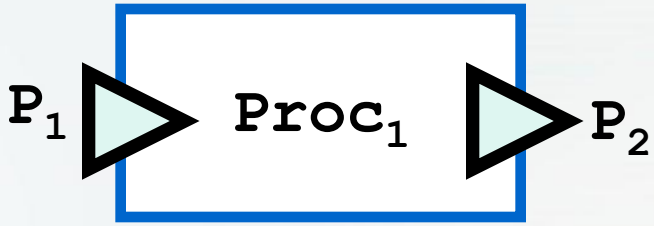
Metropolis Metamodel

Metropolis Objects

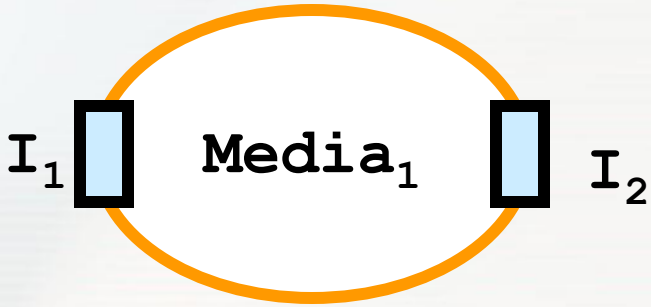
- Metropolis elements adhere to a "separation of concerns" point of view.

- **Processes (Computation)**



Active Objects
Sequential Executing Thread

- **Media (Communication)**



Passive Objects
Implement Interface Services

- **Quantity Managers (Coordination)**

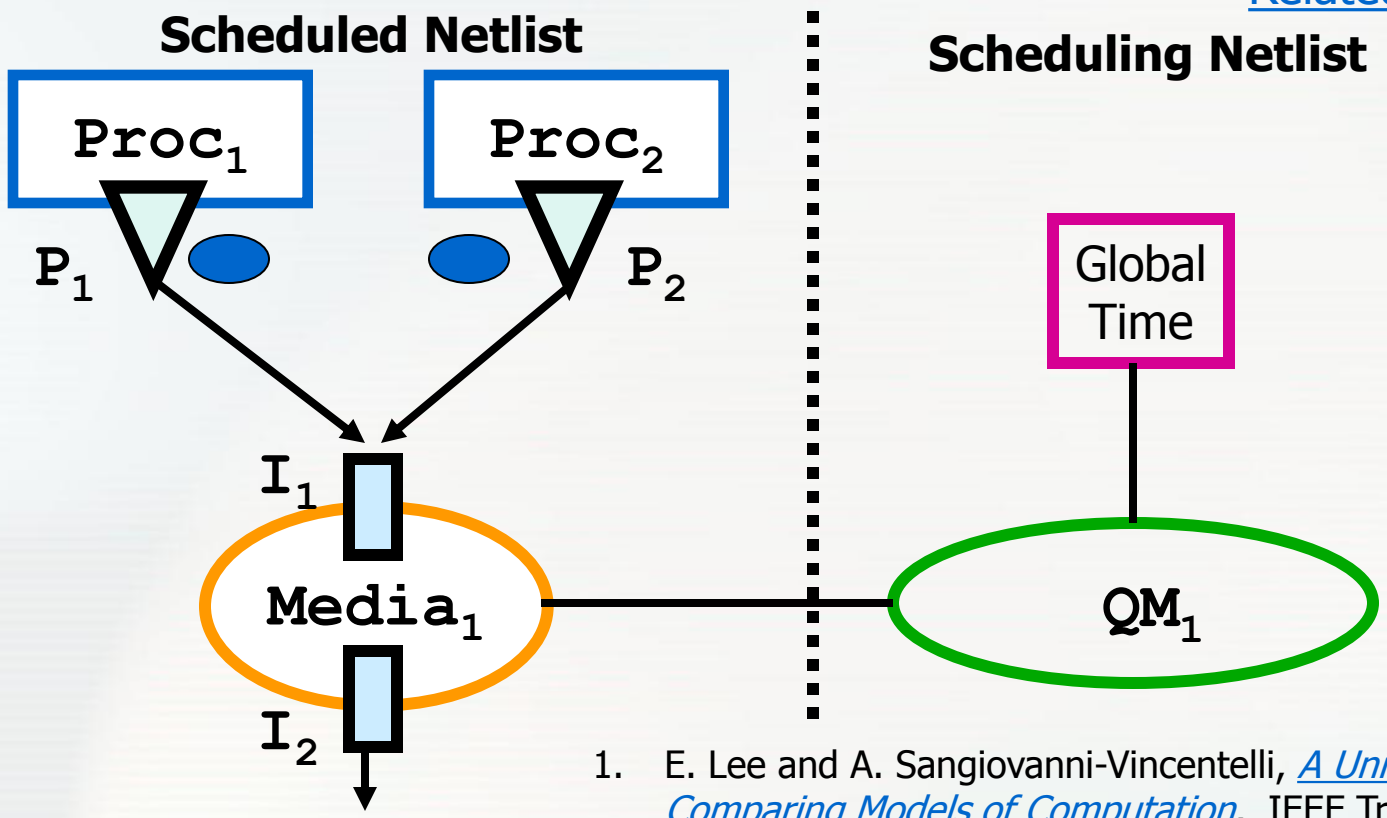
Schedule access to resources and quantities

Metro. Netlists and Events

Metropolis Architectures are created via two netlists:

- Scheduled – generate **events**[1] for services in the scheduled netlist.
- Scheduling – allow these **events** access to the services and **annotate events** with **quantities**.

Related Work



Event[1] – represents a transition in the action automata of an object. Can be annotated with any number of quantities. This allows performance estimation.

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

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

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.

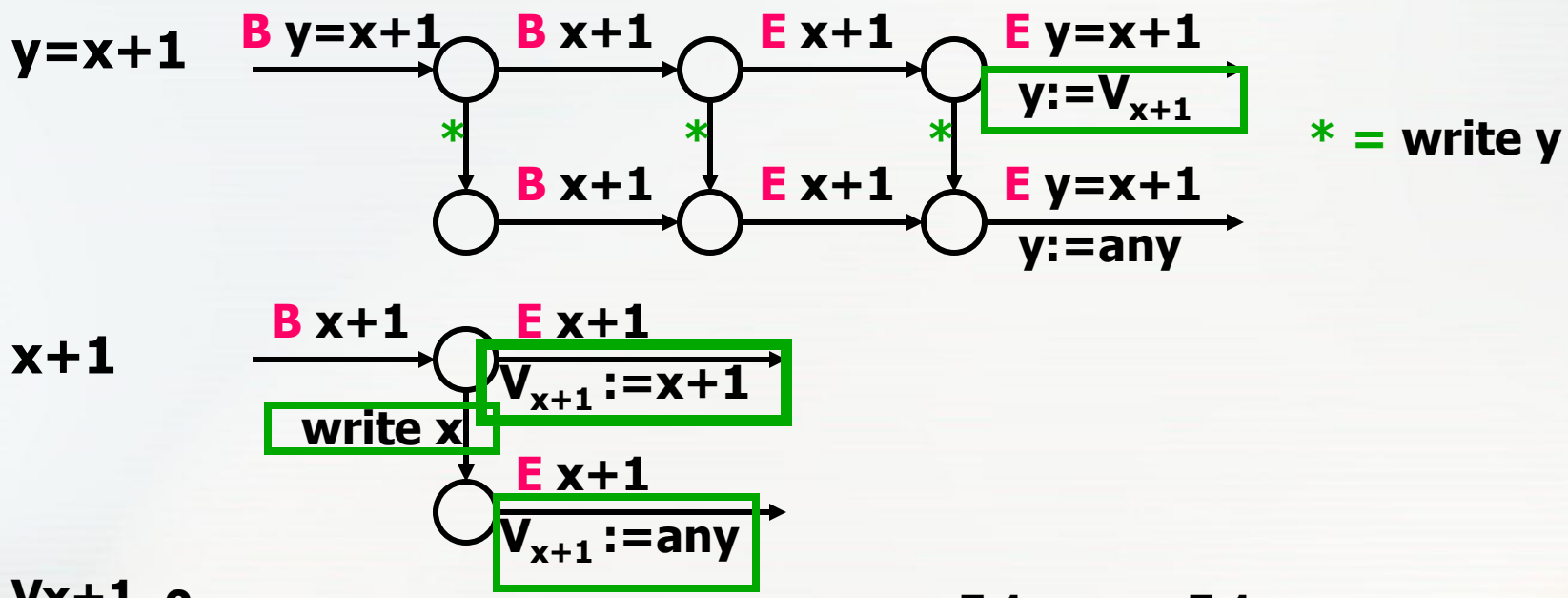
Execution semantics

Action automaton:

- **one for each action of each process**
  - defines the set of sequences of events that can happen in executing the action
- **a transition corresponds to an event:**
  - it may update shared memory variables:
    - **process and media member variables**
    - **values of actions-expressions**
  - it may have guards that depend on states of other action automata and memory variables
- **each state has a self-loop transition with the null N event.**
- **all the automata have their alphabets in common:**
  - transitions must be taken together in different automata, if they correspond to the same event.

Action Automata

- y=x+1;



Vx+1	0	5 1	5 1
y	0	0 0	5 1
x	0	0 0	0 0

B y=x+1 N B x+1 N N E x+1 E y=x+1

Return

Semantics summary

- **Processes run sequential code concurrently, each at its own arbitrary pace.**
- **Read-Write and Write-Write hazards may cause unpredictable results**
  - atomicity has to be explicitly specified.
- **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:

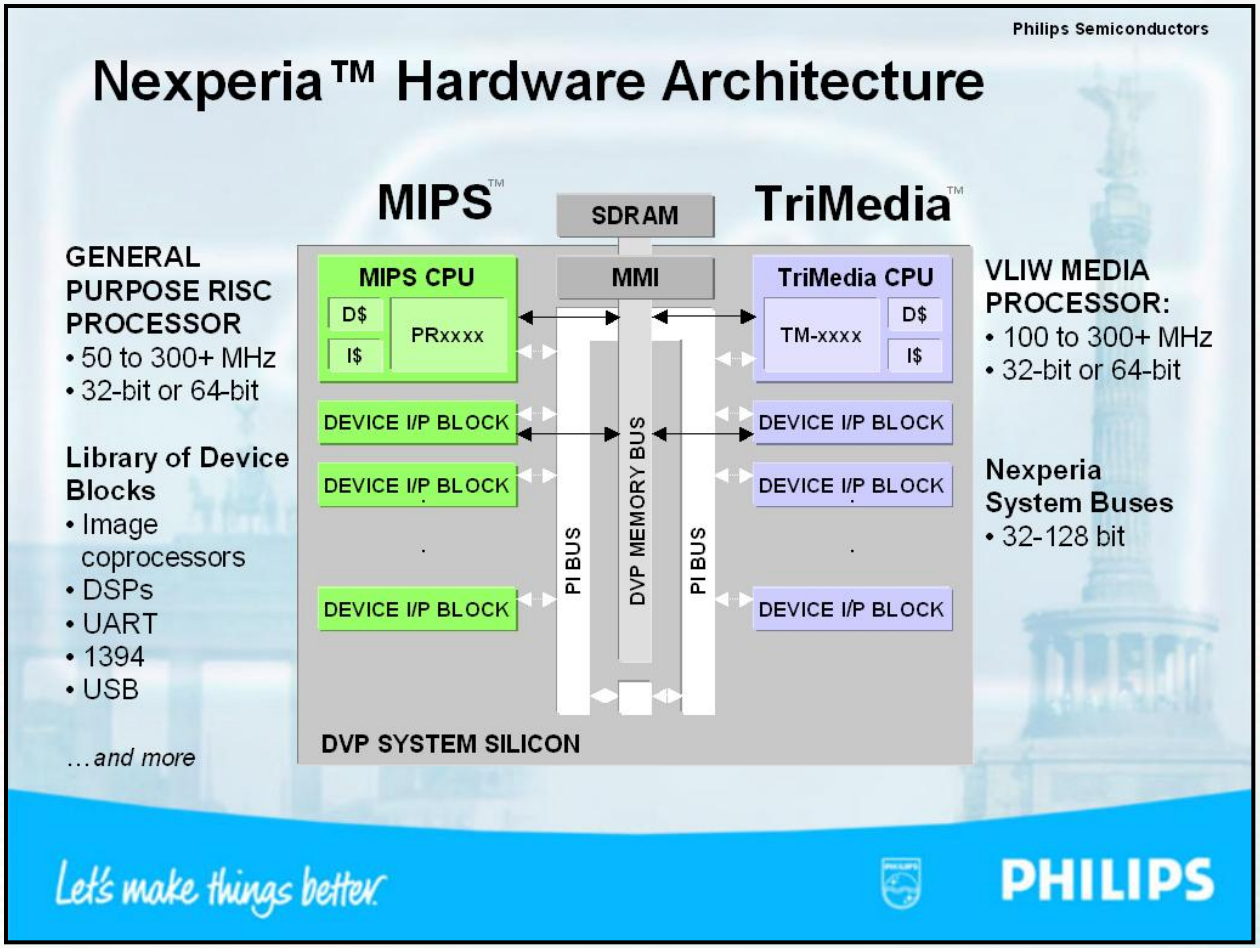
    concurrency, synchronization (awaits and constraints)

Metropolis Architecture Representation

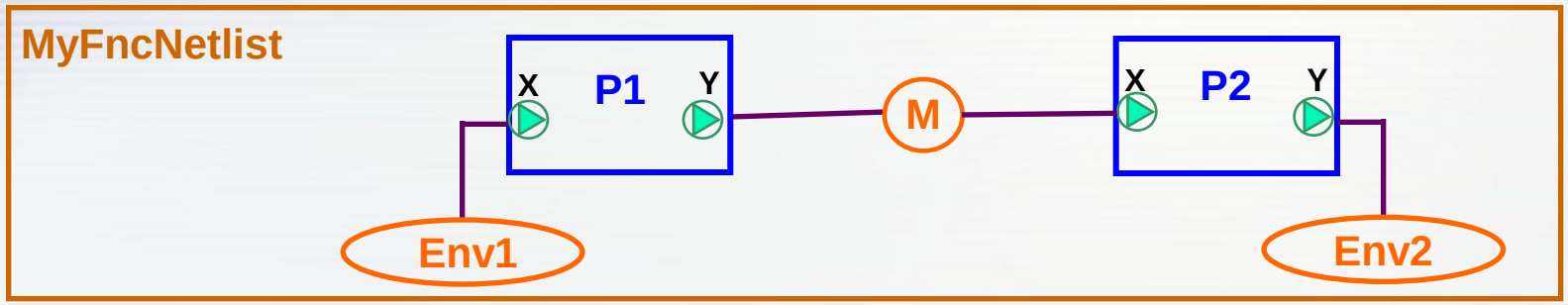
Architecture components

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

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



Meta-Model : Functional Netlist




```
process P{
  port reader X;
  port writer Y;
  thread(){
   while(true){
    ...
    z = f(X.read());
    Y.write(z);
  }}}
```

```
interface reader extends Port{
    update int read();
    eval int n();
}
```

```
interface writer extends Port{
    update void write(int i);
    eval int space();
}
```

```
medium M implements reader, writer{
   int storage;
   int n, space;
   void write(int z){
      await(space>0; this.writer ; this.writer)
            n=1; space=0; storage=z;
    }
    word read(){ ... }
}
```

Meta-Model: Architecture Components

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

- what it *can* do : interfaces
- how much it *costs* : quantities, annotation, logic of constraints

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

```
interface BusArbiterService extends Port {
  update void request(event e);
  update void resolve();
}
```

```
medium Bus implements BusMasterService …{
  port BusArbiterService Arb;
  port MemService Mem; …
  update void busRead(String dest, int size) {
     if(dest== … ) Mem.memRead(size);
   [[Arb.request(B(thisthread, this.busRead));
     GTime.request(B(thisthread, this.memRead),
          BUSCLKCYCLE +
          GTime.A(B(thisthread, this.busRead)));
   ]]
  }
  …
```

```
scheduler BusArbiter extends Quantity
          implements BusArbiterService {
  update void request(event e){ … }
  update void resolve() { //schedule }
}
```

Meta-model: quantities

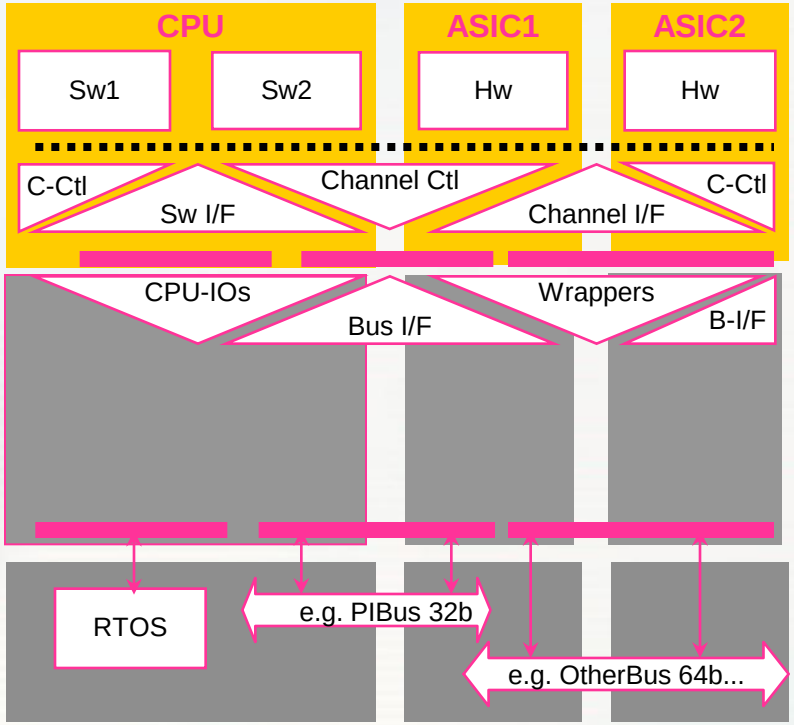
- The domain D of the quantity, e.g. *real* for the global time,
- The operations and relations on D, e.g. subtraction, <, =,
- The function from an event instance to an element of D,
- Axioms on the quantity, e.g.

  the global time is non-decreasing in a sequence of vectors of any feasible execution.

```
class GTime extends Quantity {
        double t;
        double sub(double t2, double t1){...}
        double add(double t1, double t2){...}
        boolean equal(double t1, double t2){ ... }
        boolean less(double t1, double t2){ ... }
        double A(event e, int i){ ... }
        constraints{
          forall(event e1, event e2, int i, int j):
          GXI.A(e1, i) == GXI.A(e2, j) -> equal(A(e1, i), A(e2, j)) &&
          GXI.A(e1, i) < GXI.A(e2, j) -> (less(A(e1, i), A(e2, j)) ||
    equal(A(e1, i), A(e2. j)));
}}
```

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: *on-going research*



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

Quantity resolution

The 2-step approach to resolve quantities at each state of a netlist being executed:

1. **quantity requests**

   for each process *Pi*, for each event *e* that *Pi can* take, find all the quantity constraints on *e*.

   In the meta-model, this is done by explicitly requesting quantity annotations at the relevant events, i.e. Quantity.request(event, requested quantities).

2. **quantity resolution**

   find a vector made of the candidate events and a set of quantities annotated with each of the events, such that the annotated quantities satisfy:

   - **all the quantity requests, and**
   - **all the axioms of the Quantity types.**

   In the meta-model, this is done by letting each Quantity type implement a resolve() method, and the methods of relevant Quantity types are iteratively called.

   - **theory of fixed-point computation**

Quantity resolution

- **The 2-step approach is same as how schedulers work, e.g. OS schedulers, BUS schedulers, BUS bridge controllers.**
- **Semantically, a scheduler can be considered as one that resolves a quantity called *execution index.***
- **Two ways to model schedulers:**

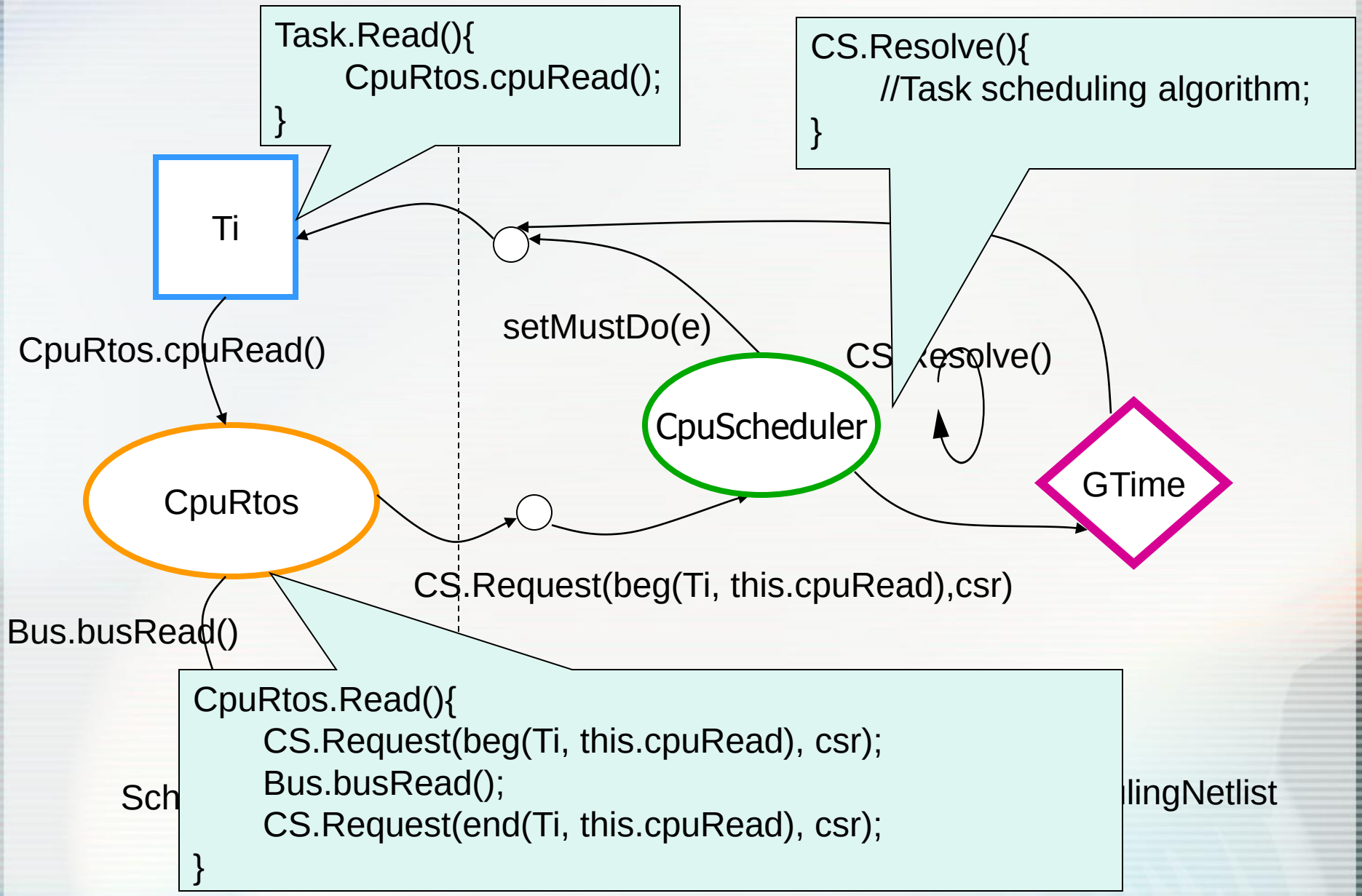
1. As processes:

- **explicitly model the scheduling protocols using the meta-model building blocks**
- **a good reflection of actual implementations**

2. As quantities:

- **use the built-in request/resolve approach for modeling the scheduling protocols**
- **more focus on resolution (scheduling) algorithms, than protocols: suitable for higher level abstraction models**

Quantity Request – Service



Meta-Model: Mapping Netlist

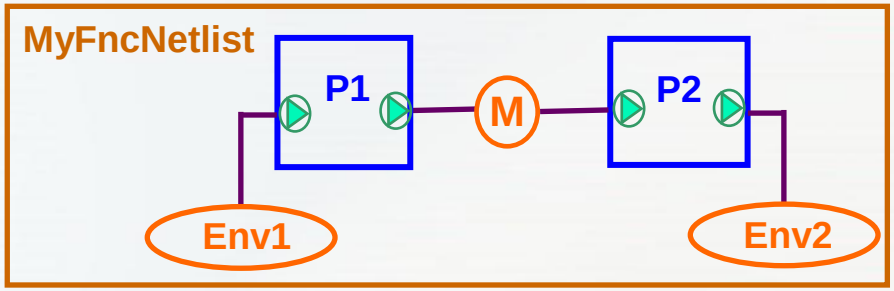
MyMapNetlist

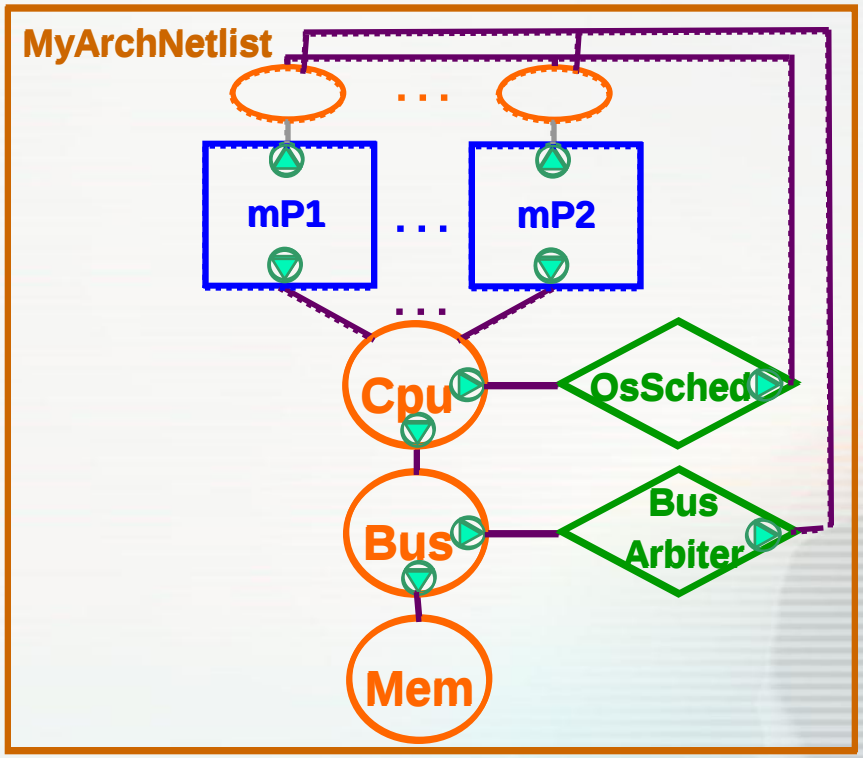
B(P1, M.write) <=> B(mP1, mP1.writeCpu); E(P1, M.write) <=> E(mP1, mP1.writeCpu);

B(P1, P1.f) <=> B(mP1, mP1.mapf); E(P1, P1.f) <=> E(mP1, mP1.mapf);

B(P2, M.read) <=> B(P2, mP2.readCpu); E(P2, M.read) <=> E(mP2, mP2.readCpu);

B(P2, P2.f) <=> B(mP2, mP2.mapf); E(P2, P2.f) <=> E(mP2, mP2.mapf);





Architecture Modeling Related Work

1. David C. Luckham and James Vera, *An Event-Based Architecture Definition Language* , IEEE Transactions on Software Engineering, Vol. 21, No 9, pg. 717-734, Sep. 1995.

2. Ingo Sander and Axel Jantsch, *System Modeling and Transformational Design Refinement in ForSyDe*, IEEE Transactions on CAD, Vol. 23, No 1, pg. 17-32, Jan. 2004.

3. Paul Lieverse, Pieter van der Wolf, Ed Deprettere, and Kees Vissers, *A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems*, IEEE Workshop in Signal Processing Systems, Taipei, Taiwan, 1999.

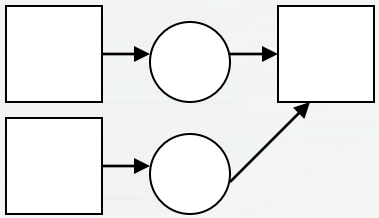
Return

	Metropolis	Rapide[1]	ForSyDe[2]	SPADE[3]
Mapping	x	x	x	x
Quantity Managers	x	No	No	No; collectors in bldg blocks
Event Based	x	x	x	No
Pure Architecture Model	x	x	No; Functional tied to Arch.	x

Naïve Approach

System Level Design does not guarantee accuracy or efficiency!!

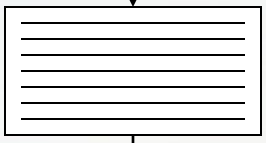
Abstract
Modular
SLD



Estimated Performance Data

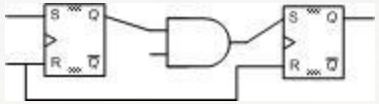
Manual

"C" Model



Manual

RTL "Golden Model"



Implementation

Disconnected
Inaccurate!

Lengthy Feedback
Inefficient
Miss **Time to Market!**

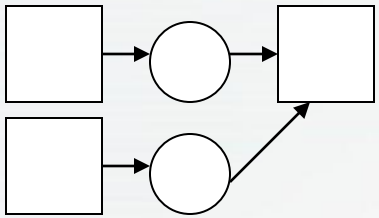
Implementation Gap!

Improved Approach

Technique 1: Modeling style and characterization for programmable platforms

Functional level blocks of programmable components

Abstract
Modular
SLD

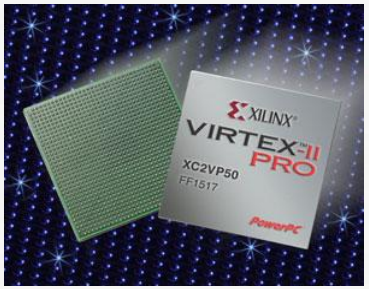


Real Performance Data

From characterization flow

Narrow the Gap

Actual Programmable Platform Description



New approach has improved **accuracy** and **efficiency** by relating **programmable devices** and their **tool flow** with **SLD (Metropolis)**. Retains **modularity** and **abstraction**.

Goals for Metro II

- **Import heterogeneous IP**
  - **Different languages**
  - **Different models of computation**
- **Key Platform-based Design Activities**
  - **Behavior-Performance Separation**
    - **Quickly change performance characteristics of models**
  - **Design Space Exploration**
    - **Relate functionality and architecture**
    - **Verify relationships between different abstraction levels**

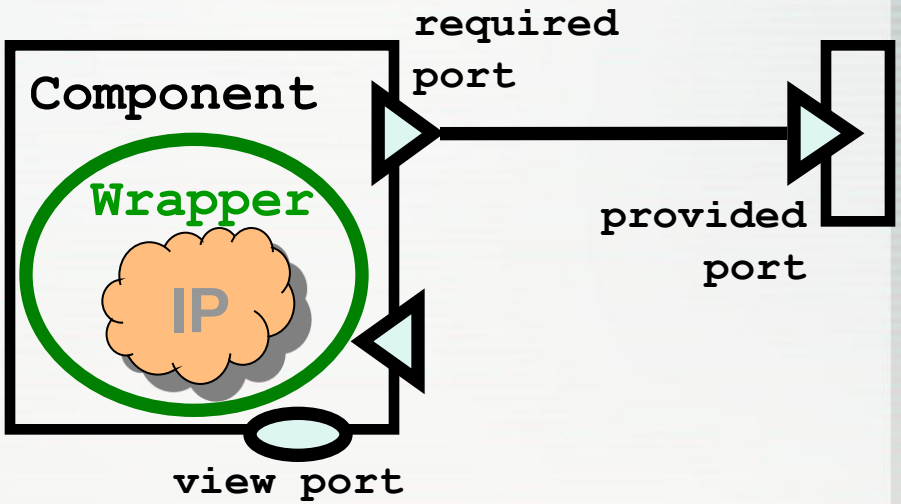
Coordination Framework

3-Phase Execution

Event-oriented Framework

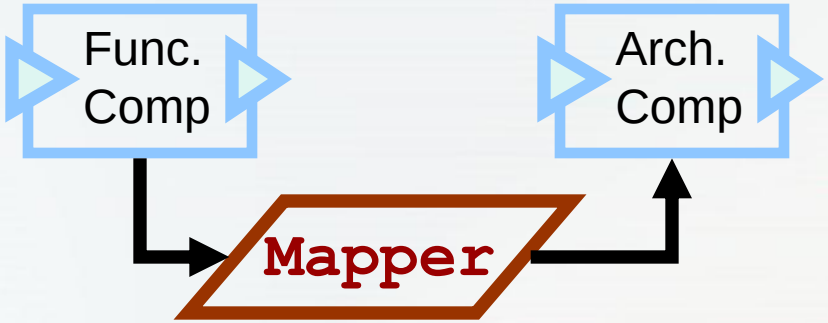
Components, Ports, and Connections

- 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

Mappers

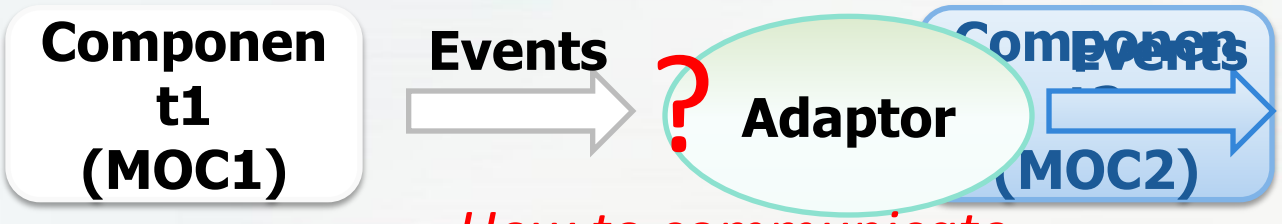


- **Enable Mapping at the component level**
  - **Between components with compatible interfaces**
  - **Possibly many functional components mapped to a single architectural component**

- **Mappers are objects that help specify the mapping**
  - **Bridge syntactic gaps only**
  - **E.g. Missing method parameters**

Adaptor

- **Bridge different models of computation (MoCs)**



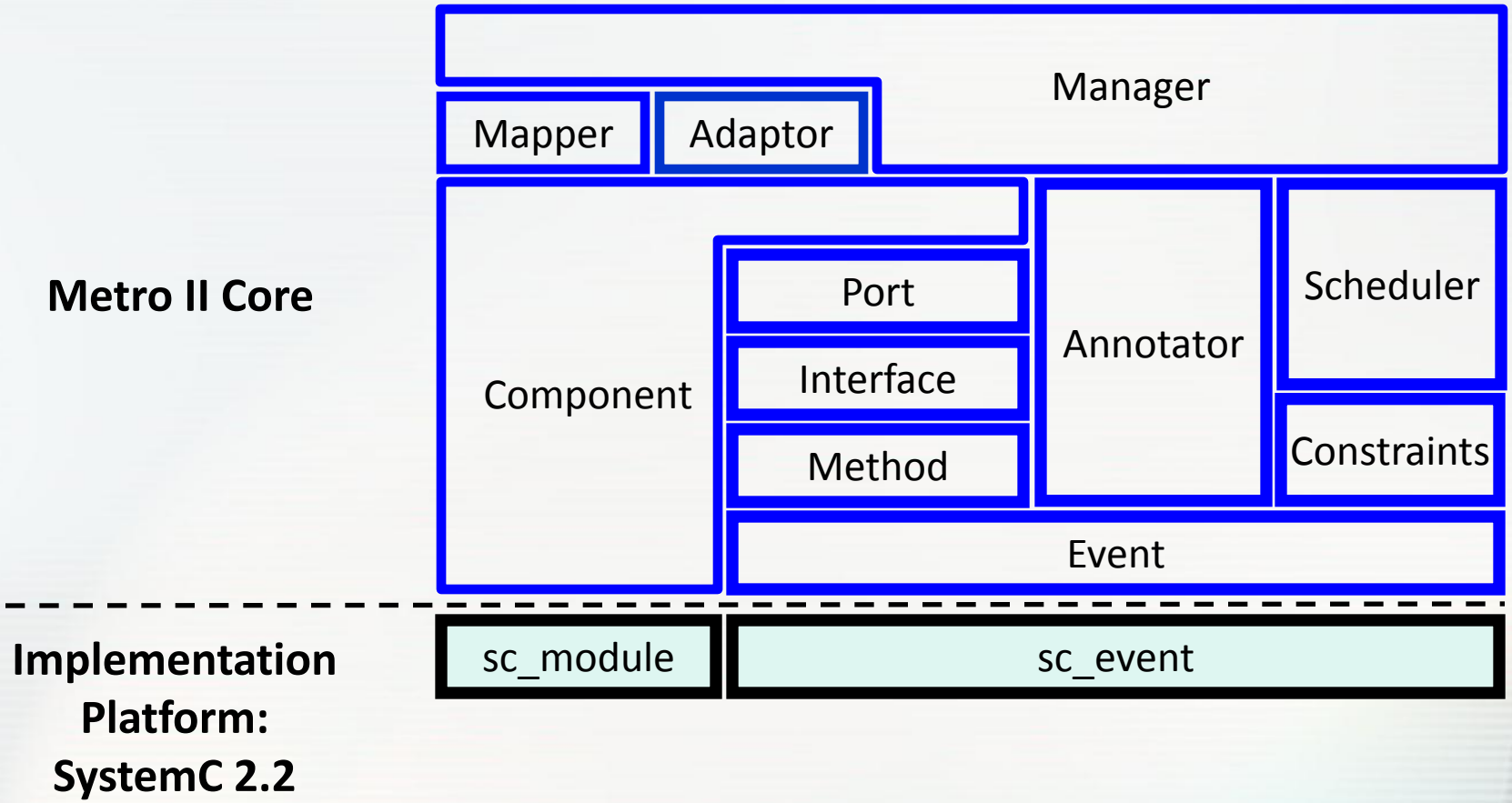
How to communicate with different MoC?

- Adaptor transforms the tags of the events to make different MoCs compatible
  - Values are not changed
  - Will not produce/discard events

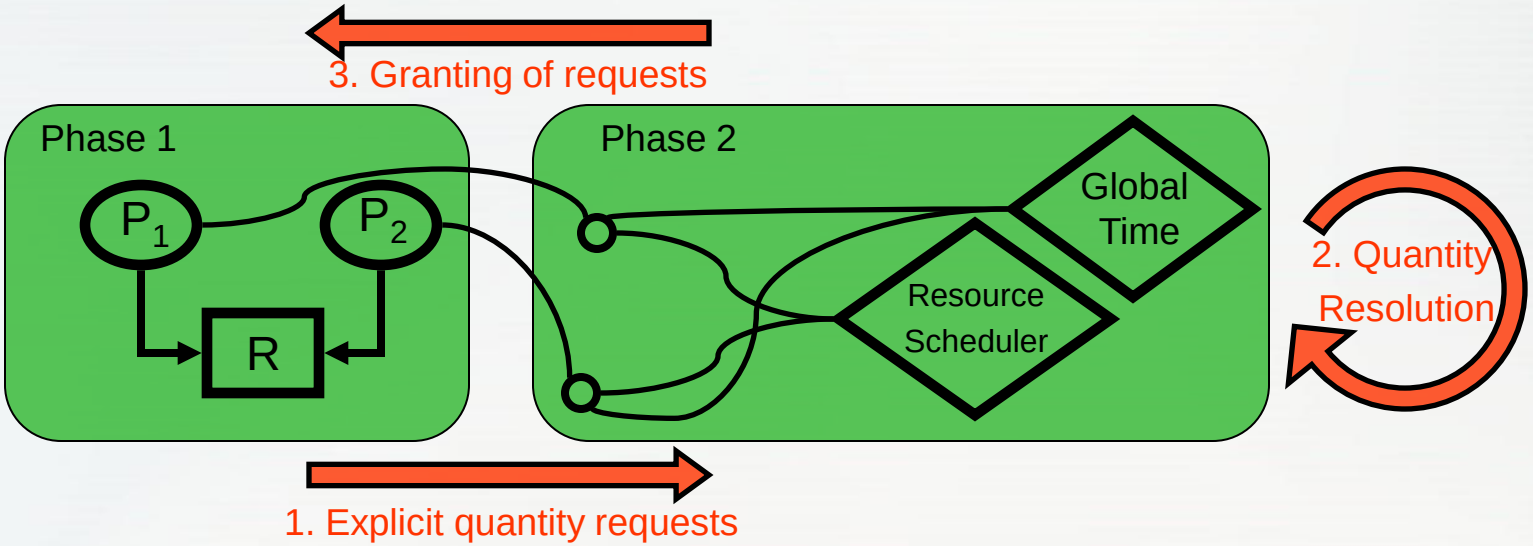
Implementation of Adaptor

- Adaptor contains internal channels for storing the information of events, and a process to transforms the tags of events
- Adaptor will be executed during the base model execution phase (phase 1)
- Test case with an adaptor between dataflow and FSM semantics
- Further tested in the cruise control and heating and cooling project

Metro II System Architecture Status



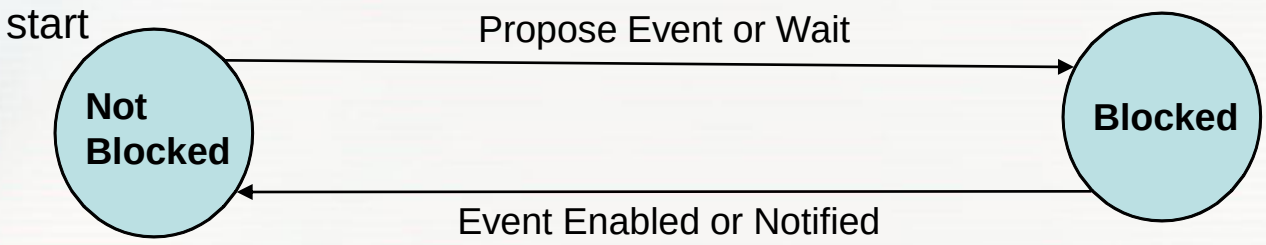
Behavior-Performance Separation in Metropolis



- **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**

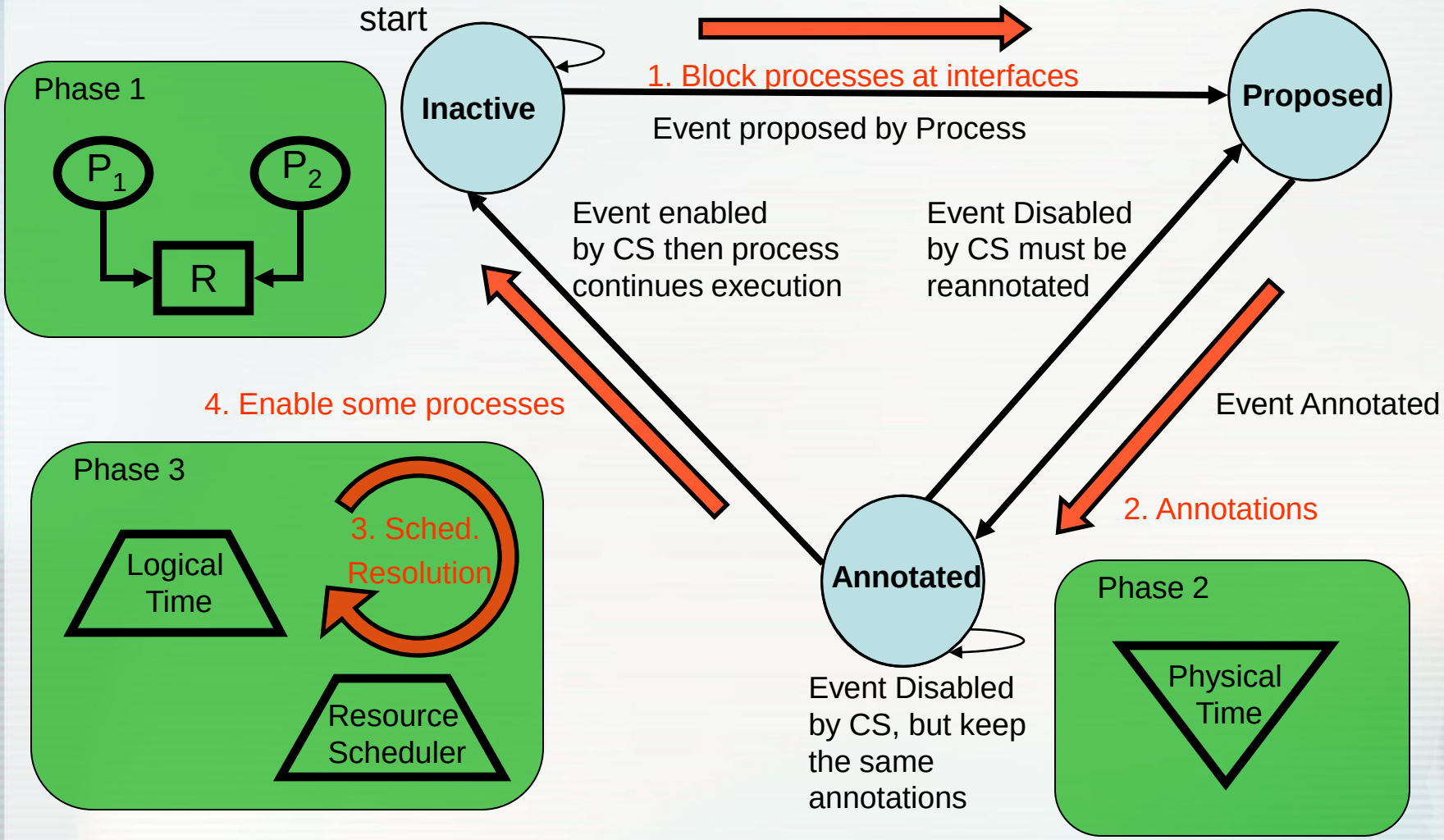
Execution Semantics in Metro II

- **Metro II components (imperative code) are run by processes (sequential thread of execution).**



Metro II Process States

Execution Semantics in Metro II



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	

Assumptions

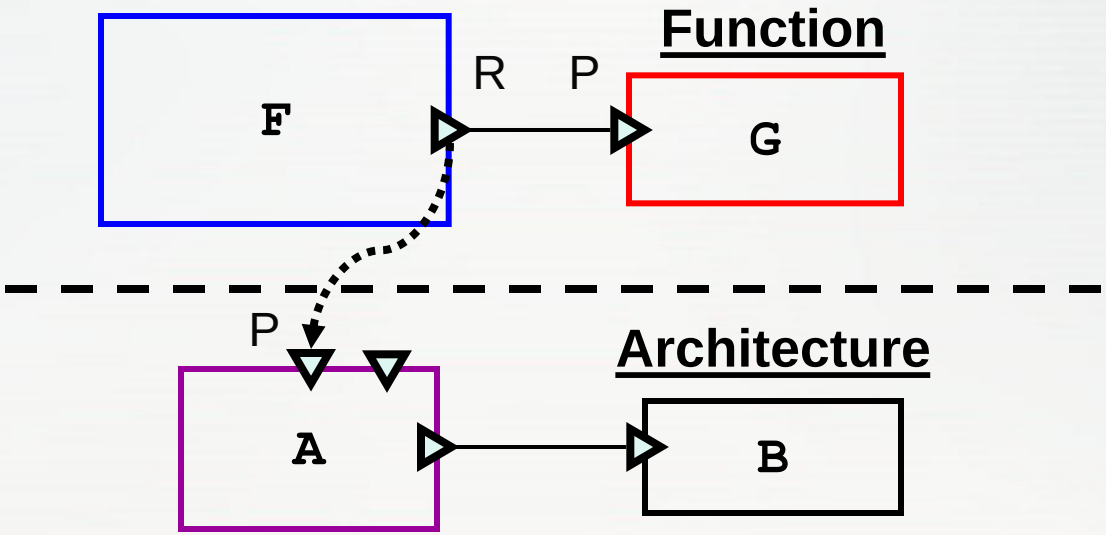
- "Blocking"
  - Both the architectural and functional models should be allowed to block
- Scheduling
  - Functional model execution is valid (i.e. doesn't deadlock)
- Mapping
  - The enabling of events in one model, correspond directly to the enabling of other events

Mapping

- **Mapping in Metro II requires:**
  - **Assigning functional operations to architecture services. Many-to-one relationship.**
    - **This is done through events.**
- **Issues to resolve:**
  - **Which types and in what order should events be related between function and architecture?**
  - **How processes present in the functional model trigger architectural components? How does simulation execution originate?**

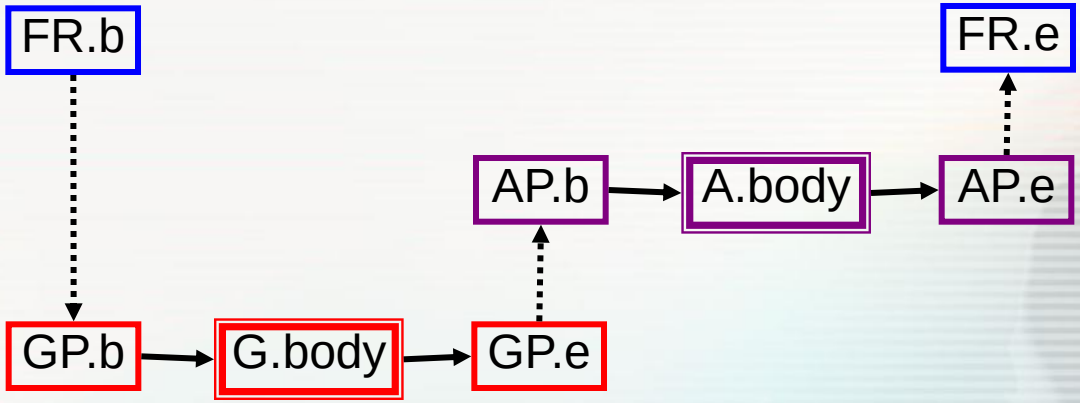
Proposal 1

Functional model initiates execution and is followed by the architecture model.



- Port Mapping Conventions
  - Required to Provided

- Call graph Example

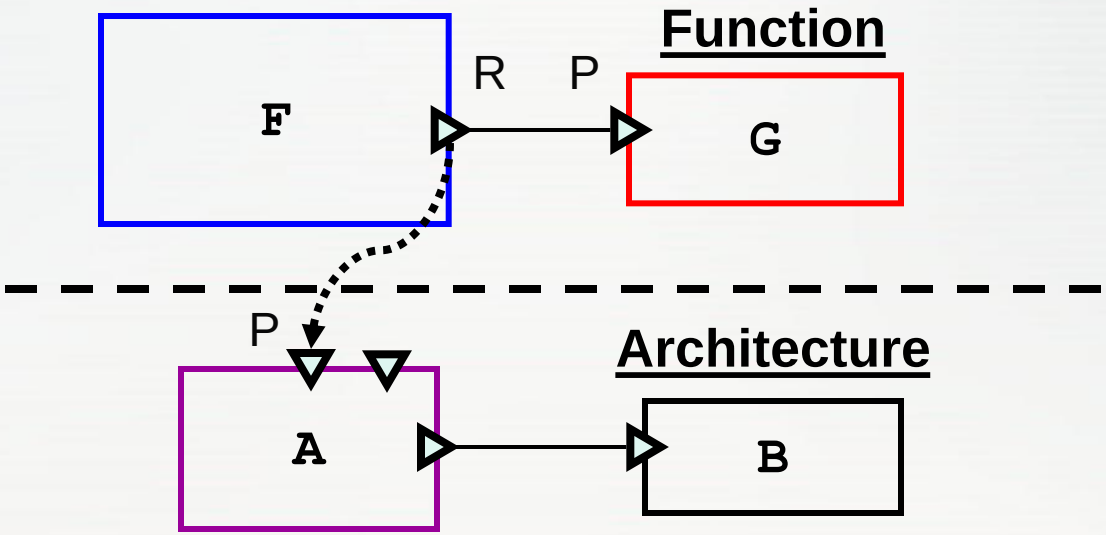


Key

Synchronized Events - - -
Direct Event Ordering ___

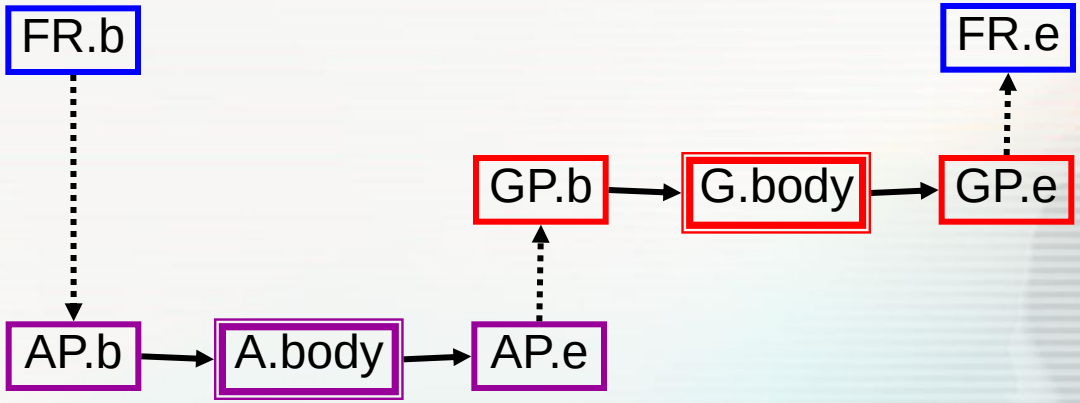
Proposal 2

Architectural model initiates execution and is followed by the functional model.



- Port Mapping Conventions
  - Required to Provided

- Call graph Example

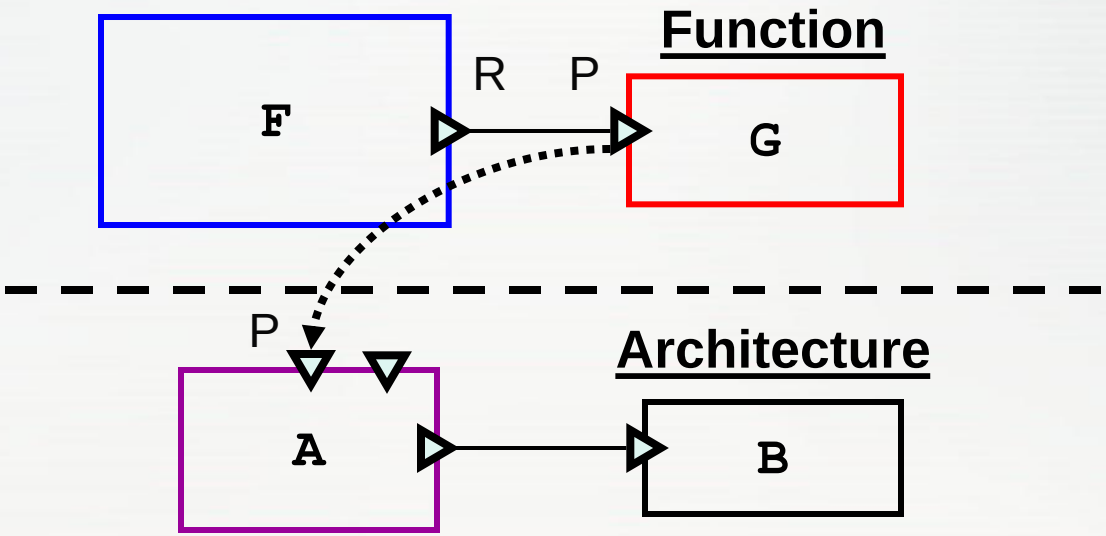


Key

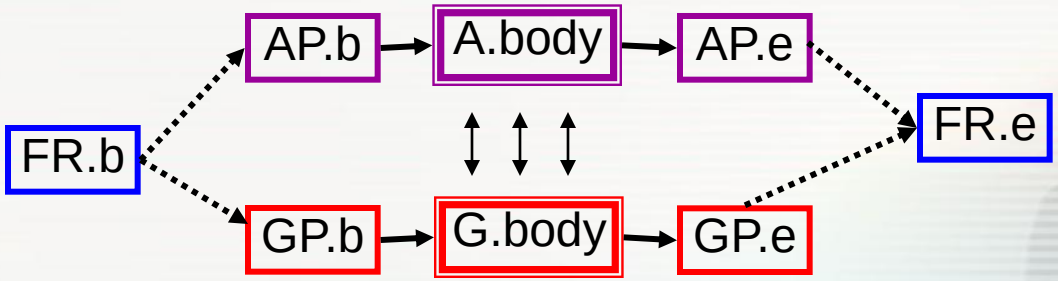
Synchronized Events - - -
Direct Event Ordering __

Proposal 3

Functional and architectural model execute concurrently.



- Port Mapping Conventions
  - Provided to Provided



- Call graph Example

Key

Synchronized Events - - -
Direct Event Ordering ___

Key Points of Proposals

- **Proposal 1 – Functional model execution cannot be determined by architectural state.**
- **Proposal 2 – Architecture model must block if the functionality blocks.**
- **Proposal 3 – Requires that the component's execution be granular enough to support explicit synchronization opportunities (i.e. protocols).**

Mapping Granularity Tradeoff

- **Granularity changes may be needed to support proposal 3.**
- **The functional and architectural models need not have the same level of granularity.**

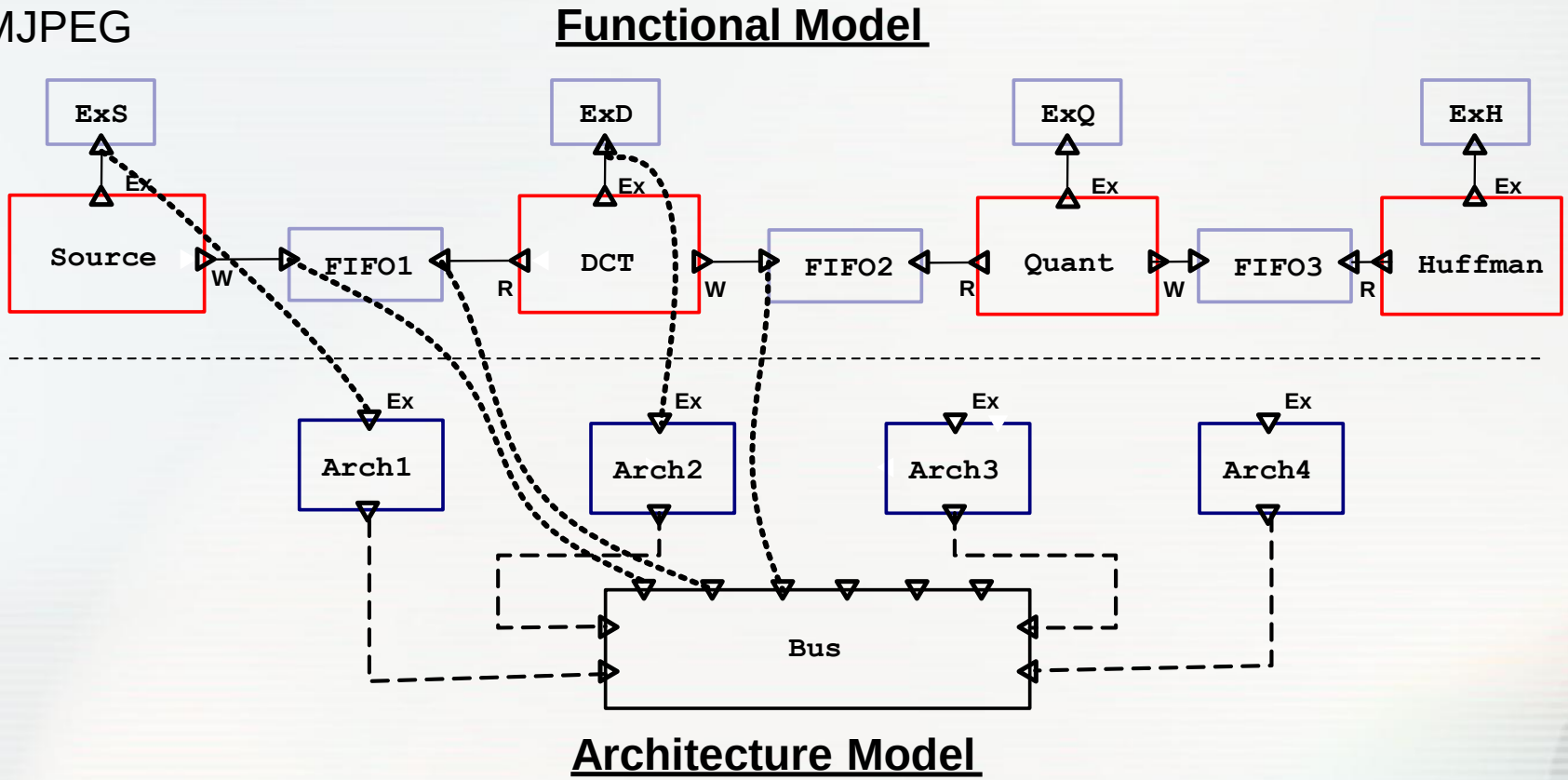
FIFO READ Begin

1. **Grab bus access**
2. **Read fifo status**
3. **If it can proceed to read/write**
   Read/Write; release bus
4. **Else**
   Release bus; wait a random number of cycles; goto 1

FIFO READ END

Example Design Scenario

MJPEG



Shared FIFO is another design scenario

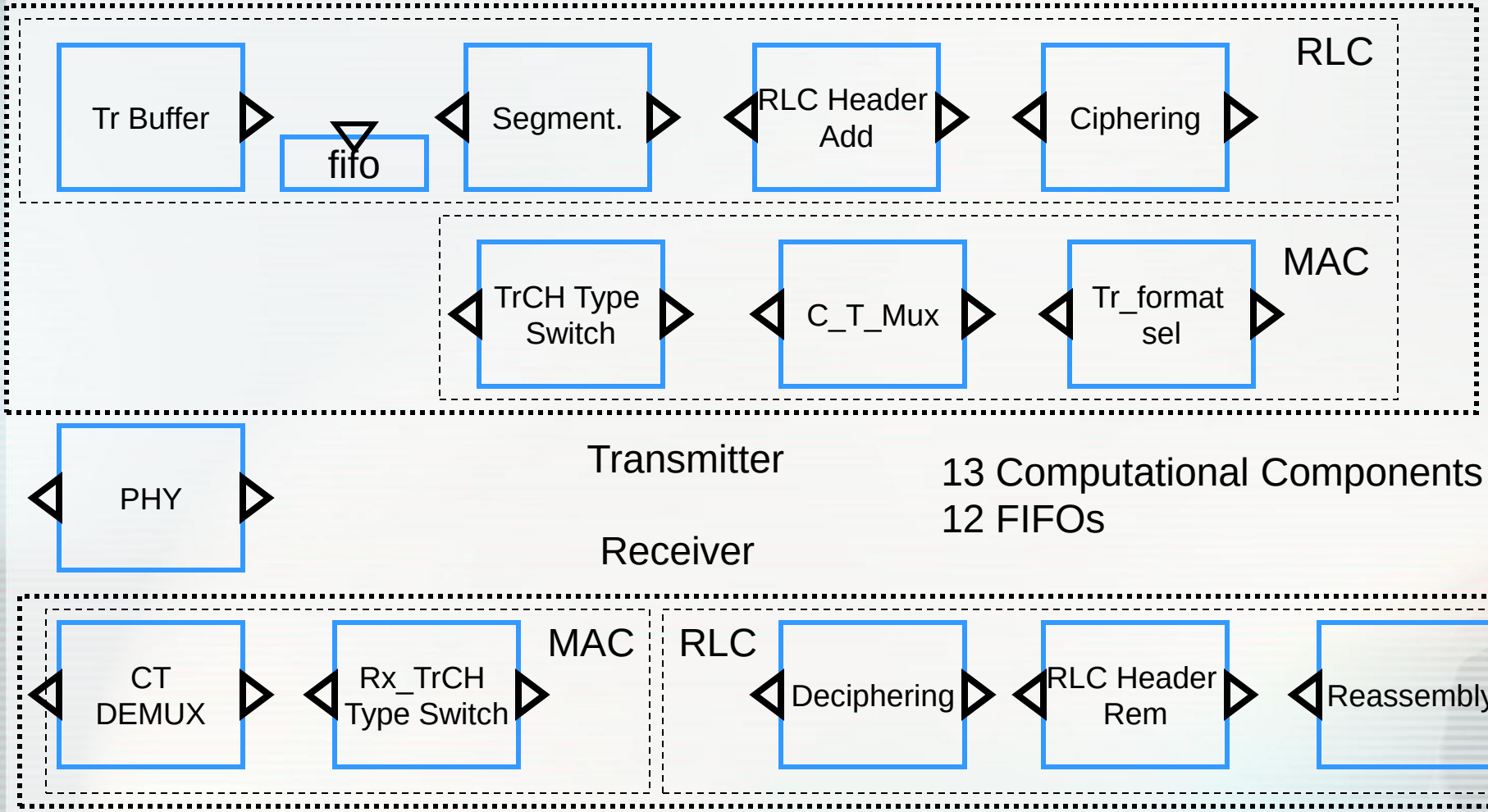
Metro II Mapping Conclusions

- Metro II mapping uses events to synchronize execution between the functional and architectural model.
- Potential tradeoffs in granularity and expressiveness depend on the mapping style (Metro II supports various).
- Established a style to describe Metro II execution and started a set of design scenarios to discuss the tradeoffs.

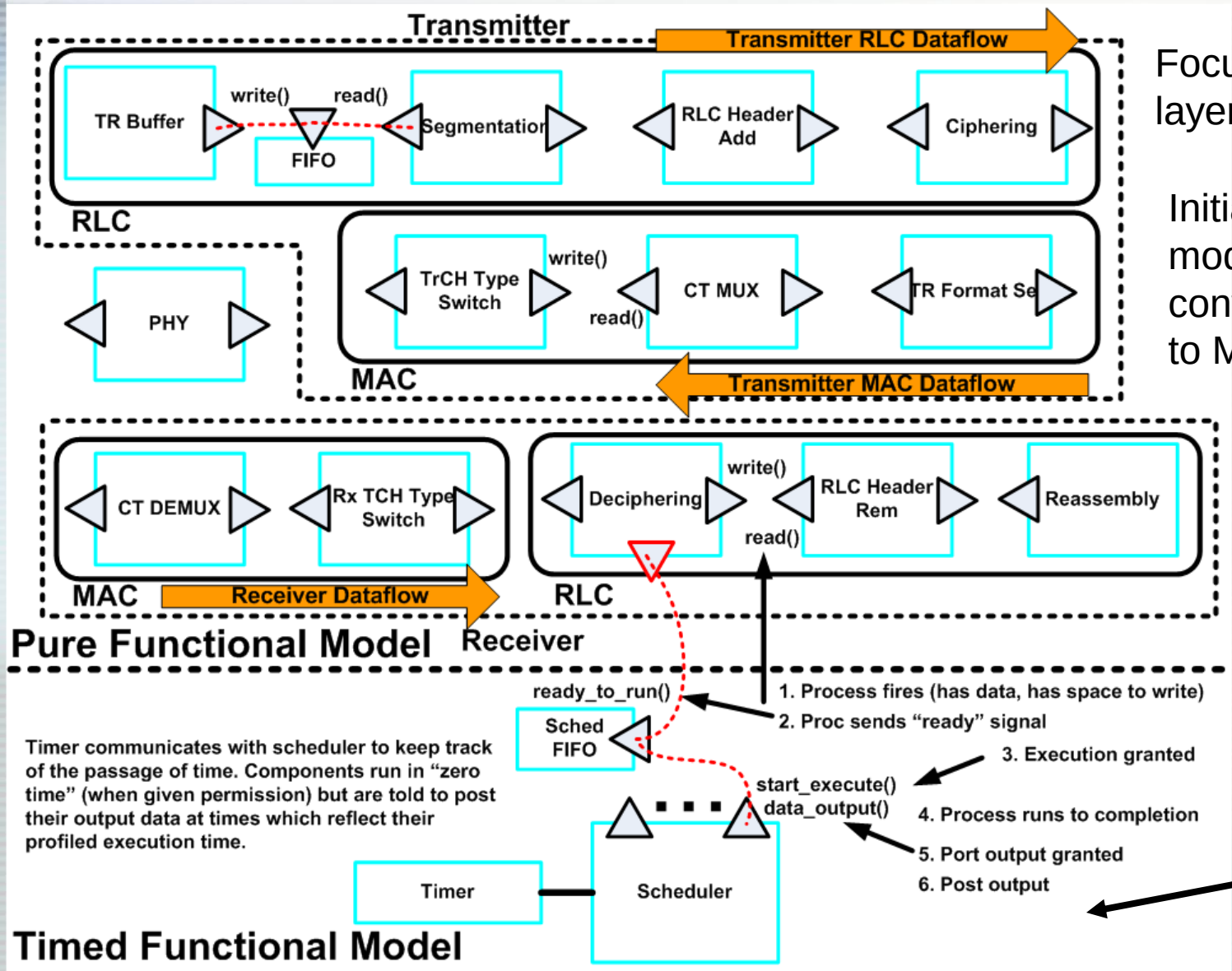
Design Activity: UMTS Case Study

- 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

UMTS DLL Function Model



Metro II UMTS Models

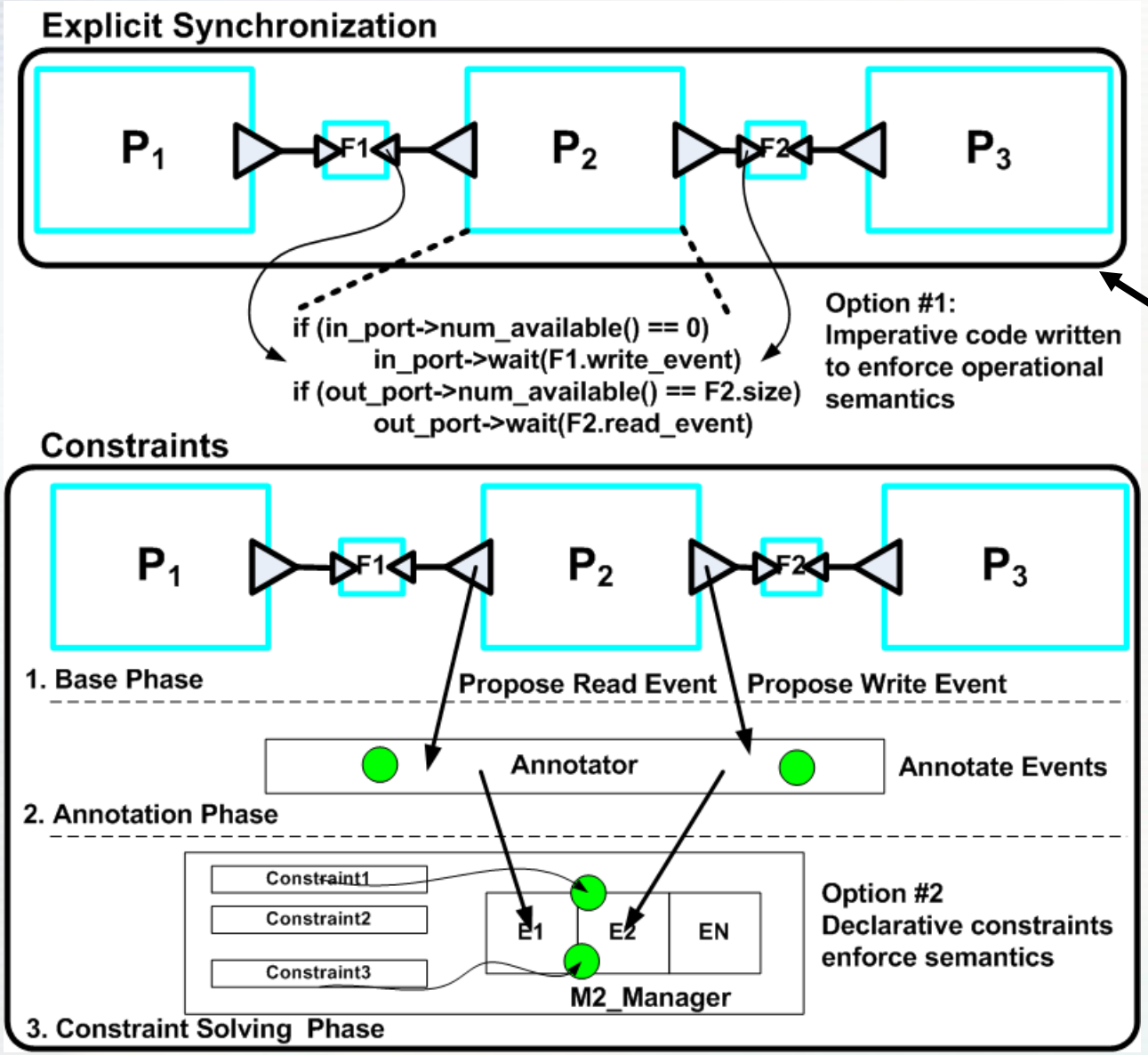


Focused on the DLL layer

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



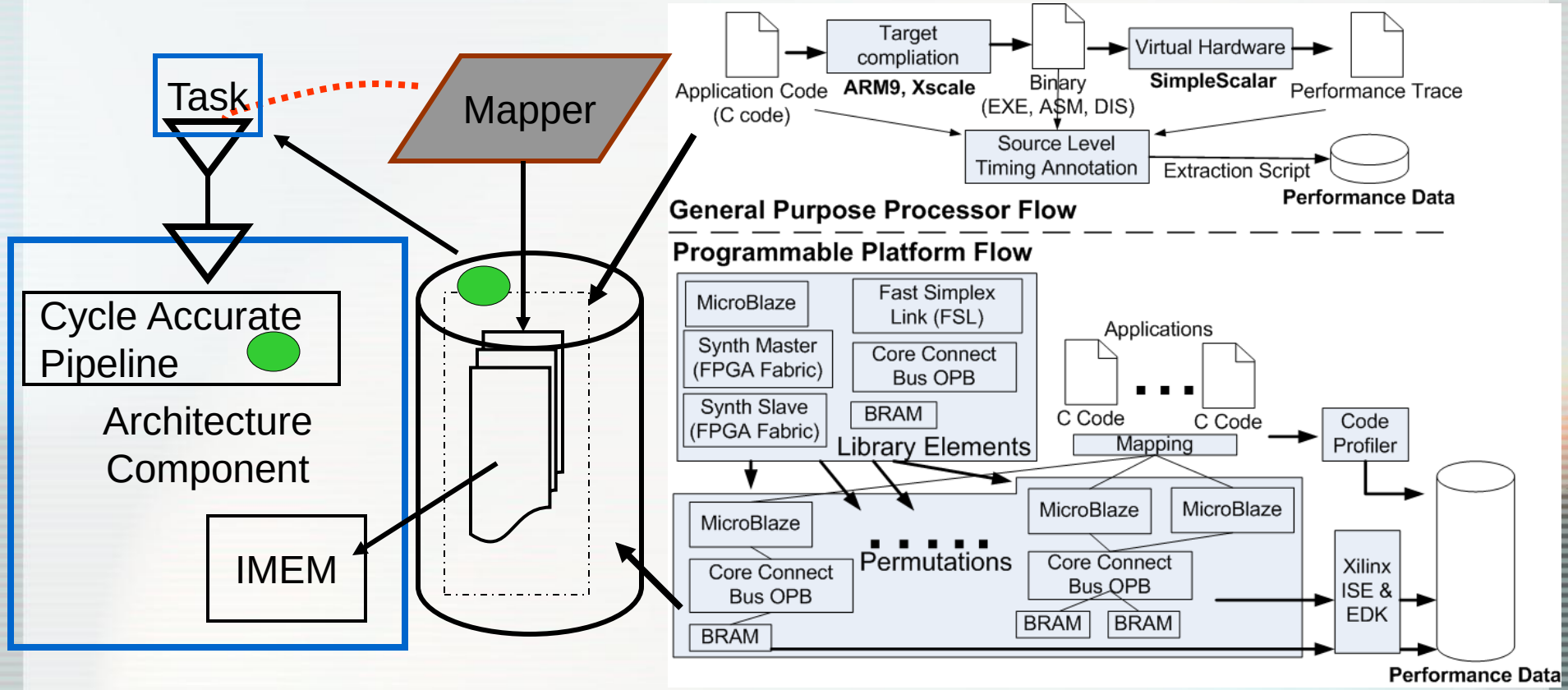
UMTS example exposed two approaches to synchronization in Metro II:

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

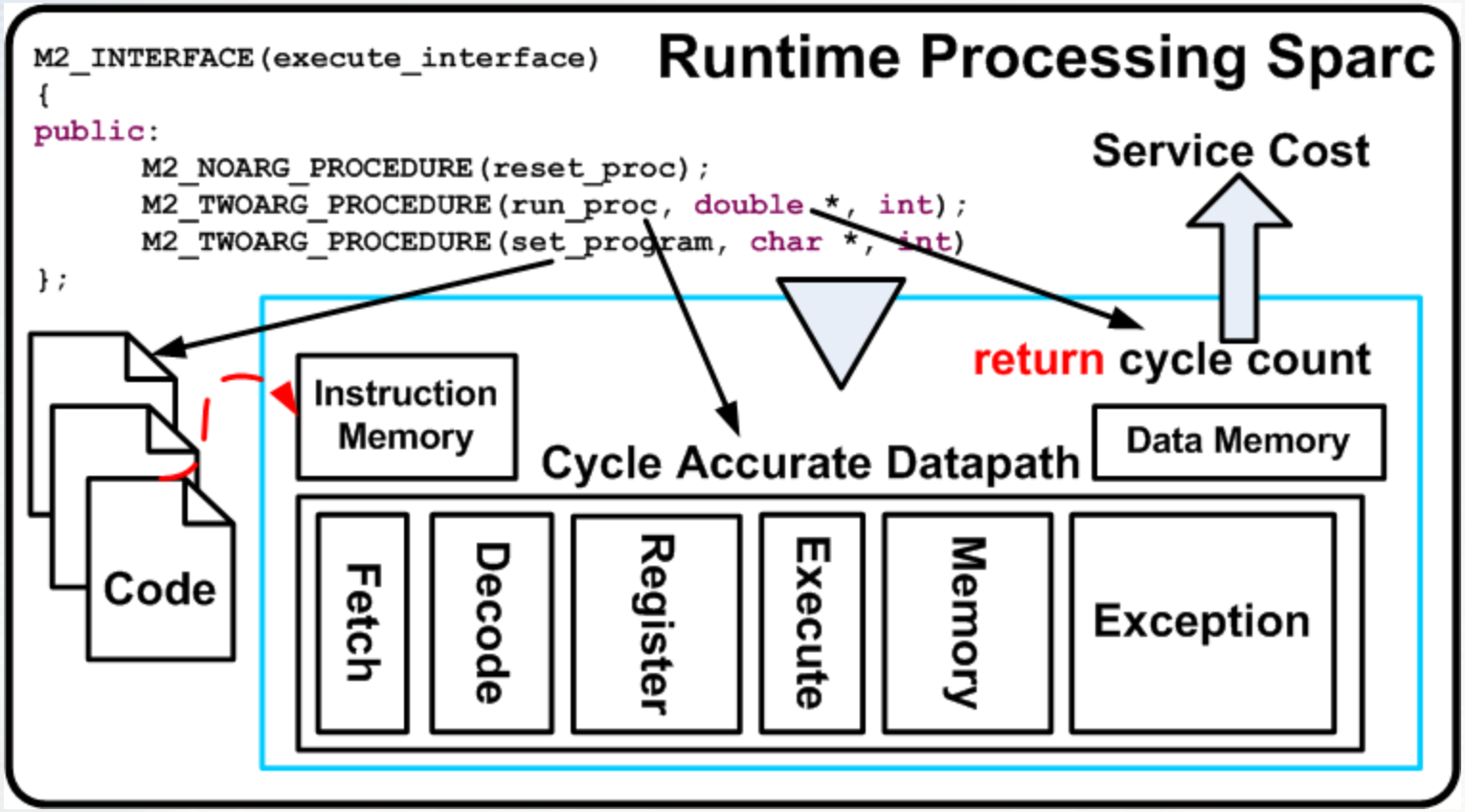
Constraints: Move synchronization from phase 1 to phase 3 completely.

Metro II: Service Modeling

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

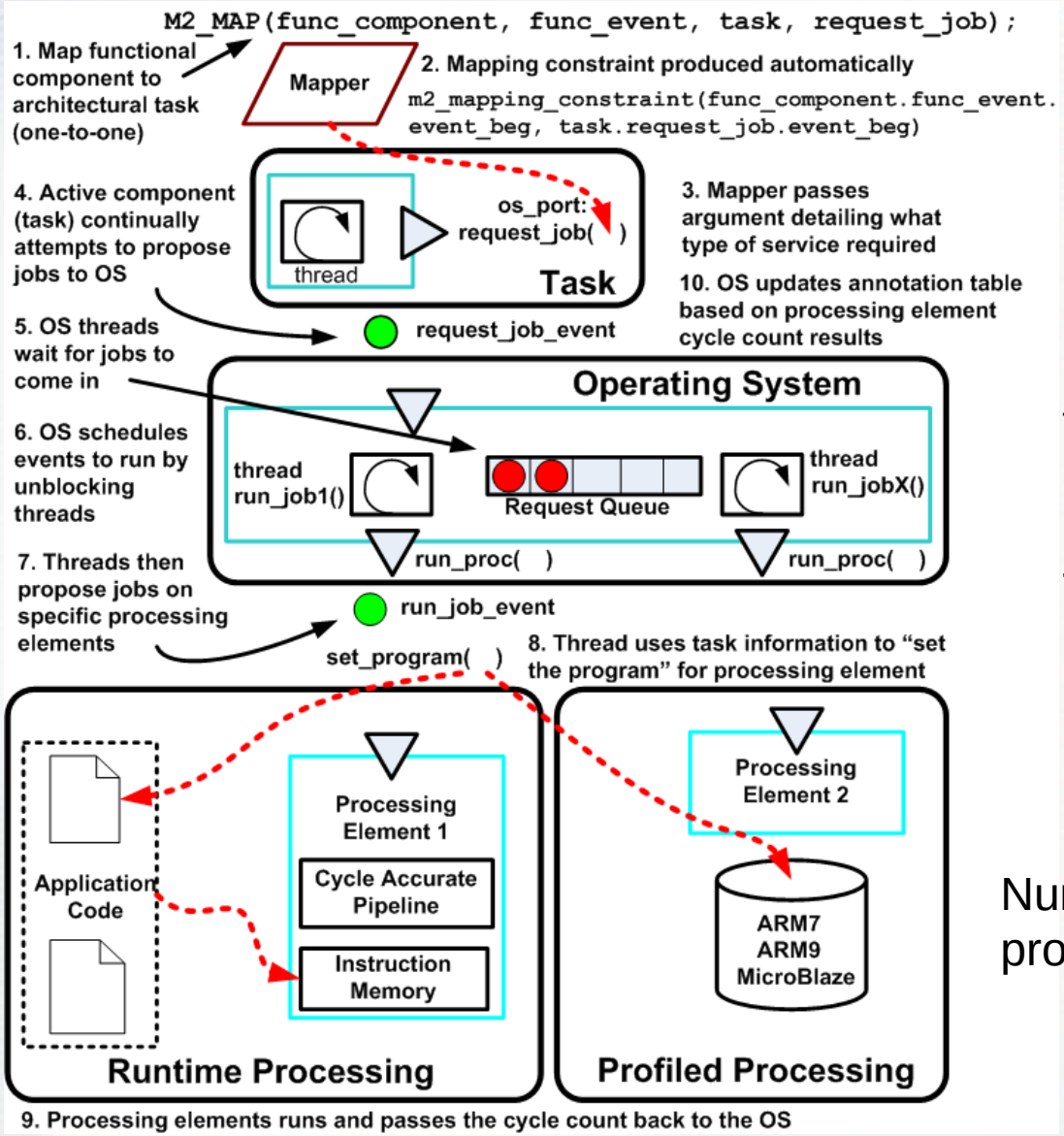


SPARC Runtime Processing Element



A runtime processing based element was created to model the Leon 3 SPARC processor

Architecture Model Overview



Tasks for mapping 1-to-1 with functional components

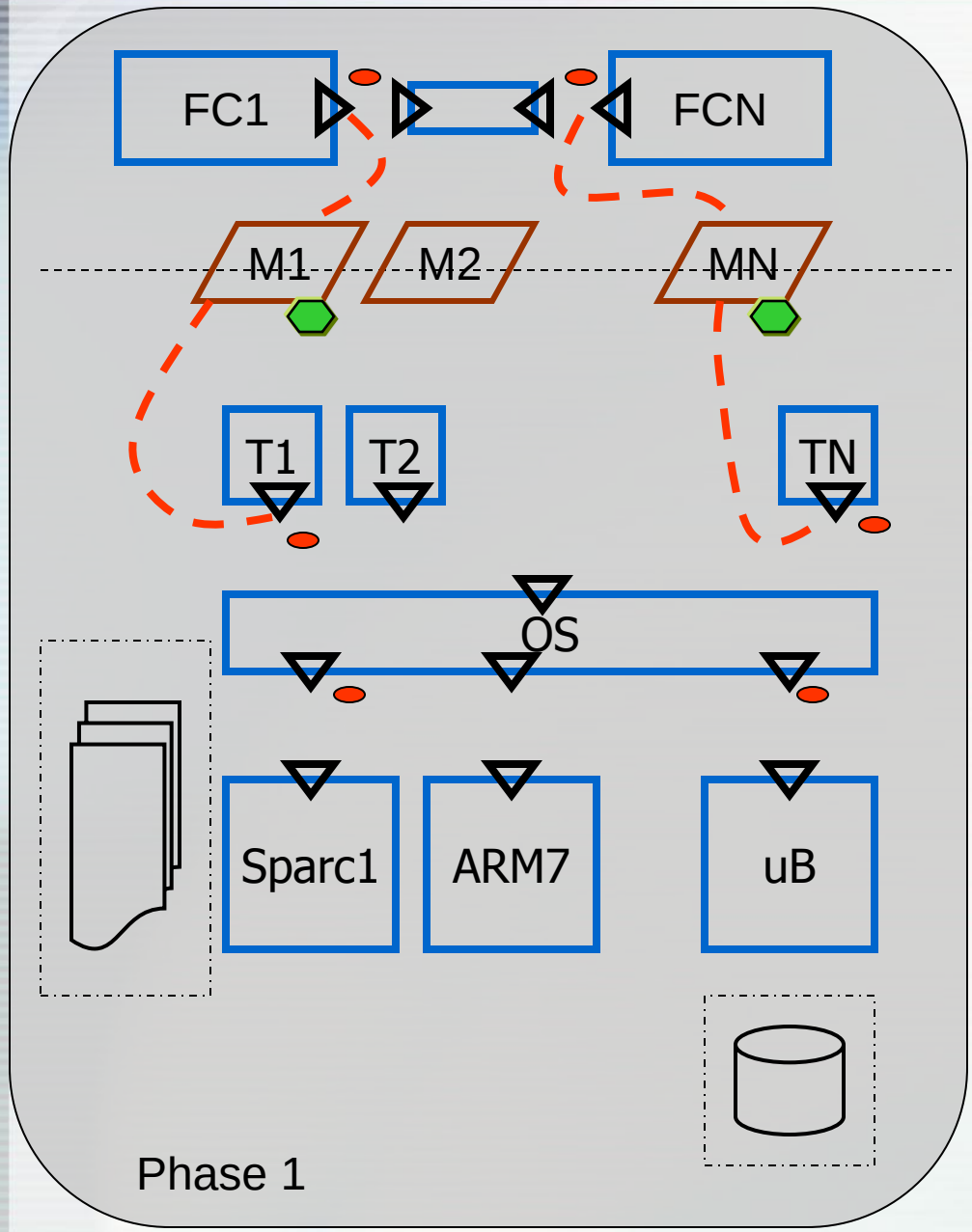
RTOS for scheduling events from N tasks to M processing elements

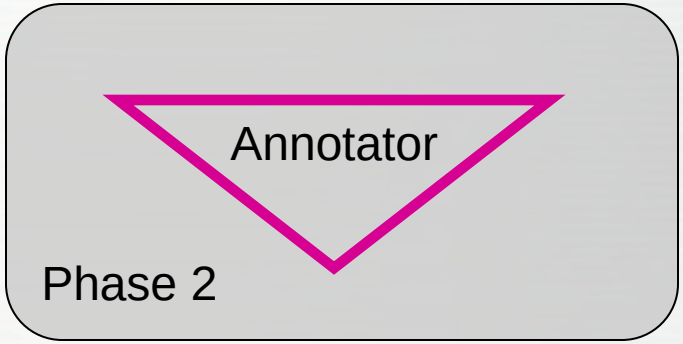
Three scheduling policies:

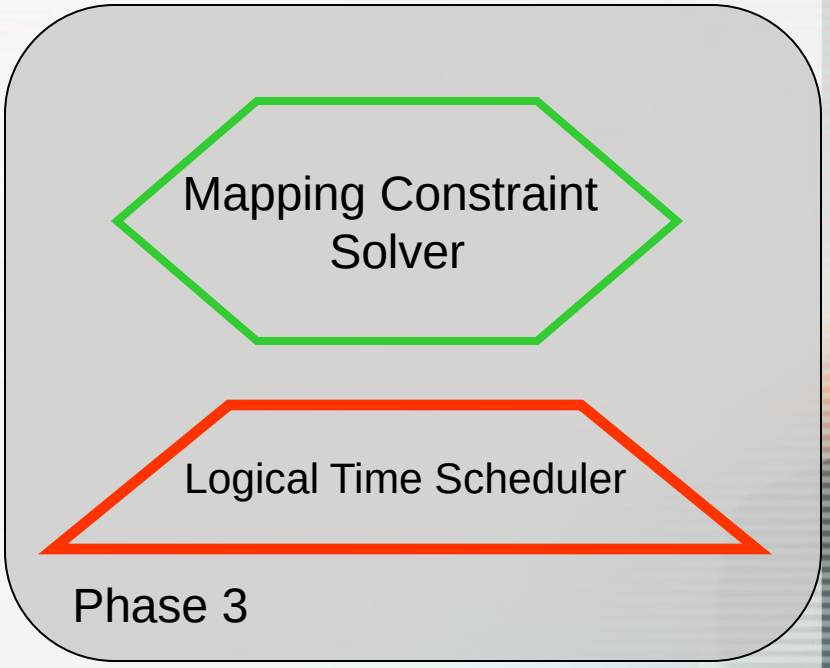
- Round Robin
- Fixed Priority
- FCFS

Numerous configurations of processing elements (48 chosen)

Metro II Complete System





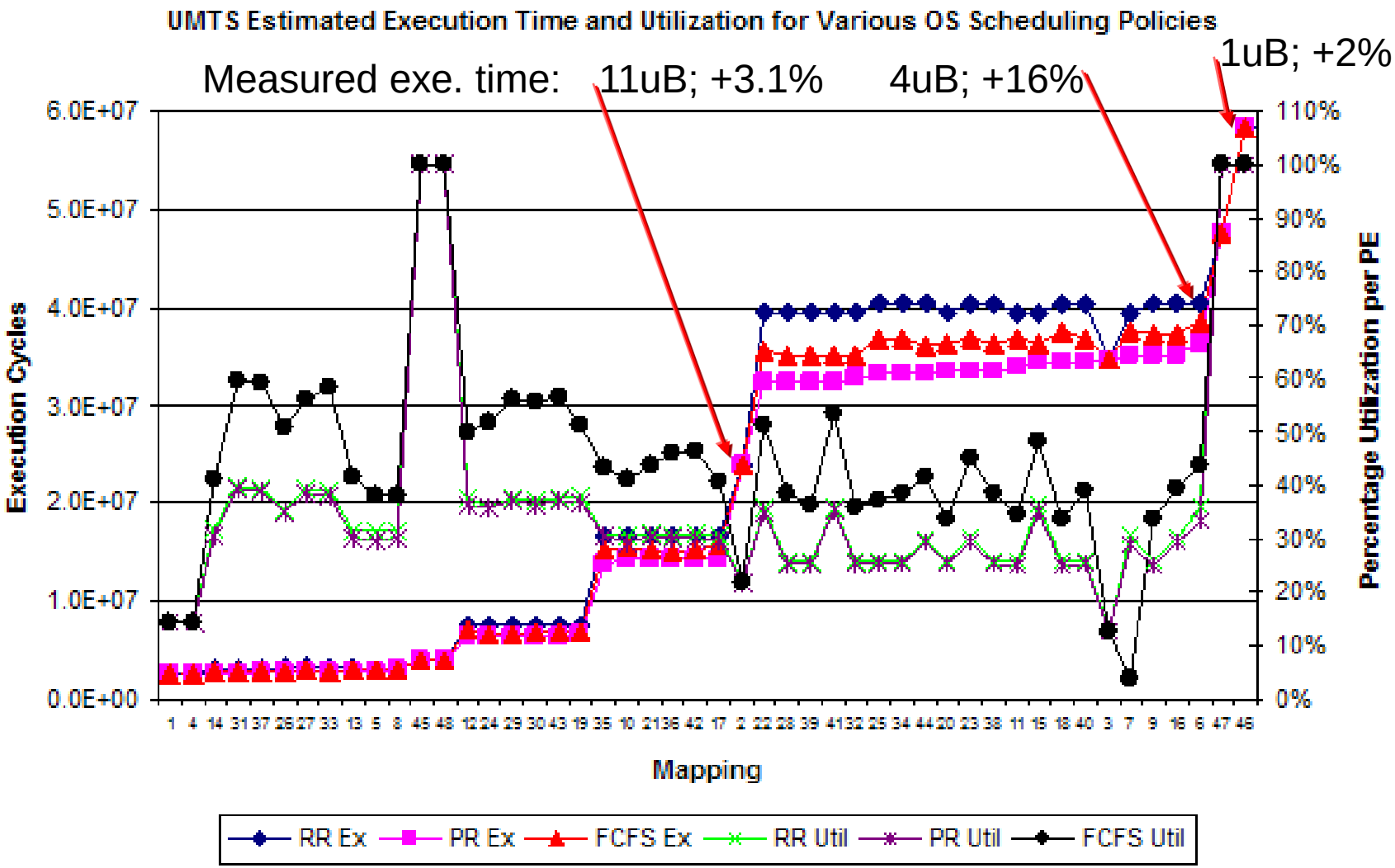


48 Mappings

#	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)

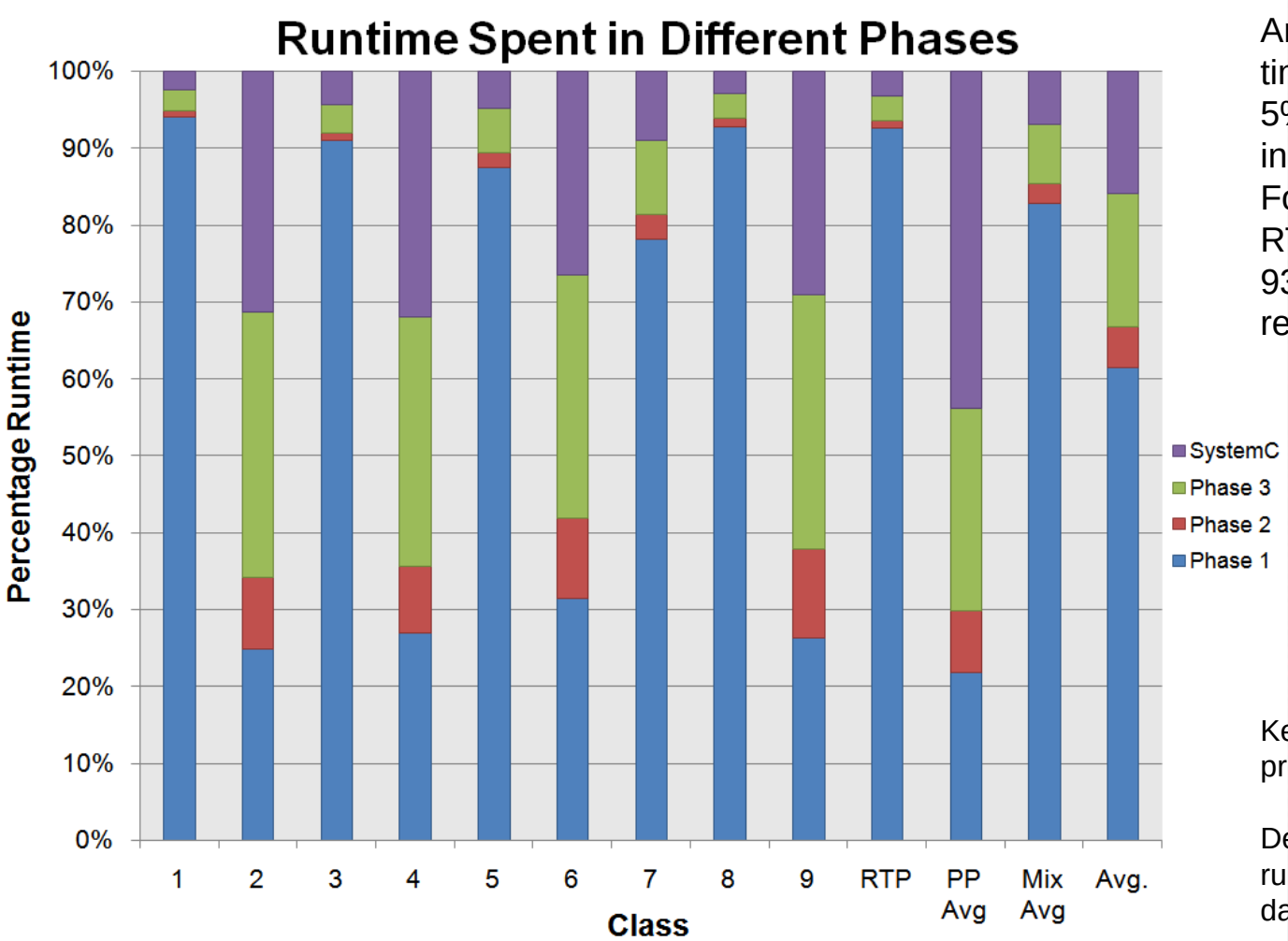
Estimated Execution Time and Utilization



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%**

Runtime analysis across phases and mapping classes



An average 61% of the time is spent in Phase 1, 5% in Phase 2 and 17% in Phase 3 (third section). For most models using RTP the averages are 93%, 0.9%, and 3% respectively.

For pure profiled (PP) mappings they are 21%, 7% and 26%.

For mixed classes the numbers are 82%, 2.6% and 7.6%.

Key message: runtime processing elements dominate.

Despite all of this, the average runtime to process 7000 bytes of data was 54 seconds.

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)