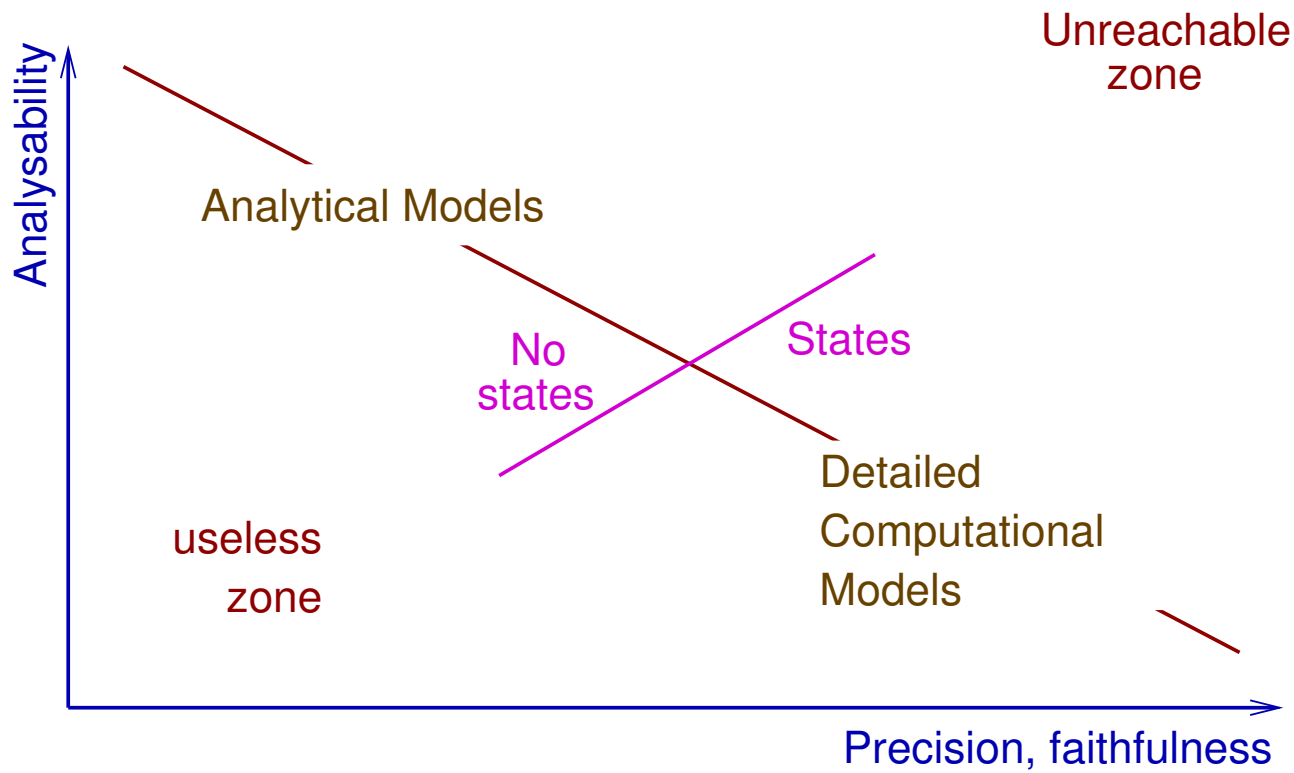
Performance evaluation of components using a granularity-based interface between real-time calculus and timed automata

Karine Altisen Yanhong Liu Matthieu Moy

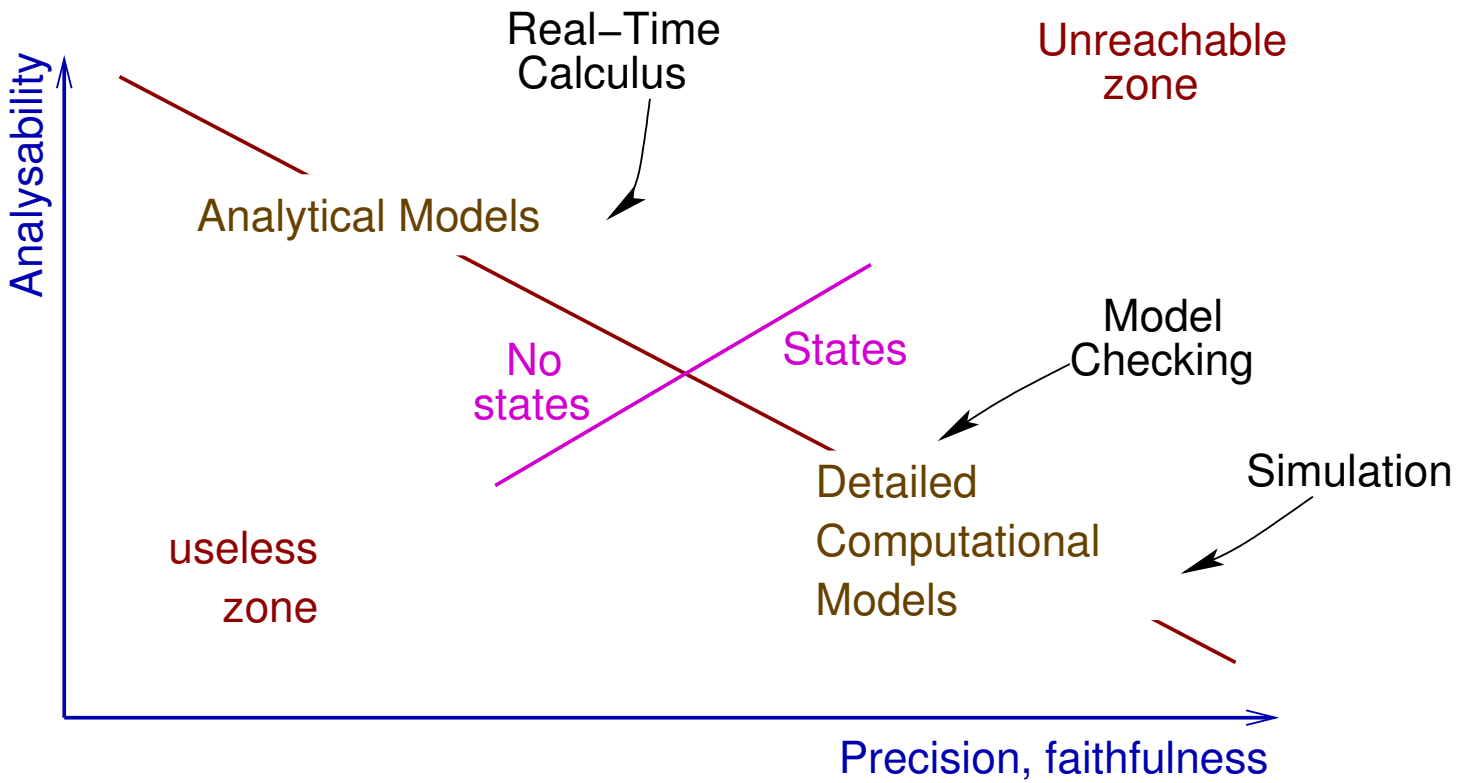
Verimag (Grenoble INP)
Grenoble
France

QAPL, 28 March 2010

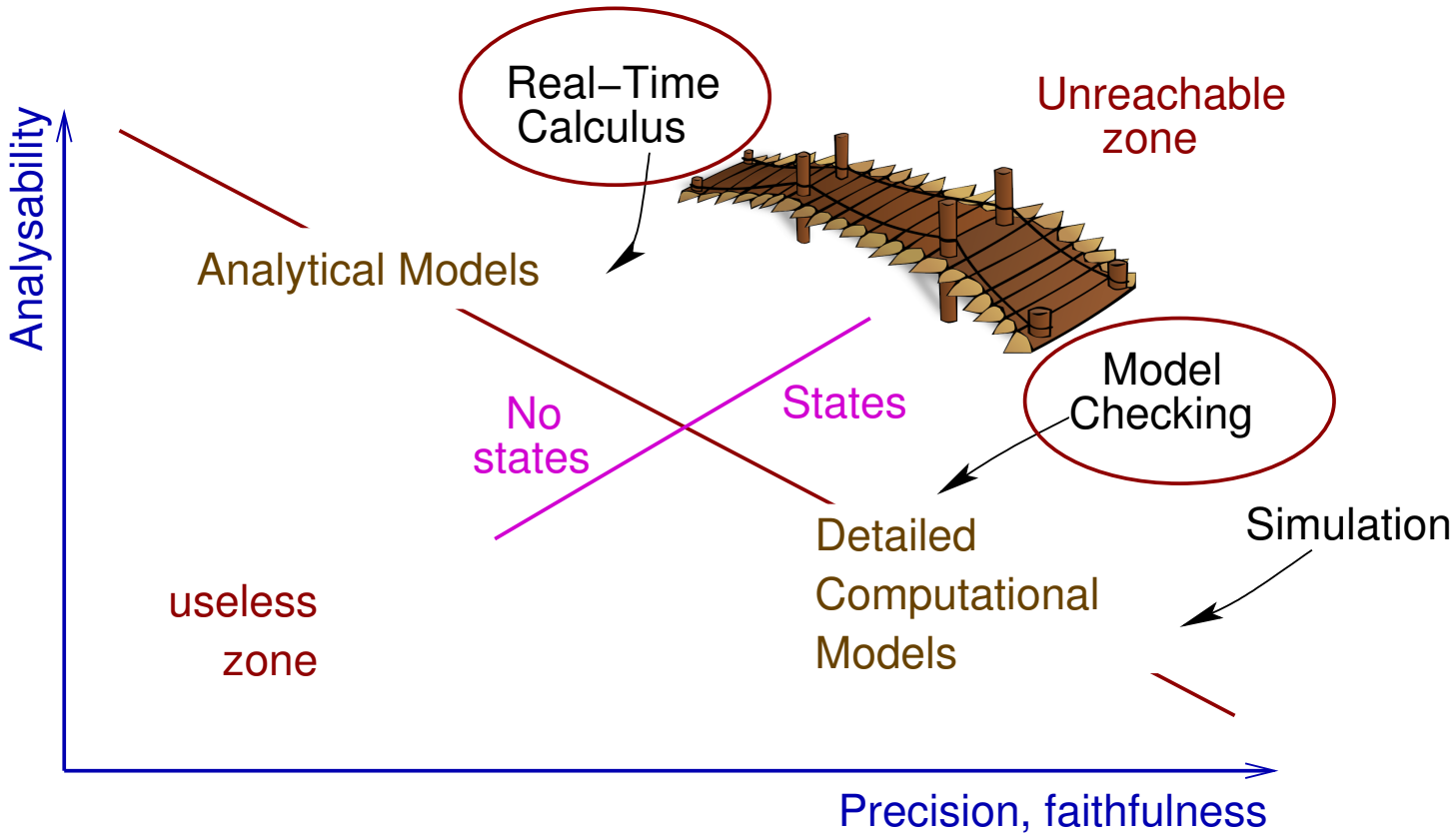
Models for Performance Analysis



Models for Performance Analysis

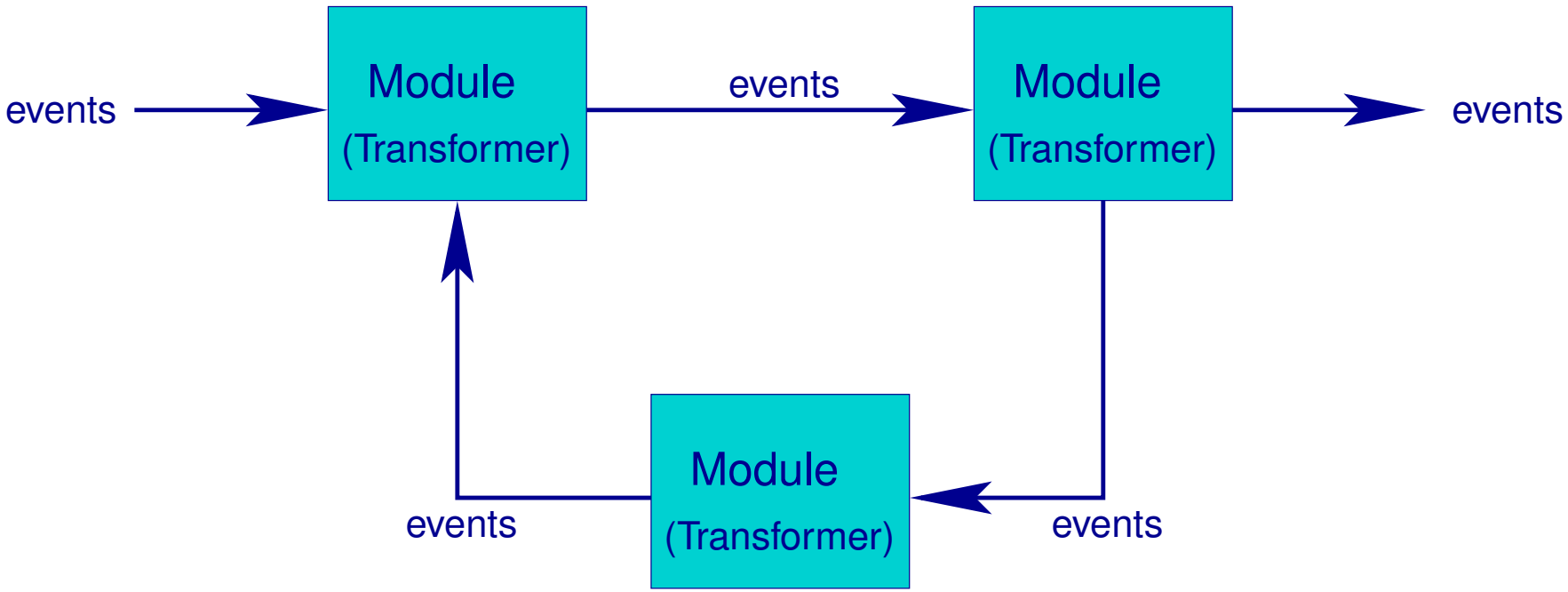


Models for Performance Analysis

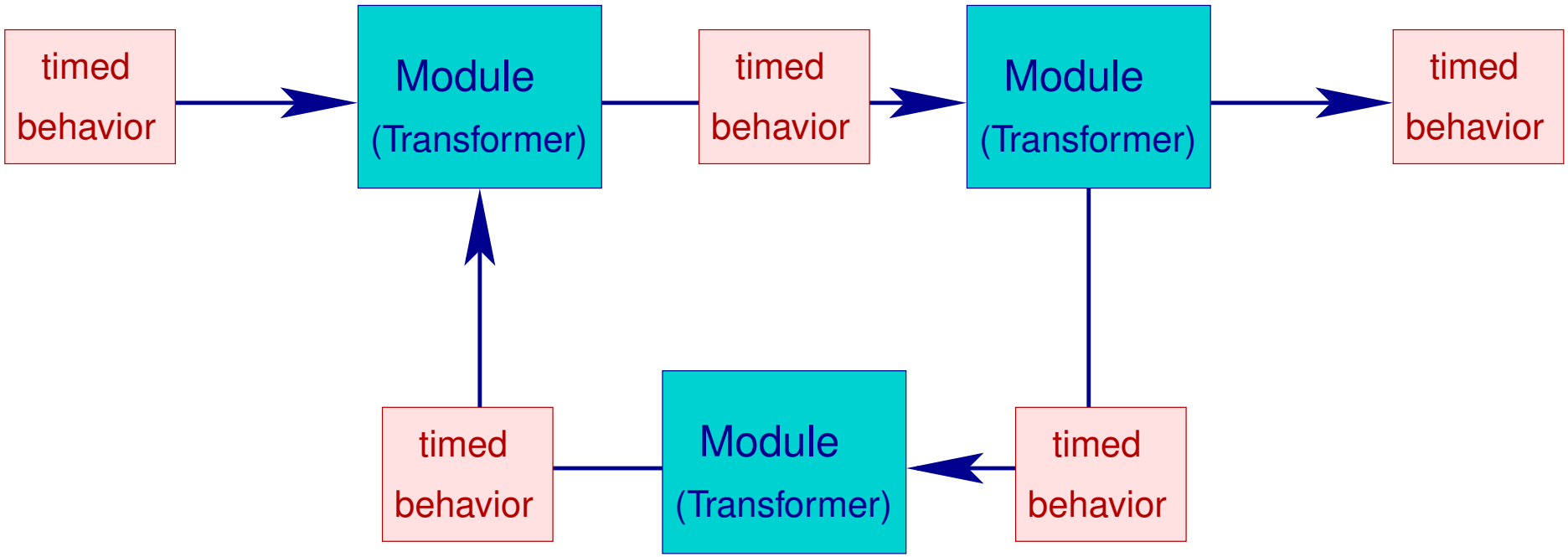


Summary

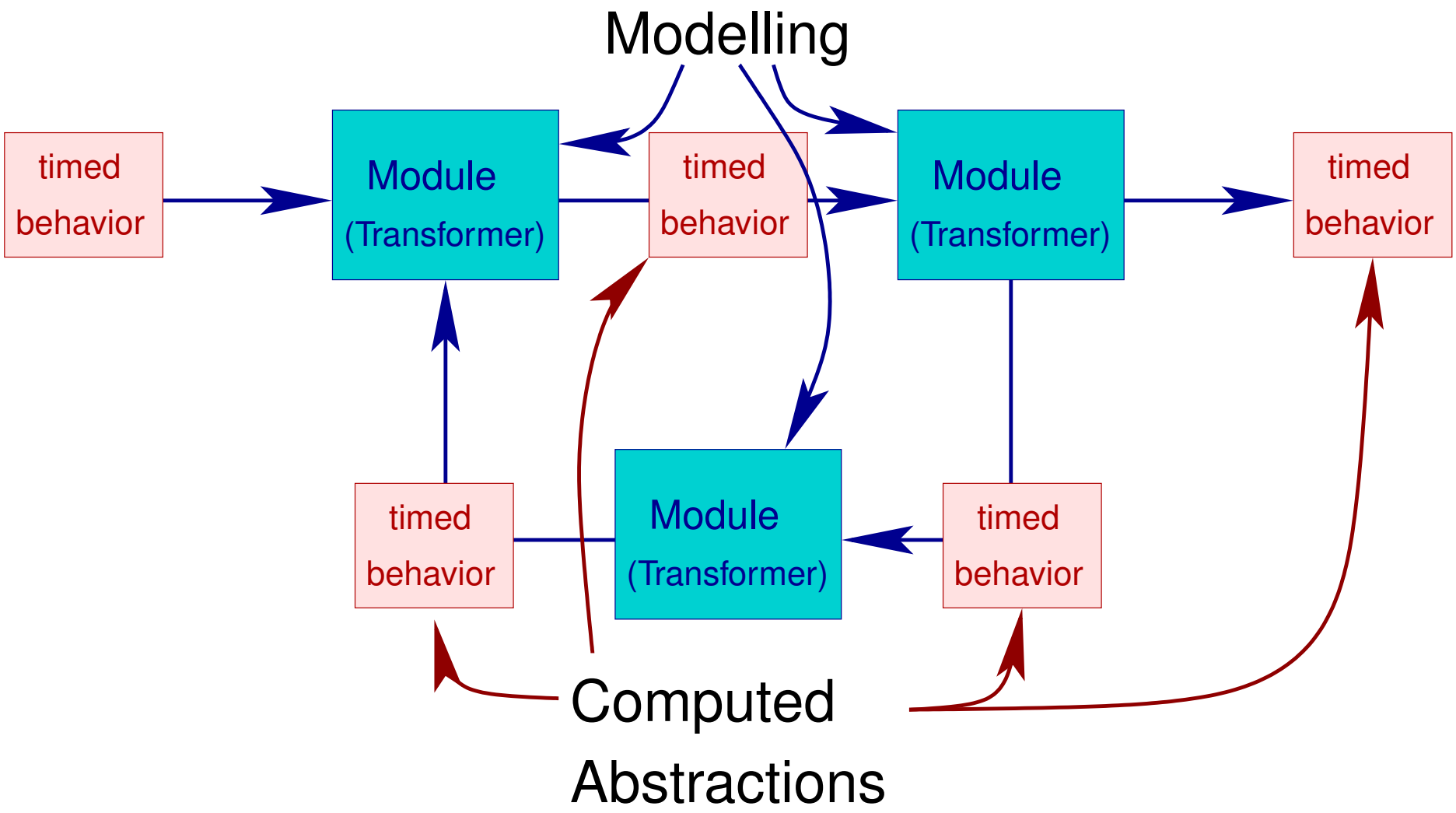
Modular Performance Analysis (MPA): The Big Picture



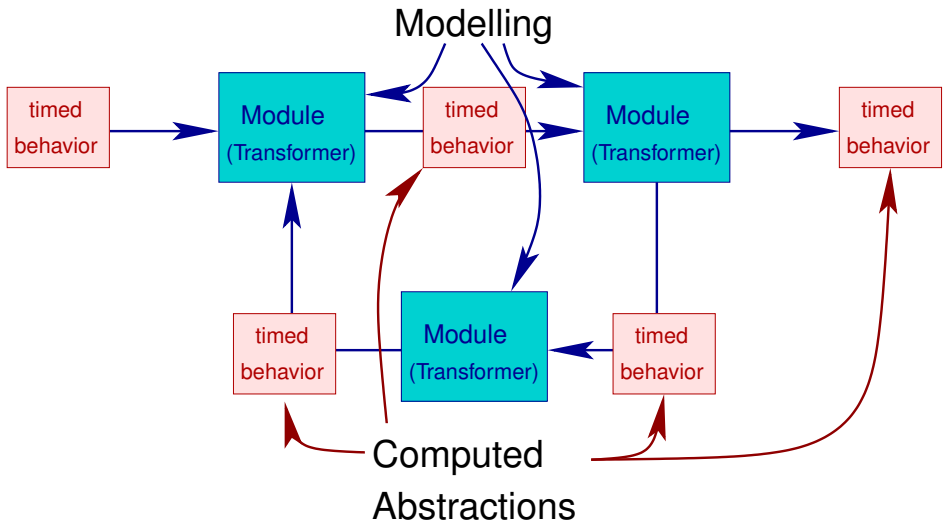
Modular Performance Analysis (MPA): The Big Picture



Modular Performance Analysis (MPA): The Big Picture

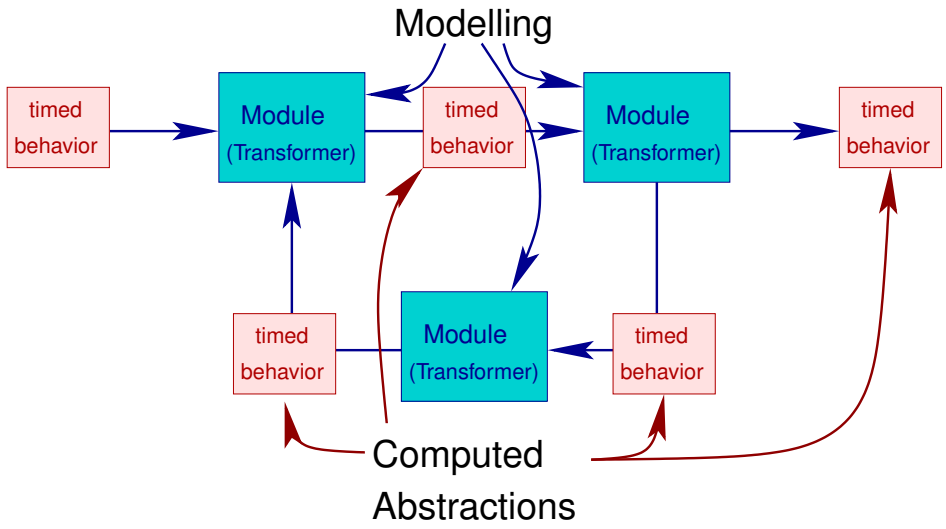


Modular Performance Analysis (MPA)



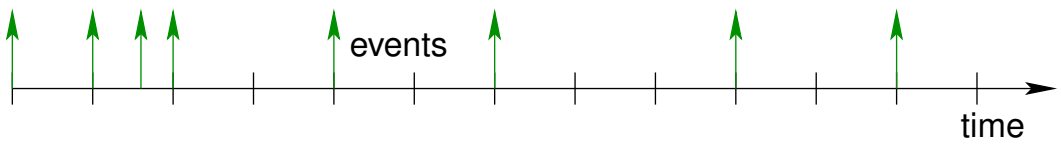
- What can “Timed Behavior” be?
  - Number of events per time unit?
  - Bounds for number of events?

Modular Performance Analysis (MPA)



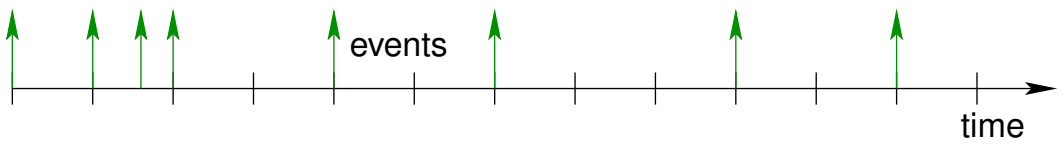
- What can “Timed Behavior” be?
  - Number of events per time unit?
  - Bounds for number of events?
  - MPA uses “Arrival Curves”.
- “Modules” = Arrival Curve transformers:
  - FIFO + processing element (defined by “service curves”)
  - Can also be a program/state machine/...

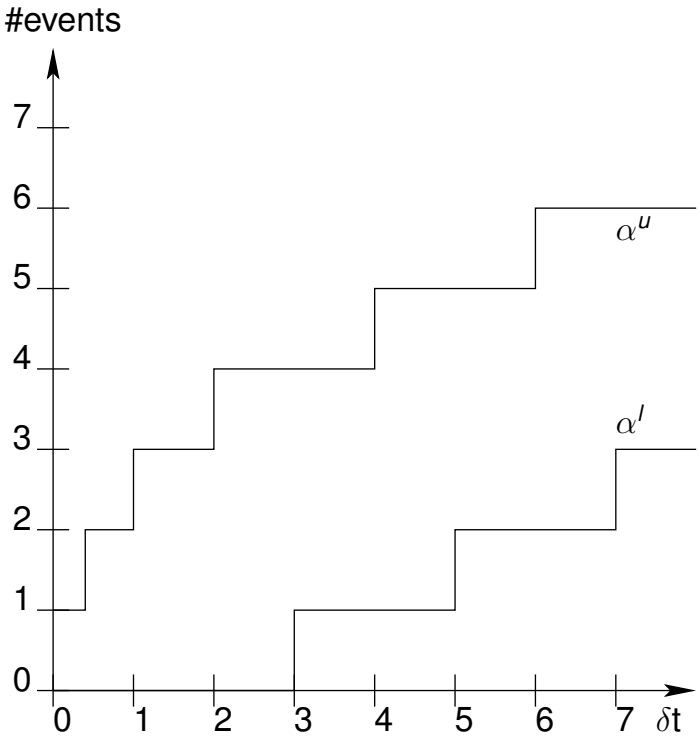
Arrival Curves in Real-Time Calculus (RTC)



- $\alpha^u(\delta)$: max number of events in any window of length δ.
- $\alpha^l(\delta)$: min number of events in any window of length δ.

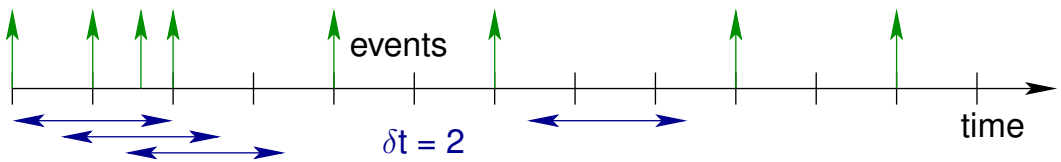
Arrival Curves in Real-Time Calculus (RTC)

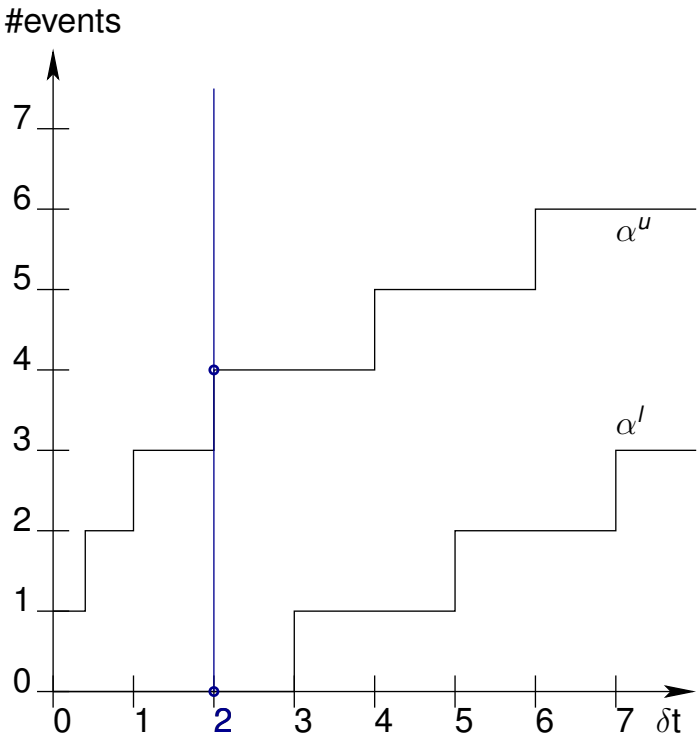




- $\alpha^u(\delta)$: max number of events in any window of length δ.
- $\alpha^l(\delta)$: min number of events in any window of length δ.

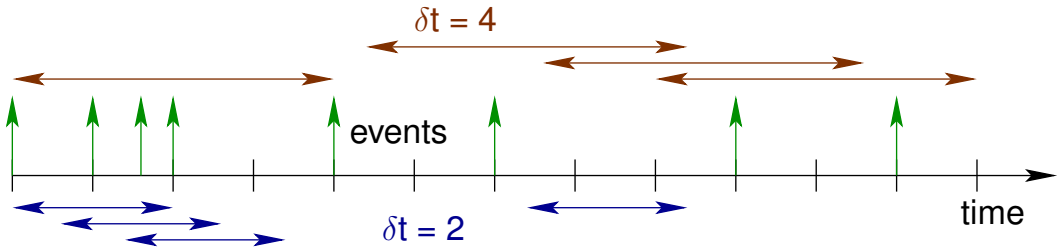
Arrival Curves in Real-Time Calculus (RTC)

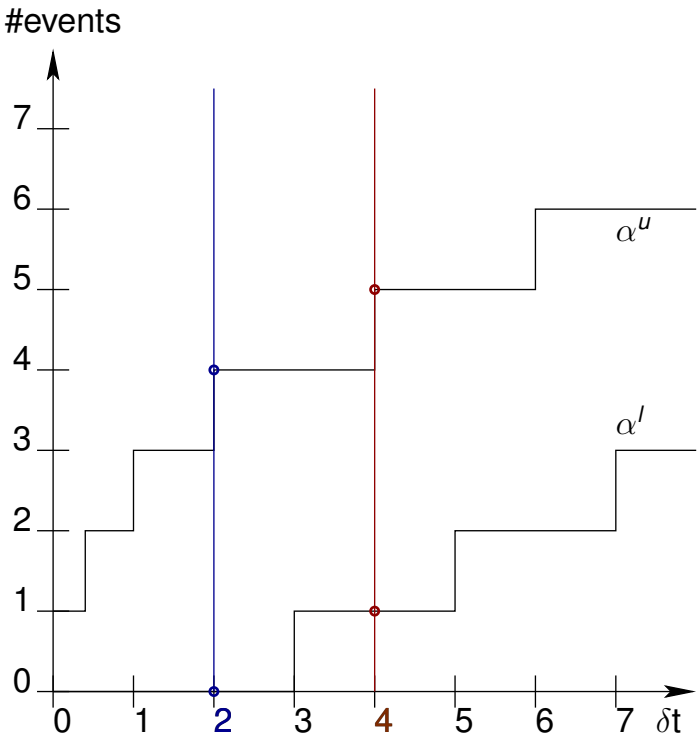




- $\alpha^u(\delta)$: max number of events in any window of length δ.
- $\alpha^l(\delta)$: min number of events in any window of length δ.

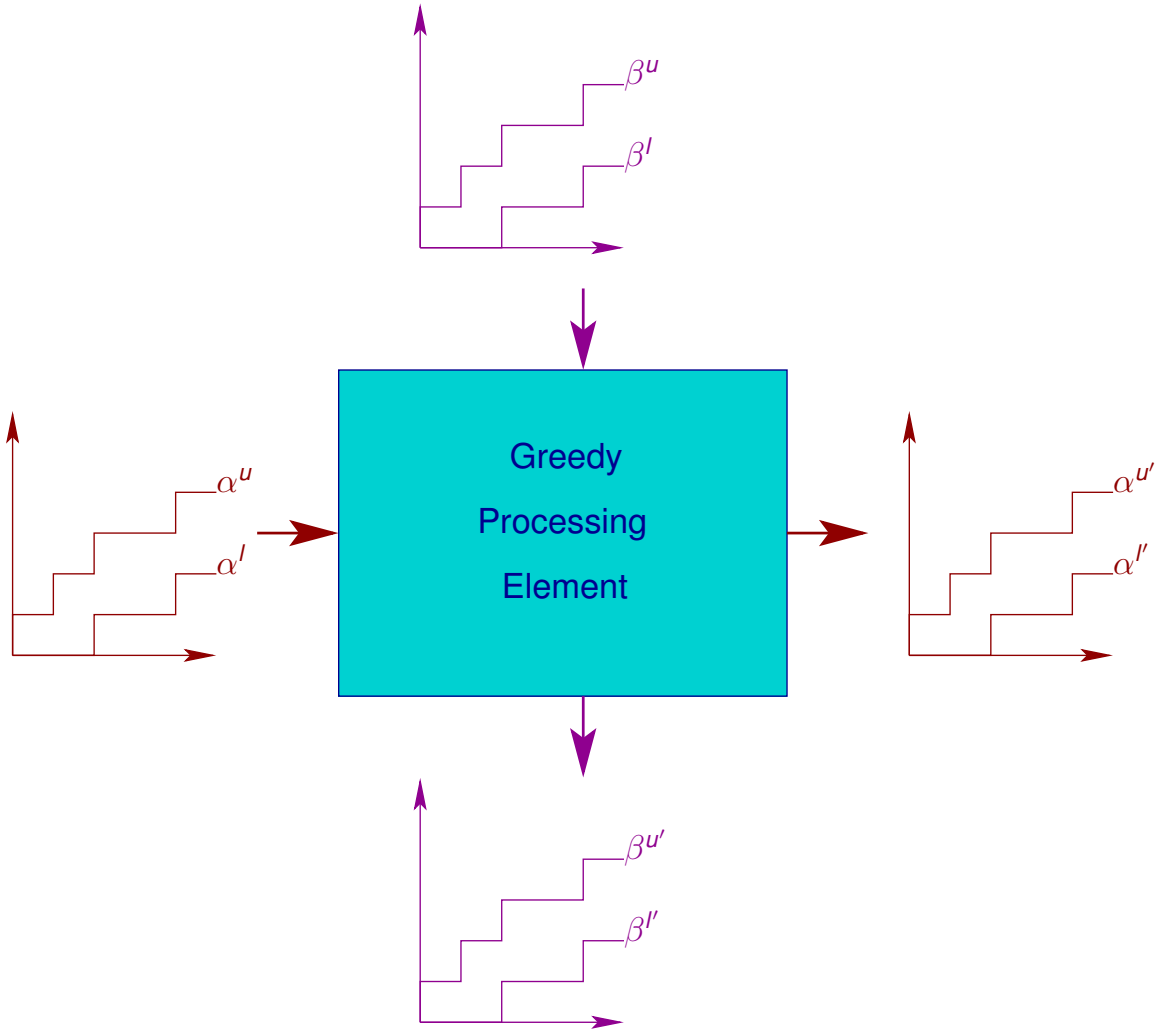
Arrival Curves in Real-Time Calculus (RTC)



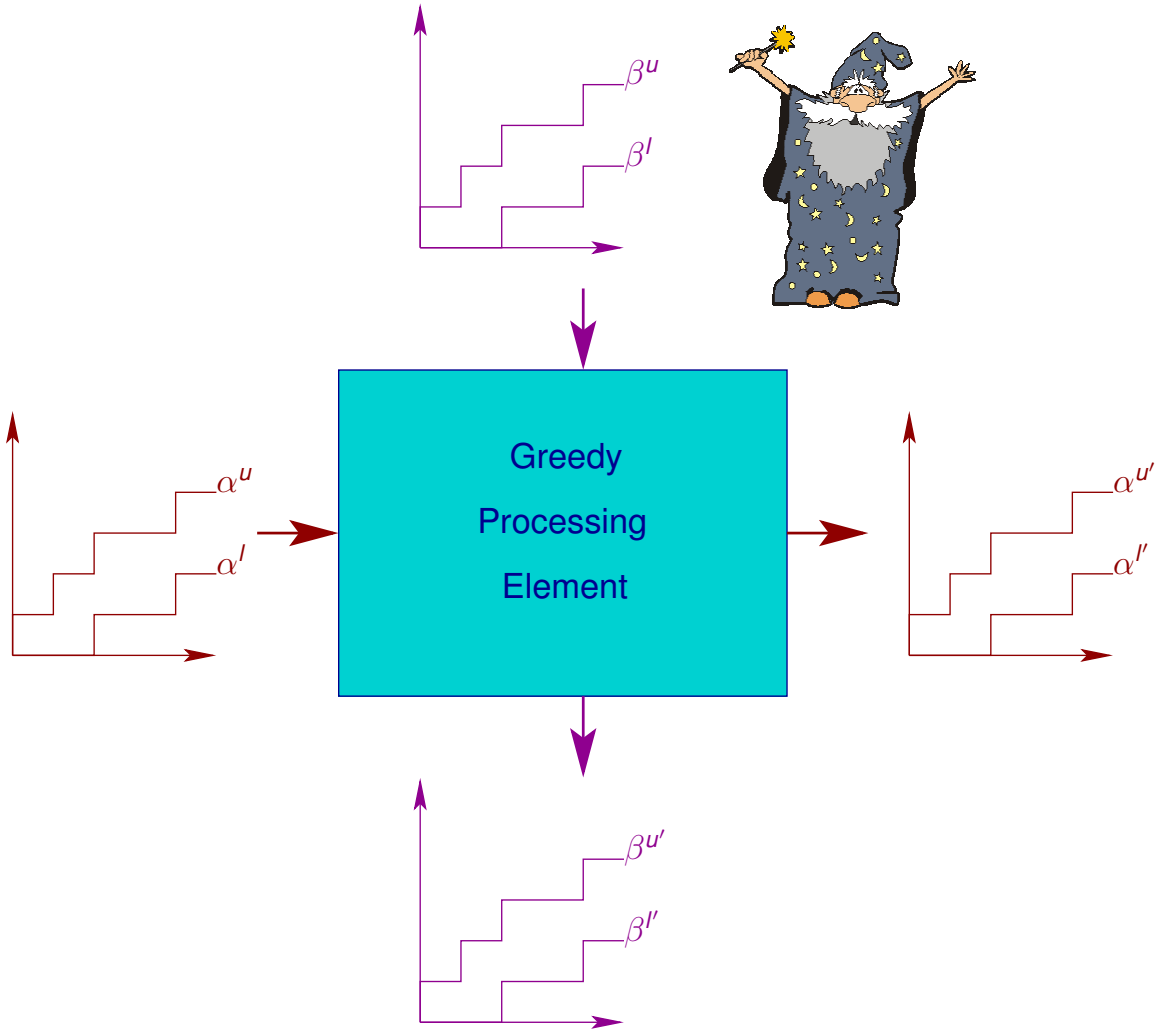


- $\alpha^u(\delta)$: max number of events in any window of length δ.
- $\alpha^l(\delta)$: min number of events in any window of length δ.

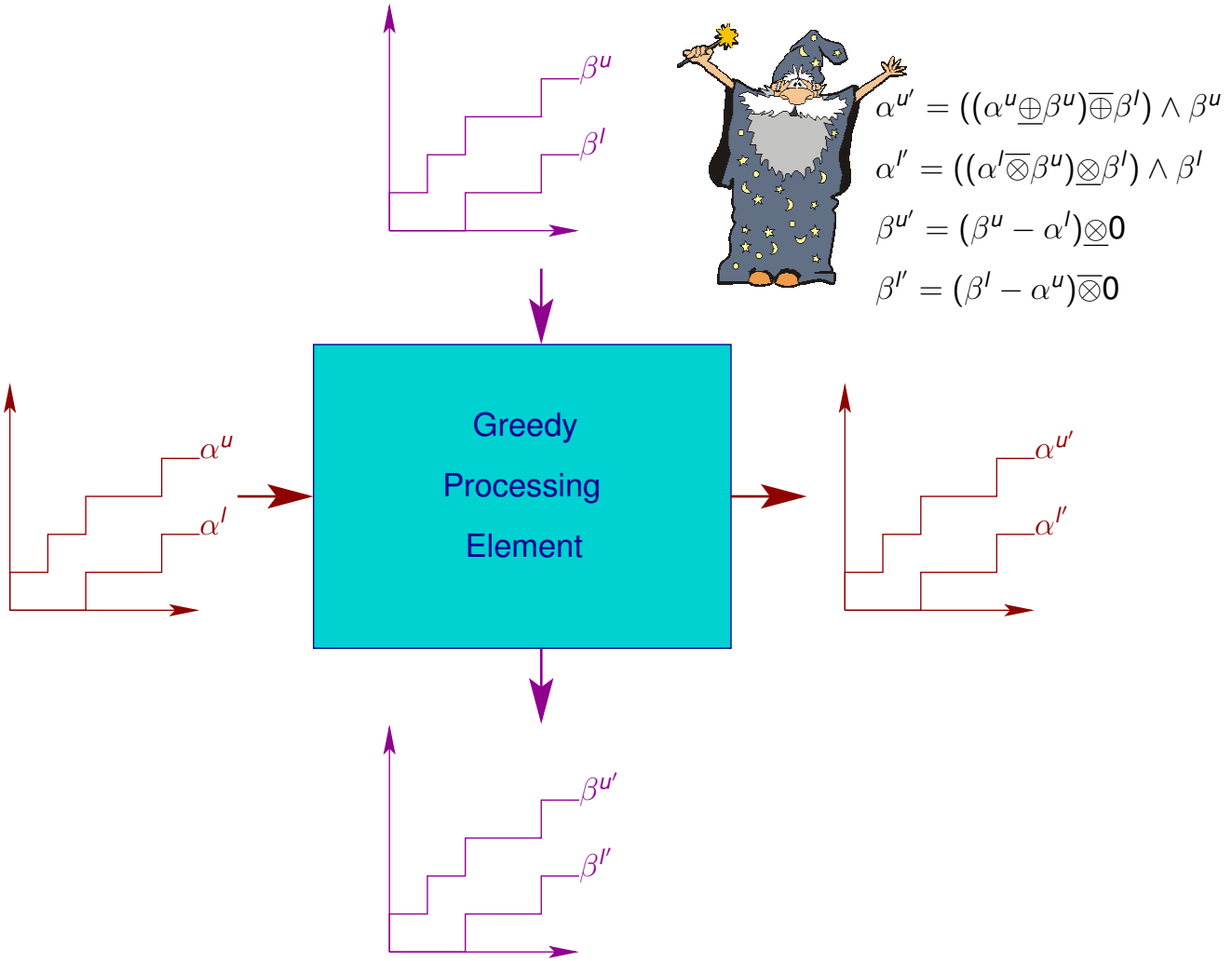
Service Curves



Service Curves



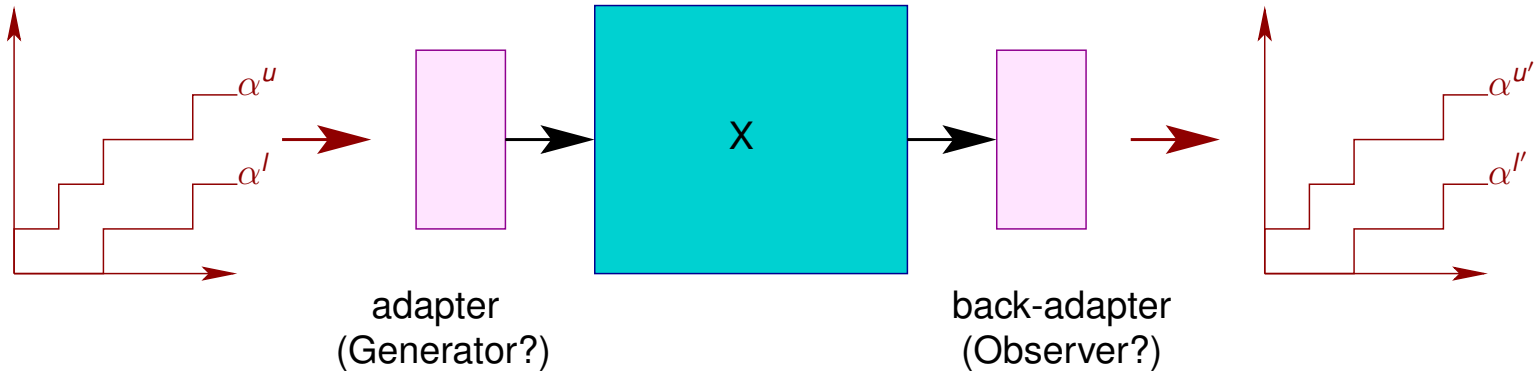
Service Curves



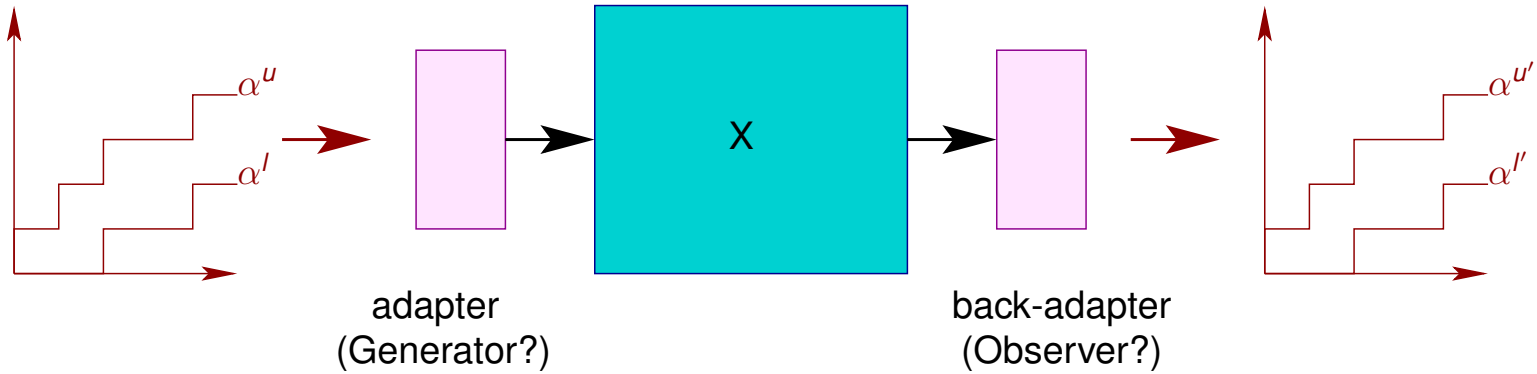
Real-Time Calculus (RTC): pros and cons

- Nice things with RTC
  - Can model: event streams, simple scheduling policies
  - Scales up nicely
  - Exact hard bounds
- Less nice thing with RTC
  - Limited expressiveness: Cannot model state-based behaviors

Allowing more Complex Modules in MPA

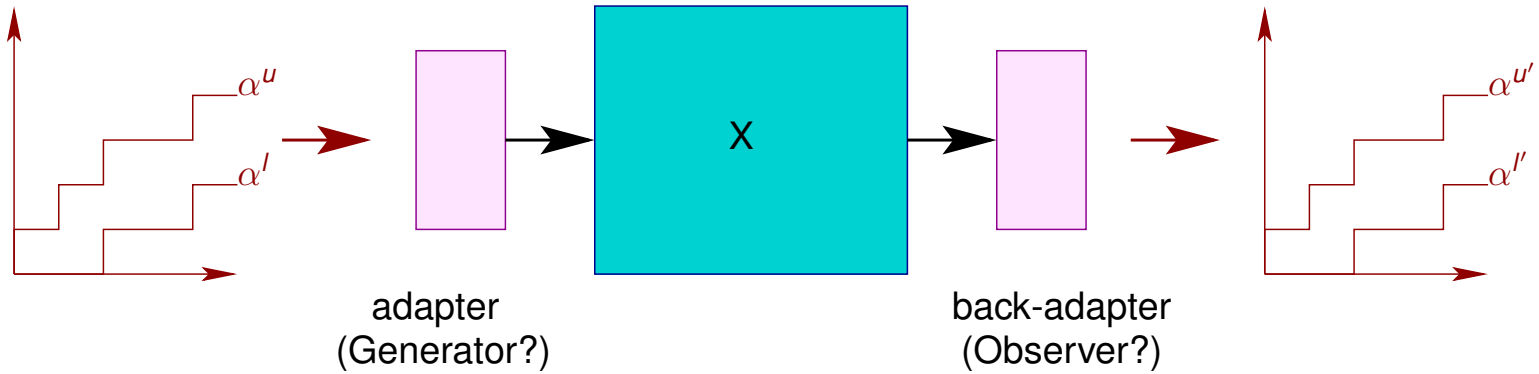


Allowing more Complex Modules in MPA



- X = Arbitrary program $\Rightarrow$ testing [L. Phan, ETHZ]
- X = Lustre $\Rightarrow$ Abstract Interpretation, SMT Solving [Verimag, accepted at ECRTS'2010]
- X = Timed Automata $\Rightarrow$ model-checking
  [K. Lampka, ETHZ; CATS tool, Uppsala; Verimag]

Allowing more Complex Modules in MPA



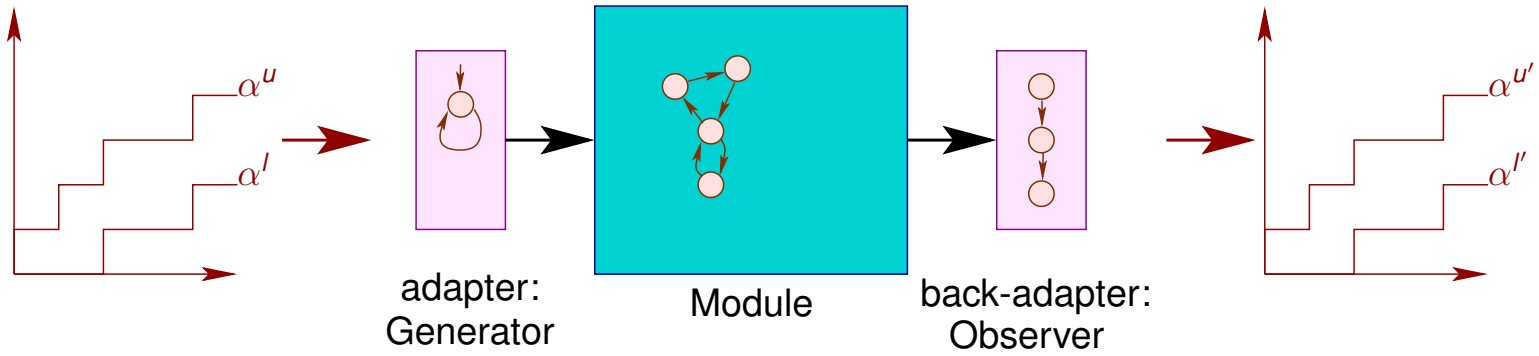
- X = Arbitrary program $\Rightarrow$ testing [L. Phan, ETHZ]
  Under-approximation of upper-bounds $\rightsquigarrow$ Unsound
- X = Lustre $\Rightarrow$ Abstract Interpretation, SMT Solving [Verimag, accepted at ECRTS'2010]
- X = Timed Automata $\Rightarrow$ model-checking
  [K. Lampka, ETHZ; CATS tool, Uppsala; Verimag]
  $\rightsquigarrow$ State explosion

This talk

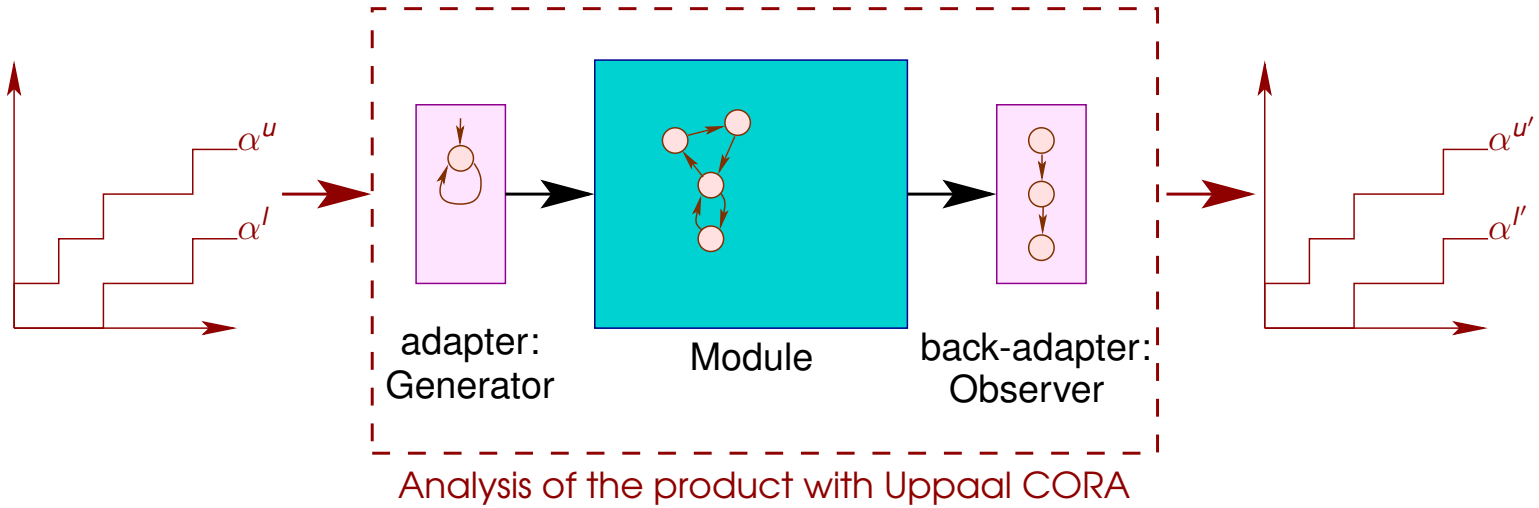
- Connection of Real-Time Calculus (RTC) and Timed Automata (based on previous works: CATS tool from Uppsala)
- Abstractions based on granularity: reduce state-explosion at the cost of accuracy.
- Application: systems with a notion of mode (typically, energy-saving slow or sleeping modes, ...)

Summary

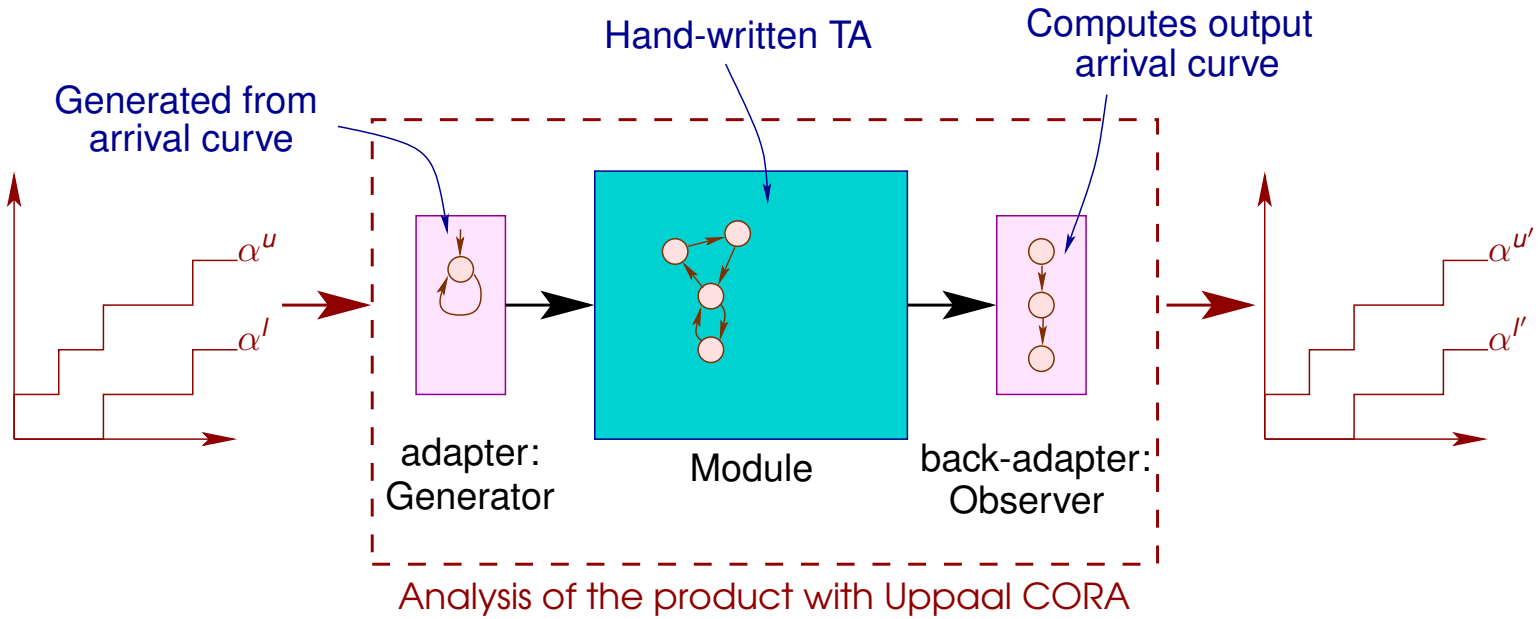
Connecting RTC and TA



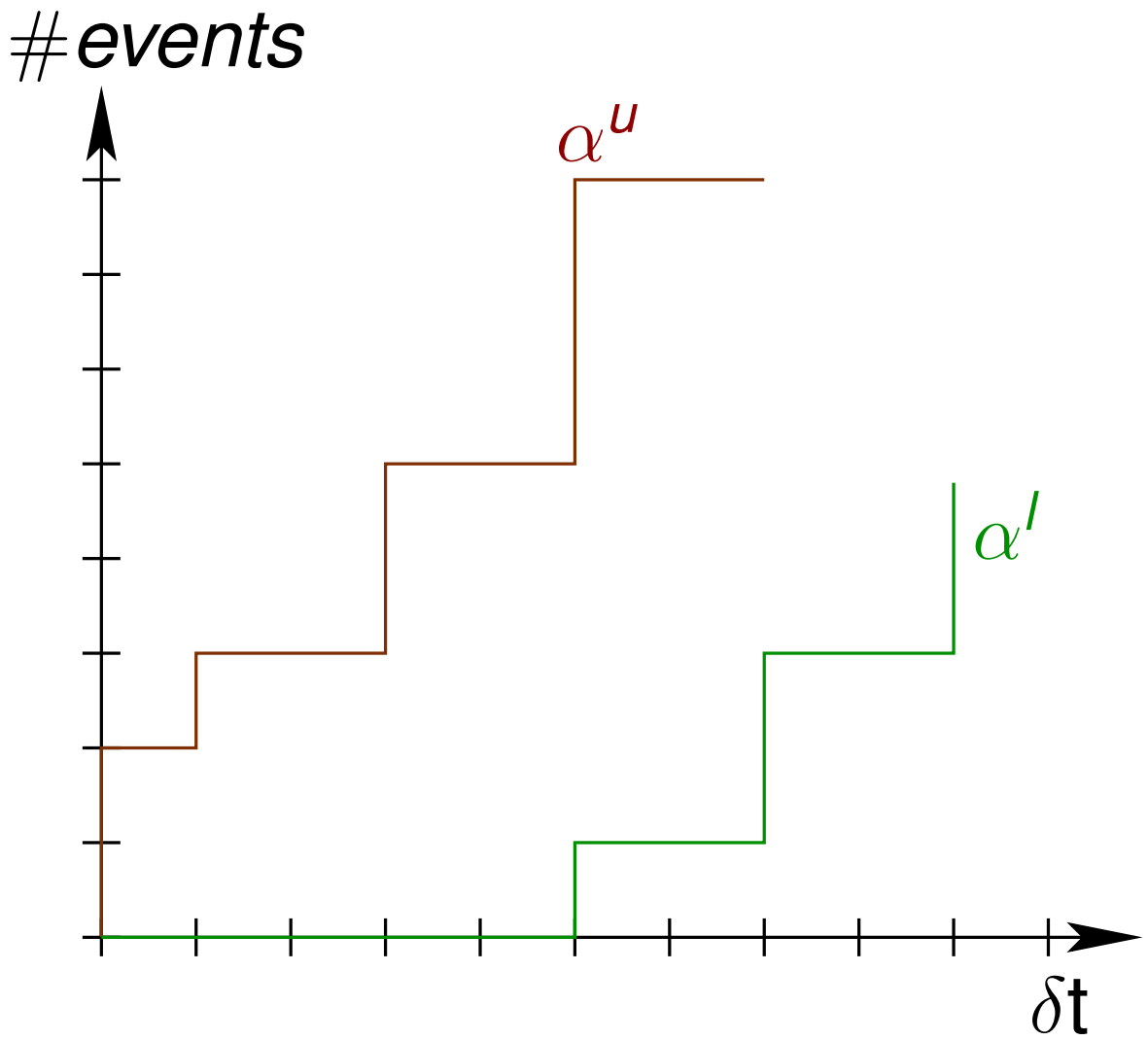
Connecting RTC and TA



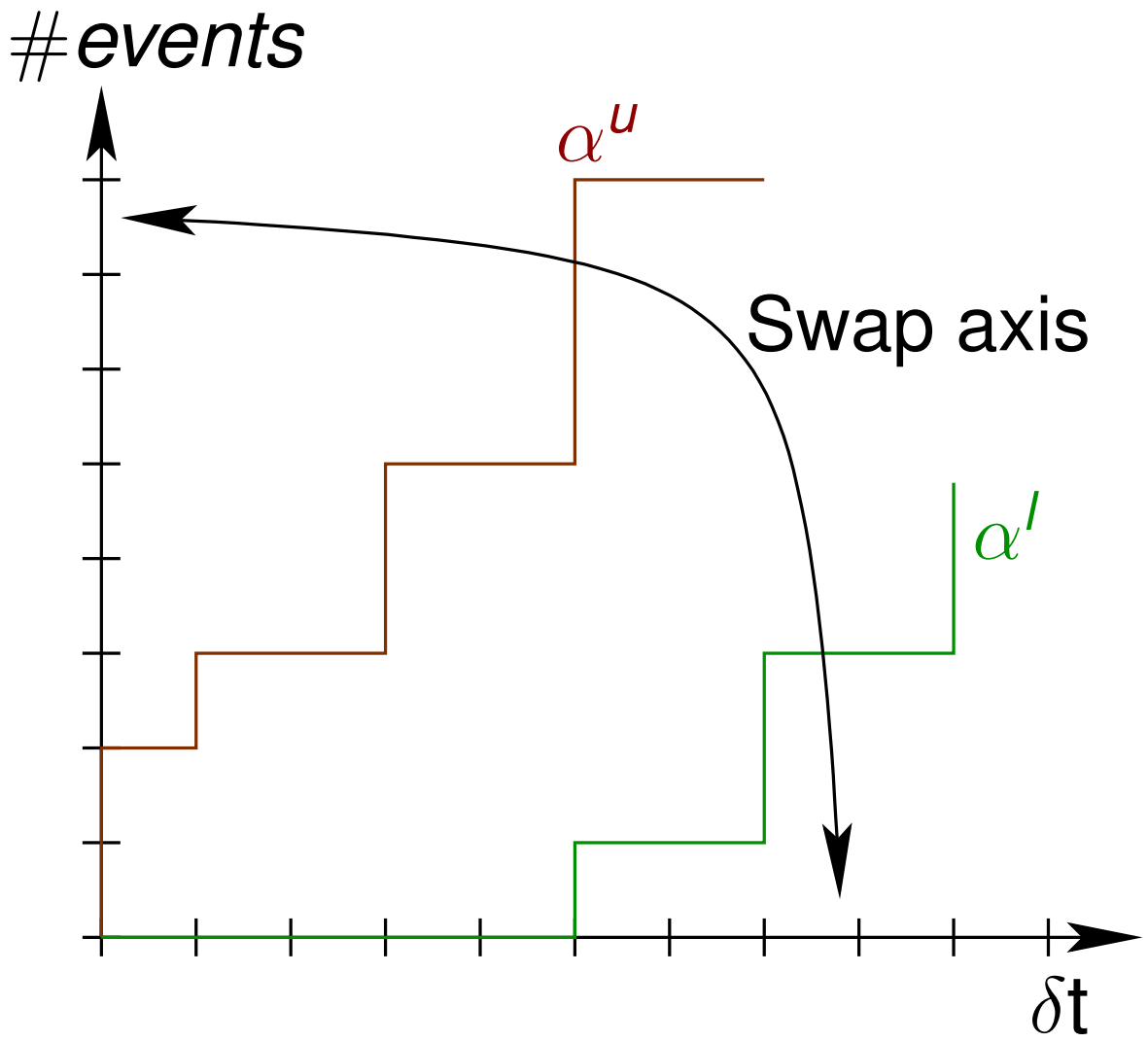
Connecting RTC and TA



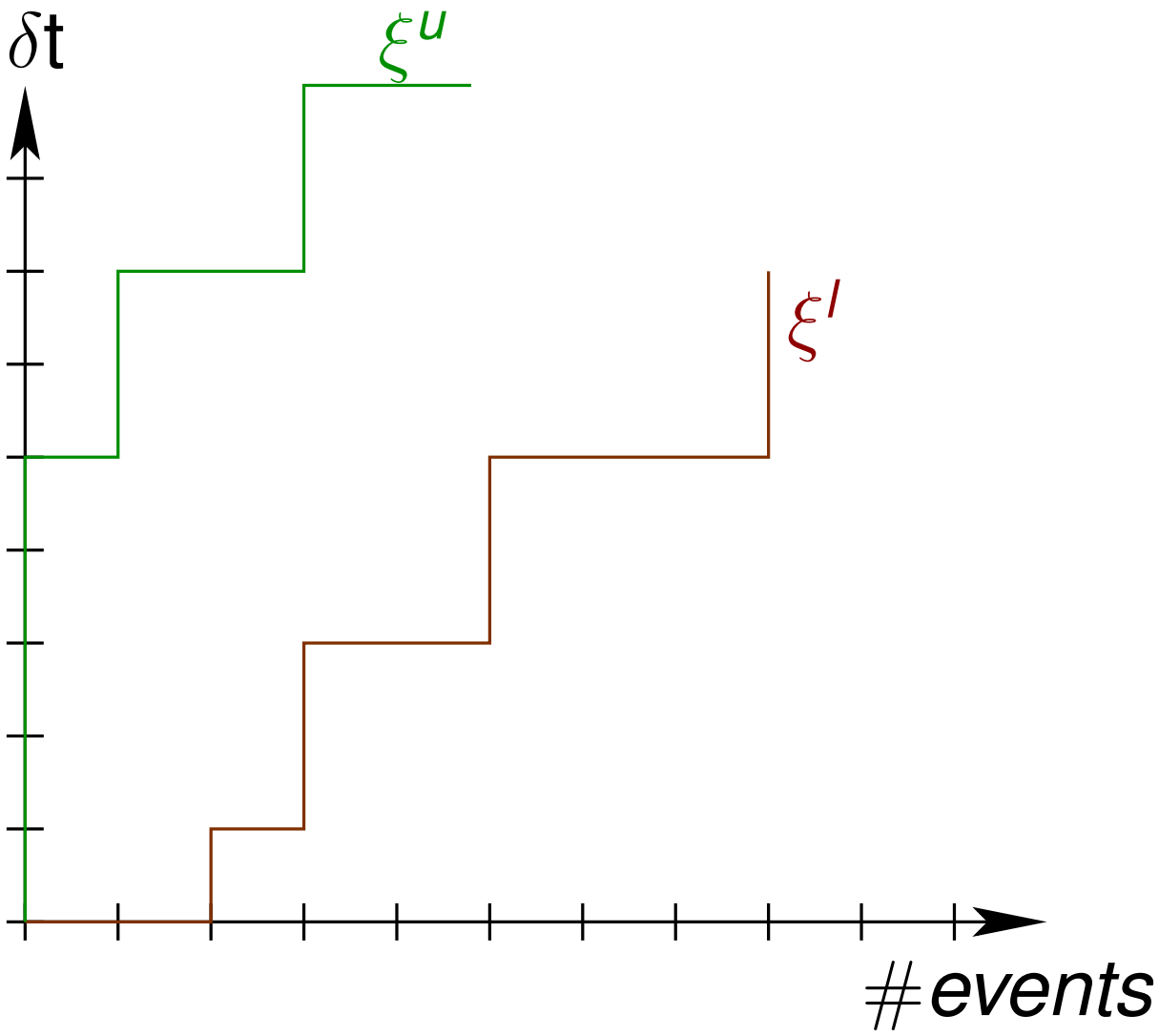
Two Views on Arrival Curves



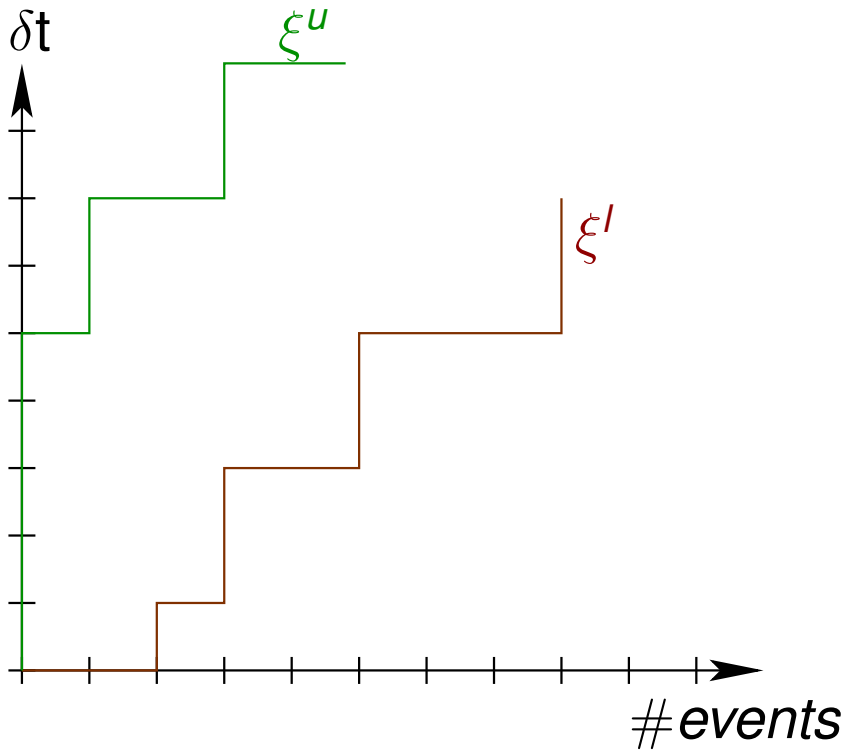
Two Views on Arrival Curves



Two Views on Arrival Curves

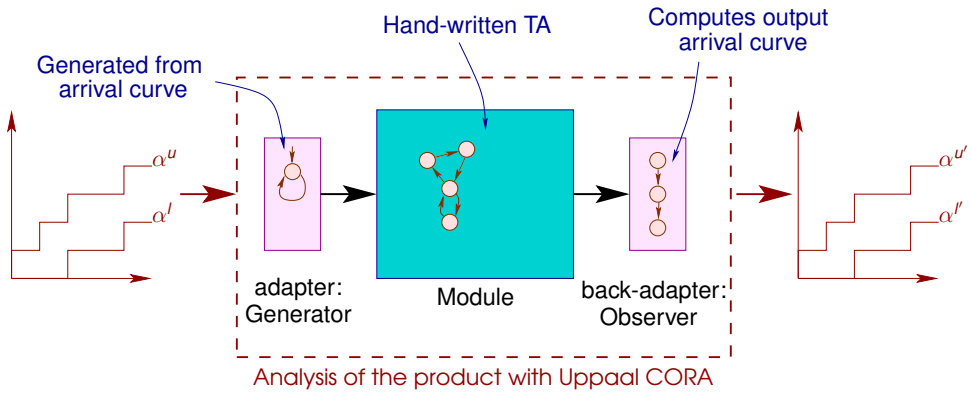


Two Views on Arrival Curves



- $\alpha^u(\delta)$, $\alpha^l(\delta)$ = min/max number of events in a time interval of length δ.
- $\xi^u(K)$, $\xi^l(K)$ = min/max length of a time interval containing K consecutive events.

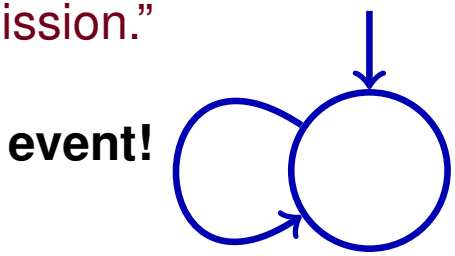
Generator



- Goal of the generator:
  - Parametrized by arrival curve (ξ^u, ξ^l)
  - Can generate any event stream conforming to (ξ^u, ξ^l)
- Idea:
  - Prevent from emitting an event until ξ^l allows it
  - Force event emission when ξ^u requires it

Generator: the Automaton

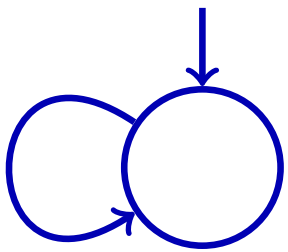
Guard: "ξ^l allows event emission."



Invariant:
"ξ^u doesn't force an event emission."

Generator: the Automaton

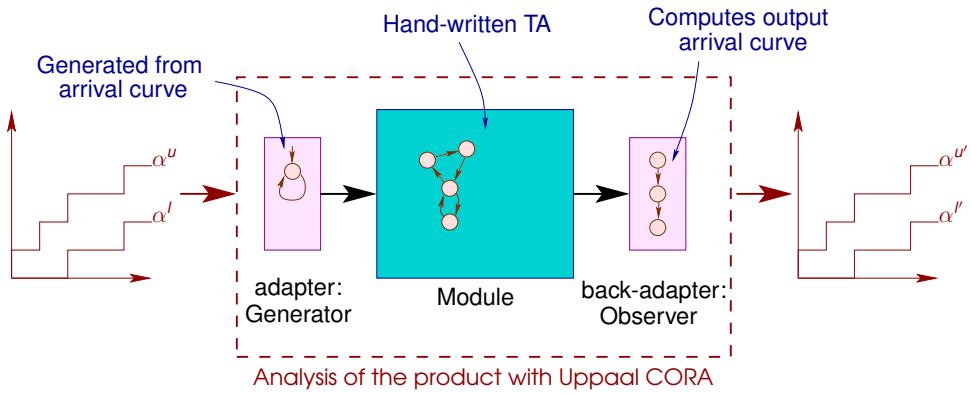
Guard: "ξ^l allows event emission."
i.e. $\mathbf{y}[(\lambda - i + N)\%N] \geq \xi^l(i), \forall i \in [1 : \theta]$
event!
Actions: $\mathbf{y}[\lambda] \rightarrow 0$,
$\lambda \rightarrow (\lambda + 1)\%N$,
$\theta \rightarrow \theta < N ? \theta + 1 : N$



Invariant:
"ξ^u doesn't force an event emission."
i.e. $\mathbf{y}[(\lambda - i + N)\%N] \leq \xi^u(i), \forall i \in [1 : \theta]$

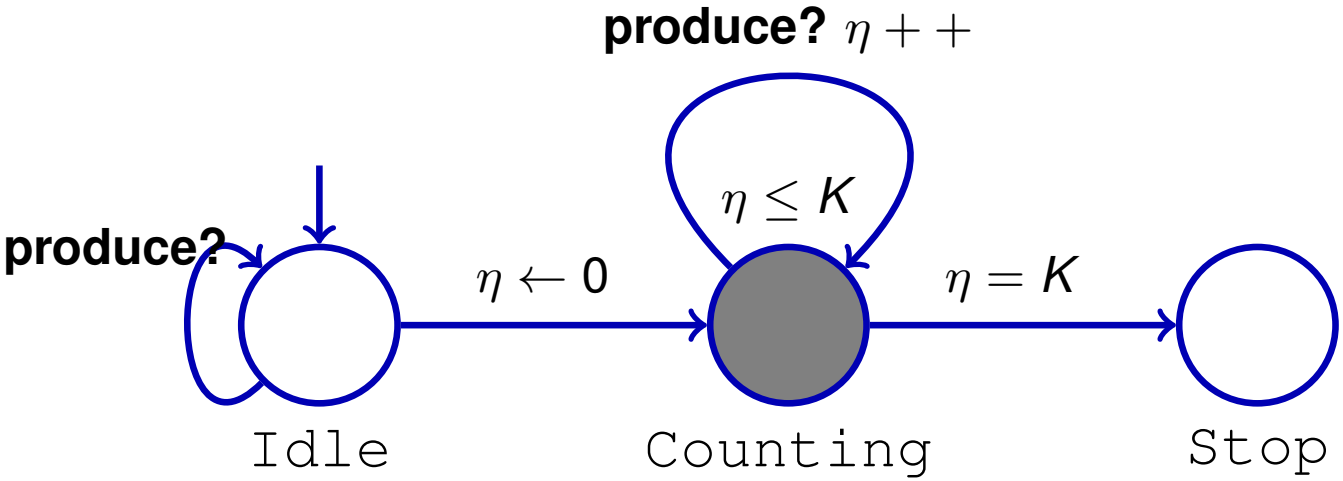
- N: number of points of arrival curves ξ^u, ξ^l,
- $\mathbf{y}$: array (circular buffer) of N clocks,
- λ: index of the current clock in the array.

Observer



- Goal of the observer:
  - find min/max length of time window containing K consecutive events.
- Idea:
  - One counter to count K events
  - Find shortest/longest path using Uppaal CORA

Observer: the automaton



- Cost in “Counting” state : 1 per time unit,
- Uppaal CORA finds minimal cost from “Idle” to “Stop” $\Rightarrow \xi^l(K)$,
- Variant for $\xi^u(K)$ (see paper).

Summary

This section

3 Granularity-based Abstraction
- Idea: Group Events into “Coarse Events”
- Automatic Abstraction of Modules

Where does State Explosion Come From?

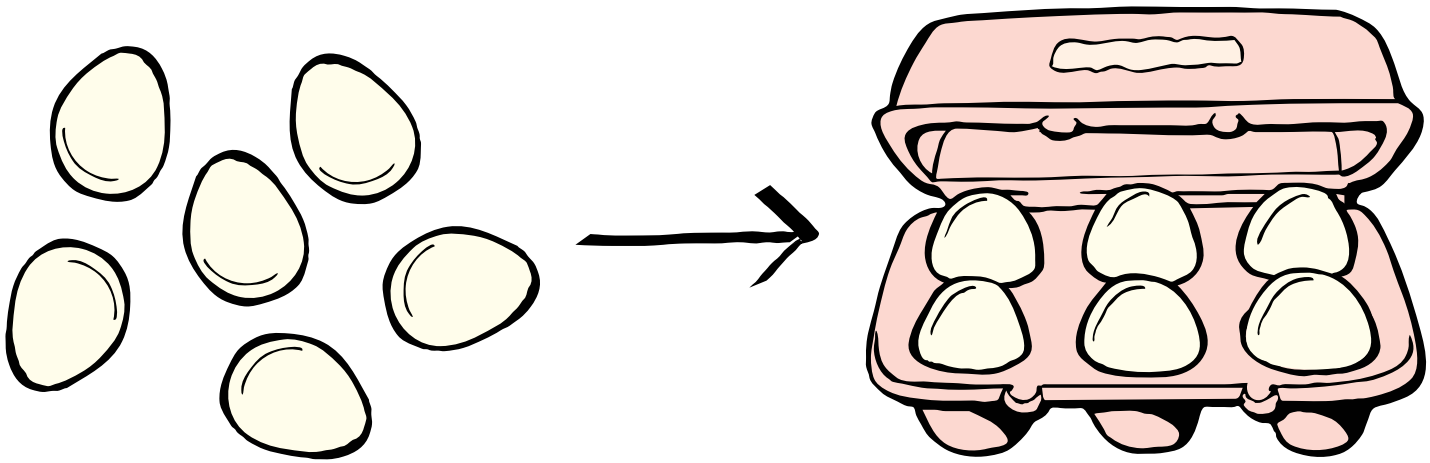
- Generator: N clocks (N = number of points of arrival curve)
- Observer: 1 clock, 1 counter bounded by N
- Module: counters to count events.

Where does State Explosion Come From?

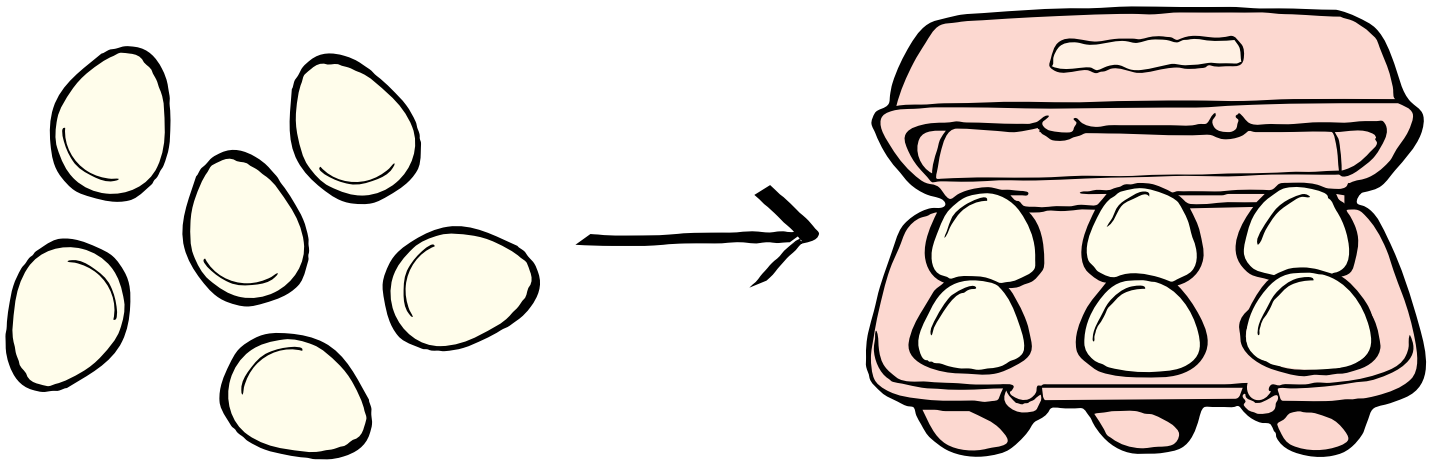
- Generator: N clocks (N = number of points of arrival curve)
- Observer: 1 clock, 1 counter bounded by N
- Module: counters to count events.

- Shorter curves $\Rightarrow$ fewer clocks
- Fewer event occurrences $\Rightarrow$ fewer possible counter values

Grouping Events



Grouping Events

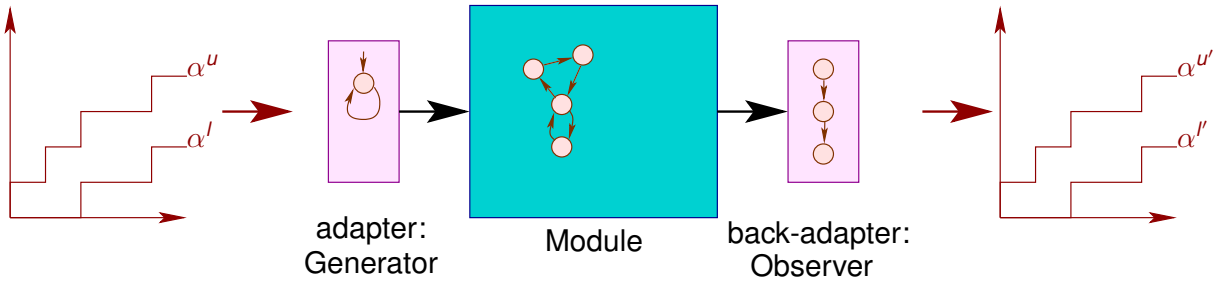


N eggs $\rightsquigarrow$ 1 egg carton

N fine events $\rightsquigarrow$ 1 coarse event.

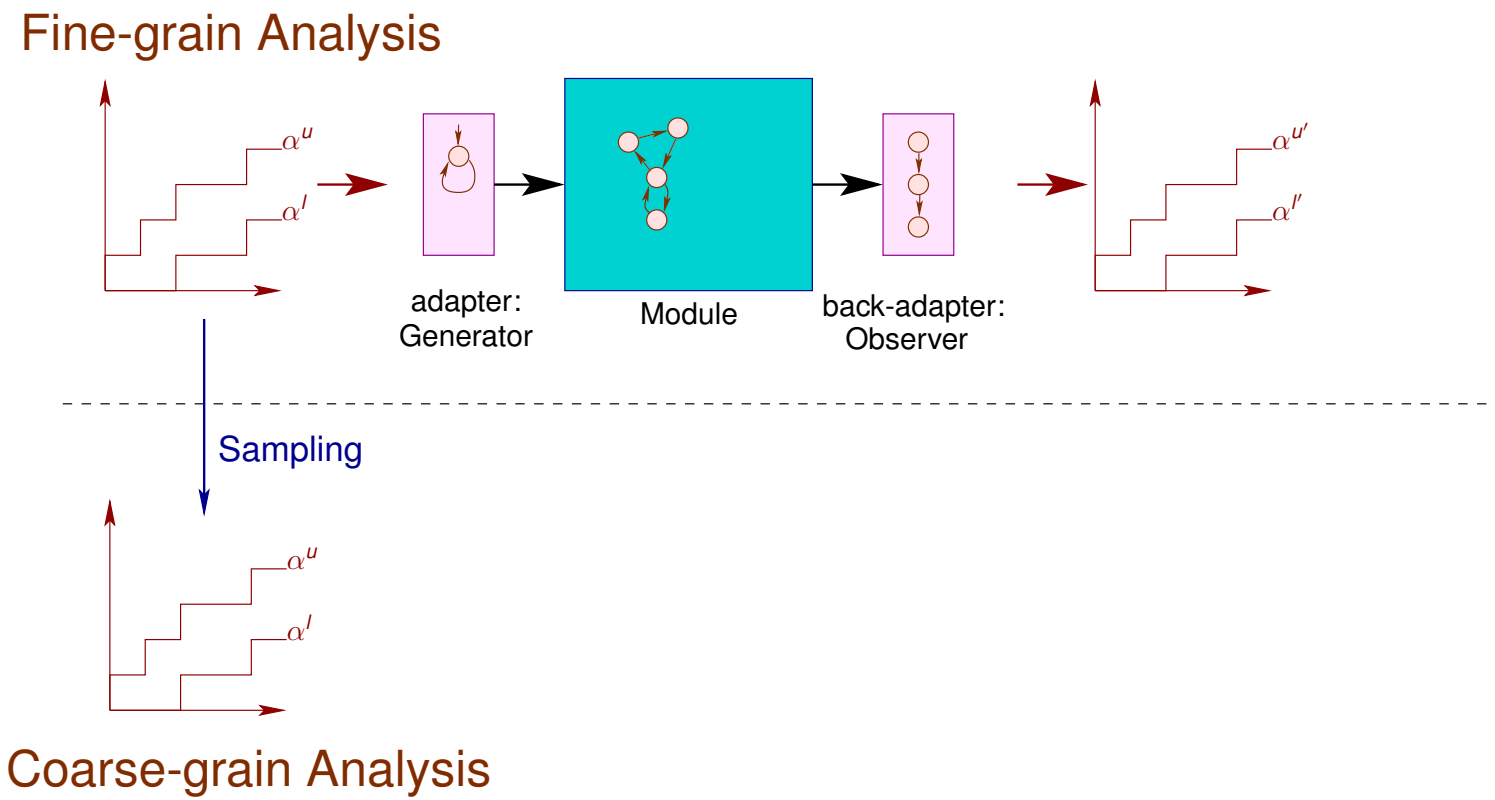
Granularity: The framework

Fine-grain Analysis

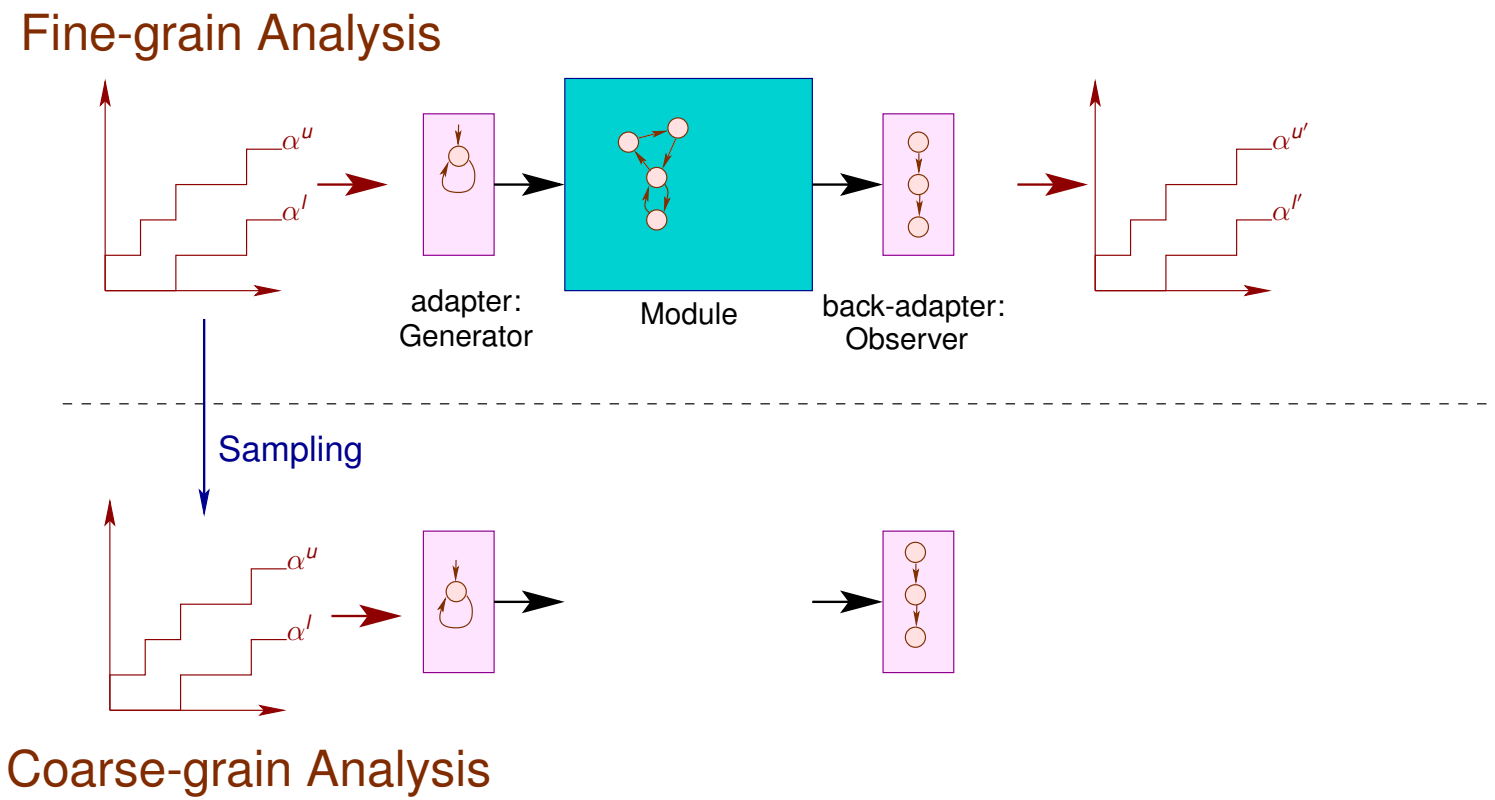


Coarse-grain Analysis

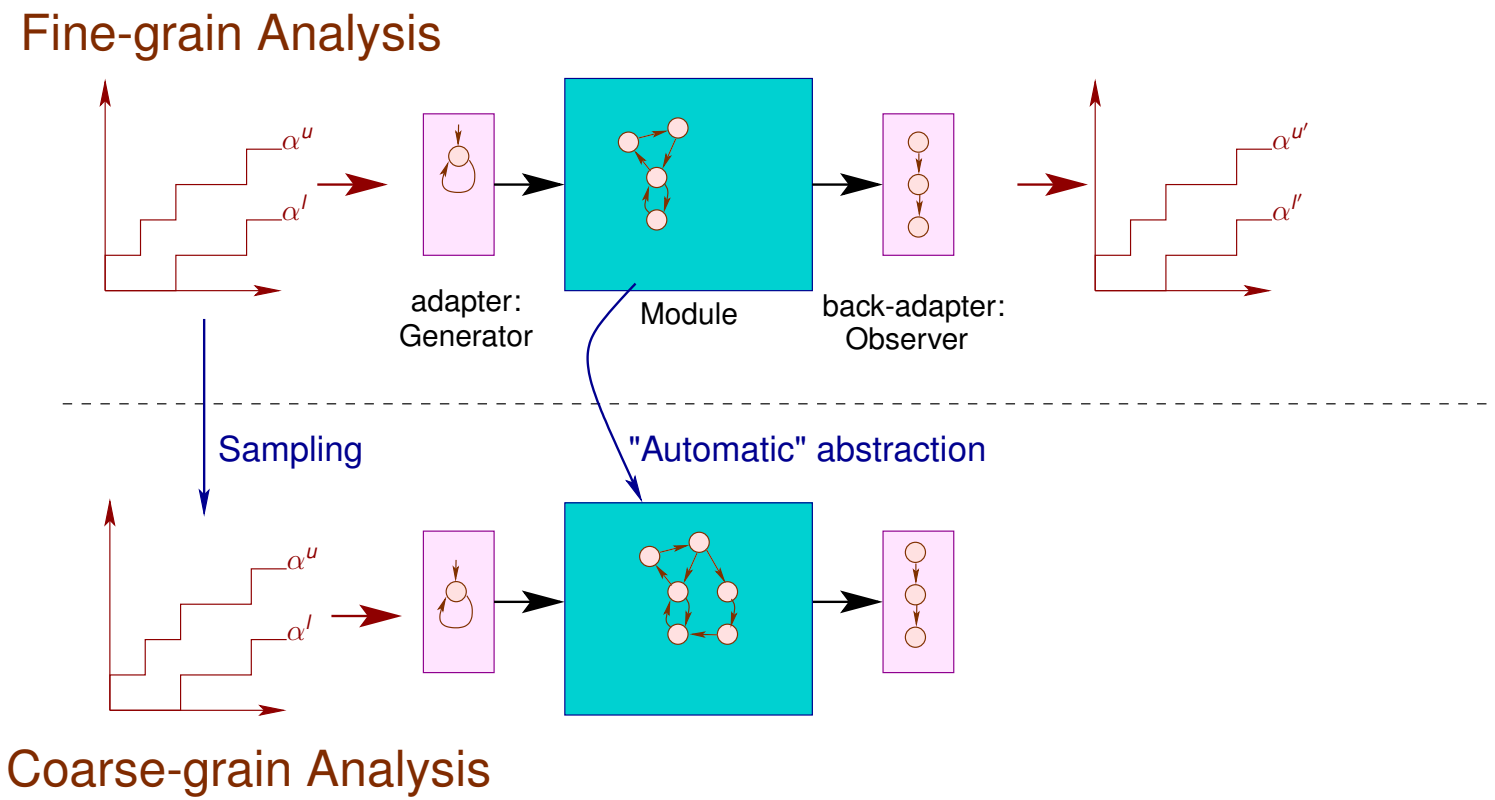
Granularity: The framework



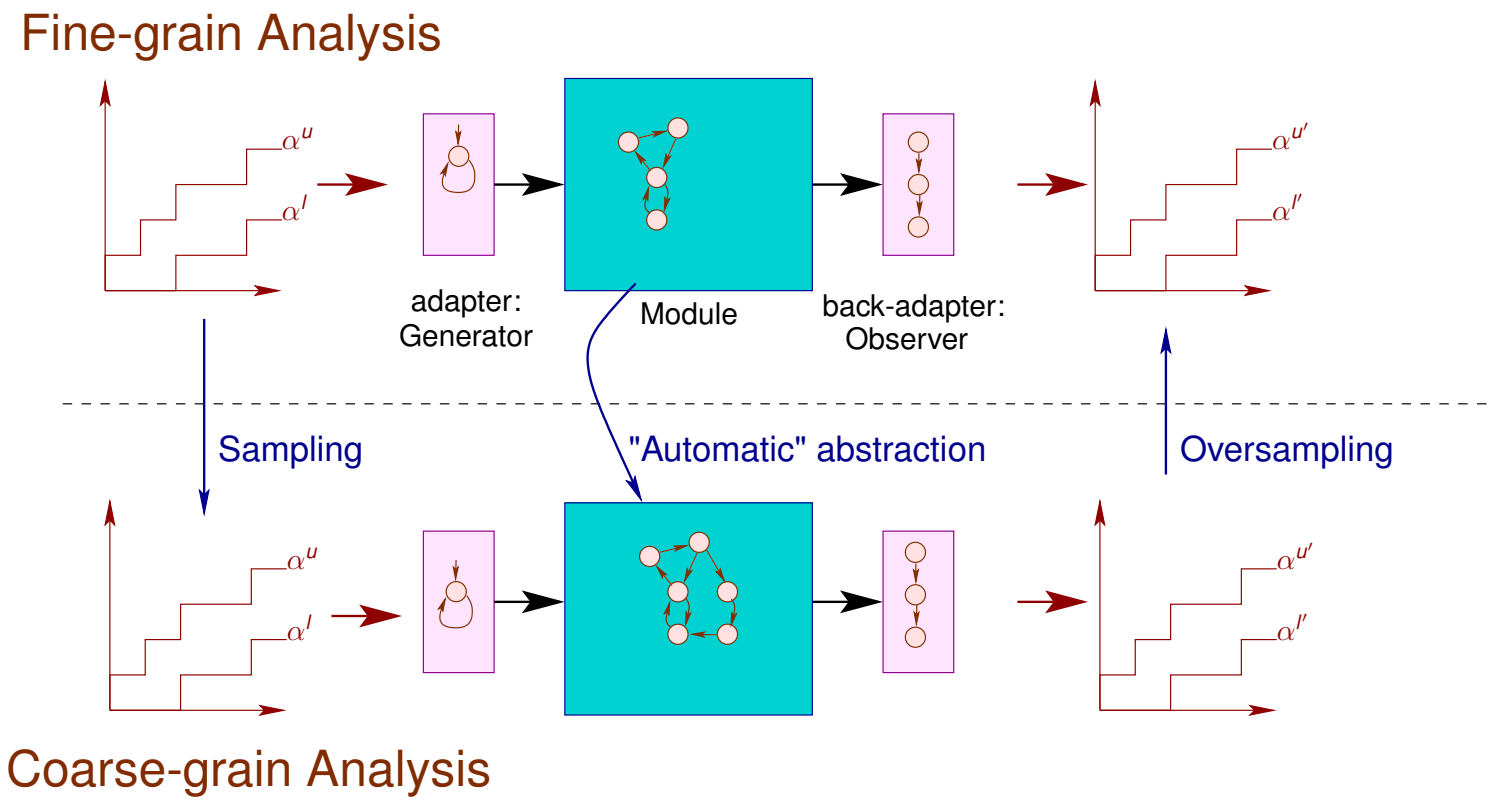
Granularity: The framework



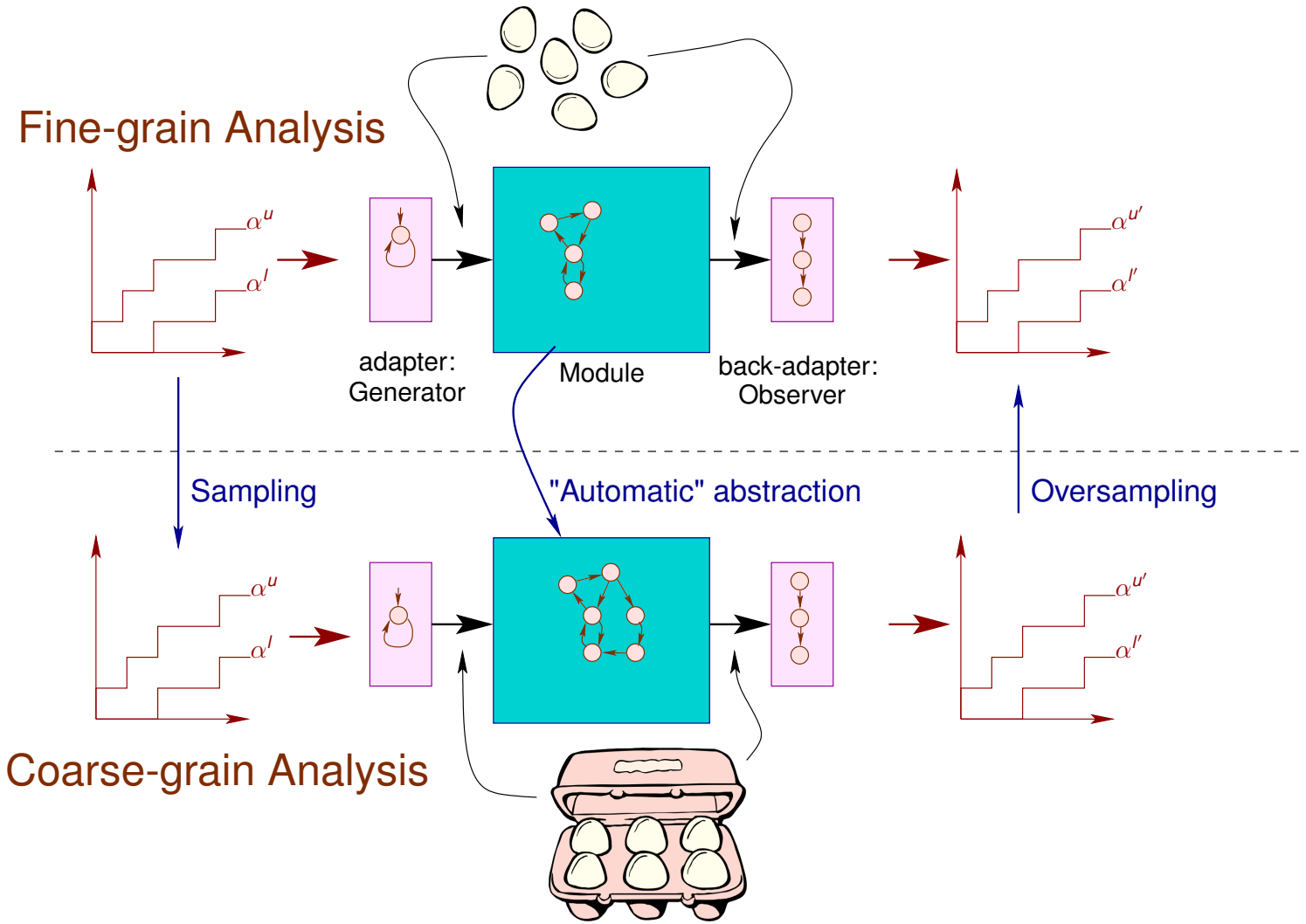
Granularity: The framework



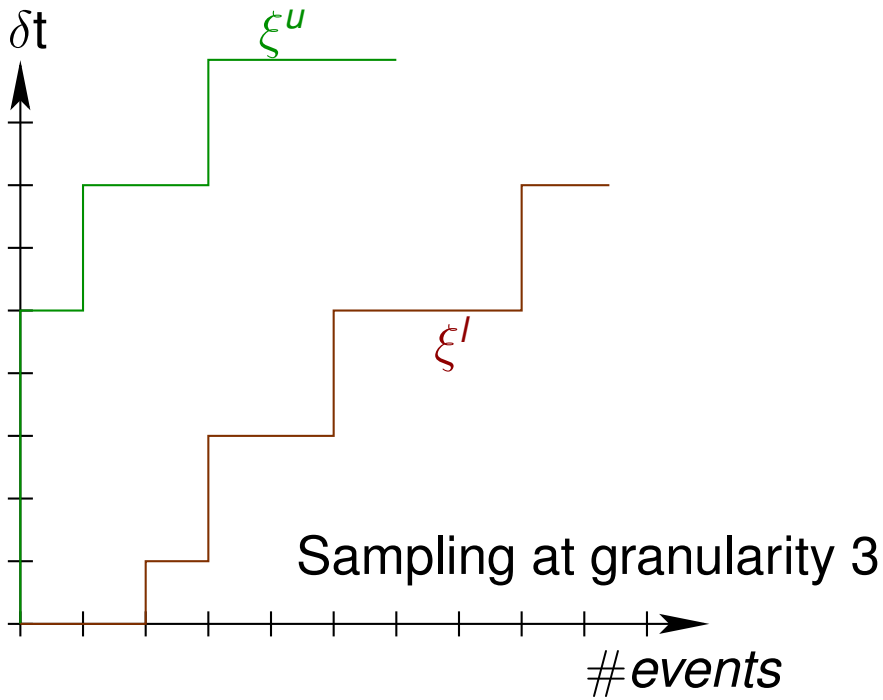
Granularity: The framework



Granularity: The framework

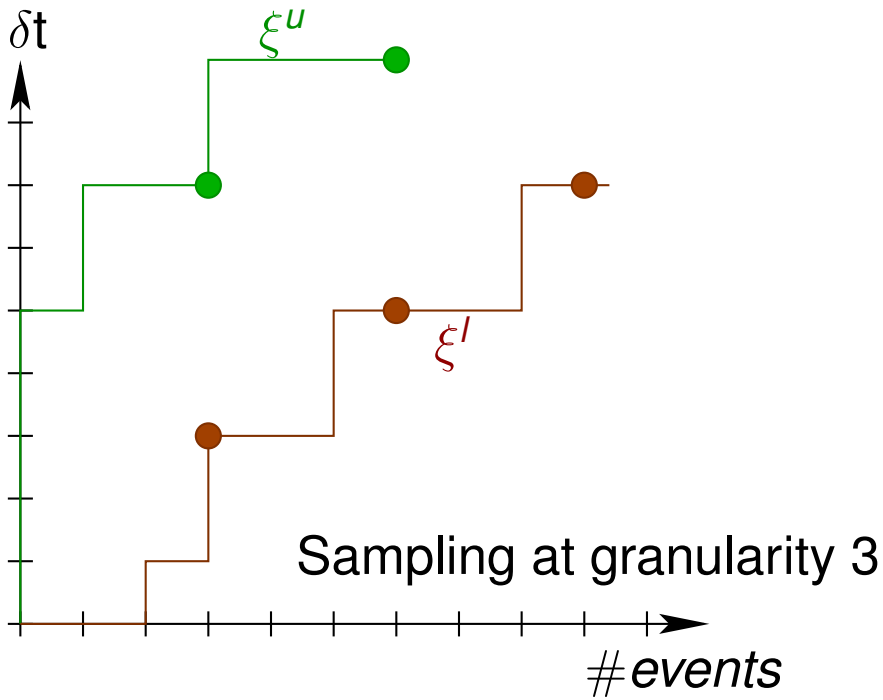


Grouping Events: Abstracting Arrival Curves



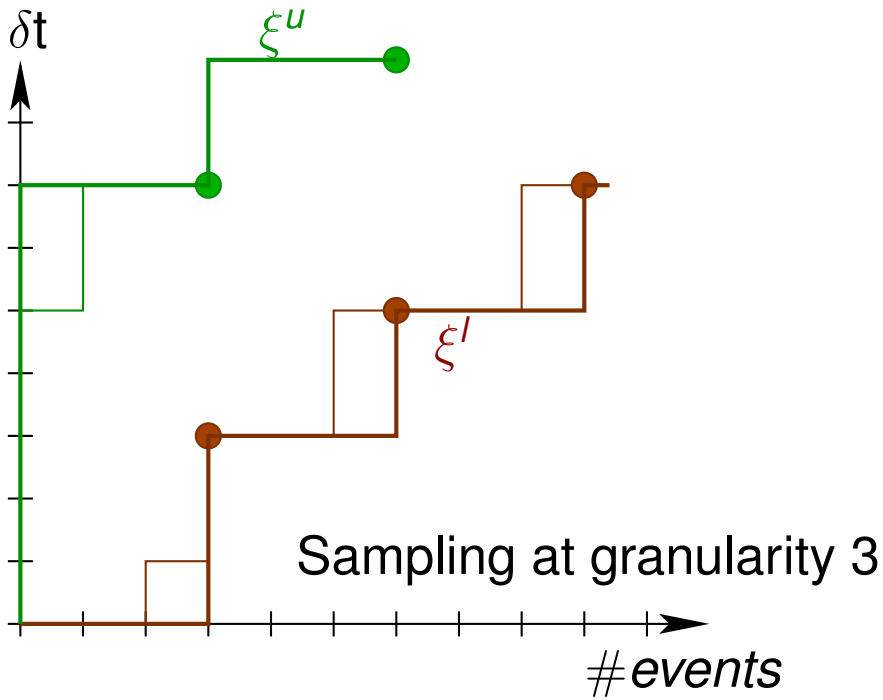
- Abstraction done by mere sampling
- Straightforward refinement (oversampling)

Grouping Events: Abstracting Arrival Curves



- Abstraction done by mere sampling
- Straightforward refinement (oversampling)

Grouping Events: Abstracting Arrival Curves



- Abstraction done by mere sampling
- Straightforward refinement (oversampling)

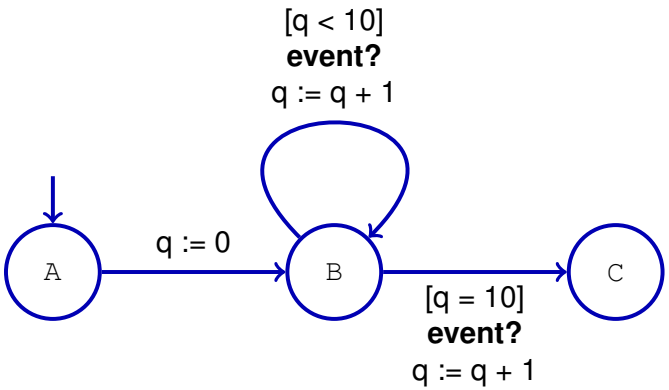
This section

3 Granularity-based Abstraction
- Idea: Group Events into “Coarse Events”
- Automatic Abstraction of Modules

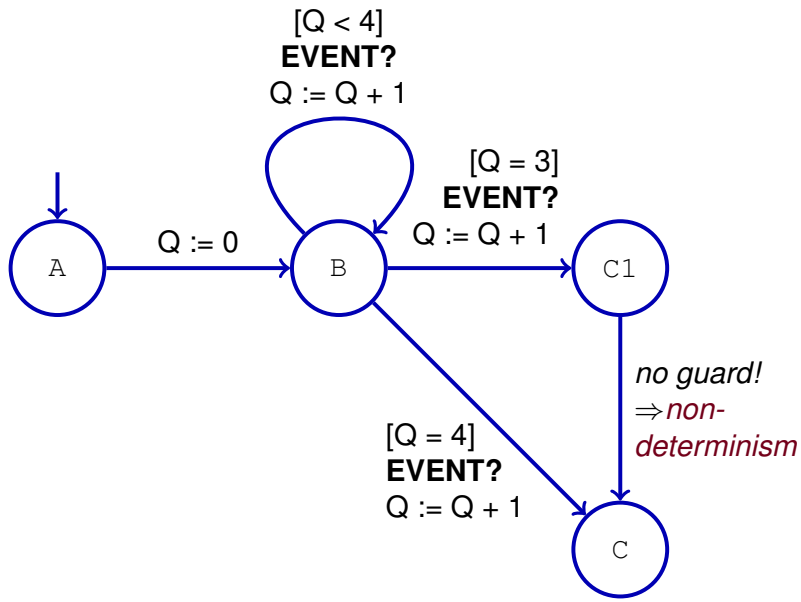
Abstracting a module: A simple example

Abstraction at granularity 3

Fine grain automaton:

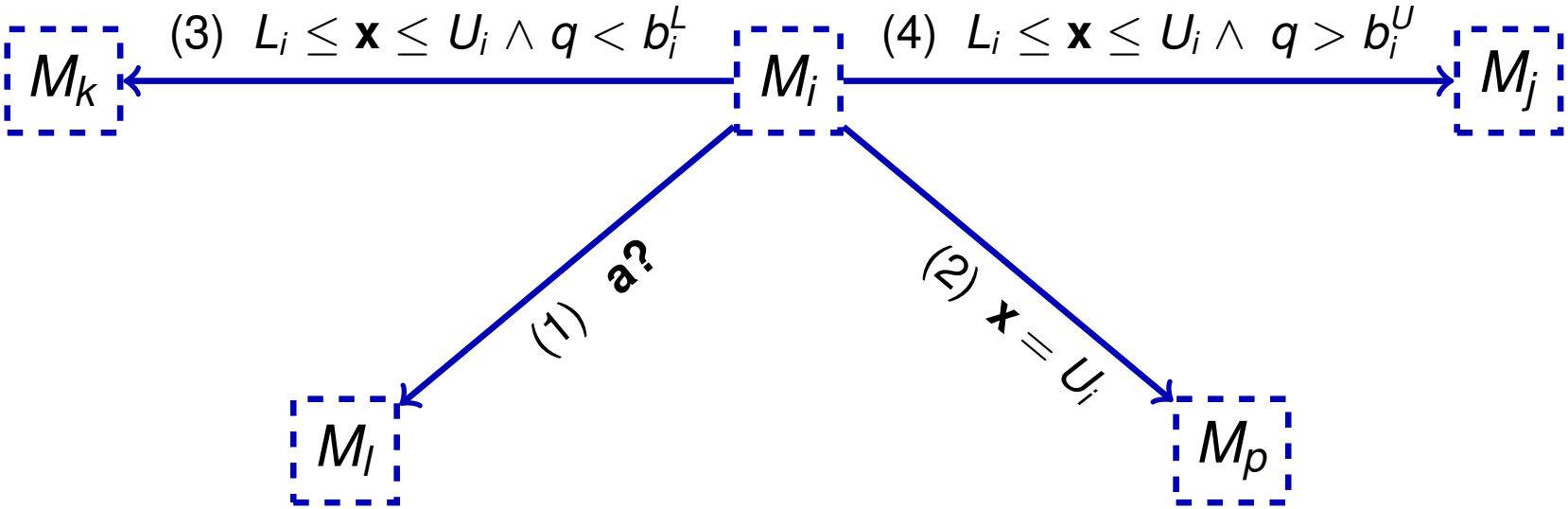


Coarse abstraction:



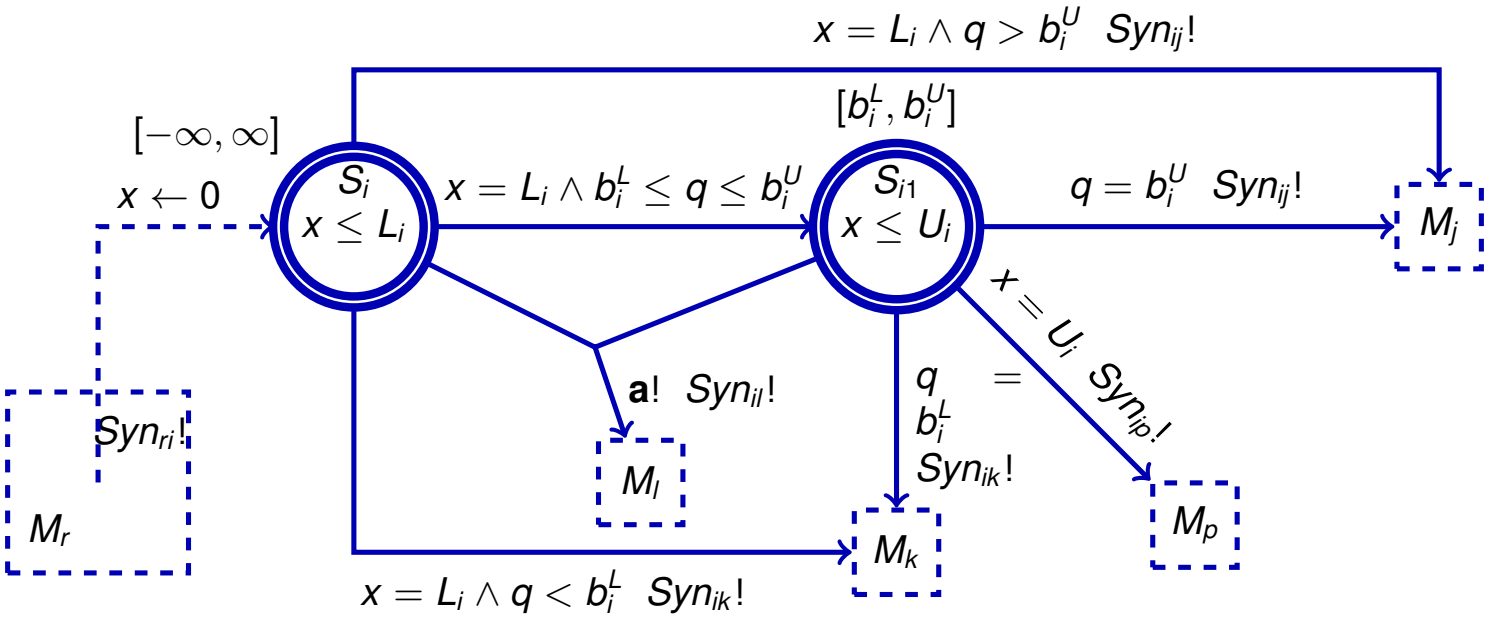
Abstracting a module: An Algorithm

- Abstraction hard to automate on general timed automata
- $\Rightarrow$ we define the translation on a subset
- Kind of transitions managed:

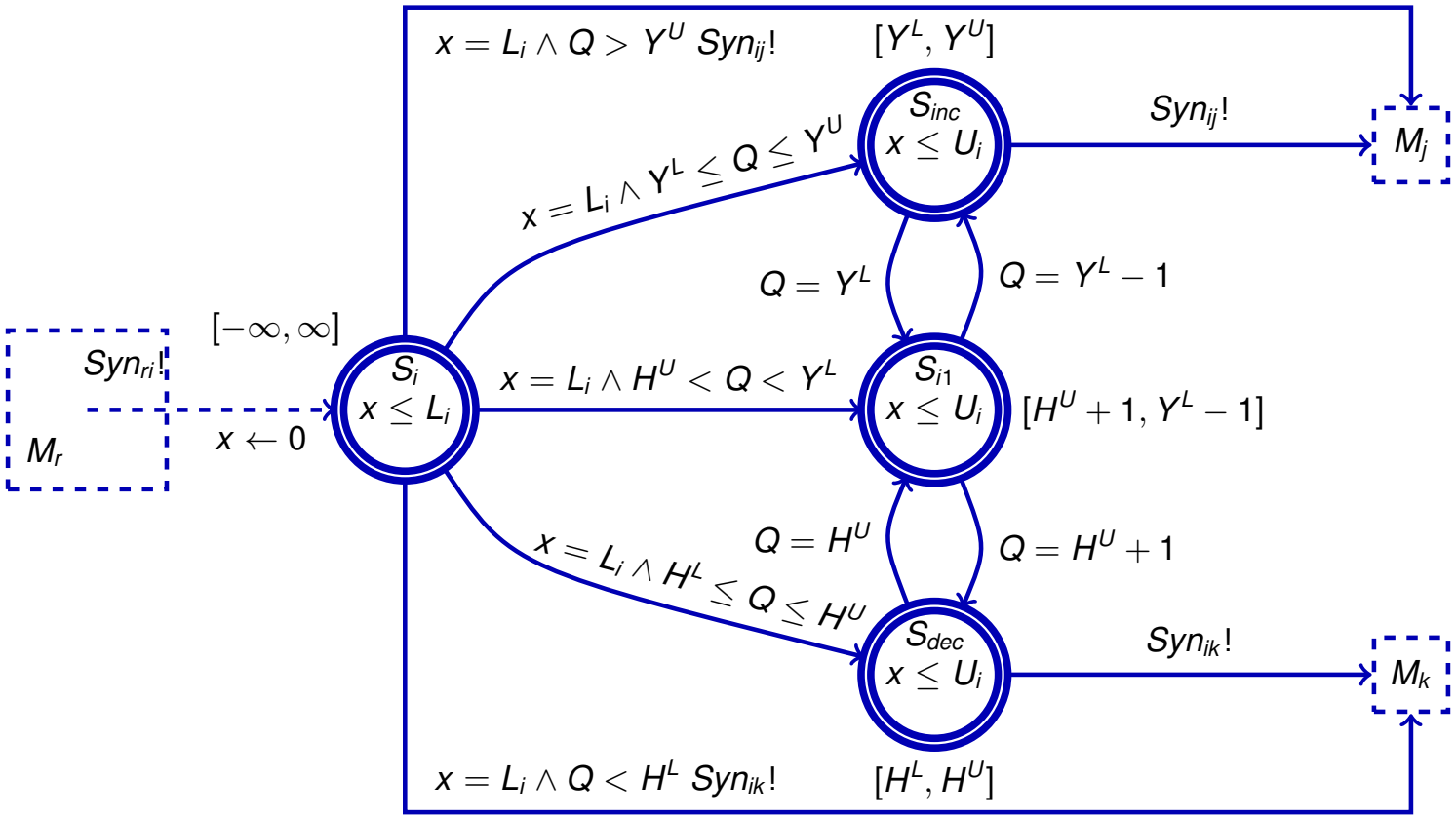


- M_i = 1 “Mode” = 1 group of control points in the time automaton.

Semantics in terms of Timed Automata



Translation to Coarse Automata

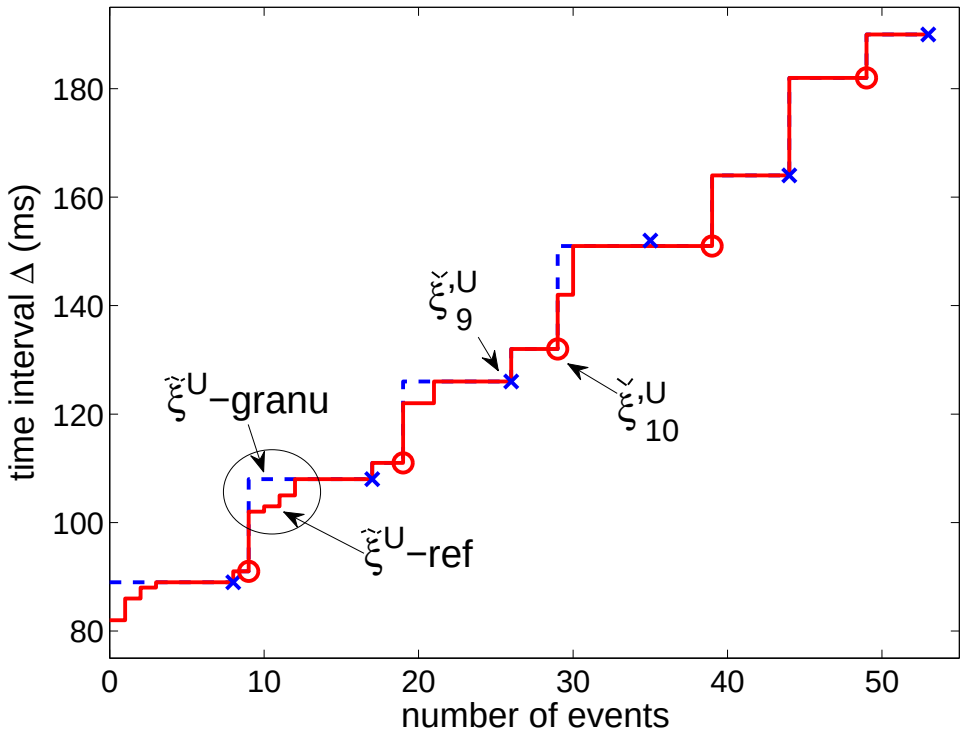


Put it all Together

1. Model the module in terms of modes and transitions,
2. Chose a granularity g,
3. Sample the input curve at granularity g,
4. Translate the model into the coarse timed automaton,
5. Compute the output curve (for granularity g),
6. Over-sample to get an approximation of the fine-grain output curve.

Combine multiple analysis

- One can run the analysis at multiple granularities (g_1, g_2, ...)
- We get multiple approximations of the same curve ...
- ... and combine them together (pointwise min/max, and a refinement algorithm based on causality closure, cf. my TACAS talk)



Summary

Summary

- Variant of existing RTC/TA interfacing, with granularity-based abstraction,
- Experimented on small example: up to 99% speedup
- Limitations:
  - Automatic algorithm specified, but not implemented as of now
  - Manual optimizations in addition to automatic algorithm needed to avoid loss of precision

Questions?