Metropolis Metro II: a design framework based on composition of heterogeneous models

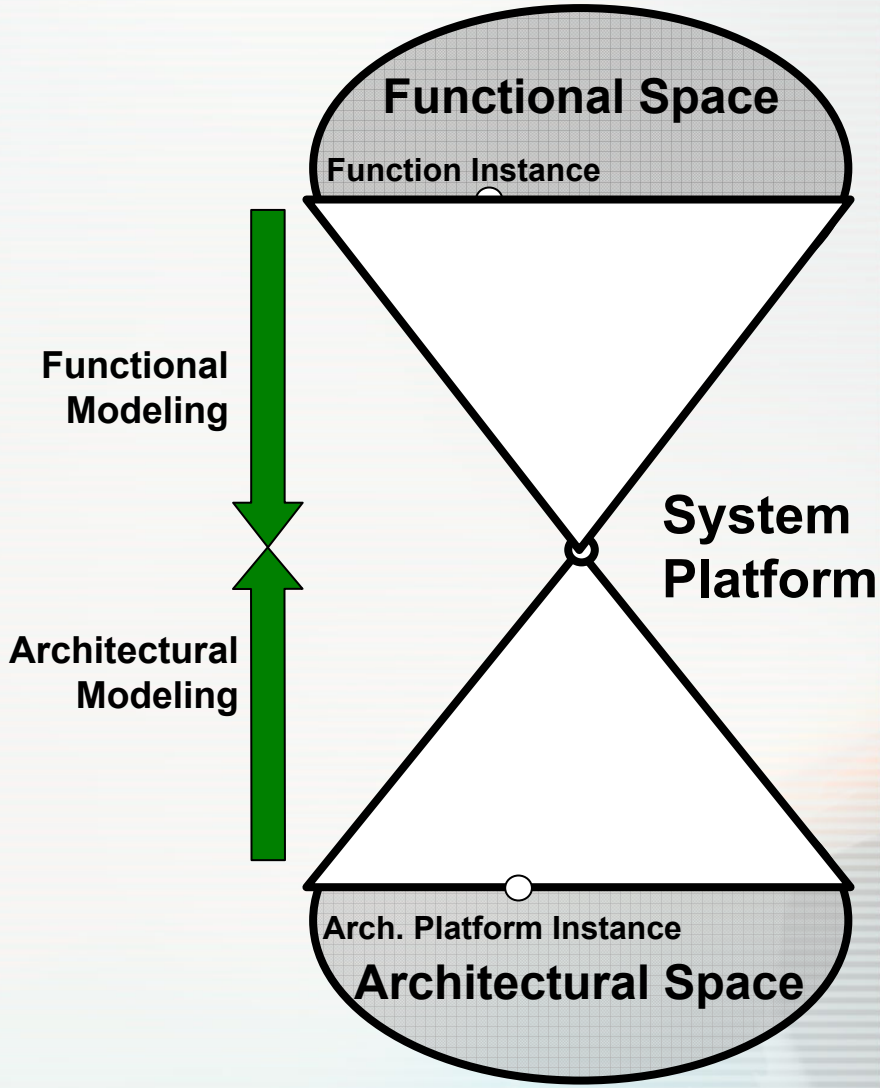
Outline

- Background on Metropolis
  - Semantics
  - Modeling Architecture
- Metropolis II infrastructure
  - Import
  - Execution semantics
  - Applications
    - UMTS
    - Energy Efficient Building

Copyright A. Sangiovanni-Vincentelli

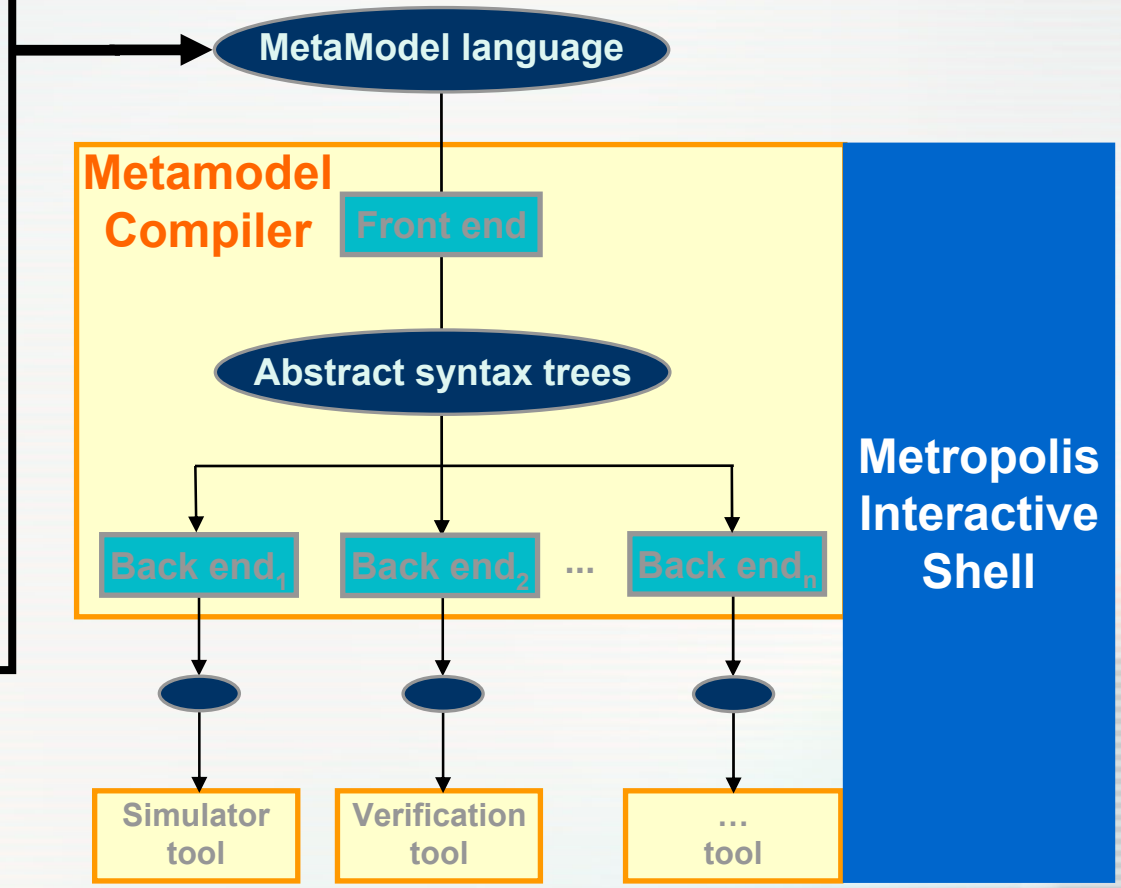
Platform-based Design

- A platform consists of a set of services
  - Associated semantics
- Functionality
  - Uses services
- Architecture
  - Provides services at some cost



Metropolis Design Framework

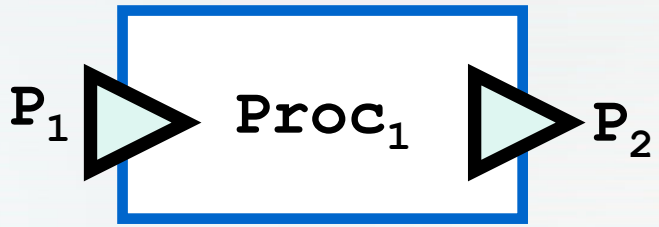
- Functionality
  What does it do?
- Constraints
  How should it do it?
- Architecture Platform
  How is it done?
  At what cost?
- Mapping
  Binding between the two



Metropolis Objects

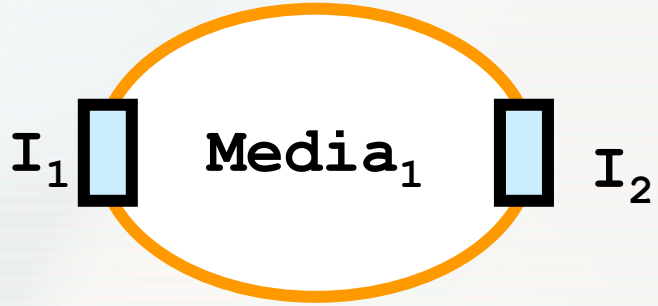
- Metropolis elements adhere to a "separation of concerns" point of view.

- **Processes (Computation)**



Active Objects
Sequential Executing Thread

- **Media (Communication)**



Passive Objects
Implement Interface Services

- **Quantity Managers (Coordination)**

Schedule access to resources and quantities

Copyright A. Sangiovanni-Vincentelli

Key Modeling Concepts

- An event is the fundamental concept in the framework
  - Represents a transition in the action automata of an object
  - An event is owned by the object that exports it
  - During simulation, generated events are termed as *event instances*
  - Events can be annotated with any number of quantities
  - Events can partially expose the state around them, constraints can then reference or influence this state
- A service corresponds to a set of sequences of events
  - All elements in the set have a common begin event and a common end event
  - A service may be parameterized with arguments

1. E. Lee and A. Sangiovanni-Vincentelli, *A Unified Framework for Comparing Models of Computation*, IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, Vol. 17, N. 12, pg. 1217-1229, December 1998

Copyright A. Sangiovanni-Vincentelli

Action Automata

- Processes take *actions*.
  - statements and some expressions, e.g.
    y = z+port.f();, z+port.f(), port.f(), i < 10, …
  - only calls to media functions are *observable actions*

- An *execution* of a given netlist is a sequence of vectors of *events*.
  - *event* : the beginning of an action, e.g. B(port.f()),
    the end of an action, e.g. E(port.f()), or null N
  - the *i*-th component of a vector is an event of the *i*-th process

- An execution is *legal* if
  - it satisfies all coordination constraints, and
  - it is accepted by all action automata.

Copyright A. Sangiovanni-Vincentelli

Semantics summary

- **Processes run sequential code concurrently, each at its own arbitrary pace.**
- **Progress may block at synchronization points**
  - awaits
  - function calls and labels to which awaits or constraints refer.
- **The legal behavior of a netlist is given by a set of sequences of event vectors.**
  - multiple sequences reflect the non-determinism of the semantics:

    concurrency, synchronization (awaits and constraints)

Copyright A. Sangiovanni-Vincentelli

Outline

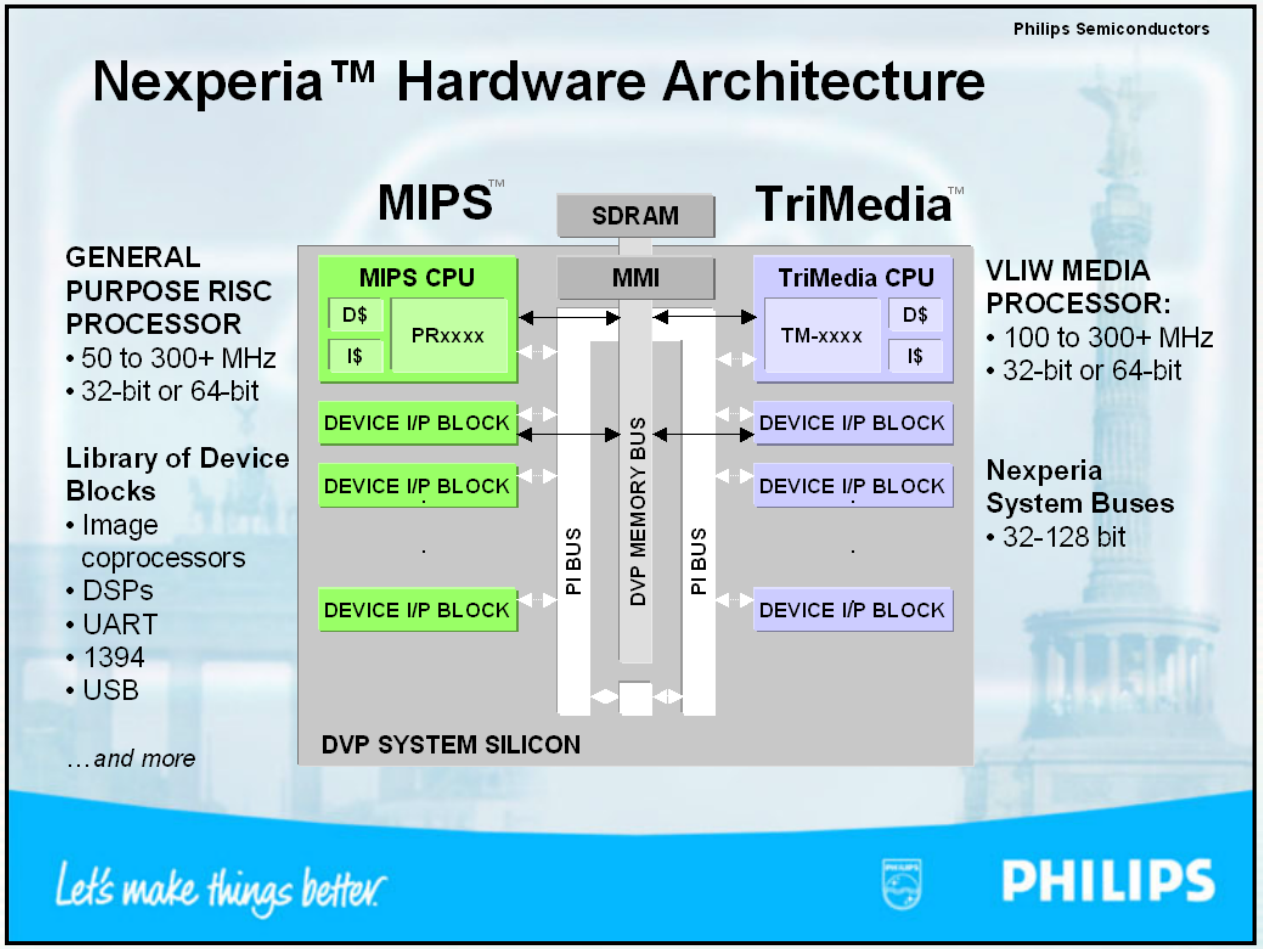
- Background on Metropolis
  - Semantics
  - Modeling Architecture
- Metropolis II infrastructure
  - Import
  - Execution semantics
  - Applications
    - UMTS
    - Energy Efficient Building

Copyright A. Sangiovanni-Vincentelli

Architecture components

An architecture component specifies *services*, i.e.

- what it *can* do
- how much it *costs*



Copyright A. Sangiovanni-Vincentelli

Meta-model: architecture components

An architecture component specifies *services*, i.e.

- what it *can* do:

  interfaces, methods, coordination (awaits, constraints), netlists

- how much it *costs:*

  quantities, annotated with events, related over a set of events

```
interface BusMasterService extends Port {
  update void busRead(String dest, int size);
  update void busWrite(String dest, int size);
}
```

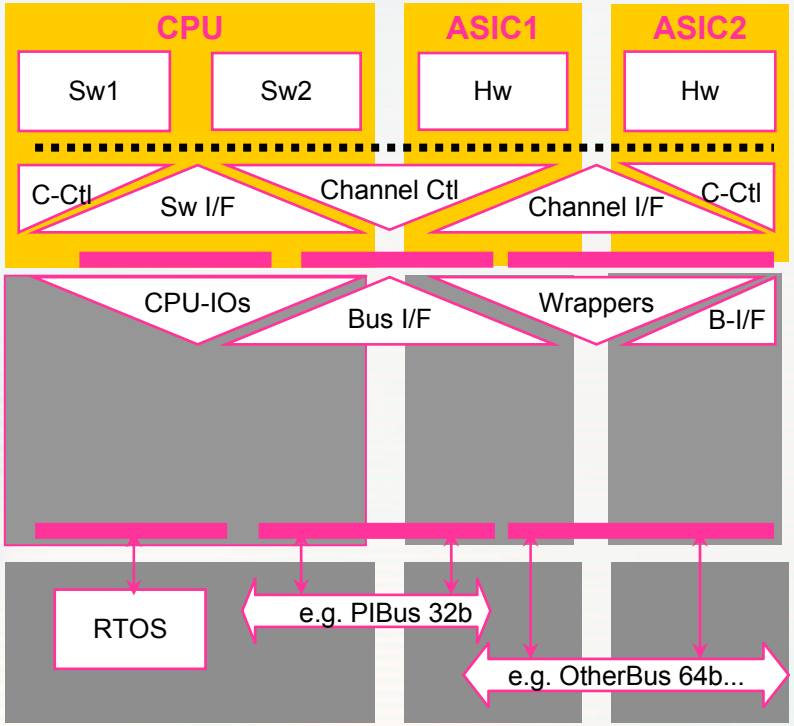
```
medium Bus implements BusMasterService …{
  port BusArbiterService Arb;
  port MemService Mem; …
  update void busRead(String dest, int size) {
     if(dest== … ) Mem.memRead(size);
   }

  …
```

Copyright A. Sangiovanni-Vincentelli

Meta-model: architecture components

- **This modeling mechanism is generic, independent of services and cost specified.**
- **Which levels of abstraction, what kind of quantities, what kind of cost constraints should be used to capture architecture components?**
  - depends on applications



Transaction:

Services:

- fuzzy instruction set for SW, execute() for HW

- bounded FIFO (point-to-point)

Quantities:

- #reads, #writes, token size, context switches

Virtual BUS:

Services:

- data decomposition/composition

- address (internal v.s. external)

Quantities: same as above, different weights

Physical:

Services: full characterization

Quantities: time

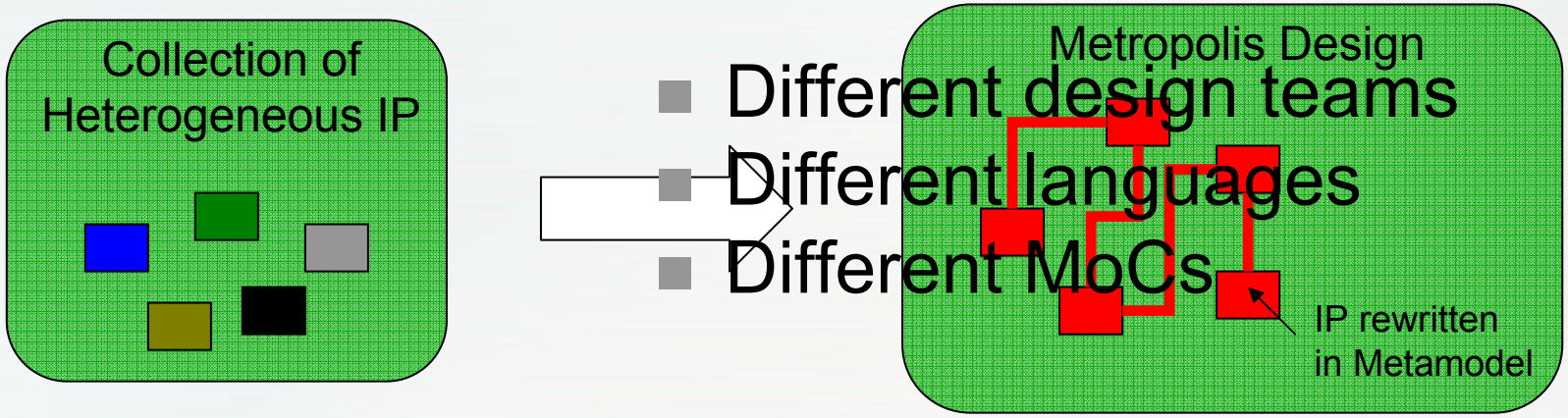
Copyright A. Sangiovanni-Vincentelli

Outline

- Background on Metropolis
  - Semantics
  - Modeling Architecture
- Metropolis II infrastructure
  - Import
  - Execution semantics
- Applications
    - UMTS
    - Energy Efficient Building

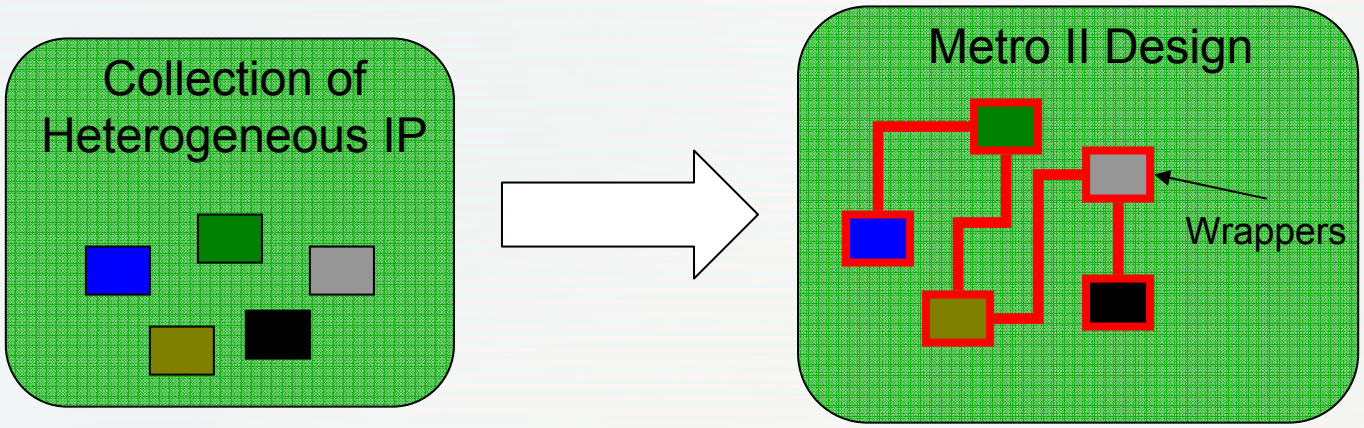
Copyright A. Sangiovanni-Vincentelli

Heterogeneous IP Import in Metropolis



- **Excessive time spent in design import**
  - **Redefining and implementing classes and methods**
  - **Memory allocation, data types, templates, etc**
- **Challenges in Infineon case study**
  - **802.11a on MuSIC (multiple SIMD core) architecture**

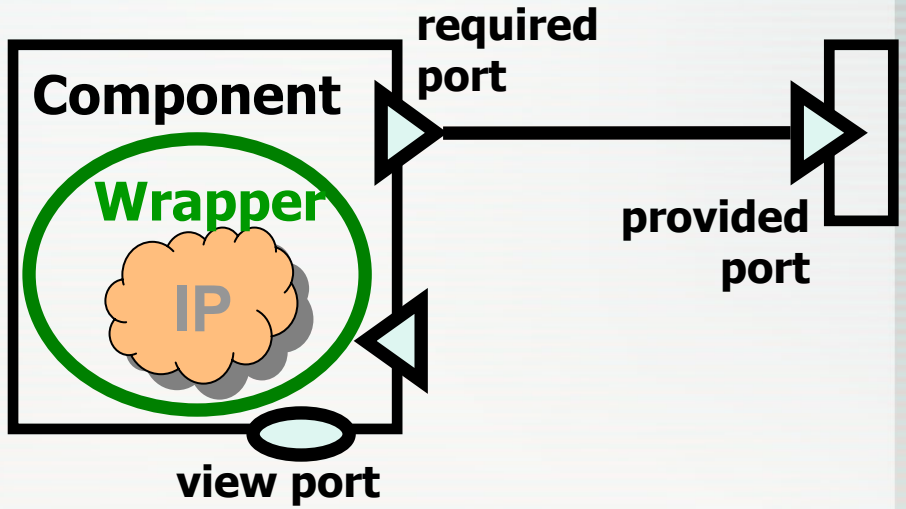
Heterogeneous IP Import in Metro II



- **Pros**
  - **Framework easier to develop and maintain**
  - **Leverage existing compilers/debuggers**
  - **Quicker import for most IP**
- **Cons**
  - **Framework has limited visibility**

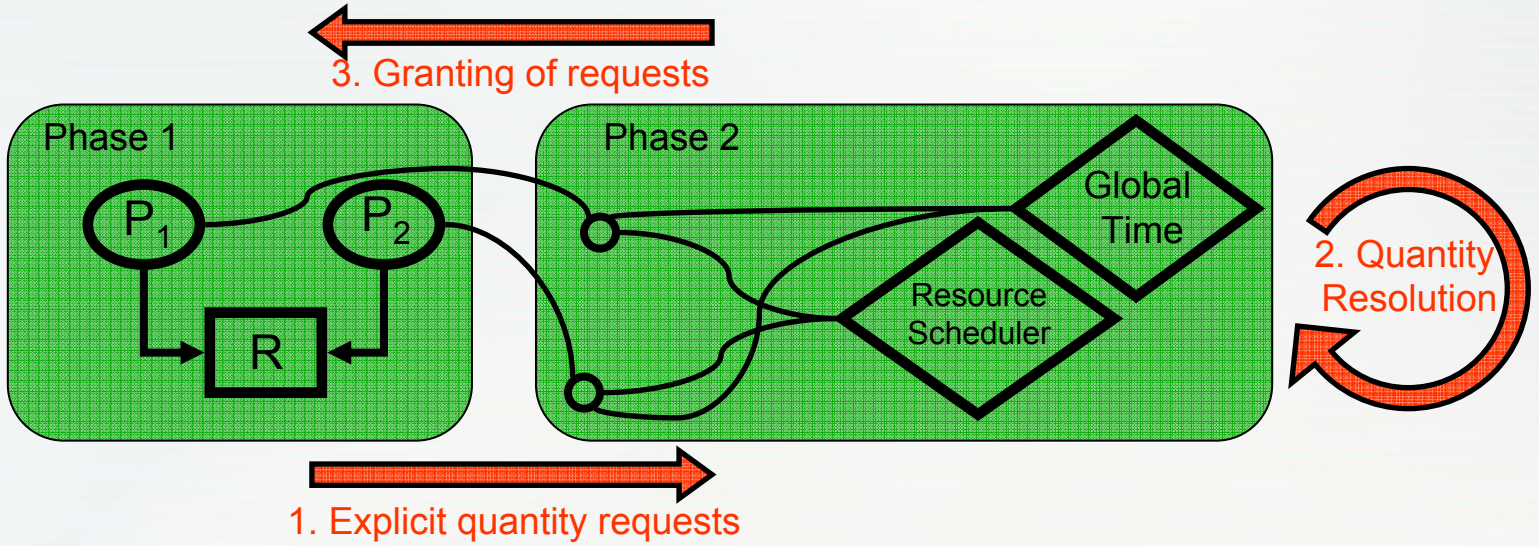
Components, Ports, and Connections

- IP is wrapped to expose framework-compatible interface
- Components encapsulate wrapped IP



- **Ports**
  - **Coordination: provided, required**
  - **View ports**
- **Connections**
  - **Each method in interface for provided-required connection associated with begin and end events**

Behavior-Performance Separation in Metropolis



- **Processes make explicit requests for annotation**
- **Annotation/scheduling are intertwined**
  - **Iteration between multiple quantity managers**
- **Challenges in GM case study**
  - **Vehicle stability application on distributed CAN architecture**
  - **Interactions between global time QM and resource QM difficult to debug**

Events

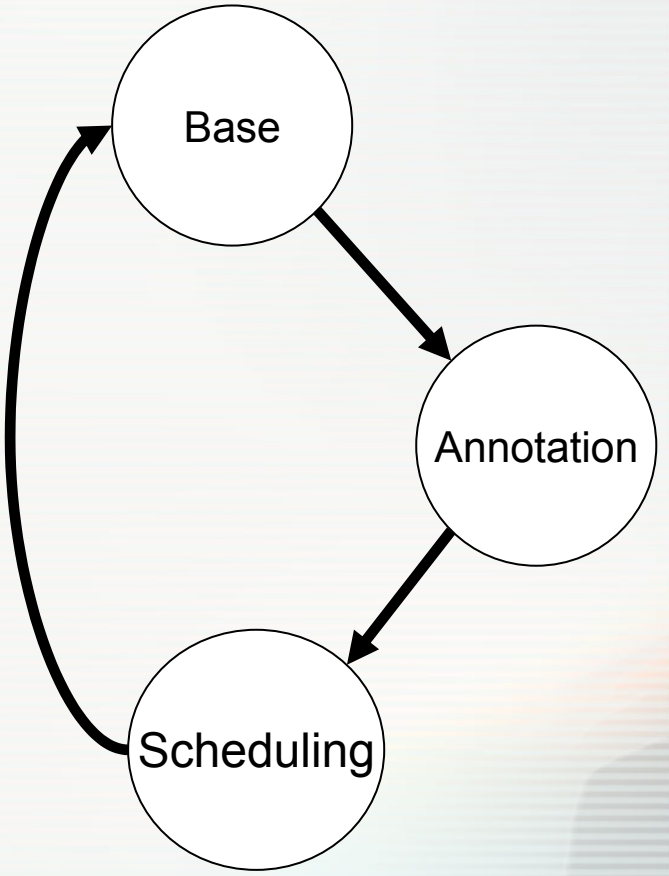
- **An event is the fundamental concept in the framework**
- **Fields:**
  - **Process: Generator of event**
  - **Value Set: Variables exposed along with event**
  - **Tag Set: Quantity annotations**
- **Tagged Signal Model***

E = <p, V, T>

* E. A. Lee and A. Sangiovanni-Vincentelli, “A Framework for Comparing Models of Computation”. IEEE Transactions on CAD, vol. 17. no. 12, December 1998.

3 Phase Execution

1. **Base**
   - Each process proposes events and suspends
   - Multiple events can be proposed simultaneously by one process
2. **Annotation**
   - Tag proposed events with quantities
3. **Scheduling**
   - Rejection of some proposed events
   - At most 1 enabled event per process

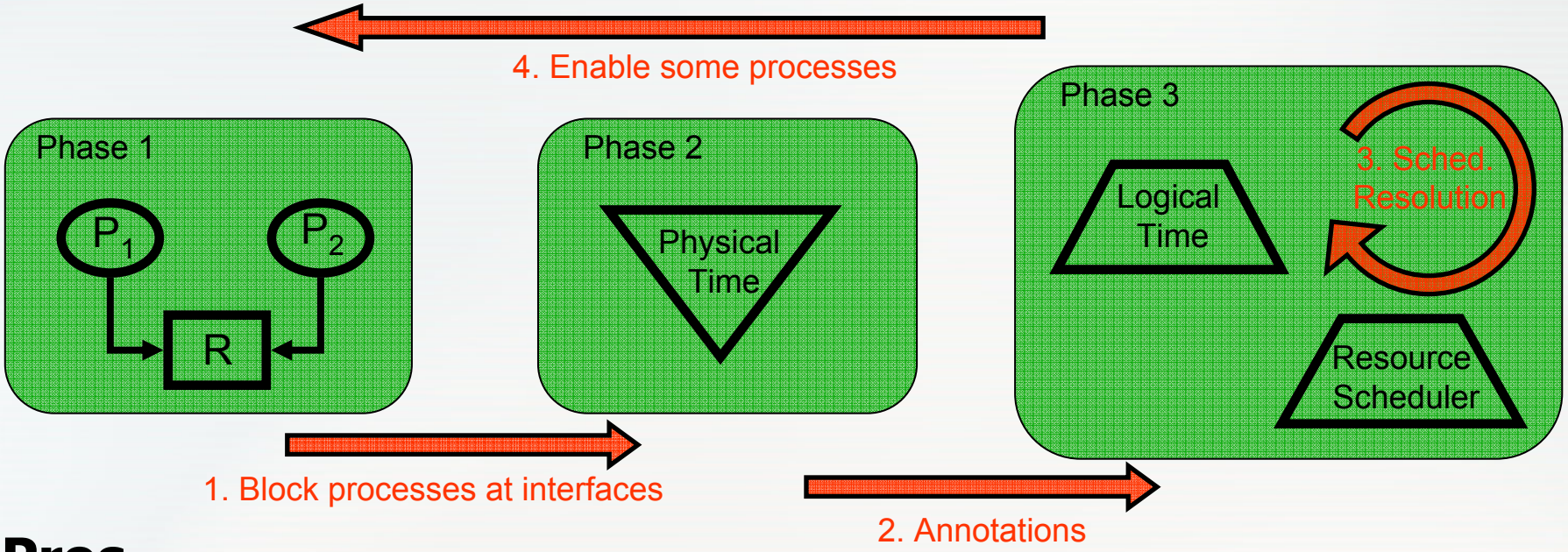


Phases and Events

- **Each phase is allowed to interact with events in a limited way**
  - **Keep responsibilities separate**

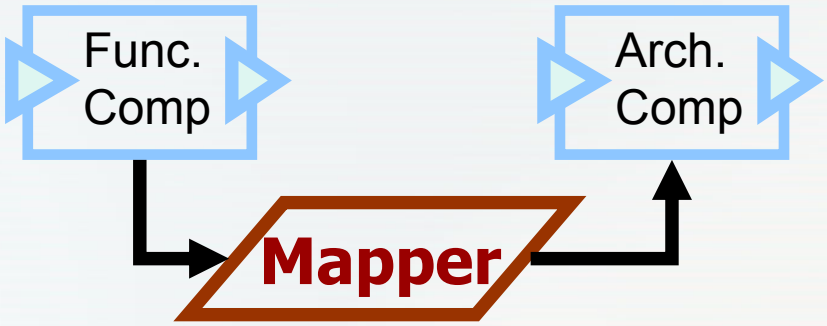
Phase	Events		Tags		Values	
	Propose	Disable	Read	Write	Read	Write
Base	Yes				Yes	Yes
Annotation			Yes	Yes	Yes	
Scheduling		Yes	Yes		Yes	

Behavior-Performance Separation in Metro II



- **Pros**
  - **Phase 1 objects no longer explicitly request annotation**
  - **Separation of quantity managers into annotators and schedulers**
    - **"Global time" separates into physical time (annotation) and logical time (scheduling)**
- **Cons (?)**
  - **Additional phase introduced into execution model**

Mappers

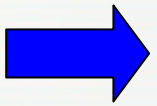


- Mapping occurs at the component level
  - Between components with compatible interfaces
  - Possibly many functional components mapped to a single architectural component

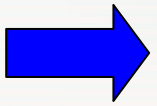
- **Mappers are objects that help specify the mapping**
  - **Bridge syntactic gaps only**

Summary: Features for Metro II

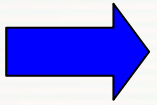
- **Import heterogeneous IP**
  - **Different languages**
  - **Different models of computation**
- **Behavior-Performance Separation**
  - **No explicit requests for annotation**
  - **Annotation separated from scheduling**
- **Operational/Denotational Separation**
  - **Restricted access to events and values**
  - **Mapping carried out at component level**



Coordination Framework



3-Phase Execution



Event-oriented Framework

Outline

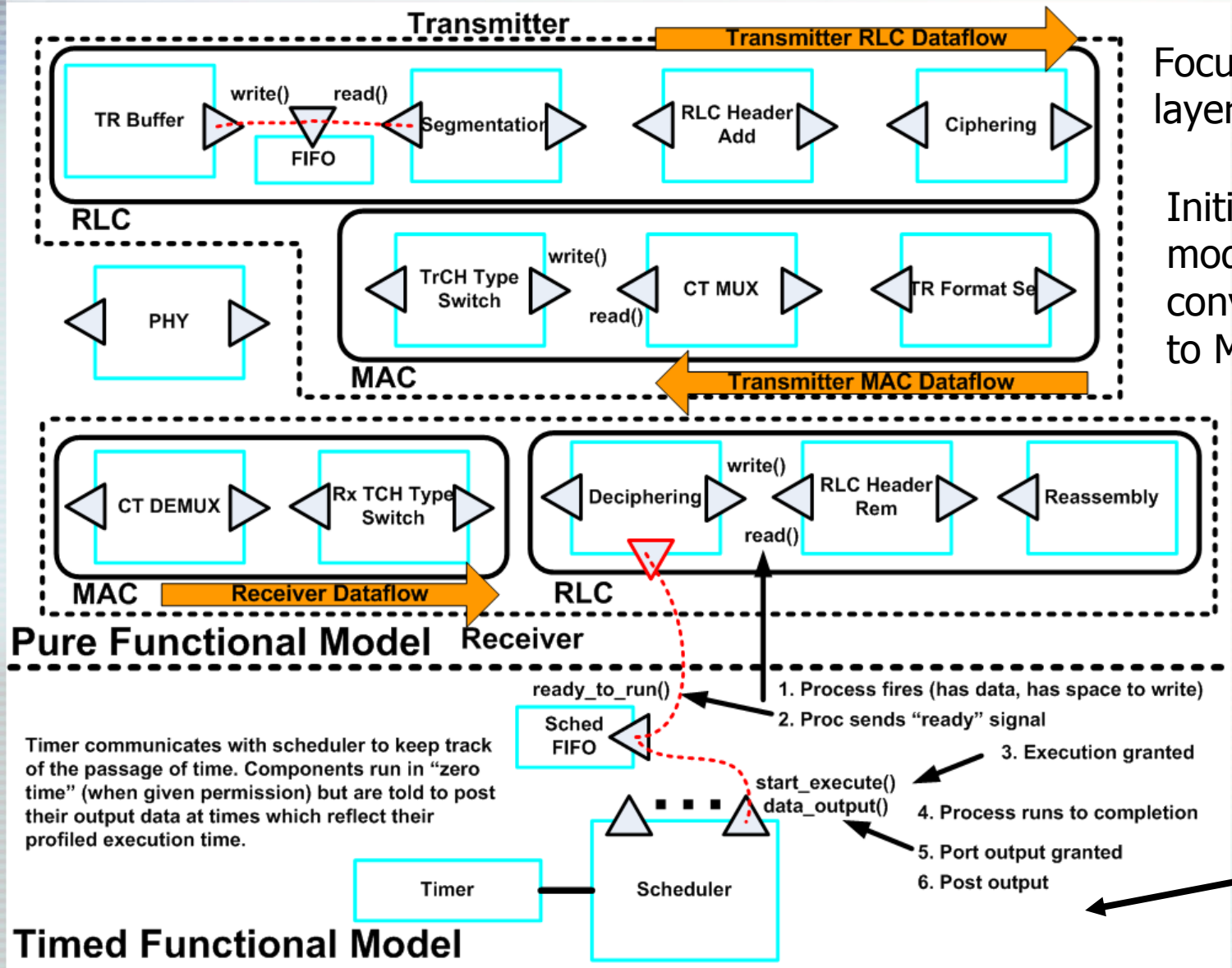
- Background on Metropolis
  - Semantics
  - Modeling Architecture
- Metropolis II infrastructure
  - Import
  - Execution semantics
- Applications
    - UMTS
    - Energy Efficient Building

Copyright A. Sangiovanni-Vincentelli

Design Activity: UMTS Case Study

- UMTS is a mobile communication protocol standard
  - Universal Mobile Telecommunications System
  - 3G cell phone technology
  - Often used in Software Defined Radio (SDR)
- Started with C and SystemC models as baseline
  - Source of Metro II functional models
  - Profiling to use in architecture models
  - Comparisons for Metro II simulation results
- Have both DLL and PHY level SystemC models
  - Converted only data link layer to Metro II

Metro II UMTS Models



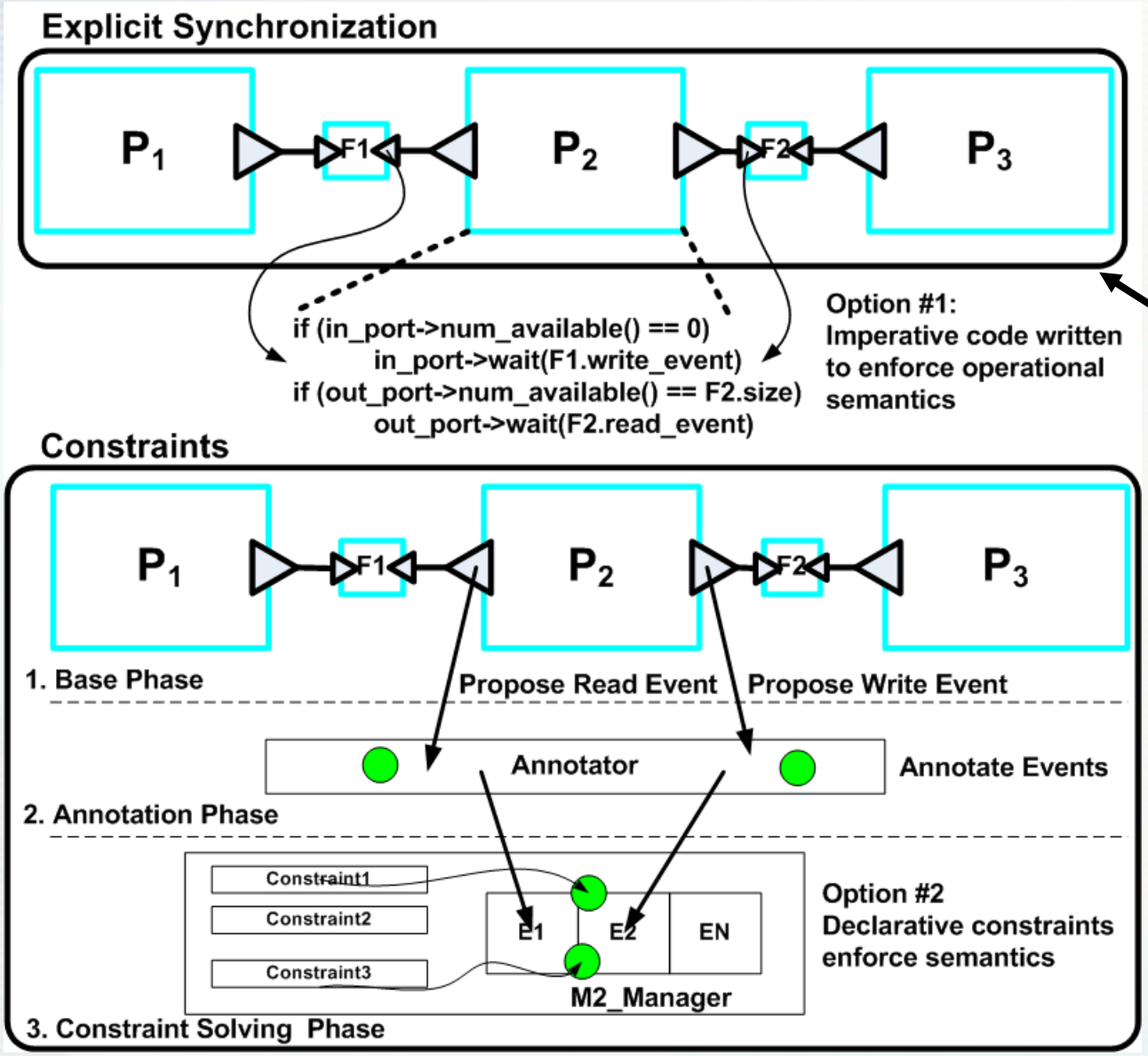
Focused on the DLL layer

Initial SystemC model was converted to Metro II

Two Models:

- Pure functional model with blocking read and write semantics.
- Timed model with a scheduler and preemption.

Synchronization Mechanisms

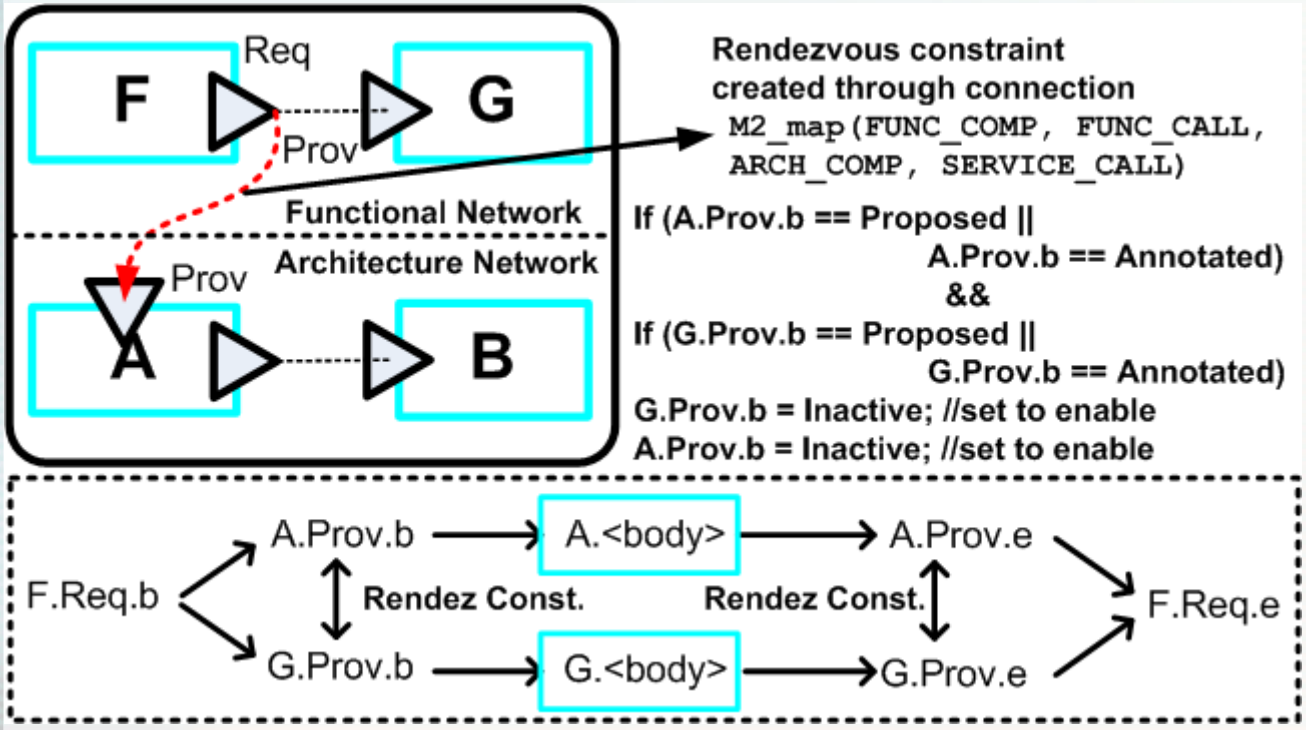


UMTS example exposed two approaches to synchronization in Metro II:

Explicit Synchronization:
Use the underlying simulation framework directly
i.e. SystemC "or/and" waits

Constraints:
Move synchronization from phase 1 to phase 3 completely.

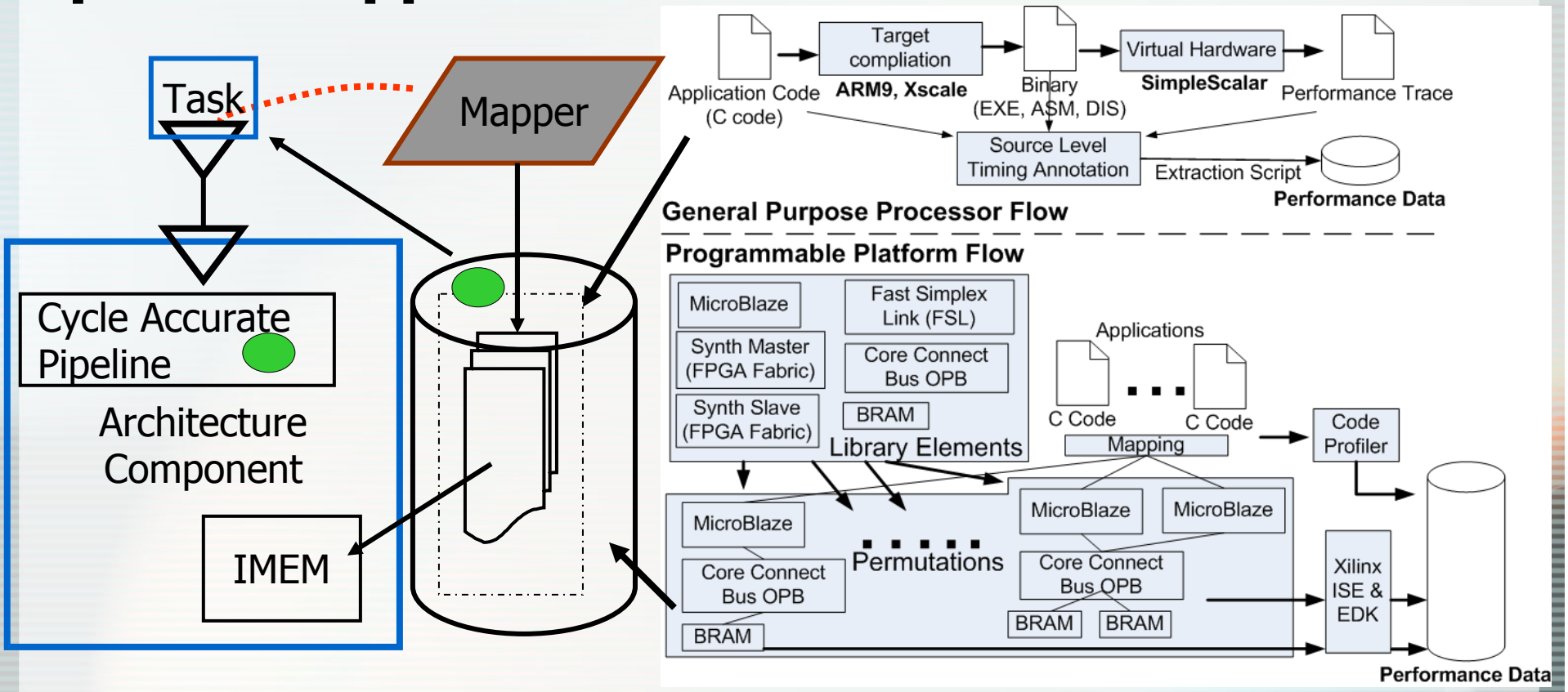
Metro II Mapping Semantics



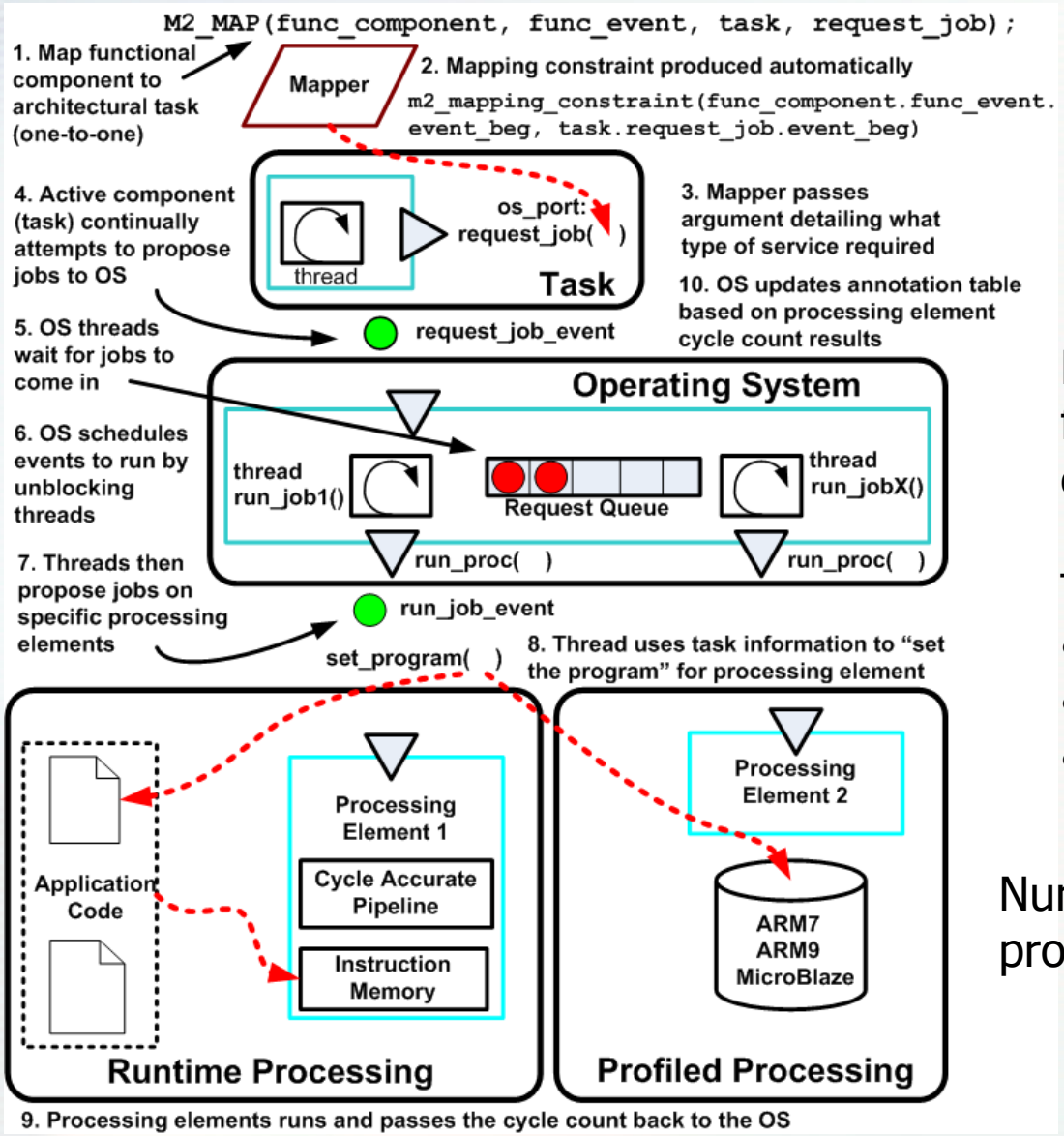
Mapping allows both functional and architectural progress "simultaneously" via rendezvous type constraints.

Metro II: Service Modeling

- **Two basic architecture modeling styles: cycle accurate runtime analysis vs. off line, pre-profiled approach**



Architecture Model Overview



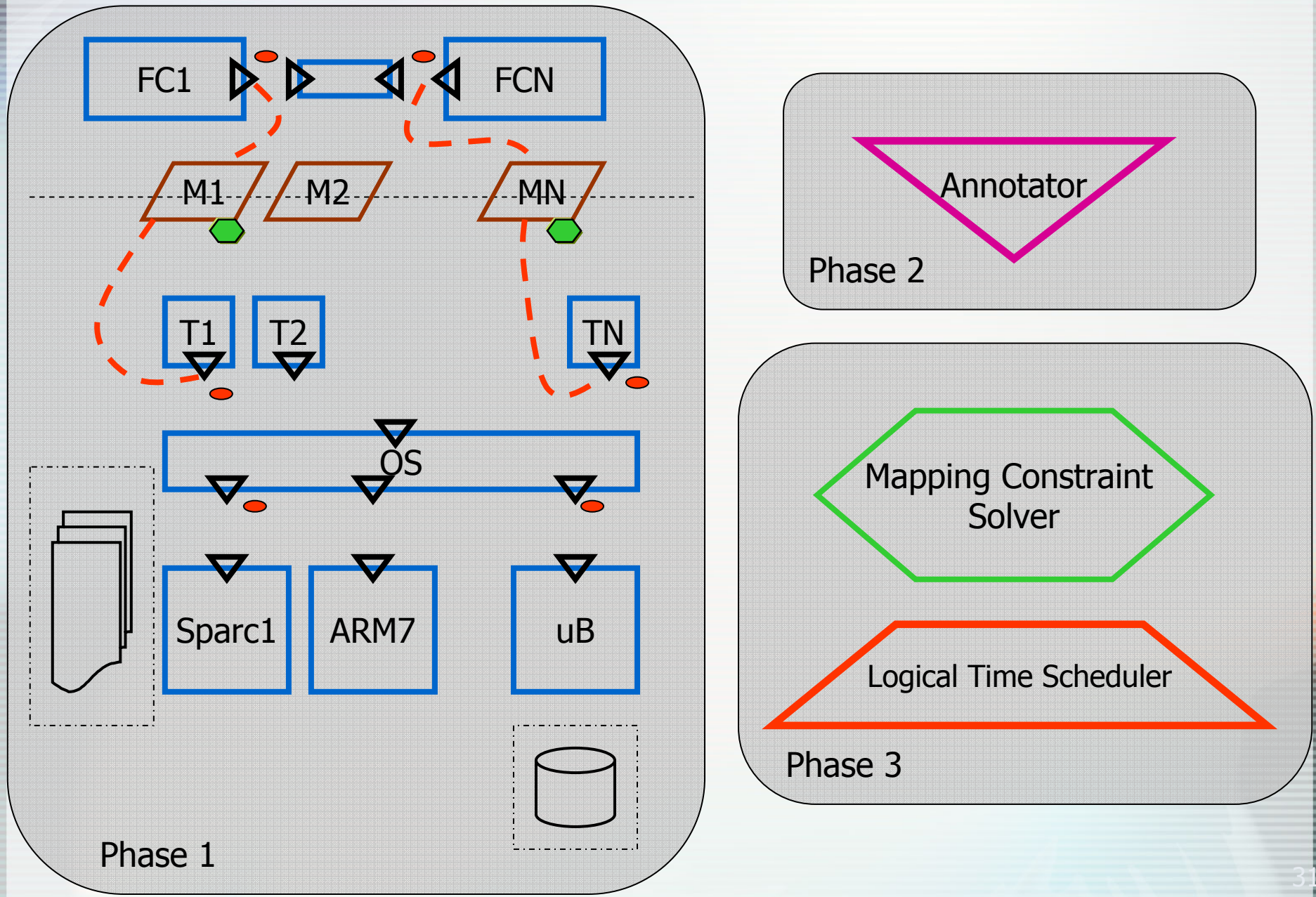
Tasks for mapping 1-to1 with functional components

RTOS for scheduling events from N tasks to M processing elements

Three scheduling policies:
- Round Robin
- Fixed Priority
- FCFS

Numerous configurations of processing elements (48 chosen)

Metro II Complete System



UMTS Case Study Outcome

- Processing elements include
  - ARM7, ARM9: GPP profiling approach
  - Microblaze: Programmable platform flow
  - SPARC: runtime profiling with C code snippets of core routines
- 48 different mappings explored
  - 11 PEs, 1 PE, combinations of 4 PEs broken down by RLC tx/rx and MAC tx/rx
  - 9 classes within the 48 mappings

48 Mappings

#	Type	Partition	#	Type	Partition	#	Type	Partition
1	1: RTP	11 Sp	17	6: PP	2 μB (2), 2 A9 (3)	33	7: MIX	A7 (4), Sp (5), μB (6), A9 (7)
2	2: PP	11 μB	18	6: PP	2 A9 (2), 2 μB (3)	34	7: MIX	A7 (4), Sp (5), A9 (6), μB (7)
3	2: PP	11 A7	19	6: PP	2 A7 (2), 2 A9 (3)	35	7: MIX	A7 (4), μB (5), Sp (6), A9 (7)
4	2: PP	11 A9	20	6: PP	2 A9 (2), 2 A7 (3)	36	7: MIX	A7 (4), μB (5), A9 (6), Sp (7)
5	3: RTP	4 Sp (1)	21	7: MIX	Sp (4), μB (5), A7 (6), A9 (7)	37	7: MIX	A7 (4), A9 (5), μB (6), Sp (7)
6	4: PP	4 μB (1)	22	7: MIX	Sp (4), μB (5), A9 (6), A7 (7)	38	7: MIX	A7 (4), A9 (5), Sp (6), μB (7)
7	4: PP	4 A7 (1)	23	7: MIX	Sp (4), A7 (5), μB (6), A9 (7)	39	7: MIX	A9 (4), Sp (5), μB (6), A7 (7)
8	4: PP	4 A9 (1)	24	7: MIX	Sp (4), A7 (5), A9 (6), μB(7)	40	7: MIX	A9 (4), Sp (5), A7 (6), μB (7)
9	5: MIX	2 Sp (2), 2 μB (3)	25	7: MIX	Sp (4), A9 (5), A7 (6), μB (7)	41	7: MIX	A9 (4), μB (5), Sp (6), A7 (7)
10	5: MIX	2 μB (2), 2 Sp (3)	26	7: MIX	Sp (4), A9 (5), μB (6), A7 (7)	42	7: MIX	A9 (4), μB (5), A7 (6), Sp (7)
11	5: MIX	2 Sp (2), 2 A7 (3)	27	7: MIX	μB (4), Sp (5), A7 (6), A9 (7)	43	7: MIX	A9 (4), A7 (5), μB (6), Sp (7)
12	5: MIX	2 A7 (2), 2 Sp (3)	28	7: MIX	μB (4), Sp (5), A9 (6), A7 (7)	44	7: MIX	A9 (4), A7 (5), Sp (6), μB (7)
13	5: MIX	2 Sp (2), 2 A9 (3)	29	7: MIX	μB (4), A7 (5), Sp (6), A9 (7)	45	8: RTP	1 Sp
14	5: MIX	2 A9 (2), 2 Sp (3)	30	7: MIX	μB (4), A7 (5), A9 (6), Sp (7)	46	9: PP	1 μB
15	6: PP	2 μB (2), 2 A7 (3)	31	7: MIX	μB (4), A9 (5), A7 (6), Sp (7)	47	9: PP	1 A7
16	6: PP	2 A7 (2), 2 μB (3)	32	7: MIX	μB (4), A9 (5), Sp (6), A7 (7)	48	9: PP	1 A9

(1 = Rx MAC, Tx MAC, Rx RLC, Tx RLC), (2 = Rx MAC, Rx RLC), (3 = Tx MAC, Tx RLC)
(4 = Rx MAC), (5)(Rx RLC), (6)(Tx MAC), (7 = Tx RLC) (Sp = Sparc, μB = Microblaze, A7 = ARM7, A9 = ARM9)

Execution Time and Utilization Analysis

- **Round Robin**
  - **Mapping #1 (fastest, 11 SPARCs) and #46 (slowest, 1 uBlaze) had a 2,167% difference**
- **Priority**
  - **Avg. execution time reduced by 13% over round robin**
  - **Avg. utilization decreases by 2%**
- **FCFS**
  - **Avg. execution time reduced by 7%**
  - **Avg. utilization increases by 27%**

SystemC vs. Metro II

- Metro II timed functional model has a 7.4% increase in runtime over SystemC timed functional model
- Mapped Metro II model is 54.8% faster than timed SystemC model
  - Metro II phases 2 and 3 have significantly less overhead than the timer-and-scheduler based system required by the SystemC timed functional model
- In a comparison of the Metro II timed model running without constraints and one running with them, the average runtime decrease was 25%

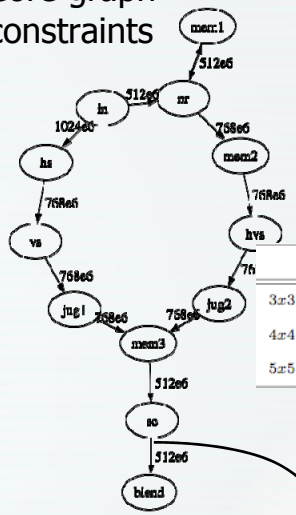
Design Effort

- Entire design
  - 85 files
  - 8,300 LOC
- Mapping change affects only 2 files
- Metro II conversion affects 1% of lines in each file
  - 58% of these lines relate to constraint registration
- SystemC SPARC model conversion adds only 3.4% to code size (92 lines)

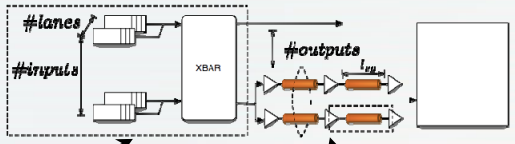
Metroll + COSI Designs

- On-Chip Communication

Core graph + constraints



Components and models



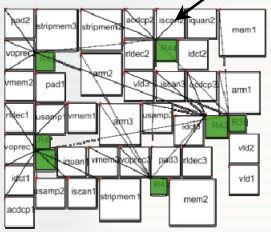
	ln	90nm 1.2V	65nm 0.9V	45nm 0.6V
3x3	1	13.6	5.6	2.2
	4	22.6	8.7	3.1
4x4	1	19.9	8	3.1
	4	35.1	13.4	4.6
5x5	1	27.1	10.9	4.1
	4	50	18.9	6.4

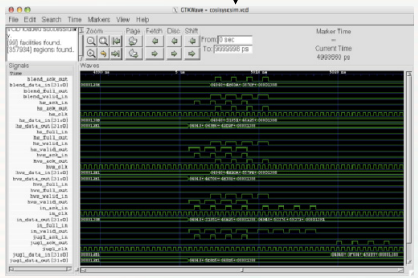
	90nm	65nm	45nm
f_{clk} (GHz)	1.5	2.25	3
V_{supp} (V)	1.2	1.0	0.6
l^*_{crit} (mm)	0.86	4.75	3.47
w^*	90	85	82
E_d (pJ/mm)	0.45	0.2	0.07
P_l(μW/mm)	0.8	1.3	1.2

Synthesis

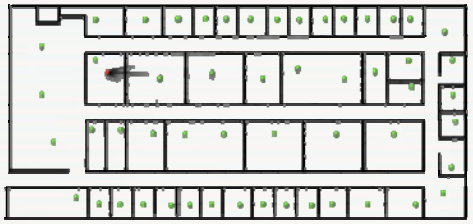
Implementation → Refinement and SystemC generation

Area
Power
N-hops

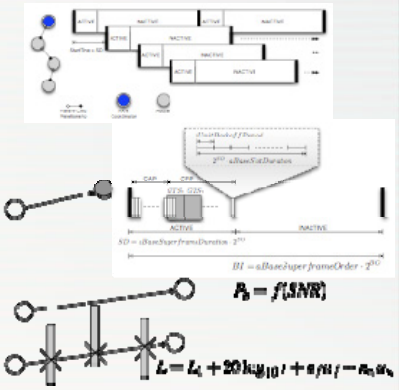




- Building automation networks



$t = 1s,\ l = 0.8s,\ b = 16, p = 10^{-8}$

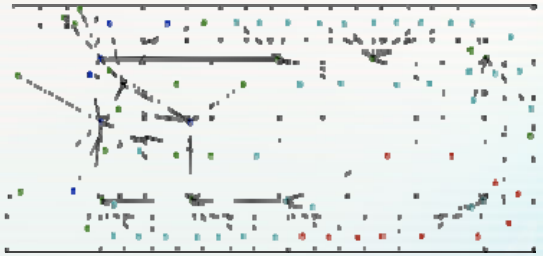


$A_s = f(SNR)$

$L = L_i + 20\log_{10} r + \alpha_{pl} r - \alpha_{sh} s$

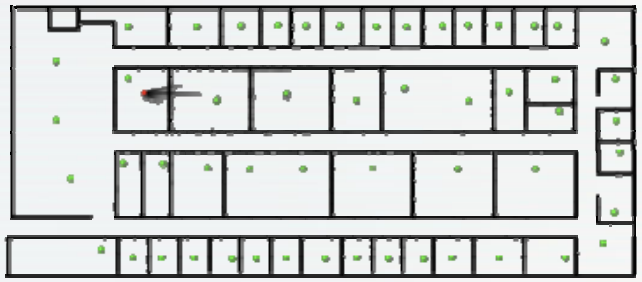
Synthesis

Implementation

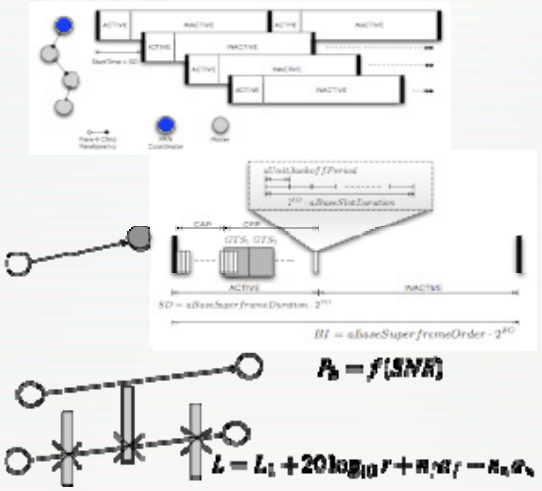


Network load
Network cost (20yr)
Latency and slack

Building automation

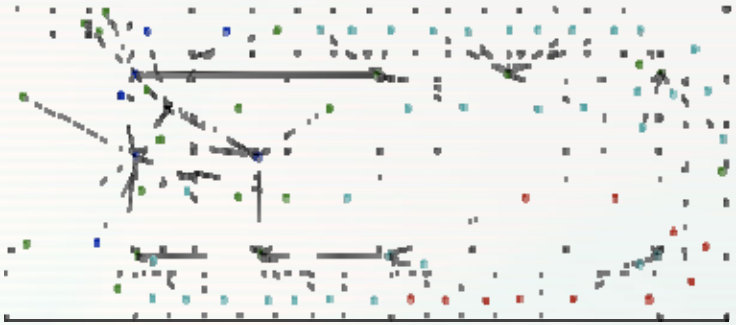


$t = 1s,\ l = 0.5s,\ b = 16, p = 10^{\ t}$



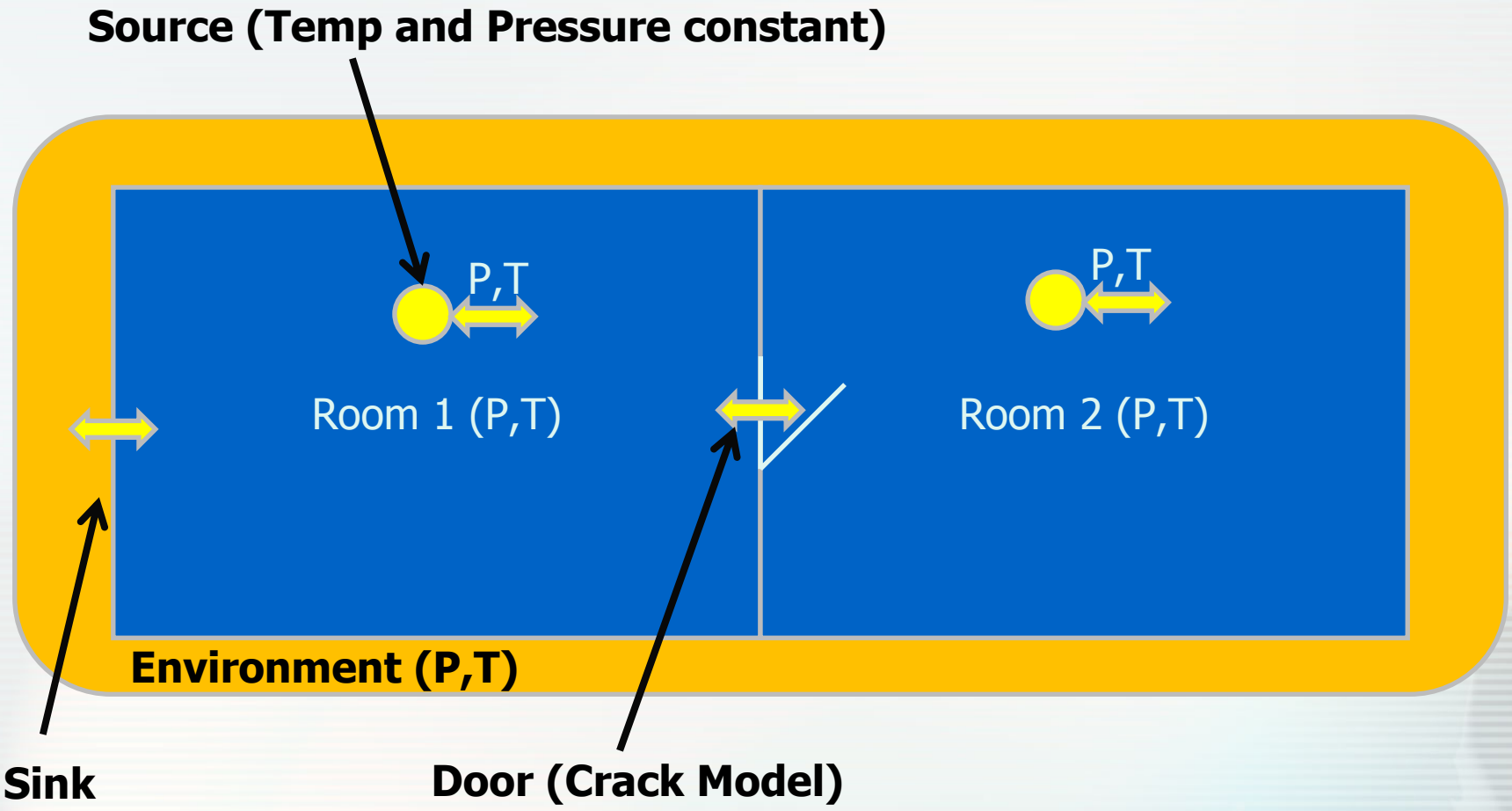
Synthesis

Implementation

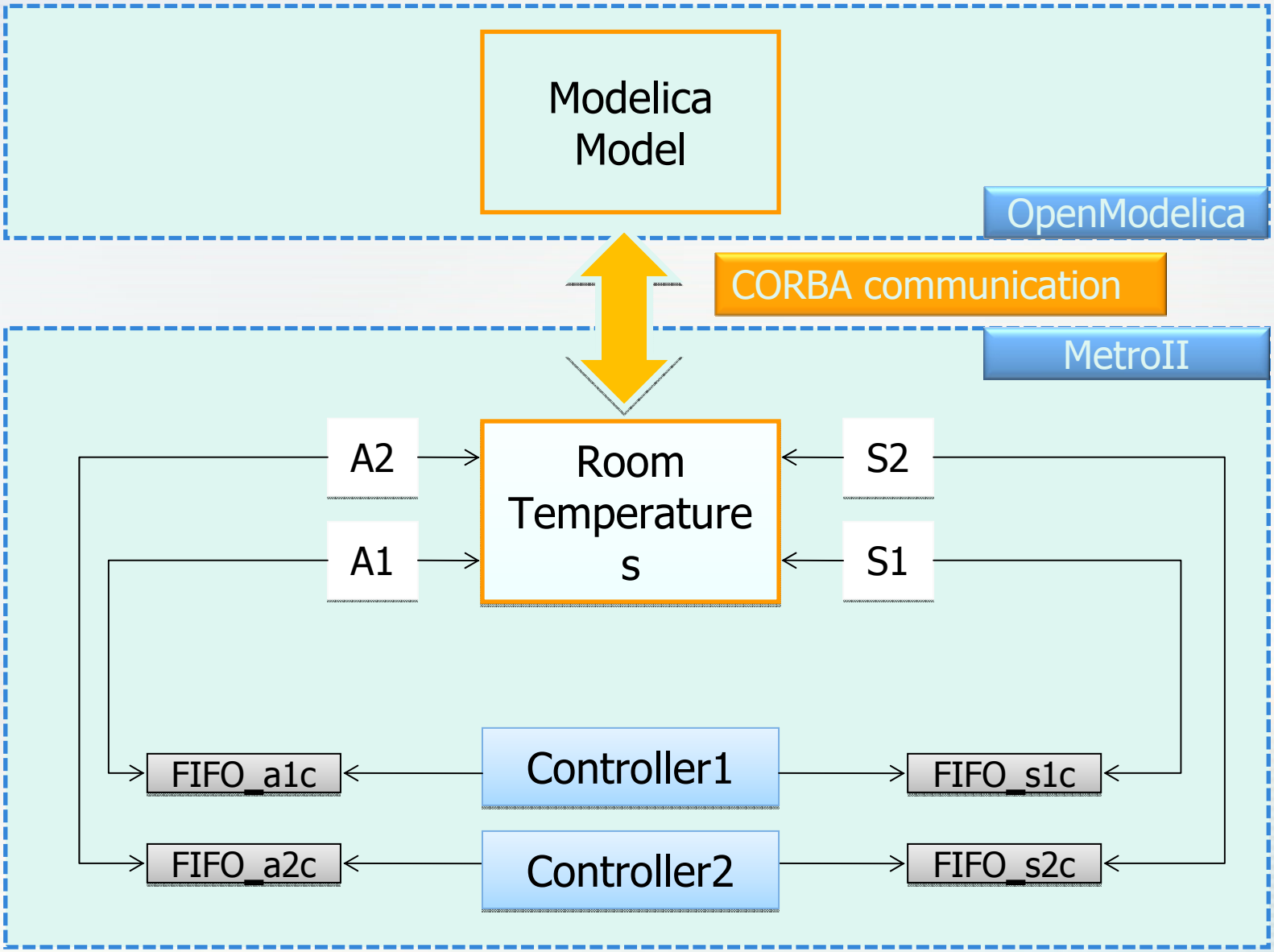


Network load
Network cost (20yr)
Latency and slack

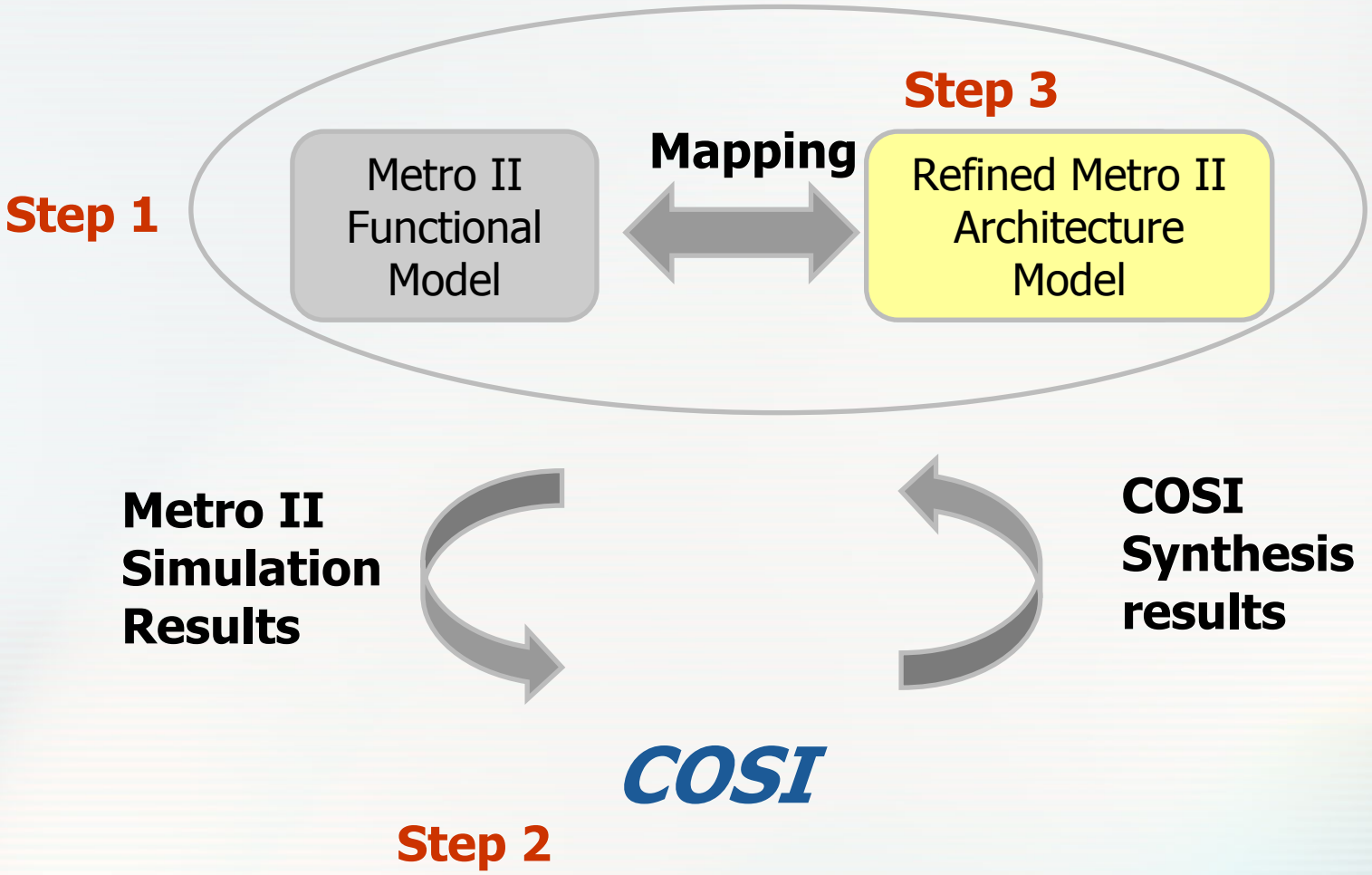
Heating and Cooling in Building Automation



Heating and Cooling Functional Model



COSI and Metro II: Design Flow



Metro II (UCB) to ASPN (UCLA) Flow Overview

Liangpeng Guo (UCB)
Douglas Densmore (UCB)
Qi Zhu (UCB)
Karthik Gururaj (UCLA)