

Petri Nets

Outline

These slides will be covered during TWO lectures.

Introduction

Basic Definition

Modeling Program Graphs

Modeling Control

Modeling Non-computer Systems

Formalism

Properties of Petri Nets

Modeling Databases & Operating Systems

Variations in Modeling

Goals for Studying Petri Nets

1. Familiarity with an important modeling tool
 - ◇ areas Petri nets can model well
 - ◇ limitations of Petri nets
 - ◇ variety of representations
 - ◇ common properties

2. Capabilities with the tool
 - ◇ understand complex models
 - ◇ develop simple models

3. Further information:

World of Petri Nets web page:

<http://www.informatik.uni-hamburg.de/TGI/PetriNets/>

look for simulators under “Tools and Software” link

Characteristics of the Model

- complex processes
 - concurrent activities
 - flow of data or other things
 may be transformed on the way
 - multiple coordinated flows
- ∴ abstraction of Data Flow Diagrams
 focusing on synchronization

Petri Nets & Finite State Automata

Some visual similarity between FSA and Petri Nets diagrams

both have circle and arrows

FSA are special cases of Petri nets

In general, Petri nets

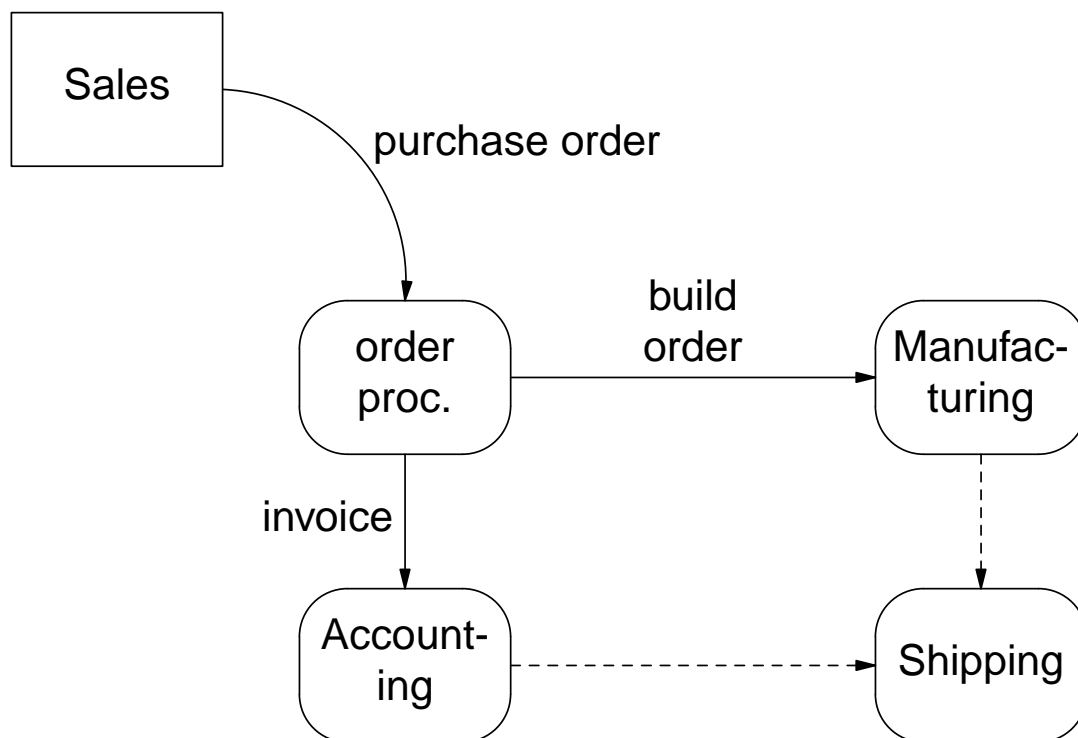
- emphasize locality of condition verses global state
- have multiple conditions active at once hence non-determinacy more global
- express complex transitions succinctly
- allow multiple instances in one condition without bound

Beyond DFDs

+ explicit coordination / synchronization

- no identity of “packets”

How do *invoice* and *build order* from the same *purchase order* get matched up?



Game Metaphor for Petri Nets

Can imagine each Petri net as

- a unique gameboard

but

- using the same kind of pieces and
- following the same general rules

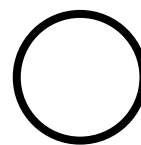
The Gameboard

Each Petri net looks like a gameboard with

- circles called conditions or places
- bars called events or transitions
- arrows, called input arcs, from conditions to events
 - ◇ condition is input to the event
- arrows, called output arcs, from events to conditions
 - ◇ condition is output from the event
- multiple parallel edges possible

Notation:

condition



event



- formally, a “bipartite directed multigraph”

The Pieces

A token on a condition

- means that condition currently is true
- is drawn by a black dot on that condition
- may share that condition with other tokens, indicating multiple objects sharing that condition
- during a given turn, may stay on a condition or move to an event

A marking of a Petri net

- is a function from the conditions to the non-negative integers
- indicates the number of tokens currently at that condition

The Rules of Play

An event is enabled when it is possible to place
on each input arc
a token from the corresponding input condition

An event fires

- only if it is enabled
- by consuming a token along each input arc
- and producing a token along each output arc

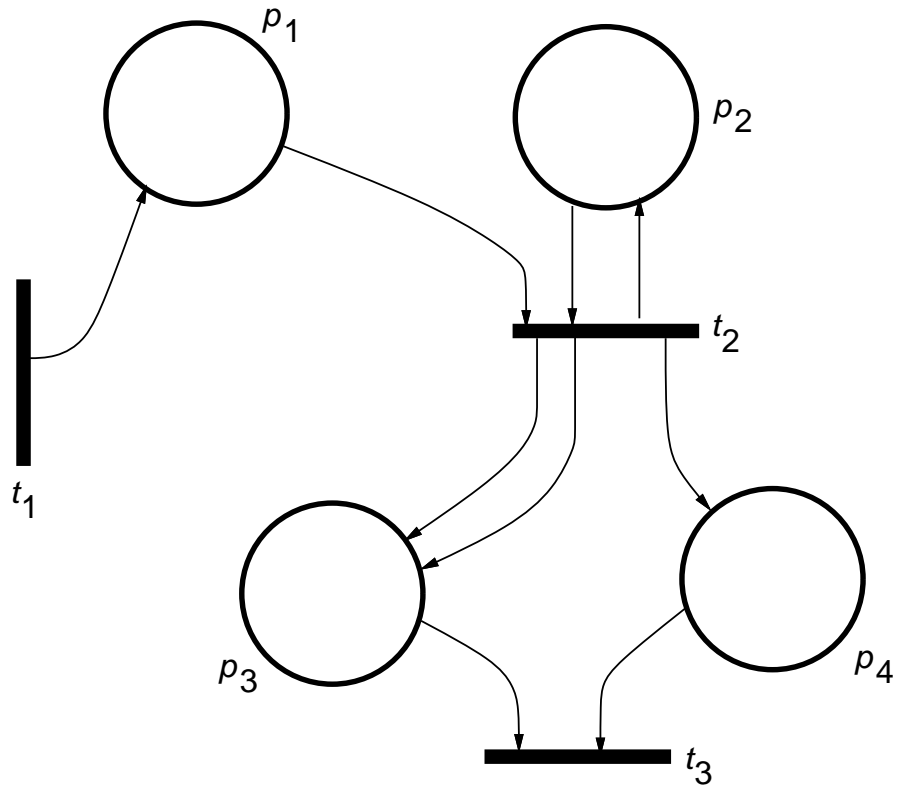
Indeterminate firing:

- If more than one event is enabled at a given time, *any* of the enabled events may fire

Concurrency:

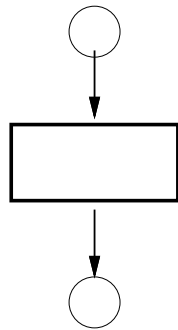
- Formally, two events may not fire at the same time
- However, this is often not relevant, so it is common to think of event firings as being concurrent

Example

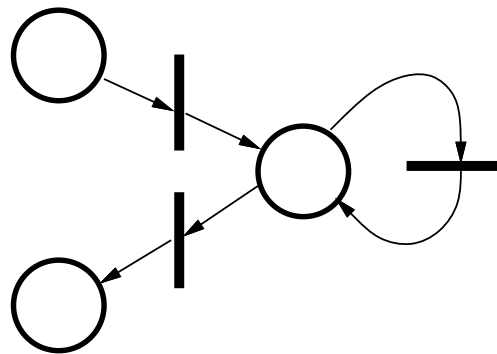
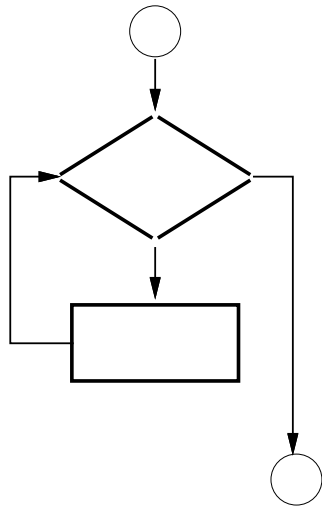
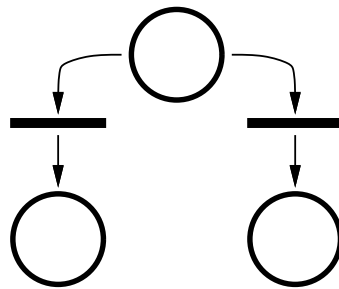
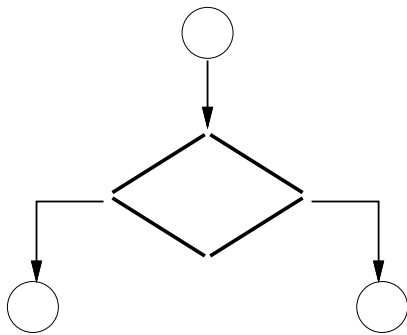
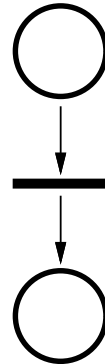


Basic Control Constructs

flowchart



Petri net

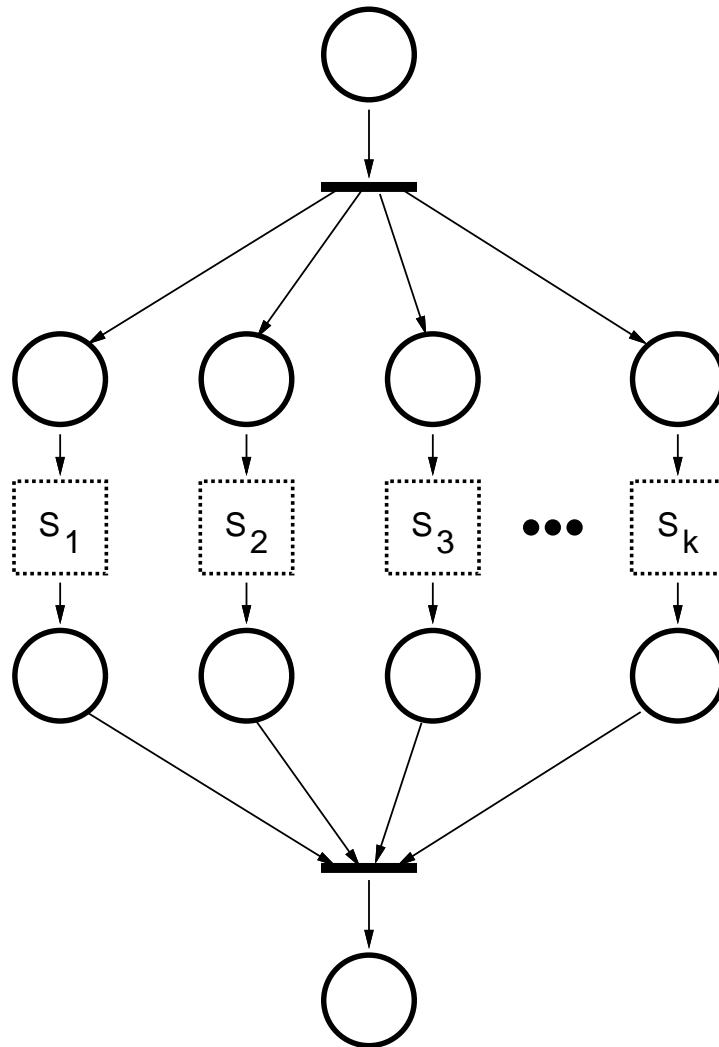


Parallel Programming

PARBEGIN

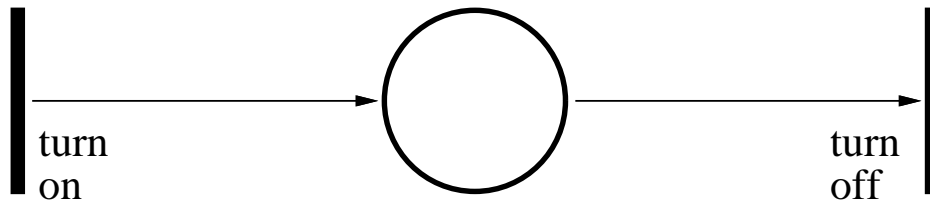
S_1, S_2, \dots, S_k

PAREND

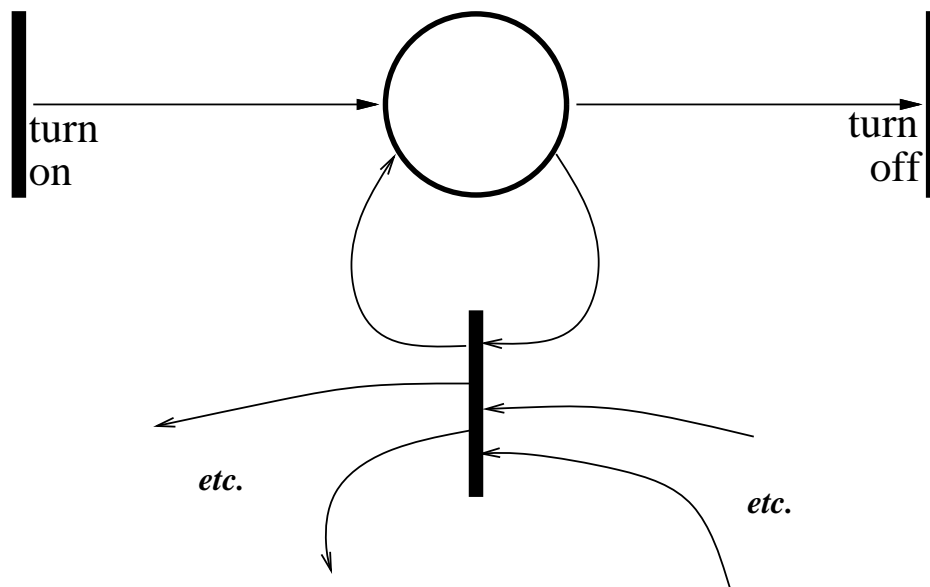


Switches

The simplest switch:



Use this to en/dis-able other transition(s)

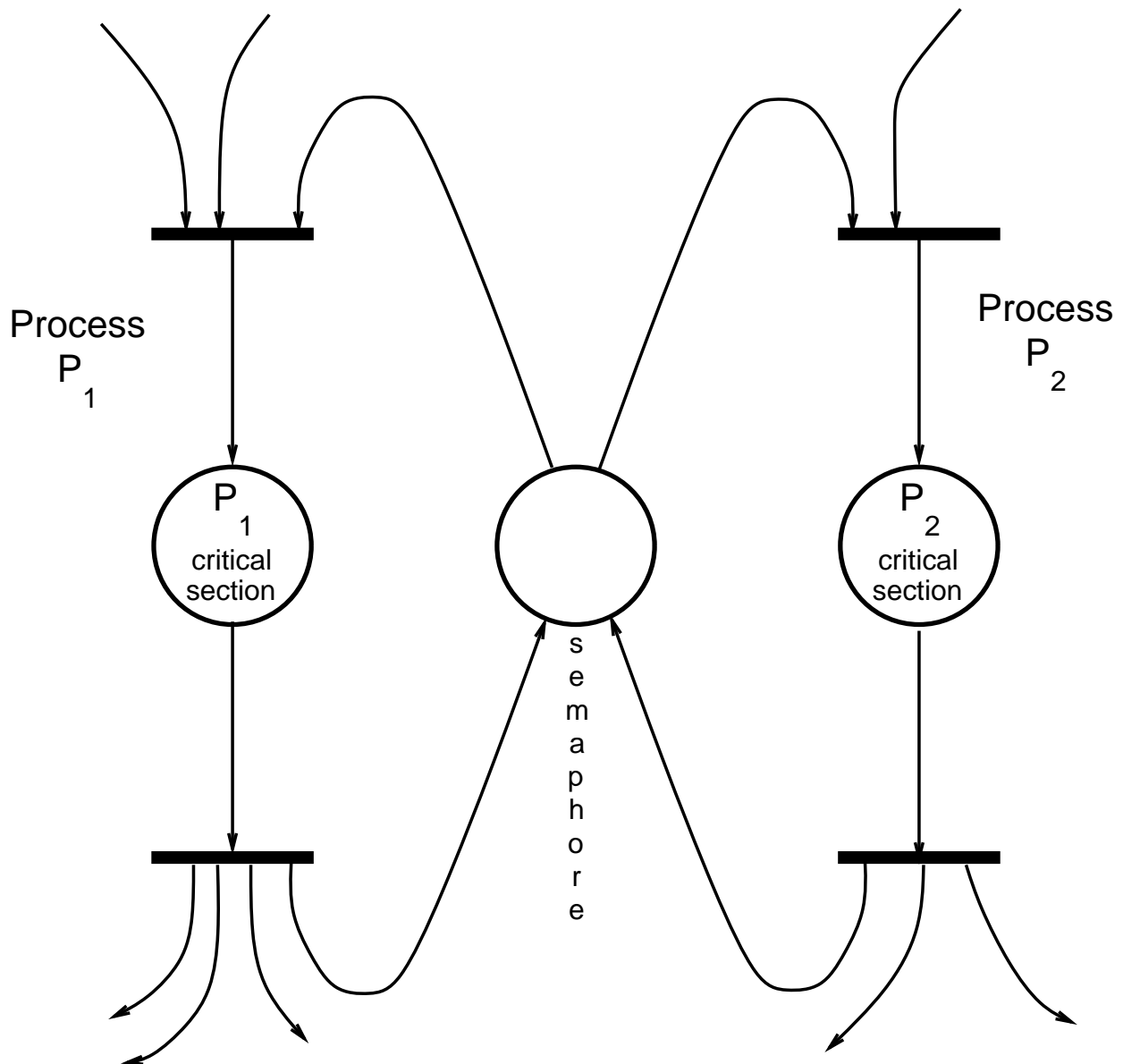


What is wrong with the above?

Mutual Exclusion

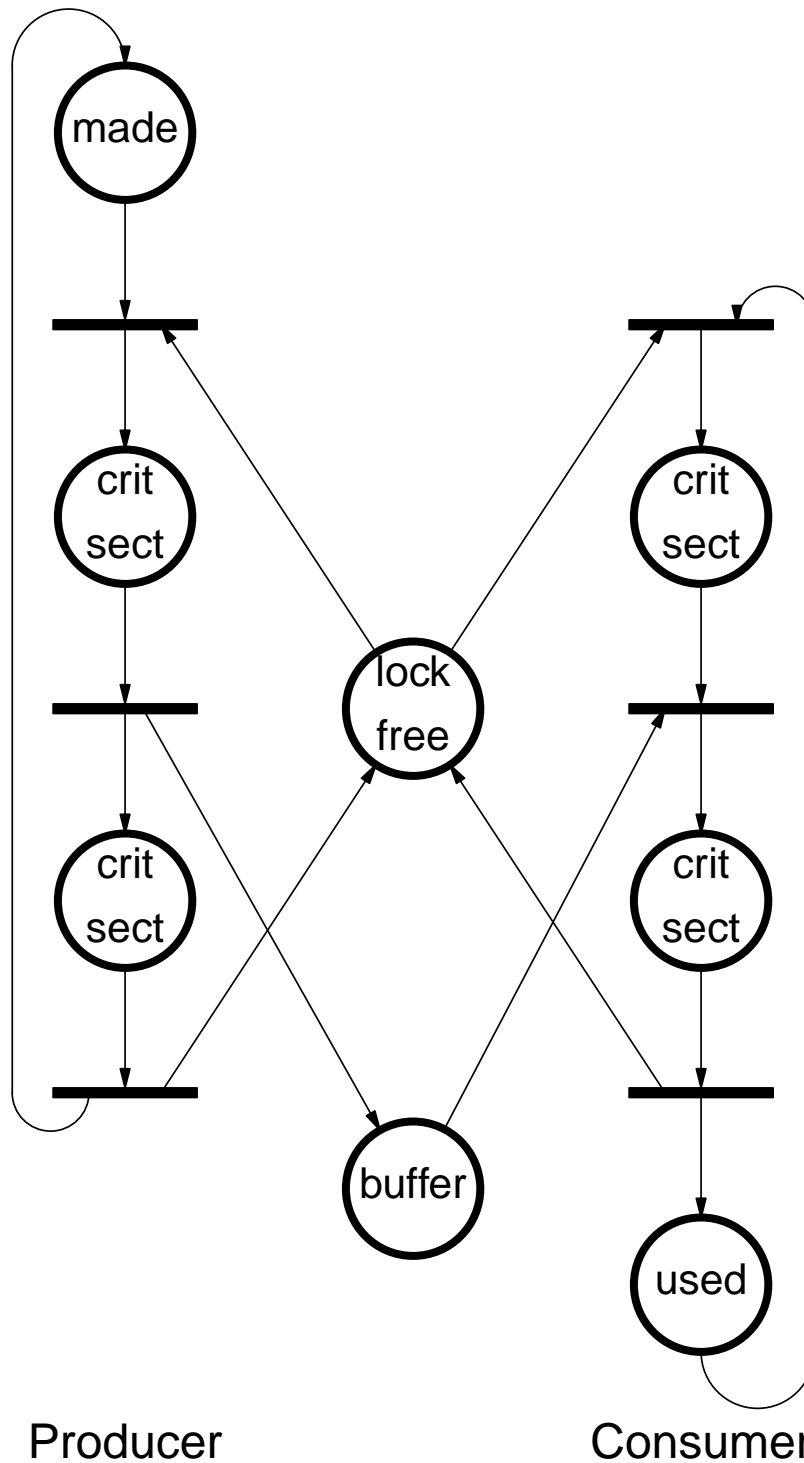
Objective: prevent two programs from accessing the same data structure simultaneously

Code where a program accesses the shared structure is called the critical section

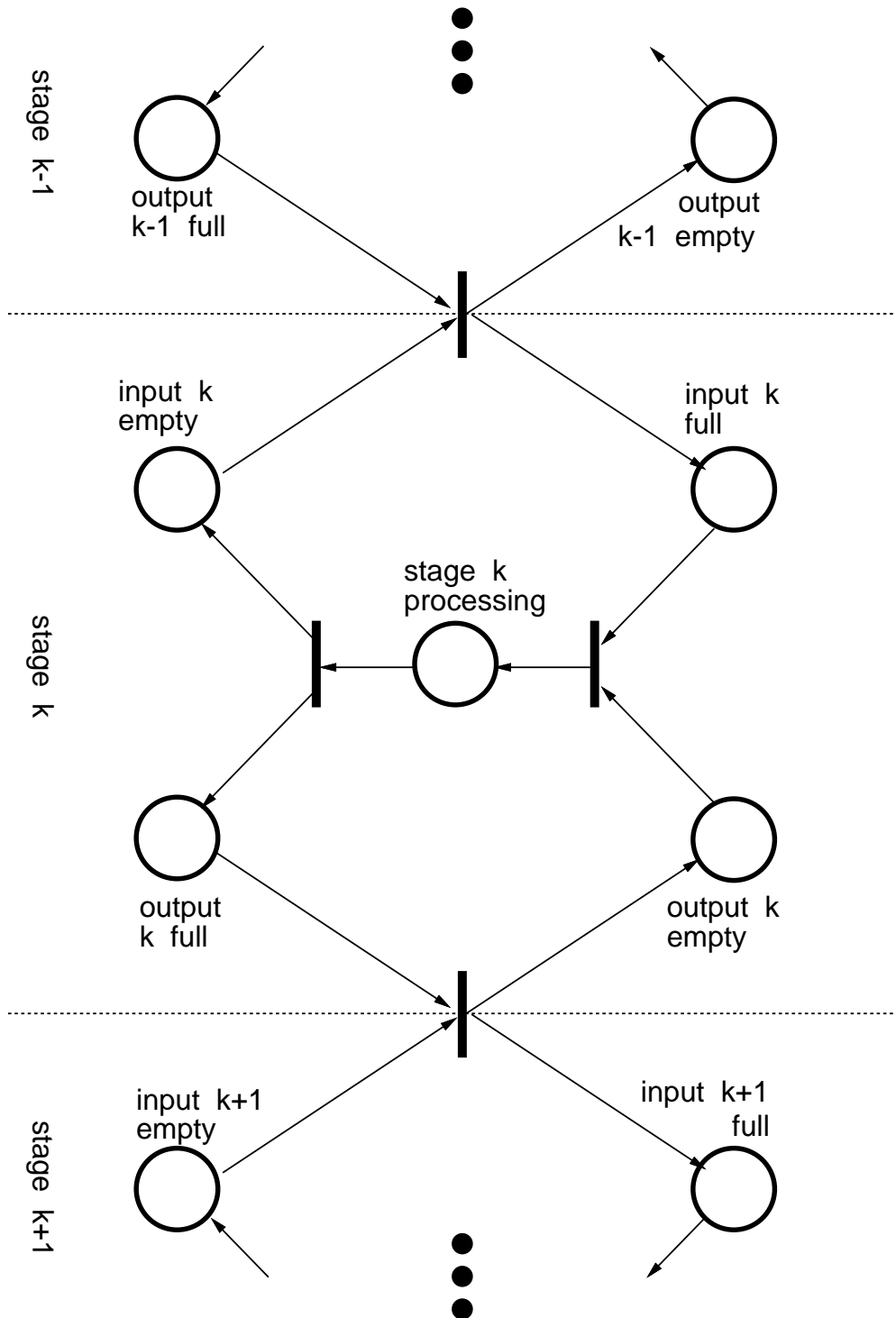


Producer/Consumer Model

Issue is lock on buffer to prevent simultaneous access



Process Pipeline



Machine Shop Specification

A machine shop has resources:

- machines M_0 , M_1 , and M_2
- operators O_1 and O_2

Subject to the constraints:

- O_1 can operate M_0 and M_1
- O_2 can operate M_0 and M_2
- an order is processed first by M_0 and then either M_1 or M_2

Significant events:

- order arrives
- O_i [begins | ends] operating M_j
- product is shipped

Significant conditions:

- order waiting for M_j
- O_i or M_j is idle (5 cases)
- O_i is operating M_j (4 cases)
- order partially done
- product awaiting shipment

Petri Net Can Model Any Flow

- PERT networks
 - ◇ example: building a house
- Many complex processes in society
 - ◇ example: civil legal action

Formal Definitions

A Petri net graph consists of

P a finite set of conditions (Places)

T a finite set of events (Transitions)

I a finite multiset of arcs from P to T (Inputs)

O a finite multiset of arcs from T to P (Outputs)

A marked Petri net consists of

G a Petri net graph $\langle P, T, I, O \rangle$

μ a function, $\mu : P \rightarrow \mathbb{Z}$
 \mathbb{Z} = non-negative integers

Firings

Formally define:

- event enabled
- firing
- firing sequence

Vector Addition Systems

An (almost) equivalent formalism:

- an n place Petri net written as a vector of conditions $\langle p_1, p_2, \dots, p_n \rangle$
- a marking of that net is written as a vector of non-negative integers $\langle m_1, m_2, \dots, m_n \rangle$
- an event is written as a vector of integers $\langle d_1, d_2, \dots, d_n \rangle$
 - ◇ $d_i < 0$ for input arcs from condition p_i
 - ◇ $d_i > 0$ for output arcs to condition p_i
 - ◇ one event cannot have same condition as both input and output

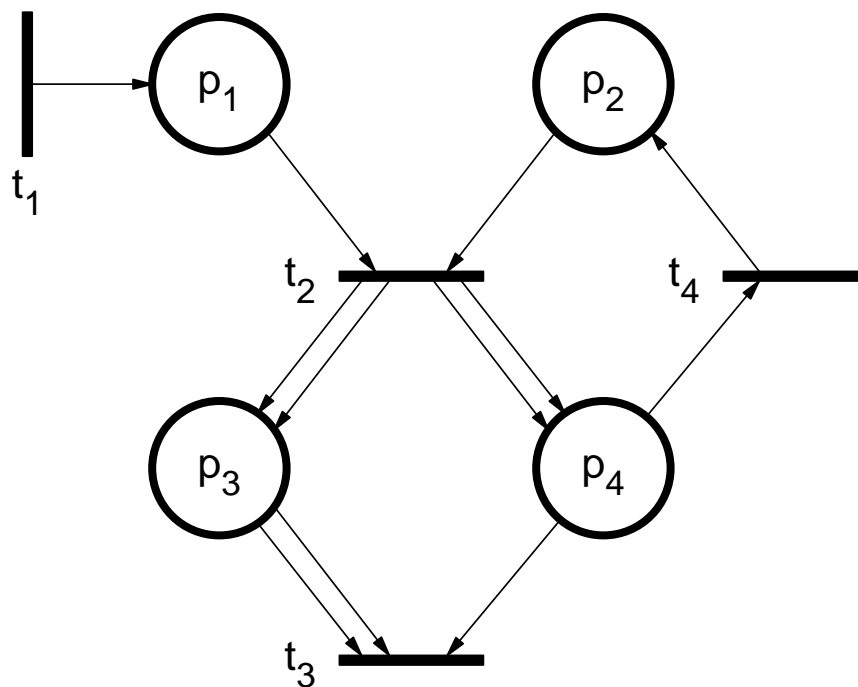
| | conditions |
|---|------------|
| e | |
| v | |
| e | |
| n | |
| t | |
| s | |

Example VAS

A Petri net in VAS notation

| | p_1 | p_2 | p_3 | p_4 |
|-------|--------------|--------|--------|----------------|
| t_1 | $\langle 1$ | $, 0$ | $, 0$ | $, 0 \rangle$ |
| t_2 | $\langle -1$ | $, -1$ | $, 2$ | $, 2 \rangle$ |
| t_3 | $\langle 0$ | $, 0$ | $, -2$ | $, -1 \rangle$ |
| t_4 | $\langle 0$ | $, 1$ | $, 0$ | $, -1 \rangle$ |

The same Petri net in graphical notation



Analysis of Behavior

Given

- a Petri net with
- an initial marking

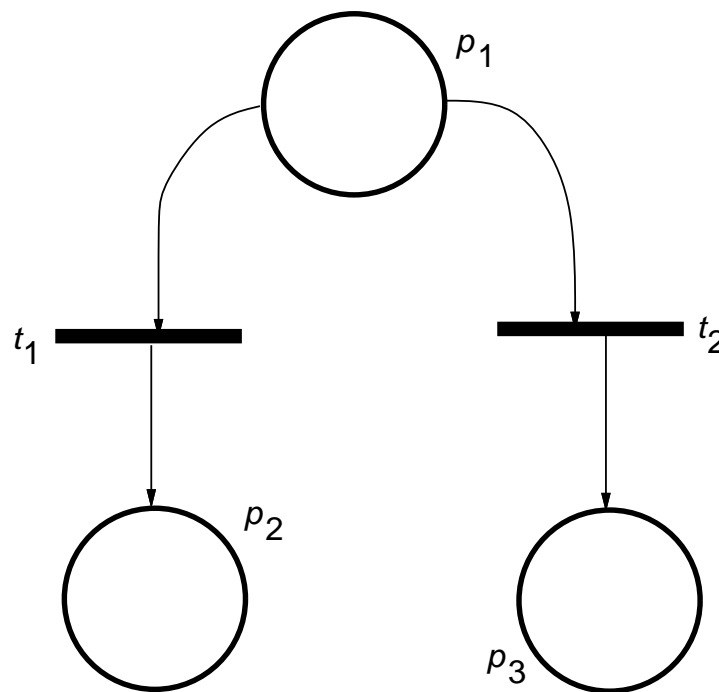
determine

- behavior patterns
- over *all possible* firing sequences

Conflict

Two events are said to be in conflict if there is a marking in which both events are enabled, but the firing of one event disables the other

A conflict:



Boundedness

A condition t is bounded by b if there is no firing sequence that marks t with a value greater than b

A condition bounded by 1 is called safe.

Liveness

An event t is

dead

if it is never enabled

live-1

if there is some firing sequence after which t is enabled

live-2

if, for every integer n , there is a firing sequence in which t fires n times

live-3

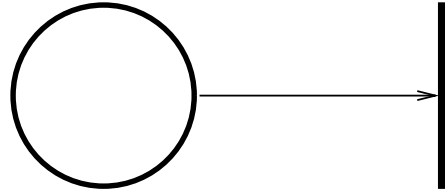
if there is an infinite firing sequence in which t fires n times, for any integer n

live-4

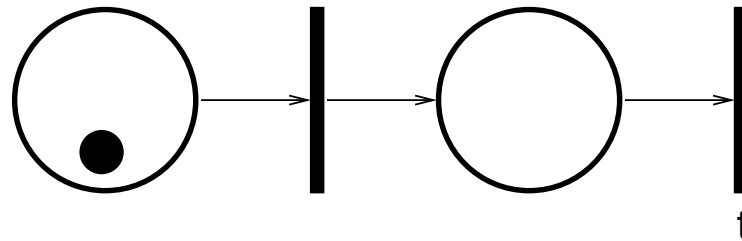
if every finite firing sequence can be extended to one that enables t

Liveness Examples

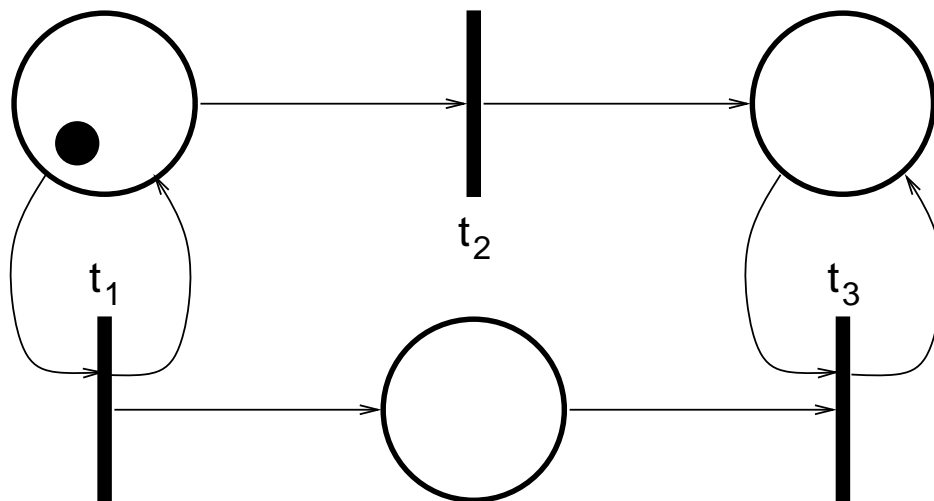
- dead



- live-1 but not live-2



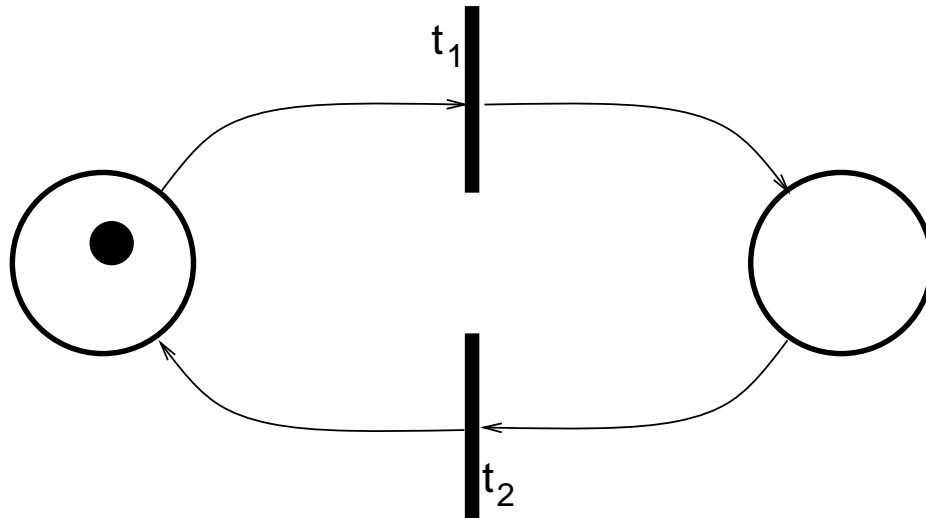
- live-2 but not live-3, for t_3



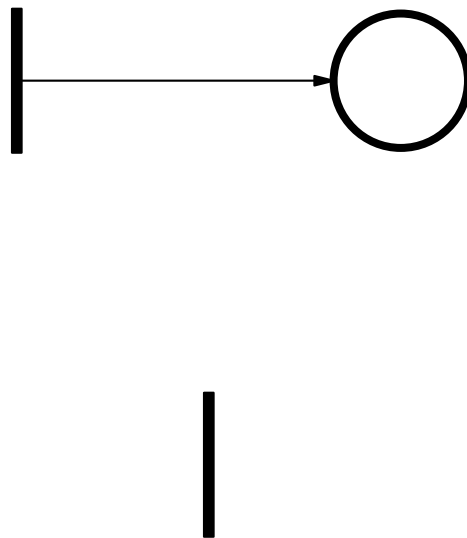
- live-3 but not live-4, for t_1 in same net

Liveness Examples (continued)

- live-4, non-trivially



- live-4, but trivial



Reachability & Coverability

Reachability question: Given

- a Petri net with an initial marking and
- a target marking \mathcal{M}

does there exist

- a firing sequence that produces \mathcal{M} ?

A marking $\langle m_1, m_2, \dots, m_n \rangle$ covers a marking $\langle k_1, k_2, \dots, k_n \rangle$ if $k_i \leq m_i$ for all i , $1 \leq i \leq n$

Coverability question: Given

- a Petri net with an initial marking and
- a target marking \mathcal{M}

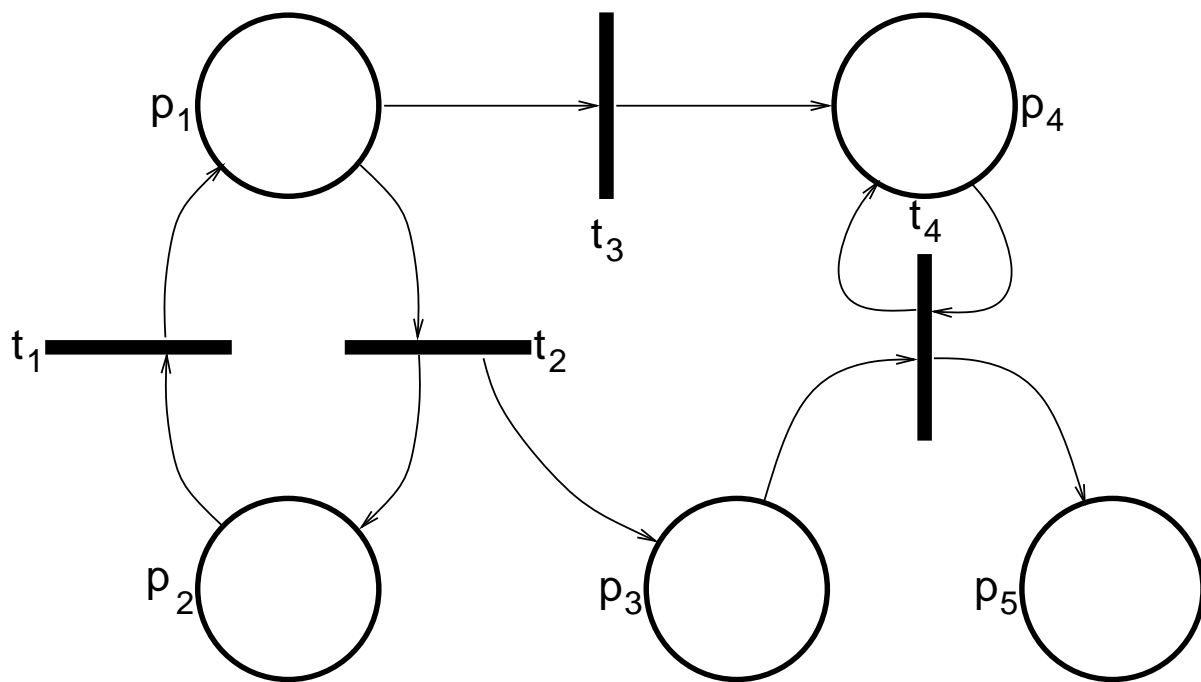
does there exist

- a firing sequence that produces a marking covering \mathcal{M} ?

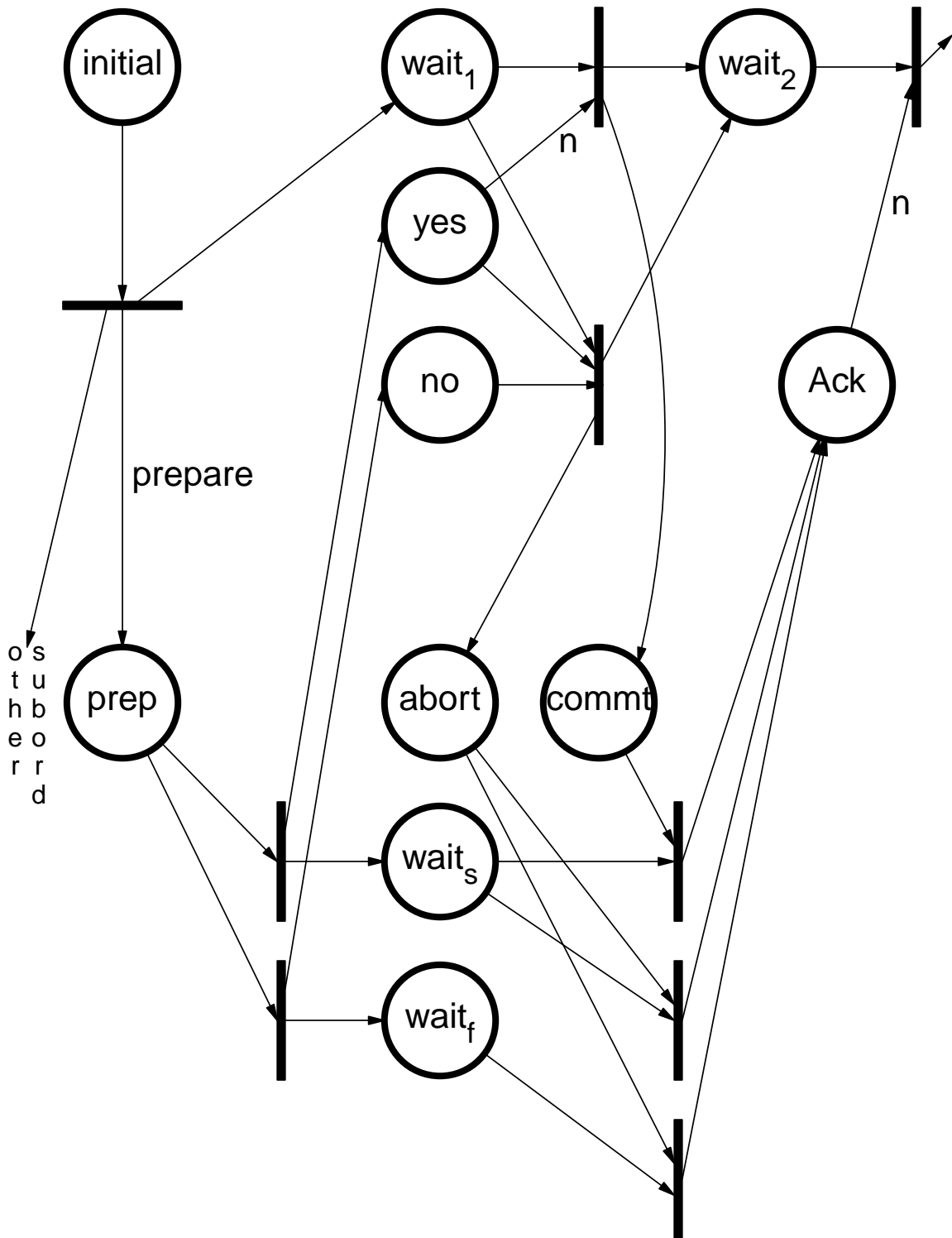
Coverability tree formed by

1. initial marking is root
2. successive markings form branches except
3. if identical marking on path to root, stop branch
4. if covers a marking on path to root, mark increased conditions with ω

Coverability Tree Example



Two-Phase Commit



OS Model Correspondences

A condition typically is

- a queue of processes
 - ◊ where many tokens may be in one condition
- an active process
 - ◊ where one token may occupy a condition

Tokens model

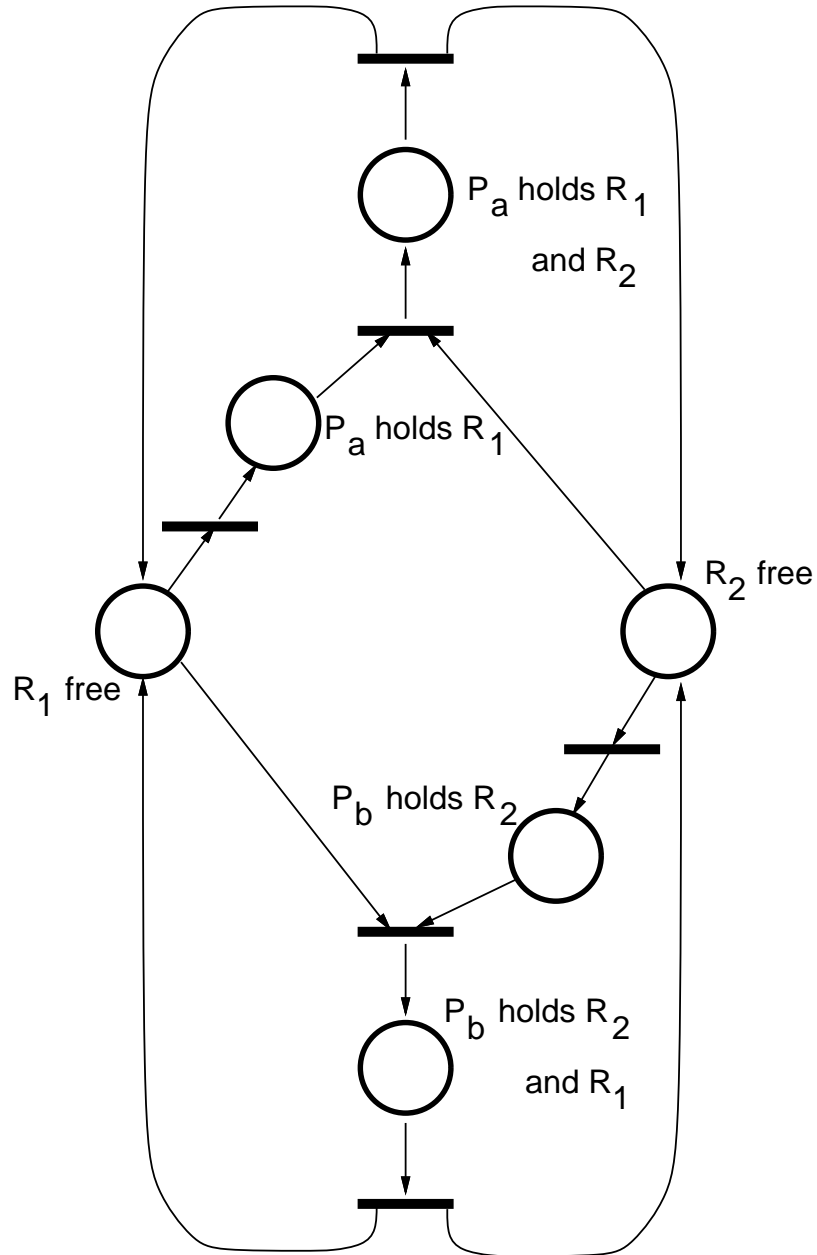
- processes
- controls (semaphores, *etc.*)

Deadlocked Programs

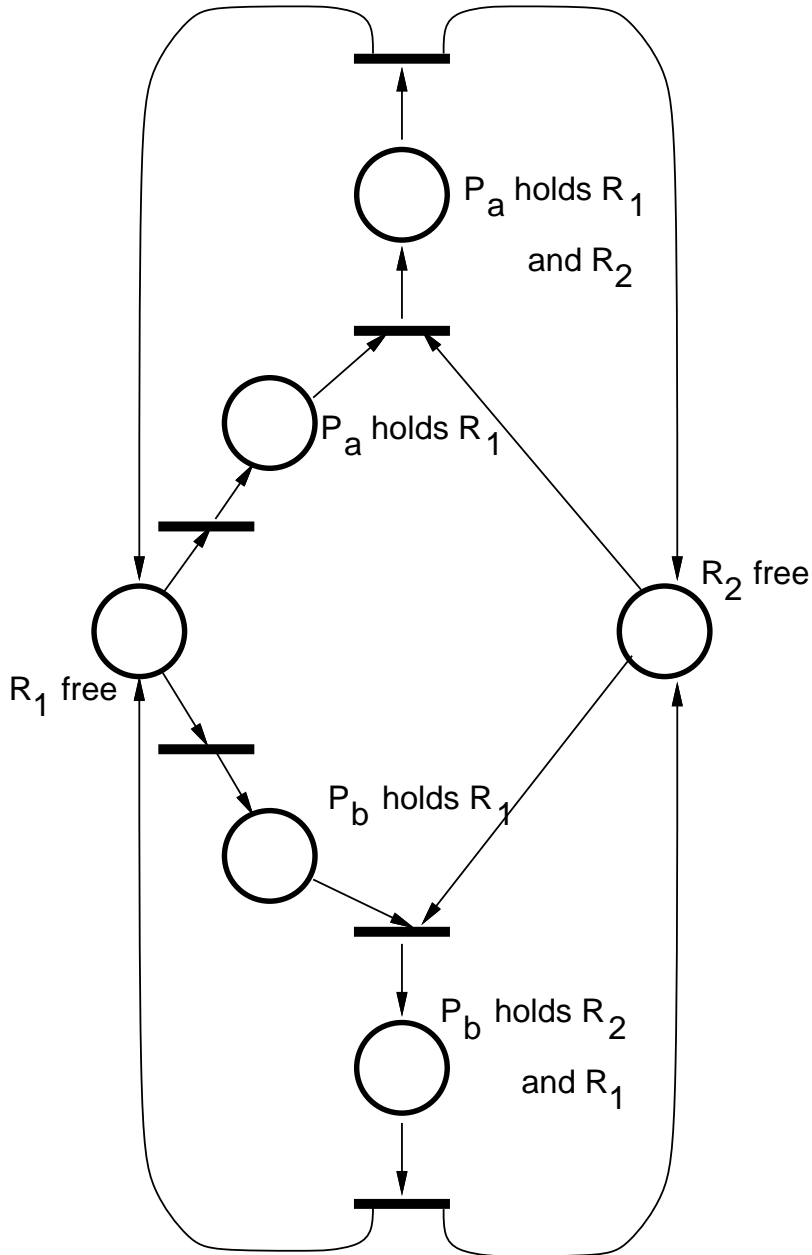
```
loop
  acquire R1;
  acquire R2;
  critical
    section;
  release R1
    and R2;
end
```

```
loop
  acquire R2;
  acquire R1;
  critical
    section;
  release R1
    and R2;
end
```

Petri Model with Possible Deadlock



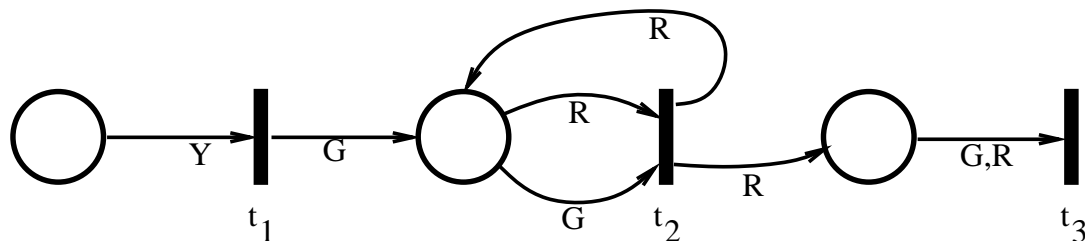
Petri Model without Deadlock



Colored Petri Nets

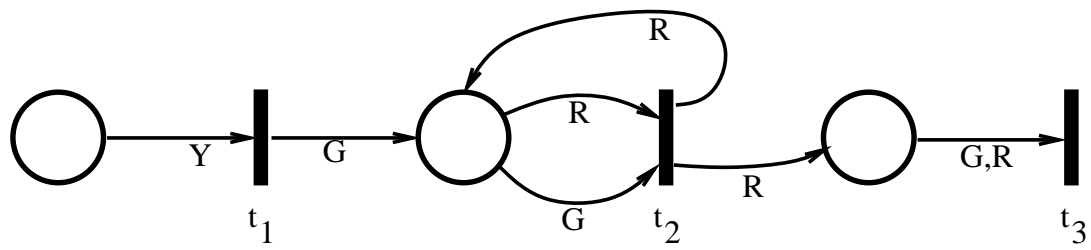
- allow tokens of various colors
- events specify colors of tokens input and output

Example:



- ◇ t_1 takes a Yellow token and outputs a Green one
- ◇ t_2 consumes a Green token as long as a Red is present, outputting a Red token
- ◇ t_3 takes either a Green or Red token as input

Coloring Doesn't Matter



Numeric Petri Nets

