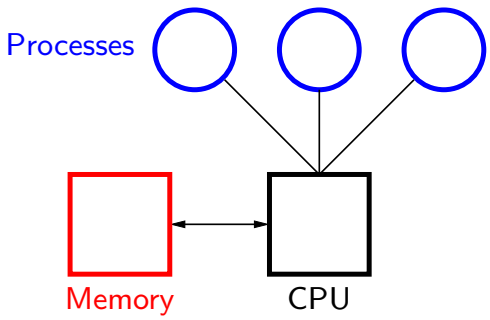
Course 02158

Transition Systems

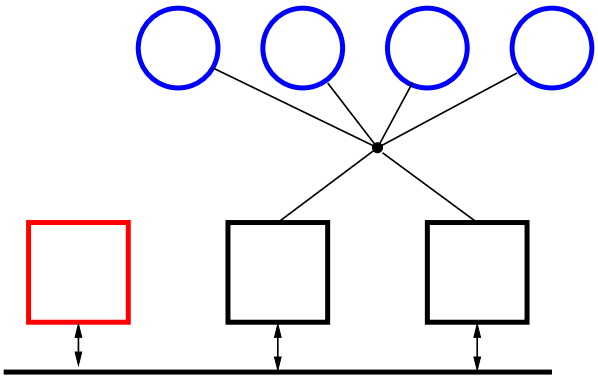
Hans Henrik Løvengreen

DTU Compute

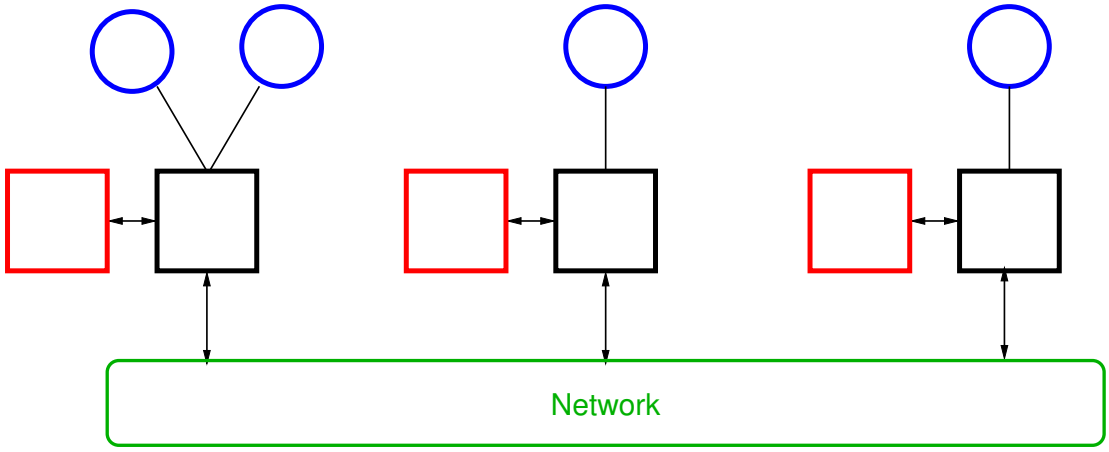
Implementation — Mono-processor



Implementation — Multi-processor

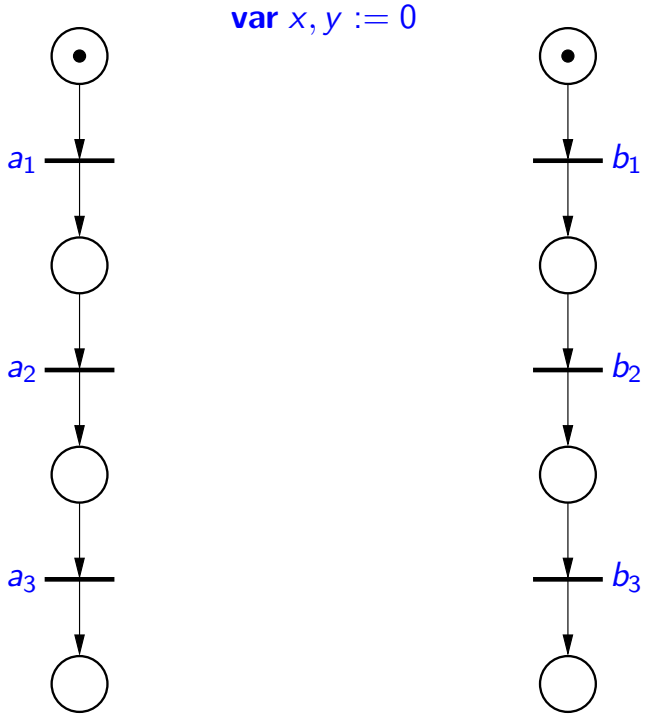


Implementation — Distributed system



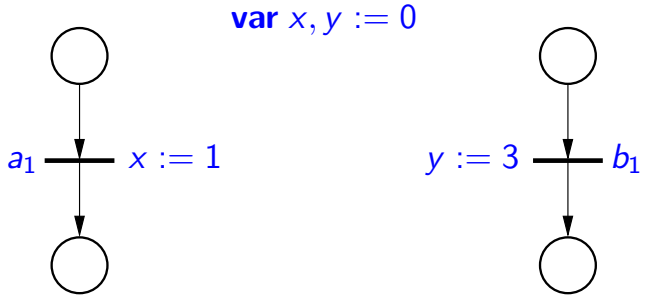
Shared Variables

var $x, y := 0$

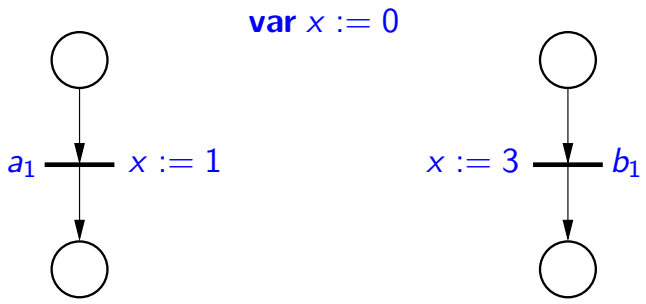


Shared Variables

var $x, y := 0$



Shared Variables



Atomic Actions

Idea

- *Atomic* = virtually indivisible

Definition

- Two actions, a and b, are *mutually atomic* iff

  $a \parallel b$ has the same effect as $a; b$ or $b; a$

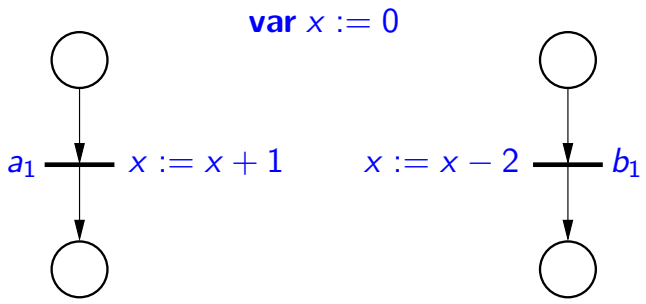
- A program has *atomic actions* if they are mutually atomic
- Assuming a to be atomic is denoted by $\langle a \rangle$

Interleaving Model

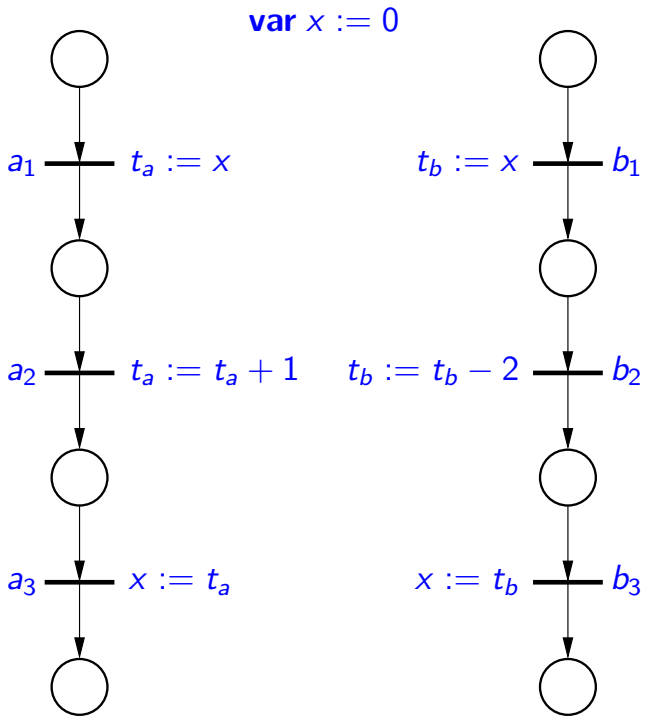
- Assume that all actions a program are atomic:

  Any (parallel) execution of the program corresponds to some sequential interleaving of the atomic actions

Shared Variables



Shared Variables



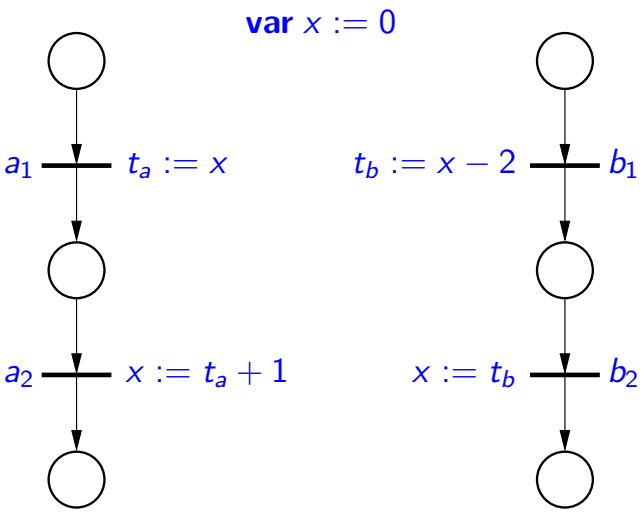
Critical References

- A *simple variable* is held in a machine word
- Access to simple variables is assumed atomic on standard HW
- A *critical reference* is either:
  - Reading a simple variable written by another process
  - Writing a simple variable accessed by another process
- Access to *non-simple variables* counts for more critical references

Rule of Critical References

- S contains at most one critical reference $\Rightarrow$ S is atomic

Shared Variables



Transition Systems

- General mathematical model of discrete behaviour

Definitions

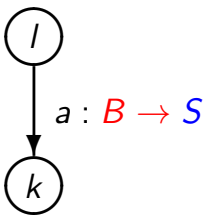
- A *(labelled) transition system* TS is a tuple $(\Sigma, \mathcal{A}, \mathcal{T}, s_0)$, where:
  - Σ is a set of *states*
  - $\mathcal{A}$ is a set of *actions* (or labels)
  - $\mathcal{T}$ $\subseteq \Sigma \times \mathcal{A} \times \Sigma$ is the *transition relation*
  - s_0 $\in \Sigma$ is the *initial state*
- $(s, a, s') \in \mathcal{T}$:
  Action a can be executed in state s resulting in a new state s'
- For a given TS, this fact is usually written $s \xrightarrow{a} s'$
- An *execution* of TS is a finite or infinite sequence

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$$

where s_0 is the initial state and $(s_i, a_i, s_{i+1}) \in \mathcal{T}$ for every i.

Transition Diagrams

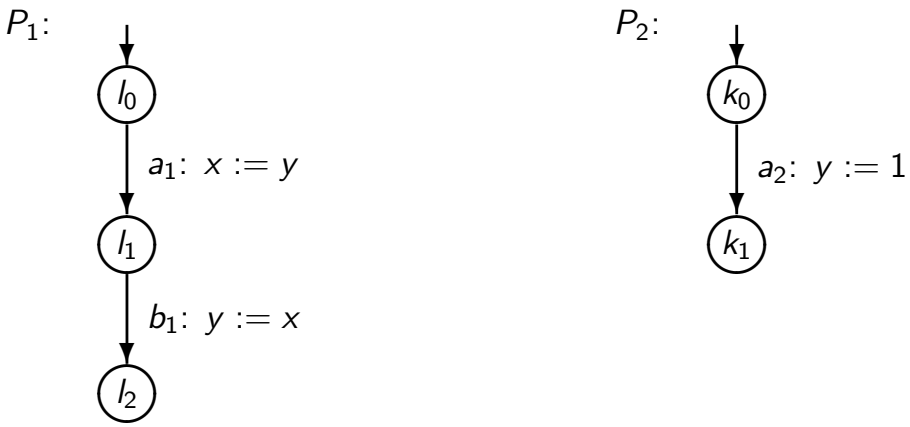
- AKA *program graphs*
- Each process is described by a graph with:
  - Nodes representing *control locations* (with one being initial)
  - Edges representing *(conditional/guarded) action execution*



- Branches are represented by using mutally exclusive guards
- Blocking can be represented by single conditional action edges
- All edges assumed to represent *atomic actions*
- A set of concurrent transition diagrams generates a transition system

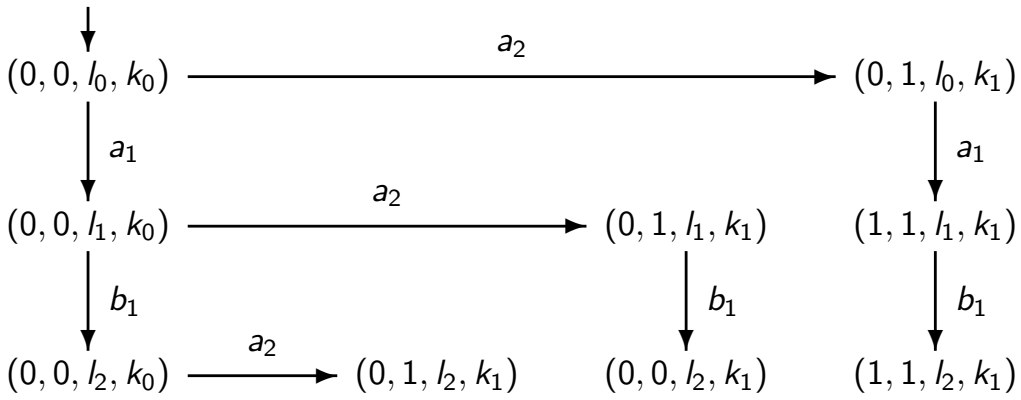
Transition Diagrams — Example

- **var** x, y : *integer* := 0;



Transition Graph

- Transition graph = transition relation of program transition system
- States: (x, y, π_1, π_2),
- Initial state: $(0, 0, l_0, k_0)$



Textual Process Definitions

- Declaring top-level processes with the **process** keyword:

  var x, y : *integer*;
  $x := 1;\quad y := 2;$

  process $p_1\{x := y + 1\}$

  process $p_1\{y := x - 1\}$

- Using **co** $S_1 \parallel S_2 \parallel \ldots \parallel S_n$ **oc**

  var x : *integer* $:= 1,\quad y$: *integer* $:= 2;$

  co $x := y + 1 \parallel y := x - 1$ **oc**

- Generalized forms:

  process $p[i$ **in** $1..n]\{\ldots; sum := sum + a[i]; \ldots\}$

  co $[i$ **in** $1..n]\{\ldots; a[i] := 0; \ldots\}$