Timed Actors and their Formal Verification

Marjan Sirjani
Professor in Software Engineering
Mälardalen University, IDT
Cyber-Physical Systems Analysis group

Sept. 18, 2023
Antwerp, Belgium

Combined 30th International Workshop on Expressiveness in Concurrency
and 20th Workshop on Structural Operational Semantics
AFFILIATED WITH CONCUR 2023
(AS PART OF CONFEST 2023)

Acknowledgement:
All the Rebeca Team,
specially
Ehsan Khamespanah

Timed Actors for Modeling and Analysis

I will talk about

Modeling

Analysis and Verification

Applications

- Actors and Timed Rebeca
- Model Checking of Timed Rebeca and Reduction Techniques, different semantics for Timed Rebeca
- Different Projects

Main messages of the talk

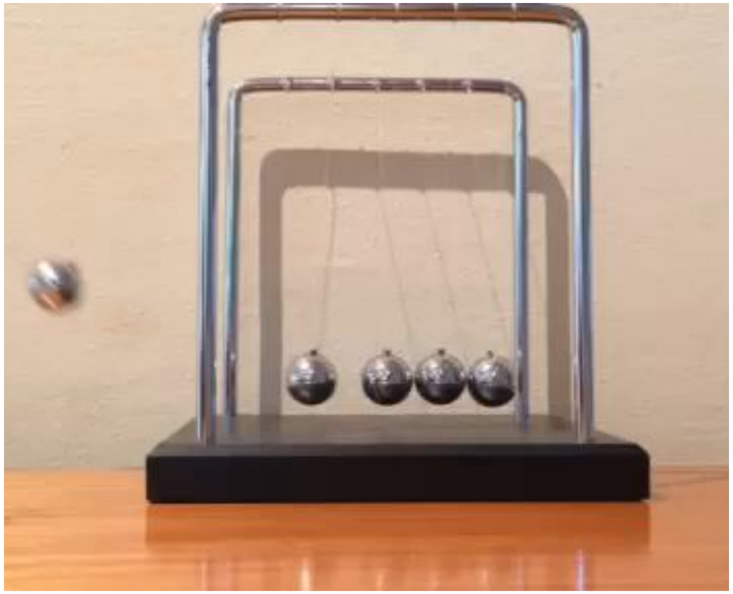
- The actor-based language, Rebeca, provides a friendly and analyzable model for distributed, concurrent, event-driven software systems and cyber-physical systems.

- Floating Time Transition System is a natural event-based semantics for timed actors, giving us a significant amount of reduction in the state space, using a non-trivial idea.

Yet another model?
Models vs. Reality

A model is any description of a system that is not the thing-in-itself.

The target: the thing being modeled



The model

$$x(t) = x(0) + \int_0^t v(\tau)d\tau$$
$$v(t) = v(0) + \frac{1}{m}\int_0^t F(\tau)d\tau,$$

In this example, the *modeling universe* is calculus and Newton's laws.

Another Model

Faithfulness is how well the model and its target match

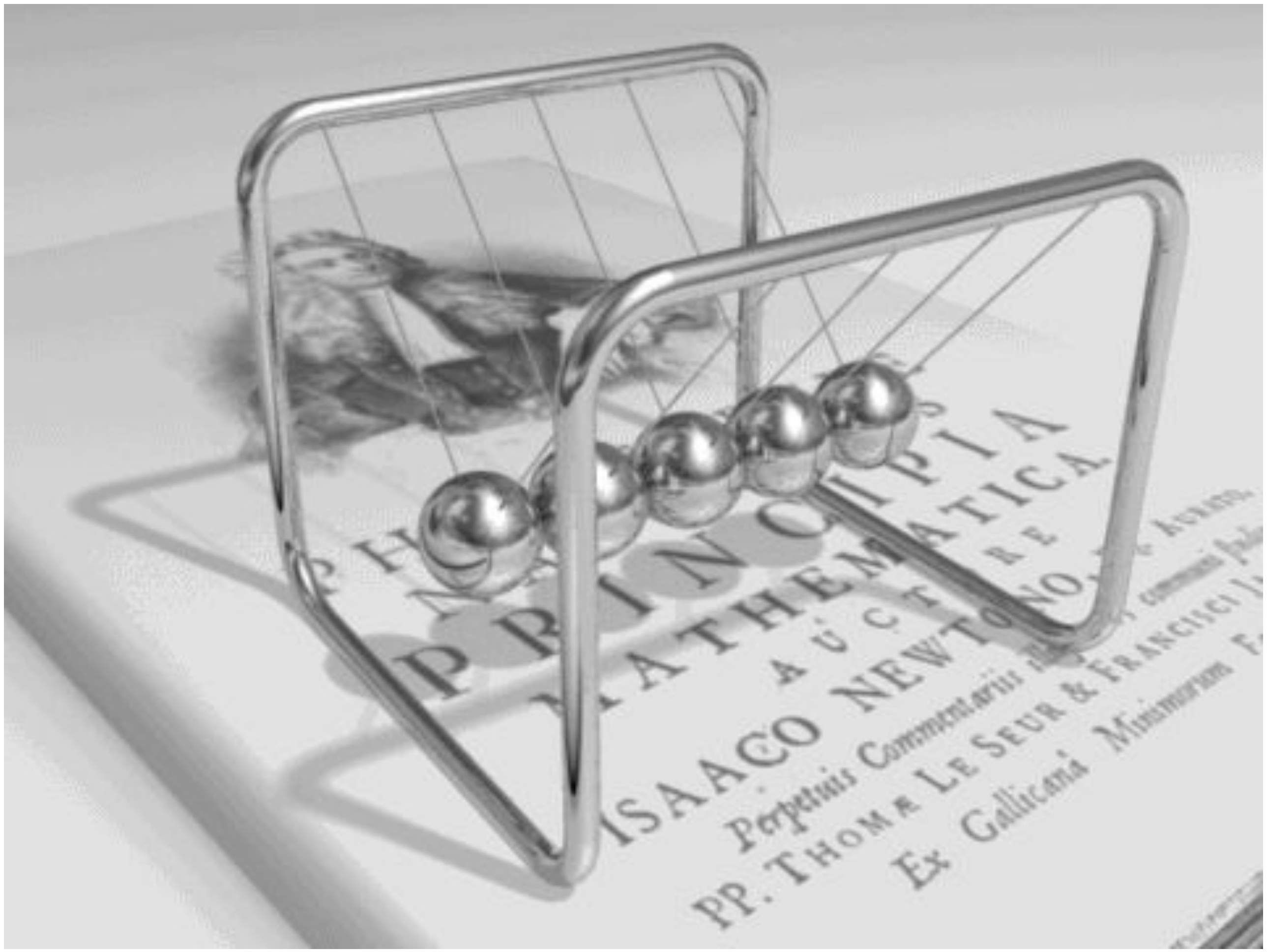
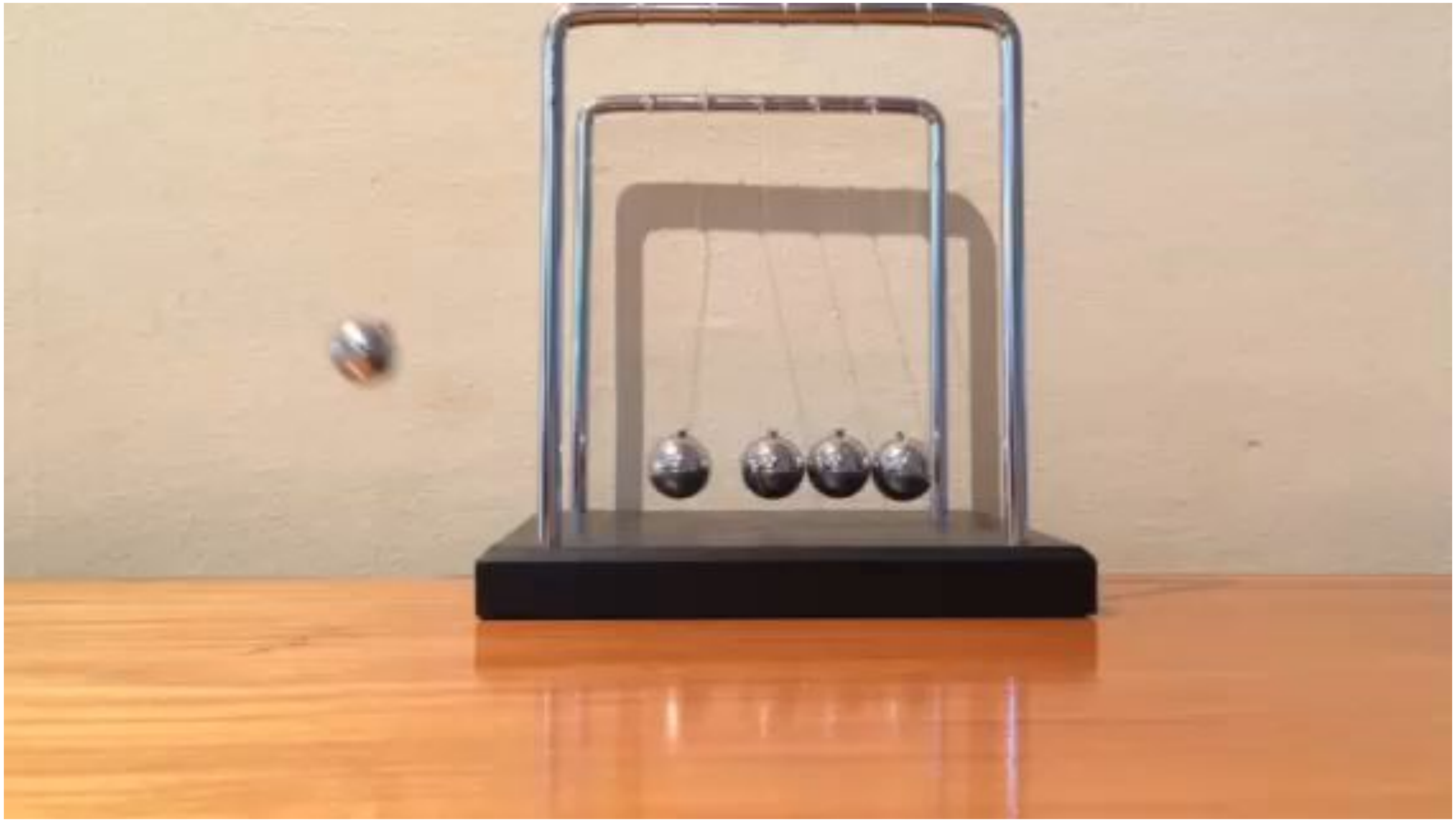


Image by Dominique Toussaint, GNU Free Documentation License, Version 1.2 or later.

A Physical Realization



The Value of Models

- In *science*, the value of a *model* lies in how well its behavior matches that of the physical system.

- In *engineering*, the value of the *physical system* lies in how well its behavior matches that of the model.

A scientist asks, “Can I make a model for this thing?”
An engineer asks, “Can I make a thing for this model?”

Useful Models and Useful Things

To a *scientist*, the model is flawed.
To an *engineer*, the realization is flawed.

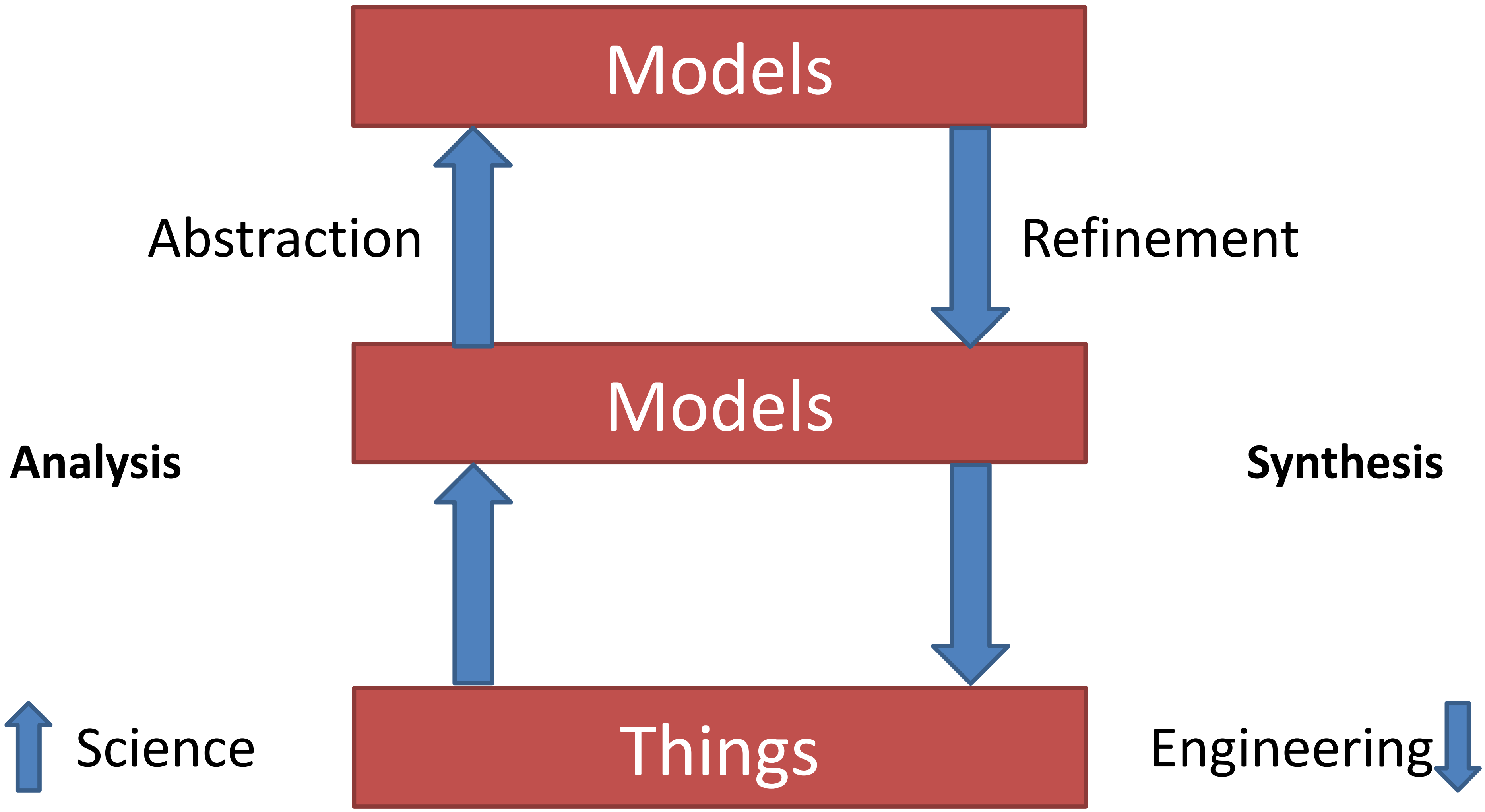
"Essentially, all models are wrong,
but some are useful."

Box, G. E. P. and N. R. Draper, 1987: *Empirical Model-Building and Response Surfaces*. Wiley Series in Probability and Statistics, Wiley.

"Essentially, all system implementations
are wrong, but some are useful."

Lee and Sirjani, "What good are models," FACS 2018.

Models and Models and Things



Faithfulness

- Faithfulness of the *modeling language* is important

- Properties of the modeling language should reflect properties of the problem domain
  - A modeling language with encapsulation, discrete events, concurrency, and asynchronous interactions will make it easier to model distributed software systems.

Power is Overrated, Go for Friendliness!

- Expressiveness versus Faithfulness and Usability in Modeling
  - Based on my experience with actors

- What is the Expressive Power of a language?
  - Generally defined as the breadth of ideas that can be represented and communicated in a language
  - Usually checked by mutually encoding the formalisms into each other

Power is Overrated, Go for Friendliness! Expressiveness, Faithfulness and Usability in Modeling - The Actor Experience, Edward Lee Festschrift, 2017

Modeling

with my engineering hat on

The Language, the Thing, the Modeler

- Expressiveness of the modeling *language*
- Faithfulness of the "modeling language" or "the model" to the *thing*
- Usability of the modeling language for the *modeler*

Power is overrated, Go for Friendliness

Friendly Models: Faithful and Usable

- Friendly to the system we want to build: Faithfulness
- Friendly to the user who builds the system: Usability

- The Map you use has to show the roads correctly, and also be easily readable.

Compare Google map and Apple map

Faithfulness

- Less semantic gap between the real world and the model
- The structures and features supported by the modeling language match the constructs of interest in the system being modeled

- Faithfulness: Leads to Domain-specific Modeling Languages

- Faithfulness is also defined as: The degree of detail incorporated in the model (but this is not my definition)

Model of Computation and Faithfulness

- MoC: a collection of rules
  - govern the execution of the [concurrent] components and
  - the communication between components

- We say a modeling language is faithful to a system if the model of computation supported by the language matches the model of computation of [the features of interest of] the system.

Different approaches for Modeling and Verification

Abstract

Mathematical

Modeling languages

CCS CSP

Petri net

RML

Timed Automata

FDR

UPPAAL

NuSMV

Spin

SMV

Promela

Verification Techniques:

- **Deduction**

  needs high expertise

- **Model checking**

  causes state explosion

Java PathFinder

Programming languages

Bandera

SLAM

Java

C

Too heavy

Not always

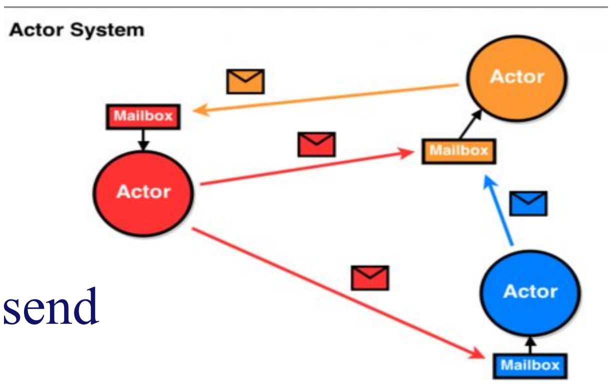
Our choice for modeling: Actors

- A reference model for concurrent computation
- Consisting of concurrent, distributed active objects

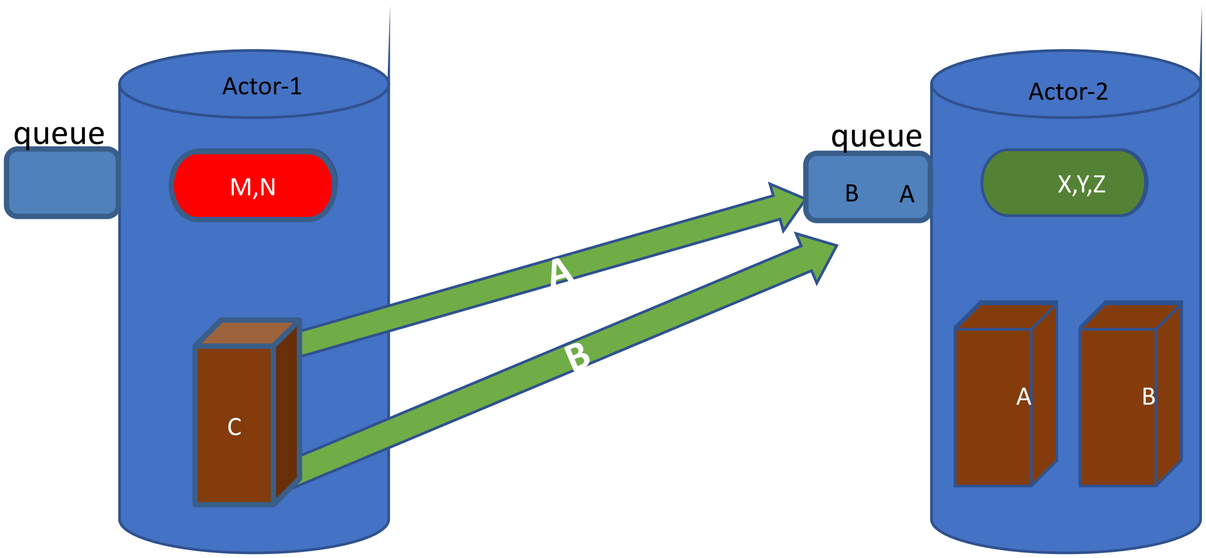
- Proposed by Hewitt as an agent-based language (MIT, 1971)
- Developed by Agha as a concurrent object-based language (Illinois, since 1984)
- Formalized by Talcott (with Agha, Mason and Smith): Towards a Theory of Actor Computation (CONCUR 1992)

Rebeca: The Modeling Language Asynchronous and Event-driven

- **Rebeca: Reactive object language** (Sirjani, Movaghar, Peresented at AVoCS 2001)
  - Based on Hewitt actors
  - Concurrent reactive objects (OO)
  - Java like syntax
- Communication:
  - Asynchronous message passing: non-blocking send
  - Unbounded message queue for each rebec
  - No explicit receive
- Computation:
  - Take a message from top of the queue and execute it
  - Event-driven



Rebeca - Behavior



An actor:

- A message queue
- Message servers
- State Variable

Rebeca - Structure

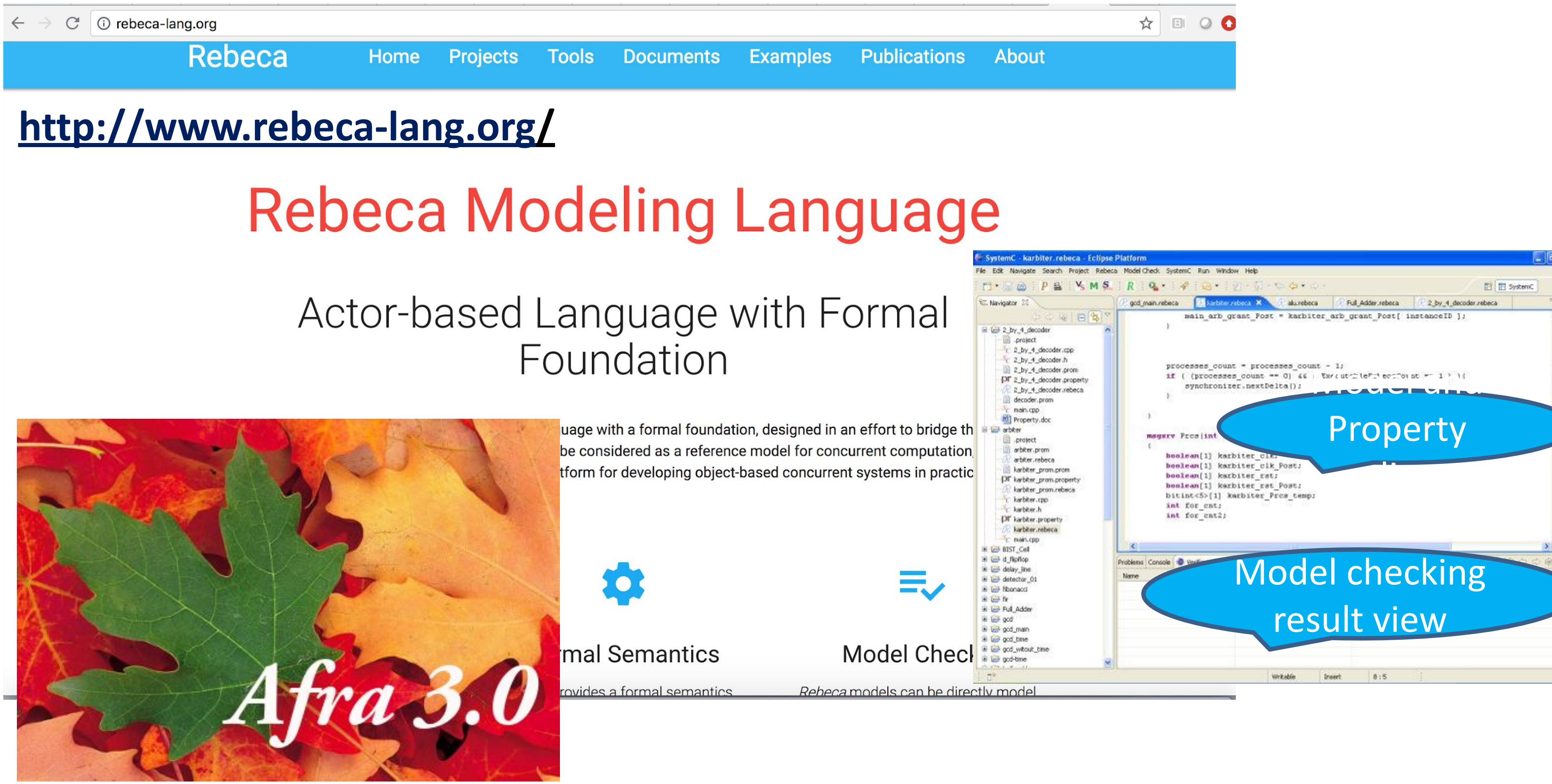
A Rebeca model consists of:

-reactive classes and their behavior definition

-instantiations of rebecs (reactive objects) to run in parallel

A **reactive class** is made of three parts:

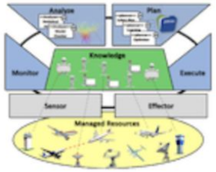
1. **known rebecs** (other rebecs to whom messages can be sent),
2. **state variables** (like attributes in object-oriented languages),
3. **message server** (defining the behavior of the actor like methods).



Festschrift Papers:

- **Ten years of Analyzing Actors: Rebeca Experience** (Sirjani, Jaghouri), Carolyn Talcott Festschrift, 70th birthday, LNCS 7000, 2011
- **On Time Actors** (Sirjani, Khamespanah), Theory and Practice of Formal Methods, Frank de Boer Festschrift, 2016
- **Power is Overrated, Go for Friendliness! Expressiveness, Faithfulness and Usability in Modeling - The Actor Experience**, Edward Lee Festschrift, 2017

Projects



SEADA

In SEADA (Self-Adaptive Actors) we will use Ptolemy to represent the architecture, and extensions of Rebeca for modeling and verification. Our models@runtime will be coded in an extension of Probabilistic Timed Rebeca, and supporting tools for customized run-time formal verification

RoboRebeca

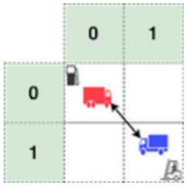
RoboRebeca is a framework which provides facilities for developing safe/correct source codes for robotic applications. In RoboRebeca, models are developed using Rebeca family language and automatically transformed into ROS compatible source codes. This framework is

HybridRebeca

Hybrid Rebeca, is an extension of actor-based language Rebeca, to support modeling of cyber-physical systems. In this extension, physical actors are introduced as new computational entities to encapsulate the physical behaviors. Learn more

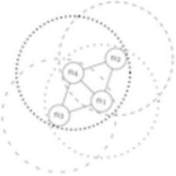
Tangramob

Tangramob offers an Agent-Based



AdaptiveFlow

AdaptiveFlow is an actor-based eulerian



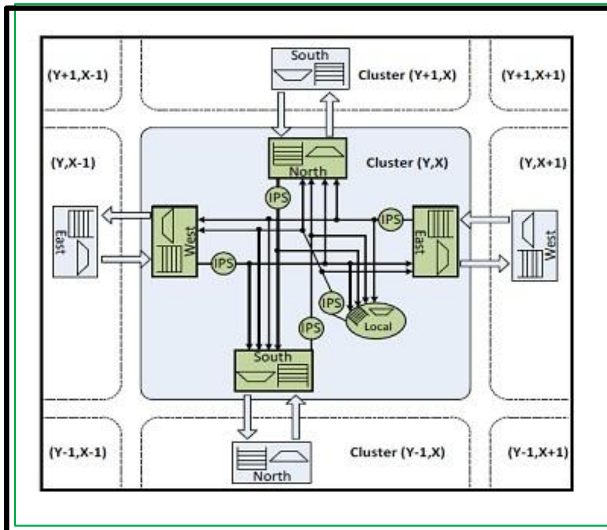
wRebeca

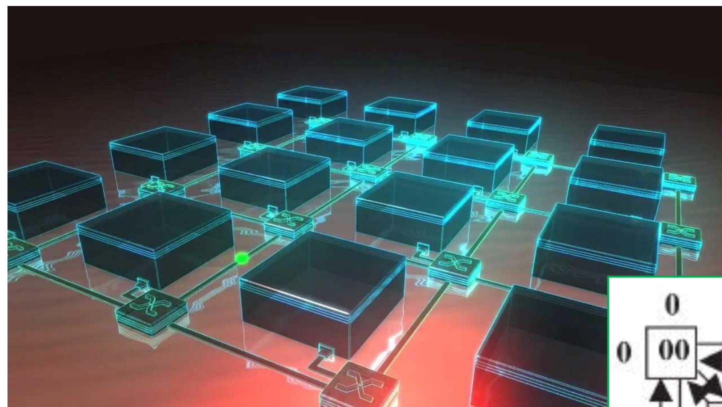
wRebeca is an actor-based modeling

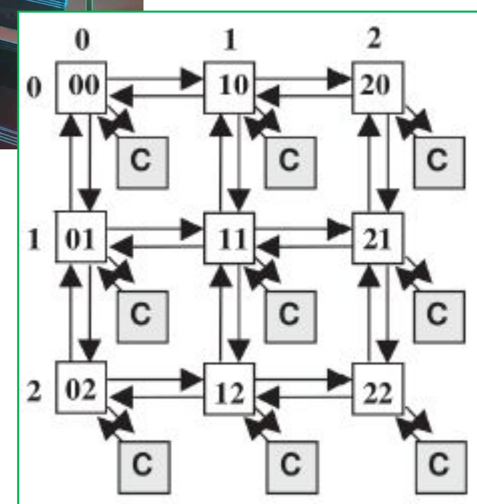
Timed Rebeca

- An extension of Rebeca for real time systems modeling
  - Computation time (delay)
  - Message delivery time (after)
  - Periods of occurence of events (after)
  - Message expiration (deadline)

Timed Rebeca with an example: Network on Chip





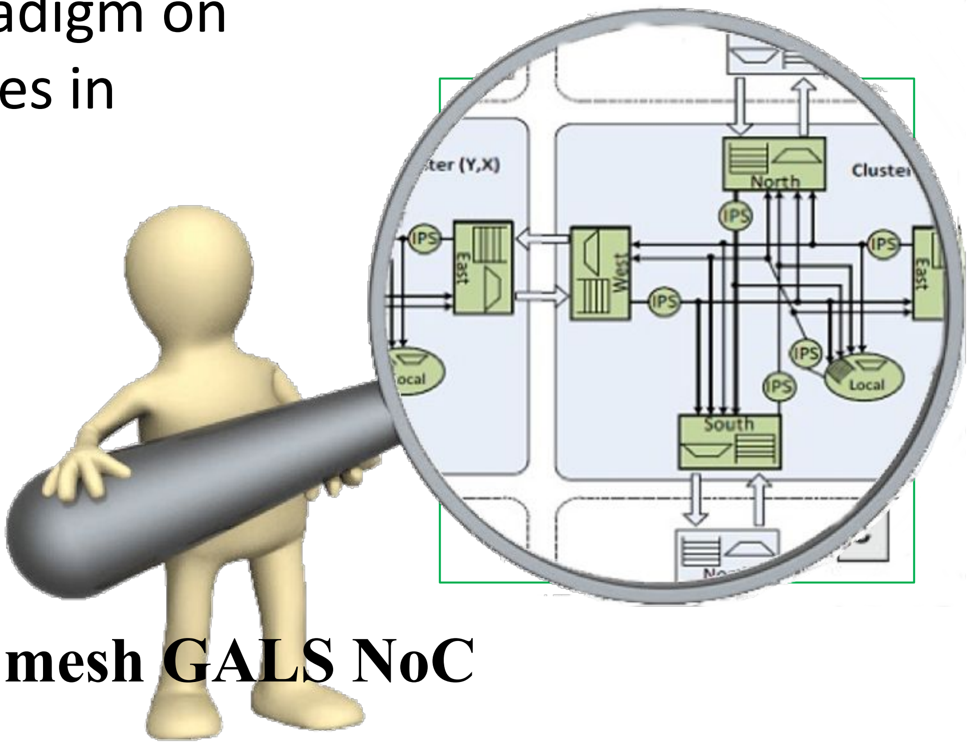


Exploring Design Decisions:

- Evaluating routing algorithms
- Buffer length
- Choose the best place for the memory

Globally Asynchronous-Locally Synchronous NoC

NoC is a communication paradigm on a chip, typically between cores in a system on a chip (SoC).

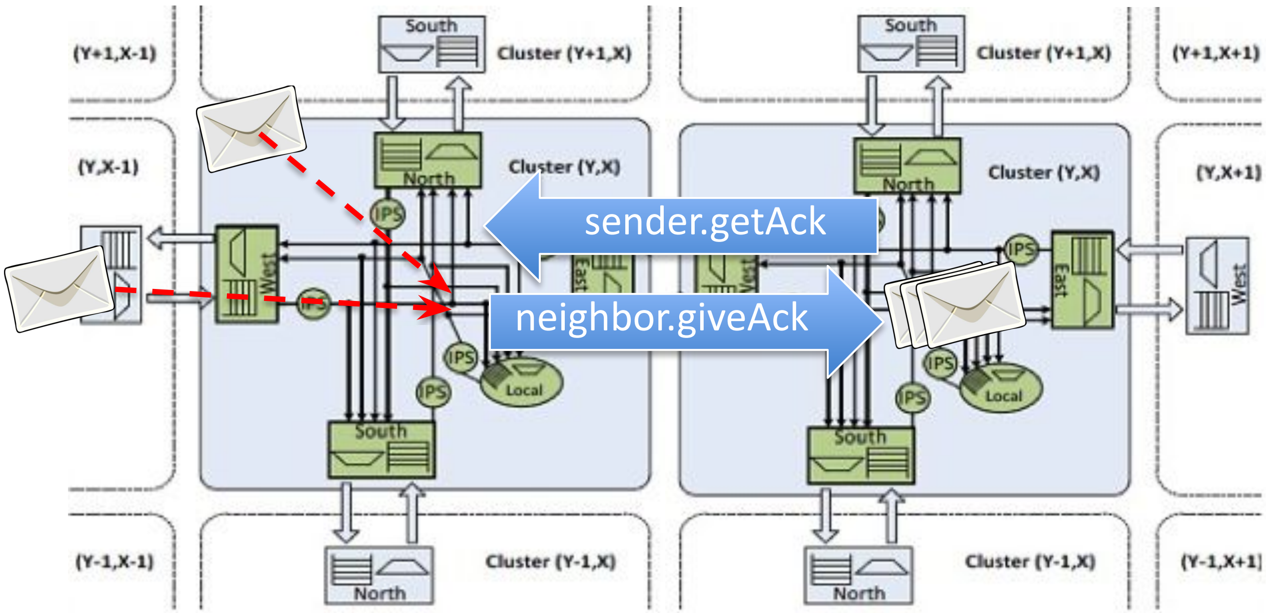


- GALS NoC

ASPIN: Two-dimensional mesh GALS NoC

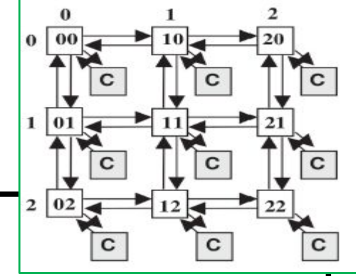
XY routing algorithms

Communication Protocol



- **Four phase handshake communication protocol**: the channel is blocked until the packet arrives to the other router.
- The sender put the packet in the output buffer along with the request signal to the receiver and doesn’t send the next packet before receiving the Ack.

ASPIN: Rebeca abstract model




```
reactiveclass Router{
    knownrebecs
        {Router[4] neighbor, Core myCore}
    statevars{int[4] buffer;}
    Router (myId-row, myId-col) { ...
    }
    msgsrv reqSend()  {
        neighbor[x]. giveAck() after(3);   . . .
    }
    msgsrv getAck()  {
        // receive ack from the receiver
        // get ready for receiving the next
           packet …
    }
    msgsrv giveAck (…) {
        //if the message is for my core use it
        myCore.forMyCore()
        //send ack to the sender
        sender.getAck() after(3);
        // if not route it to the receiver …
} }
```

```
reactiveclass Core{
    knownrebecs {Router myRouter}
    statevars{ …}
    Core ( … ) {
        ....
    }
    msgsrv forMyCore()  {
        // get the Packet and use it
        . . .
    }
main(){
    Router r00(r02,r10,r01,r20)(0,0);
    Router r01(r00,r11,r02,r21)(0,1);
    …
    Core c00(r00)
    Core c01(r01)
    …
}
```

Actor type and its message servers

Constructor

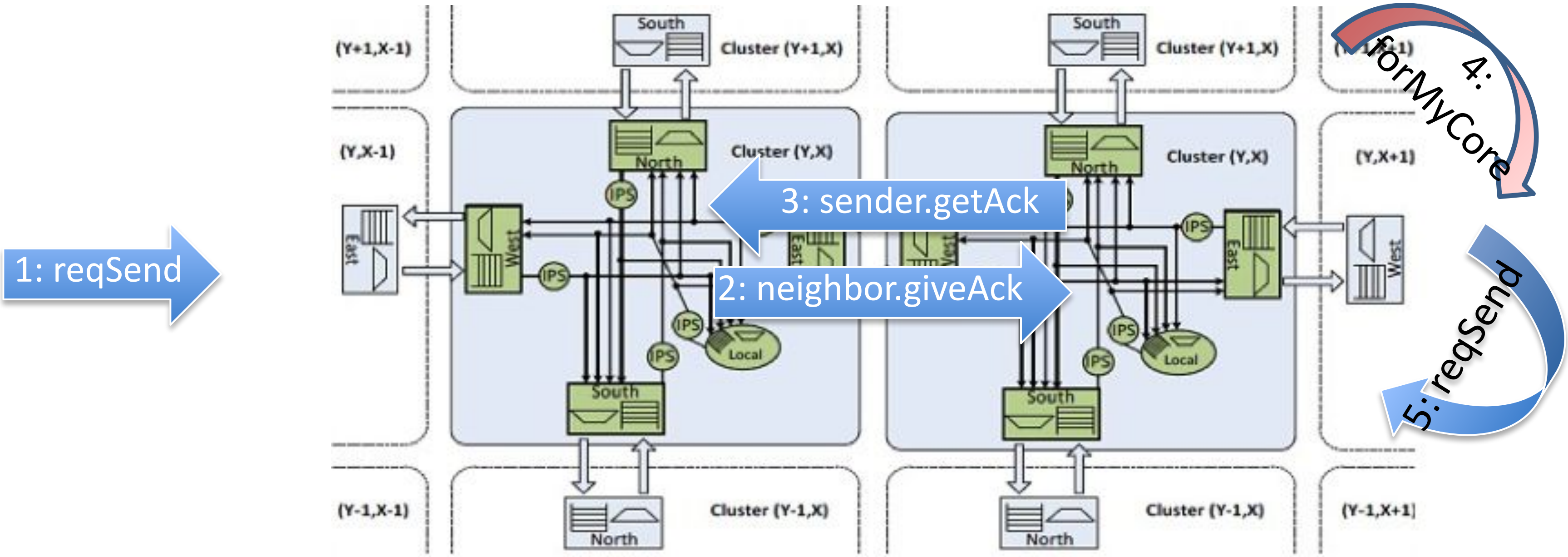
A message server

Asynchronous message sending

Instances of different actors

Parameters

Known rebecs



reqSend:
//Route the Packet
neighbor.giveAck;

getAck:
//send the Packet
//set the flag of your port to free

giveAck:
//if I am the final Receiver
//then Consume the Packet
sender.getAck;
myCore.forMyCore;

//else if my buffer is not full
//get the Packet
sender.getAck
//and route it ahead
self.reqSend;

ASPIN: Rebeca abstract model

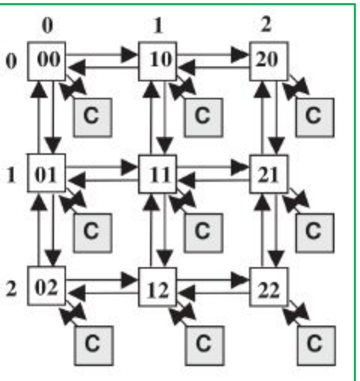
```
reactiveclass Router{
    knownrebecs {Router[4] neighbor}
    statevars{int[4] buffer;}
       Router ( … ) {
          ….
       }
       msgsrv reqSend()  {
          delay(2);
          neighbor[x]. giveAck() after(3) deadline(6);
          . . .
    }

msgsrv giveAck (…) {
     //if the message is for my core use it
     myCore.forMyCore()
     //send ack to the sender
     sender.getAck() after(3);
    // if not and buffer not full then route it to the receiver …
    // if buffer full then busy-wait until buffer empty
    else self.giveAck() after(10),
 }
      …
}
```

Deadline for the receiver

Time progress because of computation delay

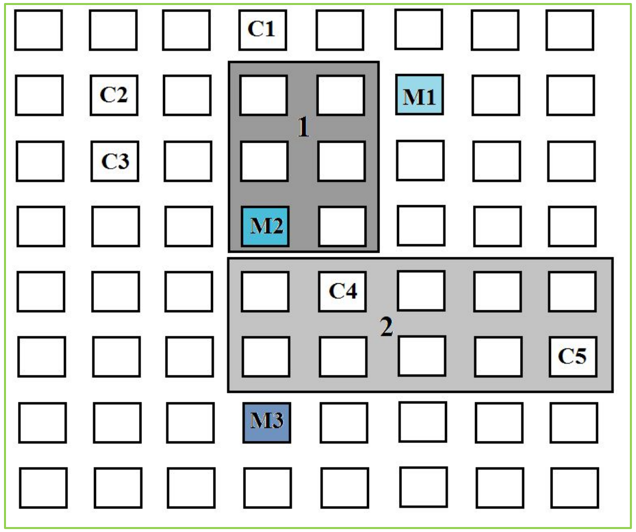
Communication delay

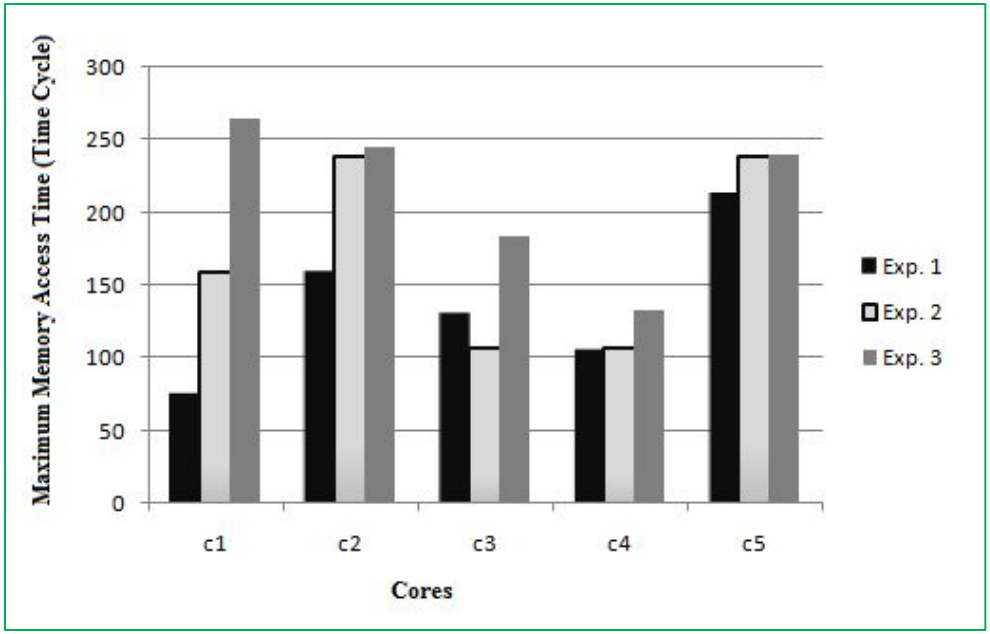
periodic tasks



Evaluation of different memory locations for ASPIN 8×8

- Consider 5 cores and their access time to the memory
- 3 choices for memory placement
- 40 packets are injected
- High congestion in area 1 and 2





- Unlike our expectation M1 is a better choice than M2
- The packet injection is based on an application (note that cores have different roles)

Modeling NoC in TRebeca

ASPIN Component	Model in Rebeca
Router + Core	Rebec
Buffer	Rebec queue (write/read delays by

Keep the constructs and features that affect the properties of interest and check the following:

1. Possible Deadlock
2. Successful sending and receiving of packets
3. Estimating the maximum end-to-end packet latency

Channel

Communication protocol

Model checking: 3 seconds

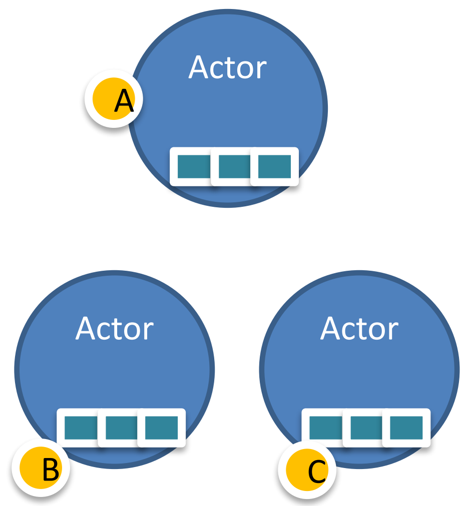
HSPICE: 24 hours

Much less details.

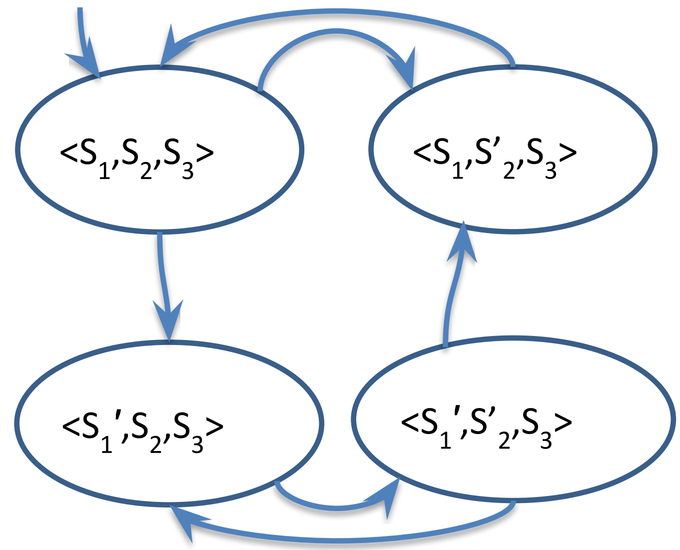
Showed the same trend.

Go Through Different models at Different Levels

Real World

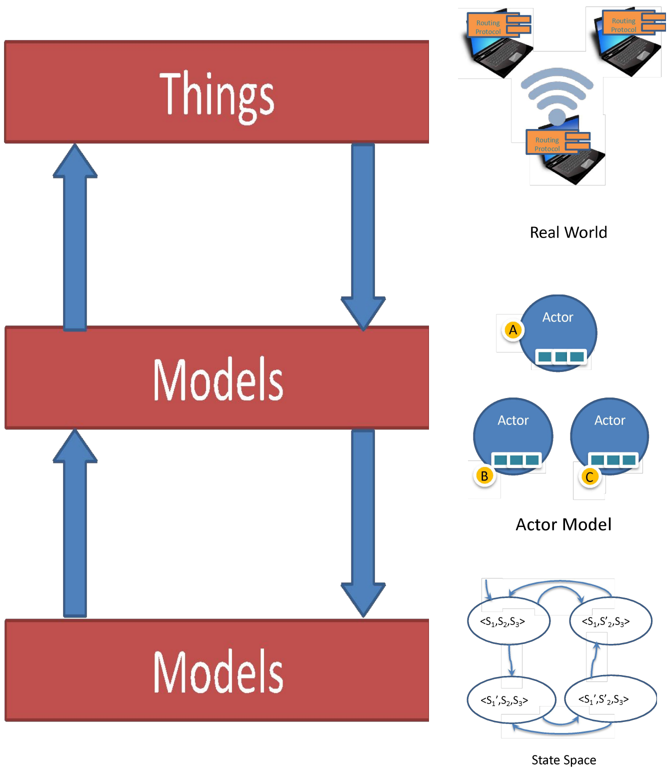


Actor Model



State Space

Efficient Model Checking of Timed Actors: Focus on Events



Model

To do Analysis

State Space

- Timed Automata
- Timed Transition System
- **Floating Time Transition System**

Standard Semantics: Timed Transition System

- In TTS transitions are of three types:
  - Passage of time
  - Taking a message from the queue to execute: event
  - Silent transition **τ**: internal actions in an actor

Semantics of a simple Timed-Rebeca Model:
Timed Transition System

```
reactiveclass RC1 (3) {
    knownrebecs {
     RC2 r2;
    }
    ;
    }
    msgsrv m1
     delay(2);
     r2.m2();
     delay(2);
     r2.m3();
     self.m1()
    }
}
```

```
reactiveclass RC2 (4) {
    knownrebecs {
     RC1 r1;
    }
    RC2() { }
    msgsrv m2() { }
    { }
```

Line number as program counter

```
  msgsrv m1() {
1     delay(2);
2     r2.m2();
3     delay(2);
4     r2.m3();
5     self.m1() after (10);
  }
```

the

```
msgsrv m1()
 1 delay(2);
 2 r2.m2();
 3 delay(2);
 4 r2.m3();
 5 self.m1() aft
}
```

time = 0

time = 2

time = 2

time = 4

time = 4

$time = time + 2$

S5		
r1	queue	-
	pc	$m1: 4$
r2	queue	-
	pc	-

$\tau(r1)$

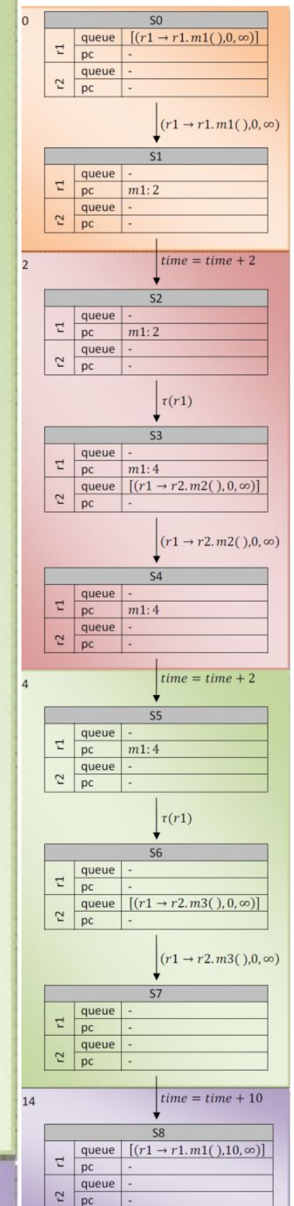
S6		
r1	queue	-
	pc	-
r2	queue	$[(r1 \rightarrow r2.m3(\,), 0, \infty)]$
	pc	-

$(r1 \rightarrow r2.m3(\,), 0, \infty)$

S7		
r1	queue	-
	pc	-
r2	queue	-
	pc	-

time = 14

$time = time + 10$



Properties in an event-based system

- Properties that we care about the most:
  - Distance of occurrence of two events
  - Event precedence

- Remember, in TTS the transitions are of three types:
  - Passage of time
  - Taking a message from the queue to execute: event
  - Silent transition **τ**: internal actions in an actor

Real-time Patterns

(Koymans, 1990), (Abid et al., 2011), (Bellini et al., 2009) and (Konrad et al., 2005), (Dwyer et al., 1999)

- Maximal distance
  - Every *e*1 is followed by an *e*2 within *x* time units
- Exact distance
  - Every *e*1 is followed by an *e*2 in exactly *x* time units
- Minimal distance
  - Two consecutive events of *e* are at least *x* time units apart

Properties that we care about the most:
- Distance of occurrence of two events
- Event precedence

maximum number of time units

- Precedence
  - Within the next *x* time units, the occurrence of *e*1 precedes the occurrence of *e*2

So, we proposed

- An event-based semantics for Timed Rebeca:
- Floating Time Transition System

Floating Time Transition System: Event-based Timed-Rebeca Semantics

- Formal semantics given as SOS rules
- The main rule is the schedular rule:

$$\frac{(\sigma_{r_i}(m), \sigma_{r_i}[rtime = max(TT, \sigma_{r_i}(now)), [\overline{arg} = \overline{v}], sender = r_j], Env, B) \xrightarrow{\tau} (\sigma'_{r_i}, Env', B')}{(\{\sigma_{r_i}\} \cup Env, \{(r_i, m(\overline{v}), r_j, TT, DL)\} \cup B) \rightarrow (\{\sigma'_{r_i}\} \cup Env', B')} C$$

The scheduler and progress of time

- The scheduler picks up messages from the bag based on their time tags and execute the corresponding methods.
- ***delay*** statements change the value of the current local time, ***now***, for the considered rebec.
- The time tag for the message is the current local time (***now***), plus value of the ***after***
- The scheduler picks the message with the smallest time tag of all the messages (for all the rebecs) in the message bag.
- The schedular checks if a ***deadline*** is missed.
- The variable ***now*** is set to the maximum between the current time of the rebec and the time tag of the selected message.

State space reduction: a simple Timed-Rebeca Model

```
reactiveclass RC1 (3) {
    knownrebecs {
      RC2 r2;
    }
    }
    msgsrv m1() {
      delay(2);
      r2.m2();
      delay(2);
       r2.m3();
      self.m1() after (1
    }
}
```

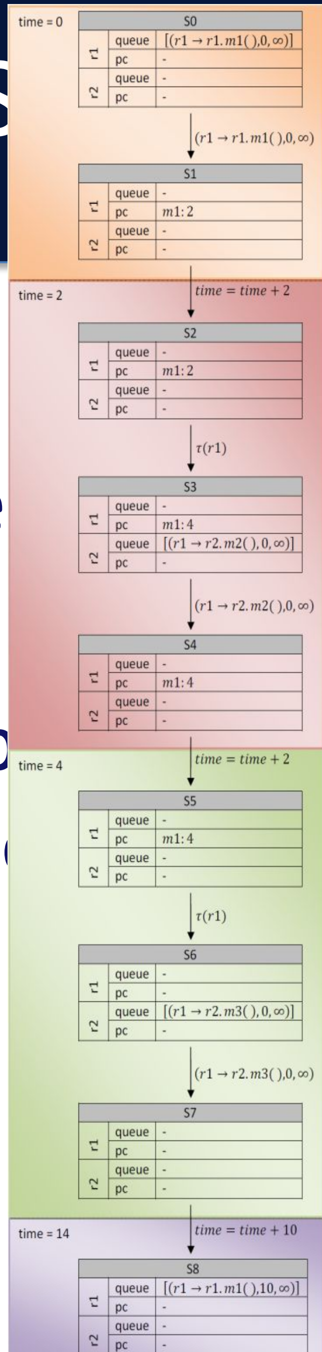
```
reactiveclass RC2 (4) {
    knownrebecs {
      RC1 r1;
    }
    RC2() { }
    msgsrv m2() { }
}
```

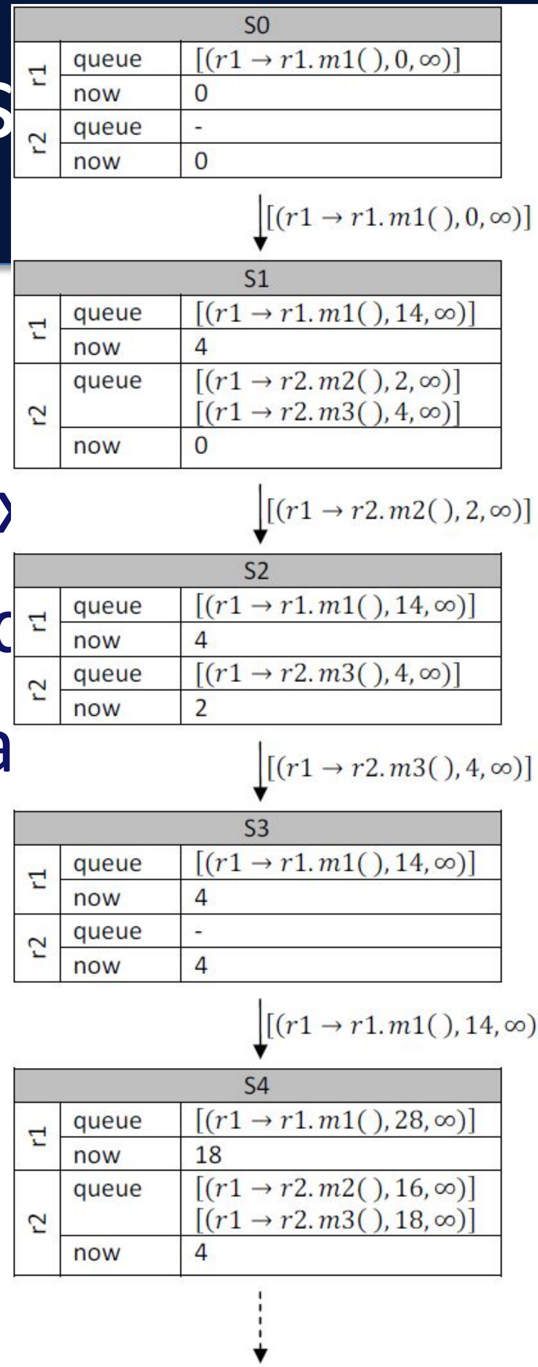
Line number as program counter

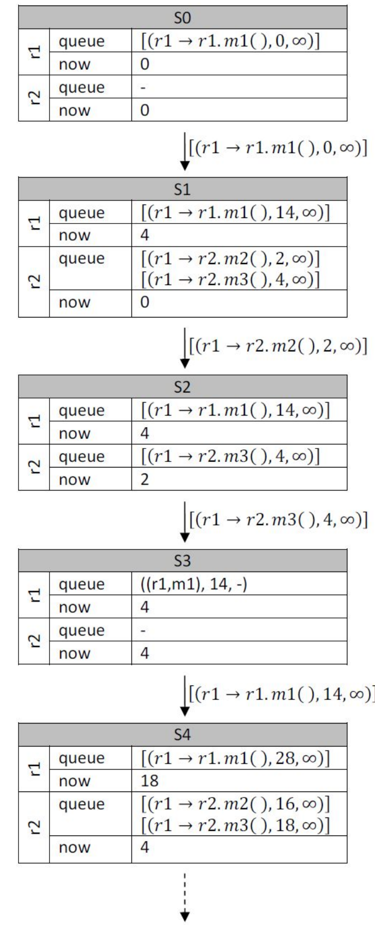
```
    msgsrv m1() {
1       delay(2);
2       r2.m2();
3       delay(2);
4        r2.m3();
5       self.m1() after (10);
    }
```

FTTS … s … el

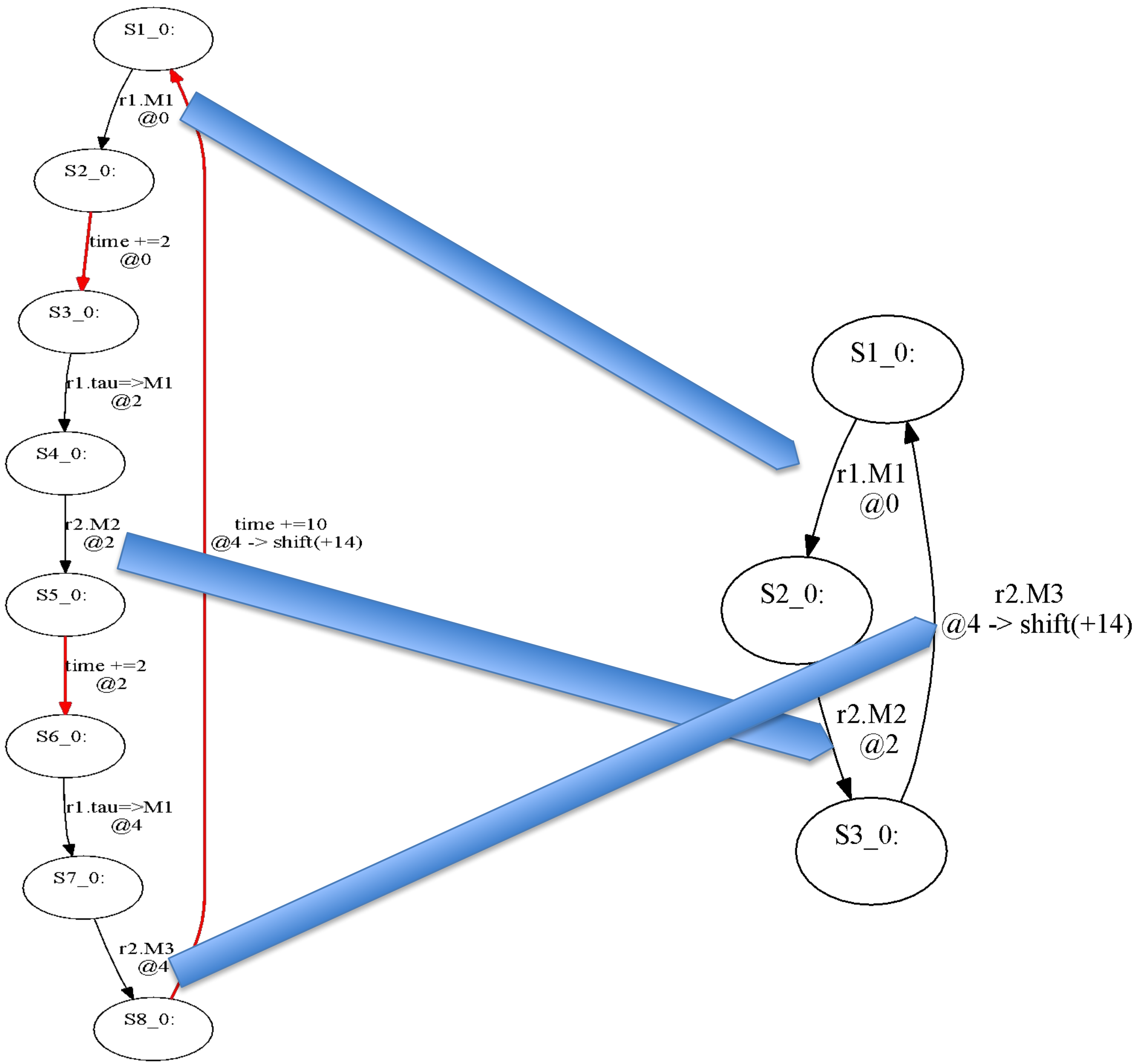
- Fou … re
  one … ex
- PCs … ted
- Unb … sta
  gen







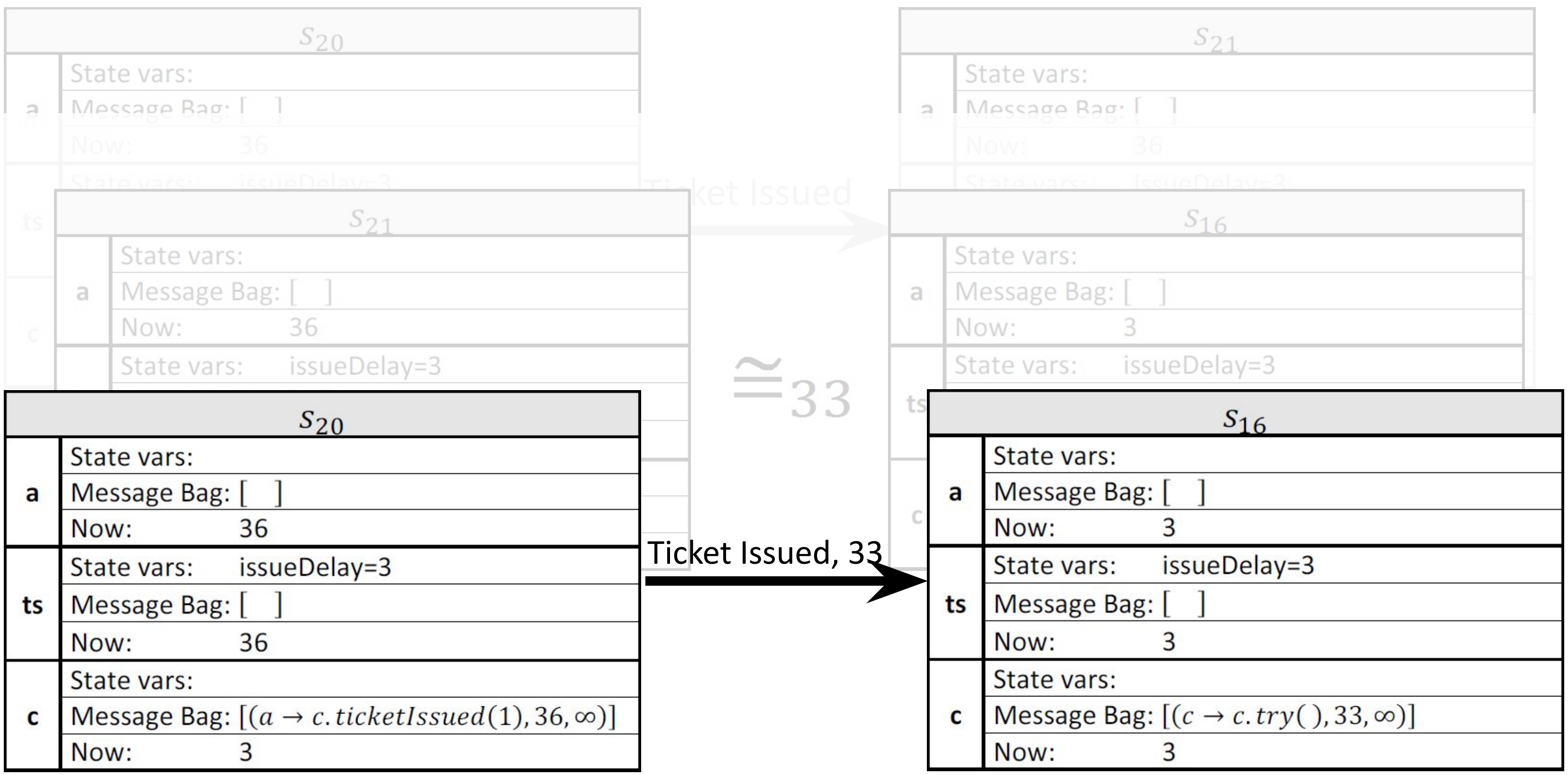
TTS versus FTTS



Bounded Floating-Time Transition System

- A notion of state equivalence by shifting the local times of rebecs
- Time in Timed-Rebeca models is relative
  - Uniform shift of time to past or future has no effect on the execution of statements

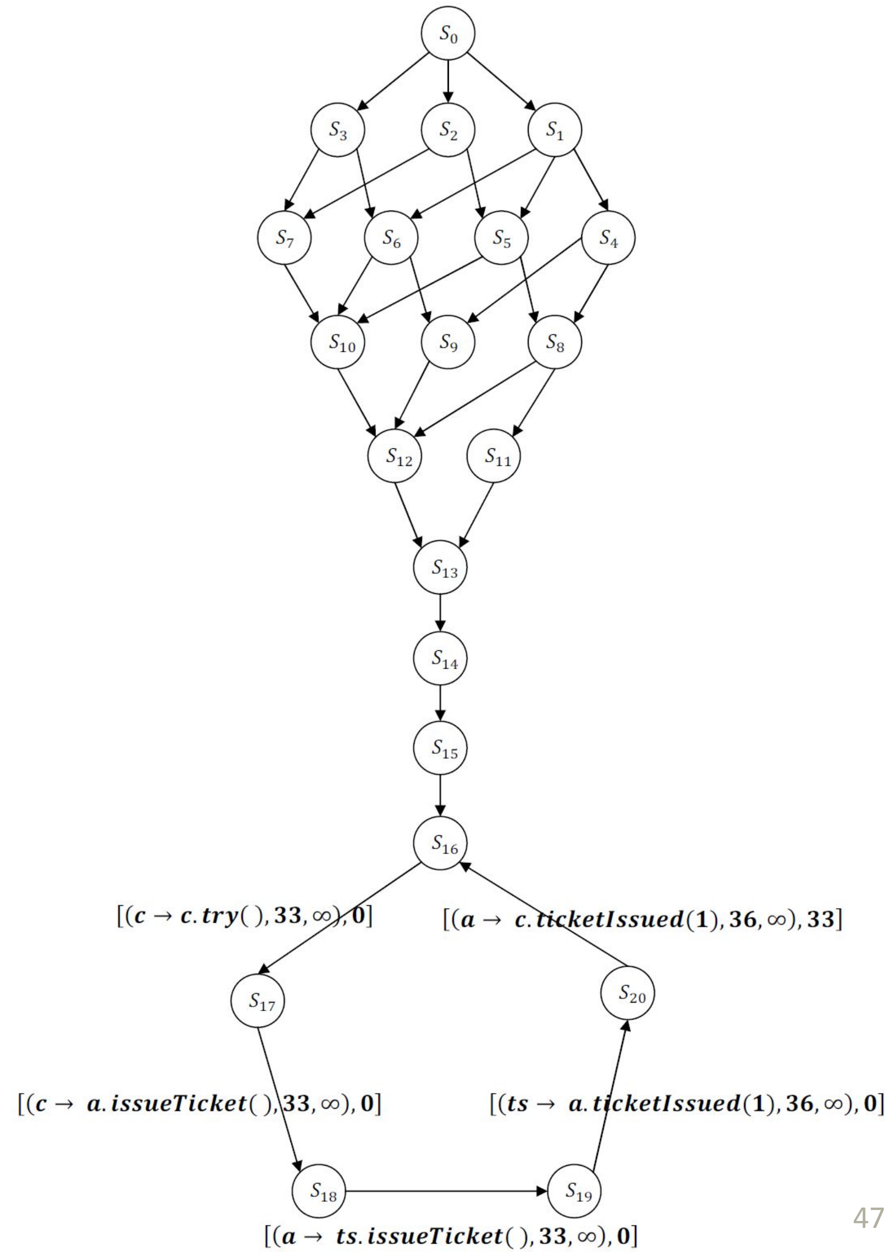
Bounding the Floating-Time Transition System



Bounded Floating-Time Transition System: an example

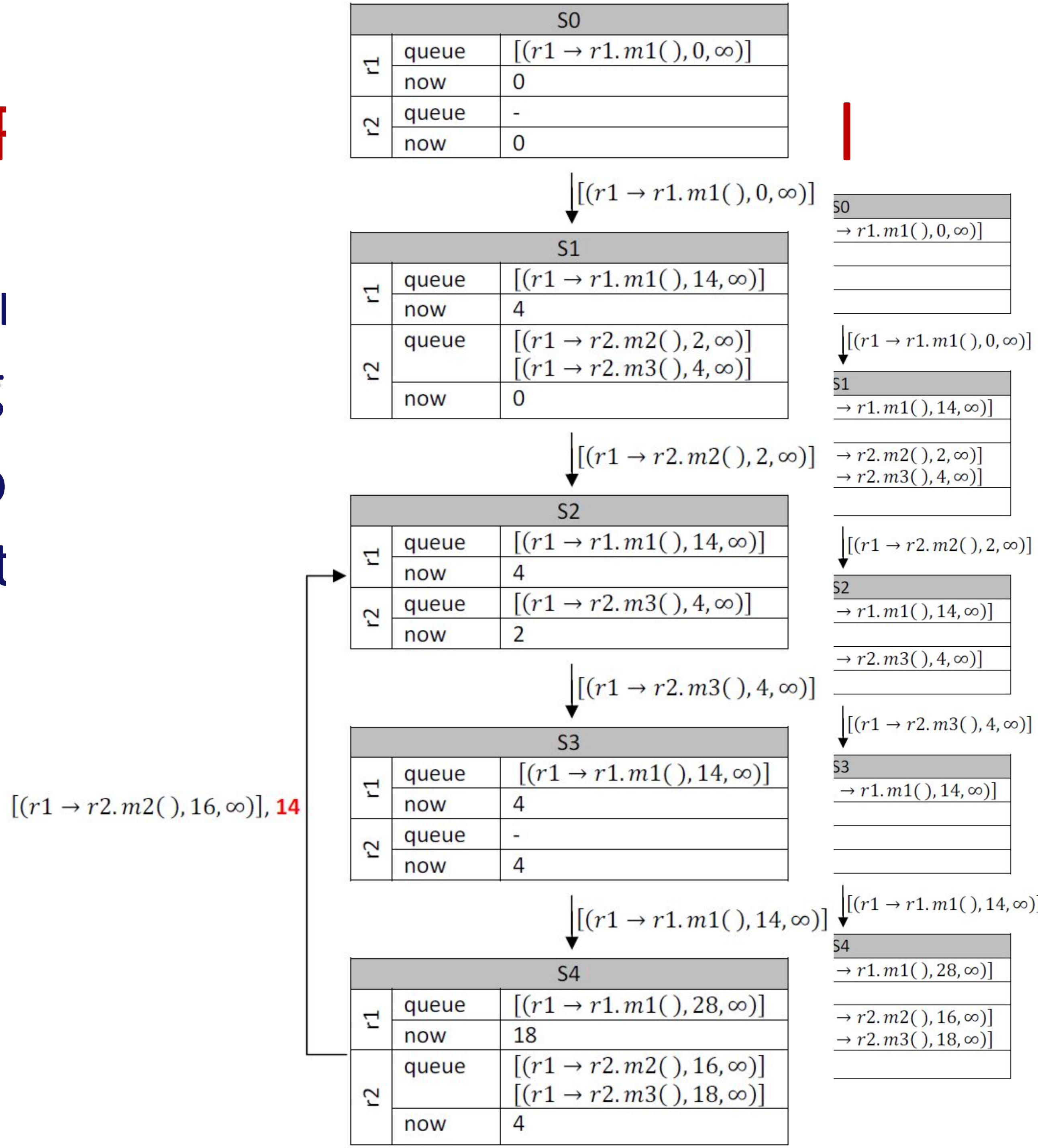
- A shift-time transition, between states 16 and 20

- Bounded floating-time transition system and floating-time transition system are bisimilar.



Bounded I

- Bounded ti system is g
- Contents o states are t as FTTS



Deadlock and schedulability check

- We keep the relative distance between values of all the timing values of each state (relative timing distances are preserved)
- Deadlines are set relatively so time shift has no effect on deadline-miss
- For checking “deadline missed” and “deadlock-freedom” relative time is enough

TTS vs FTTS State Space Size

- About 50% state space reduction

Model Name	Number of Rebecs	FTTS State Space Size	TTS State Space Size
Ticket Service System	3	6	12
	4	43	86
	5	282	532
	6	2035	3526
	7	17849	31500
CSMA/CD	4	54	108

Experimental results

- Three models, three tools

Problem	Size	Using BFTTS		Using Timed Automata		Using McErlang	
		#states	time	#states	time	#states	time
Ticket Service	1 customer	8	< 1 sec	801	<1 sec	150	<1 sec
	2 customers	51	< 1 sec	19M	5 hours	4.5k	3 secs
	3 customers	280	< 1 sec	-	>24 hours†	190K	5.1 mins
	4 customers	1.63K	< 1 sec	-	>24 hours†	> 4M‡	-
	5 customers	11K	< 1 sec	-	>24 hours†	> 4M‡	-
	6 customers	83K	2 secs	-	>24 hours†	> 4M‡	-
	7 customers	709K	3 mins	-	>24 hours†	> 4M‡	-
	8 customers	6.8M	9.7 hours	-	>24 hours†	> 4M‡	-
Sensor Network	1 sensor	183	< 1 sec	-	>24 hours†	> 6.5M‡	-
	2 sensors	2.4K	< 1 sec	-	>24 hours†	> 6M‡	-
	3 sensors	33.6K	1 sec	-	>24 hours†	> 6M‡	-
	4 sensors	588K	13 secs	-	>24 hours†	> 6M‡	-
Slotted ALOHA Protocol	1 interface	68	< 1 sec	-	>24 hours†	153K	1.8 secs
	2 interfaces	750	< 1 sec	-	>24 hours†	> 2.8M‡	-
	3 interfaces	7.84K	1 sec	-	>24 hours†	> 2.8M‡	-
	4 interfaces	45.7K	6 secs	-	>24 hours†	> 2.8M‡	-
	5 interfaces	331K	64 secs	-	>24 hours†	> 2.8M‡	-

Table 1: Model checking time and size of state space, using three different tools. The † sign on the reported time shows that model checking takes more than the time limit (24 hours). The ‡ sign on the reported number of states shows that state space explosion occurs as the model checker want to allocate more than 16GB in memory which is more than total amount of memory.

Our reduction technique: distilled

- Event-based analysis - maximum progress of time based on events (not timer ticks)
  - Generating no new states because of delays, each rebec has its own local time in each state
- Making use of isolated message server execution of actors
  - no shared variables, no blocking send or receive, single-threaded actors, non-preemptive execution of each message server
- Check the state equivalence by shifting the local times of concurrent elements in case of recurrent behaviors

Comparing to others

- Real-time Maude
  - It ticks … so, explosion
  - Bounded model checking

- Timed Automata
  - Produce many automata and many clocks for an asynchronous system – so, explosion

A Point:
FTTS, Considering only the time-tags

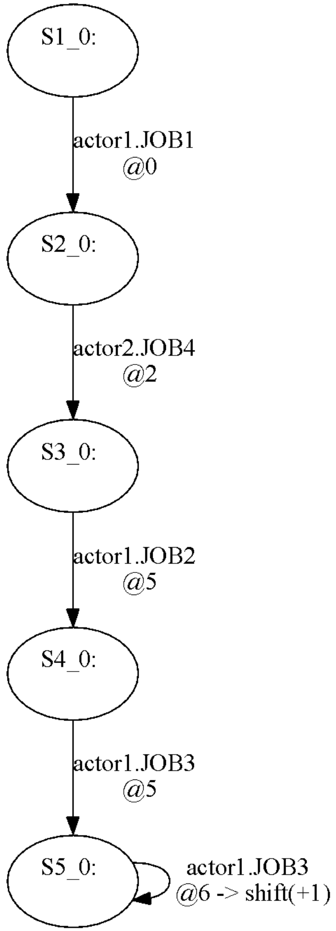
```
reactiveclass Actor1(3) {
    Actor1() {
        self.job1();
    }
    msgsrv job1() {
        self.job2() after(1);
        delay(5);
    }
    msgsrv job2() {
    }
    msgsrv job3() {
        self.job3() after(1);
    }
}
```

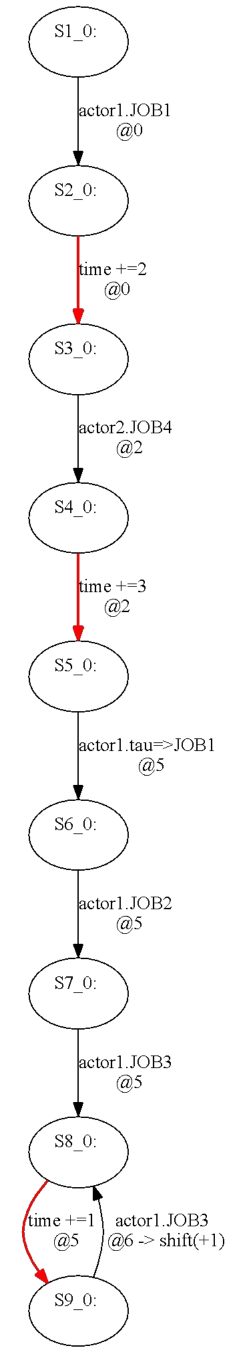
```
reactiveclass Actor2(3) {
    knownrebecs {
        Actor1 a1;
    }
    Actor2() {
        self.job4() after(2);
    }
    msgsrv job4() {
        a1.job3() after(2);
    }
}


main {
    Actor1 actor1():();
    Actor2 actor2(actor1):();
}
```

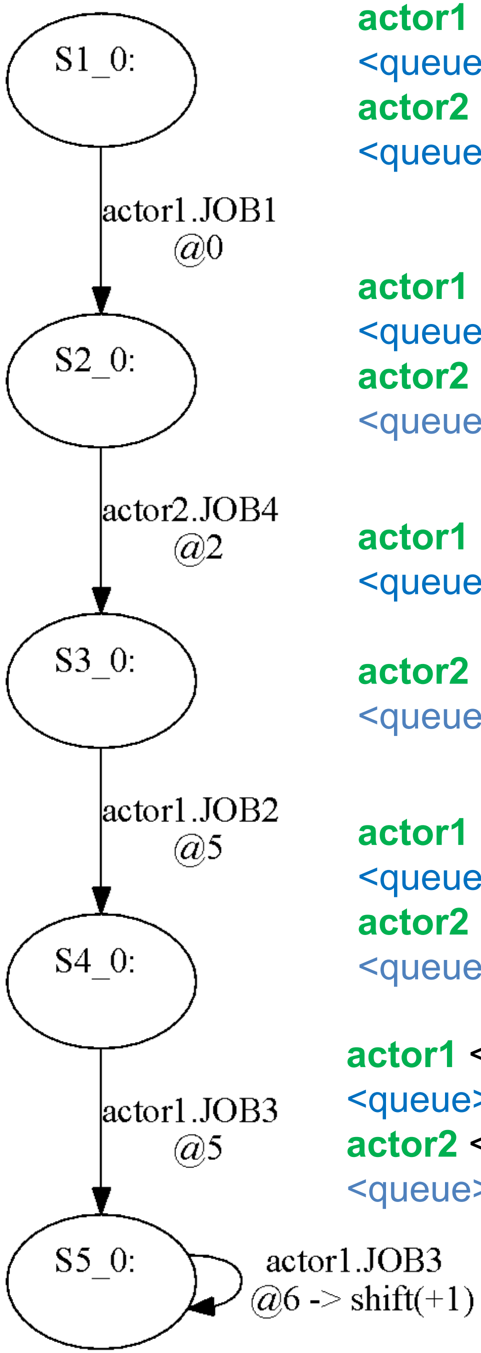
Simple FTTS: Consider only Smallest Time Tag

```
reactiveclass Actor1(3) {
    Actor1() {
        self.job1();
    }
    msgsrv job1() {
        self.job2() after(1);
        delay(5);
    }
    msgsrv job2() {
    }
    msgsrv job3() {
        self.job3() after(1);
    }
}
```

```
reactiveclass Actor2(3) {
    knownrebecs {
        Actor1 a1;
    }
    Actor2() {
        self.job4() after(2);
    }
    msgsrv job4() {
        a1.job3() after(2);
    }
}

main {
    Actor1 actor1():();
    Actor2 actor2(actor1):();
}
```







actor1 <now>0
<queue> arrival="0" deadline="infinity" sender="actor1">job1
actor2 <now>0
<queue>arrival="2" deadline="infinity" sender="actor2">job4

actor1 <now>5
<queue> arrival="1" deadline="infinity" sender="actor1">job2
actor2 <now>2
<queue> arrival="2" deadline="infinity" sender="actor2">job4

actor1 <now>5
<queue> arrival="1" deadline="infinity" sender="actor1">job2
arrival="4" deadline="infinity" sender="actor2">job3
actor2 <now>5
<queue>

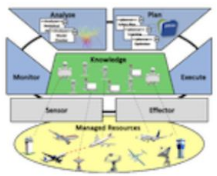
actor1 <now>5
<queue> arrival="4" deadline="infinity" sender="actor2">job3
actor2 <now>5
<queue>

actor1 <now>6
<queue> arrival="
actor2 <now>6
<queue>

```
reactiveclass Actor1(3) {
   Actor1() {self.job1();}
   msgsrv job1() {
      self.job2() after(1);
      delay(5);}
   msgsrv job2() {}
   msgsrv job3() {
      self.job3() after(1);}
}
```

```
reactiveclass Actor2(3) {
   knownrebecs {Actor1 a1;}
   Actor2() {
      self.job4() after(2);}
   msgsrv job4() {
      a1.job3() after(2);}
}
main {
   Actor1 actor1():();
   Actor2 actor2(actor1):();
}
```

Projects



SEADA

In SEADA (Self-Adaptive Actors) we will use Ptolemy to represent the architecture, and extensions of Rebeca for modeling and verification. Our models@runtime will be coded in an extension of Probabilistic Timed Rebeca, and supporting tools for customized run-time formal verification

RoboRebeca

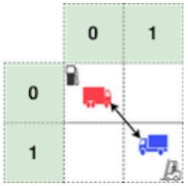
RoboRebeca is a framework which provides facilities for developing safe/correct source codes for robotic applications. In RoboRebeca, models are developed using Rebeca family language and automatically transformed into ROS compatible source codes. This framework is

HybridRebeca

Hybrid Rebeca, is an extension of actor-based language Rebeca, to support modeling of cyber-physical systems. In this extension, physical actors are introduced as new computational entities to encapsulate the physical behaviors. Learn more

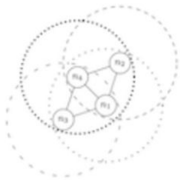
Tangramob

Tangramob offers an Agent-Based



AdaptiveFlow

AdaptiveFlow is an actor-based eulerian

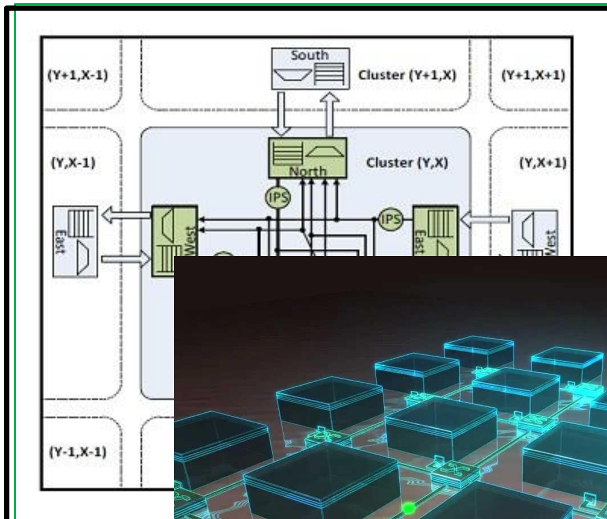


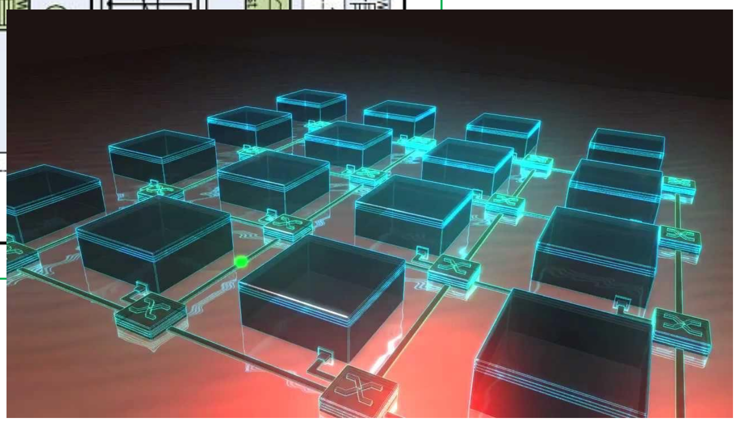
wRebeca

wRebeca is an actor-based modeling

Design Decisions
Network on Chip

Siamak Mohammadi, Zeinab Sharifi, UT



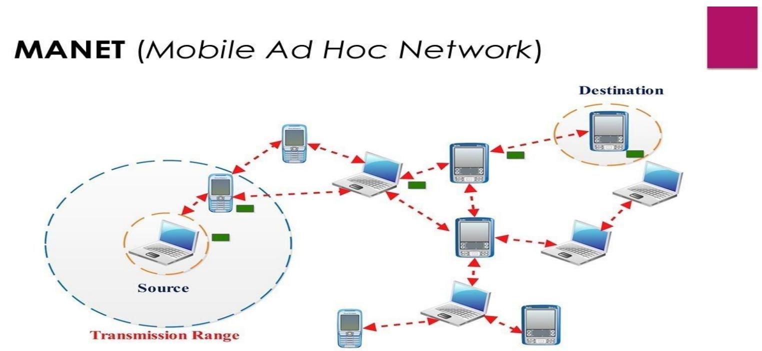


Design Decisions:
routing algorithms
Buffer length
Memory Allocation

Zeinab Sharifi, Mahdi Mosaffa, Siamak Mohammadi, and Marjan Sirjani: Functional and Performance Analysis of Network-on-Chips Using Actor-based Modeling and Formal Verification, AVoCS, 2013.
https://rebeca-lang.org/assets/papers/2013/Performance-Analysis-of-NoC.pdf

Bug Check
Network Protocols

Fatemeh Ghassemi, Ramtin Khosravi, UT



Deadlock and loop-freedom of Mobile Adhoc Networks

Behnaz Yousefi, Fatemeh Ghassemi, and Ramtin Khosravi: Modeling and Efficient Verification of Wireless Ad hoc Networks, volume 29, Issue 6, pp 1051–1086, Formal Aspects of Computing, 2017.
https://link.springer.com/article/10.1007/s00165-017-0429-z

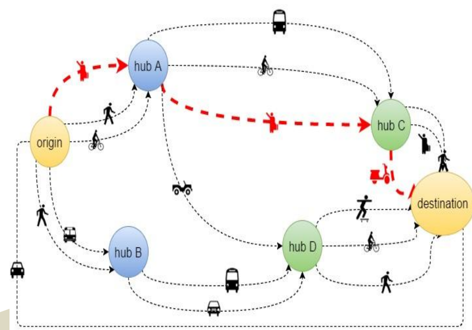
Performance Optimization
Smart Structures
Gul Agha, OSI, UIUC and Ehsan Khamespanah, UT

Resource Management
Smart Transport Hubs
Andrea Polini, Francesco De Angelis, Unicam Smart Mobility Lab.


Not only Safety and Rob[illegible]
but also Performance, Cost and
User Satisfaction

Schedulability Ana[illegible] of Distributed Real-Time Sensor Network: Finding the best configuration

Ehsan Khamespanah, Kirill Mechitov, Marjan Sirjani, Gul Agha: Modeling and Analyzing Real-Time Wireless Sensor and Actuator Networks Using Actors and Model Checking, Software Tools for Technology Transfer, 2017.
https://rebeca-lang.org/assets/papers/2017/Modeling-and-Analyzing-Real-Time-Wireless-Sensor-and-Actuator-Networks-Using-Actors-and-Model-Checking.pdf



Minimize:

Number of service disruptions

Number of mobility resources in smart hubs

Cost of mobility for commuters

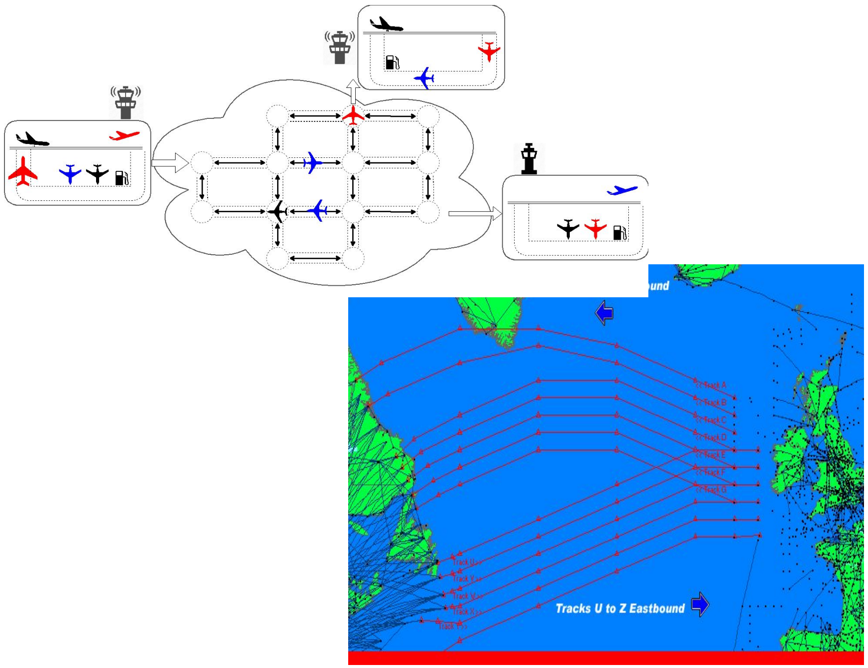
Travel time for commuters

Travel distance for commuters

Jacopo de Berardinis, Giorgio Forcina, Ali Jafari, Marjan Sirjani:
Actor-based macroscopic modeling and simulation for smart urban planning. Sci. Comput. Program. 168: 142-164 (2018)
https://www.sciencedirect.com/science/article/pii/S0167642318303459?via%3Dihub

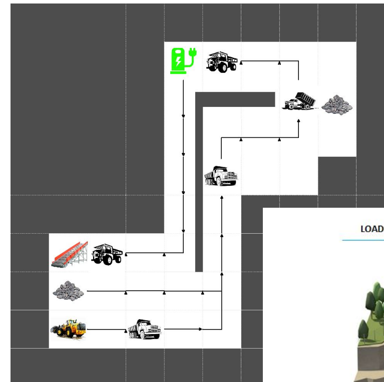
Adaptive Flow Management
Air Traffic Control
UC Berkeley, Edward Lee and Sharif, Ali Movaghar

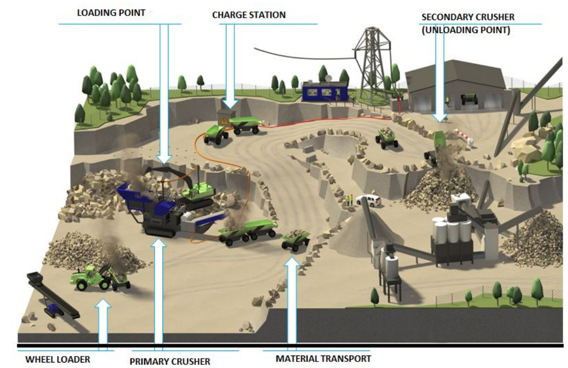
Adaptive Flow Management
Volvo CE Quarry Site
Volvo-CE, Stephan Baumgart and Torbjörn Martinsson



Adaptive Air Traffic Control: Safe rerouting of airplanes using Magnifier

Maryam Bagheri, Marjan Sirjani, Ehsan Khamespanah, Christel Baier, Ali Movaghar, Magnifier: A Compositional Analysis Approach for Autonomous Traffic Control, IEEE Transactions on Software Engineering, 2021
https://rebeca-lang.org/assets/papers/2021/Magnifier-A-Compositional-Analysis-Approach-for-Autonomous-Traffic-Control.pdf

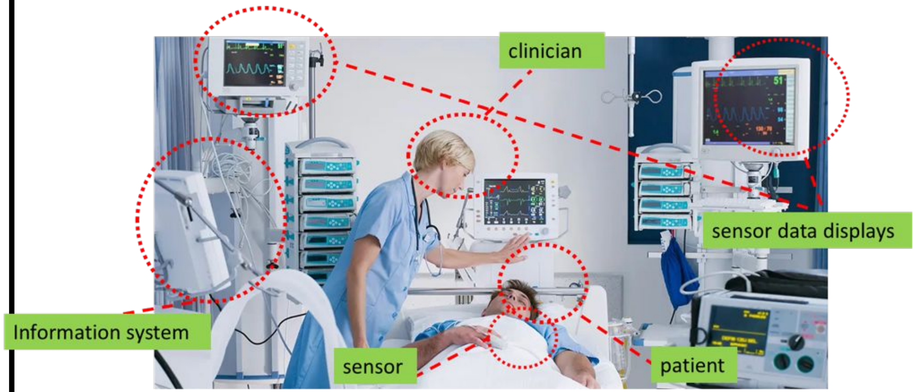




Safe and optimized fleet control

Marjan Sirjani, Giorgio Forcina, Ali Jafari, Stephan Baumgart, Ehsan Khamespanah, Ali Sedaghatbaf: An Actor-based Design Platform for System of Systems, IEEE 43th Annual Computers, Software, and Applications Conference (COMPSAC), 2019
https://rebeca-lang.org/assets/papers/2019/An-Actor-based-Design-Platform-for-System-of-Systems.pdf

Time Analysis
Connected Medical Systems
John Hatcliff, U. of Kansas, and Fatemeh Ghassemi, UT



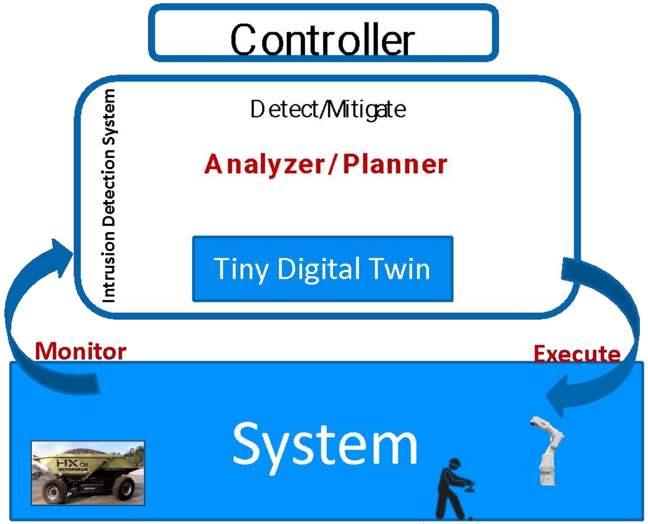
Local properties of devices are assured by the vendors at the development time.

Verify the satisfaction of timing communication requirements.

Helpful for dynamic network configuration or capacity planning.

Mahsa Zarneshan, Fatemeh Ghassemi, Ehsan Khamespanah, Marjan Sirjani, John Hatcliff: Specification and Verification of Timing Properties in Interoperable Medical Systems. Log. Methods Comput. Sci. 18(2) (2022)
https://lmcs.episciences.org/9639

Anomaly Detection
Model-Based Cyber-Security
SRI, Carolyn Talcott

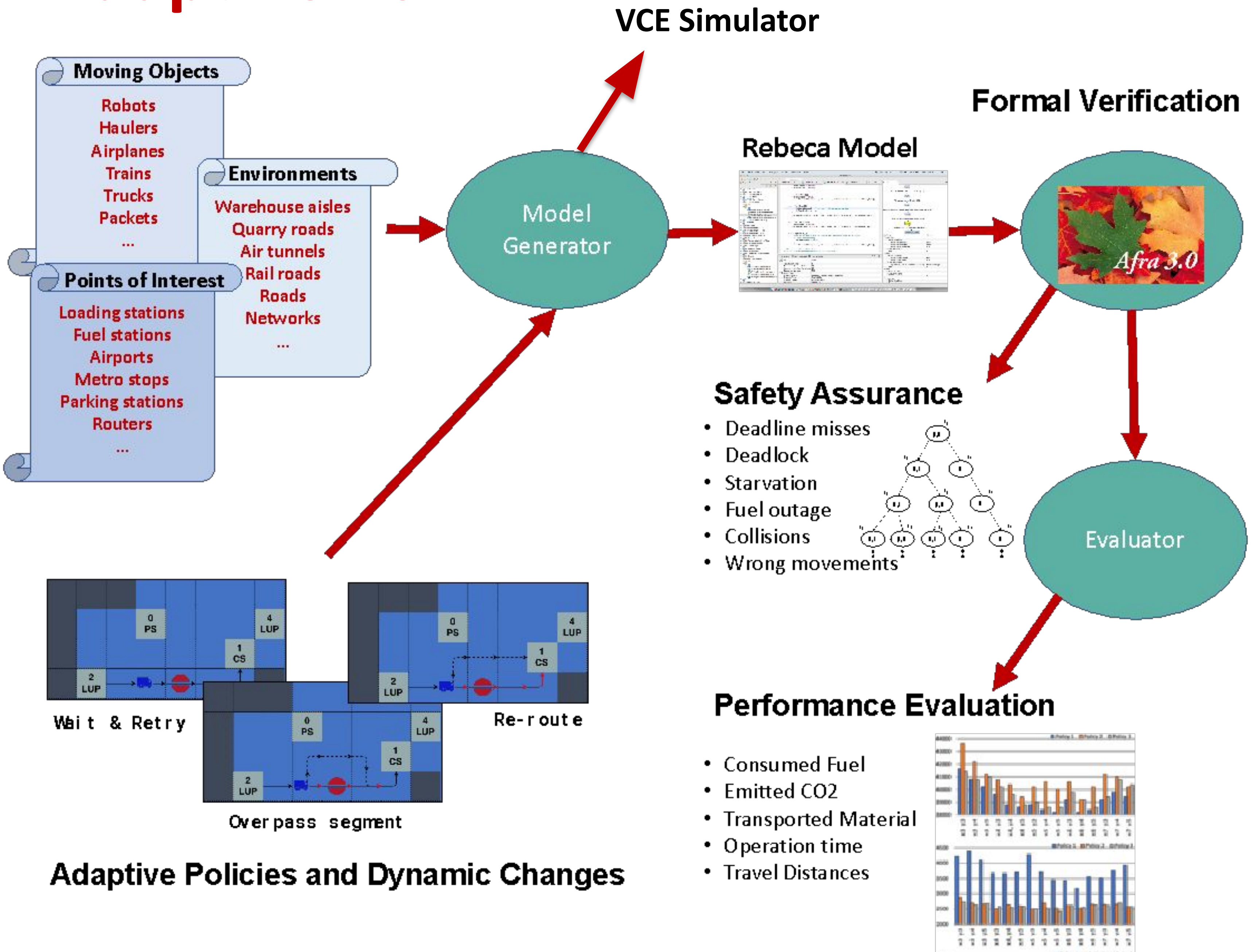


MAPE-K architecture
(Monitor- Analysis – Plan – Execute)- Knowledge

- Runtime **monitor** to check the system behavior using a **Tiny Digital Twin**

Fereidoun Moradi, Maryam Bagheri, Hanieh Rahmati, Hamed Yazdi, Sara Abbaspour Asadollah, Marjan Sirjani, Monitoring Cyber-Physical Systems using a Tiny Twin to Prevent Cyber-Attacks, 28th International Symposium on Model Checking of Software (SPIN), 2022
https://rebeca-lang.org/assets/papers/2022/Monitoring-Cyber-Physical-Systems-Using-a-Tiny-Twin-to-Prevent-Cyber-Attacks.pdf

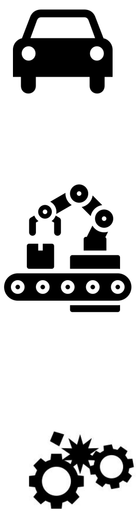
AdaptiveFlow

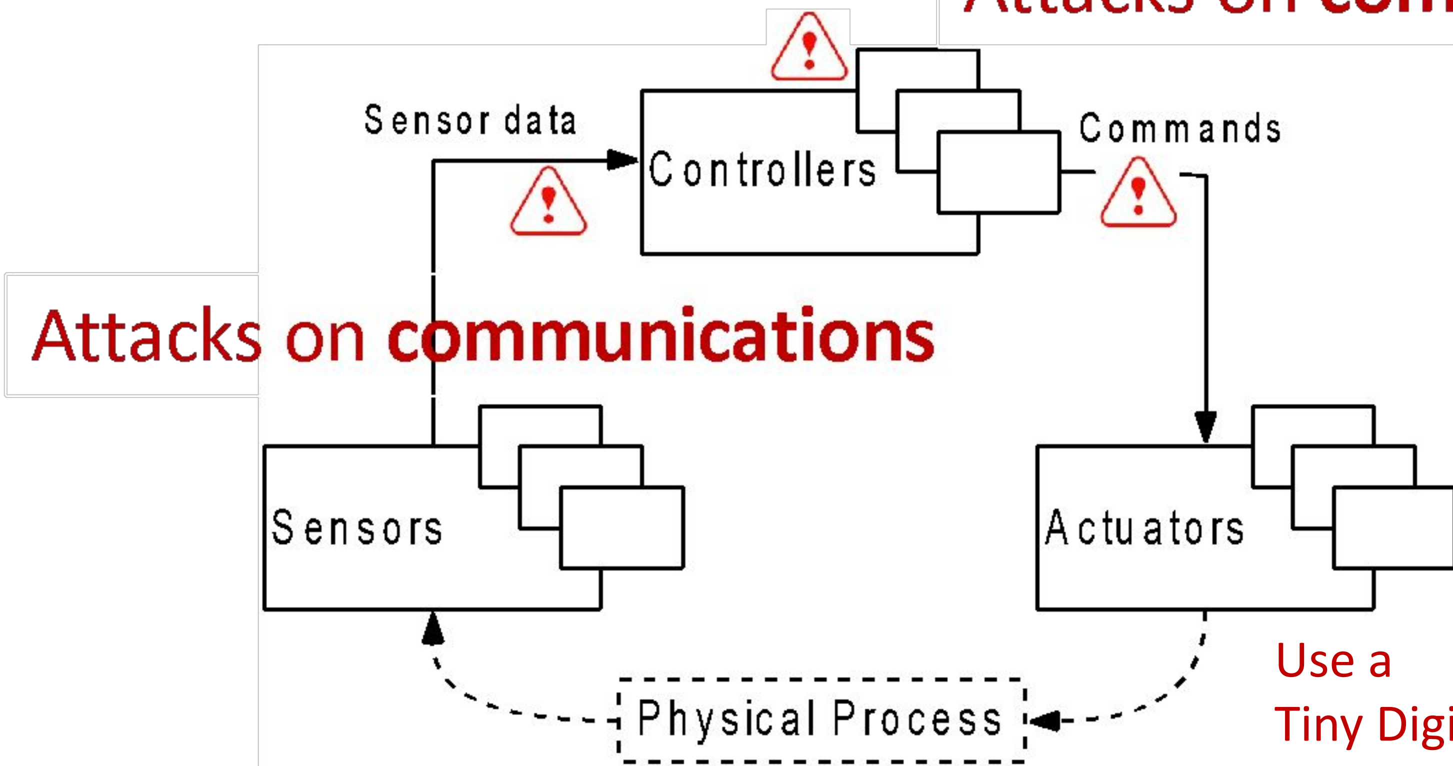


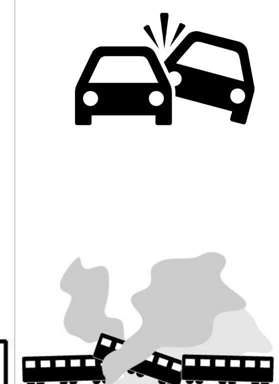
Cyber-Security Assurance Using Model Checking and Monitoring

Find the attacks like finding the anomalies

Attacks on **components**



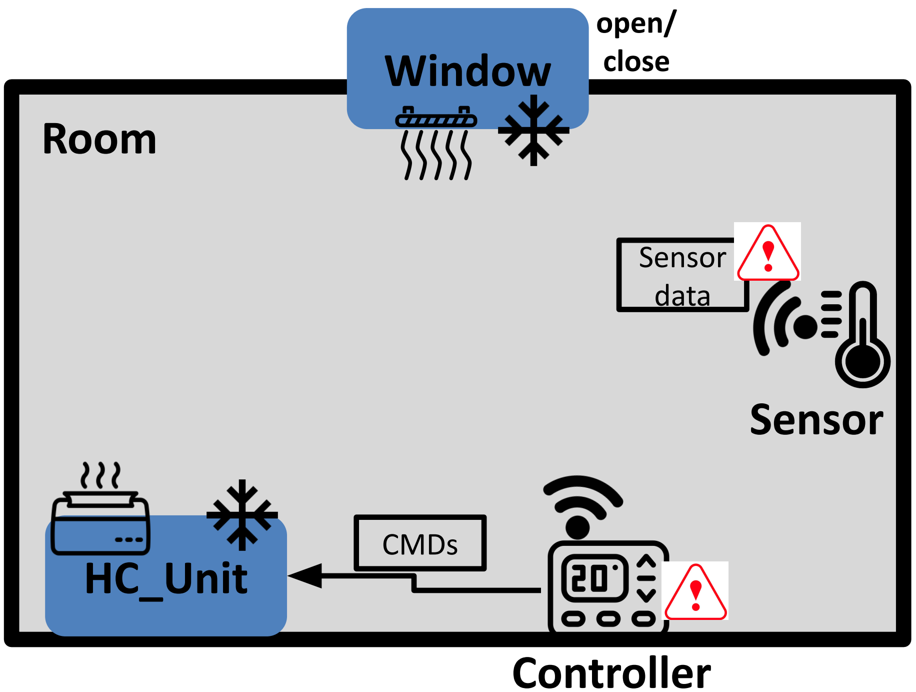




Use a
Tiny Digital Twin
as a reference model

Cyber-Physical Systems (CPSs)

Monitoring at Runtime
Temperature Control System (TPS)



The wireless communication network is vulnerable to malicious cyber-attacks!!

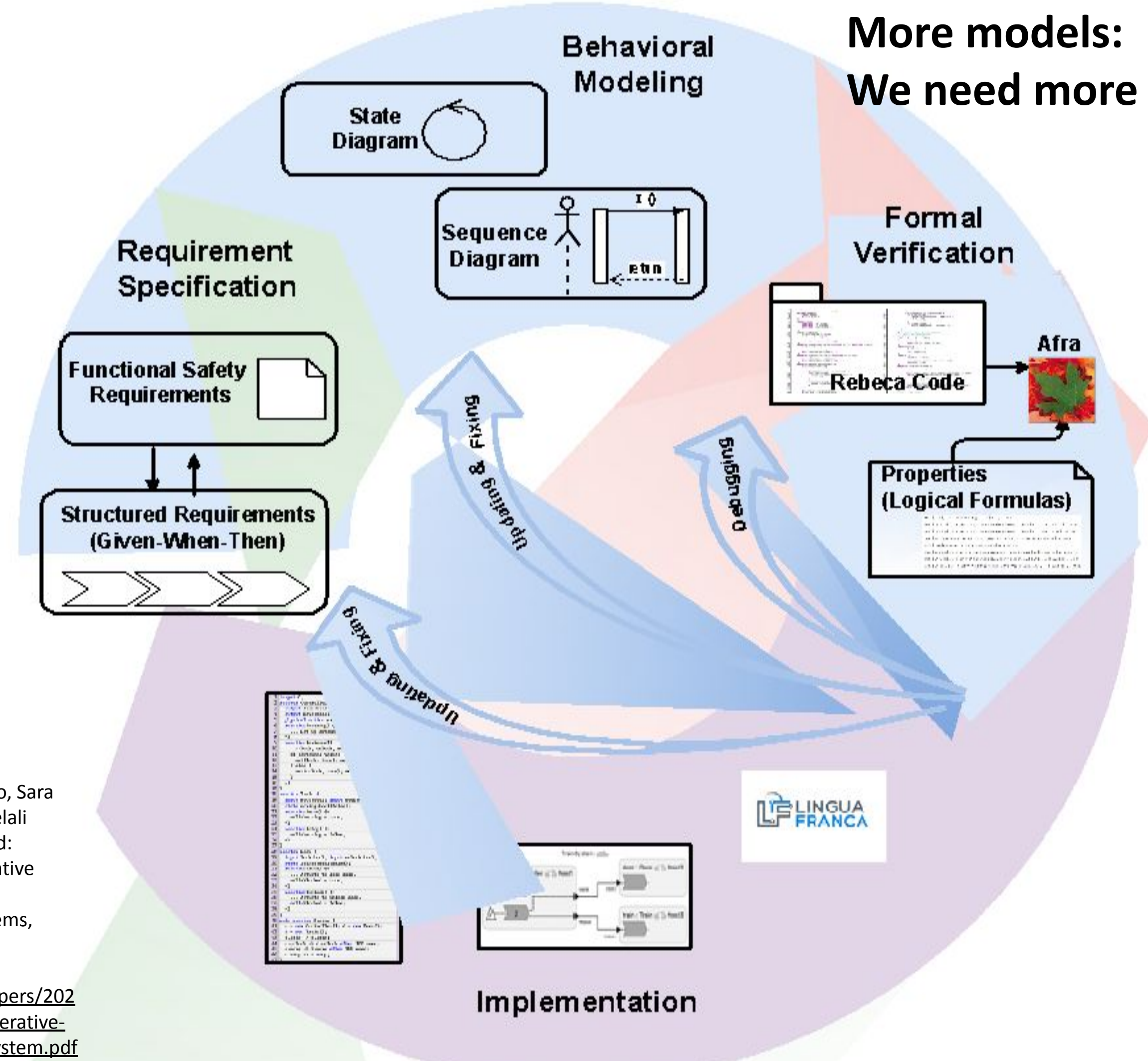
Sensor Data:
Temperature value

Commands:
Activate Heating/Cooling
Switch off

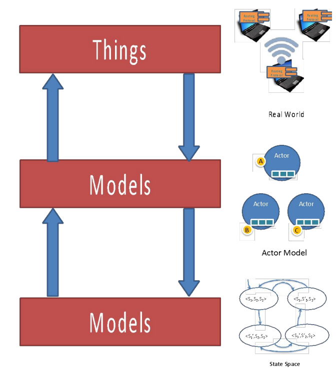
ATTACKs:
Dropping packets
False sensor data injection
Faulty control commands

DAMAGEs:
Degrades the temperature regulation process,
Pushes temperature value out of the defined range

Verification-Driven Iterative Development of Cyber-Physical System



More models:
We need more models at each phase



Marjan Sirjani, Luciana Provenzano, Sara Abbaspour Asadollah, Mahshid Helali Moghadam, Mehrdad Saadatmand: Towards a Verification-Driven Iterative Development of Software for Safety-Critical Cyber-Physical Systems, Journal of Internet Services and Applications, 2021
https://rebeca-lang.org/assets/papers/2020/Towards-a-Verification-Driven-Iterative-Development-of-Cyber-Physical-System.pdf

Verification of Cyber-Physical Systems

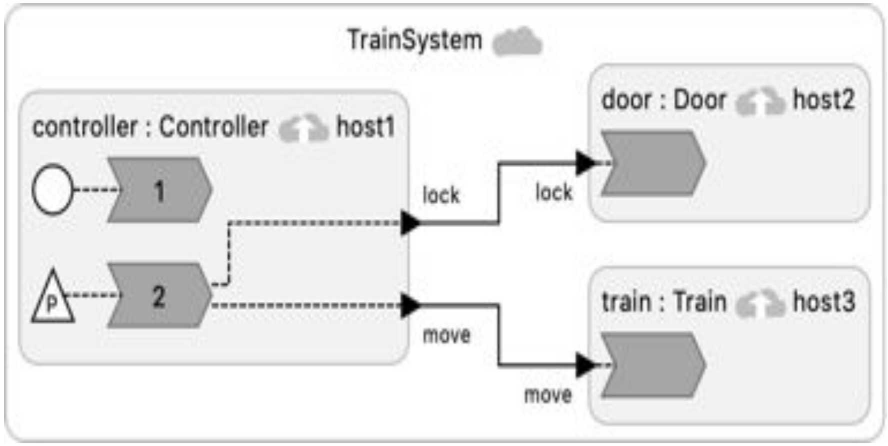
(UC Berkeley, Edward Lee)

Lingua Franca is a programming language based on the Reactor model of computation for building cyber-physical systems.

Reactors and Rebeca: Natural mapping of semantics (similar syntax)

LINGUA FRANCA

A polyglot meta-language for deterministic, concurrent, time-sensitive systems.



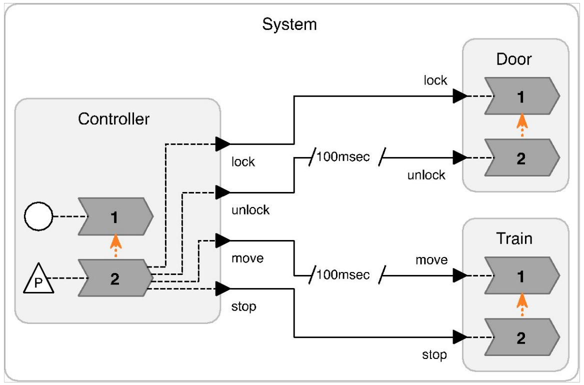
Marten Lohstroh , Martin Schoeberl, Andrés Goens, Armin Wasicek, Christopher D. Gill, Marjan Sirjani, Edward A. Lee: Actors Revisited for Time-Critical Systems. DAC 2019: 152

Verification of cyberphysical systems
M Sirjani, EA Lee, E Khamespanah
Mathematics 8 (7), 1068, 2020

```
target C;
reactor Controller {
  output lock:bool; output unlock:bool;
  output move:bool; output stop:bool;
  physical action external:bool;
  reaction(startup) {=
    ... Set up external sensing.
  =}
  reaction(external)
      ->lock, unlock, move, stop {=
    if (external_value) {
      set(lock, true); set(move, true);
    } else {
      set(unlock, true); set(stop, true);
    }
  =}
}
reactor Train {
  input move:bool; input stop:bool;
  state moving:bool(false);
  reaction(move) {=
    self->moving = true;
  =}
  reaction(stop) {=
    self->moving = false;
  =}
}
reactor Door {
  input lock:bool; input unlock:bool;
  state locked:bool(false);
  reaction(lock) {=
    ... Actuate to lock door.
    self->locked = true;
  =}
  reaction(unlock) {=
    ... Actuate to unlock door.
    self->locked = false;
  =}
}
main reactor System {
  c = new Controller(); d = new Door();
  t = new Train();
  c.lock -> d.lock;
  c.unlock -> d.unlock after 100 msec;
  c.move -> t.move after 100 msec;
  c.stop -> t.stop;
}
```

```
reactiveclass Controller(5) {
  knownrebecs{
      Door door; Train train;
  }
  statevars { boolean moveP;}
  Controller() {
    moveP = true;
    self.external_move();
  }
  msgsrv external_move() {
    int d =
    int x =
    int ext
    if (mov
          d
          t
      } els
        doc
        tra
      }
      moveF
      self.
  } }
 reactiveclas
  statevars
    boolean
    }
  Train() {
    moving
    }
  @priority
    moving = false;
  }
  @priority(2) msgsrv move() {
    moving = true;
} }
reactiveclass Door(10) {
  statevars{
    boolean is_locked;
  }
   Door() {
    is_locked = false;
   }
  @priority(1) msgsrv lock () {
      is_locked = true;
  }
  @priority(2) msgsrv unlock () {
      is_locked = false;
  }
}
main {
   @priority(1) Controller controller(door,
                            train):();
   @priority(2) Train train():();
   @priority(2) Door door():();
}
```

Lingua Franca Construct/Features	Timed Rebeca Construct/Features
reactor	*reactiveclass*
reaction	*msgsrv*
trigger	*msgsrv name*
state	*statevars*
input	*msgsrv*
output	*known rebecs*
physical action	*msgsrv*
implicit in the topology	*Priority*
main	*main*
instantiation (***new***)	*instantiation of rebecs*
connection	*implicit in calling message servers*
after	*after*
–	*delay*



Experience Distilled as Transparent Actors

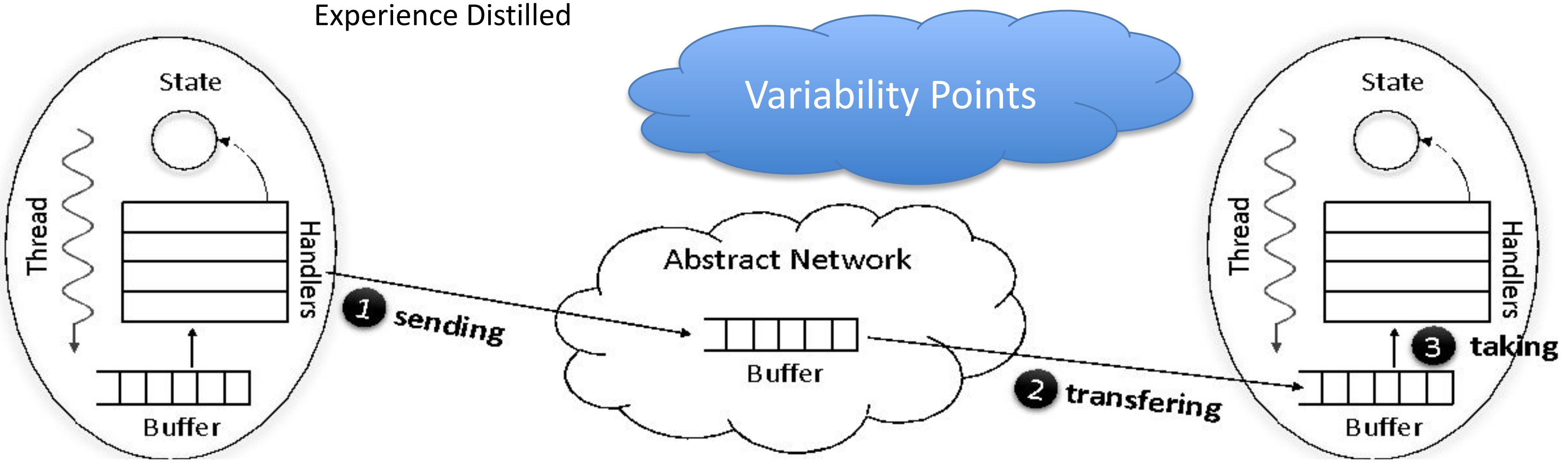
- Looking into different application domains
  - Scheduling and end-to-end delays of Sensor Networks and Cyber-Physical Systems
    - Volvo cars, Volvo Trucks, Deif – Smart Structures (Gul Agha), Interoperable Medical Systems (John Hatcliff)
  - Optimisation of Flow Management
    - Volvo CE, Isavia, NoC (Siamak Mohammadi, Smart Hubs (Andrea Polini)
  - Model Checking Network Protocols, CPS
    - AODV, LF, all the above

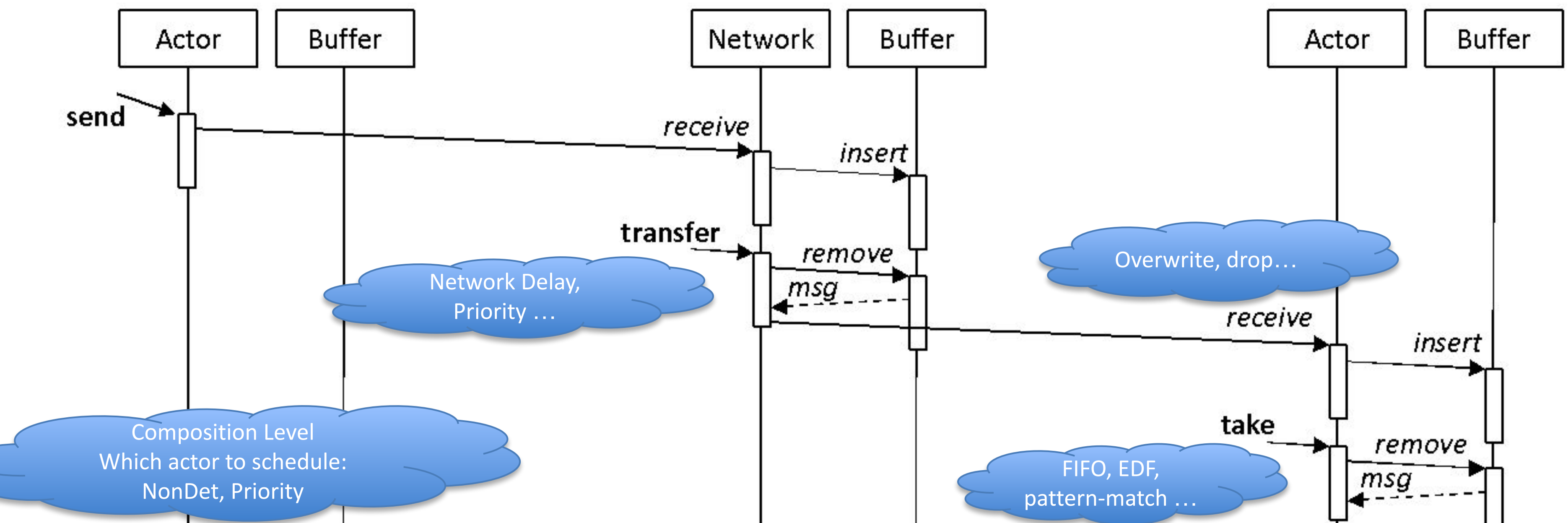
- Different Actor-based Languages
  - Rebeca, Timed Rebeca, Hewitt-Agha actor-based languages
  - Creol, ABS, Concurrent object languages
  - Lingua Franca and Edward Lee’s actors

Transparent Actors

Experience Distilled

(Fatemeh Ghassemi, Ehsan Khamespanah, Hossein Hojjat, 2023)





Fatemeh Ghassemi, Marjan Sirjani, Ehsan Khamespanah, Mahrokh Mirani, Hossein Hojjat: Transparent Actor Model. FormaliSE 2023: 97-107

References

- For publications, see

http://rebeca-lang.org/publications

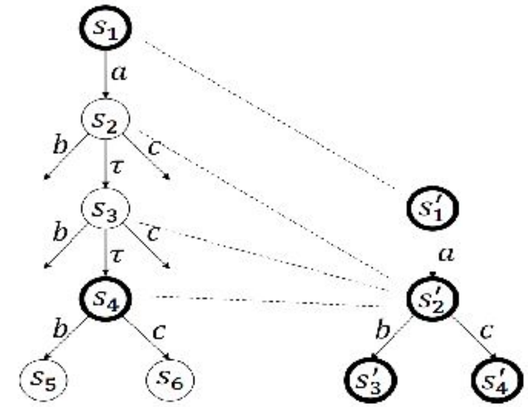
- For projects, see

http://rebeca-lang.org/projects

- QUESTIONS?

The Big Theorem

Theorem 1. *The relation R is an action-based weak bisimulation relation between states of TTS and FTTS.*



Part of a state space in TTS

Part of a state space in FTTS

- **s** $\overset{\textbf{tmsg}}{!}$ **t** completing traces are considered
- **s** ! **t** Stuttering of s

Corollary 1. *Transition systems of Timed Rebeca models in TTS and FTTS are equivalent with respect to all formulas that can be expressed in modal μ-calculus with weak modalities where the actions are taking messages from bags.* tu

22

Corollary 1. Transition systems of Timed Rebeca models in TTS and FTTS are equivalent with respect to all formulas that can be expressed in modal μcalculus with weak modalities where the actions are taking messages from bags.

Timed Rebeca Model of Ping-Pong

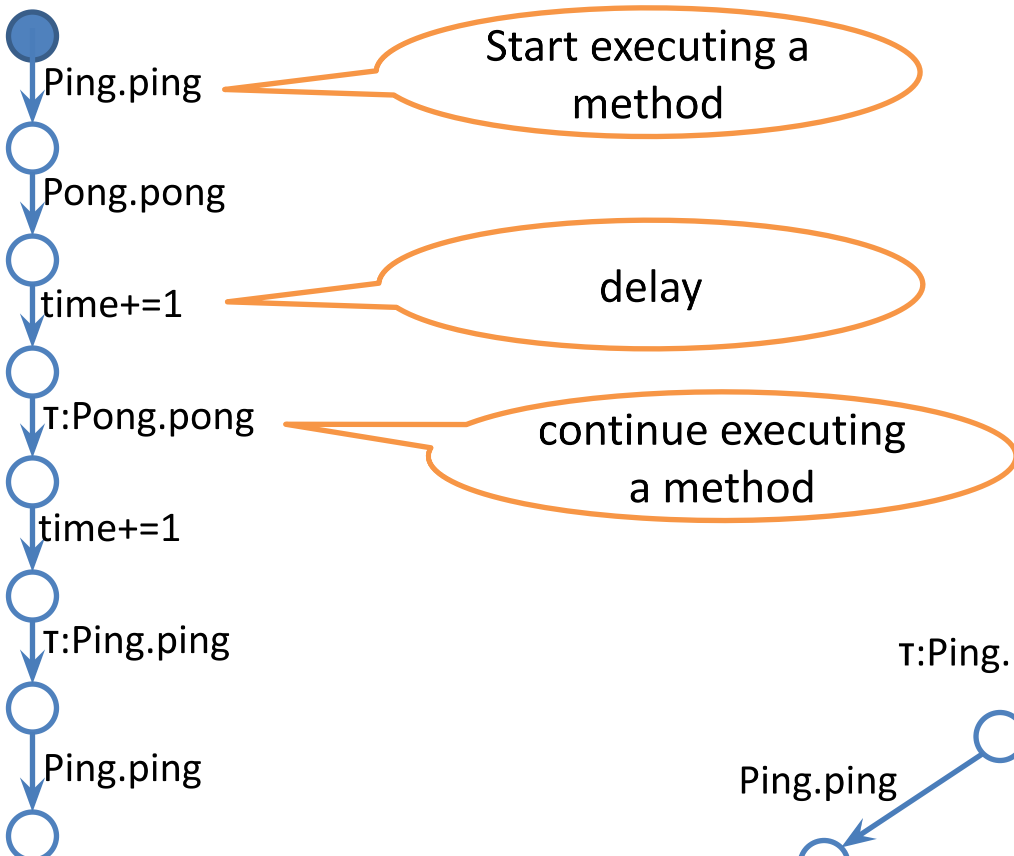
```
reactiveclass Ping(3) {
    knownrebecs {Pong pong;}
    Ping() {
      self.ping();
    }
    msgsrv ping() {
      pong.pong() after(1);
      delay(2);
    }
}
```

```
reactiveclass Pong(3) {
    knownrebecs {Ping ping;}
    Pong() {
    }
    msgsrv pong() {
      ping.ping() after (1) deadline(2);
      delay(1);
    }
}
```

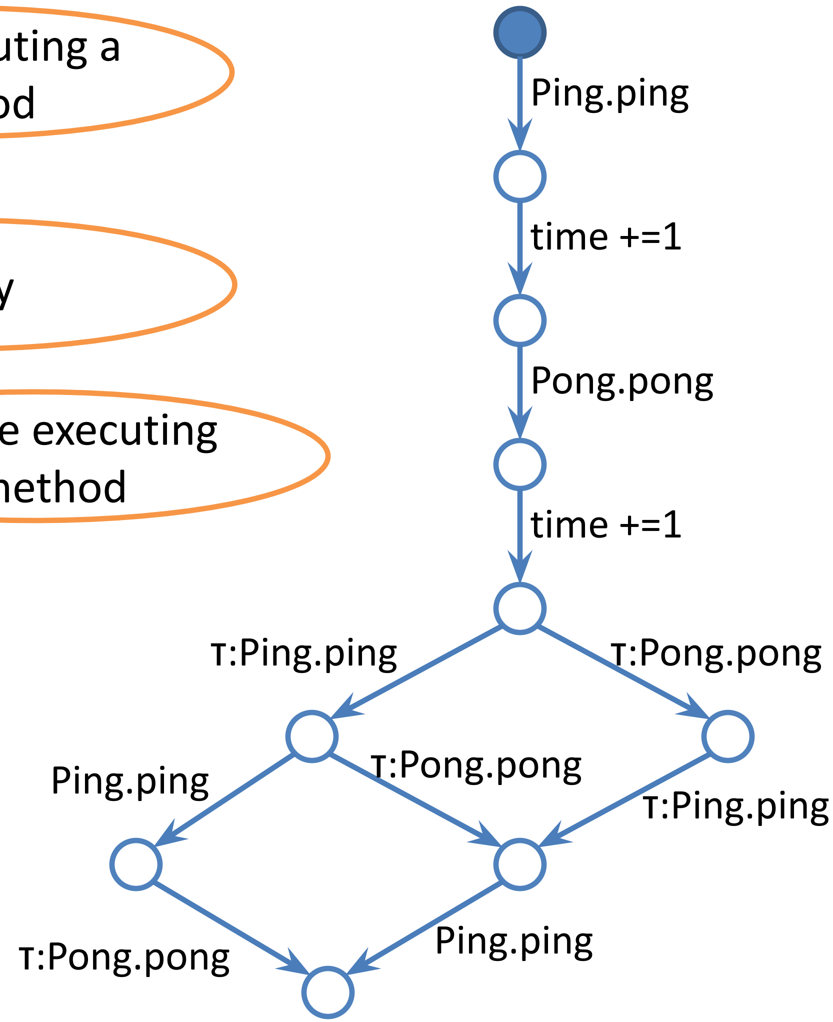
```
main {
    Ping ping(pong):();
    Pong pong(ping):();
}
```

Timed Transition System of Ping-Pong

Without ***after*** and ***deadline***



With ***after*** and ***deadline***



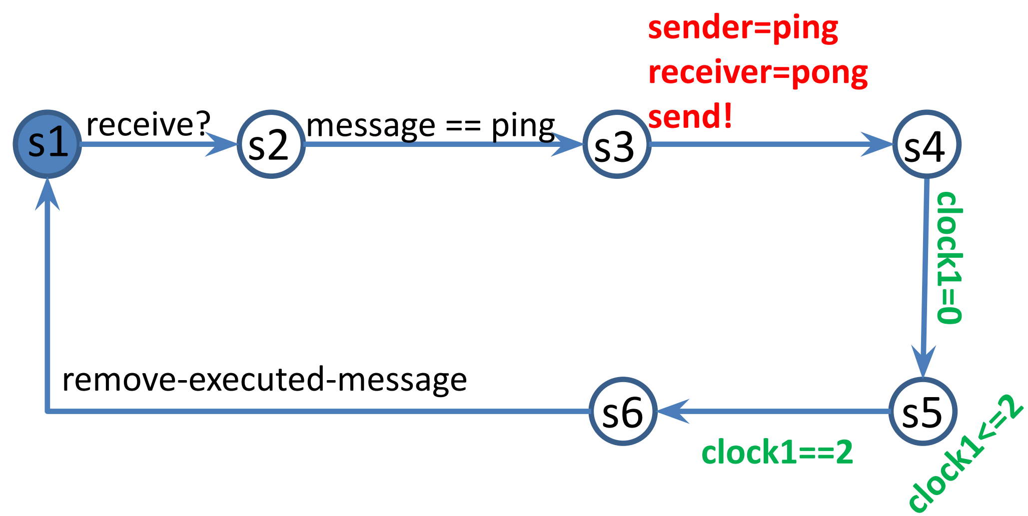
Timed Automata of Timed Rebeca Models

- Three types of automata
  - A timed automaton for modeling the behavior of each rebec
  - A timed automaton for each message bag
  - A timed automaton for simulating the behavior of *after*

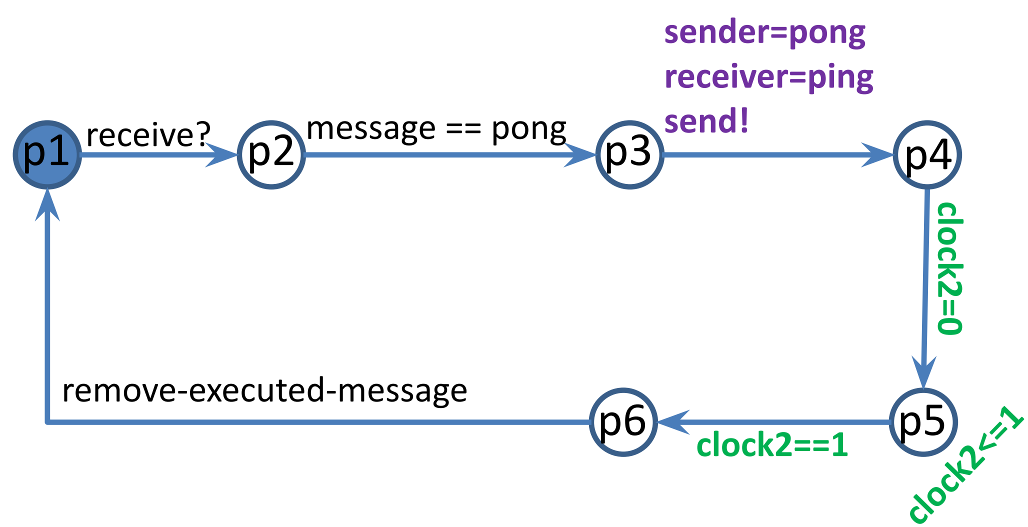
Timed Automata for Ping and Pong

(Model without ***after*** and ***deadline***)

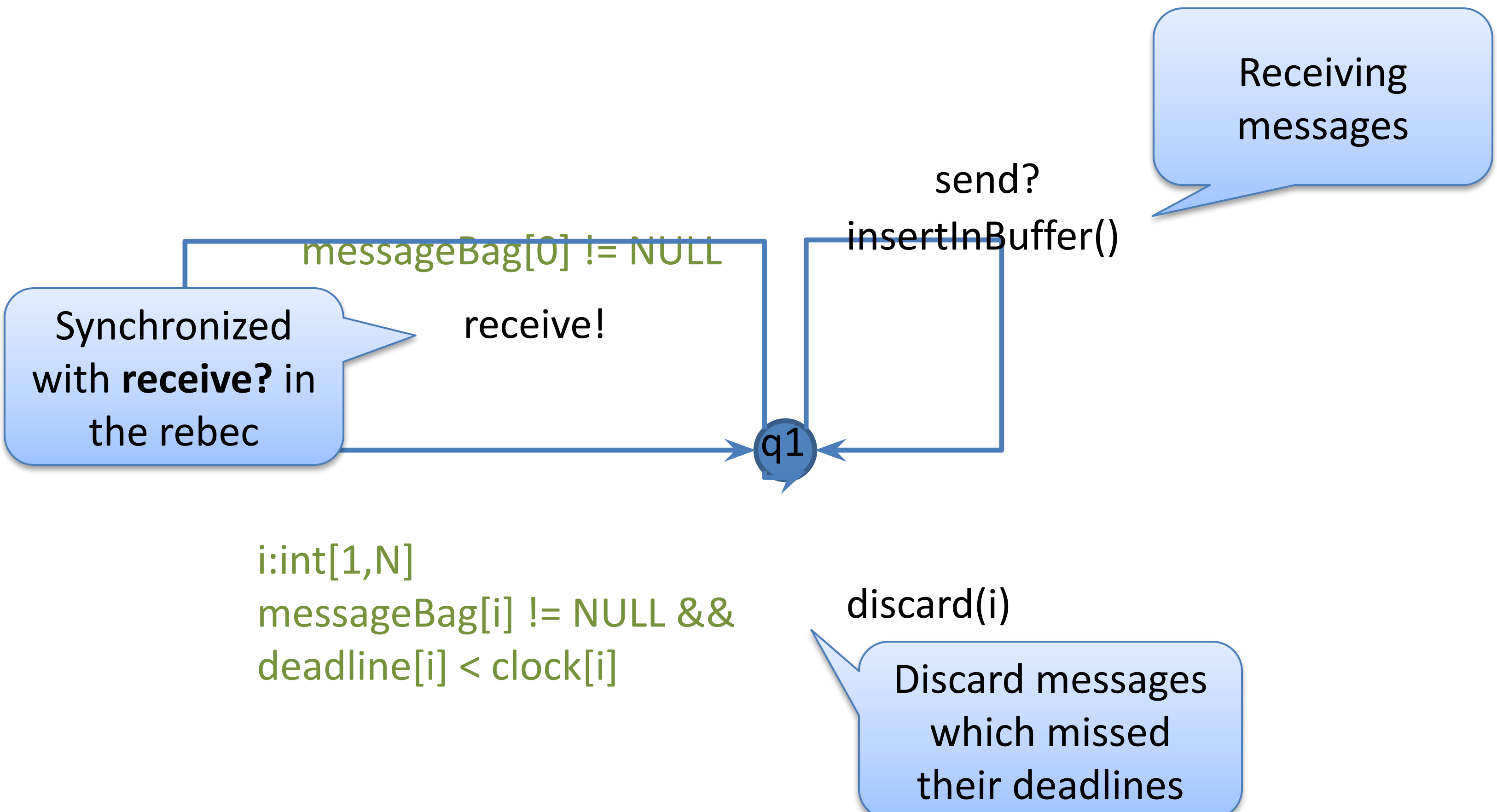
```
reactiveclass Ping(3) {
    knownrebecs {Pong pong;}
    Ping() {self.ping();}
    msgsrv ping() {
      pong.pong();
      delay(2);
    }
}
```



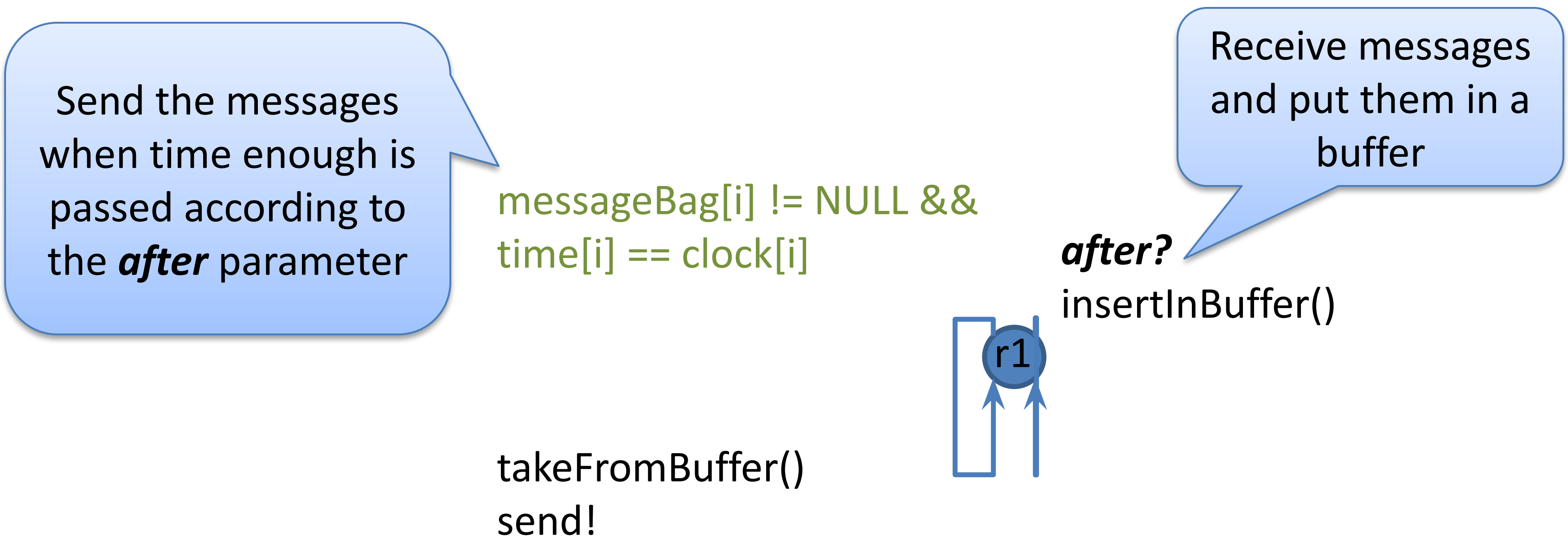

```
reactiveclass Pong(3) {
    knownrebecs {Ping ping;}
    Ping() {}
    msgsrv ping() {
      ping.ping();
      delay(1);
    }
}
```



Timed Automata for Message Buffers

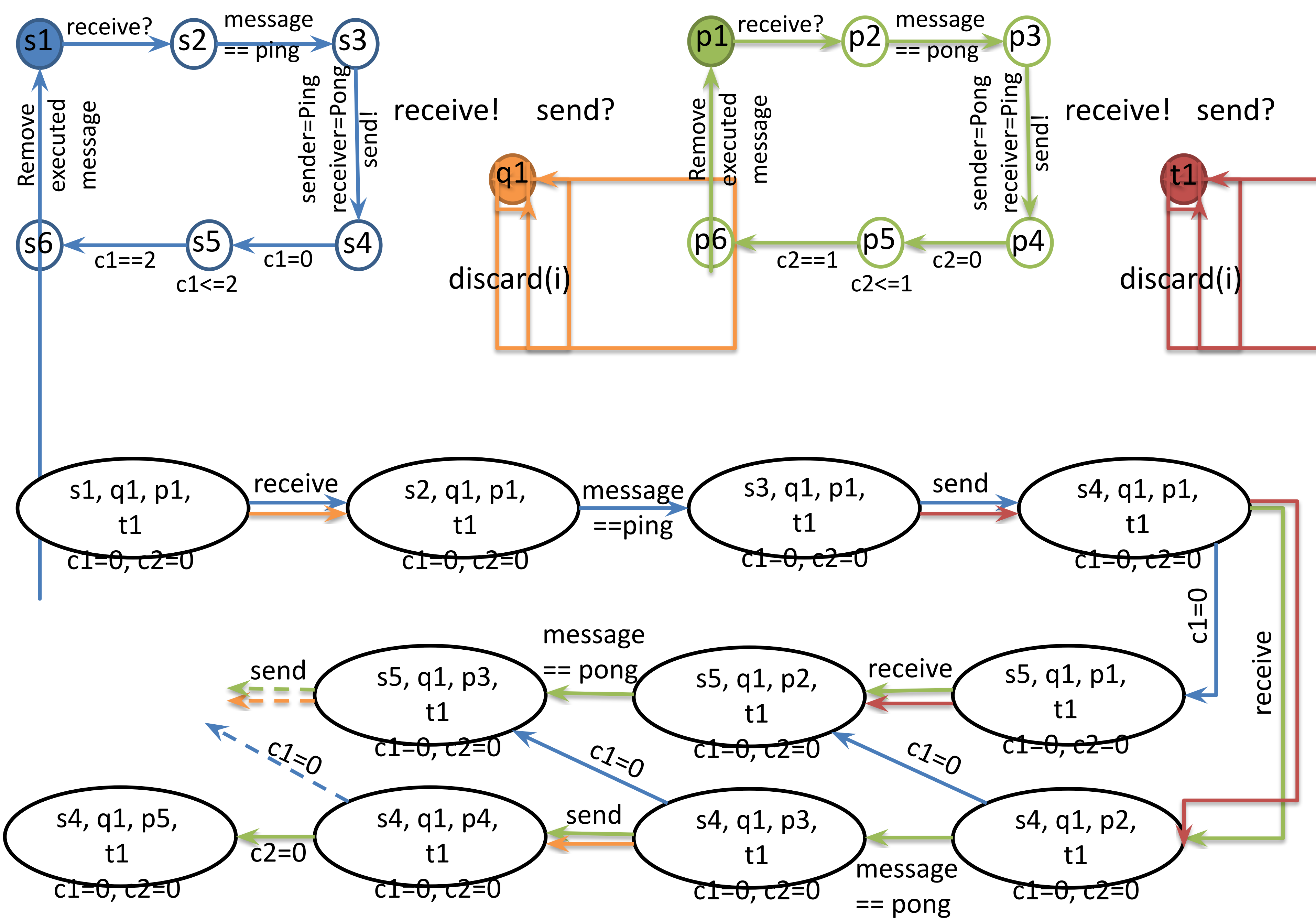


Timed Automata for *After*



Region Transition System of Timed Automata Model

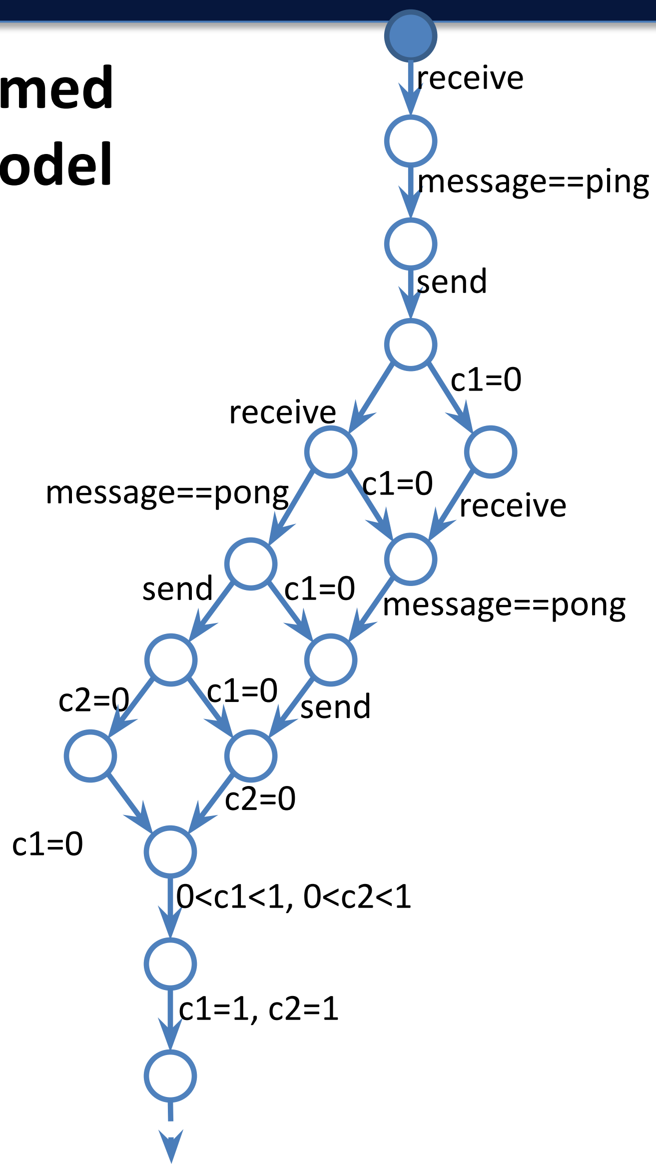
- Labels of states
  - s: Ping actor,
  - p: Pong actor,
  - q: Ping queue,
  - t: Pong queue
  - c1: local clock of Ping actor,
  - c2: local clock of Pong actor



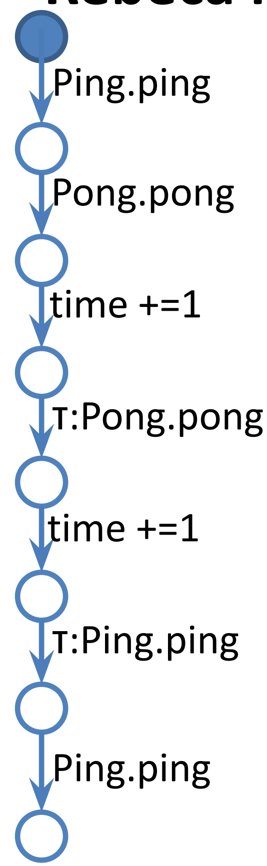
s1
receive?
s2
message
== ping
s3
sender=Ping
receiver=Pong
send!
s4
c1=0
s5
c1<=2
c1==2
s6
Remove
executed
message
receive!
send?
q1
discard(i)
p1
receive?
p2
message
== pong
p3
sender=Pong
receiver=Ping
send!
p4
c2=0
p5
c2<=1
c2==1
p6
Remove
executed
message
receive!
send?
t1
discard(i)
s1, q1, p1, t1 c1=0, c2=0
receive
s2, q1, p1, t1 c1=0, c2=0
message ==ping
s3, q1, p1, t1 c1=0, c2=0
send
s4, q1, p1, t1 c1=0, c2=0
c1=0
receive
s5, q1, p1, t1 c1=0, c2=0
receive
s5, q1, p2, t1 c1=0, c2=0
message == pong
s5, q1, p3, t1 c1=0, c2=0
send
c1=0
c1=0
c1=0
s4, q1, p2, t1 c1=0, c2=0
message == pong
s4, q1, p3, t1 c1=0, c2=0
send
s4, q1, p4, t1 c1=0, c2=0
c2=0
s4, q1, p5, t1 c1=0, c2=0

Region Transition System of Timed Automata Model (Model without ***after*** and ***deadline***)

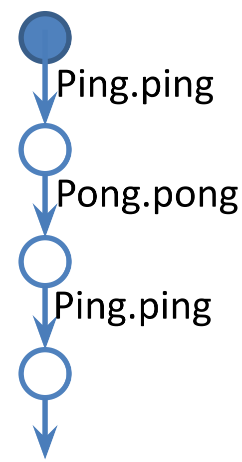
RTS of the Timed Automata model



TTS of the Timed Rebeca model



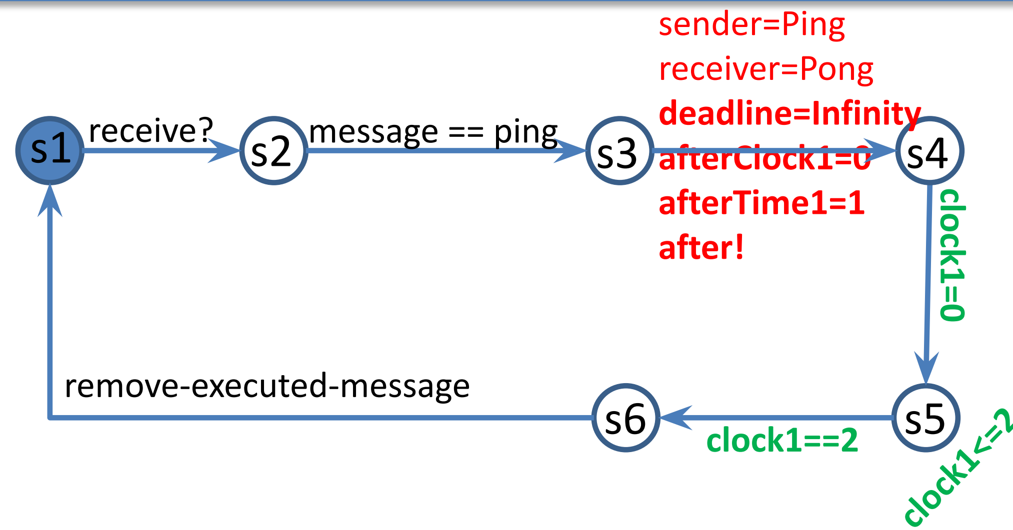
FTTS of the Timed Rebeca model



Timed Automata for Ping-Pong

(Model with ***after*** and ***deadline***)

```
reactiveclass Ping(3) {
    knownrebecs {Pong pong;}
    Ping() {self.ping();}
    msgsrv ping() {
      pong.pong() after(1);
      delay(2);
    }
}
```




```
reactiveclass Pong(3) {
    knownrebecs {Ping ping;}
    Ping() {}
    msgsrv ping() {
      ping.ping() after (1) deadline(2);
      delay(1);
    }
}
```