ECSE 421 Lecture 4: Data Flow Modeling

ESD Chapter 2

Last Time

- StateCharts
  - Specification using …
  - Hierarchical states manage …
  - Timing with …
  - Determinate?
- Synchronous Languages
  - Determinate?
- Specification and Description Language
  - Determinate?

Where Are We?

W	D	Date		Topic	ESD	PES	Out	In	Notes
1	T	12-Jan-2016	L01	Introduction to Embedded System Design	1.1-1.4				
	R	14-Jan-2016		Introduction to Embedded System Design	1.1-1.4				
2	T	19-Jan-2016	L02	Specifying Requirements / MoCs / MSC	2.1-2.3				
	R	21-Jan-2016	L03	CFSMs	2.4				
3	T	26-Jan-2016	L04	Data Flow Modeling	2.5	3.1-5,7			
	R	28-Jan-2016	L05	Petri Nets	2.6				
4	T	2-Feb-2016	L06	Discrete Event Models	2.7	4			Guest lecturer
	R	4-Feb-2016	L07	DES / Von Neumann Model of Computation	2.8-2.10	5			
5	T	9-Feb-2016	L08	Sensors	3.1-3.2	7.3,12.1-6			
	R	11-Feb-2016	L09	Processing Elements	3.3	12.6-12			
6	T	16-Feb-2016	L10	More Processing Elements / FPGAs					
	R	18-Feb-2016	L11	Memories, Communication, Output	3.4-3.6				
7	T	23-Feb-2016	L12	Embedded Operating Systems	4.1				
	R	25-Feb-2016		***Midterm exam: in-class, closed book***					Chapters 1-3
	T	1-Mar-2016		***No class***					Winter break
	R	3-Mar-2016		***No class***					Winter break
8	T	8-Mar-2016	L13	Middleware	4.4-4.5				
	R	10-Mar-2016	L14	Performance Evaluation	5.1-5.2				
9	T	15-Mar-2016	L15	More Evaluation and Validation	5.3-5.8				
	R	17-Mar-2016	L16	Introduction to Scheduling	6.1-6.2.2				
10	T	22-Mar-2016	L17	Scheduling Aperiodic Tasks	6.2.3-6.2.4				
	R	24-Mar-2016	L18	Scheduling Periodic Tasks	6.2.5-6.2.6				

Today

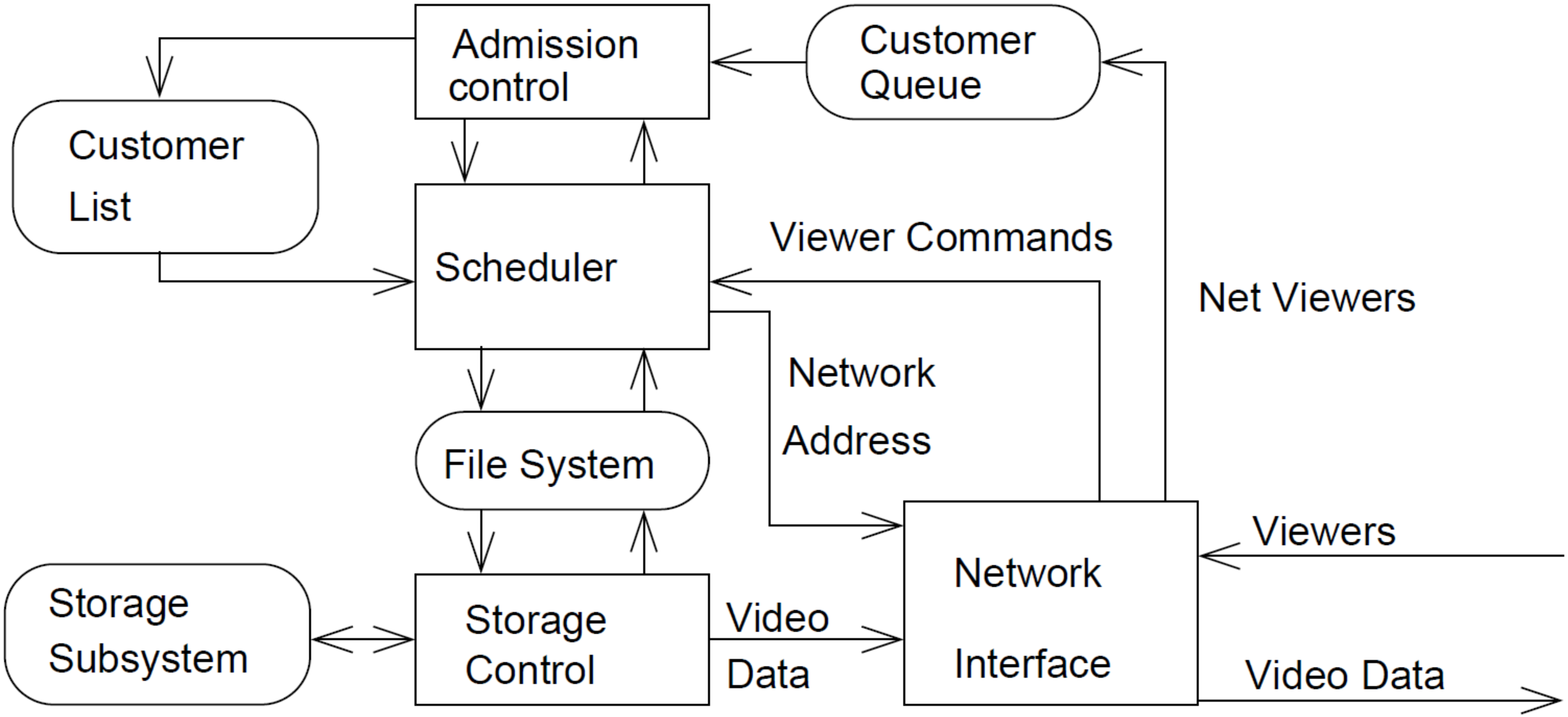
- Data Flow Modeling
  - Kahn Process Networks
  - Synchronous Data Flow
  - Simulink

MoCs Considered in 421

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Undefined components	Plain text, use cases | (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow	**(Not useful)**		**Kahn networks, SDF**
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	VHDL*, Verilog*, SystemC*, …	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

Data Flow

- A "natural" model of applications
- Example: Video-on-demand system



http://www.ece.ubc.ca/~irenek/techpaps/vod/vod.html

Data Flow Modeling

Definition: Data flow modeling is … “*the process of identifying, modeling and documenting how data moves around an information system.*

Data flow modeling examines

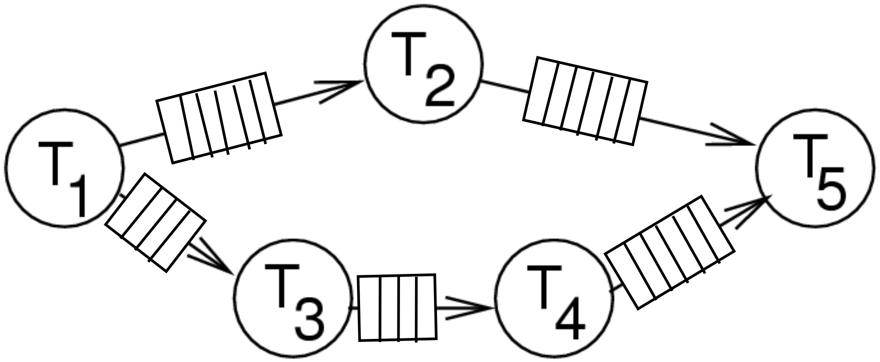
- *processes (activities that transform data from one form to another),*
- *data stores (the holding areas for data),*
- *external entities (what sends data into a system or receives data from a system), and*
- *data flows (routes by which data can flow).”*

[Wikipedia: Structured systems analysis and design method. http://en.wikipedia.org/wiki/Structured Systems Analysis and Design Methodology, 2010 (formatting added)]

Kahn Process Networks (KPN)

- A *restricted* data flow model
  - Hard to prove things about unrestricted data flows
- Each component is a program/task/process
  - Not an FSM
- Communication through FIFOs

- Overflow not considered, like SDL
  - Writes never have to wait
  - Reads wait if the FIFO is empty
- Only one sender and one receiver
  - Unlike SDL
  - No conflicts at inputs!



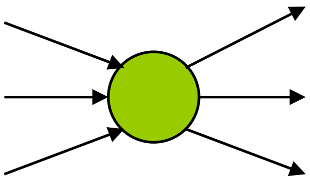
McGill

Example

```
Process f(in int u, in int v, out int w) {
    int i; bool b = true;
    for (;;) {
    // wait returns next token in FIFO, waits if empty
        i = b ? wait(u) : wait(v);
    // send writes a token into a FIFO w/o blocking
        send (i,w);
        b = !b;
    }
}
```

Properties of KPN

- Communication is via channels only
- Mapping from ≥1 input channel
- Mapping to ≥1 output channel

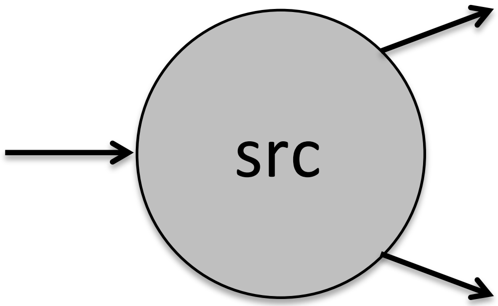


- Channels transmit information within
  - An *unpredictable*, but
  - *Finite* amount of time
- Execution times are generally unknown
  - Though this reflects the state of designs in the early stages of specification

Another Example

- Process a stream of integers
  - Separate integers them into odd and even streams
  - Increment each
  - Close the loop and re-separate and re-process every processed integer
- Four processes
  - Source
    - Generates initial data
    - Divides data
  - Even, Odd—increment the data
  - Sink—merges data and forwards back to Source

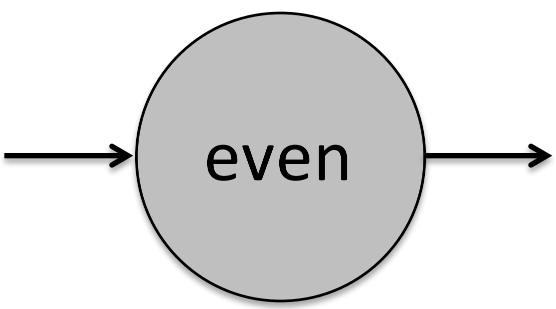
Processes



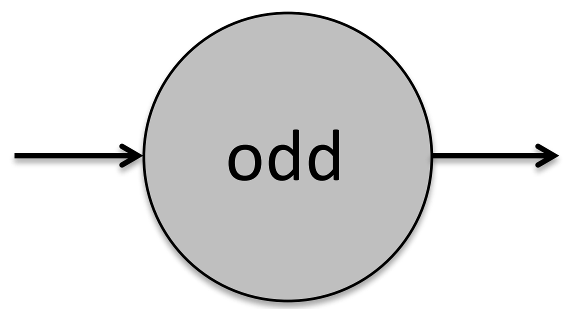

```
Process src(
            in int feedback,
            out int oddIn,
            out int evenIn) {
    int i;
    i = 0; send(i, evenIn);
    i = 1; send(i, oddIn);
    for (;;) {
        i = wait(feedback);
        i % 2 == 0 ? send(i, evenIn) : send(i, oddIn);
    }
}
```

Processes, Cont'd

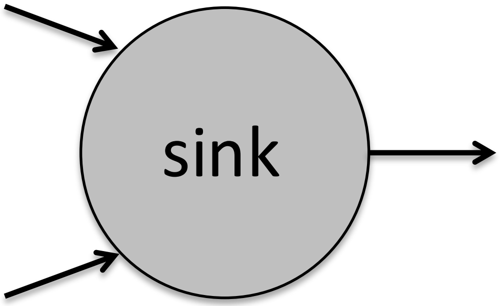
```
Process even(
        in int evenIn,
        out int evenOut) {
    int i;
    for (;;) {
        i = wait(evenIn);
        i = i+1;
        send(i, evenOut);
    }
}
```




```
Process odd(
        in int oddIn,
        out int oddOut) {
    int i;
    for (;;) {
        i = wait(oddIn);
        i = i+1;
        send(i, oddOut);
    }
}
```

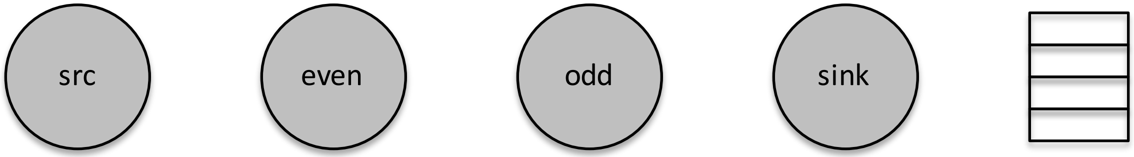


Processes, Cont'd




```
Process sink(
                in int outOdd,
                in int evenOut,
                out int feedback) {
    int i;
    bool even = true;
    for (;;) {
        even ? i = wait(evenOut) : i = wait(oddOut);
        send(i, feedback);
        even = !even;
    }
}
```

Now Assemble the KPN



Key Advantages of KPNs

- No polling
  - Processes read and block if a FIFO is empty
- No blocking on multiple FIFOs
- Therefore, operation sequence depends on order of reads
- Therefore, order of reads depends only on order of data
  - Not on its arrival time!
- Therefore, Kahn Process Networks are determinate
  - For a given input, the result will always the same
  - Regardless of the speed of the nodes!
- This has many applications in embedded system design: ***any*** *combination of fast and slow simulation and hardware prototypes* ***always*** *gives the same result*

Computational Power and Analyzability

- KPNs are Turing-complete
  - Anything that can be computed can be computed by a KPN
- It is a challenge to schedule KPNs without accumulating tokens
  - Since no assumptions are made about the speed of channels or nodes
- KPNs are computationally powerful, but difficult to analyze
  - *E.g.* what's the maximum buffer size?
- Number of processes is static (cannot change)

Further Reading on KPNs

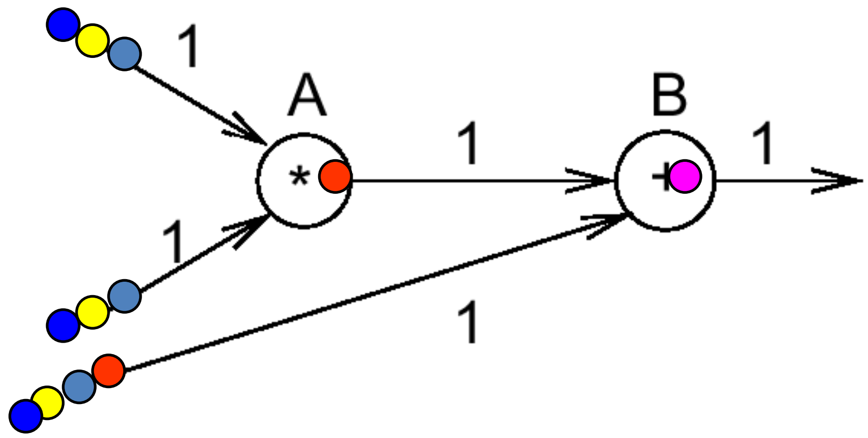
- http://ls12-www.cs.tu-dortmund.de/daes/en/lehre/downloads/levi.html
  - Animations are available
  - Website is in German (but Chrome translates it well)
- http://en.wikipedia.org/wiki/Kahn_process_networks
- See also S. Edwards
  - http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/presentations/dataflow.ppt

SDF

- Synchronous data flow (SDF)
  - Less computationally powerful
  - Easier to analyze
- Synchronous
  $\Rightarrow$ global clock controlling “firing” of nodes
- Again using asynchronous message passing
  $\Rightarrow$ tasks don’t block on writes

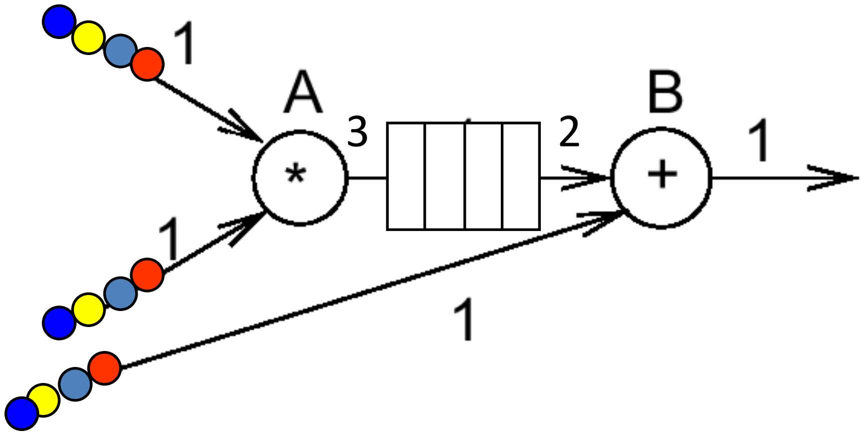
(Homogeneous-) SDF

- Nodes are called *actors*
- Actors are *ready* …
  - If the necessary number of input tokens exists, and
  - If enough buffer space at the output exists
- Ready actors can *fire* (be executed)
- Execution takes a *fixed, known time*



(Non-homogeneous-) SDF (1)

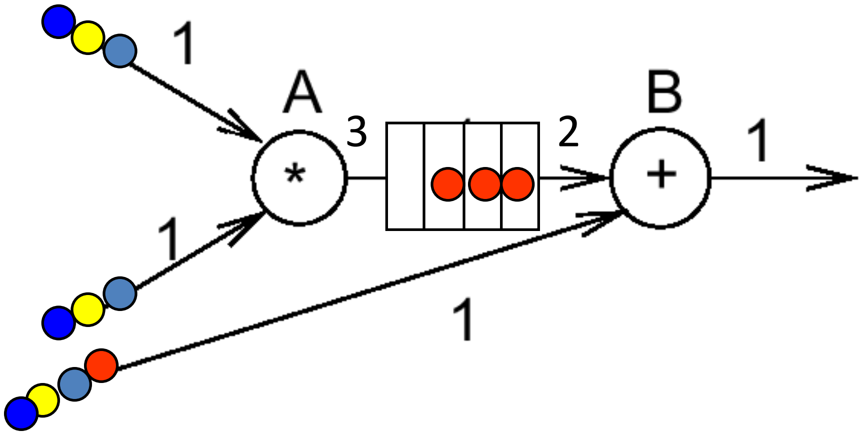
- In the general case, any number of tokens can be produced/consumed per firing
  - Firing rate depends on # of tokens …



A ready, can fire

(Non-homogeneous-) SDF (2)

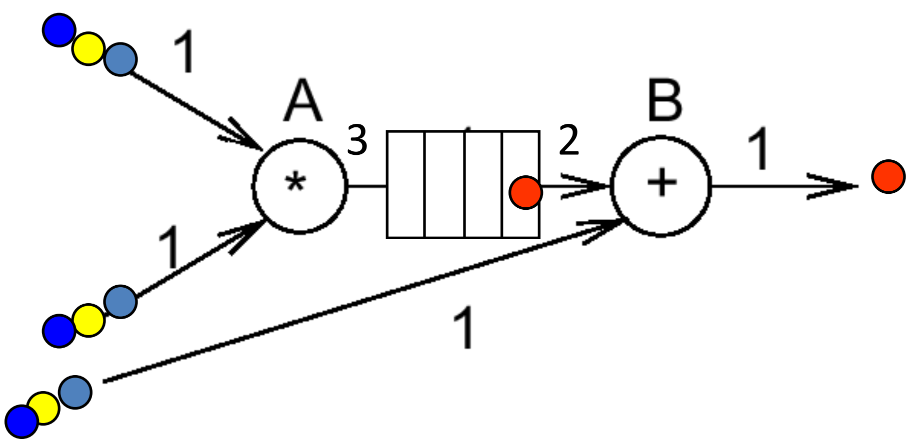
- In the general case, any number of tokens can be produced/consumed per firing
  - Firing rate depends on # of tokens …



B ready, can fire

(Non-homogeneous-) SDF (3)

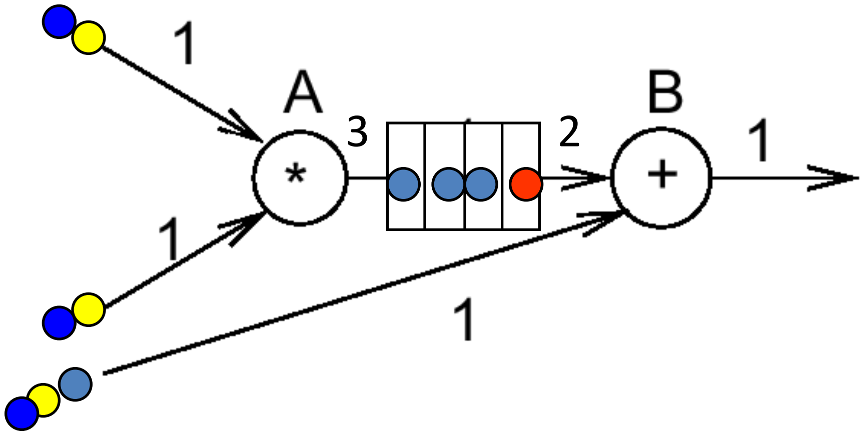
- In the general case, any number of tokens can be produced/consumed per firing
  - Firing rate depends on # of tokens …



A ready, can fire

(Non-homogeneous-) SDF (4)

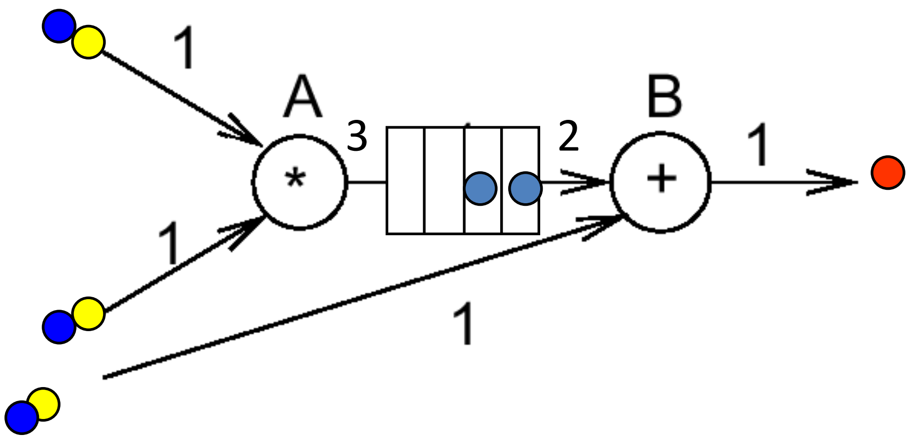
- In the general case, any number of tokens can be produced/consumed per firing
  - Firing rate depends on # of tokens …



B ready, can fire

(Non-homogeneous-) SDF (5)

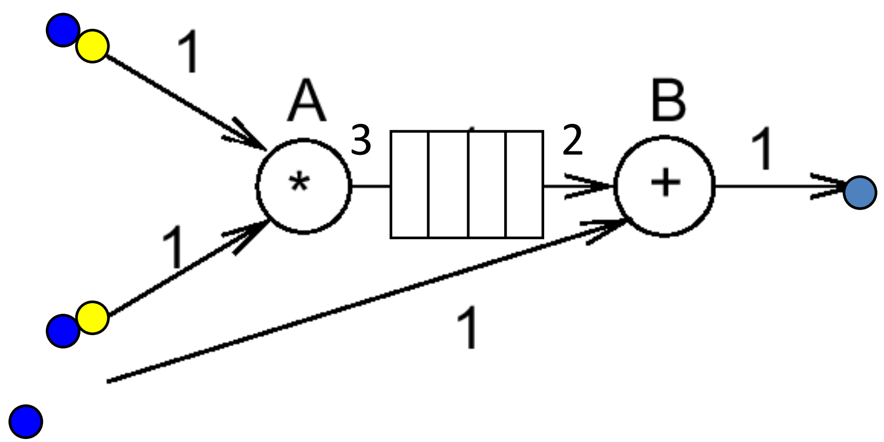
- In the general case, any number of tokens can be produced/consumed per firing
  - Firing rate depends on # of tokens …



B ready, can fire

(Non-homogeneous-) SDF (6)

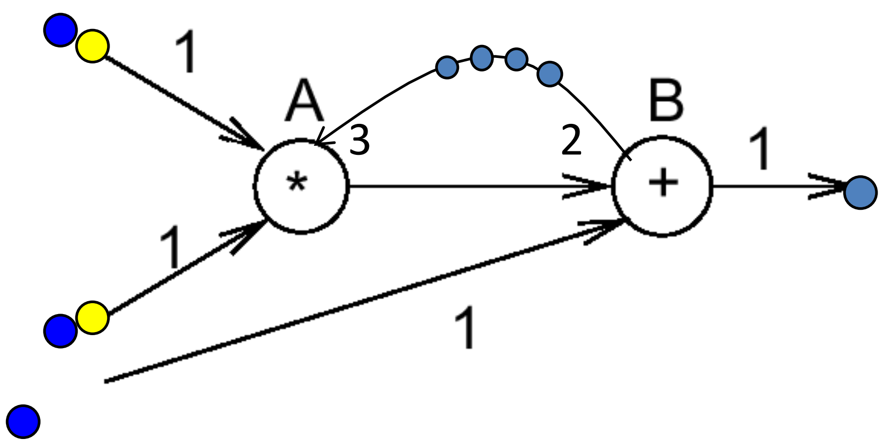
- In the general case, any number of tokens can be produced/consumed per firing
  - Firing rate depends on # of tokens …



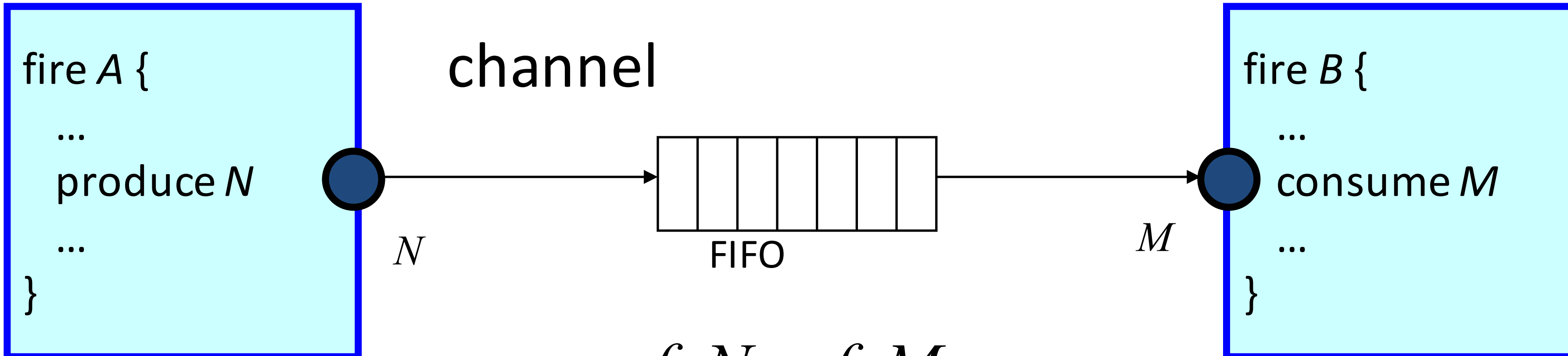
A ready, can fire

Modeling Buffer Capacity

- Buffer capacity can be modeled more easily
  - Use a ***backward*** edge
  - Initial number of tokens = buffer capacity



Multi-rate Models and Balance Equations



$$f_A N = f_B M$$

number of tokens consumed

number of tokens produced

number of firings per “iteration”

Schedulable statically.

In the general case, buffers may be needed at edges.

Decidable:

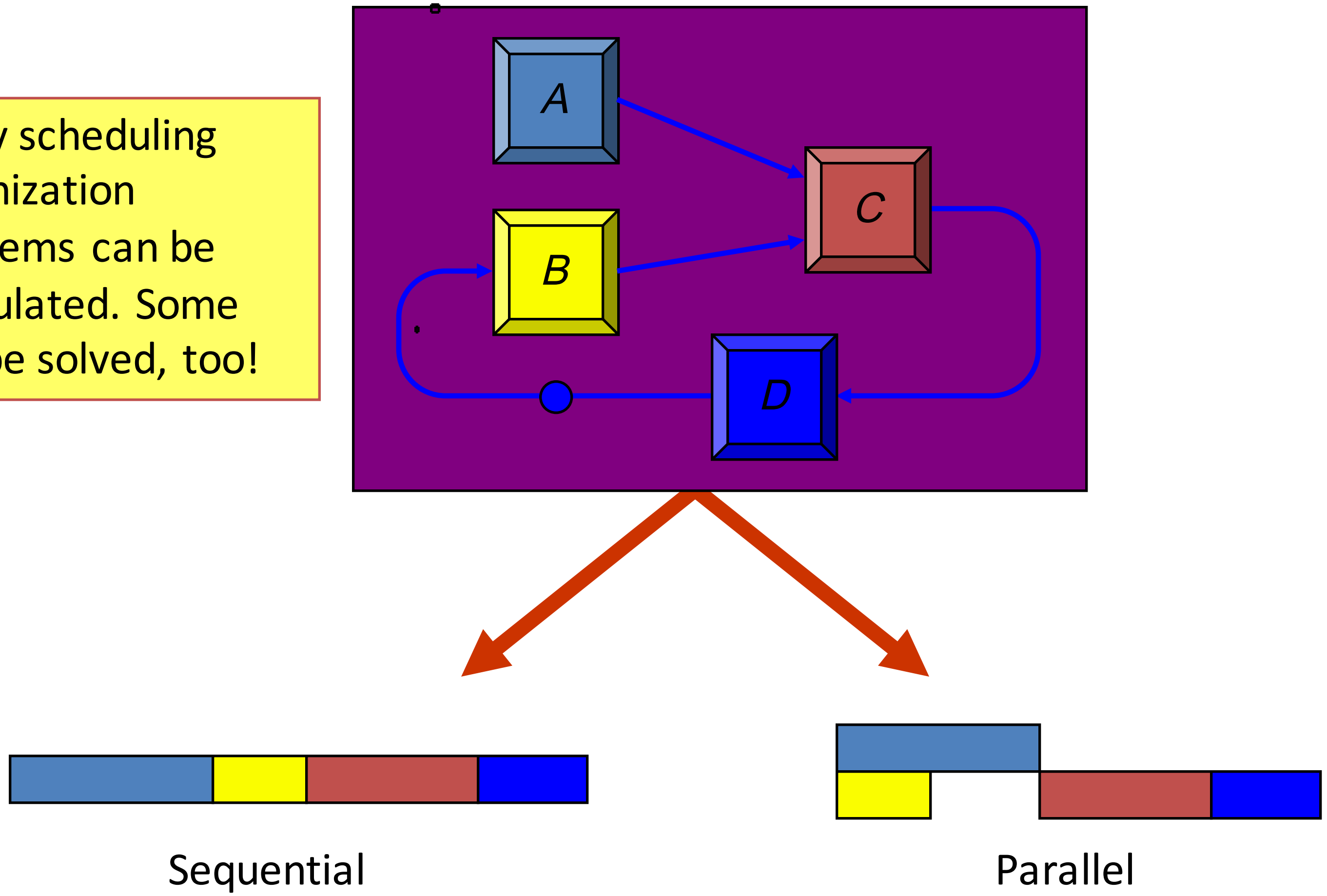
- buffer memory requirements
- deadlock

Adopted from: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

Parallel Scheduling of SDF Models

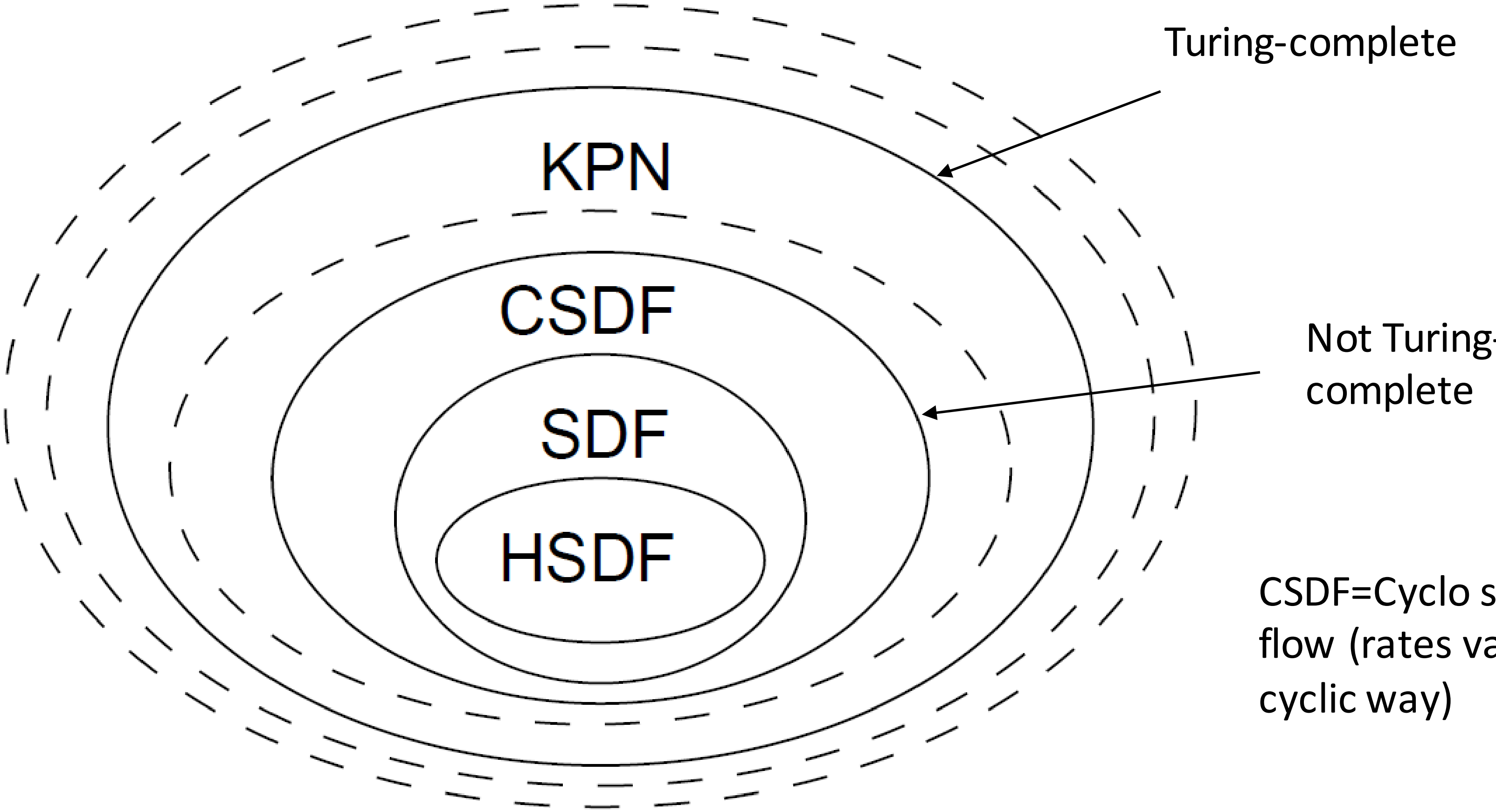
- SDF is suitable for automated parallel system design
  - Automated mapping onto parallel processors; synthesis of parallel circuits

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



Adapted from: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

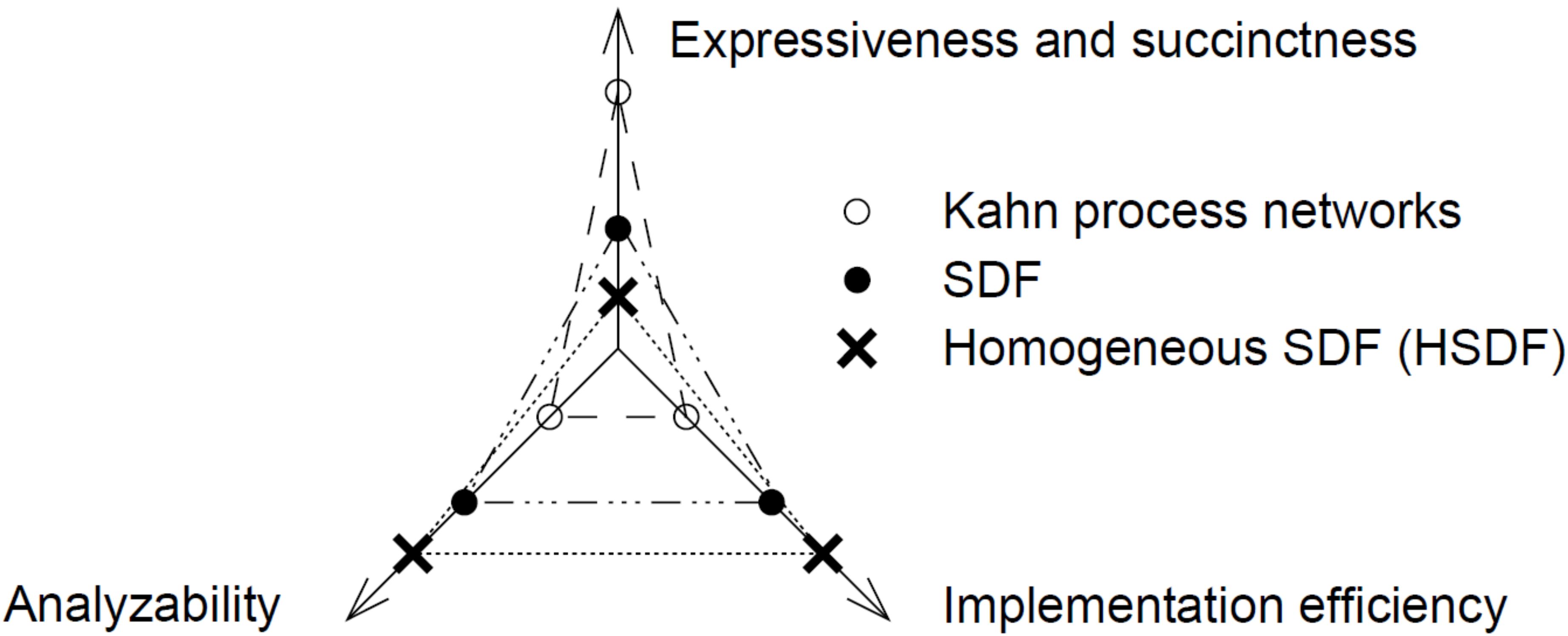
Expressiveness of data flow MoCs



CSDF=Cyclo static data flow (rates vary in a cyclic way)

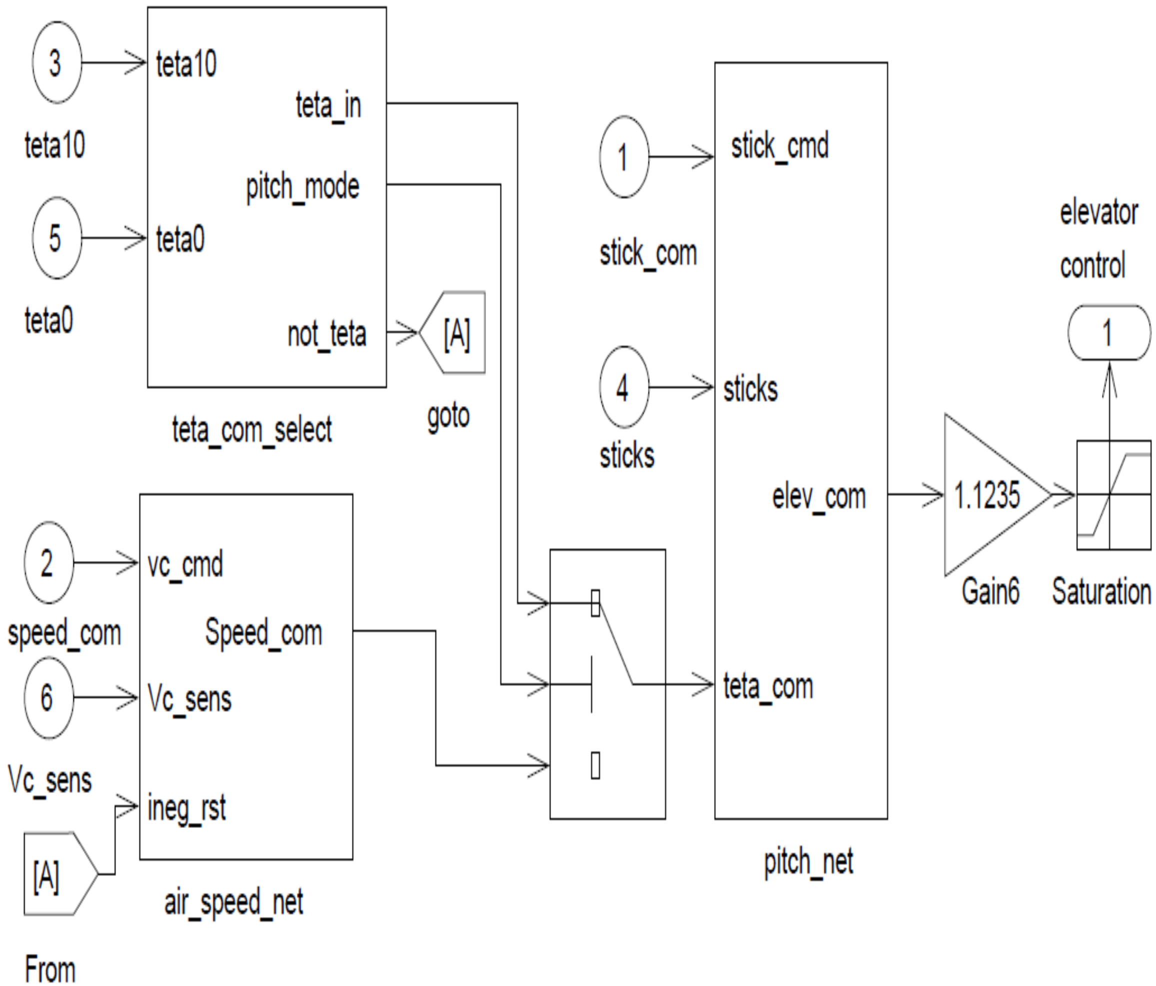
[S. Stuijk, 2007]

The Expressiveness/Analyzability Conflict



[S. Stuijk, 2007]

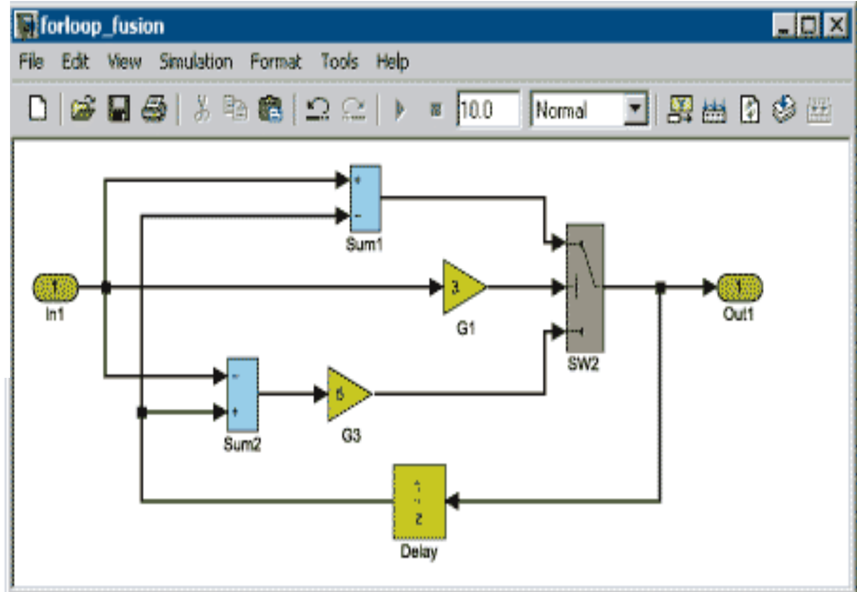
Similar MoC: Simulink



Semantics? *"Simulink uses an idealized timing model for block execution and communication. Both happen infinitely fast at exact points in simulated time. Thereafter, simulated time is advanced by exact time steps. All values on edges are constant in between time steps."* [Nicolae Marian, Yue Ma]

[mathworks]

Starting Point for "Model-based Design"

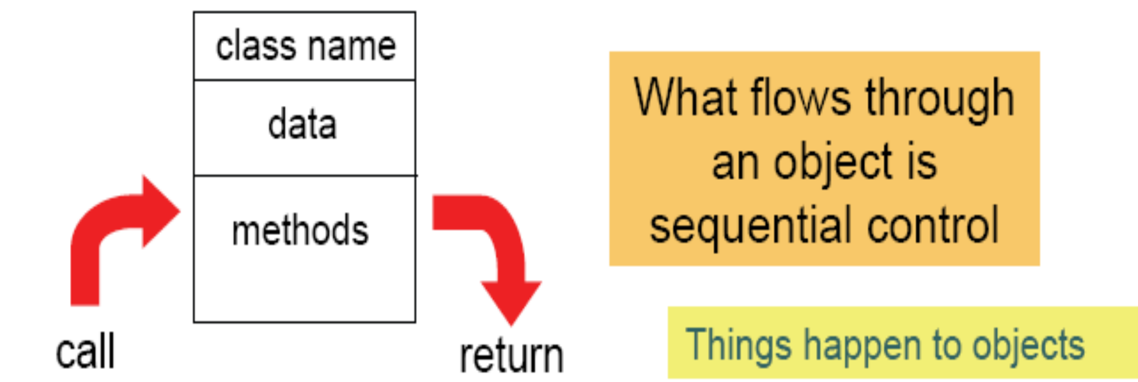



```
/* Switch: '<Root>/SW2' incorporates:
 *   Sum: '<Root>/Sum1'
 *   Gain: '<Root>/G1'
 *   Sum: '<Root>/Sum2'
 *   Gain: '<Root>/G3'
 */
for(i1=0; i1<10; i1++) {
  if(rtU.In1[i1] * 3.0 >= 0.0) {
    rtb_SW2_c[i1] = rtU.In1[i1] - rtDWork.Delay_DSTATE[i1];
  } else {
    rtb_SW2_c[i1] = (rtDWork.Delay_DSTATE[i1] - rtU.In1[i1]) * 5.0;
  }

  /* Outport: '<Root>/Out1' */
  rtY.Out1[i1] = rtb_SW2_c[i1];

  /* Update for UnitDelay: '<Root>/Delay' */
  rtDWork.Delay_DSTATE[i1] = rtb_SW2_c[i1];
}
```

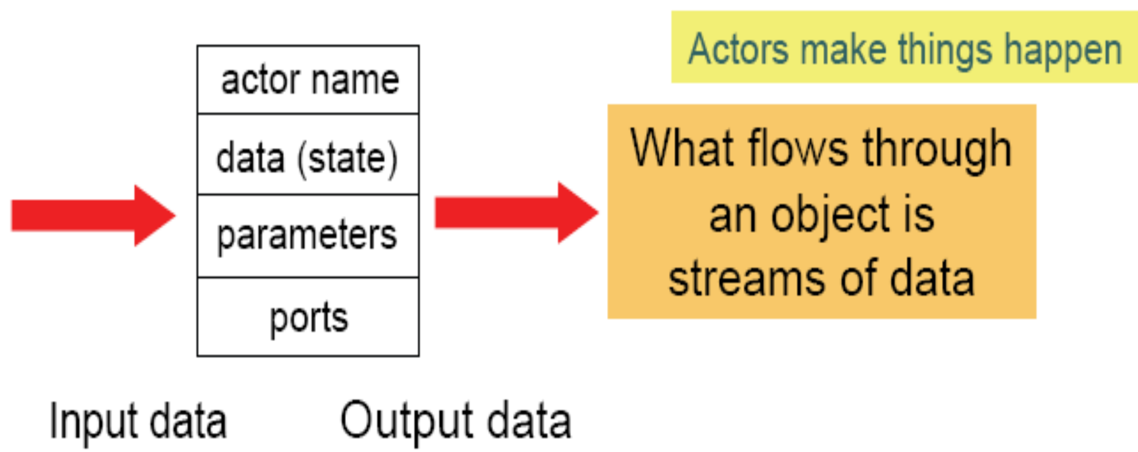
- Build a model in Simulink
- Code automatically generated (C or HDL)

SDF and Simulink are Actor Languages

The established: Object-oriented:



The alternative: Actor oriented:



Summary

- Data Flow Models
  - Must be restricted to be useful
  - Kahn Process Networks
    - Turing-complete; determinate!
  - Synchronous Data Flow
    - Less expressive; easier to analyze!
- Visual programming languages
  - Simulink: compile your model into C or HDL!

Next Time

- Petri Nets
  - Chapter 2.6