Lecture 2

Transition Systems

Modeling: informal motivation

Model is a simplification of a system for understanding and analysing particular aspects

In e.g., control engineering, models describe relation between (real-valued) quantities

In protocol and software verification, models should describe possible combinations of sequences of computation steps

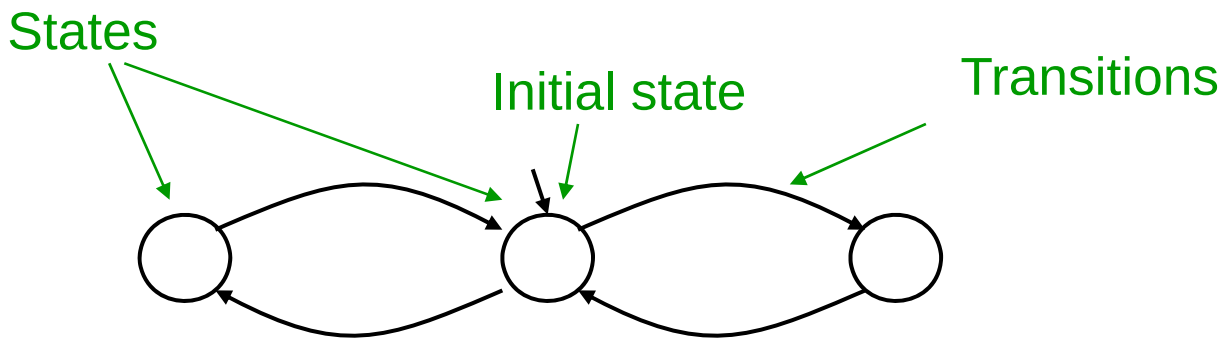
=>

Models will have discrete states and transitions between states.

Transition Systems

Model to describe the operational behavior of systems

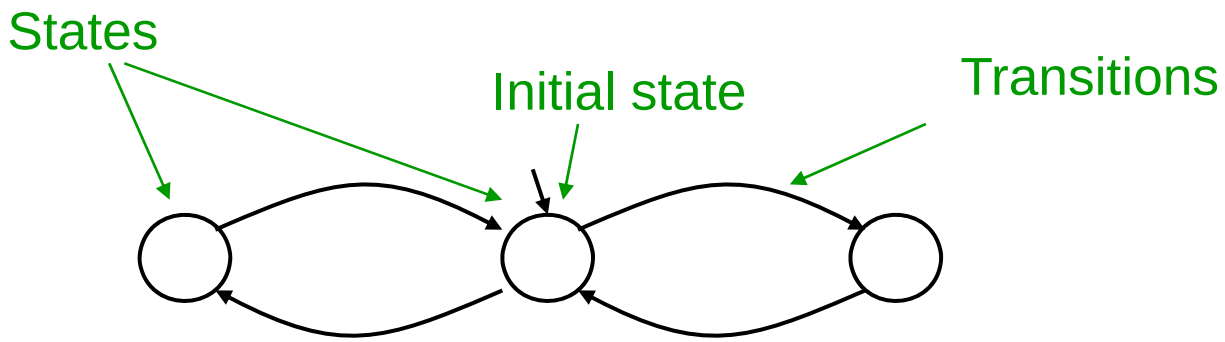
Example: Resource Allocation System:



Transition Systems

Model to describe the operational behavior of systems

Example: Resource Allocation System:



We need a mechanism to describe “what happens” in a transition system.

I.e.: A language of assertions about “current” state,

We should be able to say, e.g.,

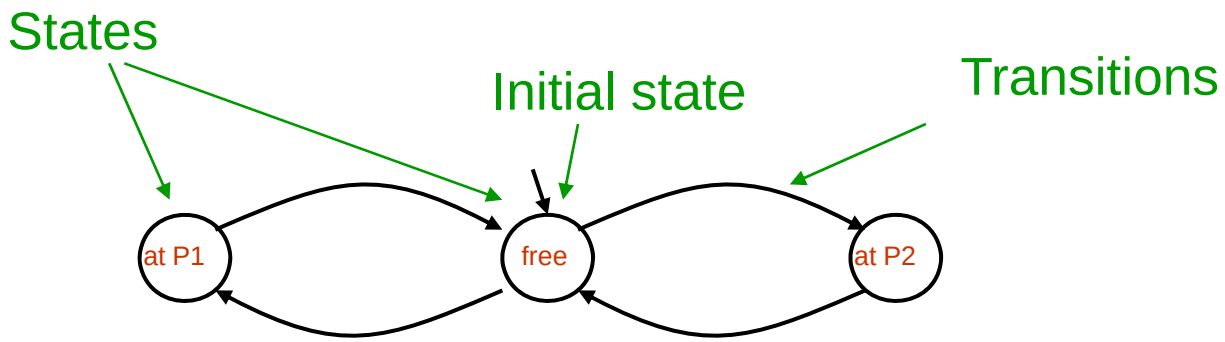
- “The process P1 holds the resource”
- “No process holds the resource”

Each assertion can be true or false in the “current state”.

Transition Systems

Model to describe the operational behavior of systems

Example: Resource Allocation System:

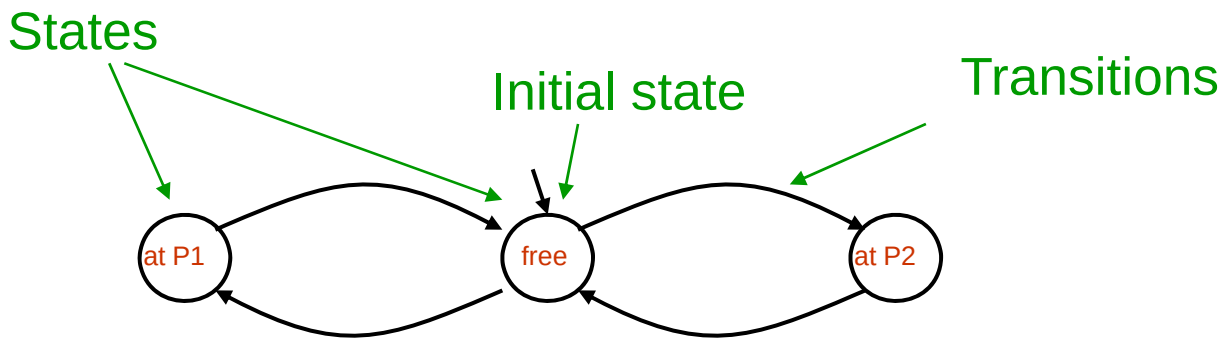


Variables: status (what can be “observed”)

Transition Systems

Model to describe the operational behavior of systems

Example: Resource Allocation System:



Variables: status (what can be “observed”)

Computation:

Transition Systems (formal definition)

A Transition System is a tuple $(S, S_0, \rightarrow, V, L)$ where

- S is a set of states
- S_0 is a set of initial states
- $\rightarrow$ (a subset of $S \times S$) is a transition relation
- V is a set of variables
- L : $S \rightarrow (V \rightarrow Val)$ gives a valuation of variables in each state

Write $s \rightarrow s'$ for $(s, s') \in \rightarrow$

Transition Systems (formal definition)

A Transition System is a tuple $(S, S_0, \rightarrow, V, L)$ where

- S is a set of states
- S_0 is a set of initial states
- $\rightarrow$ (a subset of $S \times S$) is a transition relation
- V is a set of variables
- L : $S \rightarrow (V \rightarrow Val)$ gives a valuation of variables in each state

Write $s \rightarrow s'$ for $(s, s') \in \rightarrow$

A Computation is a finite or infinite sequence of states

s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9

such that

- s_0 is an initial state
- $(s_i, s_{i+1}) \in \rightarrow$ for all relevant i

Structured Descriptions of Transition Systems

Drawing large graphs is very inconvenient.

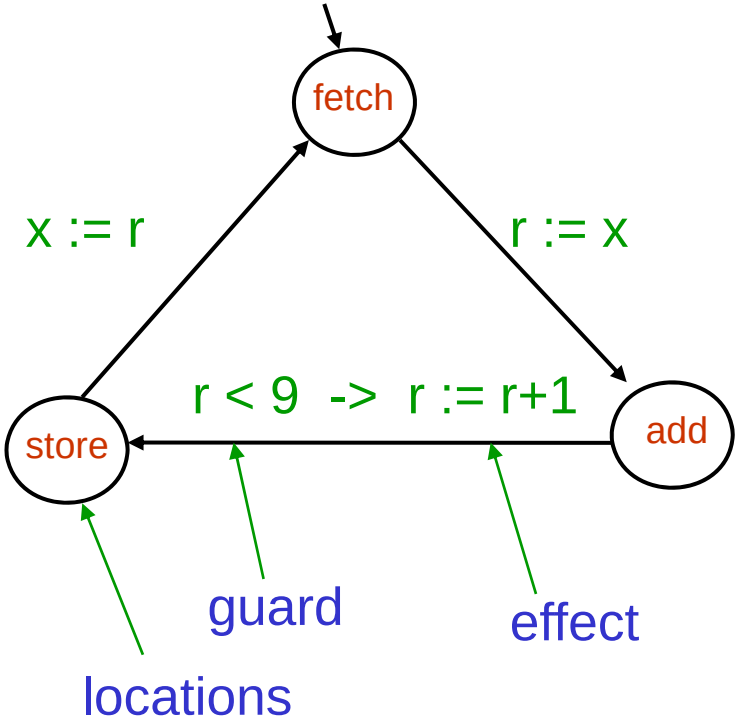
We need constructs to define transition systems conveniently

- Processes
- control structures
- (shared) variables
- communication channels
  - Can be modeled as shared variables

Adding Variables (Action Systems)

Example: simple incrementation of stored variable

Action System
Variables: x, r: integer
Initially x = 0



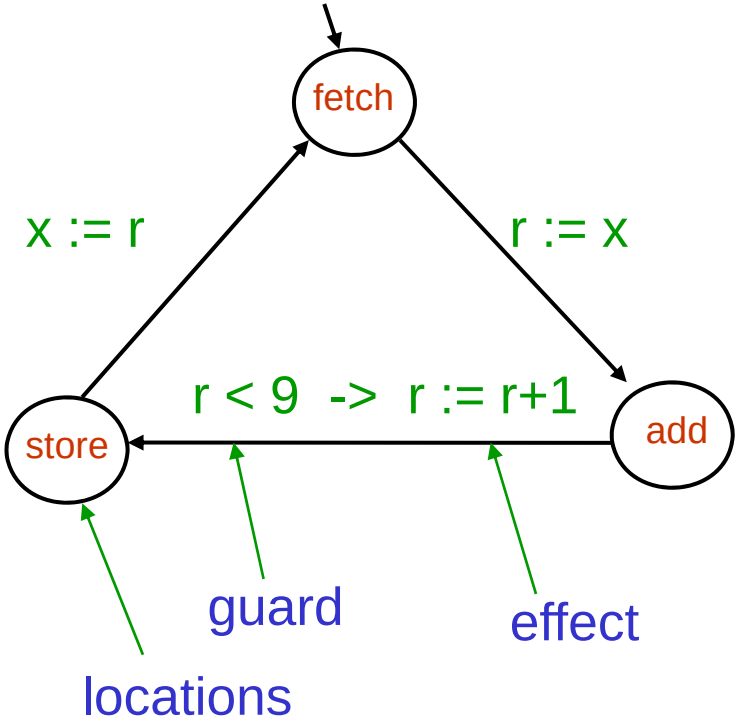
Adding Variables (Action Systems)

Example: simple incrementation of stored variable

Action System

Variables: x, r: integer

Initially x = 0



Transition system

States:

locations × assignments to variables

(add, {x -> 0 , r -> 0})

Initial States (example):

(fetch, {x -> 0 , r -> 76})

Transitions:

(add, {x -> 0 , r -> 0}) ->
(store, {x -> 0 , r -> 1})

Variables:

loc x, r

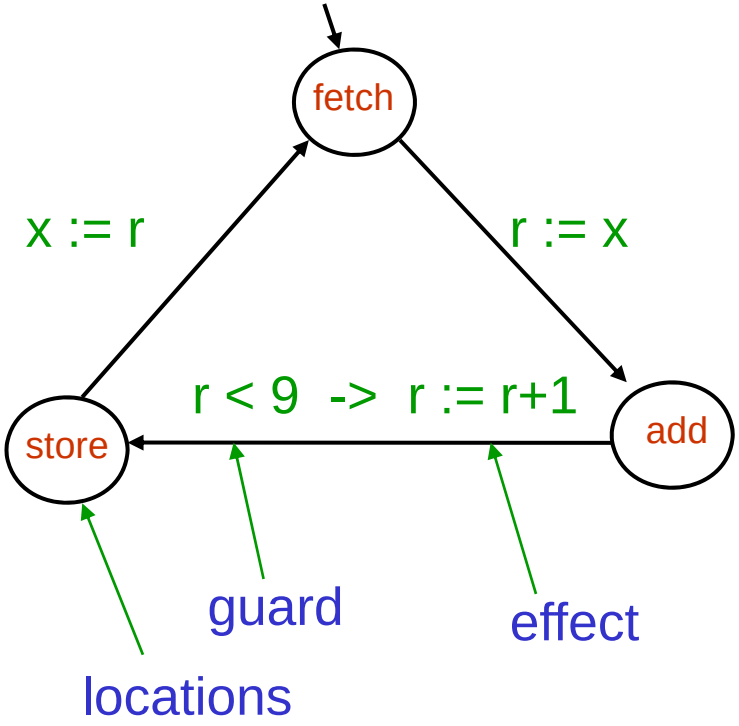
Valuation:

(add, {x -> 0 , r -> 0}) (x) = 0

(add, {x -> 0 , r -> 0}) (loc) = add

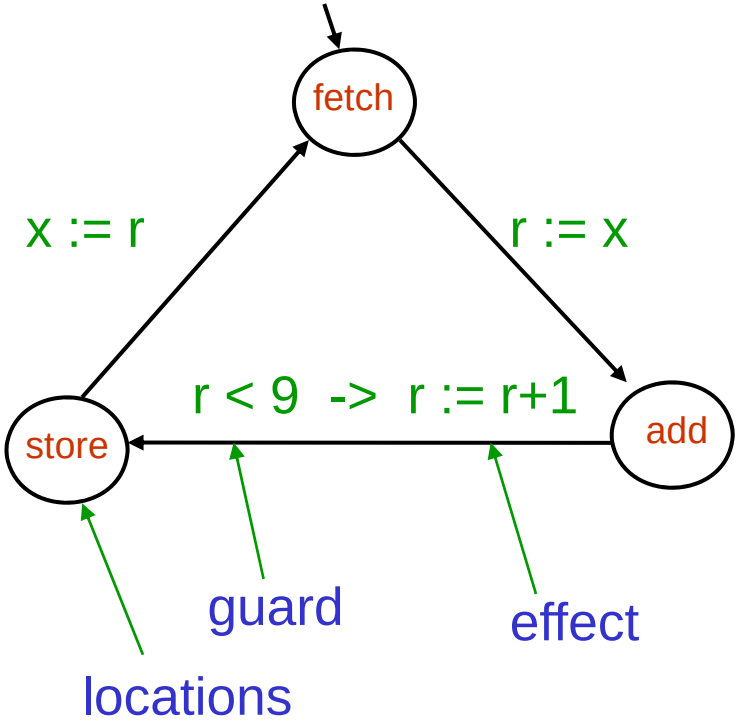
Corresponding Transition System

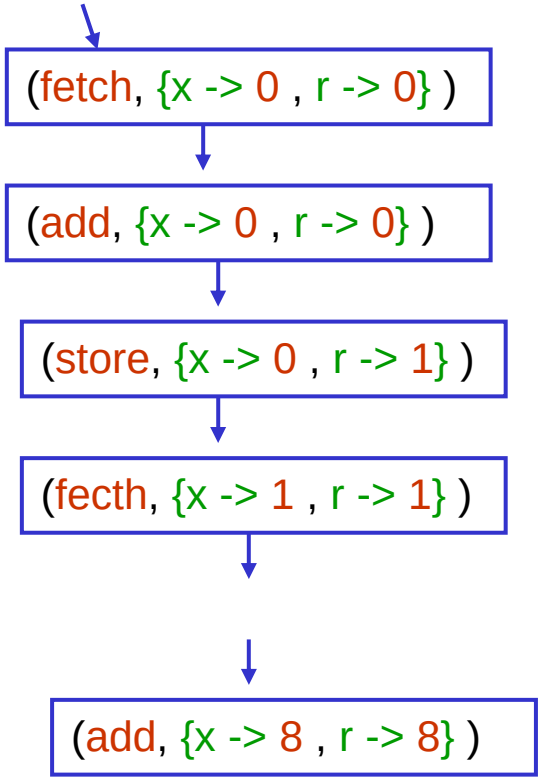
Action System
Variables: x, r: integer
Initially x = 0



Corresponding Transition System

Action System
Variables: x, r: integer
Initially x = 0

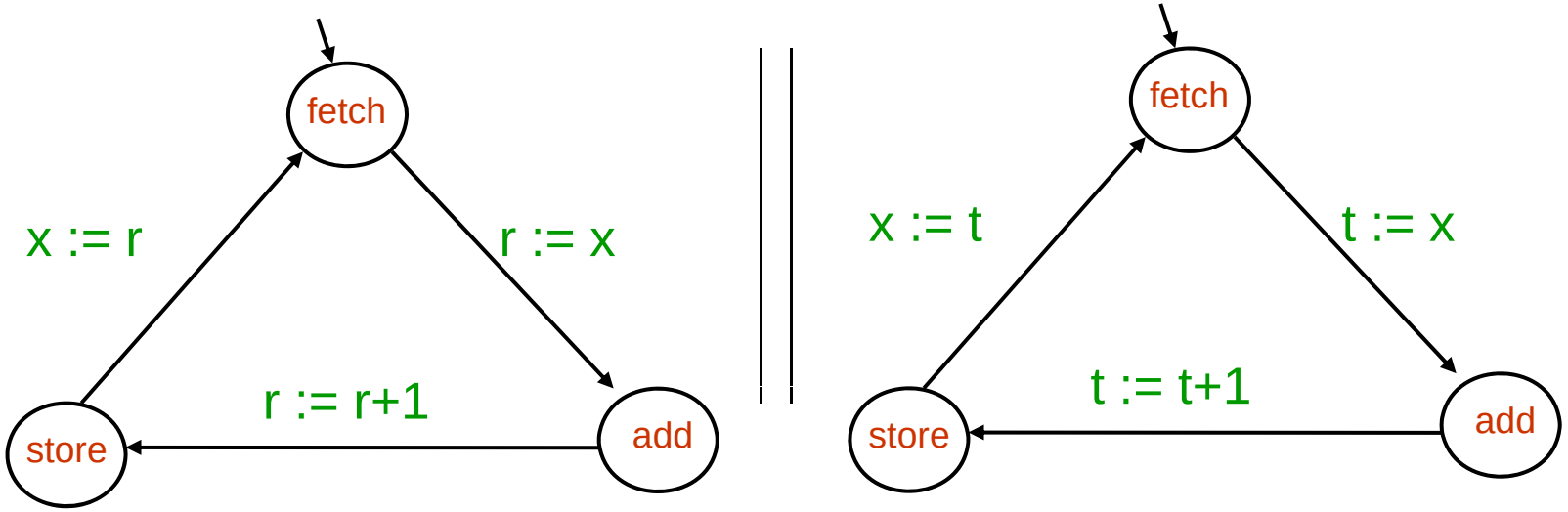




Parallel Processes

Program graphs can be put in parallel

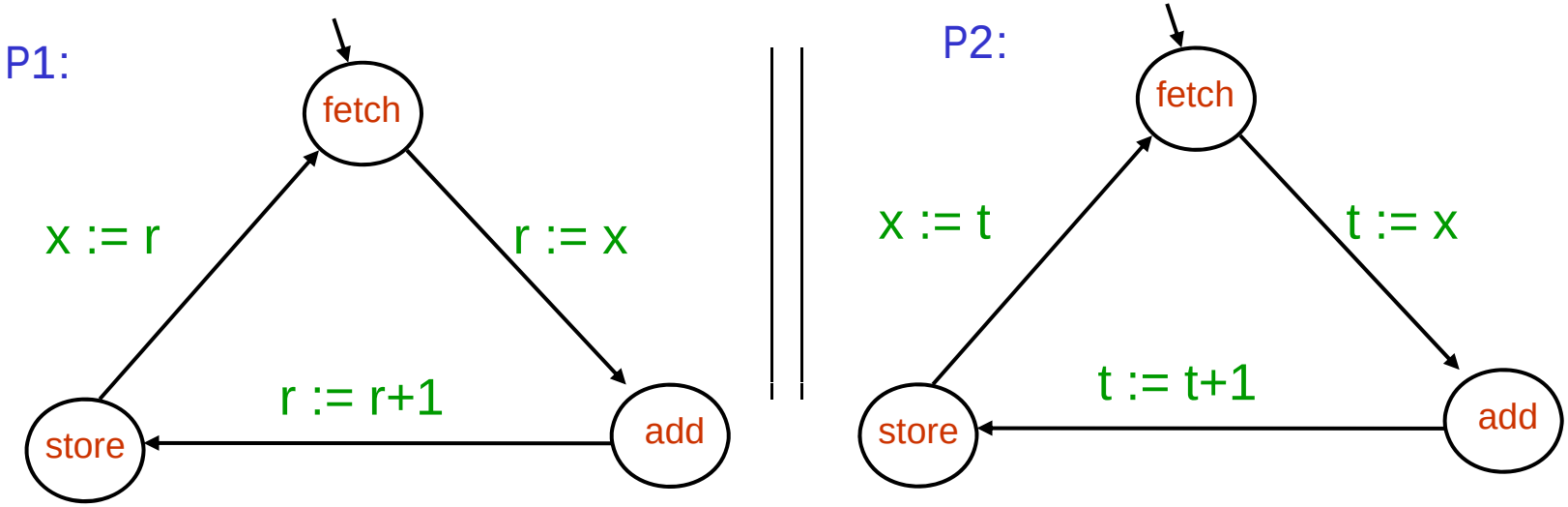
Variables: x, r, t: integer
Initially x = 0



Parallel Processes

Program graphs can be put in parallel

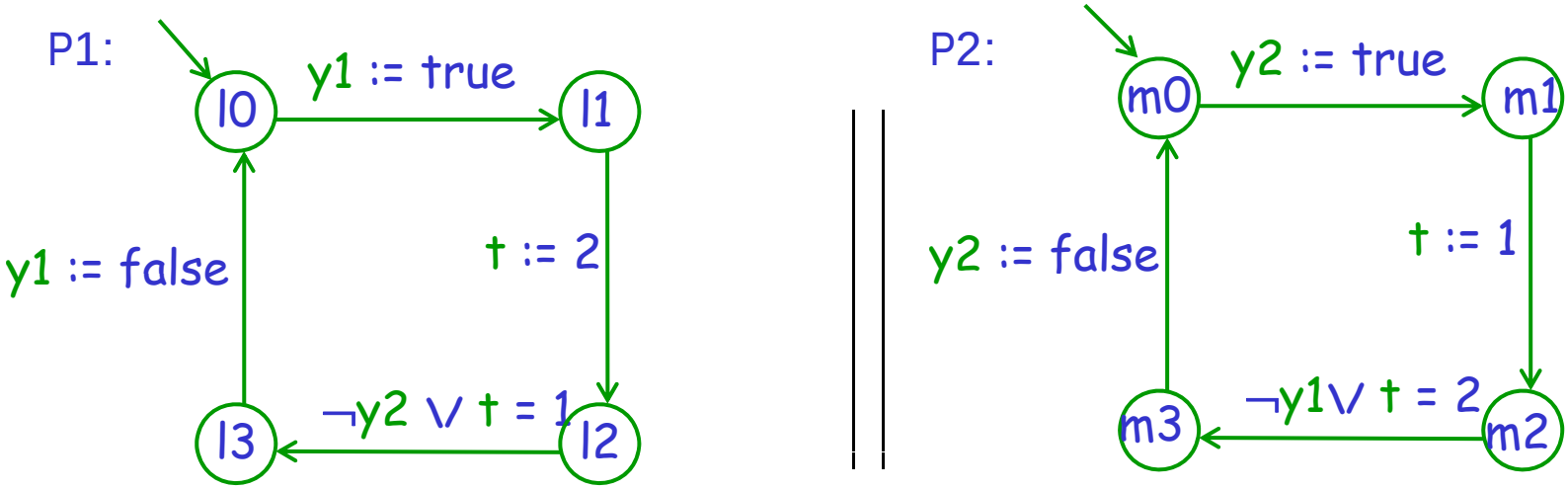
Variables: x, r, t: integer
Initially x = 0



States: (add, fetch, {x -> 0 , r -> 0 , t -> 0})
Transitions: (add, fetch, {x -> 0 , r -> 0 , t -> 0})
-> (store, fetch, {x -> 0 , r -> 1 , t -> 0})
Variables: loc[P1], loc[P2], x, r , t

Peterson-Fischer Mutual Exclusion prot.

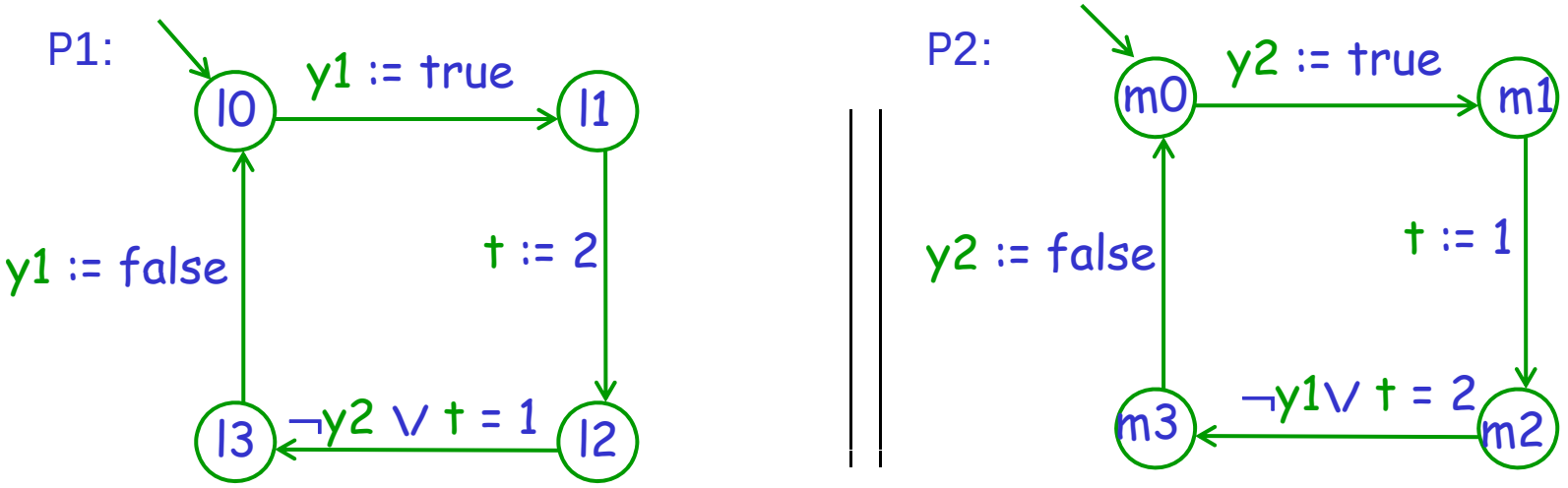
Variables: y1, y2: boolean, t: {1,2}
Initially y1 = y2 = false, t = 1



→

Peterson-Fischer Mutual Exclusion prot.

Variables: y1, y2: boolean, t: {1,2}
Initially y1 = y2 = false, t = 1



States: (l1 , m1, {y1 -> true ,y2 -> true , t -> 1})
Initial States: (l0 , m0, {y1 -> false ,y2 -> false , t -> 1})

Transition:

(l0 , m0, {y1 -> false ,y2 -> false , t -> 1}) → (l1 , m0, {y1 -> true ,y2 -> false , t -> 1})

Interleaving

Transitions of parallel components are merged or “interleaved”

- as if the model is “executed” on a single processor

The order between transitions of different components is completely arbitrary

- as if there is a scheduler with an unknown strategy
- possible that some components are scheduled “very seldom” or never at all

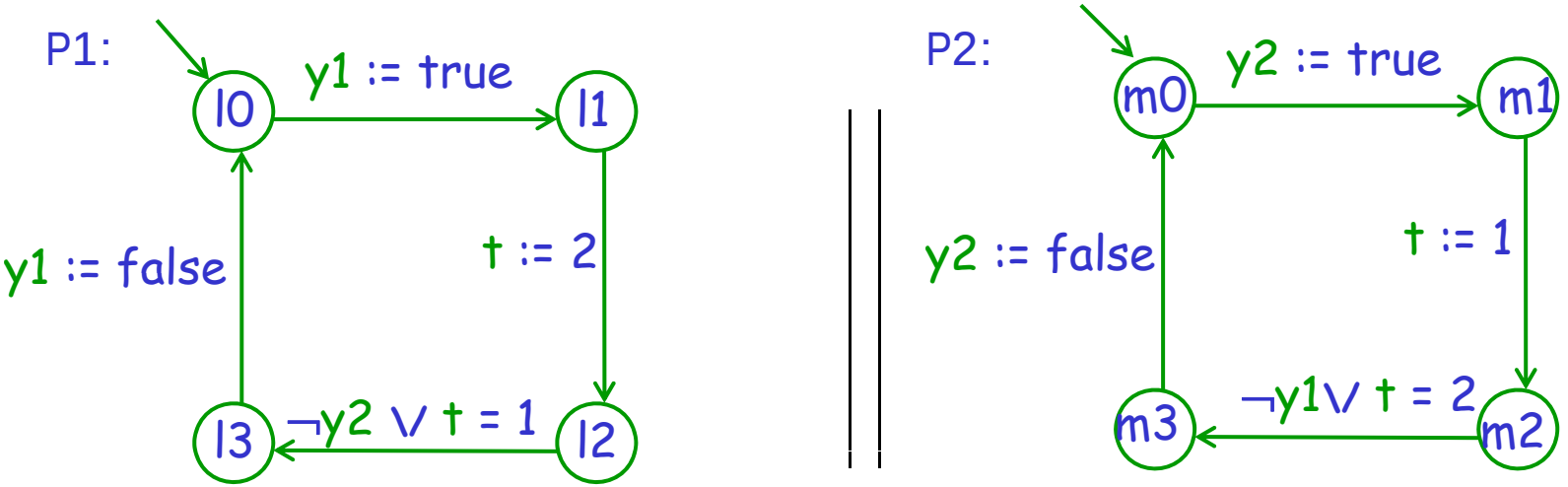
Particular scheduling strategies can be enforced by adding a “scheduler” into the model

- e.g., by using shared variable(s) to model the state of a scheduler

Certain minimal assumptions on scheduling can be represented using fairness (more about this later)

Peterson-Fischer Mutual Exclusion prot.

Variables: y1, y2: boolean, t: {1,2}
Initially y1 = y2 = false, t = 1

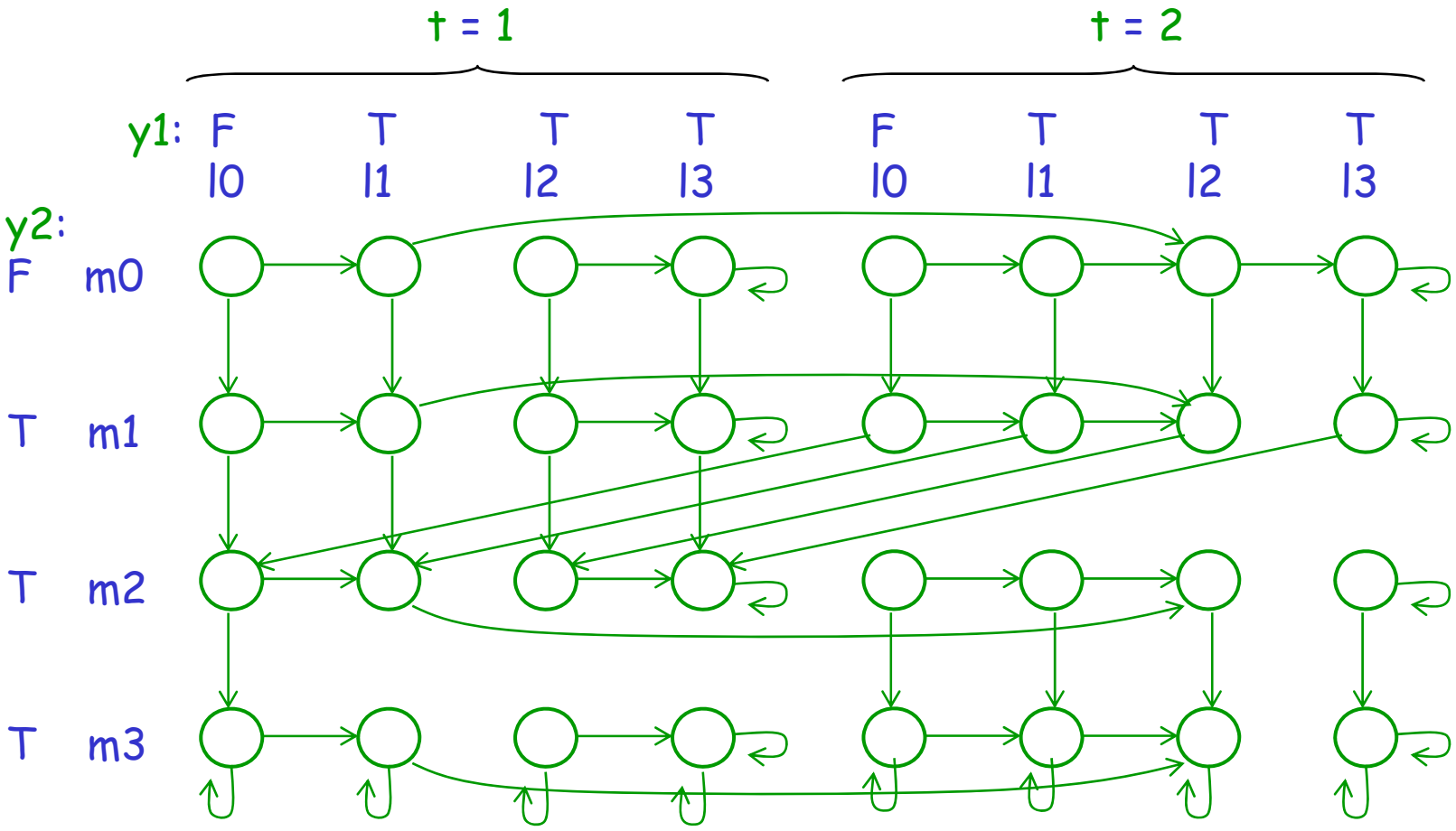


States: (l1 , m1, {y1 -> true ,y2 -> true , t -> 1})
Initial States: (l0 , m0, {y1 -> false ,y2 -> false , t -> 1})

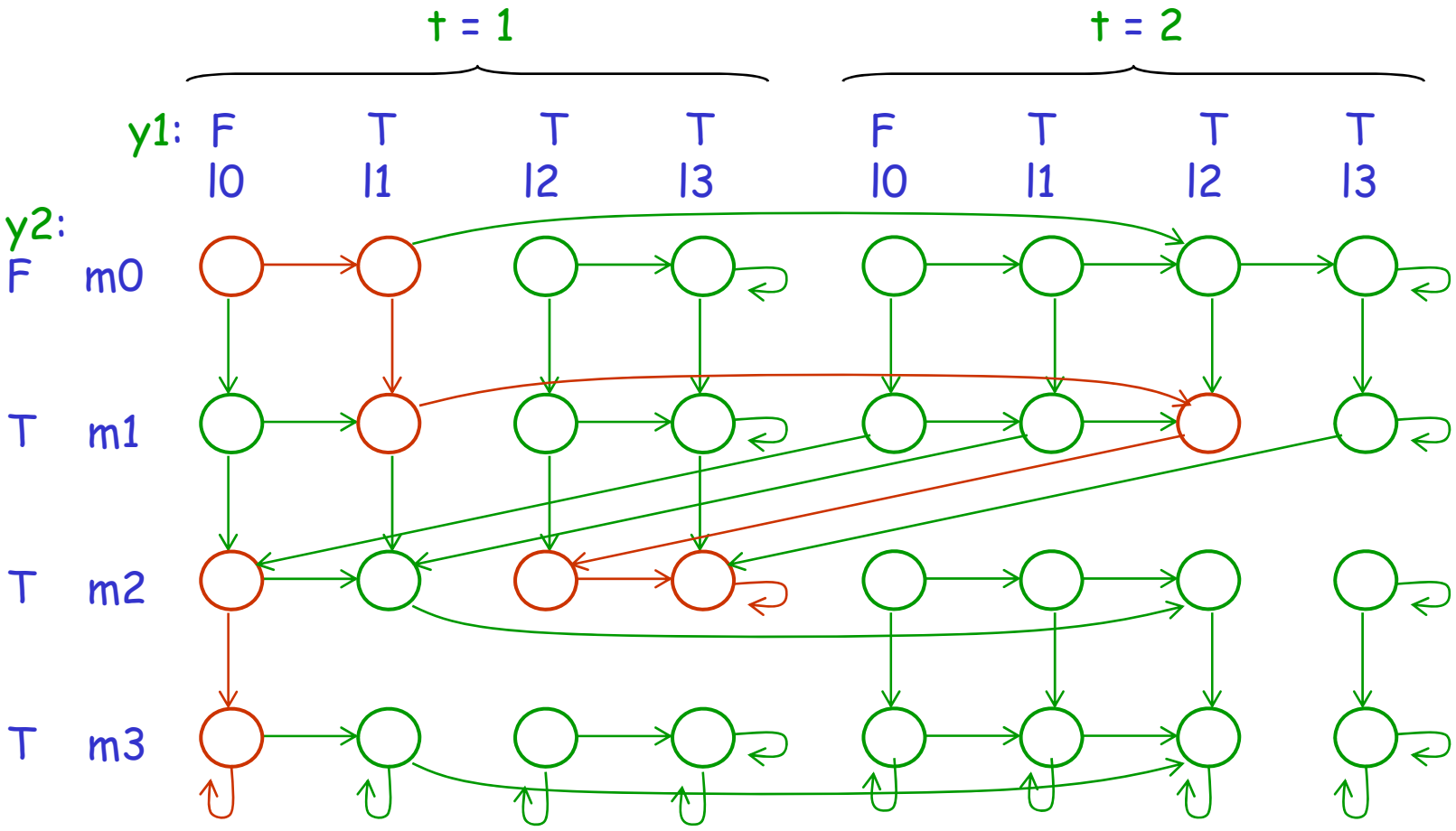
Transition:

(l0 , m0, {y1 -> false ,y2 -> false , t -> 1}) → (l1 , m0, {y1 -> true ,y2 -> false , t -> 1})

Peterson-Fischer Transition System



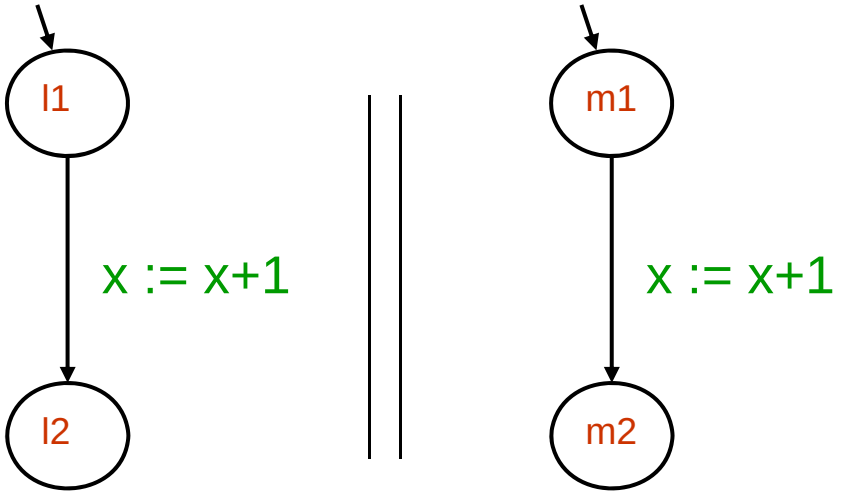
Peterson-Fischer: Computation



Atomicity

Each transition is atomic, i.e., it is an undivisible operation, which has no "intermediate states", i.e., can not be stopped "in the middle"

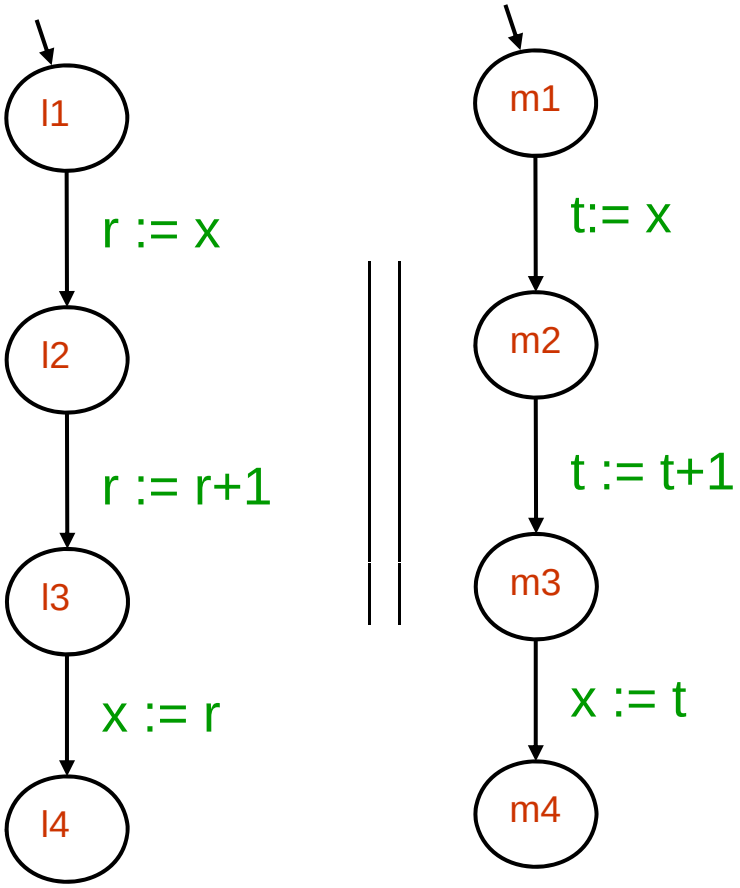
Variables: x: integer
Initially x = 0



This transition system will "terminate" with the value of x being 2

Atomicity

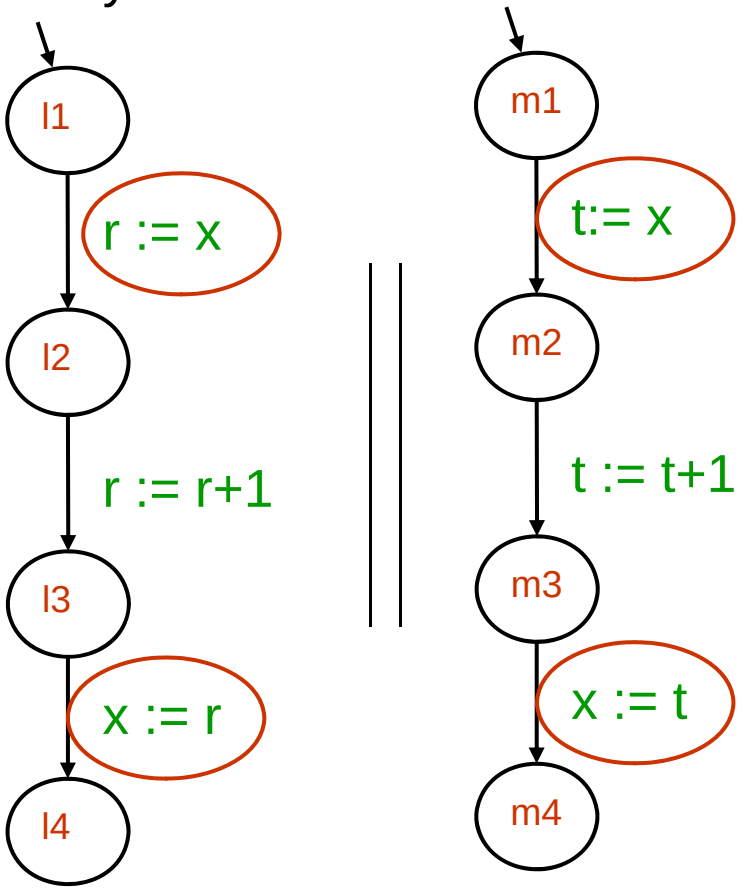
Variables: x, r, t: integer
Initially x = 0



This transition system will “terminate” with the value of x being 1 or 2

How much detail must be modeled?

Variables: x, r, t: integer
Initially x = 0



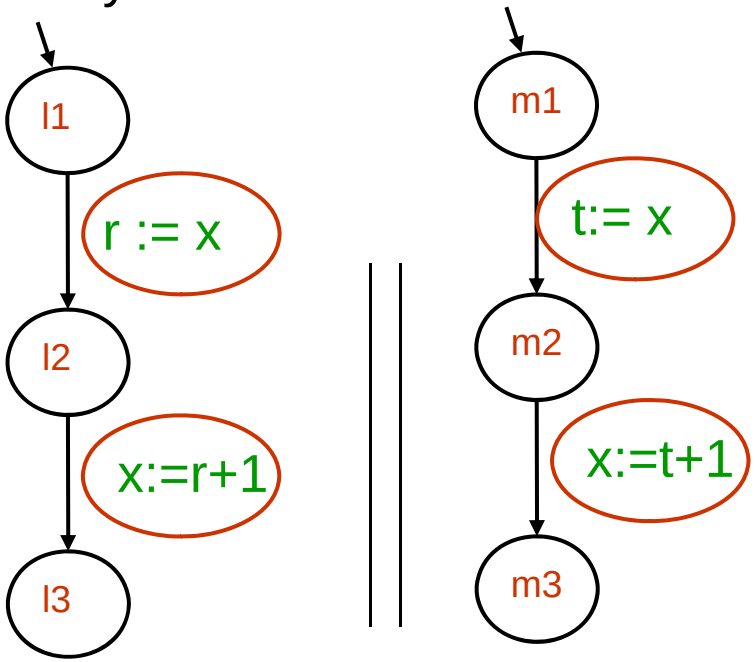
All possible sequences of ”externally interesting” transitions must be modeled

Critical event is an operation, for which is makes a difference how it is interleaved
- Reading shared varialbe
- Writing to shared variable

Recepy:
- Each transition should involve at most one critical event

Also a faithful model

Variables: x, r, t: integer
Initially x = 0



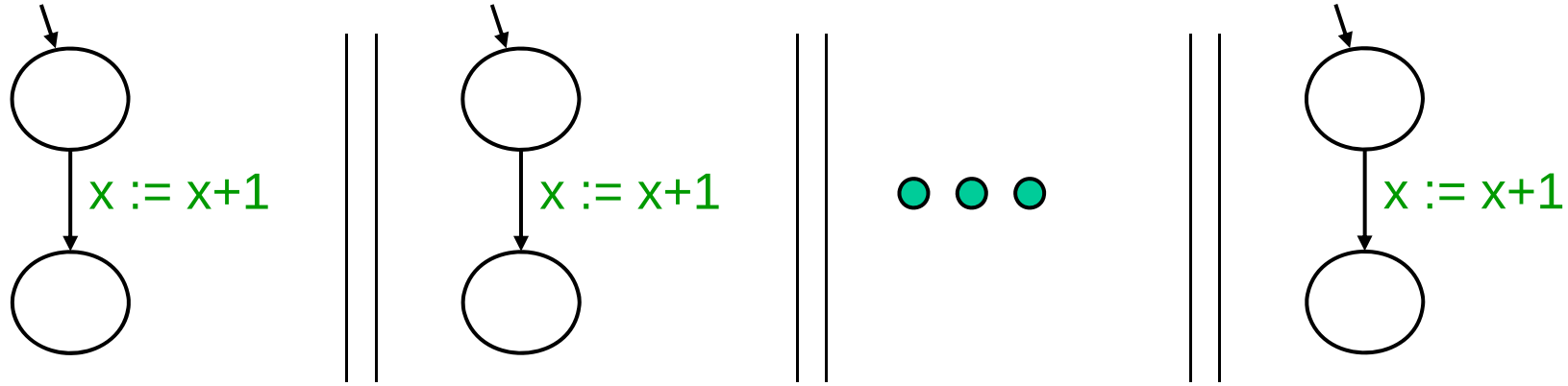
All possible sequences of ”externally interesting” transitions must be modeled

Critical event is an operation, for which is makes a difference how it is interleaved
- Reading shared varialbe
- Writing to shared variable

Recepy:
- Each transition should involve at most one critical event

Arguments for using Interleaving

- “Principle of relativity” (simultaneity can not be observed)
- Fewer transitions to consider in analysis

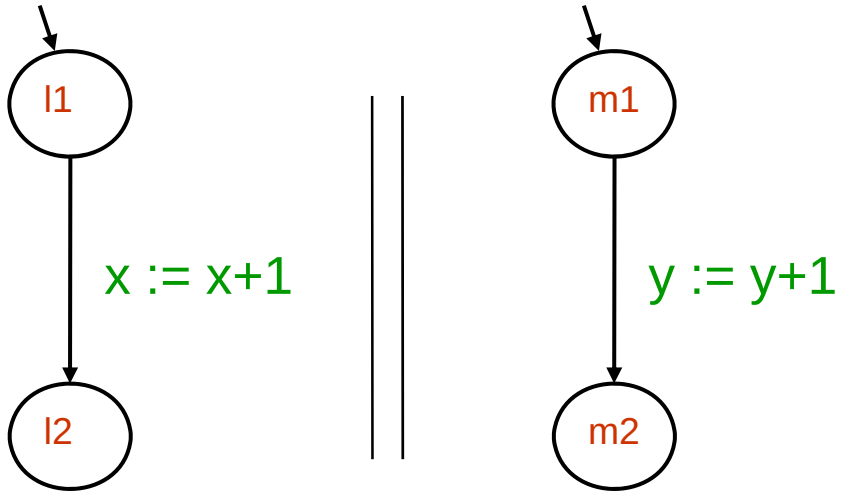


- In general: 2^n possible transitions
- With interleaving: n possible transitions

Limitations of interleaving

- Some (noninteresting) “pathological” properties are satisfied when using interleaving

Variables: x,y: integer
Initially x = y = 0



- Interleaving makes the system satisfy the (uninteresting) property:
  “there must be a (short) instant when x and y are different”

Small Idealized Example from Reality

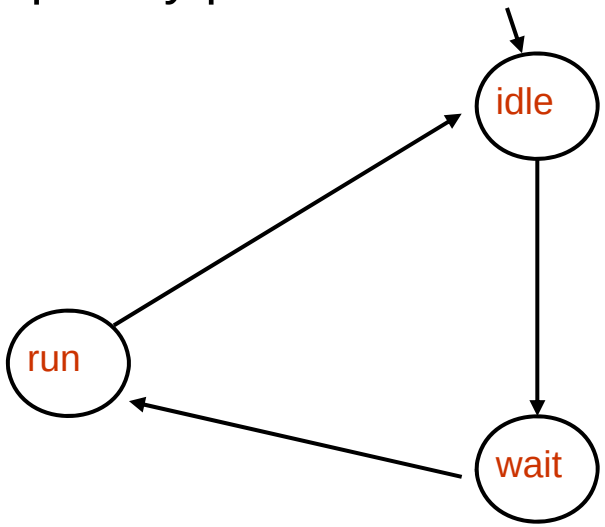
Small Example: Mars Pathfinder 1997

Typical properties of synchronization in real-time systems

- Mutual exclution
  - A process cannot access the data-bus unless it owns a mutex-lock
- Scheduling priority
  - Saving data to memory has higher priority than processing data
  - Low priority process cannot execute when high priority process is ready to execute or executes

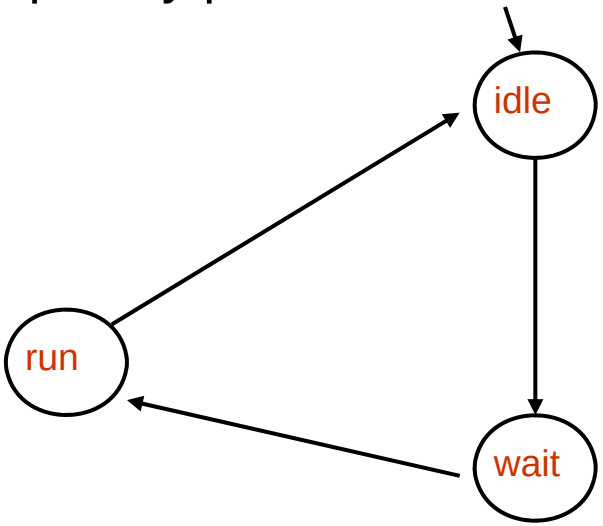
Idealized model of processes

High priority process

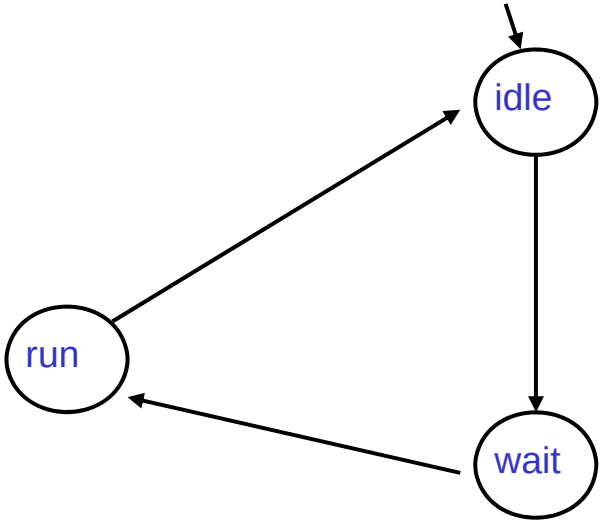


Idealized model of processes

High priority process



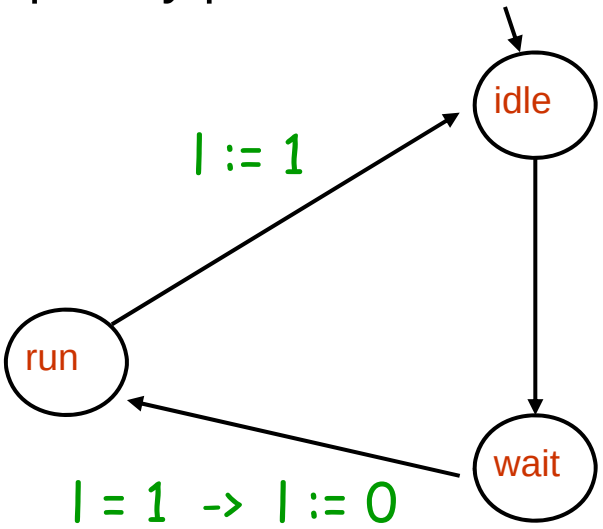
Low priority process



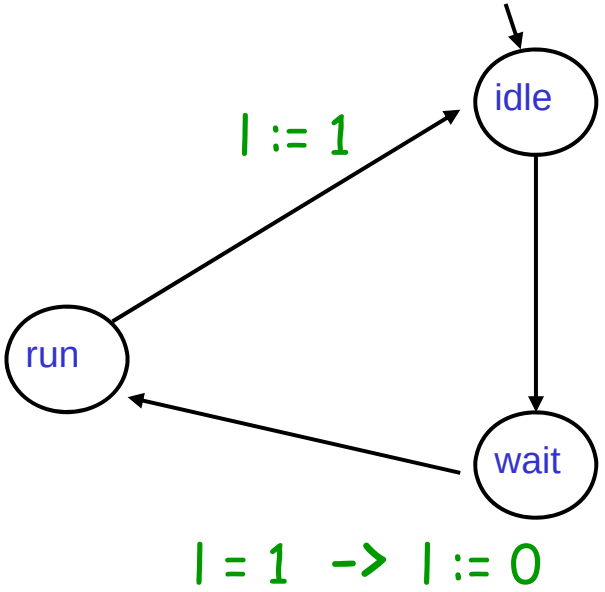
Idealized model of processes

Variables: l: integer
Initially l = 1

High priority process



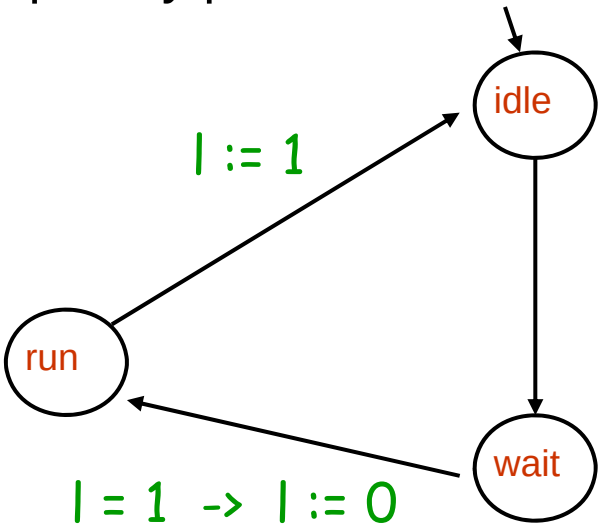
Low priority process



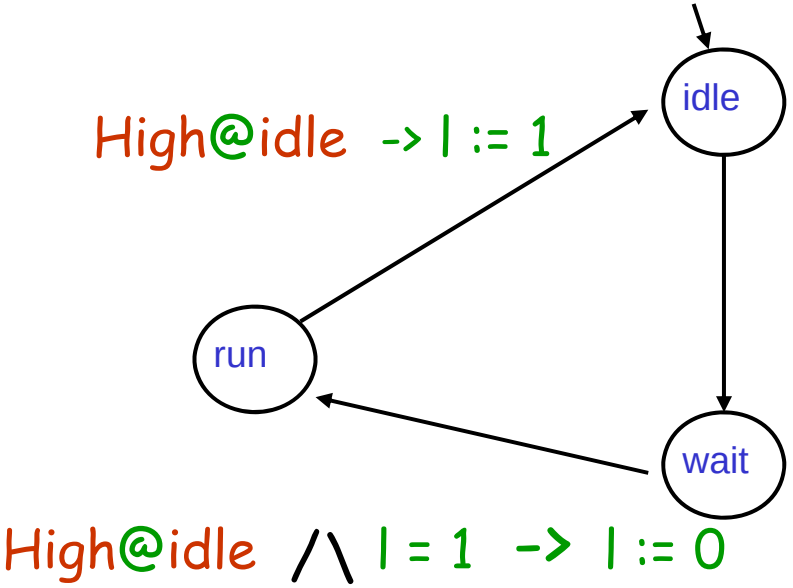
Idealized model of processes

Variables: l: integer
Initially l = 1

High priority process



Low priority process

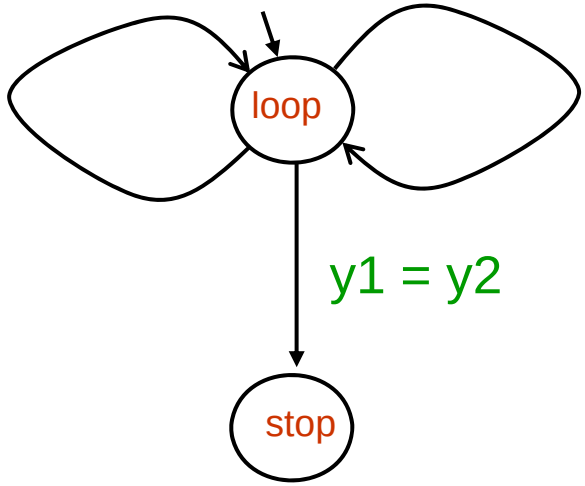


Example: GCD Computation

Action System
Variables: y1,y2: integer
Initially y1=x1, y2=x2

y1 > y2 ->
y1 := y1-y2

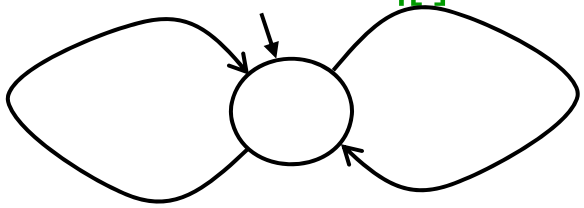
y1 < y2 ->
y2 := y2-y1



Example: Termination Detection

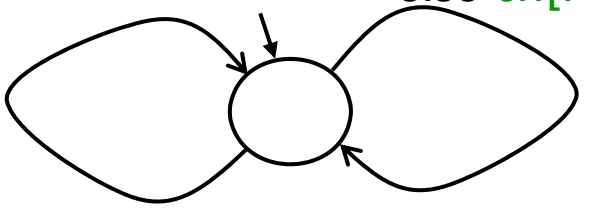
Variables: ch[N]: FIFO Channel of {black,white}
q[N]: boolean
dist = false: boolean
Processes: Q[0] || ... || Q[N-1] || P[0] || ... || P[N-1]

Q[i]:



q[i-1] -> q[i] := false ;
if i=0 then dist := true

P[i]:



ch[i]?black -> ch[i+1]!black

P[0]: