Time and Schedulability Analysis of Stateflow Models

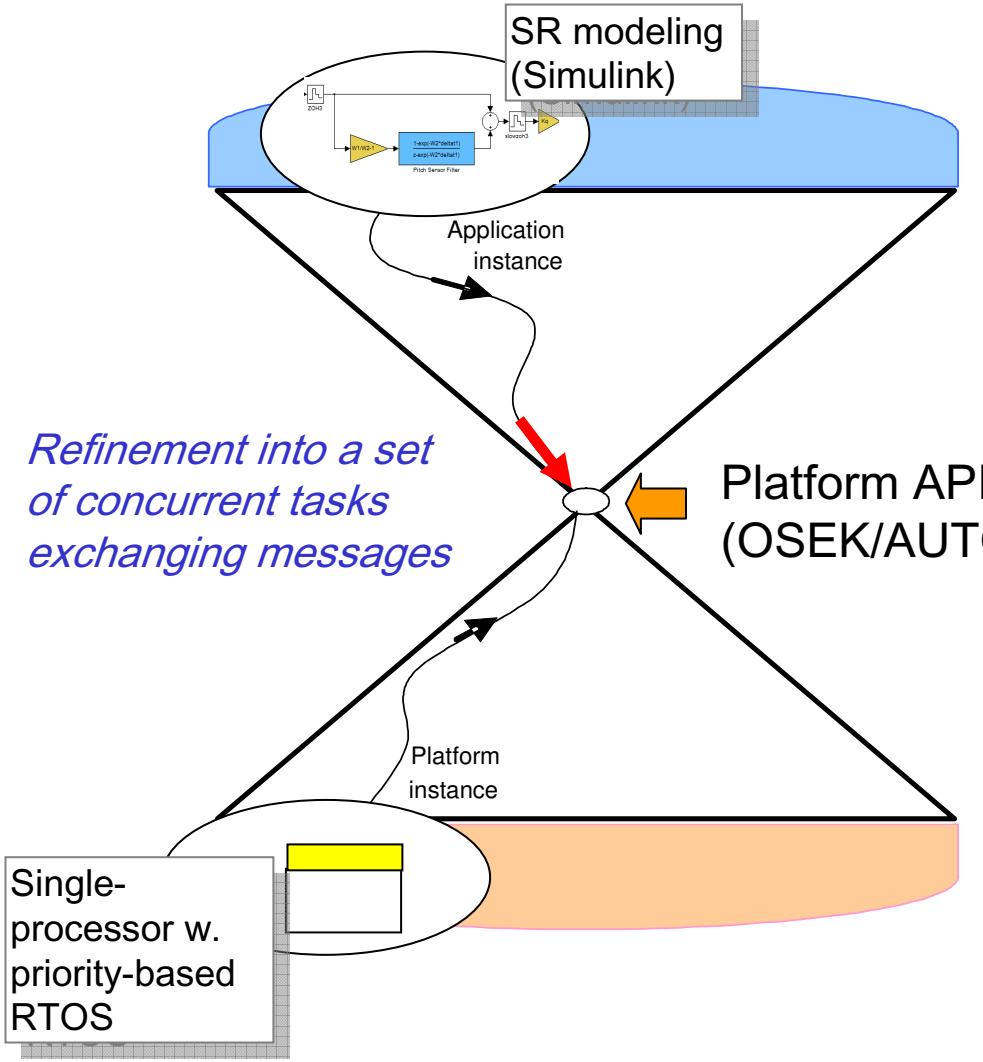
Marco Di Natale
Scuola Superiore S. Anna

Haibo Zeng
Mc Gill University

Outline

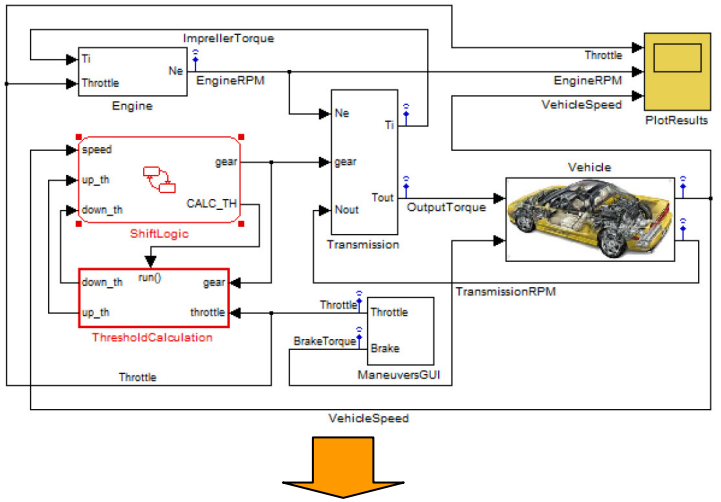
- Context: MBD of Embedded Systems
  - Relationship with PBD
- An Introduction to Schedulability Analysis
  - Independent tasks – demand-based function analysis
  - Tasks with offsets
  - Task graphs
- An Introduction to Stateflow Semantics
- Tasking Model
- The Scheduling Problem
- Speeding up the analysis with max-plus algebra
- Future steps

Context and relationship with MBD



Refinement into a set of concurrent tasks exchanging messages

Platform API
(OSEK/AUTOSAR)

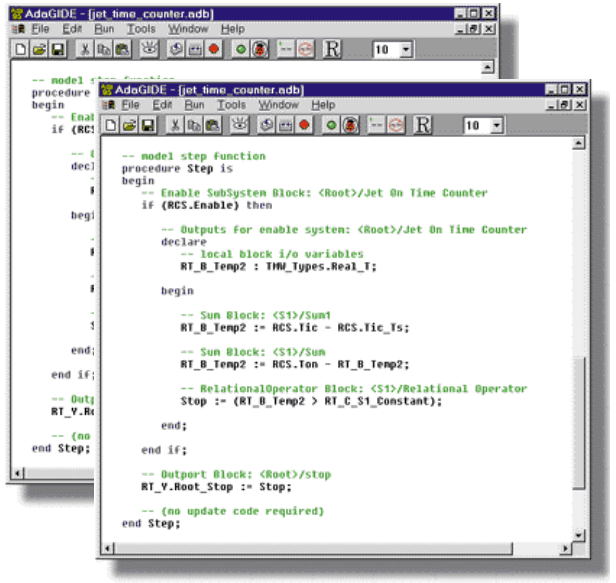


Analysis

Task and
communication model

Schedulability analysis: Independent periodic tasks

- First model (Late 70s/80s)
- Study of ADA programs
- Developments into early RTOS systems



- Activation events are periodic (period=T),
- Deadlines are timing contraints on the execution of tasks (D=T)
- Tasks are independent
- The execution time of each task is modeled by the worst case execution time (worst-case scenario)

- n tasks τ_1, τ_2, … τ_n
- Task periods T_1, T_2, ... T_n
- Execution times are C_1, C_2, ... C_n

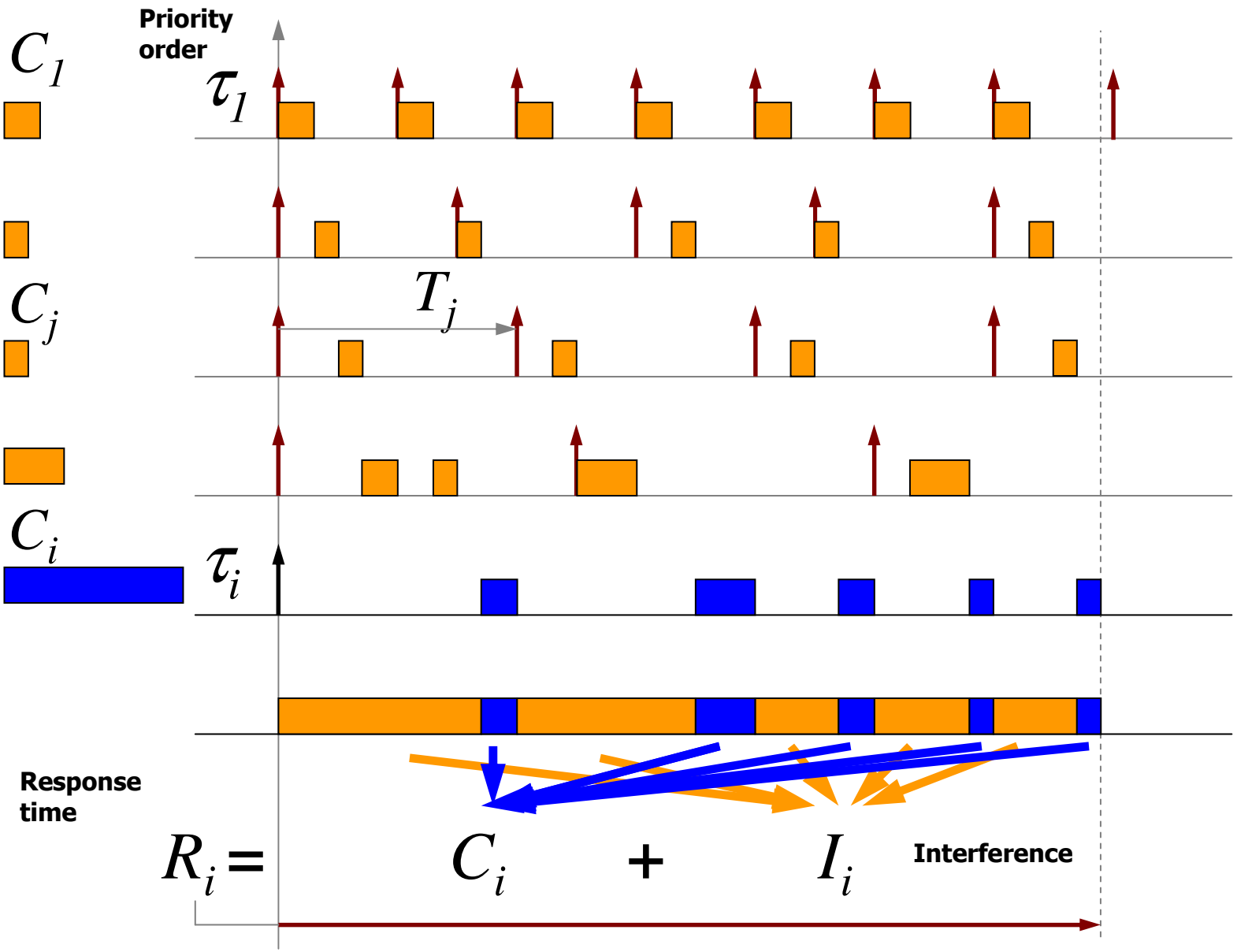
Scheduling algorithm

- Preemptive & priority driven
  - task have priorities
    - statically (design time) assigned
  - at each time the highest priority task is executed
    - if a higher piority task becomes ready, the execution of the running task is interrupted and the CPU given to the new task
- In this case, scheduling algorithm = priority assignment + priority queue management

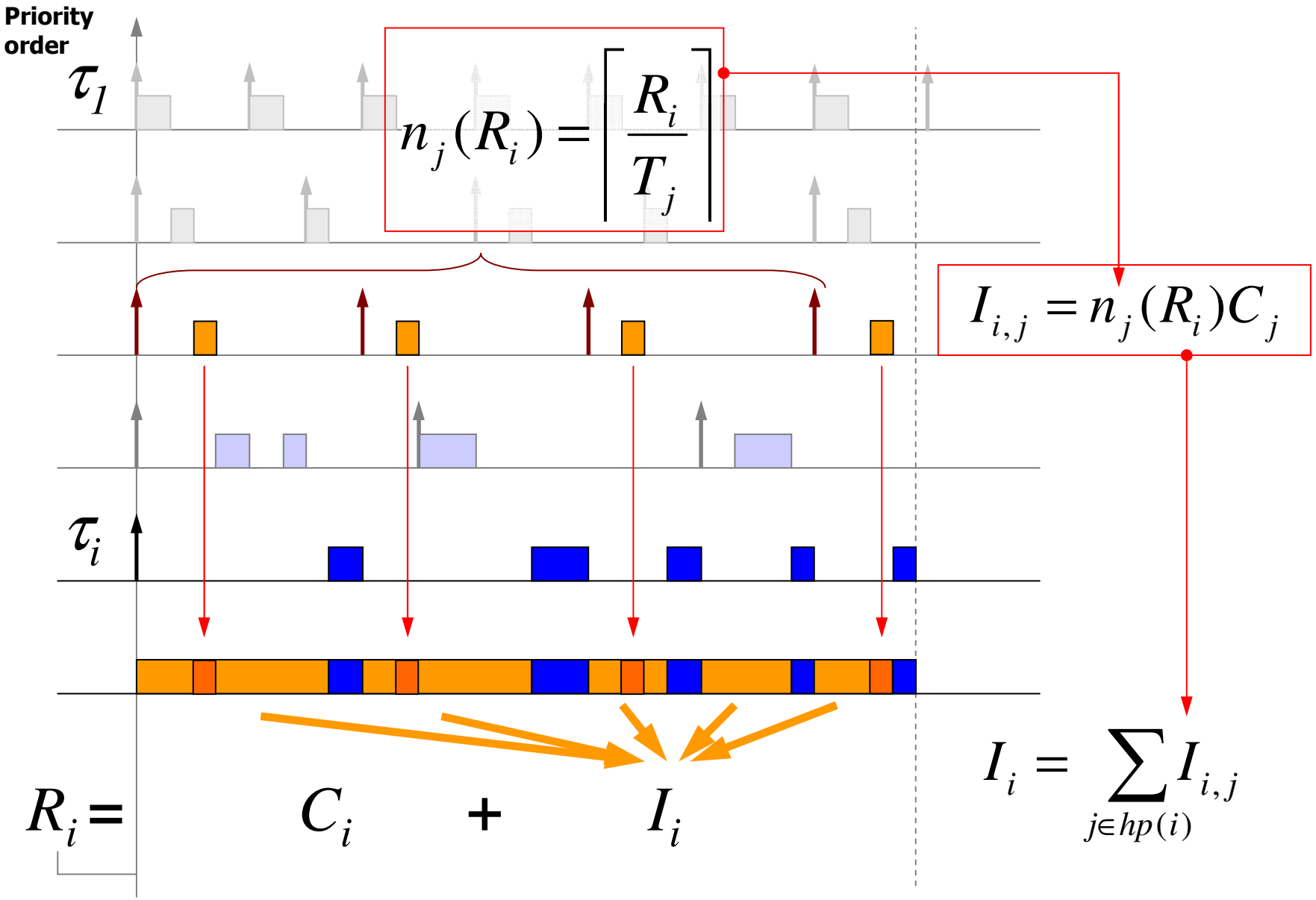
Critical instant and Worst-case response time

- Critical instant of a task = time instant t_0 such that, if the task instance is released in t_0, it has the worst possible response (completion) time (Critical instant of the system)
- *Theorem: the critical instant for each task is when the task instance is released together with (at the same time) all the other higher priority instances*
- The critical instant may be used to check if a priority assignment results in a feasible scheduling
  - if all requests at the critical instant complete before their deadlines

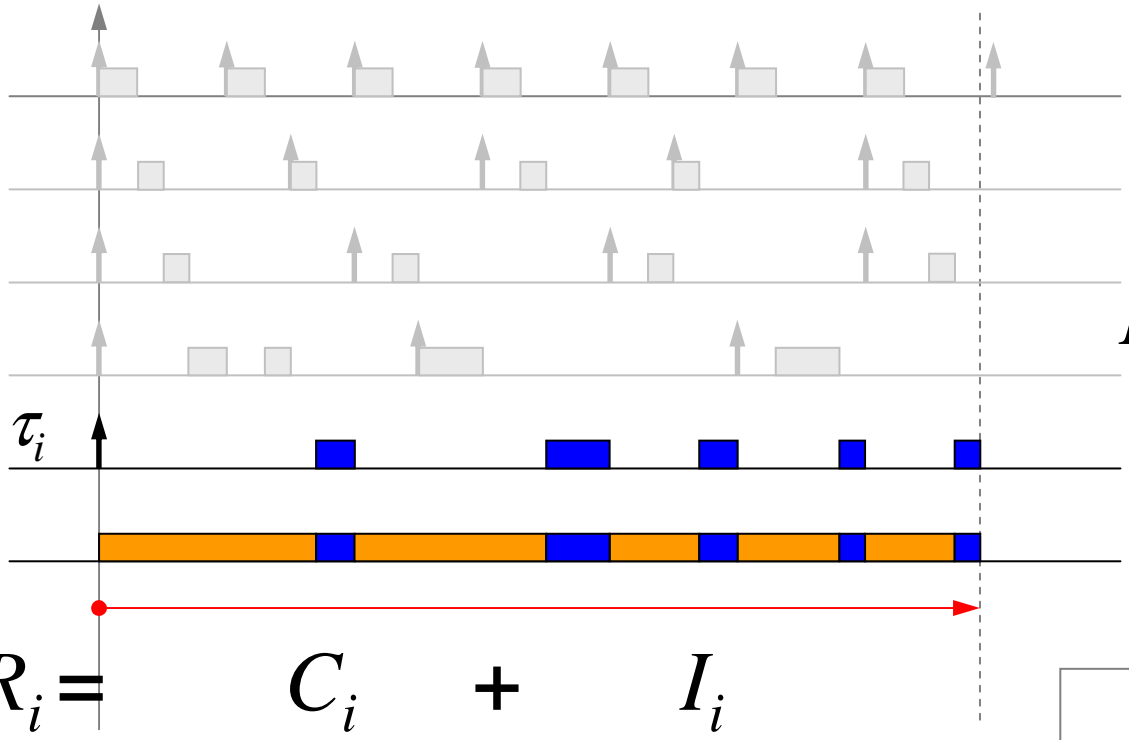
Response Time Analysis (contd.)



Response Time Analysis (contd.)



Response Time Analysis (contd.)



$$R_i = C_i + I_i$$

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

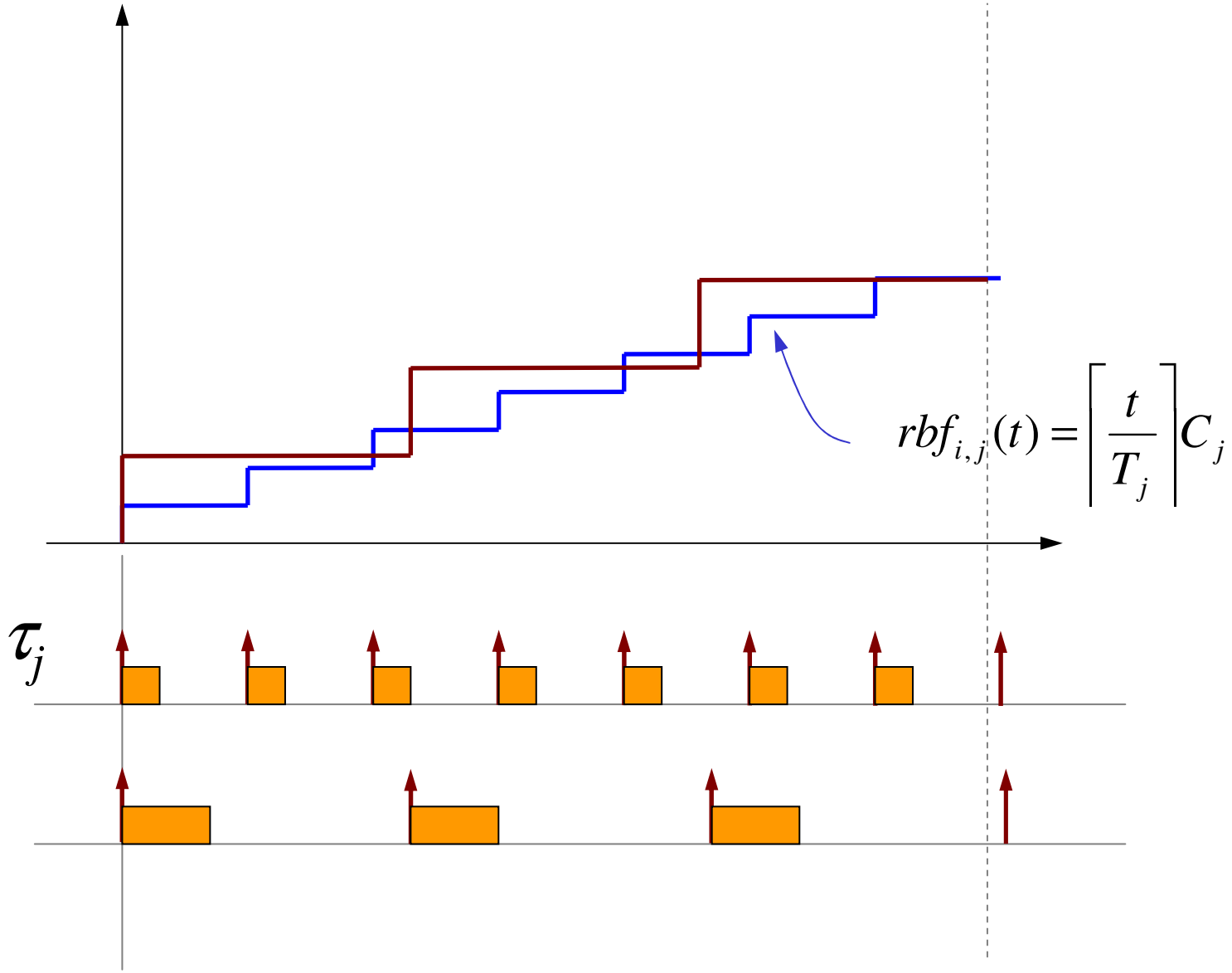
$$rbf_i(t) = C_i + \sum_{j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

Request bound function of level i

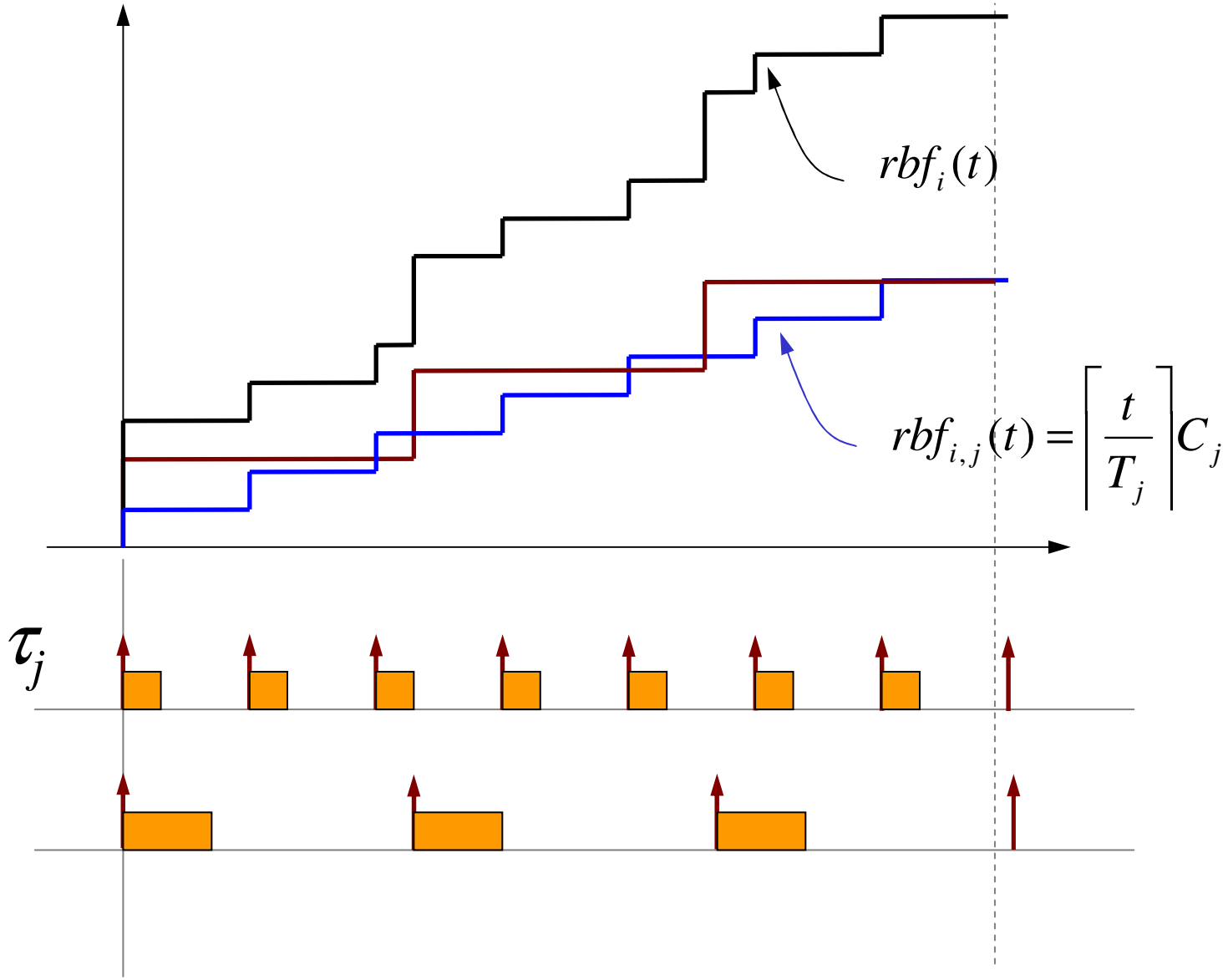
$$rbf_i(t) = \sum_j dbf_{i,j}(t)$$

$$rbf_{i,j}(t) = \left\lceil \frac{t}{T_j} \right\rceil C_j$$

Response Time Analysis (contd.)



Response Time Analysis (contd.)



Scheduling with Offsets

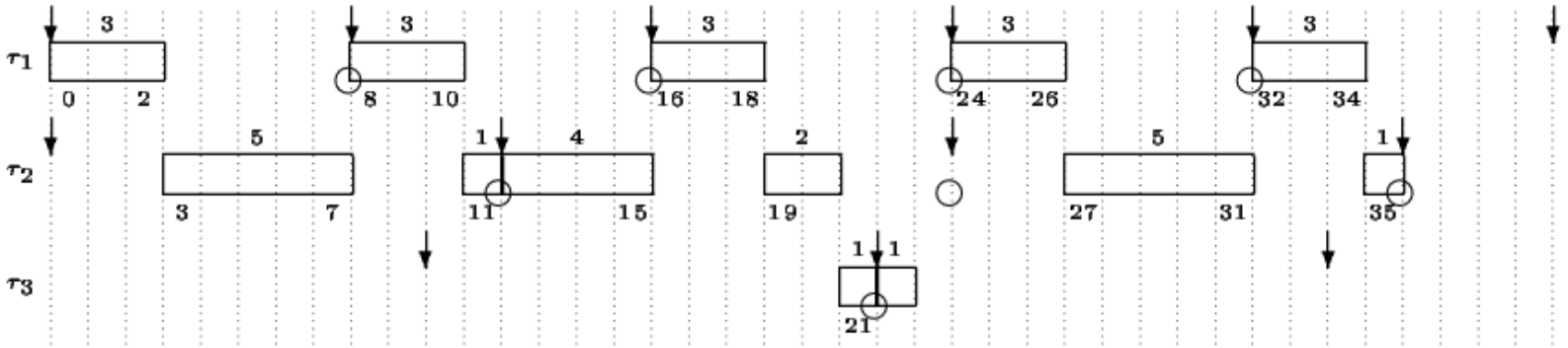
- Offset free model: the values of the O_i are not defined (analysis in the worst-case/critical instant)
- Synchronous model: O_i may be $\neq 0$, but the values are given
  - Critical instant may never happen (pessimistic assumption)

Scheduling with Offsets

- Example: consider the case (C, D, T)

τ_1=(3, 8, 8), τ_2=(6, 12, 12), τ_3=(1, 12, 12)

- Not schedulable in the offset-free model (not even with RM assignment), given that task 3 misses its deadline.
- But is schedulable with O_1=0, O_2=0, O_3=10 !



Scheduling with Offsets

Feasibility test for task sets with offsets:

- Given a set of offsets O_1, O_2, ... O_n
- [Leung82] a task set is feasible if all deadlines are met in [s, 2P], where s=max{O_1, O_2, ... O_n} and P = lcm {T_1, T_2, ... T_n}
  - in practice it is sufficient to build the schedule and check all the busy periods originating from a task release time in [s, 2P)
- For fixed priority tasks and D≤T it is possible to further restrict the interval [Audsley91]
- Theorem: Let S_i be inductively defined by

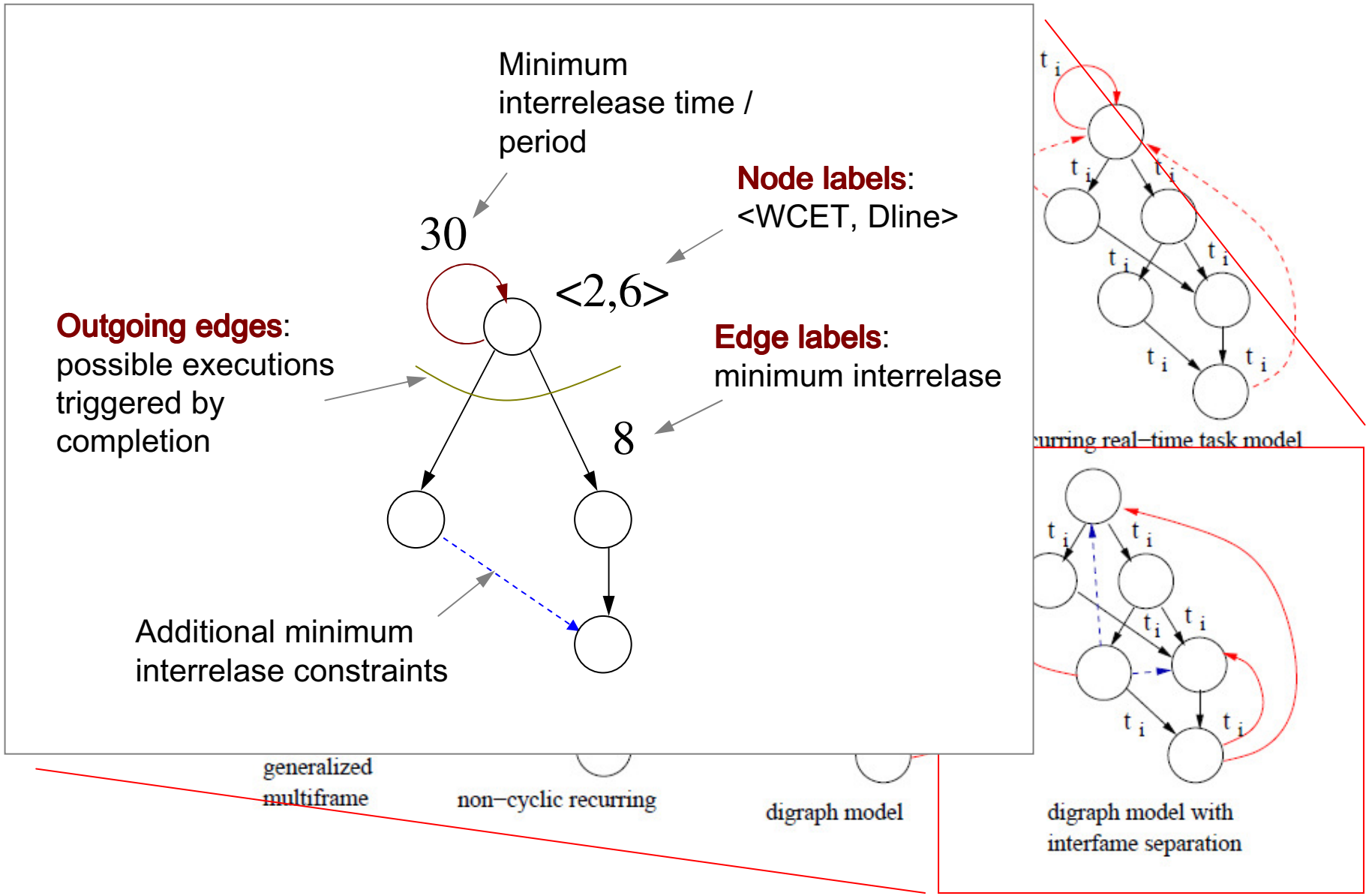
$$S_1 = O_1$$

$$S_i = \max\{O_i + \left\lceil \frac{S_{i-1} - O_i}{T_i} \right\rceil T_i$$

Then, if the task set is ordered by decreasing priorities and has a feasible schedule, it is periodic from S_n with period P = lcm{T_i st: i=1, 2, …, n}

It is sufficient to check al the busy periods in the interval [S_n, S_n+P])

Synchronous FSM vs digraph task models



Synchronous FSM vs digraph task models

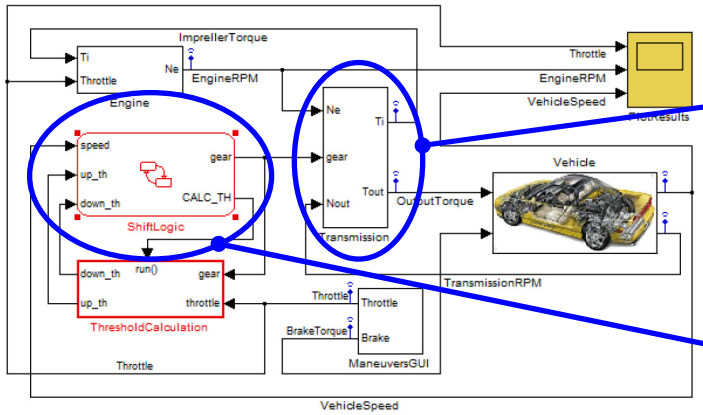
- *The request bound function* (or *rbf*) and *demand bound function* (or *dbf*) are a general tool for the analysis of task graphs.

> *Definition 6:* The *request bound function* of an FSM $\mathcal{F}$ during a time interval $\Delta = [s, f)$, (s *inclusive* and f *exclusive*), denoted as $\mathcal{F}.rbf(\Delta)$, is the maximum sum of execution times by the actions of $\mathcal{F}$ that have their activation time within Δ.
>
> *Definition 7:* The *demand bound function* of an FSM $\mathcal{F}$ during the time interval $\Delta = [s, f]$ (both s and f are included in the interval), denoted as $\mathcal{F}.dbf(\Delta)$, is the maximum sum of execution times by the actions of $\mathcal{F}$ that have their activation time *and* deadline within Δ.

- See the analysis in
  M. Stigge, P. Ekberg, N. Guan, , and W. Yi, "The digraph real-time task model," in *Proc. the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011.

Functional representation: SR Simulink modeling

- Functional model: “zero-logical-time” execution or (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
  - ***“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.



Regular type blocks

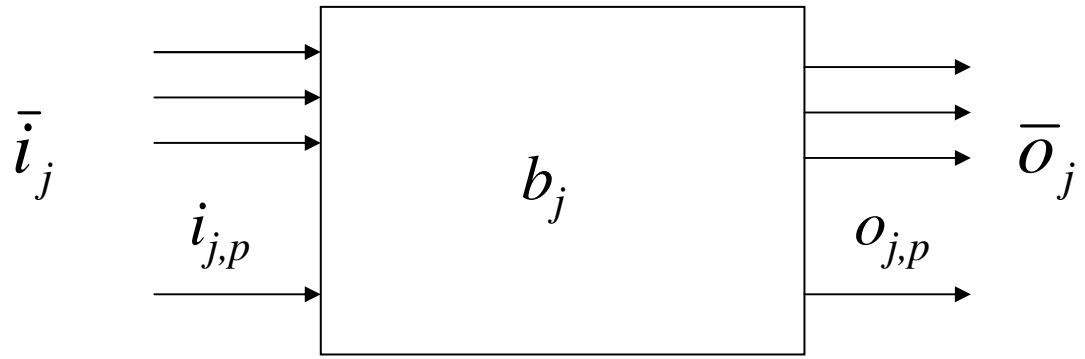
Stateflow (state machine)-type blocks

Functional representation: SR Simulink modeling

- Simulink system = networks of blocks

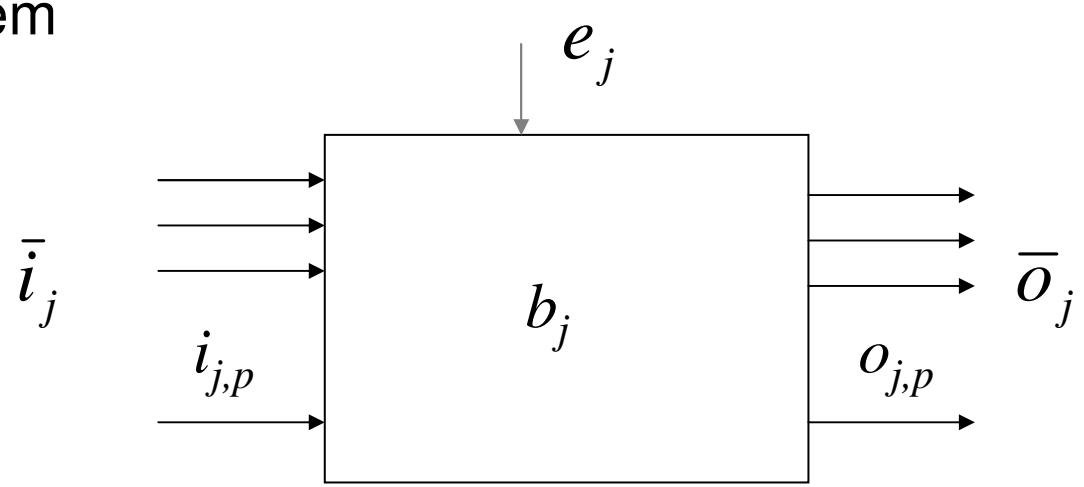
$$S = \{b_1, b_2, ..., b_n\}$$

- Blocks can be Regular or Stateflow blocks
- Regular blocks can be Continuous of Discrete type.
- All types operate on (right)continuous type signals.
- Blocks may have a state S_j or may be stateless.



Functional representation: SR Simulink modeling

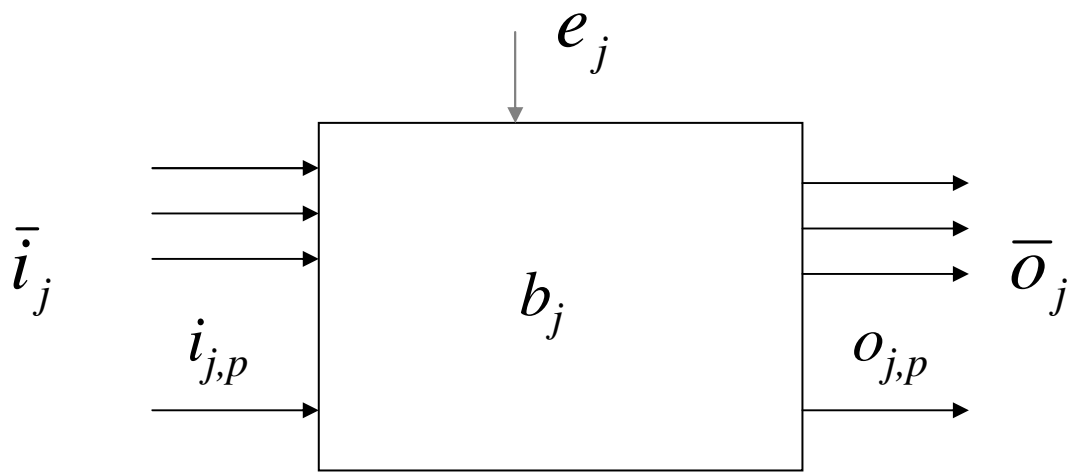
- Continuous-type blocks are defined by a set of differential equations
- Discrete-type blocks are activated at events e_j belonging to a periodic sequence with 0 offset and period T_j
- When a model generates code, continuous blocks must be implemented by a fixed-step solver, with period *Tb*
- T_b (*base period*) must be a divisor of any other T_j in the system



Functional representation: SR Simulink modeling

- At each e_j the block computes its out update and state update functions, updating the values on its output signals

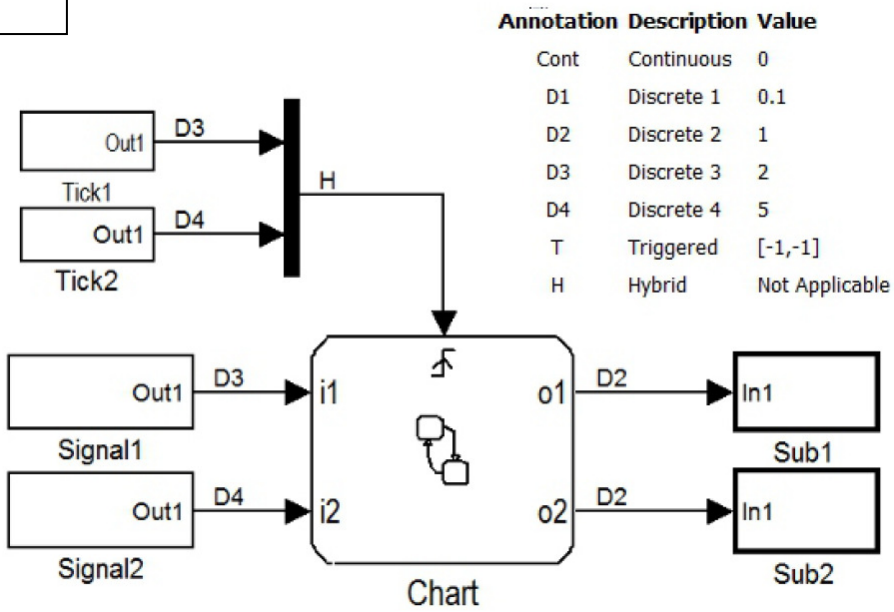
$$S_j^{new}, \bar{o}_j = f(S_j, \bar{i}_j)$$



Simulink models (execution order - feedthrough)

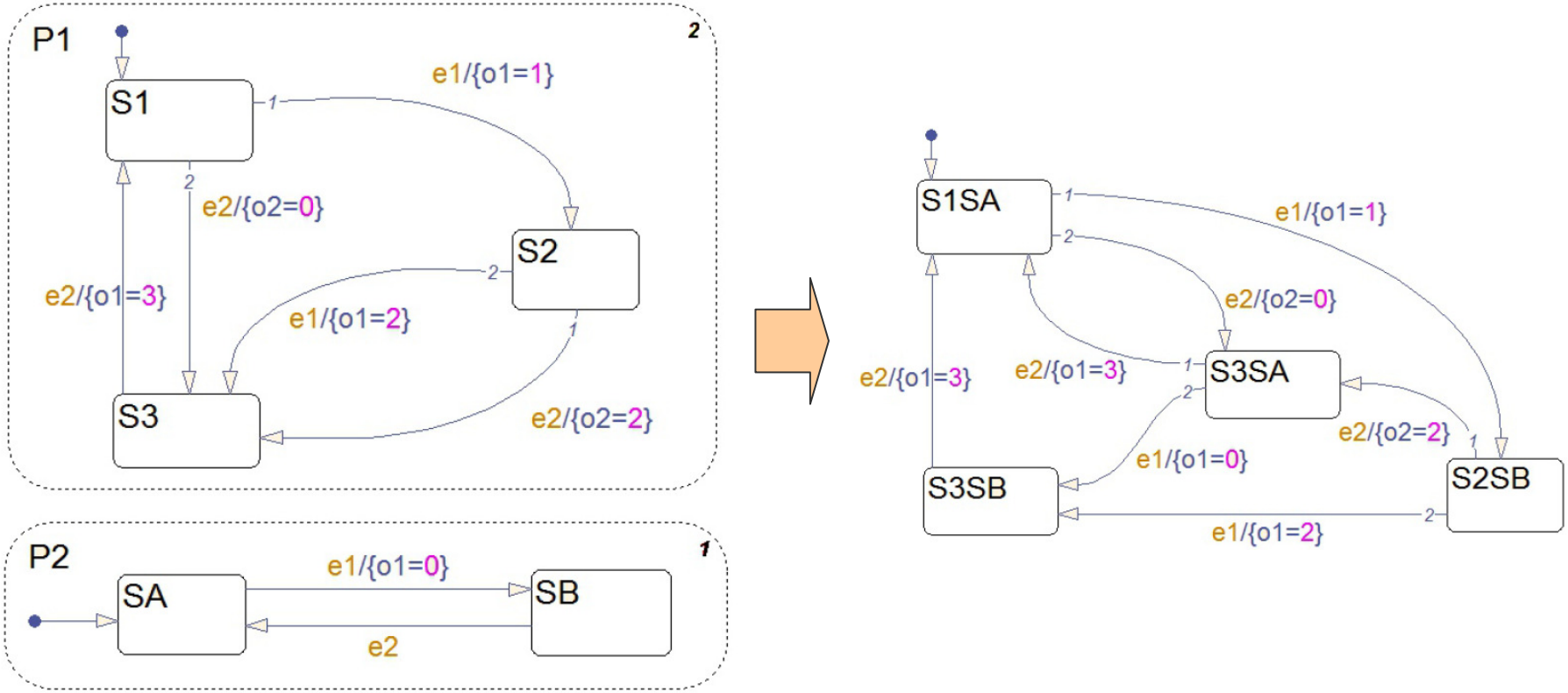
Stateflow (or state machine) blocks react to a set of events $e_{j,v}$, derived from signals (generated at each rising or falling edge).

As such, events belong to a set of discrete time bases kT_{jv}



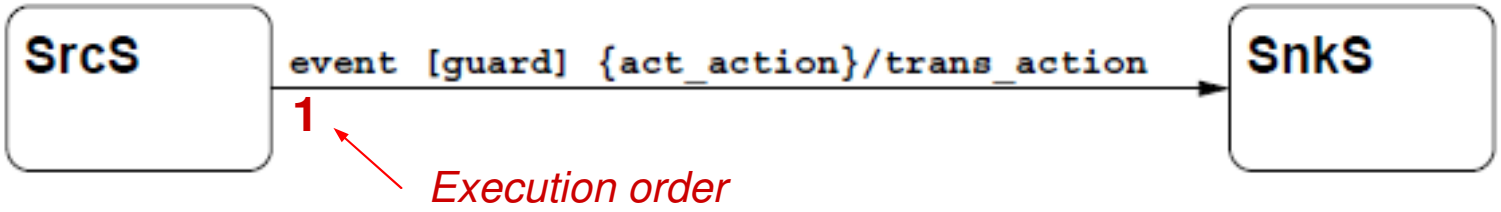
Simulink models (execution order - feedthrough)

Stateflow machines are extended (synchronous) FSMs with hierarchical and parallel states

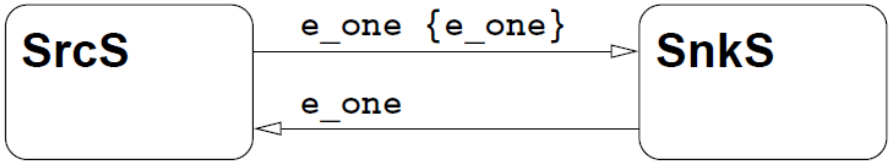


Simulink models (execution order - feedthrough)

Transition notation



And quite a few issues … (transition actions can generate events)

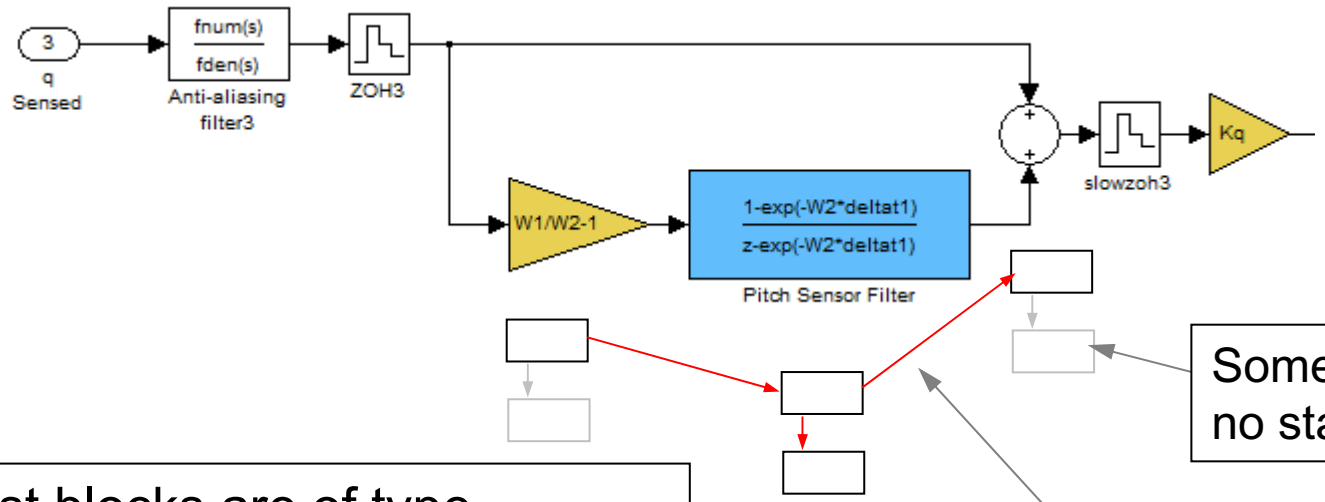


For more info:

N. Scaife, C. Sofronis, P. Caspi, S. Tripakis, and F. Maraninchi *Defining and translating a "safe" subset of Simulink/Stateflow into Lustre*. 4th ACM International Conference on Embedded Software (EMSOFT04), Pisa, Italy, September 2004

A. Tiwari. *Formal semantics and analysis methods for Simulink Stateflow models*. Technical report, SRI International, 2002.

Simulink models (execution order - feedthrough)



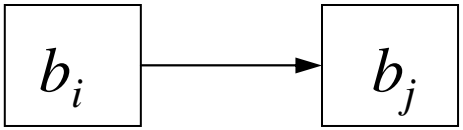
Most blocks are of type feedthrough or Mealy-type (output does depend on input)

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

Some blocks have no state

Dependencies among outputs

Simulink models (execution order - feedthrough)



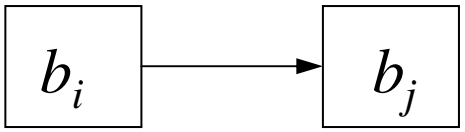
If two blocks b_i and b_j are in an input-output relationship (one of the outputs of b_i is the input of b_j), and b_j is of type feedthrough), then

$$b_i \rightarrow b_j$$

In case b_j is not of type feedthrough, then the link has a delay,

$$b_i \overset{-1}{\rightarrow} b_j$$

Simulink models (execution order - feedthrough)



Let $b_i(k)$ represent the k-th occurrence of bi (belonging to the set $\cup_v kT_{i,v}$ if a state machine block, or kT_i if a standard block), a sequence of activation times $a_i(k)$ is associated to b_i.

Given $t \geq 0$, $n_i(t)$ is the number of times b_i is activated before or at t.

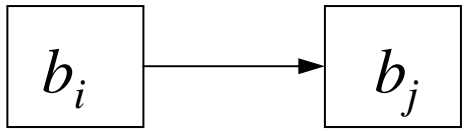
In case $bi \rightarrow bj$, if $i_j(k)$ is the input of the k-th occurrence of b_j , then this input is equal to the output of the last occurrence of b_i that is no later than the k-th occurrence of b_j

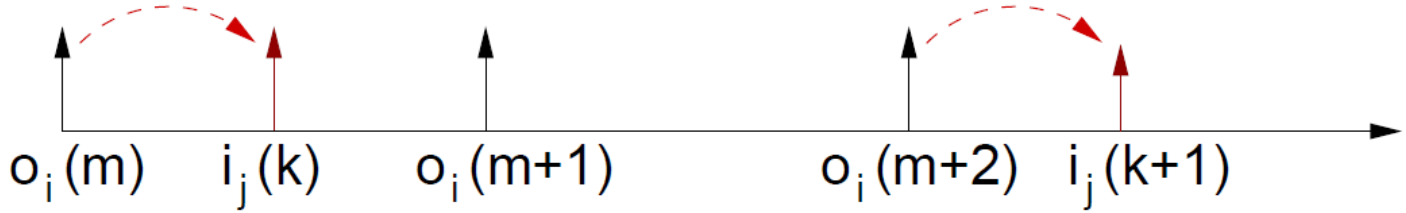
$i_j(k) = o_i(m)$; where $m = n_i(a_j(k))$

If *the link has a delay* , then the previous output value is read,

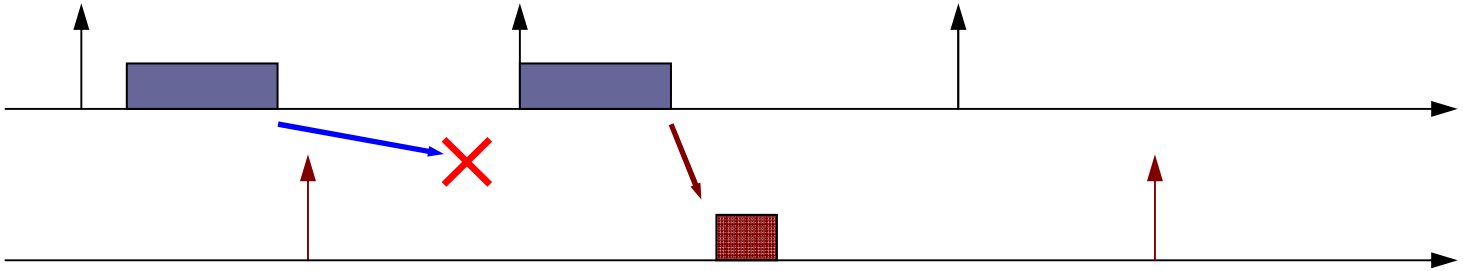
$i_j(k) = o_i(m - 1)$:

Simulink models (execution order - feedthrough)



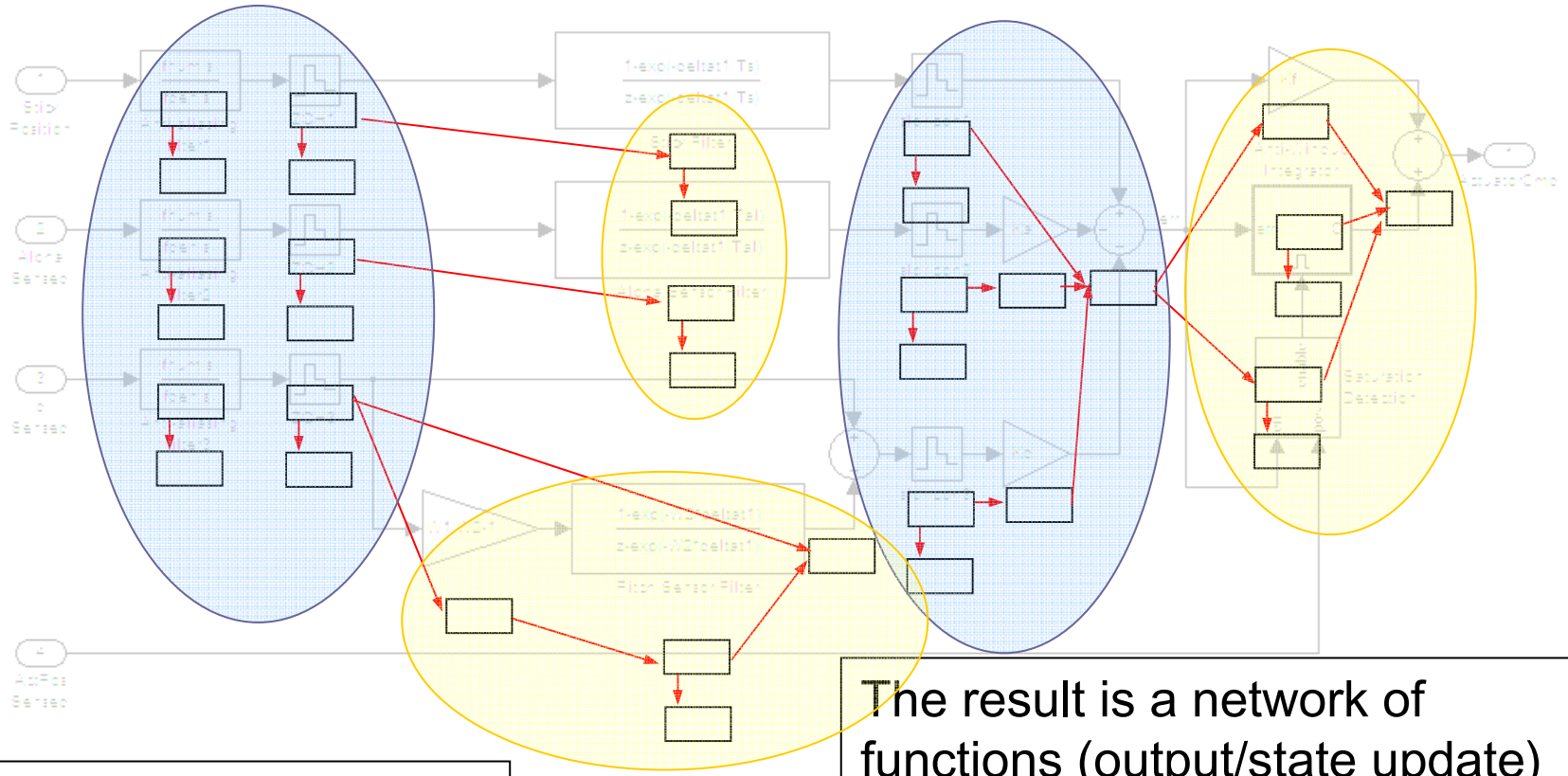


May be a problem in a code implementation with (scheduling) delays



Simulink models: rates and deadlines

the system response or reaction must be completed before the next system event



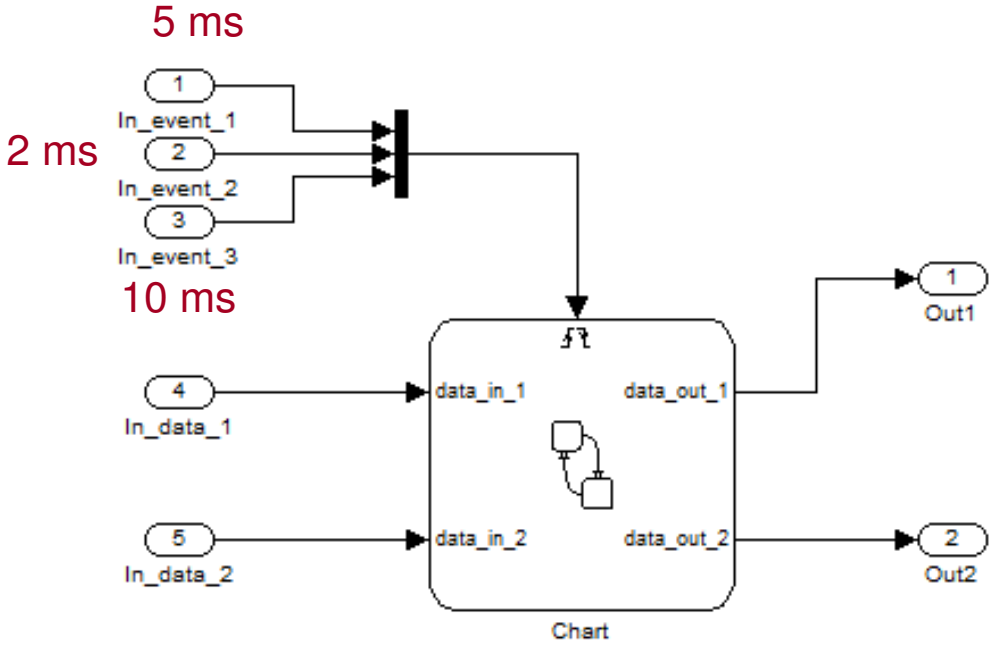
The result is a network of functions (output/state update) with a set of partial orders

Each blockset is characterized by an execution rate

Stateflow blocks: single task implementation

Each stateflow block is implemented by a periodic task executed at the period of the gcd of its input signals.

(simplest option, implemented by commercial code generators)



There could be alternative (multitask) options for semantics-preserving implementations of Stateflow FSMs

The system hyperperiod H depends on the periods of the regular blocks (tasks implementing them) and the period of other stateflow blocks in the system. The offsets are the individual periods of the blocks

Schedulability

Schedulability Analysis for Task *i*

FOR each priority level-*i* busy period *[s,f)*

IF $\exists t \in [s,f), \forall\, t' \in [s,t]$ such that

$\tau_i.dbf[s,t] + \sum_{j \in hp(i)} \tau_j.rbf[s,t') > t' - s$

THEN return **unschedulable**

ENDFOR

Return **schedulable**

The busy periods to be checked start at the release of a task with priority higher than or equal to τ_i.

Problems

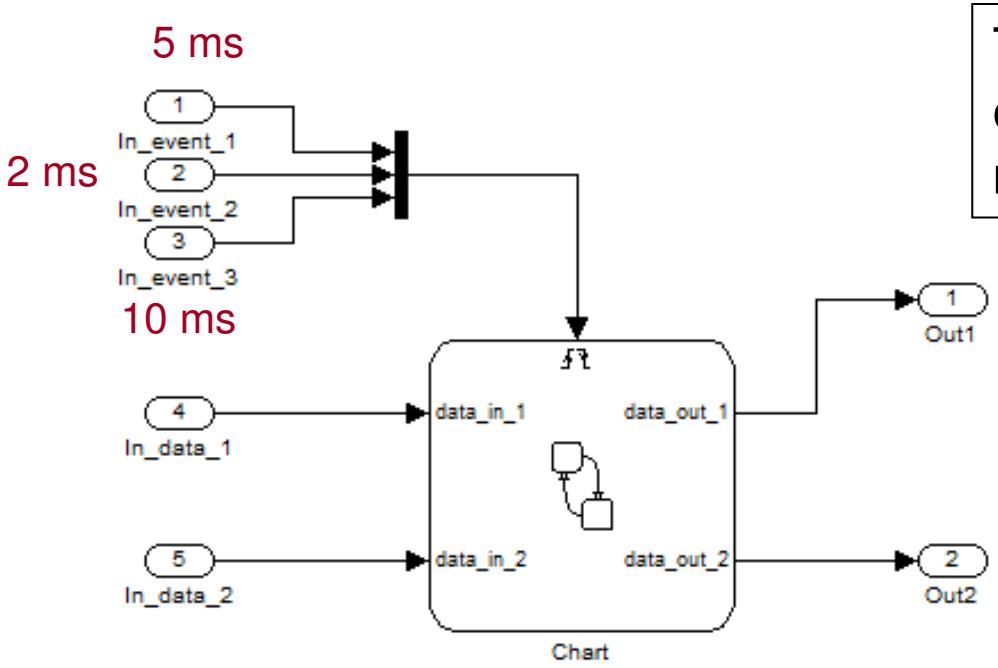
- The number of busy periods to be checked can be very large (all activation times in a hyperperiod)
- The busy period can be quite long making the computation very expensive

Steps

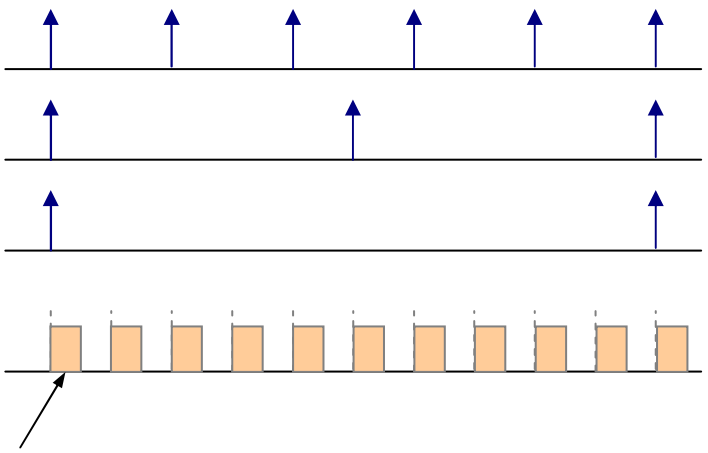
- Connect FSM analysis to recurrent task graphs
- *Improve and speed-up the analysis*

Simulink models: rates and deadlines

the system response or reaction must be completed before the next system event – if we abstract from the model and only look at the task code model, the analysis can be very pessimistic



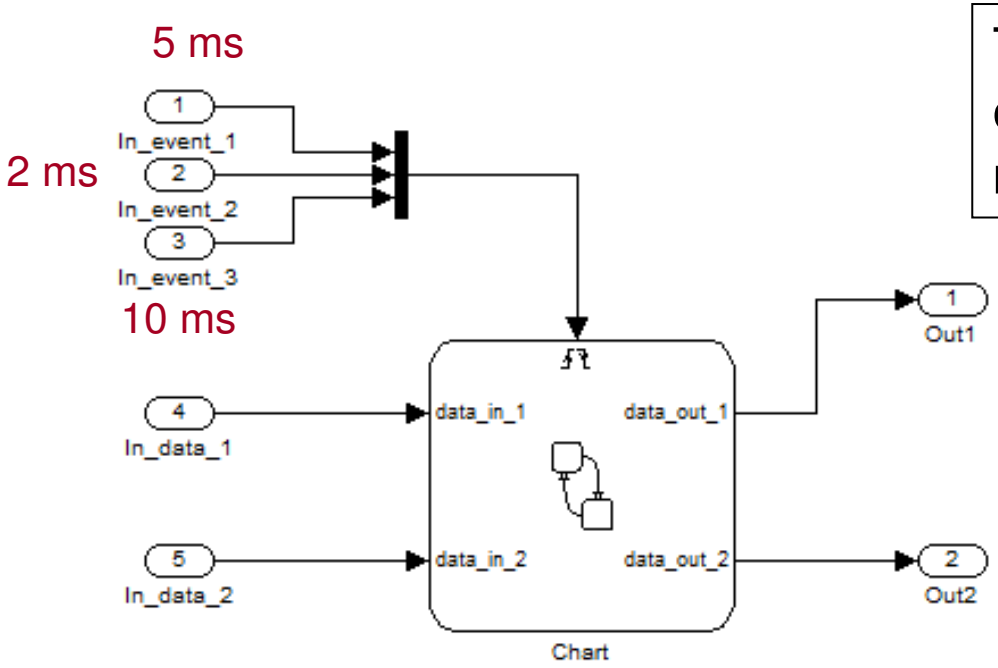
The generated code runs in the context of a 1 ms task, but reactions do not occur every 1ms



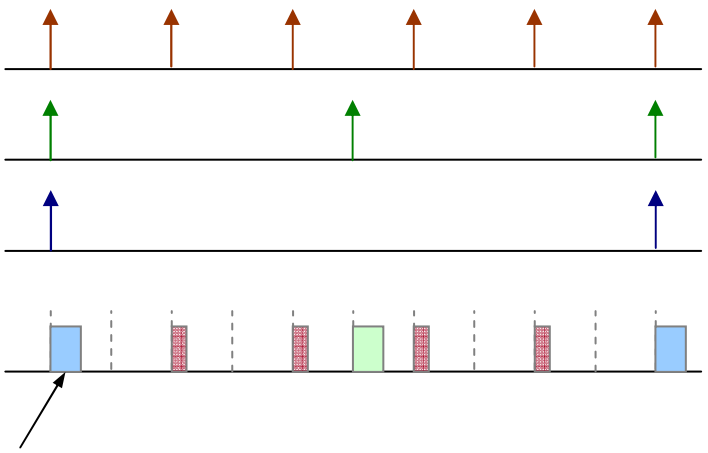
Simple periodic model: Worst-case exec time of a reaction to any event

Simulink models: rates and deadlines

the system response or reaction must be completed before the next system event – if we abstract from the model and only look at the task code model, the analysis can be very pessimistic



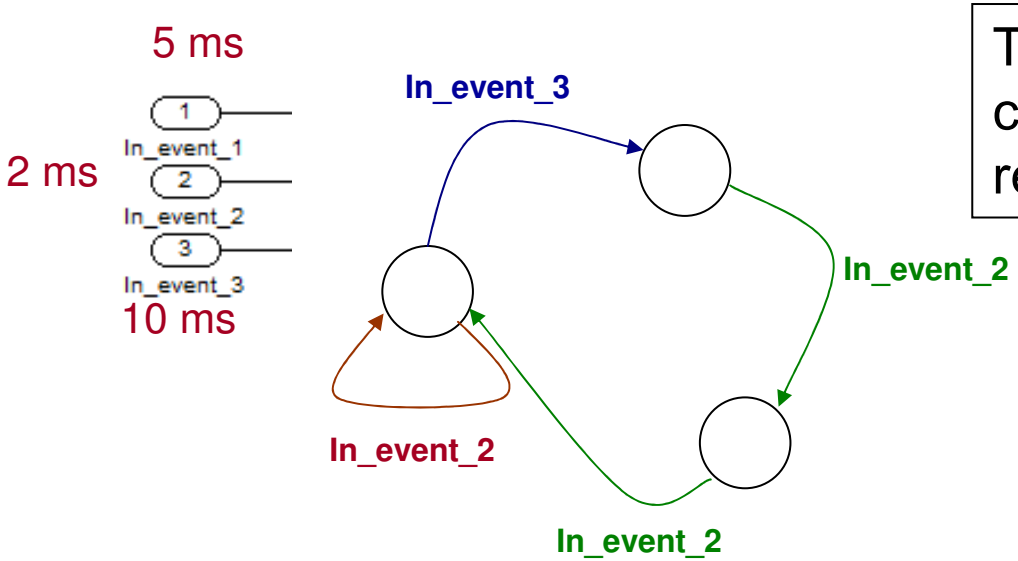
The generated code runs in the context of a 1 ms task, but reactions do not occur every 1ms



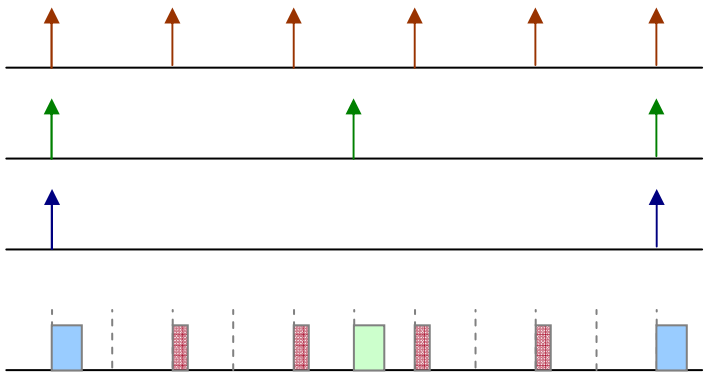
Multiframe model: worst-case exec time of a reaction to each type of event

Simulink models: rates and deadlines

the system response or reaction must be completed before the next system event – if we abstract from the model and only look at the task code model, the analysis can be very pessimistic

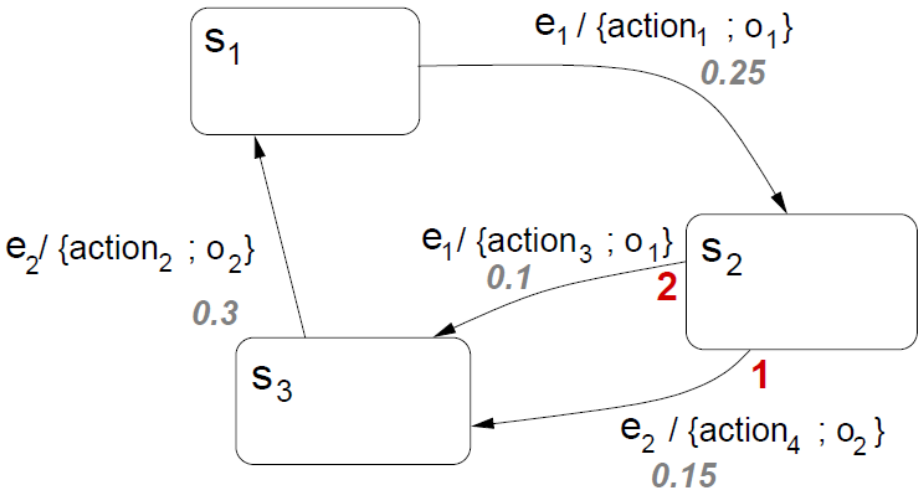


The generated code runs in the context of a 1 ms task, but reactions do not occur every 1ms

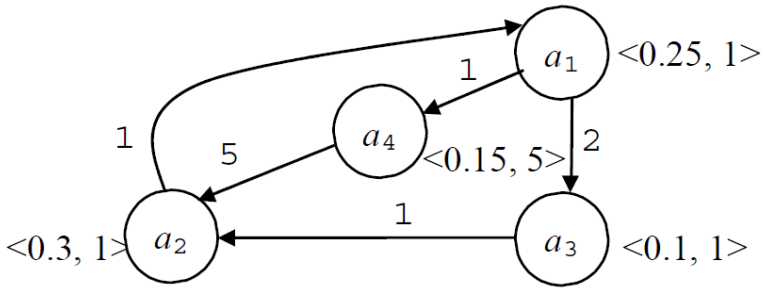


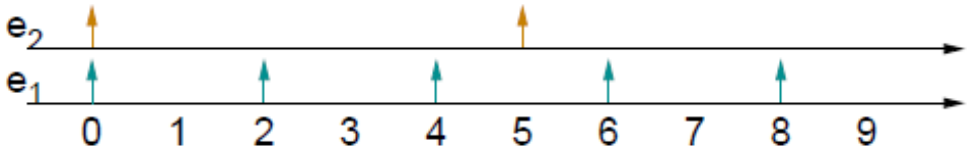
Even a multiframe model should account for state dependencies in the evaluation of the worst case execution time of each frame – need to build the right demand bound function

From Synchronous FSM to digraph task models



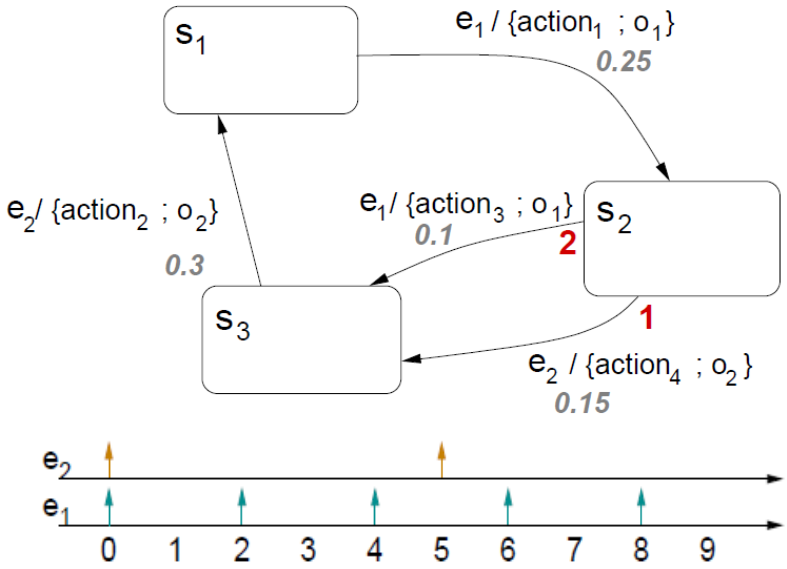
reactions → tasks

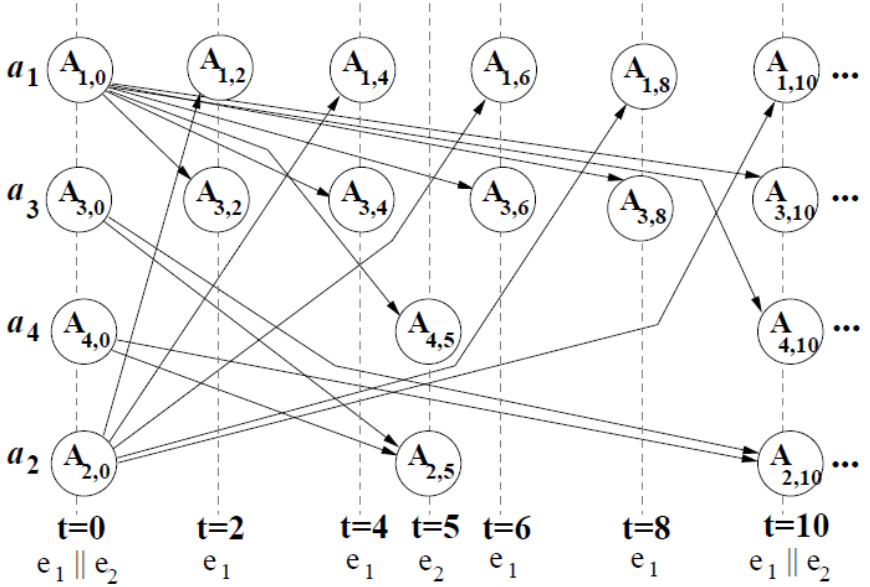




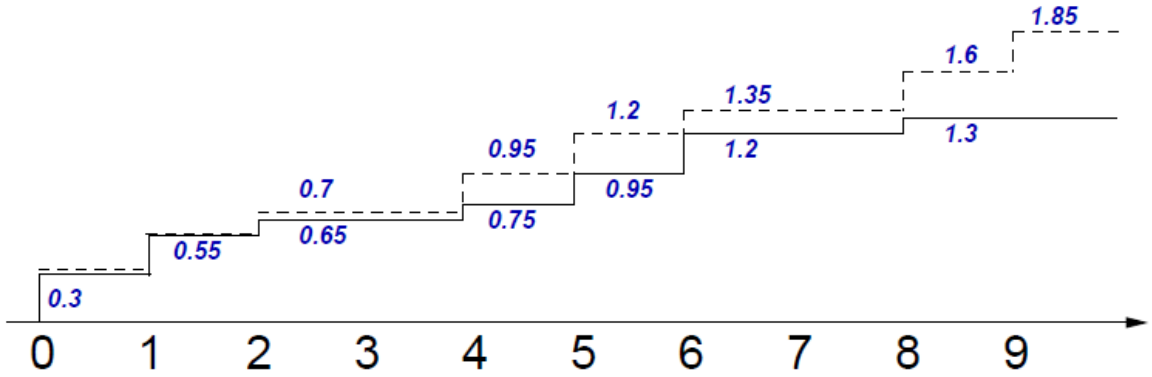
The task model is pessimistic. It allows the activation of a_2, a_1, and a_4 within $2ms$, which implies that the event sequence $e_2 \rightarrow e_1 \rightarrow e_2$ occurs in a $2ms$ interval. This is impossible.

Synchronous FSM vs digraph task models



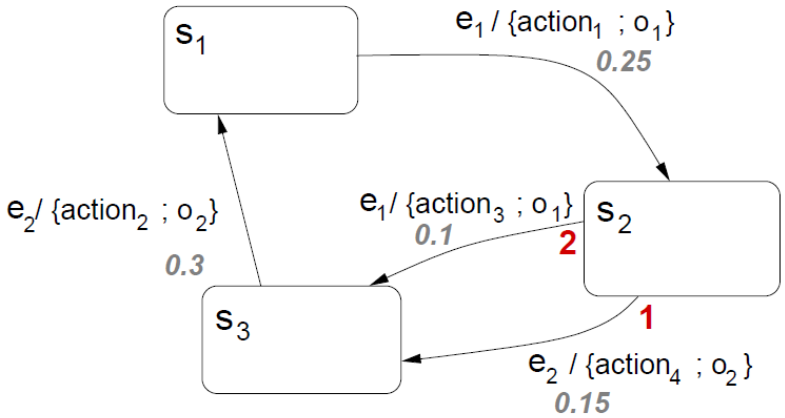


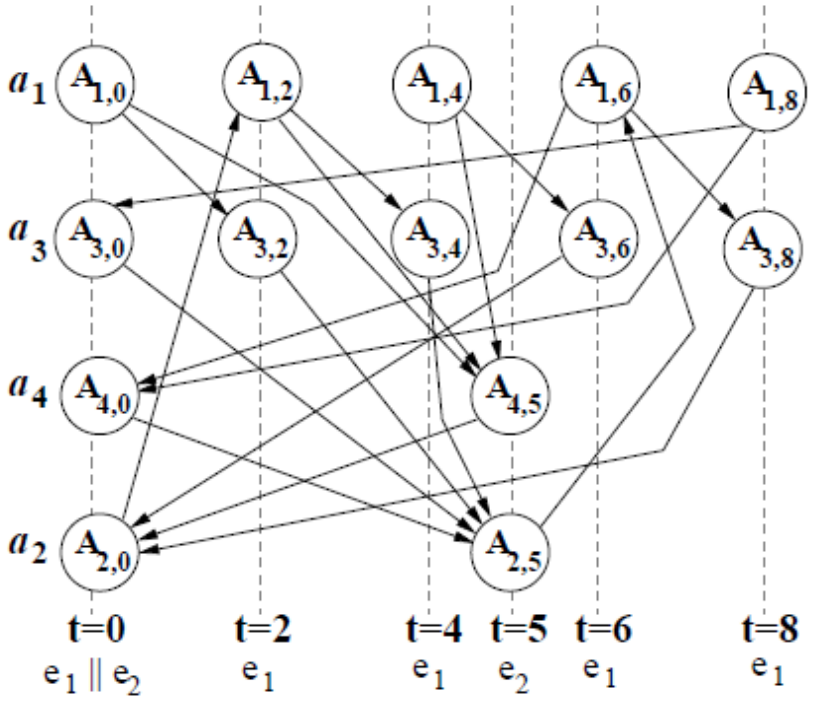
The true task graph model can be computed considering the time (in the hyperperiod) when actions can occur (WCETs and interarrival edges omitted).



This is the comparison of the rbf functions.

Synchronous FSM vs digraph task models





The digraph model can be simplified by removing the edges that are not critical to the schedulability analysis, and by folding vertices by exploiting the periodic pattern of the event arrivals in the hyperperiod.

Schedulability Analysis

Question: what is an efficient way to compute the rbf(Δ) and dbf(Δ) for a given time interval Δ?

Refinement of rbf

$rbf_{i,j}(\Delta)$

- source state of the first transition is S_i
- destination state of the last transition is S_j

$rbf_{i,j}(\Delta)$ is additive ($rbf(\Delta)$ is ***not*** additive)

$$\forall i, j \ \ \forall t \in [s; f], F.rbf_{i,j}[s; f) = \max_m(F.rbf_{i,m}[s; t)+F.rbf_{m,j}[t; f))$$

$rbf_{i,j}[s; f)$ for a long interval $[s; f)$ can be computed from its values for shorter intervals $[s; t)$ and $[t; f)$.

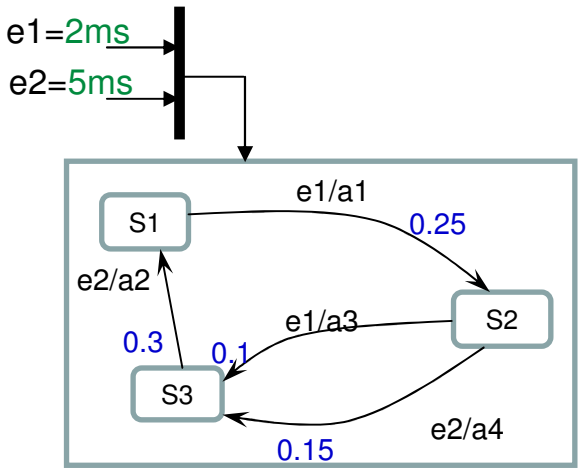
Dynamic programming can be used for computing $rbf_{i,j}(\Delta)$ for large Δ.

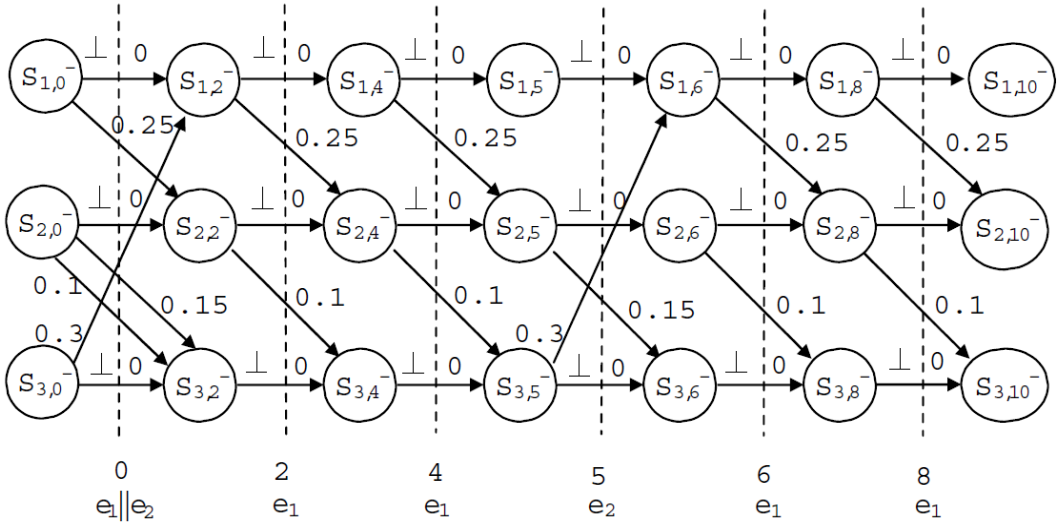
(max,+) algebra can be used to demonstrate the possible periodicity of $rbf_{i,j}(\Delta)$

Computing rbf(Δ)

Step 1: Computing the request bound function in one hyperperiod $X=(x_{i,j})$, where $x_{i,j}=rbf_{i,j}[0,H)$

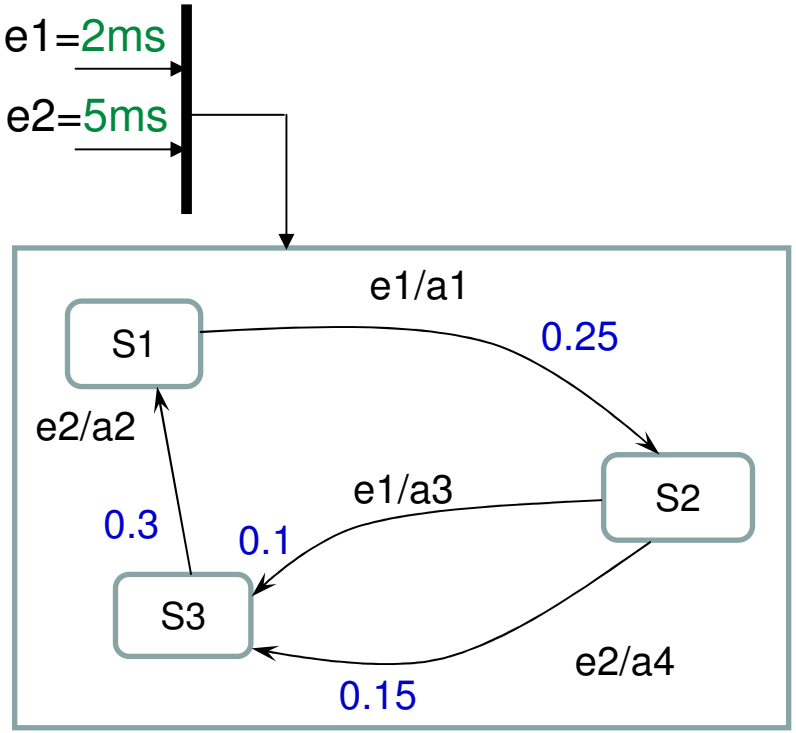
The computation of $rbf_{i,j}(\Delta)$, requires searching the reachable states within the possible sequences of events in the time interval Δ.

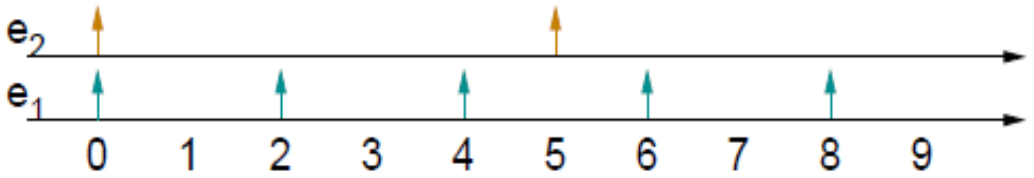




Reachability graph for the example FSM

Execution Request Matrix





event sequence in one hyperperiod

$$S1 \xrightarrow{e1/a1} S2 \xrightarrow{e1/a3} S3 \xrightarrow{e2/a2} S1 \xrightarrow{e1/a1} S2 \xrightarrow{e1/a3} S3$$

$$\Longrightarrow x_{1,3}=1.0$$

$$\Longrightarrow X=\begin{bmatrix} 0.65 & 0.9 & 1.0 \\ 0.45 & 0.7 & 0.8 \\ 0.95 & 1.2 & 1.3 \end{bmatrix}$$

Execution Request Matrix

Step 2: Computing the request bound function over multiples of a hyperperiod

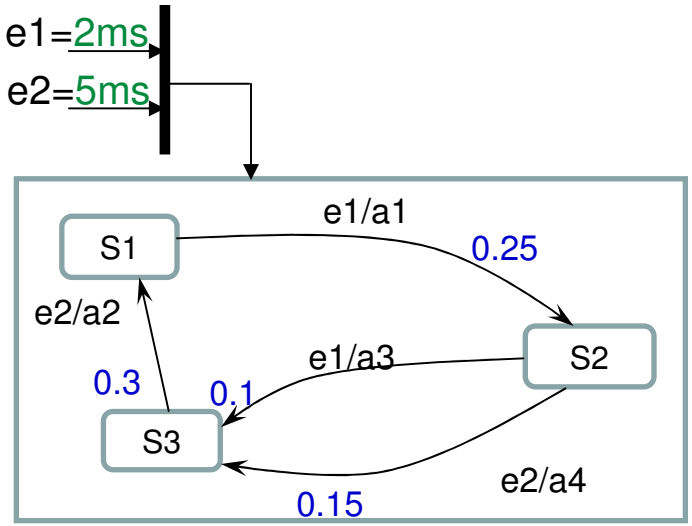
- $X^{(k)}=(x^{(k)}_{i,j})$, where $x^{(k)}_{i,j}=rbf_{i,j}[0,kH)$
- $\forall i,j,\ \forall 1 \le l<k$ $x^{(k)}_{i,j} = max_m(x^{(l)}_{i,m} + x^{(k-l)}_{m,j})$

from previous formula

$$F.rbf_{i,j}[s; f) = \max_m(F.rbf_{i,m}[s; t)+F.rbf_{m,j}[t; f))$$

Possible tool: application of (max,+) algebra

Execution Request Matrix



For our example case it is

$$-X^{(k+1)}=\begin{bmatrix}0.65 & 0.9 & 1.0\\ 0.45 & 0.7 & 0.8\\ 0.95 & 1.2 & 1.3\end{bmatrix}+k\times 1.3$$

This indicates some additional periodicity

Basics on Max-Plus Algebra

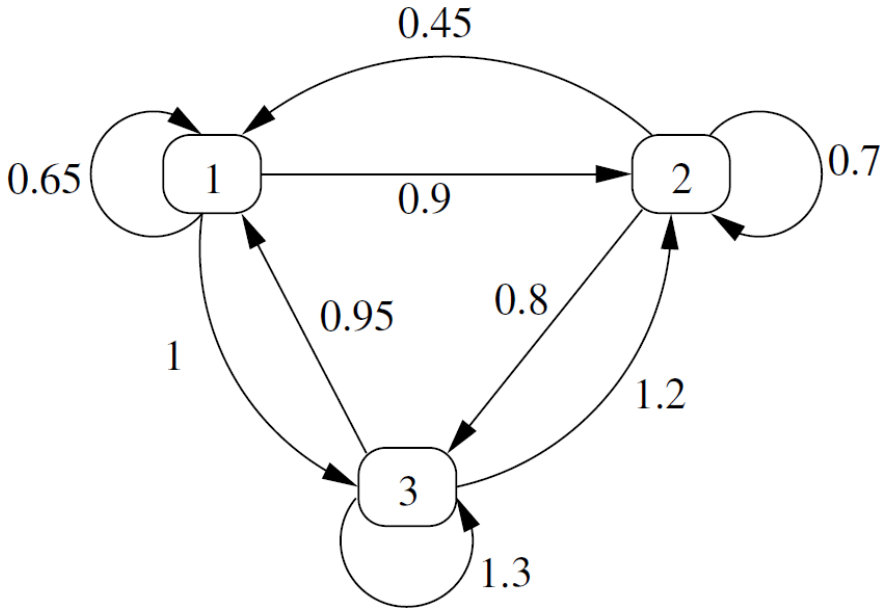
- Operations maximum (denoted by the max operator $\oplus$) and addition (denoted by the plus operator $\otimes$)
  - $a \oplus b = \max(a, b) \quad a \otimes b = a + b$

- Multiplication of two square matrix
  - $A \otimes B = C$, where
    $c_{i,j} = \oplus \left(a_{i,m} \otimes b_{m,j}\right) = \max(a_{i,m} + b_{m,j})$

Periodicity of Matrix Power in Max-Plus Algebra

- Studied by its corresponding digraph $\mathcal{G}(X)$

$$X = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0.65 & 0.9 & 1.0 \\ 0.45 & 0.7 & 0.8 \\ 0.95 & 1.2 & 1.3 \end{bmatrix} \end{array}$$



$\mathcal{G}(X)$

Periodicity of Matrix Power in Max-Plus Algebra

Definition:

Almost linear periodic sequence if

$\exists\ q \in \mathbb{R}$ and $d,p \in \mathbb{I}$ such that for $\forall k > d$

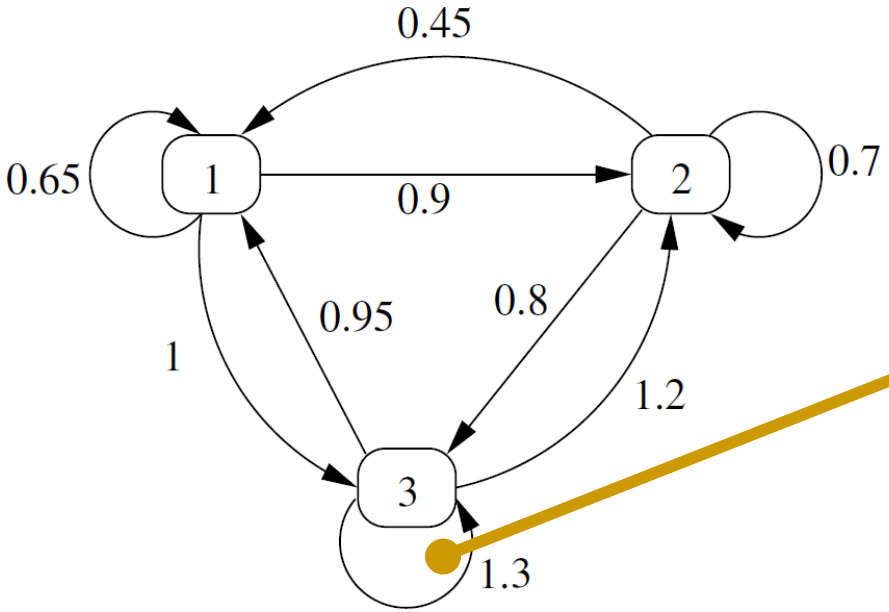
$$X^{(k+p)} = X^{k} + p \times q$$

p is the linear period $p = lper(X)$
q is the linear factor $q = lfac(X)$
d is the linear defect $d = ldef(X)$

Periodicity of Matrix Power in Max-Plus Algebra

- If the digraph $\mathcal{G}(X)$ is strongly connected,
  - $X^{(k+p)} = X^{(k)} + p \times q$ for sufficiently large k
  - p = maxim
  - q = *lcm* of all the cycles with mean equal to p

Definition: X is irreducible



$p=1.3,\ q=1 \Longrightarrow$

$$X^{(k+1)} = \begin{bmatrix} 0.65 & 0.9 & 1.0 \\ 0.45 & 0.7 & 0.8 \\ 0.95 & 1.2 & 1.3 \end{bmatrix} + k \times 1.3$$

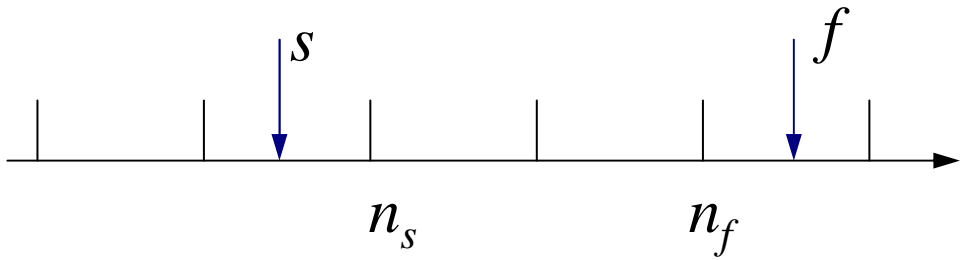
What if the digraph G is not strongly connected

- If X is reducible it could still be linear periodic, but deciding whether this is the case has been demonstrated to be NP-complete
- Periodicity of a different type can still be found (general periodicity, for which linear periodicity is a special case)

The Efficient Way of Calculating rbf

- For small interval $[s,f)$

$$n_s = \left\lceil \frac{s}{H} \right\rceil \qquad n_f = \left\lfloor \frac{f}{H} \right\rfloor$$



- small means n_f-$n_s \leq ldef(X)$ (cannot leverage periodicity)
- The rbf function can be decomposed into functions defined over the intervals $[s,n_sH)$, $[n_sH, n_fH)$ and $[n_fH, f)$
- All terms are computed by simply using the definition

$$\begin{aligned} rbf_{i,j}[s,f) = & \max_{k,l}(rbf_{i,k}[s,n_sH) + rbf_{k,l}[n_sH,n_fH) \\ & + rbf_{l,j}[n_fH,f)) \\ = & \max_{k,l}(rbf_{i,k}[s,n_sH) + x_{k,l}^{(n_f-n_s)} \\ & + rbf_{l,j}[0,f-n_fH)) \end{aligned}$$

The Efficient Way of Calculating rbf

- For large intervals, when n_f-n_s is larger than the linear defect of X, we leverage the periodicity for the computation of the intermediate term.

$$x_{k,l}^{(n_f-n_s)} = x_{k,l}^{(n)} + (n_f - n_s - n) \times q_{k,l}(k)$$
$$\text{where } n \leq d \text{ and } n + p > d, \quad n_f - n_s \equiv n \equiv k \mod p.$$

Summary and Future Work

- Efficient and accurate schedulability analysis
  - Event sequence pattern within one hyperperiod
  - Max-plus algebra for periodicity of execution request matrix
- Additional issues
  - Multi-task implementation of an FSM
    - Issues with single task implementation:
      - all actions executed at the same priority level
      - tight deadline (equal to the gcd of event periods)
  - Approximations for large FSMs

- Details in: Haibo Zeng, Marco Di Natale: *Schedulability Analysis of Periodic Tasks Implementing Synchronous Finite State Machines*, Euromicro Conference on Real-time Systems, 2012

Q&A

Thank you!