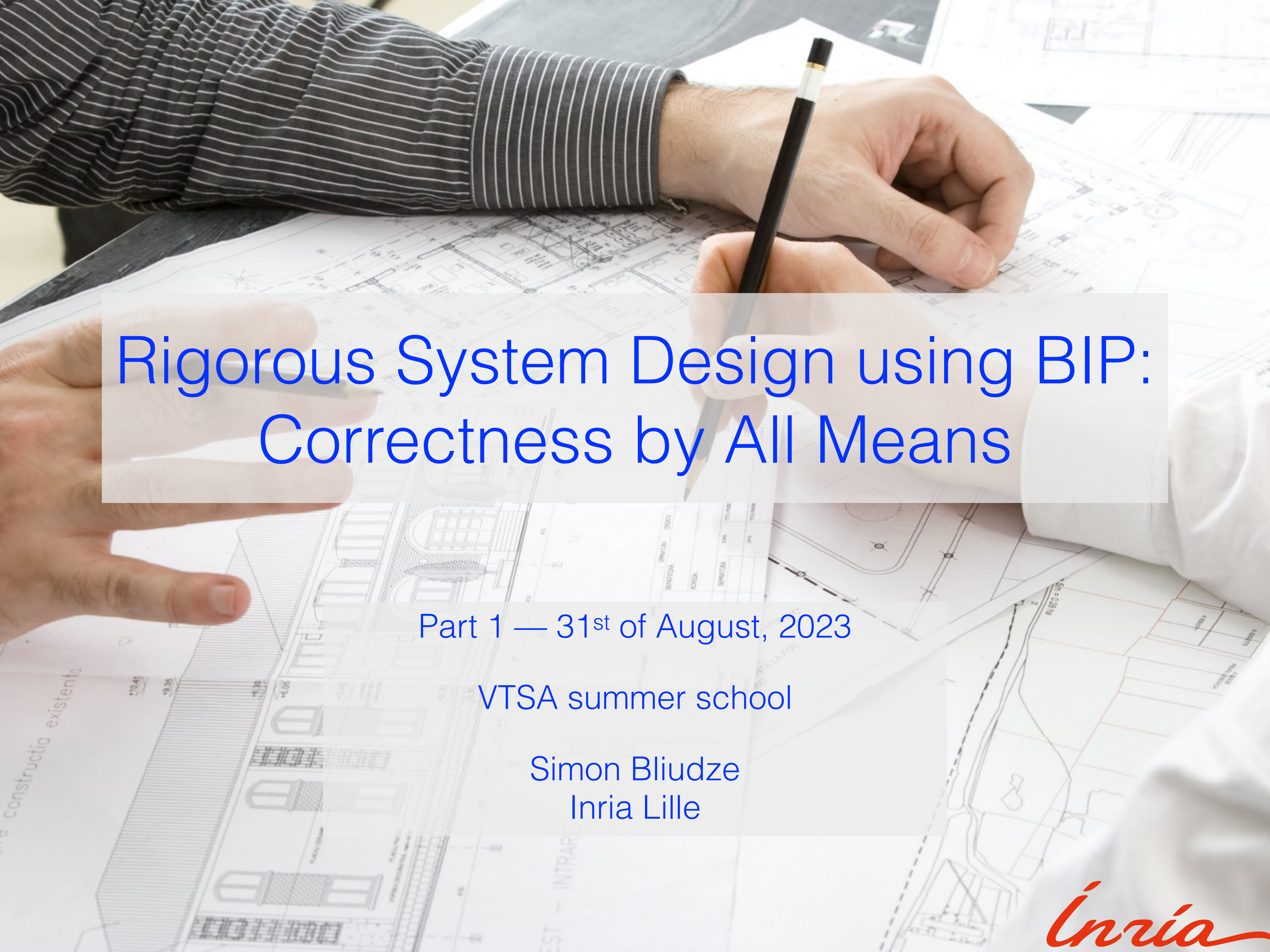


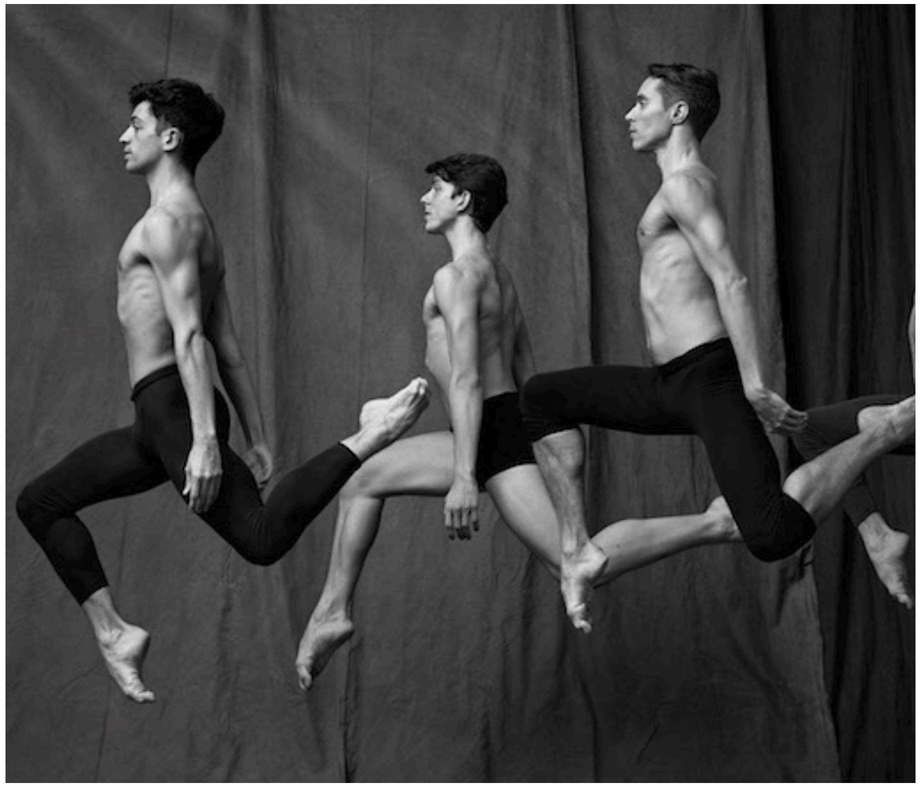
Rigorous System Design using BIP: Correctness by All Means

Part 1 — 31st of August, 2023

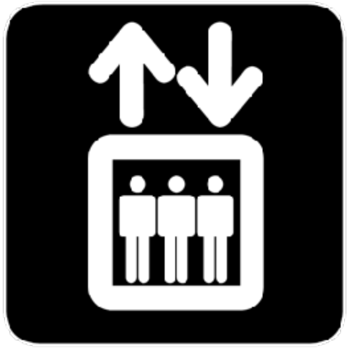
VTSA summer school

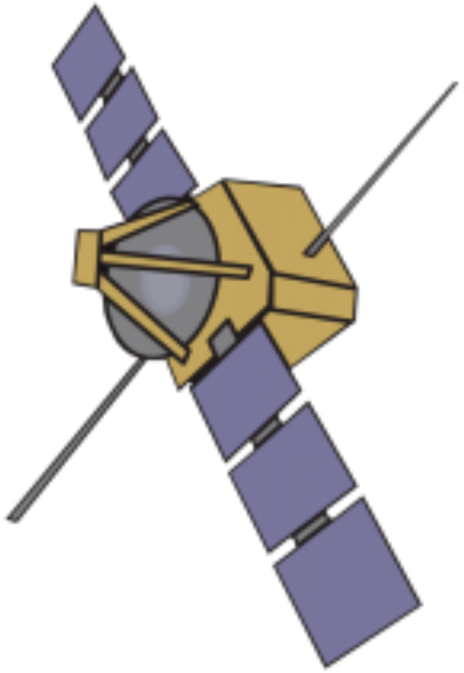
Simon Bliudze
Inria Lille

Concurrency...



...is everywhere!





Embedded

Infrastructure

Platform

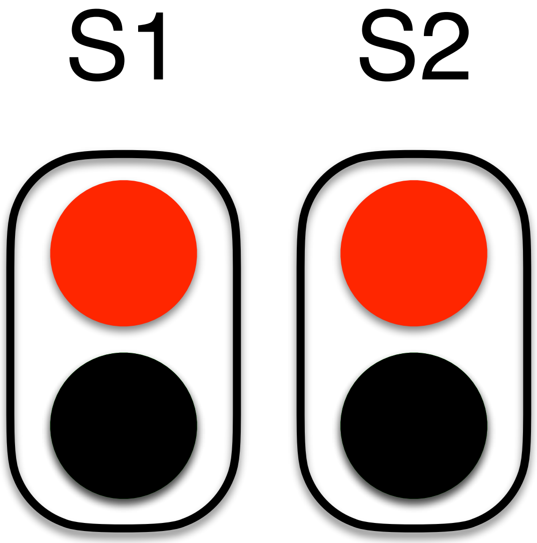
Services

...you name it!

Semaphores, locks, monitors, etc.

Coordination based on low-level primitives rapidly becomes unpractical.

Synchronisation



Process 1:

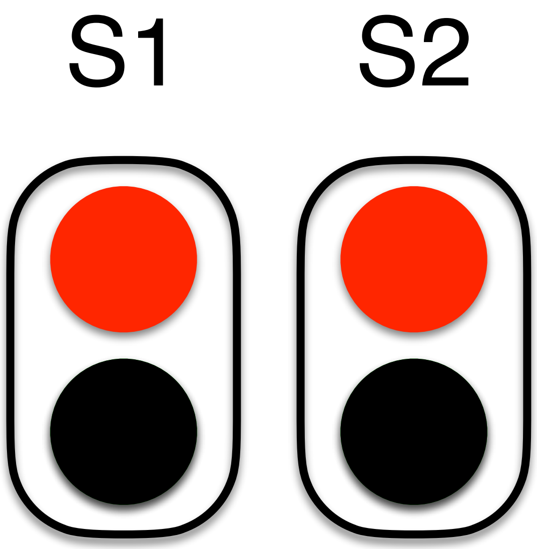
```
...
free(S1);
take(S2);
...
```

Process 2:

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

A simple synchronisation barrier

Synchronisation



Process 1:

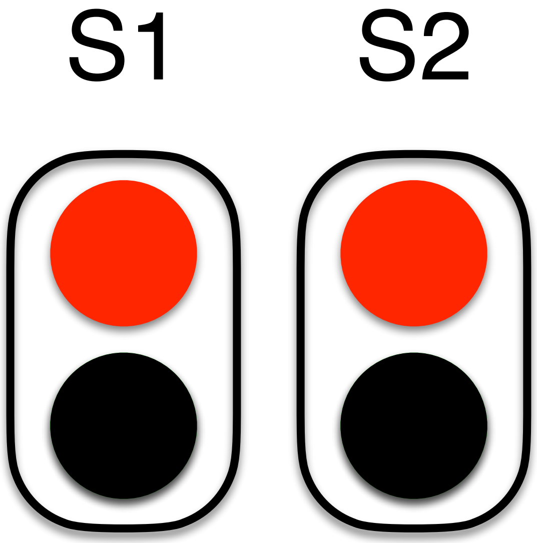
```
...
free(S1);
take(S2);
...
```

Process 2:

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

A simple synchronisation barrier

Synchronisation



Process 1:

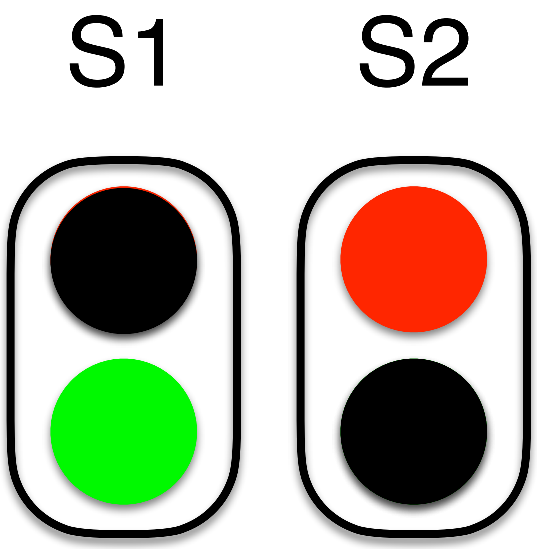
```
...
free(S1);
take(S2);
...
```

Process 2:

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

A simple synchronisation barrier

Synchronisation



Process 1:

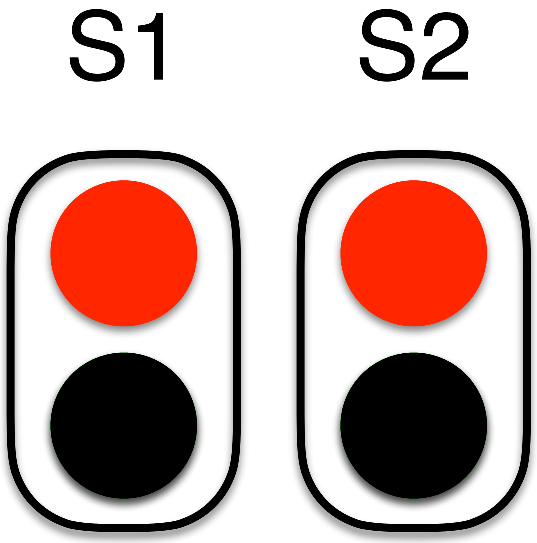
```
...
free(S1);
take(S2);
...
```

Process 2:

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

A simple synchronisation barrier

Synchronisation



Process 1:

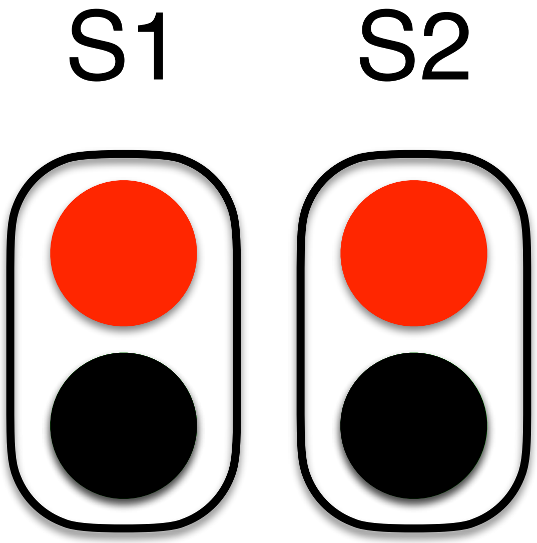
```
...
free(S1);
take(S2);
...
```

Process 2:

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

A simple synchronisation barrier

Synchronisation



Process 1:

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

Process 2:

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

A simple synchronisation barrier

Synchronisation

Process 1:

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

Process 2:

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

Process 3:

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

Three-way synchronisation barrier

Synchronisation with data transfer

Process 1:

```
x = f1(sh1,sh2);
free(S1);
take(S2);
sh1 = f2(sh1,x);
free(S1);
take(S2);
x = f3(sh1,sh2);
```

Process 2:

```
y = g1(sh1,sh2);
take(S1);
free(S2);
sh2 = g2(y,sh2);
take(S1);
free(S2);
y = g3(sh1,sh2);
```

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

Synchronisation with data transfer

Process 1:

```
x = f1(sh1,sh2);
free(S1);
take(S2);
sh1 = f2(sh1,x);
free(S1);
take(S2);
x = f3(sh1,sh2);
```

Process 2:

```
y = g1(sh1,sh2);
take(S1);
free(S2);
sh2 = g2(y,sh2);
take(S1);
free(S2);
y = g3(sh1,sh2);
```

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

Objectives

Correct-by-construction concurrent systems

Separation of computation from coordination

Outline

Practical aspects

Overview of the RSD approach

CubETH case study

Operational semantics

BIP language introduction

Theoretical aspects

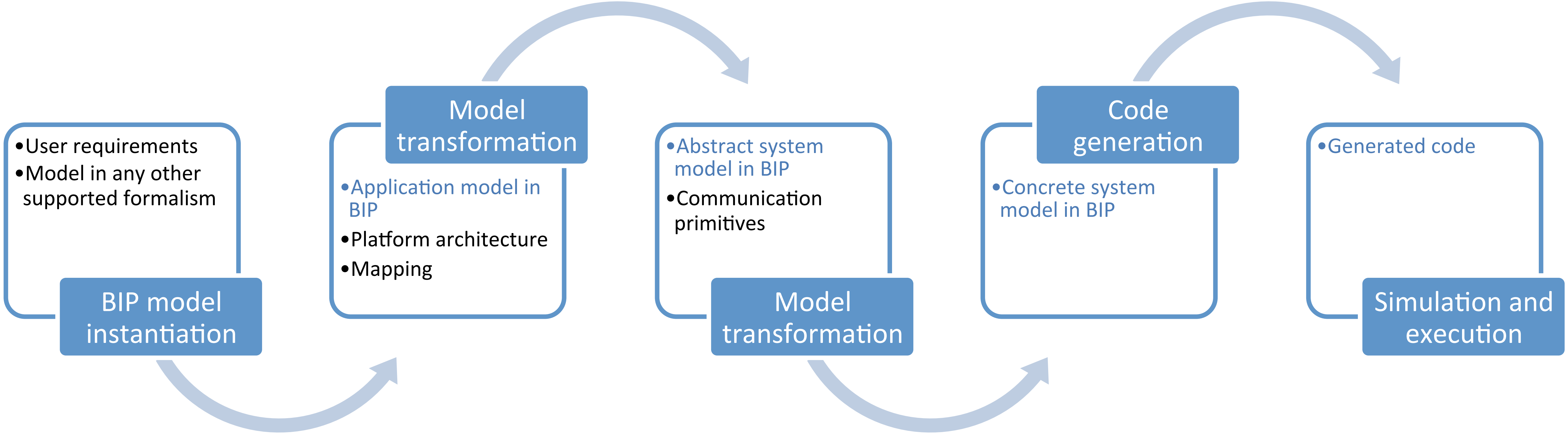
Connector modelling

Architectures: design patterns for BIP

Connector synthesis

Expressiveness study

Rigorous System Design flow



- User requirements
- Model in any other supported formalism

BIP model instantiation

- Application model in BIP
- Platform architecture
- Mapping

Model transformation

- Abstract system model in BIP
- Communication primitives

Model transformation

- Concrete system model in BIP

Code generation

- Generated code

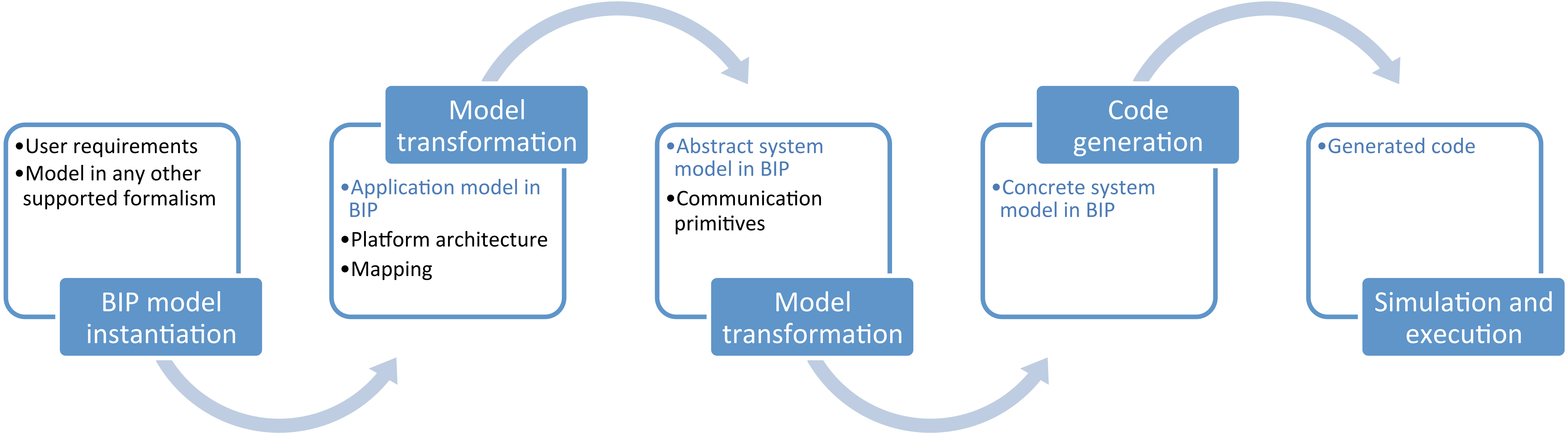
Simulation and execution

Models progressively refined with new information

In **black** — provided by the designer

In blue — generated by automatic transformation tools

Application model



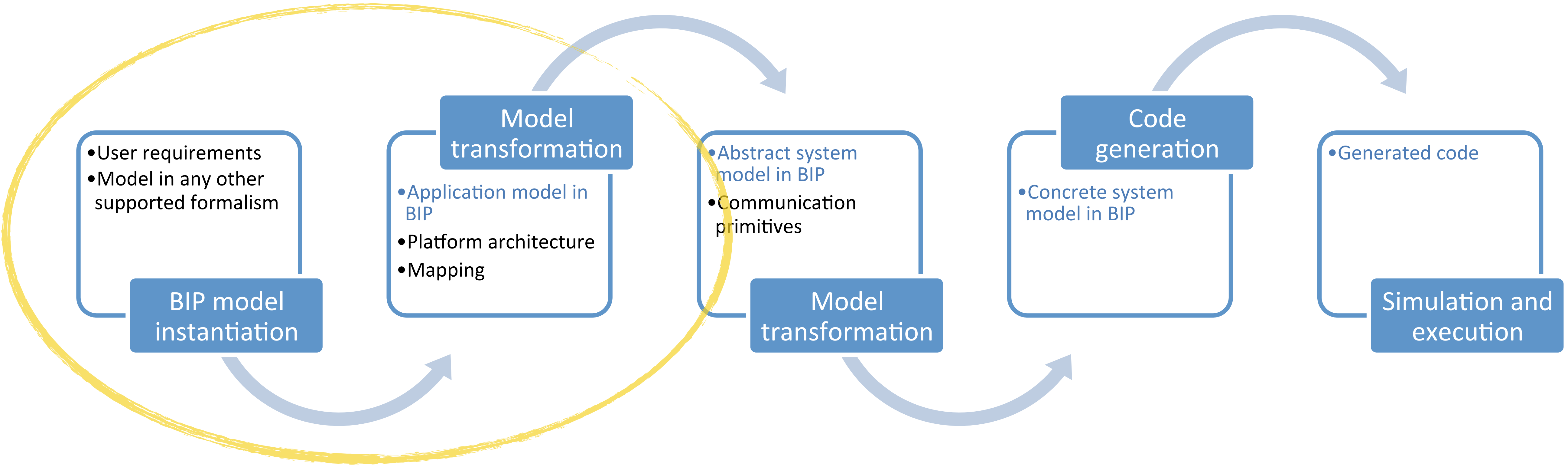
Application model is designed directly in BIP

- 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 (DFinder, later IFinder)
- Partial transformation for model-checking with nuXmv

Application model



Application model is designed directly in BIP

or using a language factory transformation from

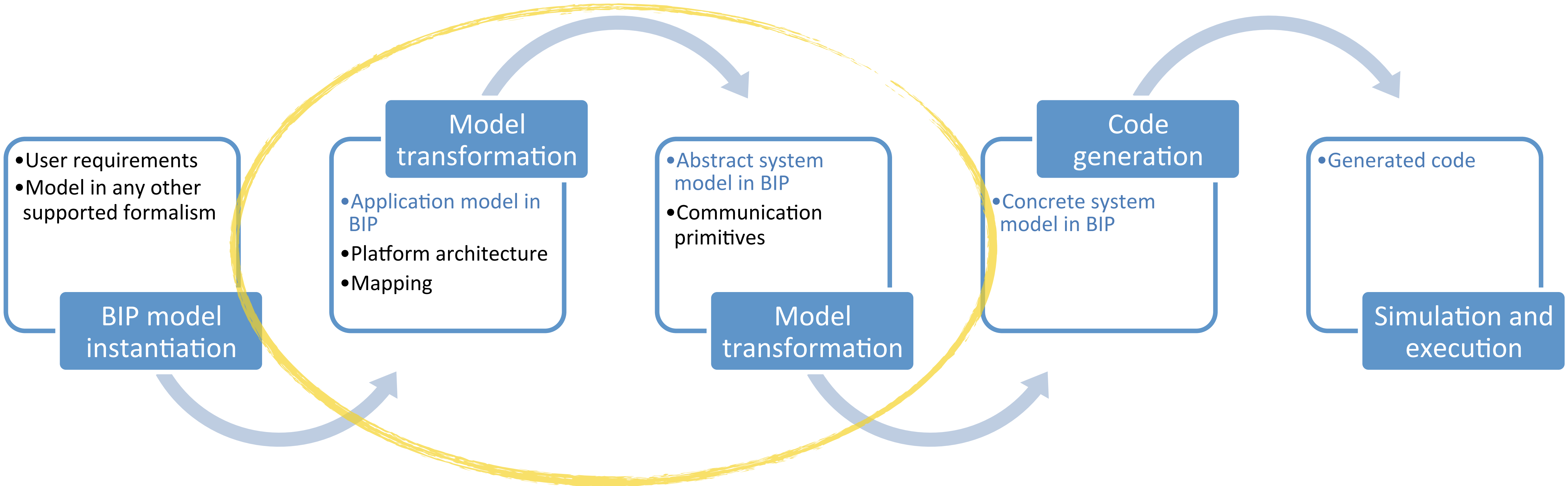
C, AADL, NesC/TinyOS, MathLab/Simulink, Lustre, DOL, GeNoM...

Safety properties are verified on this model

Compositional and incremental deadlock detection (DFinder, later IFinder)

Partial transformation for model-checking with nuXmv

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

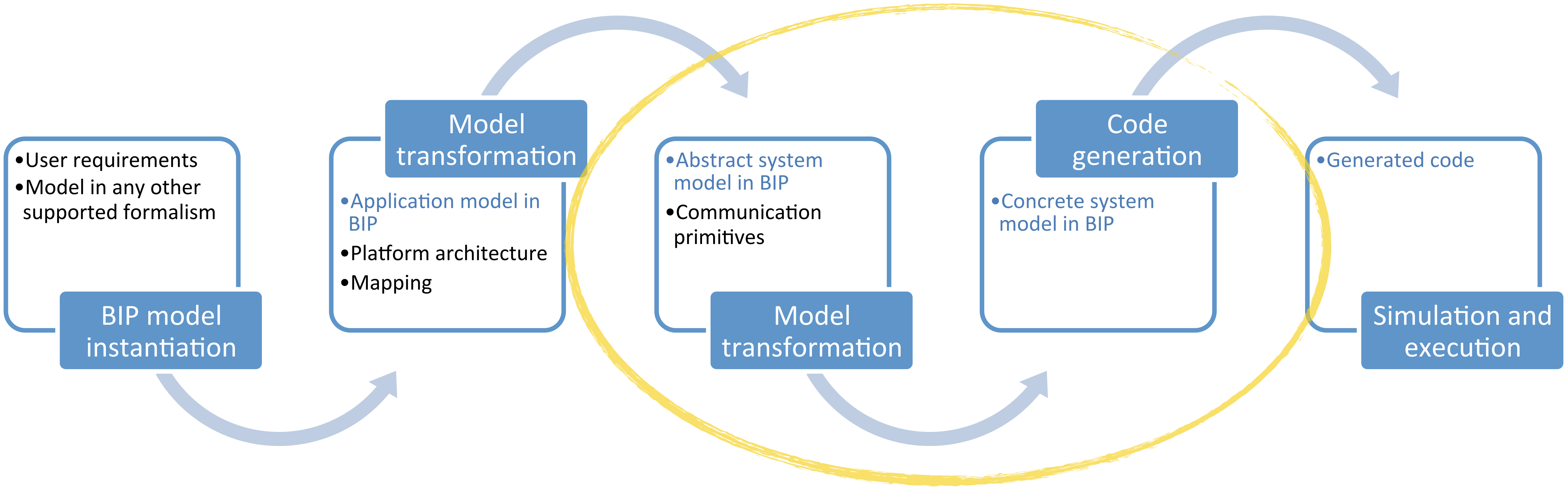
It takes in account

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…

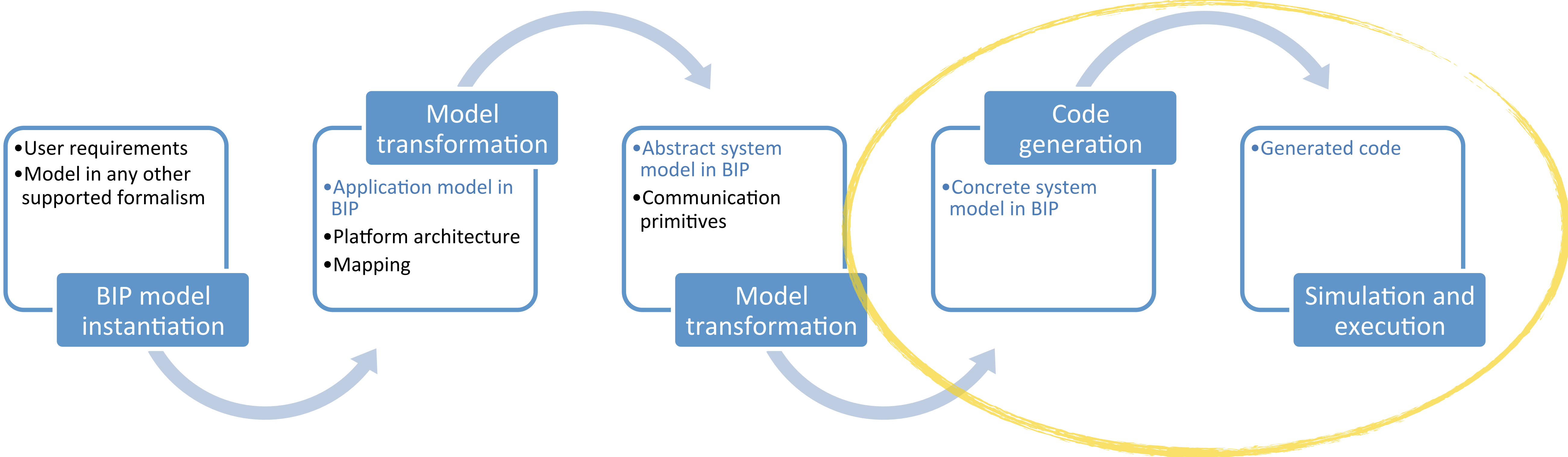
Atomic multiparty interactions

Priorities

…through the primitives of the target execution platform

For example, protocols using asynchronous message passing

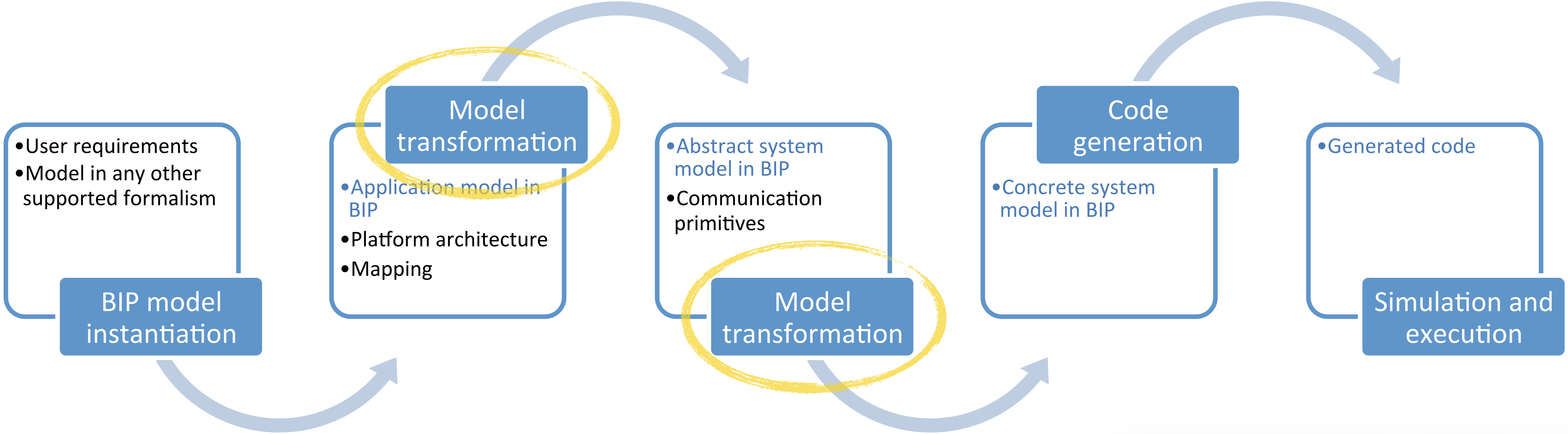
Code generation



C++ code is automatically generated for each processing unit

Generated code is monolithic, minimising the coordination overhead

Rigorous System Design flow



A series of semantics-preserving transformations

Correctness decomposed into

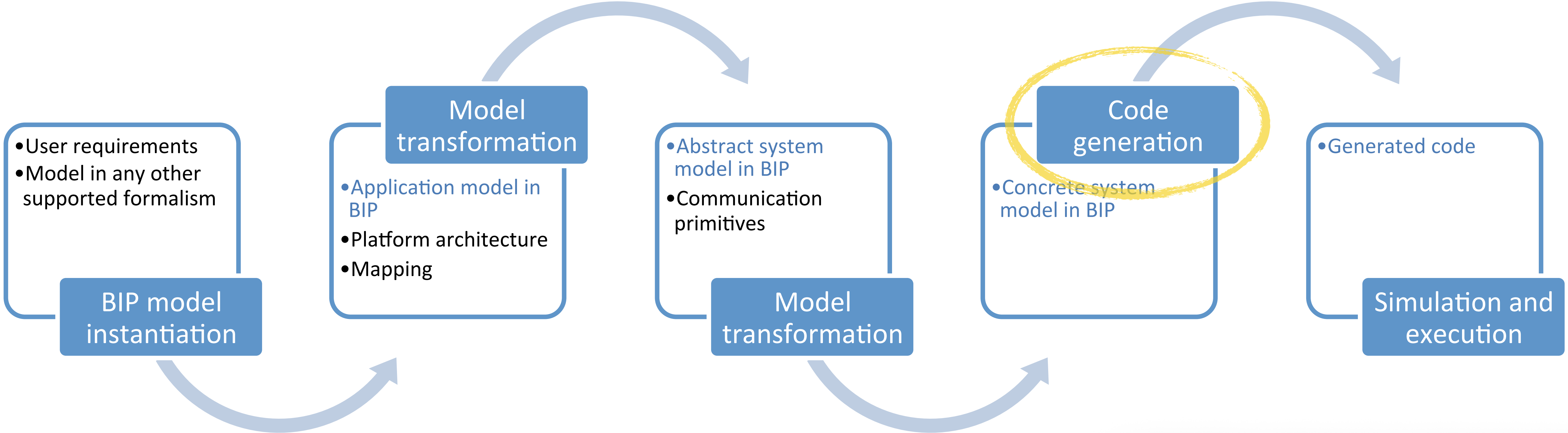
correctness of transformations

correctness of high-level models

Final implementation is **correct by construction**

□ Unifying modelling framework

Rigorous System Design flow


- User requirements
- Model in any other supported formalism

BIP model instantiation

Model transformation

- Application model in BIP
- Platform architecture
- Mapping

- Abstract system model in BIP
- Communication primitives

Model transformation

Code generation

- Concrete system model in BIP

- Generated code

Simulation and execution

□ Unifying modelling framework

□ Operational semantics

A series of semantics-preserving transformations

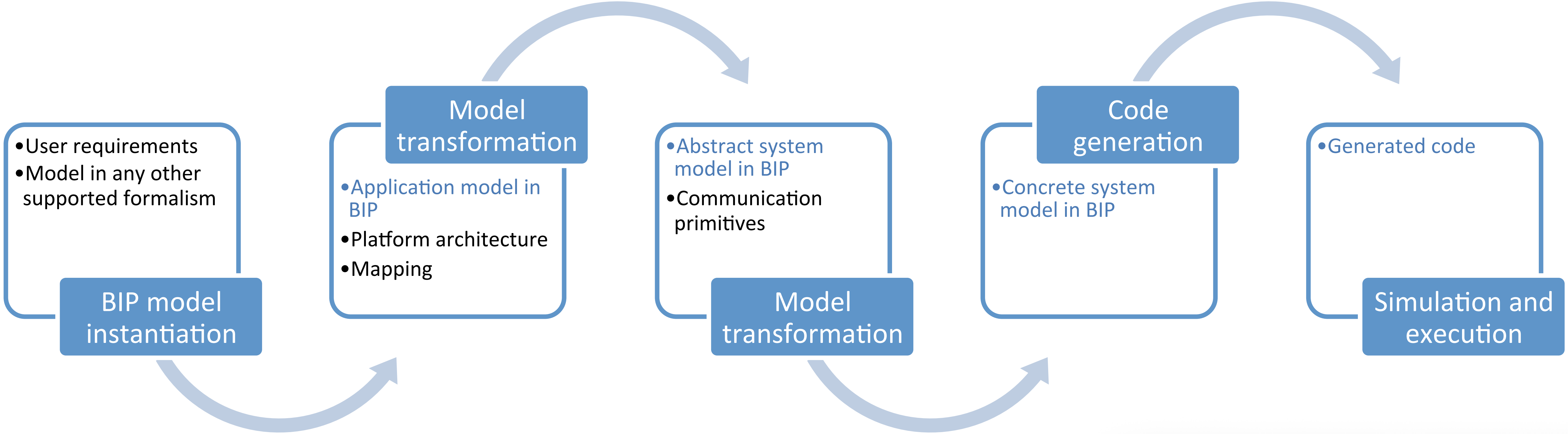
Correctness decomposed into

correctness of transformations

correctness of high-level models

Final implementation is **correct by construction**

Rigorous System Design flow



A series of semantics-preserving transformations

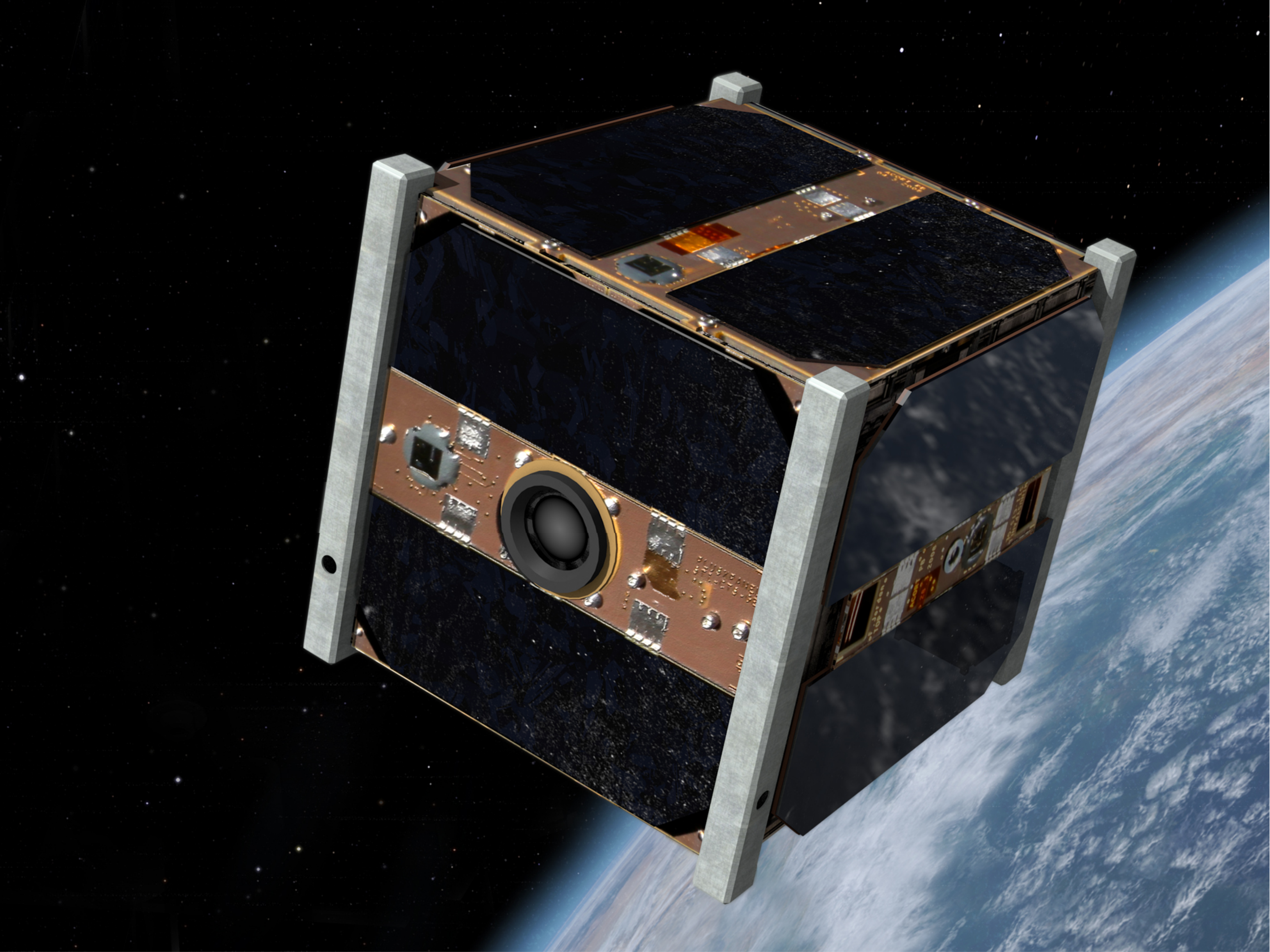
Correctness decomposed into

correctness of transformations

correctness of high-level models

Final implementation is **correct by construction**

- □ Unifying modelling framework
- □ Operational semantics
- □ Method(s) to design correct models



Satellite software design

- A collaboration with the EPFL Space Engineering Center
  - Component-based design in BIP of the control software for a nano-satellite
    - Control and Data Management System (CDMS)
    - Communication with other subsystems through an I²C bus

- A collaboration with ThalesAlenia Space (France) and Aristotle University of Thessaloniki (Greece)
  - "Catalogue of System and Software Properties"
    - Funded by ESA

eSpace
SPACE
ENGINEERING
CENTER

ThalesAlenia Space
A Thales / Finmeccanica Company

ARISTOTLE
UNIVERSITY OF
THESSALONIKI

Satellite software design

- A collaboration with the EPFL Space Engineering Center
  - Component-based design in BIP of the control software for a nano-satellite
    - Control and Data Management System (CDMS)
    - Communication with other subsystems through an I^2C bus

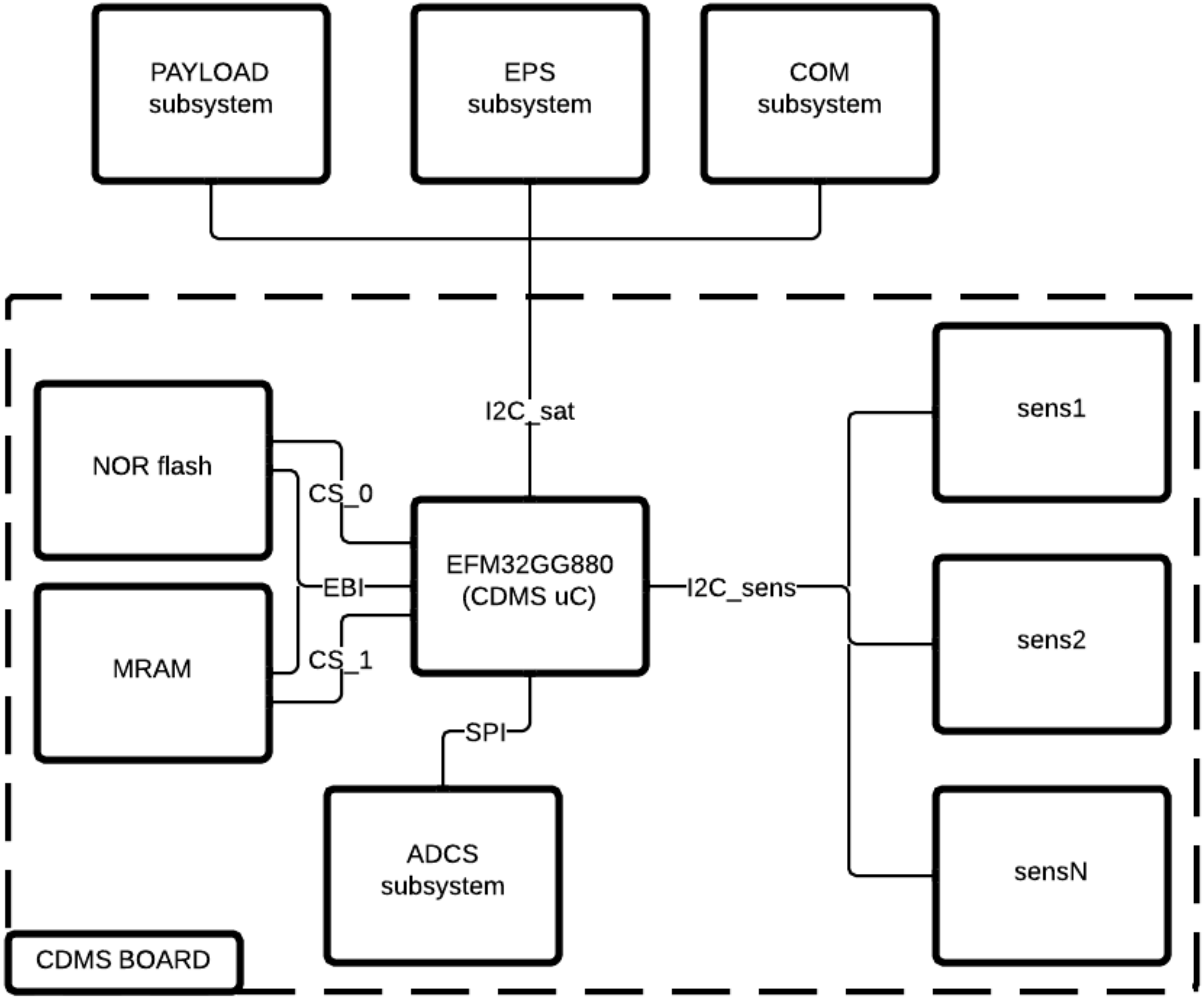
- A collaboration with ThalesAlenia Space (France) and Aristotle University of Thessaloniki (Greece)
  - “Catalogue of System and Software Properties”
    - Funded by ESA

eSpace
SPACE
ENGINEERING
CENTER

ThalesAlenia
A Thales / Finmeccanica Company
Space

ARISTOTLE
UNIVERSITY OF
THESSALONIKI

CubETH: CDMS architecture



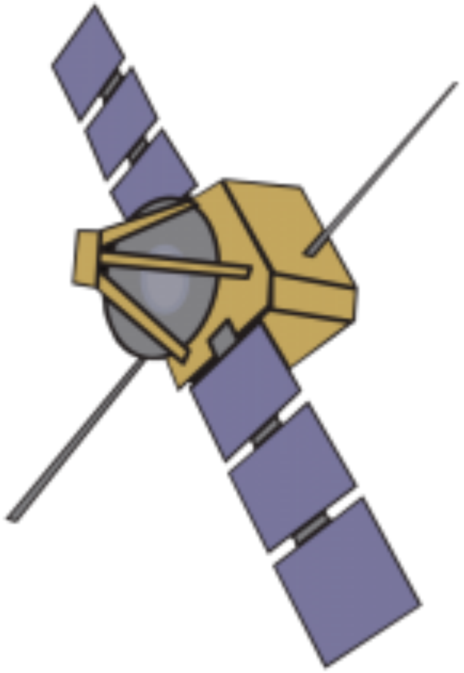


Figure courtesy of Marco Pagnamenta

CubETH: CDMS architecture

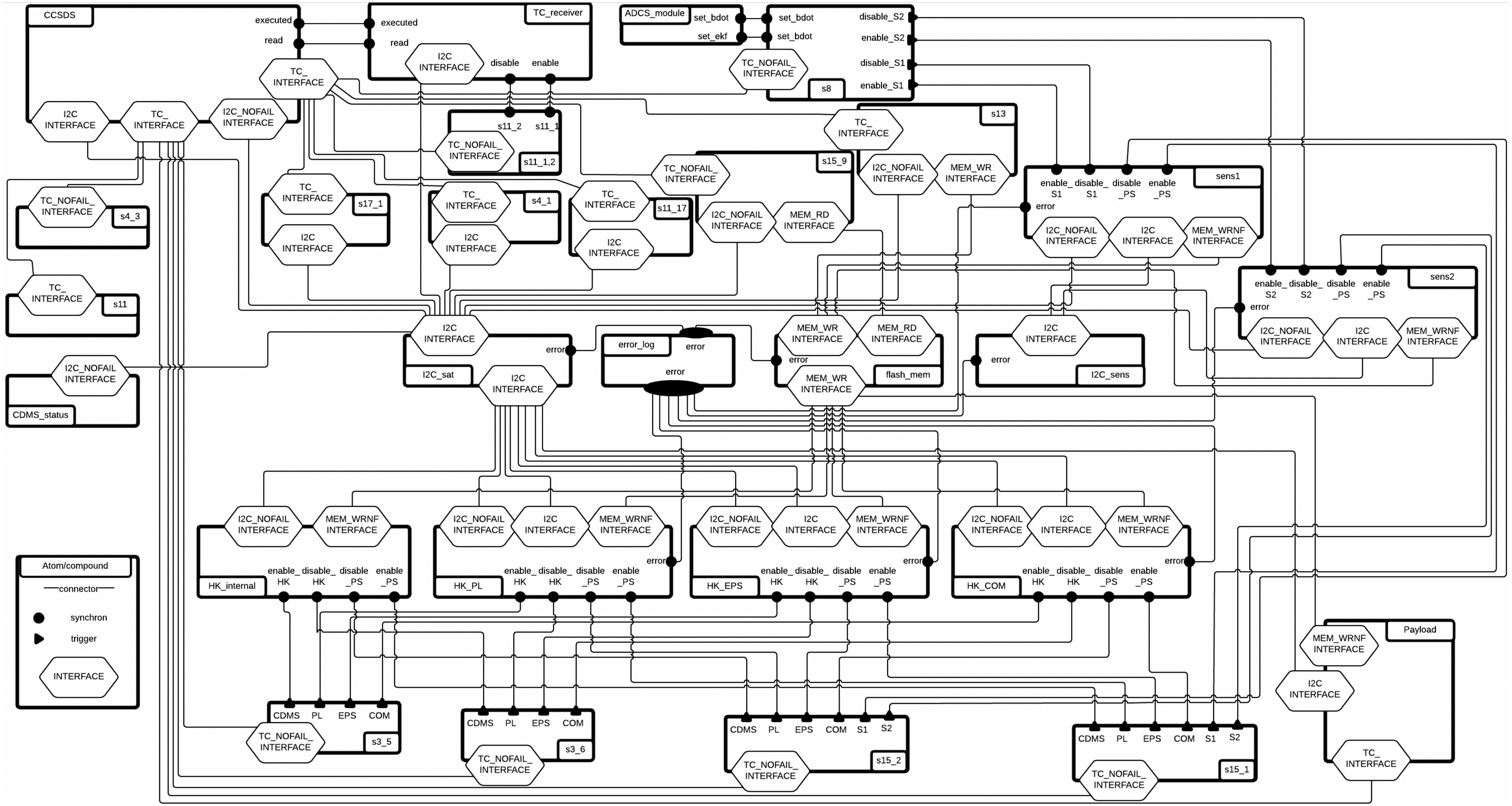
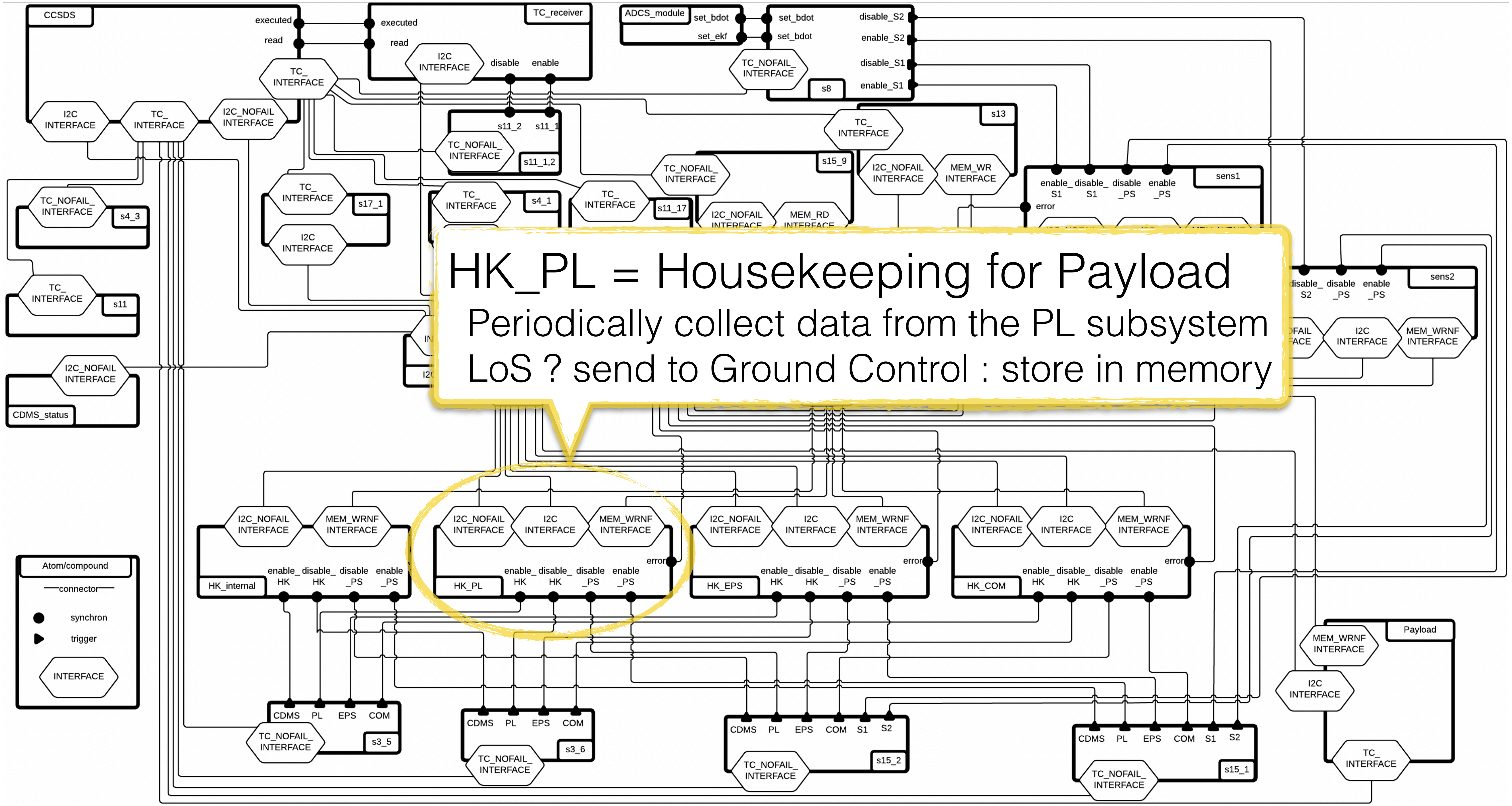


Figure courtesy of Marco Pagnamenta

CubETH: CDMS architecture



HK_PL = Housekeeping for Payload

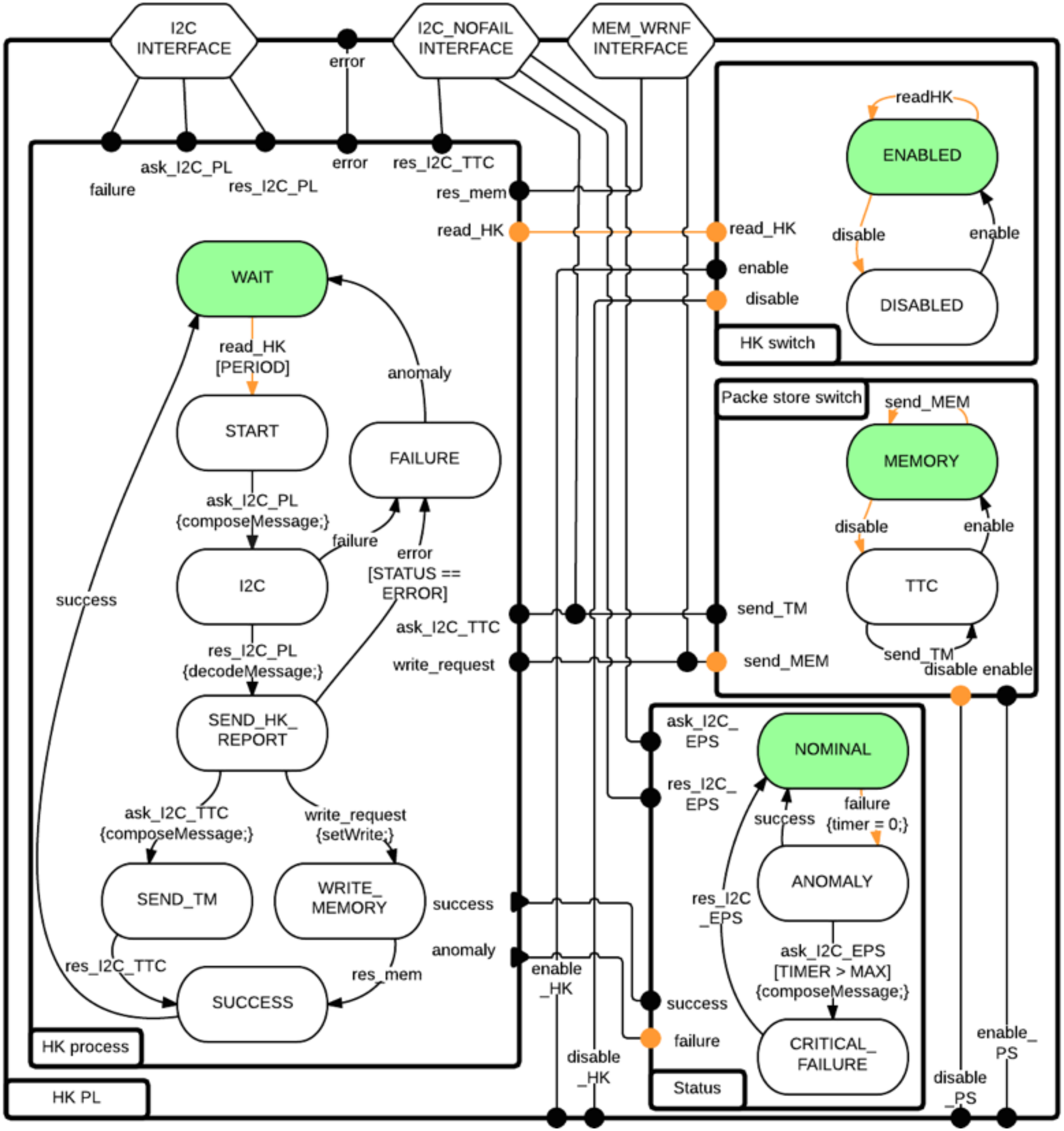
Periodically collect data from the PL subsystem

LoS ? send to Ground Control : store in memory

Figure courtesy of Marco Pagnamenta

Example 1

Nominal housekeeping routine



I2C INTERFACE

I2C_NOFAIL INTERFACE

MEM_WRNF INTERFACE

error

failure

ask_I2C_PL

res_I2C_PL

error

res_I2C_TTC

res_mem

read_HK

WAIT

read_HK [PERIOD]

START

ask_I2C_PL {composeMessage;}

I2C

failure

FAILURE

anomaly

error [STATUS == ERROR]

res_I2C_PL {decodeMessage;}

SEND_HK_ REPORT

ask_I2C_TTC

write_request

success

ask_I2C_TTC {composeMessage;}

write_request {setWrite;}

SEND_TM

WRITE_ MEMORY

res_I2C_TTC

res_mem

SUCCESS

success

anomaly

HK process

HK PL

read_HK

enable

disable

readHK

ENABLED

disable

enable

DISABLED

HK switch

Packe store switch

send_MEM

MEMORY

disable

enable

TTC

send_TM

send_MEM

send_TM

disable enable

ask_I2C_ EPS

res_I2C_ EPS

NOMINAL

success

failure {timer = 0;}

ANOMALY

res_I2C _EPS

ask_I2C_EPS [TIMER > MAX] {composeMessage;}

CRITICAL_ FAILURE

success

failure

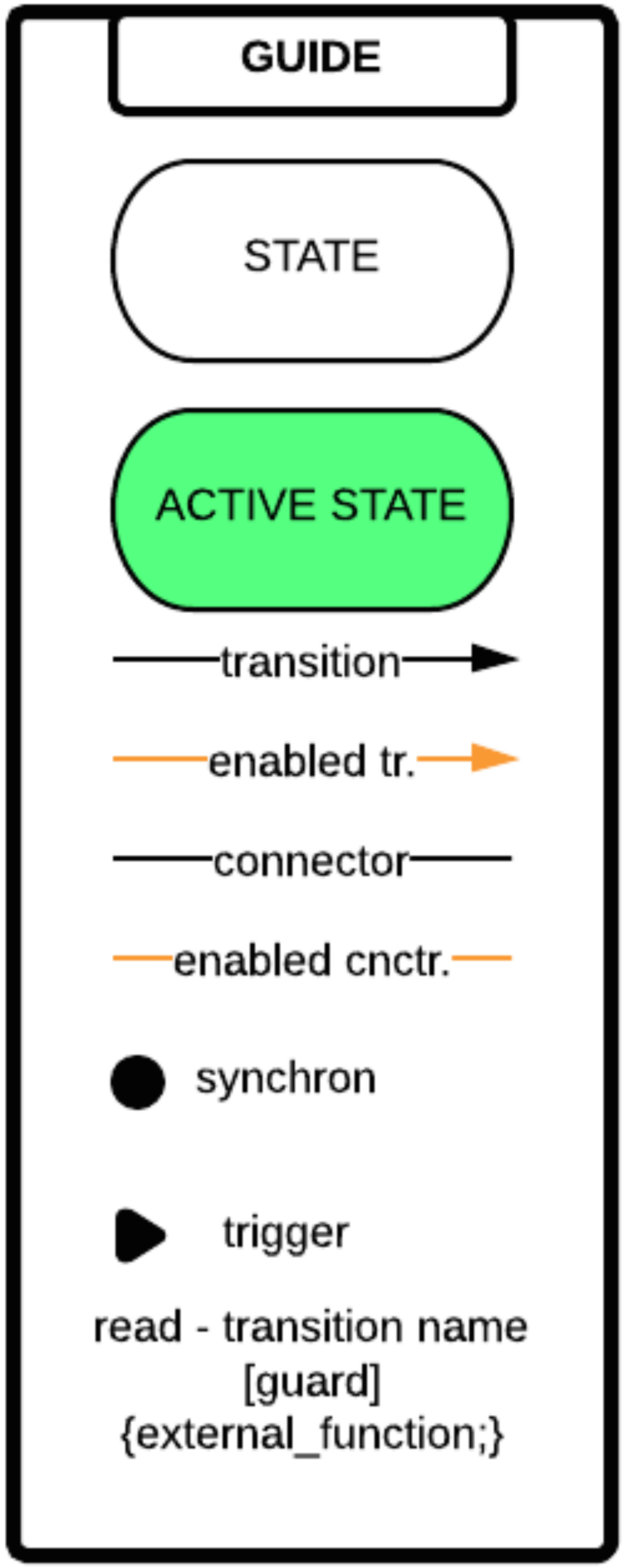
Status

enable _HK

disable _HK

disable _PS

enable_ PS



GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

connector

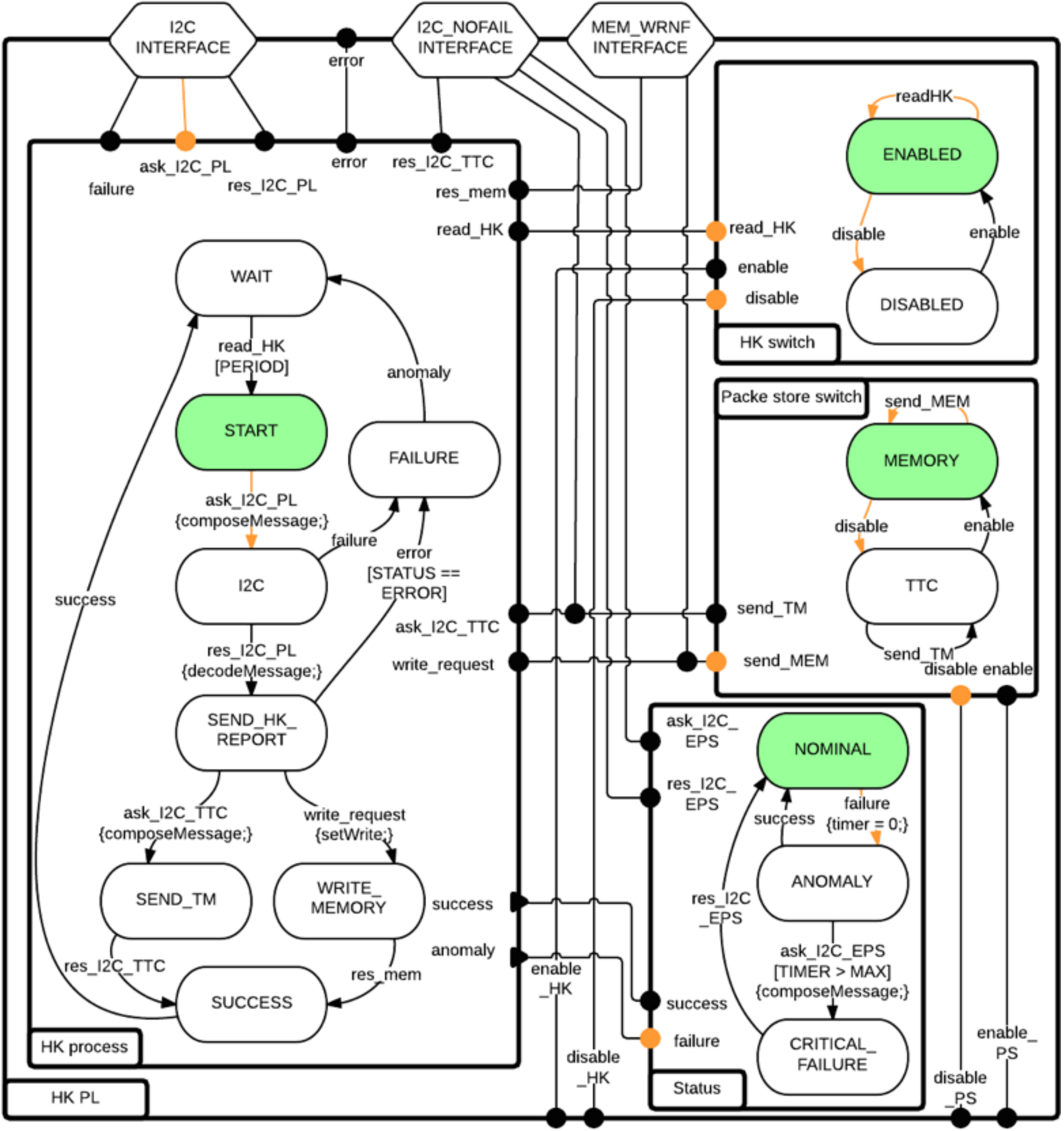
enabled cnctr.

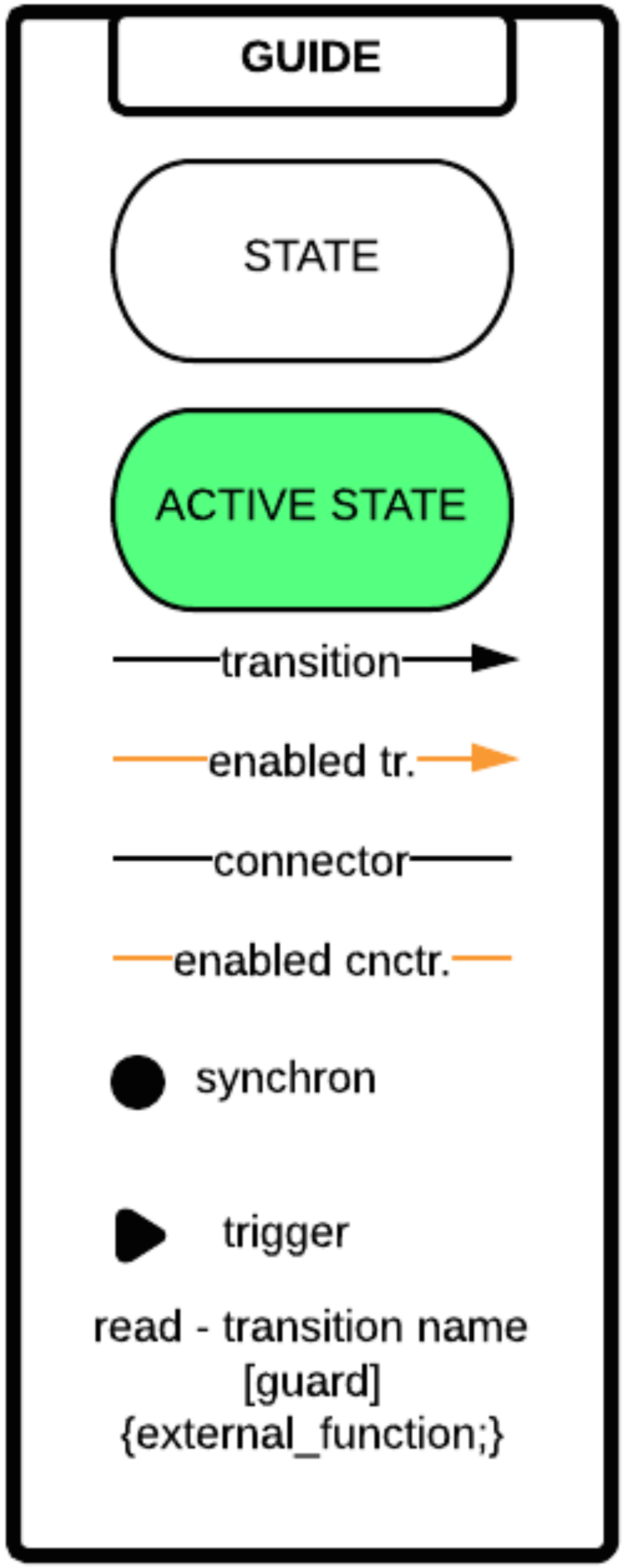
synchron

trigger

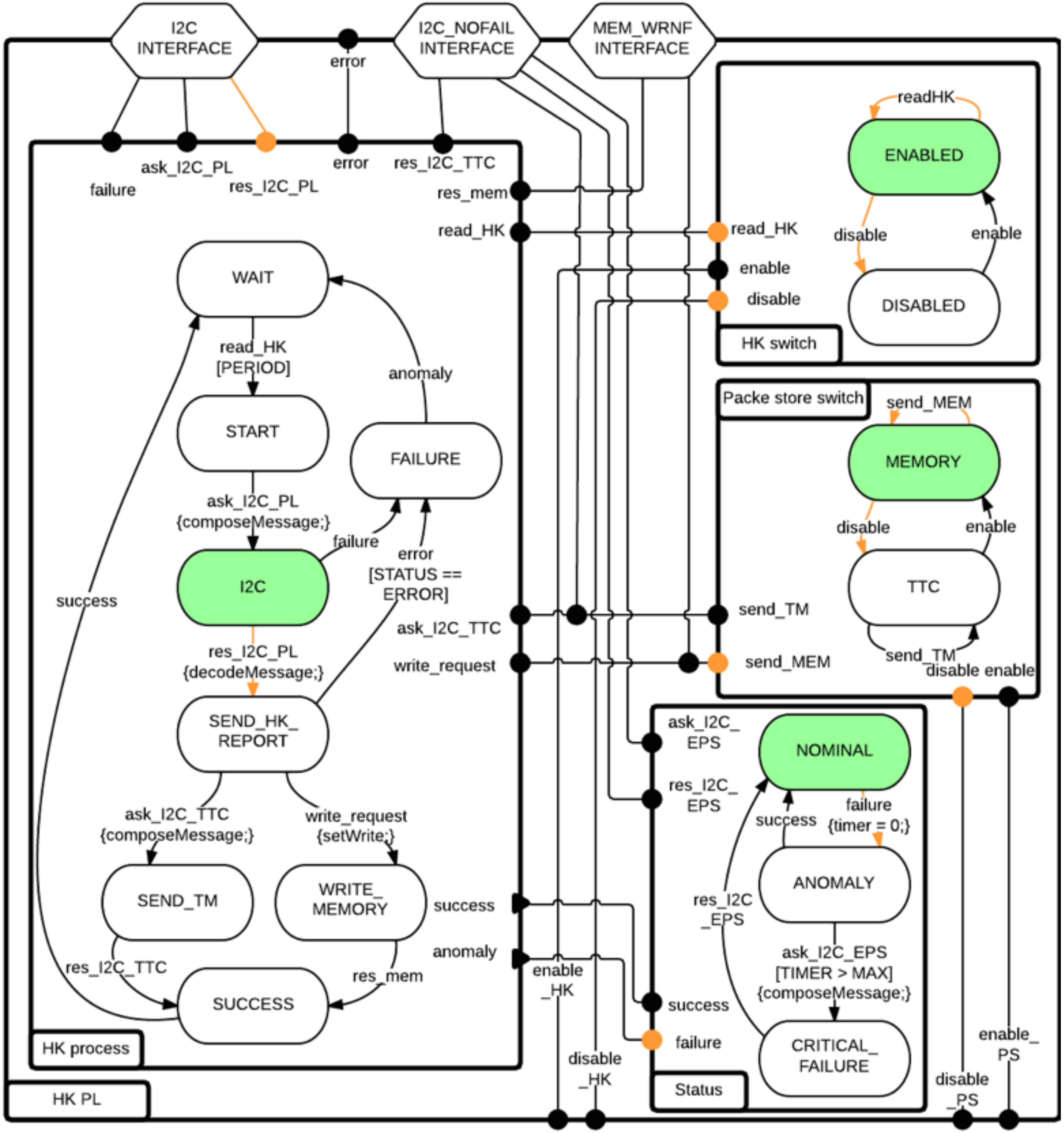
read - transition name [guard] {external_function;}

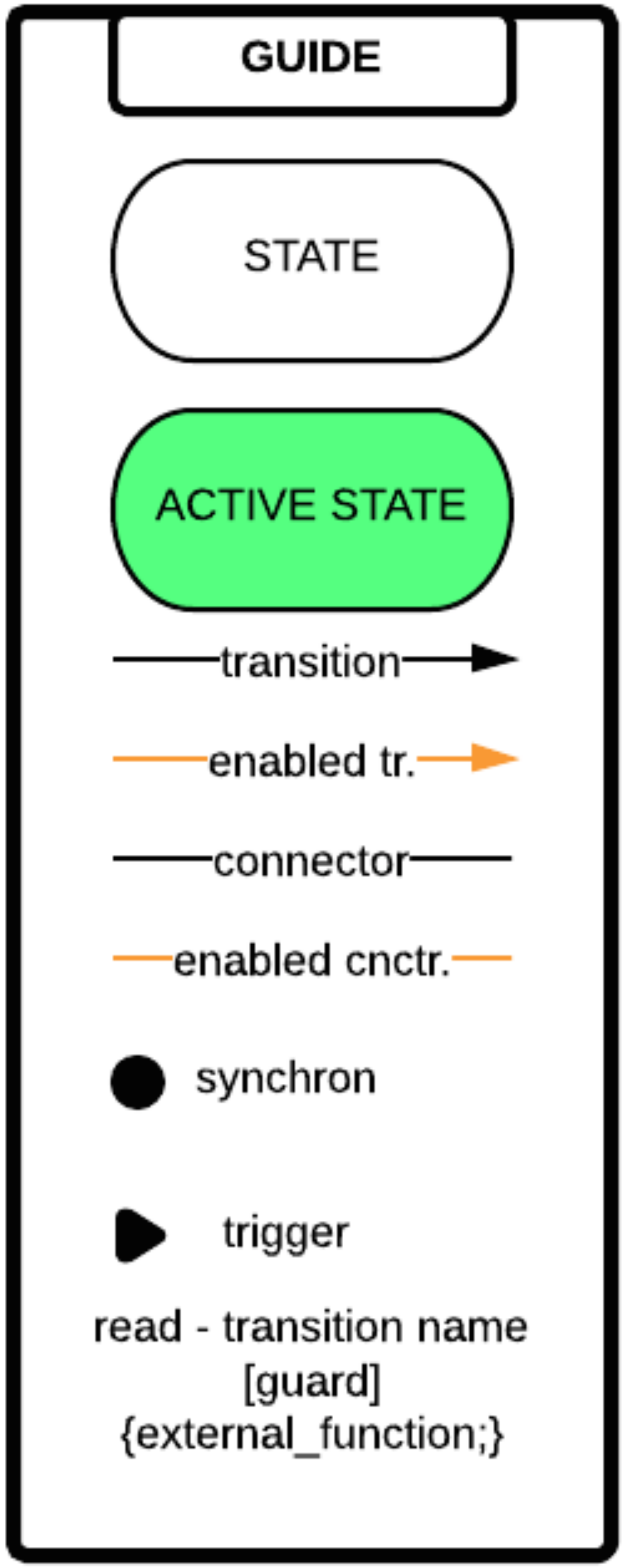
slide courtesy of Marco Pagnamenta



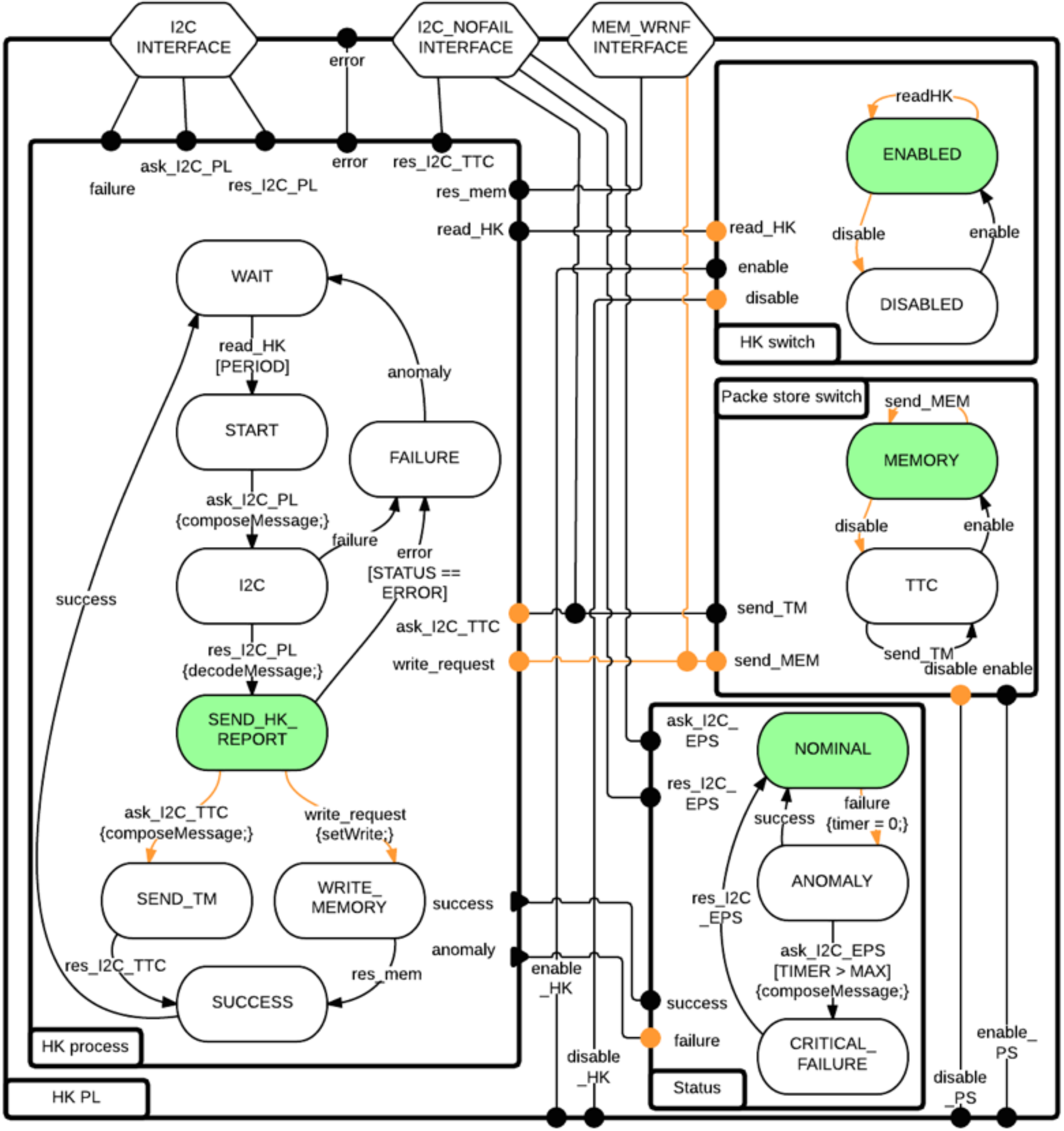


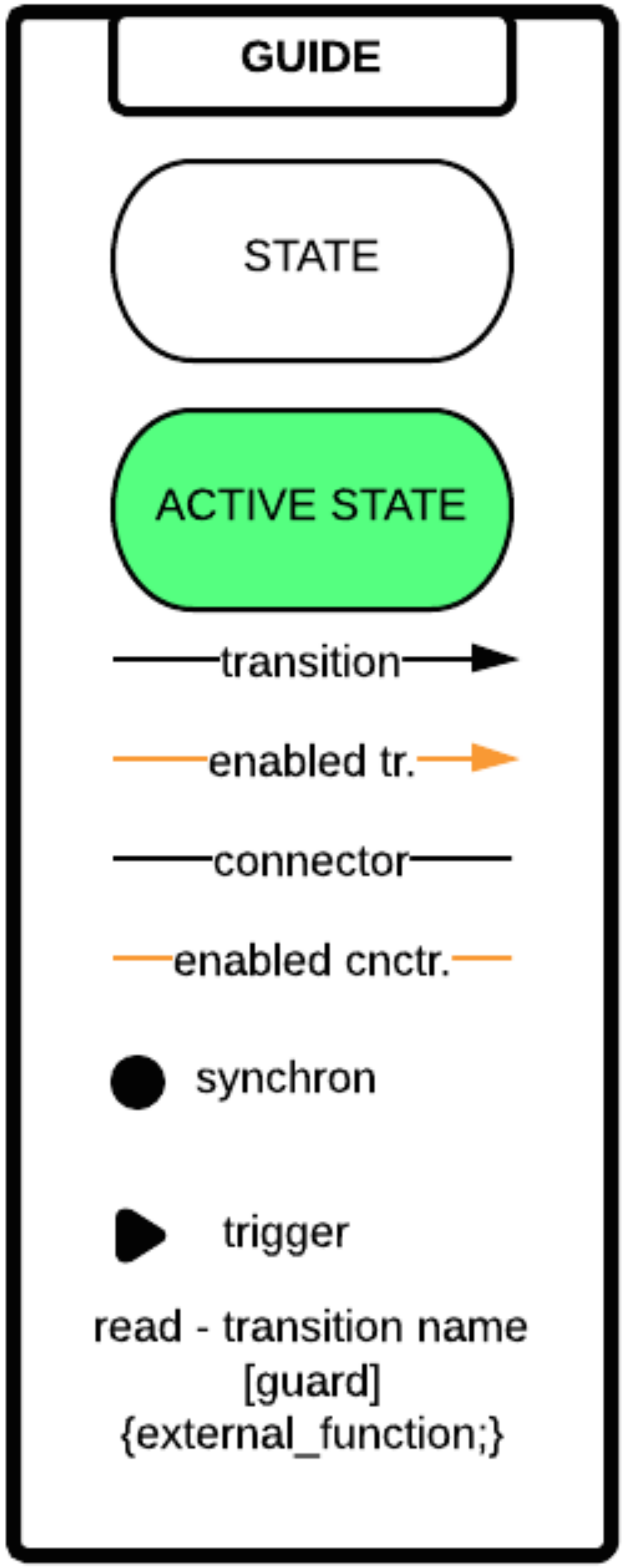
slide courtesy of
Marco Pagnamenta



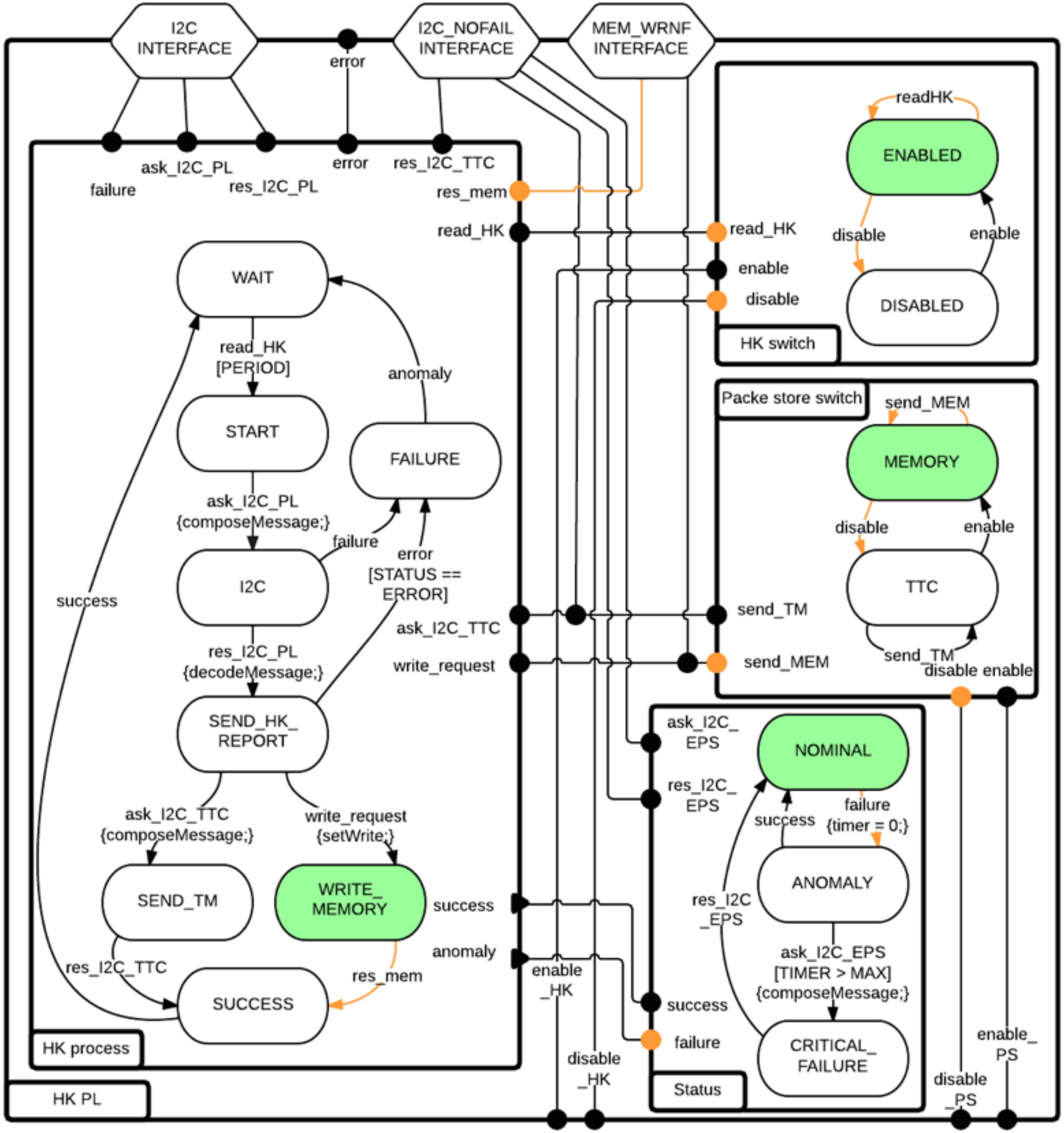


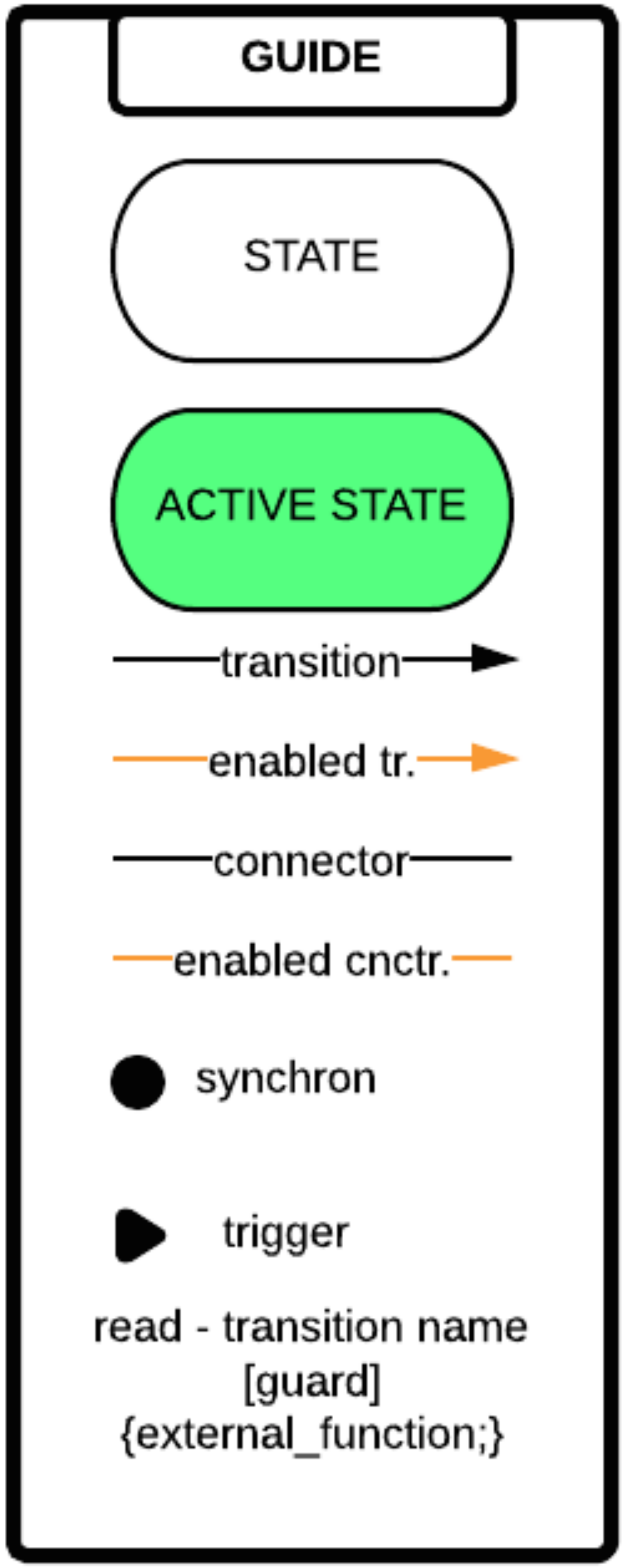
slide courtesy of
Marco Pagnamenta



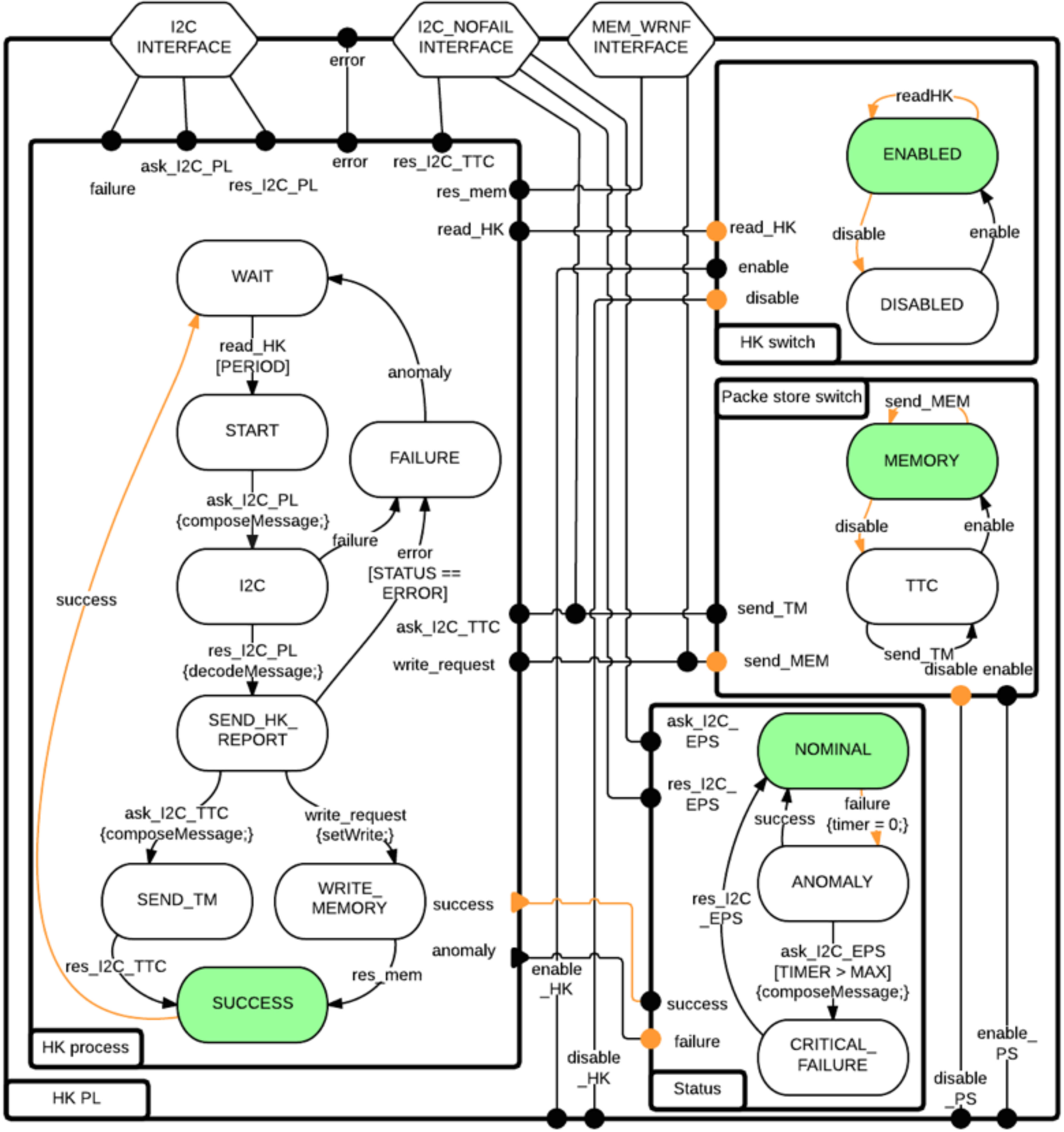


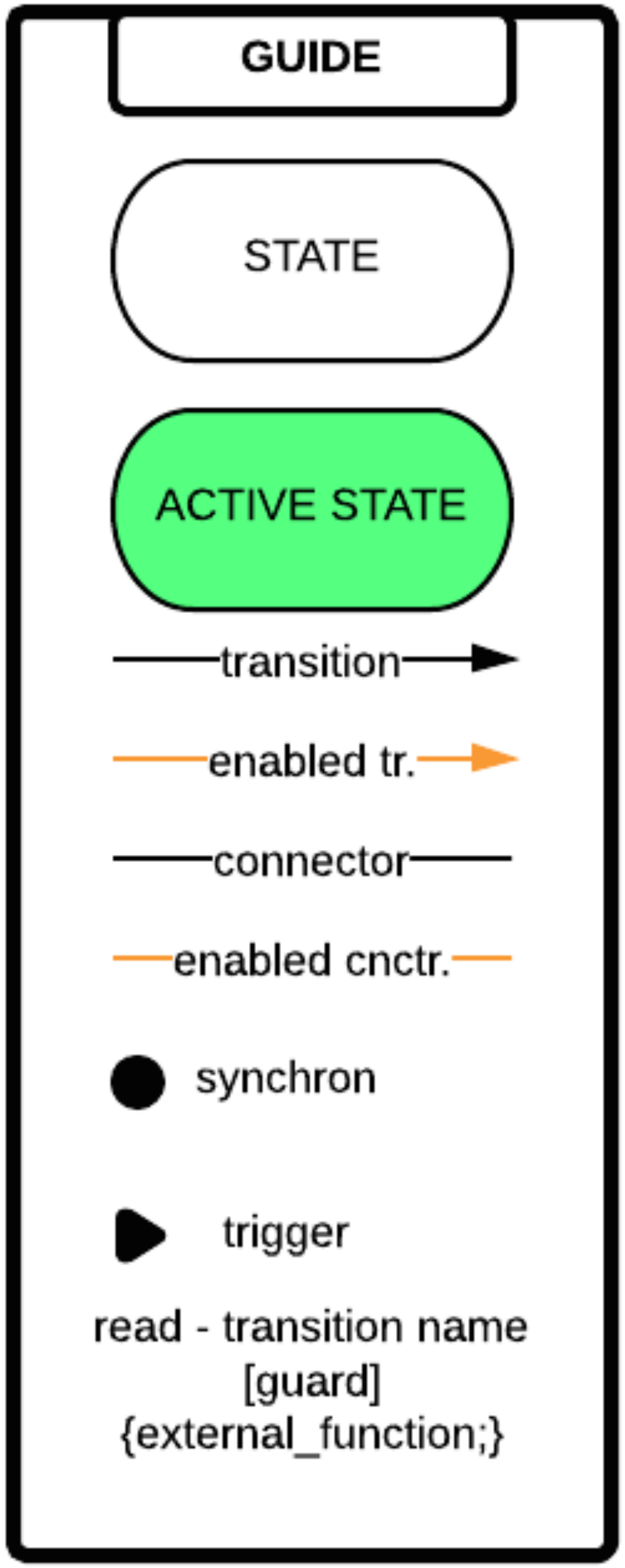
slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

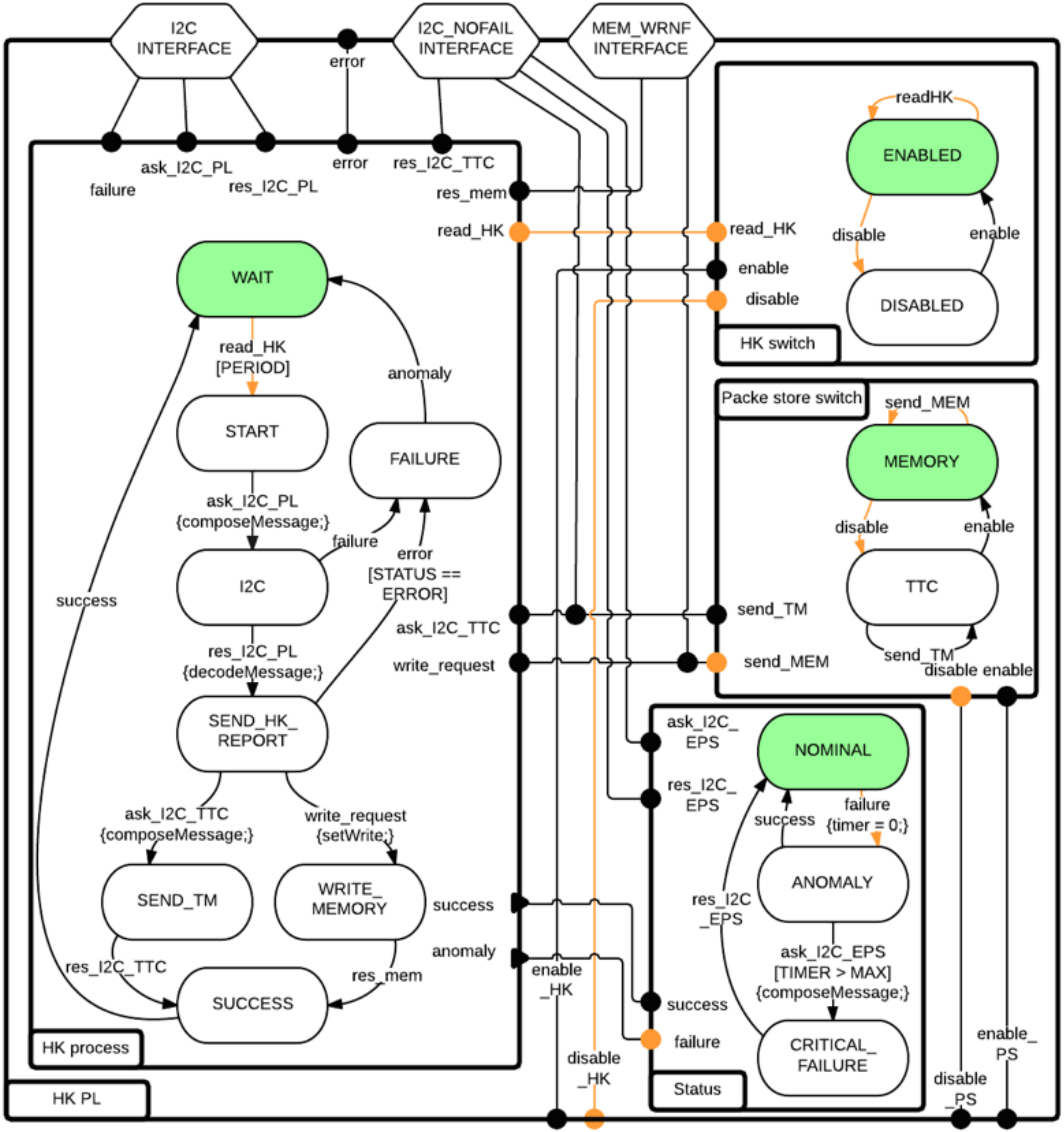


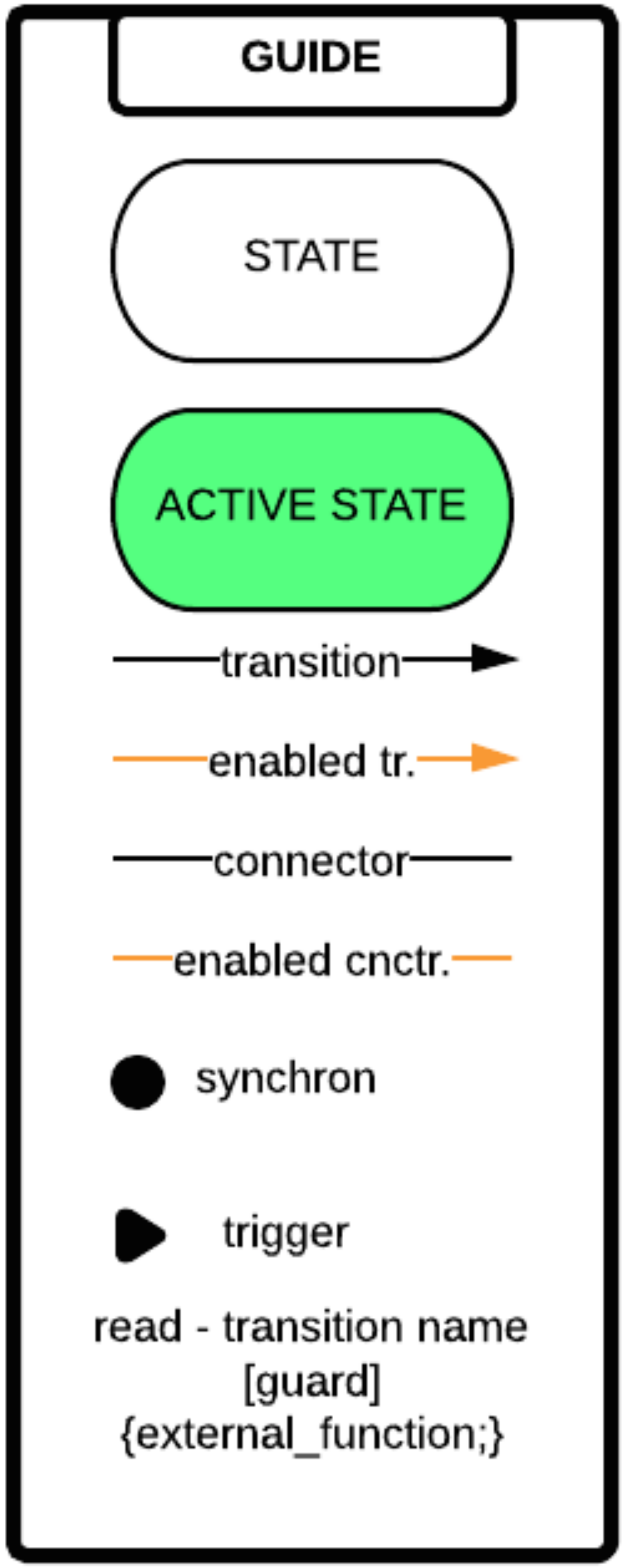


slide courtesy of
Marco Pagnamenta

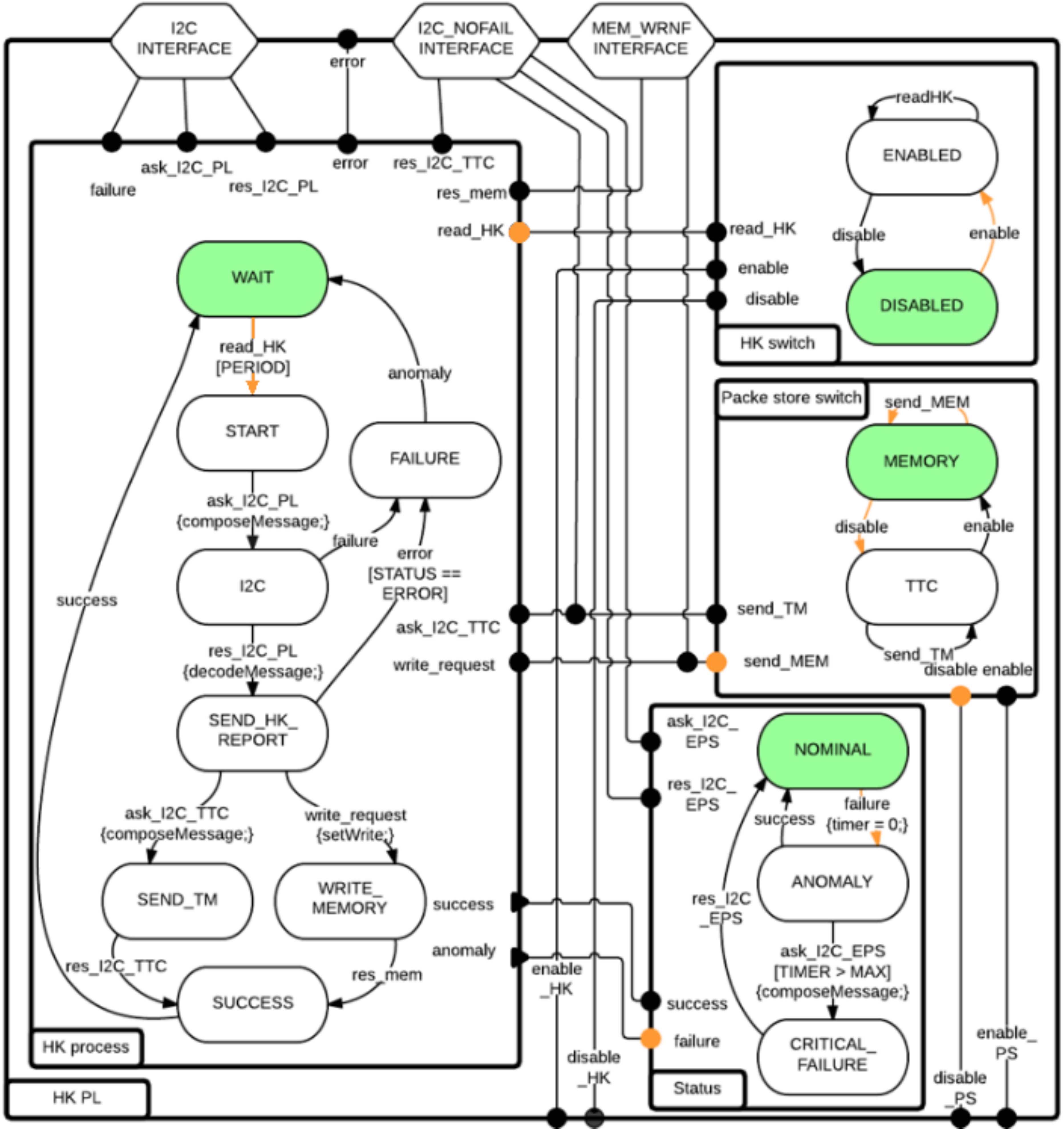
Example 2

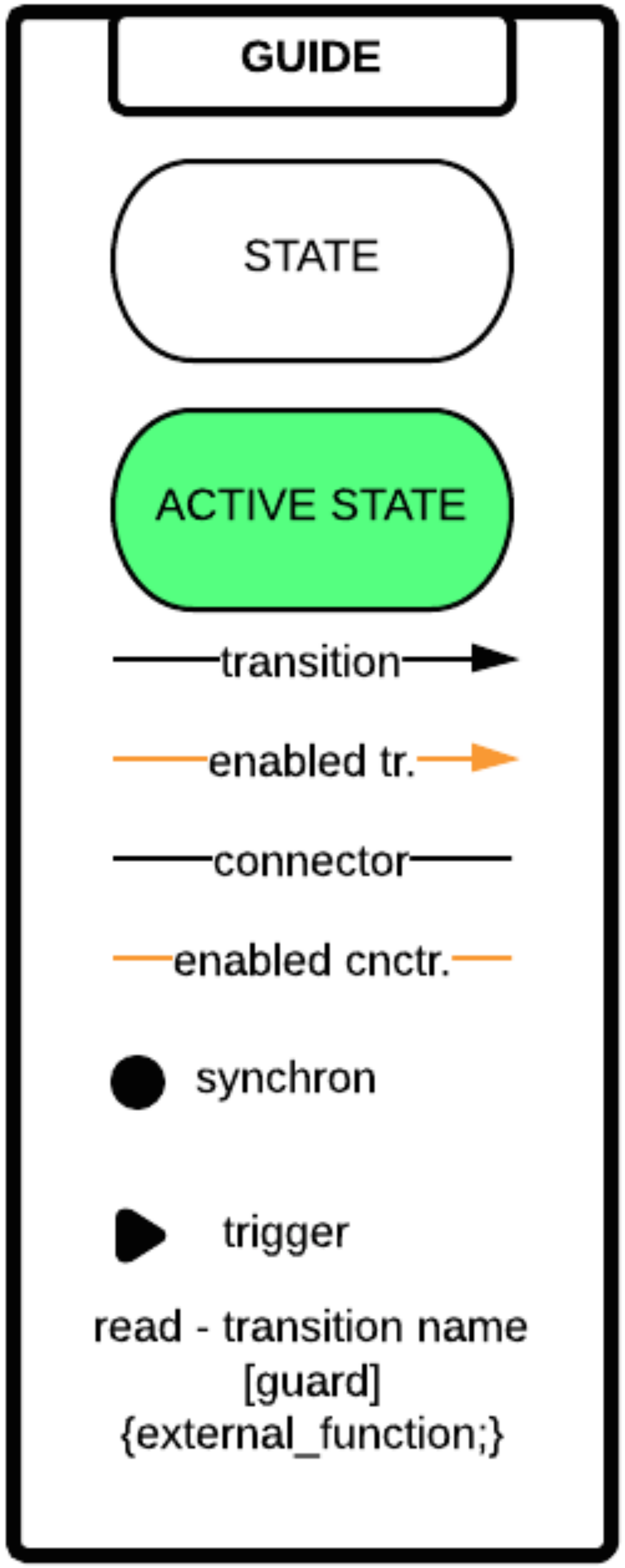
Stopping housekeeping





slide courtesy of
Marco Pagnamenta

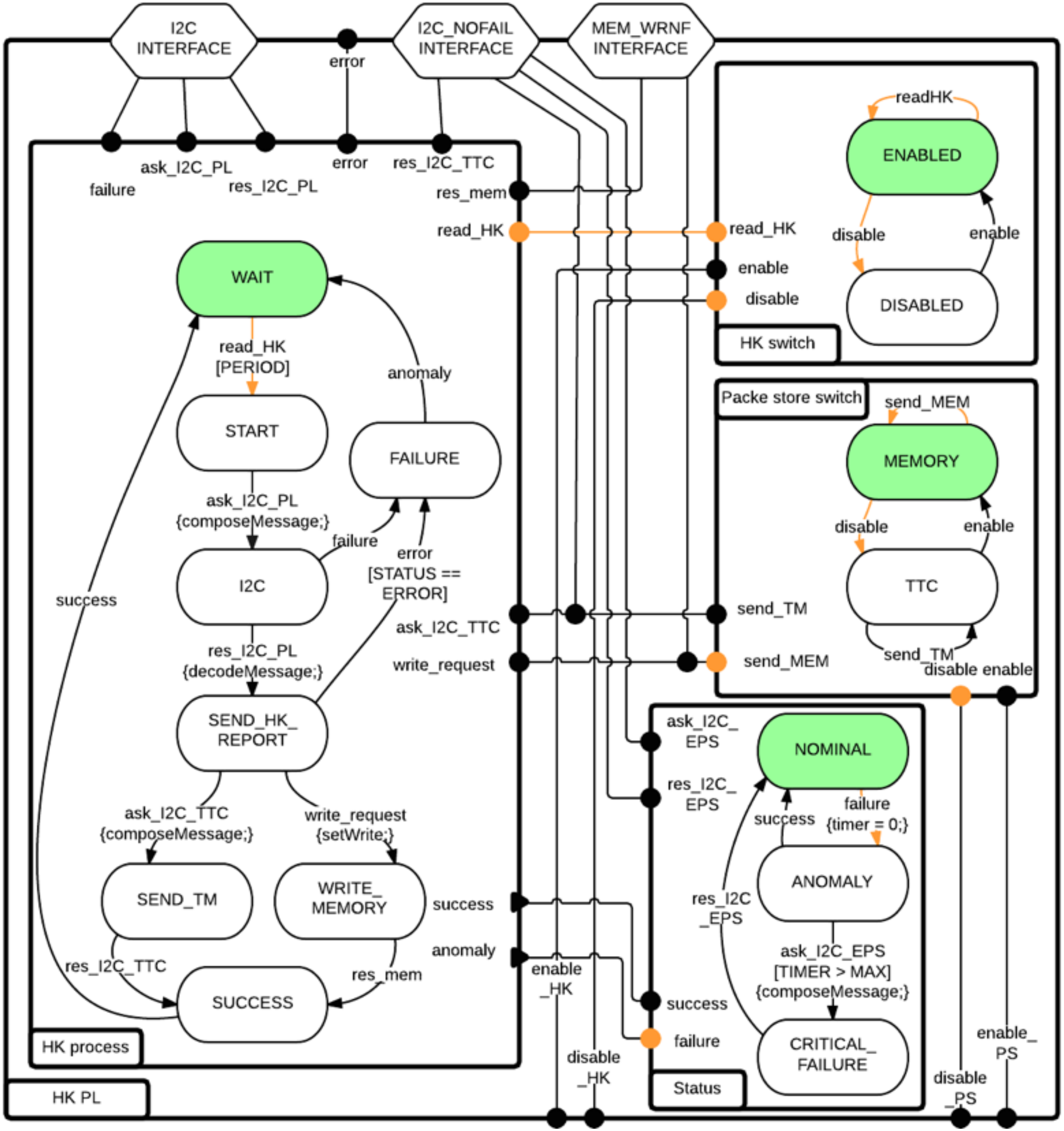


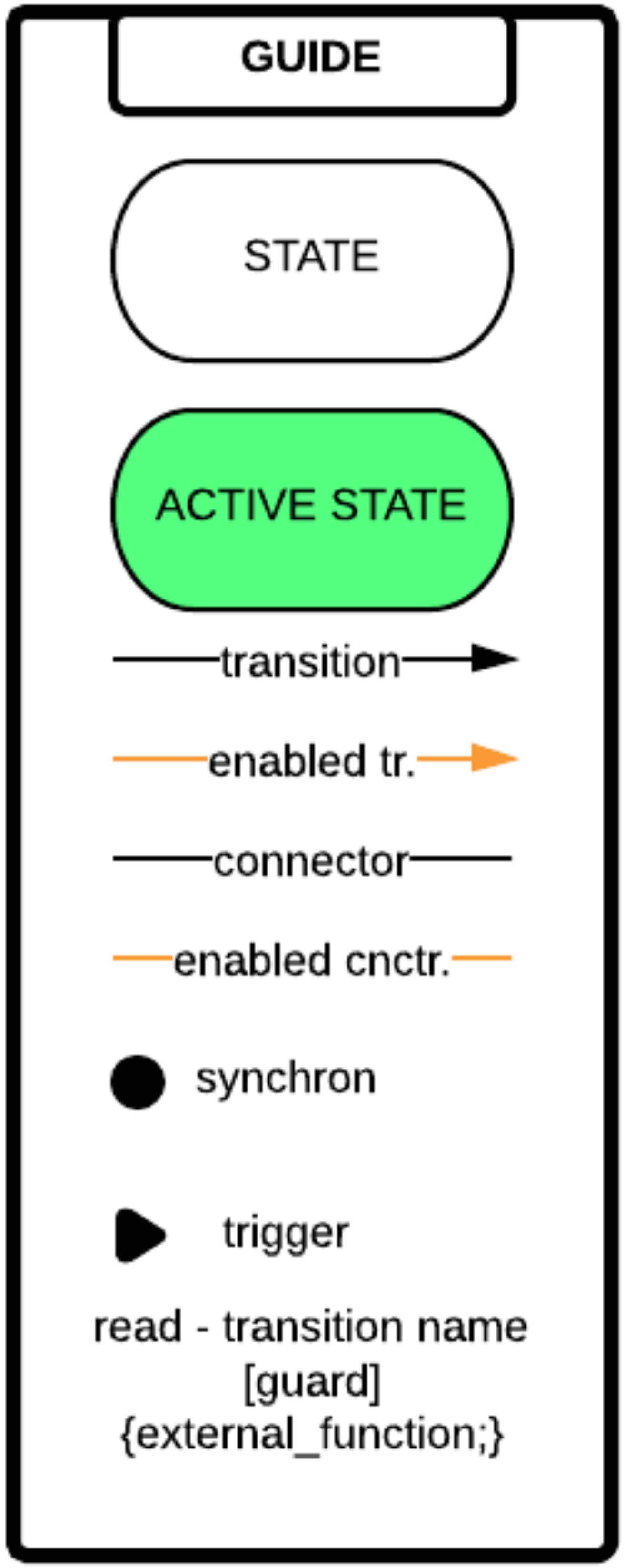


slide courtesy of
Marco Pagnamenta

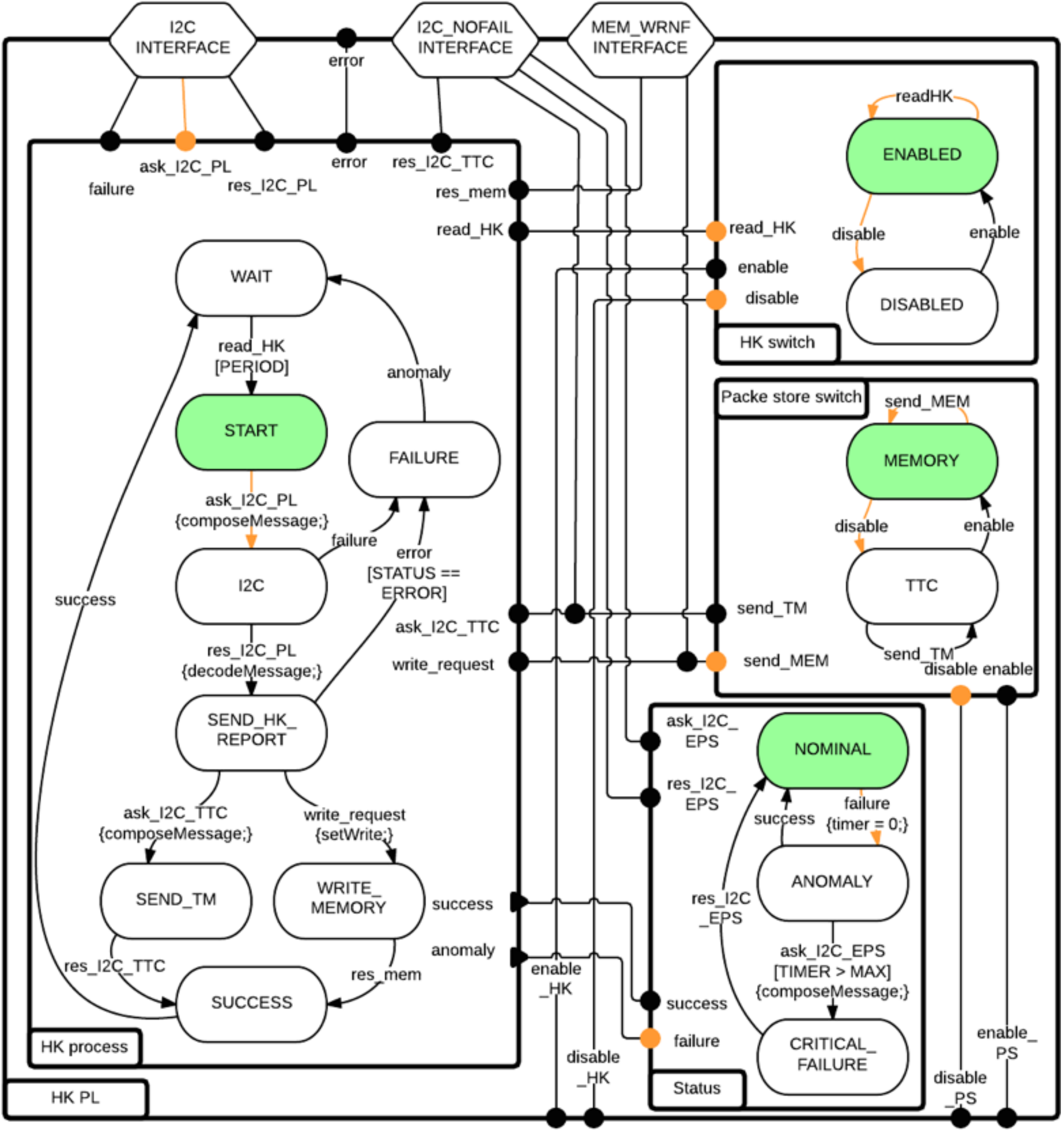
Example 3

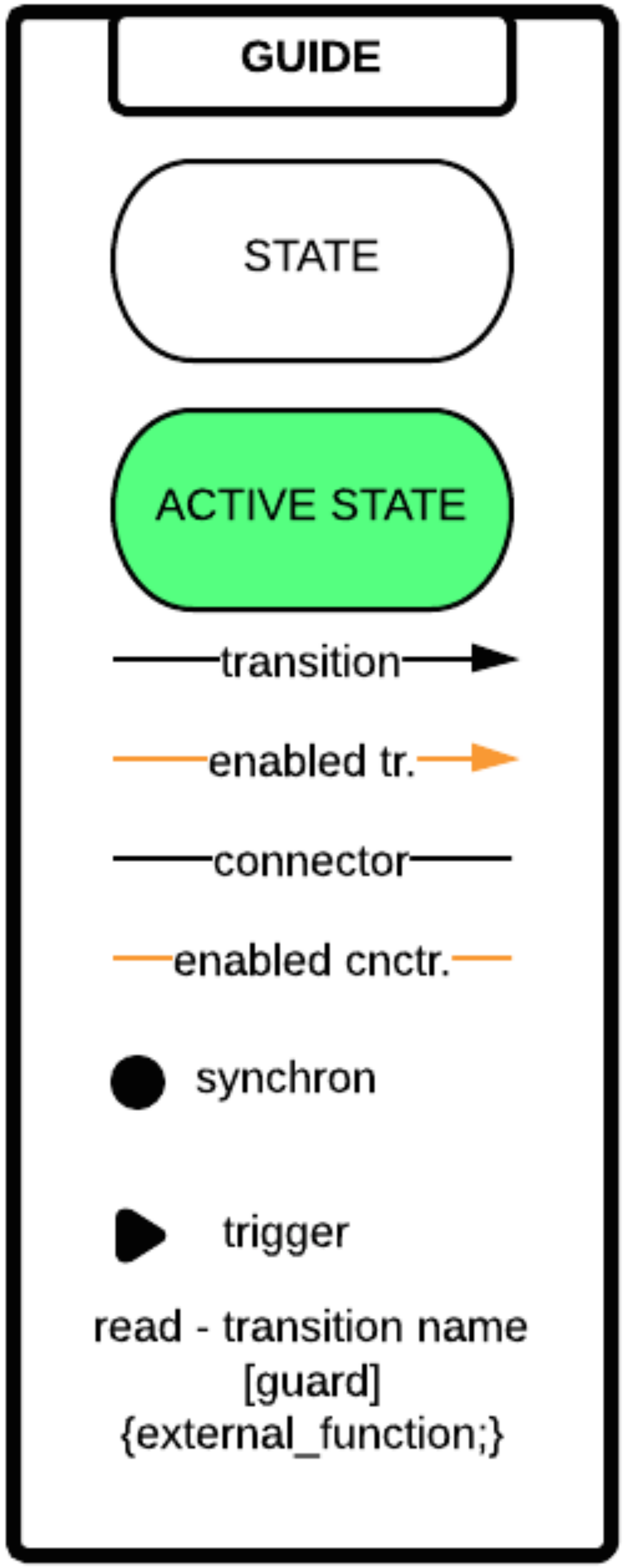
Switching destination of housekeeping data



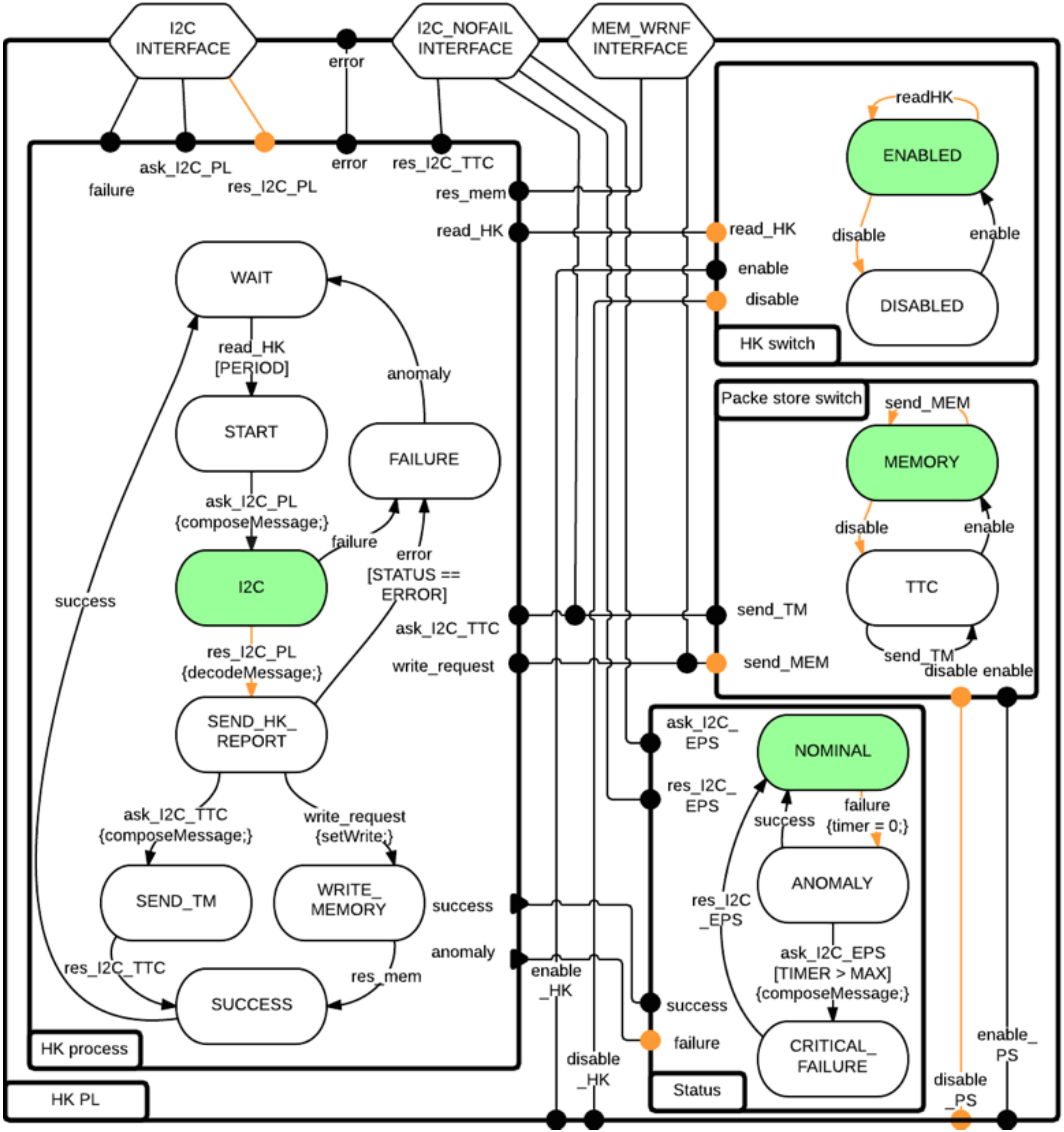


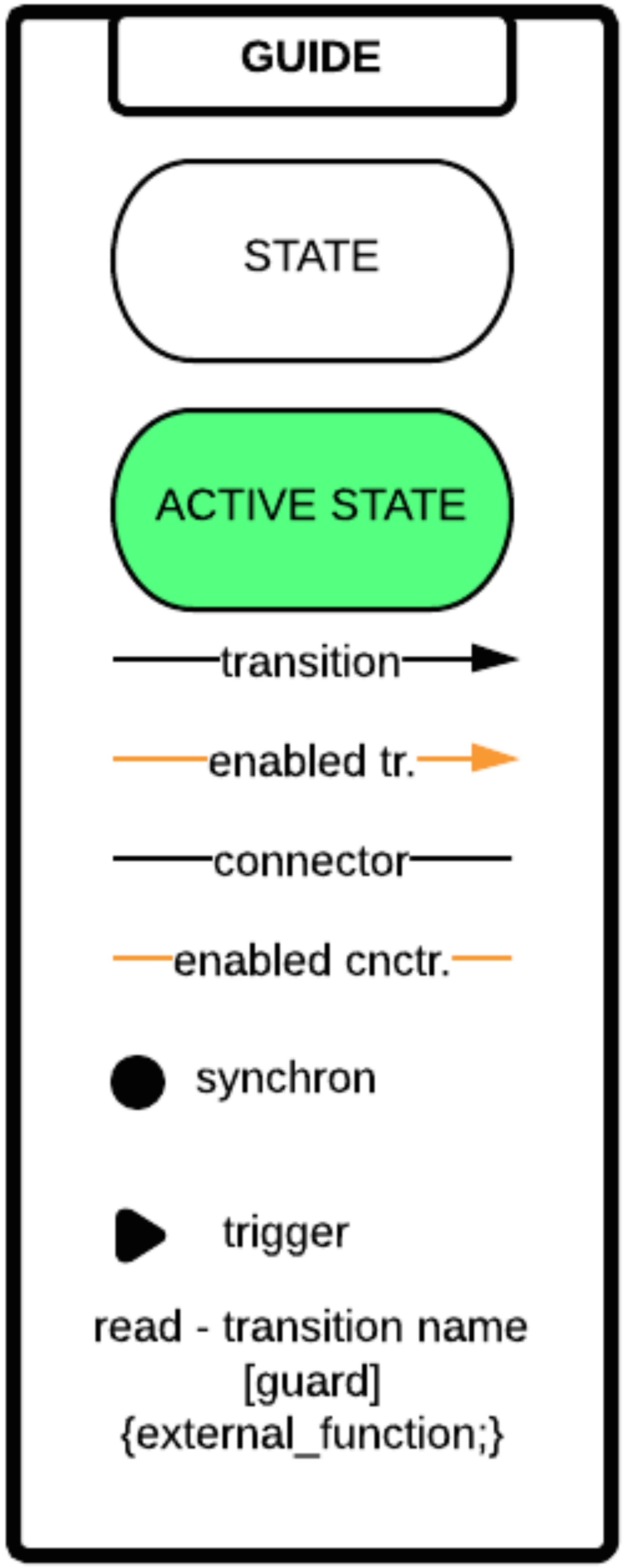
slide courtesy of
Marco Pagnamenta



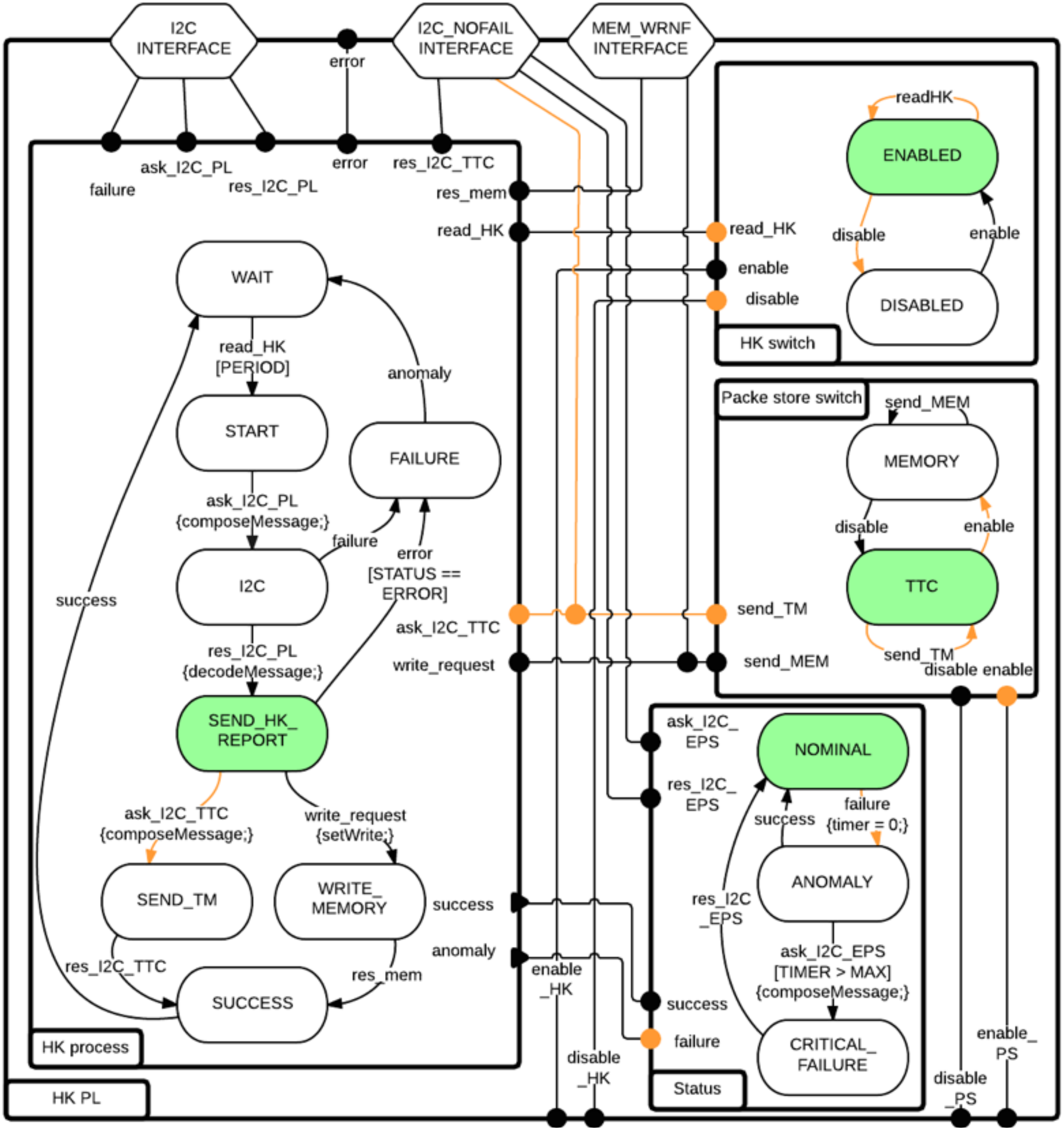


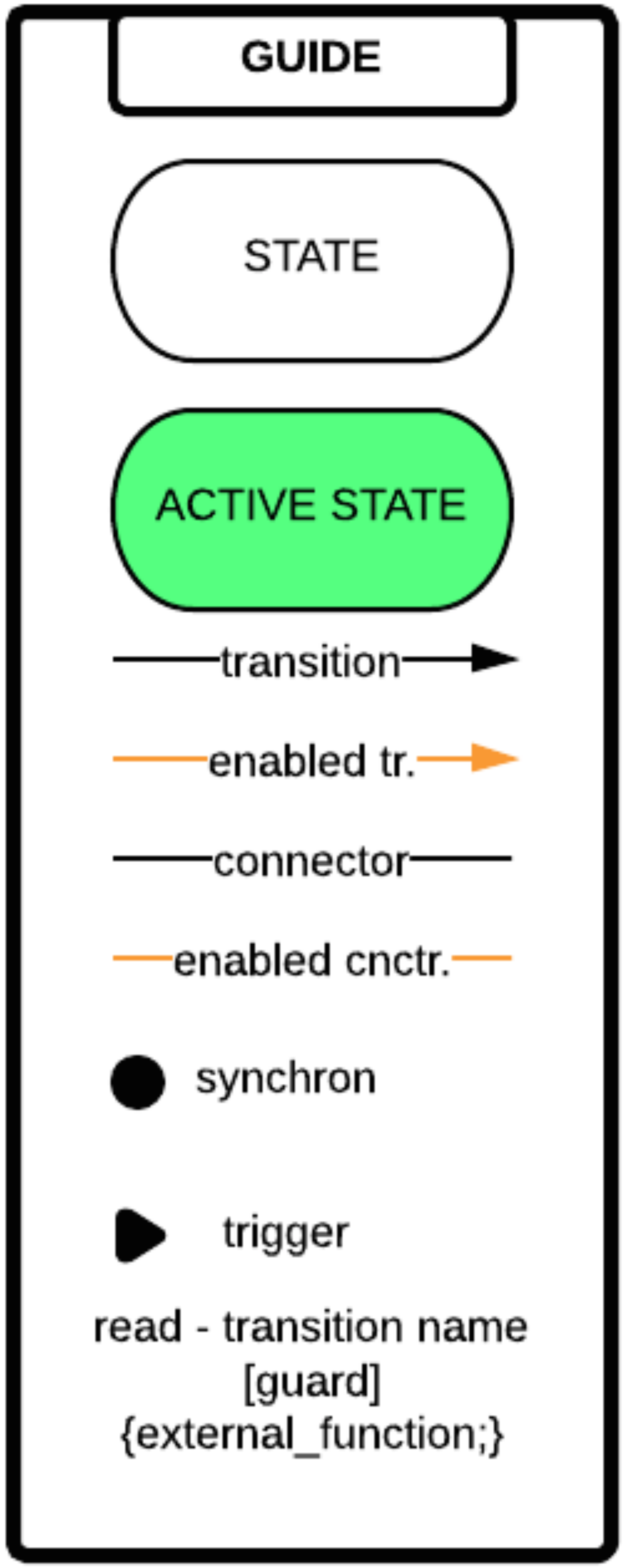
slide courtesy of
Marco Pagnamenta



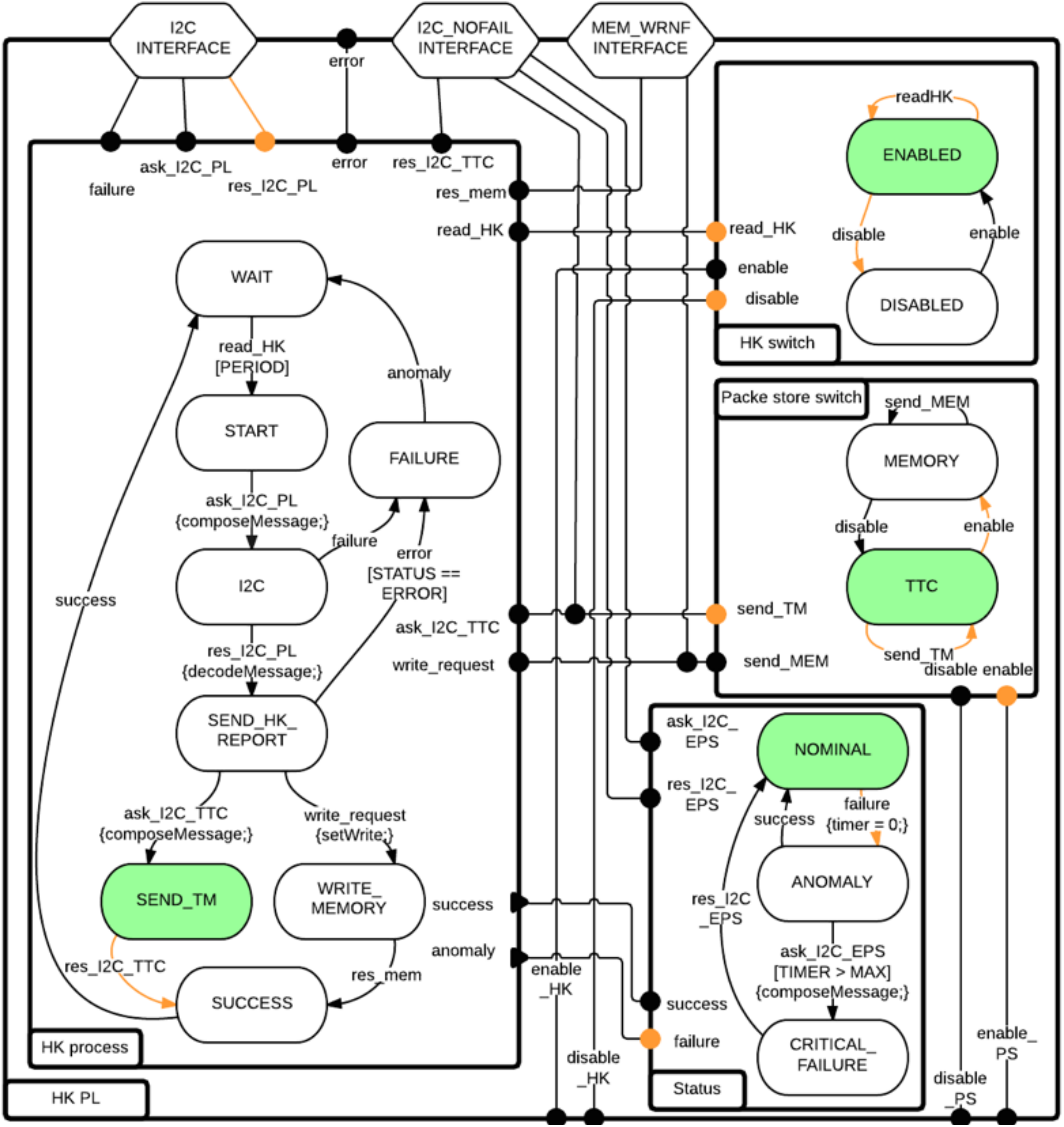


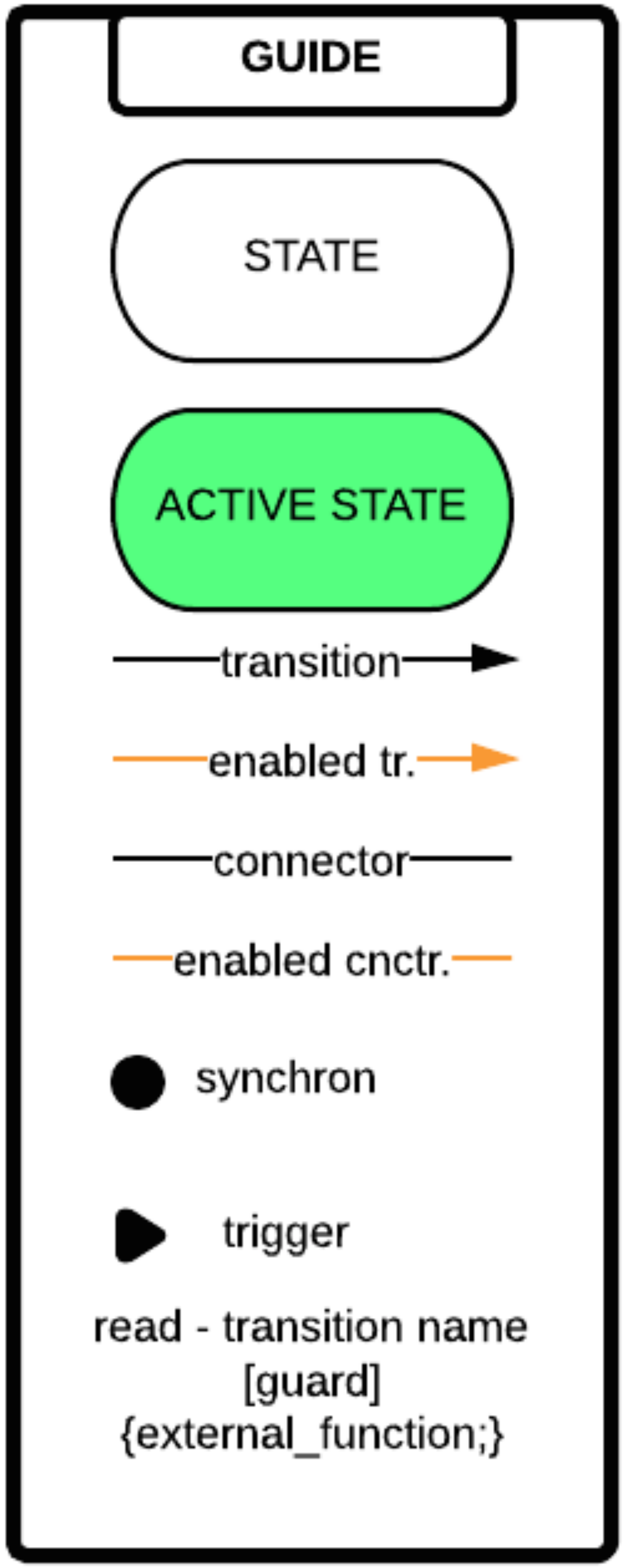
slide courtesy of
Marco Pagnamenta



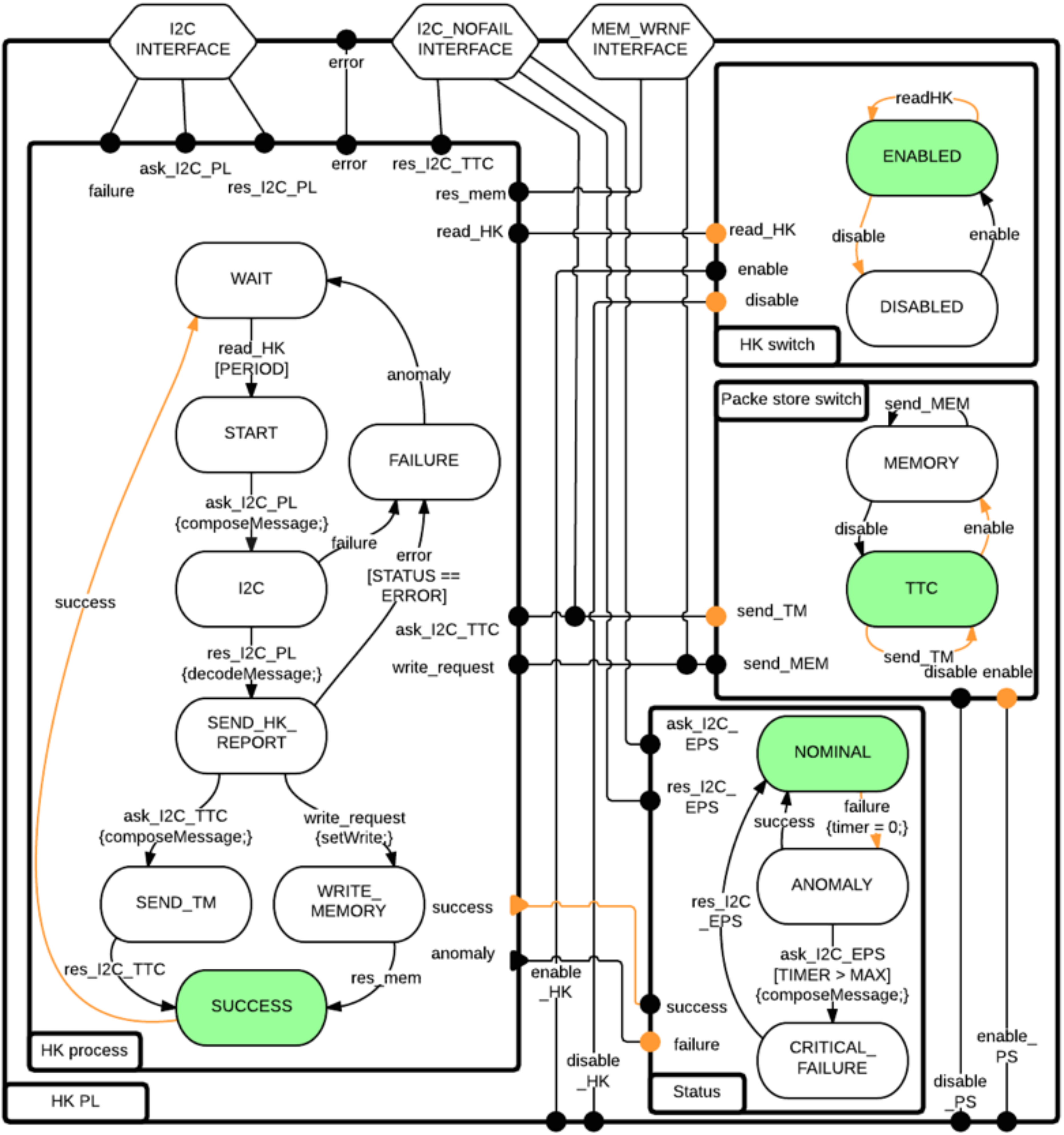


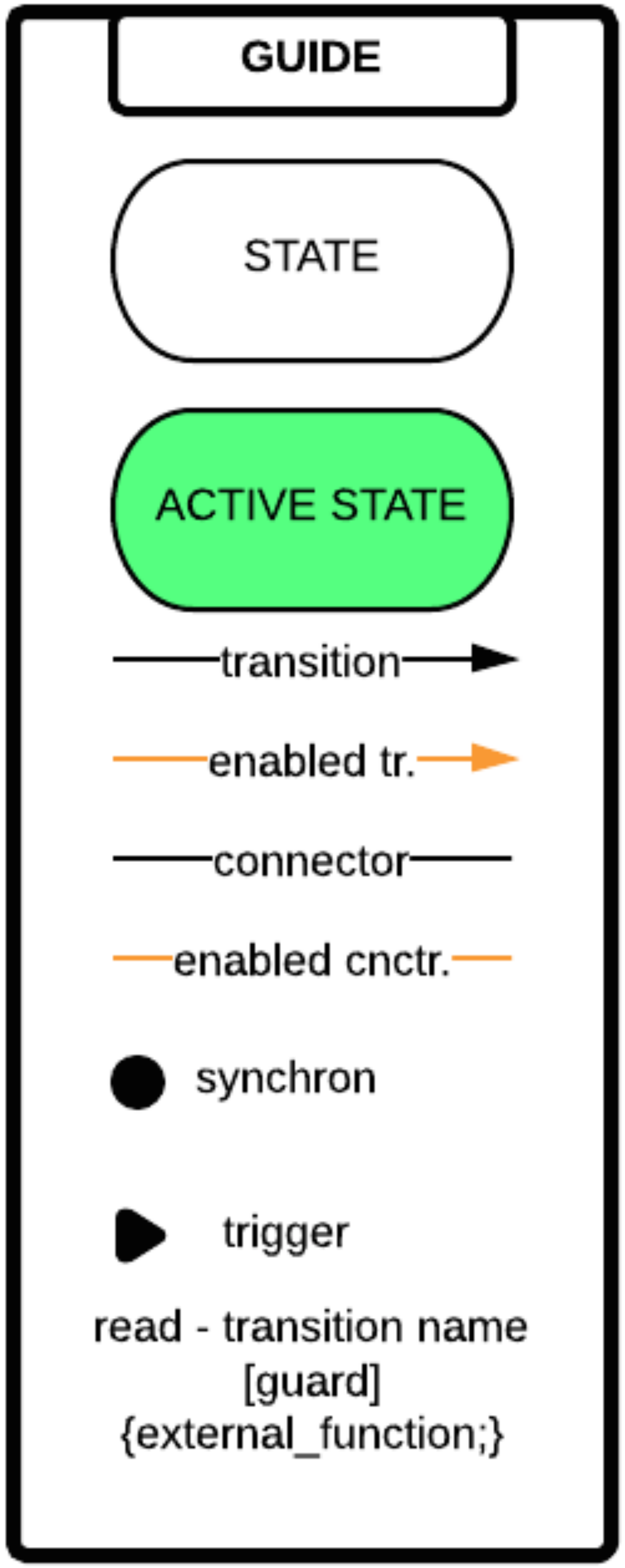
slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

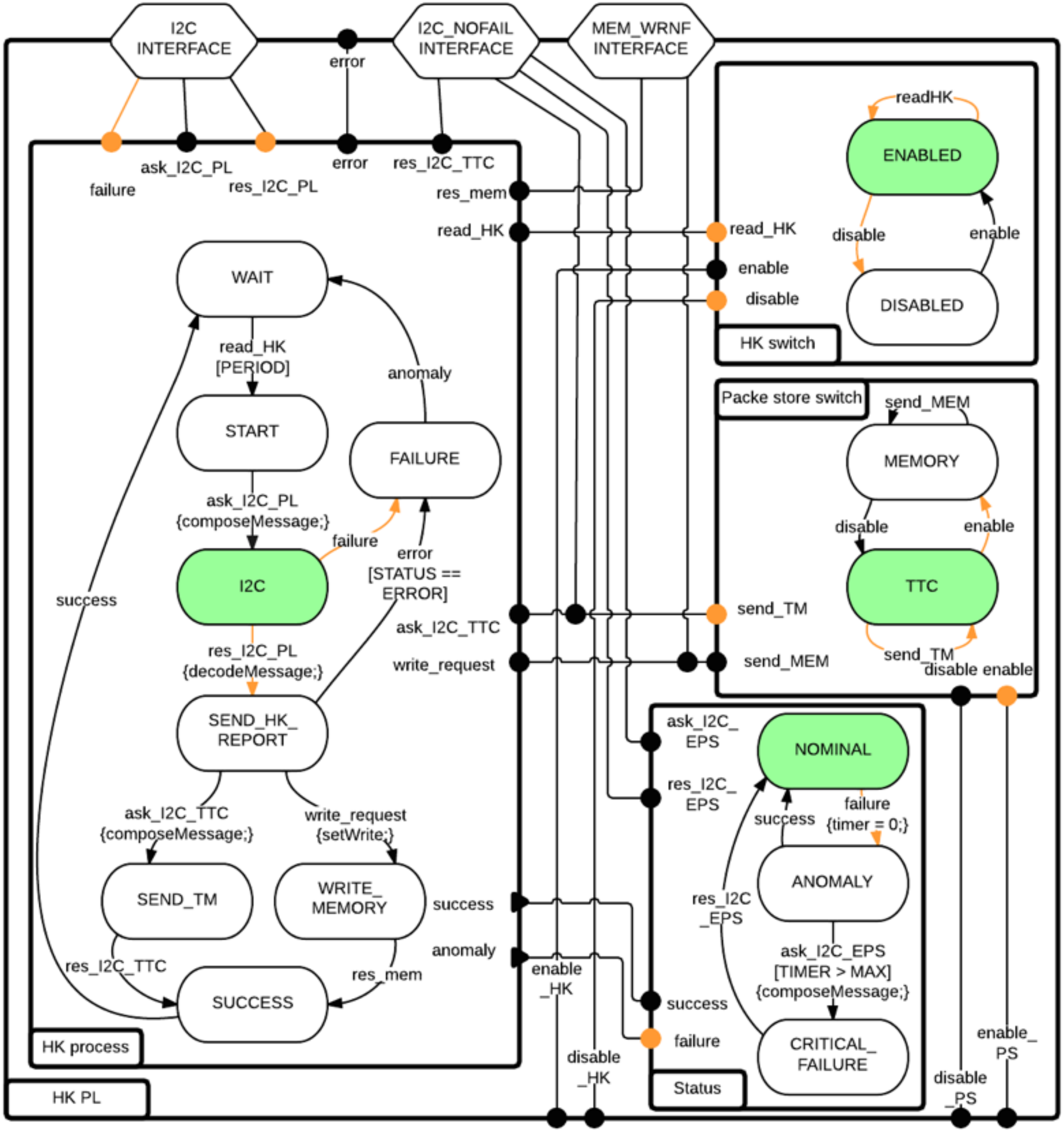


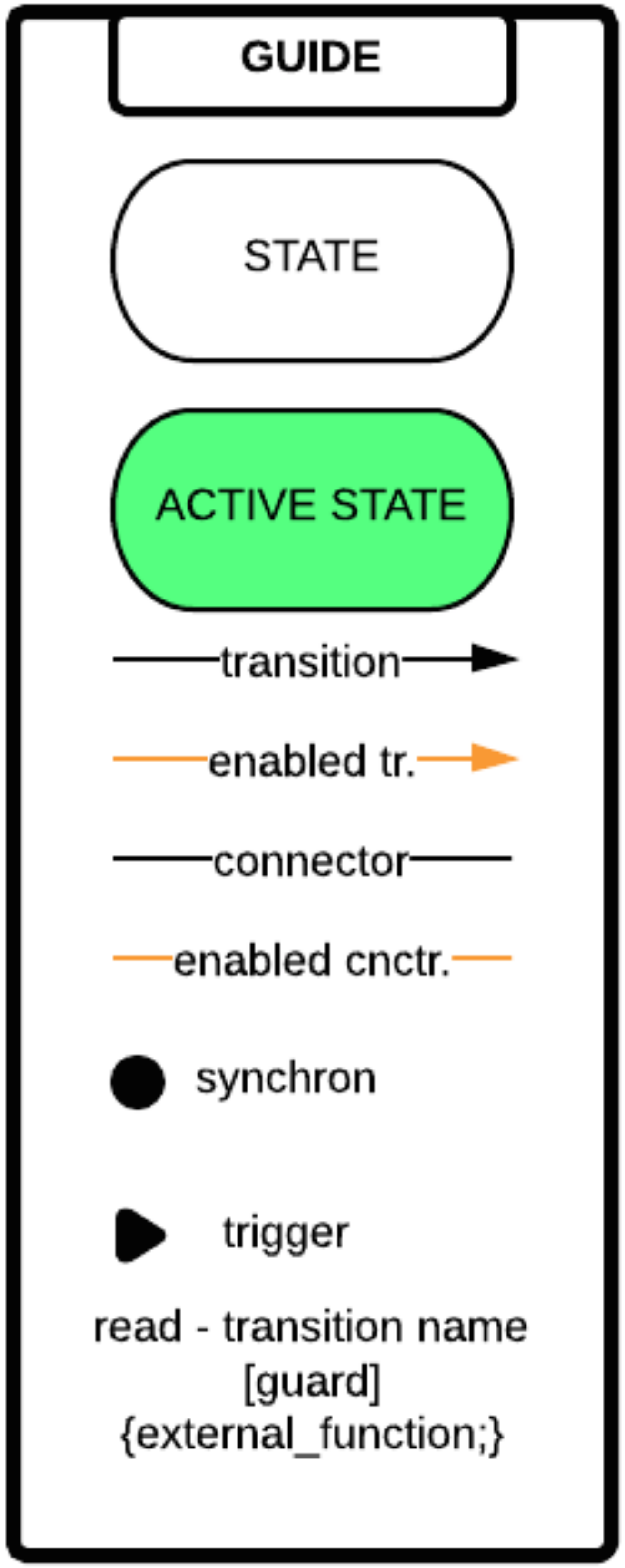


slide courtesy of
Marco Pagnamenta

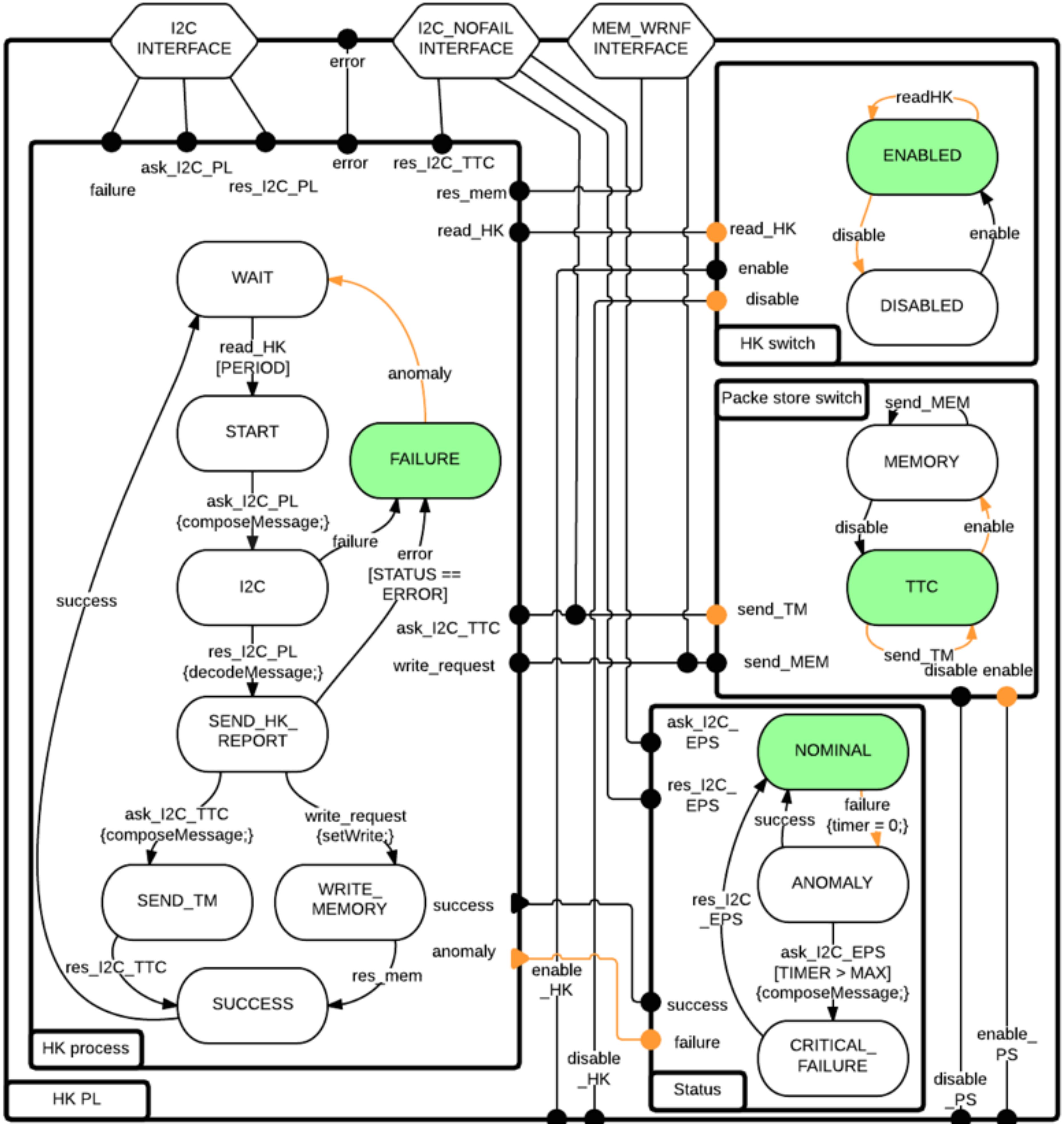
Example 4

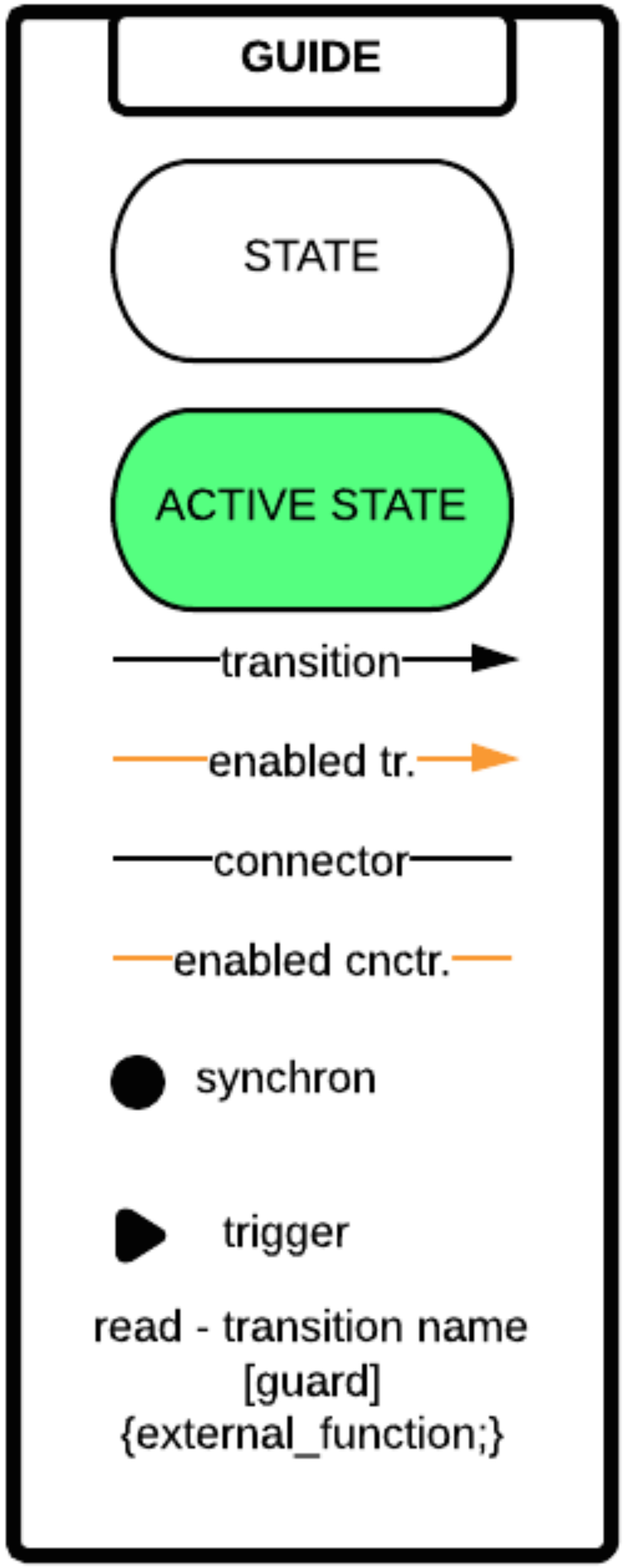
I^{2}C bus failure management



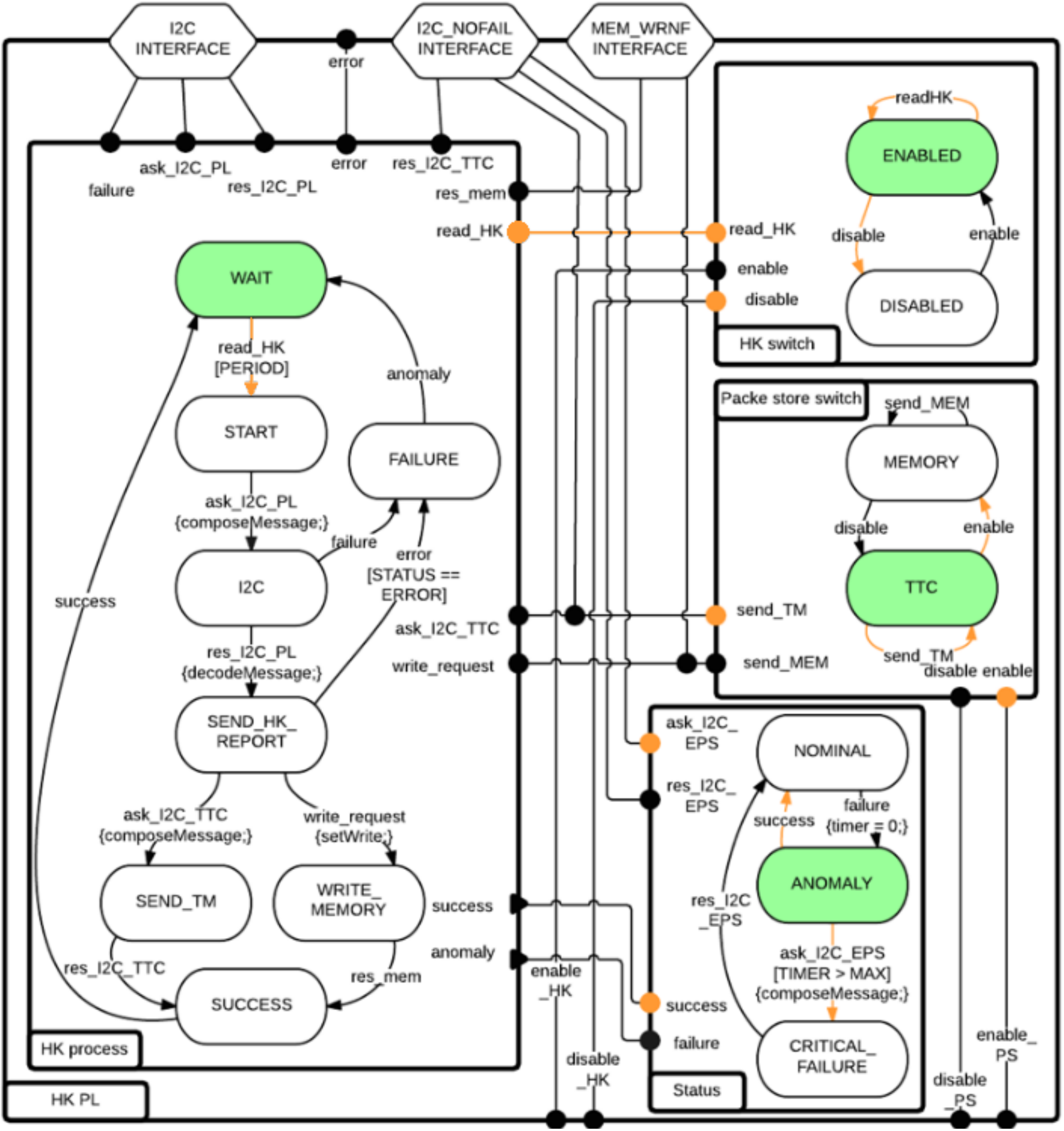


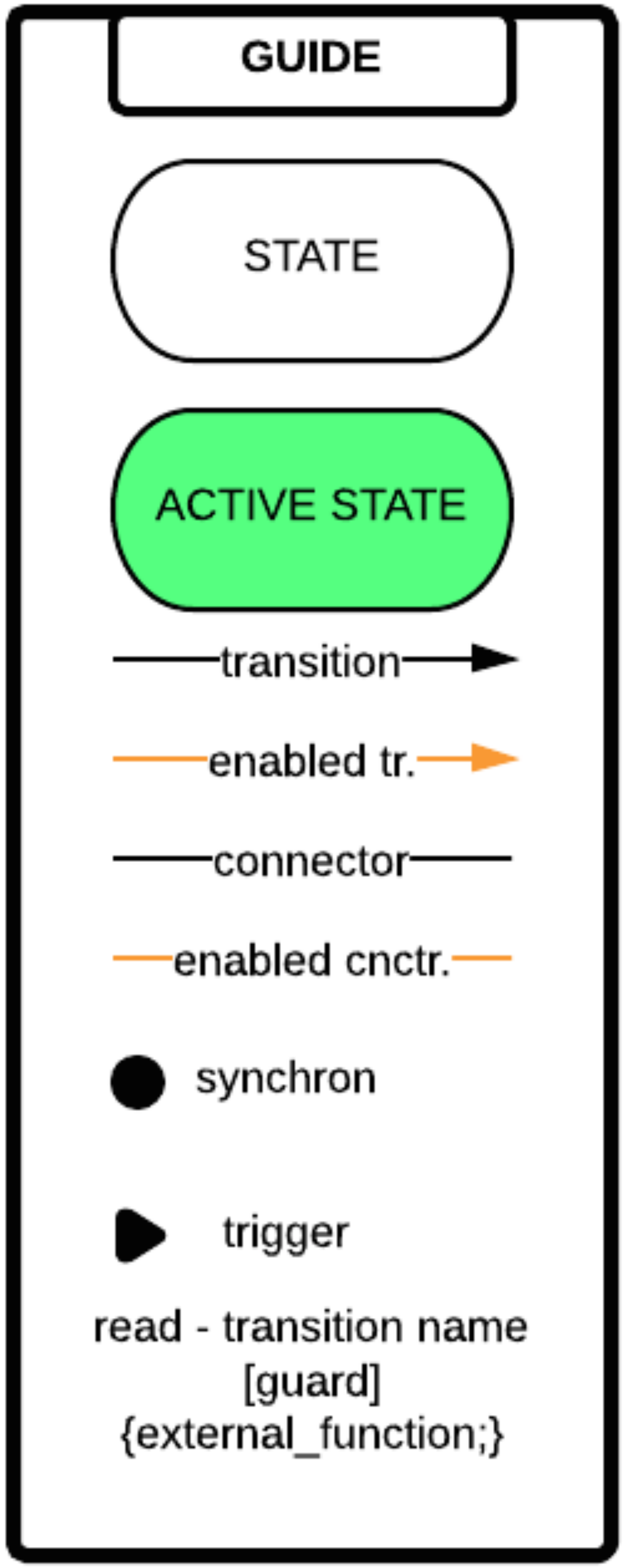
slide courtesy of
Marco Pagnamenta





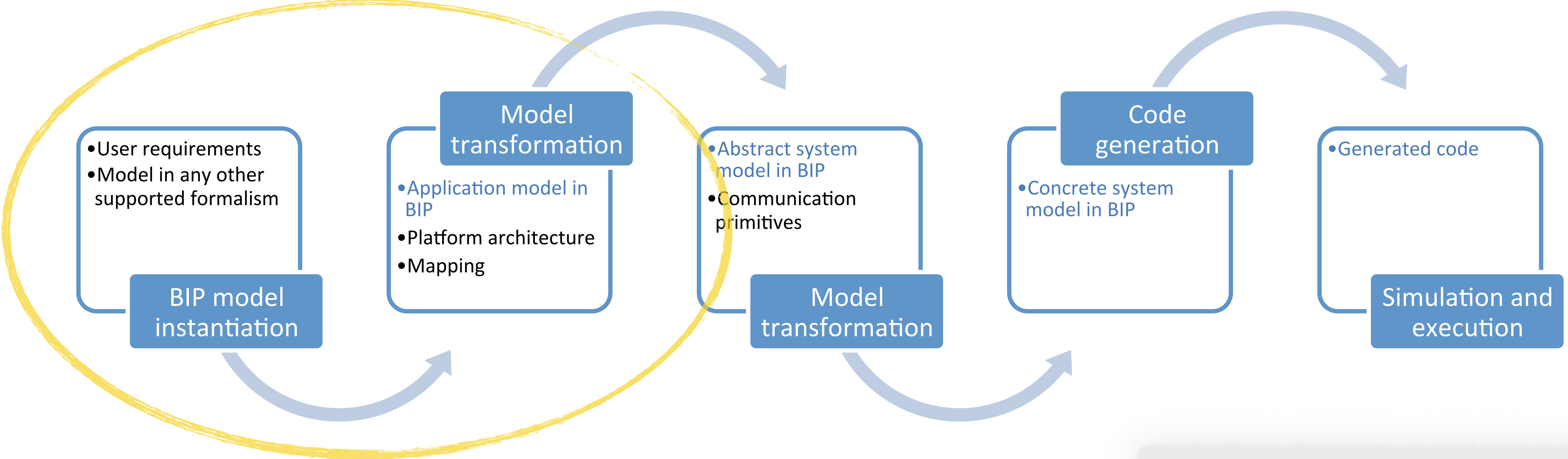
slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

Rigorous System Design flow



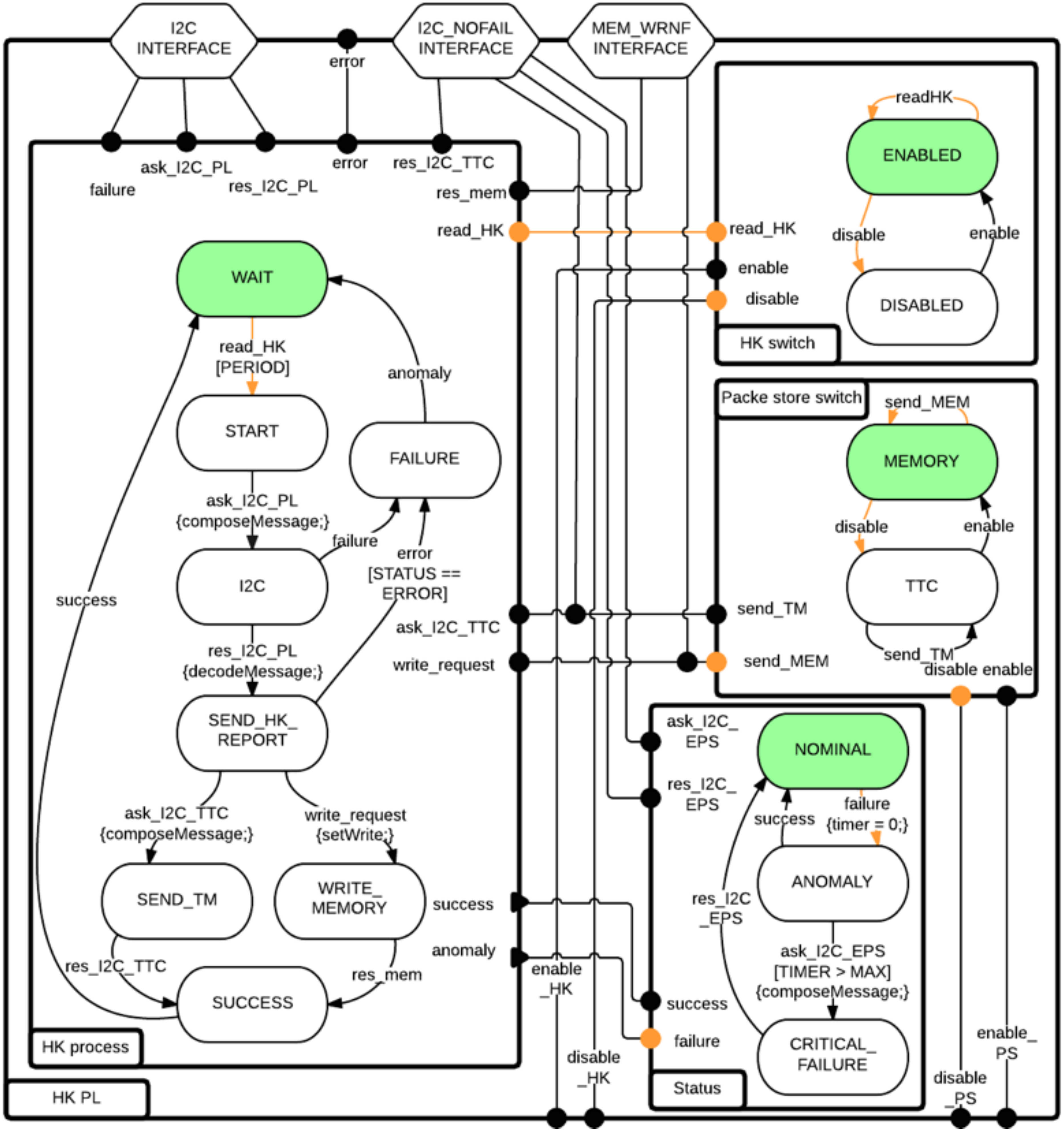
A series of semantics-preserving transformations

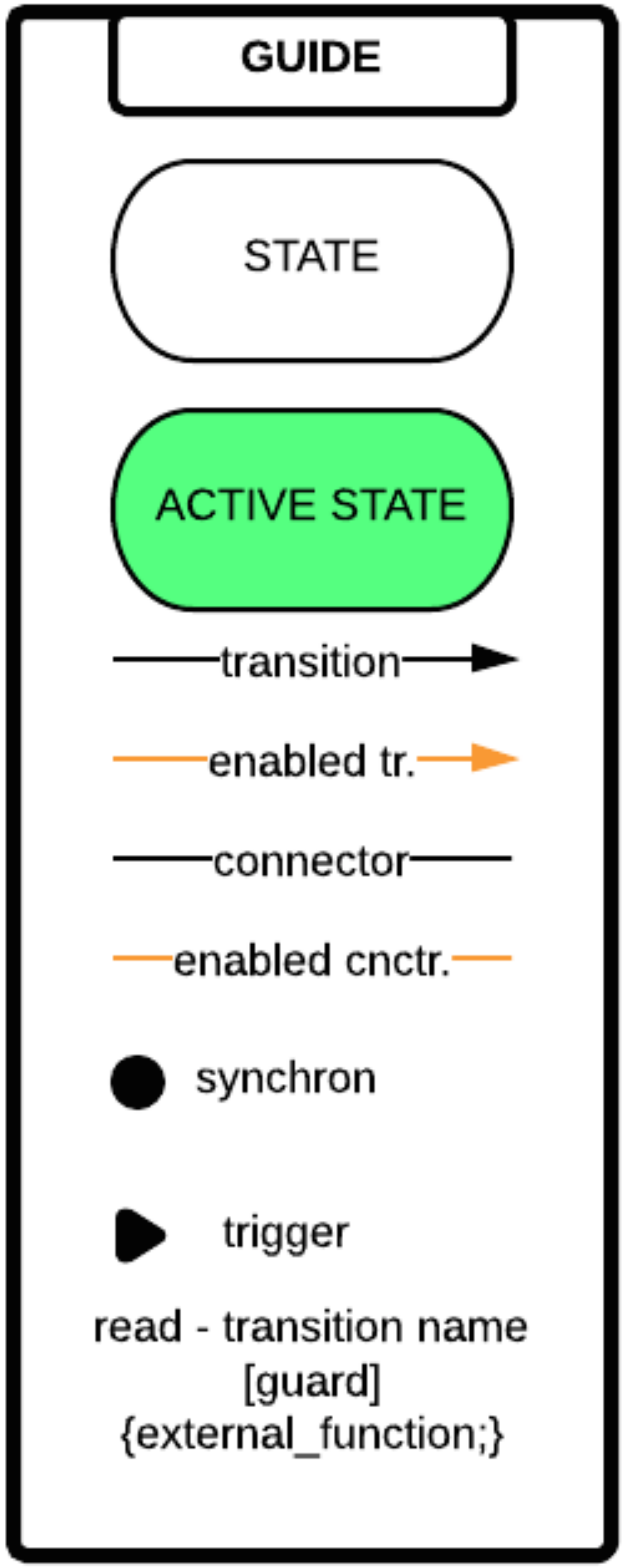
Correctness decomposed into

correctness of transformations
correctness of high-level models

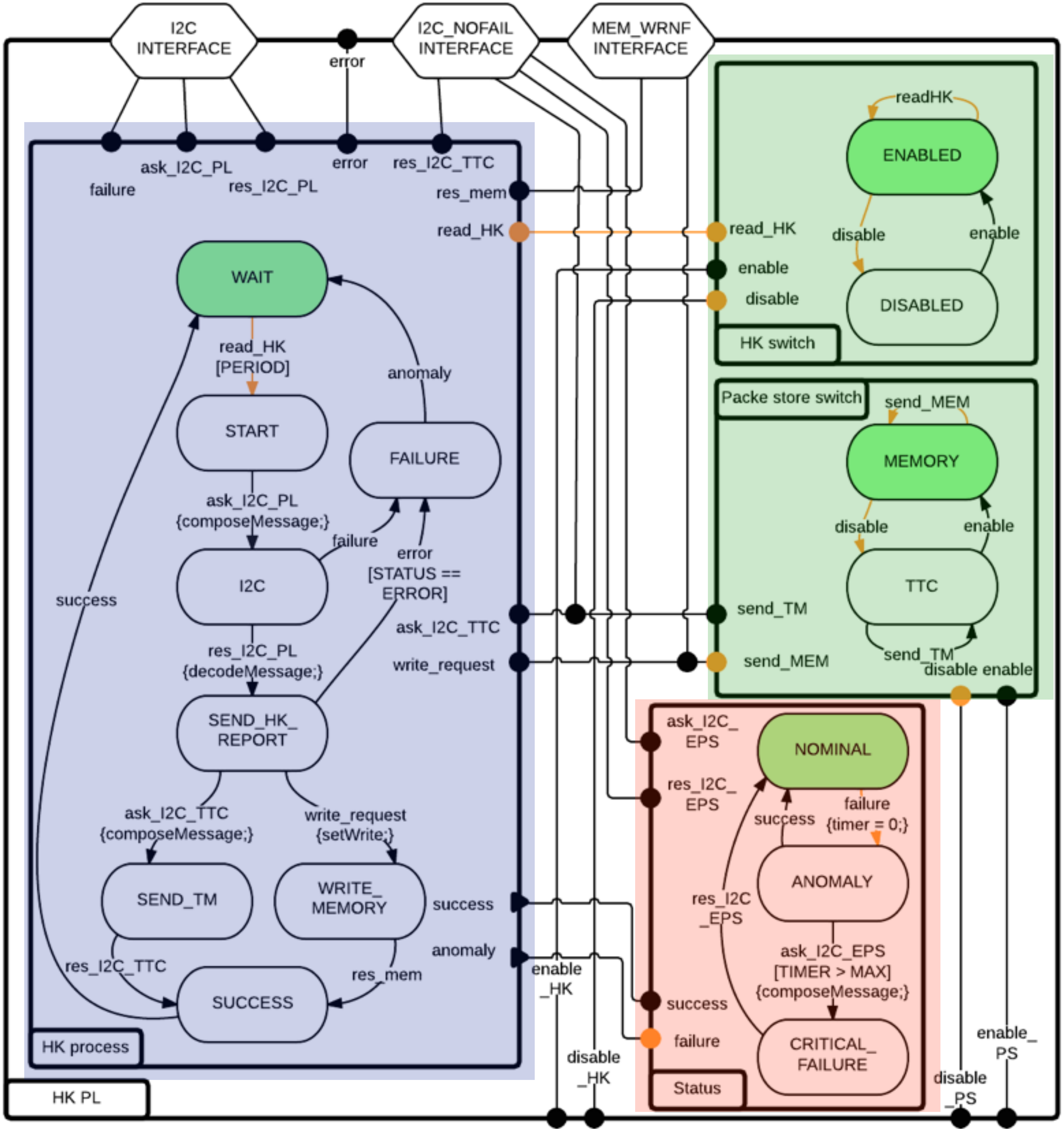
Final implementation is **correct by construction**

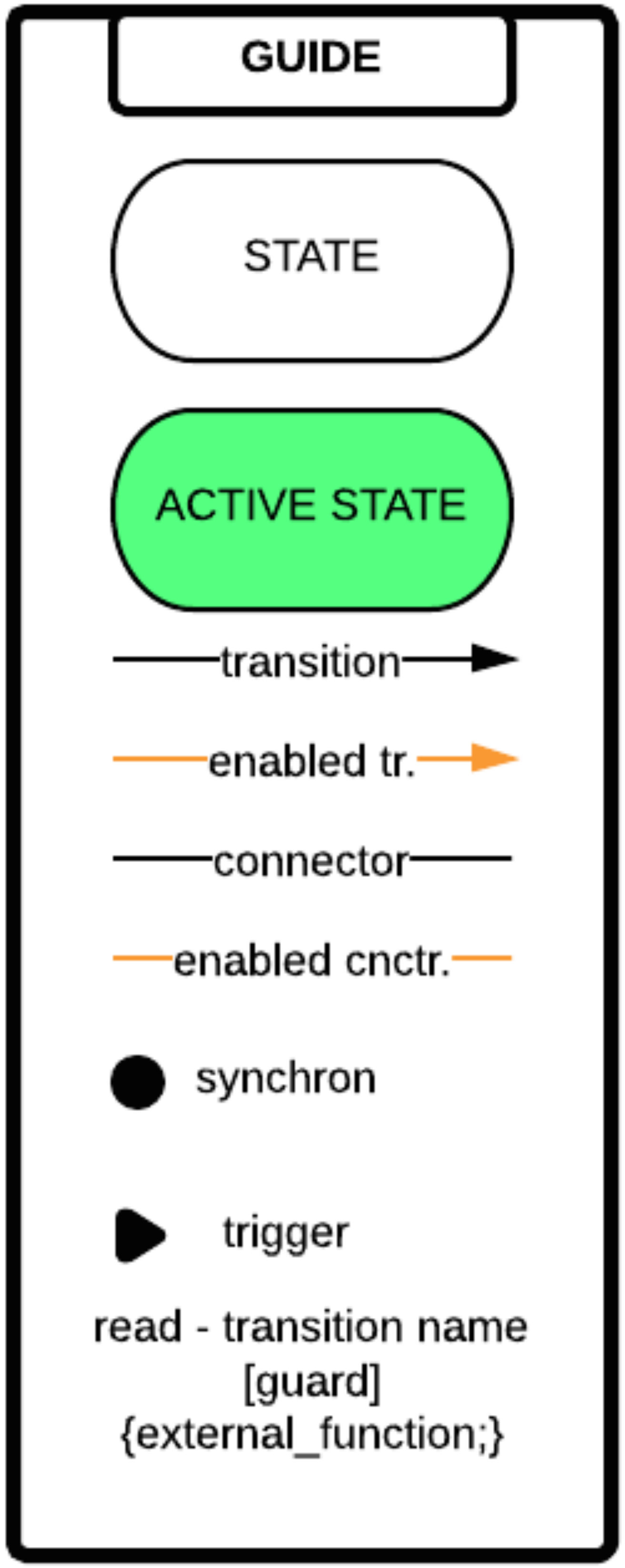
- □ Unifying modelling framework
- □ Operational semantics
- □ Method(s) to design correct models





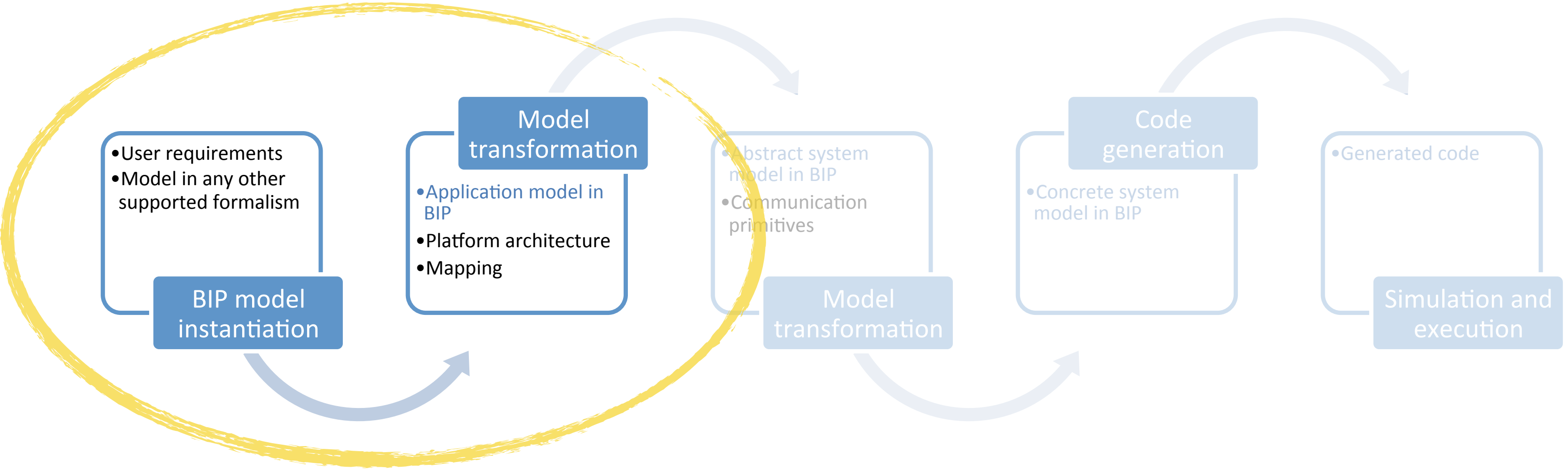
slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

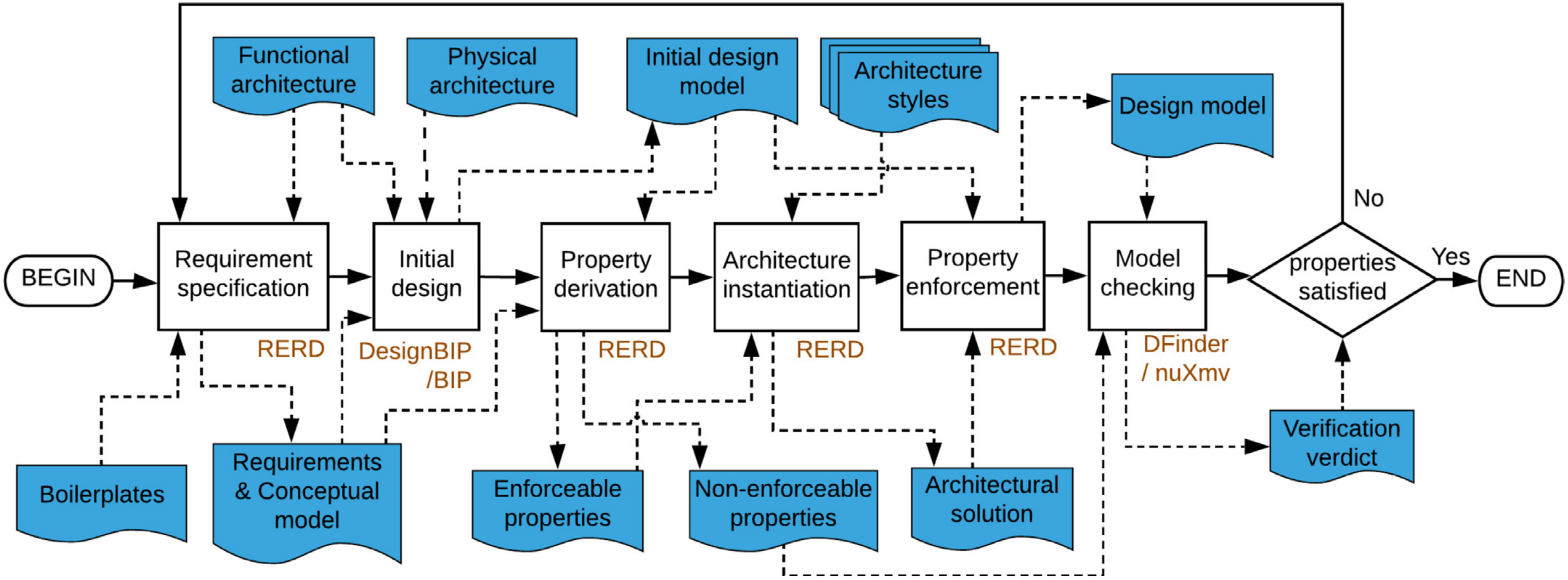
Requirements and design process



[Stachtiari et al, JSS '18]

ARISTOTLE UNIVERSITY OF THESSALONIKI

Requirements and design process



BEGIN

Requirement specification

Initial design

Property derivation

Architecture instantiation

Property enforcement

Model checking

properties satisfied

Yes

No

END

Functional architecture

Physical architecture

Initial design model

Architecture styles

Design model

Boilerplates

Requirements & Conceptual model

Enforceable properties

Non-enforceable properties

Architectural solution

Verification verdict

RERD

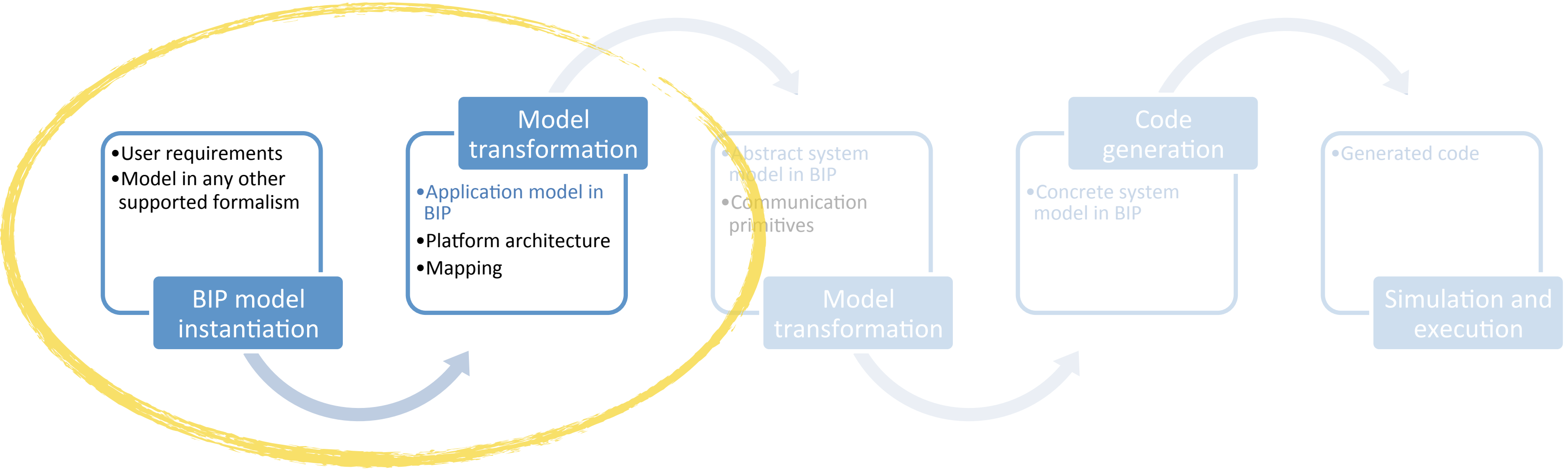
DesignBIP /BIP

RERD

RERD

RERD

DFinder / nuXmv



- User requirements
- Model in any other supported formalism

BIP model instantiation

Model transformation

- Application model in BIP
- Platform architecture
- Mapping

- Abstract system model in BIP
- Communication primitives

Model transformation

Code generation

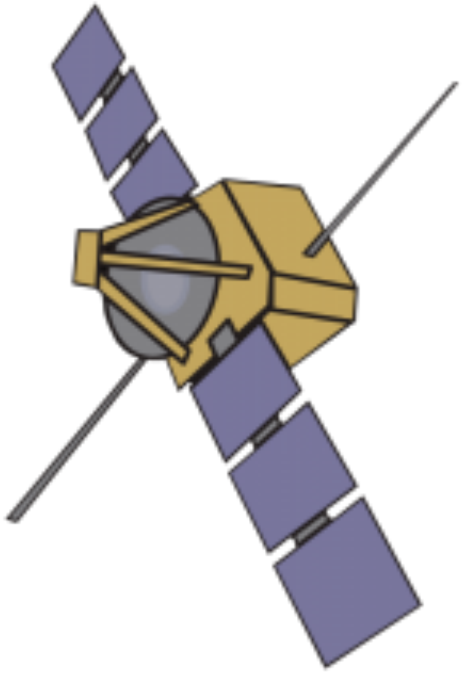
- Concrete system model in BIP

- Generated code

Simulation and execution

[Stachtiari et al, JSS '18]

CubETH case study



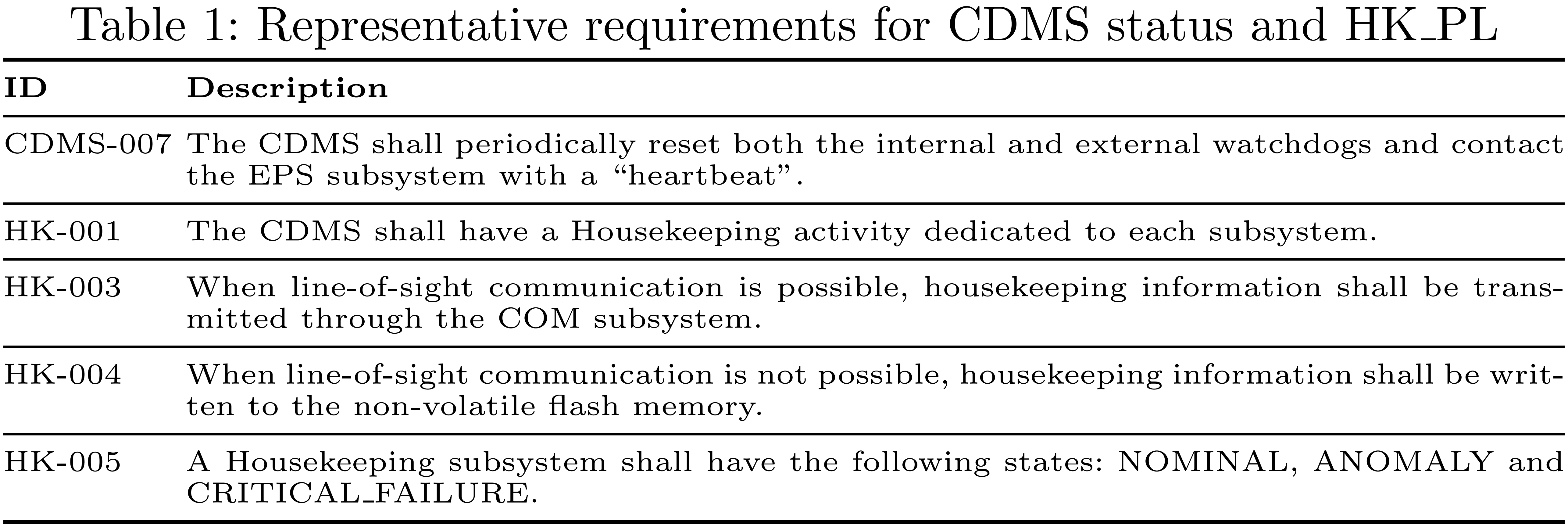
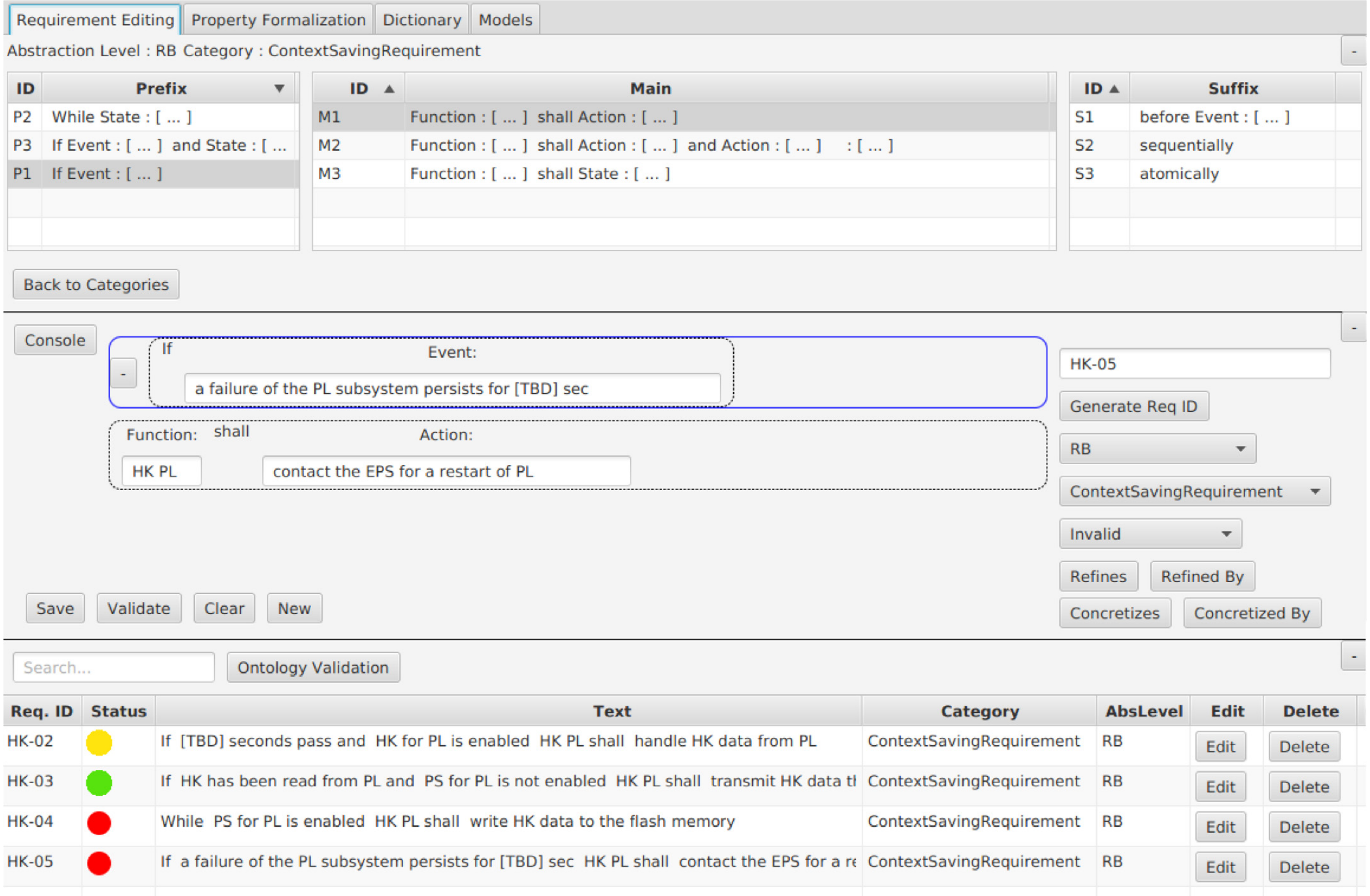


Table 1: Representative requirements for CDMS status and HK_PL

ID	**Description**
CDMS-007	The CDMS shall periodically reset both the internal and external watchdogs and contact the EPS subsystem with a “heartbeat”.
HK-001	The CDMS shall have a Housekeeping activity dedicated to each subsystem.
HK-003	When line-of-sight communication is possible, housekeeping information shall be transmitted through the COM subsystem.
HK-004	When line-of-sight communication is not possible, housekeeping information shall be written to the non-volatile flash memory.
HK-005	A Housekeeping subsystem shall have the following states: NOMINAL, ANOMALY and CRITICAL_FAILURE.

[Mavridou et al, FACS '16]

RERD tool



Abstraction Level : RB Category : ContextSavingRequirement

ID	Prefix ▼
P2	While State : [...]
P3	If Event : [...] and State : [...
P1	If Event : [...]

ID ▲	Main
M1	Function : [...] shall Action : [...]
M2	Function : [...] shall Action : [...] and Action : [...] : [...]
M3	Function : [...] shall State : [...]

ID ▲	Suffix
S1	before Event : [...]
S2	sequentially
S3	atomically

If Event: a failure of the PL subsystem persists for [TBD] sec

Function: HK PL shall Action: contact the EPS for a restart of PL

HK-05

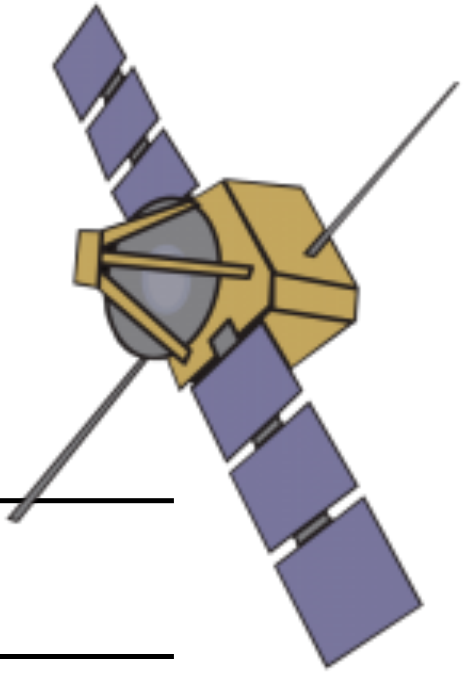
RB

ContextSavingRequirement

Invalid

Req. ID	Status	Text	Category	AbsLevel	Edit	Delete
HK-02	🟡	If [TBD] seconds pass and HK for PL is enabled HK PL shall handle HK data from PL	ContextSavingRequirement	RB	Edit	Delete
HK-03	🟢	If HK has been read from PL and PS for PL is not enabled HK PL shall transmit HK data tl	ContextSavingRequirement	RB	Edit	Delete
HK-04	🔴	While PS for PL is enabled HK PL shall write HK data to the flash memory	ContextSavingRequirement	RB	Edit	Delete
HK-05	🔴	If a failure of the PL subsystem persists for [TBD] sec HK PL shall contact the EPS for a re	ContextSavingRequirement	RB	Edit	Delete

CubETH case study

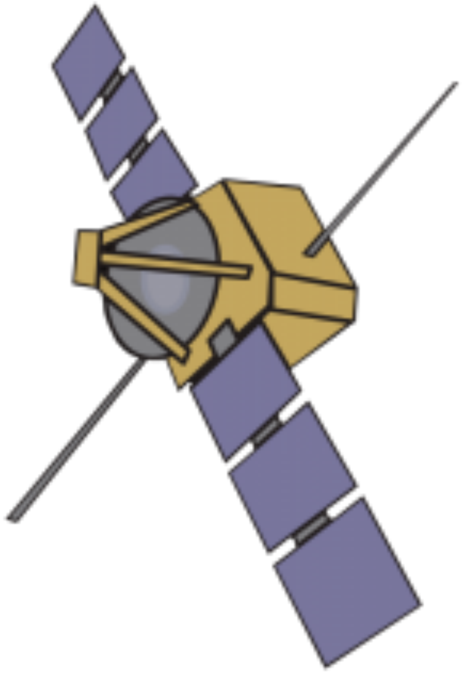


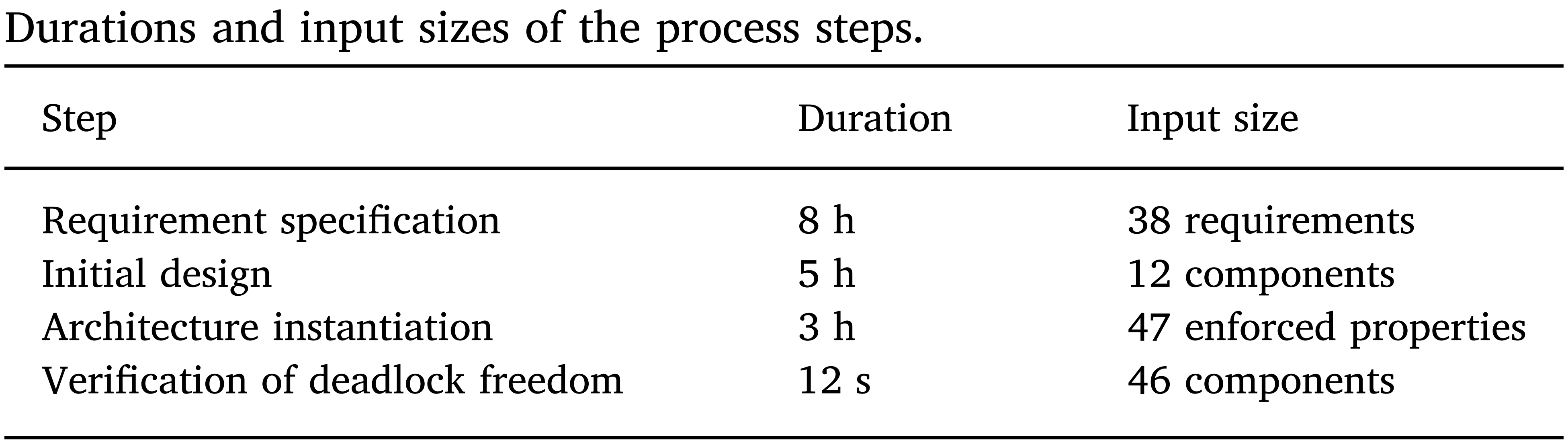
Requirements for the *HK PL* function.

ID	Requirement
HK-02	P2: if ⟨event-e003: [TBD] sec pass ⟩ and ⟨state-s003: HK collection is enabled for PL ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a004: handle HK data from the PL ⟩
HK-03	P3: if ⟨state-s002: PS[a] for PL is not enabled ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a002: transmit HK data through the TC/TM service ⟩
HK-04	P3: while ⟨state-s001: PS for PL is enabled ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a001: write HK data to the flash memory ⟩
HK-05	P1: if ⟨event-e004: a PL failure persists for [TBD] sec ⟩ M1: ⟨function: HK PL ⟩ shall ⟨action-a003: contact the EPS for a restart of the PL ⟩

[a] PS stands for a packet store structure.

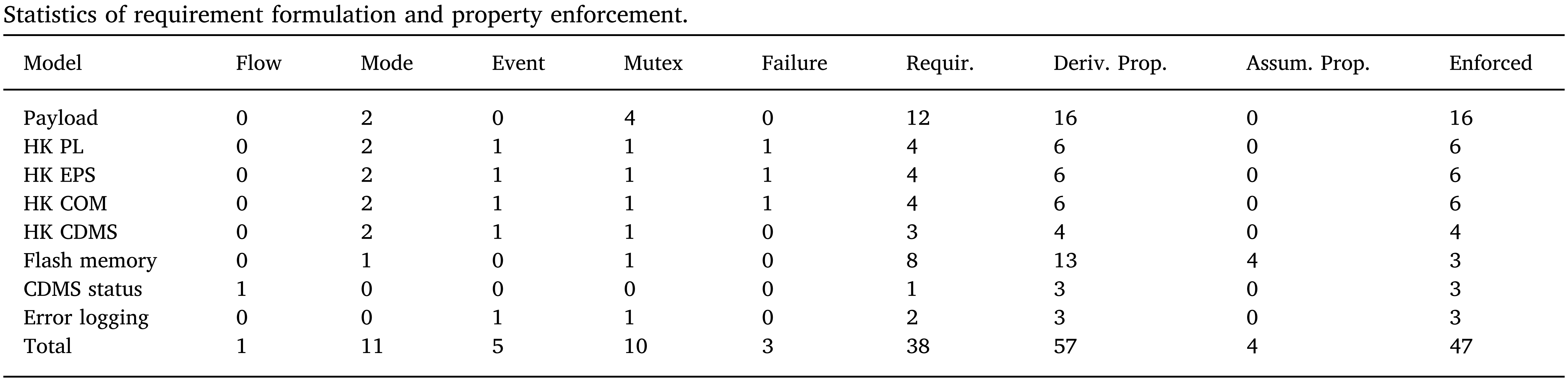
CubETH case study





Durations and input sizes of the process steps.

Step	Duration	Input size
Requirement specification	8 h	38 requirements
Initial design	5 h	12 components
Architecture instantiation	3 h	47 enforced properties
Verification of deadlock freedom	12 s	46 components



Statistics of requirement formulation and property enforcement.

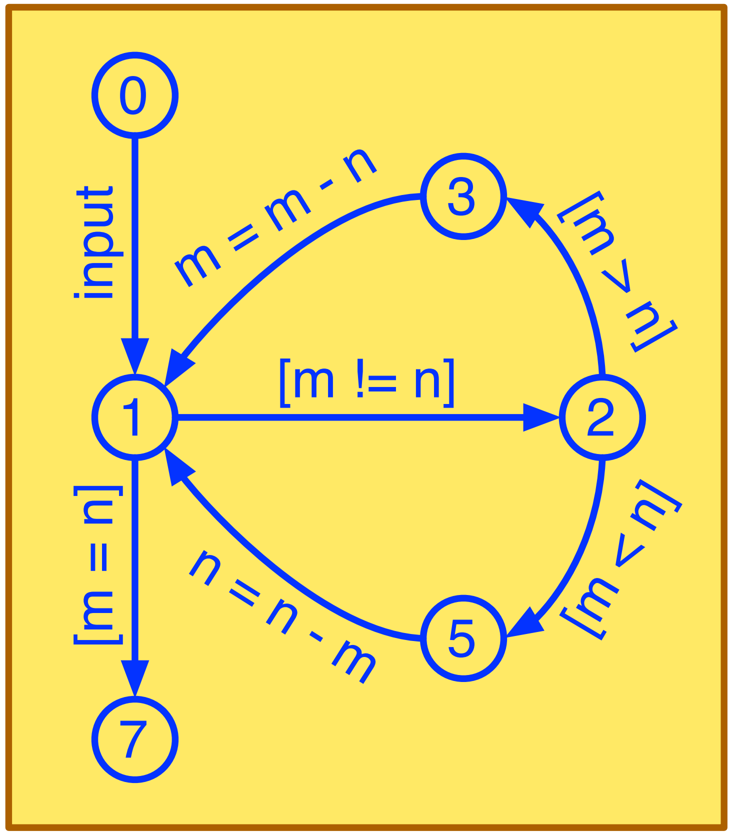
Model	Flow	Mode	Event	Mutex	Failure	Requir.	Deriv. Prop.	Assum. Prop.	Enforced
Payload	0	2	0	4	0	12	16	0	16
HK PL	0	2	1	1	1	4	6	0	6
HK EPS	0	2	1	1	1	4	6	0	6
HK COM	0	2	1	1	1	4	6	0	6
HK CDMS	0	2	1	1	0	3	4	0	4
Flash memory	0	1	0	1	0	8	13	4	3
CDMS status	1	0	0	0	0	1	3	0	3
Error logging	0	0	1	1	0	2	3	0	3
Total	1	11	5	10	3	38	57	4	47

se•man•tic (si man'...
to meaning or arising from ...
symbols. semantic change; semantic...
semantics. [1655–65; < Gk sêman...
mant(ós) marked (sêman-, base o...
verbal adj. suffix; akin to sêma si...

se•man•tics (si man'tiks), n. (u...
linguistics dealing with the stu...
meaning is structured in langu...
over time. 2. the branch of se...
tionship between signs or sy...
meaning, or an interpretation...
mence, etc.: Let's not argue...
lence. 1895–1900] —se•man'ti•c...

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)
```

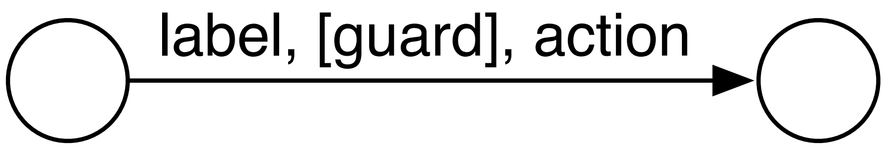


Purely sequential programs

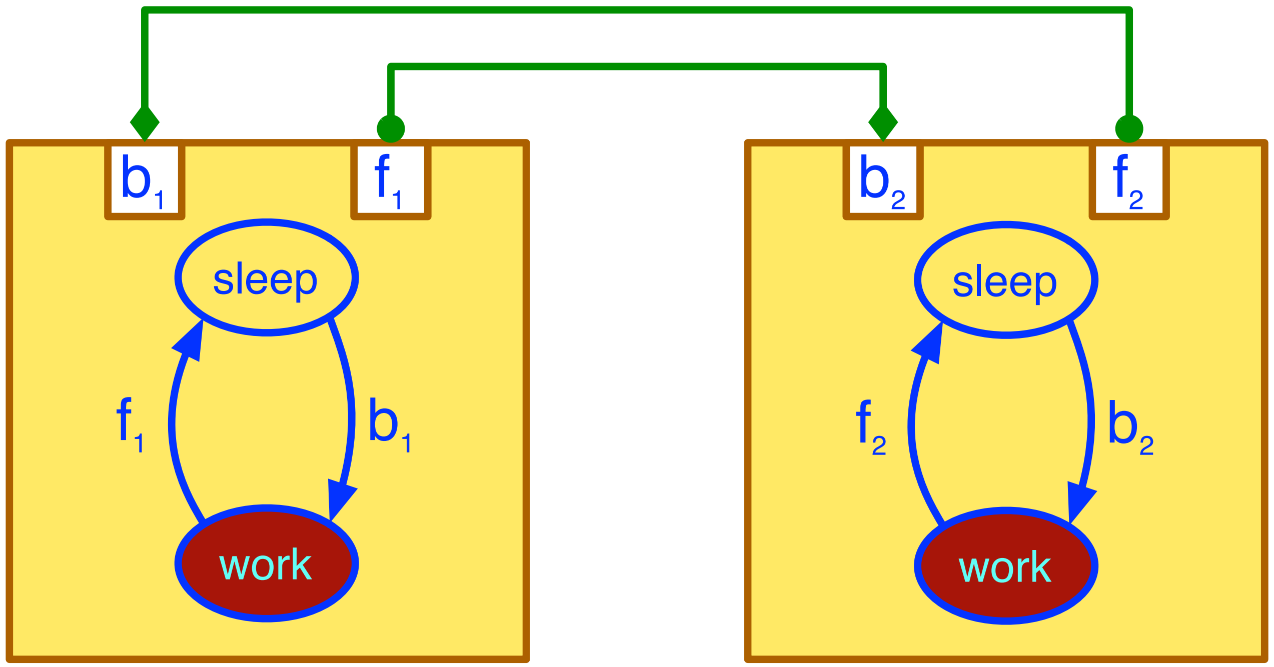
The choice of abstraction level is important

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:

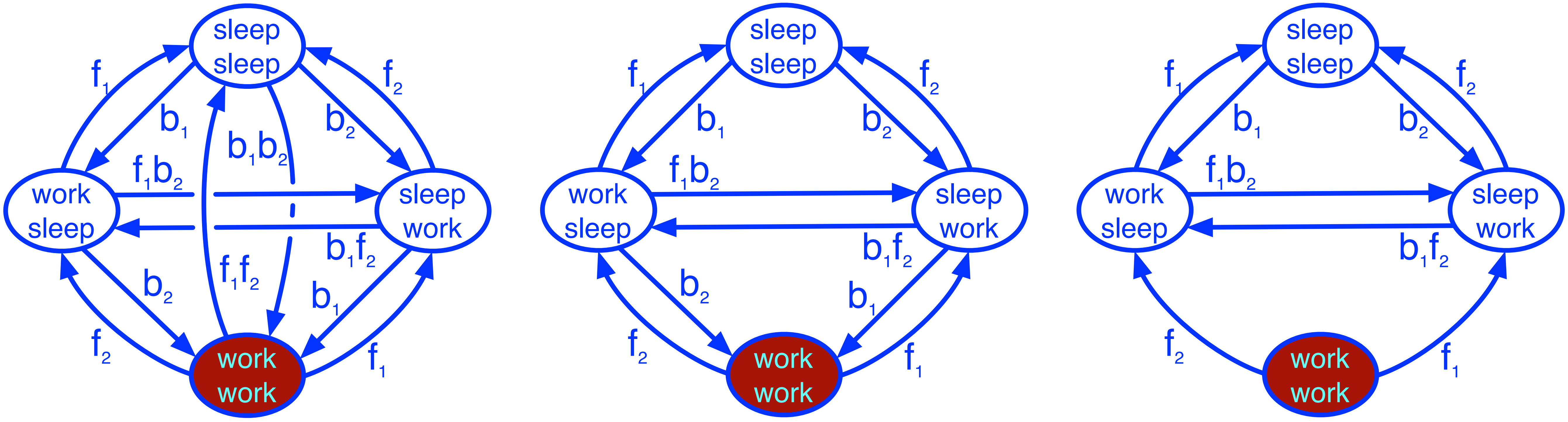
$$\{b_1,\ f_1,\ b_2,\ f_2,\ b_1 f_2,\ b_2 f_1\}$$

Maximal progress:

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

Design view

Semantic view



Semantics: Interactions

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

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

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

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

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

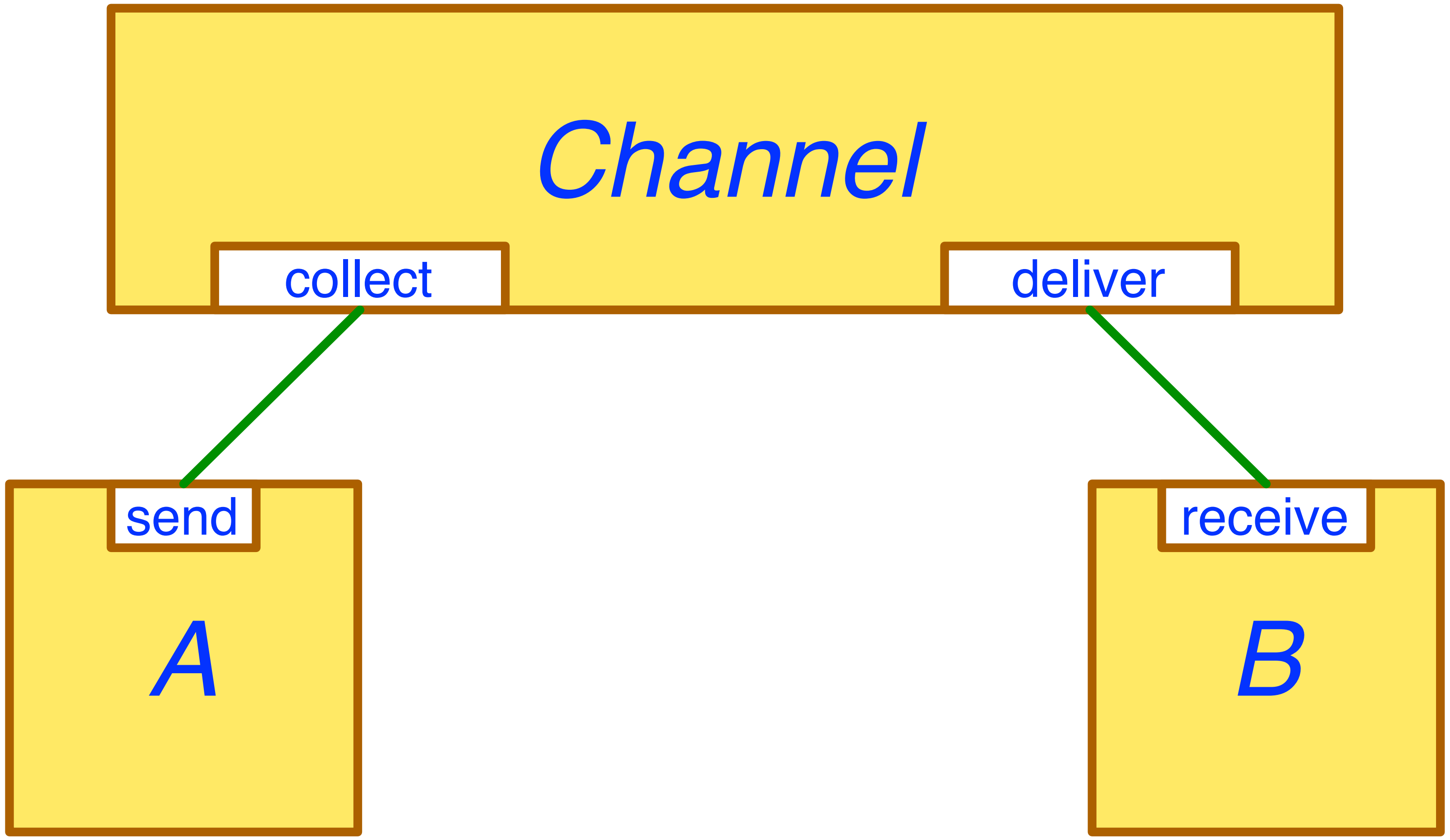
for each $a \in \gamma$.

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 $a \in 2^P$.

Unbuffered synchronous communication



A sends a message to *B*:

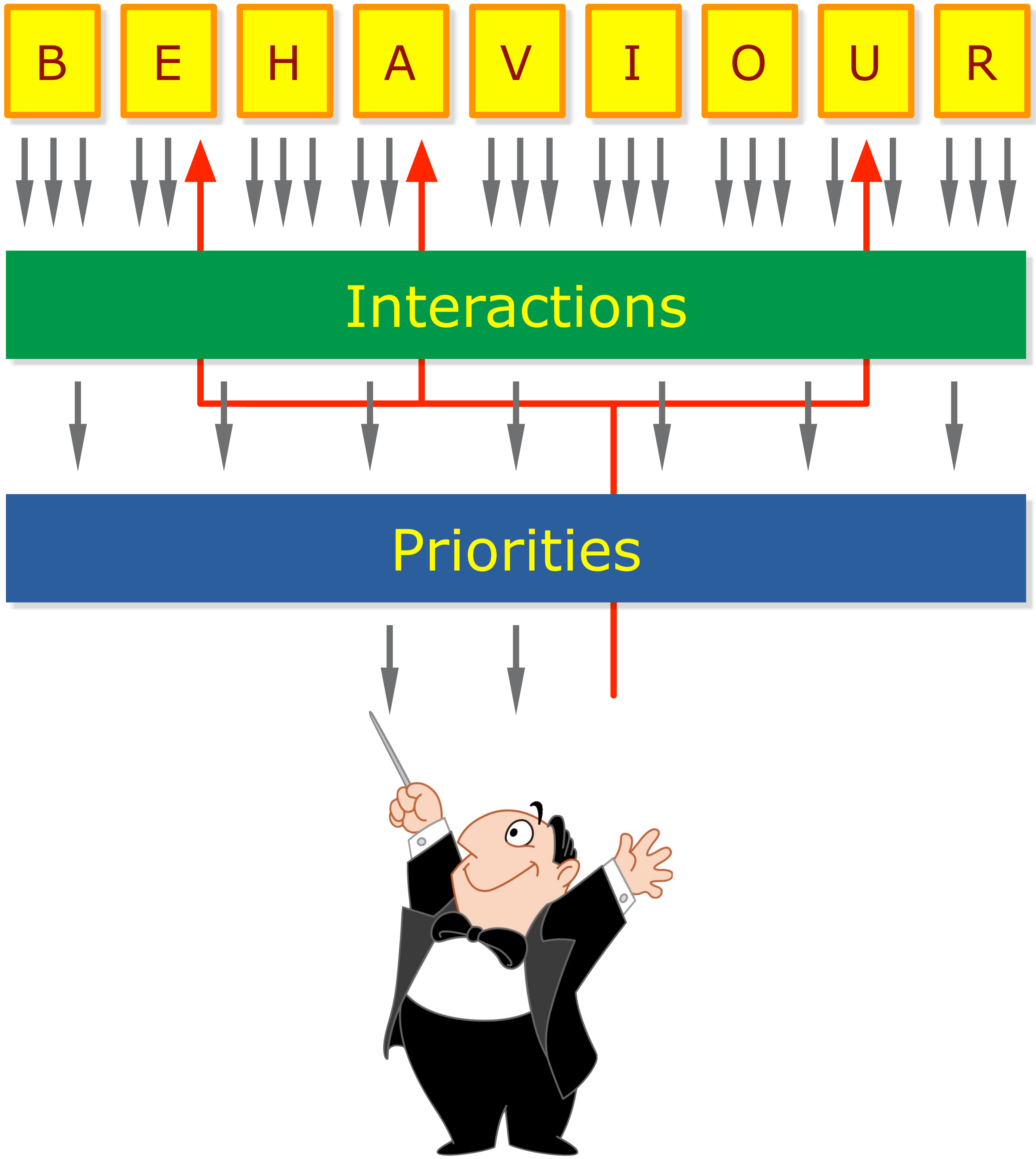
- Two synchronisations with the channel
- An explicit model of the channel behaviour

Not to be confused with synchronous *execution*

Reference implementation

Two-phase protocol:

1. Components notify the Engine about enabled transitions. ↓
2. The Engine picks an interaction and instructs the components. ↑

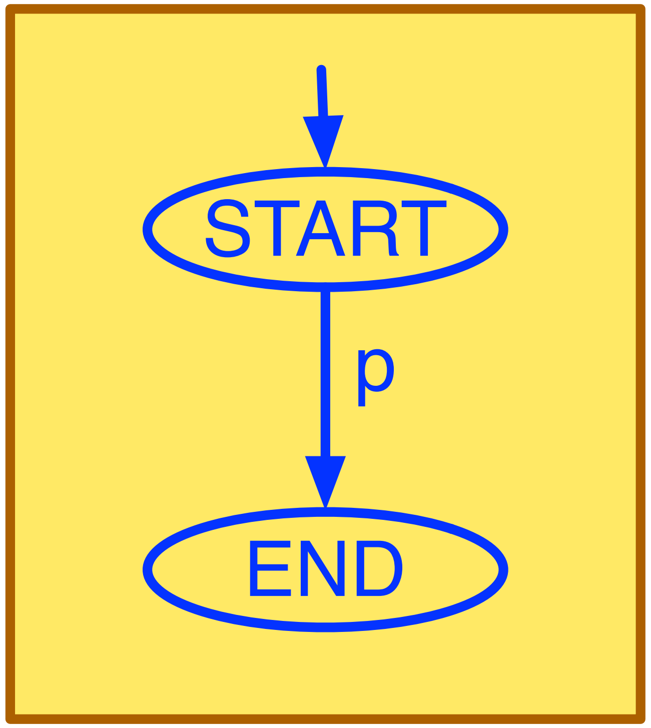


The BIP language

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
```



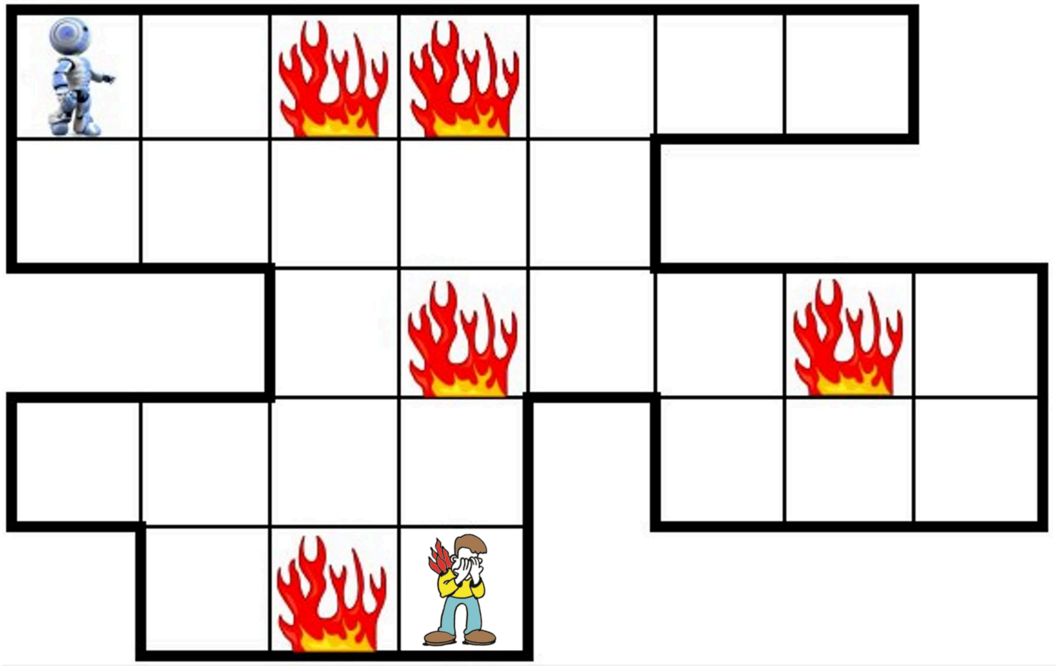
Hello World

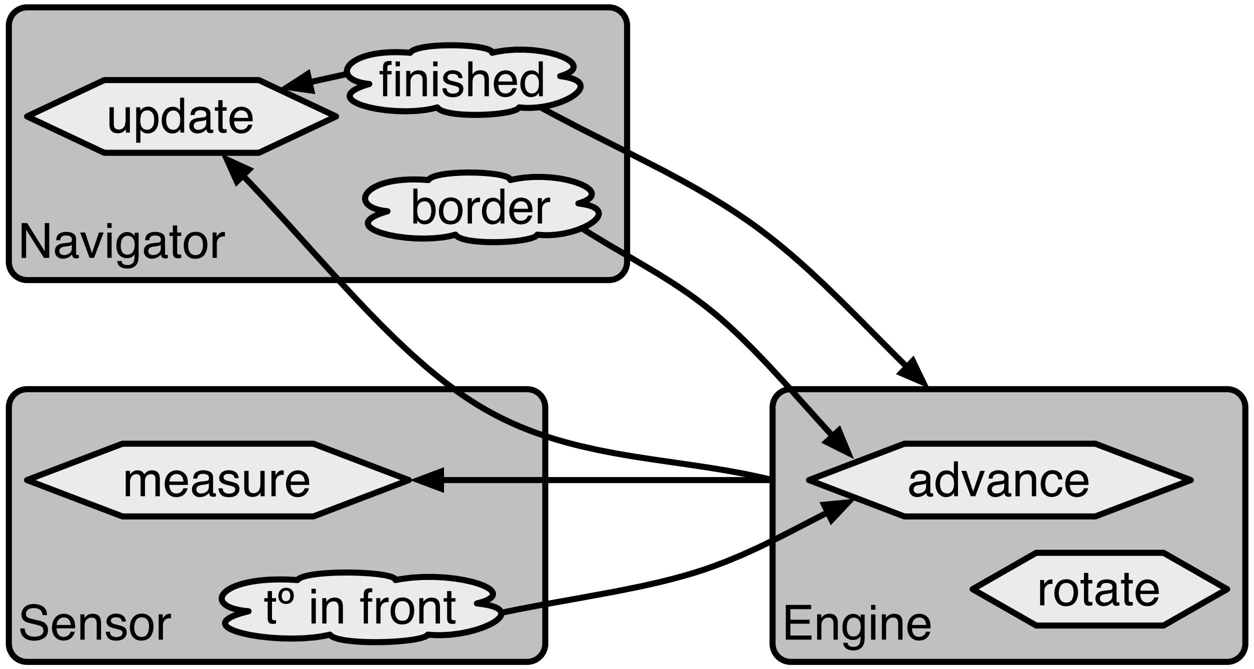
```
$ bipc.sh -I . -p HelloPackage -d "HelloCompound()" \
     --gencpp-output output
$ cd build
$ cmake ../output
$ make
$ ./build/system
```

```
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 2023.
[BIP ENGINE]:
[BIP ENGINE]: initialize components...
[BIP ENGINE]: random scheduling based o
[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

- Shall not advance and rotate at the same time
- Shall stay within the region
- Shall stay in the area that is safe or hot (but not burning)
- Shall update navigation and sensor data at each move
- When objective is found, the robot shall stop

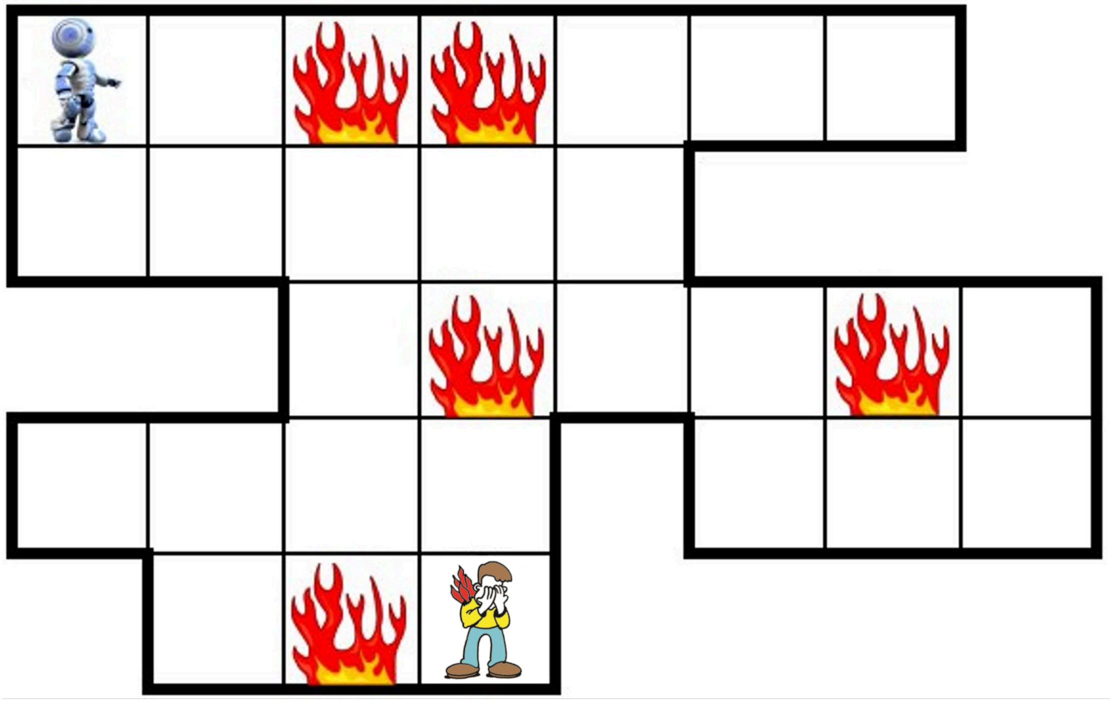
Rough plan

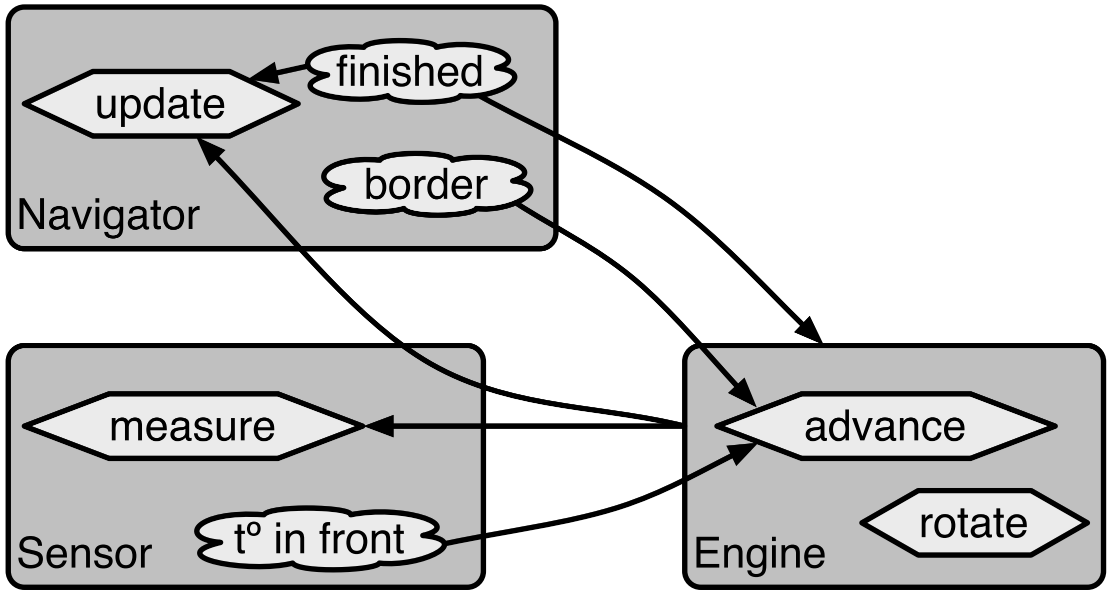
One square

N × N field (with N = 2, 5)

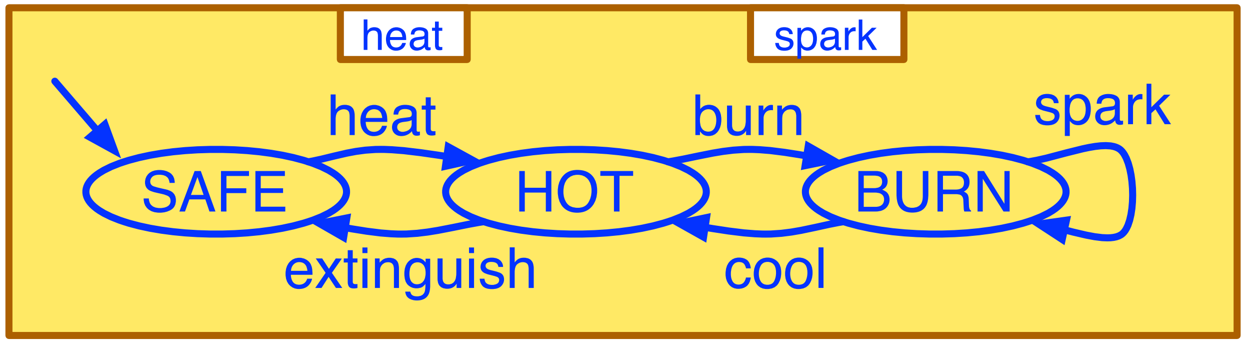
Complete with the robot

Remove the field





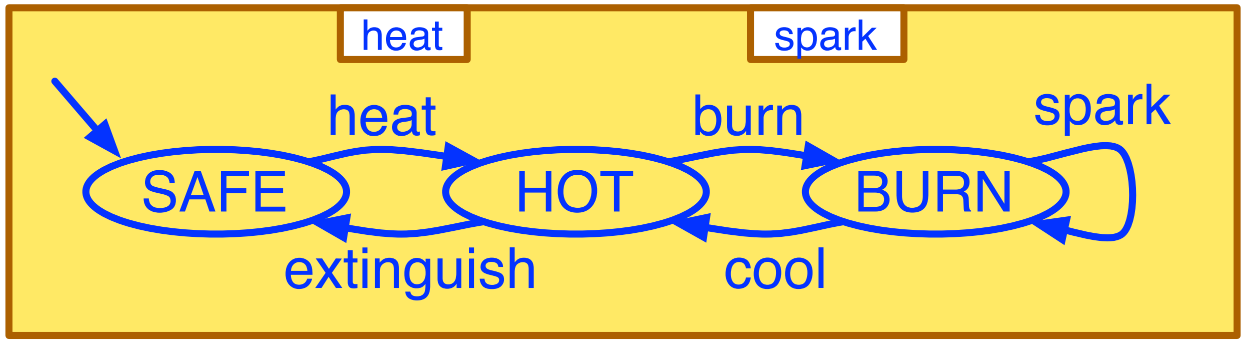
Atoms, ports and places



```
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
    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
```

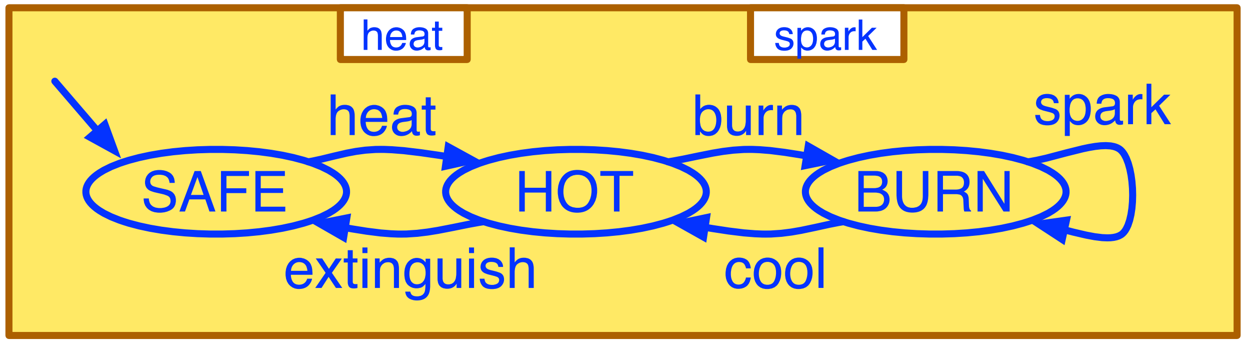
Atoms, ports and places



```
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
```

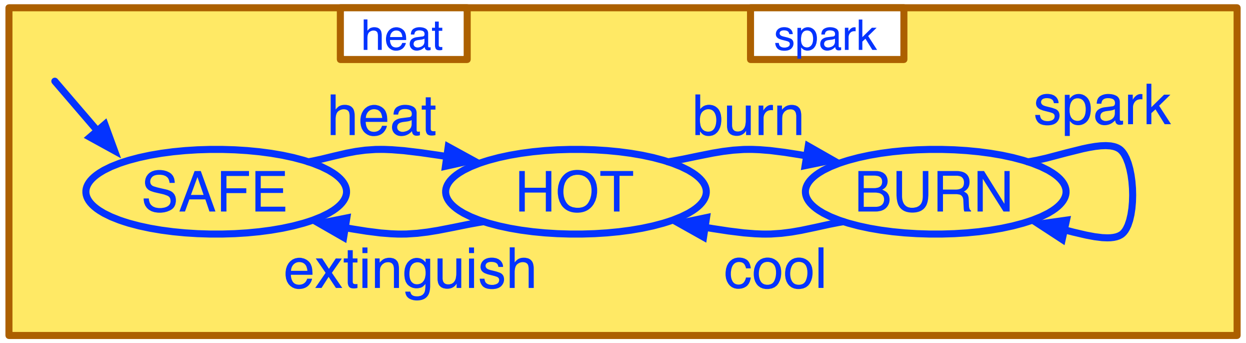
Atoms, ports and places



```
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
```

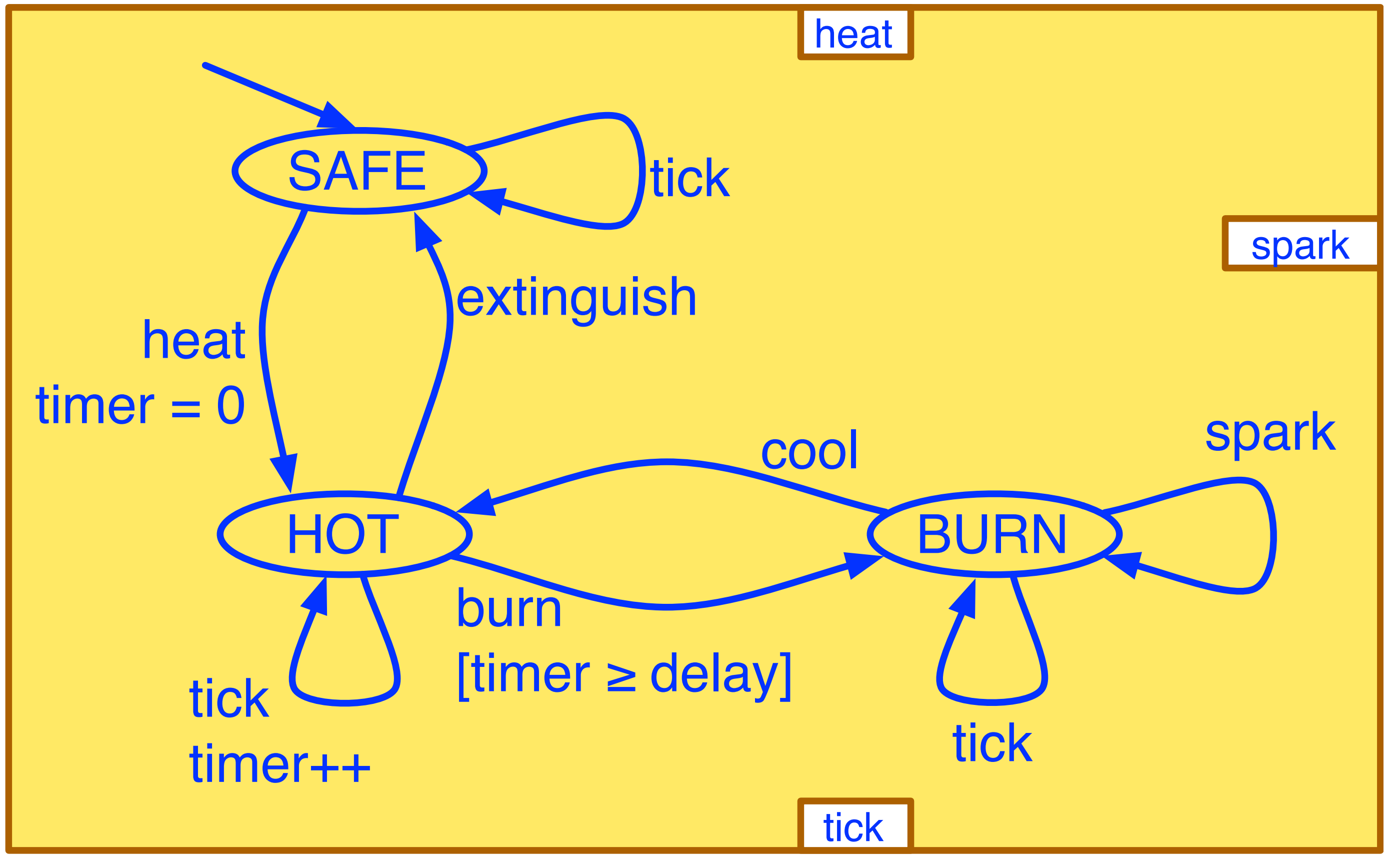
```
  connector type Singleton (Port_t p)
    define p
  end

  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



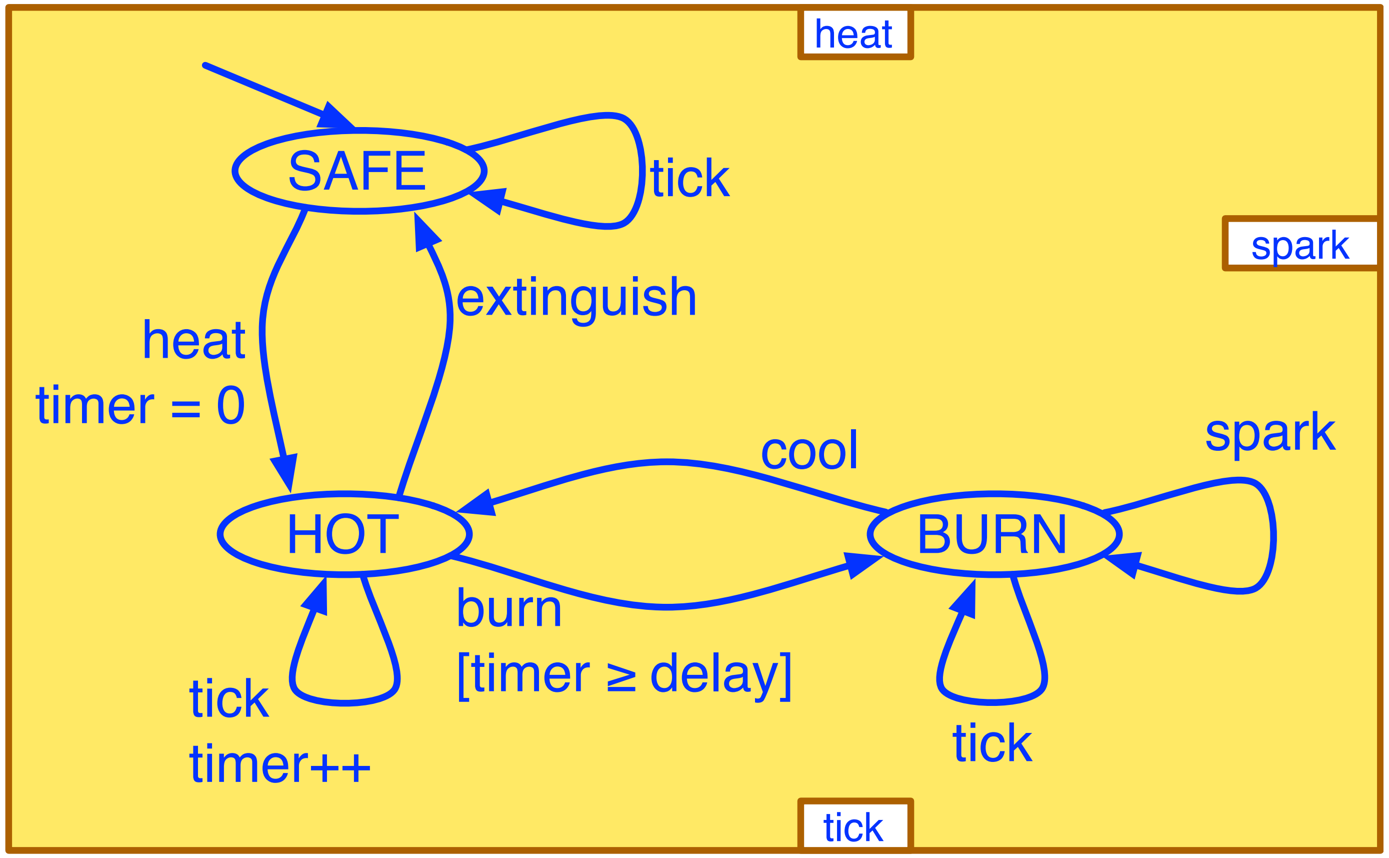
```
package RescueRobot
  port type Port_t()

  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



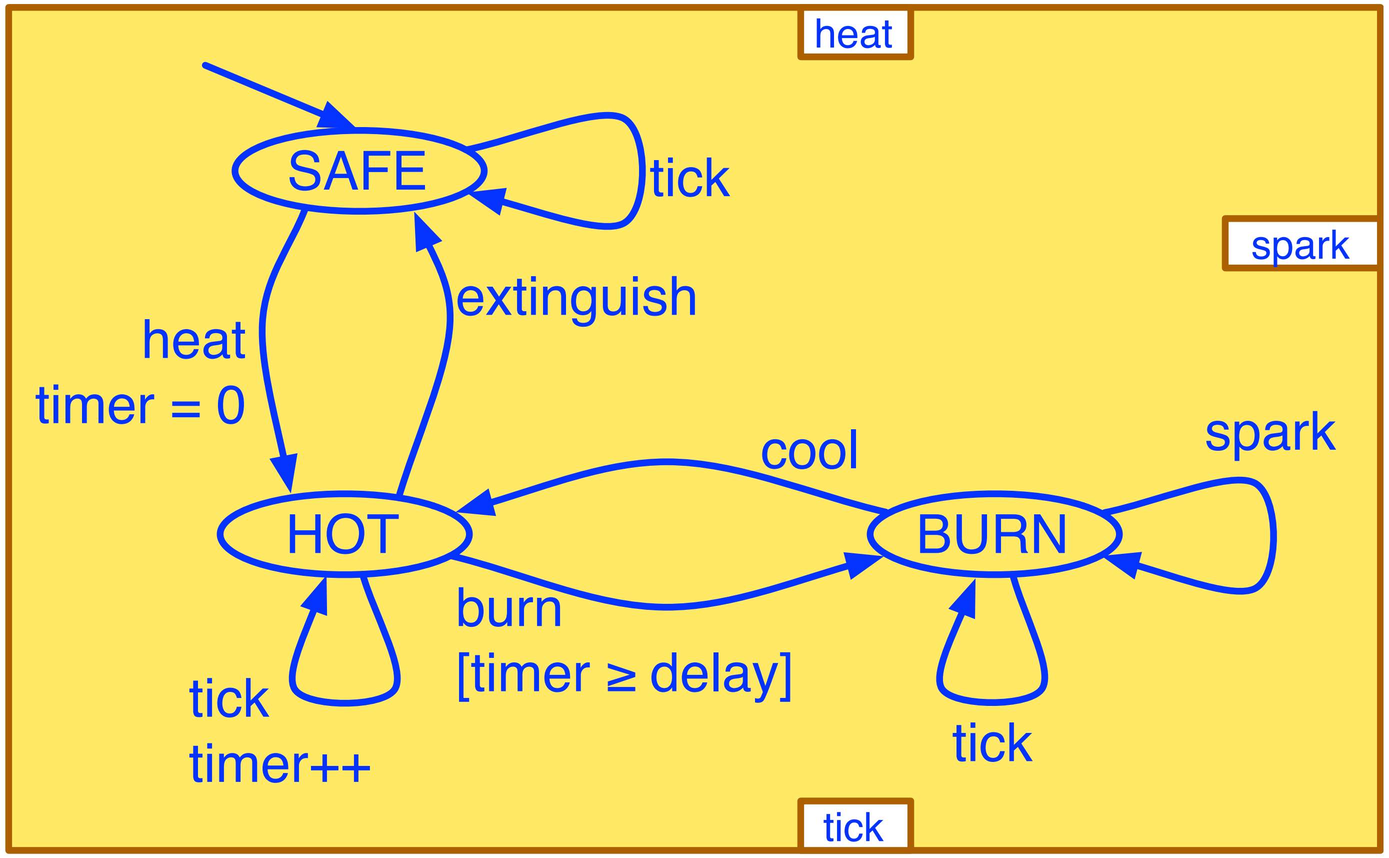
```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

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

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

  on cool from BURNING to HOT
    do {timer = 0;}
  <...>

  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



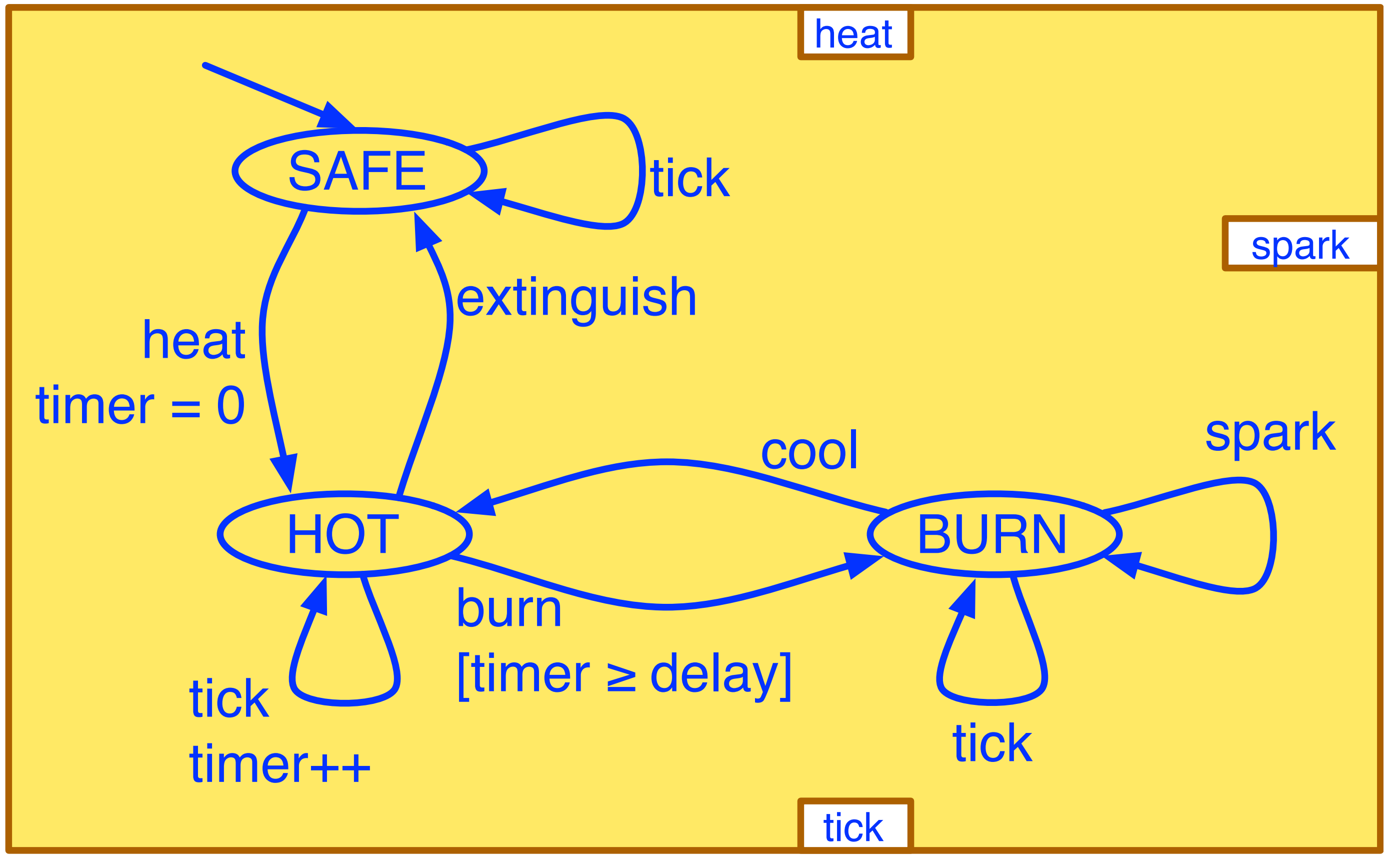
```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

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

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

  on cool from BURNING to HOT
    do {timer = 0;}
  <...>

  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



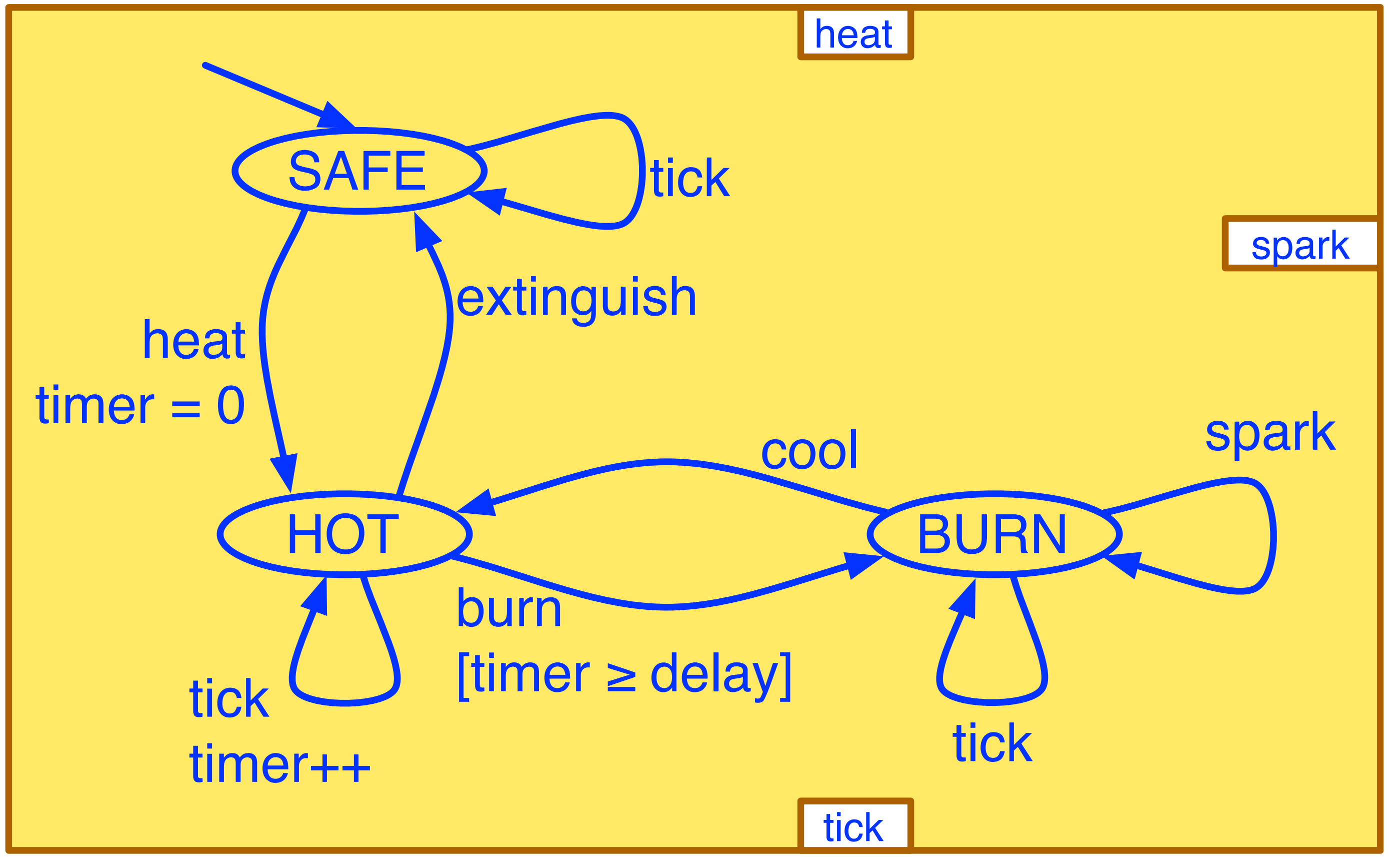
```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

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

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

  on cool from BURNING to HOT
    do {timer = 0;}
  <...>

  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



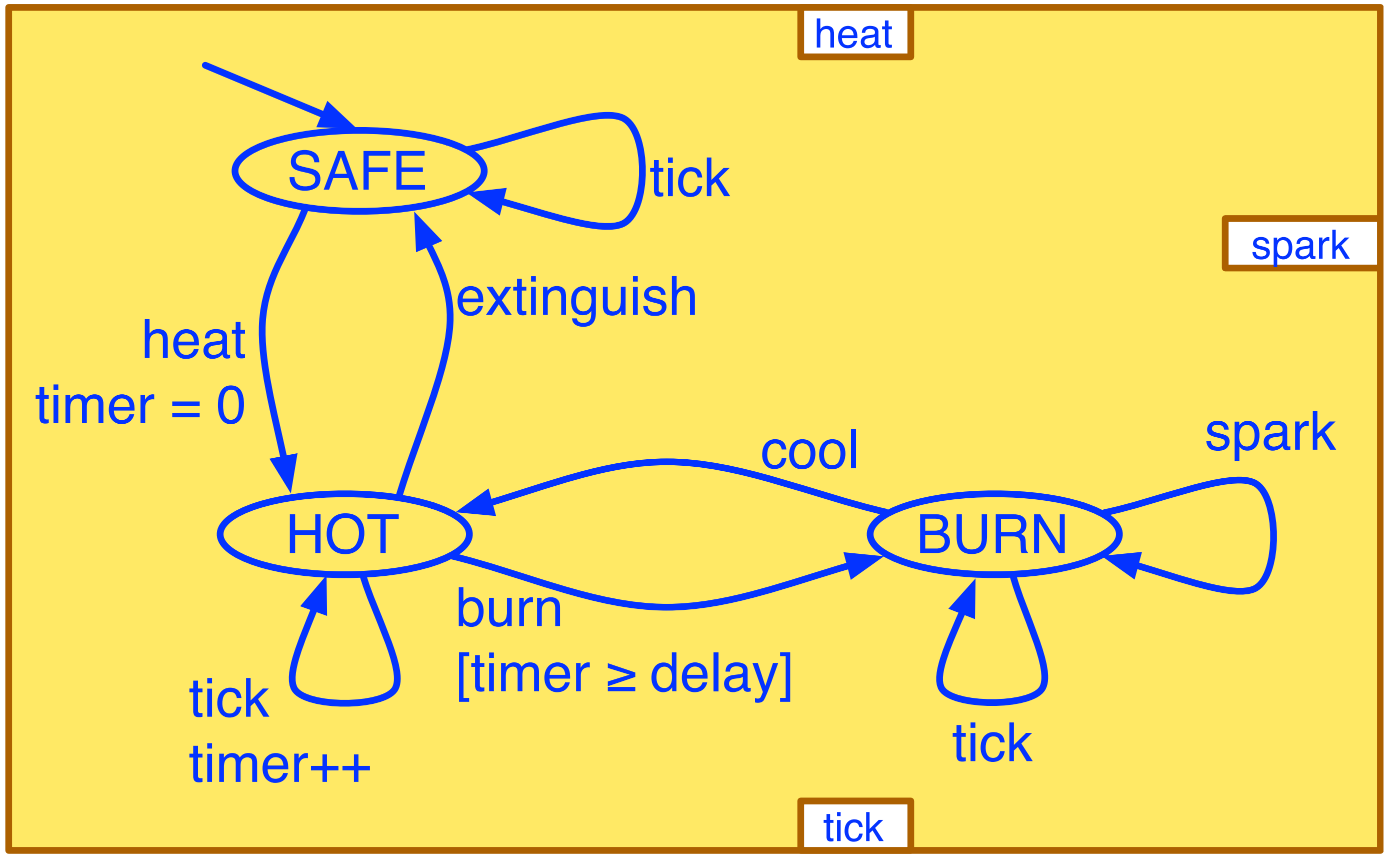
```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
  on burn from HOT to BURNING
    provided (timer >= delay)

  on cool from BURNING to HOT
    do {timer = 0;}
  <...>

  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



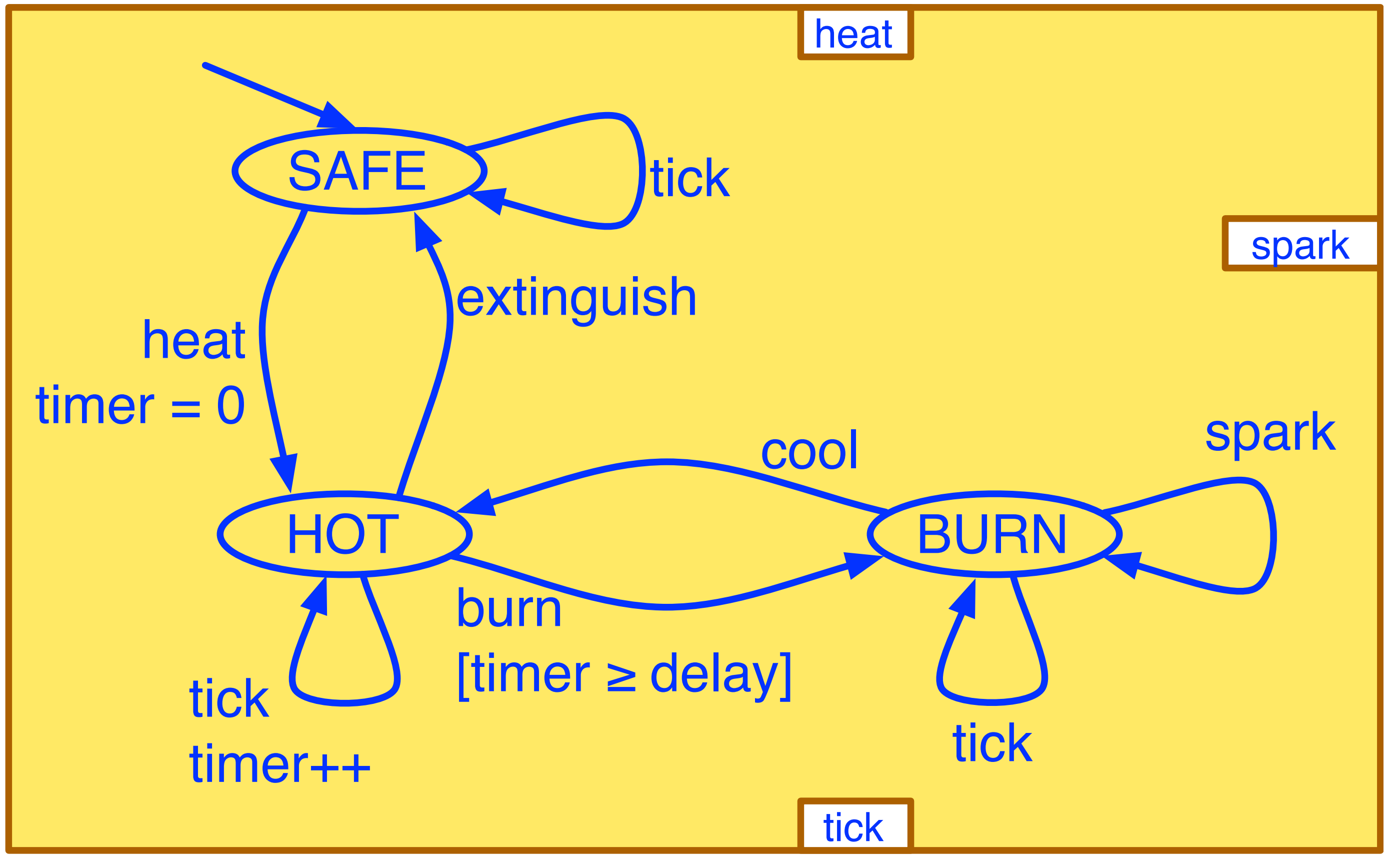
```
atom type Square (int delay)
  data int timer

  export port Port_t tick()

  <...>
  on heat from SAFE to HOT
    do {timer = 0;}
  on burn from HOT to BURNING
    provided (timer >= delay)

  on cool from BURNING to HOT
    do {timer = 0;}
  <...>

  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 cool from BURNING to HOT
    do {timer = 0;}
  <...>

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

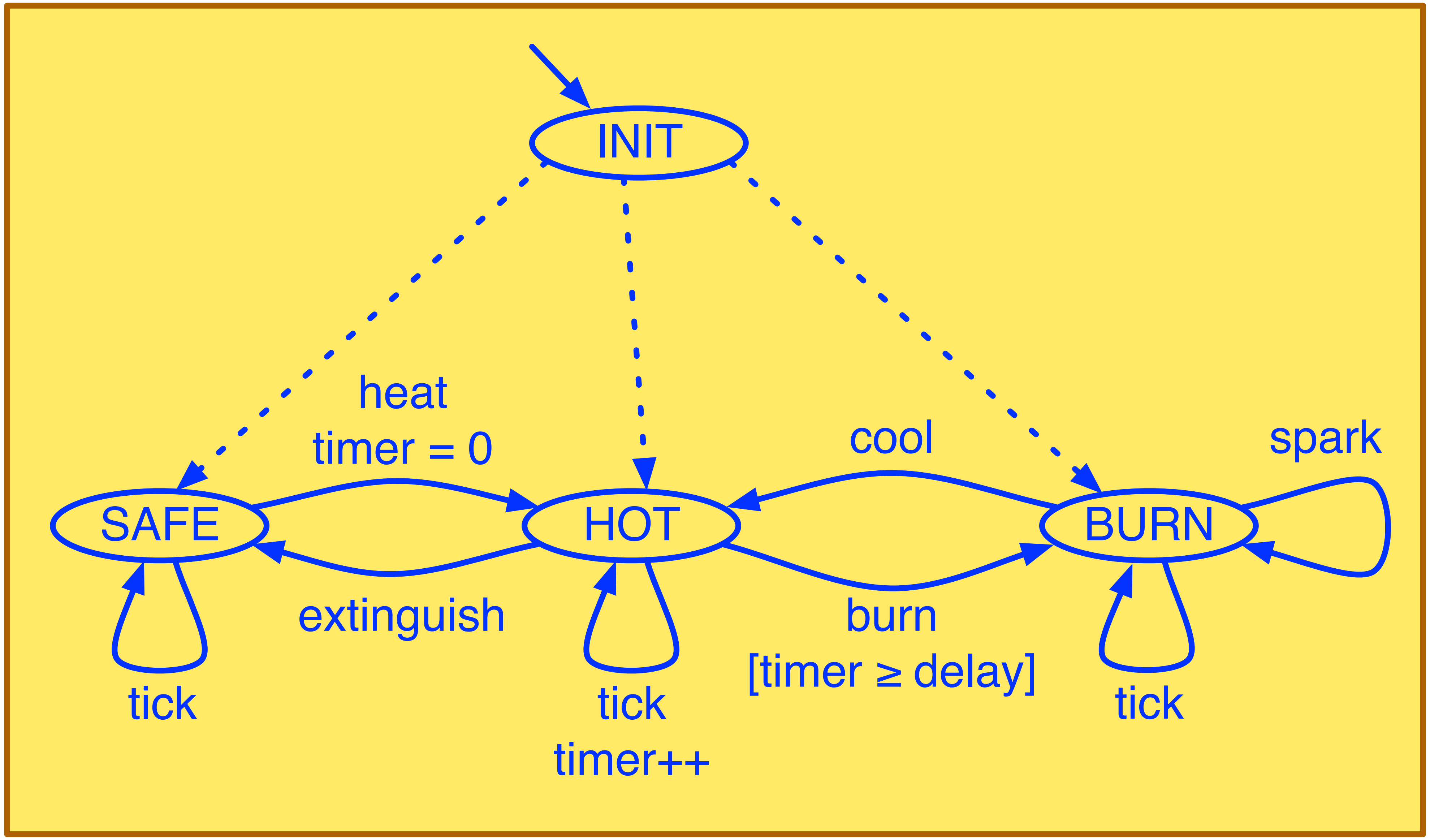
Data, guards and actions



```
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 cool from BURNING to HOT
    do {timer = 0;}
  <...>

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

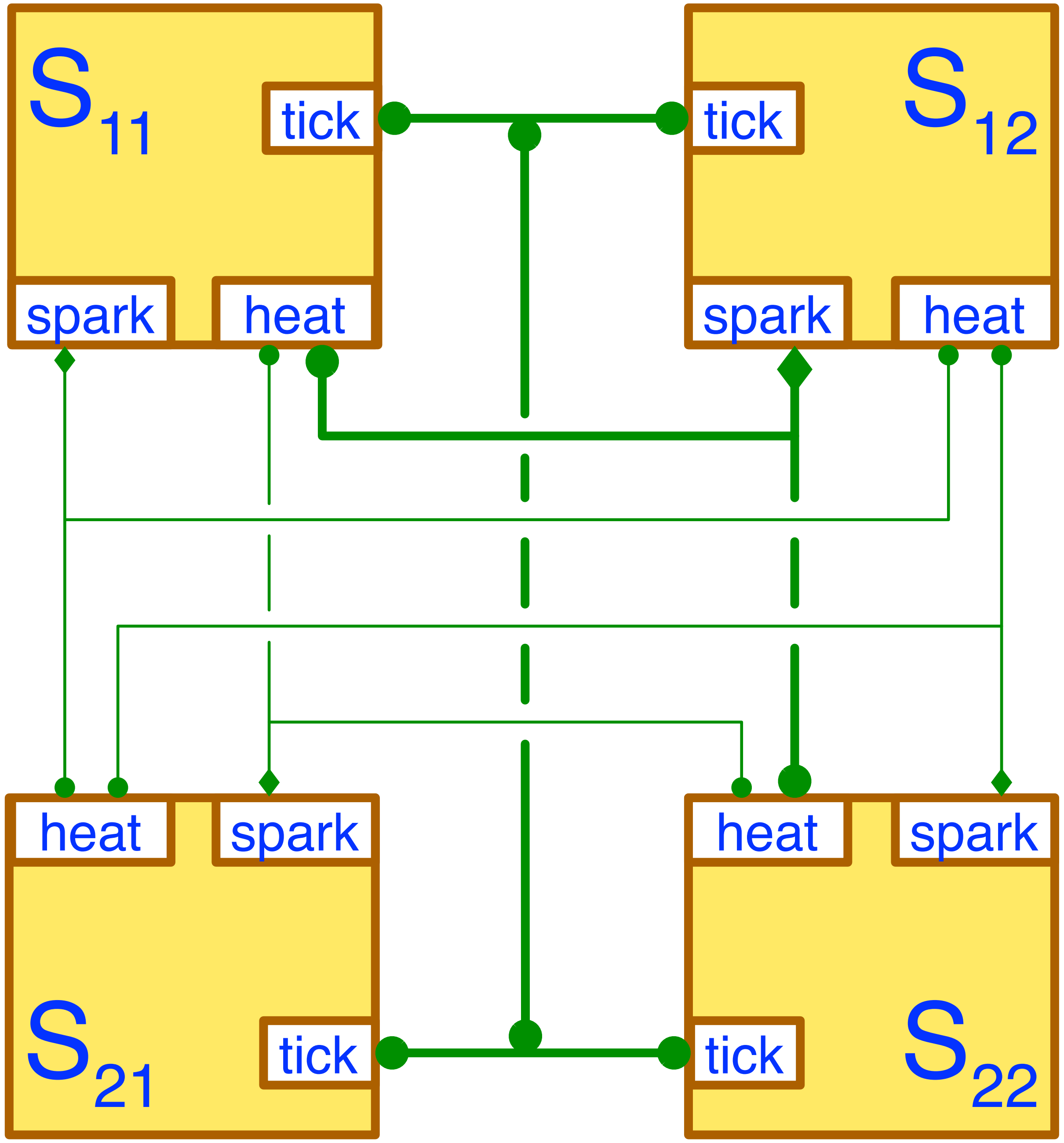
1. Add volatility
2. Add initial temperature

Internal transitions



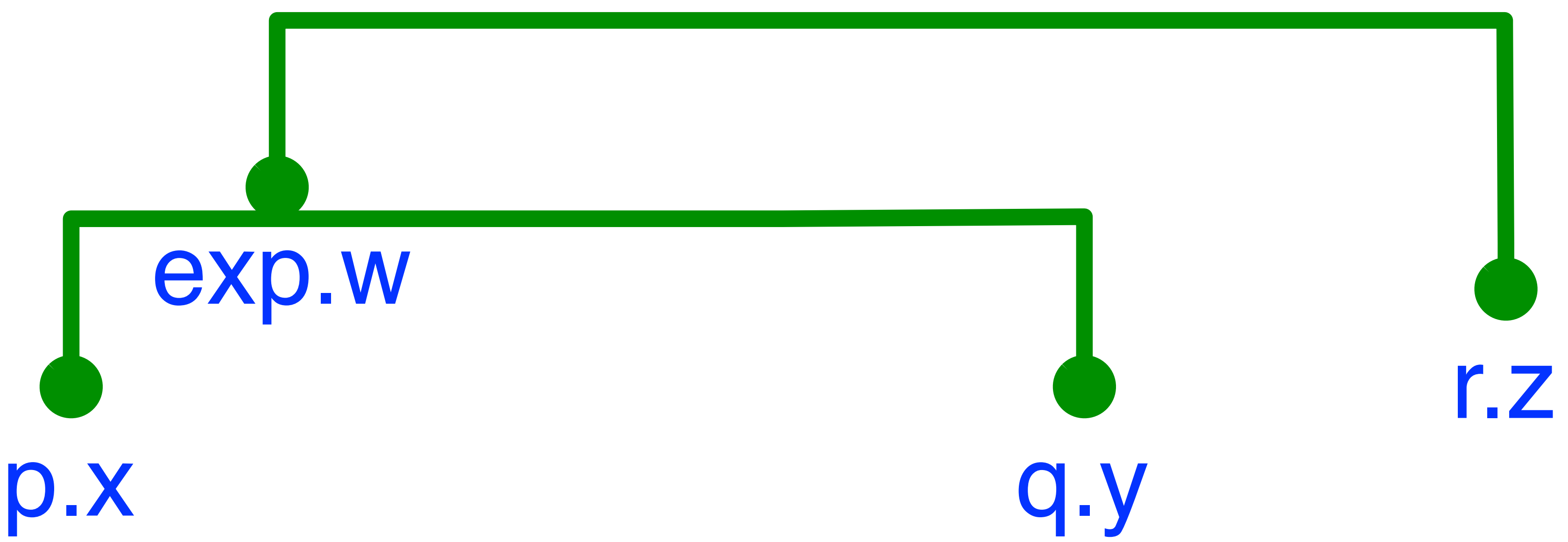
```
internal from INIT to ...
```

Connectors



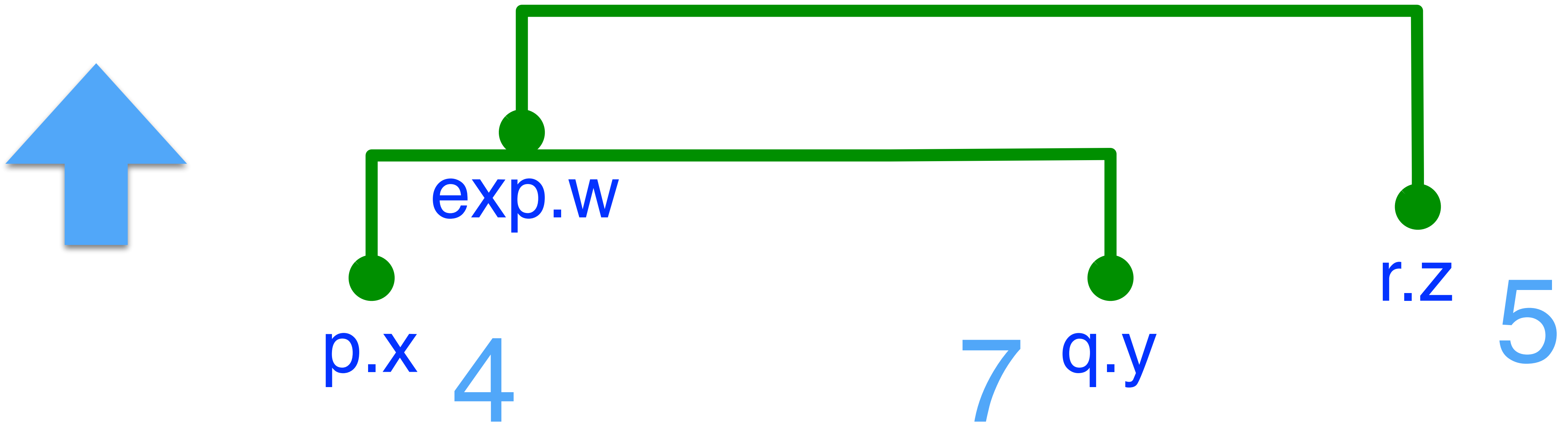
```
connector type Synchron2 (
        Port_t p, Port_t
    )
    export port Port_t sync_port()
    define p q
end

connector type Trigger2 (
        Port_t p, Port_t q, Port_t r
    )
    define p' q r
end

<...>

connector Synchron2 c_tick1 (
        square11.tick, square12.tick
)
connector Synchron2 c_tick2 (
        square21.tick, square22.tick
)

connector Synchron2 c_tick (
  c_tick1.sync_port, c_tick2.sync_port
)
```

Data transfer



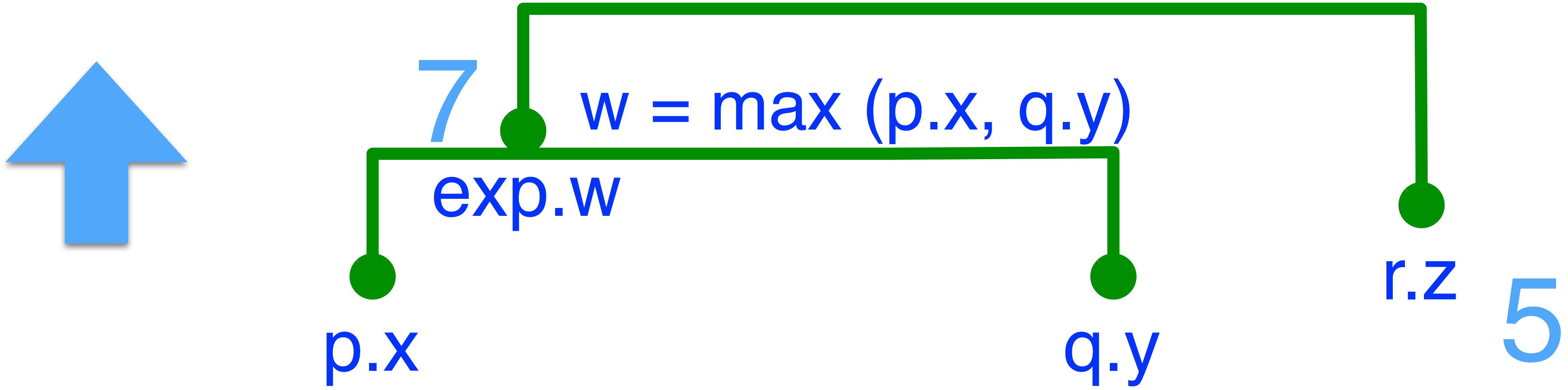
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



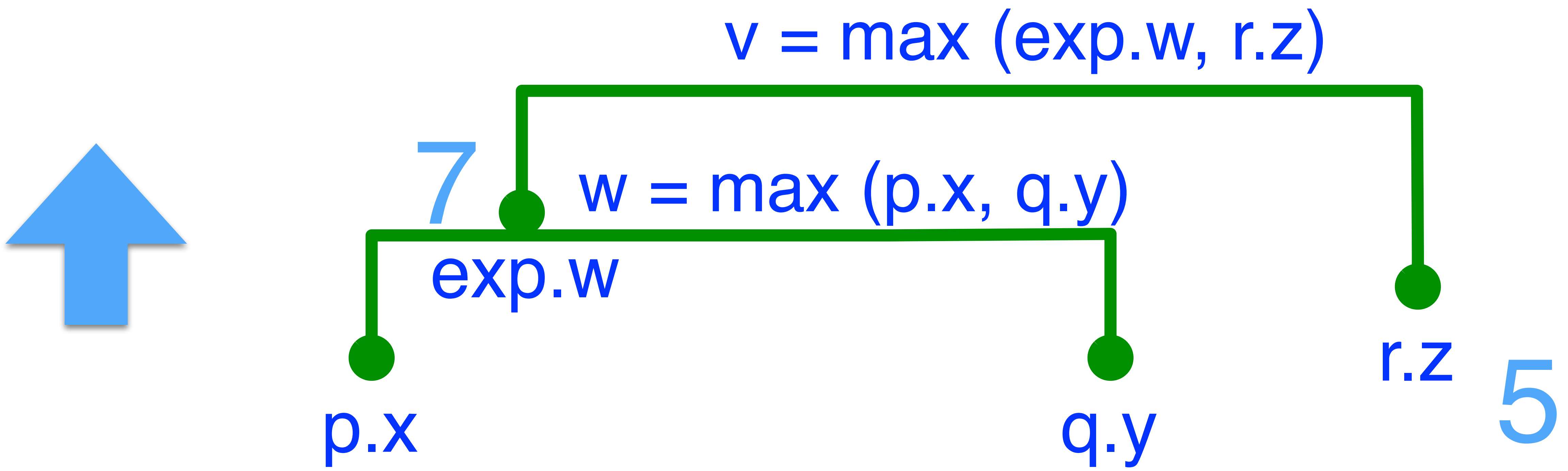
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



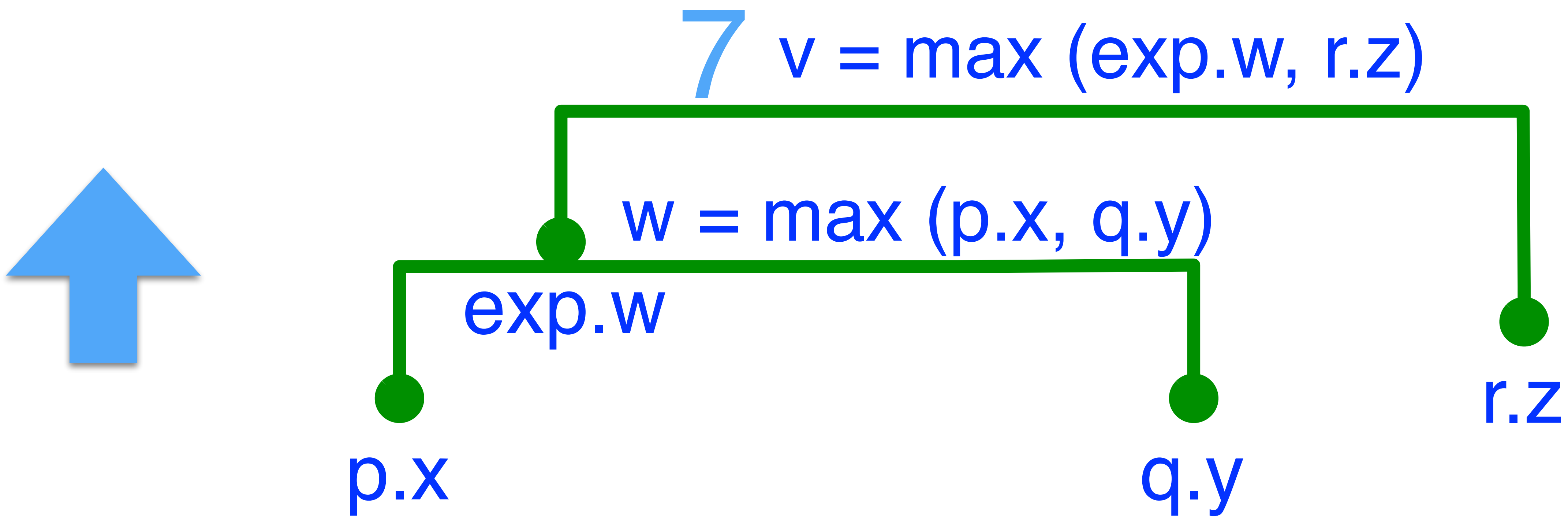
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



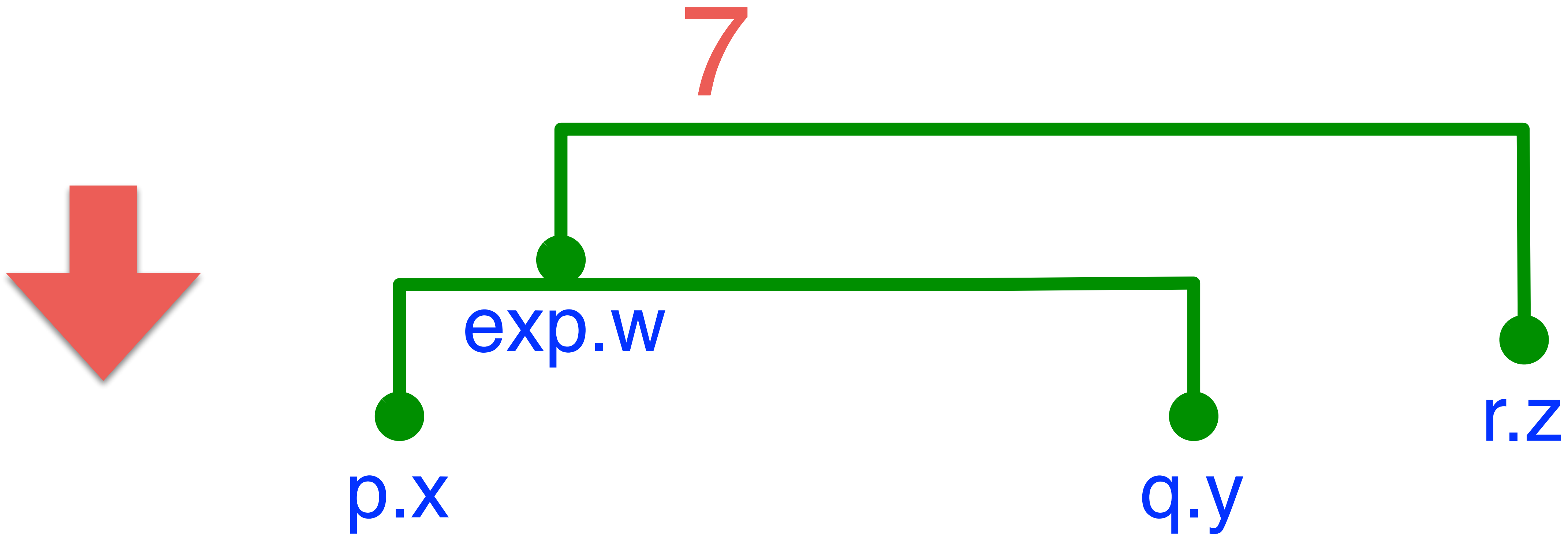
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



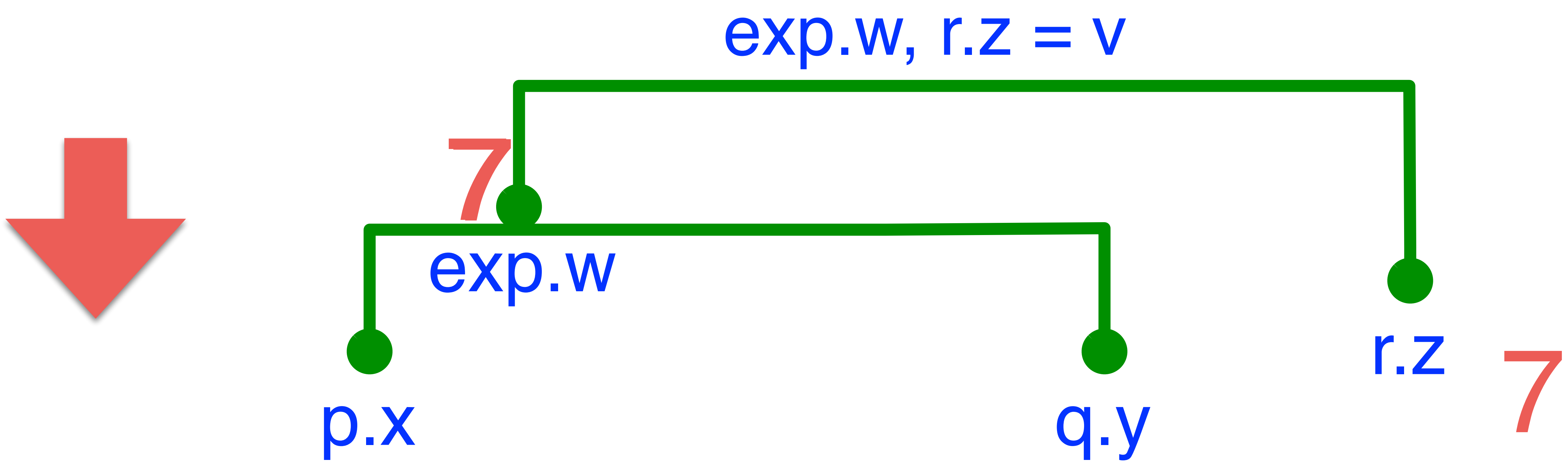
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



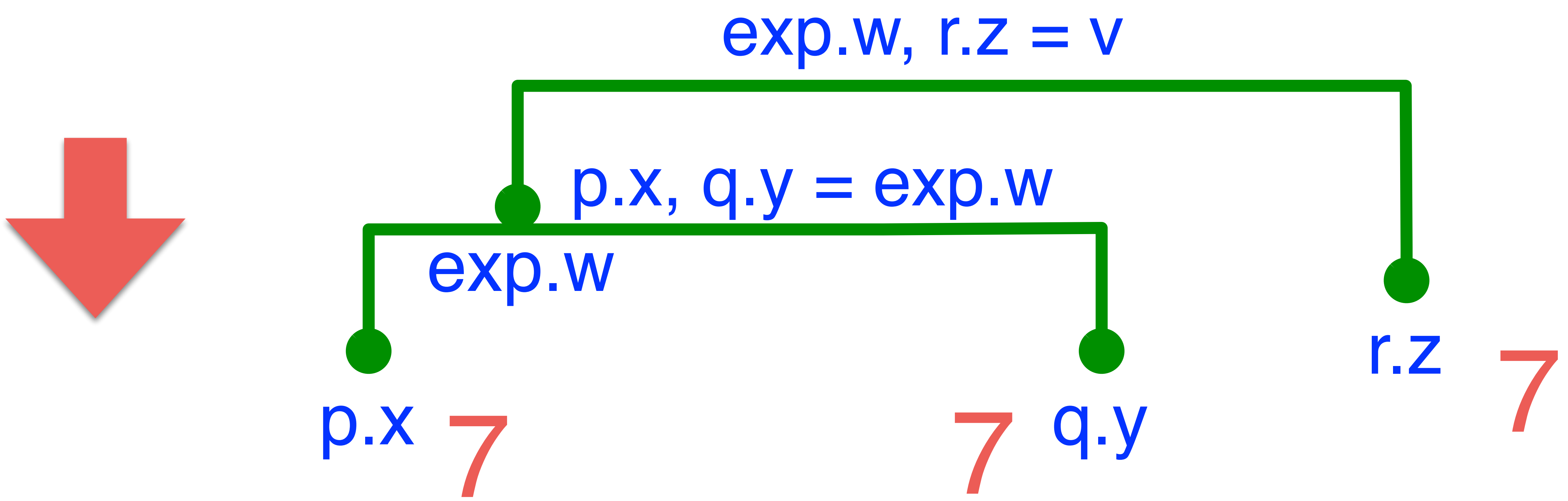
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

Data transfer



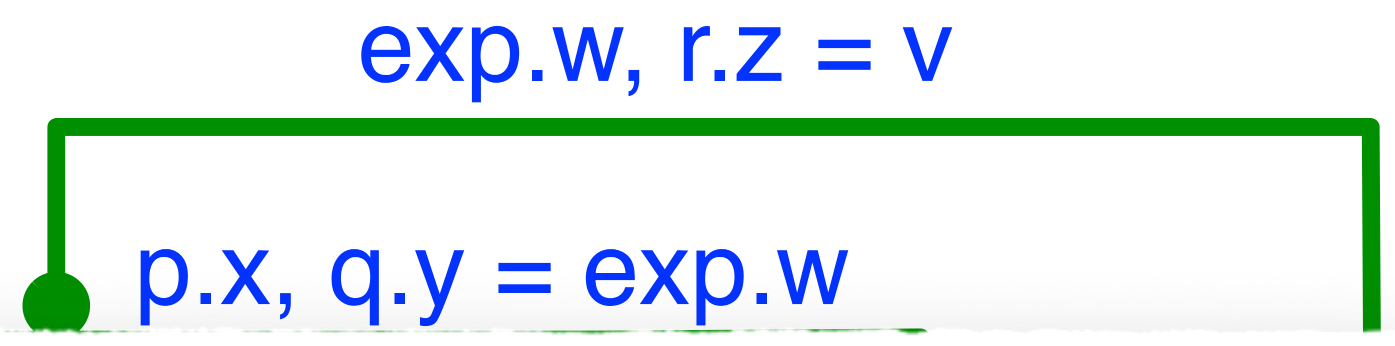
```
connector type Max (Port_int p, Port_int q)
   data int w
   export port Port_int exp(w)
   define p q
   up {w = max(p.v, q.v);}
   down {p.v = w; q.v = w;}
end
```

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



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.

```
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
```

Components of the robot

Safety constraints

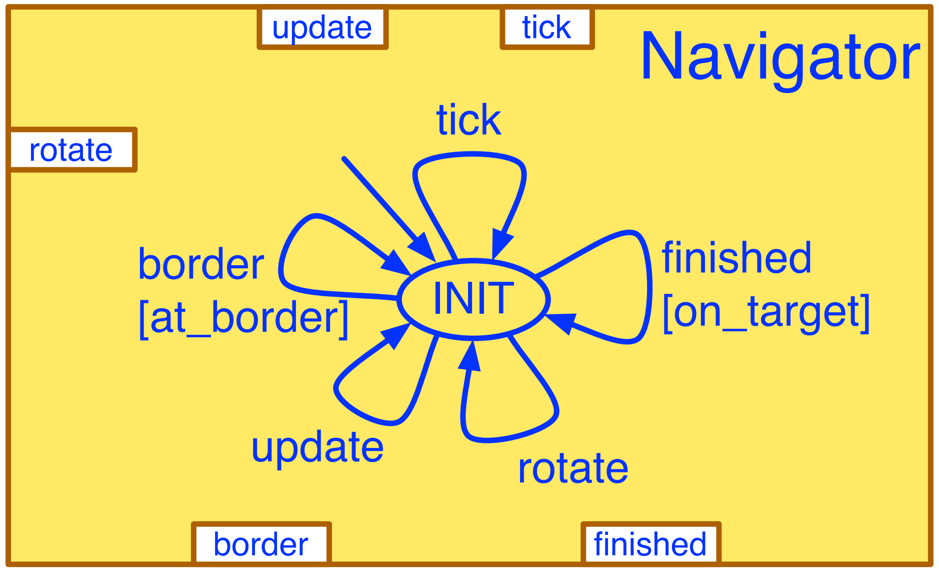
Shall not advance and rotate at the same time

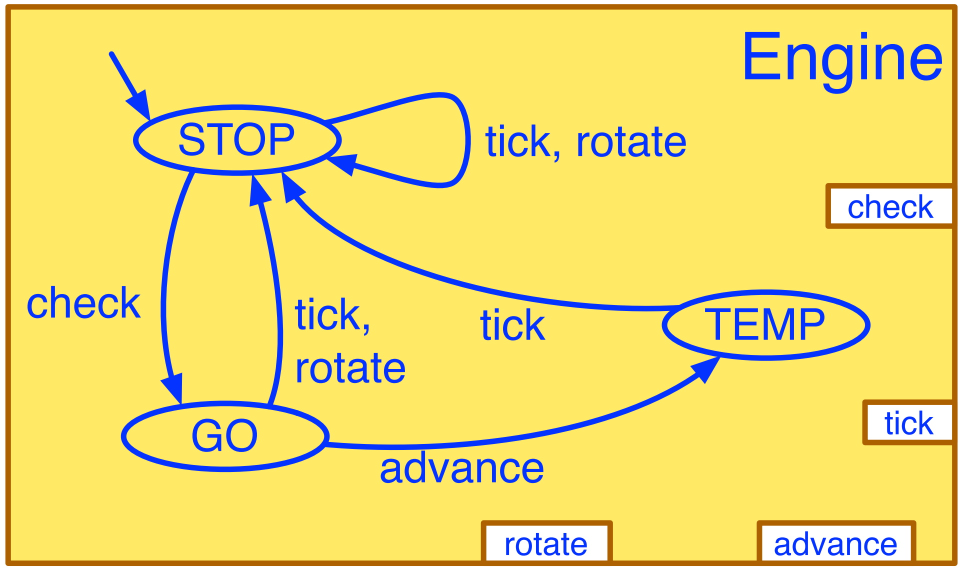
Shall stay within the region

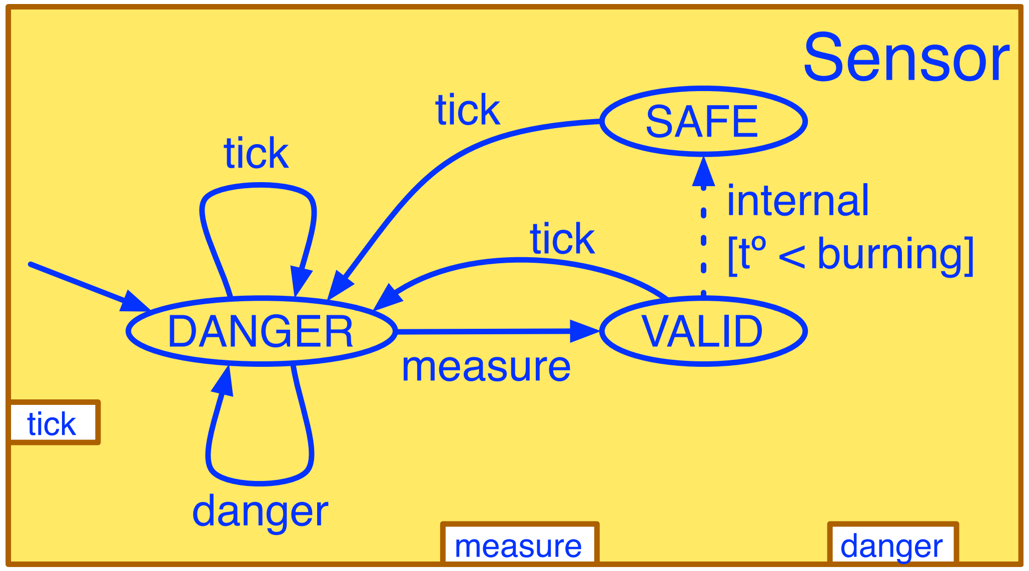
Shall stay in the area that is safe or hot (but not burning)

Shall update navigation and sensor data at each move

When objective is found, the robot shall stop







Components of the robot

Safety constraints

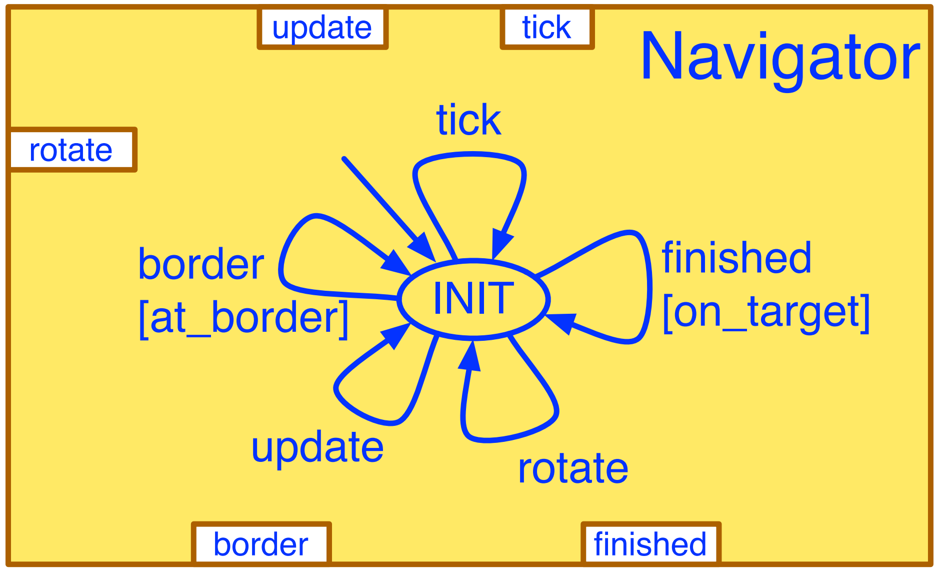
Shall not advance and rotate at the same time

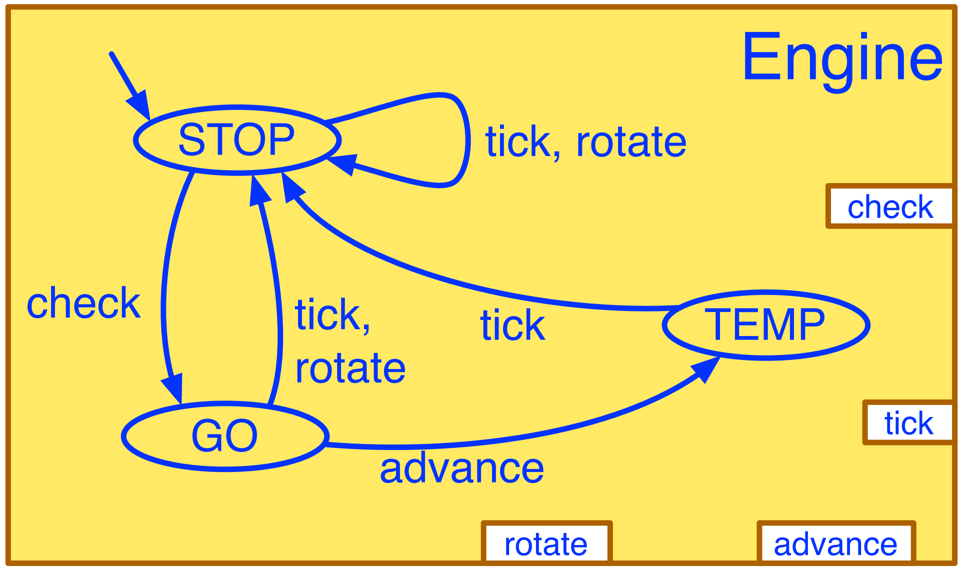
Shall stay within the region

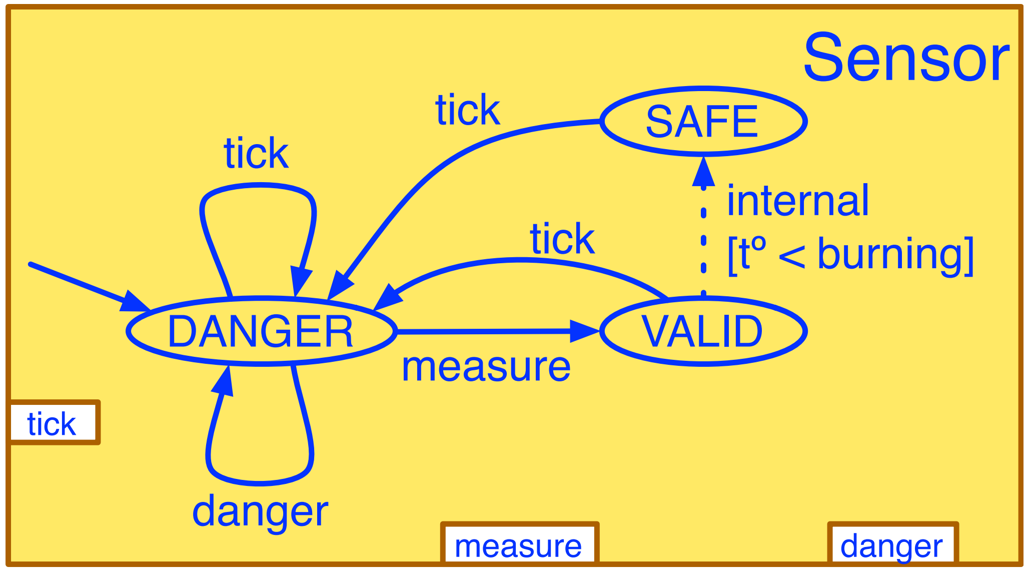
Shall stay in the area that is safe or hot (but not burning)

Shall update navigation and sensor data at each move

When objective is found, the robot shall stop







Components of the robot

Safety constraints

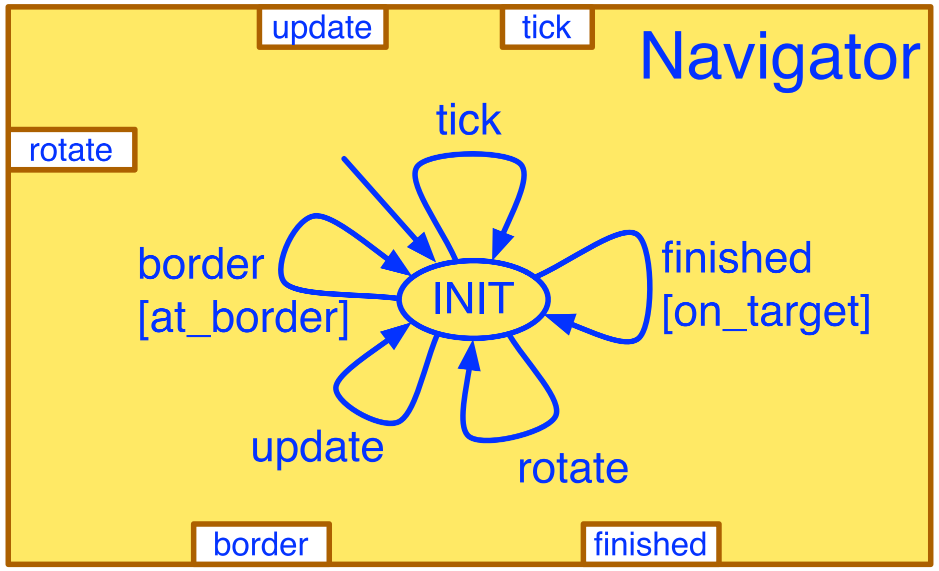
✓ Shall not advance and rotate at the same time

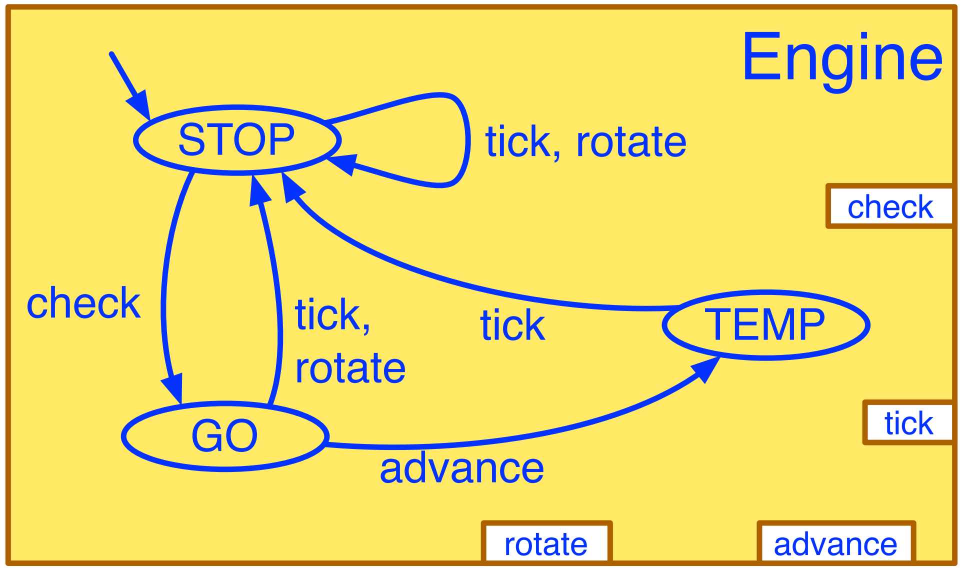
Shall stay within the region

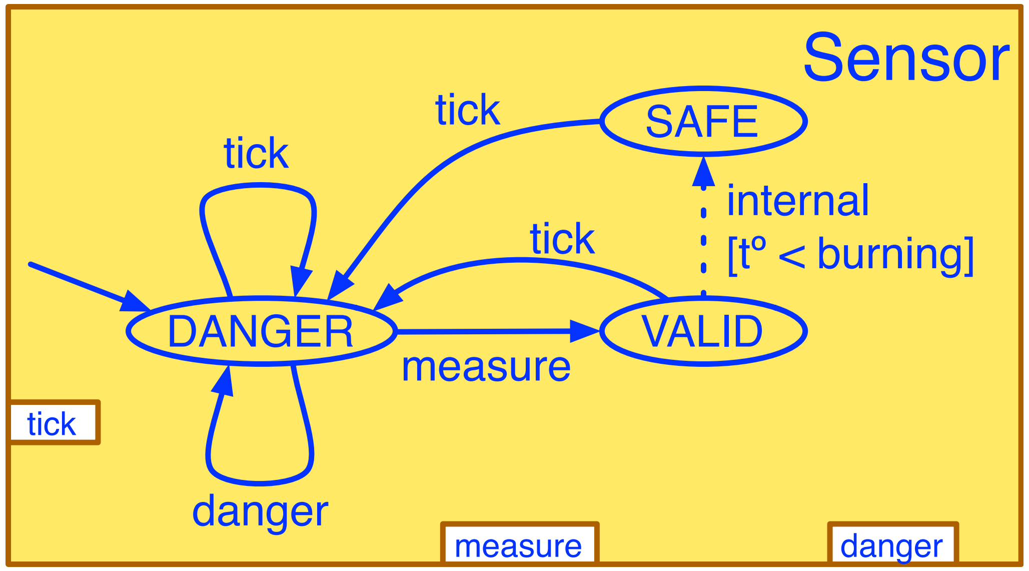
Shall stay in the area that is safe or hot (but not burning)

Shall update navigation and sensor data at each move

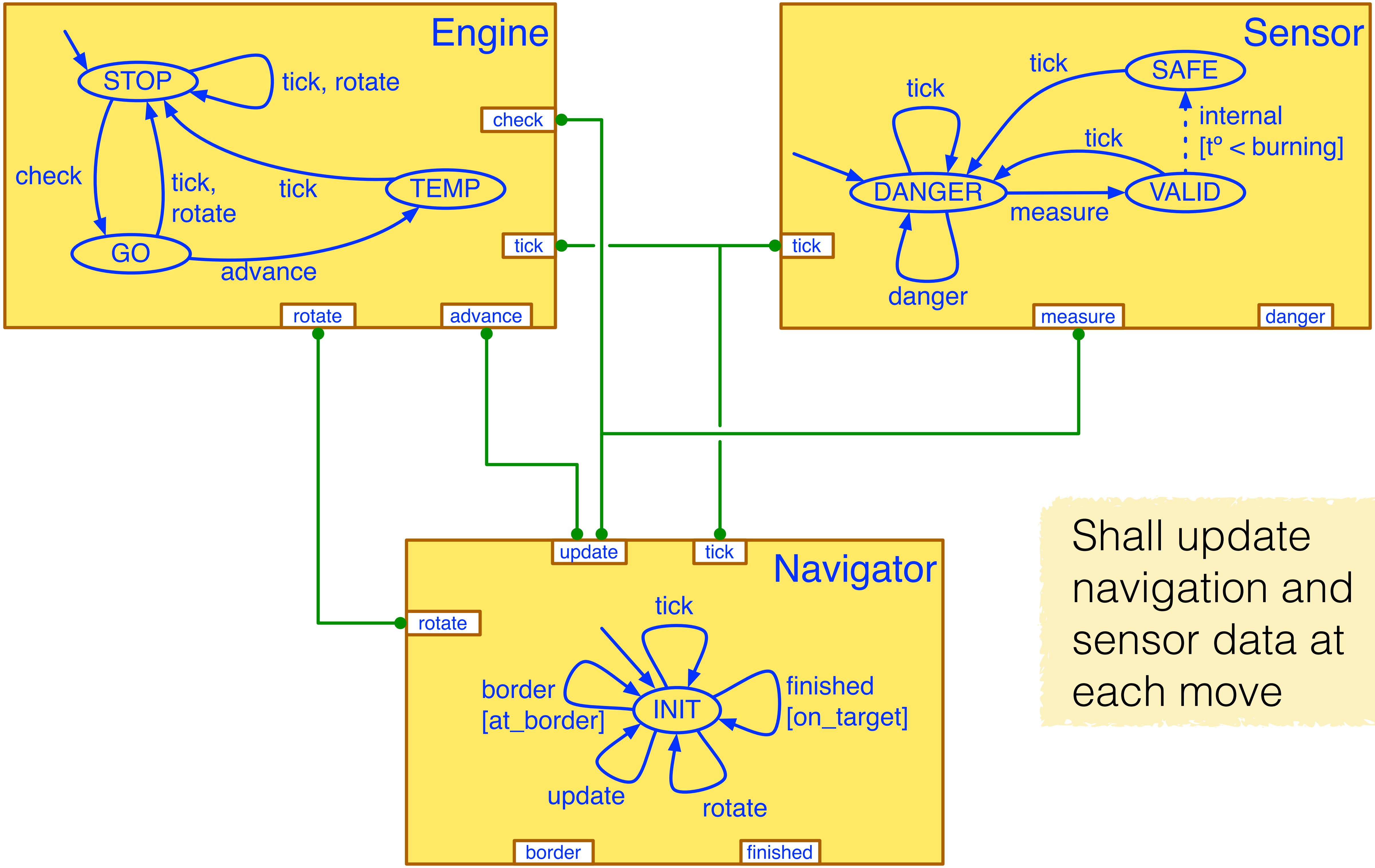
When objective is found, the robot shall stop





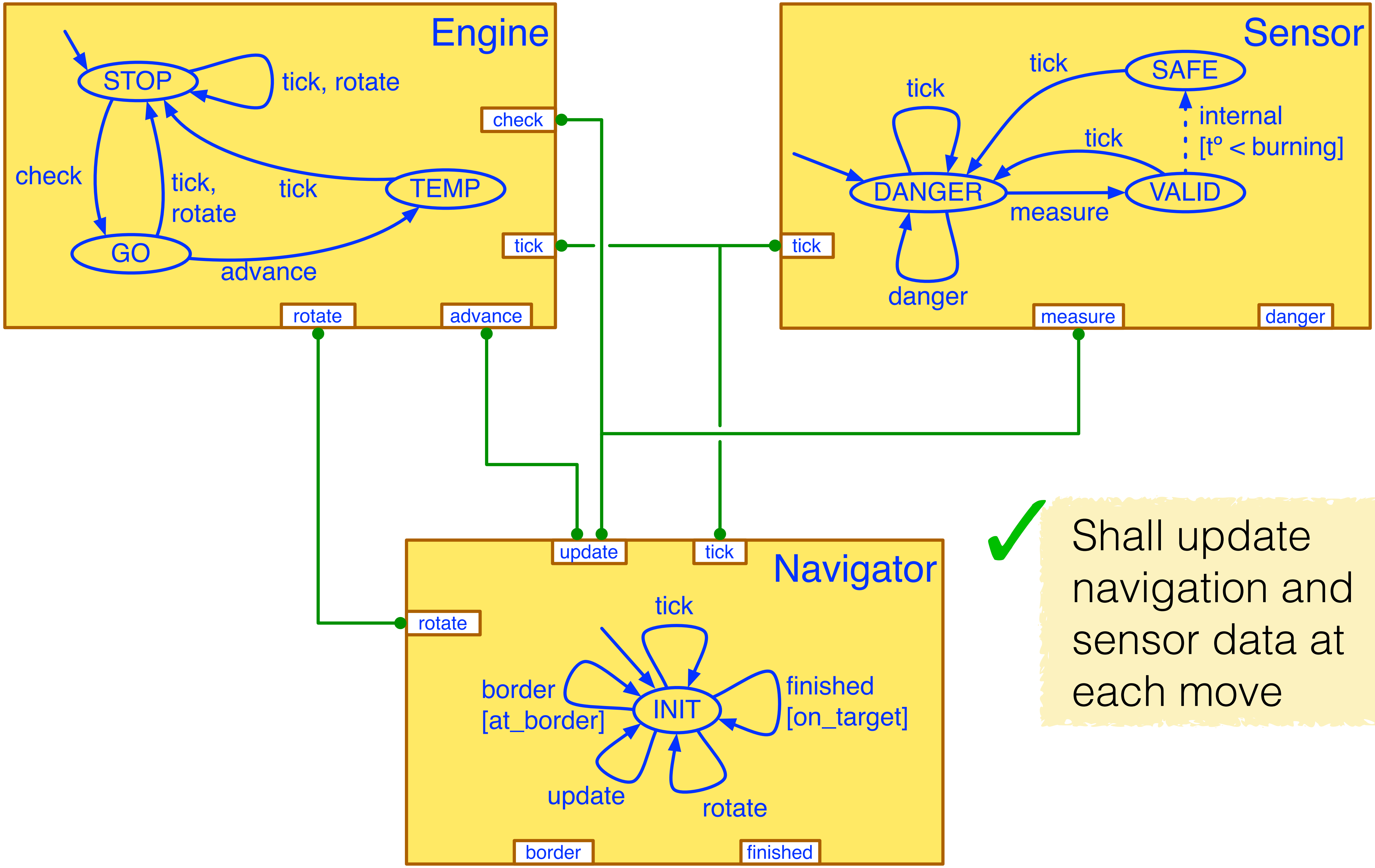


Connecting the robot



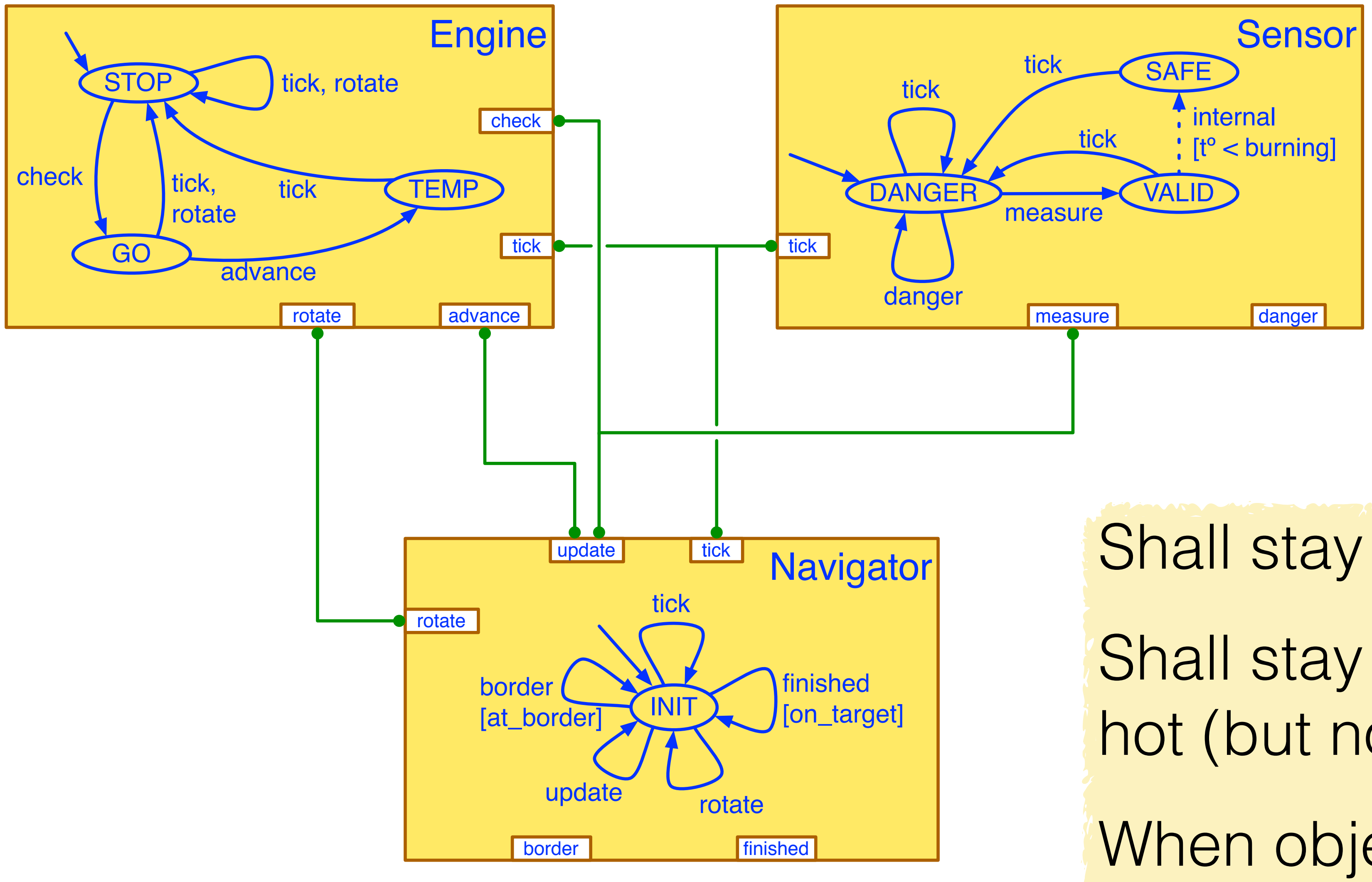
Shall update navigation and sensor data at each move

Connecting the robot



✓ Shall update navigation and sensor data at each move

Connecting the robot



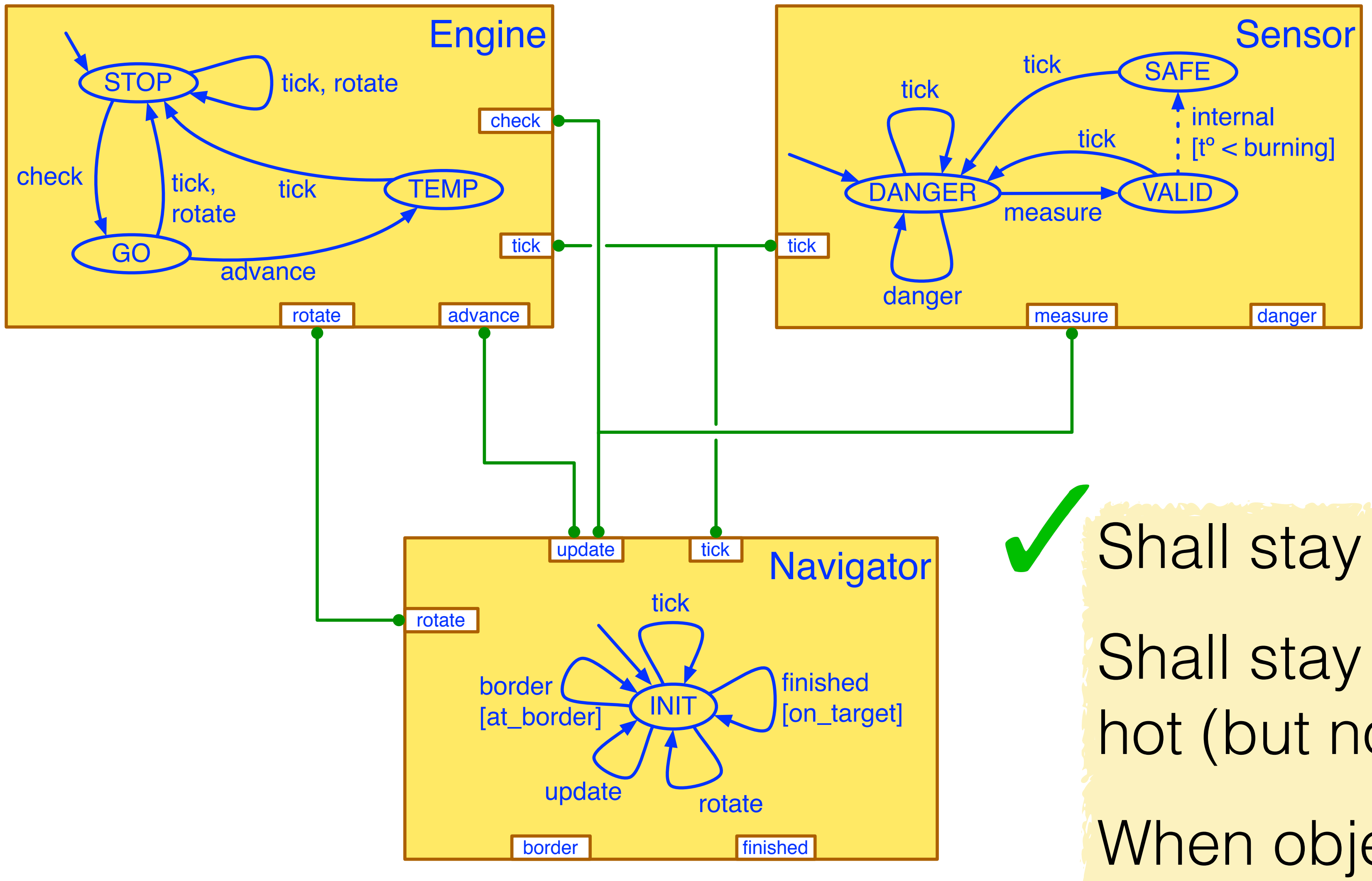
Shall stay within the region

Shall stay in the area that is safe or hot (but not burning)

When objective is found, the robot shall 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



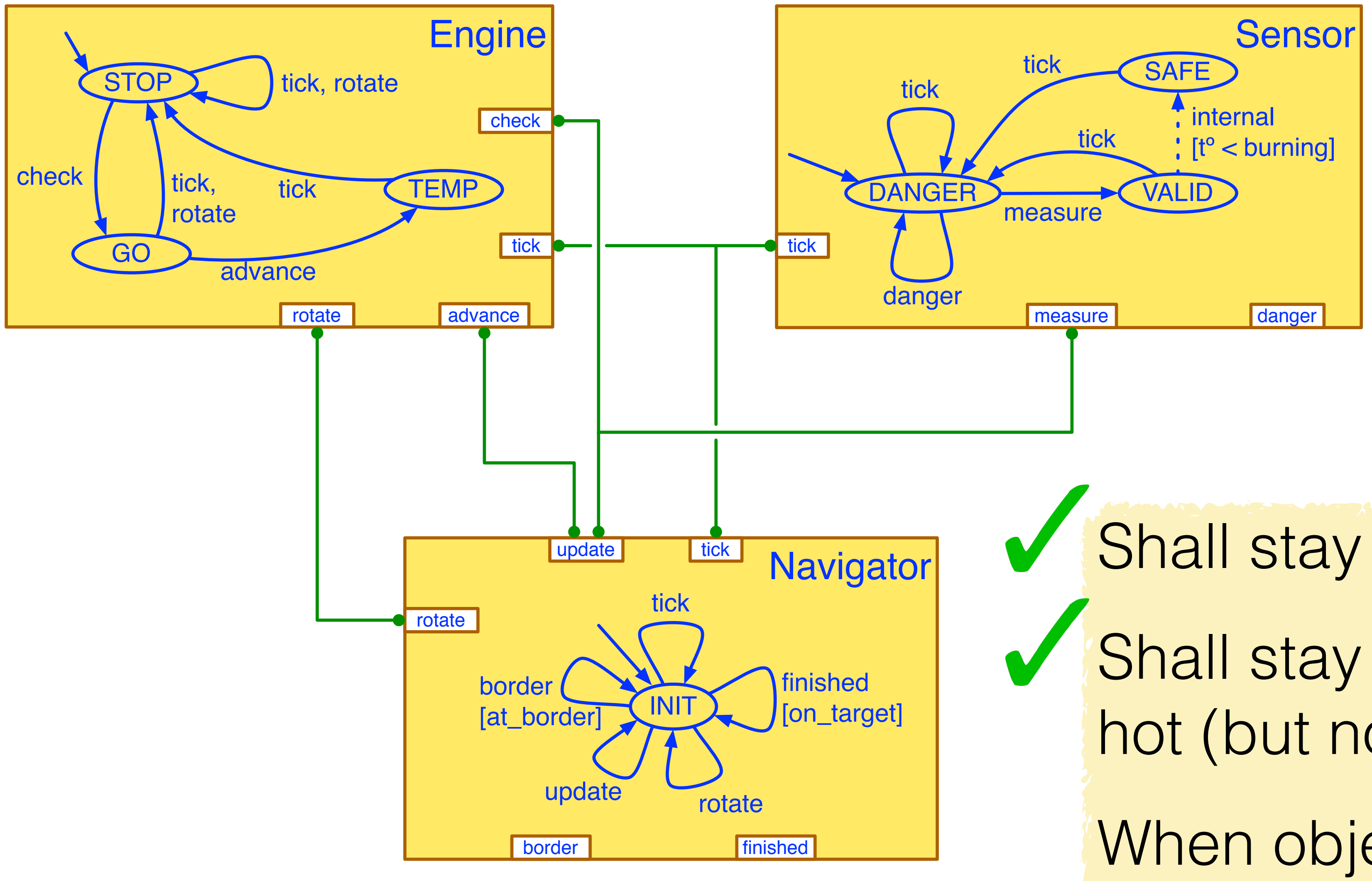
✓ Shall stay within the region

Shall stay in the area that is safe or hot (but not burning)

When objective is found, the robot shall 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



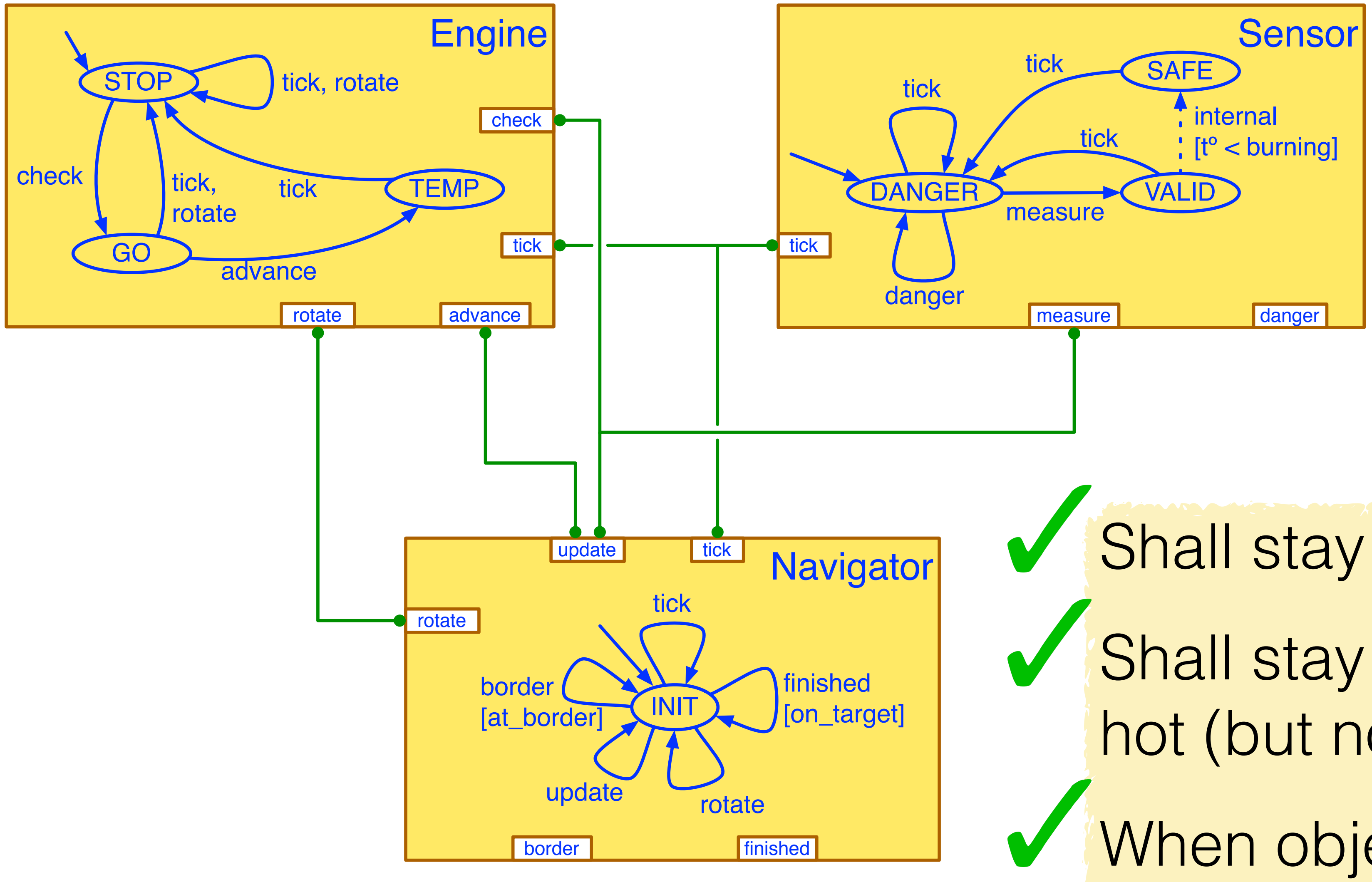
✓ Shall stay within the region

✓ Shall stay in the area that is safe or hot (but not burning)

When objective is found, the robot shall 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



✓ Shall stay within the region

✓ Shall stay in the area that is safe or hot (but not burning)

✓ When objective is found, the robot shall 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

Outline

Practical aspects

Overview of the RSD approach

CubETH case study

Operational semantics

BIP language introduction

Theoretical aspects

Connector modelling

Architectures: design patterns for BIP

Connector synthesis

Expressiveness study

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 basic language and the reference Engine

BIP framework (at different stages of maturity)

- Several other language flavours
- Several engine implementations
- Analysis & verification tools