Heterogeneous Systems Modeling and Design

Alberto Sangiovanni-Vincentelli

The Edgar L. and Harold H. Buttner Chair, Department of EECS, University of California at Berkeley

Co-founder, CTA and Member of the Board, Cadence

Scientific Director, PARADES

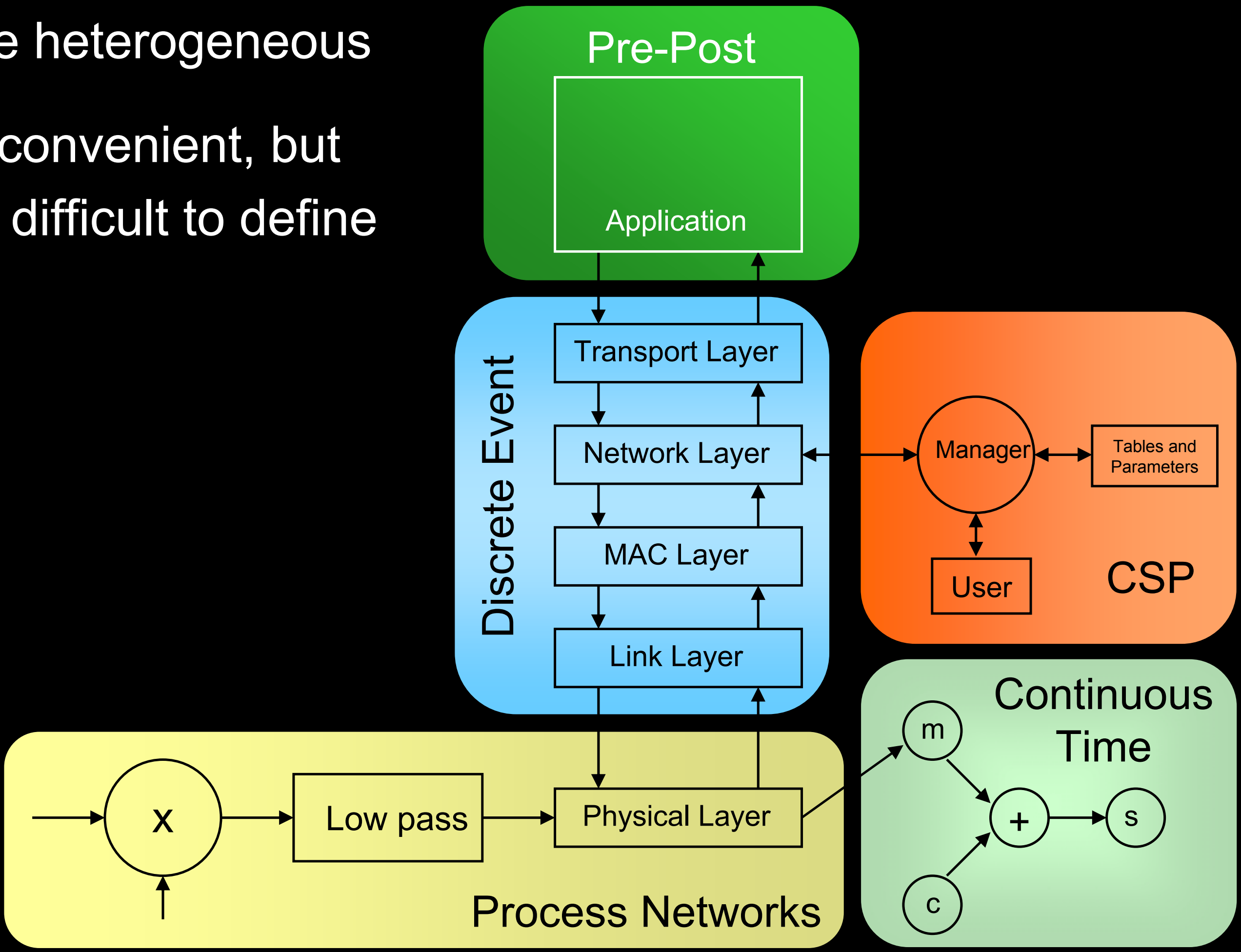
Artist, DATE, March. 2005

Outline

- Heterogeneity
- Design Environments
- Abstract Semantics
  - Tagged Signal Model
  - Ptolemy II
  - Metropolis

Motivations

- Systems are heterogeneous
- Separation convenient, but interactions difficult to define



The Design Nightmare

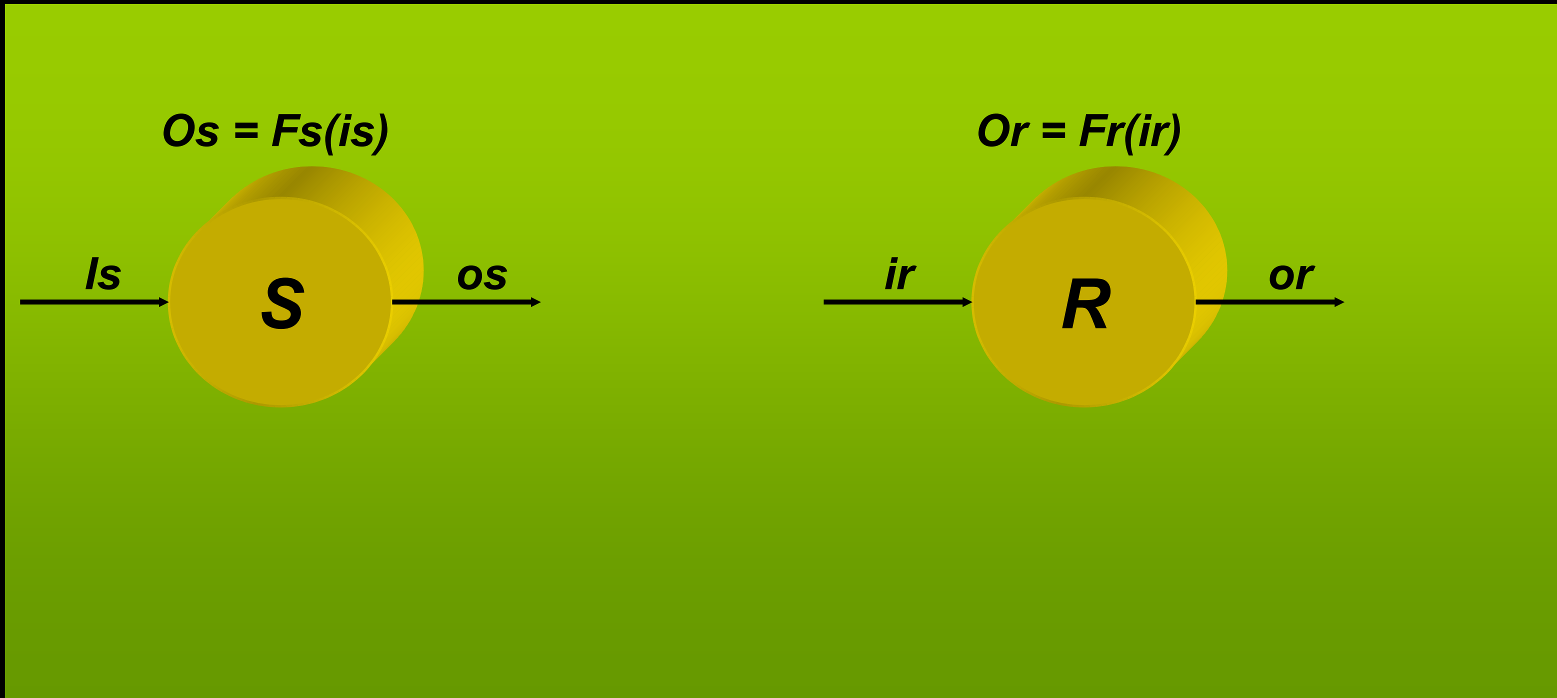
Specification:

The Design Nightmare

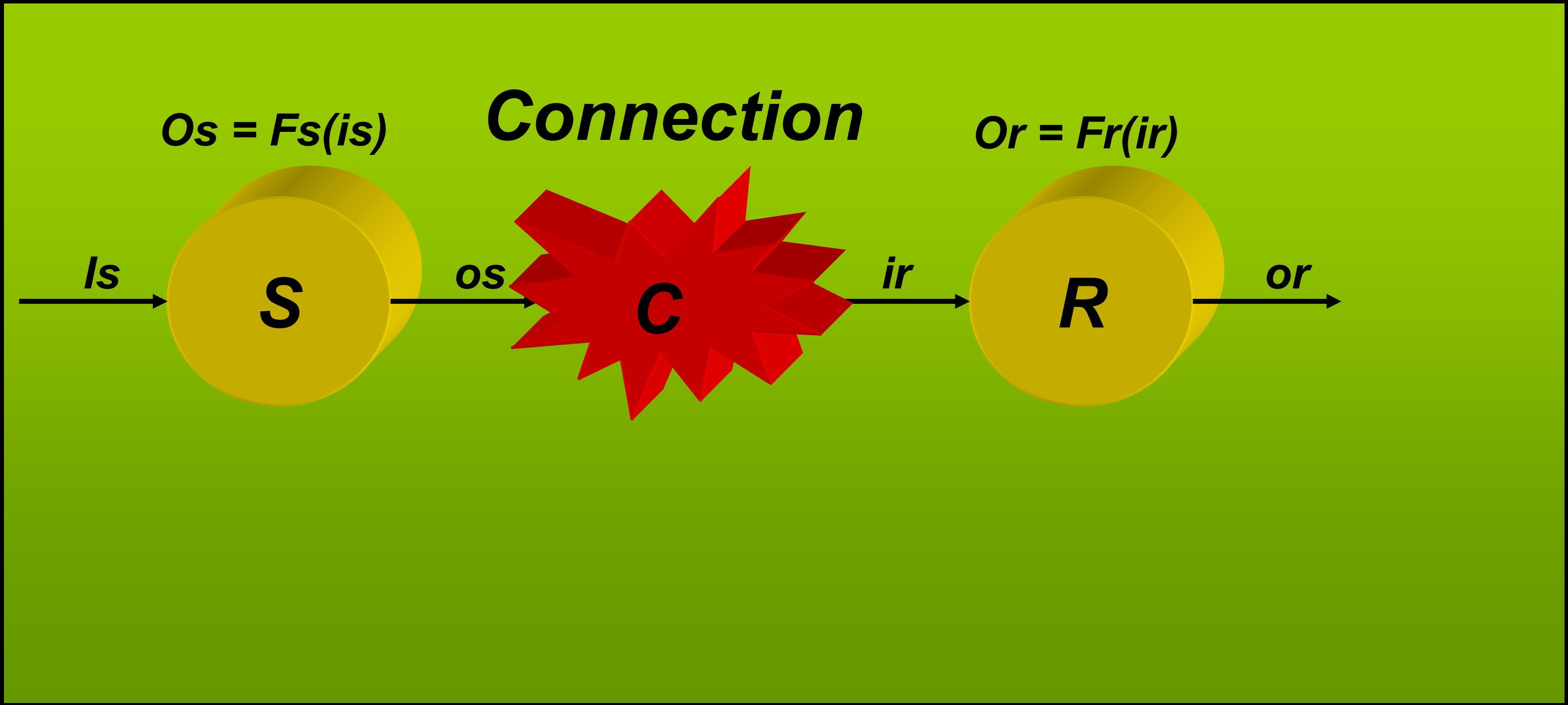
Implementation:

P. Picasso “Femme se coiffant” 1940

What is Communication?

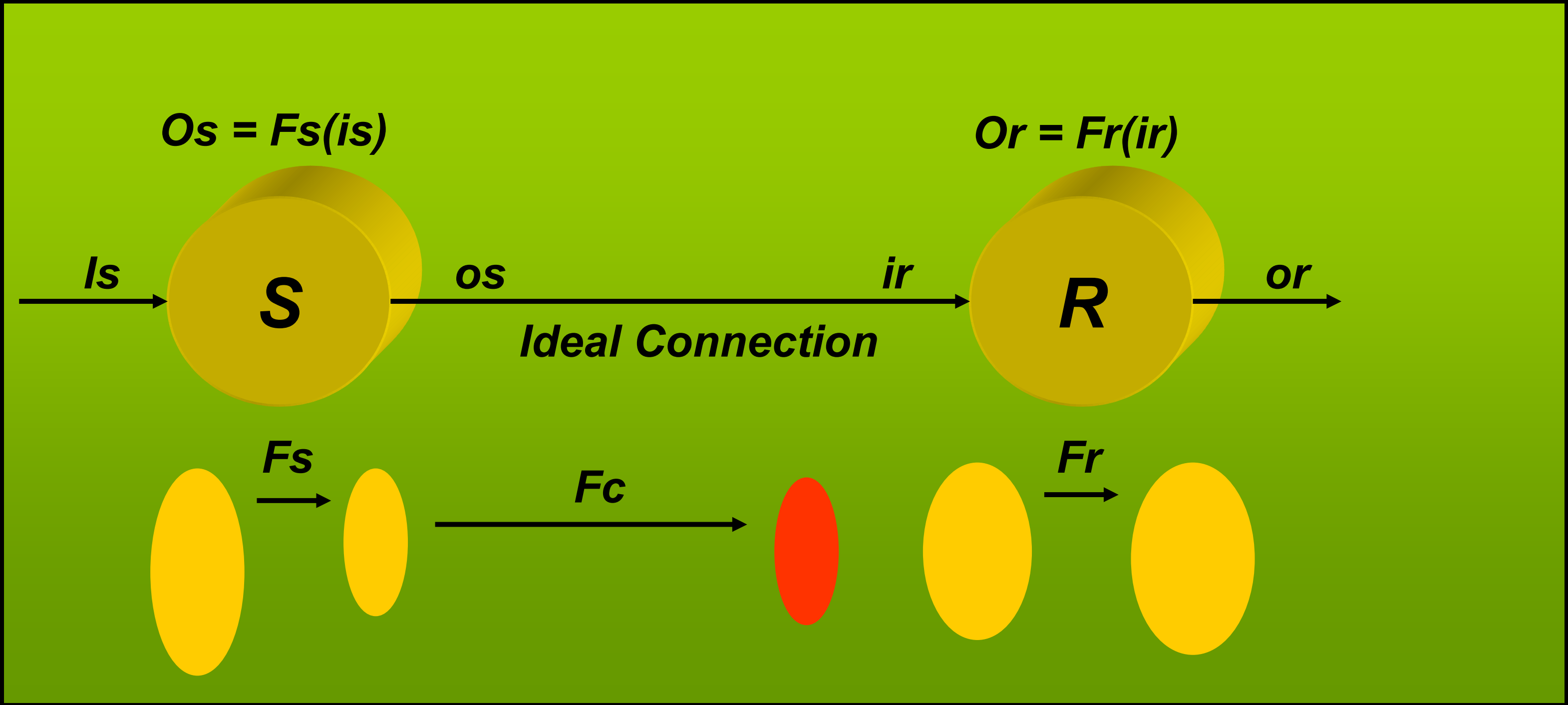


What is Communication?



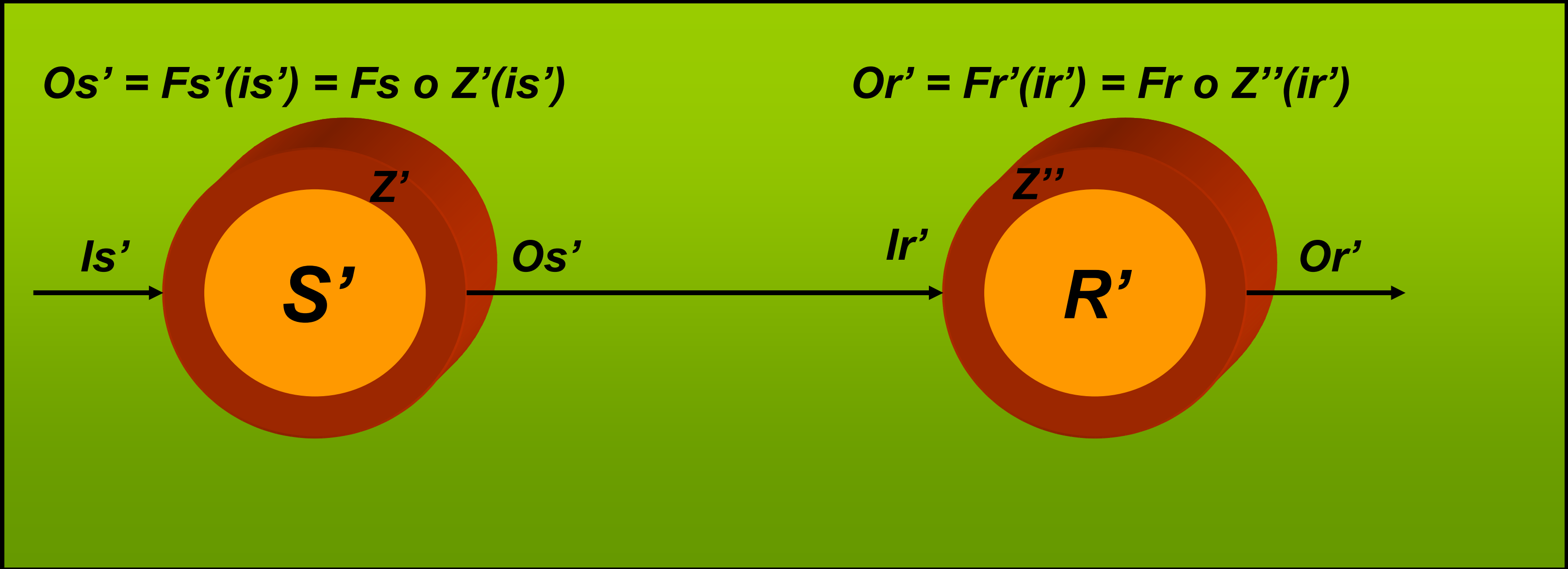
- Connection C enables the interaction between the behaviors S and R

Behavior Adaptation



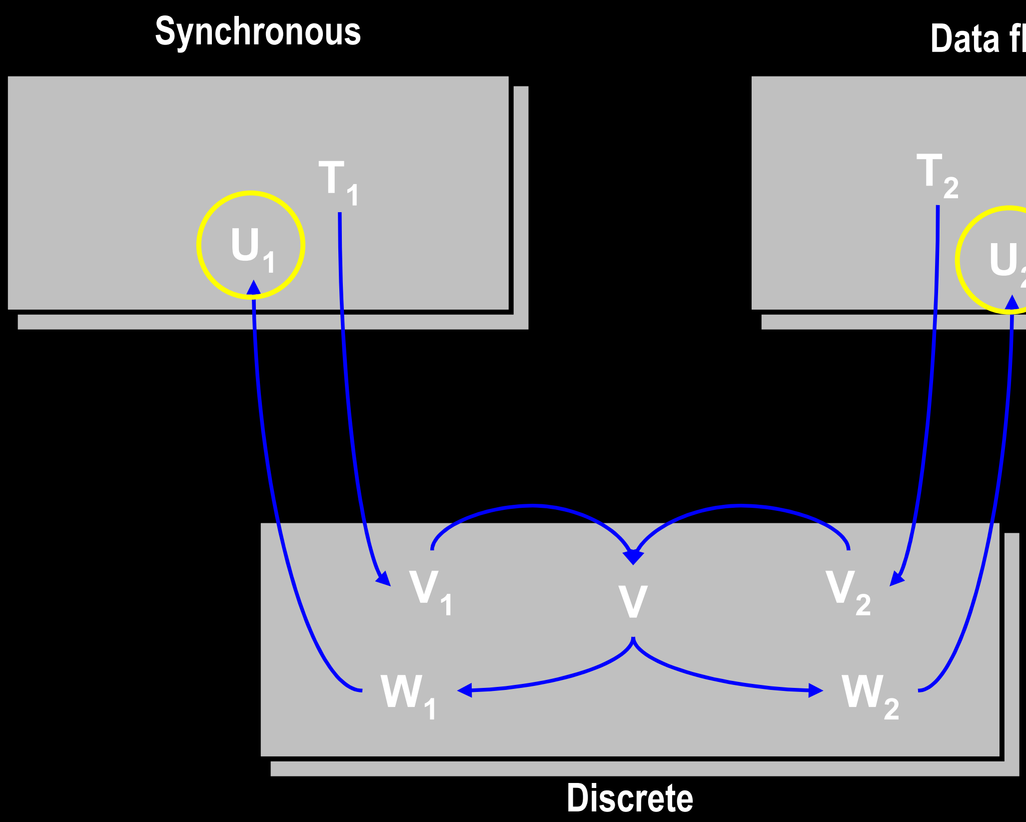
- R not defined for some (or all) outputs of S: behavior mismatch

Behavior Adaptation



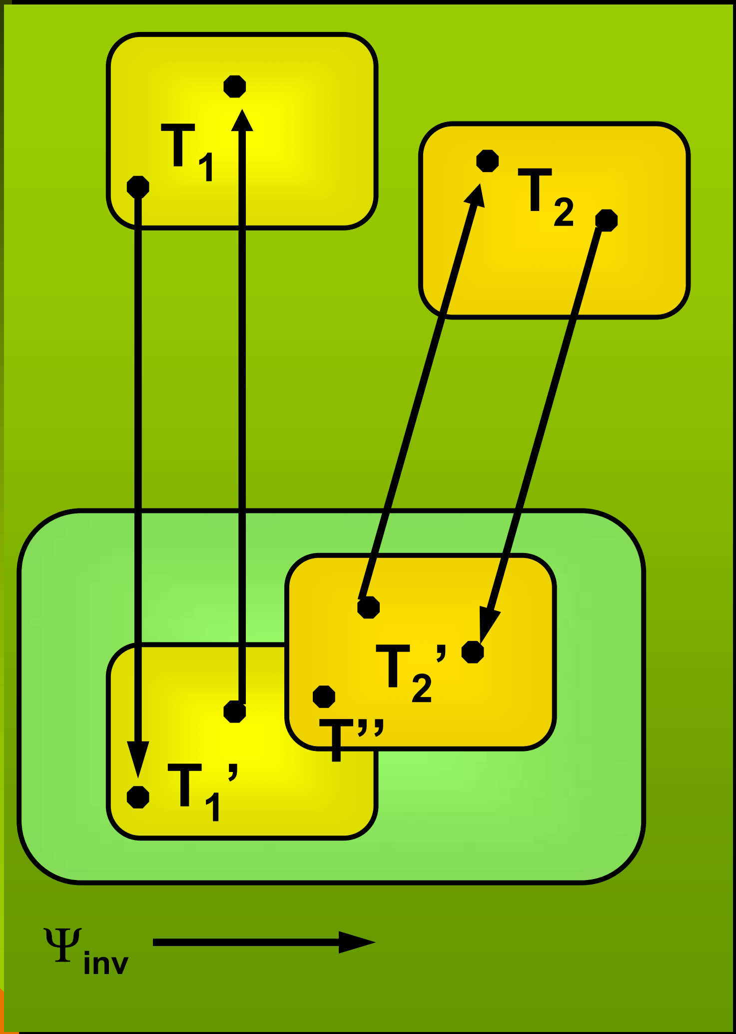
- Behavior Adapter encapsulates S and R
- S' and R' communicate successfully over an ideal connection
- Key Question: How do we build the adapters?

Interaction Propagation



1. Refinement
2. Composition
3. Projection
4. Abstraction

Combining MoCs: Conservative Approximations



Want to compose T_1 and T_2 from different trace structure algebras

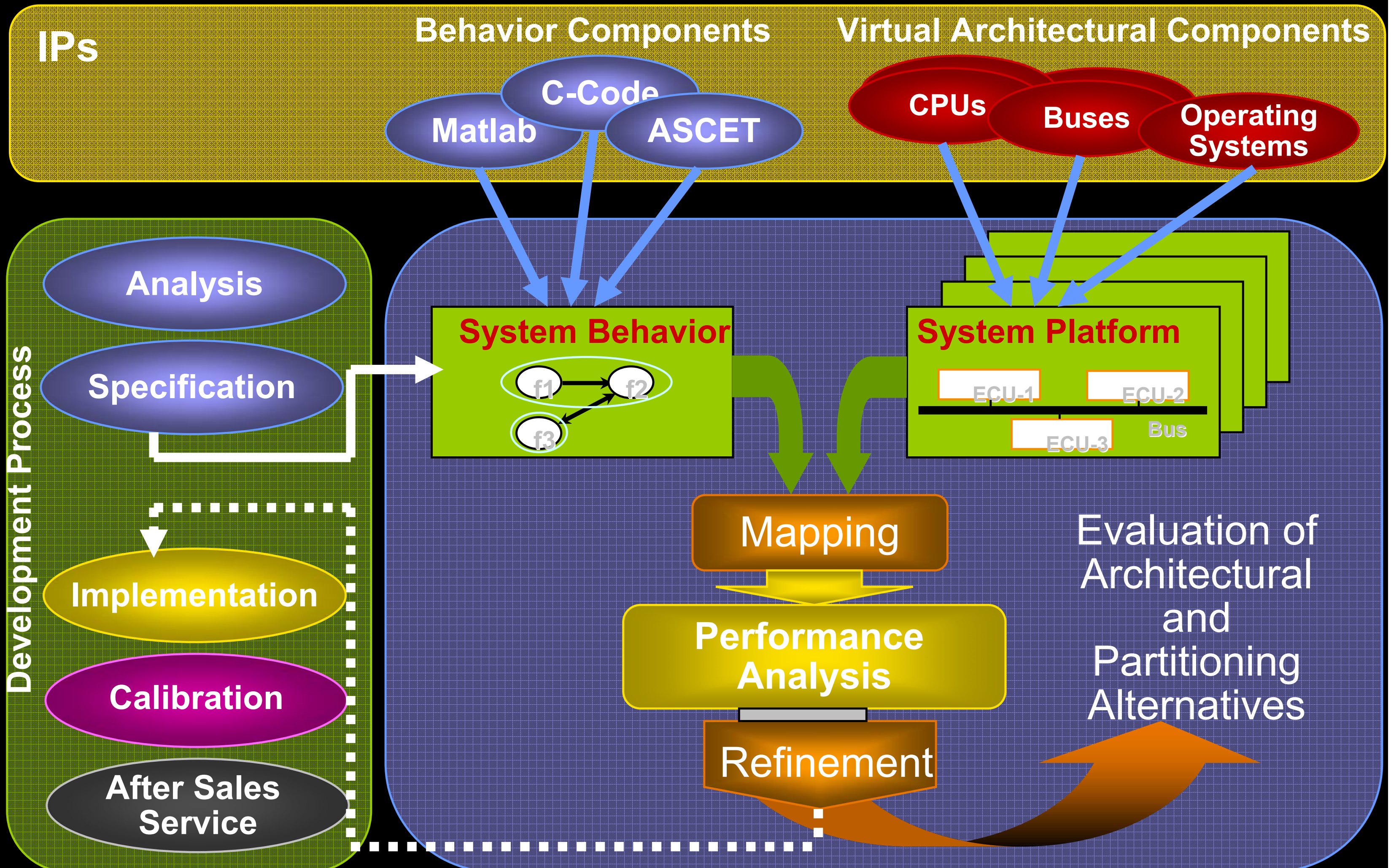
- Compose T_1' and T_2'
- Get T''
- Project back to T_1' and T_2'
- Map back in the abstract domain

- Find restrictions due to the interaction at the higher level (constraint application)
- Find greatest possible T_1 and T_2 that still have the same interaction (don't cares, synthesis)

Key points

- The outlined technique defines the effects of the interaction
- The result depends on
  - The notion of composition at the refined level
  - The particular abstraction and refinement
- We can’t define the interaction uniquely!
- How can we generalize? Need a formal approach to this problem (see Conservative Approximations, R. Passerone et al. and Tagged Systems, A. Benveniste et al.!)

Another Source of Heterogeneity:
Concurrent presence of different levels of abstractions



Outline

- Heterogeneity
- **Design Environments**
- Abstract Semantics

Putting it all together….CHALLENGE!

- We need an integration platform
  - To deal with heterogeneity:
    - Where we can deal with Hardware and Software
    - Where we can mix digital and analog
    - Where we can assemble internal and external IPs
    - Where we can work at different levels of abstraction
  - To handle the design chain
  - To support integration
    - e.g. tool integration
    - e.g. IP integration
- The integration platform must subsume the traditional design flow, rather than displacing it

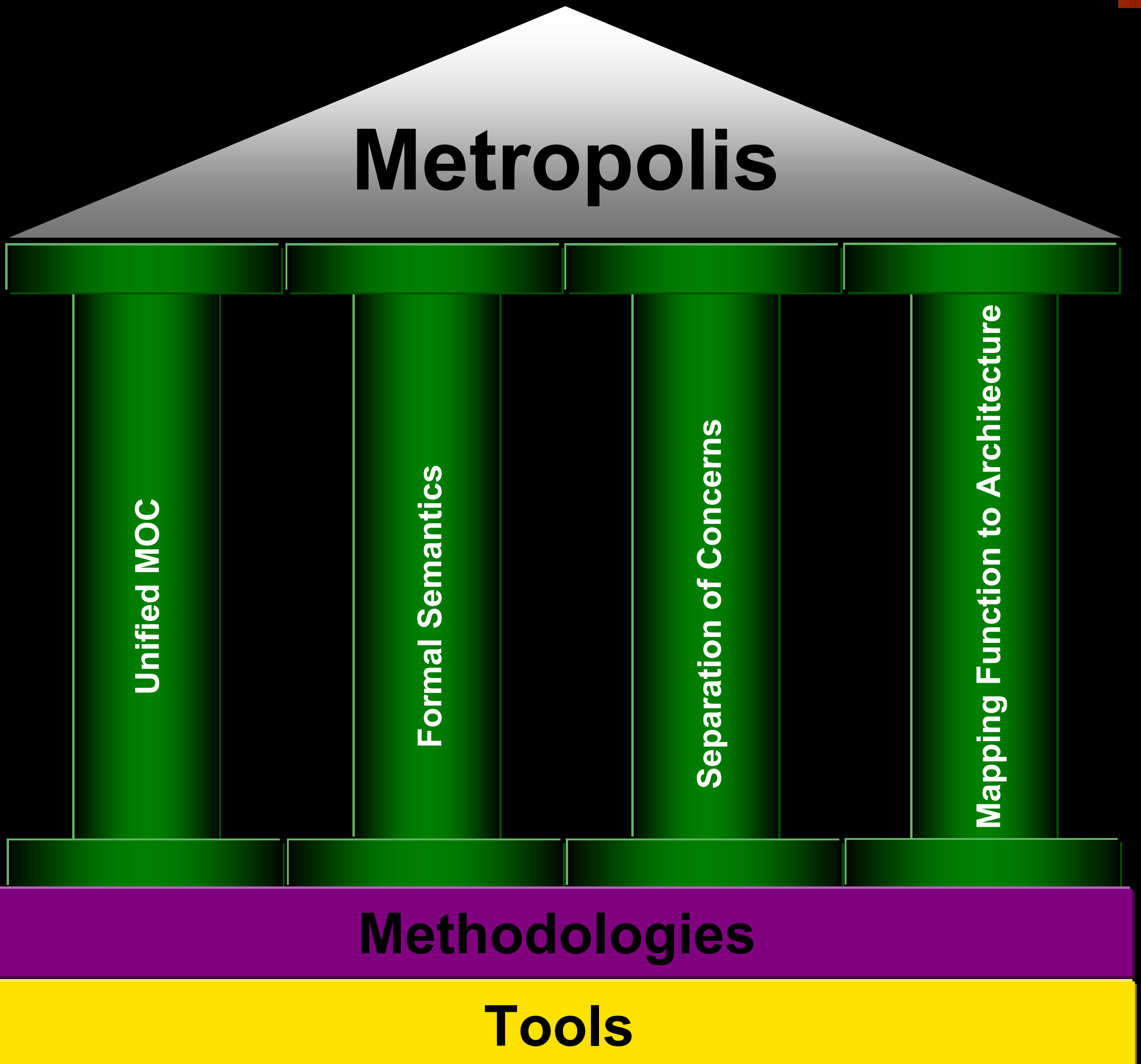
Metropolis: an Environment for System-Level Design

- Motivation
  - Both design complexity and the need for verification are increasing
  - Semantic link between specification and implementation is necessary
- Platform-Based Design
  - Meet-in-the-middle approach
  - Separation of concerns
    - Function vs. architecture
    - Capability vs. performance
    - Computation vs. communication
- Metropolis Framework
  - Extensible framework providing simulation, verification, and synthesis capabilities
  - Easily extract relevant design information and interface to external tools
- Released Sept. 15th, 2004

Fundamental Concepts

- Support for different Models of Computation
- Mix of imperative and declarative specification styles
- Quantities of interest dictated by the designer, not the framework
- Framework designed to allow interfacing with external tools

Metropolis Guiding Principles



Metropolis Framework

Function Specification

Design Constraints & Assertions

Architecture (Platform) Specification

Metropolis Infrastructure

- Design methodology
- Meta model of computation
- Base tools
  - Design imports
  - Meta model compiler
  - Simulation

Synthesis/Refinement

- Compile-time scheduling of concurrency
- Communication-driven hardware synthesis
- Protocol interface generation

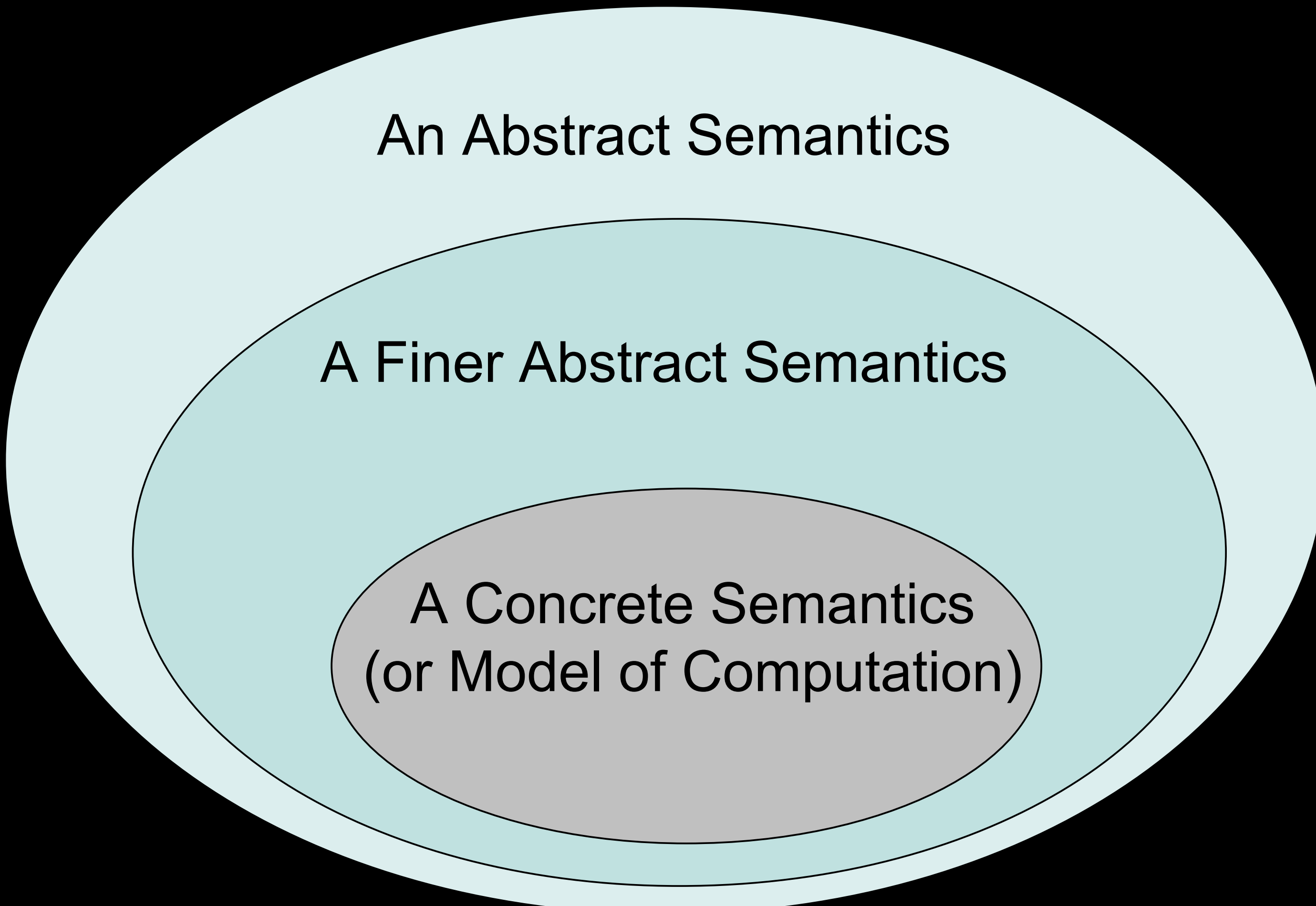
Analysis/Verification

- Static timing analysis of reactive systems
- Invariant analysis of sequential programs
- Refinement verification
- Formal verification of embedded software

Outline

- Heterogeneity
- Design Environments
- **Abstract Semantics**
  - Tagged Signal Model
  - Ptolemy II
  - Metropolis

Where We Are Headed



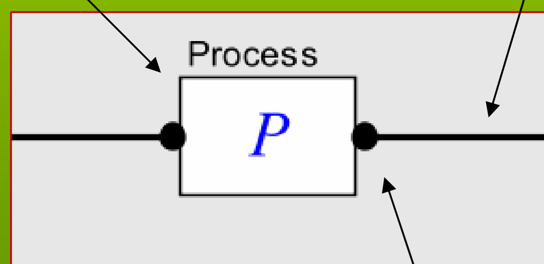
Tagged Signal Abstract Semantics:
Lee-Sangiovanni Vincentelli (LSV) Model

$$P \subset S_1 \times S_2$$

a "process" is a subset of the signals with which it interacts.

signal is a member of a set of signals, where the set depends on the model of computation and resolved data type of the connection.

$s_1 \in S_1$



$s_2 \in S_2$

port may be an input or an output, or neither or both. It is irrelevant.

This outlines a general *abstract semantics* that gets specialized. When it becomes concrete you have a *model of computation.*

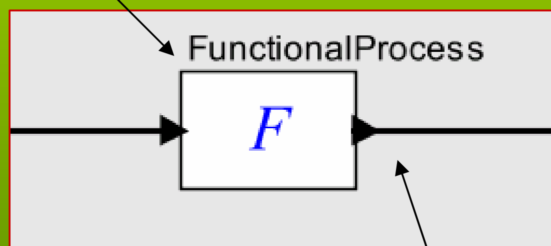
A Finer Abstraction Semantics

Functional Abstract Semantics:

a process is now a function from input signals to output signals.

$$F : S_1 \to S_2$$

$s_1 \in S_1$



$s_2 \in S_2$

port is now either an input or an output (or both).

This outlines an *abstract semantics* for deterministic producer/consumer actors.

Uses for Such an Abstract Semantics

- Give structure to the sets of signals
  - e.g. Use the Cantor metric to get a metric space.
- Give structure to the functional processes
  - e.g. Contraction maps on the Cantor metric space.
- Develop static analysis techniques
  - e.g. Conditions under which a hybrid systems is provably non-Zeno.

Another Finer Abstract Semantics

Process Networks Abstract Semantics:

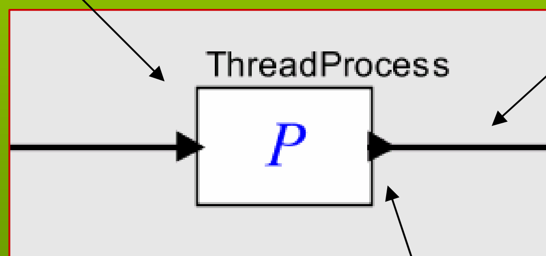
A process is a sequence of operations on its signals where the operations are the associative operation of a *monoid*

sets of signals are *monoids*, which allows us to incrementally construct them. E.g.

- stream
- event sequence
- rendezvous points …

$$P \subset S_1 \times S_2$$

$s_1 \in S_1$



$s_2 \in S_2$

process is not necessarily functional (can be nondeterministic).

port is either an input or an output or both.

This outlines an abstract semantics for actors constructed as processes that incrementally read and write port data.

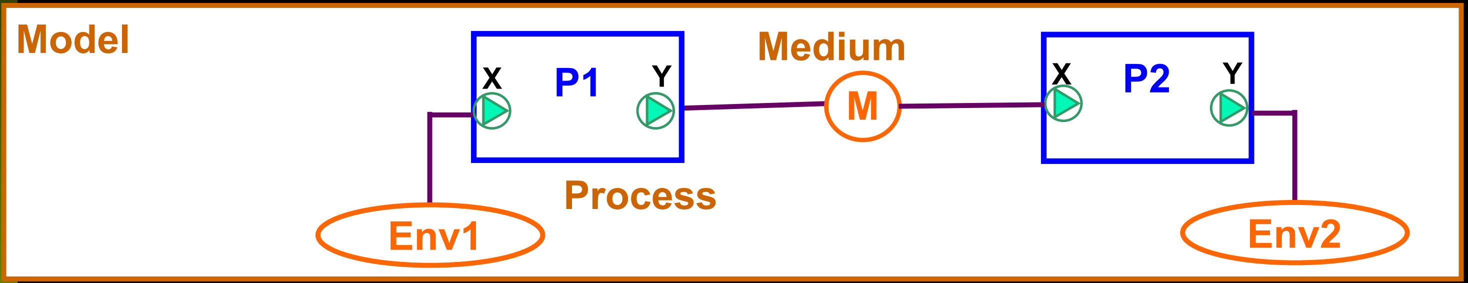
Concrete Semantics that Conform with the Process Networks Abstract Semantics

- Communicating Sequential Processes (CSP) [Hoare]
- Calculus of Concurrent Systems (CCS) [Milner]
- Kahn Process Networks (KPN) [Kahn]
- Nondeterministic extensions of KPN [Various]
- Actors [Hewitt]

Some Implementations:

- Occam, Lucid, and Ada languages
- Ptolemy Classic and Ptolemy II (PN and CSP domains)

Process Network Abstract Semantics in Metropolis




```
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(){ ... }
}
```

Leveraging Abstract Semantics for Joint Modeling of Architecture and Application

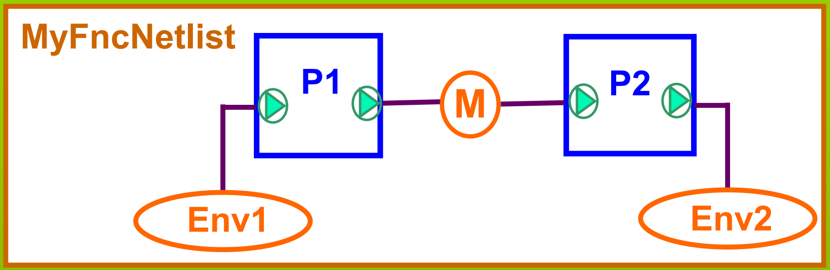
MyMapNetlist

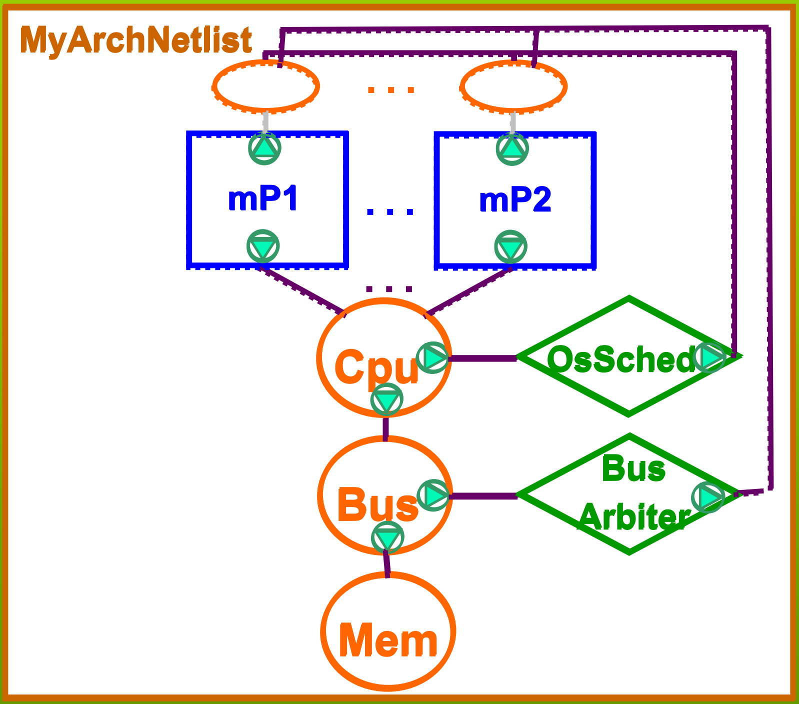
B(P1, M.write) <=> B(mP1, mP1.writeCpu); E(P1, M.write) <=> E(mP1, mP1.writeCpu);

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);





The abstract semantics provides natural points of the execution (where the monoid operations are invoked) that can be synchronized across models. Here, this is used to model operations of an application on a candidate implementation architecture.

A Finer Abstract Semantics

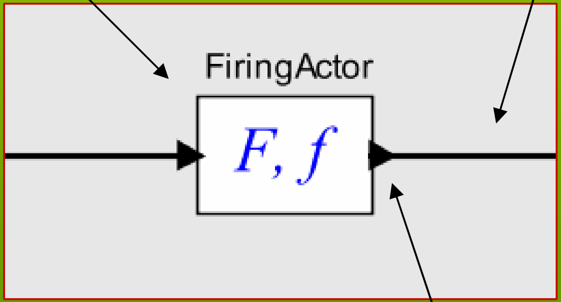
Firing Abstract Semantics:

a process still a function from input signals to output signals, but that function now is defined in terms of a firing function.

signals are monoids (can be incrementally constructed) (e.g. streams, discrete-event signals).

$$F : S_1 \rightarrow S_2$$

$$s_1 \in S_1$$



$$s_2 \in S_2$$

port is still either an input or an output.

The process function F is the least fixed point of a functional defined in terms of f.

Models of Computation that Conform to the Firing Abstract Semantics

- Dataflow models (all variations)
- Discrete-event models
- Time-driven models (Giotto)

In Ptolemy II, actors written to the *firing abstract semantics* can be used with directors that conform only to the process network abstract semantics.

Such actors are said to be *behaviorally polymorphic*.

A Still Finer Abstract Semantics

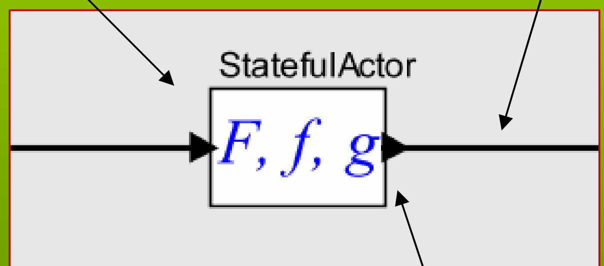
Stateful Firing Abstract Semantics:

a process still a function from input signals to output signals, but that function now is defined in terms of two functions.

signals are monoids (can be incrementally constructed) (e.g. streams, discrete-event signals).

$$F : S_1 \to S_2$$

$$s_1 \in S_1$$



$$s_2 \in S_2$$

$$f : S_1 \times \Sigma \to S_2$$

$$g : S_1 \times \Sigma \to \Sigma$$

state space

port is still either an input or an output.

The function f gives outputs in terms of inputs and the current state.
The function g updates the state.

Models of Computation that Conform to the Stateful Firing Abstract Semantics

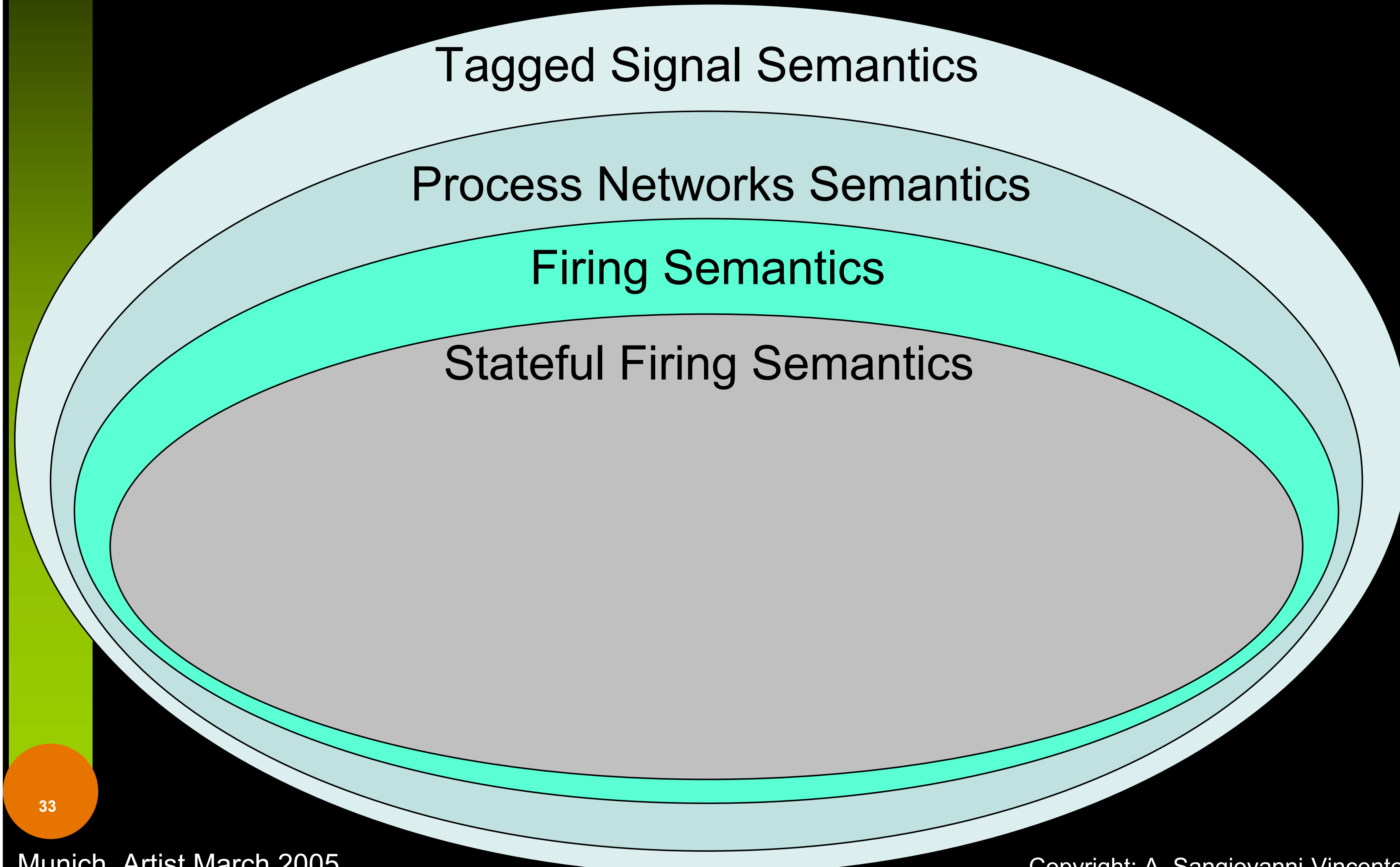
- Synchronous reactive
- Continuous time
- Hybrid systems

Stateful firing supports iteration to a fixed point, which is required for hybrid systems modeling.

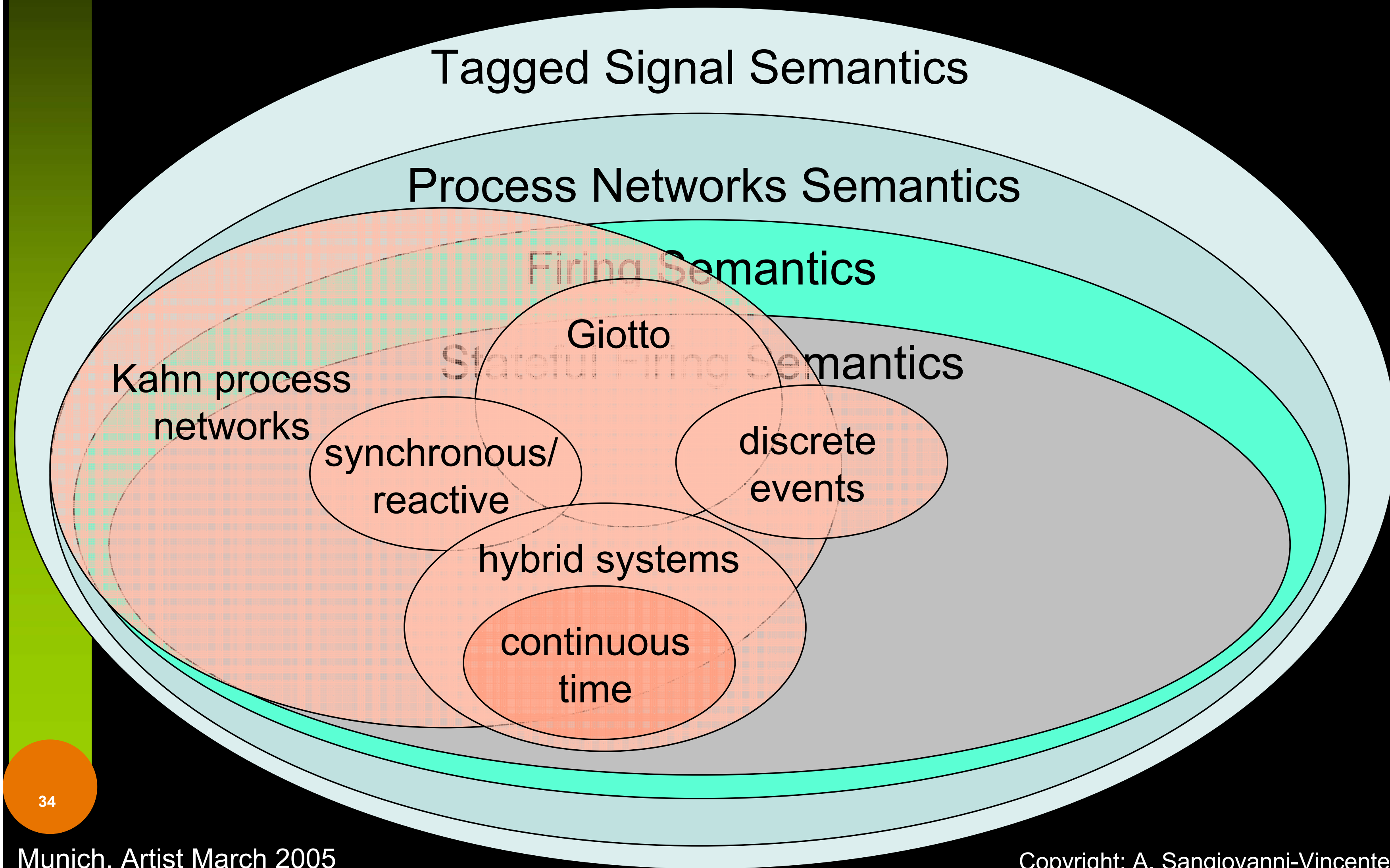
In Ptolemy II, actors written to the stateful firing abstract semantics can be used with directors that conform only to the firing abstract semantics or to the process network abstract semantics.

Such actors are said to be *behaviorally polymorphic*.

Where We Are

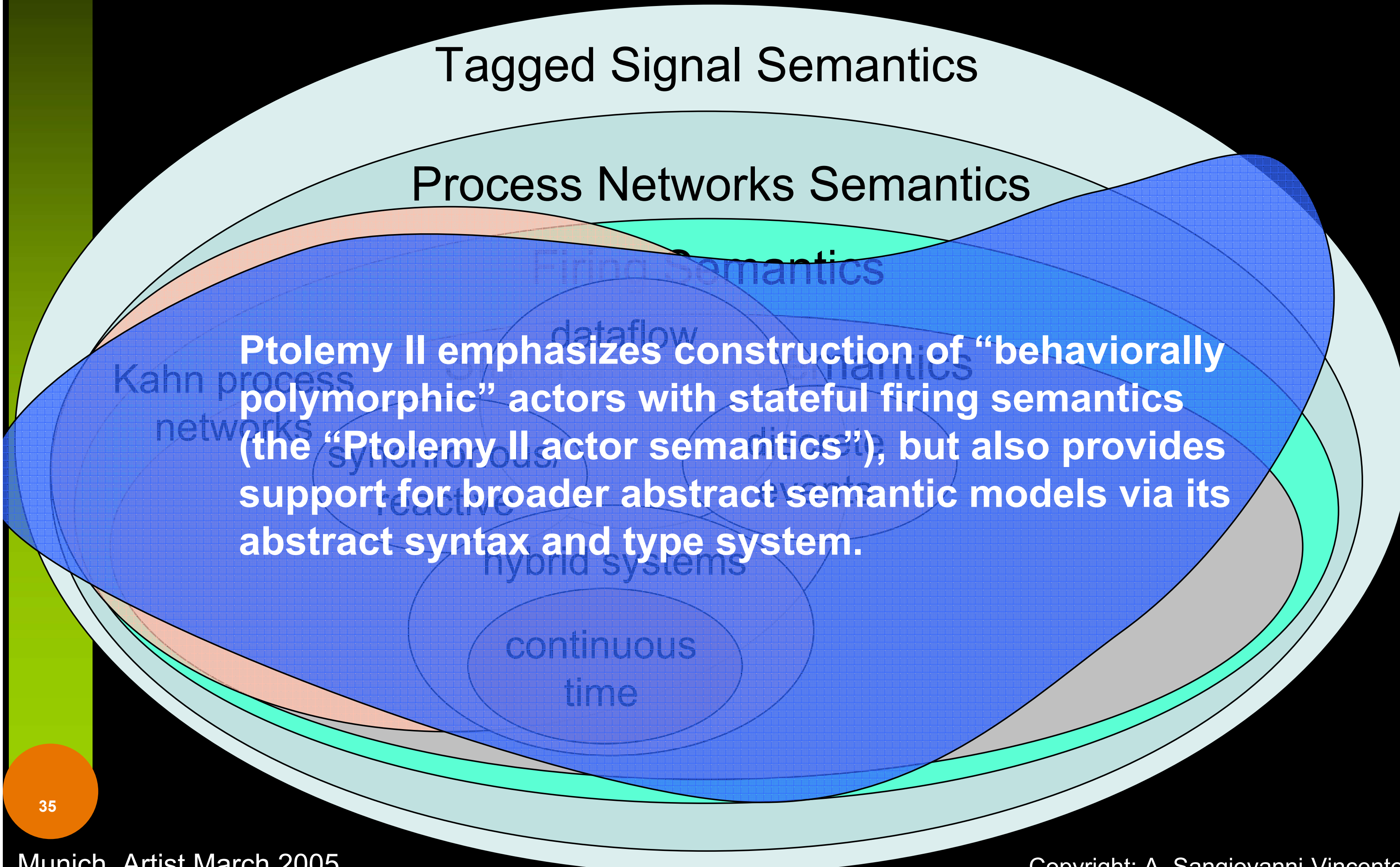

Tagged Signal Semantics
Process Networks Semantics
Firing Semantics
Stateful Firing Semantics

Where We Are



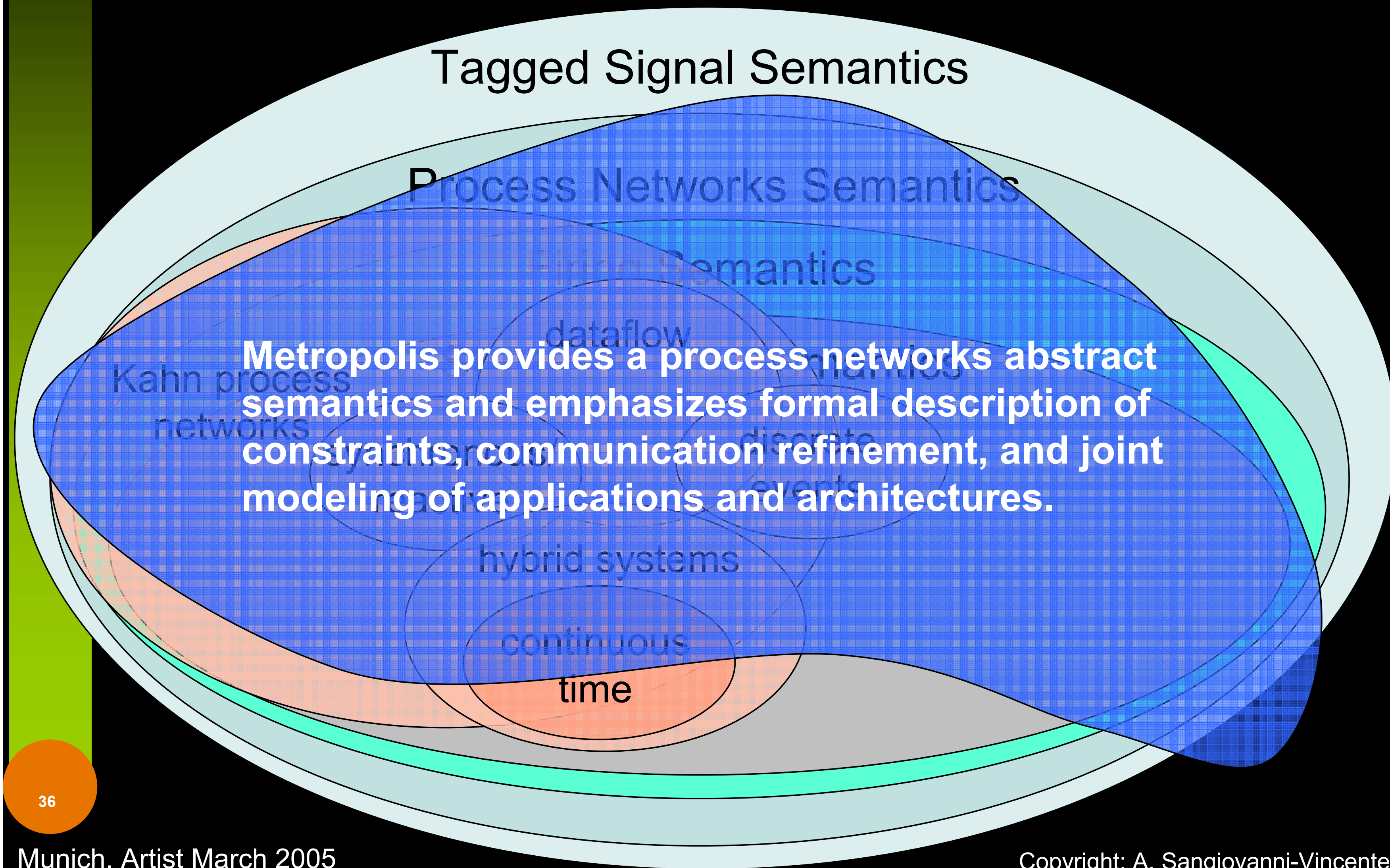
Tagged Signal Semantics
Process Networks Semantics
Firing Semantics
Stateful Firing Semantics
Giotto
Kahn process networks
synchronous/ reactive
discrete events
hybrid systems
continuous time

Meta Frameworks: Ptolemy II



Ptolemy II emphasizes construction of "behaviorally polymorphic" actors with stateful firing semantics (the "Ptolemy II actor semantics"), but also provides support for broader abstract semantic models via its abstract syntax and type system.

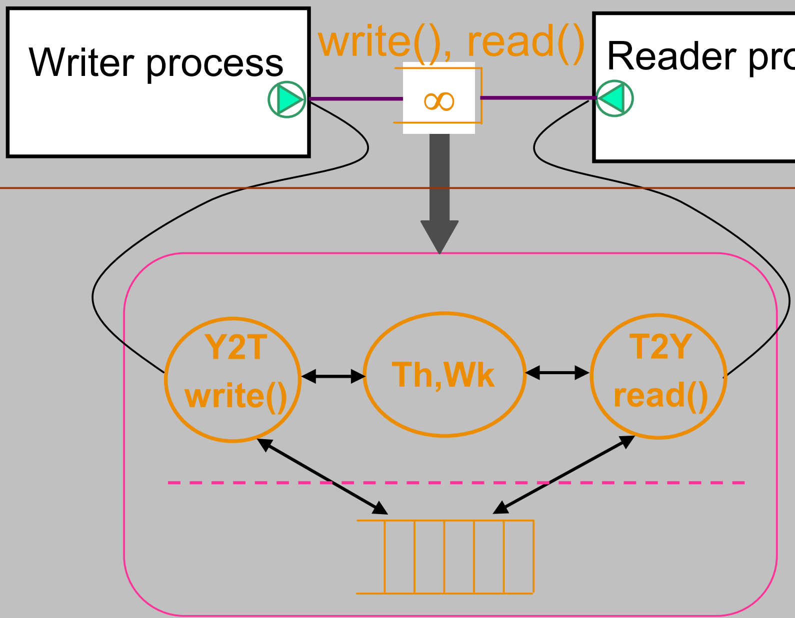
Meta Frameworks: Metropolis



Metropolis provides a process networks abstract semantics and emphasizes formal description of constraints, communication refinement, and joint modeling of applications and architectures.

Leveraging the Abstract Semantics for Refinement Verification in Metropolis

Example: a unbounded FIFO v.s. a bounded FIFO with the finer service.



Unbounded FIFO Level

Bounded FIFO Level

- Implement the upper level services using the current services

- Bounded FIFO API, e.g. release space, move data
- FIFO width and length parameterized

: refinement relation

- Metropolis represent both levels of abstraction explicitly, rather than replacing the upper level.
- Refinement relation is associated with properties to preserve through the refinement.

The Big Question:

How to Give Semantic Meta Models that are Usefully Manipulatable

Key ideas guiding us:

- Abstract semantics
- Ptolemy II directors
- Metropolis quantity managers
- The Metropolis language of constraints
- Interface theories
- Behavioral type systems
- Temporal logics (e.g. TLA)
- Set-valued semantics
- …

Conclusions

- Comparative study shows a fragmented landscape
  - Underlying models mostly incompatible
  - Key issues approached differently
- Consolidated view needed to advance the research
  - Evidence from industry using ad-hoc translators
  - Difficult for practitioners to choose the right model
- Our activities are complementary ways of approaching this problem
  - Solid and clean semantics (e.g., for hybrid systems)
  - Approximations for incompatible models

Concluding Remarks: Back to the Future?

Concluding Remarks: Back to the Future?

Renaissance!

Michelangelo: Piazza del Campidoglio, Roma

IP Re-use… Composability… Plug and Play

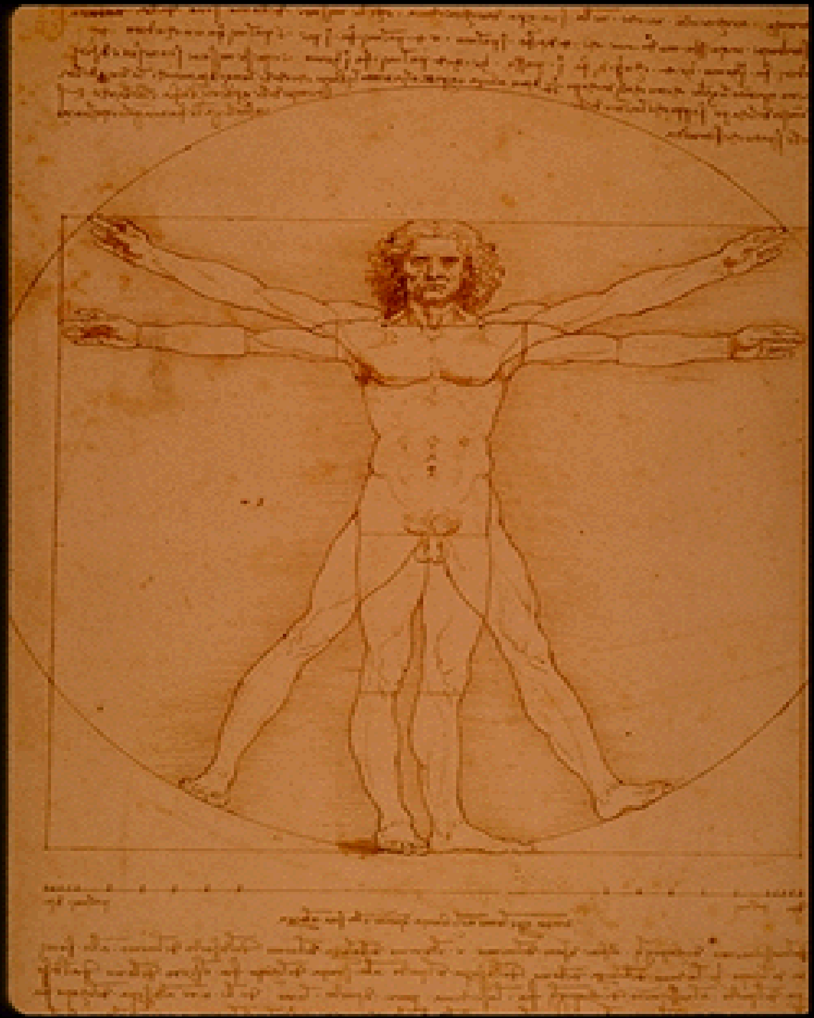
IP Re-use… Composability… Plug and Play

IP Re-use… Composability… Plug and Play

IP Re-use… Composability… Plug and Play

Educational Challenge

Educational Challenge



Educational Challenge

Educational Challenge