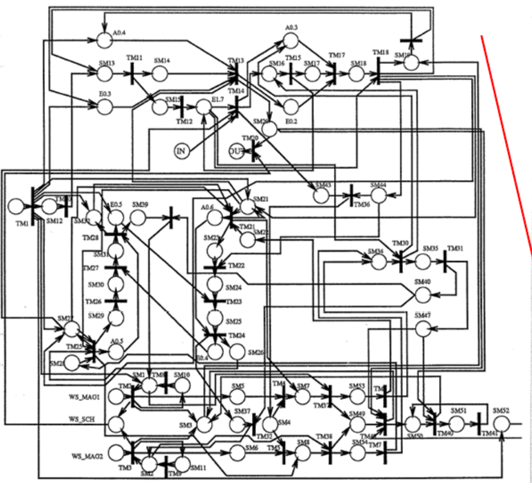
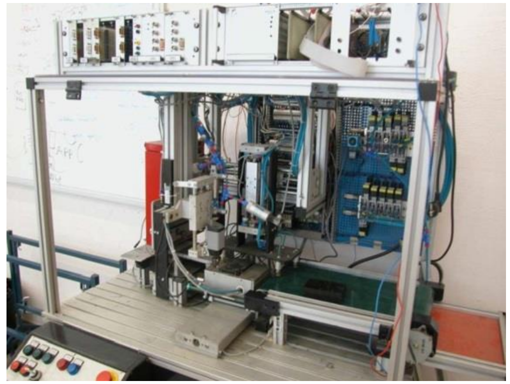
Embedded Systems 7

Production system

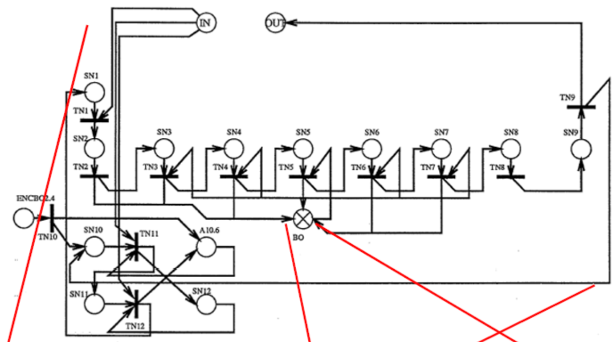
A modelbased realtime faultdiagnosis system for technical processes
Ch. Steger, R. Weiss

magazine/depot

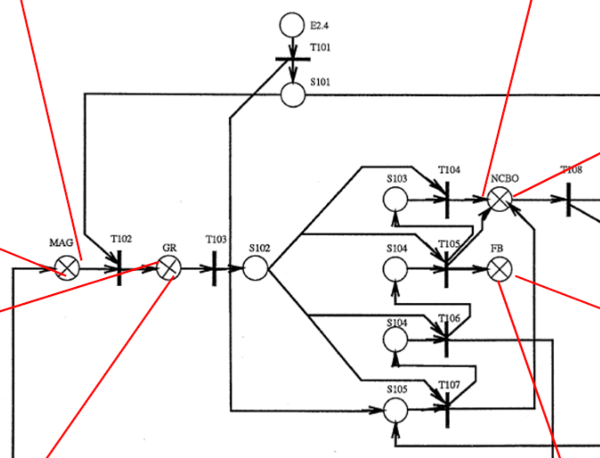


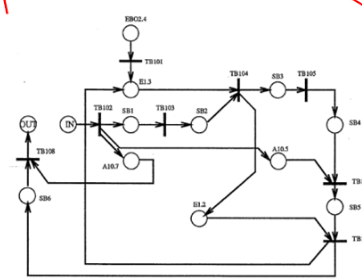


NC axis



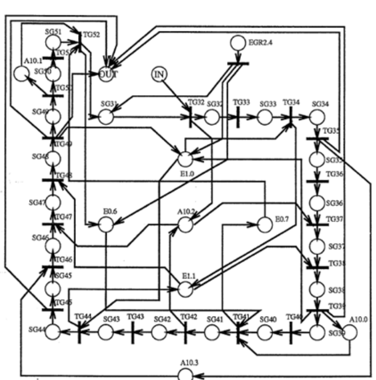
top level

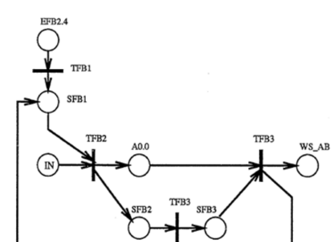




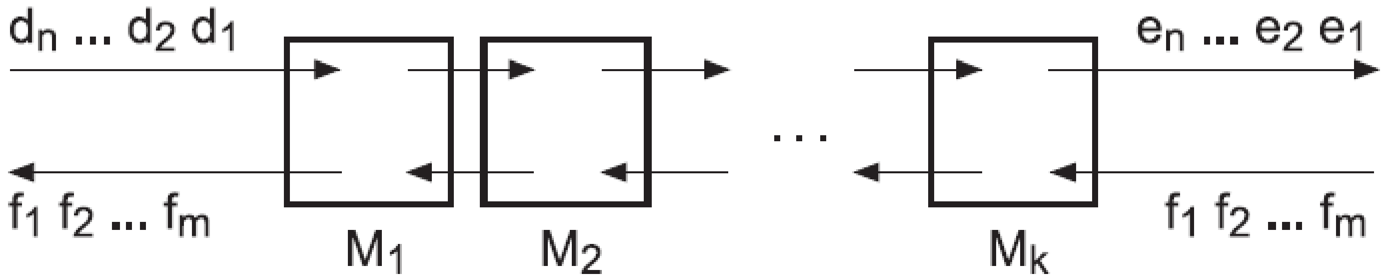
drilling machine

gripper



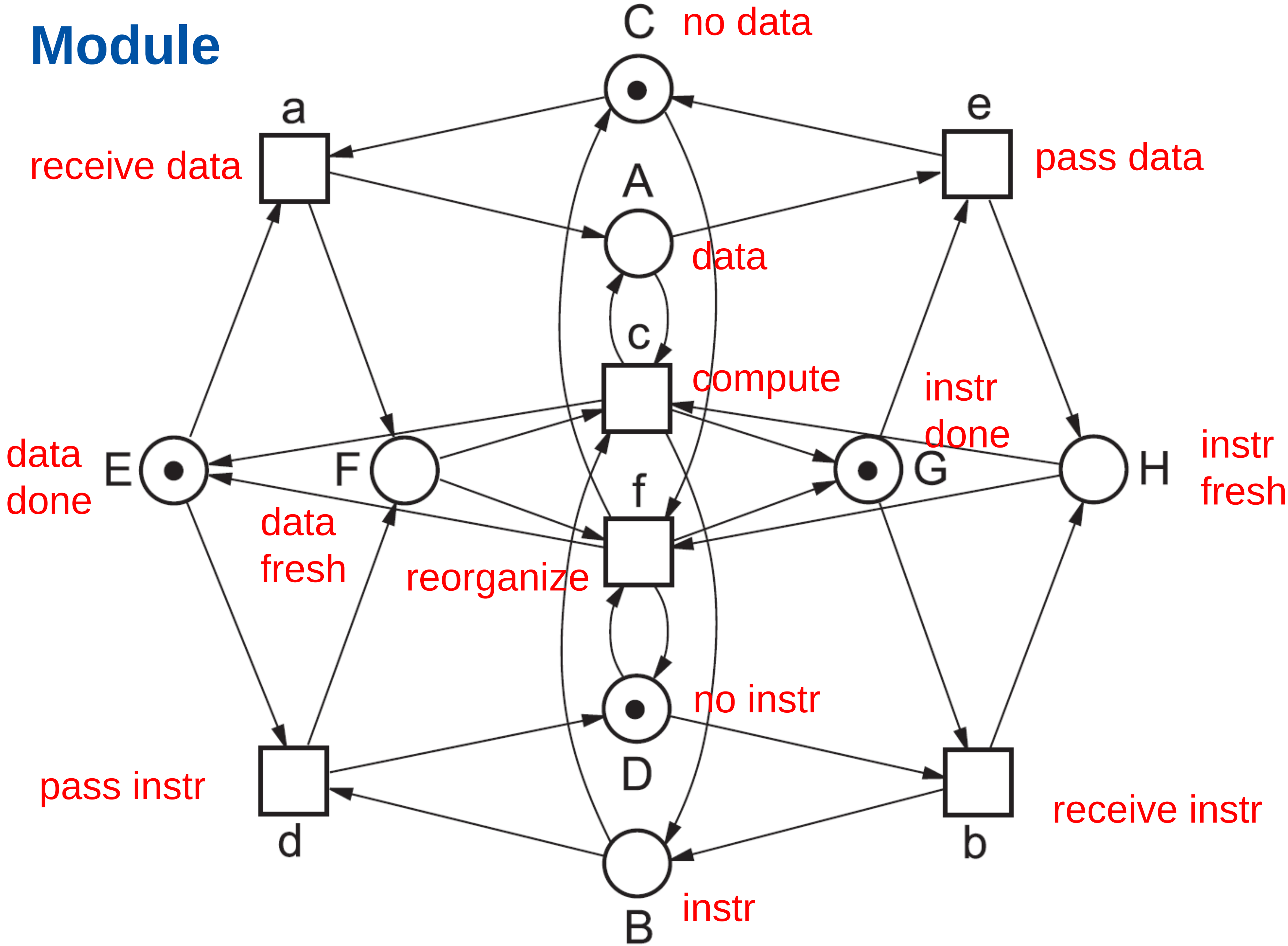


Sprout Counter Flow Pipeline-Processor

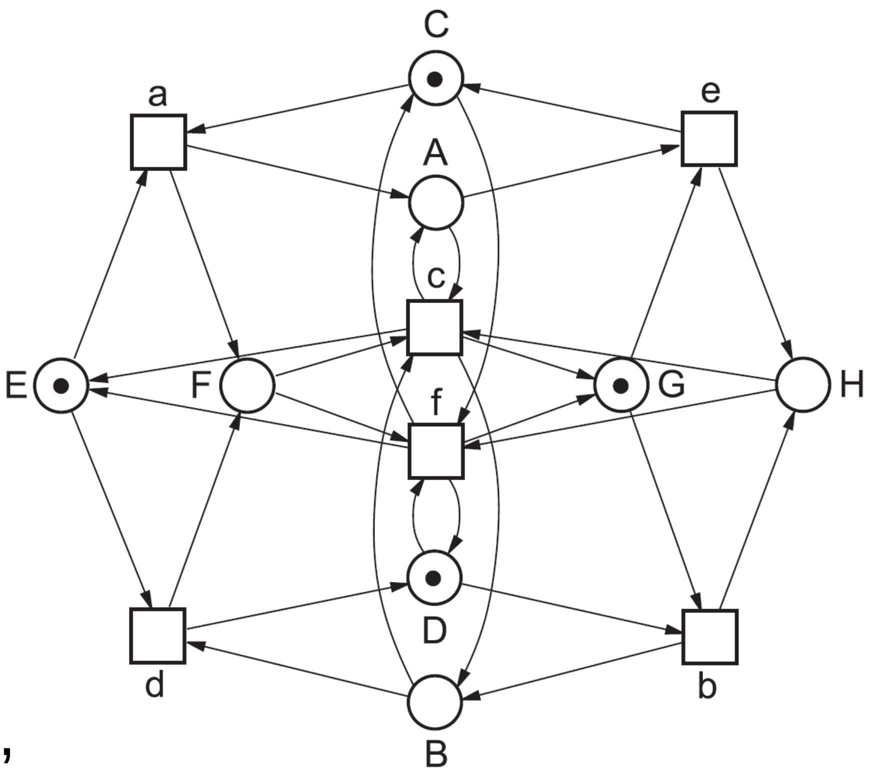


- Based on a stream of data packages $d_1 \ldots d_n$ and a stream of instructions $f_1 \ldots f_m$ compute $e_i =_{\text{def}} f_m(\ldots f_2(f_1(d_i)) \ldots)$
- Data and instructions arrive asynchronously
- Execution times of instructions vary
- Data flows from left to right
- Instructions flow from right to left

Module


C
no data
a
e
receive data
pass data
A
data
c
compute
instr
done
data
done
E
F
G
H
instr
fresh
f
data
fresh
reorganize
no instr
pass instr
D
receive instr
d
b
B
instr

Analysis



Place invariants:

A + H + E + D = 2

B + D = 1

Hence, if A and H are marked,
B must also be marked.

The edges between B and c can be removed.
(Analogously for C and f.)

Invariants & boundedness

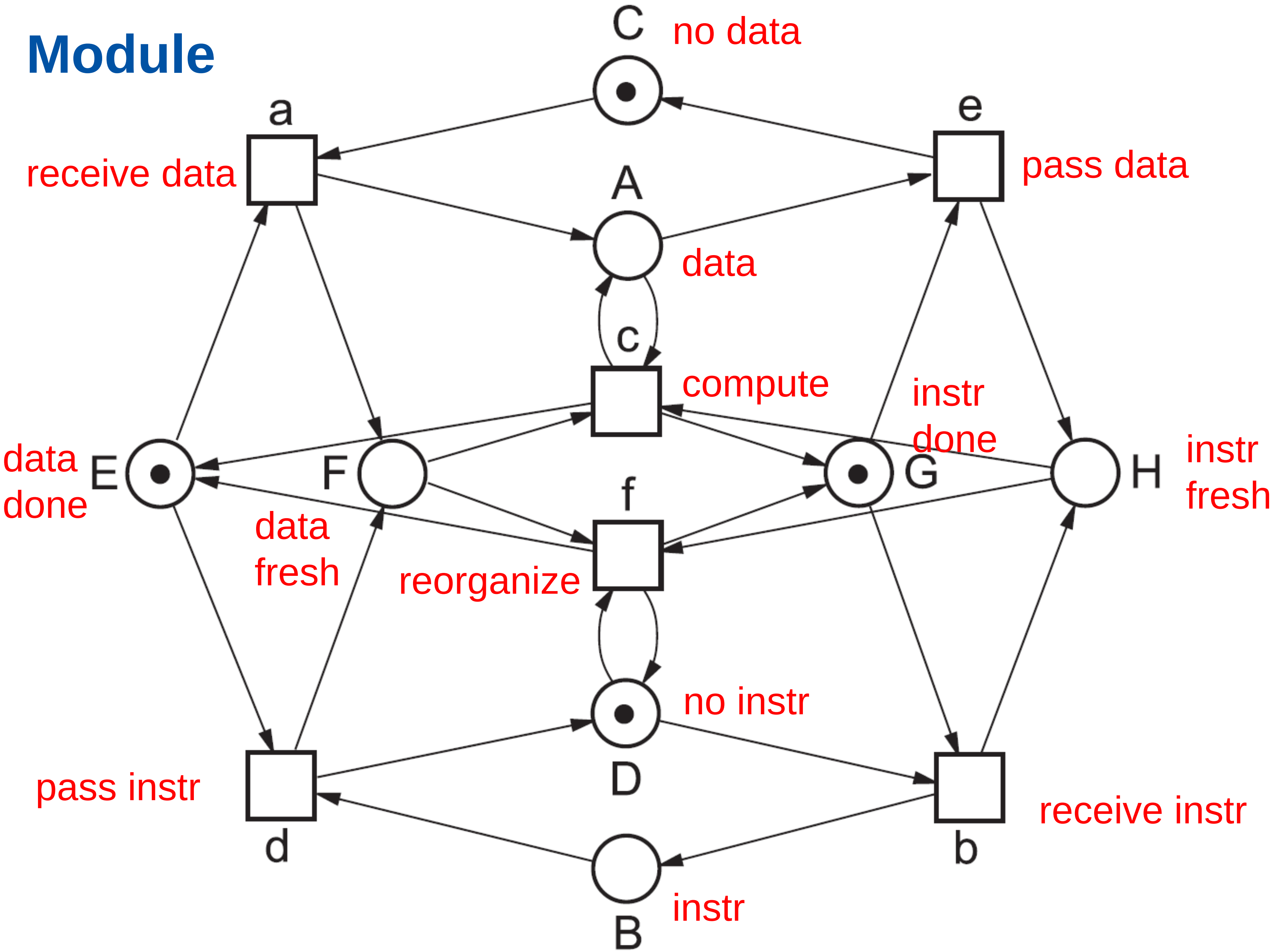
- A net is **covered** by place invariants
  iff every place is contained in some invariant.

Theorem 1:

a) If R is a place invariant and $p \in R$, then p is bounded.

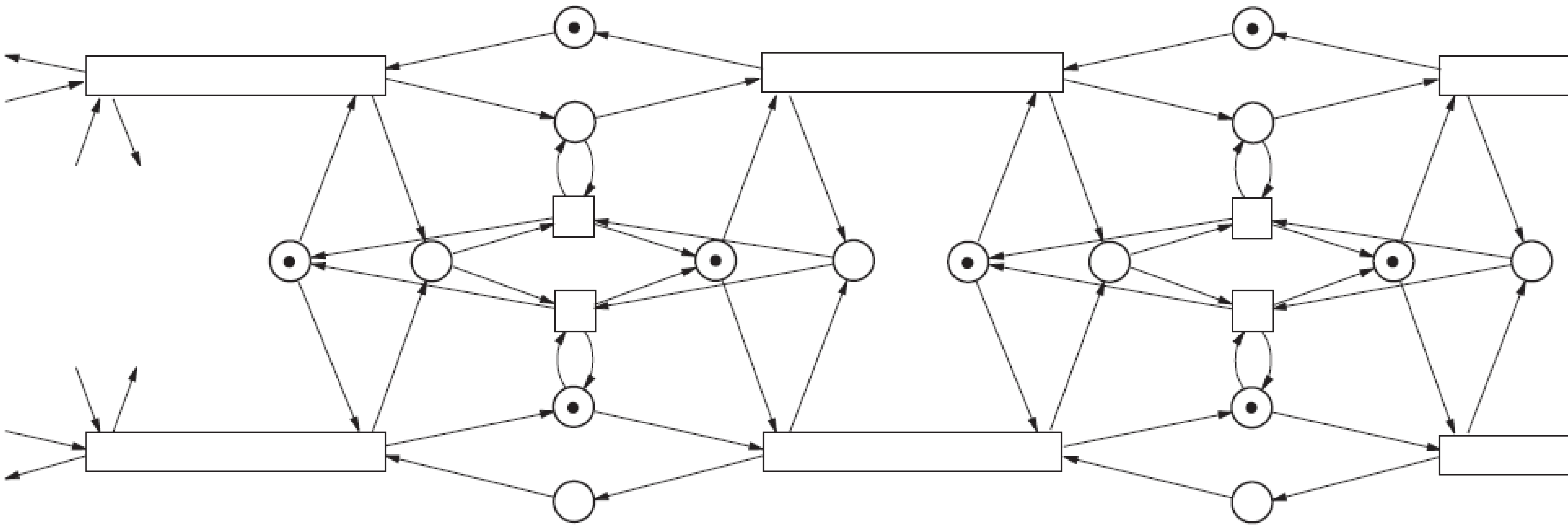
b) If a net is covered by place invariants then it is bounded.

Module



C
no data
a
e
receive data
A
pass data
data
c
compute
instr
done
data
done
E
F
G
H
instr
fresh
f
data
fresh
reorganize
no instr
pass instr
D
receive instr
d
b
instr
B

Composition of modules

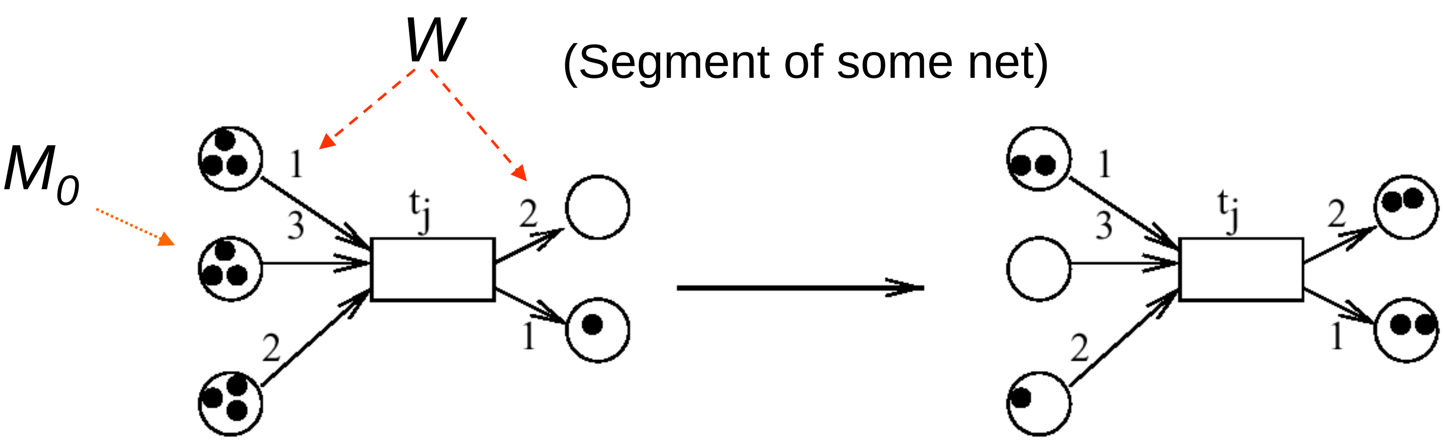


REVIEW: Place/transition nets

multiple tokens per place

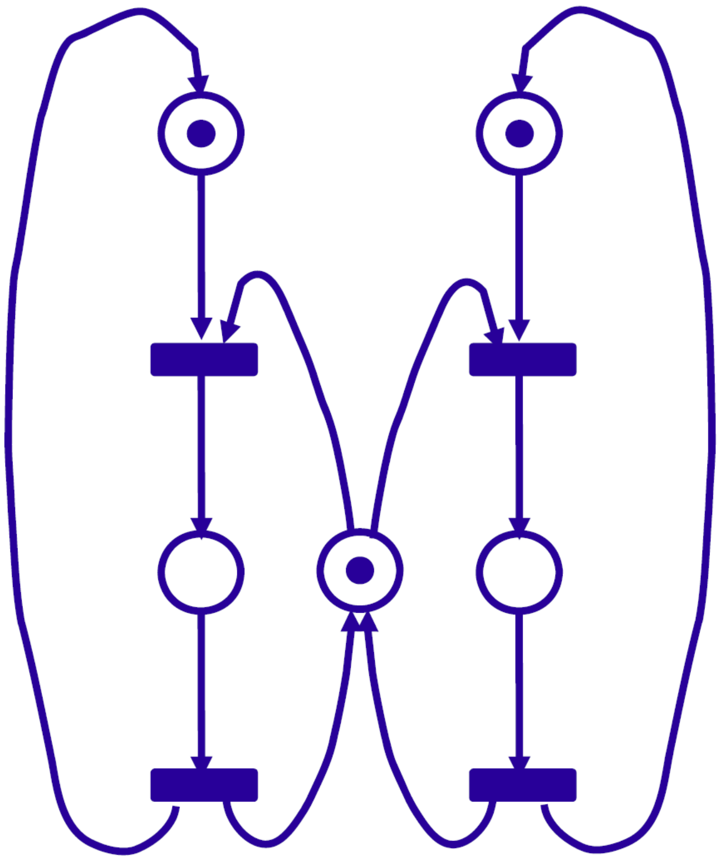
Def.: (P, T, F, K, W, M_0) is called a **place/transition net (P/T net)** iff

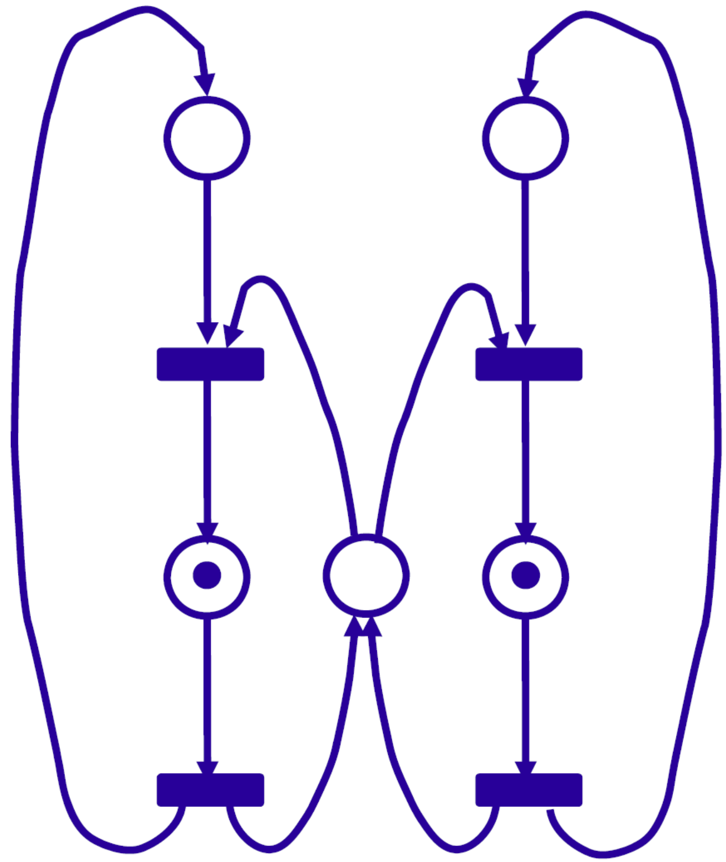
1. $N=(P,T,F)$ is a **net** with places P and transitions T
2. $K: P \rightarrow (\mathbb{N}_0 \cup \{\omega\}) \setminus \{0\}$ denotes the **capacity** of places
   (ω symbolizes infinite capacity)
3. $W: F \rightarrow (\mathbb{N}_0 \setminus \{0\})$ denotes the **weight of graph edges**
4. $M_0: P \rightarrow \mathbb{N}_0 \cup \{\omega\}$ represents the **initial marking** of places



default:
$K = \omega$
$W = 1$

REVIEW: Reachability





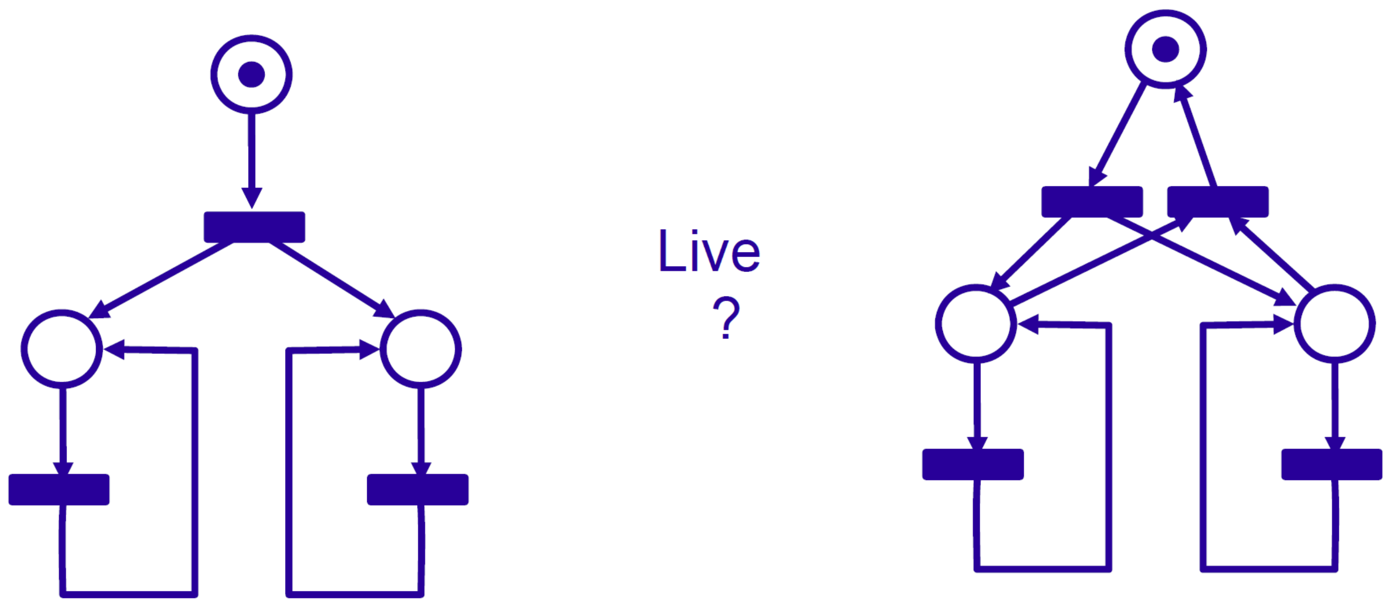
Marking
M

Is there a sequence of transition firings such that M ⟶ M'?

Marking
M'

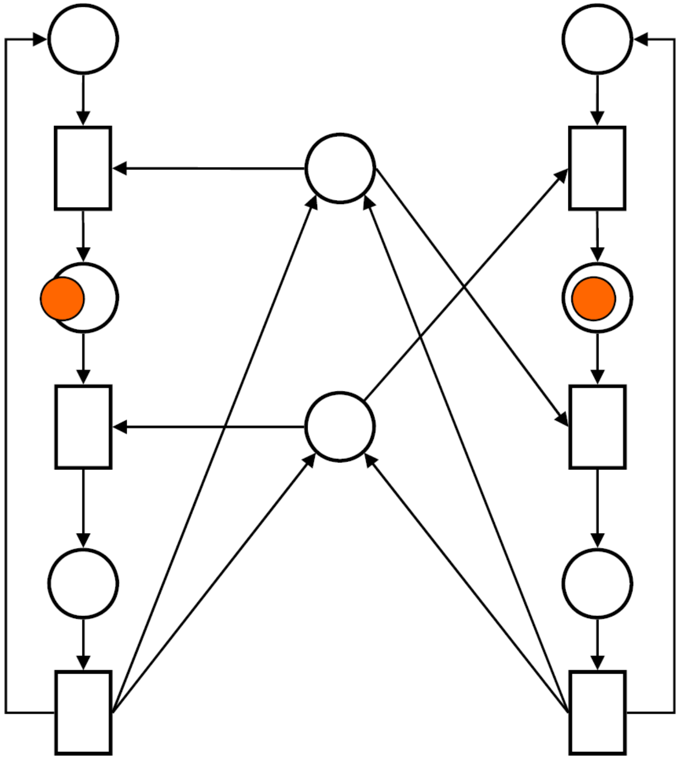
REVIEW: Liveness

- A transition is **live** if in every reachable marking there exists a firing sequence such that the transition becomes enabled
- A net is **live** if all its transitions are live

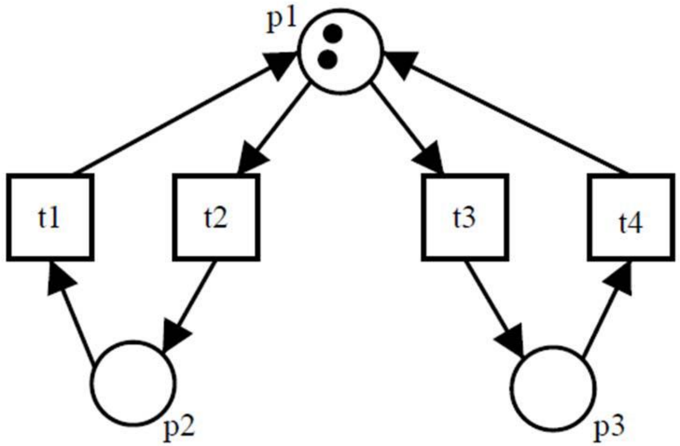


REVIEW: Deadlock

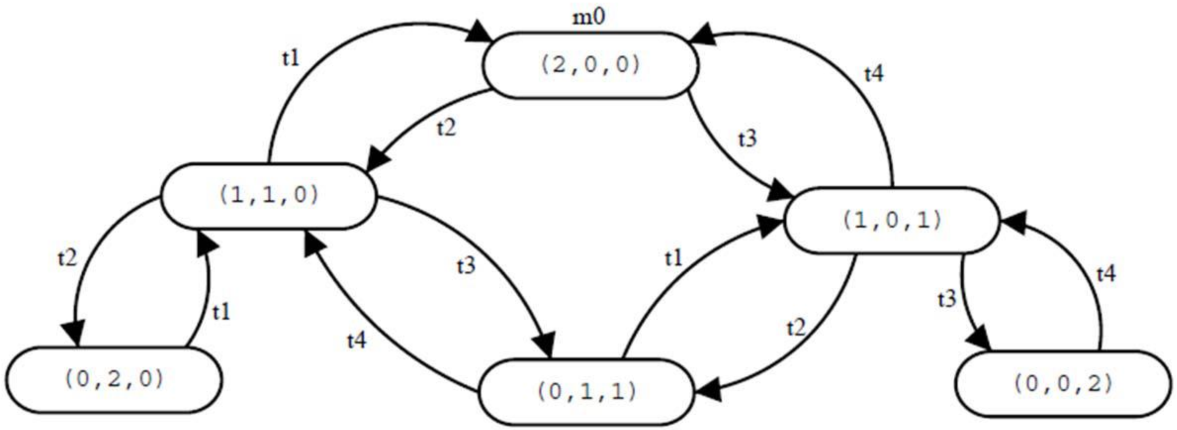
- A **dead marking (deadlock)** is a marking where no transition can fire
- A net is **deadlock-free** if no dead marking is reachable



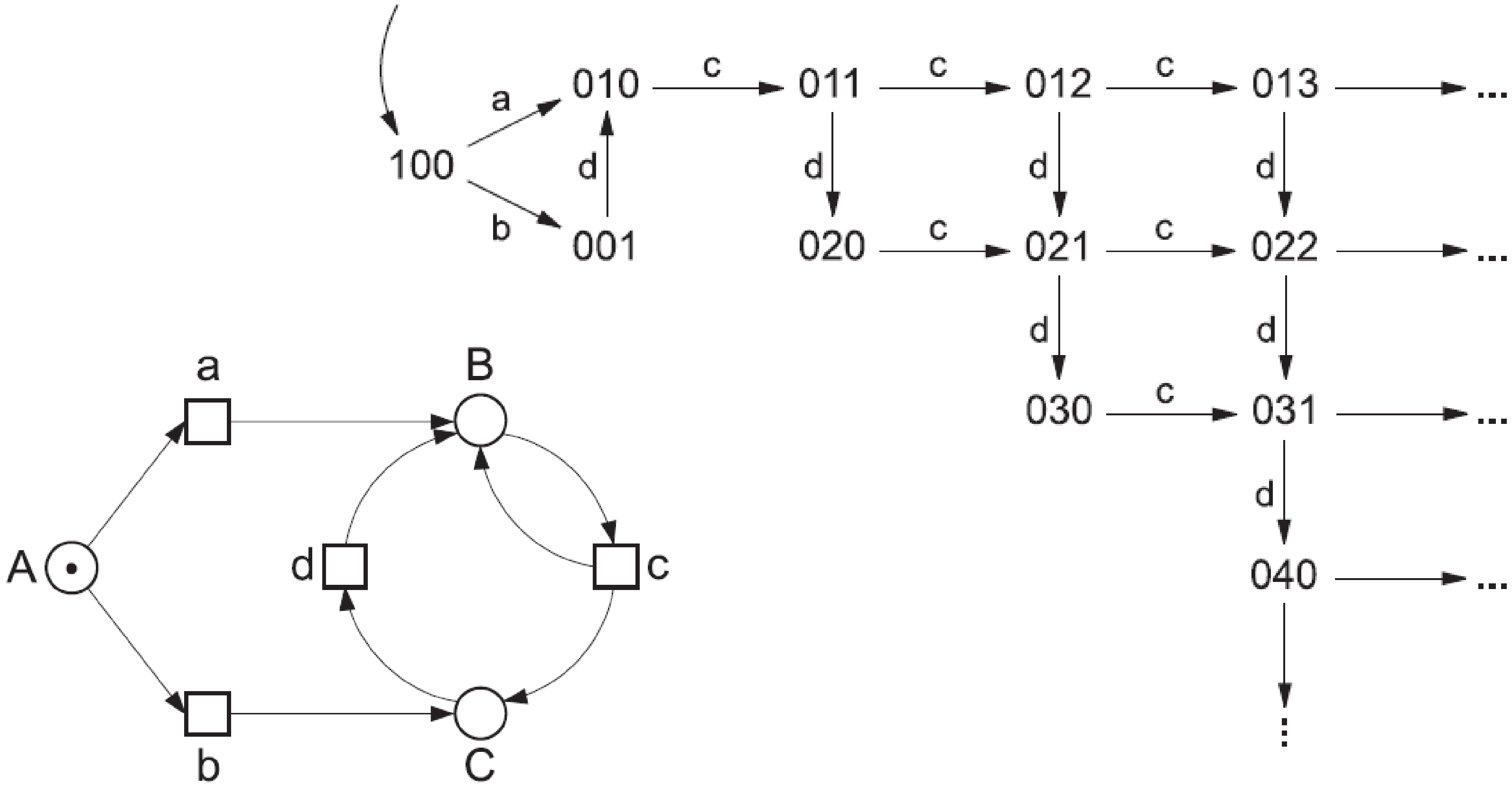
Reachability, Liveness, Deadlock are graph problems on reachability graph



Reachability graph:

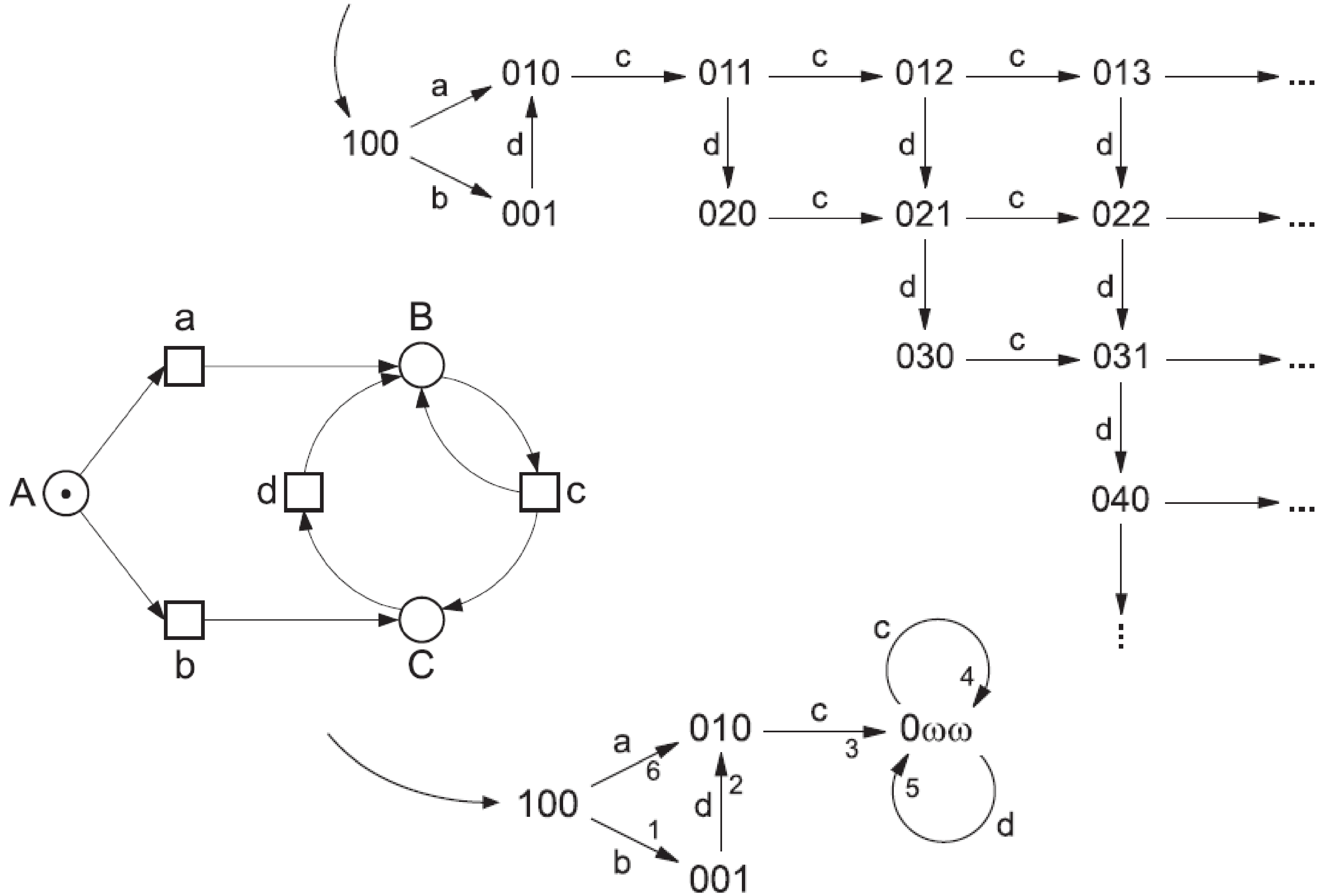


Reachability graph is in general infinite



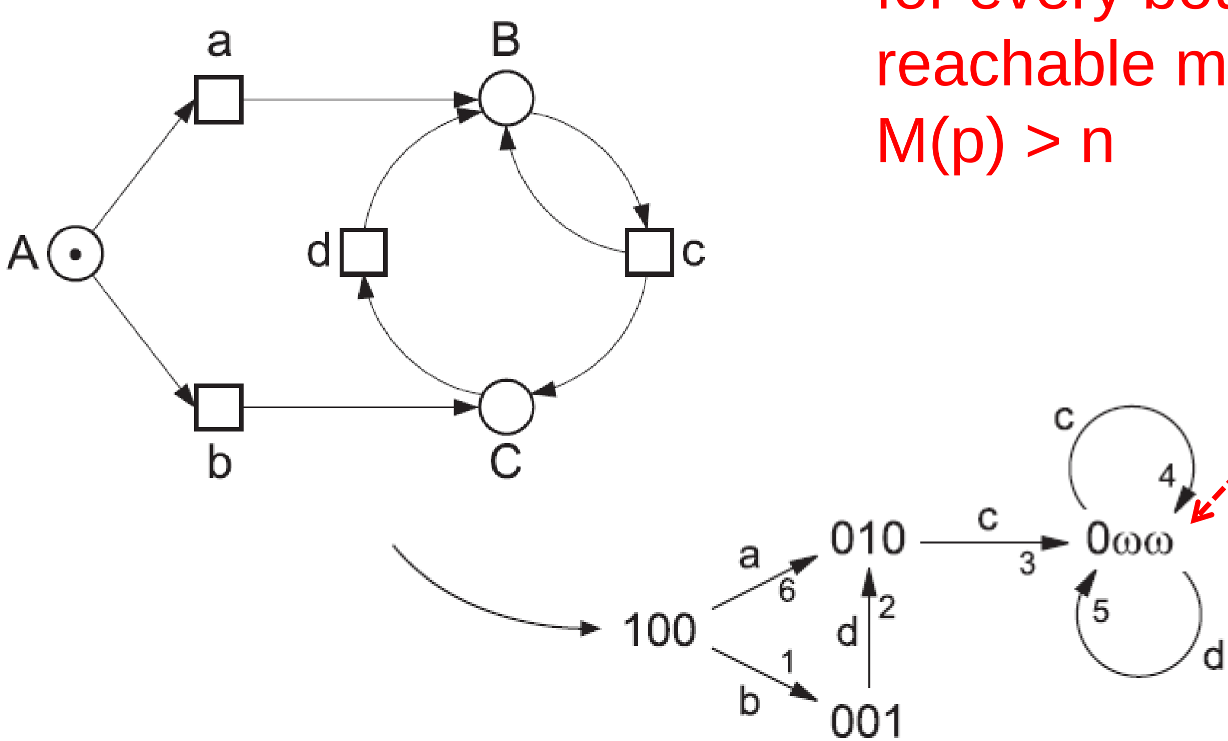
Example from Wolfgang Reisig: Petrinetze, Springer 2010

Coverability graph



Example from Wolfgang Reisig: Petrinetze, Springer 2010

Coverability graph



ω indicates that arbitrarily high values can be reached: for every bound n there is a reachable marking M with M(p) > n

Example from Wolfgang Reisig: Petrinetze, Springer 2010

Constructing the coverability graph

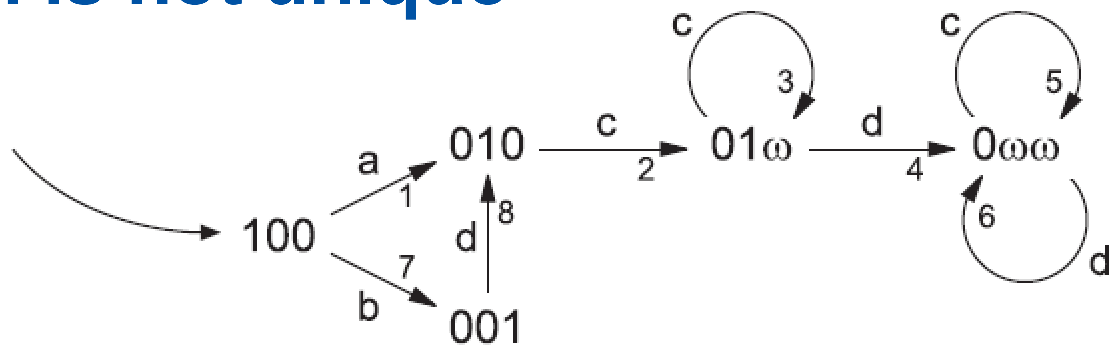
- The initial graph consists of the initial marking M_0
- Extend the graph as long as there exists a node *M* such that
  - a transition *t* can fire from *M* leading to some marking *M*'
  - but there is no outgoing edge from *M* labeled with *t*

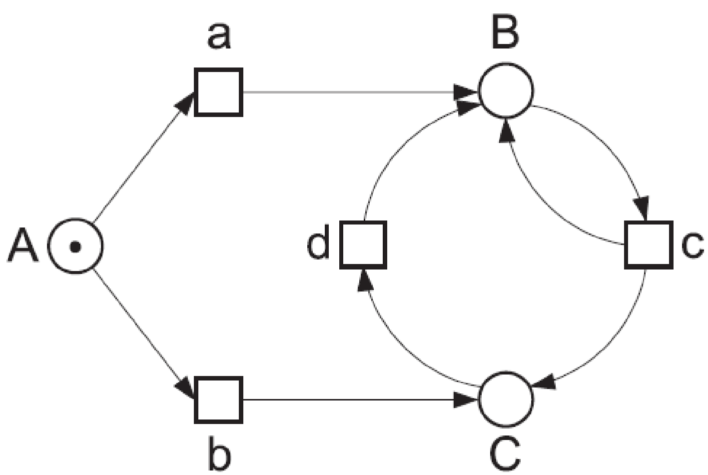
Create a *t*-labeled edge from *M* to *M*'', where *M*" is defined as follows:

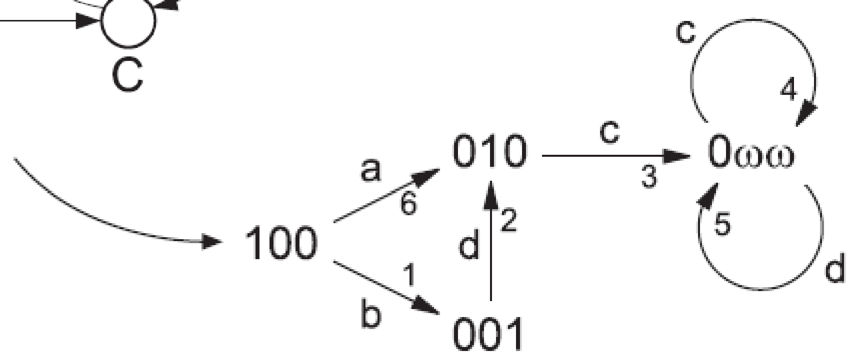
$M''(p) = \omega$ if there exists a path from M_0 to M through some node L with $L \leq M'$ and $L(p) < M'(p)$

$M''(p) = M'(p)$ otherwise

Coverability graph is not unique







Example from Wolfgang Reisig: Petrinetze, Springer 2010

Finiteness of the coverability graph

Theorem 2: Every P/T net has a finite coverability graph.

Lemma 1: Every infinite sequence of markings (M_i) contains a weakly monotonically growing infinite subsequence $(M`_i)$, i.e., for j<k, $M`_j \leq M`_k$.

Coverability theorem

A marking M **covers** a marking M' iff, for all places p,

$M(p) = M'(p)$ or $M(p) = \omega$.

A computation of a P/T net is a sequence

$$M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots$$

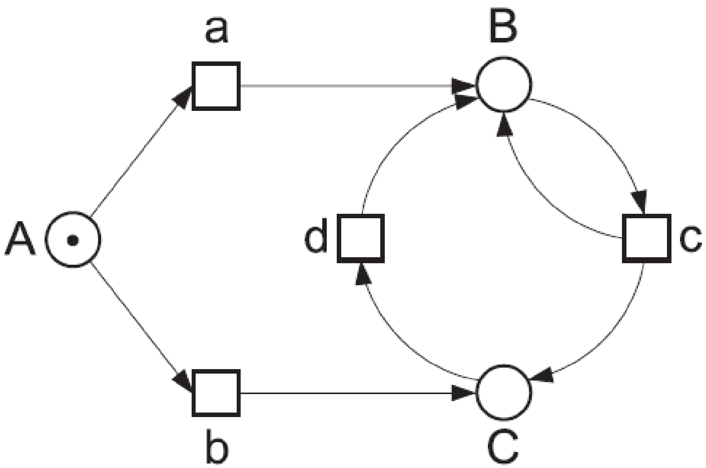
where M_0 is the initial marking and M_{i+1} is the result of firing transition t_i in marking M_i

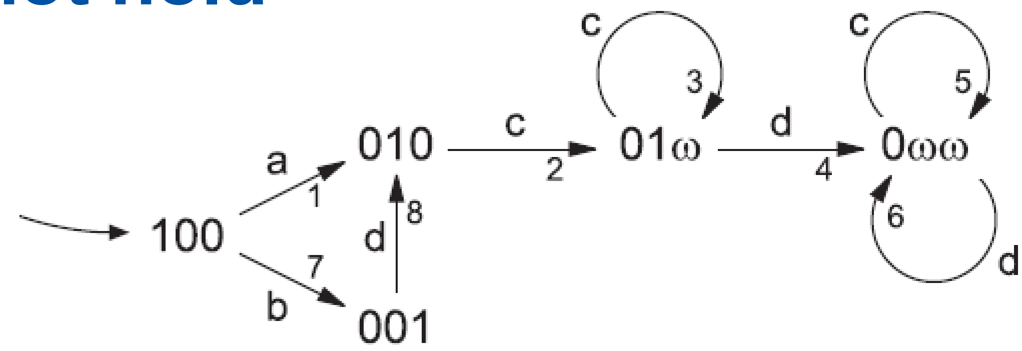
Theorem 3: For every computation

$M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots$ of a P/T net there exists, in every coverability graph, a path

$M'_0 \xrightarrow{t_0} M'_1 \xrightarrow{t_1} M'_2 \xrightarrow{t_2} \ldots$ such that M'_i covers M_i for all i.

The converse does not hold



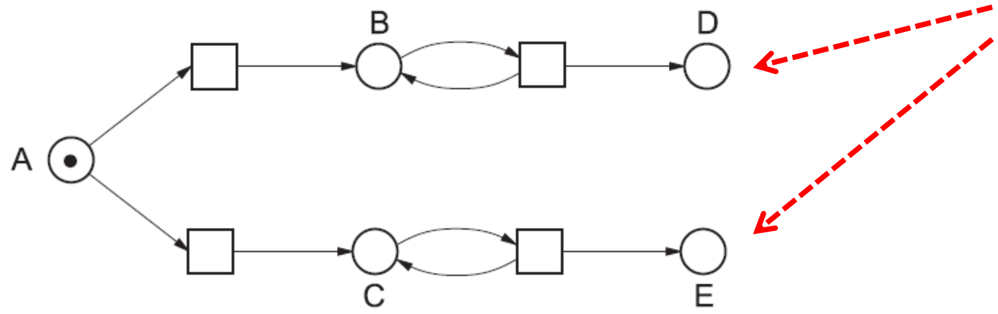


$$100 \xrightarrow{a} 010 \xrightarrow{c} 01\omega \xrightarrow{d} 0\omega\omega \xrightarrow{d} 0\omega\omega \cdots$$

Example from Wolfgang Reisig: Petrinetze, Springer 2010

Simultaneous unboundedness

A set Q of places is **simultaneously unbounded** iff, for every natural number i, there exists a reachable marking M^i where, for all $q \in Q$, $M^i(q) \geq i$.

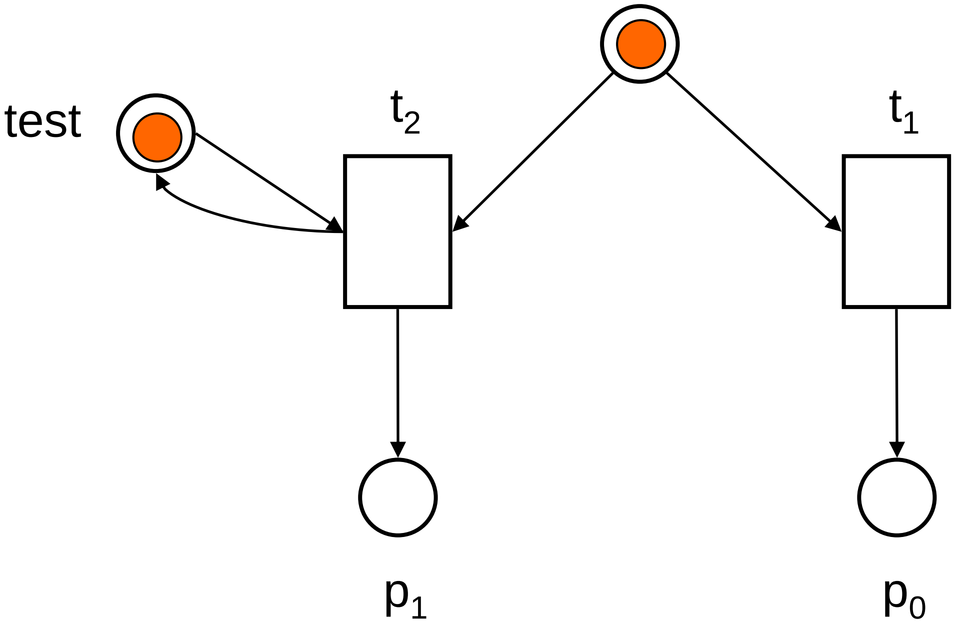


D and E are unbounded but not simultaneously unbounded

Theorem 4: For every node M in a coverability graph of some P/T net, it holds that the places in ω_M, where $p \in \omega_M$ iff $M(p) = \omega$, are **simultaneously unbounded**.

Extensions: Petri nets with priorities

- $t_1 \langle t_2$: t_2 has higher priority than t_1.



- Petri nets with priorities are Turing-complete.

Extensions: Predicate/transition nets

- 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,

  ☞ semantics can be defined by C/E nets.

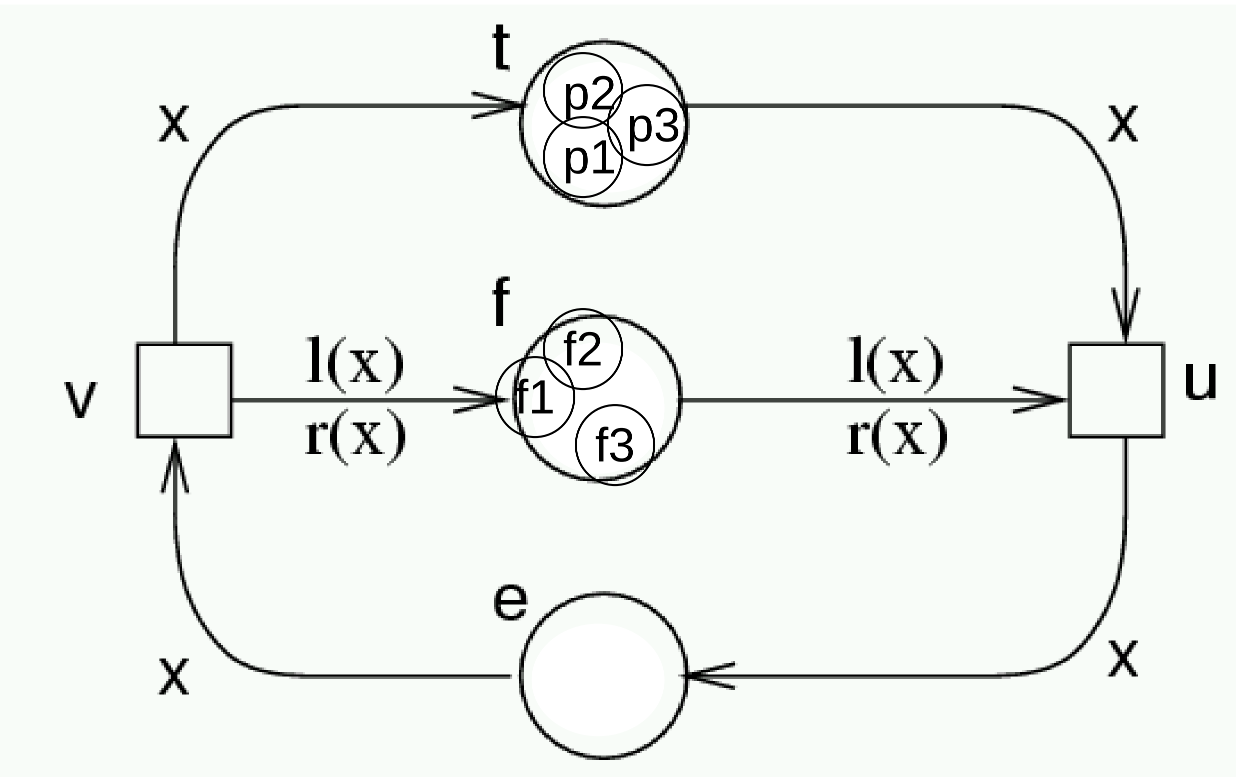
Predicate/transition model of the dining philosophers problem

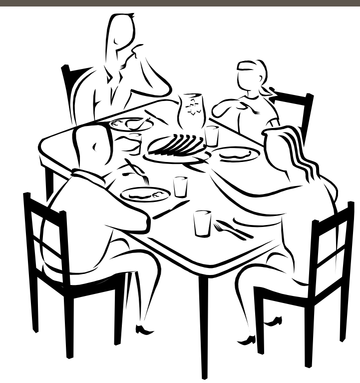
- Let x be one of the philosophers,
- let l(x) be the left fork of x,
- let r(x) be the right fork of x.

Token: individuals.

Semantics can be defined by replacing net by equivalent condition/event net.

Model can be extended to arbitrary numbers.





Petri nets - summary

- 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
- Advanced theory for analyzing properties
  (In general expensive. Reachability is EXPSPACE-hard.)

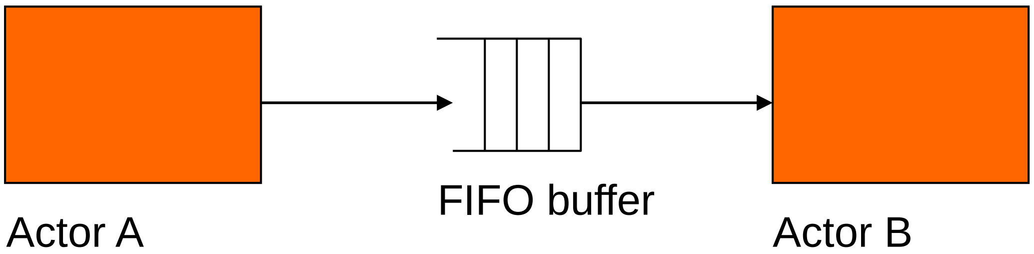
Data Flow Models

Lee/Seshia
Section 6.3

Marwedel

Section 2.5

Dataflow Models



- Buffered communication between concurrent components (*actors*).

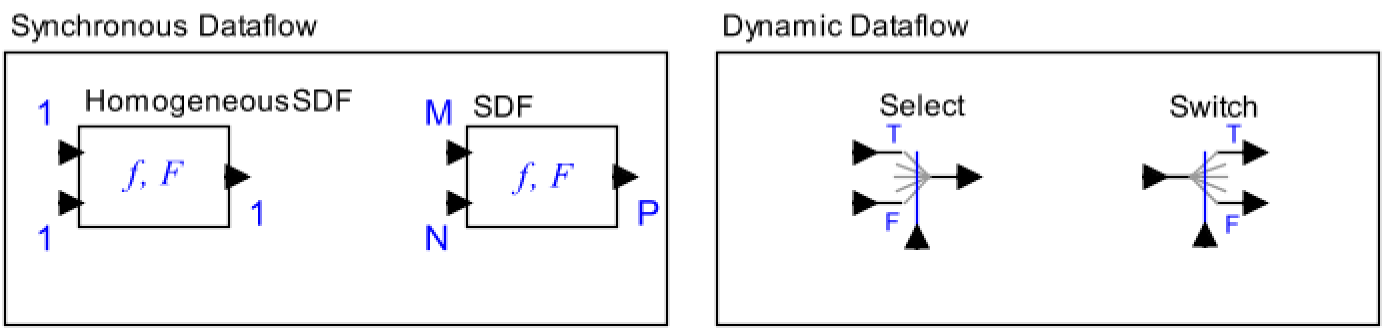
- An actor can fire whenever it has enough data (*tokens*) in its input buffers. It then produces some data on its output buffers.

- In principle, buffers are unbounded. But for implementation on a computer, we want them bounded (and as small as possible).

Streams: The basis for Dataflow models

A stream is a signal $x\colon \mathbb{N} \to R$, for some set R. There is not necessarily any relationship between $x(n)$, an element in a stream, and $y(n)$, an element in another stream. Unlike discrete-time models or SR models, they are not "simultaneous."

Dataflow



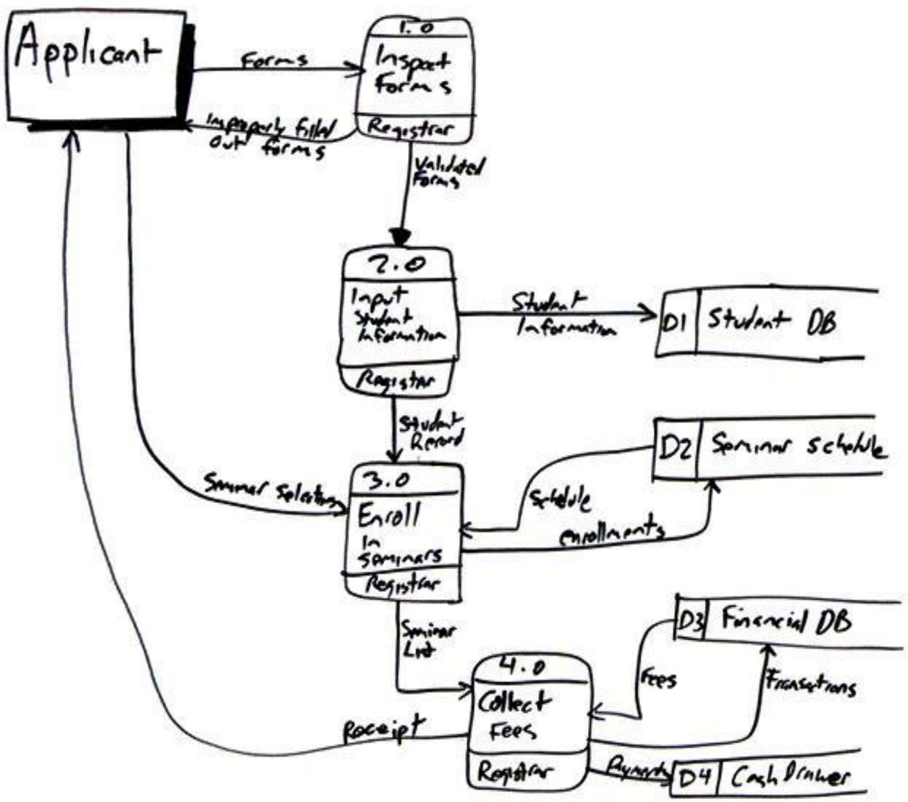
Each signal has form $x : \mathbb{N} \to R$. The function F maps such signals into such signals. The function f (the "firing function") maps prefixes of these signals into prefixes of the output. Operationally, the actor *consumes* some number of tokens and *produces* some number of tokens to construct 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.

Misleading terminology!

"synchronous dataflow" does not mean "synchronous composition"

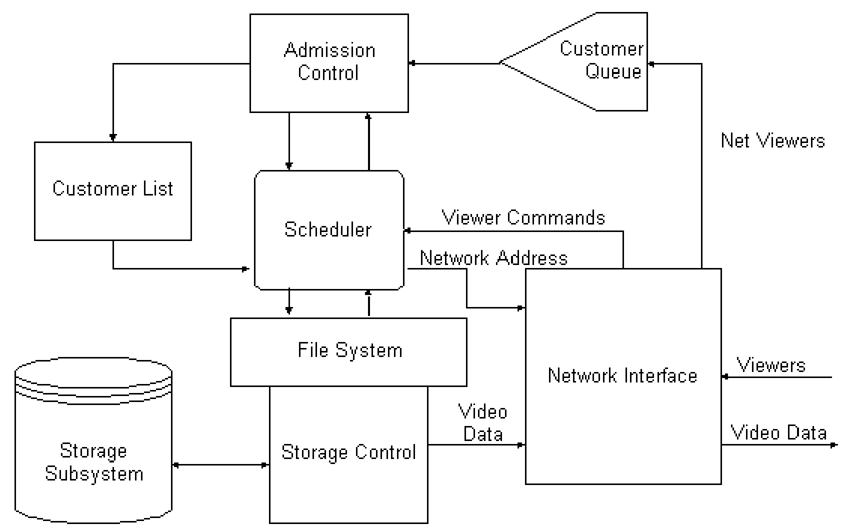
Data flow as a "natural" model of applications

Registering for courses



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

Video on demand system

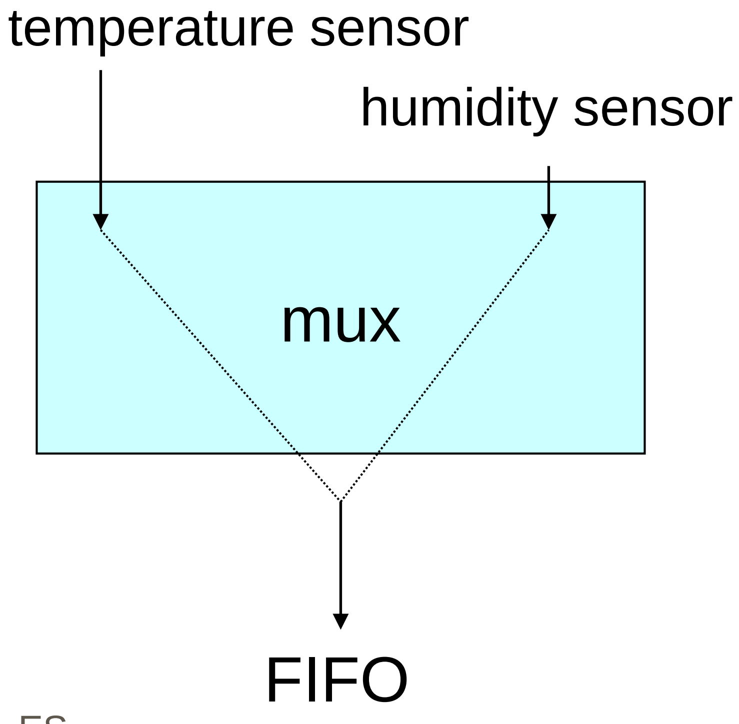


www.ece.ubc.ca/~irenek/techpaps/vod/vod.html

Process networks

Many applications can be specified in the form of a set of communicating processes.

Example: system with two sensors:



Alternating read

```
loop
 read_temp; read_humidity
until false;
```

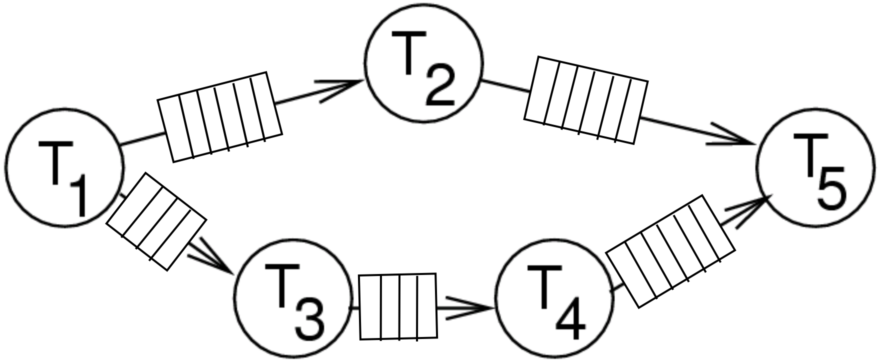
of the two sensors

not the right approach.

Reference model for dynamic data flow:
Kahn process networks (1974)

Describe computations to be performed and their dependence
but not the order in which they must be performed

communication via infinitely large FIFOs



Properties of Kahn process networks (1)

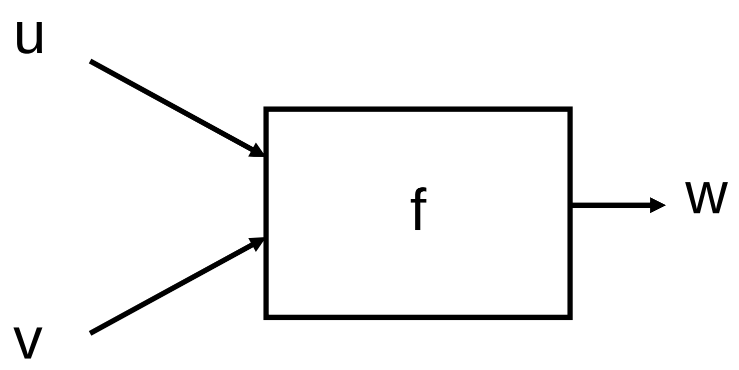
- Each node corresponds to one program/task;
- Communication is only via channels;
- Channels include FIFOs as large as needed;
- Channels transmit information within an unpredictable but finite amount of time;
- Mapping from $\geq$1 input seq. to $\geq$1 output sequence;
- In general, execution times are unknown;
- Send operations are non-blocking, reads are blocking.
- One producer and one consumer;
  i.e. there is only one sender per channel;

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 the same, regardless of the speed of the nodes.

A Kahn Process

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

Source: Gilles Kahn, The Semantics of a Simple Language for Parallel Programming (1974)

A Kahn Process

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

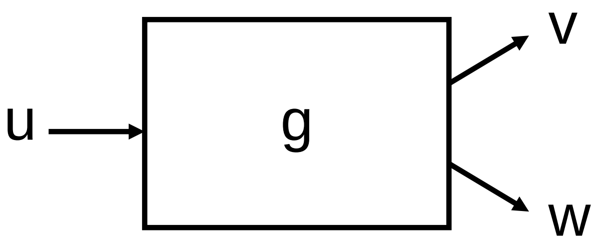
wait() returns the next token in an input FIFO, blocking if it's empty

send() writes a data value on an output FIFO

Source: Gilles Kahn, The Semantics of a Simple Language for Parallel Programming (1974)

A Kahn Process

```
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;
  }
}
```

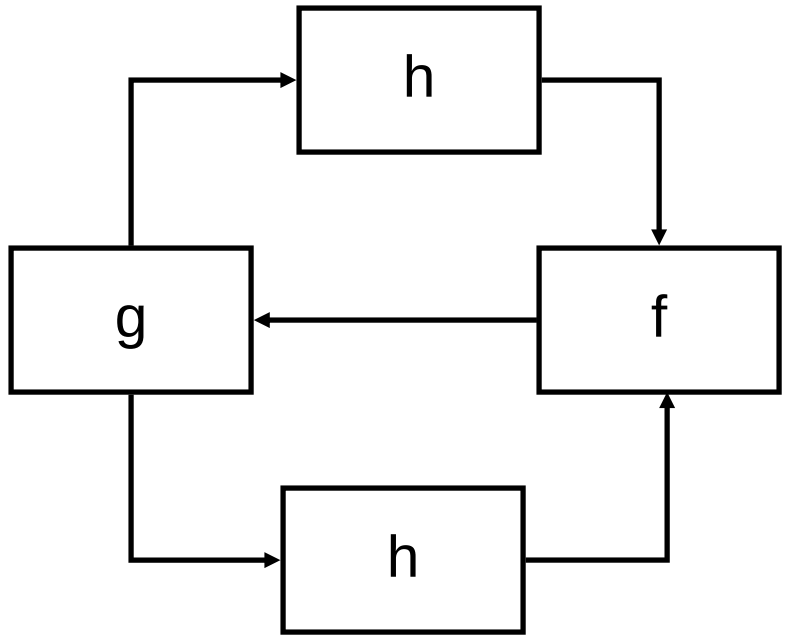


Process reads from u and alternately copies it to v and w

A Kahn System

- 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

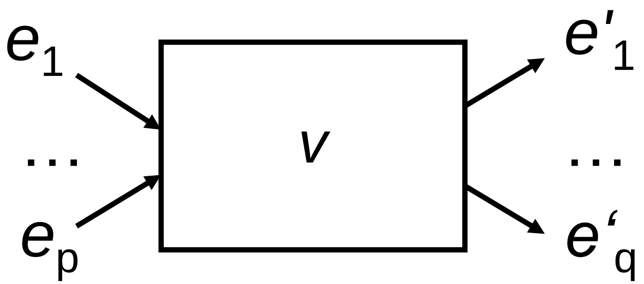
Definition: Kahn networks

A **Kahn process network** is a directed graph (V,E), where

- V is a set of **processes**,
- $E \subseteq V \times V$ is a set of **edges**,
- associated with each edge e is a **domain** D_e
- D^ω: finite or countably infinite sequences over D

D^ω is a complete partial order where
$X \leq Y$ iff X is an initial segment of Y

Definition: Kahn networks



- associated with each process $v \in V$ with incoming edges e_1, …, e_p and outgoing edges e_1', …,e_q' is a continuous **function** f_v: $D_{e_1}^{\omega} \times \ldots \times D_{e_p}^{\omega} \rightarrow D_{e'_1}^{\omega} \times \ldots \times D_{e'_q}^{\omega}$

(A function f: $A \rightarrow B$ is **continuous** if $f(\lim_A a) = \lim_B f(a)$)