Lecture 4: Checking properties in NuSMV

B. Srivathsan

Chennai Mathematical Institute

Model Checking and Systems Verification

January - April 2016

Outline

- **Module 1:** Synchronous Vs Asynchronous composition
- **Module 2:** More examples of NuSMV models and properties
- **Module 3:** A problem in concurrency
- **Module 4:** What is a property?

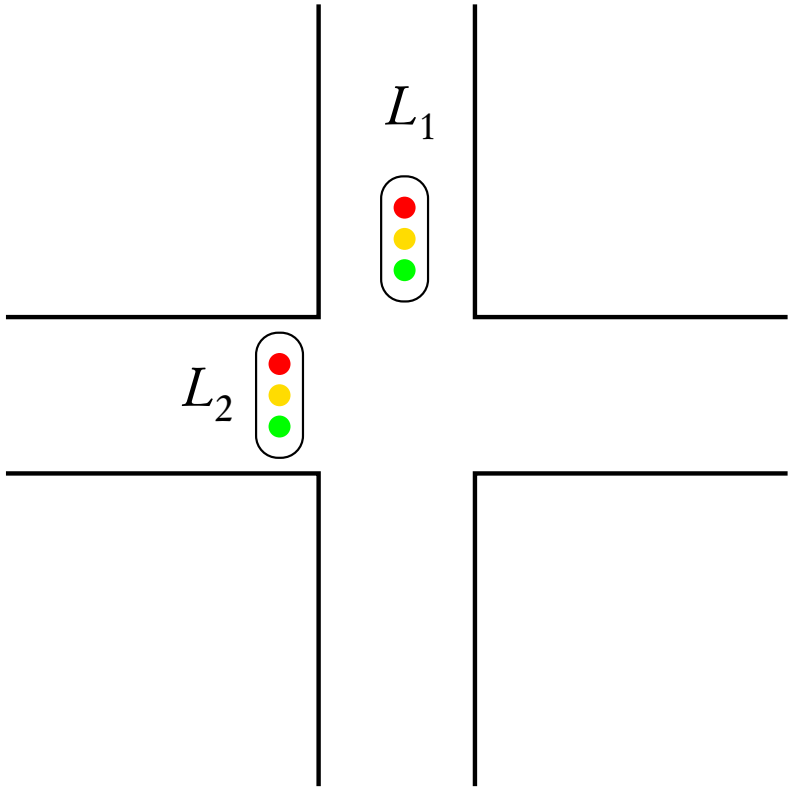
Module 1:

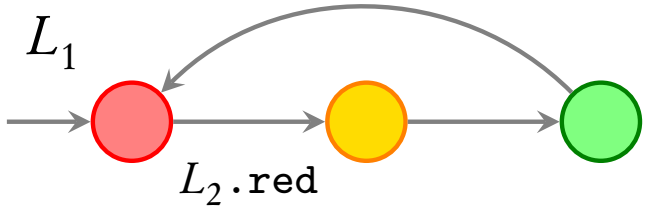
Synchronous Vs Asynchronous composition

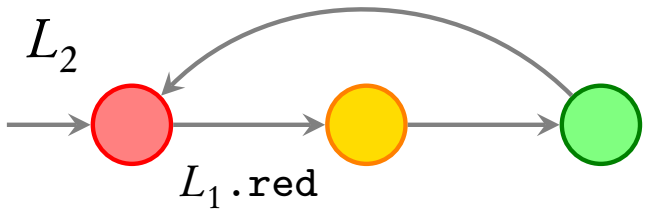
Acknowledgements:

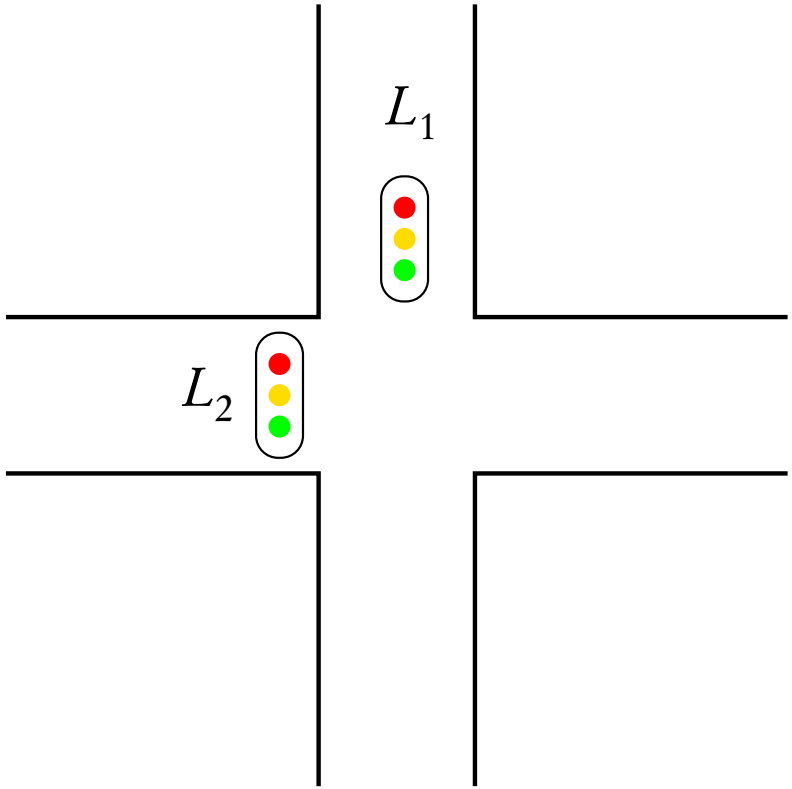
Content in this part of module taken from lecture slides of

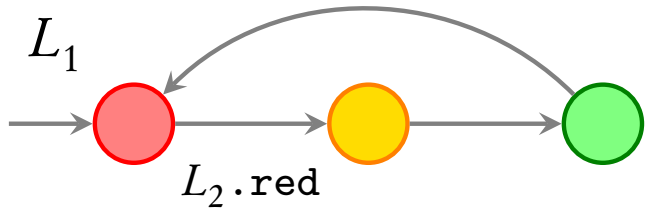
Prof. Supratik Chakraborty, IIT Bombay

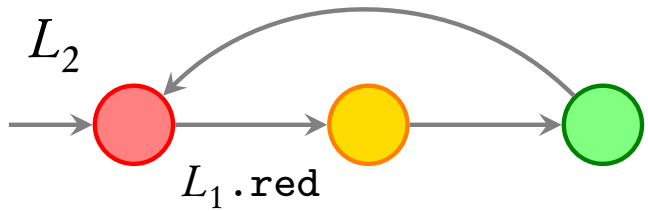

L_1
L_2


L_1
L_2.red


L_2
L_1.red







If a light is **red**, it can **stay red** for an **arbitrary period**

If it goes **yellow**, it should become **green within one cycle**

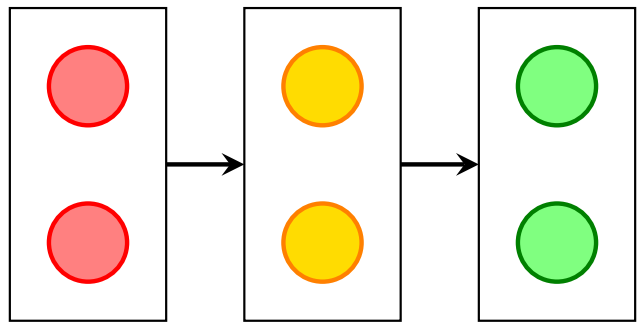
If it is **green**, it can **stay green** for an **arbitrary period**

```
MODULE light(other)
VAR
    state: {r,y,g};
ASSIGN
    init(state) := r;
    next(state) := case
                     state=r & other=r : {r, y};
                     state=y : g;
                     state=g : {g, r};
                     TRUE : state;
                   esac;

MODULE main
VAR
    tl1: light(tl2.state);  tl2: light(tl1.state);
```

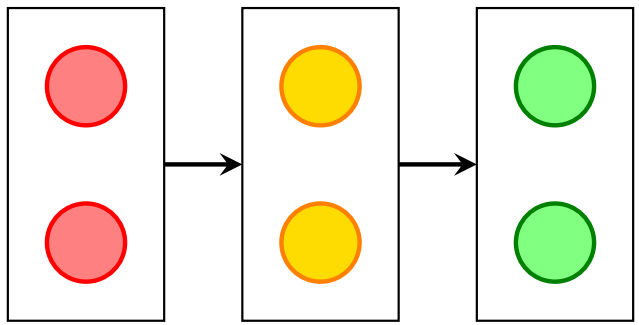
Synchronous composition

```
MODULE light(other)
VAR
    state: {r,y,g};
ASSIGN
    init(state) := r;
    next(state) := case
                     state=r & other=r : {r, y};
                     state=y : g;
                     state=g : {g, r};
                     TRUE : state;
                   esac;

MODULE main
VAR
    tl1: light(tl2.state);  tl2: light(tl1.state);
```

Synchronous composition



Synchronous composition

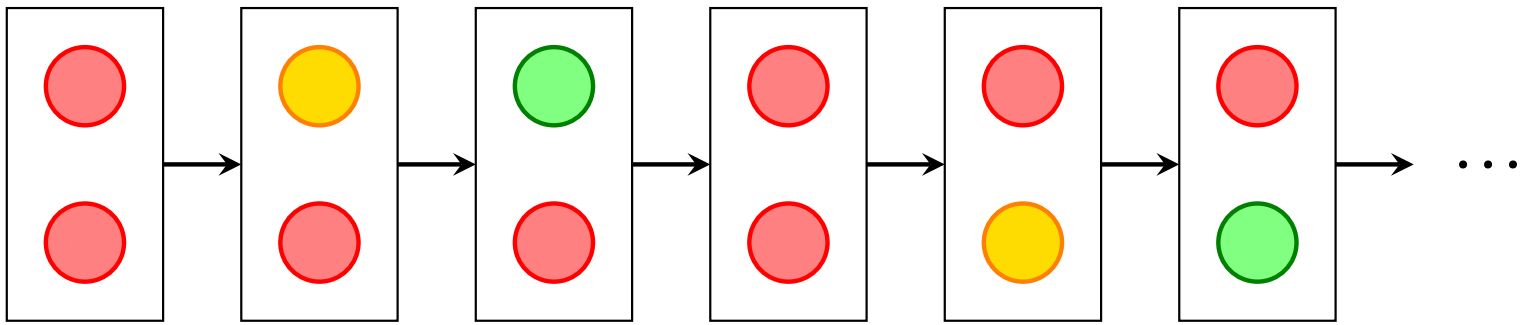


Both lights can **simultaneously become green!**

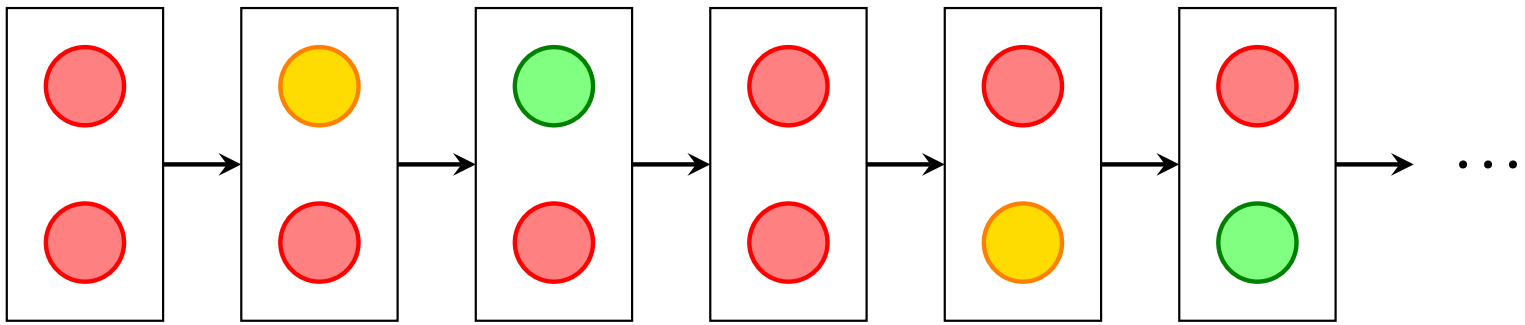
Asynchronous composition

```
MODULE light(other)
VAR
    state: {r,y,g};
ASSIGN
    init(state) := r;
    next(state) := case
                     state=r & other=r : {r, y};
                     state=y : g;
                     state=g : {g, r};
                     TRUE : state;
                   esac;

MODULE main
VAR
    tl1: process light(tl2.state);
    tl2: process light(tl1.state);
```

Asynchronous composition



Asynchronous composition



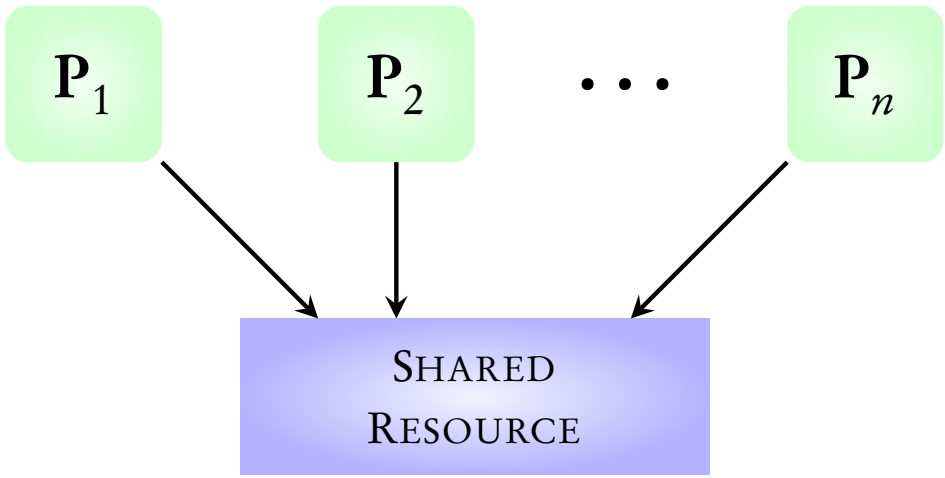
Only one light can become **green** at a time

- **Synchronous:**
  - all assignments to all modules made **simultaneously**
  - suitable when all modules are synchronized to a **global clock**
- **Asynchronous:**
  - execution of modules is **interleaved**
  - at a time, **only one** module executes
  - choice of next module to be executed is **non-deterministic**
  - suitable when **no assumptions** can be made **about communication delay** between modules

Synchronous vs. Asynchronous systems

Module 2:

More examples



(variable, printer, ...)

Mutual Exclusion: **No two processes** can access the resource **simultaneously**

P_1

loop forever

⋮ *non-critical actions*

request

critical section

release

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

request

critical section

release

⋮ *non-critical actions*

end loop

P_1

loop forever

⋮ *non-critical actions*

request

critical section

release

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

request

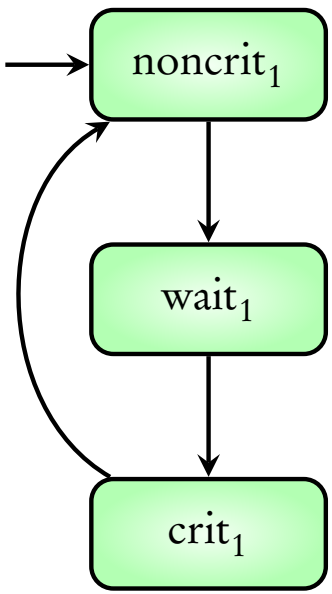
critical section

release

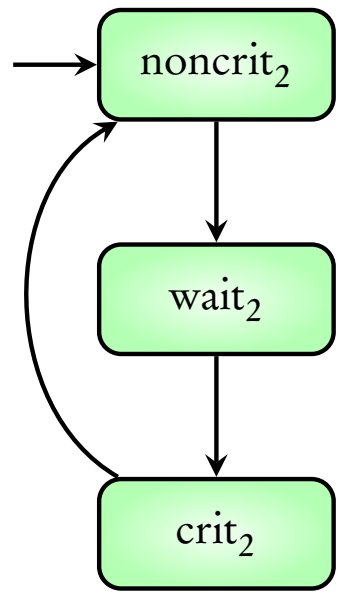
⋮ *non-critical actions*

end loop

PG_1



PG_2



P_1

loop forever

⋮ *non-critical actions*

⟨ if y>0: y:=y−1 ⟩ *request*

critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

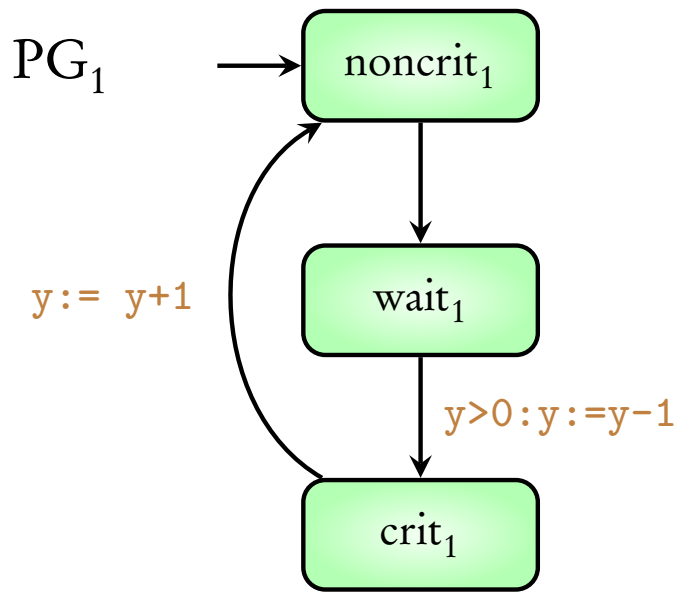
⟨ if y>0: y:=y−1 ⟩ *request*

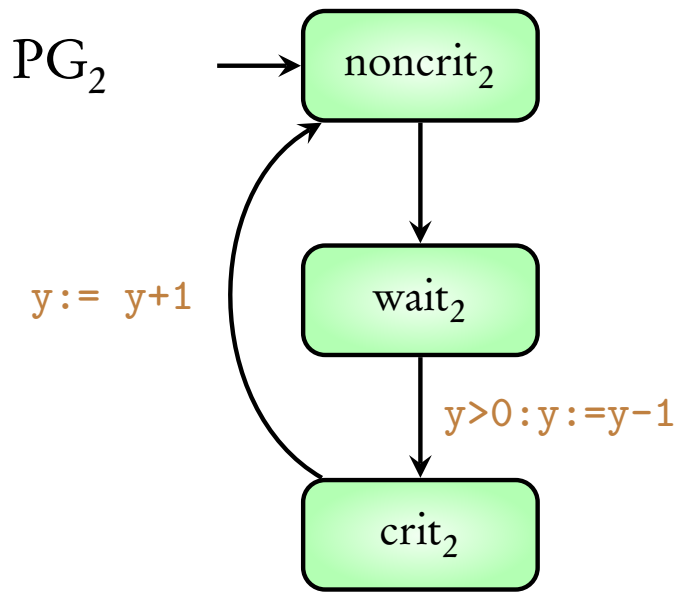
critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop





P$_1$

loop forever

⋮ *non-critical actions*

⟨ if y>0: y:=y−1 ⟩ *request*

critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop

P$_2$

loop forever

⋮ *non-critical actions*

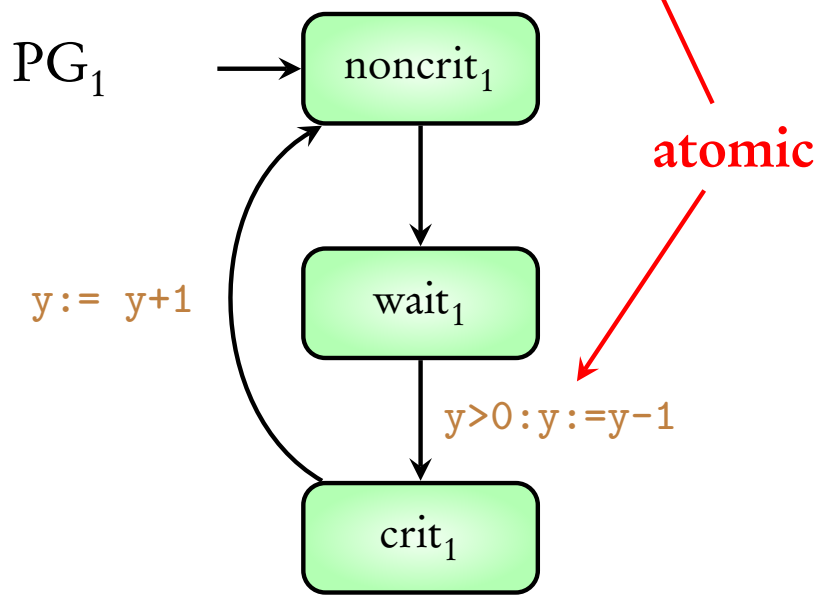
⟨ if y>0: y:=y−1 ⟩ *request*

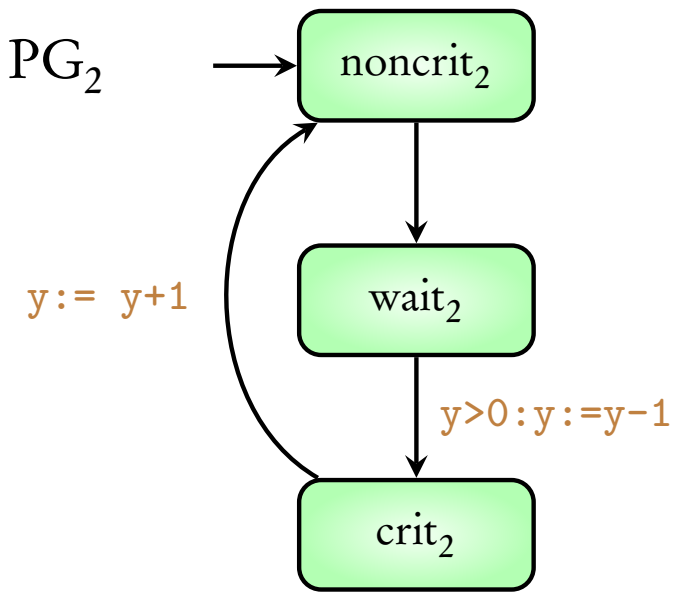
critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop





P$_1$

loop forever

⋮ *non-critical actions*

⟨ if y>0: y:=y-1 ⟩ *request*

critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop

P$_2$

loop forever

⋮ *non-critical actions*

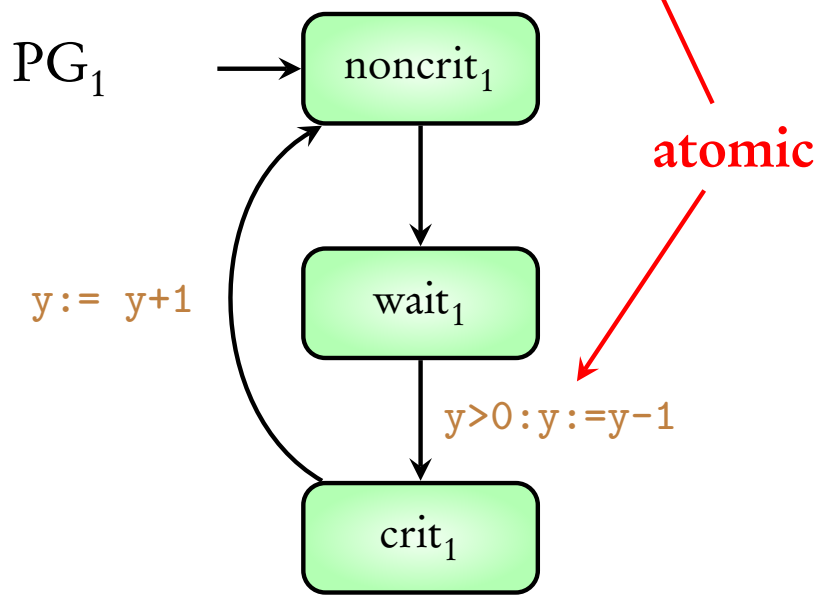
⟨ if y>0: y:=y-1 ⟩ *request*

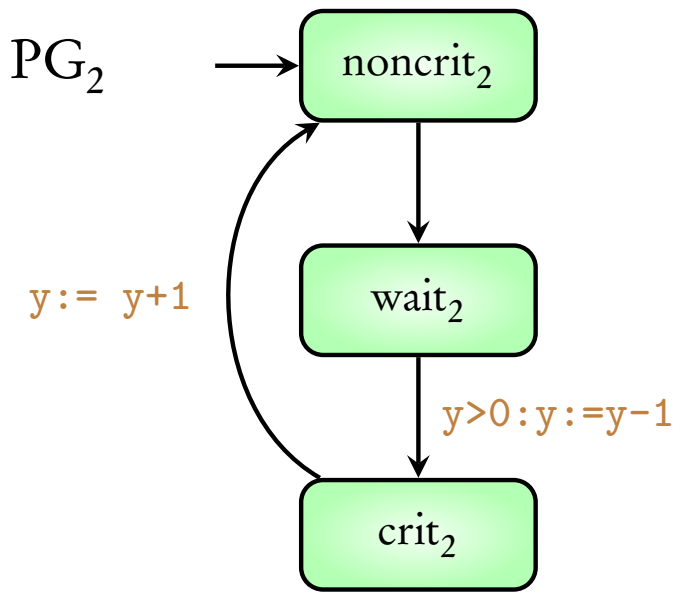
critical section

y:=y+1 *release*

⋮ *non-critical actions*

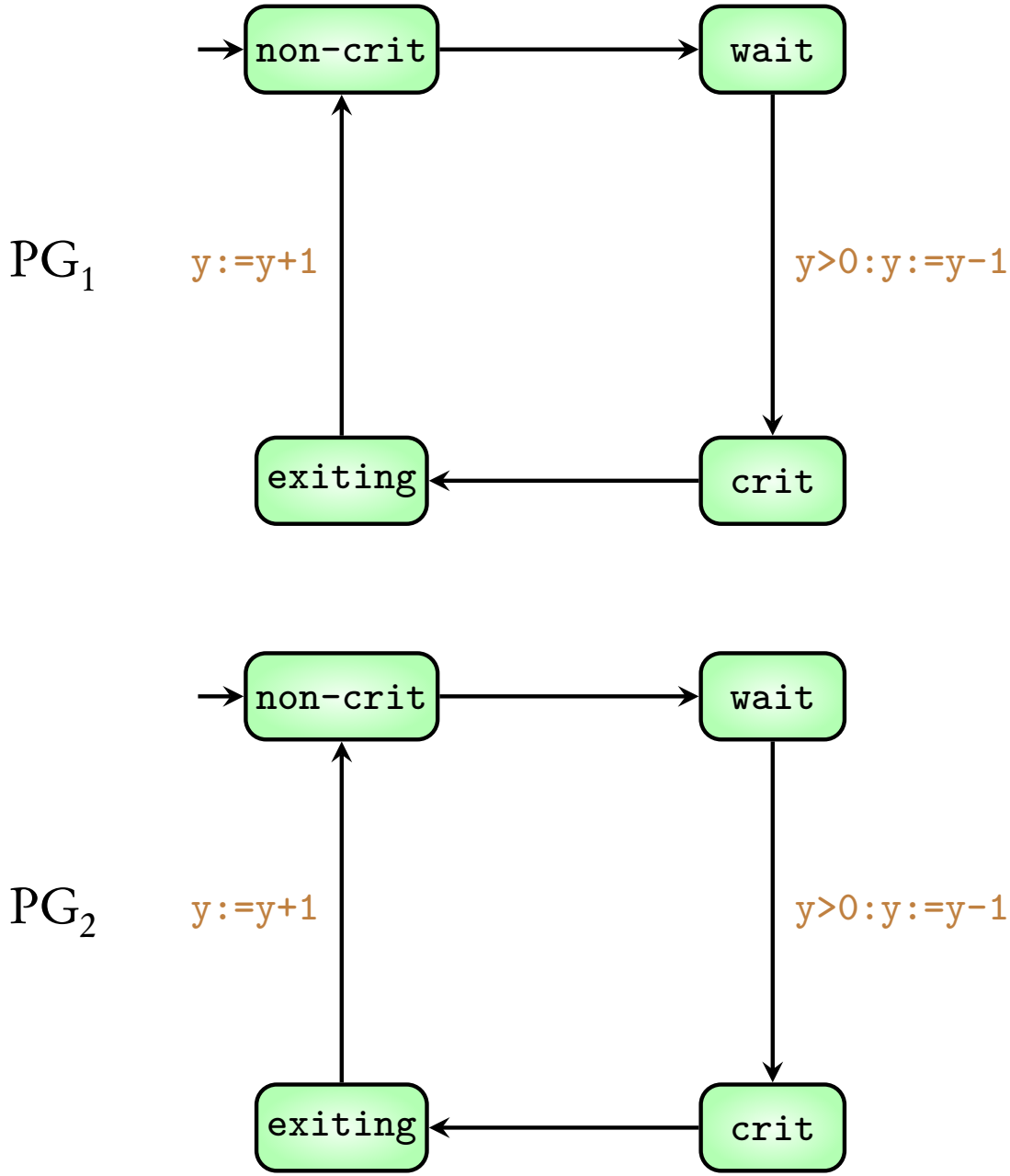
end loop

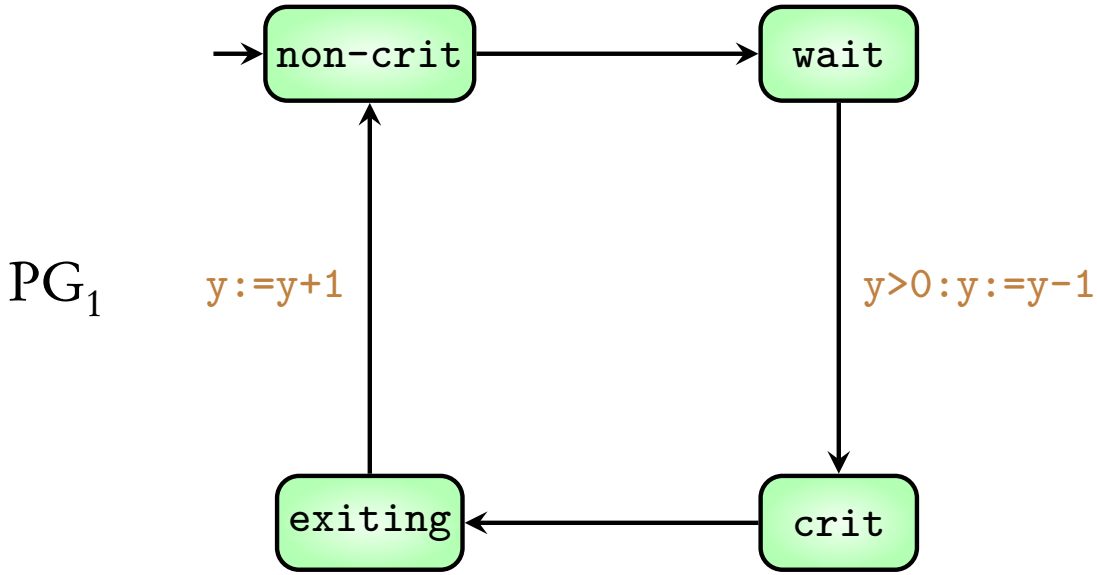


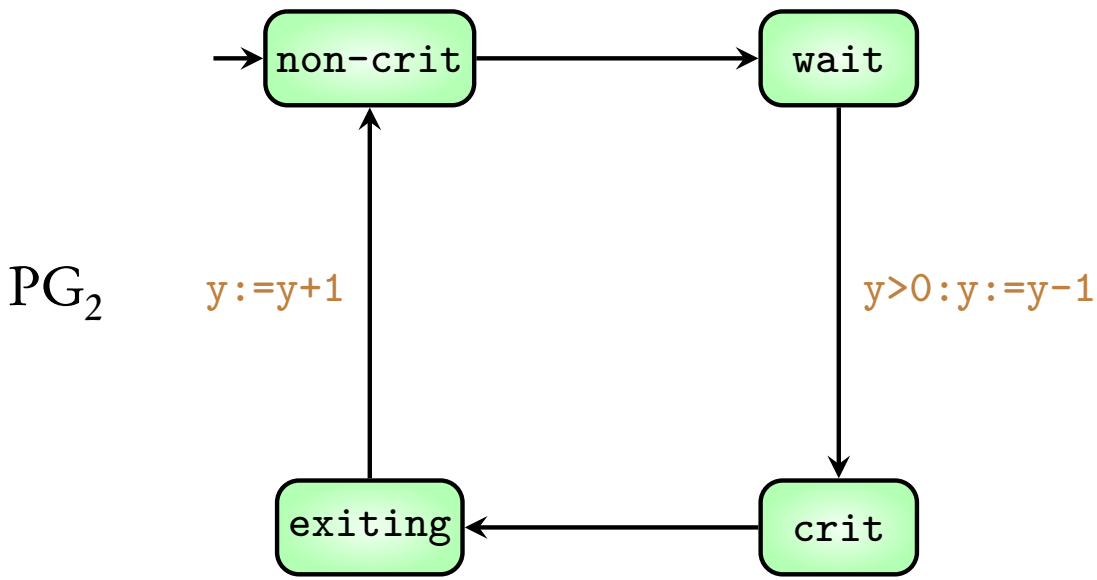


NuSMV demo: `mutex-demo.smv`

Coming next: A slight modification of previous mutual exclusion protocol


PG1
non-crit
wait
y:=y+1
y>0:y:=y-1
exiting
crit
PG2
non-crit
wait
y:=y+1
y>0:y:=y-1
exiting
crit





NuSMV demo: `mutex-demo1.smv`

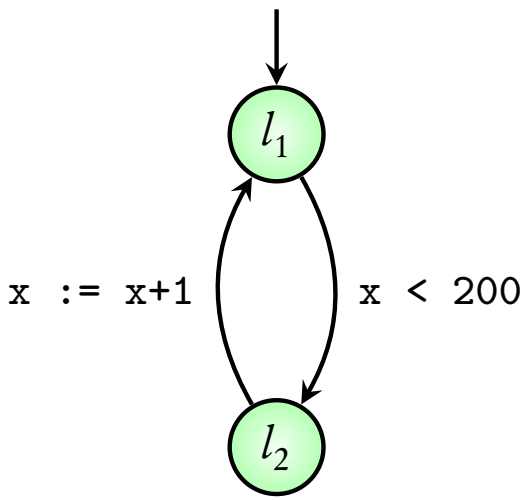
Synchronous vs. Asynchronous systems

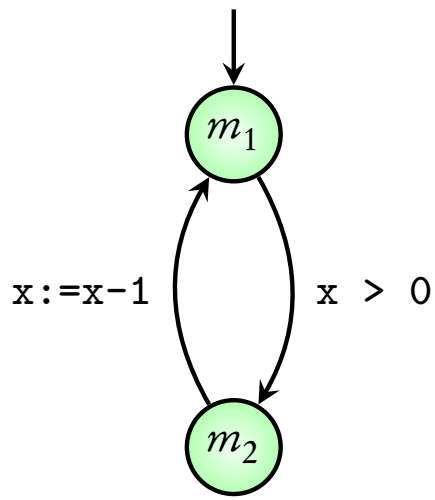
Mutual Exclusion

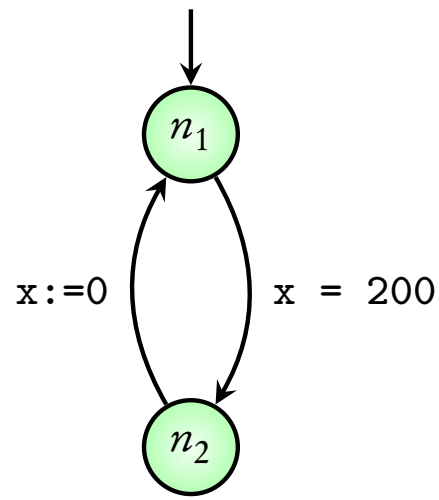
```
while x < 200
    x := x+1
```

```
while x>0
    x := x-1
```

```
while x=200
    x := 0
```



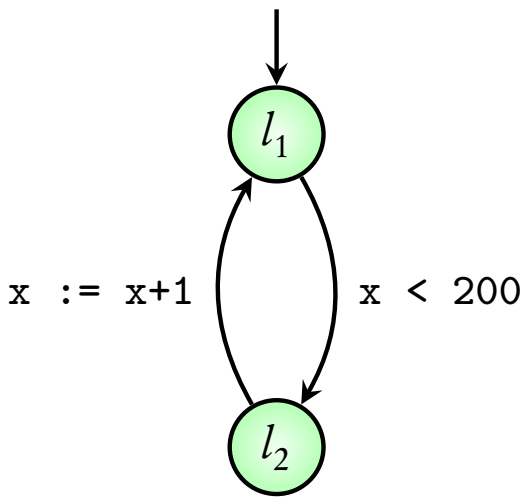


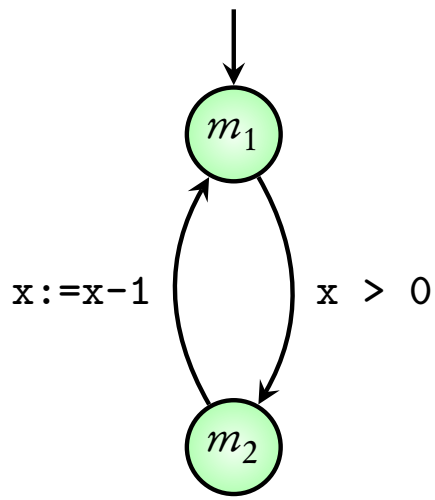


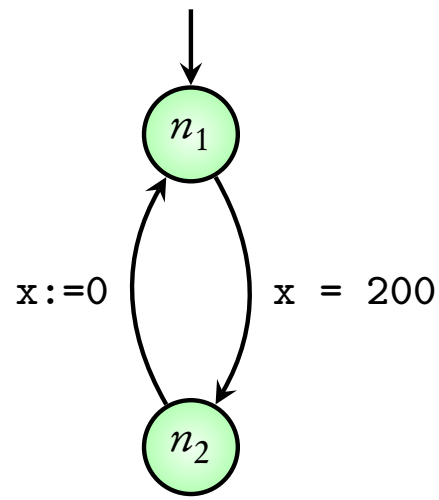
```
while x < 200
    x := x+1
```

```
while x>0
    x := x-1
```

```
while x=200
    x := 0
```







NuSMV demo: `three-program-demo.smv`

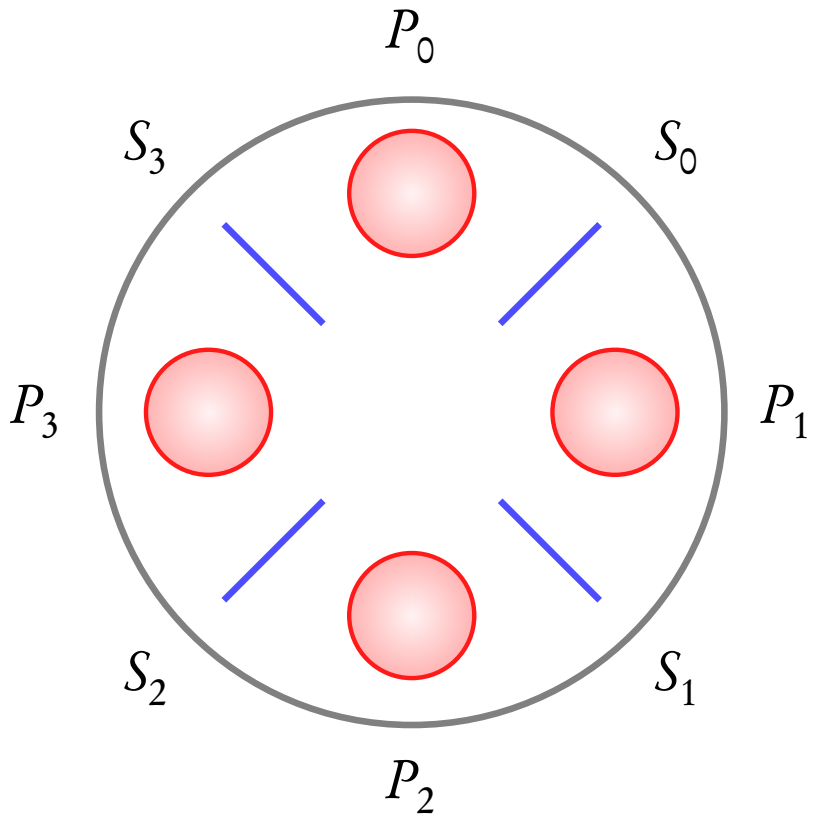
Synchronous vs. Asynchronous systems

Mutual Exclusion

Concurrent programs example

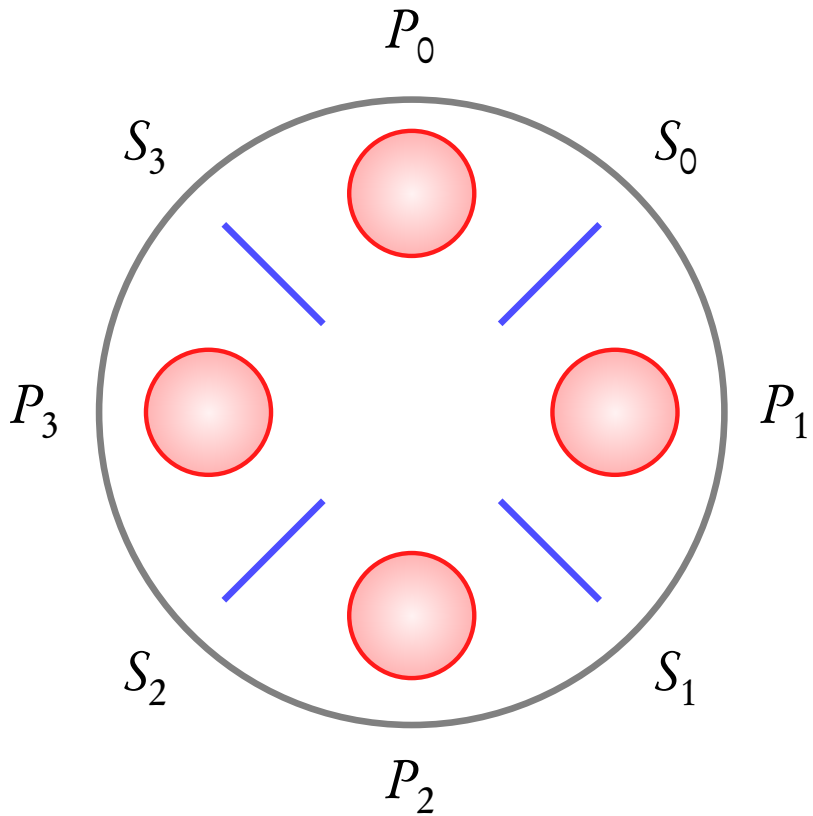
Module 3:

A problem in concurrency



$P_0 \ldots P_3$: *processes*

$S_0 \ldots S_3$: *resources*



$P_0 \ldots P_3$: *processes*

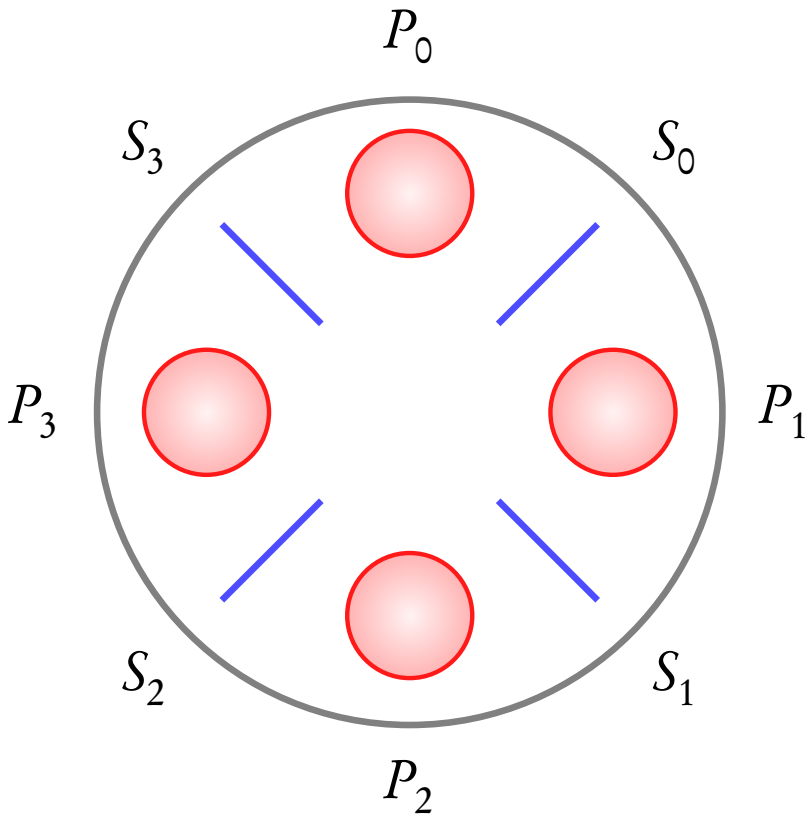
$S_0 \ldots S_3$: *resources*

Process P_i can **execute**

only if

it has access to **resources**

$S_{(i-1)}$ and S_i



$P_0 \ldots P_3$: *processes*

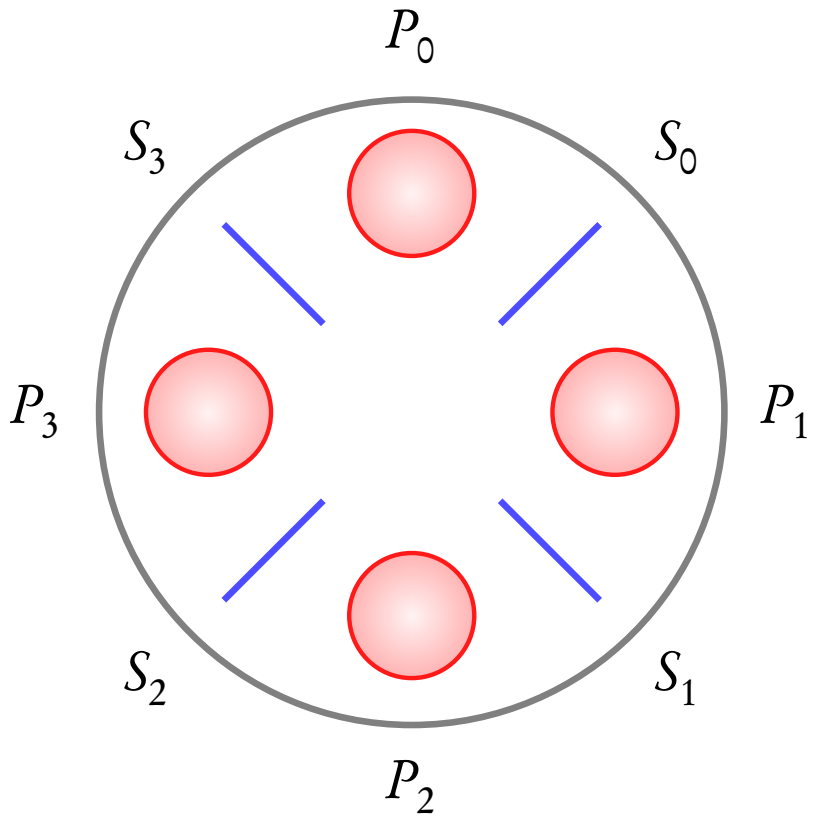
$S_0 \ldots S_3$: *resources*

Process P_i can **execute**

only if

it has access to **resources**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$



$P_0 \ldots P_3$: *processes*

$S_0 \ldots S_3$: *resources*

Process P_i can **execute**

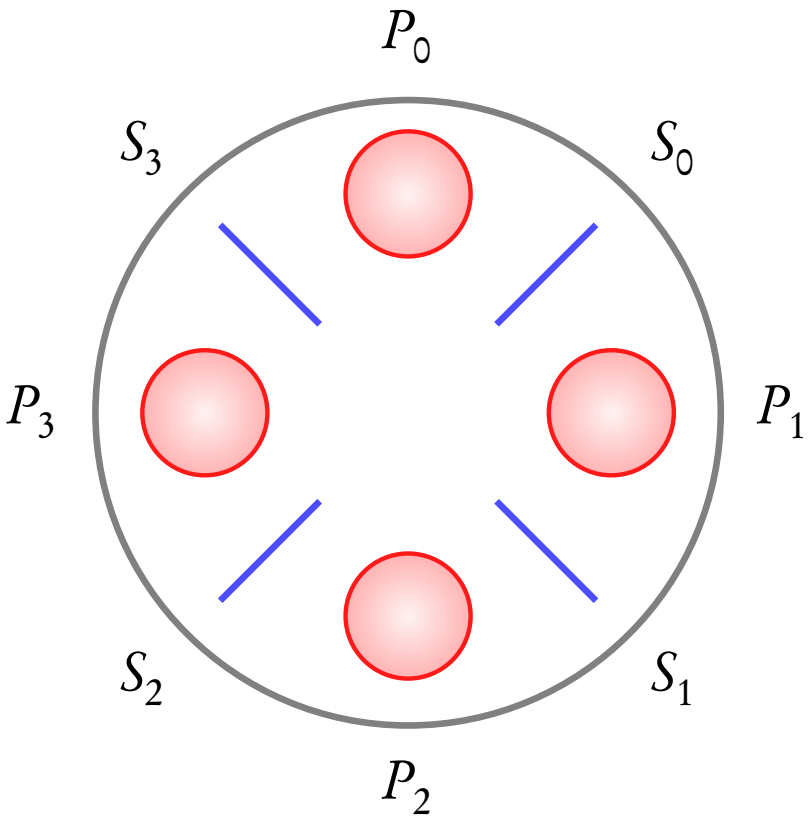
only if

it has access to **resources**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

How should the processes be **scheduled** so that **every process** can execute **infinitely often?**

Dining philosophers problem (Dijkstra)



$P_0 \ldots P_3$: *philosophers*

$S_0 \ldots S_3$: *chop-sticks*

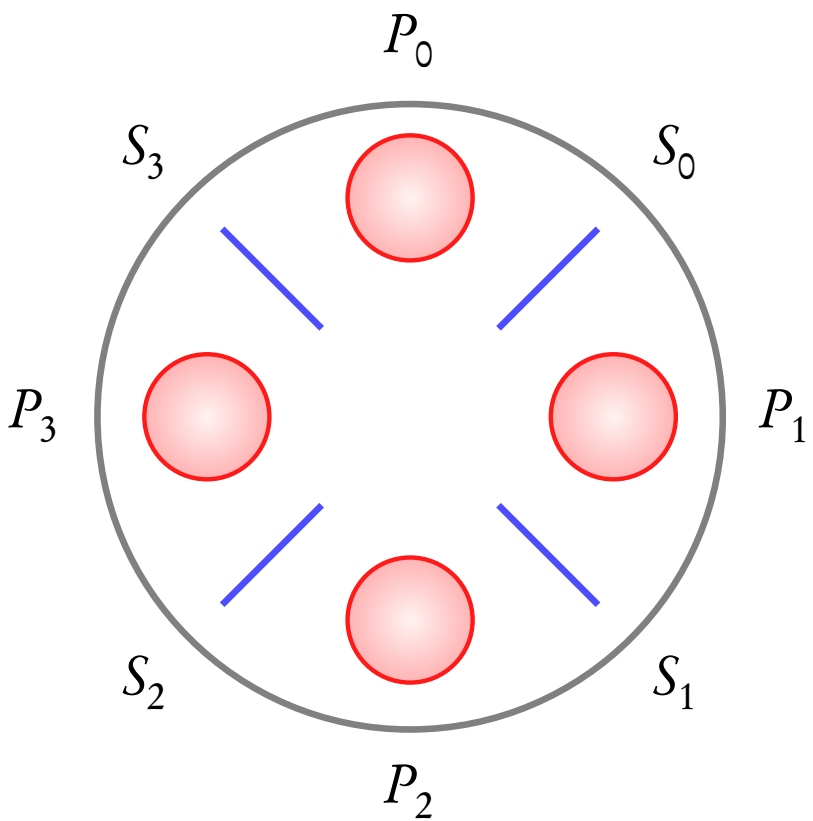
Philosopher P_i can **eat**

only if

he has access to **chop-sticks**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

Dining philosophers problem (Dijkstra)



$P_0 \ldots P_3$: *philosophers*

$S_0 \ldots S_3$: *chop-sticks*

Philosopher P_i can **eat**

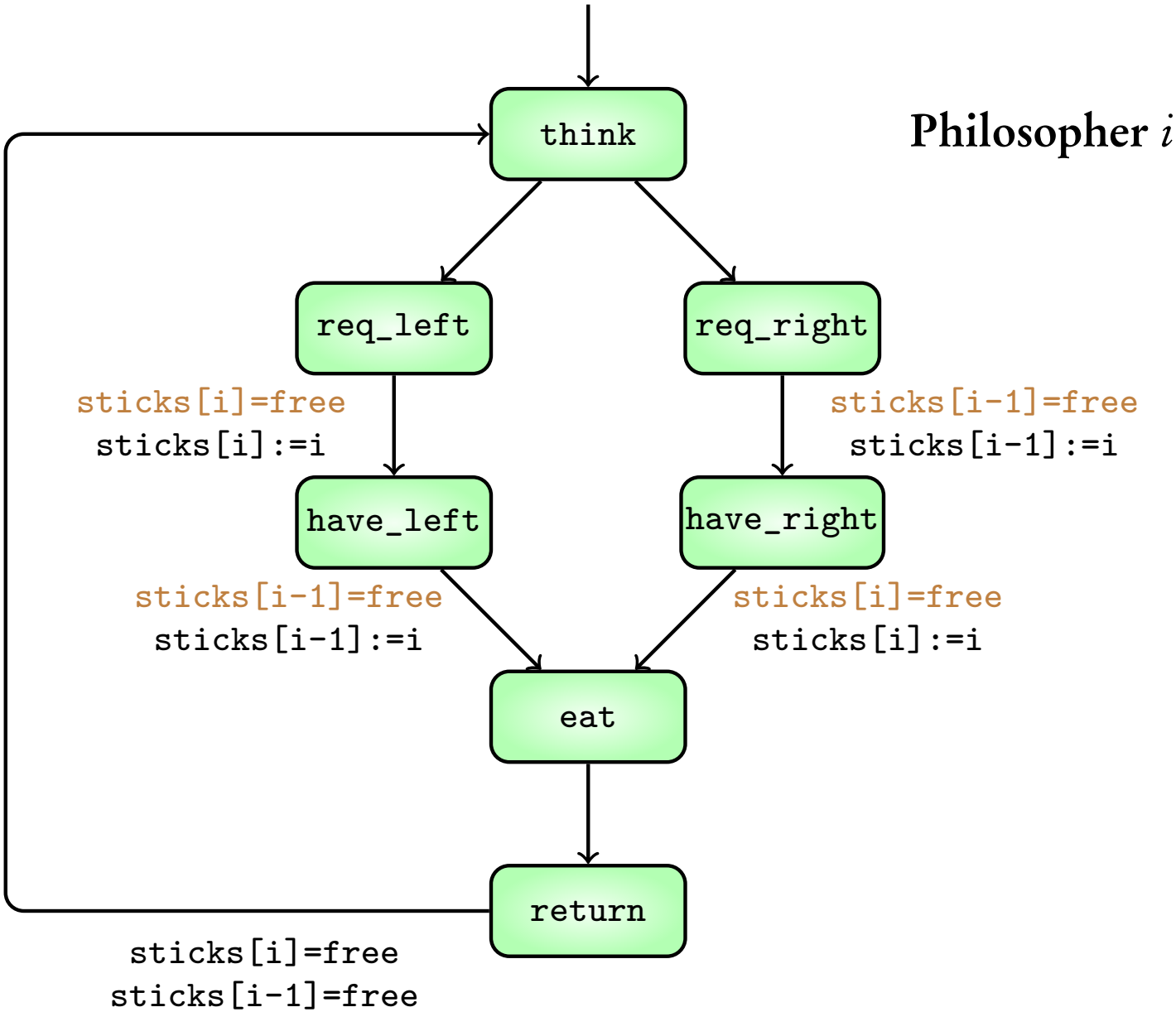
only if

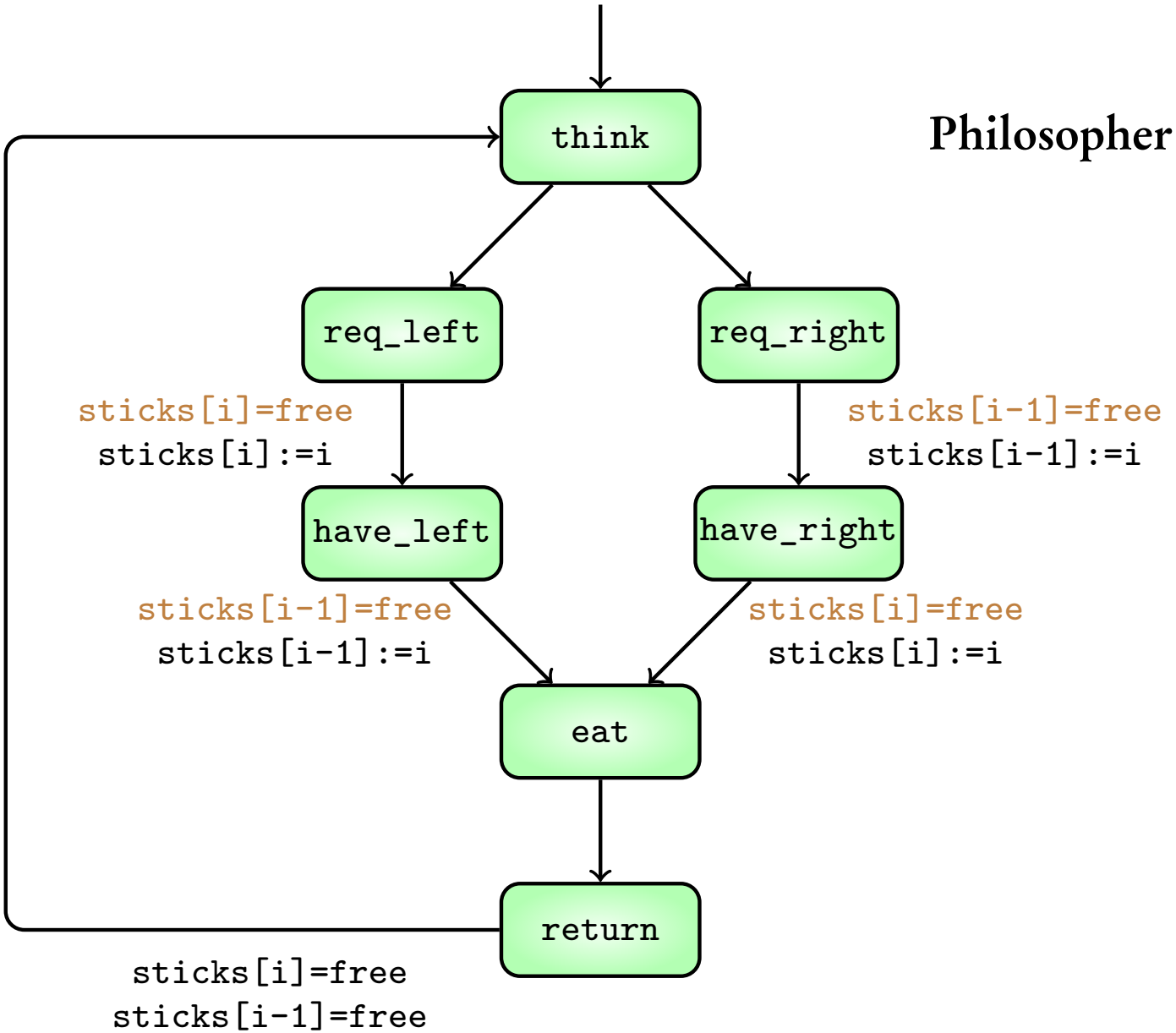
he has access to **chop-sticks**

$S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

What should the **protocol** be so that **every philosopher** can eat **infinitely often?**

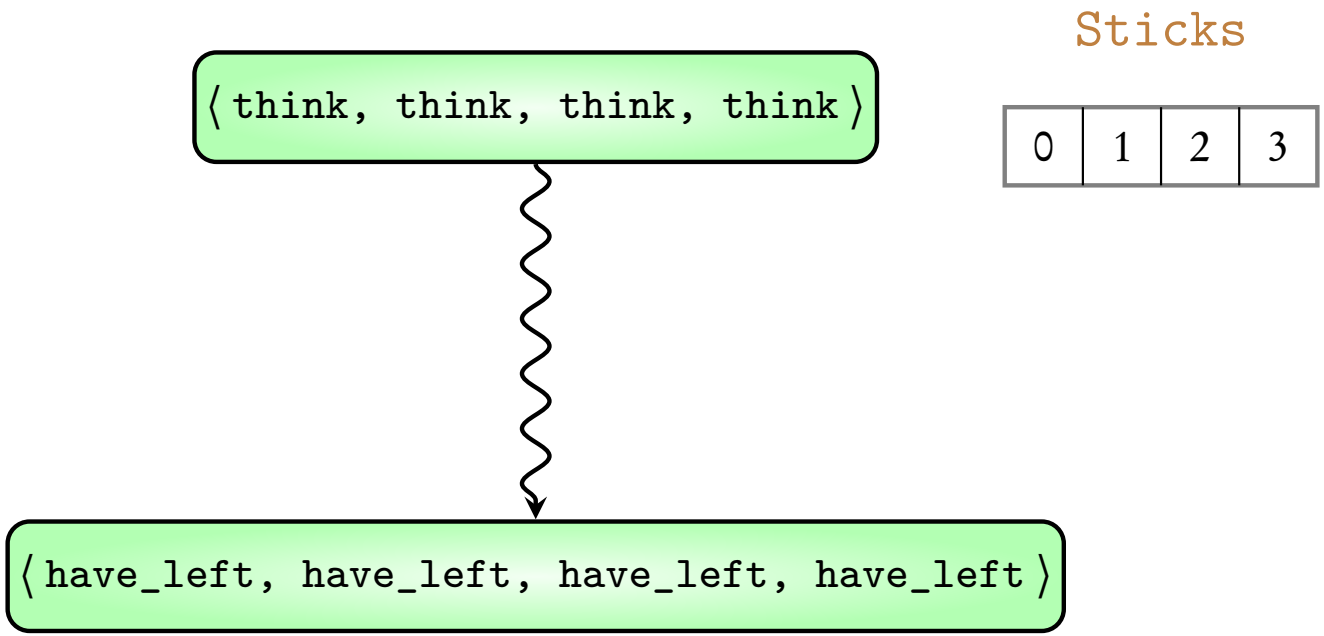
Coming next: A protocol for the dining philosophers


Philosopher i
think
req_left
req_right
sticks[i]=free
sticks[i]:=i
sticks[i-1]=free
sticks[i-1]:=i
have_left
have_right
sticks[i-1]=free
sticks[i-1]:=i
sticks[i]=free
sticks[i]:=i
eat
return
sticks[i]=free
sticks[i-1]=free


Philosopher i
think
req_left
req_right
sticks[i]=free
sticks[i]:=i
sticks[i-1]=free
sticks[i-1]:=i
have_left
have_right
sticks[i-1]=free
sticks[i-1]:=i
sticks[i]=free
sticks[i]:=i
eat
return
sticks[i]=free
sticks[i-1]=free

NuSMV demo

A deadlock


⟨ think, think, think, think ⟩
Sticks
0
1
2
3
⟨ have_left, have_left, have_left, have_left ⟩

Question: What **properties** should be checked to **detect deadlocks?**

Question: What **properties** should be checked to **detect deadlocks?**

- Next module: Attach a mathematical meaning to properties

Question: What **properties** should be checked to **detect deadlocks?**

- Next module: Attach a mathematical meaning to properties
- Next lecture: Classification of properties into various types

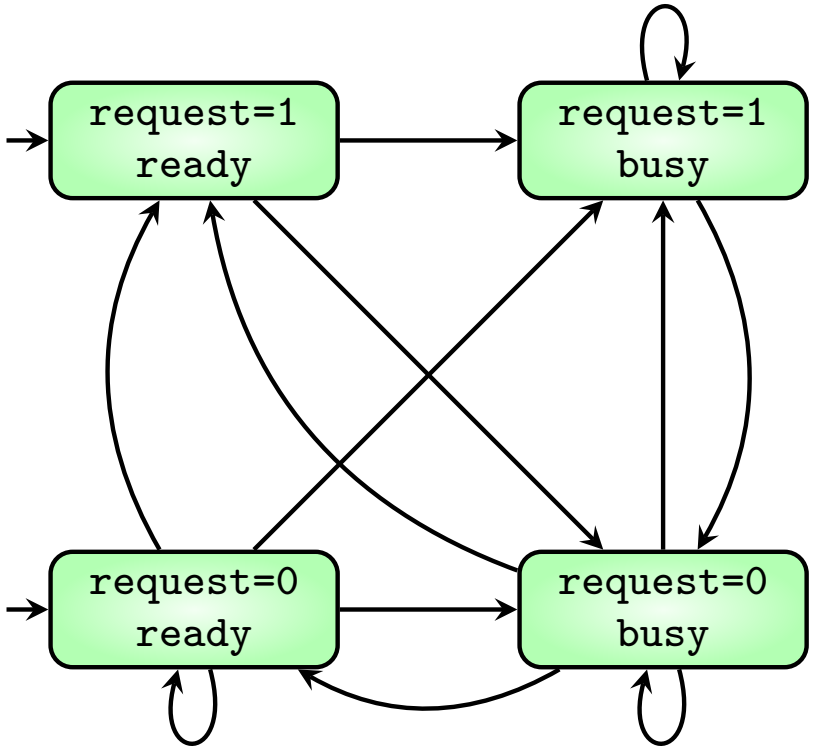
Question: What **properties** should be checked to **detect deadlocks?**

- Next module: Attach a mathematical meaning to properties
- Next lecture: Classification of properties into various types
- Next lecture: Answer to the above question

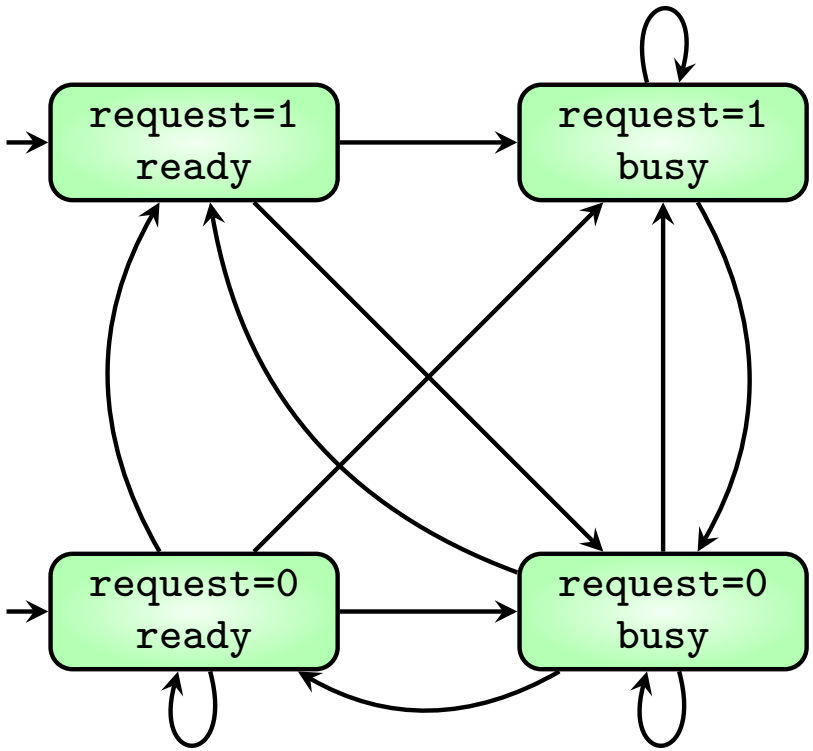
Module 4:

What is a “property”?

Goal: Attach a **mathematical meaning** to "property"

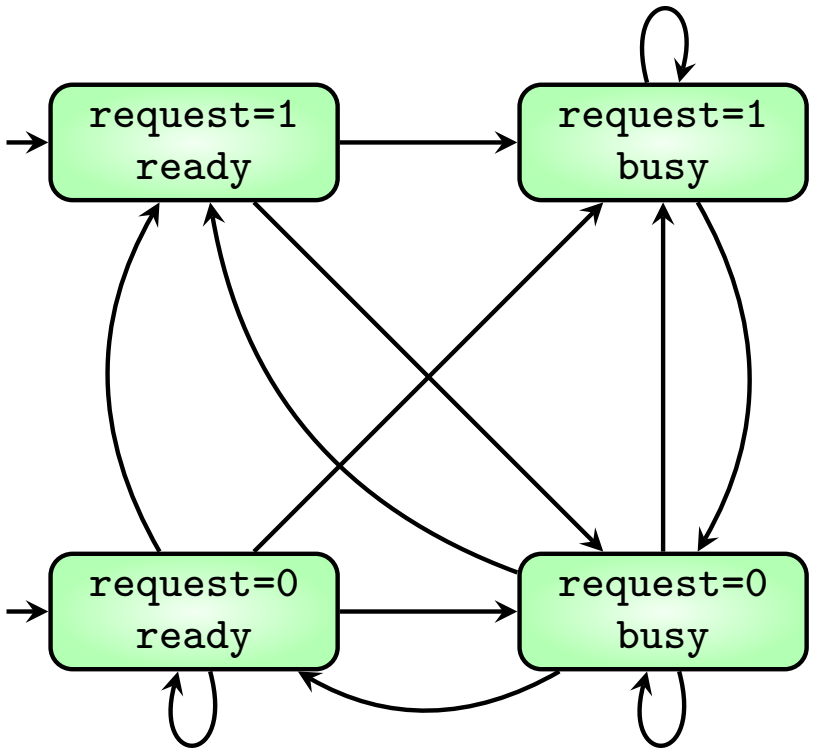



```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

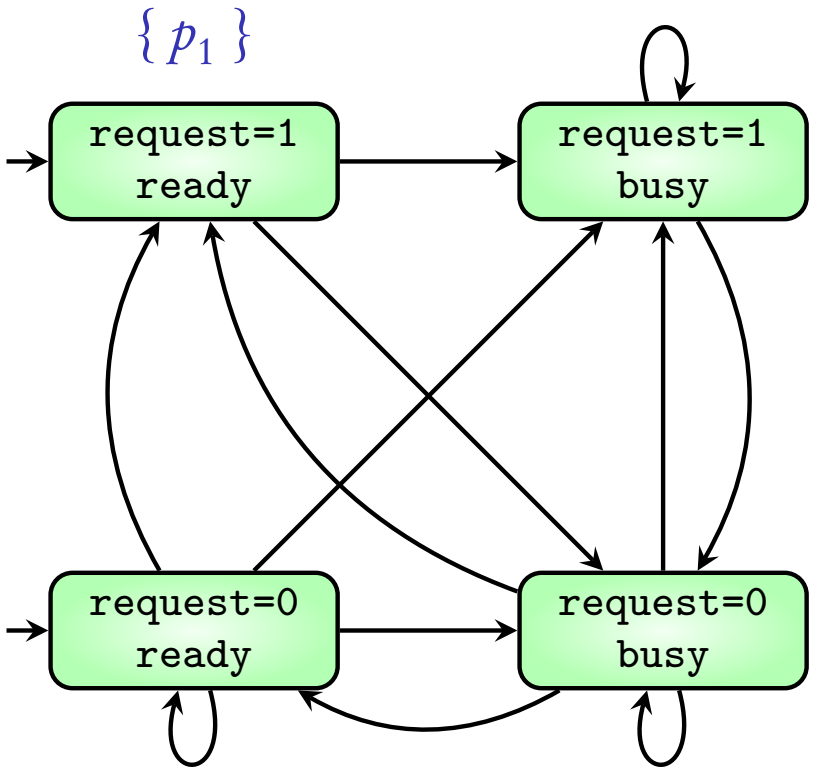
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

Atomic propositions

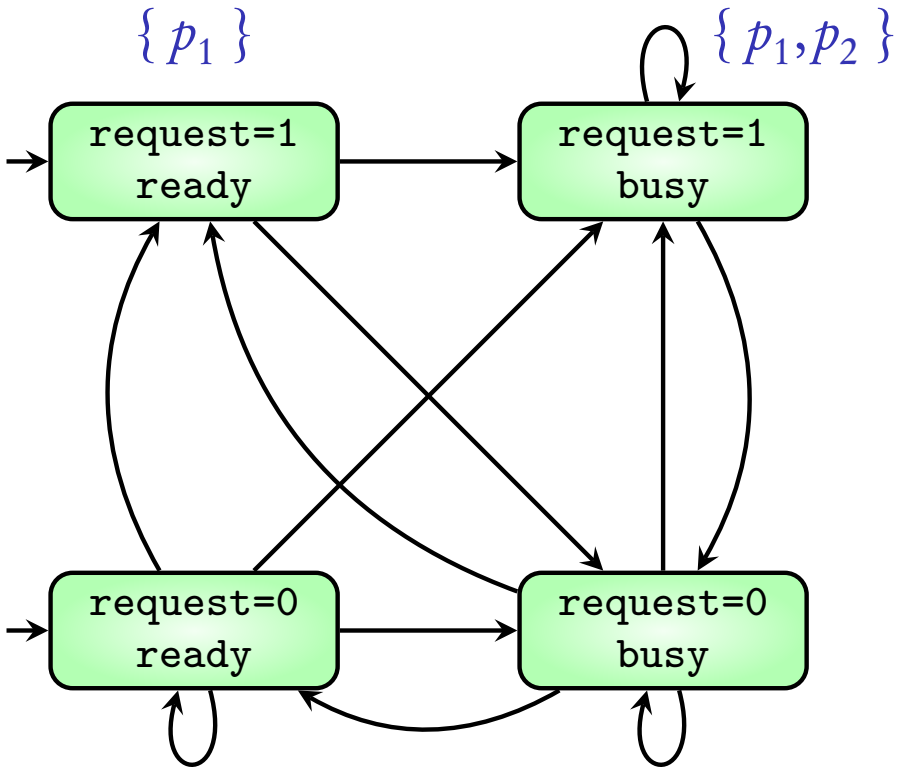
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

Atomic propositions

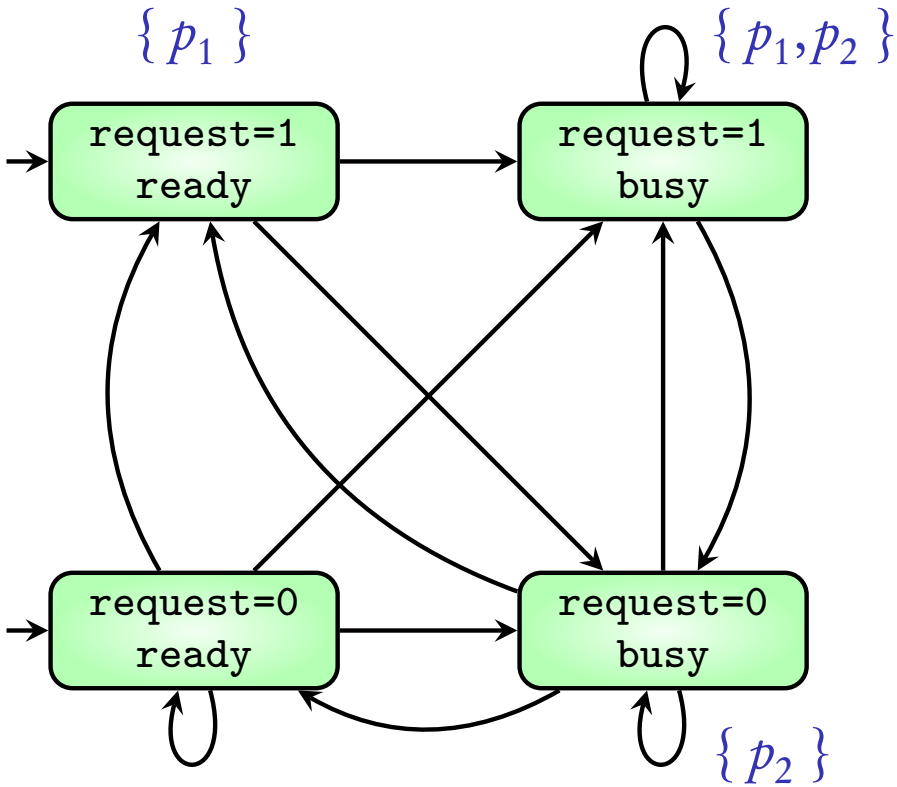
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

Atomic propositions

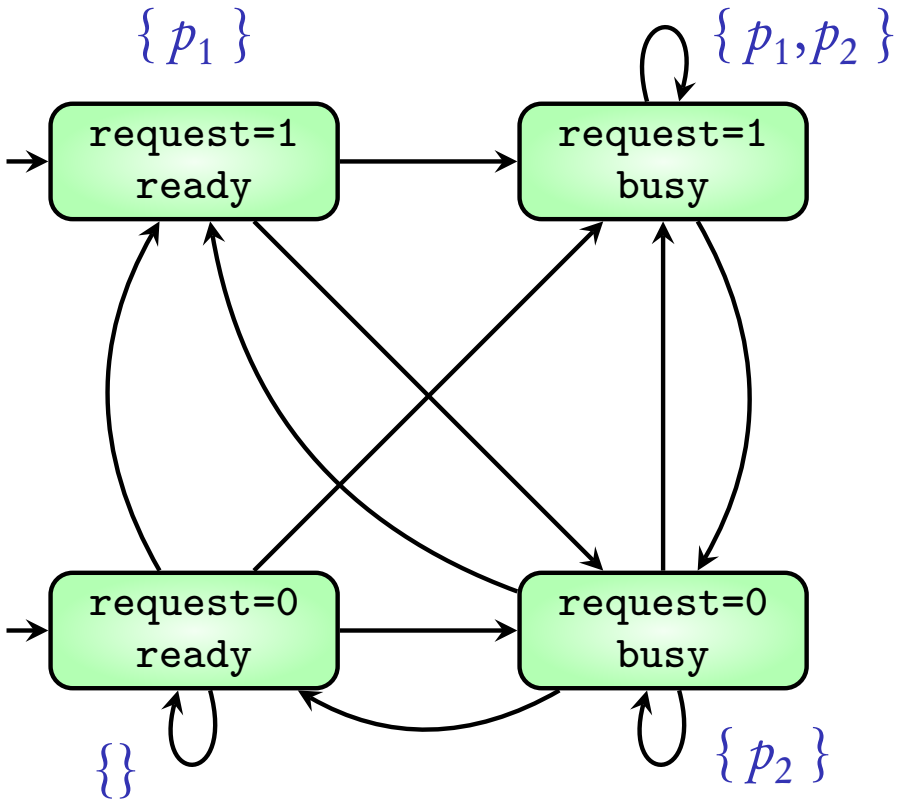
p_1: (request=1) p_2: (status=busy)




```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

Atomic propositions

p_1: (request=1) p_2: (status=busy)

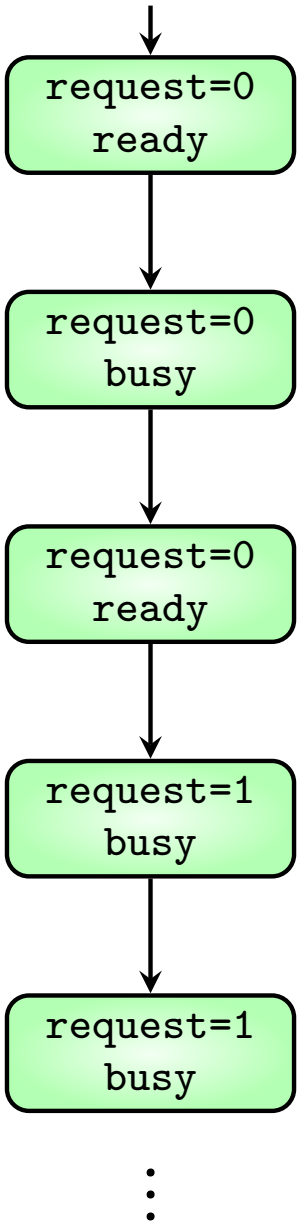



```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

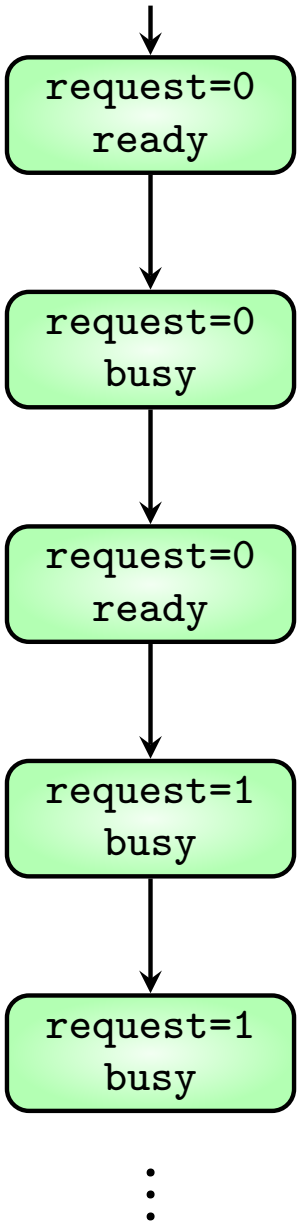
Atomic propositions

p_1: (request=1) p_2: (status=busy)

Execution

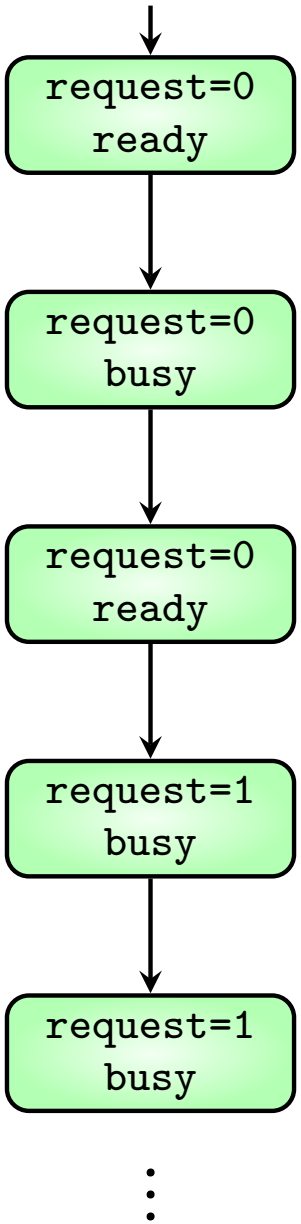

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy

Execution


request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy

{ }

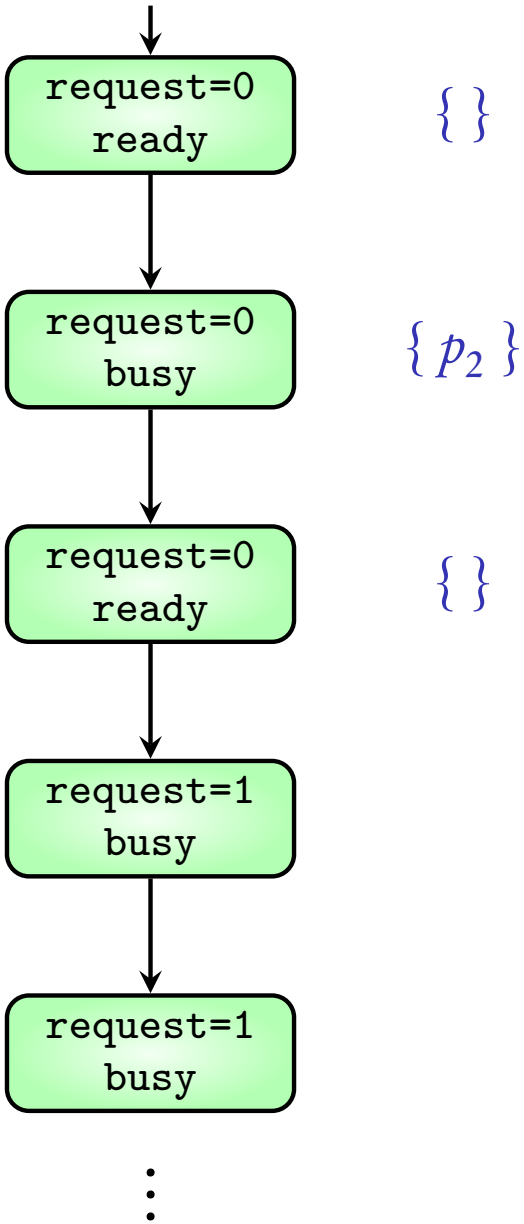
Execution



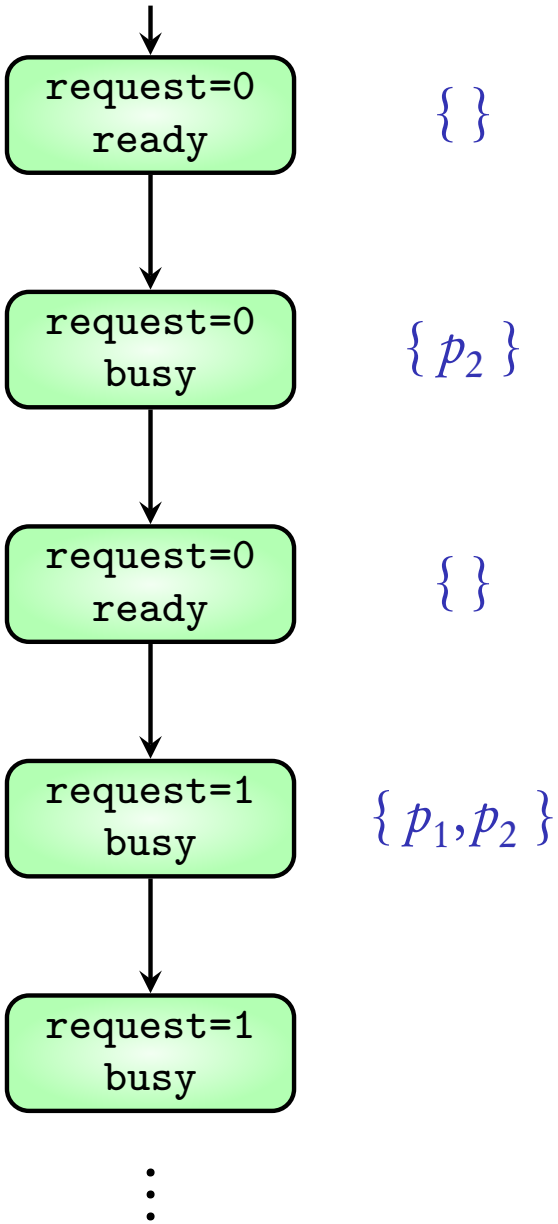
$\{\ \}$

$\{\ p_2\ \}$

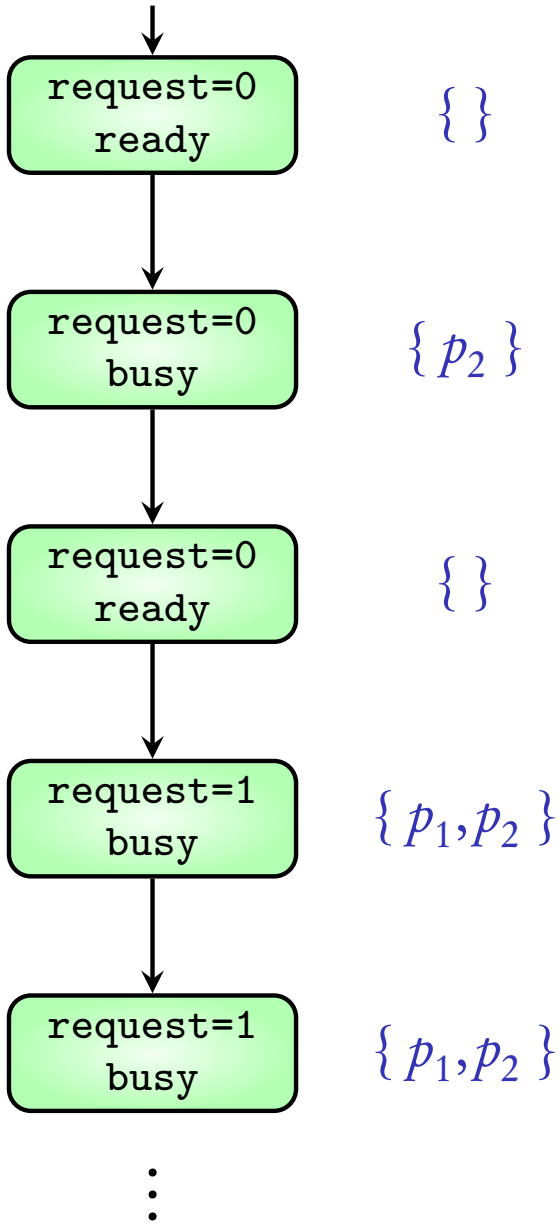
Execution

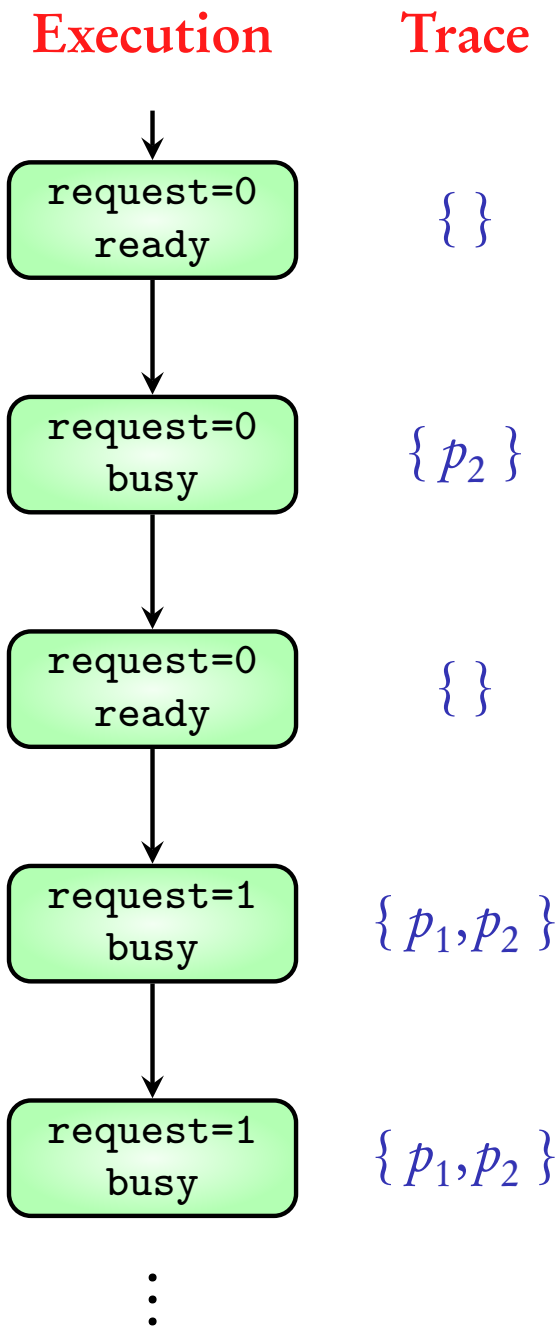

request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
request=1
busy

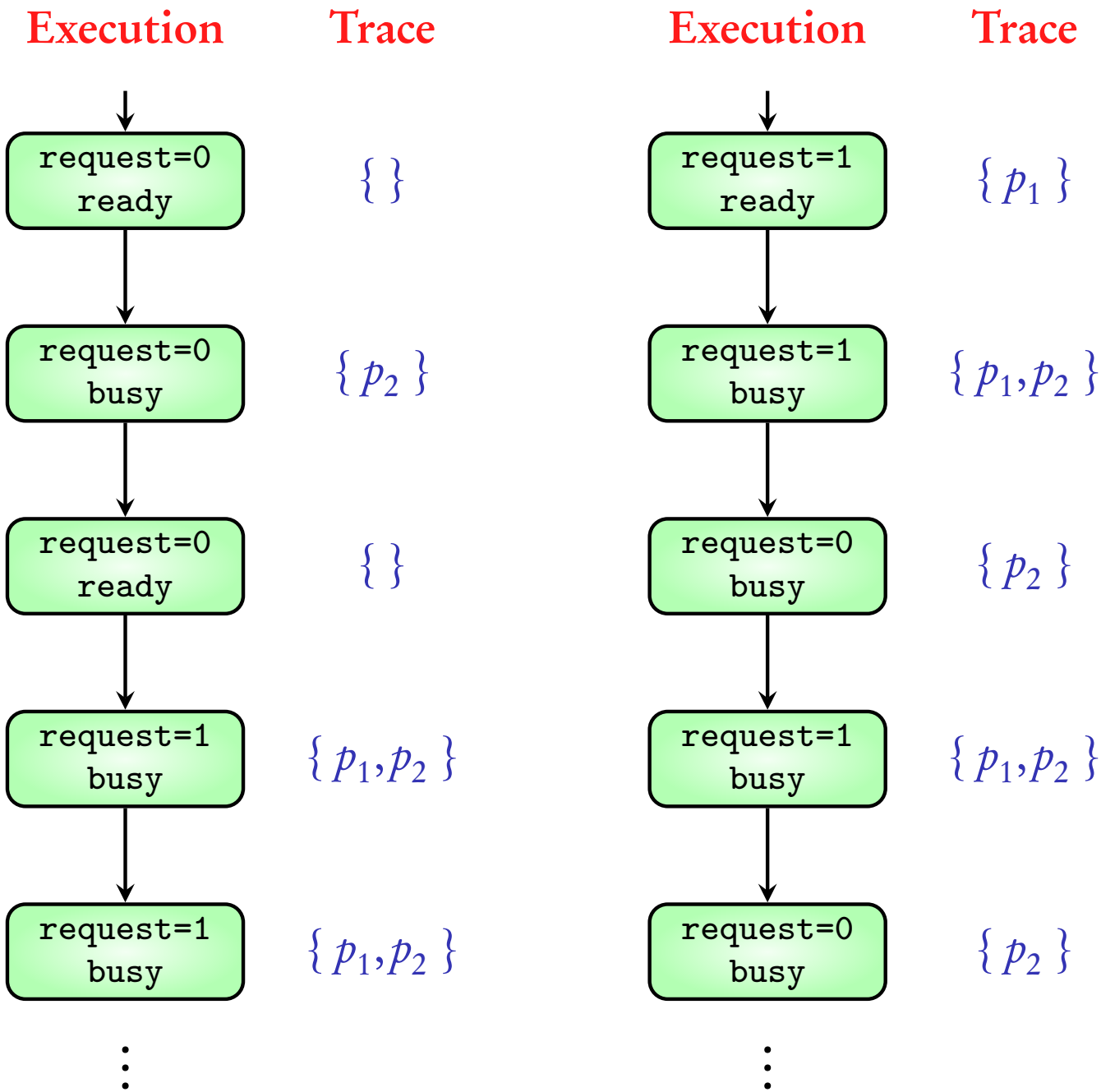
Execution


request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
{ p_1,p_2 }
request=1
busy

Execution


request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
{ p_1,p_2 }
request=1
busy
{ p_1,p_2 }


Execution
Trace
request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
{ p_1,p_2 }
request=1
busy
{ p_1,p_2 }


Execution
Trace
request=0
ready
{ }
request=0
busy
{ p_2 }
request=0
ready
{ }
request=1
busy
{ p_1,p_2 }
request=1
busy
{ p_1,p_2 }
Execution
Trace
request=1
ready
{ p_1 }
request=1
busy
{ p_1,p_2 }
request=0
busy
{ p_2 }
request=1
busy
{ p_1,p_2 }
request=0
busy
{ p_2 }

$$\mathbf{AP} = \{ p_1, p_2, \cdots, p_k \}$$

$$
\begin{aligned}
\mathbf{AP} = & \quad \{\ p_1, p_2, \ \cdots, p_k\ \} \\
\mathit{PowerSet}(\mathbf{AP}) = & \quad \{\ \{\ \},\ \{p_1\},\ \cdots,\ \{p_k\}, \\
& \qquad \{\ p_1, p_2\ \},\ \{\ p_1, p_3\ \},\ \cdots,\ \{\ p_{k-1}, p_k\ \}, \\
& \qquad\quad \cdots \\
& \qquad \{\ p_1, p_2, \ \cdots, \ p_k\ \}\ \}
\end{aligned}
$$

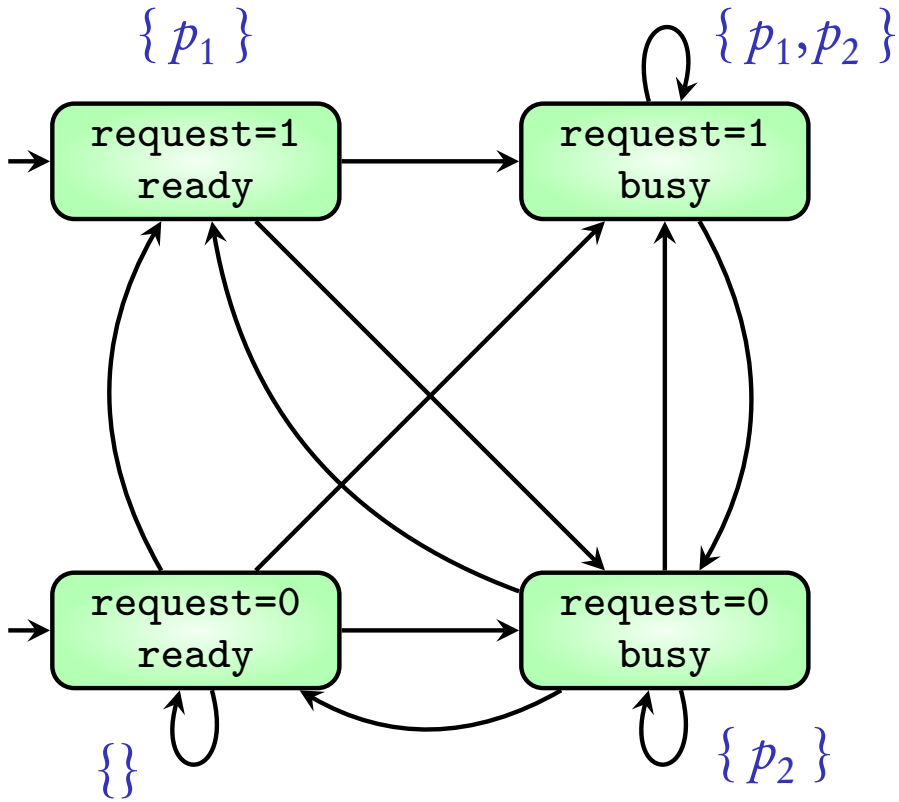
$$
\begin{aligned}
\mathbf{AP} = \quad & \{\ p_1, p_2, \ \cdots, p_k\ \} \\
\mathit{PowerSet}(\mathbf{AP}) = \quad & \{\ \{\ \},\ \{p_1\},\ \cdots,\ \{p_k\}, \\
& \quad \{\ p_1, p_2\ \},\ \{\ p_1, p_3\ \},\ \cdots,\ \{\ p_{k-1}, p_k\ \}, \\
& \qquad \cdots \\
& \quad \{\ p_1, p_2, \ \cdots, \ p_k\ \}\ \}
\end{aligned}
$$

Trace(Execution) is an **infinite word** over *PowerSet*(**AP**)

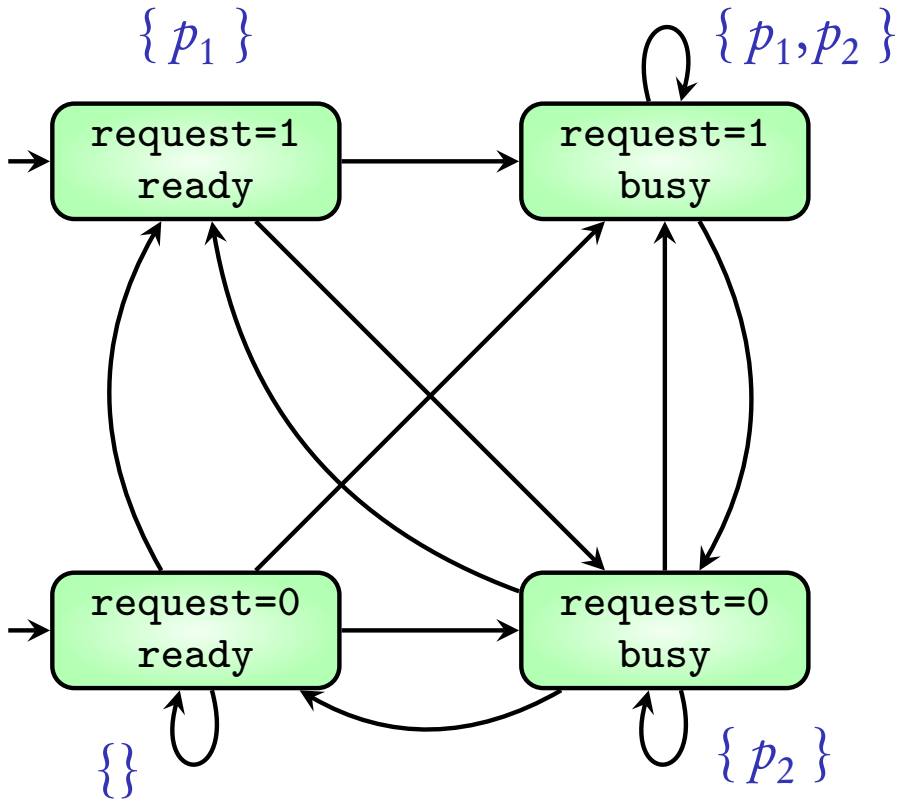
$$\begin{aligned}
\mathbf{AP} = &\ \{\ p_1, p_2, \ \cdots\ , p_k\ \} \\
\mathit{PowerSet}(\mathbf{AP}) = &\ \{\ \{\ \},\ \{p_1\},\ \cdots\ ,\ \{p_k\}, \\
&\quad \{\ p_1, p_2\ \},\ \{\ p_1, p_3\ \},\ \cdots\ ,\ \{\ p_{k-1}, p_k\ \}, \\
&\qquad \cdots \\
&\quad \{\ p_1, p_2, \ \cdots, \ p_k\ \}\ \}
\end{aligned}$$

Trace(Execution) is an **infinite word** over *PowerSet*(**AP**)

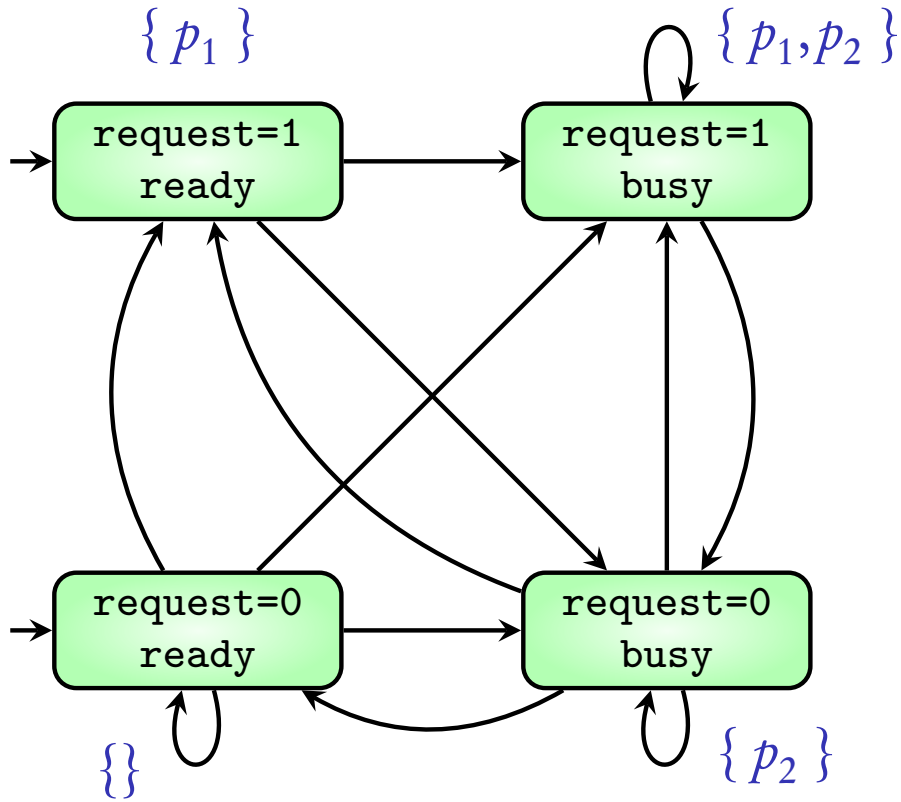
Traces(TS) is the $\{$ Trace(σ) $\mid$ σ is an execution of the TS $\}$


{ p1 }
{ p1,p2 }
request=1
ready
request=1
busy
request=0
ready
request=0
busy
{}
{ p2 }

Traces:

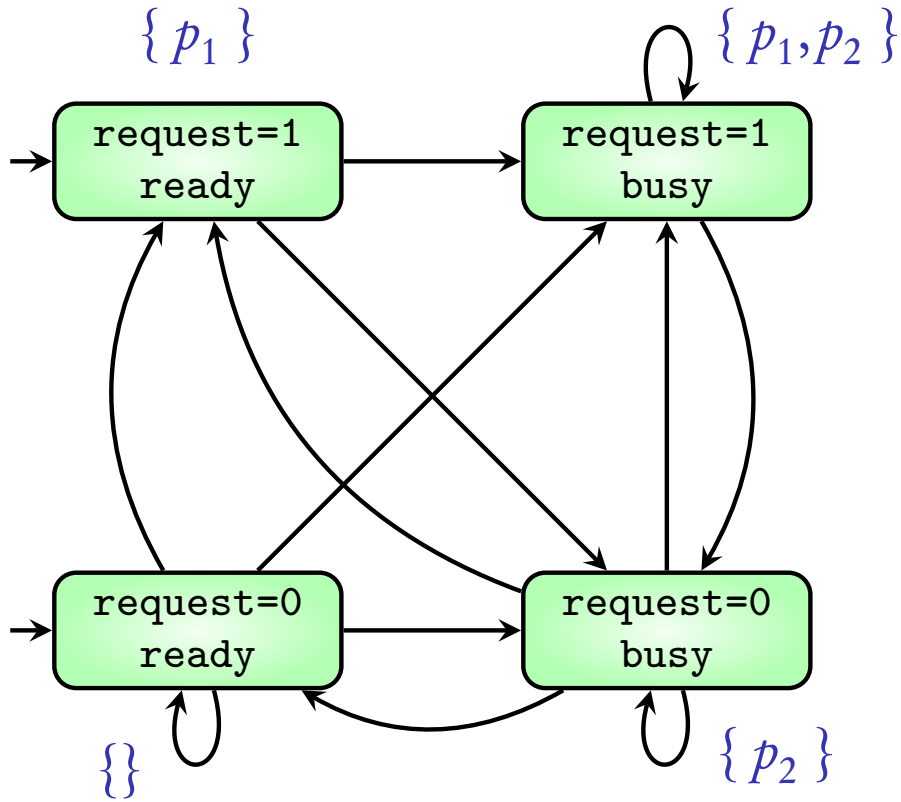


Traces: $\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\cdots$



Traces: $\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\cdots$

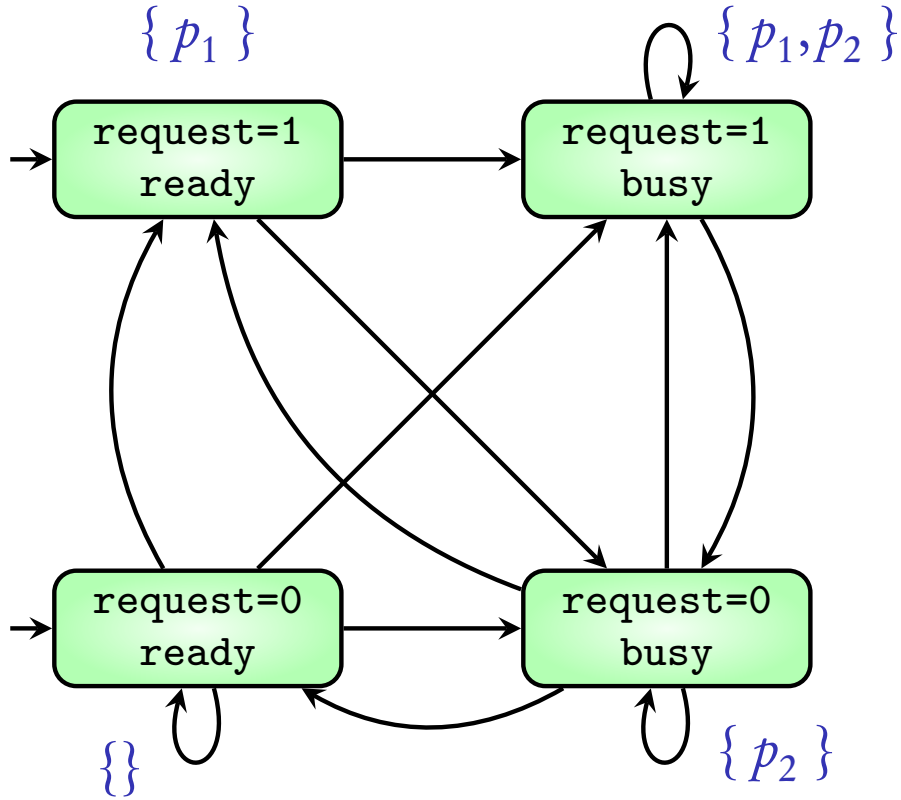
$\{\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\cdots$



Traces: $\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\cdots$

$\{\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\cdots$

$\{p_1\}\{p_1,p_2\}\{p_2\}\{p_1,p_2\}\{p_2\}\{p_1,p_2\}\cdots$

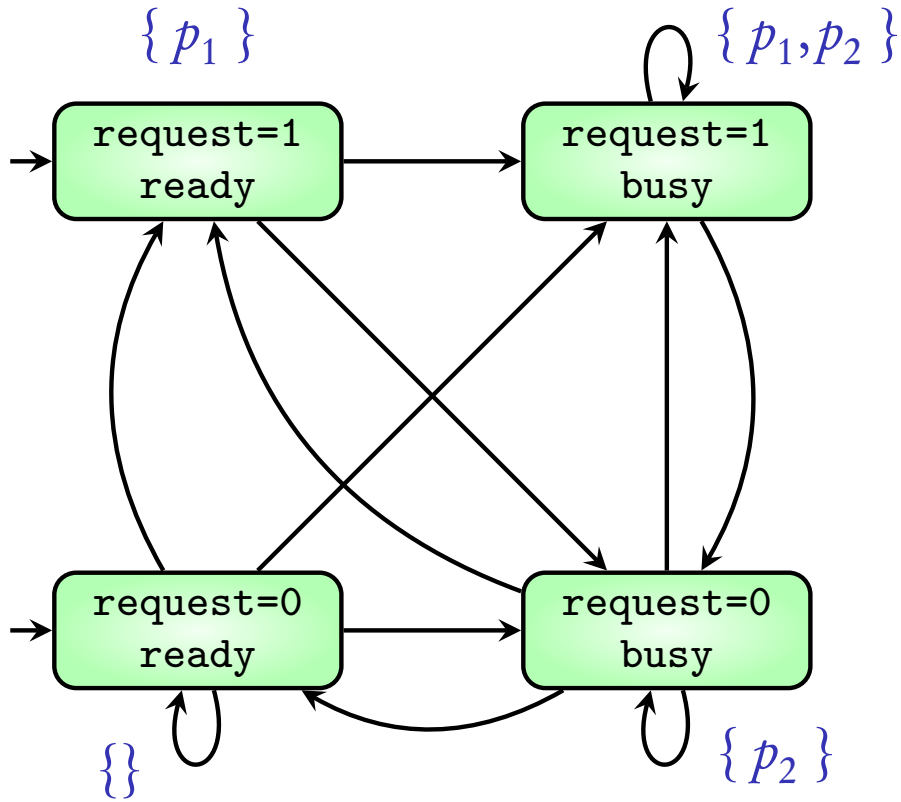


Traces: $\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\cdots$

$\{\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\cdots$

$\{p_1\}\{p_1,p_2\}\{p_2\}\{p_1,p_2\}\{p_2\}\{p_1,p_2\}\cdots$

$\{\}\{p_1,p_2\}\{p_1,p_2\}\{p_1,p_2\}\{p_1,p_2\}\{p_1,p_2\}\cdots$



Traces:

$\{\}\{\}\{\}\{\}\{\}\{\}\{\}\{\}\cdots$

$\{\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\cdots$

$\{p_1\}\{p_1,p_2\}\{p_2\}\{p_1,p_2\}\{p_2\}\{p_1,p_2\}\cdots$

$\{\}\{p_1,p_2\}\{p_1,p_2\}\{p_1,p_2\}\{p_1,p_2\}\{p_1,p_2\}\cdots$

$\vdots$

Traces of a TS describe its **behaviour** with respect to the atomic propositions

Behaviour of TS

Atomic propositions

Set of its **traces**

Coming next: What is a **property**?

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$$\{\, A_0A_1A_2\cdots \in \text{AP-INF} \mid \textbf{each } A_i \text{ contains } p_1 \,\}$$

$$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\cdots$$
$$\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\cdots$$
$$\vdots$$

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$$\{ A_0 A_1 A_2 \cdots \in \text{AP-INF} \mid \textbf{each } A_i \text{ contains } p_1 \}$$

$$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\cdots$$
$$\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\cdots$$
$$\vdots$$

Property 2: p_1 is true at least once and p_2 is always true

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

Property 1: p_1 is always true

$$\{ A_0A_1A_2\cdots \in \text{AP-INF} \mid \textbf{each } A_i \text{ contains } p_1 \}$$

$$\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\{p_1\}\cdots$$
$$\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\{p_1\}\{p_1,p_2\}\cdots$$
$$\vdots$$

Property 2: p_1 is true at least once and p_2 is always true

$$\{ A_0A_1A_2\cdots \in \text{AP-INF} \mid \textbf{exists } A_i \text{ containing } p_1 \text{ and } \textbf{every } A_j \text{ contains } p_2 \}$$

$$\{p_2\}\{p_1,p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_1,p_2\}\{p_2\}\cdots$$
$$\{p_1,p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\{p_2\}\cdots$$
$$\vdots$$

AP-INF = set of **infinite words** over *PowerSet*(**AP**)

A property over AP is a **subset** of AP-INF

Behaviour of TS

Atomic propositions

Set of its **traces**

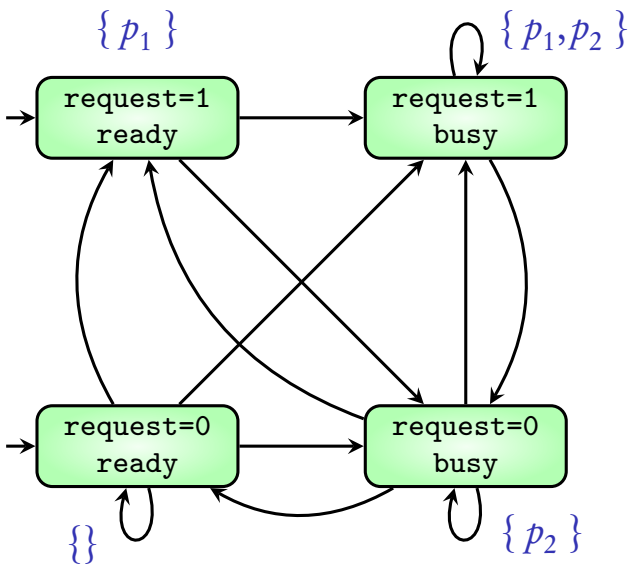
Property over AP

Subset of AP-INF

When does a transition system **satisfy** a property?

$$\mathrm{AP} = \{\ p_1, p_2\ \}$$

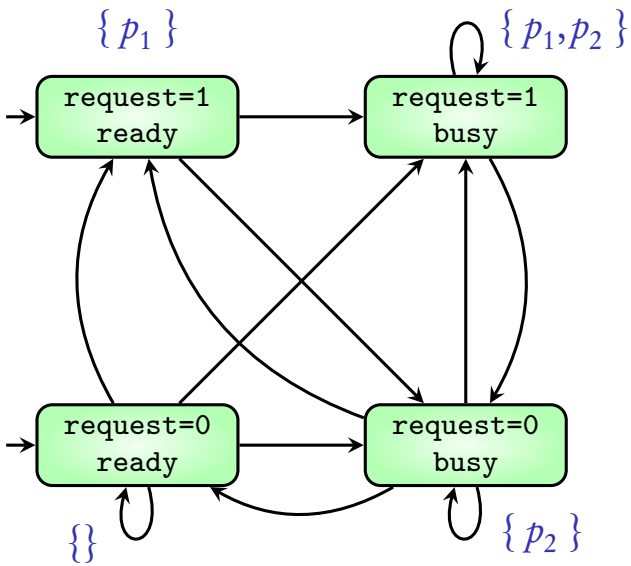
Transition System



$$\text{AP} = \{\ p_1, p_2\ \}$$

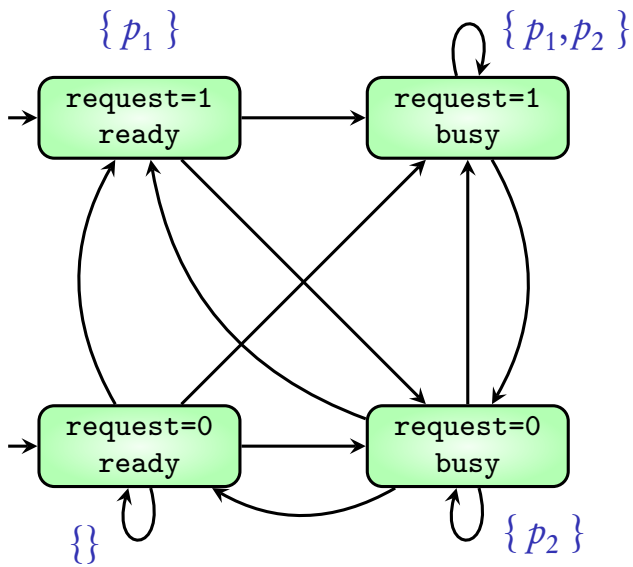
Transition System

Property



$$\mathrm{AP} = \{\ p_1, p_2\ \}$$

Transition System



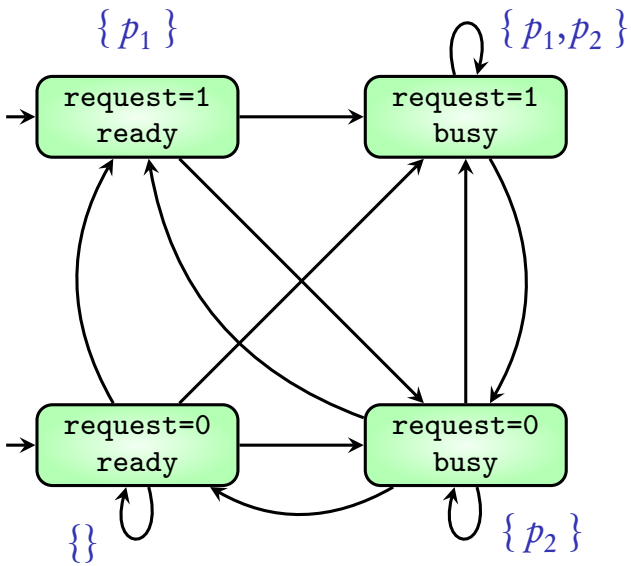
Property

$$\mathbf{G}\, p_1$$

$$\mathrm{AP} = \{\ p_1, p_2\ \}$$

Transition System

Property



$\mathbf{G}\, p_1$

Transition system TS satisfies property P if

$$\mathbf{Traces}(TS) \subseteq P$$

A property over AP is a subset of AP-INF

A property over AP is a subset of AP-INF

$\longrightarrow$ hence also called **Linear-time property**

Behaviour of TS

Atomic propositions

Set of its **traces**

Property over AP

Subset of AP-INF

When does system satisfy property?

Take-away

- Use of `MODULE` in NuSMV
- **Synchronous Vs Asynchronous** composition of modules
- Mutual exclusion: checked some kind of **safety** property
- What properties do we check for detecting **deadlocks**?
- Definition of **property**