Preambula

- There are 8 USB keys circulating, containing
  - Oracle VirtualBox
  - Ubuntu 12.04 (with the installation instructions — HTML page)
  - BIP and all necessary packages (.deb)
- Exercises and a PDF with full installation instructions at

  https://documents.epfl.ch/users/b/bl/bliudze/www/

- Update: 5 of the USB keys also contain exercises now!

Rigorous Component-Based Design in BIP

Tutorial @ CompArch
2nd of July, 2014

Simon Bliudze

École polytechnique fédérale de Lausanne
Rigorous System Design Laboratory

Semaphores, locks, monitors, etc.

Coordination based on low-level primitives rapidly becomes unpractical.

Synchronisation

Task 1:

```
...
free(S1);
take(S2);
...
```

Task 2:

```
...
take(S1);
free(S2);
...
```

A simple synchronisation barrier

Synchronisation

Task 1:

```
...
free(S1);
free(S1);
take(S2);
take(S3);
...
```

Task 2:

```
...
take(S1);
free(S2);
free(S2);
take(S3);
...
```

Task 3:

```
...
take(S1);
take(S2);
free(S3);
free(S3);
...
```

Three-way synchronisation barrier

Synchronisation with data transfer

Task 1:

```
x = f1(sh1,sh2);
free(S1);
take(S2);
sh1 = x;
free(S1);
take(S2);
x = f2(sh1,sh2);
```

Task 2:

```
y = g1(sh1,sh2);
take(S1);
free(S2);
sh2 = y;
take(S1);
free(S2);
y = g2(sh1,sh2);
```

Coordination mechanisms mix up with computation and do not scale.
Code maintenance is a nightmare!

Synchronisation with data transfer

Task 1:

```
x = f1(sh1,sh2);
free(S1);
take(S2);
sh1 = x;
free(S1);
take(S2);
x = f2(sh1,sh2);
```

Task 2:

```
y = g1(sh1,sh2);
take(S1);
free(S2);
sh2 = y;
take(S1);
free(S2);
y = g2(sh1,sh2);
```

Coordination mechanisms mix up with computation and do not scale.
Code maintenance is a nightmare!

Objectives

- Make developing correct concurrent systems easier
- Separate computation from coordination
- “Run the model you verified”

Tutorial outline

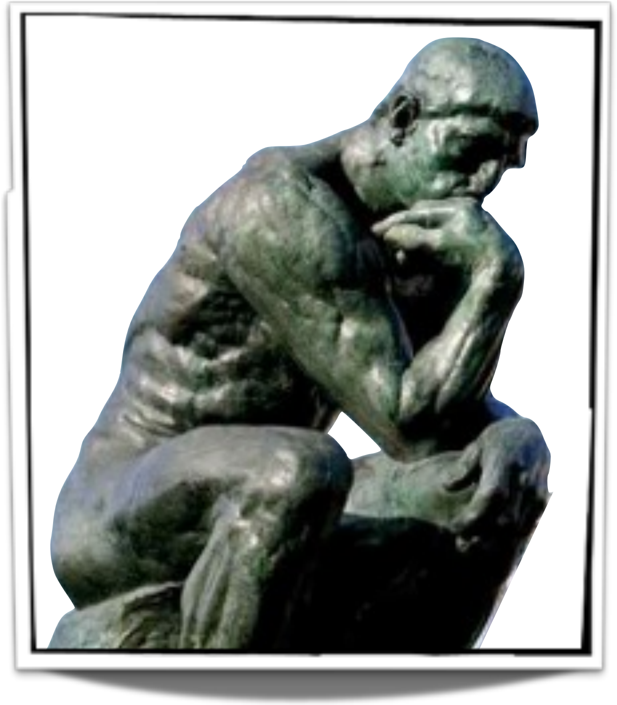
- Introduction
- Hands-on BIP
- Flavours of BIP
- Architectures in BIP (announcement)

Introduction

Motivation and Component model

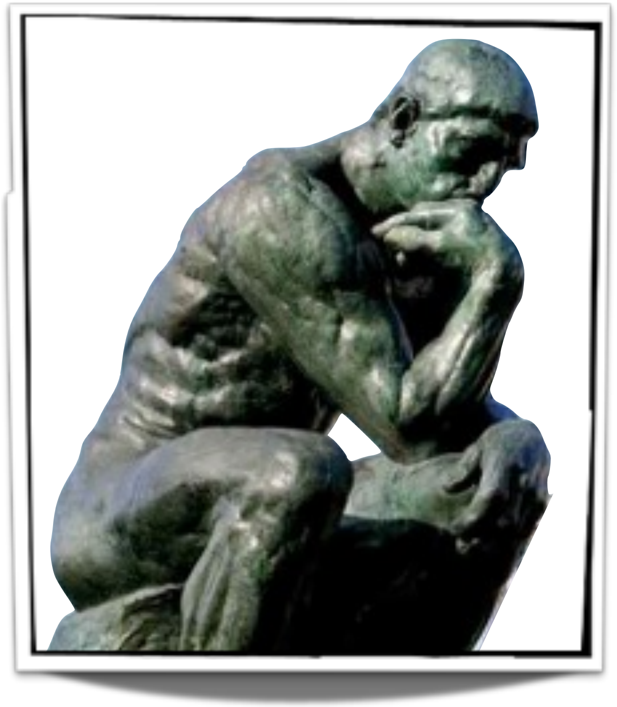
- Motivation
    - Unifying modelling formalism for managing system complexity
- BIP component model
    - Basic component model
    - Formal semantics and engine-driven execution

Managing system complexity



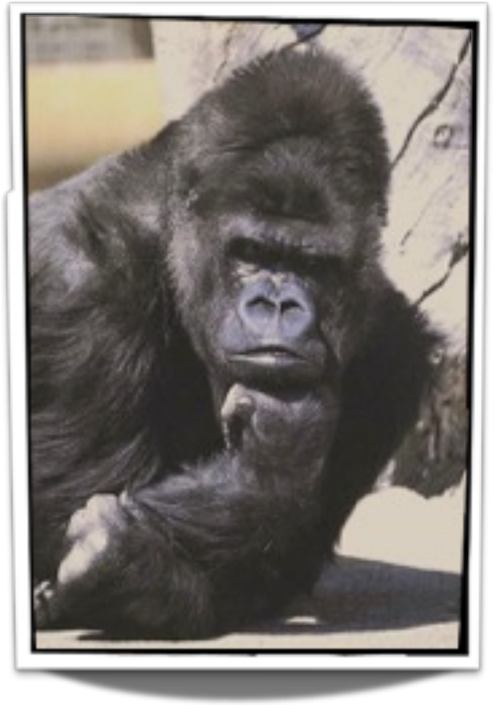
Managing system complexity

- Mastering system complexity requires
  - Manipulating models to raise the abstraction level
  - Expressive enough to avoid ad-hoc solutions
  - Simple enough to be acceptable for engineers



Managing system complexity

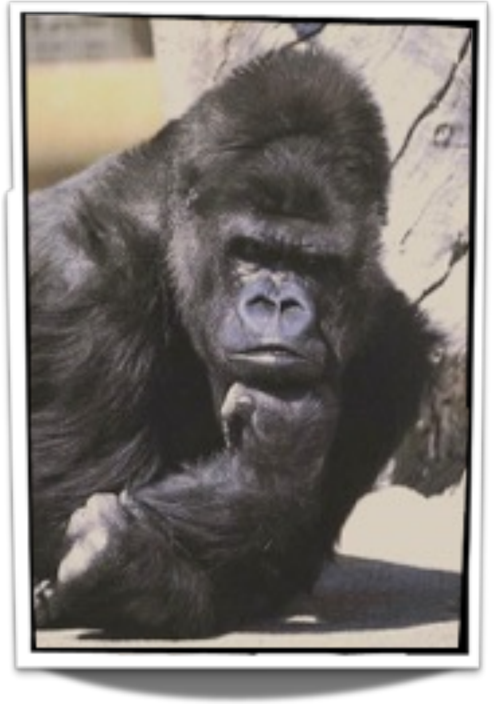
- Mastering system complexity requires
  - Manipulating models to raise the abstraction level
  - Expressive enough to avoid ad-hoc solutions
  - Simple enough to be acceptable for engineers



- Bridging the gap between high-level models and run-time code
  - Raising abstraction level increases the gap
  - Model and implementation must be provably equivalent

Managing system complexity

- Mastering system complexity requires
  - Manipulating models to raise the abstraction level
  - Expressive enough to avoid ad-hoc solutions
  - Simple enough to be acceptable for engineers
- Bridging the gap between high-level models and run-time code
  - Raising abstraction level increases the gap
  - Model and implementation must be provably equivalent
- We should build solid and light-weight bridges

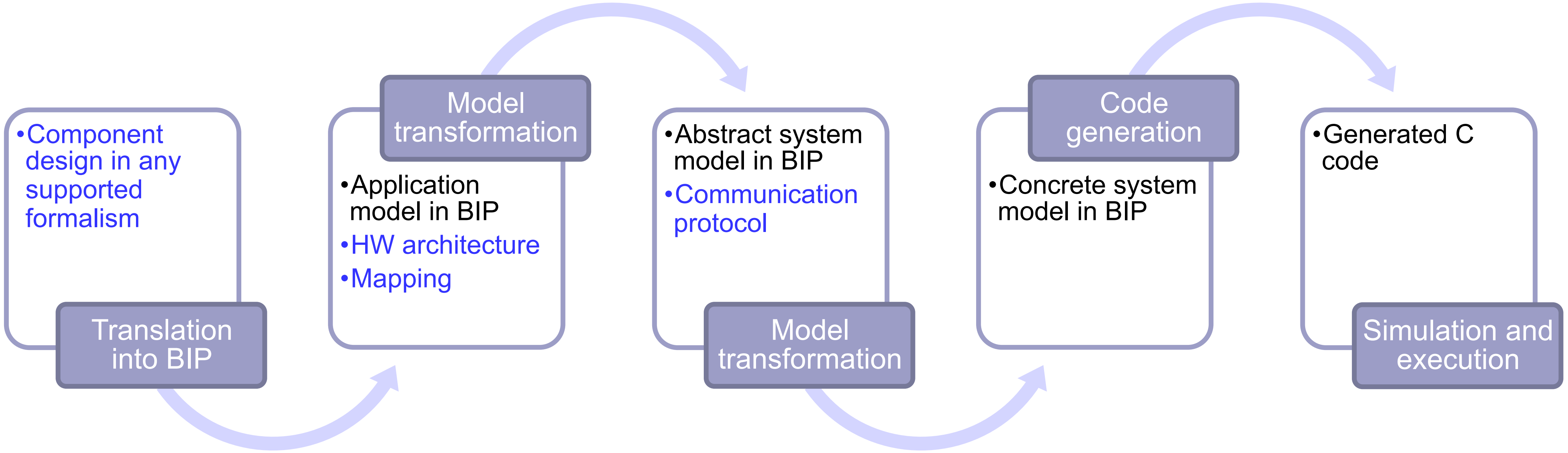


Unifying modelling formalism

- Solid:
  - Clearly established formal semantics
  - Encompassing heterogeneity
    - computation, execution, implementation
  - Proven code generation chain

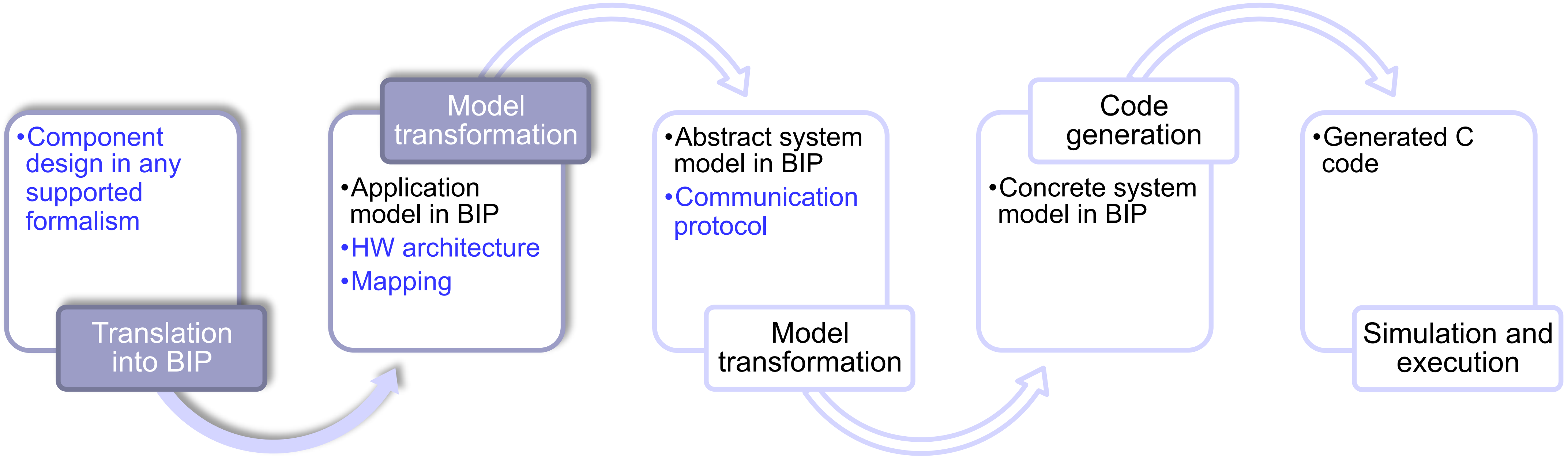
- Light-weight:
  - Clear, accessible formal semantics
  - Minimal set of primitives
  - Separation of concerns
    - computation and
    - coordination
  - Efficient implementation for popular platforms

Rigorous System Design



- Models progressively refined with new information
  - In light blue — provided by the designer
  - In **black** — generated by automatic transformation tools

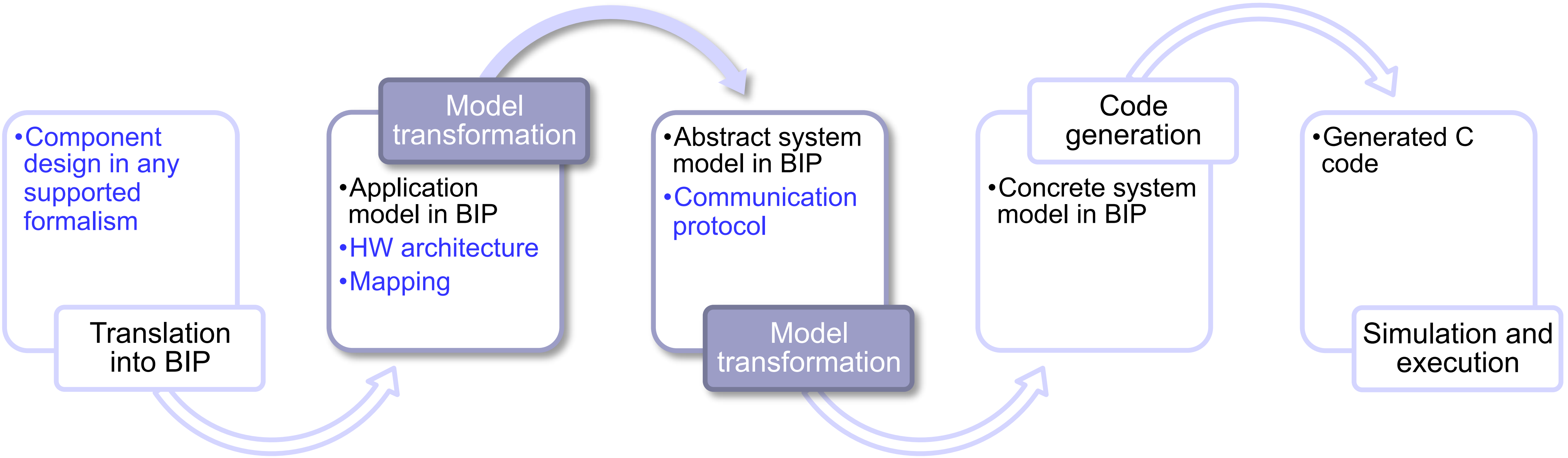
Application model



- Application model is designed directly in BIP or…
- …using a language factory transformation from
  - C, AADL, NesC/TinyOS, MathLab/Simulink, Lustre, DOL, GeNoM

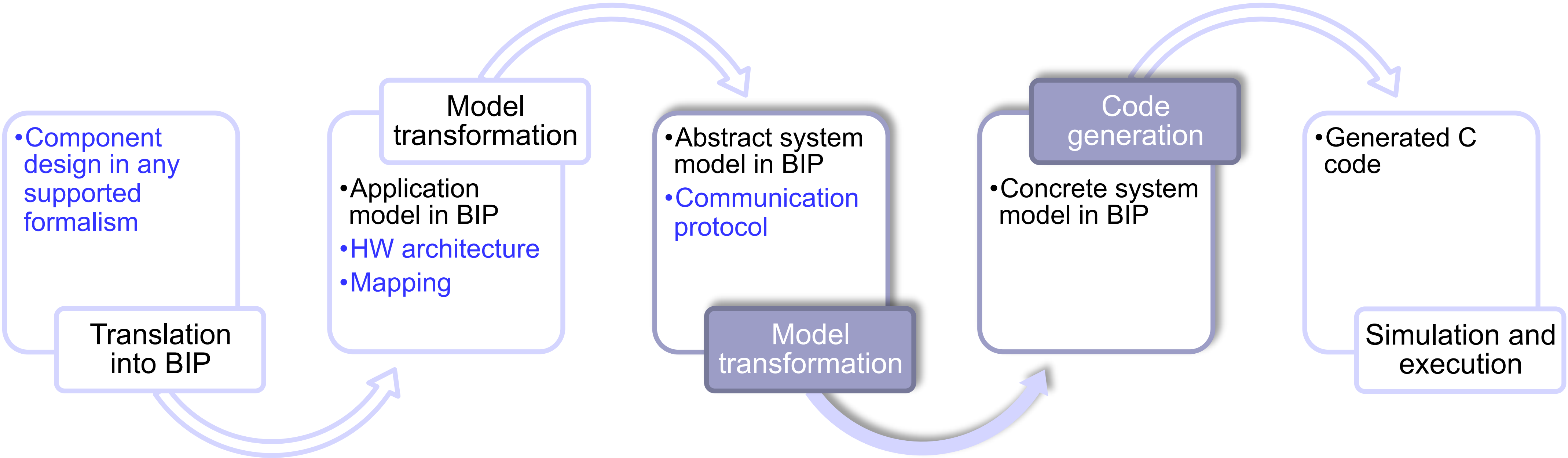
- Safety properties are verified on this model
  - Compositional and incremental deadlock detection (D-Finder tool)
  - High performance even on models that other tools fail to analyze

Abstract system model



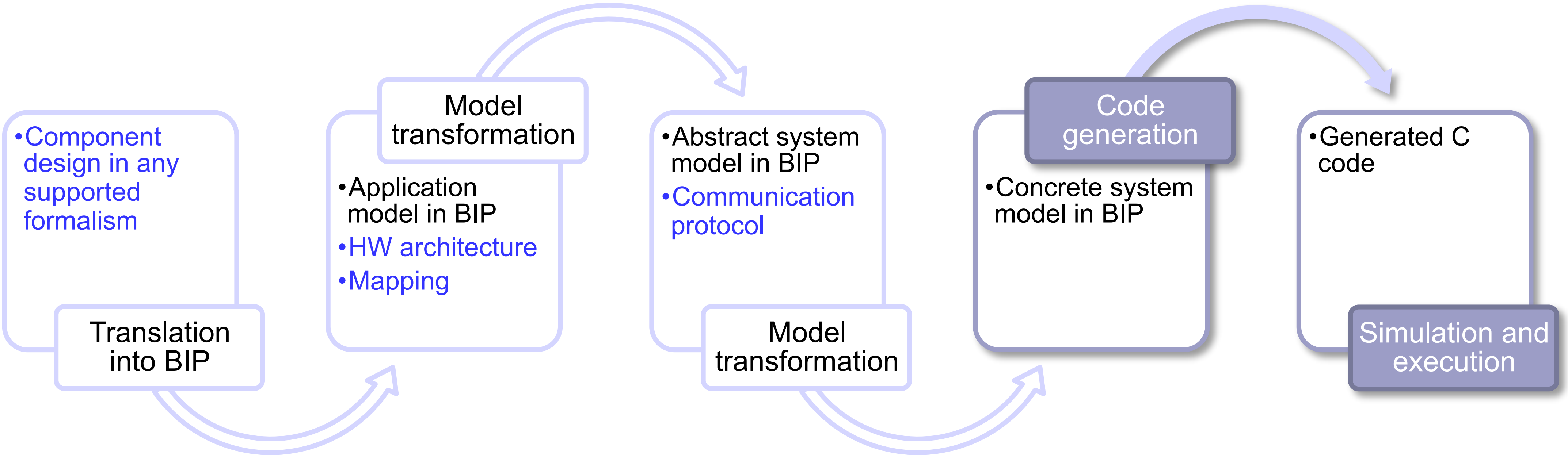
- Abstract system model is generated by a transformation using
  - The model of the target execution platform (processor(s), memory, etc.)
  - A mapping of atomic components to the processing units
- It takes in account
  - The hardware architecture constraints (e.g. mutual exclusion)
  - The execution times of atomic actions
  - The scheduling policies seeking optimal resource utilisation.

Concrete system model



- Concrete system model is obtained by expressing high level BIP coordination mechanisms…
  - Atomic multiparty interactions
  - Priorities
- …by using primitives of the execution platform
  - For examle, protocols using asynchronous message passing

Code generation



- C++ code is automatically generated for each processing unit
- Generated code is monolithic, minimising the coordination overhead

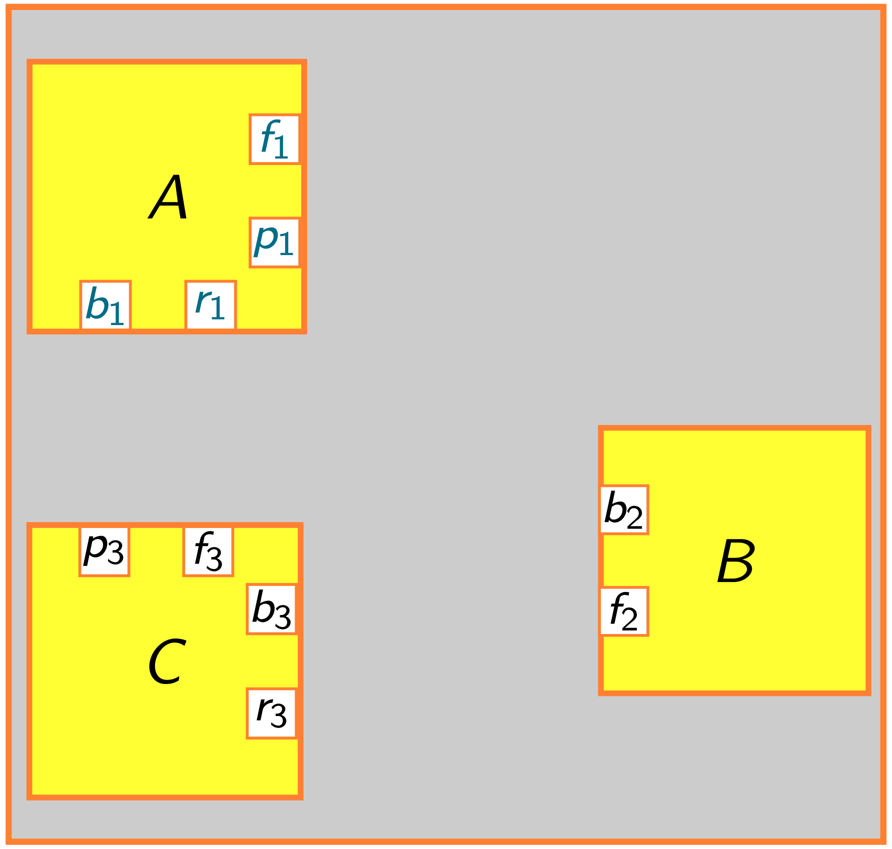
Component-based design

Component-based design

- Three layers

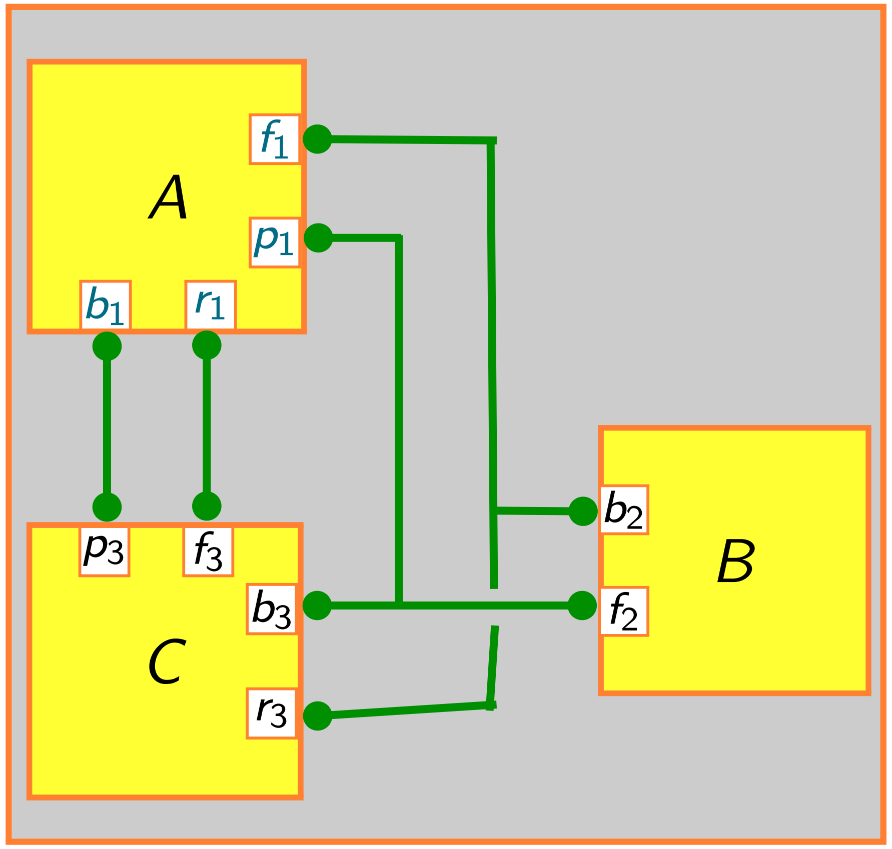
Component-based design

- Three layers
  - Component behaviour



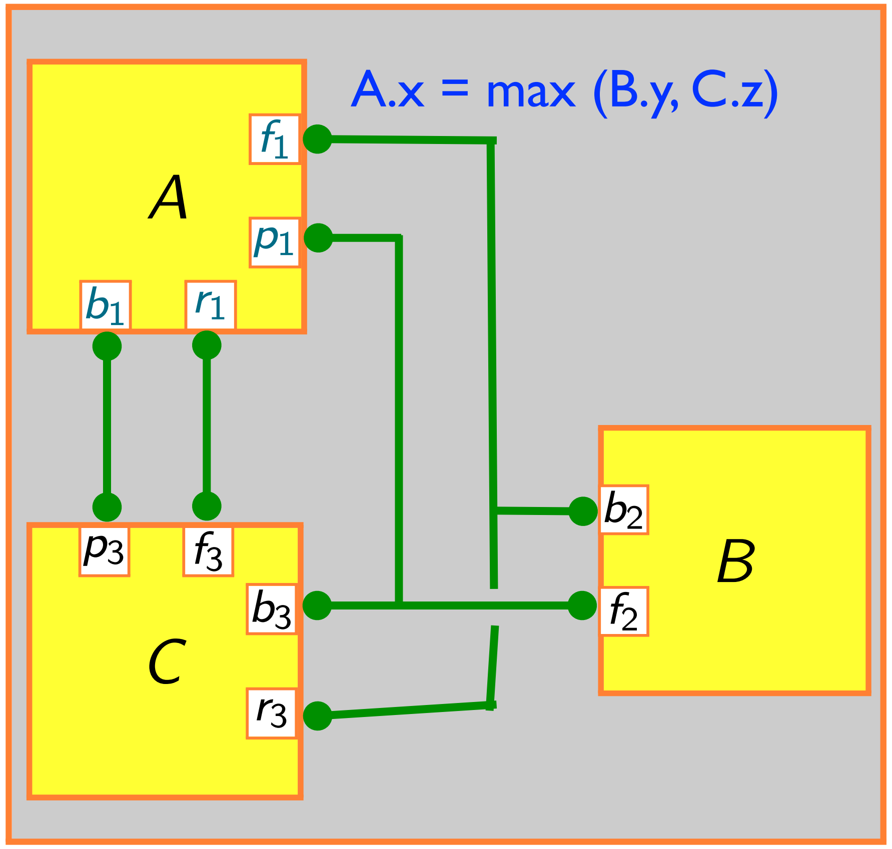
Component-based design

- Three layers
  - Component behaviour
  - Coordination

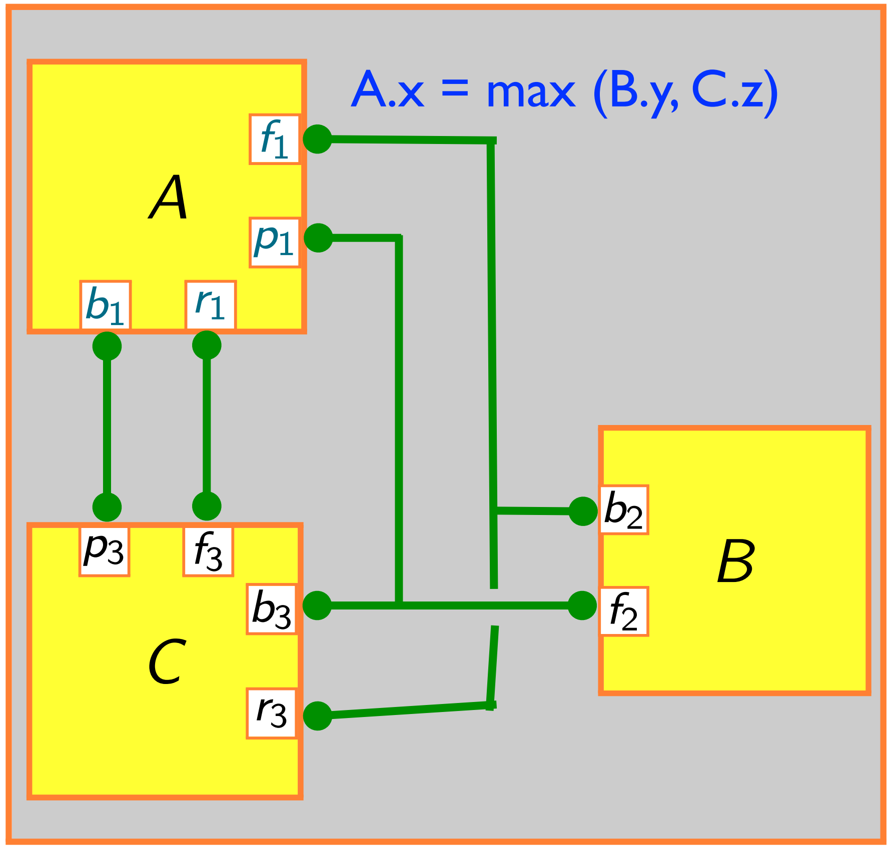


Component-based design

- Three layers
  - Component behaviour
  - Coordination
  - Data transfer



Component-based design



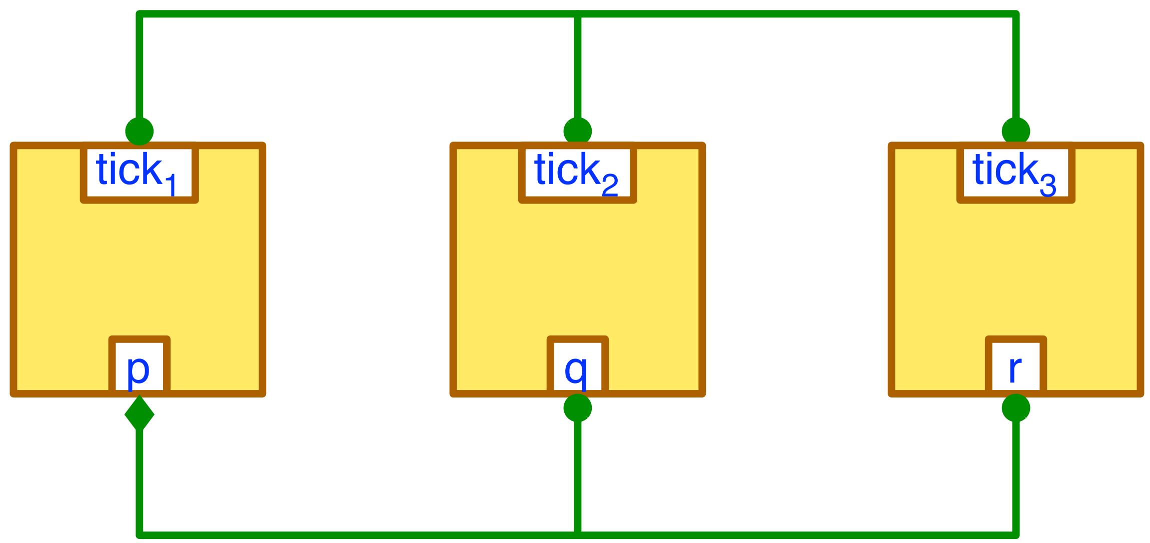
- Three layers
  - Component behaviour
  - Coordination
  - Data transfer
- Interesting results already at this abstraction level
  - Detection of synchronisation deadlocks

    S. Bensalem, M. Bozga, J. Sifakis, T.-H. Nguyen.
    DFinder: A Tool for Compositional Deadlock Detection and Verification [CAV'09]
  - Synthesis of glue for safety properties

    S. Bliudze and J. Sifakis.
    Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems [SC'11]

Connectors

tick1 tick2 tick3

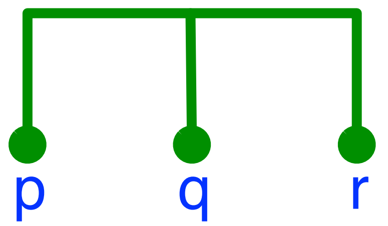


p + pq + pr + pqr

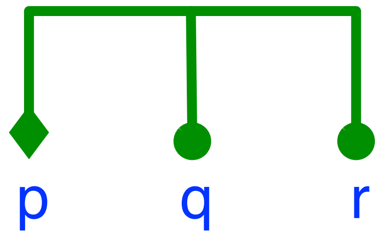
- *Connector* are tree-like structures
  - ports as leaves and nodes of two types
    - *Triggers* (diamonds) — nodes that can “initiate” an interaction
    - *Synchrons* (bullets) — nodes that can only “join” an interaction initiated by others
- In practice, *maximal progress* is implicitly assumed

Connector examples

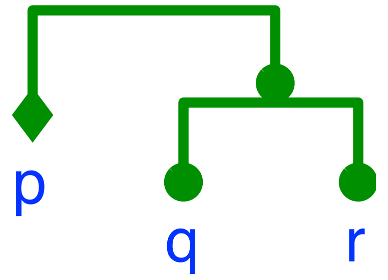
- The Algebra of Connectors



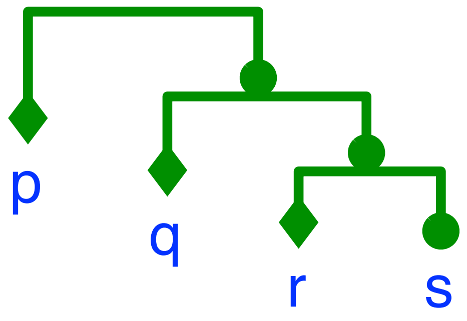
Strong synchronisation: pqr
pqr



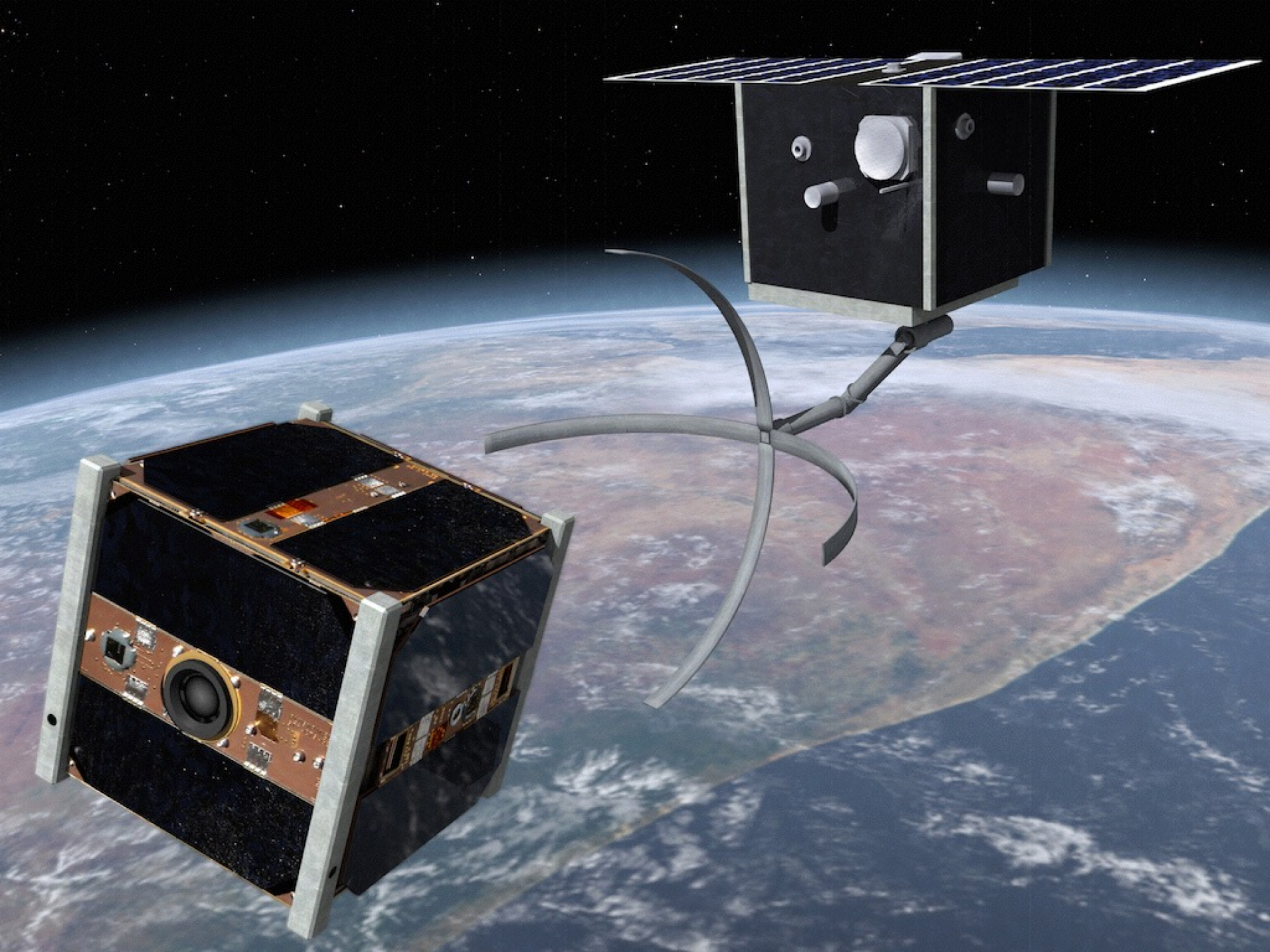
Broadcast: p + pq + pr + pqr
p’qr



Atomic broadcast: p + pqr
p’[qr]



Causal chain: p + pq + pqr + pqrs
p’[q’[r’s]]



Practical example

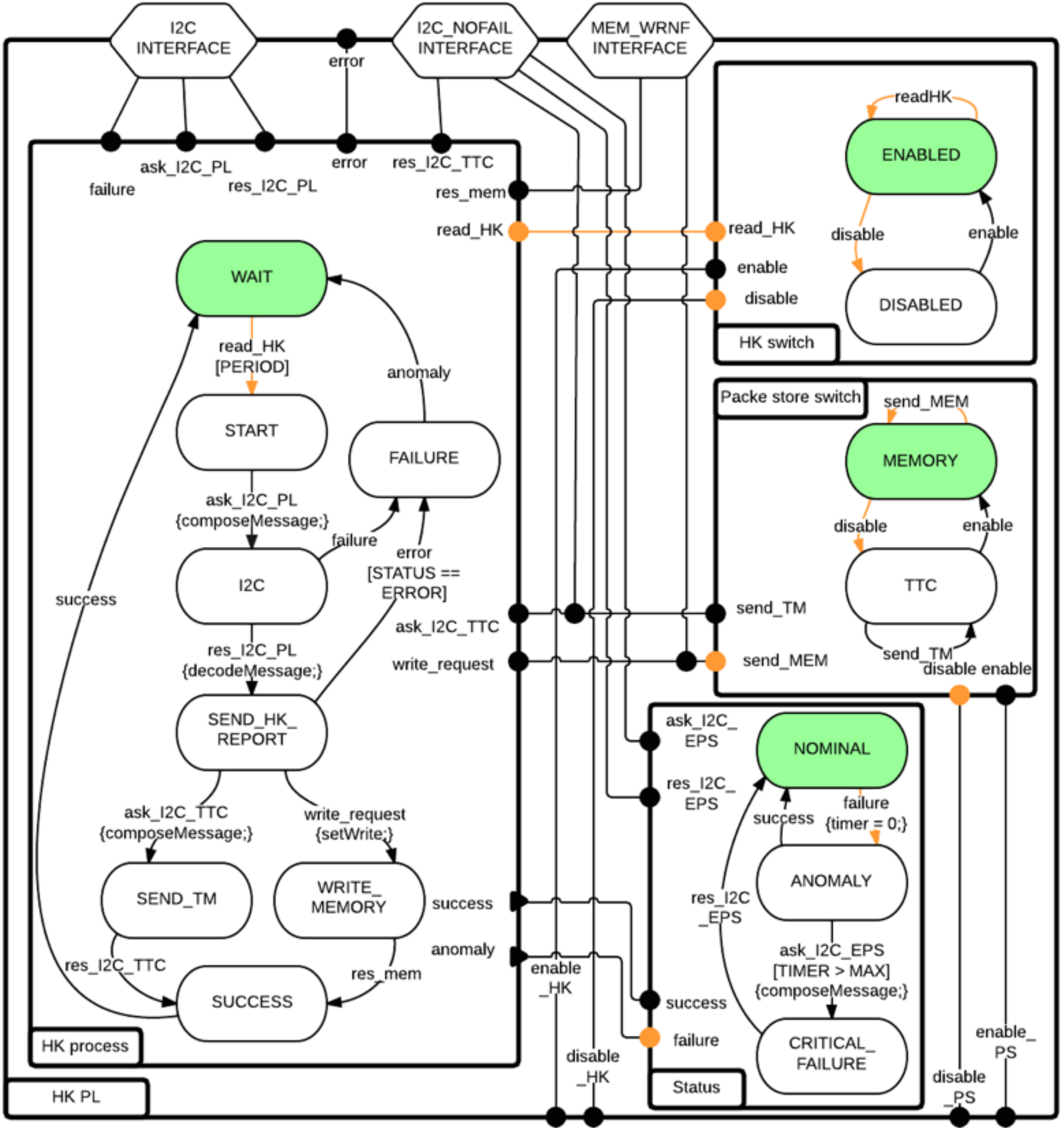
- Satellite software design
  - A collaboration with Swiss Space Center

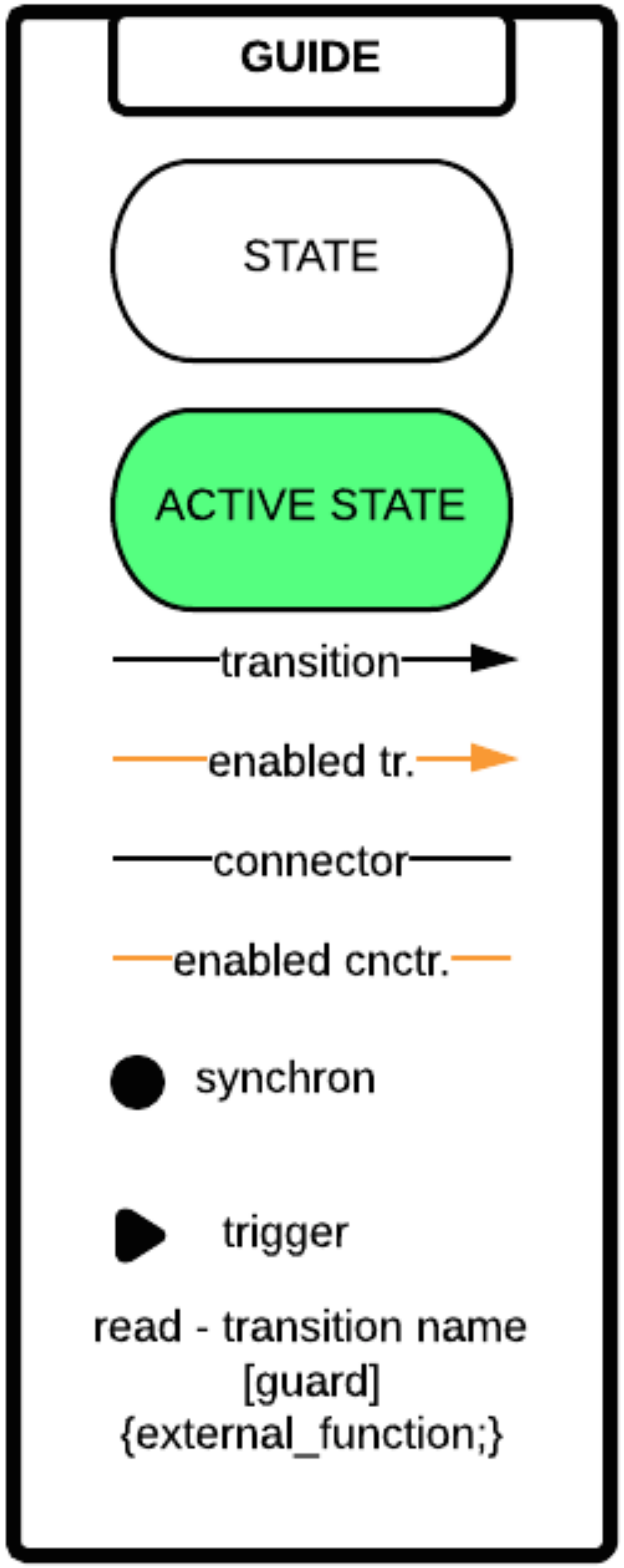
- Component-based design in BIP of the control software for a nano-satellite
  - Attitude Determination and Control System (ADCS)
  - Communication with other subsystems through an I^{2}C bus

swiss
space center

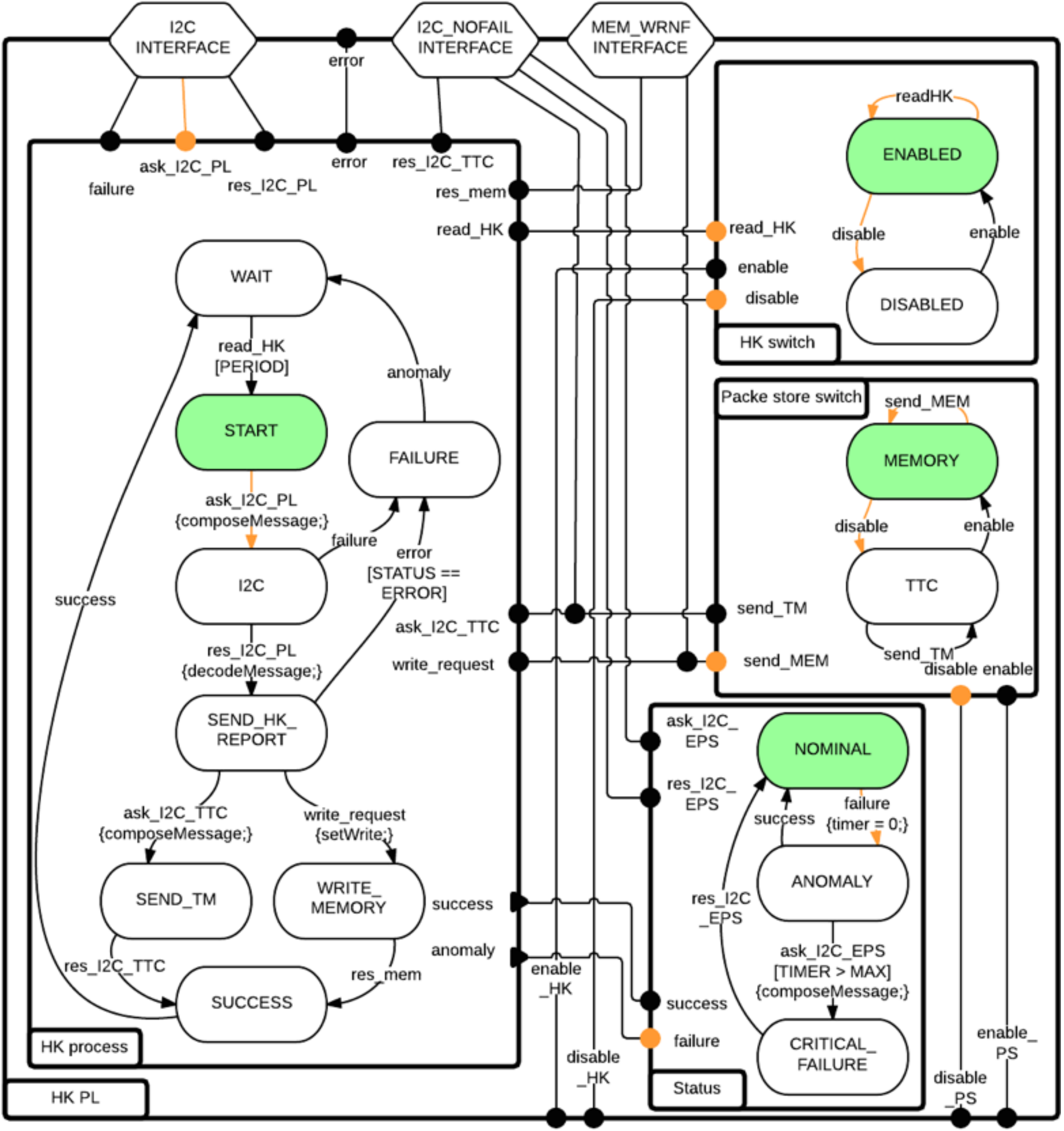
Example 1

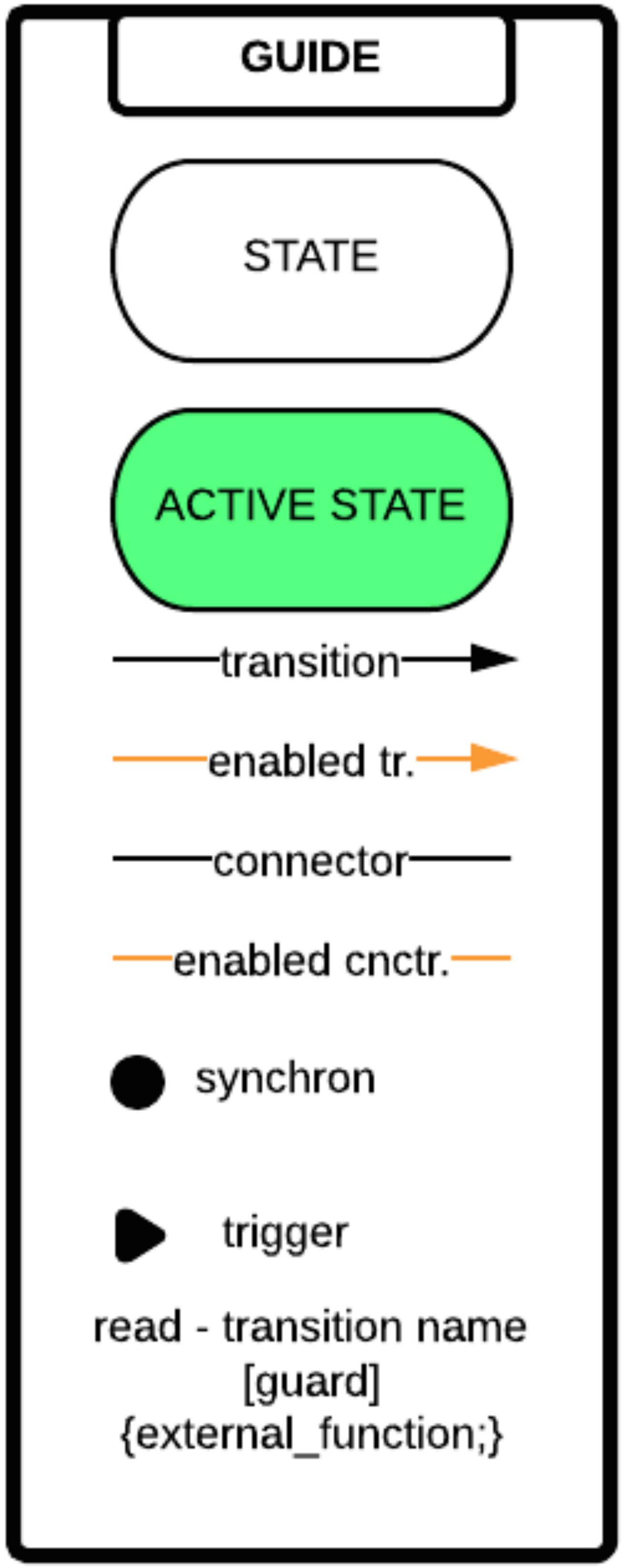
Nominal housekeeping routine



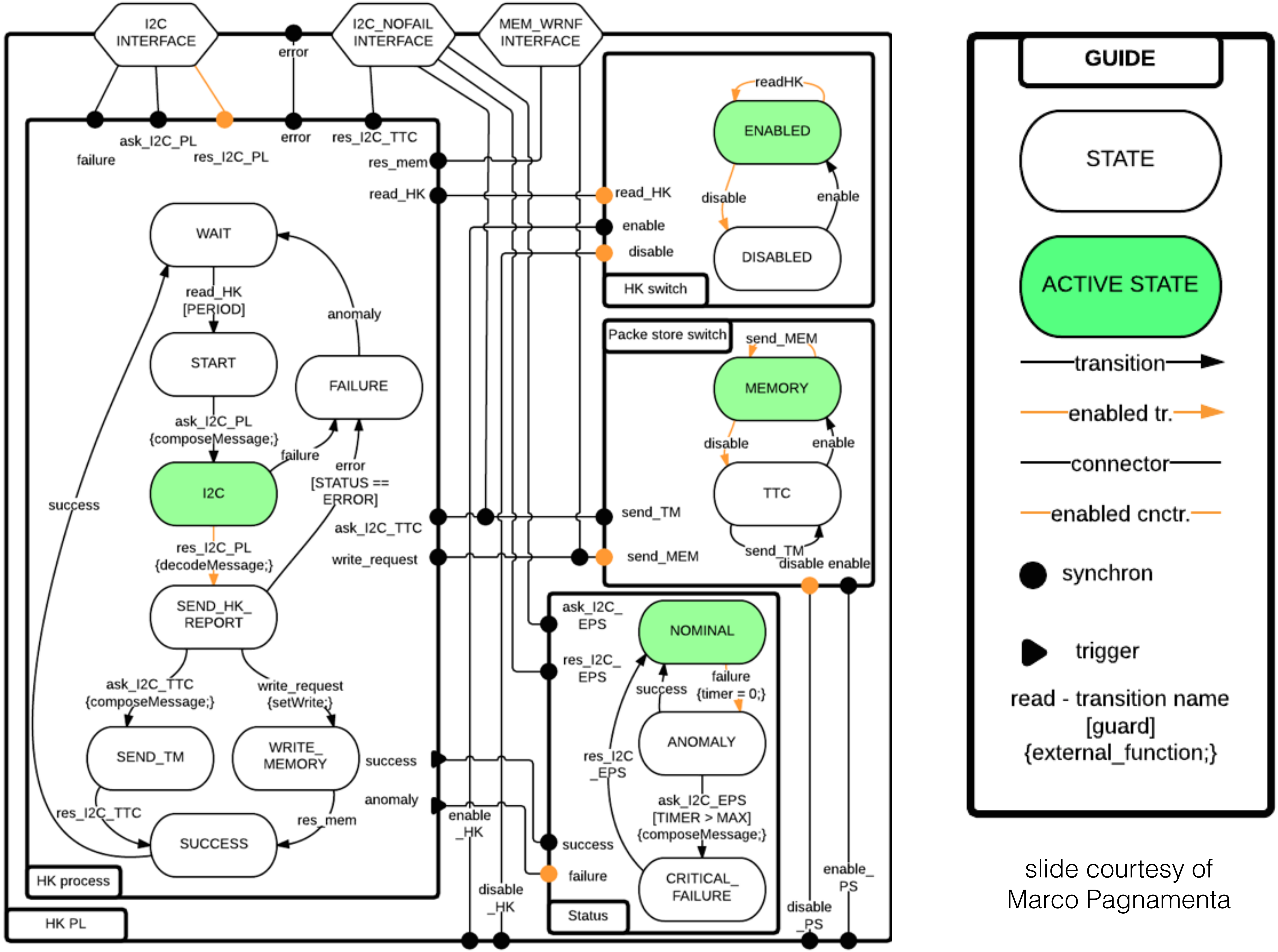


slide courtesy of
Marco Pagnamenta

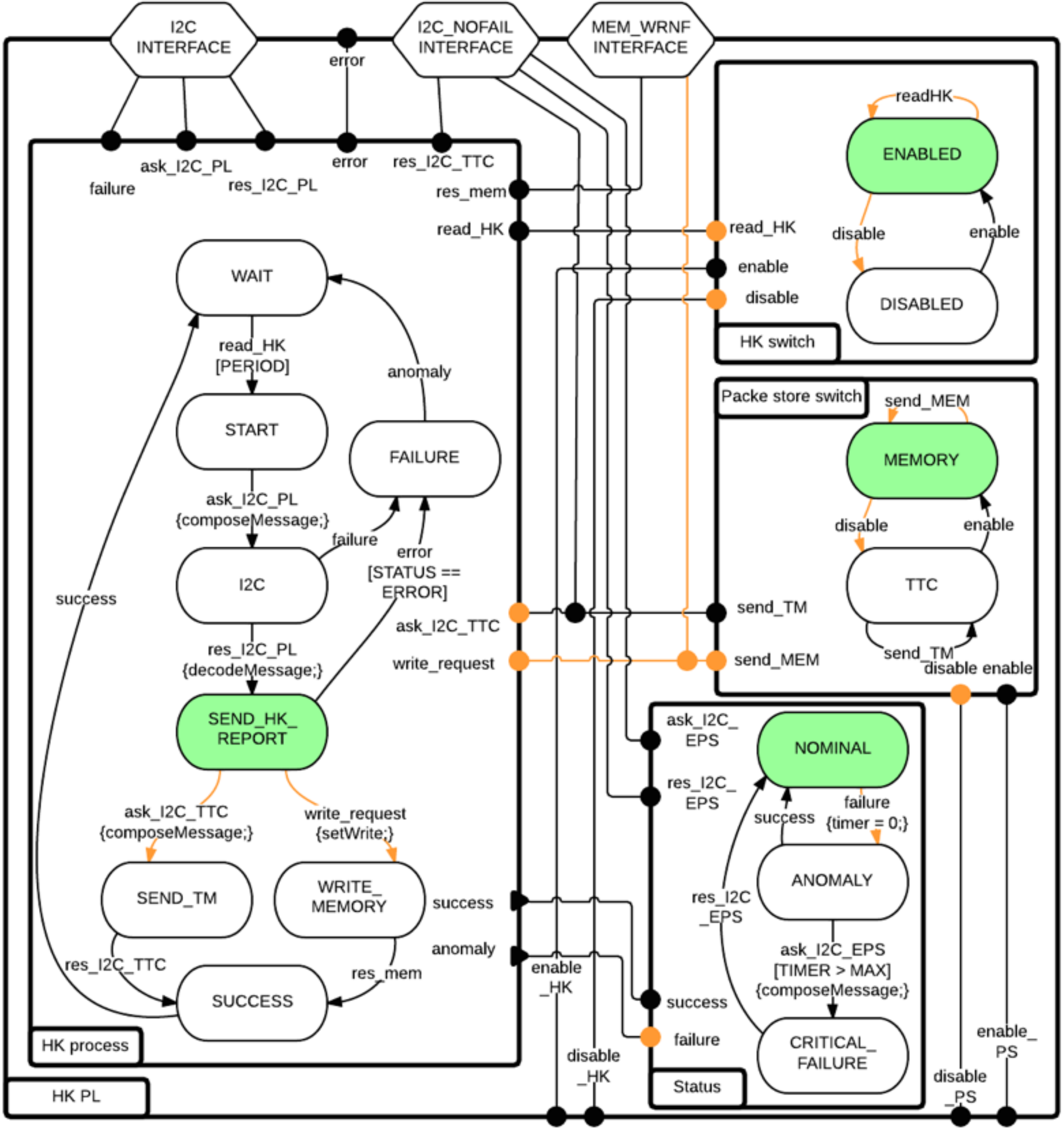


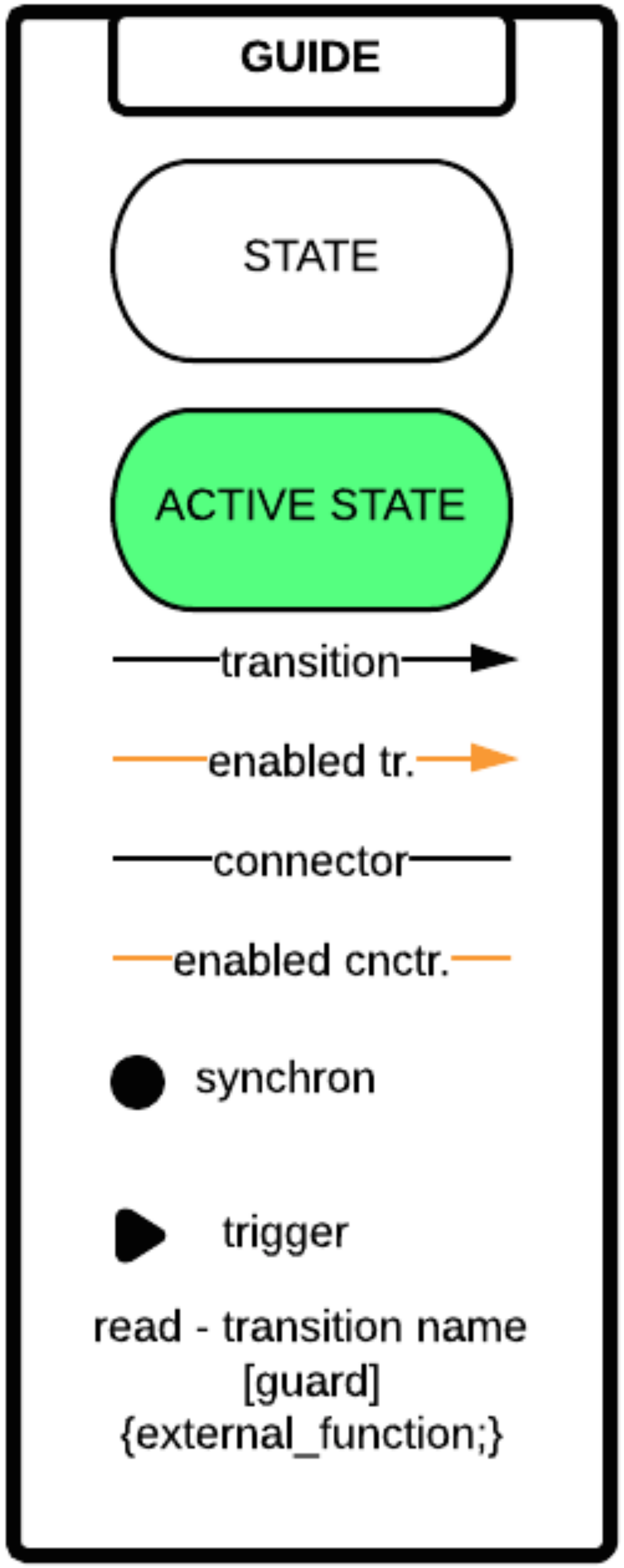


slide courtesy of
Marco Pagnamenta

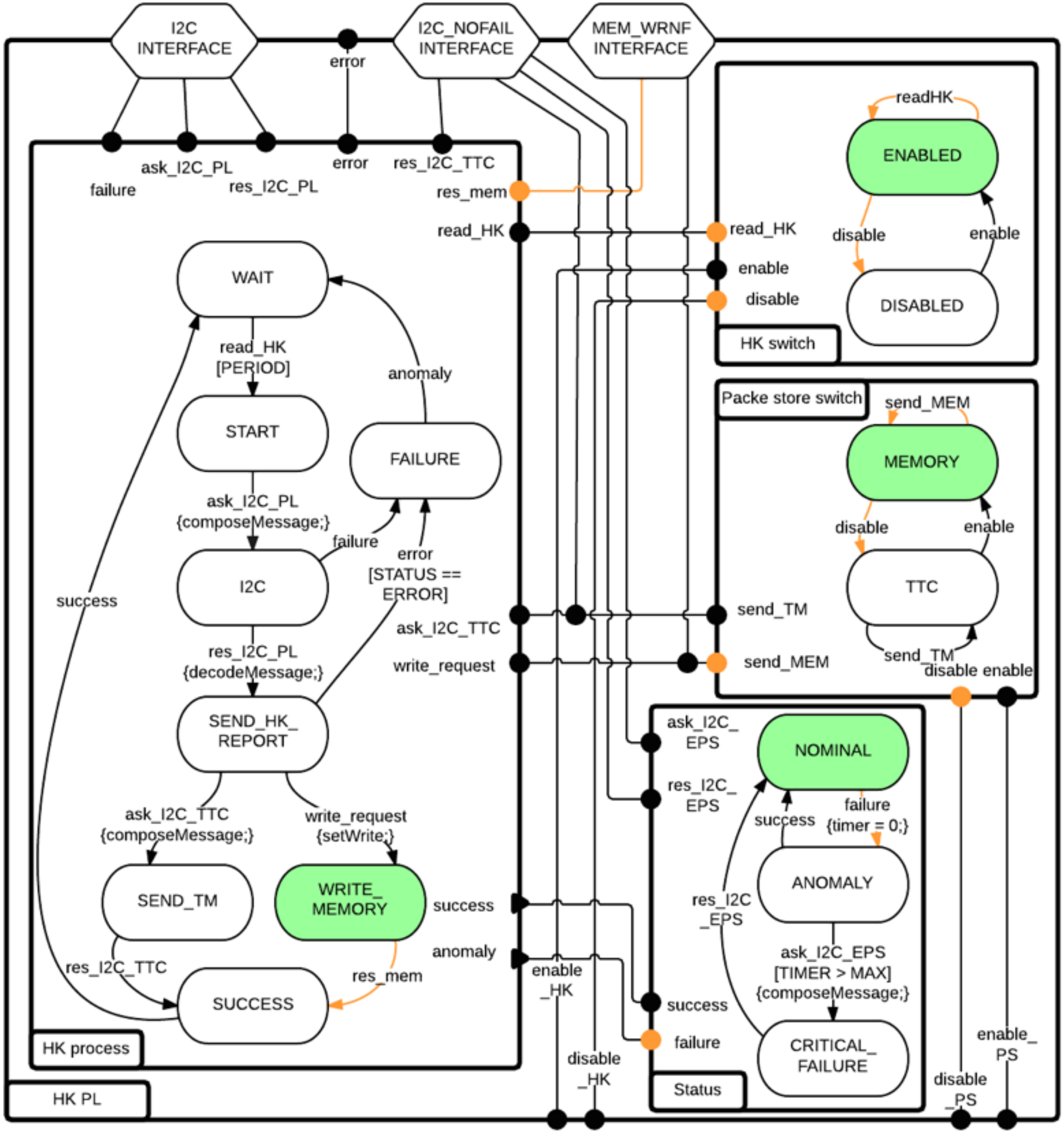


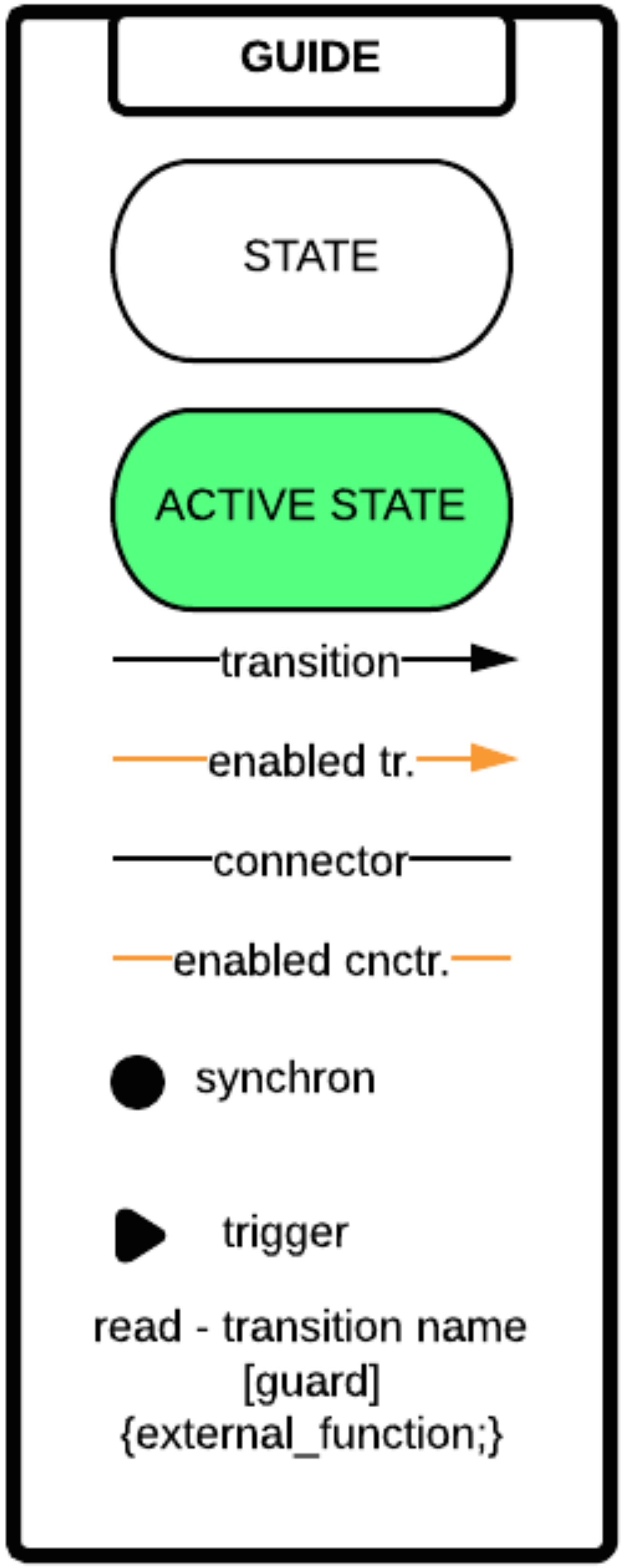
slide courtesy of
Marco Pagnamenta



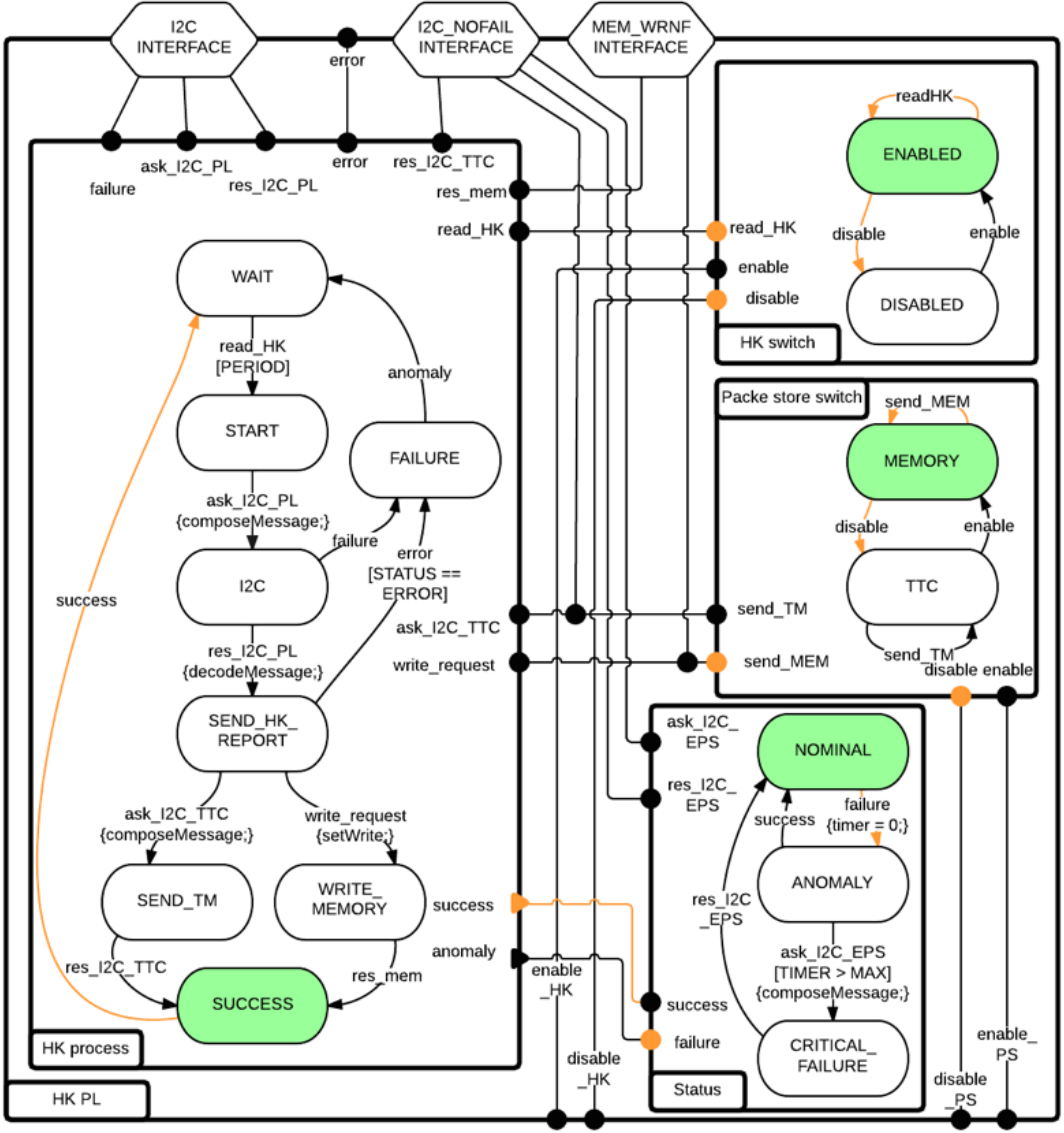


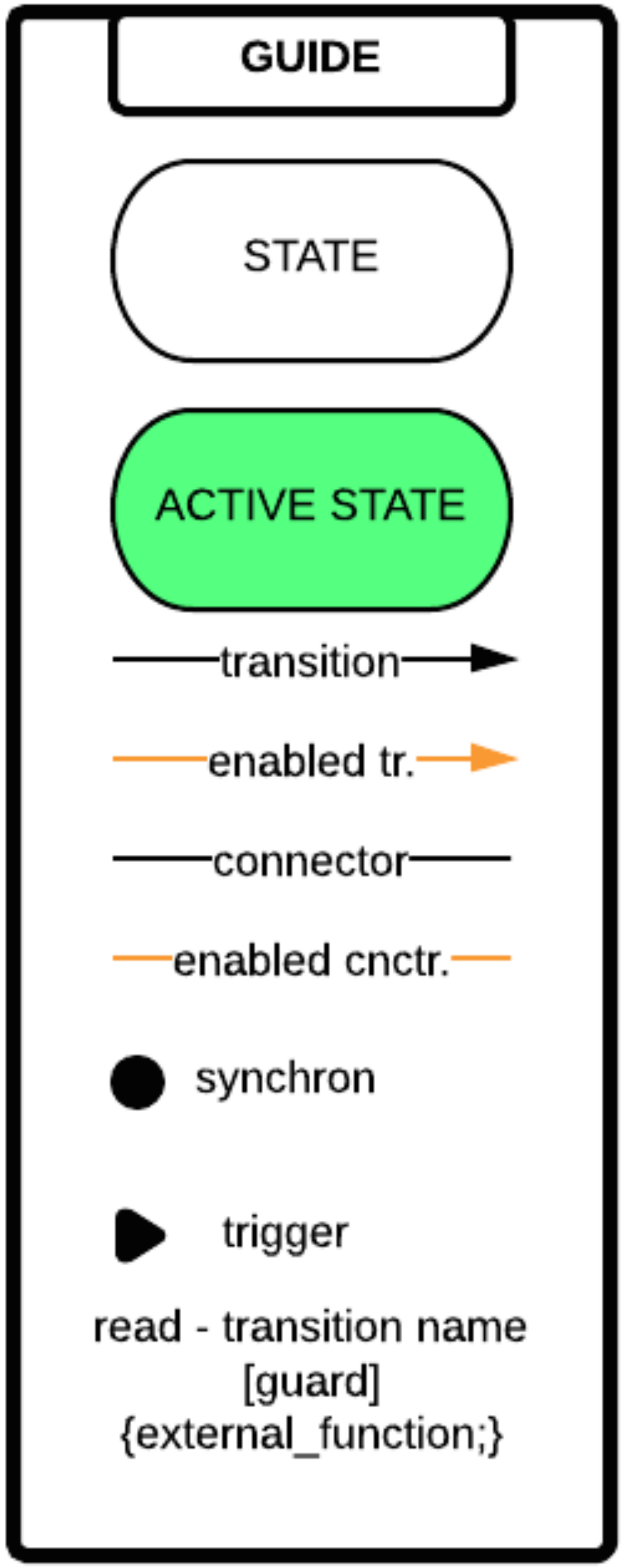
slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

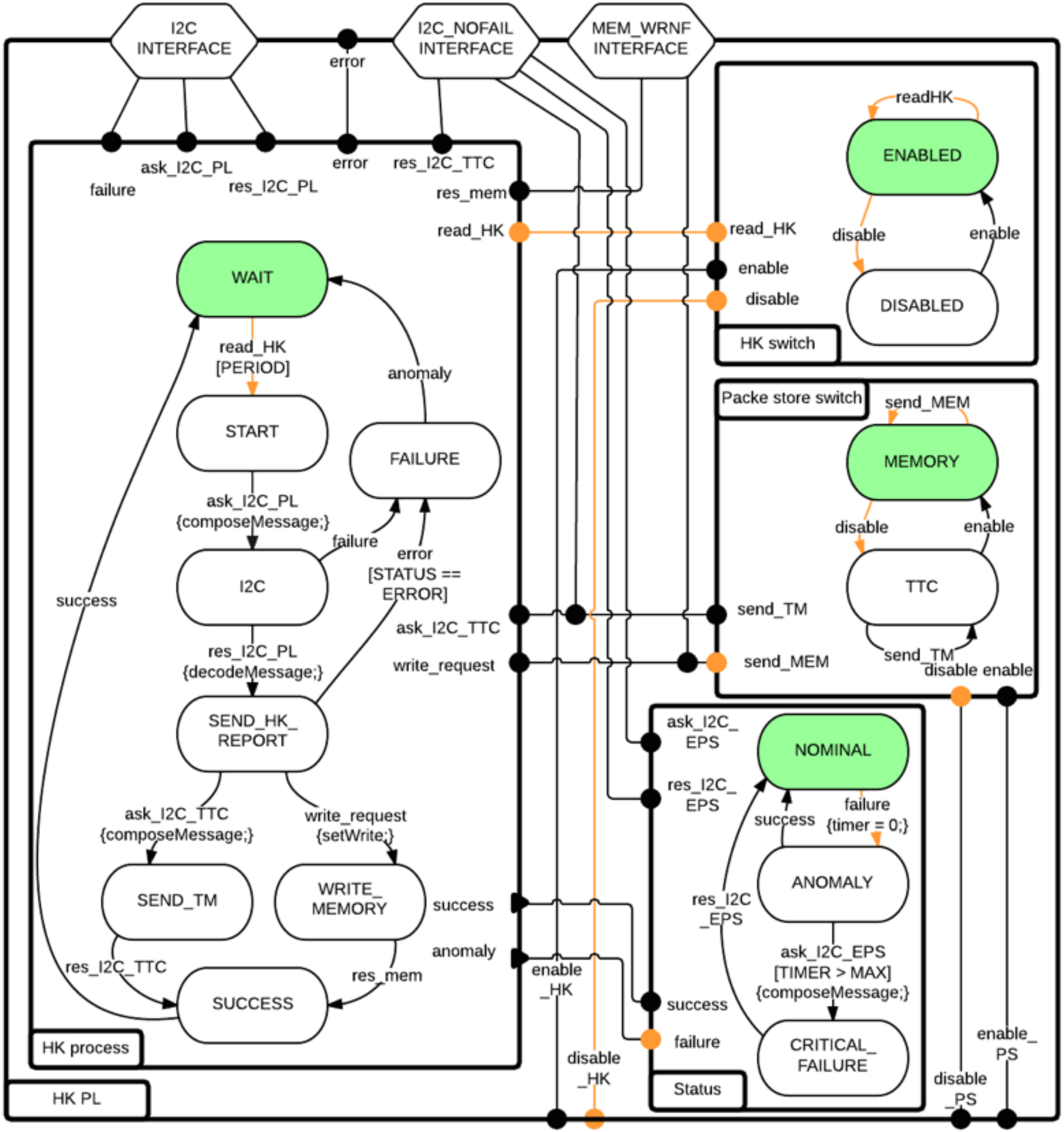


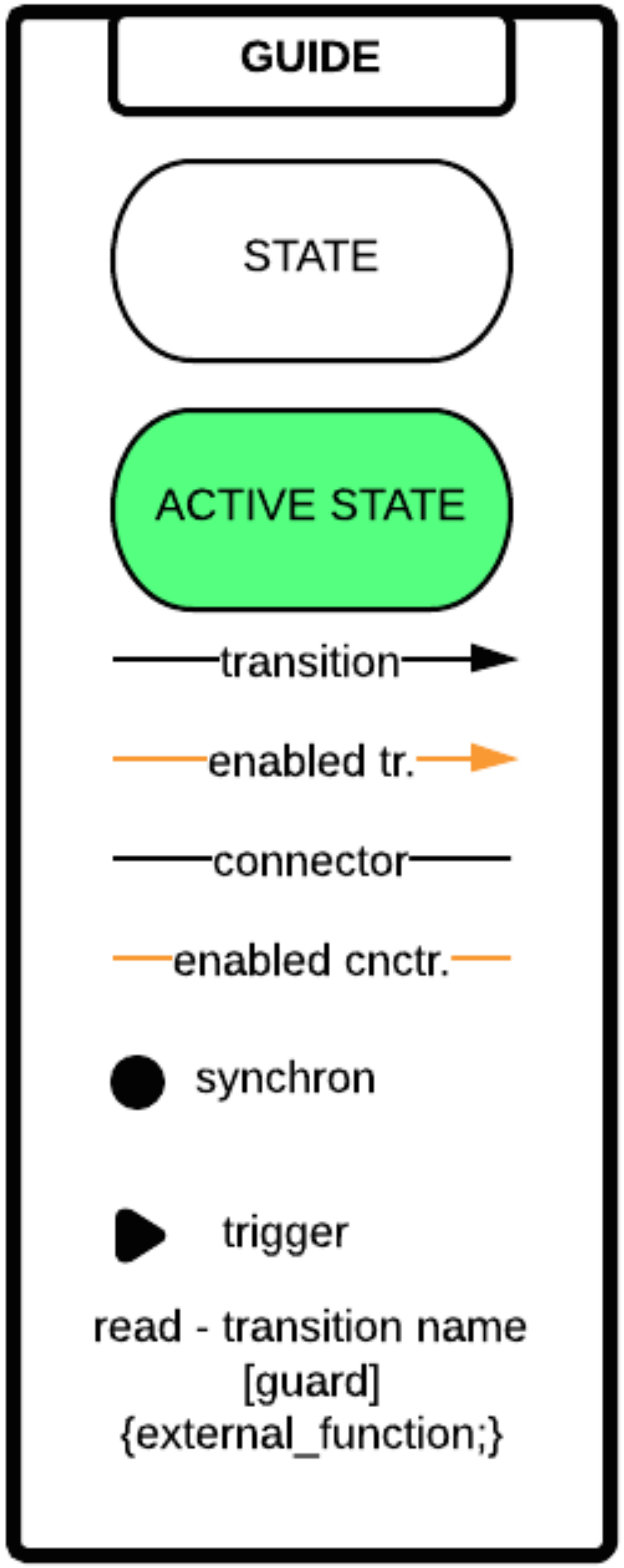


slide courtesy of
Marco Pagnamenta

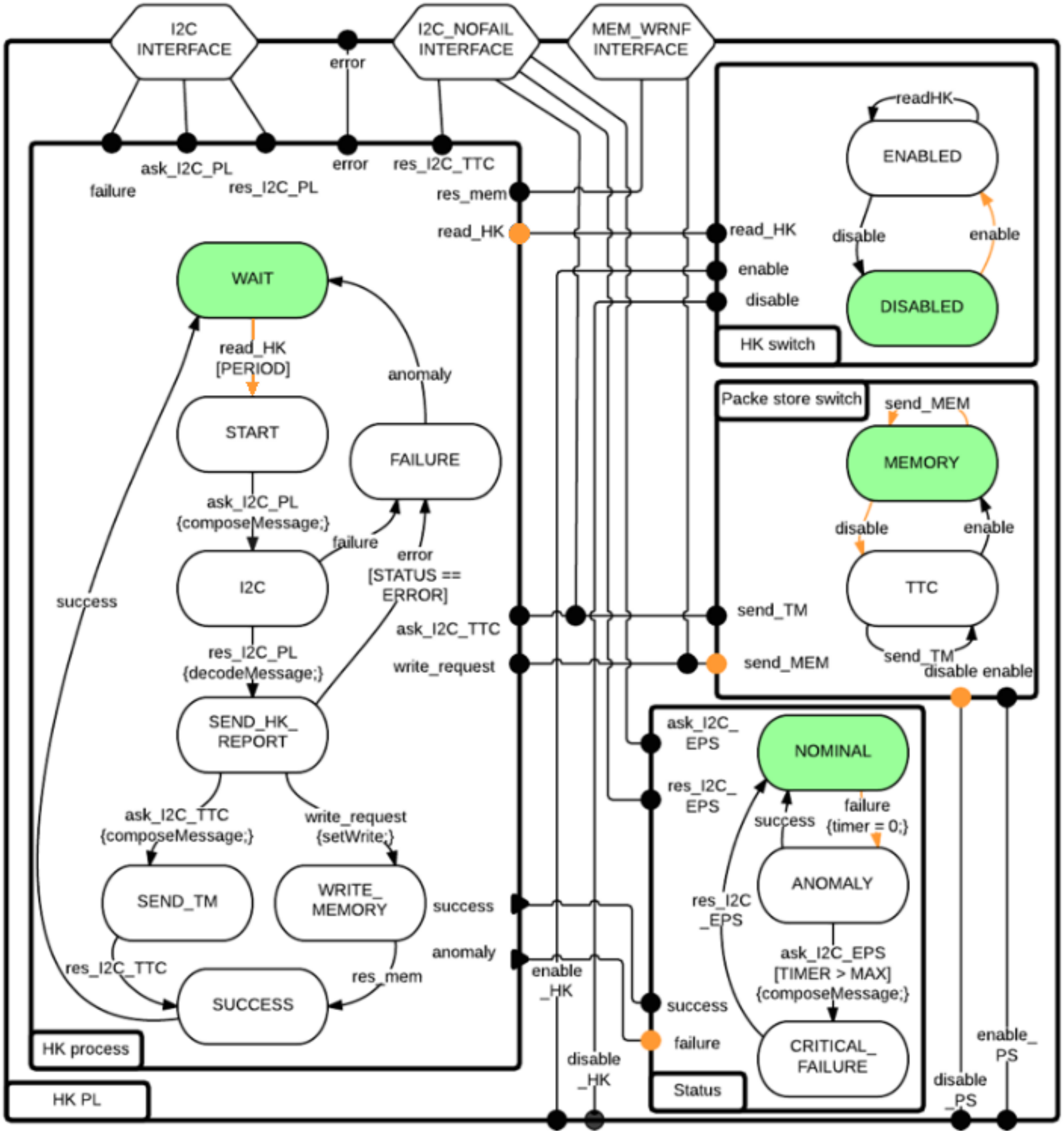
Example 2

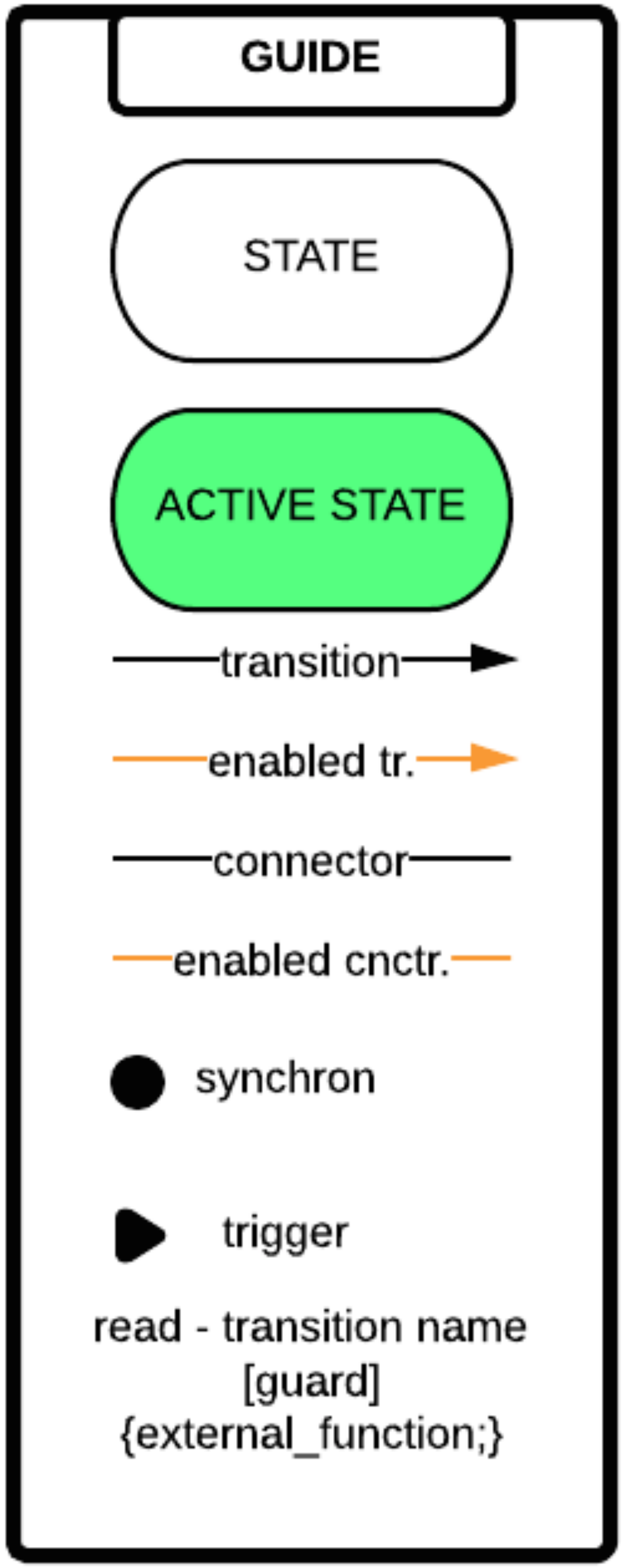
Stopping housekeeping





slide courtesy of
Marco Pagnamenta

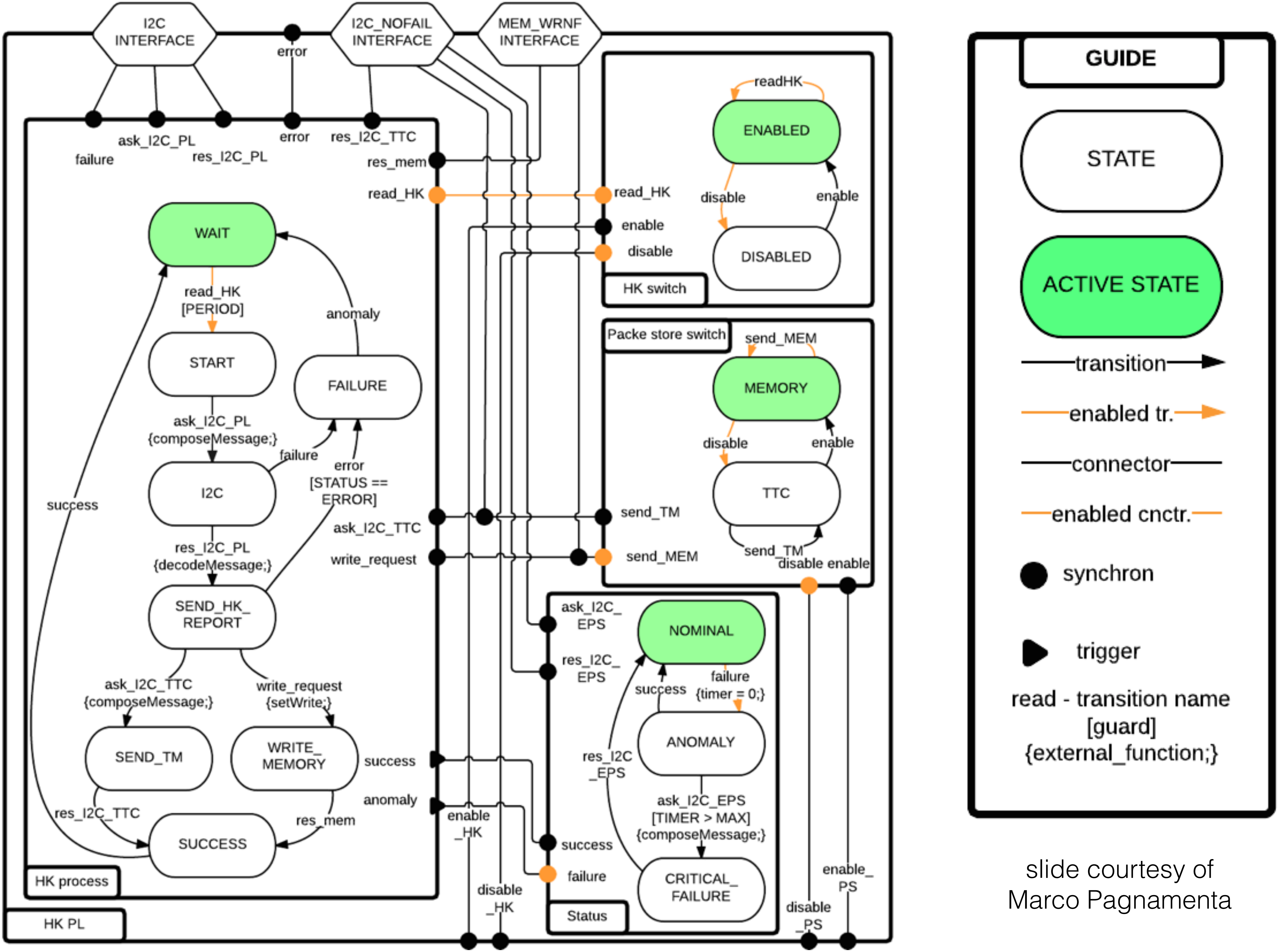




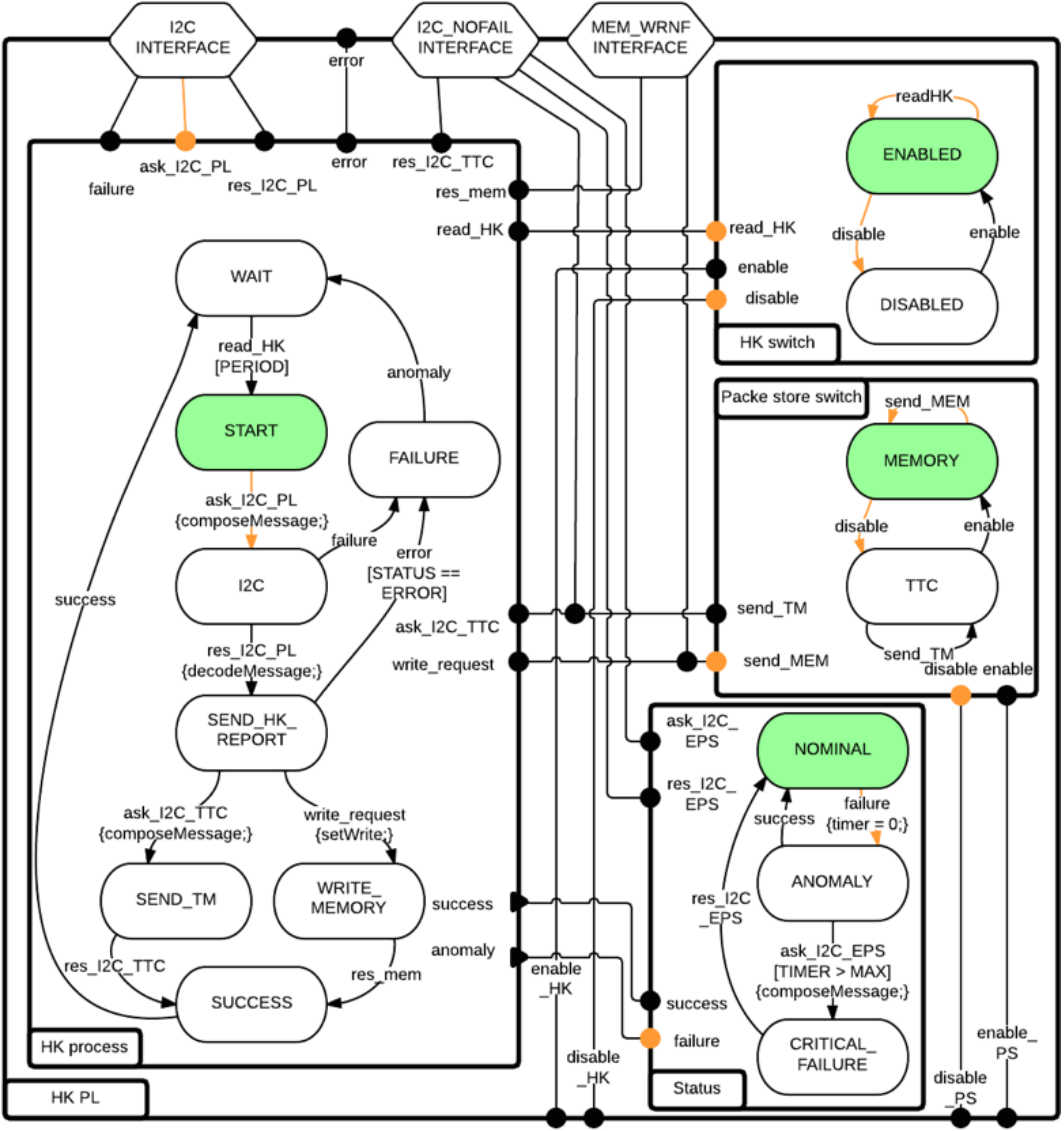
slide courtesy of Marco Pagnamenta

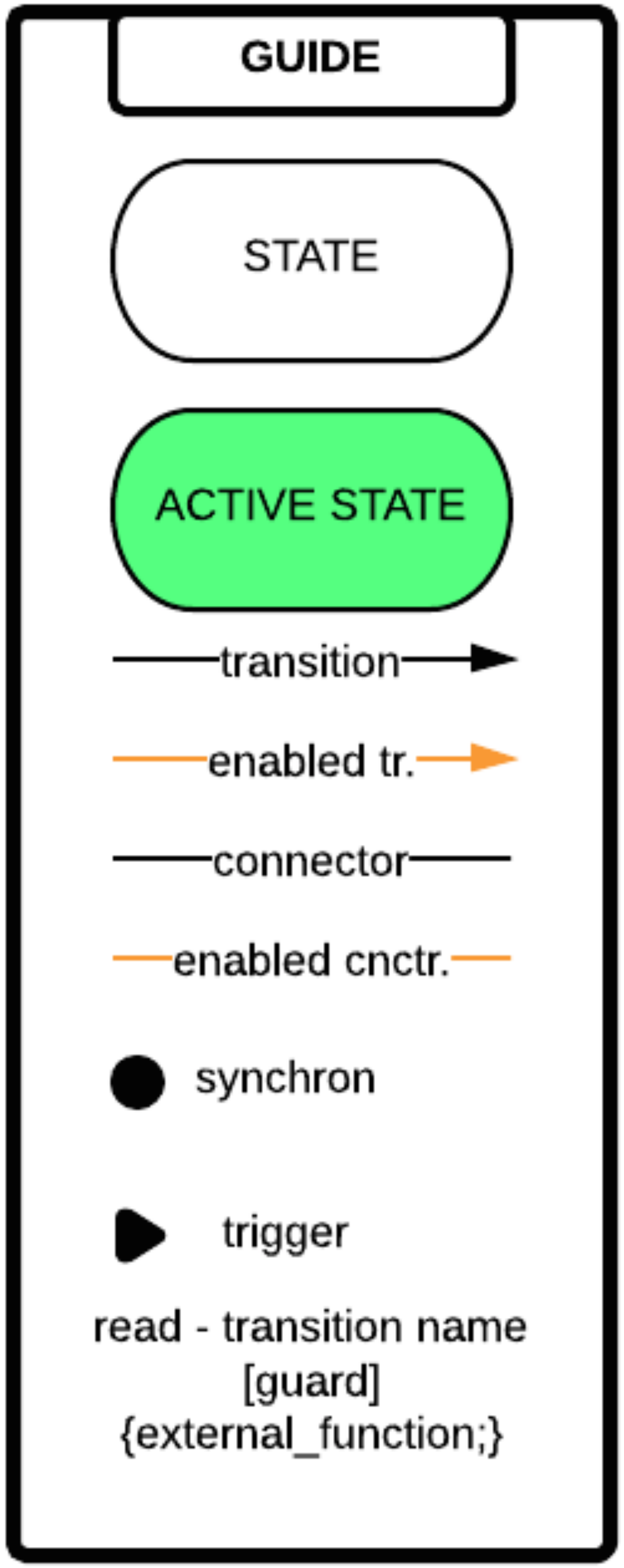
Example 3

Switching destination of housekeeping data

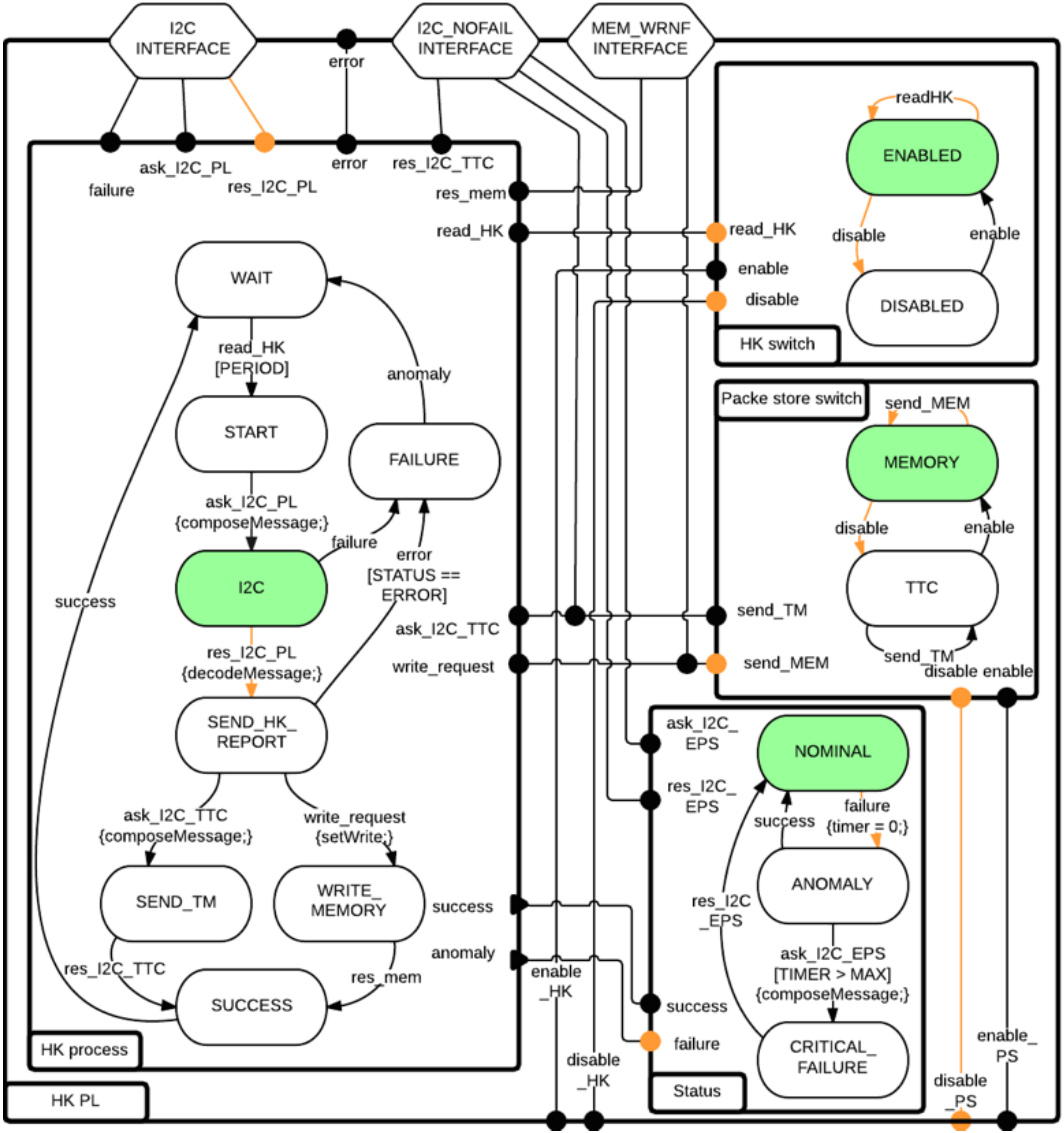


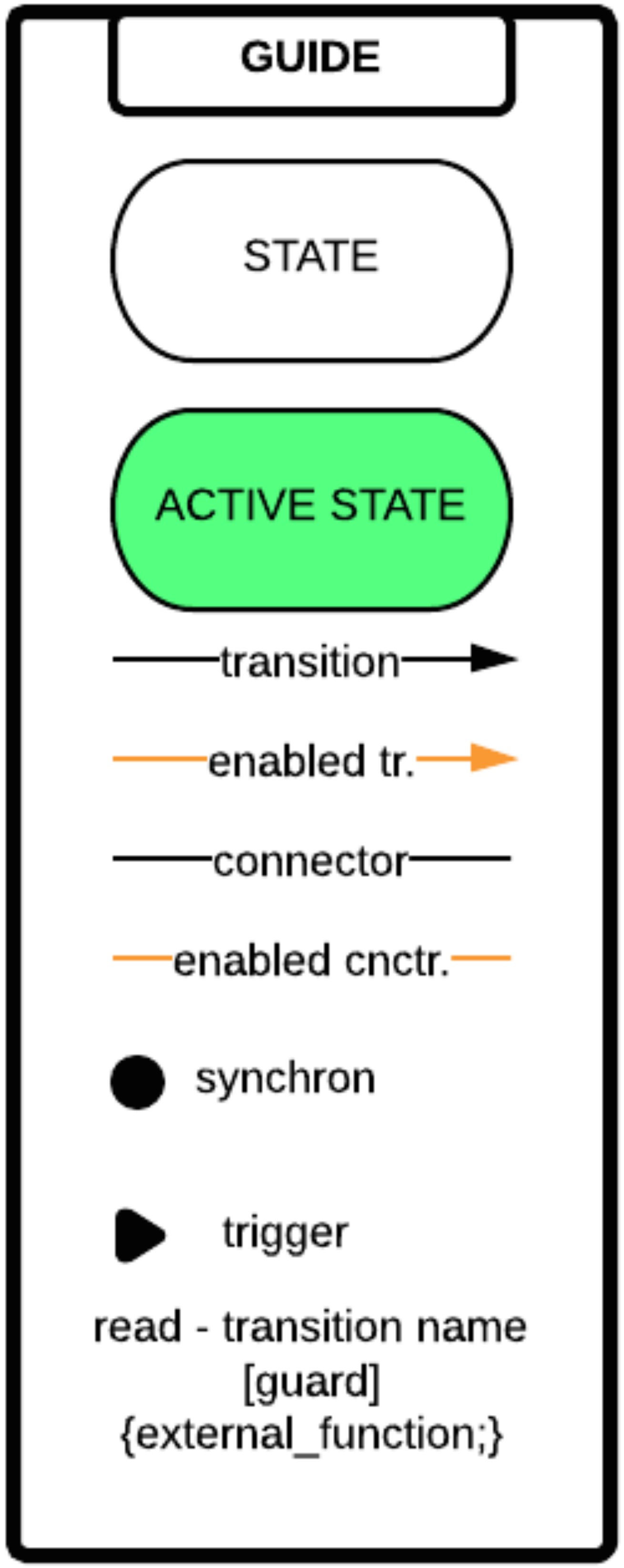
slide courtesy of
Marco Pagnamenta



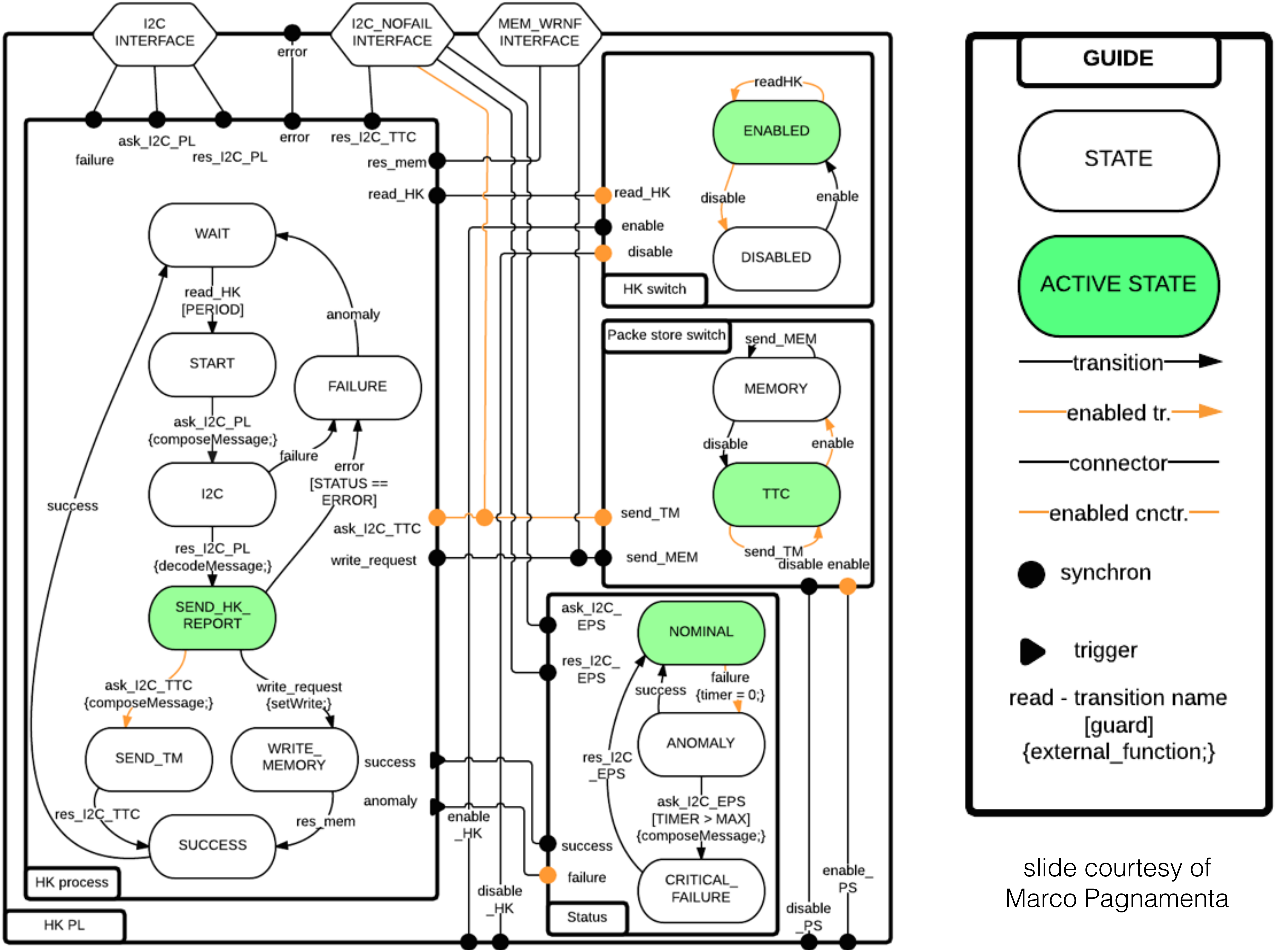


slide courtesy of
Marco Pagnamenta

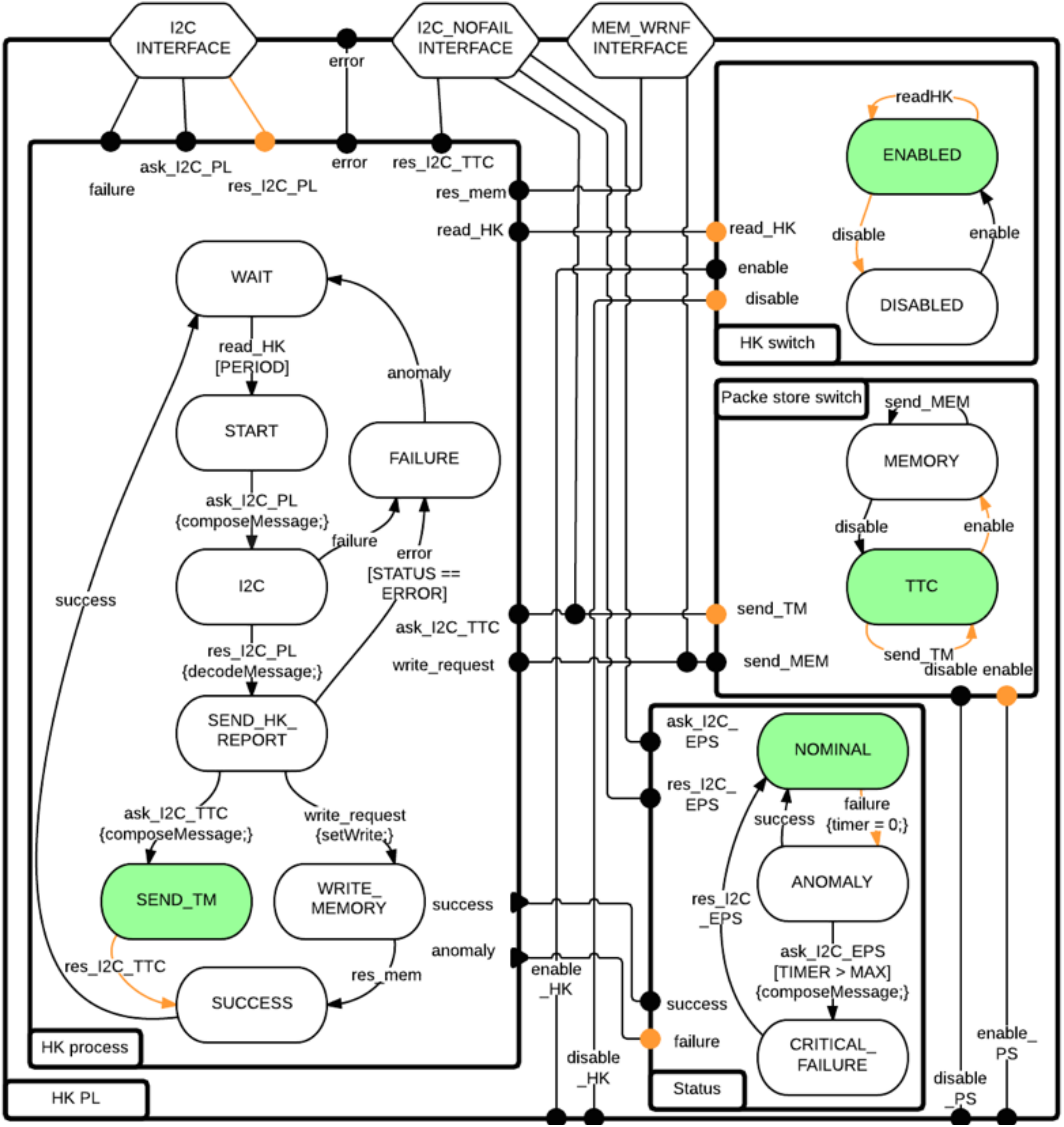


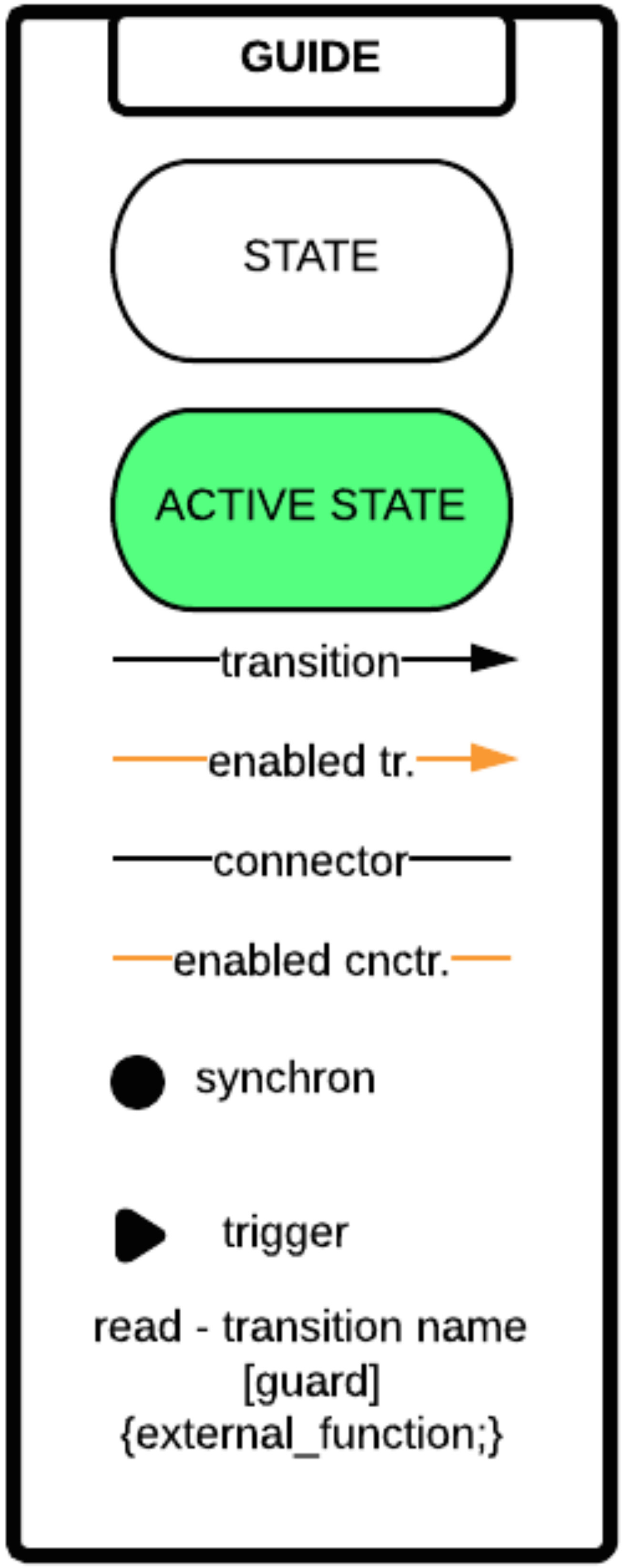


slide courtesy of
Marco Pagnamenta

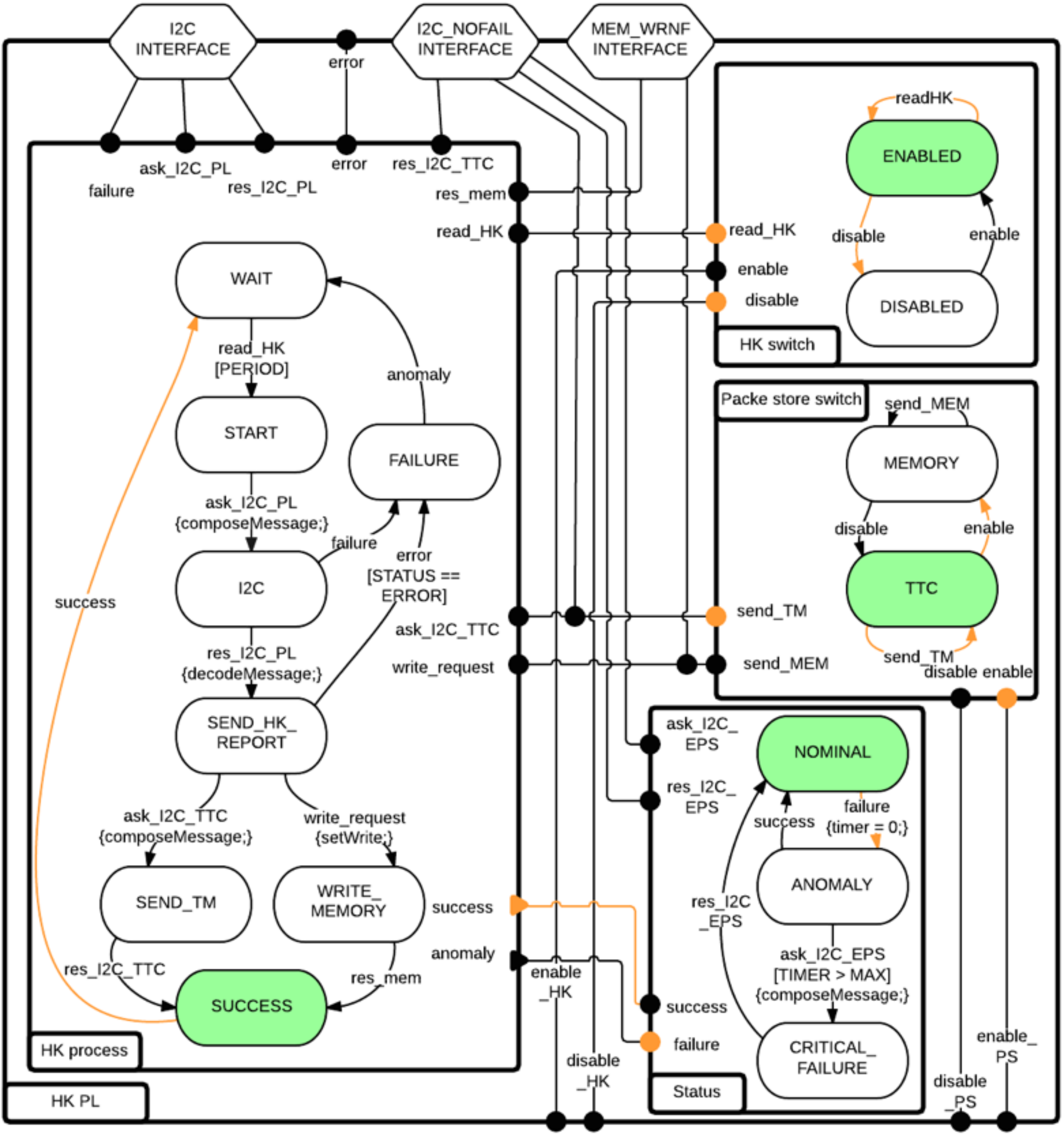


slide courtesy of Marco Pagnamenta





slide courtesy of
Marco Pagnamenta



GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

connector

enabled cnctr.

● synchron

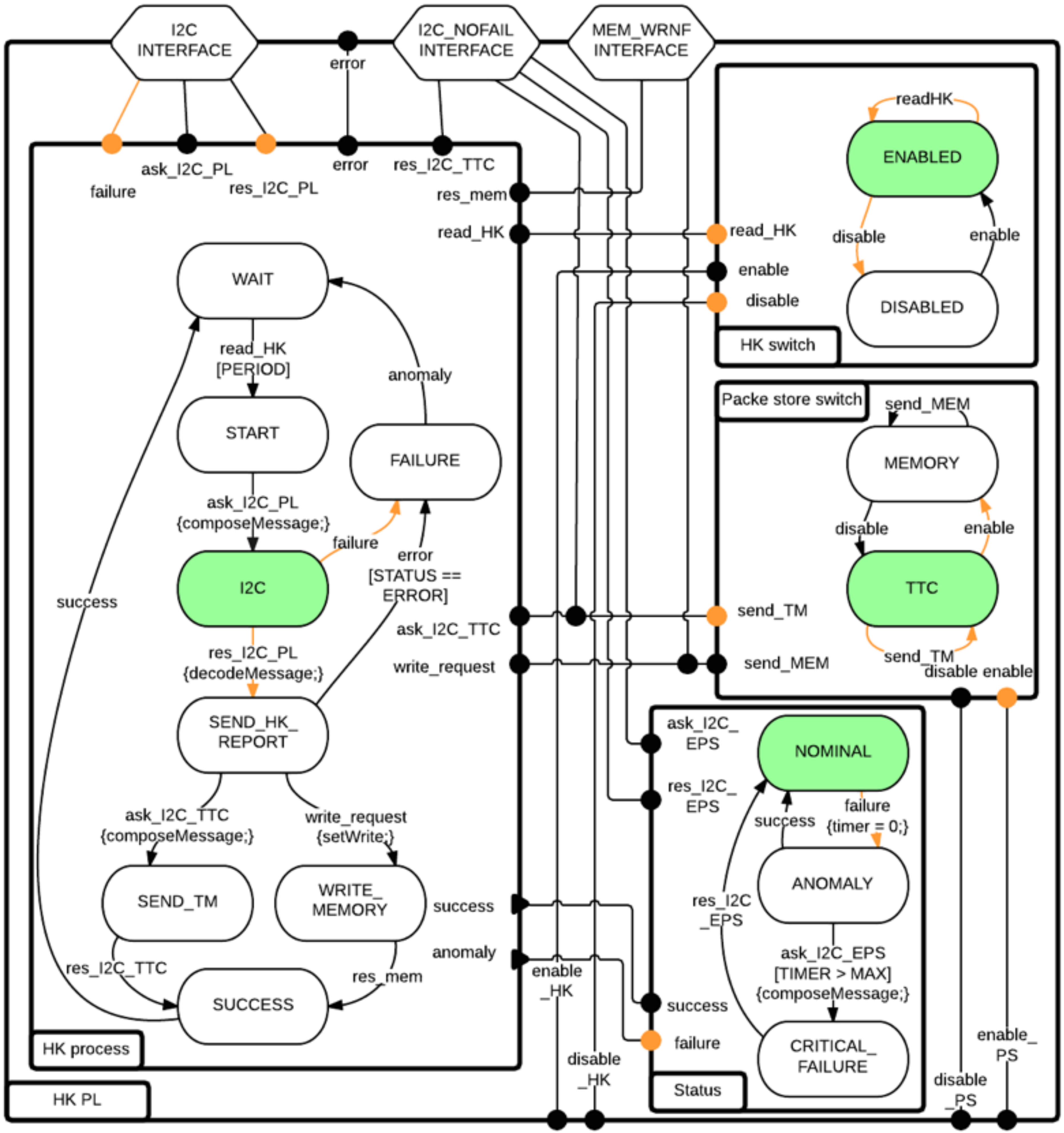
▶ trigger

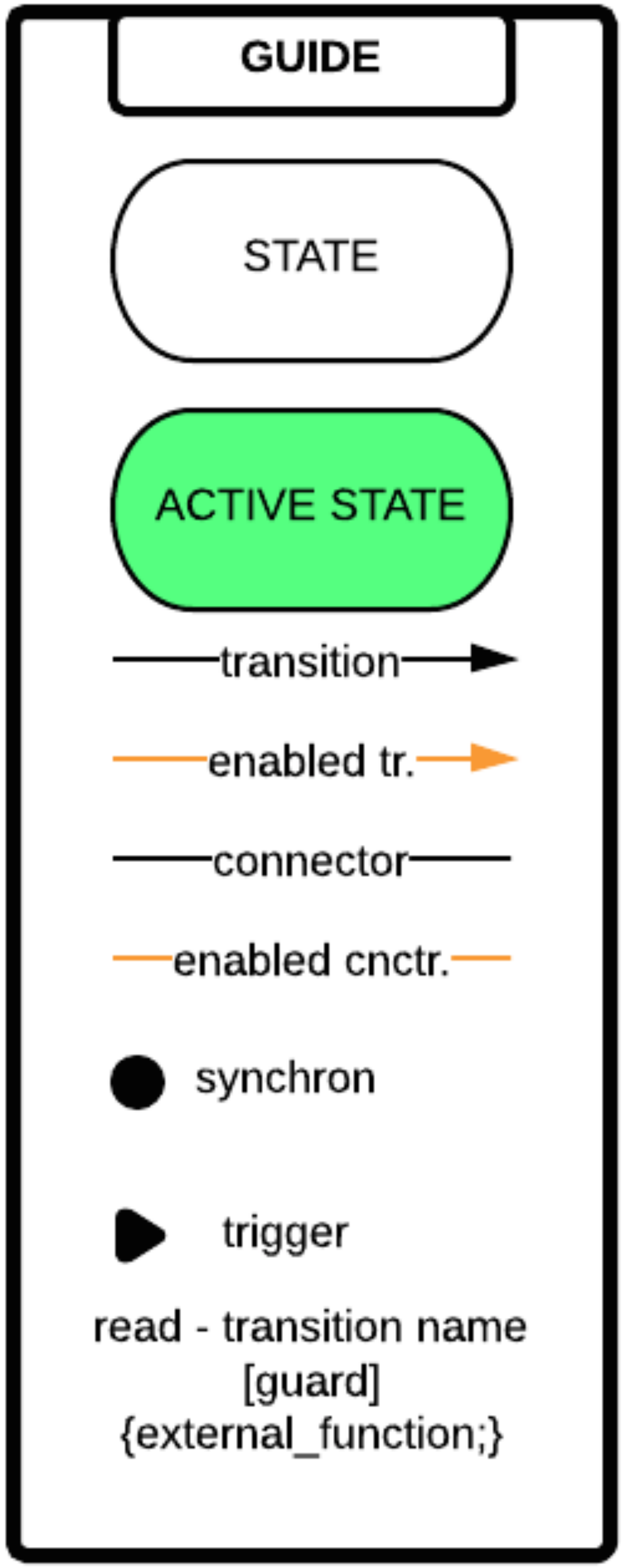
read - transition name
[guard]
{external_function;}

slide courtesy of
Marco Pagnamenta

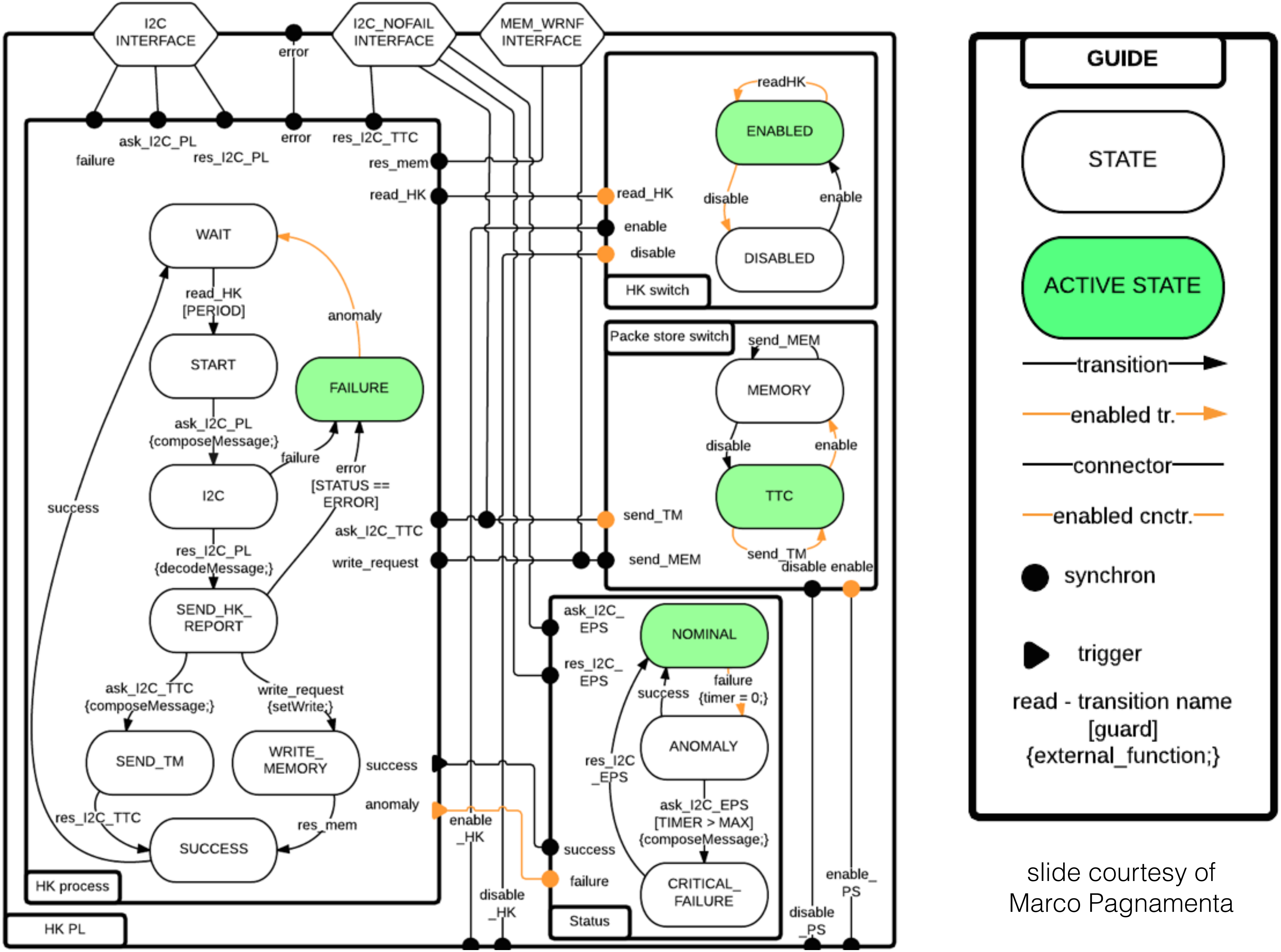
Example 4

I^{2}C bus failure management

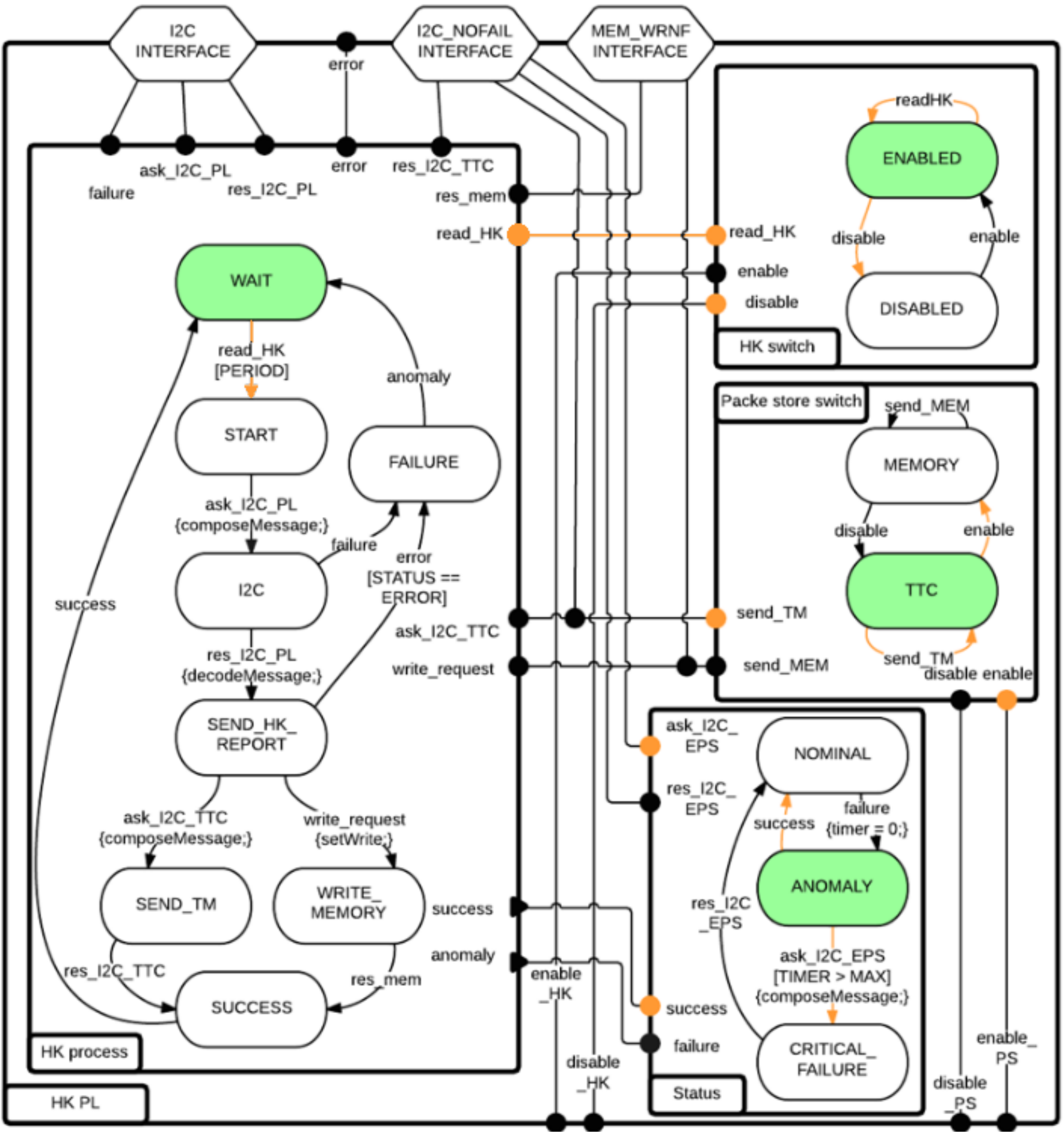


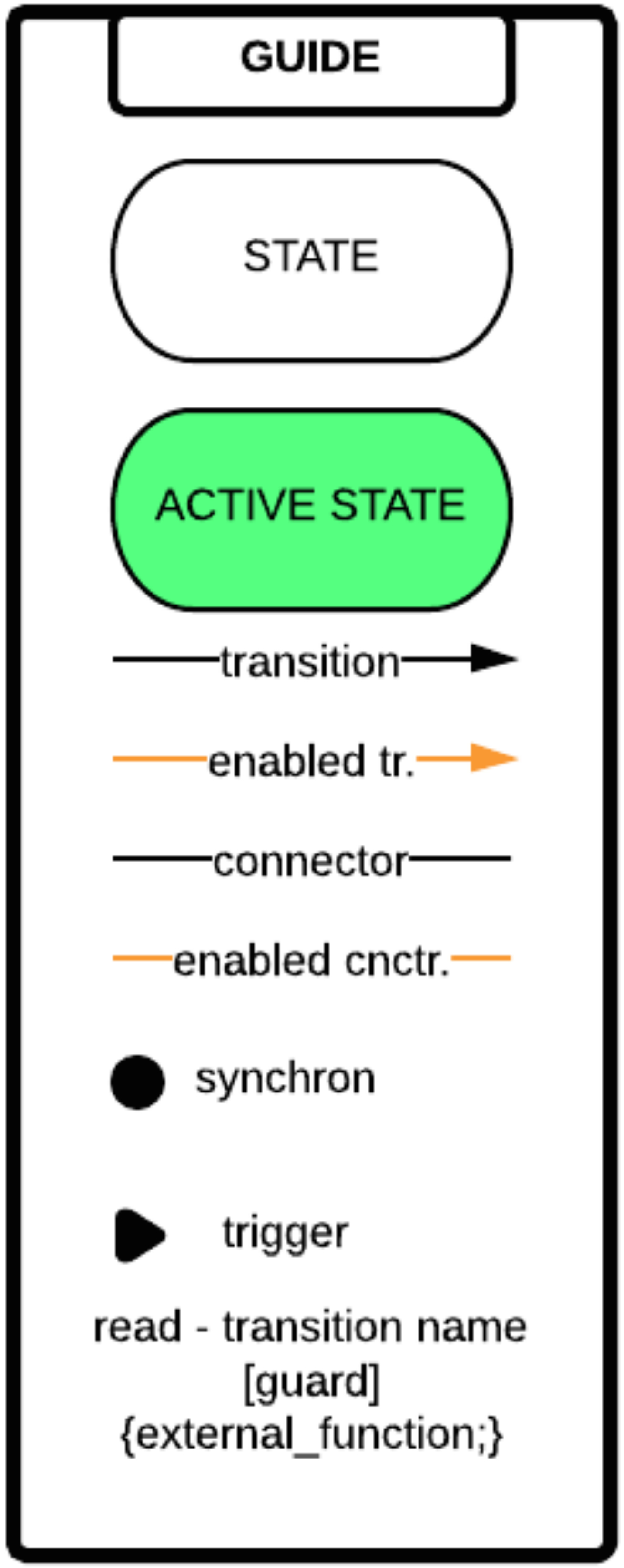


slide courtesy of
Marco Pagnamenta



slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

Sem

of semicolon.

se•man•tic (si man

to meaning or arising from

symbols: semantic change; semantic

semantics. [1655–65; < Gk sēman

semant(ós) marked (sēman-, base o

verbal adj. suffix; akin to sēma si

se•man•tics (si man′tiks), n. (u

linguistics dealing with the stu

meaning is structured in langua

over time. 2. the branch of se

tionship between signs or sy

meaning. or an interpretation

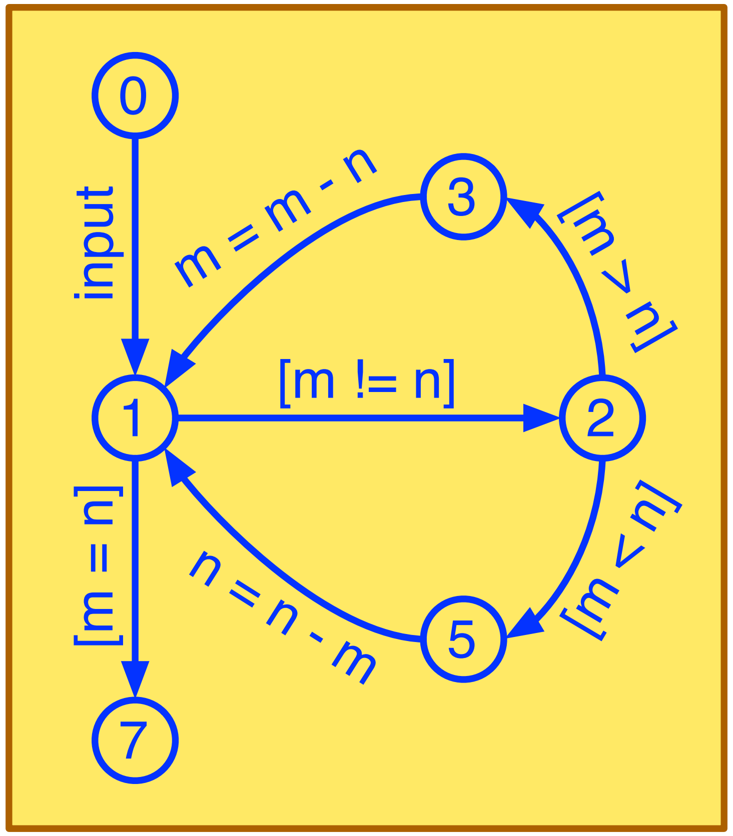
ence, etc.: Let's not argue

1895–1900] —se•man′ti•c

m), n.

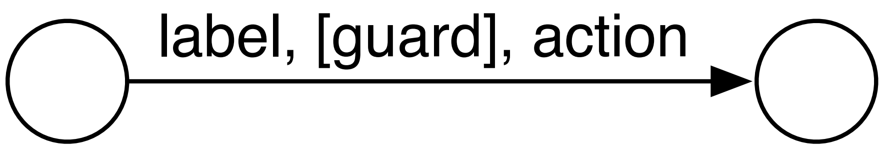
Components

```
0: input(m,n>0);
1: while(m != n){
2:  if (m > n)
3:    m = m - n;
4:  else //m < n
5:    n = n - m;
6: }
7: //m=n=gcd(m,n)
```

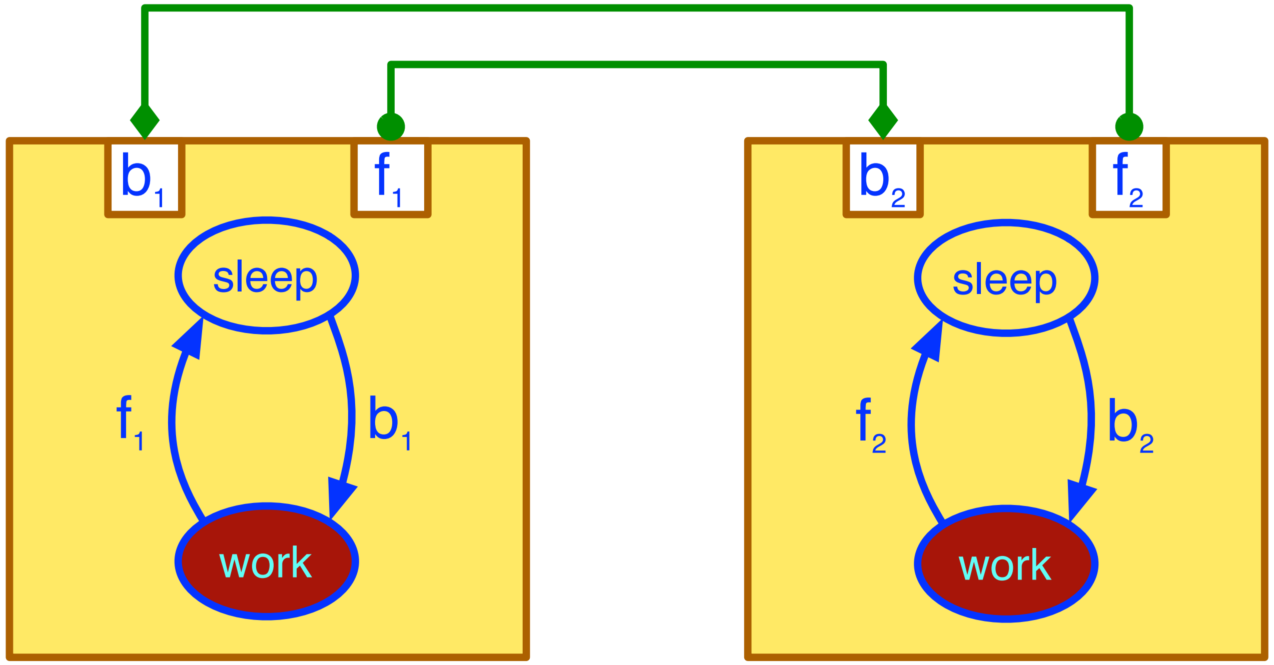


- There is a canonical transformation
  - The choice of abstraction level is important

- Taking a transition
  1. is allowed if the guard evaluates to true
  2. executes the action
  3. updates current state



BIP by example: Mutual exclusion



Interaction model:

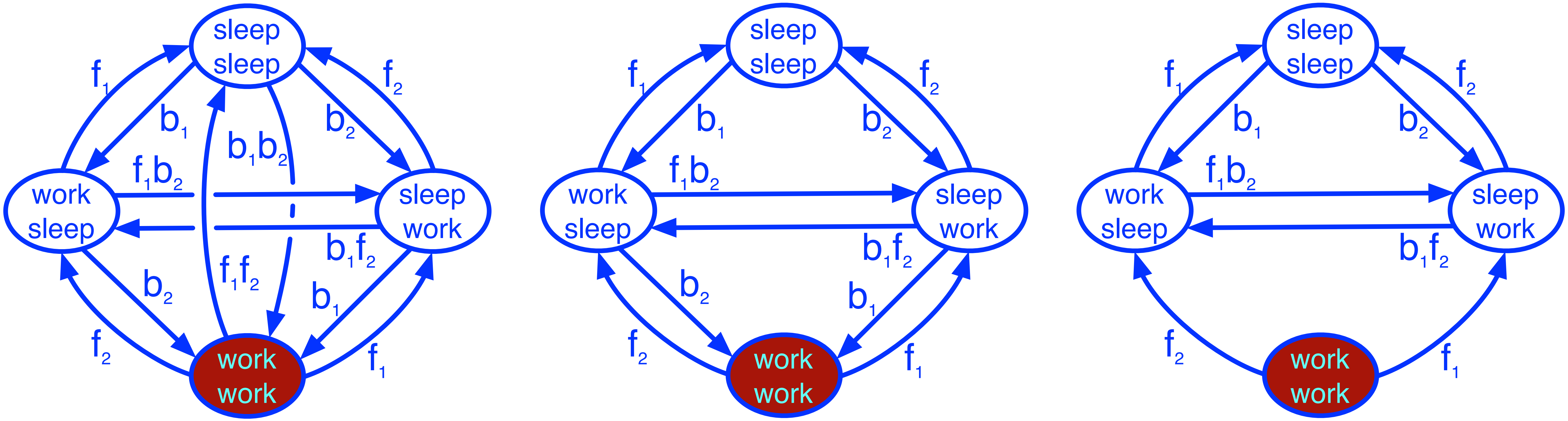
$$\{b_1,\ f_1,\ b_2,\ f_2,\ b_1 f_2,\ b_2 f_1\}$$

Maximal progress:

$$b_1 < b_1 f_2,\ b_2 < b_2 f_1$$

Design view

Semantic view



Semantics: Interactions

Consider a set of *n* behaviours, such that

$$B_i = (Q_i, P_i, \to_i), \qquad \to_i \subseteq Q_i \times 2^{P_i} \times Q_i, \qquad P = \dot{\bigcup_i} P_i$$

Interaction model: $\gamma \subseteq 2^P$ — a set of allowed interactions

$$\frac{q_i \xrightarrow{a \cap P_i} q_i' \ (\text{if } a \cap P_i \neq \emptyset) \quad q_i = q_i' \ (\text{if } a \cap P_i = \emptyset)}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}$$

for each $a \in \gamma$.

Semantics: Priority

$$B_i = (Q_i, P_i, \rightarrow_i), \qquad \rightarrow_i \subseteq Q_i \times 2^{P_i} \times Q_i, \qquad P = \dot{\bigcup_i} P_i$$

Interaction model: $\gamma \subseteq 2^P$ — a set of allowed interactions

$$\frac{q_i \xrightarrow{a \cap P_i} q_i' \;(\text{if } a \cap P_i \neq \emptyset) \quad q_i = q_i' \;(\text{if } a \cap P_i = \emptyset)}{q_1 \dots q_n \overset{a}{\rightarrow} q_1' \dots q_n'}$$

for each $a \in \gamma$.

Priority model: $\prec \subseteq 2^P \times 2^P$ — strict partial order

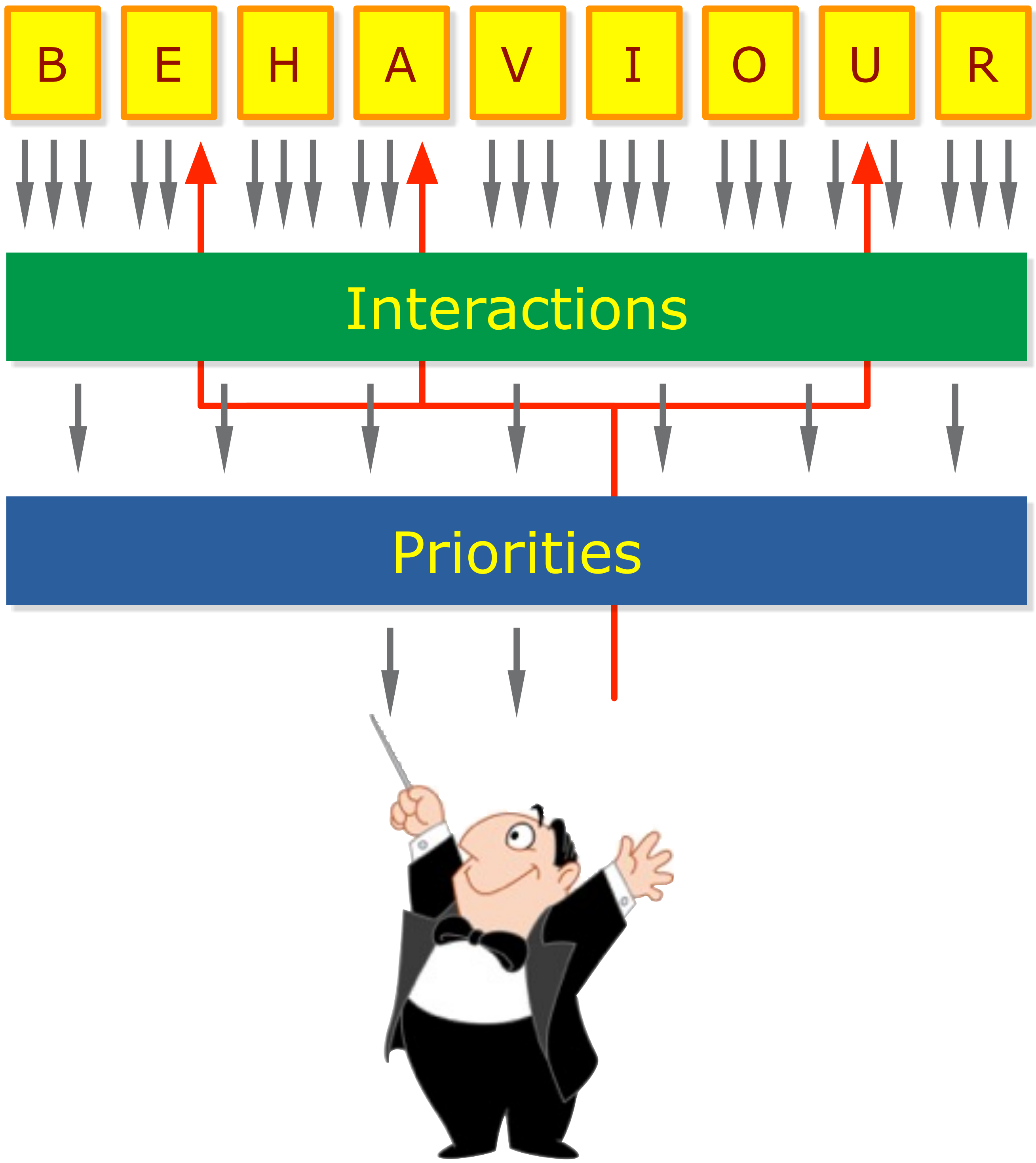
$$\frac{q \xrightarrow{a} q' \qquad \forall a \prec a',\; q \overset{a'}{\not\longrightarrow}}{q \xrightarrow{a}_{\prec} q'}$$

for each $a \in 2^P$.

Engine-based execution

1. Components notify the Engine about enabled transitions.

2. The Engine picks an interaction and instructs the components.

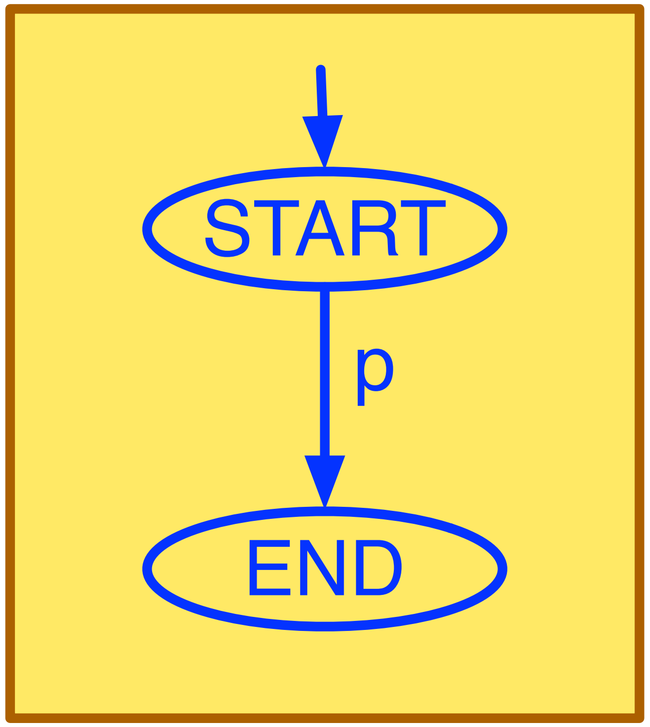


Hands-on BIP

Safe control layer of a Rescue robot

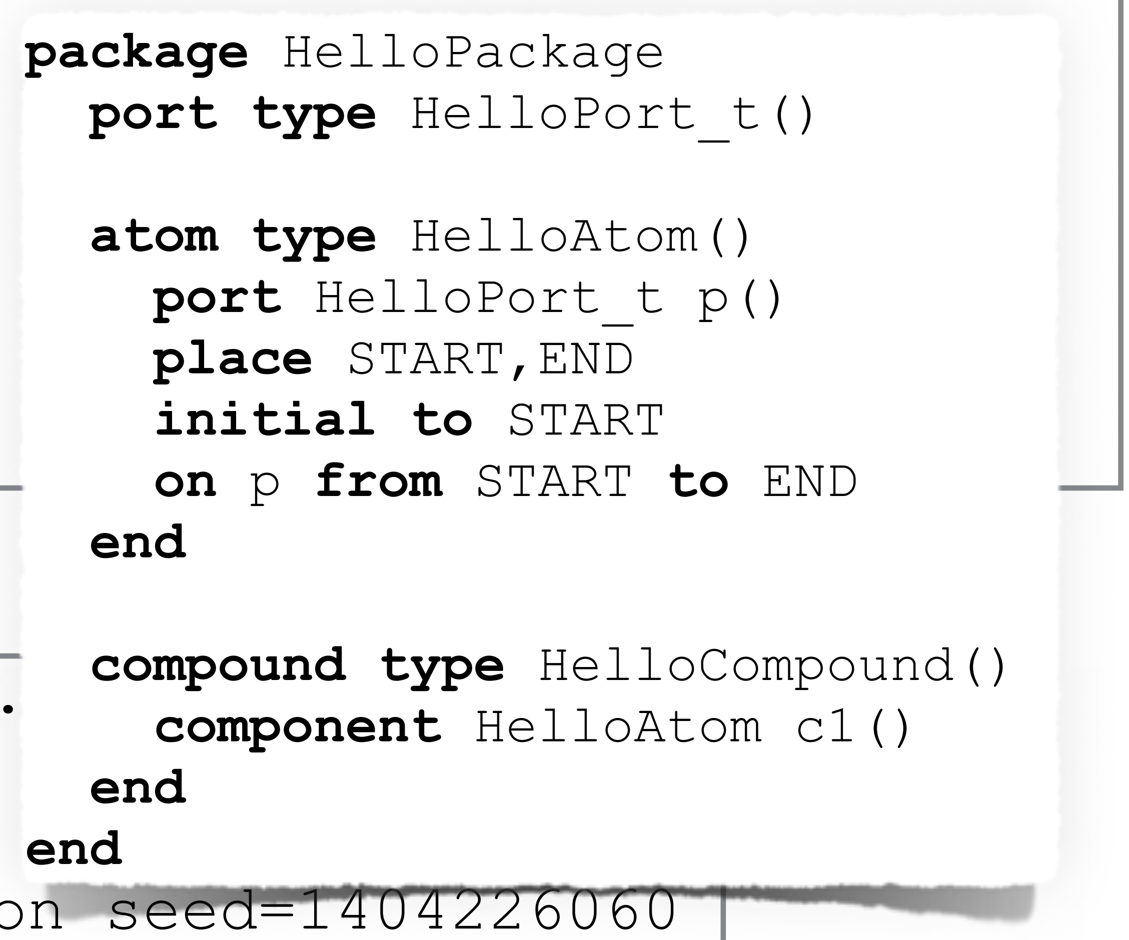
Hello World

```
package HelloPackage
  port type HelloPort_t()

  atom type HelloAtom()
    port HelloPort_t p()

    place START,END

    initial to START
    on p from START to END
  end

  compound type HelloCompound()
    component HelloAtom c1()
  end
end
```



Hello World

```
$ bipc.sh -I . -p HelloPackage -d "HelloCompound()" \
     --gencpp-output output
$ cd build
$ cmake ../output
$ make
$ ./system
```



```
package HelloPackage
  port type HelloPort_t()

  atom type HelloAtom()
    port HelloPort_t p()
    place START,END
    initial to START
    on p from START to END
  end

  compound type HelloCompound()
    component HelloAtom c1()
  end
end
```

```
[BIP ENGINE]: BIP Engine (version 2013.
[BIP ENGINE]:
[BIP ENGINE]: initialize components...
[BIP ENGINE]: random scheduling based on seed=1404226060
[BIP ENGINE]: state #0: 1 internal port:
[BIP ENGINE]:   [0] ROOT.c1.p
[BIP ENGINE]: -> choose [0] ROOT.c1.p
[BIP ENGINE]: state #1: deadlock!
```

Hello World

```
$ bipc.sh -I . -p HelloPackage -d "HelloCompound()" \
     --gencpp-output output
$ cd build
$ cmake ../output
$ make
$ ./system
```

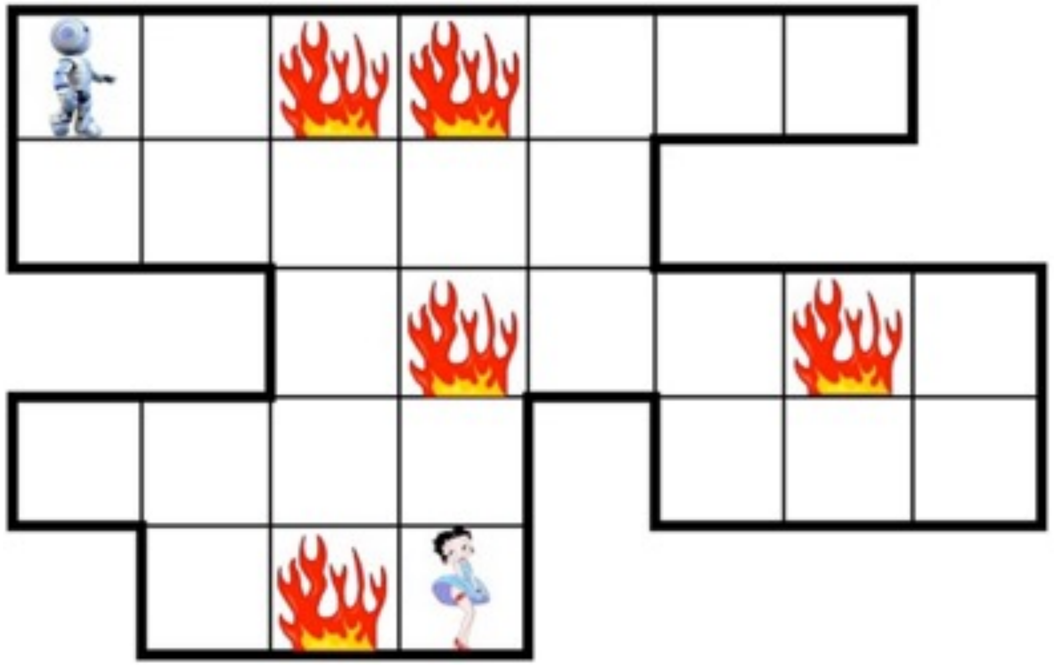
Also try options

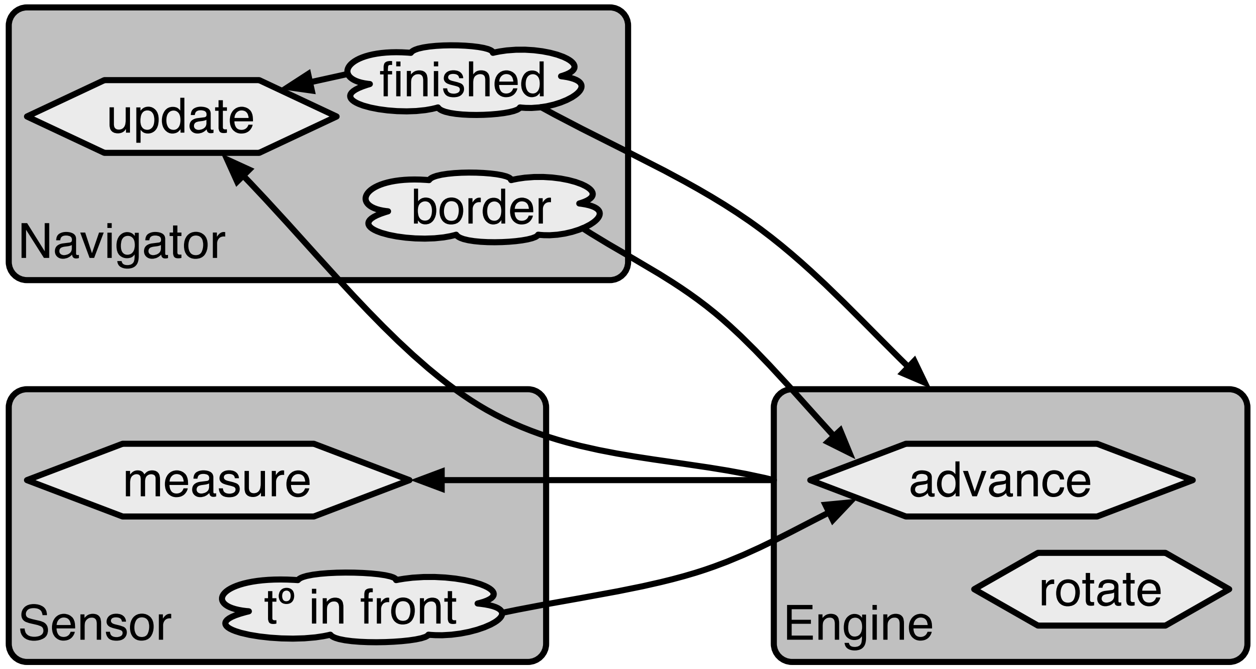
```
-i — interactive
-d — debug
```

```
package HelloPackage
  port type HelloPort_t()

  atom type HelloAtom()
    port HelloPort_t p()
    place START,END
    initial to START
    on p from START to END
  end

  compound type HelloCompound()
    component HelloAtom c1()
  end
end
```

```
[BIP ENGINE]: BIP Engine (version 2013.
[BIP ENGINE]:
[BIP ENGINE]: initialize components...
[BIP ENGINE]: random scheduling based on seed=1404226060
[BIP ENGINE]: state #0: 1 internal port:
[BIP ENGINE]:   [0] ROOT.c1.p
[BIP ENGINE]: -> choose [0] ROOT.c1.p
[BIP ENGINE]: state #1: deadlock!
```

Example: Rescue robot

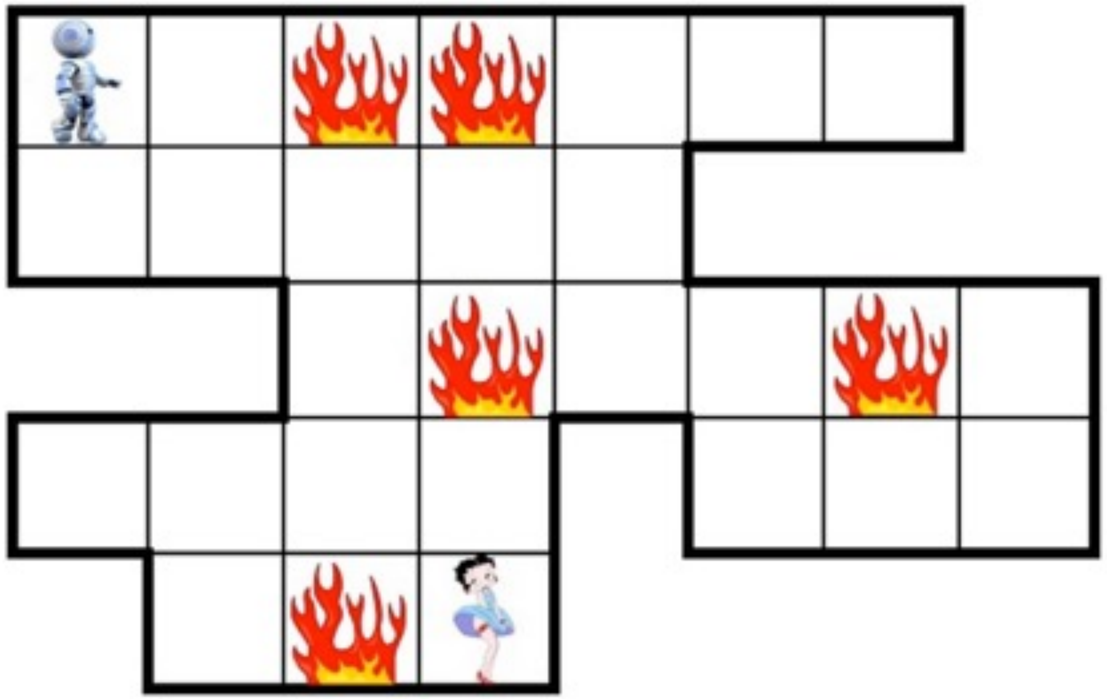


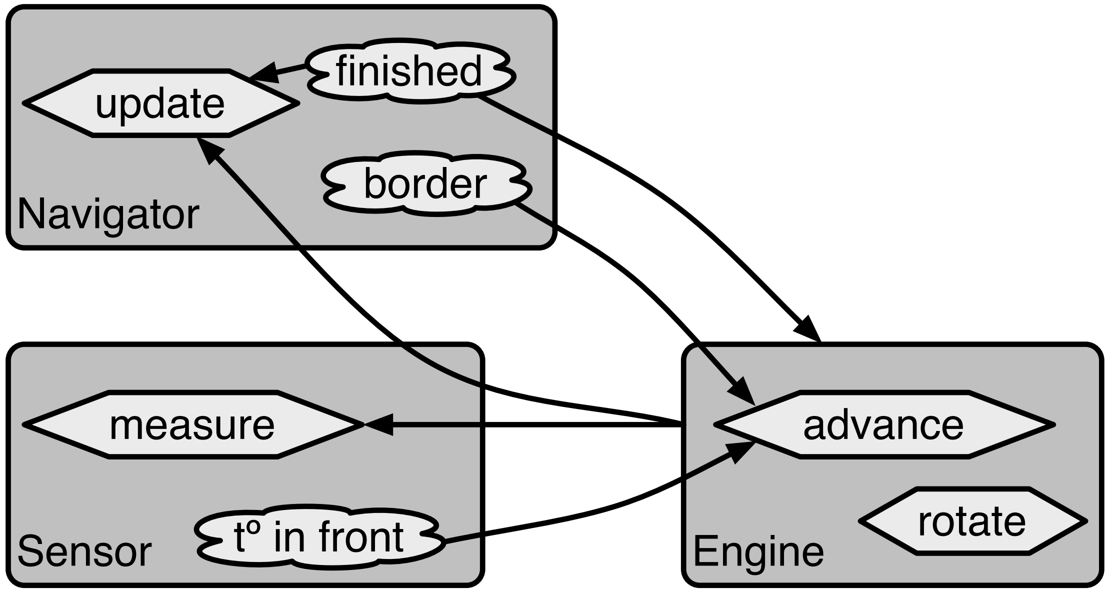


- Safety constraints
  - Must not advance and rotate at the same time
  - Must not leave the region
  - Must not move into burning areas
  - Must update navigation and sensor data at each move
  - When objective is found, must stop

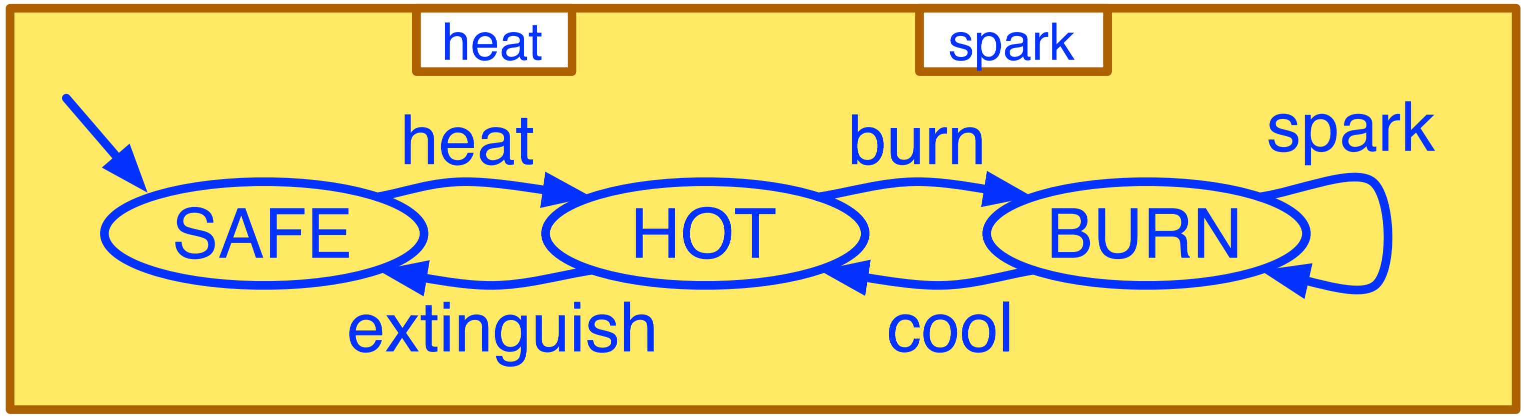
Rough plan

- One square
- N × N field (with N = 2, 5)
- Complete with the robot
- Remove the field!





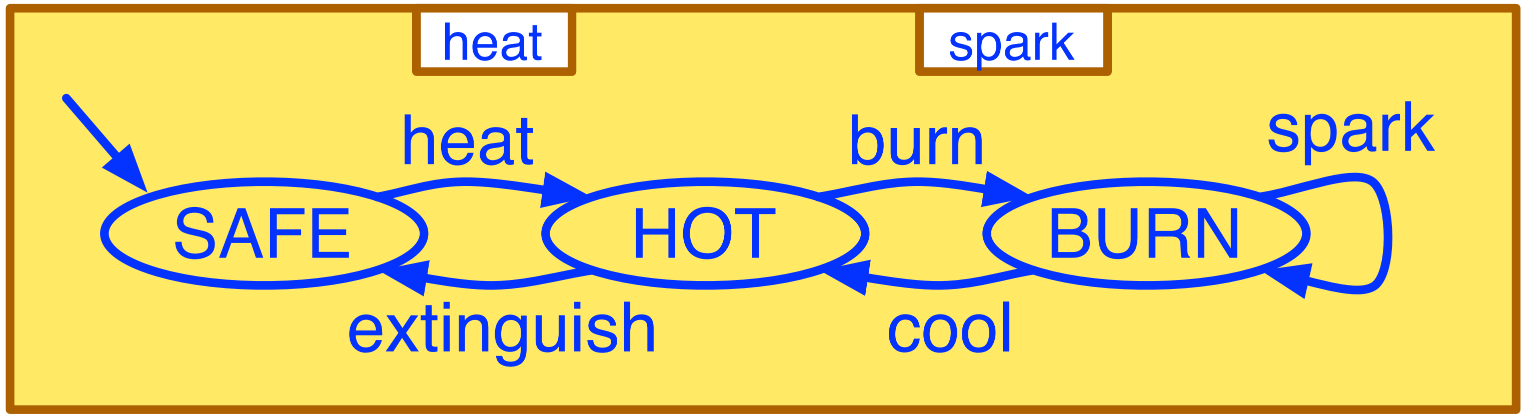
Atoms, ports and places



```
package RescueRobot
  port type Port_t()

  atom type Square()
    export port Port_t heat()
    export port Port_t spark()

    port Port_t burn()
    port Port_t cool()
    port Port_t extinguish()

    place SAFE, HOT, BURNING

    initial to SAFE
    <...>
  end
```

```
  connector type Singleton (Port_t p)
    define p
  end

  compound type Field()
    component Square square()

    connector Singleton
      c_heat(square.heat)
    connector Singleton
      c_spark(square.spark)
  end

  compound type RescueCompound()
    component Field field()
  end
end
```

Atoms, ports and places



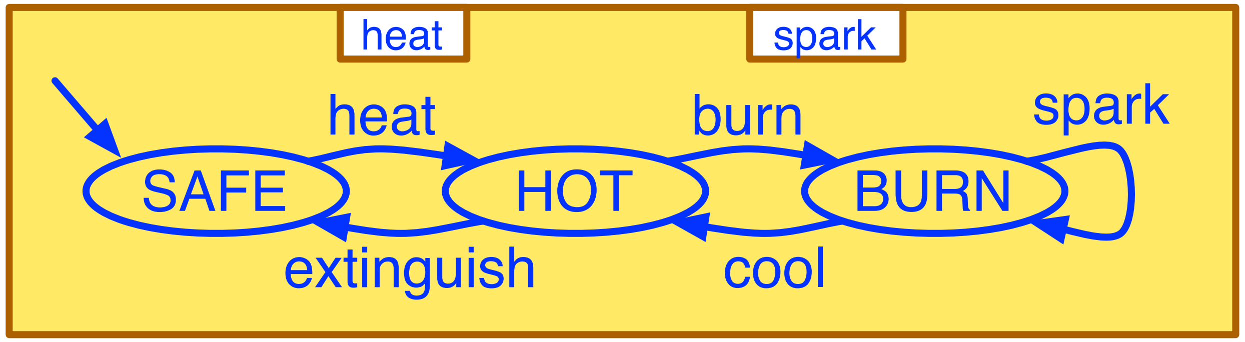
```
package RescueRobot
  port type Port_t()

  atom type Square()
    export port Port_t heat()
    export port Port_t spark()

    port Port_t burn()
    port Port_t cool()
    port Port_t extinguish()

    place SAFE, HOT, BURNING

    initial to SAFE
    <...>
  end
```

RescueRobot/10

```
  connector type Singleton (Port_t p)

  compound type Field()
    component Square square()

    connector Singleton
      c_heat(square.heat)
    connector Singleton
      c_spark(square.spark)
  end

  compound type RescueCompound()
    component Field field()
  end
end
```

Atoms, ports and places

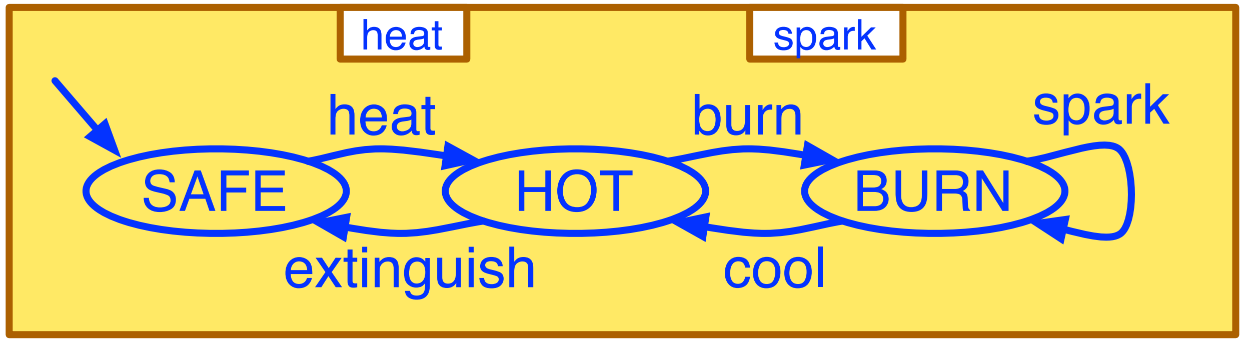
```
package RescueRobot
  port type Port_t()

  atom type Square()
    export port Port_t heat()
    export port Port_t spark()

    port Port_t burn()
    port Port_t cool()
    port Port_t extinguish()

    place SAFE, HOT, BURNING

    initial to SAFE
    on heat from SAFE to HOT
    on burn from HOT to BURNING
    on spark from BURNING to BURNING
    on cool from BURNING to HOT
    on extinguish from HOT to SAFE
  end

  connector type Singleton (Port_t p)
    define p
  end
```



```
  compound type Field()
    component Square square()

    connector Singleton
      c_heat(square.heat)
    connector Singleton
      c_spark(square.spark)
  end

  compound type RescueCompound()
    component Field field()
  end
end
```

Atoms, ports and places

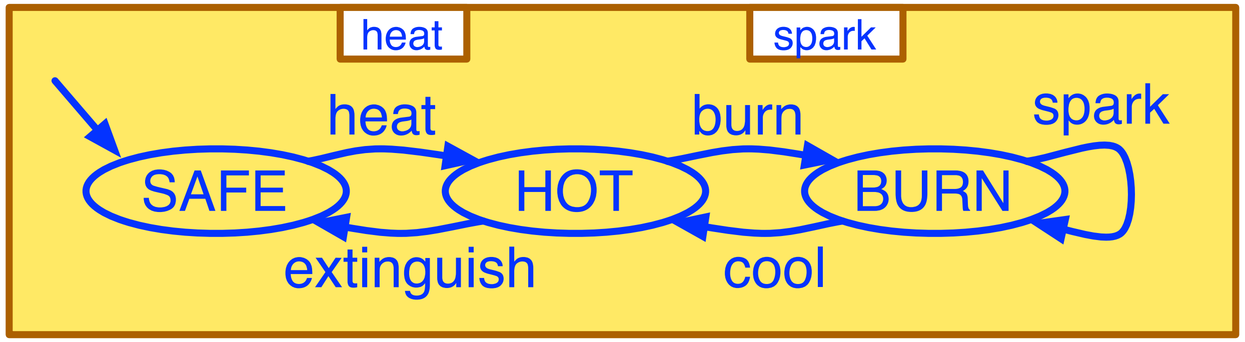
```
package RescueRobot
  port type Port_t()

  atom type Square()
    export port Port_t heat()
    export port Port_t spark()

    port Port_t burn()
    port Port_t cool()
    port Port_t extinguish()

    place SAFE, HOT, BURNING

    initial to SAFE
    on heat from SAFE to HOT
    on burn from HOT to BURNING
    on spark from BURNING to BURNING
    on cool from BURNING to HOT
    on extinguish from HOT to SAFE
  end

  connector type Singleton (Port_t p)
    define p
  end
```



```
  compound type Field()
    component Square square()

    connector Singleton
      c_heat(square.heat)
    connector Singleton
      c_spark(square.spark)
  end

  compound type RescueCompound()
    component Field field()
  end
end
```

Atoms, ports and places

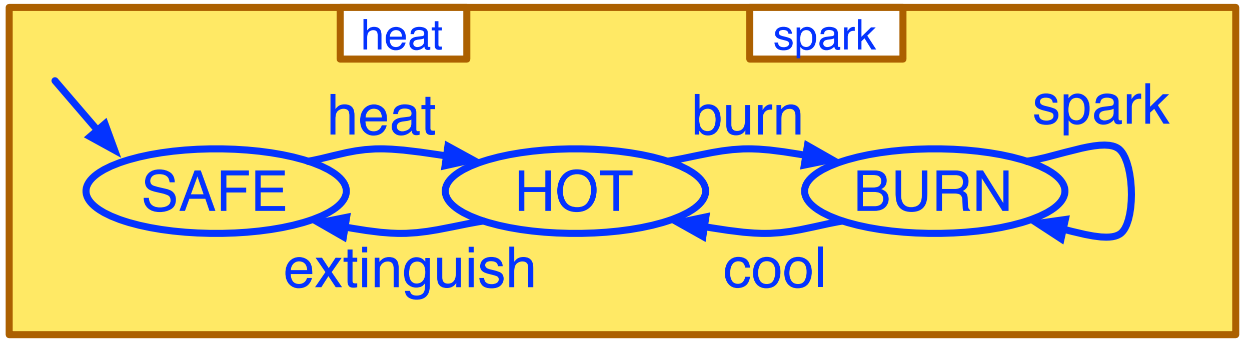
```
package RescueRobot
  port type Port_t()

  atom type Square()
    export port Port_t heat()
    export port Port_t spark()

    port Port_t burn()
    port Port_t cool()
    port Port_t extinguish()

    place SAFE, HOT, BURNING

    initial to SAFE
    on heat from SAFE to HOT
    on burn from HOT to BURNING
    on spark from BURNING to BURNING
    on cool from BURNING to HOT
    on extinguish from HOT to SAFE
  end

  connector type Singleton (Port_t p)
    define p
  end
```



```
  compound type Field()
    component Square square()

    connector Singleton
      c_heat(square.heat)
    connector Singleton
      c_spark(square.spark)
  end

  compound type RescueCompound()
    component Field field()
  end
end
```

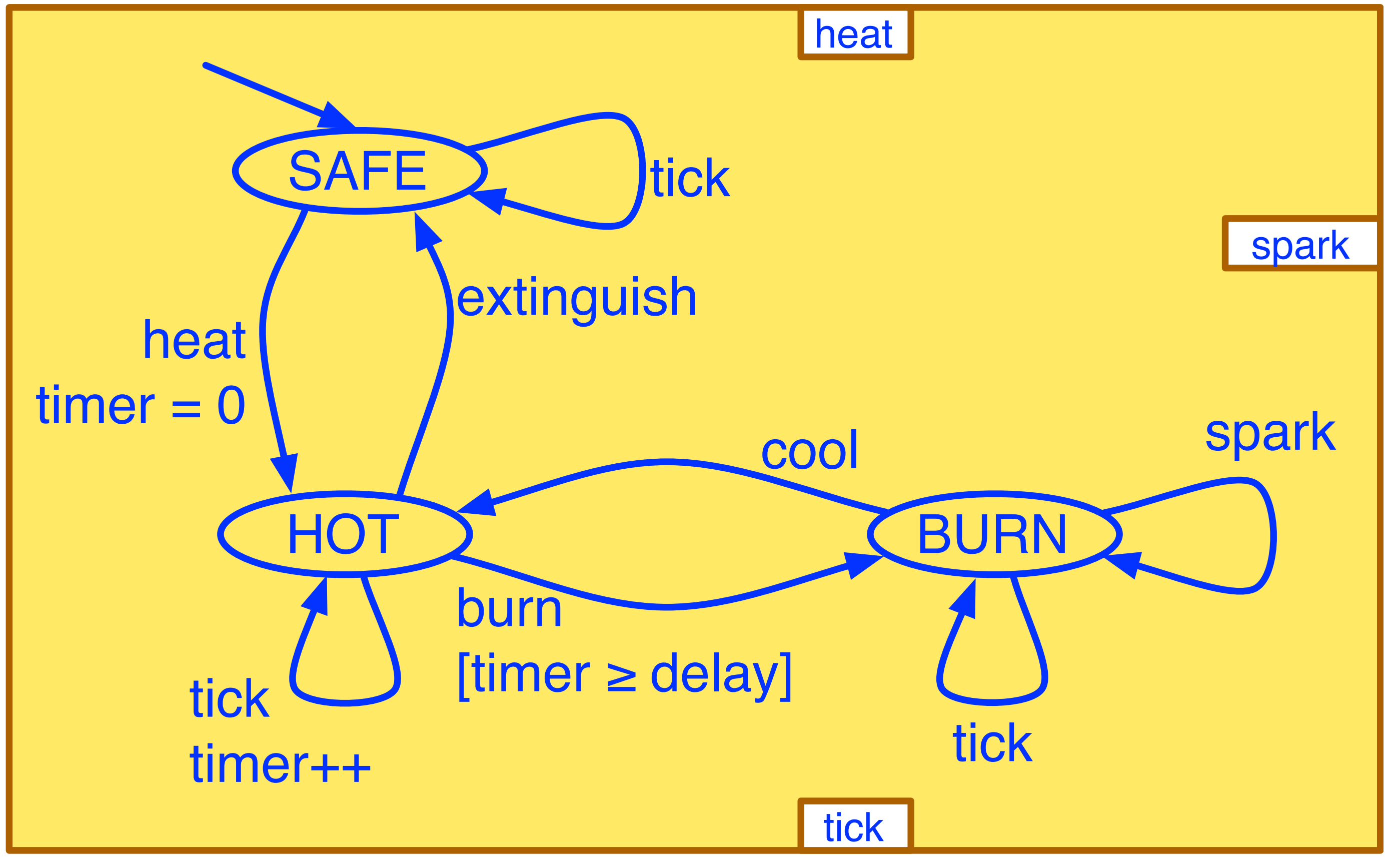
Atoms, ports and places

```
package RescueRobot
  port type Port_t()

  atom type Square()
    export port Port_t heat()
    export port Port_t spark()

    port Port_t burn()
    port Port_t cool()
    port Port_t extinguish()

    place SAFE, HOT, BURNING

    initial to SAFE
    on heat from SAFE to HOT
    on burn from HOT to BURNING
    on spark from BURNING to BURNING
    on cool from BURNING to HOT
    on extinguish from HOT to SAFE
  end

  connector type Singleton (Port_t p)
    define p
  end
```



```
  compound type Field()
    component Square square()

    connector Singleton
      c_heat(square.heat)
    connector Singleton
      c_spark(square.spark)
  end

  compound type RescueCompound()
    component Field field()
  end
end
```

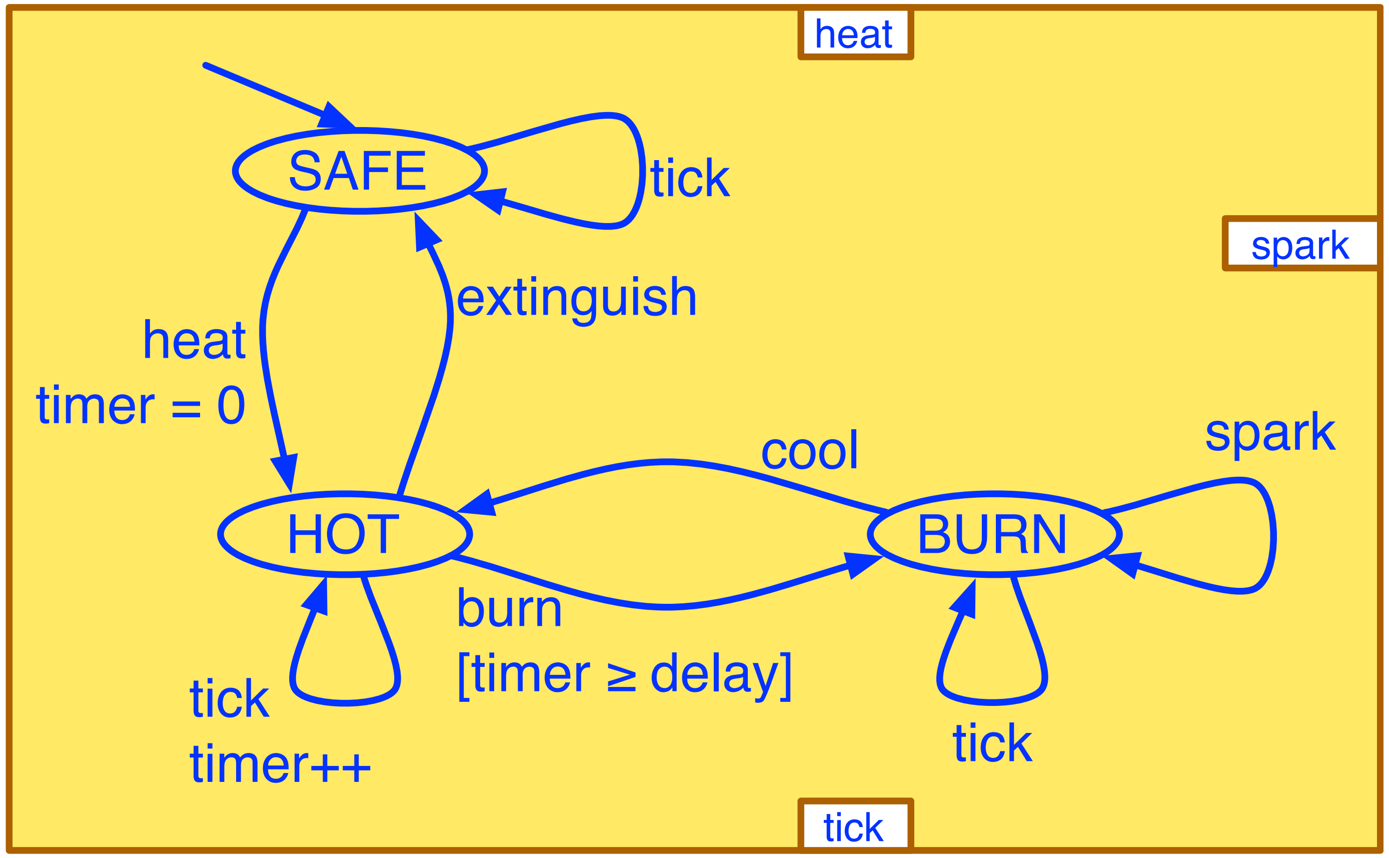
Data, guards and actions



```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
```

```
  on burn from HOT to BURNING
    provided (timer >= delay)

  <...>
  on tick from SAFE to SAFE
  on tick from HOT to HOT
    do {timer = timer + 1;}
  on tick from BURNING to BURNING
end
```

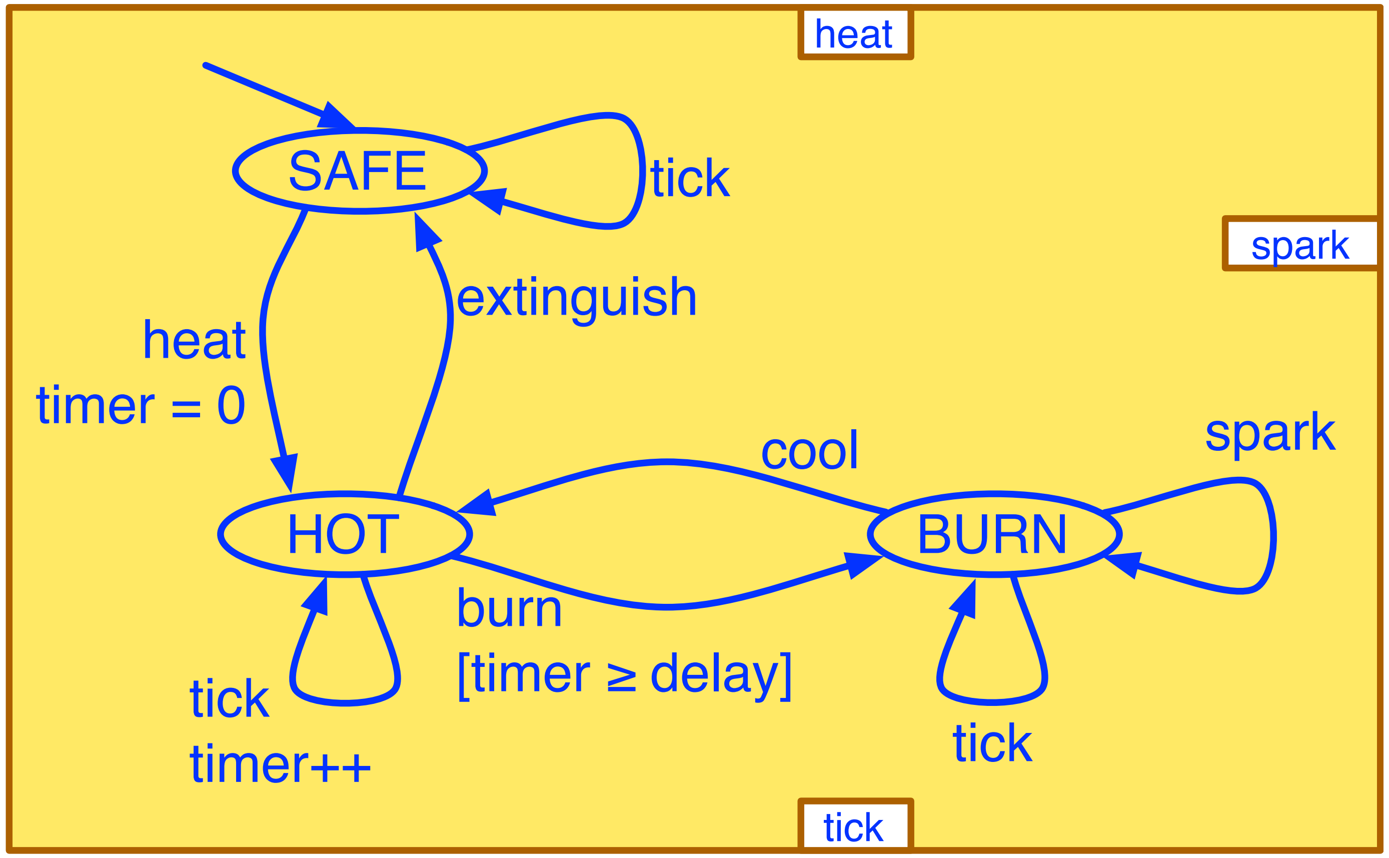
Data, guards and actions



```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
```

```
  on burn from HOT to BURNING
    provided (timer >= delay)

  <...>
  on tick from SAFE to SAFE
  on tick from HOT to HOT
    do {timer = timer + 1;}
  on tick from BURNING to BURNING
end
```

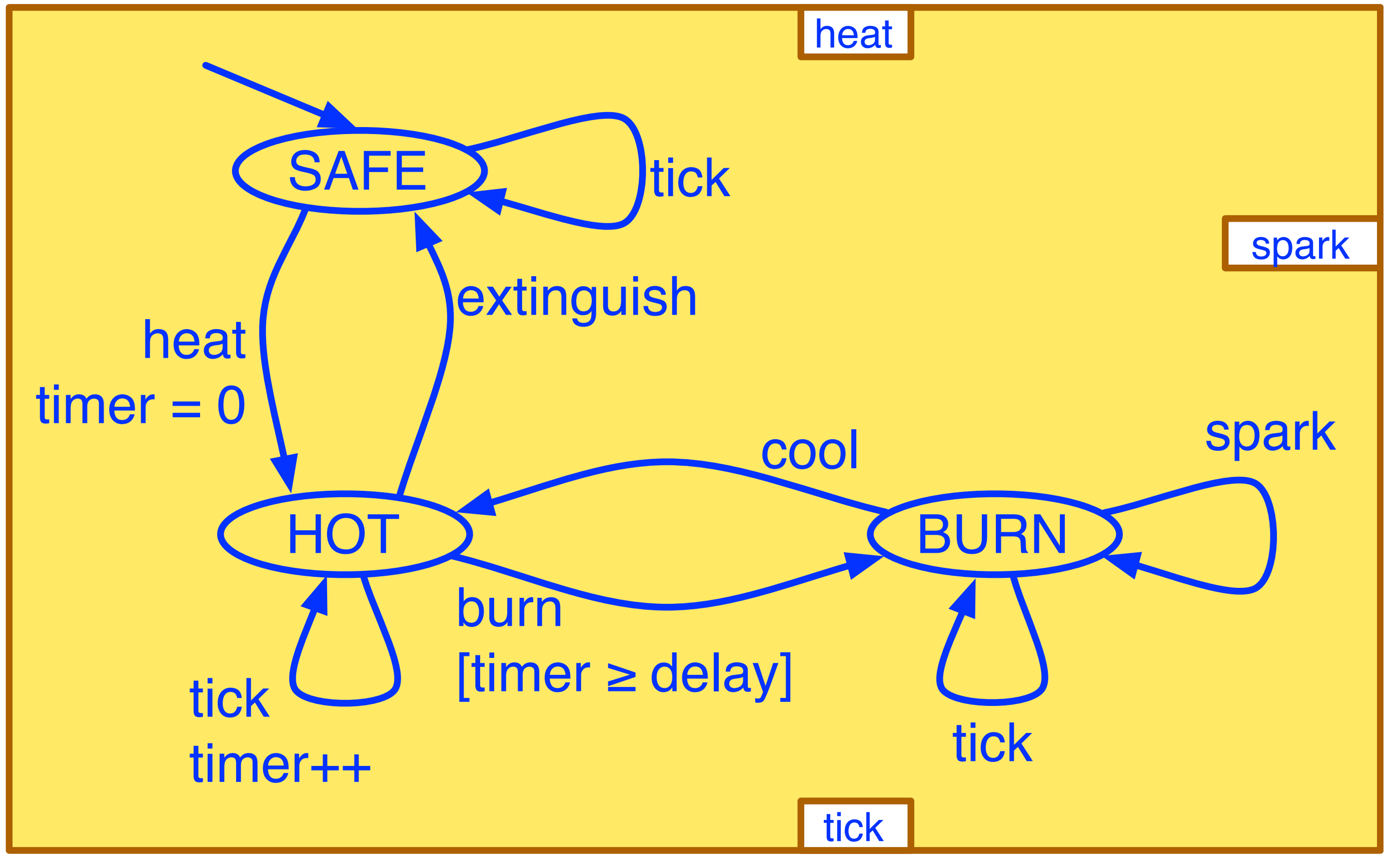
Data, guards and actions



```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
```

```
  on burn from HOT to BURNING
    provided (timer >= delay)

  <...>
  on tick from SAFE to SAFE
  on tick from HOT to HOT
    do {timer = timer + 1;}
  on tick from BURNING to BURNING
end
```

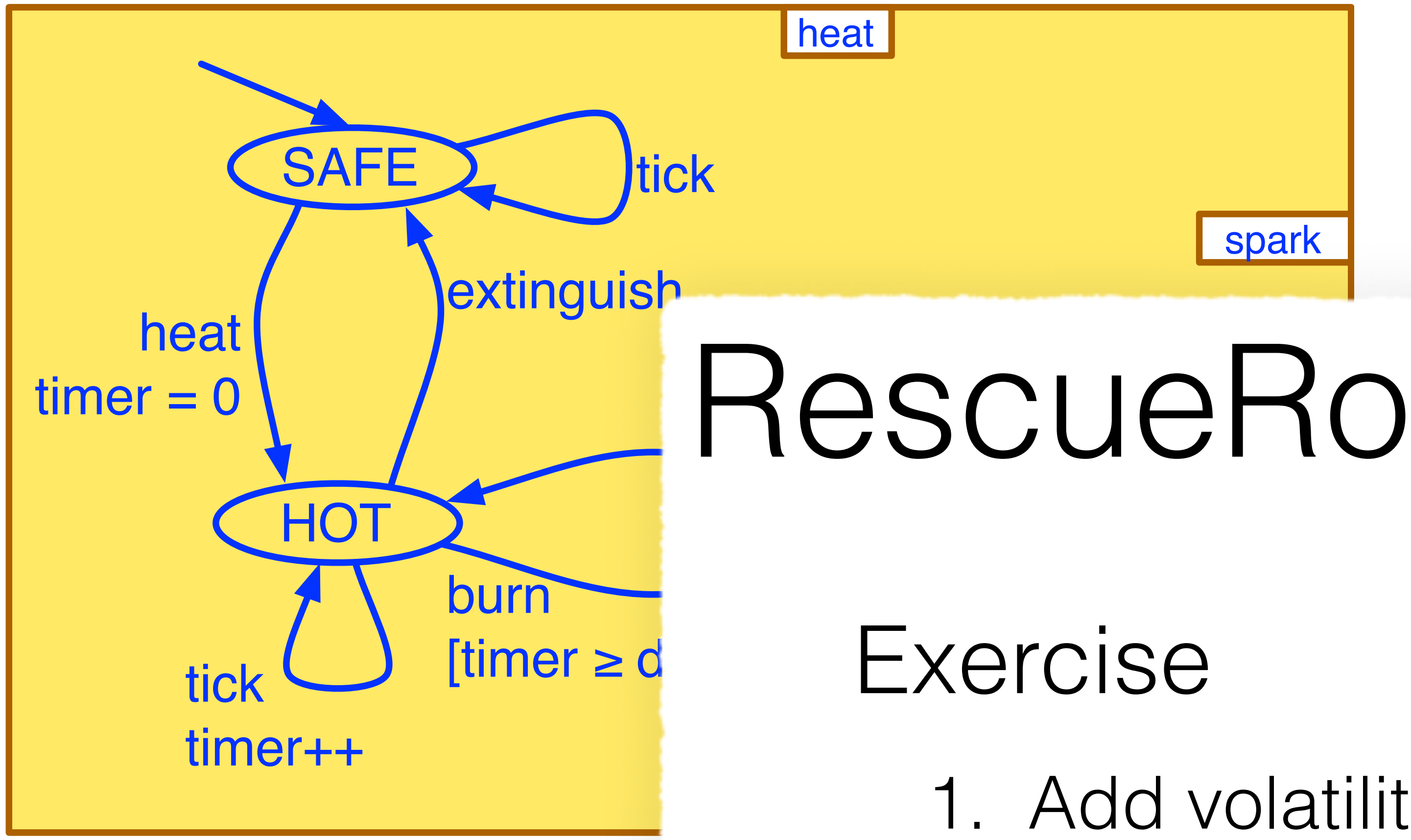
Data, guards and actions



```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
```

```
  on burn from HOT to BURNING
    provided (timer >= delay)

  <...>
  on tick from SAFE to SAFE
  on tick from HOT to HOT
    do {timer = timer + 1;}
  on tick from BURNING to BURNING
end
```

Data, guards and actions



RescueRobot/20

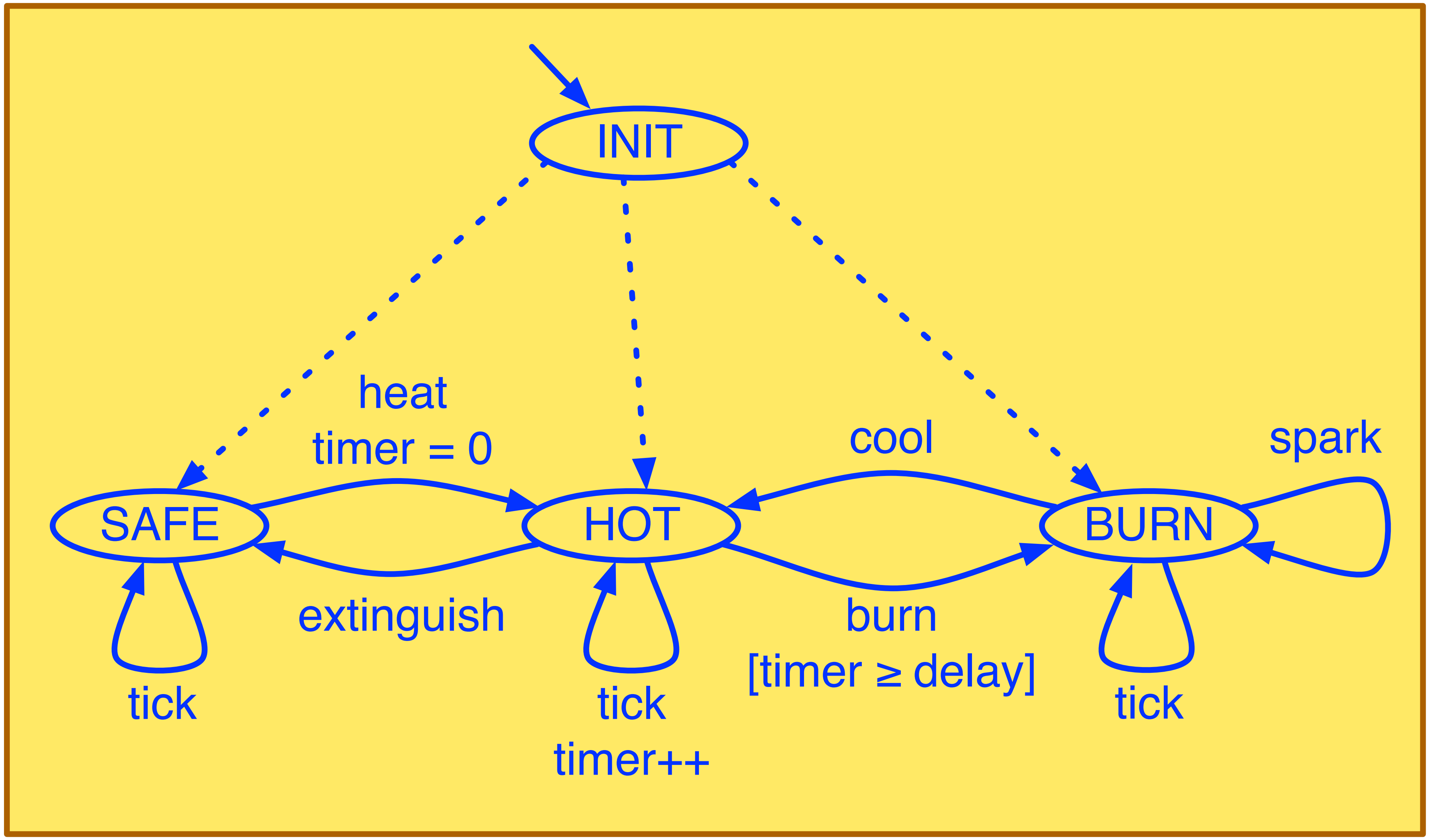
Exercise

1. Add volatility
2. Add initial temperature

```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
```

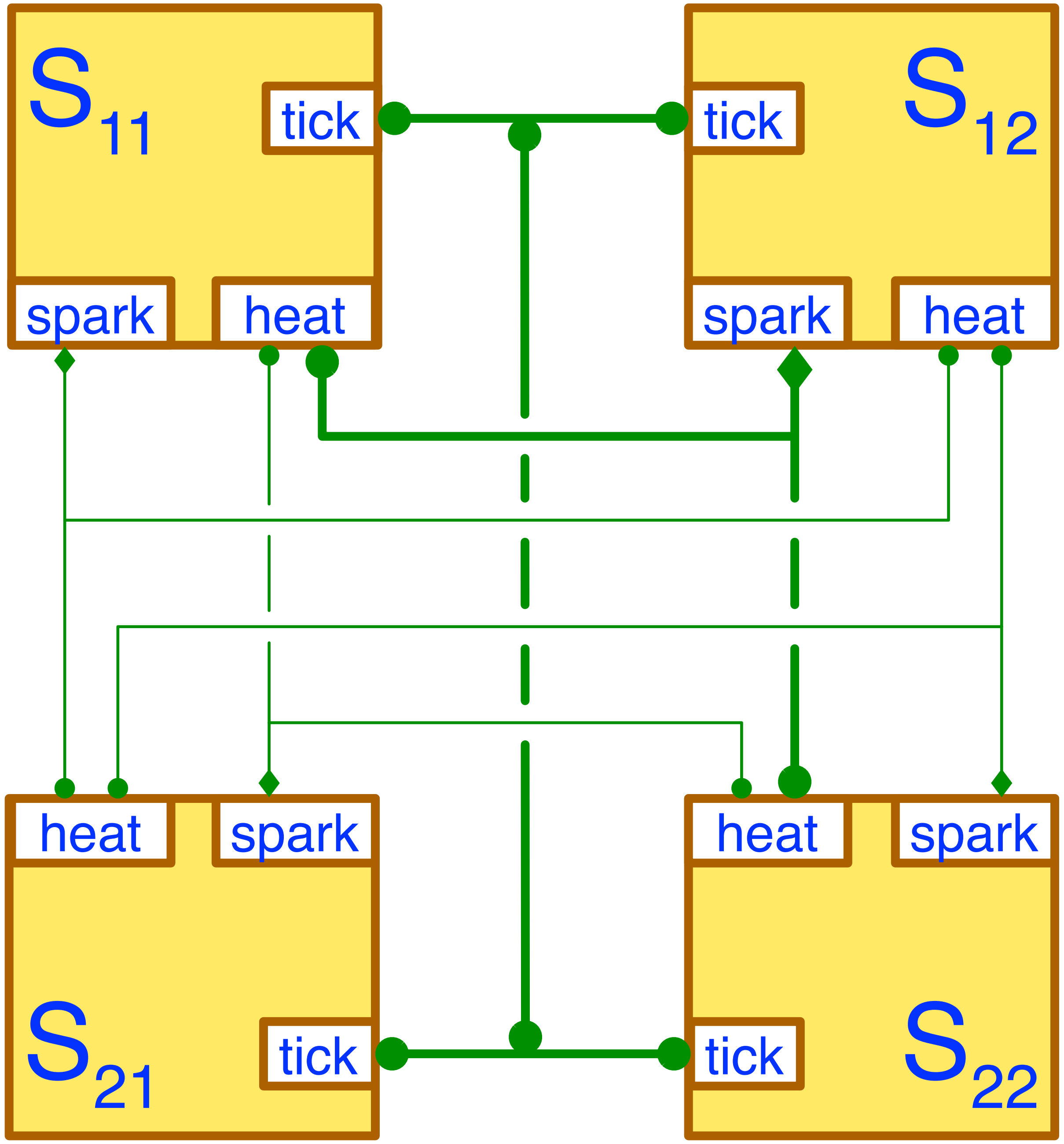
```
  on burn from HOT to BURNING
    provided (timer >= delay)

  <...>
  on tick from SAFE to SAFE
  on tick from HOT to HOT
    do {timer = timer + 1;}
  on tick from BURNING to BURNING
end
```

Internal transitions



```
internal from INIT to ...
```

Connectors



```
connector type Synchron2 (
  Port_t p, Port_t q
)
   export port Port_t sync()
   define p q
end
```

Notice:

- $[[\text{tick}_{11}\ \text{tick}_{12}]\ [\text{tick}_{21}\ \text{tick}_{22}]]$

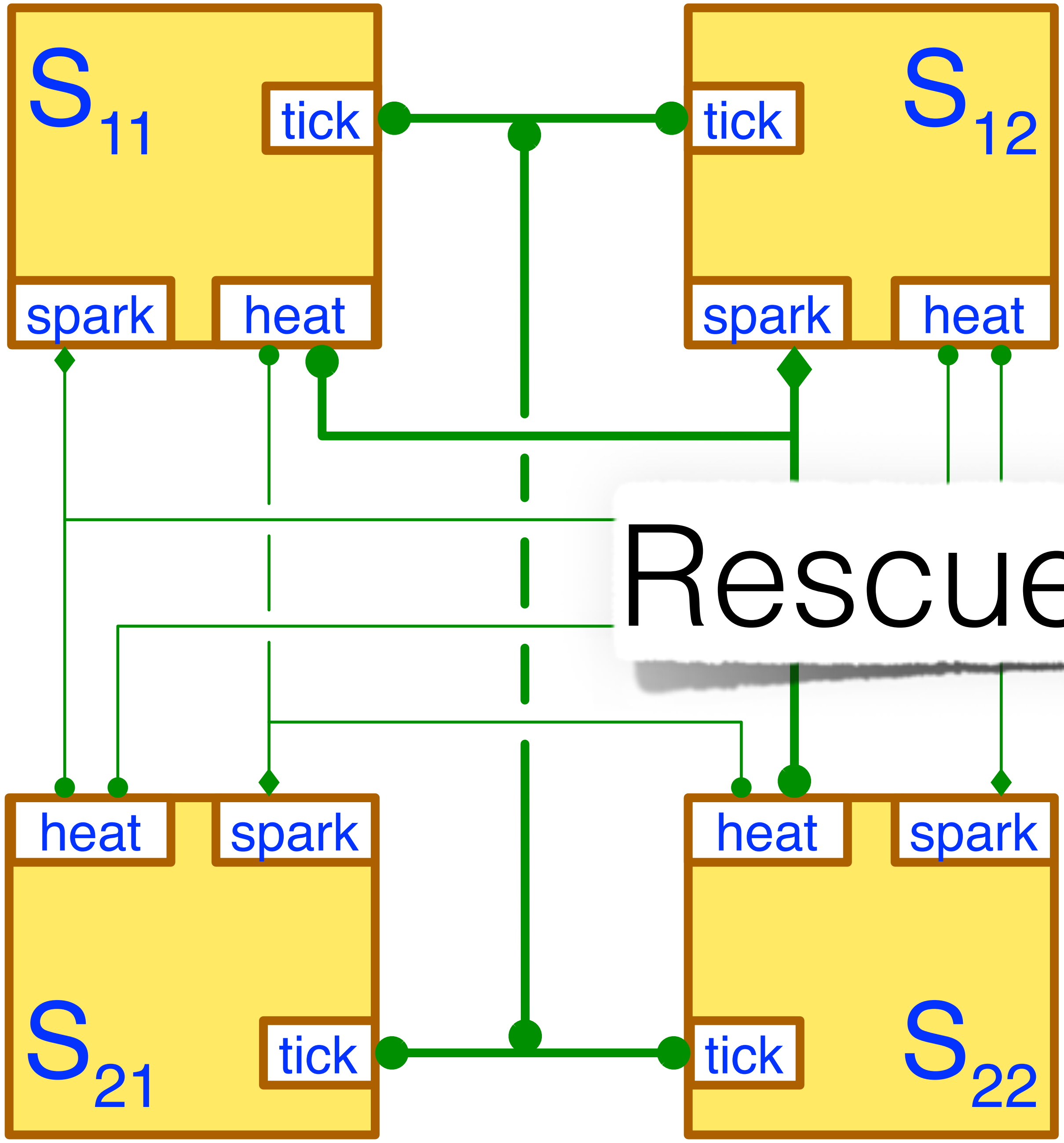
  $\sim [\text{tick}_1\ \text{tick}_2\ \text{tick}_3\ \text{tick}_4]$

- $\text{spark}_{12}'\ \text{heat}_{11}\ \text{heat}_{22}$

  $\sim [\text{spark}_{12}'\ \text{heat}_{11}]'\ \text{heat}_{22}$

S. Bliudze, J. Sifakis.
The Algebra of Connectors—Structuring Interaction in BIP [EMSOFT'07]

Connectors



```
connector type Synchron2 (
  Port_t p, Port_t q
)
   export port Port_t sync()
   define p q
end
```

Notice:

- $\left[[\text{tick}_{11}\ \text{tick}_{12}]\ [\text{tick}_{21}\ \text{tick}_{22}] \right]$

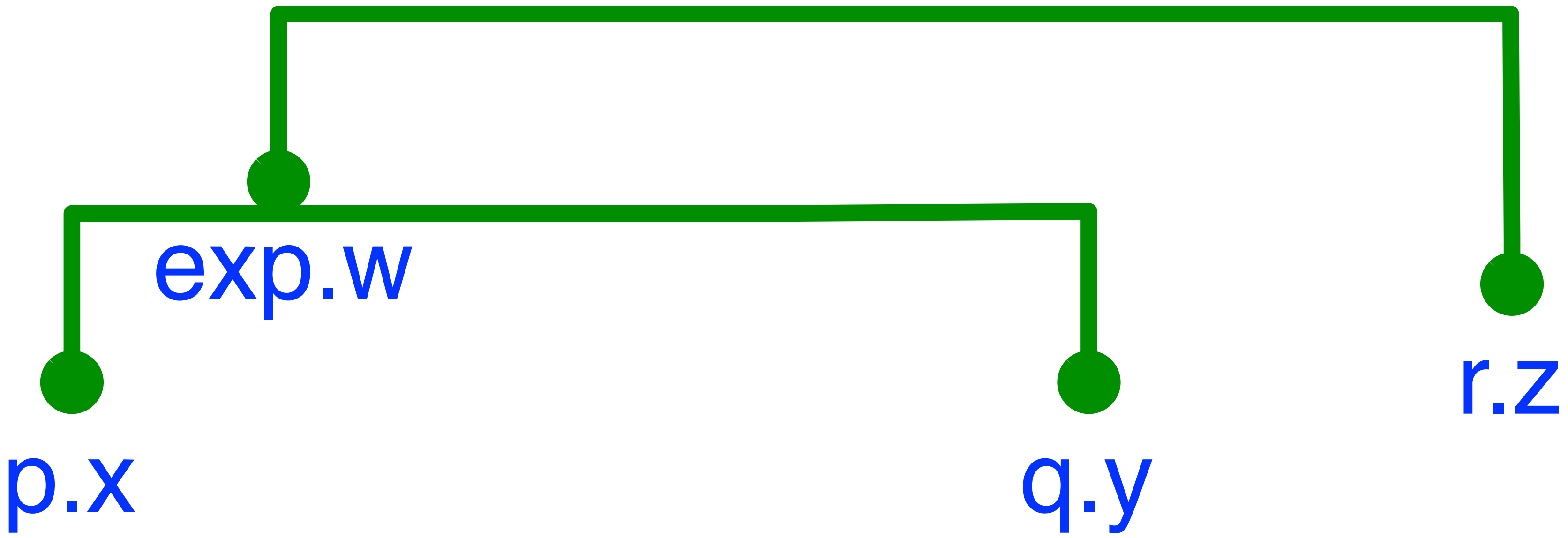
  $\sim \left[\text{tick}_1\ \text{tick}_2\ \text{tick}_3\ \text{tick}_4 \right]$

- $\text{spark}_{12}'\ \text{heat}_{11}\ \text{heat}_{22}$

  $\sim \left[\text{spark}_{12}'\ \text{heat}_{11} \right]'\ \text{heat}_{22}$

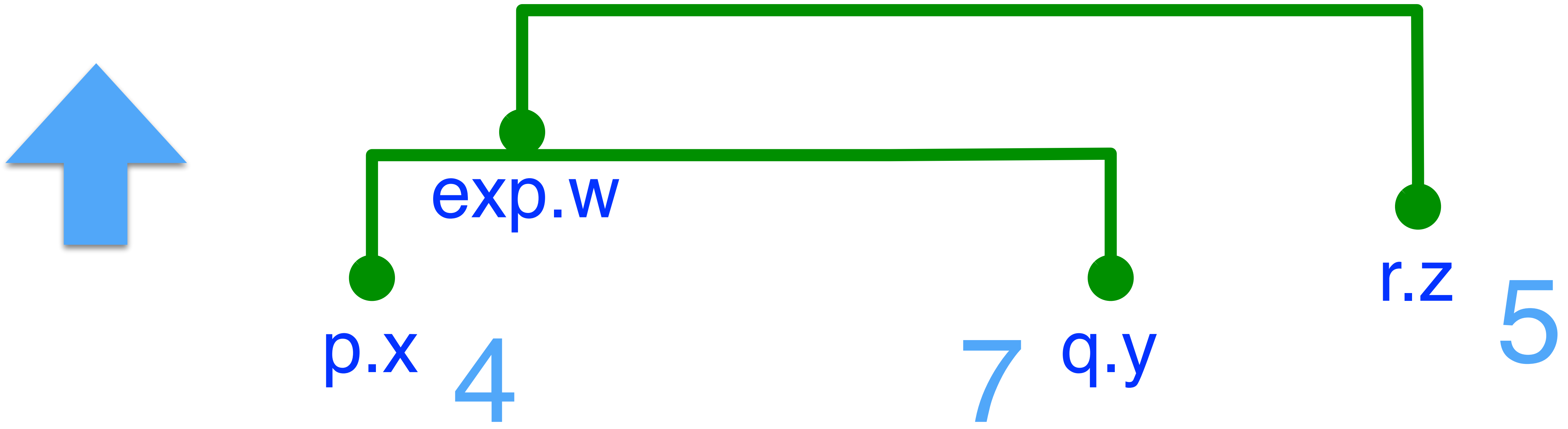
S. Bliudze, J. Sifakis.
The Algebra of Connectors—Structuring Interaction in BIP [EMSOFT'07]

Data transfer



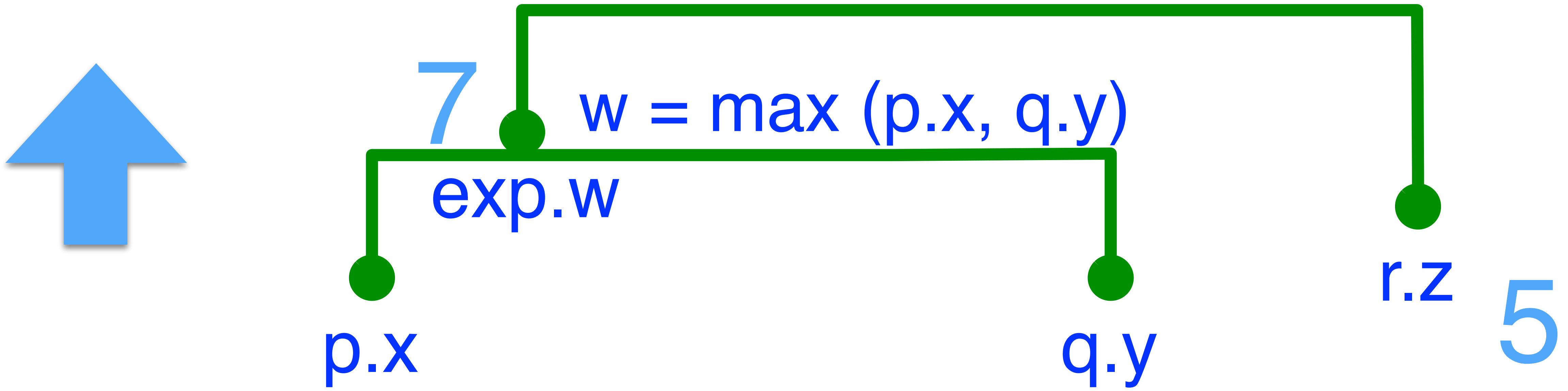
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



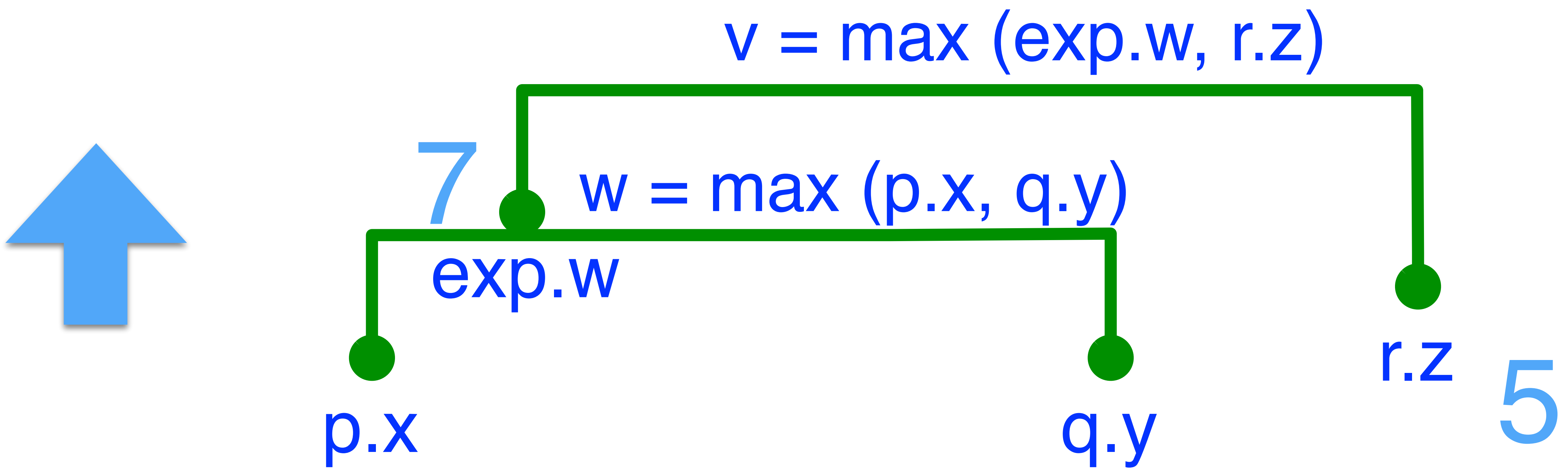
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



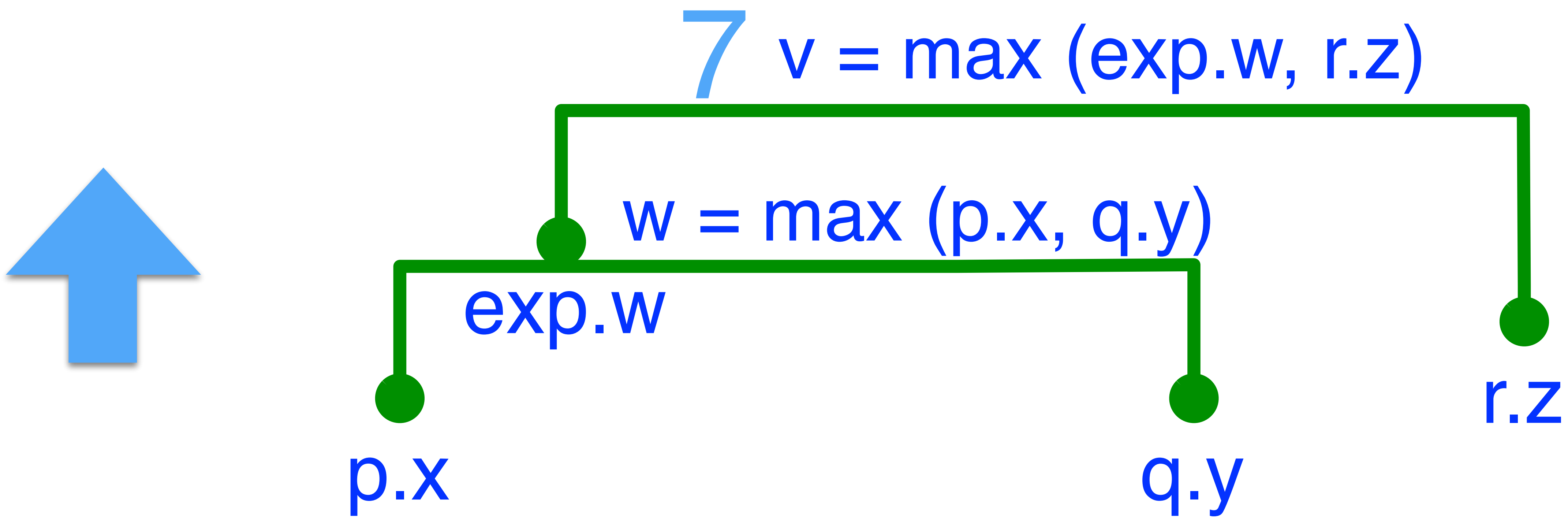
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



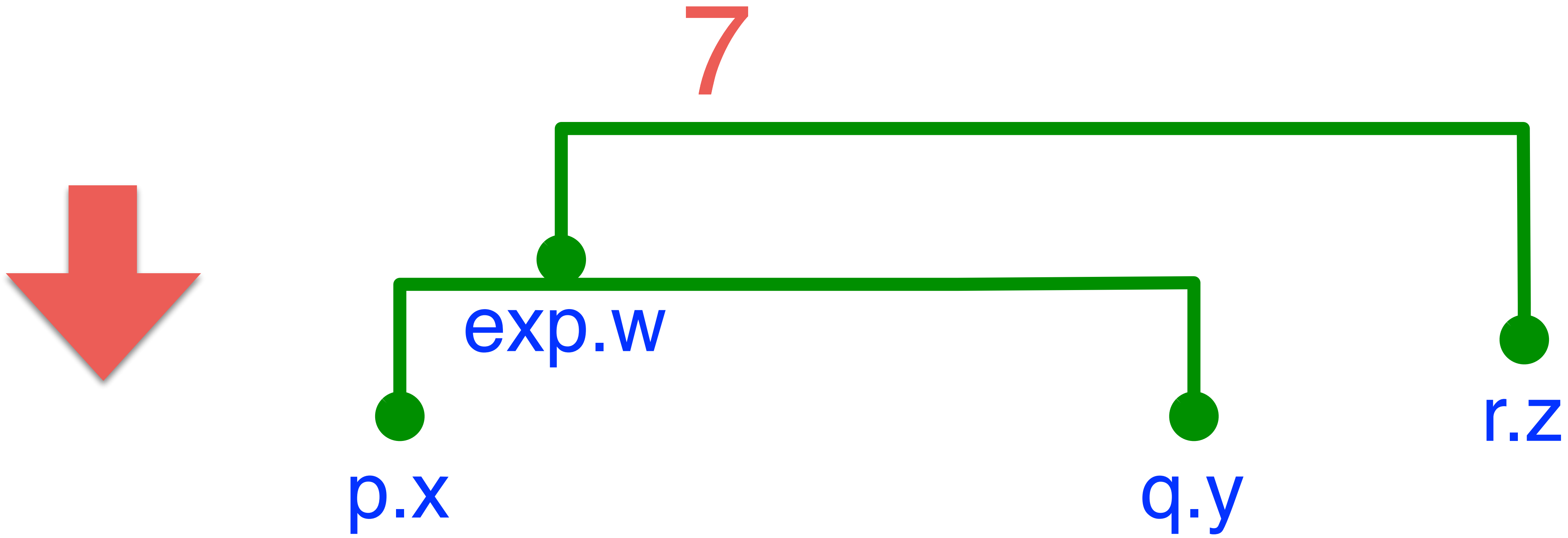
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



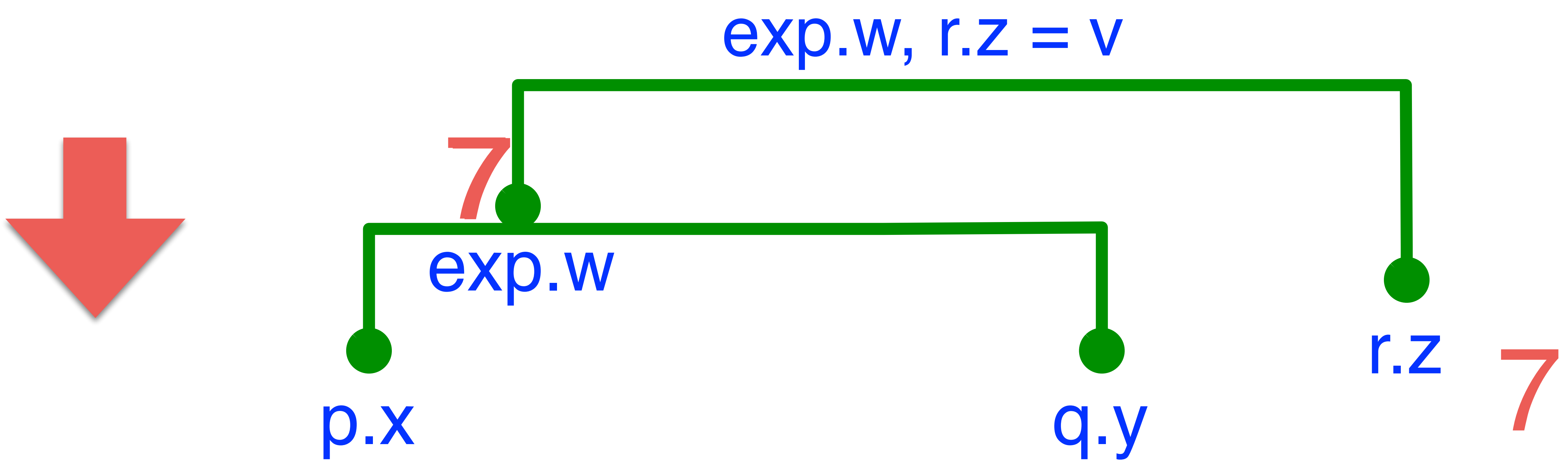
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



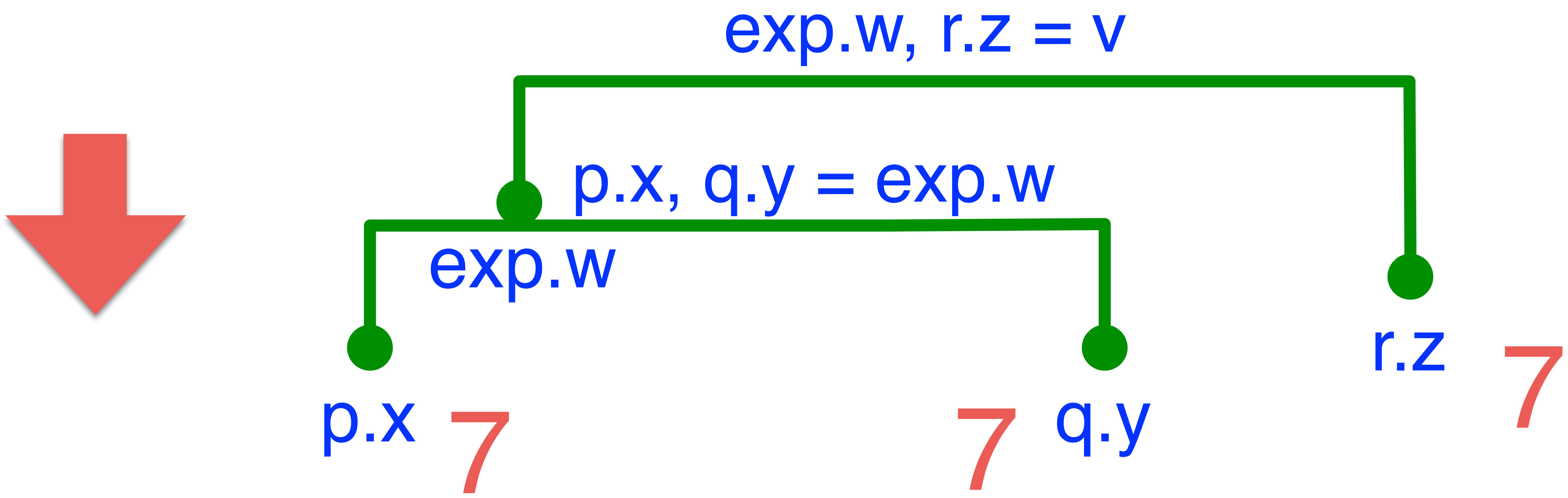
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

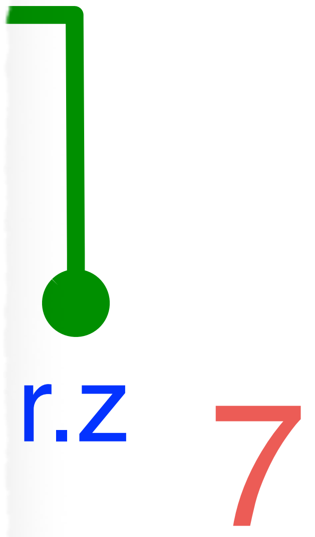
Data transfer



```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer

exp w, r.z = v



RescueRobot/35

Exercise

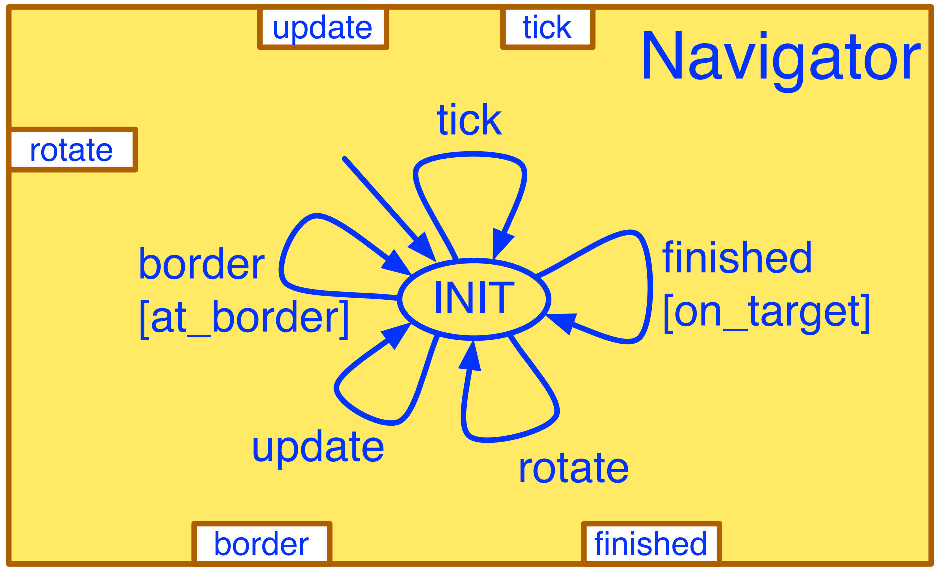
1. Add connectors to gather and print information about the temperature in all squares of the field.
2. Add an atom to enforce this after each tick of the clock.

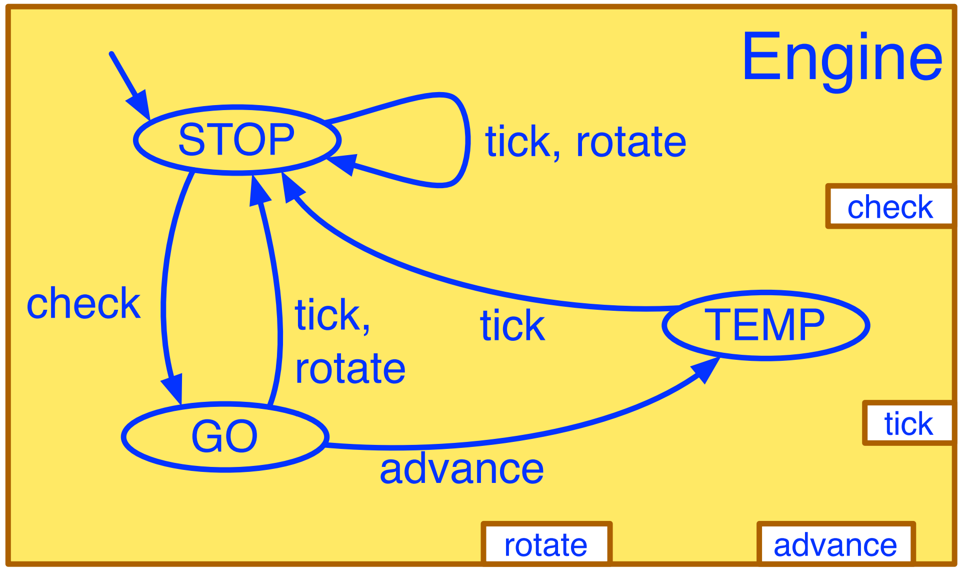
(Notice also the @cpp(…) annotation in the 1st line.)

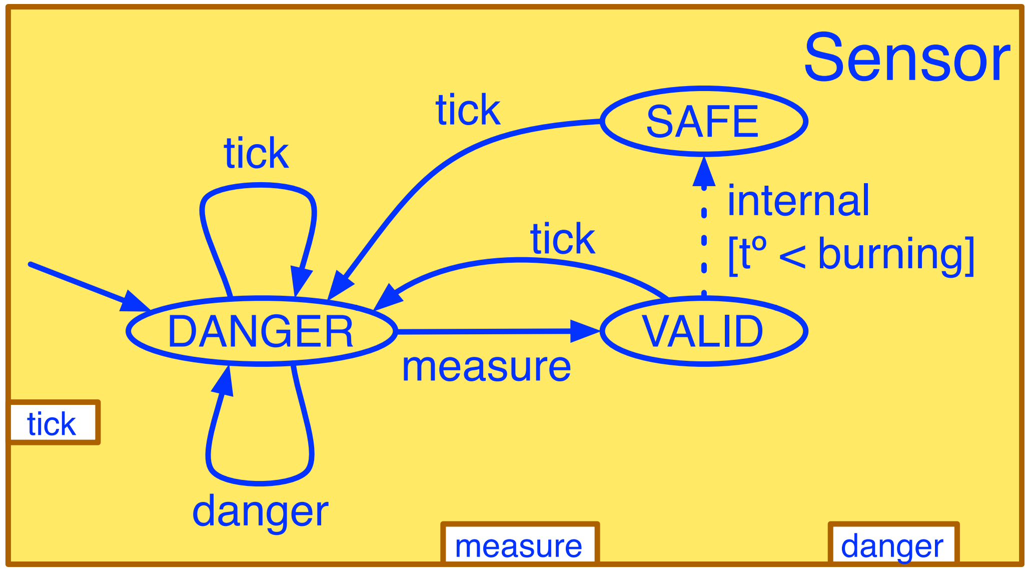
```
connecto
   data i
   export
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Components of the robot

- Safety constraints
  - Must not advance and rotate at the same time
  - Must not leave the region
  - Must not move into burning areas
  - Must update navigation and sensor data at each move
  - When objective is found, must stop

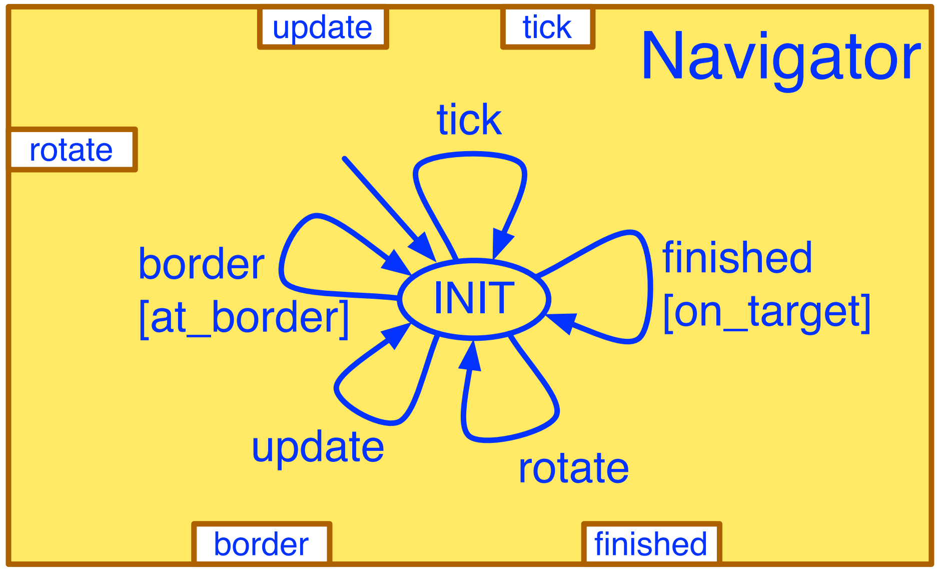


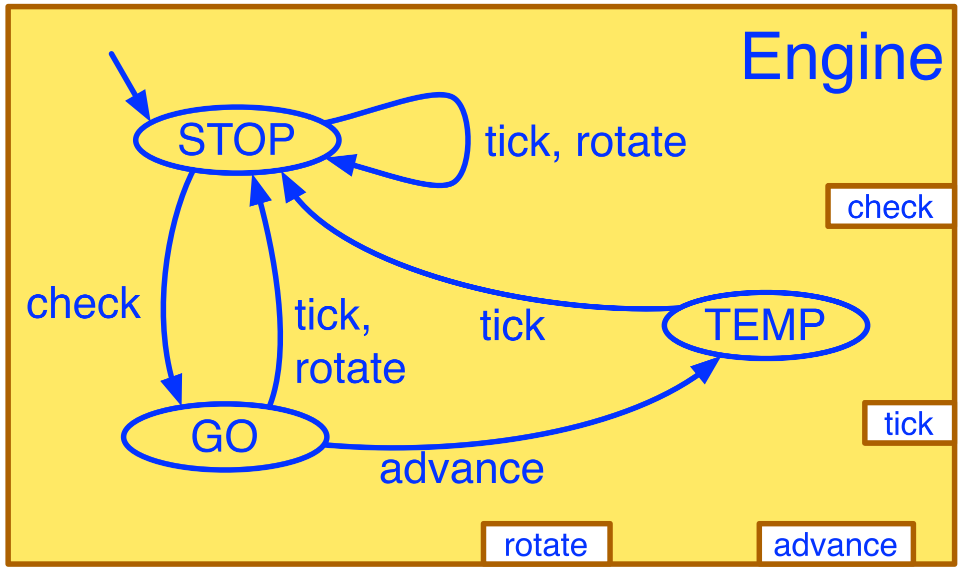


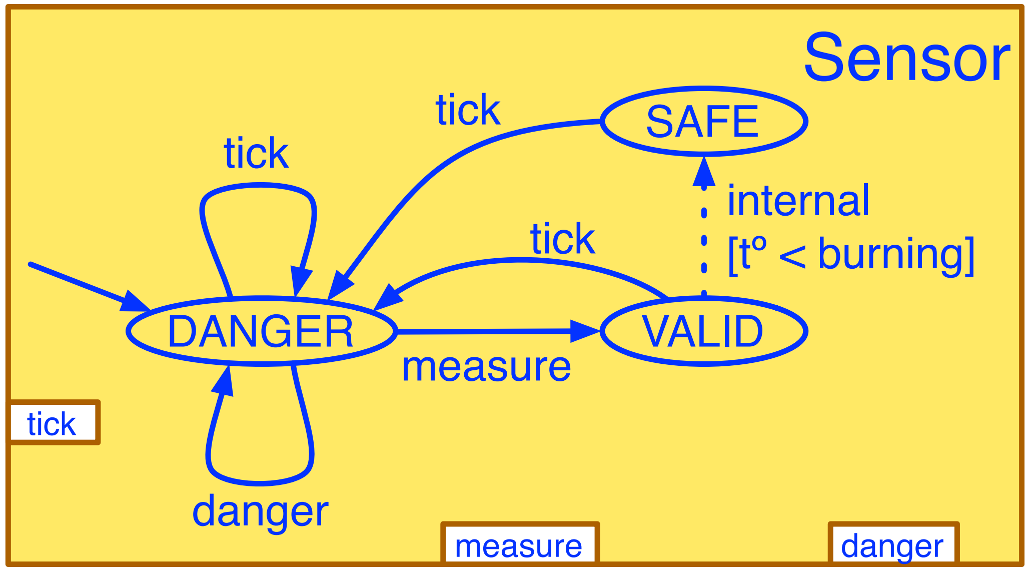


Components of the robot

- Safety constraints
  - Must not advance and rotate at the same time
  - Must not leave the region
  - Must not move into burning areas
  - Must update navigation and sensor data at each move
  - When objective is found, must stop

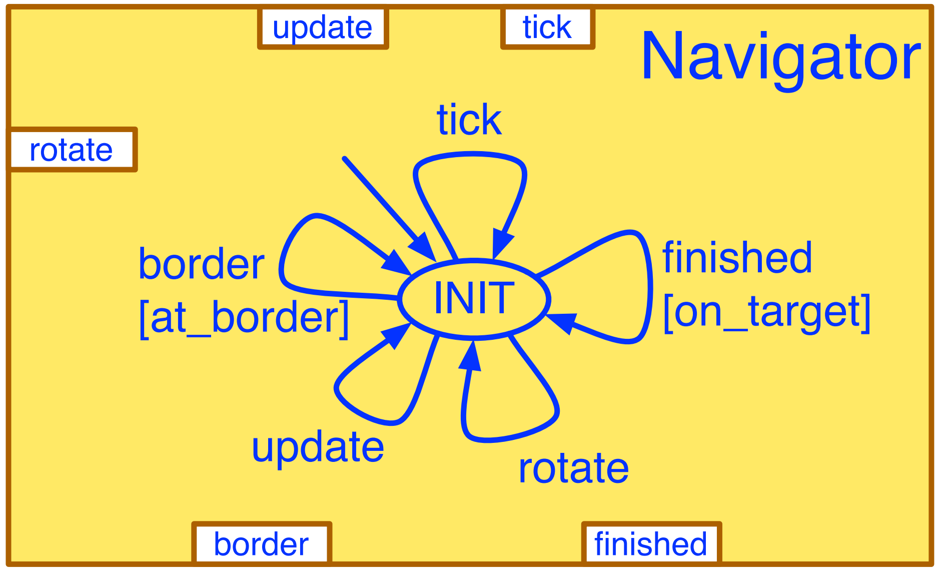


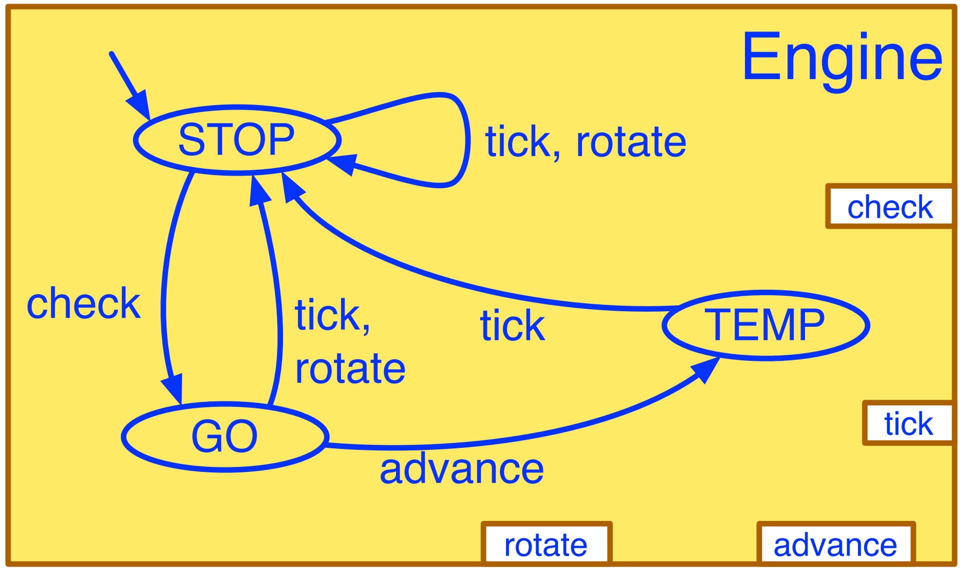


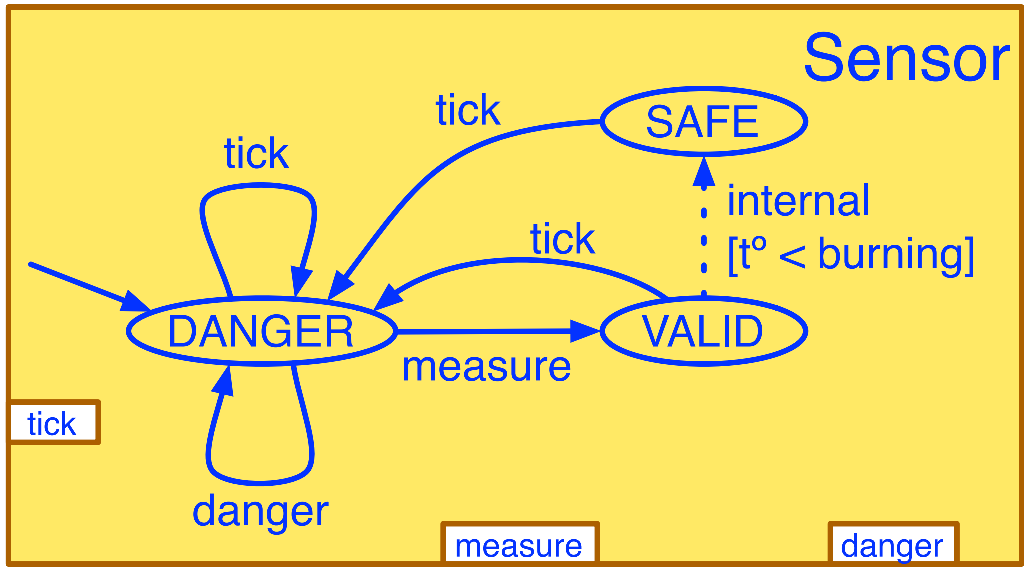


Components of the robot

- Safety constraints
  - ✓ Must not advance and rotate at the same time
  - Must not leave the region
  - Must not move into burning areas
  - Must update navigation and sensor data at each move
  - When objective is found, must stop



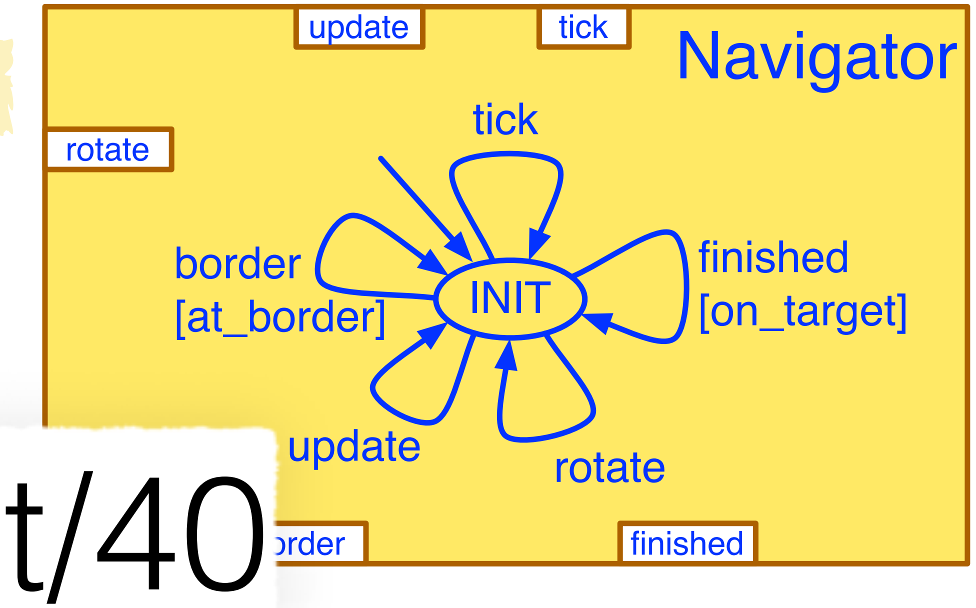


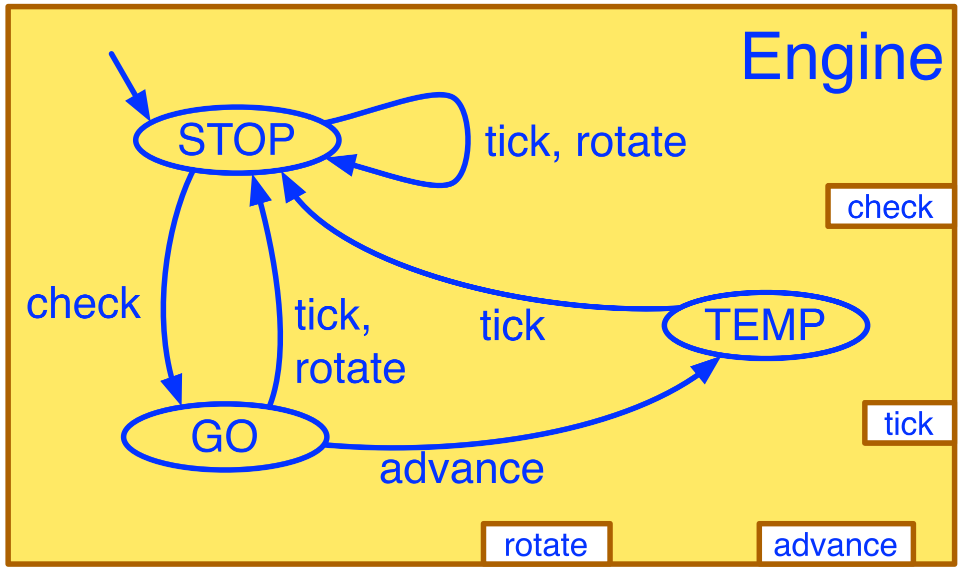


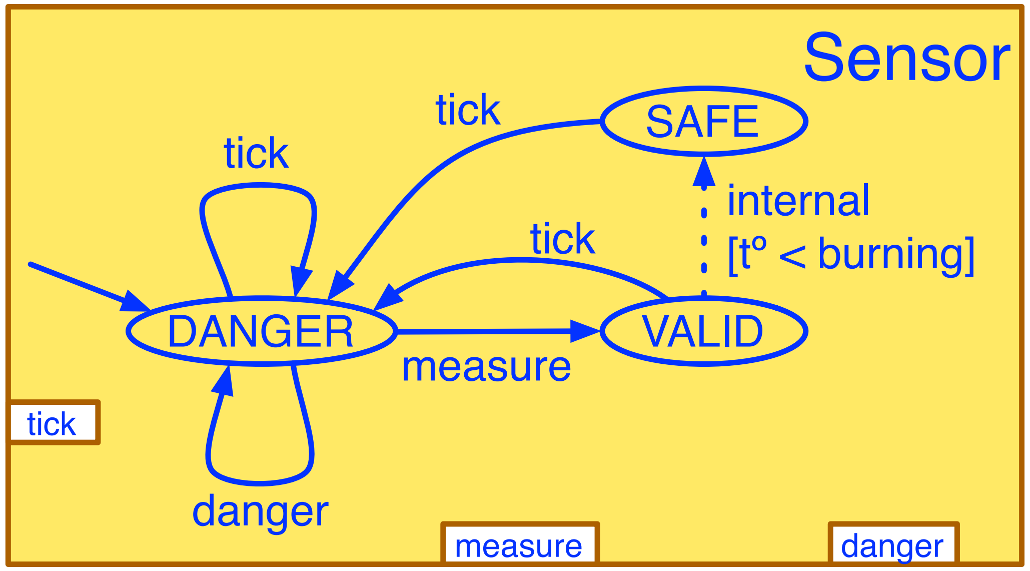
Components of the robot

- Safety constraints
  - ✓ Must not advance and rotate at the same time
  - Must not leave the region
  - Must not move into burning areas
  - Must update navi each move
  - When objective is found, must stop

RescueRobot/40

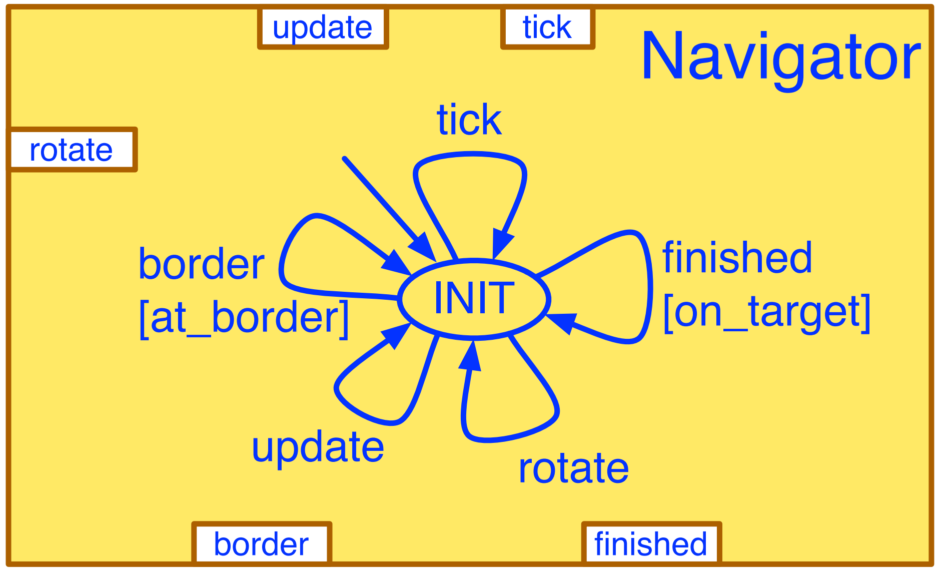


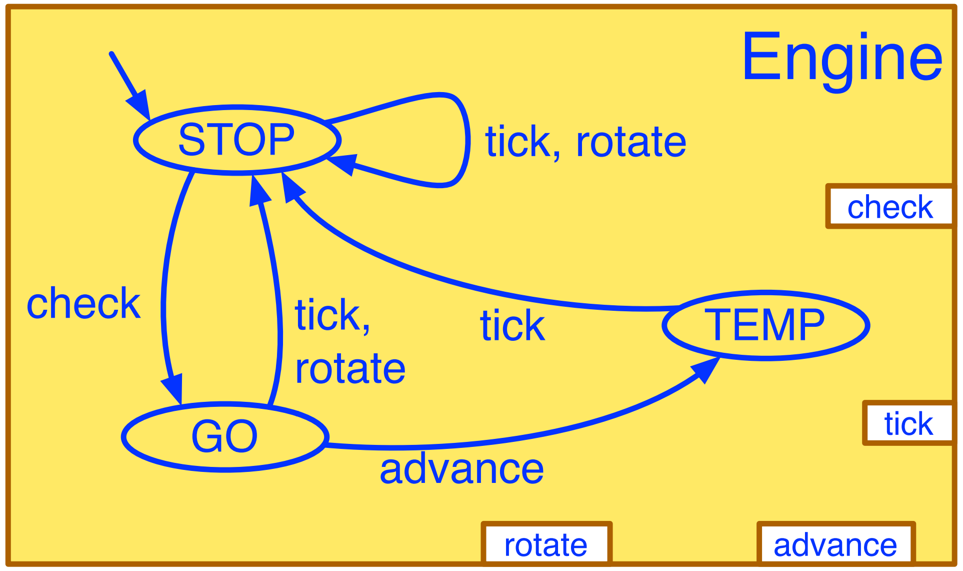


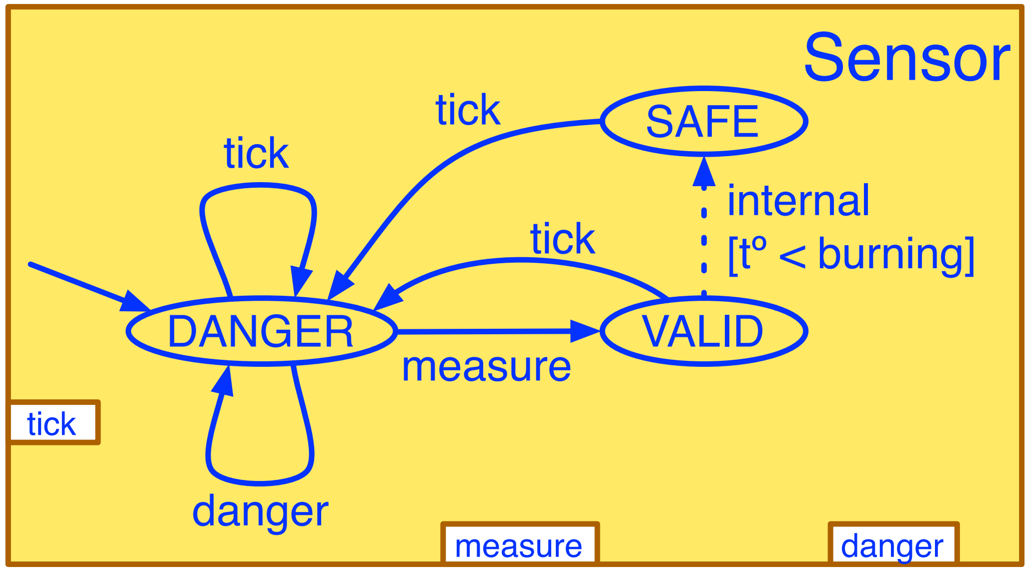


Components of the robot

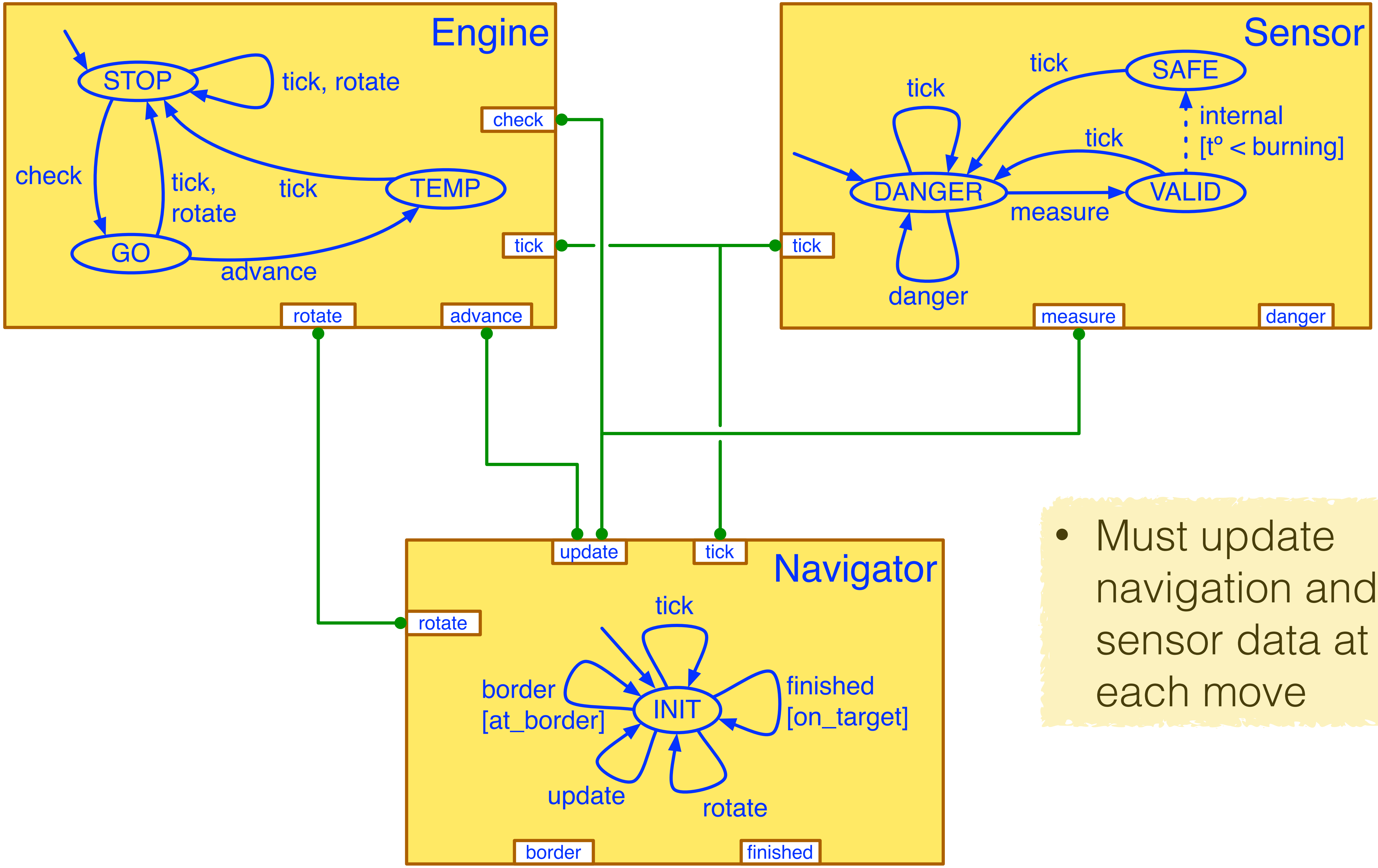
- Safety constraints
  - ✓ Must not advance and rotate at the same time
  - Must not leave the region
  - Must not move into burning areas
  - Must update navigation and sensor data at each move
  - When objective is found, must stop





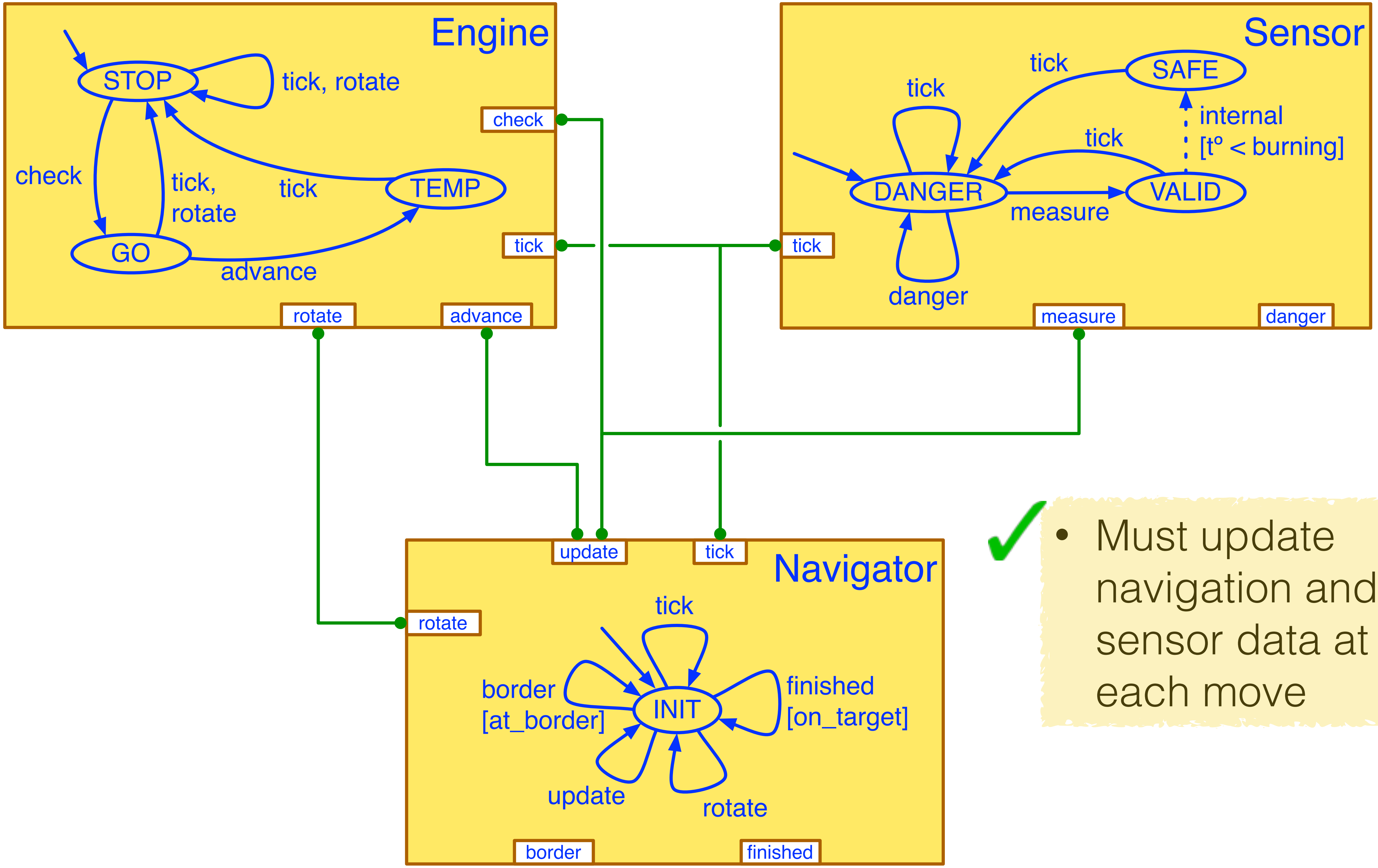


Connecting the robot



- Must update navigation and sensor data at each move

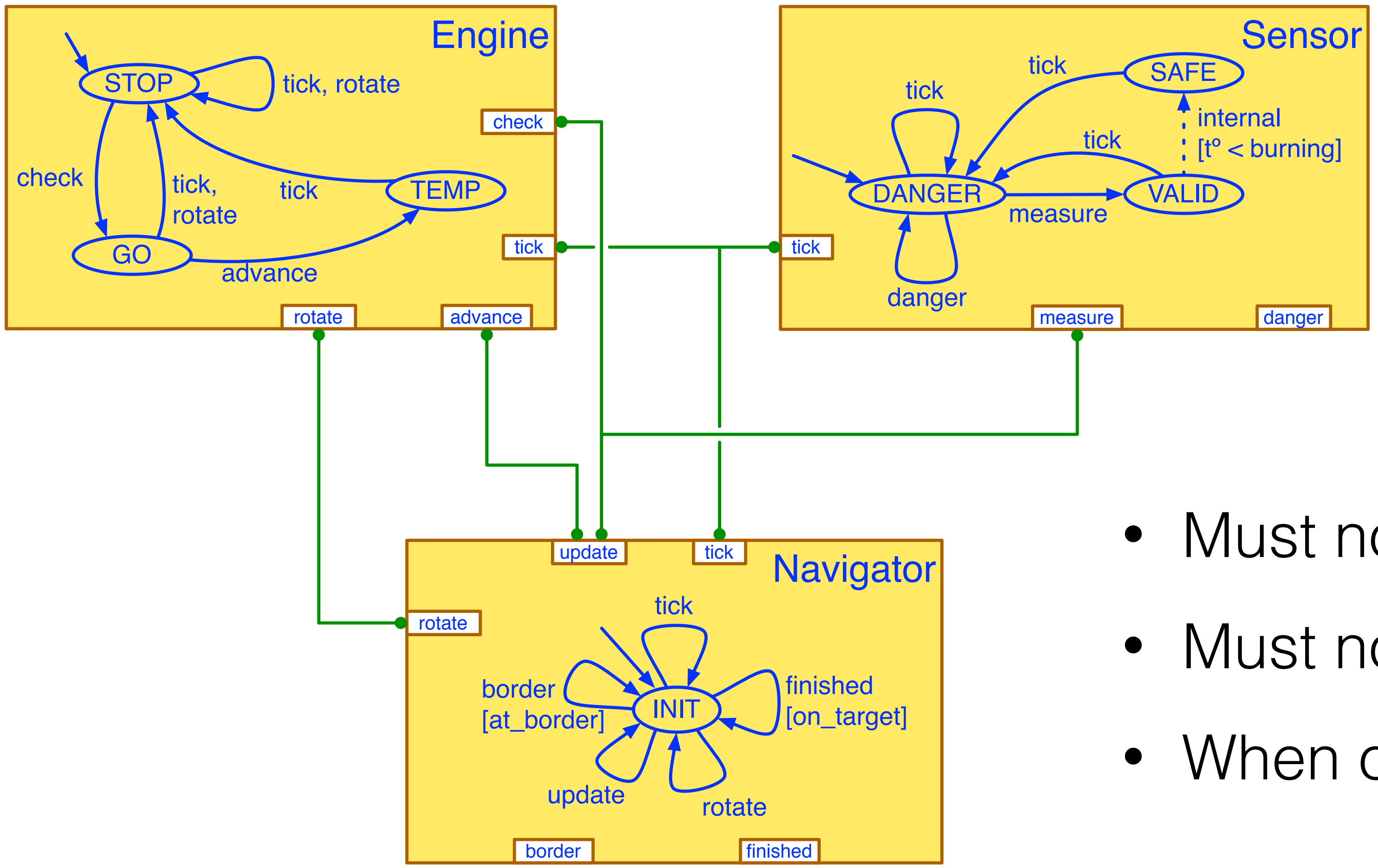
Connecting the robot



✓

- Must update navigation and sensor data at each move

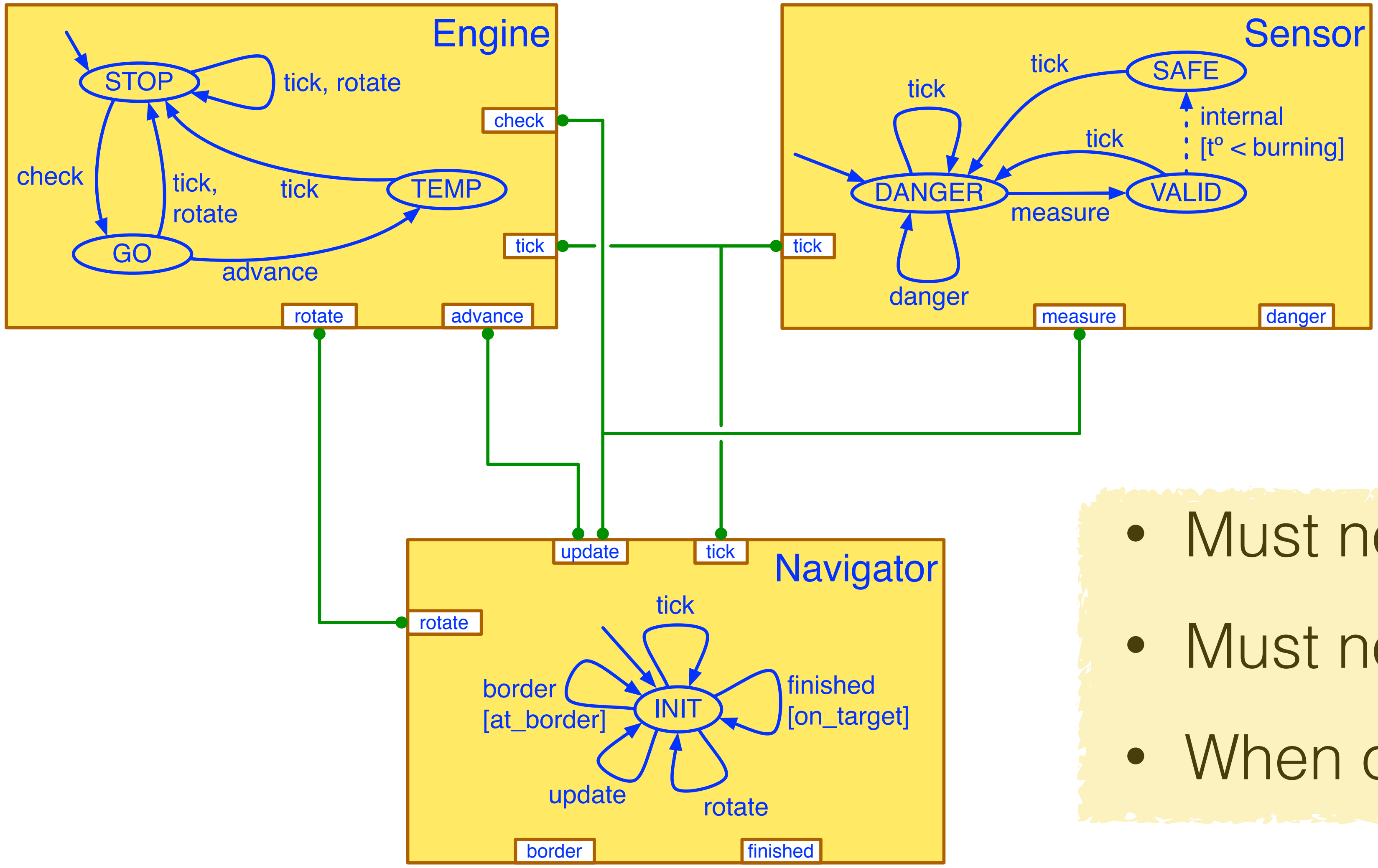
Connecting the robot



- Must not leave the region
- Must not move into burning areas
- When objective is found, must stop

```
priority p_rotate   c_rotate:*  < c_finished:*
priority p_advance1 c_advance:* < c_finished:*
priority p_advance2 c_advance:* < c_danger:*
priority p_advance3 c_advance:* < c_border:*
```

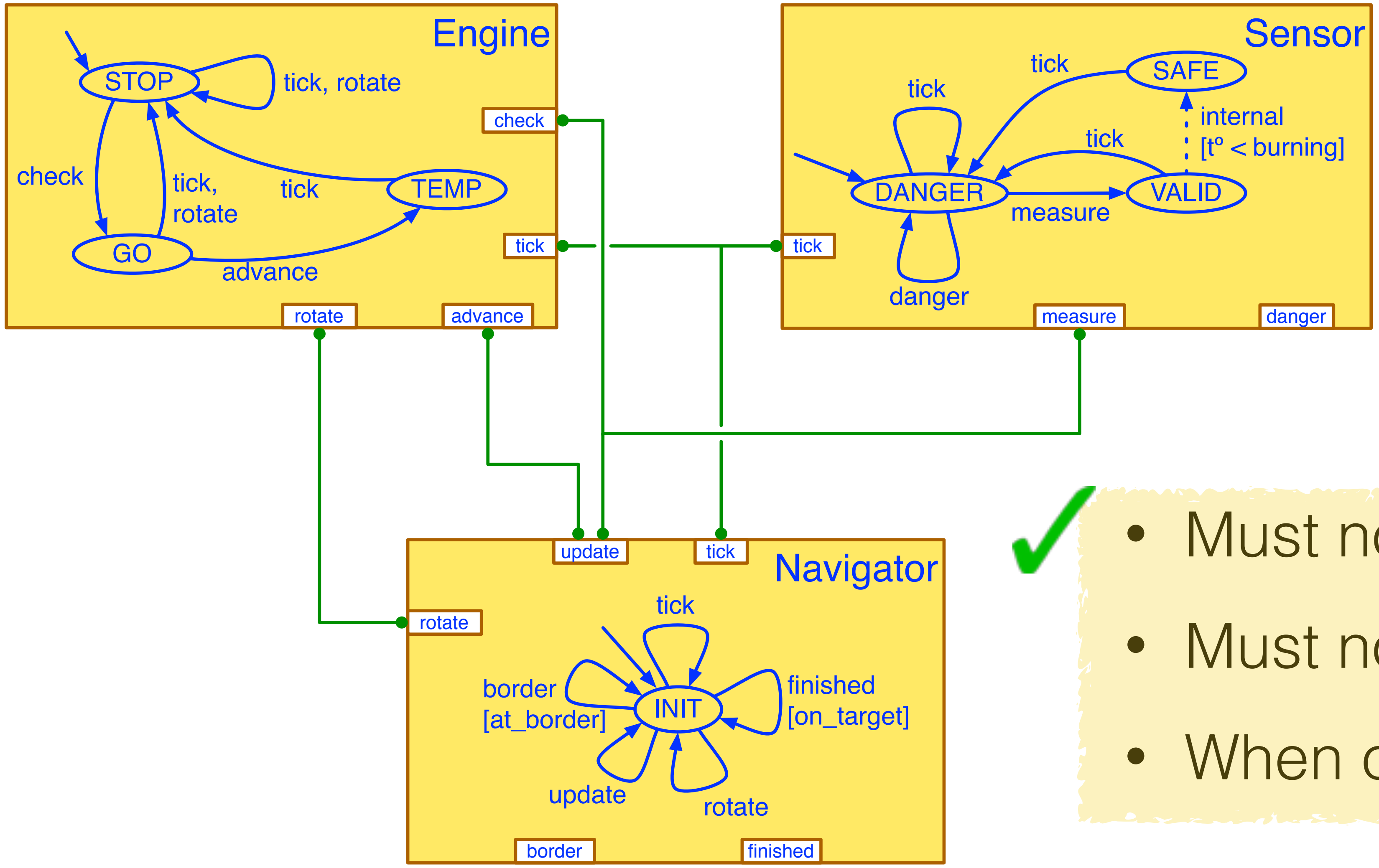
Connecting the robot



- Must not leave the region
- Must not move into burning areas
- When objective is found, must stop

```
priority p_rotate   c_rotate:*  < c_finished:*
priority p_advance1 c_advance:* < c_finished:*
priority p_advance2 c_advance:* < c_danger:*
priority p_advance3 c_advance:* < c_border:*
```

Connecting the robot

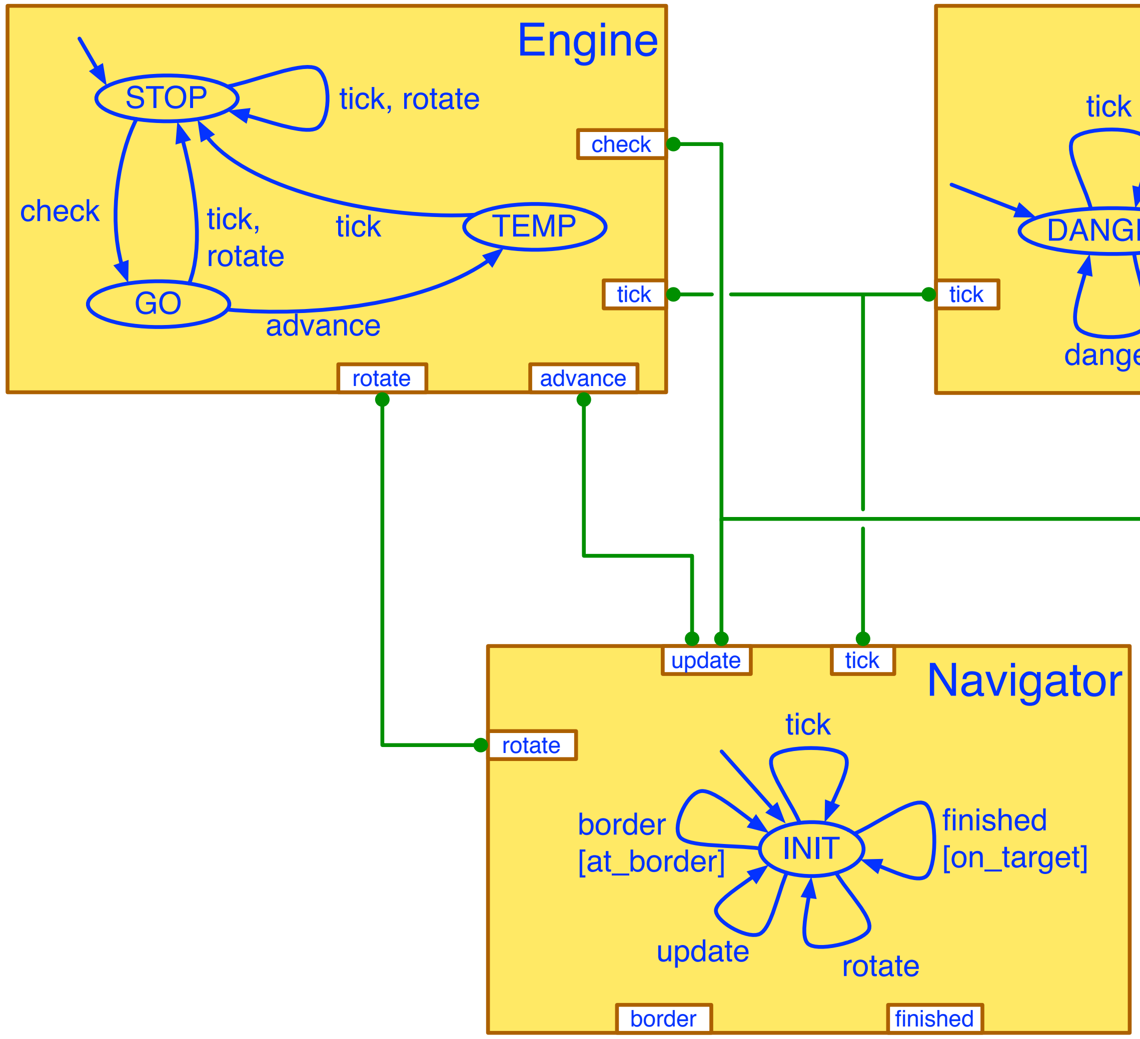


✓

- Must not leave the region
- Must not move into burning areas
- When objective is found, must stop

```
priority p_rotate   c_rotate:*  < c_finished:*
priority p_advance1 c_advance:* < c_finished:*
priority p_advance2 c_advance:* < c_danger:*
priority p_advance3 c_advance:* < c_border:*
```

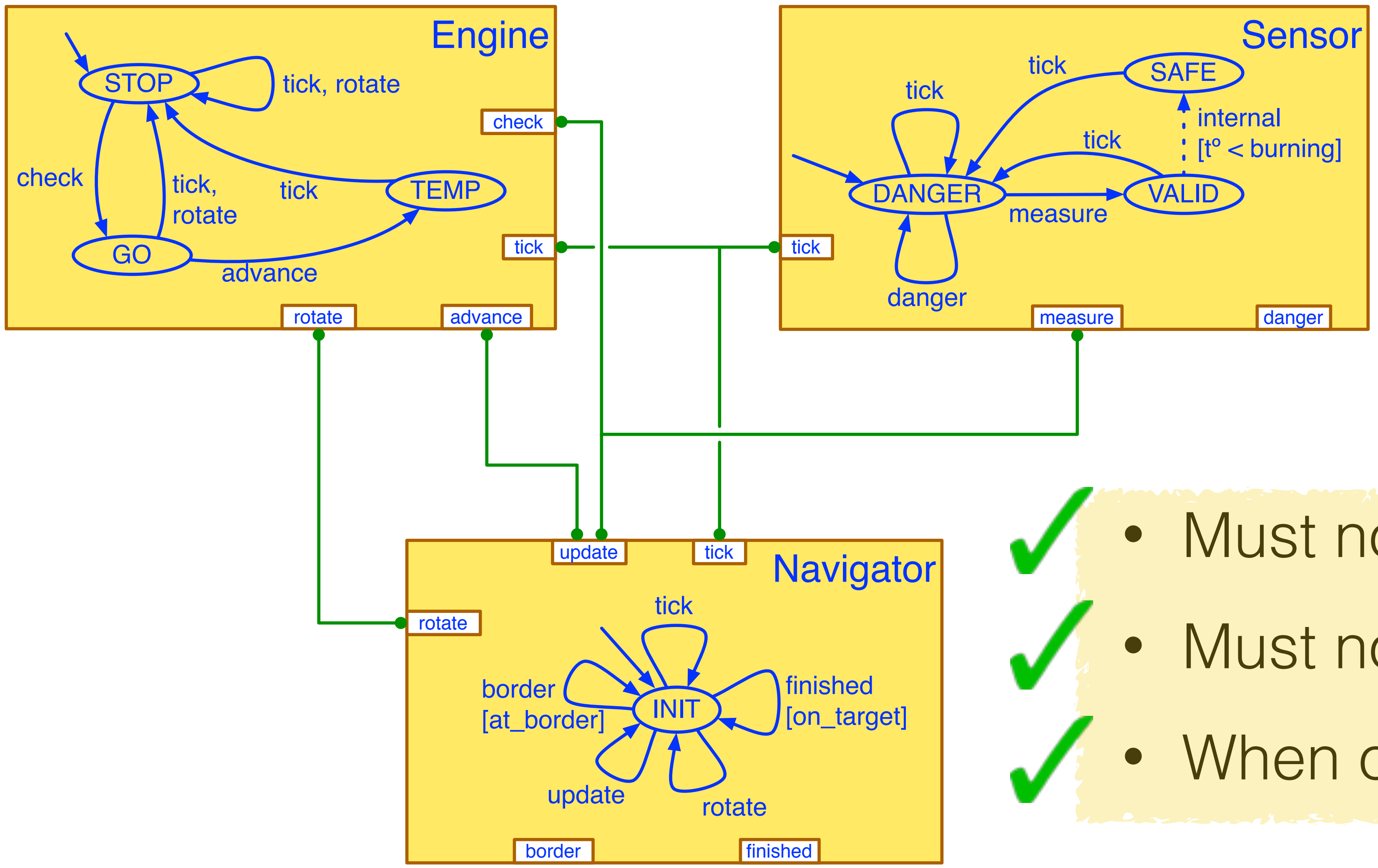
Connecting the robot



- ✓ Must not leave the region
- ✓ Must not move into burning areas
- When objective is found, must stop

```
priority p_rotate   c_rotate:*  < c_finished:*
priority p_advance1 c_advance:* < c_finished:*
priority p_advance2 c_advance:* < c_danger:*
priority p_advance3 c_advance:* < c_border:*
```

Connecting the robot



- ✓ Must not leave the region
- ✓ Must not move into burning areas
- ✓ When objective is found, must stop

```
priority p_rotate   c_rotate:*  < c_finished:*
priority p_advance1 c_advance:* < c_finished:*
priority p_advance2 c_advance:* < c_danger:*
priority p_advance3 c_advance:* < c_border:*
```

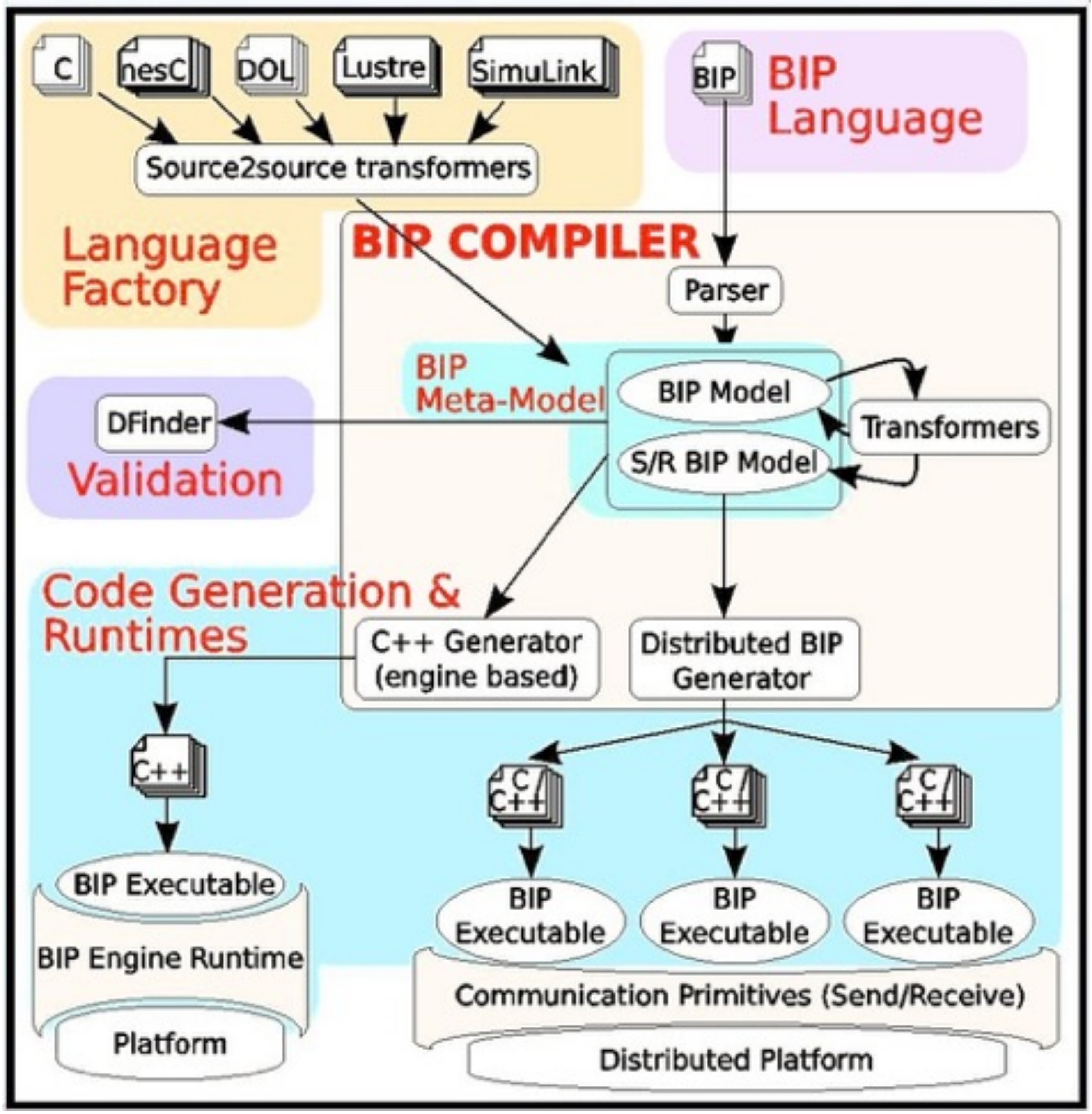
The final step

- Remove the model of the environment
- Replace “interface” elements with corresponding primitives
- Generate executable code from the remaining model

The flavours of BIP

Real-time, Dynamic, Java, Scala,…

Core BIP tool-set for ES

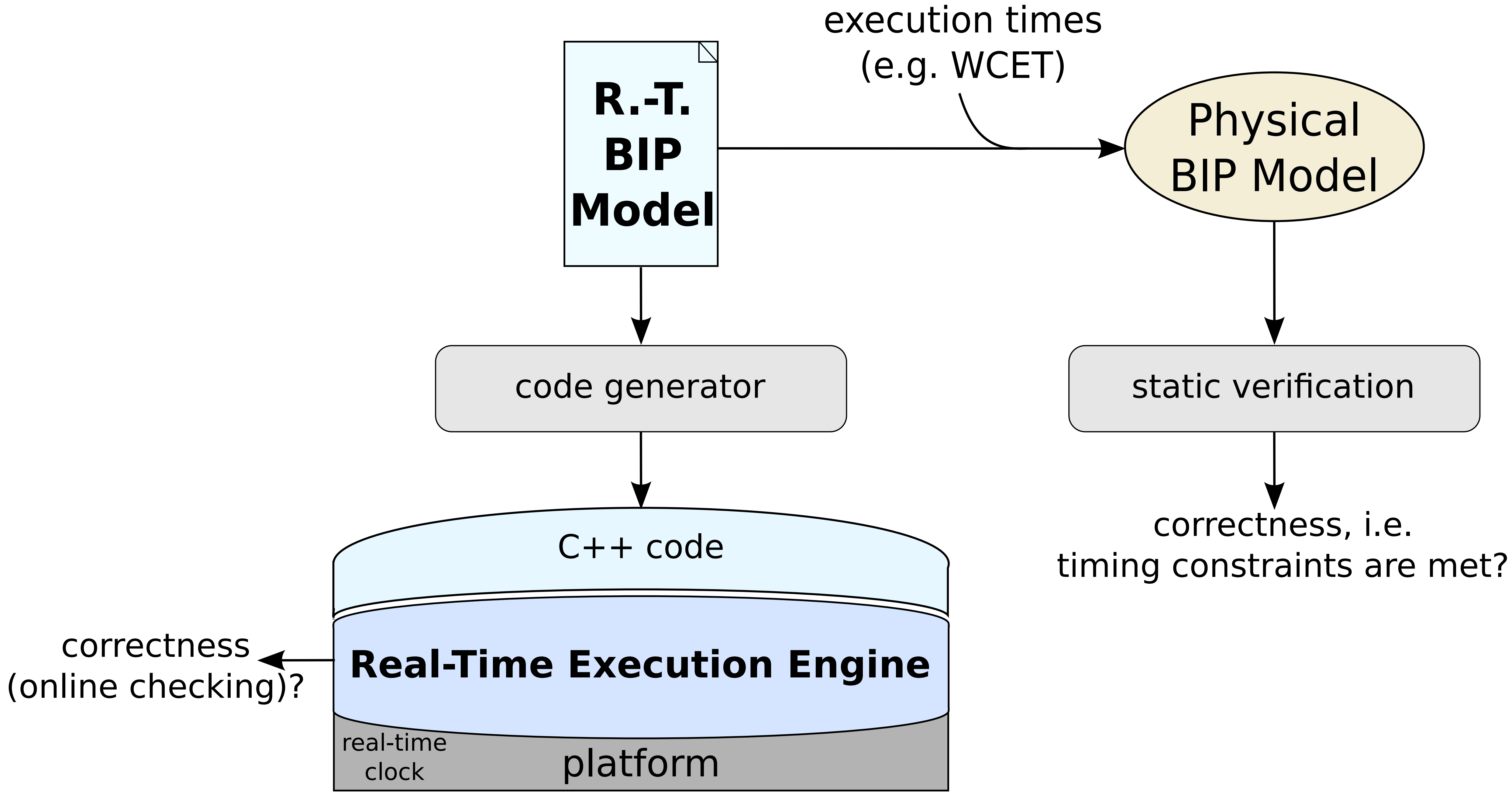


- Uses an EMF model as a pivot.
- Targets a C/C++ implementation.
- Complete code generation.

Real-time BIP

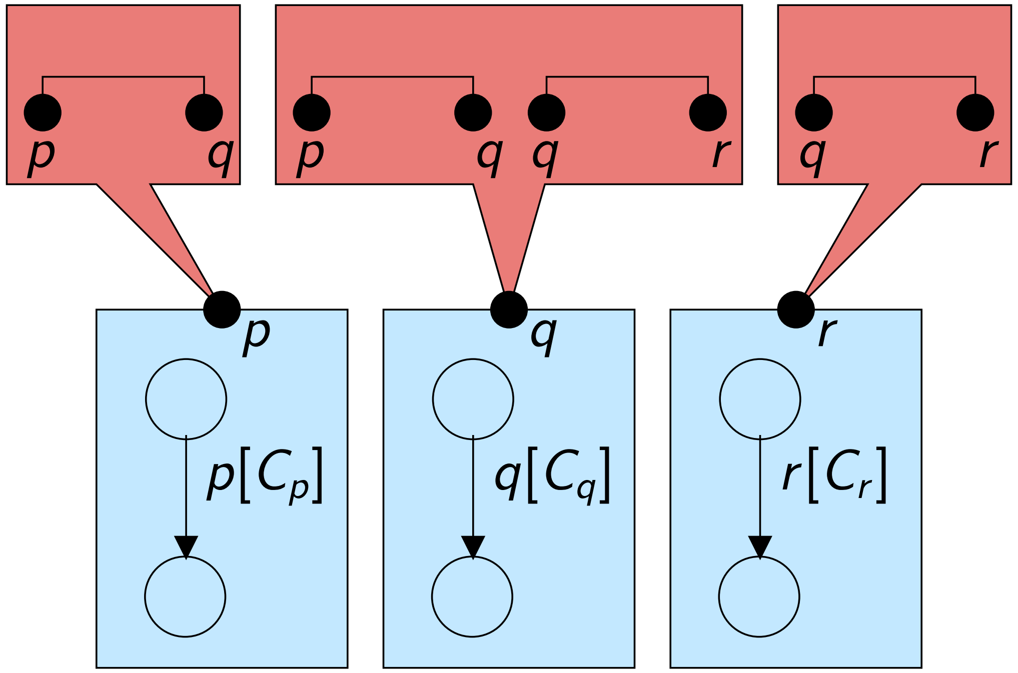
- Real-Time extension of the BIP language and tools
  - **abstract** model: **timed** automata representing user requirements
  - real-time execution on the target platform (actual execution times)
  - static verification for known properties on execution times

RT-BIP methodology



slide courtesy of
Jacques Combaz

Dynamic BIP (Dy-BIP)

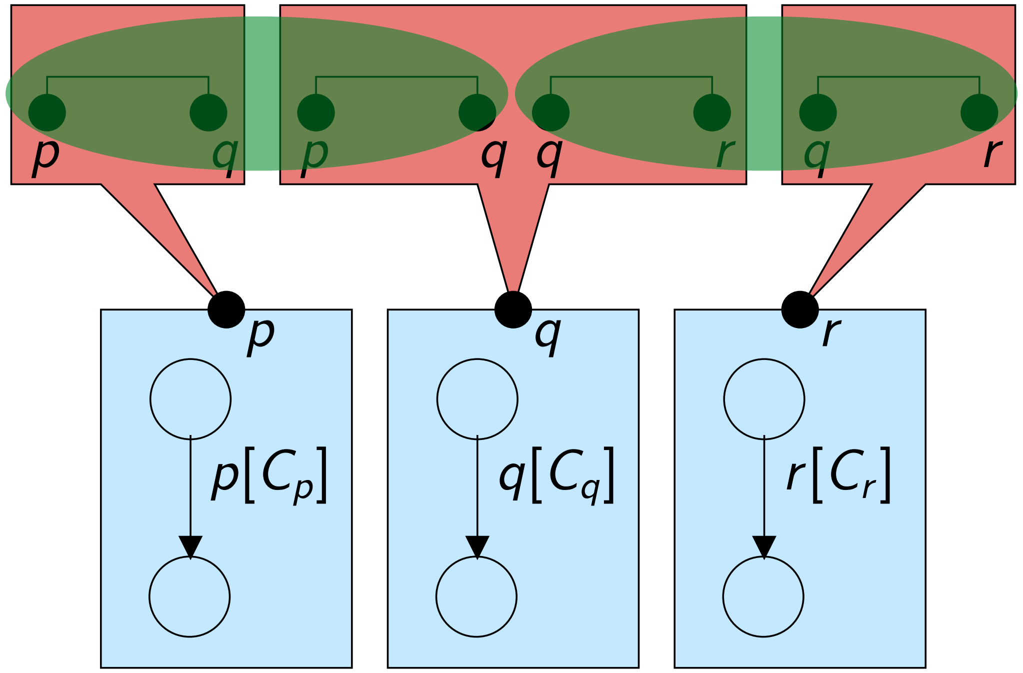


M. Bozga, M. Jaber, N. Maris, J. Sifakis. *Modeling Dynamic Architectures using Dy-BIP* [SC'12]

- Dynamic interconnection is necessary for modern systems
  - web services, robotic systems, reconfigurable middleware, wireless sensor networks, fault-tolerant systems, etc.

- Architecture is the composition of dynamically changing architecture constraints defined by components
  - A feasible interaction satisfies the constraints of all the involved components.

Dynamic BIP (Dy-BIP)

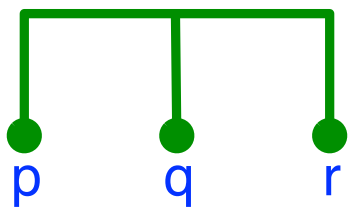


M. Bozga, M. Jaber, N. Maris, J. Sifakis. *Modeling Dynamic Architectures using Dy-BIP* [SC'12]

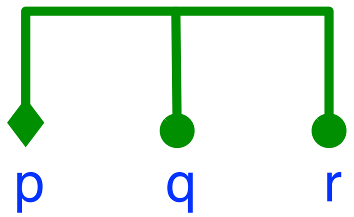
- Dynamic interconnection is necessary for modern systems
  - web services, robotic systems, reconfigurable middleware, wireless sensor networks, fault-tolerant systems, etc.
- Architecture is the composition of dynamically changing architecture constraints defined by components
  - A feasible interaction satisfies the constraints of all the involved components.

Interaction constraints

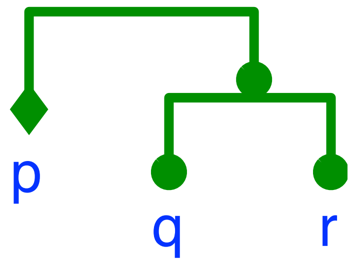
- Sets of ports can be characterised by boolean constraints
  - **p ⇒ false** — p is absent from the interaction; **p ⇒ true** — no constraints



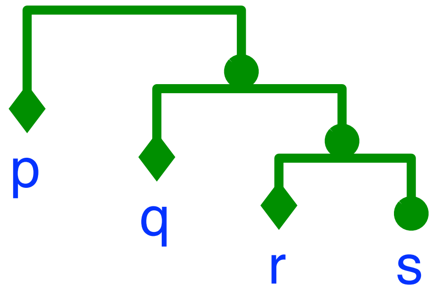
Strong synchronisation: pqr
p ⇒ q, q ⇒ r, r ⇒ p



Broadcast: p + pq + pr + pqr
q ⇒ p, r ⇒ p



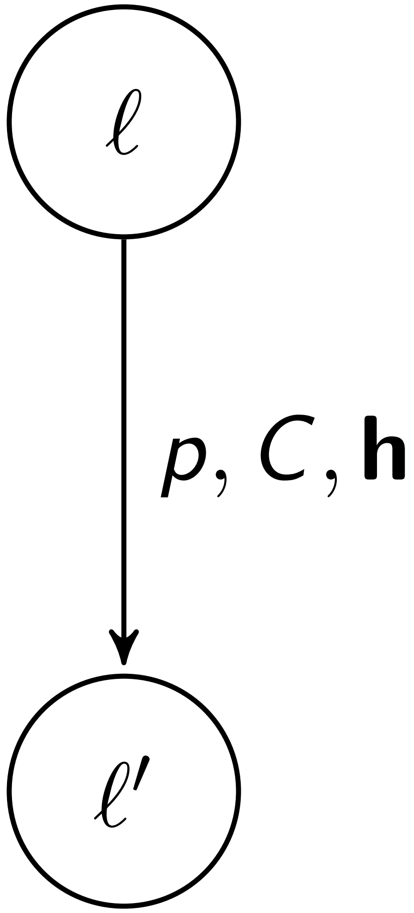
Atomic broadcast: p + pqr
q ⇒ pr, r ⇒ pq



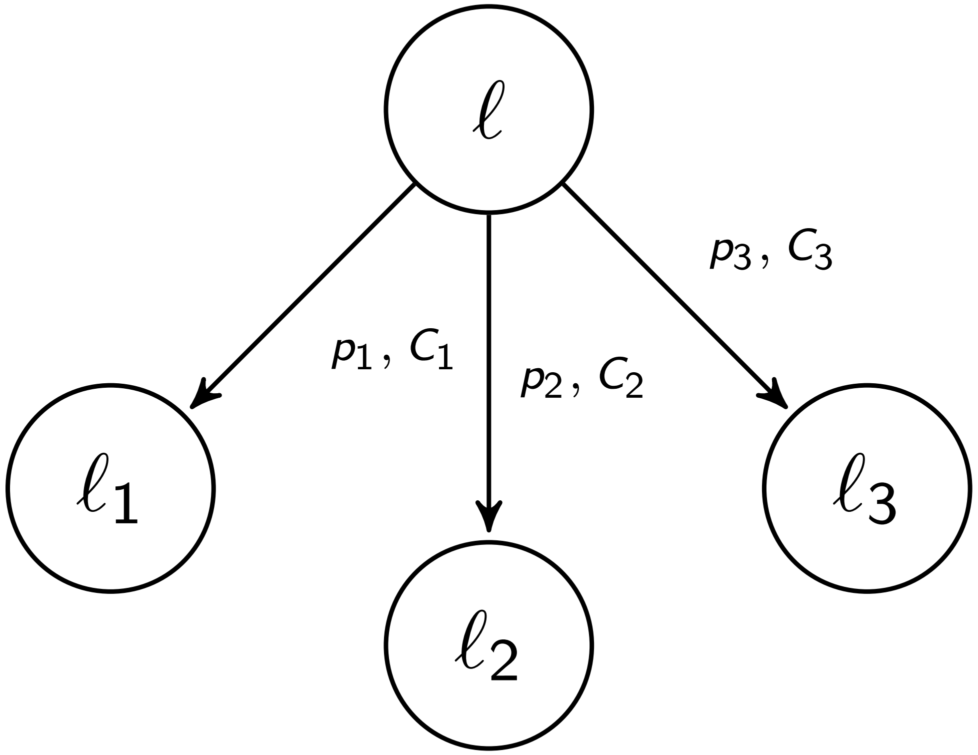
Causal chain: p + pq + pqr + pqrs
q ⇒ p, r ⇒ q, s ⇒ r

Transition constraints

- A transition $(\ell, p, C, \mathbf{h}, \ell')$
  - $\ell, \ell' \in L$, are the source, target locations
  - $p \in P$, is the port offered for interaction
  - $C \in \mathcal{C}$, is the interaction constraint
  - $\mathbf{h} \subseteq H$, is the set of history variables to be updated



Location constraints



The location constraint characterises the contribution of the component to a global interaction:

$$CL(\ell, s) = \bigvee_{\ell \xrightarrow{p, C, \mathbf{h}} \ell'} \left(p \wedge C(s) \wedge \bigwedge_{p' \in P \setminus \{p\}} \neg p' \right) \vee \bigwedge_{p \in P} \neg p$$

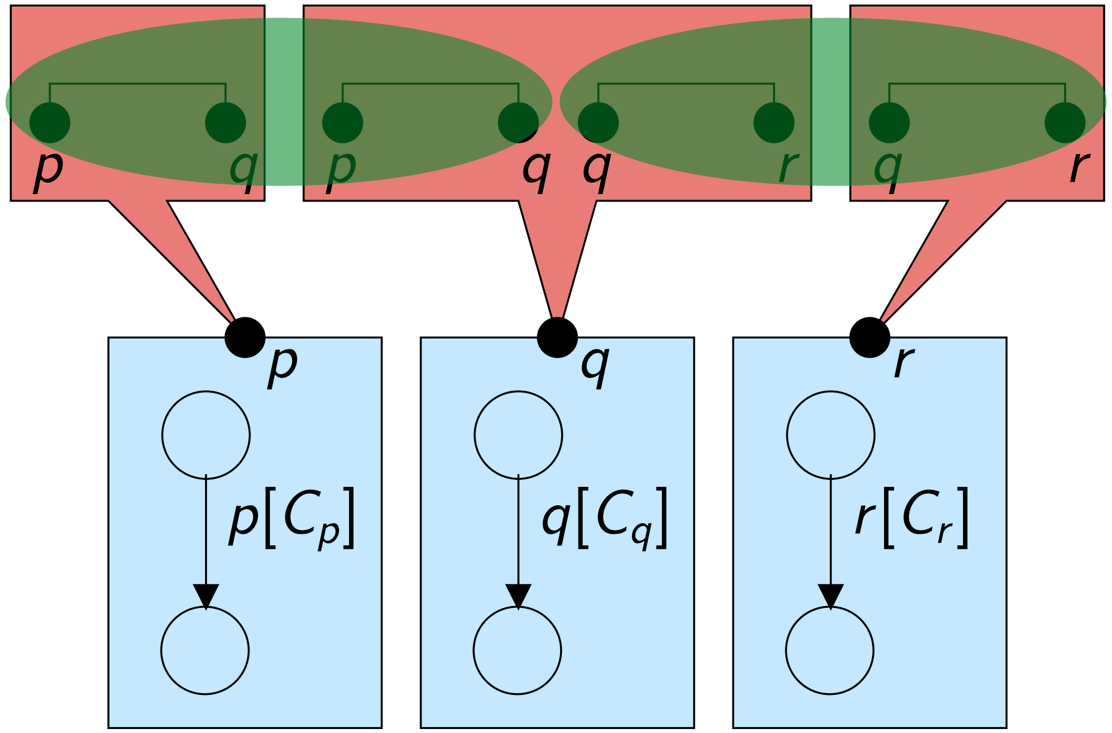
Symbolic execution engine

- Atoms send location contraints encoded as BDDs

- The engine performs the global conjunction

- If satisfiable, it picks one (maximal) solution

- Notifies the atoms

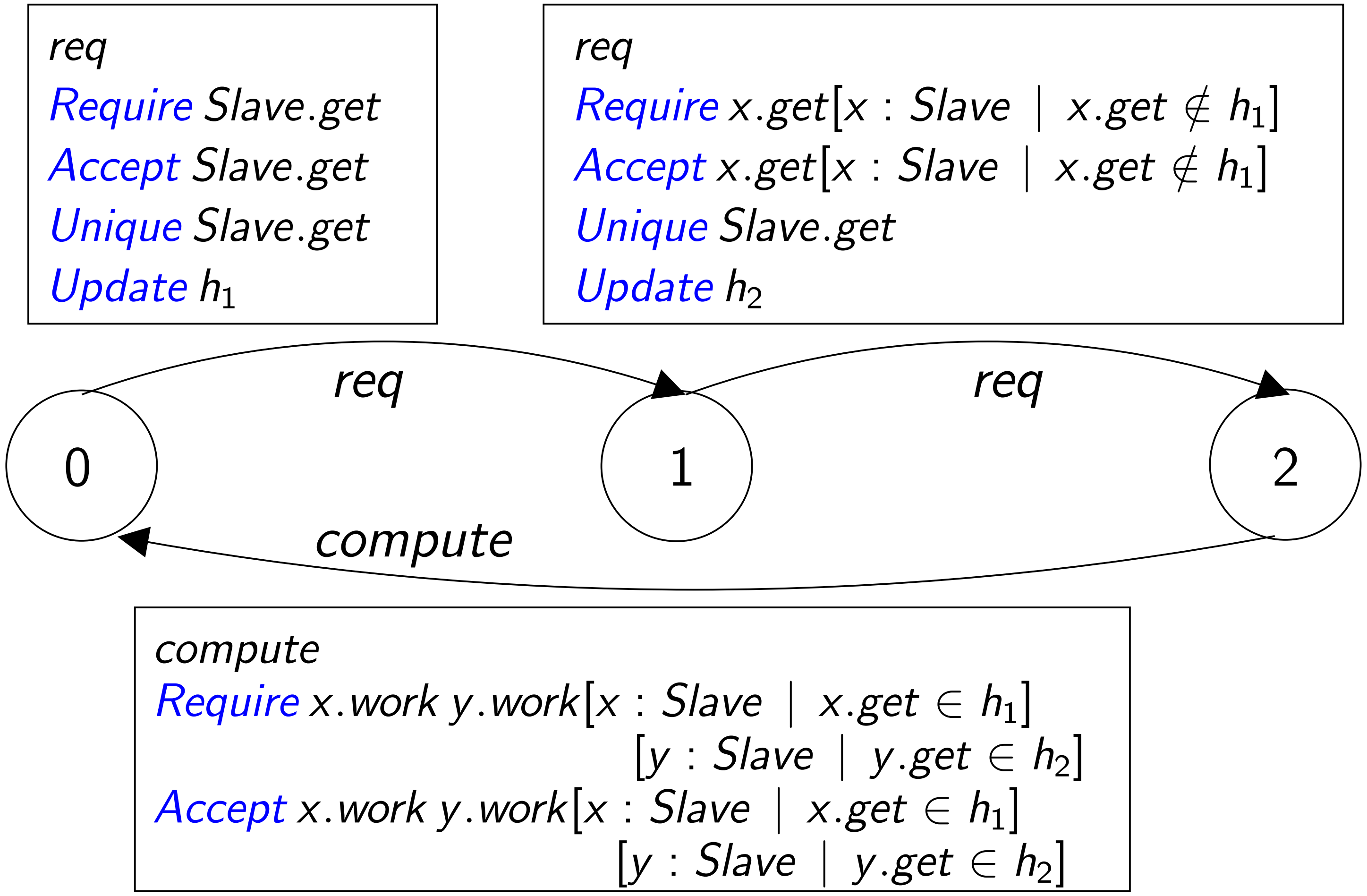


Macro notation

- Main types of constraints for a given port p
  - Causal constraints:
    of the form $p \Rightarrow (q \wedge r) \vee (s \wedge t)$, meaning that one of $q\,r$ and $s\,t$ is required
  - Acceptance constraints:
    of the form $p \Rightarrow \neg q$, meaning that q is forbidden

- Macro notation for constraints:
  Let A, B be component types with instances a_1, a_2, a_3, b_1, b_2
  - Require A.q *translates to*: $p \Rightarrow a_1.q \vee a_2.q \vee a_3.q$
  - Accept A.r, B.q *translates to*: $p \Rightarrow \bigwedge_{t \not\in \{p,\, a1.r,\, a2.r,\, a3.r,\, b1.q,\, b2.q\}} \neg t$
  - Unique A.q *translates to*: $p \Rightarrow (a_1.q\ \neg a_2.q\ \neg a_3.q) \vee$
    $(\neg a_1.q\ a_2.q\ \neg a_3.q) \vee (\neg a_1.q\ \neg a_2.q\ a_3.q)$

Example: Master and Slaves

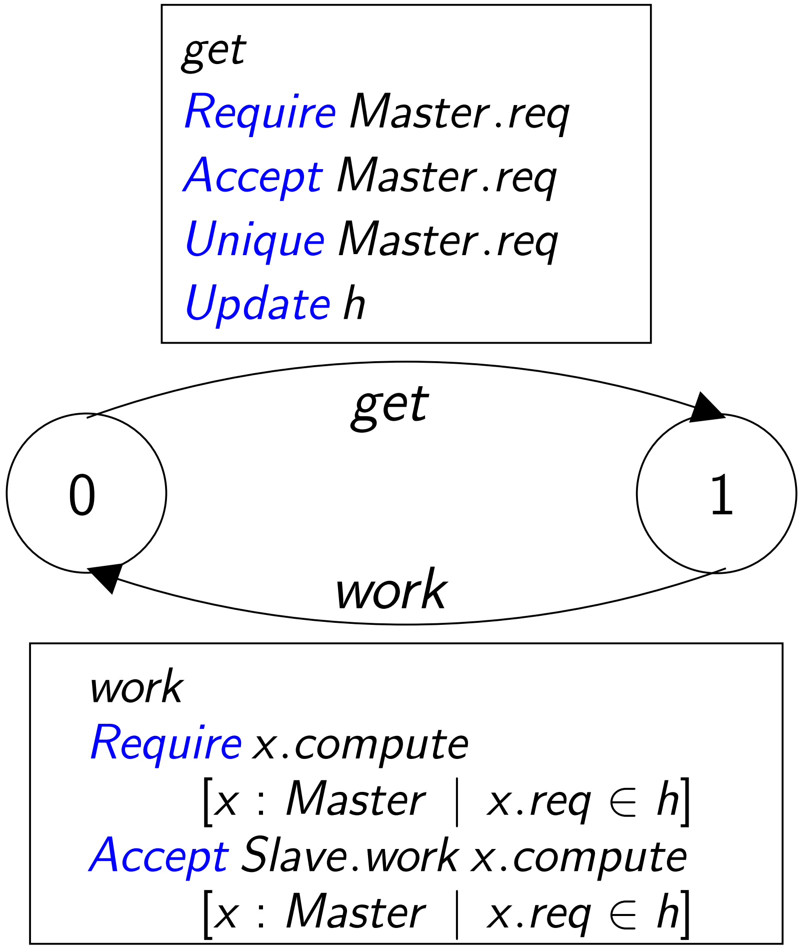


req
Require $Slave.get$
Accept $Slave.get$
Unique $Slave.get$
Update h_1

req
Require $x.get[x : Slave \mid x.get \notin h_1]$
Accept $x.get[x : Slave \mid x.get \notin h_1]$
Unique $Slave.get$
Update h_2

compute
Require $x.work\ y.work[x : Slave \mid x.get \in h_1]$
$[y : Slave \mid y.get \in h_2]$
Accept $x.work\ y.work[x : Slave \mid x.get \in h_1]$
$[y : Slave \mid y.get \in h_2]$

Master Type



get
Require $Master.req$
Accept $Master.req$
Unique $Master.req$
Update h

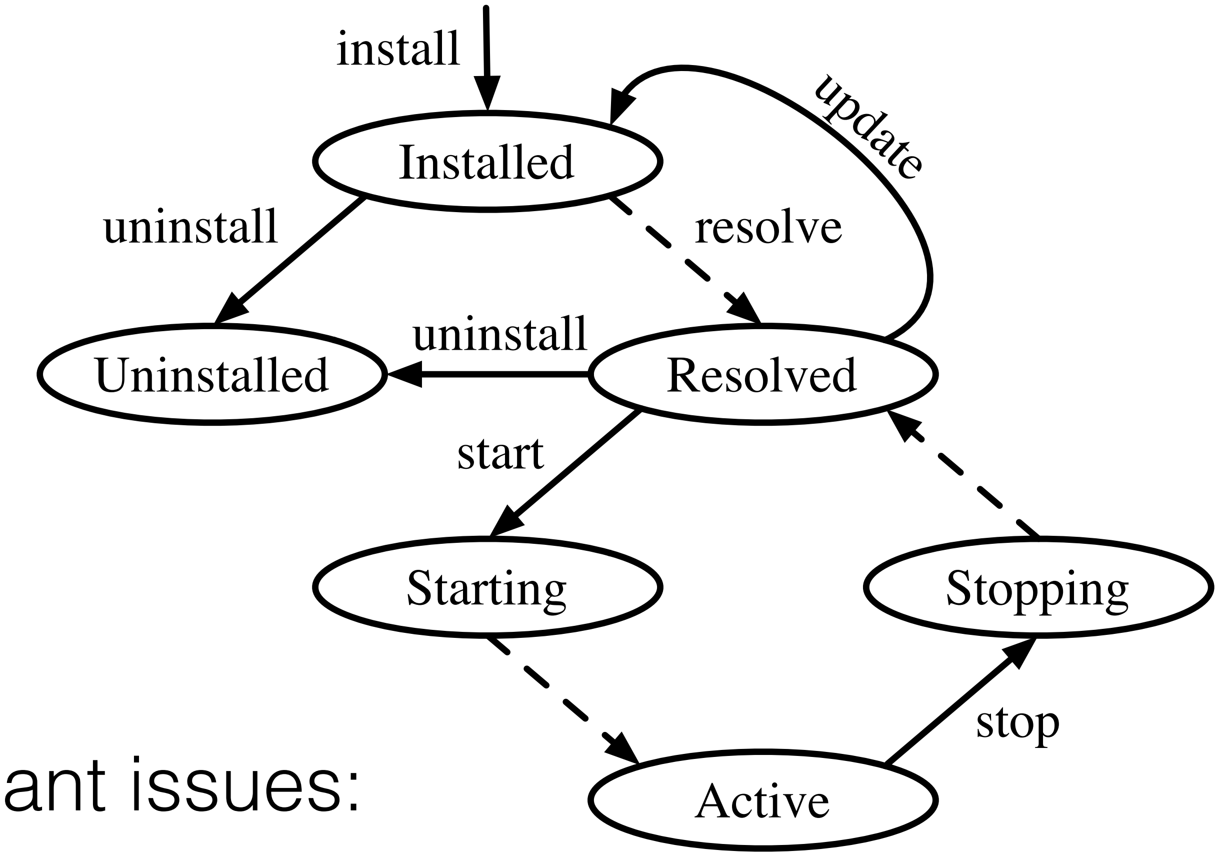
work
Require $x.compute$
$[x : Master \mid x.req \in h]$
Accept $Slave.work\ x.compute$
$[x : Master \mid x.req \in h]$

Slave Type

Each master sends requests sequentially to two slaves, and then performs some computation involving both of them.

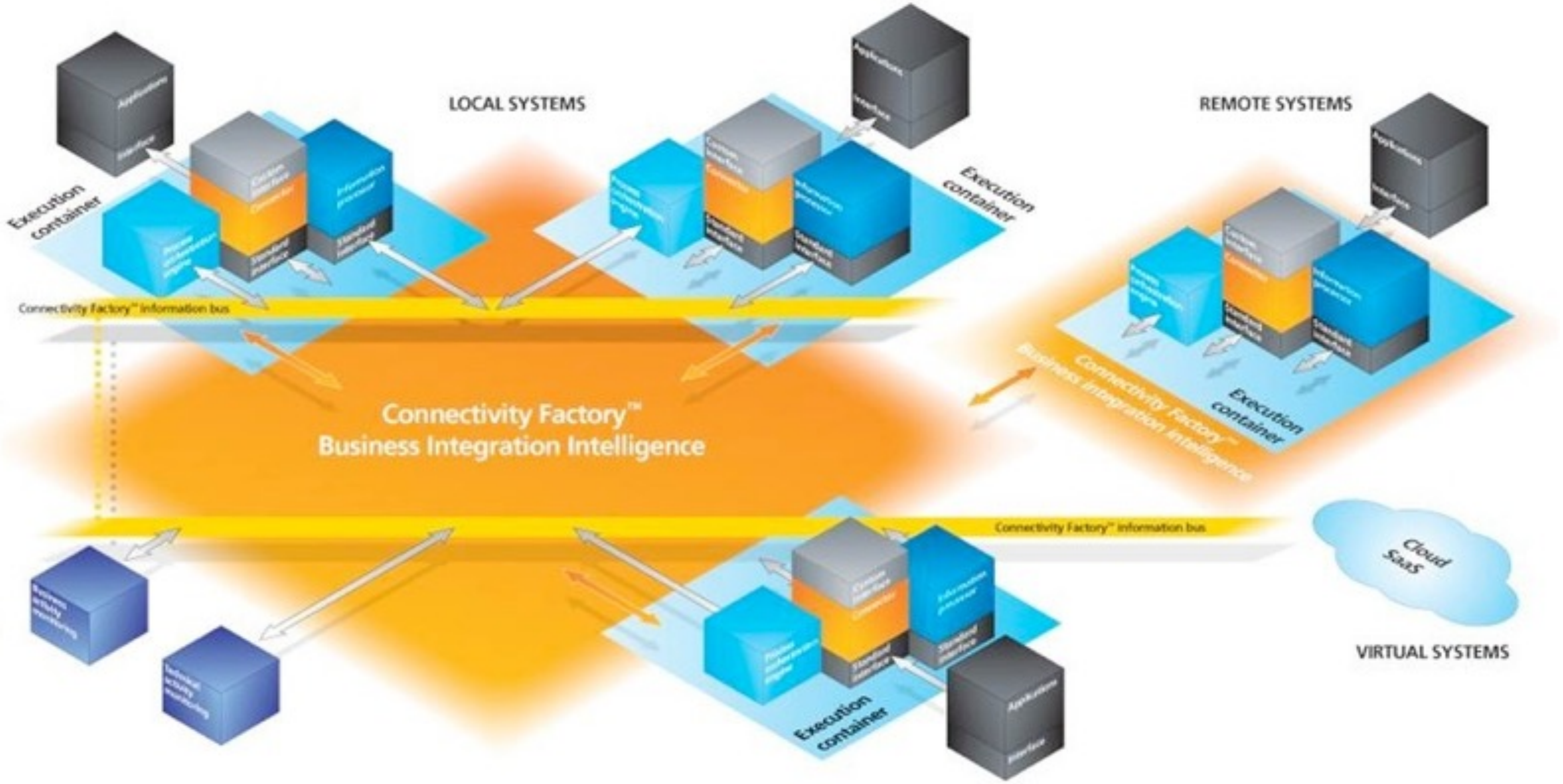
BIP coordination for Java

S. Bliudze, A. Mavridou, R. Szymanek, A. Zolotukhina. *Coordination of software components with BIP: application to OSGi*. [MiSE 2014]



- BIP framework addresses three important issues:
  - High-level abstraction for synchronisation
  - Atomicity of state manipulation (e.g. as opposed to threads)
  - Separation of concerns: coordination is defined independently of component code
- State-of-practice: AKKA — asynchronous communication between actors
- Coordination mechanisms must not disrupt the existing software stack

Use case: Camel Routes


LOCAL SYSTEMS
REMOTE SYSTEMS
Applications
Interface
Execution container
Execution container
Execution container
Execution container
Connectivity Factory™ information bus
Connectivity Factory™ Business Integration Intelligence
Connectivity Factory™ Business Integration Intelligence
Connectivity Factory™ information bus
Cloud SaaS
VIRTUAL SYSTEMS
Business activity monitoring
Technical activity monitoring

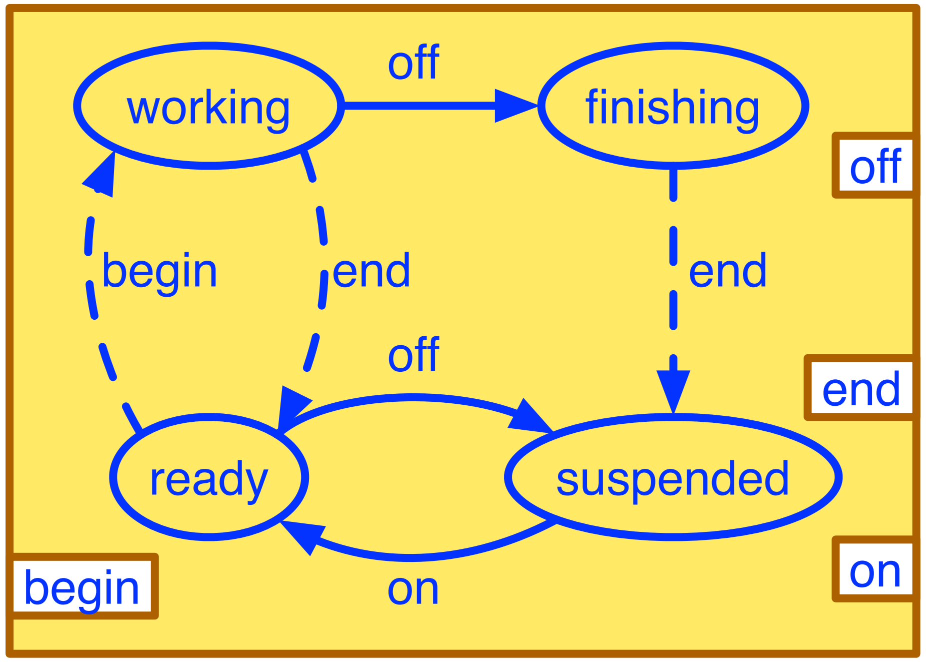
- Many independent routes share memory
  - We have to control the memory usage
  - e.g., by limiting to only a safe number of routes simultaneously

Camel routes

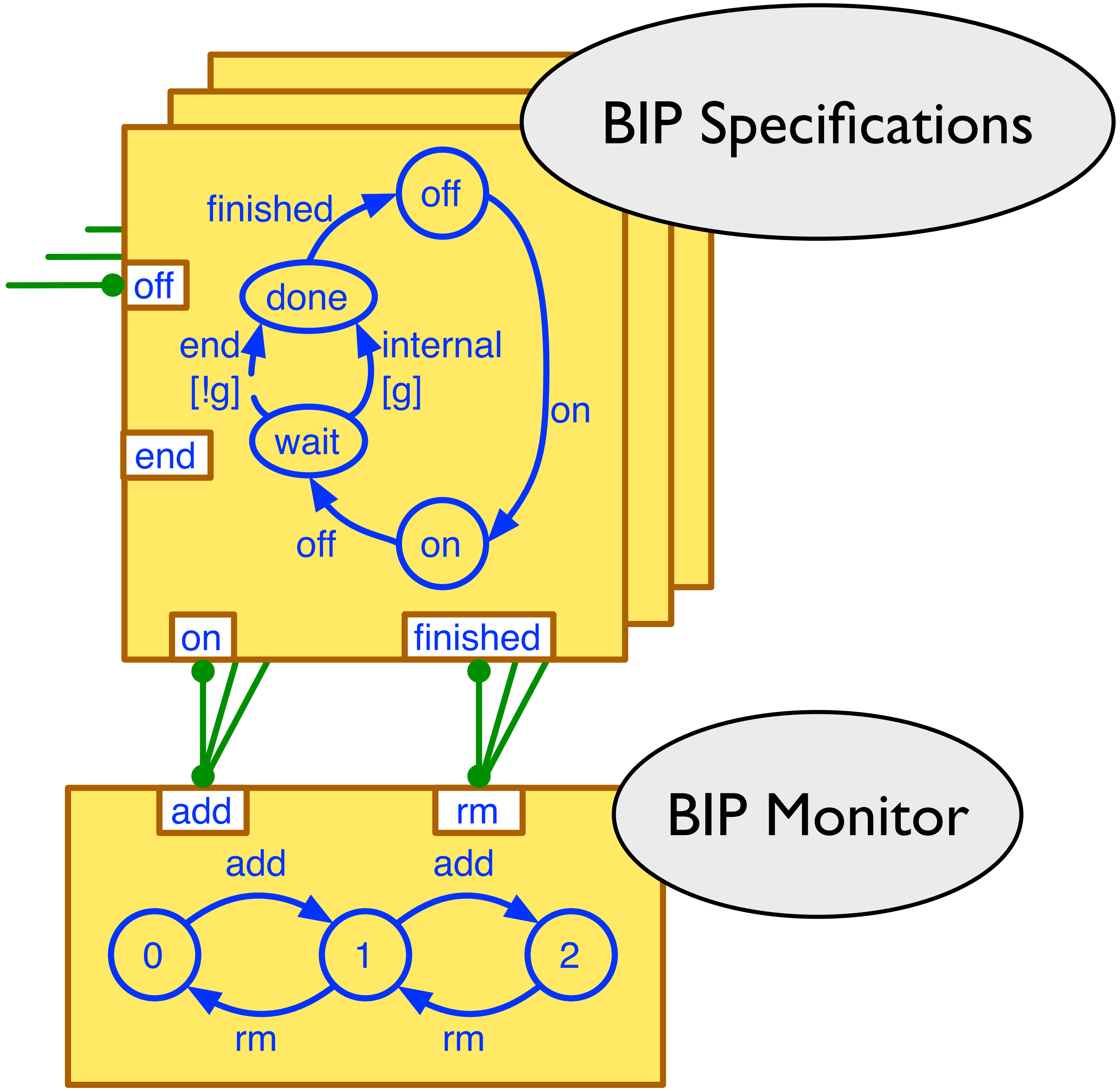
```
public class RouteBuilder(...)
{
  from(…).process(…).to(…);
}
```

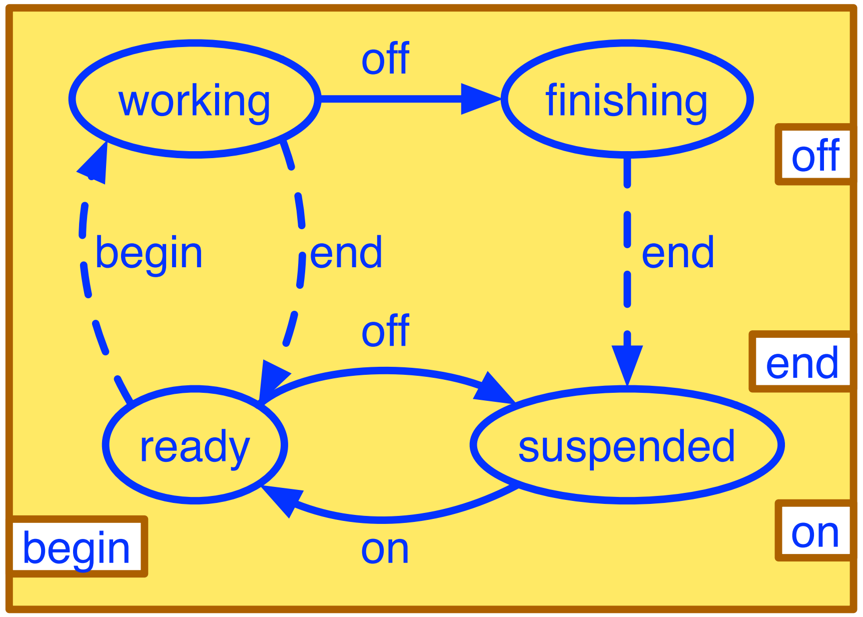
Camel API: suspendRoute and resumeRoute

- Transition types:
  - **Enforceable** — can be controlled by the Engine
  - **Spontaneous** — inform about uncontrollable external events



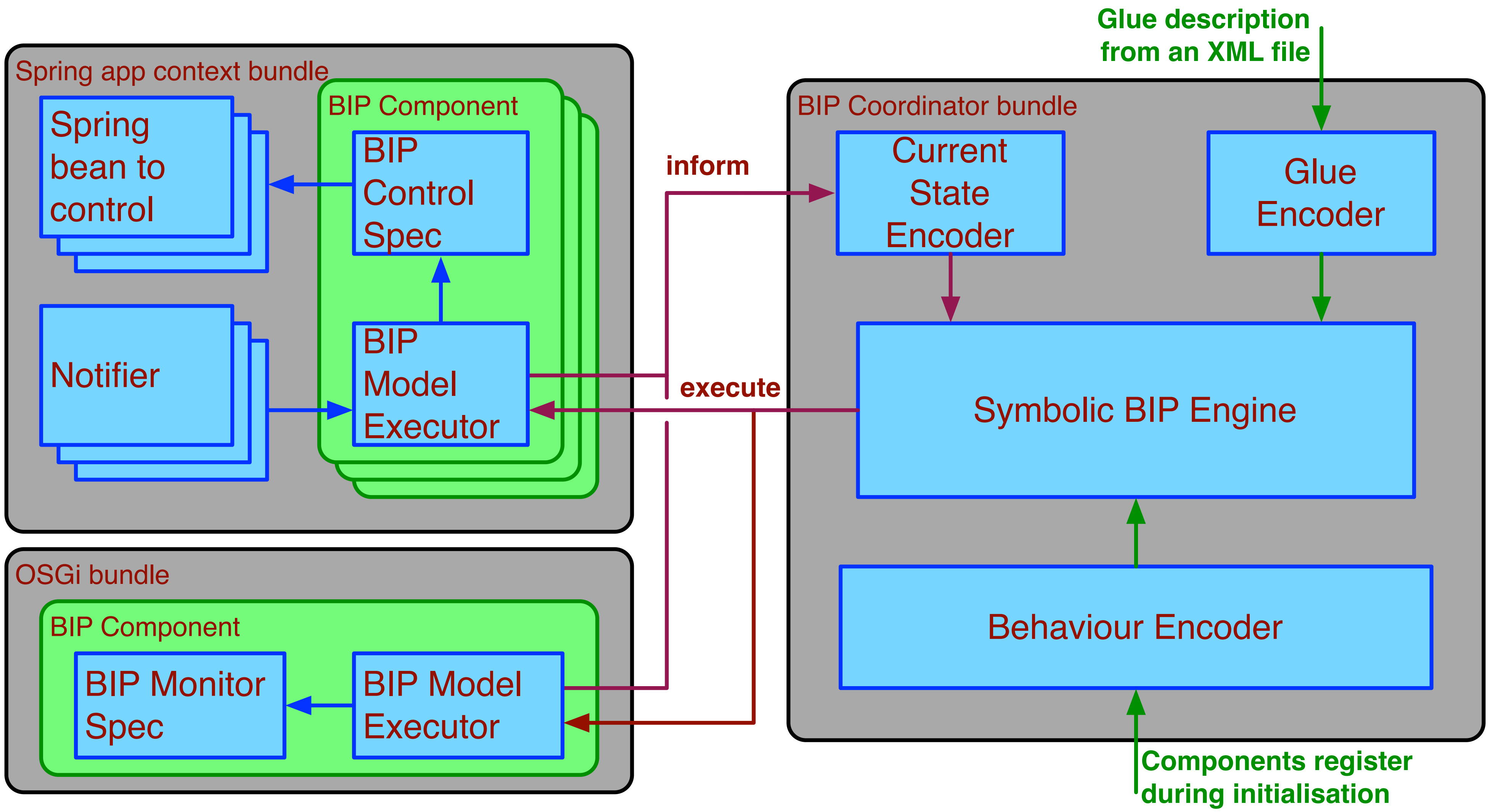
Use case: BIP model





The Monitor component limits the number of active routes to two

Implemented architecture

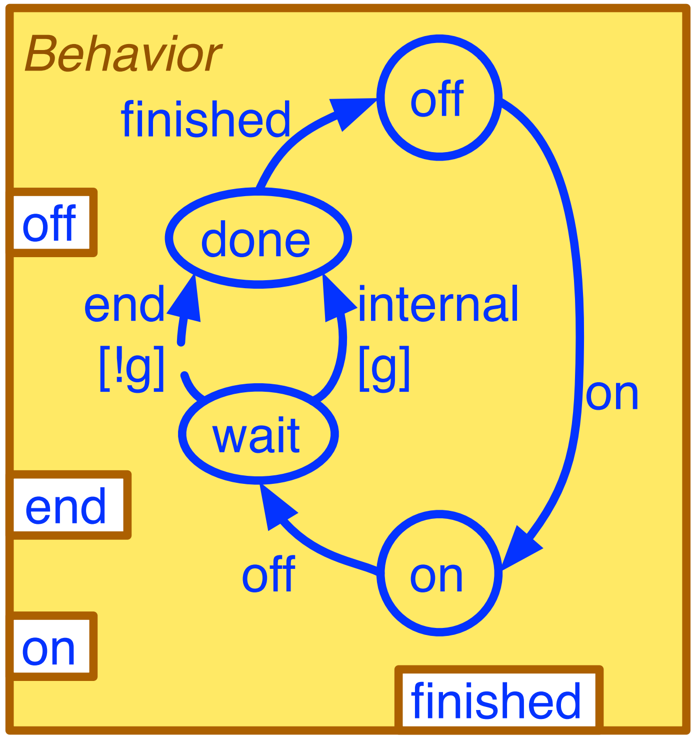


Arrows

- Blue — API calls between model and entity
- Red — OSGi-managed through published services
- Green — called once at initialisation phase

BIP Specification: Ports, Initial state

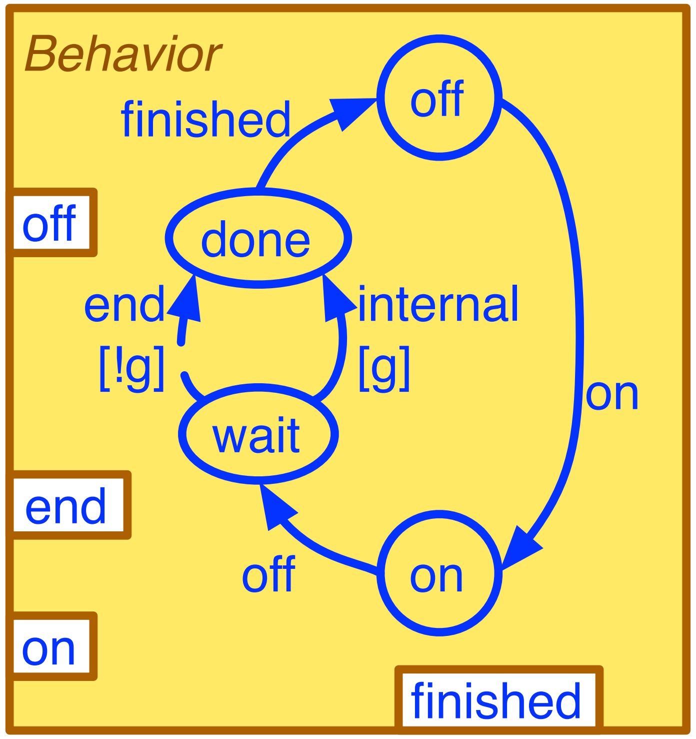
```
@bipPorts({
  @bipPort(name = "end", type = "spontaneous"),
  @bipPort(name = "off", type = "enforceable"),
  …
})

@bipComponentType(
  initial = "off",
  name = "org.bip.spec.switchableRoute")

public class SwitchableRoute
   implements CamelContextAware,
              InitializingBean,
              DisposableBean
{ … }
```



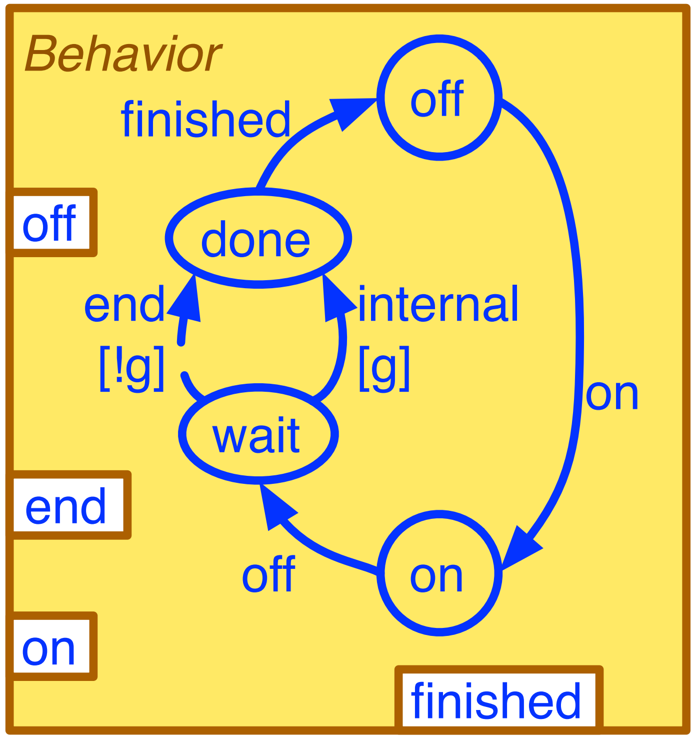
BIP Specification: Transitions

```
@bipTransition(name = "off",
  source = "on", target = "wait", guard = "")

public void stopRoute() throws Exception {
    camelContext.suspendRoute(routeId);
}
```

- Transition annotations
  - Label, i.e. a port, declared by `@bipPort`
  - Source and target states
  - Guard expression

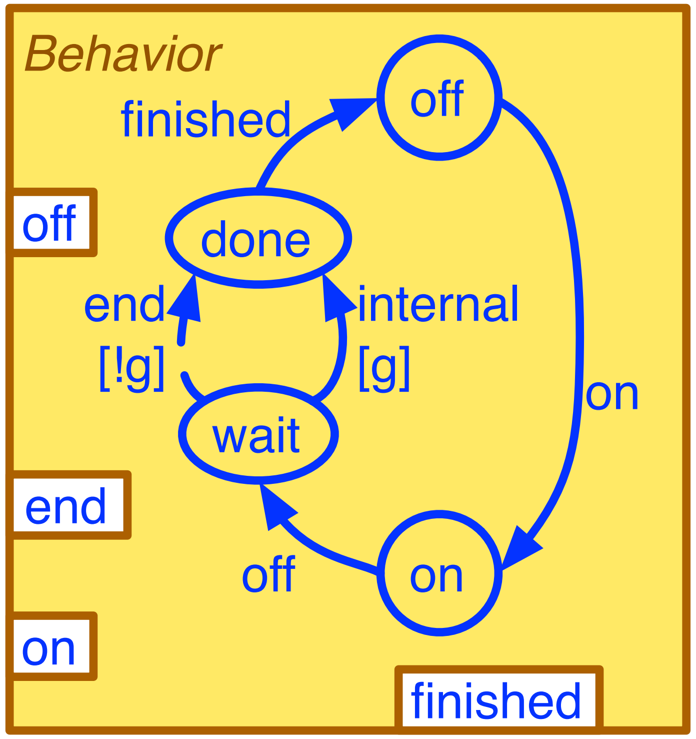


BIP Specification: Guards



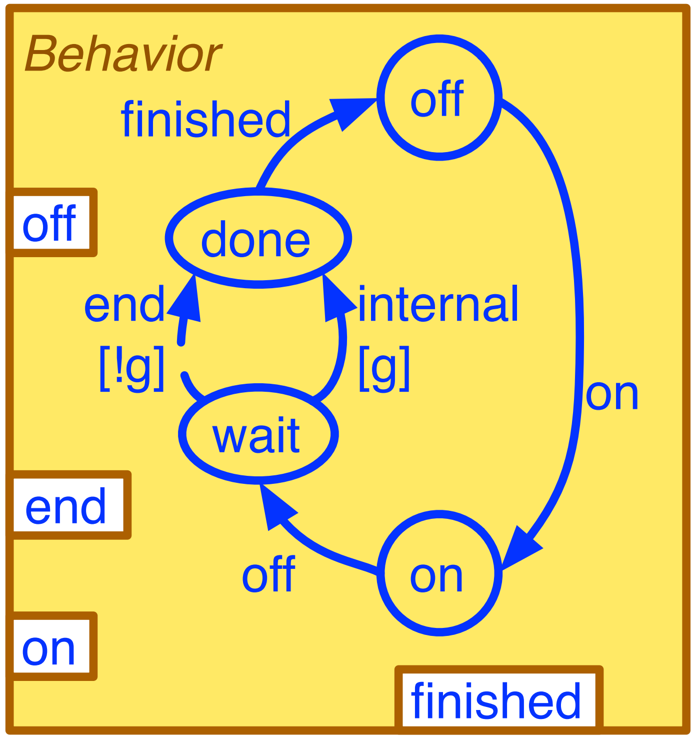
BIP Specification: Guards

```
@bipTransition(name = "end",
  source = "wait", target = "done",
  guard = "!isFinished")
public void spontaneousEnd() throws Exception { … }

@bipTransition(name = "",
  source = "wait", target = "done",
  guard = "isFinished")
public void internalEnd() throws Exception { … }
```



BIP Specification: Guards

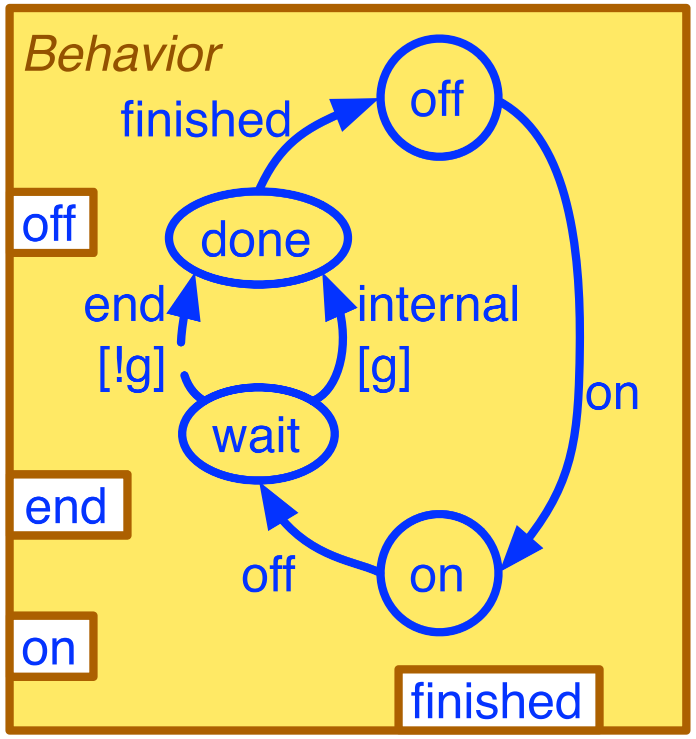
```
@bipTransition(name = "end",
  source = "wait", target = "done",
  guard = "!isFinished")
public void spontaneousEnd() throws Exception { … }

@bipTransition(name = "",
  source = "wait", target = "done",
  guard = "isFinished")
public void internalEnd() throws Exception { … }

@bipGuard(name = "isFinished")
public boolean isFinished() {
  CamelContext cc = camelContext;
  return
    cc.getInflightRepository().size(
      cc.getRoute(routeId).getEndpoint()
    ) == 0;
}
```



BIP Component interface

```
public interface BIPComponent extends BIPSpecification
{
   void execute(String portID);
   void inform(String portID);
}
```

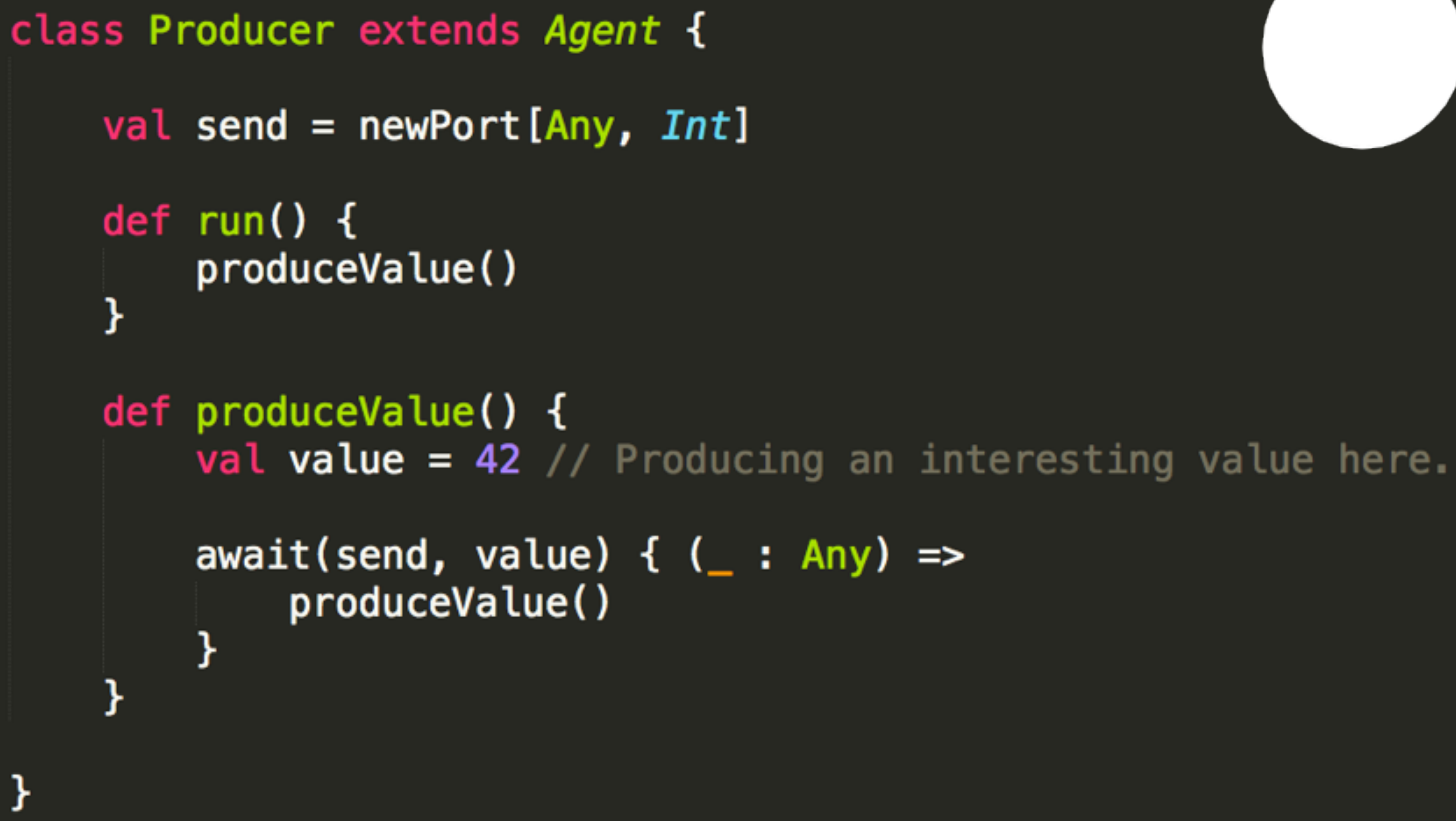
- Interface methods:
  - execute — called by the Engine to execute an enforceable transition
  - inform — called by Notifiers to inform about spontaneous events



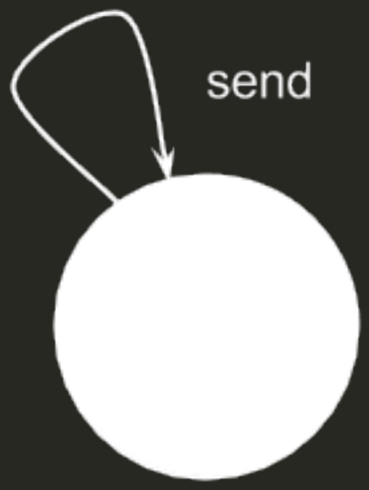
BIP in functional languages

Example in Scala

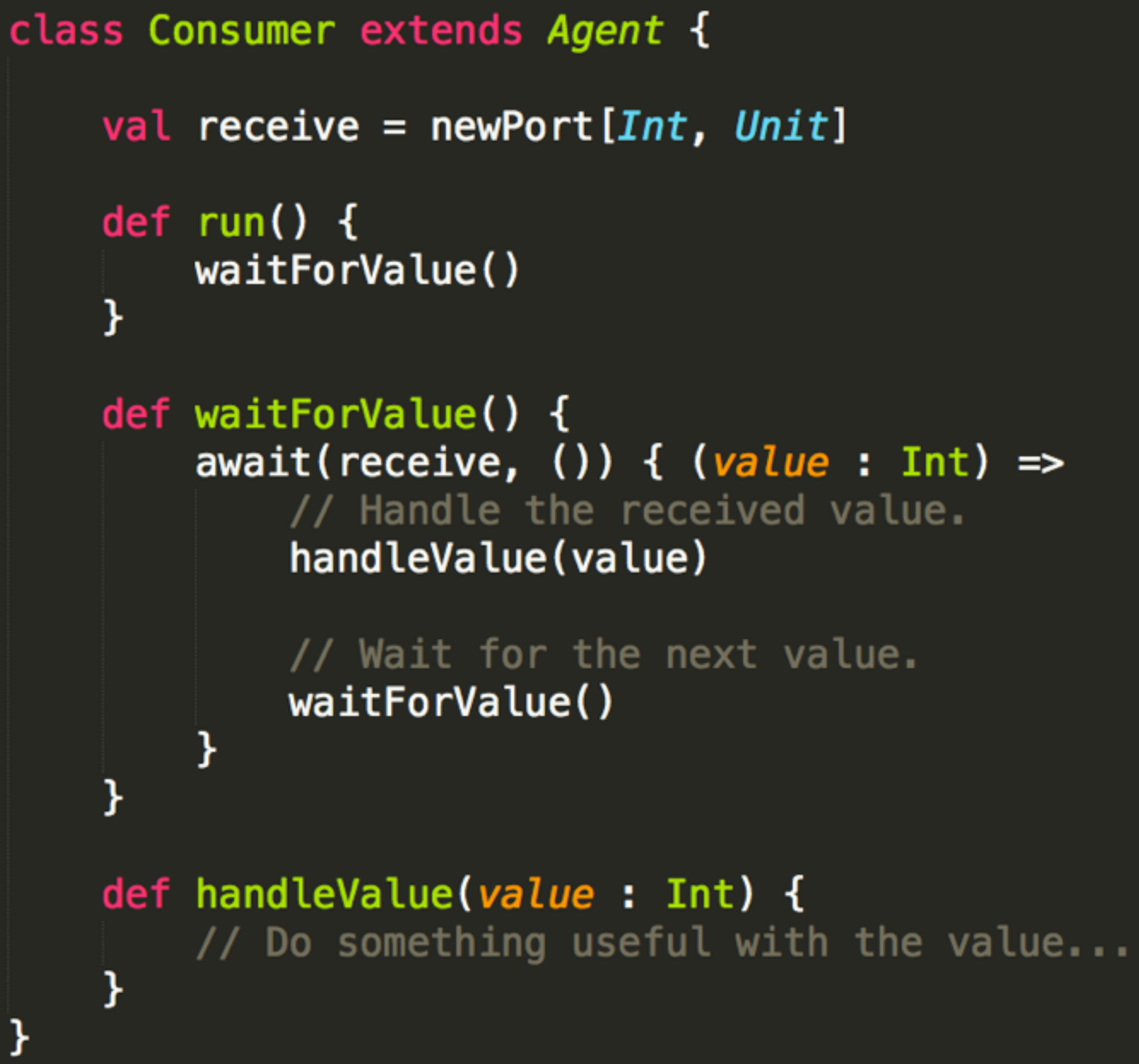
slide courtesy of
Romain Edelmann



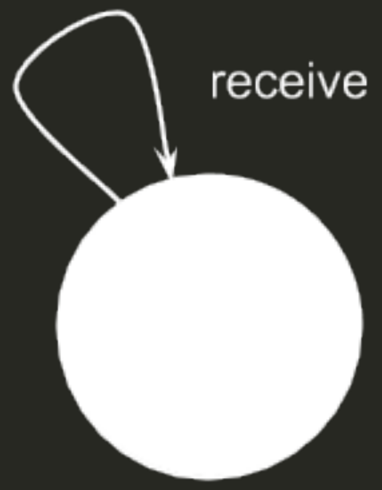
```scala
class Producer extends Agent {

    val send = newPort[Any, Int]

    def run() {
        produceValue()
    }

    def produceValue() {
        val value = 42 // Producing an interesting value here.

        await(send, value) { (_ : Any) =>
            produceValue()
        }
    }

}
```



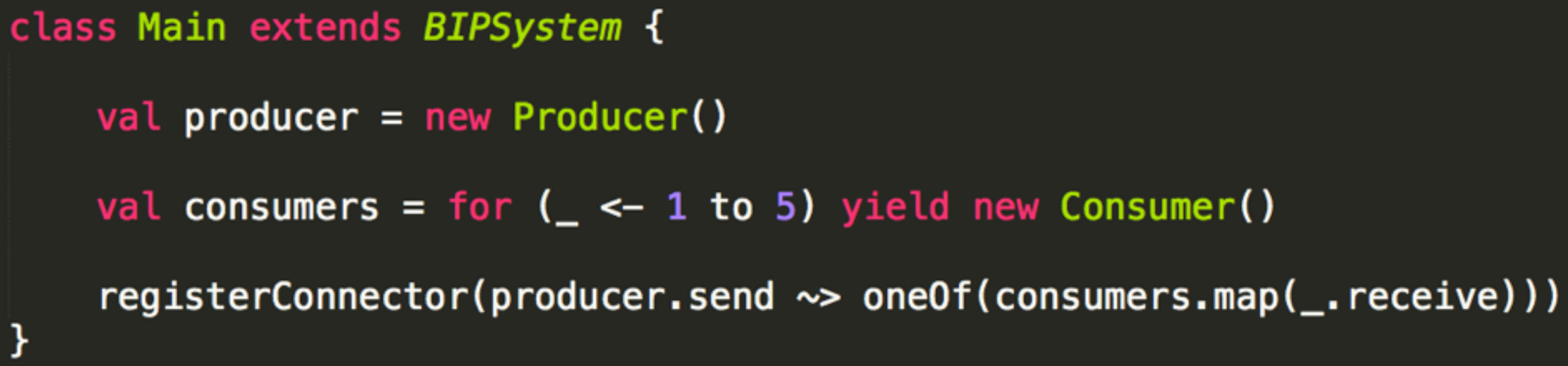
slide courtesy of
Romain Edelmann



```
class Consumer extends Agent {

    val receive = newPort[Int, Unit]

    def run() {
        waitForValue()
    }

    def waitForValue() {
        await(receive, ()) { (value : Int) =>
            // Handle the received value.
            handleValue(value)

            // Wait for the next value.
            waitForValue()
        }
    }

    def handleValue(value : Int) {
        // Do something useful with the value...
    }
}
```



slide courtesy of
Romain Edelmann

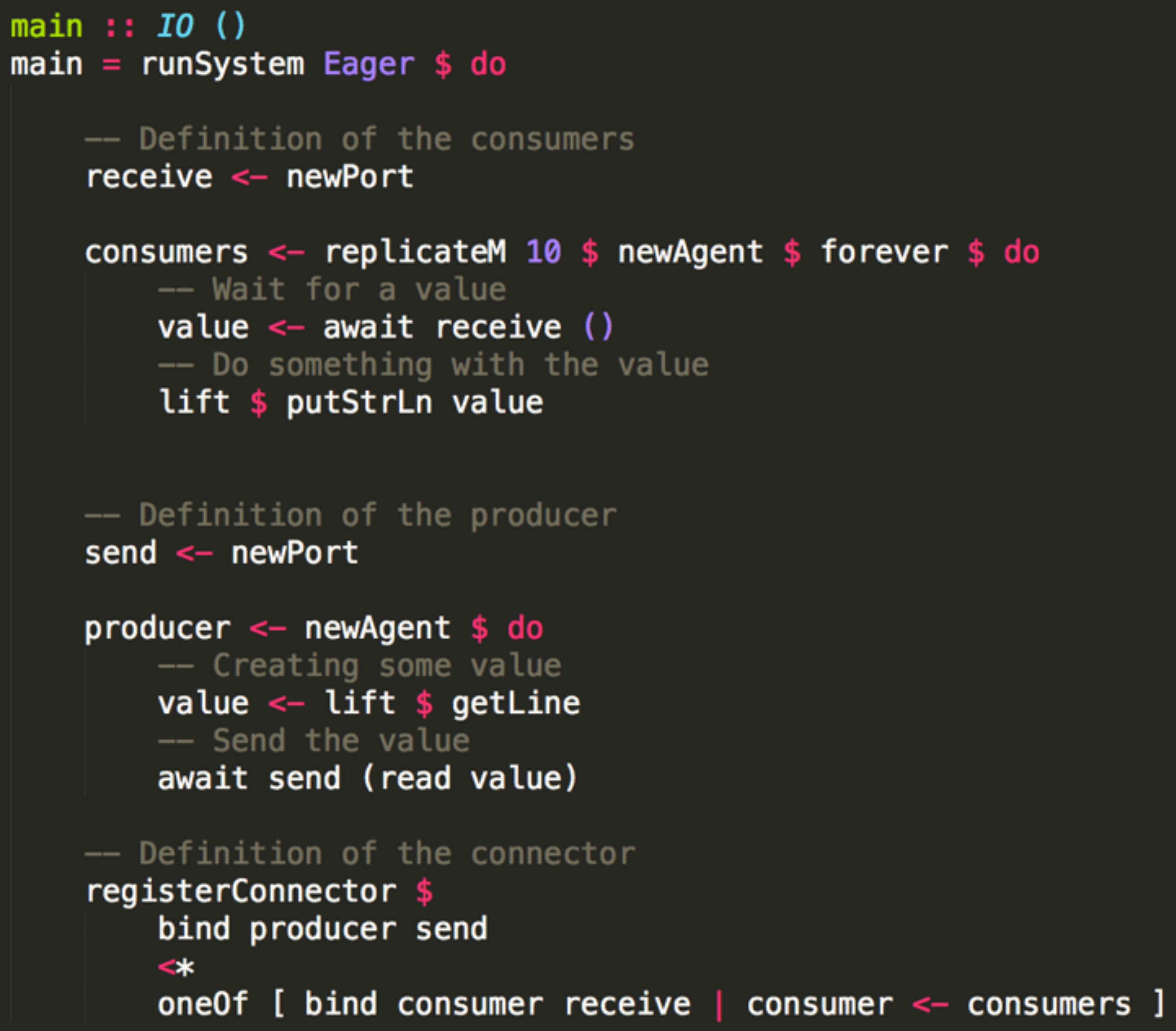


```
class Main extends BIPSystem {

    val producer = new Producer()

    val consumers = for (_ <- 1 to 5) yield new Consumer()

    registerConnector(producer.send ~> oneOf(consumers.map(_.receive)))
}
```

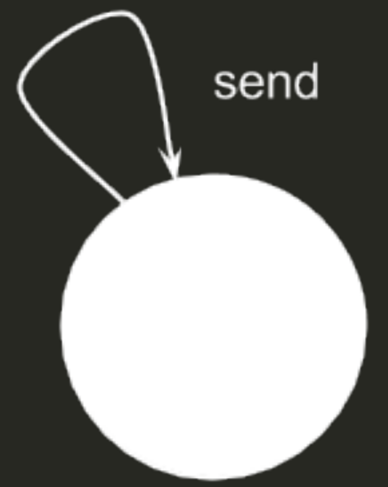
slide courtesy of
Romain Edelmann

Example in Haskell

slide courtesy of
Romain Edelmann



```haskell
main :: IO ()
main = runSystem Eager $ do

    -- Definition of the consumers
    receive <- newPort

    consumers <- replicateM 10 $ newAgent $ forever $ do
        -- Wait for a value
        value <- await receive ()
        -- Do something with the value
        lift $ putStrLn value


    -- Definition of the producer
    send <- newPort

    producer <- newAgent $ do
        -- Creating some value
        value <- lift $ getLine
        -- Send the value
        await send (read value)

    -- Definition of the connector
    registerConnector $
        bind producer send
        <*
        oneOf [ bind consumer receive | consumer <- consumers ]
```



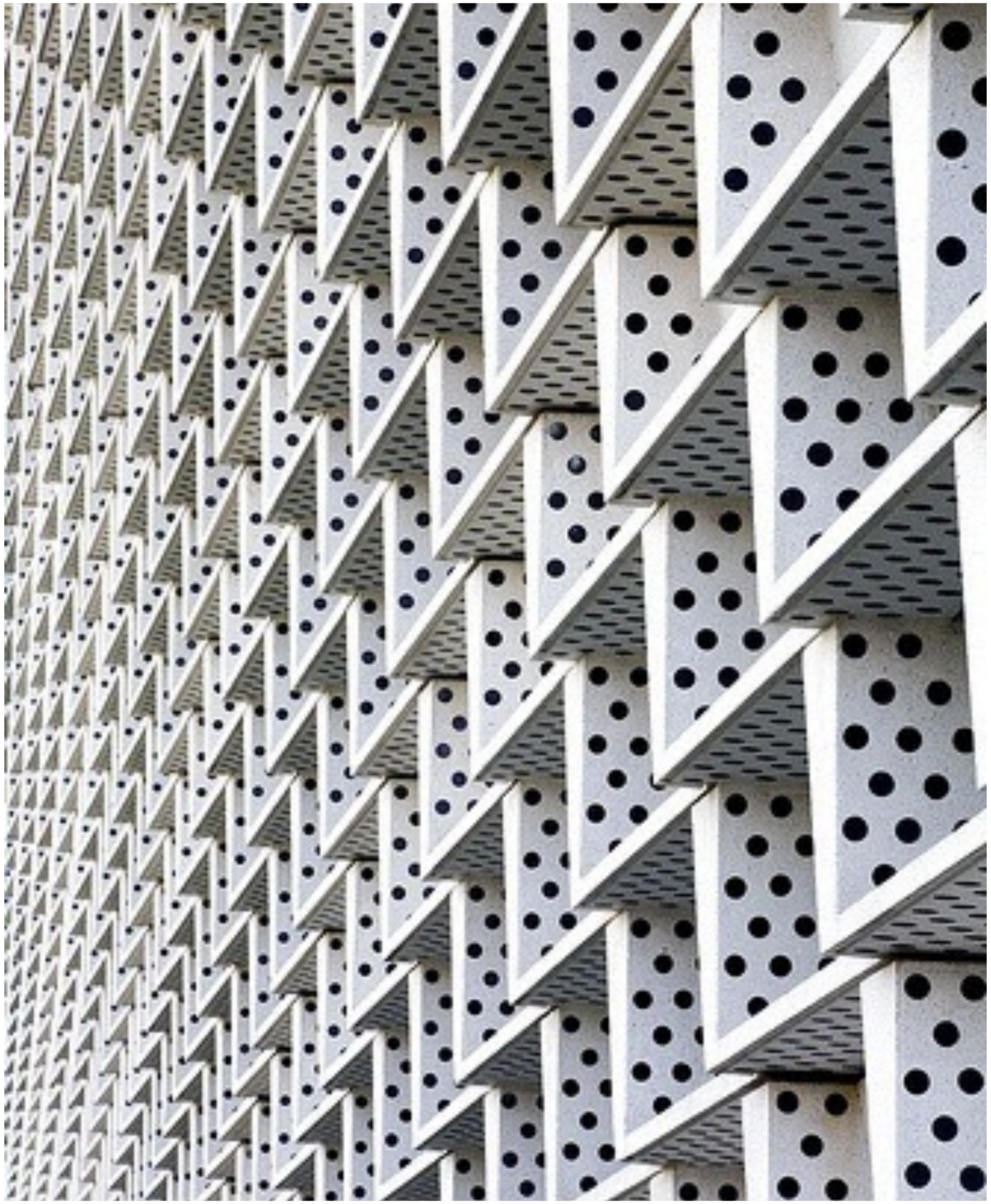
slide courtesy of
Romain Edelmann

The theory of architectures

One of the current research directions

Reusable design patterns

- Systems are not built from scratch
- Maximal re-use of building blocks
(off-the-shelf components)
- Maximal re-use of solutions (libraries, design patterns, etc.)
- Express coordination constraints in declarative manner

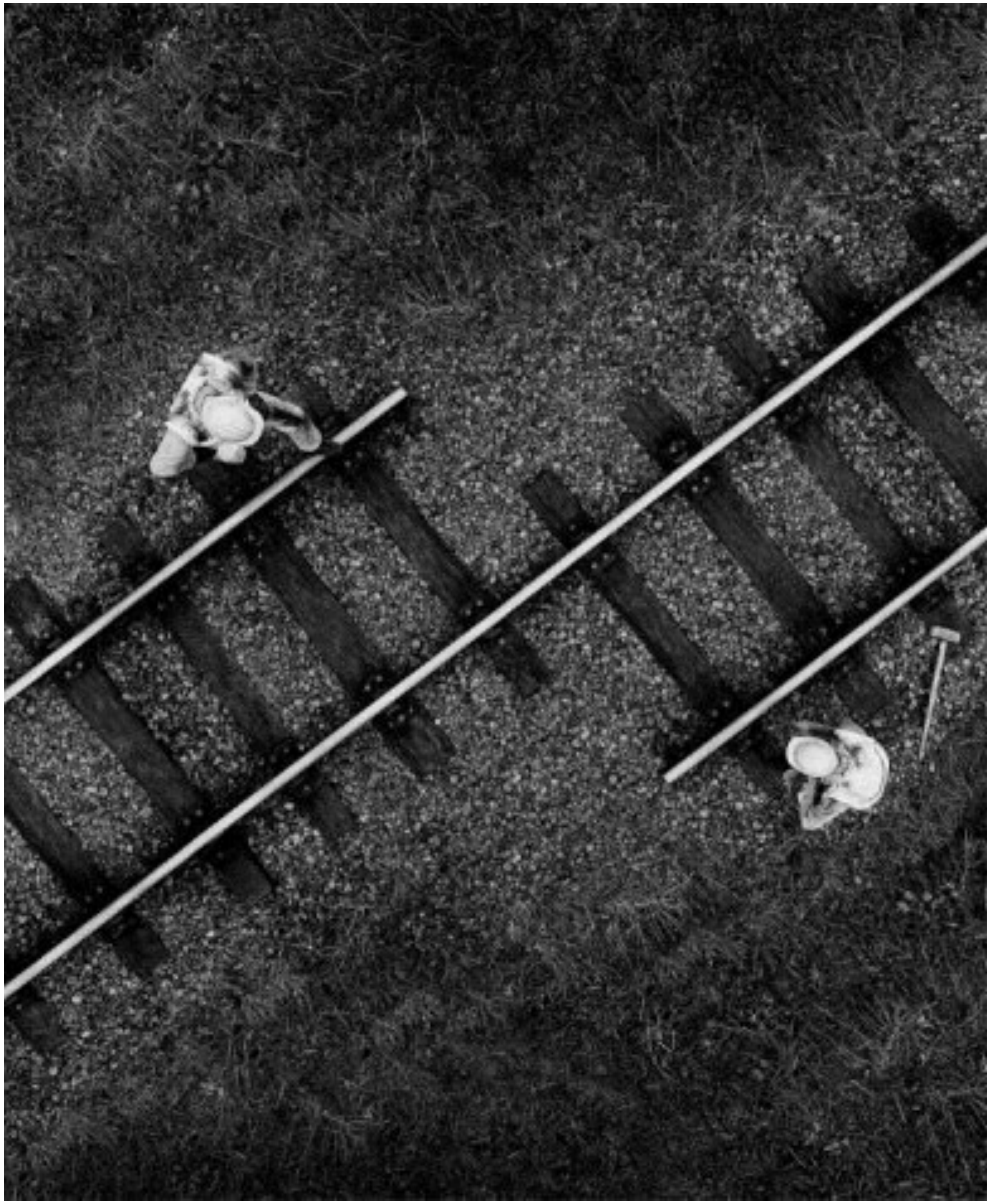


Applications

- Concurrency:
  (a)synchronous, time-triggered, token-ring,
  mutual exclusion
- Interface adaptation:
  communication protocols, data access control
- Robustness:
  fault detection & recovery, resource management
- etc.

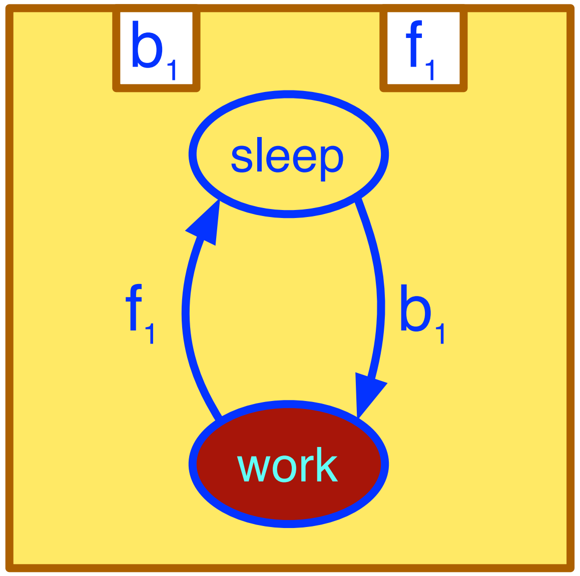
Theory of architectures

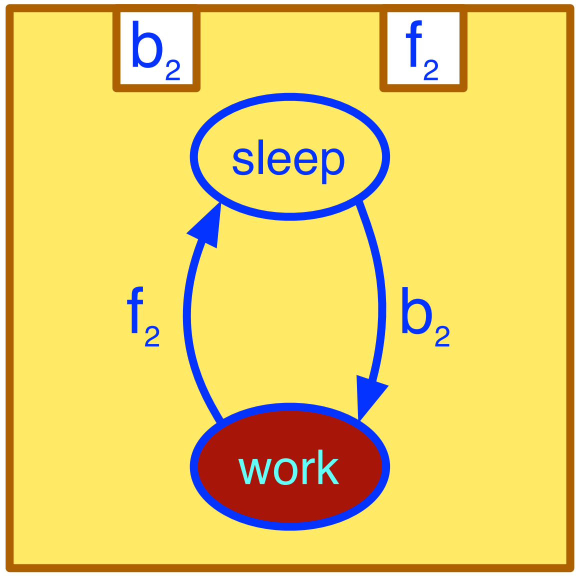
- How to model?
- How to specify?
- How to combine?
- Are properties preserved?



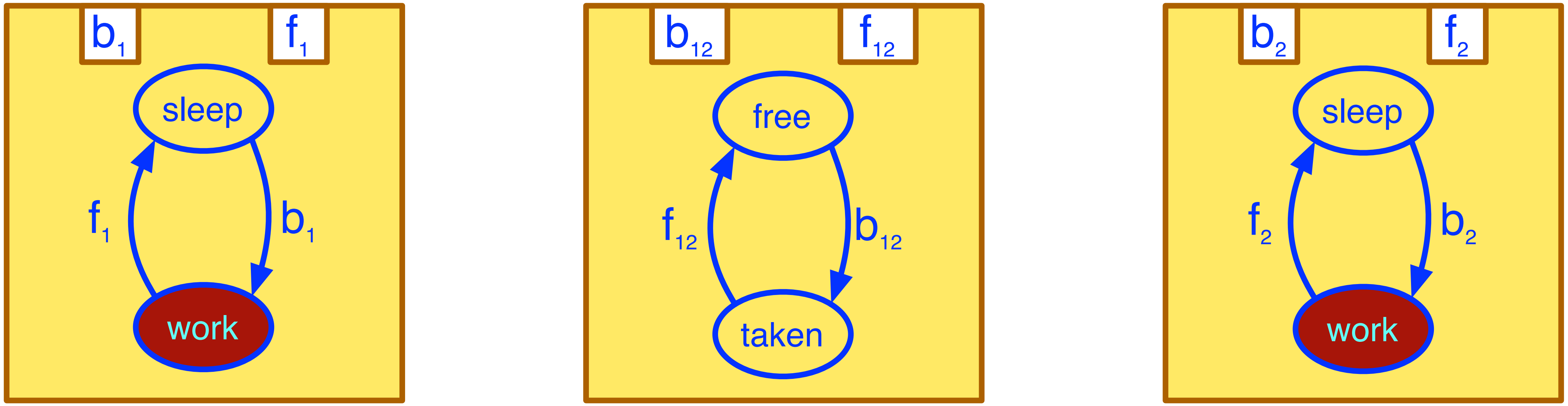
Example in BIP

Example in BIP

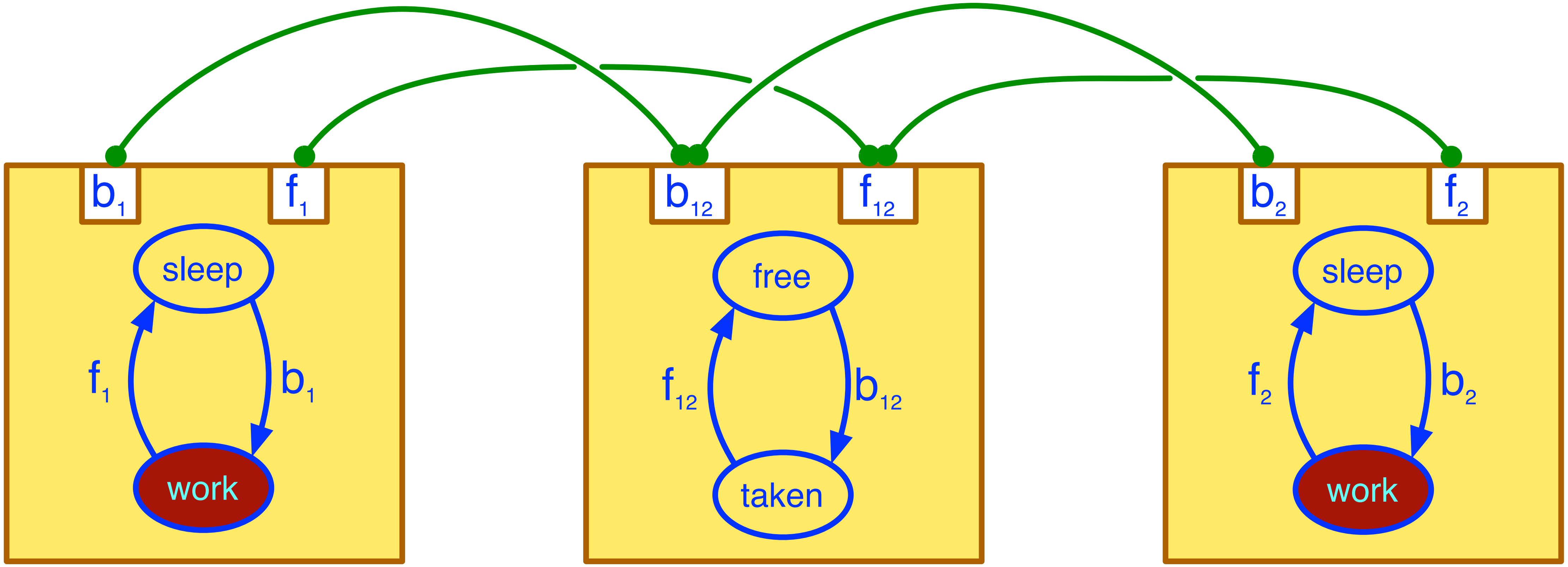




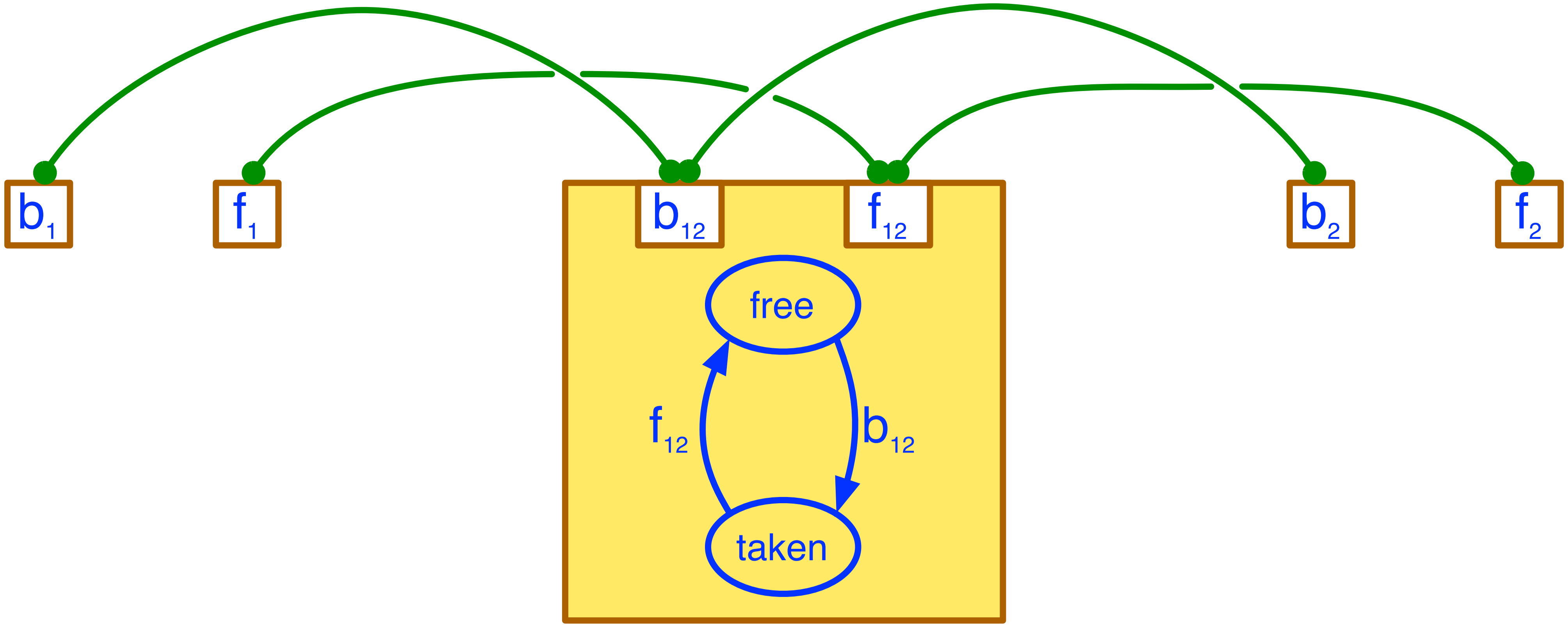
Example in BIP



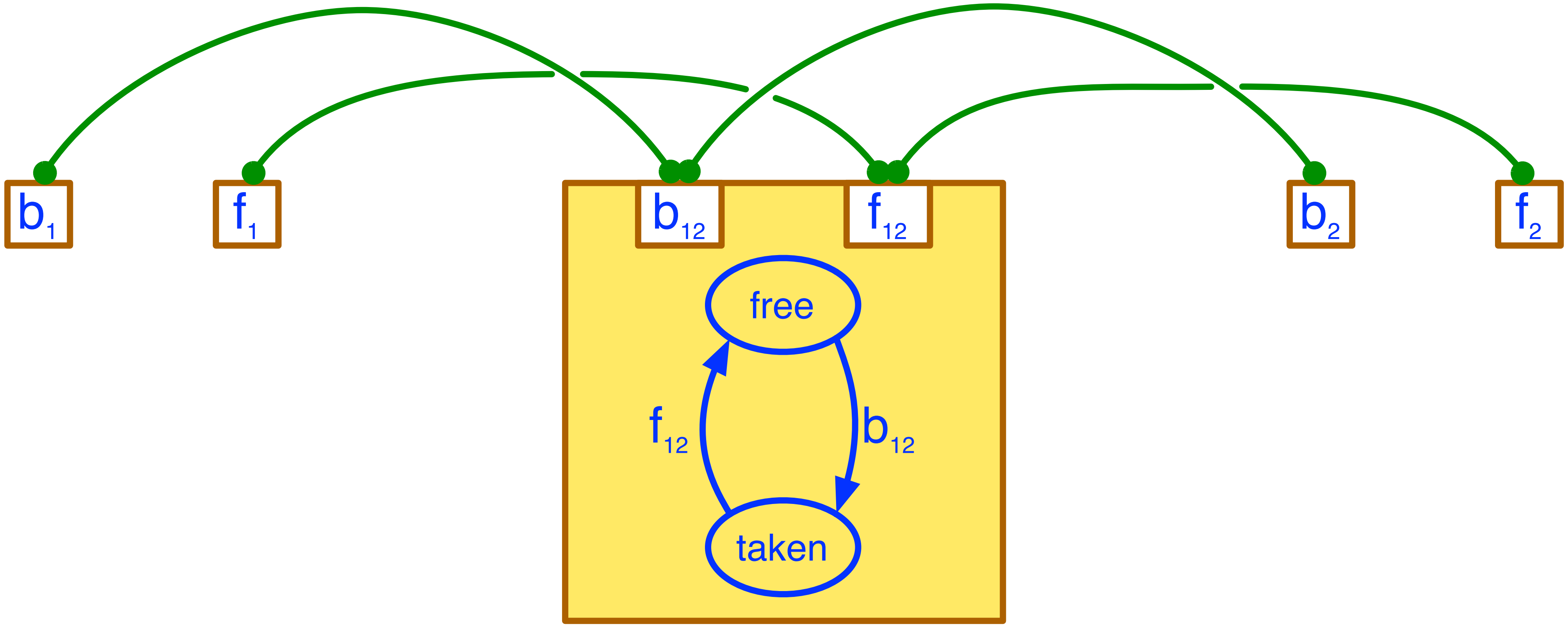
Example in BIP



Example in BIP



Example in BIP



$$\gamma_{12} = \{\emptyset,\ b_1 b_{12},\ b_2 b_{12},\ f_1 f_{12},\ f_2 f_{12}\}$$

Enforcing properties

- Consider behaviour $B = (Q, q^0, P, \to)$
  - A property: $\Phi \subseteq Q$ initial: $q^0 \in \Phi$
  - An invariant: $\forall q \in \Phi,\ \forall a \in 2^P,\ (q \xrightarrow{a} q' \Rightarrow q' \in \Phi)$

- An architecture A imposes a property Φ on $\mathcal{B}$ if Φ is an initial invariant of the projection of the reachable behaviour of $A(\mathcal{B})$ onto $\mathcal{B}$

$$A(\mathcal{B}) \models \Phi$$

Main result

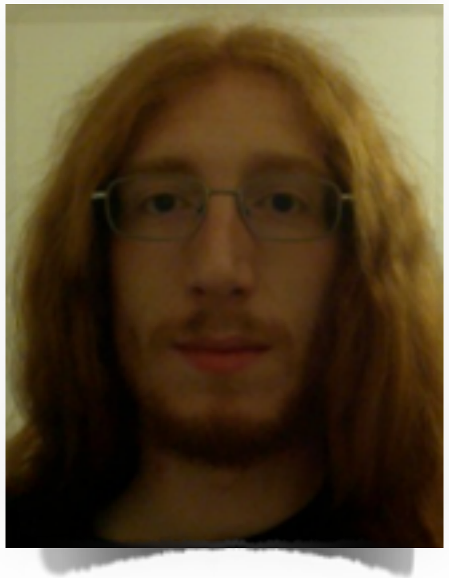
- Safety

$$\left.\begin{array}{l} A_1(\mathcal{B}) \models \Phi_1 \\ A_2(\mathcal{B}) \models \Phi_2 \end{array}\right\} \quad \Longrightarrow \quad \big(A_1 \oplus A_2\big)(\mathcal{B}) \models \Phi_1 \cap \Phi_2$$

- Also an efficient testing methodology for liveness
- Will be presented at SEFM'14 in Grenoble

Summary

- Rigorous design workflow
    - Validate first, then generate the code
    - A sequence of semantics-preserving transformations
- BIP language: provide higher-level abstraction for coordination of **concurrent** components
    - We used the general language and the basic Engine
- BIP framework (at different stages of maturity)
    - Several other language flavours
    - Several engine implementations
    - Analysis & verification tools



Verimag

…and many others.

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE