Interleaving for Program Graph (PG)

. . . for modeling parallel systems with
subprocesses communicating via shared variables

Interleaving for PG

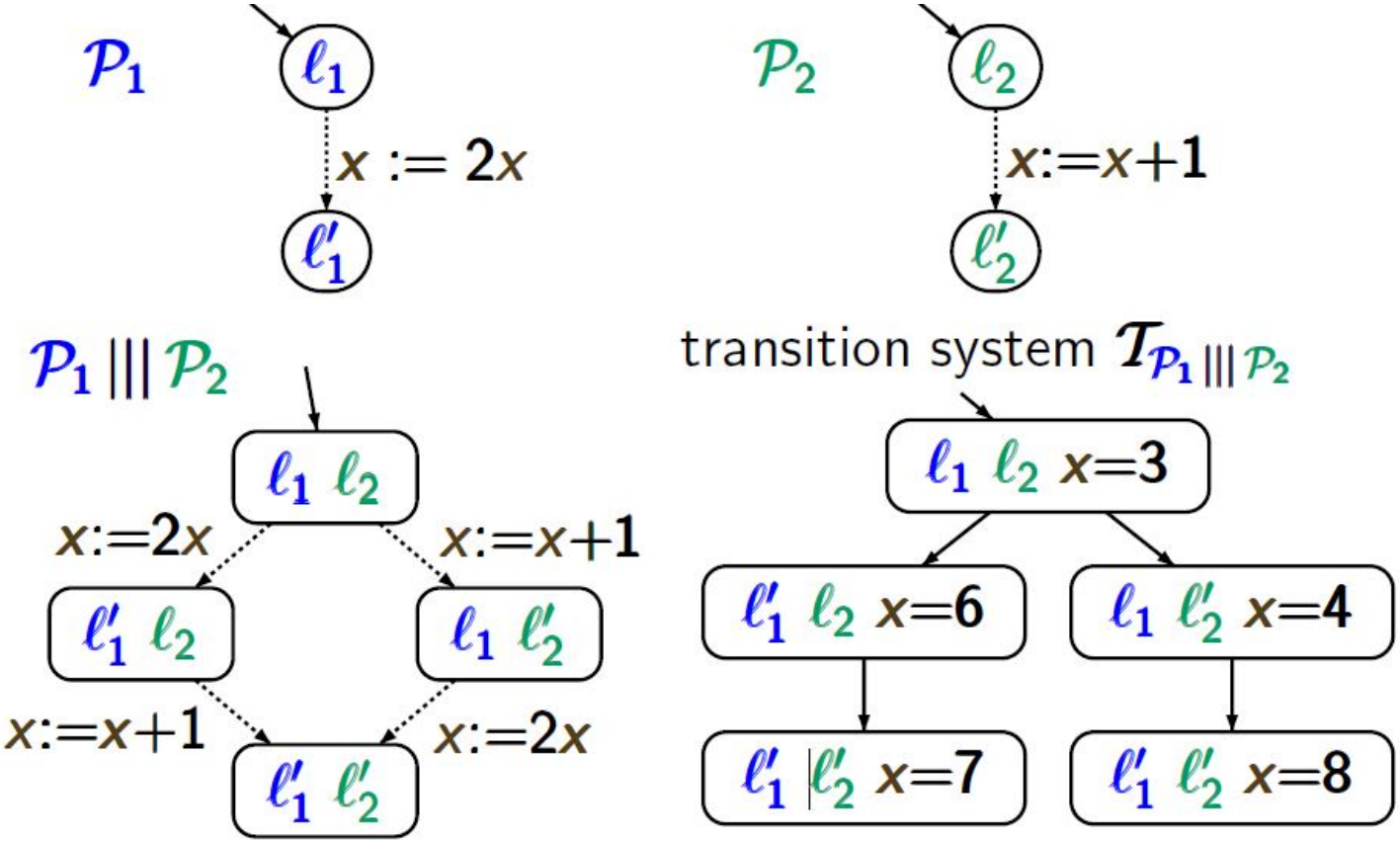
program graph $\mathcal{P}_1$ $(Loc_1, \ldots, \hookrightarrow_1, \ldots)$	program graph $\mathcal{P}_2$ $(Loc_2, \ldots, \hookrightarrow_2, \ldots)$

interleaving operator

$$\mathcal{P}_1 \,|||\, \mathcal{P}_2 \;=\; (Loc_1 \times Loc_2, \ldots, \hookrightarrow, \ldots)$$

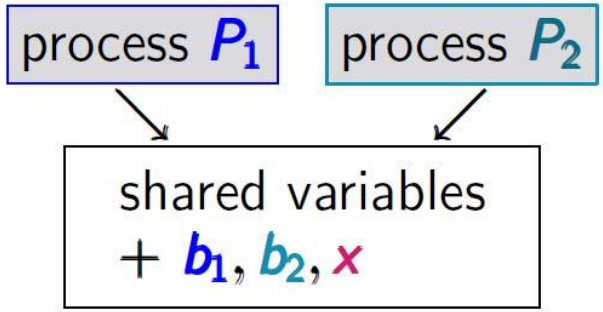
$$\frac{\ell_1 \xhookrightarrow{g\,:\,\alpha}_1 \ell_1'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g\,:\,\alpha} \langle \ell_1', \ell_2 \rangle} \qquad \frac{\ell_2 \xhookrightarrow{g\,:\,\alpha}_2 \ell_2'}{\langle \ell_1, \ell_2 \rangle \xhookrightarrow{g\,:\,\alpha} \langle \ell_1, \ell_2' \rangle}$$

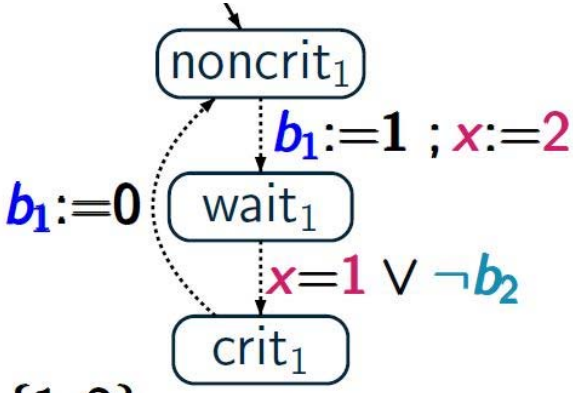
Interleaving for PG



note: $\mathcal{T}_{\mathcal{P}_1 ||| \mathcal{P}_2} \neq \mathcal{T}_{\mathcal{P}_1} ||| \mathcal{T}_{\mathcal{P}_2}$

Peterson Algorithm (PA) for mutual exclusion
Gary L Peterson 1981

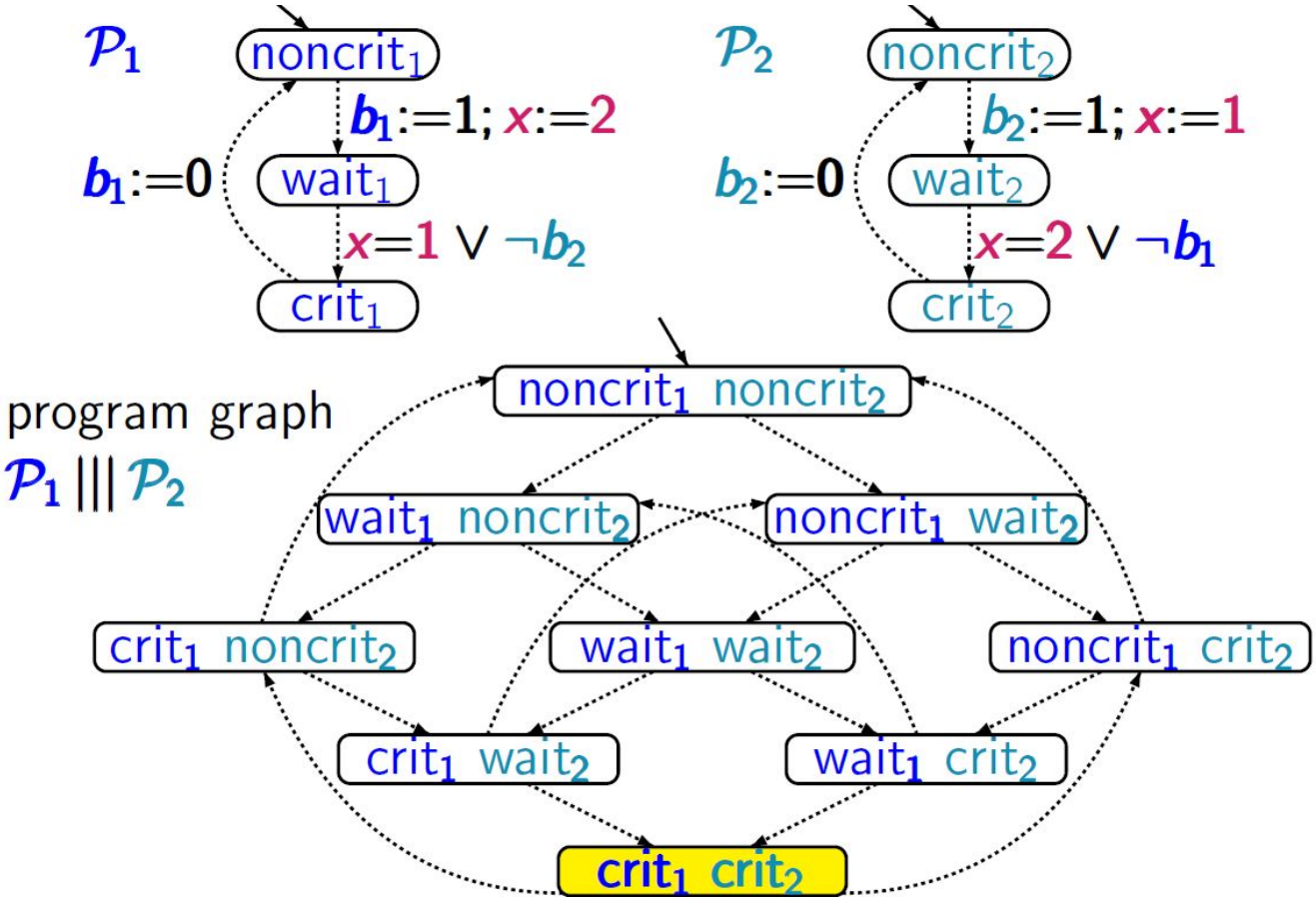




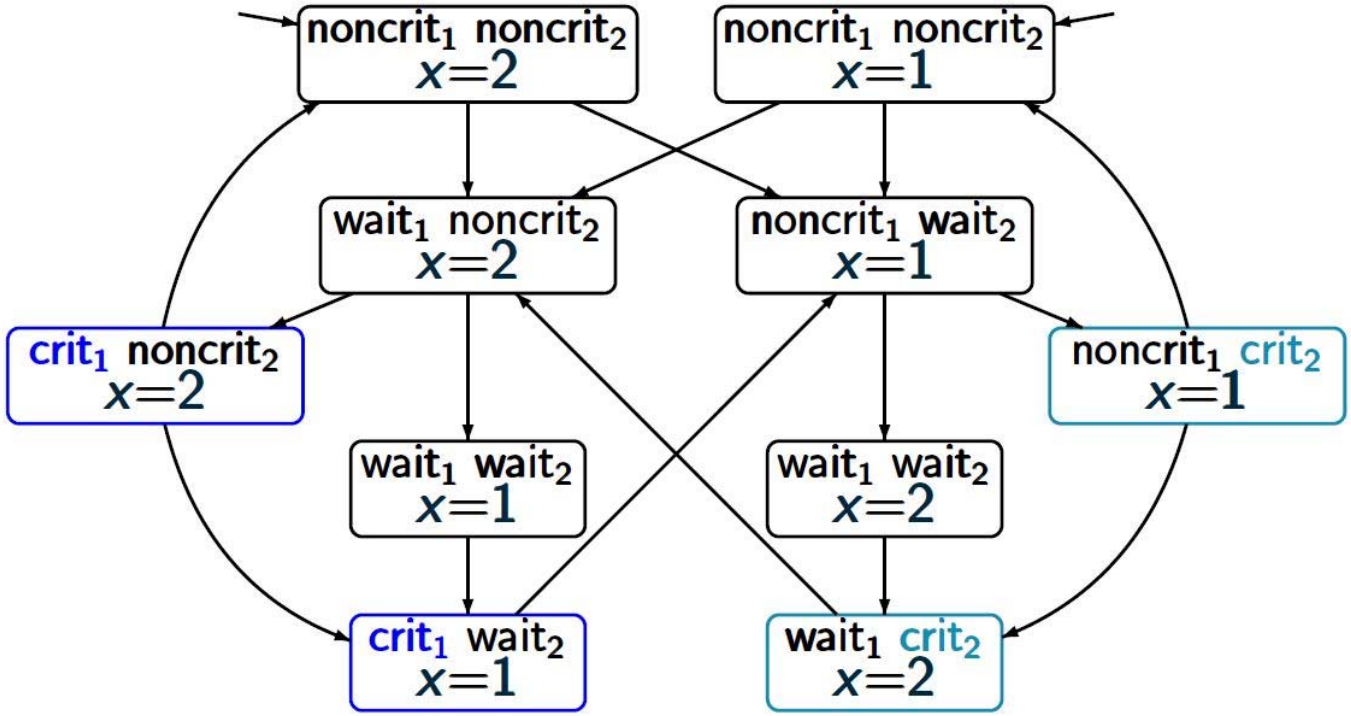
b_1, b_2 Boolean variables, $x \in \{1, 2\}$

```
LOOP  FOREVER                    (* protocol for P1 *)
    noncritical actions;
    b1:=1 ;  x:=2;
    AWAIT  x=1 ∨ ¬b2 DO  critical section OD
    b1:=0
END  LOOP
```

Program Graph for PA



Transition System for PG

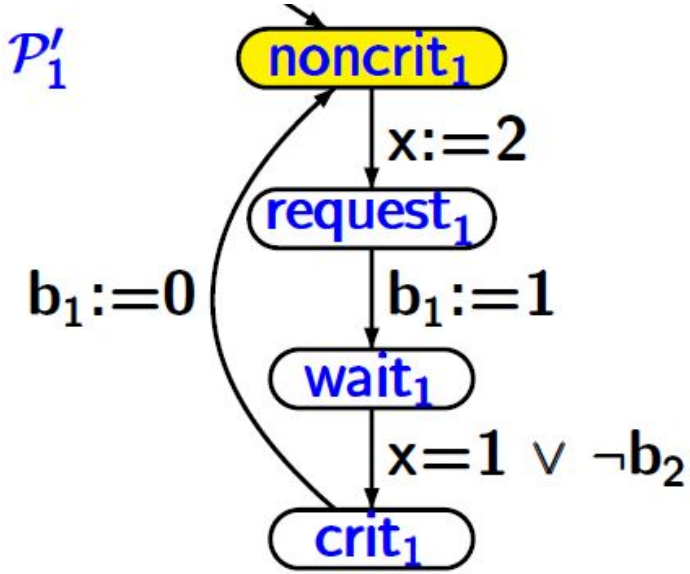


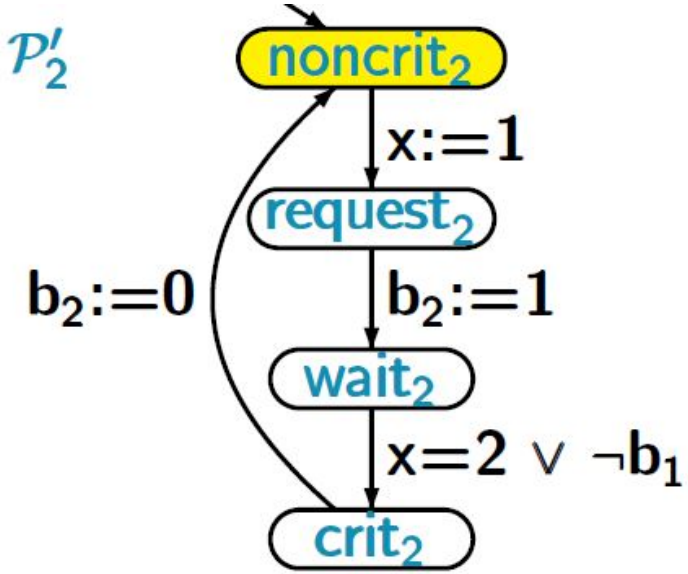
value of b_1 is given by $\mathbf{wait}_1 \vee \mathbf{crit}_1$

value of b_2 is given by $\mathbf{wait}_2 \vee \mathbf{crit}_2$

+ unreachable states

Variant of PA

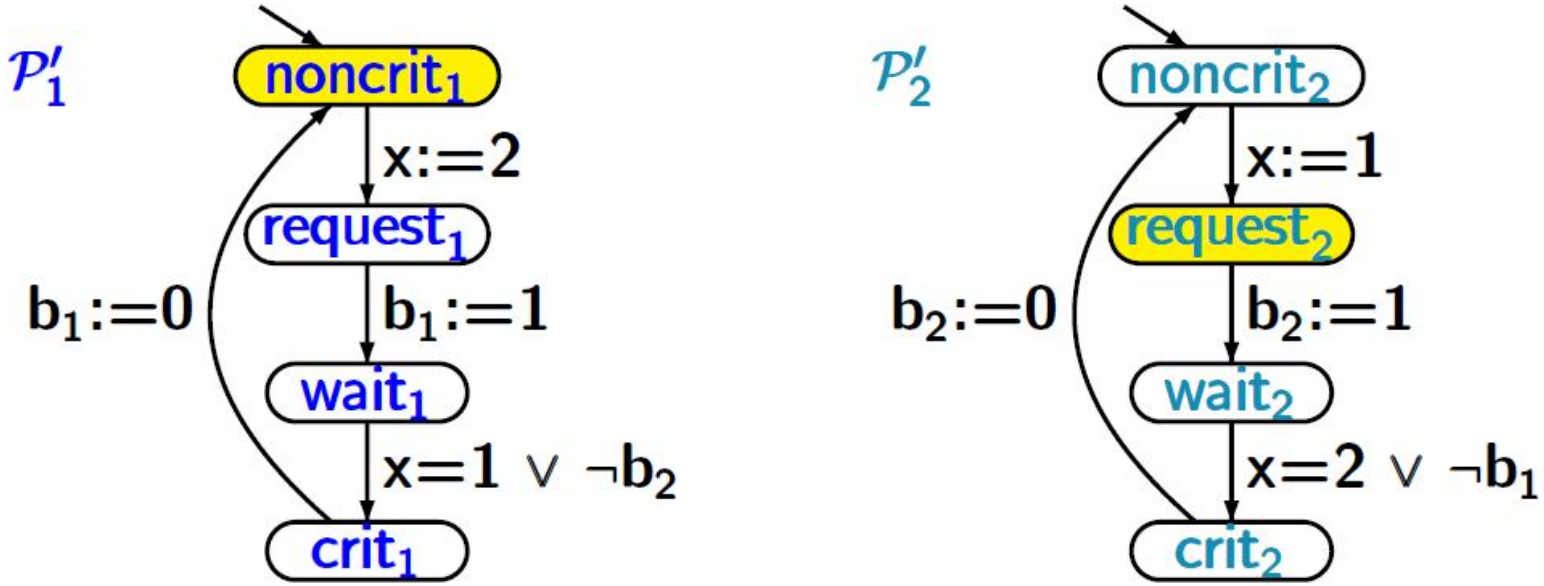




possible executions

noncrit$_1$ noncrit$_2$ x=1 ¬b$_1$ ¬b$_2$

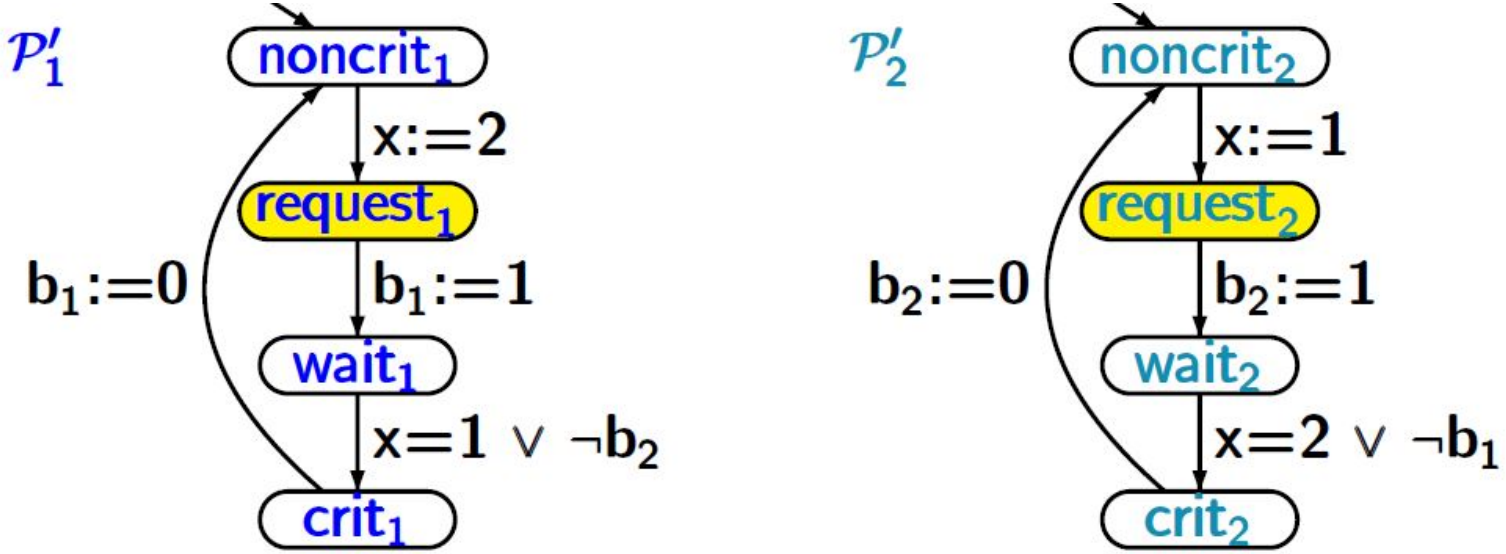
Variant of PA



possible executions

noncrit$_1$	noncrit$_2$	x=1	¬b$_1$	¬b$_2$
noncrit$_1$	request$_2$	x=1	¬ b$_1$	¬b$_2$

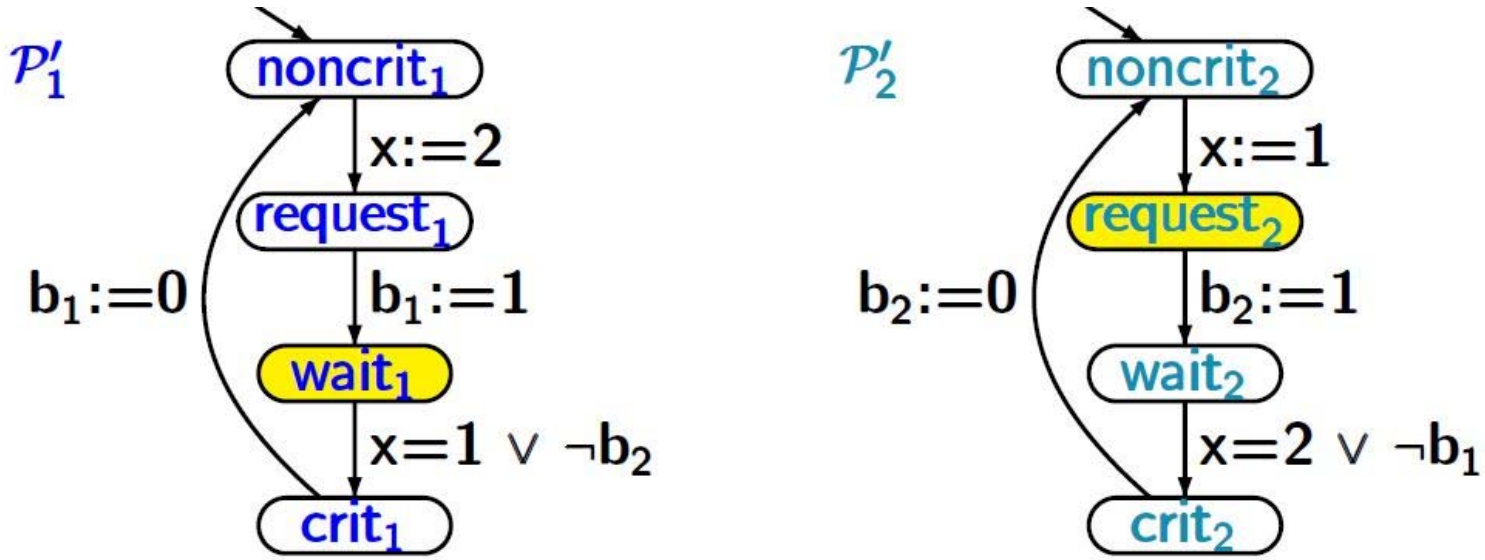
Variant of PA



possible executions

noncrit$_1$	noncrit$_2$	x=1	¬b$_1$	¬b$_2$
noncrit$_1$	request$_2$	x=1	¬ b$_1$	¬b$_2$
request$_1$	request$_2$	x=2	¬b$_1$	¬b$_2$

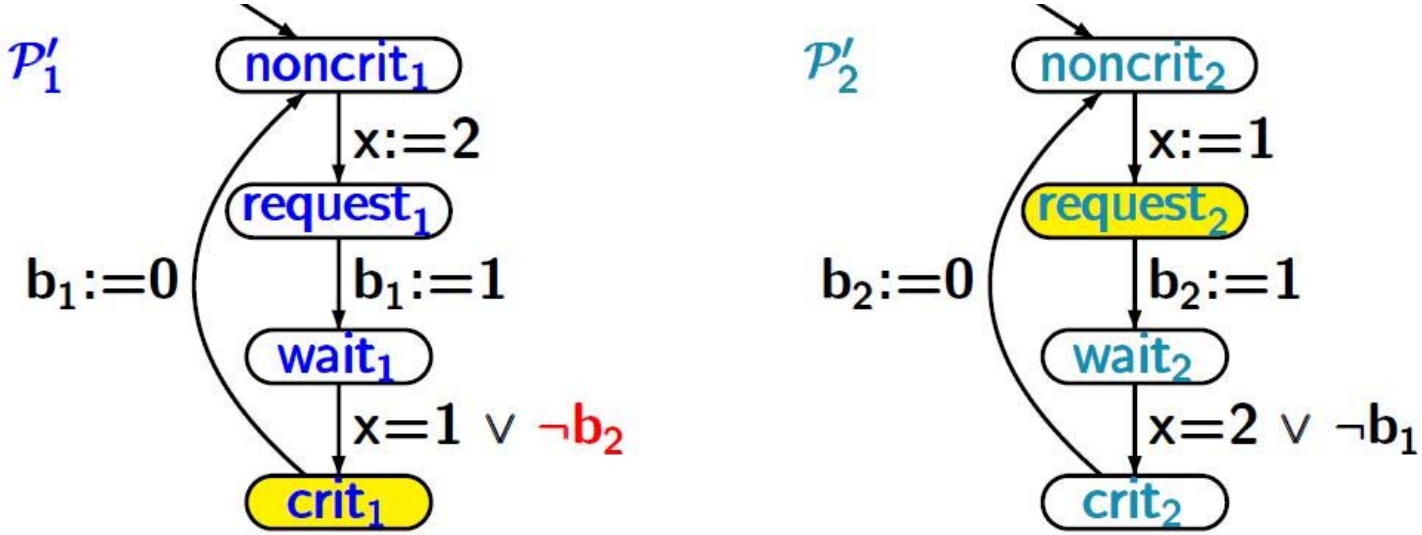
Variant of PA



possible executions

noncrit$_1$	noncrit$_2$	x=1	¬b$_1$	¬b$_2$
noncrit$_1$	request$_2$	x=1	¬ b$_1$	¬b$_2$
request$_1$	request$_2$	x=2	¬b$_1$	¬b$_2$
wait$_1$	request$_2$	x=2	b$_1$	¬b$_2$

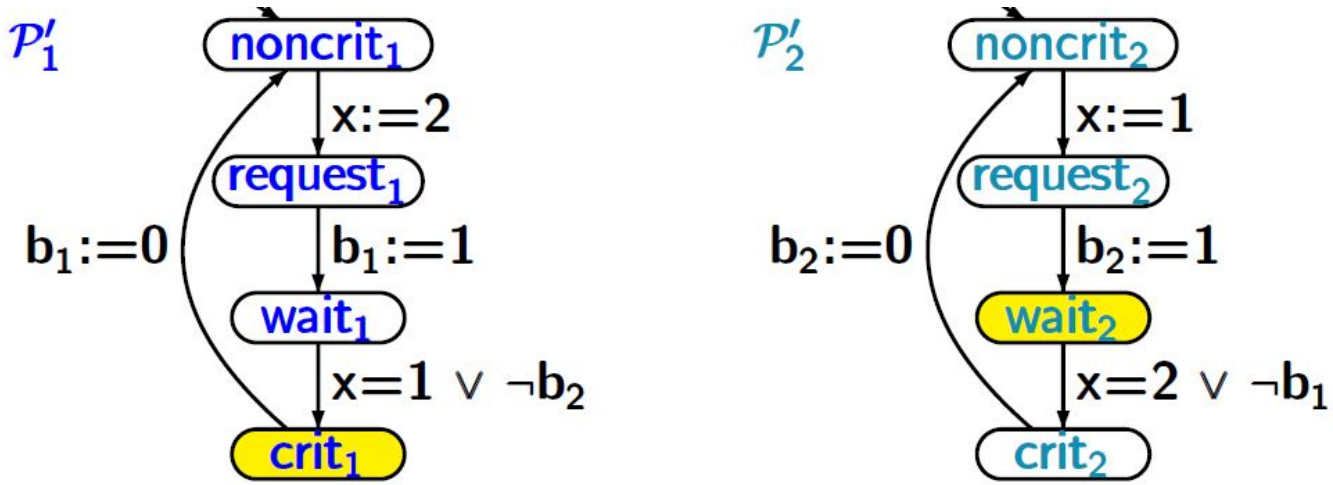
Variant of PA



possible executions

noncrit$_1$	noncrit$_2$	x=1	¬b$_1$	¬b$_2$
noncrit$_1$	request$_2$	x=1	¬ b$_1$	¬b$_2$
request$_1$	request$_2$	x=2	¬b$_1$	¬b$_2$
wait$_1$	request$_2$	x=2	b$_1$	¬b$_2$
crit$_1$	request$_2$	x=2	b$_1$	¬b$_2$

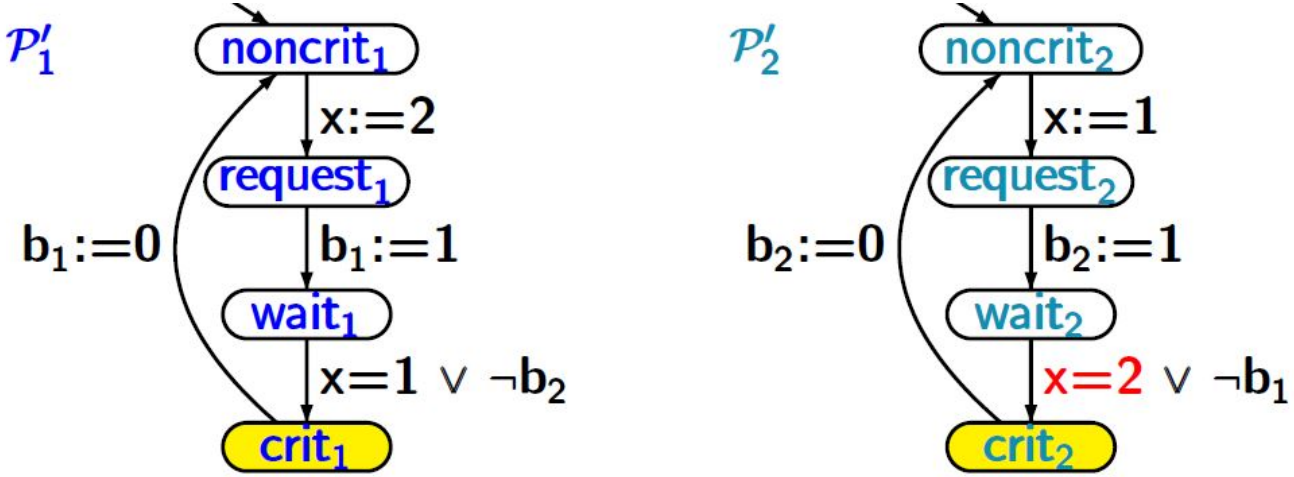
Variant of PA



possible executions

noncrit$_1$	noncrit$_2$	x=1	¬b$_1$	¬b$_2$
noncrit$_1$	request$_2$	x=1	¬ b$_1$	¬b$_2$
request$_1$	request$_2$	x=2	¬b$_1$	¬b$_2$
wait$_1$	request$_2$	x=2	b$_1$	¬b$_2$
crit$_1$	request$_2$	x=2	b$_1$	¬b$_2$
crit$_1$	wait$_2$	x=2	b$_1$	b$_2$

Variant of PA incorrect!



possible executions

noncrit$_1$	noncrit$_2$	x=1	¬b$_1$	¬b$_2$
noncrit$_1$	request$_2$	x=1	¬ b$_1$	¬b$_2$
request$_1$	request$_2$	x=2	¬b$_1$	¬b$_2$
wait$_1$	request$_2$	x=2	b$_1$	¬b$_2$
crit$_1$	request$_2$	x=2	b$_1$	¬b$_2$
crit$_1$	wait$_2$	x=2	b$_1$	b$_2$
crit$_1$	crit$_2$	x=2	b$_1$	b$_2$

Operators for parallelism and communication

- true concurrency: interleaving operator $|||$ for TS
  (no communication, no dependencies)
- communication via shared variables
  - description of subsystems by program graphs
  - interleaving $|||$ for program graphs
  - TS is obtained by "unfolding"
- synchronous message passing $\longleftarrow$ data abstract
  - operator $\|_{Syn}$ for TS
  - interleaving for independent actions
  - synchronization over actions in Syn
- channel systems
  communication via shared variables + via channels

Synchronous message passing

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

composite transition system:

$$\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$$

for modeling the concurrent execution of $\mathcal{T}_1$ and $\mathcal{T}_2$ with synchronization over all actions in Syn

Synchronous message passing

$\mathcal{T}_1 = (S_1, Act_1, \rightarrow_1, \ldots)$, $\mathcal{T}_2 = (S_2, Act_2, \rightarrow_2, \ldots)$ TS

$Syn \subseteq Act_1 \cap Act_2$ set of synchronization actions

composite transition system:

$$\mathcal{T}_1 \parallel_{Syn} \mathcal{T}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \ldots)$$

interleaving for all actions $\alpha \in Act_i \setminus Syn$:

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

handshaking (rendezvous) for all $\alpha \in Syn$:

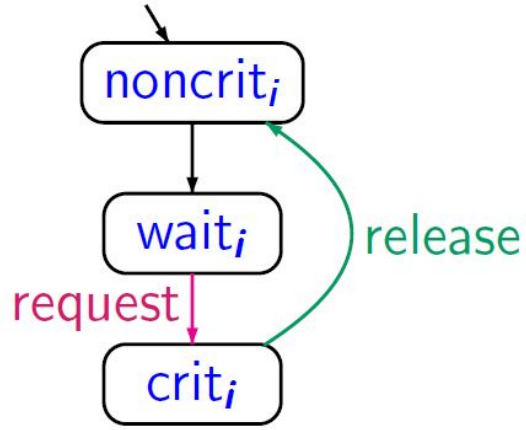
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \quad \wedge \quad s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle}$$

Mutual exclusion with an arbiter (1/3): synchronous message passing using an arbiter

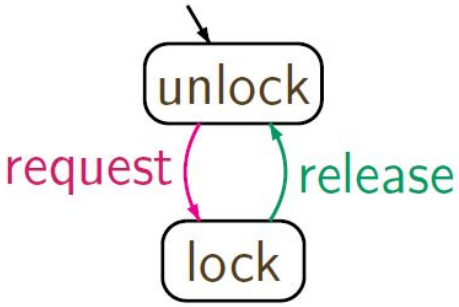
protocol for process P_i

```
LOOP FOREVER DO
    noncritical actions
    request
    critical section
    release
    noncritical actions
OD
```

transition system $\mathcal{T}_i$



Arbiter:
selects nondeterministically
a synchronization partner
$\mathcal{T}_1$ or $\mathcal{T}_2$



Mutual exclusion with an arbiter (2/3)

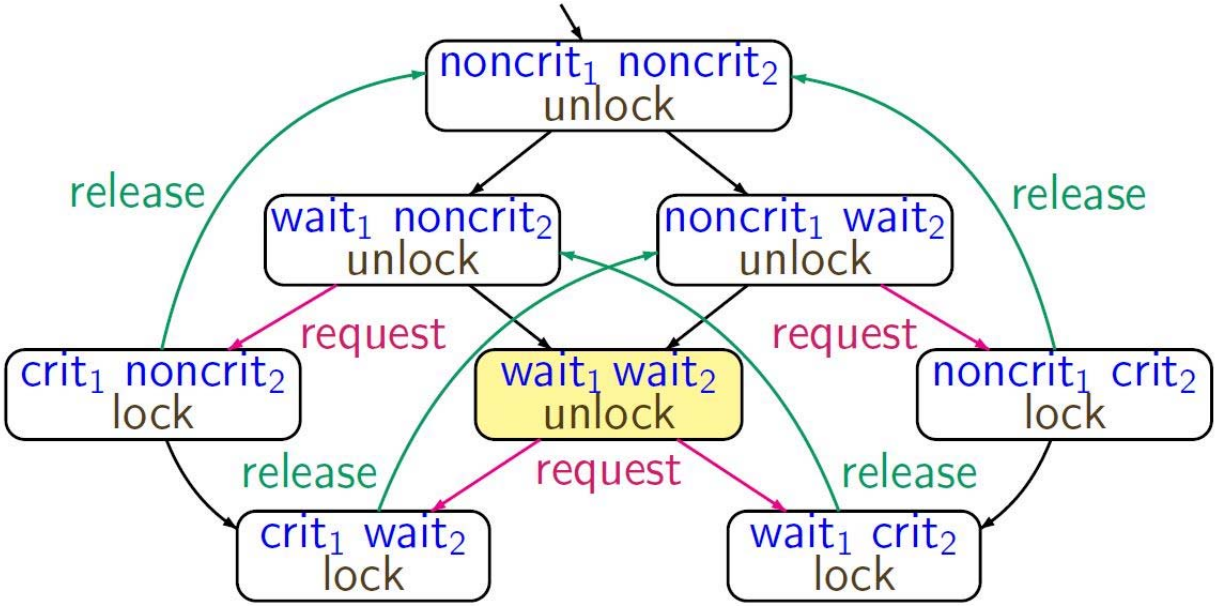
$$(\mathcal{T}_1 \ |||\ \mathcal{T}_2) \ \|_{Syn} \ \textit{Arbiter} \text{ where } Syn = \{\text{request}, \text{release}\}$$

"pure" interleaving for TS

handshaking for actions request and release

Mutual exclusion with an arbiter (3/3)

$(\mathcal{T}_1 \,|||\, \mathcal{T}_2) \parallel_{Syn}$ ***Arbiter*** where $Syn = \{\text{request}, \text{release}\}$



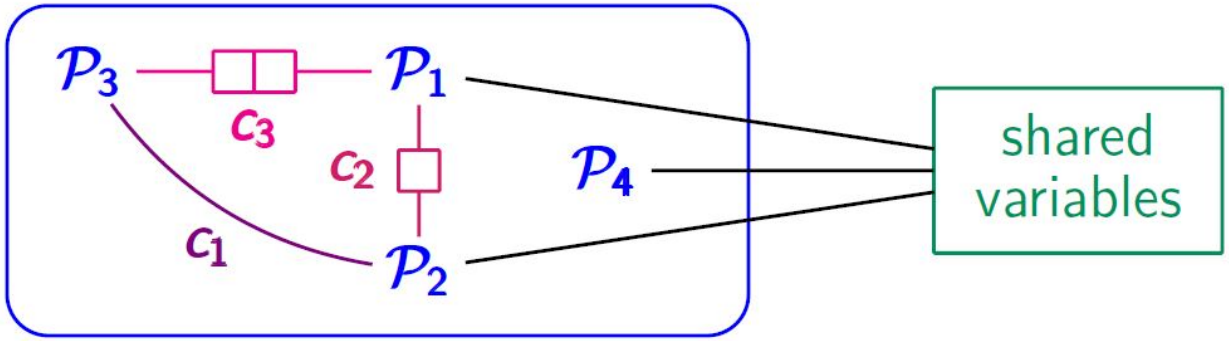
nondeterministic choice: who enters the critical section**?**

Channel System (CS) (1/4)

representation of data-dependent parallel systems with

- communication over shared variables
- synchronous message passing
- asynchronous message passing

} communication over channels



channel types: synchronous or FIFO

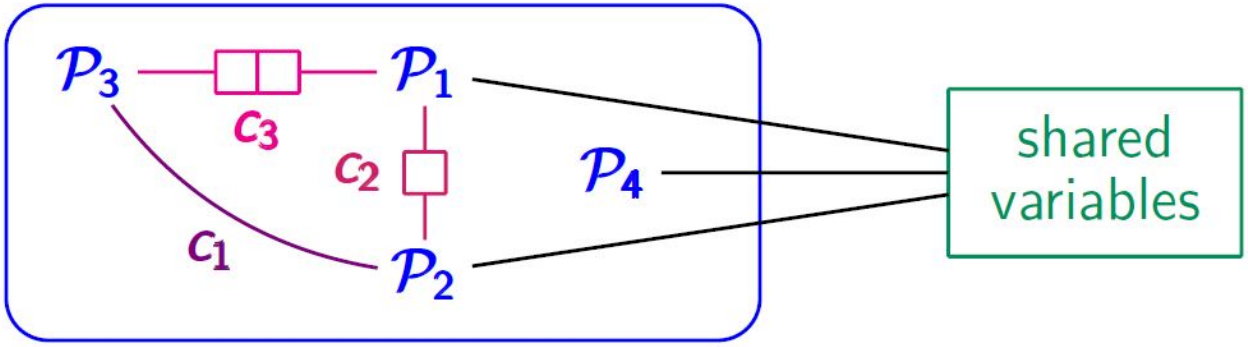
↑

capacity $\widehat{=}$ number of buffer cells

CS (2/4)

representation of data-dependent parallel systems with

- communication over shared variables
- synchronous message passing ⟵ capacity **0**
- asynchronous message passing ⟵ capacity $\geq$ **1**



channel types: synchronous or FIFO

capacity **0** ↗ ↑ capacity $\widehat{=}$ number of buffer cells

CS (3/4)

representation of data-dependent parallel systems with

- communication over shared variables
- synchronous message passing
- asynchronous message passing

} communication over channels

formalization through program graphs for $\mathcal{P}_1, \ldots, \mathcal{P}_n$

- with conditional transitions $\ell_i \xrightarrow{g:\alpha} \ell'_i$ (as before)
- and communication actions

$\ell_i \xrightarrow{c!v} \ell'_i$	sending value v via channel c
$\ell_i \xrightarrow{c?x} \ell'_i$	receiving a value for variable x via channel c

CS (4/4)

$[\mathcal{P}_1 \,|\, \mathcal{P}_2 \,|\, \ldots \,|\, \mathcal{P}_n]$ where $\mathcal{P}_i$ are program graphs over a pair $(Var, Chan)$

Var	set of typed variables
$Chan$	set of typed channels with capacities $cap(\cdot)$ and domains $Dom(\cdot)$

program graphs $\mathcal{P}_i = (Loc_i, Act_i, Effect_i, \hookrightarrow_i, Loc_{0,i}, g_0)$
with conditional transitions

$\ell \xhookrightarrow{g:\alpha}_i \ell'$ guarded command

$\ell \xhookrightarrow{c!v}_i \ell'$ sending value v via channel c

$\ell \xhookrightarrow{c?x}_i \ell'$ receiving a value for variable x via channel c

Effect of communication actions in CS

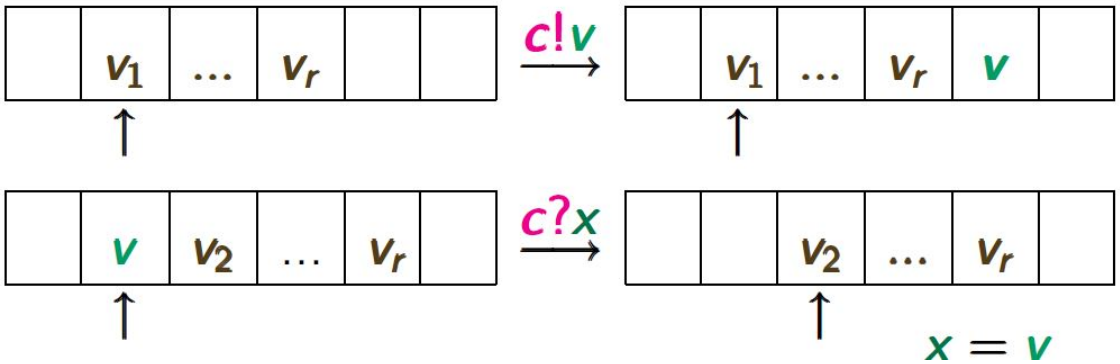
synchronous message passing via channels of capacity **0**

- $c!v$ and $c?x$ are executed at the same time
- effect $x := v$

Effect of communication actions in CS

asynchronous message passing via channels of capacity ≥ 1

	enabled if ...	effect
sending $c!v$	channel c not full	$add(c, v)$
receiving $c?x$	channel c not empty $v = front(c)$	$x := v$ $remove(c)$



TS-semantics of CS

channel system over $(Var, Chan)$

$$\mathcal{C} = [\mathcal{P}_1 | \ldots | \mathcal{P}_n]$$

$$\Downarrow$$

transition system $\mathcal{T}_{\mathcal{C}}$

states of $\mathcal{T}_{\mathcal{C}}$ have the form

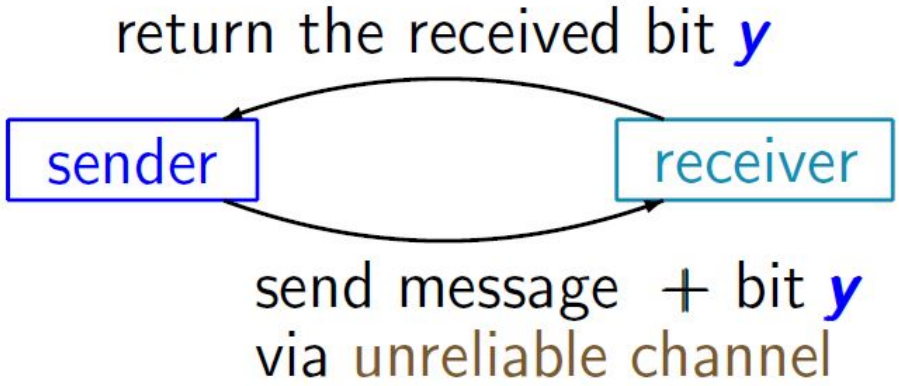
$$\langle \ell_1, \ldots, \ell_n, \eta, \xi \rangle$$

locations of $\mathcal{P}_1, \ldots, \mathcal{P}_n$

variable valuation

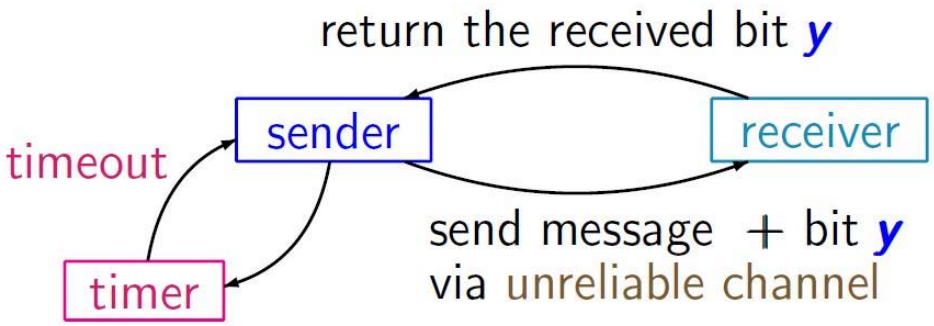
channel evaluation

Alternating bit protocol (ABP)



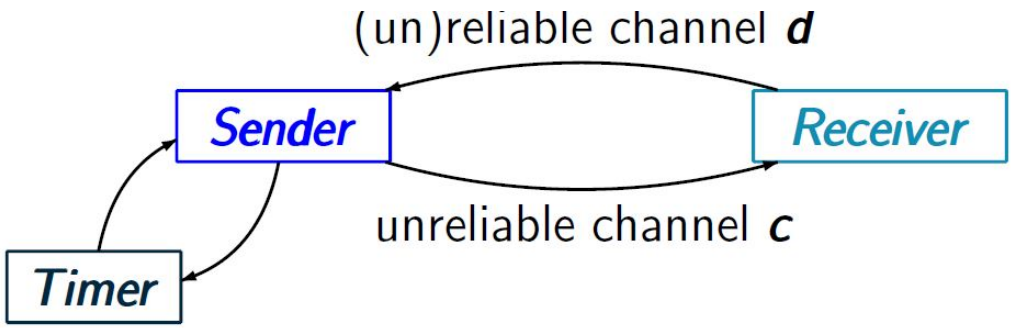
Is there any possibility that the msg gets lost?

Protocol for the sender




```
LOOP FOREVER
(1)  send message + bit y and activate timer
(2)  AWAIT timeout or acknowledgement DO
          IF timeout THEN goto (1)
                     ELSE turn off timer; y:=¬y
          FI
     OD
```

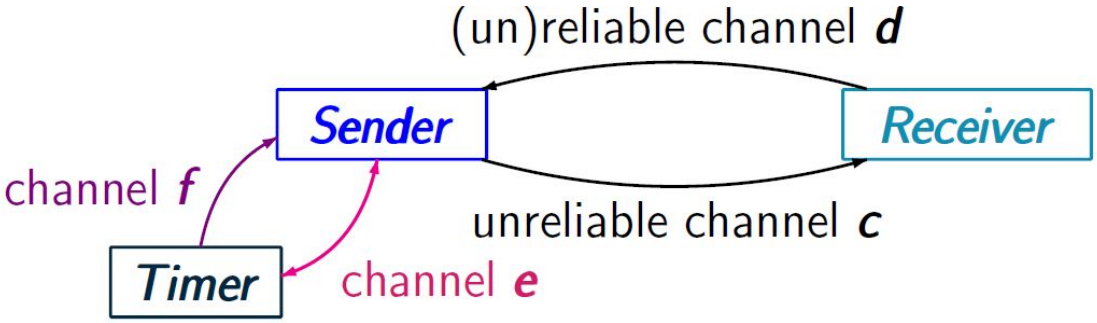
ABP



channel system: [***Sender*** | ***Timer*** | ***Receiver***]

message passing between ***Timer*** and ***Sender***
message passing between ***Receiver*** and ***Sender***

ABP

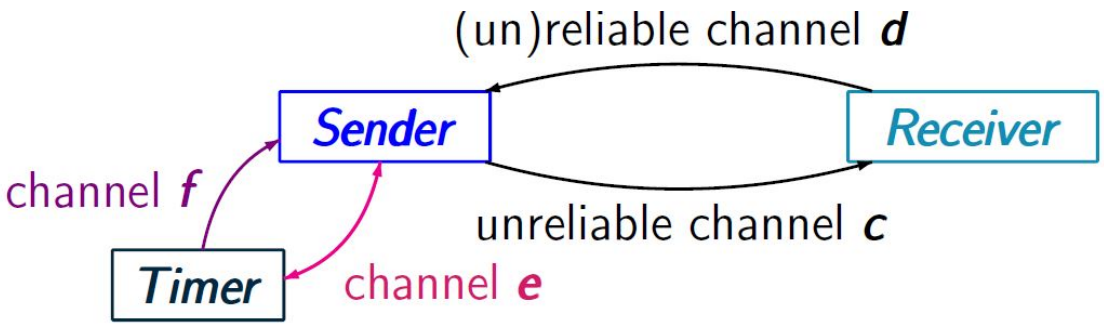


channel system: [*Sender* | *Timer* | *Receiver*]

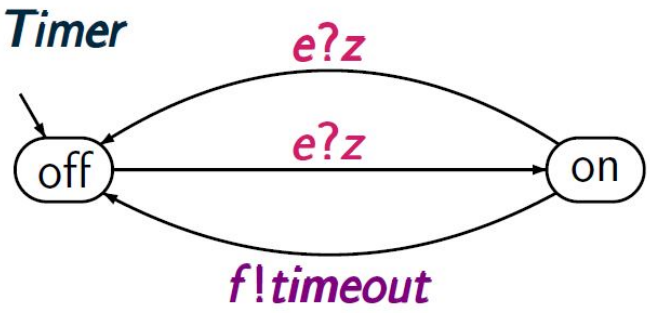
synchronous message passing between
Timer and *Sender* ← channels *e* and *f*

asynchronous message passing between
Receiver and *Sender* ← channels *c*, *d*

ABP



channel system: [*Sender* | *Timer* | *Receiver*]



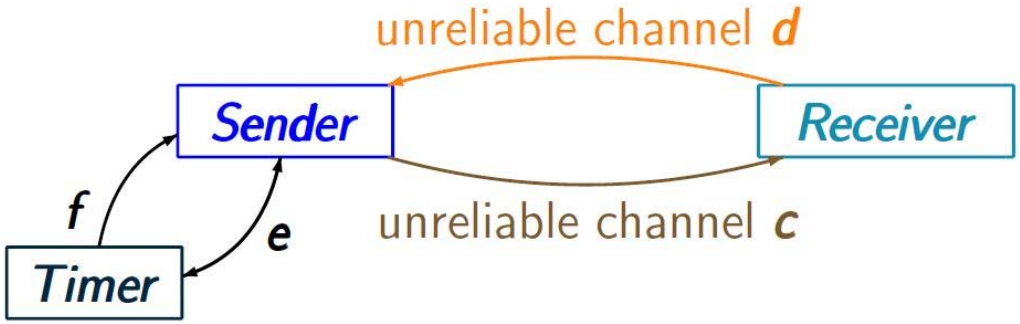
actions of *Sender*:

⋮

e!timer_on
e!timer_off
f?z′

⋮

PG for the sender (1/4)

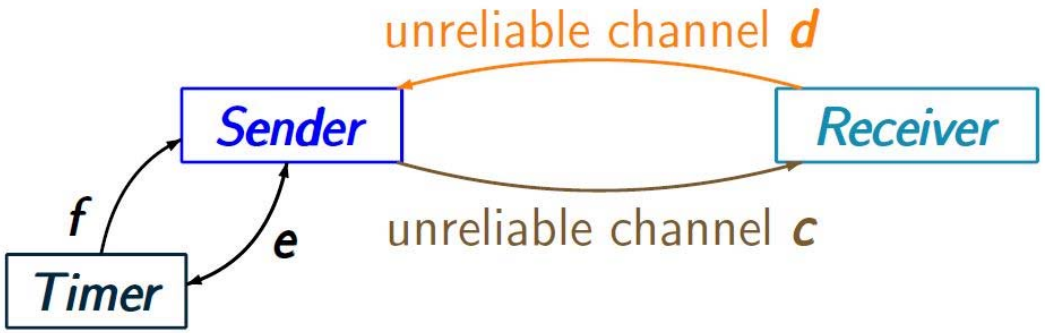


specify the sender by a program graph using

- asynchronous channels *c* and *d*
- synchronous channels *e* and *f*

simply write **!*timeout***
?*timer_on*
?*timer_off*

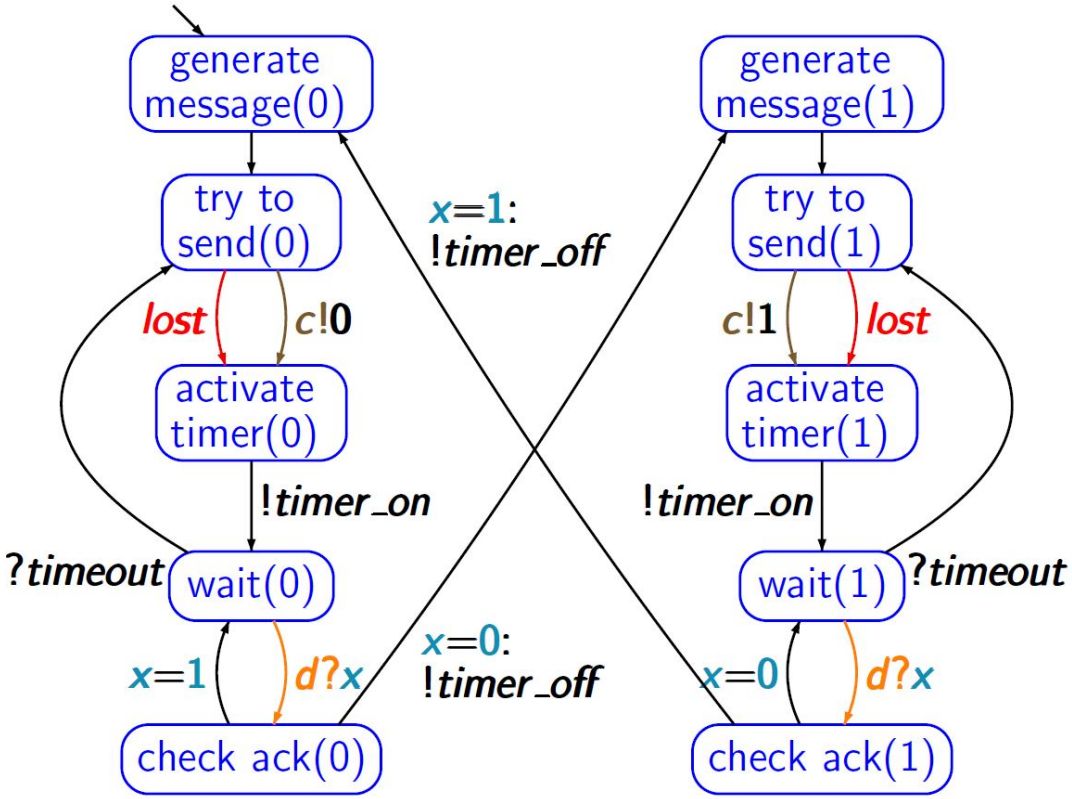
PG for the sender (2/4)



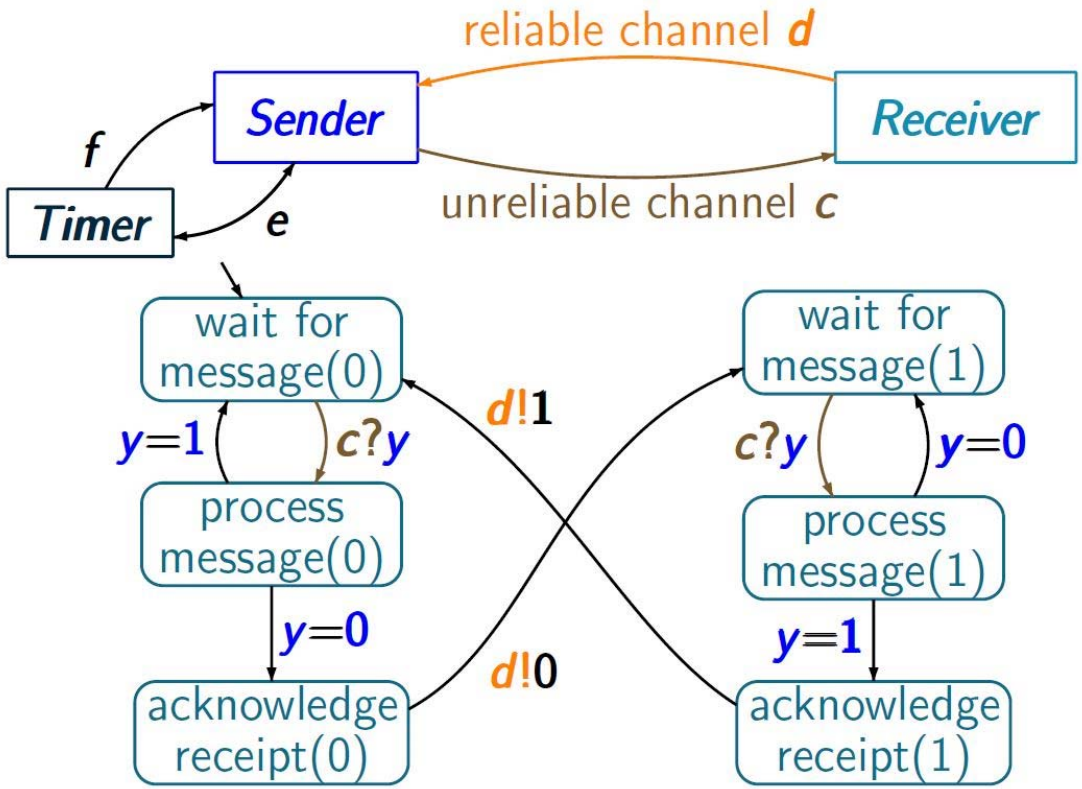
specify the sender by a program graph using

- asynchronous channels ***c*** and ***d***
- synchronous channels ***e*** and ***f***
- Boolean variable ***x*** for the acknowledgement bit sent by the receiver

PG for the sender (3/4)



PG for the receiver (4/4)



Synchronous product

for parallel systems with fully synchronized processes

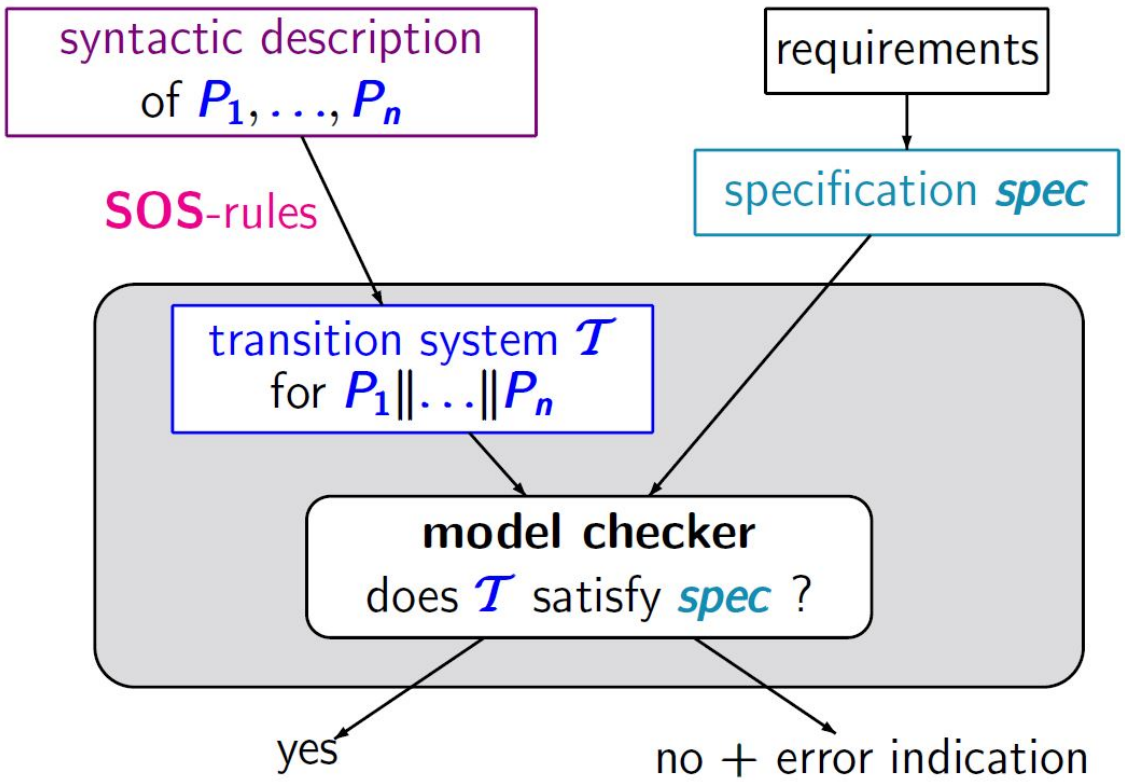
$$\left.\begin{array}{l} \mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, \ldots) \\ \mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, \ldots) \end{array}\right\} \text{ two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \;=\; (S_1 \times S_2, Act, \longrightarrow, \ldots)$$

where the action set Act is given by a function

$$Act_1 \times Act_2 \longrightarrow Act, \quad (\alpha, \beta) \mapsto \alpha * \beta$$

action name for the concurrent execution of α and β

Model checking