# Modeling Component-based Systems in BIP

ETR 2007

Nantes, Sept. 2007

Joseph Sifakis

VERIMAG

sifakis@imag.fr

In collaboration with G. Goessler (INRIA),
A. Basu, M. Bozga, H. Nguyen (Verimag)

# Component-based construction – Objectives

Develop a rigorous and general basis for real-time system design and implementation

- Concept of component and associated composition operators for incremental description and correctness by construction

- Concept for real-time architecture encompassing heterogeneity, paradigms and styles of computation e.g.
  - Synchronous vs. asynchronous execution
  - Event driven vs. time driven computation
  - Distributed vs. centralized execution

- Automated support for component integration and generation of glue code meeting given requirements

# Approches involving components

- Theory such as process algebras and automata
- SW Component frameworks, such as
  - Coordination languages extensions of programming languages : Linda, Javaspaces, TSpaces, Concurrent Fortran, NesC, BPEL
  - Middleware e.g. Corba, Javabeans, .NET
  - Software development environments: PCTE, SWbus, Softbench, Eclipse
- System modeling languages: SystemC, Statecharts, UML, Simulink/Stateflow, Metropolis, Ptolemy
- Architecture Description Languages focusing on non-functional aspects e.g. AADL

*Lack of*

- *frameworks treating interactions and system architecture as first class entities that can be composed and analyzed (usually, interaction by method call)*
- *rigorous models for behavior and in particular aspects related to time and resources.*

# Sources of heterogeneity [Henzinger&Sifakis FM06]

Heterogeneity of interaction

- Atomic or non atomic
- Rendezvous or Broadcast
- Binary or n-ary

Heterogeneity of execution

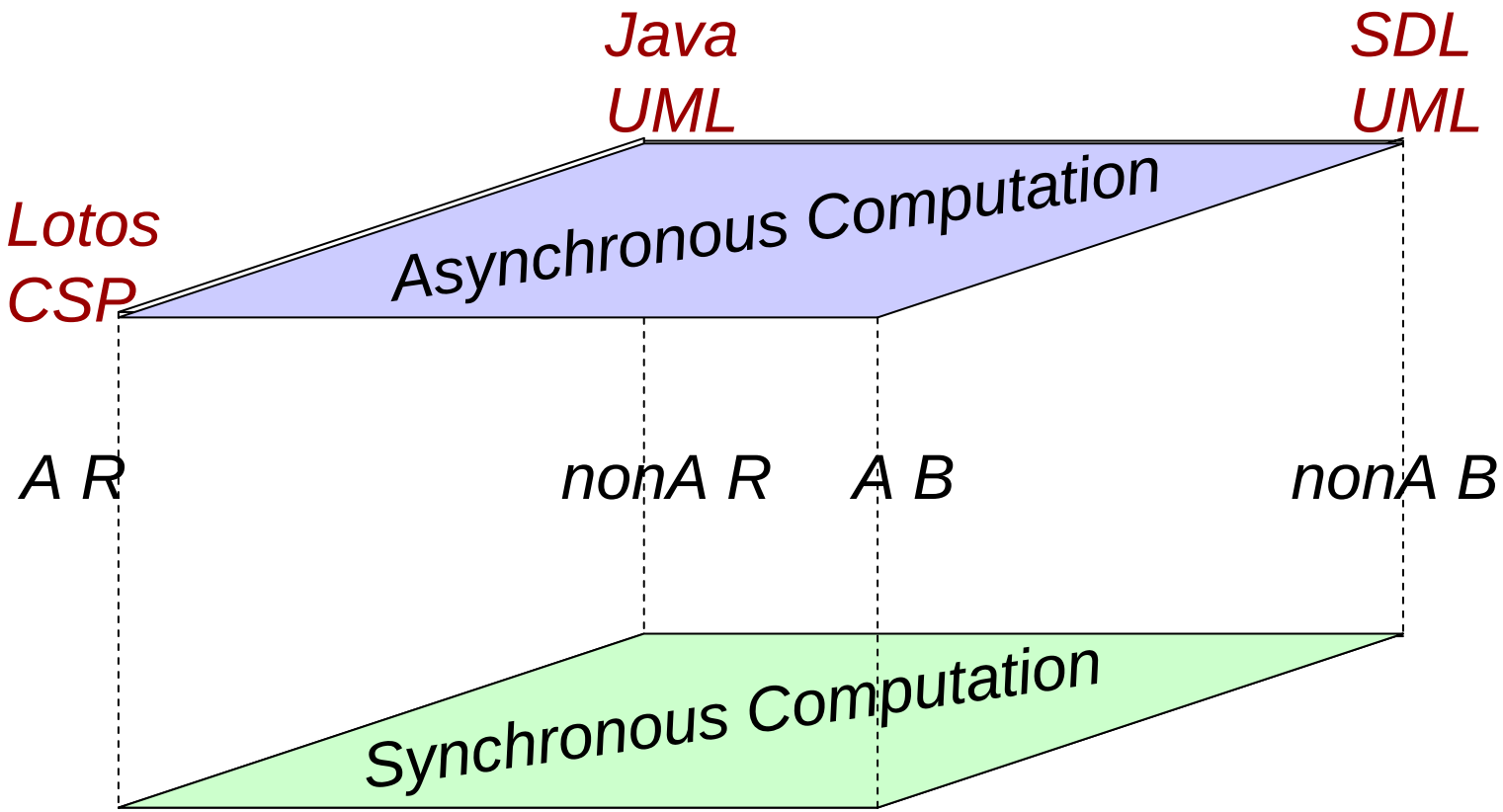
- Synchronous execution
- Asynchronous execution
- Combinations of them

Heterogeneity of abstraction e.g. granularity of execution

# Sources of Heterogeneity - Example

*A: Atomic interaction*

*R: Rendezvous*
*B: Broadcast*



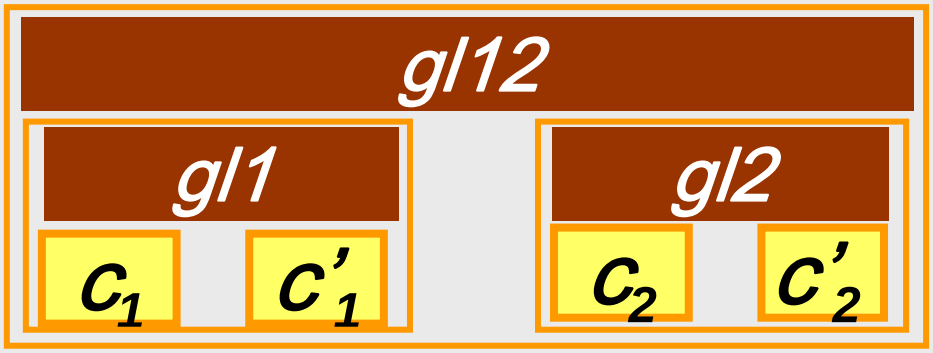
# Overview

- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# Component-based Construction: The Problem

Pb: Build a component *C* satisfying a given property *P, from*

- $\mathcal{C}_0$ a set of atomic components
- $\mathcal{GL} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of *glue operators* on components



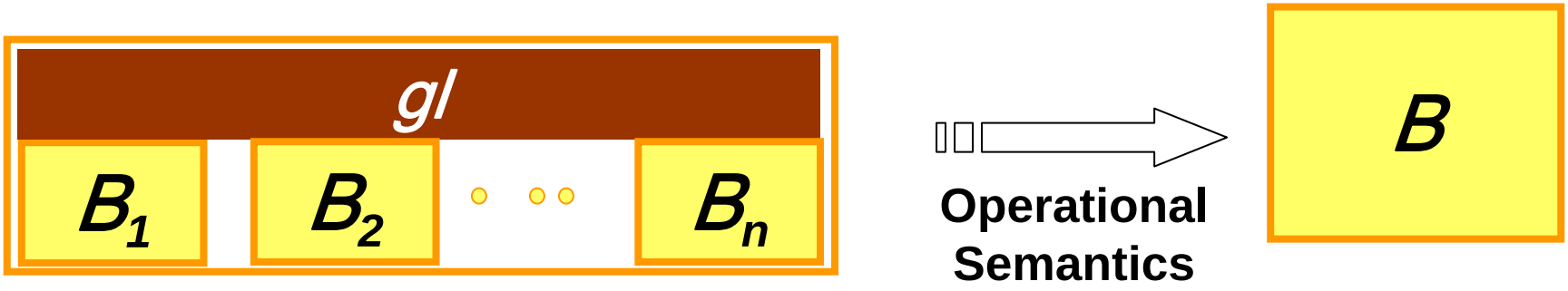

sat P

Glue operators

- model mechanisms used for communication and control such as protocols, schedulers, buses
- restrict the behavior of their arguments, that is the projection of the behavior of $gl(C_1, C_2, .., C_n)$ on actions of $C_1$ is contained in the behavior of $C_1$

# Component-based Construction: Formal Framework

Operational Semantics: the meaning of a compound component is an atomic component

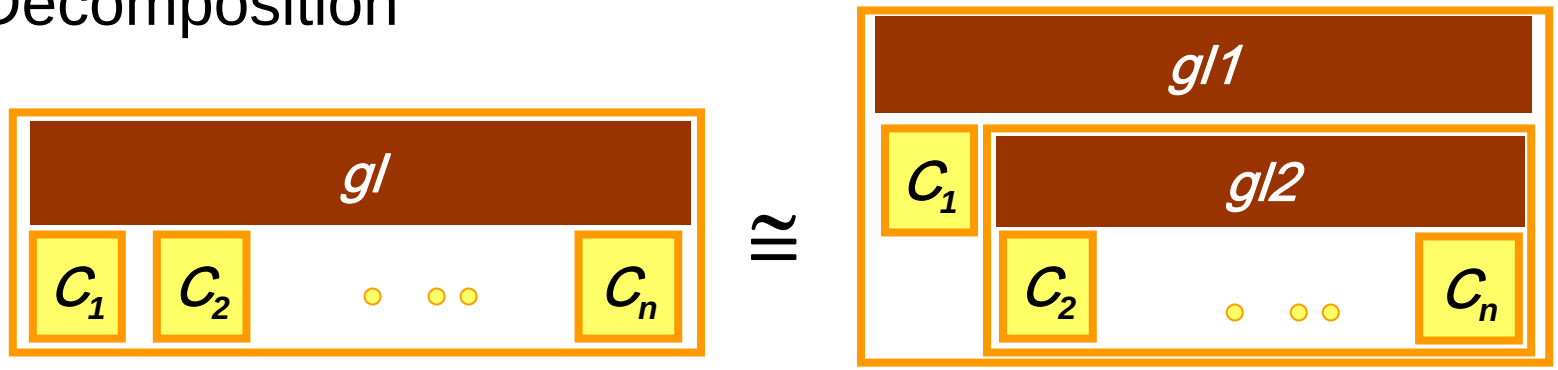



Algebraic framework:

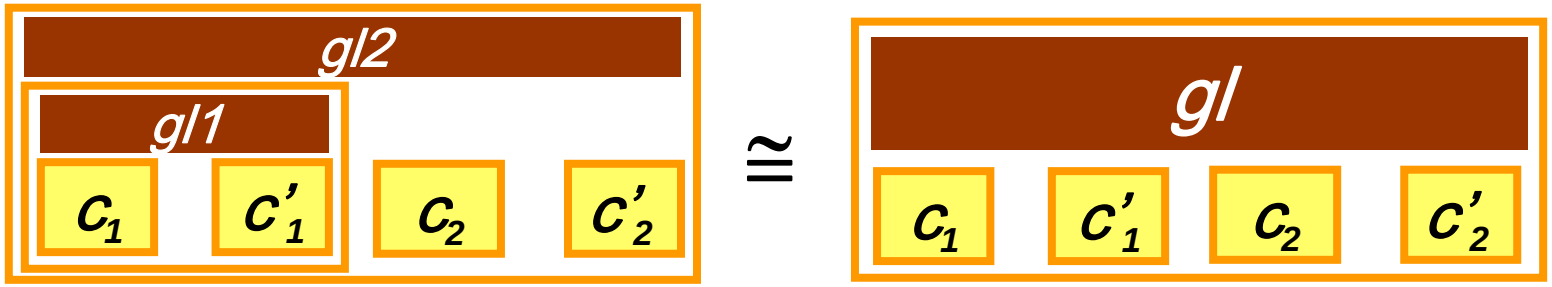
- Components are terms of an algebra of terms $(\mathcal{C}, \cong)$ generated from $\mathcal{C}_0$ by using operators from $\mathcal{GL}$
- $\cong$ is a congruence compatible with operational semantics

# Component-based Construction: Constructivity – Incremental description

## 1. Decomposition




## 2. Flattening



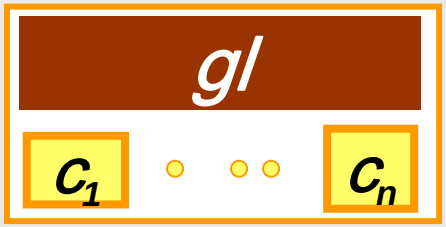

Flattening can be achieved by introducing an idempotent operation $\oplus$ such that (GL, $\oplus$) is a commutative monoid and

$gl(gl'(C_1, C_2, .., C_n)) \cong gl \oplus gl'(C_1, C_2, .., C_n)$

# Component-based Construction: Constructivity – Compositionality

Build correct systems from correct components: rules for proving global properties from properties of individual components

$c_i$ sat $P_i$ implies $\forall gl\ \exists \widetilde{gl}$

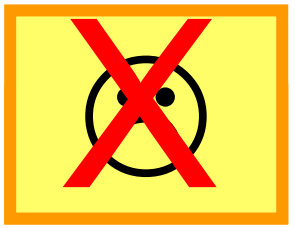



sat $\widetilde{gl}(P_1, .., P_n)$

*We need compositionality results about preservation of progress properties such as deadlock-freedom and liveness and extra-functional properties*

# Component-based Construction: Constructivity – Composability

*Make the new without breaking the old: Rules guaranteeing non interference of solutions*




sat P and


sat P'

implies


sat $P \wedge P'$

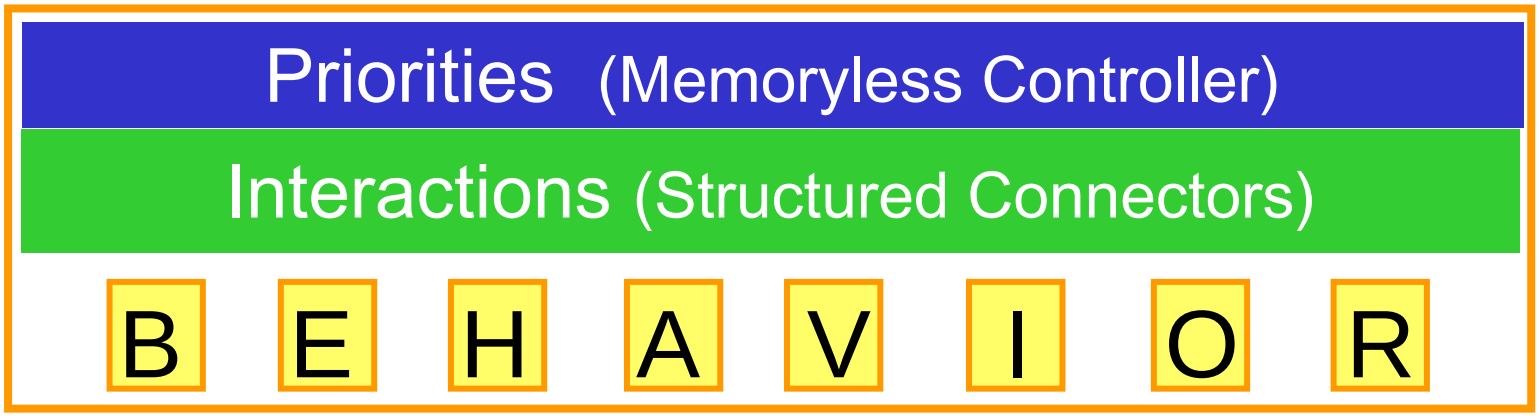
*Property stability phenomena are poorly understood.*
*We need composability results e.g. feature interaction in middleware, composability of scheduling algorithms, theory for reconfigurable systems*

# Overview

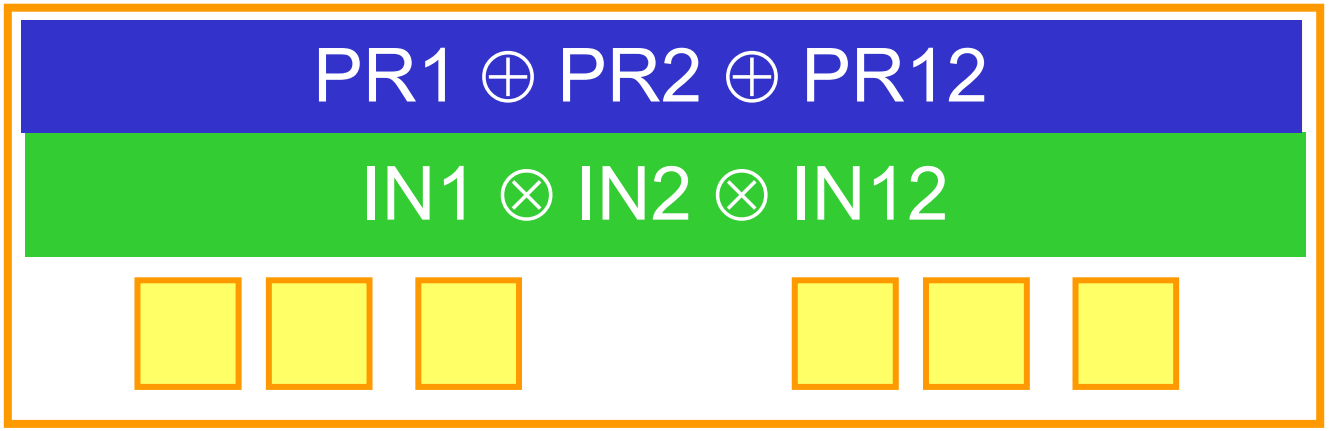
- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# BIP: Basic Concepts

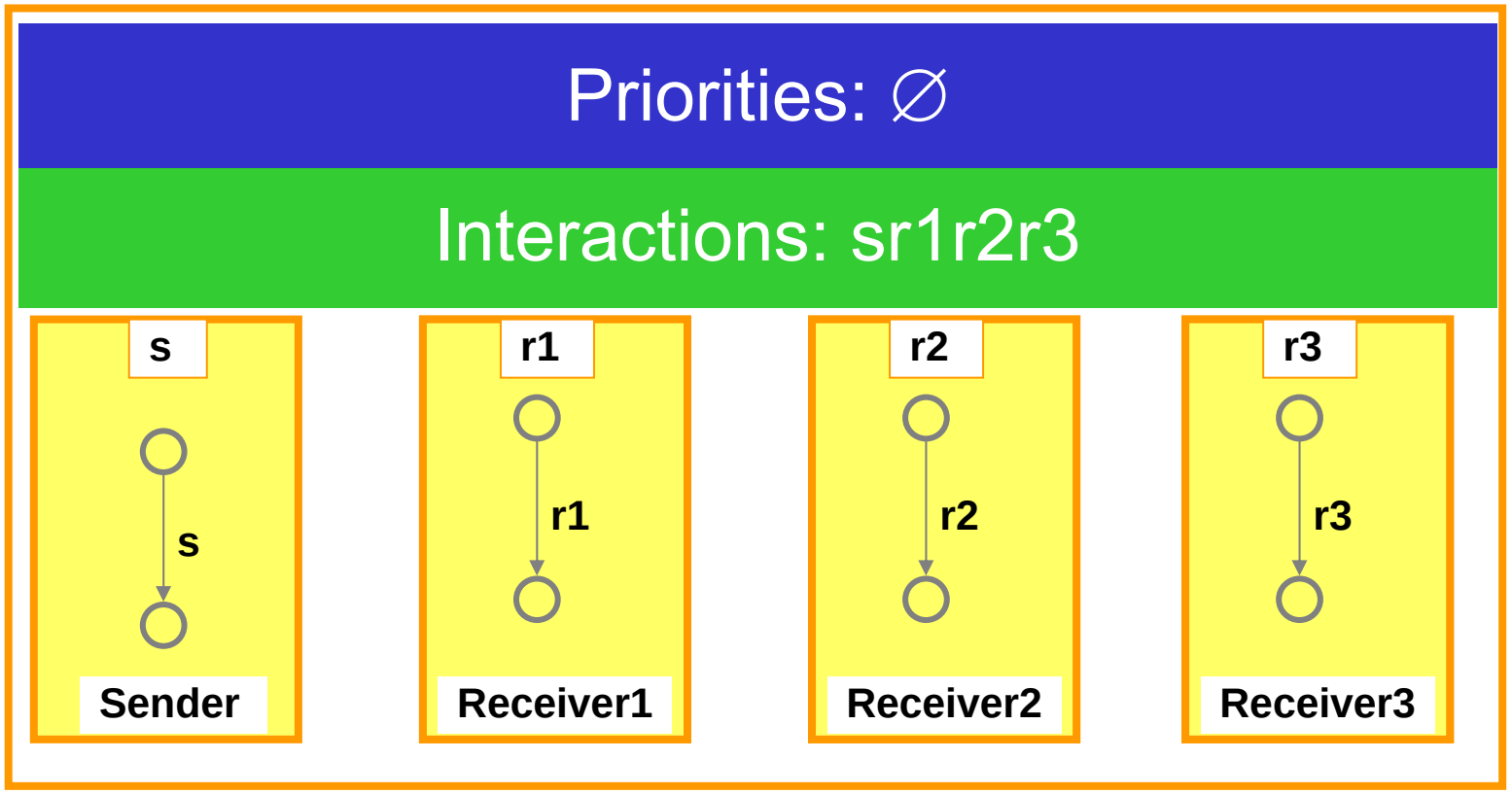
Layered component model




Composition (incremental description)

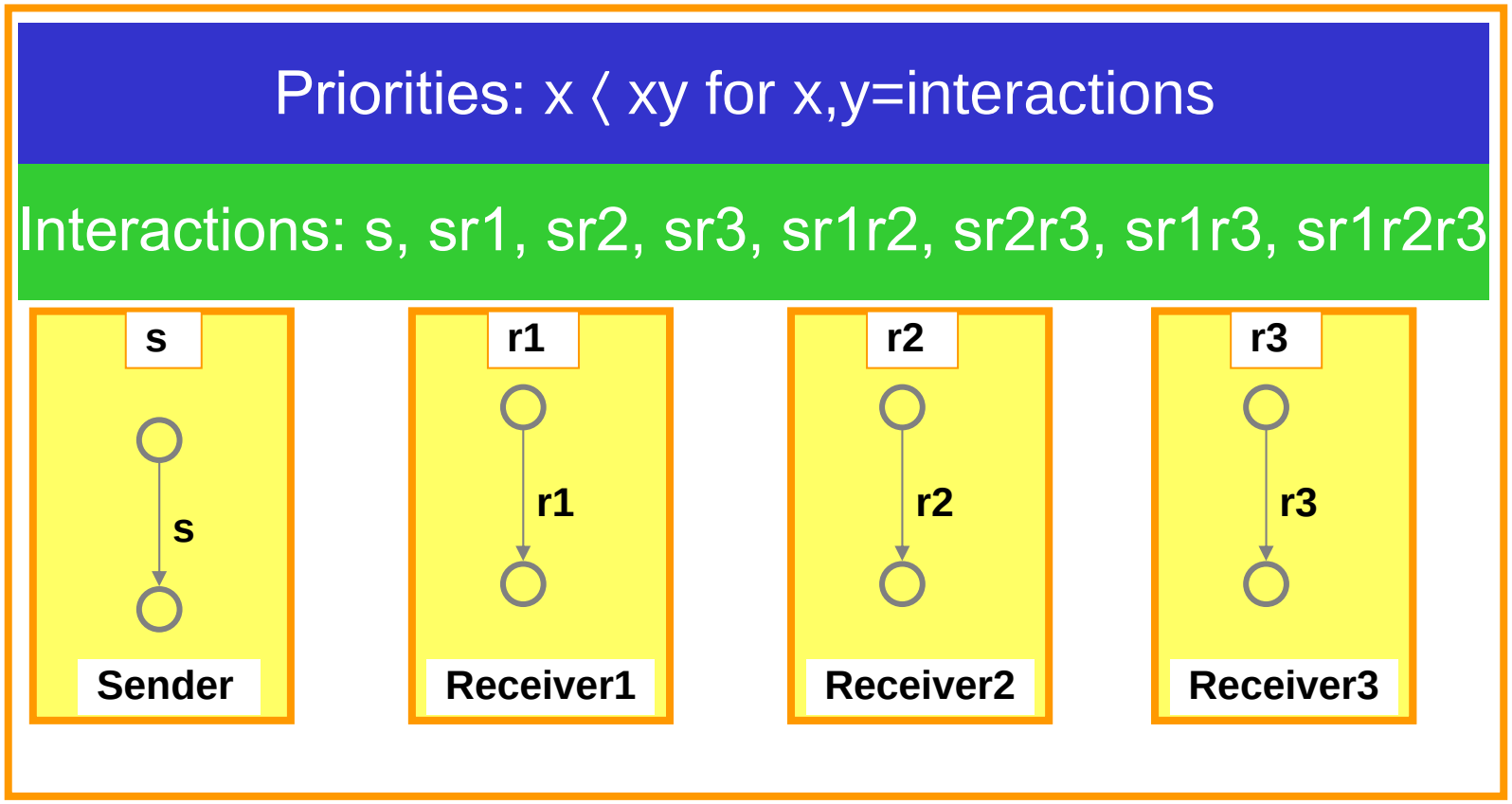


# BIP: Basic Concepts



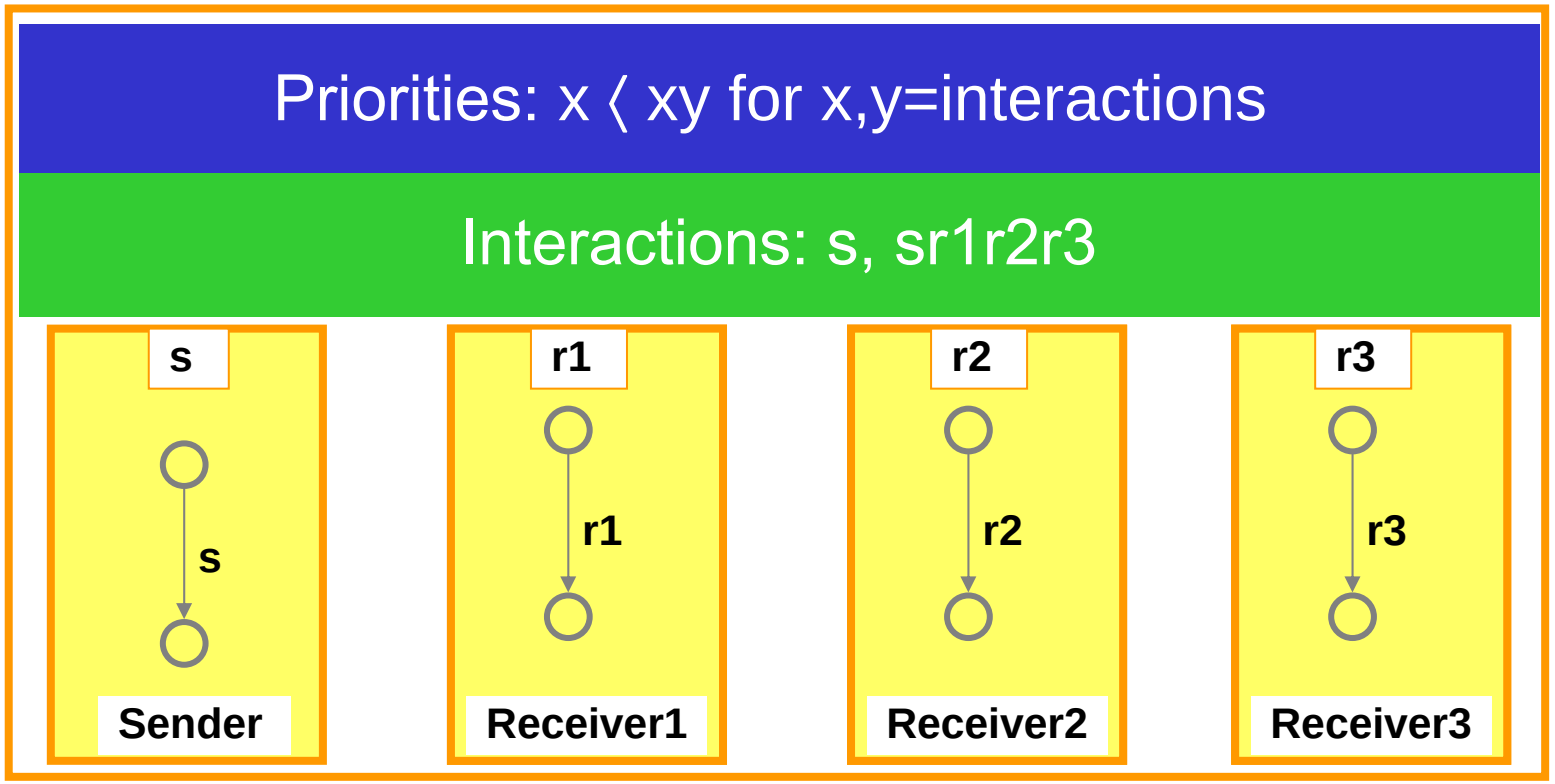

Rendezvous

# BIP: Basic Concepts



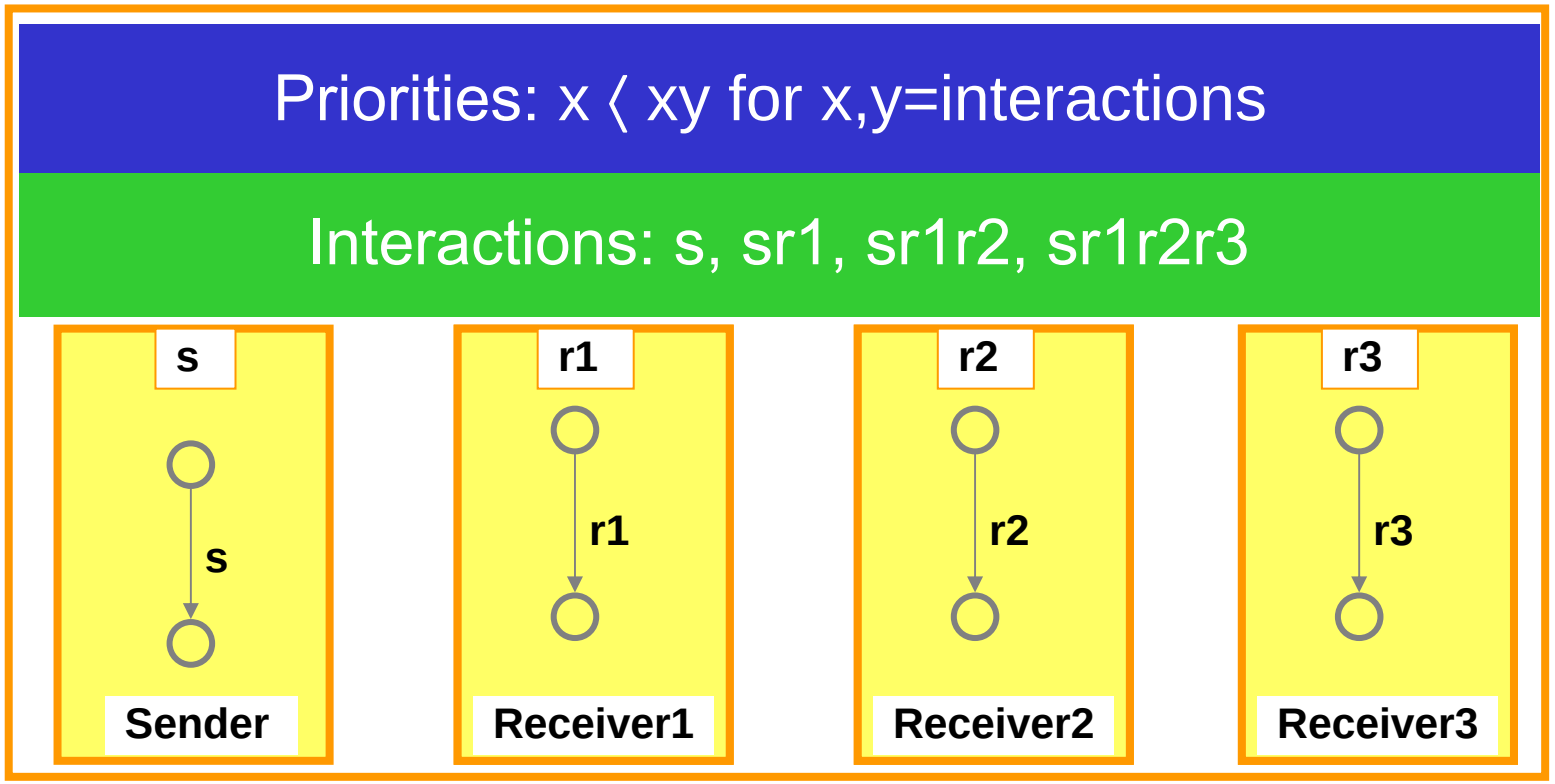

Broadcast

# BIP: Basic Concepts




Atomic Broadcast

# BIP: Basic Concepts




Causal Chain

# BIP: Basic Concepts - Semantics for Interactions

Given

- a set of atomic components $\{B_i\}_{i=1..n}$ where $B_i=(Q_i, P_i, \rightarrow_i)$
- a set of interactions $\gamma \in 2^P$ where $P=\cup_{i=1..n} P_i$

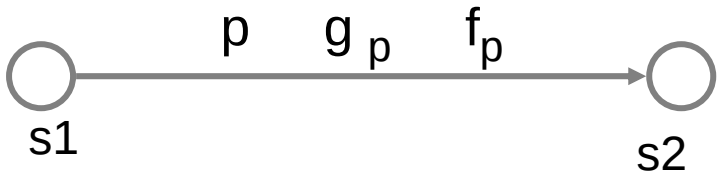
Define $\gamma(B_1,.., B_n)=(Q, P, \rightarrow_\gamma)$ where $Q=\times_{i=1..n} Q_i$ by the rule

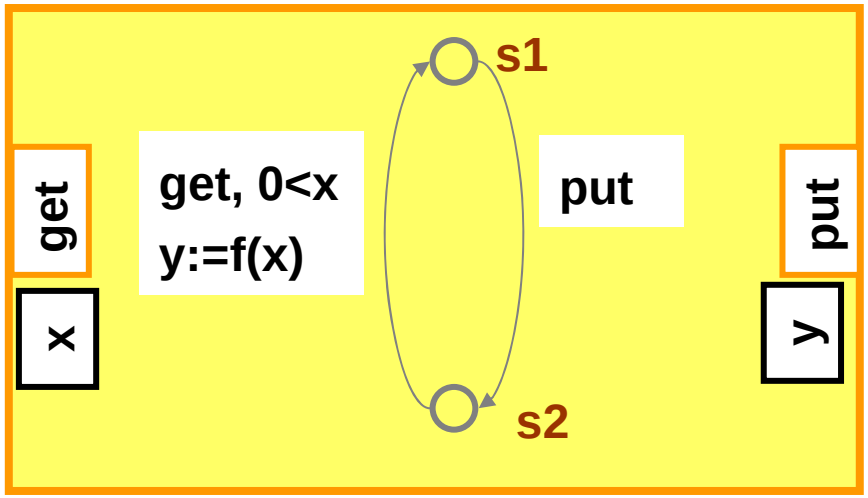
$$\frac{a=\{p_i\}_{i\in I} \in \gamma \wedge \forall i \in I\ (p_i \in P_i \wedge q_i - p_i \rightarrow_i q'_i)}{(q_1 ,.., q_n) - a \rightarrow_\gamma (q'_1 ,.., q'_n) \text{ where } q'_l = q_l \text{ if } i \notin I}$$

# BIP: Basic Concepts - Behavior Modeling

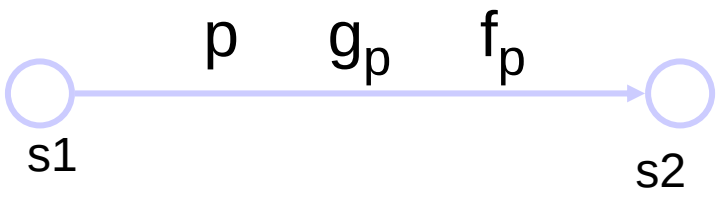
An atomic component has

- A set of ports P
- A set of control locations S
- A set of variables V
- A set of transitions of the form
  - p is a port
  - $g_p$ is a guard, boolean expression on V
  - $f_p$ is a function on V (block of code)



# BIP: Basic Concepts - Behavior Modeling




p: a port through which interaction is sought
$g_p$: a pre-condition for interaction through p
$f_p$ : a computation (local state transformation)

**Semantics:** interaction followed by computation

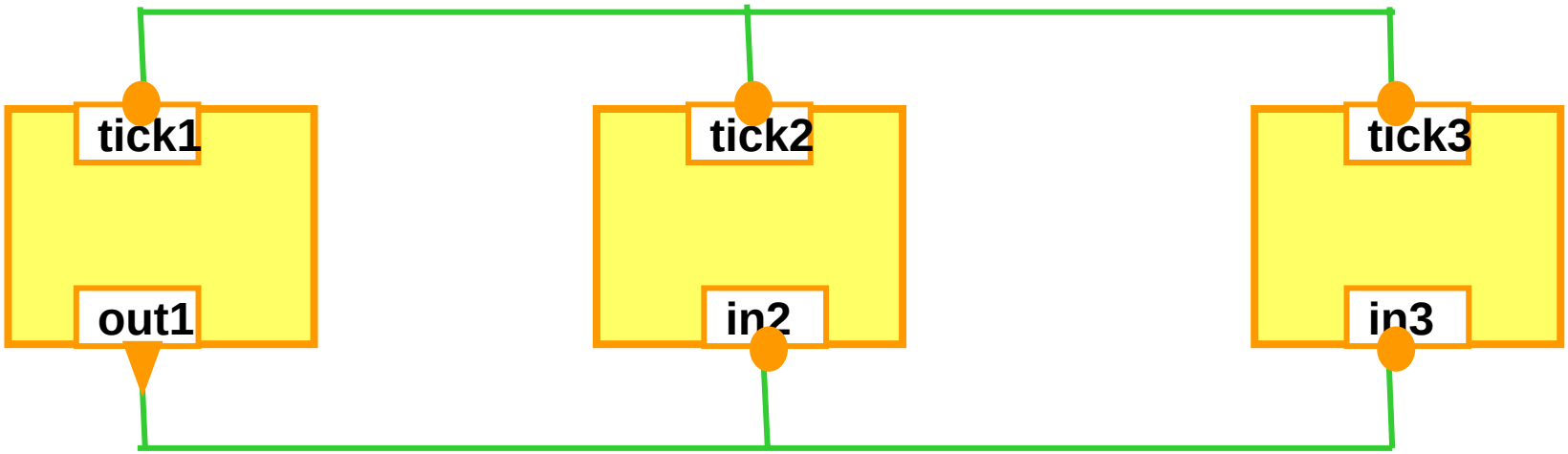
- A transition is enabled if $g_p$ is true and some interaction involving p is possible
- The execution of the enabled transition involves the execution of an interaction involving p followed by the execution of $f_p$

# Overview

- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# Interaction Modeling: Connectors

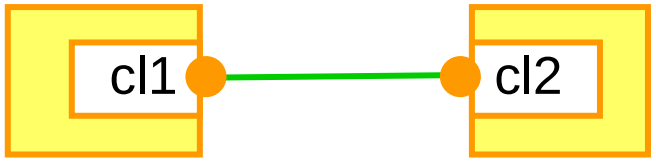
- A ***connector*** is a set of ports which can be involved in an interaction
- Port attributes **(*trigger*** ▼**, *synchron*** ●**)** are used to model rendezvous and broadcast.
- An ***interaction*** of a connector is a set of ports such that: either it contains some trigger or it is maximal.

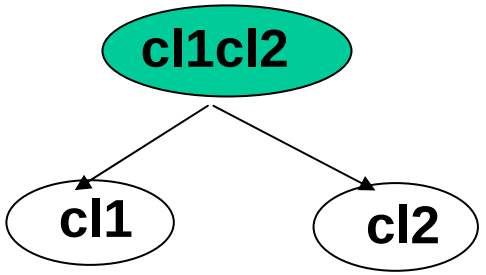


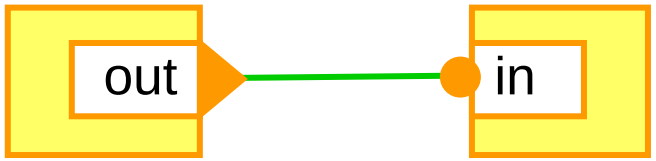

Interactions:

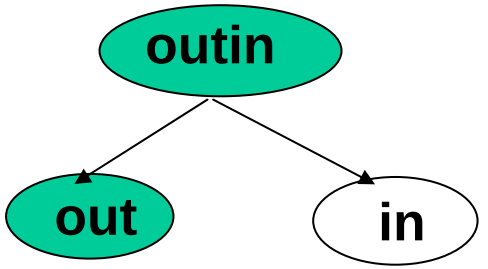
tick1tick2tick3, out1, out1in2, out1in3, out1in2in3

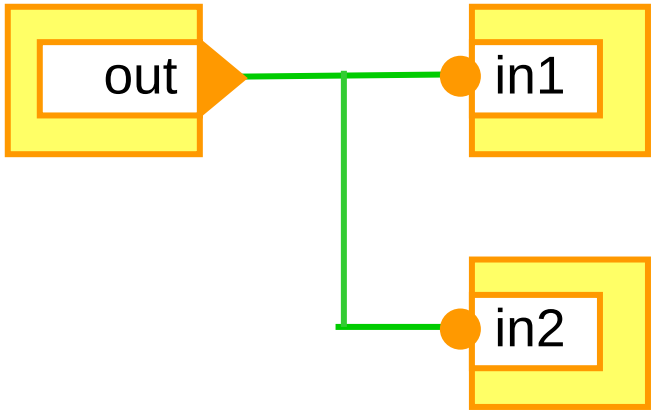
# Interaction Modeling: Connectors

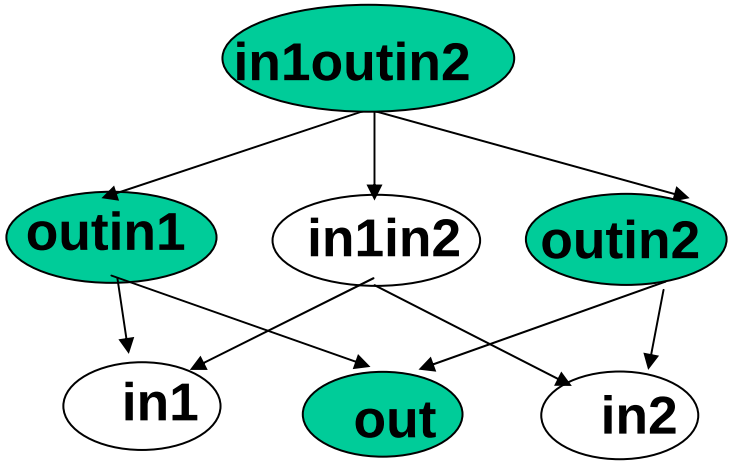


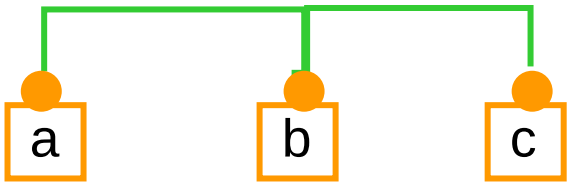



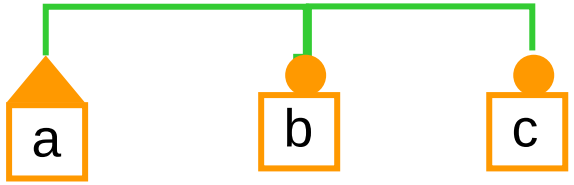



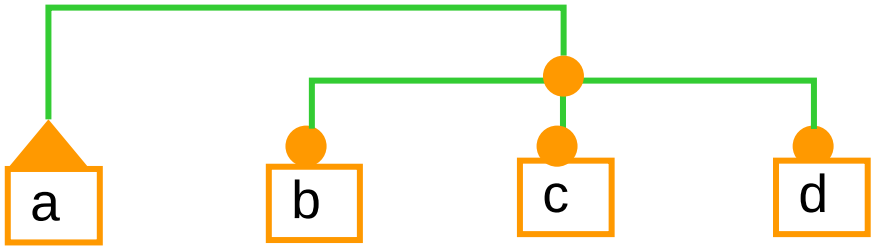
# Interaction Modeling: Hierarchical Connectors



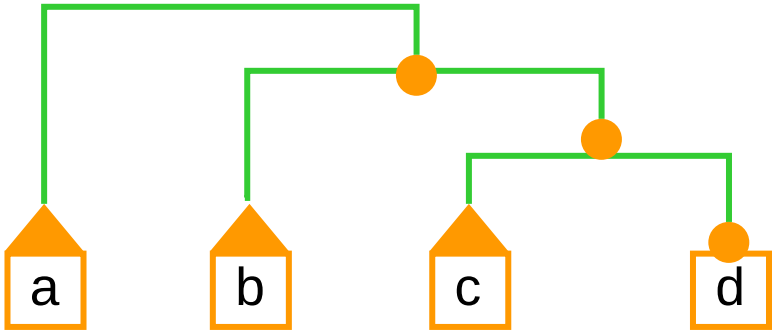

Rendezvous: abc




Broadcast:
a(1+b)(1+c)=a+ab+ac+abc




Atomic Broadcast:
a(1+ bcd) =a + abcd




Causal chain:
a(1+b(1+c(1+d)))
=a+ab+abc+abcd

# Interaction Modeling: The Algebra of Connectors

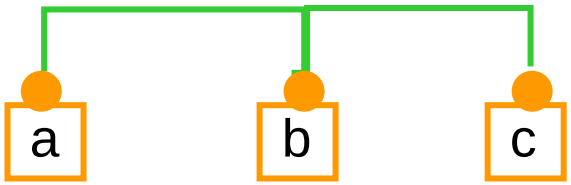
**Syntax:** For P be a set of ports, such that $0,1 \notin P$, AC(P) is defined by

s ::= [0] | [1] | [p] |[x]  (synchrons)
t ::= [0]' | [1]' | [p]' | [x]'  (triggers)
x ::= s | t | x.x | x + x

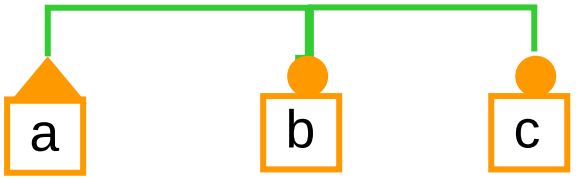
where

- [ ], [ ]' are unary *typing* operators
- . is the *fusion* operator which is associative, commutative, [1] is an identity element and [0] is an absorbing element
- + is the *union* operator.

# Interaction Modeling: Hierarchical Connectors

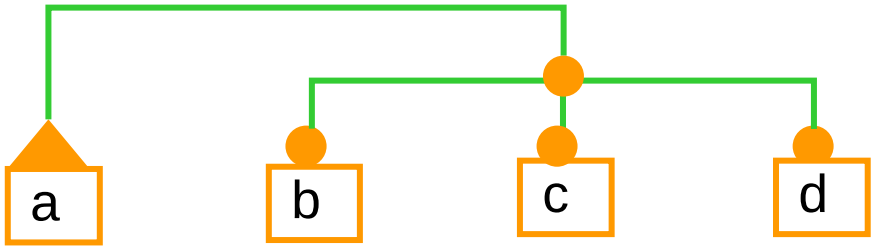



Rendezvous: abc



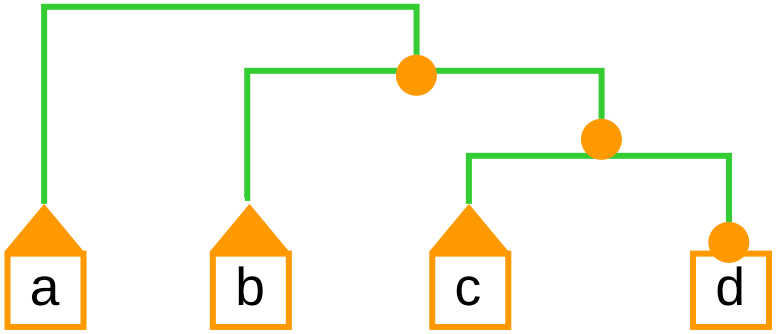

Broadcast:

a’bc= a(1+b)(1+c)




Atomic Broadcast:

a’[bcd] =a(1+ bcd)



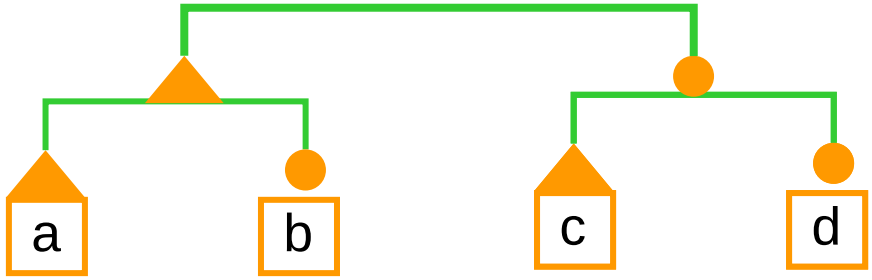

Causal chain:

a’[b’[c’d]]=a(1+b(1+c(1+d)))

# Interaction Modeling: The Algebra of Connectors - Fusion vs. Typing

For two connectors $\gamma_1$=a'.b and $\gamma_2$=c'.d

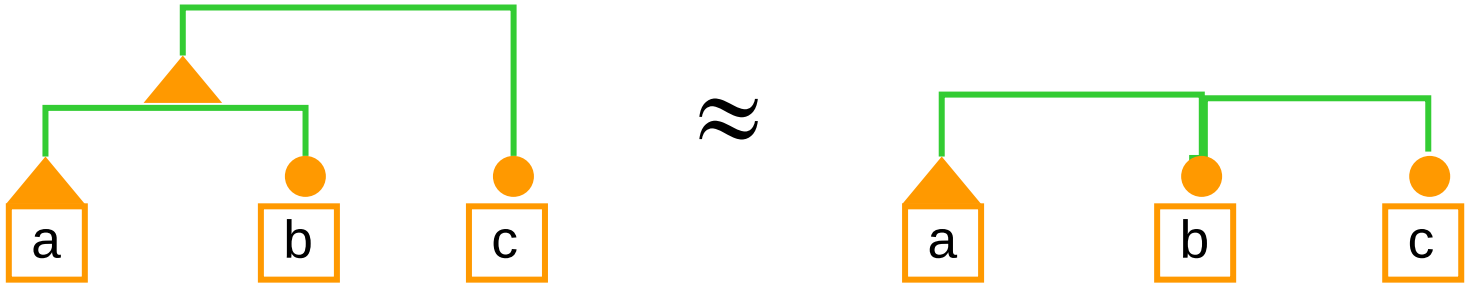

$\gamma_1.\gamma_2$=a'bc'd= a'bcd + abc'd




$[\gamma_1]'.[\gamma_{21}]$=[a'b]'.[c'd]

# Interaction Modeling:
## The Algebra of Connectors - Equivalence vs.Congruence

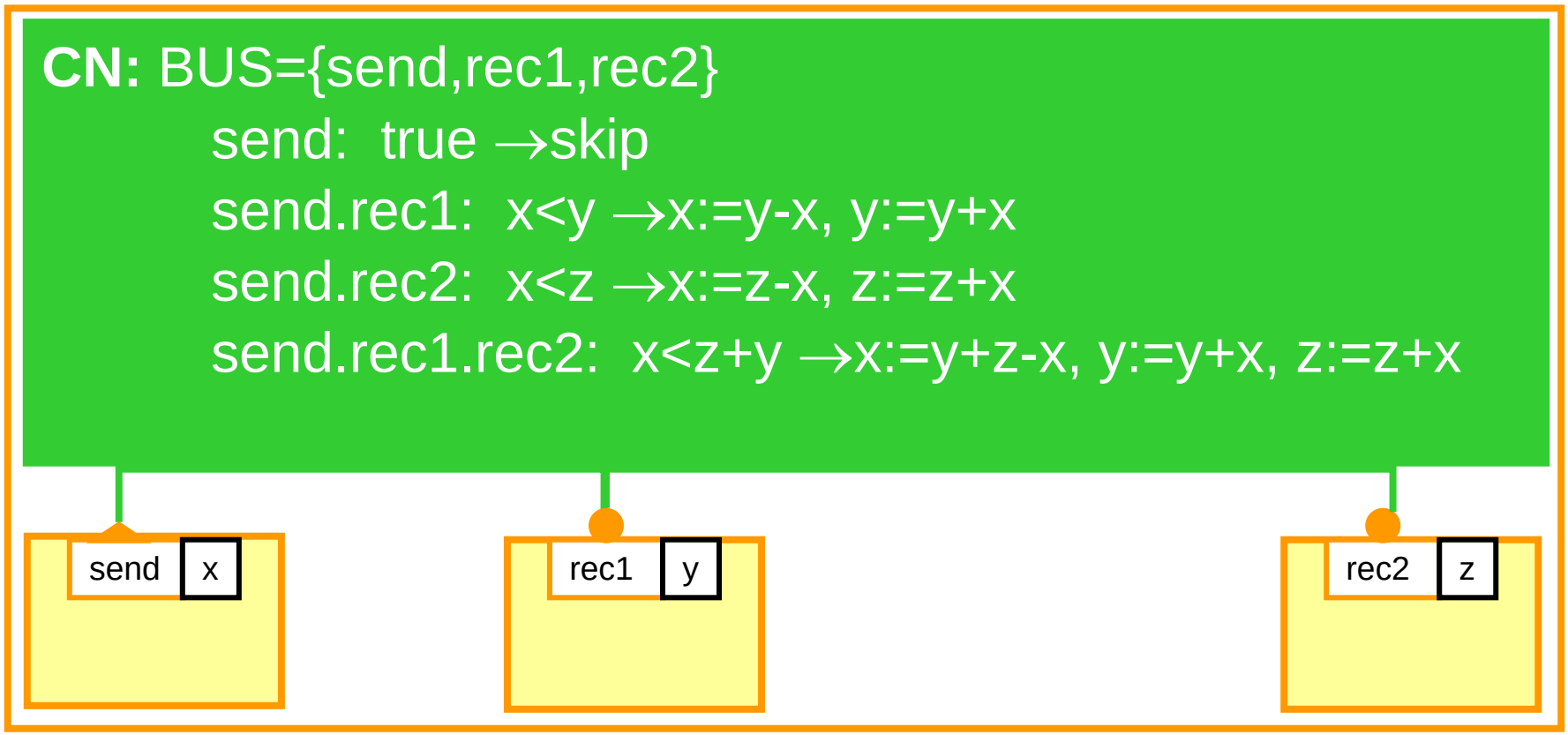
Definition: $\gamma_1 \approx \gamma_2$ if $\gamma_1$ and $\gamma_2$ represent the same set of interactions




Remark: $\approx$ is not a congruence as it is not preserved by fusion, for example a'b $\approx$ a+ab and a'bc $\not\approx$ ac+abc

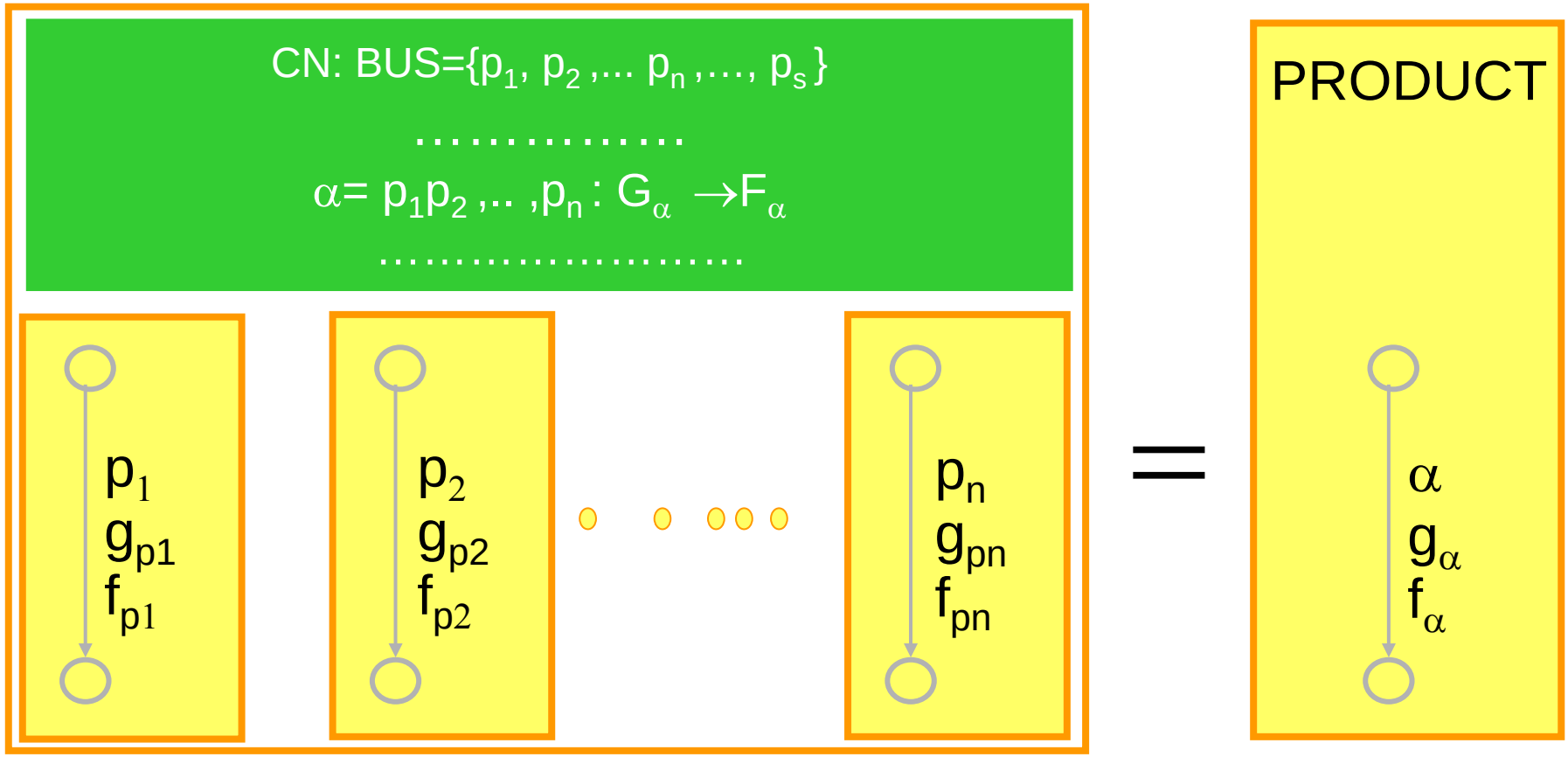
Semantics in terms of causality trees preserving the causality relation induced by triggers

# Interaction Modeling: Composition - Data Transfer



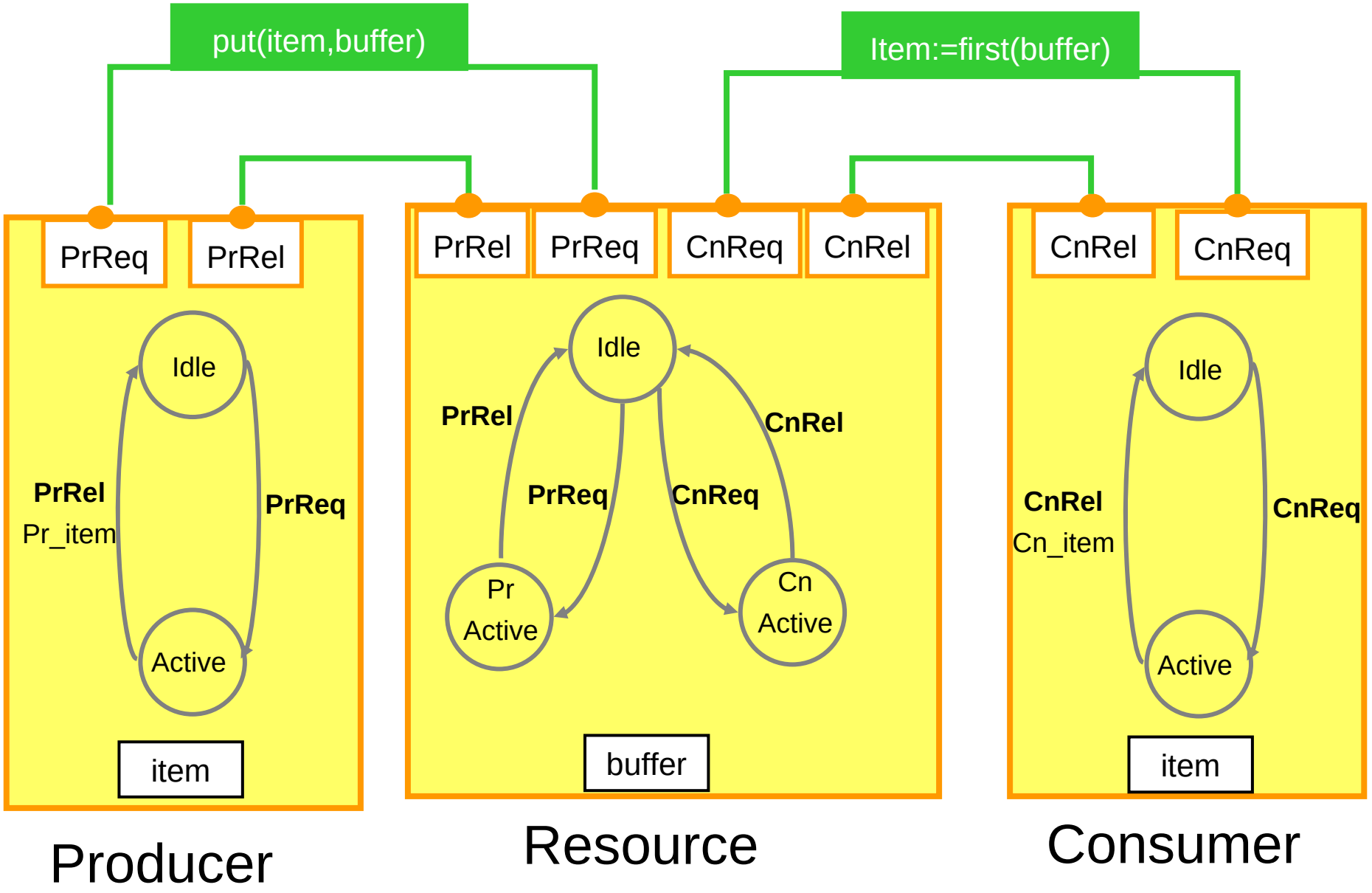

**Maximal progress:** execute a maximal enabled interaction

# Interaction Modeling: Composition - Semantics



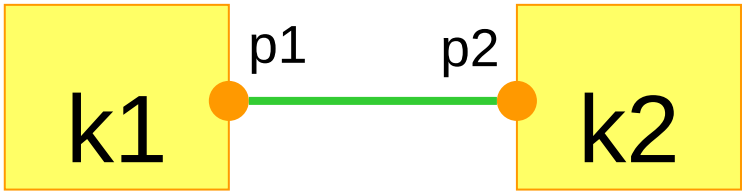

$g_\alpha = g_{p1} \wedge g_{p2} \wedge \ \ldots \wedge g_{pn} \wedge G_\alpha$ $\qquad$ $f_\alpha = F_\alpha; f_{p1}, f_{p2}, \ldots, f_{pn}$

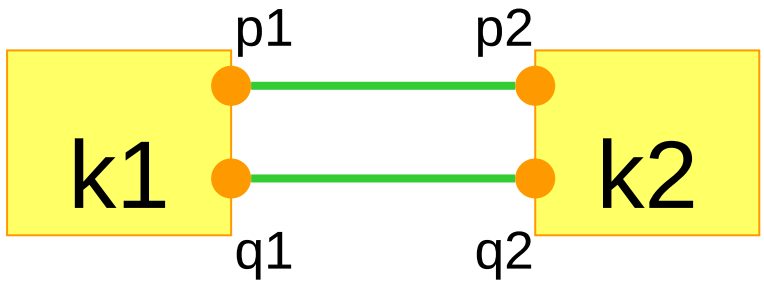
# Interaction Modeling: Producer-Consumer



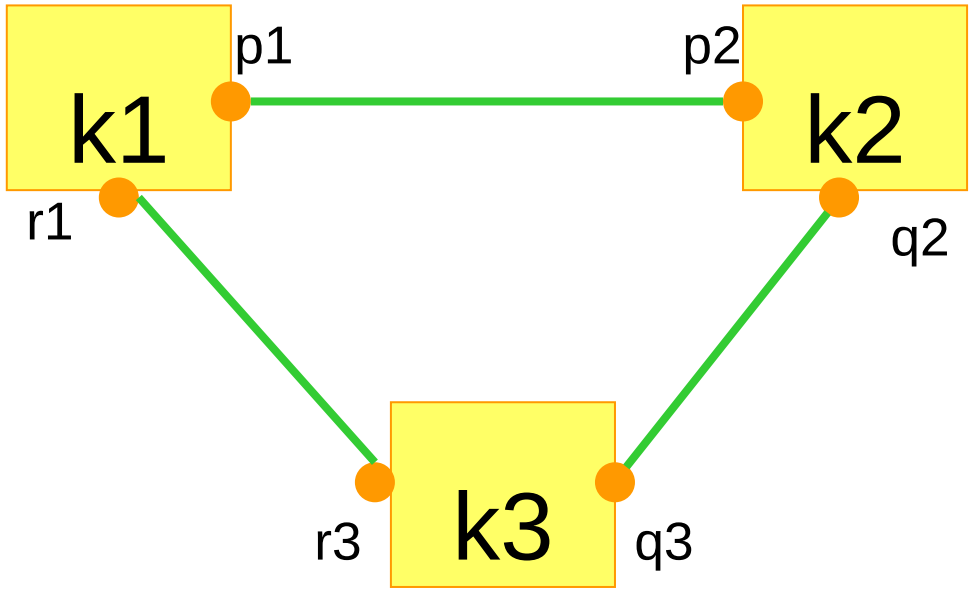
# Interaction Modeling: Checking for Deadlock-freedom

For K1,K2,K3 deadlock-free components




$en(p1) \wedge \neg en(p2) \wedge \neg en(q1) \wedge en(q2)$




$en(p1) \wedge \neg en(p2) \wedge en(q2) \wedge \neg en(q3) \wedge en(r3) \wedge \neg en(r1)$

# Overview

- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# Priorities

Priorities are a powerful tool for restricting non-determinism

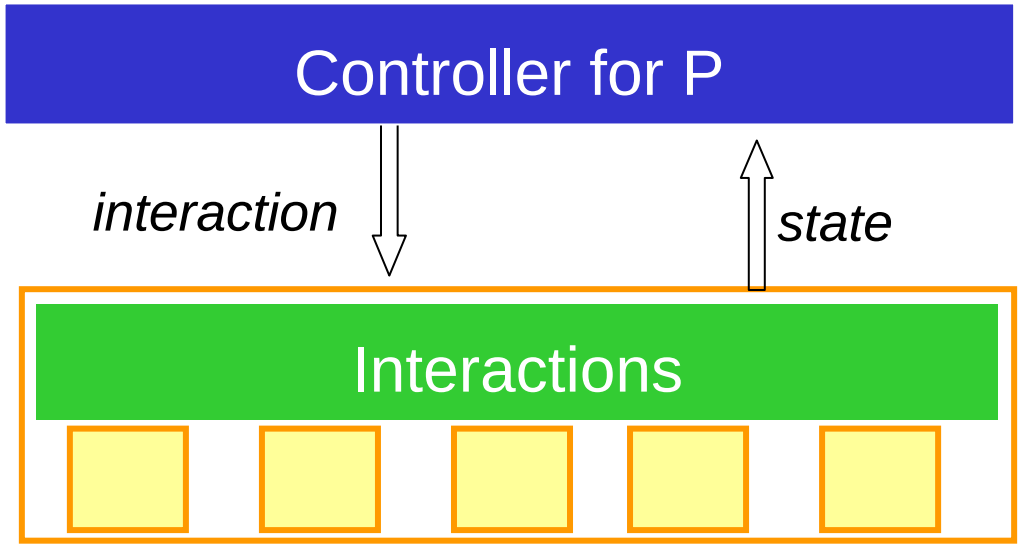
- they allow straightforward modeling of urgency and scheduling policies for real-time systems

- run to completion and synchronous execution can be modeled by assigning  priorities to threads

- they can advantageously replace (static) restriction of process algebras

# Priorities: Priorities as Controllers

A controller restricts the non determinism of system S to enforce  a property P

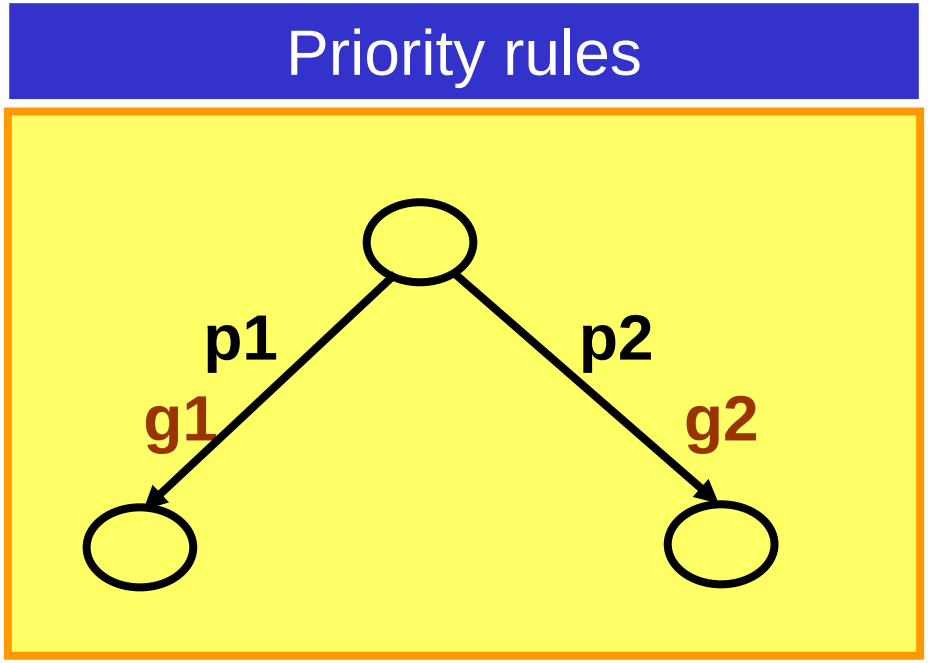



Results [Goessler&Sifakis, FMCO2003] :

- Restrictions induced by controllers enforcing deadlock-free state invariants can be described by dynamic priorities

- Conversely, for any restriction induced by dynamic priorities there exists  a controller enforcing a  deadlock-free control invariant

# Priorities: Definition

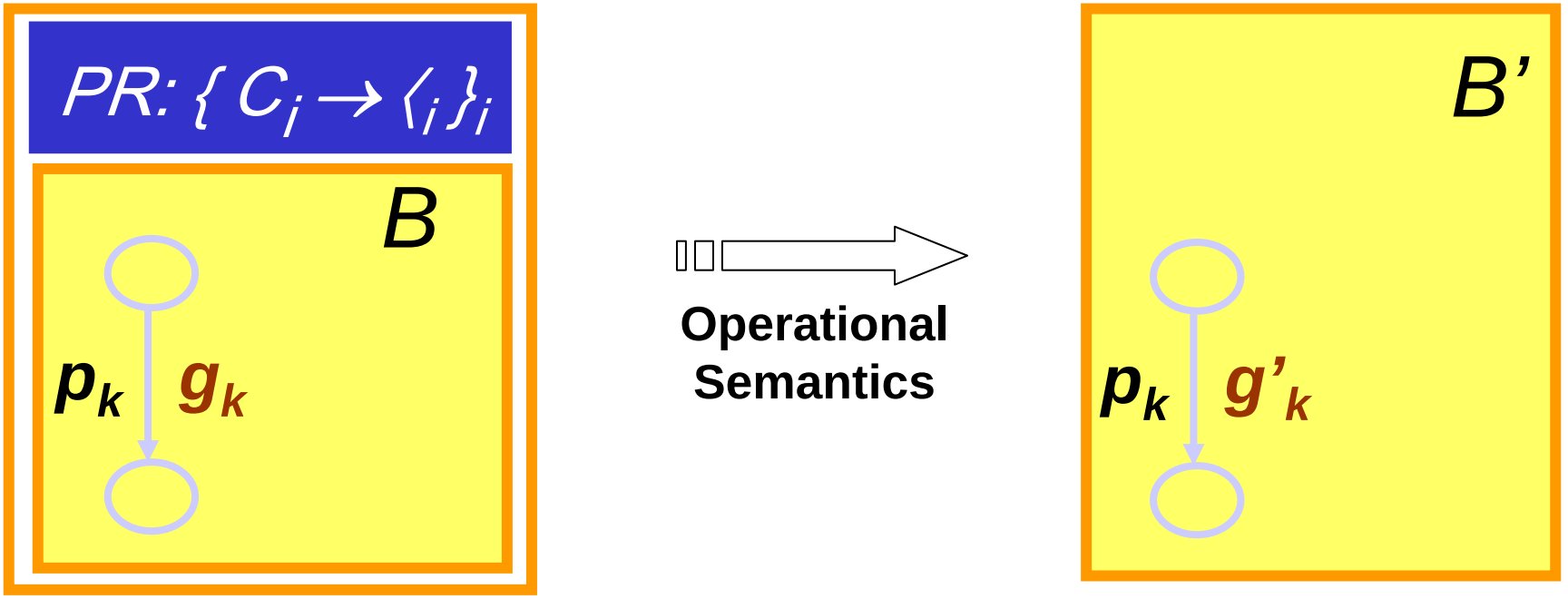



| Priority rule | Restricted guard g1’ |
|---|---|
| true $\rightarrow$ p1 $\langle$ p2 | g1’ = g1 $\wedge \neg$ g2 |
| C $\rightarrow$ p1 $\langle$ p2 | g1’ = g1 $\wedge \neg$(C $\wedge$ g2 ) |

# Priorities: Definition

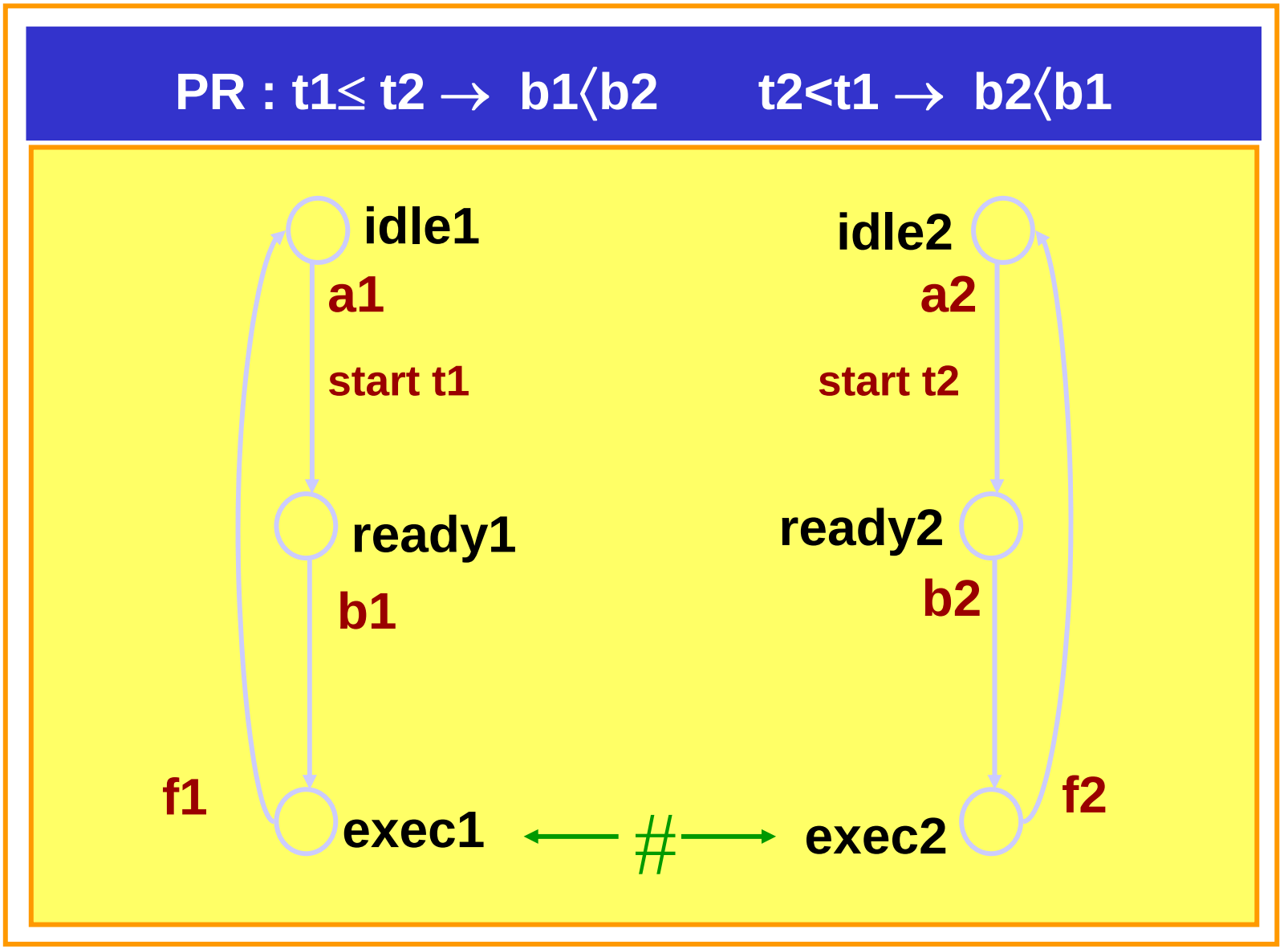
A *priority order* is a strict partial order $\prec \subseteq Inter \times Inter$

A set of *priority rules*, $PR = \{ C_i \rightarrow \prec_i \}_i$ where $\{C_i\}_i$ is a set of disjoint state predicates

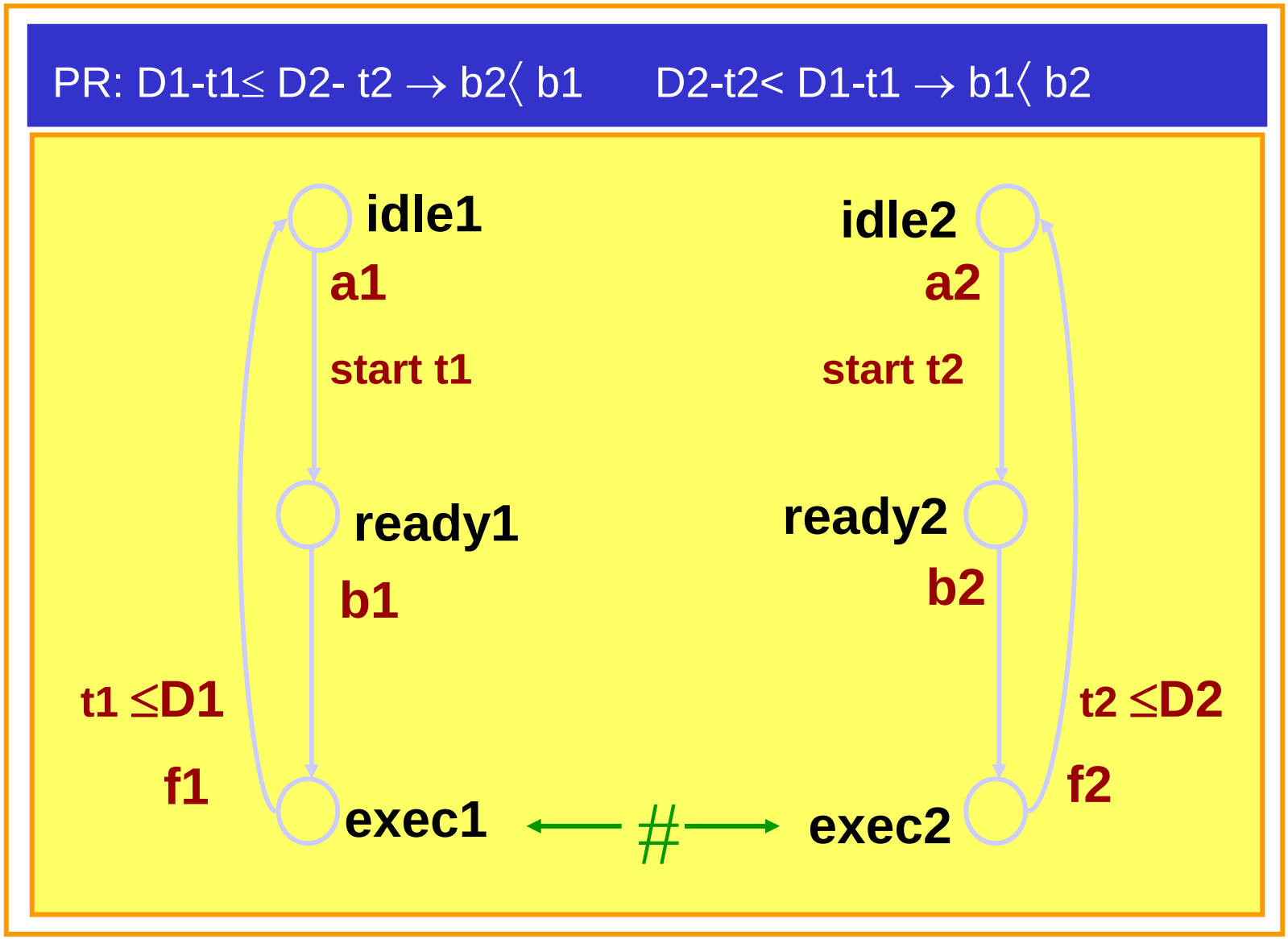



$$g'_k = g_k \wedge \bigwedge_{C \rightarrow \prec \in PR} (C \Rightarrow \bigwedge_{pk \prec pi} \neg g_i)$$

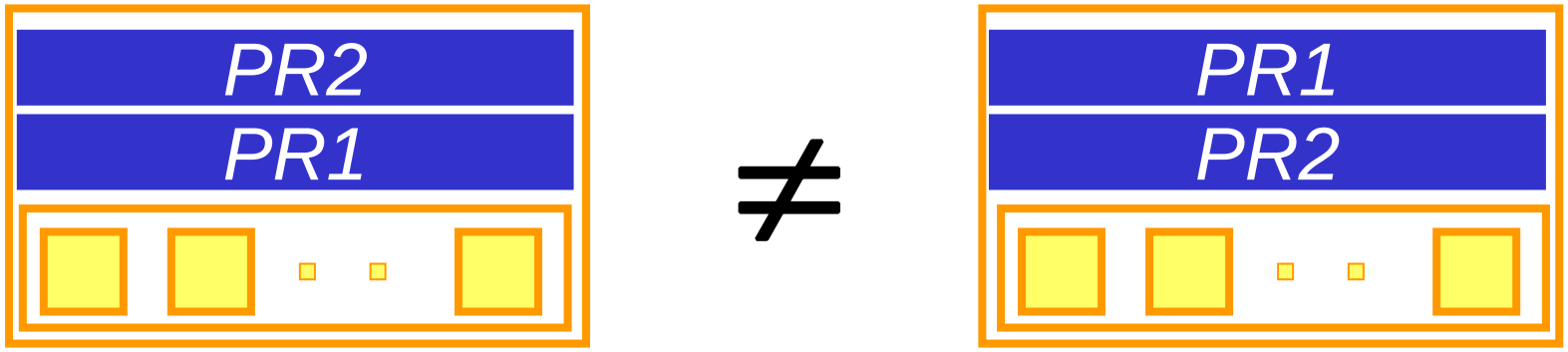
# Priorities: FIFO policy

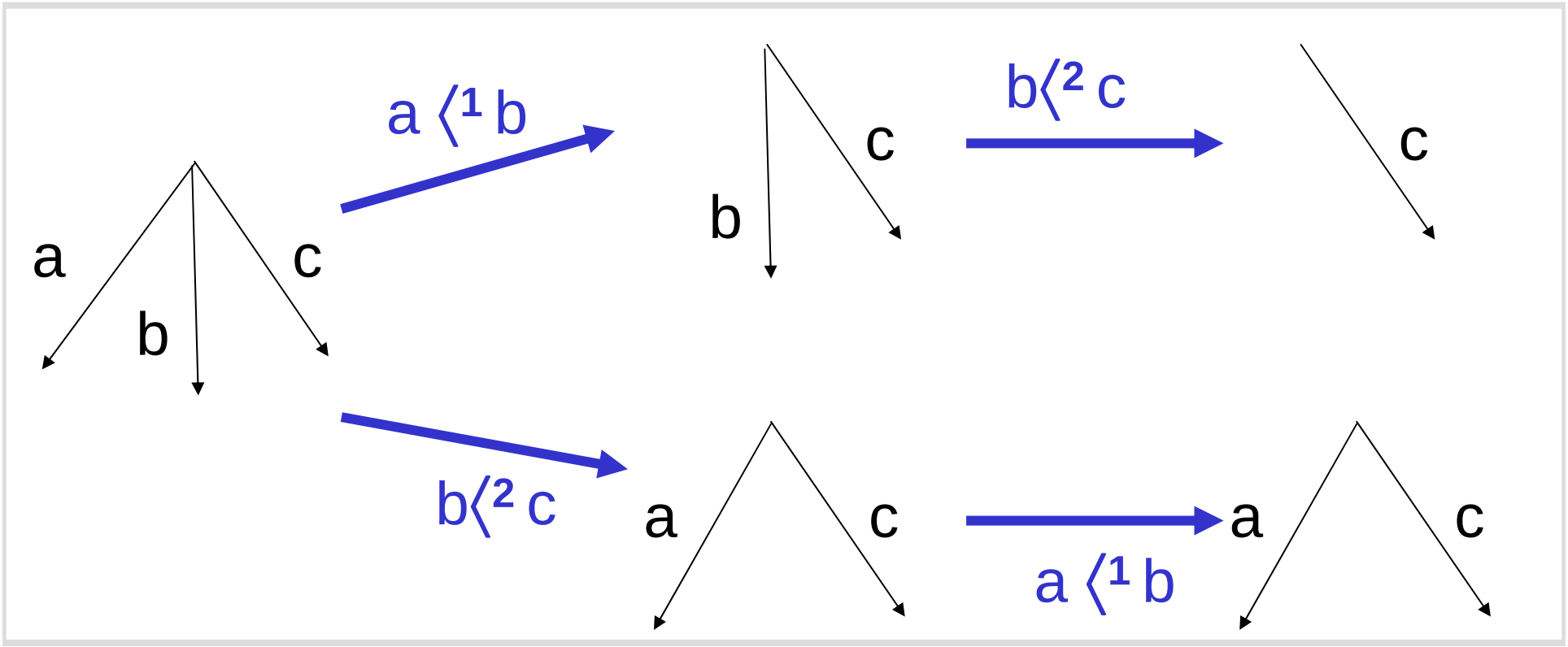


# Priorities: EDF policy



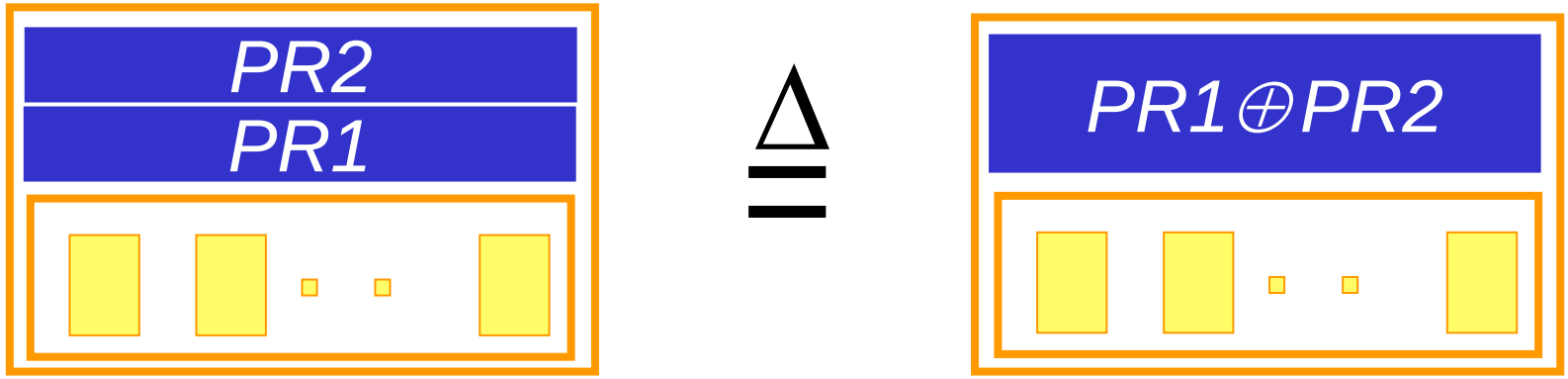
# Priorities: Composition



# Priority modeling– Composition (2)

We take:

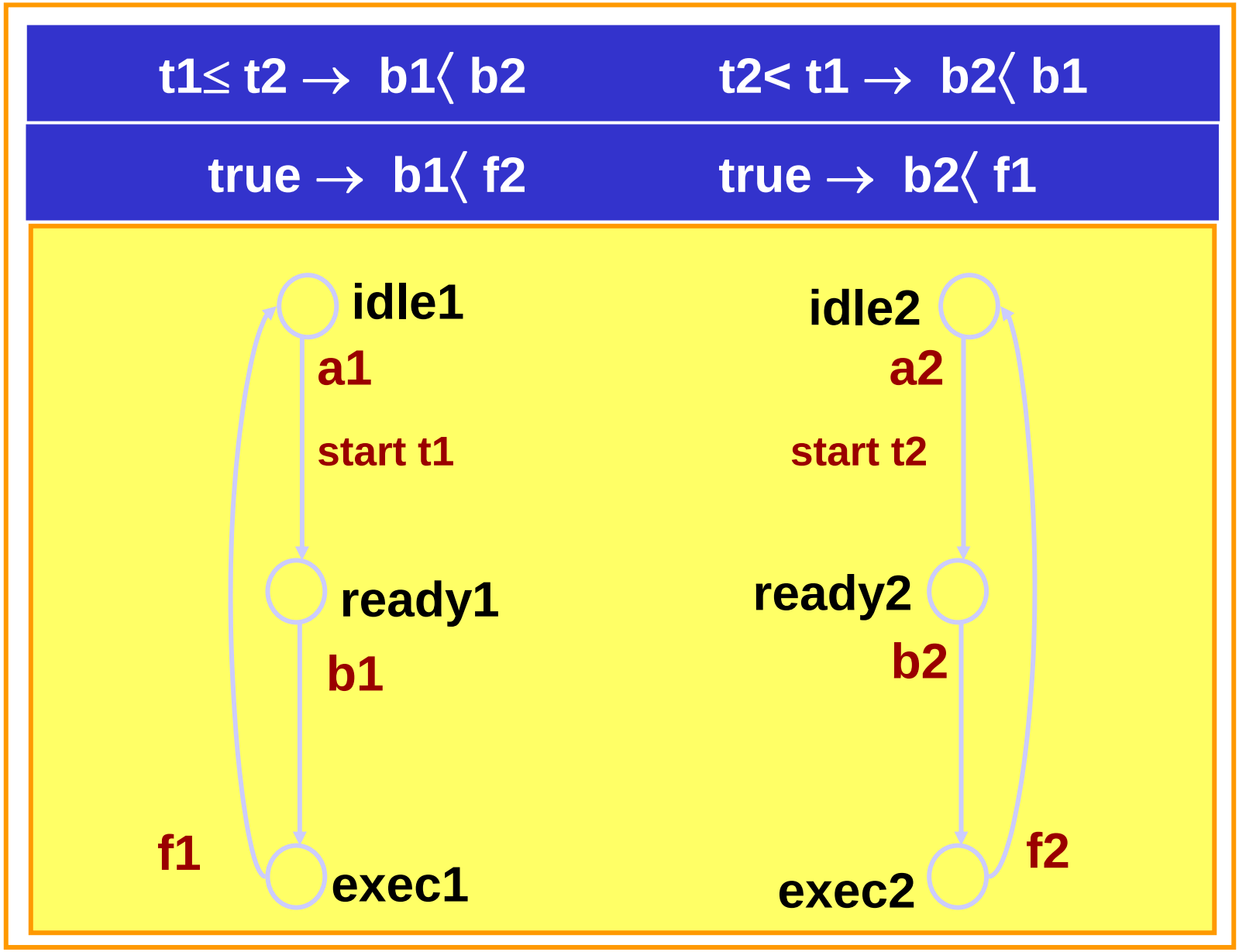



PR1⊕ PR2 is the least priority containing PR1∪PR2

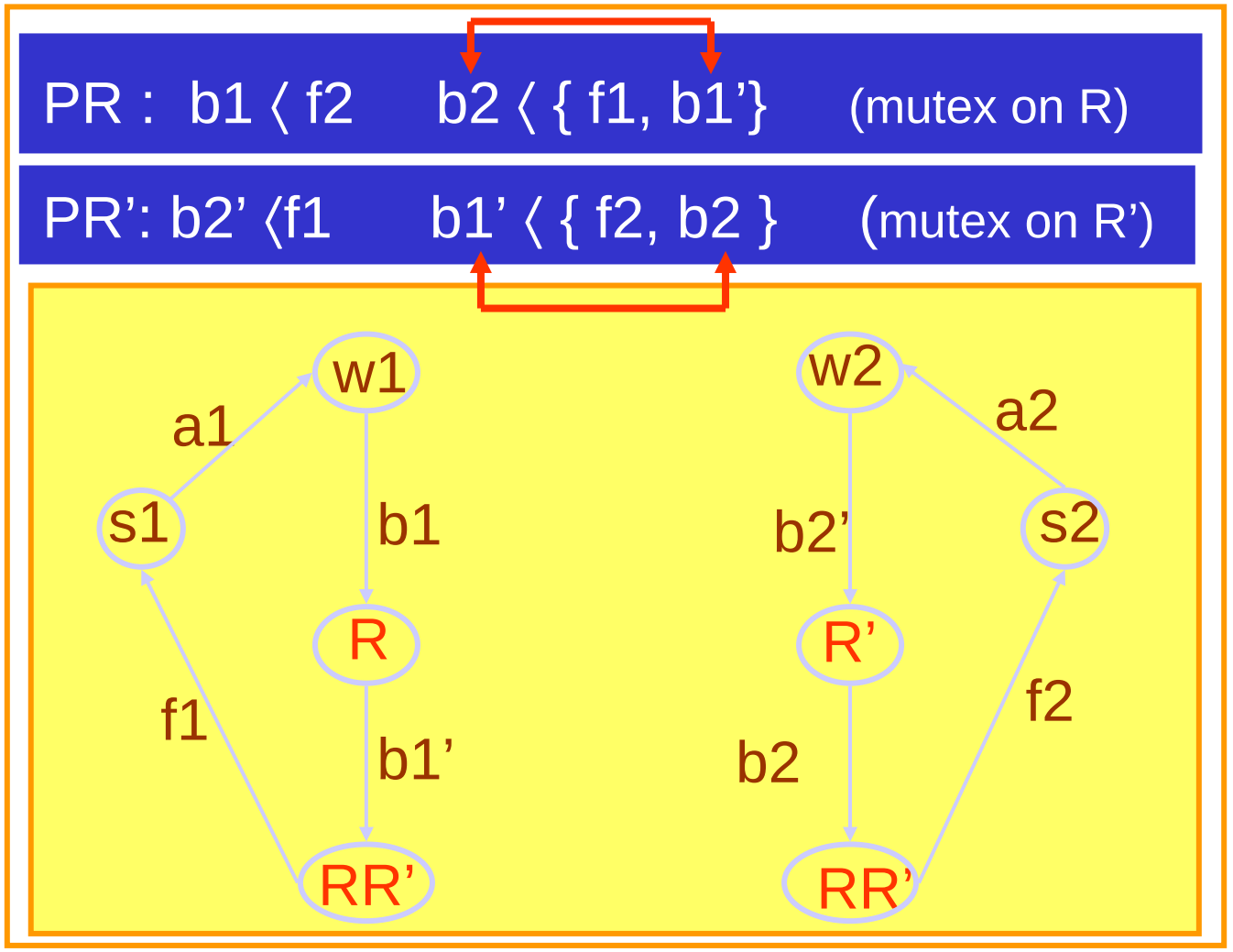
**Results :**

- The operation ⊕ is partial, associative and commutative
- PR1(PR2(B)) ≠PR2(PR1(B))
- PR1⊕ PR2(B) *refines* PR1∪PR2(B) *refines* PR1(PR2(B))
- Priorities preserve deadlock-freedom

# Priorities: Mutual Exclusion + FIFO policy



# Priorities: Mutual Exclusion - Example

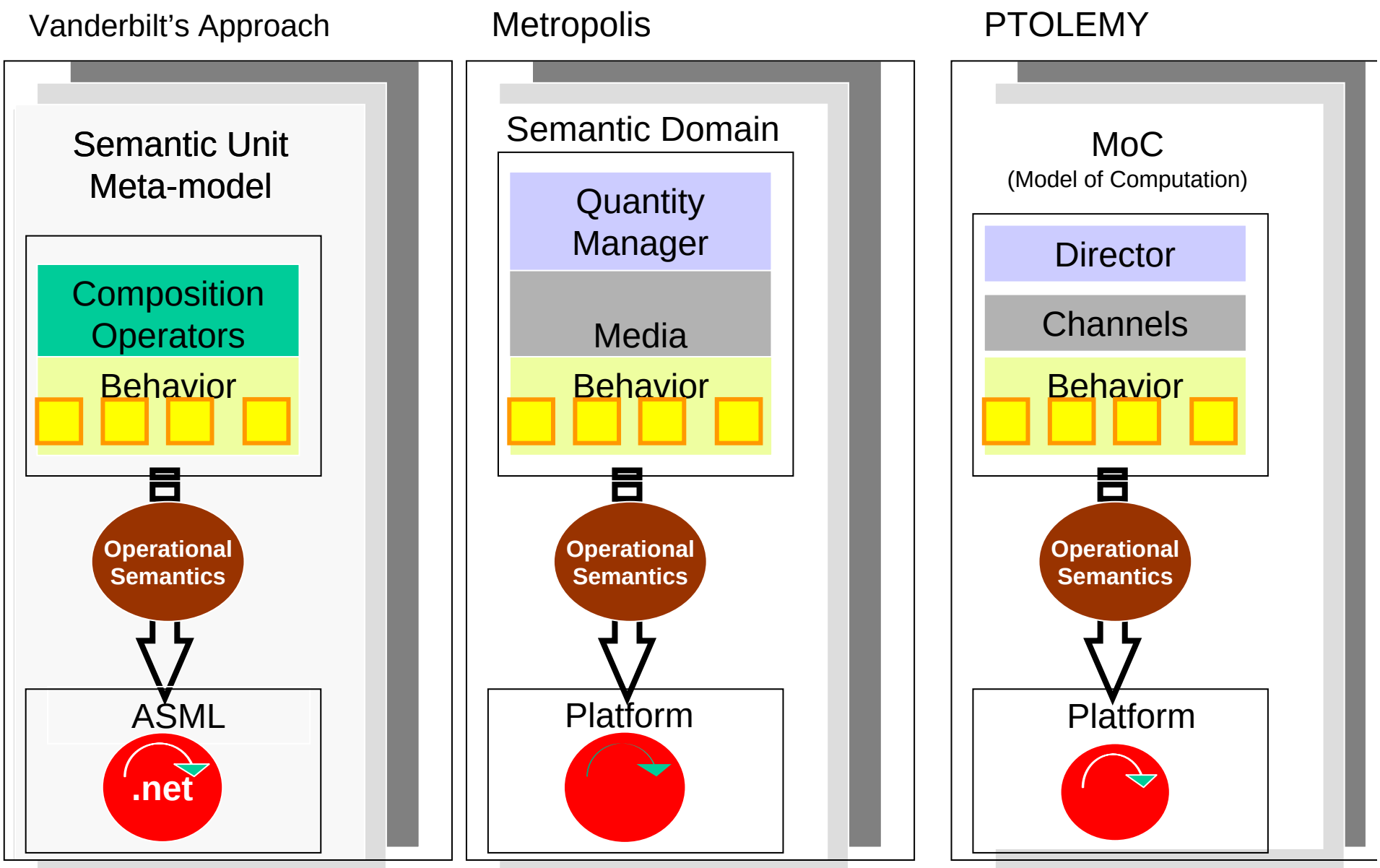



Risk of deadlock: PR⊕PR' is not defined

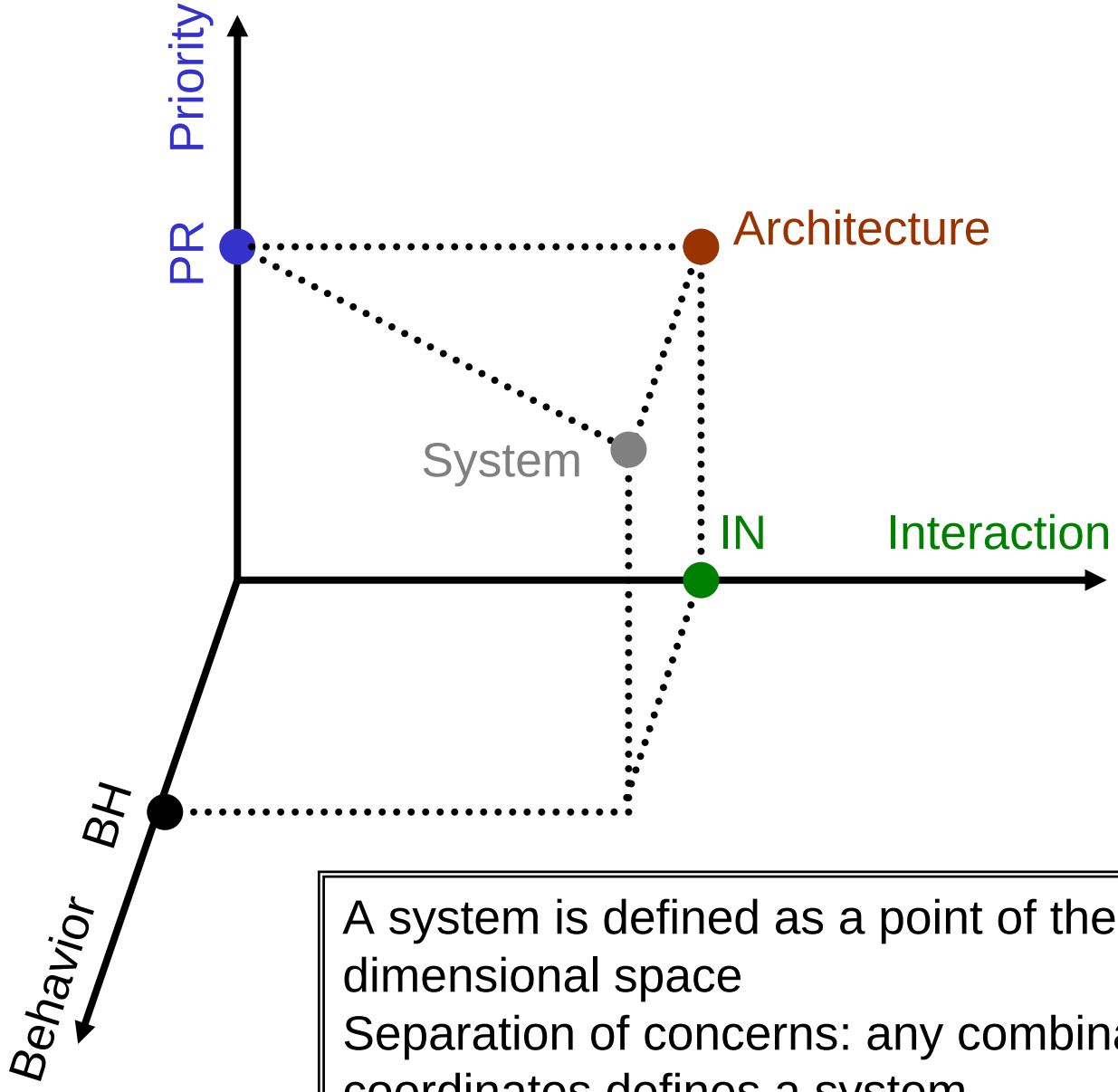
# Overview

- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# The BIP Framework: Model Construction Space - Related Approaches

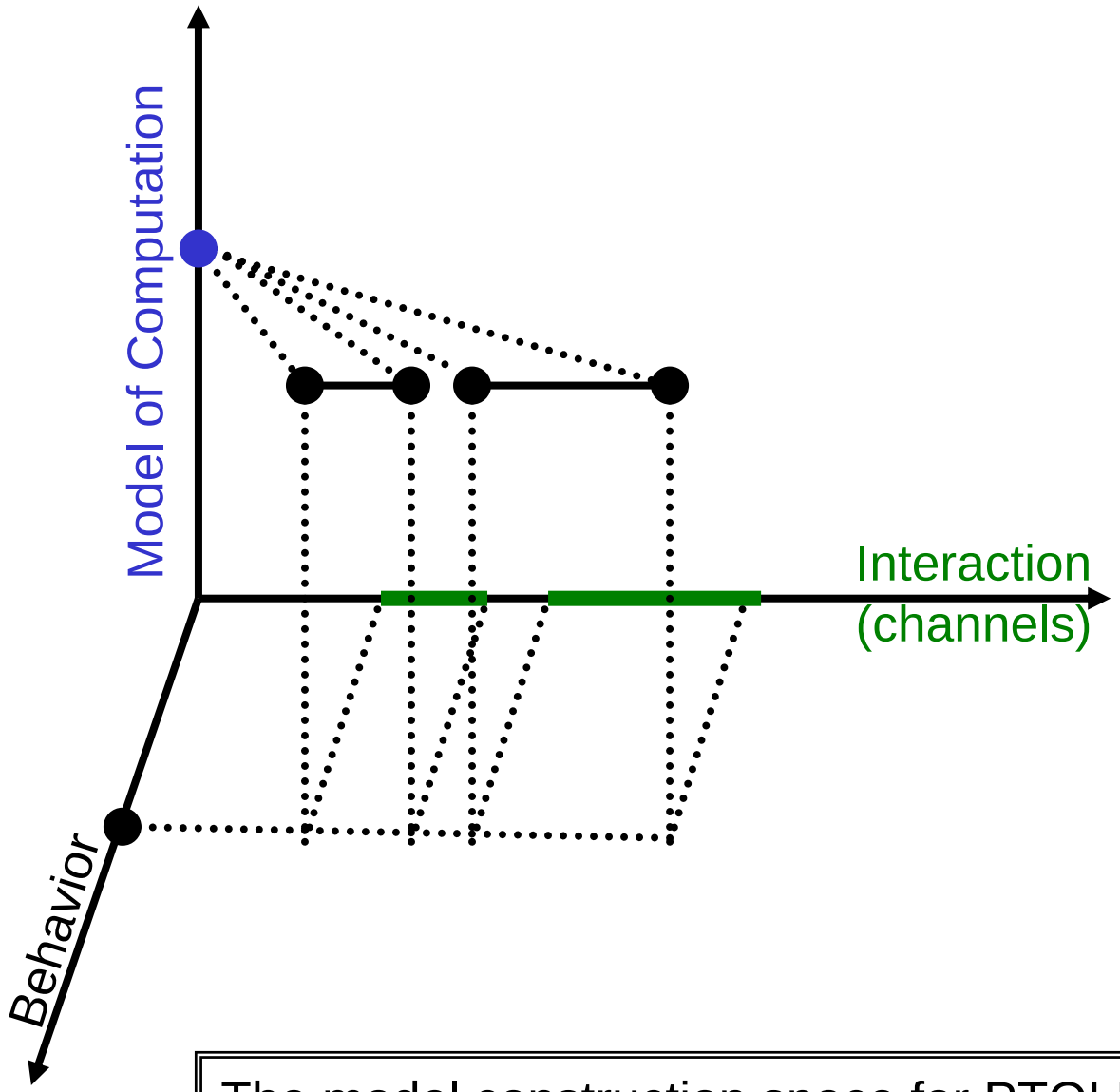


# The BIP Framework: Model Construction Space



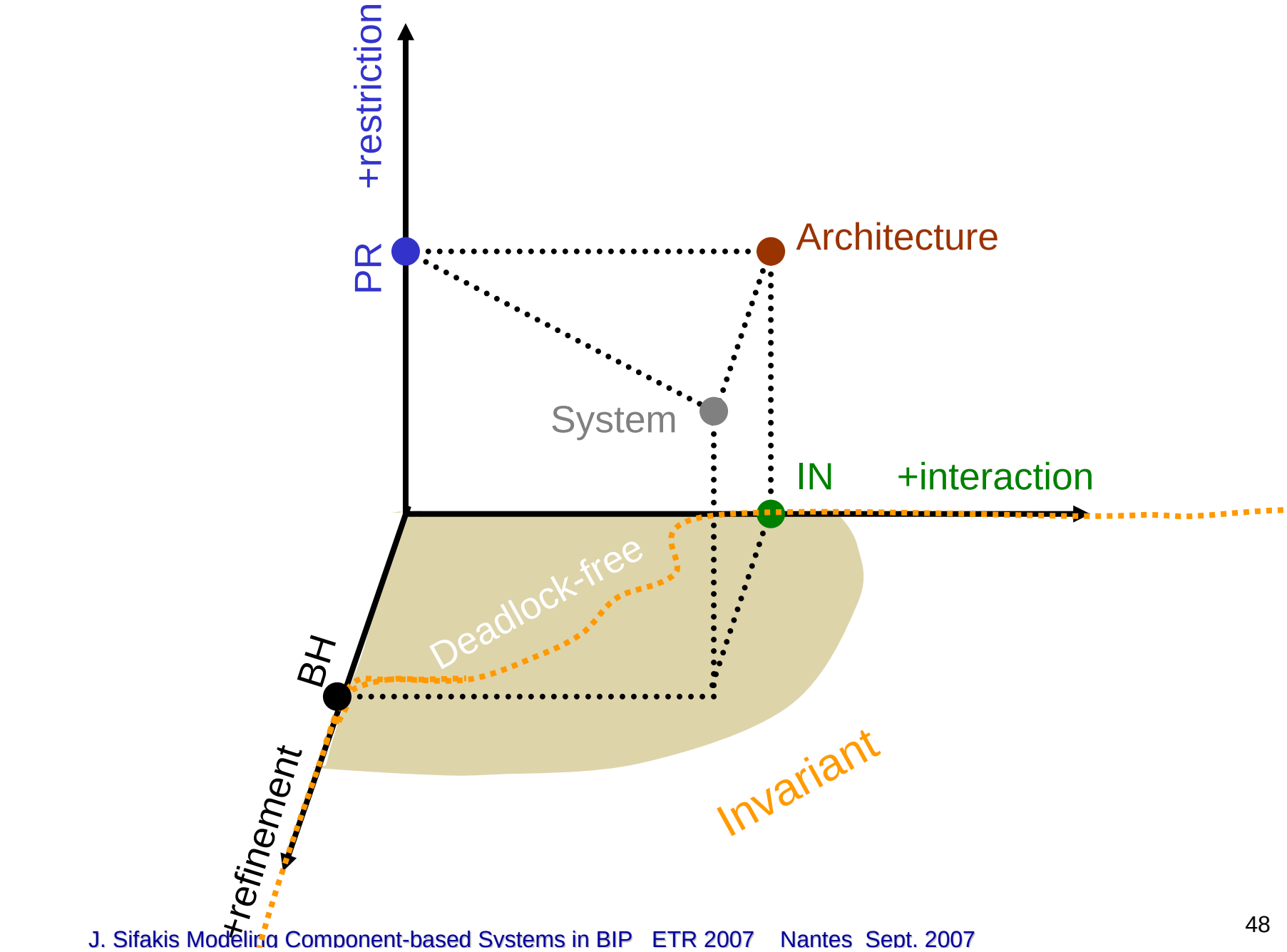

A system is defined as a point of the 3-dimensional space
Separation of concerns: any combination of coordinates defines a system

# The BIP Framework: Model Construction Space

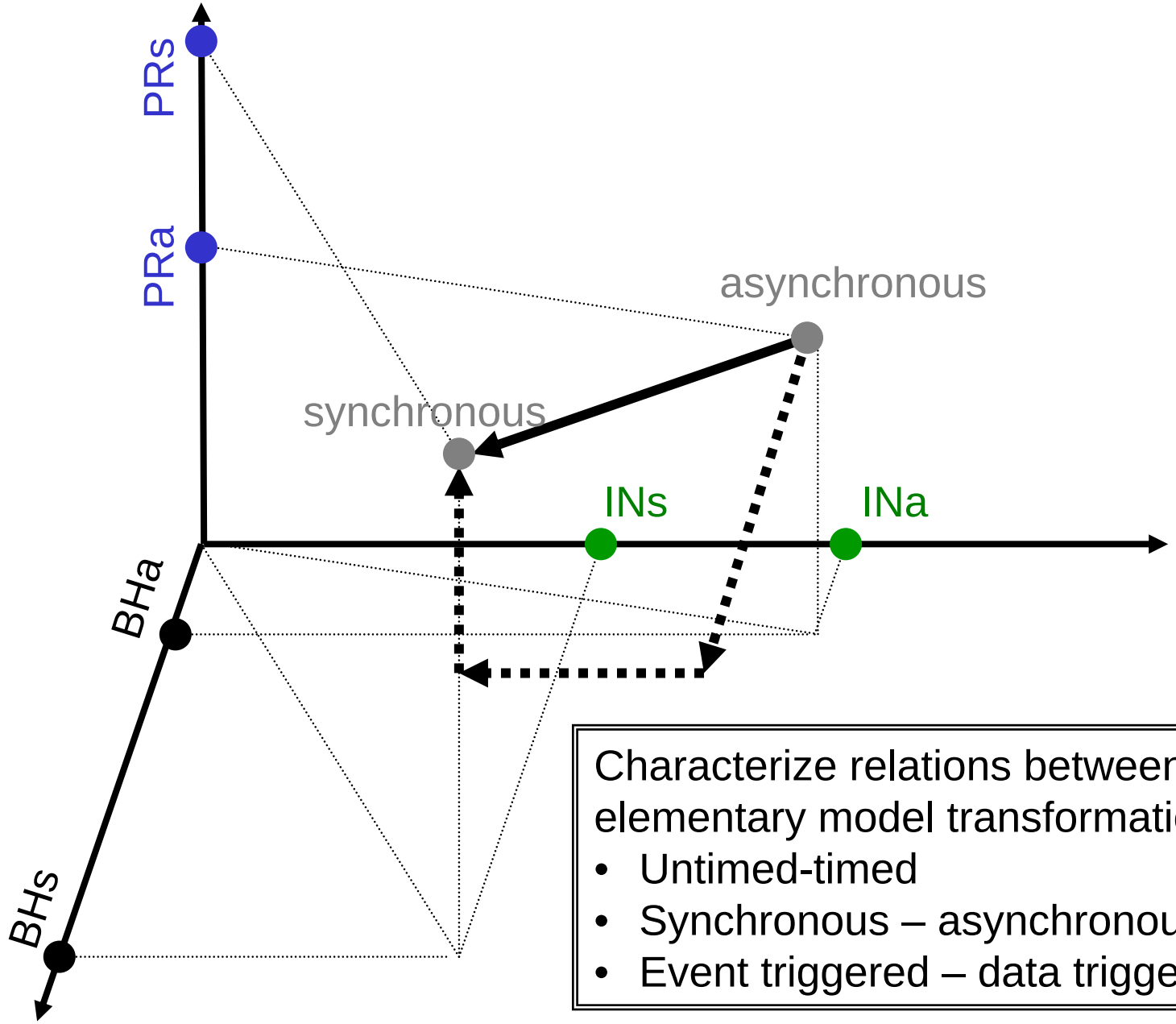



The model construction space for PTOLEMY

# The BIP Framework: Model Construction Space – Property Preservation



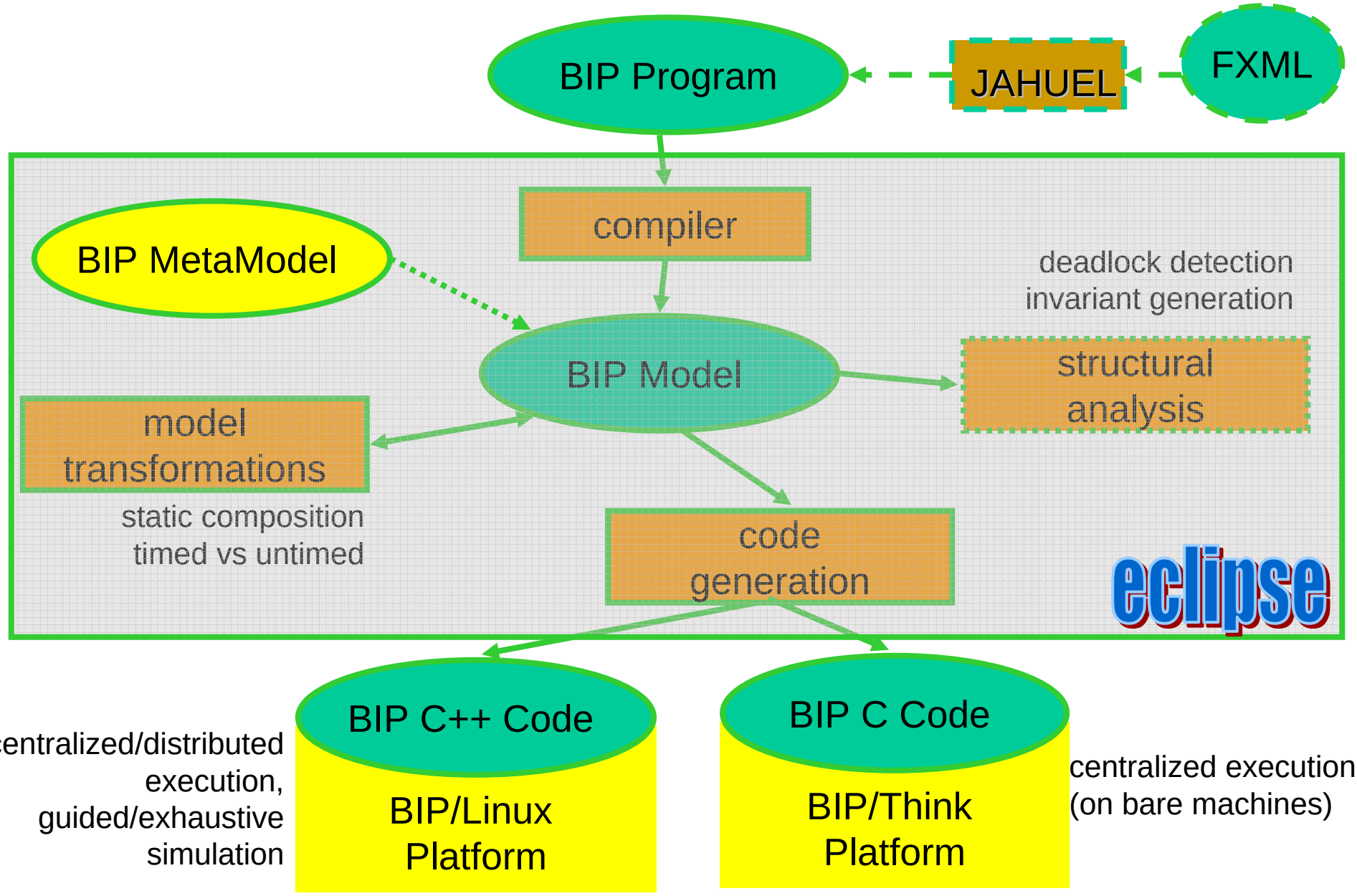
# The BIP Framework: Model Construction Space– Classes of Components



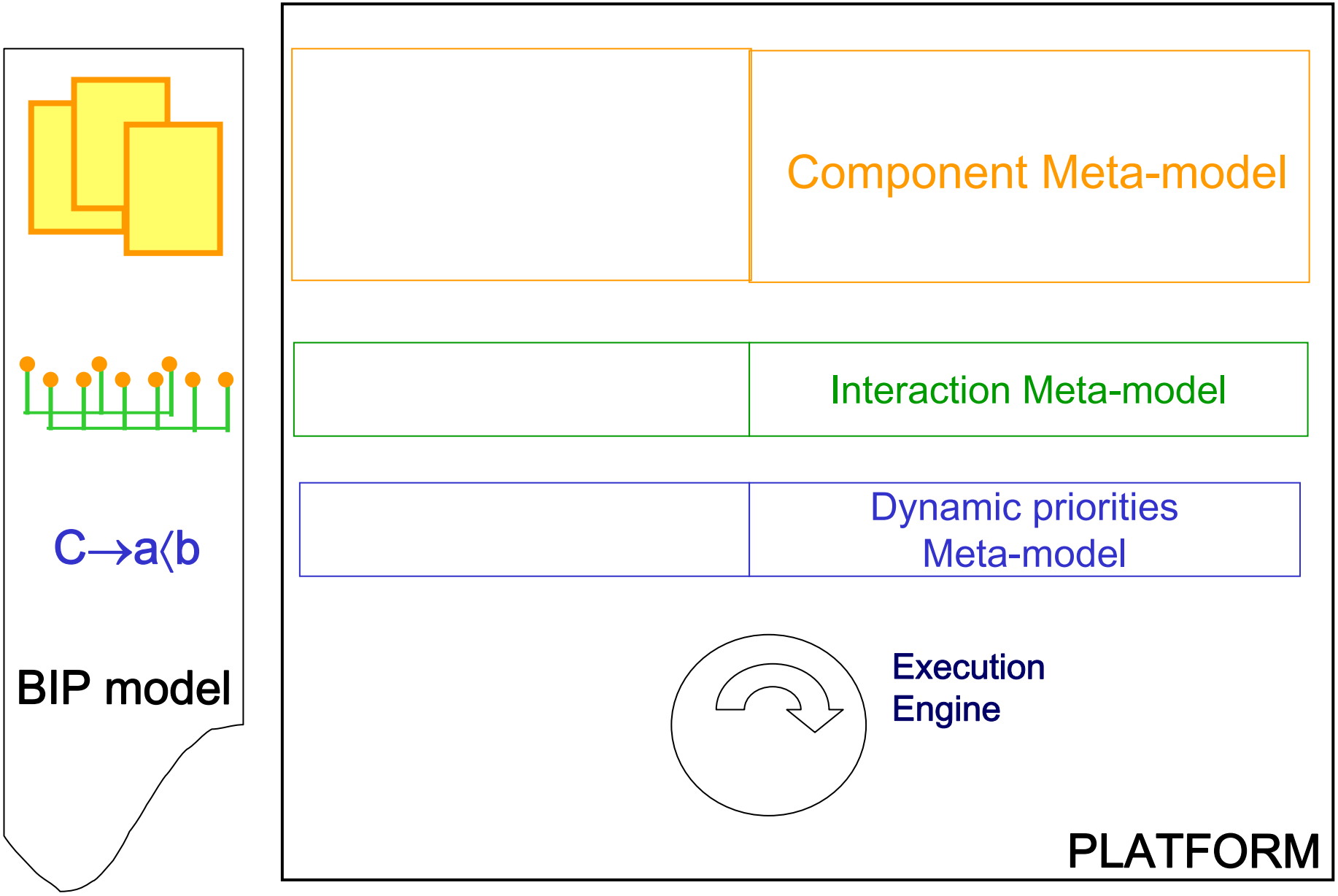

Characterize relations between classes by elementary model transformations:

- Untimed-timed
- Synchronous – asynchronous
- Event triggered – data triggered

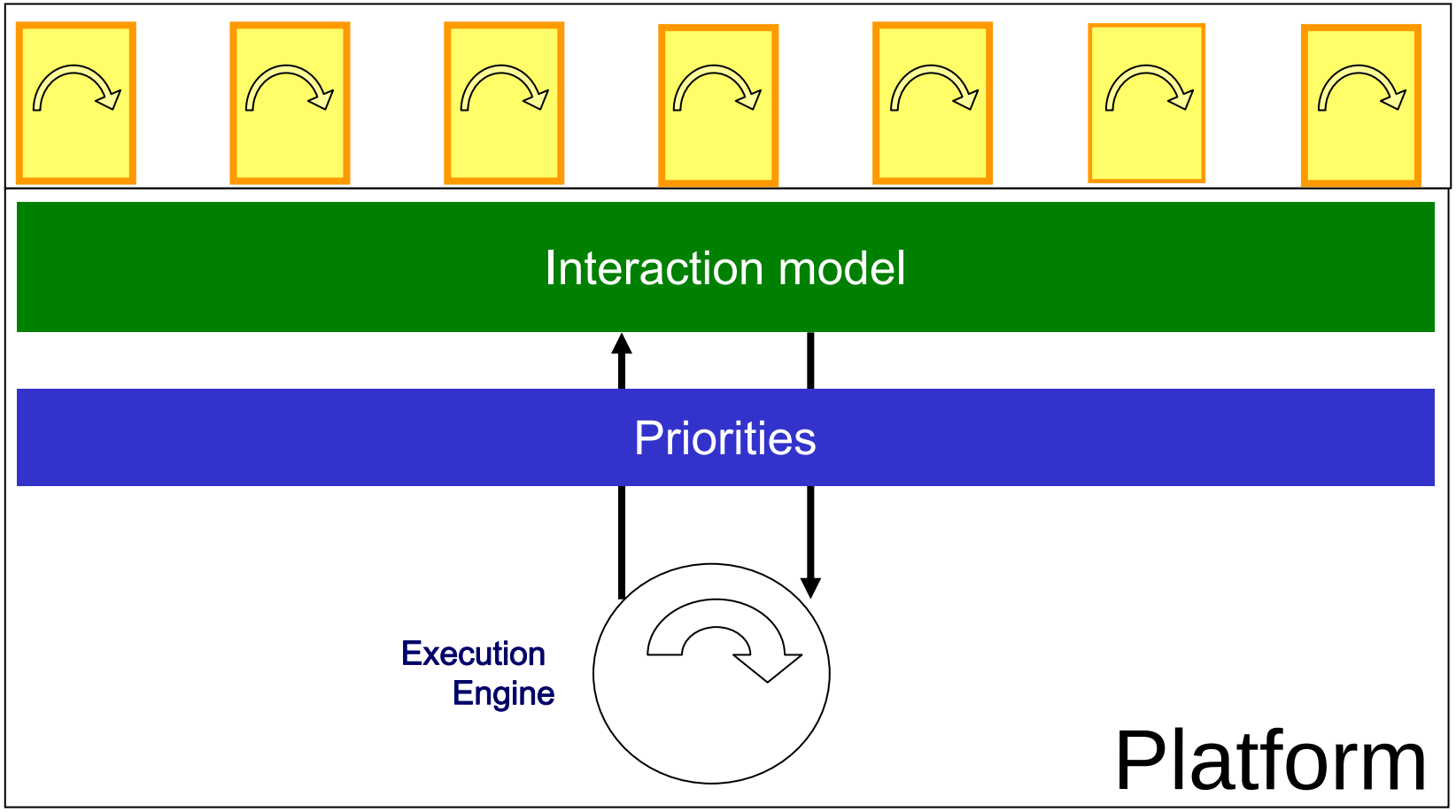
# The BIP Framework: Implementation



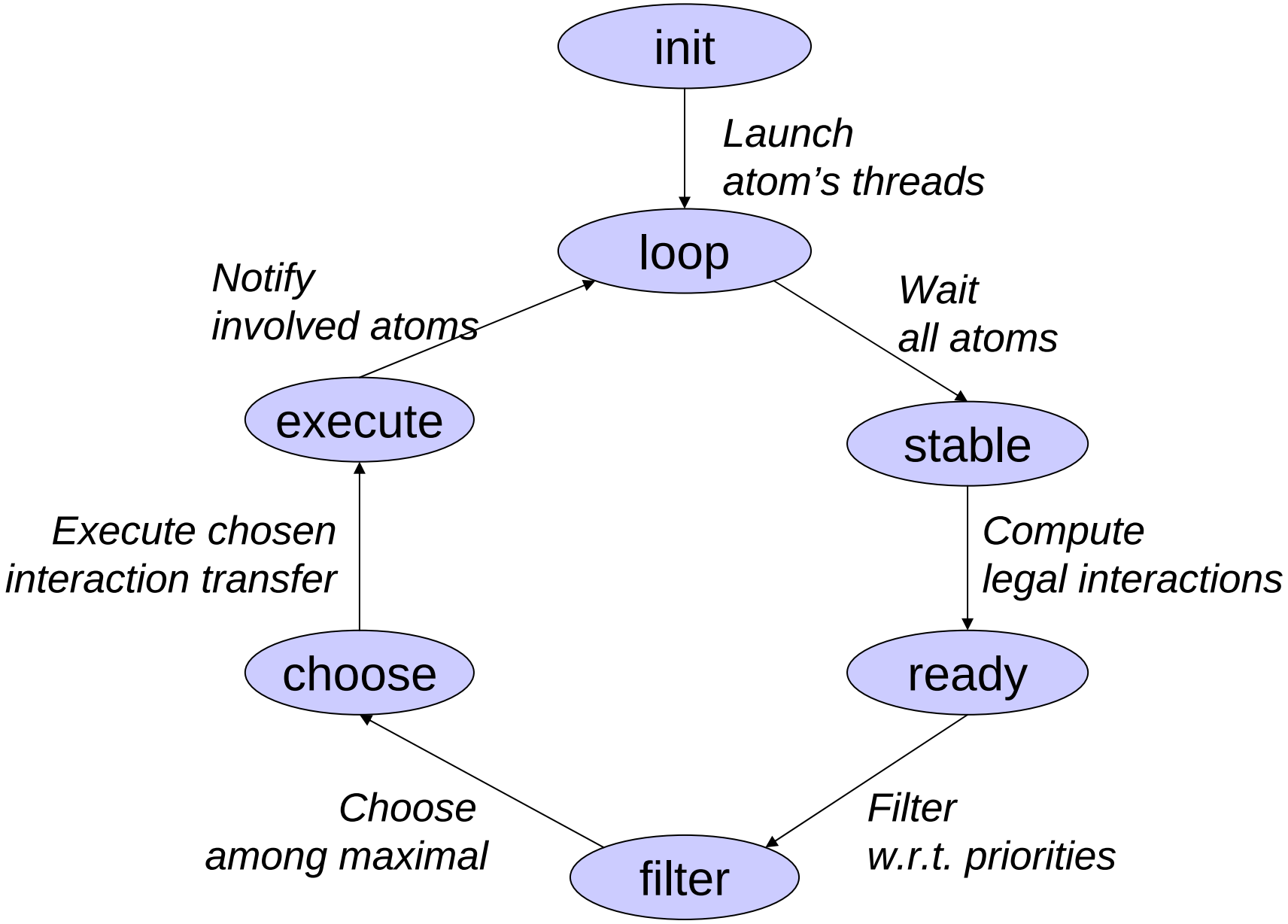
# The BIP Framework: Implementation - generation of C++ code



# The BIP Framework: Implementation - The Execution Platform



# The BIP Framework: Implementation – The BIP Execution Engine

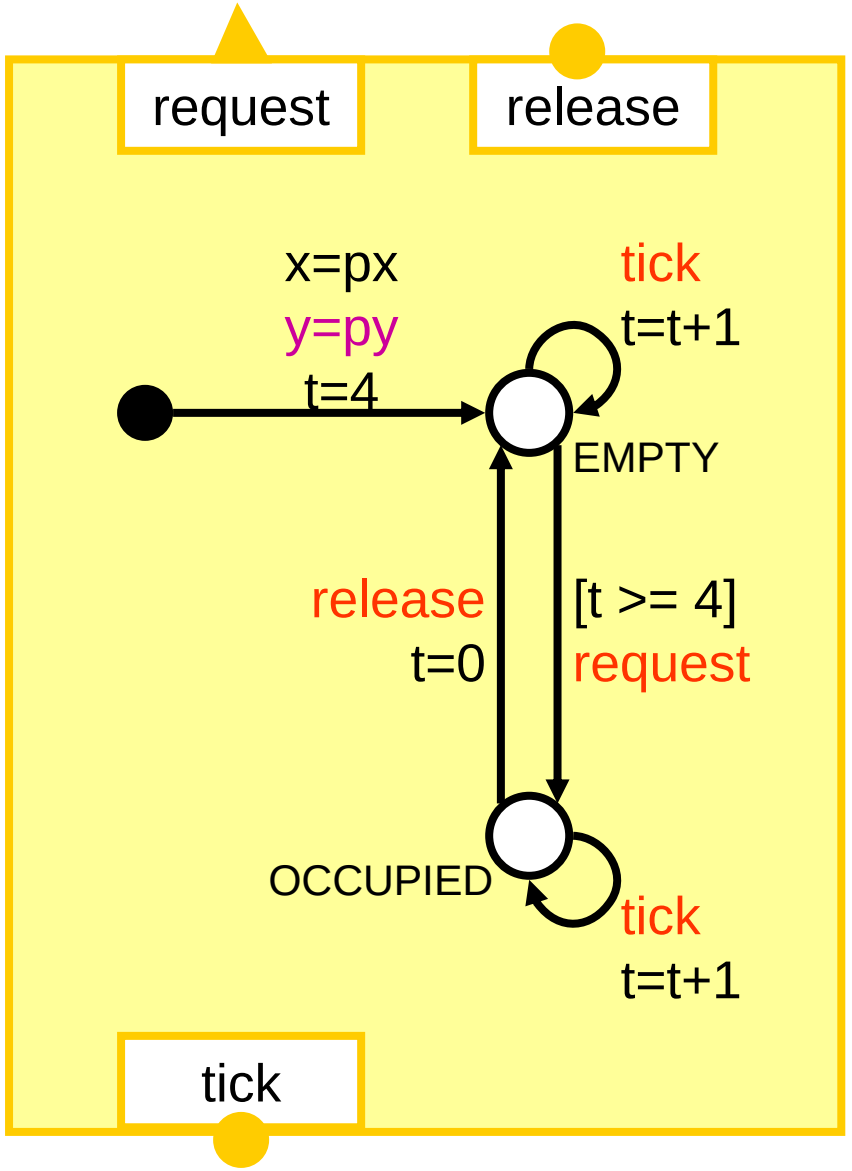


# The BIP Framework: The Language – Atomic Components

```
component C
port  complete: p1, … ;  incomplete: p2, …
data  {# int x, float y, bool z, …. #}
init {# z=false; #}
   behavior
         state s1
                  on p1 provided g1 do f1 to  s1'
                  ……….      ……
                  on pn provided gn do fn to  sn'

         state s2
                  on …..
           ….

         state sn
                  on ....
   end
end
```

# The BIP Framework: The Language – Atomic Components

```
component CallingUnit(int px, int py)

  complete port request // request port
  port release // release port
  port tick // time synchronization port

  data int x // x coordinate of the calling unit
  data int y // y coordinate of the calling unit
  data int t  // local clock

  behavior
    initial do x = px; {# y = py; #} t = 4 to EMPTY
    state EMPTY // empty state
      on request provided t >= 4 to OCCUPIED
      on tick do t := t + 1 to EMPTY
   state OCCUPIED // occupied state
      on release do t := 0 to EMPTY
      on tick to OCCUPIED
  end

end
```



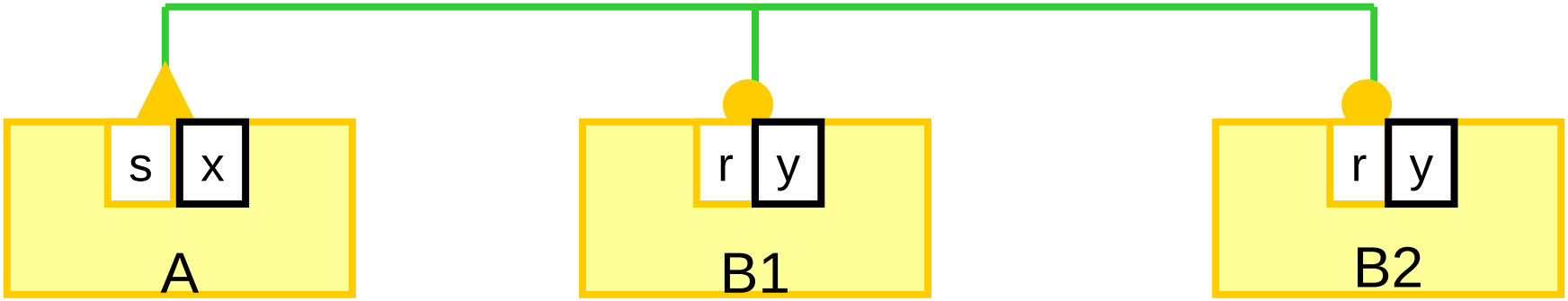
# The BIP Framework: The Language – Connectors and Priorities

**connector** BUS= {p, p', … , }
**complete**()
  **behavior**
    **on** $\alpha 1$ **provided** $g_{\alpha 1}$ **do** $f_{\alpha 1}$
    **on** $\alpha 2$ **provided** $g_{\alpha 2}$ **do** $f_{\alpha 2}$
  **end**

**priority** PR
    **if** C1 ($\alpha 1 < \alpha 2$), ($\alpha 3 < \alpha 4$) , …
    **if** C2 ($\alpha$ < …), ($\alpha$ <…) , …
    …
    **if** Cn ($\alpha$ <…), ($\alpha$ <…) , …

# The BIP Framework: The Language – Connectors and Priorities

```
connector Bus = A.s, B1.r, B2.r
behavior
  on A.s provided true do A.x = A.x + 1
  on A.s, B1.r provided A.x < B1.y do B1.y = A.x
  on A.s, B2.r provided A.x > B2.y do B2.y = A.x
  on A.s, B1.r, B2.r provided true do A.x = (B1.y + B2.y) / 2
end
```




```
priority mutex
  if  (C1.t >   C2.t)  C1.begin < C2.begin
  if  (C1.t <= C2.t)  C2.begin < C1.begin
end
```

# The BIP Framework: The Language – Compound Components

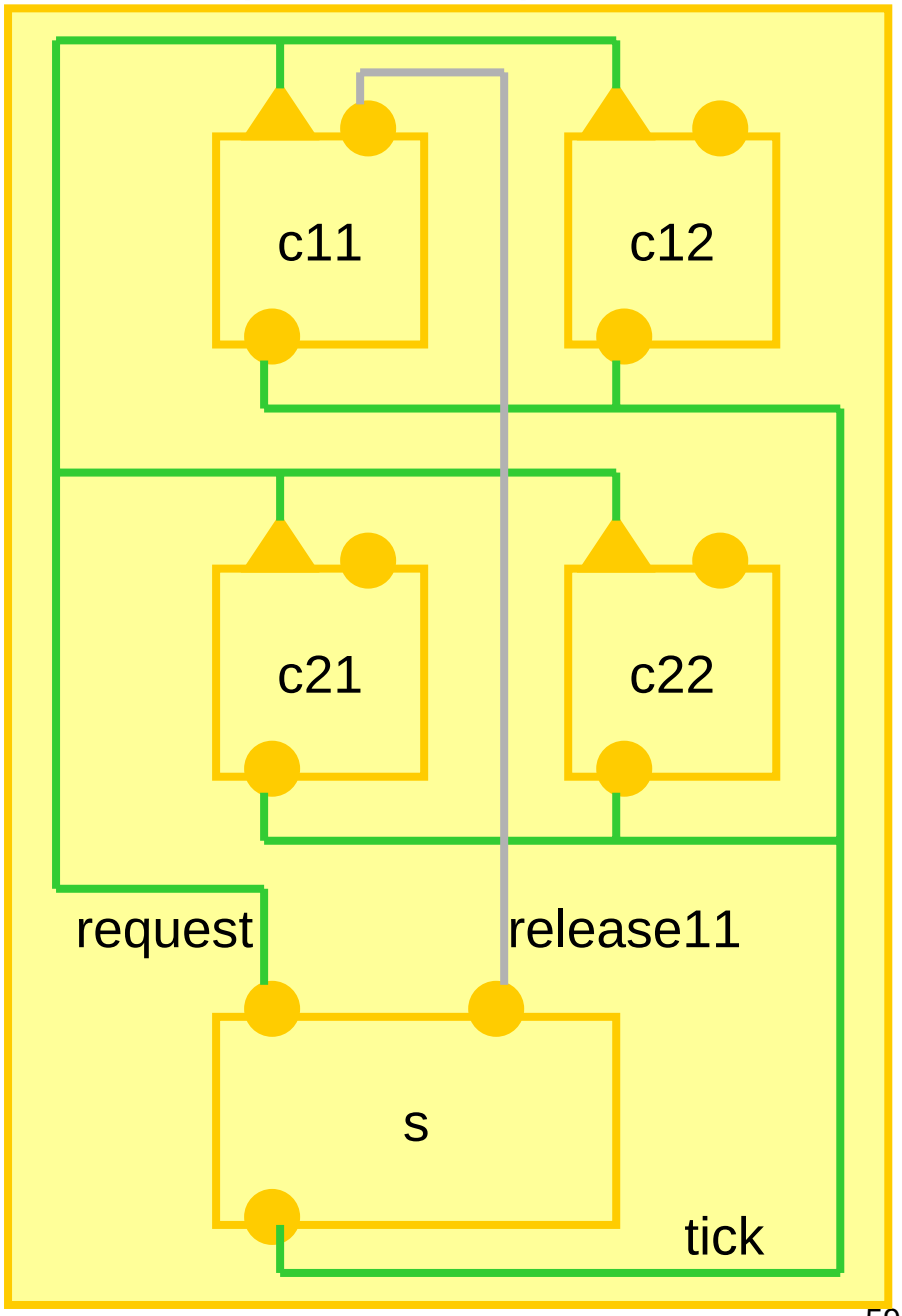
**component name**
**contains c_name1 i_name1(par_list)**
**……**
**contains c_namen i_namen(par_list)**

**connector** name1
……
**connector** namem

**priority** name1
……
**priority** namek
end

# The BIP Framework: The Language – Compound Components

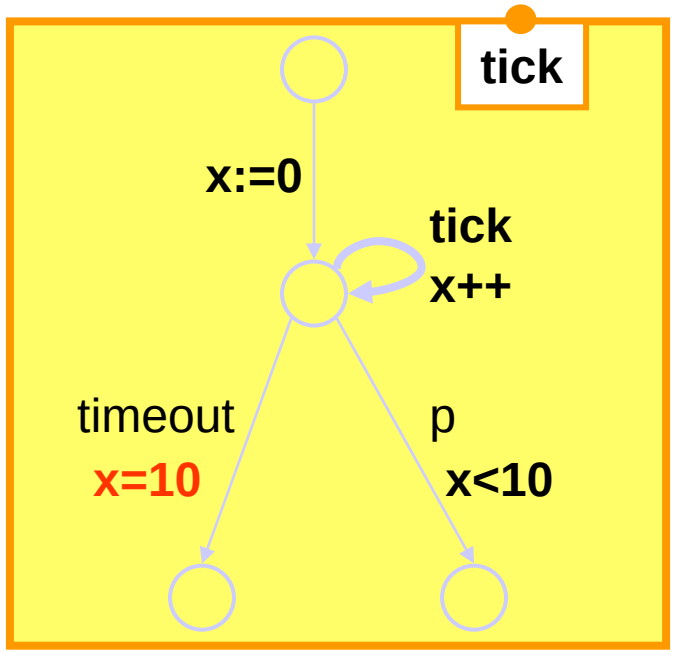
```
component Utopar

    contains CallingUnit c11(1,1)
    contains CallingUnit c12(1,2)
    contains CallingUnit c21(2,1)
    contains CallingUnit c22(2,2)

    contains CentralStation s

    connector request = c11.request,
            c12.request, c21.request,
            c22.request, s.request
            ...
    connector release11 = c11.release,
            s.release
            ...
    connector tick = c11.tick, c12.tick,
            c21.tick, c22.tick, s.tick
            ...
end
```

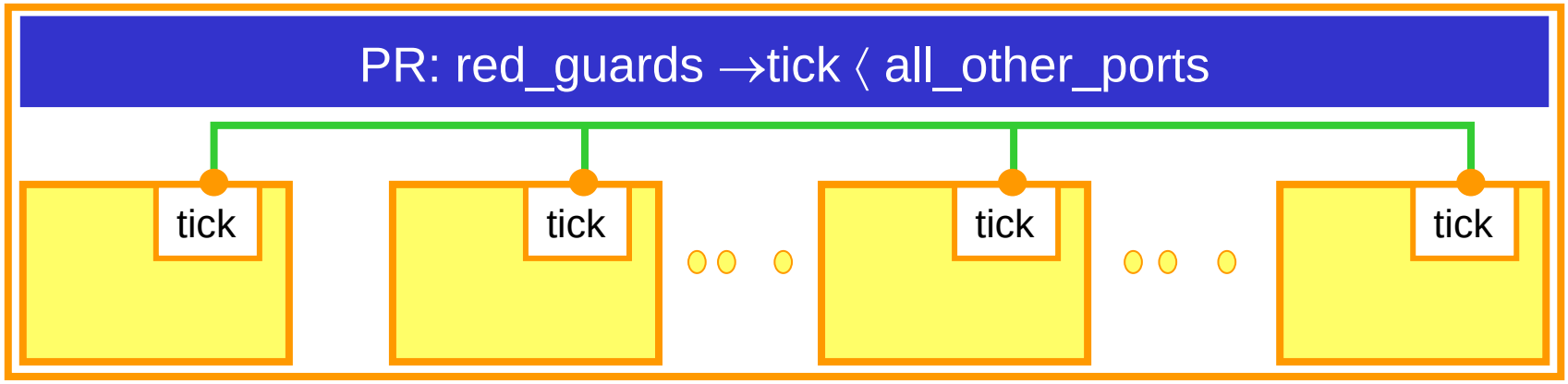


# Overview

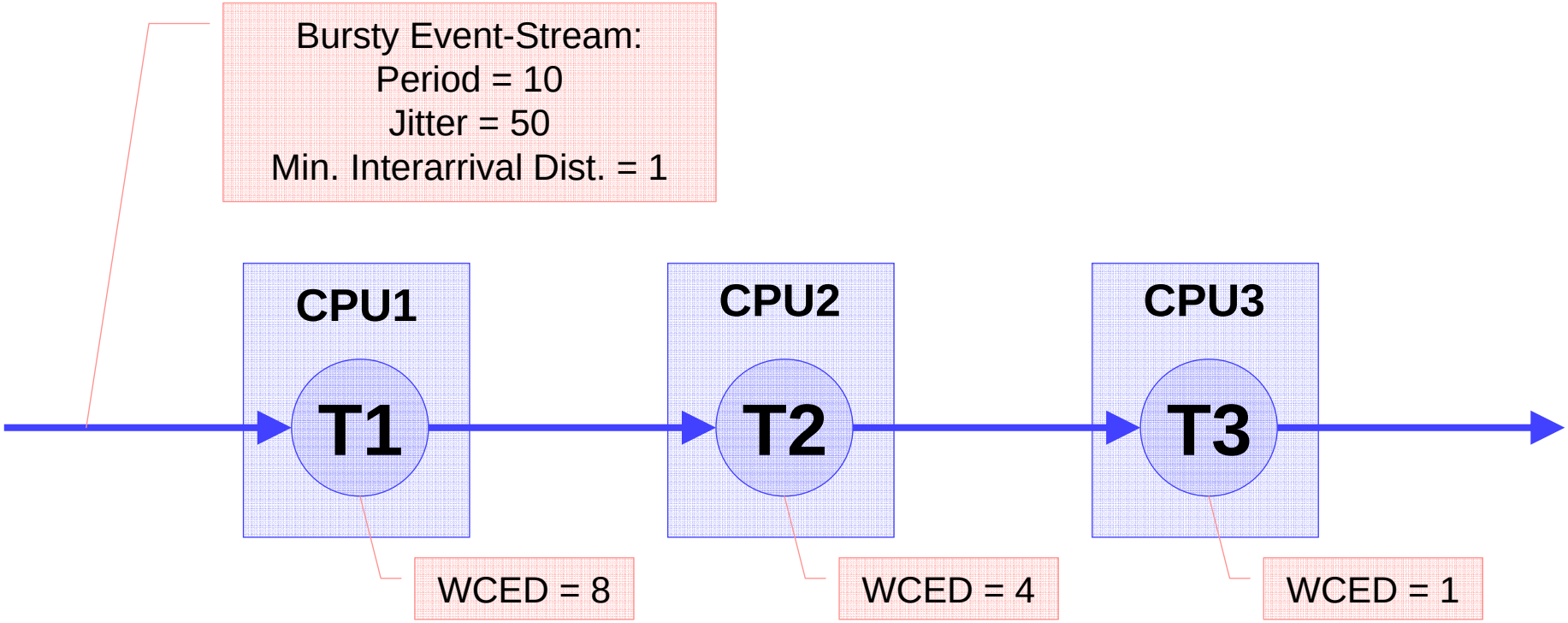
- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# Timed Components – Timed vs. Untimed

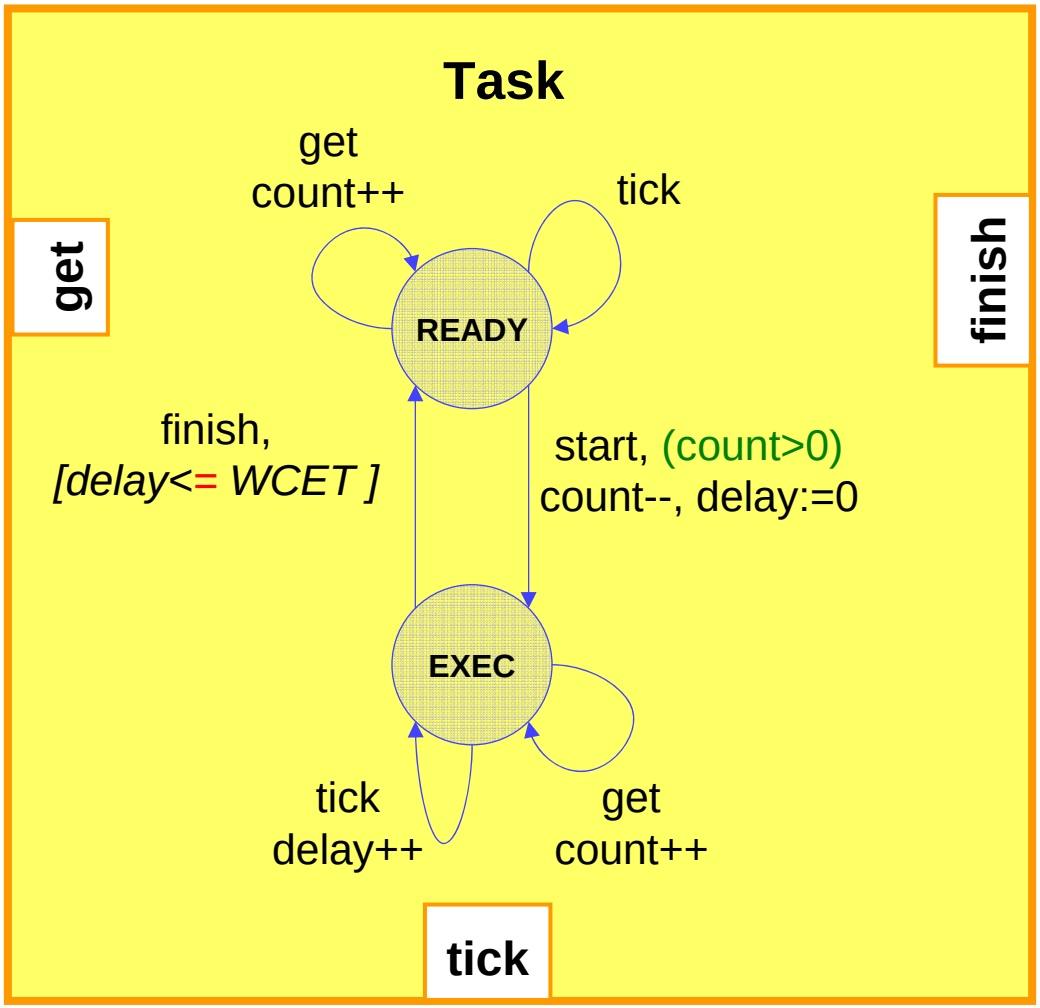


# Timed components - Problem 1



# Timed components: Problem 1- Task

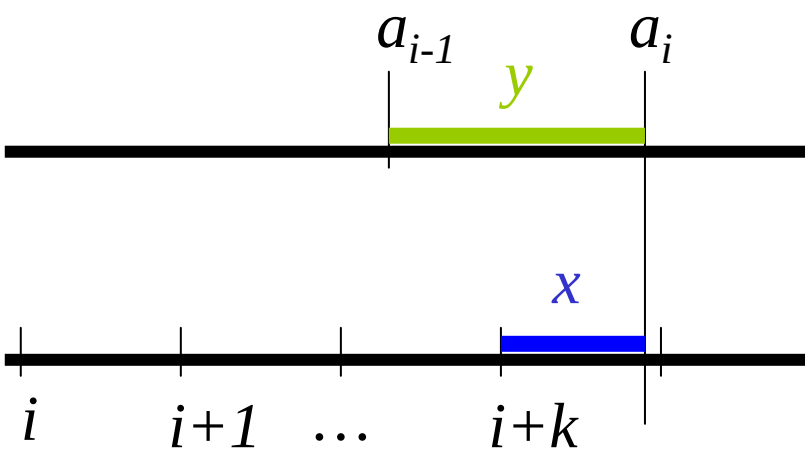


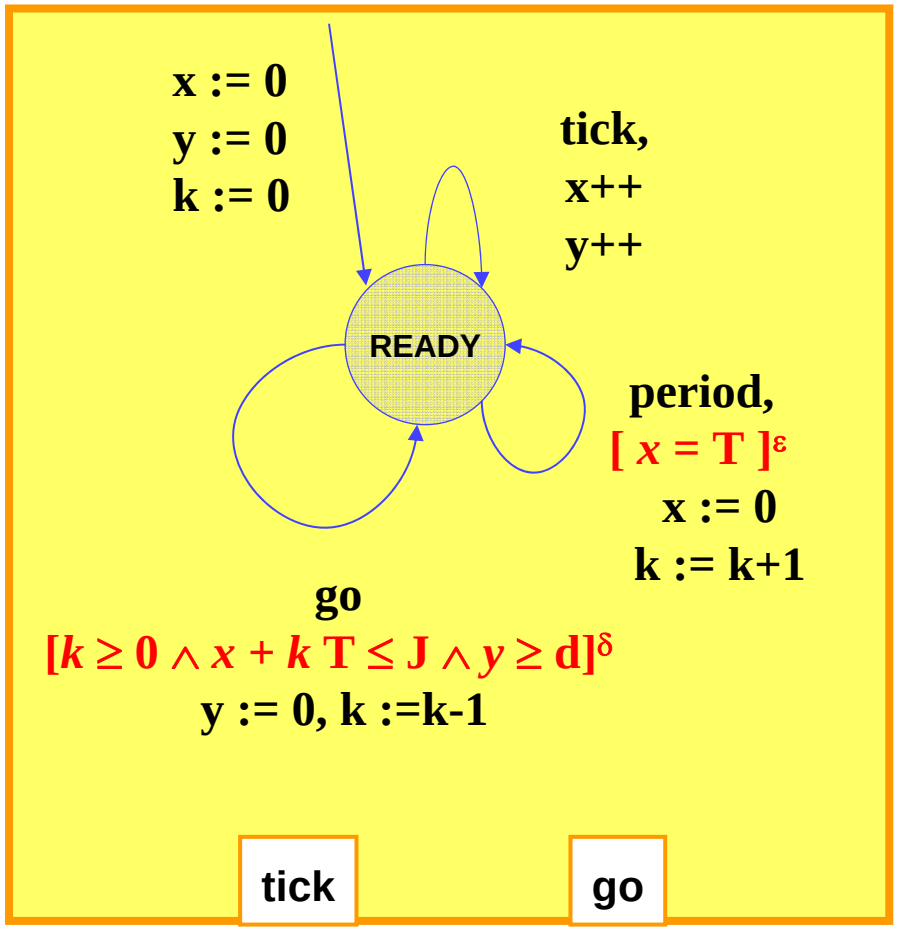
# Timed components: Problem 1 - BIP code snippet for Task

```
component Task (int wcet)
  port get, start, tick, finish
  data {# int count, delay; #}
  …
  init {# count = 0;
          WCET = wcet;
           …
       #}
  behavior
    state READY
      on get do {# count++; #} to READY
      on start provided {# count > 0 #} do {# count--; delay = 0; #} to EXEC
      on tick to READY
    state EXEC
      on get do {# count++; #} to EXEC
      on finish when ({# delay <= WCET #}, delayable) to READY
      on tick do {# delay++; #} to EXEC
  end
  …
end
```

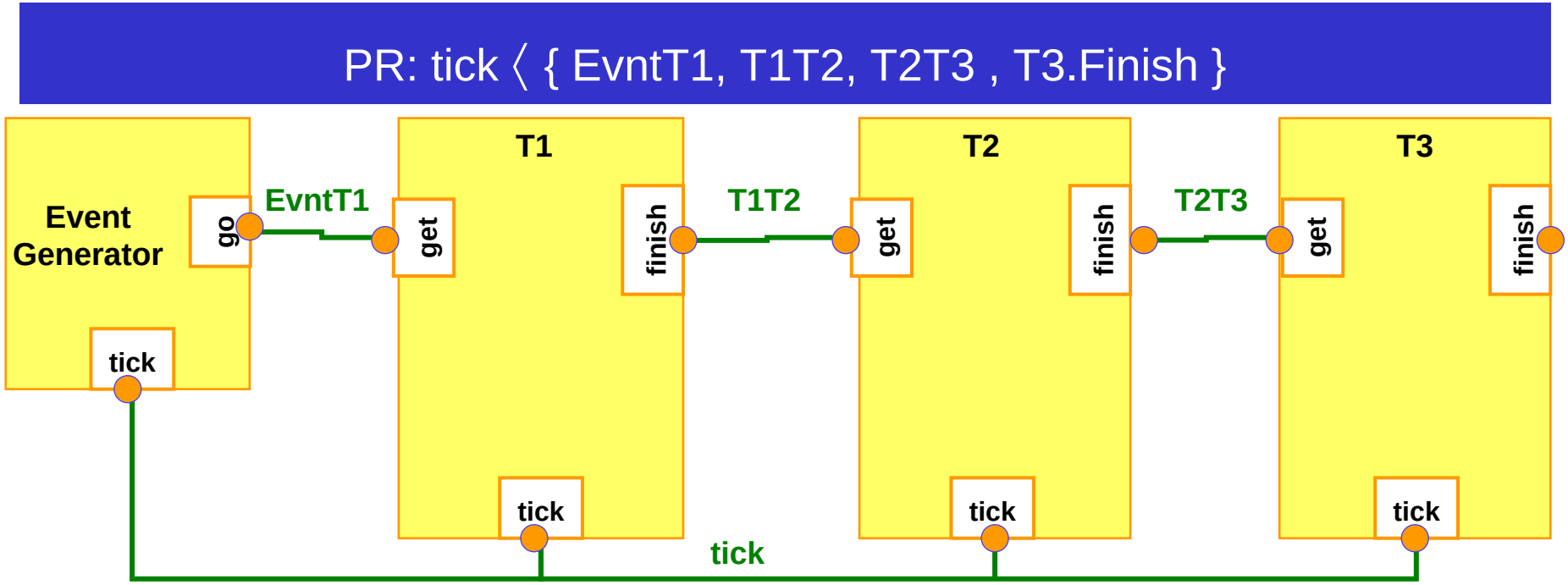
# Timed components: Problem 1- Bursty Event Stream Generator

Bursty Event-Stream for
Period = T
Jitter = J
Min. Interarrival Dist. = d



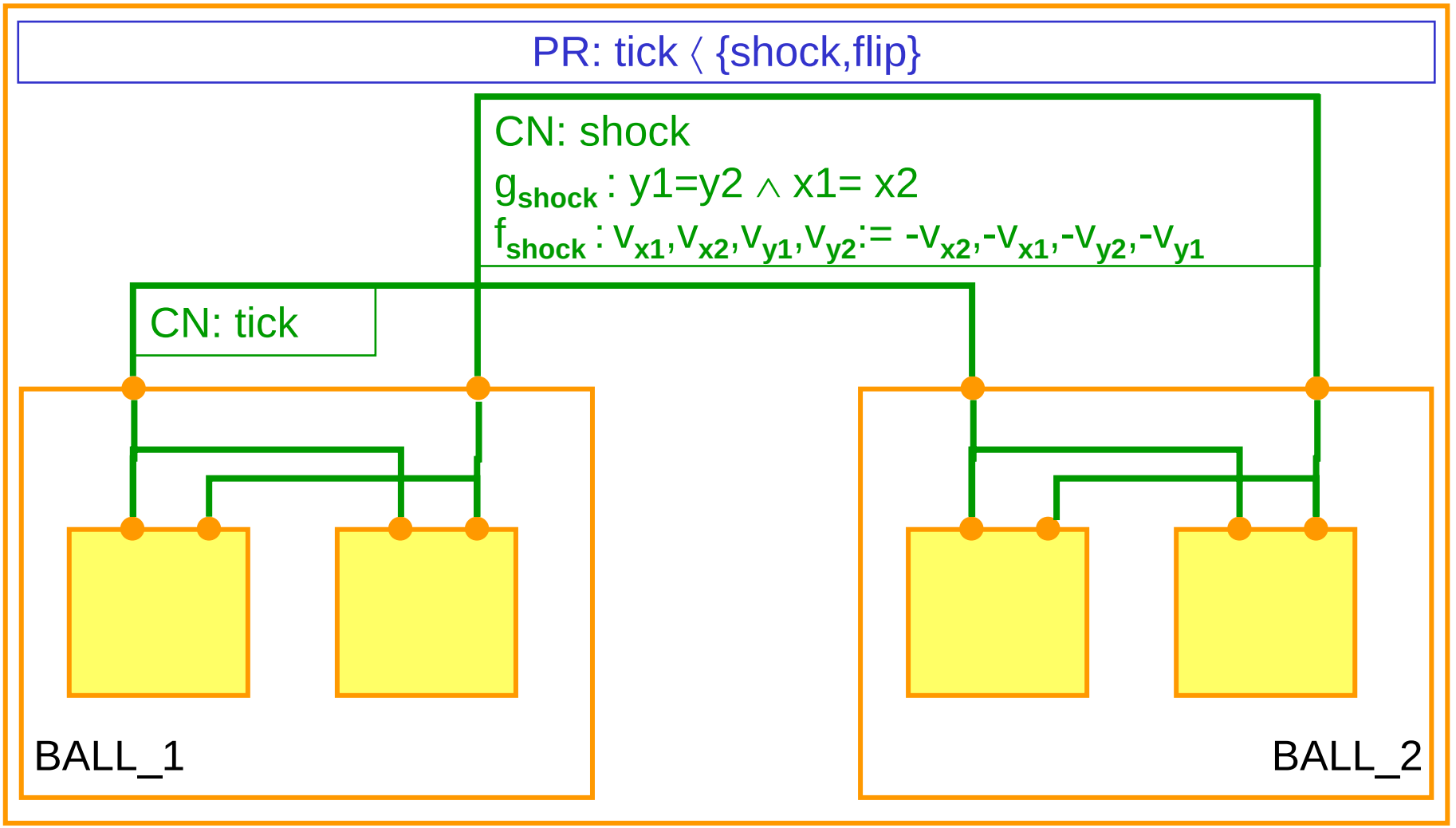
# Timed components: Problem 1 – Architecture



# Timed components: Problem 1- BIP code snippet for Architecture

```
component System
  contains Launcher eventGenerator(10, 5, 1)
  contains Task T1(8), T2(4), T3(1)

  connector Tick = eventGenerator.tick, T1.tick, T2.tick, T3.tick
    behavior
    end
 connector EvntT1 = eventGenerator.go, T1.get
    behavior
    end
  …
priority  // start < get ( no event losses )
    getStart1 T1.Start : T1.start < EvntT1 : T1.get
    …
priority // finish < get ( no event losses )
    getFin1 T1T2 : T1.finish < EvntT1 : T1.get
    …
priority // tick < get_i ( => tick < finishi_i-1 )
   getTick2 if (T1.delay == T1.WCET)  Tick : T2.tick < T1T2 : T2.get
   …
```

# Timed components: Billiards - the Model



# MPEG4 Video Encoder

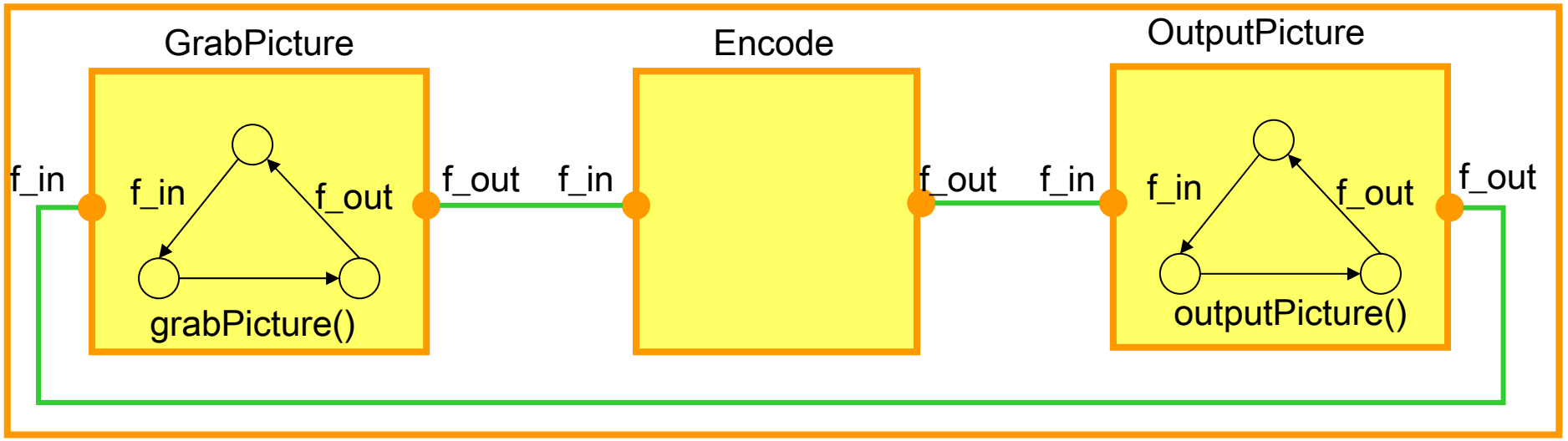
Transform a monolithic program into a componentized one

++ reconfigurability, schedulability

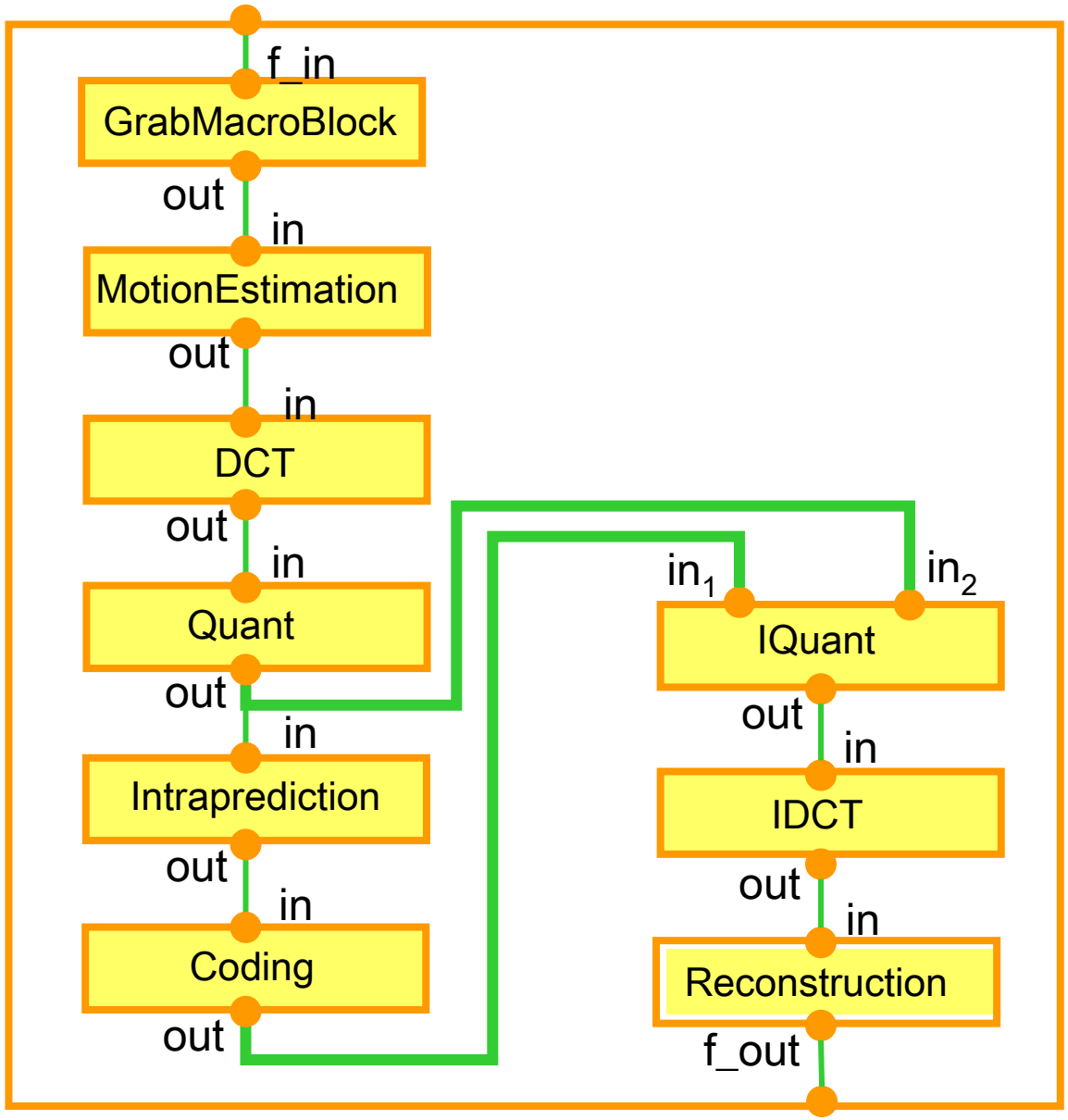
– – overheads (memory, execution time)

Video encoder characteristics:

- 12000 lines of C code
- Encodes one frame at a time:
  - grabPicture() : gets a frame
  - outputPicture() : produces an encoded frame



# MPEG4 Video Encoder: Architecture

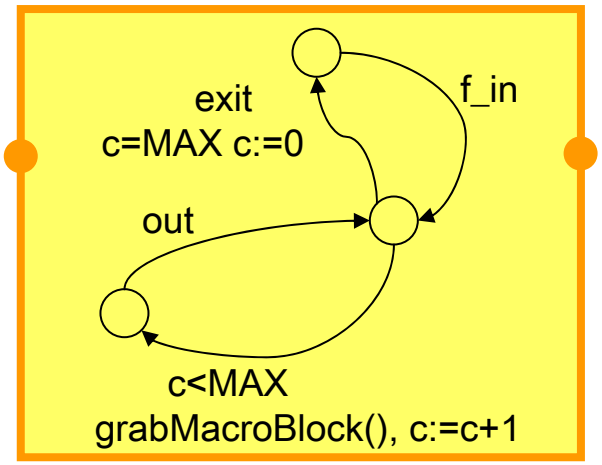


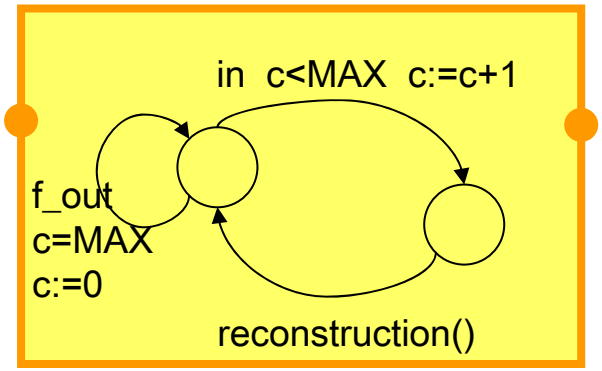

GrabMacroBlock: splits a frame in (W*H)/256 macro blocks, outputs one at a time

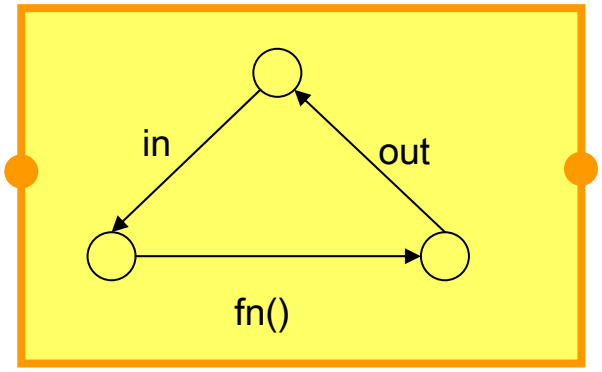
Reconstruction: regenerates the encoded frame from the encoded macro blocks.

: buffered connections

# MPEG4 Video Encoder: Atomic Components




MAX=(W*H)/256
W=width of frame
H=height of frame

# MPEG4 Video Encoder: Features

- BIP code describes a control skeleton for the encoder
  - Consists of 20 atomic components and 34 connectors
  - ~ 500 lines of BIP code
  - Functional components call routines from the encoder library
- The generated C++ code from BIP is ~ 2,000 lines
- The size of the BIP binary is 288 Kb compared to 172 Kb of monolithic binary.

# MPEG4 Video Encoder: BIP Componentization Overhead

Overhead in execution time wrt monolithic code:

- ~66% due to communication (can be reduced by composing components at compile time)
  - –function calls by atomic components to the execution engine for synchronization.
- ~34% due to resolution of non determinism (can be reduced by narrowing the search space at compile time)
  - – time spent by engine to evaluate feasible interactions

Problem: Reduce execution time overhead for componentized code

# Overview

- About component-based construction
- Basic Concepts
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

# Discussion: The BIP Framework - Summary

Framework for component-based modeling encompassing heterogeneity and relying on a **minimal set of constructs and principles**

Clear separation between behavior and architecture

- Architecture = interaction + priority
- Correctness-by-construction techniques for deadlock-freedom and liveness, based on suficient conditions on architecture

Other applications at Verimag

- IF toolset allows layered description of timed systems
- Methodology and tool support for generating scheduled code for real-time applications (work by S. Yovine et al.)

# Discussion: The BIP Framework – Towards a Taxonomy of Systems

A component is defined as a point in the space:

$$\text{Behavior} \times \text{Interaction} \times \text{Priority}$$

Classes of components can be obtained by application of simple transformations

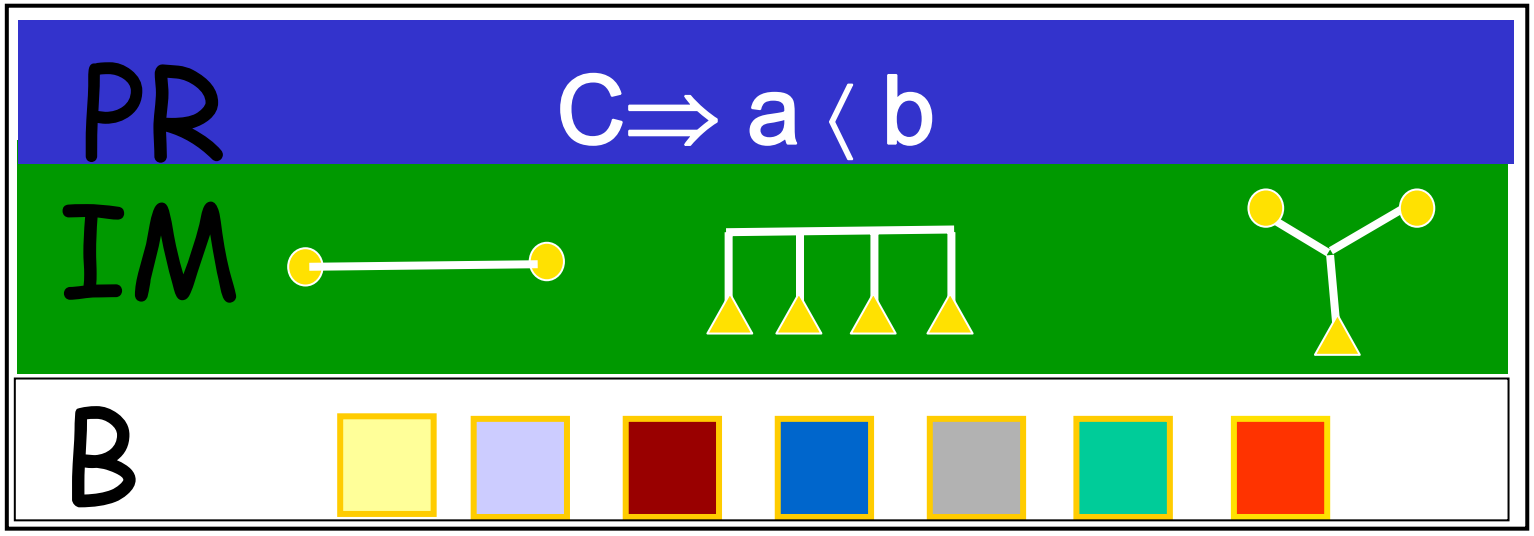
- Behavior: Coupling/Decoupling interaction and computation
- Interaction models : synchronous execution
- Priorities : adding/removing priority rules

Basis for property preservation results and correctness by construction

# Discussion: The BIP Framework – Expressiveness

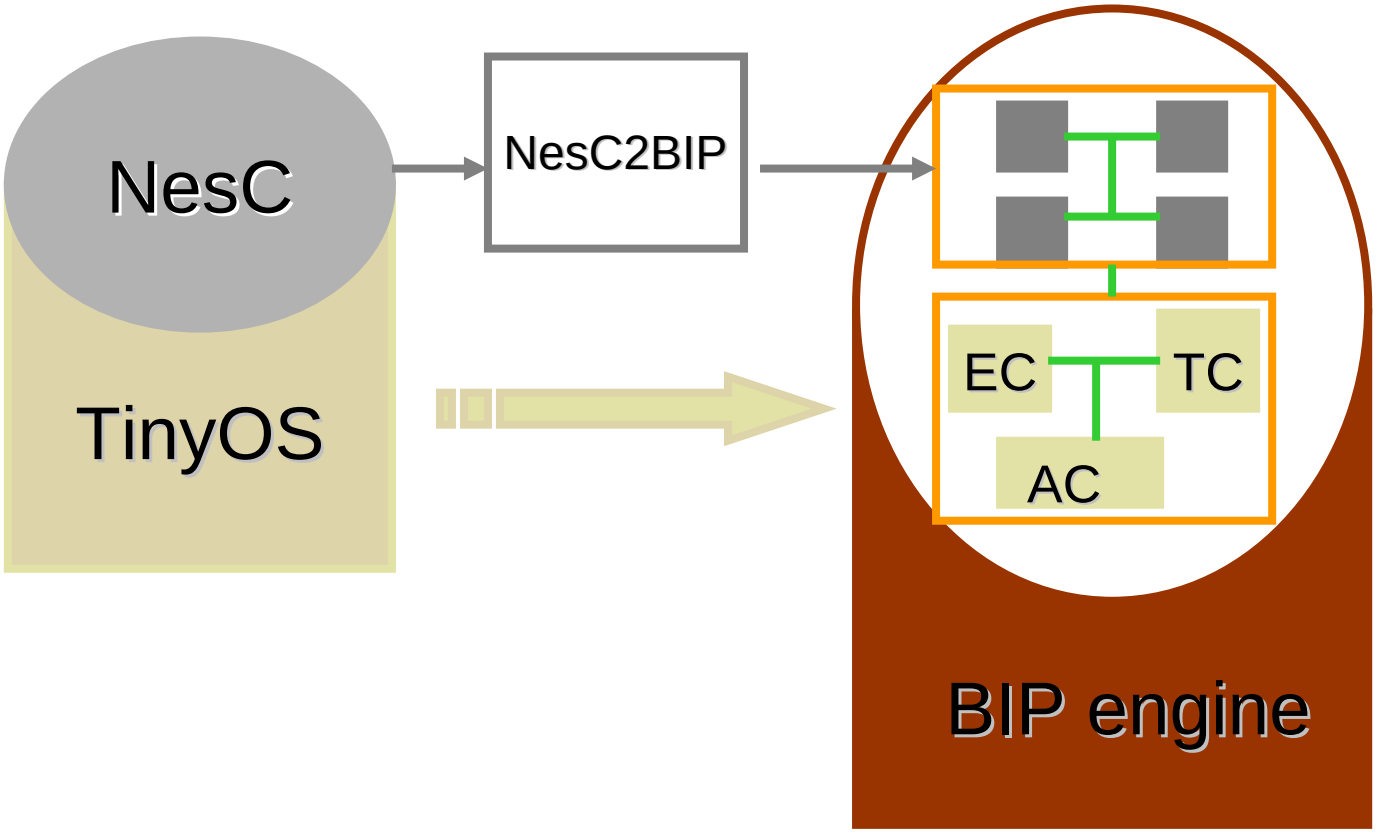
Study new notions of expressiveness where architecture is a first class entity. Existing notions consider glue operators as behavior transformers through operational semantics.

Problem: For given B, IM and PR which coordination problems can be solved?

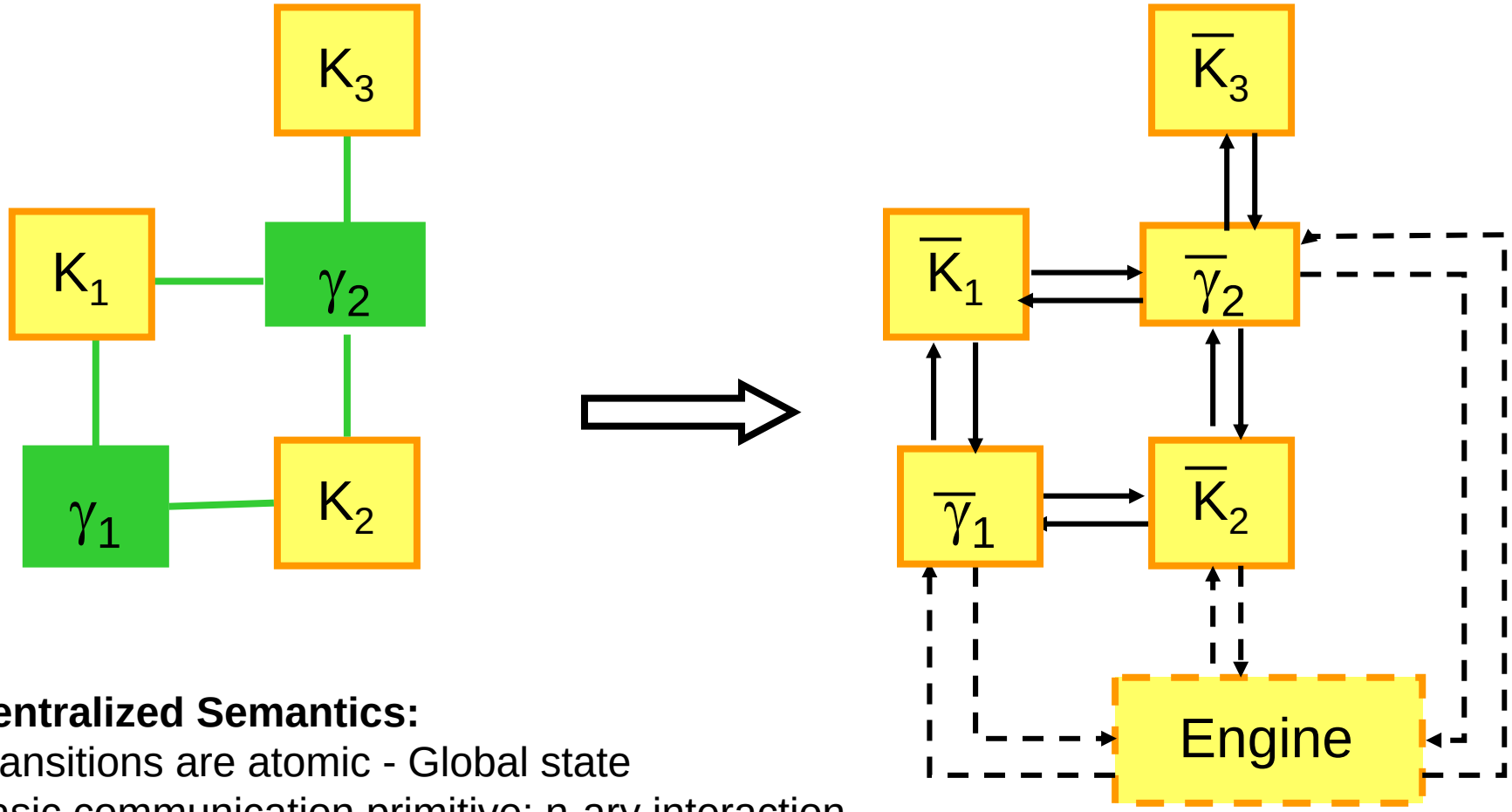


# Discussion: The BIP Framework – Unification

How?



# Discussion: The BIP Framework - Distribution




**Centralized Semantics:**
Transitions are atomic - Global state
Basic communication primitive: n-ary interaction

**Distributed Semantics:**
Transitions are non atomic - Partial state
Basic primitive: send/receive

# References

**Papers available at: http://www-verimag.imag.fr/~sifakis/**


• “The Algebra of Connectors – Structuring Interaction in BIP” EmSoft07

• "Modeling Heterogeneous Real-time Systems in BIP“ SEFM06, IEEE.

• "The Embedded Systems Design Challenge", Invited Paper, FM06.

•“A Framework for Component-based Construction", SEFM05 Keynote talk, September 7-9, 2005, Koblenz, pp 293-300.

• “Composition for Component-Based Modeling”, Science of Computer Programming, vol. 55, pp. 161-183 (March 2005)

• “Scheduler modeling based on the controller synthesis paradigm’’ Journal of Real-time Systems, Vol. 23, pp.55-84, 2002

•“Component-based construction of deadlock-free systems’’, FSTTCS03, LNCS 2194.

• “Priority Systems’’ Proceedings of FMCO’03, LNCS 3188

• “TAXYS: a tool for the development and verification real-time embedded systems” CAV'01. Paris, France, 2001.


**BIP web page:**

http://www-verimag.imag.fr/%7Easync/index.php?view=components