Rigorous Component-Based Design in BIP

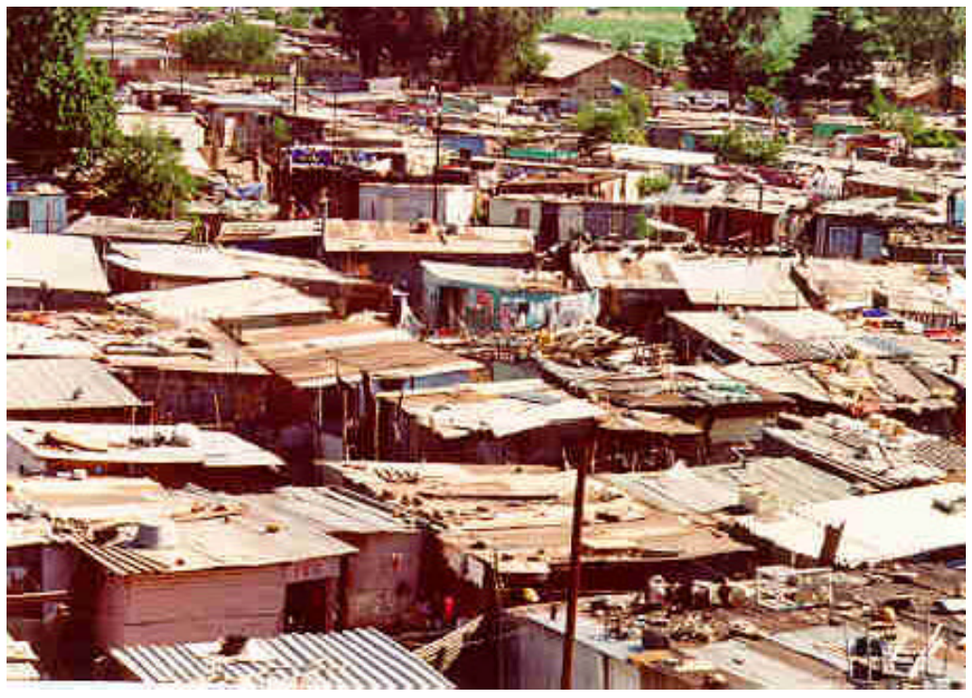
Anastasia Mavridou

Many slides courtesy of Simon Bliudze

Component-based engineering

- Systems are not built from scratch
- Right level of abstraction for:
  - composition
  - reuse of building blocks
  - analysis
  - guarantee properties by construction

Two main approaches



Architecture-agnostic

Architecture-based

Architecture-agnostic approach

Coordination primitives are mixed-up with functional behaviour

Producer-Consumer system

Consumer

```
...
15   public Command consume() {
16     this.lock.lock();
17     try {
18         while(this.commandList.isEmpty()){
19             this.notEmpty.await();
20         }
21         Command c=this.commandList.remove();
22         this.notFull.signal();
23         return c;
24     } catch (InterruptedException e) {
25         e.printStackTrace();
26     } finally{
27         this.lock.unlock();
28     }
29     return null;
30   }
...
```

Architecture-agnostic approach

Coordination primitives are mixed-up with functional behaviour

Producer-Consumer system

Consumer

```
...
15   public Command consume() {
16    this.lock.lock();
17     try {
18         while(this.commandList.isEmpty()){
19             this.notEmpty.await();
20         }
21         Command c=this.commandList.remove();
22         this.notFull.signal();
23         return c;
24     } catch (InterruptedException e) {
25         e.printStackTrace();
26     } finally{
27         this.lock.unlock();
28     }
```

Often results in software that is hard to analyse and maintain

Architecture-based approach

Coordination defined independently from functional behaviour

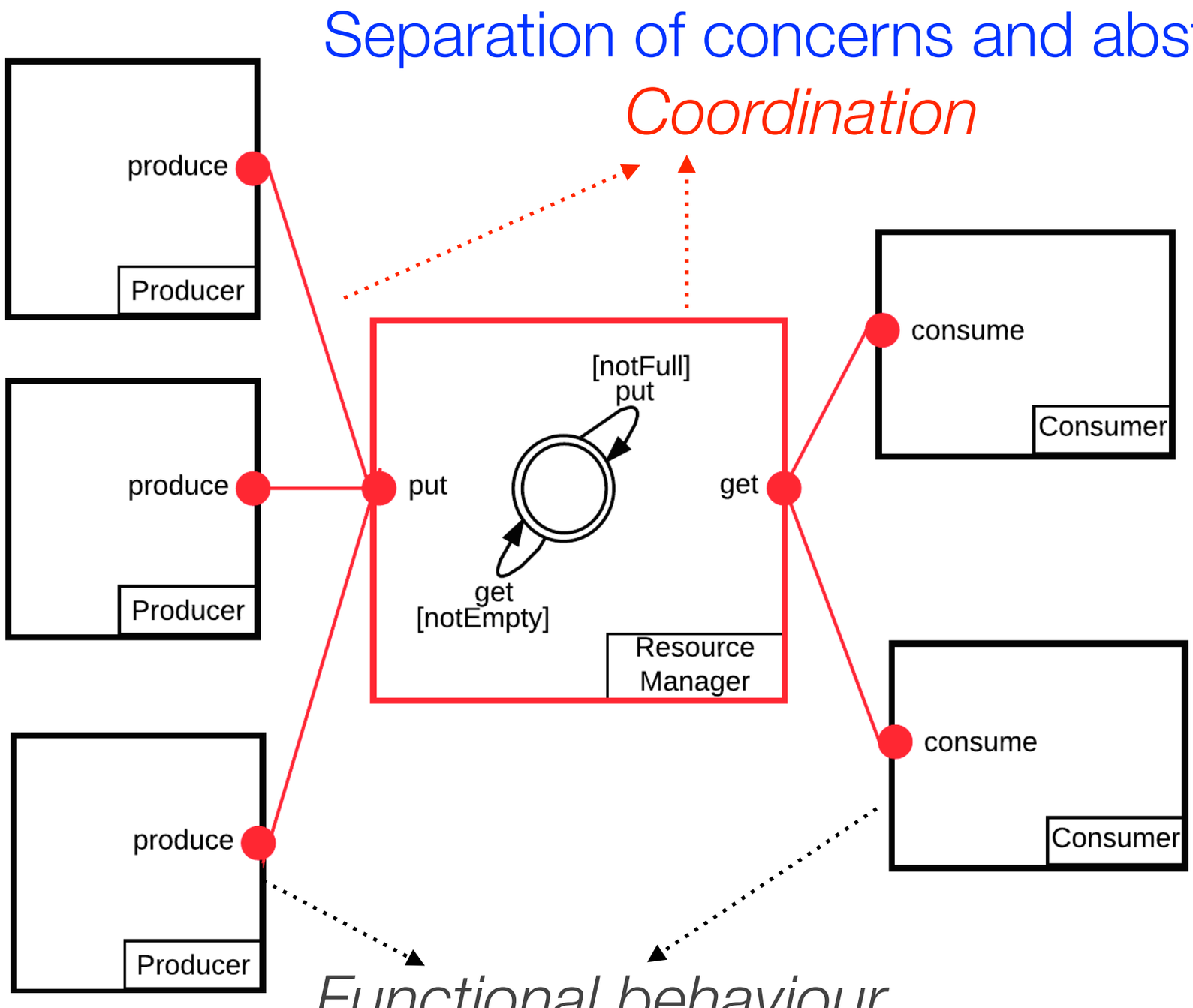
Producer-Consumer design pattern

Consumer

```
...
15   public Command consume() {
16
17      try {
18
19
20
21          Command c=this.commandList.remove();
22
23          return c;
24      } catch (InterruptedException e) {
25          e.printStackTrace();
26
27
28
29      return null;
30   }
...
```

Architecture-based approach

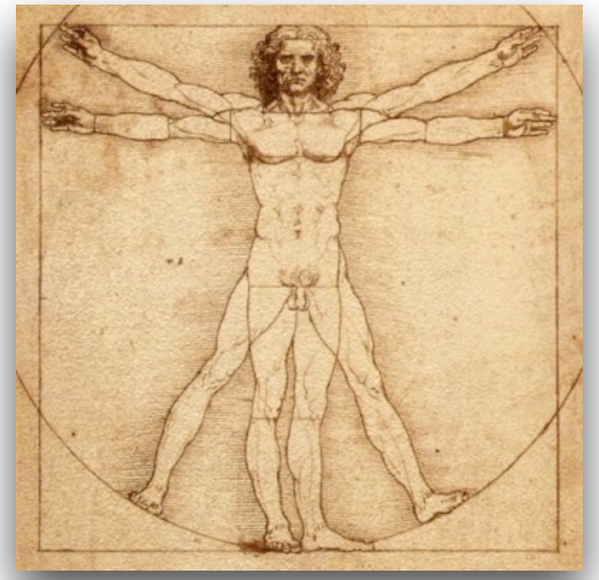
Shifts the focus of developers from low-level code to high-level structures



Objectives

- Manage system complexity
  - Raise the abstraction level
    - Expressive enough to avoid ad-hoc solutions
    - Simple enough to be acceptable by engineers
- Bridging the gap between models and code
  - Raising abstraction level increases the gap
  - Model and implementation must be equivalent
- Build solid and light-weight bridges

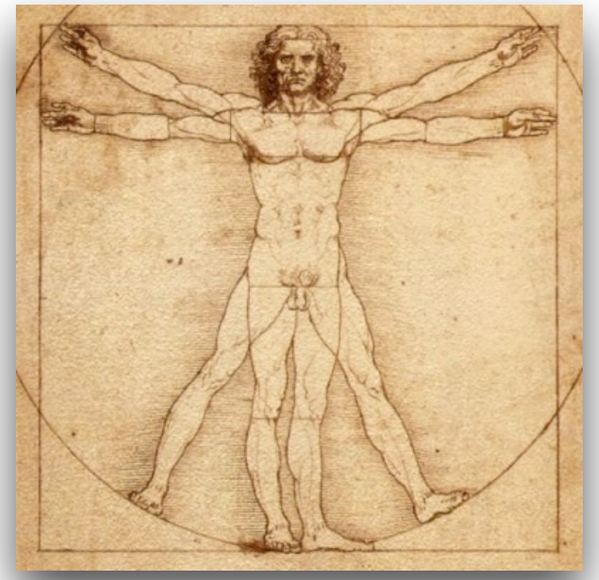
Unifying modelling formalism



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

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

Unifying modelling formalism



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

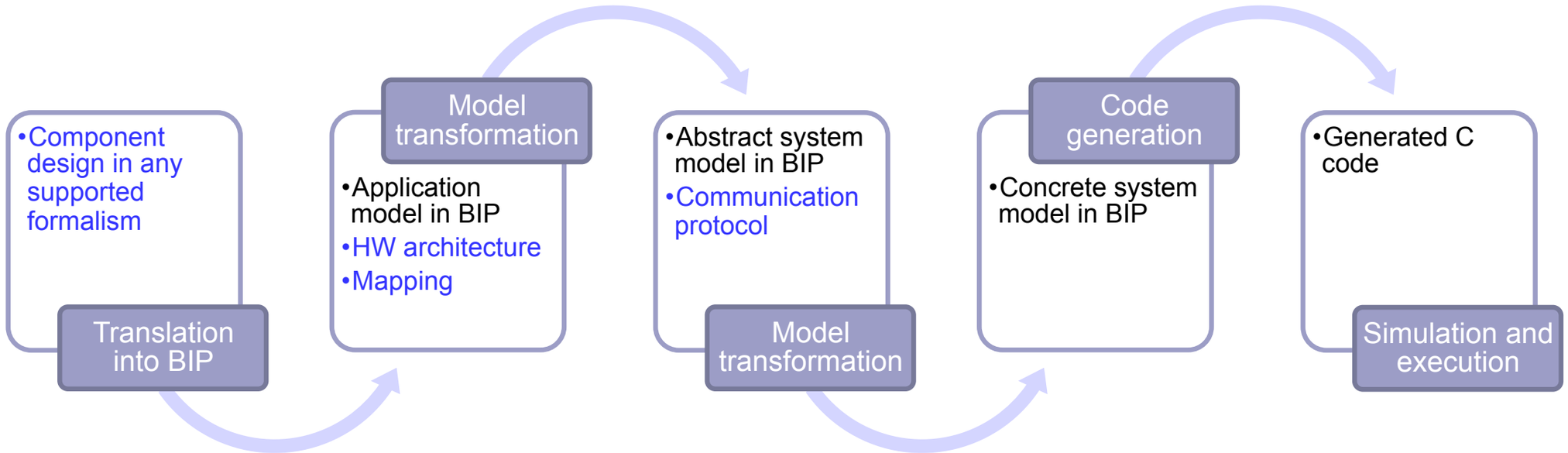
- Light-weight:
  - Clear, accessible formal semantics
  - Minimal set of primitives
  - Separation of concerns
    - computation and
    - coordination
  - Efficient implementation

BIP (Behaviour-Interaction-Priority) is such a formalism

Rigorous system design

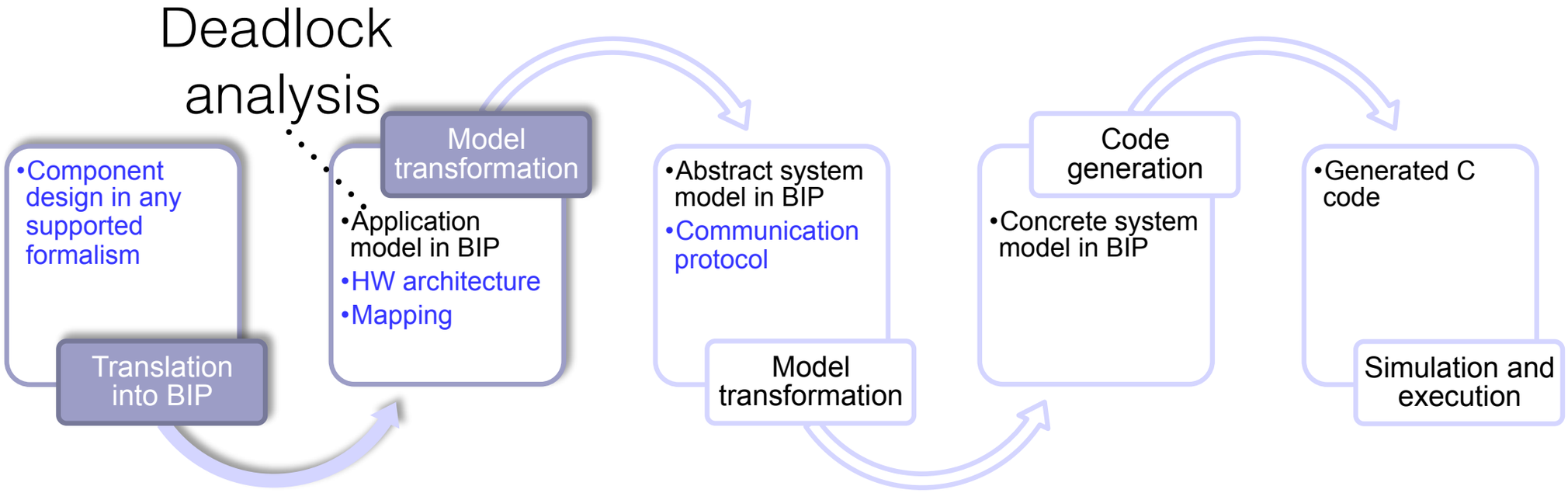
A sequence of semantics-preserving transformations

Rigorous system design



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

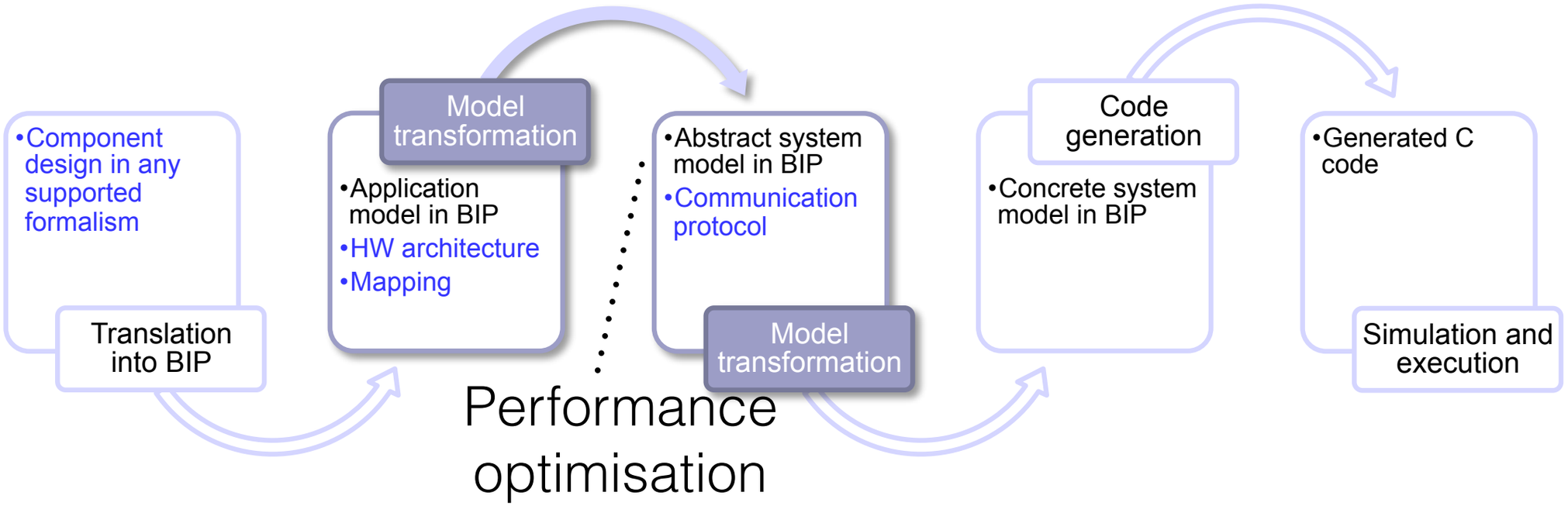
Application model



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

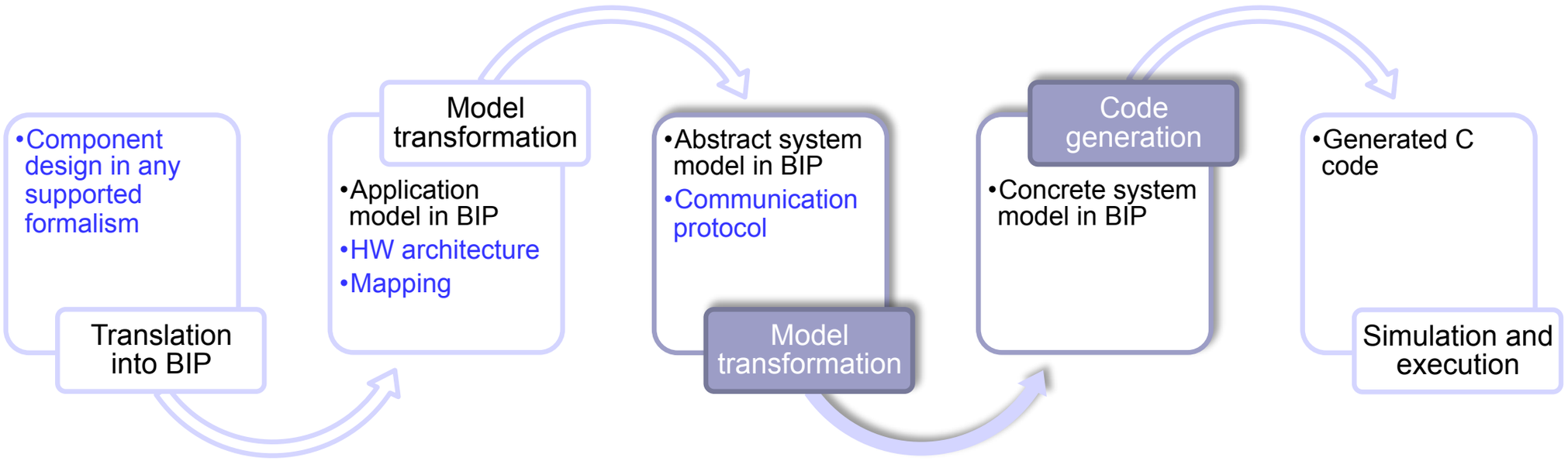
- Safety properties are verified on this model
  - Compositional and incremental deadlock detection

Abstract system model



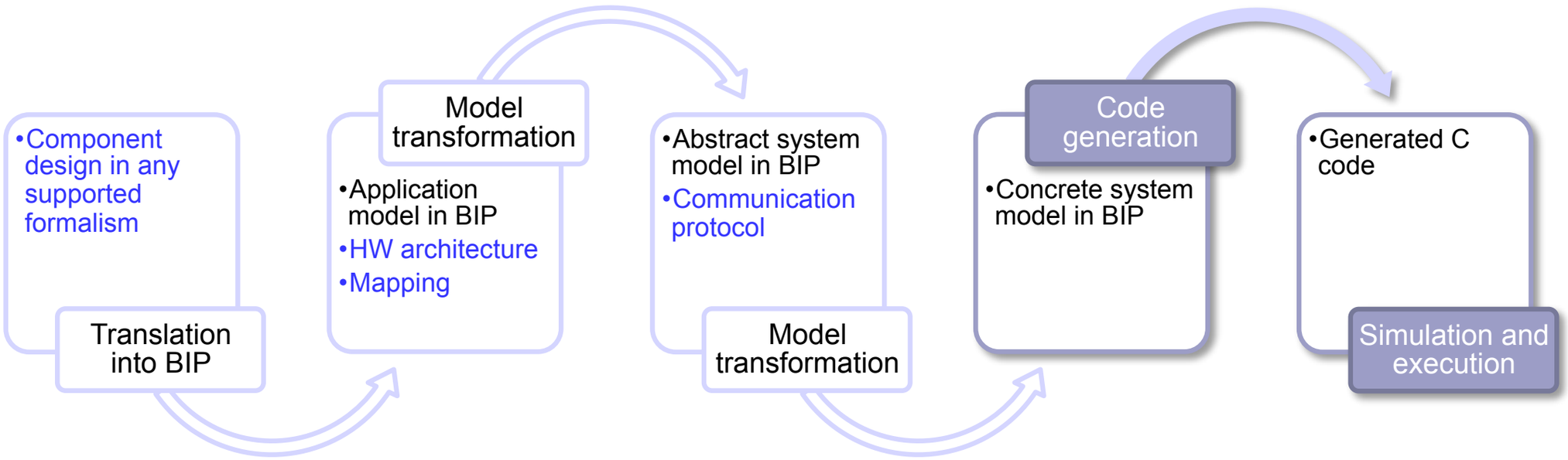
- Abstract system model is generated by a transformation using
  - The model of the target execution platform (processor(s), memory, etc.)
  - A mapping of atomic components to the processing units
- It takes in account
  - Hardware architecture constraints (e.g. mutual exclusion)
  - Execution times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

Concrete system model

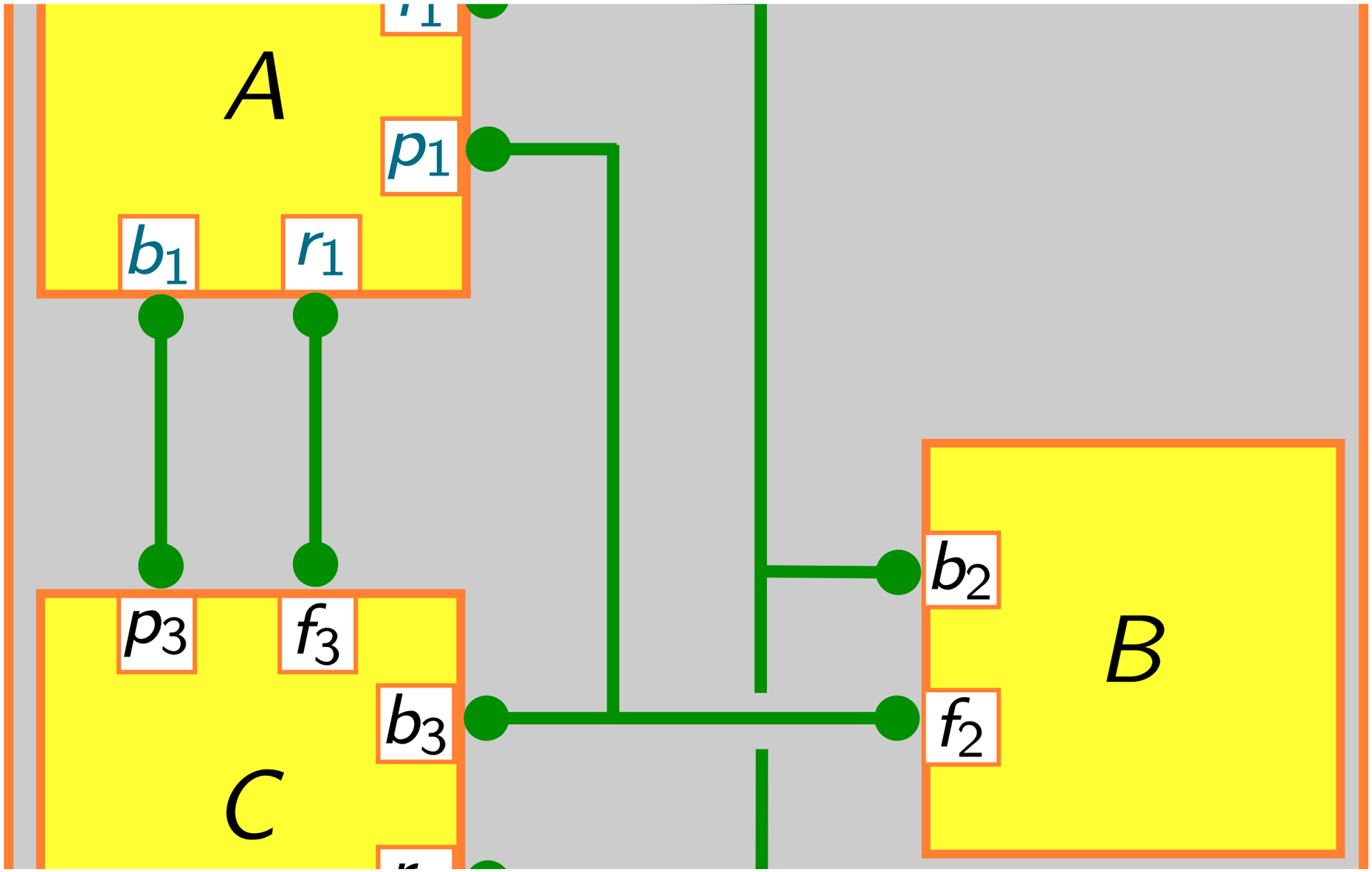


- Concrete system model is obtained by expressing high level BIP coordination mechanisms…
  - Atomic multiparty interactions
  - Priorities
- …by using primitives of the execution platform
  - For example, protocols using asynchronous message passing

Code generation



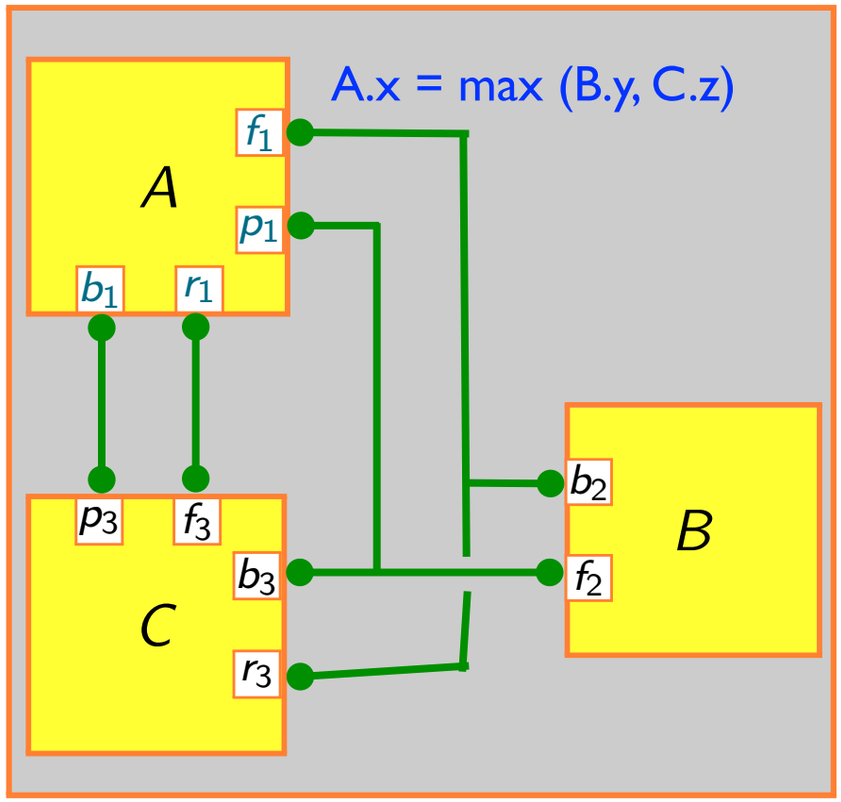
- C++ code automatically generated
- Generated code is monolithic,
  - minimising the coordination overhead



The BIP framework

Operational semantics, engine-based execution, practical examples

Component-based design



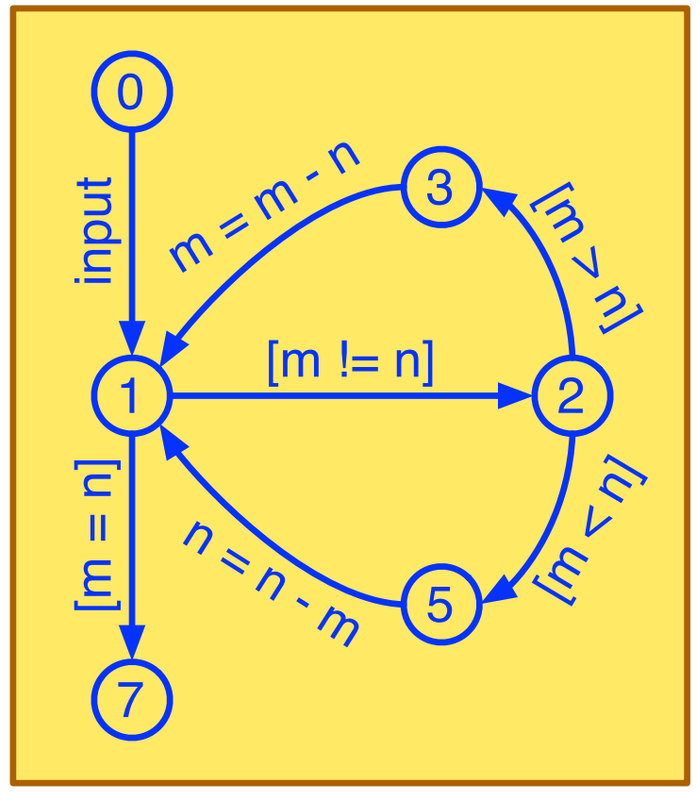
- Three layers
  - Component behaviour
  - Coordination
  - Data transfer
- Interesting results already at this abstraction level
  - Detection of synchronisation deadlocks

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

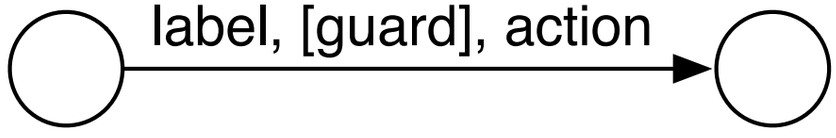
    S. Bliudze and J. Sifakis.
    Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems [SC'11]

Components

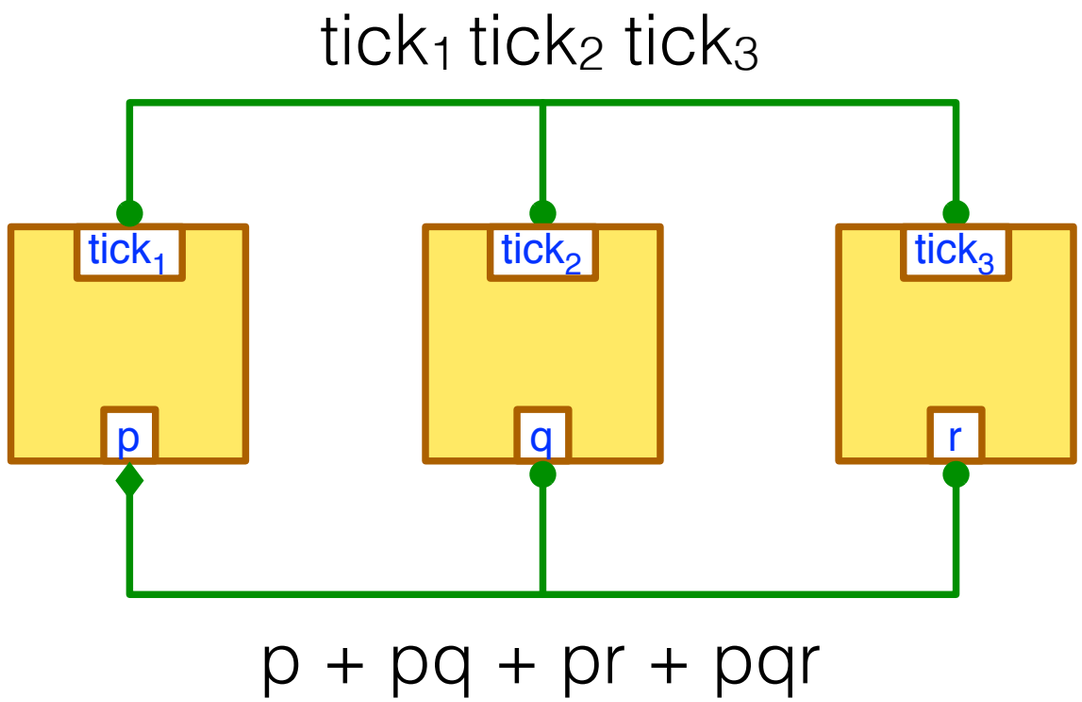
```
0: input(m,n>0);
1: while(m != n){
2:   if (m > n)
3:     m = m - n;
4:   else //m < n
5:     n = n - m;
6: }
7: //m=n=gcd(m,n)
```



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



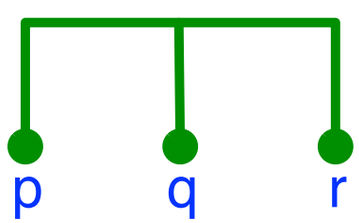
Connectors



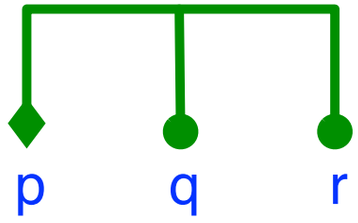
p + pq + pr + pqr

- *Connectors* are tree-like structures
  - ports as leaves and nodes of two types
    - *Triggers* (diamonds) — nodes that can “initiate” an interaction
    - *Synchrons* (bullets) — nodes that can only “join” an interaction initiated by others

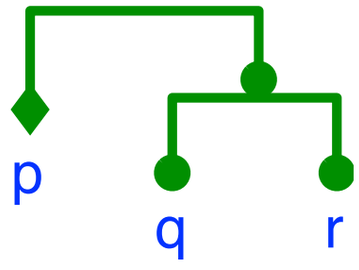
Connector examples

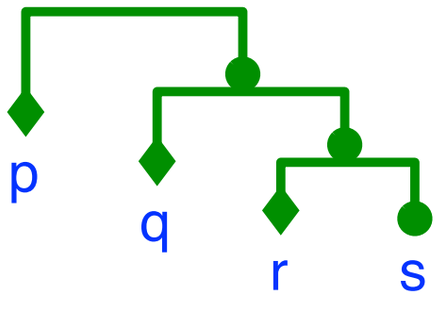


Strong synchronisation: pqr

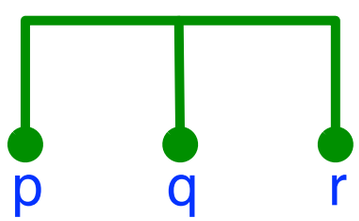


Broadcast: p + pq + pr + pqr

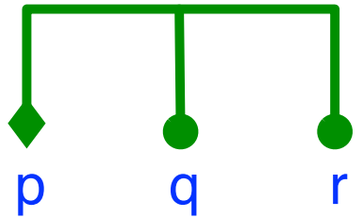




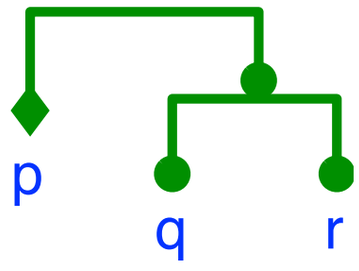
Connector examples



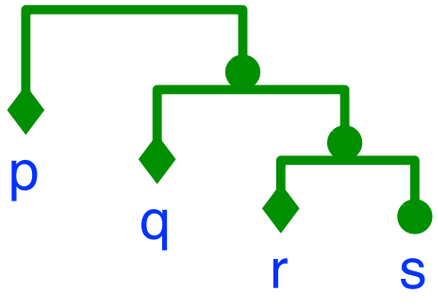
Strong synchronisation: pqr



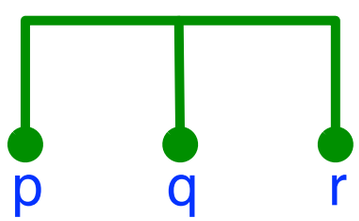
Broadcast: p + pq + pr + pqr



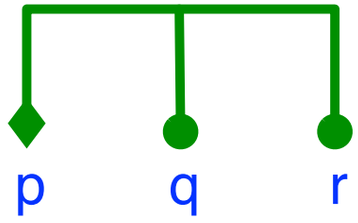
Atomic broadcast: p + pqr



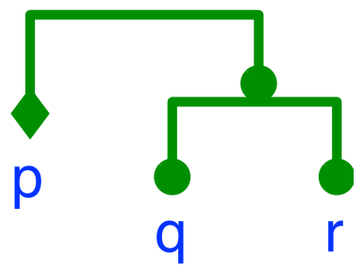
Connector examples



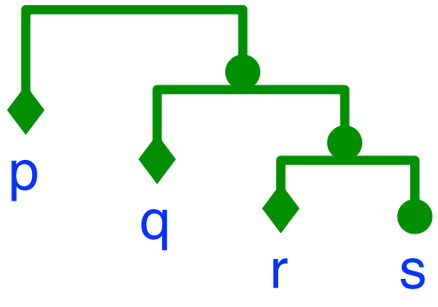
Strong synchronisation: pqr



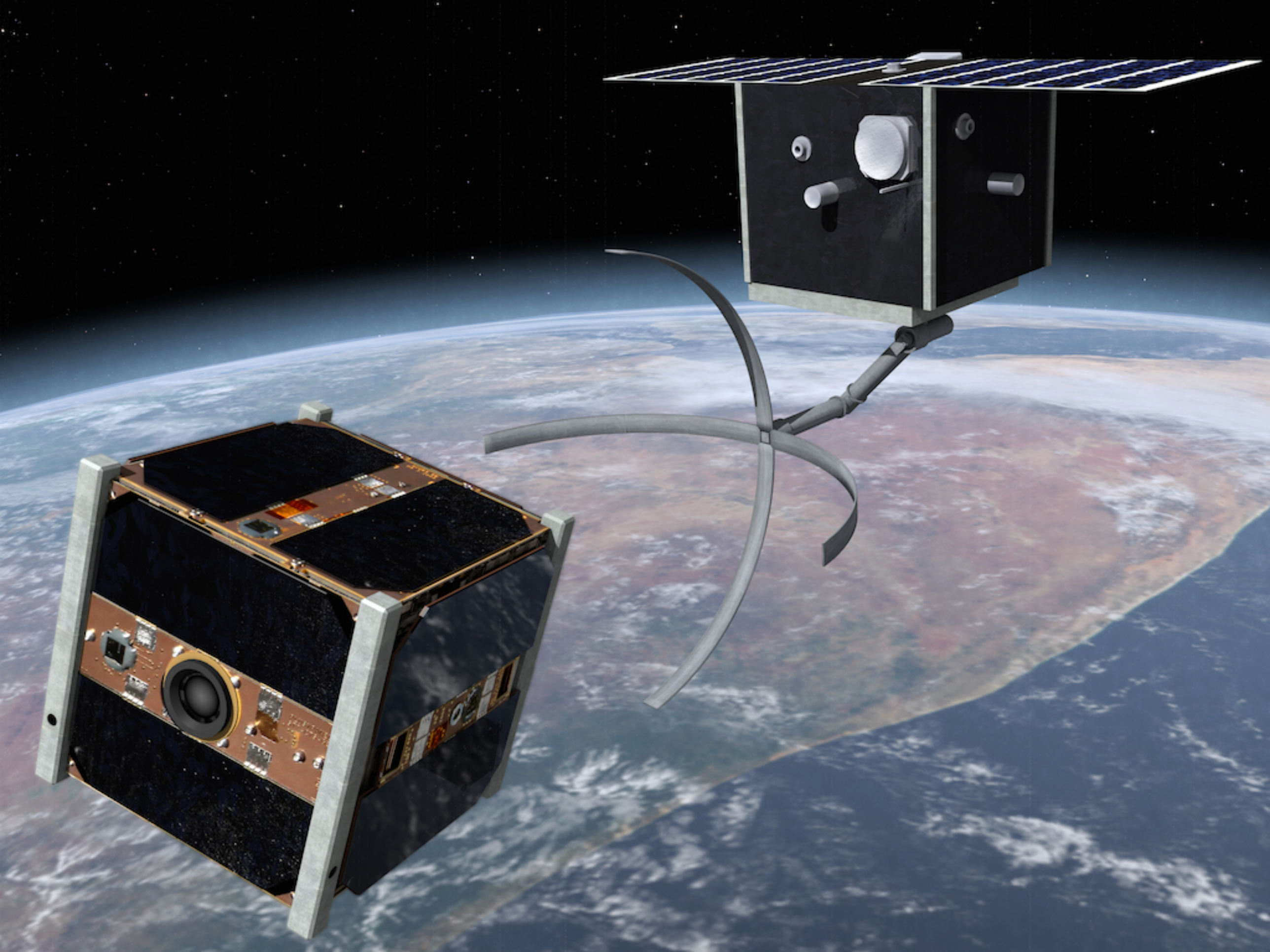
Broadcast: p + pq + pr + pqr



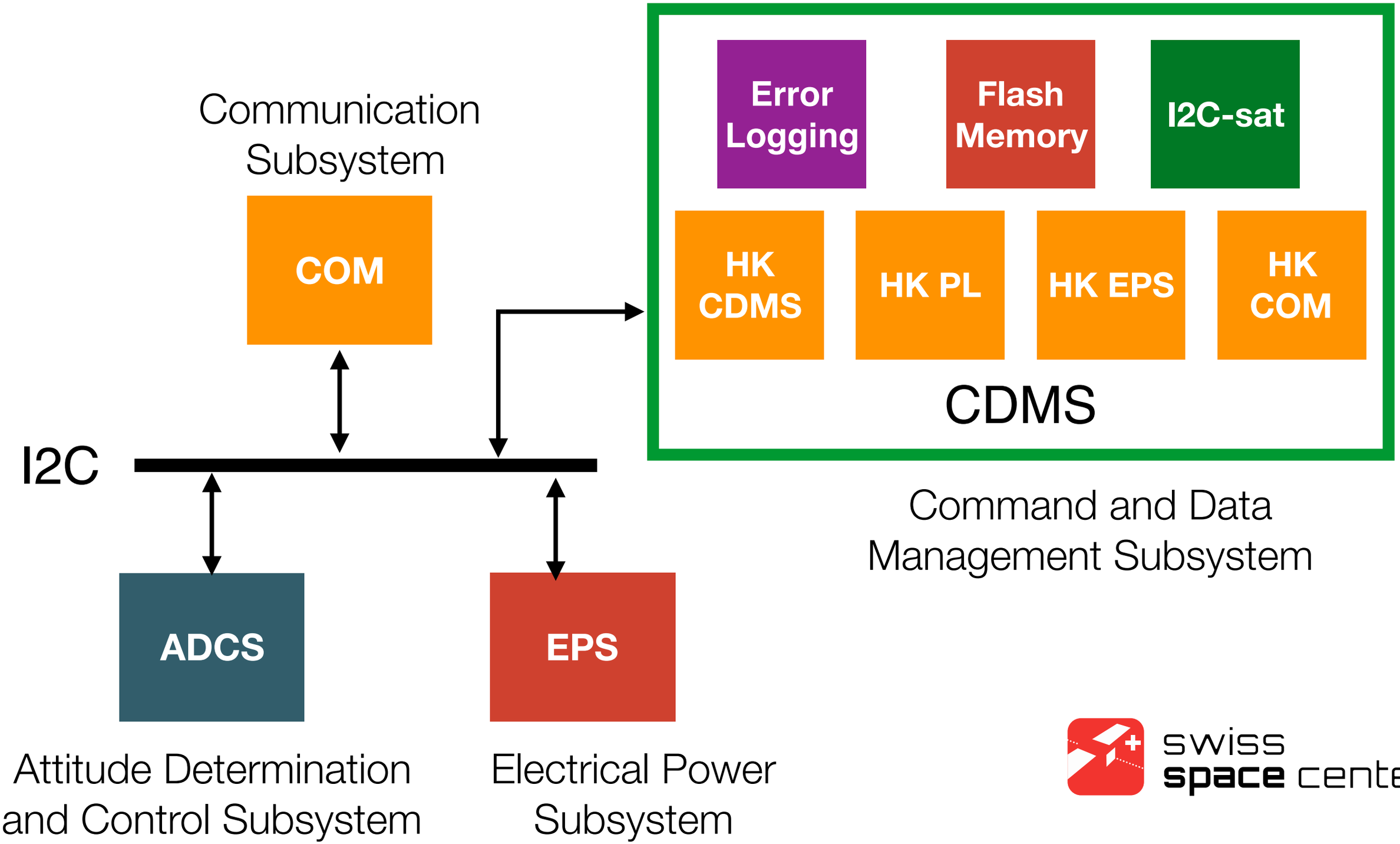
Atomic broadcast: p + pqr



Causal chain: p + pq + pqr + pqrs



Practical example



swiss space center

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
error
[STATUS ==
ERROR]
anomaly
FAILURE
success
res_I2C_PL
{decodeMessage;}
SEND_HK_
REPORT
ask_I2C_TTC
{composeMessage;}
write_request
{setWrite;}
SEND_TM
WRITE_
MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
ask_I2C_TTC
write_request
enable_HK
disable_HK
read_HK
enable
disable
HK switch
readHK
ENABLED
disable
enable
DISABLED
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;}
success
failure
CRITICAL_
FAILURE
Status
enable_
PS
disable
_PS

GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

connector

enabled cnctr.

synchron

trigger

read - transition name
[guard]
{external_function;}

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
error
[STATUS ==
ERROR]
anomaly
FAILURE
success
res_I2C_PL
{decodeMessage;}
ask_I2C_TTC
write_request
SEND_HK_
REPORT
ask_I2C_TTC
{composeMessage;}
write_request
{setWrite;}
SEND_TM
WRITE_
MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
read_HK
enable
disable
HK switch
readHK
ENABLED
disable
enable
DISABLED
Packe store switch
send_MEM
MEMORY
disable
enable
TTC
send_TM
send_TM
send_MEM
disable enable
ask_I2C_
EPS
res_I2C_
EPS
NOMINAL
success
failure
{timer = 0;}
ANOMALY
res_I2C
_EPS
ask_I2C_EPS
[TIMER > MAX]
{composeMessage;}
success
failure
CRITICAL_
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;}

I2C INTERFACE
I2C_NOFAIL INTERFACE
MEM_WRNF INTERFACE
error
failure
ask_I2C_PL
res_I2C_PL
error
res_I2C_TTC
res_mem
read_HK
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
WAIT
read_HK
[PERIOD]
START
ask_I2C_PL
{composeMessage;}
I2C
failure
error
[STATUS ==
ERROR]
anomaly
FAILURE
success
res_I2C_PL
{decodeMessage;}
SEND_HK_
REPORT
ask_I2C_TTC
write_request
ask_I2C_TTC
{composeMessage;}
write_request
{setWrite;}
SEND_TM
WRITE_
MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
ask_I2C_TTC
write_request
enable
_HK
disable
_HK
ask_I2C_
EPS
res_I2C_
EPS
NOMINAL
success
failure
{timer = 0;}
ANOMALY
res_I2C
_EPS
ask_I2C_EPS
[TIMER > MAX]
{composeMessage;}
success
failure
CRITICAL_
FAILURE
Status
disable
_PS
enable_
PS

GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

connector

enabled cnctr.

synchron

trigger

read - transition name
[guard]
{external_function;}

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
error
[STATUS ==
ERROR]
FAILURE
anomaly
success
res_I2C_PL
{decodeMessage;}
ask_I2C_TTC
write_request
SEND_HK_
REPORT
ask_I2C_TTC
{composeMessage;}
write_request
{setWrite;}
SEND_TM
WRITE_
MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
read_HK
enable
disable
HK switch
readHK
ENABLED
disable
enable
DISABLED
Packe store switch
send_MEM
MEMORY
disable
enable
TTC
send_TM
send_TM
send_MEM
disable enable
ask_I2C_
EPS
res_I2C_
EPS
NOMINAL
success
failure
{timer = 0;}
ANOMALY
res_I2C
_EPS
ask_I2C_EPS
[TIMER > MAX]
{composeMessage;}
success
failure
CRITICAL_
FAILURE
Status
enable
_HK
disable
_HK
disable
_PS
enable_
PS

GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

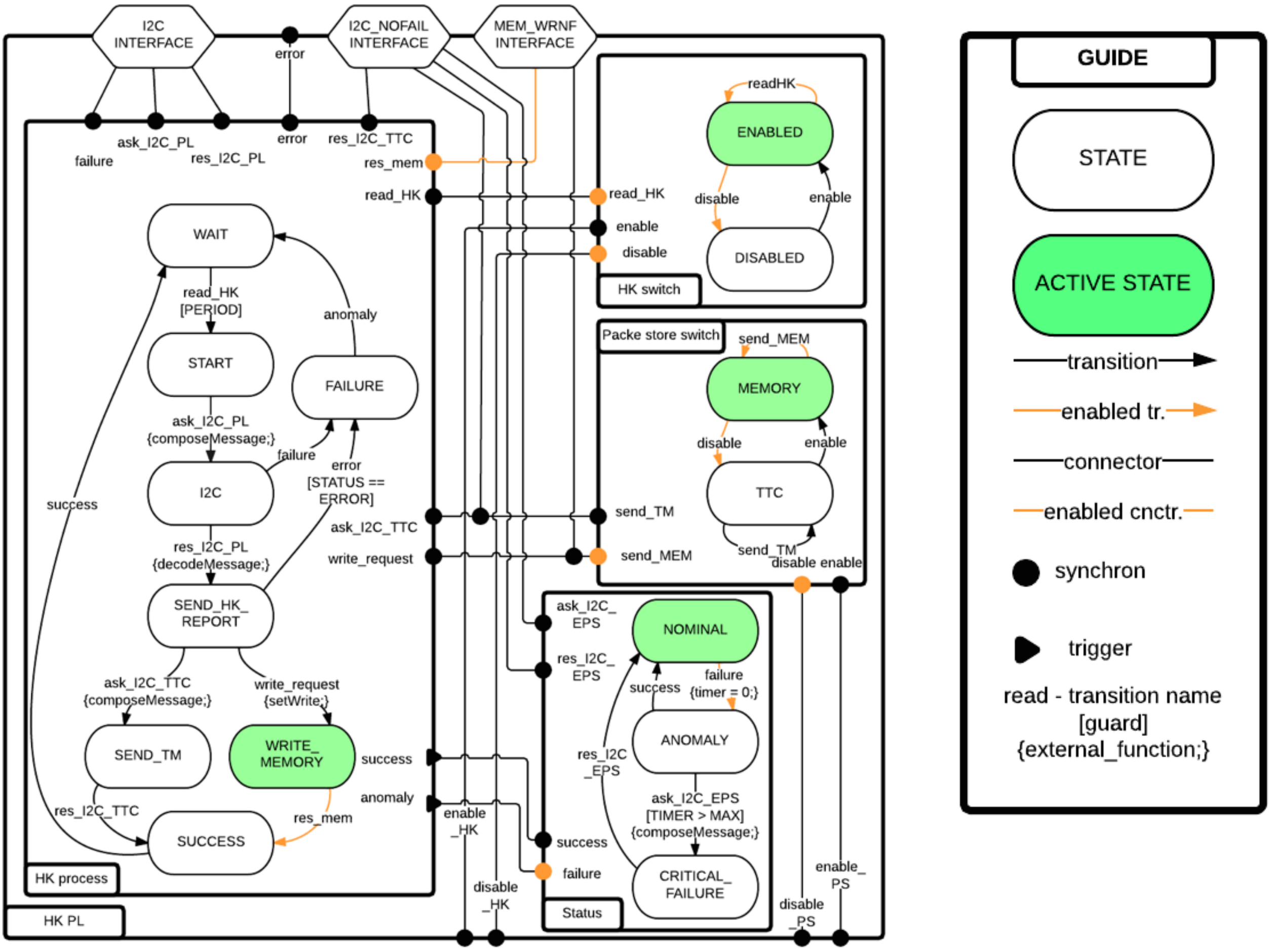
connector

enabled cnctr.

synchron

trigger

read - transition name
[guard]
{external_function;}


I2C INTERFACE
I2C_NOFAIL INTERFACE
MEM_WRNF INTERFACE
error
ask_I2C_PL
failure
res_I2C_PL
error
res_I2C_TTC
res_mem
read_HK
WAIT
read_HK [PERIOD]
START
ask_I2C_PL {composeMessage;}
I2C
failure
error [STATUS == ERROR]
FAILURE
anomaly
success
res_I2C_PL {decodeMessage;}
SEND_HK_ REPORT
ask_I2C_TTC {composeMessage;}
write_request {setWrite;}
SEND_TM
WRITE_ MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
ask_I2C_TTC
write_request
enable_HK
disable_HK
read_HK
enable
disable
HK switch
readHK
ENABLED
disable
enable
DISABLED
Packe store switch
send_MEM
MEMORY
disable
enable
TTC
send_TM
send_MEM
send_TM
disable enable
ask_I2C_EPS
res_I2C_EPS
success
failure
NOMINAL
success
failure {timer = 0;}
ANOMALY
res_I2C_EPS
ask_I2C_EPS [TIMER > MAX] {composeMessage;}
CRITICAL_ FAILURE
success
failure
Status
disable_PS
enable_PS
GUIDE
STATE
ACTIVE STATE
transition
enabled tr.
connector
enabled cnctr.
synchron
trigger
read - transition name
[guard]
{external_function;}

I2C INTERFACE
I2C_NOFAIL INTERFACE
MEM_WRNF INTERFACE
error
failure
ask_I2C_PL
res_I2C_PL
error
res_I2C_TTC
res_mem
read_HK
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
WAIT
read_HK
[PERIOD]
START
ask_I2C_PL
{composeMessage;}
I2C
failure
error
[STATUS ==
ERROR]
anomaly
FAILURE
success
res_I2C_PL
{decodeMessage;}
ask_I2C_TTC
write_request
SEND_HK_
REPORT
ask_I2C_TTC
{composeMessage;}
write_request
{setWrite;}
SEND_TM
WRITE_
MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
enable
_HK
disable
_HK
ask_I2C_
EPS
res_I2C_
EPS
NOMINAL
success
failure
{timer = 0;}
ANOMALY
res_I2C
_EPS
ask_I2C_EPS
[TIMER > MAX]
{composeMessage;}
success
failure
CRITICAL_
FAILURE
Status
enable_
PS
disable
_PS

GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

connector

enabled cnctr.

synchron

trigger

read - transition name
[guard]
{external_function;}

Example 2

Stopping housekeeping

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
error [STATUS == ERROR]
FAILURE
anomaly
success
res_I2C_PL {decodeMessage;}
ask_I2C_TTC
write_request
SEND_HK_ REPORT
ask_I2C_TTC {composeMessage;}
write_request {setWrite;}
SEND_TM
WRITE_ MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
read_HK
enable
disable
HK switch
readHK
ENABLED
disable
enable
DISABLED
Packe store switch
send_MEM
MEMORY
disable
enable
TTC
send_TM
send_TM
send_MEM
disable enable
ask_I2C_ EPS
res_I2C_ EPS
NOMINAL
success
failure {timer = 0;}
ANOMALY
res_I2C _EPS
ask_I2C_EPS [TIMER > MAX] {composeMessage;}
success
failure
CRITICAL_ FAILURE
Status
enable_HK
disable_HK
enable_PS
disable_PS

GUIDE

STATE

ACTIVE STATE

transition

enabled tr.

connector

enabled cnctr.

synchron

trigger

read - transition name
[guard]
{external_function;}

I2C INTERFACE
I2C_NOFAIL INTERFACE
MEM_WRNF INTERFACE
error
ask_I2C_PL
failure
res_I2C_PL
error
res_I2C_TTC
res_mem
read_HK
WAIT
read_HK
[PERIOD]
START
ask_I2C_PL
{composeMessage;}
I2C
failure
error
[STATUS ==
ERROR]
FAILURE
anomaly
success
ask_I2C_TTC
res_I2C_PL
{decodeMessage;}
write_request
SEND_HK_
REPORT
ask_I2C_TTC
{composeMessage;}
write_request
{setWrite;}
SEND_TM
WRITE_
MEMORY
success
anomaly
res_I2C_TTC
res_mem
SUCCESS
HK process
HK PL
read_HK
enable
disable
HK switch
readHK
ENABLED
disable
enable
DISABLED
Packe store switch
send_MEM
MEMORY
disable
enable
TTC
send_TM
send_TM
send_MEM
disable enable
ask_I2C_
EPS
res_I2C_
EPS
NOMINAL
failure
{timer = 0;}
success
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;}

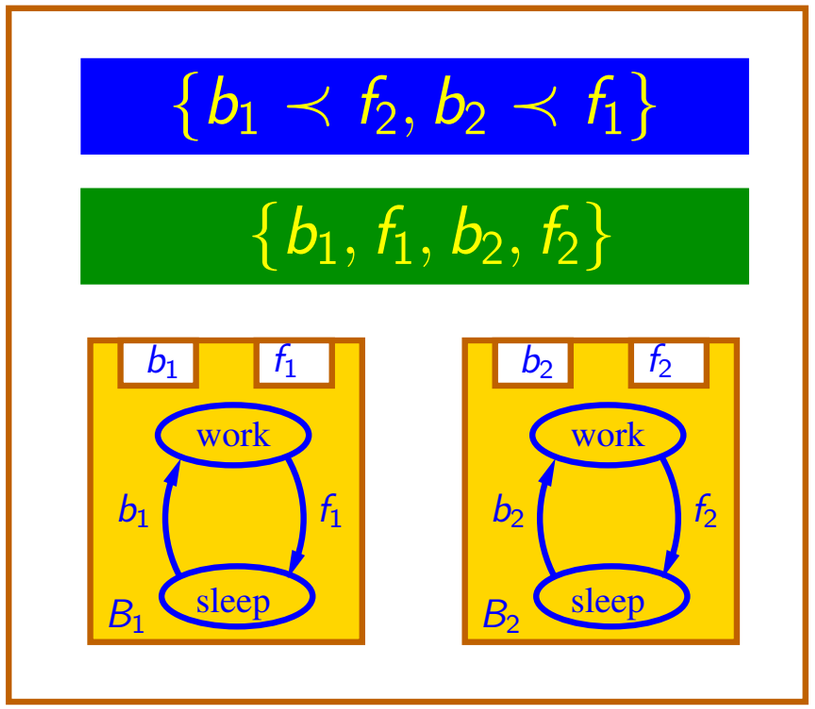
sem. or semicolon.
se·man·tic (si man...
to meaning or arising from...
symbols: semantic change; semantic...
semantics. [1655–65; < Gk sêman...
mant(ós) marked (sêman-, base o...
verbal adj. suffix; akin to *sêma* si...
se·man·tics (si man′tiks), *n.* (u...
linguistics dealing with the stu...
meaning is structured in langua...
over time. **2.** the branch of se...
tionship between signs or sy...
meaning, or an interpretation...
mence, etc.: *Let's not argue*...
tence. [1895–1900] —**se·man′ti·c**...
...n). n.

The BIP language

- BIP stands for Behaviour, Interaction, Priority

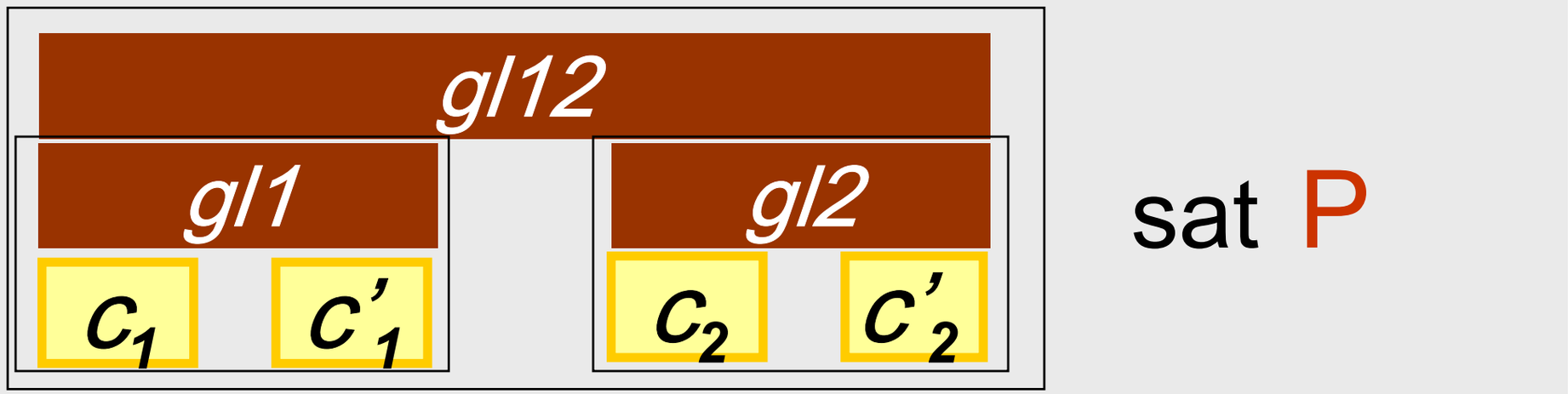
  A behaviour is a Finite State Machine
  An interaction is a set of ports that synchronise
  Interaction and Priority are collectively called Glue

3 layers



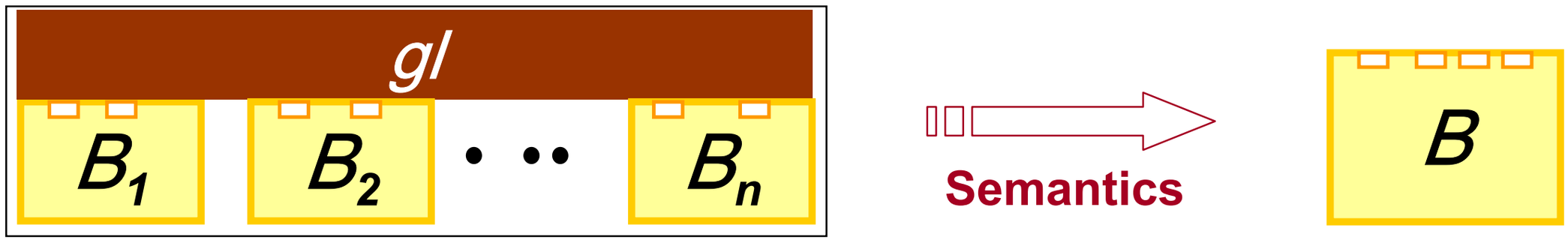
Component-based construction

- Build a component C satisfying a given property P from:
  - *C0* a set of atomic components
  - *GL = {gl1, …, gli, …}* a set of glue operators on components



Component-based construction

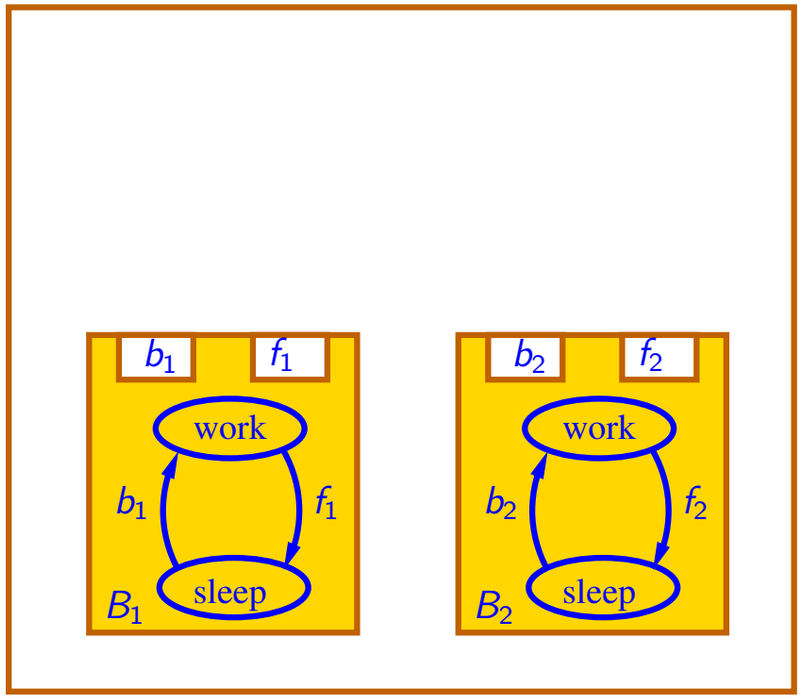
- Glue operators transform set of components into components



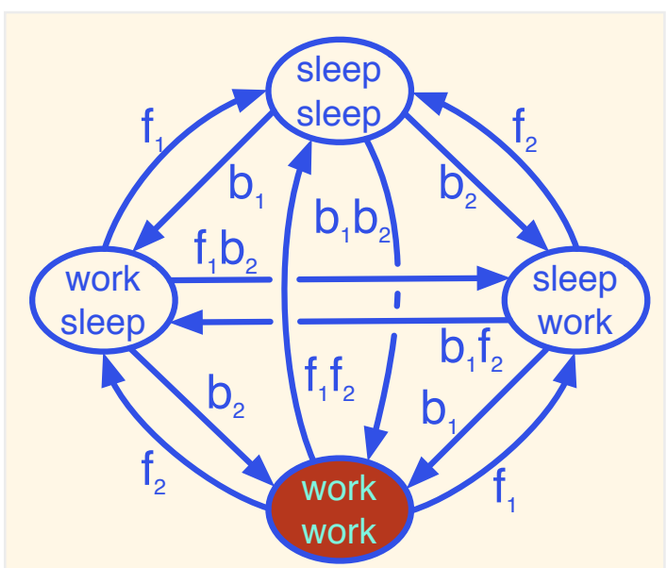
- Glue operators *gl(B1, B2, .., Bn)*
  - model communication mechanisms (protocols, schedulers, buses)
  - restrict the behaviour of their arguments

BIP composition example

Safety property:
Mutual exclusion

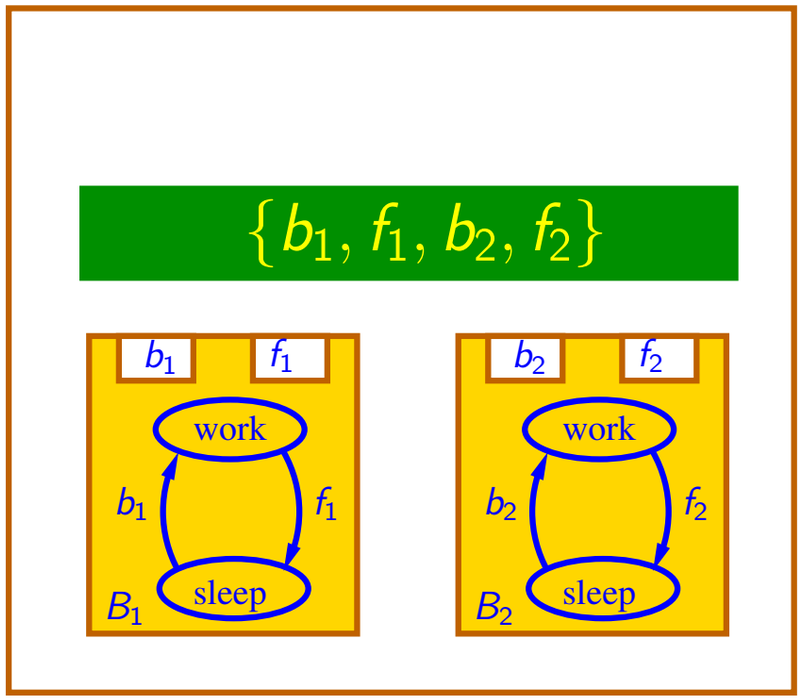


No restrictions

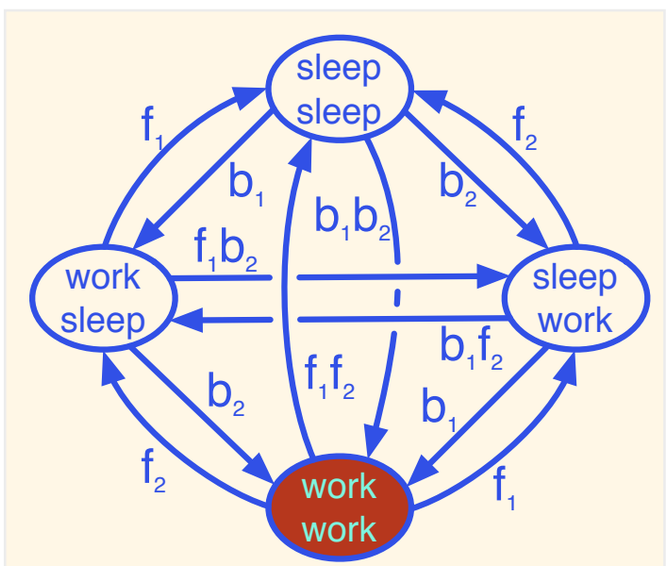


BIP composition example

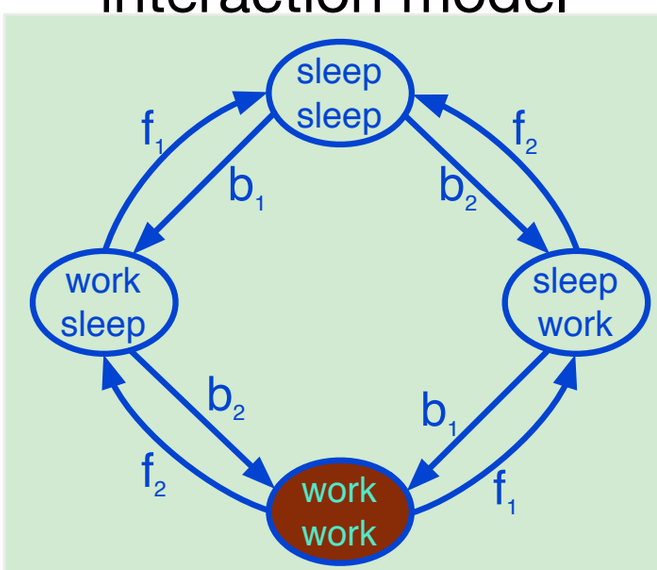
Safety property:
Mutual exclusion



No restrictions

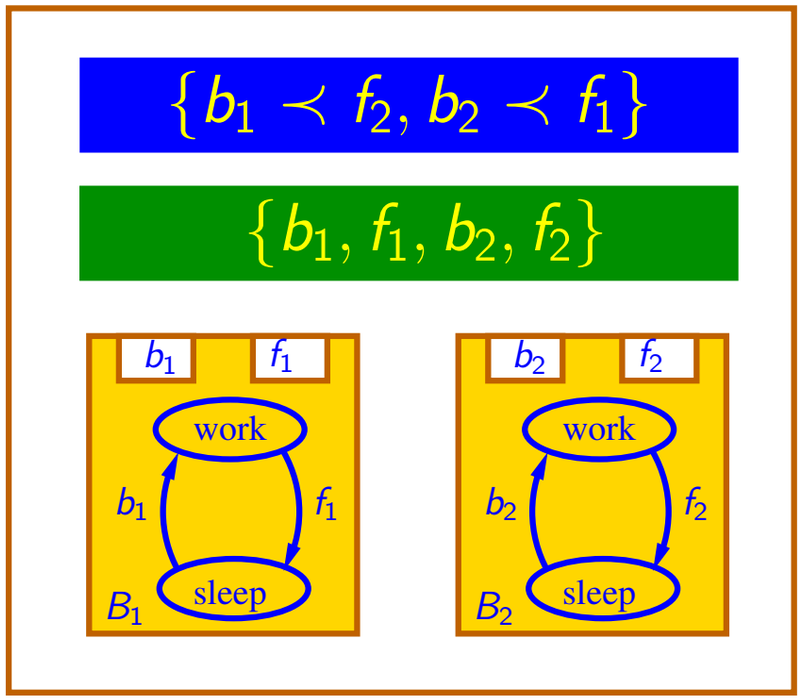


Restrictions from the interaction model

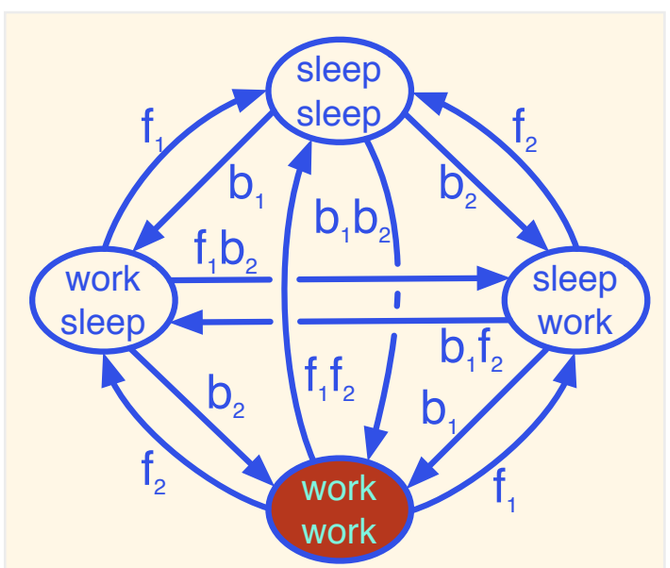


BIP composition example

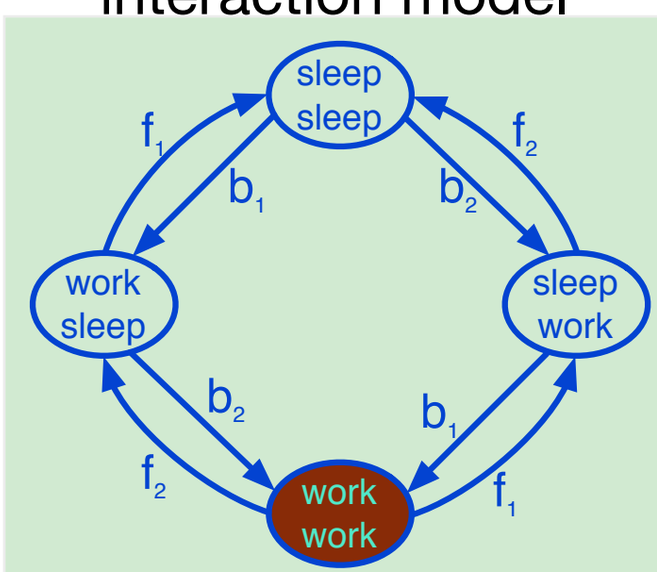
Safety property:
Mutual exclusion



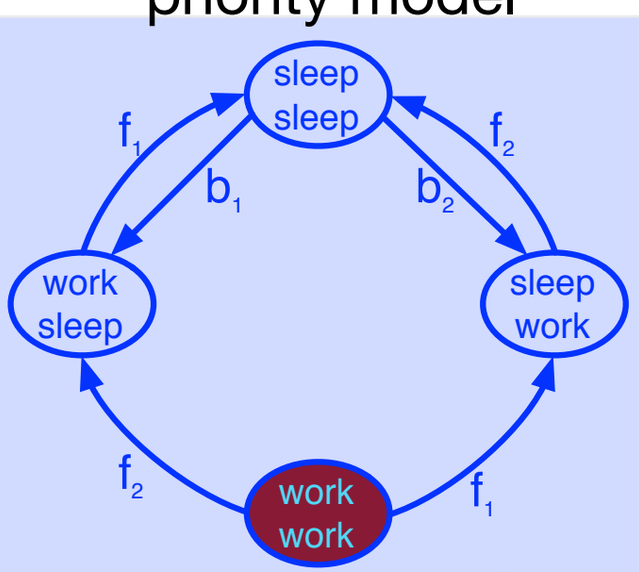
No restrictions



Restrictions from the interaction model



Restrictions from the priority model



Semantics: Interactions

Consider a set of *n* components, such that

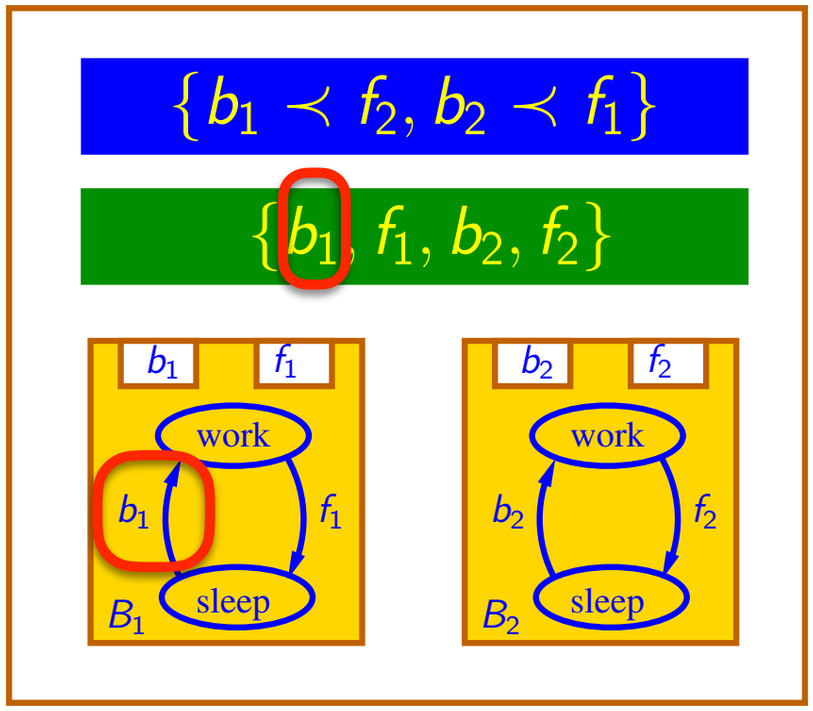
$$B_i = (Q_i, P_i, \to_i), \qquad \to_i \subseteq Q_i \times 2^{P_i} \times Q_i, \qquad P = \bigcup_i P_i$$

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 $a \in \gamma$.

Interaction semantics example



Component composition

The composition of components $\{B_i\}_{i=1}^n$, parameterized by a set of interactions $\gamma \subseteq 2^P$, is the transition system $B = (Q, P, \rightarrow_\gamma)$, where $Q = \bigotimes_{i=1}^n Q_i$ and $\rightarrow_\gamma$ is the least set of transitions satisfying the rule:

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

for each $a \in \gamma$. We write $B = \gamma(B_1, \dots, B_n)$.

Semantics: Priority

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

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

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

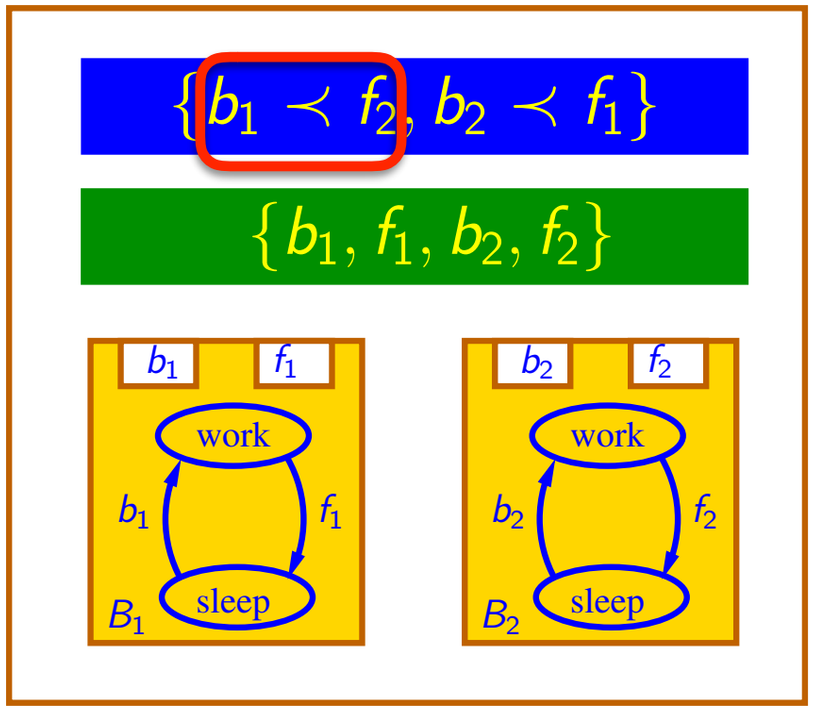
for each $a \in \gamma$.

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 $a \in 2^P$.

Priority semantics example



BIP priority model

Composability

Deadlock-freedom is preserved by priority orders: if B is deadlock-free then $(B, \prec)$ is deadlock-free for any priority order $\prec$.

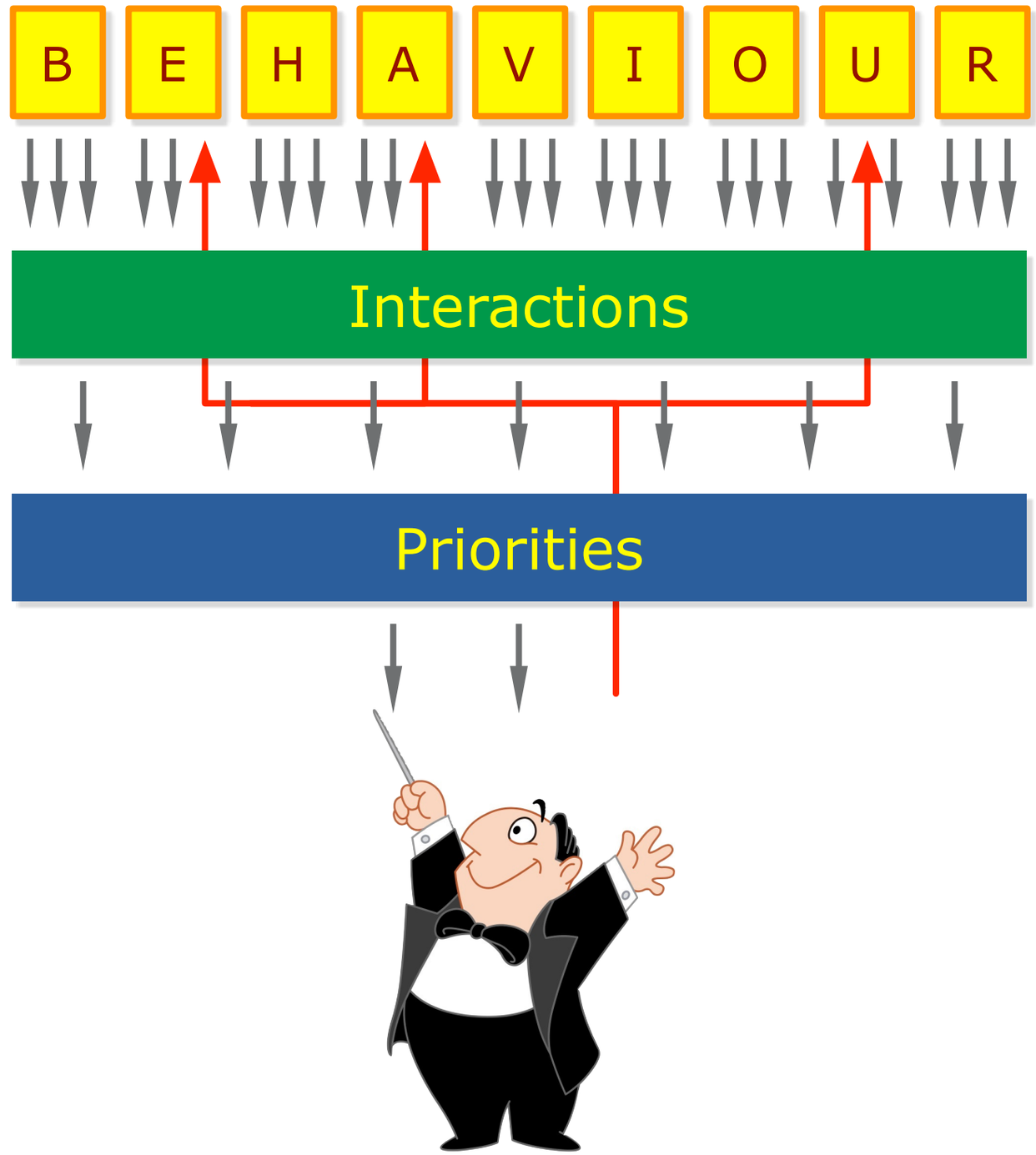
Compositionality

Deadlock-freedom is preserved by composition: if $(B_1, \prec^1)$ and $(B_2, \prec^2)$ are deadlock-free, then $(B_1, \prec^1) \| (B_2, \prec^2)$ is deadlock-free.

Engine-based execution

1. Components notify the Engine about enabled transitions.

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



Related publications

- Bliudze, Simon, and Joseph Sifakis. "The algebra of connectors—structuring interaction in BIP." IEEE Transactions on Computers 57.10 (2008): 1315-1330.

- Gössler, Gregor, and Joseph Sifakis. "Composition for component-based modeling." Science of Computer Programming 55.1-3 (2005): 161-183.

- Basu, Ananda, Bensalem Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. "Rigorous component-based system design using the BIP framework." IEEE software 28, no. 3 (2011): 41-48.