Rigorous Component-based System Design Using the BIP Framework

Saddek Bensalem

Join work with

A. Basu, M. Bozga, P. Bougos, J. Sifakis

VERIMAG Laboratory (Grenoble, France)

Fifth Annual Layered Assurance Workshop

December 5-6, 2011 Orlando, Florida

Outline

- Introduction
- The BIP Framework
  - Basic Concepts and Results
  - The BIP Language and the associated tools
- The Rigorous System Design Flow
- Discussion

System Design

Today

We master, at high cost:
- critical systems of low to medium complexity
  ex: flight controllers
- complex best effort systems
  ex: telecommunication systems

Tomorrow

We need
- affordable critical systems
  ex: active safety, health, robots
- integration of systems of systems
  ex: internet of things, smart grids, ambient intelligence

A long way to go …

Complexity: mainly for building systems by integration of existing components

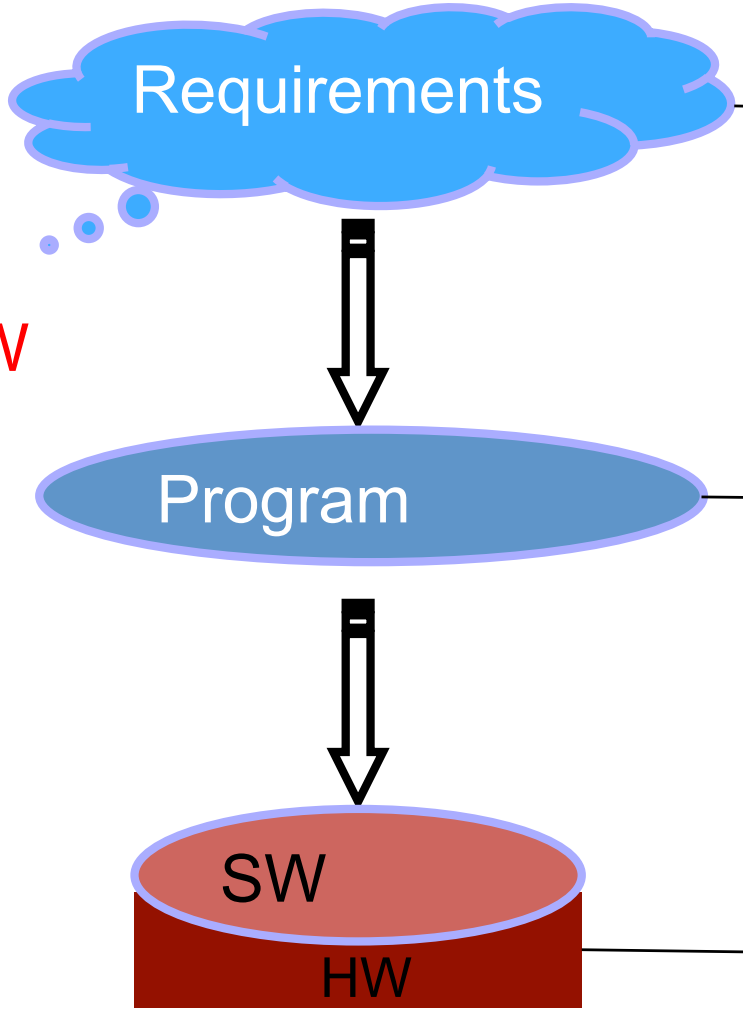
Design Approaches:
- empirical and based on the expertise of the teams
- reuse/extend/improve solutions that have been proven efficient and robust

Lack of constructivity results:
correctness cannot be guaranteed by design, validation is mandatory

System Design

System design is the process leading to a mixed software-hardware system meeting given requirements

Different from pure SW or pure HW design!



The expected behavior of the system to be designed with respect to its potential users and its environment

Executable platform-independent model meeting the requirements

System composed of HW and SW – the HW platform may be given

System Design vs Software Design

Programs and Algorithms

- Terminating
- Deterministic
- Behaviour: relations independent from physical resources needed for their execution
- Correctness independent in the dynamic characteristics of the execution platform

Systems

- Non-terminating
- Non-deterministic
- Behaviour: relations between histories of inputs and histories of outputs
- Correctness dependent on the dynamic characteristics of the execution platform

Trends in System Design

ES must jointly meet technical requirements

- **Reactivity:** **responding within a known and bounded delay.**
- **Autonomy:** **providing continuous service without human intervention.**
- **Dependability:** **invulnerability to threats including attacks, hardware failures, software execution errors**
- **Scalability:** **performance increase is commensurable with the increase of resources**

In addition

- ES must meet requirements for **optimal/quality** as they are integrated in **mass-market** products

What is needed?

Foundations for a rigorous system design

Rigorous System Design

Three grand Challenges

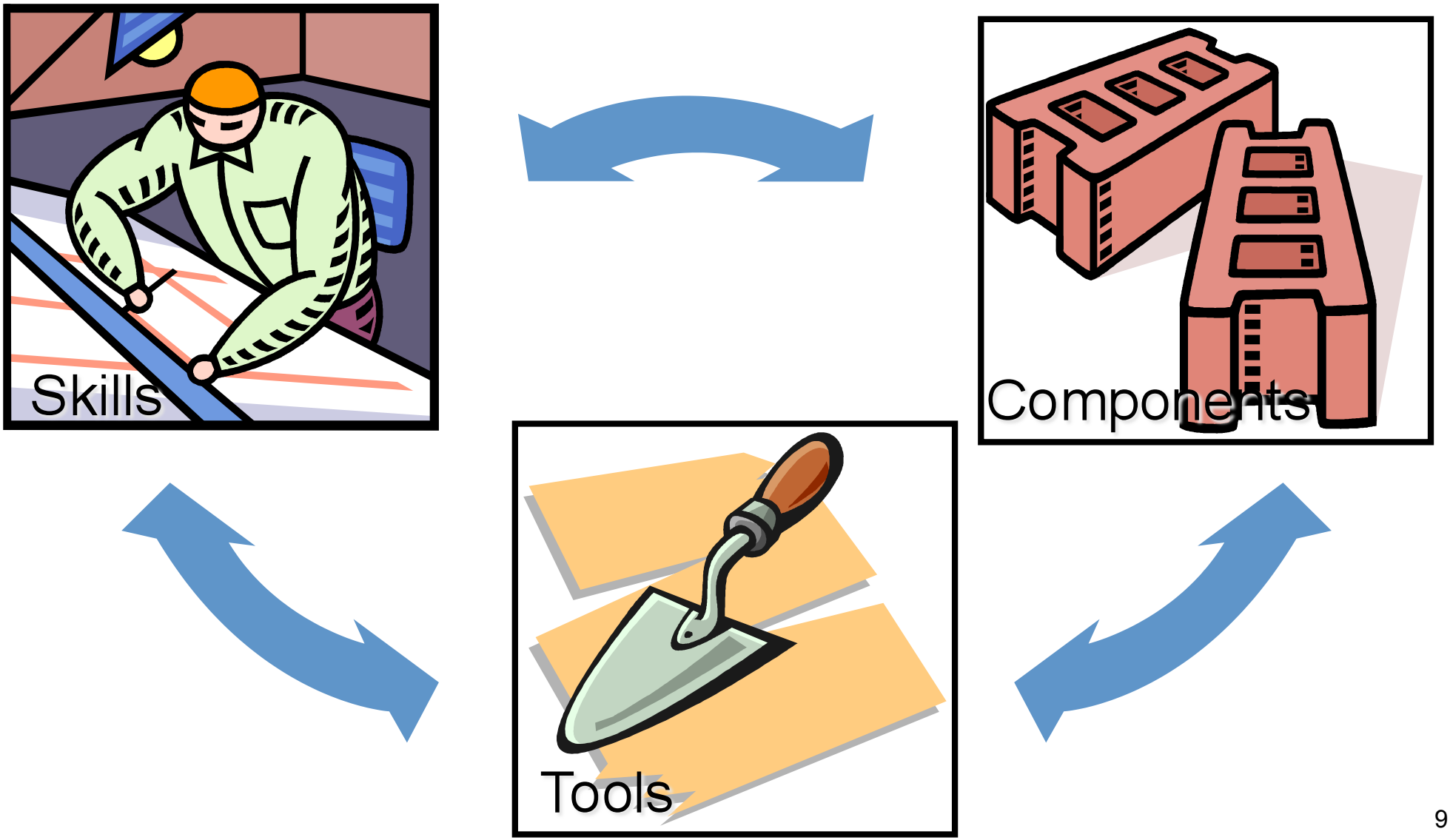
1. Marrying Physicality and Computation
   - theory and models encompassing continuous and discrete dynamic to predict the global behavior of a system interacting with its physical environment.

2. Component based Design
   - theory, models and tools for the cost-effective building of complex systems by assembling heterogeneous components

3. Adaptivity
   - systems must provide a service meeting given requirements in interaction with uncertain environments.

Rigorous System Design – Essential Properties: Productivity

Efficiency of the design process

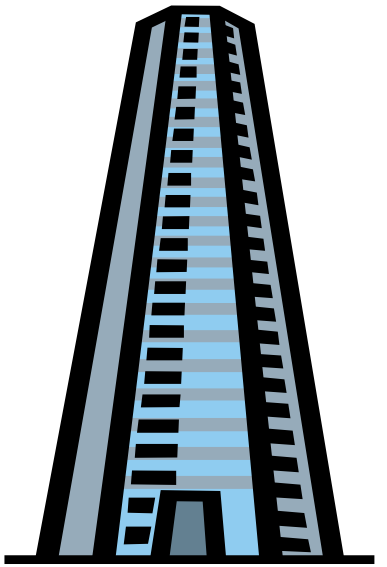


Rigorous System Design – Essential Properties: Performance

- Languages for describing feasible (correct) design solutions

- Optimal use of resources through design space exploration to resolve choices such as
  - reducing parallelism (through mapping on the same processor)
  - reducing non determinism (through scheduling)
  - fixing parameters (quality, frequency, voltage)



System Design - Essential Properties: Correctness

- Avoid design errors or eliminate them as early as possible
- Incremental construction and validation – scalability
- Traceability between application software and implementation

Our Approach

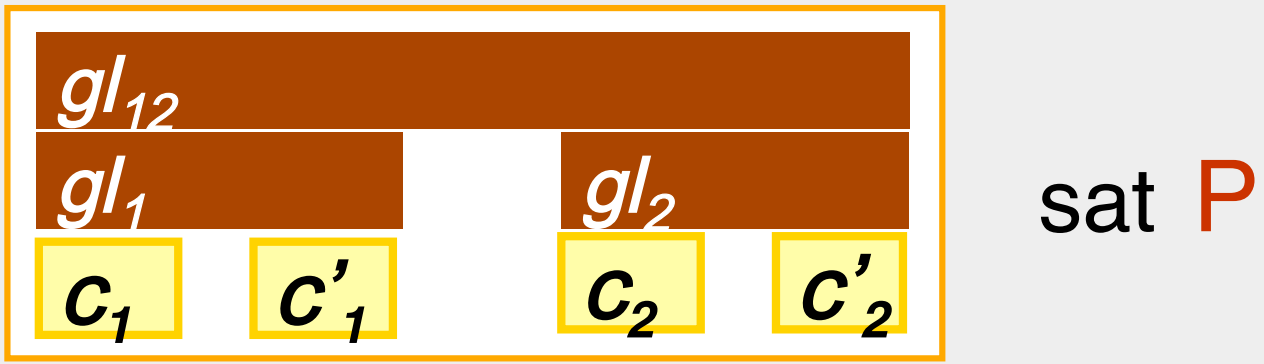
Develop the BIP framework:

- model-based and component-based design

Component-Based Construction: Formal Framework

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

- C_0 a set of atomic components modeling behavior
- *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

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

Our Approach

Develop the BIP framework:

- model-based and component-based design
- expressive enough to encompass heterogeneity of
  - execution: synchronous and asynchronous components
  - interaction: function call, broadcast, rendez-vous
  - abstraction levels: hardware, middleware, application software

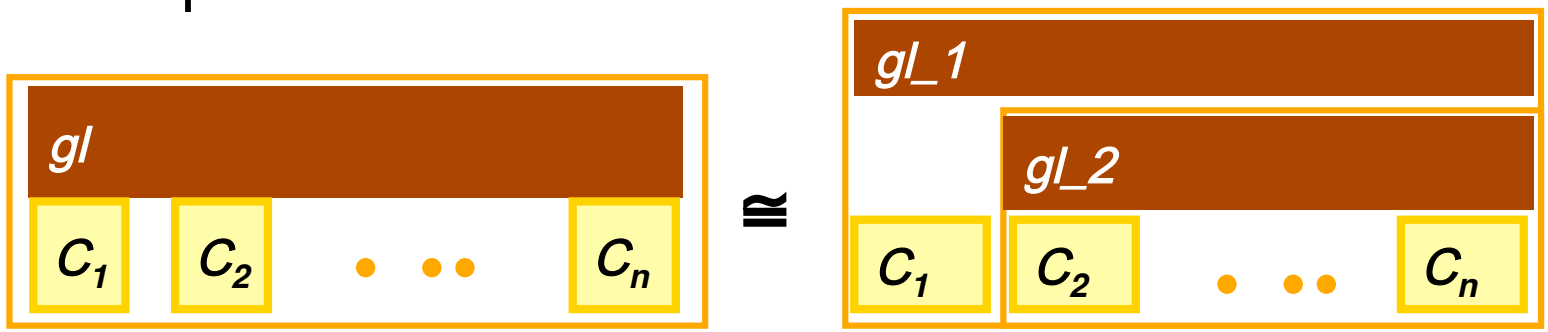
Our Approach

Develop the BIP framework:

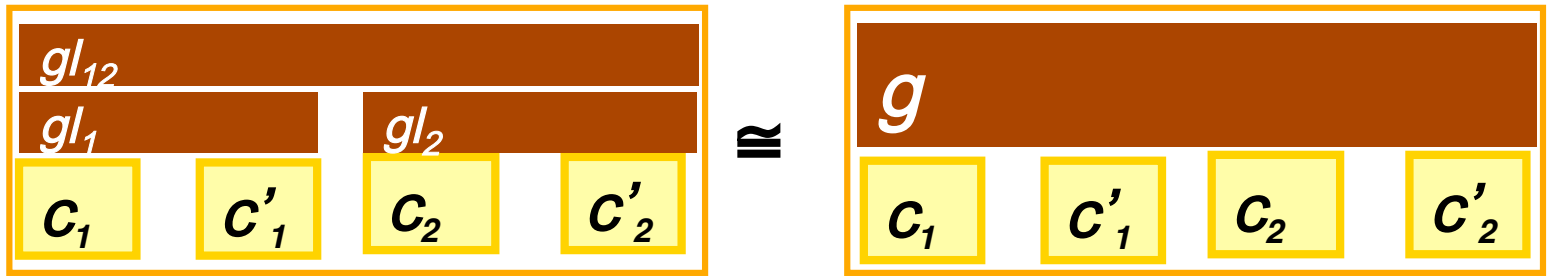
- model-based and component-based design
- expressive enough to encompass heterogeneity of
  - execution: synchronous and asynchronous components
  - interaction: function call, broadcast, rendez-vous
  - abstraction levels: hardware, middleware, application software
- using a minimal set of constructs and principles for guaranteeing correctness by construction.

Component-Based Construction: Incremental Description

1. Decomposition



2. Flattening



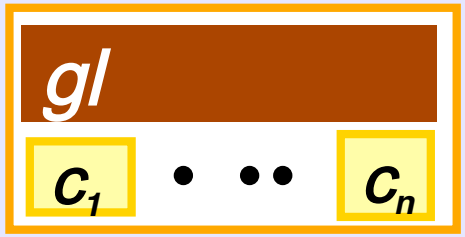
Flattening can be achieved by using a (partial) associative operation $\oplus$ on GL

Component-Based Construction: Constructivity – Compositionality

Building correct systems

from correct components

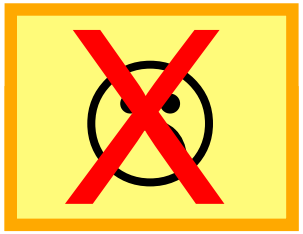
c_i sat P_i implies $\forall gl\ \exists \tilde{gl}$



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

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∧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

Our Approach

Develop the BIP component framework:

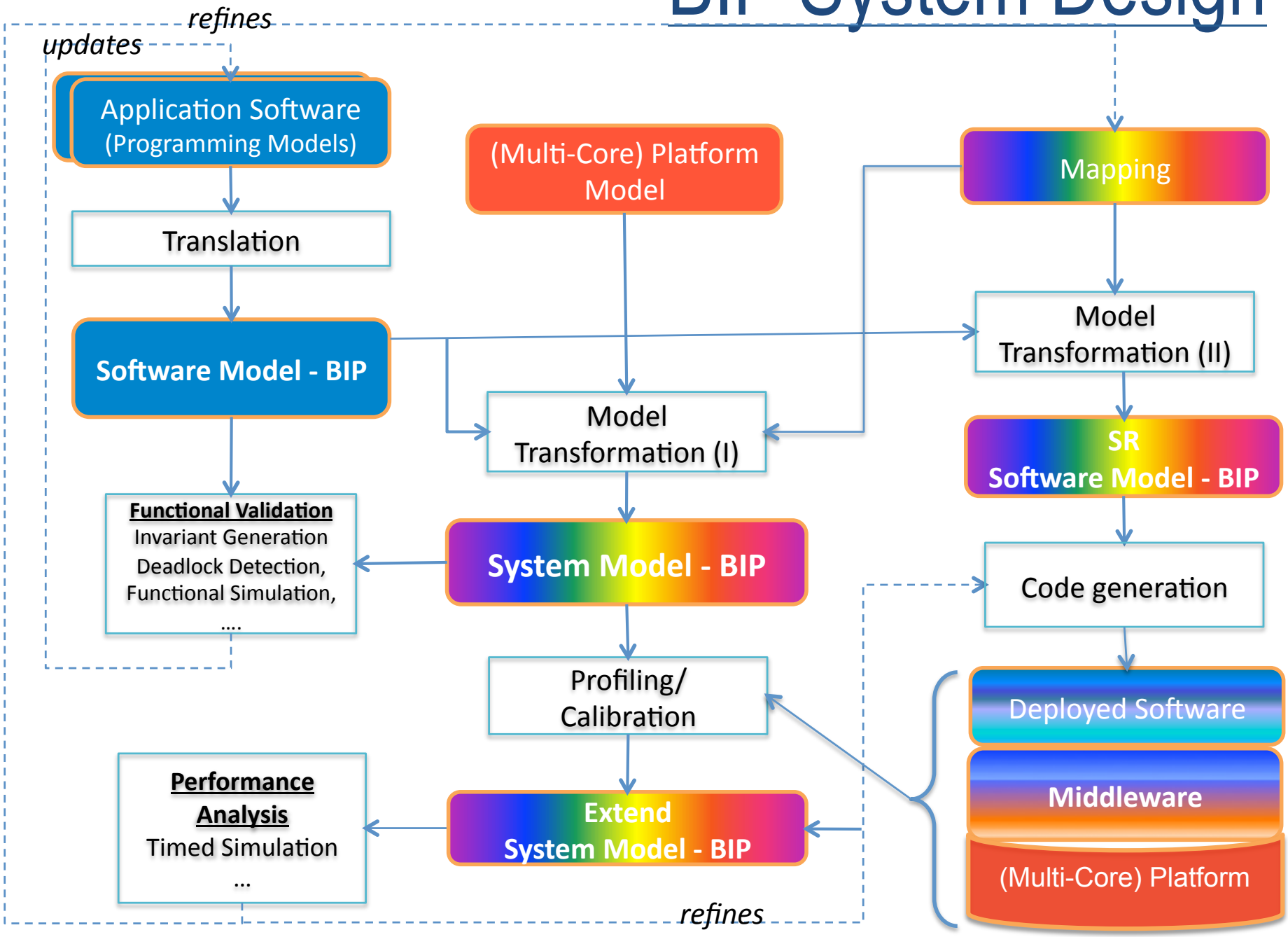
- model-based and component-based design
- expressive enough to encompass heterogeneity of
  - execution: synchronous and asynchronous components
  - interaction: function call, broadcast, rendez-vous
  - abstraction levels: hardware, middleware, application software
- using a minimal set of constructs and principles for guaranteeing correctness by construction.
- treating interaction and system architectures as first class entities that can be composed and analyzed independently of the behavior of individual components

Our Approach

Develop the BIP component framework:

- model-based and component-based design
- expressive enough to encompass heterogeneity of
  - execution: synchronous and asynchronous components
  - interaction: function call, broadcast, rendez-vous
  - abstraction levels: hardware, middleware, application software
- using a minimal set of constructs and principles for guaranteeing correctness by construction
- treating interaction and system architectures as first class entities that can be composed and analyzed independently of the behavior of individual components
- providing automated support for efficient implementation on given platforms
- providing automated support for validation and performance analysis

BIP System Design

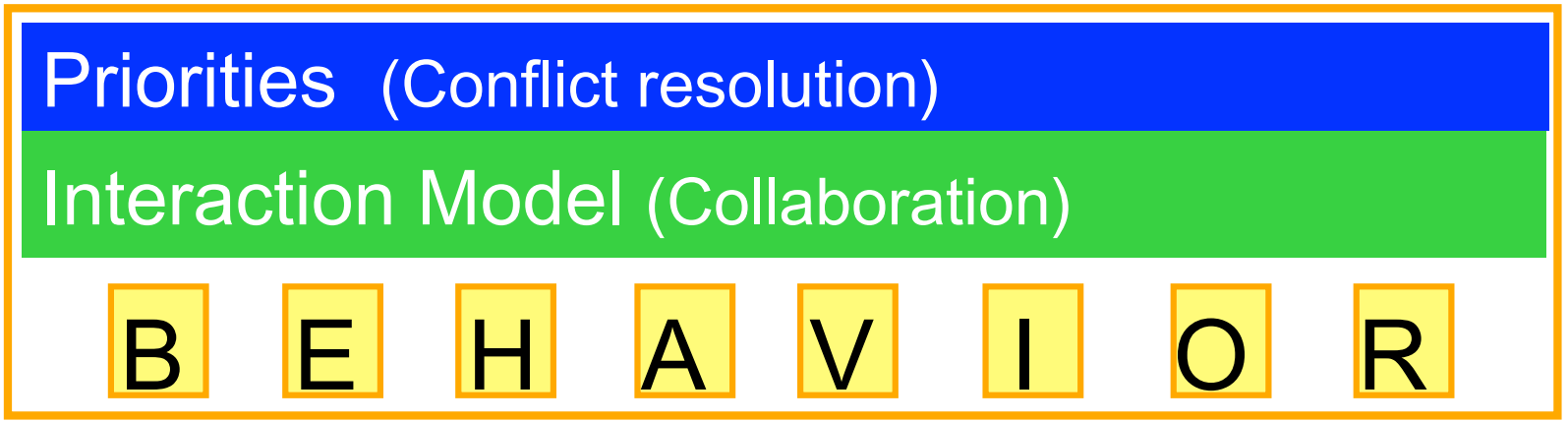


Outline

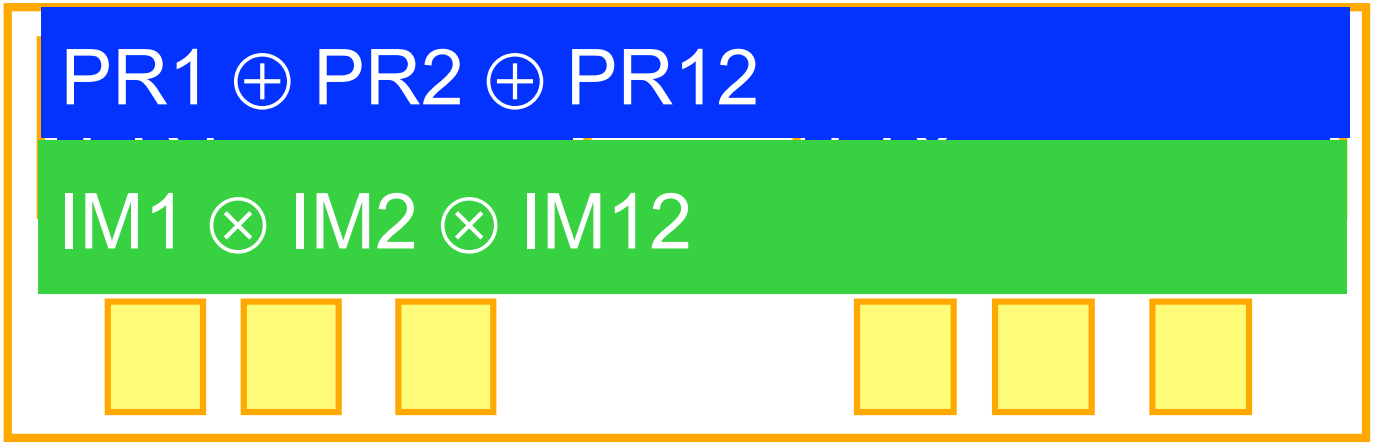
- Introduction
- The BIP Framework
  - Basic Concepts and Results
  - The BIP Language and the associated tools
- The Rigorous System Design Flow
- Discussion

Component-Based Construction: The BIP Framework

Layered component model



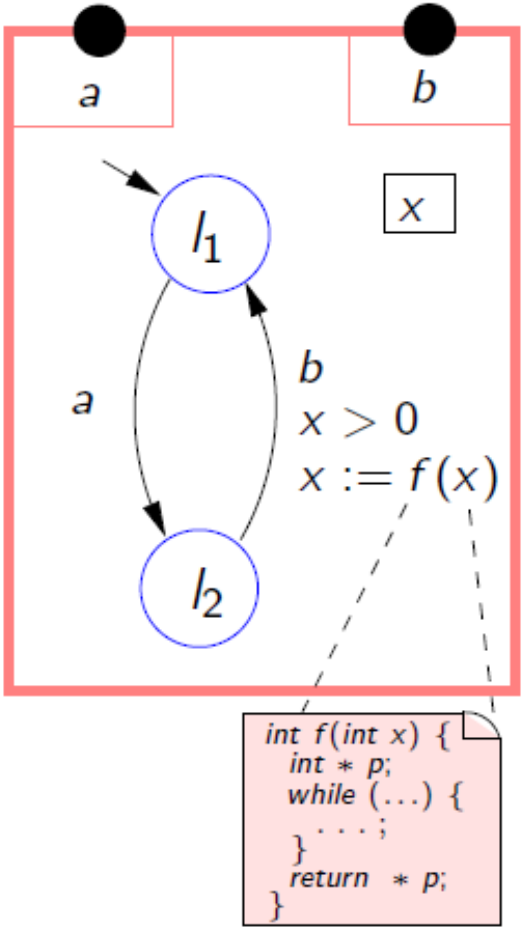
Composition (incremental description)

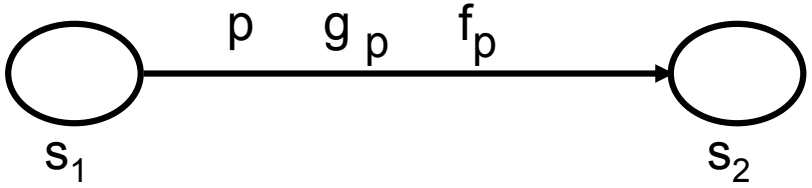


The BIP Framework: Behavior

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 C code)

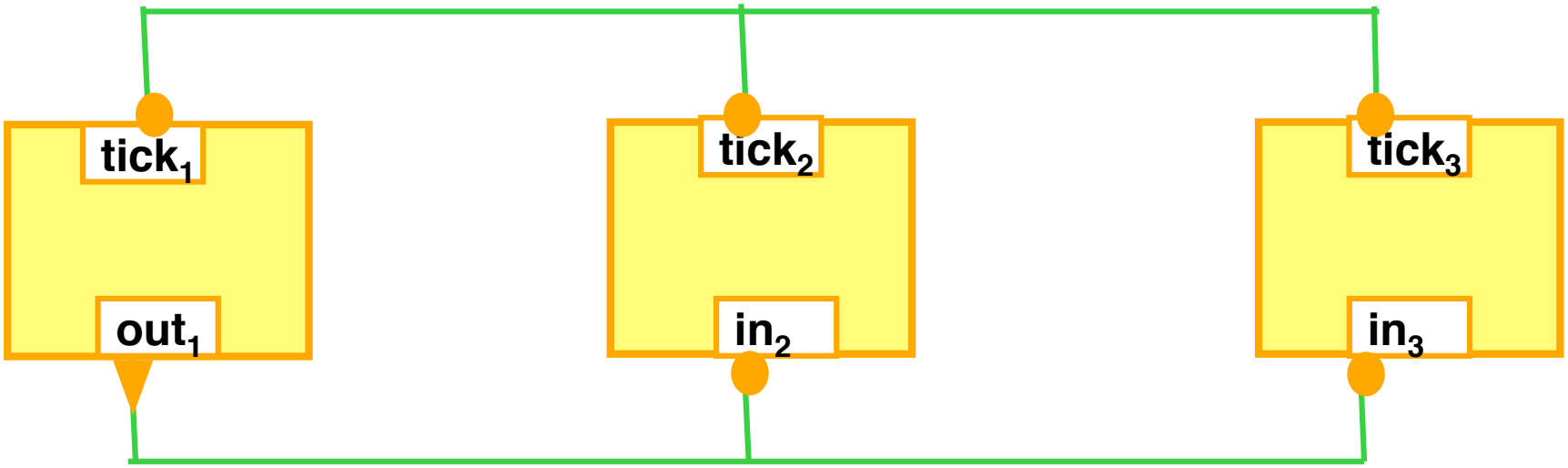




The BIP Framework: Interaction Model

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

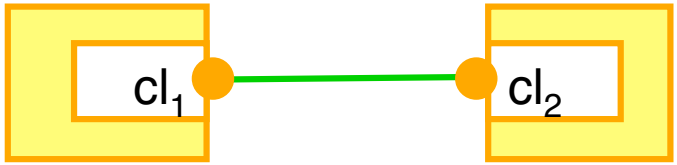
- Port attributes (***complete***, ***incomplete***) are used to distinguish between rendezvous and broadcast.
- An ***interaction*** of a connector is a set of ports such that: either it contains some complete port or it is maximal.



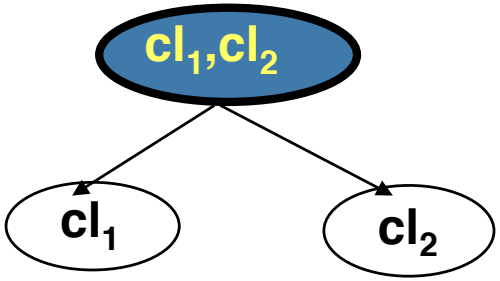
Interactions:

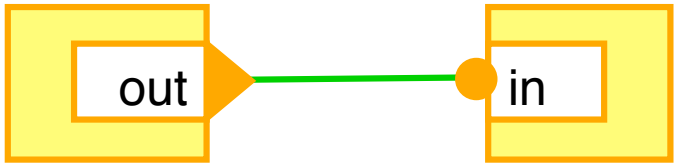
$\{tick_1, tick_2, tick_3\}$ $\{out_1\}$ $\{out_1, in_2\}$ $\{out_1, in_3\}$ $\{out_1, in_2, in_3\}$

The BIP Framework: Interaction Model

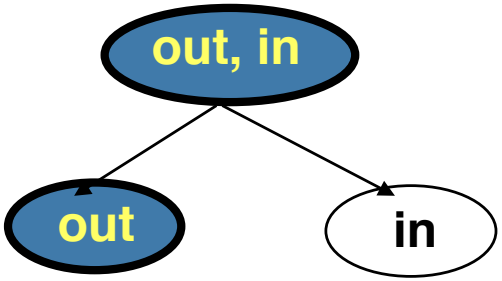


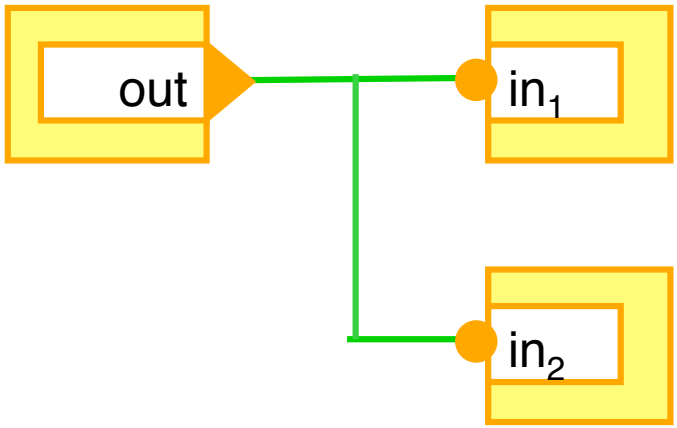
CN:{cl_1,cl_2}
CP: ∅



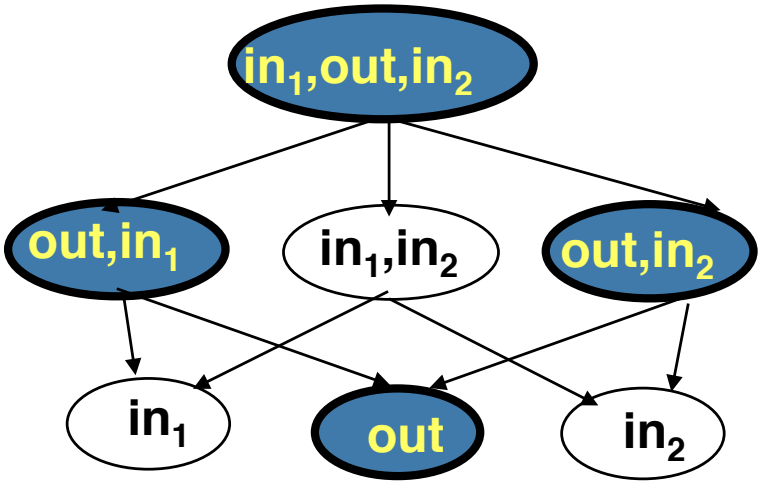


CN:{out,in}
CP: {out}



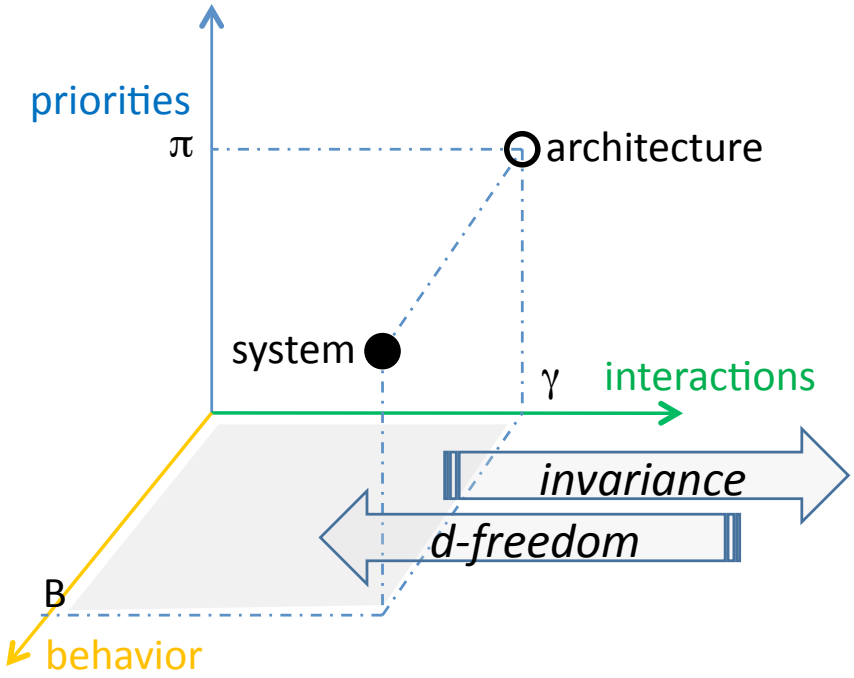


CN:{in_1,out,in_2}
CP: {out}



BIP Construction Space

- separation of concerns: between behavior and architecture (interaction and priority)
- semantic unification: heterogeneous components can be unified through transformation in the construction space
- correctness by construction: basis for study of preservation of properties under architecture or behavior transformations



Outline

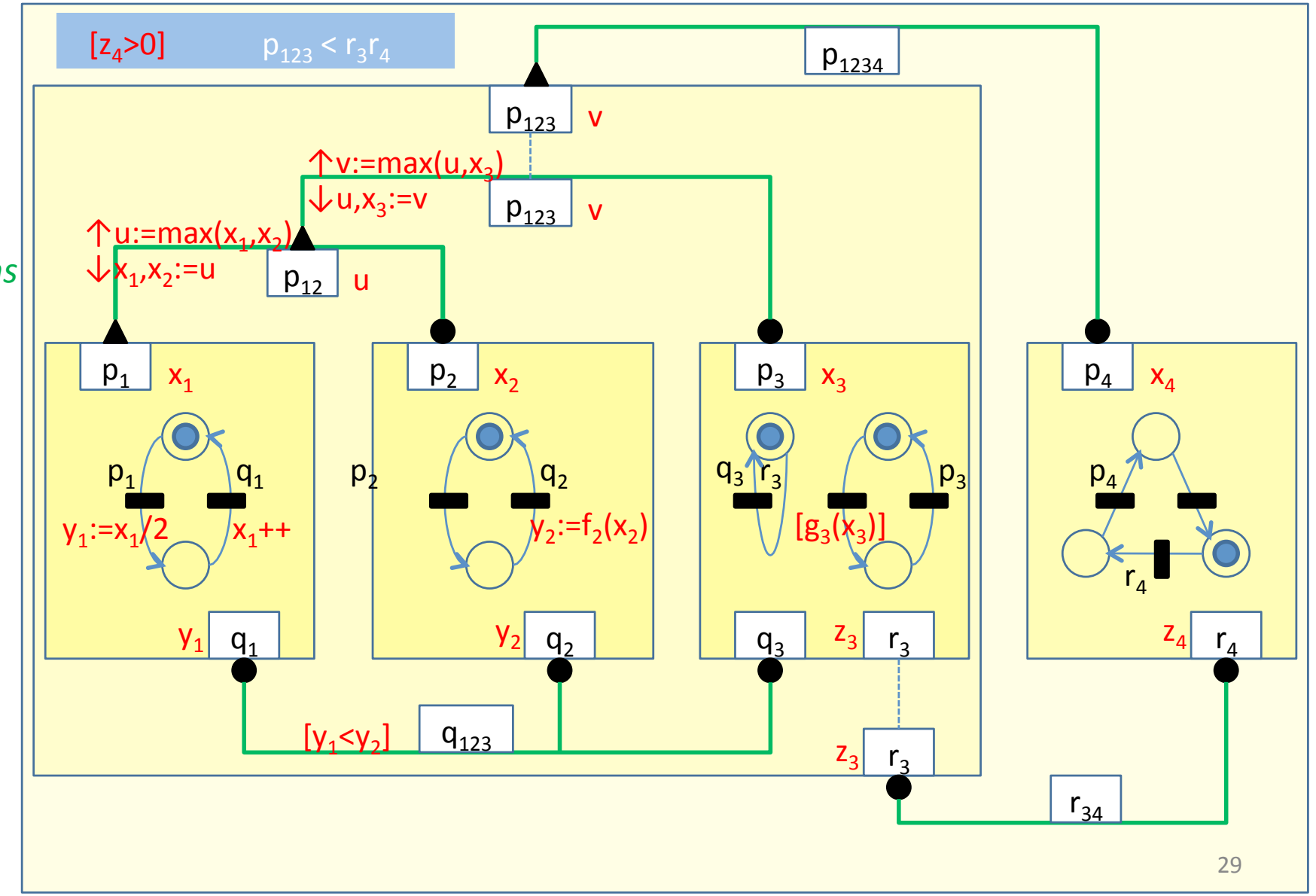
- Introduction
- The BIP Framework
  - Basic Concepts and Results
  - The BIP Language and the associated tools
- The Rigorous System Design Flow
- Discussion

The BIP Framework: An example

Priorities Glue

Interactions Glue

Behavior



The BIP Language

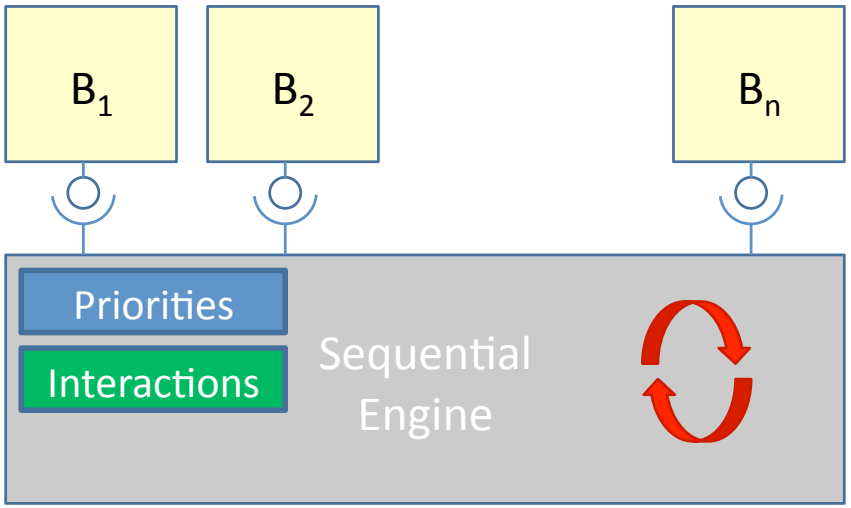
```
// atomic component definitio

atomic type Atom(int p, int q
  data int x, y, z, …
  data DataType u, v, w, …
  port MyPort p1(x)
  port TypePort2 p2(y, u)

  place s1, s2, s3, s4, …

  initial to s1
    do { /* initialization co
  on p1 from s1 to s2
     provided guard1
     do { /* transition code
  on p2 from s2 to s3
     provided x < y
     do { {# plain C code #}
  …
  export port MyPort p1 is r1

end
```

```
// connector type definition

connector type Bus (PortType1 p1
                    PortType2 p2
  define port-expression
  data int y
  …
  on interaction1 provided guard
     up { /*interaction code */
     down { /* interaction code
  …
  on p1 p2 provided p1.x > 0
    up {y = p1.x + p2.x }
    down { {# p1.x = p2.x = y; #
  …
  export port PortType  p0(y)

end
```

```
// compound component type definition

compound type Compo(int p, …)
  component CompType_1 c1(p, …)
  …
  component CompType_n cn

  connector ConType_1 x1( c1.p, … c2.q )
  …
  connector ConType_k xk( x1.p0, cn.r )

  priority prio1
     provided guard
     xi:interaction1 < xj:interaction2
  …
  export port PortType1 c1.p is p
  export port PortTypek xk.p0 is q
    …
end
```

Sequential Implementation

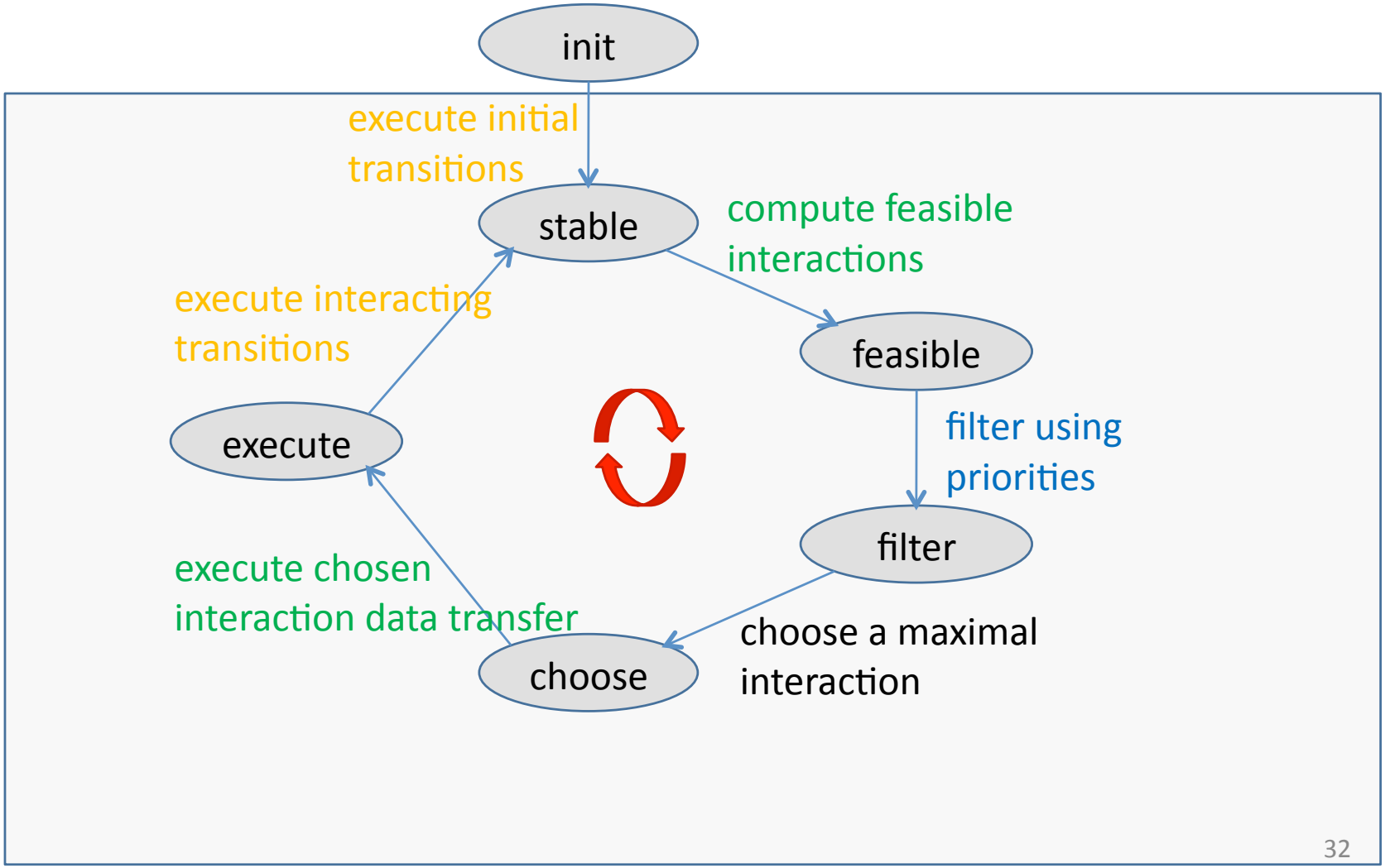
- The reference implementation for BIP models
- Separate compilation of component's code and coordination code



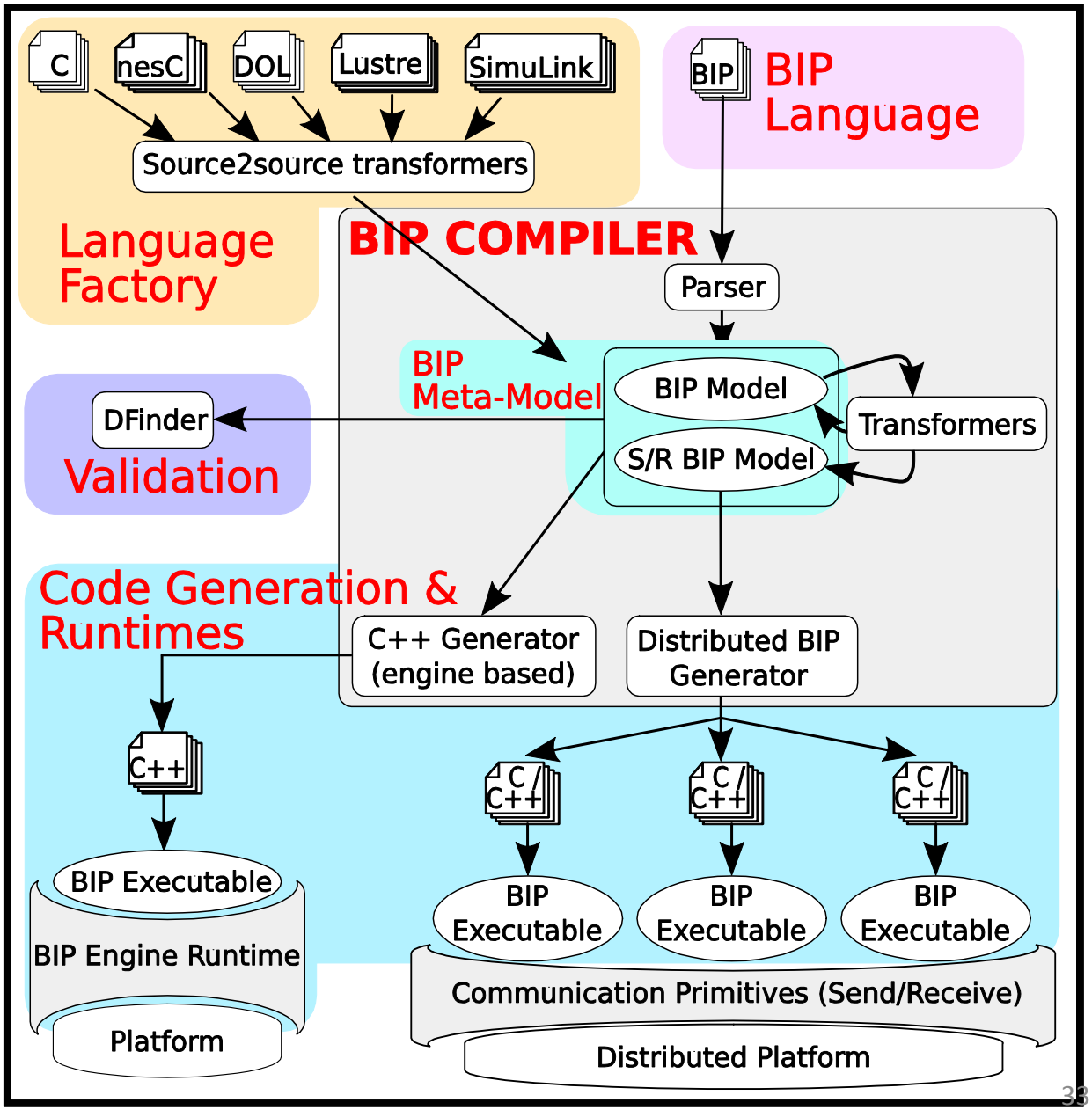
- The sequential engine runs one execution trace according to the BIP semantics
- The sequential engine provides extra-functionality for run-time verification and model-checking

Sequential Implementation

- Execution of the Engine



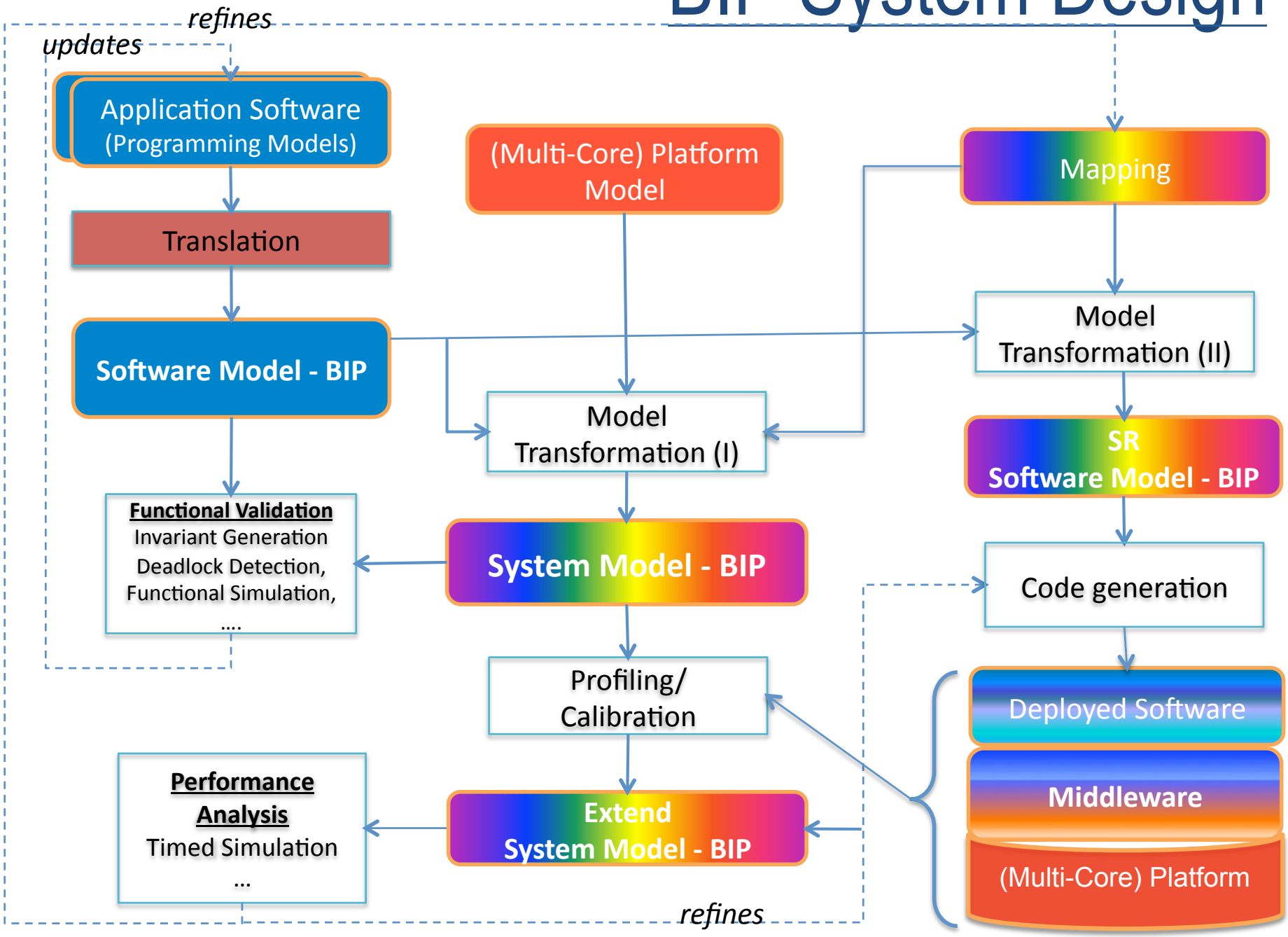
The BIP Toolbox



Outline

- Introduction
- The BIP Framework
  - Basic Concepts and Results
  - The BIP Language and the associated tools
- The Rigorous System Design Flow
- Discussion

BIP System Design



BIP Language Factory

- Use BIP as a unifying semantics model for various programming models

  Translations defined so far:
  - (Discretized) Timed and Hybrid Systems
  - Synchronous Systems (Lustre, MATLAB/Simulink, Scicos, ...)
  - Architecture Description Languages (AADL)
  - Domain Specific Languages and MoCs
    - Autonomous Robotic Applications (GeNoM)
    - Wireless Sensor Network Applications (TinyOs + nesC)
    - Process Networks in the Distributed Operation Layer (DOL)
  - ...

- Systematic approach based on two level translation into BIP:
  - structural translation of the language constructs, the programmers view
  - structural translation of the language operational semantics, the execution model view

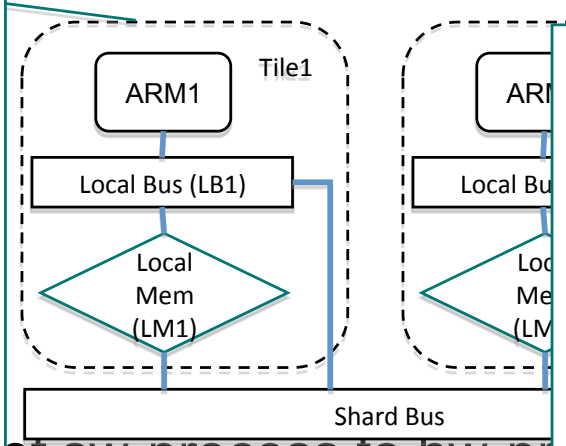
DOL (Distributed Operation Layer)

A framework from ETHZ for programming parallel applications and specifying their mapping onto multicore arch

```
<!-- arm core 1 -->
  <processor name="ARM1" type="RISC">
    <configuration name="memory" value="pm1"/>
  </processor>

  <memory name="sc_mem1" type="CACHE">
       <configuration name="cycles" value="1"/>
  </memory>

  <hw_channel name="lbus1" type="BUS">
    <configuration name="frequency"
value="200000000"/>
    <configuration name="bytespercycle"
value="1"/>
  </hw_channel>
…
<!-- distributed external memory -->
  <memory name="sh_mem" type="RAM">
       <configuration name="cycles" value="1"/>
       <!--configuration name="cycles" value="6"/
-->
  </memory>

   <!-- bus -->
  <hw_channel name="ahb" type="BUS">
    <configuration name="frequency"
value="100000000"/>
    <configuration name="bytespercycle"
value="4"/>
  </hw_channel>
…
```

rocess network of sw-proc
ML), process behavior (C f

Computation resources int
nication paths (XML descr
es communication between C



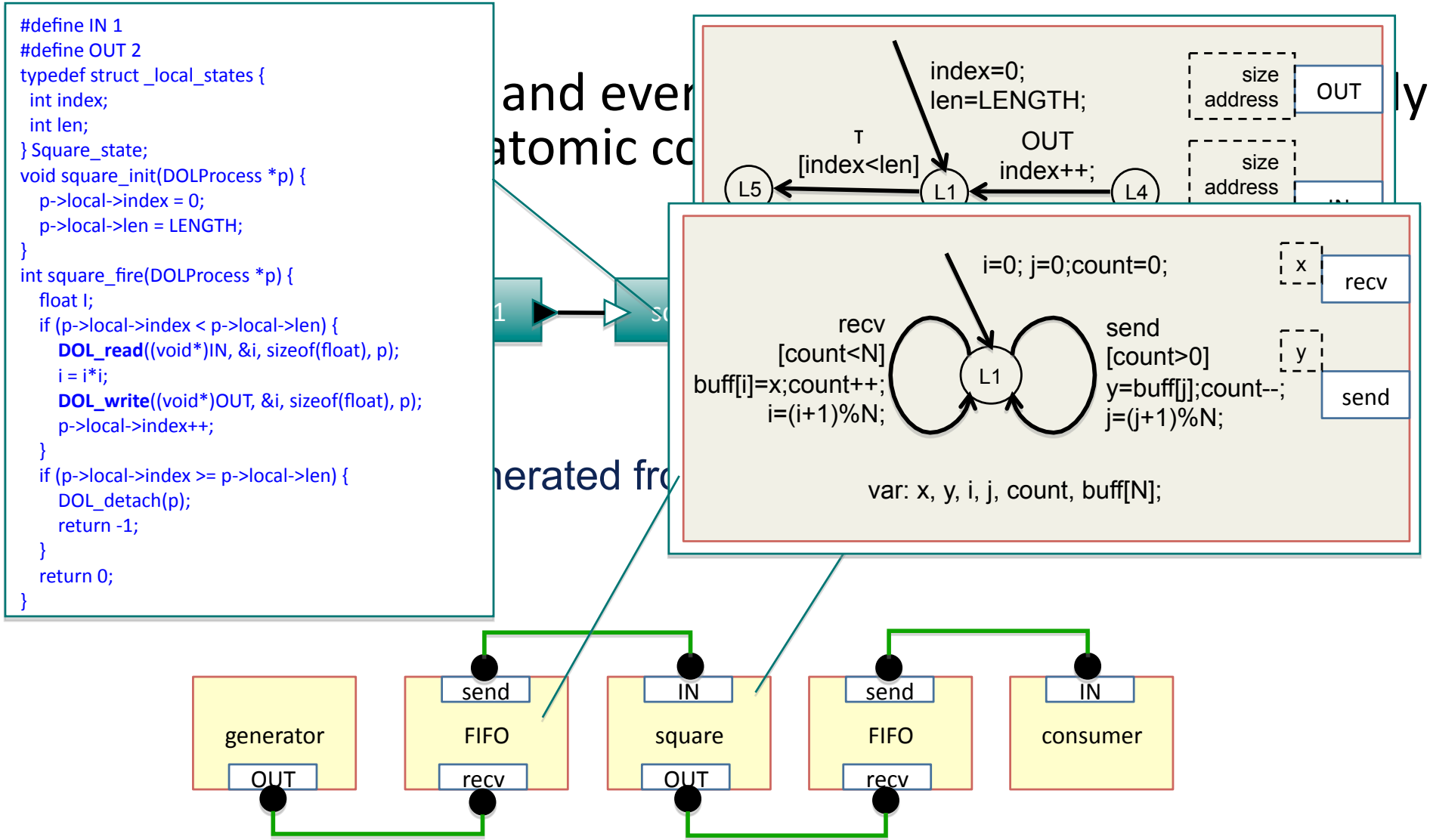
of sw-process to hw-pr
description)

```
…
<process name="square">
    <port type="input" name="1"/>
    <port type="output" name="2"/>
    <source type="c" location="square.c"/>
  </process>
…
  <!-- sw_channels -->
  <sw_channel type="fifo" size="10" name="C1">
    <port type="input" name="0"/>
    <port type="output" name="1"/>
  </sw_channel>
…
<connection name="g-c">
    <origin name="generator">
      <port name="1"/>
    </origin>
    <target name="C1">
```

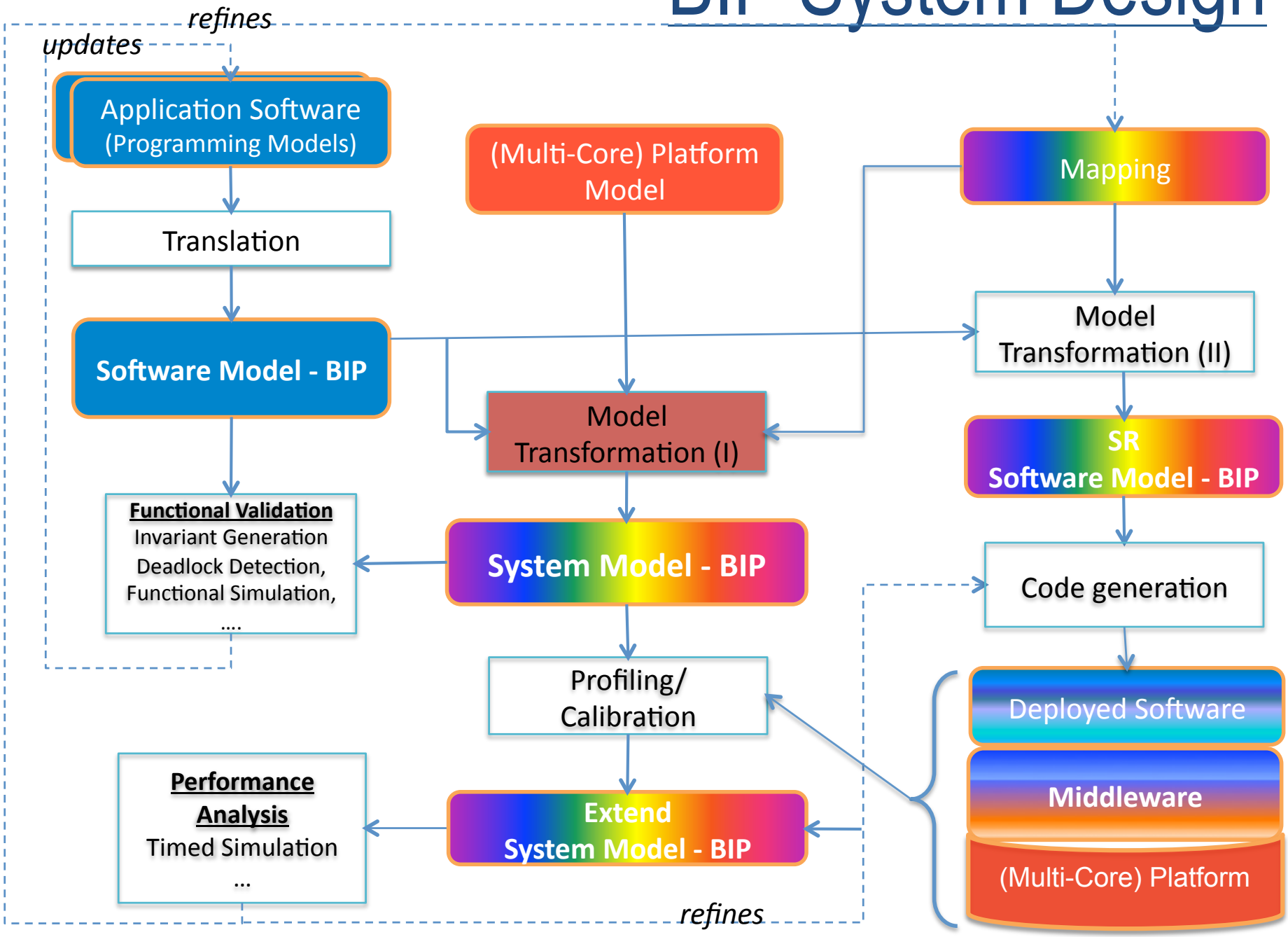
```
…
<!-- process bindings -->
  <binding name="binding_generator"
xsi:type="computation">
    <process name="generator"/>
    <processor name="ARM1"/>
  </binding>
…
<!-- channel bindings -->
  <binding name="binding_fifoC1"
xsi:type="communication">
    <sw_channel name="C1"/>
    <writepath name="pm1ahbsh_mem"/>
    <readpath name="sh_memahbpm2"/>
  </binding>
…
```

Generation of Application SW Model

```
#define IN 1
#define OUT 2
typedef struct _local_states {
  int index;
  int len;
} Square_state;
void square_init(DOLProcess *p) {
   p->local->index = 0;
   p->local->len = LENGTH;
}
int square_fire(DOLProcess *p) {
   float I;
   if (p->local->index < p->local->len) {
      DOL_read((void*)IN, &i, sizeof(float), p);
      i = i*i;
      DOL_write((void*)OUT, &i, sizeof(float), p);
      p->local->index++;
   }
   if (p->local->index >= p->local->len) {
      DOL_detach(p);
      return -1;
   }
   return 0;
}
```

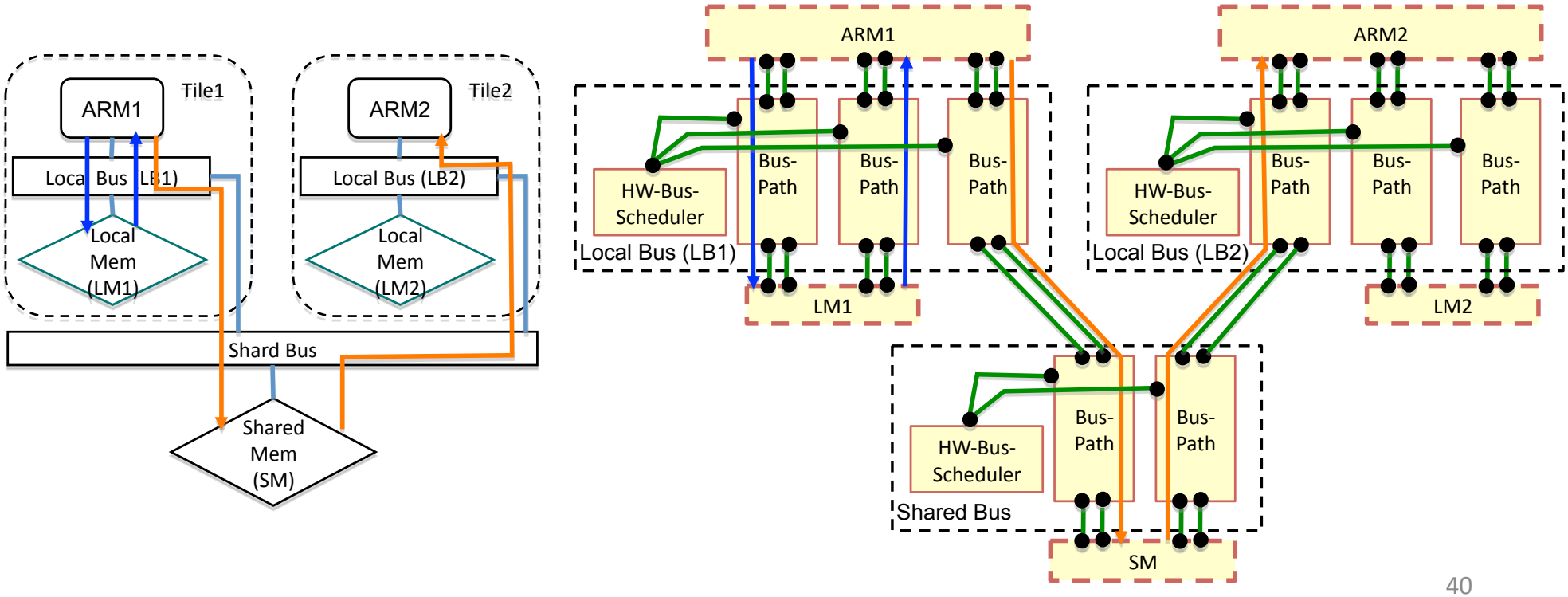


BIP System Design



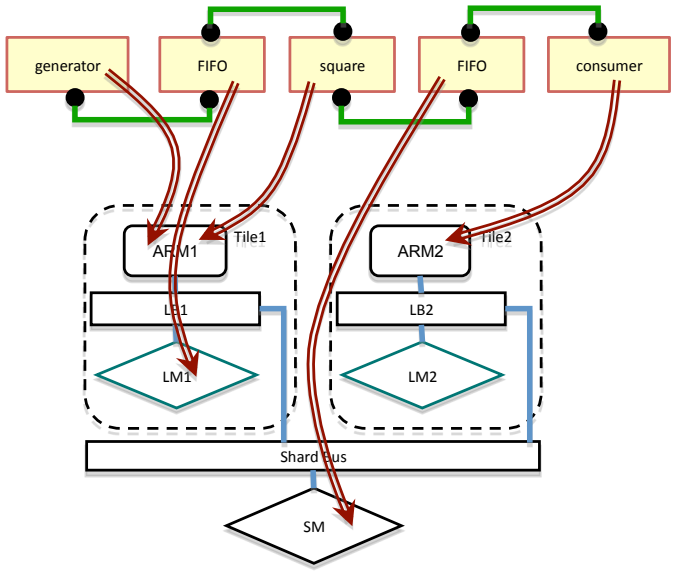
Construction of HW Template

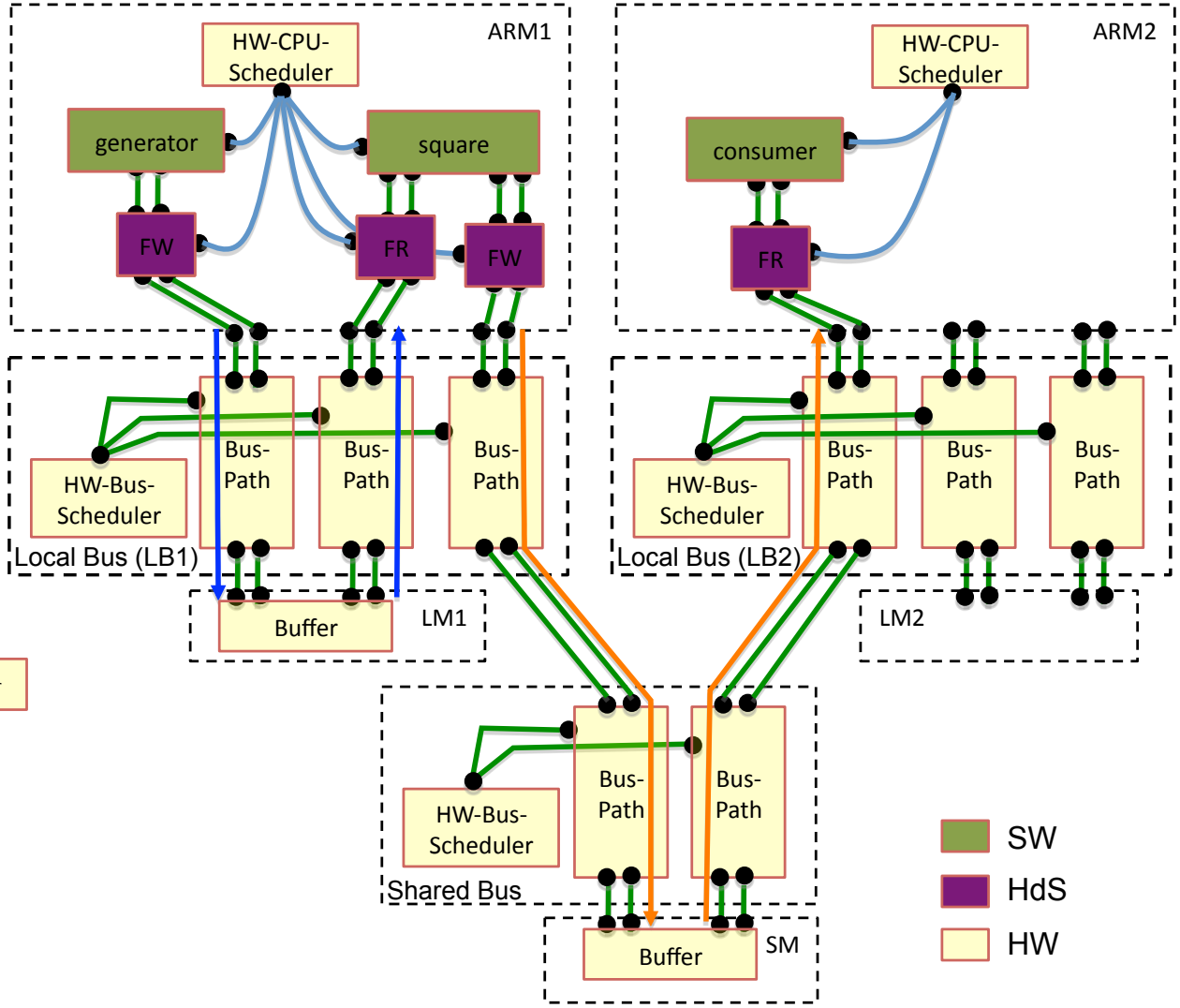
- Collection of HW-processor, memory and bus components connected as defined in the architecture
  - HW-processor and memory are placeholder
  - uses HW component library



Mapping: Fill up the HW Templates

- Transformation on sw model:
  - Splitting FIFO channel
  - Breaking atomic read/write
  - Adding interactions with CPU-Scheduler
  - FIFO buffers mapped to memory
- Transformations fully preserve functional behavior
- Uses HdS component library

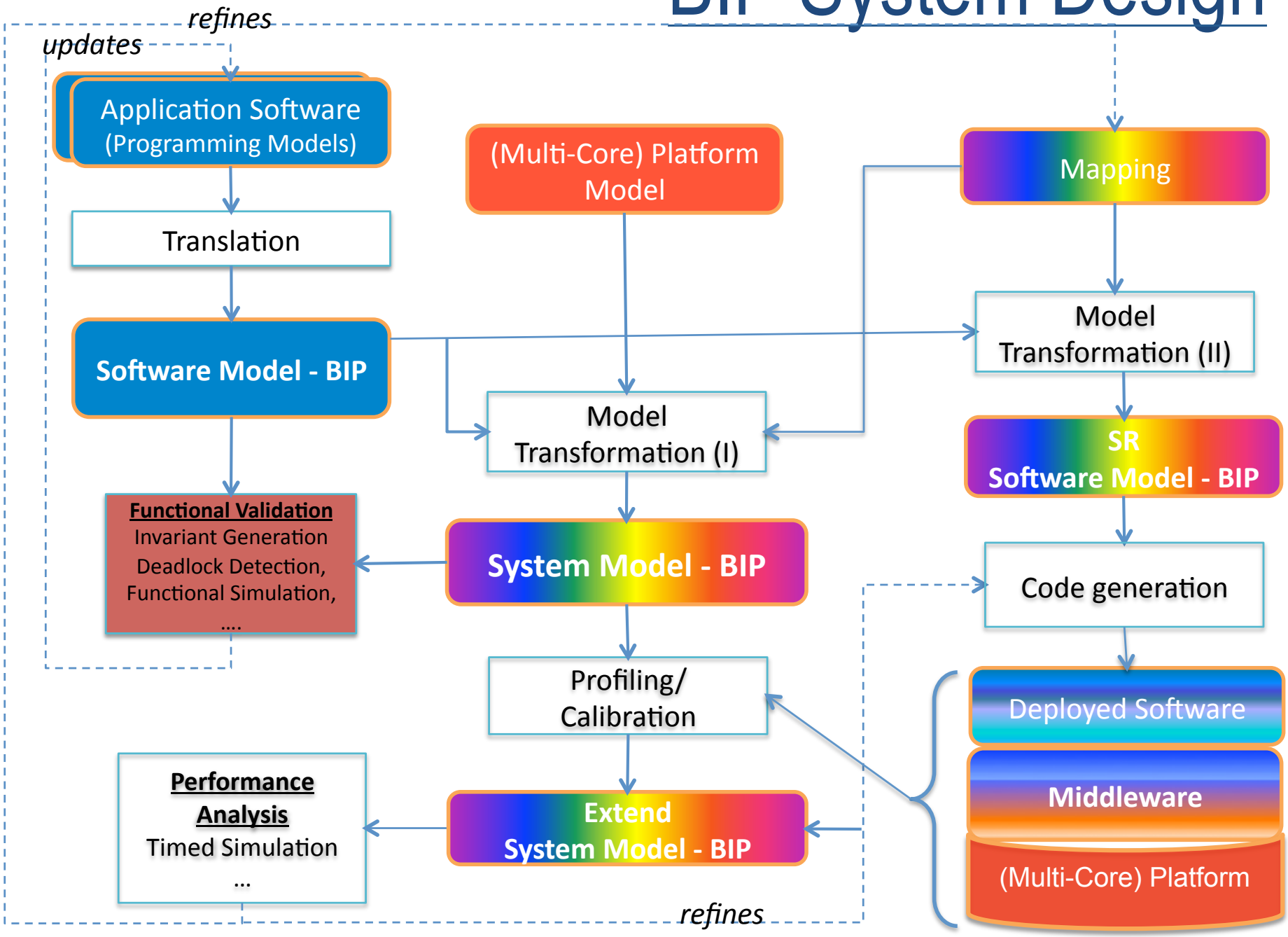




System Model Construction

- System model generated by applying a fixed number of transformations on the software model
  - splitting software channels
  - breaking atomicity of read/write operations in processes
  - inserting HdS components
  - …
- Transformations fully preserve functional behavior
  - ensure correctness-by-construction !
- No deadlocks are introduced
- Using a given set of BIP library components (characterized by the HW architecture, OS)

BIP System Design



Compositional Verification

- Compositional rule for proving state invariants:

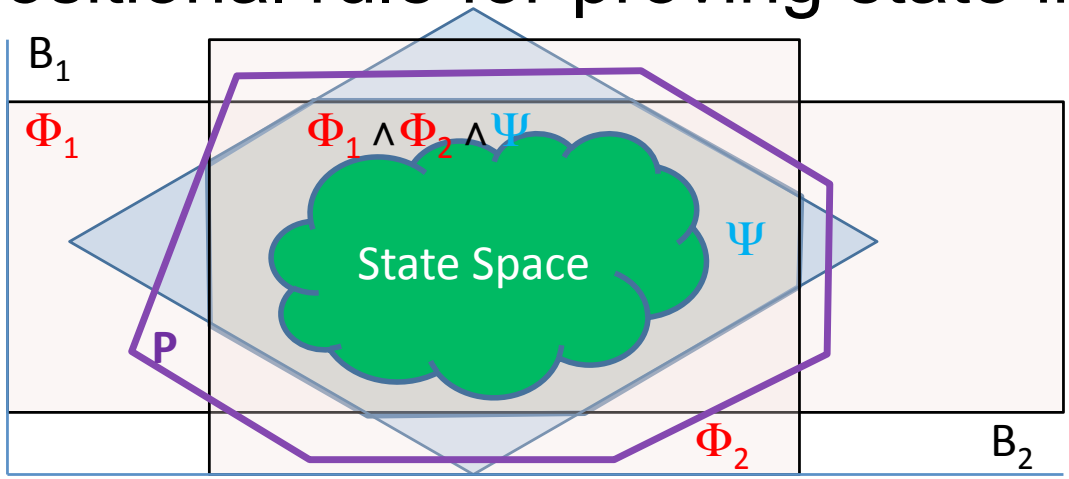
$$\frac{(B_i \models \Box \Phi_i)_{i=1,n} \quad \Psi \in \mathit{II}(\gamma(B_1, \ldots, B_n), \Phi_1, \ldots, \Phi_n) \quad \wedge_{i=1,n} \Phi_i \wedge \Psi \Rightarrow P}{\gamma(B_1, \ldots, B_n) \models \Box P}$$

- Combine two categories of particular invariants:
  - Component Invariants $(\Phi_i)_{i=1,n}$
  - Interaction Invariants Ψ

  automatically generated from BIP models !

Compositional Verification

- Compositional rule for proving state invariants:



- Combine two categories of particular invariants:
  - Component Invariants $(\Phi_i)_{i=1,n}$
  - Interaction Invariants Ψ

  automatically generated from BIP models !

Compositional Verification

- Compositional rule for proving safety properties:

$$\frac{(B_i \models \Box \Phi_i)_{i=1,n} \quad \Psi \in /\!/(\gamma(B_1, \ldots, B_n), \Phi_1, \ldots, \Phi_n) \quad \wedge_{i=1,n} \Phi_i \wedge \Psi \Rightarrow P}{\gamma(B_1, \ldots, B_n) \models \Box P}$$

Component Invariants:

- over-approximations of the set of reachable states of atomic components
- computed using static analysis of behavior

Compositional Verification

- Compositional rule for proving safety properties:

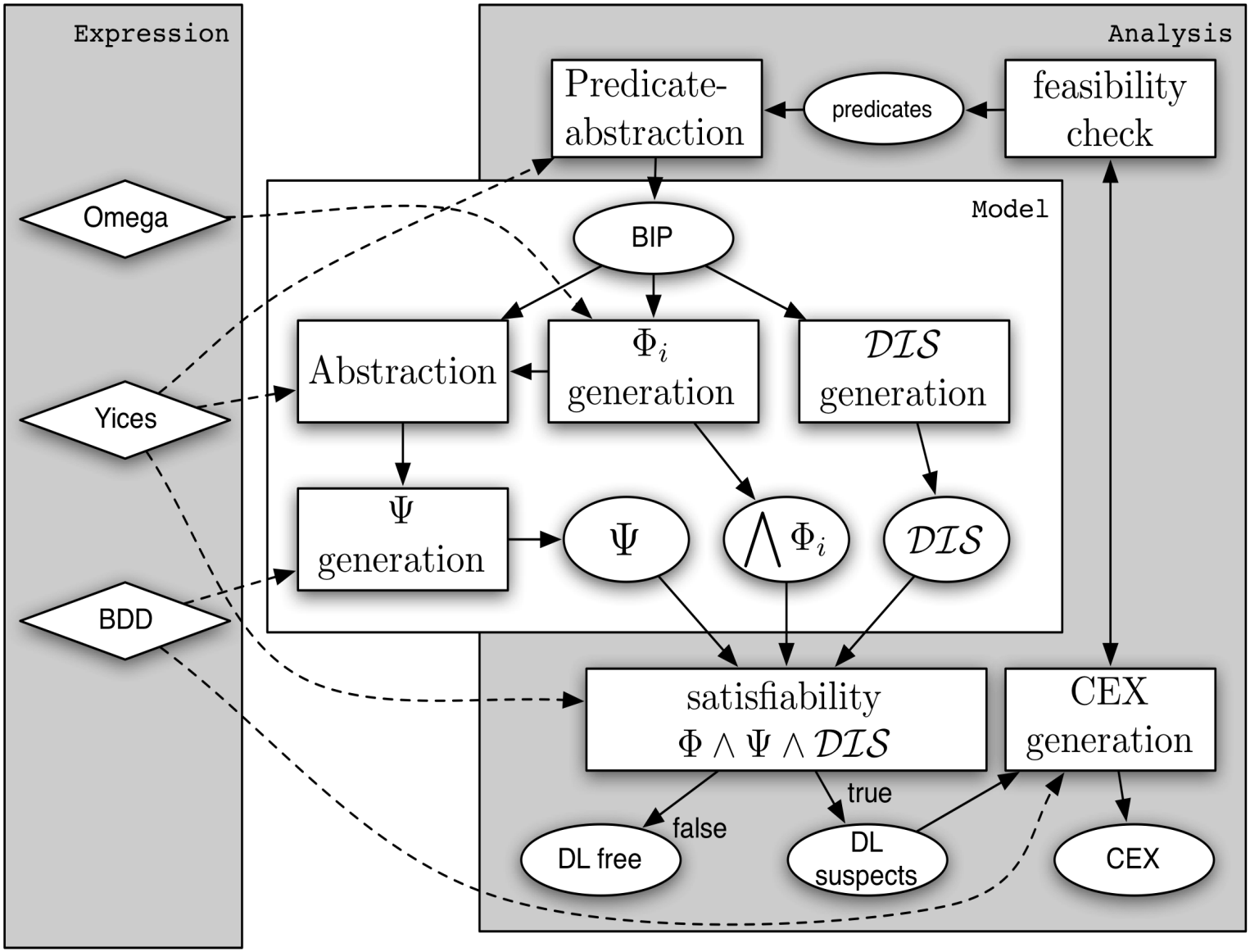
$$\frac{(B_i \models \Box \Phi_i)_{i=1,n} \quad \Psi \in /\!/(\gamma(B_1, \ldots, B_n), \Phi_1, \ldots, \Phi_n) \quad \wedge_{i=1,n} \Phi_i \wedge \Psi \Rightarrow P}{\gamma(B_1, \ldots, B_n) \models \Box P}$$

Interaction Invariants:

- characterize constraints on the global state space induced by synchronizations between components.

- computed by static analysis of interaction structures

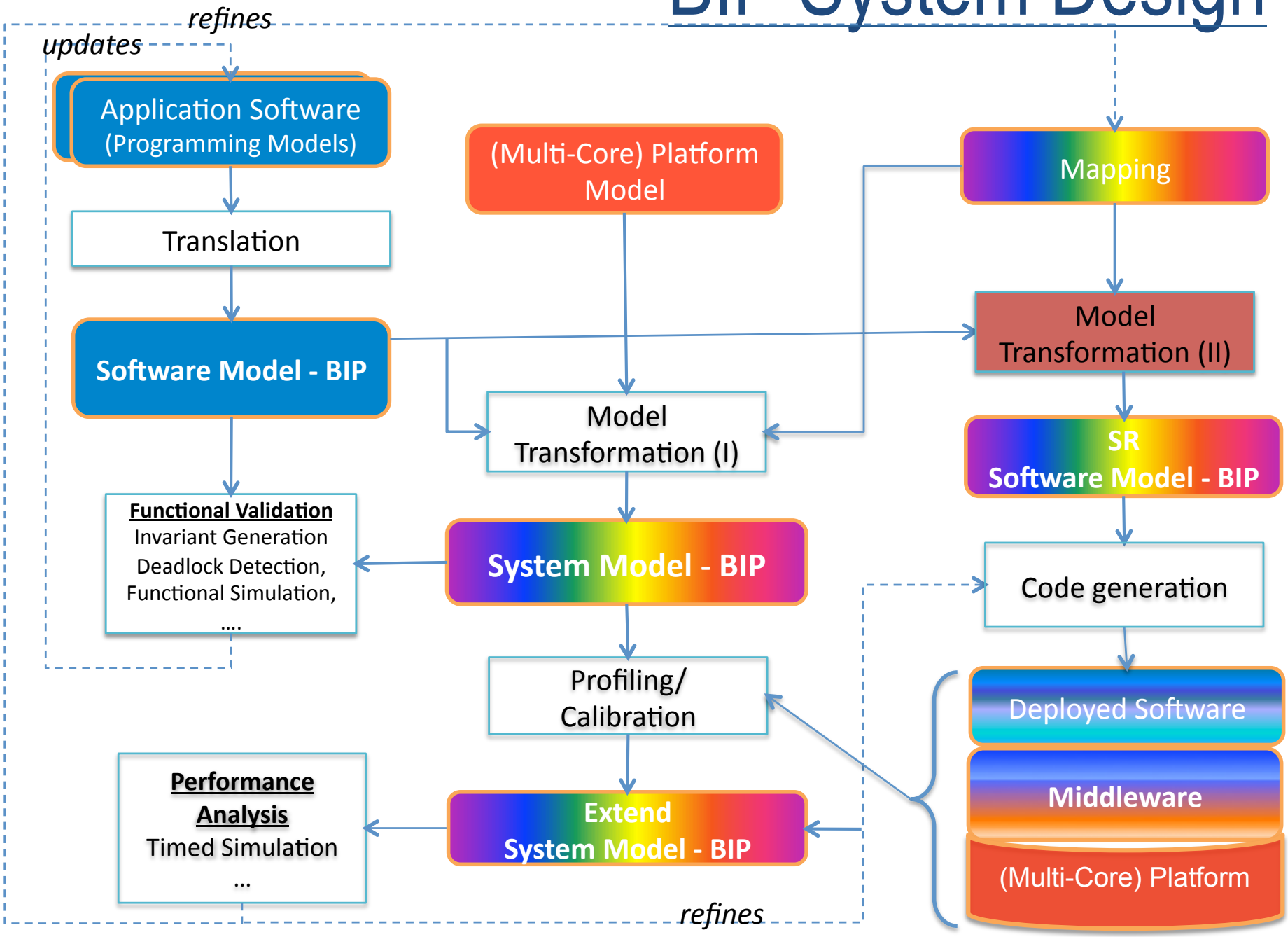
Implementation: D-Finder



Results on deadlock-freedom checking of all the modules

Modules	Components	Locations	Interactions	States	LOC	Minutes
LaserRF	43	213	202	$2^{20}x3^{29}x34$	4353	1:22
Aspect	29	160	117	$2^{17}x3^{23}$	3029	0:39
NDD	27	152	117	$2^{22}x3^{14}x5$	4013	8:16
RFLEX	56	308	227	$2^{34}x3^{35}x1045$	8244	9:39
Antenna	20	97	73	$2^{12}x3^{9}x13$	1645	0:14
Battery	30	176	138	$2^{22}x3^{17}x5$	3898	0:26
Heating	26	149	116	$2^{17}x3^{14}x145$	2453	0:17
PTU	37	174	151	$2^{19}x3^{22}x35$	8669	0:59
Hueblob	28	187	156	$2^{12}x3^{10}x35$	3170	5:42
VIAM	41	227	231	$2^{10}x3^{6}x665$	5099	4:14
DTM	34	198	201	$2^{28}x3^{20}x95$	4160	13:42
Stereo	33	196	199	$2^{27}x3^{20}x95$	3591	13:20
P3D	50	254	219	$2^{13}x3^{5}x5^{4}x629$	6322	3:51
LaserRF+Aspect+NDD	97	523	438	$2^{58}x3^{66}x85$	11395	40:57
NDD+RFLEX	82	459	344	$2^{56}x3^{49}x5^{2}x209$	12257	73:43

BIP System Design



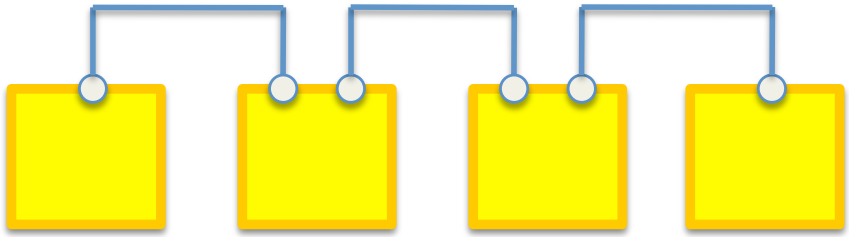
Distributed Implementation

Requirements

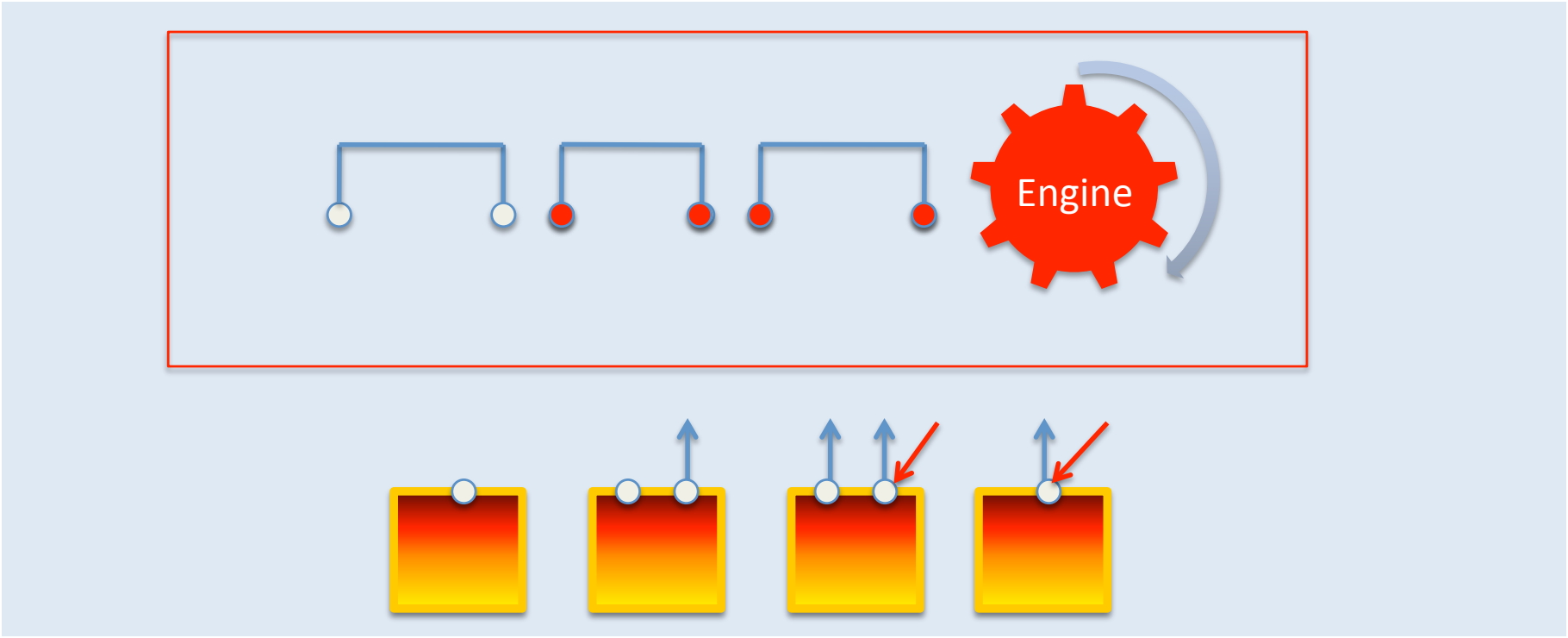
- produce efficient decentralized execution models
- allow for concurrent execution of interactions and internal computation of components

- collection of atomic processes/threads intrinsically concurrent – no global state
- point-to-point communication by asynchronous message passing
- ensure correctness-by-construction, that is, the initial model is equivalent to the implementation

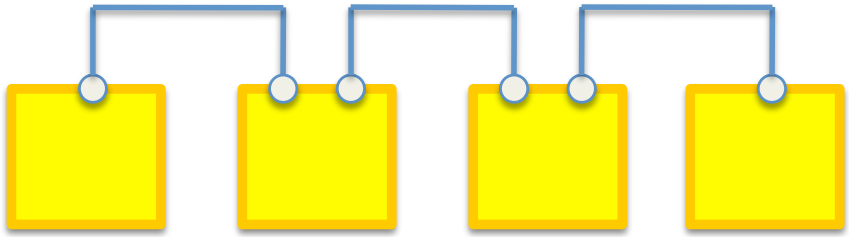
Centralized Implementation



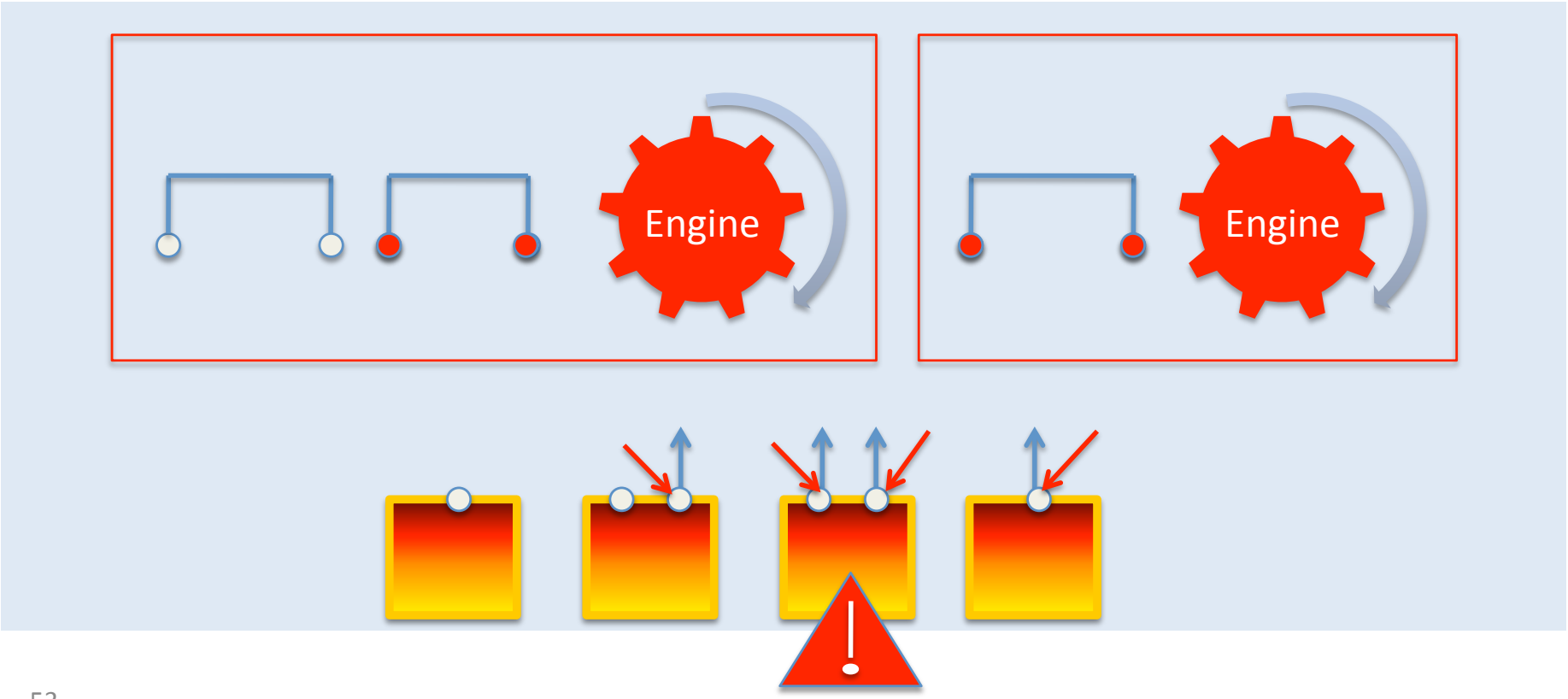
Centralized Implementation: one Engine play *all* interactions!



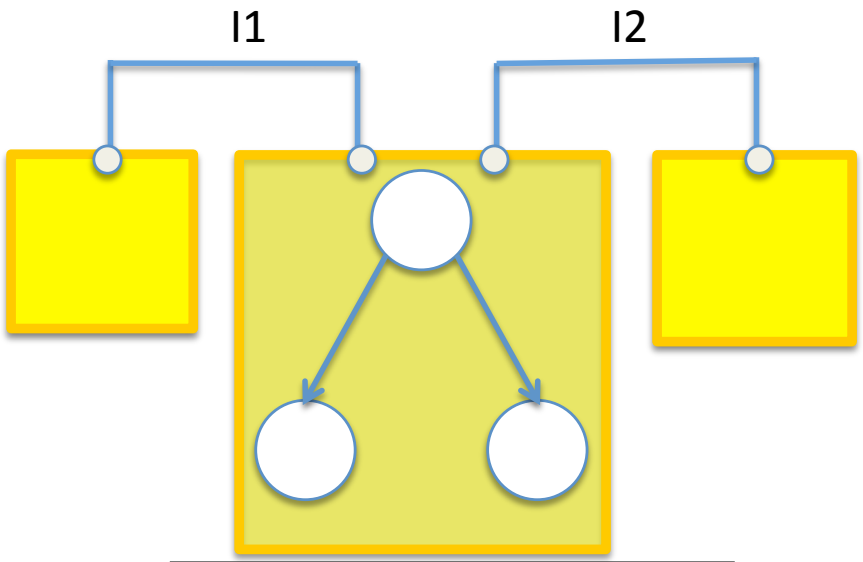
Decentralized Implementation



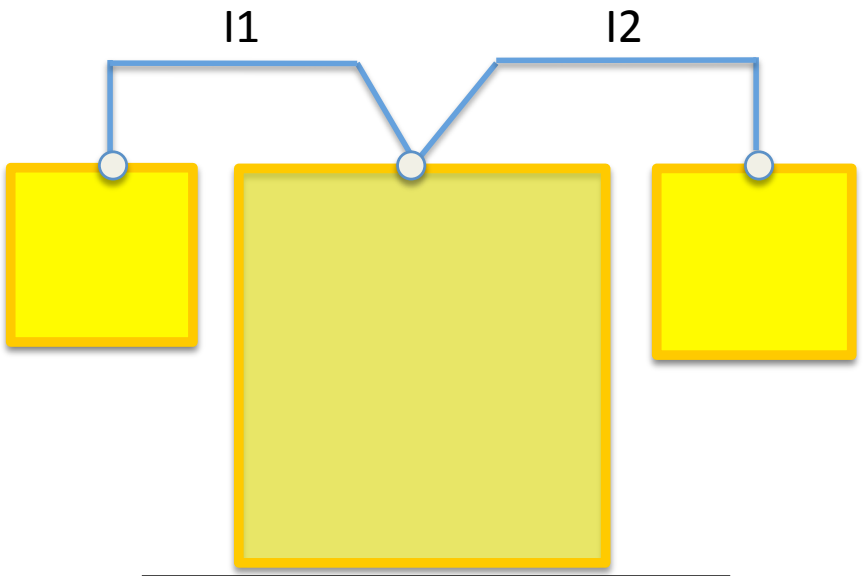
Decentralized Implementation: dispatch interactions across *multiple* engines!



Conflicting Interactions



I1 and I2 are using both sides ports of a choice in a component

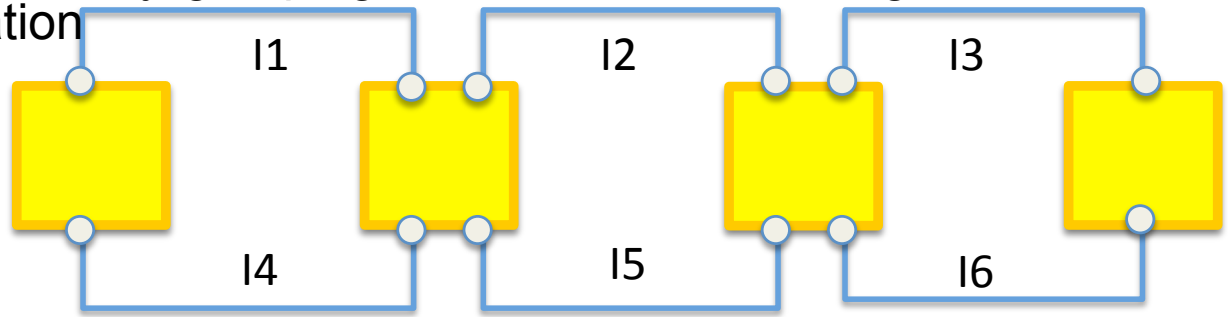


I1 and I2 share a common port

I1 and I2 are conflicting (I1 # I2)

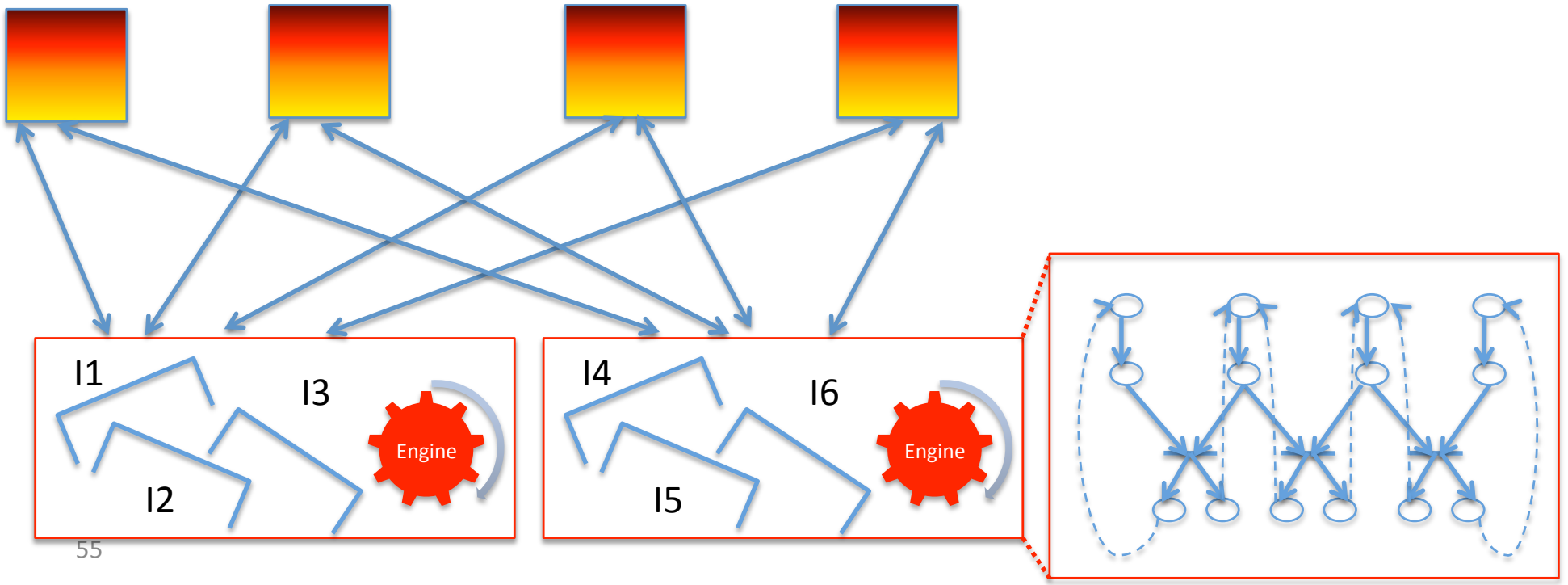
1st sol: Conflict-Free Distributed Engines

- Distributed Engines Conflict-Free by Construction, by grouping interactions according to the transitive closure of the conflict relation

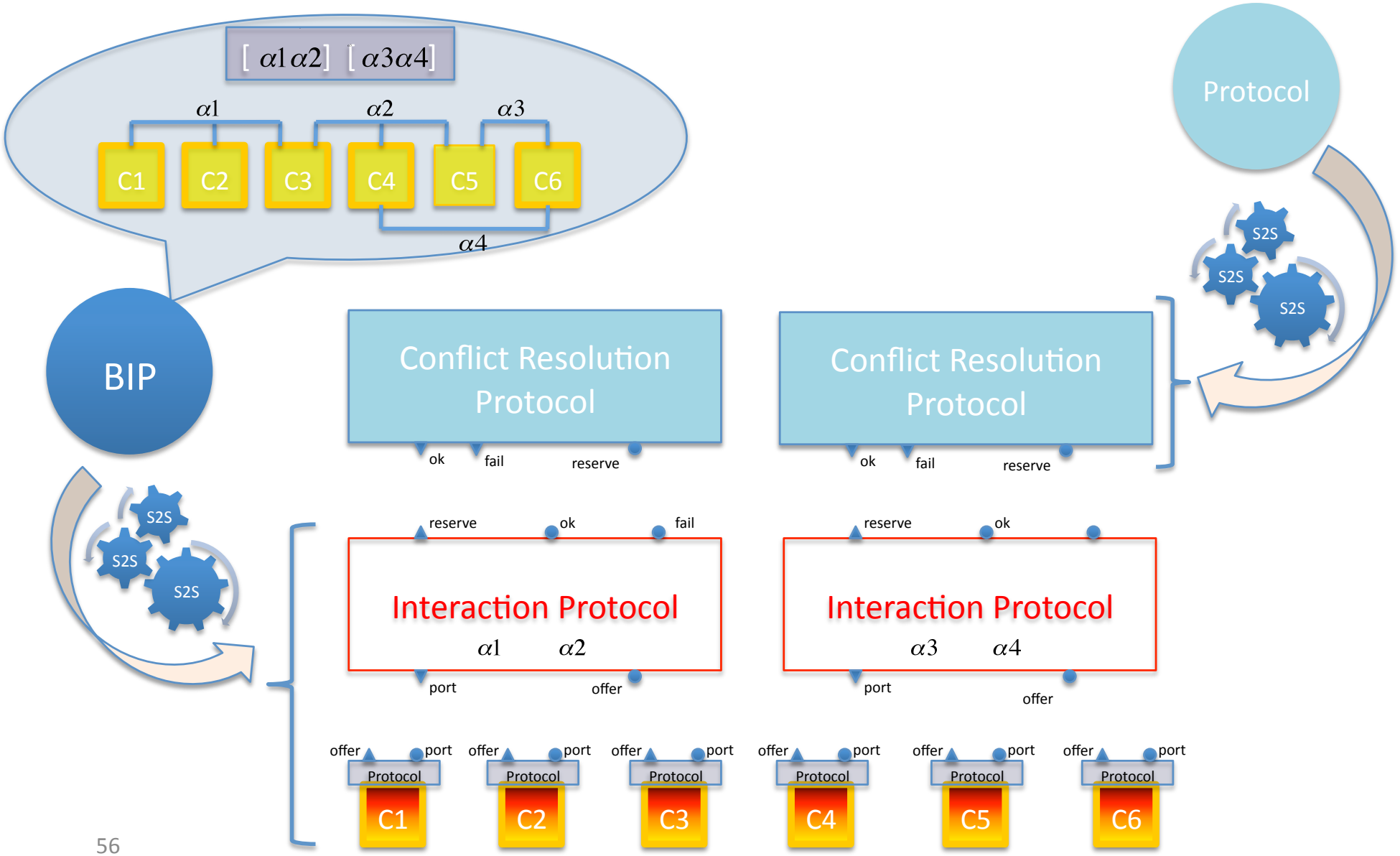


I1 # I2 # I3

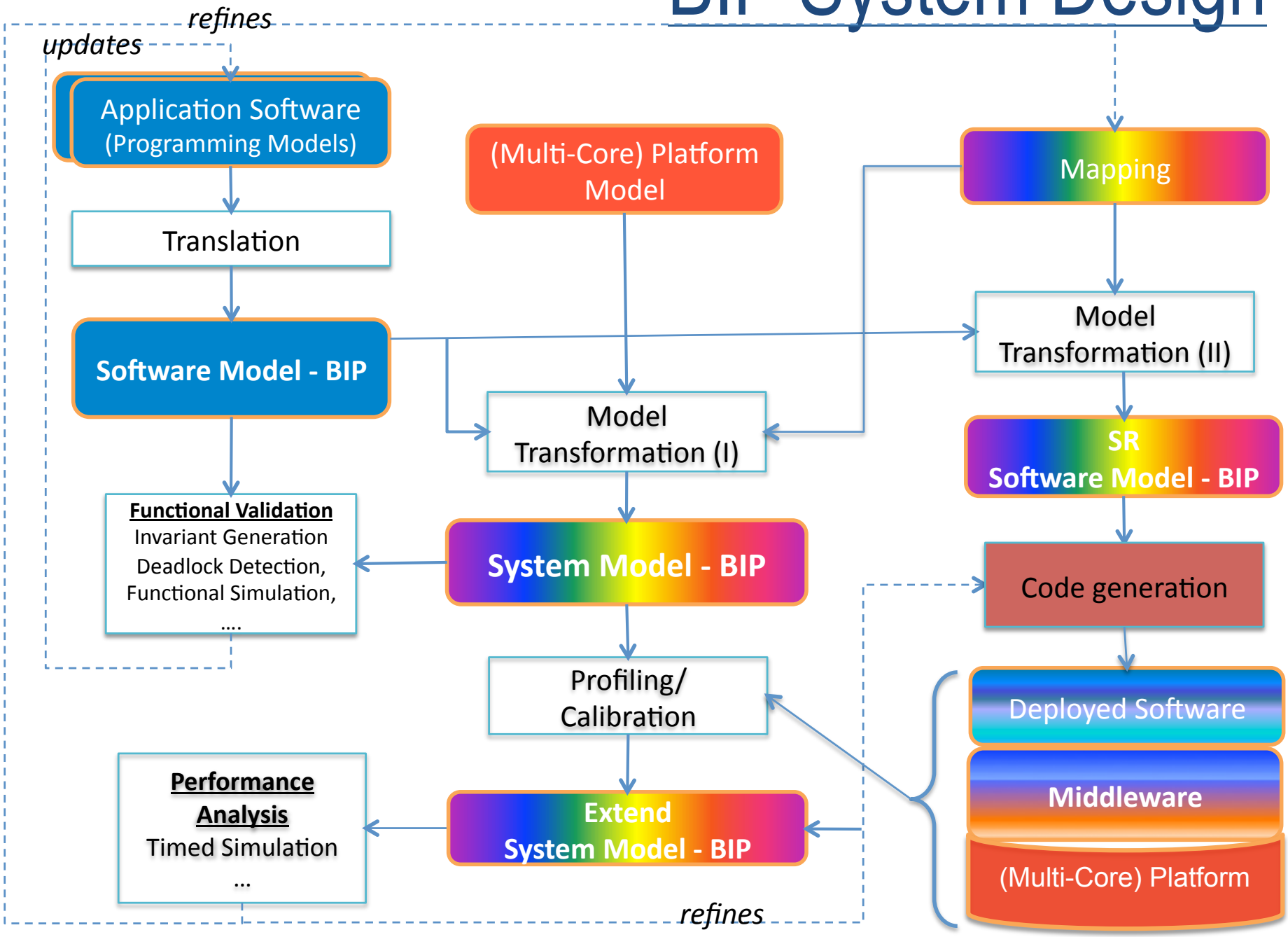
I4 # I5 # I6



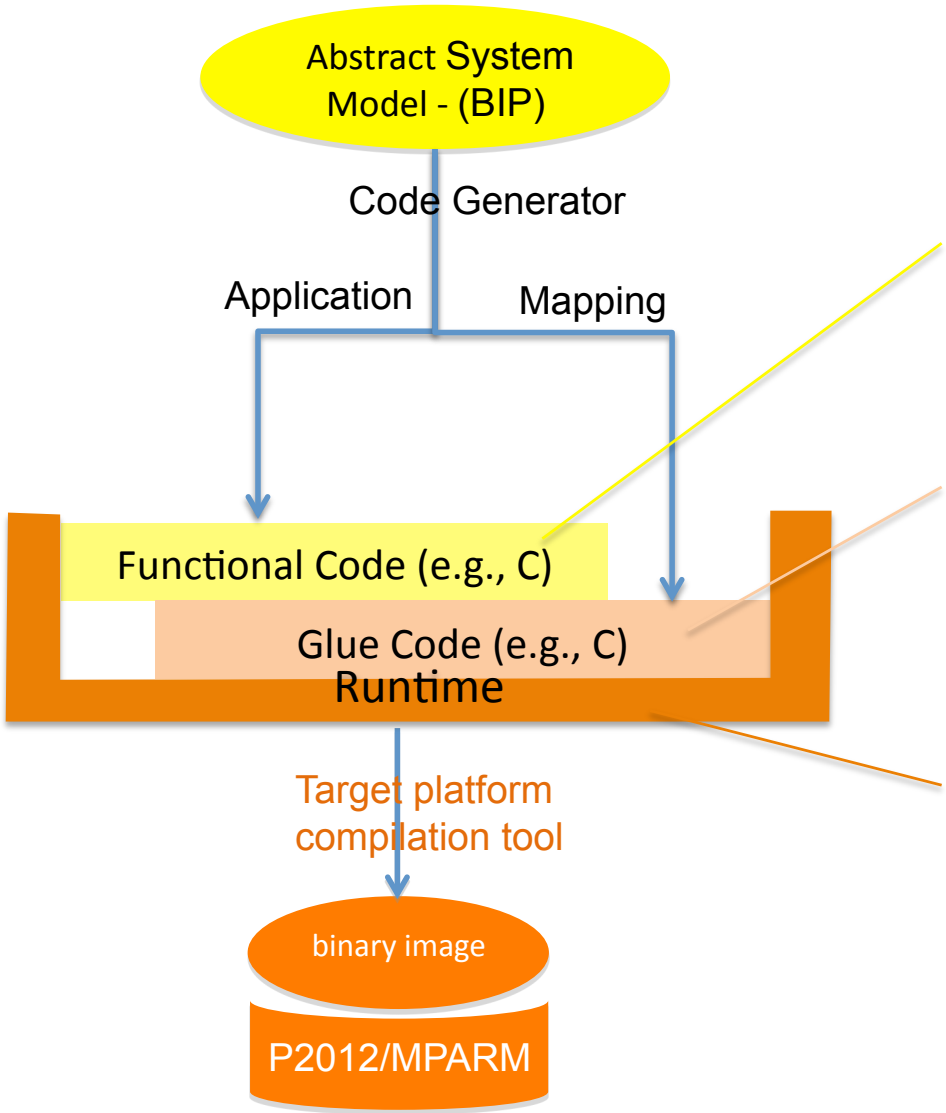
Send/Receive BIP



BIP System Design



Code Generation: Overview

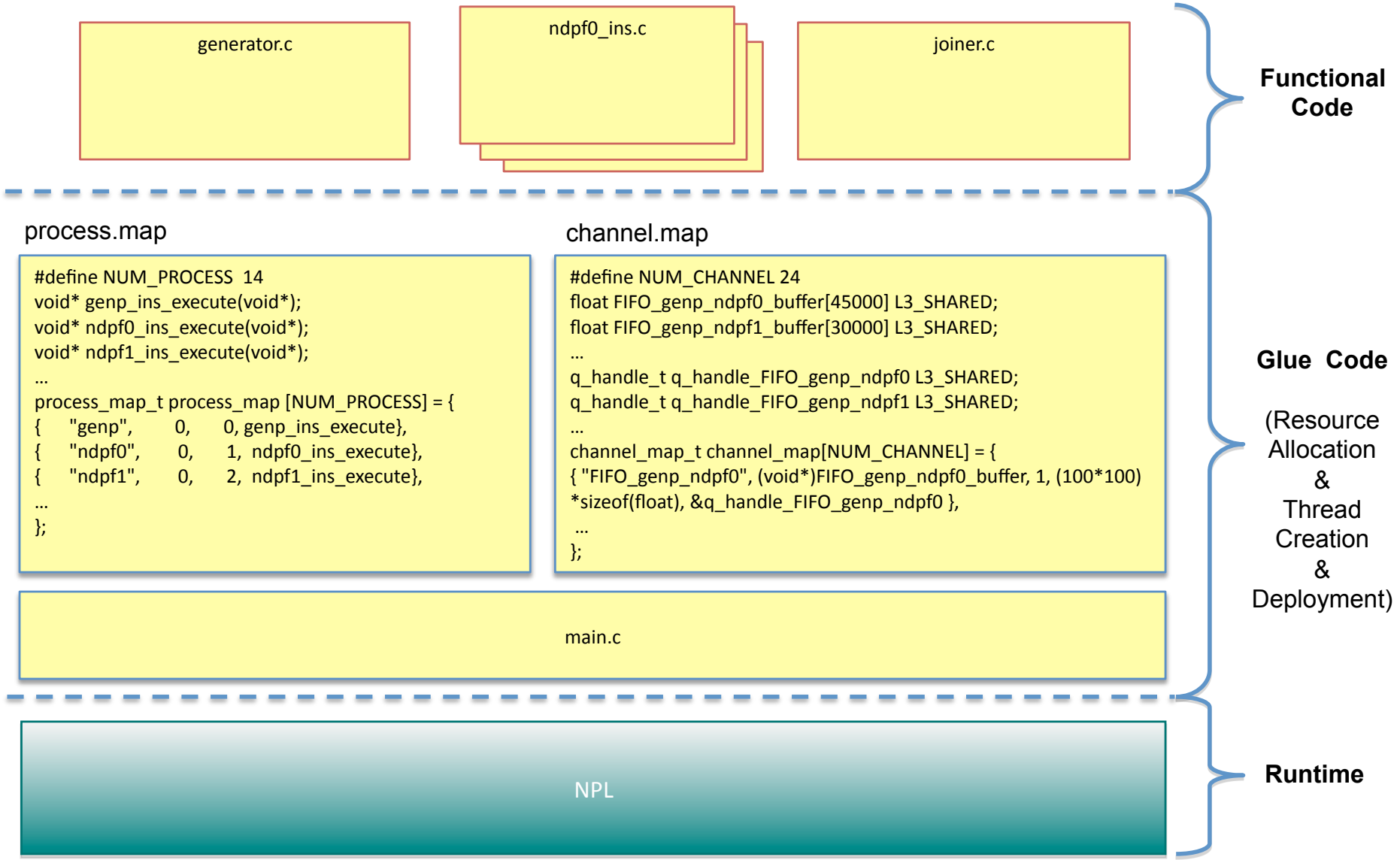


- Application task: data + thread routine
- Communication: API calls provided by runtime

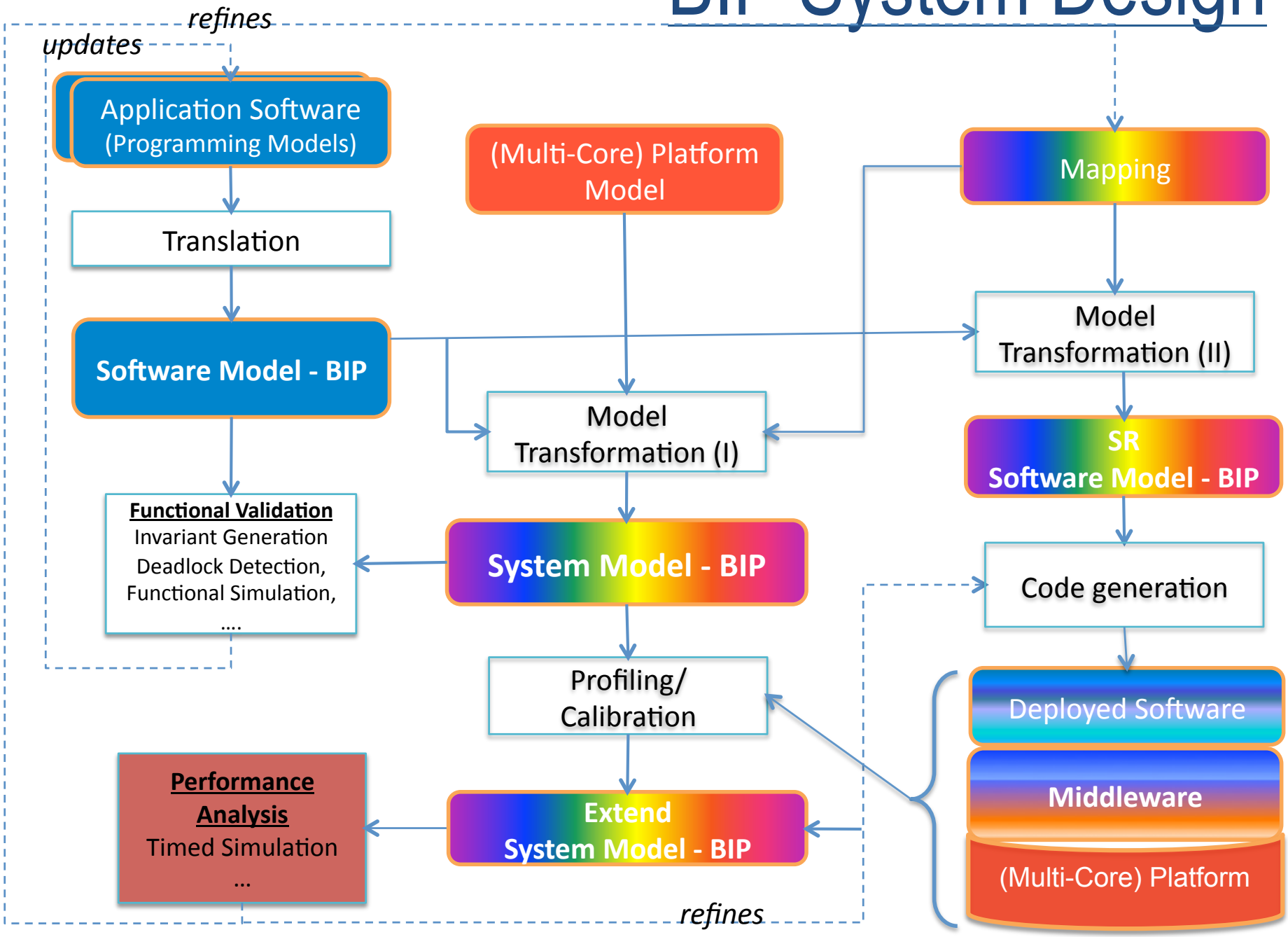
- Main application routine
- Deployment
  - threads to cores
  - data to memories
- Data allocation: thread stacks; FIFO queues

- API
  - Thread Management
  - Memory allocation
  - Communication
  - Synchronization

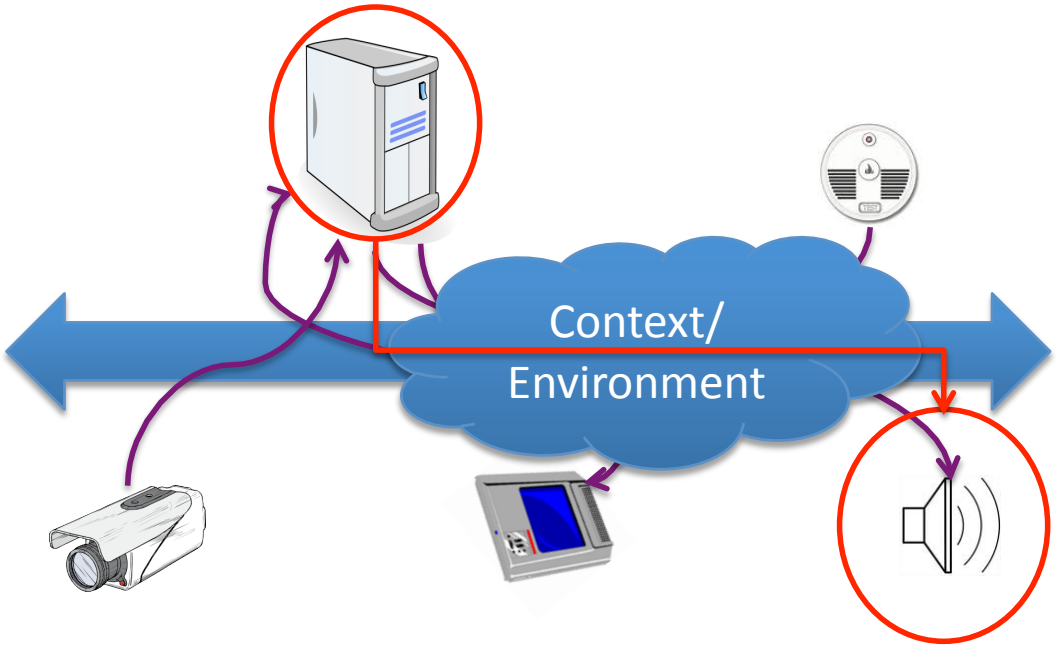
Example: P2012 Code Generation



BIP System Design



Our Methodology



1. Build executable model of the overall system
2. Learn probability distribution of key characteristics impacting application
3. Plug distributions and build a stochastic abstract model of the context
4. Apply Statistical model checking on the reduced model

Statistical Model Checking (1)

- Statistical methods to decide if a property is satisfied
- Estimate the probability that a system satisfies a property
- An alternative to avoid exhaustive exploration of state-space of a system
- Results might not be always correct, but possible to bound the probability of making errors
- Accuracy of estimates depends on no. and length of simulations
- Simple to implement and use
- Less memory and time intensive (compared to Model Checking)

Statistical Model Checking (2)

What are the questions ?

- Qualitative Question: Does $S \models P_{\geq\theta}(\phi)$?
- Quantitative Question: What is the probability for S to satisfy ϕ ?

Principle :

- Reason on a finite set of executions and answer the question

Discussion

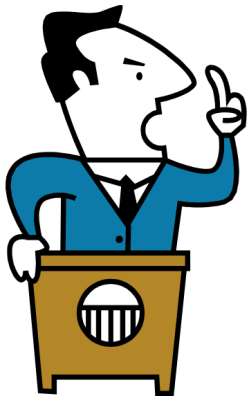
- Component framework encompassing heterogeneous composition
  - **Separation of concerns** between behavior and architecture (Interaction + Priority) involving a minimal set of constructs and principles
  - **Expressiveness** : BIP is as expressive as the universal glue
- Rigorous Design Flow
  - **Correctness-by-construction**
    - Source to source transformations
    - Verification based on compositionality, composability and incrementality
  - **System-level analysis** techniques jointly taking into account
    - application, hardware resources and mapping
    - Component and interaction partitioning
- Applications
  - Software **componentization**
  - **Programming** multicore systems
  - Complex systems **modeling and analysis** e.g., IMA

BIP Related Projects

ACOSE	BGLE	Speculative and Exploratory Design in System Engineering
ManyCoreLabs	BGLE	Generic Embedded Systems Platform
SPICES	ITEA	Support for Predictable Integration of Mission Critical Embedded Systems
COMBEST	FP7	Component-Based Embedded Systems Design Techniques
PRO3D	FP7	Programming for Future 3D Architectures with Many Cores
SMECY	ARTEMIS	Smart Multicore Embedded Systems
ACROSS	ARTEMIS	Artemis Cross-Domain Architecture
ASCENS	FP7	Autonomic Service Component Ensembles
CERTAINTY	FP7	CErtification of Real Time Applications desIgNed for mixed criticaliTY
MIND	Minalogic	Technologie d'assemblage des composants logiciels embarques
CHAPI	ANR	Calcul Embarque Hautes Performances pour des Applications Industrielles
MARAE	FRAE	Methodes et Architectures Robustes pour l'Autonomie dans l'Espace
GOAC	ESA	Goal Oriented Autonomous Controller
SYMPAA		Controlleur de Paiement Monetique de type Automate sur Autoroute

http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html

- Thanks for your attention.



Questions?