Block Diagrams for Modeling and Design

blockdiagrams.fm

Edward A. Lee
Professor

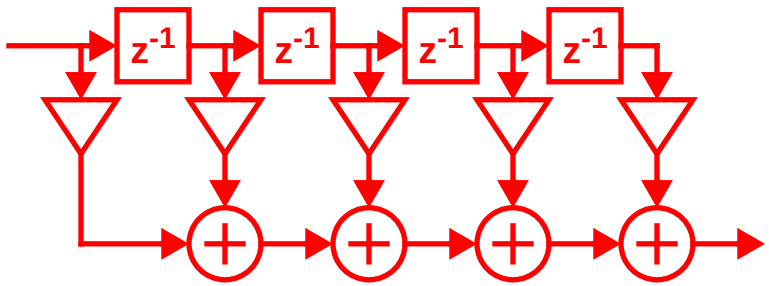
UC Berkeley
Dept. of EECS

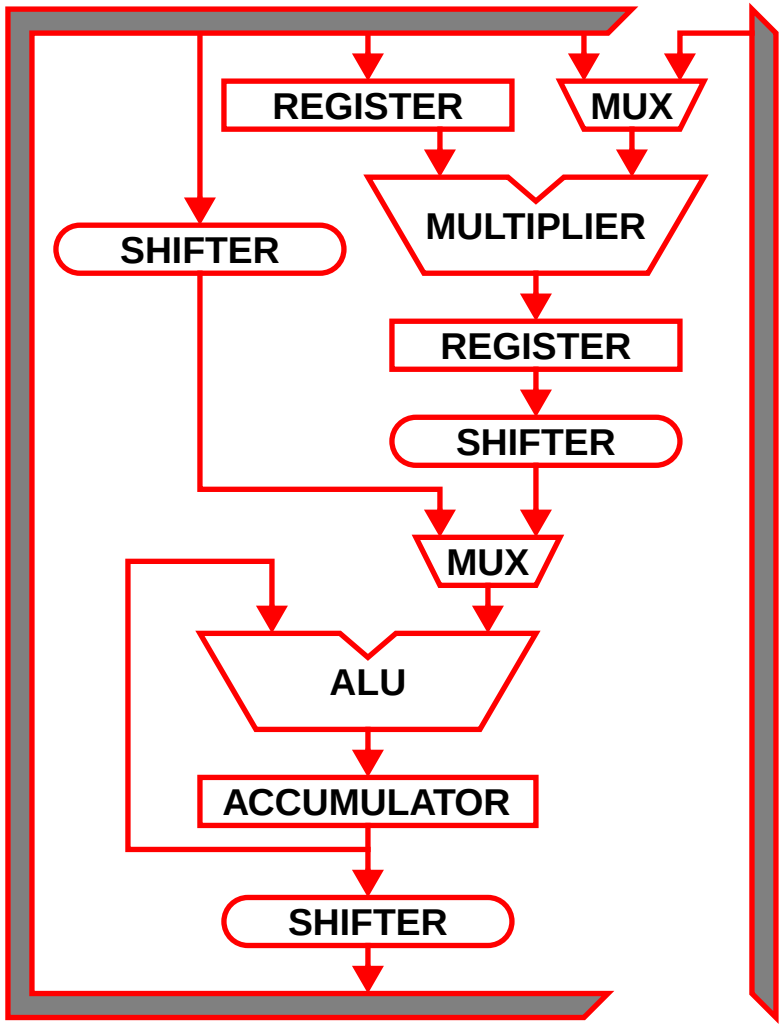
Abstract

Visual depictions of electronic systems have always held a strong human appeal, making them extremely effective in conveying information about a design. A few attempts to use such depictions to completely and formally specify systems have succeeded, most notably in circuit design, where schematic diagrams can capture all of the essential information needed to implement some systems. Others have failed dramatically, for example flowcharts for capturing the behavior of software. Recently, a number of innovative visual formalisms have been garnering support, including visual dataflow, hierarchical concurrent finite state machines, and object models. This talk focuses on the subset of these that are recognizable as "block diagrams." Such diagrams represent concurrent systems, but there are many possible concurrency semantics. Formalizing the semantics is essential if these diagrams are to be used for system specification and design. This talk explores some of the possible concurrency semantics, arguing that their strengths and weaknesses make them complementary rather than competitive, so that no single model is likely to emerge as a universally useful model. I will also describe some recent innovations where concurrency models are combined with automata for sequential control. So-called hybrid systems are a special case of such combinations.

Domains where Block Diagrams are Common

- Circuit schematics
- Computer architecture
- Dynamical systems
- Control theory
- Signal processing
- Communications



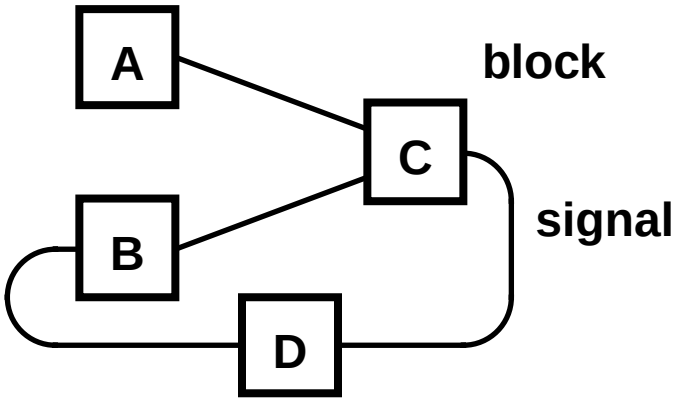


But the meaning of these diagrams can be quite different.

Properties of Block Diagrams

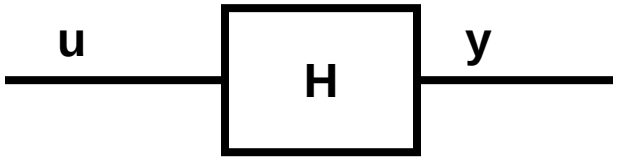
- **Modular**
  - Large designs are composed of smaller designs
  - Modules encapsulate specialized expertise
- **Hierarchical**
  - Composite designs themselves become modules
  - Modules may be very complicated
- **Concurrent**
  - Modules logically operate simultaneously
  - Implementations may be sequential or parallel or distributed
- **Abstract**
  - The interaction of modules occurs within a “model of computation”
  - Many interesting and useful MoCs have emerged
- **Domain Specific**
  - Expertise encapsulated in MoCs and libraries of modules.

Blocks and Signals



- **Blocks represent activities**
  - **May have inputs and outputs, or not**
  - **May be implemented concurrently, or not**
  - **Are conceptually concurrent**
- **Signals represent shared information**
  - **Shared variables**
  - **Functions of time**
  - **Sequences of tokens**
  - **Events in time**

Specifying Blocks



- **Denotationally:**
  - **A relation between signals (constraints on acceptable signals)**
  - **A function mapping input signals to output signals**

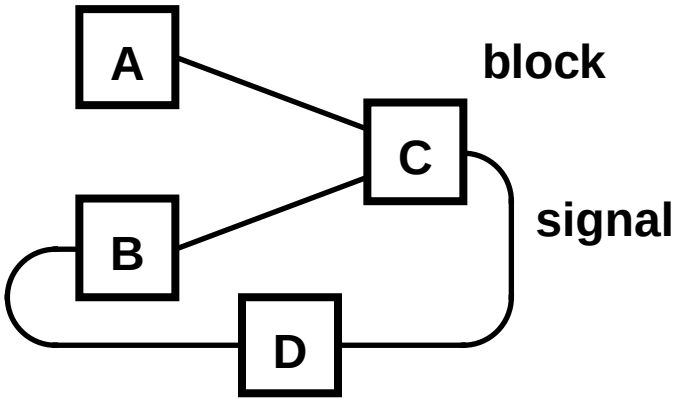
$$\textbf{e.g. } Y(z) = H(z)U(z)$$

- **Operationally:**
  - **Given observations of some signals, how do we change other signals**

$$\textbf{e.g. } \dot{\vec{x}}(n+1) = A\vec{x}(n) + \vec{b}u(n)$$

$$y(n) = \vec{c}^{\,T}\vec{x}(n) + du(n)$$

Semantics



The *meaning* of an interconnection of blocks (a *system*)

- **Denotational semantics:**

  The set of properties that signals must have in a particular interconnection

- **Operational semantics:**

  How to compute the signal values for a particular interconnection

Determinacy

- A *behavior* of a system is a set of signal values that obeys the semantics.
- A system is *determinate* if knowing the inputs, there is at most one behavior.
- A system is *receptive* if for all inputs there is at least one behavior.
- A semantics is determinate if all systems are determinate.

Nondeterminacy can be useful in *modeling*: a family of behaviors is described and analyzed compactly.

However, nondeterminism is risky in *design* if it means that behavior is underspecified.

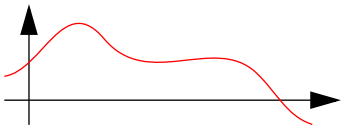
Nondeterminacy can be viewed as a family of behaviors.

Some Candidate Semantics

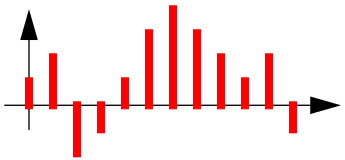
1. Analog computers (differential equations)
2. Discrete time (difference equations)
3. Discrete-event systems
4. Synchronous-reactive systems
5. Process networks
6. Dataflow
7. Sequential processes that rendezvous

Basic claim of this talk: each of these has its place.

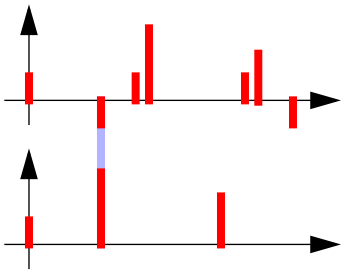
Essential Differences — Models of Time



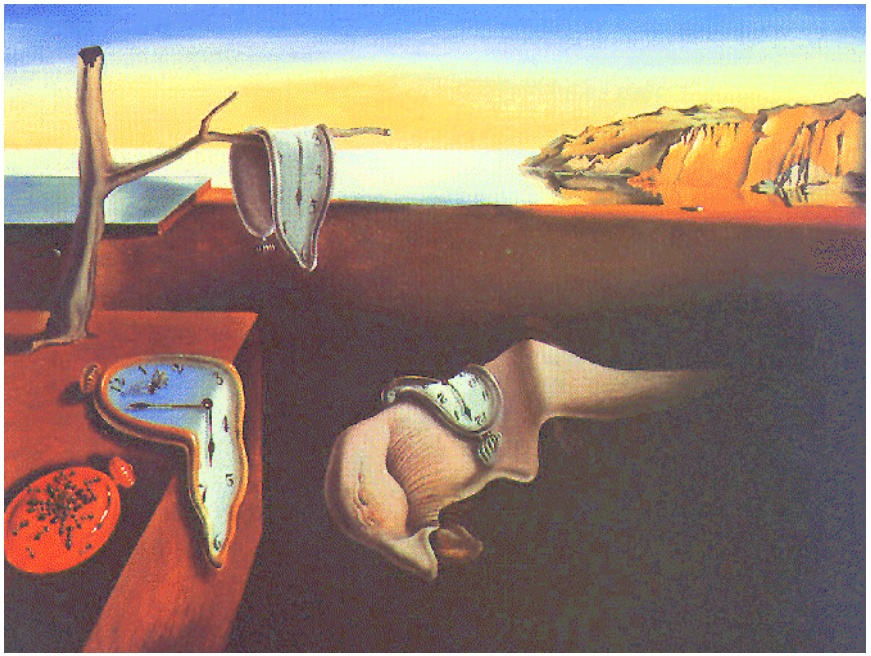
continuous time



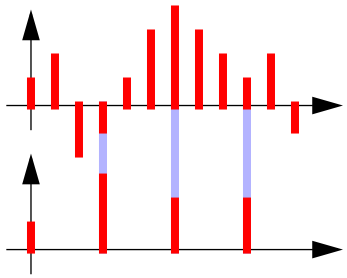
discrete time



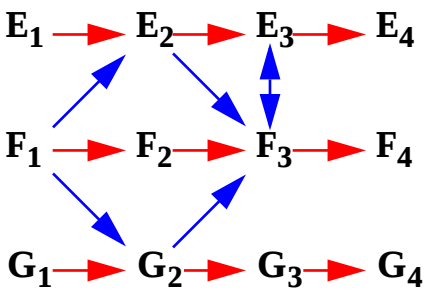
totally-ordered discrete events



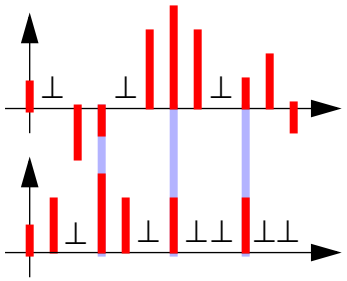
Salvador Dali, *The Persistence of Memory*, 1931



multirate discrete time



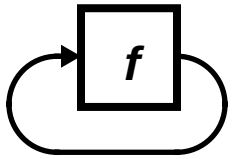
partially-ordered discrete events



synchronous/reactive

Key Semantic Issues

- **Does a composition of blocks have a behavior? More than one behavior?**
  - **Typical hard case:**



Denotationally, the behavior here is a signal that is fixed point of *f*.

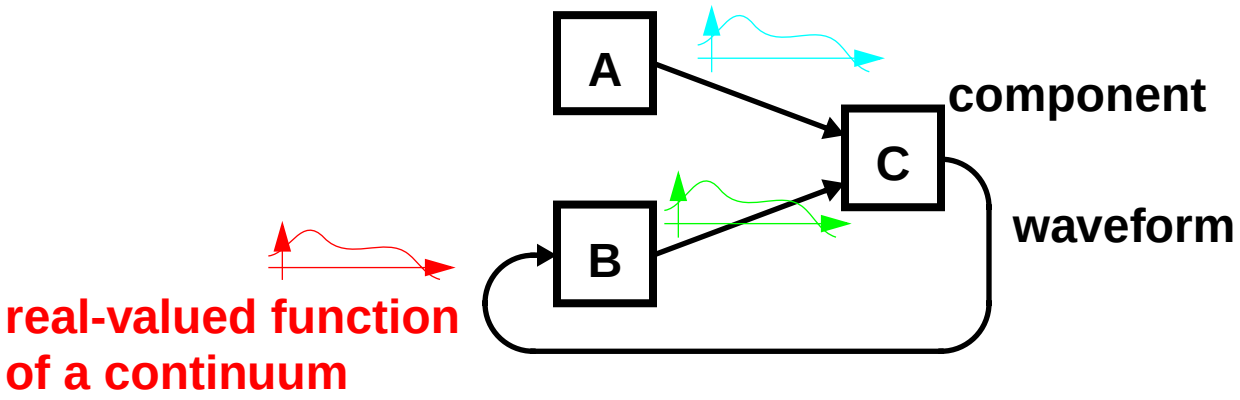
- **Can a simulation or analysis strategy find a behavior? All behaviors? A subset of behaviors satisfying some property?**
  - **The "strategy" is an operational semantics, and we need to know whether this semantics is the same as the denotational semantics (full abstraction).**

- **Does a block diagram have the same semantics as a block?**
  - **This is sometimes called the "compositionality" property.**

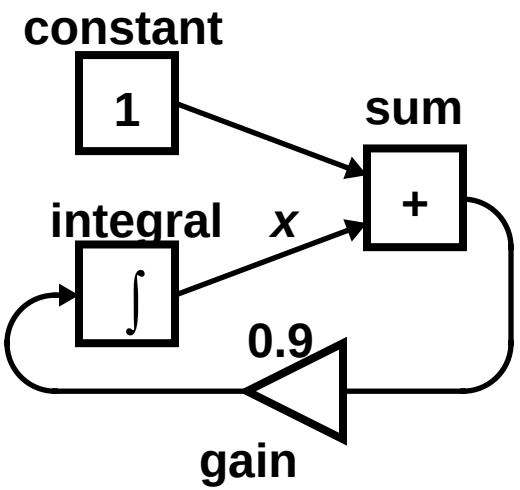
Key Practical Issues

- Can it be simulated?
  - Bounded memory
  - Bounded time for at least a partial solution
  - Simulation speed
- Can it be implemented?
  - Bounded memory
  - Bounded time for at least a partial solution
  - Synthesis algorithms
- How many ways can it be implemented?
  - Software vs. hardware
  - Parallel vs. sequential
  - Scheduling algorithms
  - Avoiding overspeficiation

1. Analog Computers



Example: First-order differential equation:



$$\dot{x} = 0.9x + 0.9$$

Properties

Semantics:

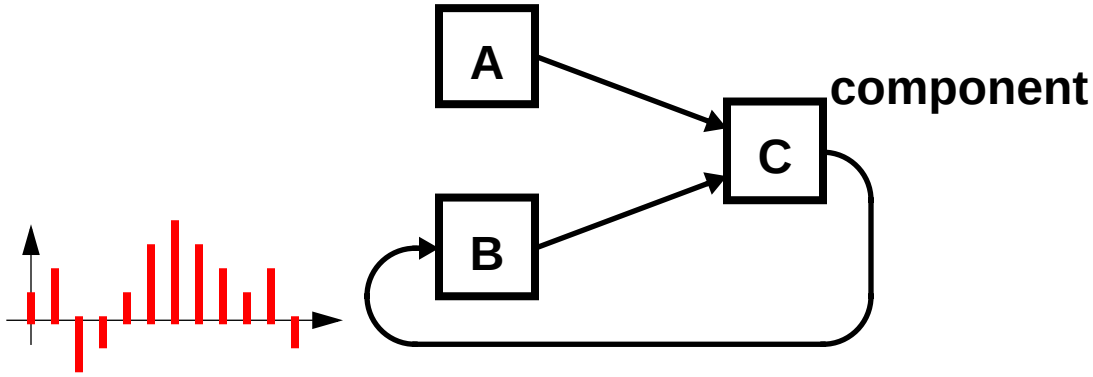
- blocks are relations between functions of time
- fixed point is a set of functions of time satisfying these relations

Strengths:

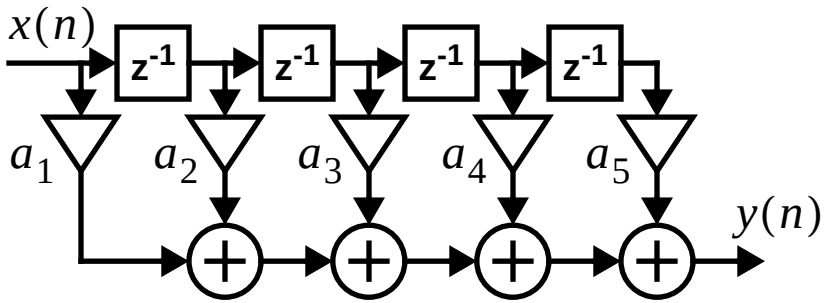
- Accurate model for many physical systems
- Determinate under simple conditions (strict causality in feedback loops)
- Established and mature (approximate) simulation techniques

Weaknesses:

- Covers a narrow application domain
- Tightly bound to an implementation
- Relatively expensive to simulate
- Difficult to implement in software



Example: Difference equation:



$$y(n) = a_1 x(n) + a_2 x(n-1) + a_3 x(n-2) + a_4 x(n-3) + a_5 x(n-4)$$

Properties

Semantics:

- blocks are relations between functions of discrete time
- fixed point is a set of functions of discrete time satisfying these relations

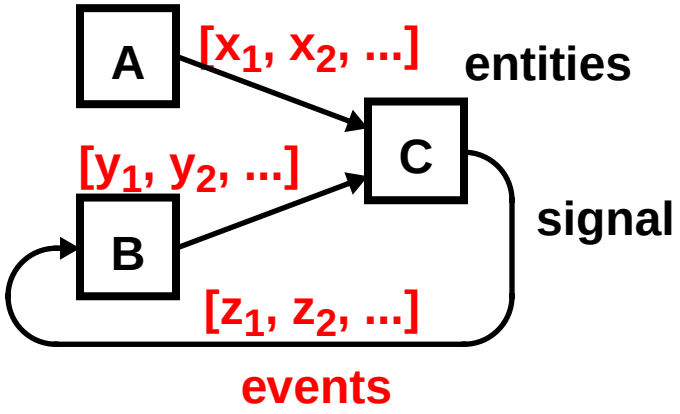
Strengths:

- Useful model for many embedded signal processing systems
- Determinate under simple conditions (strict causality in feedback loops)
- Easy simulation (cycle-based)
- Easy implementation (synchronous circuits or software)

Weaknesses:

- Covers a narrow application domain
- Global synchrony may overspecify some systems

3. Discrete-Event Models

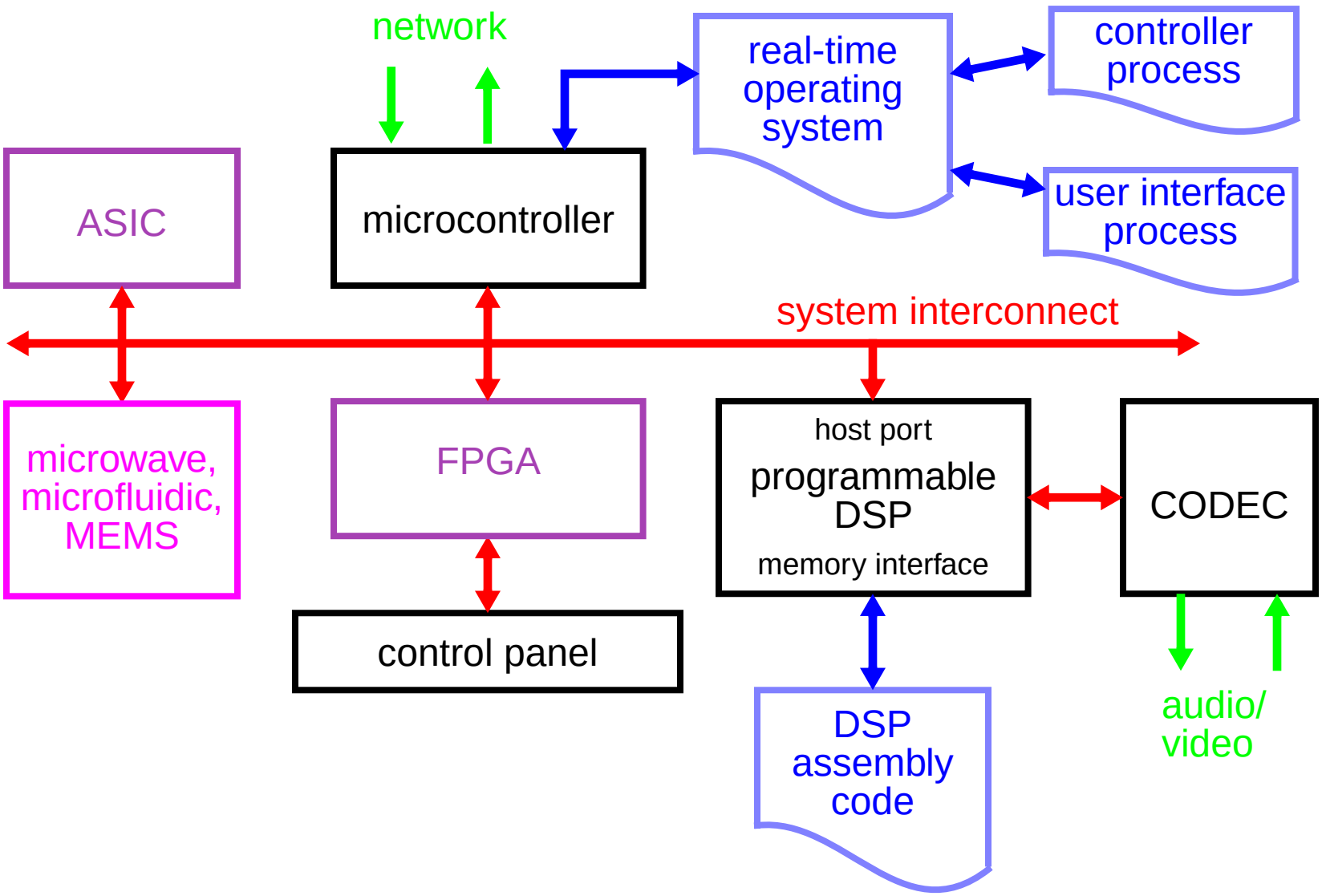


Events occur at discrete points on a time line that is usually a continuum. The entities react to events in chronological order.

Example application areas:

- **Communication networks**
- **Queueing systems**
- **Manufacturing systems**
- **Hardware architecture**

Example: Hardware Architecture



Properties

Semantics:

- Signals are sets of events placed in time (finite or infinite)
- Blocks are relations between signals
- Fixed point is a set of signals

Strengths:

- Natural description of asynchronous digital hardware
- Global synchronization
- Determinate under simple conditions (strict causality in feedback loops)
- Simulatable under simple conditions (delta causality in feedback loops)

Weaknesses:

- Expensive to implement in software
- May over-specify and/or over-model systems (global time)

Machinery for Studying Semantics of 1,2, and 3

- **The Cantor metric:**

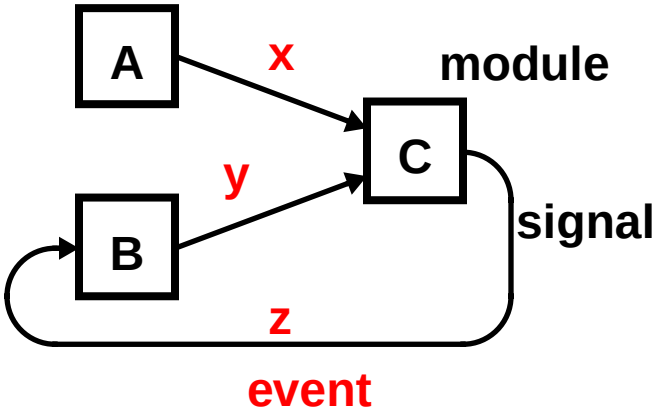
  $$d(s_1, s_2) = \frac{1}{2^{\tau}},$$

  where τ is the glb of the times where s_1 and s_2 differ.

- **Metric space theorems provide conditions for the existence and uniqueness of fixed points.**

***Example result*: VHDL (a DE language) permits programs where a fixed point exists but no simulator can find it.**

4. Synchronous/Reactive Models



A discrete model of time progresses as a sequence of "ticks." At a tick, the signals are defined by a fixed point equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_{A,t}(1) \\ f_{B,t}(z) \\ f_{C,t}(x, y) \end{bmatrix}$$

Application areas:

- **Anything with elaborate control logic**
- **User interfaces**

Properties

Semantics:

- Each tick represents a new fixed point computation
- Convergence to fixed points (when possible) is finite

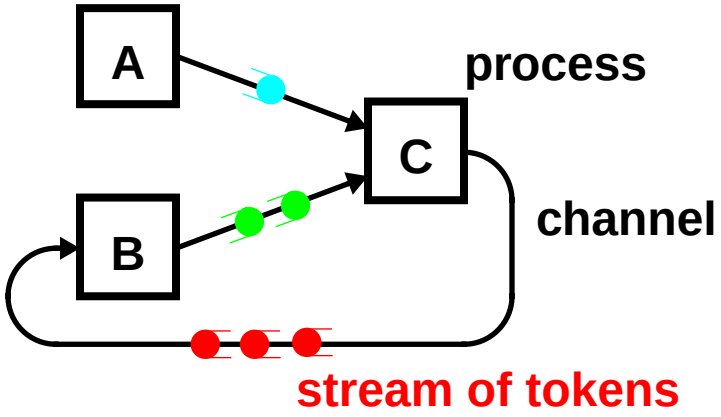
Strengths:

- Good match for control-intensive systems
- Tightly synchronized
- Determinate in most cases (use constructive semantics)
- Maps well to hardware and software

Weaknesses:

- Computation-intensive systems are overspecified
- Modularity is compromised
- Causality loops are possible (no fixed point or multiple fixed points)
- Causality loops are hard to detect

5. Process Networks



Possible application areas:

- **User interfaces (determinate replacement for threads)**
- **Asynchronous, multitasking, reactive systems**

Process networks, *per se*, are not actually used (to my knowledge) today. But the understanding of efficient implementations is very recent, and they hold much promise (IMO).

Properties

Semantics:

- Blocks are relations between (possibly infinite) sequences
- Operationally: sequences are constructed token by token
- Any finite execution produces a prefix of the denotation

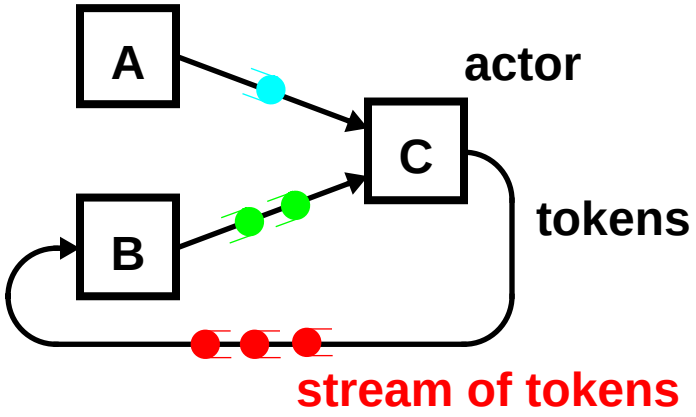
Strengths:

- Loose synchronization (distributable)
- Determinate under simple conditions (monotonic processes)
- Implementable under simple conditions (continuous processes)
- Maps easily to threads, but much easier to use
- Turing complete (expressive)

Weaknesses:

- Control-intensive systems are hard to specify
- Turing complete (deadlock and bounded memory are undecidable)

6. Dataflow

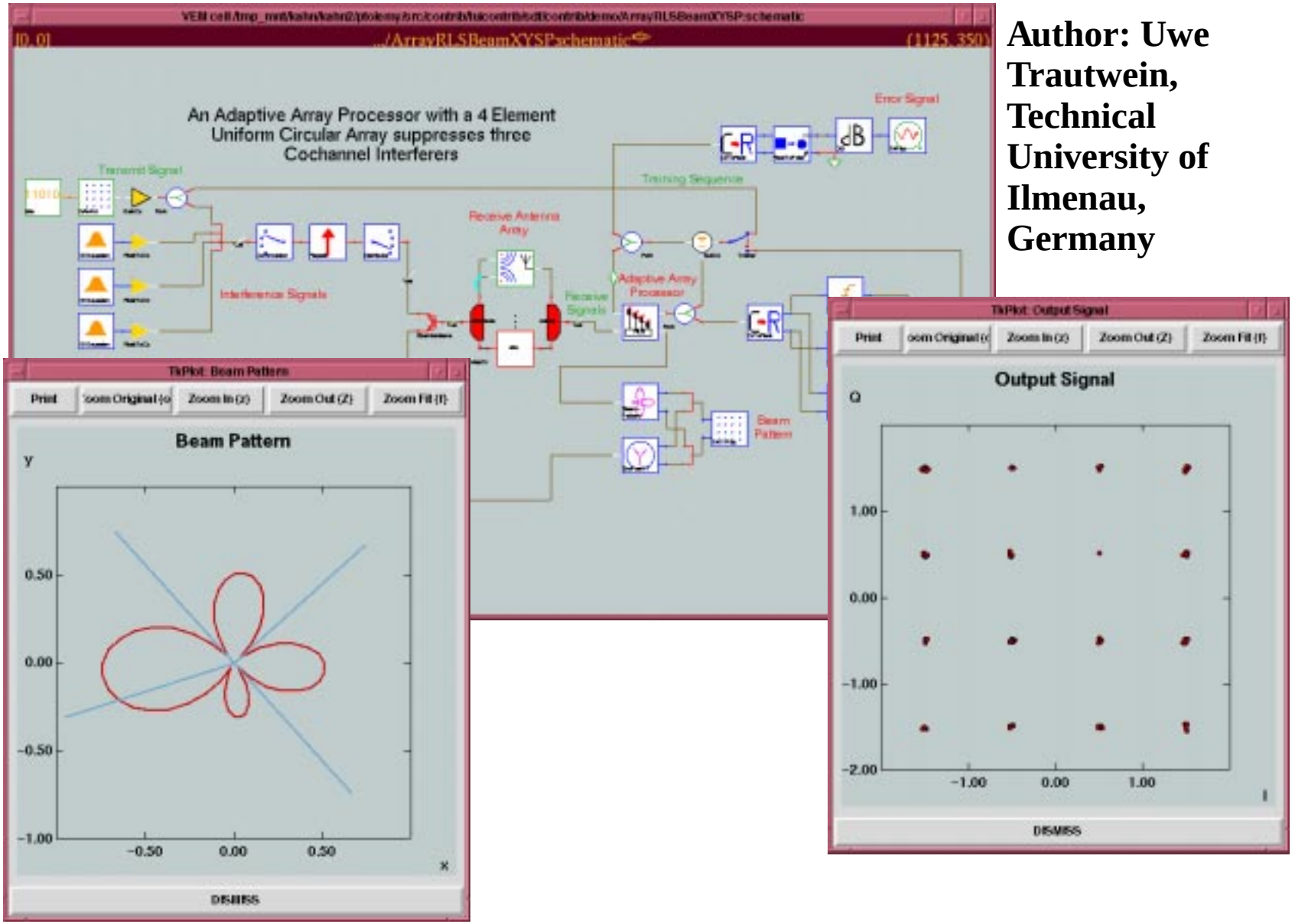


A special case of process networks where a process is made up of a sequence of firings (finite, atomic computations).

Application areas:

- **Signal processing**
- **Computer architecture (dynamic instruction scheduling)**
- **Compilers (an analysis technique)**

Dataflow for Signal Processing



Author: Uwe Trautwein, Technical University of Ilmenau, Germany

Properties

Semantics:

- Firing functions are composed using higher-order functions to get processes
- Decidable special case: “synchronous dataflow”

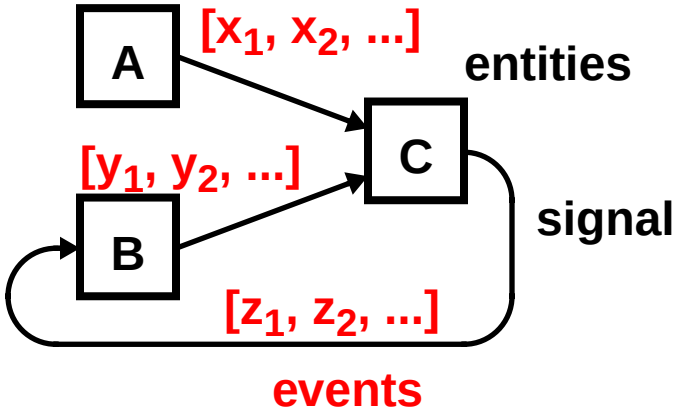
Strengths:

- Good match for signal processing
- Loose synchronization (distributable)
- Determinate under simple conditions
- Special cases map well to hardware and embedded software

Weaknesses:

- Control-intensive systems are hard to specify

7. Rendezvous Models



Events represent rendezvous of a sender and a receiver. Communication is unbuffered and instantaneous. Examples include CSP and CCS.

Application areas:

- **Client/server systems**
- **Object-request brokers**
- **Resource sharing**

Properties

Semantics:

- Rendezvous is atomic (indivisible)
- Traces (interleavings of rendezvous events)

Strengths:

- Models resource sharing well
- Partial-order synchronization (distributable)
- Supports naturally nondeterminate interactions

Weaknesses:

- Oversynchronizes some systems
- Difficult to make determinate (and useful)

A Key Property of Block Diagrams

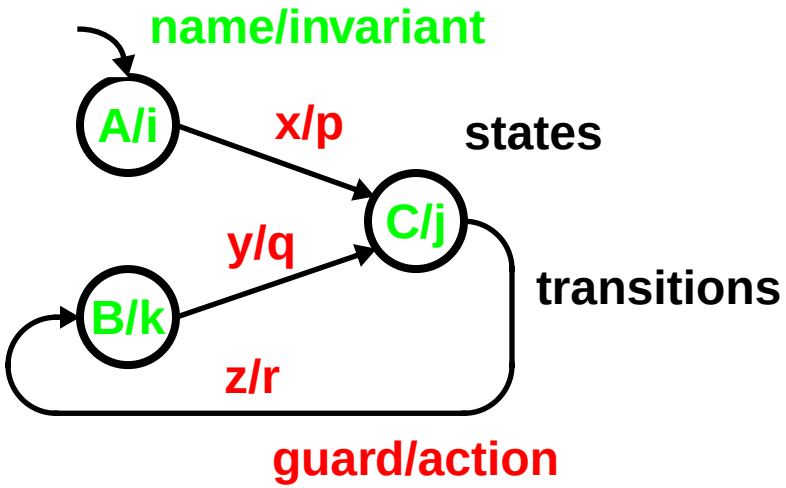
They are Static!

A consequence is that they are typically used in application areas where fixed algorithms prevail for all time:

- **Circuits**
- **Computer architecture**
- **Dynamical systems**
- **Control theory**
- **Signal processing**
- **Communications**

We can generalize them by hierarchically combining with automata.

Sequential Example — Finite State Machines



Guards specify when a transition may be made from one state to another, and actions assert events. Invariants specify when remaining a state is allowed.

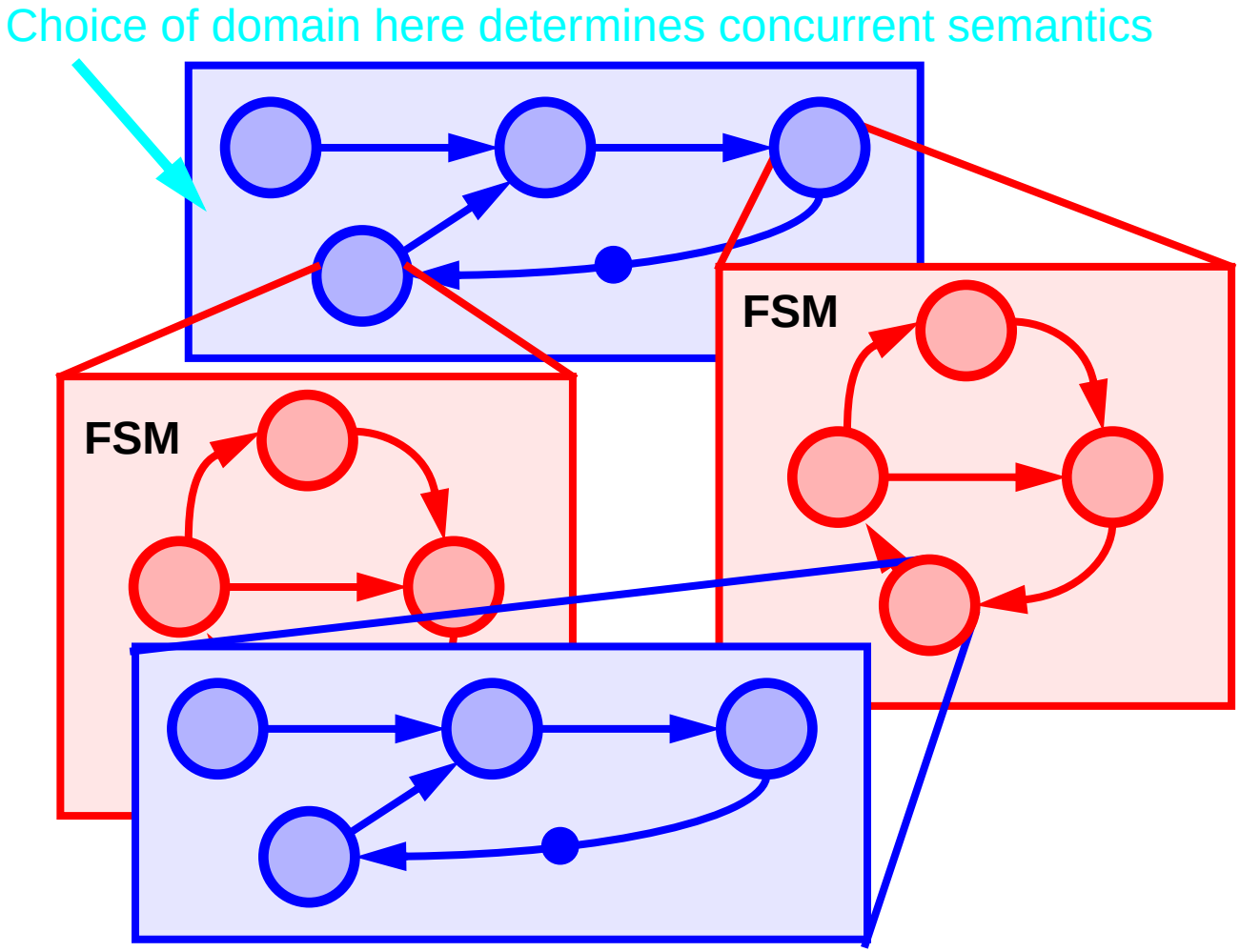
Strengths:

- Natural description of sequential control
- Behavior is decidable
- Can be made determinate (often is not, however)
- Easy to implement in hardware or software

Weaknesses:

- Awkward to specify numeric computation
- Size of the state space can get large

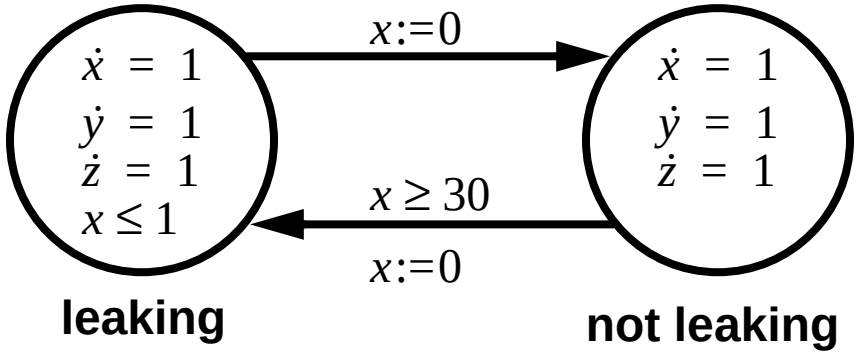
Mixing Control and Concurrency — *Charts



Hybrid Systems

- **A discrete program combined with an analog system**
- **A combination of automata and analog computers**

Traditional syntax (example: leaking gas burner):

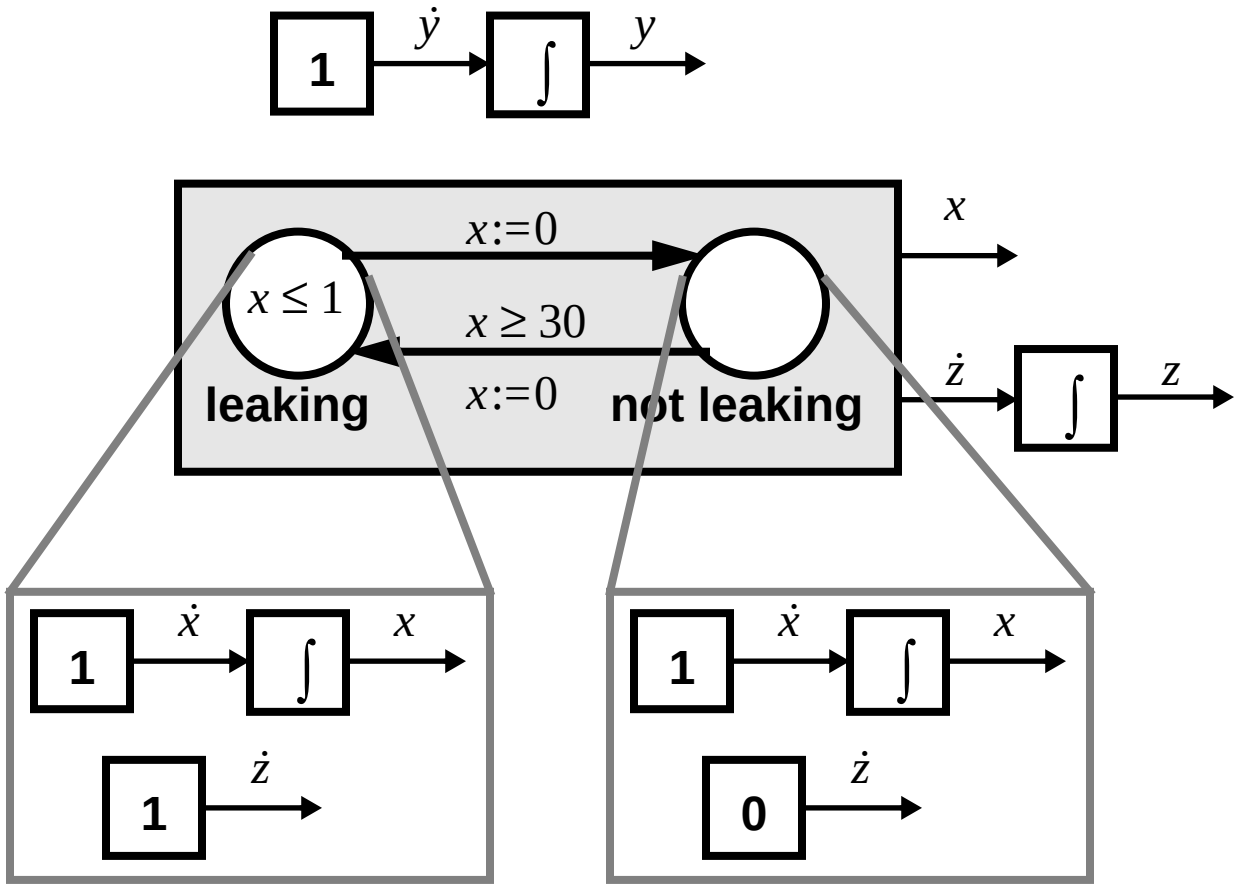


Here, the differential equations hardly look like a concurrency model, but in fact, in a trivial way, they are.

Alternative View of Hybrid Systems

***charts with analog computers as the concurrency model and a particular style of nondeterminate automata.**

For example (leaking gas burner):

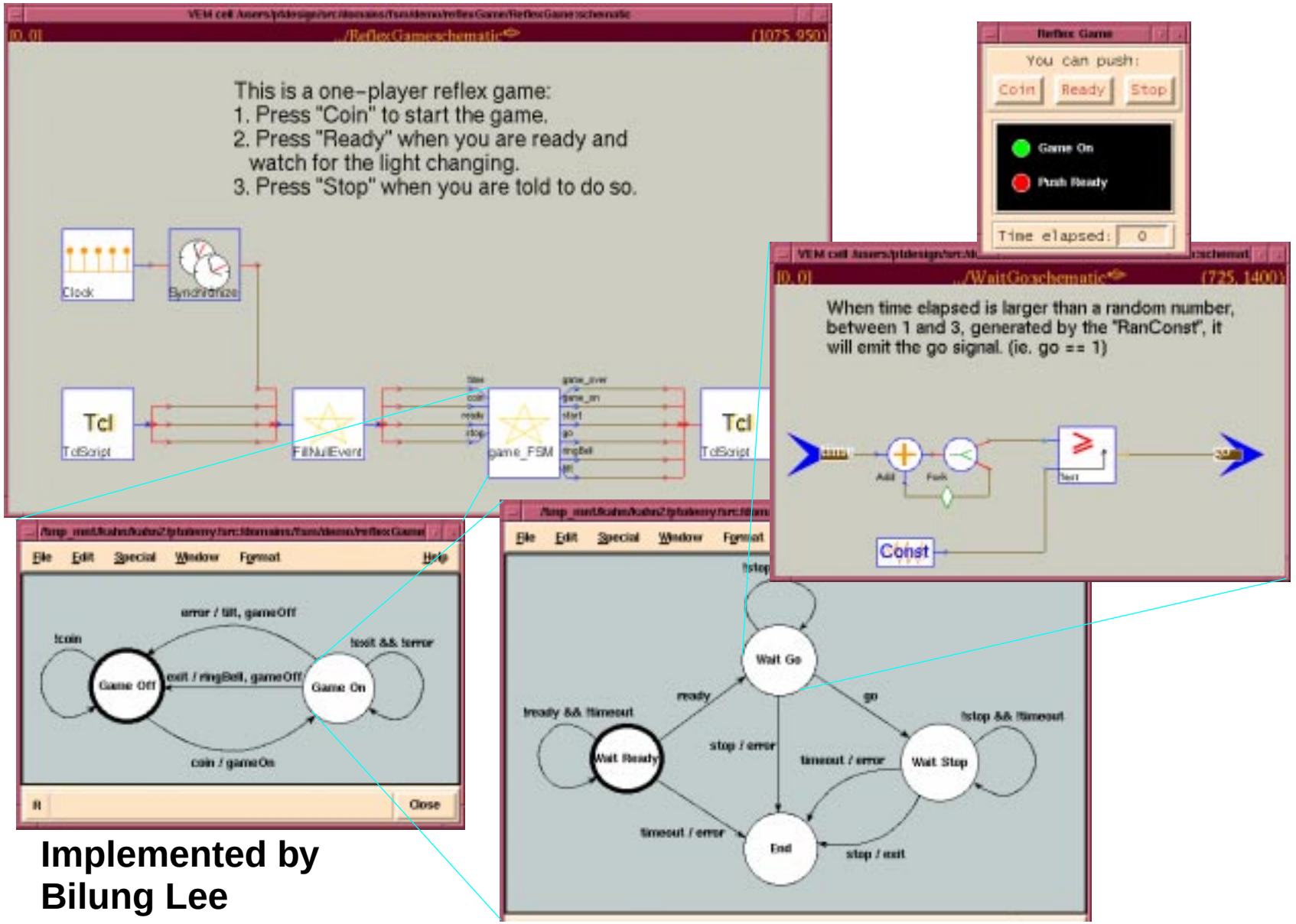


In general

- A concurrent system contains modules that are internally automata.
- States of an automaton contain subsystems defined in a concurrent semantics.
- Transitions and guards depend on variables in the subsystems as well as inputs to the automaton.
- Transitions have actions on the subsystems of the destination state.
- Multiple states may share the same subsystem.

- If multiple concurrent semantics can be nested (as in Ptolemy), then subsystems need not have the same the semantics as the system containing the automaton.

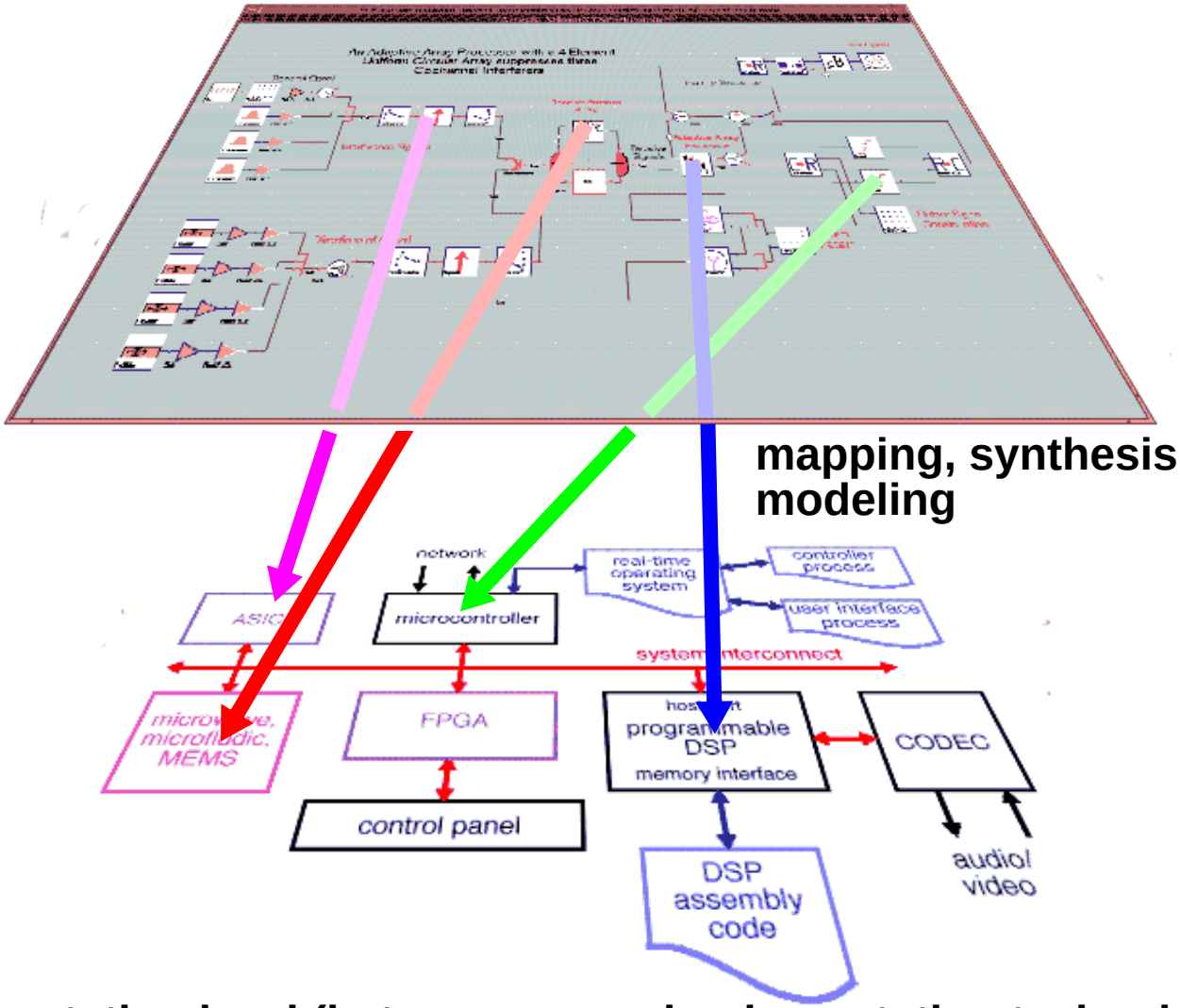
Example: DE, Dataflow, and FSMs



Implemented by Bilung Lee

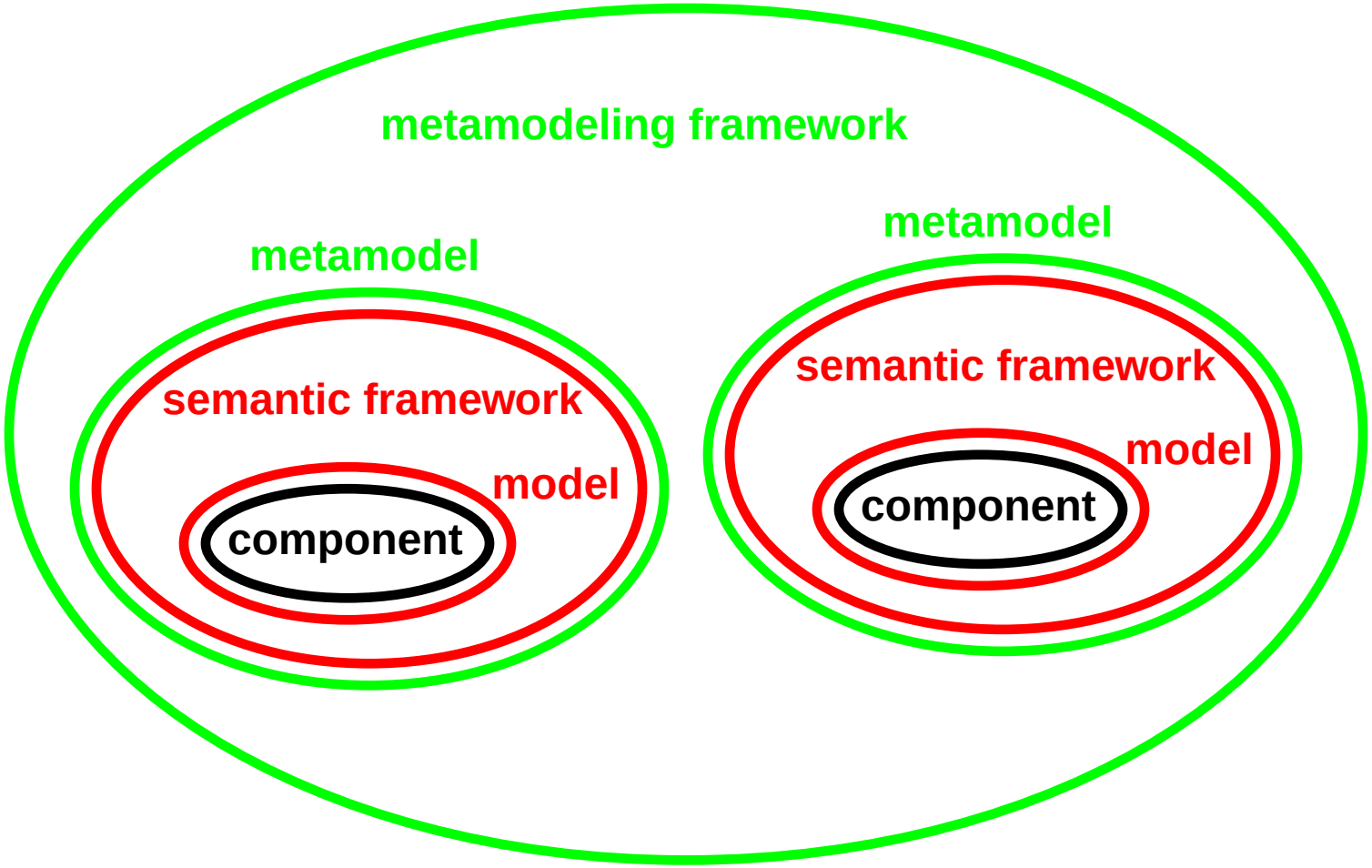
Heterogeneous System-Level Specification & Modeling

problem level (heterogeneous models of computation)



implementation level (heterogeneous implementation technologies)

Metamodeling



More Information

The following papers by the speaker give more detail and lots of references:

http://ptolemy.eecs.berkeley.edu/papers/...

- **97/preliminaryStarcharts/**

  (on automata combined with concurrency)
- **97/denotational/**

  (on comparing concurrency semantics)
- **97/dataflow/**

  (on the semantics of dataflow)
- **98/realtime/**

  (on the semantics of discrete events)