Component-based Construction of Real-Time Systems

Joseph Sifakis
VERIMAG

ARTIST2 MOTIVES School
Trento, February 19-23, 2007

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. data 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
  - 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

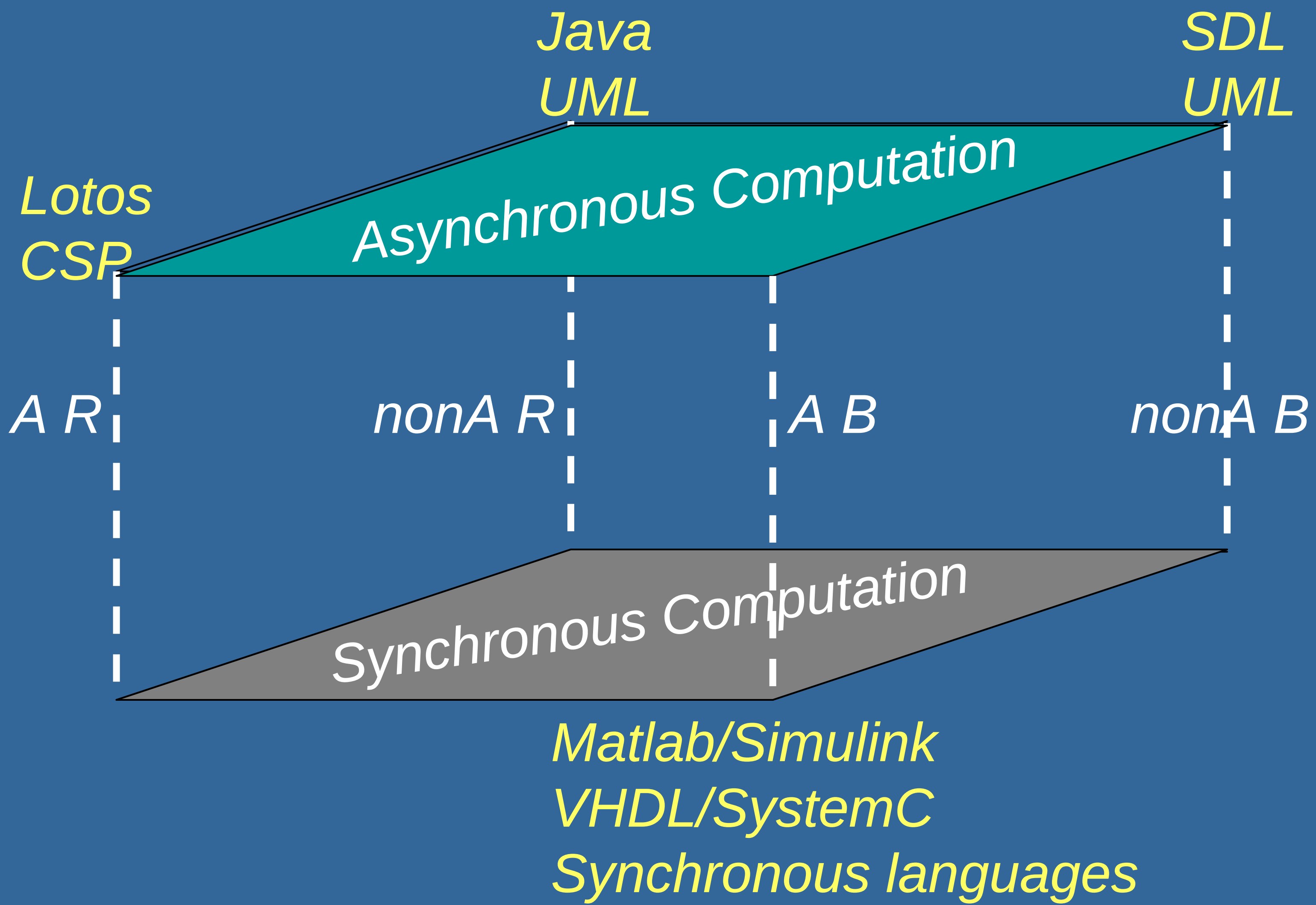
Heterogeneity of execution

- 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*

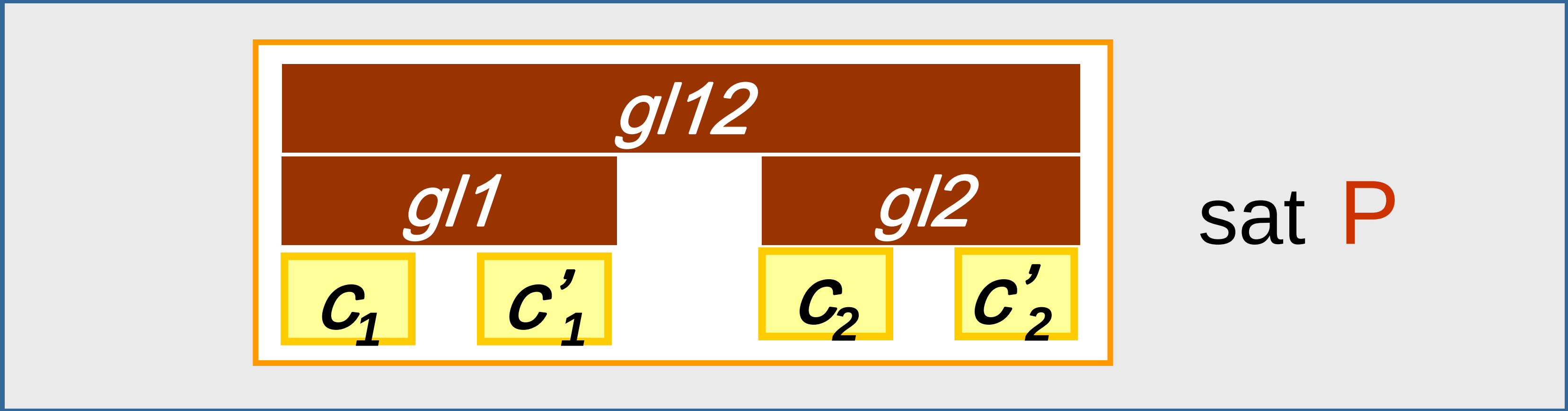


Overview

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

Component-based construction – Formal framework

Build a component *C* satisfying a given property *P,* from
- $\mathcal{C}_0$ a set of atomic components modeling behavior
- $\mathcal{GL} = \{gl_1, \ldots, gl_i, \ldots\}$ a set of glue operators on components



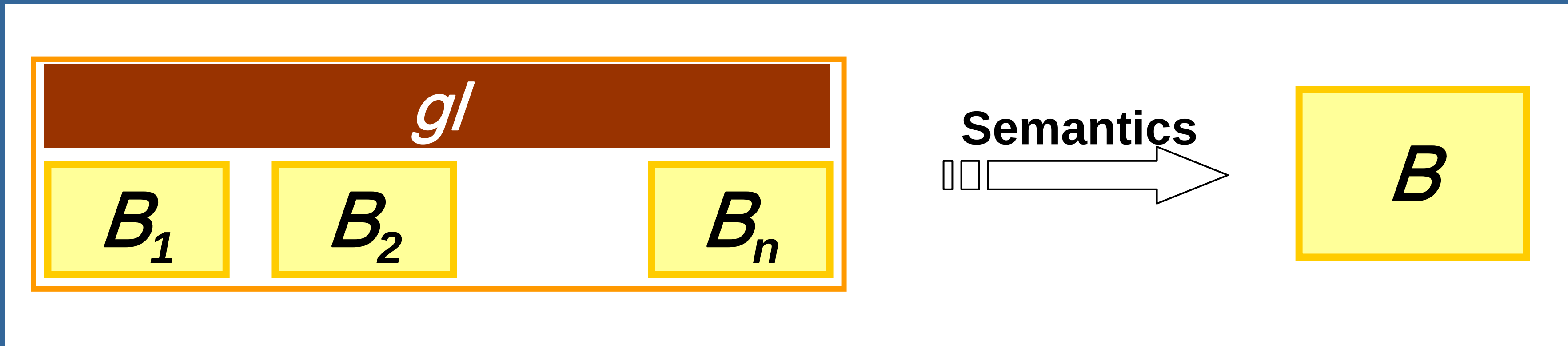
Glue operators
- model mechanisms used for communication and control such as protocols, controllers, buses.
- restrict the behavior of their arguments, that is

$$gl(C_1, C_2, .., C_n) | A_1 \text{ refines } C_1$$

Component-based construction – Formal framework

Semantics:

- Atomic components $\rightarrow$ behavior
- Glue operators transform sets of components into components



The process algebra paradigm

- 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 semantics

Component-based construction - Requirements

Find sets of glue operators meeting the following requirements:

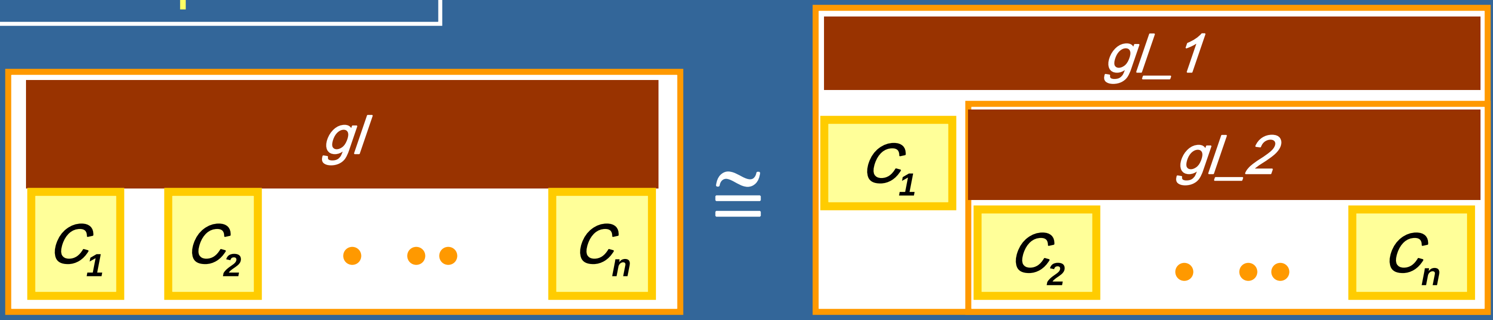
1. Incremental description

2. Correctness-by-construction

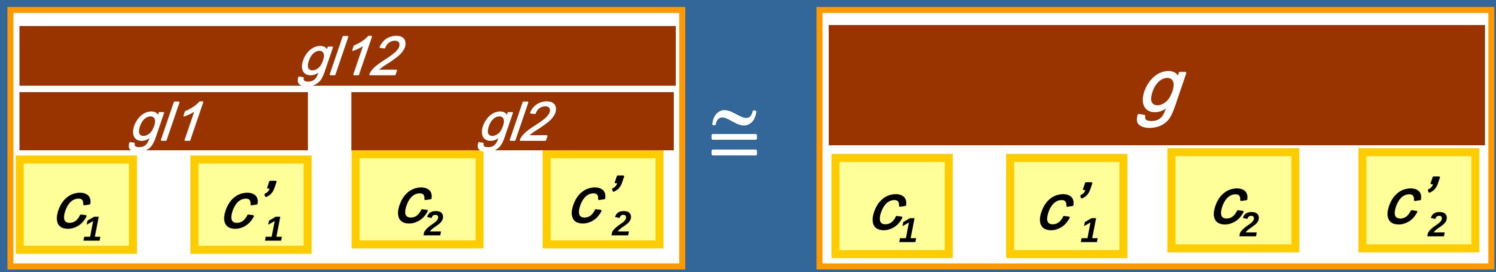
3. Expressiveness (discussed later)

Component-based construction – Incremental description

1. Decomposition



2. Flattening



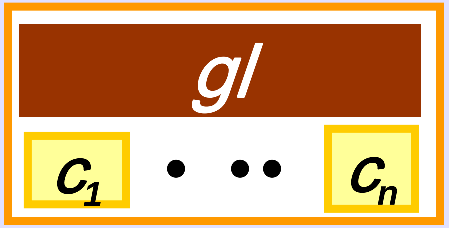
Flattening can be achieved by using a (partial) associative operation ⊕ on GL

Component-based construction –
Correctness by construction : Compositionality

Building correct systems

from correct components

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

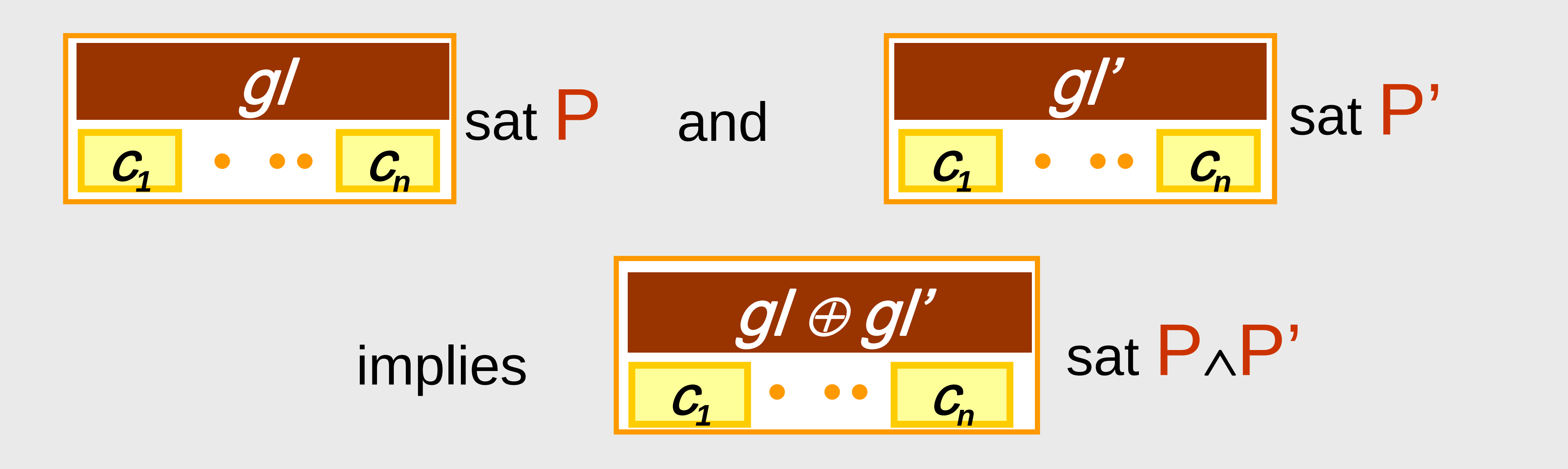


sat $\widetilde{gl}(P_1, .., P_n)$

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

Component-based construction – Correctness by construction : Composability

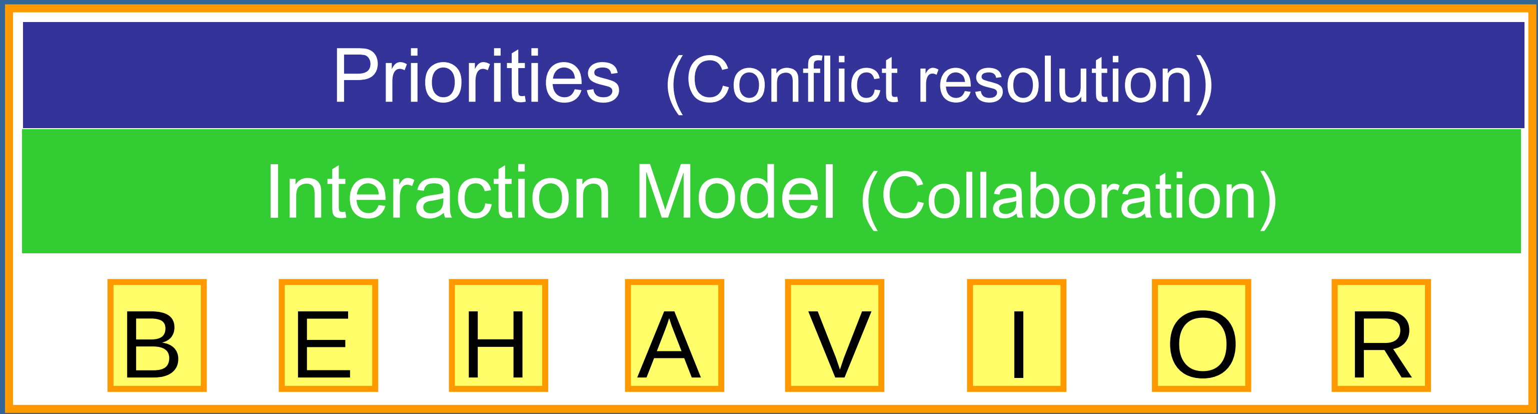
Integrated components preserve essential properties



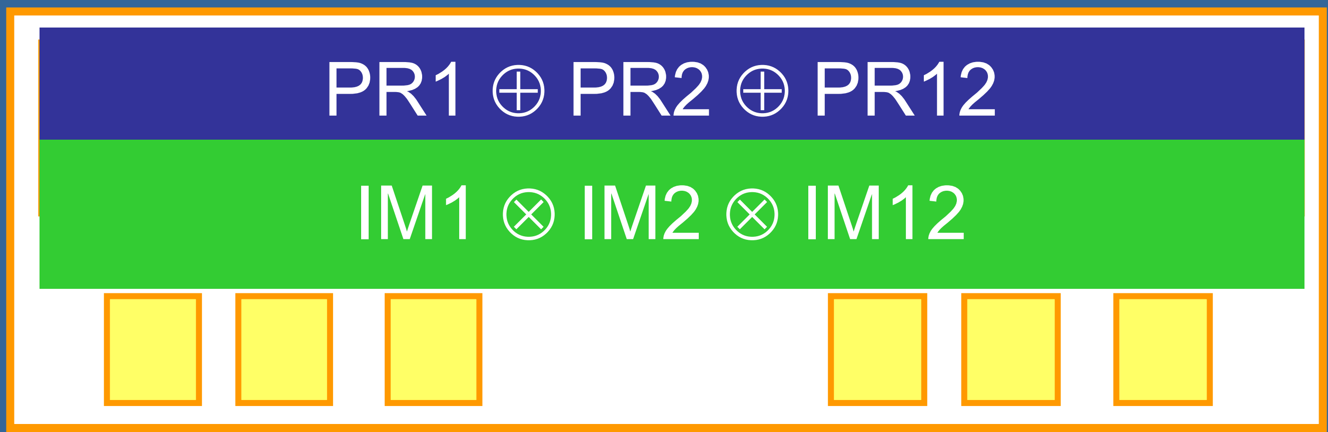
Composability means non interference of properties of integrated components. Lack of results for guaranteeing property stability e.g.

- *non composability of scheduling algorithms*
- *feature interaction*

Layered component model



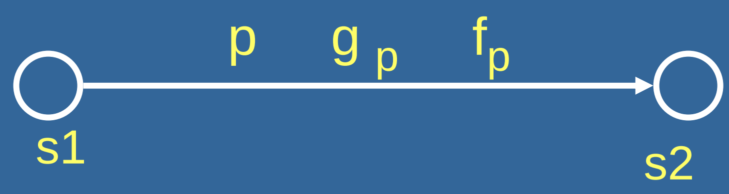
Composition (incremental description)

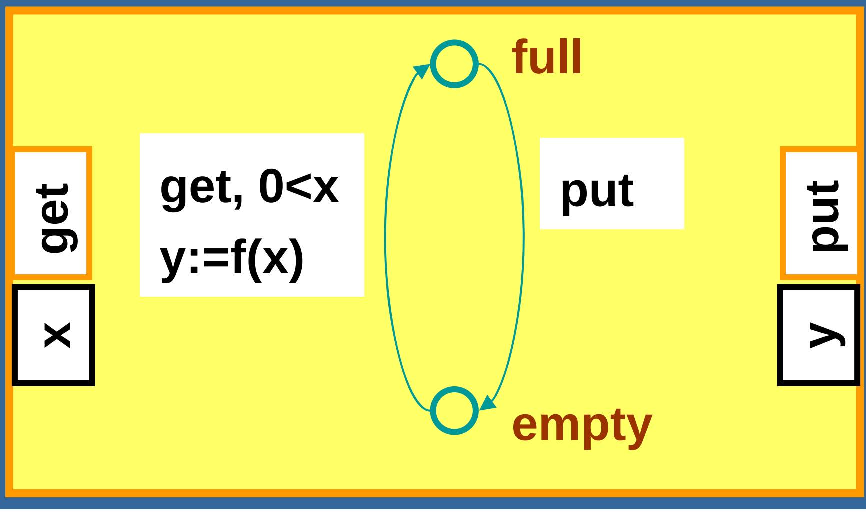


Component-based construction –
The BIP framework: Behavior

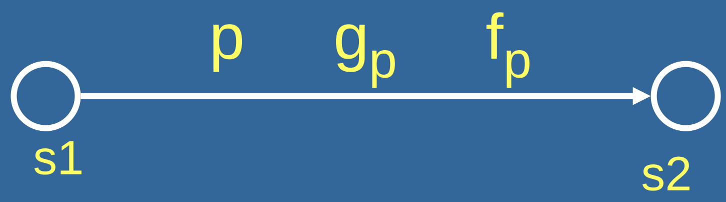
An atomic component has

- A set of ports P, for interaction with other components
- A set of control states 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)





Component-based construction – The BIP framework: Behavior



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

- **Enabledness:** g_p is true and some interaction involving p is possible
- **Execution:** interaction involving p followed by the execution of f_p

Overview

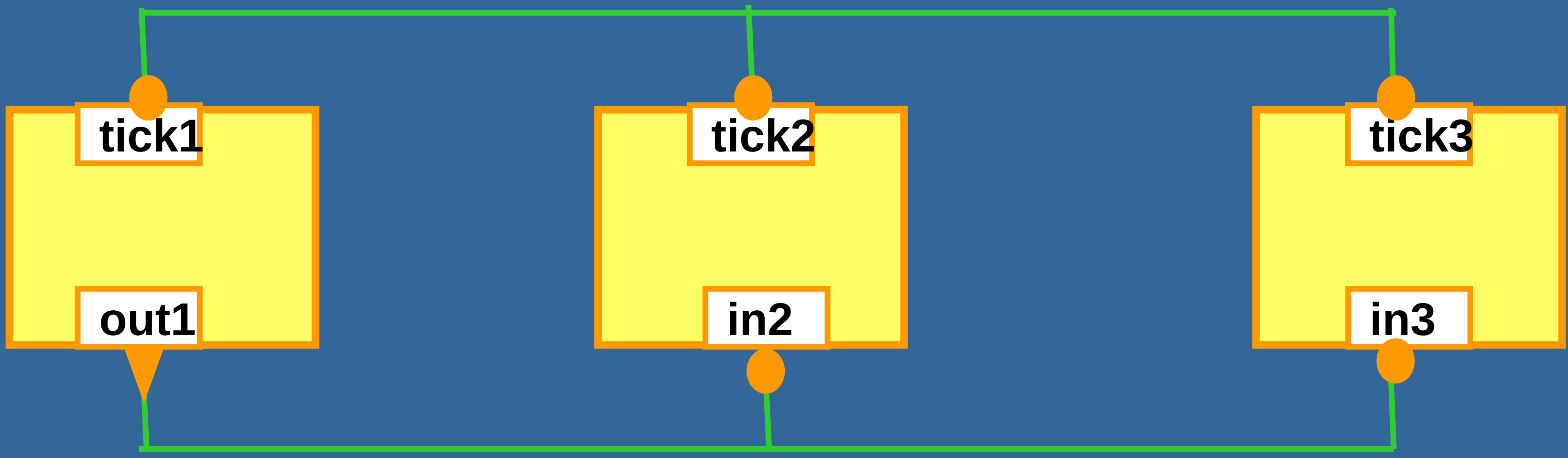
- About component-based construction
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

Interaction modeling

• A ***connector*** is a set of ports which can be involved in an interaction

• Port attributes **(*complete*, *incomplete*)** are used to distinguish between rendezvous and broadcast.

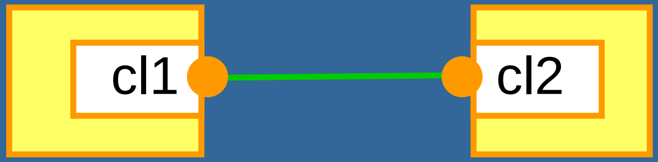
• An ***interaction*** of a connector is a set of ports such that: either it contains some complete port or it is maximal.



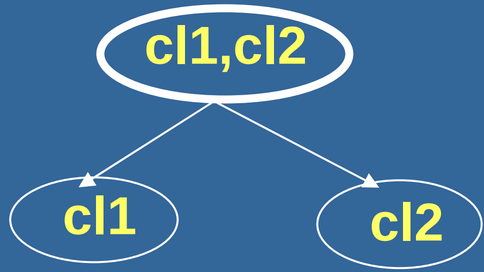
Interactions:

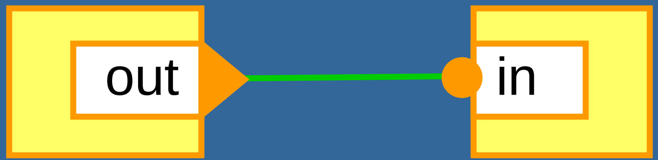
{tick1,tick2,tick3} {out1} {out1,in2} {out1,in3} {out1,in2, in3}

Interaction modeling - Examples

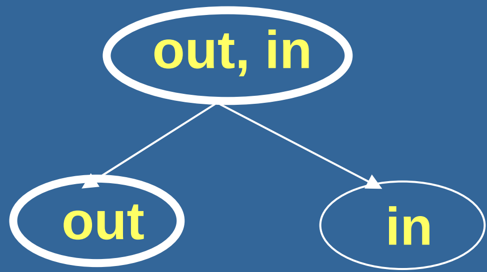


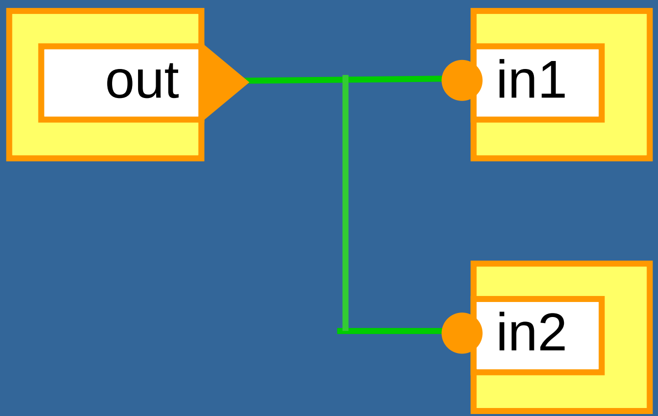
CN:{cl1,cl2}
CP: ∅



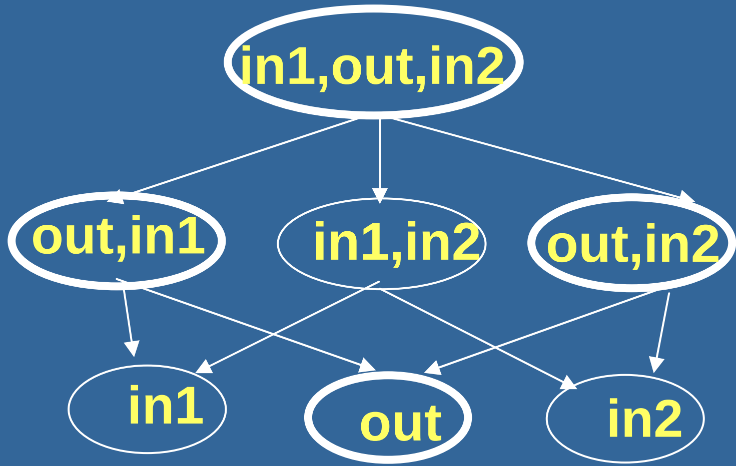


CN:{out,in}
CP: {out}

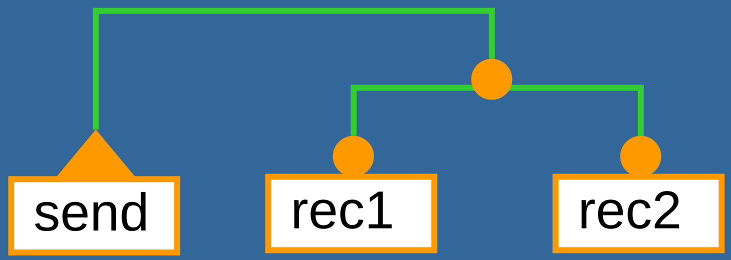




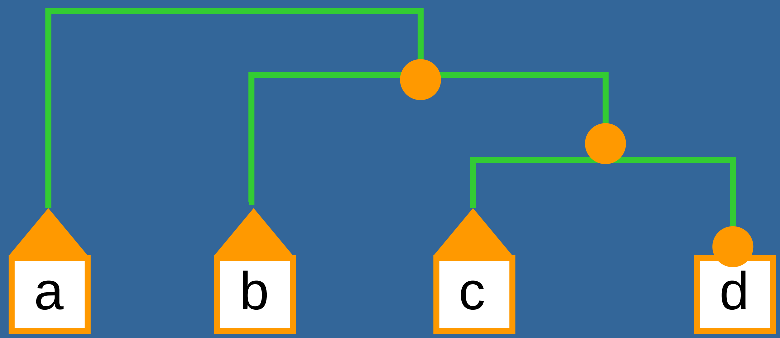
CN:{in1,out,in2
}
CP: {out}



Interaction modeling – Hierarchical connectors

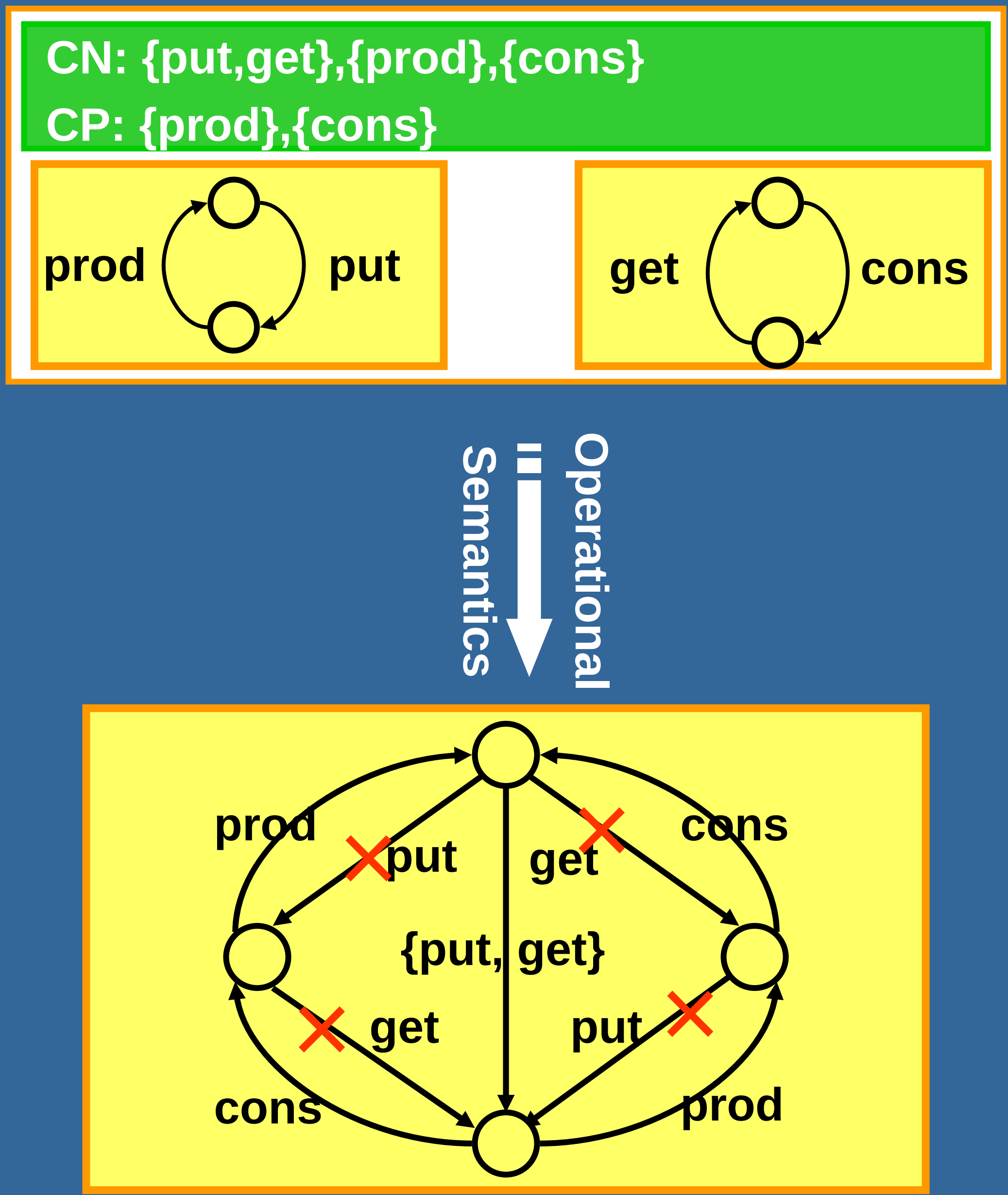


Atomic Broadcast:
send + send rec1 rec2

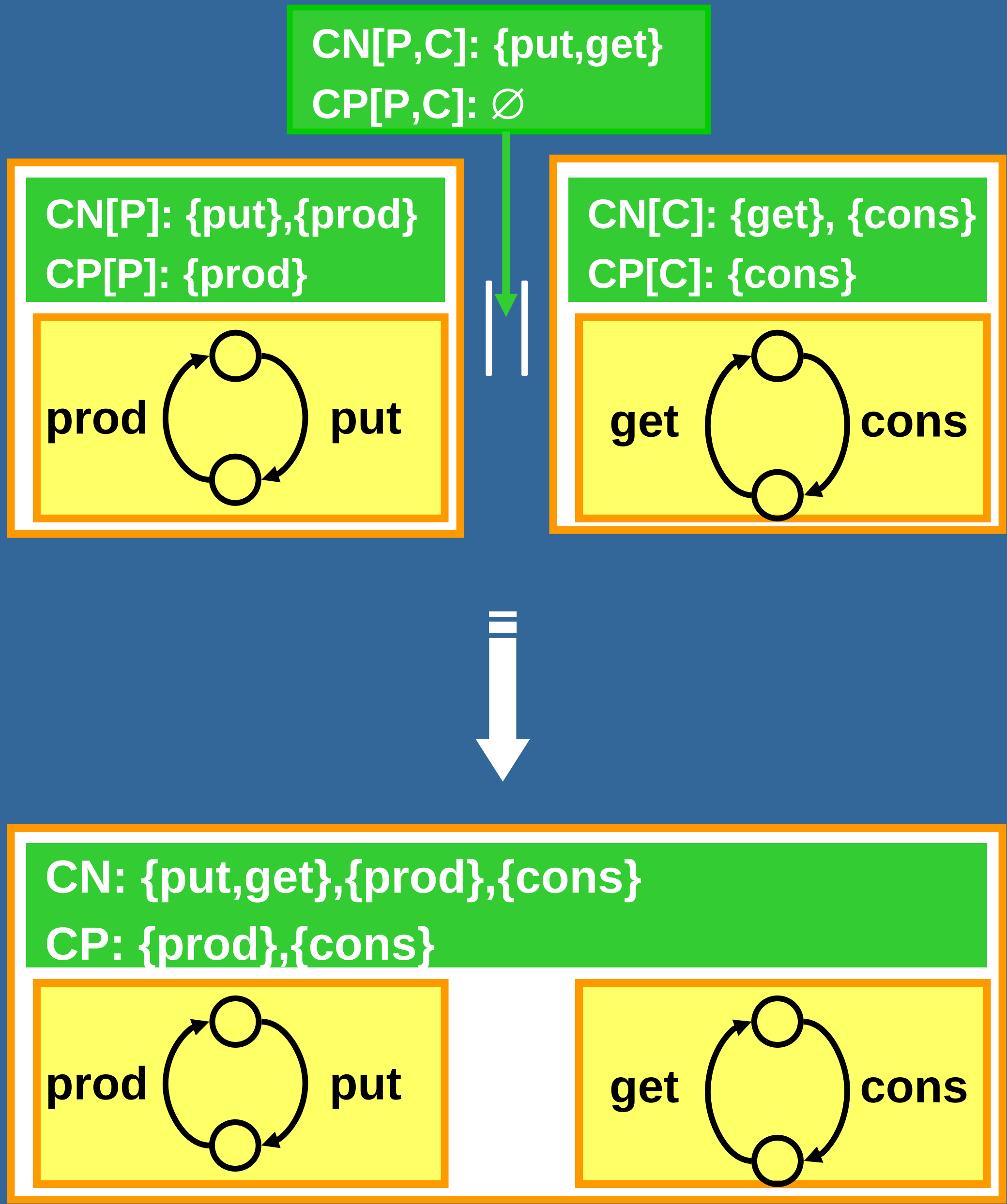


Causal chain:
a+ab+abc+abcd

Interaction modeling – Operational semantics

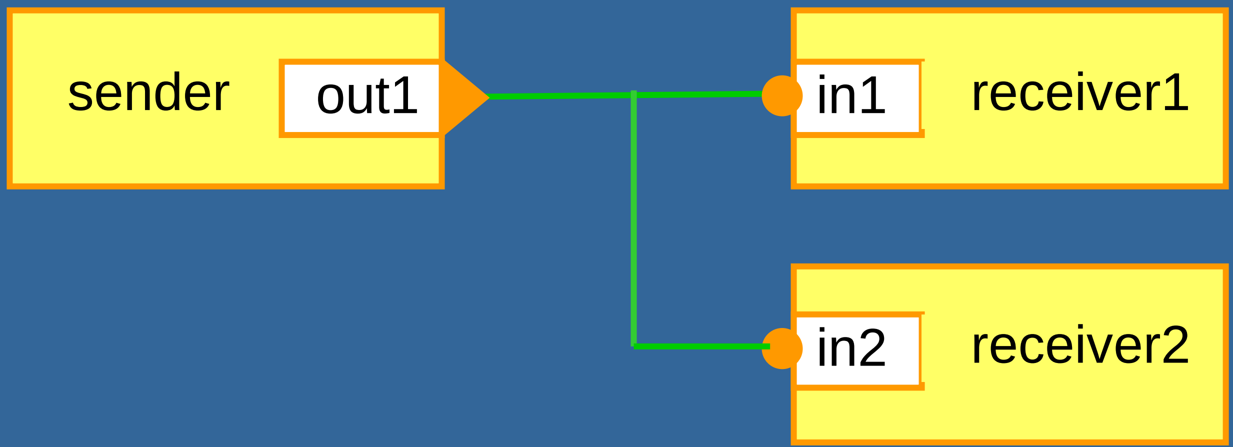


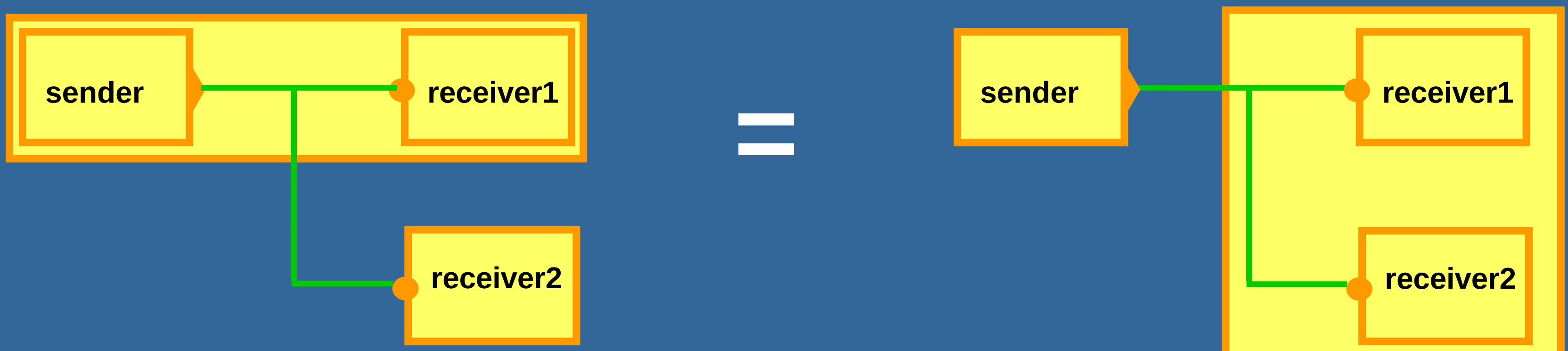
Interaction modeling – Composition



Interaction modeling – Composition: Results [Goessler&SifakisSCP2005]

Incremental commutative composition directly encompassing rendezvous and broadcast





Interaction modeling – Data transfer

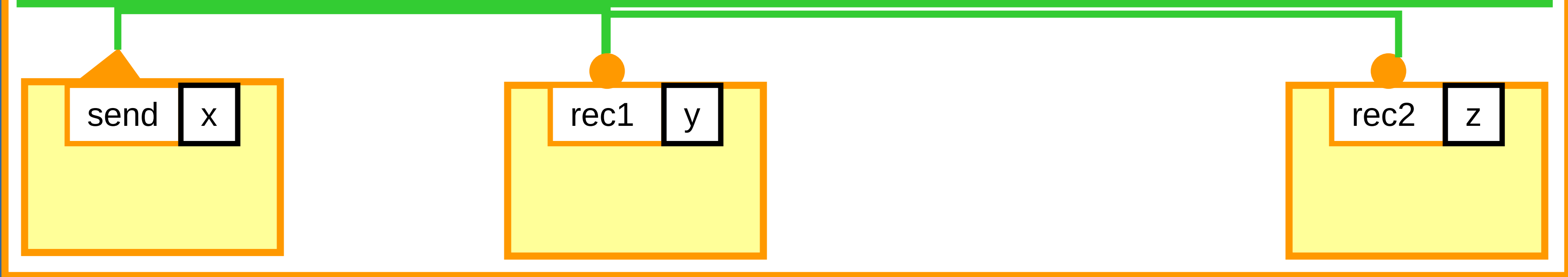
CN: BUS={send,rec1,rec2}

$\{send\}$: $true \rightarrow skip$

$\{send, rec1\}$: $x<y \rightarrow x:=y-x,\ y:=y+x$

$\{send, rec2\}$: $x<z \rightarrow x:=z-x,\ z:=z+x$

$\{send, rec1, rec2\}$: $x<z+y \rightarrow x:=y+z-x,\ y:=y+x,\ z:=z+x$



- Notice the difference between control flow and data flow (input, output)
- Maximal progress: execute a maximal enabled interaction

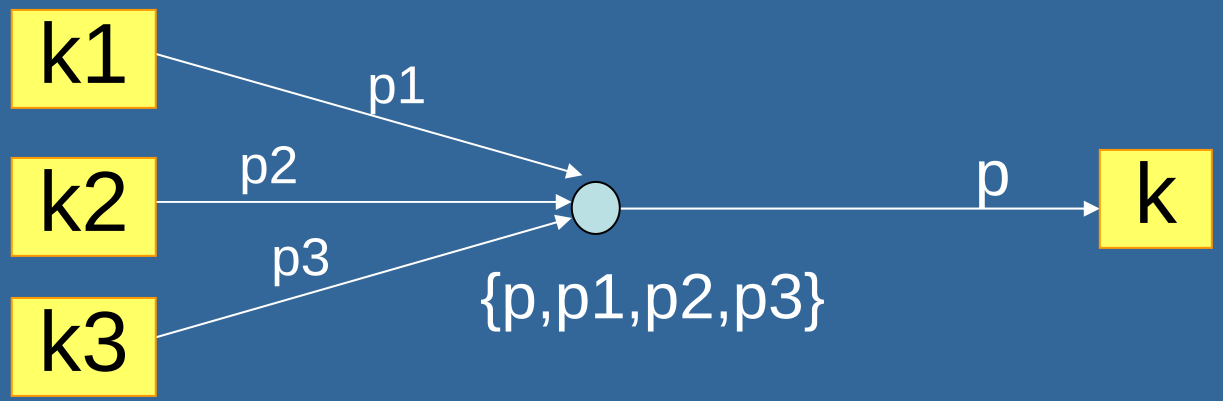
Interaction modeling – Checking for deadlock-freedom[Goessler&Sifakis FSTTCS2003]

For a given system (set of components + interaction model), its **dependency graph** is a bipartite labeled graph with

Nodes N = Set of components $\cup$ Set of minimal interactions

Edges E

- $(\alpha, p, k) \in E$ if α is an interaction, $p \in \alpha$ is an incomplete port of k
- $(k1, p1, \alpha) \in E$ if $p1 \in \alpha$ is a port of k1



Blocking condition for an incomplete port p:

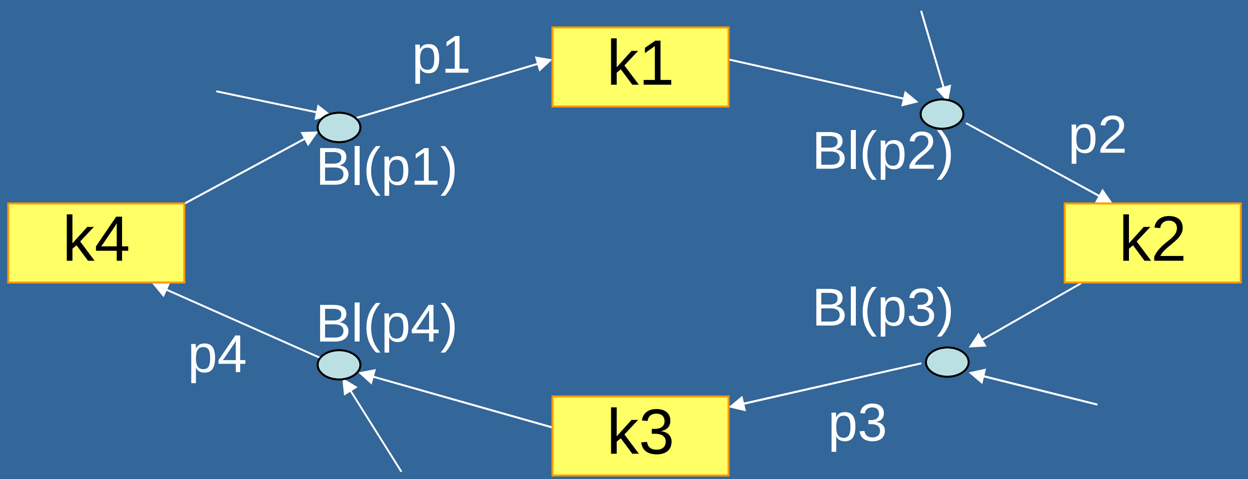
$$Bl(p) = g_p \wedge \neg (g_{p1} \wedge g_{p2} \wedge g_{p3})$$

Interaction modeling – Checking for deadlock-freedom (2)

Possibility of deadlock for the components of circuits ω

such that Bl $(\omega) = \wedge_{p\in\omega}$ Inc$(\omega)\wedge$Bl(p) = *false*

where Inc$(\omega) = \wedge_{k\in\omega}$Inc(k) with Inc(k) the set of the control states of k offering only incomplete ports



Overview

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

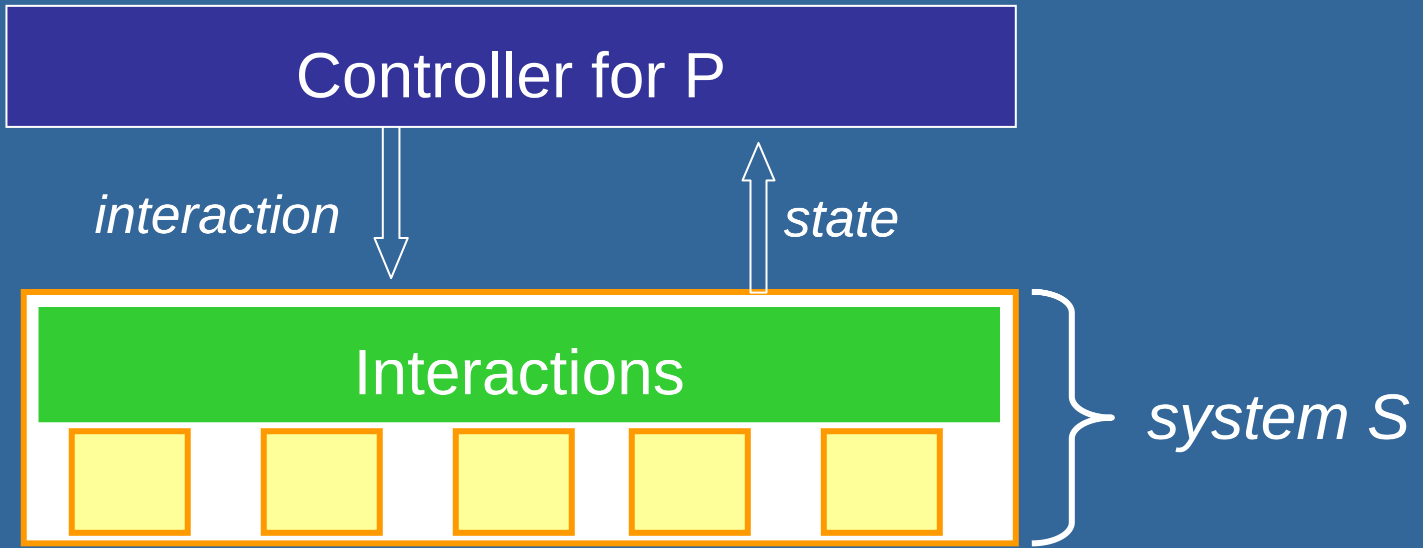
Priorities

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 behavior (non determinism) of system S to enforce a property P



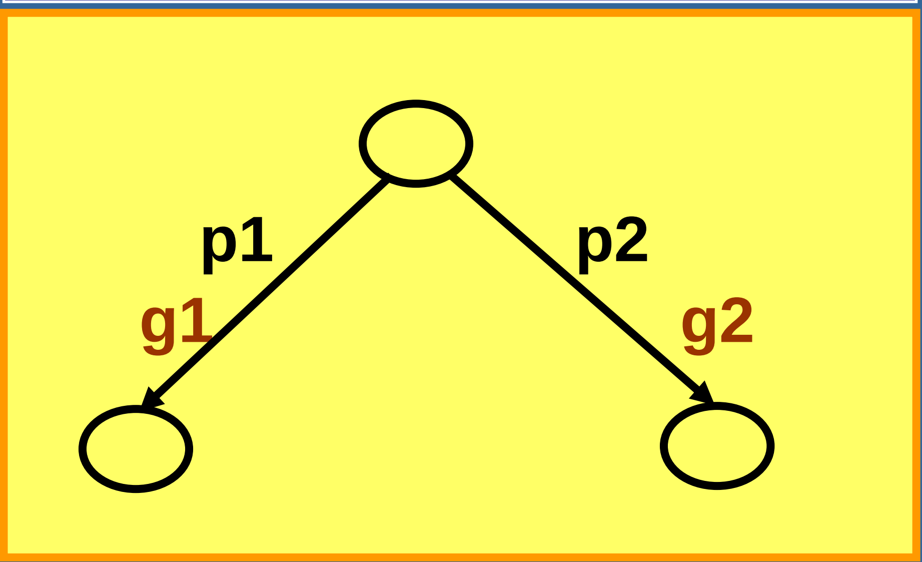
Results [Goessler&Sifakis, FMCO2003] :

• 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 rules



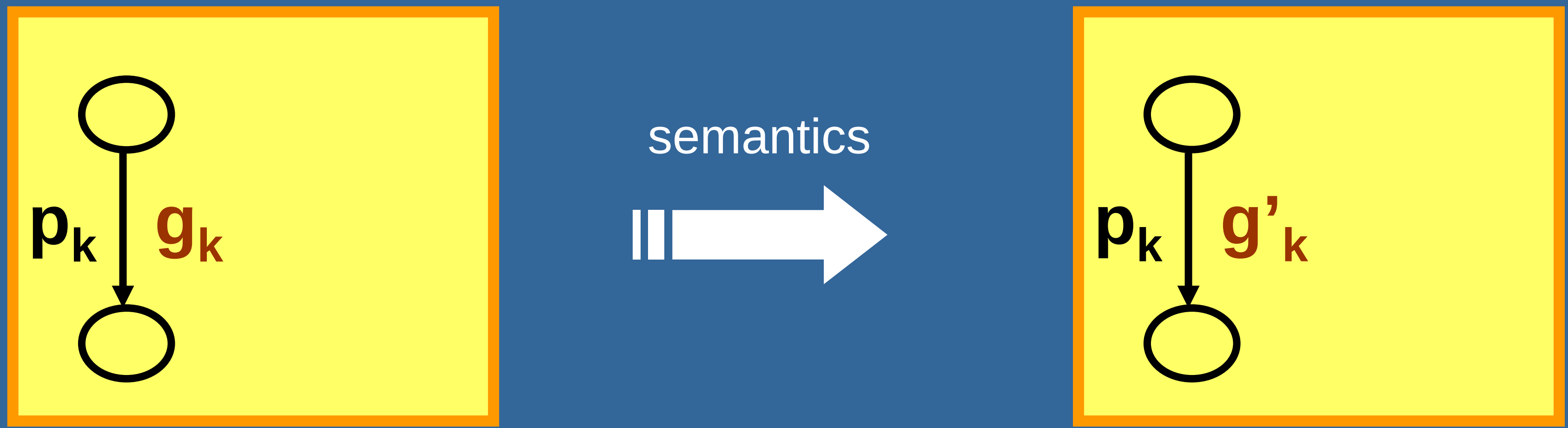
Priority rule	Restricted guard g1'
true $\rightarrow$ p1 $\langle$ p2	$g1' = g1 \wedge \neg\, g2$
C $\rightarrow$ p1 $\langle$ p2	$g1' = g1 \wedge \neg(C \wedge g2)$

Priorities – Definition (2)

$pr = \{ C_i \rightarrow \langle_l \}_i$ is a set of *priority rules*, where

- $\{C_i\}_i$ is a set of disjoint state predicates
- $\langle_i \subseteq$ Interactions $\times$ Interactions is a strict partial order

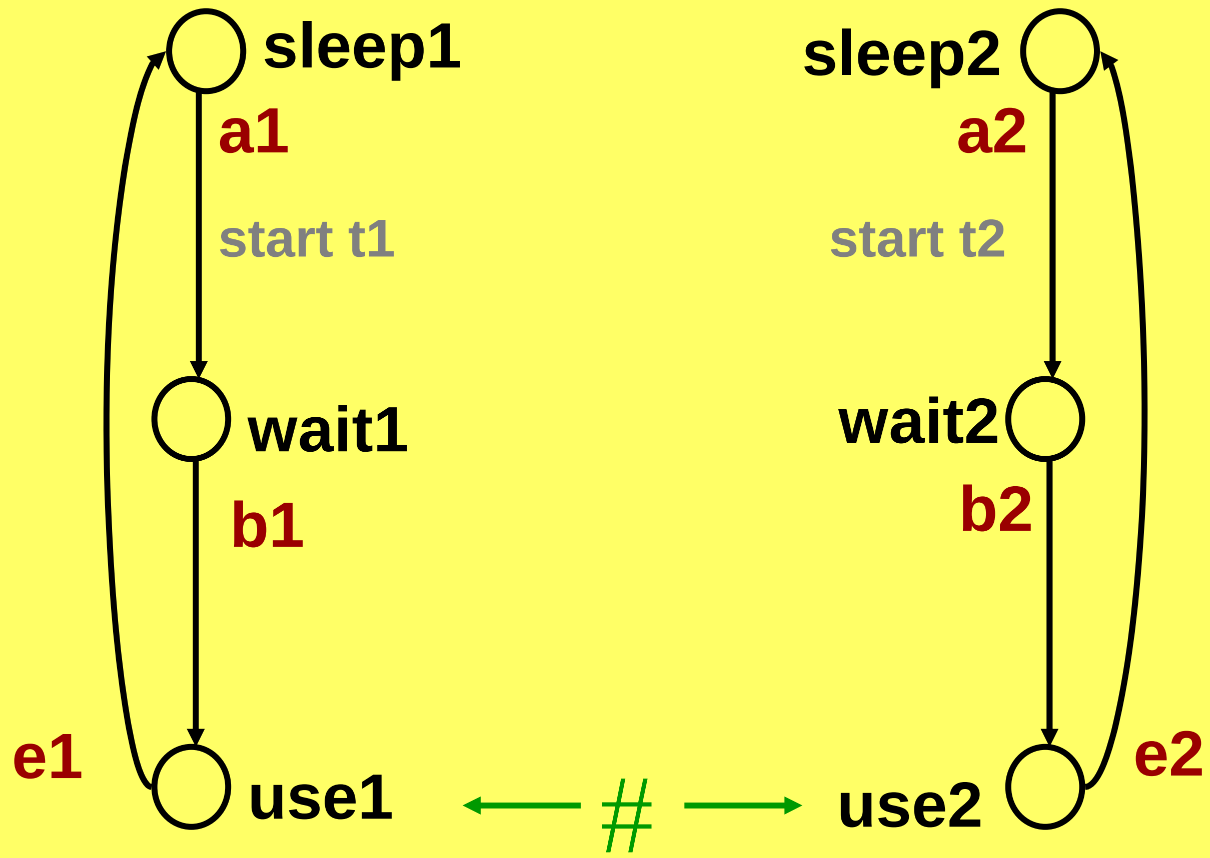
$pr = \{ C_i \rightarrow \langle_i \}_i$



$$g'_k = g_k \wedge \bigwedge_{C \rightarrow \langle \in pr} \left(C \Rightarrow \bigwedge_{pk \langle pi} \neg g_i \right)$$

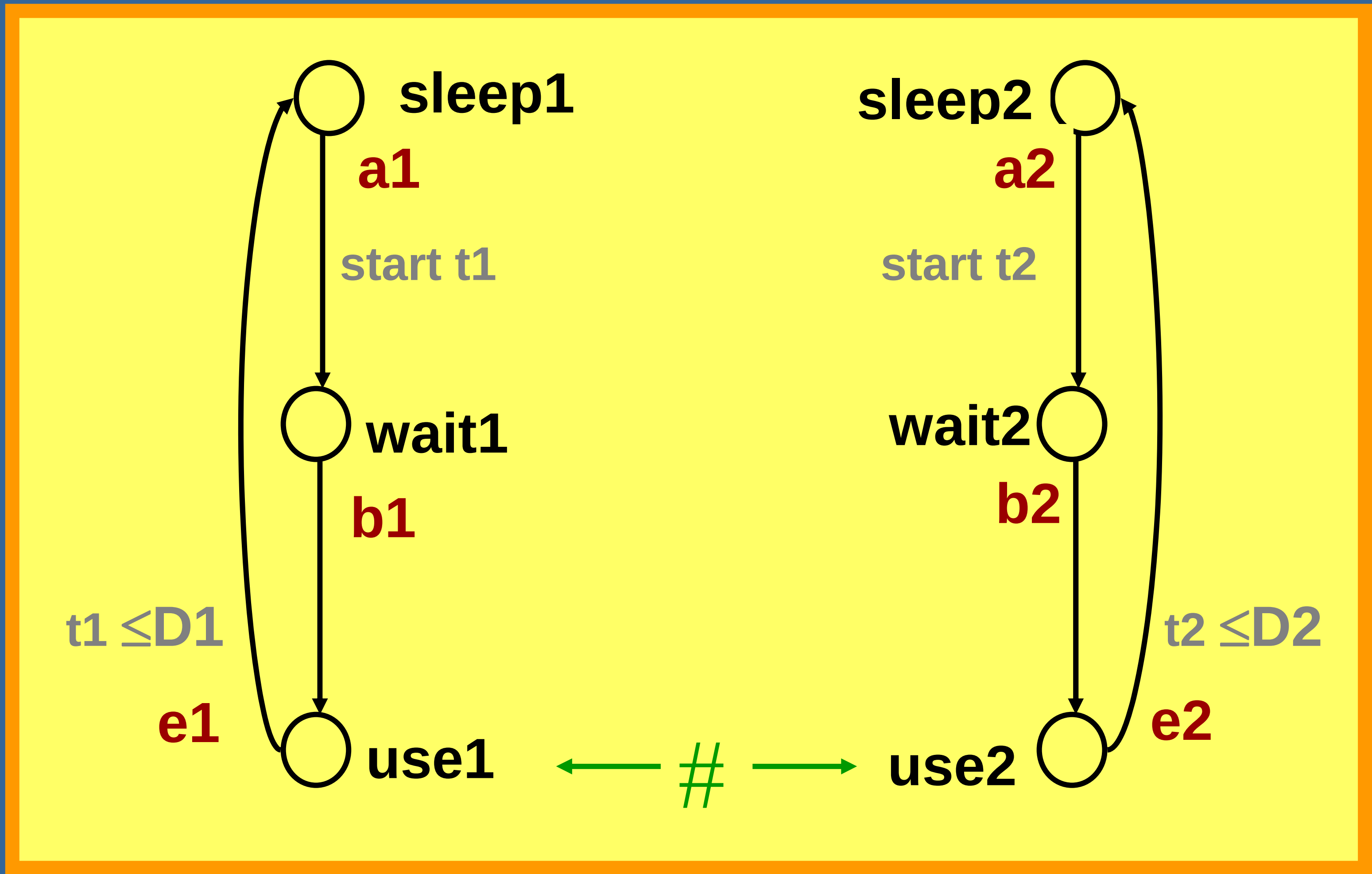
Priorities – Example: FIFO policy

$t1 \leq t2 \rightarrow b1 \langle\, b2$ $t2 < t1 \rightarrow b2 \langle\, b1$

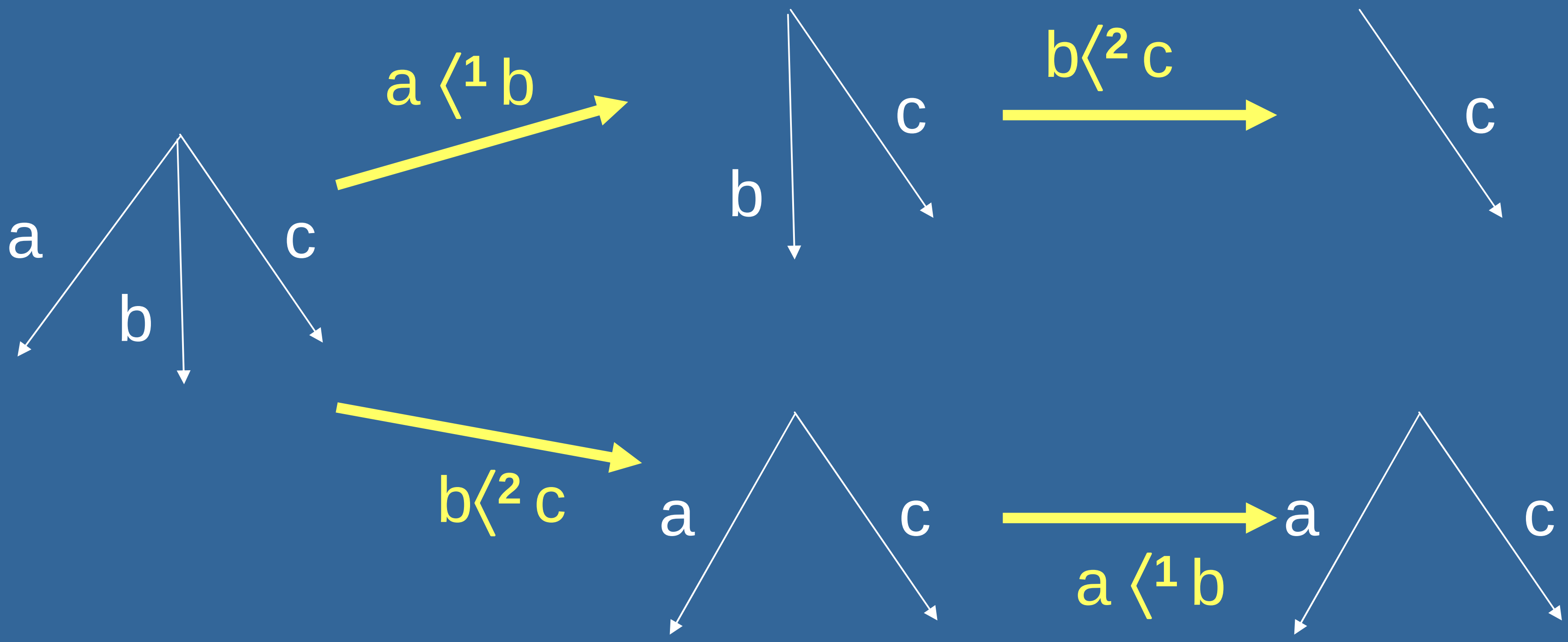


Priorities – Example: EDF policy

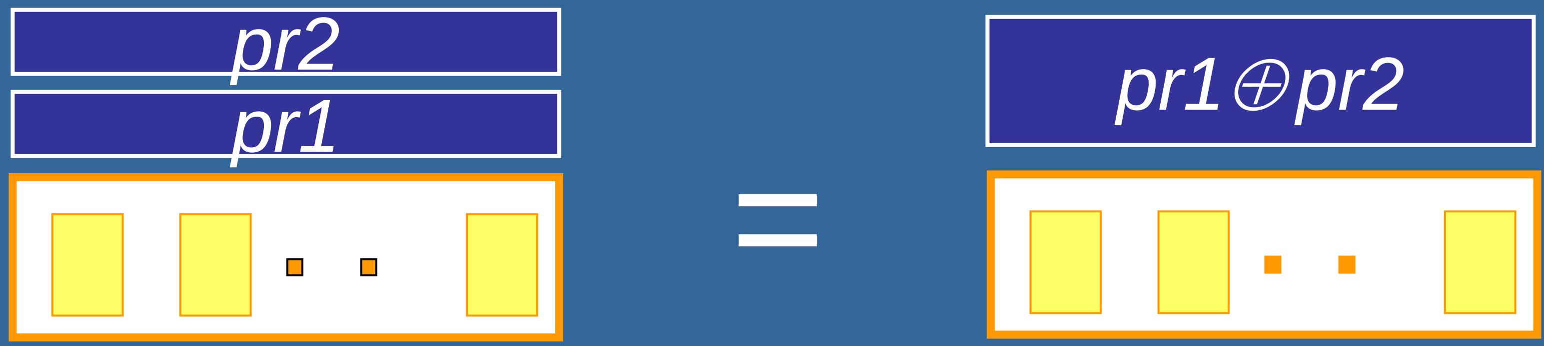
$D1-t1 \leq D2-t2 \rightarrow b2 \langle b1$ $D2-t2 < D1-t1 \rightarrow b1 \langle b2$



Priorities – Composition



Take:



pr1⊕ pr2 is the least priority containing pr1∪pr2

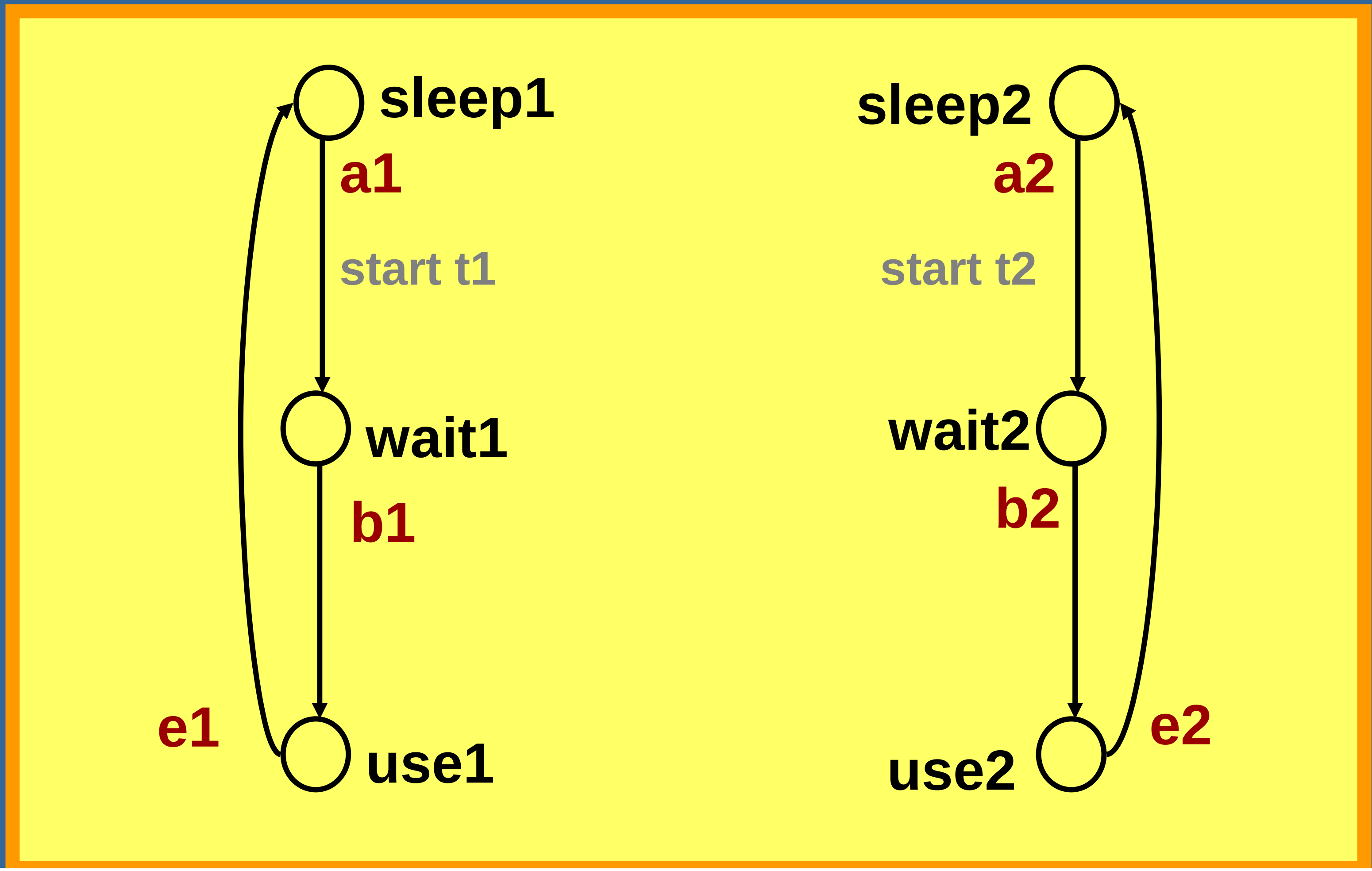
Results :

- The operation ⊕ is partial, associative and commutative
- pr1(pr2(B)) ≠pr1(pr2(B))
- pr1⊕ pr2(B) *refines* pr1∪pr2(B) *refines* pr1(pr2(B))
- Priorities preserve deadlock-freedom

Priorities – Example: Mutual exclusion + FIFO policy

$t1 \leq t2 \rightarrow b1 \langle b2$ $t2 < t1 \rightarrow b2 \langle b1$

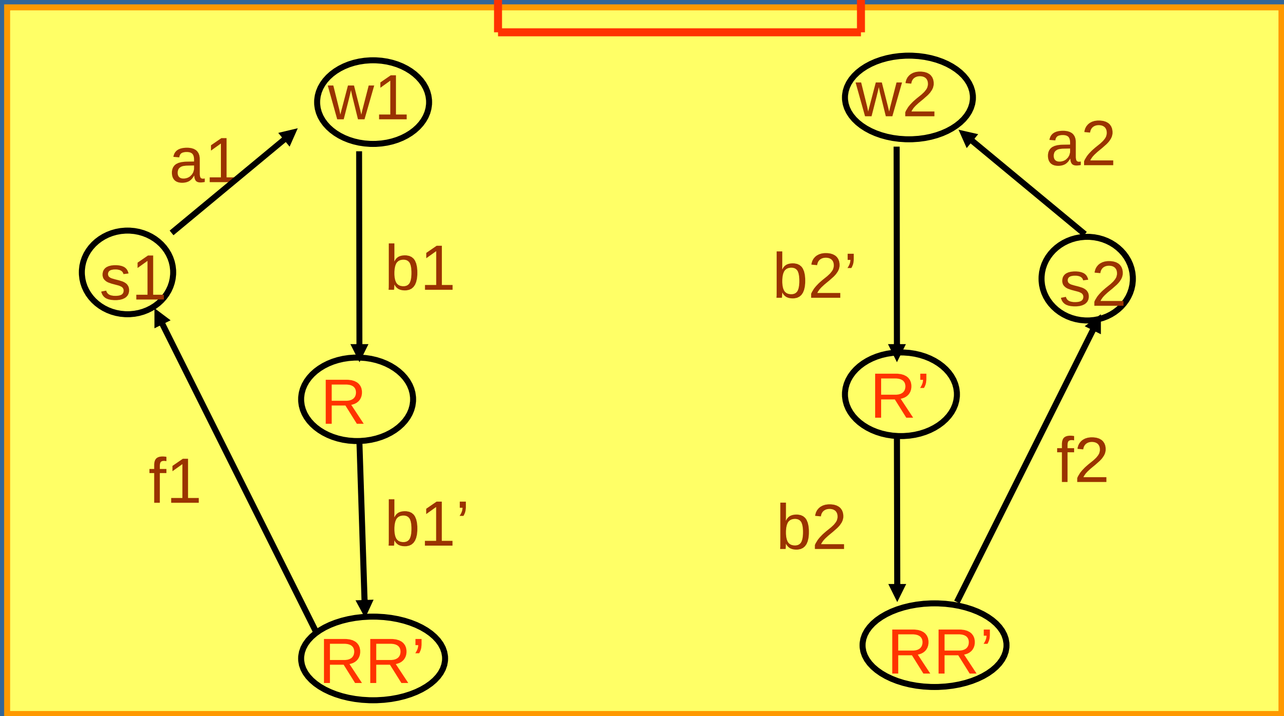
$true \rightarrow b1 \langle e2$ $true \rightarrow b2 \langle e1$



Priorities – Checking for deadlock-freedom: Example

Mutex on R : b1 〈 f2   b2 〈 { f1, b1'}

Mutex on R' : b1' 〈 { f2, b2 }   b2' 〈f1

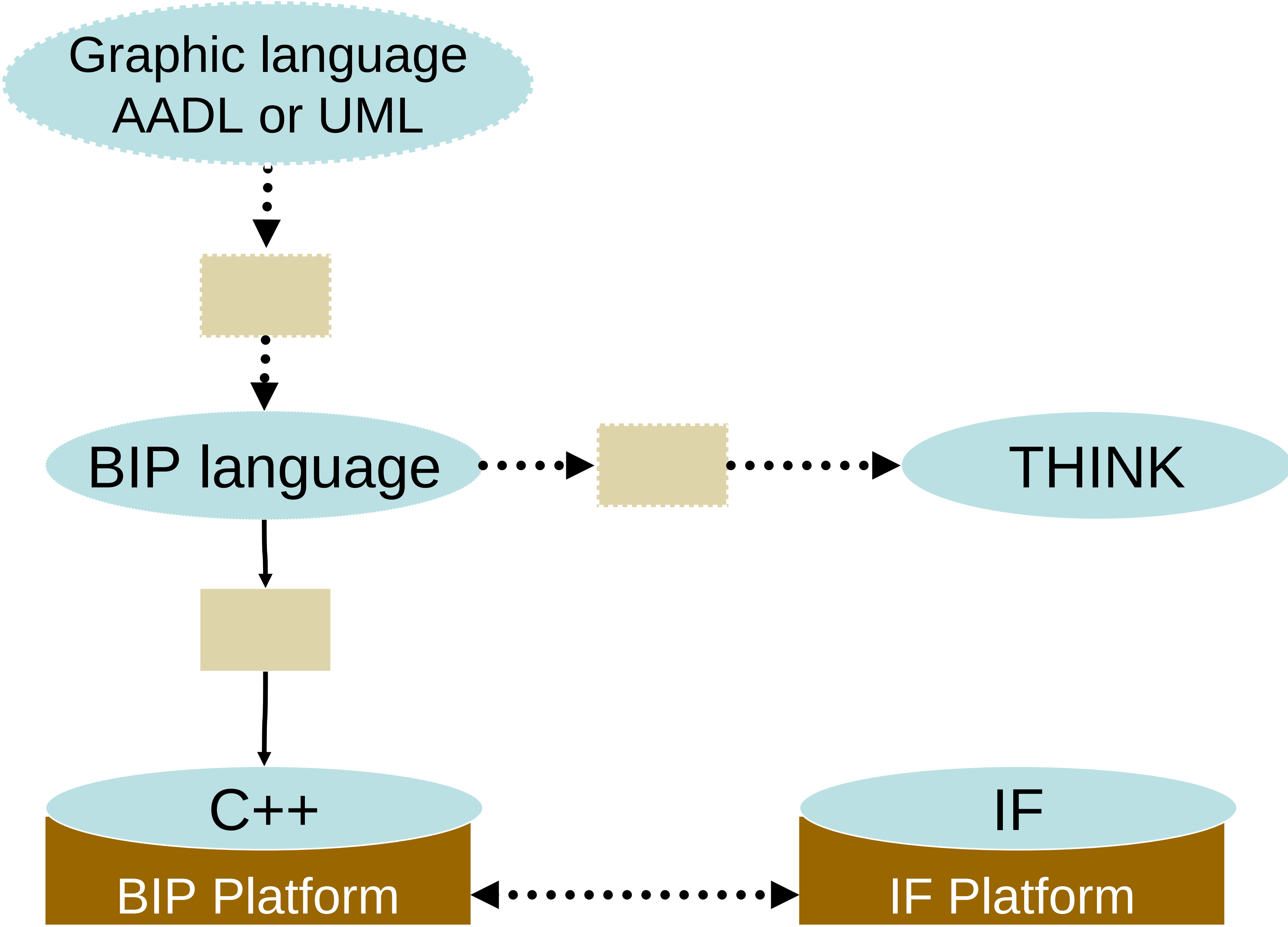


Risk of deadlock: b1' 〈 b2 and b2 〈 b1'

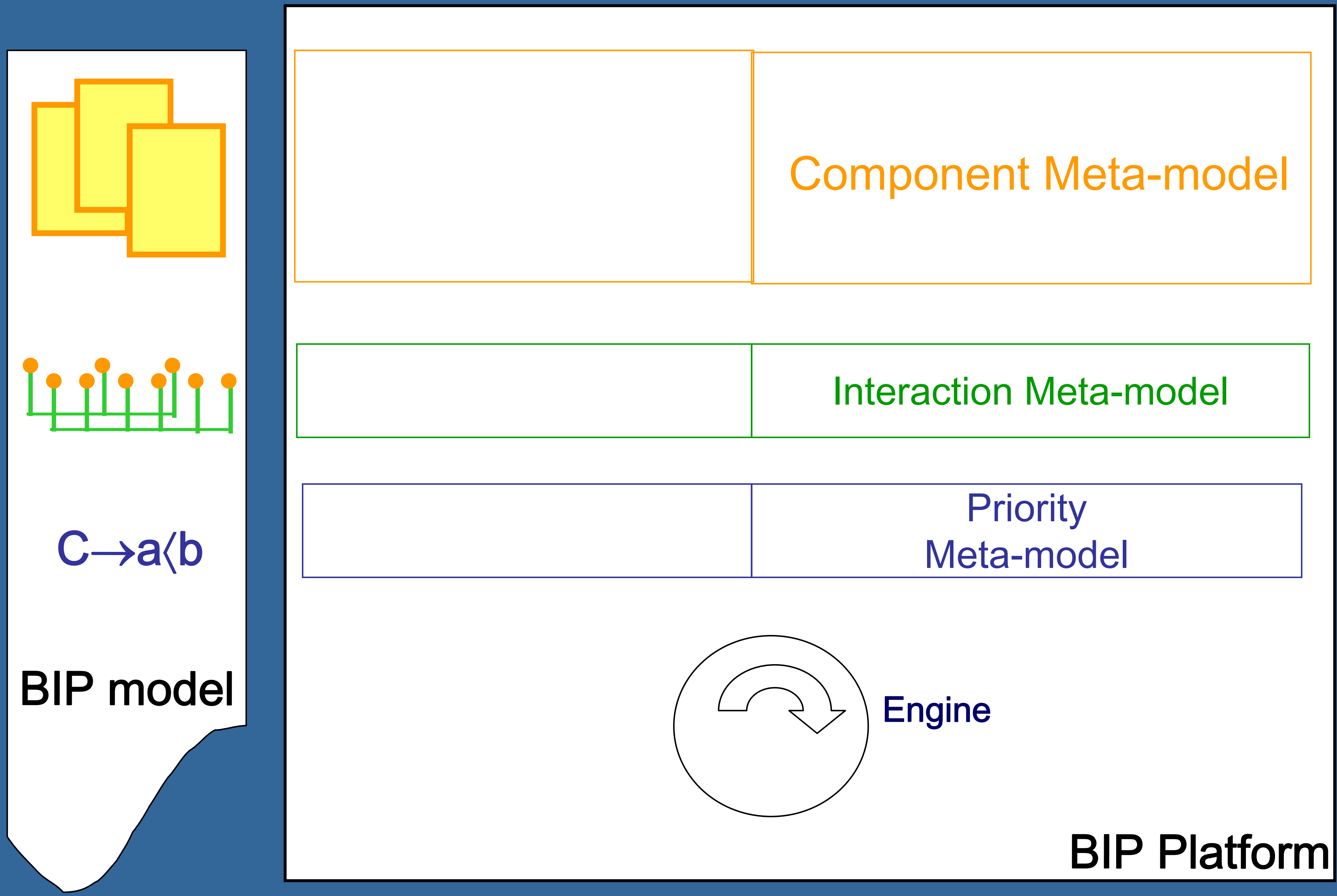
Overview

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

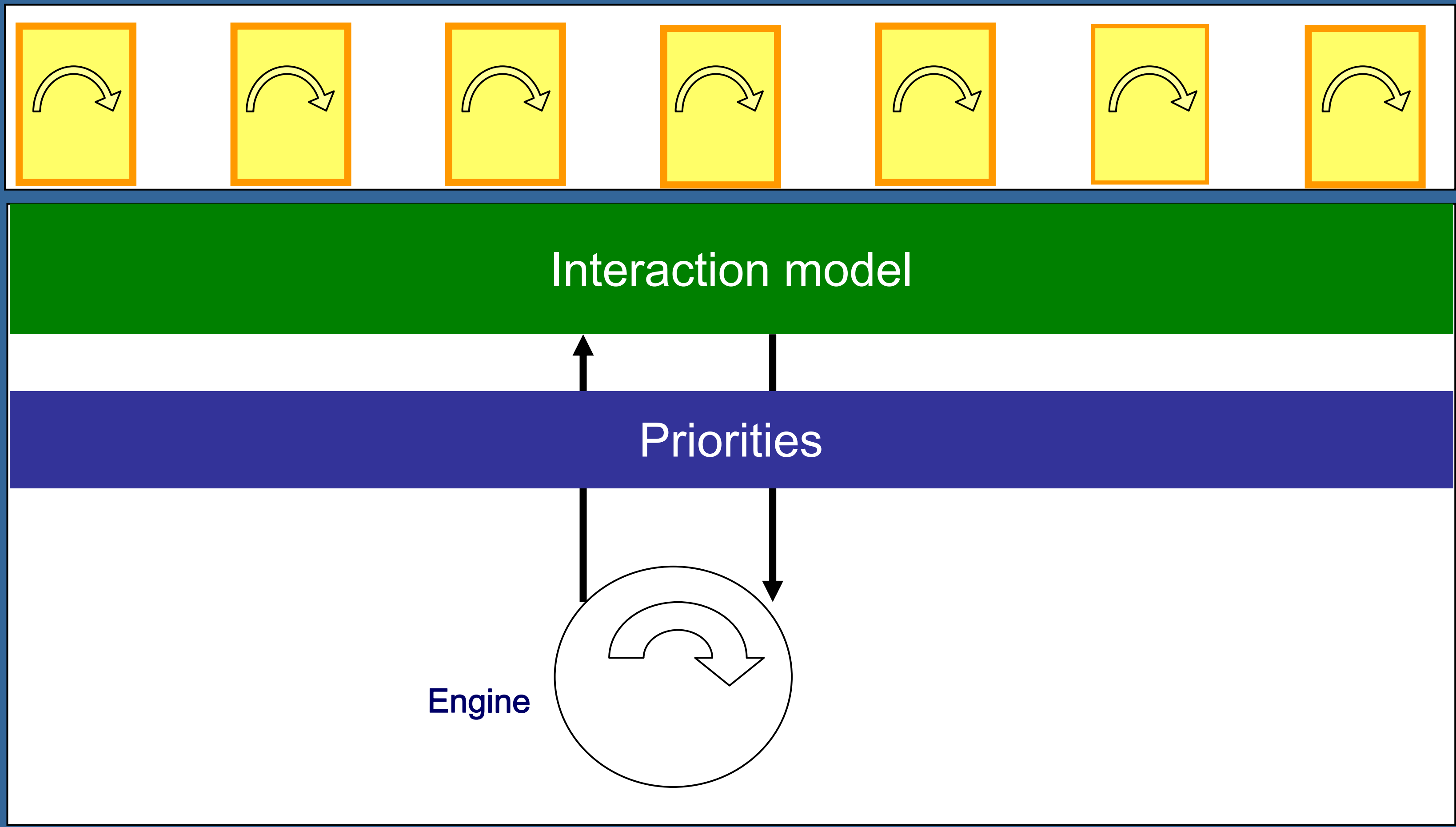
Implementation – the BIP toolset



Implementation – C++ code generation for the BIP platform

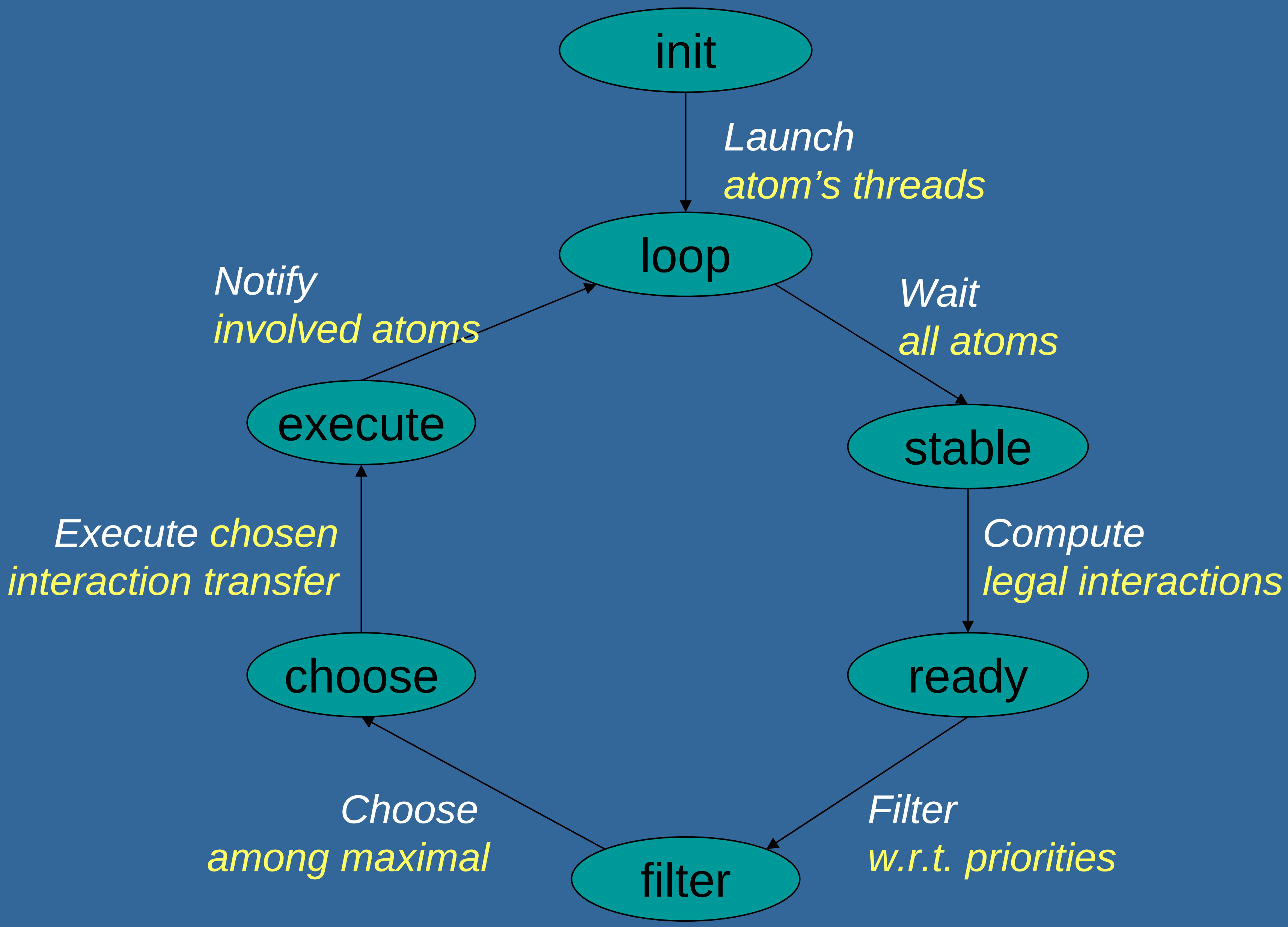


Implementation – The BIP platform



• Code execution and state space exploration features
• Implementation in C++ on Linux using POSIX threads.

Implementation – The BIP platform: The engine



Implementation – the BIP language: atomic component

```
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
```

Implementation – the BIP language: connectors and priorities

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

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

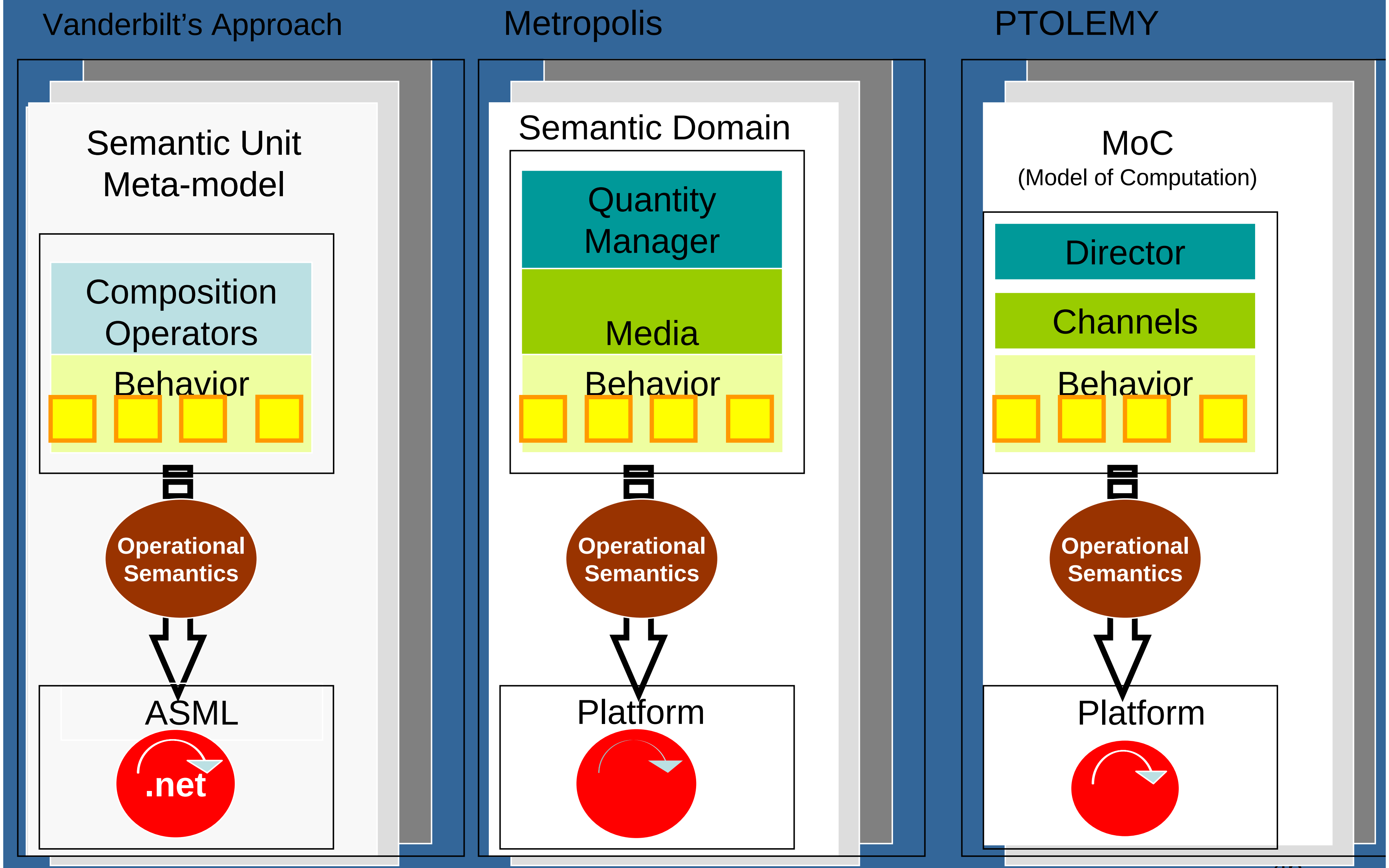
Implementation – the BIP language: compound component

```
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
```

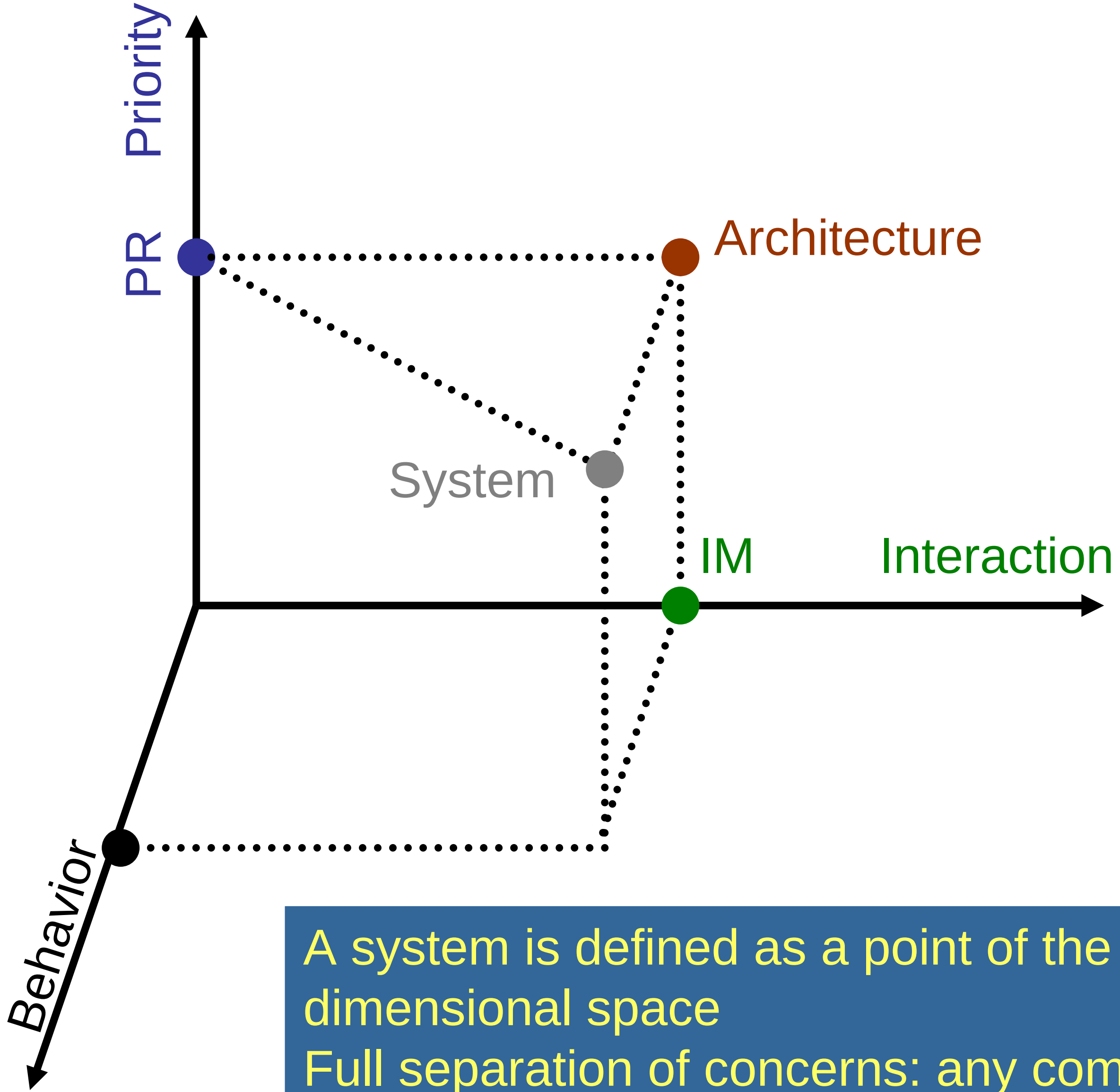
Overview

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

Modeling in BIP–
Other frameworks encompassing heterogeneity

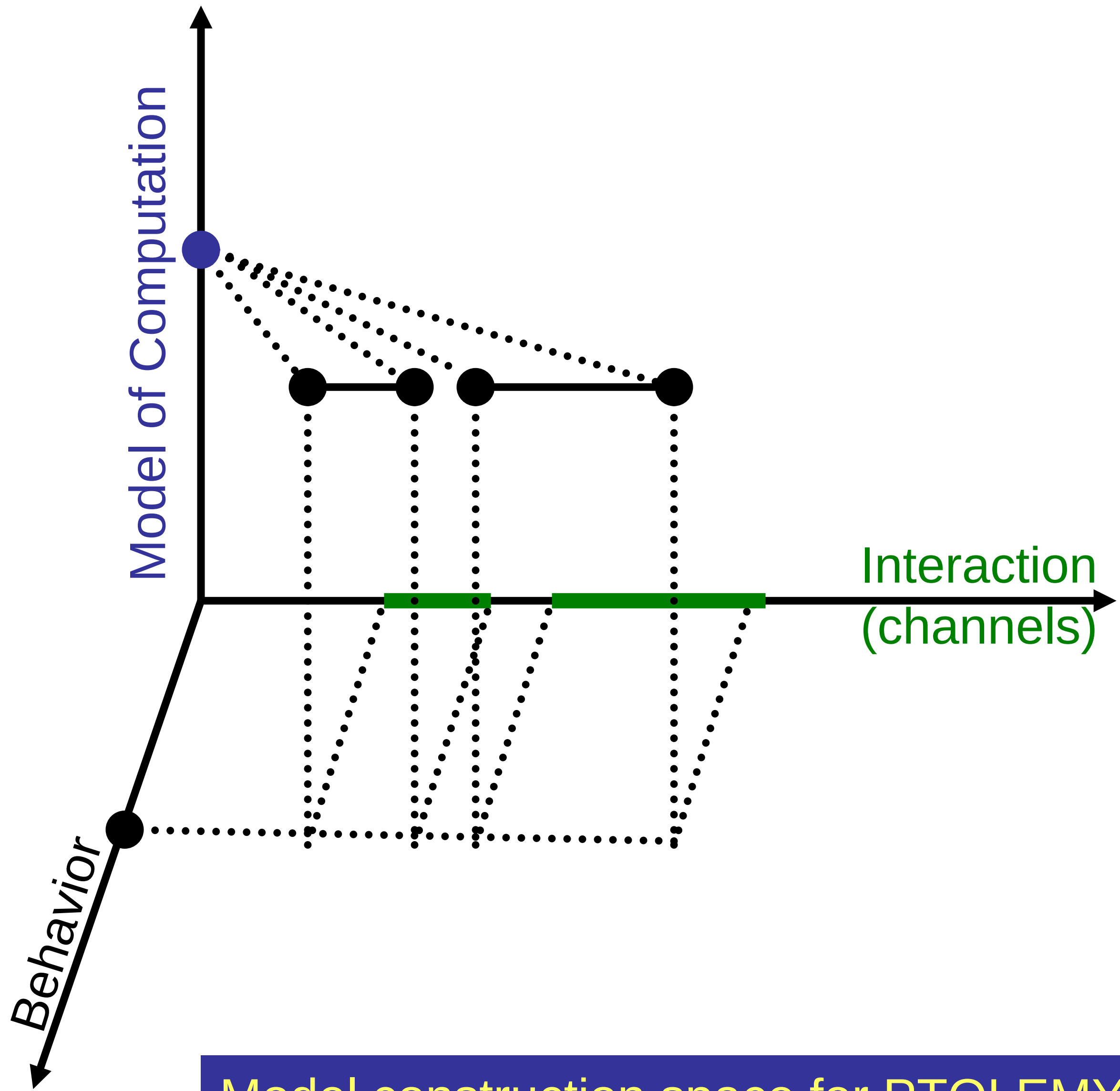


Modeling in BIP– Model construction space



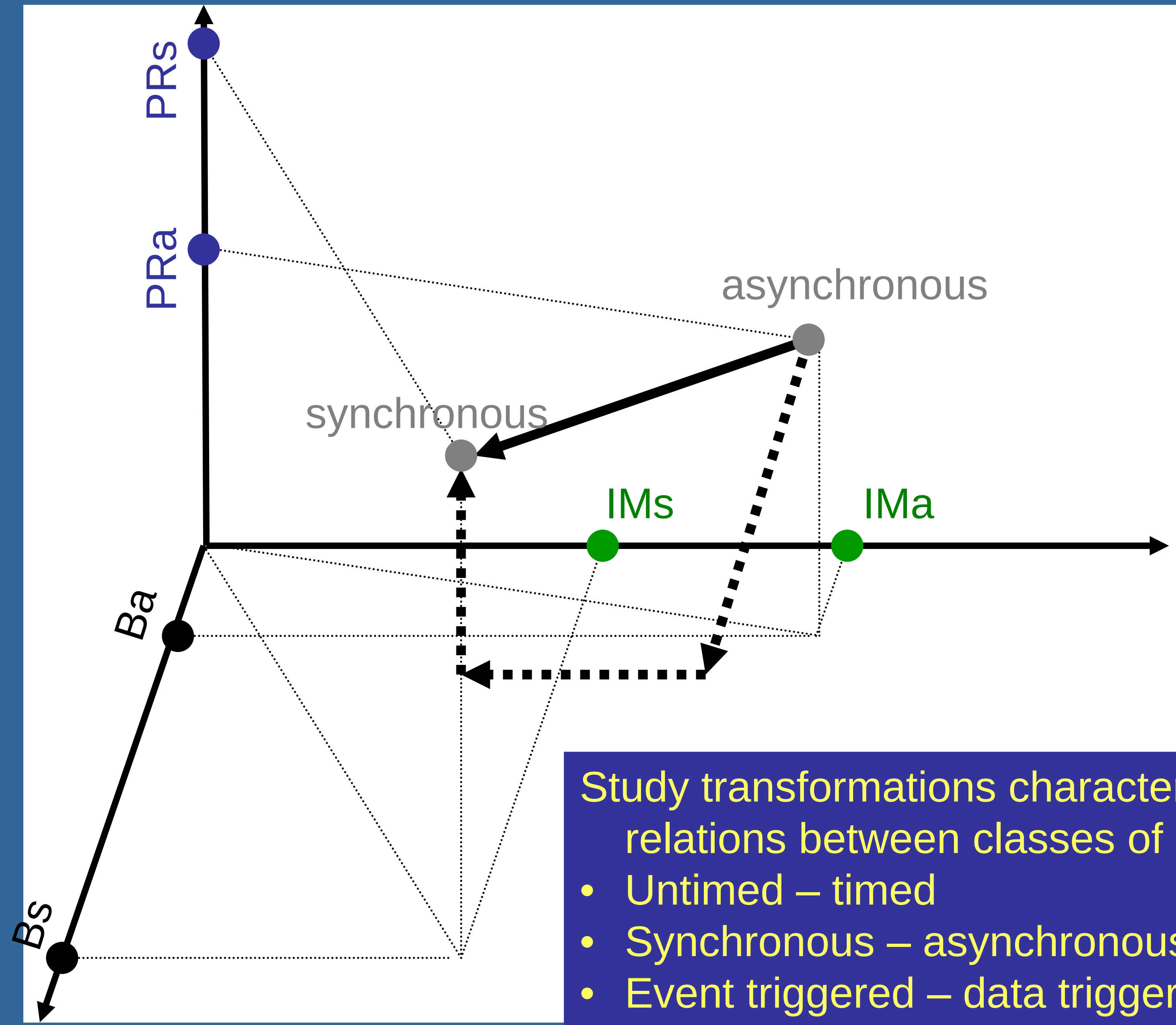
A system is defined as a point of the 3-dimensional space
Full separation of concerns: any combination of coordinates defines a system

Modeling in BIP – Model construction space (2)



Model construction space for PTOLEMY

Modeling in BIP – Relating classes of components

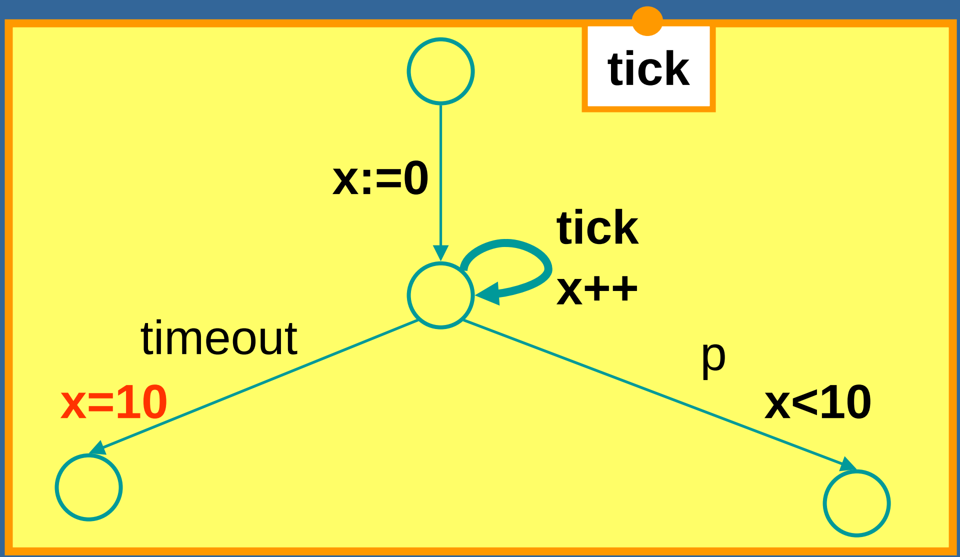


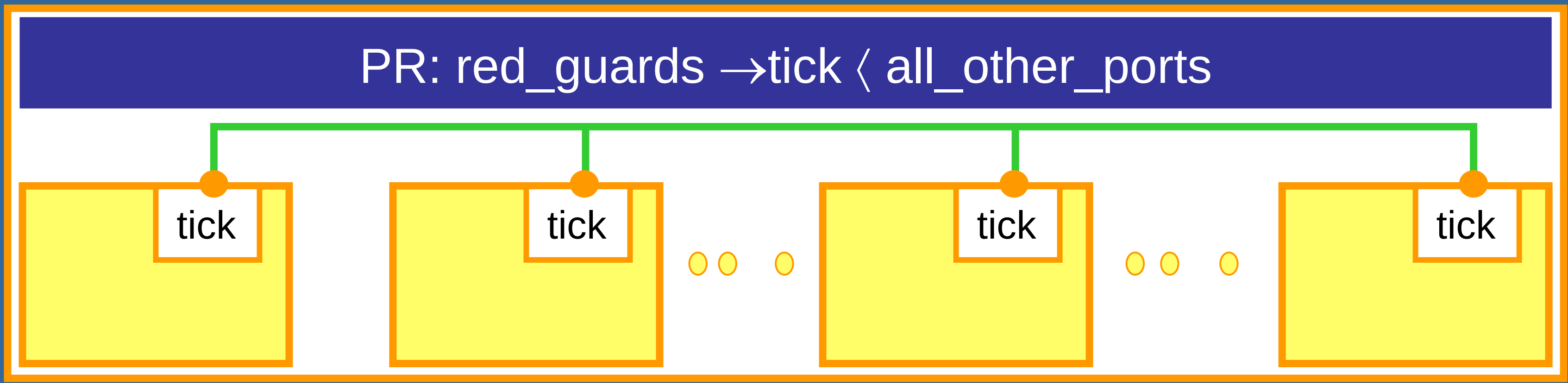
Study transformations characterizing relations between classes of systems:

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

Modeling in BIP – Timed systems

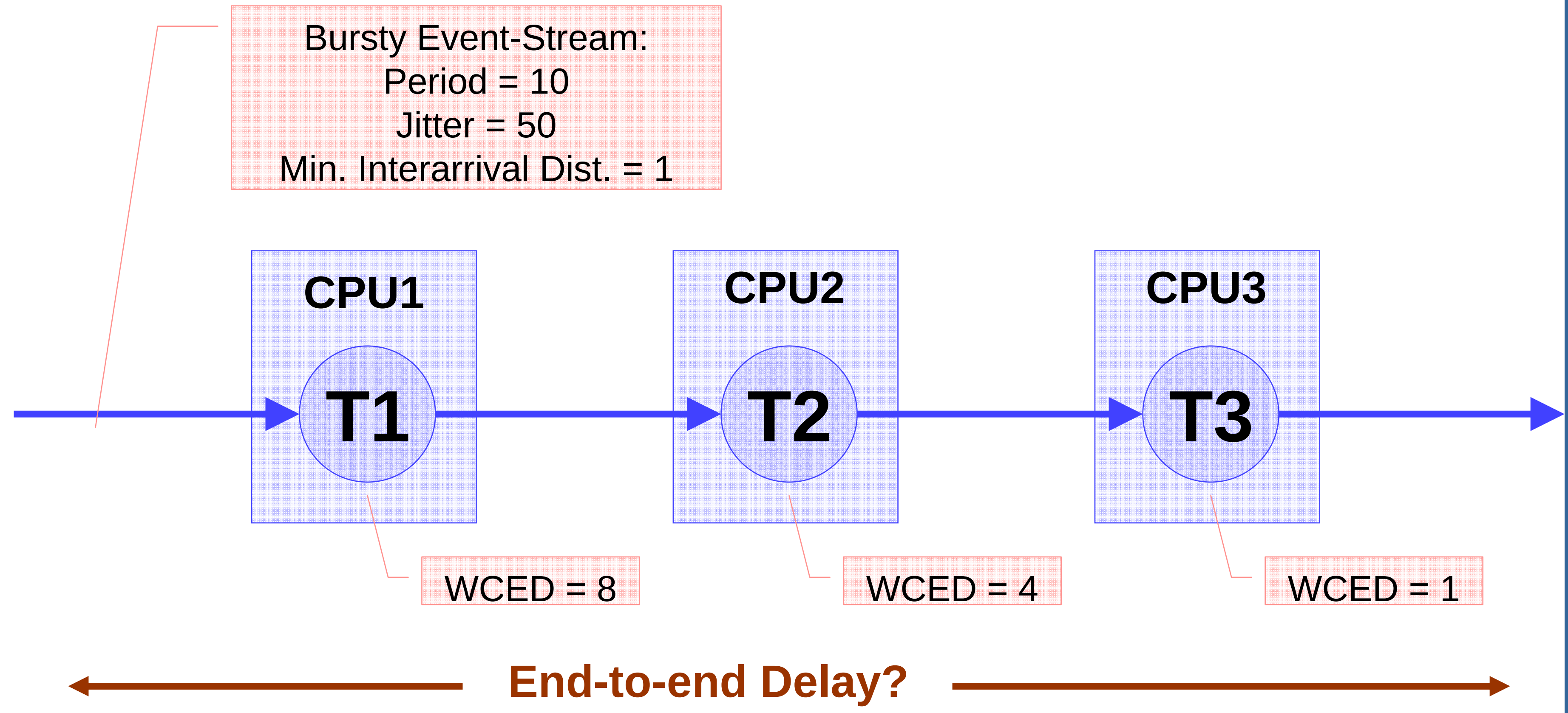
Timed Component





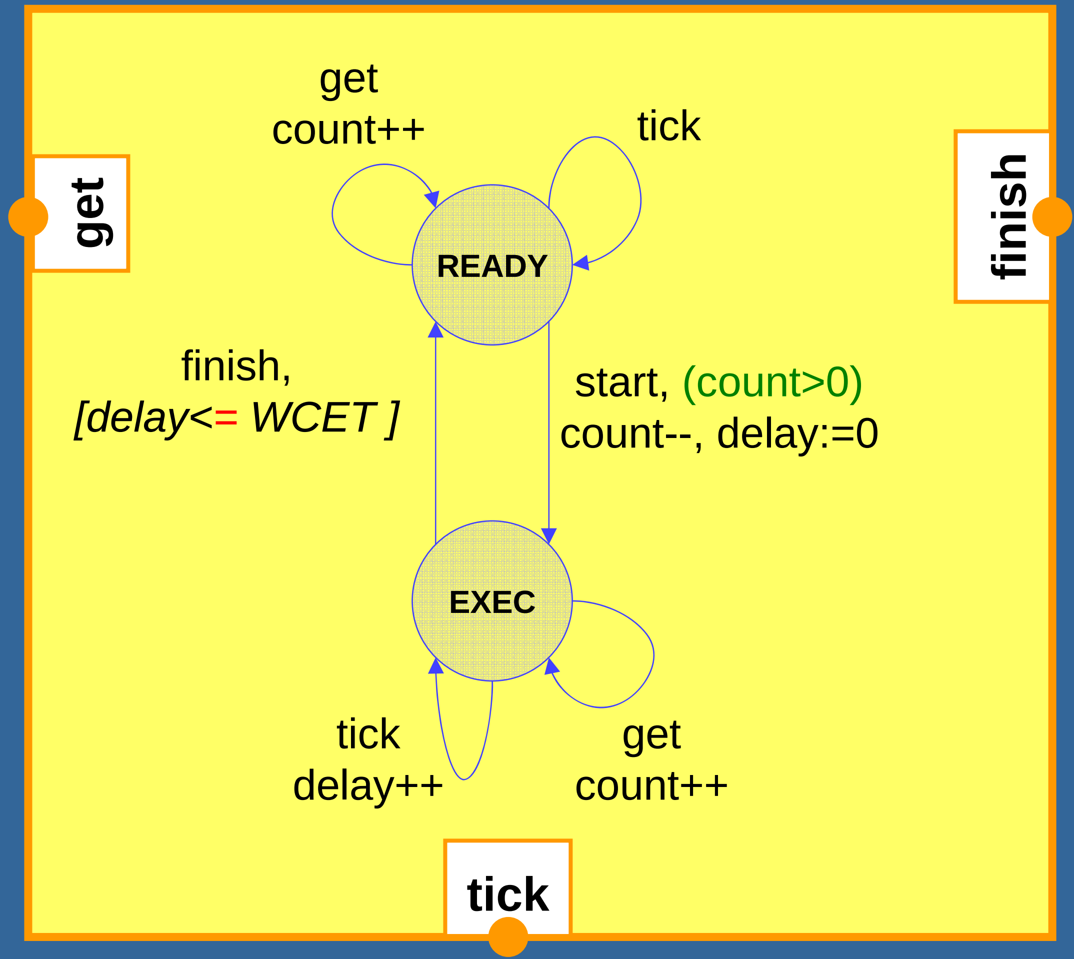
Timed architecture

Modeling in BIP – Timed systems: Example



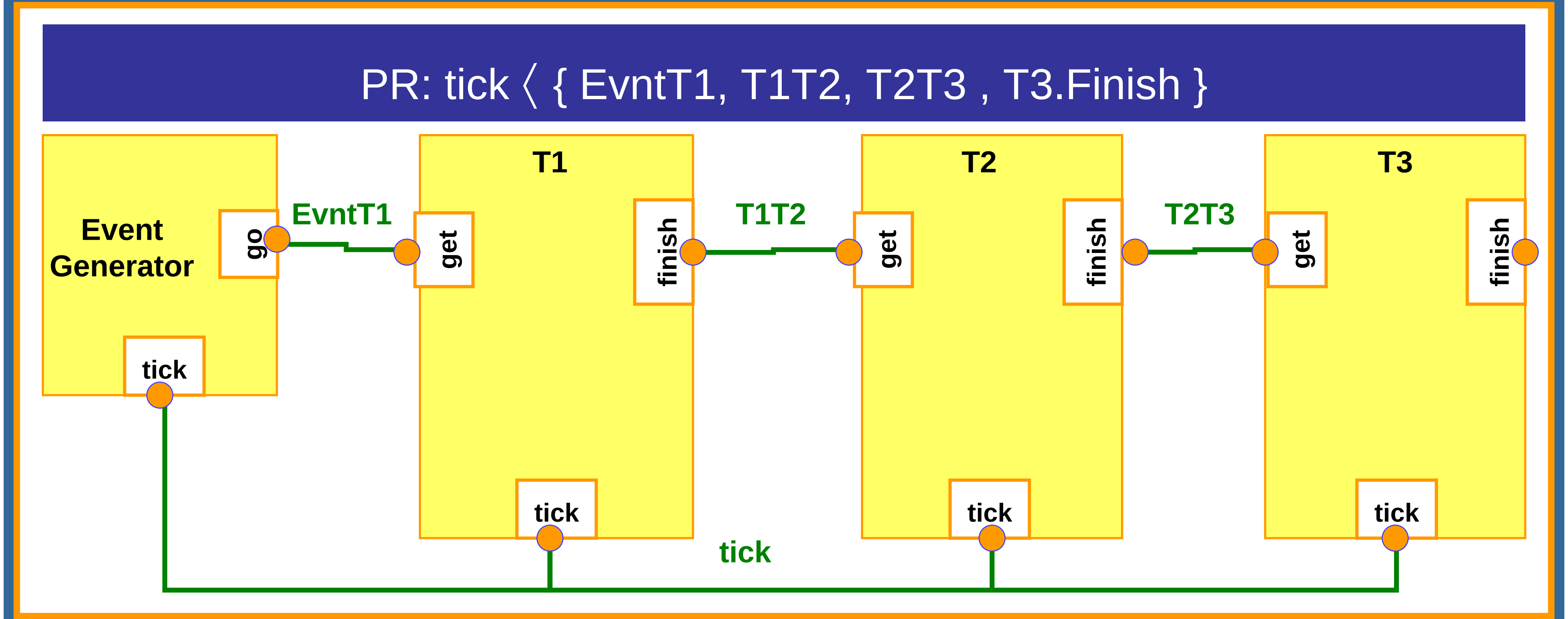
Source: http://www.tik.ee.ethz.ch/~leiden05
Workshop on Distributed Embedded Systems, Leiden, November 21-24, 2005

Modeling in BIP – Timed systems: Example (2)



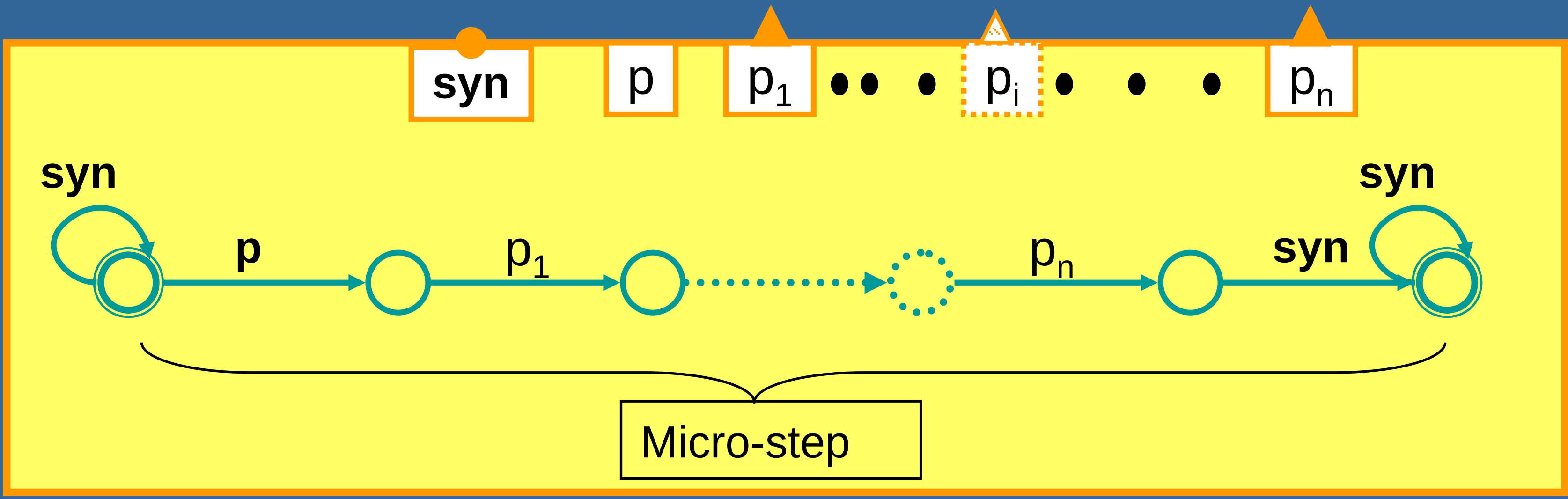
Task Component

Modeling in BIP – Timed systems: Example (3)

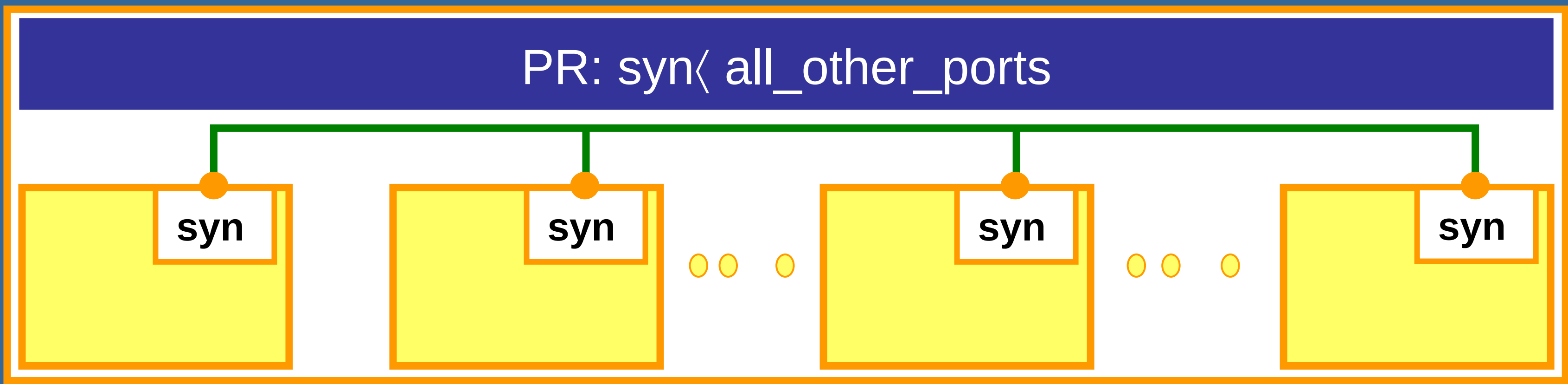


System architecture

Modeling in BIP – Synchronous systems

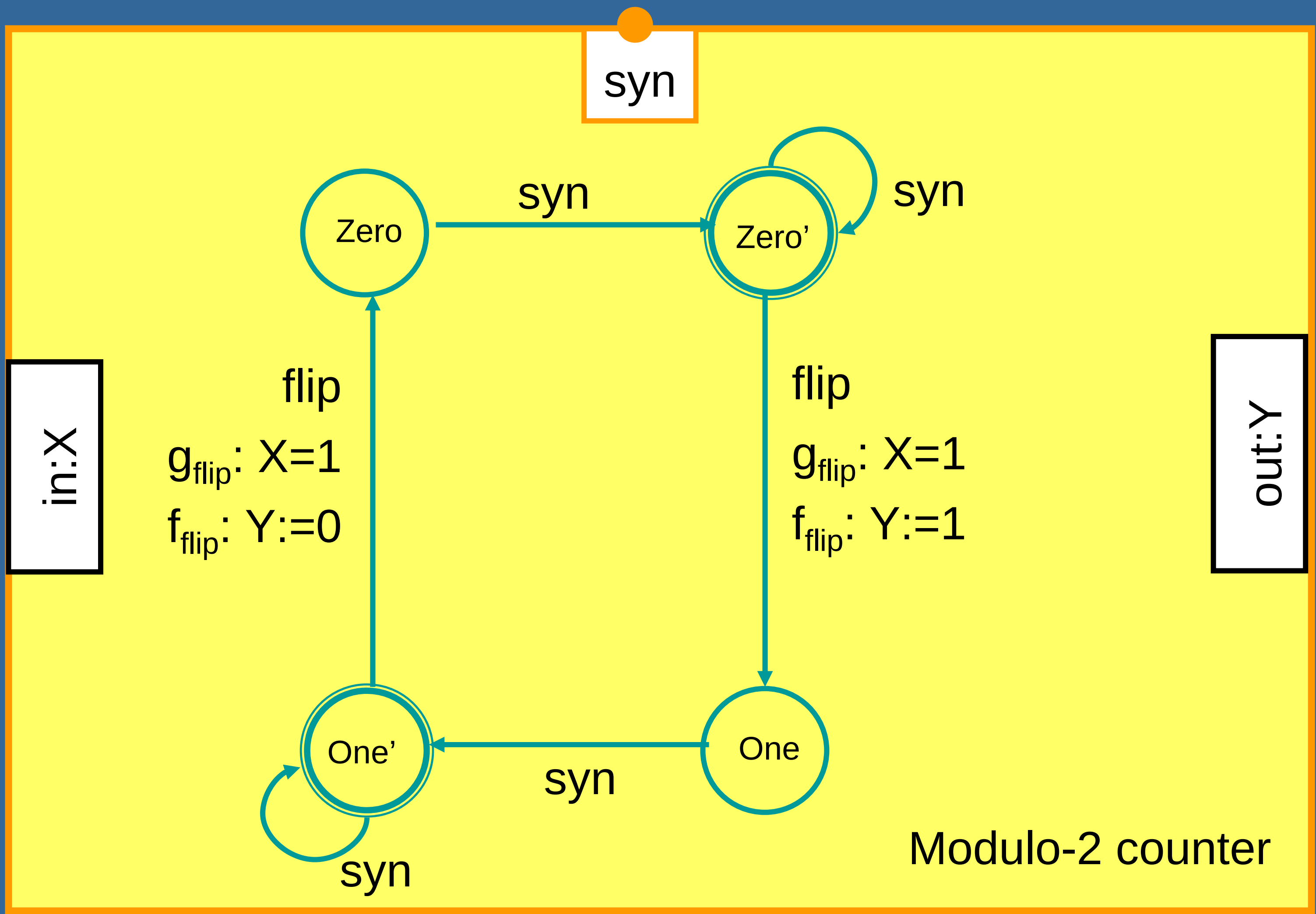


Synchronous component

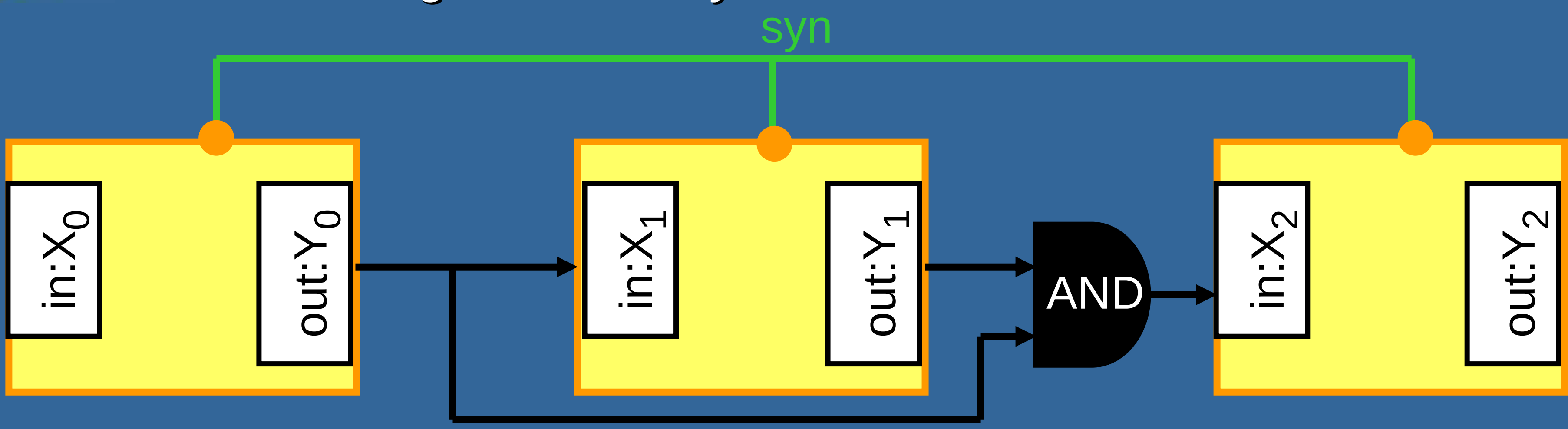


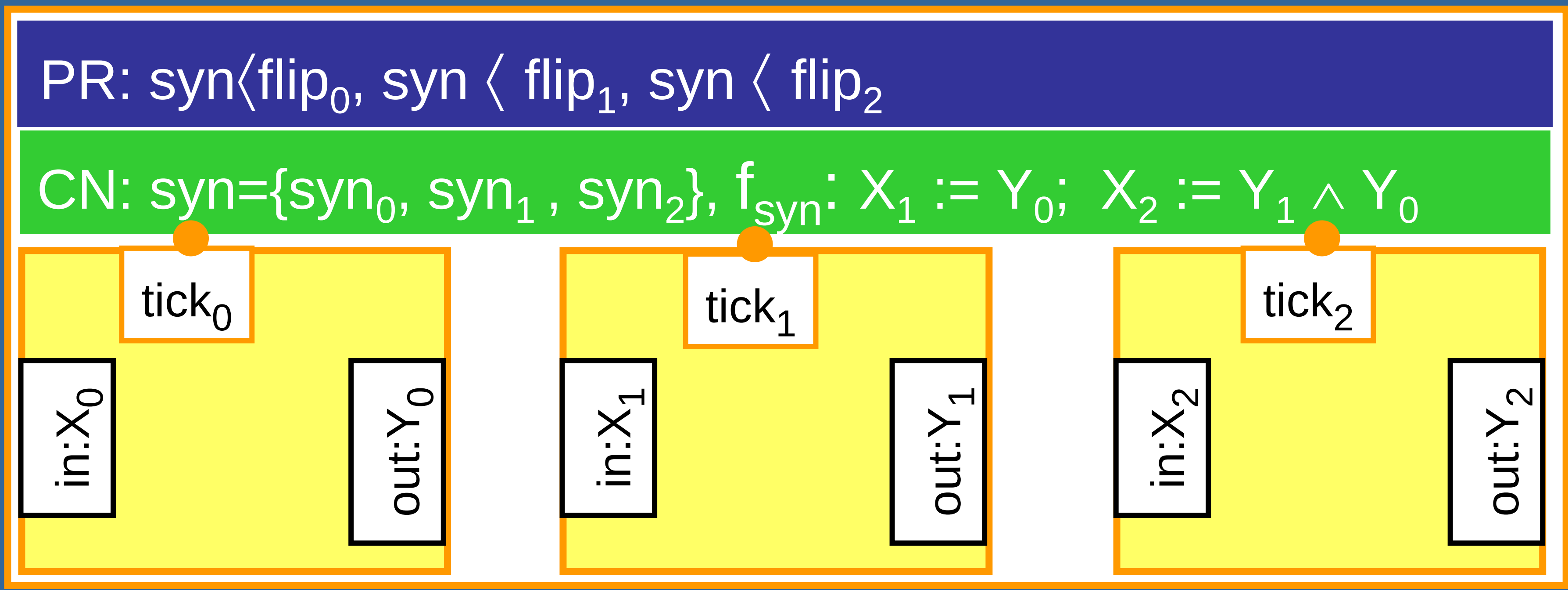
Synchronous architecture

Modeling in BIP – Synchronous mod2 counter



Modeling in BIP – Synchronous mod8 counter





Modeling in BIP –
MPEG4 Video encoder: Componentization

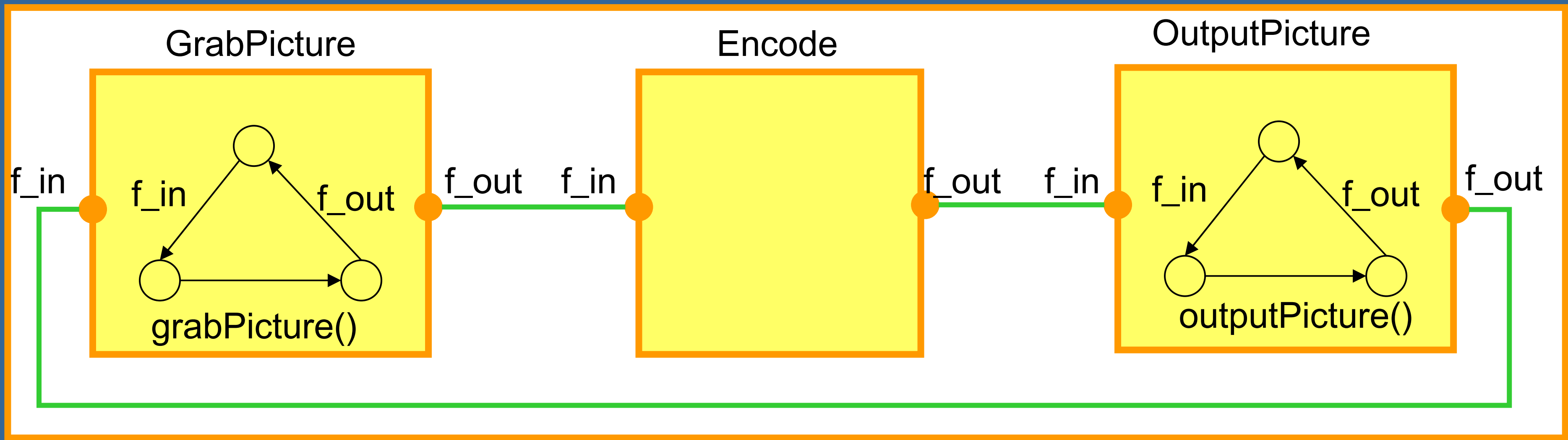
Transform a monolithic program into a componentized one

++ reconfigurability, schedulability

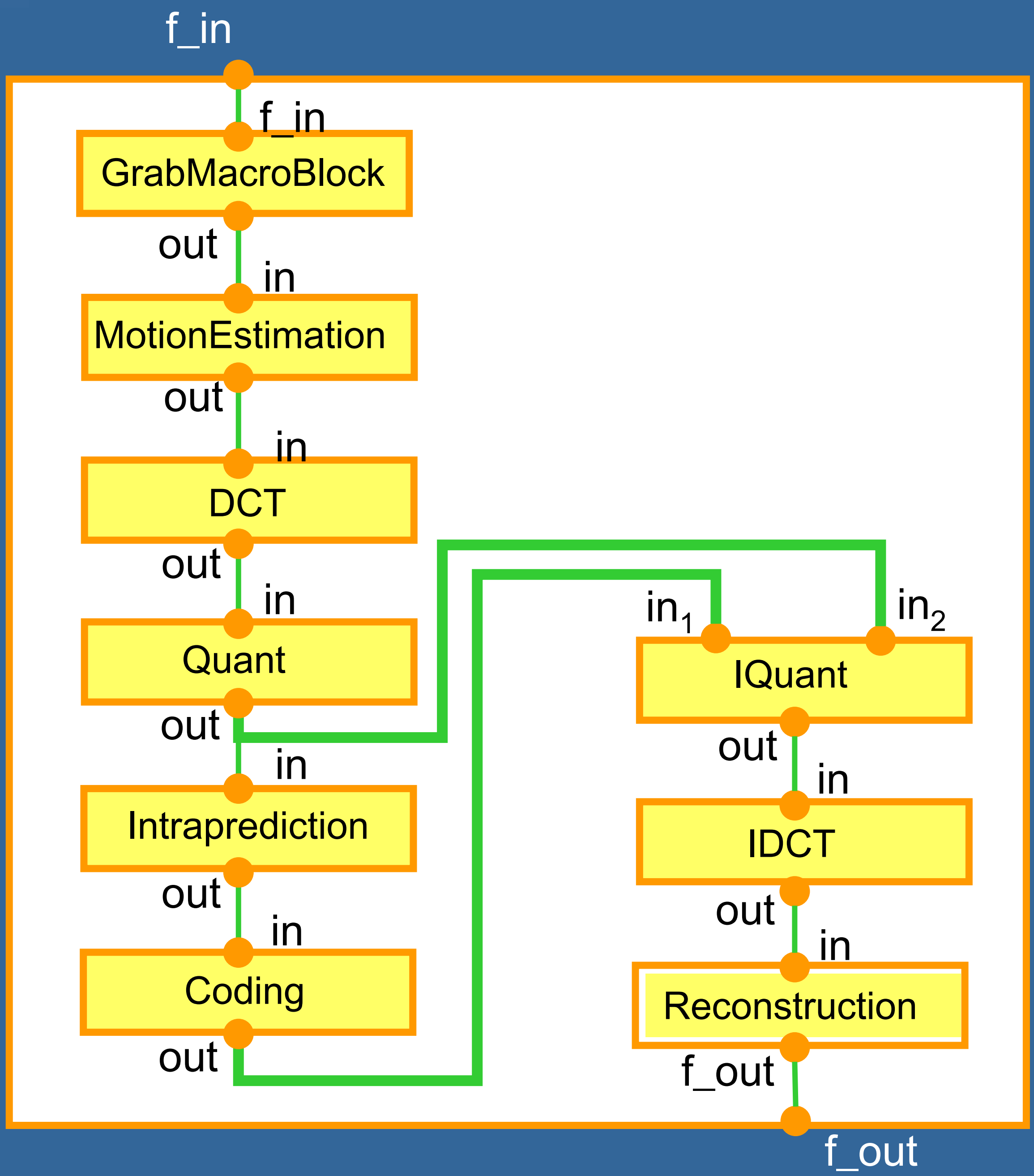
– – overheads (memory, execution time)

Video encoder characteristics:

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



Modeling in BIP –Video encoder: The Encode component

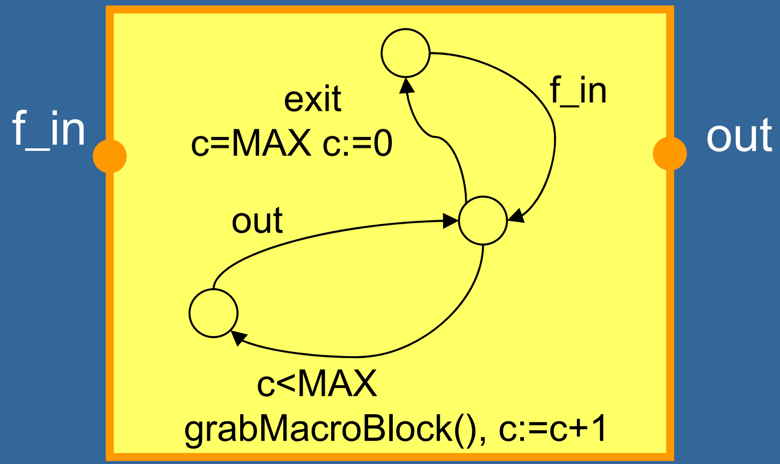


GrabMacroBlock: splits a frame in (W*H)/256 macro blocks, outputs one at a time

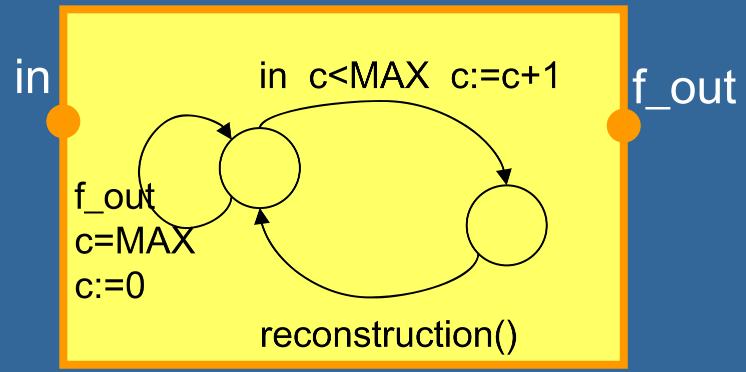
Reconstruction: regenerates the encoded frame from the encoded macro blocks.

: buffered connections

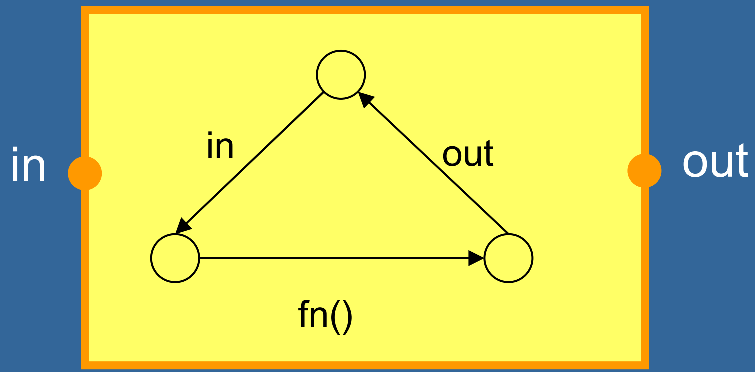
Modeling in BIP – Video encoder : Atomic components



GrabMacroBlock



Reconstruction



Generic Functional component

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

Modeling in BIP – Video encoder: The BIP 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.

Modeling in BIP – Video encoder : 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
- Interaction modeling
- Priority modeling
- Implementation
- Modeling systems in BIP
- Discussion

Discussion – The BIP framework: summary

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

Clear separation between structure (interaction +priority) and behavior

- Structure is a first class entity
- Layered description => separation of concerns => incrementality
- Correctness-by-construction techniques for deadlock-freedom and liveness, based (mainly) on sufficient conditions on the structure

Discussion – The BIP framework: Work directions

Theory

- Study Component Algebras CA= (**B**, GL,⊕, ≅), where
  - (GL,⊕) is a monoid and ⊕ is idempotent
  - ≅ is a congruence compatible with operational semantics

- Study notions of **expressiveness** characterizing structure**:** Given two component algebras defined on the same set of atomic components,

CA1 **is more expressive** than CA2

if $\forall P \;\; \exists gl2 \in GL2 \; gl2(B1, .., Bn) \text{ sat } P \Rightarrow \exists \; gl1 \in GL1.\; gl1(B1, \ldots Bn) \text{ sat } P$

- Model transformations
  - relating classes of systems
  - preserving properties

- Distributed implementations of BIP

Discussion - The BIP framework: Work directions (2)

Methodology

- Using BIP as a programming model
- Reference architectures in BIP

BIP toolset Implementation

- Generation of BIP models from system description languages such as SysML (IST/SPEEDS project), AADL and SystemC (ITEA/Spices project)
- Model transformation techniques in particular for code optimization
- Validation techniques
  - connection to Verimag's IF simulation/validation environment
  - specific techniques e.g. checking conditions for correctness by construction

More about BIP:

- http://www-verimag.imag.fr/index.php?page=tools
- Email to Joseph.Sifakis@imag.fr

THANK YOU