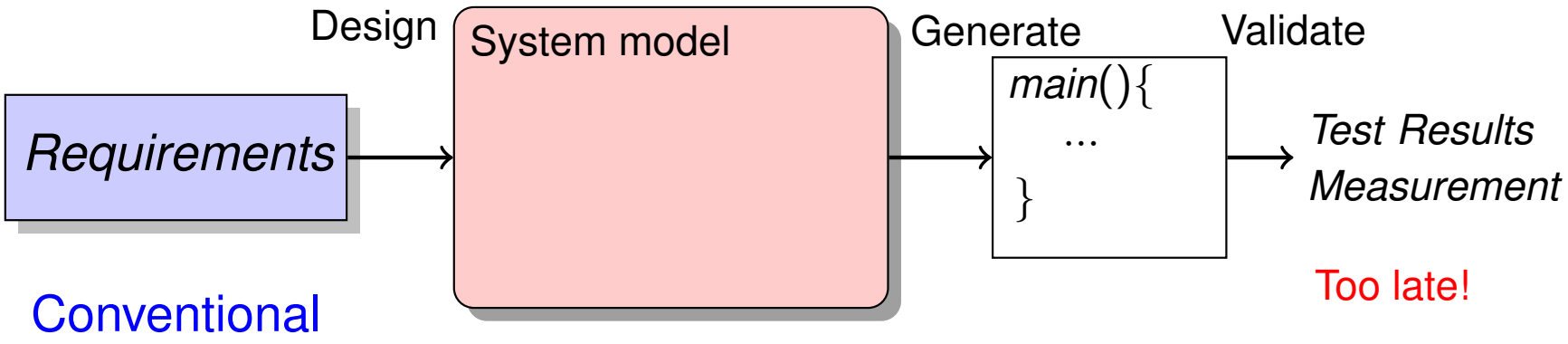
Analysis and Implementation of Embedded System Models: Example of Tags in Item Management Application

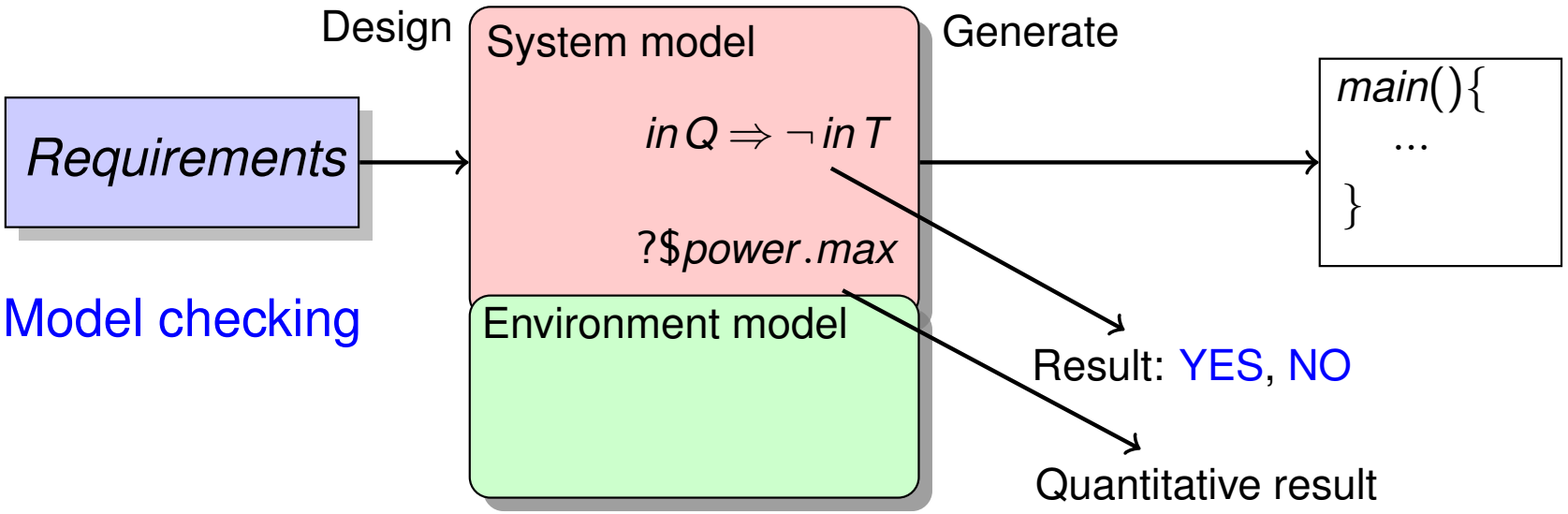
Bojan Nokovic and Emil Sekerinski

Department of Computing and Software
McMaster University

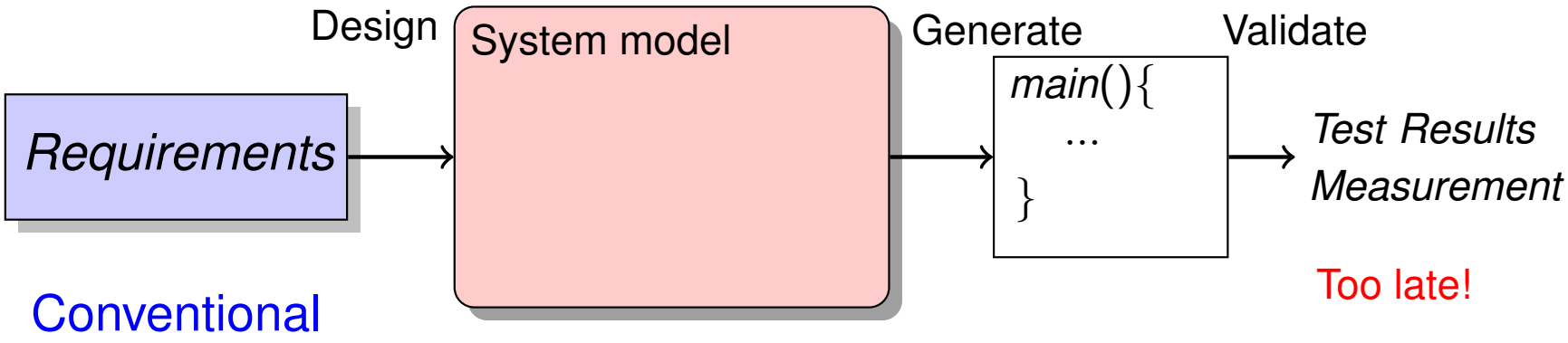
March 13, 2015

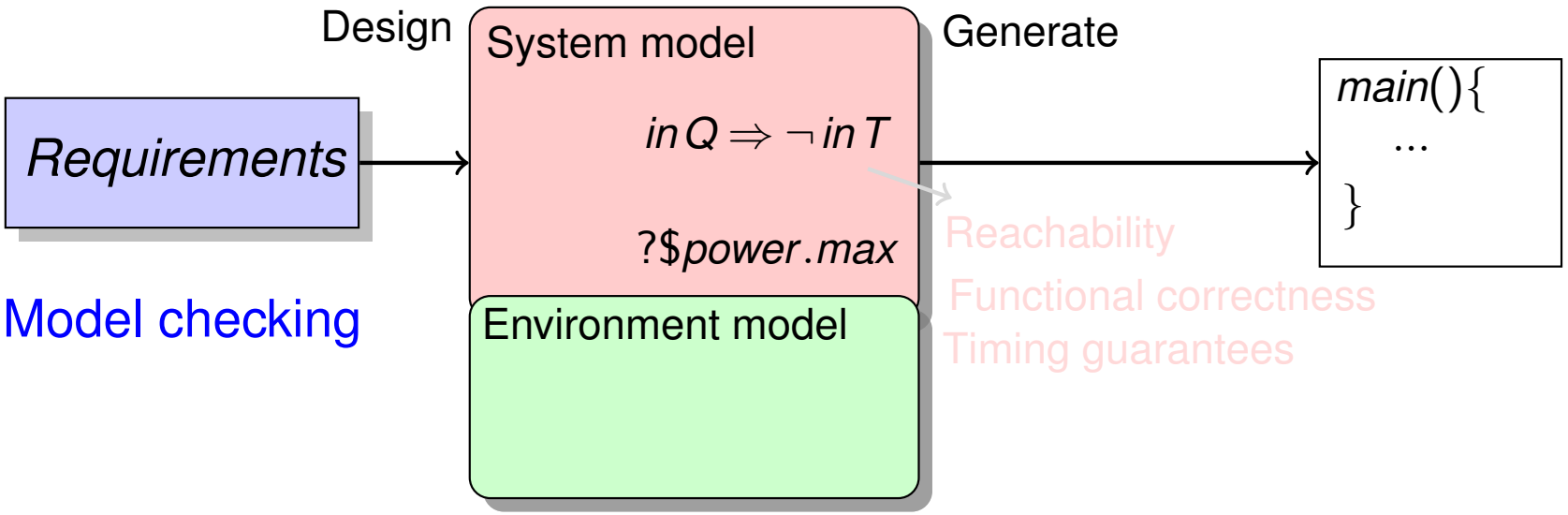
Design Process



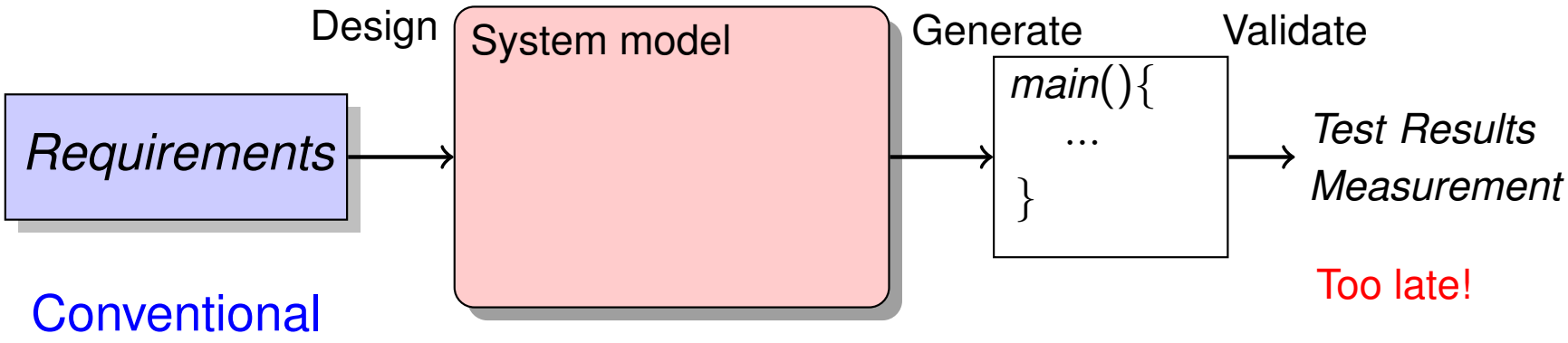


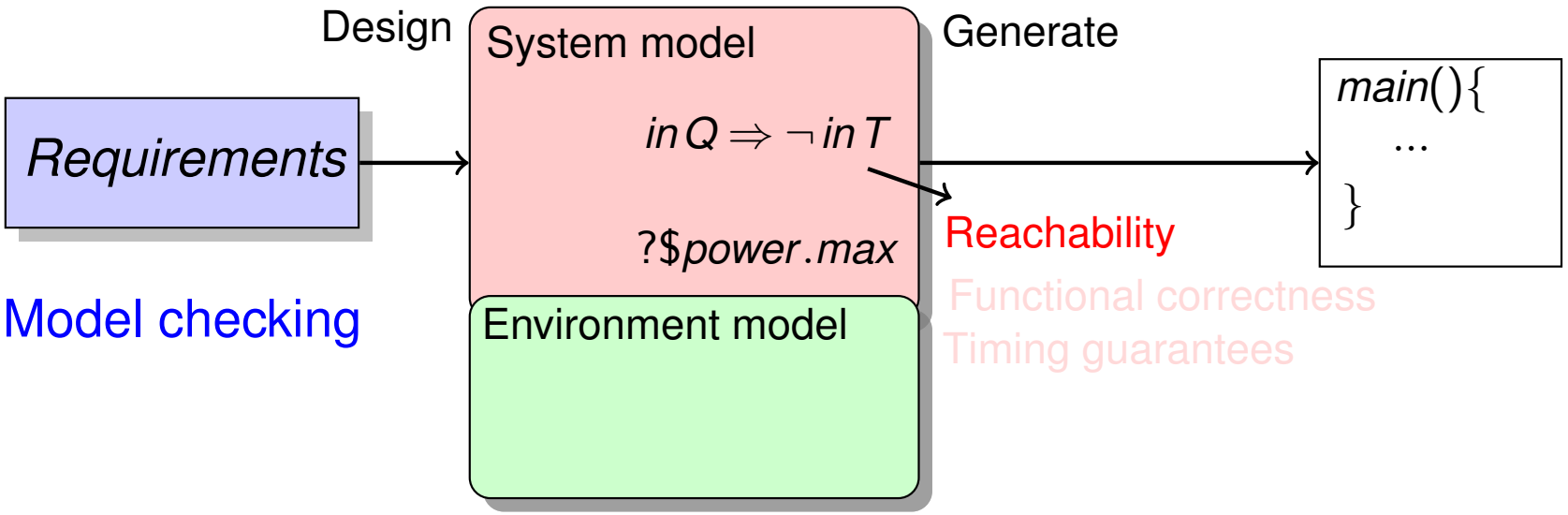
Design Process



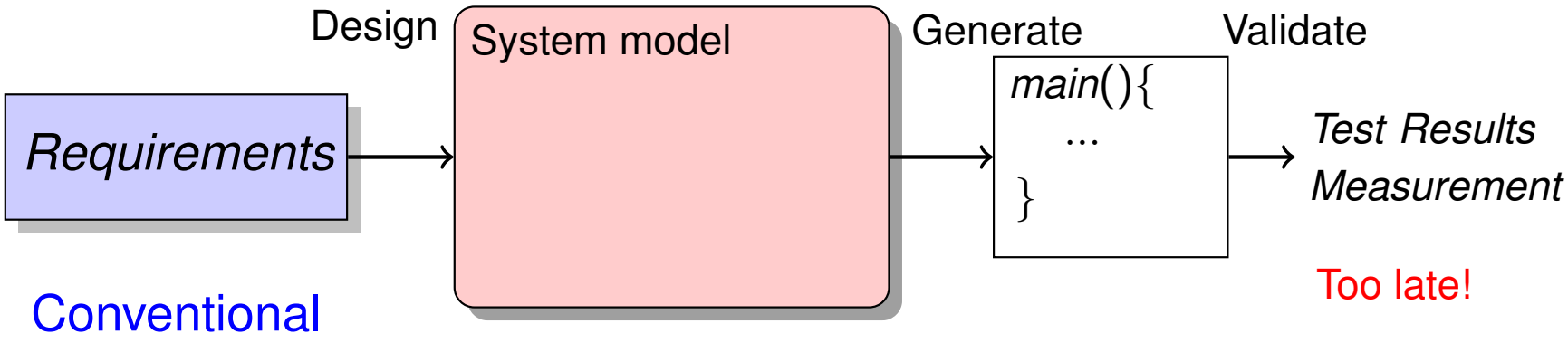


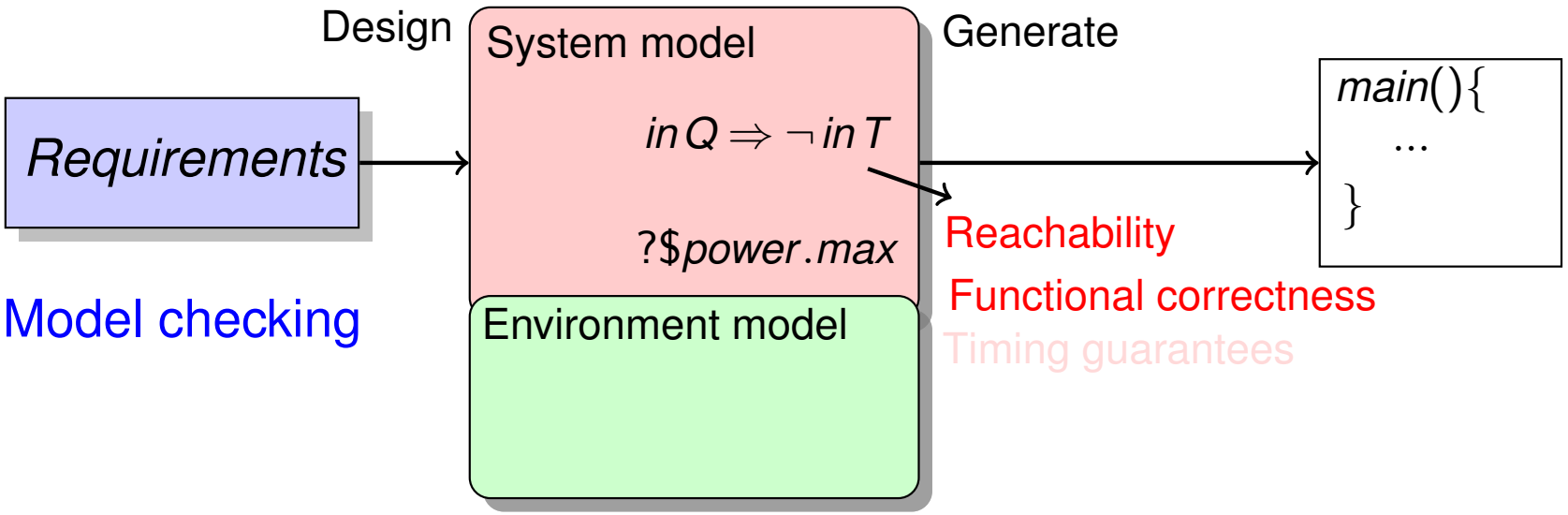
Design Process



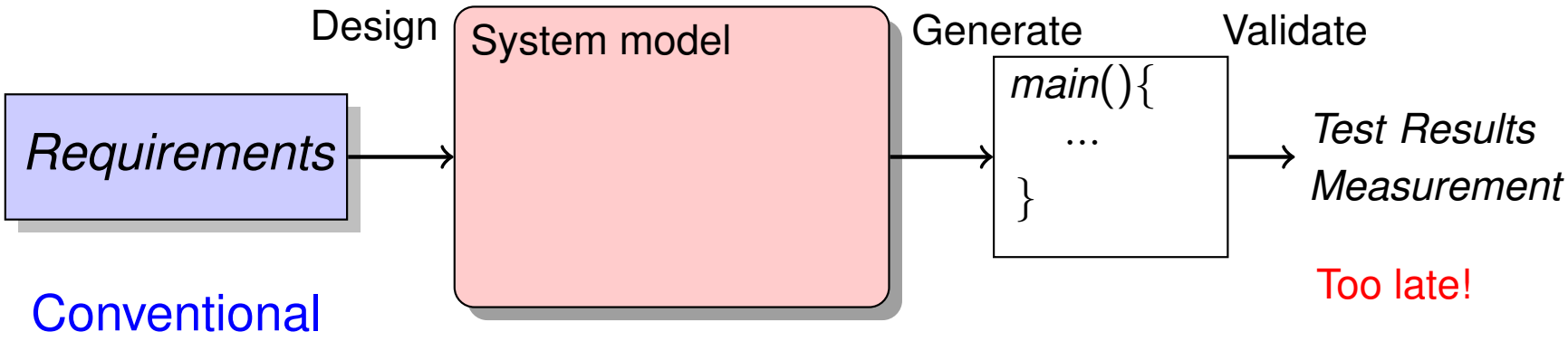


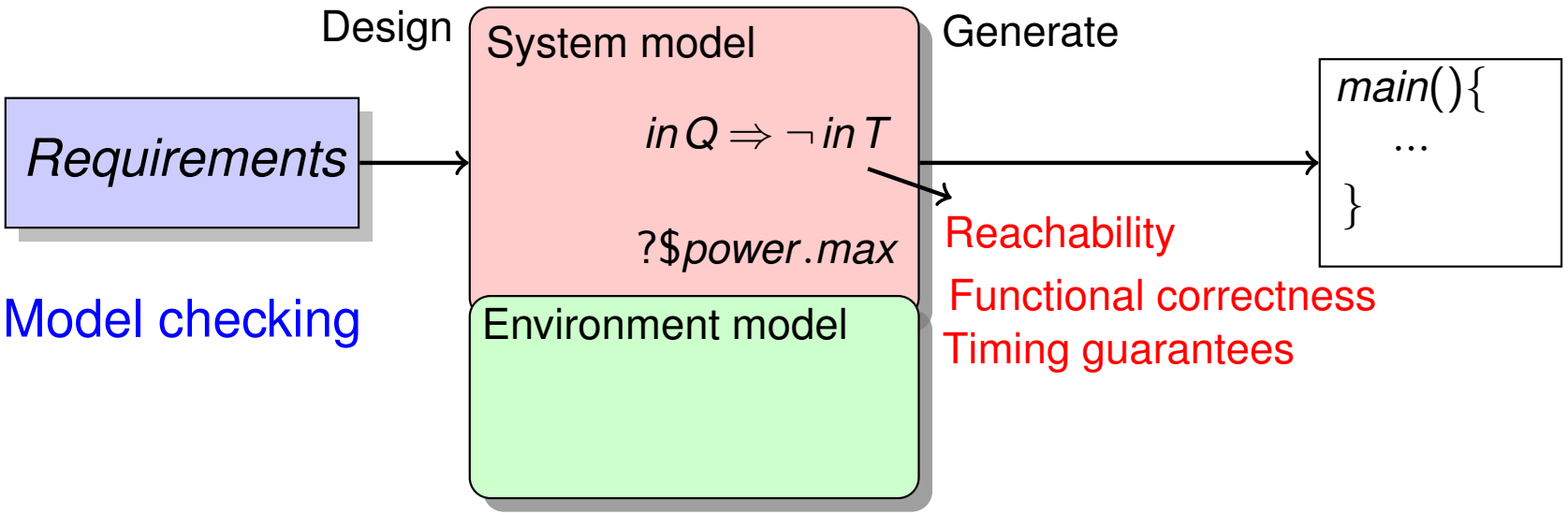
Design Process



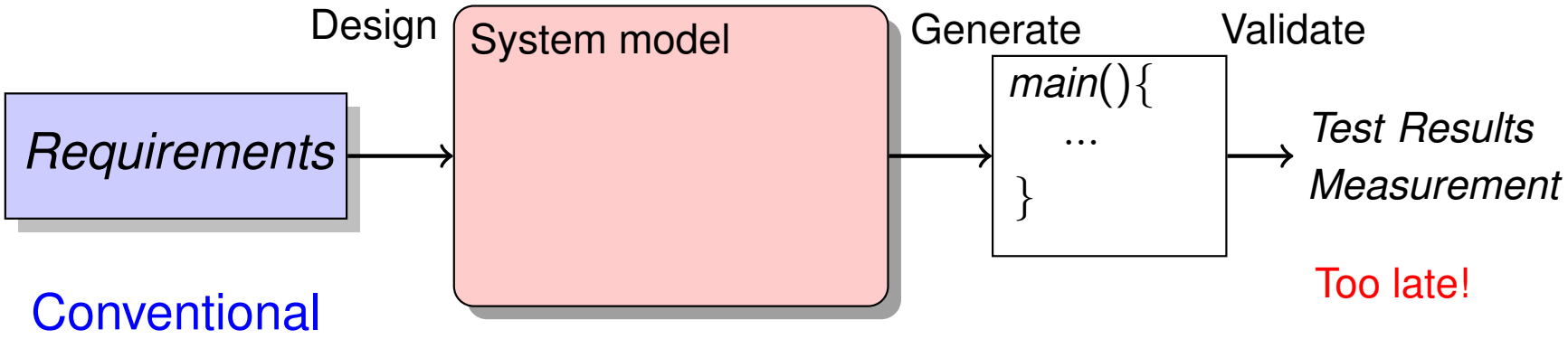


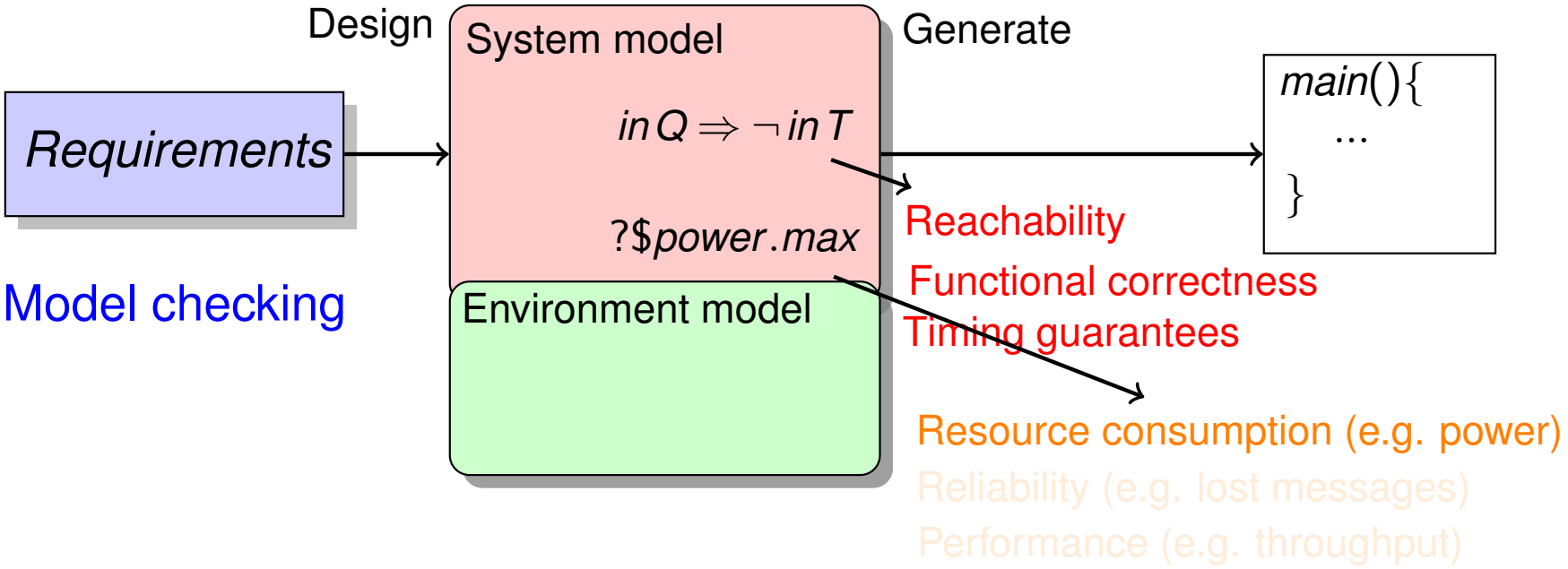
Design Process



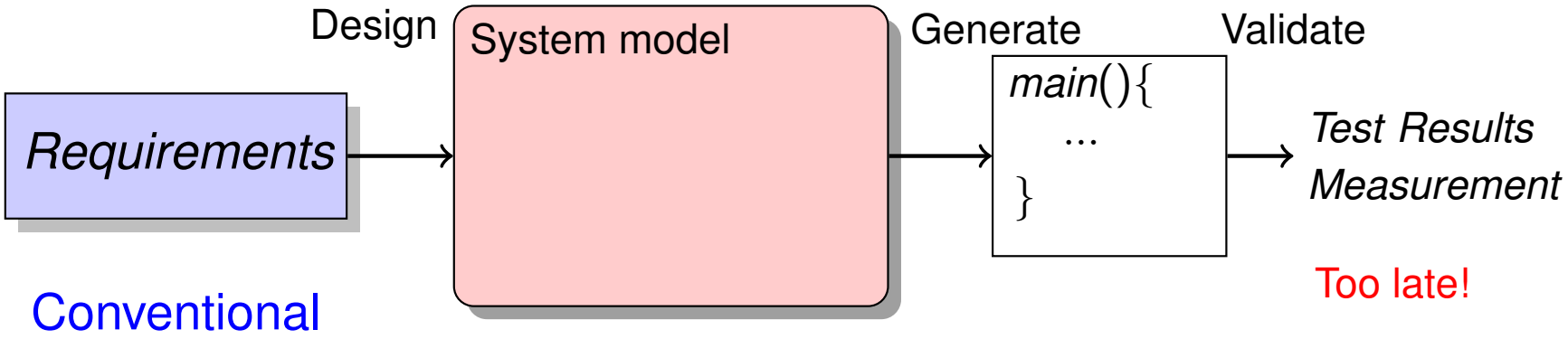


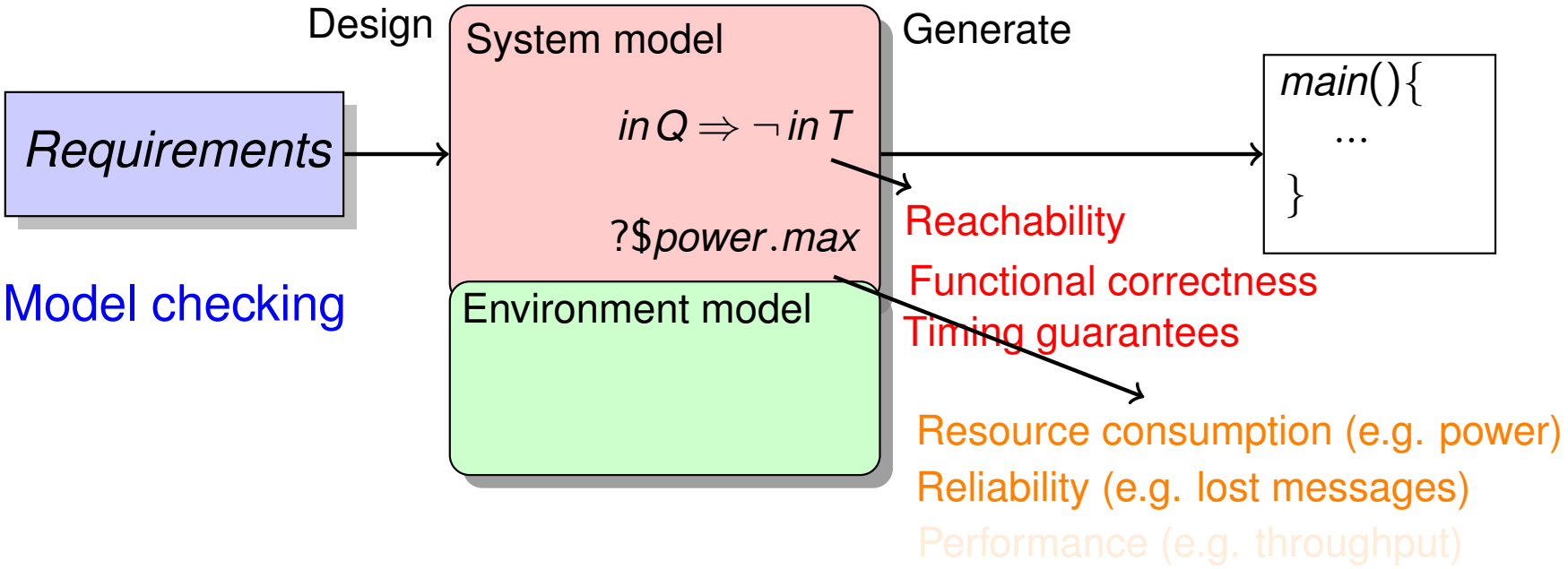
Design Process



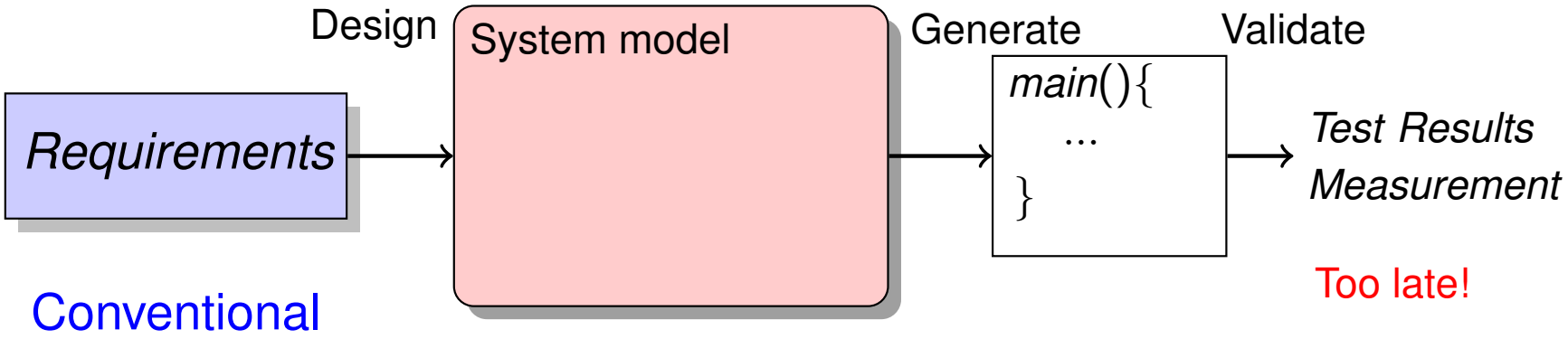


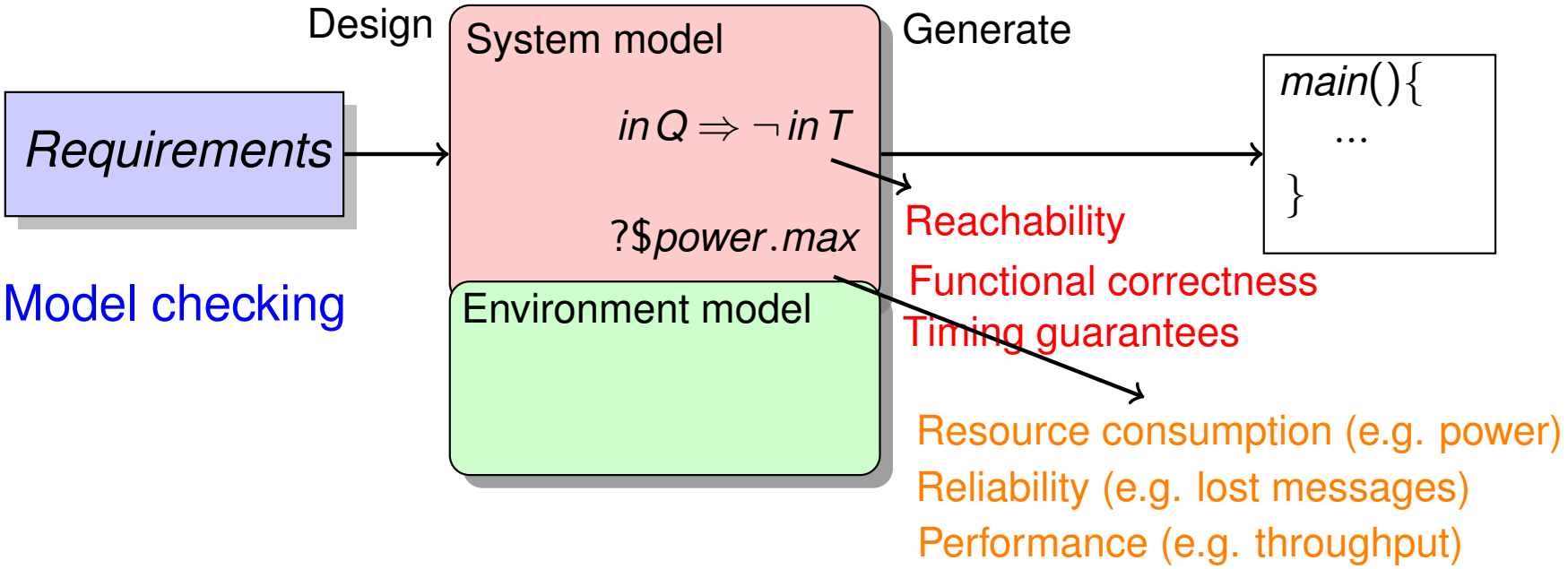
Design Process



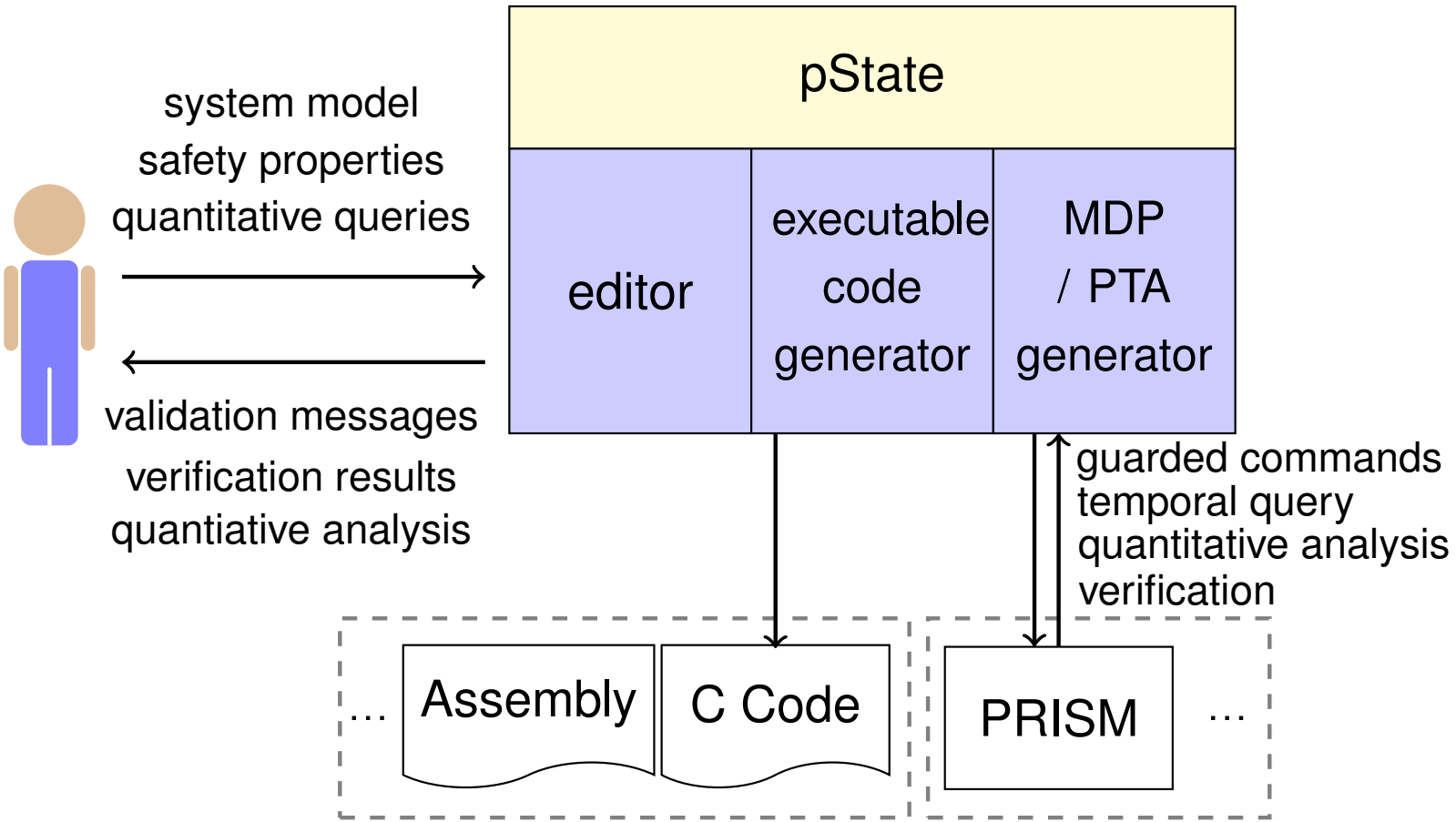


Design Process





pState Architecture



[PRISM - probabilistic model checker developed at University of Birmingham and University of Oxford]

Hierarchical State Machines

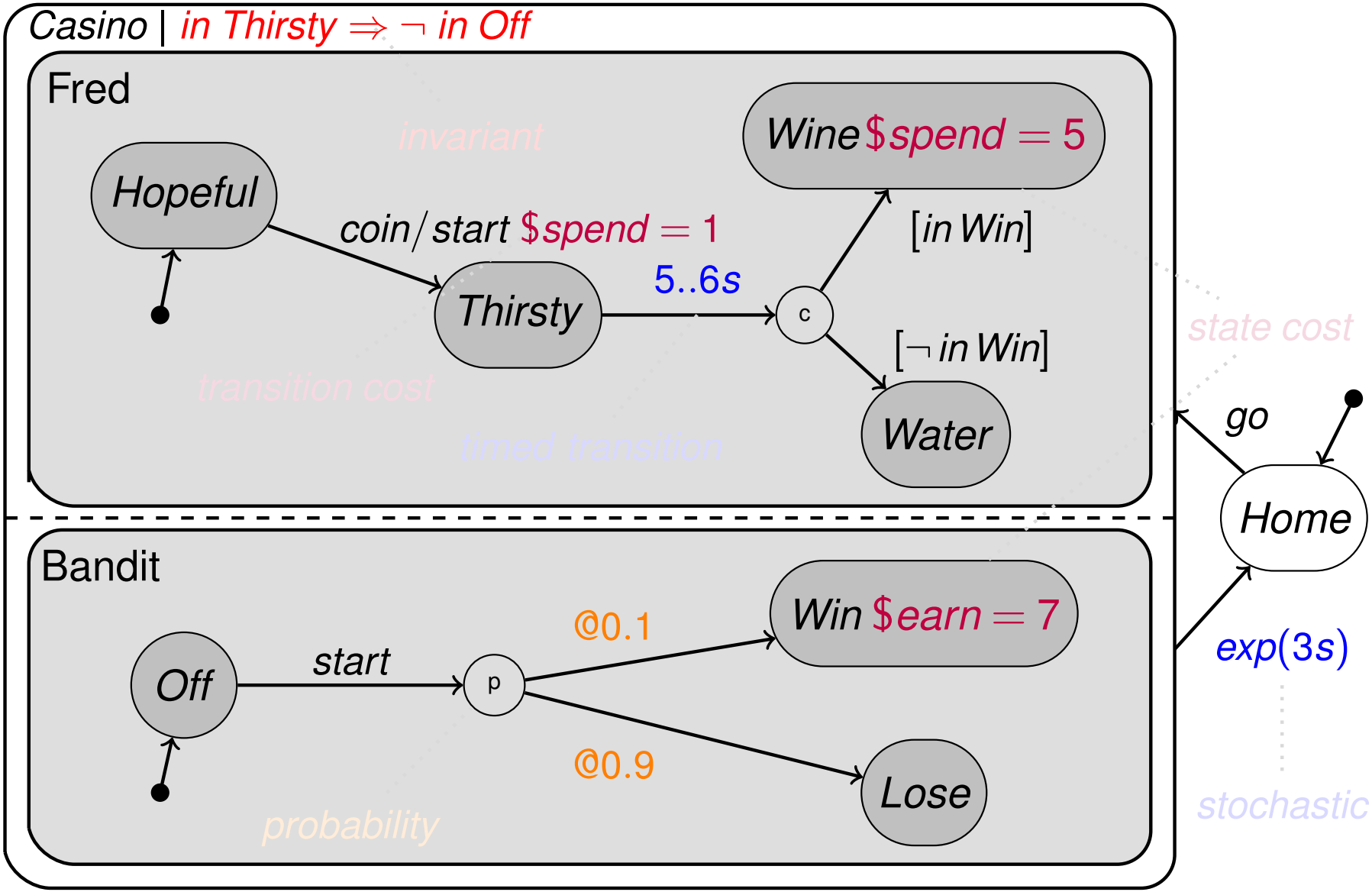
A graphical language designed to describe behaviour of discrete-state reactive systems

Statecharts = State Diagram + Hierarchy + Concurrency + Broadcast [Harel 87]

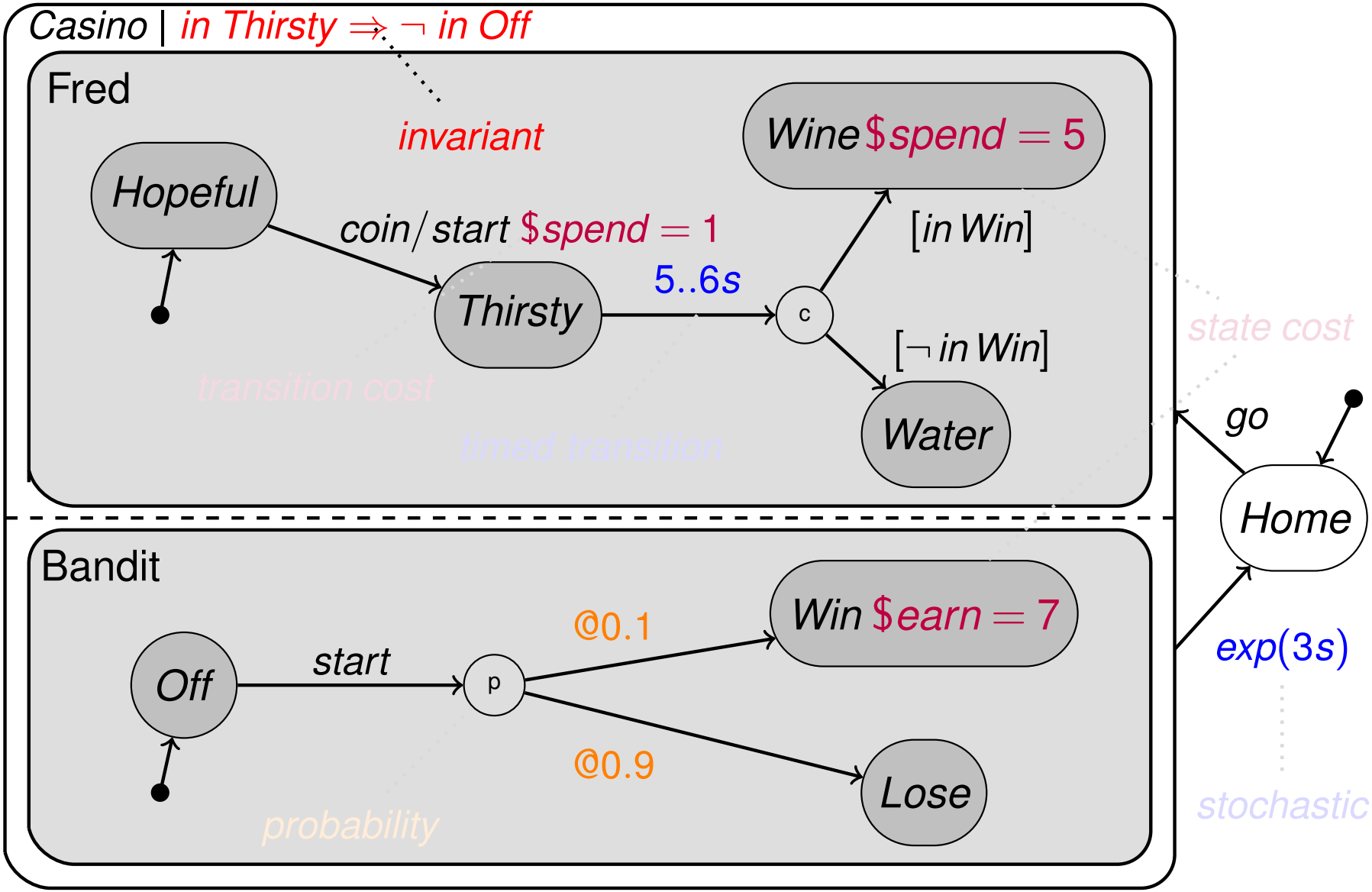
Invariantcharts = Statecharts + State Invariants [Back 06, Sek. 07]

pCharts = Invariantcharts + Costs + Probability + Timed Transitions [Nok., Sek. 14]

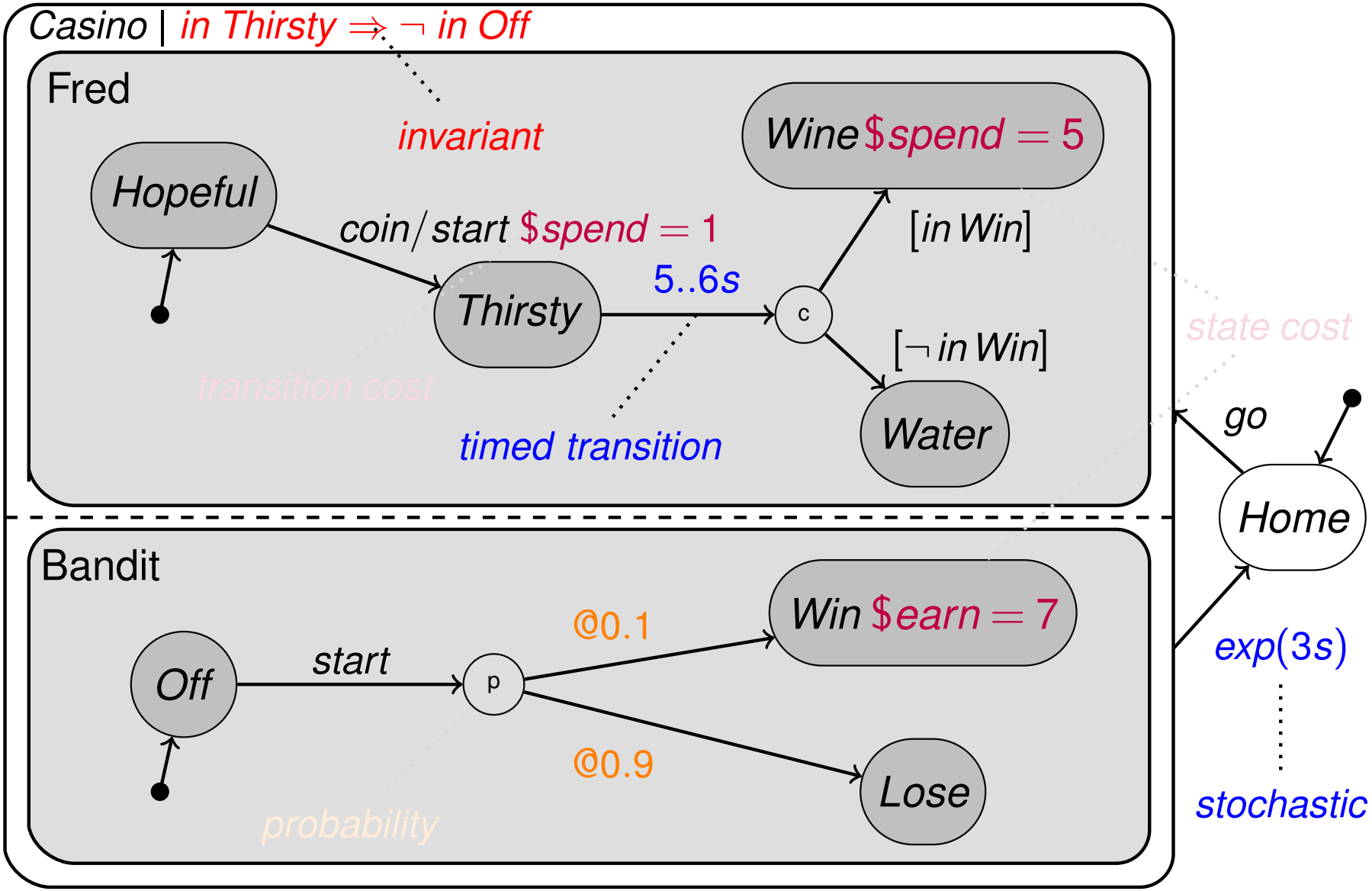
pCharts


Casino | in Thirsty ⇒ ¬ in Off
Fred
invariant
Hopeful
Wine $spend = 5
coin / start $spend = 1
[in Win]
5..6s
Thirsty
c
[¬ in Win]
state cost
transition cost
Water
timed transition
go
Home
Bandit
Win $earn = 7
@0.1
Off
start
p
exp(3s)
@0.9
Lose
stochastic
probability

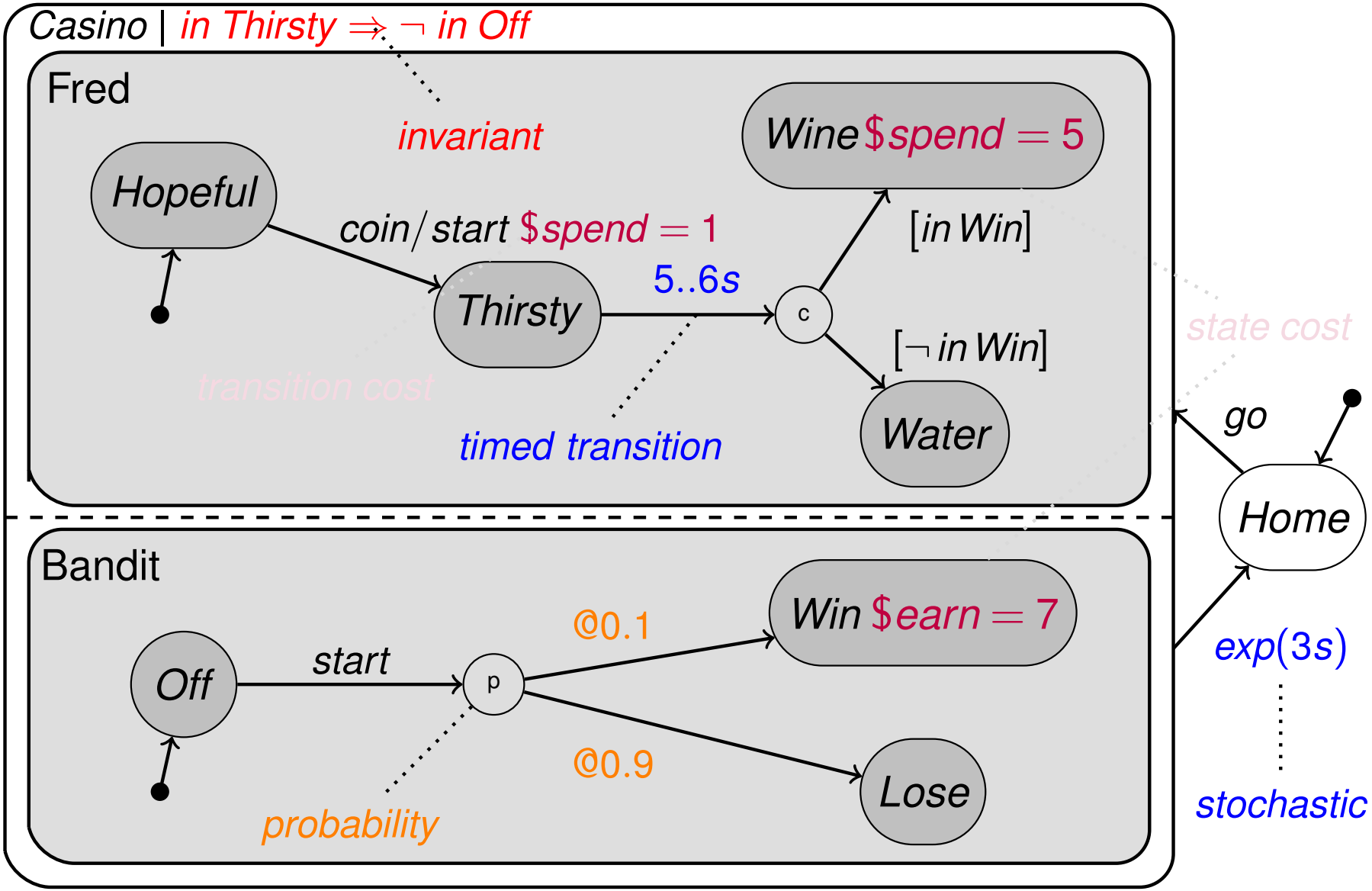
pCharts


Casino | in Thirsty ⇒ ¬ in Off
Fred
invariant
Hopeful
Wine $spend = 5
coin / start $spend = 1
5..6s
[in Win]
Thirsty
c
[¬ in Win]
Water
transition cost
timed transition
state cost
go
Home
Bandit
Off
start
p
@0.1
Win $earn = 7
@0.9
Lose
probability
exp(3s)
stochastic

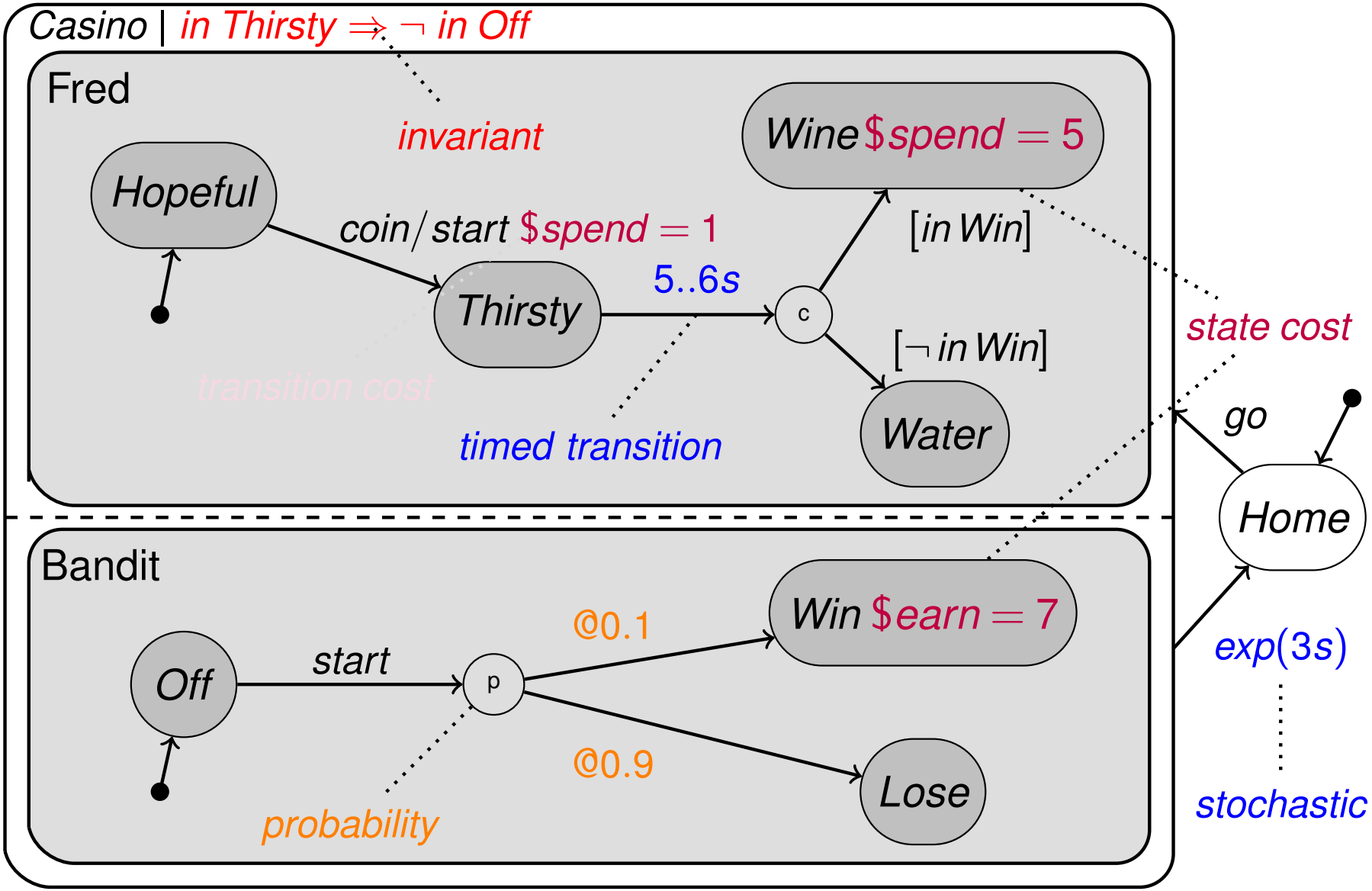
pCharts


Casino | in Thirsty ⇒ ¬ in Off
Fred
invariant
Hopeful
Wine $spend = 5
coin/start $spend = 1
[in Win]
5..6s
Thirsty
c
[¬ in Win]
transition cost
timed transition
Water
state cost
go
Home
Bandit
Win $earn = 7
@0.1
start
Off
p
exp(3s)
@0.9
Lose
probability
stochastic

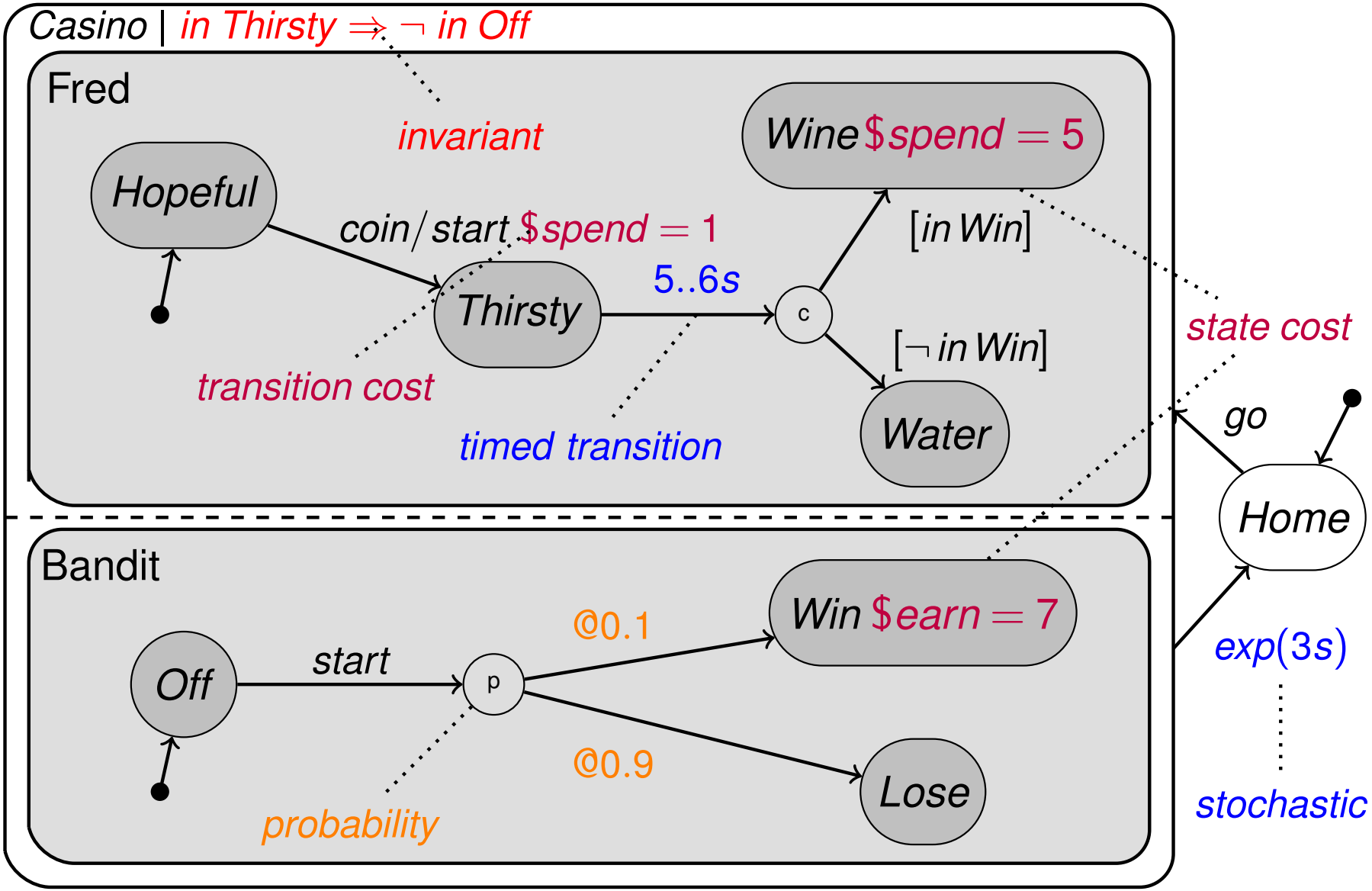
pCharts


Casino | in Thirsty ⇒ ¬ in Off
Fred
invariant
Hopeful
Wine $spend = 5
coin / start $spend = 1
[in Win]
5..6s
Thirsty
c
[¬ in Win]
state cost
transition cost
timed transition
Water
go
Home
Bandit
Win $earn = 7
@0.1
Off
start
p
exp(3s)
@0.9
Lose
probability
stochastic

pCharts


Casino | in Thirsty ⇒ ¬ in Off
Fred
invariant
Hopeful
Wine $spend = 5
coin / start $spend = 1
[in Win]
5..6s
Thirsty
c
state cost
[¬ in Win]
transition cost
timed transition
Water
go
Home
Bandit
Win $earn = 7
@0.1
Off
start
p
exp(3s)
@0.9
Lose
probability
stochastic

pCharts

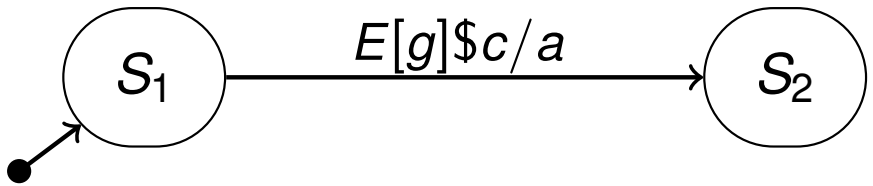

Casino | in Thirsty ⇒ ¬ in Off
Fred
invariant
Hopeful
Wine $spend = 5
coin / start $spend = 1
[in Win]
5..6s
Thirsty
c
state cost
[¬ in Win]
transition cost
timed transition
Water
go
Home
Bandit
Win $earn = 7
@0.1
Off
start
p
exp(3s)
@0.9
Lose
probability
stochastic

Operations

The *operation* of an event E is a statement that captures the joint meaning of all transitions in a chart on E

pState semantic is characterized as *event-centric* where the main structure of the code is that of events

Translation schemes of other tools like UML, Statemate, ESTEREL Scade Suite can be characterized as *state-centric* in which events are data in queues



$$op(E) = (in(S_1) \wedge g \rightarrow goto(S_2)) \parallel a\,\$c$$

Event Centric vs. State Centric

These interpretations are called *requirements-oriented* and *implementation-oriented* semantics [Eshuis et al. 02]

Event-centric interpretation has requirements-oriented semantics

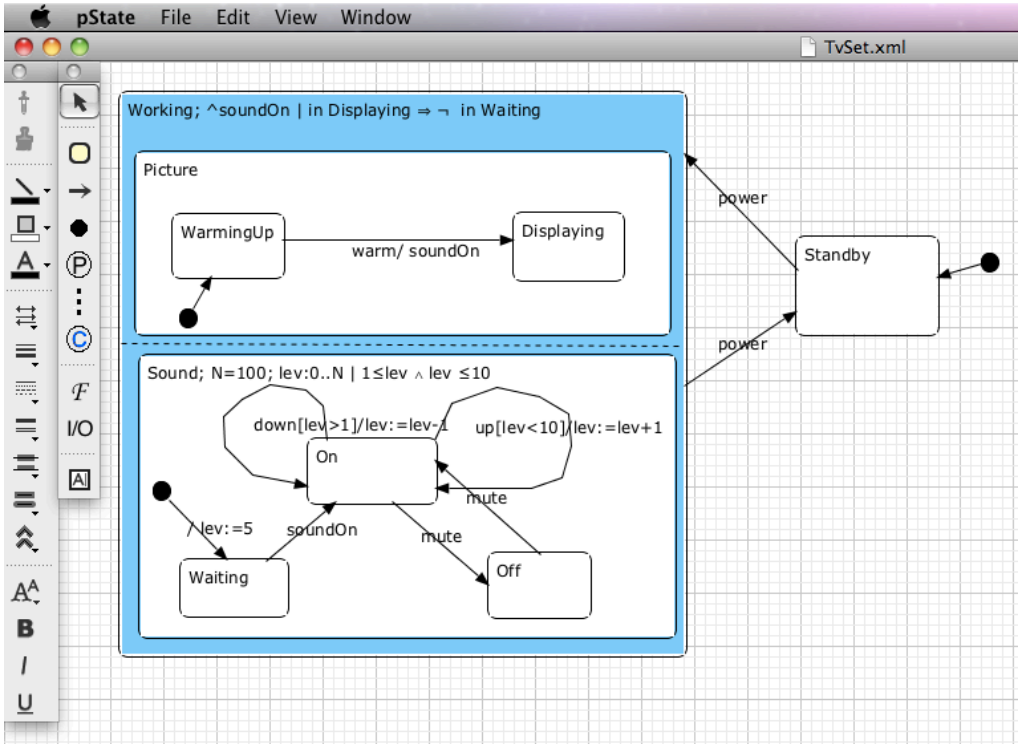
A number of quantitative extensions of statecharts similar to pCharts have been proposed[Liu et al. 00, Jansen 03, Leitner et al. 11]

All follow the state-centric interpretation in which the state of a chart is given by the configuration, a set of events, and the valuation of the variables

In pCharts, the state consists only of the configuration and the valuation of the variables, thus reducing the state space and facilitating model-checking

Implementation - pState

Interface created using JHotDraw 7.6 graphics framework



Visitor Pattern

Transform to intermediate code

Translate to (1) executable code (c, asm)

(2) input code for PRISM probabilistic model checker

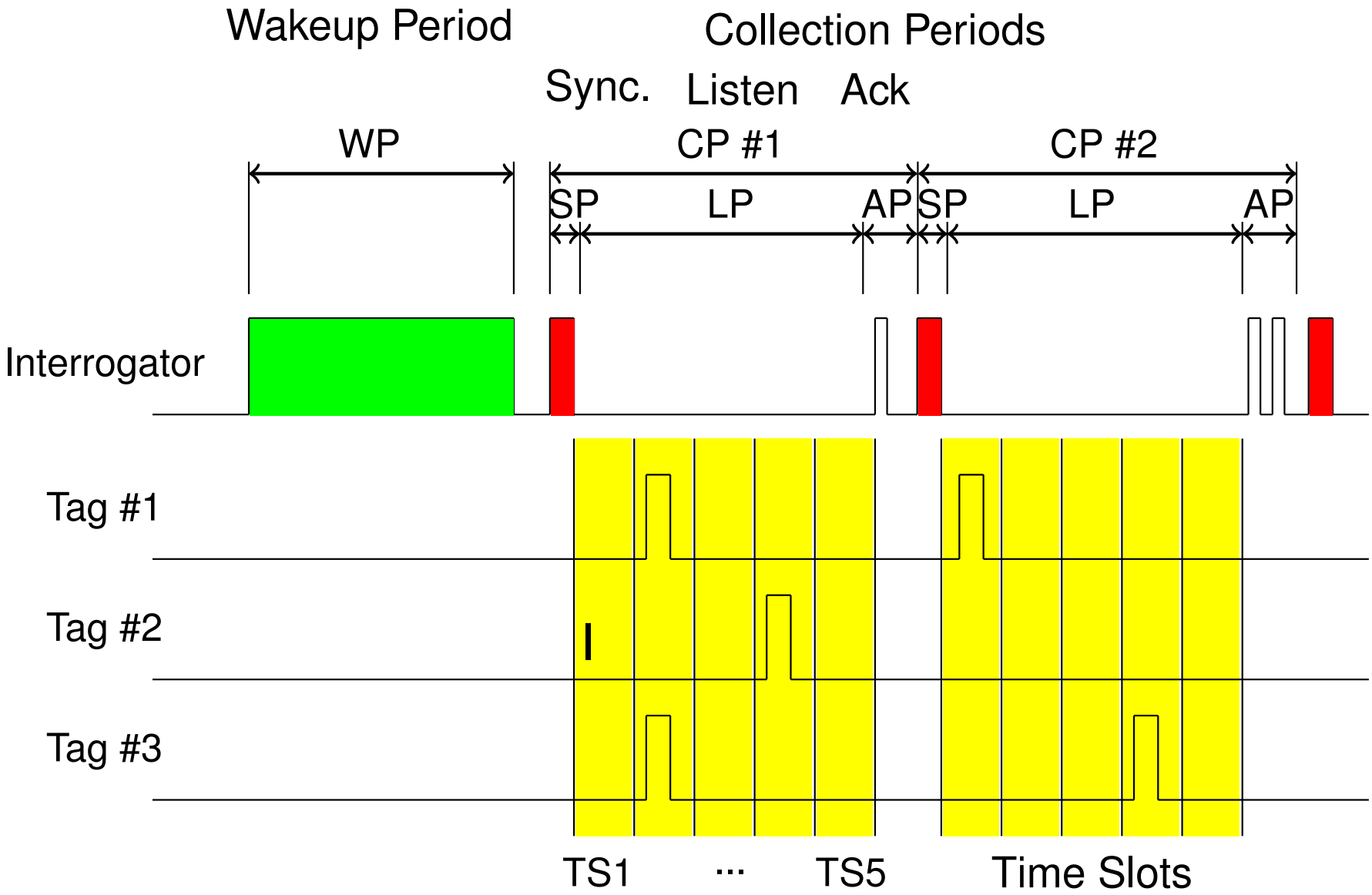
RFID Asset Management

Hand held interrogators

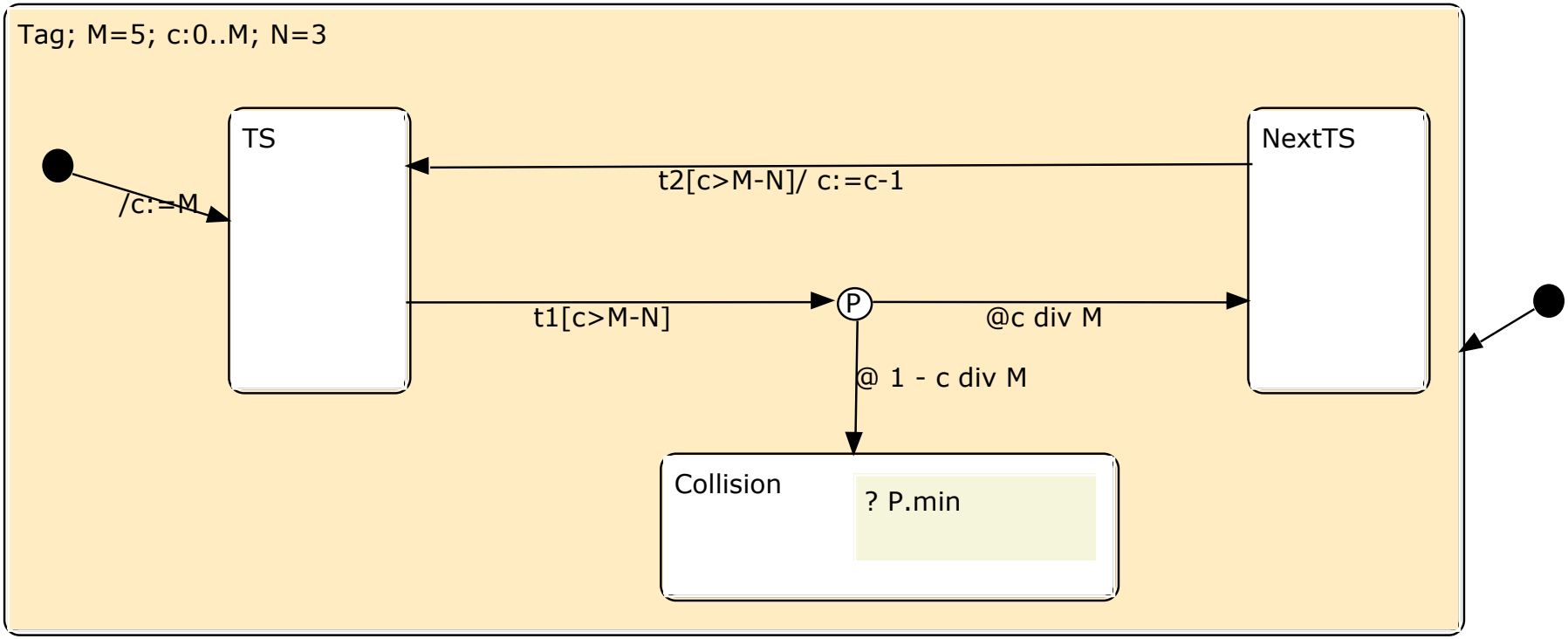
RFID tag attached on the asset (box)

No communication between tags

Contention Resolution in DASH-7 ISO/IEC 18000-7.2



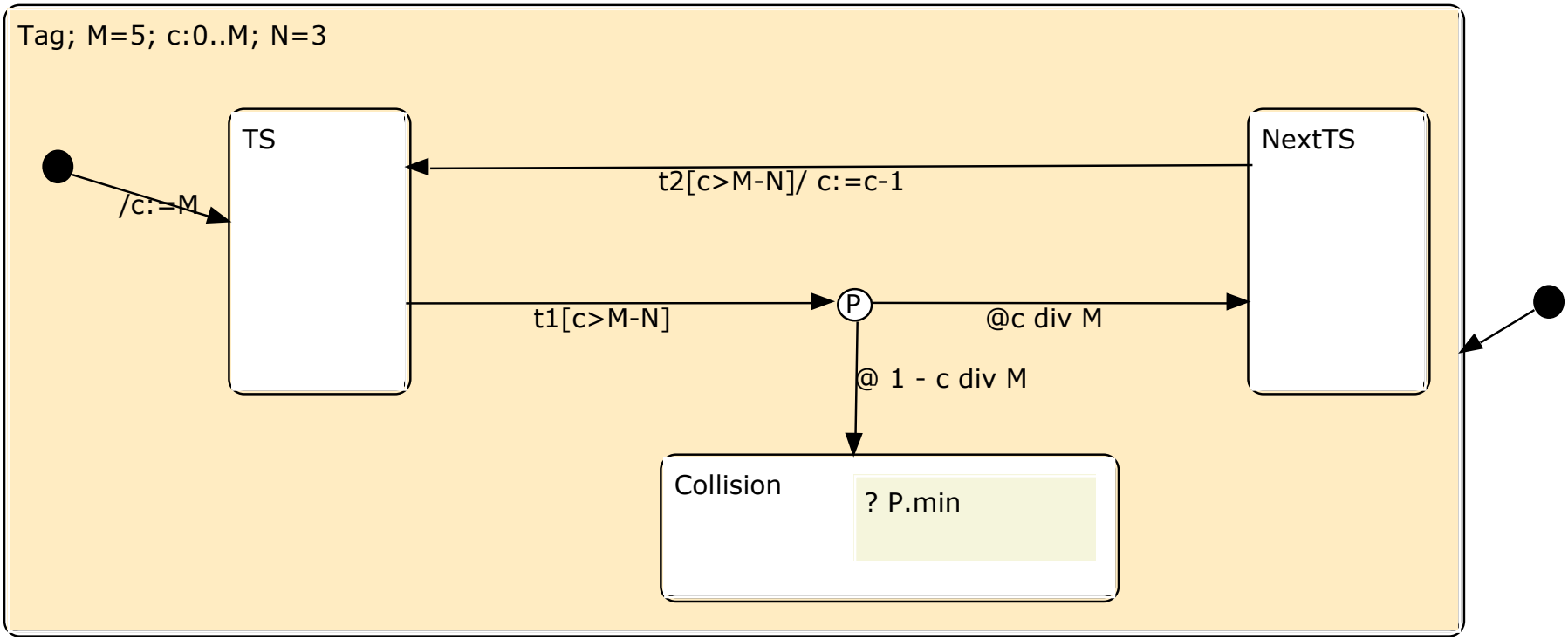
Collision Model, Three Tags, Five Time Slots



[t1] (tag=TS)&(c>(M−N)) −> (c/M):(tag'=NextTS) + (1−(c/M)):(tag'=Collision);
[t2] (tag=NextTS)&(c>(M−N)) −> (c'=(c−1))&(tag'=TS);

$?P.min$ is translated to PCTL formula: $Pmin = ?[F(tag = Collision)]$

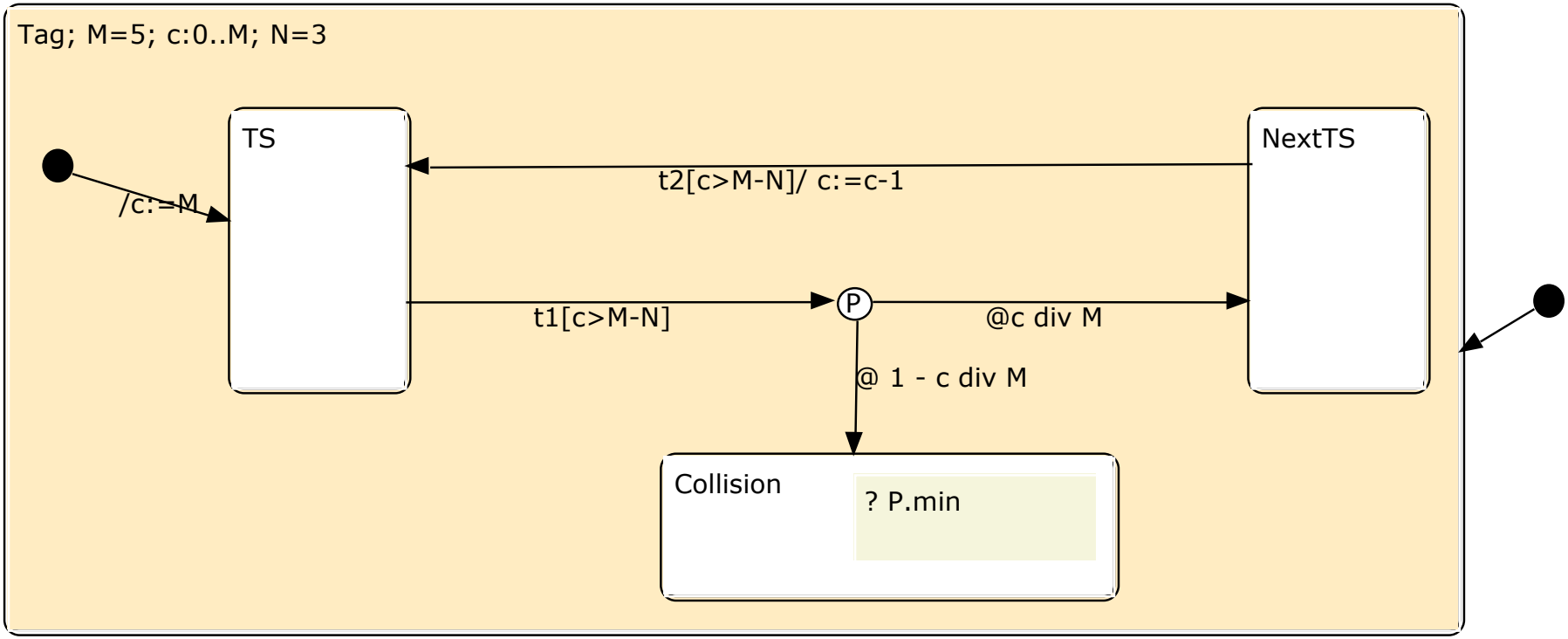
Collision Model, Three Tags, Five Time Slots



[t1] (tag=TS)&(c>(M−N)) −> (c/M):(tag'=NextTS) + (1−(c/M)):(tag'=Collision);
[t2] (tag=NextTS)&(c>(M−N)) −> (c'=(c−1))&(tag'=TS);

$?P.min$ is translated to PCTL formula: $Pmin = ?[F(tag = Collision)]$

Collision Model, Three Tags, Five Time Slots



[t1] (tag=TS)&(c>(M−N)) −> (c/M):(tag'=NextTS) + (1−(c/M)):(tag'=Collision);
[t2] (tag=NextTS)&(c>(M−N)) −> (c'=(c−1))&(tag'=TS);

?*P.min* is translated to PCTL formula: $Pmin =?[F(tag = Collision)]$

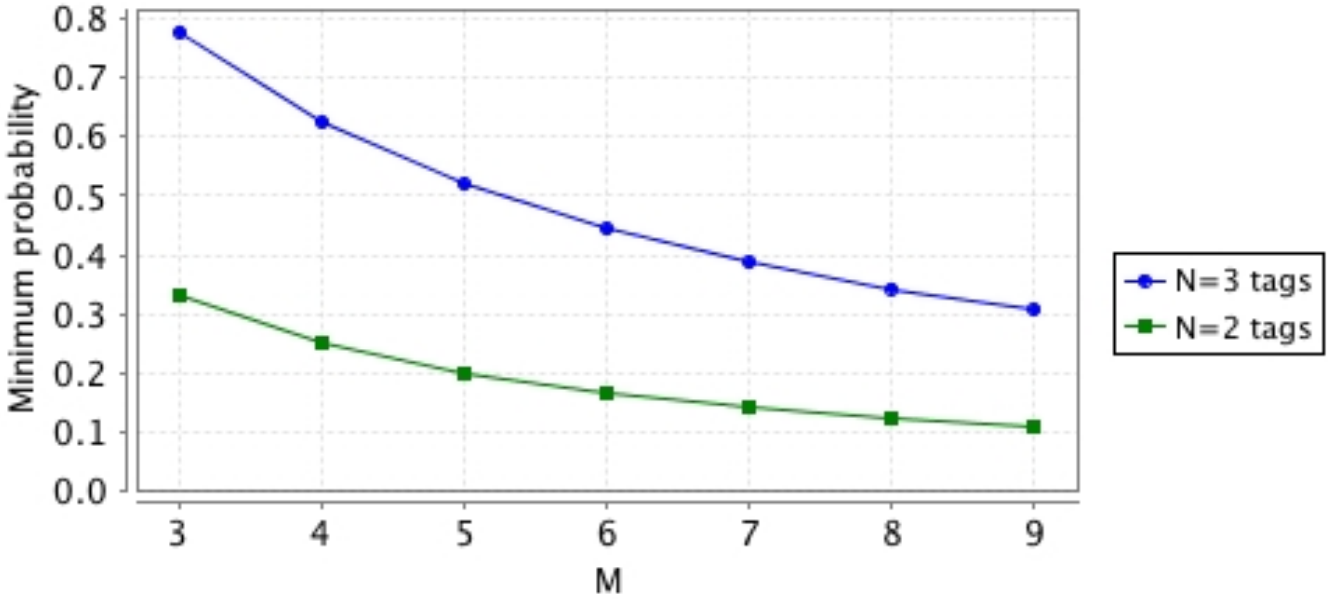
Generated PRISM Code

```
mdp  // Markov Decision Process

const M=5; const N=3; // Constants
const NextTS=0; const Collision=1; const TS=2;

module collision
    tag :[0..2]  init  TS;    // Variables
    c :[0.. M] init  M;

    [t2]  (tag=NextTS)&(c>(M−N)) −> (c'=(c−1))&(tag'=TS); // Commands
    [t1]  (tag=TS)&(c>(M−N)) −> (c/M):(tag'=NextTS) + (1−(c/M)):(tag'=Collision);
endmodule
```

Model checking: $Pmin=?[F(tag = Collision)]$

Result: 0.52

Collision Probability

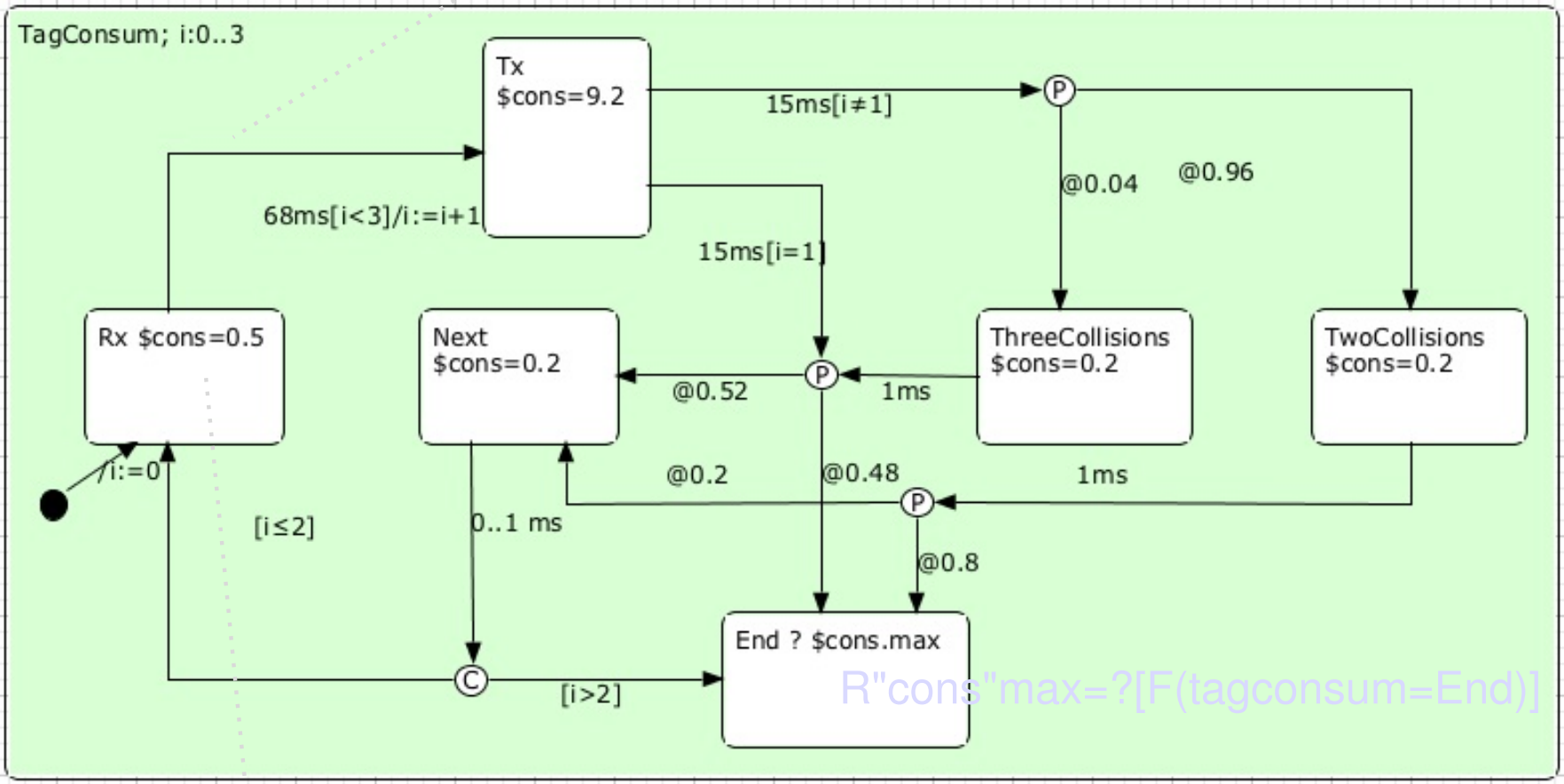


For fixed N, the collision probability decreases if M increases

For fixed M, the collision probability decreases if N decreases

Collection Period Power Consumption

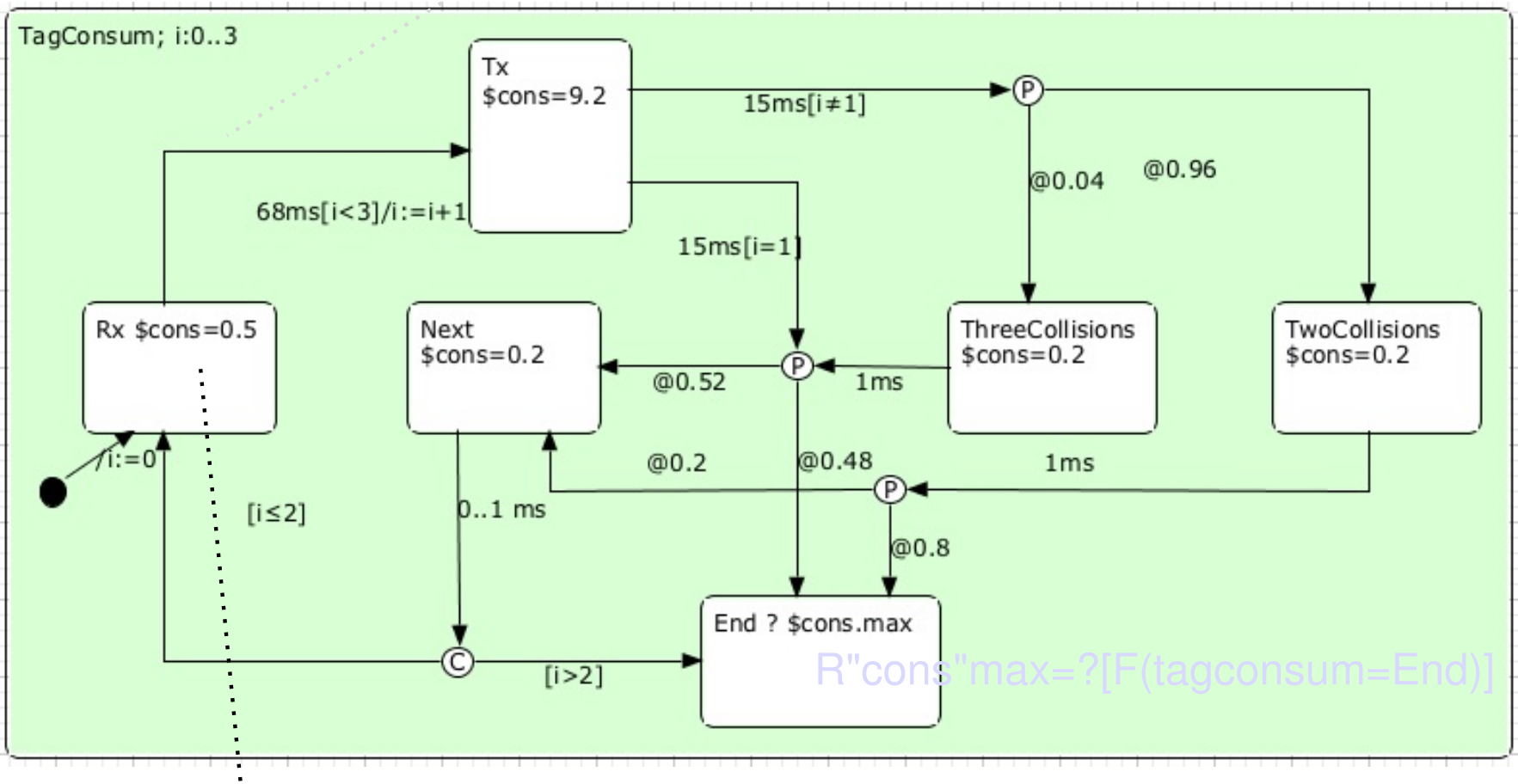
[] (tagconsum=Rx)&(i<3)&(tagconsumclk=68) ->
(i'=(i+1))&(tagconsum'=Tx)&(tagconsumclk'=0);



Clock constraint: tagconsum=Rx=>tagconsumclk<=68
Costs: (tagconsum=Rx): 0.5;

Collection Period Power Consumption

[] (tagconsum=Rx)&(i<3)&(tagconsumclk=68) ->
(i'=(i+1))&(tagconsum'=Tx)&(tagconsumclk'=0);

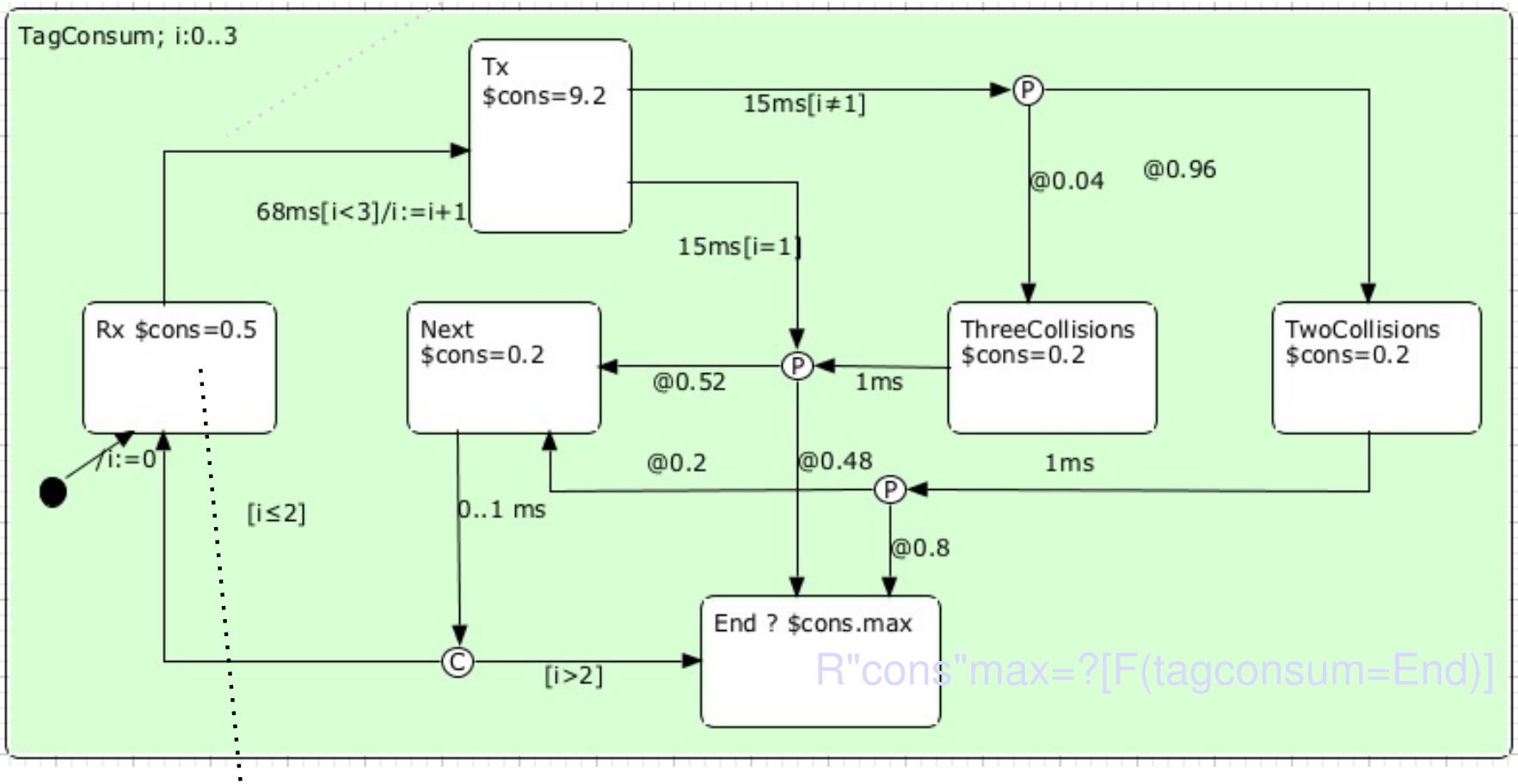


Clock constraint: tagconsum=Rx=>tagconsumclk<=68

Costs: (tagconsum=Rx): 0.5;

Collection Period Power Consumption

[] (tagconsum=Rx)&(i<3)&(tagconsumclk=68) ->
(i'=(i+1))&(tagconsum'=Tx)&(tagconsumclk'=0);

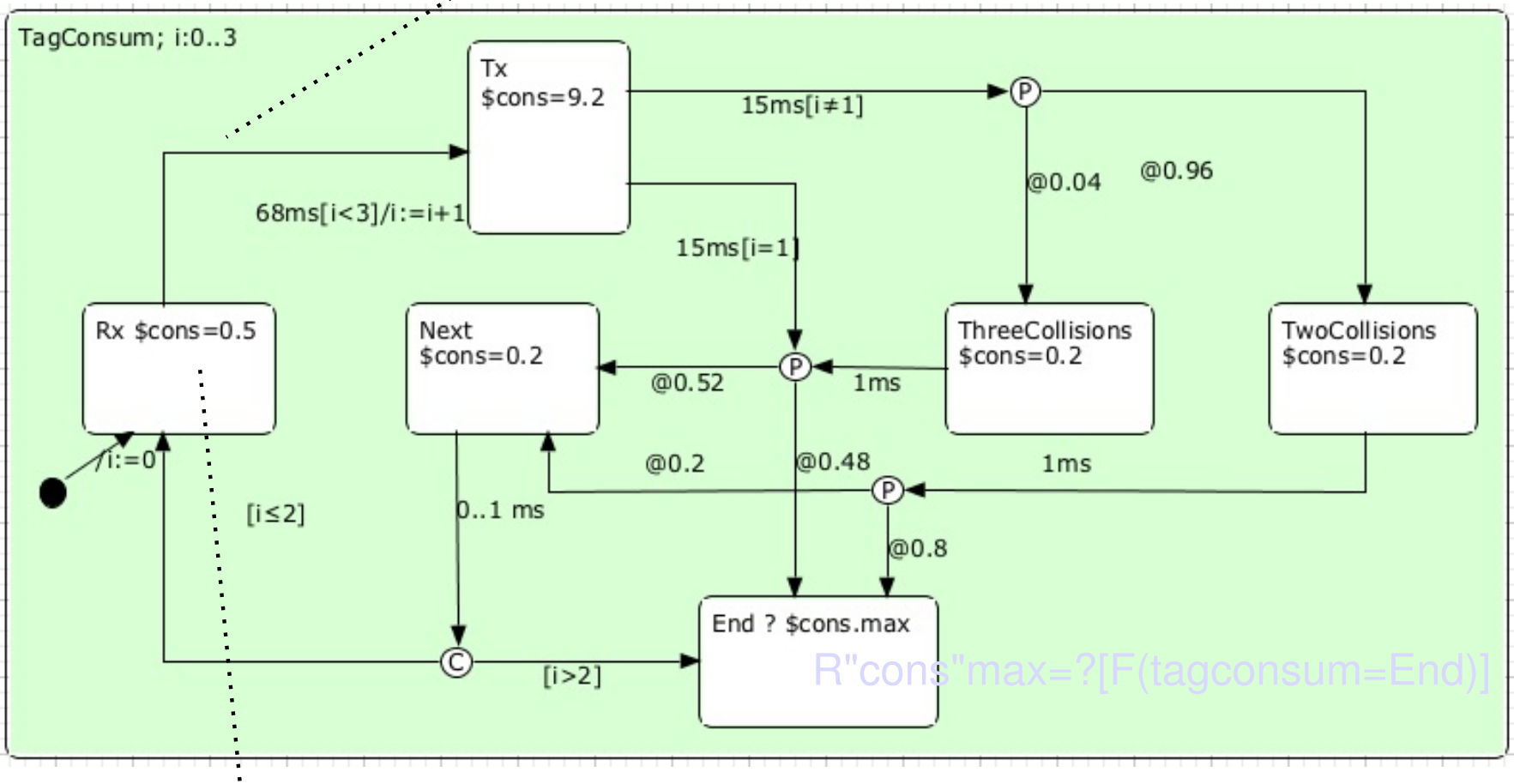


Clock constraint: tagconsum=Rx=>tagconsumclk<=68

Costs: (tagconsum=Rx): 0.5;

Collection Period Power Consumption

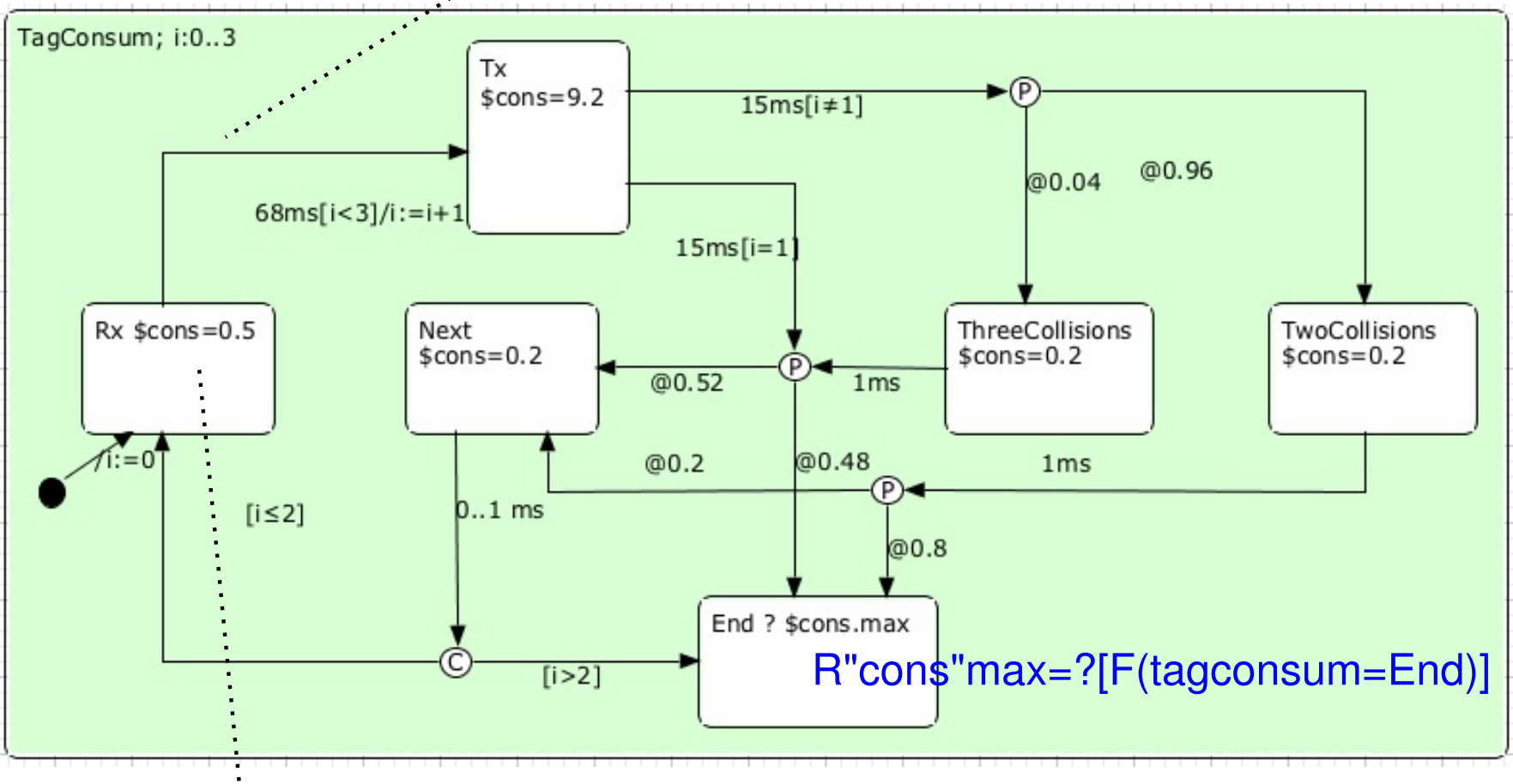
[] (tagconsum=Rx)&(i<3)&(tagconsumclk=68) ->
(i'=(i+1))&(tagconsum'=Tx)&(tagconsumclk'=0);



Clock constraint: tagconsum=Rx=>tagconsumclk<=68
Costs: (tagconsum=Rx): 0.5;

Collection Period Power Consumption

[] (tagconsum=Rx)&(i<3)&(tagconsumclk=68) ->
(i'=(i+1))&(tagconsum'=Tx)&(tagconsumclk'=0);



Clock constraint: tagconsum=Rx=>tagconsumclk<=68
Costs: (tagconsum=Rx): 0.5;

Generated PRISM Code PTA

```
pta //   Probabilistic  Timed Automata

const Rx=0; const TwoTags=1; const ThreeCollisions=2; // Constants
const Tx=3; const End=4; const Next=5;

module powerconscp2
    tagconsum:[0..5] init  Rx;      //  Variables
    tagconsumclk : clock;
    i :[0..3]   init  0;

    invariant
        (tagconsum=Rx=>tagconsumclk<=68)
        & (tagconsum=TwoTags=>tagconsumclk<=1)
        & (tagconsum=ThreeCollisions=>tagconsumclk<=1)
        & (tagconsum=Tx=>tagconsumclk<=15)
        & (tagconsum=Next=>tagconsumclk<=1)
    endinvariant
     ...
```

Generated PRISM Code PTA Continue

```
    [] (tagconsum=Tx)&(i=0)&(tagconsumclk=15) -> 0.52:(tagconsum'=Next)
        &(tagconsumclk'=0) + 0.48:(tagconsum'=End)&(tagconsumclk'=0);
    [] (tagconsum=Rx)&(i<3)&(tagconsumclk=68) -> (i'=(i+1))&(tagconsum'=Tx)
        &(tagconsumclk'=0);
    [] (tagconsum=Next)&(tagconsumclk>=0)&(tagconsumclk<=1) ->
        (tagconsum'=(i<=2)?Rx:End)&(tagconsumclk'=0);
  ...
endmodule

rewards "cons"            // State rewards [mA]
    (tagconsum=Rx): 0.5;
    (tagconsum=TwoTags): 0.2;
    (tagconsum=ThreeCollisions): 0.2;
    (tagconsum=Tx): 9.2;
    (tagconsum=Next): 0.2;
endrewards
```

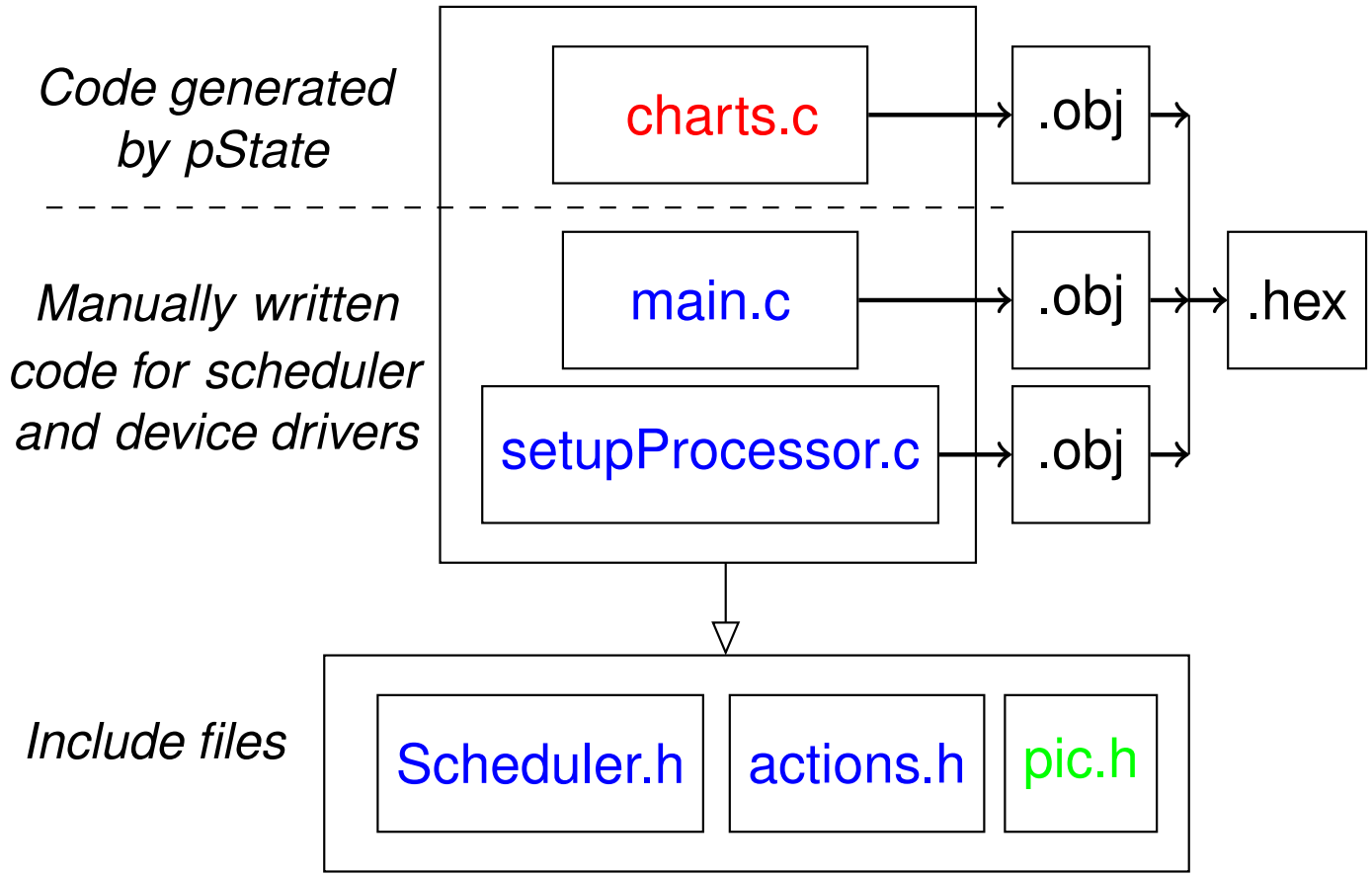
R"cons"max=?[F(tagconsum=End)] returns 216.69 [mAms]

Pmax=?[F(tagconsum=End)&(i=3)] returns 0.046 [probability]

Executable Code Generation

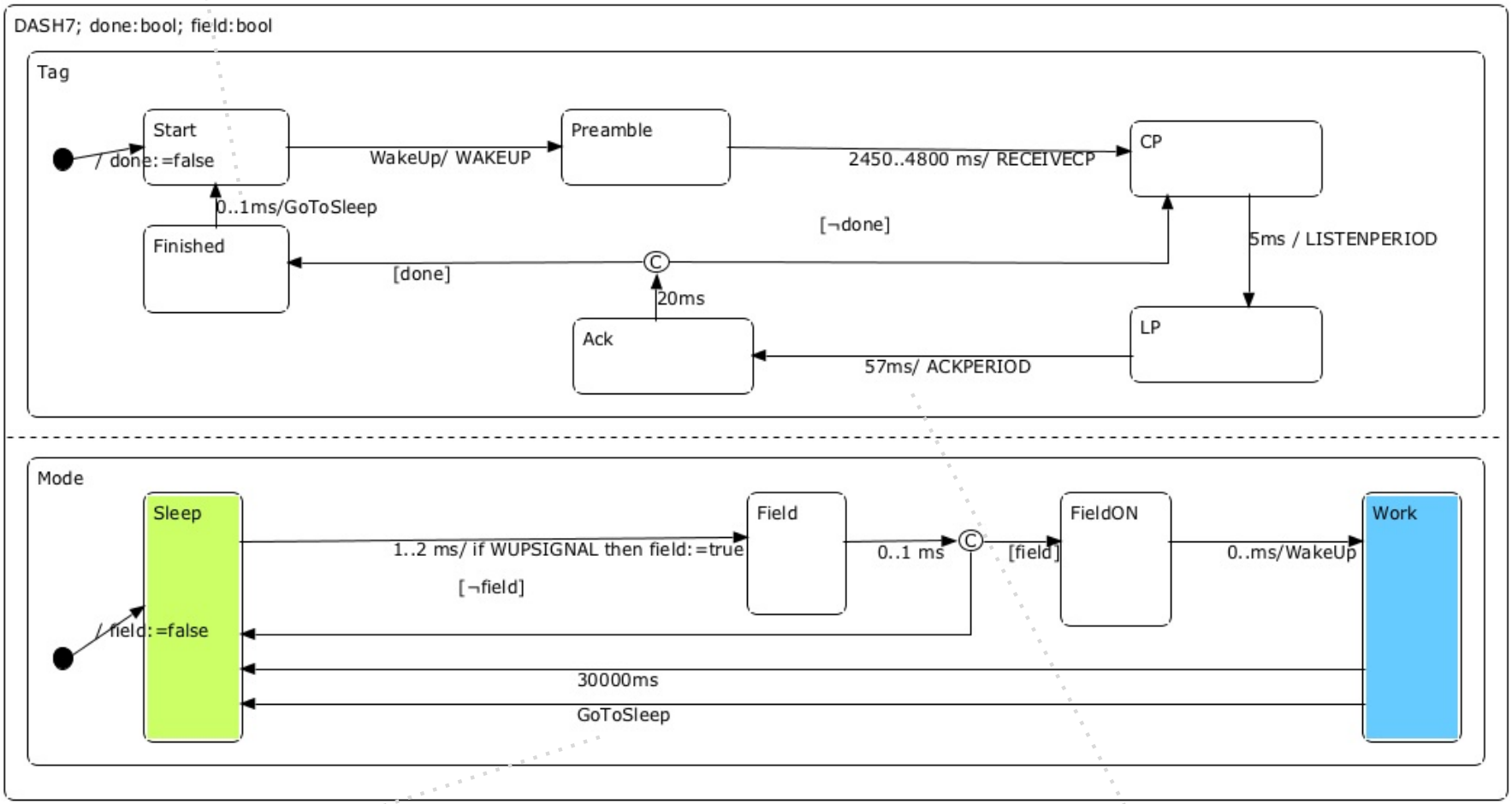
To generate code we need to know the detailed hardware design

pState generates the framework of the executable application with code for all events



Tag Model

void between0(unsigned int t){GoToSleep(); tag = Start; done=false;}

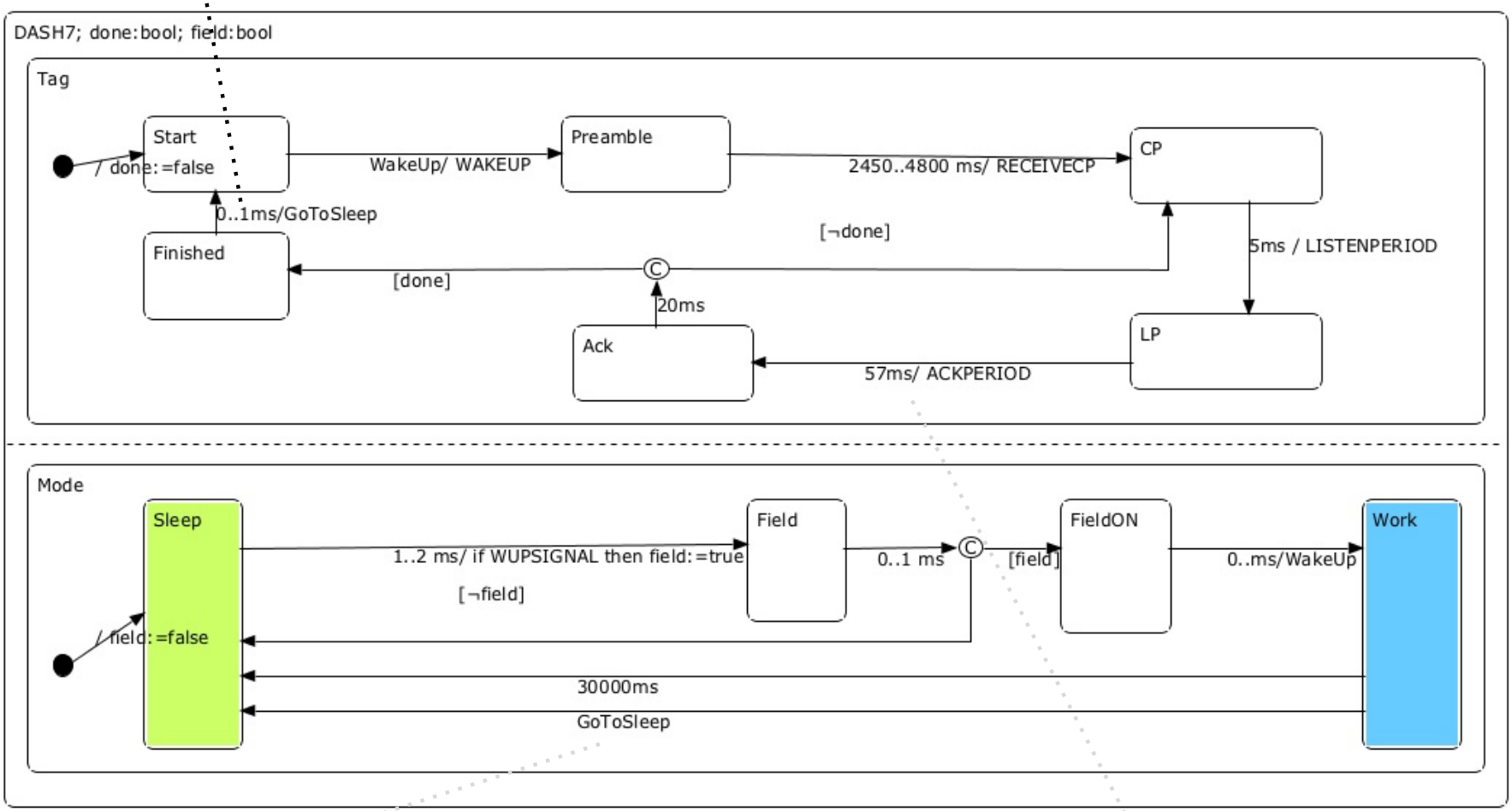


void GoToSleep(){ if ((mode == Work)) {
mode = Sleep; field=false;
schedule(between8, 1, 1); cancel(exactly5); }}

void exactly1(unsigned int t){
ACKPERIOD(); tag = Ack;
schedule(exactly3, 20, 1); }

Tag Model

void between0(unsigned int t){GoToSleep(); tag = Start; done=false;}

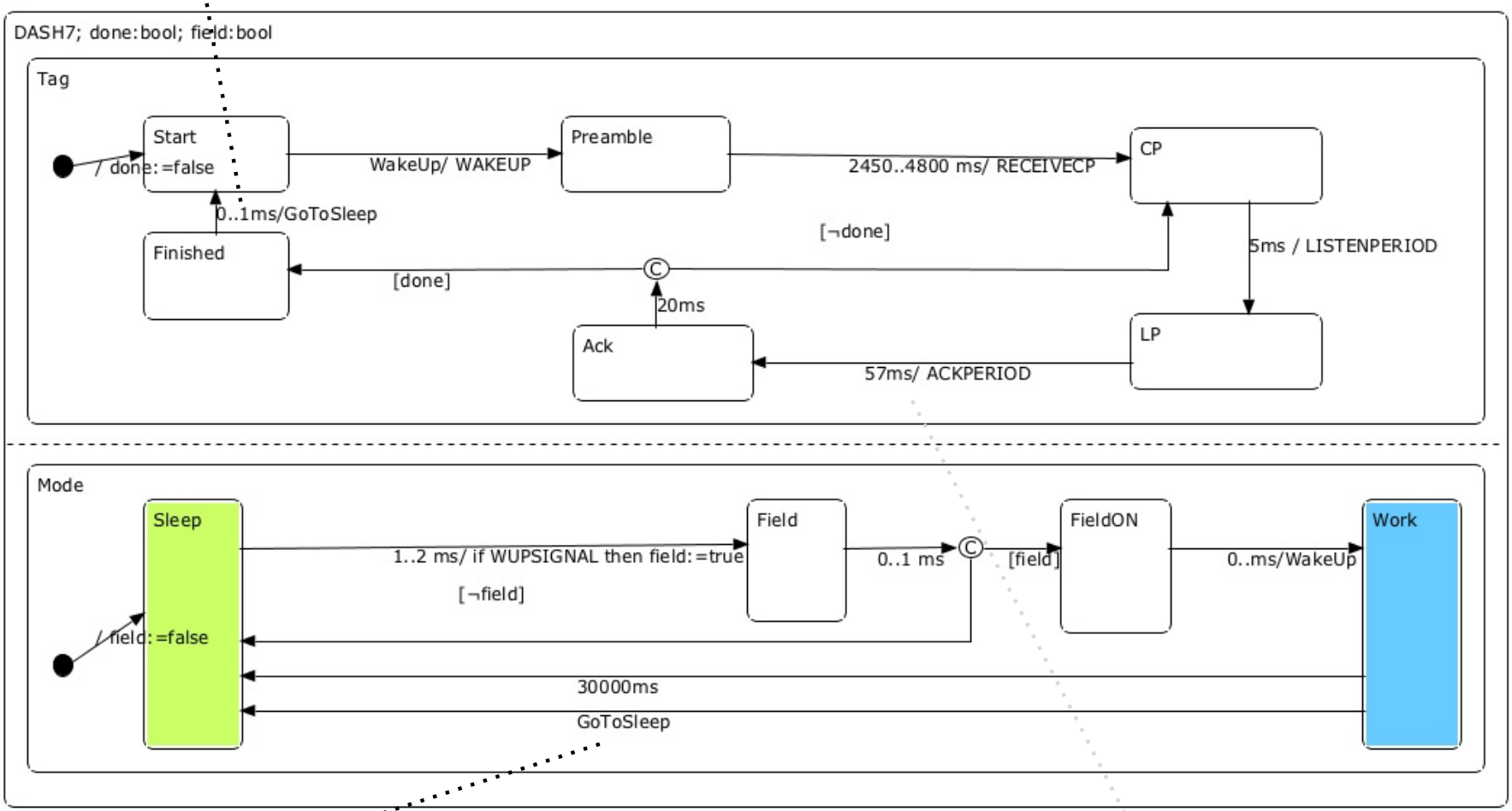


void GoToSleep(){ if ((mode == Work)) {
mode = Sleep; field=false;
schedule(between8, 1, 1); cancel(exactly5); }}

void exactly1(unsigned int t){
ACKPERIOD(); tag = Ack;
schedule(exactly3, 20, 1); }

Tag Model

```
void between0(unsigned int t){GoToSleep(); tag = Start; done=false;}
```

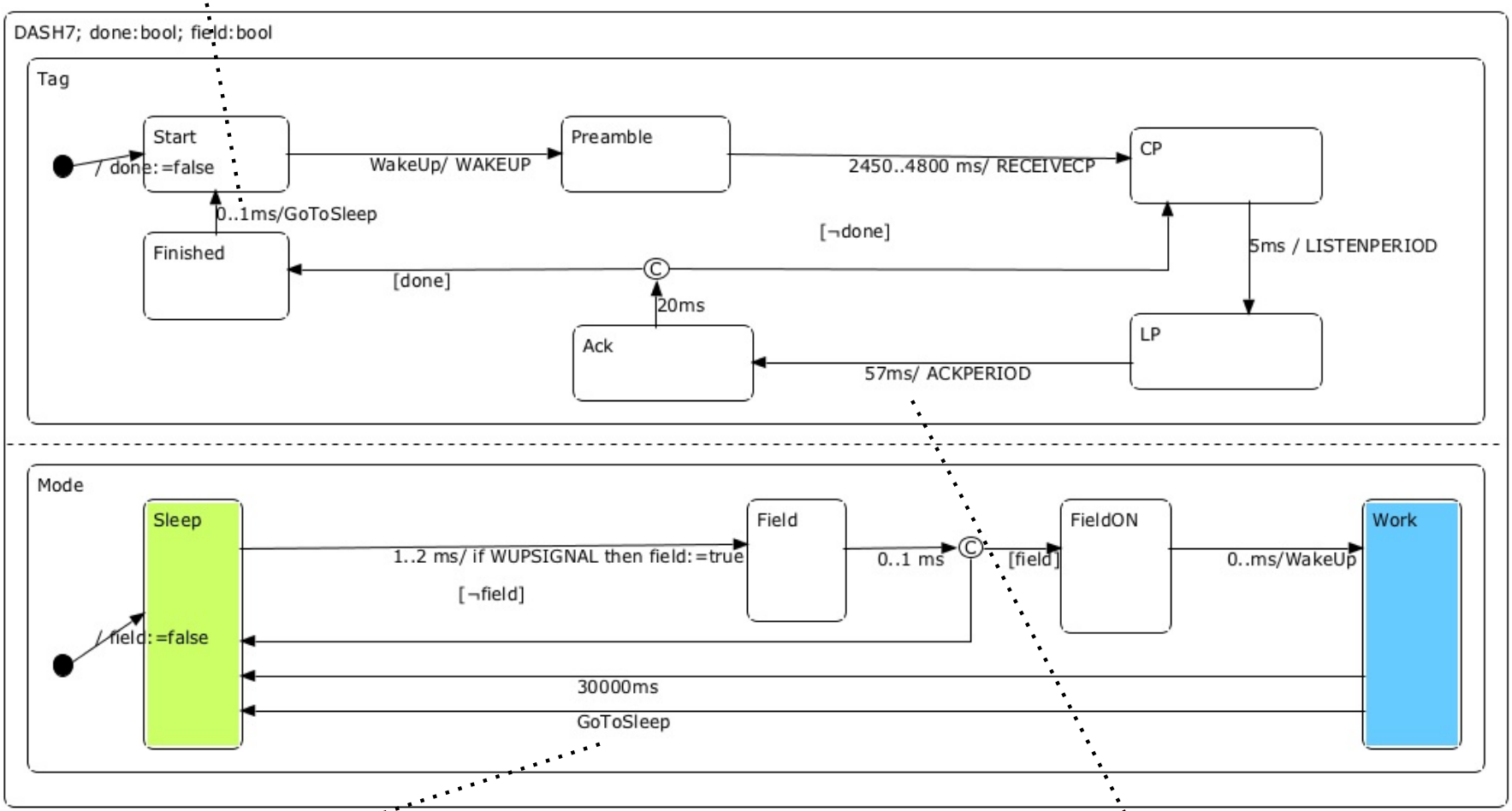



```
void GoToSleep(){ if ((mode == Work)) {
    mode = Sleep; field=false;
    schedule(between8, 1, 1); cancel(exactly5); }}
```

```
void exactly1(unsigned int t){
    ACKPERIOD(); tag = Ack;
    schedule(exactly3, 20, 1); }
```

Tag Model

void between0(unsigned int t){GoToSleep(); tag = Start; done=false;}



void GoToSleep(){ if ((mode == Work)) {
mode = Sleep; field=false;
schedule(between8, 1, 1); cancel(exactly5); }}

void exactly1(unsigned int t){
ACKPERIOD(); tag = Ack;
schedule(exactly3, 20, 1); }

Generated C Code (charts.c)

```
#include <pic.h>
 ...
#include "scheduler.h"
#include "actions.h"

enum tag_status {Finished, LP, Preamble, Ack, CP, Start} tag;  //  Enum variables
enum mode_status {Work, FieldON, Field, Sleep} mode;

bool done; bool field;  //  Global Variables

void  between0 (unsigned int); //  Timed events
void  exactly1  (unsigned int);
void  between2 (unsigned int);
 ...
void   InitVariables (void)  {  //    Initialize    variables
      done=false;

      n = 0;  tr  = 0;  ir  = 1;  //  Event scheduler variables
}
```

Generated C Code (charts.c)

```
void InitChart(void) { // Chart initialization
    tag = Start; done=false; mode = Sleep;
    schedule(between8, 1, 1);
}
void between0(unsigned int t){          // Timed event
    GoToSleep();
    tag = Start; done=false;
}
void exactly1(unsigned int t){   // Timed event
    ACKPERIOD(); tag = Ack;
    schedule(exactly3, 20, 1);
}
void GoToSleep(){ // Broadcasted event
    if ((mode == Work)) {
        mode = Sleep; field=false;
        schedule(between0, 1, 1);
        cancel(exactly1);
    }
}
 ...
```

Conclusion and Future Work

Automated approach from modelling and analysis to code generation

One tool can be used for:

- System property analysis (collision probability)
- Device property analysis (power consumption)
- Device code generation

We can perform formal verification during system specification process

Determining execution time in order to estimate probabilistic worst-case execution time (pWCET)

Thank You

http://pstate.mcmaster.ca/

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

pCharts Structure

$p\mathscr{S} = (Basic, AND, XOR, Root, parent, Event, Transition, default, inv, costs, costt, alt, prob)$

$Basic, AND, XOR$: Mutually disjoint set of states

$Root$: Distinguished XOR state, $Root \in XOR$

$parent : State \nrightarrow State$

$State = Basic \cup Composite$ - set of all states

$Composite$ states $= AND \cup XOR$

$Event$: Finite set of event names

$default : XOR \nrightarrow Transition$ - maps XOR states to default transitions

$inv : State \rightarrow Expr$ - Invariant attached to a state

$costs : State \rightarrow Expr$ - Cost of being in a state

pCharts Structure

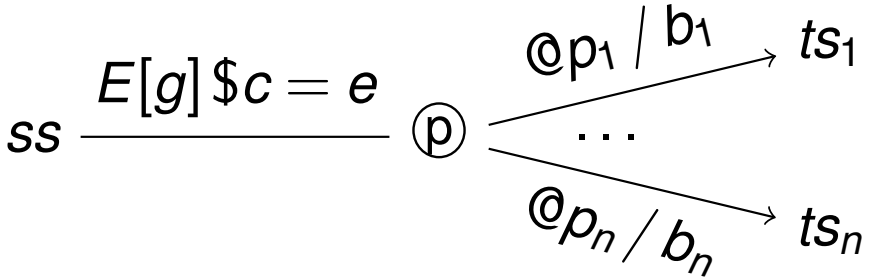
$p\mathscr{S}$ = (*Basic*, *AND*, *XOR*, *Root*, *parent*, *Event*, *Transition*, *default*, *inv*, *costs*, *costt*, *alt*, *prob*)

Basic, *AND*, *XOR* : Mutually disjoint set of states
Root : Distinguished XOR state, $Root \in XOR$
parent : $State \nrightarrow State$
    $State = Basic \cup Composite$ - set of all states
    Composite states = $AND \cup XOR$
Event : Finite set of event names
default : $XOR \nrightarrow Transition$ - maps *XOR* states to default transitions
inv : $State \rightarrow Expr$ - Invariant attached to a state
costs : $State \rightarrow Expr$ - Cost of being in a state

pCharts Structure - Transition

costt : *Transition* $\rightarrow$ *Expr* - Cost of taking transition
alt : *Transition* $\rightarrow$ *Alternative* - Transition probabilistic alternatives
prob : *Alternative* $\rightarrow$ *Expr* - Probability of an alternative

Transition :



$ss \subseteq$ *State* - Set of source states; $ts_i \subseteq$ *State* - Set of target states
$E \in$ *Event* - Transition event; g - Boolean expression, transition guard
b_i - Statement, transition body; c - Costs of a transition
p_i - Transition probability $\sum_{i=1}^{n} p_i = 1$, $0 \leq p \leq 1$, and $p \in R$, $n \in N$

Translation

Form charts to statements:

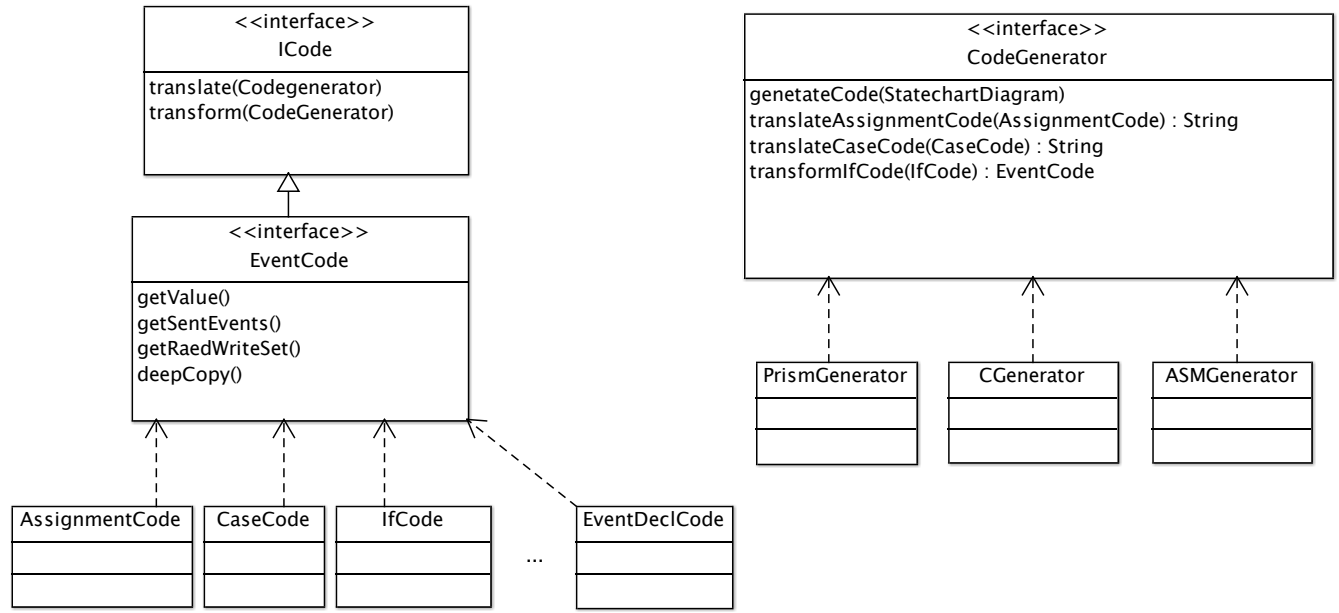
- Nested statements suitable for programming languages
- Flat guarded commands suitable for model checkers

$$
\begin{array}{lcl}
\mathit{test}(s) & \widehat{=} & \mathit{lc}(\mathit{parent}(s)) = \mathit{uc}(s) \\
\mathit{assign}(s) & \widehat{=} & \mathit{lc}(\mathit{parent}(s)) := \mathit{uc}(s) \\
\mathit{in}(ss) & \widehat{=} & \forall\, s \in ss \cap \mathit{children}[\mathit{XOR}] \;.\; \mathit{test}(s) \\
\mathit{goto}(ss) & \widehat{=} & \parallel s \in ss \cap \mathit{children}[\mathit{XOR}] \;.\; \mathit{assign}(s)
\end{array}
$$

Event translation:

$$
\begin{array}{lcl}
\mathit{trigger}(t) & \widehat{=} & \mathit{in}(\mathit{exited}(t)) \wedge \mathit{guard}(t) \\
\mathit{effect}(t) & \widehat{=} & \oplus\, c \in \mathit{alt}(t) \;.\; \mathit{prob}(c) : \\
 & & \quad (\parallel (s,d) \in \mathit{comp}(t,c) \;. \\
 & & \qquad \mathit{body}(d) \parallel \mathit{goto}(\mathit{entered}(s,d)))
\end{array}
$$

Visitor pattern




```
public String translate(CodeGenerator cg) {
    return cg.translateIfCode(this);
 }

 public EventCode transform(CodeGenerator cg) {
    return cg.transformIfCode(this);
 }
```

```
public String translate(CodeGenerator cg) {
    return cg.translateEventDeclCode(this);
 }

 public EventCode transform(CodeGenerator cg) {
    return cg.transformEventDeclCode(this);
 }
```

pCharts Semantics

- Synchronous reactive system
- Preemtive - transition in top level state machine has priority over transitions in the refinement state machine
- Reset always initializes the refinement of the destination state to its initial state
- No history transitions
- More compact flatten state machine
- We can think about hierarchical state machine as syntactic sugar, compact representation of flat-state machine