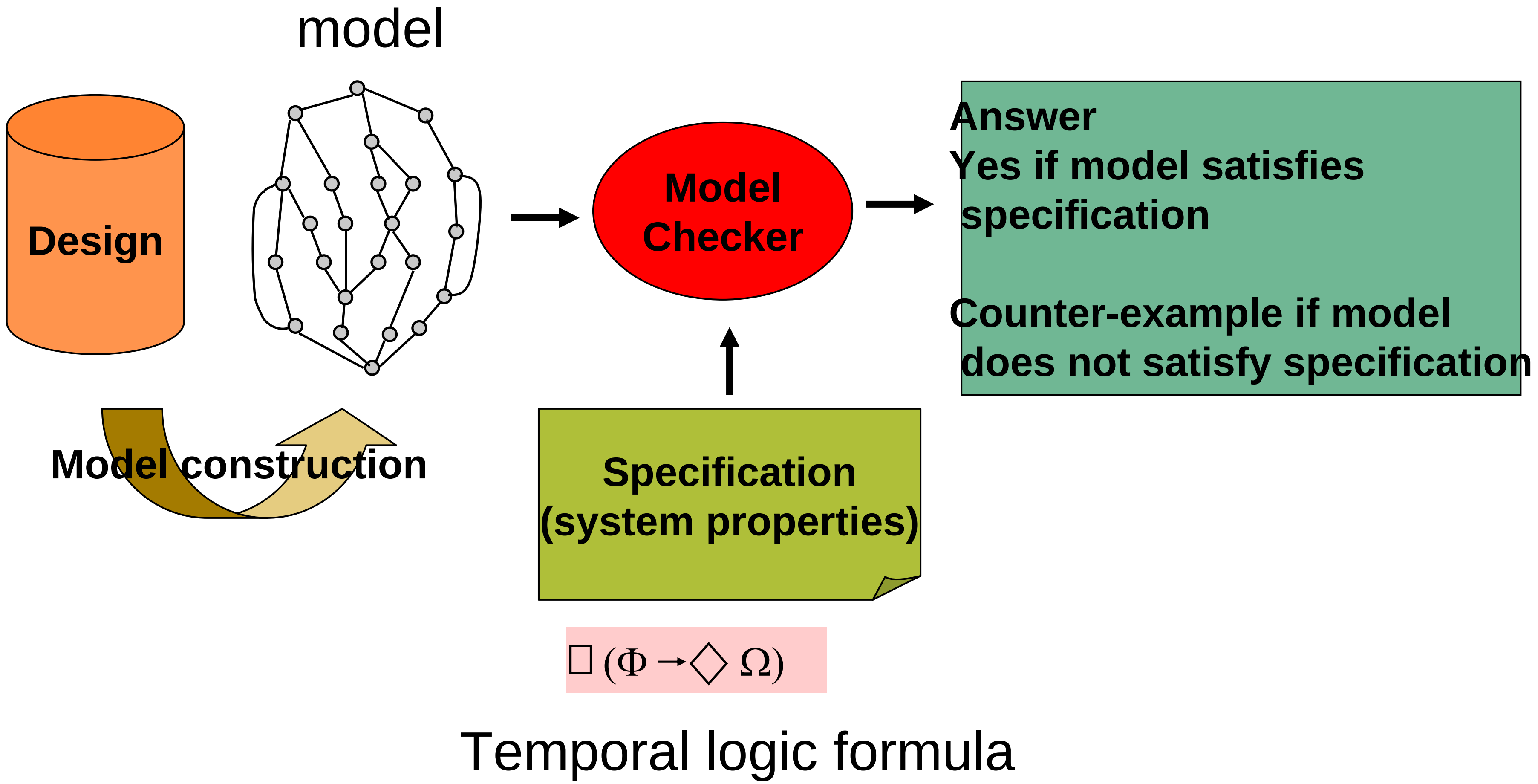
Temporal Logics and Model Checking

Outline

- Temporal Logic
  - Linear
    - LPTL (Linear time Propositional Temporal Logics)
  - Branching
    - CTL (Computation Tree Logics)
    - CTL* (the full branching temporal logics)
- Models
  - Kripke structure
  - Timed automata (TA)
  - Communicating Timed Automata (CTA)
- Model checking example
  - A SPIN example for mutual exclusion problem
  - A RED example for fisher’s timed mutual exclusion algorithm

Model Checking Framework



Temporal Logics

- Linear
  - LPTL (Linear time Propositional Temporal Logics)
- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

Temporal Logics：Catalog

propositional	↔	first-order
global	↔	compositional
branching	↔	linear-time
points	↔	intervals
discrete	↔	continuous
past	↔	future

Linear Time Propositional Temporal Logics (LPTL)

Basic assumption :

- Isomorphism: *(N*，<)
  - discrete ；suitable for digital computer
  - Initial point（0） ；computer needs reboot
  - Infinite future ；finite and infinite
- Every element in N is a state
  - Every state only have one successor

LPTL

Conventional notation：

- propositions : ***p, q, r, …***
- sets : ***A, B, C, D, …***
- states : ***s***
- state sequences : ***S***
- formulas : φ**,** ϕ
- Set of natural number：*N* ***= {0, 1, 2, 3, …}***
- Set of real number：*R*

LPTL

Given ***P*** : a set of propositions,

a Linear-time structure : *state sequence*

$$\boldsymbol{S = s_0\, s_1\, s_2\, s_3\, s_4 \ldots\, s_k \ldots\ldots}$$

$\boldsymbol{s_k}$ is a function of P where $\boldsymbol{P \rightarrow \{true,false\}}$

Syntax of LPTL

$$\phi ::= \textbf{\textit{true}} \mid \textbf{\textit{p}} \mid \neg \phi \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 U \phi_2$$

abbreviation

$$\textbf{\textit{false}} \equiv \neg\, true$$

$$\phi_1 \wedge \phi_2 \equiv \neg ((\neg \phi_1) \vee (\neg \phi_2))$$

$$\phi_1 \rightarrow \phi_2 \equiv (\neg \phi_1) \vee \phi_2$$

$$\Diamond \phi \equiv \textbf{\textit{true}}\ U \phi$$

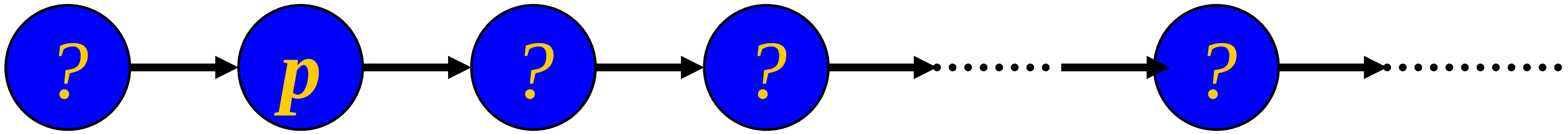
$$\Box \phi \equiv \neg \Diamond \neg \phi$$

Syntax of LPTL

Exam.	Symbol in CMU	
○***p***	**X*p***	***p is*** **true on next state**
p**U* ***q	***p**U* ***q***	**From now on, *p* is always true until *q* is true**
◇***p***	**F*p***	**From now on, there will be a state where *p* is eventually (sometimes) true**
□***p***	**G*p***	**From now on, *p* is always true**

Syntax of LPTL

○*p* **X*p*** ***p is*** **true on next state**



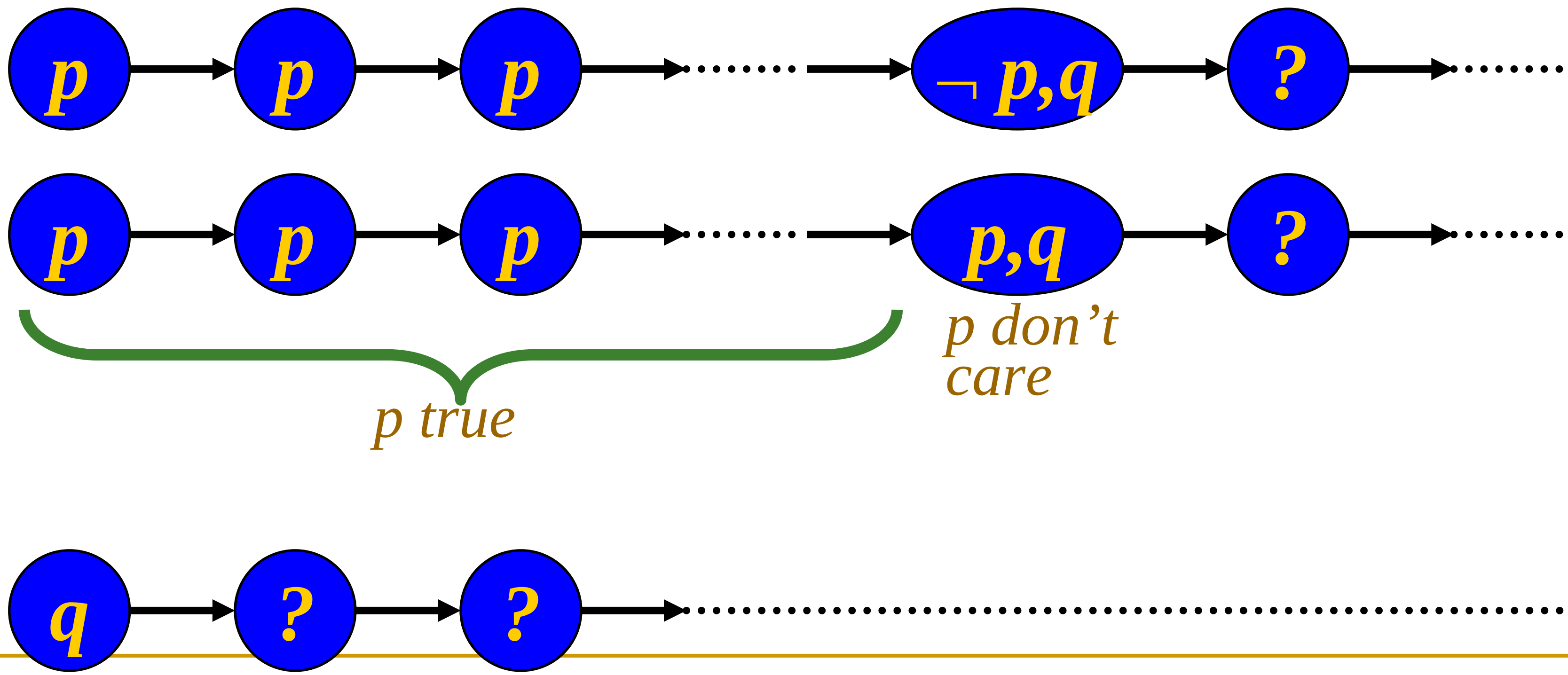
? : don’t care

Syntax of LPTL

p *U* ***q***

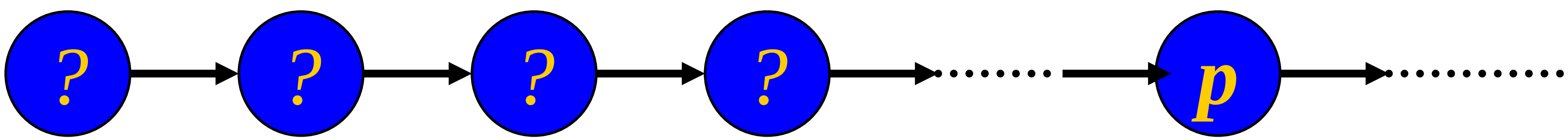
p *U* ***q***

From now on, *p* is always true until *q* is true

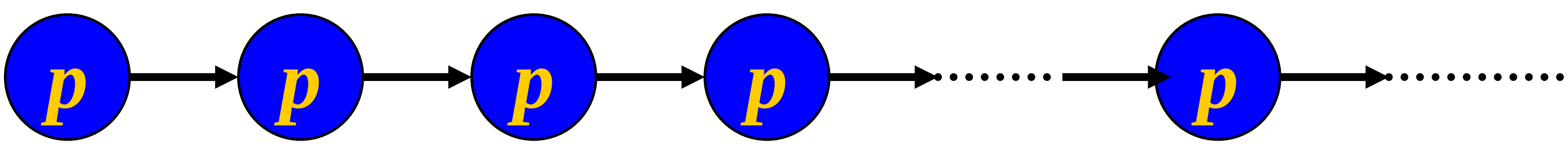


Syntax of LPTL

$\Diamond p$ | Fp | **From now on, there will be a state where *p* is eventually (sometimes) true**



$\Box p$ | Gp | **From now on, *p* is always true**



Syntax of LPTL

Two operator for Fairness

- $\Diamond^{\infty} \boldsymbol{p} \equiv \Box \Diamond \boldsymbol{p}$; ***p*** **will happen infinitely many times**
  infinitely often

- $\Box^{\infty} \boldsymbol{p} \equiv \Diamond \Box \boldsymbol{p}$; ***p*** **will be always true after some time in the future**
  almost everywhere

Semantics of LPTL

suffix path :

$$S = s_0 s_1 s_2 s_3 s_4 s_5 \ldots\ldots\ldots$$

$$S^{(0)} = s_0 s_1 s_2 s_3 s_4 s_5 \ldots\ldots\ldots$$

$$S^{(1)} = s_1 s_2 s_3 s_4 s_5 s_6 \ldots\ldots\ldots$$

$$S^{(2)} = s_2 s_3 s_4 s_5 s_6 \ldots\ldots\ldots$$

$$S^{(3)} = s_3 s_4 s_5 s_6 \ldots\ldots\ldots$$

$$S^{(k)} = s_k s_{k+1} s_{k+2} s_{k+3} \ldots\ldots\ldots$$

Semantics of LPTL

Suppose there is a state sequence

$$S = s_0 s_1 s_2 s_3 s_4 \ldots s_k \ldots\ldots$$

We define $S \models \phi$ (*S* **satisfies** ϕ) **as**：

- $S \models true$
- $S \models p \Leftrightarrow s_0(p)=true$, *or equivalently* $p \in s_0$
- $S \models \neg\phi \Leftrightarrow S \models \phi$ **is** ***false***
- $S \models \phi_1 \vee \phi_2 \Leftrightarrow S \models \phi_1$ **or** $S \models \phi_2$
- $S \models \bigcirc\phi \Leftrightarrow S^{(1)} \models \phi$
- $S \models \phi_1 U \phi_2 \Leftrightarrow \exists k \geq 0 (S^{(k)} \models \phi_2 \wedge \forall 0 \leq j < k (S^{(j)} \models \phi_1))$

Semantics of LPTL

- Assume there is a state sequence ***S*** **which satisfies** φ ***(S*** $\models \varphi$***)***

  then *S* is one of the model of φ***.***

- Assume there is a state sequence ***S sat*** φ ，

  then φ **is *satisfiable*; otherwise** φ ***is unsatisfiable*** 。

- **If for all state sequence *S*** $\models \varphi$ ，

  then φ **is *valid*** 。（$\models \varphi$）

Semantics of LPTL

example： $\Box$（*found enemy* $\rightarrow$ $\Diamond$*destroy enemy*）

- Can't conveniently express enemies appear concurrently.

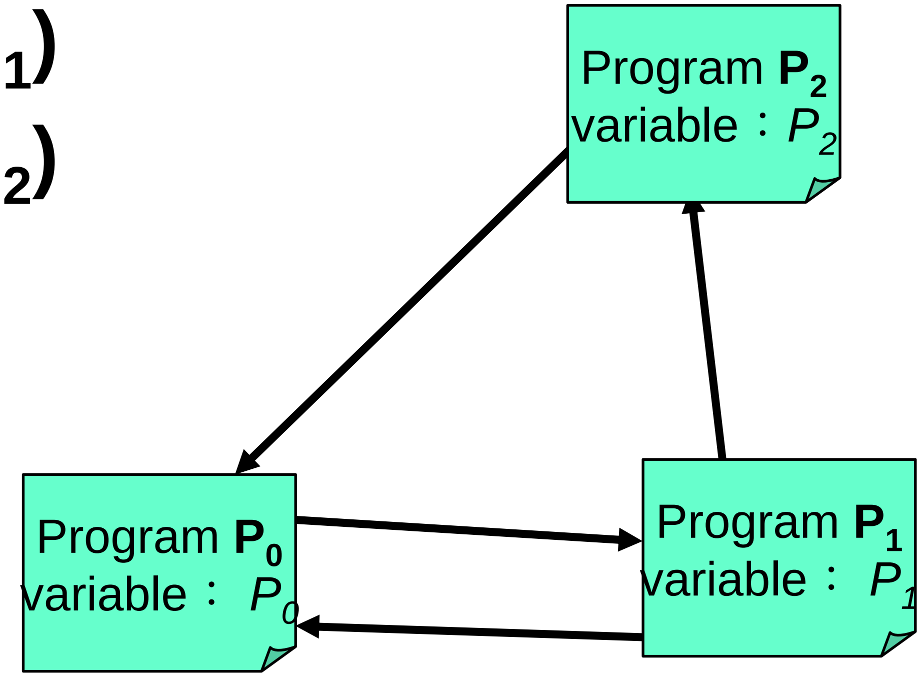
example： $\Box\neg$（***A*** *is executing* $\wedge$ ***B*** is executing）

Example of writing LPTL formula (I)

$\mathbf{P_0}$**:(*p_0*:=0 | *p_0* := *p_0* $\vee$ *p_1* $\vee$ *p_2*)**

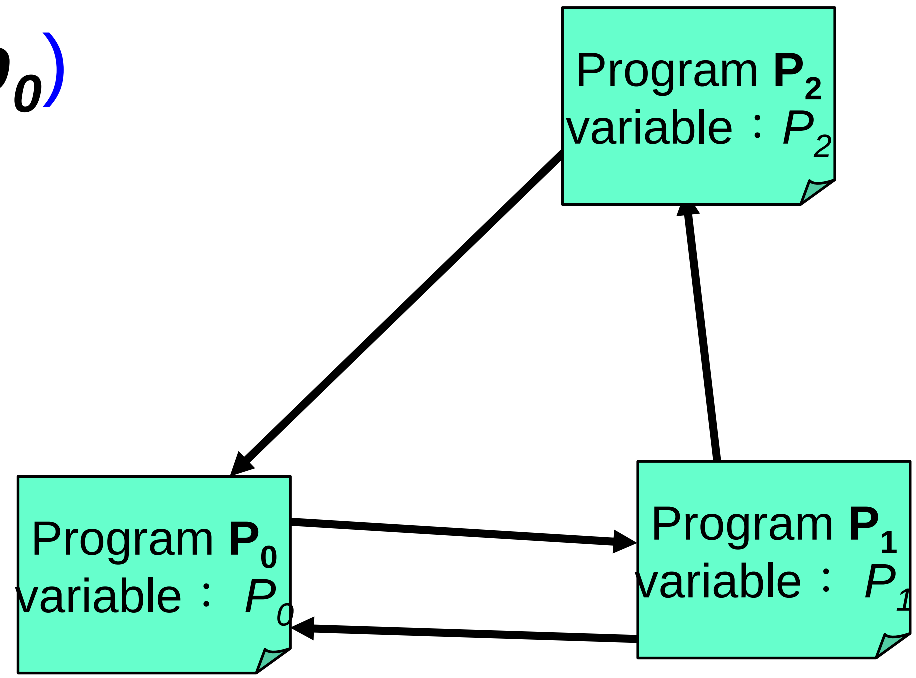
P_1**:(*p_1*:=0 | *p_1* := *p_0* $\vee$ *p_1*)**

P_2**:(*p_2*:=0 | *p_2* := *p_1* $\vee$ *p_2*)**



Example of writing LPTL formula (I)

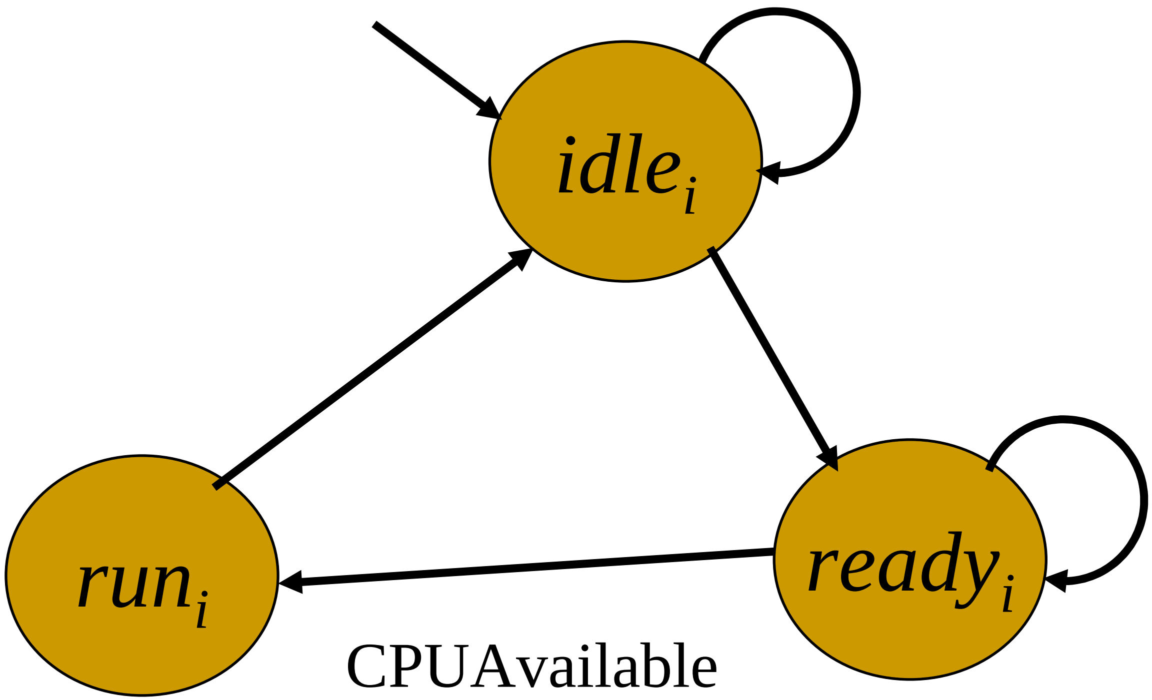
$$
\begin{aligned}
\Box(\; & \bigcirc\neg p_0 \\
& \vee ((p_0 \vee p_1 \vee p_2) \leftrightarrow \bigcirc p_0) \\
& \vee \bigcirc\neg p_1 \\
& \vee ((p_0 \vee p_1) \leftrightarrow \bigcirc p_1) \\
& \vee \bigcirc\neg p_2 \\
& \vee ((p_1 \vee p_2) \leftrightarrow \bigcirc p_2) \\
)\;
\end{aligned}
$$



Example of writing LPTL formula (II)

Process$_i$, $1 \leq i \leq m$

Also describe the mutual exclusion condition



Example of writing LPTL formula (II)

$\bigwedge_{1 \le i \le m} \mathit{idle}_i$

$\wedge \bigwedge_{1 \le i \le m} \Box($

$\quad (\mathit{idle}_i \leftrightarrow (\bigcirc \mathit{idle}_i \vee \bigcirc \mathit{ready}_i))$

$\quad \vee (\mathit{ready}_i \leftrightarrow \bigcirc \mathit{ready}_i)$

$\quad \vee ((\mathit{idle}_i \wedge \bigwedge_{1 \le j \le m} \neg \mathit{run}_j) \leftrightarrow \bigcirc \mathit{run}_i)$

$\quad \vee (\mathit{run}_i \leftrightarrow \bigcirc \mathit{idle}_i)$

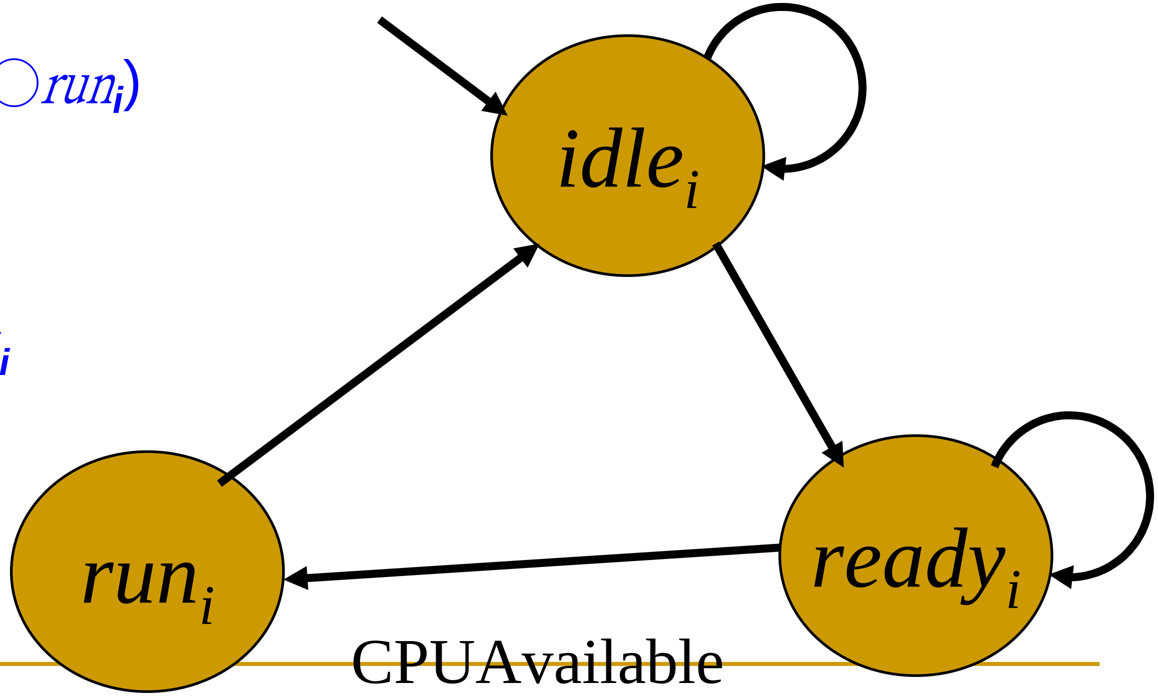
$\quad \vee (\quad \mathit{run}_i \leftrightarrow \bigcirc \mathit{run}_i$

$\qquad \wedge\ \mathit{idle}_i \leftrightarrow \bigcirc \mathit{idle}_i$

$\qquad \wedge\ \mathit{ready}_i \leftrightarrow \bigcirc \mathit{ready}_i$

$\qquad)$

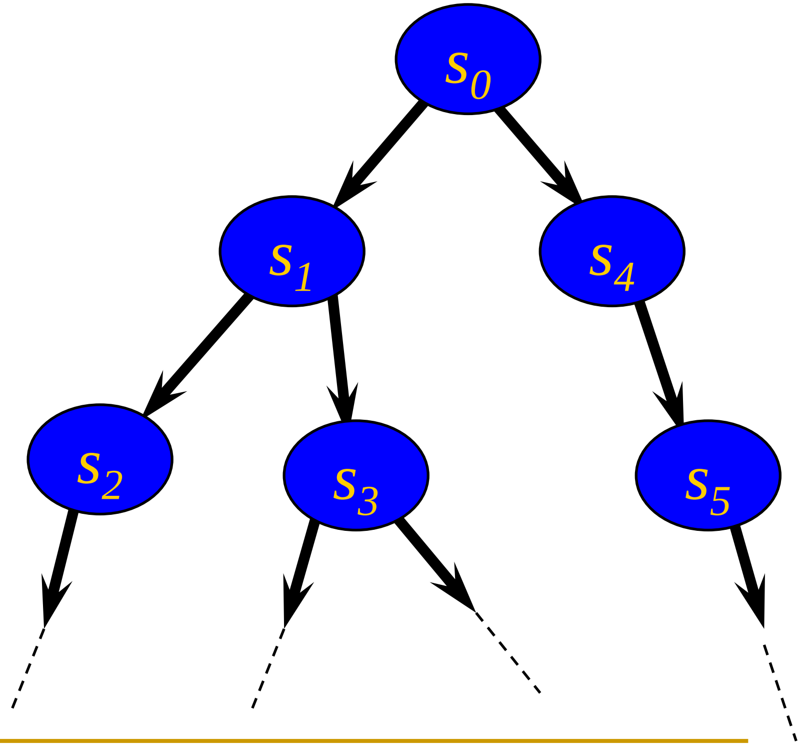
$)$



Branching Temporal Logic

Basic assumption of tree-like structure

- Every node is a function of P→{true,false}
- Every state may have many successors



Branching Temporal Logic

Basic assumption of tree-like structure

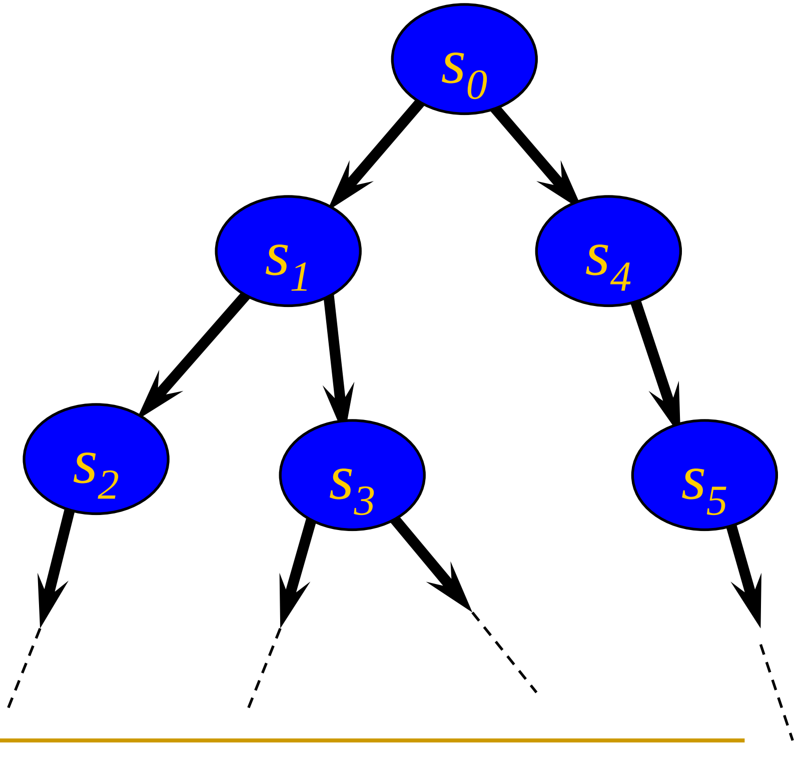
- Every path is isomorphic as ***N***
  - Correspond to a state sequence

Path : $s_0\ s_1\ s_3\ldots\ldots$

$s_0\ s_1\ s_2\ldots\ldots$

$s_1\ s_3\ldots\ldots$

$s_4\ s_5\ldots\ldots$



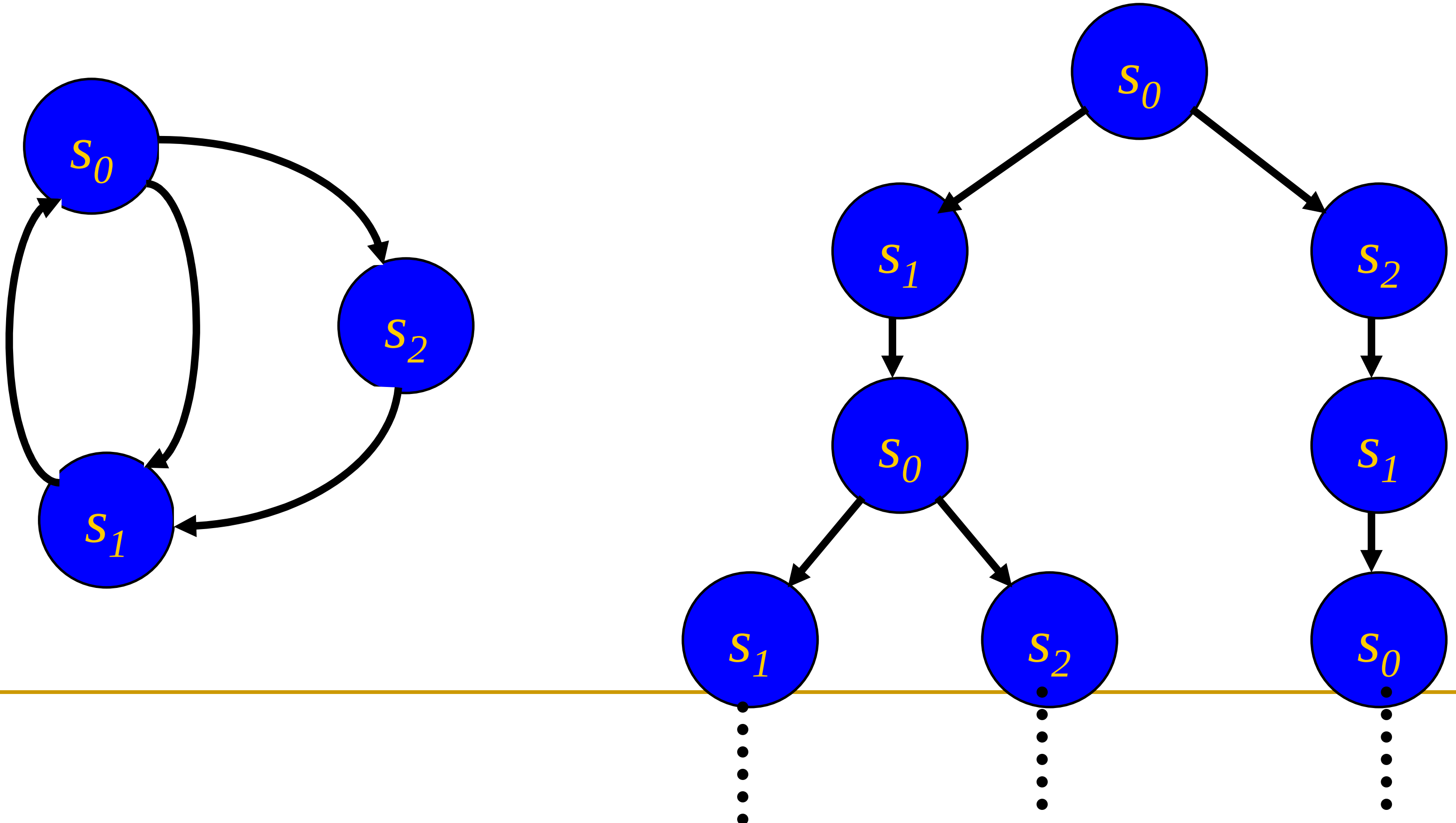
Branching Temporal Logic

It can accommodate infinite and dense state successors

- In CTL and CTL*, it can't tell
  - Finite and infinite
    - Is there infinite transitions？
  - Dense and discrete
    - Is there countable（ω）transitions？

Branching Temporal Logic

Get by flattening a finite state machine



Syntax of CTL(Computation Tree Logic)

$$\varphi ::= \textbf{\textit{true}} \mid \textbf{\textit{p}} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists \bigcirc \varphi \mid \forall \bigcirc \varphi \mid \exists\, \varphi_1 U \varphi_2 \mid \forall\, \varphi_1 U \varphi_2$$

abbreviation：

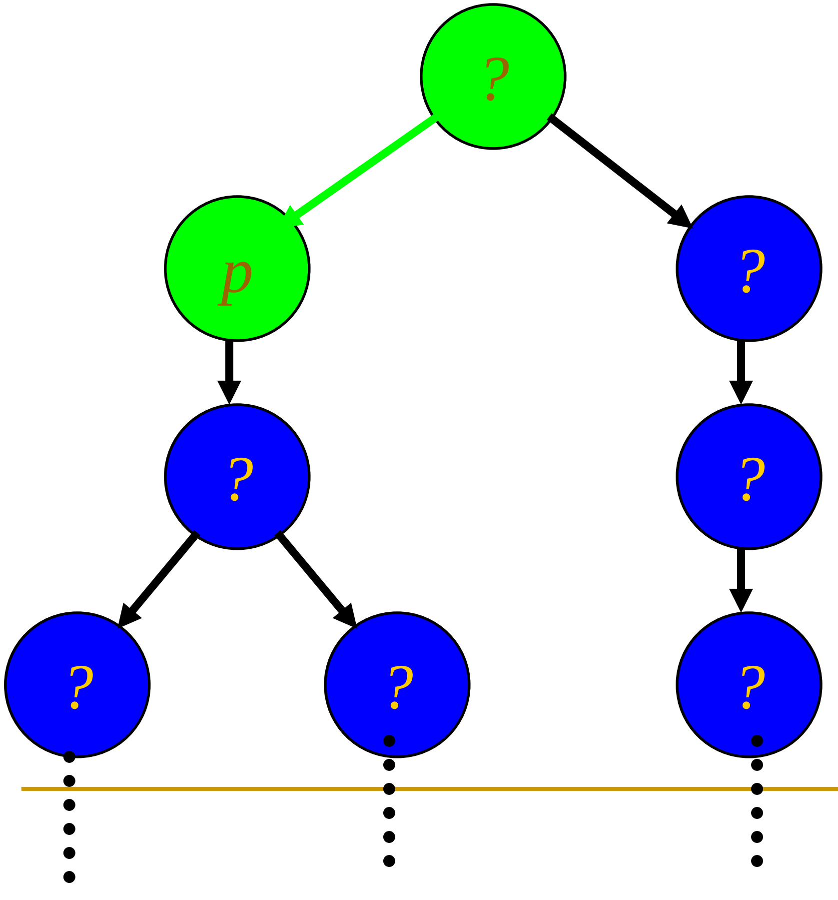
$\textbf{\textit{false}}$	$\equiv$	$\neg\, \textit{true}$
$\varphi_1 \wedge \varphi_2$	$\equiv$	$\neg((\neg\varphi_1) \vee (\neg\varphi_2))$
$\varphi_1 \rightarrow \varphi_2$	$\equiv$	$(\neg\varphi_1) \vee \varphi_2$
$\exists\Diamond\varphi$	$\equiv$	$\exists \textbf{\textit{true}}\, U \varphi$
$\forall\Box\varphi$	$\equiv$	$\neg\exists\Diamond\neg\varphi$
$\forall\Diamond\varphi$	$\equiv$	$\forall \textbf{\textit{true}}\, U \varphi$
$\exists\Box\varphi$	$\equiv$	$\neg\forall\Diamond\neg\varphi$

Semantics of CTL

example	symbol in CMU	
$\exists \bigcirc p$	**EX*p***	**there exists a path where *p* is true on next state**
$\exists p \, \mathcal{U} \, q$	***p*EU*q***	**from now on, there is a path where *p* is always true until *q* is true**
$\forall \bigcirc p$	**AX*p***	**for all path where *p* is true on next state**
$\forall p \, \mathcal{U} \, q$	***p*AU*q***	**from now on, for all path where *p* is always true until *q* is true**

Semantics of CTL

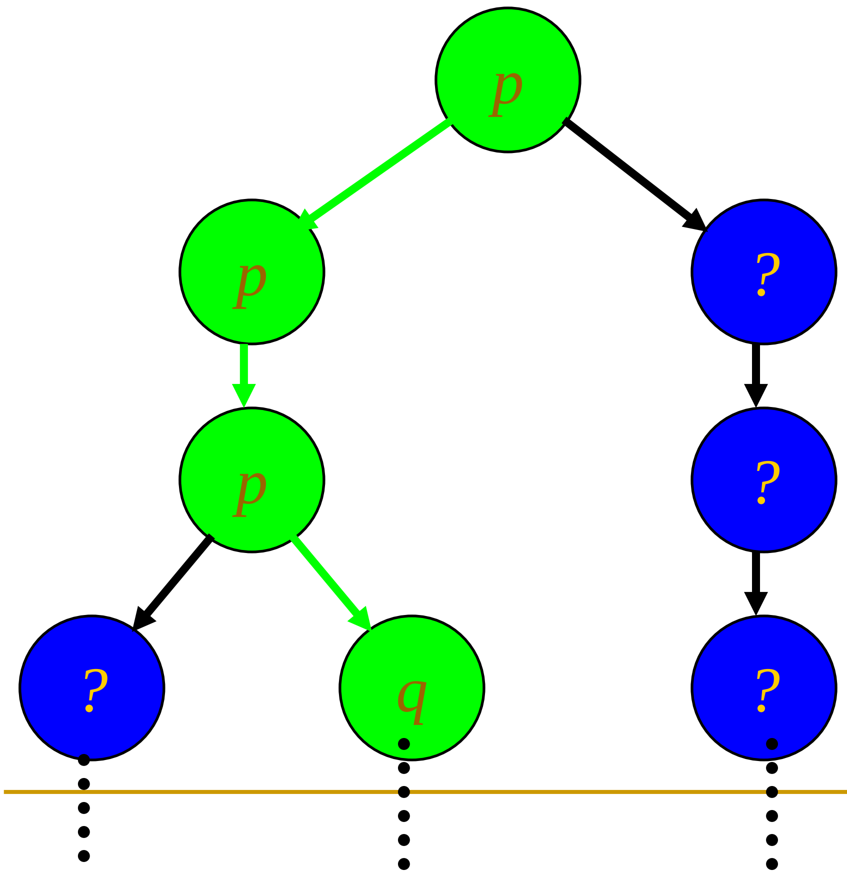
$\exists \bigcirc p$ **EX*p*** **there exists a path where *p* is true on next state**



Semantics of CTL

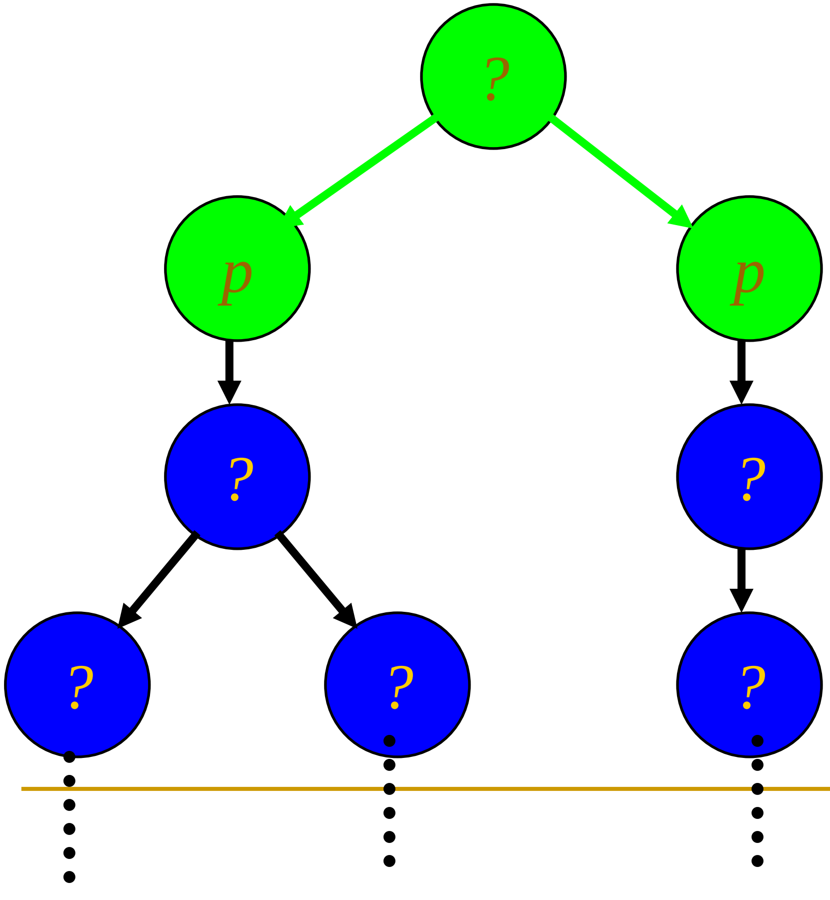
$\exists p \, U \, q$ **pEUq**

from now on, there is a path where *p* is always true until *q* is true



Semantics of CTL

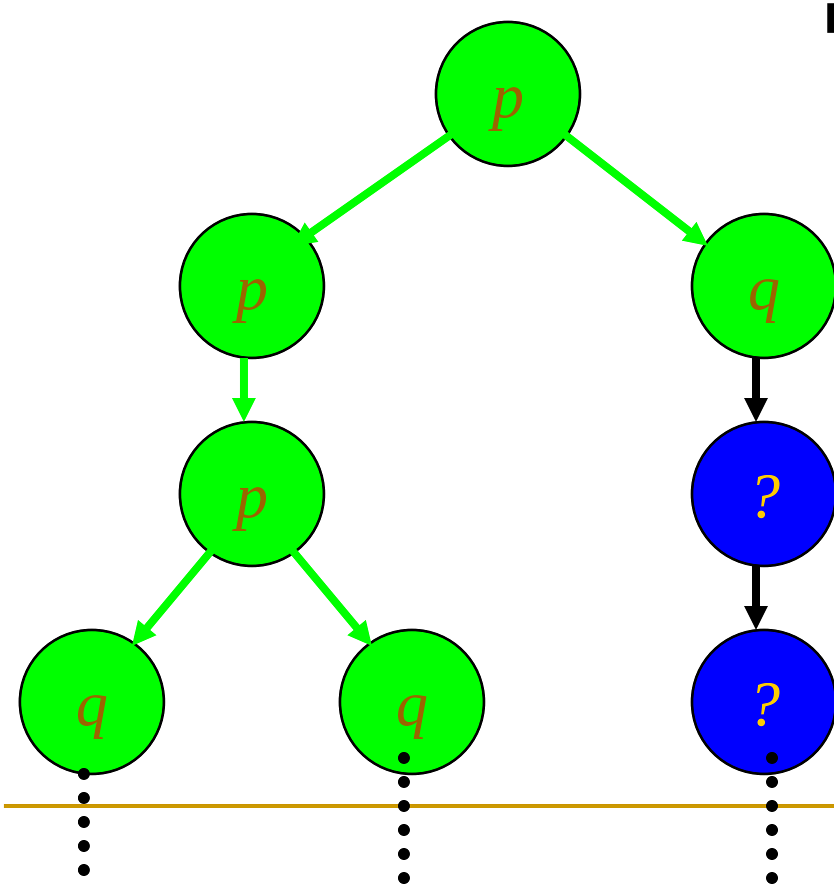
$\forall \bigcirc p$ **AX*p*** **for all path where *p* is true on next state**



Semantics of CTL

$\forall p \, U \, q$ **pAUq**

from now on, for all path where *p* is always true until *q* is true



Semantic of CTL

Assume there is a tree stucture ***M***、one state ***s*** in ***M*** and a CTL fomula φ

Define ***M,s*** $\models \varphi$ which means ***s*** in ***M*** satisfy φ

Semantics of CTL

s-path：a path in *M* which stats from ***s***

s_0 *-path:*

$s_0\ s_1\ s_2\ s_3\ s_5 \ldots\ldots\ldots$

$s_0\ s_1\ s_6\ s_7\ s_8 \ldots\ldots\ldots$

s_1 *-path:*

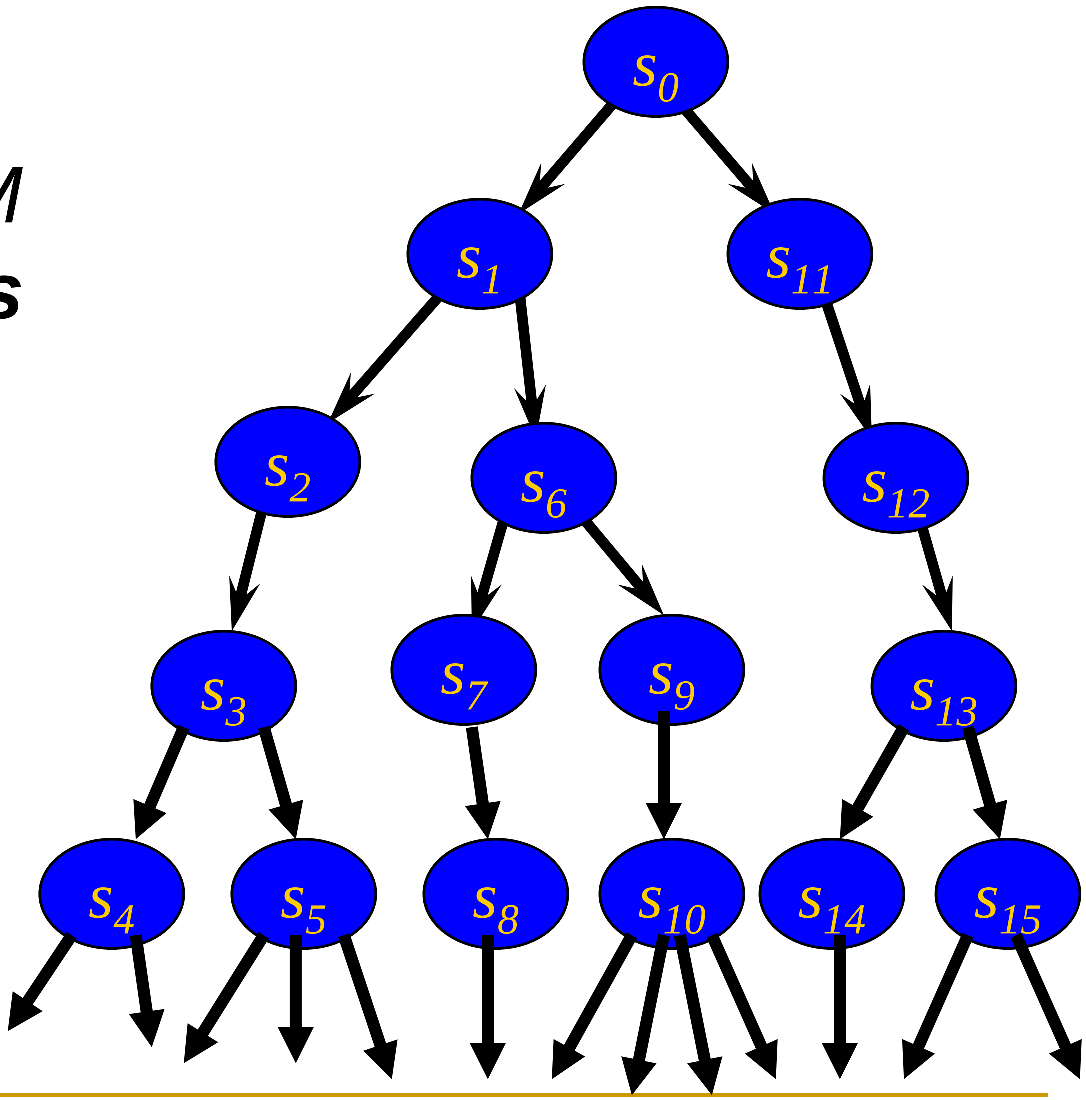
$s_1\ s_2\ s_3\ s_5 \ldots\ldots\ldots\ldots$

s_2 *-path:*

$s_2\ s_3\ s_5 \ldots\ldots\ldots\ldots$

s_{13} *-path:*

$s_{13}\ s_{15} \ldots\ldots\ldots\ldots$



Semantics of CTL

- $M,s \models true$
- $M,s \models p \Leftrightarrow p \in s$
- $M,s \models \neg \varphi \Leftrightarrow M,s \models \varphi$ 是*false*
- $M,s \models \varphi_1 \vee \varphi_2 \Leftrightarrow M,s \models \varphi_1$ or $M,s \models \varphi_2$
- $M,s \models \exists \bigcirc \varphi \Leftrightarrow \exists$ s-path $= s_0\, s_1 \ldots\ldots (M,s_1 \models \varphi)$
- $M,s \models \forall \bigcirc \varphi \Leftrightarrow \forall$ s-path $= s_0\, s_1 \ldots\ldots (M,s_1 \models \varphi)$
- $M,s \models \exists\, \varphi_1 U \varphi_2 \Leftrightarrow \exists$ s-path $= s_0\, s_1 \ldots\ldots, \exists k \geq 0\ (M,s_k \models \varphi_2 \wedge \forall 0 \leq j < k (M,s_j \models \varphi_1)$
- $M,s \models \forall\, \varphi_1 U \varphi_2 \Leftrightarrow \forall$ s-path $= s_0\, s_1 \ldots\ldots, \exists k \geq 0\ (M,s_k \models \varphi_2 \wedge \forall 0 \leq j < k (M,s_j \models \varphi_1)$

Syntax of CTL*

- ***CTL**** fomula (***state***-fomula)

$$\varphi ::= true \mid p \mid \neg \varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \exists \phi \mid \forall \phi$$

- ***path***-fomula

$$\phi ::= \varphi \mid \neg \phi_1 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi_1 \mid \phi_1 U \phi_2$$

CTL* is set of all ***state-***fomula *!*

Example of CTL*

- In a fair concurrent environment, jobs will eventually finish.

$$\forall(((\Box\Diamond \mathit{execute}_1) \wedge (\Box\Diamond \mathit{execute}_2)) \rightarrow \Diamond \mathit{finish})$$

or

$$\forall(((\Diamond^{\infty} \mathit{execute}_1) \wedge (\Diamond^{\infty} \mathit{execute}_2)) \rightarrow \Diamond \mathit{finish})$$

Semantics of CTL*

suffix path :

$S = s_0\, s_1\, s_2\, s_3\, s_5 \ldots\ldots\ldots$

$S^{(0)} = s_0\, s_1\, s_2\, s_3\, s_5 \ldots\ldots\ldots$

$S^{(1)} = s_1\, s_2\, s_3\, s_5 \ldots\ldots\ldots$

$S^{(2)} = s_2\, s_3\, s_5 \ldots\ldots\ldots$

$S^{(3)} = s_3\, s_5 \ldots\ldots\ldots$

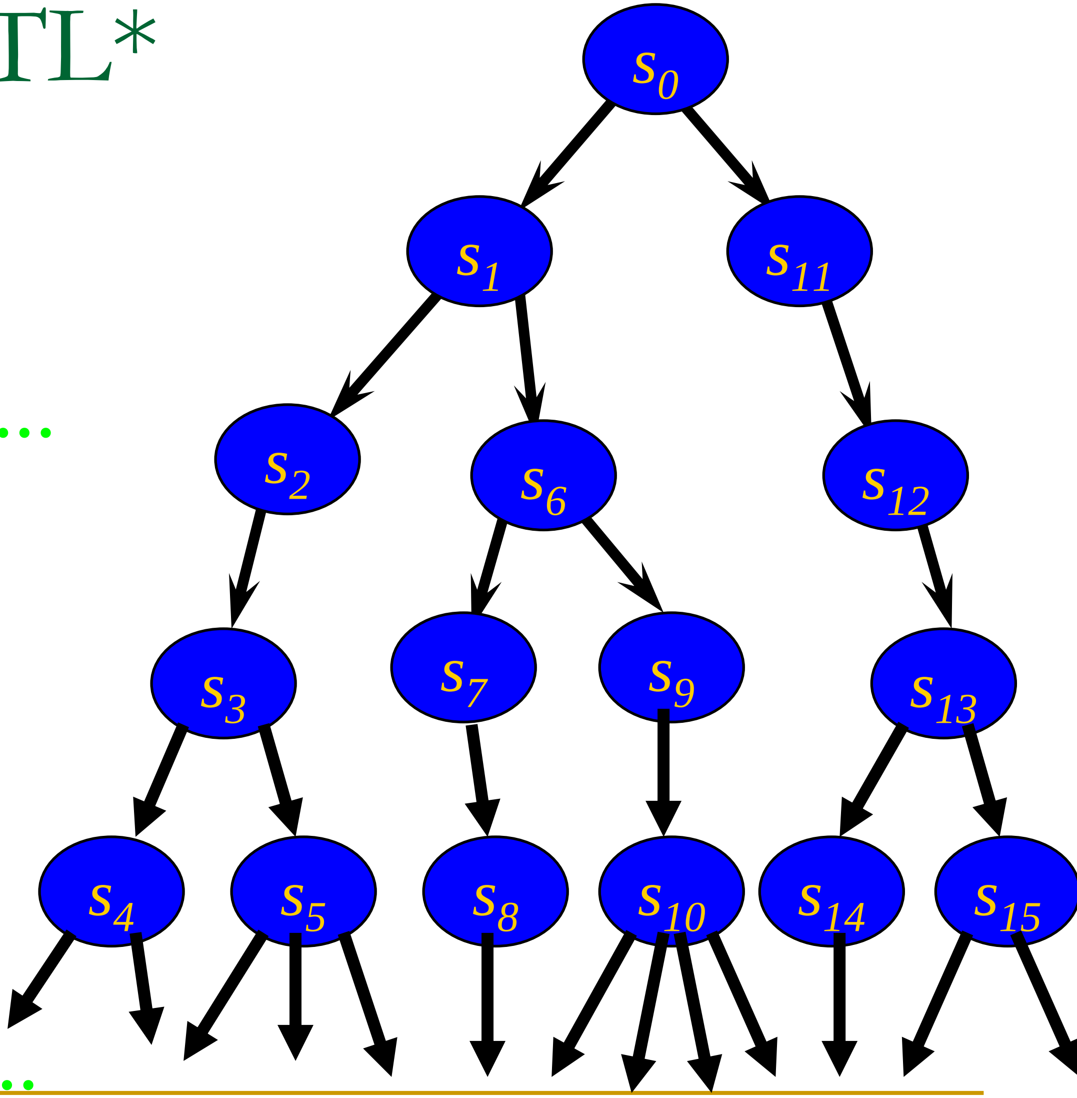
$S^{(4)} = s_5 \ldots\ldots\ldots$

$S = s_0\, s_1\, s_6\, s_7\, s_8 \ldots\ldots\ldots$

$S^{(2)} = s_6\, s_7\, s_8 \ldots\ldots\ldots$

$S = s_0\, s_{11}\, s_{12}\, s_{13}\, s_{15} \ldots\ldots\ldots$

$S^{(3)} = s_{13}\, s_{15} \ldots\ldots\ldots$



Semantics of CTL*

state-fomula

φ ::= *true* | *p* | $\neg\,\varphi_1$ | $\varphi_1 \vee \varphi_2$ | $\exists\,\phi$ | $\forall\,\phi$

- ***M,s*** $\models$ ***true***
- ***M,s*** $\models$ ***p*** $\Leftrightarrow$ ***p*** $\in$ ***s***
- ***M,s*** $\models \neg\,\varphi \Leftrightarrow$ ***M,s*** $\models \varphi$ 是***false***
- ***M,s*** $\models \varphi_1 \vee \varphi_2 \Leftrightarrow$ ***M,s*** $\models \varphi_1$ **or** ***M,s*** $\models \varphi_2$
- ***M,s*** $\models \exists\,\phi \Leftrightarrow \exists$ ***s-path = S (S*** $\models \phi$**)**
- ***M,s*** $\models \forall\,\phi \Leftrightarrow \forall$ ***s-path = S (S*** $\models \phi$**)**

Semantics of CTL*

path-fomula

$\phi ::= \varphi \mid \neg \phi_1 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 U \phi_2$

- If $S = s_0 s_1 s_2 s_3 s_4 \ldots\ldots\ldots, S \models \varphi \Leftrightarrow M, s_0 \models \varphi$
- $S \models \neg \phi_1 \Leftrightarrow S \models \phi_1$ 是***false***
- $S \models \phi_1 \vee \phi_2 \Leftrightarrow S \models \phi_1$ **or** $S \models \phi_1$
- $S \models \bigcirc \phi \Leftrightarrow S^{(1)} \models \phi$
- $S \models \phi_1 U \phi_2 \Leftrightarrow \exists k \geq 0 \, (S^{(k)} \models \phi_2 \wedge \forall 0 \leq j < k (S^{(j)} \models \phi_1))$

Models

- Kripke Structure
- Timed Automata (TA)
- Communicating Timed Automata (CTA)

Kripke Structure

- A Kripke structure M over a set of atomic propositions, AP, is a four tuple $\mathrm{M} = (S, S_0, R, L)$ where
  - S is a finite set of states.
  - $S_0 \subseteq S$ is the set of initial states.
  - $R \subseteq S \times S$ is a transition relation that must be total, that is $\forall s \in S. \exists s'. R(s, s')$
  - $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

Example of Kripke Structure

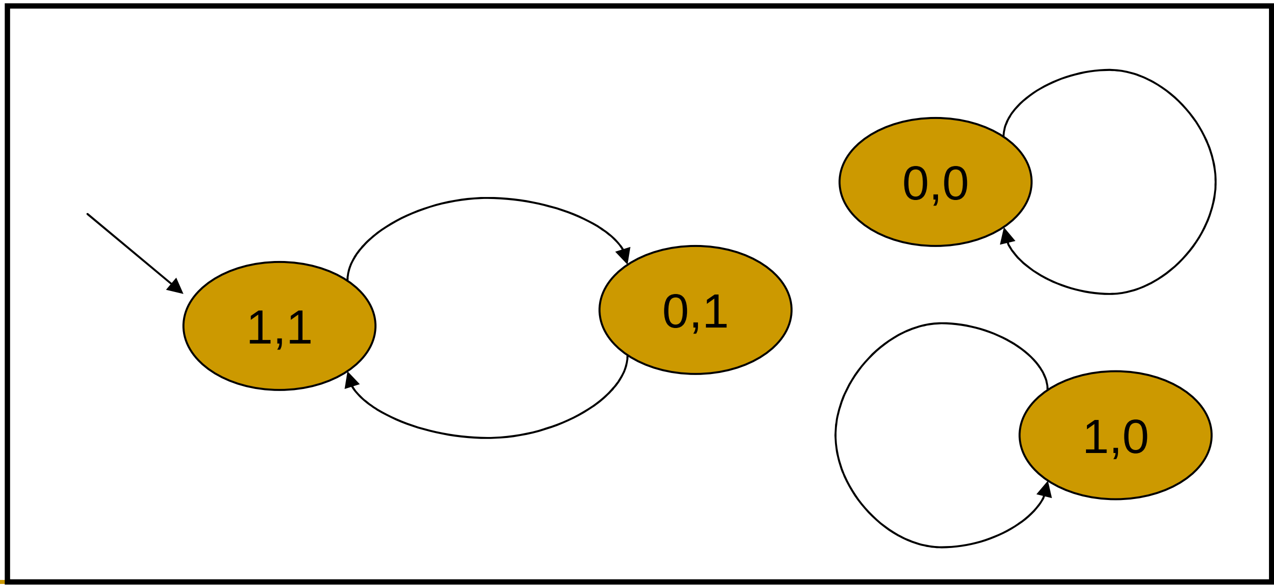
Suppose there is a program

```
initially x=1 and y=1;
while true do
x:=(x+y) mod 2;
endwhile
```

where x and y range over $D=\{0,1\}$

Example of Kripke Structure

- $S = D\text{x}D$
- $S_0=\{(1,1)\}$
- $R=\{((1,1),(0,1)),((0,1),(1,1)),((1,0),(1,0)),((0,0),(0,0))\}$
- L((1,1))={x=1,y=1},L((0,1))={x=0,y=1},L((1,0))={x=1,y=0},L((0,0))={x=0,y=0}



Timed Automata

$\boldsymbol{A} = \langle Q, q_0, P, X, \mu, E, \tau, \pi \rangle$

Q : set of control locations

q_0 : initial location

P : set of propositions

X : set of clock variables

$\mu : Q \rightarrow \Gamma(P,X)$; invariant

$E \subseteq Q \times Q$: set of transitions

$\tau : E \rightarrow \Gamma(P, X)$; triggering condition

$\pi : E \rightarrow 2^X$: set of clocks to be reset

Example of Timed Automata

- Suppose we will lunch a missile which will aim at enemy and fix its direction every 50ms until the missile hits the target in 500ms.

Example of Timed Automata

$A=\langle Q, q_0, P, X, \mu, E, \tau, \pi\rangle$

$Q=\{aim，hit\}$

$E=\{(aim,aim),(aim,hit)\}$

$q_0=aim$

$\tau\,(aim,aim)= z=50$

$P=\{\,\}, \quad X=\{x, y\}$

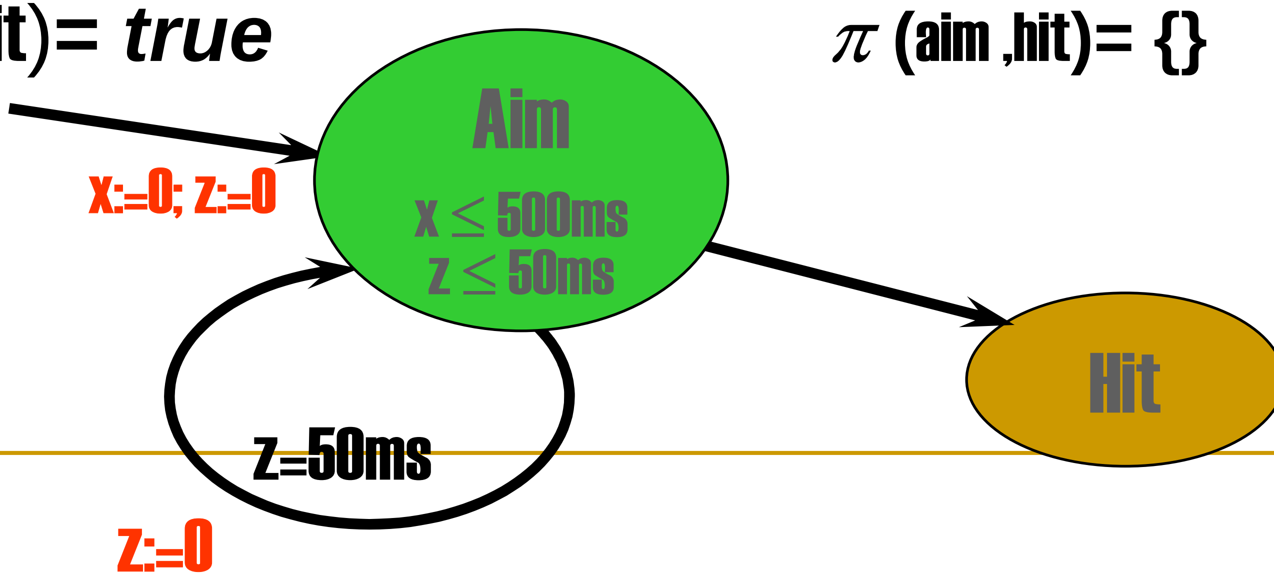
$\tau\,(aim,hit)= true$

$\mu\,(aim)= x\le 500 \wedge z\le 50$

$\pi\,(aim,aim)=\{z\}$

$\mu\,(hit)= true$

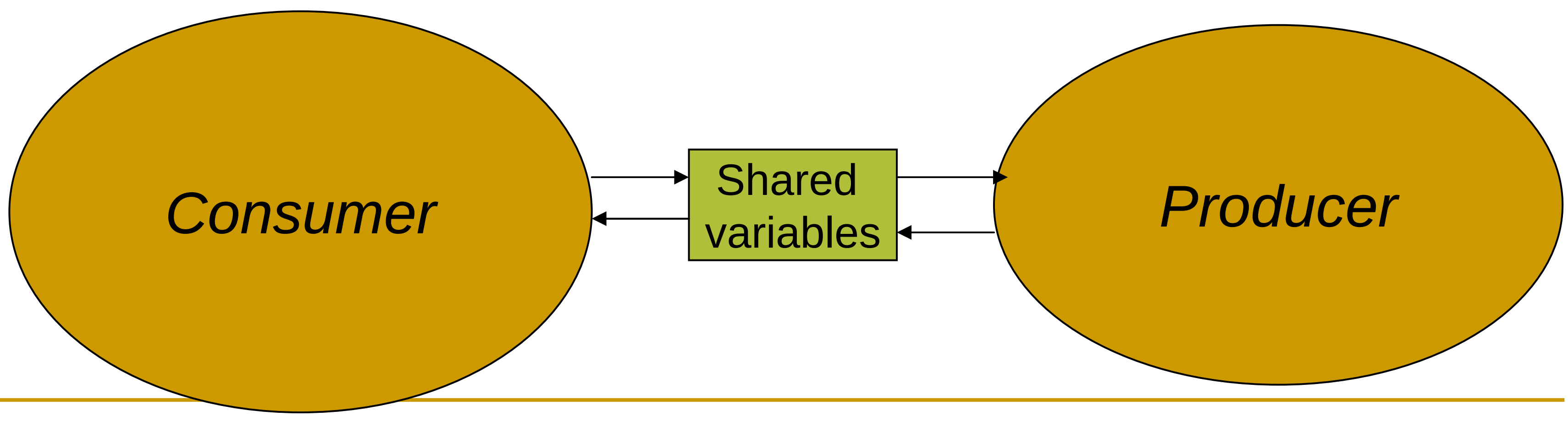
$\pi\,(aim,hit)=\{\}$



Communicating TA (CTA)

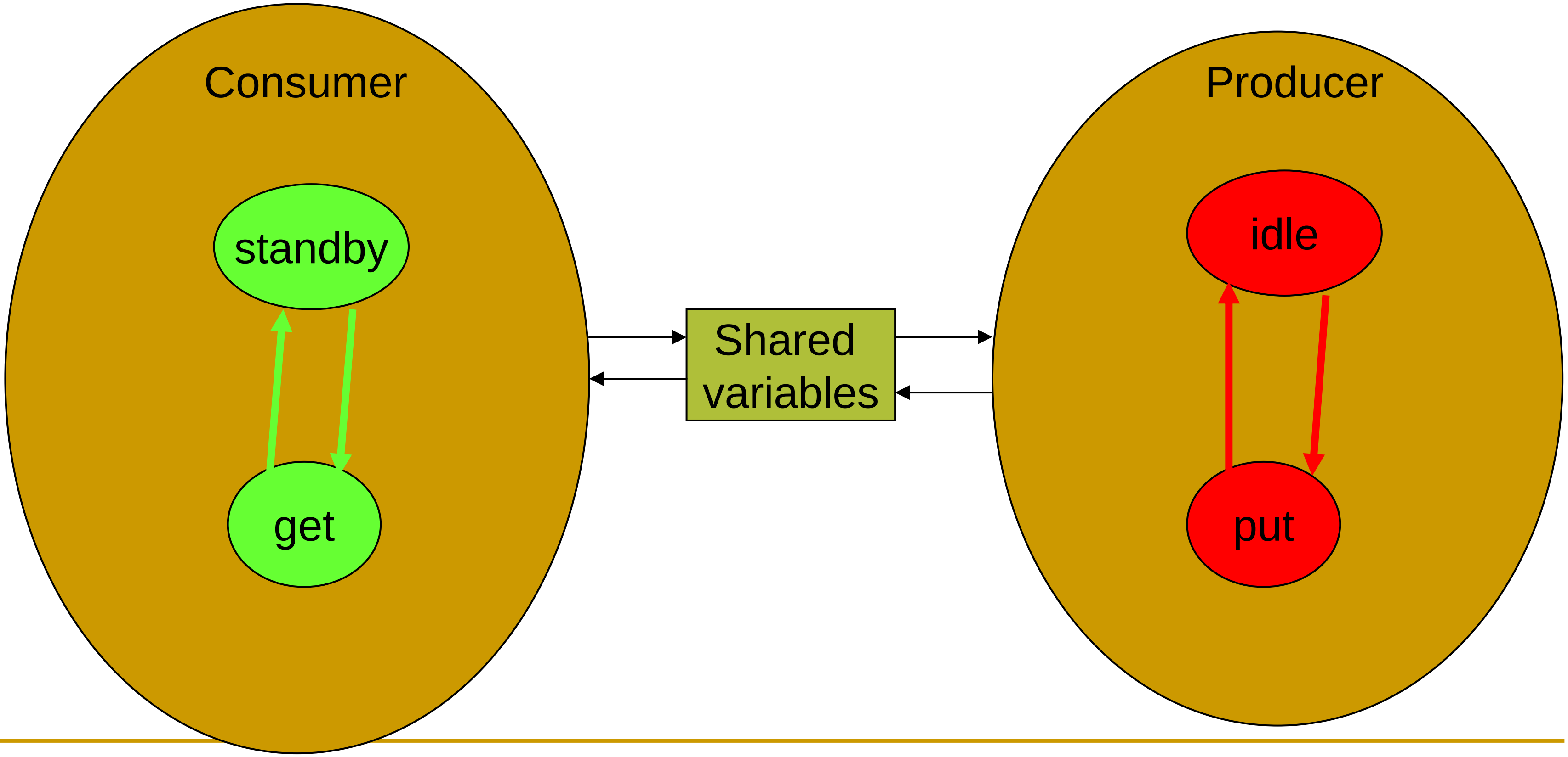
A set of TAs, communicate each other by synchronizers and shared variables

What is a legal concurrent computing？



Communicating TA (CTA)

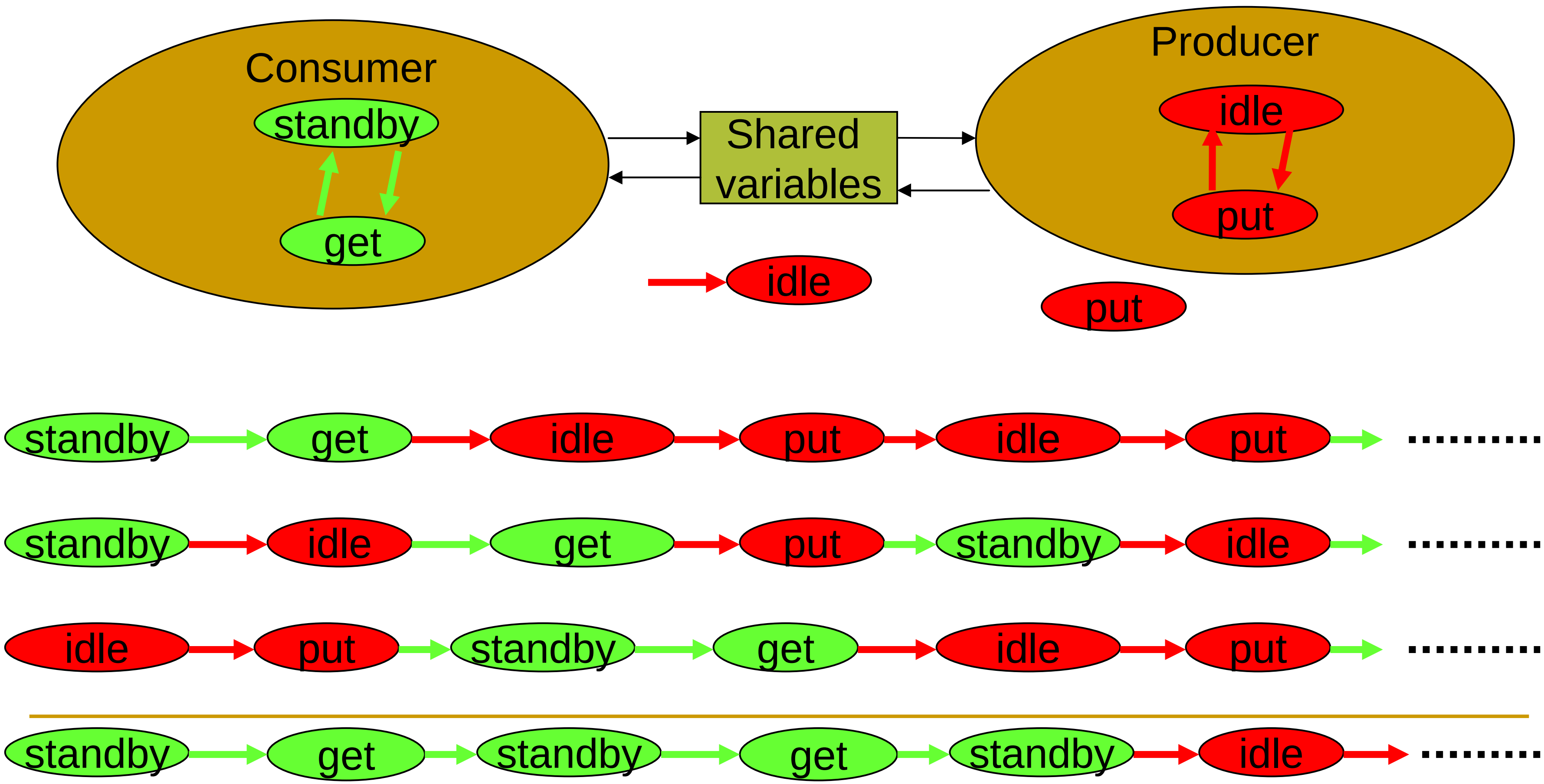
What is a legal concurrent computing？



Communicating TA (CTA)

Interleaving Semantics

Atomic, instantaneous, non-overlapping

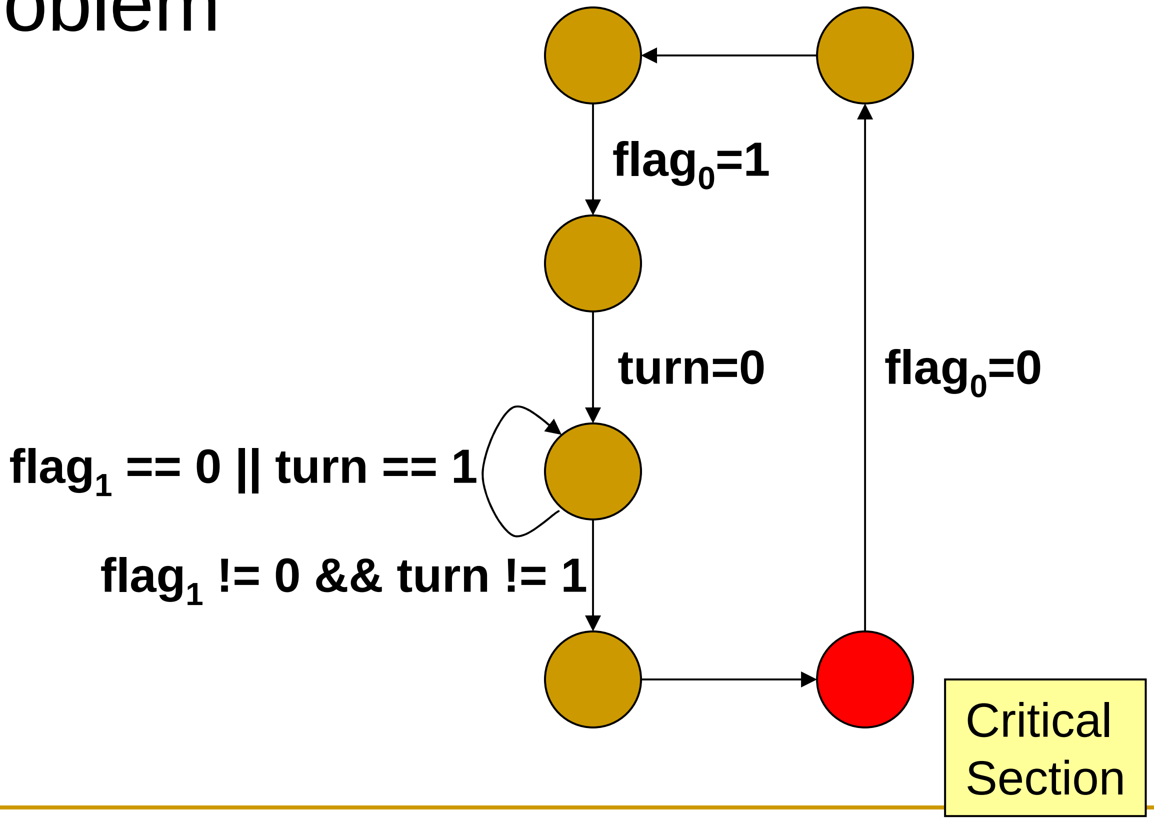


Model checking Examples

- A SPIN example for mutual exclusion problem
- A RED example for fisher’s timed mutual exclusion algorithm

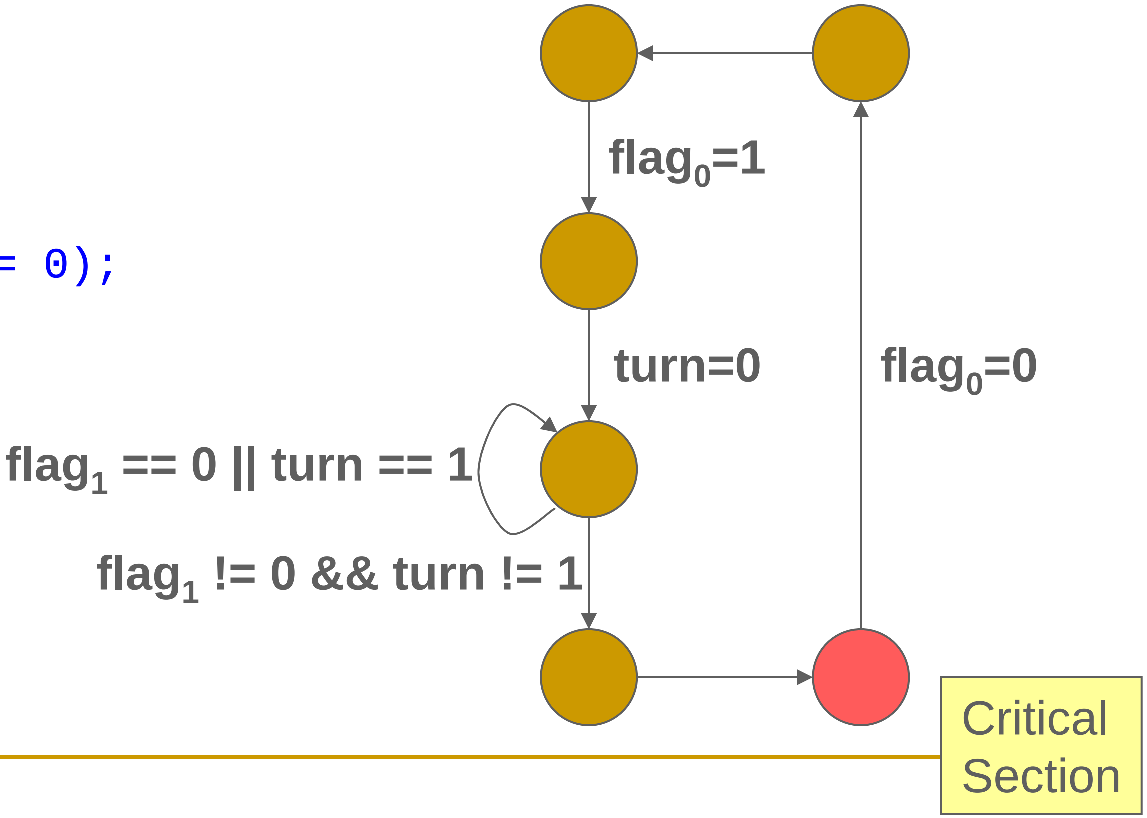
Mutual Exclusion

- Peterson's solution to the mutual exclusion problem



Mutual Exclusion in SPIN

```
bool turn;
bool flag[2];
proctype mutex0() {
again:
  flag[0] = 1;
  turn = 0;
  (flag[1] == 0 || turn == 0);
  /* critical section */
  flag[0] = 0;
  goto again;
}
```



Mutual Exclusion in SPIN

```
bool turn, flag[2];

active [2] proctype user()
{
  assert(_pid == 0 || __pid == 1);
again:

= 0 || turn == 1 - _pid);

        /* critical section */

  flag[_pid] = 0;
  goto again;
}
```

Active process:
automatically creates instances of processes

_pid:
Identifier of the process

assert:
Checks that there are only at most two instances with identifiers 0 and 1

Mutual Exclusion in SPIN

```
bool turn, flag[2];
byte ncrit;

active [2] proctype user()
{
  assert(_pid == 0 || __pid == 1);
again:
  flag[_pid] = 1;
  turn = _pid;
  (flag[1 - _pid] == 0 || turn == 1 - _pid);

  ncrit++;
  assert(ncrit == 1); /* critical section */
  ncrit--;

  flag[_pid] = 0;
  goto again;
}
```

ncrit:
Counts the number of
Process in the critical section

assert:
Checks that there are always
at most one process in the
critical section

Mutual Exclusion in SPIN

```
bool turn, flag[2];
bool critical[2];

active [2] proctype user()
{
  assert(_pid == 0 || __pid == 1);
again:
  flag[_pid] = 1;
  turn = _pid;
  (flag[1 - _pid] == 0 || turn == 1 - _pid);

  critical[_pid] = 1;
  /* critical section */
  critical[_pid] = 0;

  flag[_pid] = 0;
  goto again;
}
```

LTL Properties:

```
[] (critial[0] || critical[1])

[] <> (critical[0])
[] <> (critical[1])

[] (critical[0] ->
  (critial[0] U
    (!critical[0] &&
      ((!critical[0] &&
        !critical[1]) U critical[1]))))
[] (critical[1] ->
  (critial[1] U
    (!critical[1] &&
      ((!critical[1] &&
        !critical[0]) U critical[0]))))
```

CTA Examples

Fischer's timed mutual exclusion

A pointer variable *lock* initially NULL;

A clock variable *x* for each process;

Each process {

Initially at idle state;

idle: *lock* == NULL → *x*=0; goto ready;

ready: if *x*<= A → *x*=0; *lock* = P; goto waiting;

waiting: if *x*> B && *lock* == P → goto critical;

if *lock* != P → goto idle;

critical: *lock* = NULL; goto idle;

}

CTA: Fischer's timed mutual exclusion algorithm

Why mutual exclusion when $A<B$?

Consider a naive wrong locking algorithm.

```
While (true), do {
    while (lock != NULL) ;
    lock = P;
}
```

Atomic operations: lock != NULL ------- (test)

lock = P; ------- (assignment)

CTA: Fischer's timed mutual exclusion algorithm

Why mutual exclusion when A<B?

How can this naïve algorithm be wrong?

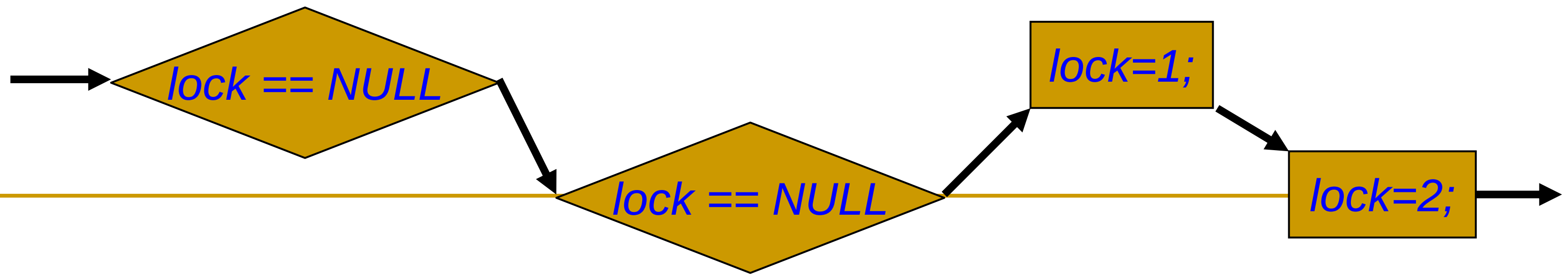
```
While (true), do {
   while (lock != NULL) ;
   lock = P;
   /* critical section */
}
```

Interleaving can happen and mess up.

- *distributed computing*
- *scheduling policy*

P1:

P2:



CTA: Fischer's timed mutual exclusion algorithm

Why mutual exclusion when $A<B$?

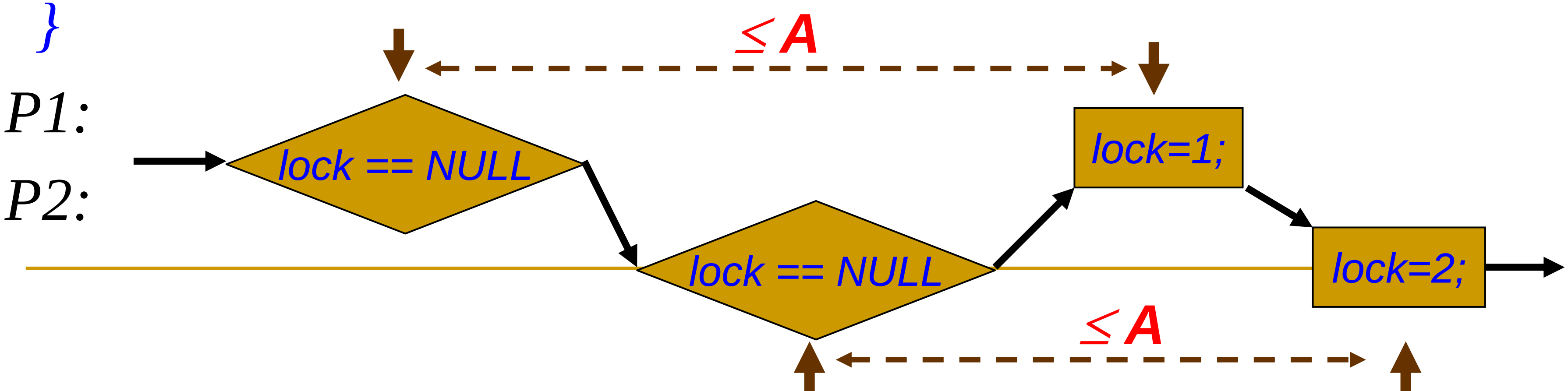
How can this naïve algorithm be wrong?

```
While (true), do {
    while (lock != NULL) ;
    lock = P;
    /* critical section */
}
```

But, assuming no scheduling mess-up,

how can lock=P be postponed indefinitely

in a concurrent system ?



CTA: Fischer's timed mutual exclusion algorithm

Why mutual exclusion when A<B?

Remedy to the naïve algorithm:

While (true), do {

while (lock != NULL) ;

/ in between, take at most A sec. */*

lock = P;

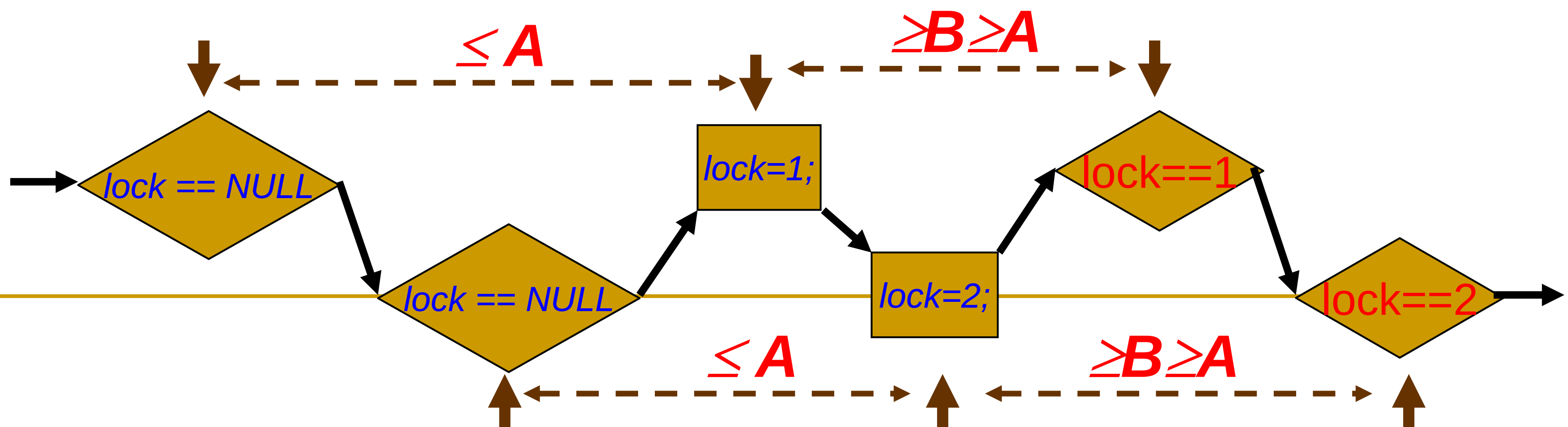
wait for B > A sec to enter critical section if lock = P still.

/ critical section */*

}

P1:

P2:



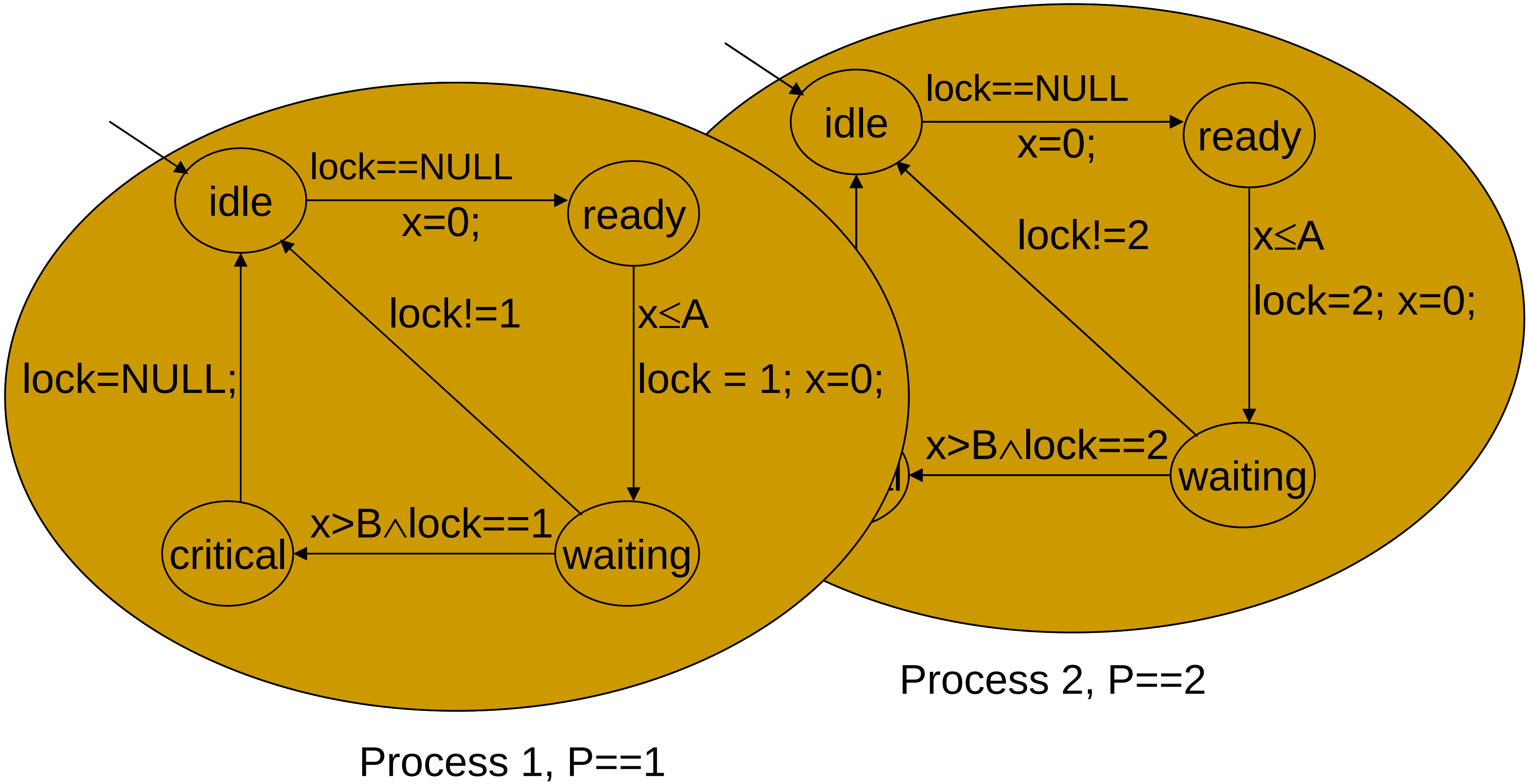
CTA: Fischer's timed mutual exclusion algorithm

Why mutual exclusion when A<B?

- *Assumption: all clocks advance their dense readings at the same rate.*

1. When a process is in waiting, no more process can enter ready.
2. A process test the lock in waiting only when all ready processes have entered waiting, since A < B.

CTA: Fischer's timed mutual exclusion algorithm



CTA: Fischer's algorithm in red format

```
/* Fischer's protocol with 2 processes */
process count = 2;
global pointer lock;
local clock x;
mode idle true {
  when lock == NULL may x= 0; goto ready;
}
mode ready true {
  when x < 10 may x= 0; lock= P; goto waiting;
}
mode waiting true {
  when (x > 19 and lock == P) may goto critical;
  when lock != P may goto idle;
}
mode critical true {
  when true may lock = NULL; goto idle;
}
initially lock == NULL and forall pi, (idle[pi] and x[pi] == 0);
risk critical[1] and critical[2];
```