Correctness by Construction: Design of Component-Based Systems Using BIP

18th of May, 2018

Simon Bliudze
Équipe-projet SPIRALS

Me in a nutshell

1993–1998	University of St. Petersbourg (Russia)	MSc in Mathematics
1998–2000	IST Ltd. (Londres, UK)	Consultant (ERP systems)
2000–2001	Paris 6	DEA Algorithmique
2001–2006	LIX, École polytechnique (Palaiseau)	PhD with Daniel Krob
2006–2008	Verimag (Grenoble)	Post-doctorate with Joseph Sifakis
2008–2011	CEA LIST (Saclay)	Research engineer
2011–2017	École polytechnique fédérale de Lausanne	Scientific collaborator (Sifakis, Le Boudec)
since 2017	Inria Lille – Nord Europe	Chargé de recherche

Team SPIRALS

Self-adaptation for distributed services and large software systems

My project

Rigorous Component-Based Design
of Correct-by-Construction Software and Systems

Formal semantics of concurrent systems

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

BIP

Design patterns

JavaBIP

Post-doc

EPFL

Formal semantics of concurrent syste

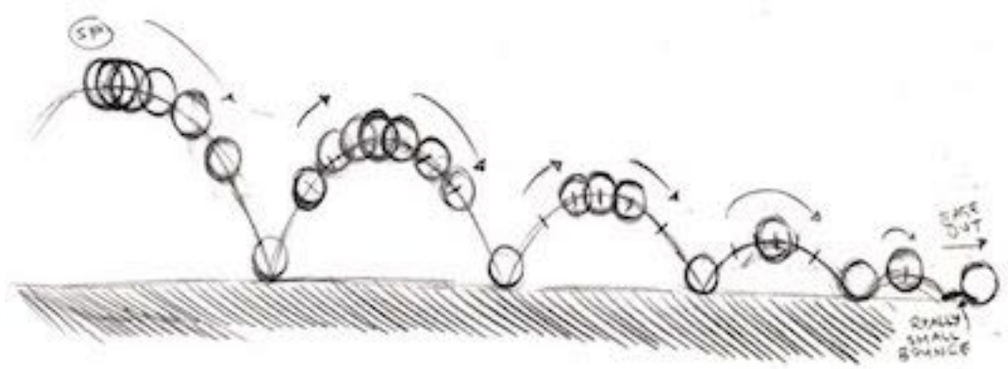
$$\frac{q \xrightarrow{a} q' \qquad \forall a \prec a',\ }{q \xrightarrow{a}_{\prec} q'}$$

BIP

Design patterns

JavaBIP

Semantics of hybrid systems



PhD continuous / discrete

« Turing machine »

EPFL Signals Modelica (Zélus)

Non-standard analysis

Formal semantics of concurrent systems

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

BIP

Design pattern

JavaBIP

Semantics of hybrid systems

Real-time systems

Modelling and implementation

Model transformation

Delay tolerance

Redundancy

CEA

EPFL

Formal semantics of concurrent s

$$\frac{q \xrightarrow{a} q' \qquad \forall a \prec}{q \xrightarrow{a}_{\prec}}$$

BIP

Design pat

JavaBI

Real-time sy

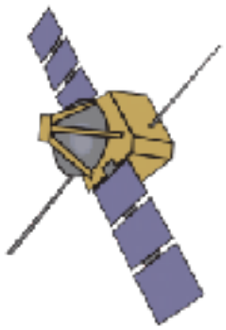
Modelling and imple

Model transfor

Delay tolera ce

Redundancy

Modelling and analysis of critical systems



Control software

Nanosatellite Smartgrid EPFL

Power plant

CEA

Static analysis

Formal semantics of concurrent systems

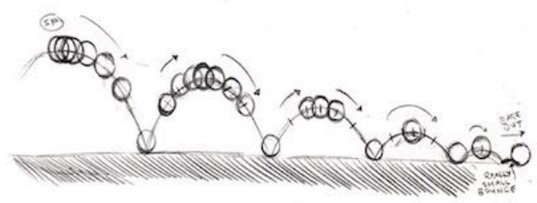
$$\frac{q \xrightarrow{a} q' \qquad \forall a \prec a',\ q \not\xrightarrow{a'}}{q \xrightarrow{a}_{\prec} q'}$$

BIP

Design patterns

JavaBIP

Semantics of hybrid systems



continuous / discrete

« Turing machine »

Signals Modelica (Zélus)

Non-standard analysis

Extend the system design methodology

Real-time systems

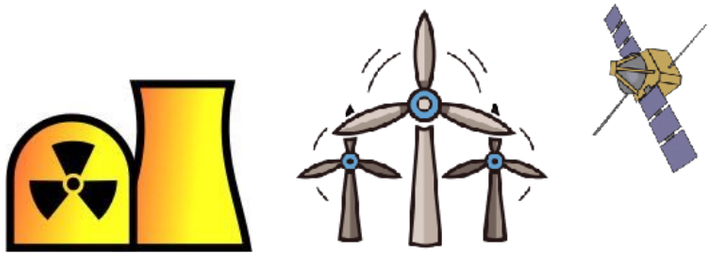
Modelling and implementation

Model transformation

Delay tolerance

Redundancy

Modelling and analysis of critical systems



Control software

Nanosatellite Smartgrid

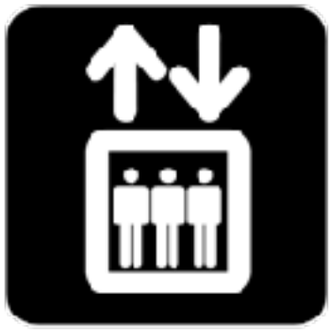
Power plant

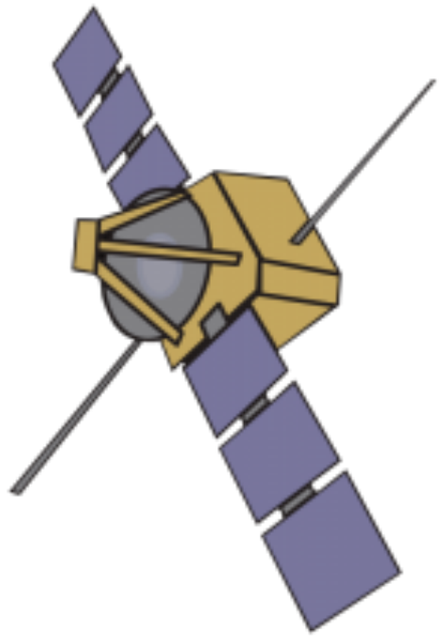
Static analysis

to adaptive software and automatic computing

Concurrency...

…is everywhere!





Embedded

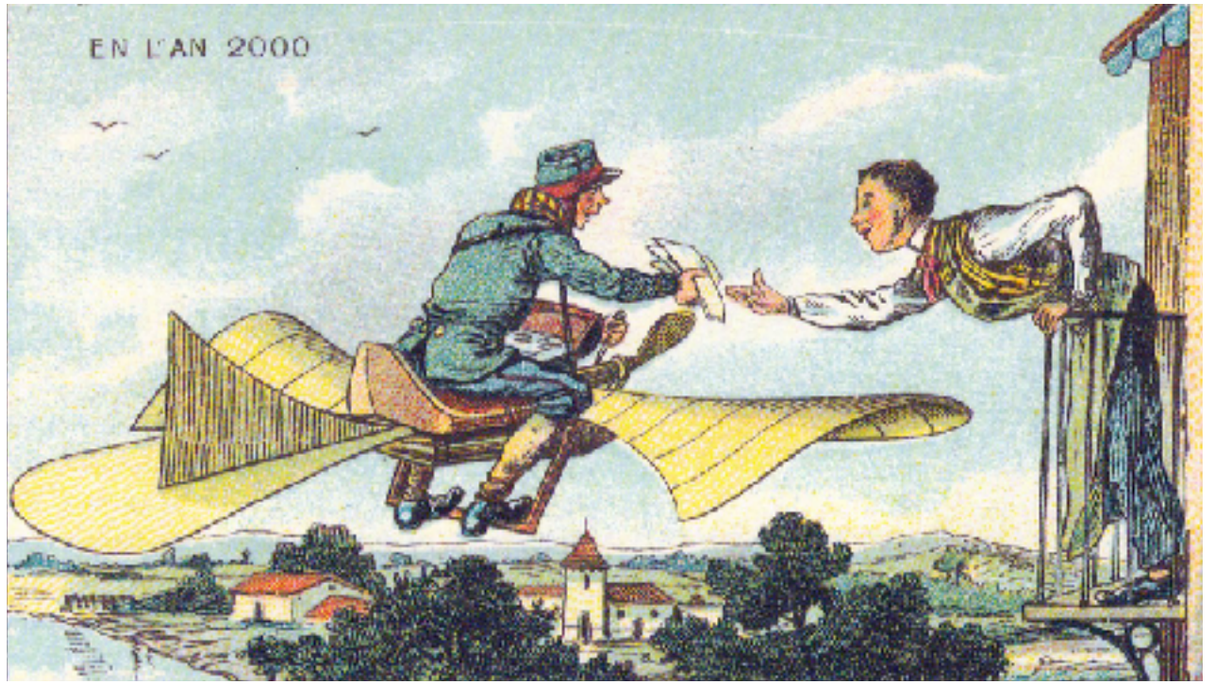
Infrastructure

Platform

Services

…you name it!

Coordination



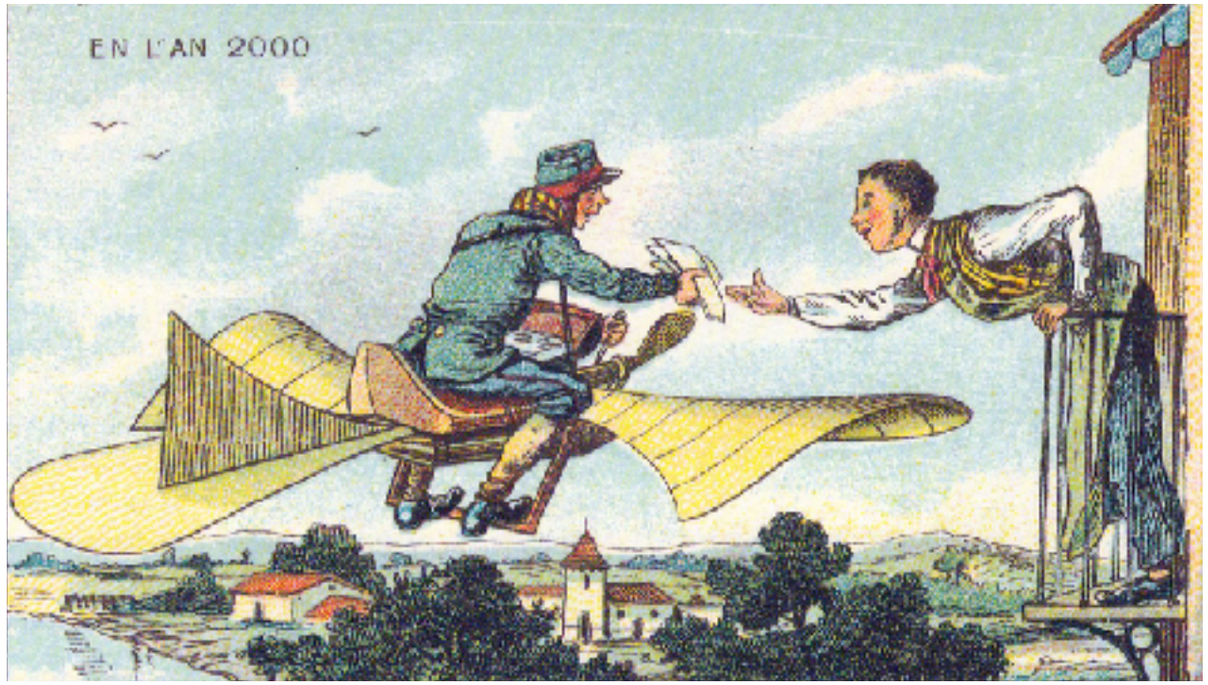
Control-centric

- Synchronisation is primitive
- Locks, semaphores etc.
- Concurrent execution
- Critical systems

Data-centric

- Data exchange is primitive
- Messages, split-join etc.
- Distributed execution
- Data-intensive computation

Coordination



The two views are complementary

Control-centric

- Synchronisation is primitive
- Locks, semaphores etc.
- Concurrent execution
- Critical systems

Data-centric

- Data exchange is primitive
- Messages, split-join etc.
- Distributed execution
- Data-intensive computation

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);
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);
```

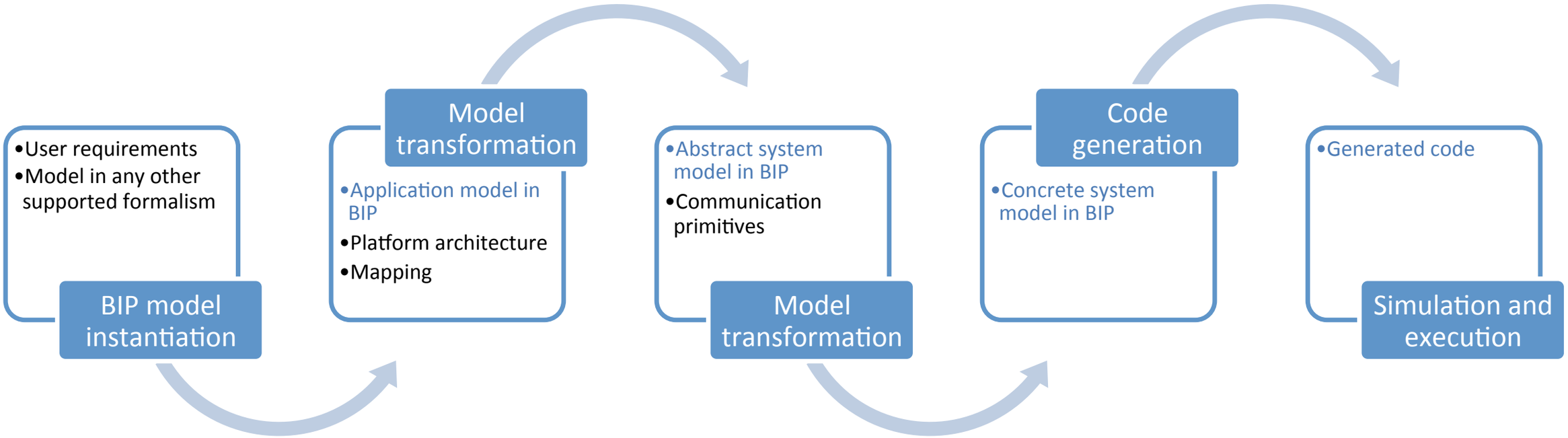
Coordination mechanisms mix up with computation and do not scale.
Code maintenance is a nightmare!

Objectives

Correct-by-construction concurrent systems

Separation of computation from coordination

Rigorous System Design flow

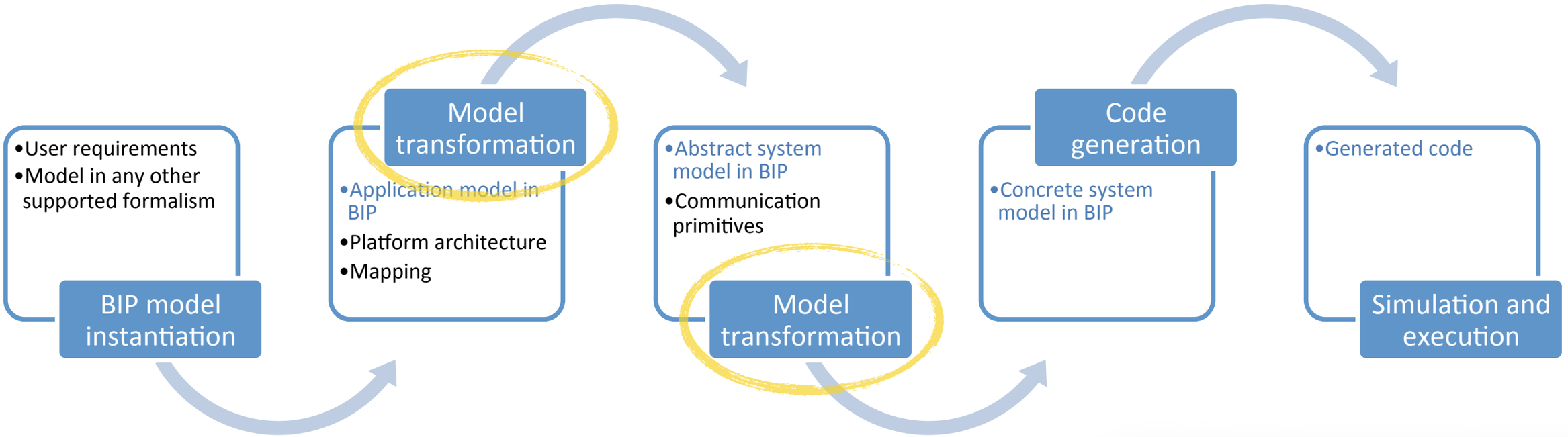


A series of semantics-preserving transformations

Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

Rigorous System Design flow



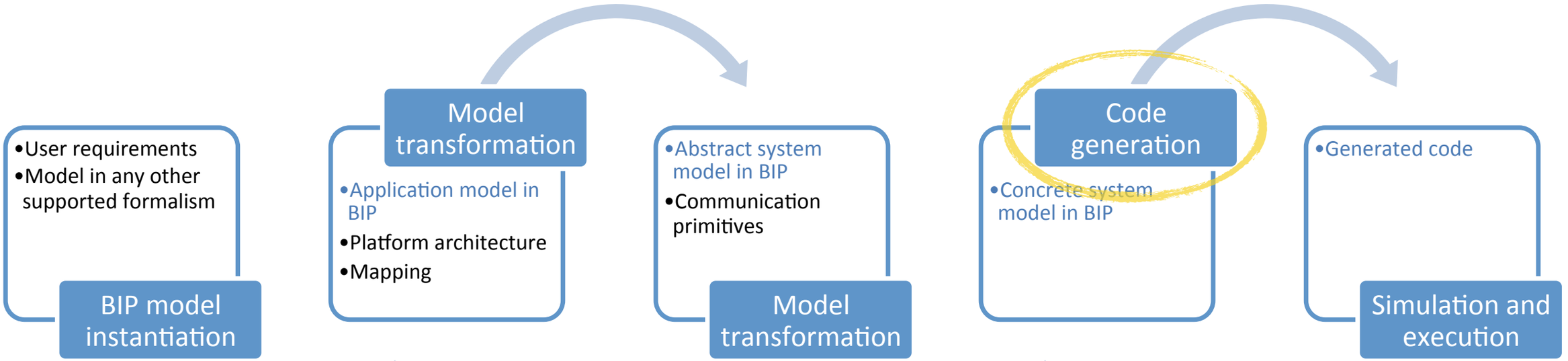
A series of semantics-preserving transformations

Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

- ☐ Unifying modelling framework

Rigorous System Design flow



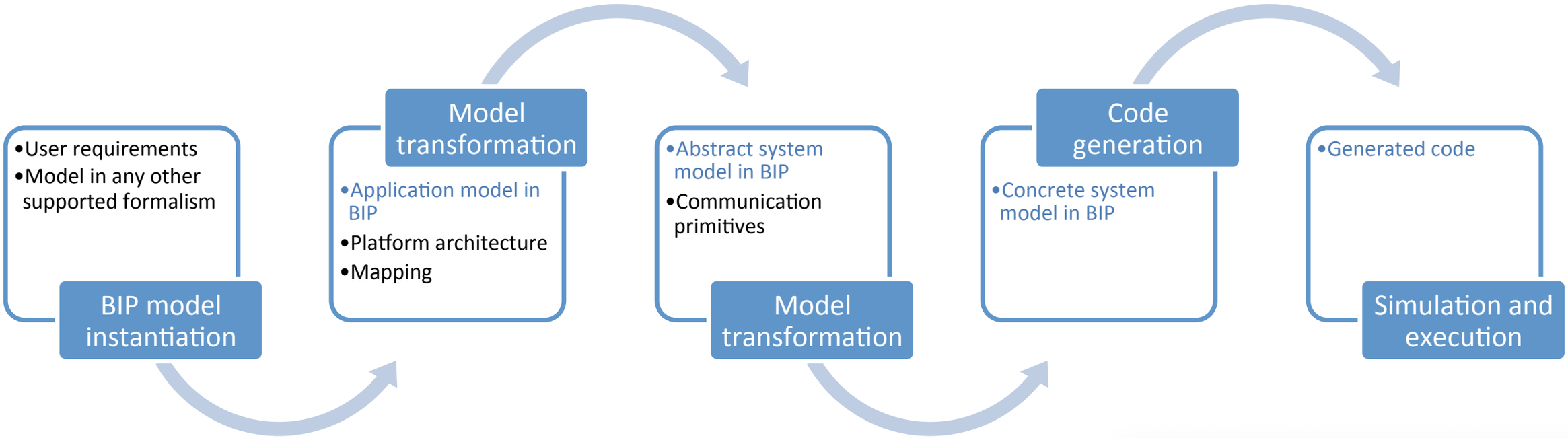
A series of semantics-preserving transformations

Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

- ☐ Unifying modelling framework
- ☐ Operational semantics

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 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 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

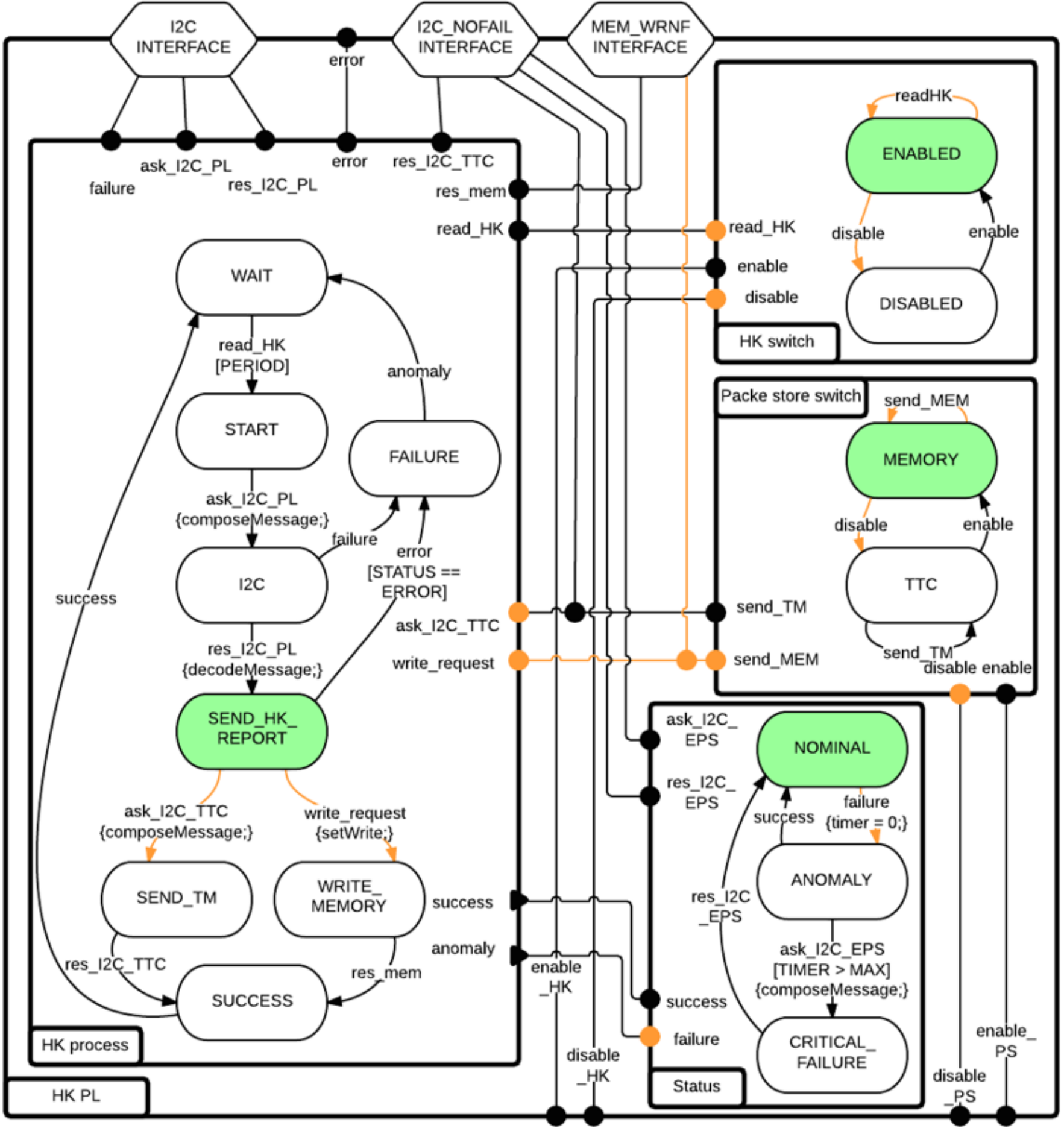
Control and Data Management System (CDMS)

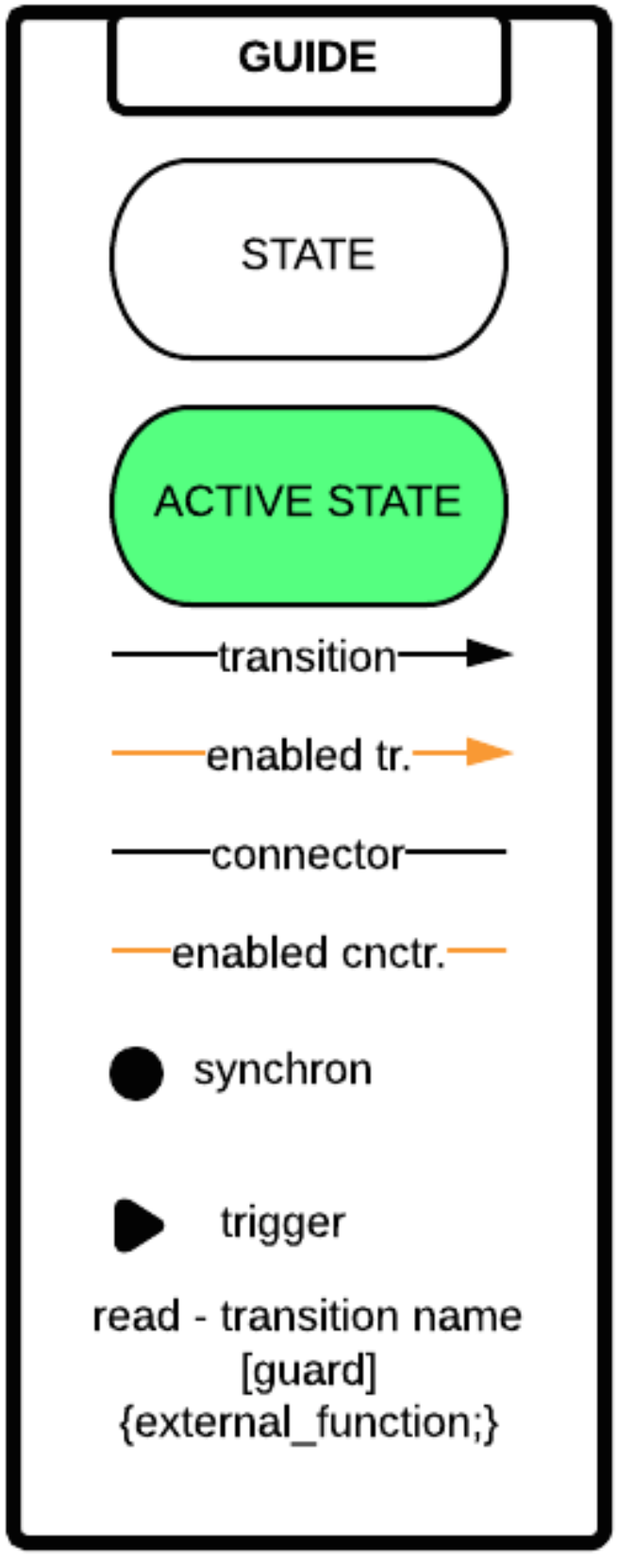
BIP = Behaviour-Interaction-Priority

A collaboration with ThalesAlenia Space (France) and Aristotle University of Thessaloniki (Greece)

"Catalogue of System and Software Properties"

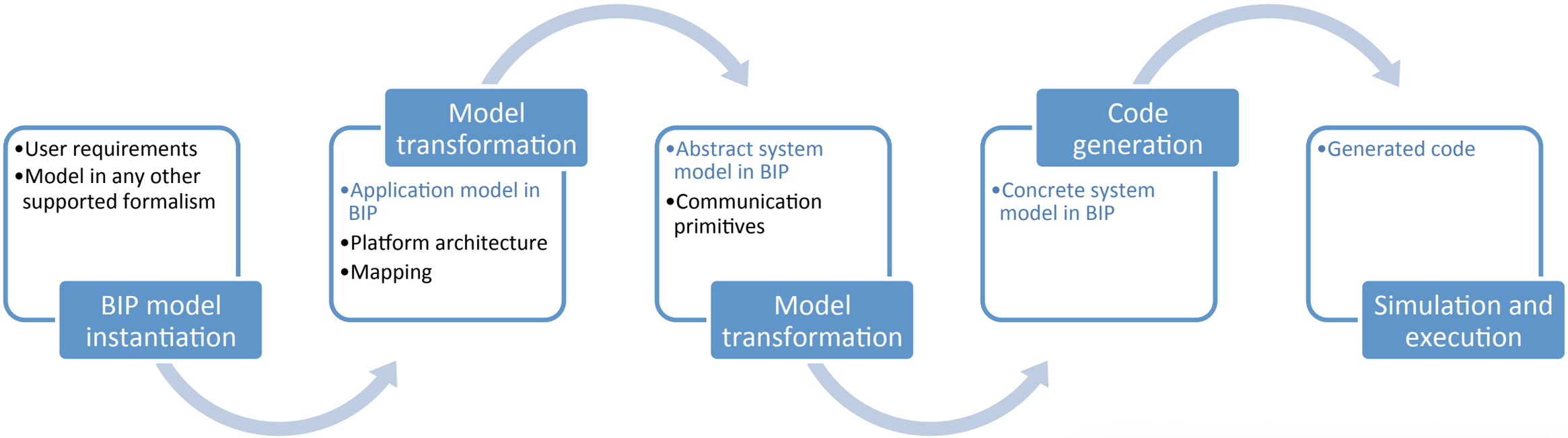
Funded by ESA





slide courtesy of
Marco Pagnamenta

Rigorous System Design flow



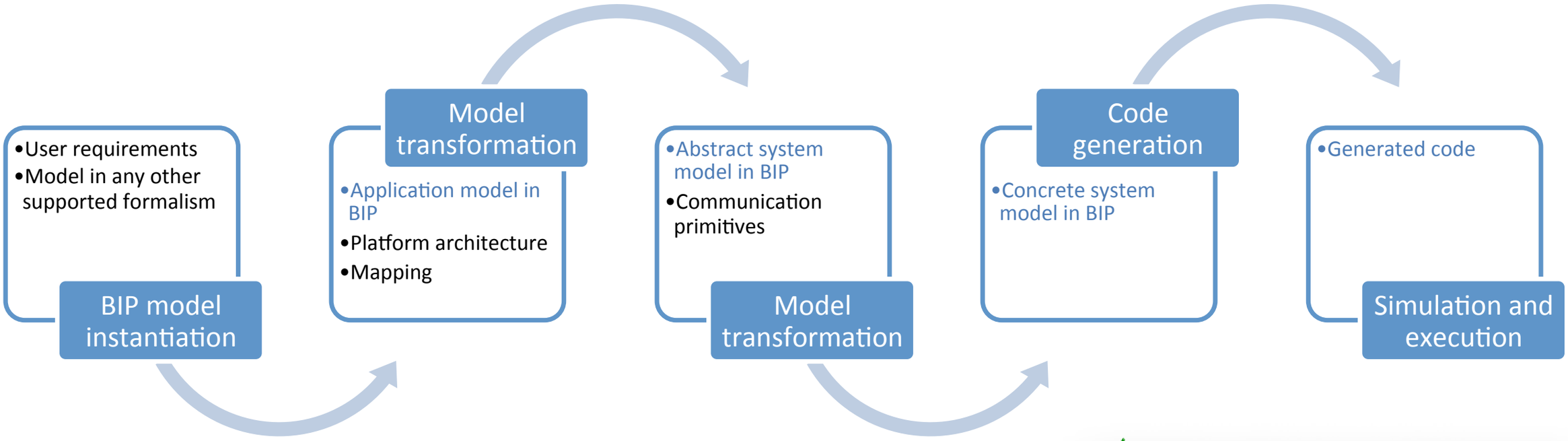
A series of semantics-preserving transformations

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

Rigorous System Design flow



A series of semantics-preserving transformations

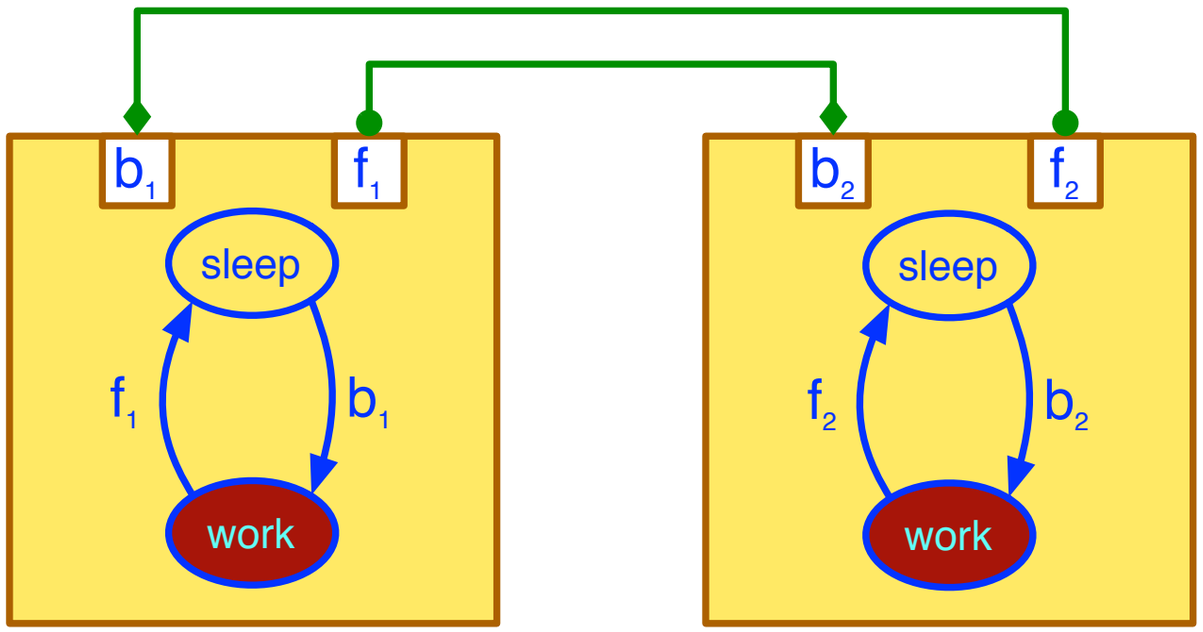
Correctness decomposed into
- correctness of transformations
- correctness of high-level models

Final implementation is **correct by construction**

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

se•man•tic
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 langua
over time. 2. the branch of se
tionship between signs or sy
meaning. or an interpretation
Let's not argue
—se•man′ti•c

BIP by example: Mutual exclusion



Interaction model:

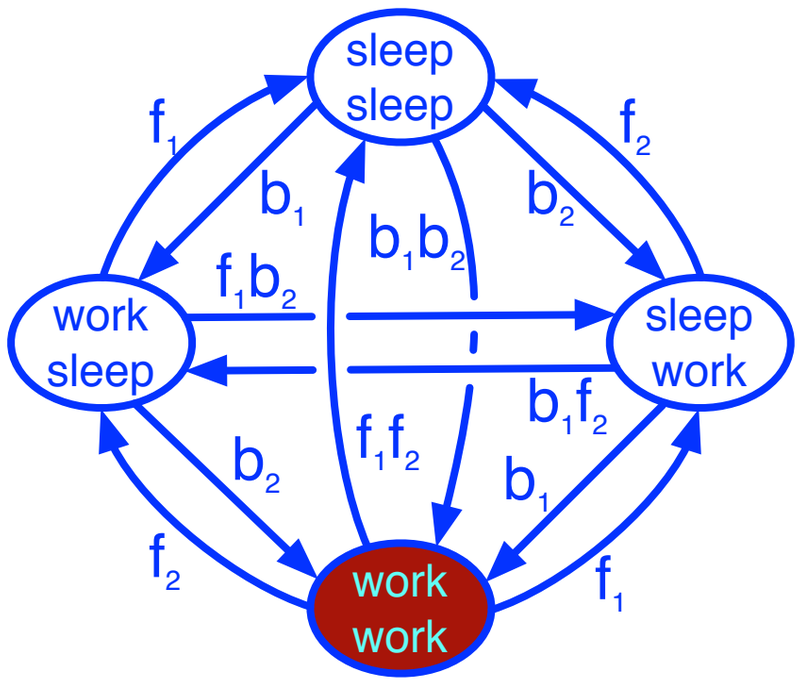
$$\{b_1,\ f_1,\ b_2,\ f_2,\ b_1f_2,\ b_2f_1\}$$

Maximal progress:

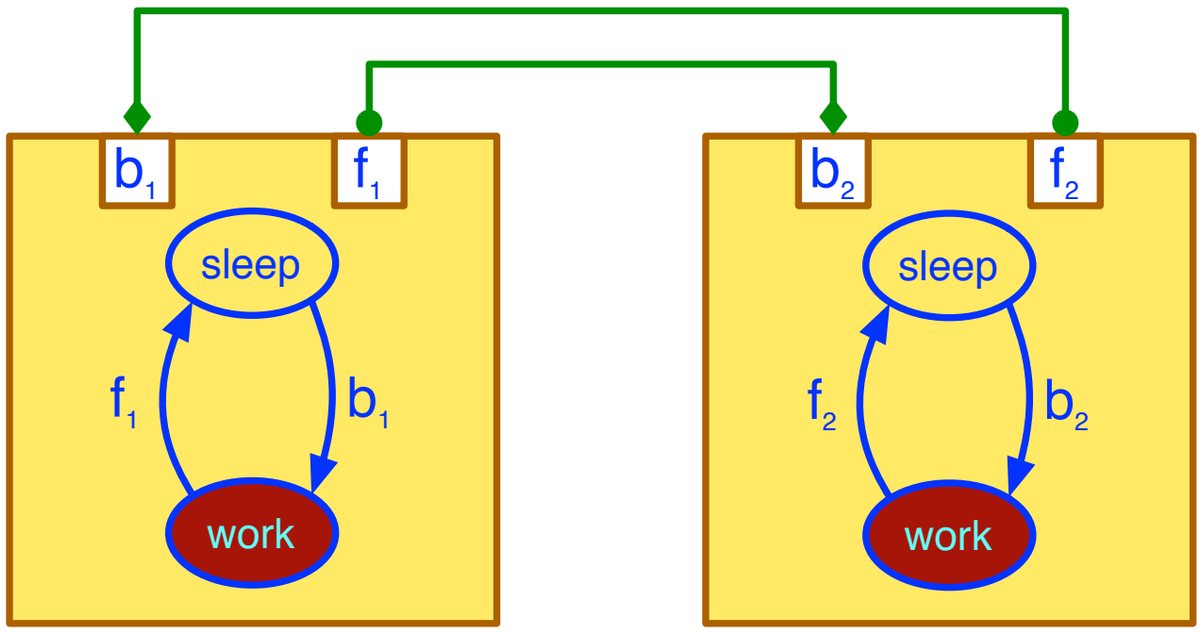
$$b_1 < b_1f_2,\ b_2 < b_2f_1$$

Design view

Semantic view



BIP by example: Mutual exclusion



Interaction model:

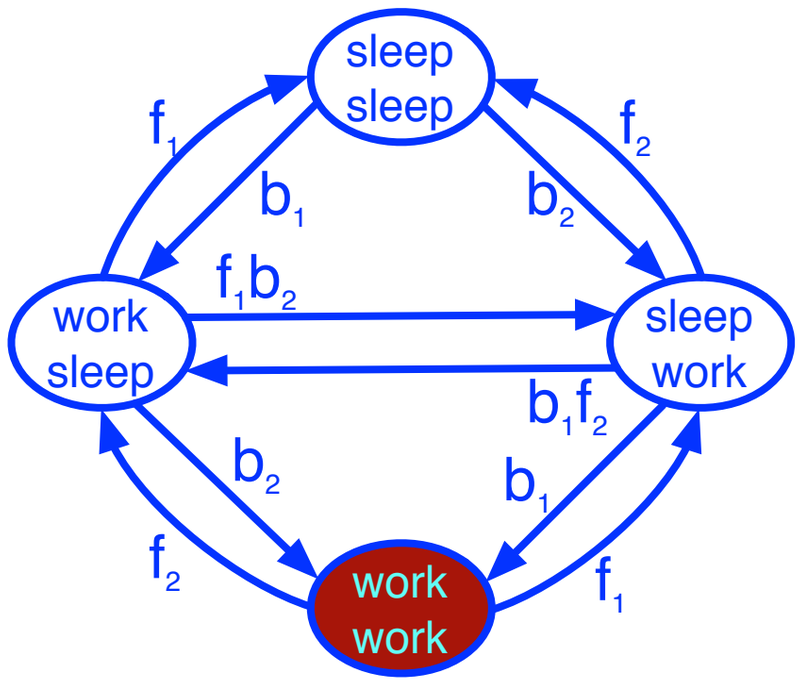
$$\{b_1,\ f_1,\ b_2,\ f_2,\ b_1f_2,\ b_2f_1\}$$

Maximal progress:

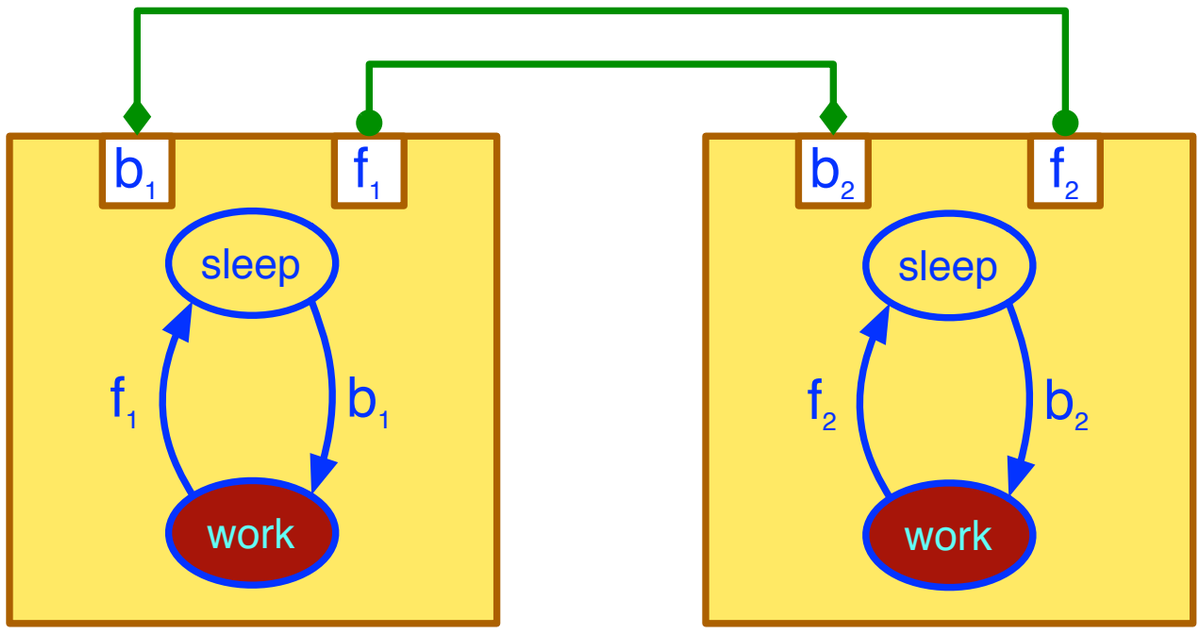
$$b_1 < b_1f_2,\ b_2 < b_2f_1$$

Design view

Semantic view



BIP by example: Mutual exclusion



Interaction model:

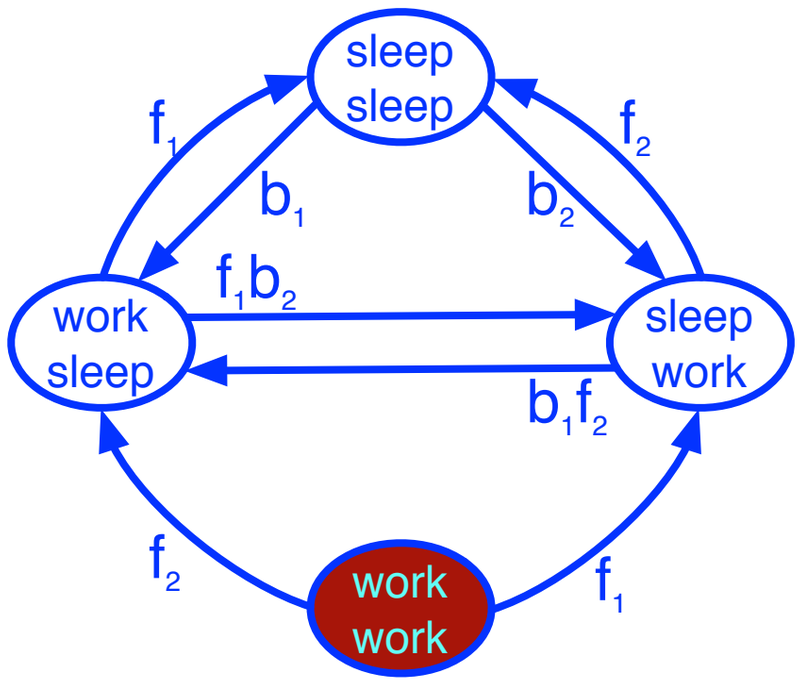
$$\{b_1,\ f_1,\ b_2,\ f_2,\ b_1f_2,\ b_2f_1\}$$

Maximal progress:

$$b_1 < b_1f_2,\ b_2 < b_2f_1$$

Design view

Semantic view



Semantics: Interactions

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

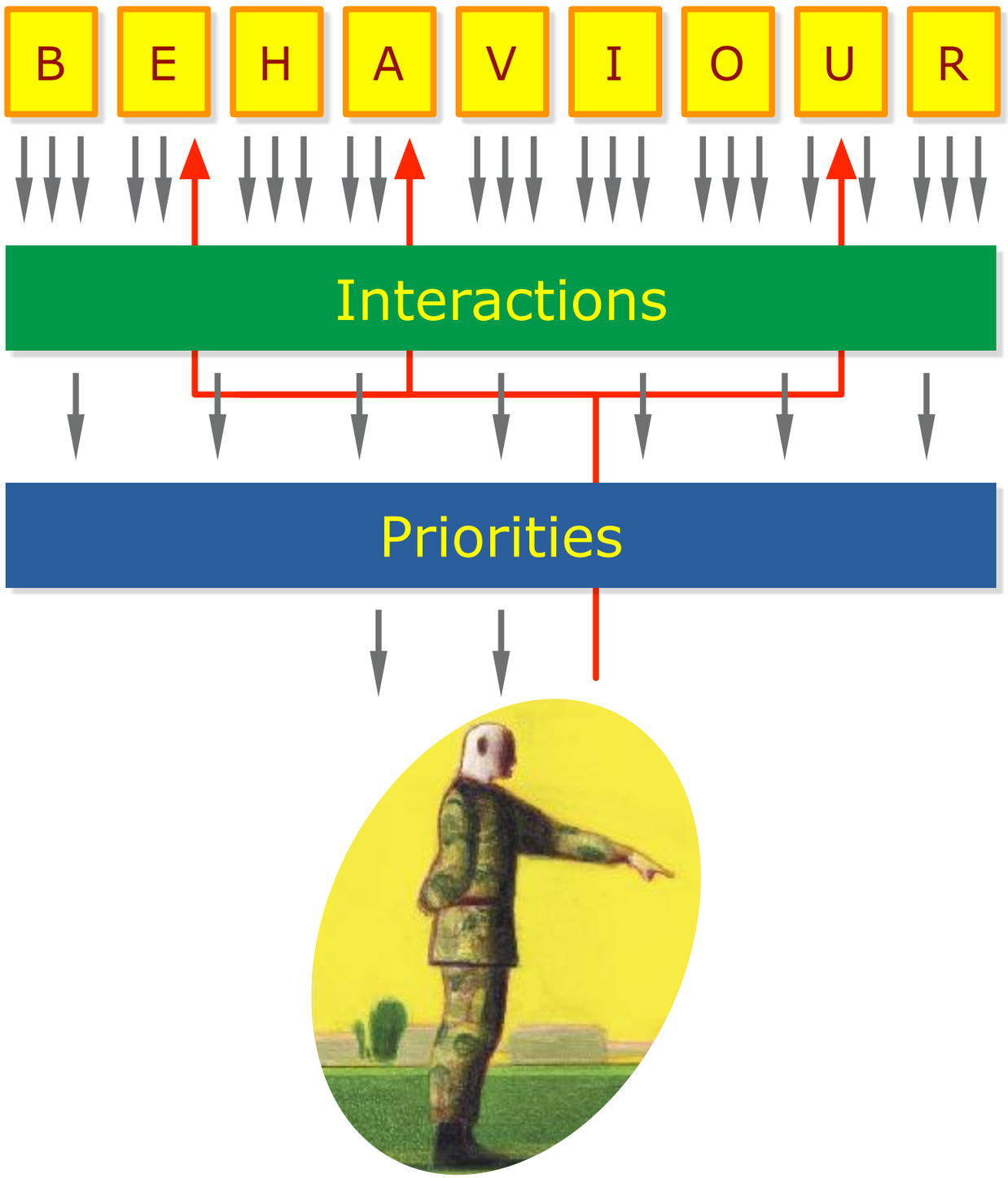
$$\gamma(B_1, \ldots, B_n) = (Q, P, \to) \qquad Q = \prod_{i=1}^{n} Q_i \qquad P = \bigcup_{i}^{\cdot} P_i$$

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

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

for each $a \in \gamma$.

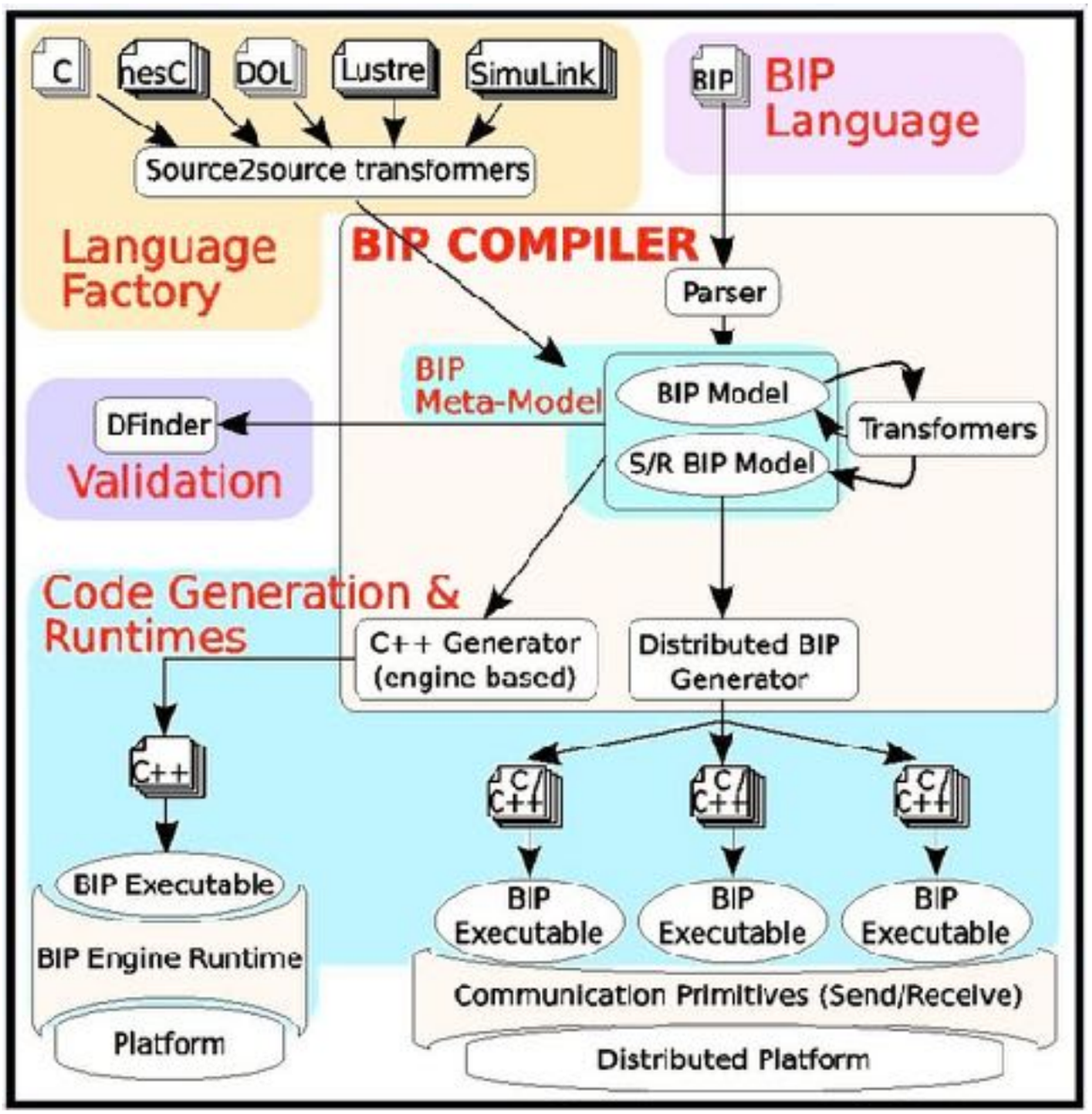
Engine-based execution



1. Components notify the Engine about enabled transitions.

2. The Engine picks an interaction and instructs the components about next actions to take.

Core BIP tool-set @ Verimag

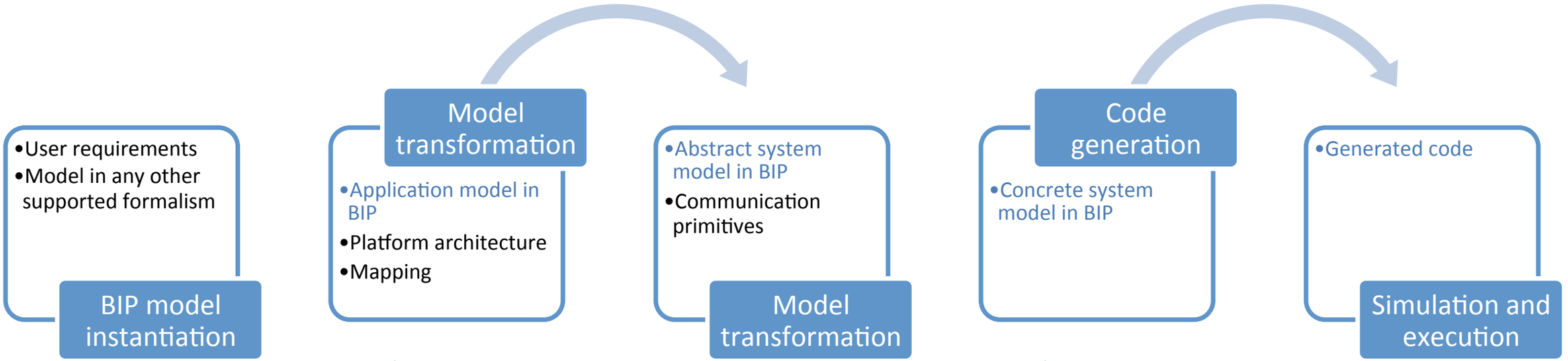


Ecore meta-model

Model-checking tools

C++ code generation

Rigorous System Design flow



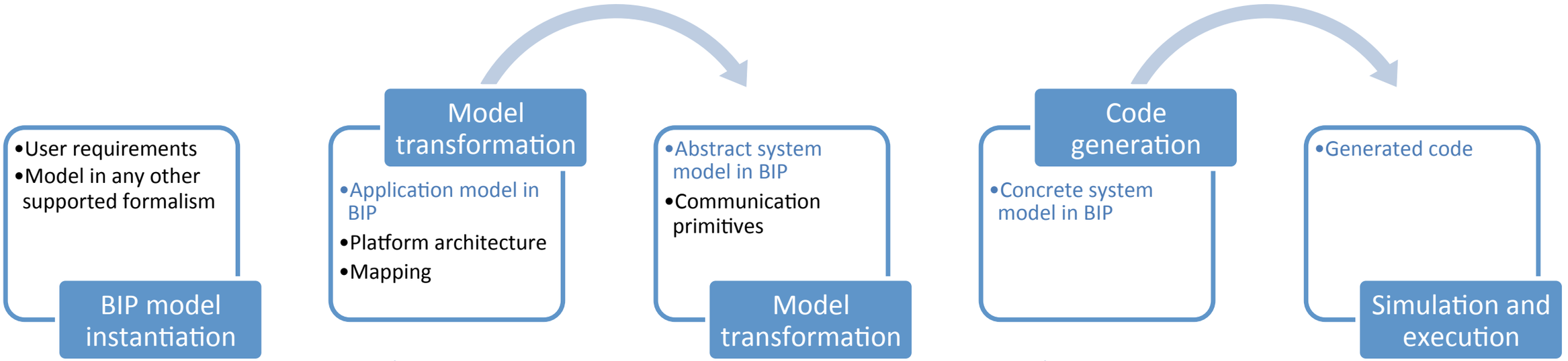
A series of semantics-preserving transformations

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

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

Korinthische Ordnung

Kapitäl u. Basis vom Monument des Lysikrates zu Athen.

Kapitäl u. Basis vom Tempel der Athene zu Athen.

Ionische Ordnung

Kapitäl vom Tempel der Athene zu Priene.

Kapitäl vom Tempel am Ilissos zu Athen.

Korinthisch Römisch-Korinthisch.

Zu 1. 2. 3.

a Mutuli (Dielenkopfe)
b Triglyphen (Dreischlitze)
c Metopen
d Rinnchen
e Kannelirungen
f Sima (Rinnleiste)

1

Geison
Fries
Architrav
Abacus
Echinus
Säulenhals
Säulenschaft

2

3

4

5

6

7

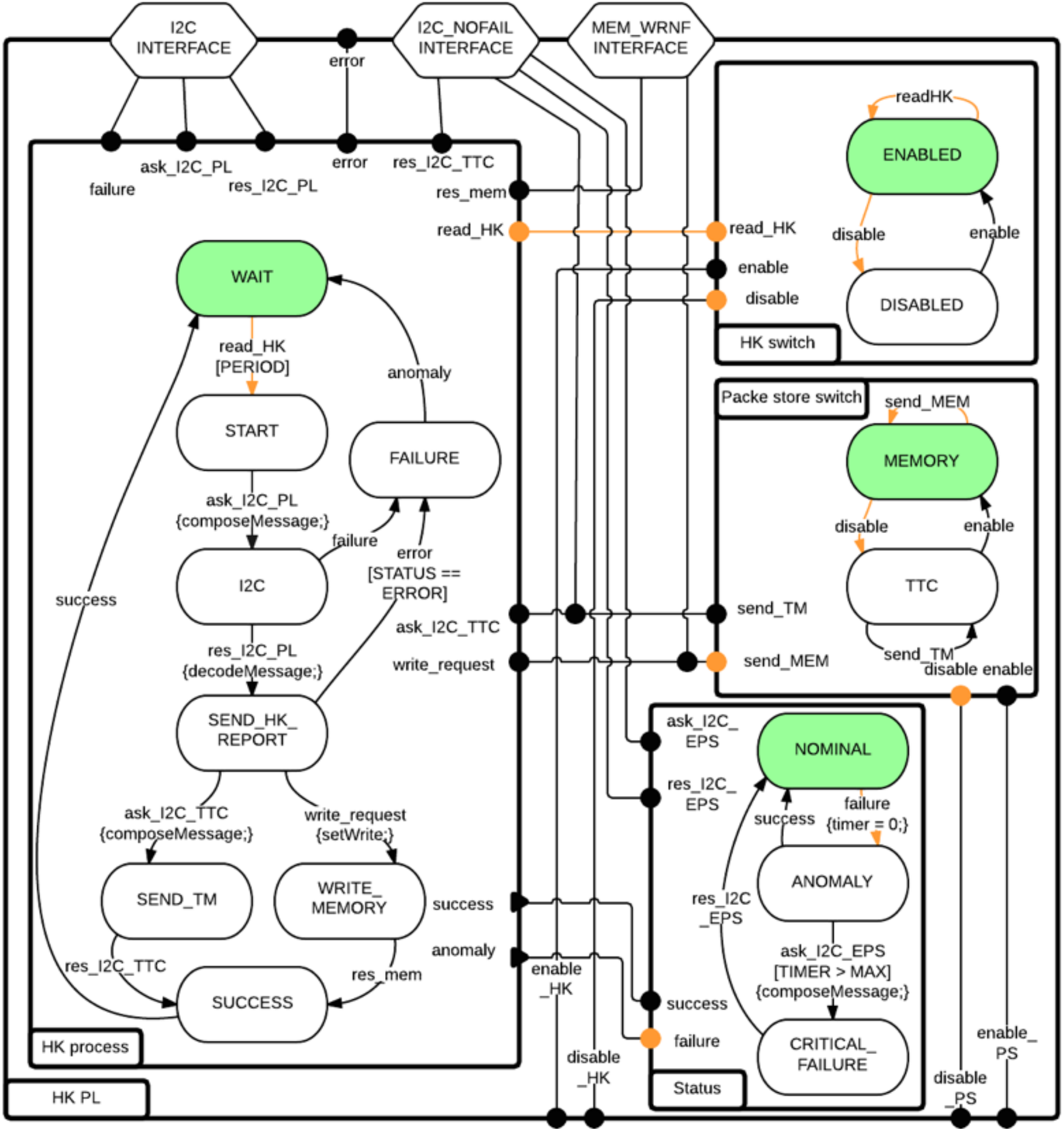
8

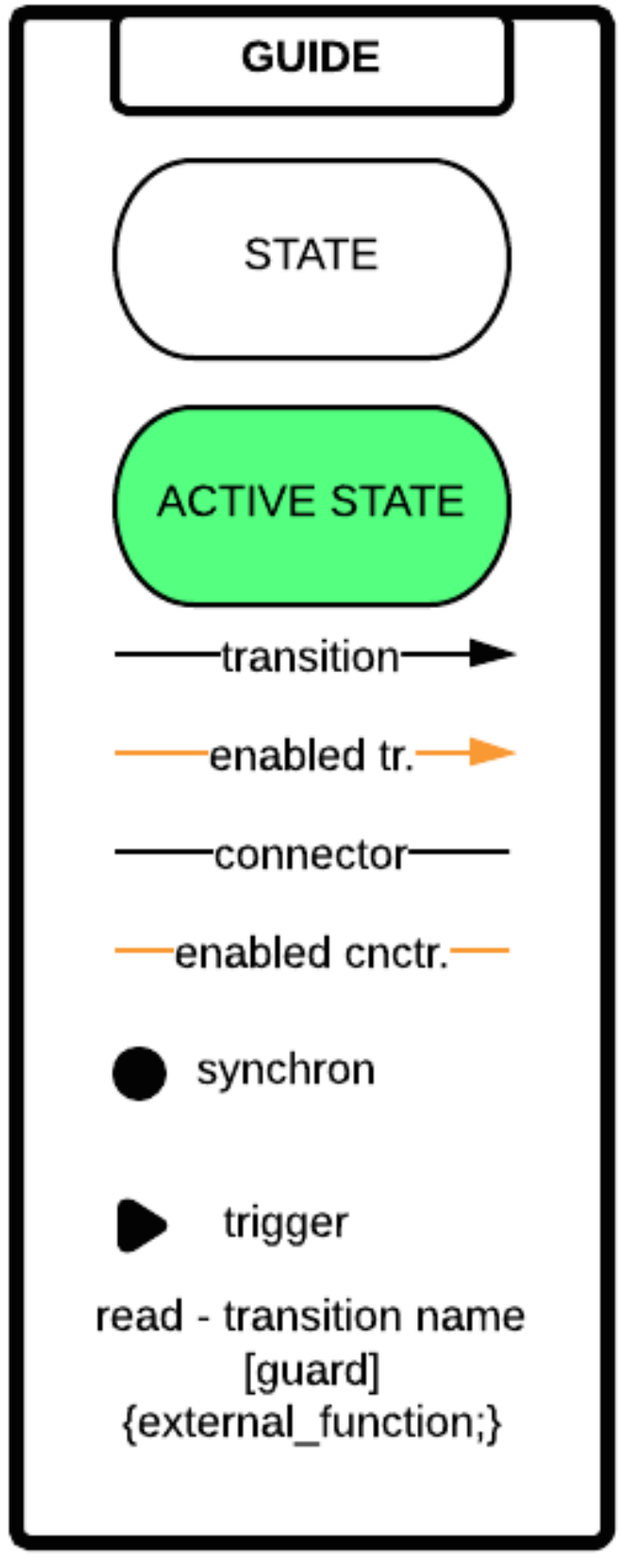
Kranzgesims
Fries
Architrav
Kapitäl
Säulenschaft
Basis

a Rinnleiste
b Kranzleiste
c Konsole
d Kymation
e Abacus
f Voluta
g Blattreihe
h Säulenhals
i Wulste
k Hohlkehlen
l Plättchen
m Plinthe

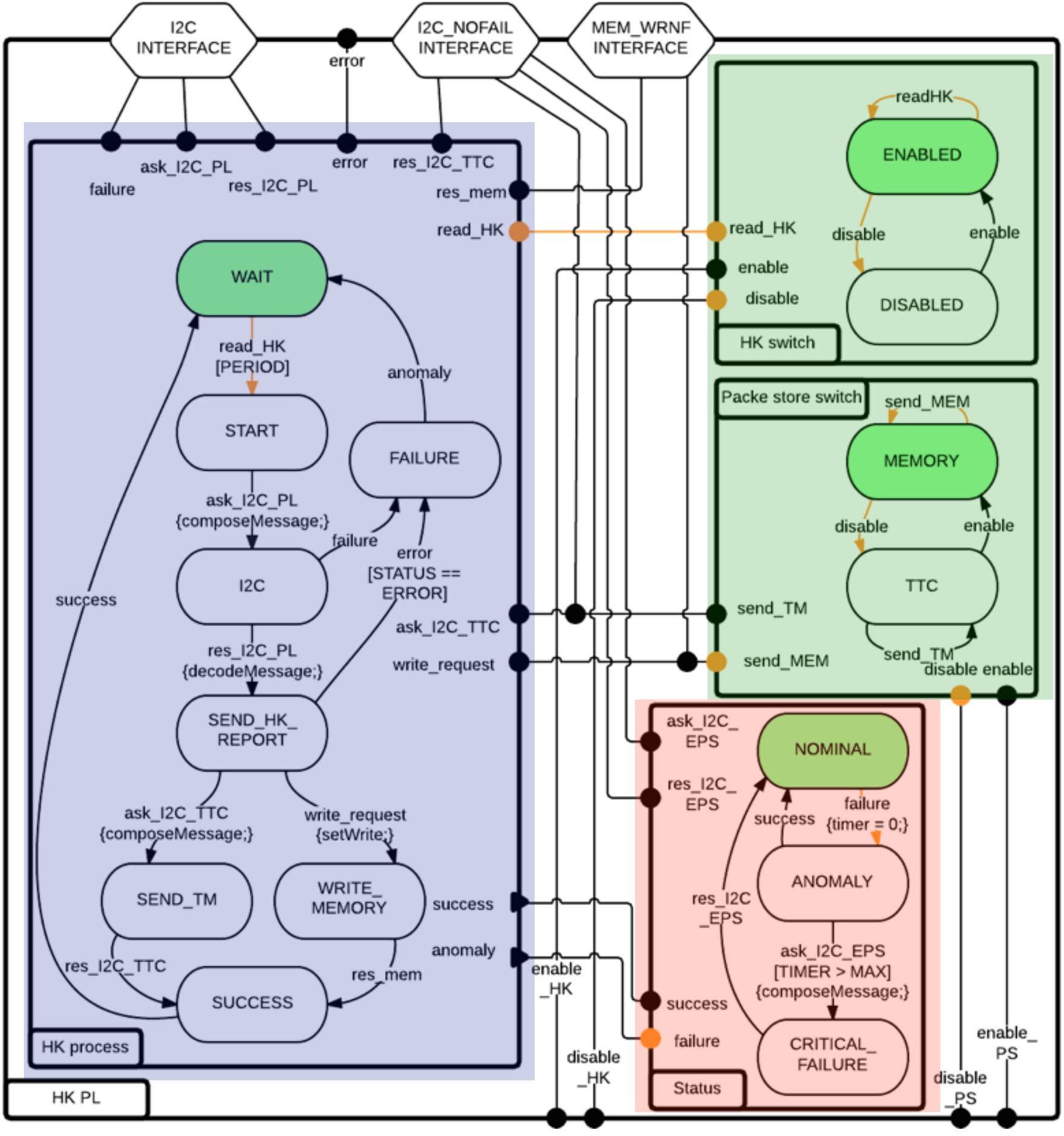
Vom Tempel in Pastum
Vom Partheson in Athen
Vom Tempel des Nemeischen Zeus
Vom Tempel am Ilissos in Athen
Vom Tempel d. Athene Polias in Priene
Vom Tempel d. Athene Polias in Athen
Vom Monument des Lysikrates in Athen
Vom Tempel d. Jupiter Stator in Rom

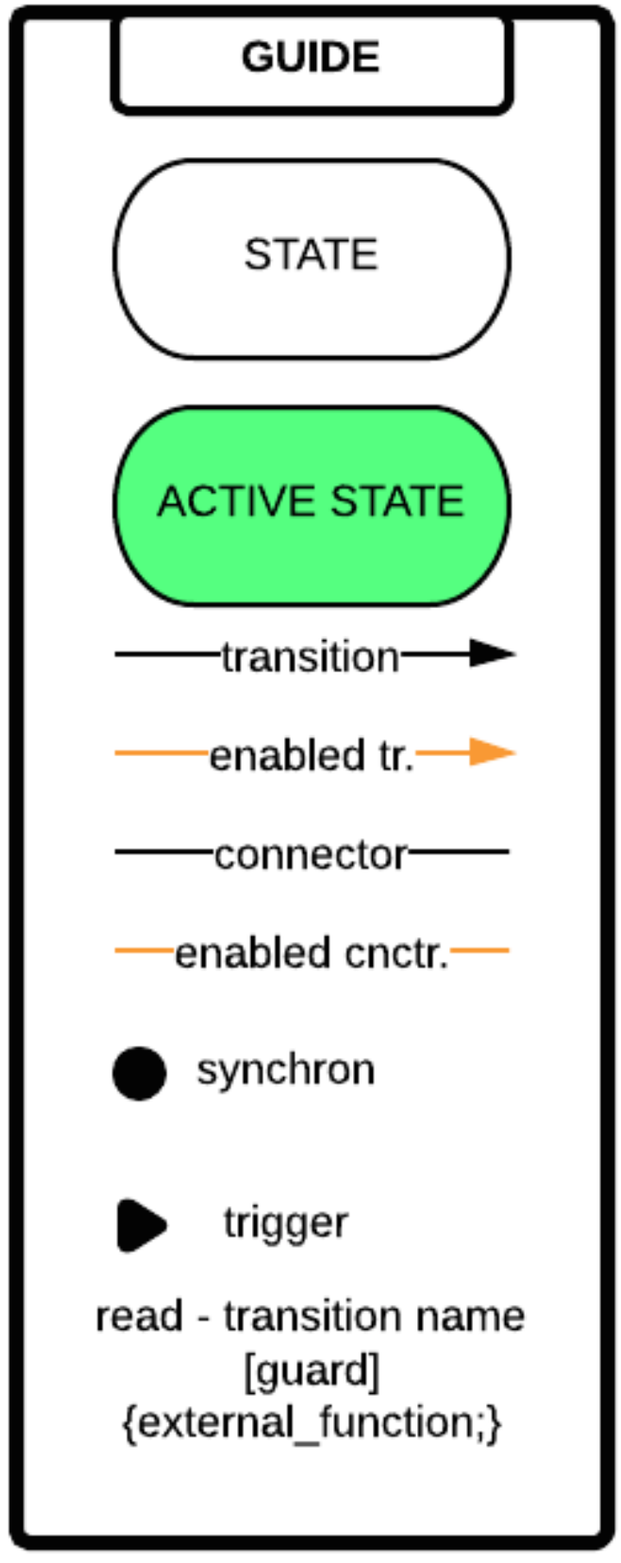
Dorische Säulenordnung
Ionische Säulenordnung
Korinthisch u. Römisch-Korinthisch





slide courtesy of
Marco Pagnamenta





slide courtesy of
Marco Pagnamenta

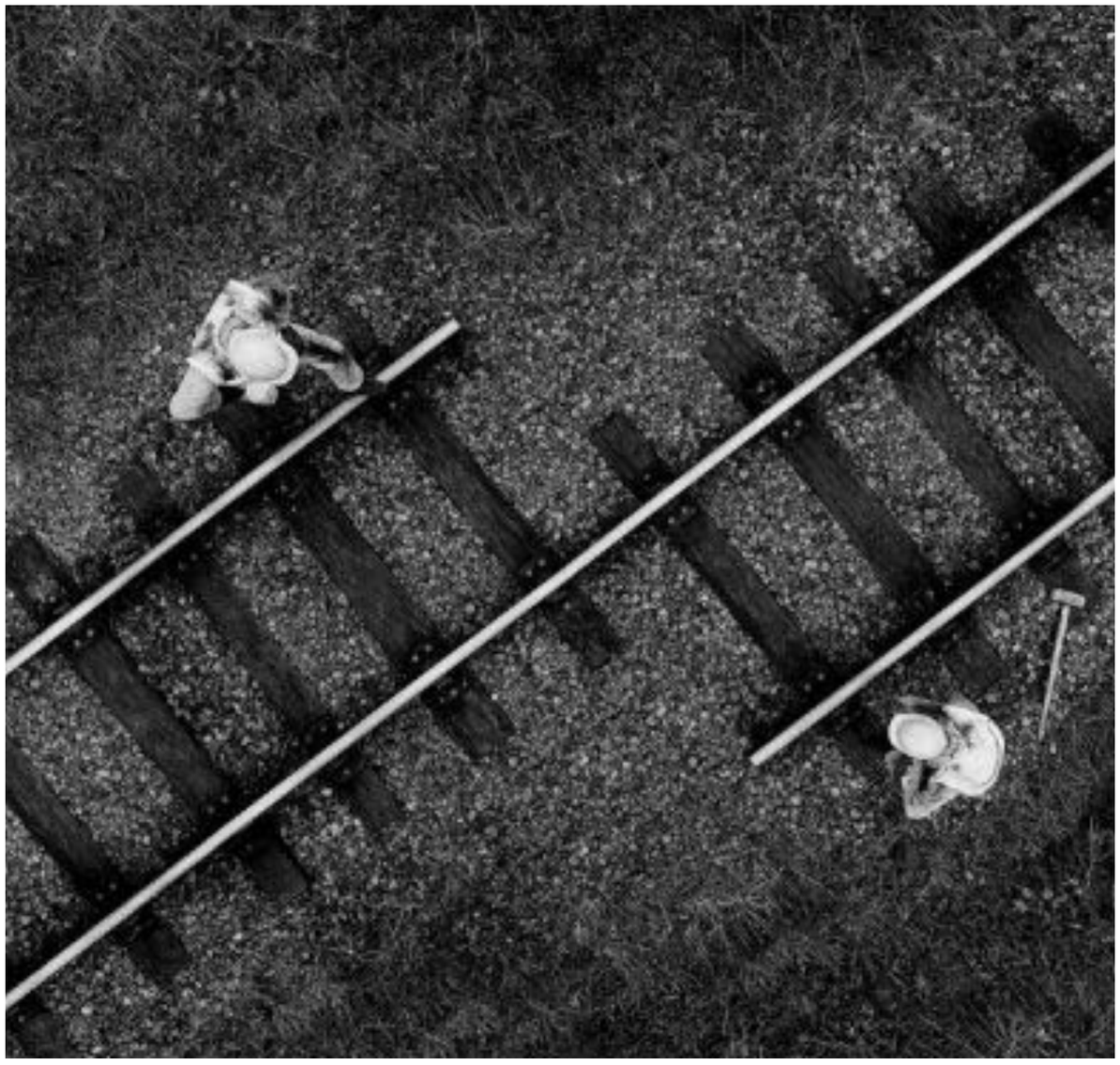
Theory of architectures

Design patterns for BIP

How to model?

How to combine?

How to specify?

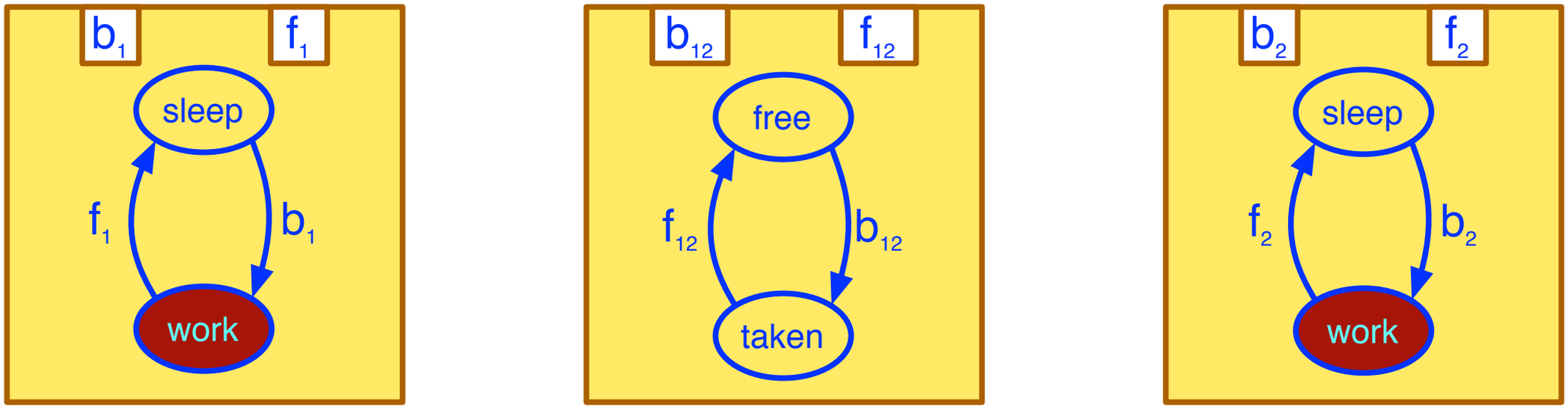


Architectures enforce characteristic properties. The crucial question is whether these are preserved by composition?

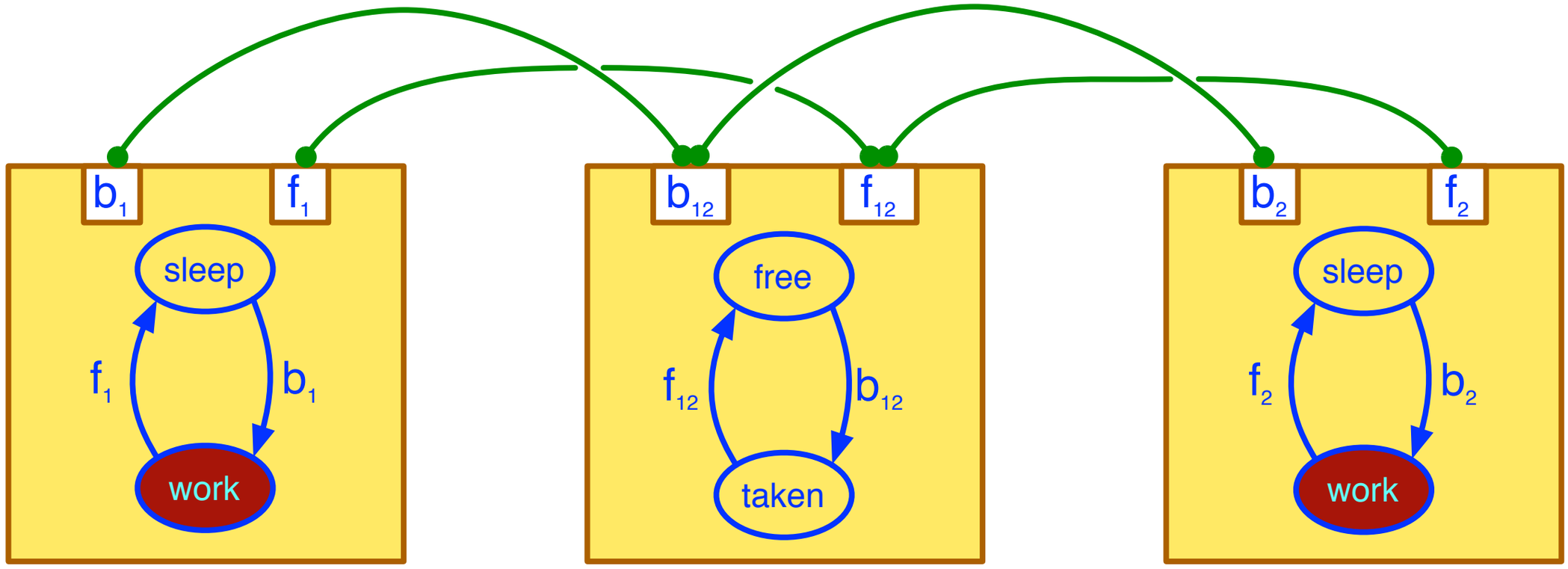
How to model?

Example: Lock

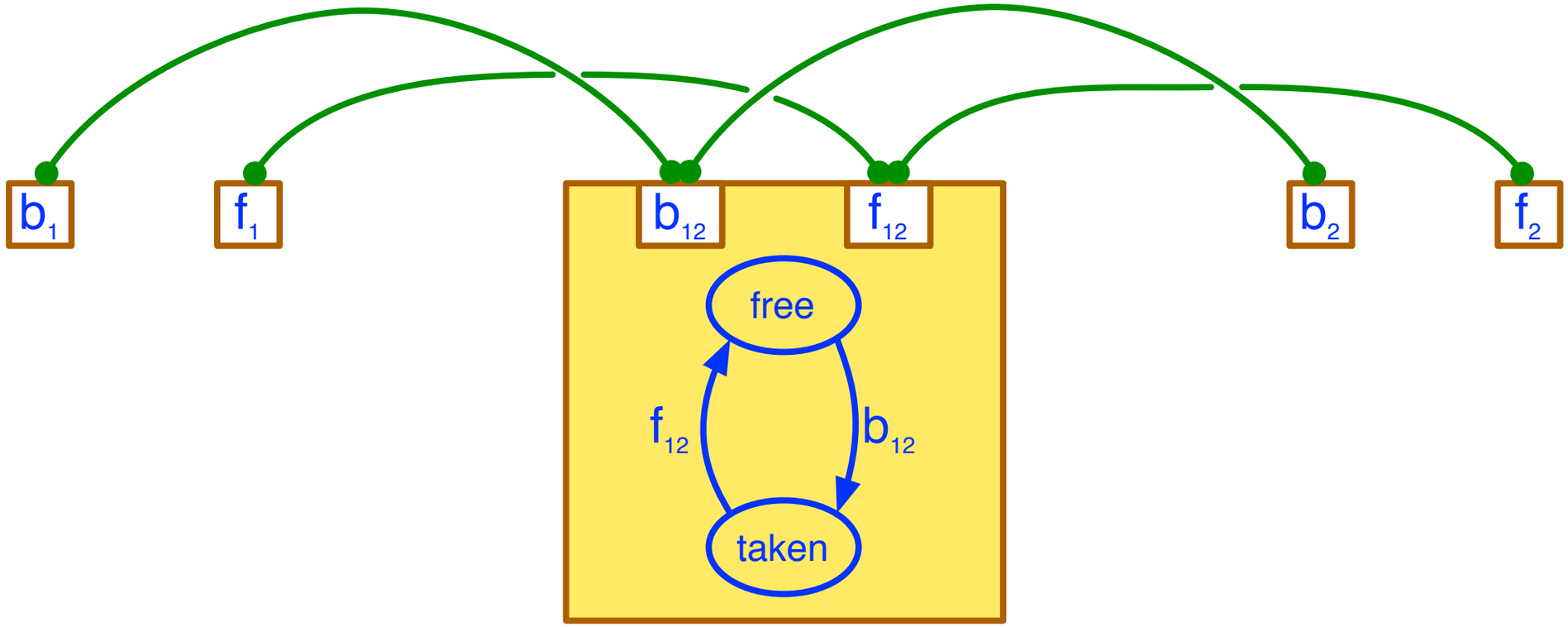
Example: Lock



Example: Lock



Example: Lock



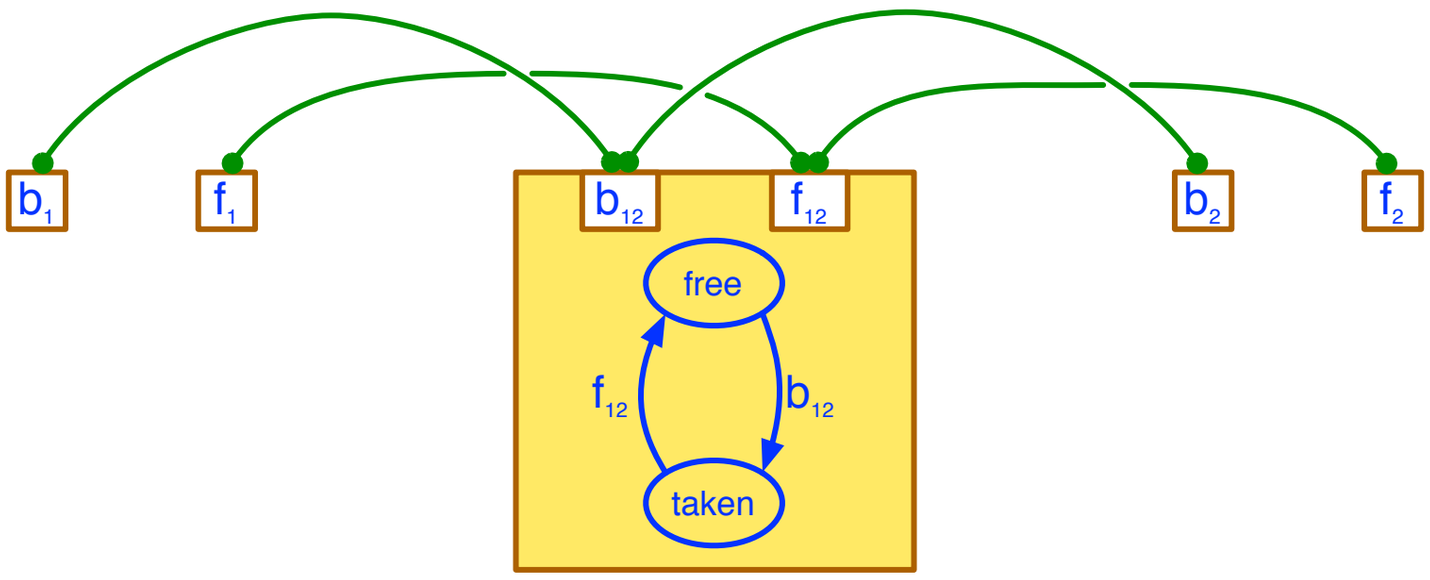
An architecture is...

$$A = (\mathcal{C}, P_A, \gamma)$$

Set of coordinating behaviours

Interface (ports)

Interaction model



…an operator…

$$A = (\mathcal{C}, P_A, \gamma)$$

…transforming

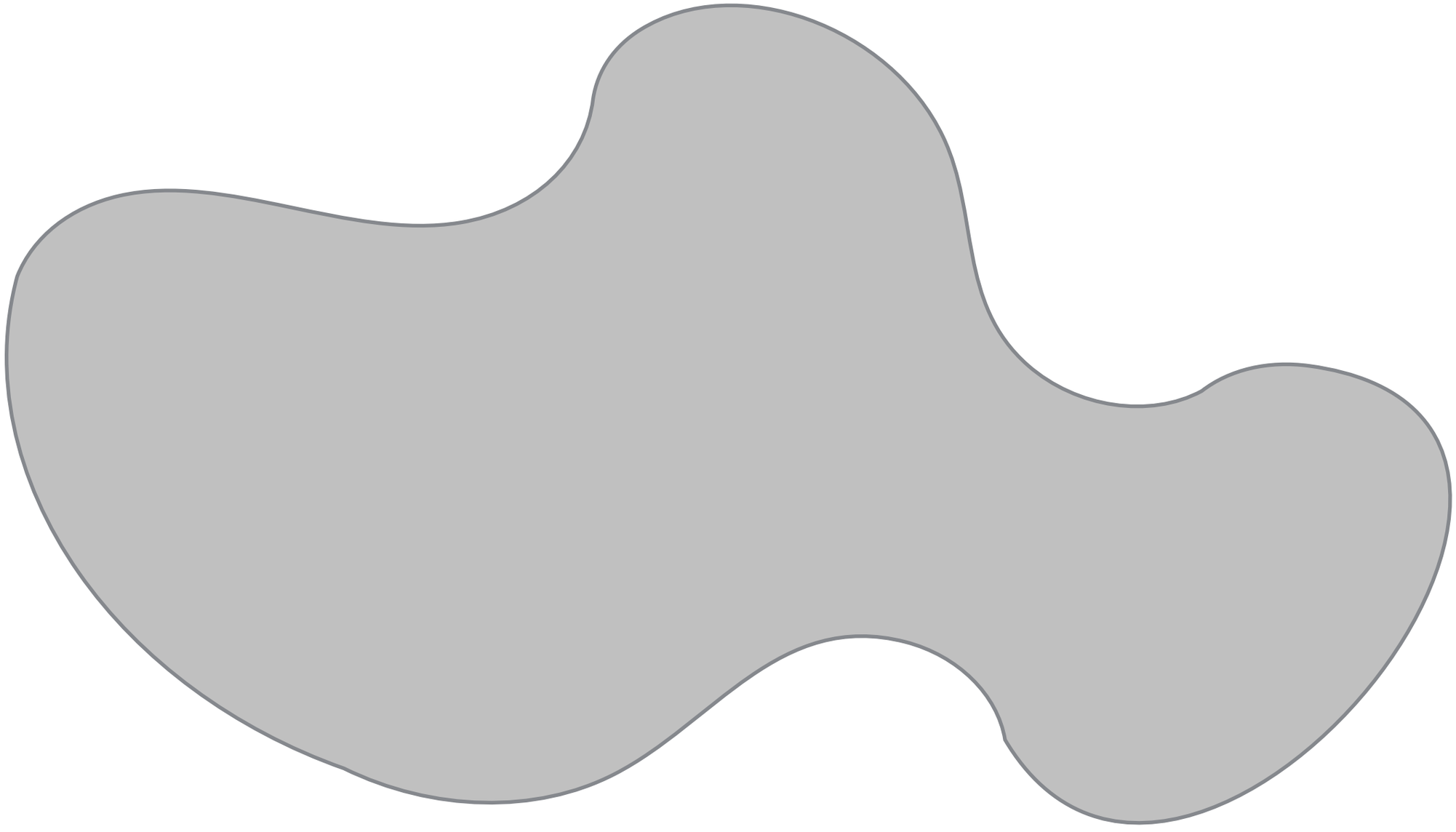
a set of components $\mathcal{B}$

into a composed BIP system

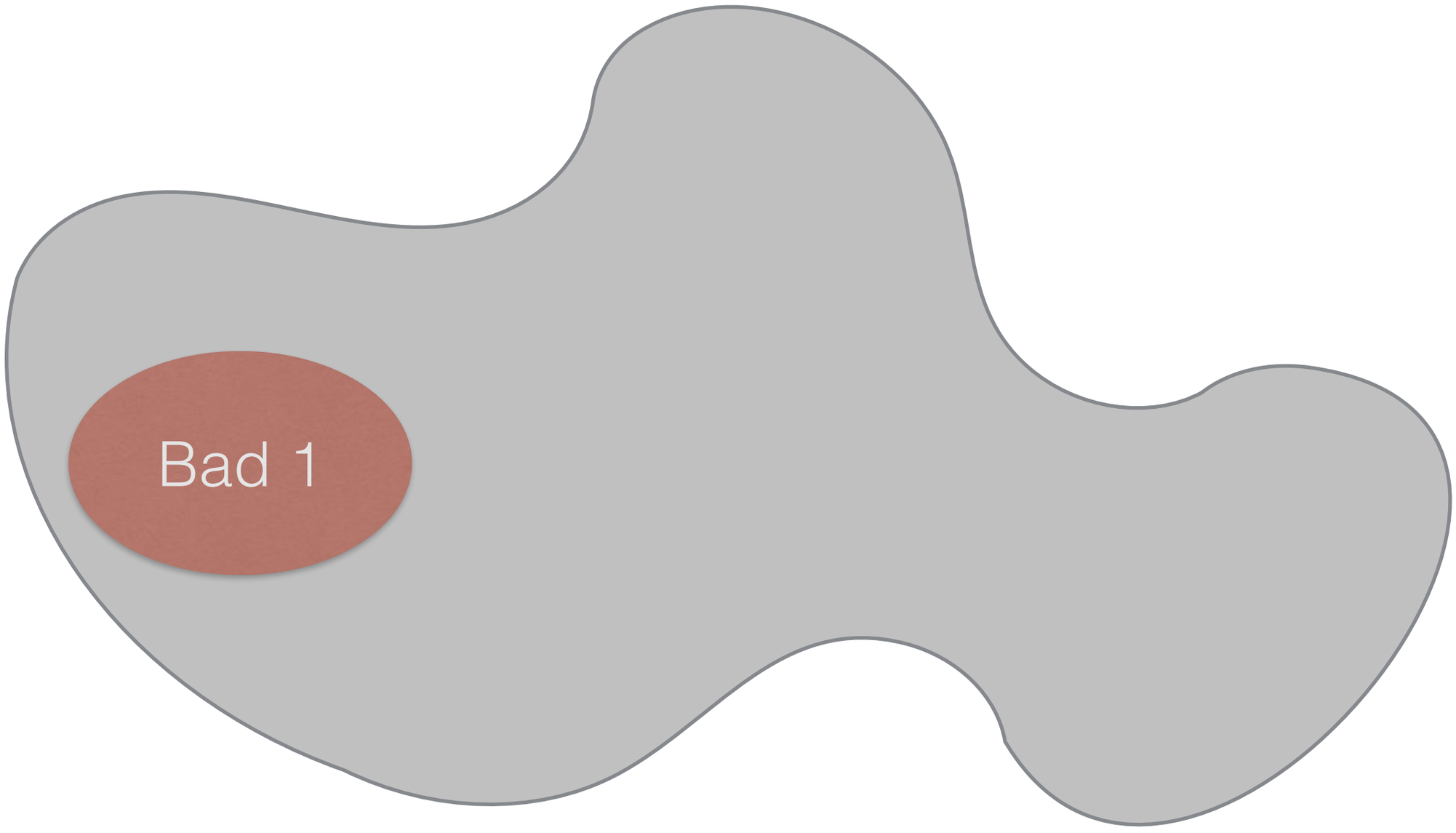
$$A(\mathcal{B}) \stackrel{def}{=} \Big(\gamma \ltimes P\Big)(\mathcal{B} \cup \mathcal{C}) \quad \Big/ \text{where } P \stackrel{def}{=} \bigcup_{B \in \mathcal{B} \cup \mathcal{C}} P_B \Big/$$

How to combine?

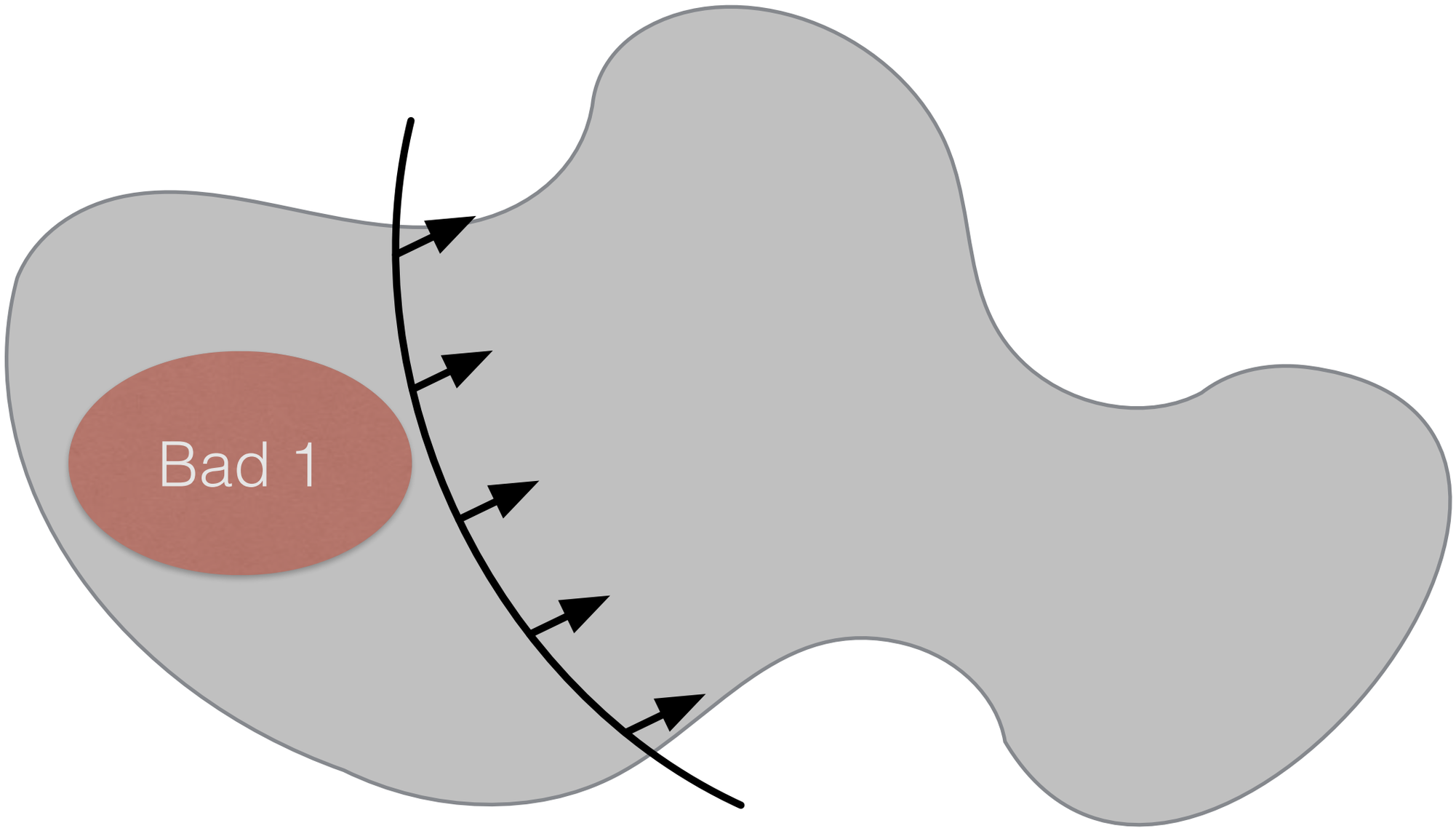
Constraints intuition



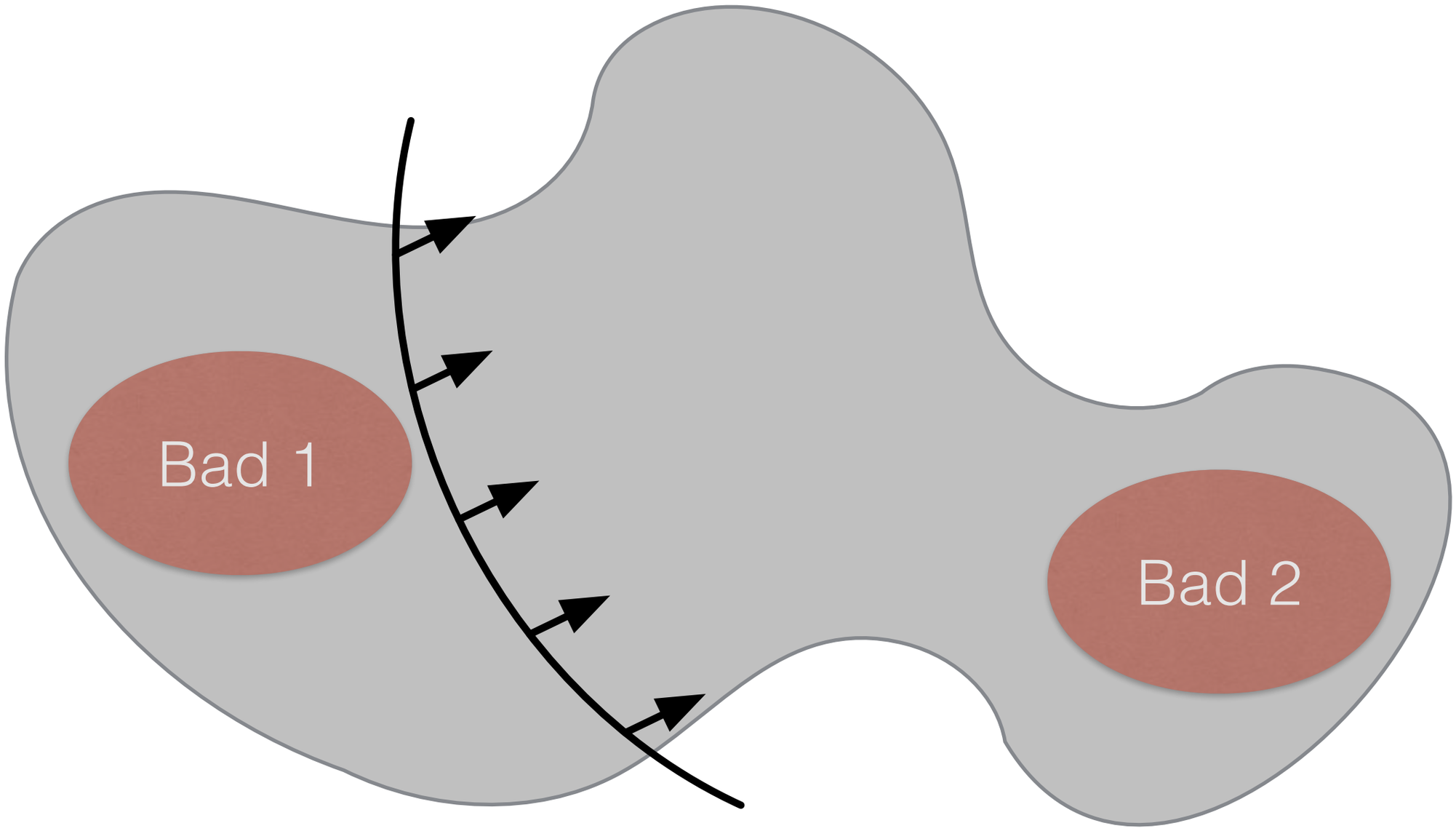
Constraints intuition



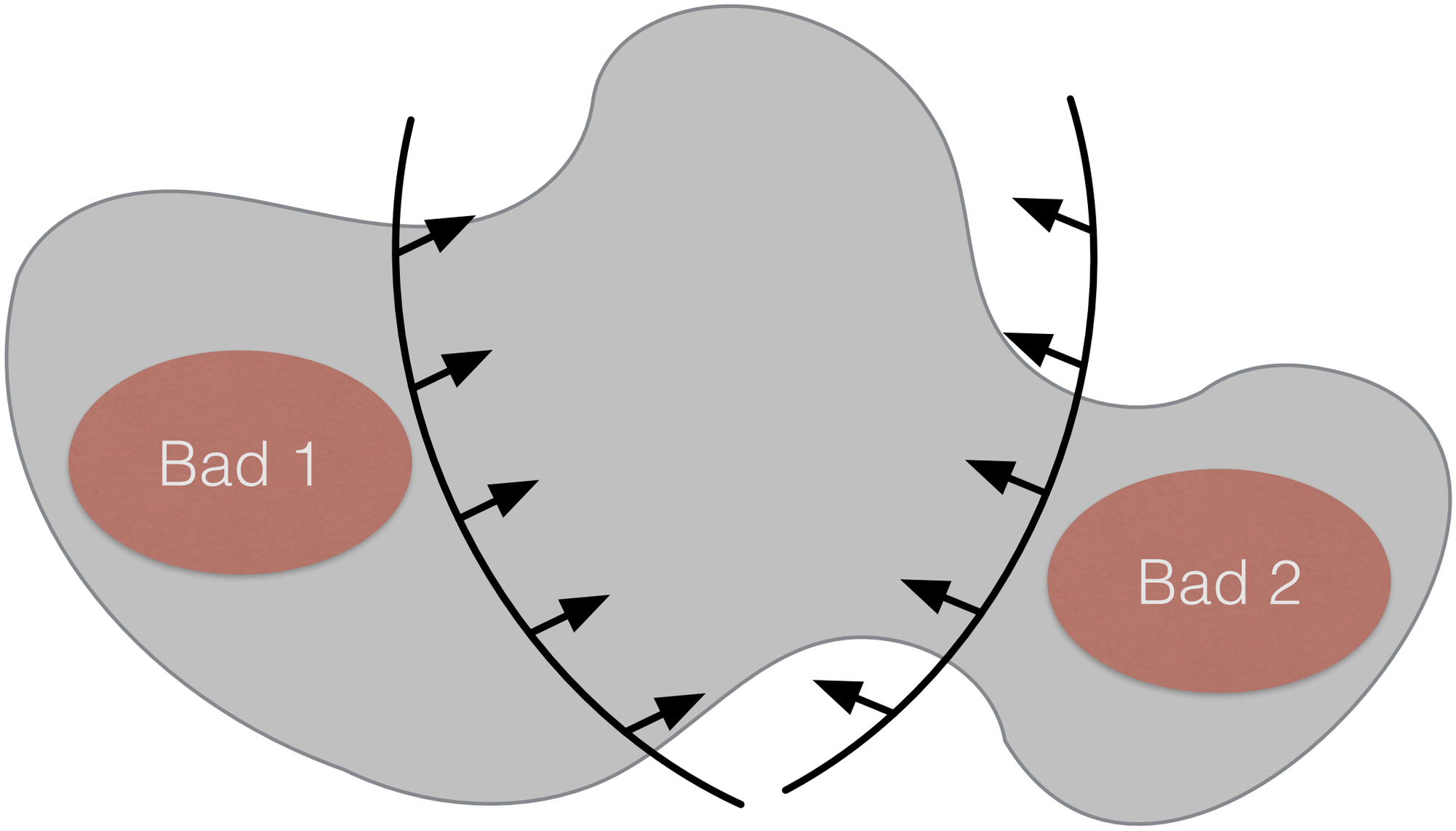
Constraints intuition



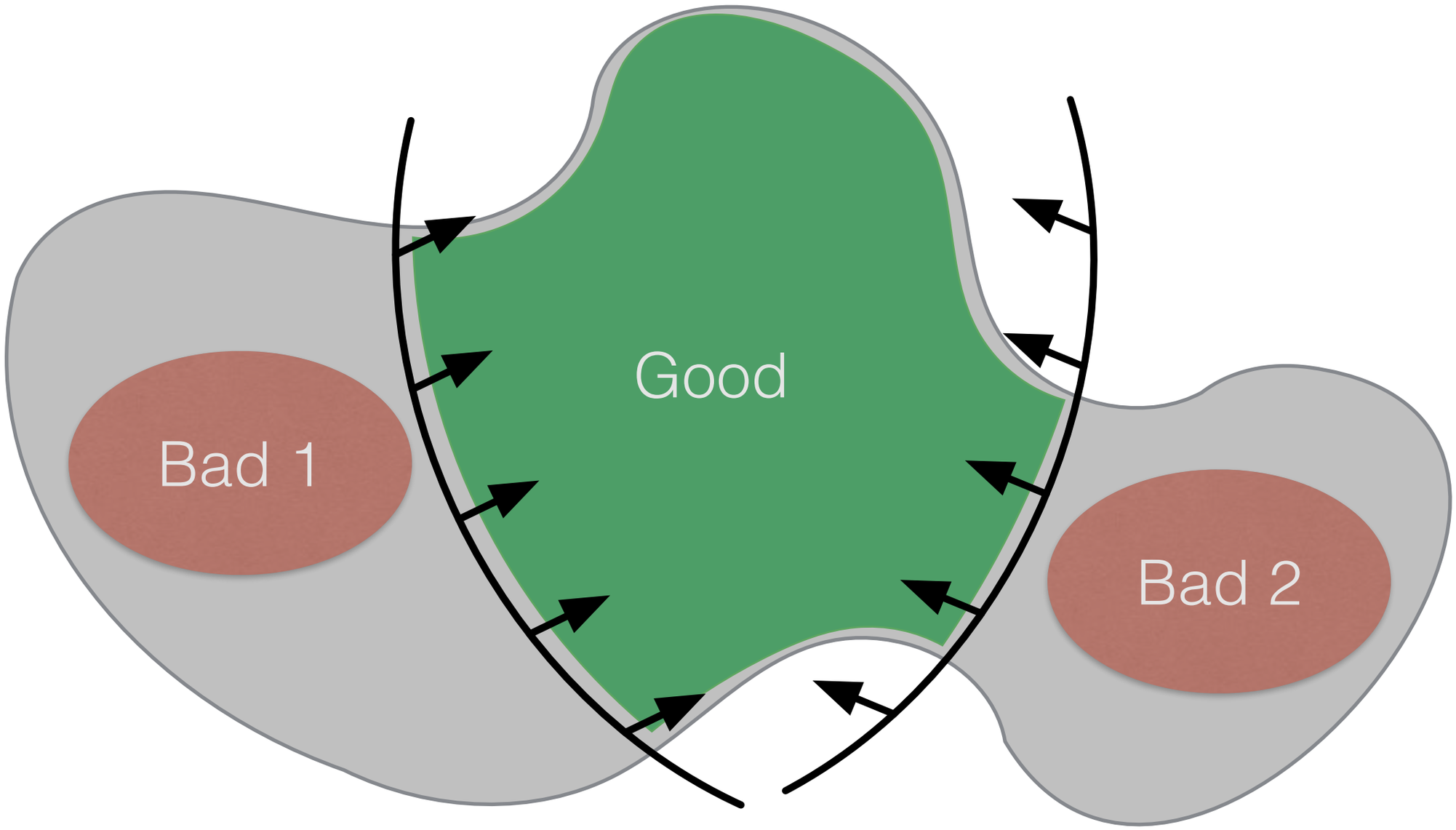
Constraints intuition



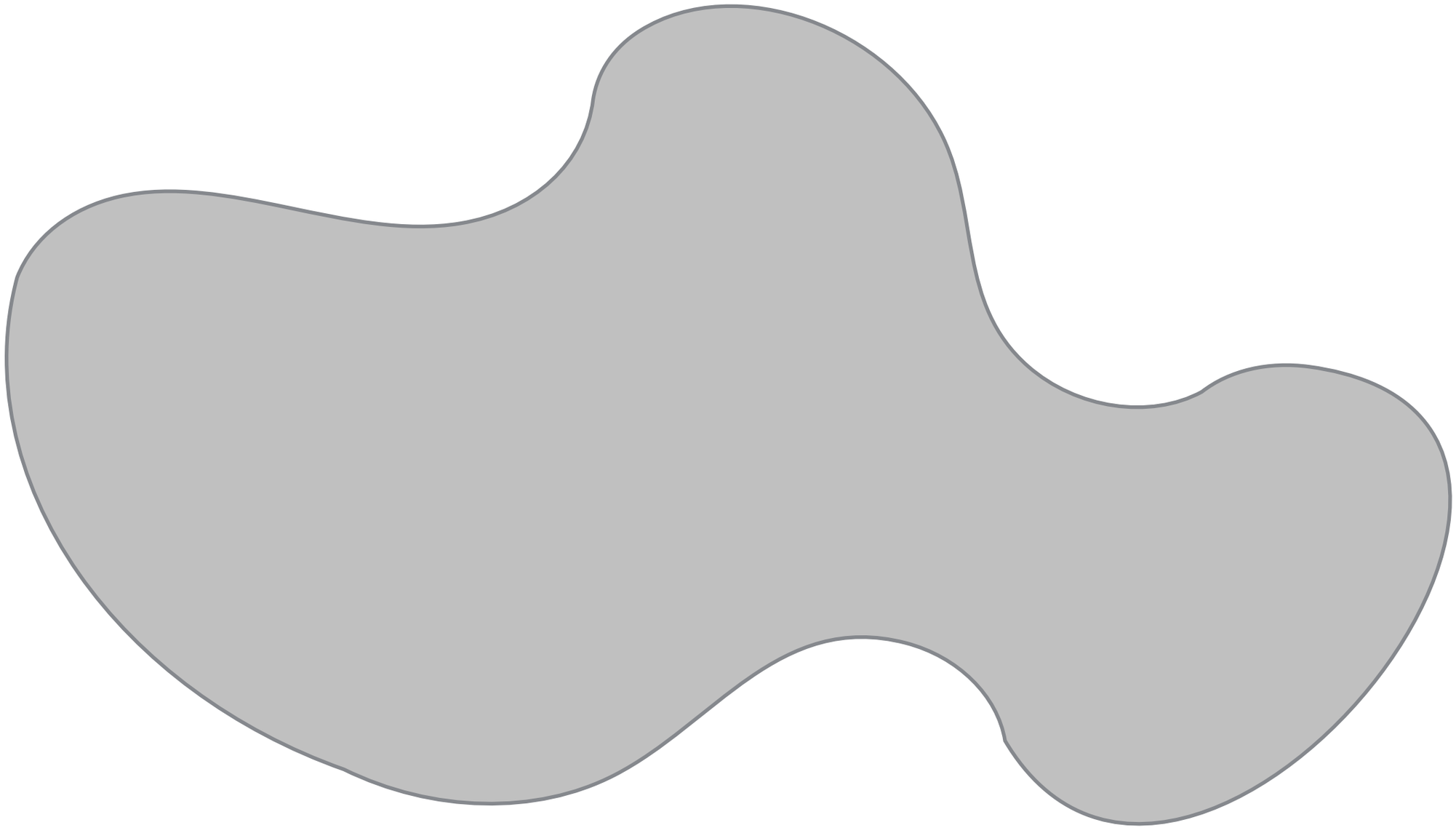
Constraints intuition



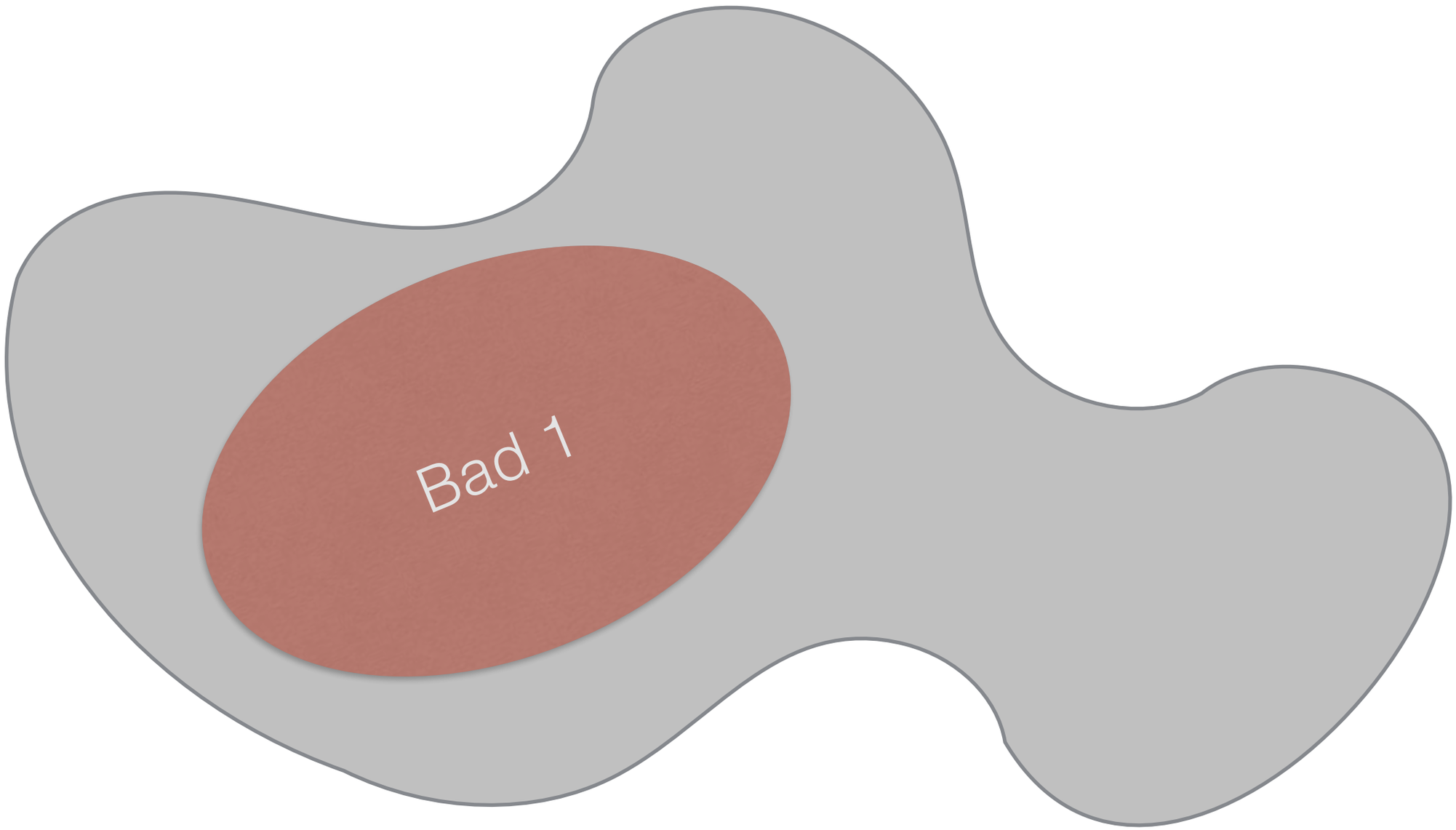
Constraints intuition



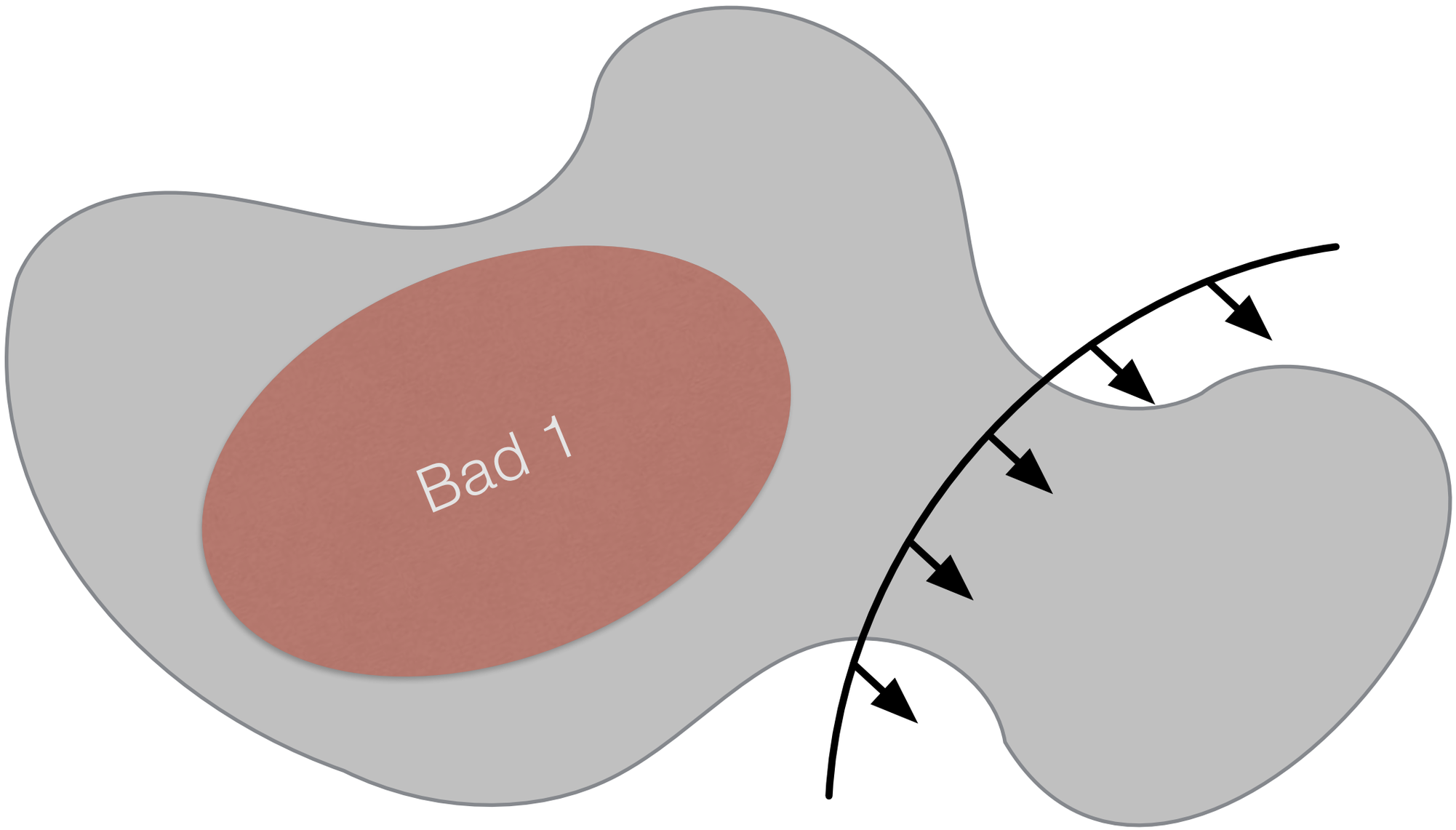
Limits of white magic



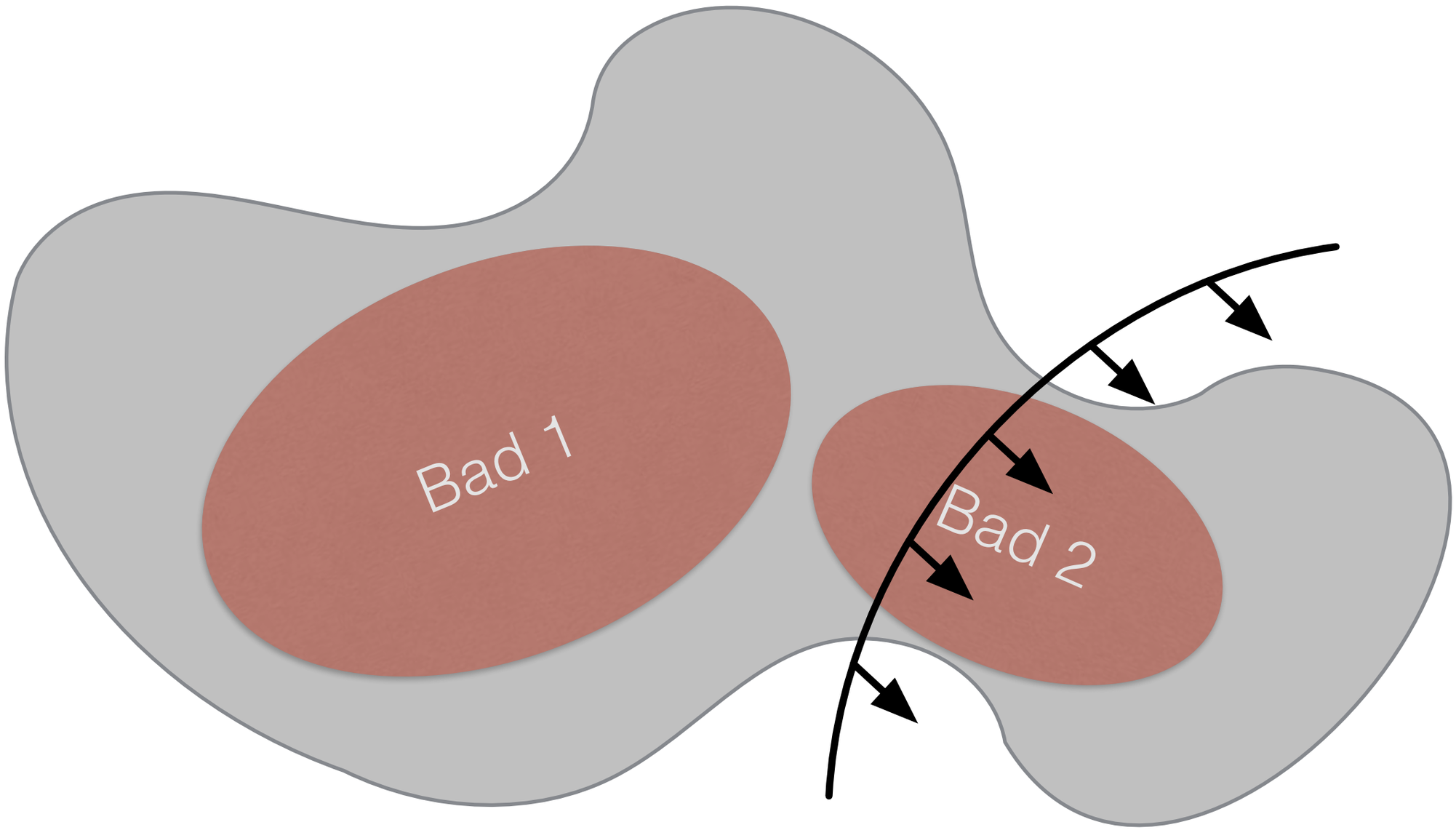
Limits of white magic



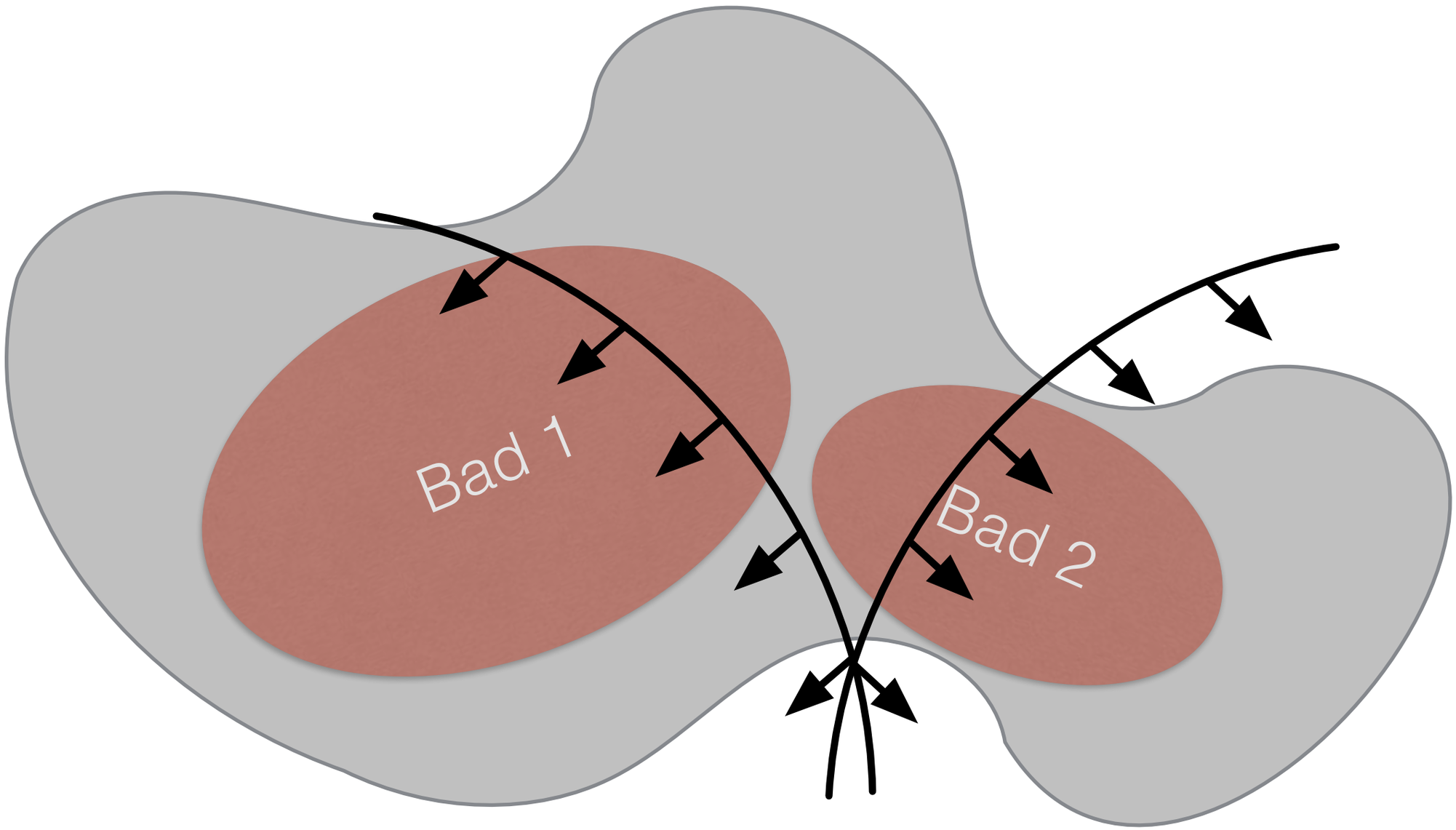
Limits of white magic



Limits of white magic



Limits of white magic



Main idea

Characteristic predicate for $\gamma \subseteq 2^P$

$$\varphi_\gamma : \mathbb{B}^P \to \mathbb{B} \qquad \varphi_\gamma \stackrel{\Delta}{=} \bigvee_{a \in \gamma} \left(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \right)$$

Interaction models to predicates and back

$$v : P \to \mathbb{B}, \qquad \varphi(v) = \mathtt{tt} \iff \{p \in P \,|\, v(p) = \mathtt{tt}\} \in \gamma$$

$$A_1 \oplus A_2 \stackrel{def}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_\varphi) \qquad \varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$$

Main idea

Characteristic predicate for $\gamma \subseteq 2^P$

$$\varphi_\gamma : \mathbb{B}^P \to \mathbb{B} \qquad \varphi_\gamma \stackrel{\Delta}{=} \bigvee_{a \in \gamma} \left(\bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \overline{p} \right)$$

Interaction models to predicates and back

$$v : P \to \mathbb{B}, \qquad \varphi(v) = \mathtt{tt} \iff \{p \in P \,|\, v(p) = \mathtt{tt}\} \in \gamma$$

$$A_1 \oplus A_2 \stackrel{def}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, P_1 \cup P_2, \gamma_\varphi) \qquad \varphi = \varphi_{\gamma_1} \wedge \varphi_{\gamma_2}$$

Main results: Safety

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

Safety = *"Bad states never occur"*

Main results: Liveness

$$\left.\begin{array}{r}\mathcal{A} \\ \text{pairwise non-interfering} \\ \text{live}\end{array}\right\} \Longrightarrow \bigoplus \mathcal{A} \quad \text{live}$$

$$\text{w.r.t. } \mathcal{B}$$

Liveness = *"Good states occur infinitely often"*

CubETH case study

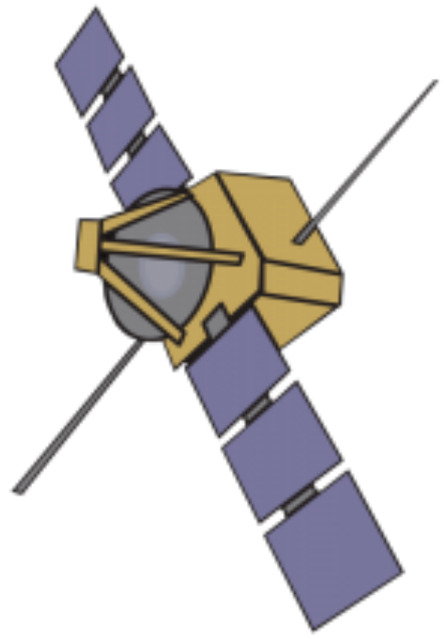
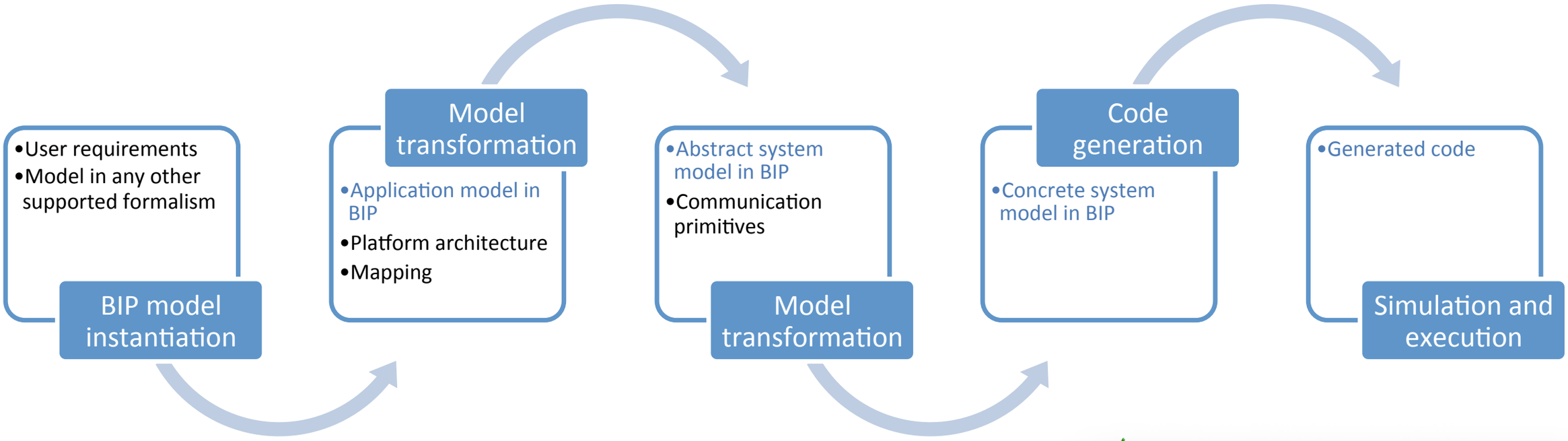


Table 1: Representative requirements for CDMS status and HK_PL

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.

Rigorous System Design flow



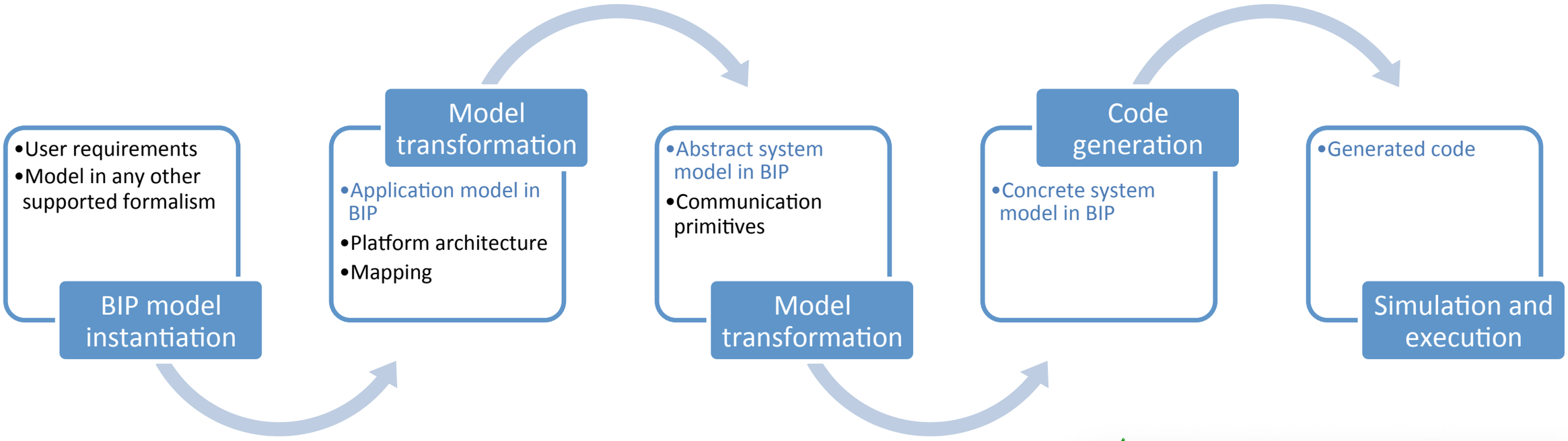
A series of semantics-preserving transformations

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

Rigorous System Design flow



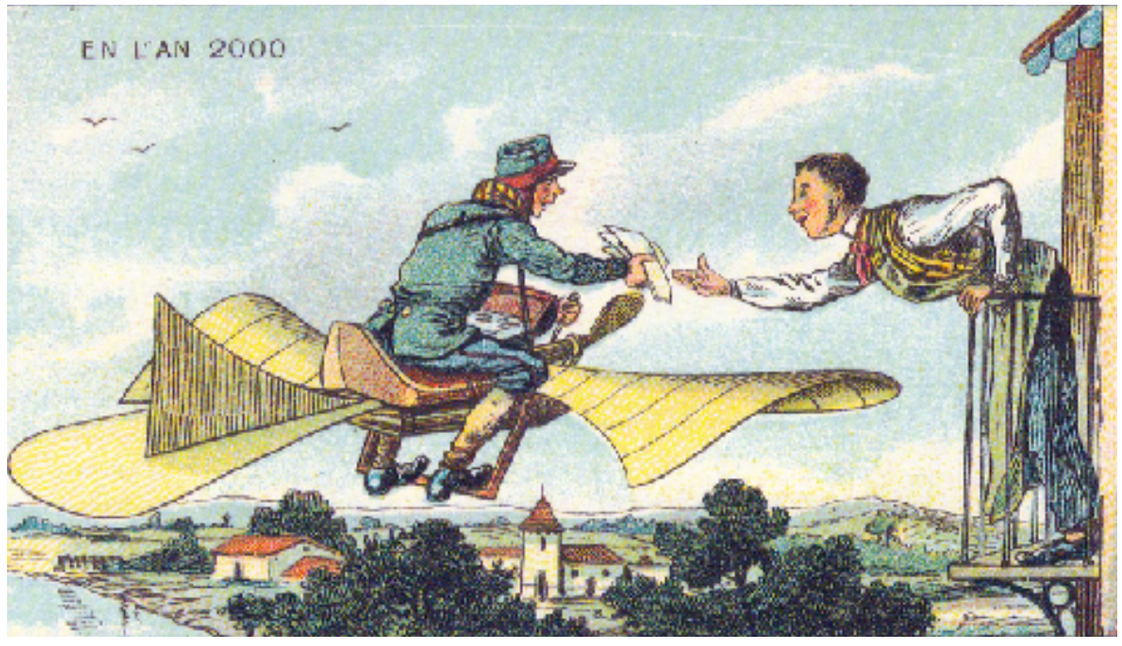
A series of semantics-preserving transformations

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

Future work



BIP

Dynamicity, distribution, self-adaptation

Architectures

Case studies, case studies, case studies and taxonomies (libraries)

Real-time, Synthesis, Dynamicity

DSLs for usability

Verification and proof of architectures and architecture styles

Tool support

JavaBIP...

JavaBIP

```
 1 @Ports({
 2   @Port(name="put", type=PortType.enforceable),
 3   @Port(name="get", type=PortType.enforceable)
 4 })
 5
 6 @ComponentType(initial="0", name="CommandBuffer")
 7 public class CommandBuffer {
 8   private LinkedList<Command> commandList;
 9   private int bufferSize;
10
11   public CommandBuffer(int bufferSize){
12     this.bufferSize = bufferSize;
13     this.commandList = new LinkedList<Command>();
14   }
15
16   @Transition(name="get", source="0", target="0", guard="notEmpty")
17   public void get() {
18     commandList.remove();
19   }
20
21   @Guard(name="notEmpty")
22   public boolean notEmpty() { return !commandList.isEmpty(); }
23
24   @Transition(name="put", source="0", target="0", guard="notFull")
25   public void put(@Data(name="input") Command cmd) {
26     commandList.add(cmd);
27   }
28
29   @Guard(name="notFull")
30   public boolean notFull() { return commandList.size() < bufferSize; }
31
32   @Data(name="command")
33   public Command getNextCommand() { return commandList.get(0); }
34 }
```

JavaBIP

```
 1 @Ports({
 2   @Port(name="put", type=PortType.enforceable),
 3   @Port(name="get", type=PortType.enforceable)
 4 })
 5
 6 @ComponentType(initial="0", name="CommandBuffer")
 7 public class CommandBuffer {
 8   private LinkedList<Command> commandList;
 9   private int bufferSize;
10
11   public CommandBuffer(int bufferSize){
12     this.bufferSize = bufferSize;
13     this.commandList = new LinkedList<Command>();
14   }
15
16   @Transition(name="get", source="0", target="0", guard="notEmpty")
17   public void get() {
18     commandList.remove();
19   }
20
21   @Guard(name="notEmpty")
22   public boolean notEmpty() { return !commandList.isEmpty(); }
23
24   @Transition(name="put", source="0", target="0", guard="notFull")
25   public void put(@Data(name="input") Command cmd) {
26     commandList.add(cmd);
27   }
28
29   @Guard(name="notFull")
30   public boolean notFull() { return commandList.size() < bufferSize; }
31
32   @Data(name="command")
33   public Command getNextCommand() { return commandList.get(0); }
34 }
```

Ports

JavaBIP

```
 1 @Ports({
 2   @Port(name="put", type=PortType.enforceable),
 3   @Port(name="get", type=PortType.enforceable)
 4 })
 5
 6 @ComponentType(initial="0", name="CommandBuffer")
 7 public class CommandBuffer {
 8   private LinkedList<Command> commandList;
 9   private int bufferSize;
10
11   public CommandBuffer(int bufferSize){
12     this.bufferSize = bufferSize;
13     this.commandList = new LinkedList<Command>();
14   }
15
16   @Transition(name="get", source="0", target="0", guard="notEmpty")
17   public void get() {
18     commandList.remove();
19   }
20
21   @Guard(name="notEmpty")
22   public boolean notEmpty() { return !commandList.isEmpty(); }
23
24   @Transition(name="put", source="0", target="0", guard="notFull")
25   public void put(@Data(name="input") Command cmd) {
26     commandList.add(cmd);
27   }
28
29   @Guard(name="notFull")
30   public boolean notFull() { return commandList.size() < bufferSize; }
31
32   @Data(name="command")
33   public Command getNextCommand() { return commandList.get(0); }
34 }
```

States

JavaBIP

```
 1 @Ports({
 2   @Port(name="put", type=PortType.enforceable),
 3   @Port(name="get", type=PortType.enforceable)
 4 })
 5
 6 @ComponentType(initial="0", name="CommandBuffer")
 7 public class CommandBuffer {
 8   private LinkedList<Command> commandList;
 9   private int bufferSize;
10
11   public CommandBuffer(int bufferSize){
12     this.bufferSize = bufferSize;
13     this.commandList = new LinkedList<Command>();
14   }
15
16   @Transition(name="get", source="0", target="0", guard="notEmpty")
17   public void get() {
18     commandList.remove();
19   }
20
21   @Guard(name="notEmpty")
22   public boolean notEmpty() { return !commandList.isEmpty(); }
23
24   @Transition(name="put", source="0", target="0", guard="notFull")
25   public void put(@Data(name="input") Command cmd) {
26     commandList.add(cmd);
27   }
28
29   @Guard(name="notFull")
30   public boolean notFull() { return commandList.size() < bufferSize; }
31
32   @Data(name="command")
33   public Command getNextCommand() { return commandList.get(0); }
34 }
```

Transitions

JavaBIP

```
 1 @Ports({
 2   @Port(name="put", type=PortType.enforceable),
 3   @Port(name="get", type=PortType.enforceable)
 4 })
 5
 6 @ComponentType(initial="0", name="CommandBuffer")
 7 public class CommandBuffer {
 8   private LinkedList<Command> commandList;
 9   private int bufferSize;
10
11   public CommandBuffer(int bufferSize){
12     this.bufferSize = bufferSize;
13     this.commandList = new LinkedList<Command>();
14   }
15
16   @Transition(name="get", source="0", target="0", guard="notEmpty")
17   public void get() {
18     commandList.remove();
19   }
20
21   @Guard(name="notEmpty")
22   public boolean notEmpty() { return !commandList.isEmpty(); }
23
24   @Transition(name="put", source="0", target="0", guard="notFull")
25   public void put(@Data(name="input") Command cmd) {
26     commandList.add(cmd);
27   }
28
29   @Guard(name="notFull")
30   public boolean notFull() { return commandList.size() < bufferSize; }
31
32   @Data(name="command")
33   public Command getNextCommand() { return commandList.get(0); }
34 }
```

Data

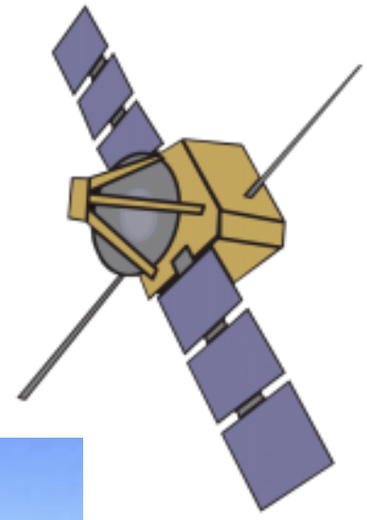
JavaBIP

```
@Ports({
  @Port(name="put", type=PortType.enforceabl
  @Port(name="get", type=PortType.enforceabl
})

@ComponentType(initial="0", name="CommandBuf
public class CommandBuffer {
  private LinkedList<Command> commandList;
  private int bufferSize;

  public CommandBuffer(int bufferSize){
    this.bufferSize = bufferSize;
    this.commandList = new LinkedList<Command>();
  }

  @Transition(name="get", source="0", target="0", guard="notEmpty")
  public void get() {
    commandList.remove();
  }

  @Guard(name="notEmpty")
  public boolean notEmpty() { return !commandList.isEmpty(); }

  @Transition(name="put", source="0", target="0", guard="notFull")
  public void put(@Data(name="input") Command cmd) {
    commandList.add(cmd);
  }

  @Guard(name="notFull")
  public boolean notFull() { return commandList.size() < bufferSize; }

  @Data(name="command")
  public Command getNextCommand() { return commandList.get(0); }
}
```

Connectors

```
<require>
  <effect id="give" specType="TCPReader"/>
    <causes>
      <port id="put" specType="CommandBuffer"/>
  </causes>
</require>
<accept>
  <effect id="give" specType="TCPReader"/>
   <causes>
      <port id="put" specType="CommandBuffer"/>
   </causes>
</accept>
```

Conclusion

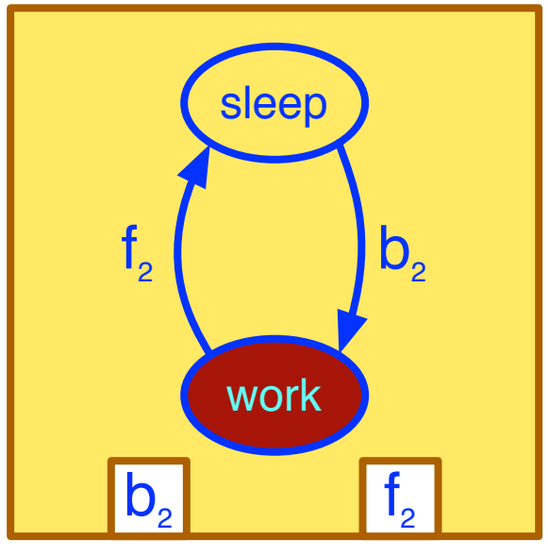


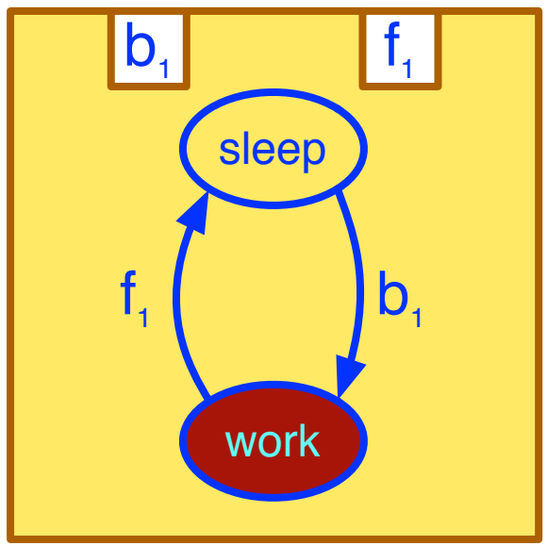
A lot of knowledge and experience is available for building reliable systems

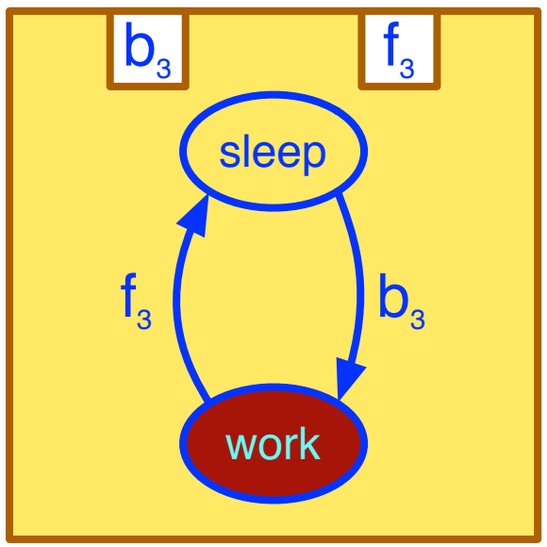
We must strive to expand these to general-purpose software!

Appendices

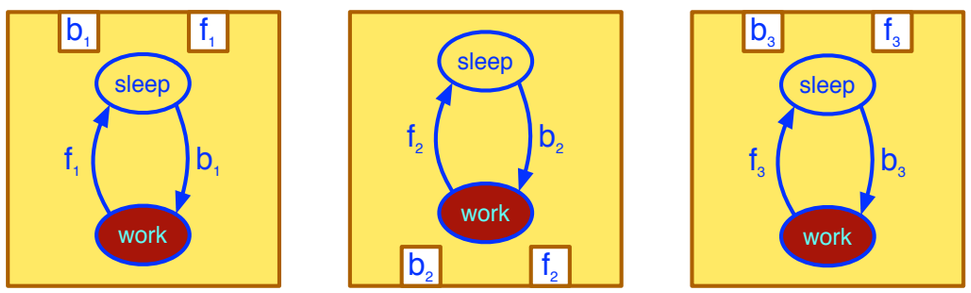
Example continued



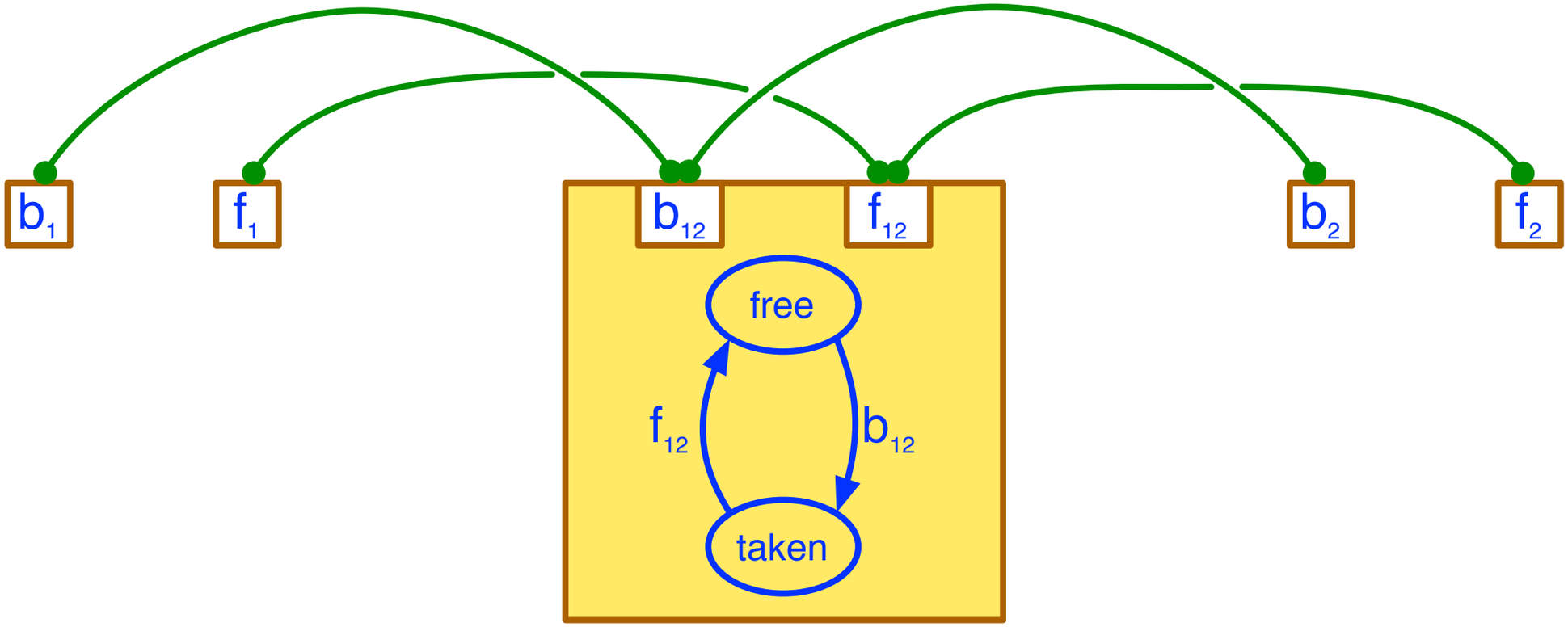


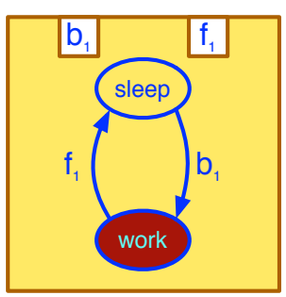


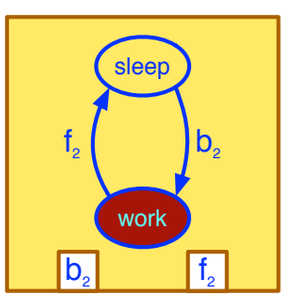
Example continued

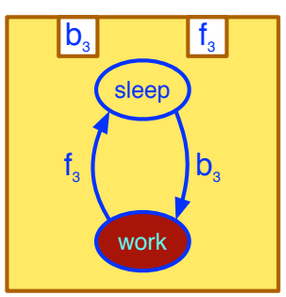


Example continued



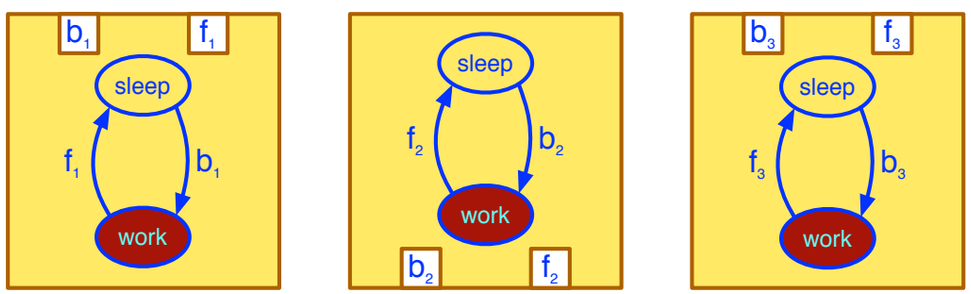


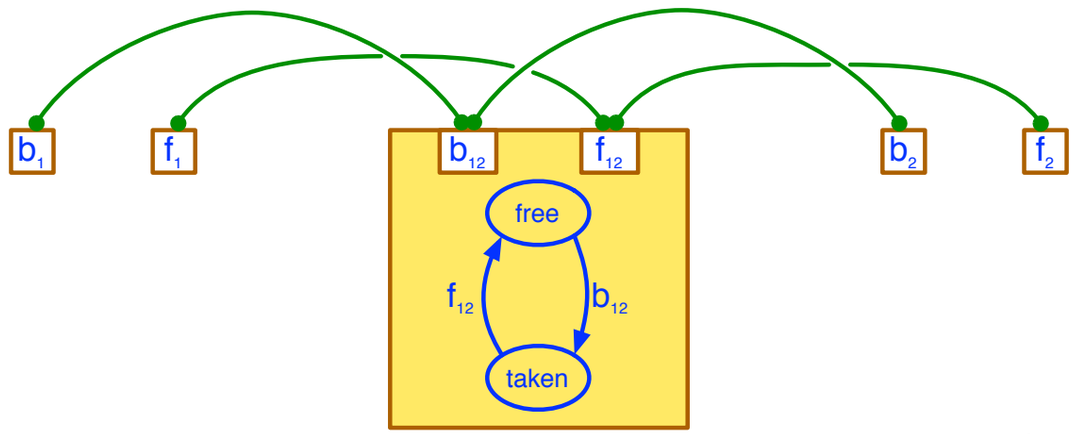




Example continued

$$\varphi_{\gamma_{12}} \equiv (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \wedge$$
$$(b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})$$

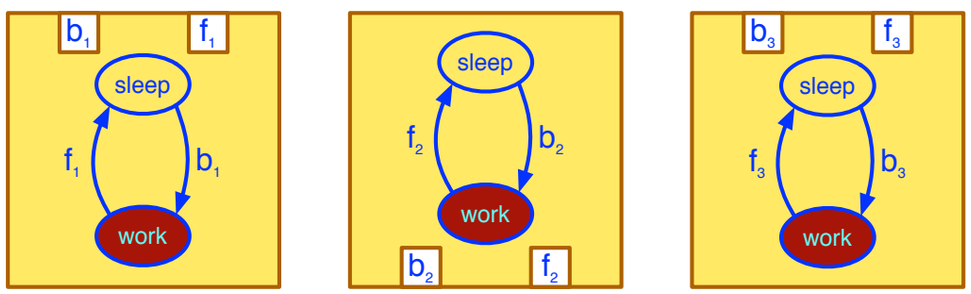


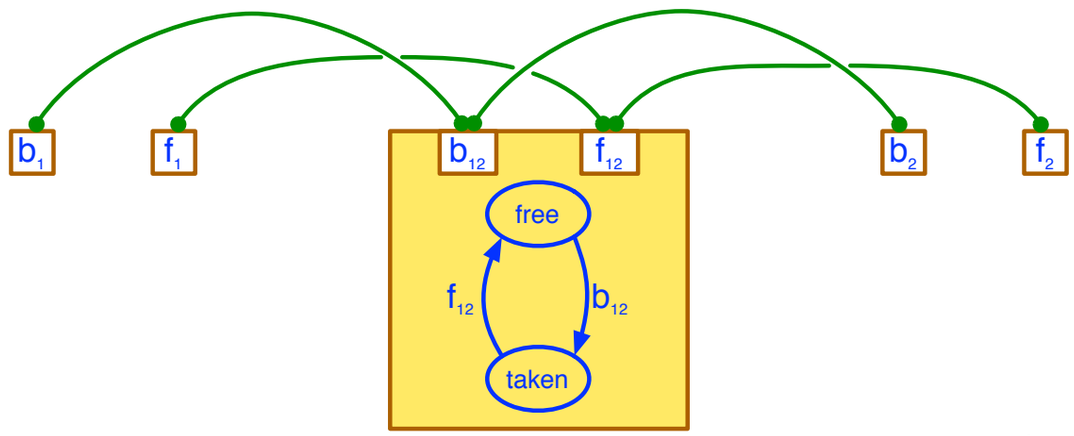


Example continued

$$\varphi_{\gamma_{12}} \equiv (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \wedge$$
$$(b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})$$

$$b_1 \Rightarrow b_{12} \wedge b_{13}\,, \quad f_1 \Rightarrow f_{12} \wedge f_{13}\,, \quad b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2\,, \quad f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2\,, \quad b_{12} \Rightarrow \overline{f_{12}}$$
$$b_2 \Rightarrow b_{12} \wedge b_{23}\,, \quad f_2 \Rightarrow f_{12} \wedge f_{23}\,, \quad b_{13} \Rightarrow b_1 \,\texttt{XOR}\, b_3\,, \quad f_{13} \Rightarrow f_1 \,\texttt{XOR}\, f_3\,, \quad b_{13} \Rightarrow \overline{f_{13}}$$
$$b_3 \Rightarrow b_{13} \wedge b_{23}\,, \quad f_3 \Rightarrow f_{13} \wedge f_{23}\,, \quad b_{23} \Rightarrow b_2 \,\texttt{XOR}\, b_3\,, \quad f_{23} \Rightarrow f_2 \,\texttt{XOR}\, f_3\,, \quad b_{23} \Rightarrow \overline{f_{23}}$$



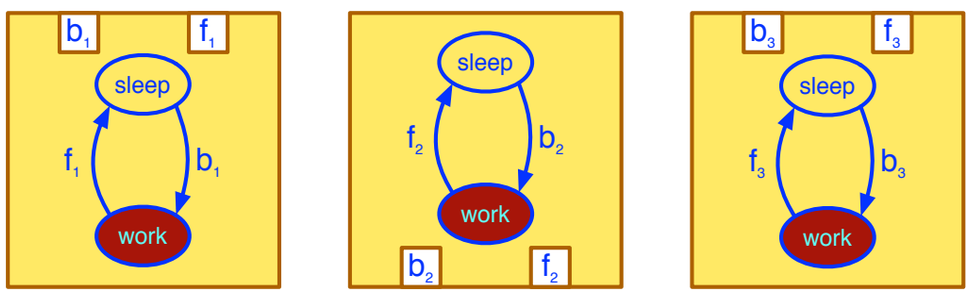


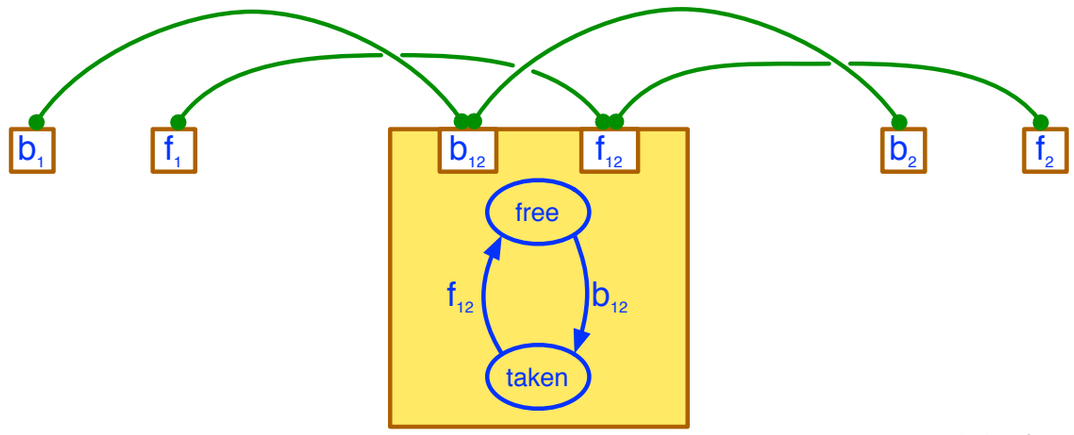
Example continued

$$\varphi_{\gamma_{12}} \equiv (b_1 \Rightarrow b_{12}) \wedge (f_1 \Rightarrow f_{12}) \wedge (b_2 \Rightarrow b_{12}) \wedge (f_2 \Rightarrow f_{12}) \wedge$$
$$(b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2) \wedge (f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2) \wedge (b_{12} \Rightarrow \overline{f_{12}})$$

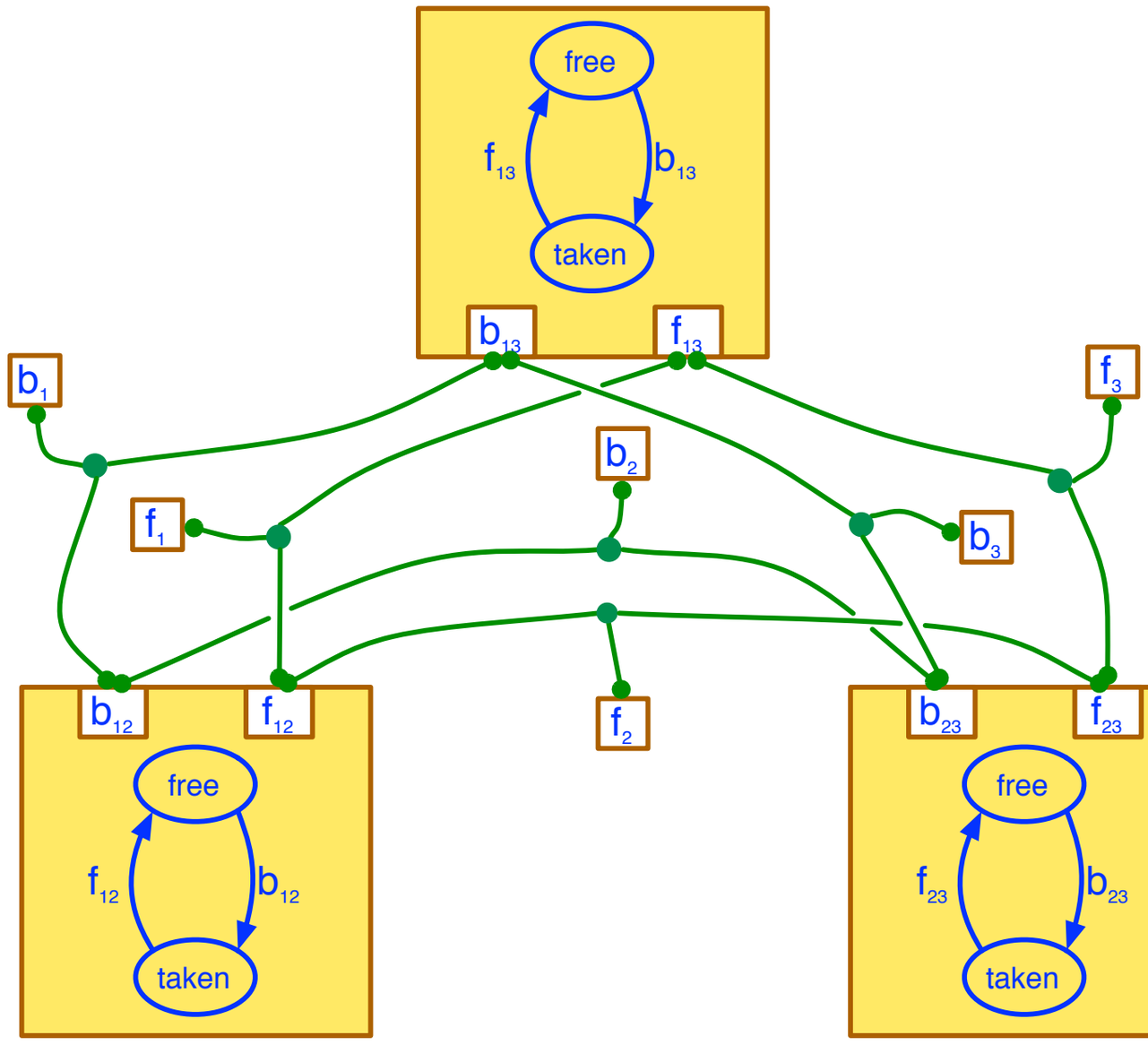
$$b_1 \Rightarrow b_{12} \wedge b_{13}\,, \quad f_1 \Rightarrow f_{12} \wedge f_{13}\,, \quad b_{12} \Rightarrow b_1 \,\texttt{XOR}\, b_2\,, \quad f_{12} \Rightarrow f_1 \,\texttt{XOR}\, f_2\,, \quad b_{12} \Rightarrow \overline{f_{12}}$$
$$b_2 \Rightarrow b_{12} \wedge b_{23}\,, \quad f_2 \Rightarrow f_{12} \wedge f_{23}\,, \quad b_{13} \Rightarrow b_1 \,\texttt{XOR}\, b_3\,, \quad f_{13} \Rightarrow f_1 \,\texttt{XOR}\, f_3\,, \quad b_{13} \Rightarrow \overline{f_{13}}$$
$$b_3 \Rightarrow b_{13} \wedge b_{23}\,, \quad f_3 \Rightarrow f_{13} \wedge f_{23}\,, \quad b_{23} \Rightarrow b_2 \,\texttt{XOR}\, b_3\,, \quad f_{23} \Rightarrow f_2 \,\texttt{XOR}\, f_3\,, \quad b_{23} \Rightarrow \overline{f_{23}}$$

$$\{\emptyset\,,\ b_1 b_{12} b_{13}\,,\ f_1 f_{12} f_{13}\,,\ b_2 b_{12} b_{23}\,,\ f_2 f_{12} f_{23}\,,\ b_3 b_{13} b_{23}\,,\ f_3 f_{13} f_{23}\}$$





Example continued



$$\{\emptyset\,,\ b_1b_{12}b_{13}\,,\ f_1f_{12}f_{13}\,,\ b_2b_{12}b_{23}\,,\ f_2f_{12}f_{23}\,,\ b_3b_{13}b_{23}\,,\ f_3f_{13}f_{23}\}$$

Enforcing properties (safety)

Consider a behaviour $B = (Q, q^0, P, \rightarrow)$

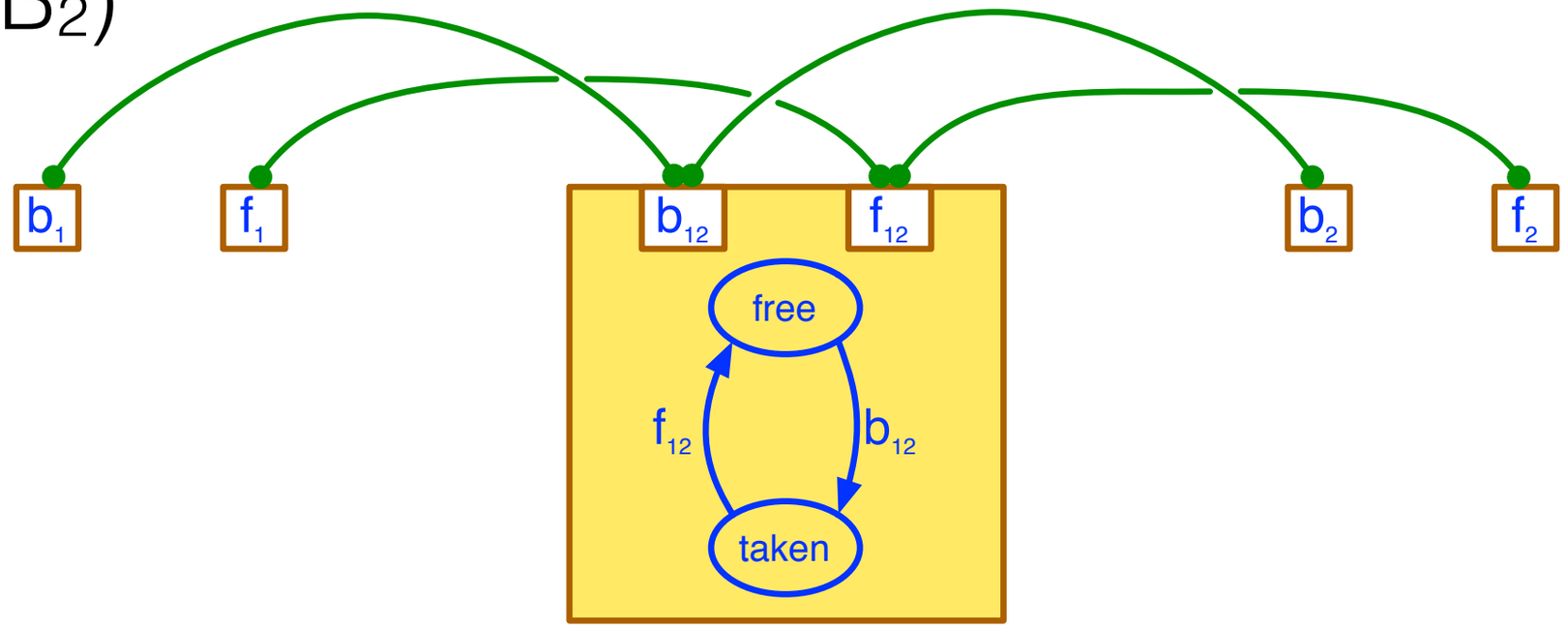
A property: $\Phi \subseteq Q$

An architecture A imposes a property Φ on $\mathcal{B}$ if Φ contains the projection of all the reachable states of $A(\mathcal{B})$ onto $\mathcal{B}$

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

Properties

MUX (B_1, B_2)



Assumed:

AG (f_1 => **A** [$state_1 \neq work$ **W** b_1])

AG (f_2 => **A** [$state_2 \neq work$ **W** b_2])

Characteristic:

AG ($state_1 \neq work$ v $state_2 \neq work$)

Nice properties

- Under suitable conditions
  - Architectures can be composed before applying

$$A_2(A_1(\mathcal{B})) = (A_1 \oplus A_2)(\mathcal{B})$$

  - Architecture application can be restricted

$$A_2(A_1(\mathcal{B}_1, \mathcal{B}_2)) = A_2(A_1(\mathcal{B}_1), \mathcal{B}_2)$$

  - Architecture can be applied partially

$$A(\mathcal{B}_1, \mathcal{B}_2) = A[\mathcal{B}_1](\mathcal{B}_2)$$

Semantics: Interactions

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

$$\gamma(B_1, \ldots, B_n) = (Q, P, \to) \qquad Q = \prod_{i=1}^{n} Q_i \qquad P = \bigcup_{i}^{\cdot} P_i$$

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

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

for each $a \in \gamma$.

Semantics: Priority

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

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

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

for each $a \in 2^P$.

BIP coordination for Java…

SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (2017)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2495

Exogenous coordination of concurrent software components with JavaBIP

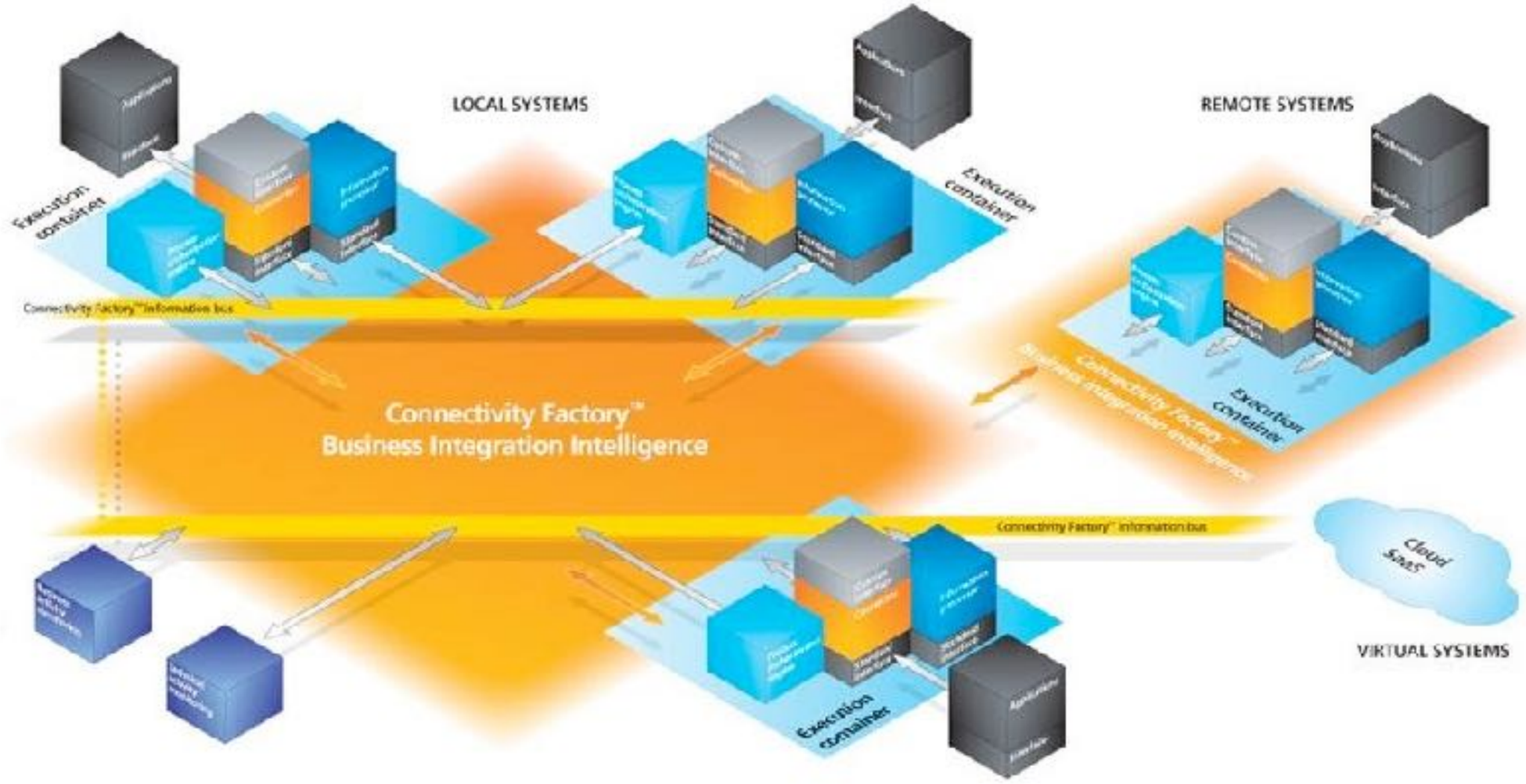
Simon Bliudze[1,*,†], Anastasia Mavridou[2], Radoslaw Szymanek[3] and Alina Zolotukhina[1]

[1]*Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland*
[2]*Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA*
[3]*Crossing-Tech S.A., EPFL Innovation Park, 1015 Lausanne, Switzerland*

SUMMARY

A strong separation of concerns is necessary in order to make the design of domain-specific functional components independent from cross-cutting concerns, such as concurrent access to the shared resources of the execution platform. Native coordination mechanisms, such as locks and monitors, allow developers to address these issues. However, such solutions are not modular; they are complex to design, debug, and main-

Use case: Camel Routes



Many independent routes share memory

We have to control the memory usage

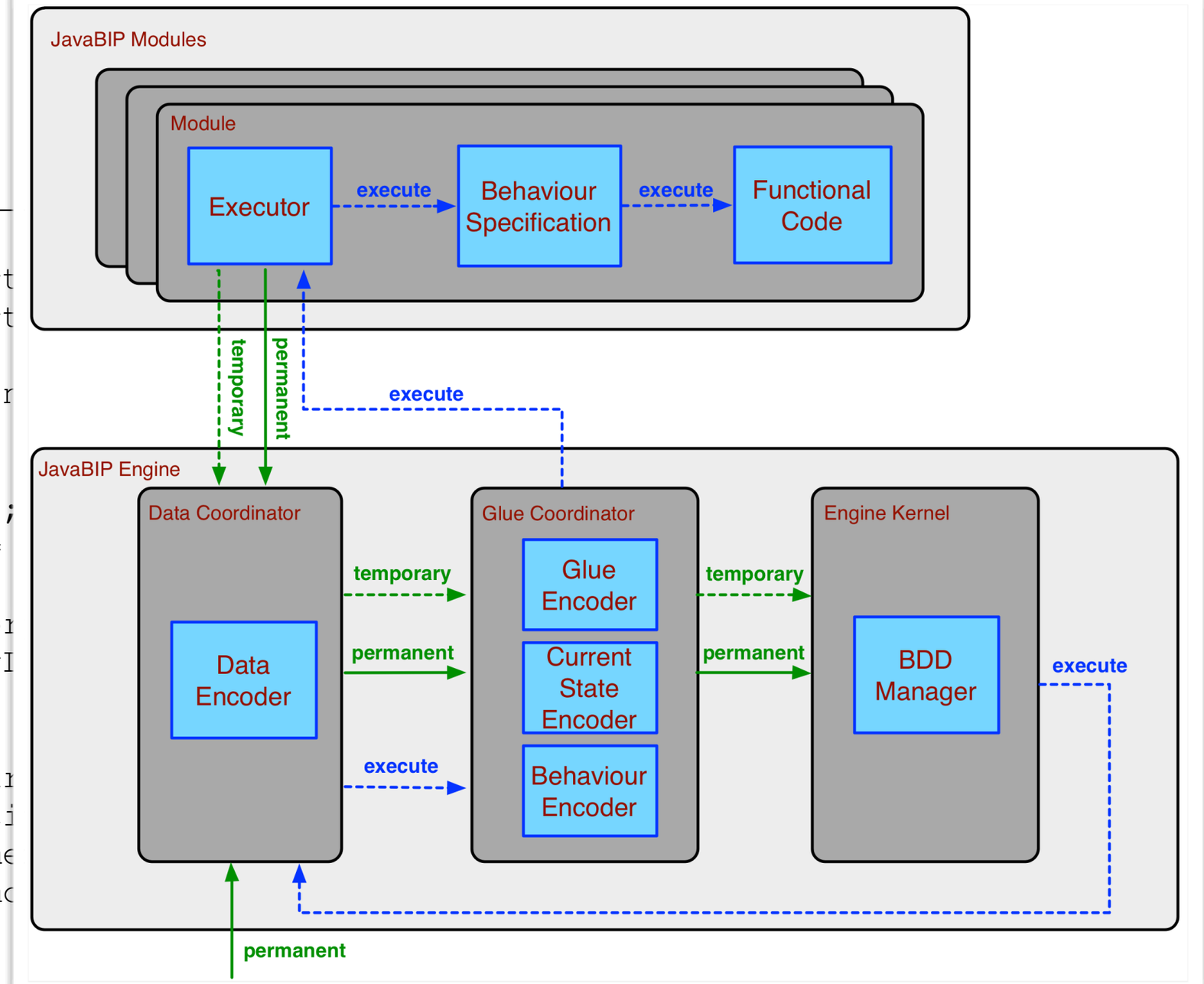
e.g., by limiting to only a safe number of routes simultaneously

JavaBIP

```
 1 @Ports({
 2  @Port(name = "add", type = PortType.enforceable),
 3   @Port(name = "rm", type = PortType.enforceable)
 4 })
 6 @ComponentType(initial = "on", name = "MemoryMonitor")
 7 public class MemoryMonitor {
 8
 9   final private int memoryLimit;
10   private int currentCapacity = 0;
11
12   public MemoryMonitor(int memoryLimit) {
13       this.memoryLimit = memoryLimit;
14       }
15
16   @Transition(name = "add", source = "on", target = "on",
17               guard = "hasCapacity")
18   public void addRoute(@Data("memoryUsage") Integer deltaMemory) {
19     currentCapacity += deltaMemory;
20   }
21
22   @Transition(name = "rm", source = "on", target = "on", guard = "")
23   public void removeRoute(@Data(name="memoryUsage") Integer deltaMemory) {
24     currentCapacity -= deltaMemory;
25   }
26
27   @Guard(name = "hasCapacity")
28   public boolean hasCapacity(@Data("memoryUsage") Integer memoryUsage) {
29     return currentCapacity + memoryUsage < memoryLimit;
30   }
31 }
```

JavaBIP

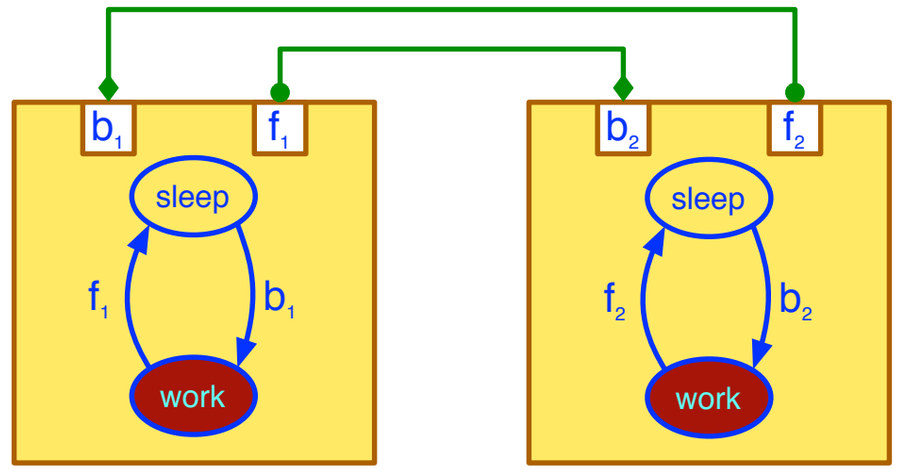
```
 1 @Ports({
 2  @Port(name = "add", type = Port
 3   @Port(name = "rm", type = Port
 4 })
 6 @ComponentType(initial = "on", r
 7 public class MemoryMonitor {
 8
 9   final private int memoryLimit;
10   private int currentCapacity =
11
12   public MemoryMonitor(int memor
13       this.memoryLimit = memoryL
14       }
15
16   @Transition(name = "add", sour
17               guard = "hasCapaci
18   public void addRoute(@Data("me
19     currentCapacity += deltaMemo
20   }
21
22   @Transition(name = "rm", source = "on", target = "on", guard = "")
23   public void removeRoute(@Data(name="memoryUsage") Integer deltaMemory) {
24     currentCapacity -= deltaMemory;
25   }
26
27   @Guard(name = "hasCapacity")
28   public boolean hasCapacity(@Data("memoryUsage") Integer memoryUsage) {
29     return currentCapacity + memoryUsage < memoryLimit;
30   }
31 }
```



…and for functional languages

BIP in Haskel (and Scala)

```
main = runSystem $ do

  begin  <- newPort
  finish <- newPort

  let behavior = forever $ do
    await begin ()
    work
    await finish ()
```

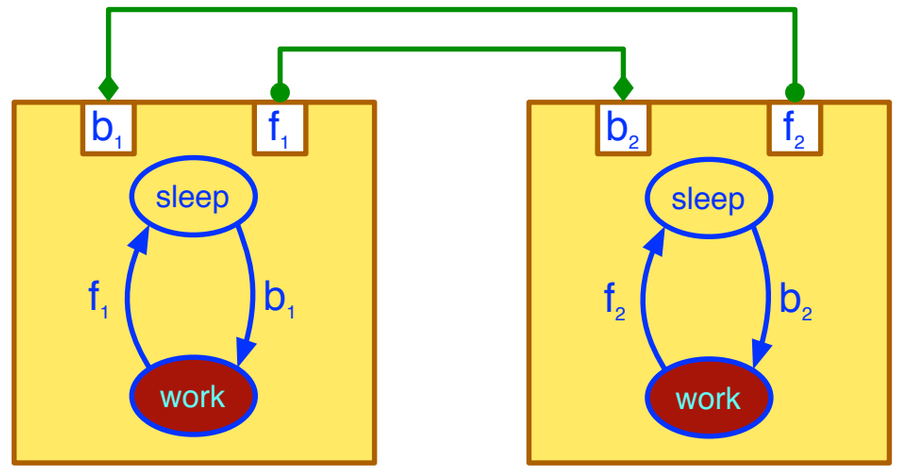



```
a1 <- newAtom behavior
a2 <- newAtom behavior

registerConnector $ firstOf
  [ anyOf
    [ bind a1 finish <> anyOf [bind a2 begin, pure ()]
    , bind a2 finish <> anyOf [bind a1 begin, pure ()]
    ]
  , anyOf
    [ bind a1 begin
    , bind a2 begin
    ]
  ]
```

BIP in Haskel (and Scala)

```
main = runSystem $ do

  begin  <- newPort
  finish <- newPort

  let behavior = forever $ do
        await begin ()
        work
        await finish ()
```

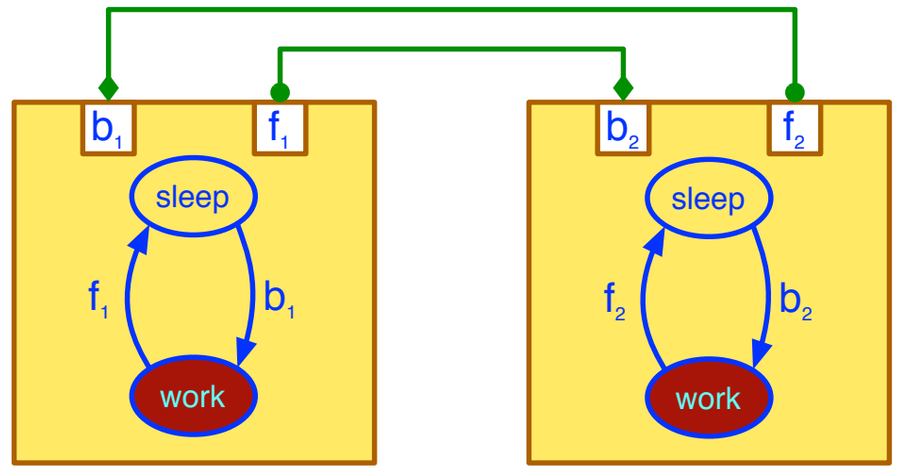
Ports




```
  a1 <- newAtom behavior
  a2 <- newAtom behavior

  registerConnector $ firstOf
    [ anyOf
      [ bind a1 finish <> anyOf [bind a2 begin, pure ()]
      , bind a2 finish <> anyOf [bind a1 begin, pure ()]
      ]
    , anyOf
      [ bind a1 begin
      , bind a2 begin
      ]
    ]
```

BIP in Haskel (and Scala)

```
main = runSystem $ do

  begin  <- newPort
  finish <- newPort

  let behavior = forever $ do
        await begin ()
        work
        await finish ()
```

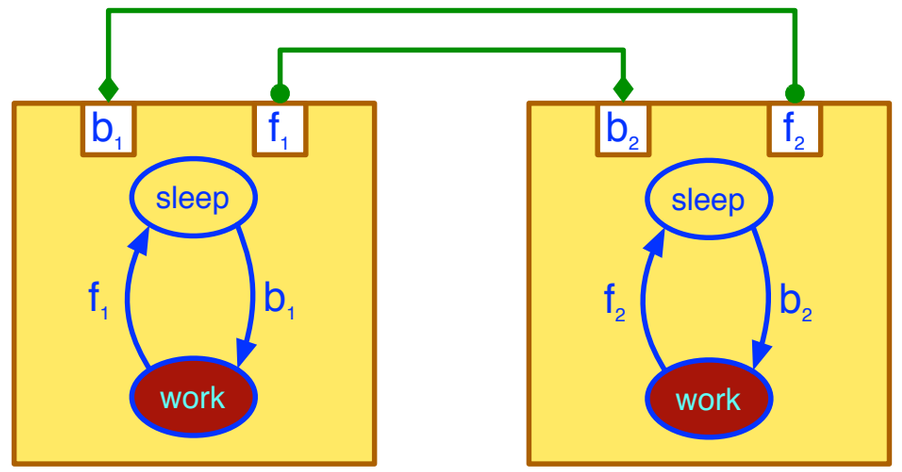


Atom behaviour

```
  a1 <- newAtom behavior
  a2 <- newAtom behavior

  registerConnector $ firstOf
    [ anyOf
      [ bind a1 finish <> anyOf [bind a2 begin, pure ()]
      , bind a2 finish <> anyOf [bind a1 begin, pure ()]
      ]
    , anyOf
      [ bind a1 begin
      , bind a2 begin
      ]
    ]
```

BIP in Haskel (and Scala)

```
main = runSystem $ do

  begin  <- newPort
  finish <- newPort

  let behavior = forever $ do
    await begin ()
    work
    await finish ()
```

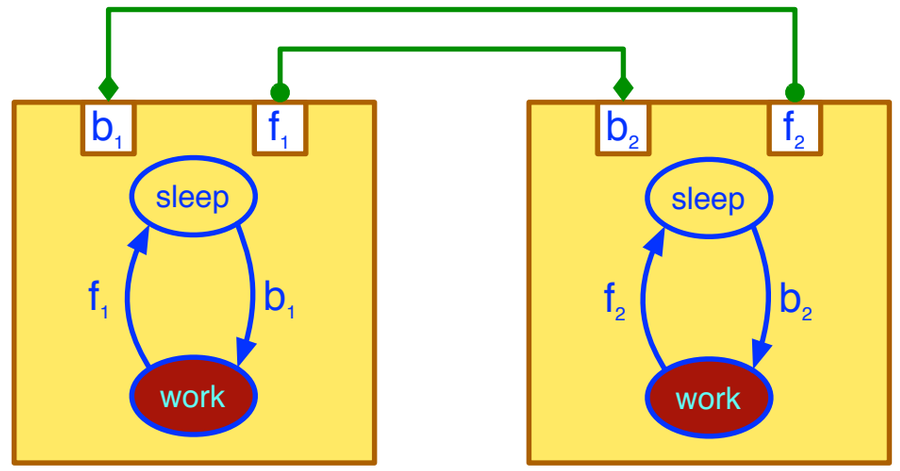



```
  a1 <- newAtom behavior
  a2 <- newAtom behavior

  registerConnector $ firstOf
    [ anyOf
      [ bind a1 finish <> anyOf [bind a2 begin, pure ()]
      , bind a2 finish <> anyOf [bind a1 begin, pure ()]
      ]
    , anyOf
      [ bind a1 begin
      , bind a2 begin
      ]
    ]
```

Atom instantiation

BIP in Haskel (and Scala)

```
main = runSystem $ do

  begin  <- newPort
  finish <- newPort

  let behavior = forever $ do
    await begin ()
    work
    await finish ()
```




```
  a1 <- newAtom behavior
  a2 <- newAtom behavior

  registerConnector $ firstOf
    [ anyOf
      [ bind a1 finish <> anyOf [bind a2 begin, pure ()]
      , bind a2 finish <> anyOf [bind a1 begin, pure ()]
      ]
    , anyOf
      [ bind a1 begin
      , bind a2 begin
      ]
    ]
```

The connector