Modeling Heterogeneous Systems

- Design for Understanding -

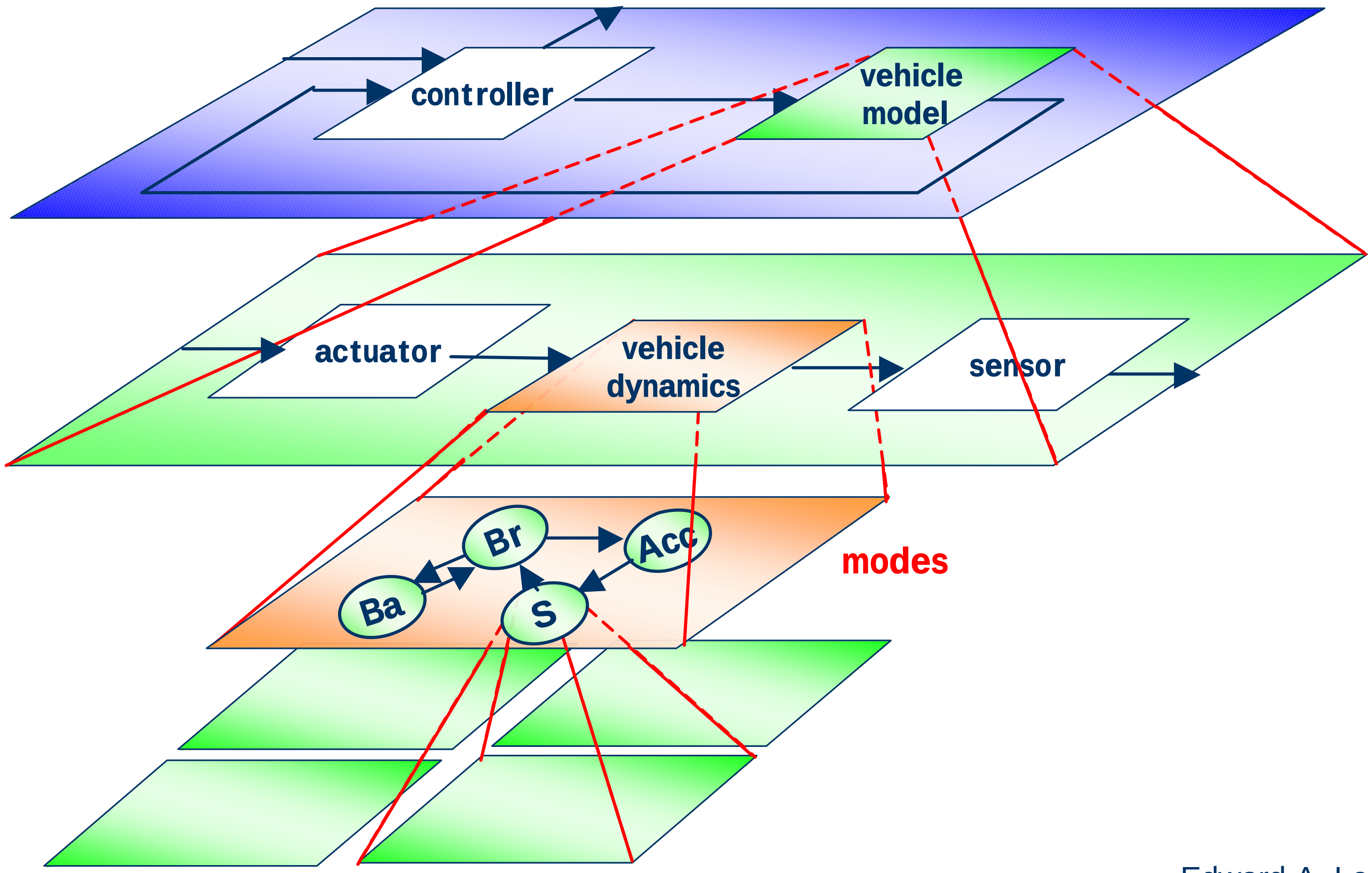
Design for Safety Workshop

NASA Ames Research Center
Mountain View, CA

11 October, 2000

Edward Lee
UC Berkeley

Components and Composition



Common Approaches

- Threads or processes
  - Sun says in the on-line Java tutorial:

    “The first rule of using threads is this: avoid them if you can. Threads can be difficult to use, and they tend to make programs harder to debug.”
- Semaphores, monitors, mutex
  - Deadlock, livelock, liveness – hard to understand
- Priorities, deadlines
  - Plug and pray

Understanding Component Interactions - Frameworks

- What is a component (ontology)?
  - States? Processes? Threads? Differential equations? Constraints? Objects (data + methods)?
- What knowledge do components share (epistemology)?
  - Time? Name spaces? Signals? State?
- How do components communicate (protocols)?
  - Rendezvous? Message passing? Continuous-time signals? Streams? Method calls?
- What do components communicate (lexicon)?
  - Objects? Transfer of control? Data structures? ASCII text?

A Laboratory for Exploring Component Frameworks

Ptolemy II –

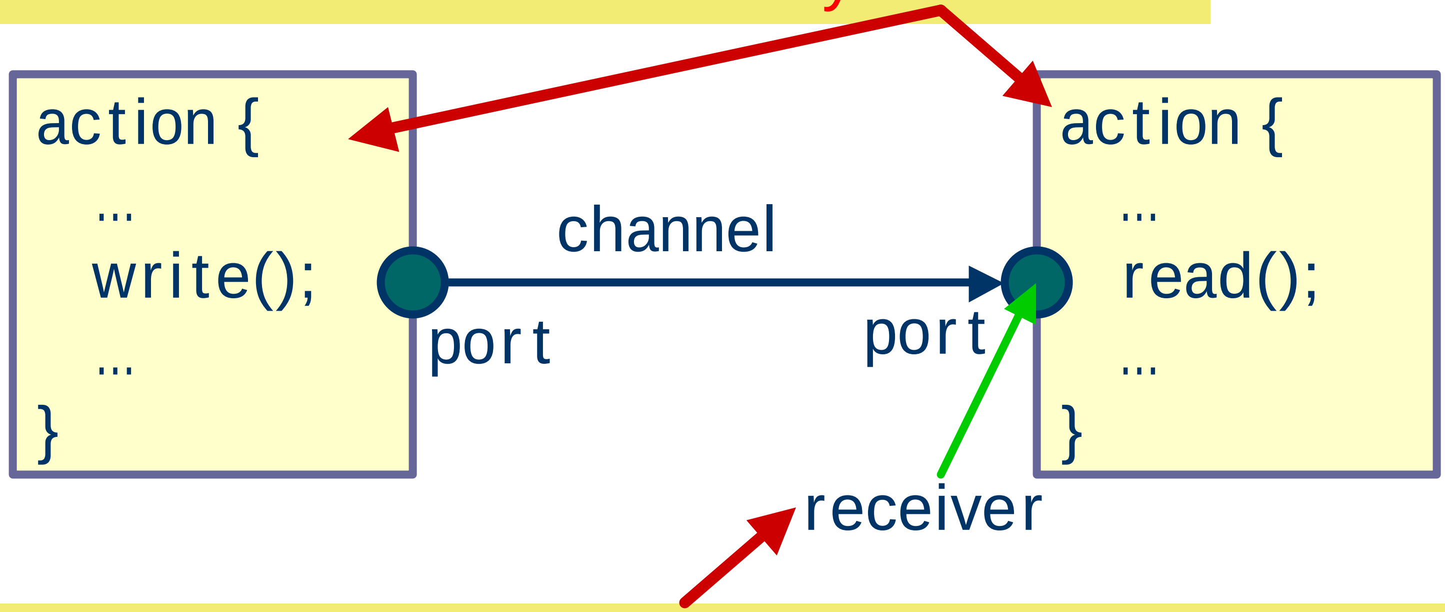
- Java based, network integrated
- Several frameworks implemented

A realization of a framework is called a “domain.” Multiple domains can be mixed hierarchically in the same model.

http://ptolemy.eecs.berkeley.edu

One Class of Component Interaction Semantics: Producer / Consumer

Are actors active? passive? reactive?
Flow of control is mediated by a *director*.



Are communications timed? synchronized? buffered?
Communications are mediated by *receivers*.

Domain – A Realization of a Component Framework

- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DT – discrete time (cycle driven)
- PN – process networks
- PN' – Petri nets
- SDF – synchronous dataflow
- SR – synchronous/reactive
- PS – publish-and-subscribe

Each is realized as a director and a receiver class in Ptolemy II

Each of these defines a component ontology and an interaction semantics between components. There are many more possibilities!

1. Continuous Time (Coupled ODEs)

Semantics:

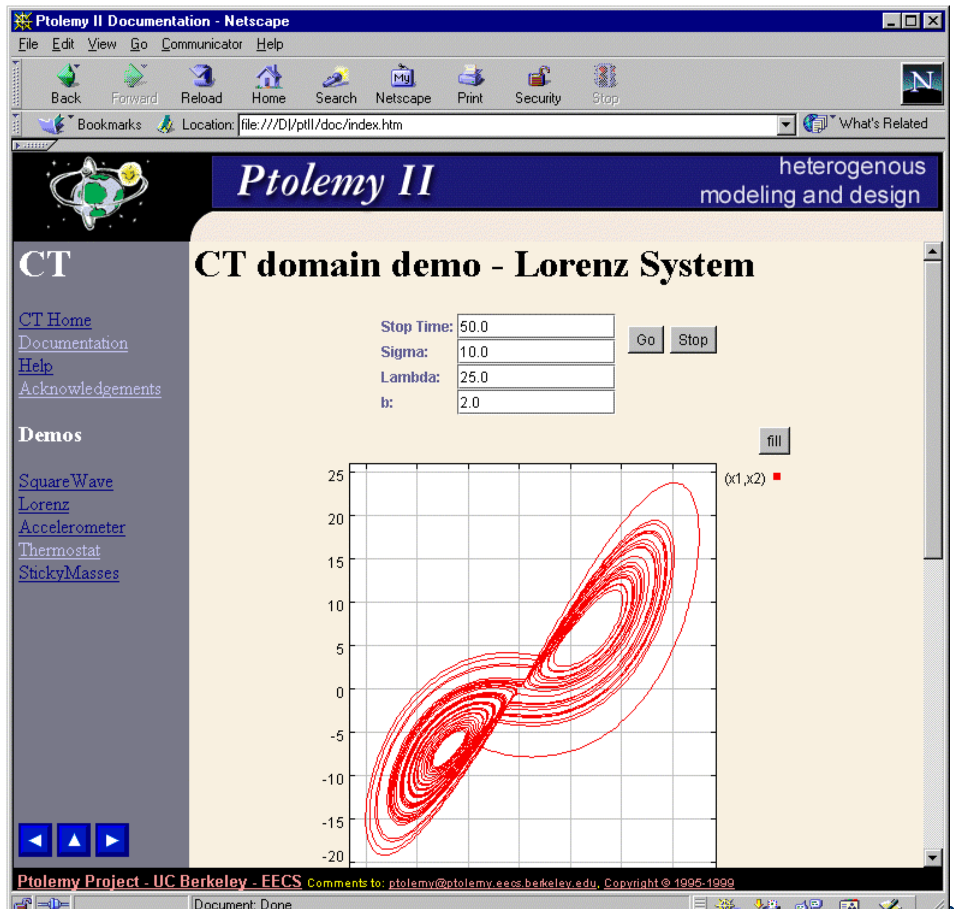
- actors define relations between functions of time (ODEs or algebraic equations)
- a behavior is a set of signals satisfying these relations

Examples:

- Spice,
- HP ADS,
- Simulink,
- Saber,
- Matrix X,
- ...

1. Continuous Time in Ptolemy II

The continuous time (CT) domain in Ptolemy II models components interacting by continuous-time signals. A variable-step size, Runge-Kutta ODE solver is used, augmented with discrete-event management (via modeling of Dirac delta functions).



1. CT: Strengths and Weaknesses

Strengths:

- Accurate model for many physical systems
- Determinate under simple conditions
- Established and mature (approximate) simulation techniques

Weaknesses:

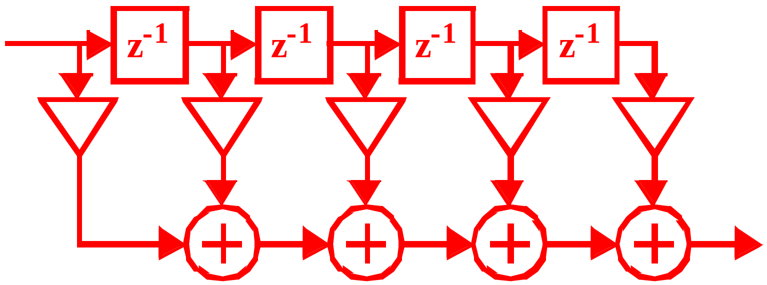
- Covers a narrow application domain
- Tightly bound to an implementation
- Relatively expensive to simulate
- Difficult to implement in software

2. Discrete Time

Semantics:

- blocks are relations between functions of discrete time (difference equations)

- a behavior is a set of signals satisfying these relations



Examples:
- System C
- HP Ptolemy,
- SystemView,
- ...

2. DT: Strengths and Weaknesses

Strengths:

- Useful model for embedded DSP
- Determinate under simple conditions
- Easy simulation (cycle-based)
- Easy implementation (circuits or software)

Weaknesses:

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

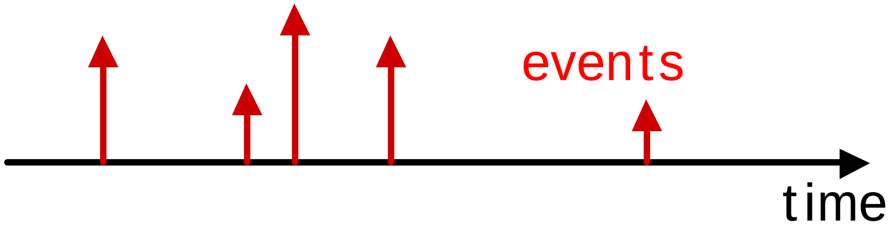
3. Discrete Events

Semantics:

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

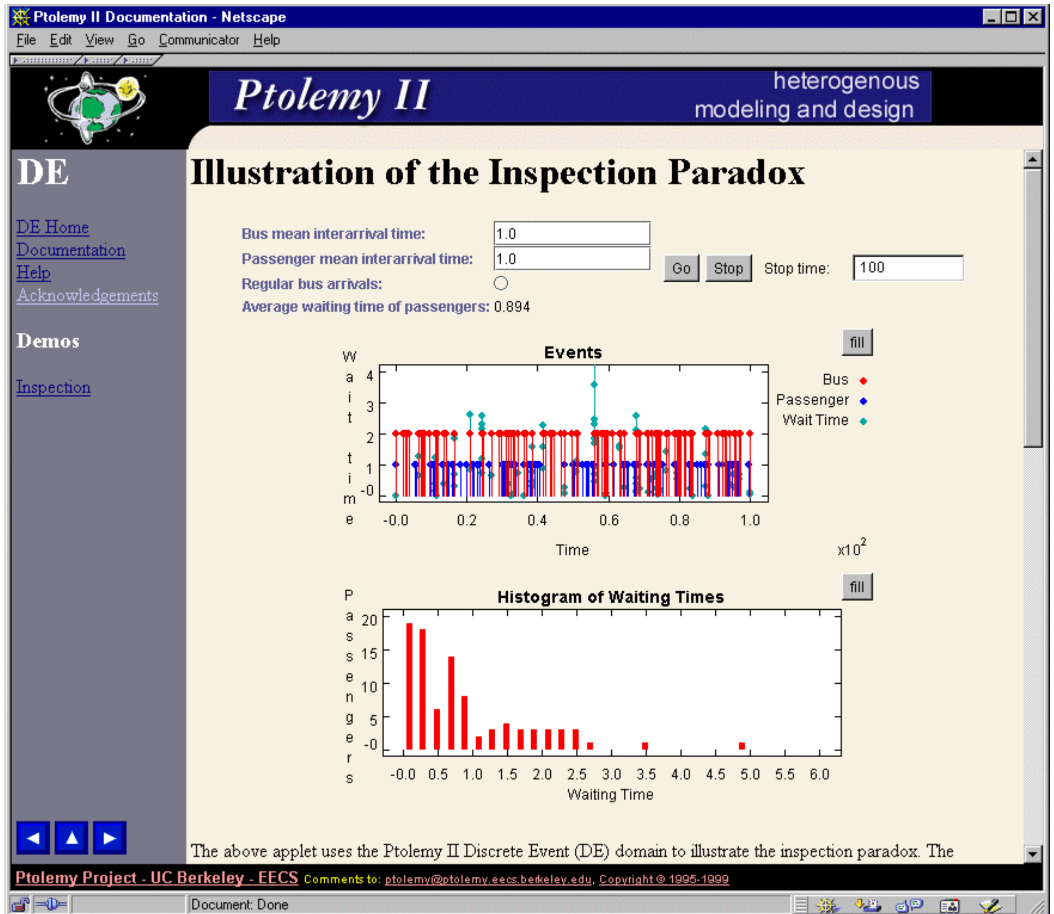
Examples:

- SES Workbench,
- Bones,
- VHDL
- Verilog
- ...



3. Discrete-Events in Ptolemy II

The discrete-event (DE) domain in Ptolemy II models components interacting by discrete events placed in time. A calendar queue scheduler is used for efficient event management, and simultaneous events are handled systematically and deterministically.



3. DE: Strengths and Weaknesses

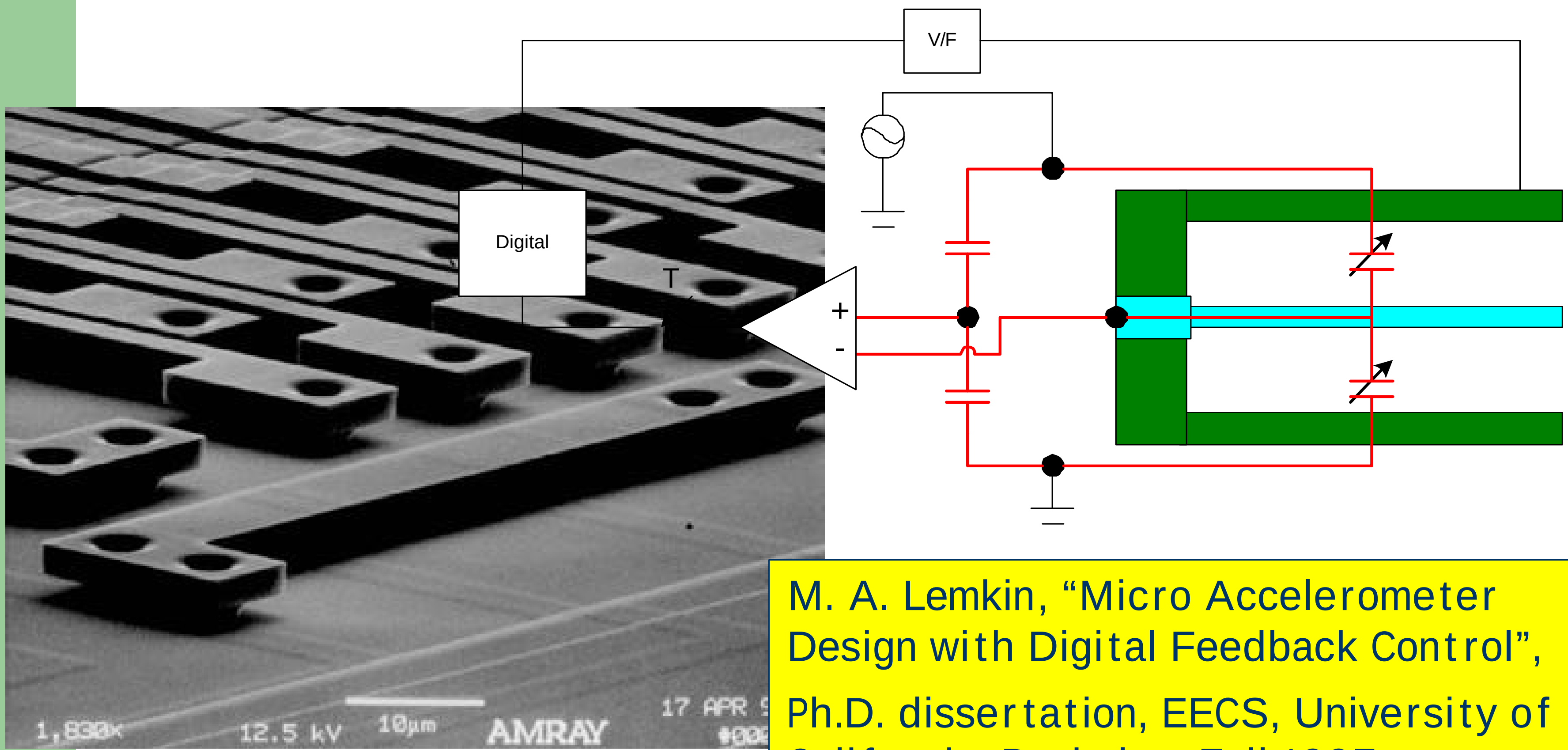
Strengths:

- Natural for asynchronous digital hardware
- Global synchronization
- Determinate under simple conditions
- Simulatable under simple conditions

Weaknesses:

- Expensive to implement in software
- May over-specify and/or over-model systems

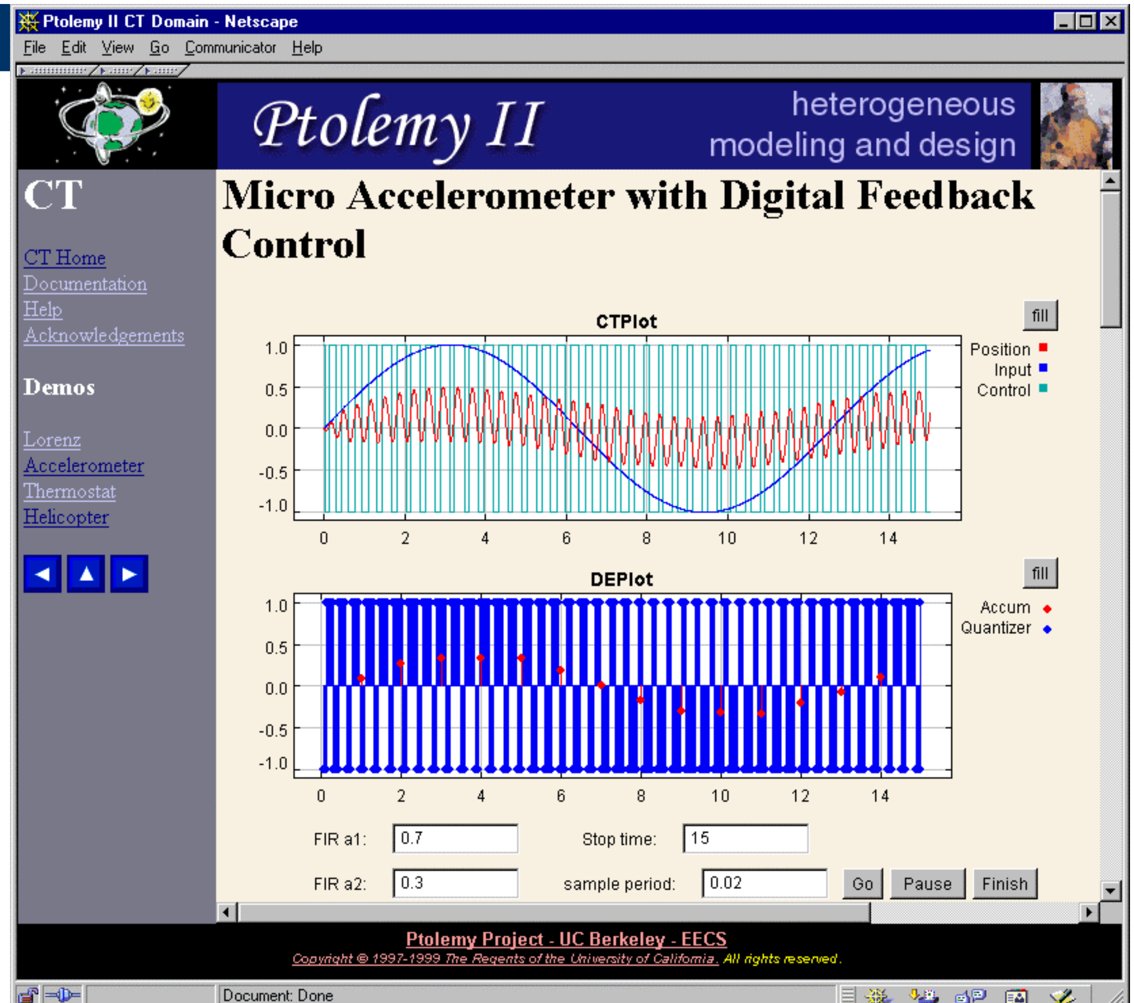
Mixing Domains
Example: MEMS Accelerometer



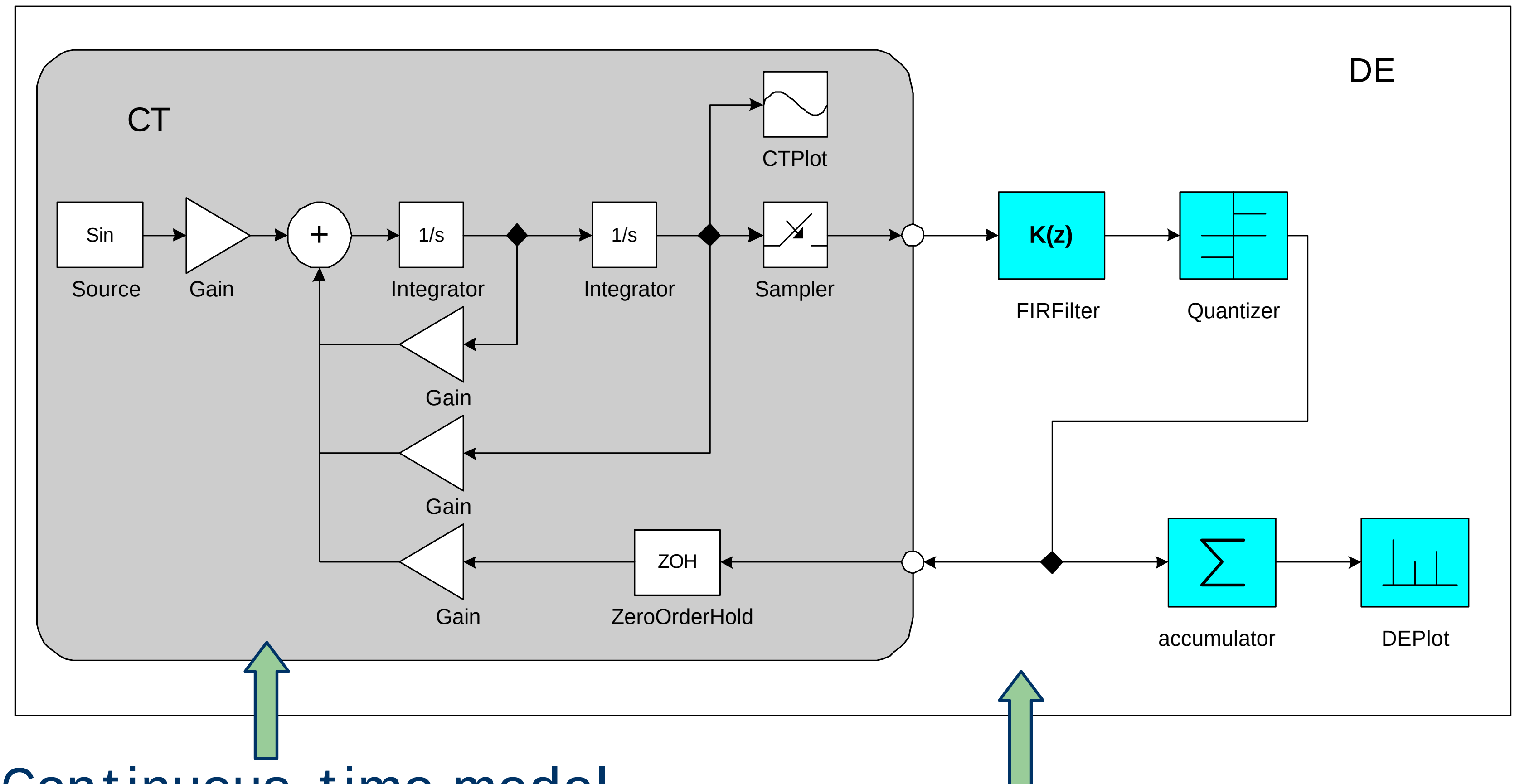
M. A. Lemkin, “Micro Accelerometer Design with Digital Feedback Control”, Ph.D. dissertation, EECS, University of California, Berkeley, Fall 1997

Accelerometer Applet

This model mixes two Ptolemy II domains, DE (discrete events) and CT (continuous time).



Hierarchical Heterogeneous Models

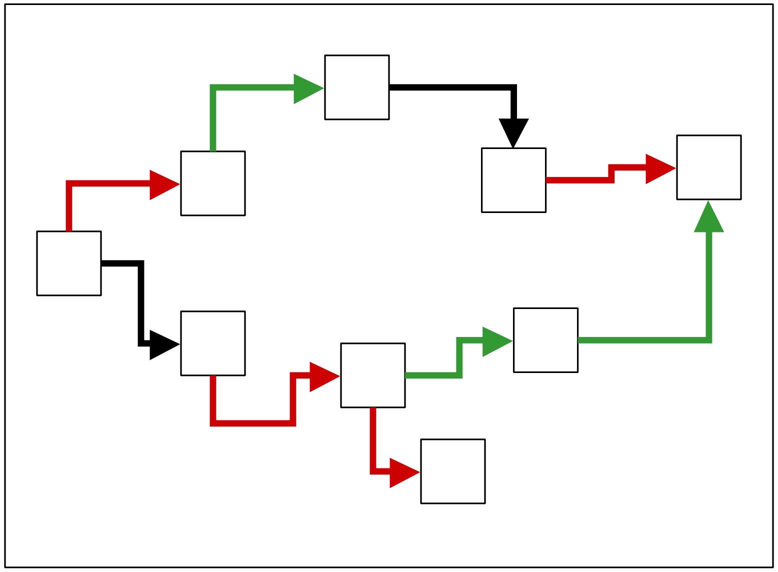


Continuous-time model

Discrete-event model

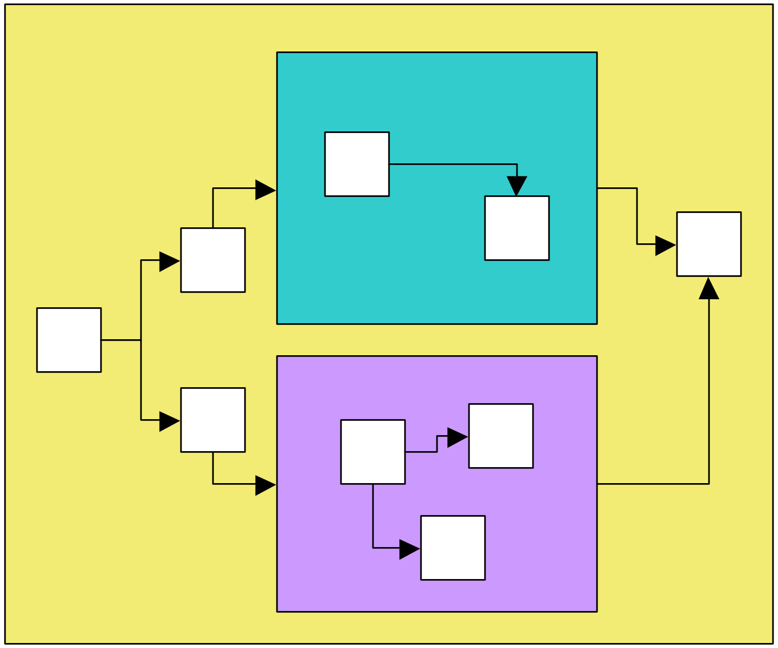
Hierarchical Heterogeneity vs. Amorphous Heterogeneity

Amorphous



Color is a communication protocol only, which interacts in unpredictable ways with the flow of control.

Hierarchical

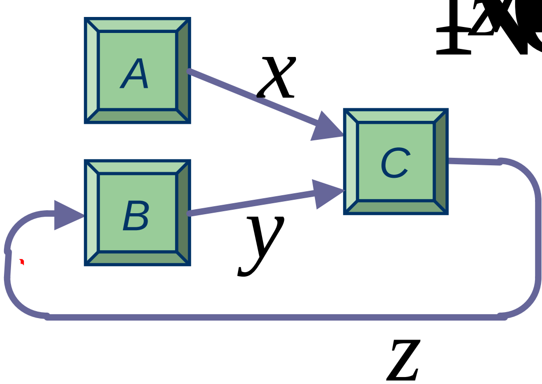


Color is a domain, which defines both the flow of control and interaction protocols.

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:

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



Examples:
- Esterel,
- Lustre,
- Signal,
- Argos,
- ...

4. SR: Strengths and Weaknesses

Strengths:

- Good match for control-intensive systems
- Tightly synchronized
- Determinate in most cases
- Maps well to hardware and software

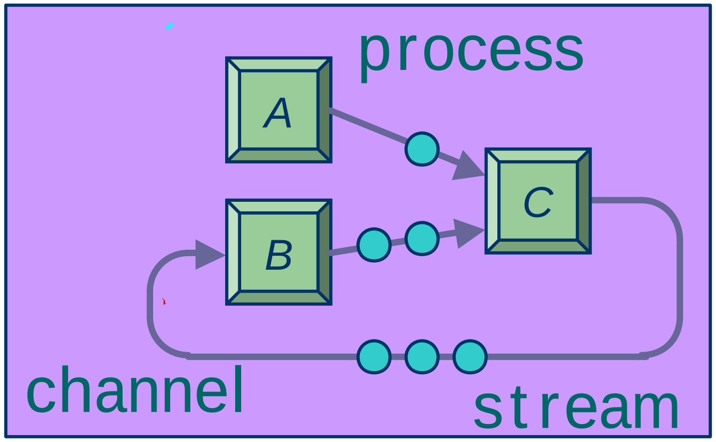
Weaknesses:

- Computation-intensive systems are overspecified
- Modularity is compromised
- Causality loops are possible
- Causality loops are hard to detect

5. Process Networks

- Processes are prefix-monotonic functions mapping sequences into sequences.
- One implementation uses blocking reads, non-blocking writes, and unbounded FIFO channels.
- Dataflow special cases have strong formal properties.

Examples:
- SDL,
- Unix pipes,
- ...



5. Strengths and Weaknesses

Strengths:

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

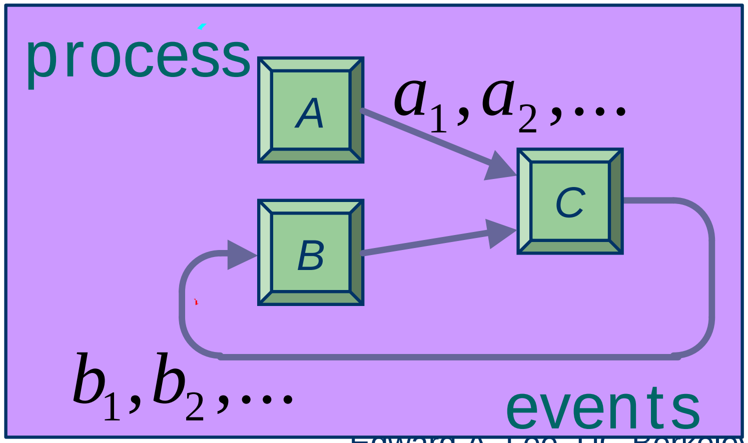
Weaknesses:

- Control-intensive systems are hard to specify
- Bounded resources are undecidable

6. Rendezvous Models

- Events represent rendezvous of a sender and a receiver. Communication is unbuffered and instantaneous.
- Often implicitly assumed with “process algebra” or even “concurrent.”

Examples:
- CSP,
- CCS,
- Occam,
- Lotos,
- ...



6. Strengths and Weaknesses

Strengths:

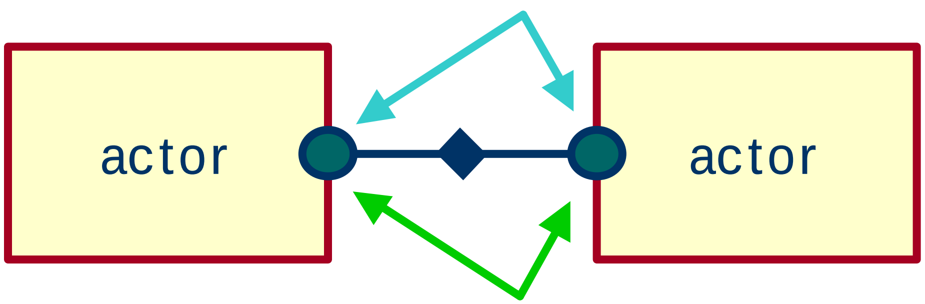
- Models resource sharing well
- Partial-order synchronization (distributable)
- Supports naturally nondeterminate interactions

Weaknesses:

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

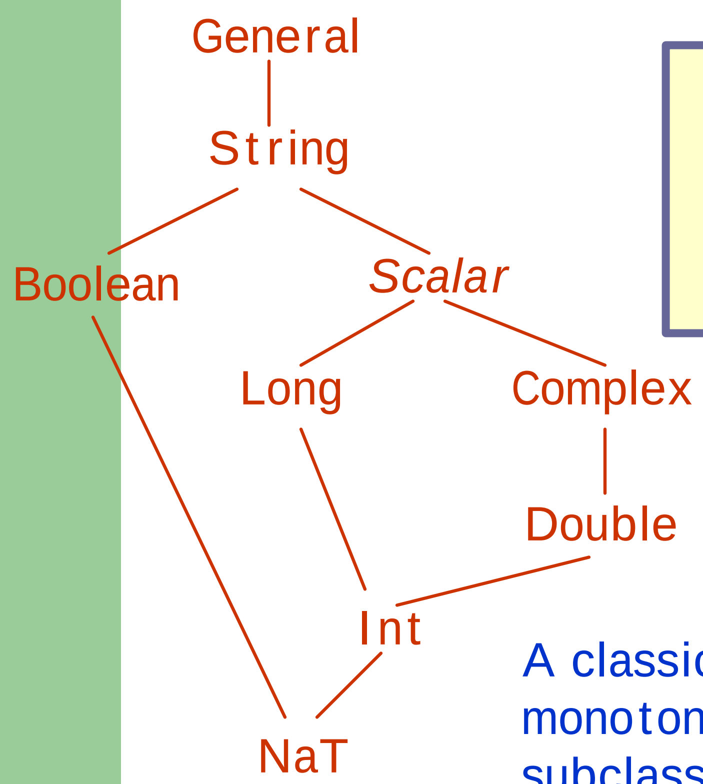
Making Sense of the Options: Component Interfaces

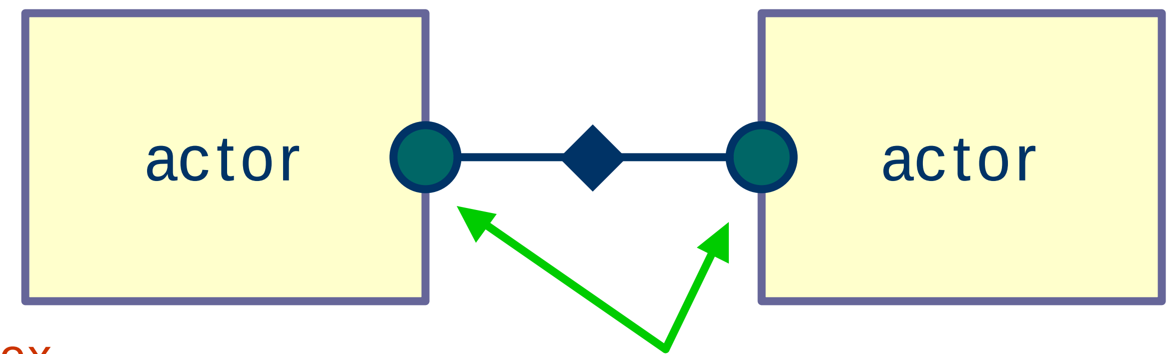
represent data types for messages exchanged on ports. ⟵ **classical type system**



represent interaction semantics as types on these ports. ⟵ **system-level types**

Approach – System-Level Types

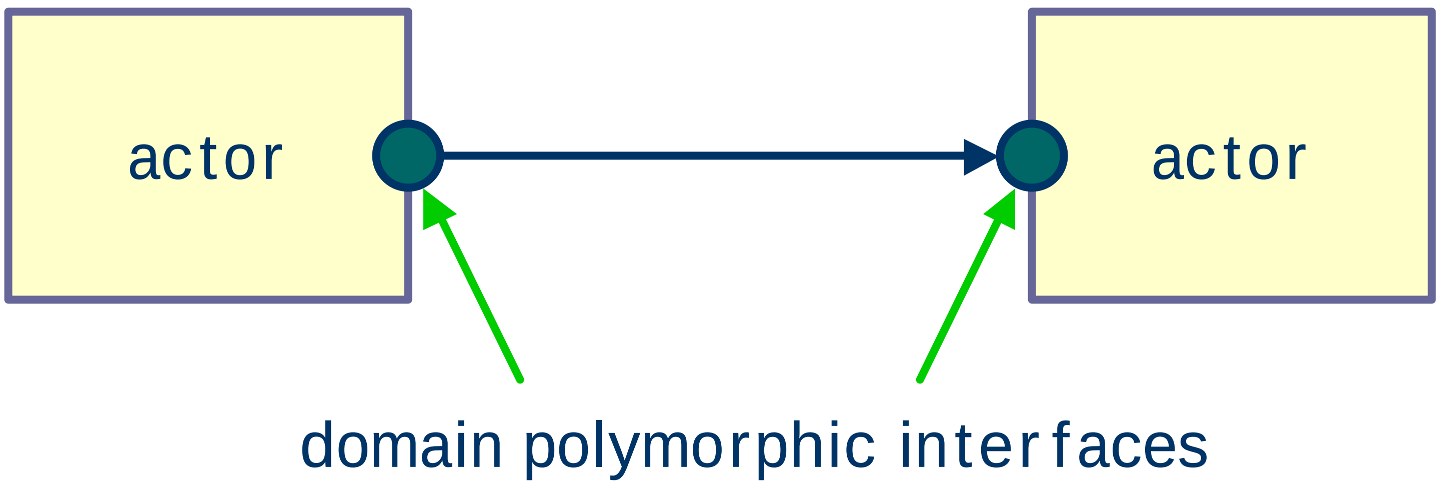




represent interaction semantics as types on these ports.

A classical type system is based on fixed-points of monotonic functions on a lattice where order represents subclassing. Our system-level types are use the simulation relation between automata to provide an order relation.

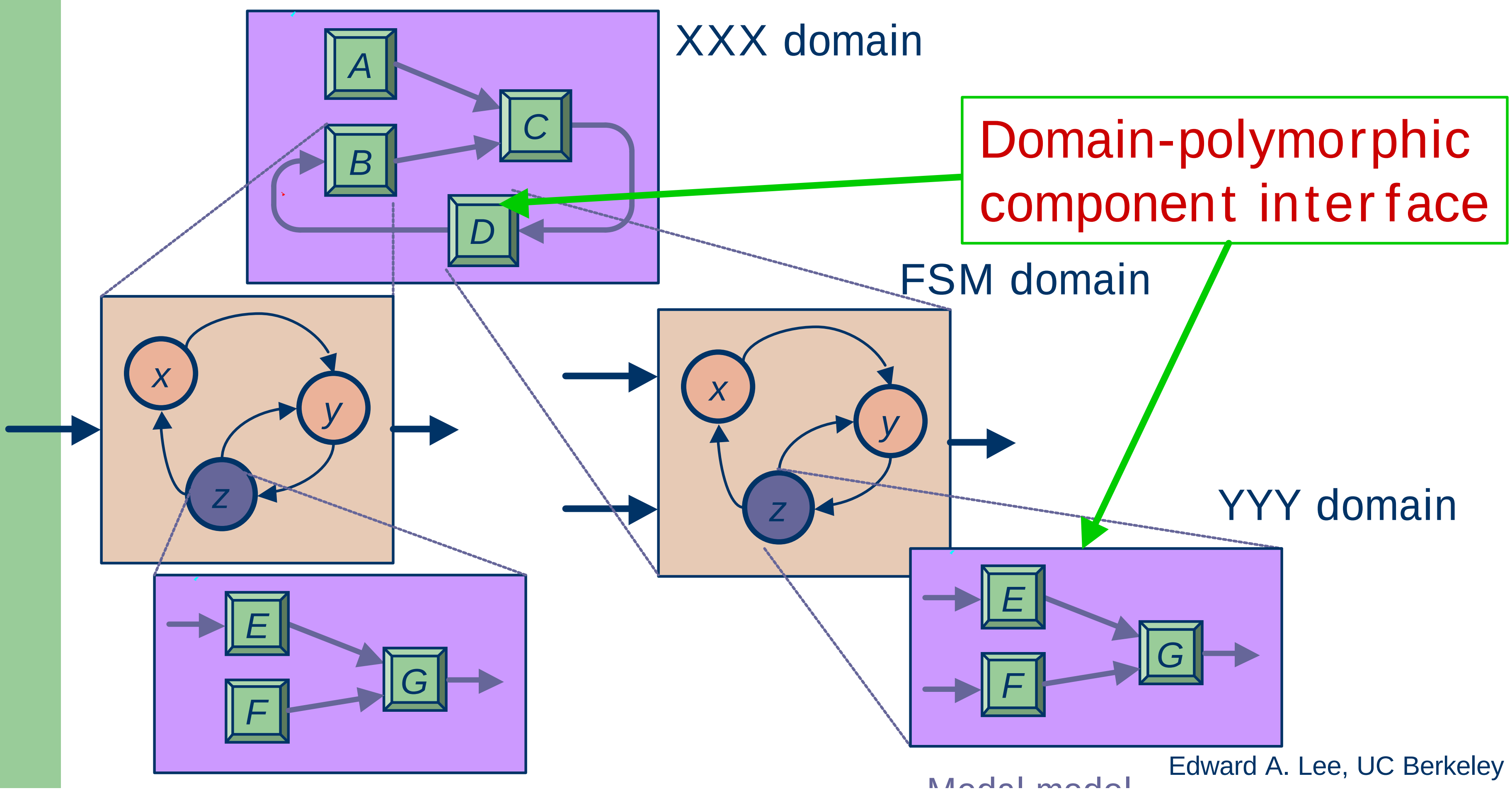
Our Hope –
Domain Polymorphic Interfaces



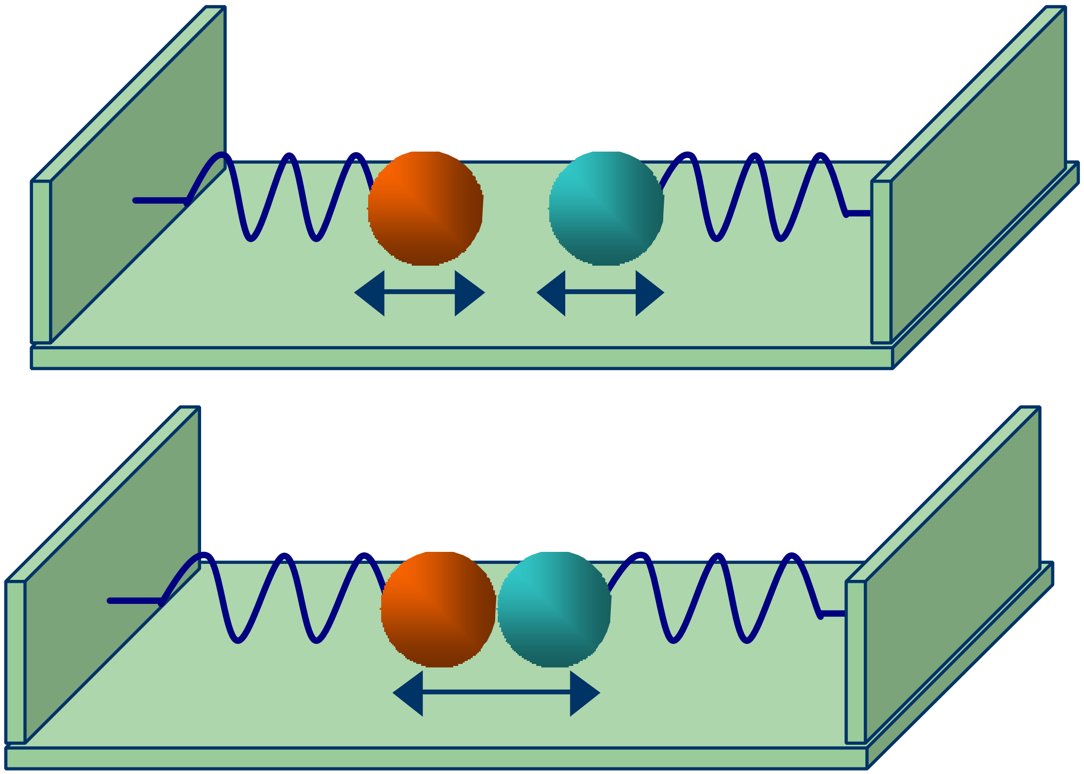
Benefits of System-Level Types

- Clarify assumptions of components
- Understandable component composition
- Data polymorphic component libraries
- Domain polymorphic component libraries
- More efficient synthesis (?)

*Charts: Exploiting Domain Polymorphism



Special Case: Hybrid Systems

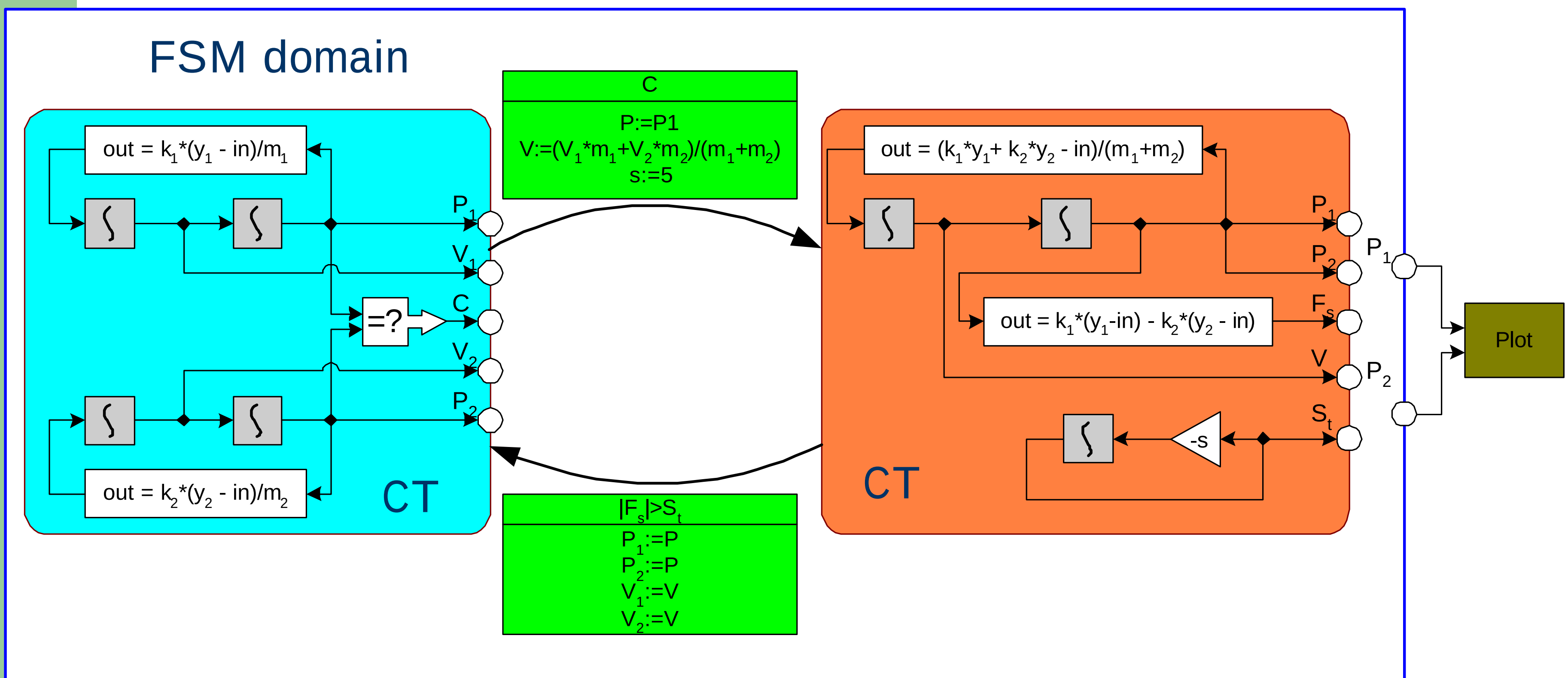


Example: Two point masses on springs on a frictionless table. They collide and stick together.

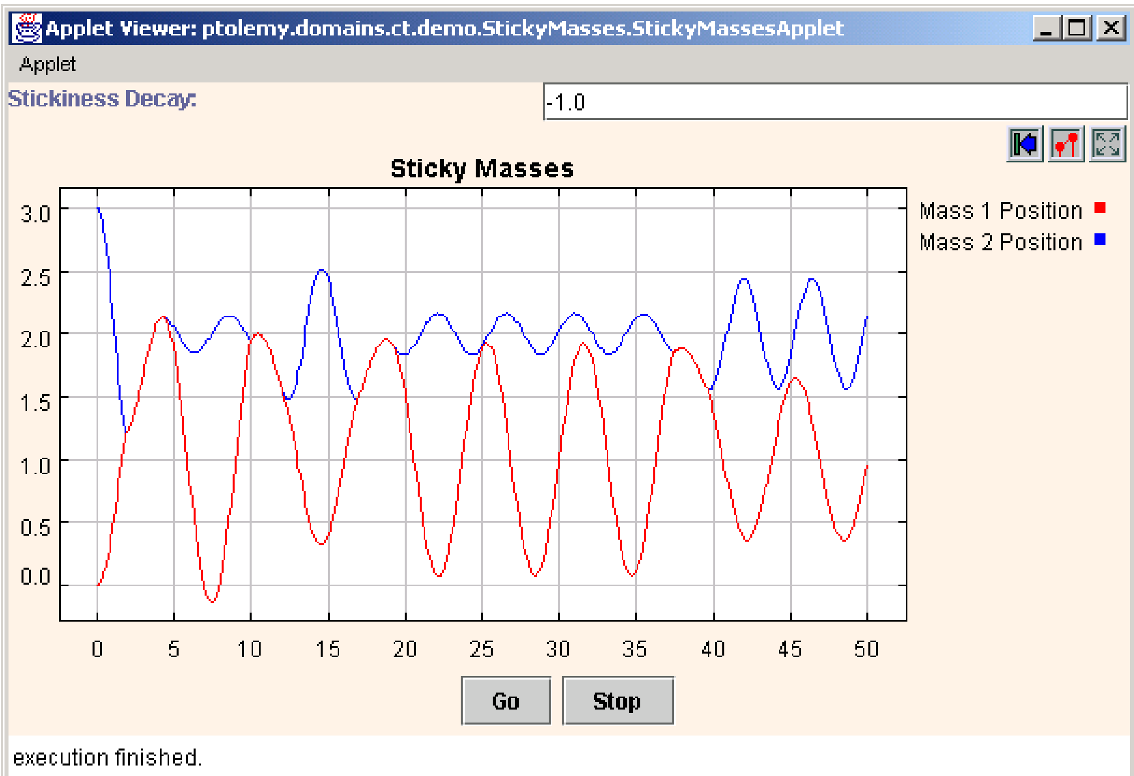
The stickiness is exponentially decaying with respect to time.

Hybrid System: Block Diagram

CT domain



Ptolemy II Execution



Because of domain polymorphism, moreover, Ptolemy II can combine FSMs hierarchically with any other domain, delivering models like statecharts (with SR) and SDL (with process networks) and many other modal modeling techniques.

Summary

- There is a rich set of component interaction models
  - models of computation
  - domains
- Hierarchical heterogeneity
  - yields more understandable designs than amorphous heterogeneity
- System-level types
  - Define the dynamics of a component interface
- Domain polymorphism
  - More flexible component libraries
  - A very powerful approach to heterogeneous modeling

Acknowledgements

The entire Ptolemy project team contributed immensely to this work, but particularly

- John Davis
- Chamberlain Fong
- Tom Henzinger
- Christopher Hylands
- Jie Liu
- Xiaojun Liu
- Steve Neuendorffer
- Sonia Sachs
- Neil Smyth
- Kees Vissers
- Yuhong Xiong