

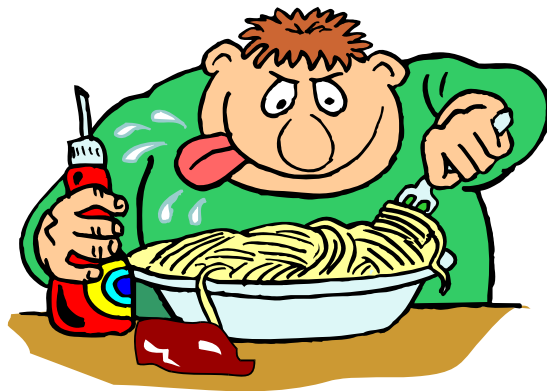
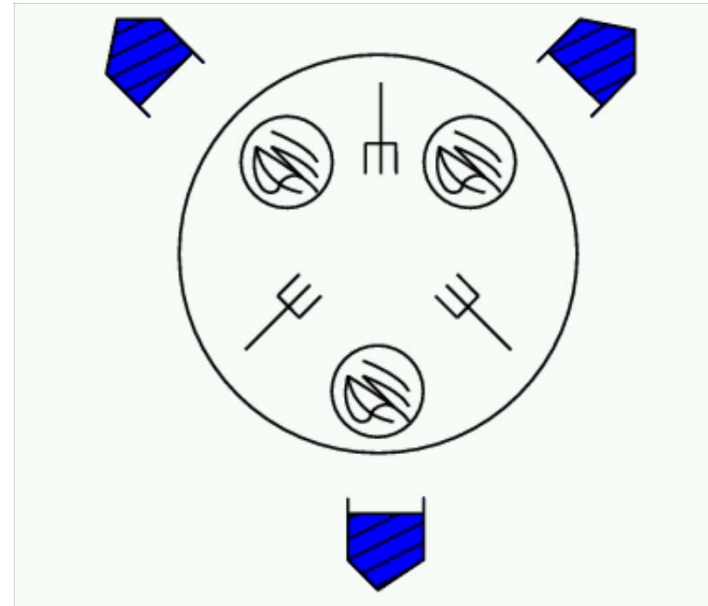
Embedded Systems



- Goal: compact representation of complex systems.
- Key changes:
 - Tokens are becoming individuals;
 - Transitions enabled if functions at incoming edges true;
 - Individuals generated by firing transitions defined through functions
- Changes can be explained by folding and unfolding C/E nets

Example: Dining philosophers problem REVIEW

- $n > 1$ philosophers sitting at a round table;
- n forks,
- n plates with spaghetti;
- philosophers either thinking or eating spaghetti (using left and right fork).



2 forks
needed!

How to model conflict for forks?

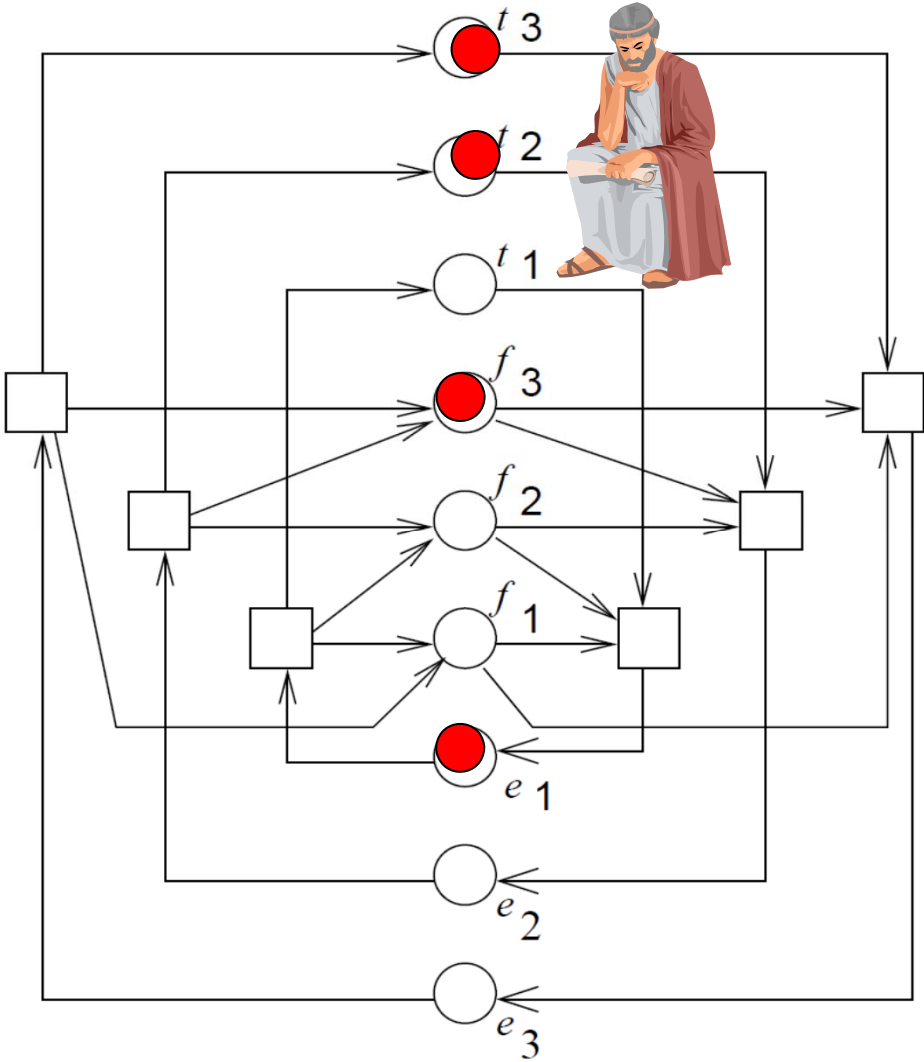
How to guarantee avoiding
starvation?

Condition/event net model of the dining philosophers problem

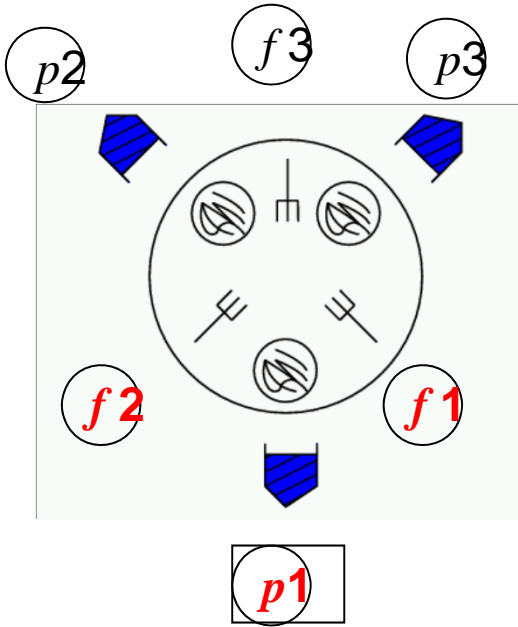
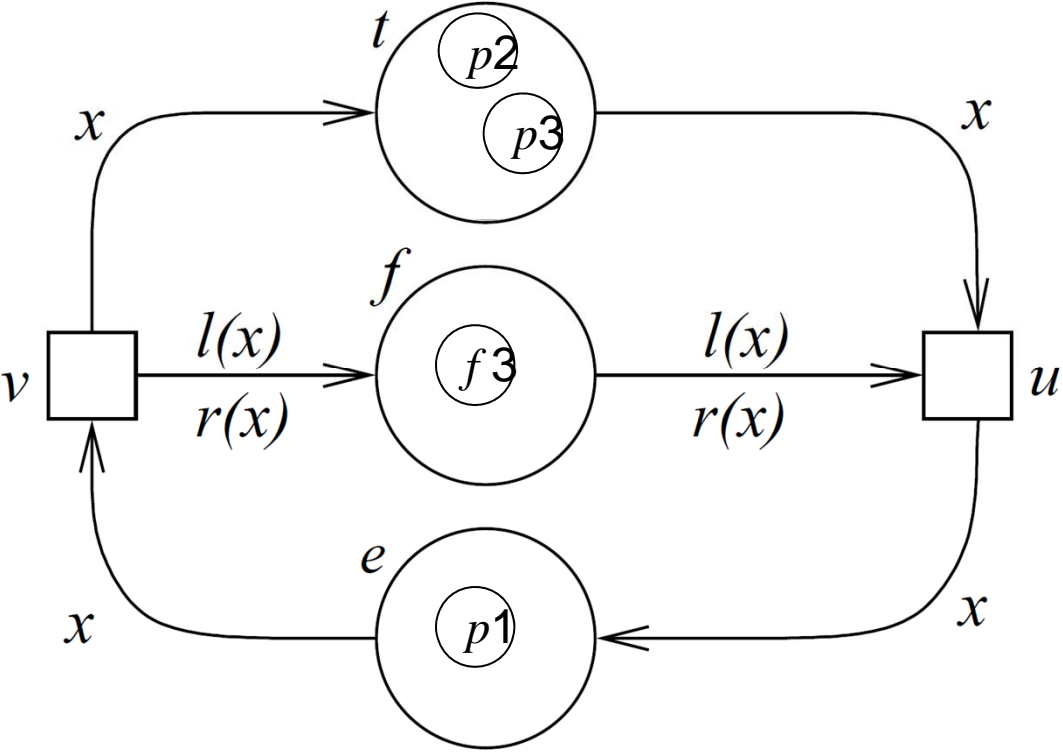
REVIEW

- Let $x \in \{1..3\}$
- t_x : x is thinking
- e_x : x is eating
- f_x : fork x is available

Model quite clumsy.
Difficult to extend to more philosophers.



Predicate/transition model of the dining philosophers problem (1) REVIEW



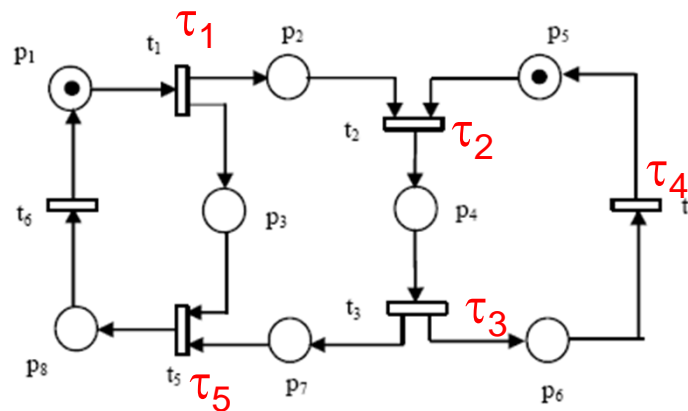
Time and Petri Nets

- e.g.: Petri nets tell us that "a new request can be issued only after the resource is released"
- Nothing about time
- In literature, time has been added to PNs in many different ways (notion of temporal constraints for: transitions, places, arcs) → TPN

Timed Petri Nets

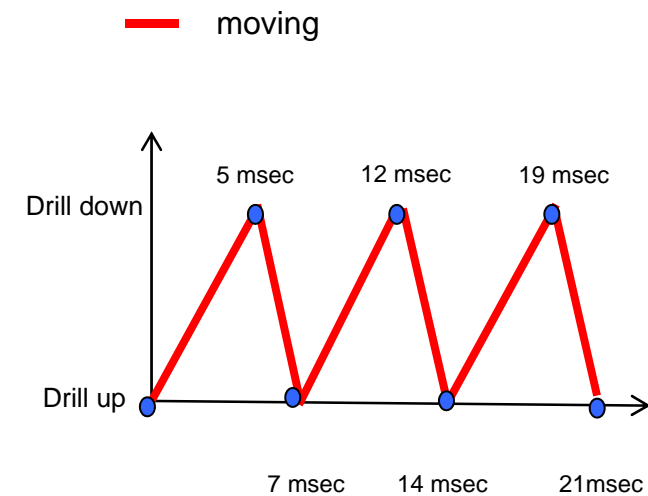
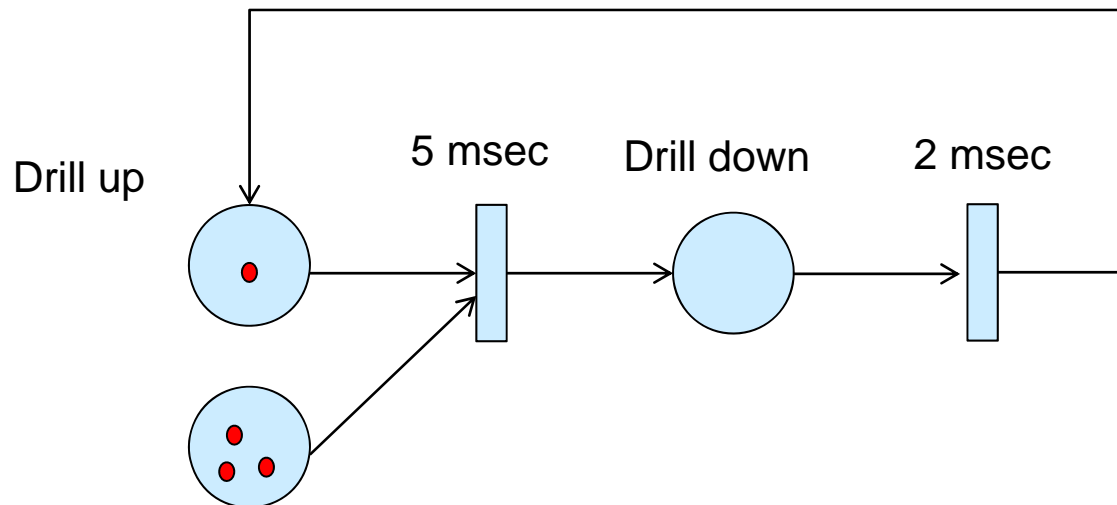
REVIEW

- TPN
 - Each transition is defined precisely based on connectivity and tokens needed for transition
 - Given an initial condition, the **exact system state at an arbitrary future time T can be determined**
- Timed Petri Nets becomes a 7-tuple system
 - $PN = (P, T, F, W, K, M_0, \tau)$
 - $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ is a finite set of deterministic time delays to **corresponding t_i**

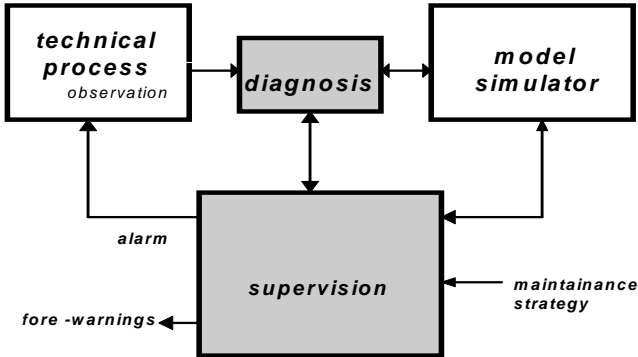
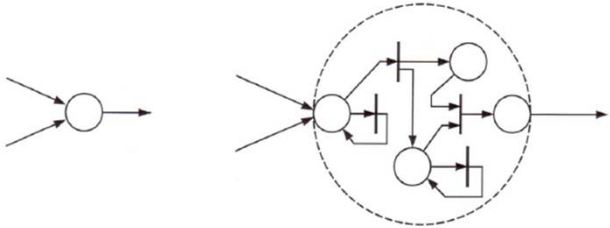
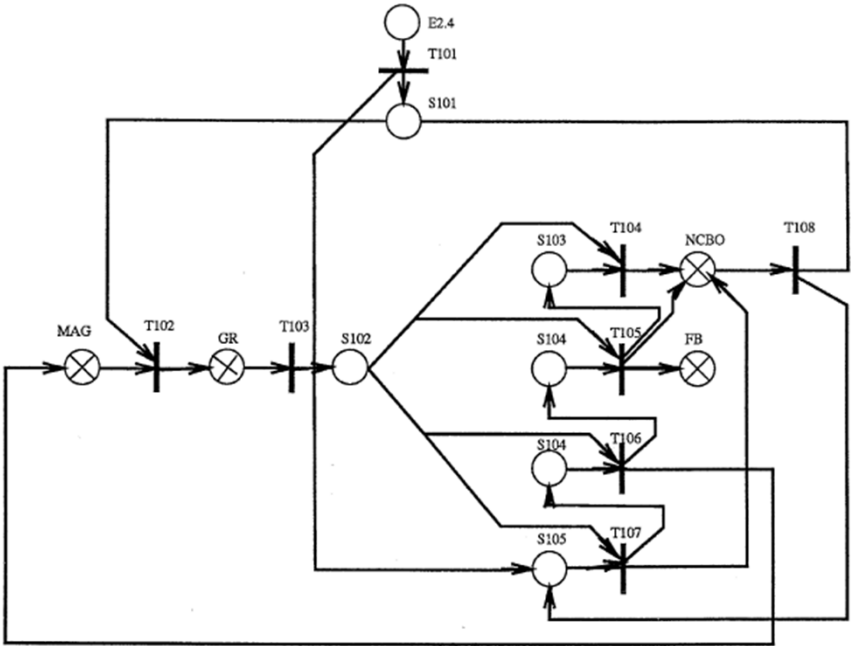
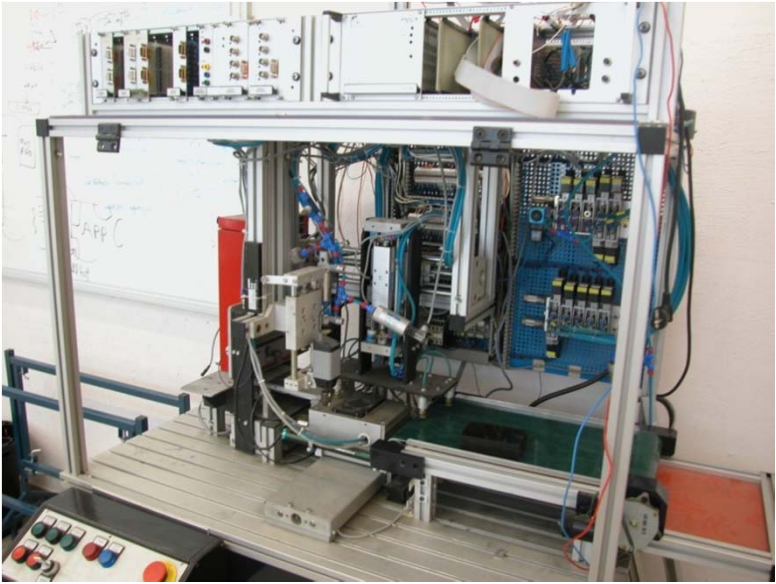


Time and Petri Nets (TPN)

- adding (quantitative) time to PNs is to introduce temporal constraints on its elements:
 - e.g., a transition must fire after 5 msec



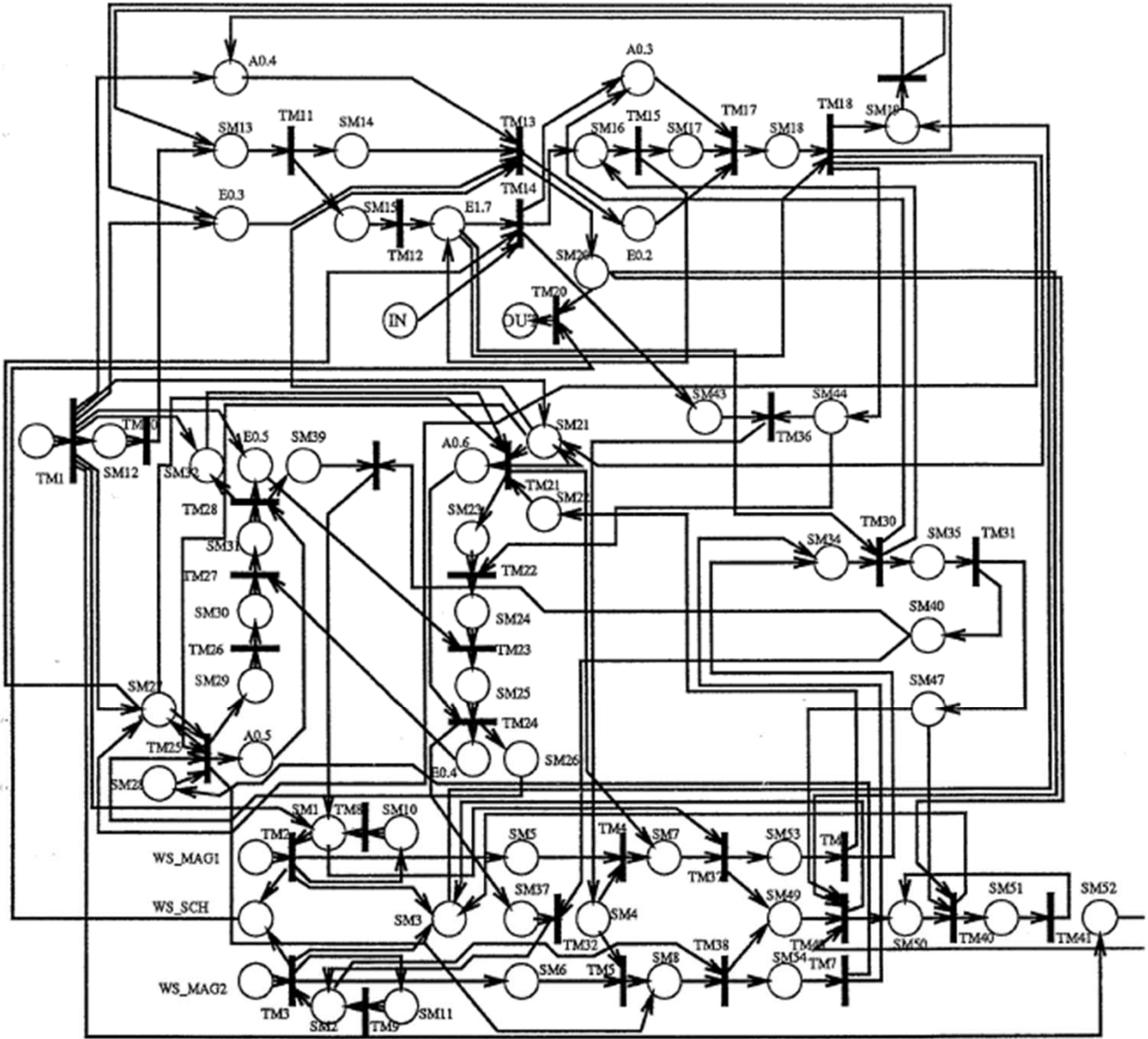
Production system - Top level petri net REVIEW



A Distributed Real-Time Expert System for Model-Based Fault Diagnosis in Modular Production Systems

magazine/depot

REVIEW



- **Pros:**
 - Appropriate for distributed applications,
 - Well-known theory for formally proving properties,
- **Cons :**
 - PN problems with modeling timing (extensions in TPN)
 - no programming elements, no hierarchy (extensions available)
- **Extensions:**
 - Enormous amounts of efforts on removing limitations.
- **Remark:**
 - A FSM can be represented by a subclass of Petri nets, where each transition has exactly one incoming edge and one outgoing edge.

Summary


REVIEW

- Petri nets: focus on causal dependencies
 - Condition/event nets
 - Single token per place
 - Place/transition nets
 - Multiple tokens per place
 - Predicate/transition nets
 - Tokens become individuals
 - Dining philosophers used as an example
 - Extensions required to get around limitations

REVIEW

SDL - Specification and Description *Language*

SDL - Specification and Description *Language*

- Used here as a (prominent) example of a model of computation based on **asynchronous message passing communication**.
-  appropriate also for distributed systems
- 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
- Another acronym SDL (“System Design Languages”)

SDL - Specification and Description *Language*

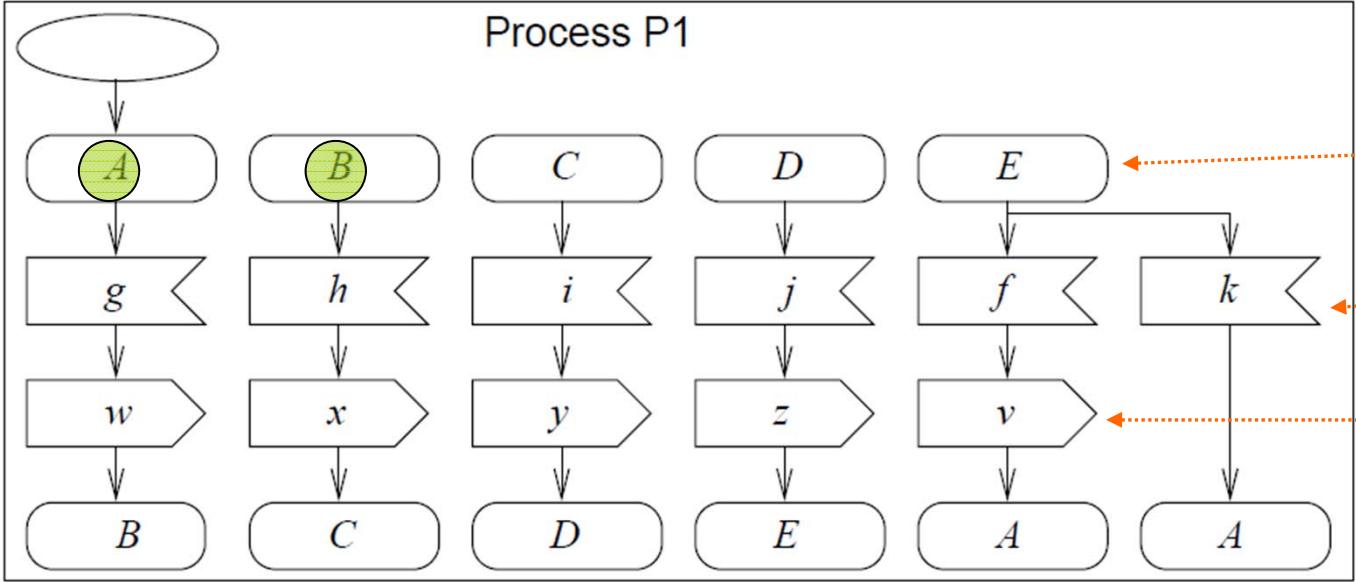
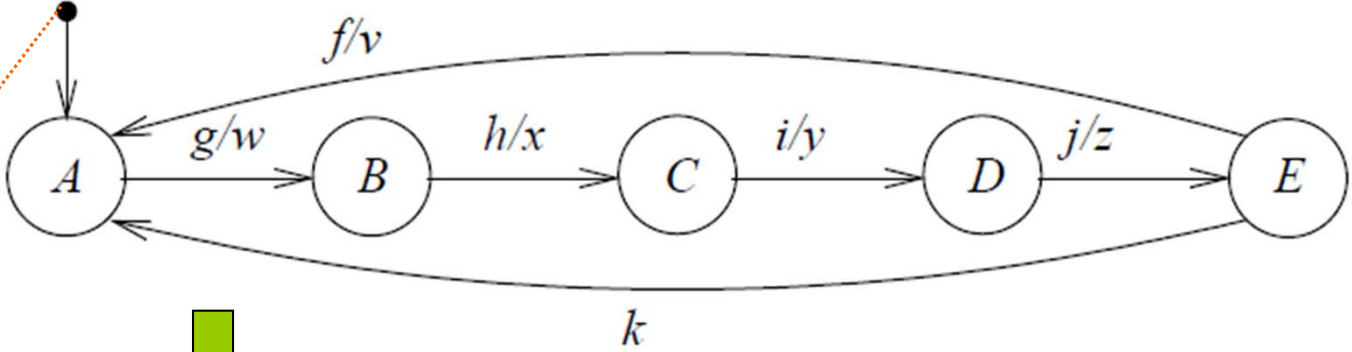
- Provides textual (tool processing) and graphical formats (user interaction)
- Ability to be used as a wide spectrum language from requirements to implementation
- Just like StateCharts, it is based on the CFSM (Communicating FSM) model of computation; each FSM is called a **process**.
- With SDL the protocol behaviour is completely specified by communicating FSM.
- The formal basis of SDL enables the use of code generation tool chains, which allows an automated implementation of the specification.

SDL - Specification and Description *Language*

- However, it uses **message passing** instead of shared memory for communications
- SDL supports operations on data
- object oriented description of components.

REVIEW

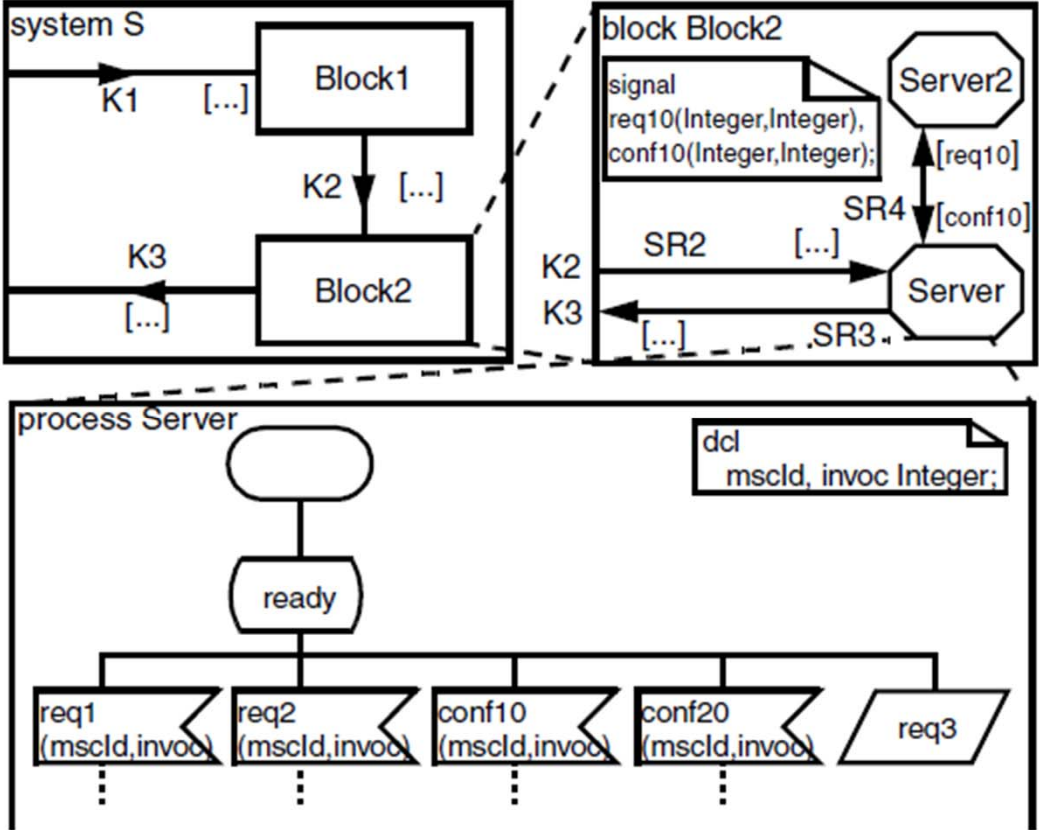
SDL-representation of FSMs/processes REVIEW



state
input
output

Hierarchy of a SDL specification

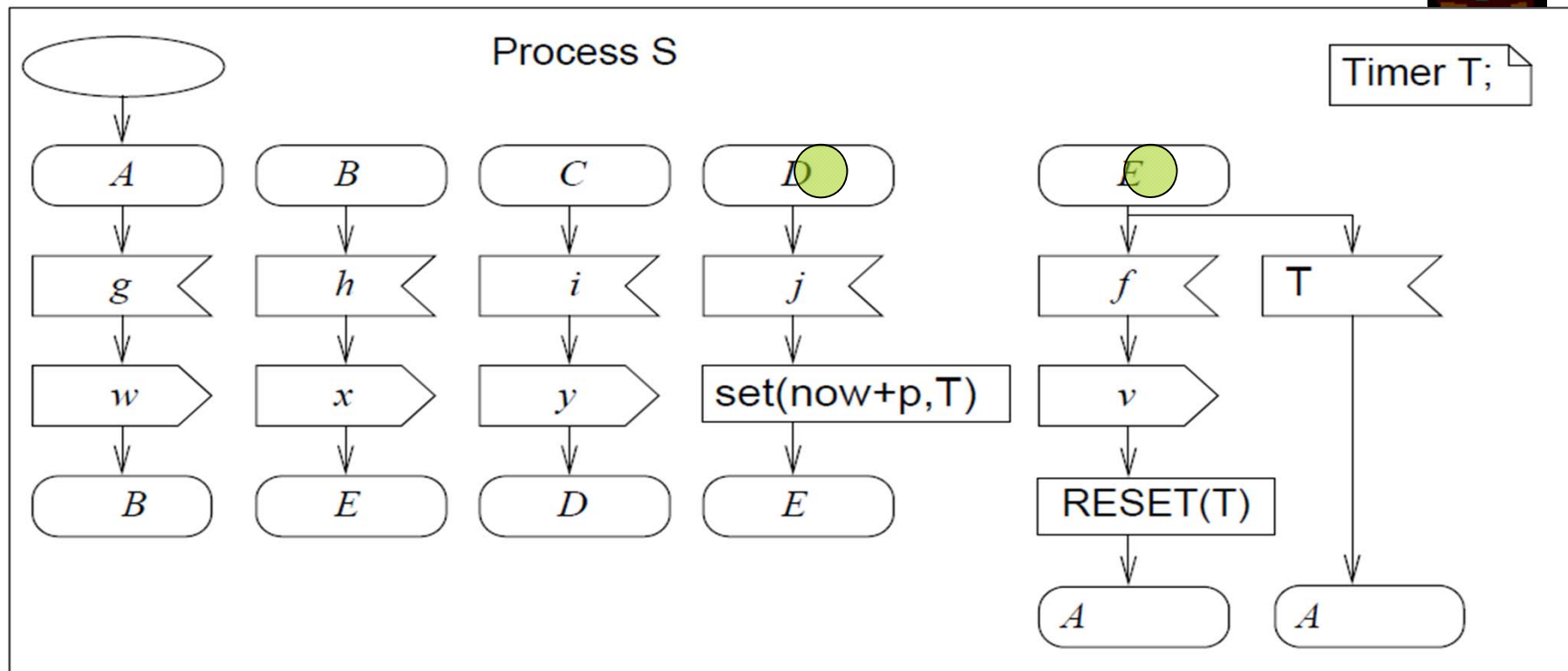
REVIEW



Timers

REVIEW

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



SDL application

REVIEW

The semantics of SDL defines the state space of the specification. This state space can be used for various analyses and transformation techniques, e.g.:

- state space exploration, simulation
- checking the SDL-specification for deadlocks/livelocks
- deriving test cases automatically
- code generation for an executable prototype or end system

Summary

REVIEW

- MoC: finite state machine components
+ non-blocking message passing communication
- Representation of processes
- Communication & block diagrams
- Timers and other language elements
- Excellent for distributed applications (e.g., *Integrated Services Digital Network* (ISDN))
- Commercial tools available from SINTEF, Telelogic, Cinderella ([//www.cinderella.dk](http://www.cinderella.dk))

Message Sequence Charts

Motivation: Scenario-based Specification

- “Well, the controller of my ATM can be in waiting-for-user-input mode or in connecting-to-bank-computer mode or in delivering-money-mode; in the first case, here are the possible inputs and the ATM’s reactions, . . .; in the second case, here is what happens, . . ., etc.”.

VS.

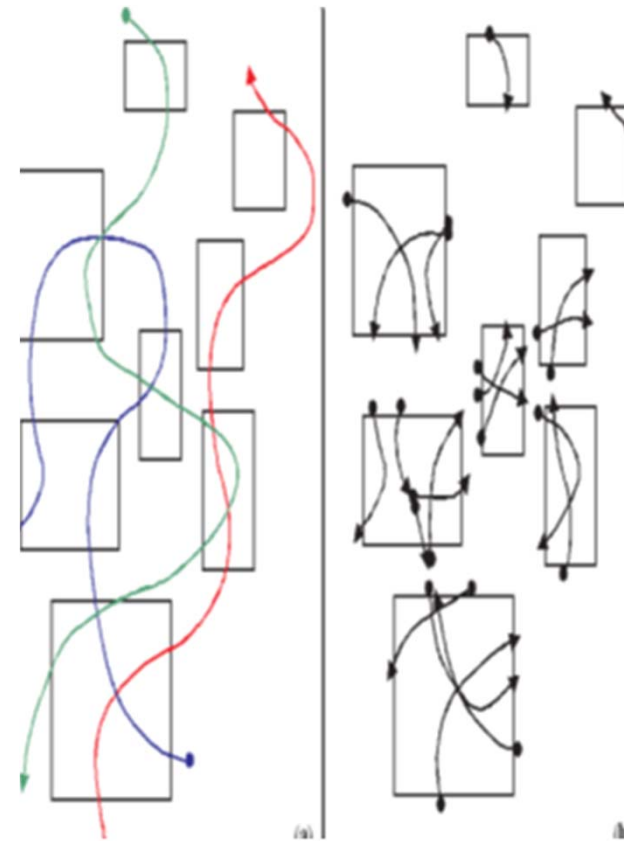
- “If I insert my card, and then press this button and type in my PIN, then the following shows up on the display, and by pressing this other button my account balance will show”.

Motivation: Scenario-based Specification

- **Claim:** it is more natural to *describe* and discuss the reactive behavior of a system by the **scenarios** it enables rather than by the state-based reactivity of each of its components.
- In order to *implement* the system, as opposed to stating its required behavior or preparing test suites, **state-based modeling** is needed, whereby we *must specify for each component the complete array of possibilities for incoming events and changes and the component's reactions* to them.

Inter-Object vs Intra-Object

- **inter-object approach**
‘one story for all relevant objects’
scenario-based behavioral descriptions, which cut across the boundaries of the components (or objects) of the system, in order to provide coherent and comprehensive descriptions of scenarios of behavior
- **intra-object approach**
‘all pieces of stories for one object’
statebased behavioral descriptions, which remain within the component, or object, and are based on providing a complete description of the reactivity of each one



Message Sequence Charts

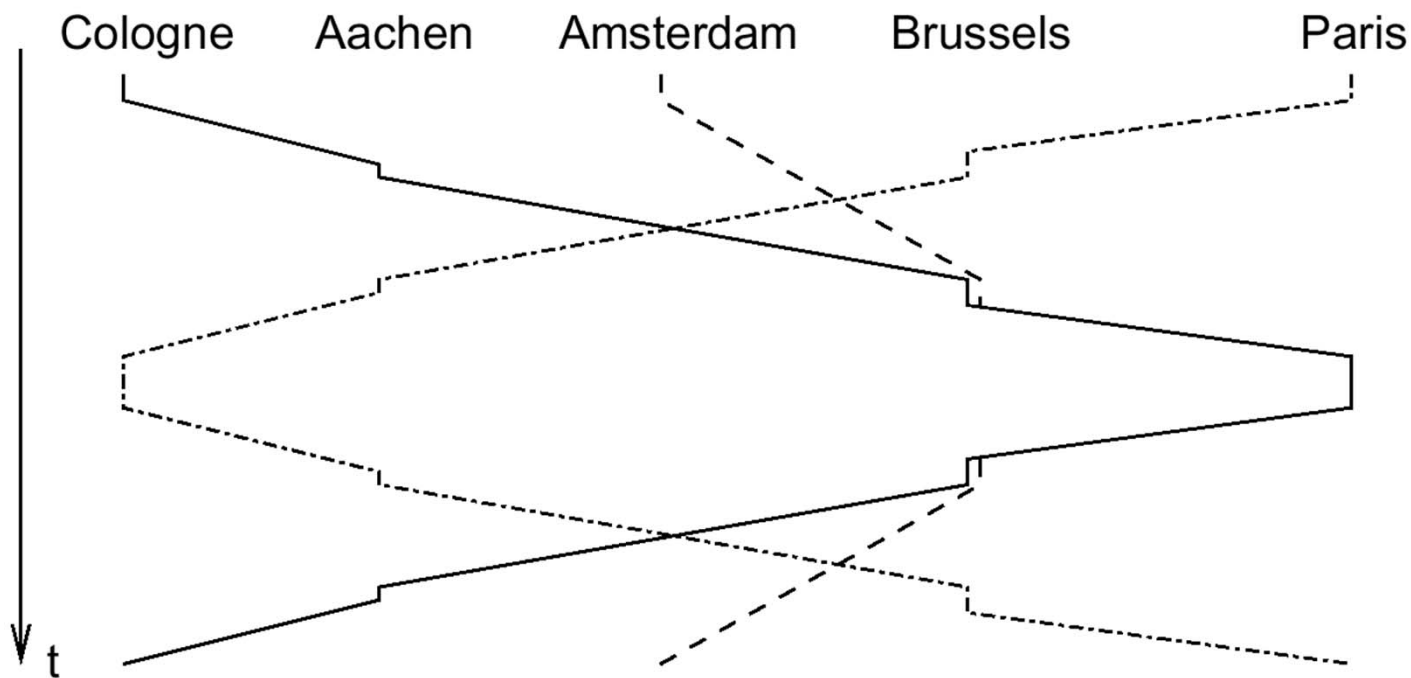
- Message Sequence Charts (MSC) is a language to describe the **interaction between a number of independent message-passing instances.**
- Defined by ITU (International Telecommunication Union)
 - Z.120 recommendation
- MSC is
 - a scenario language
 - graphical
 - formal
 - practical
 - widely applicable

MSC

- In telecommunication industry, MSCs are the first choice to describe example traces of the system under development. MSCs are used throughout the whole protocol life cycle from requirements analysis to testing.
- To define longer traces hierarchically, simple MSCs can be composed by operators in high-level MSC (HMSC).
- Message Sequence Charts may be used for requirement specification, simulation and validation, test-case specification and documentation of real-time systems.

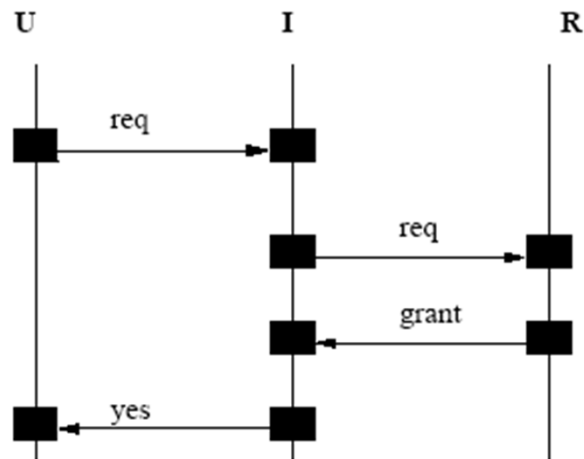
Message sequence charts (MSC)

- Graphical means for representing schedules; time used vertically, “geographical” distribution horizontally.



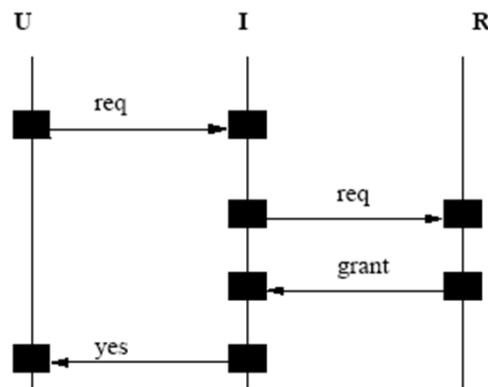
MSC: Example

- user (U) sends a **request** to an **interface (I)** to gain access to a **resource R**
- interface in turn sends a request to the resource, receives “grant” as a response
- Sends “yes” to U.

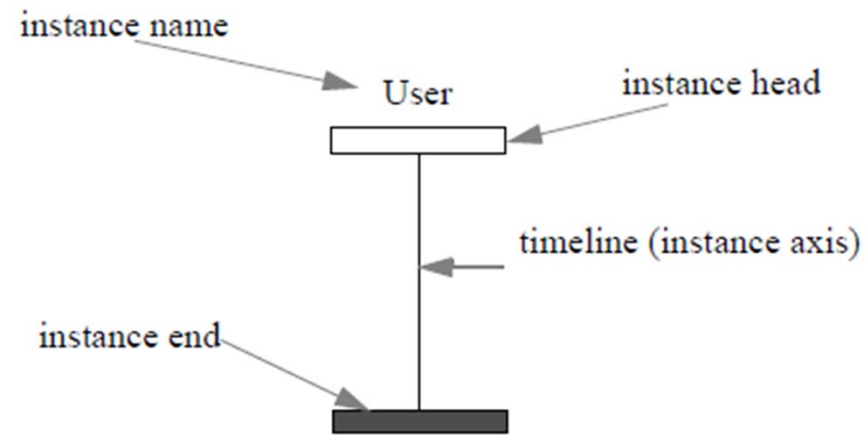


Visual representation

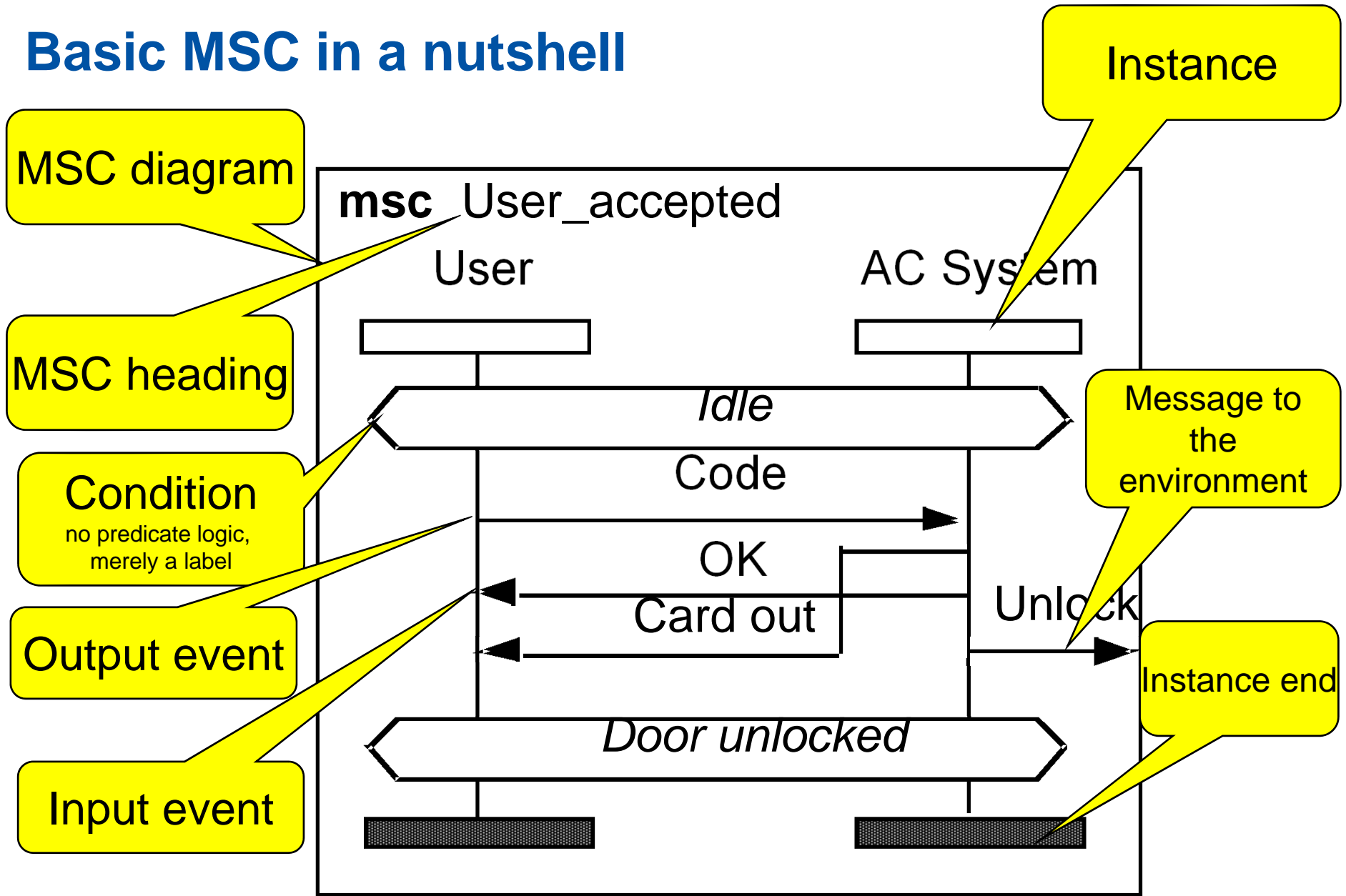
- **Processes** (instances) – vertical lines or „life lines“
- time flows downwards along each life-line
- **Messages** – horizontal arrows across the life lines representing „causal links“ from a send event (the source of the arrow) to the corresponding receive event (the target of the arrow)
- label on the arrow denotes the message being **transmitted**



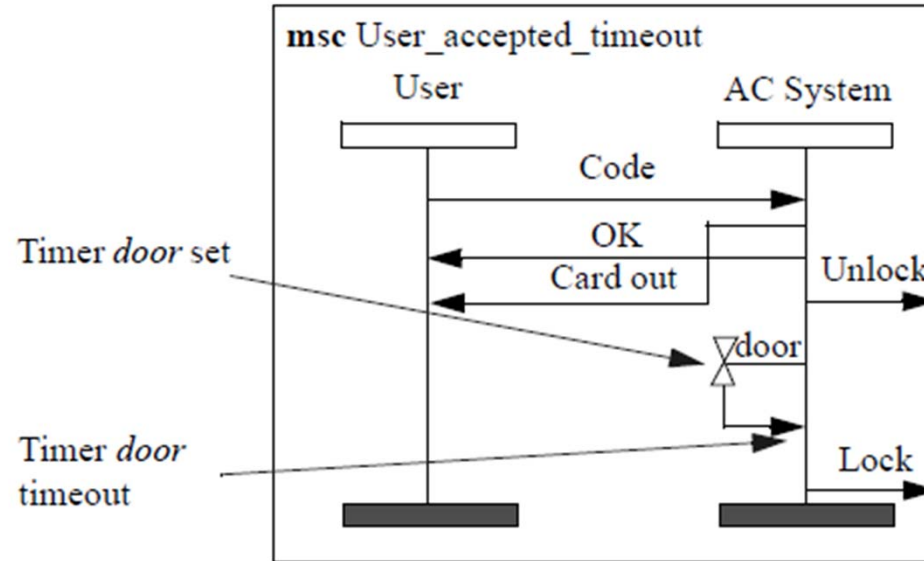
Instance



Basic MSC in a nutshell

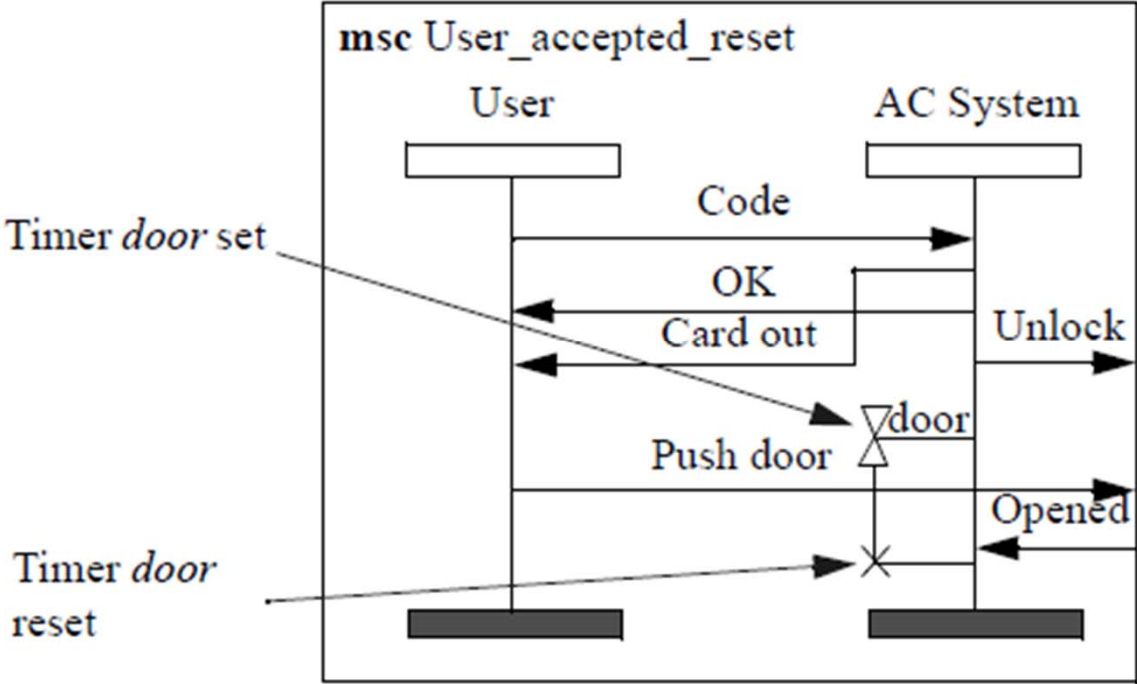


Timer set and timeout



- User is accepted → forget to push the door
- AC system will detect this through the expiration of the timer → Lock

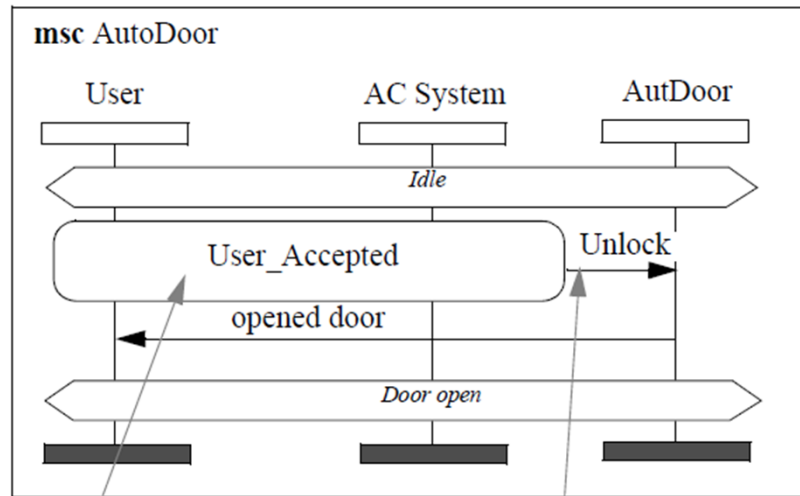
Preferred situation



MSC reference

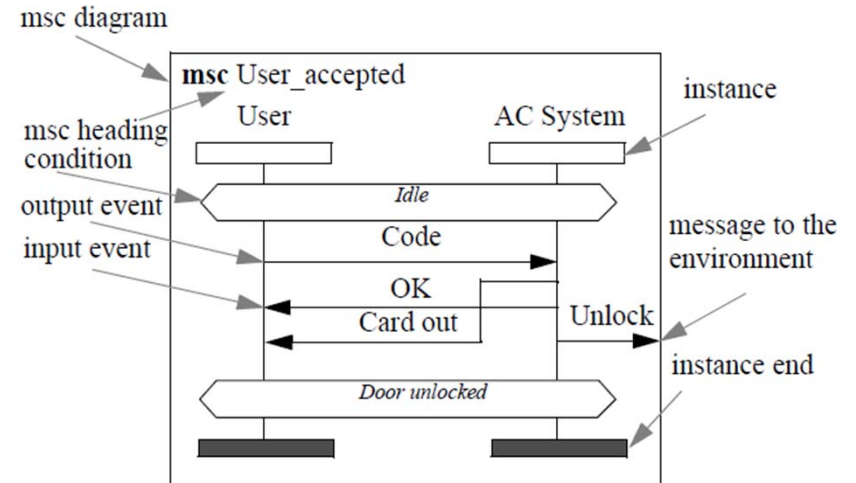
- In almost all description/programming/specification languages there is a way to **isolate subparts** of the description in a separate named construct (procedures, functions, classes, packages)
- In MSC there are MSCs which can be **referred** from other MSCs.

MSC reference



msc reference

actual gate

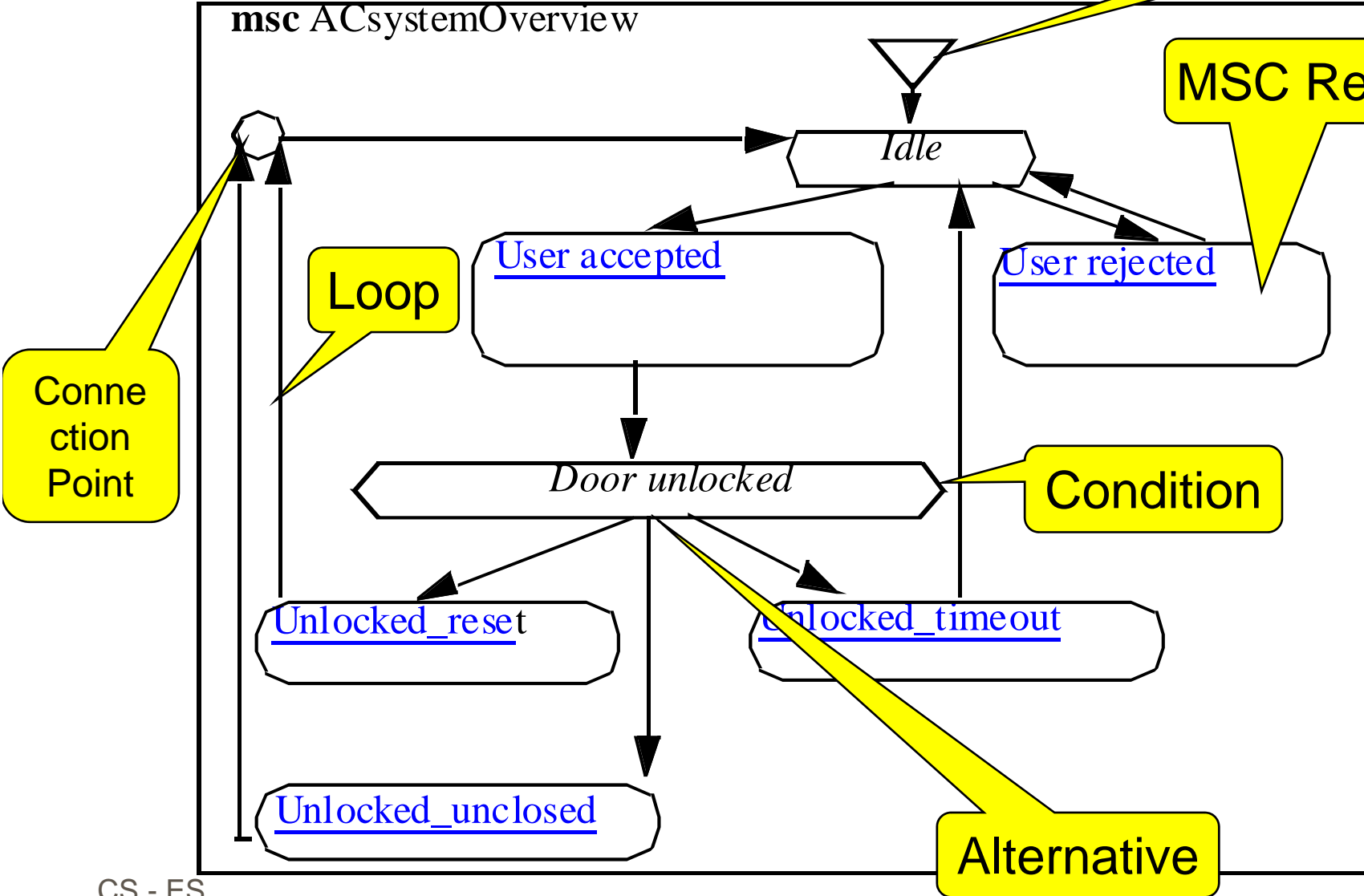


- Assume that the scenario where the user is accepted is **part of a larger context** where there is an automatic door. When the **door is unlocked it automatically opens.**
- The **MSC reference symbol** is a box with rounded corners.

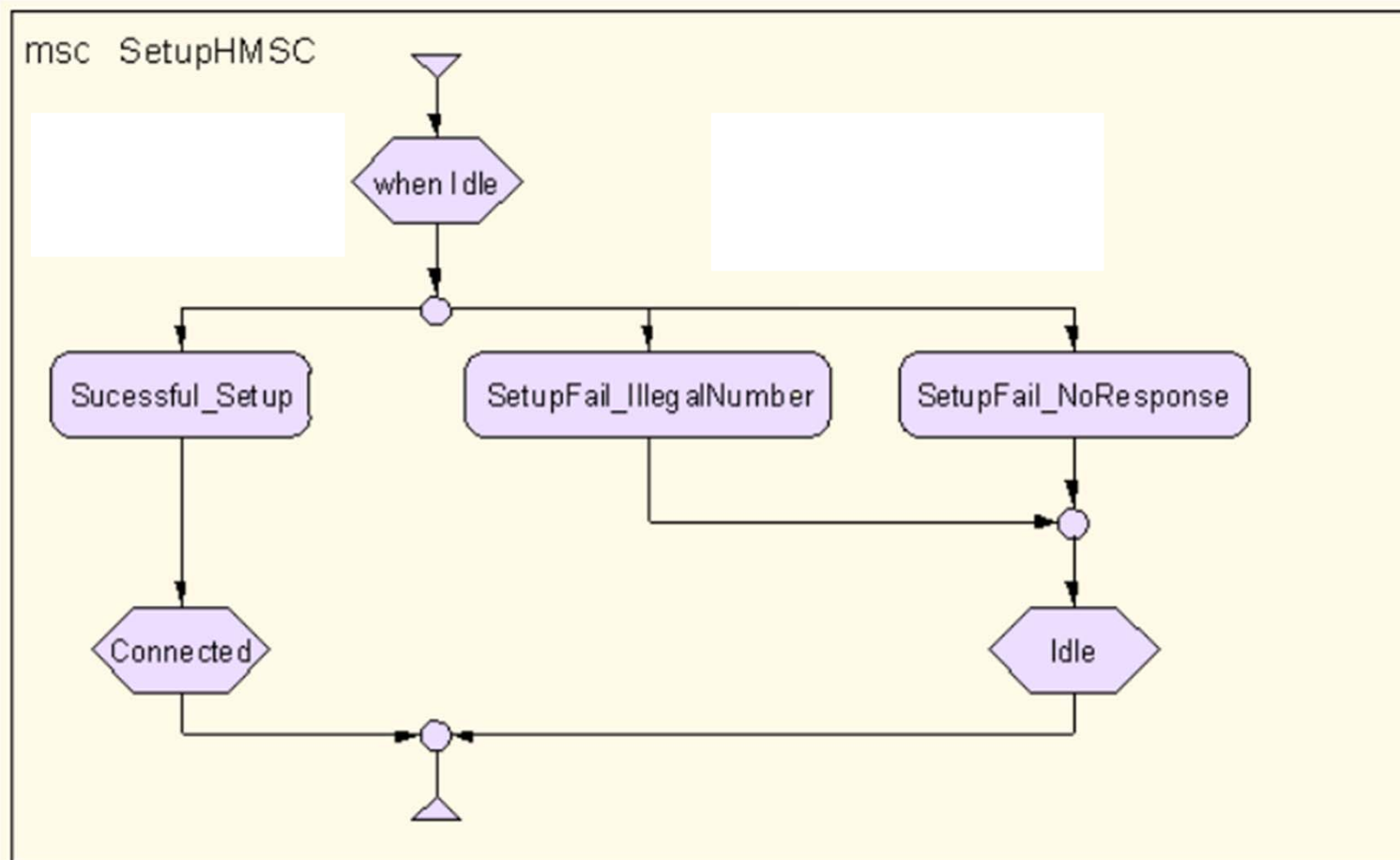
HMSC (High Level MSC)

HMSC Start

MSC Reference

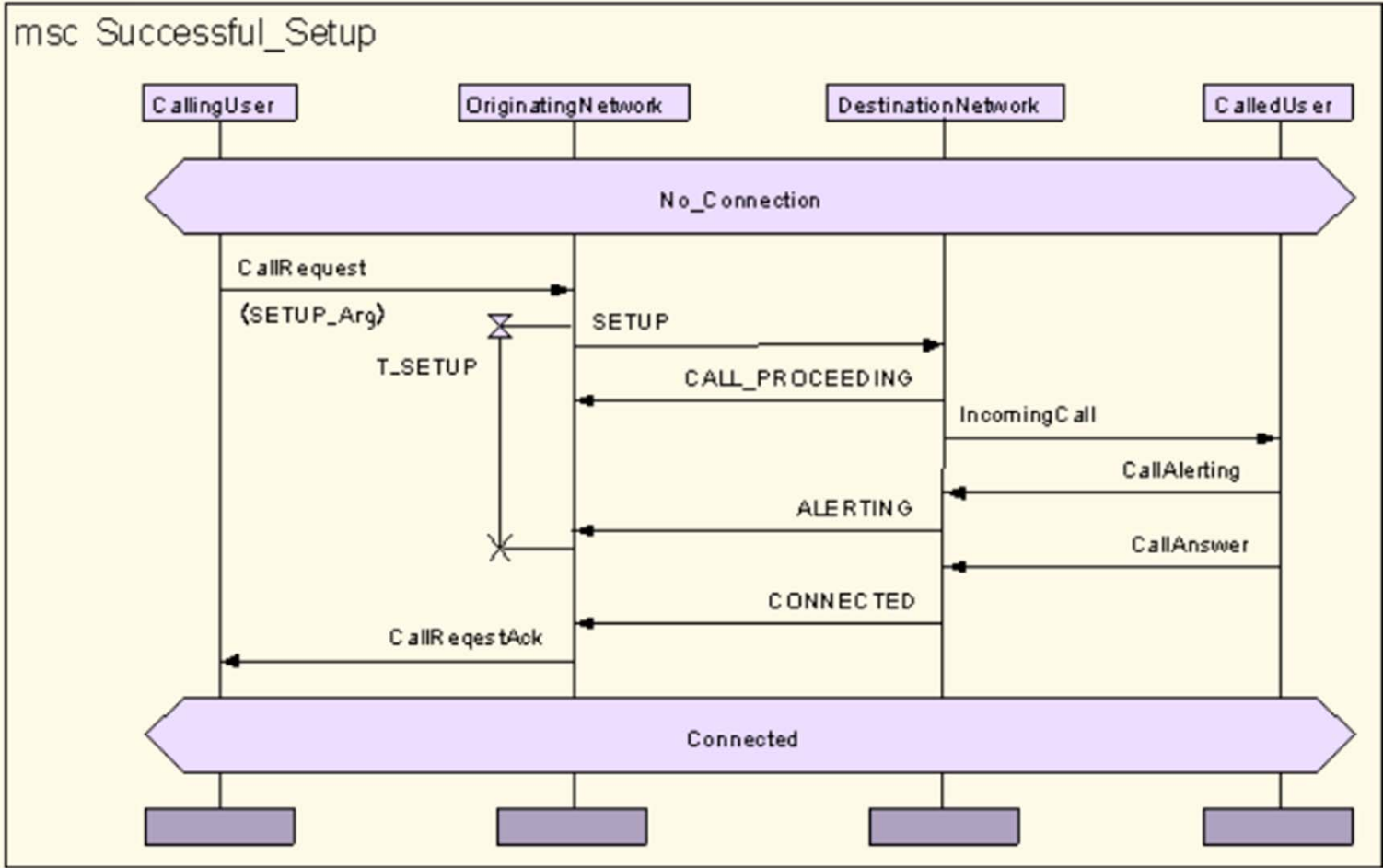


HMSC



The symbols named "Successful_Setup", "SetupFail_IllegalNumber" and "SetupFail_NoResponse" are references to interactions described in basic MSC diagrams.

Basic MSC for a "Successful_Setup" connection setup scenario



What is new in MSC-2000 relative to MSC-96

- Improved structural concepts and object orientation
- Data with the data language of your choice
- Time observations and time constraints (Observing absolute/relative time)
- Method calls for more synchronizing communication

Data in MSC-2000

- MSC has no data language of its own!
- MSC has parameterized data languages such that
 - fragments of your favorite (data) language can be used
 - C, C++, **SDL**, Java, ...
 - MSC can be parsed without knowing the details of the chosen data language
 - the interface between MSC and the chosen data language is given in a set of interface functions

Data Flow Models

Data flow modeling

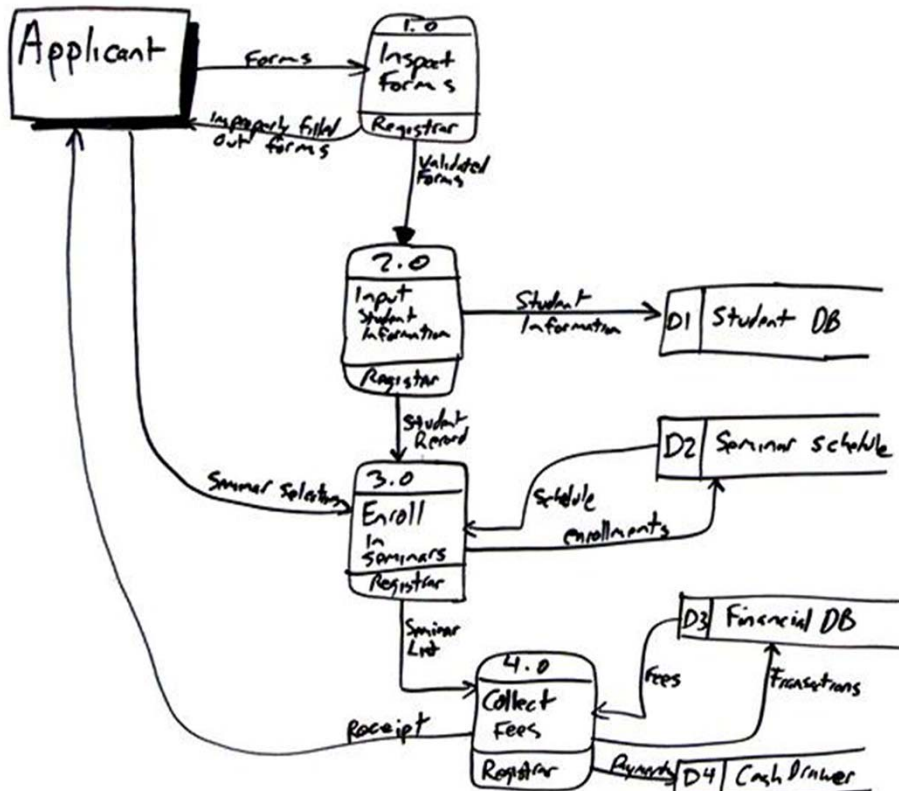
- **Def.:** The process of identifying, modeling and documenting how data moves around an information system.

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).

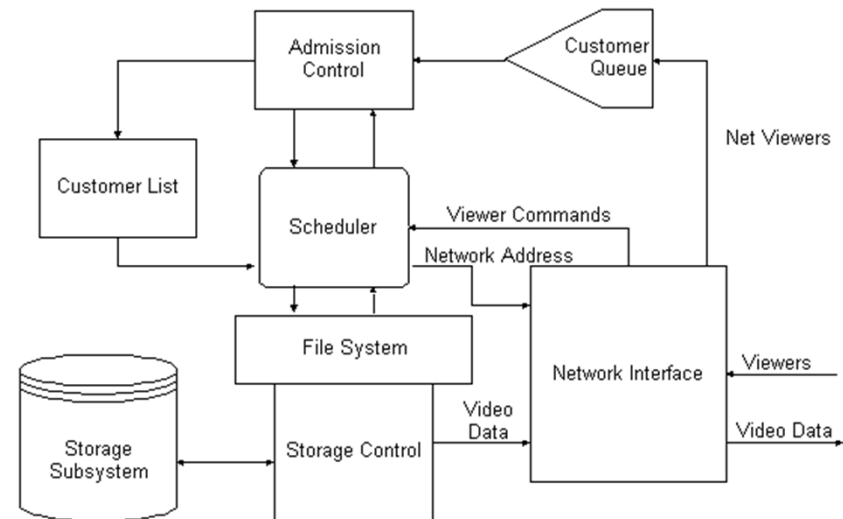
Data flow as a “natural” model of applications

Registering for courses



<http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm>

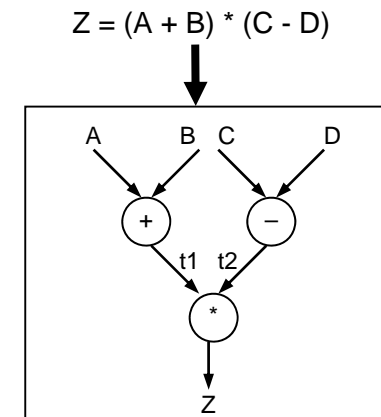
Video on demand system



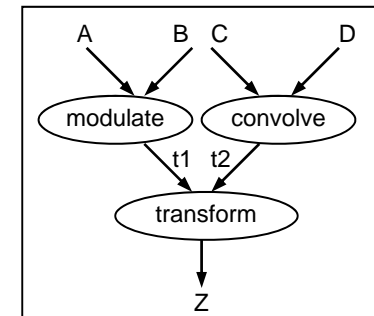
www.ece.ubc.ca/~irenek/techpaps/vod/vod.html

Dataflow model

- Nodes **represent transformations**
 - May execute concurrently
- Edges **represent flow of tokens (data)** from one node to another
 - May or may not have token at any given time
- When all of node's input edges have **at least one token**, node may fire
- When node fires, it **consumes input tokens processes** transformation and **generates output token**
- Nodes may fire **simultaneously**
- Several **commercial tools support graphical** languages for capture of dataflow model
 - Can automatically translate to concurrent process model for implementation
 - Each node becomes a process



Nodes with arithmetic transformations



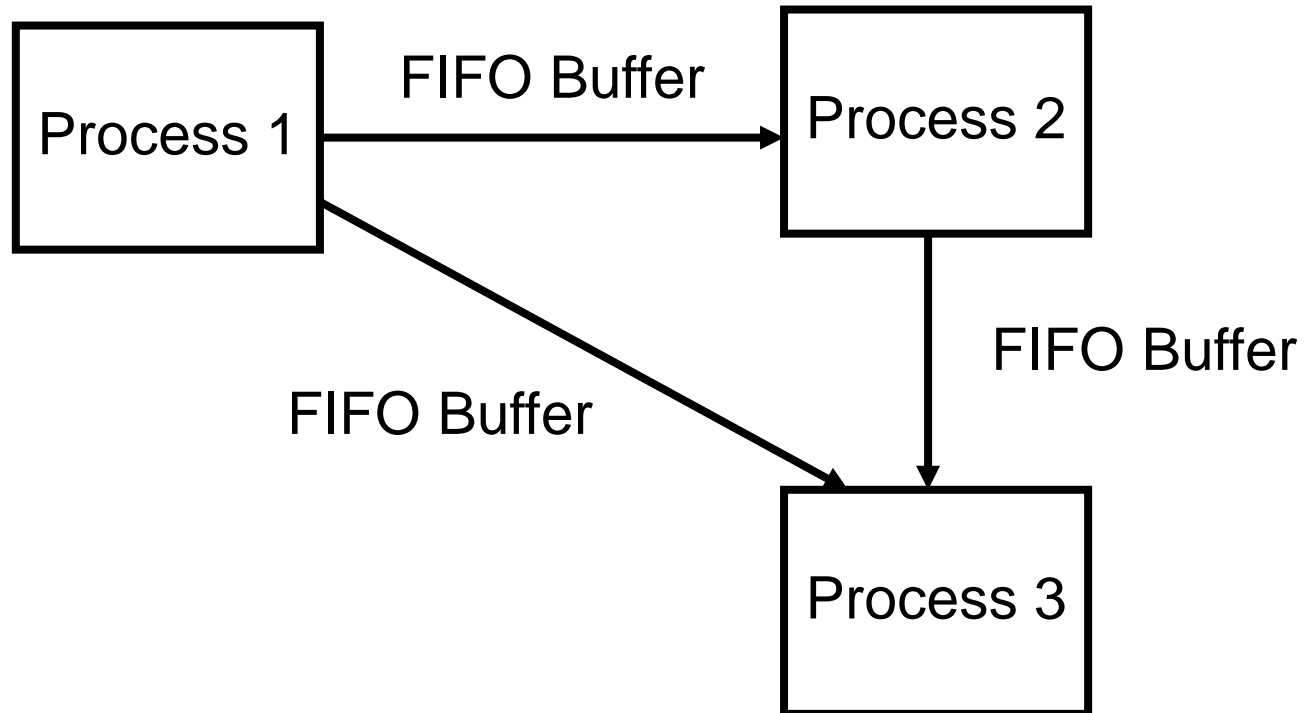
Nodes with more complex transformations

Philosophy of Dataflow Languages

- Drastically different way of looking at computation
- Von Neumann imperative language style: program counter controls everything
- Dataflow language: movement of data the priority
- Scheduling responsibility of the system, not the programmer

Dataflow Language Model

- Processes communicating through FIFO buffers



Dataflow Languages

- Every process runs simultaneously
- Processes can be described with imperative code
- Compute ... receive ... compute ... transmit

Dataflow Communication

- Communication is *only* through buffers
- Buffers usually treated as unbounded for flexibility
- Destructive read: reading a value from a buffer removes the value
- Fundamentally concurrent: should map more **easily to parallel hardware**

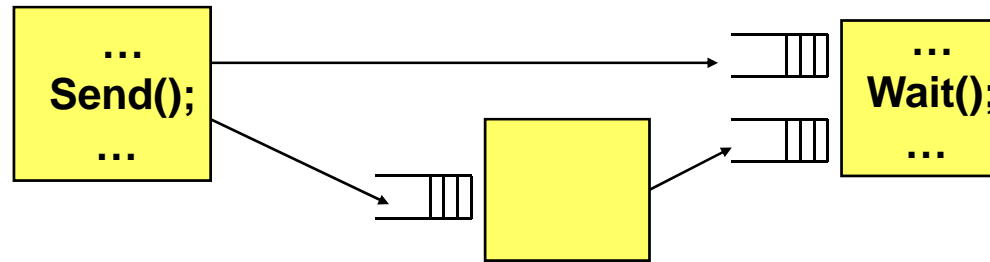
Applications of Dataflow

- signal-processing applications
- Anything that deals with a continuous stream of data
- Becomes easy to parallelize
- Buffers typically used for signal processing applications anyway

Applications of Dataflow

- Perfect fit for block-diagram specifications
 - Circuit diagrams
 - Linear/nonlinear control systems
 - Signal processing
- Suggest dataflow semantics
- Common in Electrical Engineering
- Processes are blocks, connections are buffers

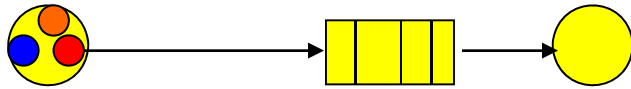
Kahn Process Networks



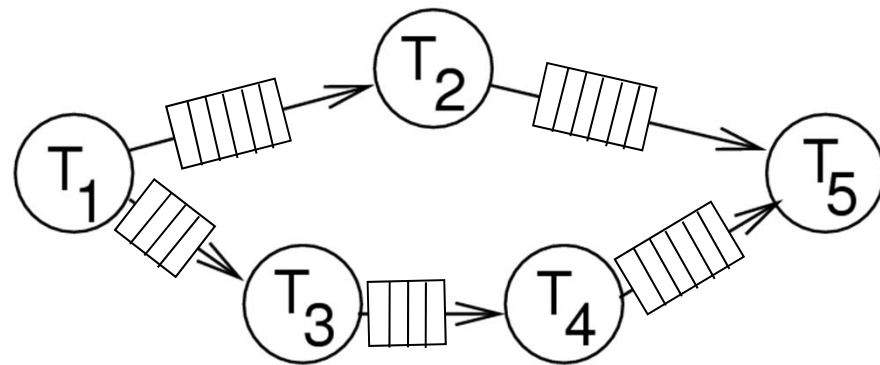
- Proposed by Kahn in 1974 as a general-purpose scheme for parallel programming
- Theoretical foundation for dataflow
- Unique attribute: deterministic
- Difficult to schedule
- Too flexible to make efficient, not flexible enough for a wide class of applications
- Never put to widespread use

Reference model for data flow: Kahn process networks

For asynchronous message passing:
communication between tasks is buffered

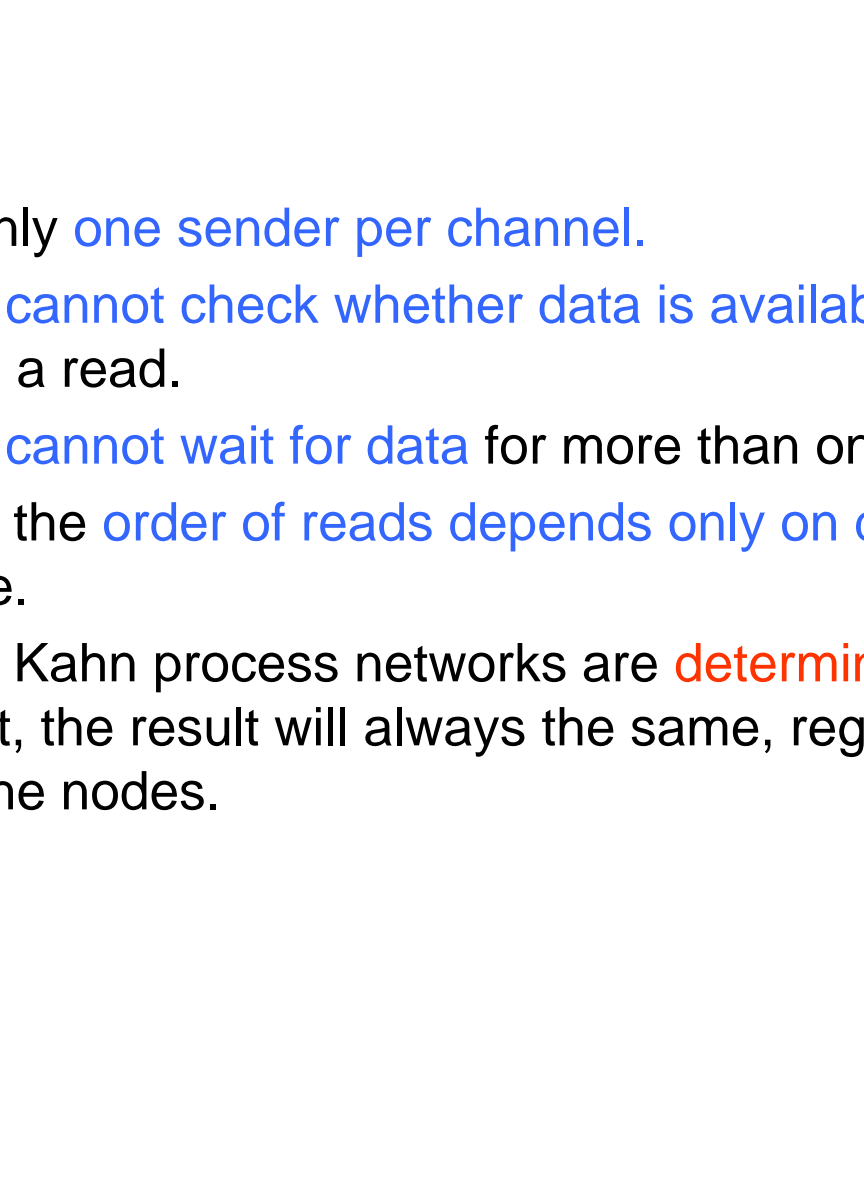


Special case: Kahn process networks:
executable task graphs;
Communication via infinitely large FIFOs



Properties of Kahn process networks (2)

- There is only **one sender per channel**.
- 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 **deterministic (!)**; for a given input, the result will always be the same, regardless of the speed of the nodes.



This is the
key beauty
of KPNs!

Kahn Process Networks

- Key idea:

Reading an empty channel blocks until data is available

- No other mechanism for sampling communication channel's contents
- Can't check to see whether buffer is empty
- Can't wait on **multiple channels at once**

Kahn Processes

- A C-like function (Kahn used Algol)
- Arguments include FIFO channels
- Language augmented with `send()` and `wait()` operations that write and read from channels

Sample parallel program S

```
Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer in U,V; integer out W) ;
    Begin integer I ; logical B ;
        B := true ;
        Repeat Begin
(4)     I := if B then wait(U) else wait(V) ;
(7)     print (I) ;
(5)     send I on W ;
        B := ¬B ;
        end ;
    End ;
Process g(integer in U ; integer out V, W) ;
    Begin integer I ; logical B ;
        B := true ;
        Repeat Begin
            I := wait (U) ;
            if B then send I on V else send I on W ;
            B := ¬B ;
        End ;
    End ;
(3) Process h(integer in U;integer out V; integer INIT);
    Begin integer I ;
        send INIT on V ;
        Repeat Begin
            I := wait(U) ;
            send I on V ;
        End ;
    End ;
    Comment : body of mainprogram ;
(6) f(Y,Z,X) par g(X,T1,T2) par h(T1,Y,0) par h(T2,Z,1)
    End ;
```

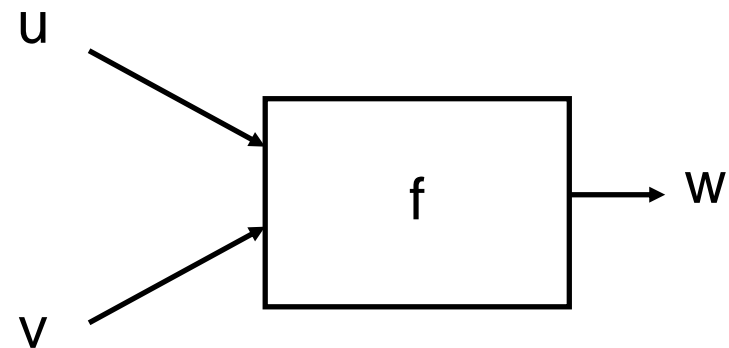
(1) ... channel declation

processes f, g, h are declared

A Kahn Process

- From Kahn's original 1974 paper

```
process f(in int u, in int v, out int w)
{
  int i; bool b = true;
  for (;;) {
    i = b ? wait(u) : wait(v);
    printf("%i\n", i);
    send(i, w);
    b = !b;
  }
}
```



What does this do?

Process alternately reads from u and v, prints the data value, and writes it to w

A Kahn Process

- From Kahn's original 1974 paper:

```
process f(in int u, in int v, out int w)
{
  int i; bool b = true;
  for (;;) {
    i = b ? wait(u) : wait(v);
    printf("%i\n", i);
    send(i, w);
    b = !b;
  }
}
```

Process interface includes FIFOs

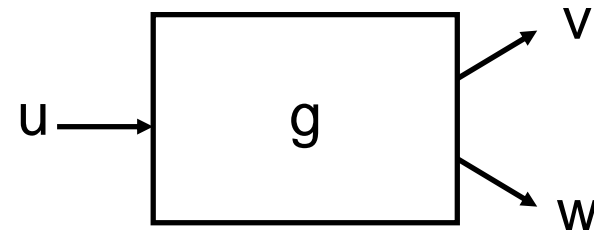
wait() returns the next token in an input FIFO, blocking if it's empty

send() writes a data value on an output FIFO

A Kahn Process

- From Kahn's original 1974 paper:

```
process g(in int u, out int v, out int w)
{
  int i; bool b = true;
  for(;;) {
    i = wait(u);
    if (b) send(i, v); else send(i, w);
    b = !b;
  }
}
```



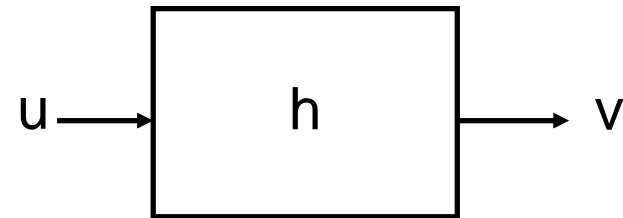
What does this do?

Process reads from u and alternately copies it to v and w

A Kahn Process

- From Kahn's original 1974 paper:

```
process h(in int u, out int v, int init)
{
  int i = init;
  send(i, v);
  for(;;) {
    i = wait(u);
    send(i, v);
  }
}
```



What does this do?

Process sends initial value, then passes through values.

Sample parallel program S

```
Begin
(1) Integer channel X, Y, Z, T1, T2 ;
(2) Process f(integer in U,V; integer out W) ;
    Begin integer I ; logical B ;
        B := true ;
        Repeat Begin
(4)     I := if B then wait(U) else wait(V) ;
(7)     print (I) ;
(5)     send I on W ;
        B := ¬B ;
        end ;
    End ;
Process g(integer in U ; integer out V, W) ;
    Begin integer I ; logical B ;
        B := true ;
        Repeat Begin
            I := wait (U) ;
            if B then send I on V else send I on W ;
            B := ¬B ;
        End ;
    End ;
(3) Process h(integer in U;integer out V; integer INIT);
    Begin integer I ;
        send INIT on V ;
        Repeat Begin
            I := wait(U) ;
            send I on V ;
        End ;
    End ;
    Comment : body of mainprogram ;
(6) f(Y,Z,X) par g(X,T1,T2) par h(T1,Y,0) par h(T2,Z,I)
    End ;
```

(1) ... channel declation

processes f, g, h are declared

(6) ... body of the main program:

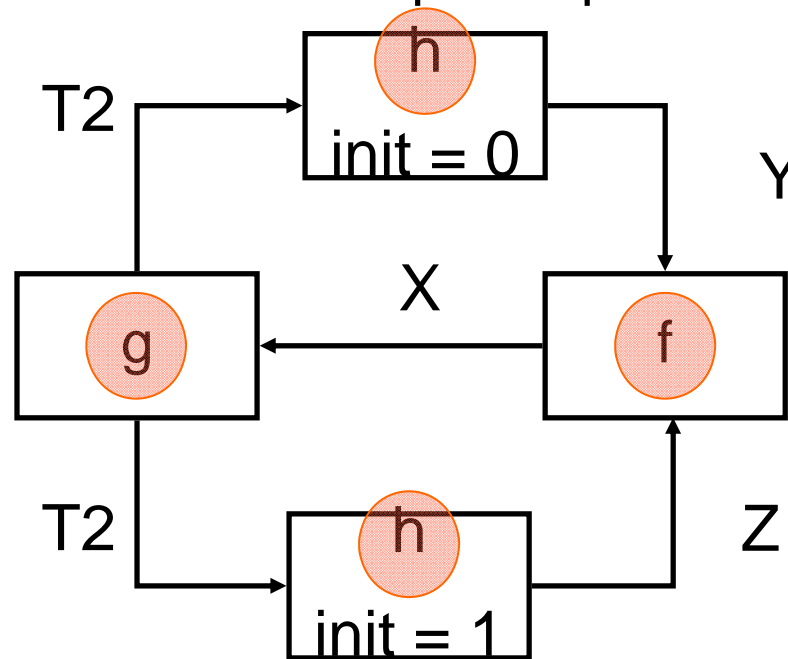
- calling instances of the processes
- actual names of the channels are bound to the formal parameters
- infix operator **par** → **concurrent activation** of the processes

A Kahn System

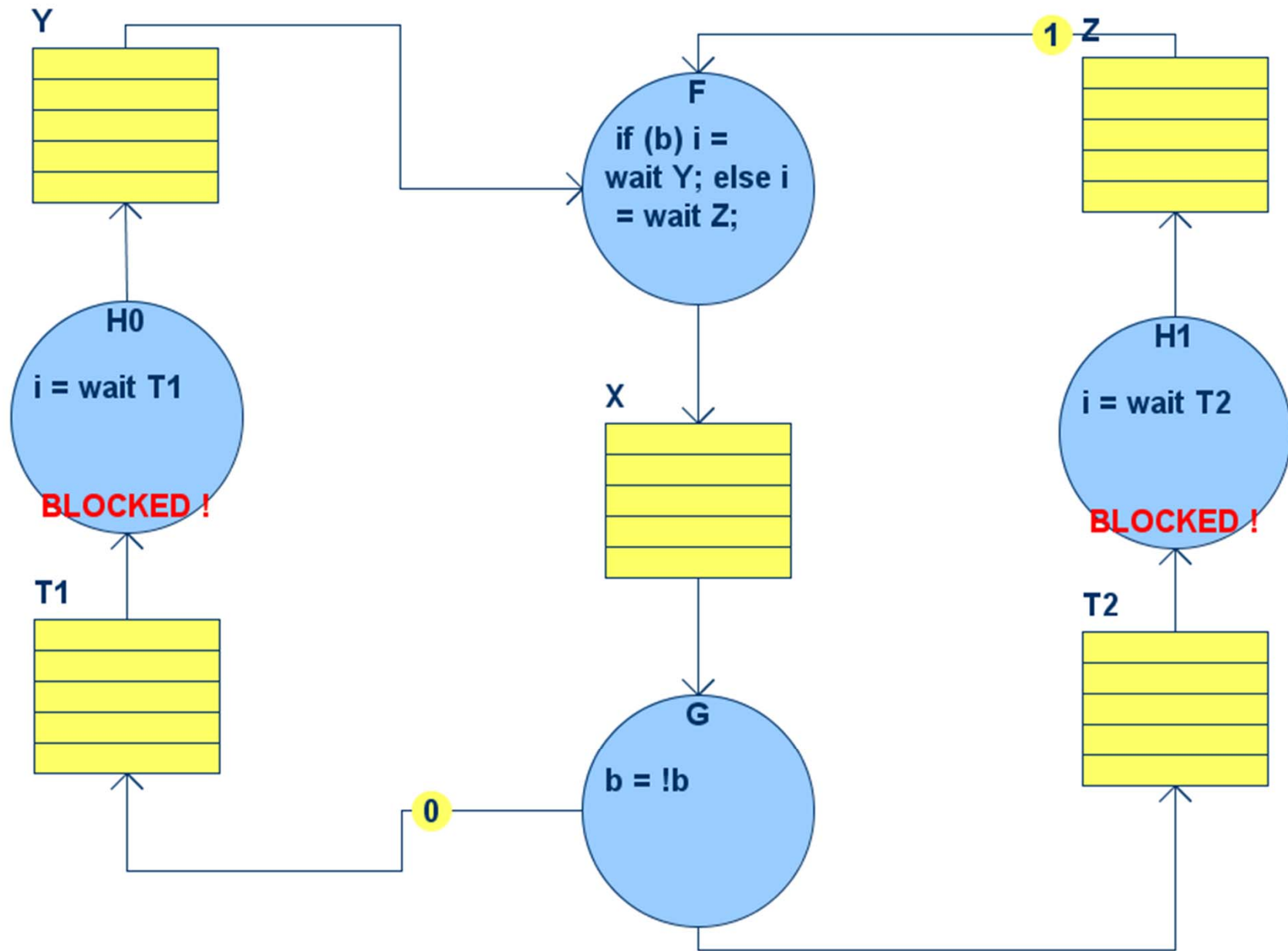
- What does this do?

Prints an alternating sequence of 0's and 1's

Emits a 1 then copies input to output

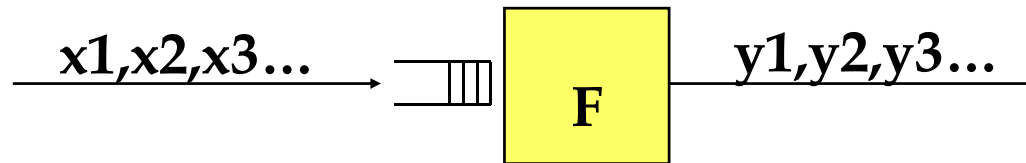


Emits a 0 then copies input to output



Kahn network animation

Determinism



- Process: “continuous mapping” of input sequence to output sequences
- Continuity: process uses prefix of input sequences to produce prefix of output sequences. Adding more tokens does not change the tokens already produced
- The state of each process depends on token values rather than their arrival time
- Unbounded FIFO: the speed of the two processes does not affect the sequence of data values

Determinism

- Another way to see it:
- If I'm a process, I am only affected by the sequence of tokens on my inputs
- I can't tell whether they arrive early, late, or in what order
- I will behave the same in any case
- Thus, the **sequence of tokens I put on my outputs** is the same **regardless** of the **timing of the tokens on my inputs**

Kahn Process Networks

- Their beauty is that the scheduling algorithm does not affect their functional behavior
- Difficult to schedule because of need to balance relative process rates
- System inherently gives the scheduler few hints about appropriate rates

Synchronous Dataflow (SDF)

- Edward Lee and David Messerchmitt, Berkeley, 1987
Ptolemy System
- Restriction of Kahn Networks to allow compile-time scheduling
- Basic idea: each process reads and writes a fixed number of tokens each time it fires:

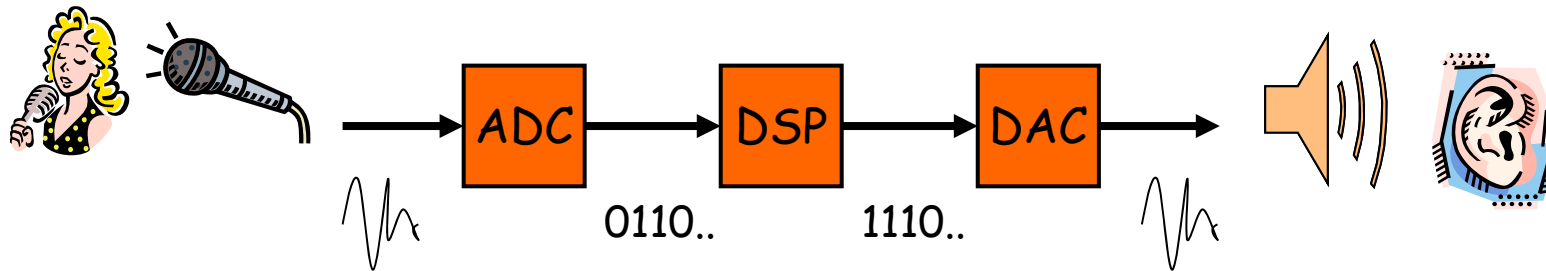
loop

read 3 A, 5 B, 1 C ...compute...write 2 D, 1 E, 7 F

end loop

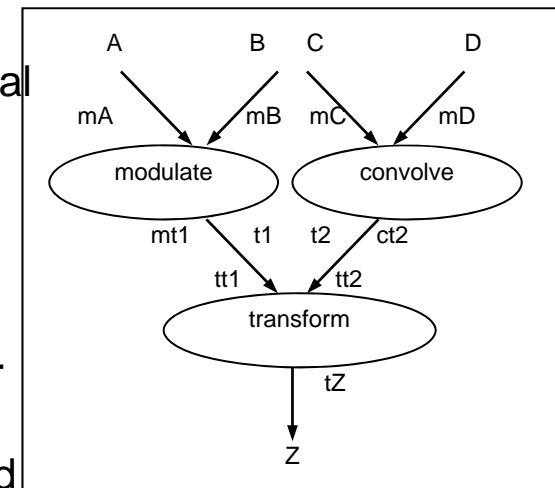
Synchronous dataflow

- With digital signal-processors (DSPs), data flows at fixed rate



Synchronous dataflow

- Multiple tokens consumed and produced per firing
- Synchronous dataflow model takes advantage of this
 - Each edge labeled with number of tokens consumed/produced each firing
 - Can statically schedule nodes, so can easily use sequential program model
 - Don't need real-time operating system and its overhead
- Algorithms developed for scheduling nodes into “single-appearance” schedules
 - Only **one statement** needed to call each node's associated procedure
 - Allows procedure inlining without code explosion, thus reducing overhead even more



Synchronous dataflow