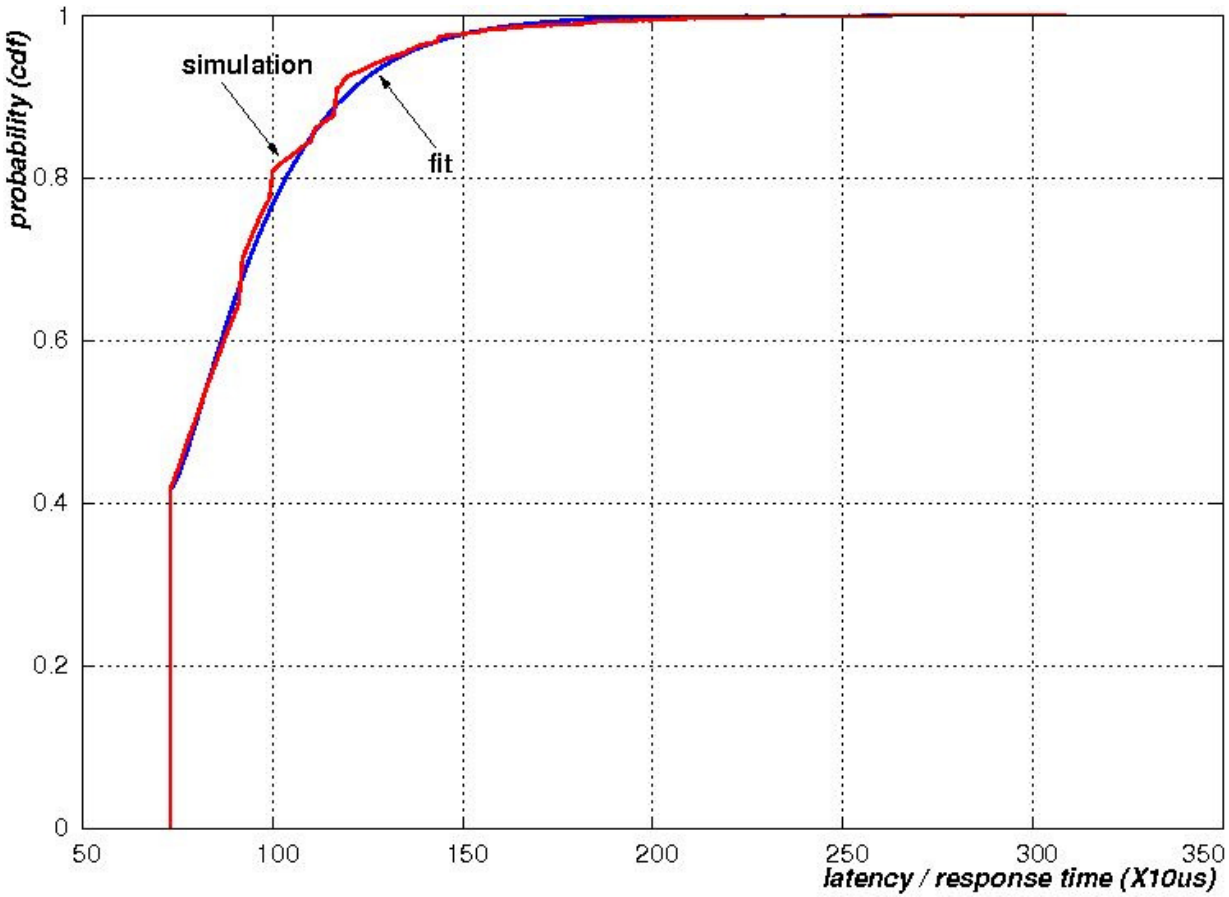
- Collected distributions of CAN message latencies by simulation on automotive buses (5 “realistic msgs configurations” and 20+ more obtained by derivation with changes in the load)



Typical shape of cdf

Statistical analysis of CAN msgs

- Can we fit the latency cdf with a “well-known” statistical distribution?
- What would be the accuracy?

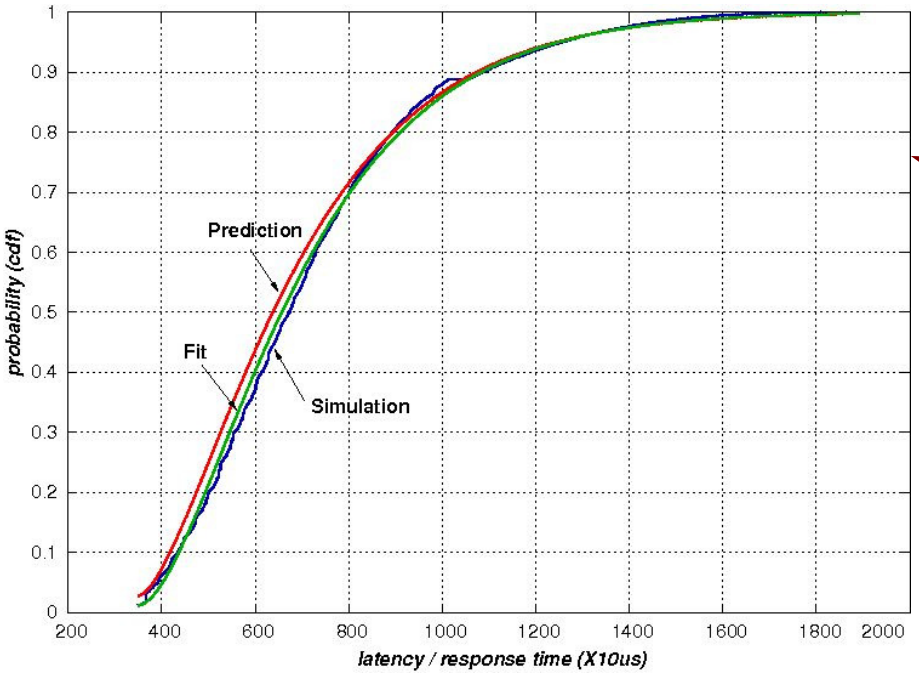


Fitting with a gamma distribution

An exponential fitting also returns good results!

Statistical analysis of CAN msgs

- Finally, can we estimate the offsets and the parameters of the Gamma distribution (a, b) or (μ,b) for each message by regression from parameters of the message set like U_i^r, U_i^{hr},Q_i, Q_i^{hr} ?



Example: medimum priority msg

Using regression formulas as predictors for X_{off}, Y_{off}, μ and b

Example: formula for μ

$$\mu_{i,k} = (Q_{i,k} + \beta_5)e^{\beta_6 + \beta_7 U_i^{hr}} + (Q_i^{hr} + \beta_8)U_i^{hr} e^{\beta_9 + \beta_{10} U_i^{hr}}$$

Conclusions

- Schedulability theory and worst-case timing analysis …
  - From the run-time domain to the design domain (already happening)
  - From the analysis domain to the optimization (synthesis) domain
  - Complemented by sensitivity analysis and uncertainty evaluation
- *However …*
  - *Typical deadline analysis is not enough!*
  - *Tasks and messages are not the starting point (semantics preservation issues from functional models to tasking models)*
  - *Worst case analysis needs to be complemented*
  - *Mixed domains (time-triggered / event-triggered)*

Q&A

Thank you!