The OASIS model for developpement of deterministic safety-critical multitask real-time systems

Matthieu Lemerre

CEA LIST
LaSTRE (V. David)

Synchron'08
Aussois, December 5th, 2008

Introduction

- OASIS is an industrialized toolchain (compiler, runtime kernel...)
- based on a time-triggered model of computation
- OASIS model : computations take time (no zero-time abstraction)
  - Computations happens during *intervals* of time (not point in time)
  - Use results of *past* computations (not that of parallel ones)
- Model still independent of execution time
  (used only for scheduling)

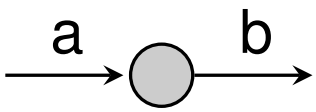
Introduction

- OASIS is an industrialized toolchain (compiler, runtime kernel...)
- based on a time-triggered model of computation
- OASIS model : computations take time (no zero-time abstraction)
  - Computations happens during *intervals* of time (not point in time)
  - Use results of *past* computations (not that of parallel ones)
- Model still independent of execution time
  (used only for scheduling)
- Ψ is a language extension
  - for expressing the time-triggered constraints of the model
  - in a procedural language (e.g. C or Ada)

Outline

Outline

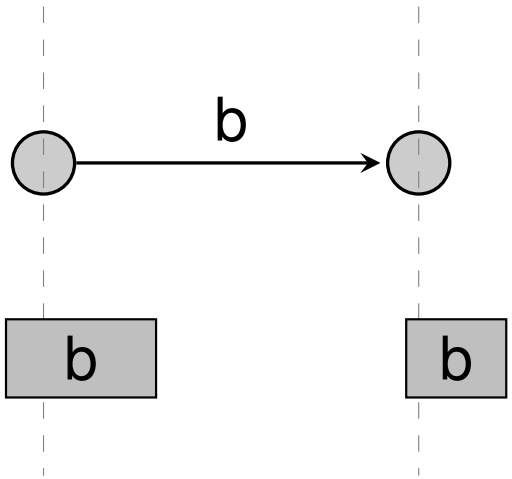
Chains



- Notions

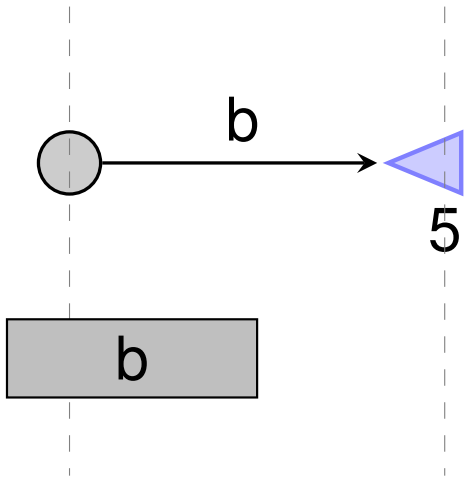
Block	Sequence of instructions, represented by an arc.
Nodes	Separates 2 blocks
Chain	Sequence of blocks and nodes. *b* runs after *a*.

Temporal constraints



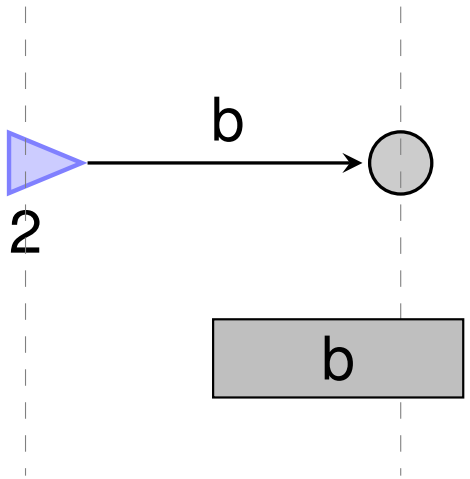
- Goal: to specify temporal constraints on a block
- 2 possibilities: to make it start after a certain date OR to make it end before another date

Temporal constraints



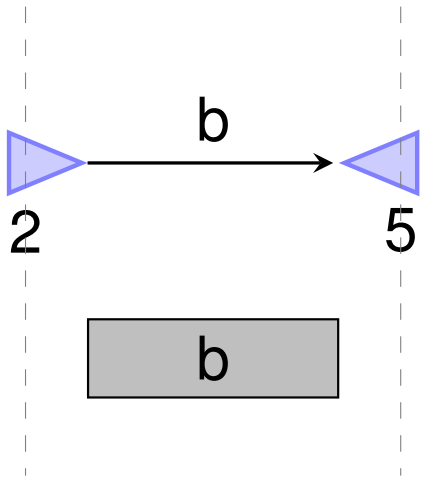
- Goal: to specify temporal constraints on a block
- 2 possibilities: to make it start after a certain date OR to make it end before another date
- We chose to make the adjascent nodes bear the constraints:
- “before” nodes constraint the *preceding* block

Temporal constraints



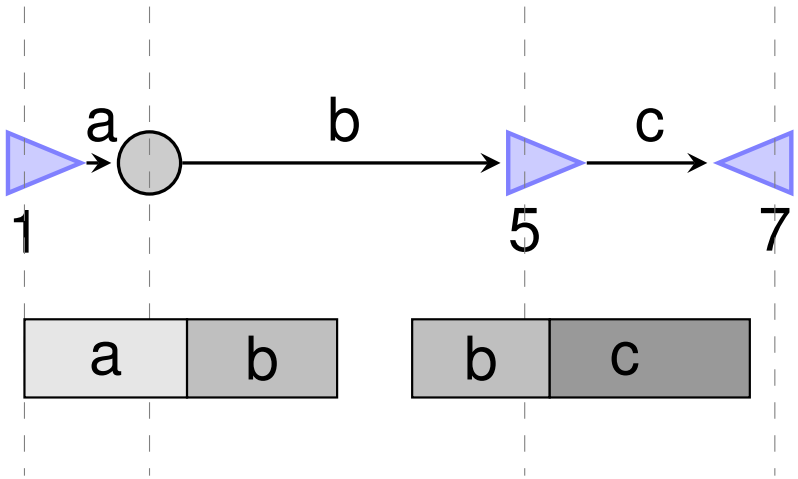
- Goal: to specify temporal constraints on a block
- 2 possibilities: to make it start after a certain date OR to make it end before another date
- We chose to make the adjascent nodes bear the constraints:
- “before” nodes constraint the *preceding* block
- “after” nodes constraint the succeeding *block*

Temporal constraints



- Goal: to specify temporal constraints on a block
- 2 possibilities: to make it start after a certain date OR to make it end before another date
- We chose to make the adjascent nodes bear the constraints:
- “before” nodes constraint the *preceding* block
- “after” nodes constraint the succeeding *block*

Temporal constraints

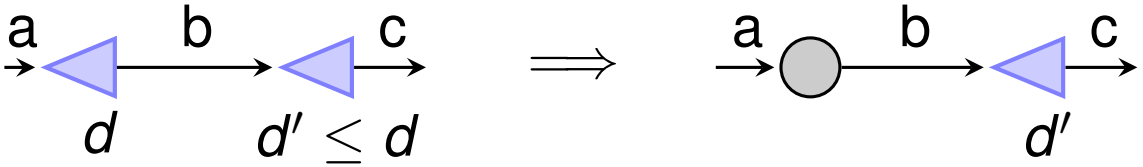


- Goal: to specify temporal constraints on a block
- 2 possibilities: to make it start after a certain date OR to make it end before another date
- We chose to make the adjascent nodes bear the constraints:
- “before” nodes constraint the *preceding* block
- “after” nodes constraint the succeeding *block*

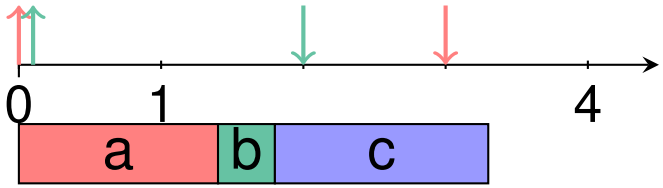
Lemma: constraint extension

A block *b* is implicitly constrained by all preceding “after” nodes, and by all following “before” nodes

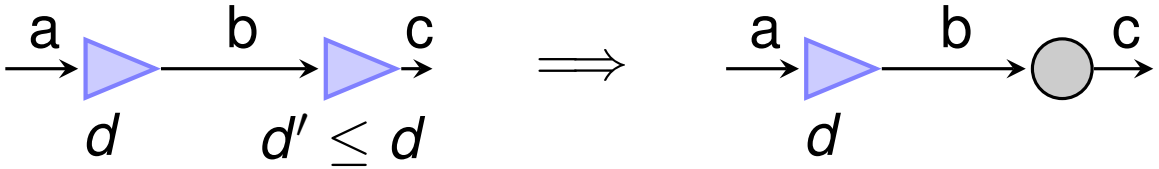
Combining constraints

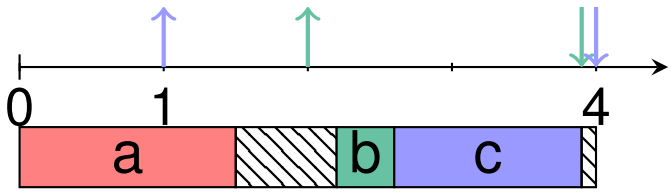


- Some constraints can be simplified



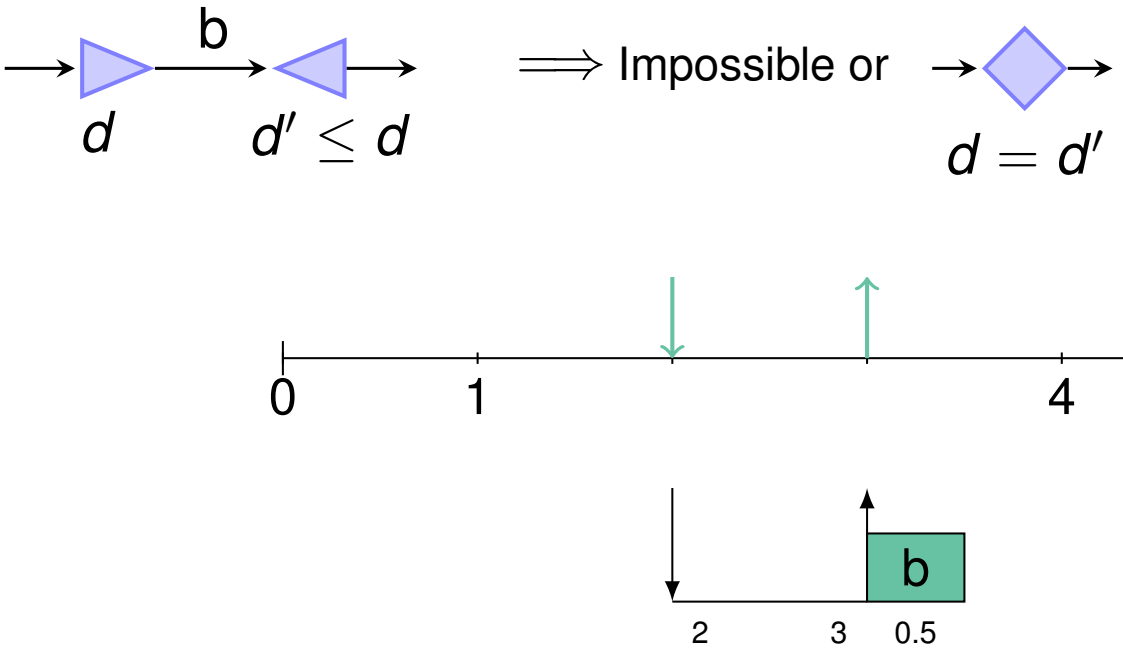
Combining constraints





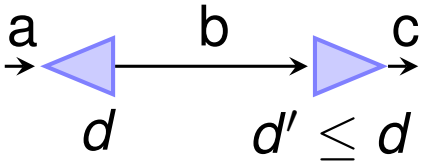
- Some constraints can be simplified

Combining constraints

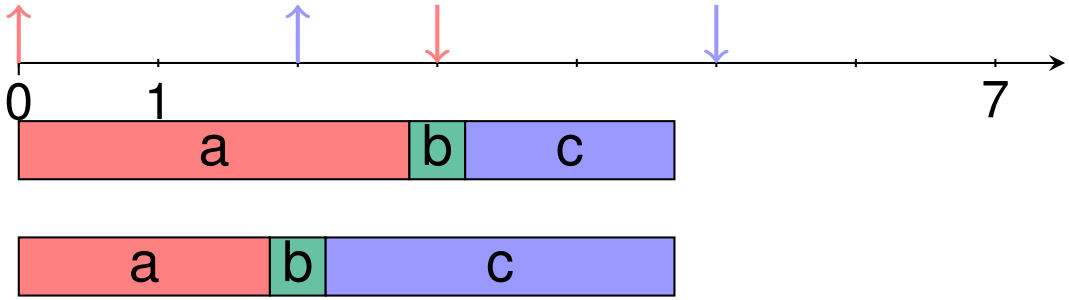


- Some constraints can be simplified

Combining constraints

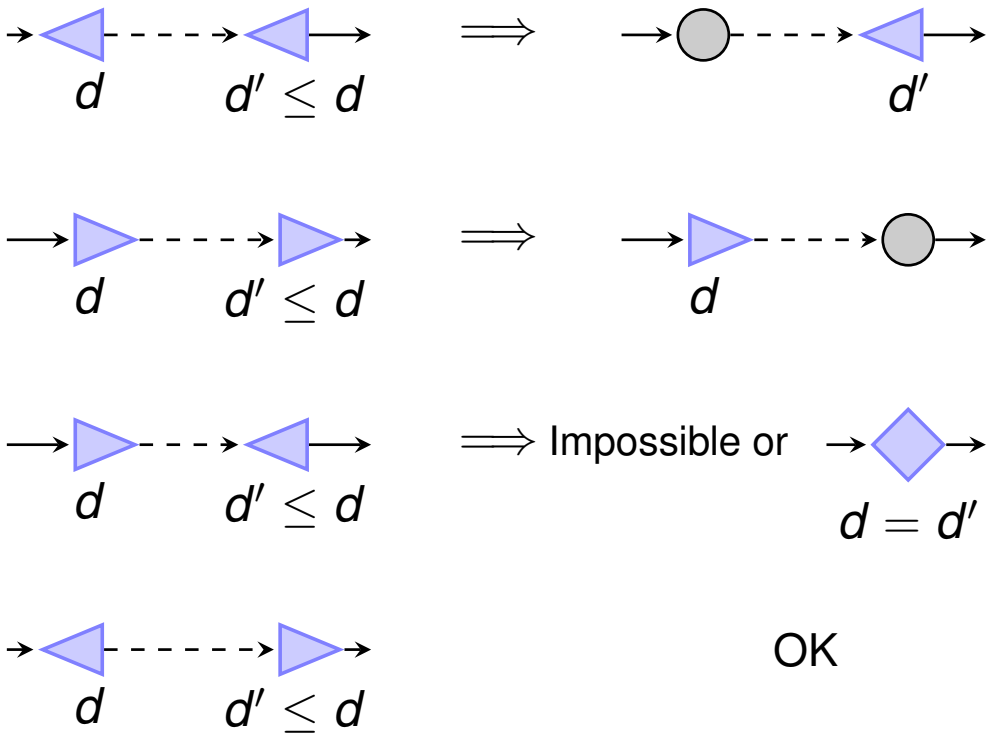


OK



- Some constraints can be simplified

Combining constraints

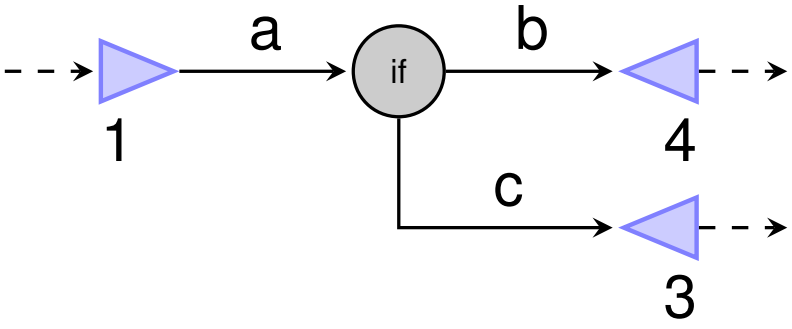


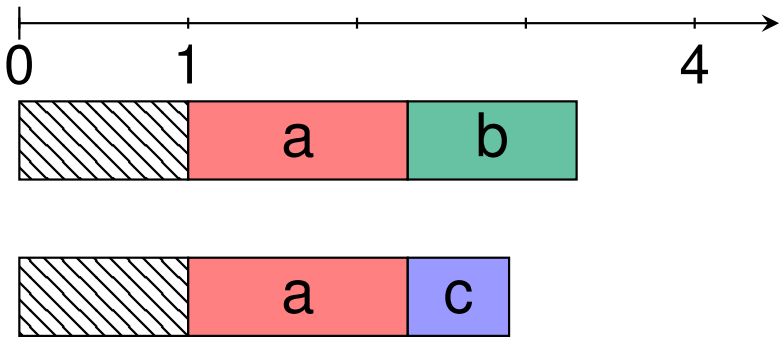
- Some constraints can be simplified
- Reductions (*a*) and (*b*) allows for relative labeling of constraints:

Lemma (relative labeling)

All constraints can be expressed as a stricly positive increment from the last “after” node.
(In an equivalent, simplified automaton)

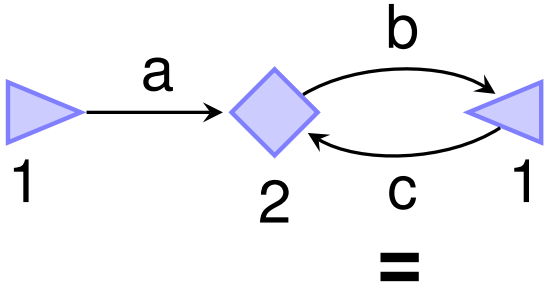
From chains to trees: Handling choices



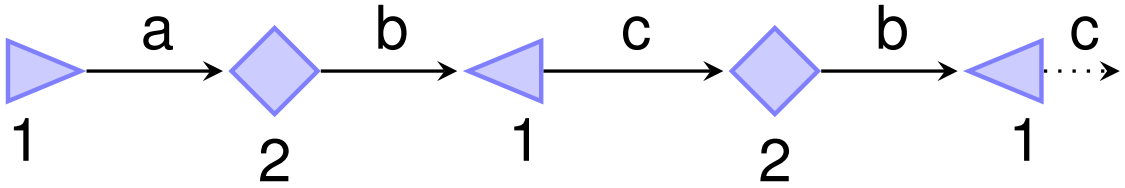


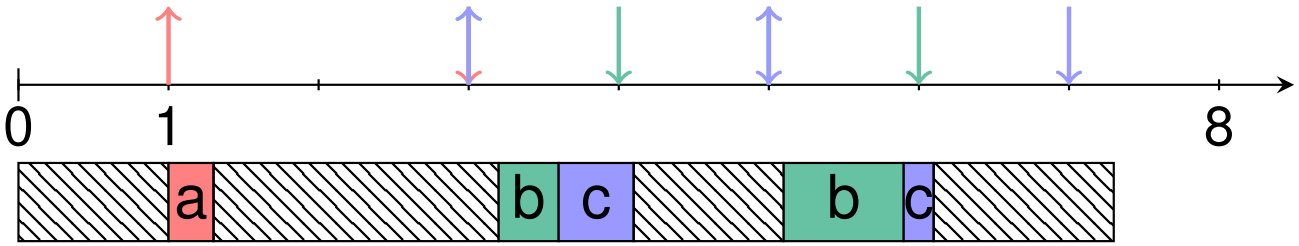
- either *b* or *c* will be executed.
- $a \rightarrow b$ has 3 unit of times to complete
- $a \rightarrow c$ only has 2.
- Important:
  - Choice between *b* and *c* is done dynamicallly
  - And is known only after *a*'s execution
  - $\rightarrow$ Both choices must be doable until *a* finishes
  - This is reflected in scheduling semantics (later)

From trees to automata: Handling loops and joins



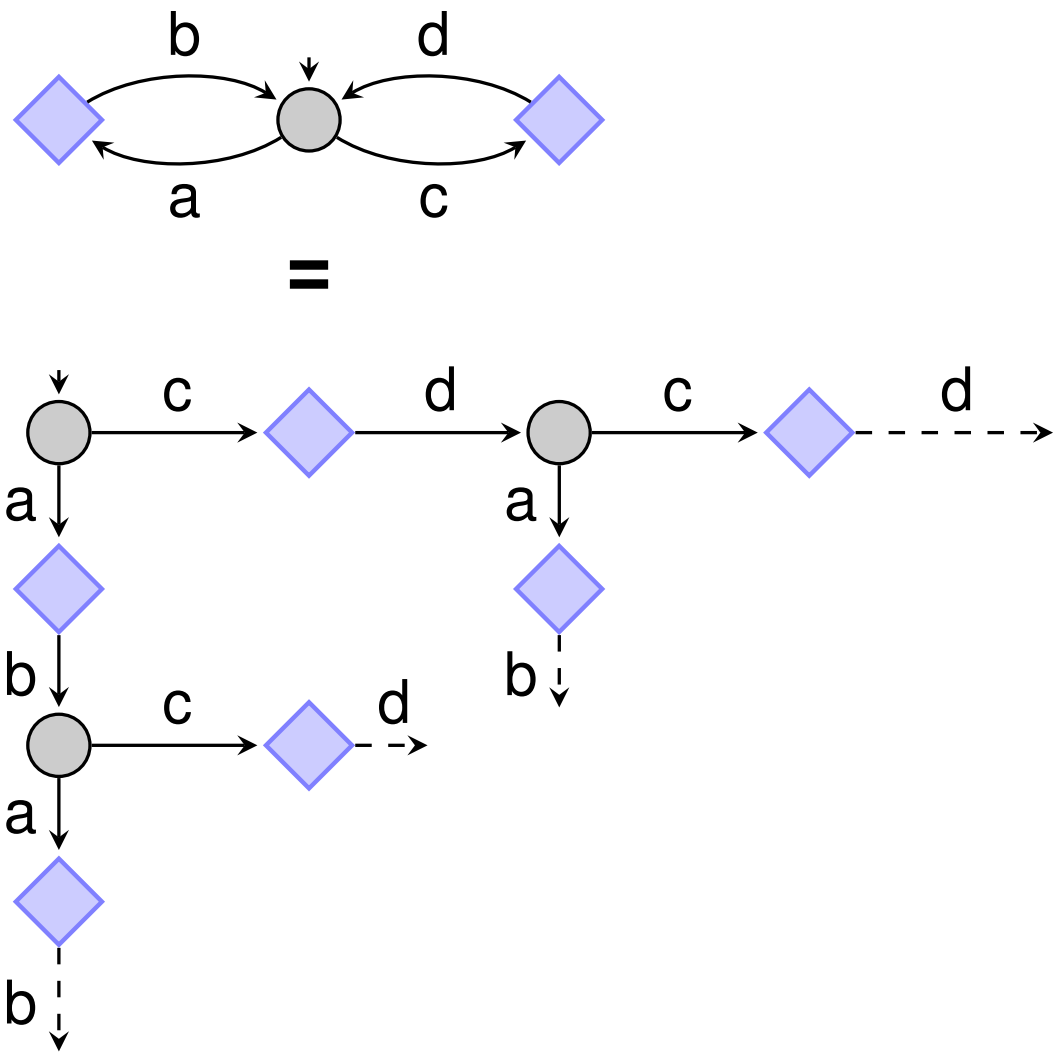
=





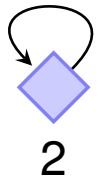
- Semantic of unfolding : "copy" of the traversed tree
- Must use *relative labeling* to be useful

From trees to automata: Handling loops and joins

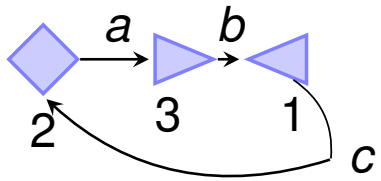


- Semantic of unfolding : “copy” of the traversed tree
- Must use *relative labeling* to be useful
- When both loops and choices: unfolding gives an infinite tree!
- Execution trace
  = path in the tree
  = chain

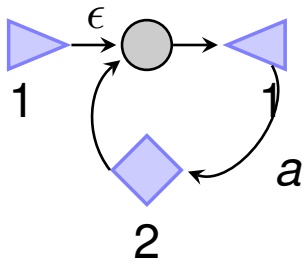
Example uses

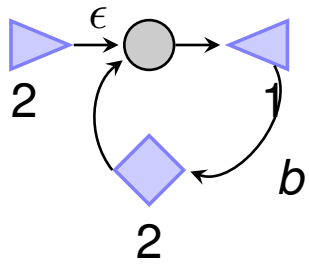


(a) Implicit-deadline periodic task of period 2



(b) A periodic task (period 5). *b* is constrained with fine-grained jitter specification (maximum jitter 1)

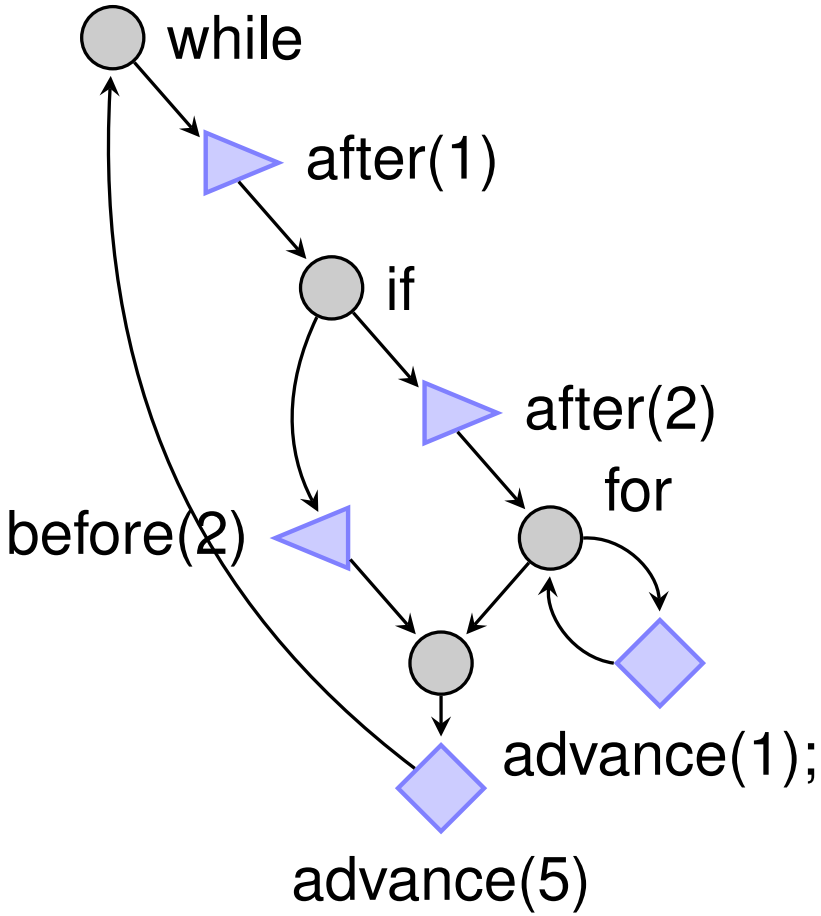




(c) Two constrained-deadline periodic tasks of period 2 and deadline 1, with respective phase 1 and 2. *a* and *b* are in mutual exclusion

Generation of the automaton from ΨC code

```
while(1) {
  after(1);
  if(...) {
    after(2);
    for(i=0;i<10;i++)
     { advance(1); }
  }
  else before(2);
  advance(5);
}
```



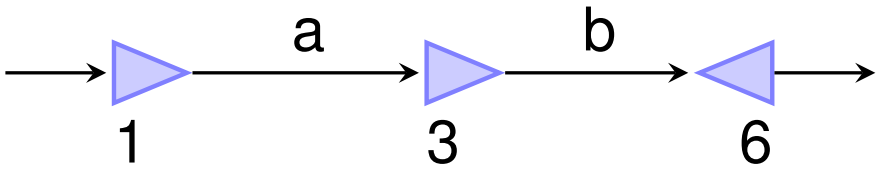
- The CFG defines the automaton.
- Tasks define sequential execution
- Multiple tasks to get parallel execution

Outline

Semantics for scheduling chains

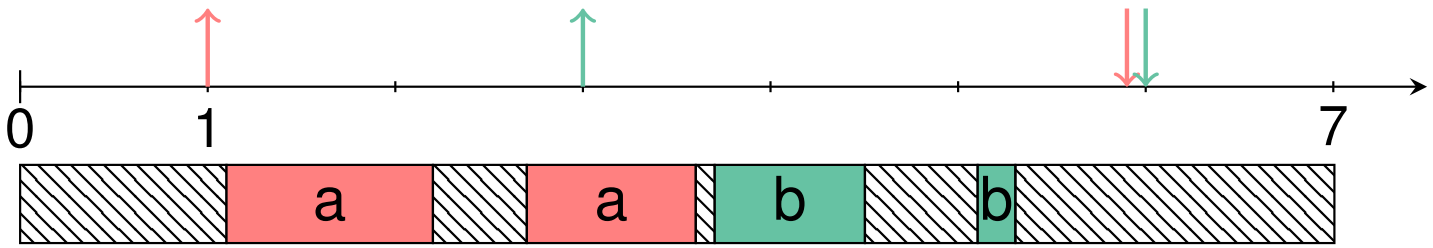
Conditions for a correct schedule for a chain:

- To respect the “before” and “after” constraints
- Blocks must be executed in order
- Blocks must be executed for their required execution time $||b||$



$||a|| = 2$

$||b|| = 1$



Optimal scheduling with EDF-dyn

- In conventional scheduling models, tasks dynamically release fix jobs (i.e. 1 starttime, 1 deadline, 1 execution time)
- In the OASIS task model, a task is one job that changes dynamically
- $\longrightarrow$ cannot use conventional scheduling algorithms “as is”

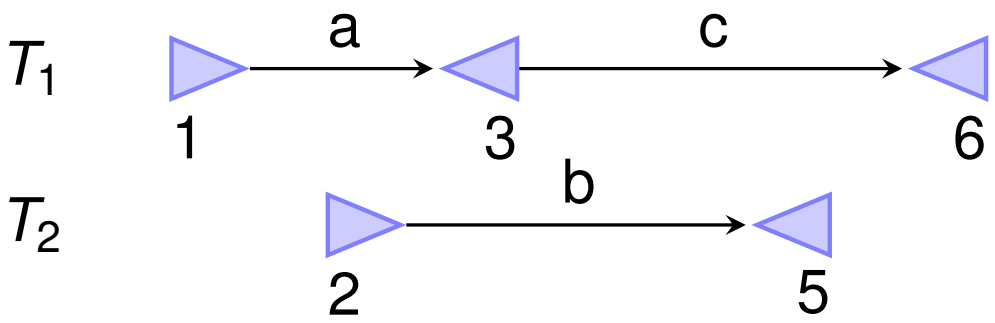
Definition

EDF-Dyn: EDF with dynamic change of deadlines

Optimal scheduling with EDF-dyn

Theorem

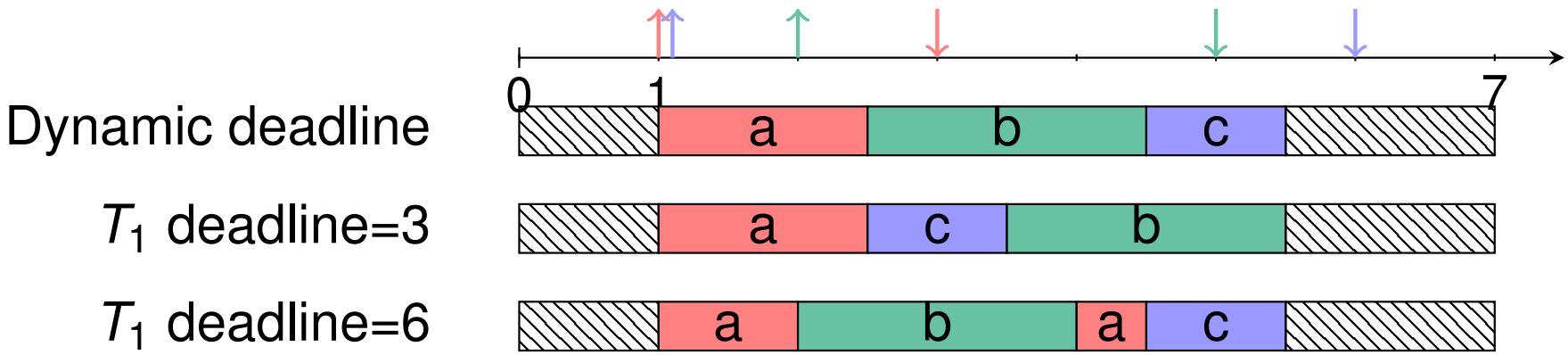
EDF-Dyn is optimal for scheduling OASIS tasks.



$$||a|| = 1.5$$

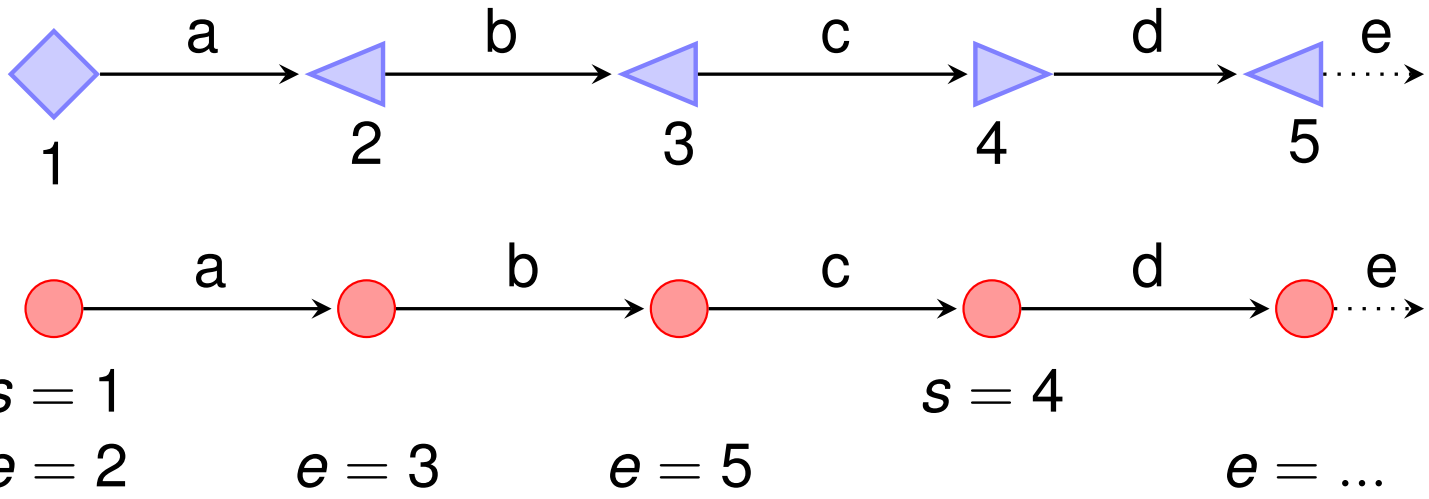
$$||b|| = 2$$

$$||c|| = 1$$

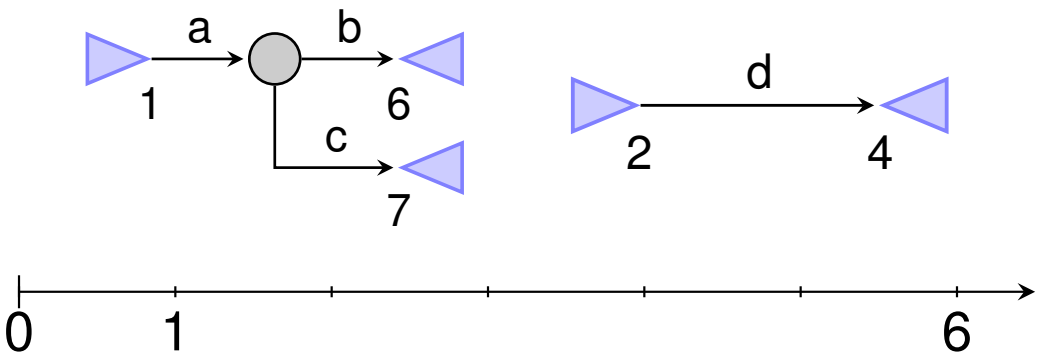


Implementation

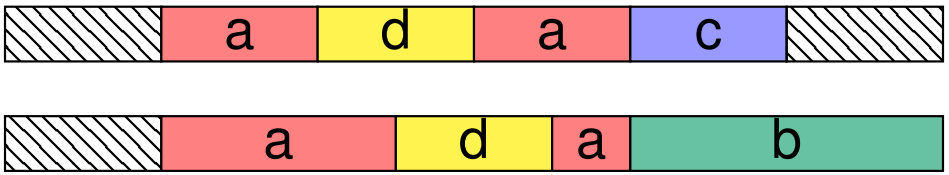
- Scheduling decisions have to be made only when
  - The date of an after node is reached
  - An after node is reached
  - A before node is reached
- Scheduler implementation:
  - Deletion of useless constraints
  - Translation into system calls to the scheduler



Semantics for scheduling trees



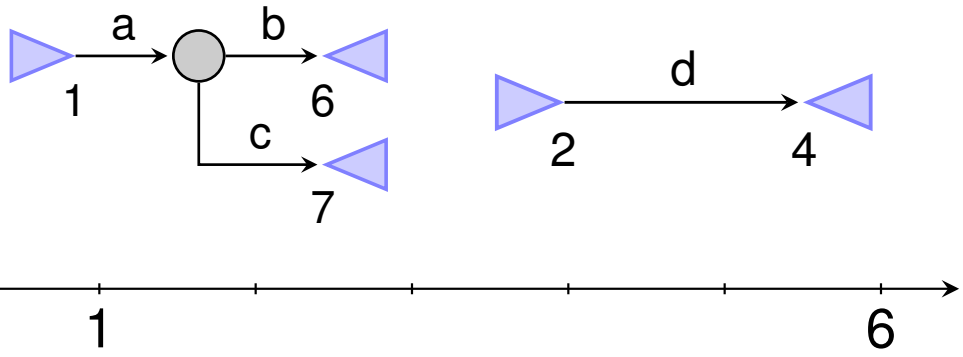
(f) Valid schedule



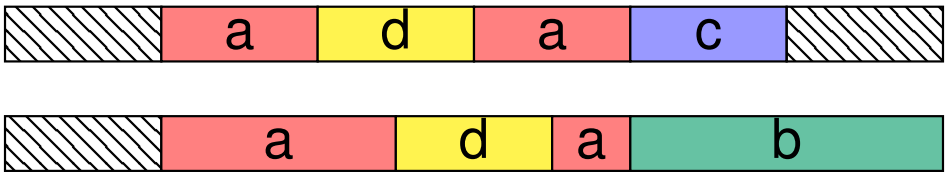
(g) Invalid schedule : differ before the instant of choice

- 1 tree schedule = one schedule for chain per set of choices made
- Schedules for chain must be the same up to their differenciating choice
- Each schedule for chain must be correct

Semantics for scheduling trees



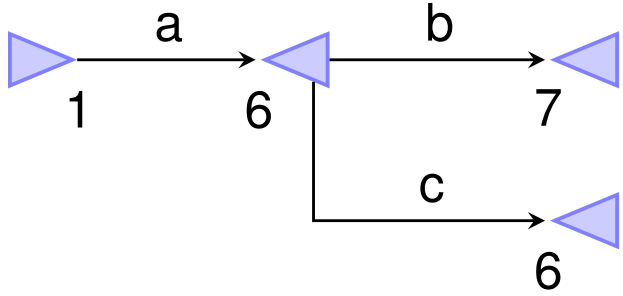
(j) Valid schedule



(k) Invalid schedule : differ before the instant of choice

Definition

Choice deadline inheritance: a transformation where all choice nodes have the earliest possible following constraint



Theorem

CDI is a necessary condition

EDF wich choice deadline inheritance

Definition

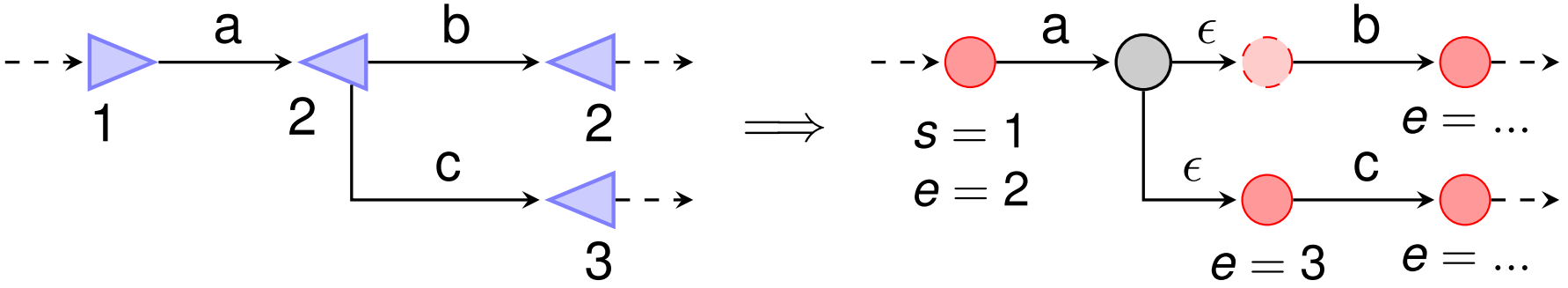
EDF-Dyn-min: EDF dyn with the earliest possible deadline

Theorem

EDF-dyn-min is optimal for scheduling time-triggered trees on a single processor

Implementation:

- Translation of choice nodes into "`update deadline`" system calls.



Scheduling of automata

- Same as scheduling the unfolded tree
- Relative $\longrightarrow$ absolute conversion done dynamically
- Execution of the code unfolds the automaton on the fly
- Additional checking of reachability of nodes
- Exact feasibility analysis is possible jump

Outline

Communication primitives

- The different tasks need to communicate
- Synchronizations (rendez-vous/semaphores) render feasibility analysis and scheduling very difficult
- $\longrightarrow$ All synchronisations done solely using time
- We additionally need to ensure that buffer size is sufficient
- $\longrightarrow$ automatic computation of buffer size

A simple communication primitive

- Simple example:
- Using shared memory for communication
- Synchronisation using time

(l) Synchronisation using time:*s* writes to shared memory before time 3, *r* reads it after time 3

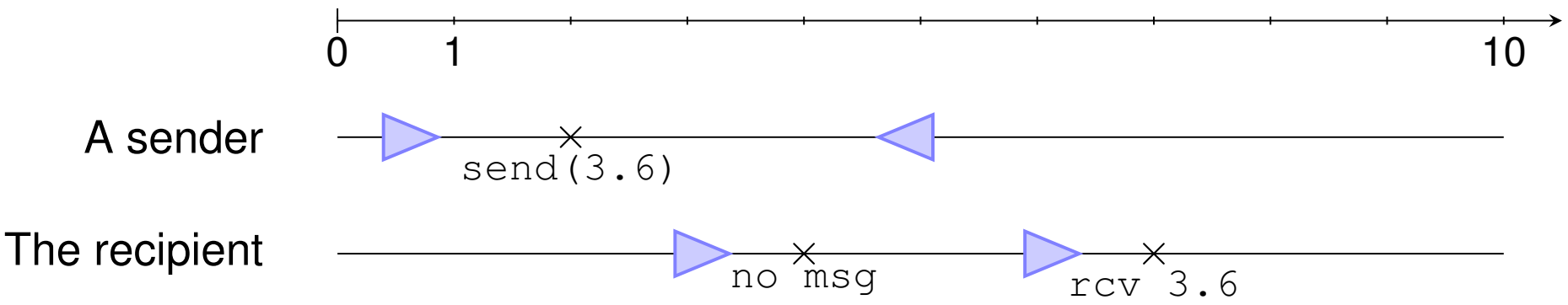
A simple communication primitive

- Simple example:
- Using shared memory for communication
- Synchronisation using time

(m) Synchronisation using time:*s* writes to shared memory before time 3, *r* reads it after time 3

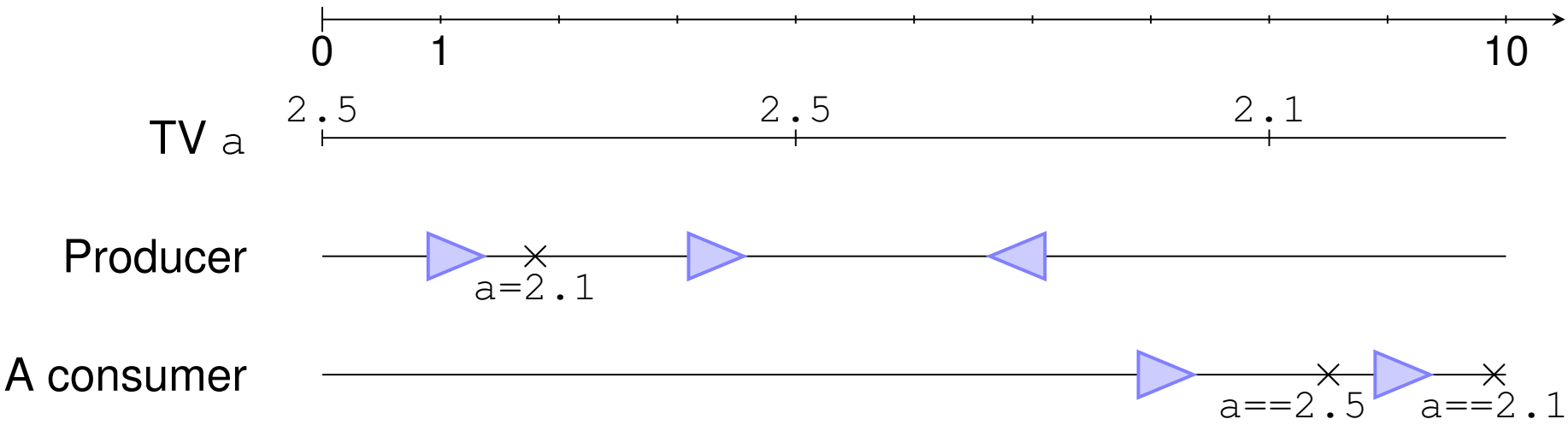
- Interactions may still be undeterministic
- $\longrightarrow$ provide only communication primitives that preserve determinism
- $\longrightarrow$ *guaranteed* determinism

A communication primitive: the message



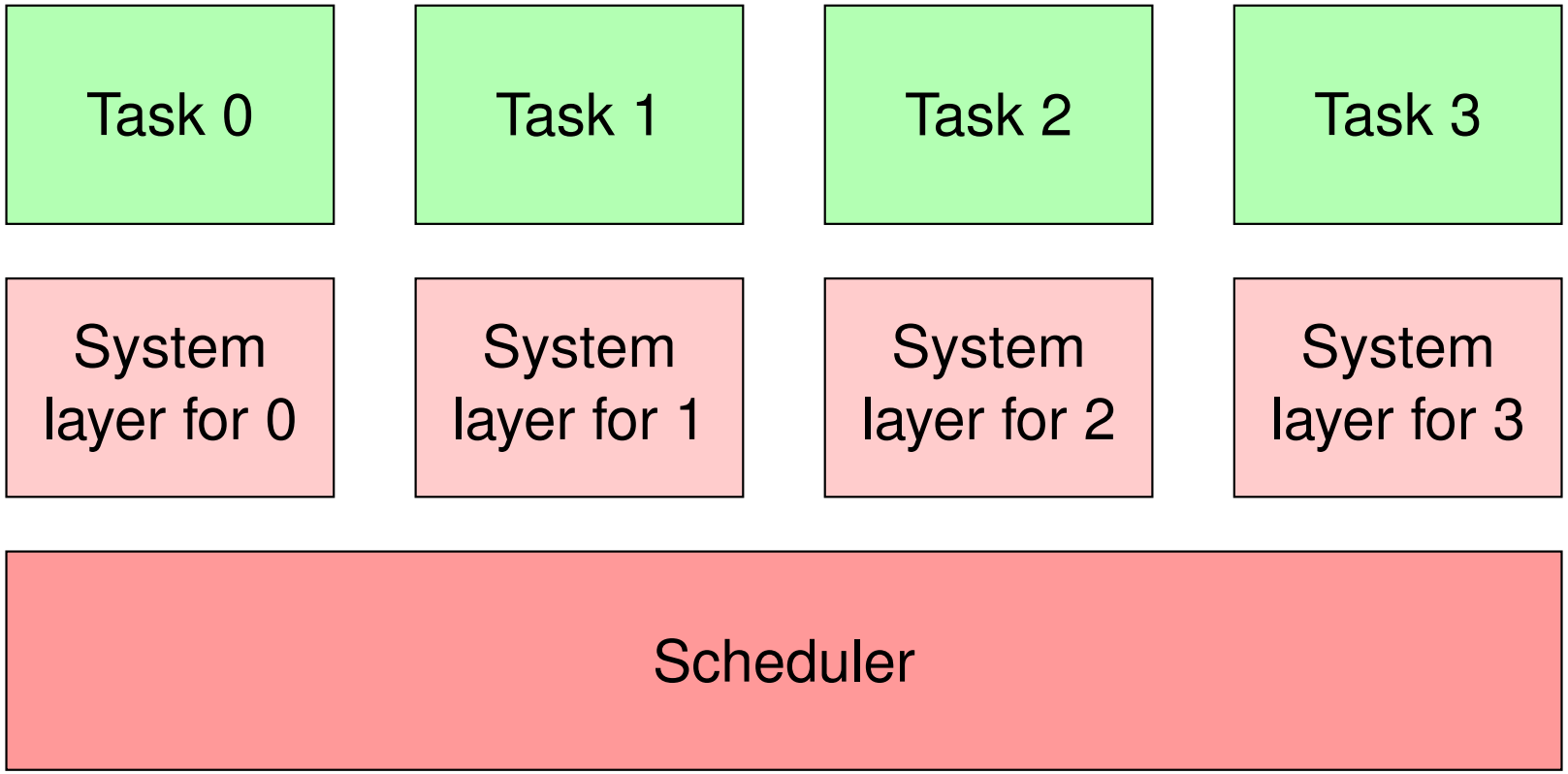
- *N* to 1 communication
- Typed mailboxes
- Mailbox have “peremption dates” → buffer sizes can be computed
- `send` instruction specifies a visibility date (a before constraint)
- Value read does not depend on the actual time of send/recv → determinism
- Synchronous model would need only afters
- Wait-free implementation

A communication primitive: the temporal variable



- The temporal variable is updated periodically
- Updates are independent of any task rhythm
- One producer, *N* consumers
- Value read does not depend on the actual time of read
  $\longrightarrow$ determinism
- Wait-free implementation

Runtime



Designed for safety:

- All parts separated in different memory contexts (e.g. process)
- Implementation on bare hardware (OASIS kernel) or simulation on Linux

Outline

Summary

- The OASIS model is a very expressive model for expressing time-triggered timing constraints
- Industrially used to realize complex safety-critical systems
- Easy to prove application timing properties
- Safety-oriented design and implementation
- Special scheduling semantics, quite unusual
- Communication primitives that guarantee determinism despite multitasking
- Independence from hardware:
  - correct execution guaranteed if schedulable (i.e. hardware fast enough)
- Really high efficiency:
  - Multitasking
  - No lock
  - Optimal scheduling and exact feasibility analysis

Conclusion

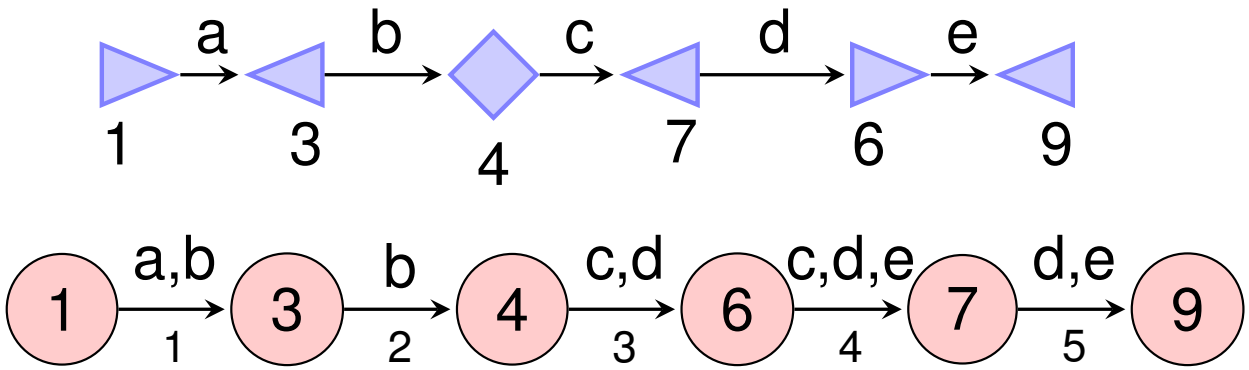
Present and future extensions:

- OASIS for the automative industry
- OASIS for SMP
- Intra-task parallelism (multithreading)
- Distributed OASIS
- Formal transition from specifications to implementation

Thanks for your attention

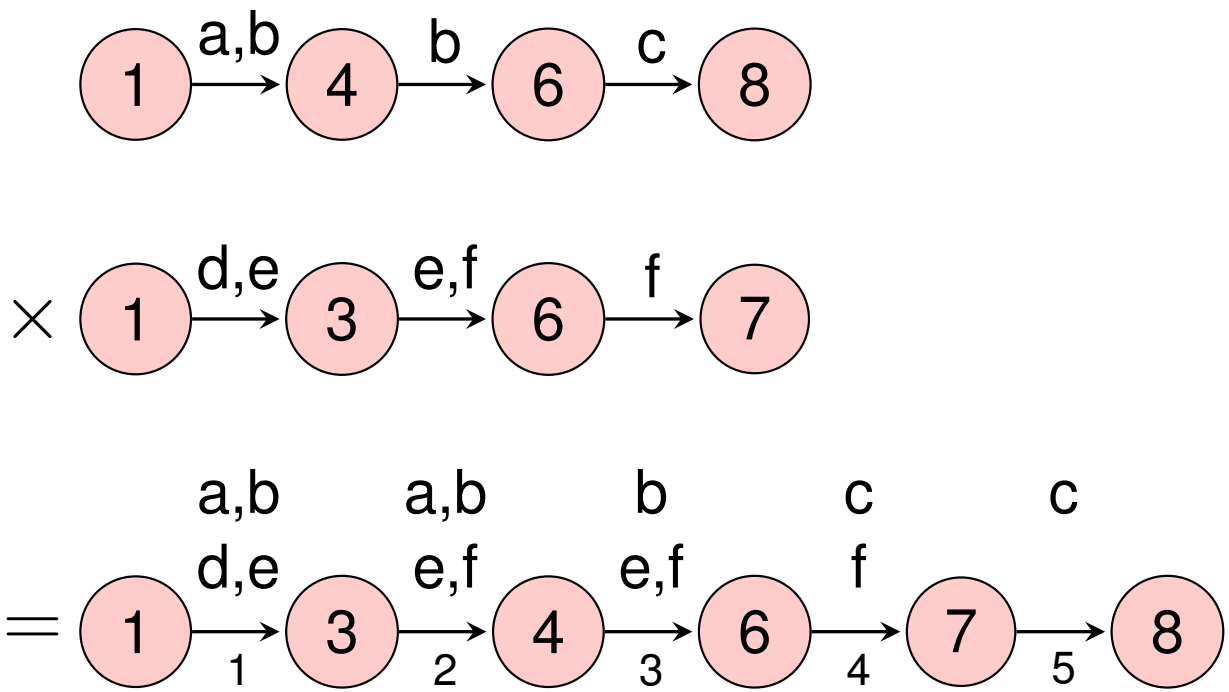
Questions?

Transformation to temporal chains



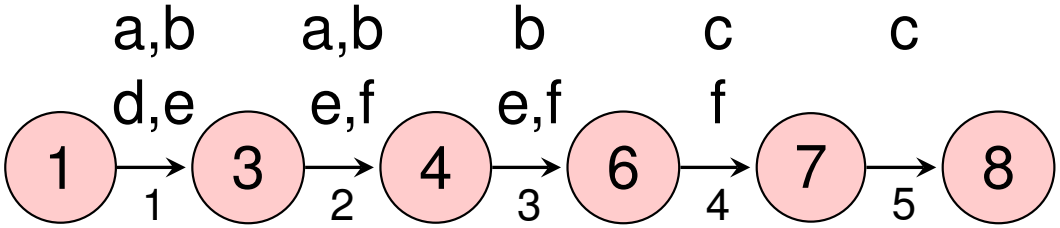
- Temporal chains:
  - Nodes represent dates, and are ever-increasing
  - Arcs represent intervals
  - Arcs carry blocks that can be executed during the interval (above)
  - Arcs are labelled (below)
  - Algorithm for transforming TT chains into temporal chains is simple

Composition of temporal chains



- Result shows which blocks can be executed on each interval
- Separation of blocks coming from different chains
- $\mathcal{A}_b$ set of arcs of a block b
- $\mathcal{B}_a$ set of blocks on an arc a

Feasibility of temporal chains



$$\forall n \text{ arc}, \quad \sum_{b \in \mathcal{B}_n} b_n \leq |l_n| \qquad \forall b \text{ block}, \quad \sum_{n \in \mathcal{A}_b} b_n = ||b||$$

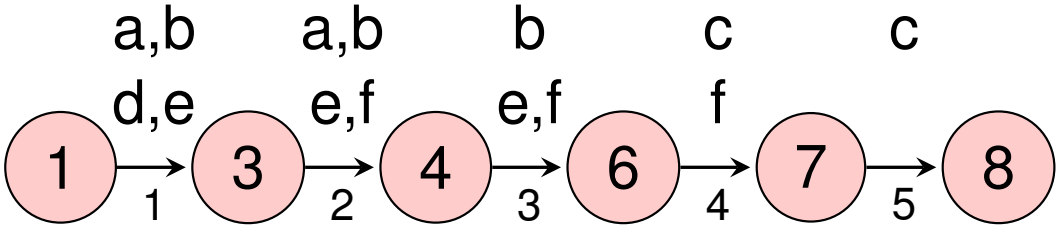
$$\begin{aligned} a_1 + b_1 + d_1 + e_1 &\leq 3 - 1 \\ a_2 + b_2 + e_2 + f_2 &\leq 4 - 3 \\ b_3 + e_3 + f_3 &\leq 6 - 4 \\ c_4 + f_4 &\leq 7 - 6 \\ c_5 &\leq 8 - 7 \end{aligned}$$

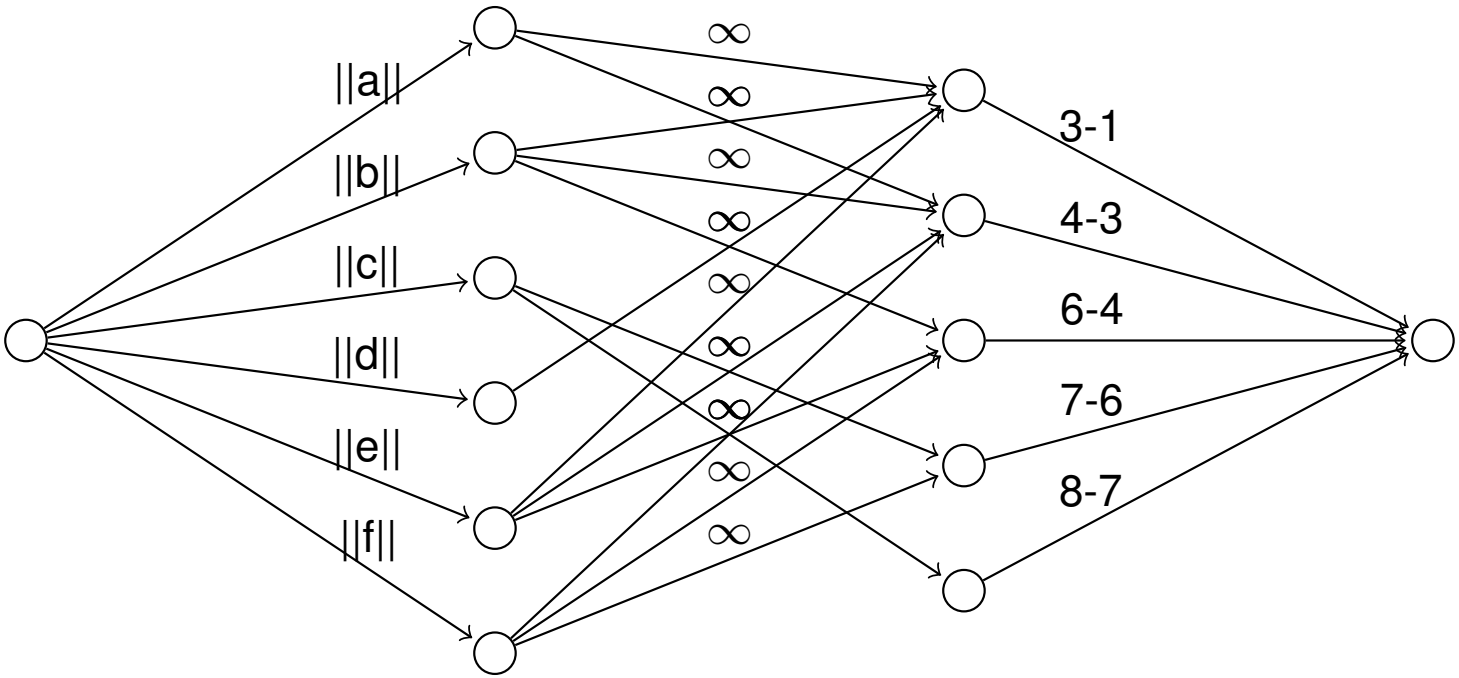
$$\begin{aligned} a_1 + a_2 &= ||a|| \\ b_1 + b_2 + b_3 &= ||b|| \\ c_4 + c_5 &= ||c|| \\ d_1 &= ||d|| \\ e_1 + e_2 + e_3 &= ||e|| \\ f_2 + f_3 + f_4 &= ||f|| \end{aligned}$$

Theorem: feasibility of temporal chains

A temporal chain is feasible on a single-processor computer if, and only if, there is a solution to the previous equations.

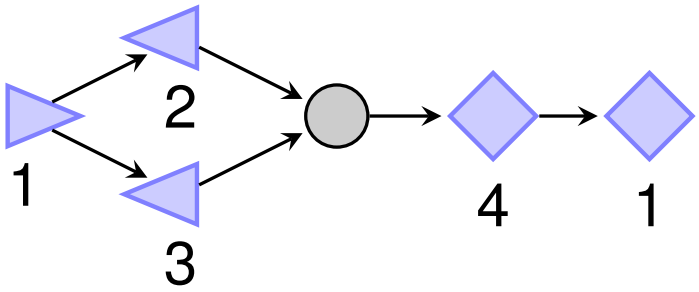
Resolution using network flows

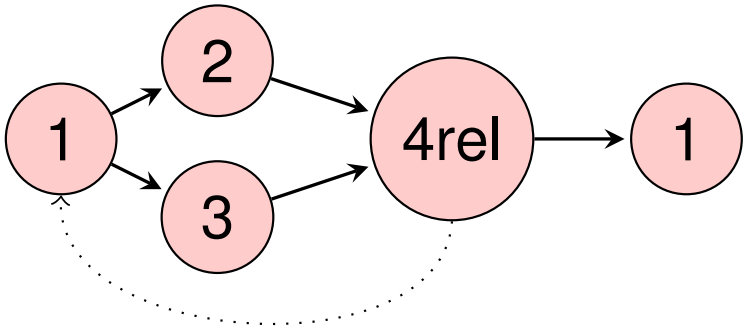


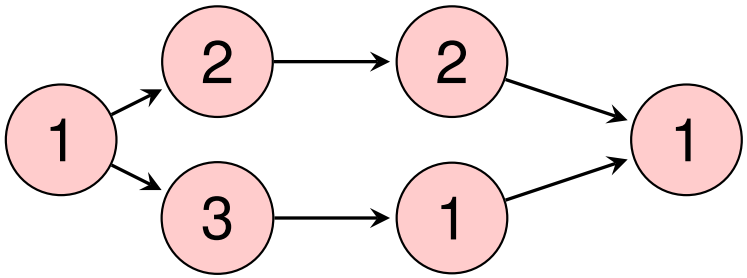


- Use a maximal flow algorithm
- Allows a more efficient resolution than LP

Conversion of TT automaton to temporal automaton

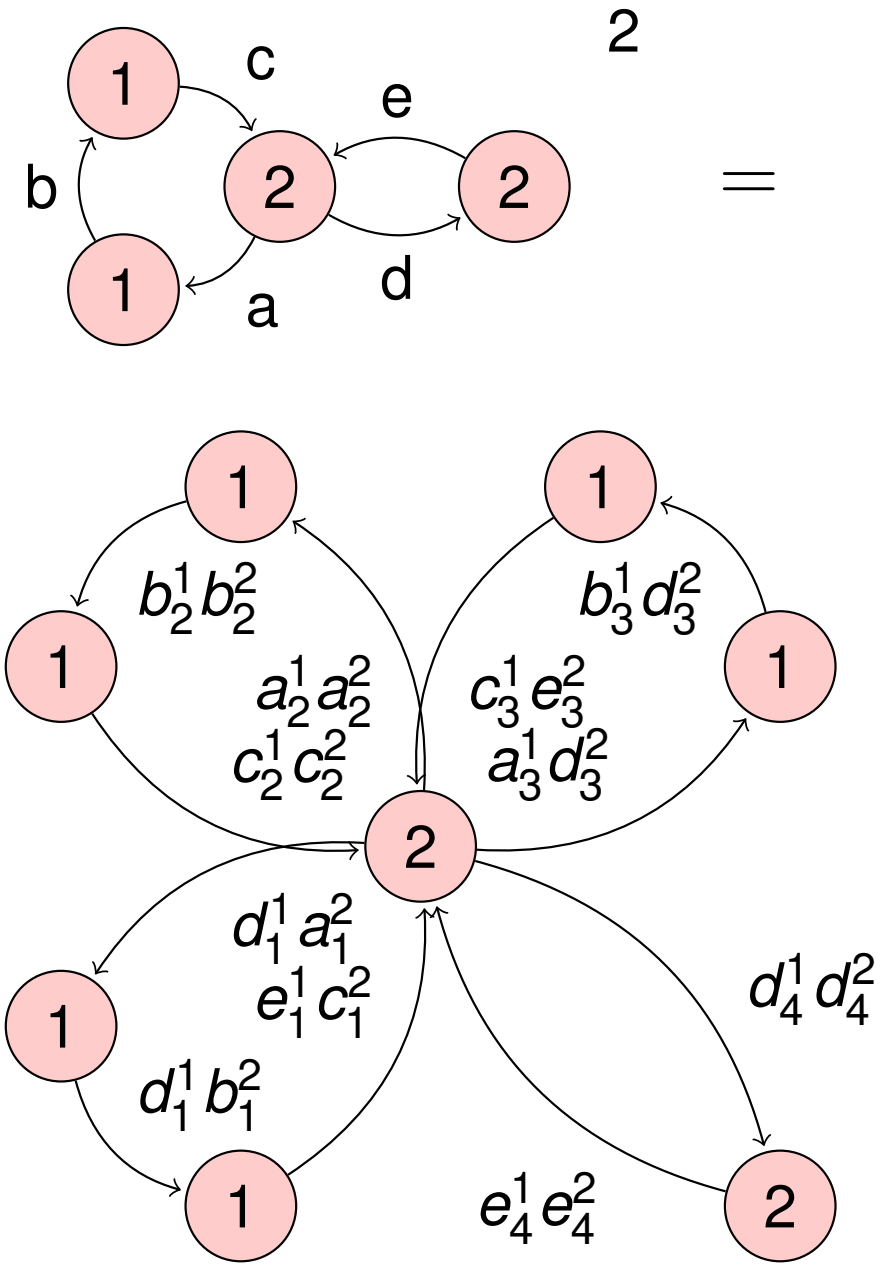






- Steps:
  1. Suppression of “no-constraints” nodes
  2. “Partial relative” labelling
  3. “Complete relative” labelling (need duplicating nodes)
- Other operations:
  - expanding variable temporal constraints
  - expanding temporal constraints with a different clock
- $\longrightarrow$ not simple algorithm!
- Current OASIS algorithm: unfold everything and re-convert to an automaton

Product of temporal automata



- Product of temporal automata
- We assume all nodes are labelled by integers
- Close to classical product of automata
- (if we split nodes labeled n by n nodes labelled 1)
- Duplication of blocks $\rightarrow$ additional indices to differentiate them
- Can be used to check e.g. mutual excusion constraints
- Size increase with the number of different automata

Feasibility of temporal automata

Theorem

A temporal automaton is feasible on a single-processor computer if, and only if, there is a solution to the following equations

$$\forall n \text{ arc}, \quad \sum_{b \in \mathcal{B}_n} b_n \leq |I_n| \tag{1}$$

$$\forall b \text{ block}, \quad \sum_{n \in \mathcal{A}_b} b_n = ||b|| \tag{2}$$

- It is immediate that a solution on the automaton gives a solution to every trace,
- But the opposite is more difficult to prove!
- So steps for feasibility analysis:
  1. Convert TT automatons into temporal automatons
  2. Combine them using synchronized product
  3. Use network flow and this theorem to perform analysis