Component-based Construction of Heterogeneous Real-time Systems in BIP

Heraklion, July 22, 2008

Joseph Sifakis
in collaboration with A. Basu, S. Bensalem, S. Bliudze, M. Bozga, Hung Nguyen
VERIMAG Laboratory

Motivation

Building systems from heterogeneous components

- SW Component frameworks, such as
  - Coordination languages, extensions of programming languages : Linda, Javaspaces, Concurrent Fortran, NesC, BPEL
  - Middleware e.g. Corba, Javabeans, .NET
  - Software development environments: PCTE, SWbus, Softbench, Eclipse
- System modeling languages: Statecharts, UML, Simulink/Stateflow, Metropolis, Ptolemy
- Hardware description languages: VHDL, Verilog, SystemC

We need an all-encompassing component-based construction framework for

- Mastering complexity through componentization
- Enhanced verifiability by compositional reasoning
- Comparing different architectural solutions of the same problem

Motivation

Provide a **framework** for describing and analyzing coordination between components in terms of **tangible, well-founded and organized concepts**
The framework should

- be **expressive** enough to directly encompass heterogeneity of synchronization (rendezvous and broadcast) and execution mechanisms (synchronous and asynchronous) – adequate notion of expressiveness
- use a **minimal set of constructs** and principles
- treat interaction and system architecture as **first class entities** that can be composed and analyzed - independently of the behavior of individual components
- provide automated support for component integration and generation of glue code meeting given requirements

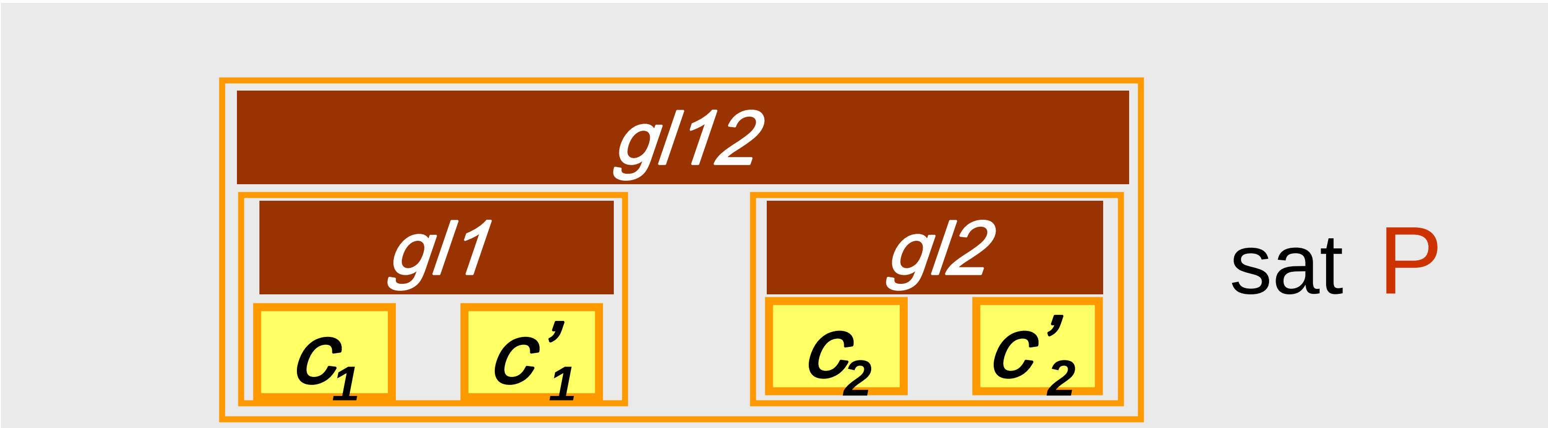
OVERVIEW

- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

Component-based Construction: The Problem

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

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



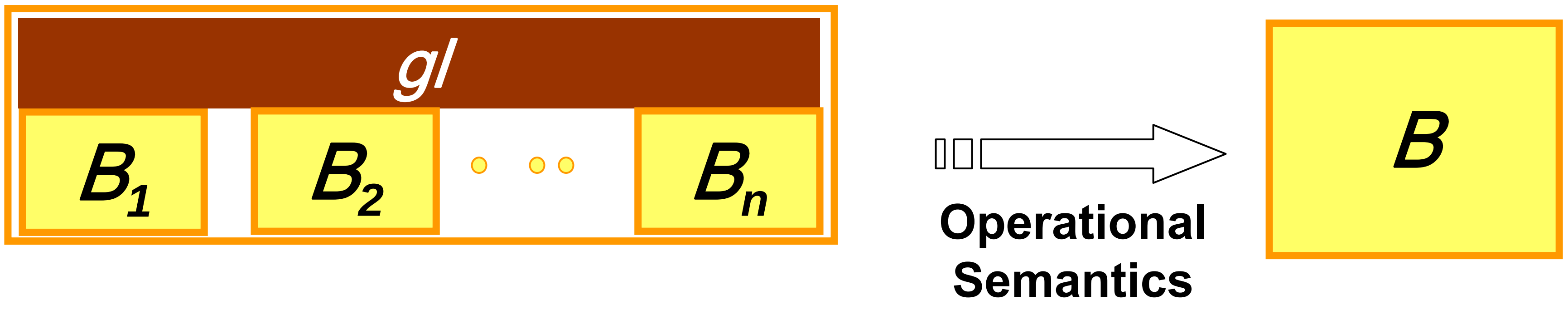
- Move from frameworks based on single composition operators to frameworks based on families of composition operators

- Enhanced expressiveness for modeling coordination mechanisms such as such as protocols, schedulers, buses

Glue operators

Operational Semantics

- The meaning of a composite component is an atomic component



Algebraic framework

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

Glue operators

A **glue operator** is a set of derivation rules of the form

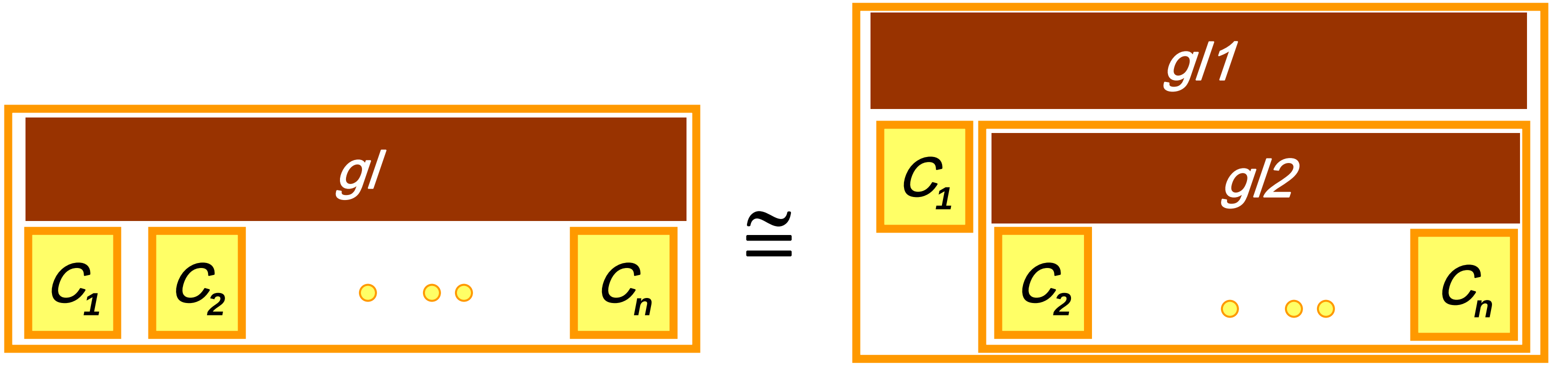
$$\frac{\{q_i - a_i \rightarrow q'_i\}_{i \in I} \quad \{\neg\, q_k - a_{ks} \rightarrow\}_{k \in K}}{(q_1, \ldots, q_n) - a \rightarrow (q'_1, \ldots, q'_n)}$$

- $a = \bigcup_{i \in I} a_I$ is an interaction
- $q'_i = q_i$ for $i \notin I$
- there is at most one positive premise for each argument (component)
- there is at least one positive premise

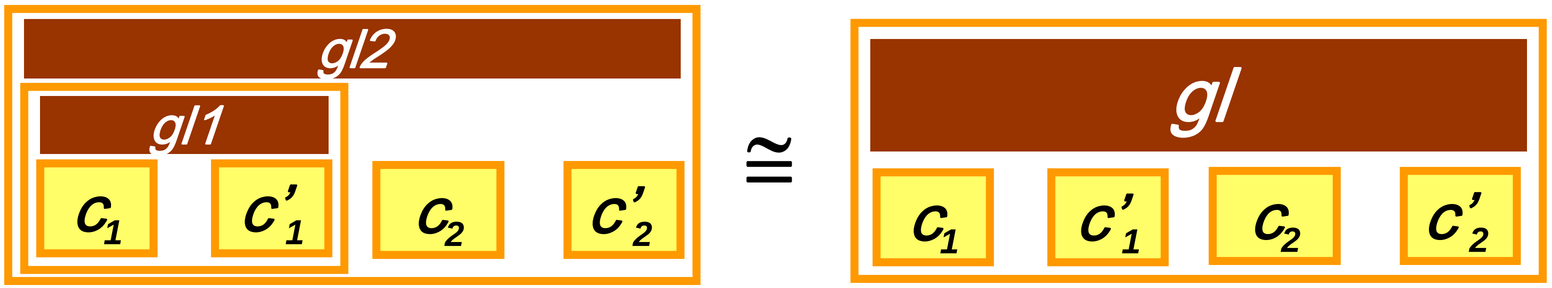
A **glue** is a set of glue operators

Incrementality

1. Decomposition



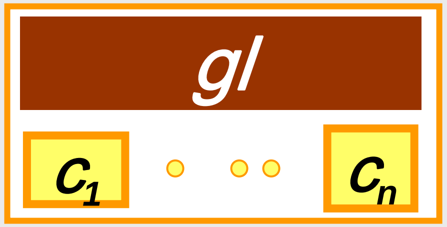
2. Flattening



Compositionality

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

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

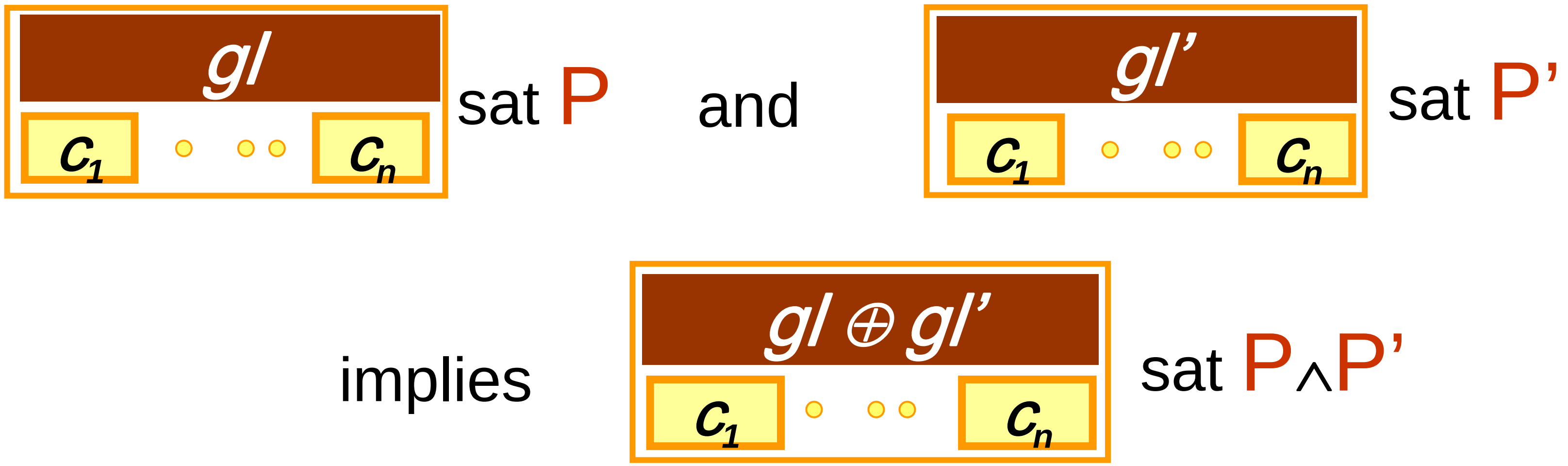


sat $\widetilde{gl}(P_1, .., P_n)$

We need compositionality results for the preservation of progress properties such as deadlock-freedom and liveness as well as extra-functional properties

Composability

Rules for property-preserving composition of designs



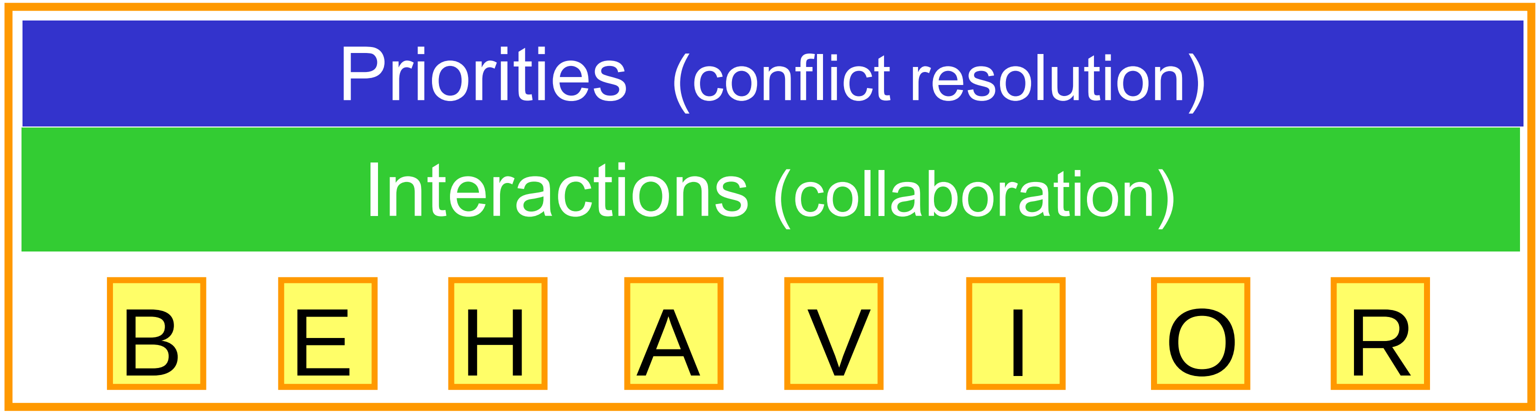
Property stability phenomena are poorly understood.
We need composability results e.g. interaction of features in middleware, composability of scheduling algorithms, theory for reconfigurable systems

- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

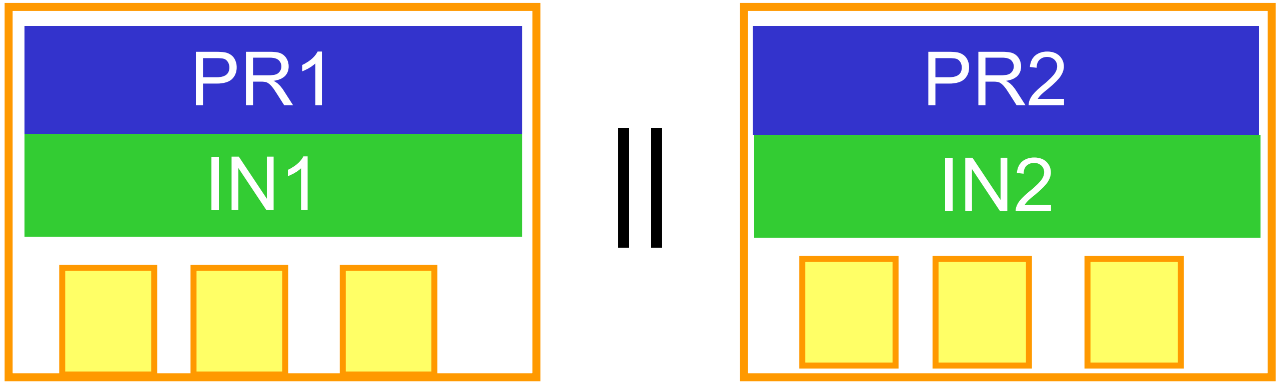
O V E R V I E W

BIP: Basic Concepts

Layered component model

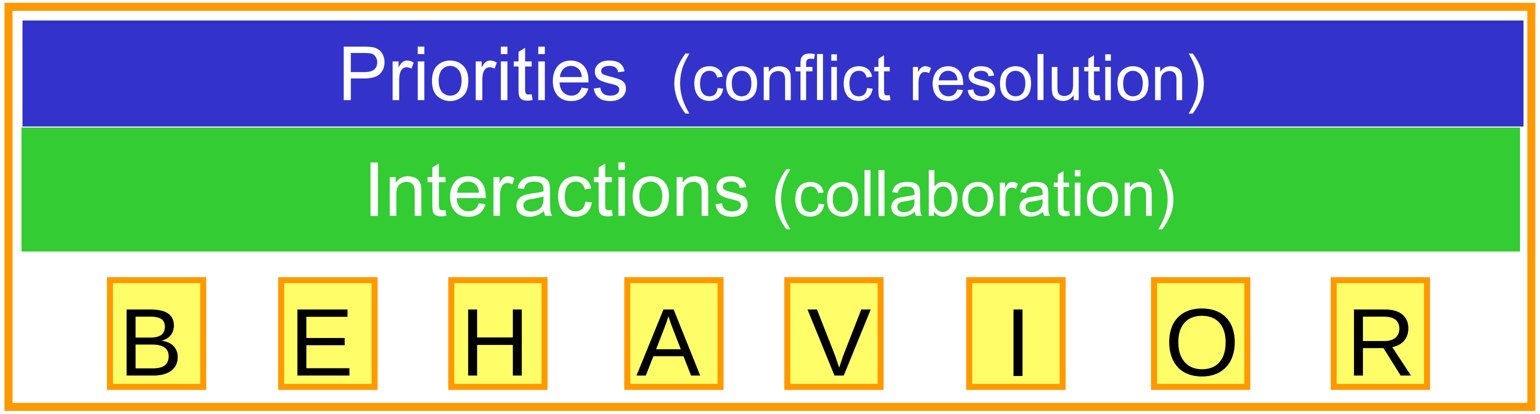


Composition (incremental description)

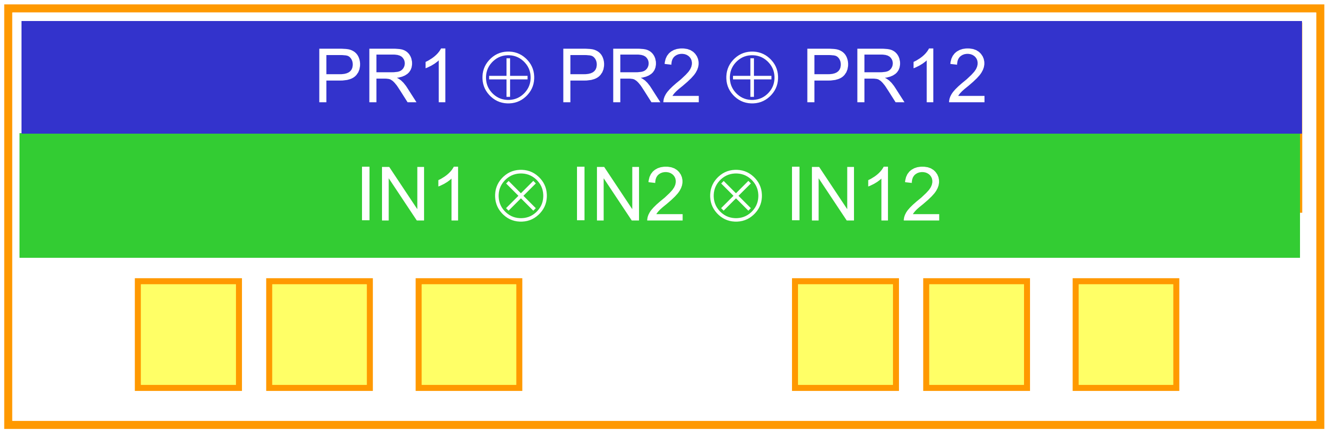


BIP: Basic Concepts

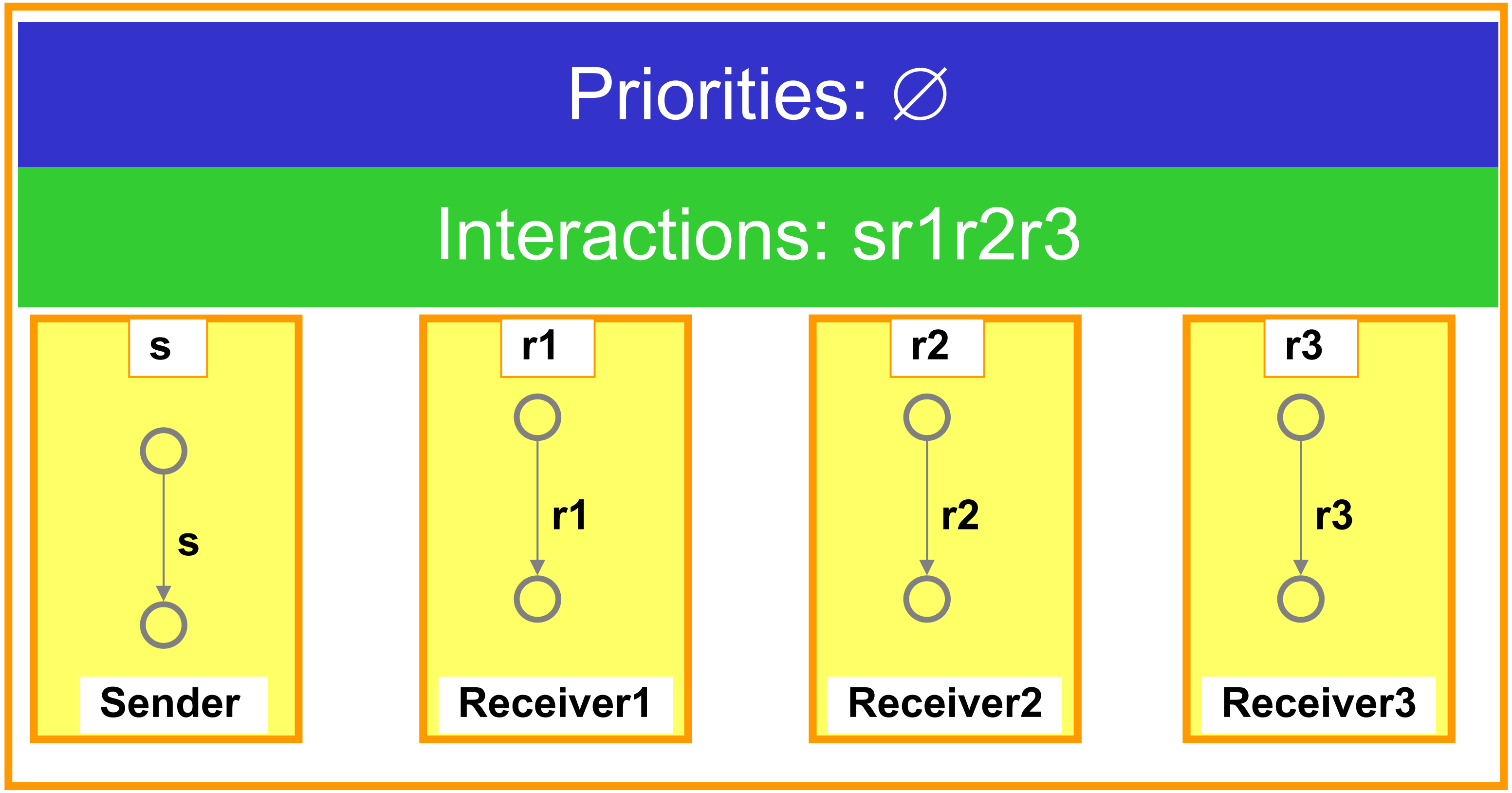
Layered component model



Composition (incremental description)

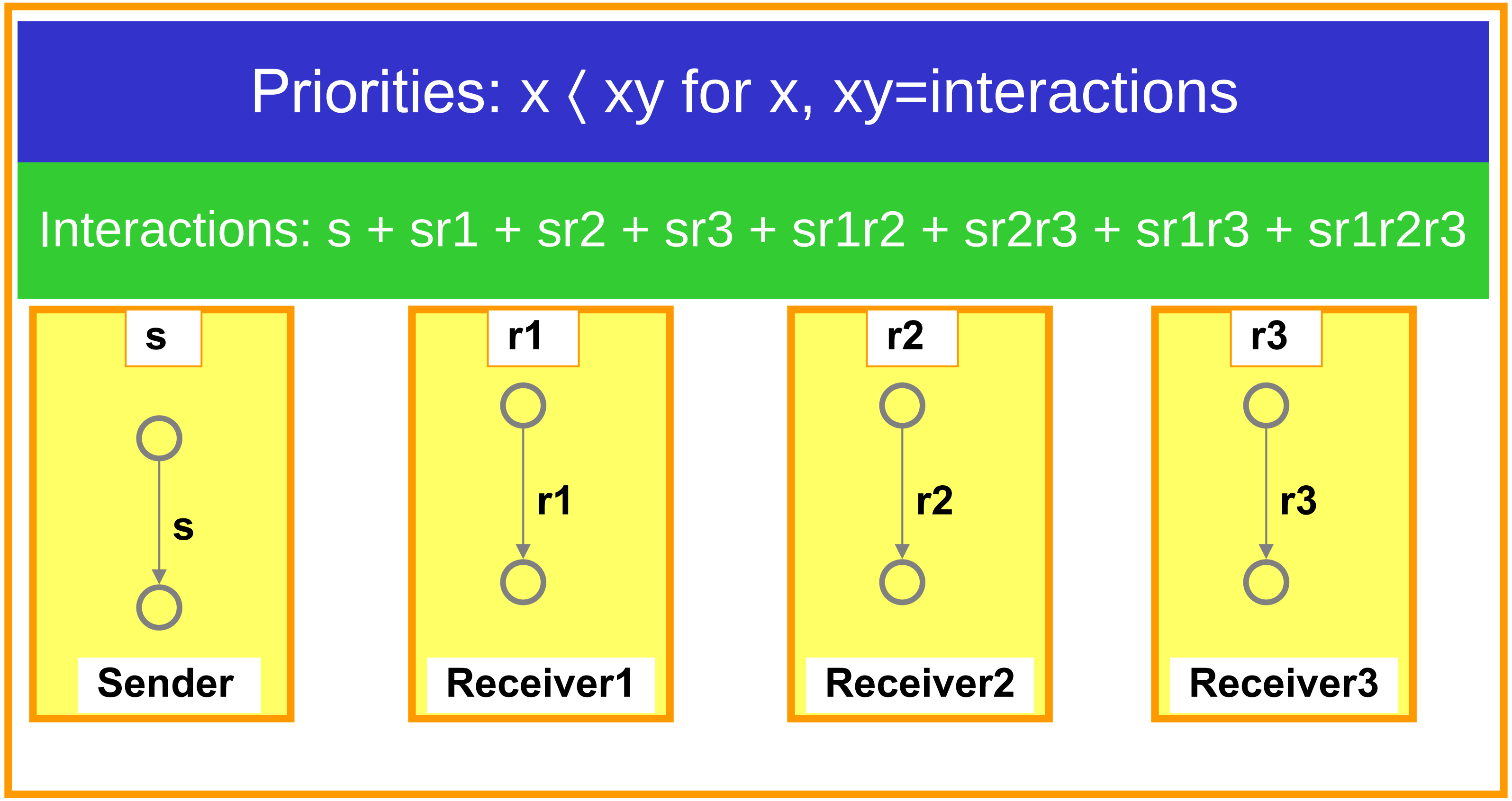


BIP: Basic Concepts



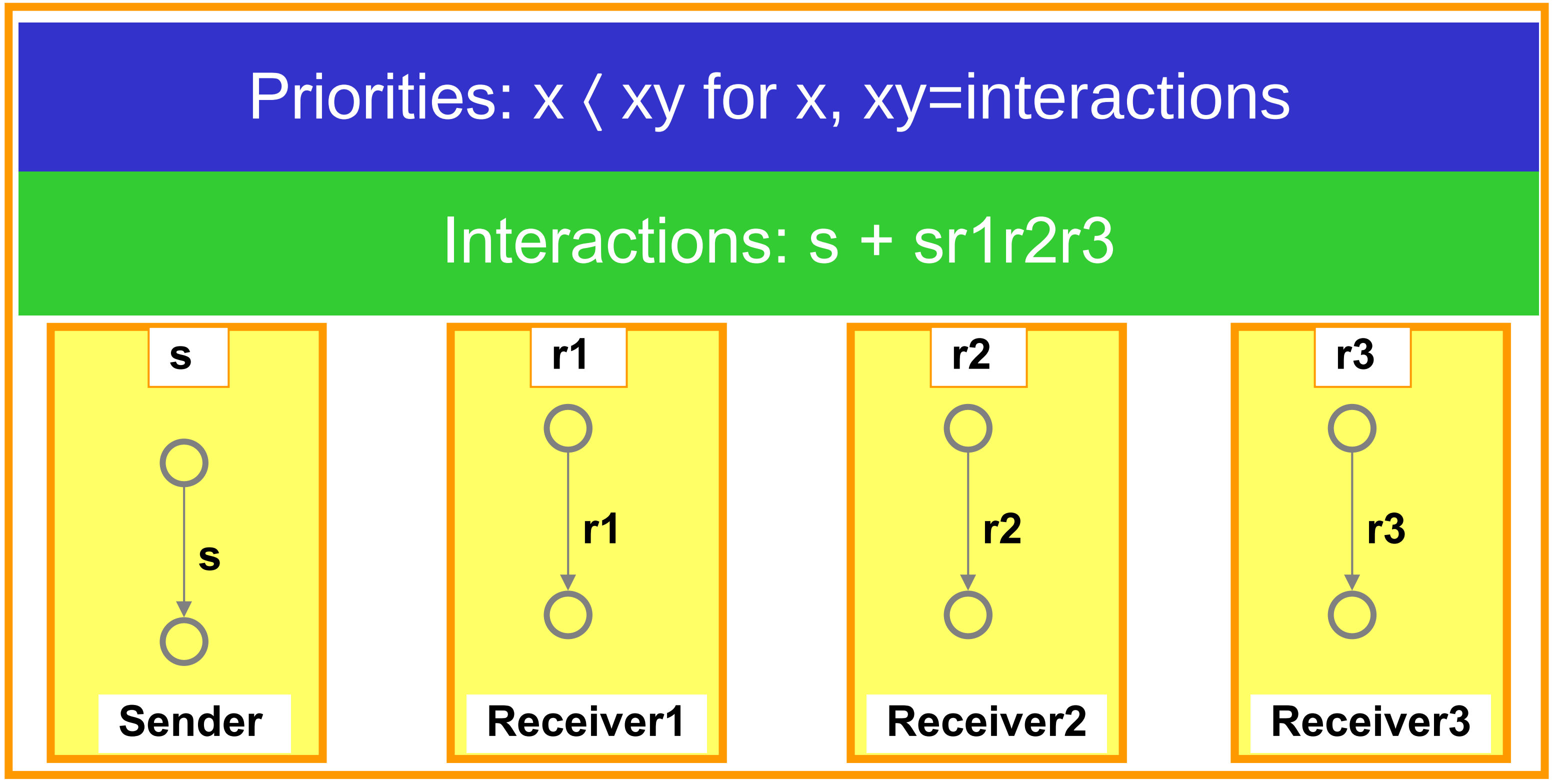
Rendezvous

BIP: Basic Concepts



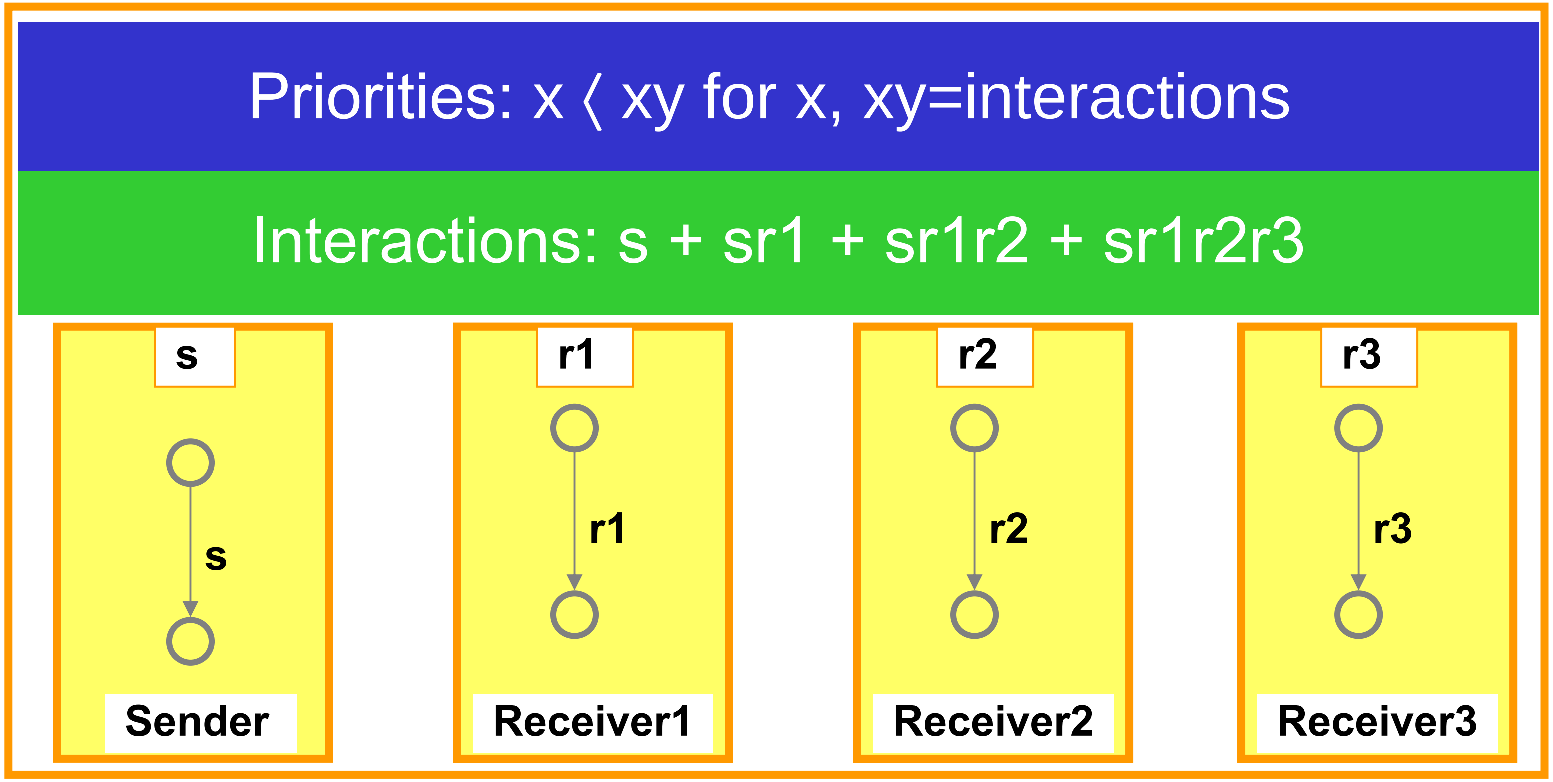
Broadcast

BIP: Basic Concepts



Atomic Broadcast

BIP: Basic Concepts



Causal Chain

BIP: Basic Concepts - Semantics

- a set of atomic components $\{B_i\}_{i=1..n}$ where $B_i = (Q_i, 2^{P_i}, \rightarrow_i)$
- a set of interactions γ
- priorities π, partial order on interactions

$\Big\}\ \pi\,\gamma\,(B1,.., Bn)$

Interactions

$$\frac{a \in \gamma \ \wedge\ \forall i \in [1,n]\ \ q_i \text{ - } a \cap P_i \rightarrow_i q'_i}{(q_1 ,.., q_n) \text{ - } a \rightarrow_\gamma (q'_1 ,.., q'_n) \text{ where } q'_l = q_l \text{ if } a \cap P_l = \emptyset}$$

Priorities

$$\frac{q\text{- } a \rightarrow_\gamma q' \wedge \neg (\exists\ q\text{- } b \rightarrow_\gamma \ \wedge a\ \pi\ b\,)}{q\text{- } a \rightarrow_\pi q'}$$

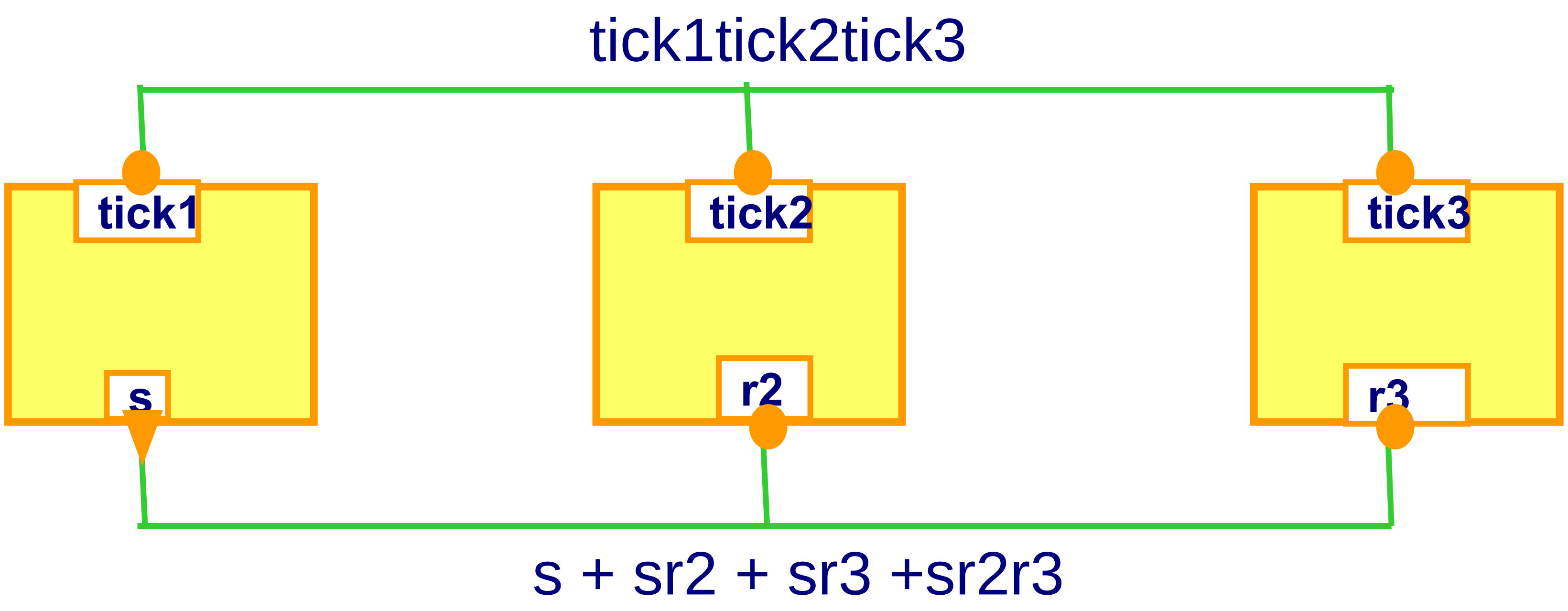
O
V
E
R
V
I
E
W

- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

Simple Connectors

Express interactions by combining two protocols: rendezvous and broadcast

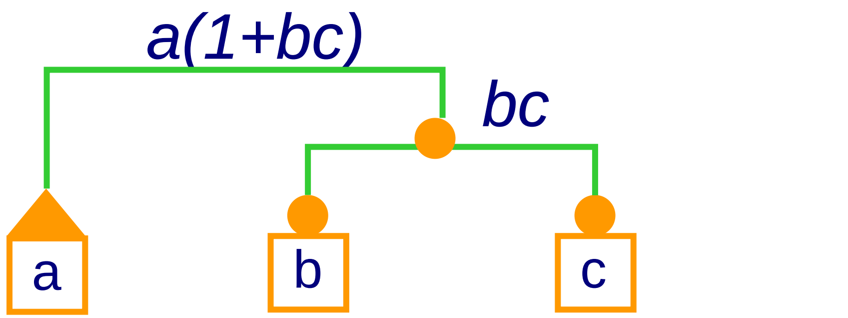
- A ***connector*** is a set of ports that can be involved in an interaction
- Port attributes (***trigger*** ▼ , ***synchron*** ●) are used to model rendezvous and broadcast.
- An ***interaction*** of a connector is a set of ports such that: either it contains some trigger or it is maximal.



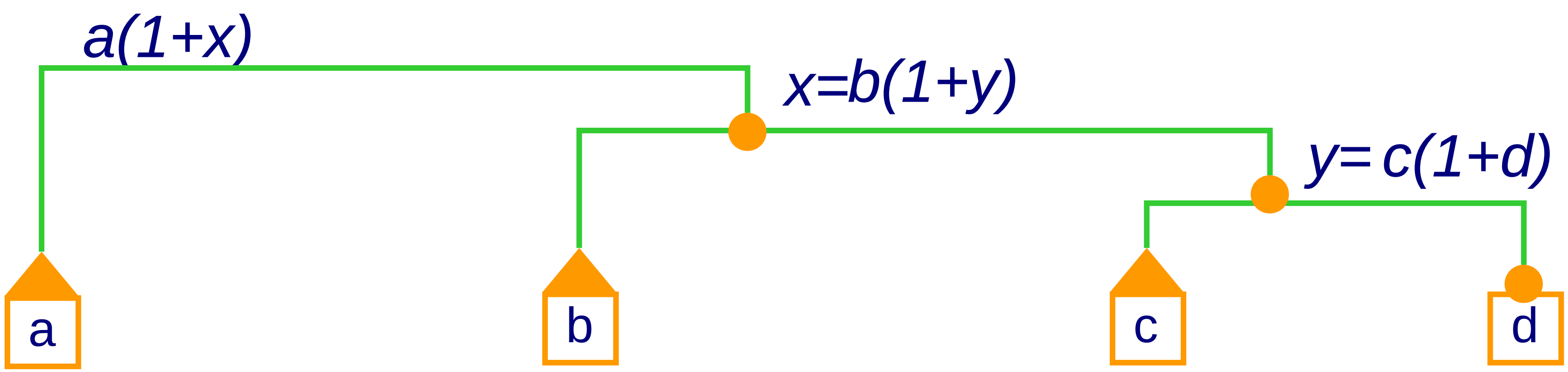
Hierarchical Connectors

Atomic Broadcast:

a+abc

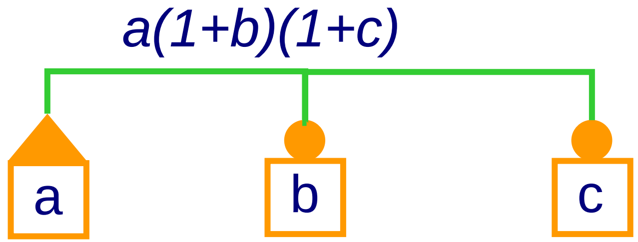


Causality chain: a+ab+abc+abcd

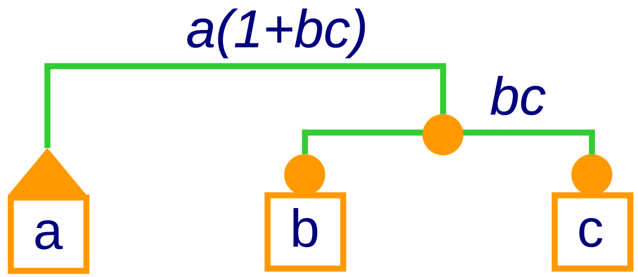


The Algebra of Connectors

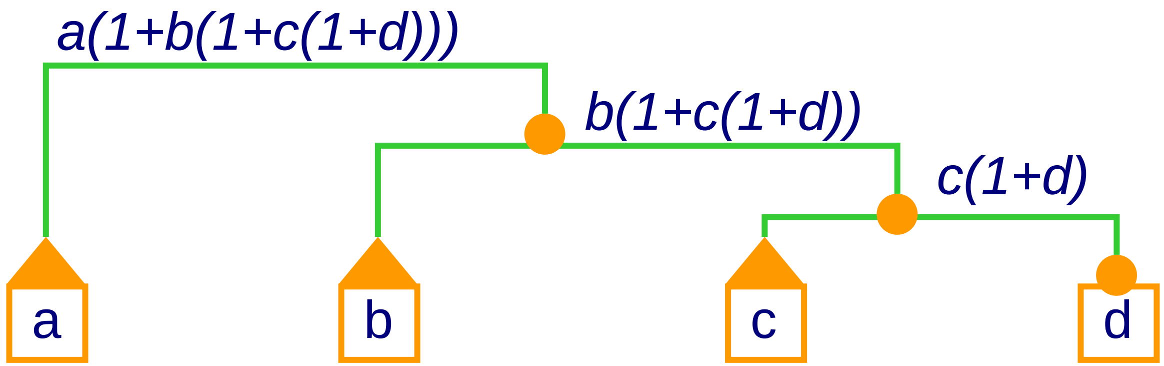
Broadcast
a’bc



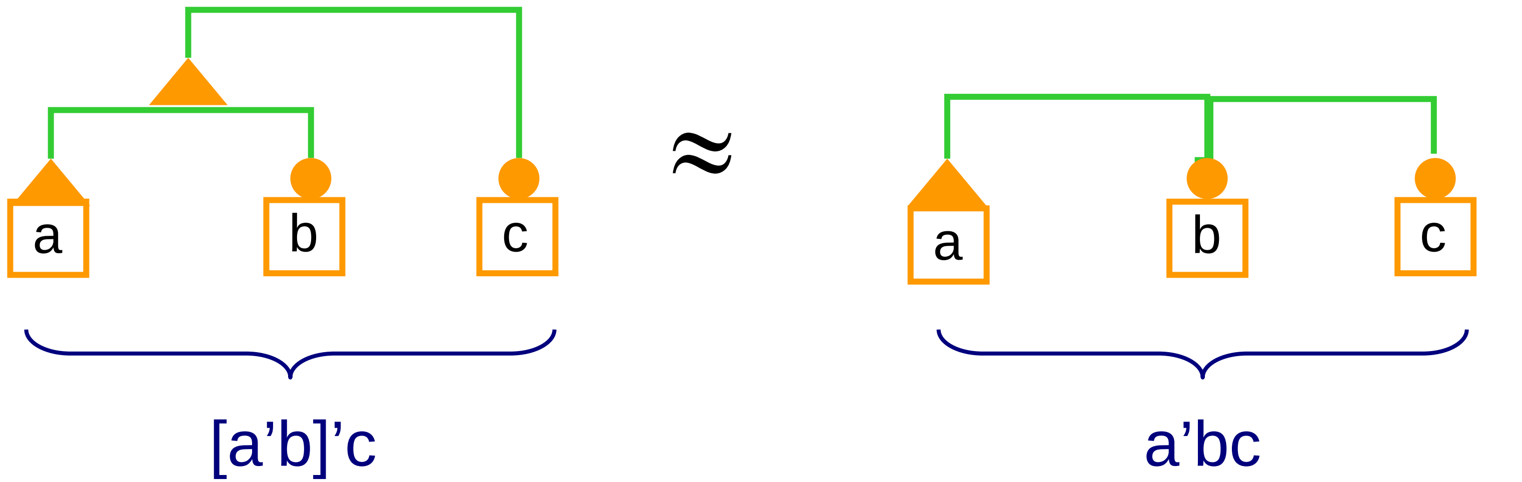
Atomic Broadcast
a’[bc]

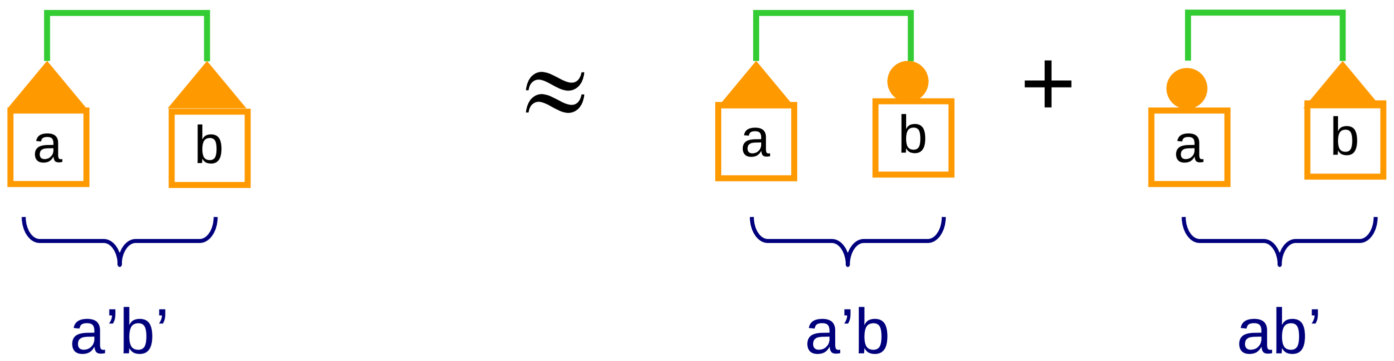


Causality chain
a’[b’[c’d]]



The Algebra of Connectors





The Algebra of Connectors AC(P)

Syntax: s ::= [0] | [1] | [p] |[x] (synchrons)

t ::= [0]' | [1]' | [p]' | [x]' (triggers)

x ::= s | t | x.x | x + x

where P is a set of ports, such that $0,1 \notin P$

+	*union*	idempotent, associative, commutative, identity [0]
.	*fusion*	idempotent, associative, commutative, identity [1], distributive wrt + ([0] is not absorbing)
[ ], [ ]'	*typing*	unary operators

Semantics: defined as a function | |: AC(P) $\rightarrow 2^{2^P}$

Results *[Bliudze&Sifakis, EmSoft 07]***:**

- Axiomatization
- Boolean representation allowing efficient implementation

The Algebra of Connectors: Boolean representation

β: $AC(P) \rightarrow B(P)$ where $B(P)$ is the boolean calculus on P

For $P=\{p,q,r,s,t\}$

$\beta(pq)$ $= p \wedge q \wedge \neg r \wedge \neg s \wedge \neg t$

$\beta(p'qr)$ $= p \wedge \neg s \wedge \neg t$

$\beta(p+q)$ $=(p \wedge \neg q \vee \neg p \wedge q) \wedge \neg r \wedge \neg s \wedge \neg t$

$\beta(0)$ $= false$

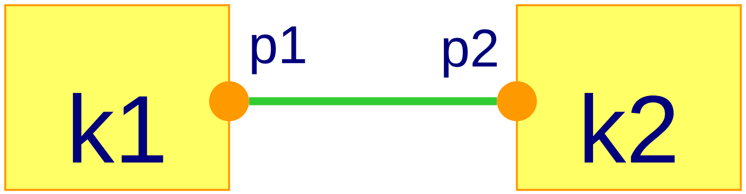
$\beta(1)$ $= \neg p \wedge \neg q \wedge \neg r \wedge \neg s \wedge \neg t$

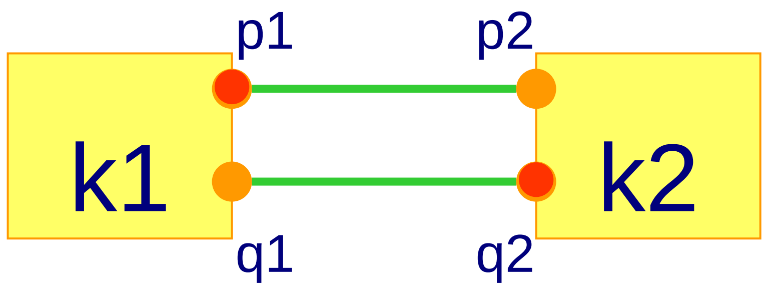
$\beta(1+p'q'r's't')$ $= true$

Boolean representation depends on the set of ports P

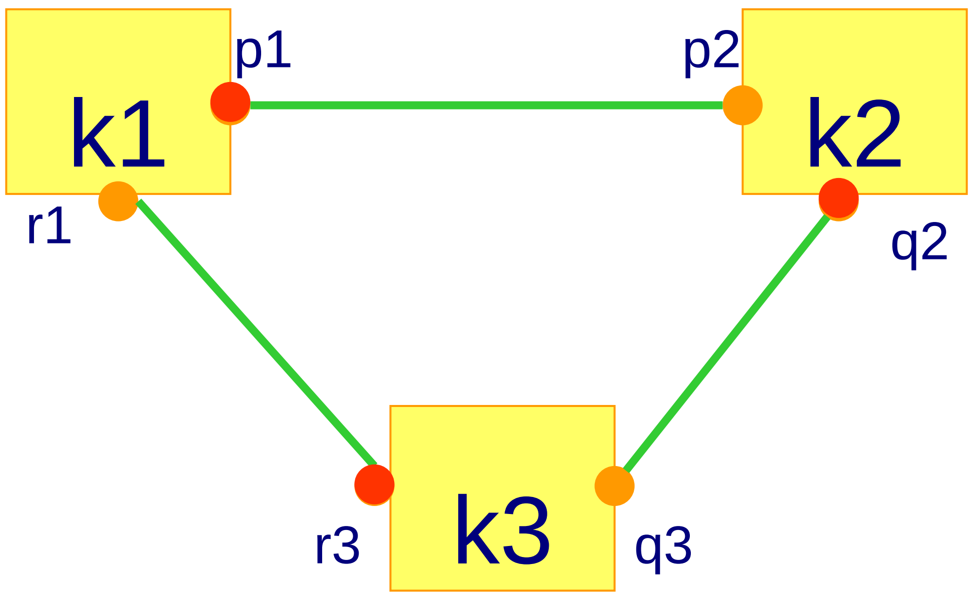
Compositional Deadlock Verification

For K1,K2,K3 deadlock-free components





$$D = en(p1) \wedge \neg\, en(p2) \wedge en(q2) \wedge \neg\, en(q1)$$



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

Compositional Deadlock Verification

Eliminate potential deadlocks D by checking that

$I \wedge D = false$

for some global invariant I computed compositionally

Example	Nb Comp	Nb Ctrl St	Nb Bool Var	Nb Int Var	Nb Pot Deadl	Nb Rm Deadl	time
Temperature Control (2 rods)	3	6	0	3	8	8	3s
Temperature Control (4 rods)	5	10	0	5	32	15	1m05s
UTOPAR (4 cars,9 CU)	14	45	4	26	??	0	1m42s
UTOPAR (8 cars,16 CU)	25	91	8	50	??	0	22m02s
R/W (50 readers)	52	106	0	1	~10^15	0	1m15s
R/W (100 readers)	102	206	0	1	~10^30	0	15m28s
R/W (130 readers)	152	266	0	1	~10^39	0	29m13s

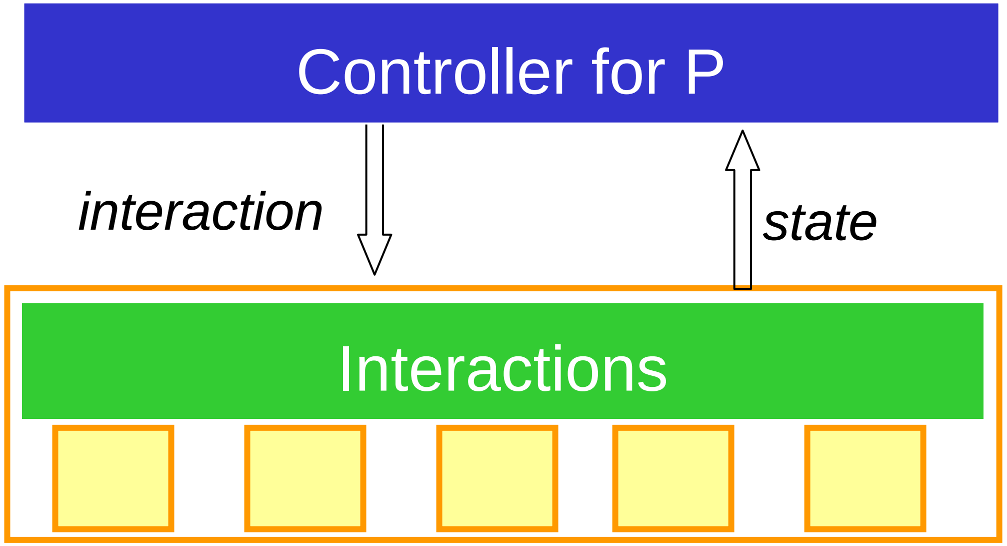
Results obtained by using the D-Finder tool: http://www-verimag.imag.fr/~thnguyen/tool/

OVERVIEW

- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

Priorities as Controllers

Controller restricts non determinism to enforce a property P

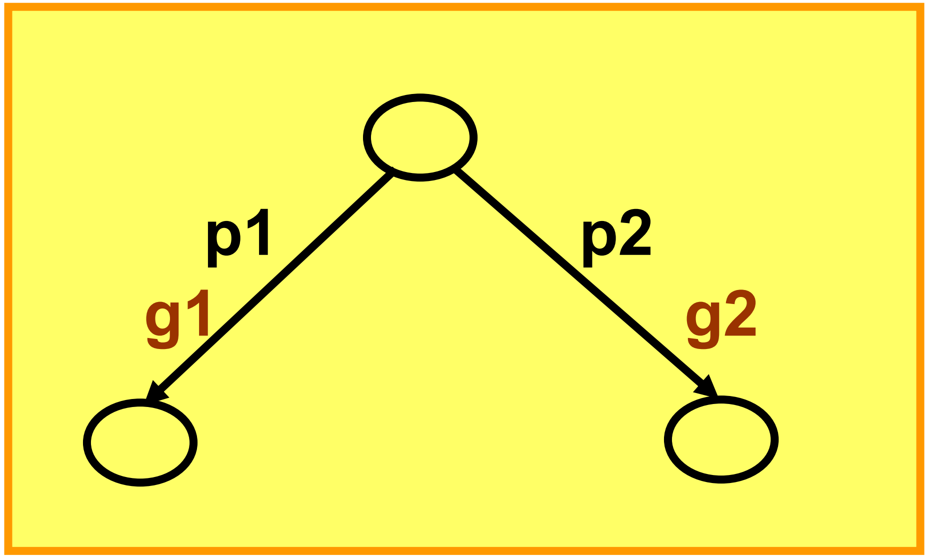


Results [Goessler&Sifakis, FMCO2003] :

- Controllers enforcing deadlock-free state invariants can be described by dynamic priorities
- Conversely, for any dynamic priorities there exists a controller enforcing a deadlock-free state invariant

Priorities: Definition

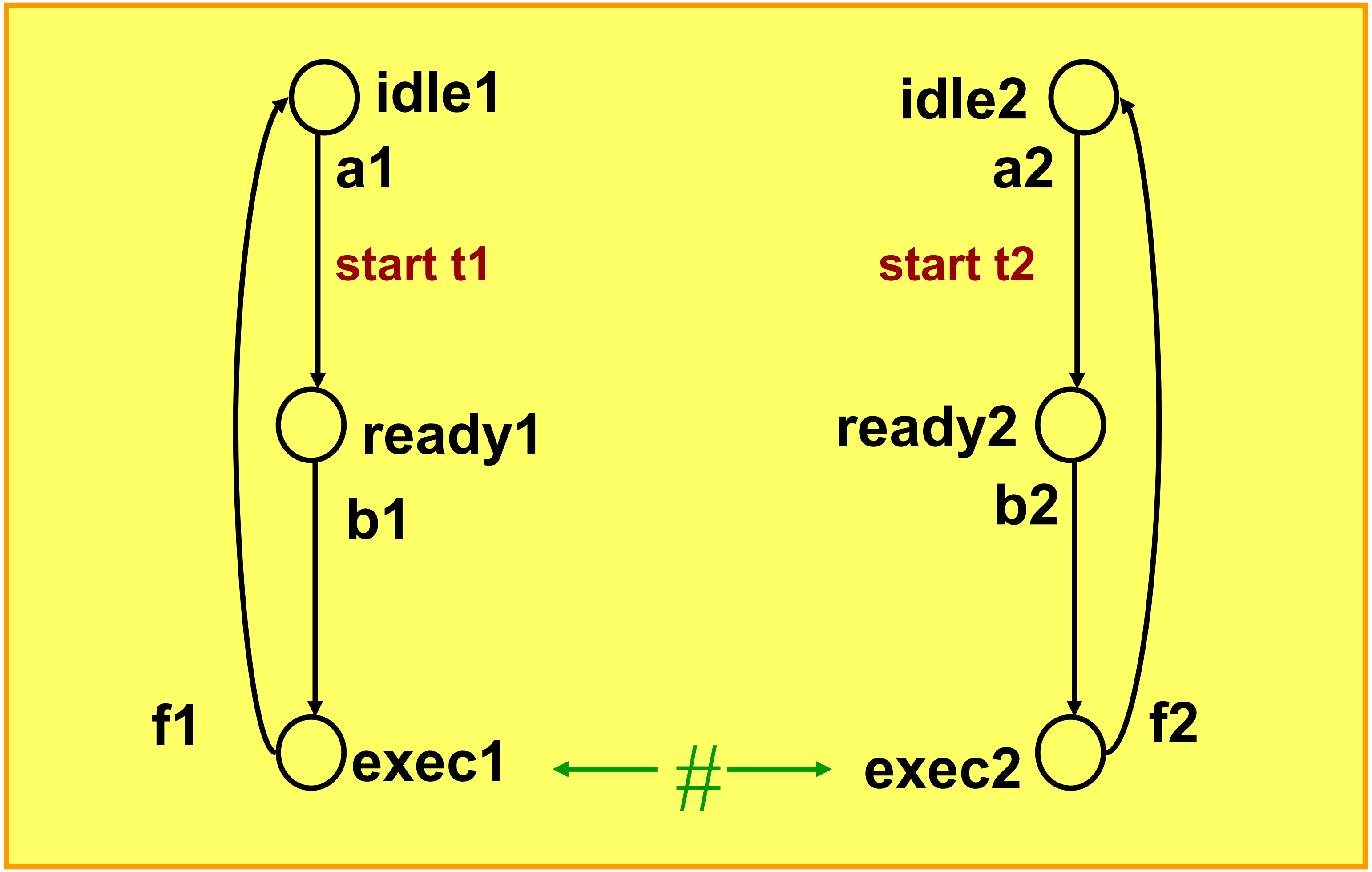
Priority rules



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

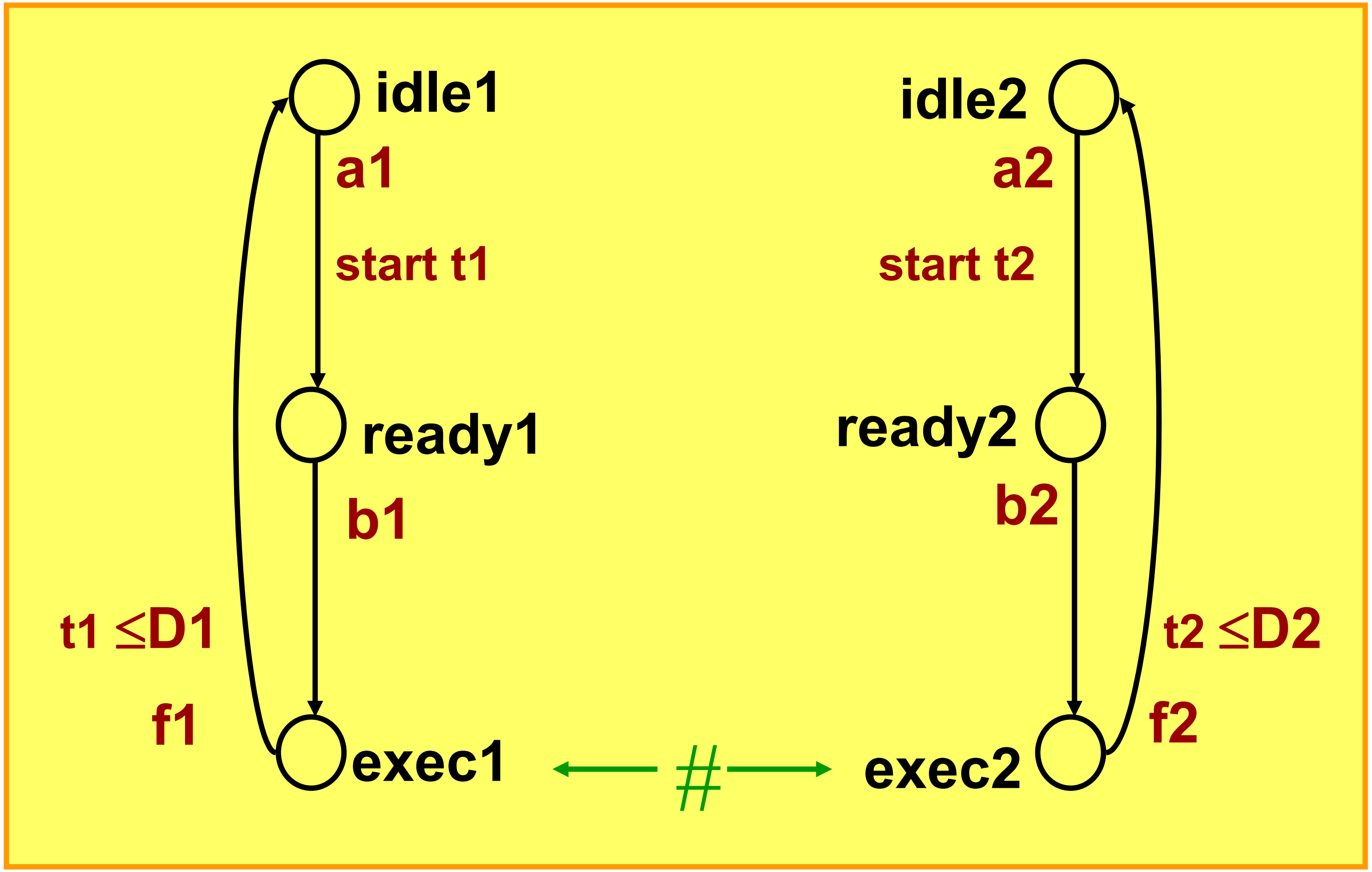
Priorities: FIFO policy

$$PR : t1 \leq t2 \rightarrow b1 \langle b2 \qquad t2 < t1 \rightarrow b2 \langle b1$$

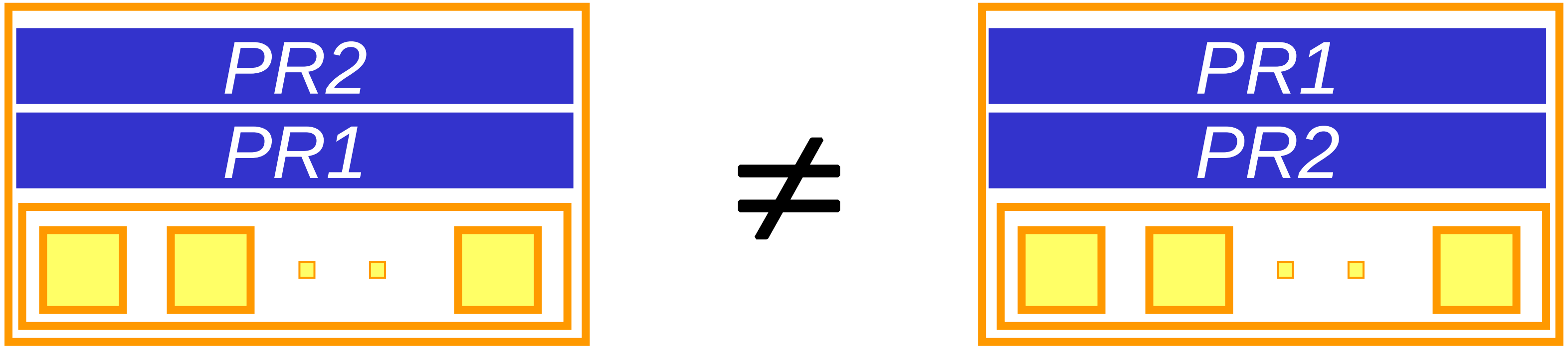


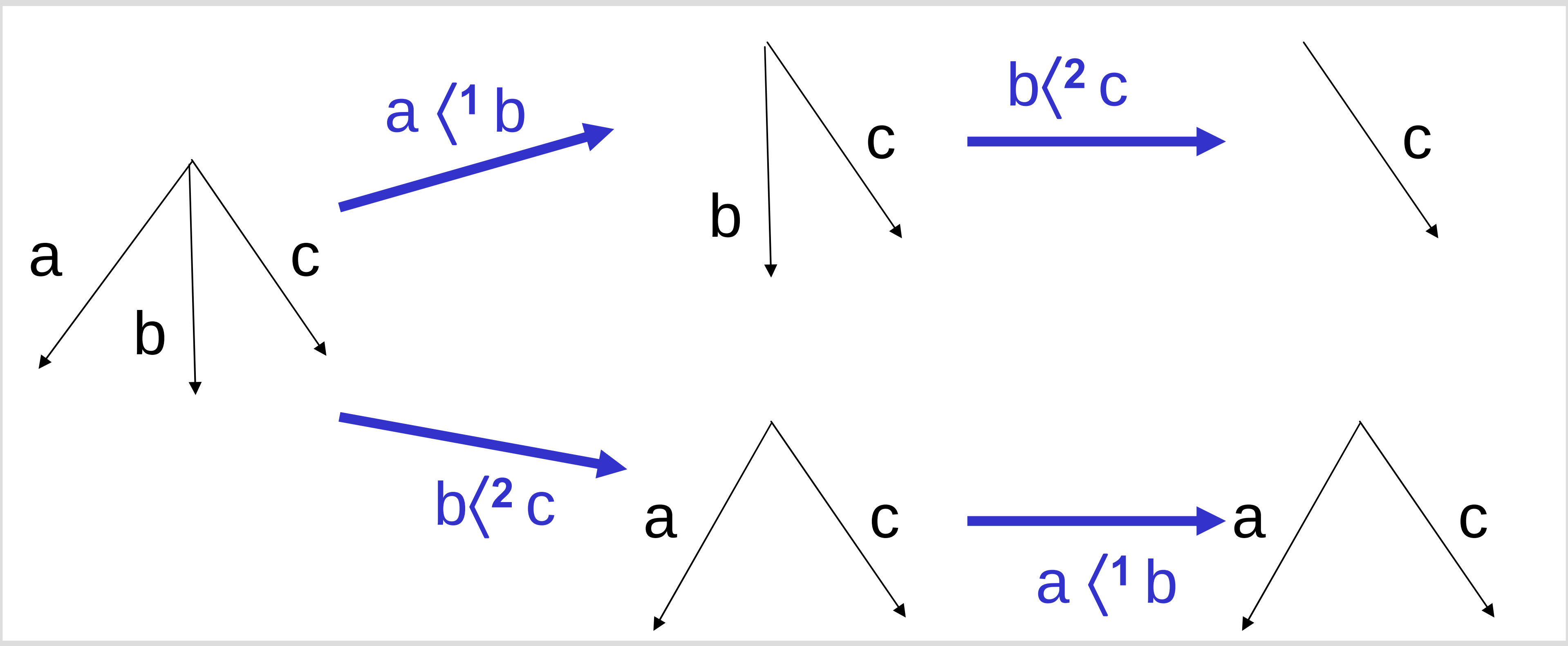
Priorities: EDF policy

PR: D1-t1≤ D2- t2 → b2〈 b1   D2-t2< D1-t1 → b1〈 b2



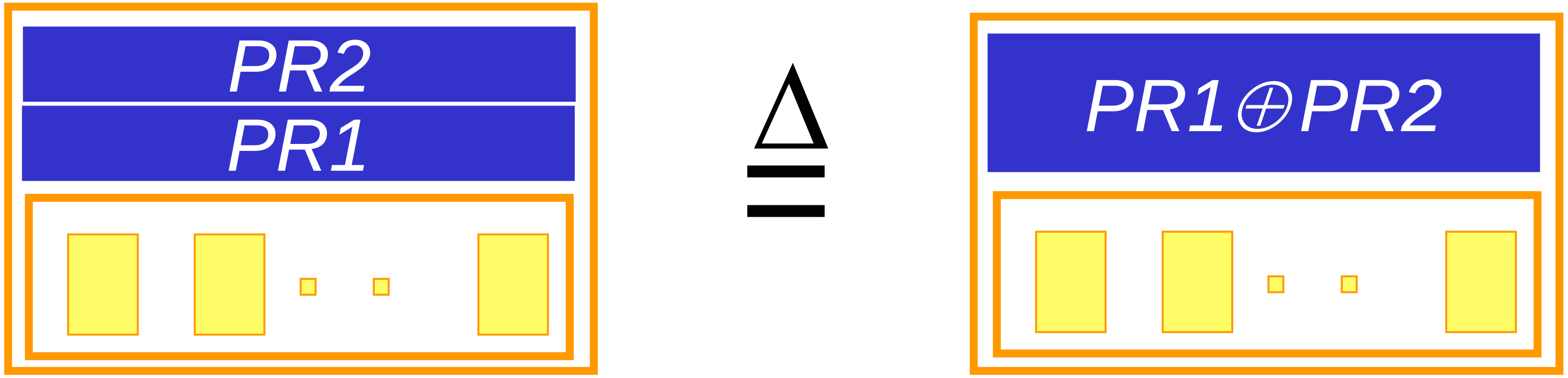
Priorities: Composition





Priorities: Composition

We take:



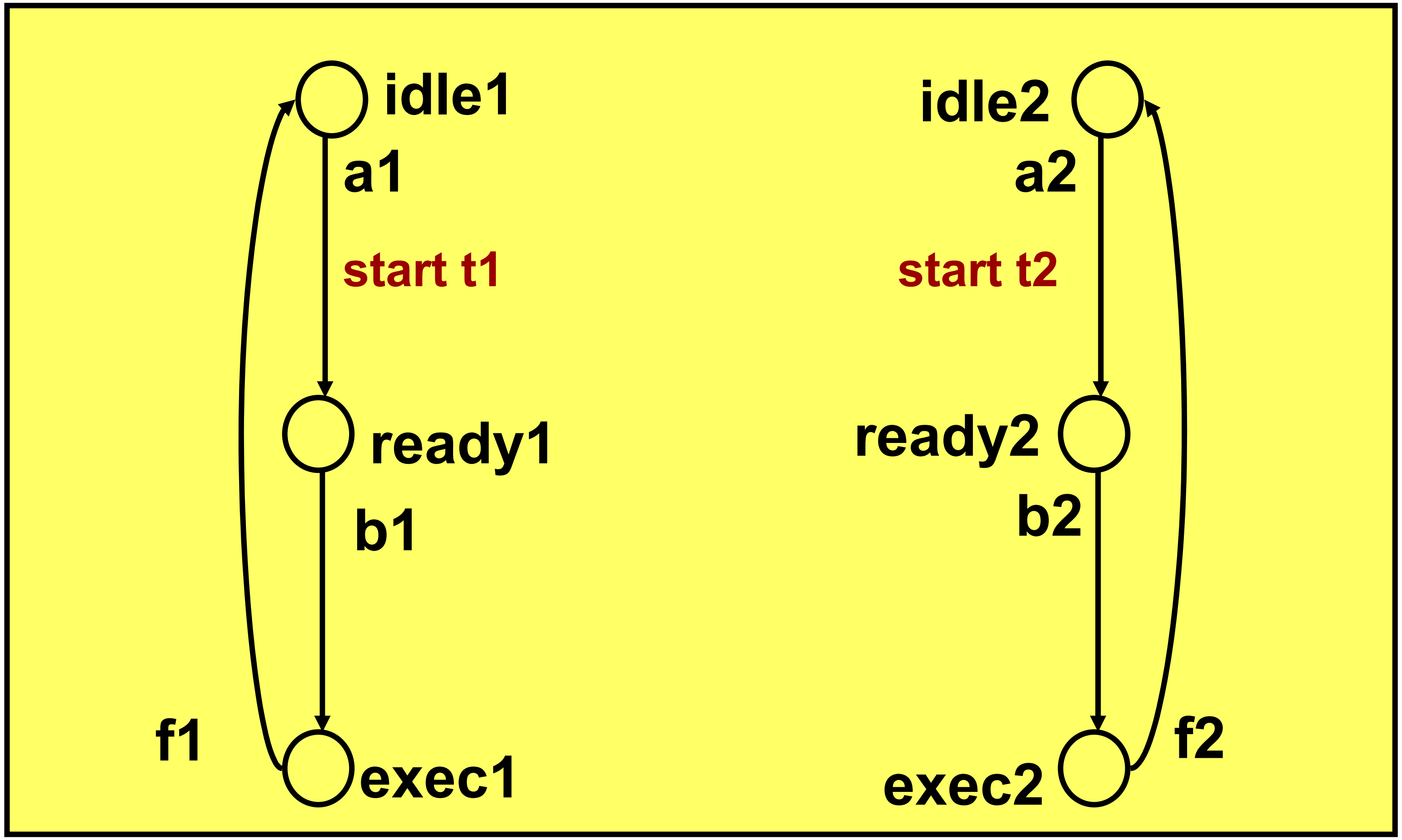
PR1⊕ PR2 is the least priority containing PR1∪PR2

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

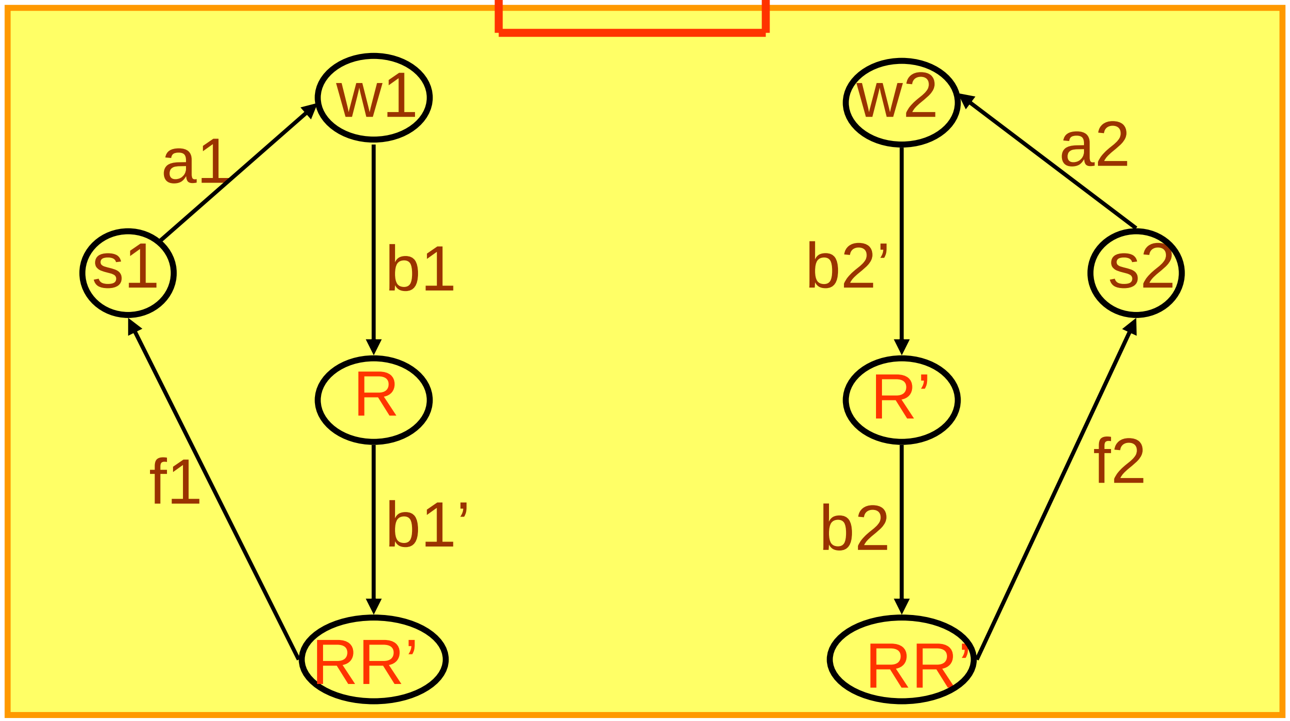
t1≤ t2 → b1⟨ b2	t2< t1 → b2⟨ b1
true → b1⟨ f2	true → b2⟨ f1



Priorities: Mutual Exclusion - Example

PR : b1 ⟨ f2 b2 ⟨ { f1, b1’} (mutex on R)

PR’: b2’ ⟨f1 b1’ ⟨ { f2, b2 } (mutex on R’)

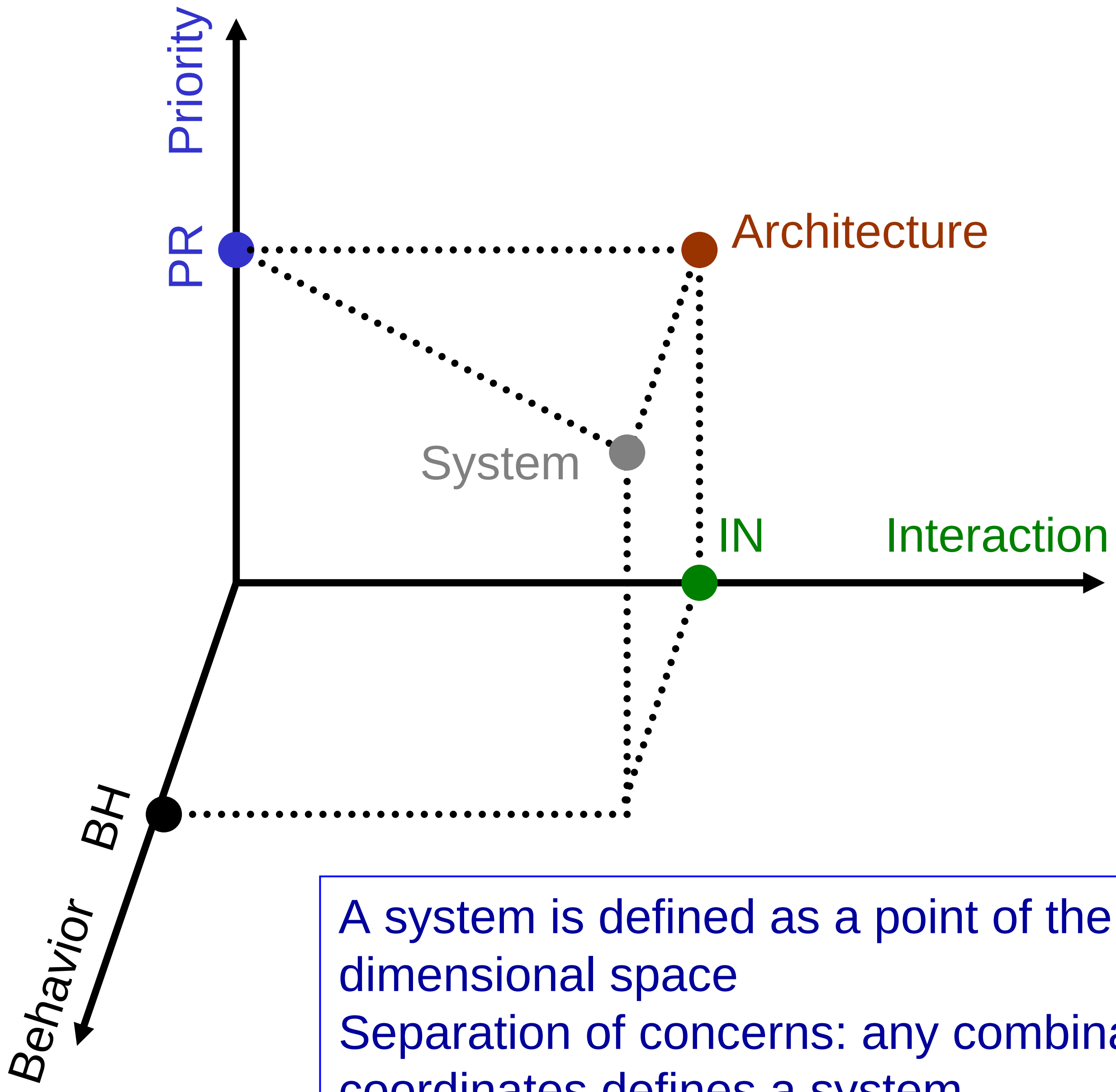


Risk of deadlock: PR⊕PR’ is not defined!

OVERVIEW

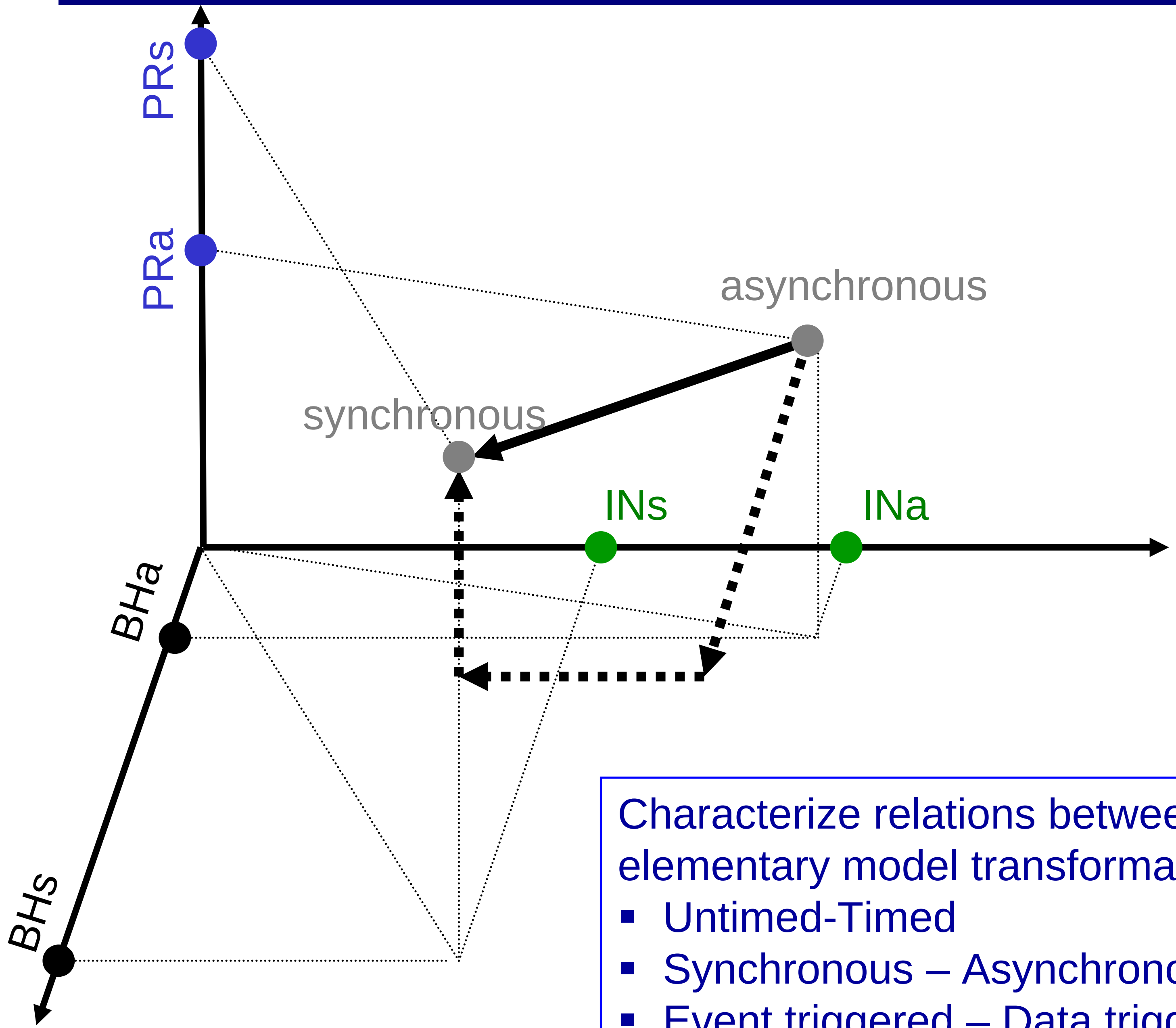
- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

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

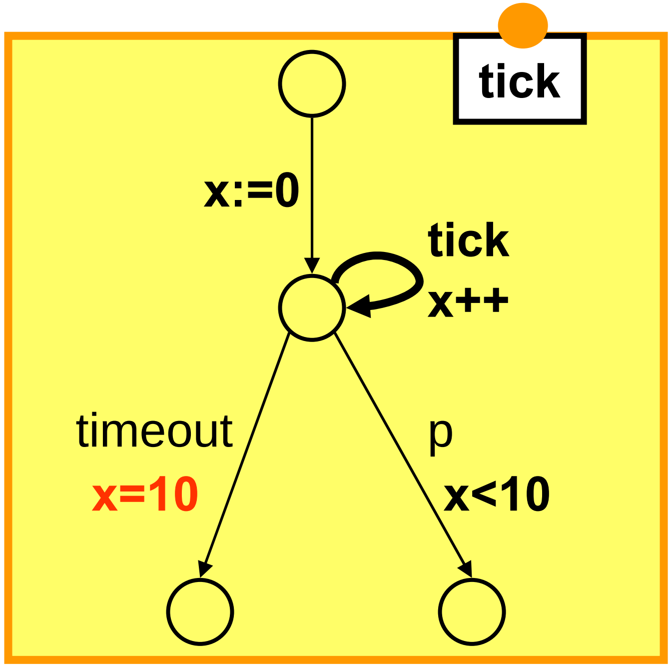
The BIP Framework: Model Construction Space

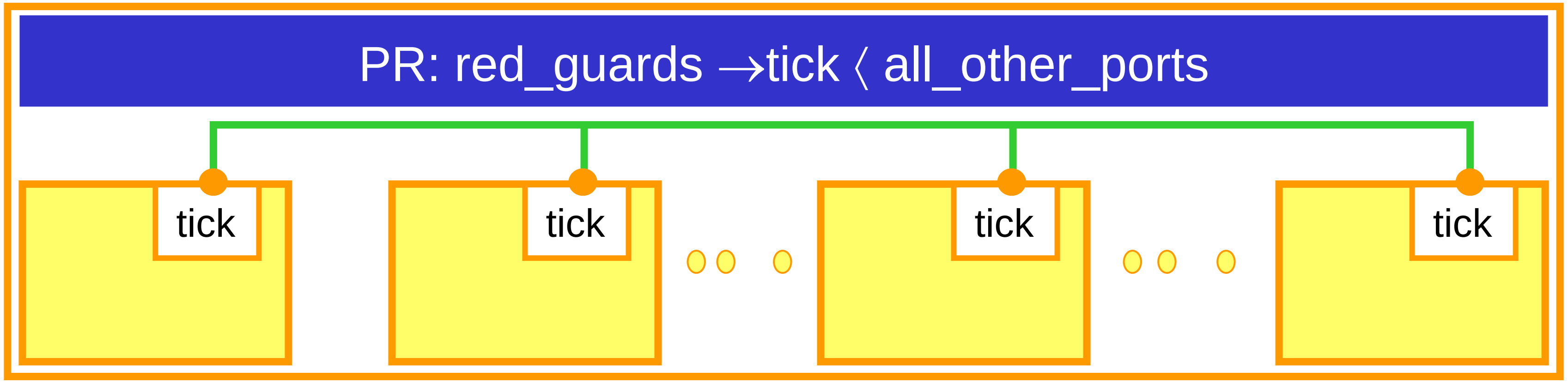


Characterize relations between classes by elementary model transformations:

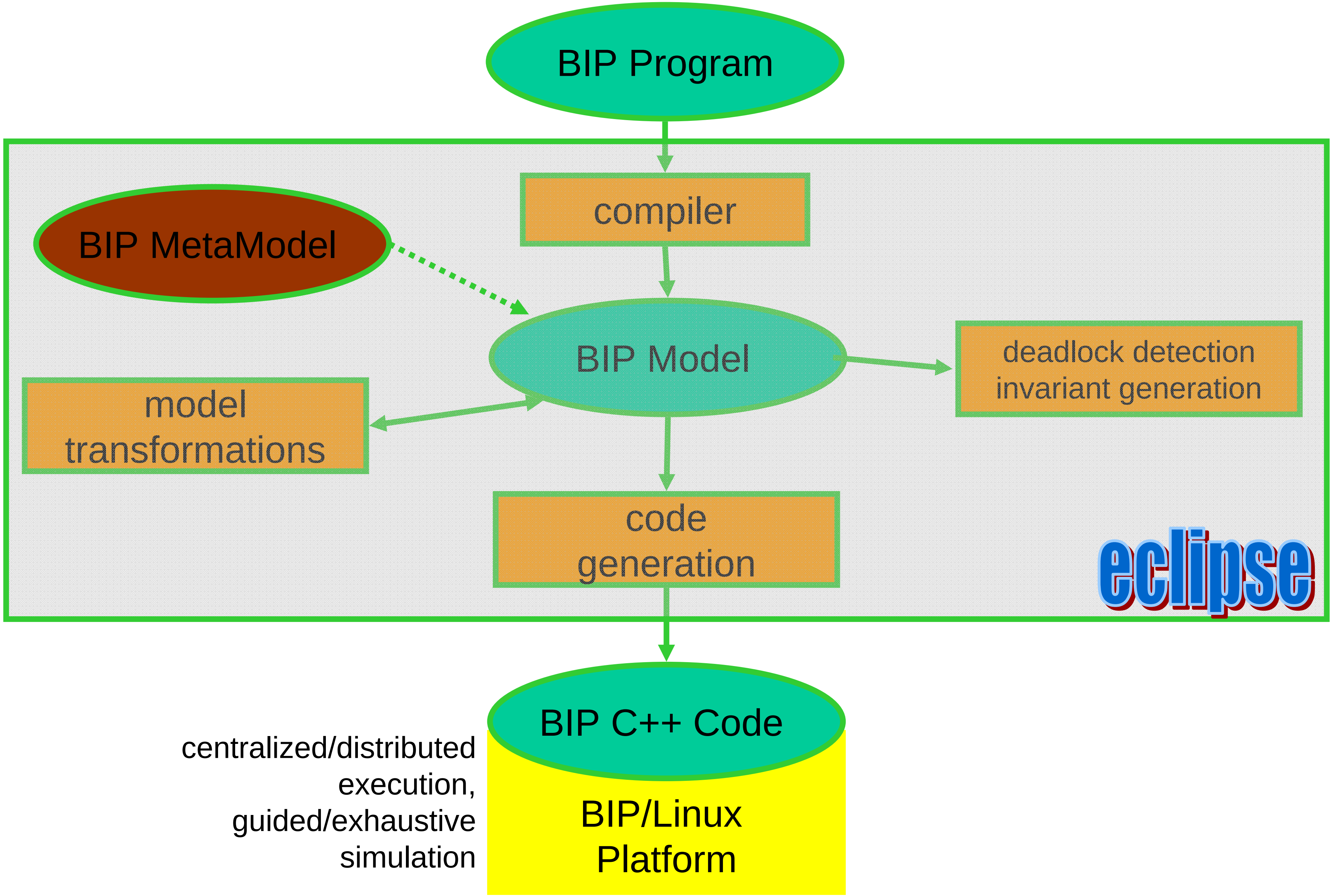
- Untimed-Timed
- Synchronous – Asynchronous
- Event triggered – Data triggered

The BIP Framework: Timed vs. Untimed

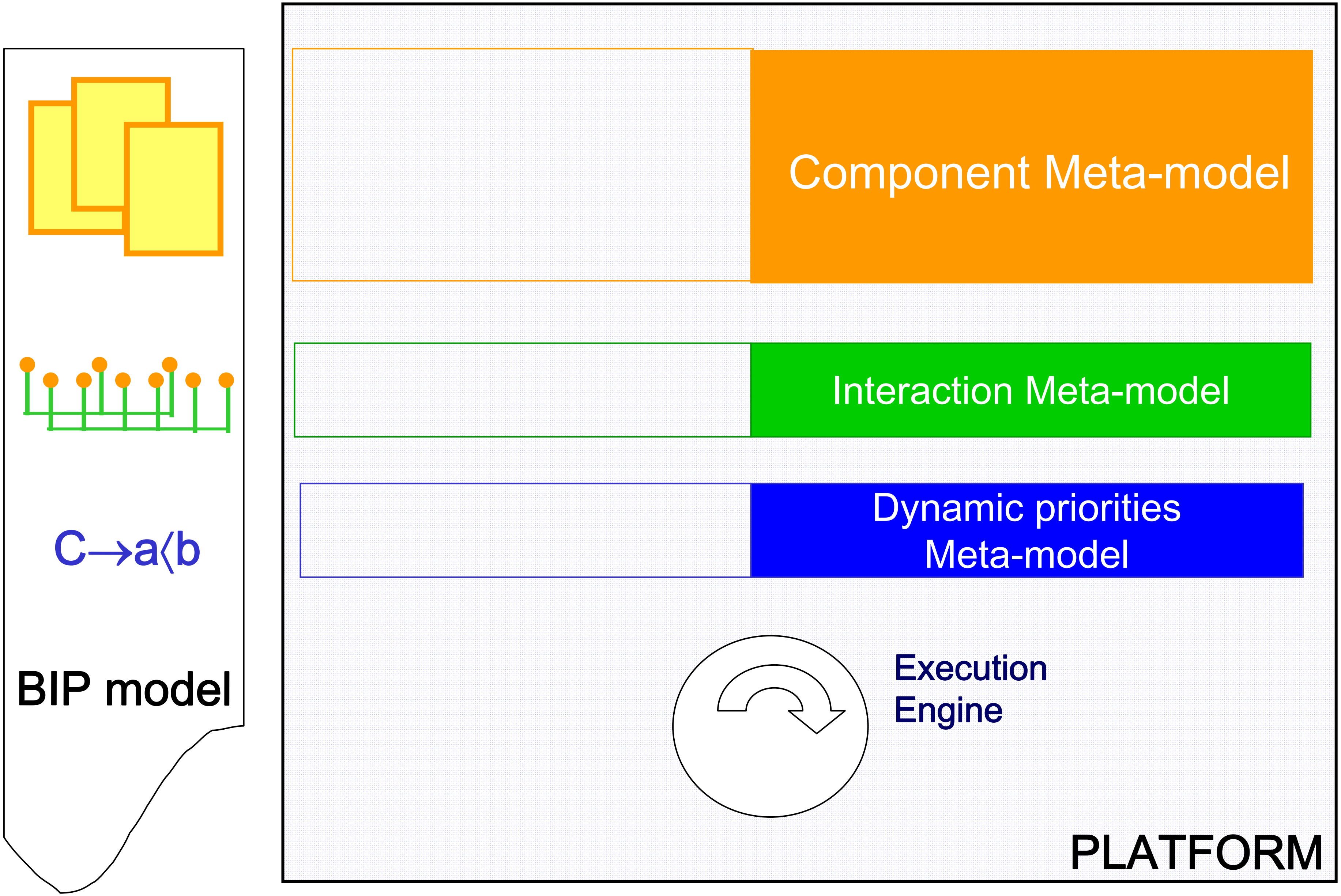




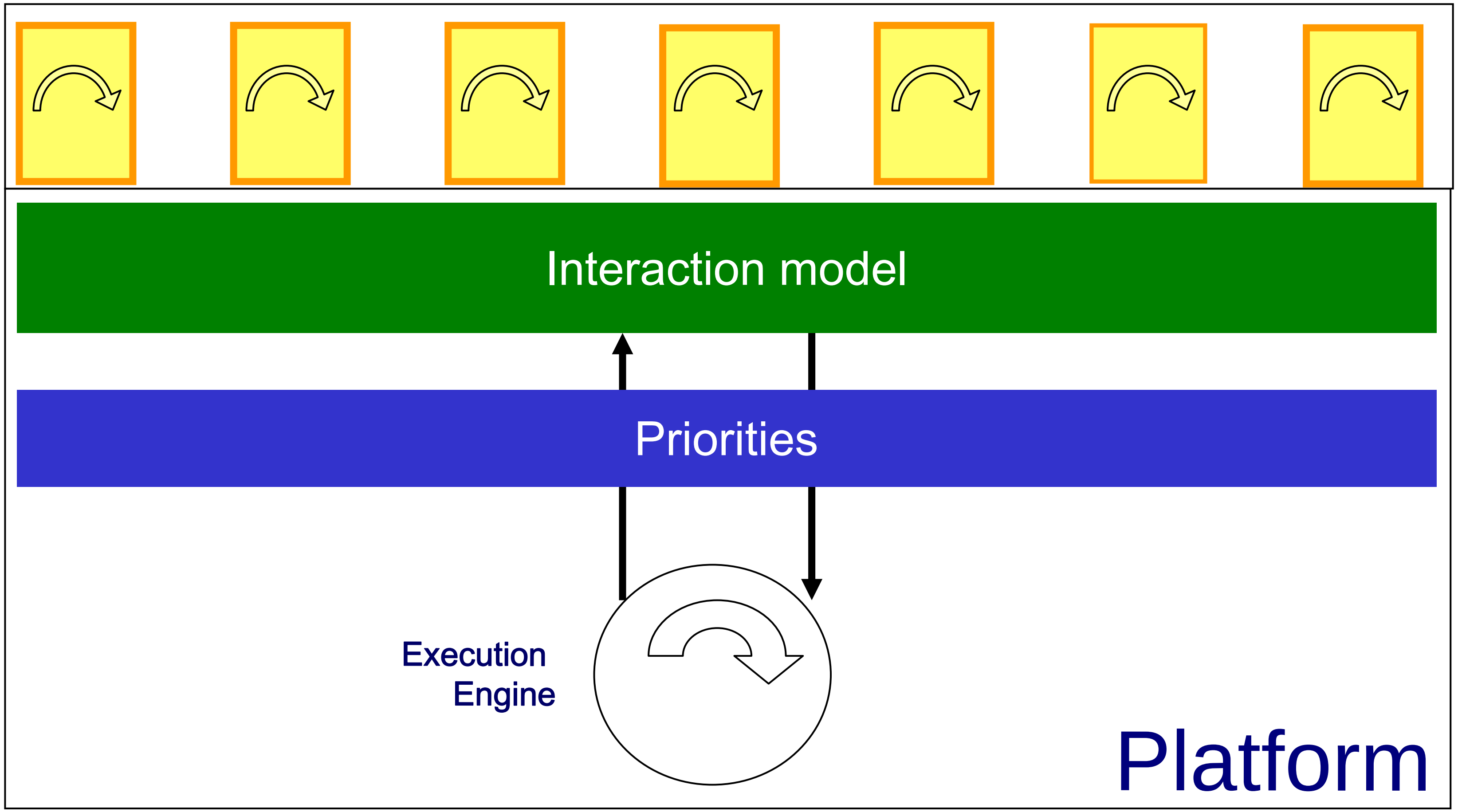
Implementation: Overall architecture



Implementation: generation of C++ code



Implementation: The Execution Platform



The Language: Atomic Components

```
component C
port  trigger: p1, … ;  synchron: 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 Language: Connectors and Priorities

connector BUS= {p, p', … , }
trigger()
  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 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
```

MPEG4 Video Encoder

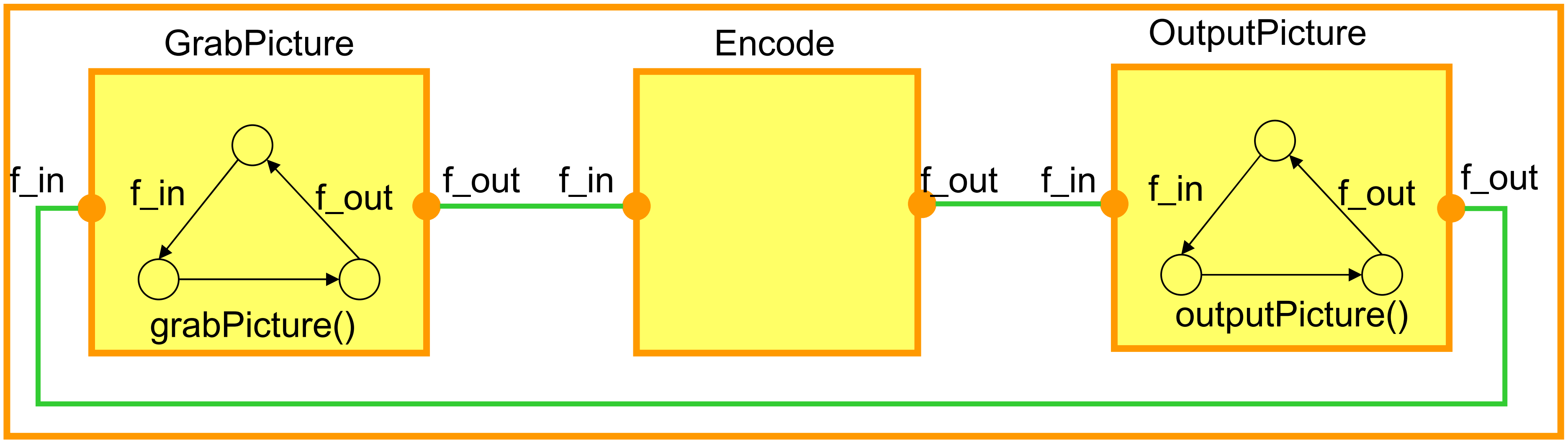
Transform a monolithic program into a componentized one

++ reconfigurability, schedulability

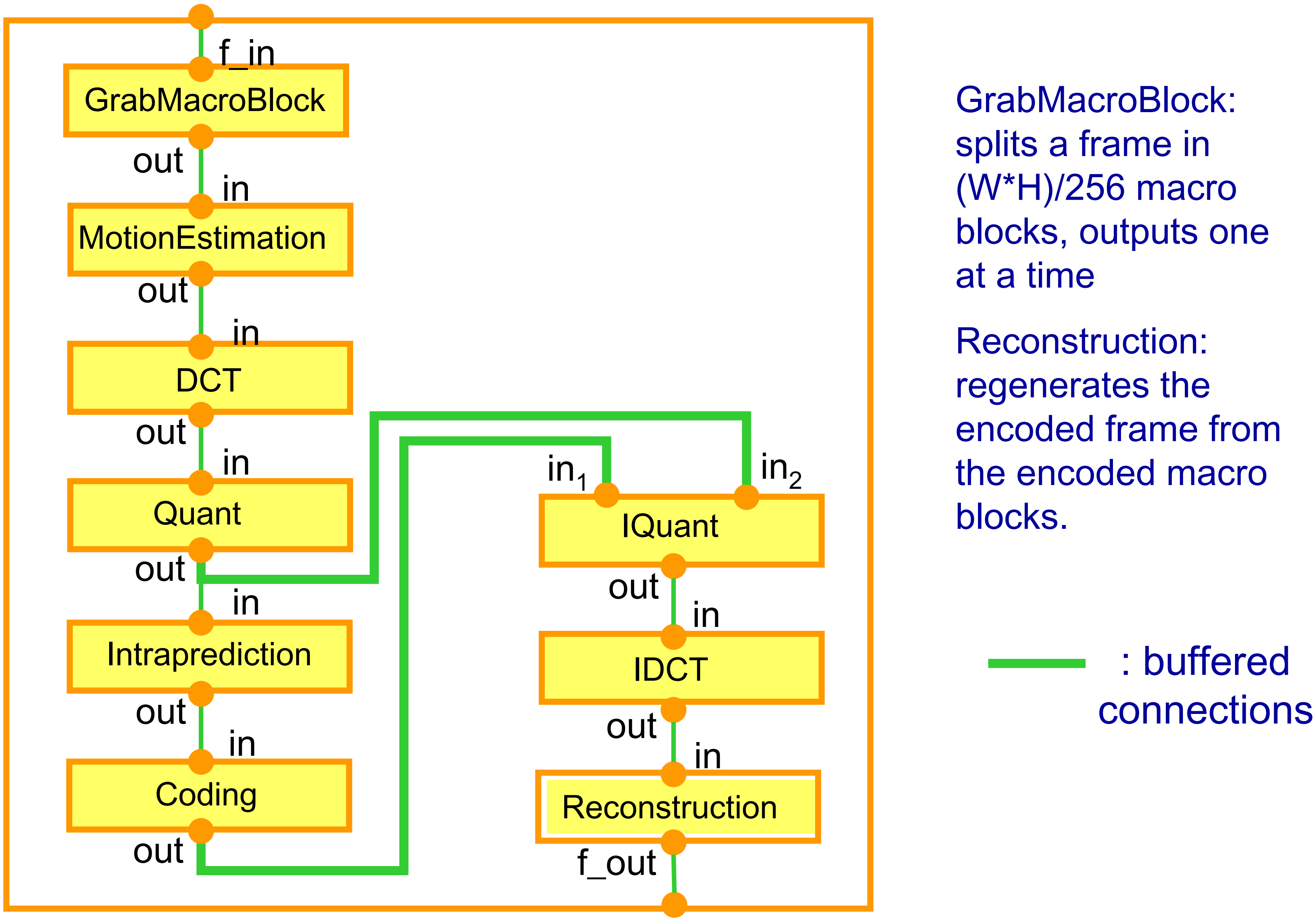
– – overheads (memory, execution time)

Video encoder characteristics:

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



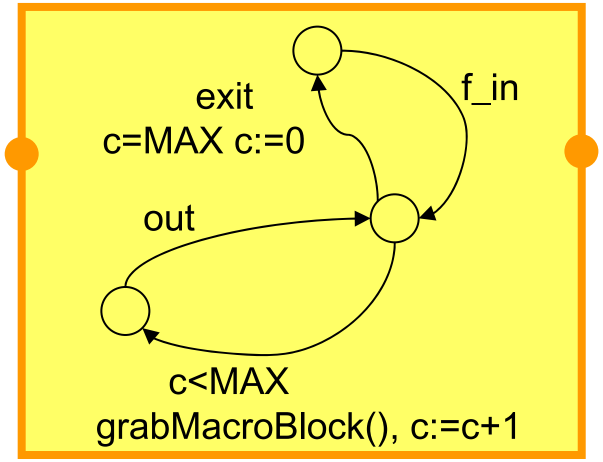
MPEG4 Video Encoder: Architecture

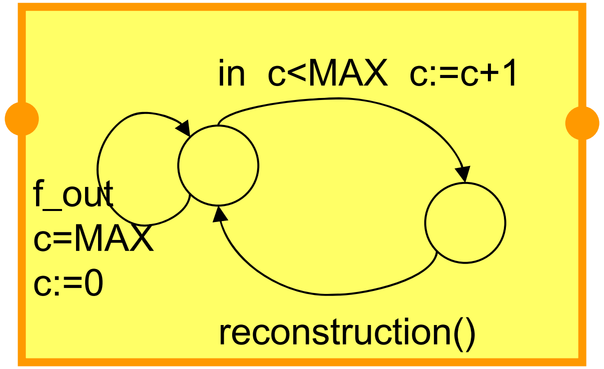


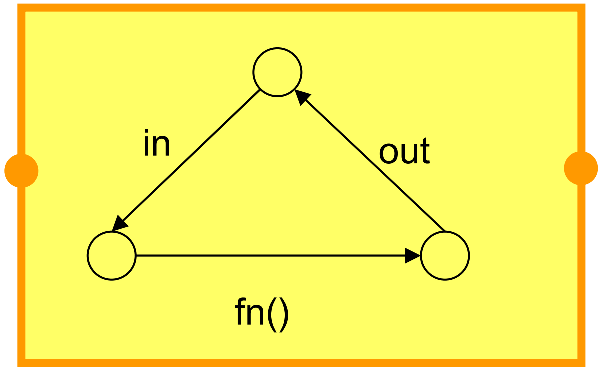
GrabMacroBlock: splits a frame in (W*H)/256 macro blocks, outputs one at a time

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

MPEG4 Video Encoder: Atomic Components







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

MPEG4 Video Encoder: Experimental results

- 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: Experimental results

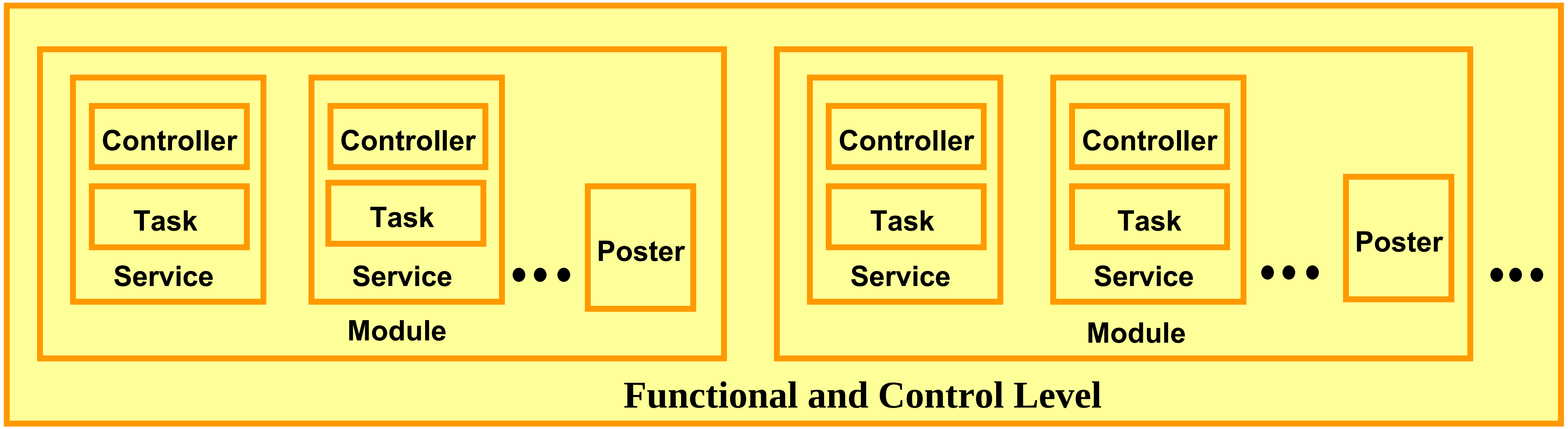
Overhead in execution time with respect to monolithic code:

- ~66% due to communication (can be reduced by composing components at compile time)
  .

- ~34% due to resolution of non determinism (can be reduced by computing restricted guards at compile time)

We know how to reduce execution time overhead
for componentized code

The DALA Robot: Architecture



Functional and Control Level ::= (Module)$^+$

Module ::= (Service)$^+$. (Poster)

Service ::= (Service Controller) . (Service Task)

Service Controller ::= (Event Triggered Controller) | (Cyclic Controller)

Cyclic Controller ::= (Event Triggered Controller) . (Cyclic Trigger)

Service Task ::= (Timed Task) | (Untimed Task)

The DALA Robot: Event Triggered Controller

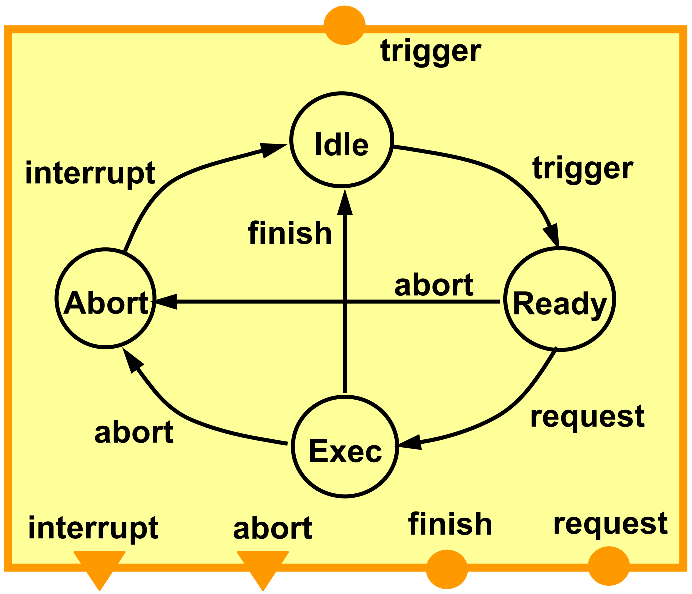
Triggered by events provided by the Decision Level

Idle: the Service is idle

Ready: checks the possibility for starting a new Task of the Service

Exec: exécution of the Task of the Service

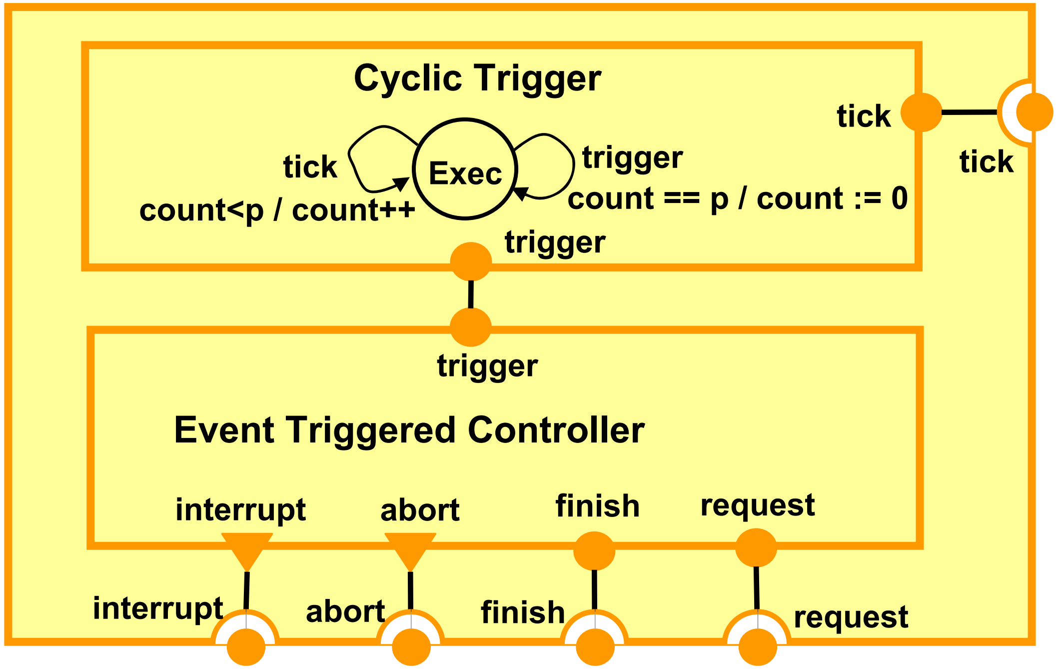
Abort: Service is aborted



The DALA Robot: Event Triggered Controller

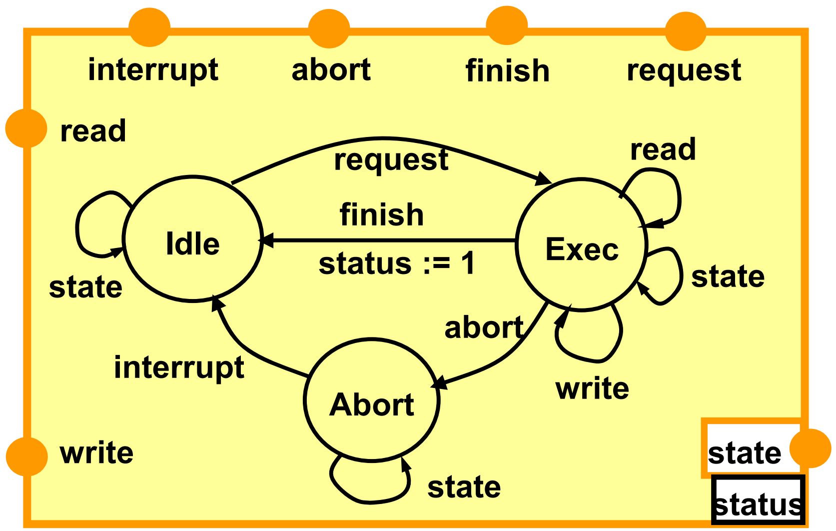
Cyclic Controller ::=
(Event Triggered Controller) . (Cyclic Trigger)

The Cyclic Trigger starts the Event Triggered Controller every period p



The DALA Robot: Untimed Task

Triggered by *request*



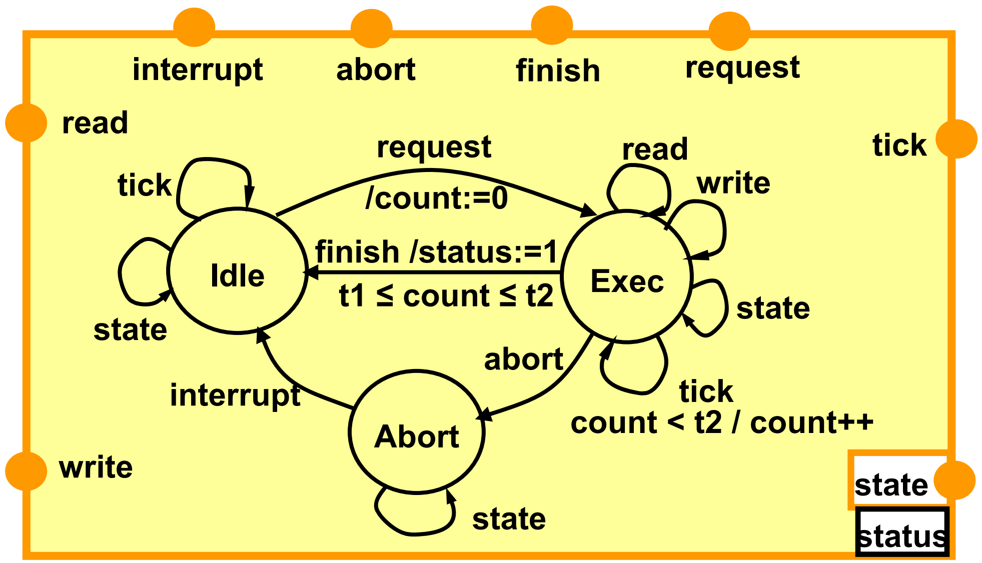
The variable ***status*** specifies the previous state of Task

status == 1 : Task successfully executed

status == 0 : Task aborted

The DALA Robot: Timed Task

- Obtained from an Untimed Task
- Its execution time is in [t1,t2]

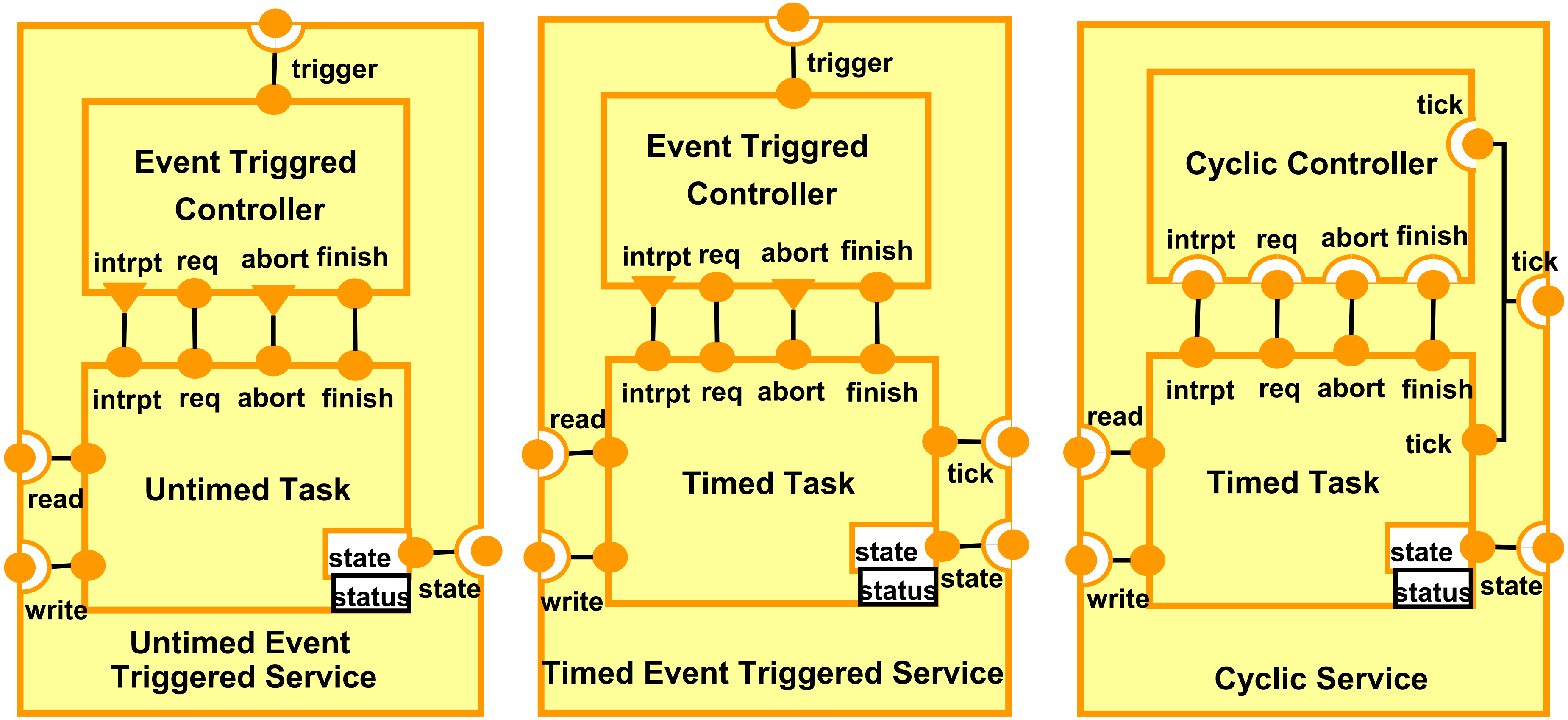


The DALA Robot: Different types of Services

Untimed Event Triggered Service ::= Event Triggered Controller. Untimed Task

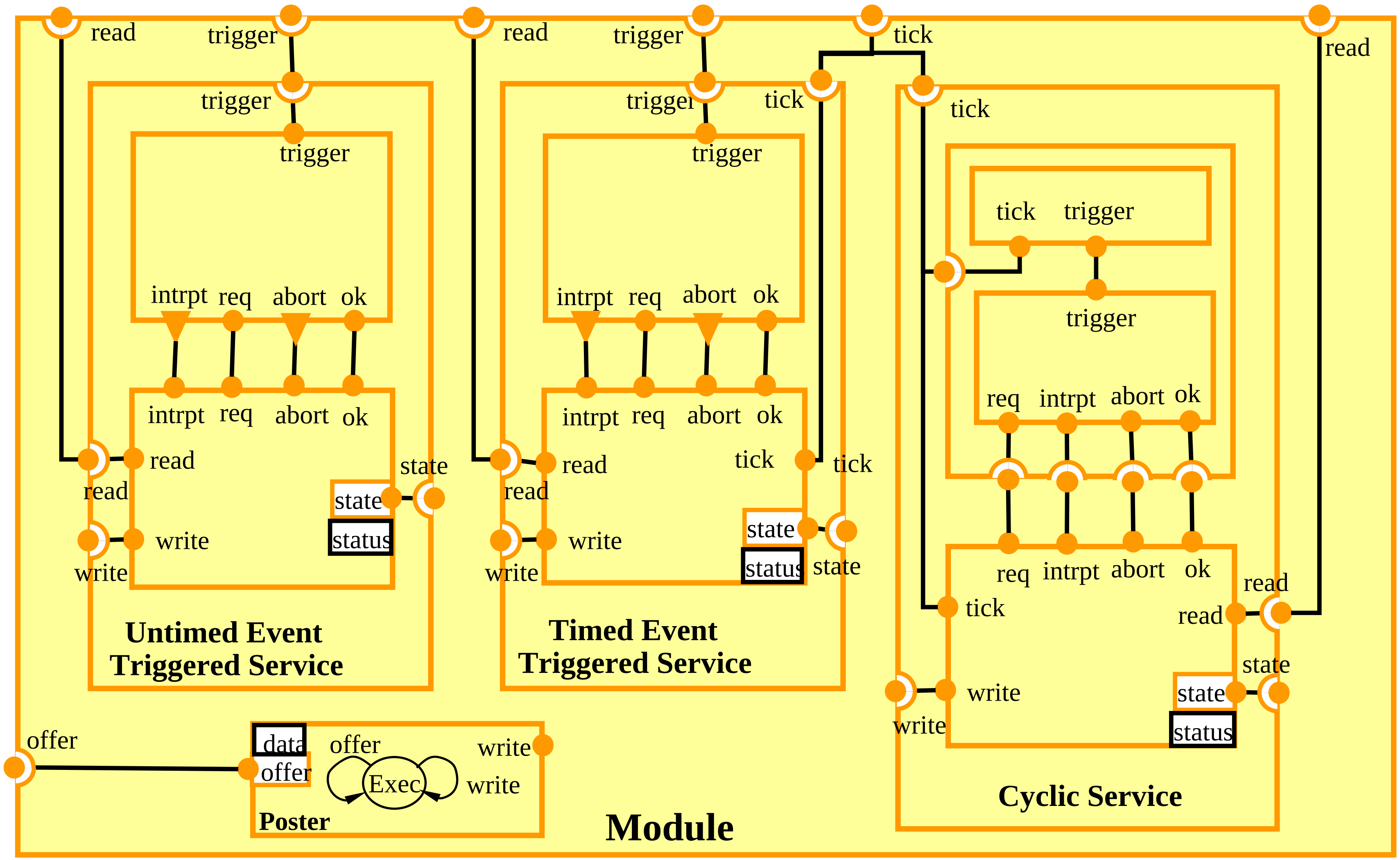
Timed Event Triggered Service ::= Event Triggered Controller. Timed Task

Cyclic Service ::= Cyclic Controller . Timed Task



The DALA Robot: A Module

A module composed of 3 services and a poster



OVERVIEW

- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

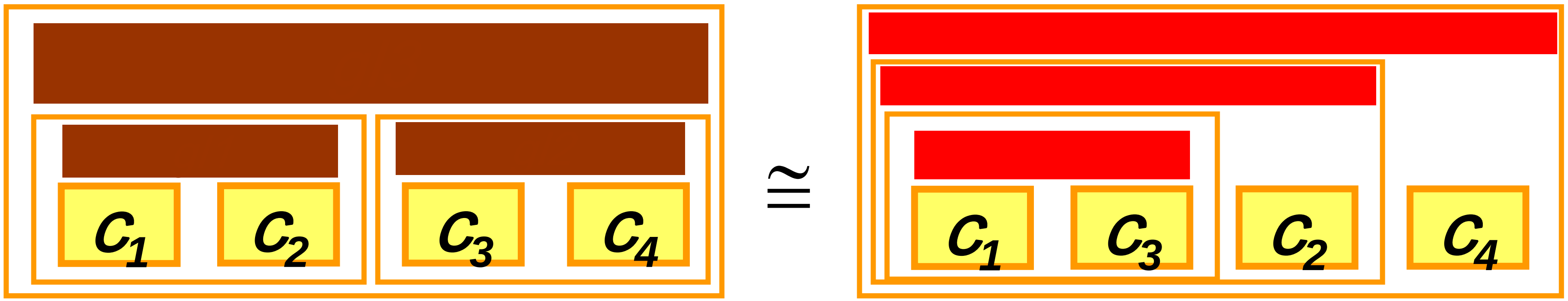
Expressiveness for component-based systems

Given two glues G_1 , G_2

G_2 ***is strongly more expressive than*** G_1

if for any component built by using G_1 and $\mathcal{C}_0$

there exists an equivalent component built by using G_2 and $\mathcal{C}_0$



Expressiveness for component-based systems

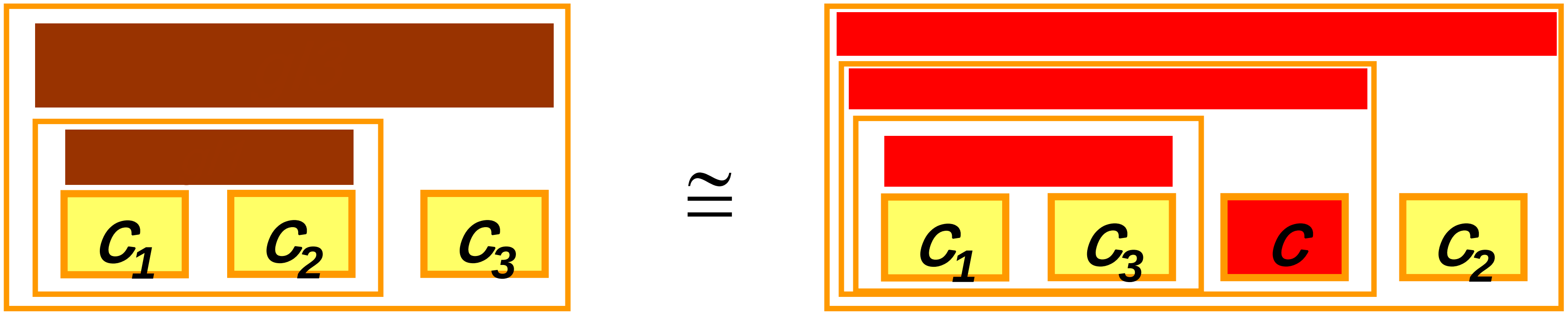
Given two glues G_1 , G_2

G_2 ***is weakly more expressive than*** G_1

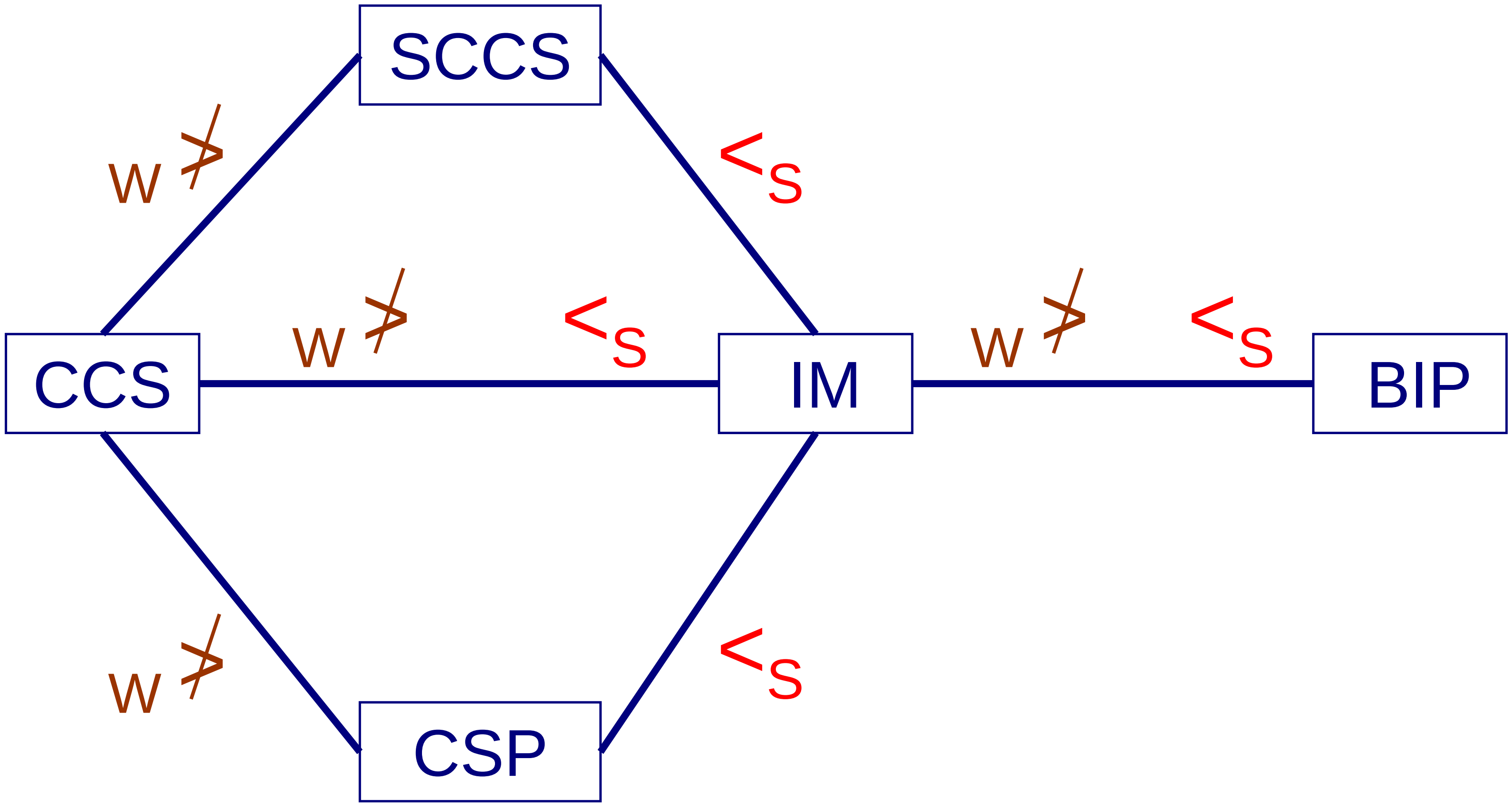
if for any component built by using G_1 and $\mathcal{C}_0$

there exists an equivalent component built by using G_2 and $\mathcal{C}_0 \cup \mathcal{C}$

where $\mathcal{C}$ is a finite set of coordination behaviors.



Expressiveness for component-based systems



[Bliudze&Sifakis, Concur 08]

OVERVIEW

- Component-based Construction
- BIP: Basic Concepts
- Modeling Interactions
- Modeling Priorities
- The BIP framework
- Expressiveness
- Discussion

Discussion

Clear separation between behavior and architecture

- Architecture = interaction + priority
- Minimal set of constructs and principles
- Correctness-by-construction techniques for deadlock-freedom and liveness, based on sufficient conditions on architecture

Expressiveness results

- BIP is as expressive as the most general glue
- Separation between interaction and priority for enhanced analysis and system construction methodology

Applications

- Software componentization
- Modeling mixed HW / SW systems e.g. sensor networks