technische universität
dortmund

fakultät für informatik
informatik 12

Finite state machines + message passing: SDL

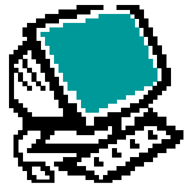
Peter Marwedel
TU Dortmund,
Informatik 12

2010/10/20

Context

Communication/ Computation	Shared memory	Message passing	
		blocking	Non-blocking
FSM	StateCharts		SDL

SDL used here as a (prominent) example of a model of computation based on **asynchronous message passing communication**.



☞ appropriate also for distributed systems

SDL

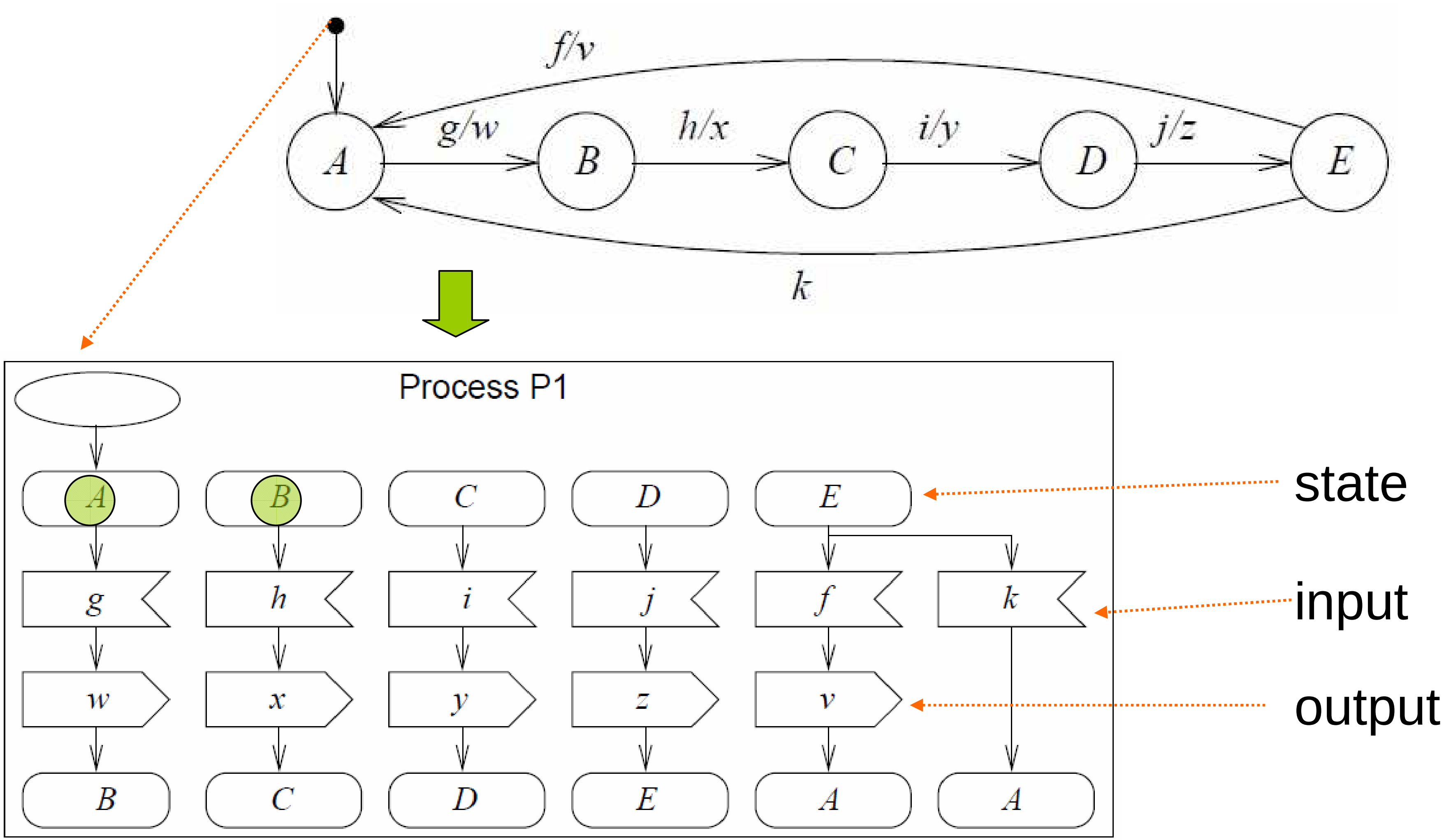
Language designed for specification of distributed systems.

- Dates back to early 70s,
- Formal semantics defined in the late 80s,
- Defined by ITU (International Telecommunication Union):
  Z.100 recommendation in 1980
  Updates in 1984, 1988, 1992, 1996 and 1999

SDL

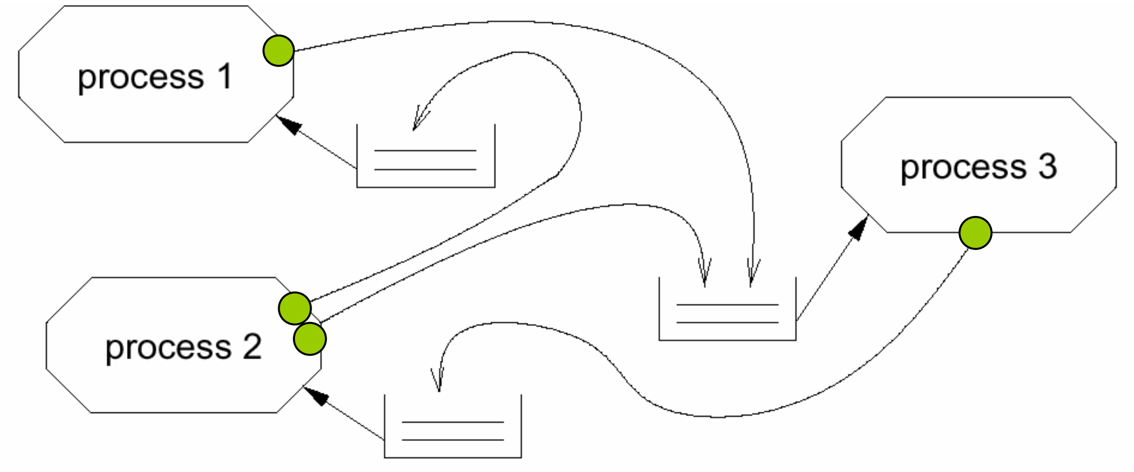
- Provides textual and graphical formats to please all users,
- Just like StateCharts, it is based on the CFSM model of computation; each FSM is called a **process**,
- However, it uses message passing instead of shared memory for communications,
- SDL supports operations on data.

SDL-representation of FSMs/processes



Communication among SDL-FSMs

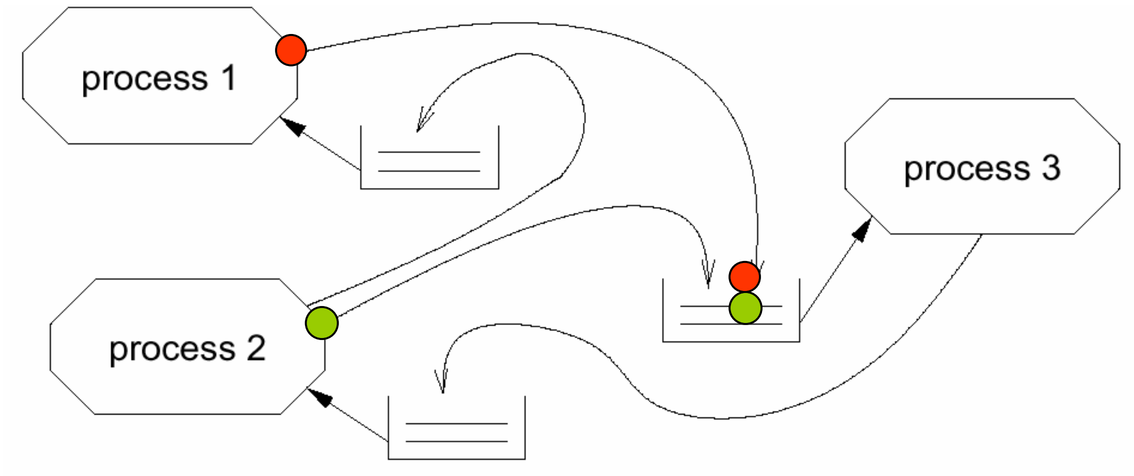
Communication between FSMs (or “processes“) is based on **message-passing**, assuming a **potentially indefinitely large FIFO-queue**.



- Each process fetches next entry from FIFO,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is ignored (exception: SAVE-mechanism).

Determinate?

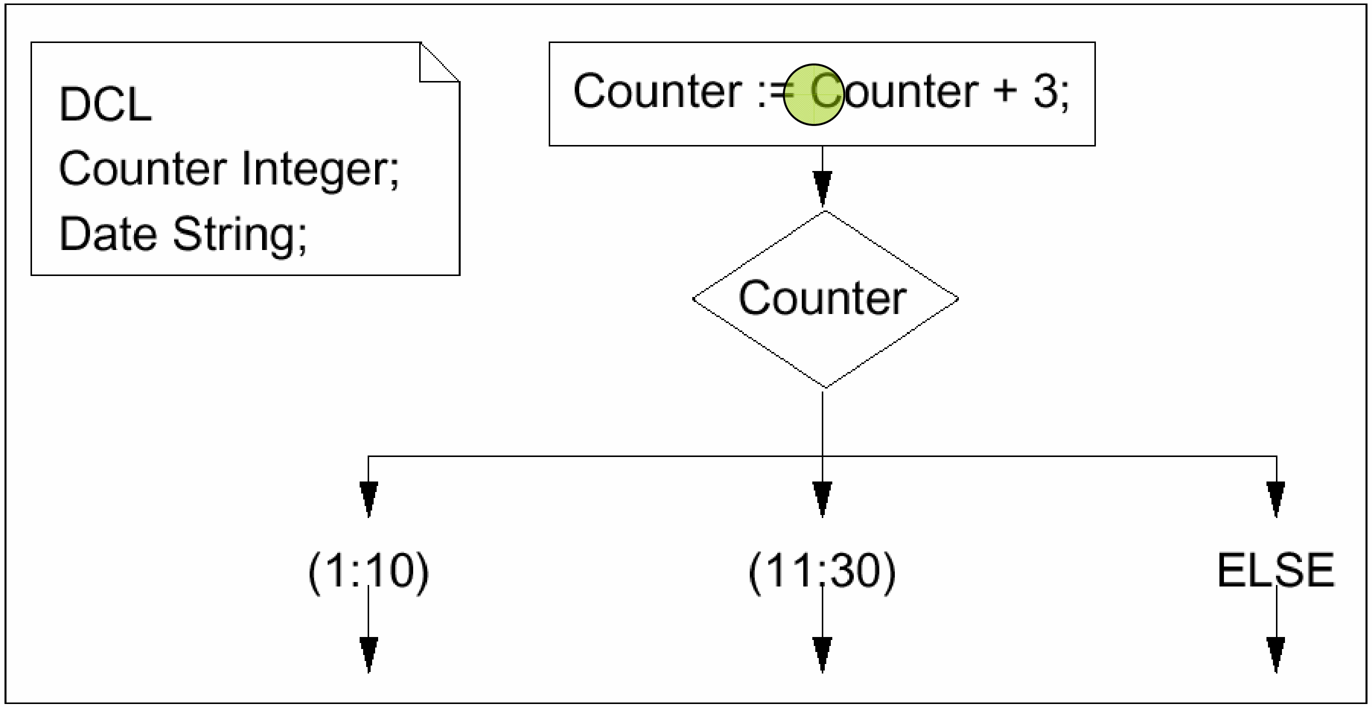
Let tokens be arriving at FIFO at the same time:
☞Order in which they are stored, is unknown:



All orders are legal: ☞simulators can show different behaviors for the same input, all of which are correct.

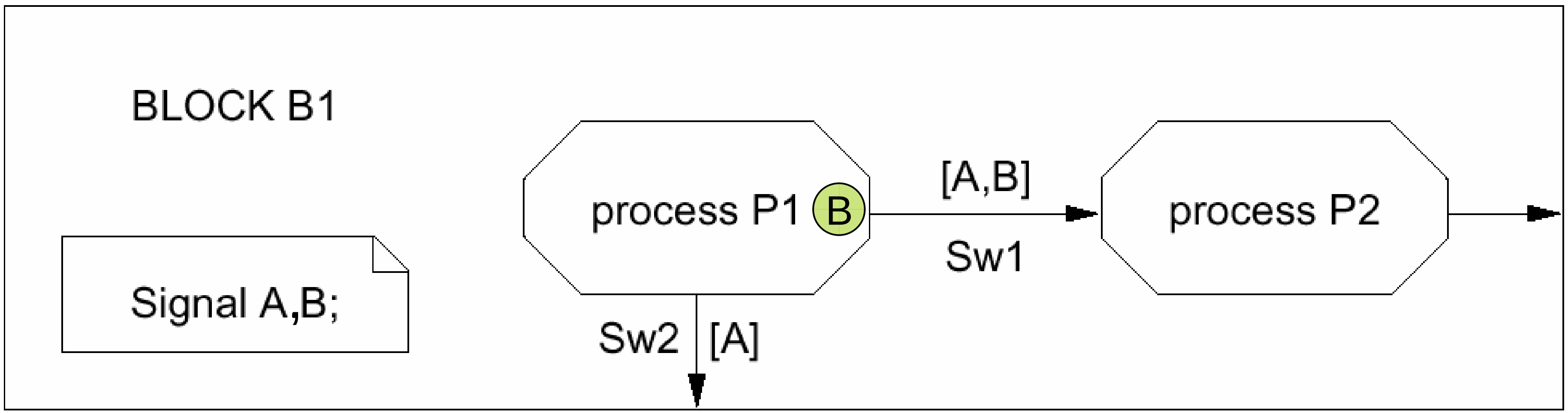
Operations on data

Variables can be declared locally for processes.
Their type can be predefined or defined in SDL itself.
SDL supports abstract data types (ADTs). Examples:



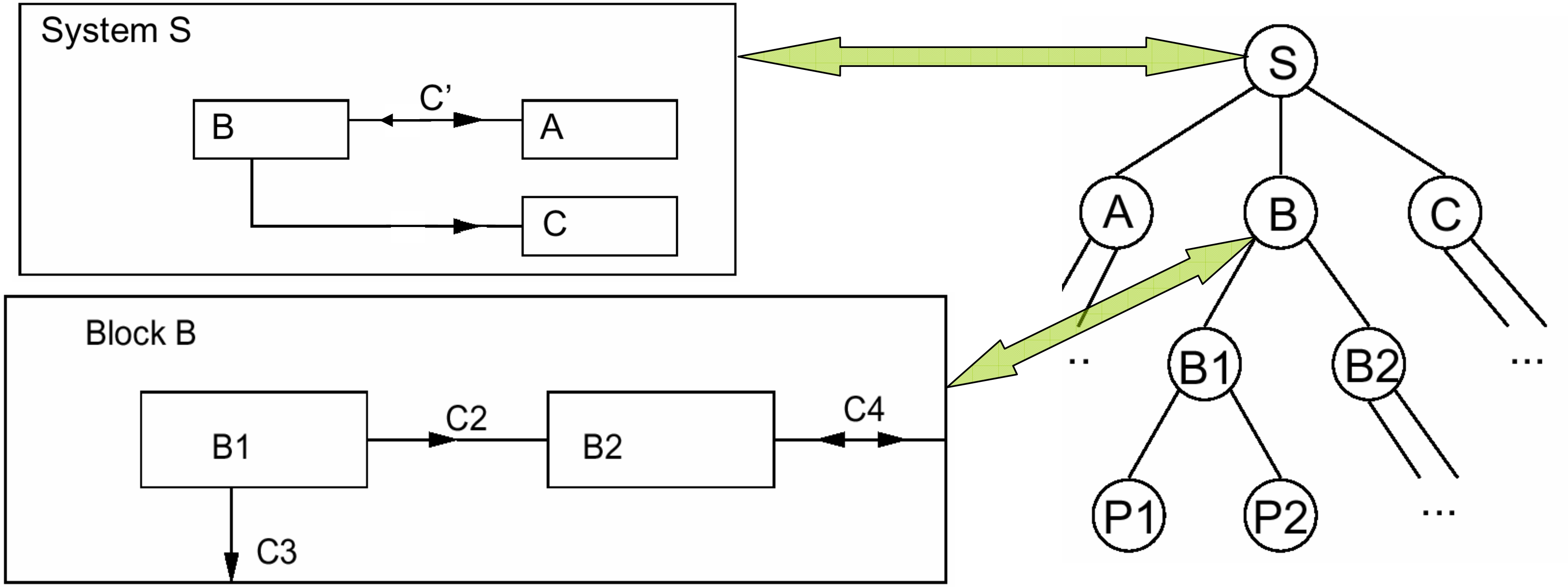
Process interaction diagrams

Interaction between processes can be described in process interaction diagrams (special case of block diagrams).
In addition to processes, these diagrams contain channels and declarations of local signals.
Example:



Hierarchy in SDL

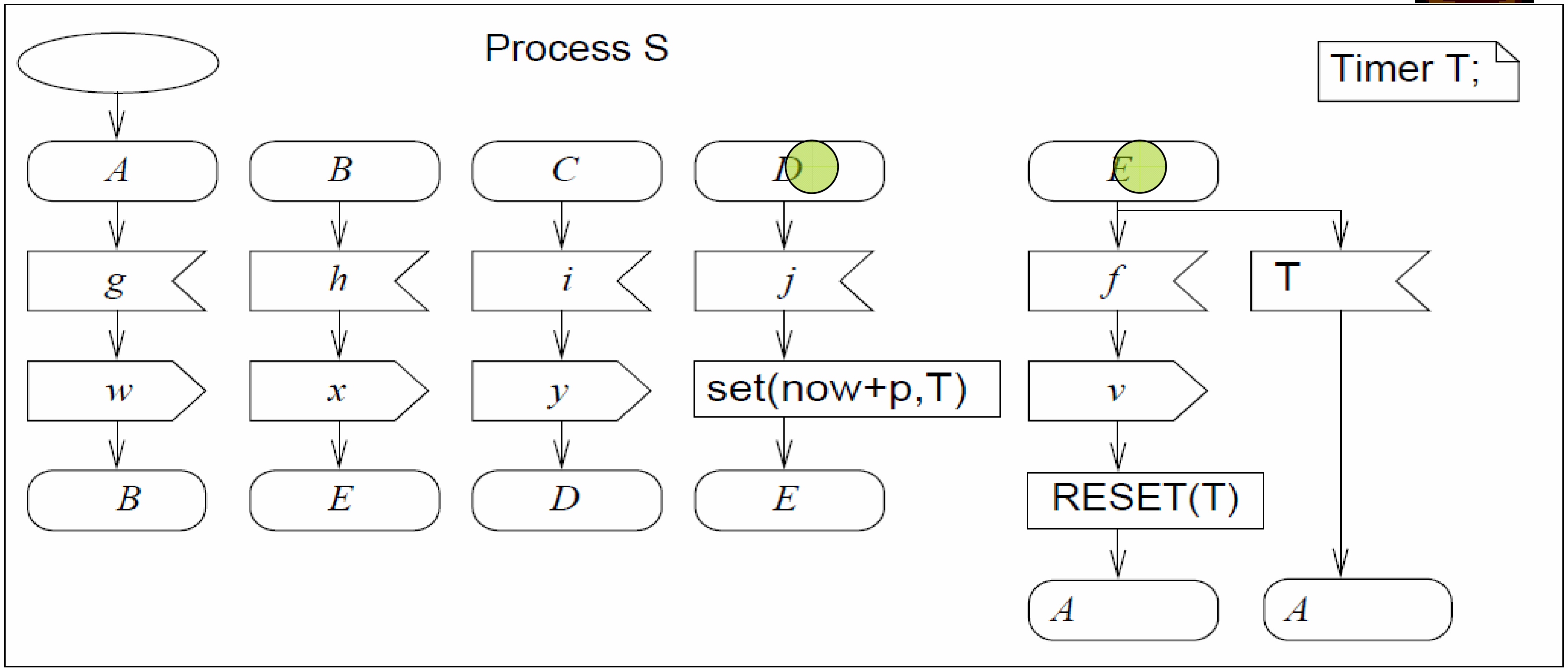
Process interaction diagrams can be included in **blocks.**
The root block is called **system.**



Processes cannot contain other processes, unlike in StateCharts.

Timers

Timers can be declared locally. Elapsed timers put signal into queue (not necessarily processed immediately). RESET removes timer (also from FIFO-queue).

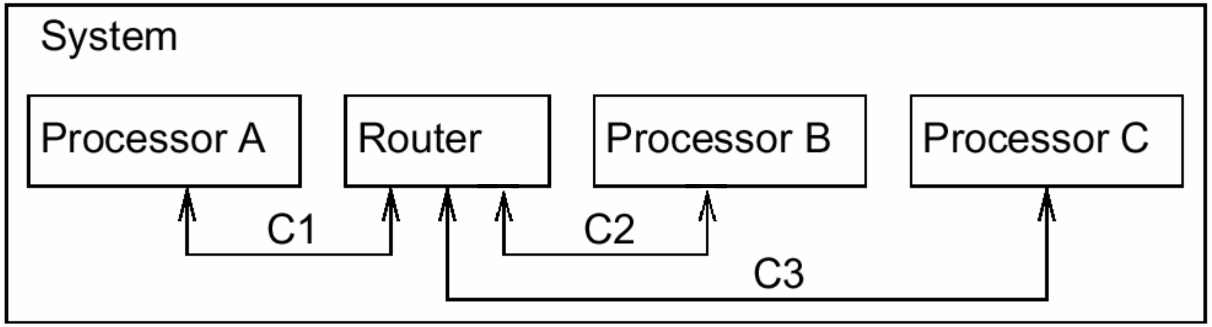


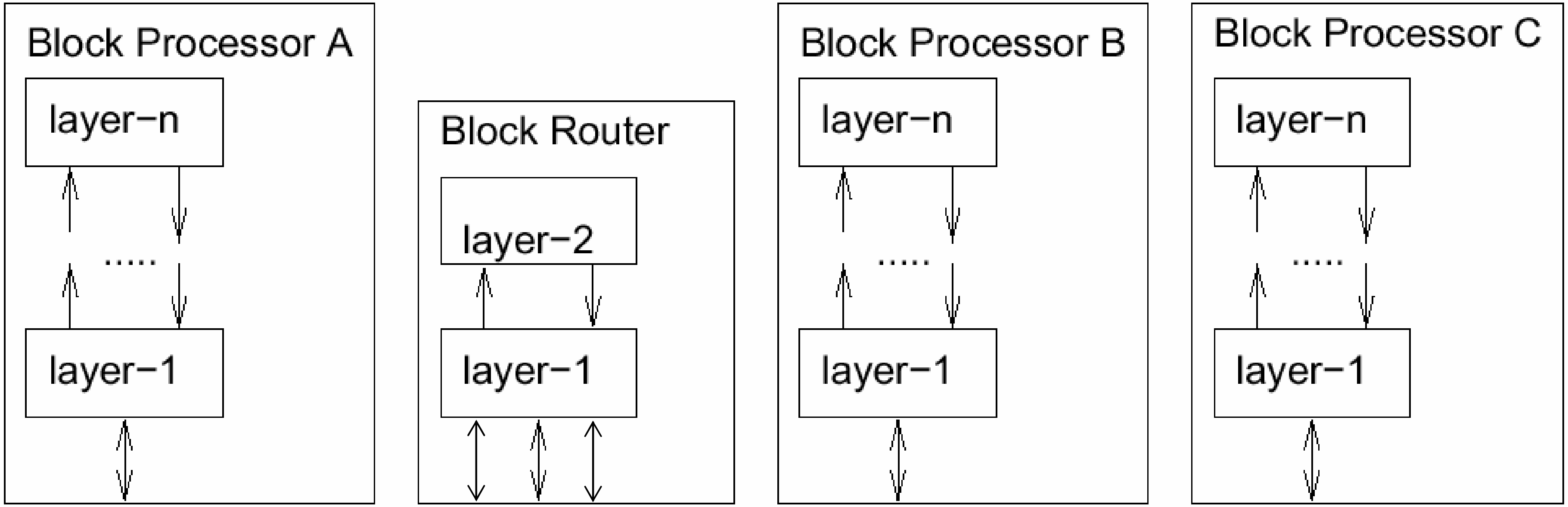
Additional language elements

SDL includes a number of additional language elements, like

- procedures
- creation and termination of processes
- advanced description of data
- More features added for SDL-2000 (not well accepted)

Application: description of network protocols



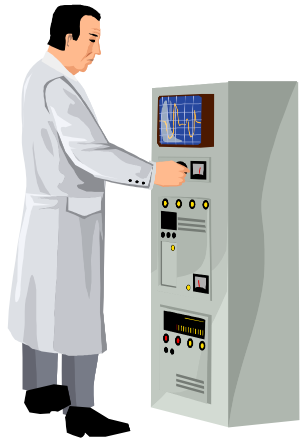


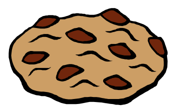
Larger example: vending machine

Machine° selling pretzels, (potato) chips, cookies, and doughnuts:

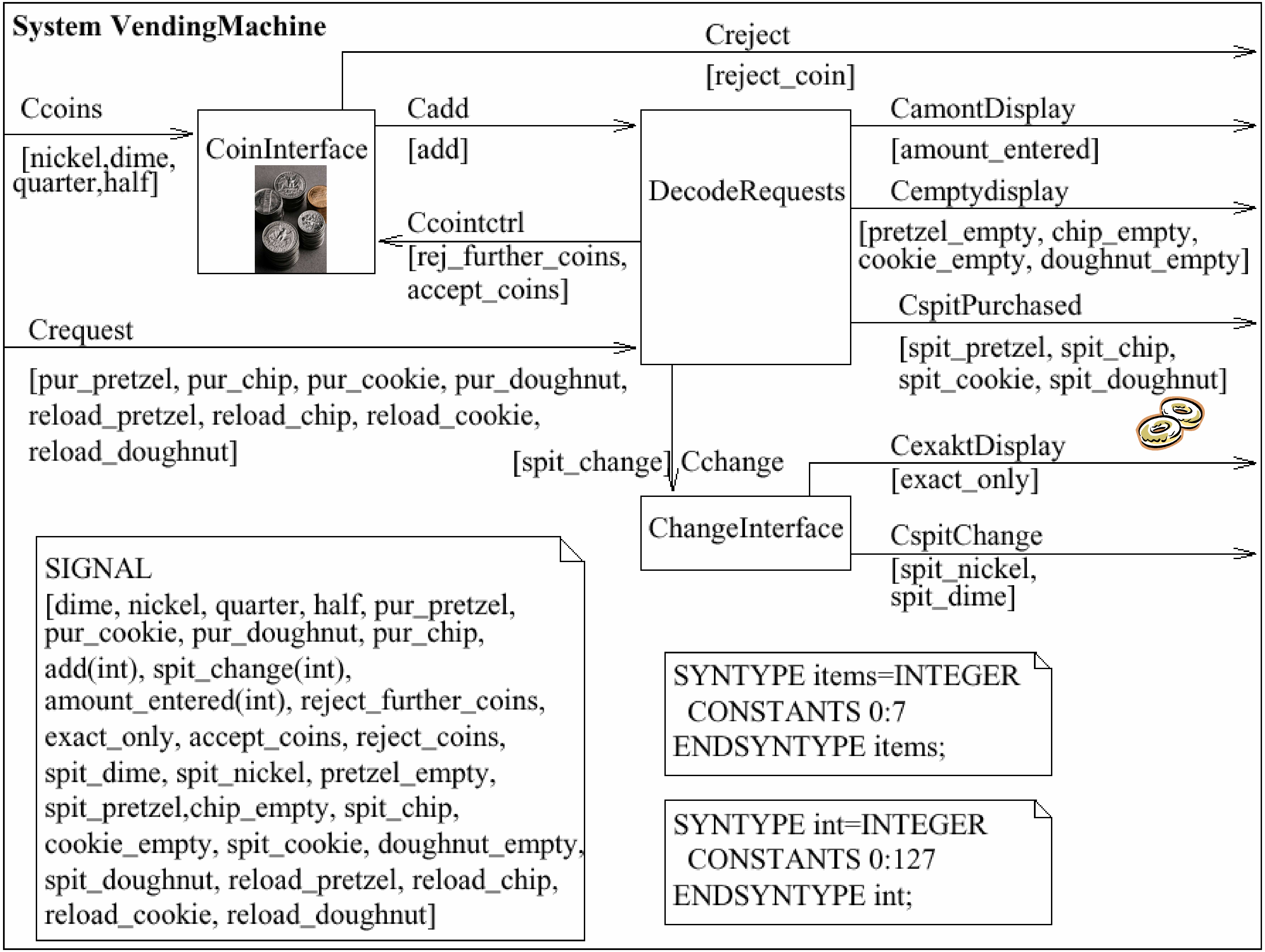
accepts nickels, dime, quarters, and half-dollar coins.

Not a distributed application.

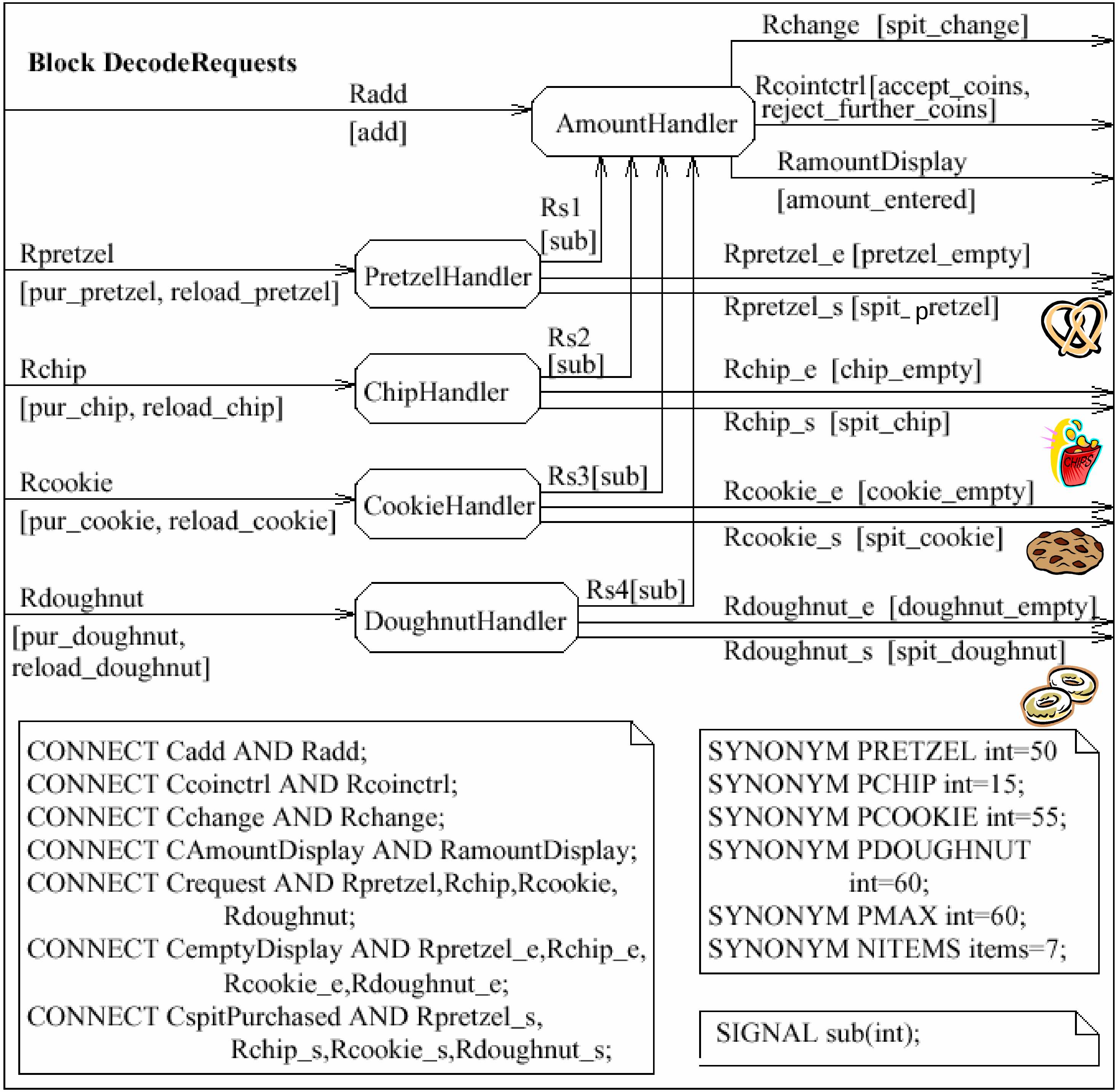




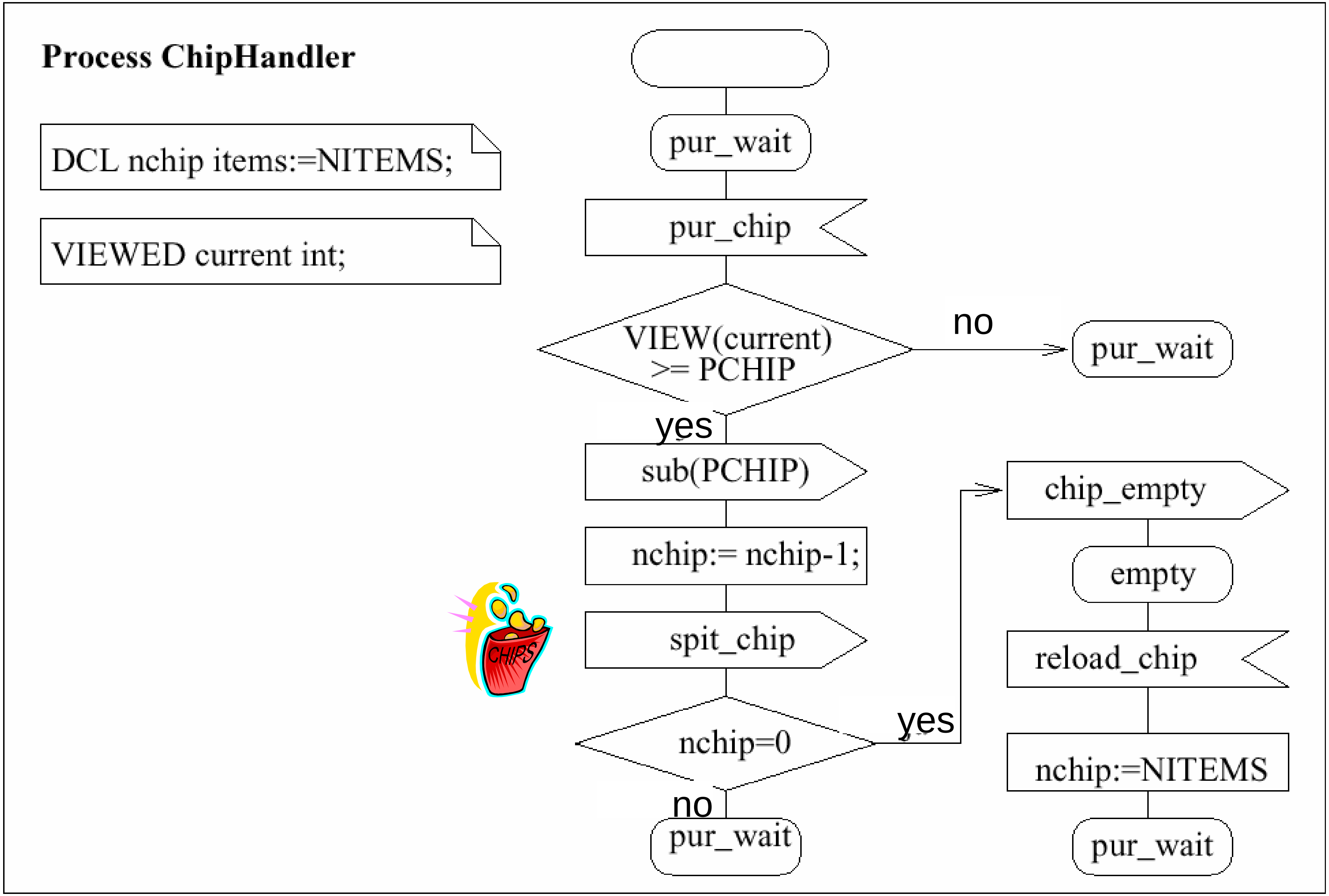
° [J.M. Bergé, O. Levia, J. Roullard: High-Level System Modeling, Kluwer Academic Publishers, 1995]


System VendingMachine
Creject
[reject_coin]
Ccoins
[nickel,dime, quarter,half]
CoinInterface
Cadd
[add]
DecodeRequests
CamontDisplay
[amount_entered]
Cemptydisplay
[pretzel_empty, chip_empty, cookie_empty, doughnut_empty]
Ccointctrl
[rej_further_coins, accept_coins]
CspitPurchased
[spit_pretzel, spit_chip, spit_cookie, spit_doughnut]
Crequest
[pur_pretzel, pur_chip, pur_cookie, pur_doughnut, reload_pretzel, reload_chip, reload_cookie, reload_doughnut]
[spit_change]
Cchange
CexaktDisplay
[exact_only]
ChangeInterface
CspitChange
[spit_nickel, spit_dime]
SIGNAL
[dime, nickel, quarter, half, pur_pretzel, pur_cookie, pur_doughnut, pur_chip, add(int), spit_change(int), amount_entered(int), reject_further_coins, exact_only, accept_coins, reject_coins, spit_dime, spit_nickel, pretzel_empty, spit_pretzel,chip_empty, spit_chip, cookie_empty, spit_cookie, doughnut_empty, spit_doughnut, reload_pretzel, reload_chip, reload_cookie, reload_doughnut]
SYNTYPE items=INTEGER
CONSTANTS 0:7
ENDSYNTYPE items;
SYNTYPE int=INTEGER
CONSTANTS 0:127
ENDSYNTYPE int;

Decode Requests



ChipHandler



Evaluation & summary

- FSM model for the components,
- Non-blocking message passing for communication,
- Implementation requires bound for the maximum length of FIFOs; may be very difficult to compute,
- Excellent for distributed applications (used for ISDN),
- Commercial tools available (see http://www.sdl-forum.org)
- Not necessarily determinate
  (order, in which FSMs are reading input is unknown)
- Timer concept adequate just for soft deadlines,
- Limited way of using hierarchies,
- Limited programming language support,
- No description of non-functional properties,
- Becoming less popular
- Examples: small network + vending machine

technische universität
dortmund

fakultät für informatik
informatik 12

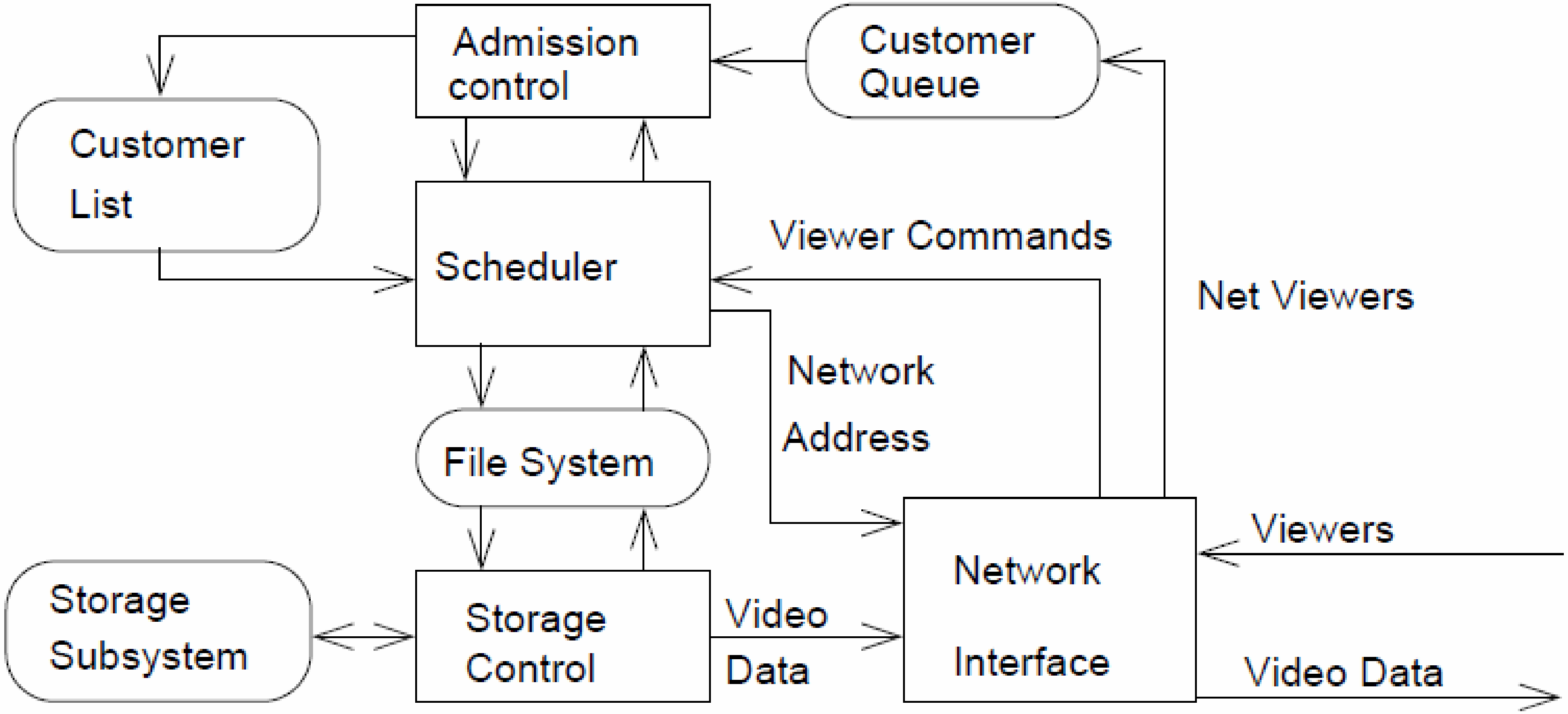
Data flow models

Peter Marwedel
TU Dortmund,
Informatik 12

2010/10/10

Data flow as a "natural" model of applications

Example: Video on demand system



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)*".

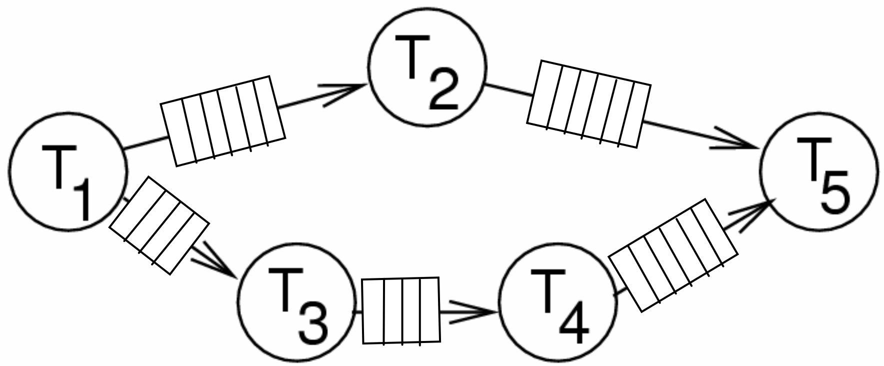
[Wikipedia: Structured systems analysis and design method. *http://en.wikipedia.org/wiki/Structured Systems Analysis and Design Methodology,* 2010 (formatting added)].

Models of computation considered in this course

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	|

Kahn process networks

- Each component is a program/task/process, not an FSM
- Communication is by FIFOs; no overflow considered
  - ☞ writes never have to wait,
  - ☞ reads wait if FIFO is empty.



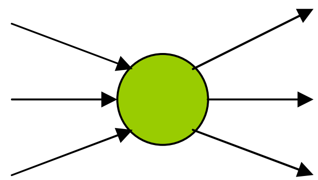
- Only one sender and one receiver per FIFO
  - ☞ no SDL-like conflicts at FIFOs

Example

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

Properties of Kahn process networks (1)

- Communication is only via channels;
- Mapping from ≥1 input channel to ≥1 output channel;
- Channels transmit information within an unpredictable but finite amount of time;
- In general, execution times are unknown.



Key beauty of KPNs

- A process cannot check whether data is available before attempting a read.
- A process cannot wait for data for more than one port at a time.
- Therefore, the order of reads depends only on data, not on the 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.

☞ Many applications in embedded system design:
Any combination of fast and slow simulation & hardware prototypes always gives the same result.

Computational power and analyzability

- KPNs are Turing-complete (anything which can be computed can be computed by a KPN)
- It is a challenge to schedule KPNs without accumulating tokens
- KPNs are computationally powerful, but difficult to analyze (e.g. what's the maximum buffer size?)
- Number of processes is static (cannot change)

More information about KPNs

- http://ls12-www.cs.tu-dortmund.de/teaching/download/levi/index.html: Animation
- 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

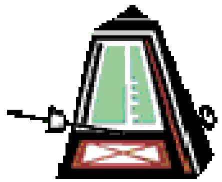
SDF

Less computationally powerful, but easier to analyze:

Synchronous data flow (SDF).

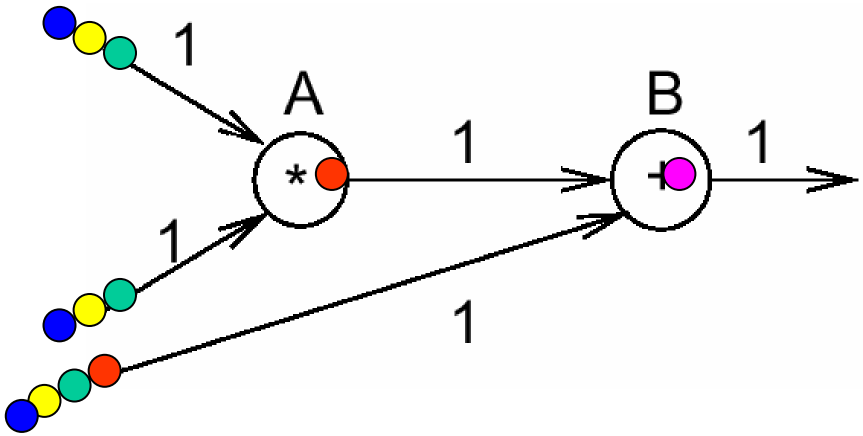
Again using asynchronous message passing.

Synchronous data flow (SDF)



Asynchronous message passing=
tasks do not have to wait until output is accepted.

Synchronous data flow =
global clock controlling "firing" of nodes



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

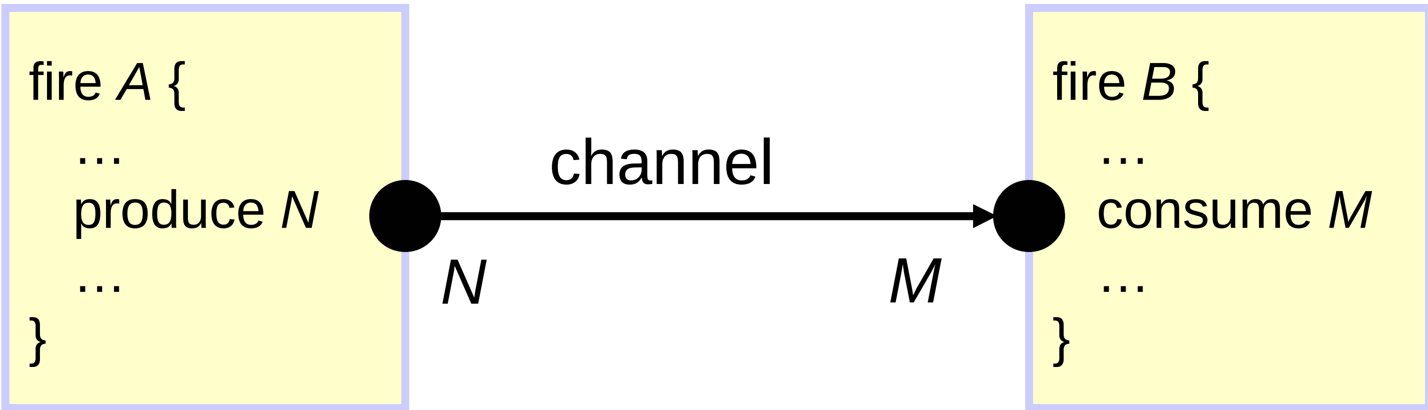
Balance equations (one for each channel)

$$f_A N = f_B M$$

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.

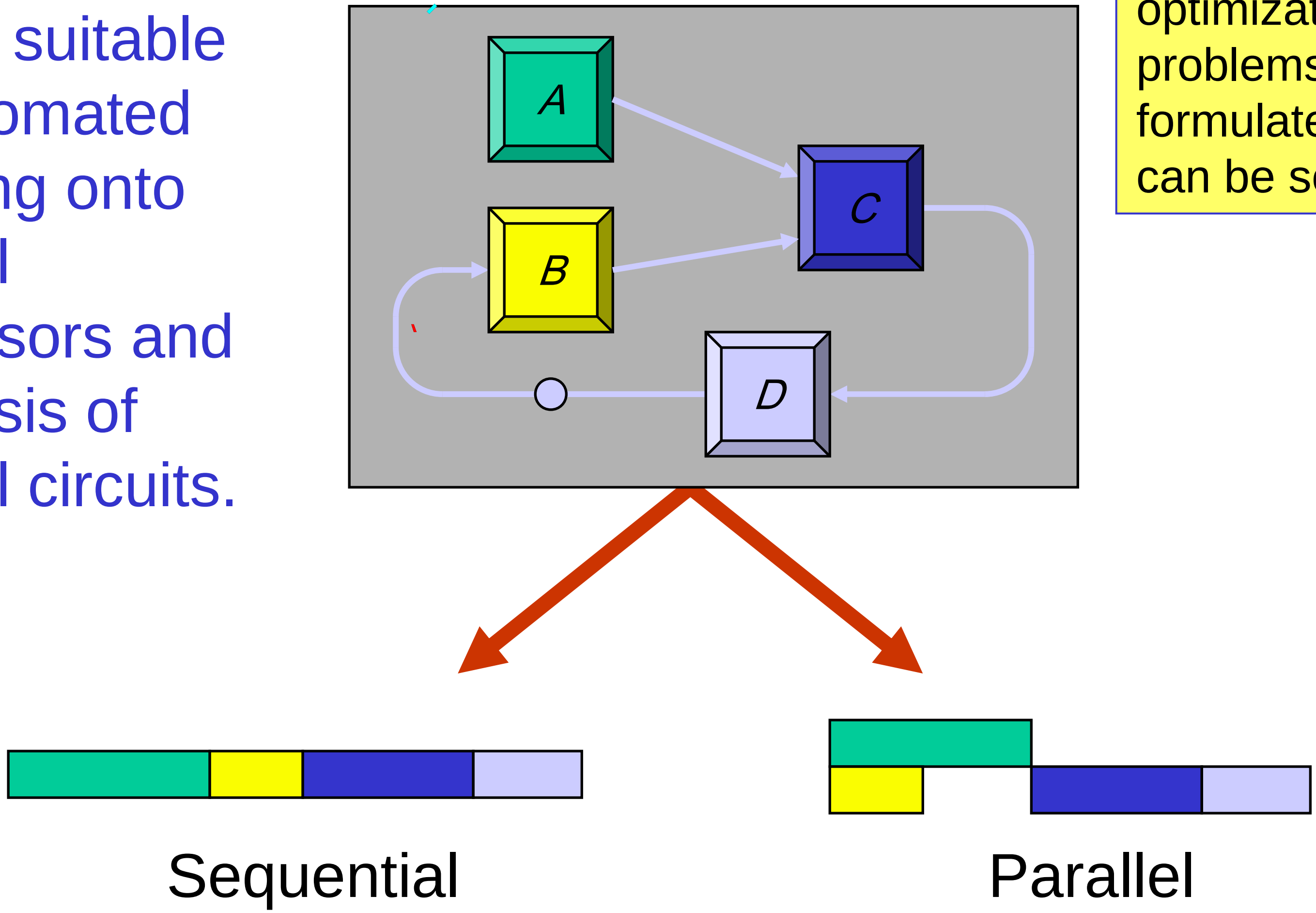
Decidable:

- buffer memory requirements
- deadlock

Source: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

Parallel Scheduling of SDF Models

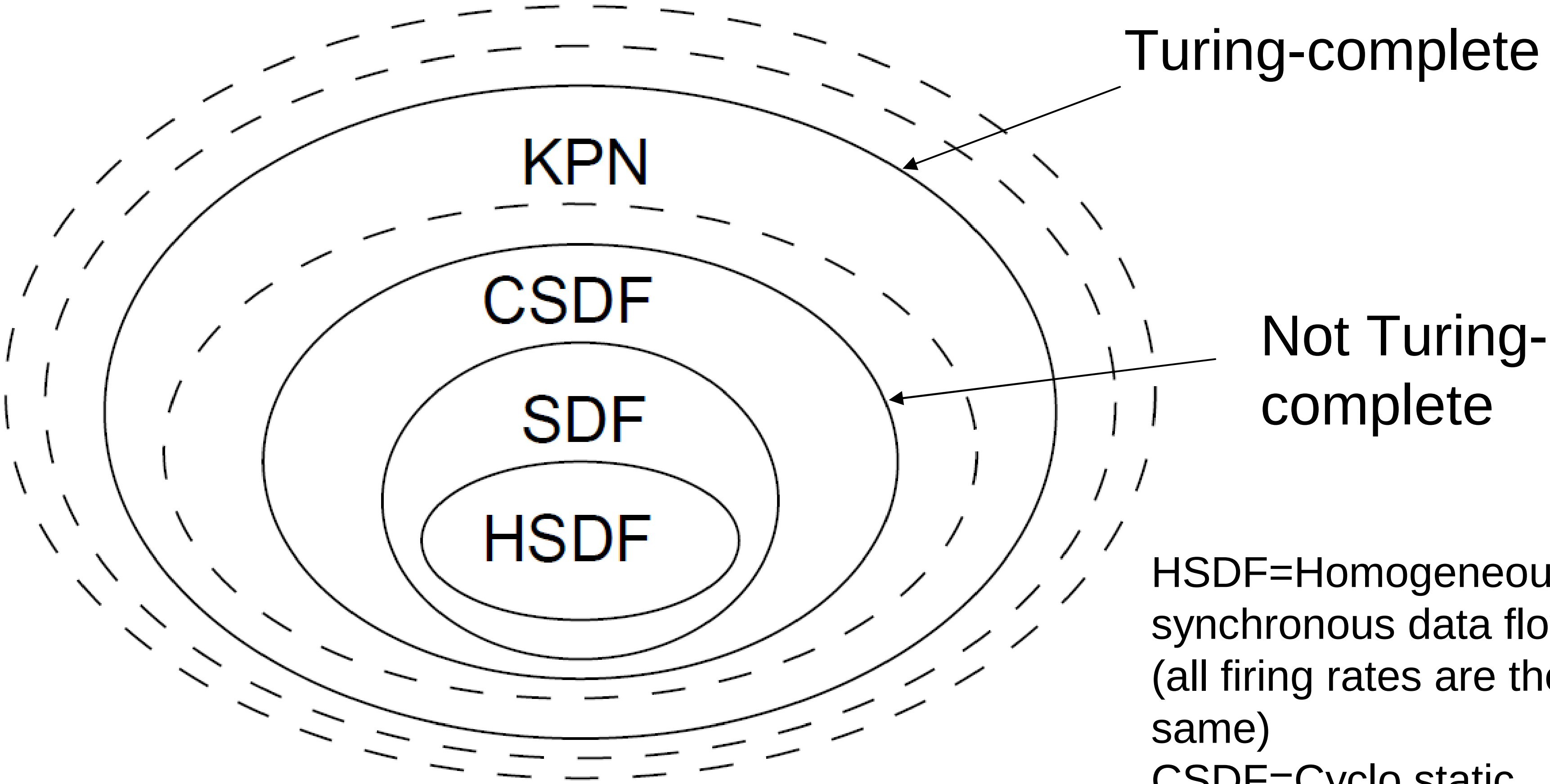
SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.



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

Source: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

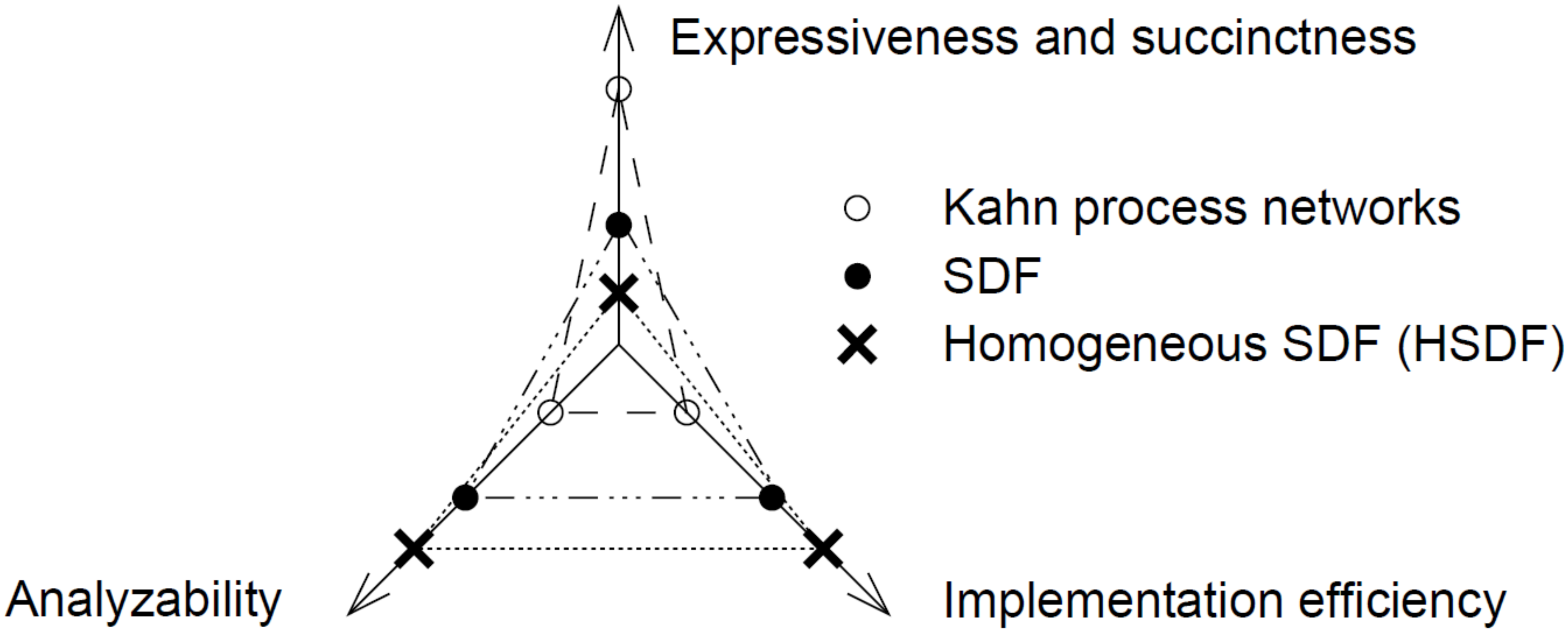
Expressiveness of data flow MoCs



HSDF=Homogeneous synchronous data flow (all firing rates are the same)
CSDF=Cyclo static data flow (rates vary in a cyclic way)

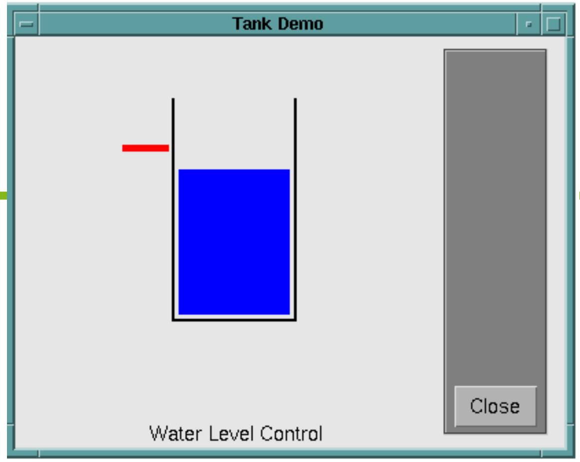
[S. Stuijk, 2007]

The expressiveness/analyzability conflict

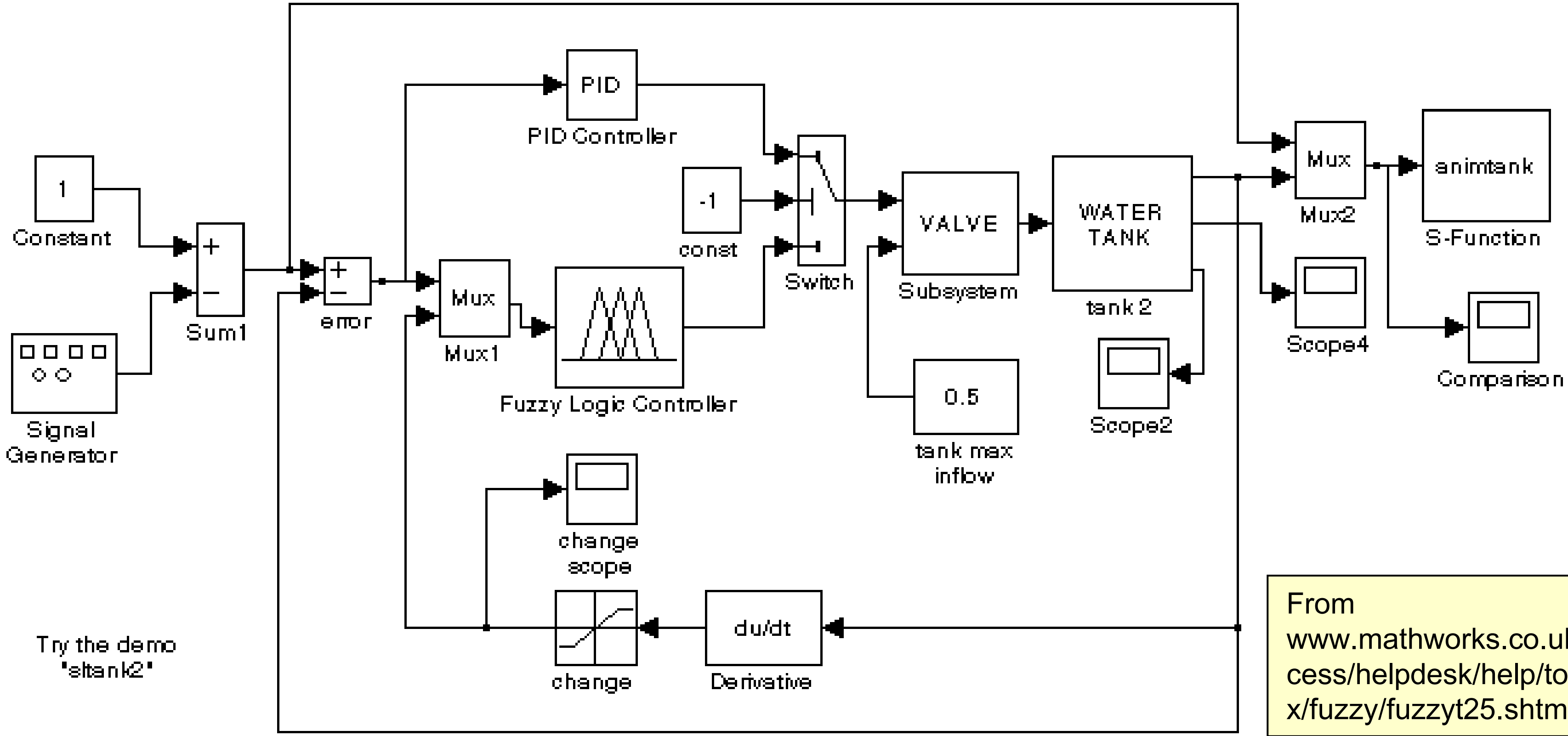


[S. Stuijk, 2007]

Similar MoC: Simulink
- example -



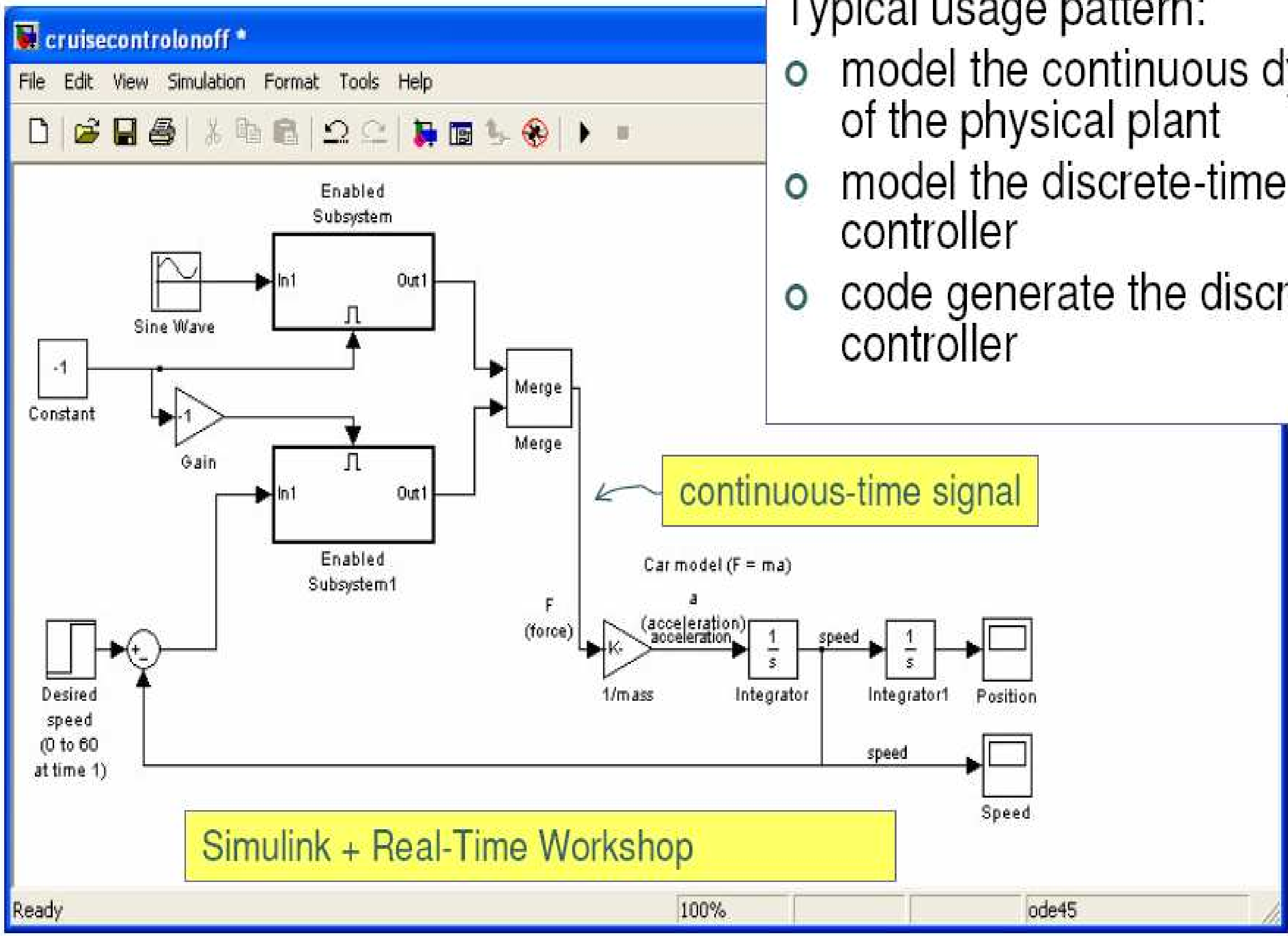
Semantics? *"Simulink uses an idealized timing model for block execution and communication. Both happen infinitely fast at exact points in simulated time. Thereafter, simulated time is advanced by exact time steps. All values on edges are constant in between time steps."* [Nicolae Marian, Yue Ma]



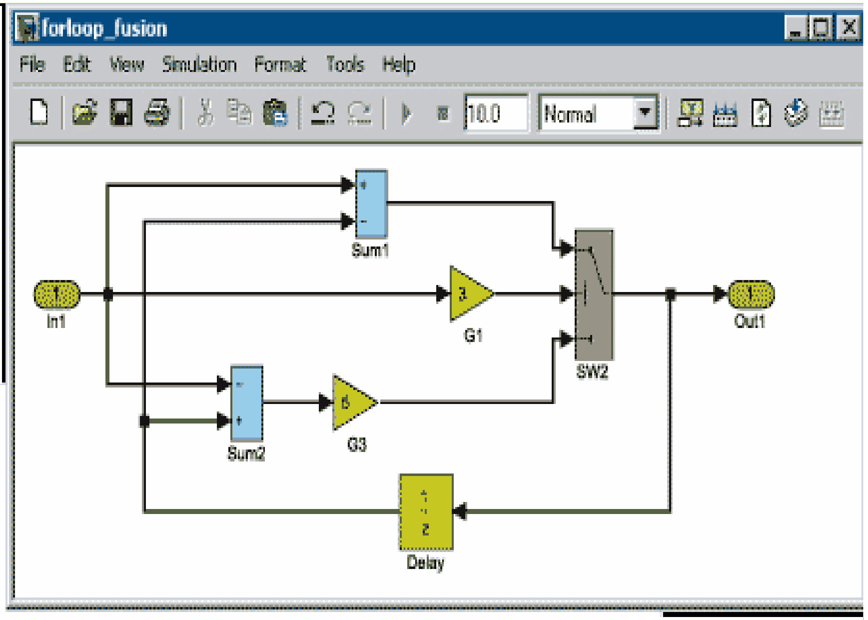
From www.mathworks.co.uk/access/helpdesk/help/toolbox/fuzzy/fuzzyt25.shtml

Threads are Not the Only Possibility: 6th example: Continuous-Time Languages

Typical usage pattern:
- model the continuous dynamics of the physical plant
- model the discrete-time controller
- code generate the discrete-time controller



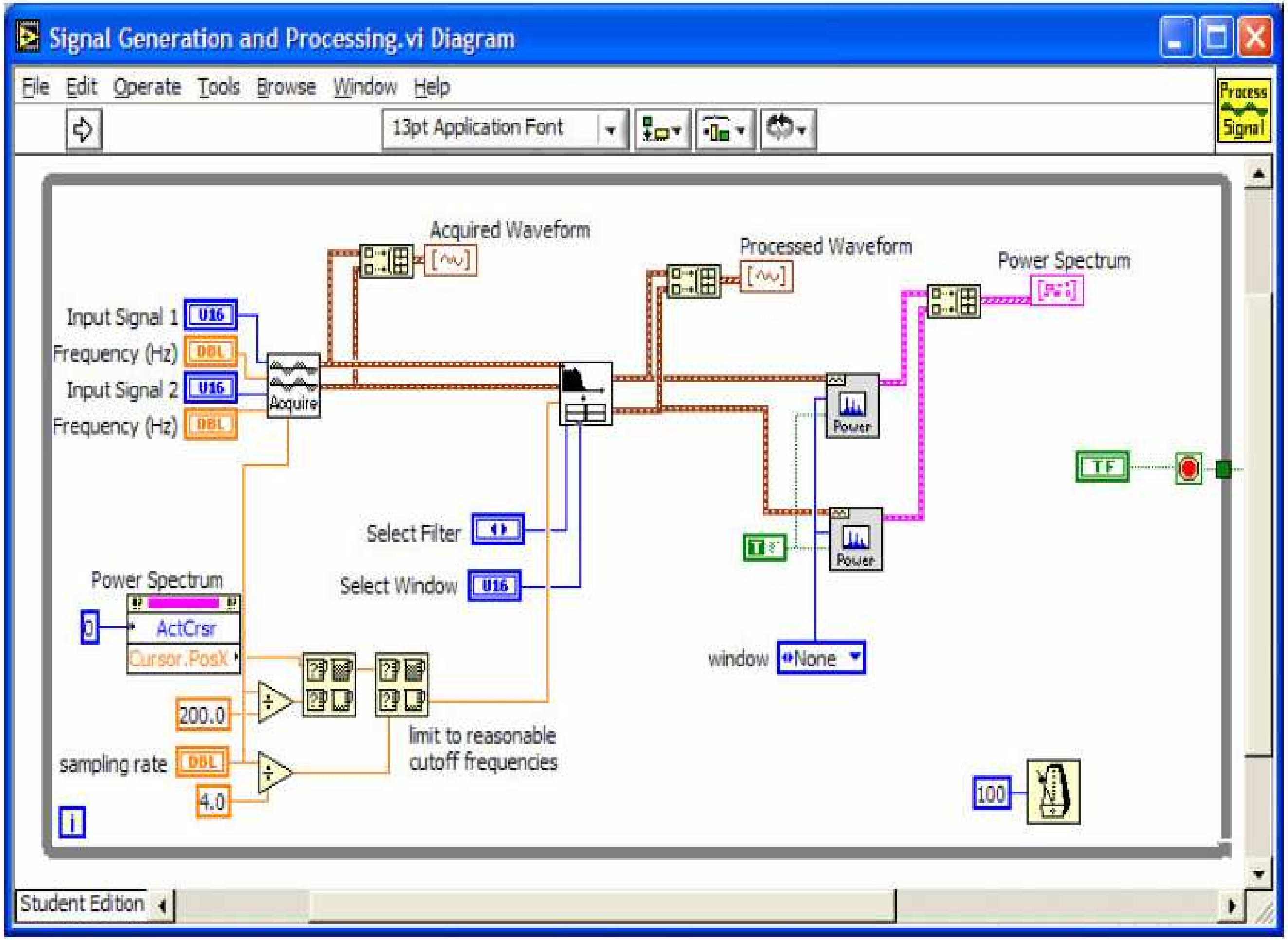
Starting point for “model-based design”



Code automatically generated

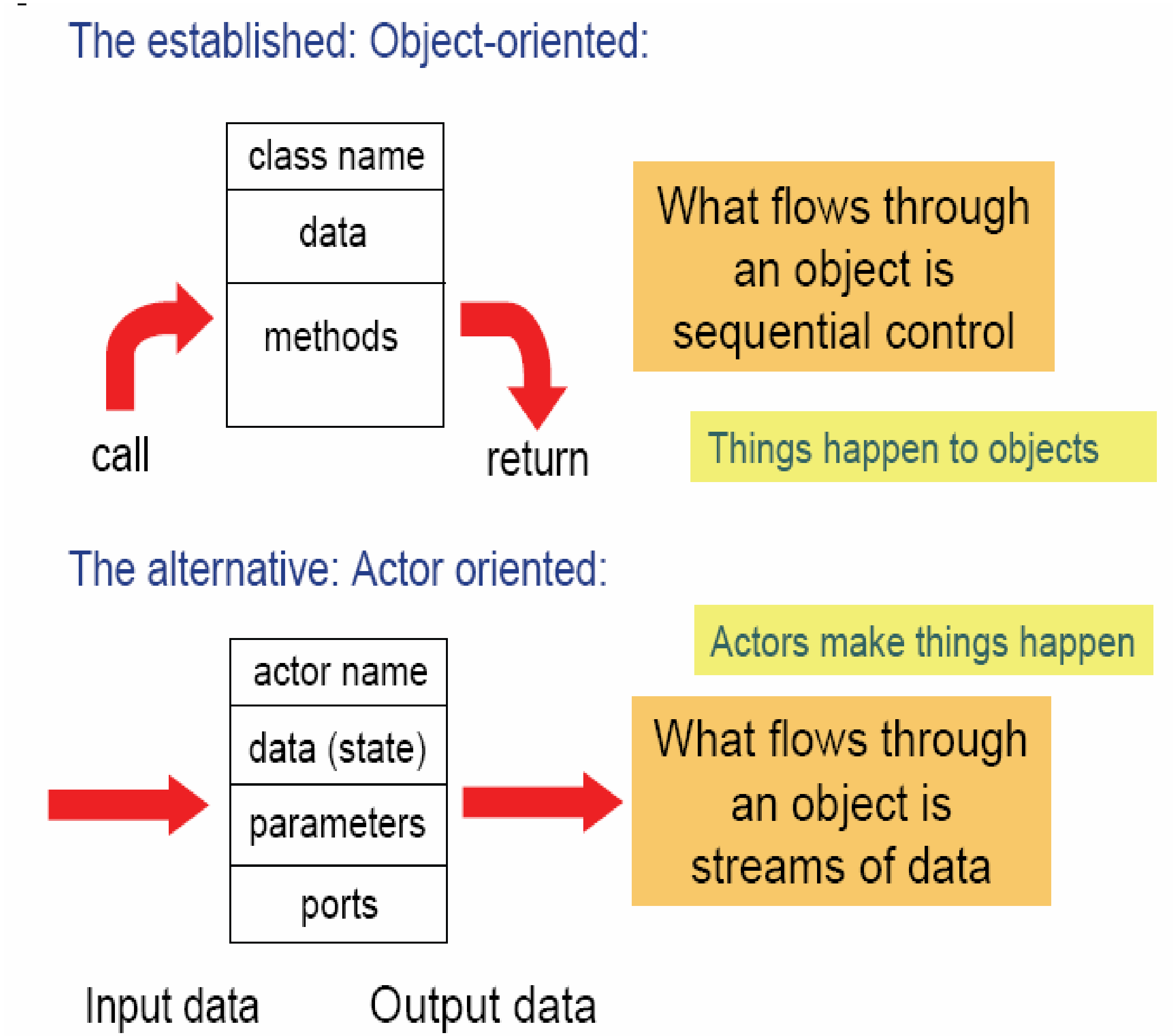
```
 *  Gain: '<Root>/G1'
 *  Sum: '<Root>/Sum2'
 *  Gain: '<Root>/G3'
 */
for(i1=0; i1<10; i1++) {
  if(rtU.In1[i1] * 3.0 >= 0.0) {
    rtb_SW2_c[i1] = rtU.In1[i1] - rtDWork.Delay_DSTATE[i1];
  } else {
    rtb_SW2_c[i1] = (rtDWork.Delay_DSTATE[i1] - rtU.In1[i1]) * 5.0;
  }

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

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

Threads are Not the Only Possibility: 5th example: Instrumentation Languages



e.g. LabVIEW, Structured dataflow model of computation

Actor languages



Summary

Data flow model of computation

- Motivation
- Kahn process networks
- SDF
- Visual programming languages
  - Simulink