Streaming Models of Computation in The Ptolemy Project

Edward A. Lee
Professor
UC Berkeley

Workshop on Streaming Systems,
Endicott House, Dedham, MA
August 22-23, 2003

Ptolemy Project Participants

Director:

- Edward A. Lee

Staff:

- Christopher Hylands
- Susan Gardner (Chess)
- Nuala Mansard
- Mary P. Stewart
- Neil E. Turner (Chess)
- Lea Turpin (Chess)

Postdocs, Etc.:

- Joern Janneck, Postdoc
- Rowland R. Johnson, Visiting Scholar
- Kees Vissers, Visiting Industrial Fellow
- Daniel Lázaro Cuadrado, Visiting Scholar

Graduate Students:

- J. Adam Cataldo
- Chris Chang
- Elaine Cheong
- Sanjeev Kohli
- Xiaojun Liu
- Eleftherios D. Matsikoudis
- Stephen Neuendorffer
- James Yeh
- Yang Zhao
- Haiyang Zheng
- Rachel Zhou


At Work in the Chess Software Lab
Chess = Center for Hybrid and Embedded Software Systems

Software Legacy of the Project

- Gabriel (1986-1991)
  - Written in Lisp
  - Aimed at signal processing
  - Synchronous dataflow (SDF) block diagrams
  - Parallel schedulers
  - Code generators for DSPs
  - Hardware/software co-simulators
- Ptolemy Classic (1990-1997)
  - Written in C++
  - Multiple models of computation
  - Hierarchical heterogeneity
  - Dataflow variants: BDF, DDF, PN
  - C/VHDL/DSP code generators
  - Optimizing SDF schedulers
  - Higher-order components
- Ptolemy II (1996-2022)
  - Written in Java
  - Domain polymorphism
  - Multithreaded
  - Network integrated and distributed
  - Modal models
  - Sophisticated type system
  - CT, HDF, CI, GR, etc.

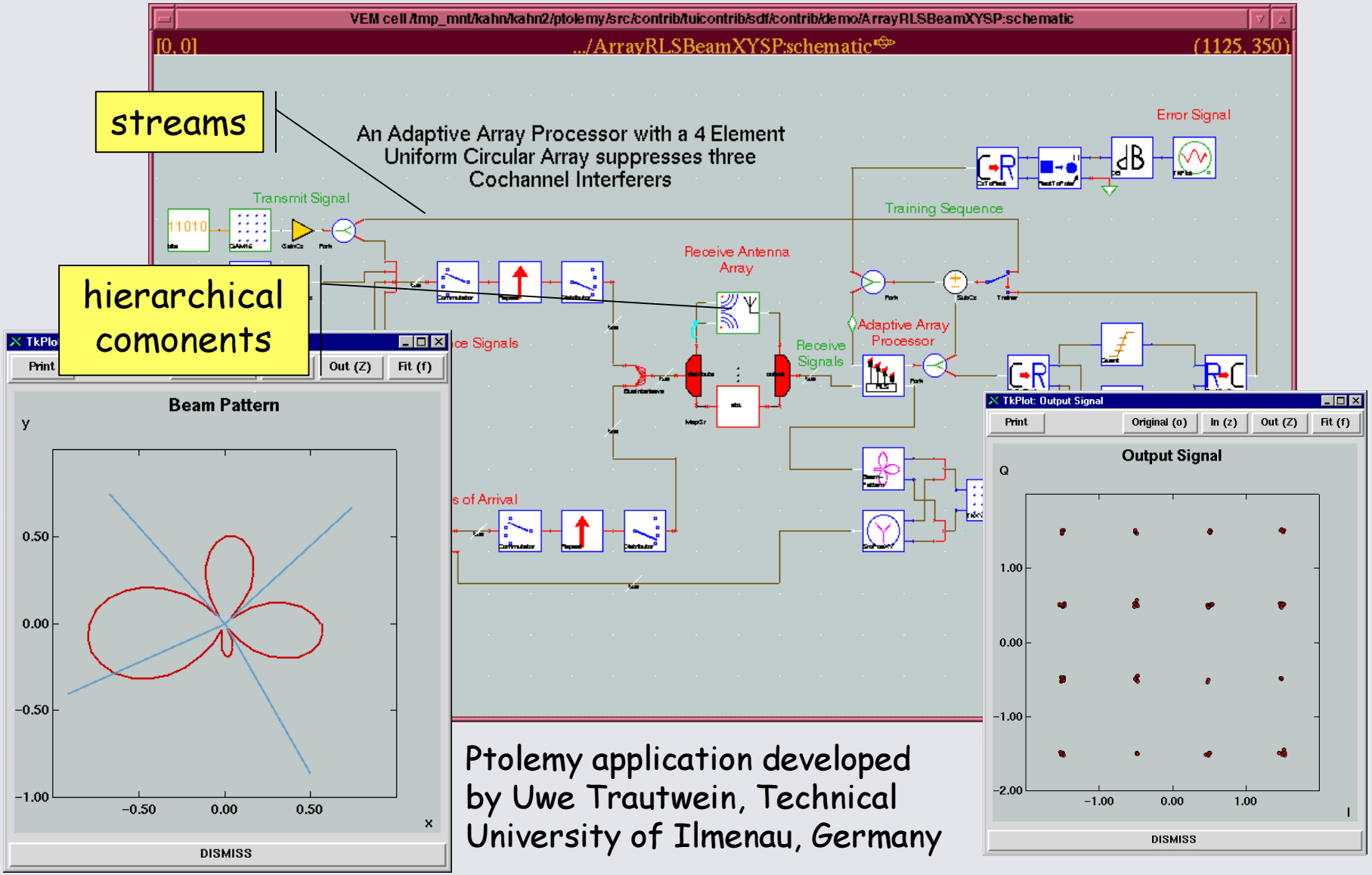
Each of these served us, first-and-foremost, as a laboratory for investigating design.

- PtPlot (1997-??)
  - Java plotting package
- Tycho (1996-1998)
  - Itcl/Tk GUI framework
- Diva (1998-2000)
  - Java GUI framework

Focus has always been on embedded software.

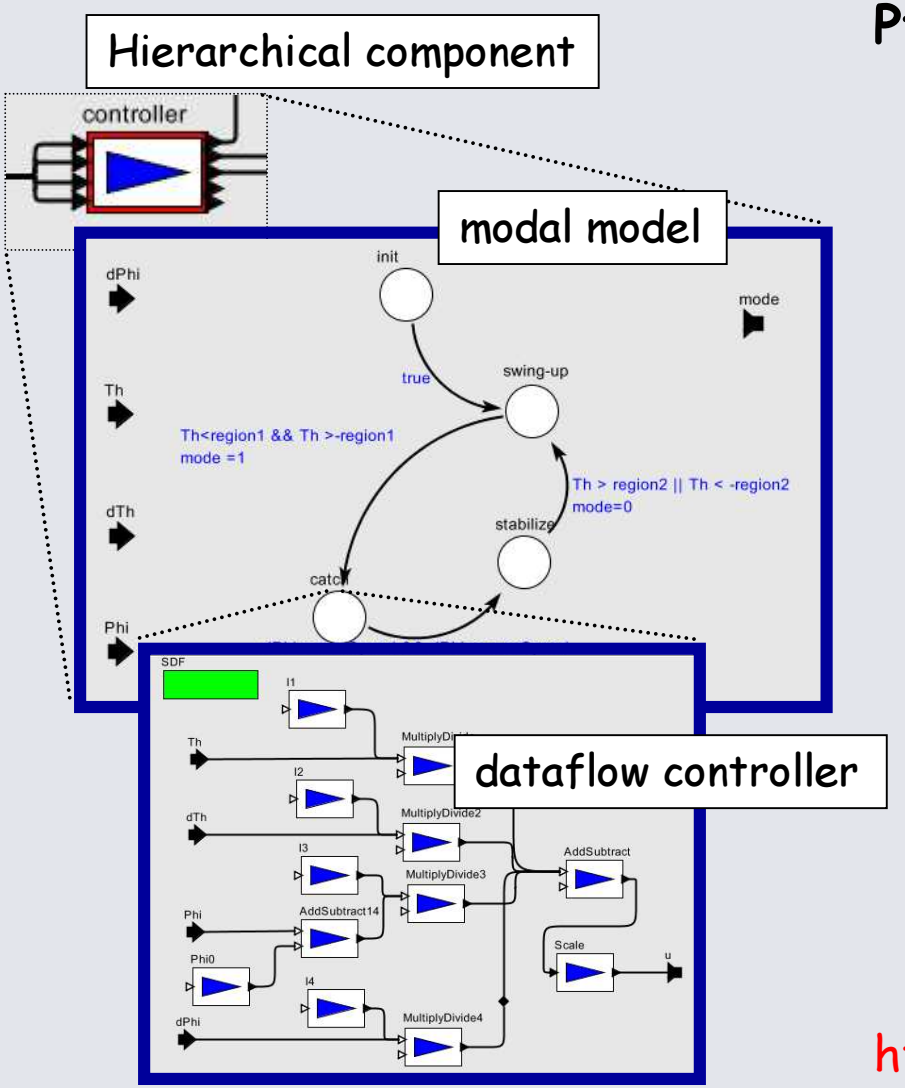
Ptolemy Classic Example From 1995

(adaptive nulling in an antenna array)



Ptolemy application developed by Uwe Trautwein, Technical University of Ilmenau, Germany

Ptolemy II



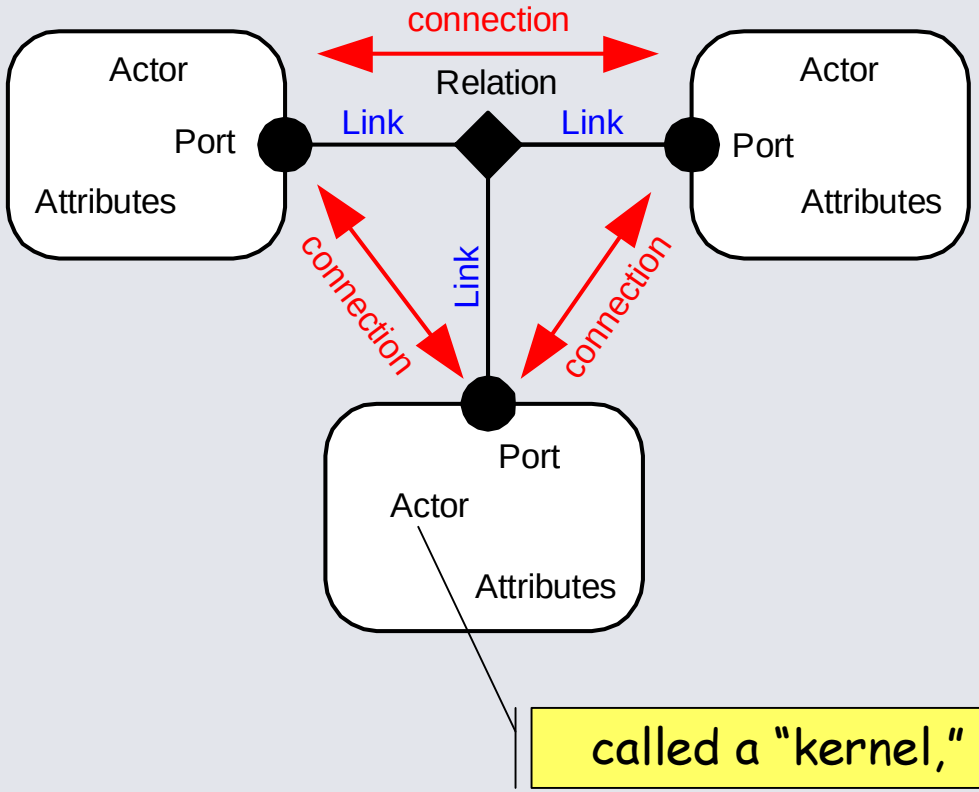
example Ptolemy II model: hybrid control system

Ptolemy II:

Our current framework for experimentation with actor-oriented design, concurrent semantics, visual syntaxes, and hierarchical, heterogeneous design.

http://ptolemy.eecs.berkeley.edu

Actor-Oriented Design
Actors with Ports and Attributes



Model of Computation:

- Messaging schema
- Flow of control
- Concurrency

Examples:

- Dataflow
- Process networks
- Synchronous
- Time triggered
- Discrete-event systems
- Publish & subscribe

Most Ptolemy II models of computation are "actor oriented." But the precise semantics depends on the selected "director," which implements a model of computation.

Other Examples of Actor-Oriented Component Frameworks

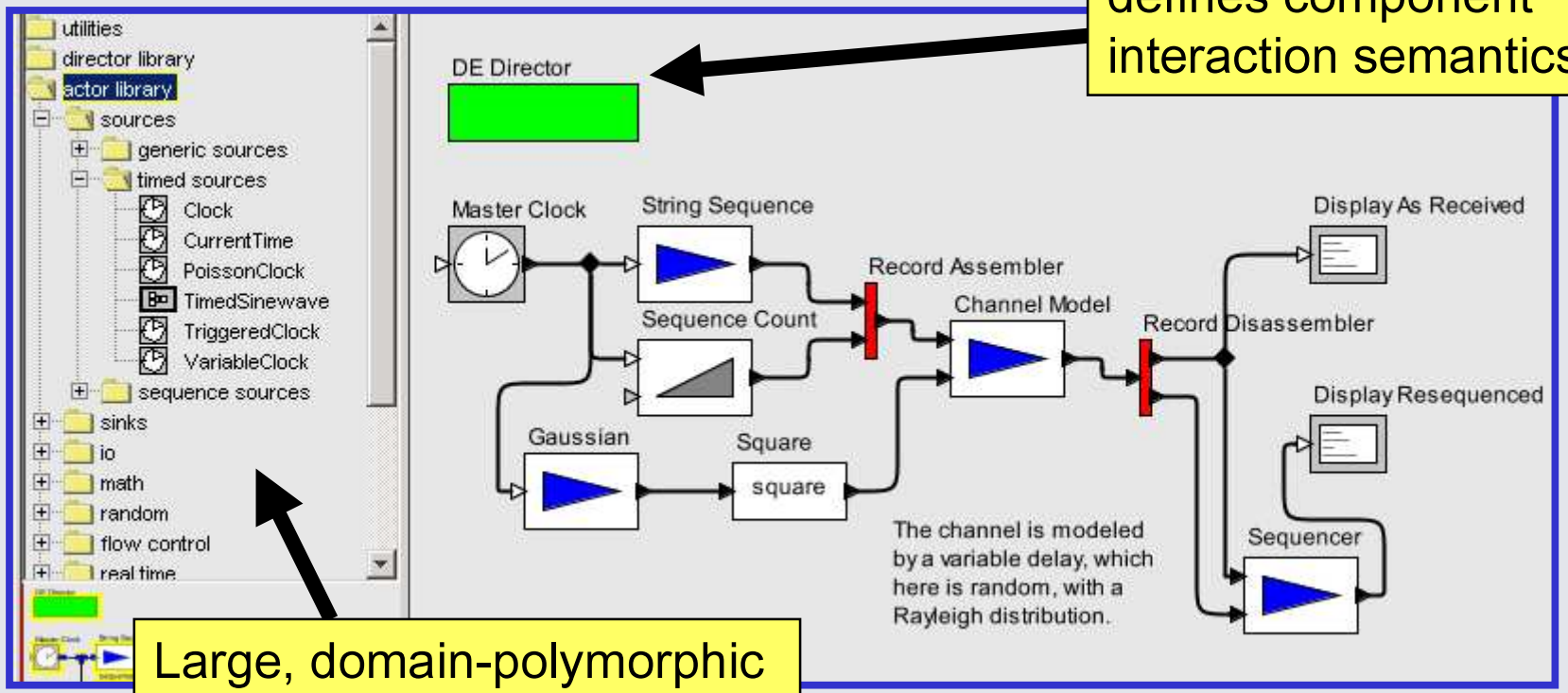
- Simulink (The MathWorks)
- Labview (National Instruments)
- Modelica (Linkoping)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Easy5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

Unlike Ptolemy II, all of these define a fixed model of computation.

Ptolemy Project Principle

The model of computation is not built in to the software framework.

Example of Ptolemy II model:

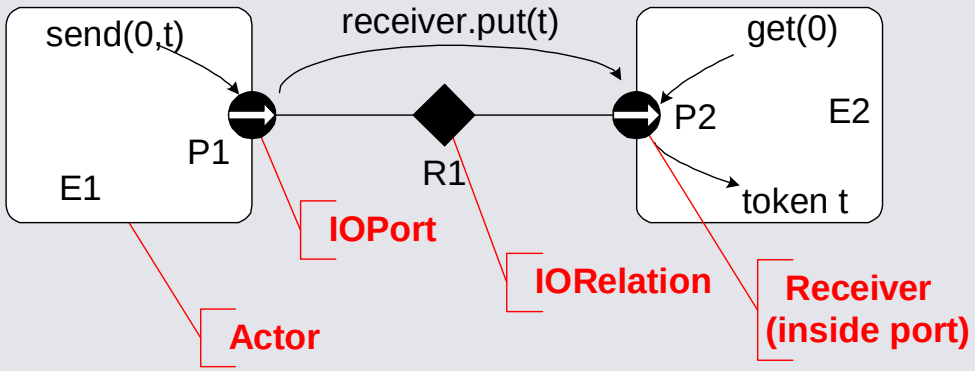


Director from a library defines component interaction semantics

Large, domain-polymorphic component library.

Actor View of Producer/Consumer Components

Basic Transport:



Models of Computation are implemented in Ptolemy II by a "director" and a "receiver" (which is supplied by the director). Examples we have built:

- dataflow (several variants)
- process networks
- push/pull
- continuous-time
- CSP (rendezvous)
- discrete events
- synchronous
- time-driven
- publish/subscribe
- …

Ptolemy II uses the object-oriented principle of polymorphism to realize multiple actor-oriented models of computation.

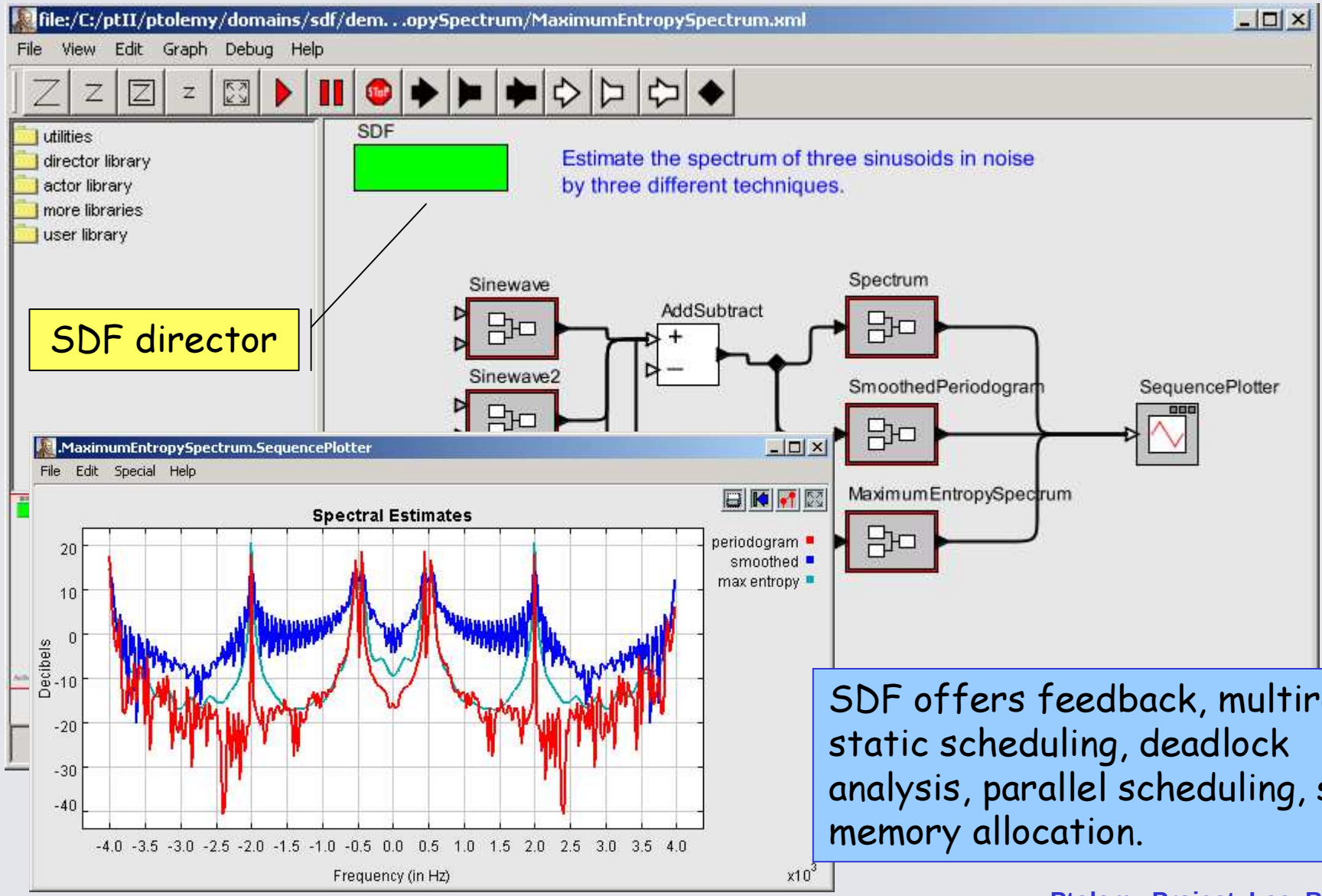
Focus on Dataflow (a few variants)

- Computation graphs [Karp & Miller - 1966]
- Process networks [Kahn - 1974]
- Static dataflow [Dennis - 1974]
- Dynamic dataflow [Arvind, 1981]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986]
- Structured dataflow [Kodosky, 1986]
- PGM: Processing Graph Method [Kaplan, 1987]
- Synchronous languages [Lustre, Signal, 1980's]
- Well-behaved dataflow [Gao, 1992]
- Boolean dataflow [Buck and Lee, 1993]
- Multidimensional SDF [Lee, 1993]
- Cyclo-static dataflow [Lauwereins, 1994]
- Integer dataflow [Buck, 1994]
- Bounded dynamic dataflow [Lee and Parks, 1995]
- Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- …

Many tools, software frameworks, and hardware architectures have been built to support one or more of these.

Synchronous Dataflow (SDF)

(Lee and Messerschmitt, 1986)



SDF offers feedback, multirate, static scheduling, deadlock analysis, parallel scheduling, static memory allocation.

Synchronous Dataflow (SDF)

Fixed Production/Consumption Rates

- Balance equations (one for each channel):

$$f_A N = f_B M$$

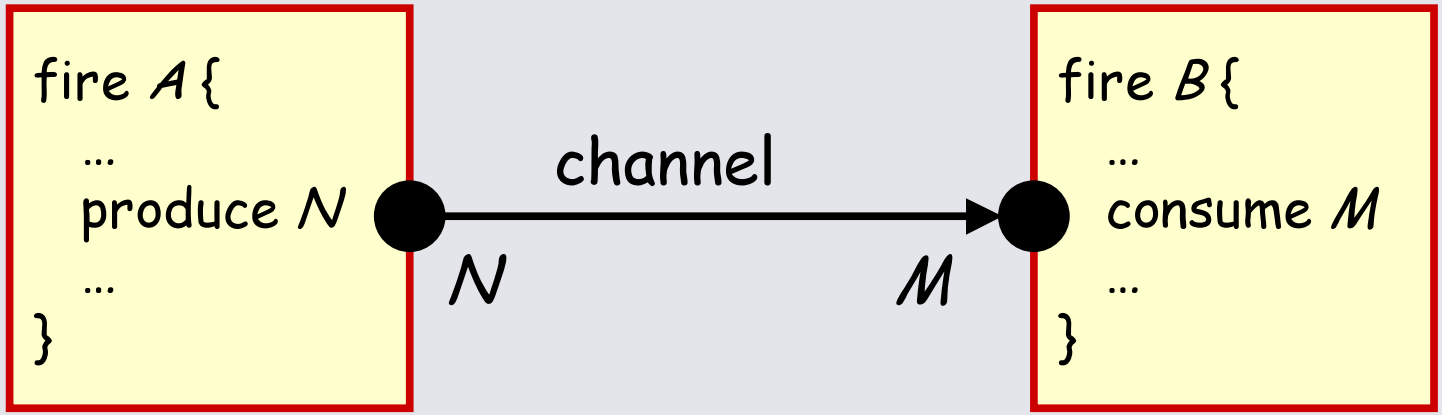
f_A: number of firings per "iteration"

N: number of tokens produced

f_B: number of firings per "iteration"

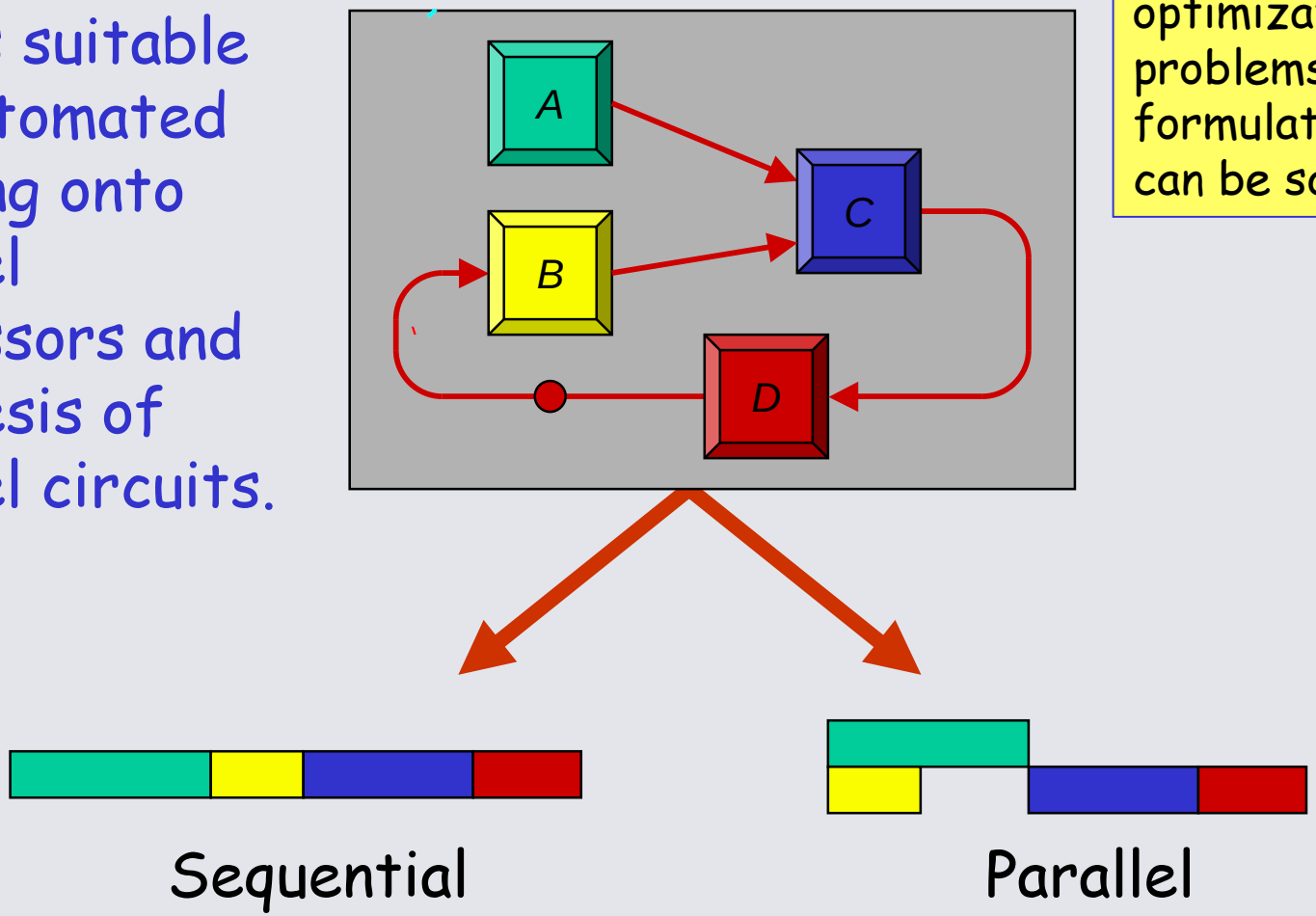
M: number of tokens consumed

- Schedulable statically
- Decidable:
  - buffer memory requirements
  - deadlock



Parallel Scheduling of SDF Models

SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.



Many scheduling optimization problems can be formulated. Some can be solved, too!

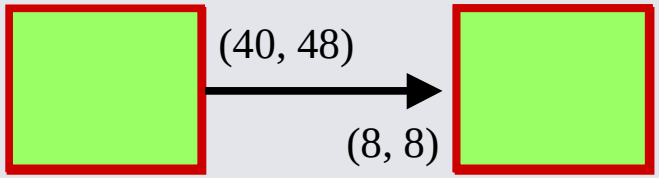
Selected Generalizations

- Multidimensional Synchronous Dataflow (1993)
  - Arcs carry multidimensional streams
  - One balance equation per dimension per arc
- Cyclo-Static Dataflow (Lauwereins, et al., 1994)
  - Periodically varying production/consumption rates
- Boolean & Integer Dataflow (1993/4)
  - Balance equations are solved symbolically
  - Permits data-dependent routing of tokens
  - Heuristic-based scheduling (undecidable)
- Dynamic Dataflow (1981-)
  - Firings scheduled at run time
  - Challenge: maintain bounded memory, deadlock freedom, liveness
  - Demand driven, data driven, and fair policies all fail
- Kahn Process Networks (1974-)
  - Replace discrete firings with process suspension
  - Challenge: maintain bounded memory, deadlock freedom, liveness
- Heterochronous Dataflow (1997)
  - Combines state machines with SDF graphs
  - Very expressive, yet decidable

Multidimensional SDF

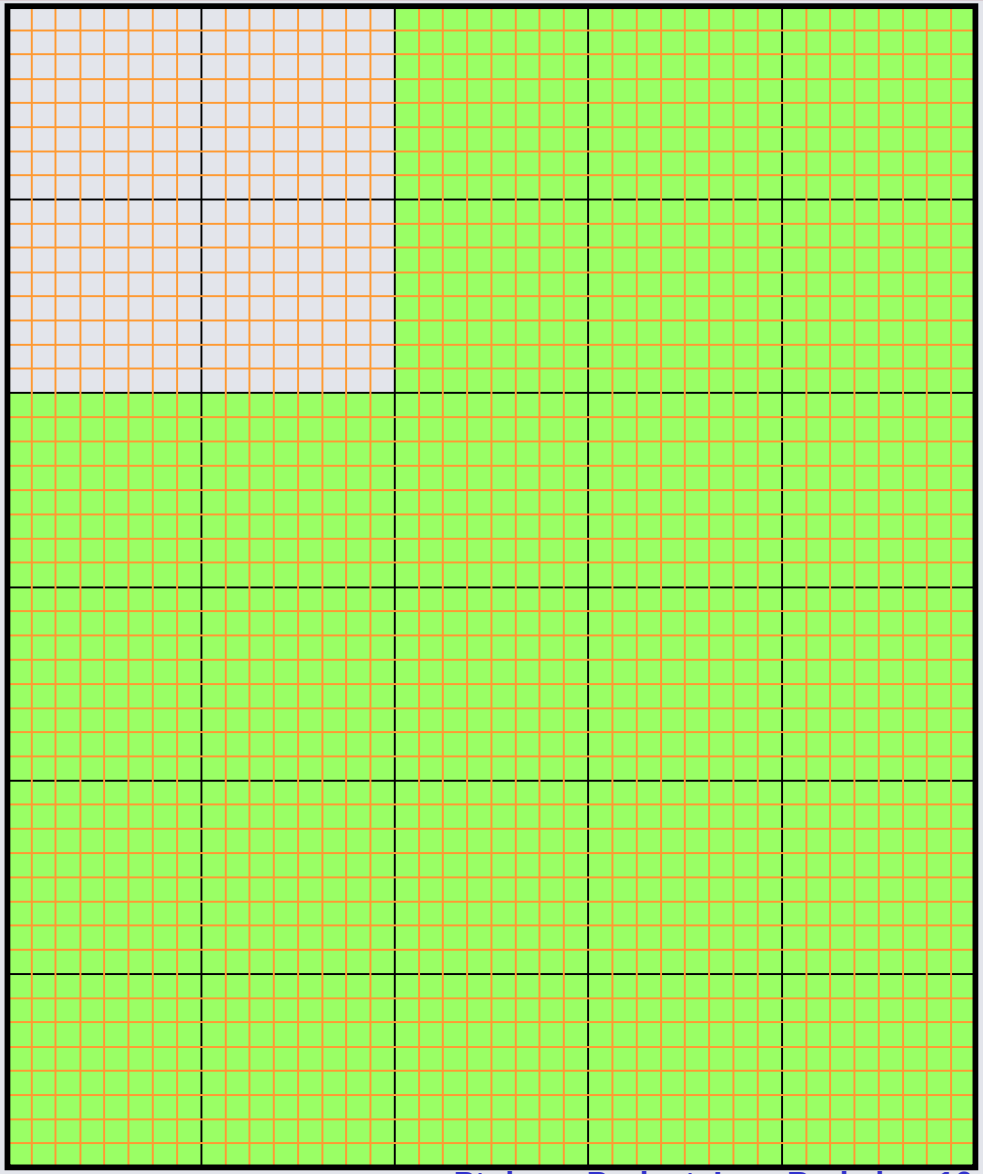
(Lee, 1993)

- Production and consumption of N-dimensional arrays of data:

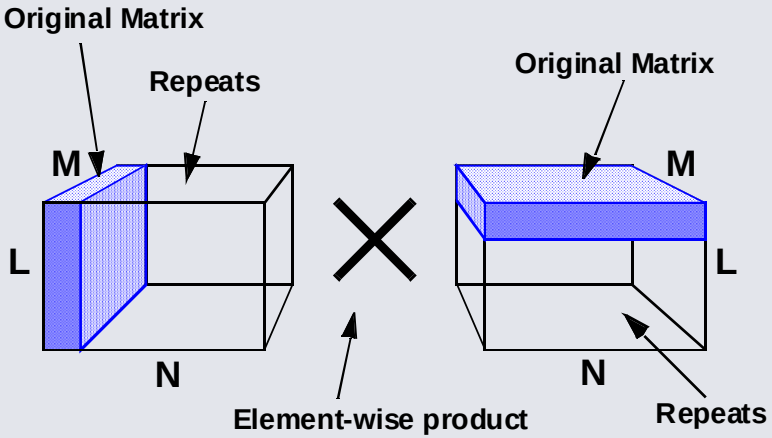


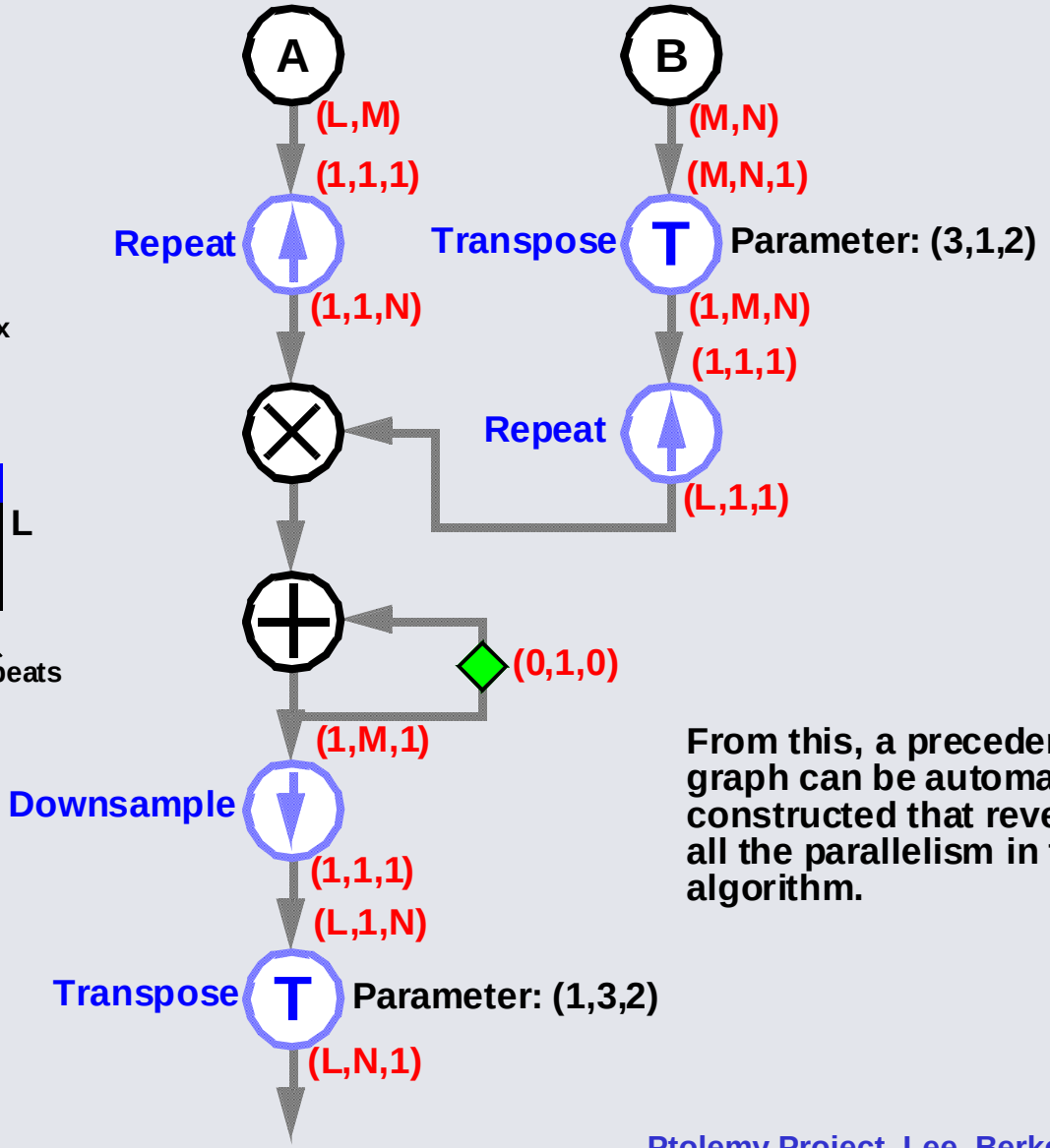
- Balance equations and scheduling policies generalize.
- Much more data parallelism is exposed.

Similar (but dynamic) multidimensional streams have been implemented in Lucid.



MDSDF Structure Exposes Fine-Grain Data Parallelism





From this, a precedence graph can be automatically constructed that reveals all the parallelism in the algorithm.

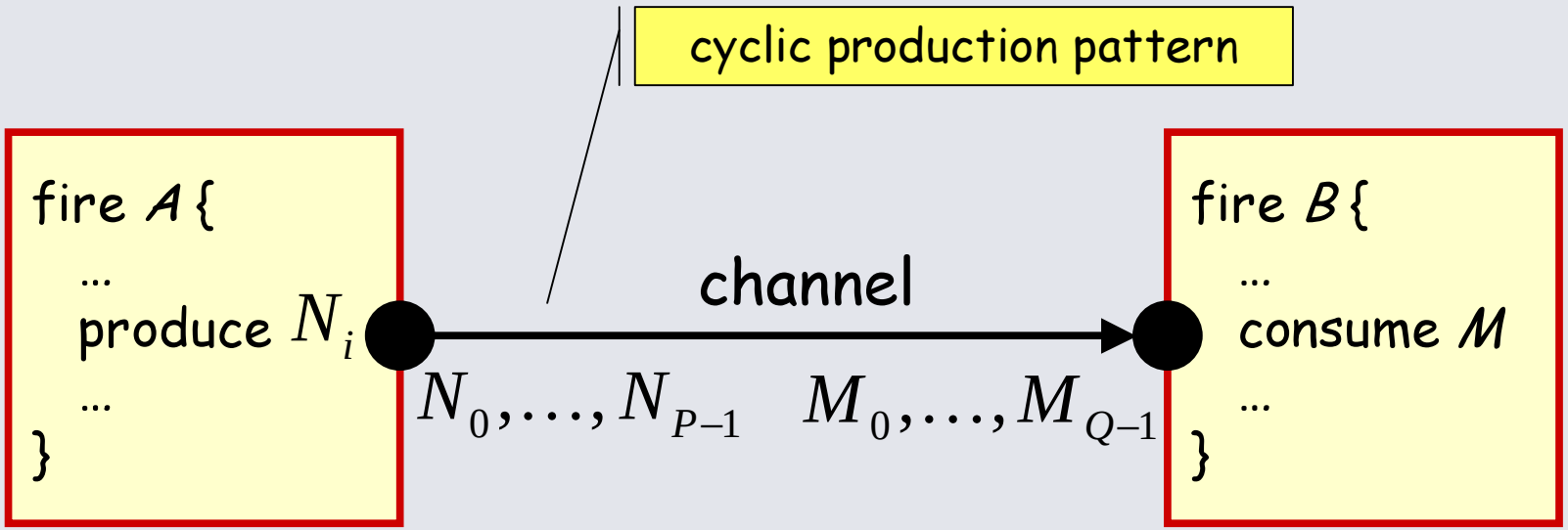
However, such programs are extremely hard to write (and to read).

Cyclostatic Dataflow (CSDF)

(Lauwereins et al., TU Leuven, 1994)

- Actors cycle through a regular production/consumption pattern.
- Balance equations become:

$$f_A \sum_{i=0}^{R-1} N_{i \bmod P} = f_B \sum_{i=0}^{R-1} M_{i \bmod Q};\ R = lcm(P,Q)$$



Boolean and Integer Dataflow (BDF, IDF)

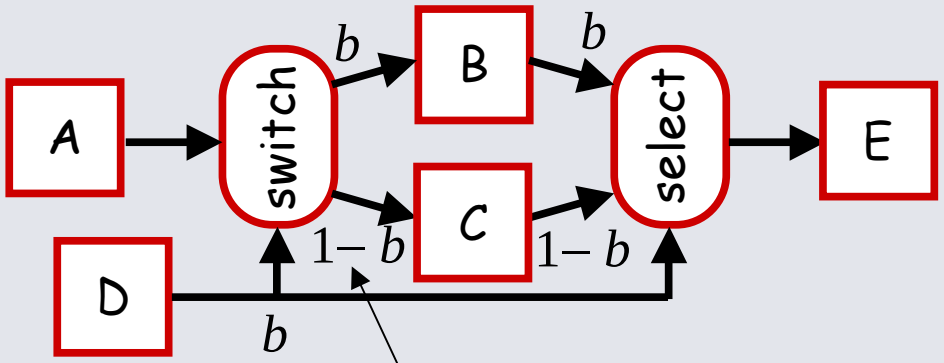
(Lee and Buck, 1993)

- Balance equations are solved symbolically in terms of unknowns that become known at run time.
- An annotated schedule is constructed with predicates guarding each action.
- Existence of such an annotated schedule is undecidable (as is deadlock & bounded memory)

$$f_{switch} b = f_B$$

$$f_{switch} (1-b) = f_C$$

...



Production rate is unknown and is represented symbolically by a variable (*b*).

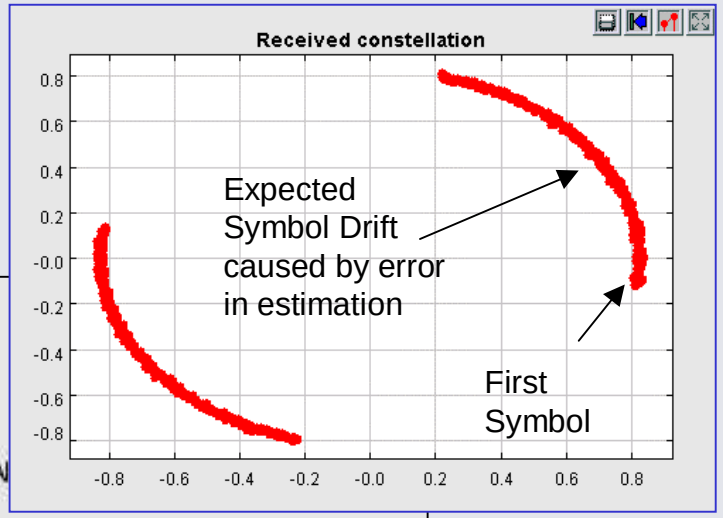
Dynamic Dataflow (DDF)

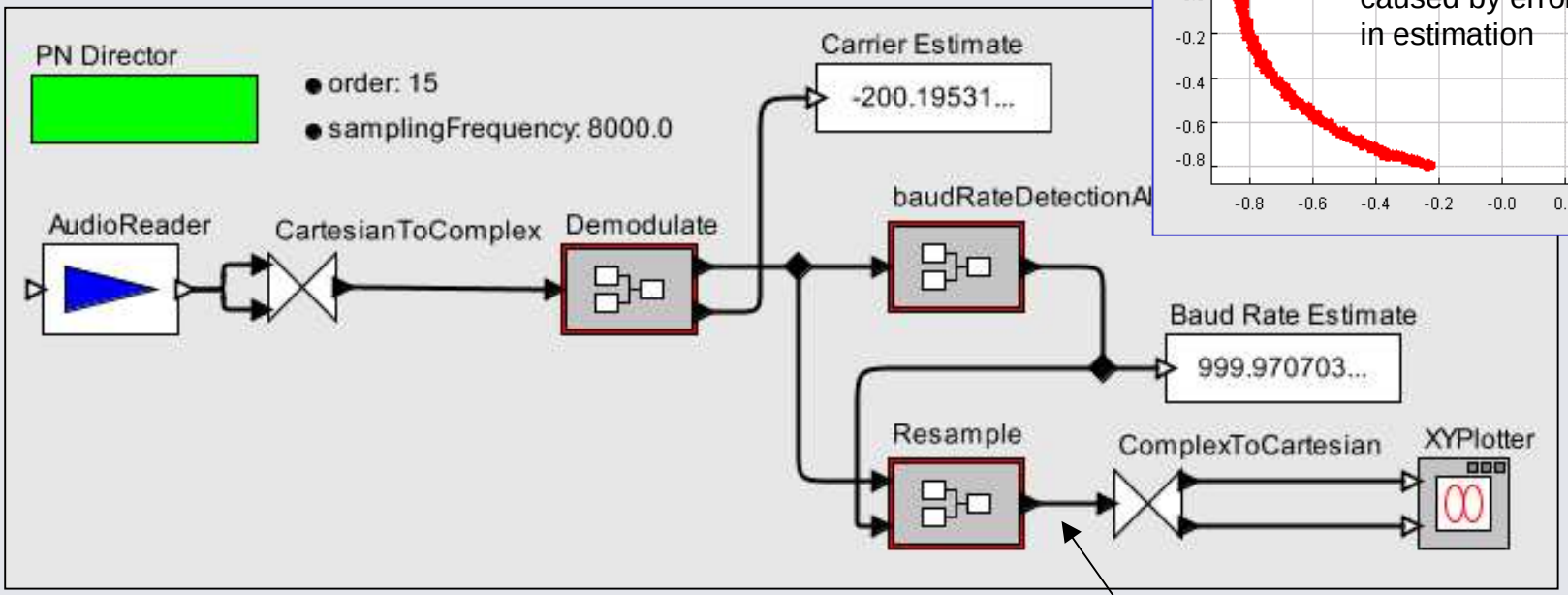
- Actors have *firing rules*
  - Set of finite prefixes on input sequences
  - For determinism: No two such prefixes are joinable under a prefix order
  - Firing function applied to finite prefixes yield finite outputs
- Scheduling objectives:
  - Do not stop if there are executable actors
  - Execute in bounded memory if this is possible
  - Maintain determinacy if possible
- Policies that fail:
  - Data-driven execution
  - Demand-driven execution
  - Fair execution
  - Many balanced data/demand-driven strategies
- Policy that succeeds (Parks 1995):
  - Execute with bounded buffers
  - Increase bounds only when deadlock occurs

DDF, like BDF and IDF is undecidable (deadlock, bounded memory, schedule)

Dynamic Dataflow and Process Networks are Formally Similar, Practically Different

Example from signal intelligence: Detection of unknown signal (PSK in this case)



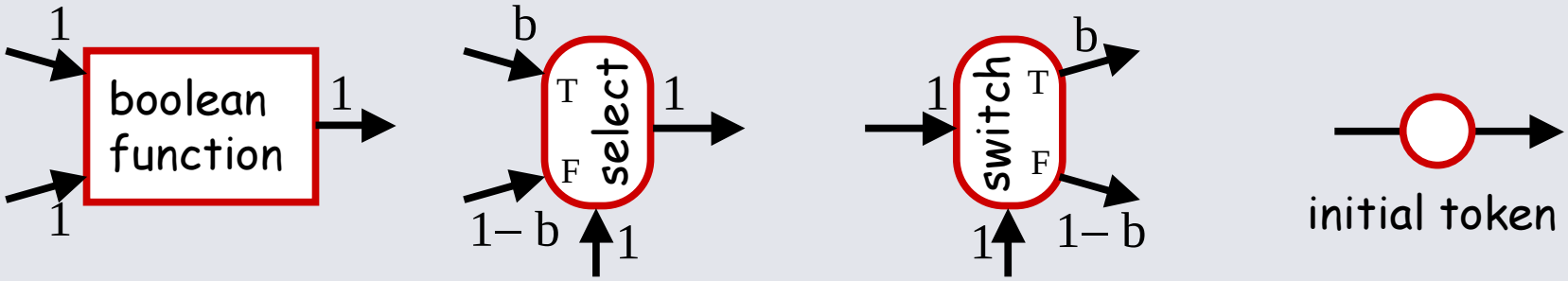


Output data sequence, at detected baud rate. (not known *apriori*)

OEP problem under DARPA MoBIES (Model-based design of embedded software),

Undecidability
(Buck '93)

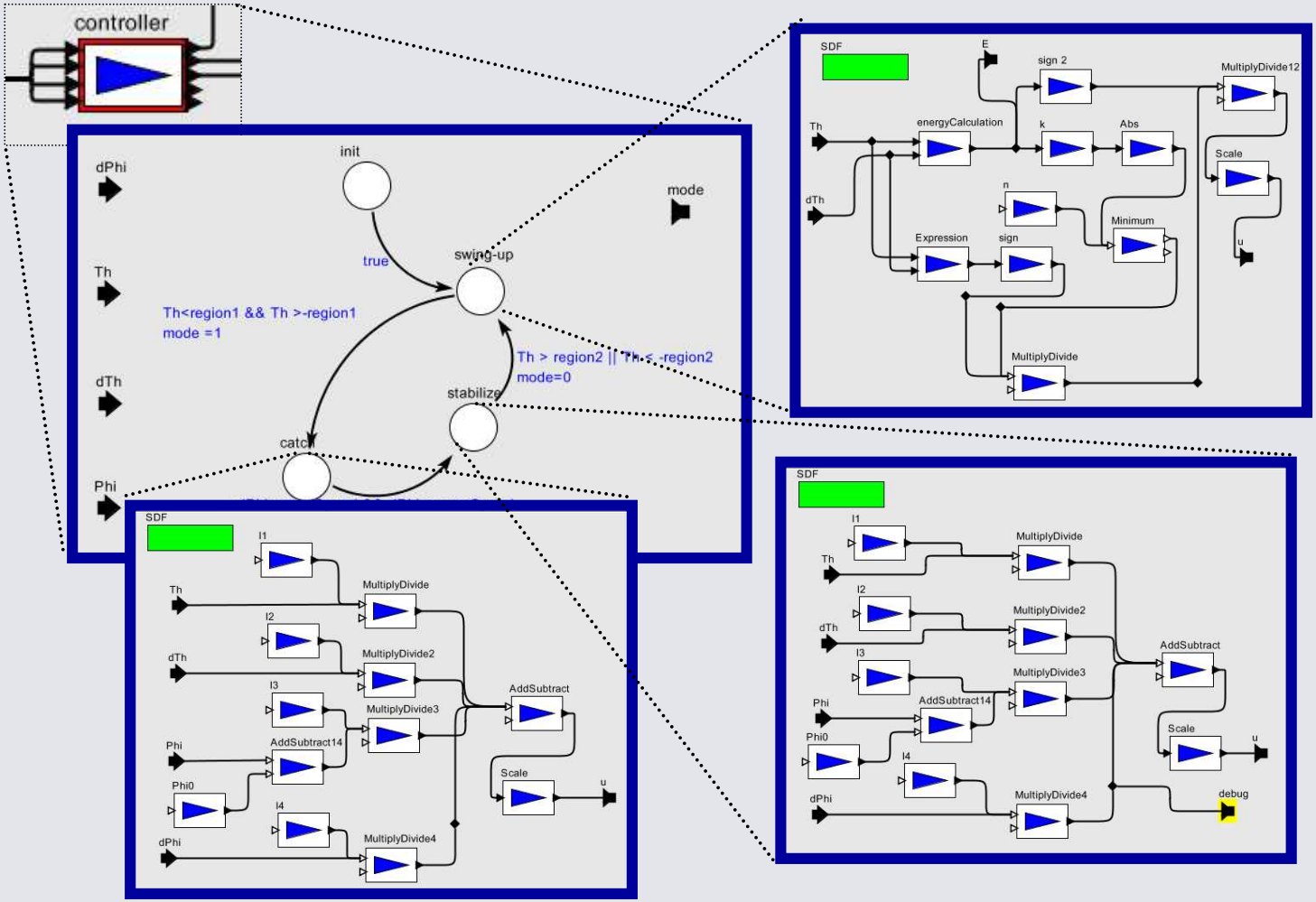
- Sufficient set of actors for undecidability:
  - boolean functions on boolean tokens
  - switch and select
  - initial tokens on arcs



- Undecidable:
  - deadlock
  - bounded buffer memory
  - existence of an annotated schedule

BDF, IDF, DDF, and PN are all undecidable in this sense. Fortunately, we can identify a large decidable subset, which we call heterochronous dataflow (HDF).

Example of a Heterochronous Dataflow (HDF) Model



An actor consists of a state machine and refinements to the states that define behavior.

Heterochronous Dataflow (HDF)

(Girault, Lee, and Lee, 1997)

- An interconnection of actors.
- An actor is either SDF or HDF.
- If HDF, then the actor has:
  - a state machine
  - a refinement for each state
  - where the refinement is an SDF or HDF actor
- Operational semantics:
  - with the state of each state machine fixed, graph is SDF
  - in the initial state, execute one complete SDF iteration
  - evaluate guards and allow state transitions
  - in the new state, execute one complete SDF iteration
- HDF is decidable
  - but complexity can be high

Other Stream-Like Models of Computation

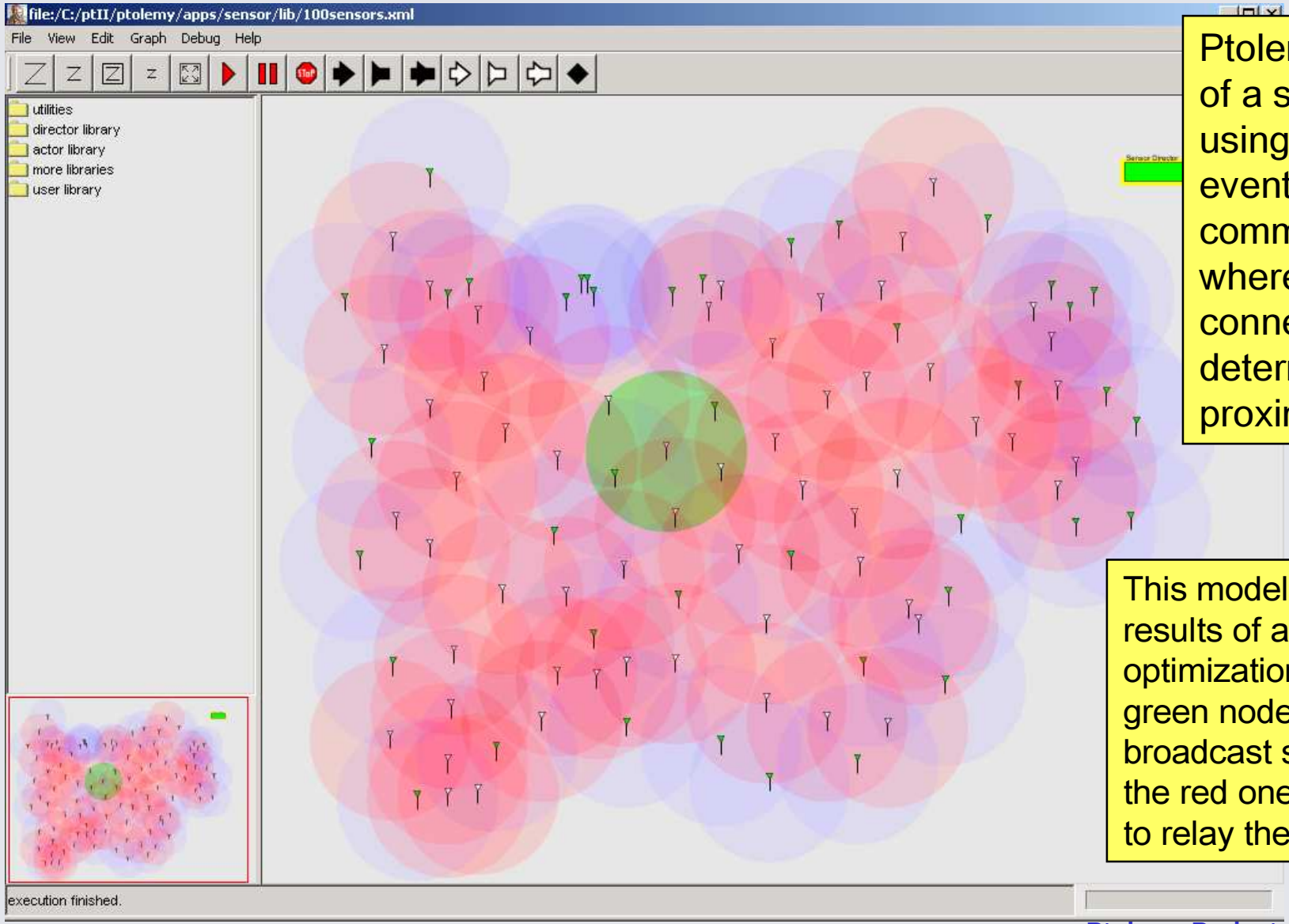
(all implemented in Ptolemy II)

- Push/Pull
  - dataflow with disciplined nondeterminism
  - e.g. Click (Kohler, 2001)
- Discrete events
  - data tokens have time stamps
  - e.g. NS
- Continuous time
  - streams are a continuum of values
  - e.g. Simulink
- Synchronous languages
  - sequence of values, one per clock tick
  - fixed-point semantics
  - e.g. Esterel
- Time triggered
  - similar, but no fixed-point semantics
  - e.g. Giotto
- Modal models
  - state machines + stream-like MoCs, hierarchical
  - e.g. Hybrid systems

all of these include a logical notion of time

Discrete-Event Models

Example: Sensor Nets Modeling

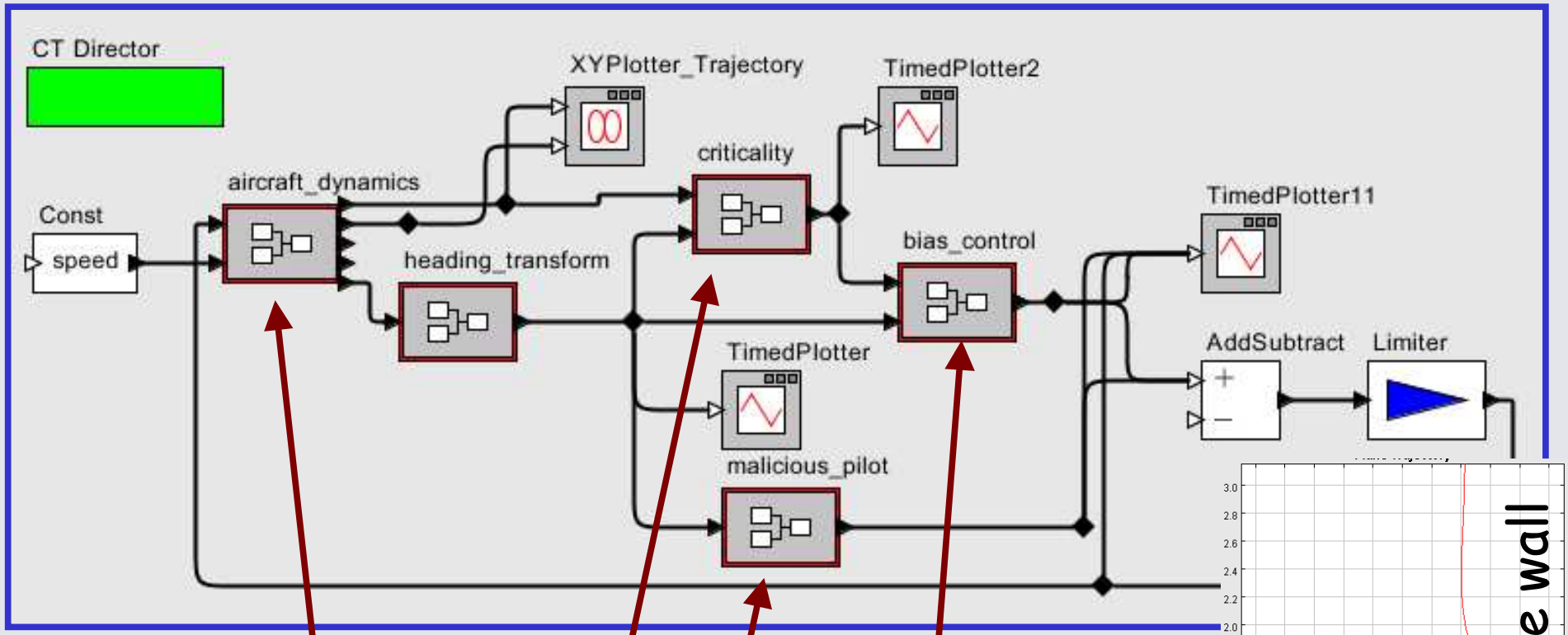


Ptolemy II model of a sensor net using discrete-event communication where connectivity is determined by proximity.

This model shows the results of a power optimization where the green node issues a broadcast signal and the red ones retransmit to relay the signal.

Continuous-Time Models

Example: Soft Walls Avionics System



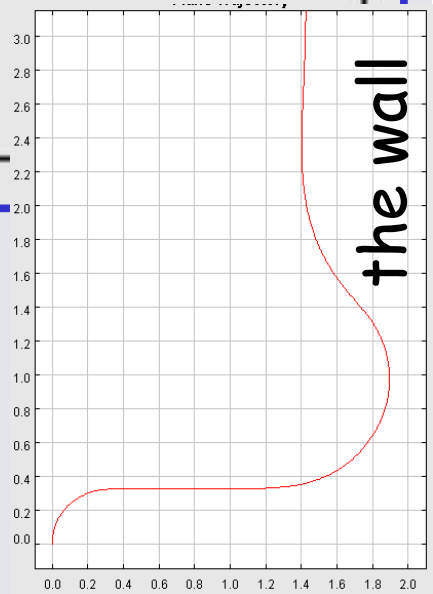
aircraft model

criticality calculation

pilot model

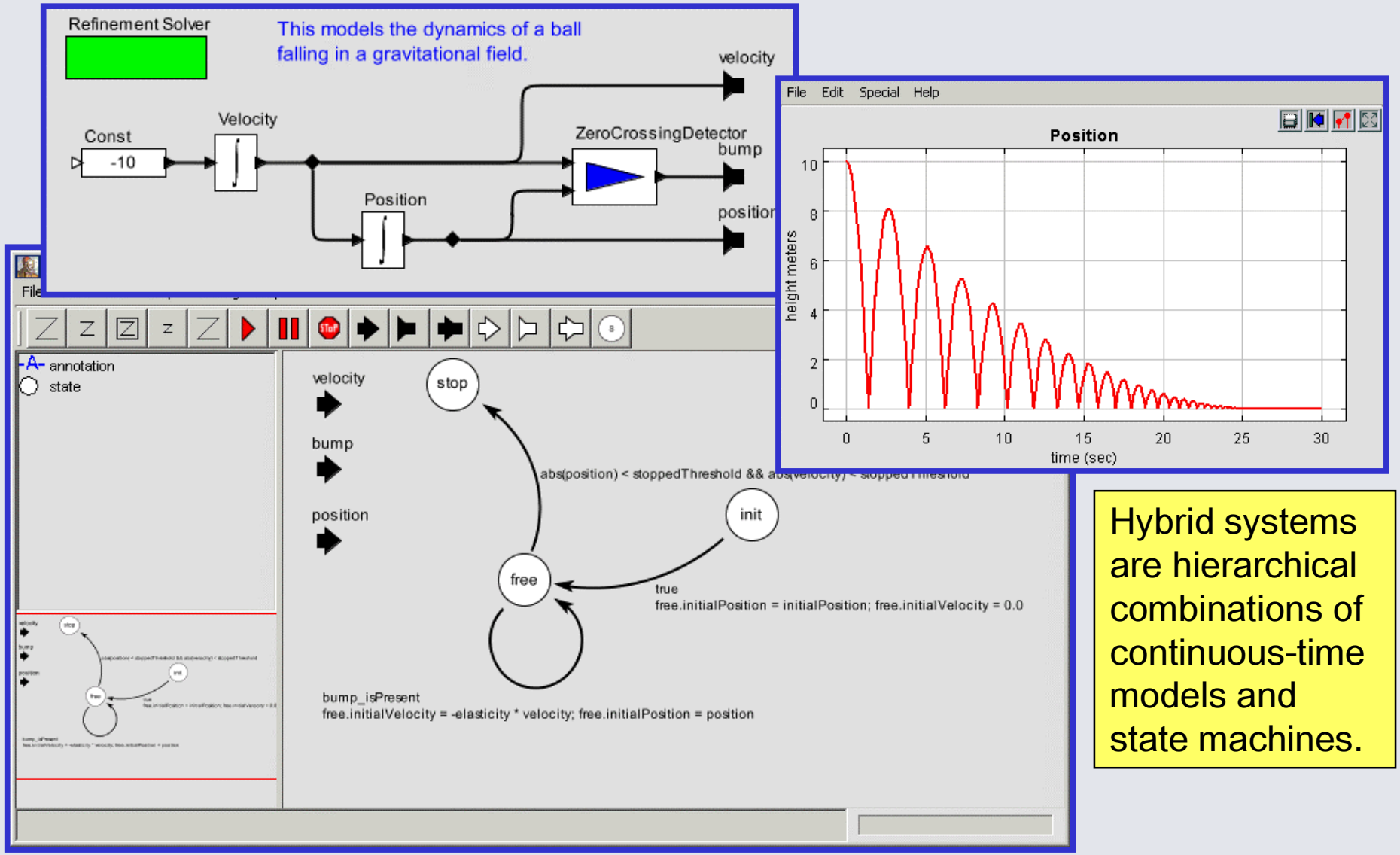
bias control

Analog Computers!



Modal Models

Example: Hybrid Systems



Hybrid systems are hierarchical combinations of continuous-time models and state machines.

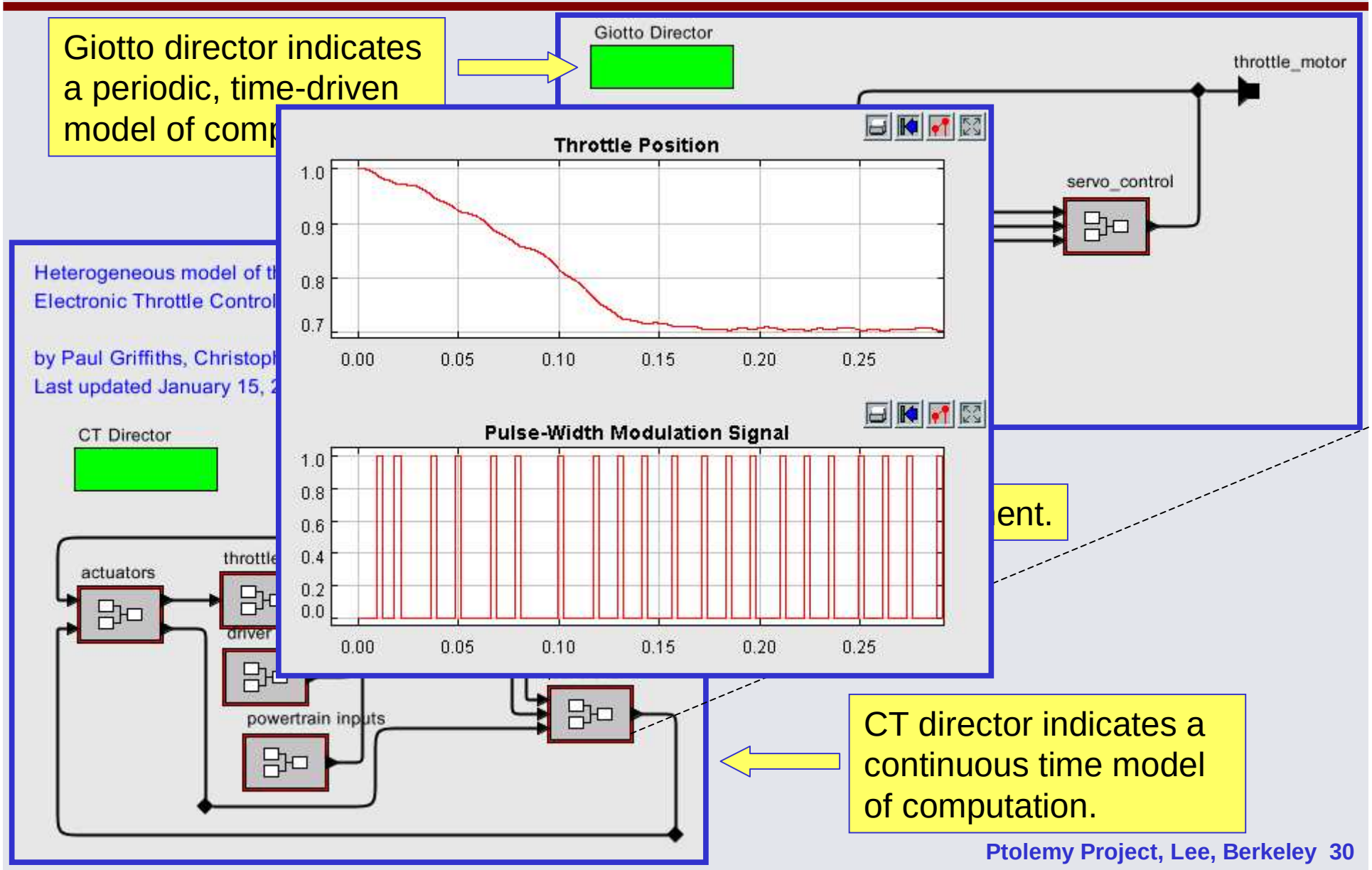
Heterogeneous Models

Modal models are one example of a family of hierarchically heterogeneous models, where diverse models of computation are combined in a hierarchy.

Heterogeneous Models: Periodic/Time-Driven

Example: Control Inside Continuous Time

Giotto director indicates a periodic, time-driven model of comp

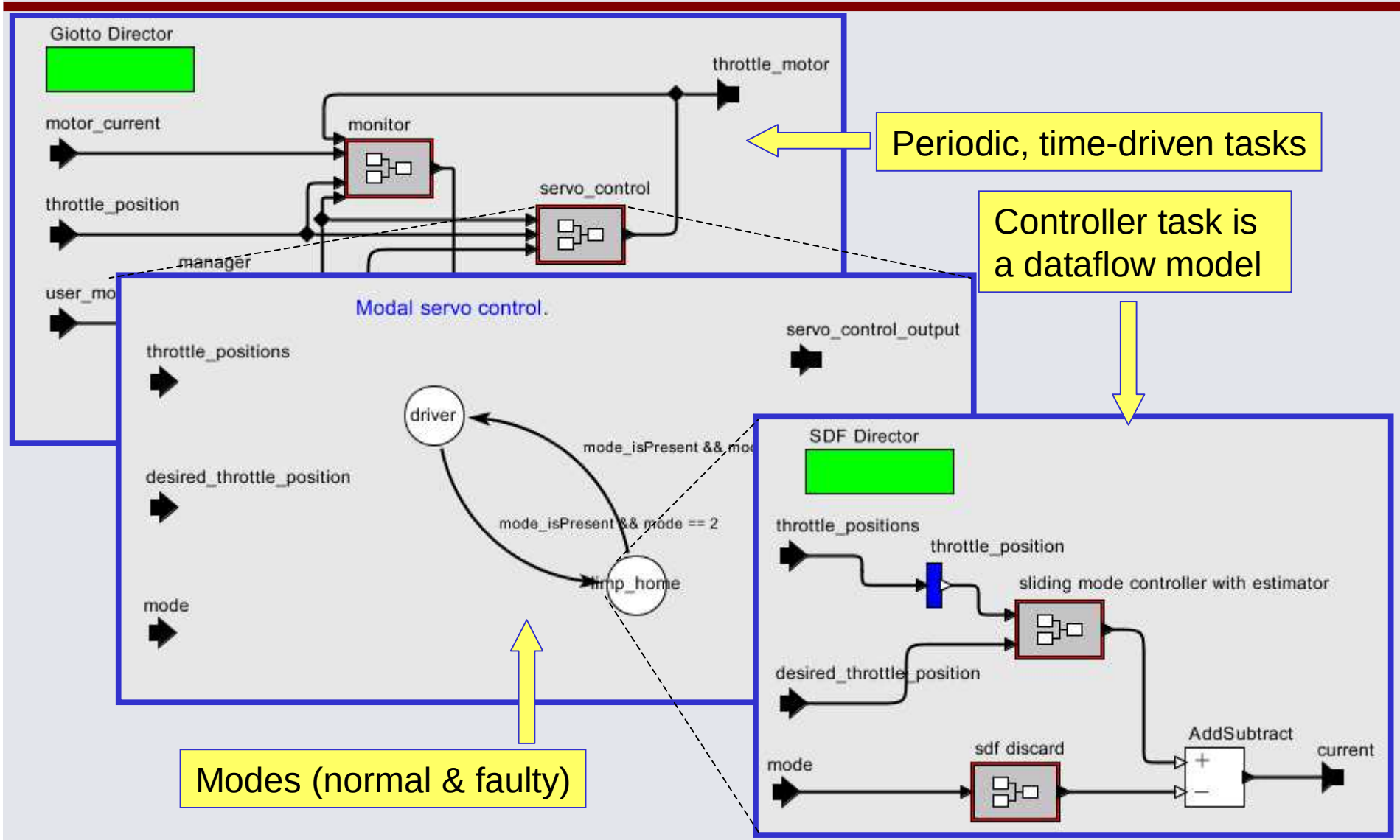


ent.

CT director indicates a continuous time model of computation.

Heterogeneous Modal Models

Example: Modal Control System



Other Topics Not Dealt With

- Scheduling problems
  - throughput, latency, jitter, memory, load balancing, …
- Higher-order components
  - like higher-order functions, but actor-oriented
- Distributed models
  - giving middleware concurrent, actor-oriented semantics
- Domain polymorphism
  - designing components to operate with multiple models of computation
- Behavioral type systems
  - components declare interfaces that make explicit requirements of the MoC

Ptolemy II is open source & free
http://ptolemy.eecs.berkeley.edu