Metropolis Metamodel

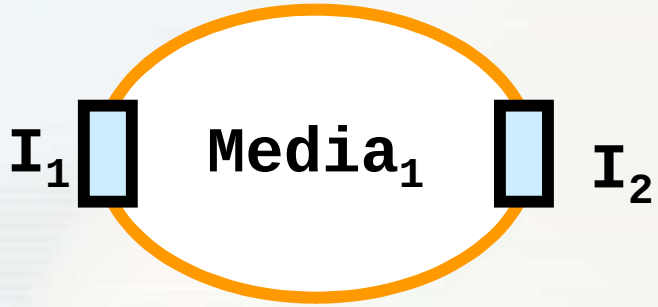
Metropolis Objects

- 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)**

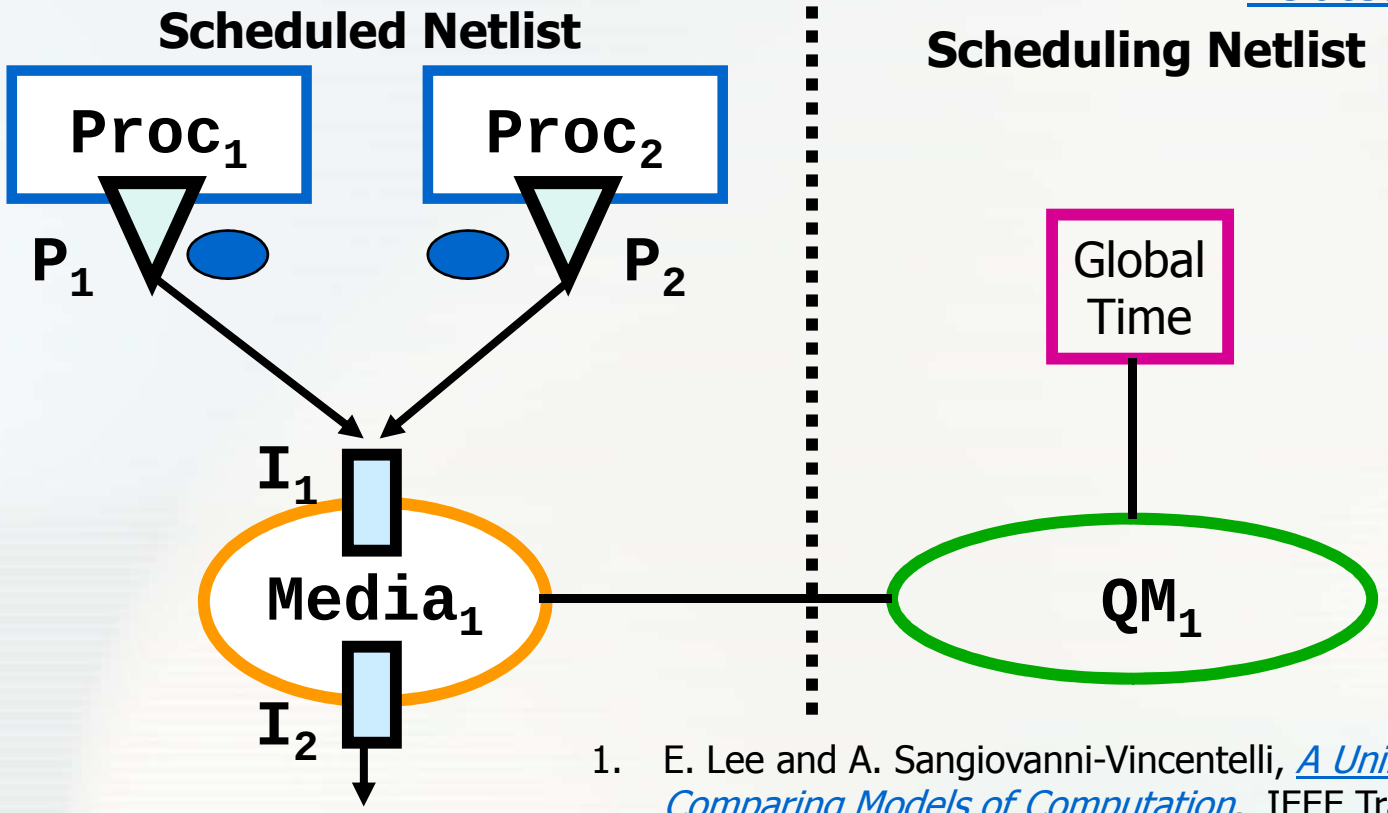
Schedule access to resources and quantities

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.

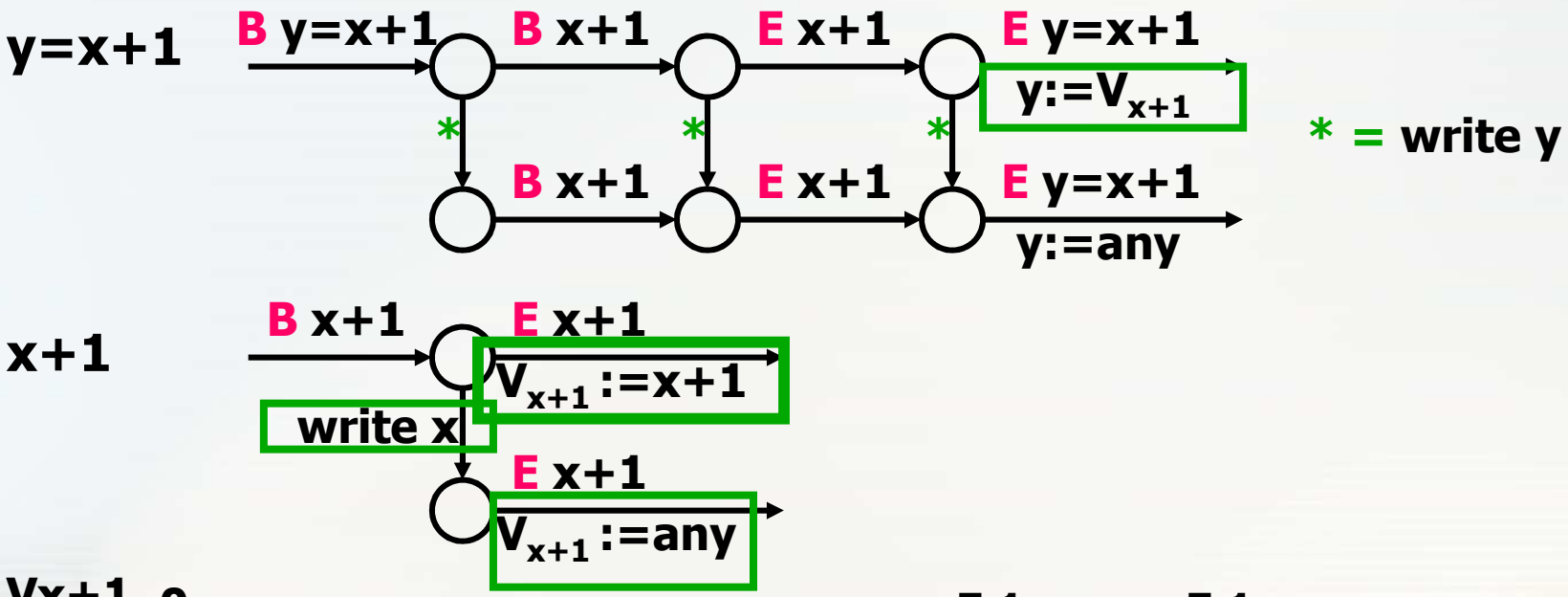
Execution semantics

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:

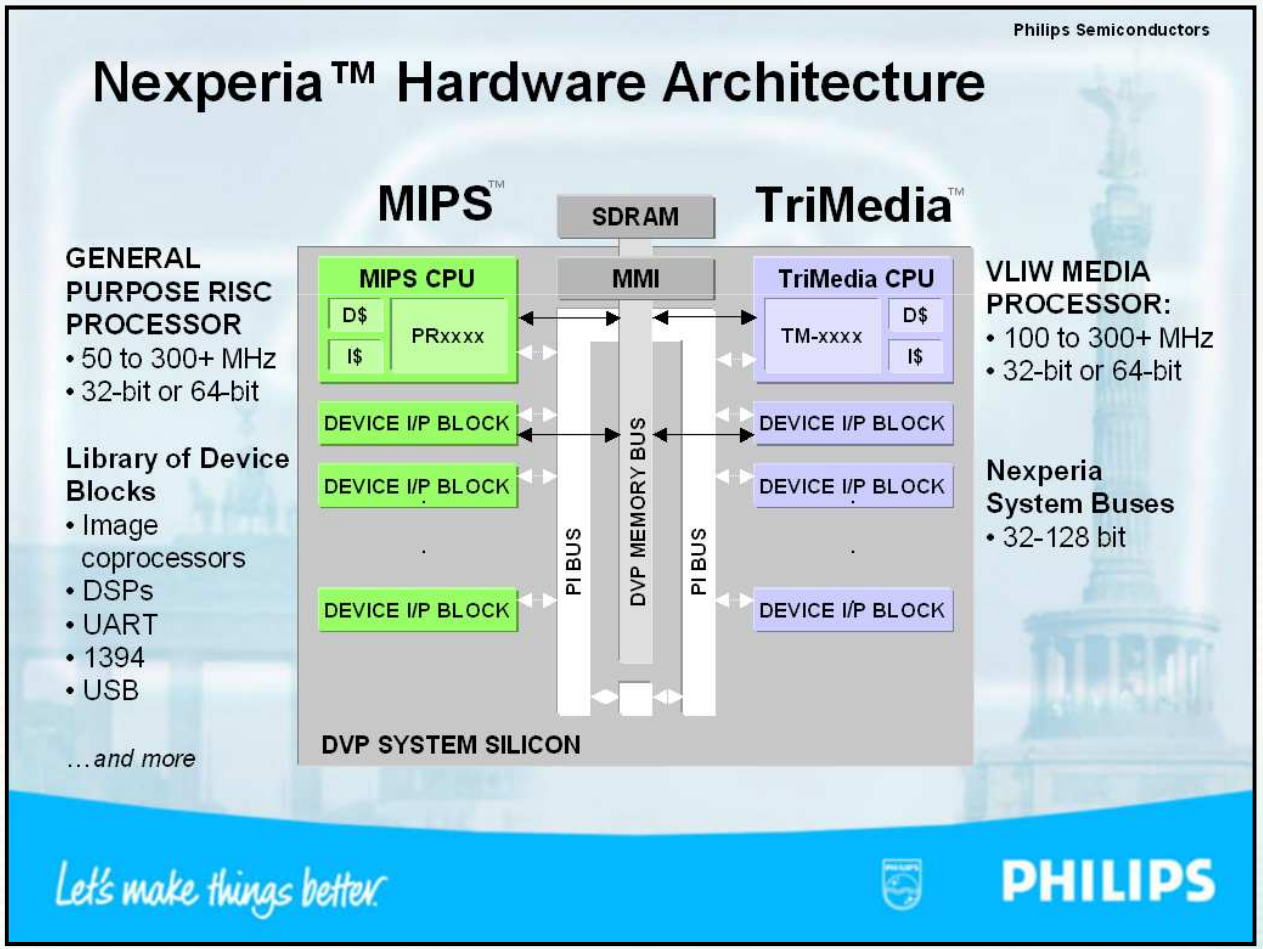
    concurrency, synchronization (awaits and constraints)

Metropolis Architecture Representation

Architecture components

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

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



Meta-model: 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);
   }

  ...
```

Meta-model: quantities

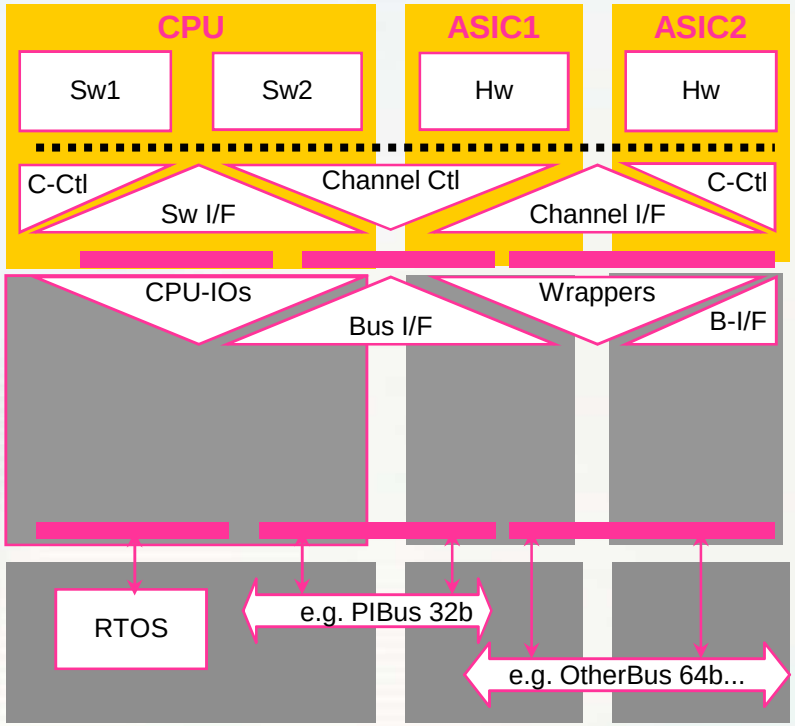
- The domain D of the quantity, e.g. *real* for the global time,
- The operations and relations on D, e.g. subtraction, <, =,
- The function from an event instance to an element of D,
- Axioms on the quantity, e.g.

  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**

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

Programmable Arch. Modeling

- **Computation Services**

PPC405 | MicroBlaze | SynthMaster | SynthSlave

Computation Services
Read (addr, offset, cnt, size), Write(addr, offset, cnt, size),
Execute (operation, complexity)

- **Communication Services**

Processor Local Bus (PLB) | On-Chip Peripheral Bus (OPB) | BRAM

Communication Services
addrTransfer(target, master)
addrReq(base, offset, transType, device)
addrAck(device)

dataTransfer(device, readSeq, writeSeq)
dataAck(device)

- **Other Services**

OPB/PLB Bridge | Mapping Process

Task Before Mapping
Read (addr, offset, cnt, size)

Programmable Arch. Modeling

- **Coordination Services**

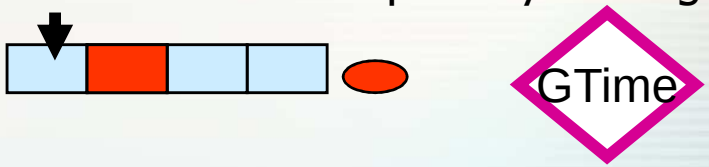
Request (event e)

-Adds event to pending queue of requested events

PostCond()
Resolve()

-Augment event with information (annotation). This is typically the interaction with the quality manager

-Uses algorithm to select an event from the pending queue



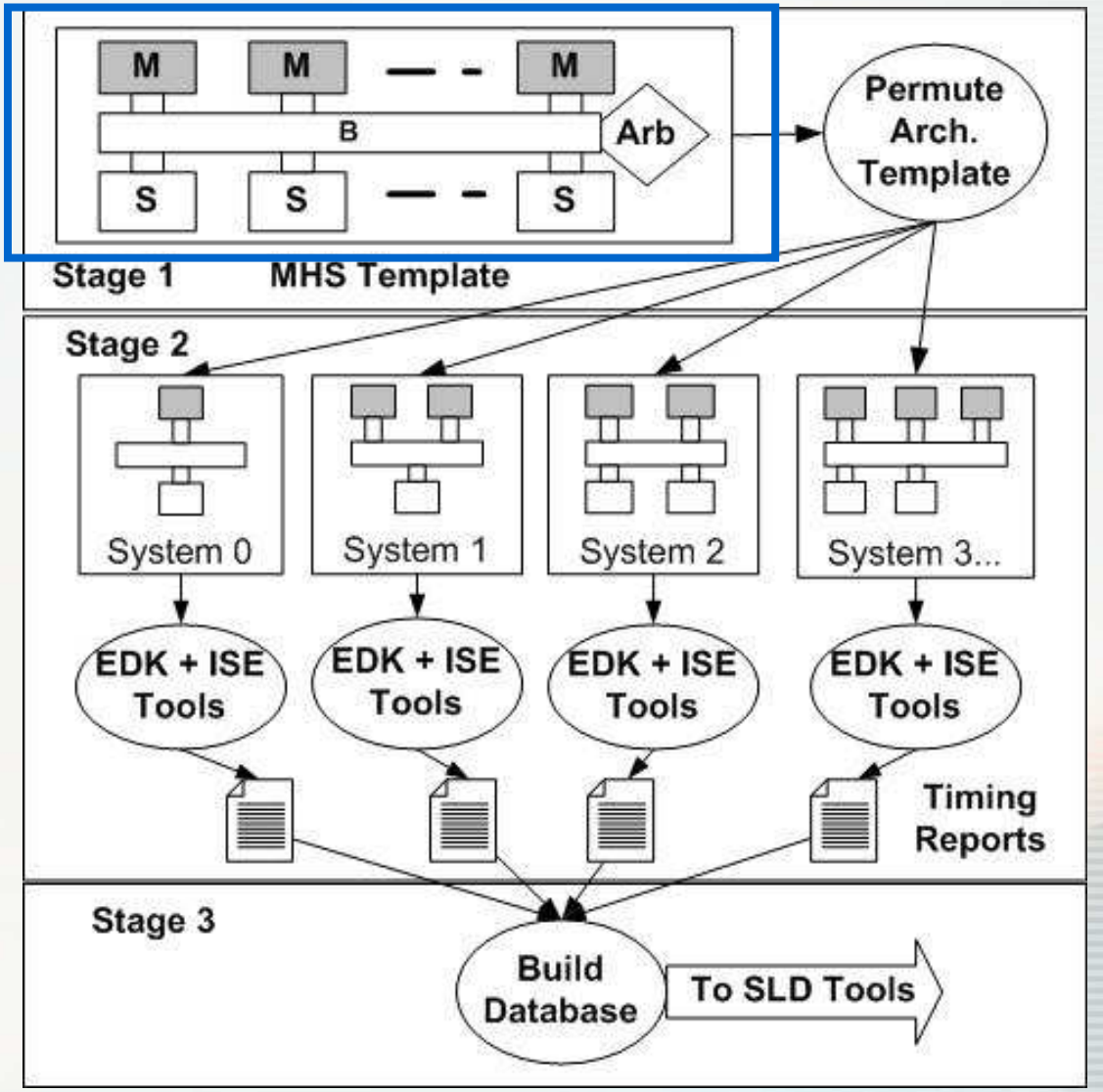
Prog. Platform Characterization

Need to tie the model to actual implementation data!

1. Create template system description.

2. Generate many permutations of the architecture using this template and run them through programmable platform tool flow.

3. Extract the desired performance information from the tool reports for database population.



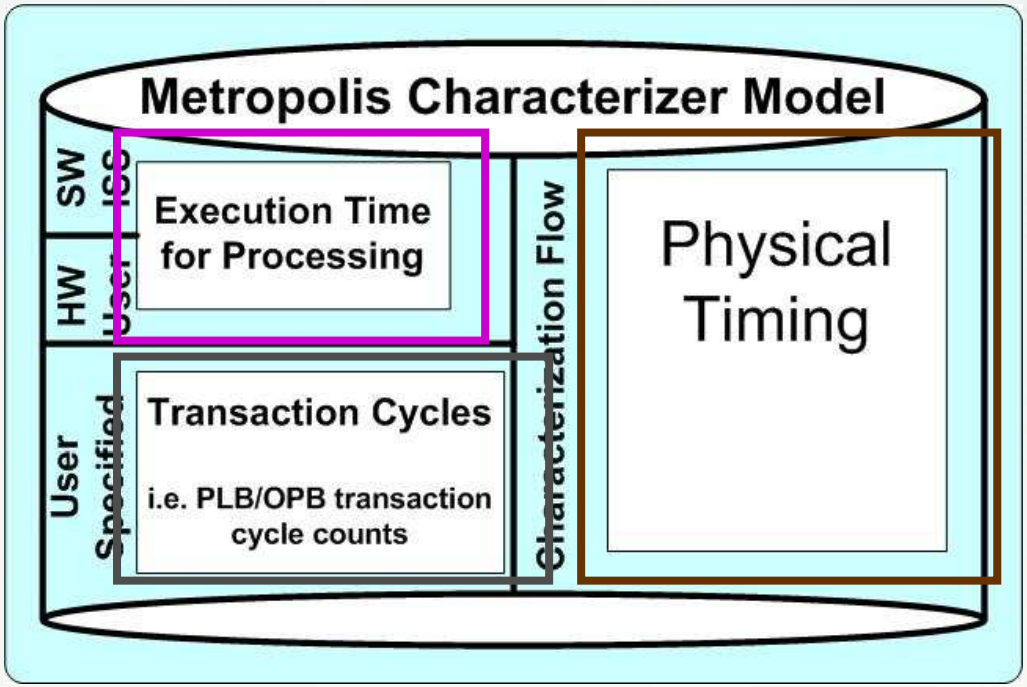
Prog. Platform Characterization

Create database **ONCE** prior to simulation and populate with independent (modular) information.

1. Data detailing performance based on physical implementation.

2. Data detailing the composition of communication transactions.

3. Data detailing the processing elements computation.



From Char Flow Shown

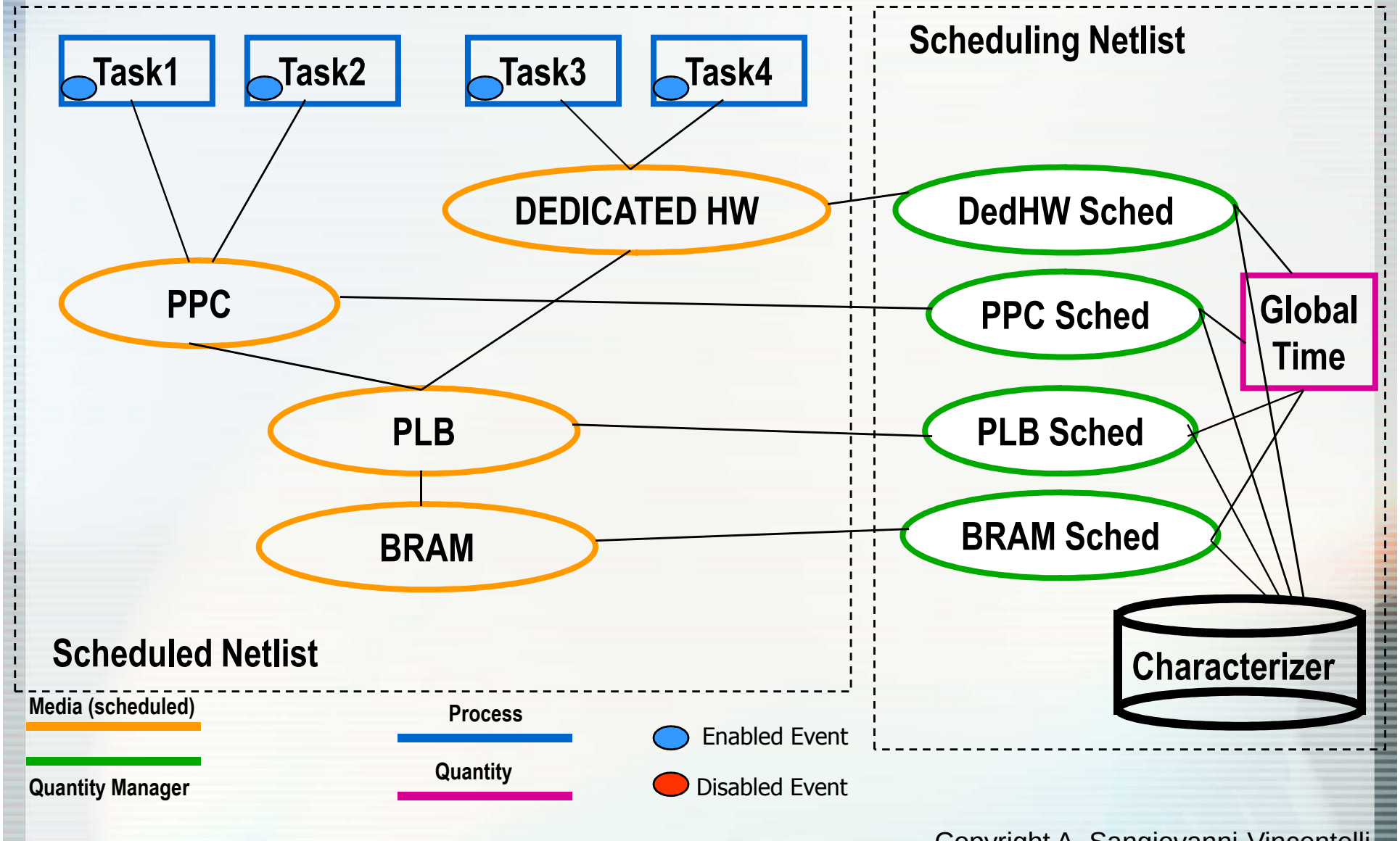
From Metro Model Design

From ISS for PPC

Work with Xilinx Research Labs

1. Douglas Densmore, Adam Donlin, A.Sangiovanni-Vincentelli, *FPGA Architecture Characterization in System Level Design*, Submitted to CODES 2005.
2. Adam Donlin and Douglas Densmore, *Method and Apparatus for Precharacterizing Systems for Use in System Level Design of Integrated Circuits*, Patent Pending.

Modeling & Char. Review



Arch. Refinement Verification

- Architectures often involve hierarchy and multiple abstraction levels.
- These techniques are limited if it is not possible to check if elements in hierarchy or less abstract components are implementations of their counterparts.
- Asks "Can I substitute M1 for M2?"

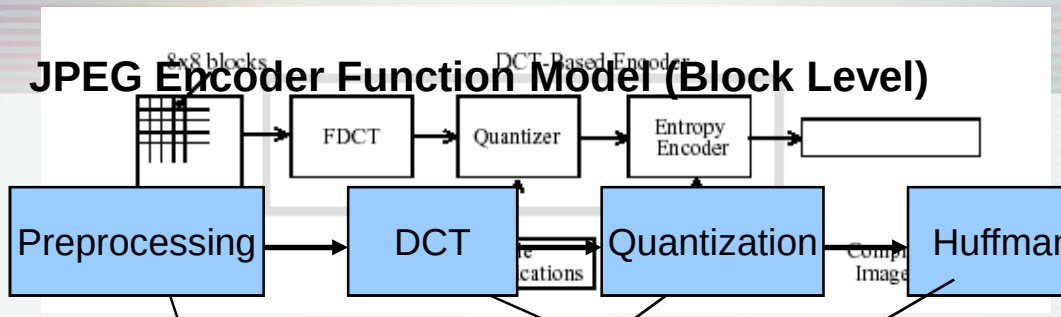
1. **Representing the internal structure of a component.**
2. **Recasting an architectural description in a new style.**
3. **Applying tools developed for one style to another style.**

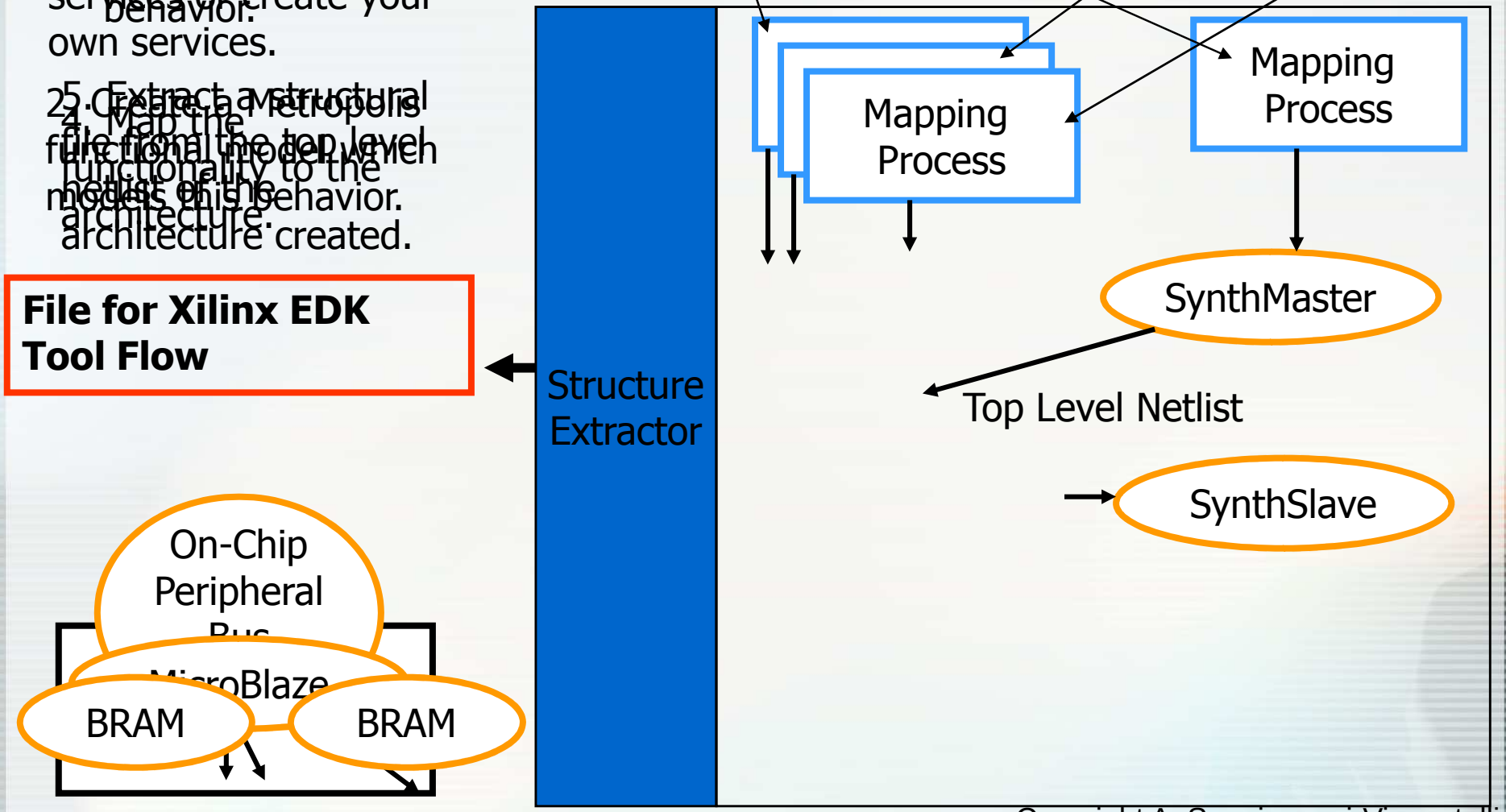
D. Garlan, *Style-Based Refinement for Software Architectures*, SIGSOFT 96, San Francisco, CA, pg. 72-75.

Refinement Technique	Description	Metropolis
Style/Pattern Based	Define template components. Prove they have a desired relationship once. Build arch. from them.	Potential; TTL YAPI
Event Based	Properties (behaviors) expressed as event lists. Explicitly look for this event patterns.	**Discussed**
Interface Based	Create structure capturing all behavior of a components interface. Compare two models.	**Discussed**

Example Design

1. Select an application architecture from library and understand its behavior.
2. Create a Metropolis functional model which models this behavior.
3. Assemble an architecture from library services or create your own services.
4. Map the functionality to the architecture created.
5. Extract a structural netlist of the architecture.





Example Design Cont.

File for Xilinx EDK Tool Flow

1. Feed the captured permutations to the Xilinx tools and extract the data structure information for permutation generator.
2. Feed the permutations to the Xilinx tools and extract the data structure information for permutation generator.
3. Provide characterization information for software and hardware services.
4. Provide characterization information for software and hardware services.

Permutation Generator

Metropolis

Permutation 1 | **Permutation 2** | **Permutation N**

Platform Characterization Tool (Xilinx EDK/ISE Tools)

XILINX

ISS Info

Transaction Info

Char Data

Characterizer Database

Software Routines

```
int DCT (data){
Begin
  calculate …
…
} 32Bit Read = Ack, Add
```

Man

Automatic

Manual
Hardware Routines
DCT1 = 10 Cycles
DCT2 =5 Cycles
FFT = 5 Cycles

Example Design Cont.

Backend Tool Process:

1. Abstract Syntax Tree (AST) retrieves structure.

2. Control Data Flow Graph - **Depth**

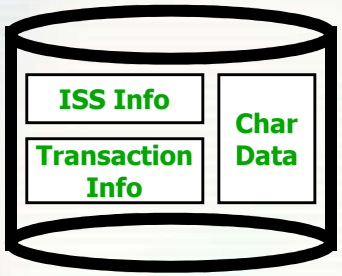
FORTE – Intel Tool

Reactive Models – UC Berkeley

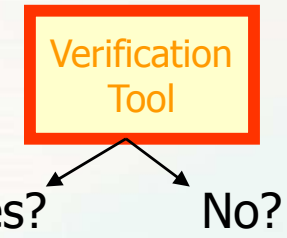
3. Event Traces – **Refinement Properties.**

Vertical Refinement

Horizontal Refinement



Vertical Refinement



1. Simulate the design and observe the performance.

Execution time 100ms
Bus Cycles 4000
Ave Memory Occupancy 500KB

2. Refine design to meet performance requirements.

3. Use Refinement Verification to check validity of design changes.

- **Depth, Vertical, or Horizontal**
- **Refinement properties**

4. Re-simulate to see if your goals are met.

Execution time 200ms
Bus Cycles 1000
Ave Memory Occupancy 100KB