



Balancing Expressiveness and Analyzability in Stream Formalisms

Edward A. Lee

*Robert S. Pepper Distinguished Professor
EECS Department, UC Berkeley*

Invited Talk

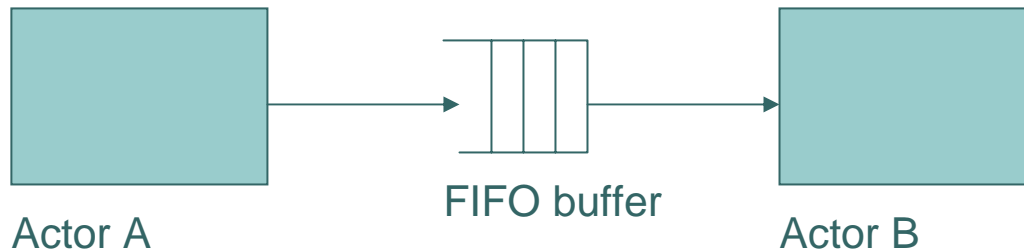
Exploiting Concurrency: Efficiency and Correctness (EC2)

*A Workshop, in conjunction with the 20th International Conference
on Computer Aided Verification (CAV 2008)*

Princeton, NJ, July 7 and 8, 2008



Stream Models



Buffered communication between concurrent components (*actors*).

- **Static scheduling:** Precompute a sequence of actor invocations (*firings*) or thread interleavings (for *process networks*)
- **Dynamic scheduling:** When a compute resources becomes available, determine which actor can execute and choose one.



Streams are not a new idea, of course 1960s and 1970s

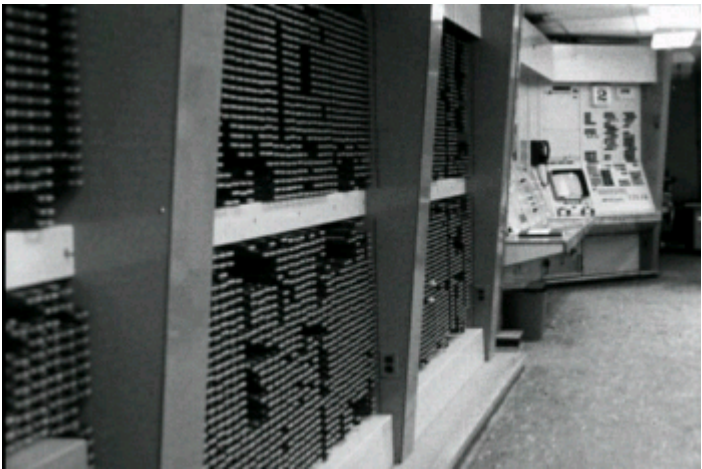
- Visual programs [Sutherland 66]
- Computation graphs [Karp & Miller 66]
- Unix pipes [?? 70's]
- Dennis dataflow [Dennis 74]
- Kahn networks [Kahn 74, K & MacQueen 77]
- ...

Interest rekindled due to parallelism...

The First (?) Stream Programming Language

The On-Line Graphical Specification of Computer Procedures

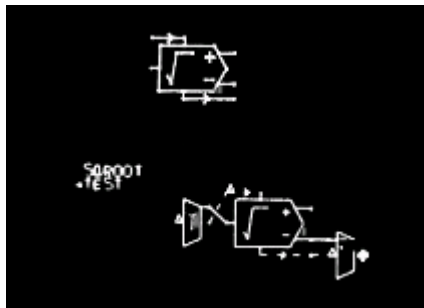
W. R. Sutherland, Ph.D. Thesis, MIT, 1966



MIT Lincoln Labs TX-2 Computer



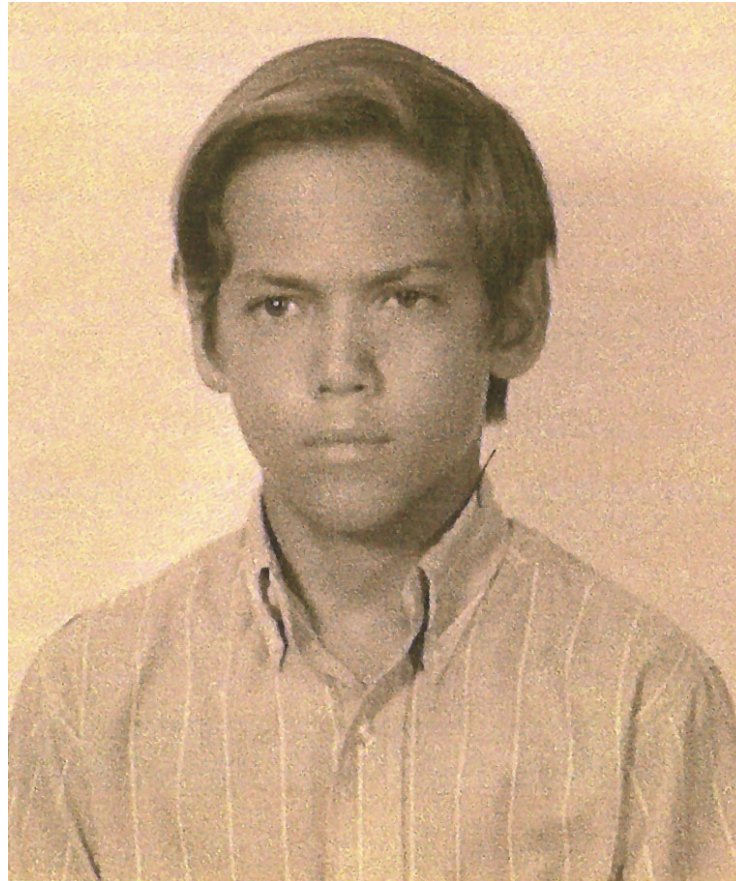
Bert Sutherland with a light pen

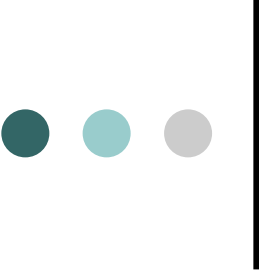


Bert Sutherland used the first acknowledged object-oriented framework (Sketchpad, created by his brother, Ivan Sutherland) to create the first actor-oriented programming language (which had a visual syntax and a stream-like semantics).

Partially constructed iterative square-root program with a class definition (top) and instance (below).

Your Speaker in 1966





The Next Generation 1980s and 1990s

- Dynamic dataflow [Arvind, 1981]
- Structured dataflow [Matwin & Pietrzykowski 1985]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986]
- Structured dataflow and LabVIEW [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.



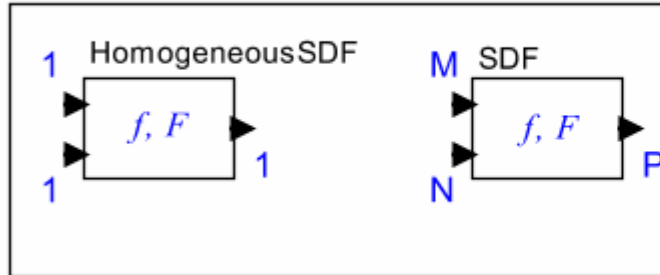
Some Questions

- Termination, deadlock, and livelock (halting)
- Bounding the buffers.
- Fairness
- Parallelism
- Data structures and shared data
- Determinism
- Syntax

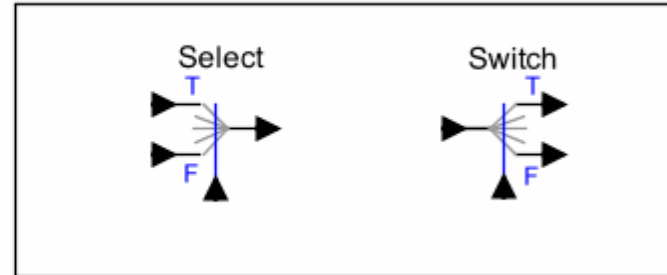
Dennis-Style Dataflow

Firing rules:
the number of
tokens
required to fire
an actor.

Synchronous Dataflow



Dynamic Dataflow



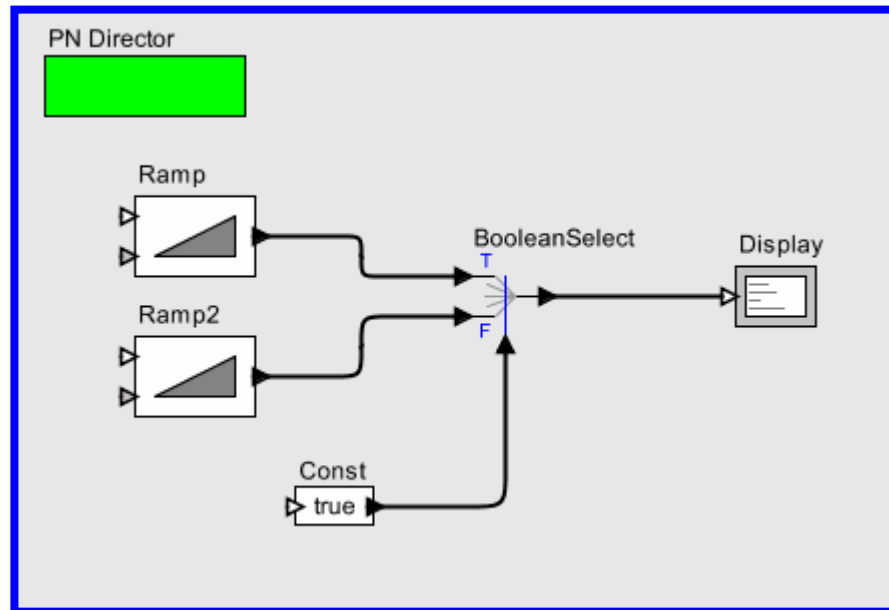
Each signal has form $x: \mathbb{N} \rightarrow R$. The function F maps such signals into such signals. The function f (the “firing function”) maps prefixes of these signals into prefixes of the output. Operationally, the actor *consumes* some number of tokens and *produces* some number of tokens to construct the output signal(s) from the input signal(s). If the number of tokens consumed and produced is a constant over all firings, then the actor is called a *synchronous dataflow* (SDF) actor.

A *signal* or *stream* is a (potentially infinite) sequence of communicated data tokens..

Question 1:

Is “Fair” Scheduling a Good Idea?

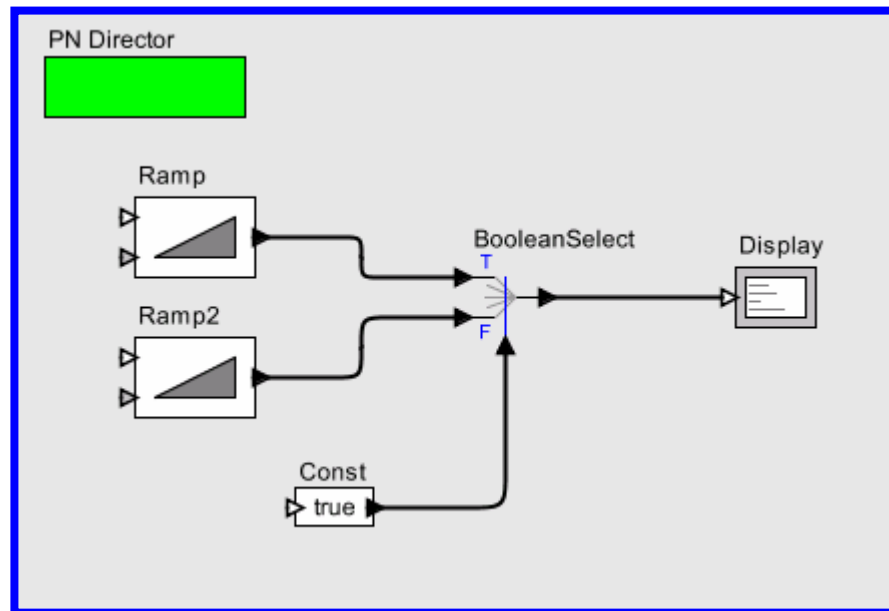
In the following model, what happens if every actor is given an equal opportunity to run?



Question 2:

Is “Data-Driven” Execution a Good Idea?

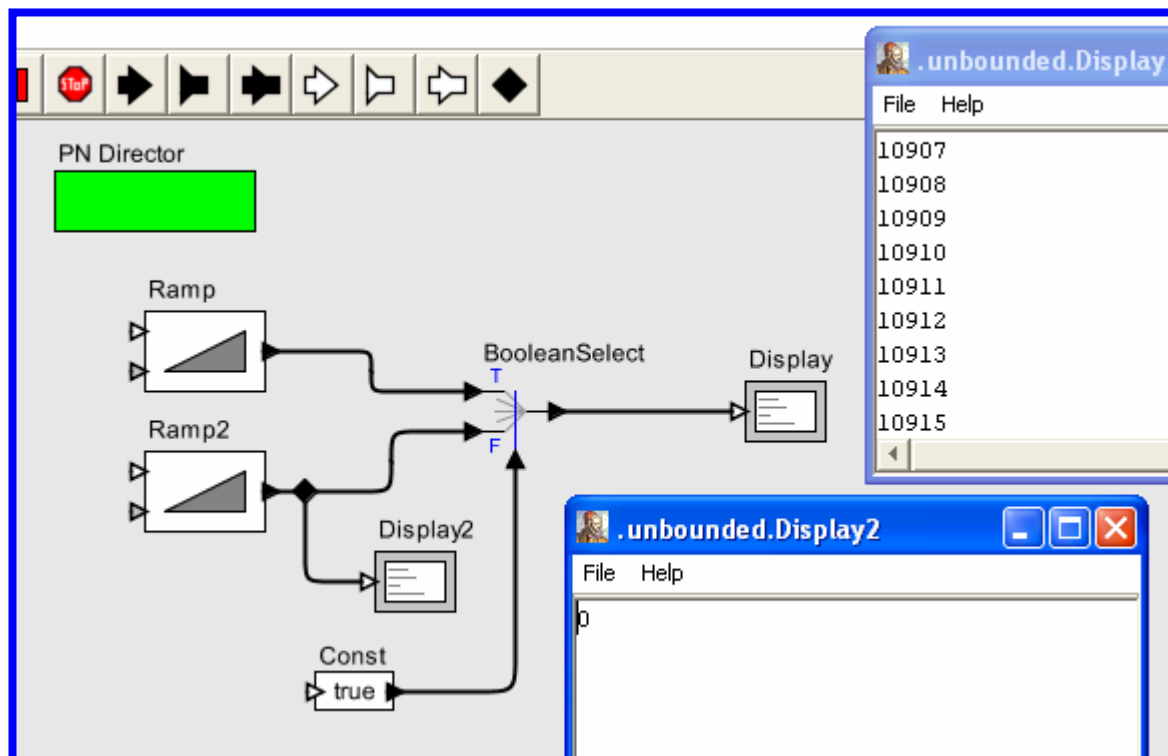
In the following model, if actors are allowed to run when they have input data on connected inputs, what will happen?



Question 3:

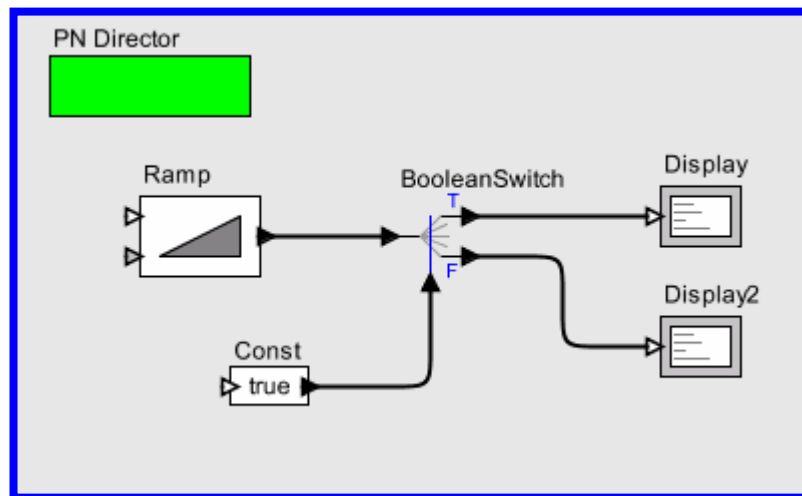
When are Outputs Required?

Is the execution shown for the following model the “right” execution?

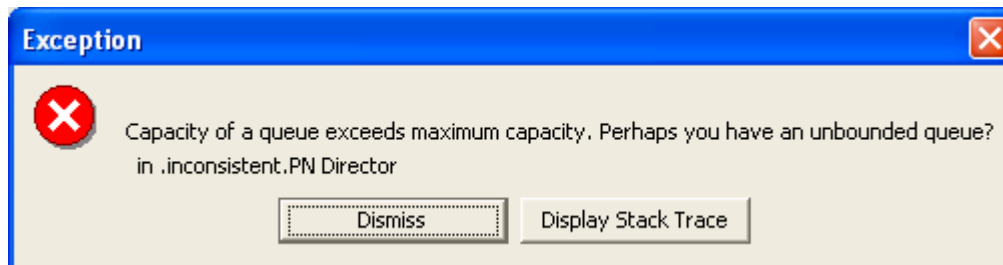
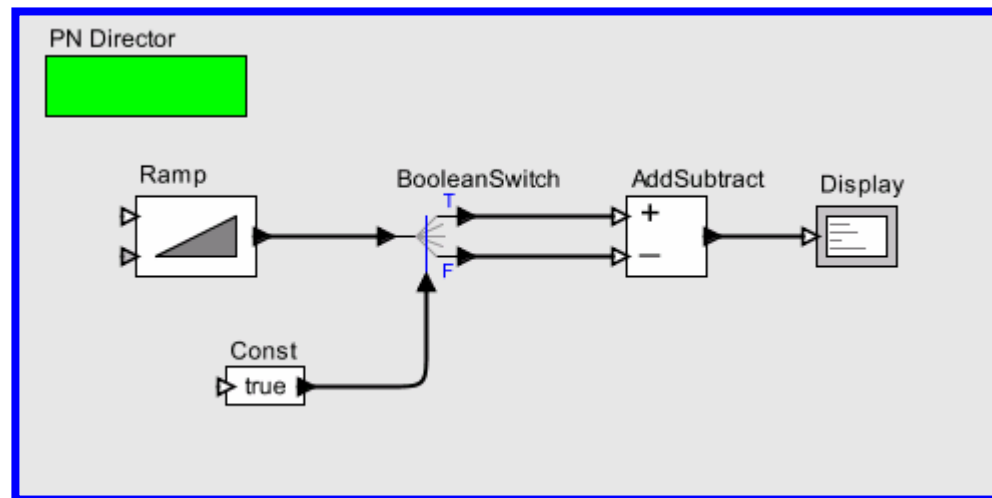


Question 4: Is “Demand-Driven” Execution a Good Idea?

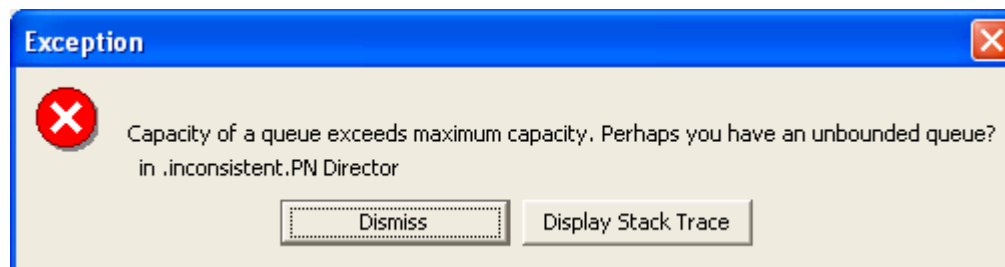
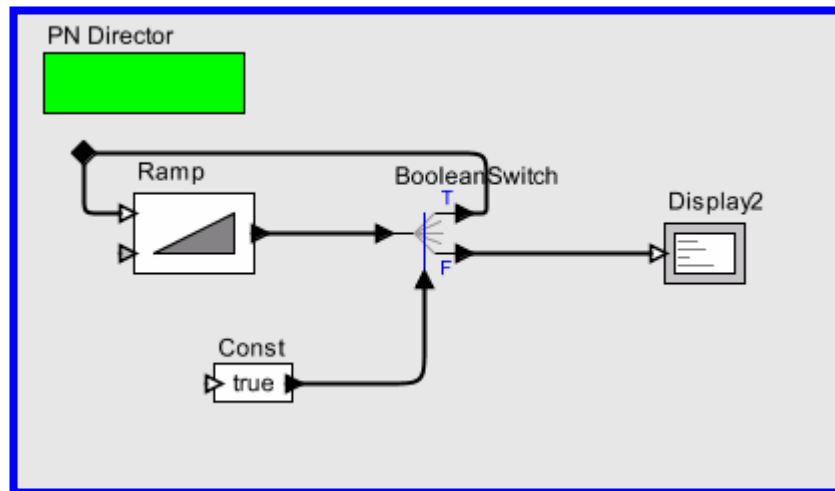
In the following model, if actors are allowed to run when another actor requires their outputs, what will happen?



Question 5: What is the “Correct” Execution of This Model?



Question 6: What is the Correct Behavior of this Model?





Naïve Schedulers Fail

- Fair
- Demand driven
- Data driven
- Most mixtures of demand and data driven



A Practical Policy

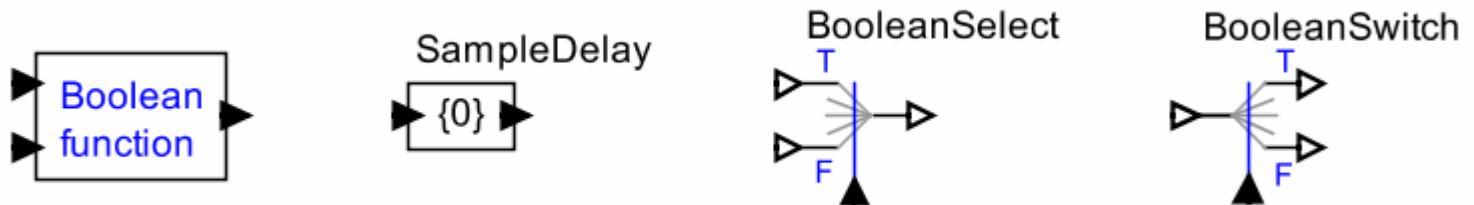
Define a ***correct execution*** to be any execution for which after any finite time every signal is a prefix of the signal given by the (Kahn) least-fixed-point semantics.

Define a ***useful execution*** to be a correct execution that satisfies the following criteria:

1. For every non-terminating model, after any finite time, a useful execution will extend at least one stream in finite (additional) time.
2. If a correct execution satisfying criterion (1) exists that executes with bounded buffers, then a useful execution will execute with bounded buffers.

Undecidability and Turing Completeness [Buck 93]

Given the following four actors and Boolean streams, you can construct a universal Turing machine:



Hence, the following questions are undecidable:

- Will a model deadlock (terminate)?
- Can a model be executed with bounded buffers?



Parks' Strategy [Parks 95]

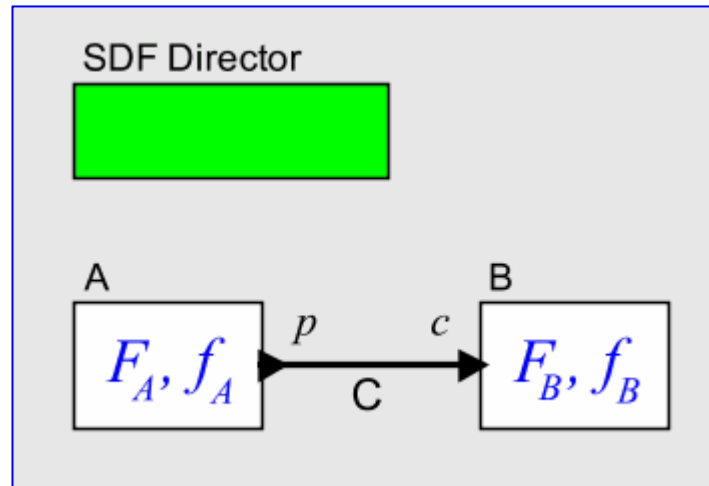
This “solves” the undecidable problems:

- Start with an arbitrary bound on the capacity of all buffers.
- Execute as much as possible.
- If deadlock occurs and at least one actor is blocked on a write, increase the capacity of at least one buffer to unblock at least one write.
- Continue executing, repeatedly checking for deadlock.

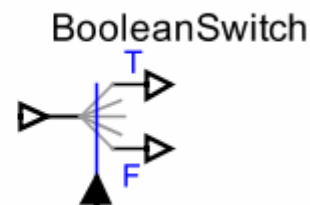
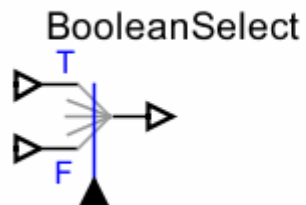
This delivers a useful execution, but may take infinite time to tell you whether a model deadlocks and how much buffer memory it requires.

Synchronous Dataflow (SDF)

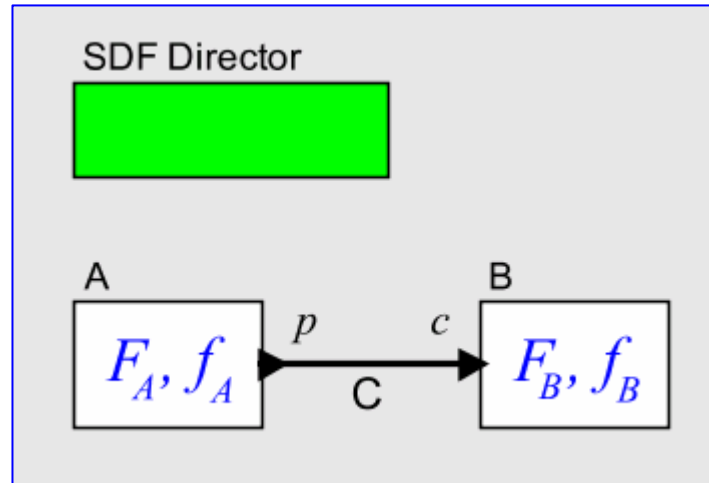
[Lee & Messerschmitt, 87]



Limit the expressiveness by constraining the number of tokens consumed and produced on each firing to be constant. Eliminates:



Balance Equations



Let q_A, q_B be the number of firings of actors A and B.

Let p_C, c_C be the number of token produced and consumed on a connection C.

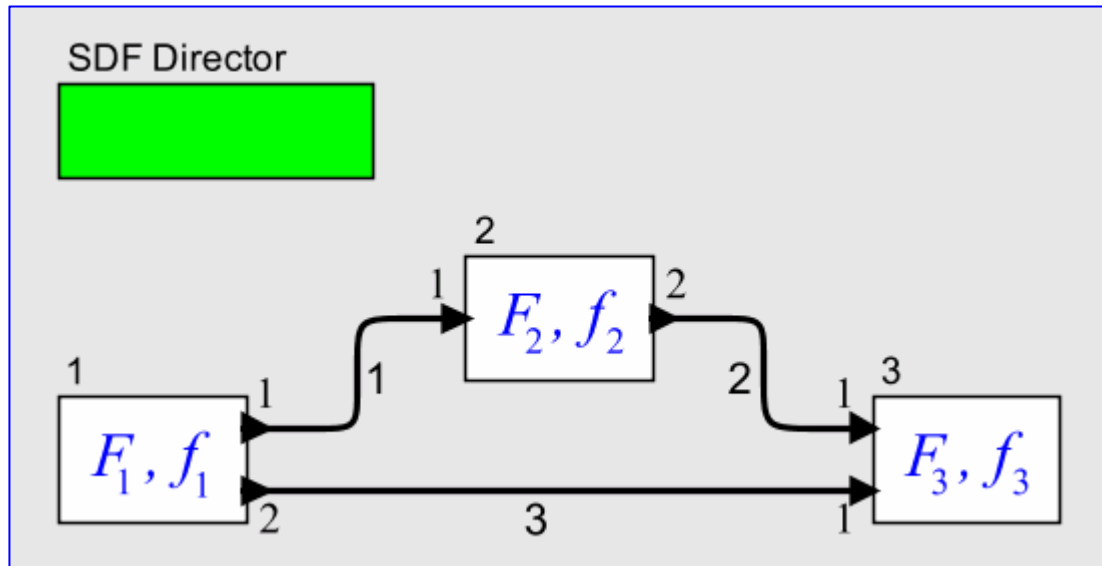
Then the system is *in balance* if for all connections C

$$q_A p_C = q_B c_C$$

where A produces tokens on C and B consumes them.

Example

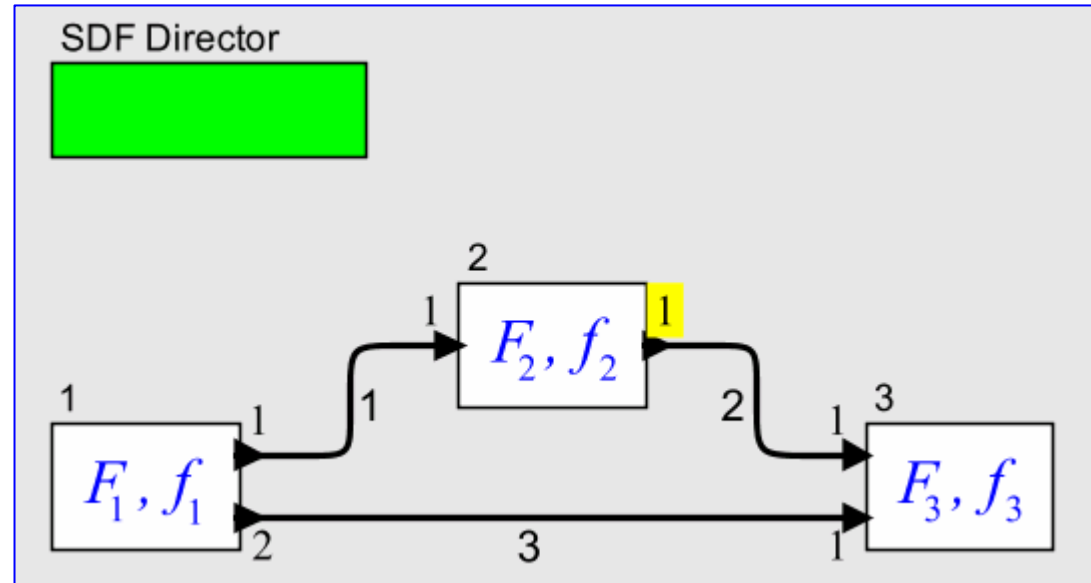
Consider this example:



The balance equations imply that actor 3 must fire twice as often as the other two actors.

Inconsistent Models have no Non-Trivial Solution to the Balance Equations

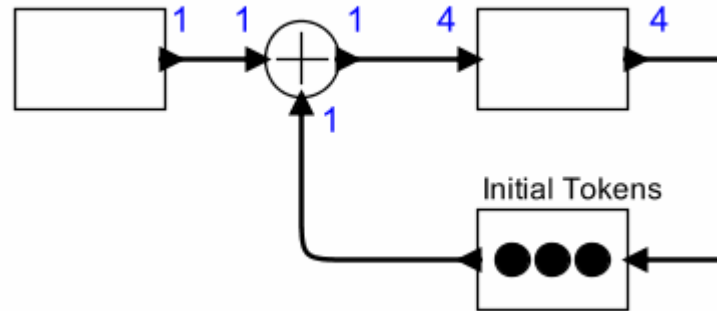
$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$



This production/consumption matrix has rank 3, so there are no nontrivial solutions to the balance equations.

Note that this model can execute forever, but it requires unbounded memory.

Deadlock



Some dataflow models cannot execute forever. In the above model, the feedback loop injects initial tokens, but not enough for the model to execute.



Decidable Models

For SDF, boundedness and deadlock are decidable. Moreover, parallel scheduling can be done statically, and useful optimization problems can be solved. See for example:

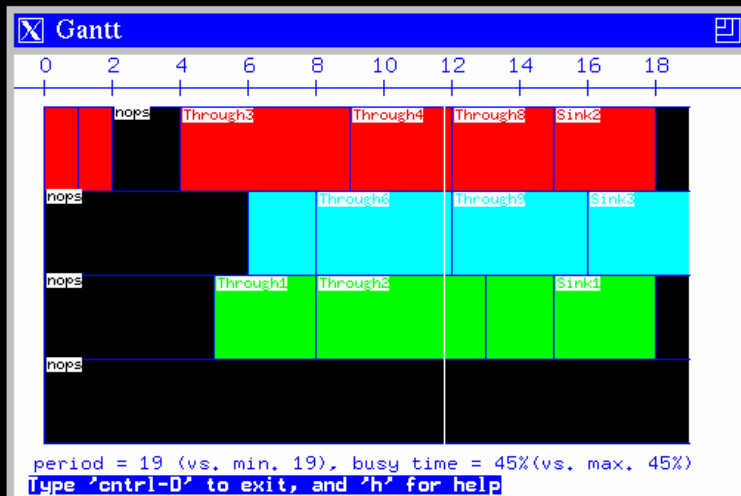
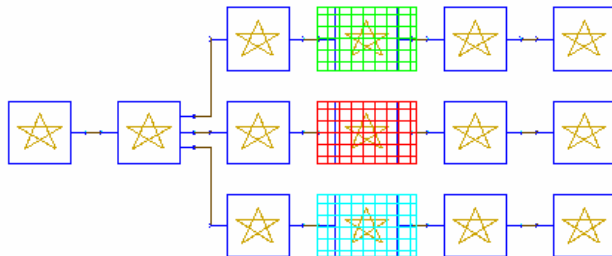
1. Ha and Lee, "Compile-Time Scheduling and Assignment of Dataflow Program Graphs with Data-Dependent Iteration," *IEEE Trans. on Computers*, November, 1991.
2. Sih and Lee, "Declustering: A New Multiprocessor Scheduling Technique," *IEEE Trans. on Parallel and Distributed Systems*, June 1993.
3. Sih and Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. on Parallel and Distributed Systems*, February 1993.

Gabriel and Ptolemy Classic Leveraged SDF to Generate Parallel Code

SDF model, parallel schedule, and synthesized parallel code (1990)

[0, 0] .../Sih-4-1:schematic (1000, 775)

Scheduling example from Gil Sih's dissertation
Figure 4-1 (modified slightly)



```
codeblock(std) {
: initialize address registers for coef and
delayLineove    #saddr(coef)+$val(coefLen)-1,r3
: insert here
move            $ref(delayLineStart),r5
: delayLine
move            #$val(stepSize),x1
move            $ref(error),x0
mpyr            x0,x1,a
move            a,x0
move            x:(r3),b            y:(r5)+,y0
}

codeblock(loop) {
do              #$val(loopVal),$label(endloop)
macr            x0,y0,b
move            b,x:(r3)-
move            x:(r3),b            y:(r5)+,y0
$label(endloop)
}

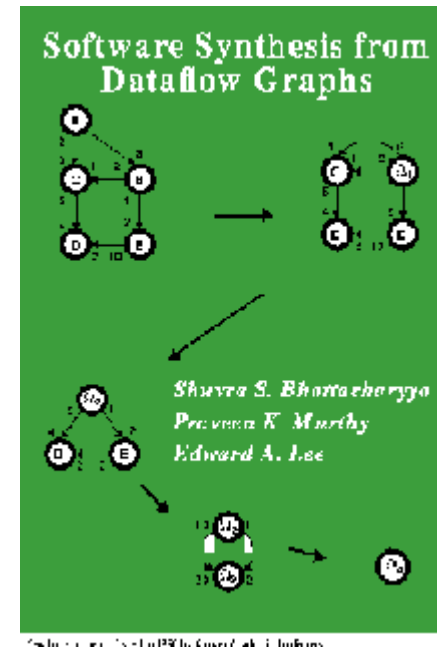
codeblock(noloop) {
macr            x0,y0,b
move            b,x:(r3)-
move            x:(r3),b            y:(r5)+,y0
}
```

It is an interesting (and rich) research problem to minimize interlocks and communication overhead in complex multirate applications.

● ● ● | Although this makes scheduling decidable,
many complex optimization problems
remain.

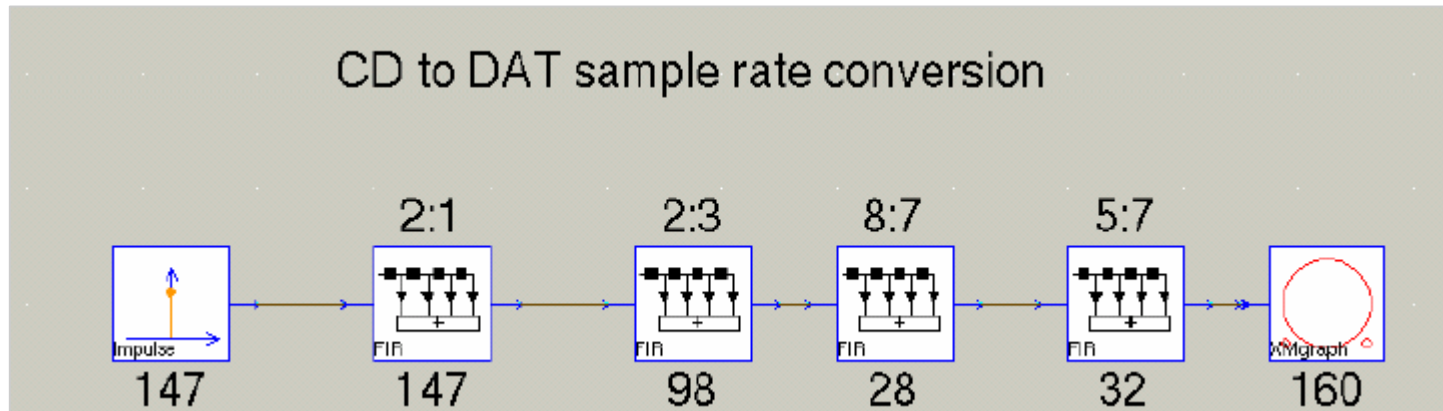
Optimization criteria that might be applied:

- Minimize buffer sizes.
- Minimize the number of actor activations.
- Minimize the size of the representation of the schedule (code size).
- Maximize the throughput.
- Minimize latency.



See Bhattacharyya, Murthy, and Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Press, 1996.

Example: Minimum Buffer Schedule for a 6-Actor Dataflow Model



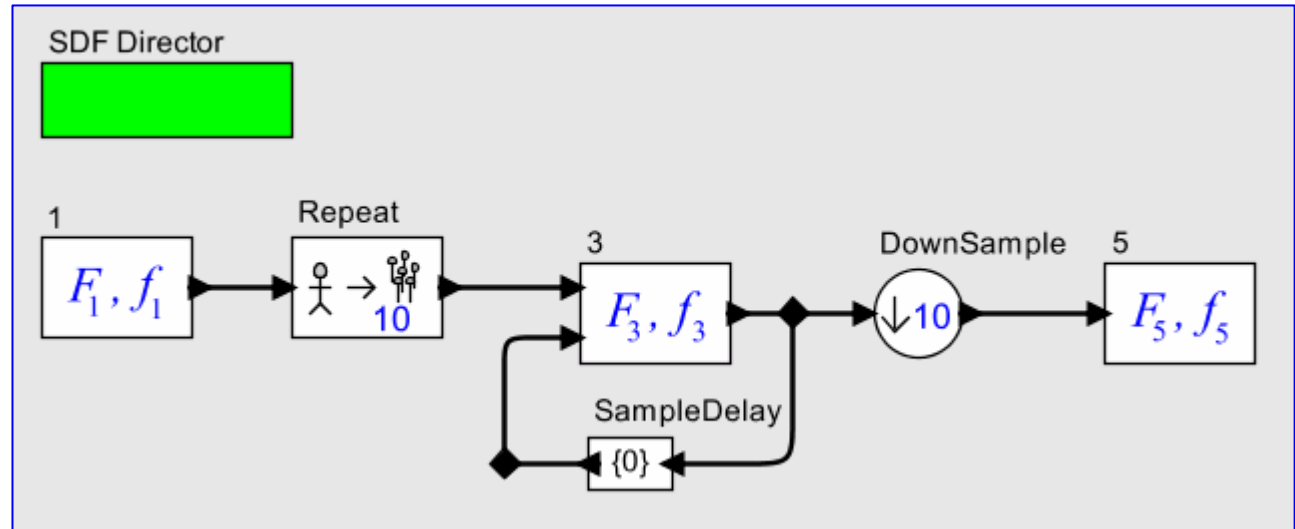
```

ABABCABABCABABCDEAFFFFFFBABCABABCABABCDE
AFFFFFFBCABABCABABCDEAFFFFFFBCABABCABABC
DEAFFFFFFBABCABABCABABCDEAFFFFFFBABCABCA
BABCDEAFFFFFFBCABABCABABCDEAFFFFFFFEBCA
FFFFFFBABCABABCDEAFFFFFFBABCABABCABABCDEAF
FFFFFFBABCABABCABABCDEAFFFFFFBCABABCABABC
DEAFFFFFFBCABABCABABCDEAFFFFFFBABCABABCABCA
BCDEAFFFFFFBABCABABCABABCDEAFFFFFFFEBCAFFFFFFB
ABCABABCABABCDEAFFFFFFBCABABCABABCDEAFFFFFFBA
BCABABCABABCABABCDEAFFFFFFBABCABABCABABCDEAFFF
FFBCABABCABABCABABCDEAFFFFFFBCABABCABABCDEAF
FFFFFFBABCABABCABABCABABCDEAFFFFFFFEBAFFFFFFBABCABC
ABABCDEAFFFFFFBCABABCABABCABABCDEAFFFFFFBCA
BABCABABCDEAFFFFFFBABCABABCABABCABABCDEAFFFFFFB
ABCABABCABABCDEAFFFFFFBCABABCABABCABABCDEAF
FFFFFFBCABABCABABCDEFFFFFFFEFFFFF
  
```

Expressiveness is not as bad as it might seem: e.g. Manifest Iteration in SDF

Imperative
equivalent:

```
while (true) {  
  x = f1();  
  y = 0;  
  for I in (1..10) {  
    y = f3(x, y);  
  }  
  f5(y);  
}
```



Manifest iteration (where the number of iterations is a fixed constant) is expressible in SDF. But data-dependent iteration is not (without the help of some structure).



Variants that Play Different Tradeoffs in Expressiveness

- Structured Dataflow [Kodosky 86, Thies et al. 02]
- (the other) Synchronous Dataflow [Halbwachs et al. 91]
- Cyclostatic Dataflow [Lauwereins 94]
- Multidimensional SDF [Lee & Murthy 96]
- Heterochronous Dataflow [Girault, Lee, and Lee, 97]
- Parameterized Dataflow [Bhattacharya et al. 00]
- Teleport Messages [Thies et al. 05]

All of these remain decidable

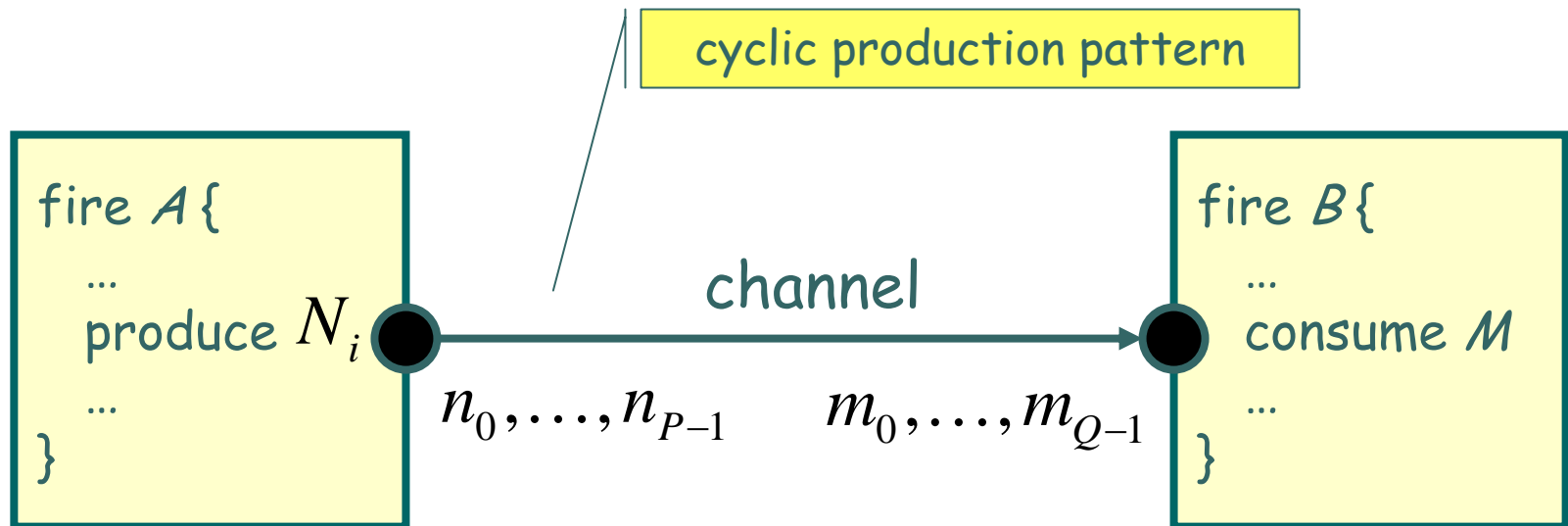
Cyclostatic Dataflow (CSDF)

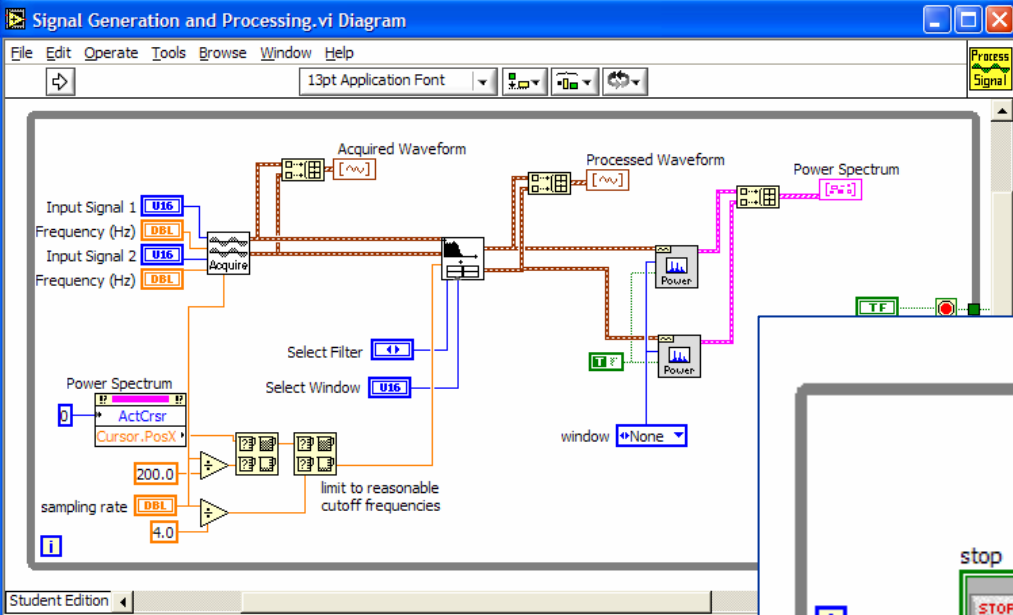
(Lauwereins et al., TU Leuven, 1994)

Actors cycle through a regular production/consumption pattern.

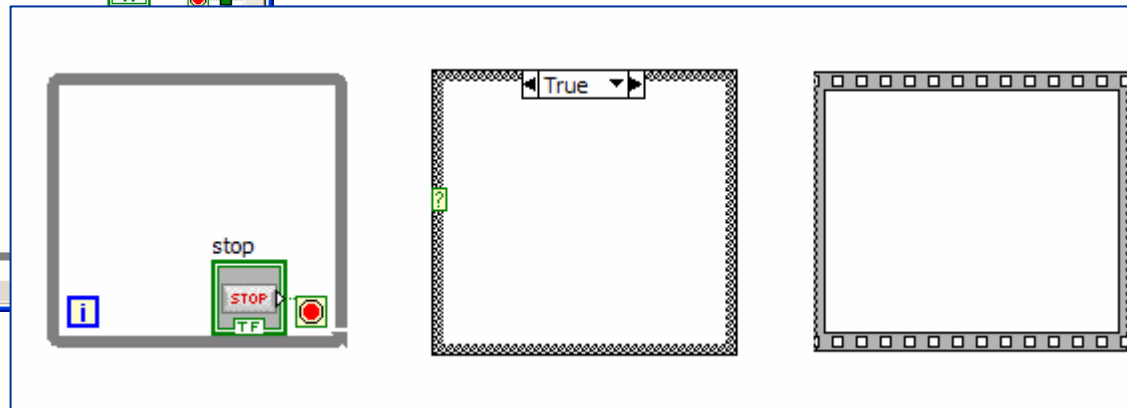
Balance equations become:

$$q_A \sum_{i=0}^{R-1} n_{i \bmod P} = q_B \sum_{i=0}^{R-1} m_{i \bmod Q}; R = lcm(P, Q)$$





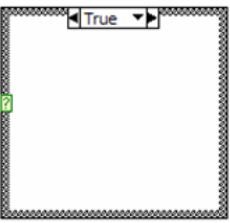
Structured Dataflow [Kodosky 86]



LabVIEW uses homogeneous SDF augmented with syntactically constrained forms of feedback and rate changes:

- While loops
- Conditionals
- Sequences

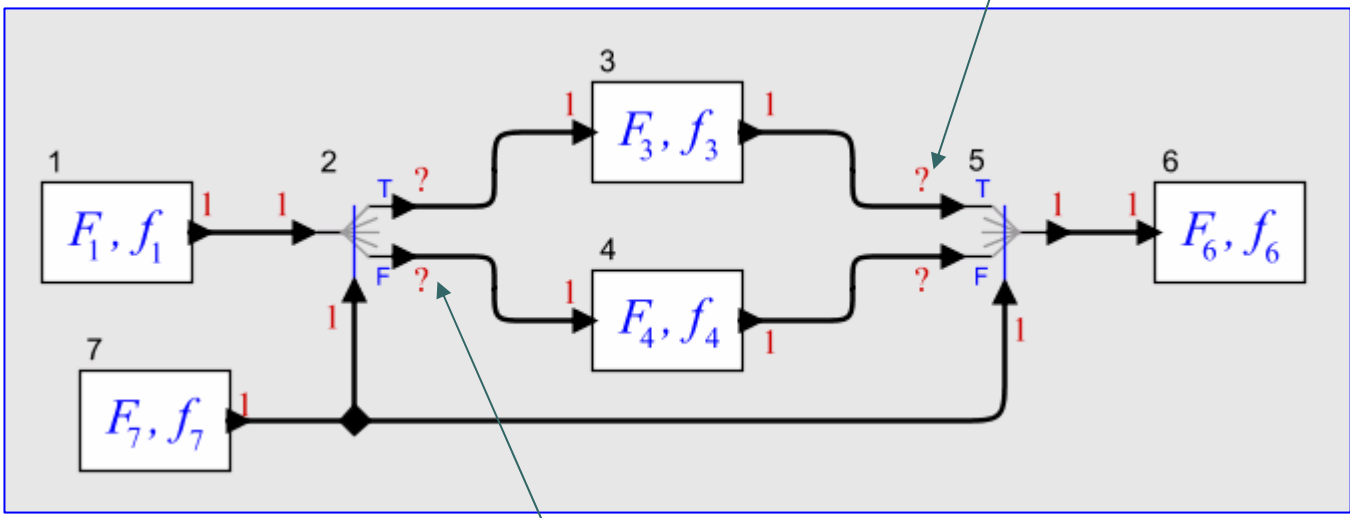
LabVIEW models are decidable.



vs. Dynamic Dataflow, which uses token routing for control flow

Imperative equivalent:

```
while (true) {
  x = f1();
  b = f7();
  if (b) {
    y = f3(x);
  } else {
    y = f4(x);
  }
  f6(y);
}
```



What consumption rate?

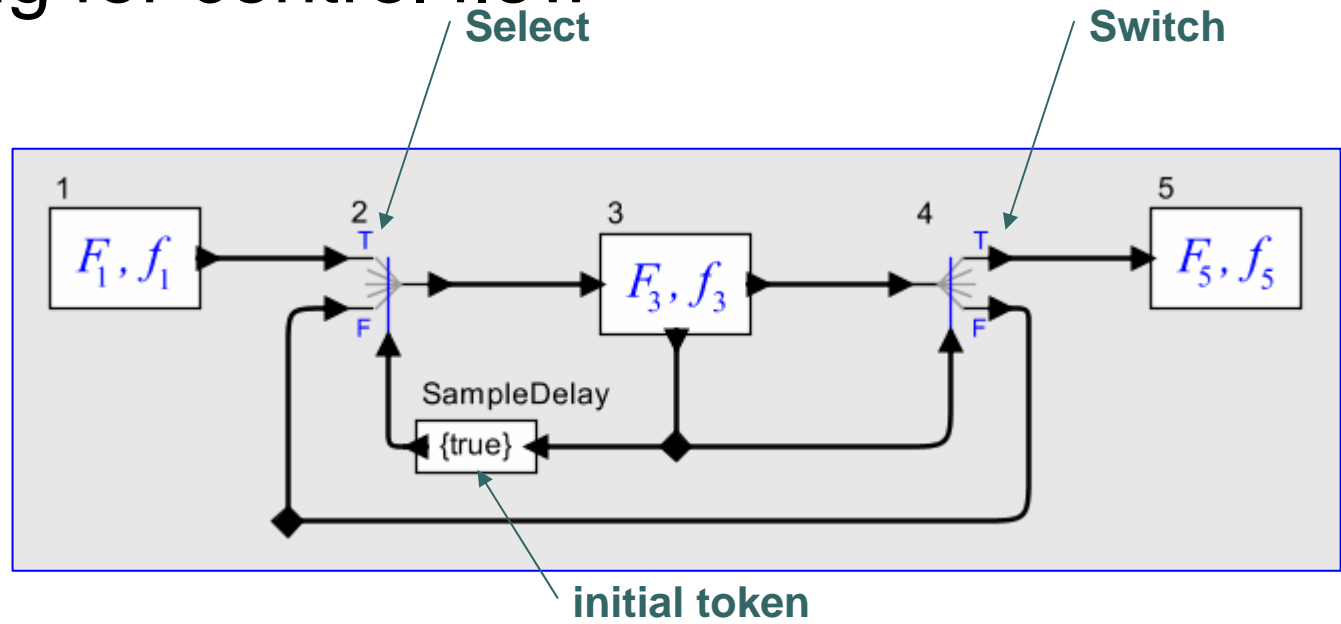
What production rate?

The if-then-else model is not SDF. But we can clearly give a bounded *quasi-static* schedule for it:
(1, 7, 2, b?3, !b?4, 5, 6)

guard



vs. Dynamic Dataflow, which uses token routing for control flow

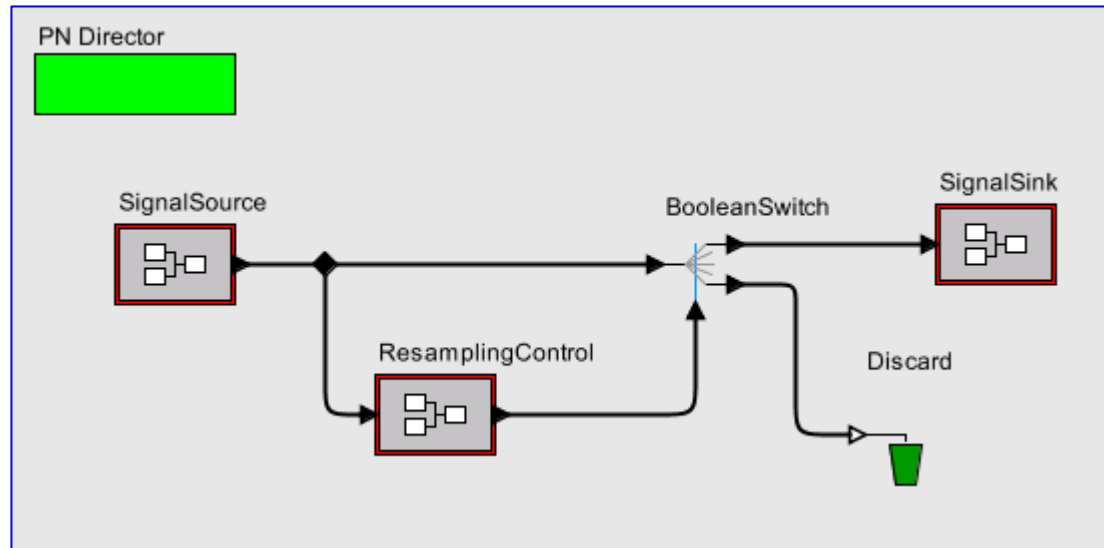


Imperative
equivalent:

```
while (true) {  
    x = f1();  
    b = false;  
    while(!b) {  
        (x, b) = f3(x);  
    }  
    f5(x);  
}
```

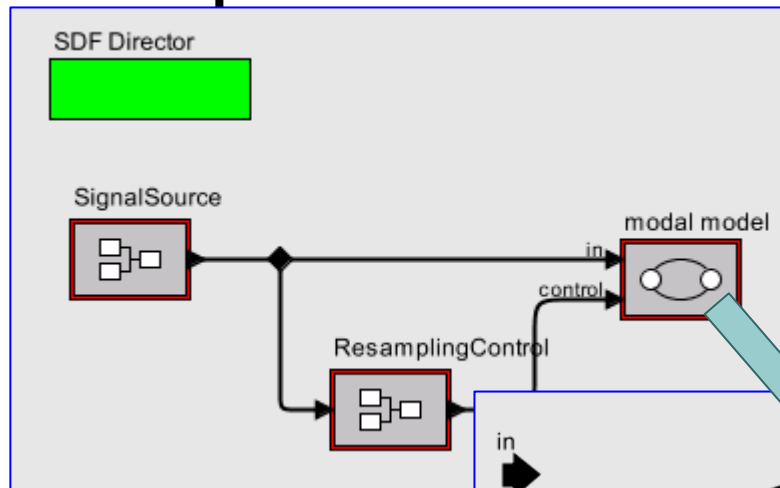
This model uses conditional routing of tokens to iterate a function a data-dependent number of times.

Application of Dynamic Dataflow: Resampling of Streaming Media

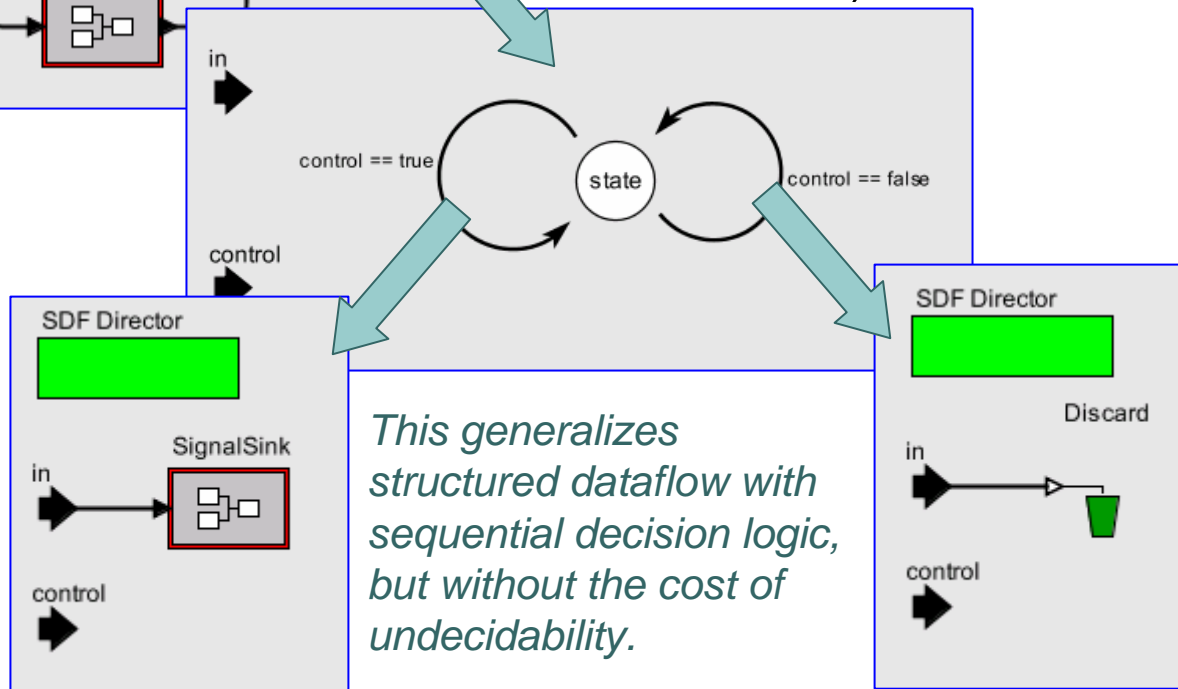


- This pattern requires the use of a semantically richer dataflow model than SDF because the BooleanSwitch is not an SDF actor.
- This has a performance cost and reduces the static analyzability of the model.

Resampling Design Pattern using Modal Models

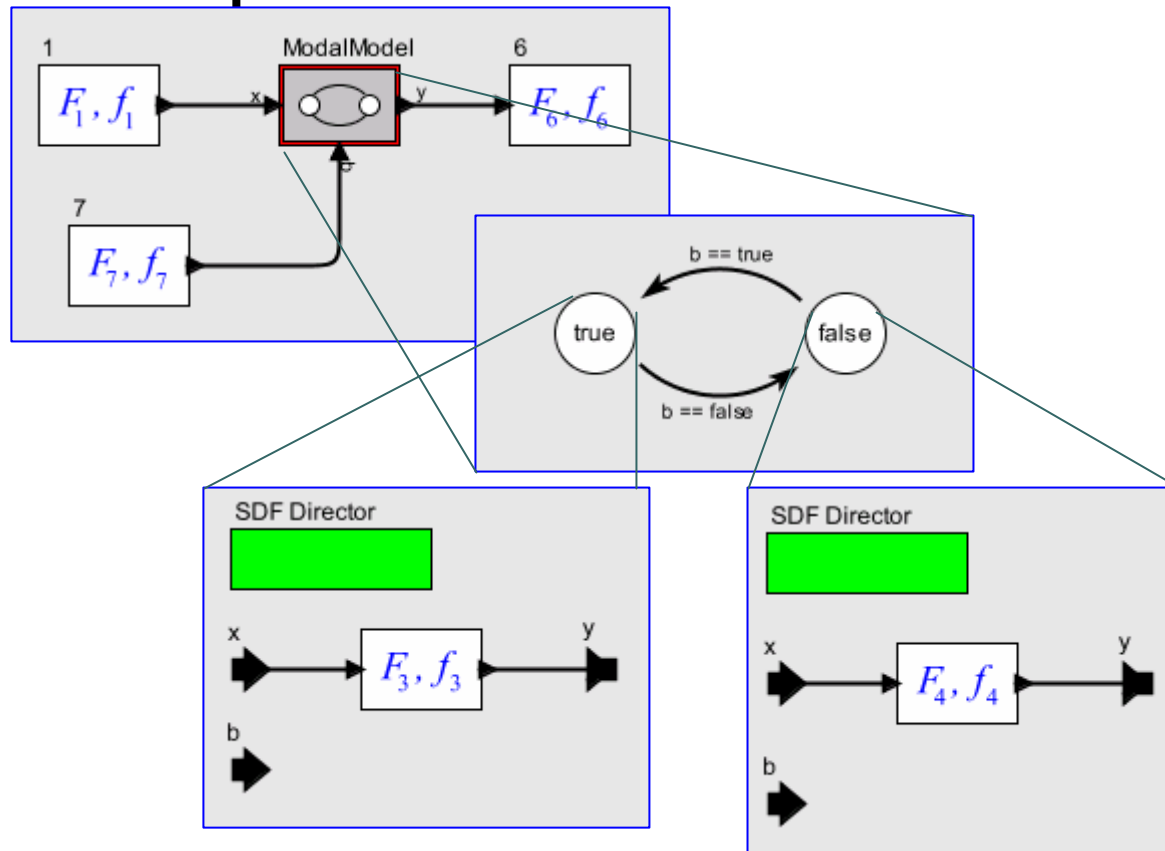


Hierarchically mixing synchronous dataflow with finite state machines offers a much more powerful model of computation than either alone. And everything remains decidable (if you are careful)!



This generalizes structured dataflow with sequential decision logic, but without the cost of undecidability.

Generalization on the Edge of Undecidability: Heterochronous Dataflow (HDF) [Girault, Lee, Lee, 97]



Rough imperative equivalent:

```
b = true;
while (true) {
    x = f1();
    if (b) {
        y = f3(x);
    } else {
        y = f4(x);
    }
    f6(y);
    b = f7();
}
```

Semantics of HDF:

- Execute SDF model for one complete iteration
- Take state transitions to get a new SDF model.

HDF is on the edge of the tradeoff between expressiveness and decidability of static analysis questions (boundedness, deadlock, scheduling).

Syntax: Graphical or Textual?

component technologies

- Sutherland (66)

- Prograph (85)

- LabVIEW (86)

- Gabriel (86)

- Show and Tell (86)

- Cantata (91)

- Ptolemy Classic (94)

- Ptolemy II (00)

- Scade (05)

- ...

- Lucid (77)

- Id (78)

- VAL (79)

- Sisal (83)

- Lustre (86)

- Signal (90)

- Granular Lucid (95)

- StreamIT (02)

- Cal (03)

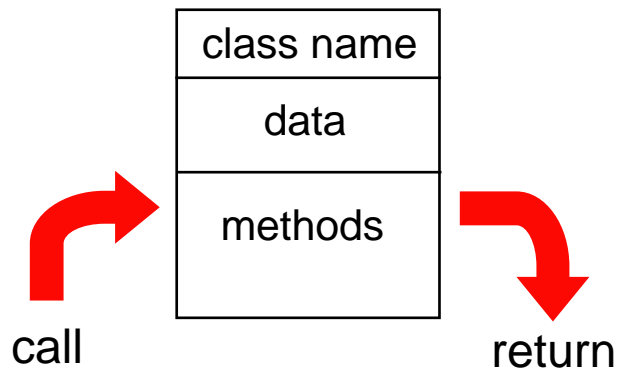
- ...

The graphical vs. textual debate obscures a more important question:

Are actors and streams a programming language technology or a software component technology?

Actors and Streams as a Software Component Technology

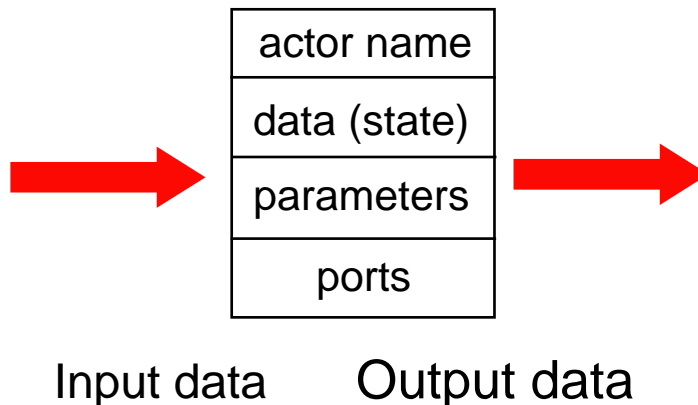
The established: Object-oriented:



What flows through an object is sequential control

Things happen to objects

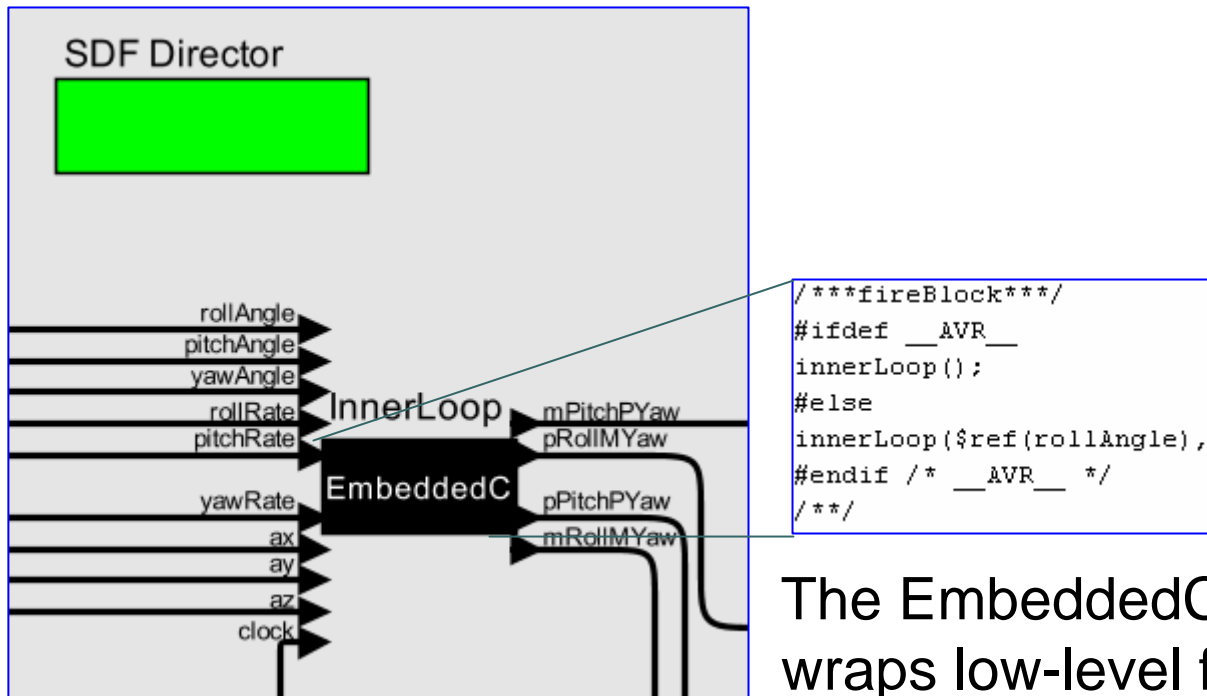
The alternative: Actor oriented:



Actors make things happen

What flows through an object is evolving data

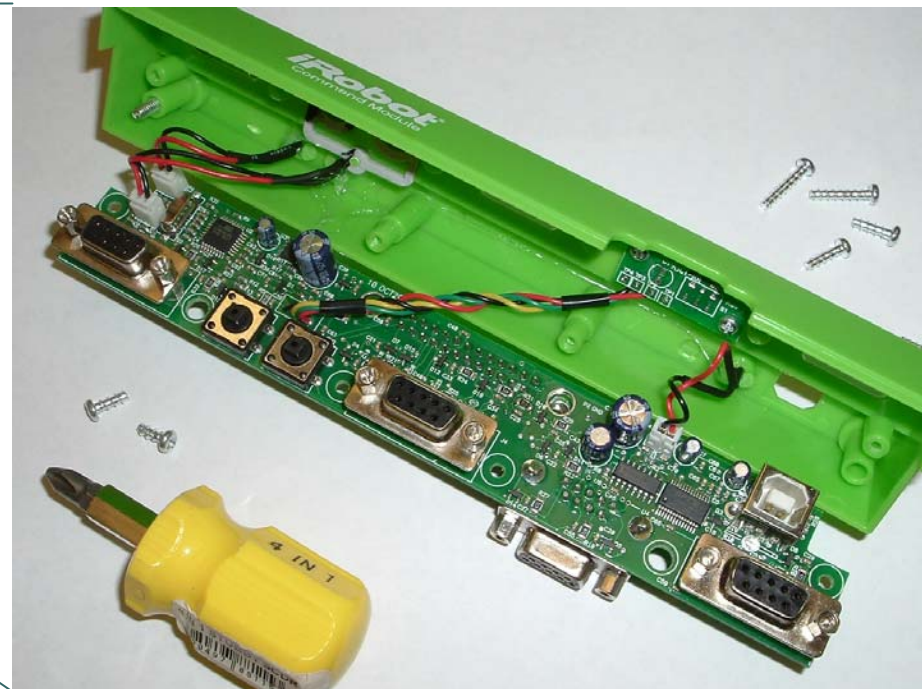
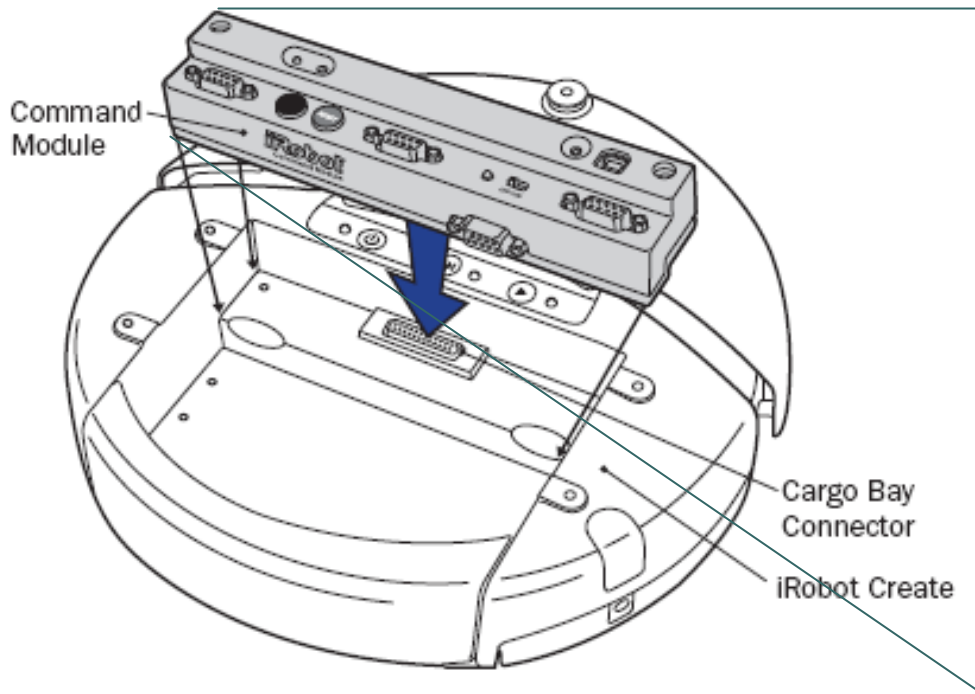
Actors as Components



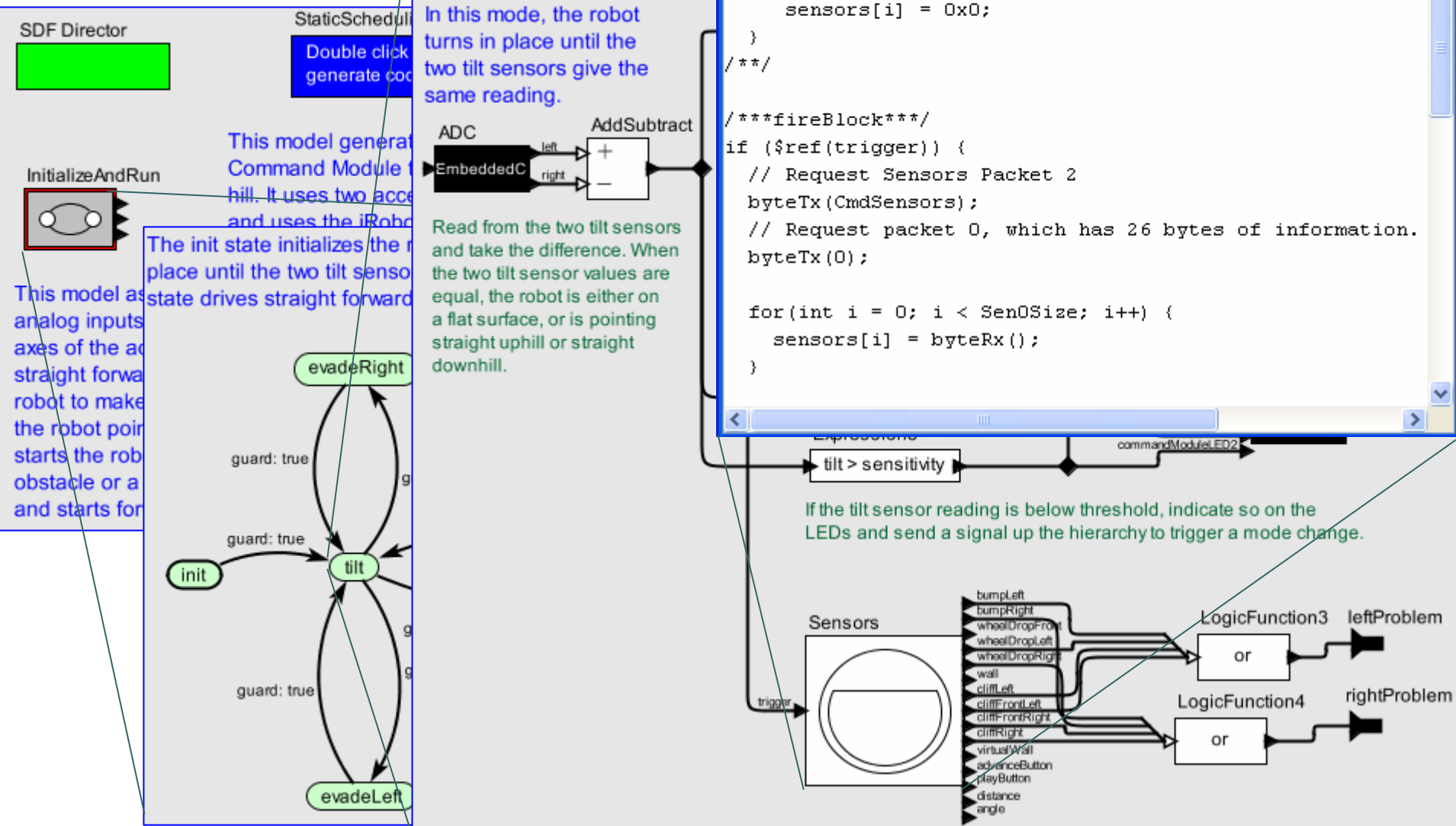
The **EmbeddedC** actor in Ptolemy II wraps low-level functionality (written in C) to define an actor. This approach makes it easy to build actor-oriented models and to generate efficient, platform-specific C implementations.

Example Bringing it all Together: Embedded Target

Programming the iRobot Create (the platform for the Roomba vacuum cleaner) with a pluggable Command Module containing an 8-bit Atmel microcontroller.



This design of a hill-climbing control algorithm wraps code provided by iRobot as demo code into actors in Ptolemy II for accessing sensors and actuators.





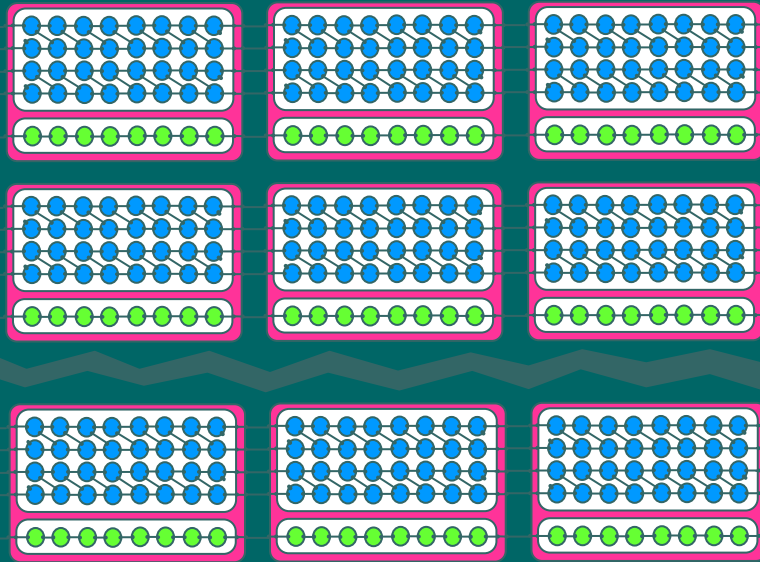
Some of the Points I've Made

- Streams and actors have subtle issues
- They can be used for languages or components
- Token routing for control is like unstructured goto
- State machines combined with dataflow offer an interesting foundation for language and/or component architecture design.

My opinion: The time is right for streams to take off (again?). The challenge is in the pragmatics of language design and component architectures.

Scalability is less well developed with visual syntaxes. but do not confuse immaturity of development with conceptual flaws.

Pragmatics: Scalable Composition Languages Big Systems with Small Descriptions



We have released a specification language that we call “Ptolon” for such systems, integrated into Ptolemy II.

```
System is {  
    Matrix(Component(2),20,3);  
}
```

```
Component is {  
    param n;  
    port in[n*2+1];  
    port out[n*2+2];  
} in {  
    Blue(n, in[1..n*2],  
          out[1..n*2]);  
  
    Green(n, in[n*2+1],  
           out[n*2+1]);  
}
```