Building Unreliable Systems out of Reliable Components: *The Real Time Story*

Edward A. Lee

Professor, Chair of EE, and Associate Chair of EECS

CHESS: Center for Hybrid and Embedded Software Systems

UC Berkeley

Shannon Lecture

IEEE-Computer Society, Silicon Valley Chapter

Stanford University, November 17, 2005.

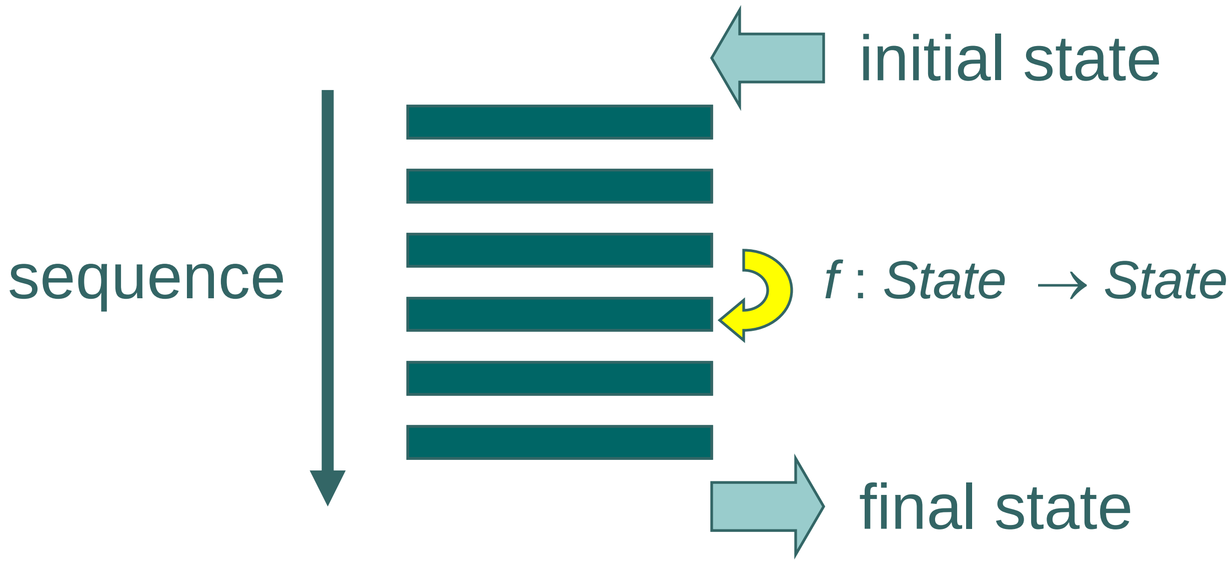
Electronics Technology Delivers Timeliness

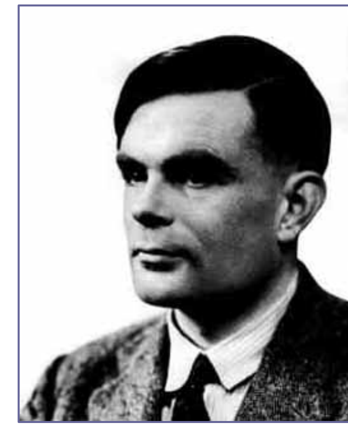
… and the overlaying abstractions discard it.

Computation in the 20th Century

$$f : \{0,1\}^* \to \{0,1\}^*$$

Computation in the 20th Century





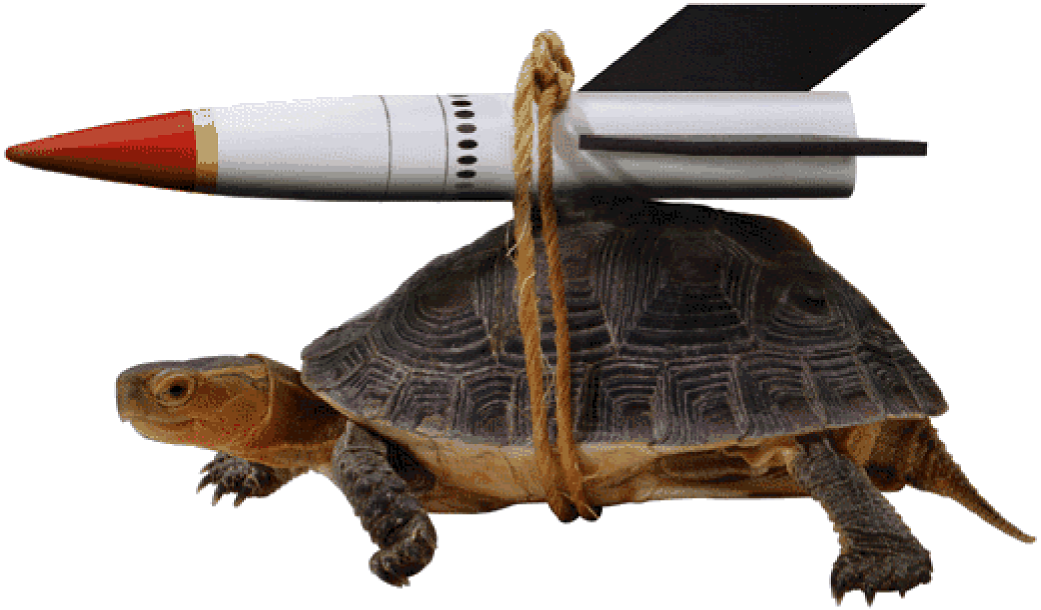
Alan Turing

- Time is irrelevant
- All actions are ordered
- Nontermination is a defect
- Concurrency is an illusion

Exploiting the 20th Century Abstraction

- Programming languages
- Debuggers
- Virtual memory
- Caches
- Dynamic dispatch
- Speculative execution
- Power management (voltage scaling)
- Memory management (garbage collection)
- Just-in-time (JIT) compilation
- Multitasking (threads and processes)
- Networking (TCP)
- Theory (complexity)

What about timeliness?



Moore's
law has
saved us!

In Core Software Abstractions: Real-Time is Not

- Time is not in the semantics of programs.
  - Have to step outside the semantics to specify timing.
- Timing is a consequence of implementation not a property of design.
  - Measured on the bench
  - For a particular realization
- Resulting systems are brittle.
  - Small changes have big consequences
  - Ports to new platforms require redesign

The Myth of WCET
Worst-Case Execution Time

- True WCET can be thousands of times bigger than actual execution time.
- In many implementations, true WCET is not a useful number.
- Dubious WCET is what is actually used.
- Correctness of even safety-critical systems depends on WCET being correct.

What is Done in Practice

- Real-time systems are boxes, not software services.
- Critical real-time systems use idiosyncratic, non-mainstream processors (like DSPs).
- Designs are bench tested, then encased.

APOT

The question: What would have to change to achieve *absolutely, positively on time* (APOT)?

The answer: *nearly everything.*

What to do?

- Put time into programming languages
  - *Promising start*: Simulink, Giotto, Discrete-event models
- Rethink the OS/programming language split
  - *Promising start*: TinyOS/nesC
- Rethink the hardware/software split
  - *Promising start*: FPGAs with programmable cores
- Memory hierarchy with predictability
  - *Promising start*: Scratchpad memories vs. caches
- Memory management with predictability
  - *Promising start*: Bounded pause time garbage collection
- Predictable, controllable deep pipelines
  - *Promising start*: Pipeline interleaving + stream-oriented languages
- Predictable, controllable, understandable concurrency
  - *Promising start*: Synchronous languages, SCADE
- Networks with timing
  - *Promising start*: Time triggered architectures, time synchronization
- Computational dynamical systems theory
  - *Promising start*: Hybrid systems, schedulability analysis

Recall: Computation in the 20th Century

$$f : \{0,1\}^* \rightarrow \{0,1\}^*$$

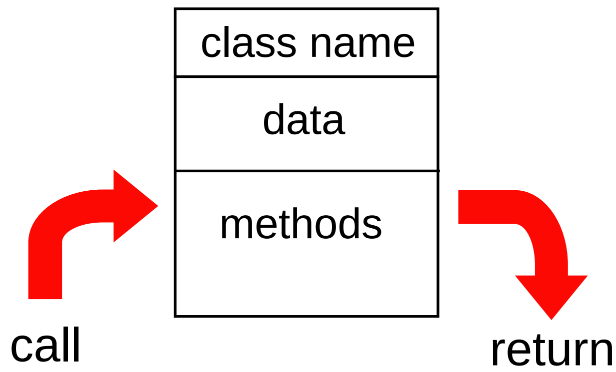
Computation in the 21st Century

$$f : [T \to \{0,1\}^*]^P \to [T \to \{0,1\}^*]^P$$

…where T is a set representing time, precedence ordering, causality, synchronization, etc.

A Consequence: Component Abstractions Need to Change

Object-oriented:

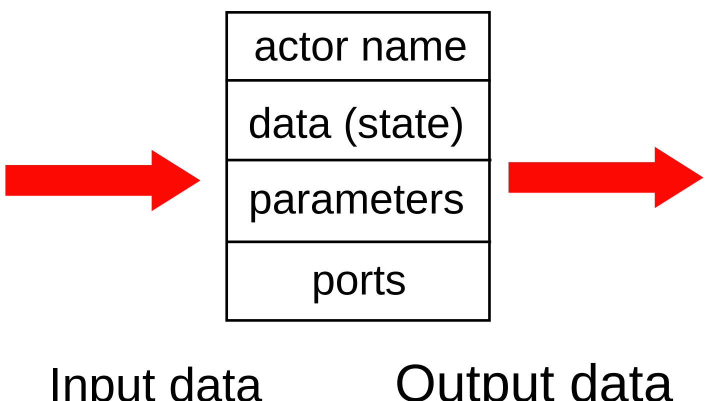


What flows through an object is sequential control

Stuff happens to objects

Actor oriented:

Actors make things happen

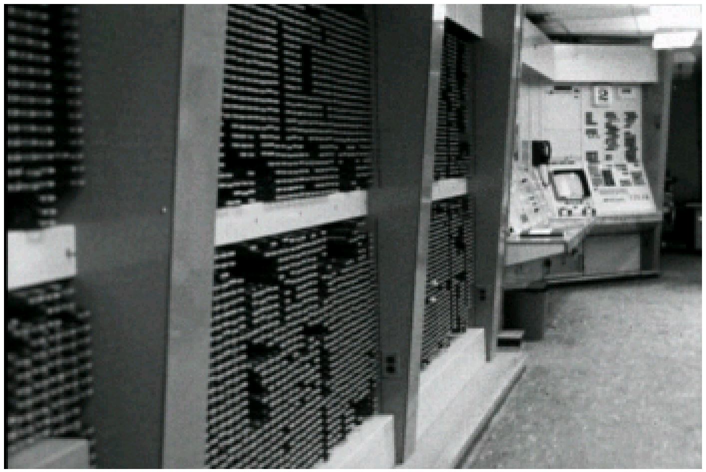


What flows through an object is streams of data

The First (?) Actor-Oriented Platform

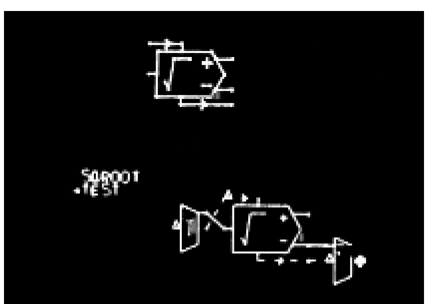
The On-Line Graphical Specification of Computer Procedures

W. R. Sutherland, Ph.D. Thesis, MIT, 1966



MIT Lincoln Labs TX-2 Computer

Bert Sutherland with a light pen



Bert Sutherland used the first acknowledged object-oriented framework (Sketchpad, created by his brother, Ivan Sutherland) to create the first actor-oriented programming framework.

Partially constructed actor-oriented model with a class definition (top) and instance (below).

Your Speaker in 1966

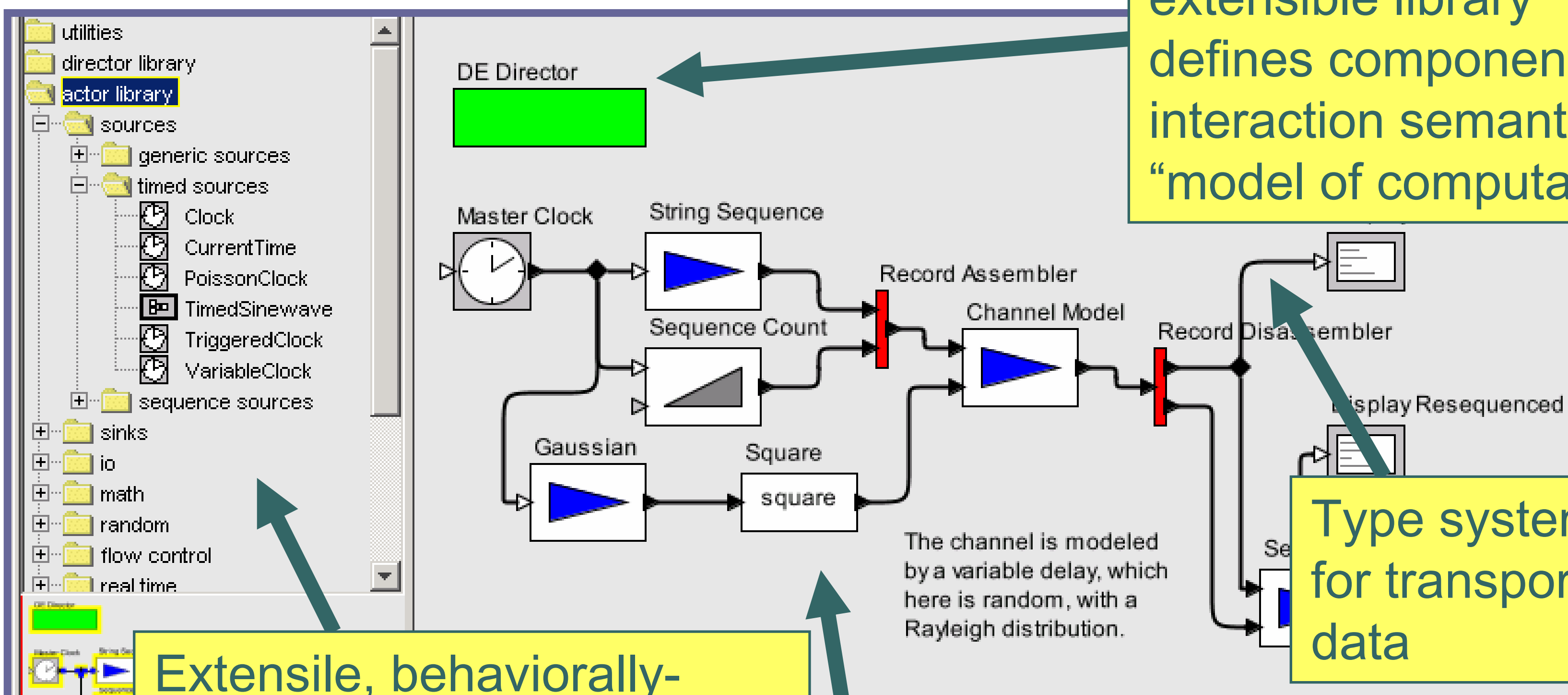
Modern Examples of Actor-Oriented Platforms

- Simulink (The MathWorks)
- LabVIEW (National Instruments)
- Modelica (Linkoping)
- OPNET (Opnet Technologies)
- Giotto and xGiotto (UC Berkeley)
- Polis & Metropolis (UC Berkeley)
- Gabriel, Ptolemy, and Ptolemy II (UC Berkeley)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Easy5 (Boeing)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- …

Ptolemy II: Our Laboratory for Actor-Oriented Models of Computation

Concurrency management supporting dynamic model structure.

Director from an extensible library defines component interaction semantics or "model of computation."



Type system for transported data

Extensile, behaviorally-polymorphic component library.

Visual editor supporting an abstract syntax

Models of Computation Implemented in Ptolemy II

- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- DDF – Dynamic dataflow
- DPN – distributed process networks
- DT – discrete time (cycle driven)
- FSM – finite state machines
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Most of these are actor oriented.

A New Foundation for Computation

$$f : [T \to \{0,1\}^*]^P \to [T \to \{0,1\}^*]^P$$

…where T is a set representing time, precedence ordering, causality, synchronization, etc.

A Start on a 21st Century Theory of Computation: The Tagged Signal Model

[Lee & Sangiovanni-Vincentelli, 1998]

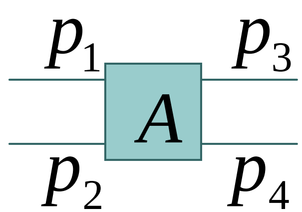
- A set of *values* V and a set of *tags* T
- An *event* is $e \in T \times V$
- A *signal* s is a set of events. I.e. $s \subset T \times V$
- A *functional signal* is a (partial) function $s: T \to V$
- The set of all signals $S = 2^{T \times V}$

Related models:

- Interaction Categories [Abramsky, 1995]
- Interaction Semantics [Talcott, 1996]
- Abstract Behavioral Types [Arbab, 2005]

Actors, Ports, and Behaviors

An *actor* has a set of *ports* P



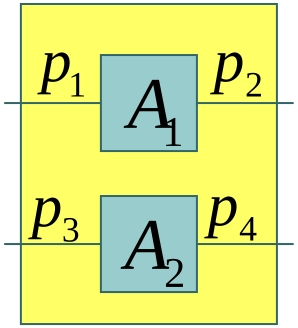
$$P_A = \{ p_1, p_2, p_3, p_4 \}$$

A behavior is a function $\sigma \colon P_A \to S$

An *actor* is a set of *behaviors* $A \subset [P_A \to S\ ] = S^{P_A}$

Actor Composition

Composition is simple intersection
(of sets of functions)



$$P_1 = \{ p_1, p_2 \}$$

$$P_2 = \{ p_3, p_4 \}$$

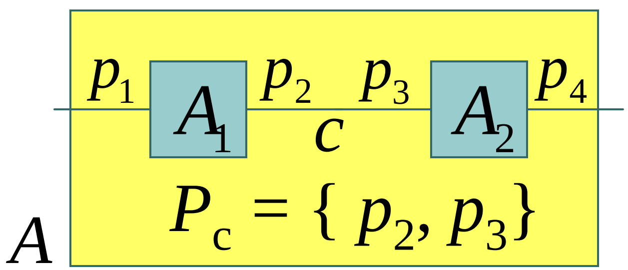
$$A = A_1 \wedge A_2 \qquad P = P_1 \cup P_2$$

$$A = A_1 \wedge A_2 = \{ \sigma \mid \sigma \downarrow_{P_1} \in A_1 \text{ and } \sigma \downarrow_{P_2} \in A_2 \} \subset [P \rightarrow S]$$

Connectors

Connectors are trivial actors.

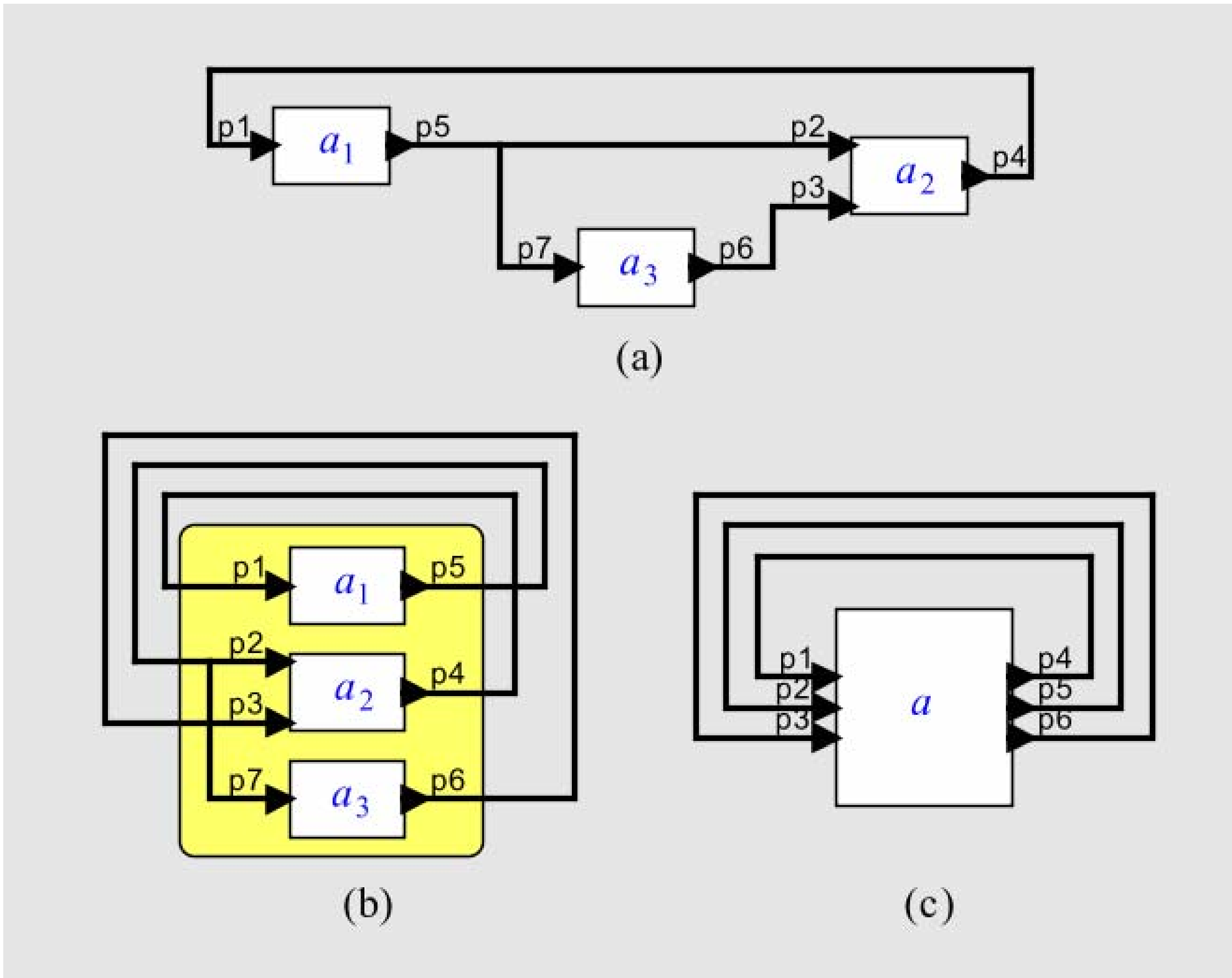
$$P_1 = \{ p_1, p_2 \} \quad P_2 = \{ p_3, p_4 \}$$



$$c \subset [P_c \to S], \ \forall \sigma \in c, \ \forall p_1, p_2 \in P_c, \ \ \sigma(p_1) = \sigma(p_2)$$

$$A = A_1 \wedge A_2 \wedge c$$

Tagged Signal Model Gives a Fixed-Point Semantics to Arbitrary Composition



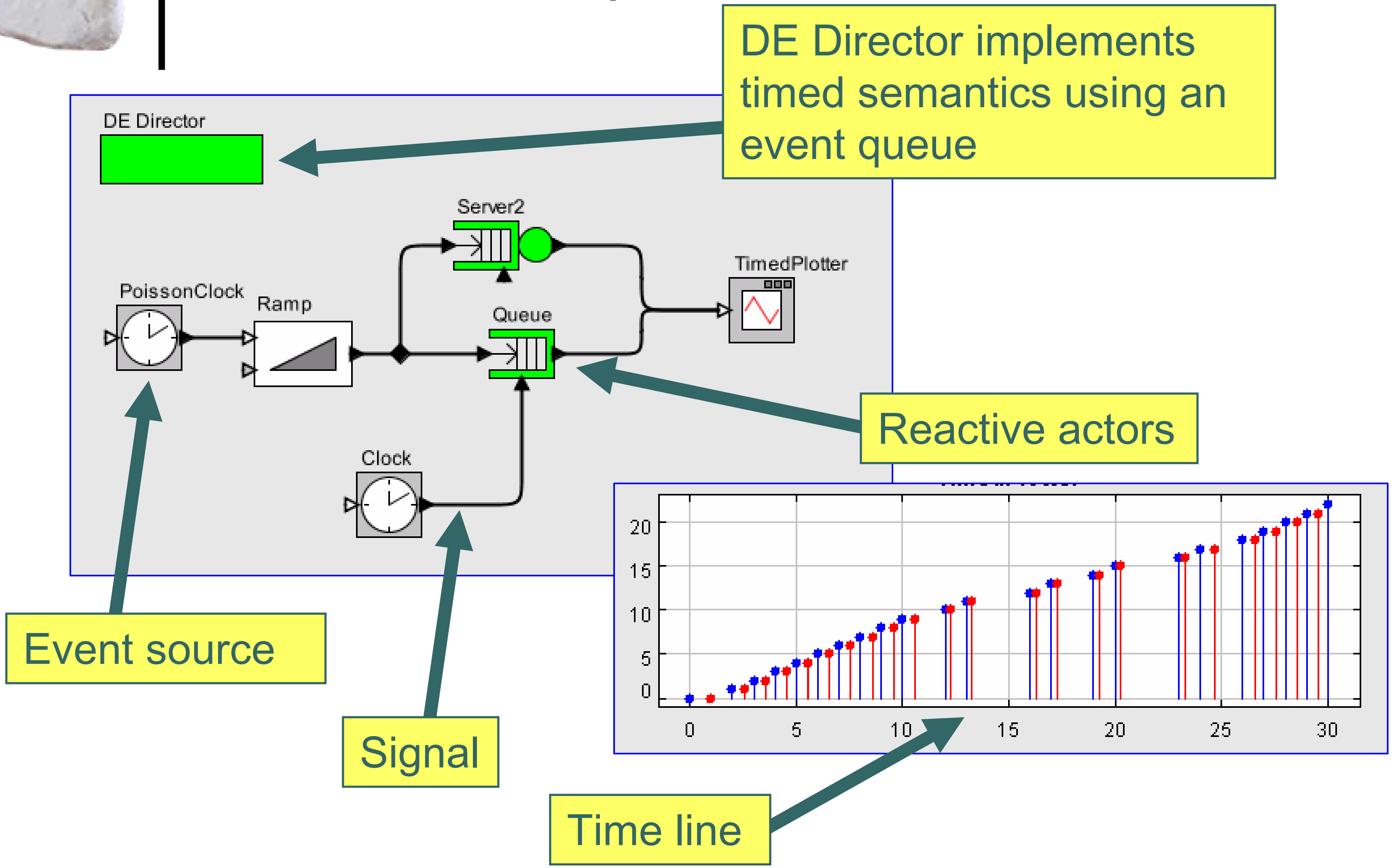
Tagged Signal Model can be used on a Wide Variety of Concurrent and Timed Models of Computation

- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDF – Dynamic dataflow
- DT – discrete time
- Giotto – synchronous periodic
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive

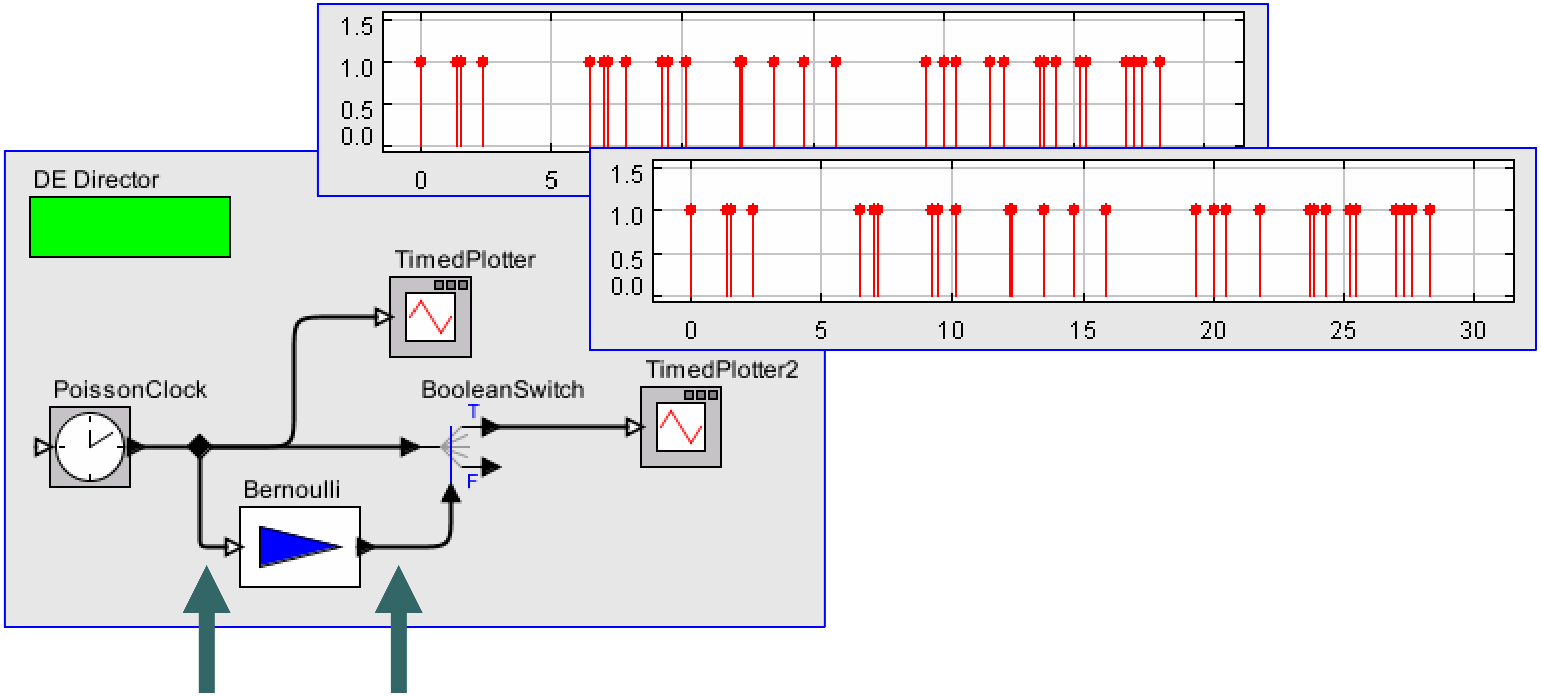
Application of this Theory of Computation: Discrete-Event Systems

- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- DDF – Dynamic dataflow
- DPN – distributed process networks
- DT – discrete time (cycle driven)
- FSM – finite state machines
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Discrete Events (DE): A Timed Concurrent Model of Computation

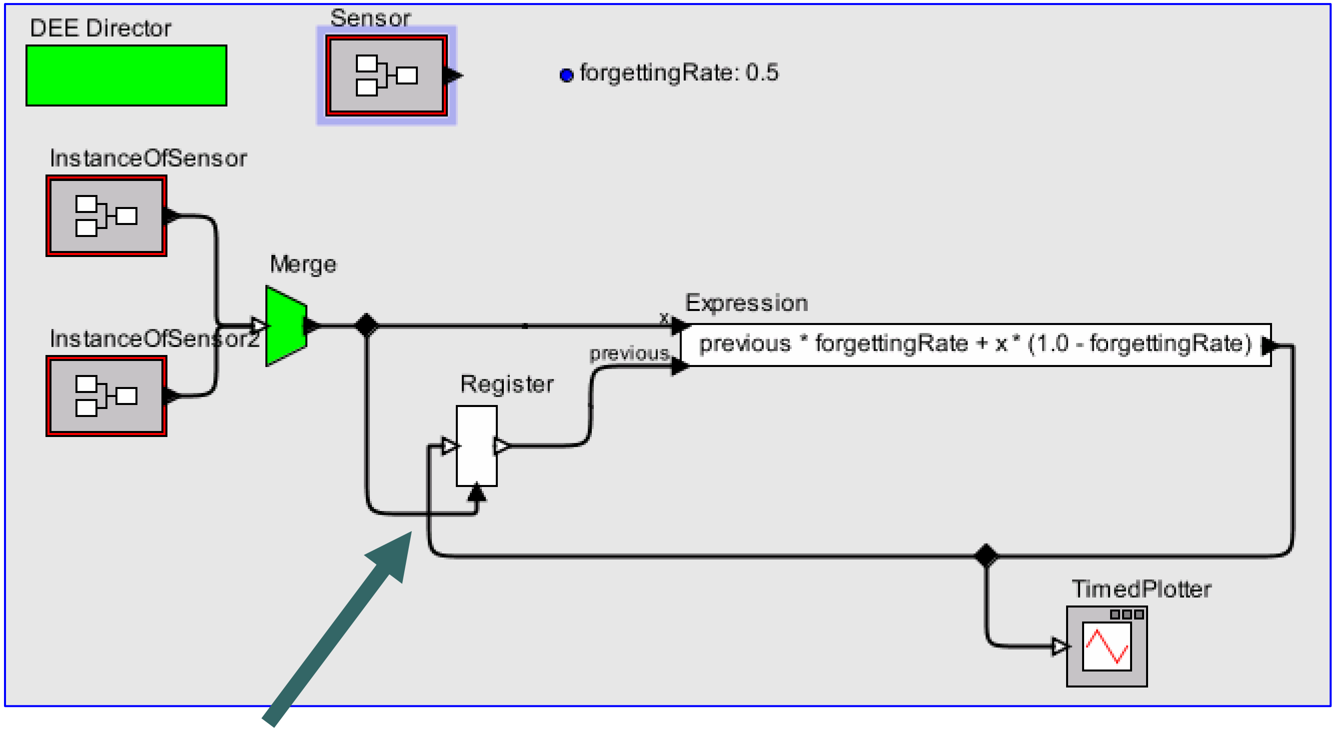


Semantics Clears Up Subtleties: Simultaneous Events



By default, an actor produces events with the same time as the input event. But in this example, we expect (and need) for the BooleanSwitch to "see" the output of the Bernoulli in the same "firing" where it sees the event from the PoissonClock. Events with identical time stamps are also ordered, and reactions to such events follow data precedence order.

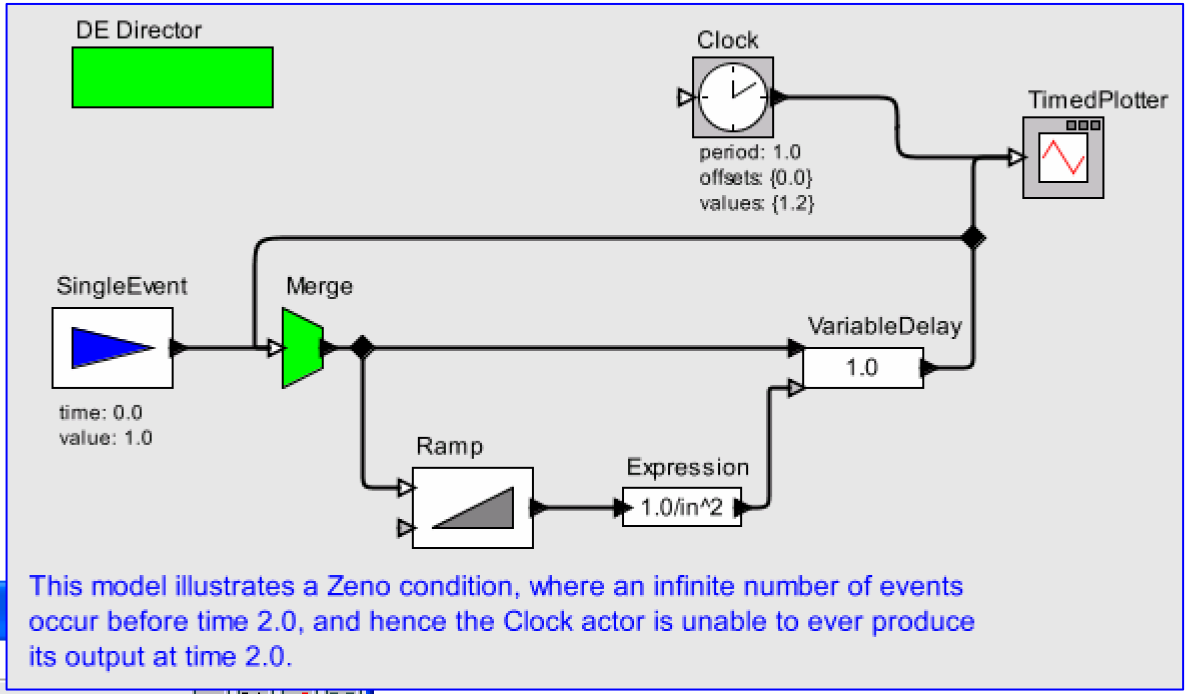
Semantics Clears Up Subtleties: Feedback

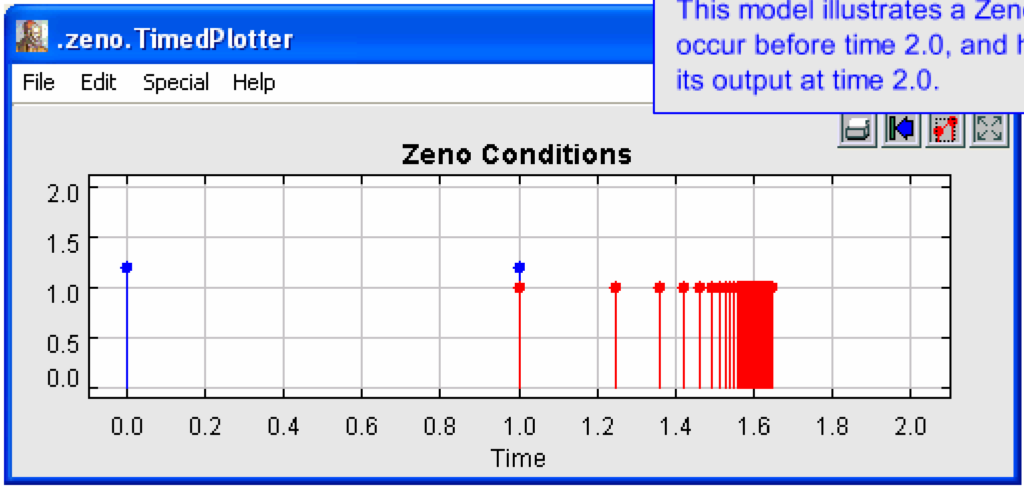


Data precedence analysis has to take into account the non-strictness of this actor (that an output can be produced despite the lack of an input).

Semantics Clears Up Subtleties: Zeno Systems

DE systems may have an infinite number of events in a finite amount of time. Carefully constructed semantics gives these systems meaning.





Example of Current Research Challenges

Use distributed discrete-event systems as a timed model of computation for embedded software in unreliable, sporadically connected networks, such as wireless sensor networks.

The most interesting possibilities are based on distributed consensus algorithms (as in Croquet, Reed, Lamport).

Research challenges include:

- Defining the semantics
- Combining the semantics heterogeneously with others. E.g.:
  - Signal processing for channel modeling
  - TinyOS for node functionality
- Creating efficient runtime environments
- Building the design environment

Current Projects in the Ptolemy Group

- Abstract semantics (Cataldo, Liu, Matsikoudis, Zheng)
  - Behavioral polymorphism
  - Actor semantics (prefire, fire, postfire)
  - Compositional directors
  - Time semantics
  - Causality interfaces
- Distributed computing (Feng, Zhao)
  - Robust distributed consensus
  - Data coherence (distributed caches)
  - Time synchronization
- Real-time software (Bandyopadhyay, Cheong, Zhou)
  - Time-based models vs. dataflow models
  - Deterministic, understandable multitasking
  - Memory hierarchy with scratchpad memory
  - Code generation
- Hybrid systems (Cataldo, Zheng)
  - Operational semantics
  - Stochastic hybrid systems
  - Aspect-oriented multi-view modeling
  - Code generation

Conclusion

The time is right to create the 21-st century theory of (embedded) computing.