One of the major challenges in the design and verification of manycore systems is cache coherency. In bus-based architectures, this is a well-studied problem. When replacing the bus by a communication network, however, new problems arise. Cross-layer deadlocks can occur even when the protocol and the network are both deadlock-free when considered in isolation. To counter this problem, we propose a methodology for deriving cross-layer invariants. These invariants relate the state of the protocols run by the cores to the state of the communication network. We show how they can be used to prove the absence of cross-layer deadlocks. Our approach is generally applicable and shows promising scalability.

CROSS-LAYER INVARIANTS FOR NOCS

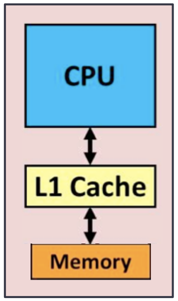
Freek Verbeek,

Pooria Mohammadi Yaghini, Ashkan Eghbal and Nader Bagherzadeh

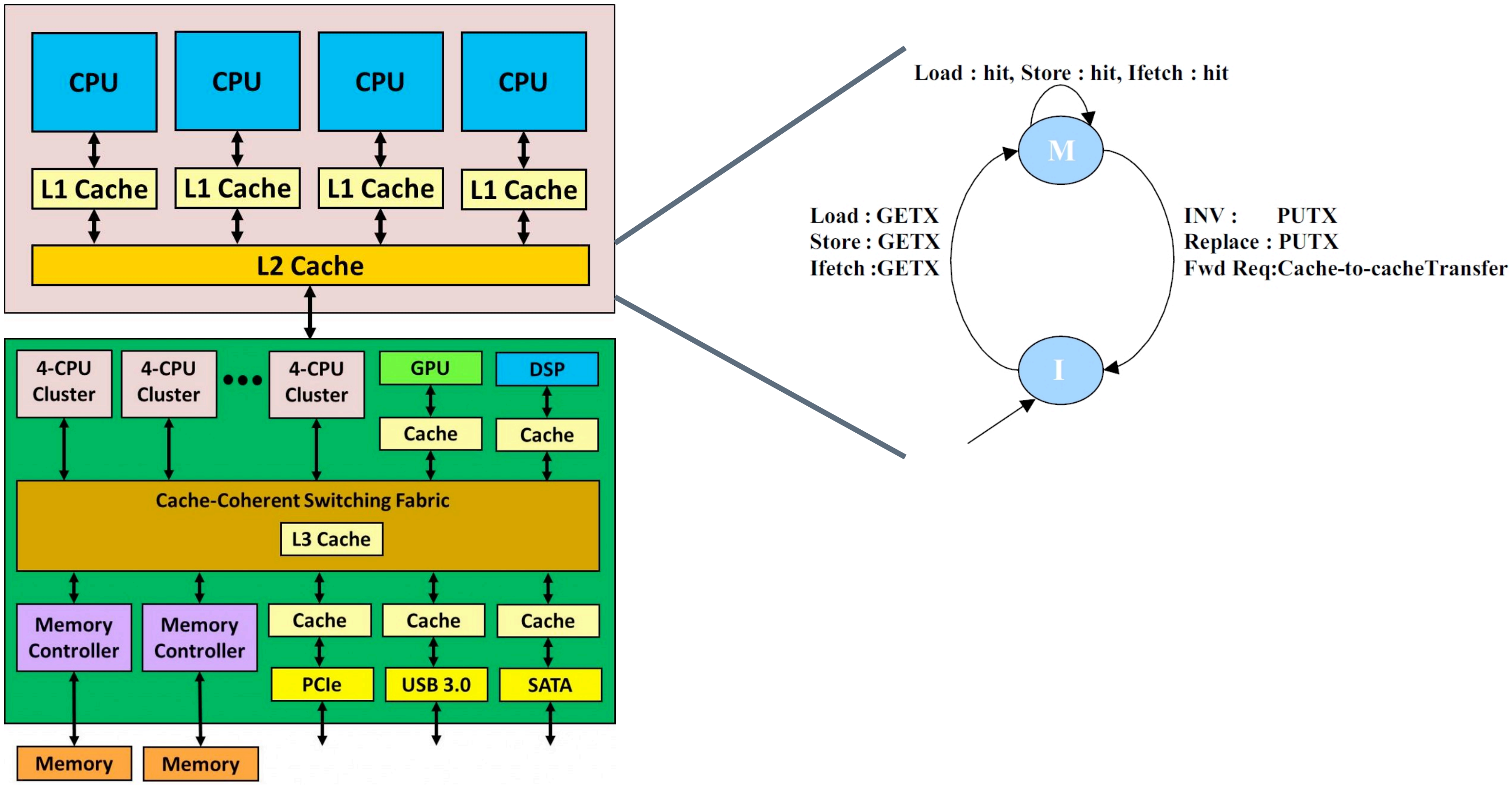
UNIVERSITY of CALIFORNIA IRVINE

Open Universiteit
www.ou.nl

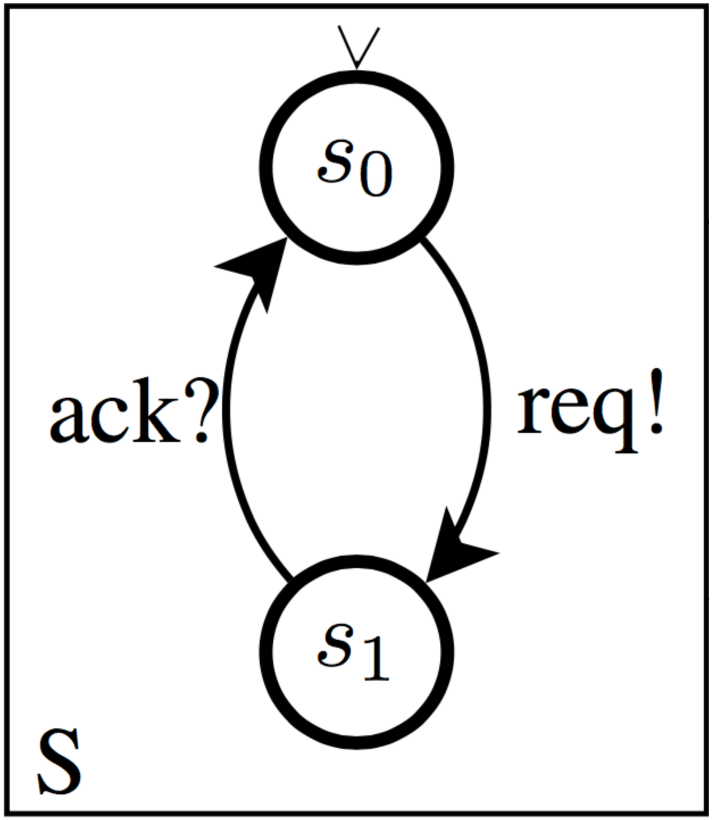
Cache Coherence & Singlecore

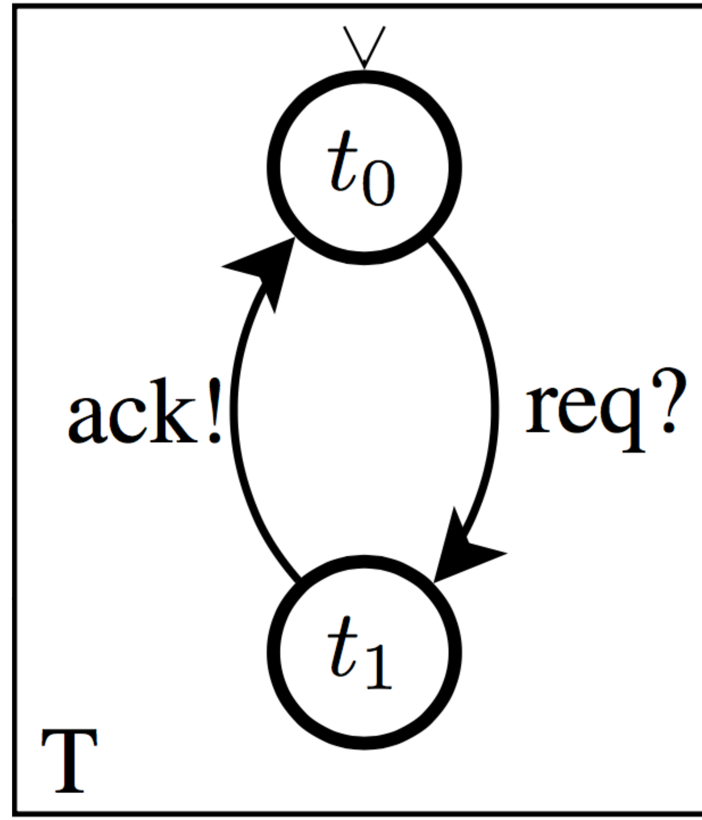


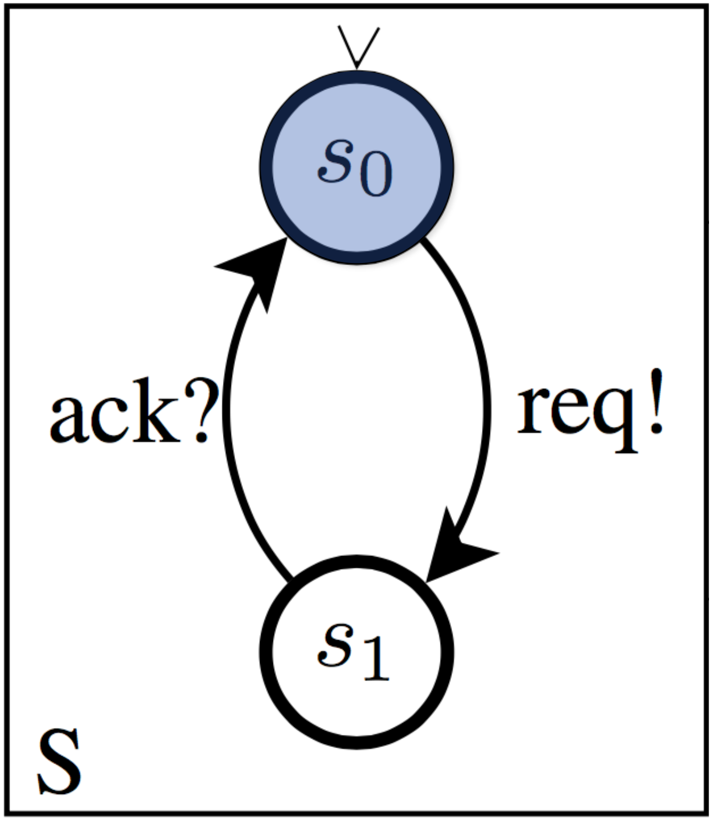
Cache Coherence & Multicore

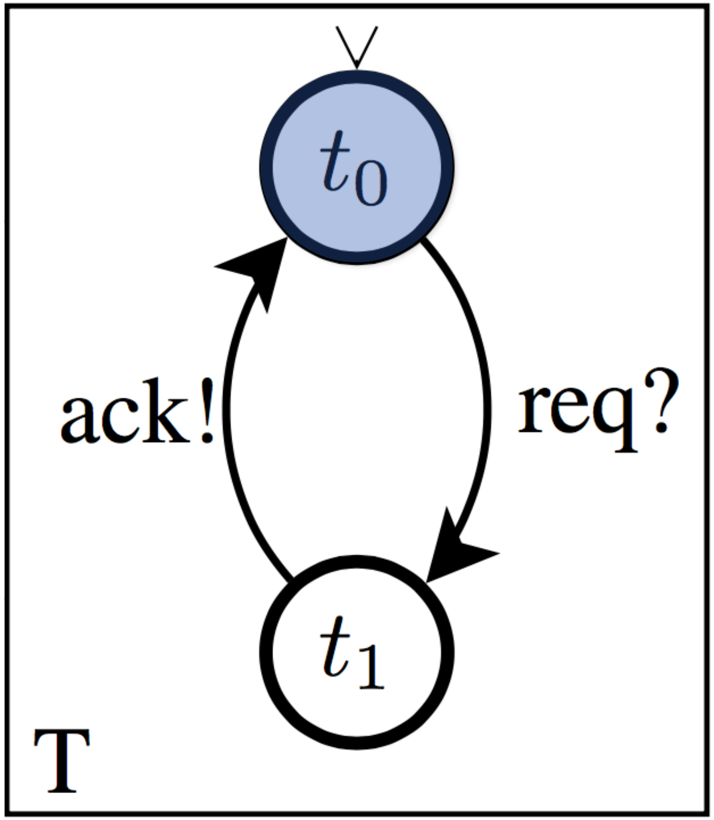


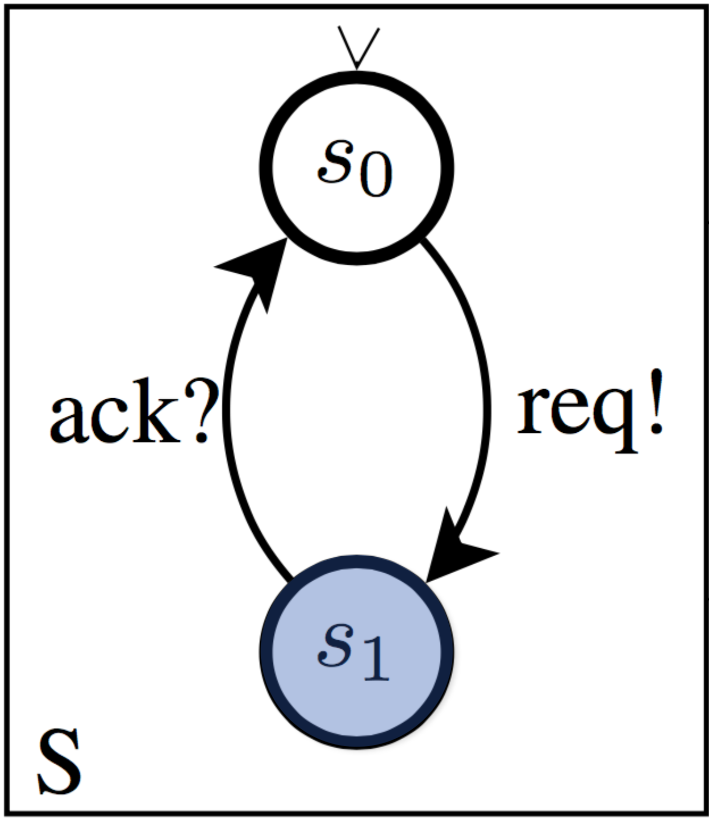
edacafe.com

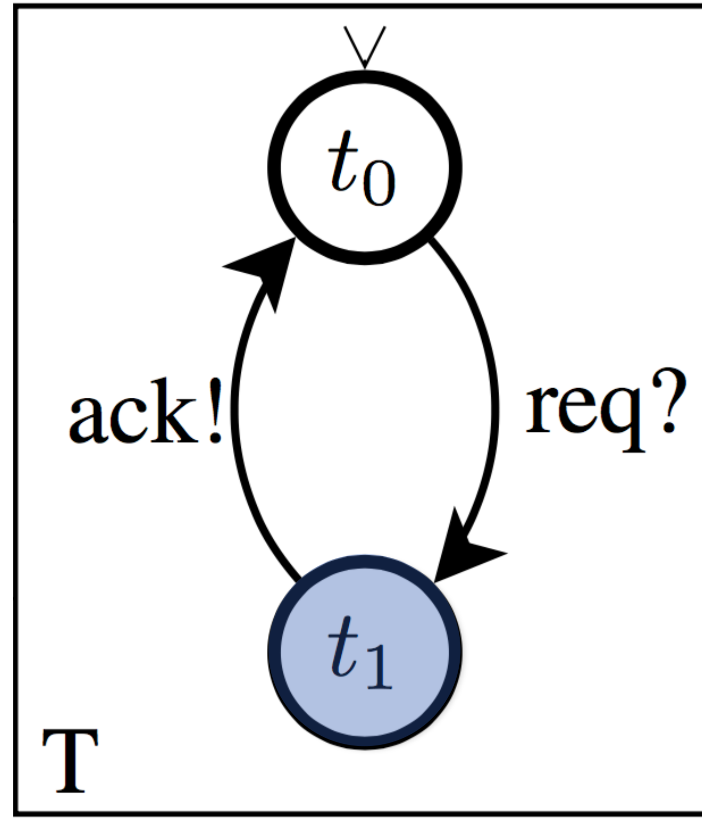

s_0
ack?
req!
s_1
S

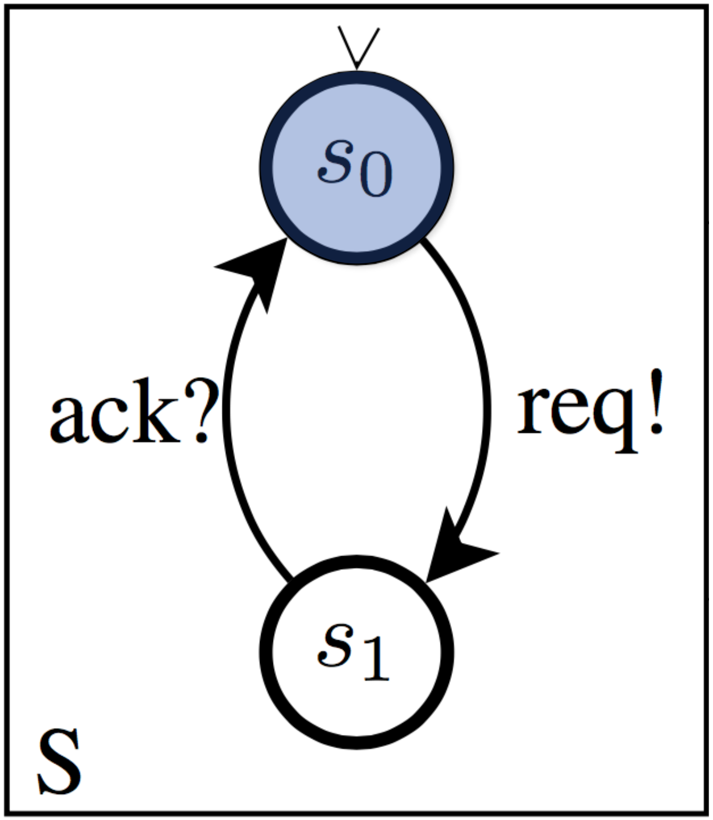

t_0
ack!
req?
t_1
T


s_0
ack?
req!
s_1
S

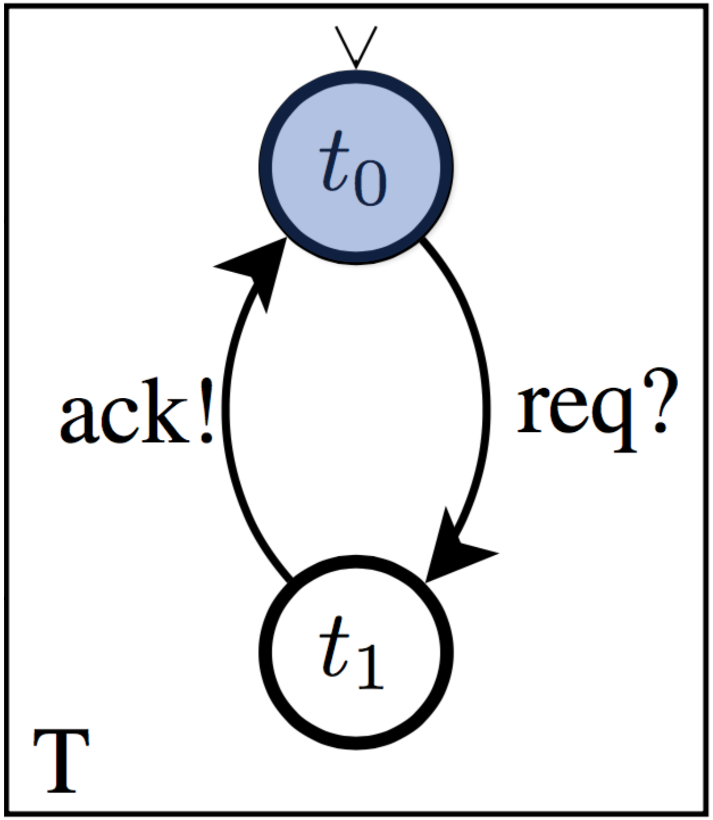

t_0
ack!
req?
t_1
T


s_0
ack?
req!
s_1
S

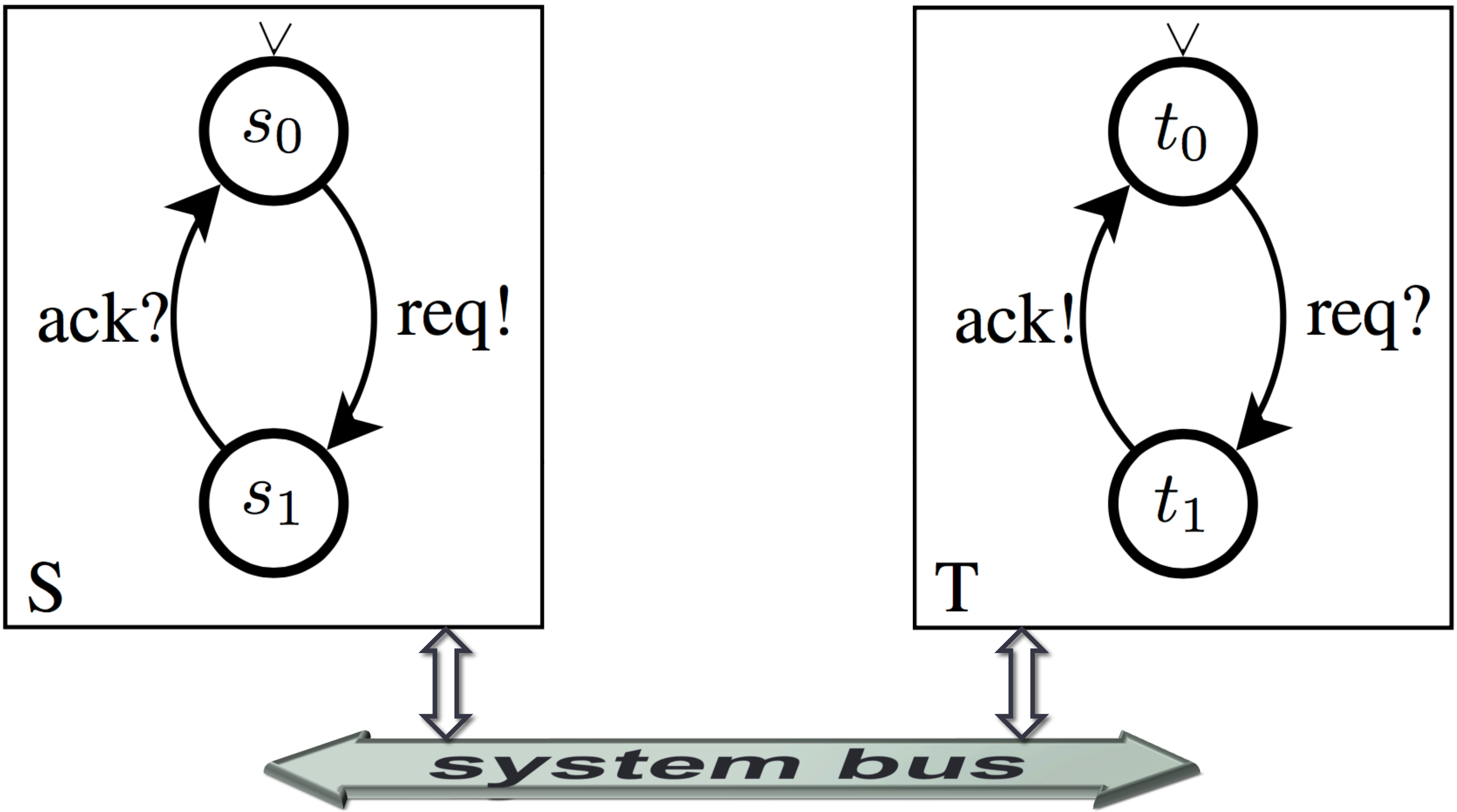

t_0
ack!
req?
t_1
T



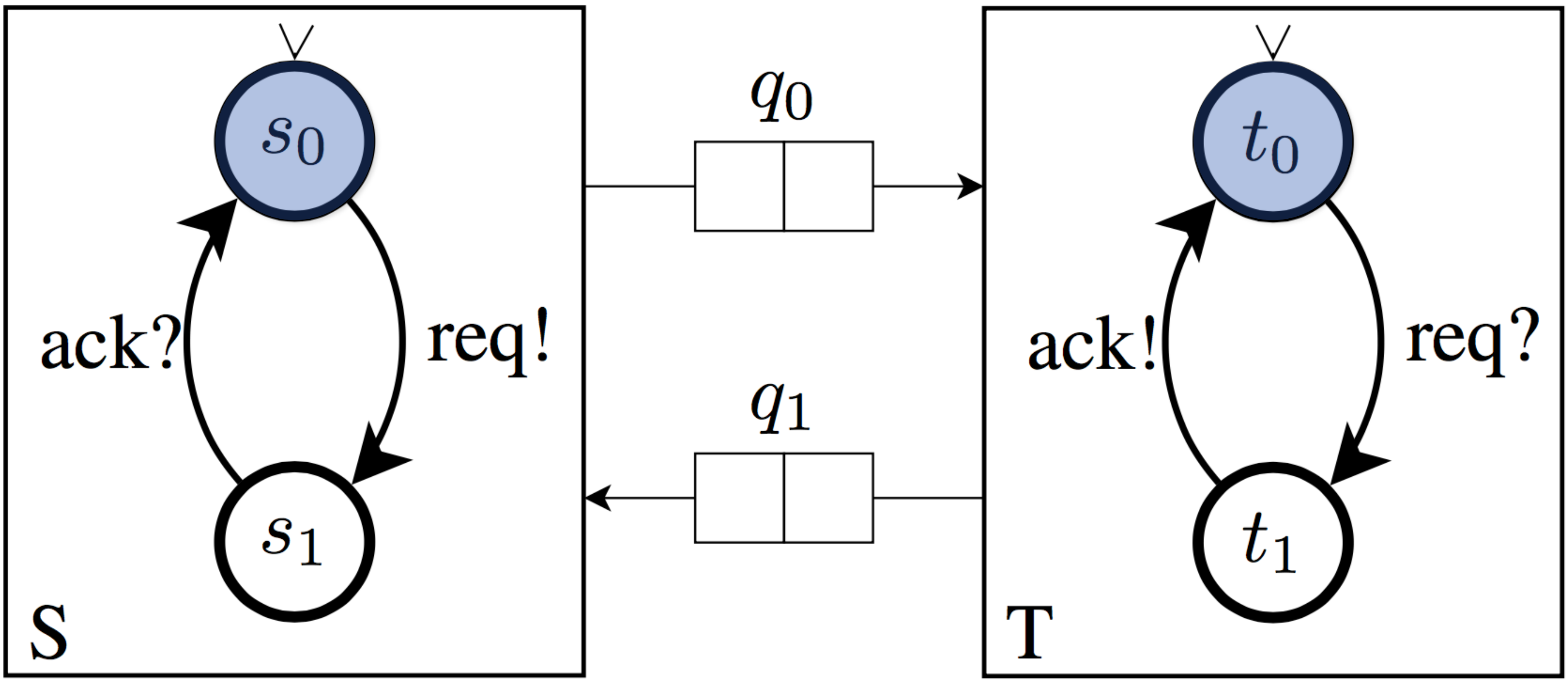
s_0
ack?
req!
s_1
S



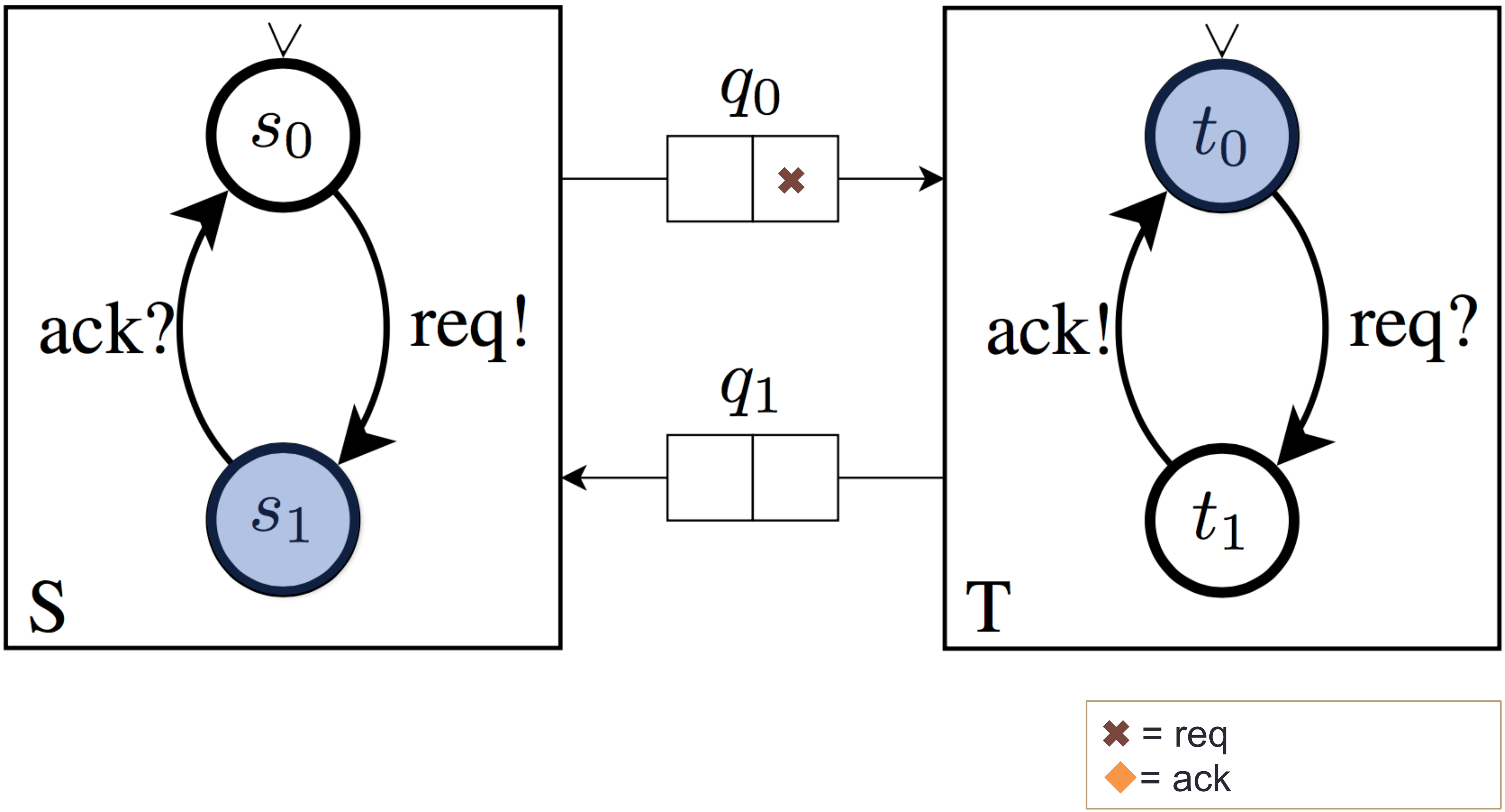
t_0
ack!
req?
t_1
T



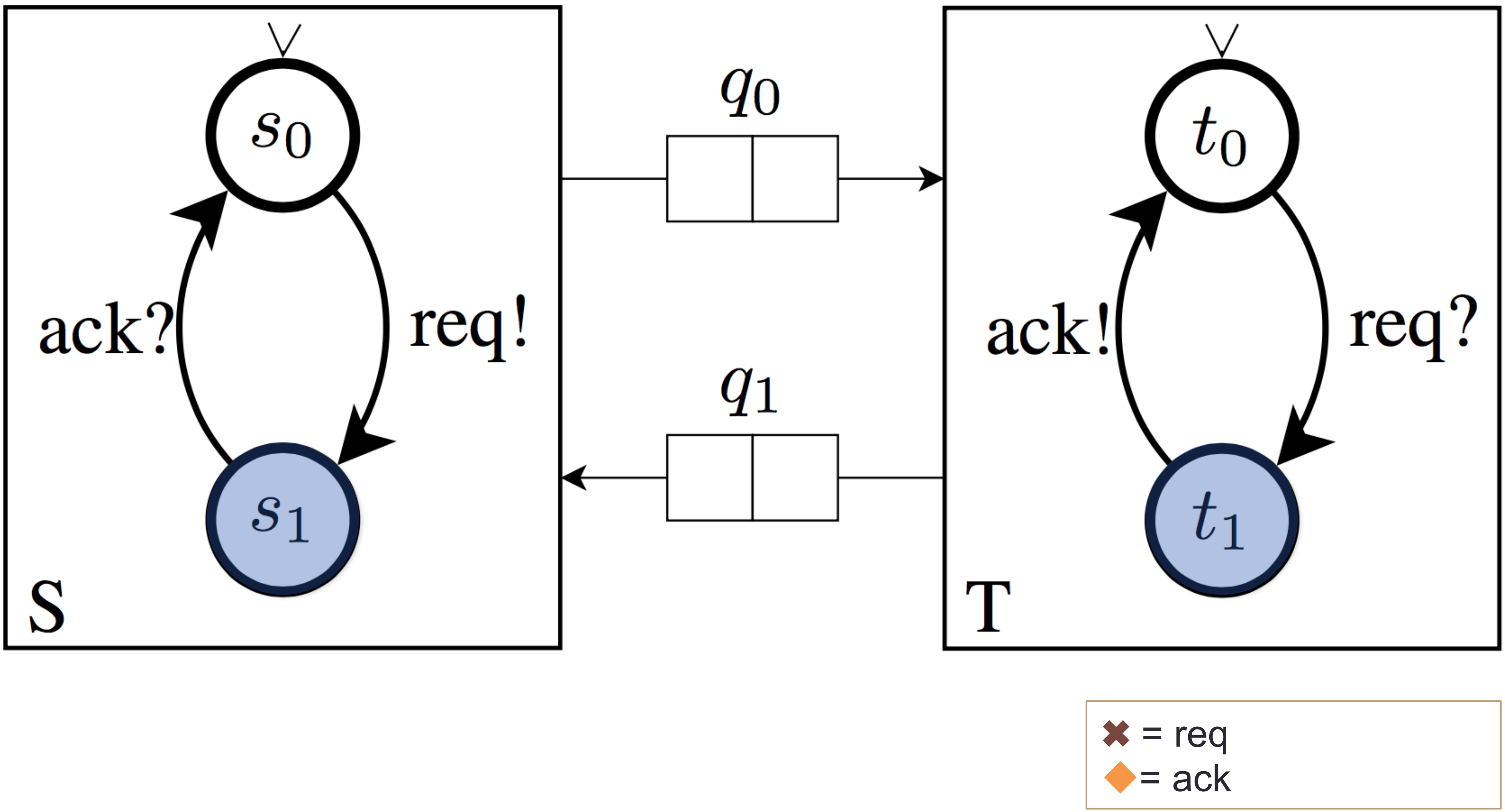
s_0
ack?
req!
s_1
S
t_0
ack!
req?
t_1
T
system bus



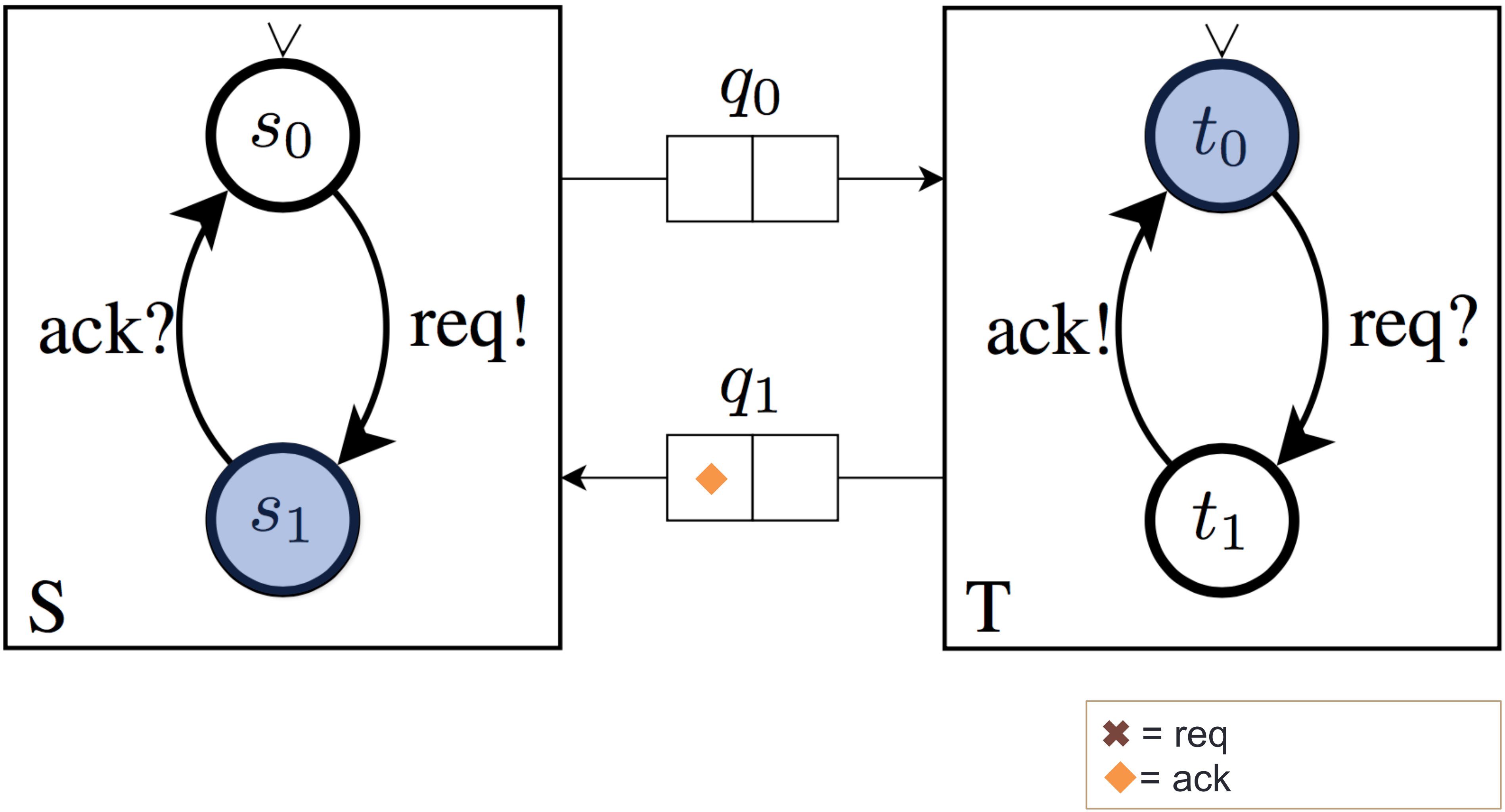
✖ = req
◆ = ack



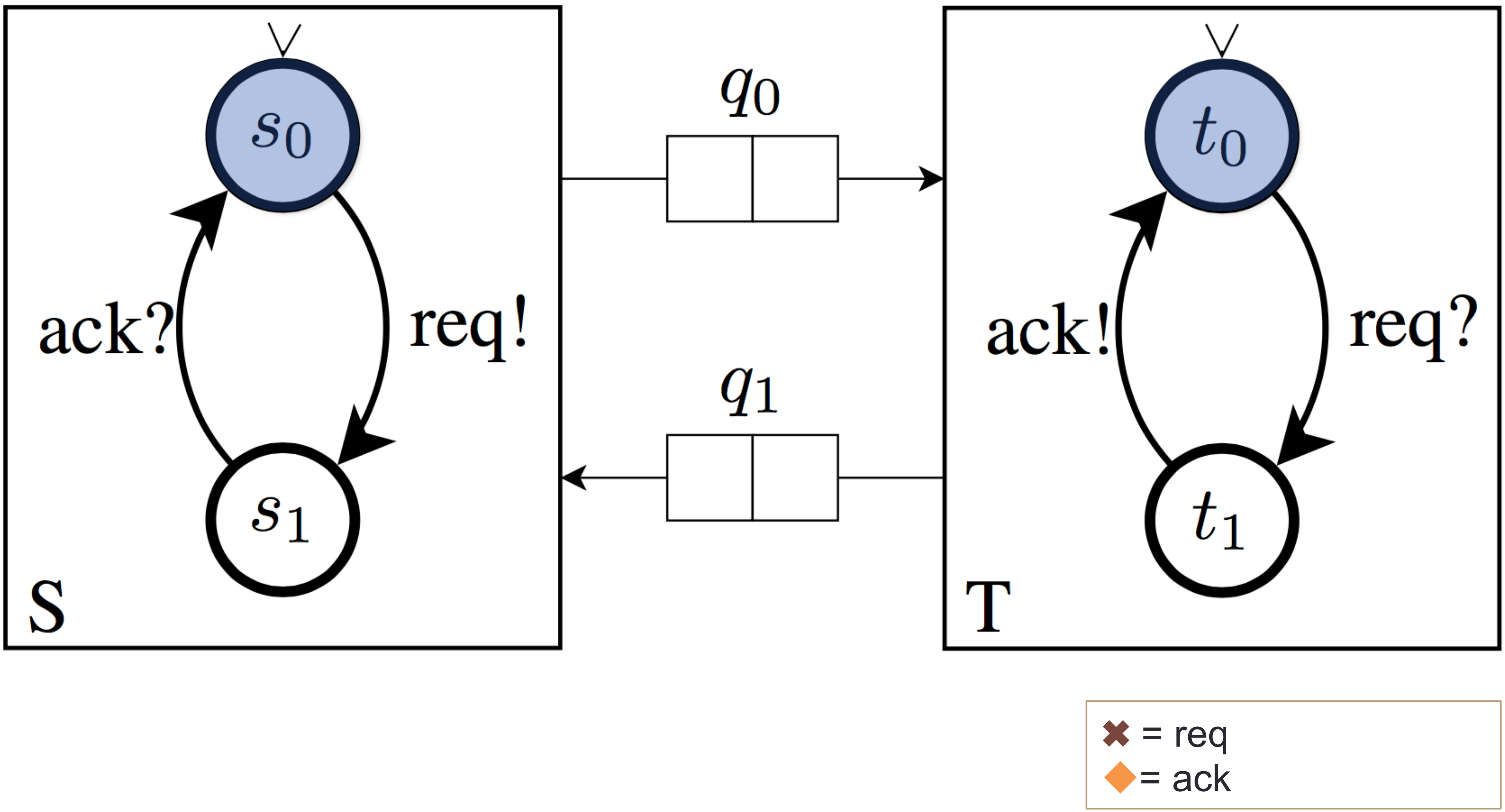
q_0
s_0
t_0
ack?
req!
ack!
req?
q_1
s_1
t_1
S
T
= req
= ack



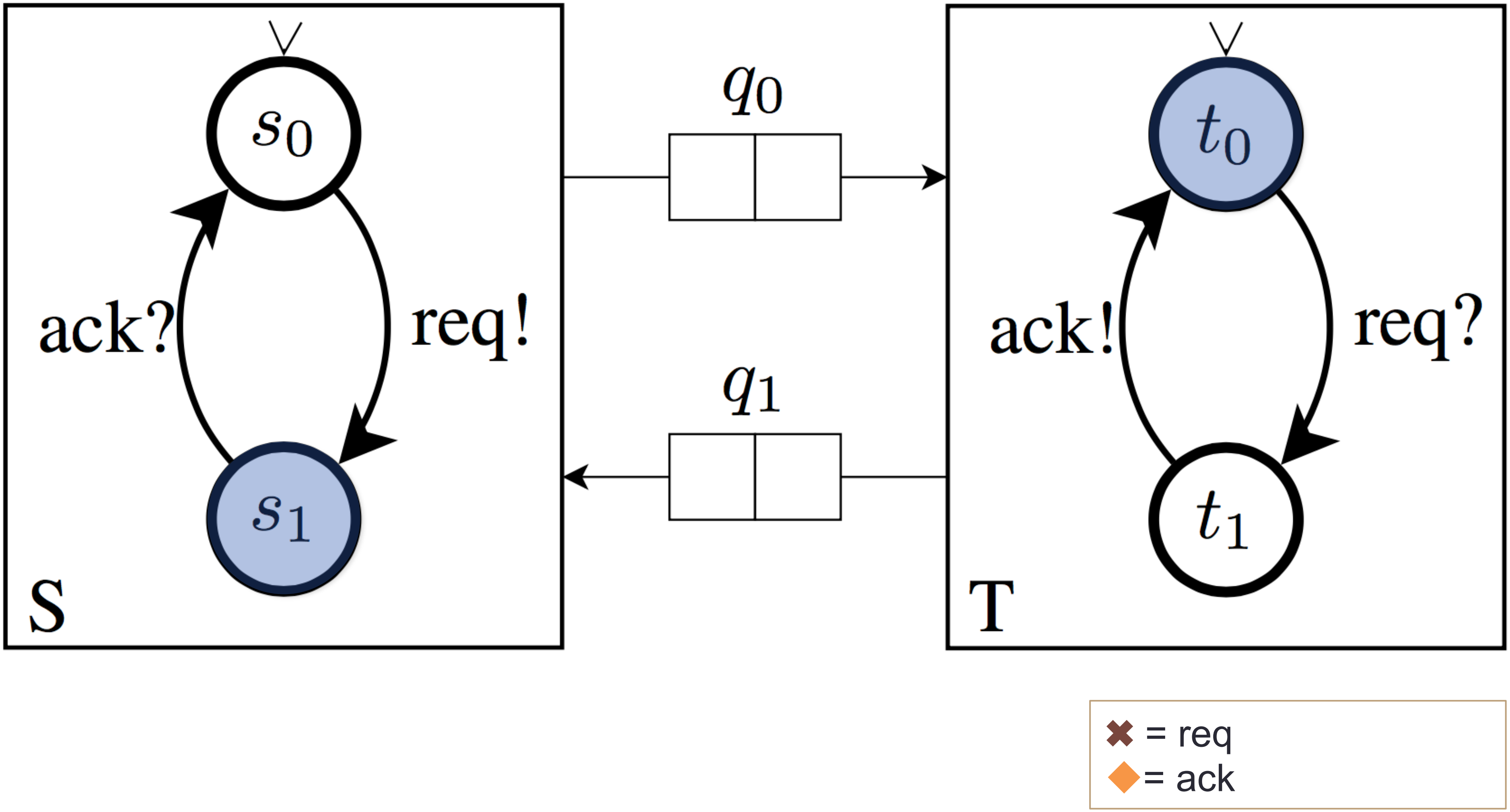
q_0
q_1
s_0
s_1
ack?
req!
S
t_0
t_1
ack!
req?
T
= req
= ack



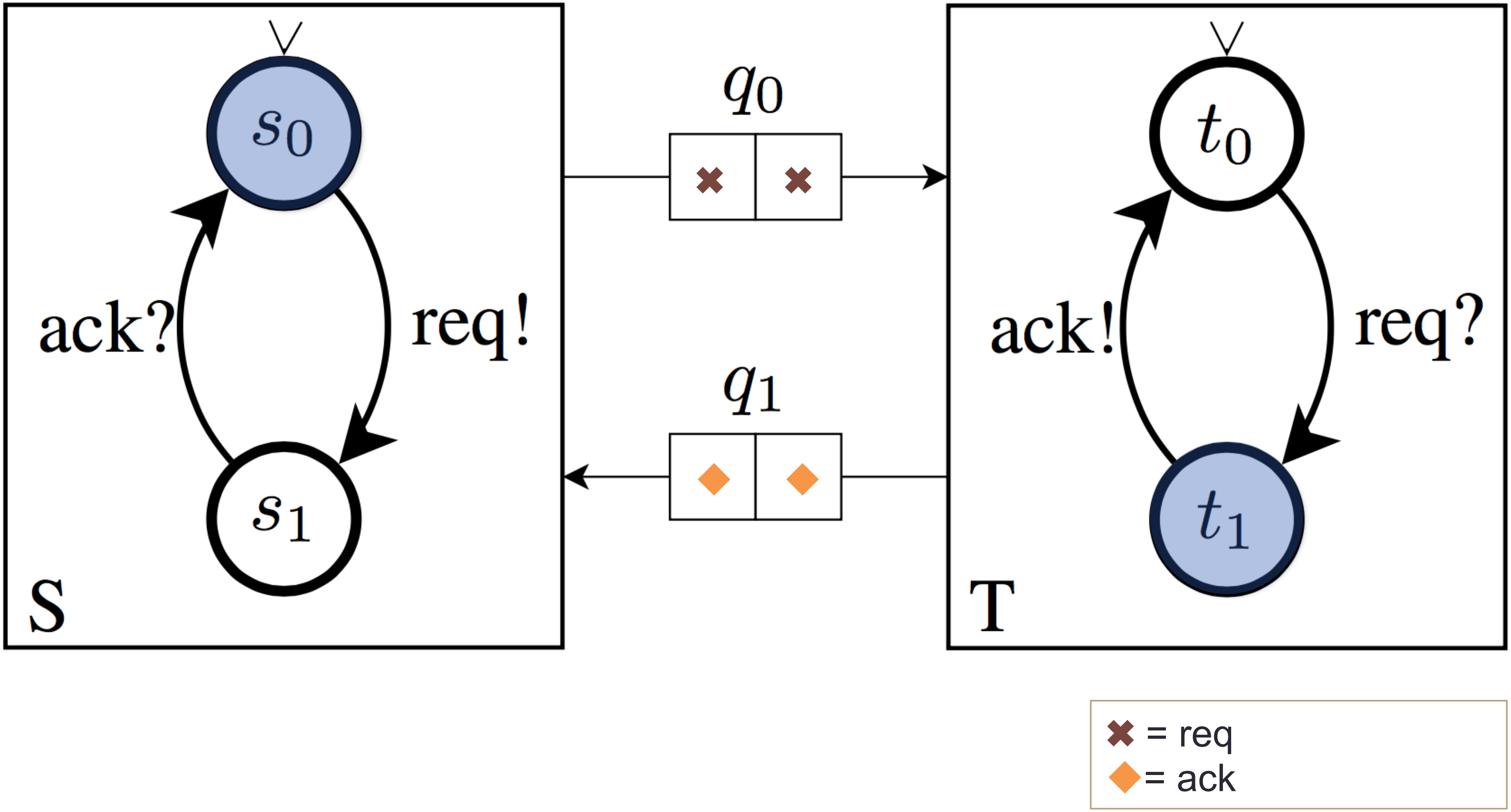
s_0
s_1
ack?
req!
S
q_0
q_1
t_0
t_1
ack!
req?
T
= req
= ack


S
T
s_0
s_1
t_0
t_1
ack?
req!
ack!
req?
q_0
q_1
= req
= ack

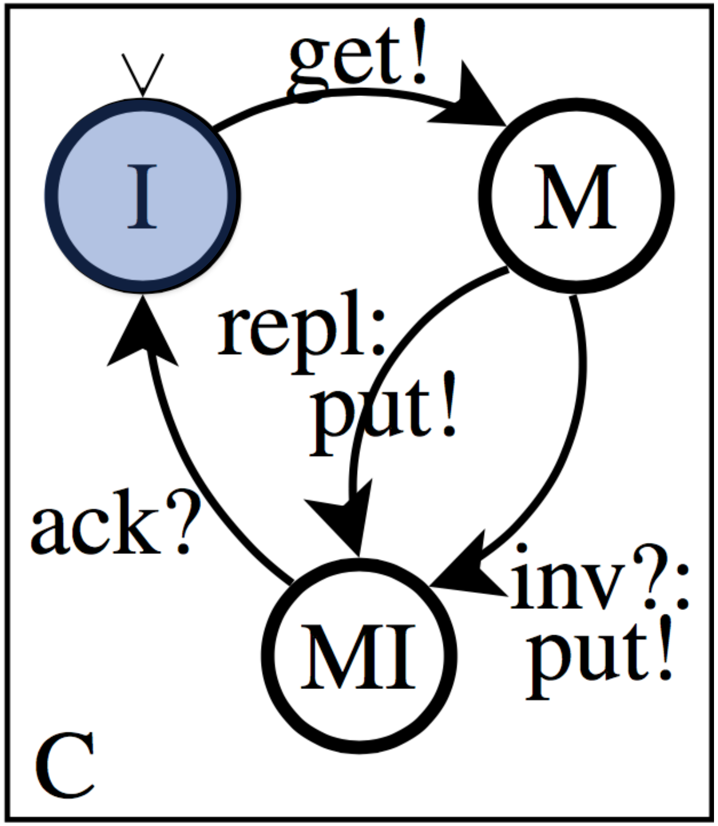
Deadlock?

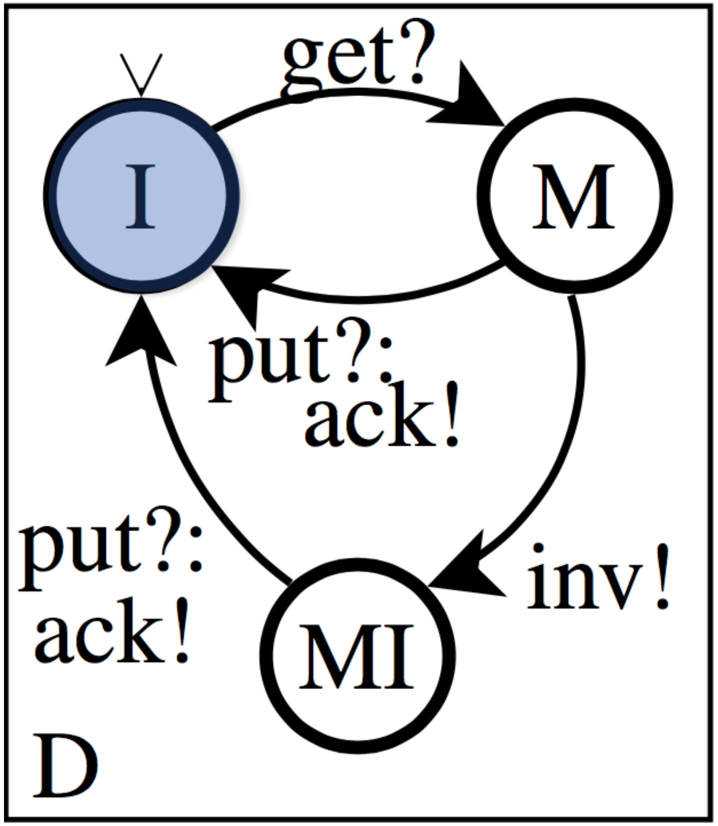


Deadlock?

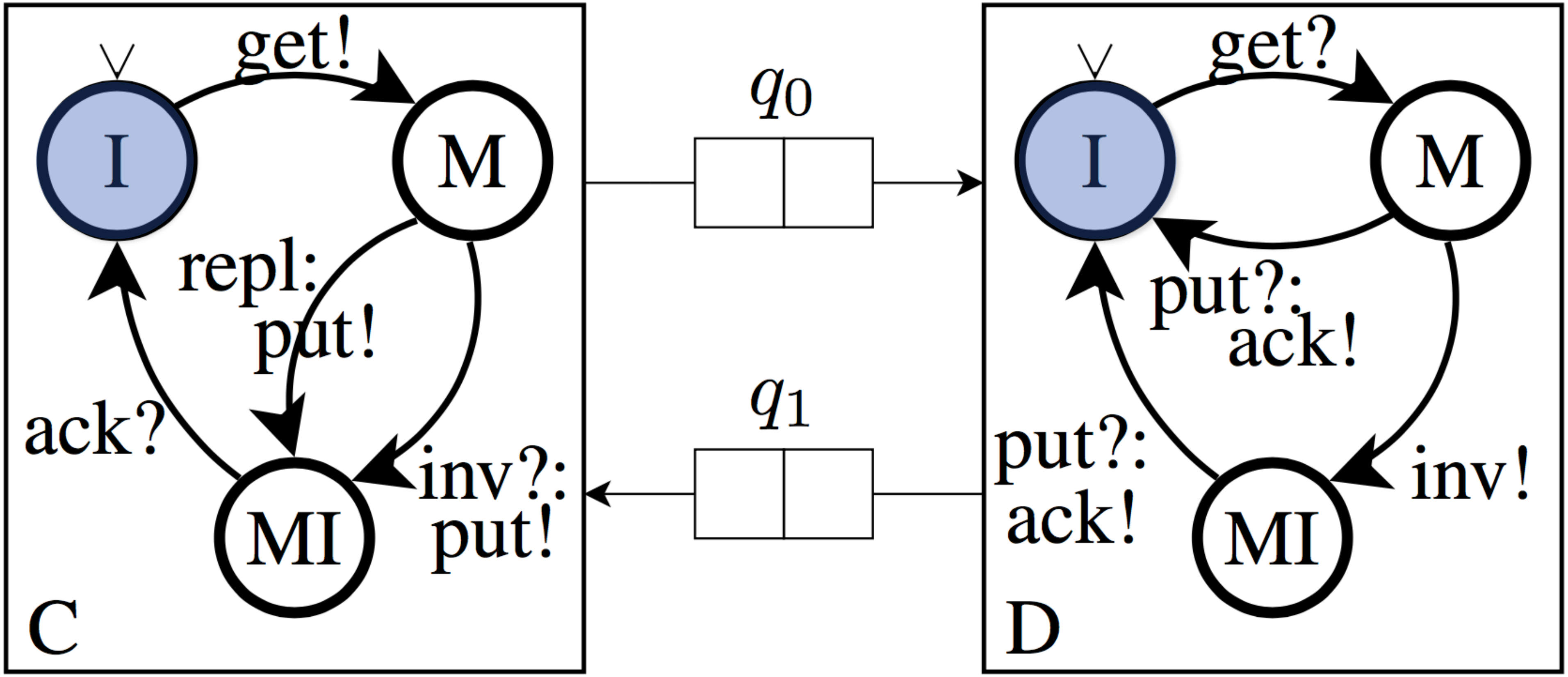


Example: cross-layer deadlock

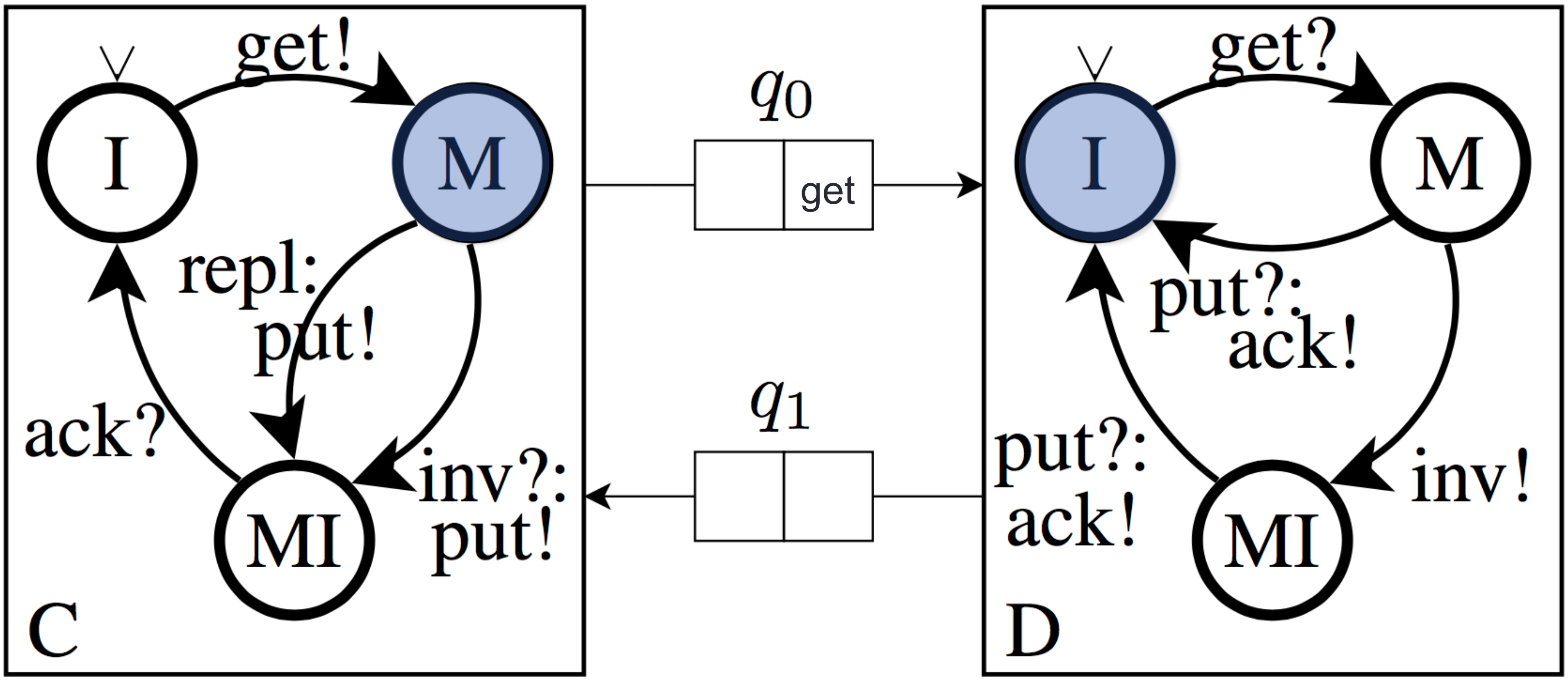




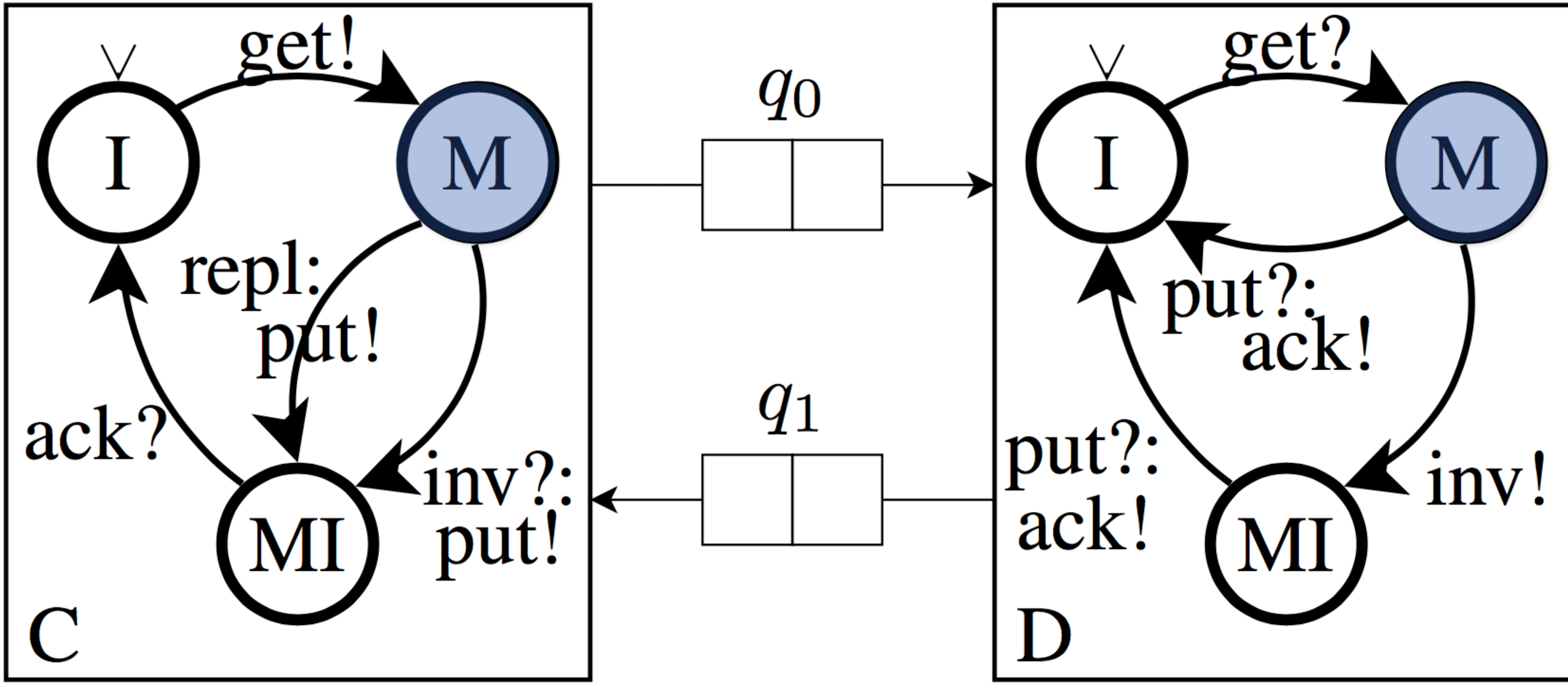
Example: cross-layer deadlock



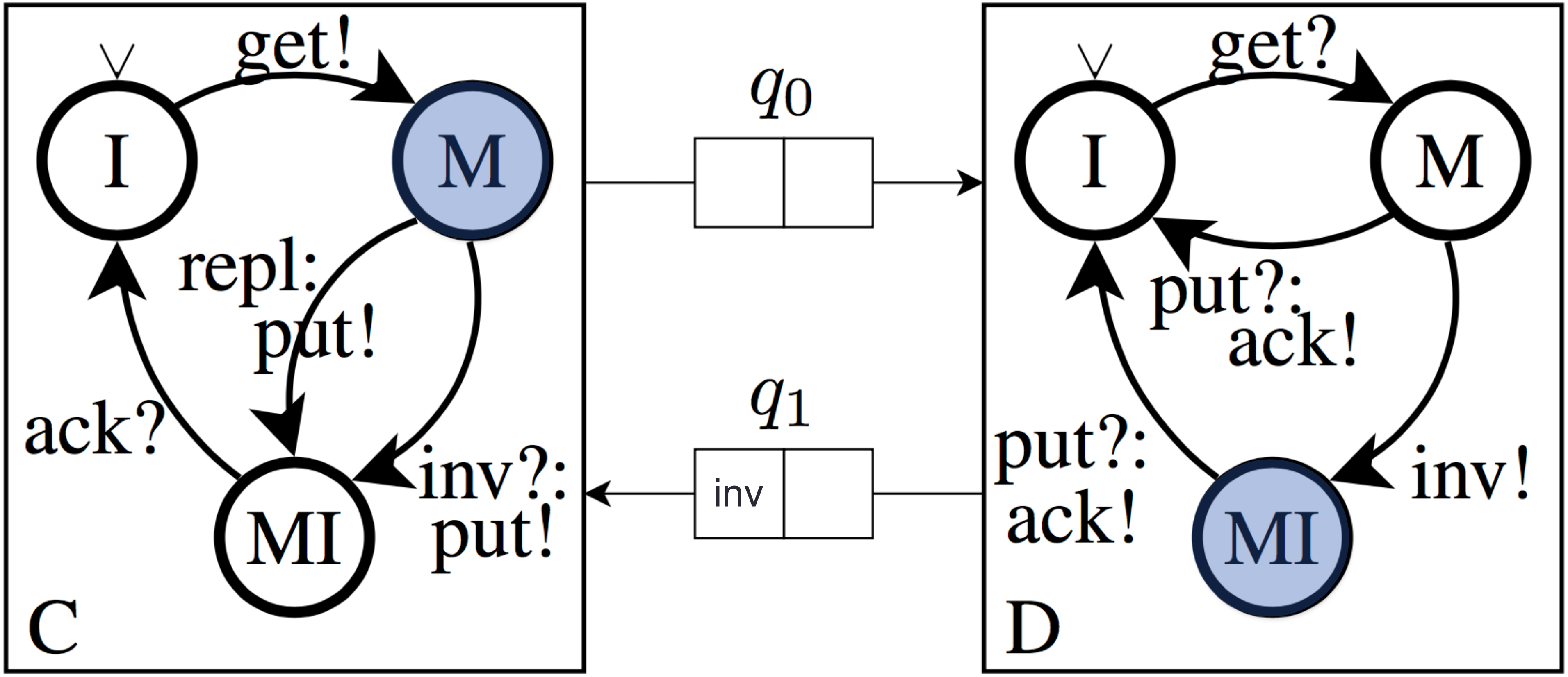
Example: cross-layer deadlock



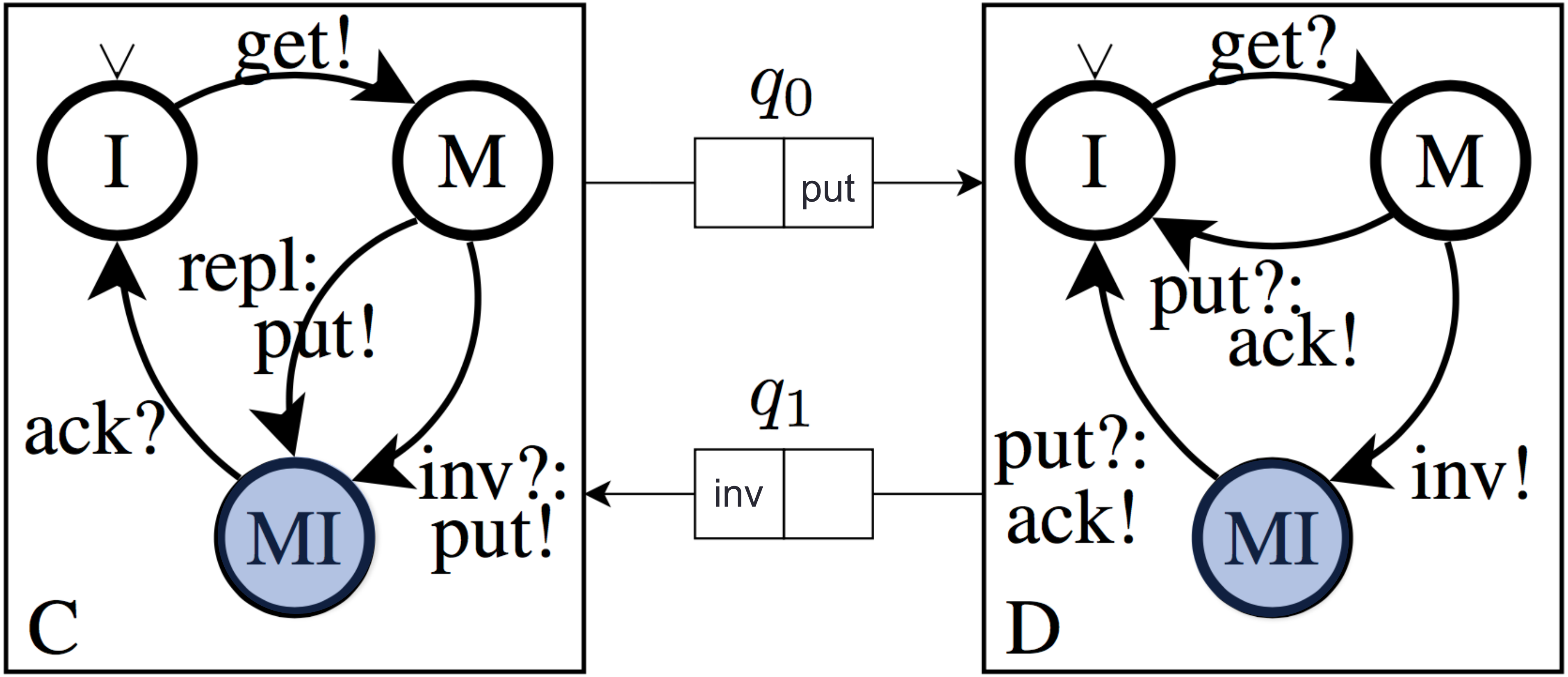
Example: cross-layer deadlock



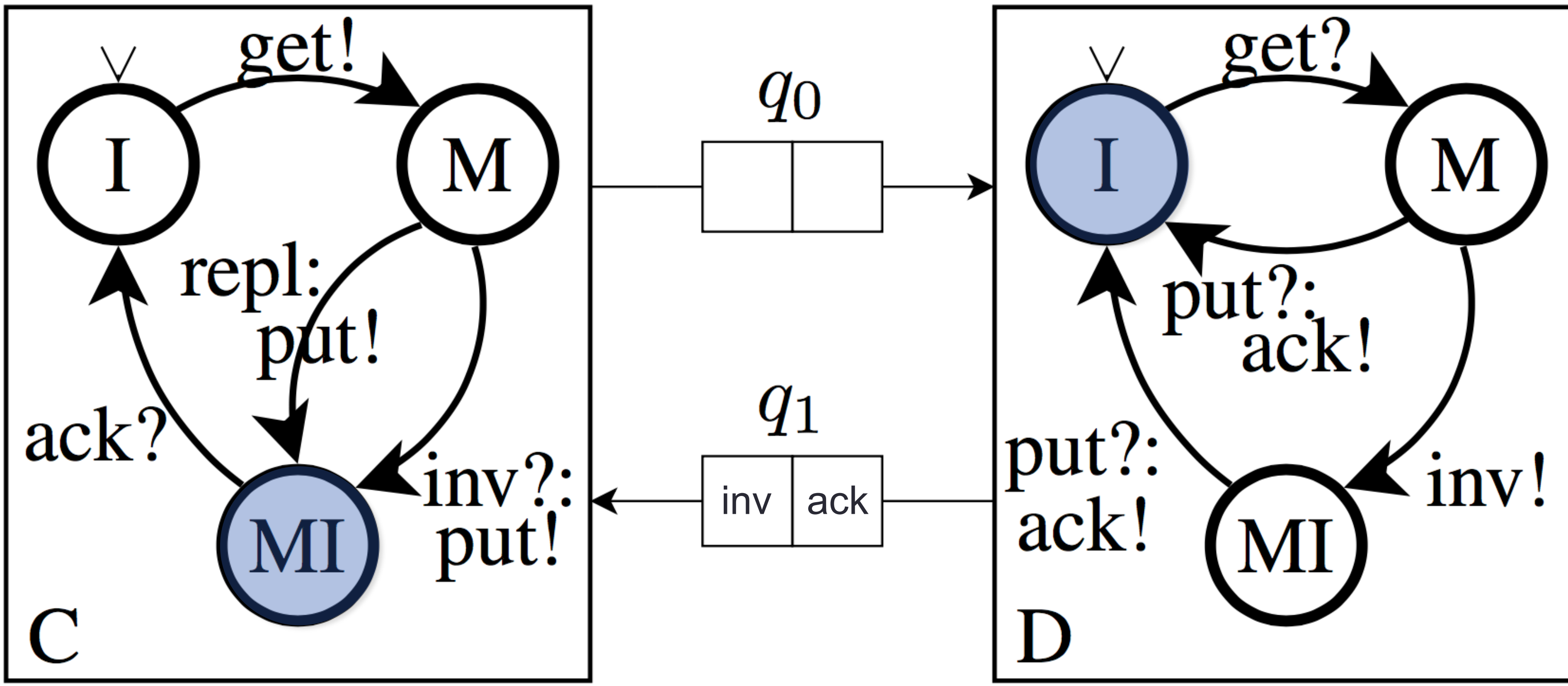
Example: cross-layer deadlock



Example: cross-layer deadlock



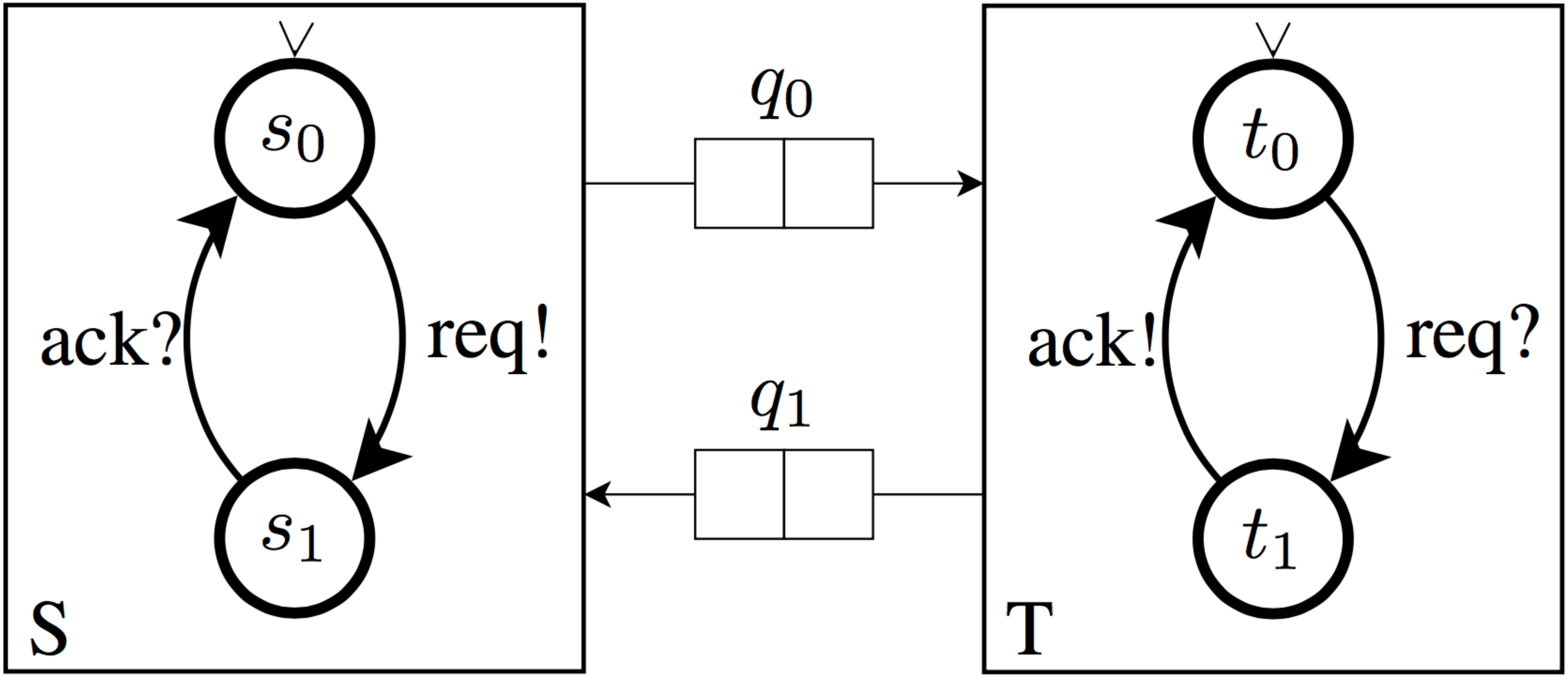
Example: cross-layer deadlock



Deadlock Detection

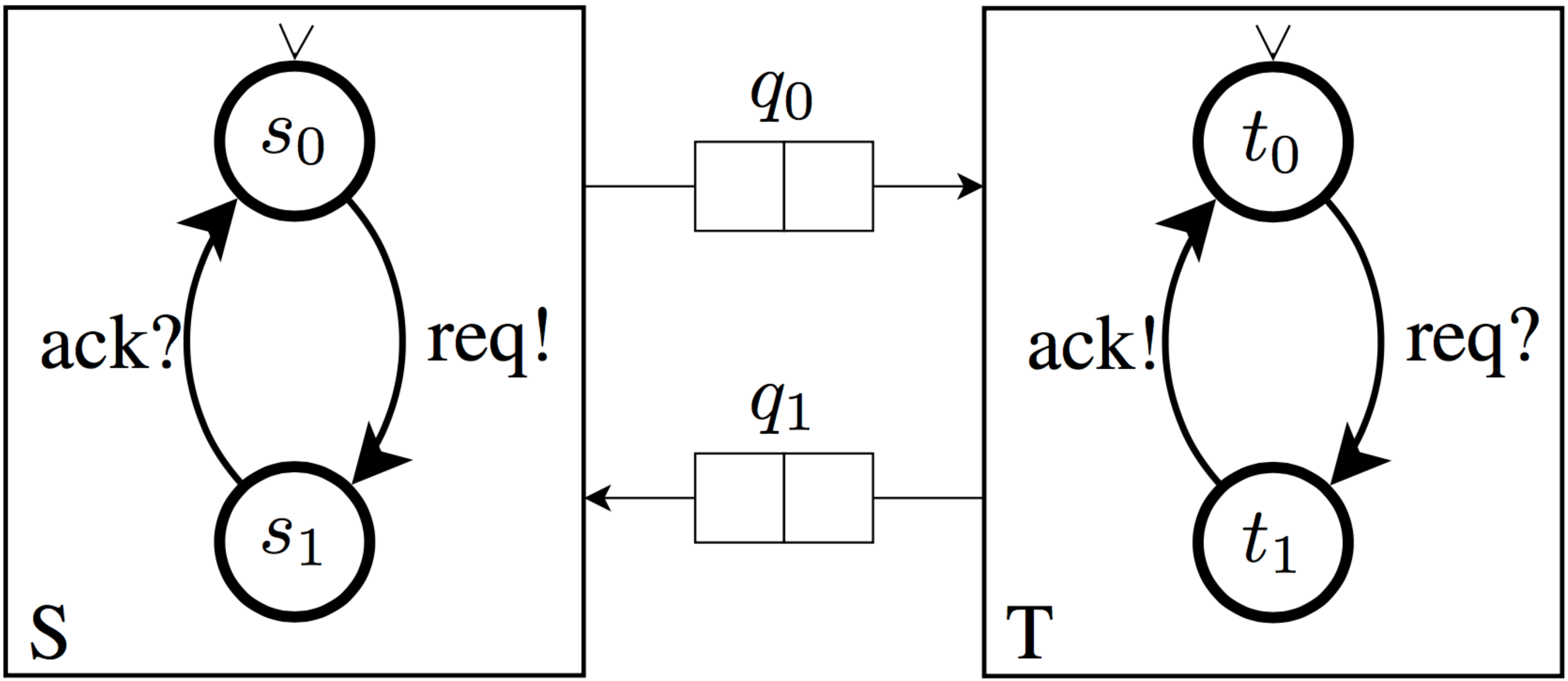
1. Model **protocol + interconnect**
2. Overapproximate deadlocks
3. Use invariants to rule out unreachable deadlocks
4. Use SMT solver to prove deadlock-freedom or find possible deadlock

1.) Overapproximate deadlocks



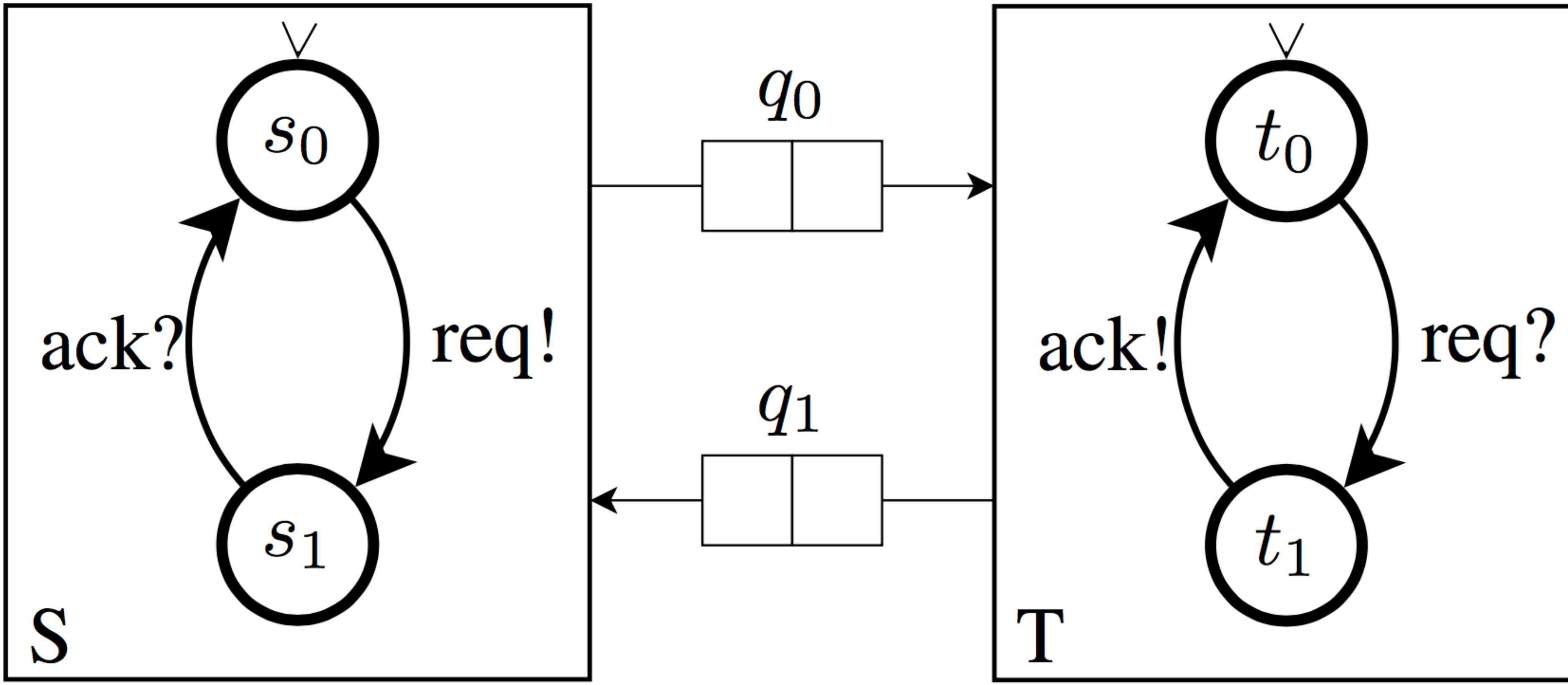
$$\bigvee \begin{matrix} s_0 \wedge \#q_0 = q_0.\text{size} \wedge t_1 \wedge \#q_1 = q_1.\text{size} \\ s_1 \wedge \#q_1 = 0 \wedge t_0 \wedge \#q_0 = 0 \end{matrix}$$

2.) Derive invariants



$$T.t_0 - S.s_0 = \#q_0.req + \#q_1.ack$$

3.) Use SMT solver

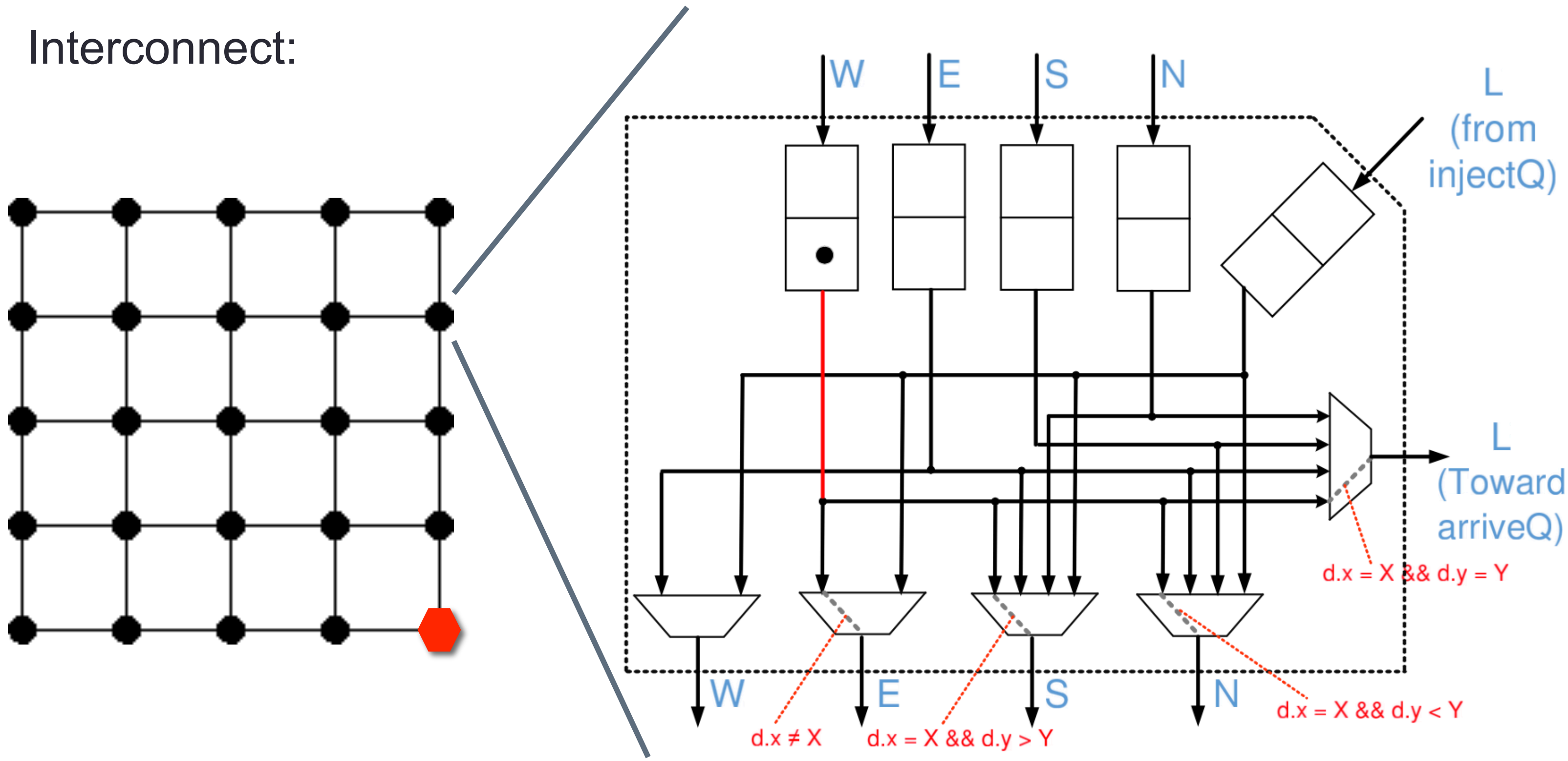


$$\bigvee \begin{array}{l} s_0 \wedge \#q_0 = q_0.\text{size} \wedge t_1 \wedge \#q_1 = q_1.\text{size} \\ s_1 \wedge \#q_1 = 0 \wedge t_0 \wedge \#q_0 = 0 \end{array}$$

$$\bigwedge \; T.t_0 - S.s_0 = \#q_0.req + \#q_1.ack$$

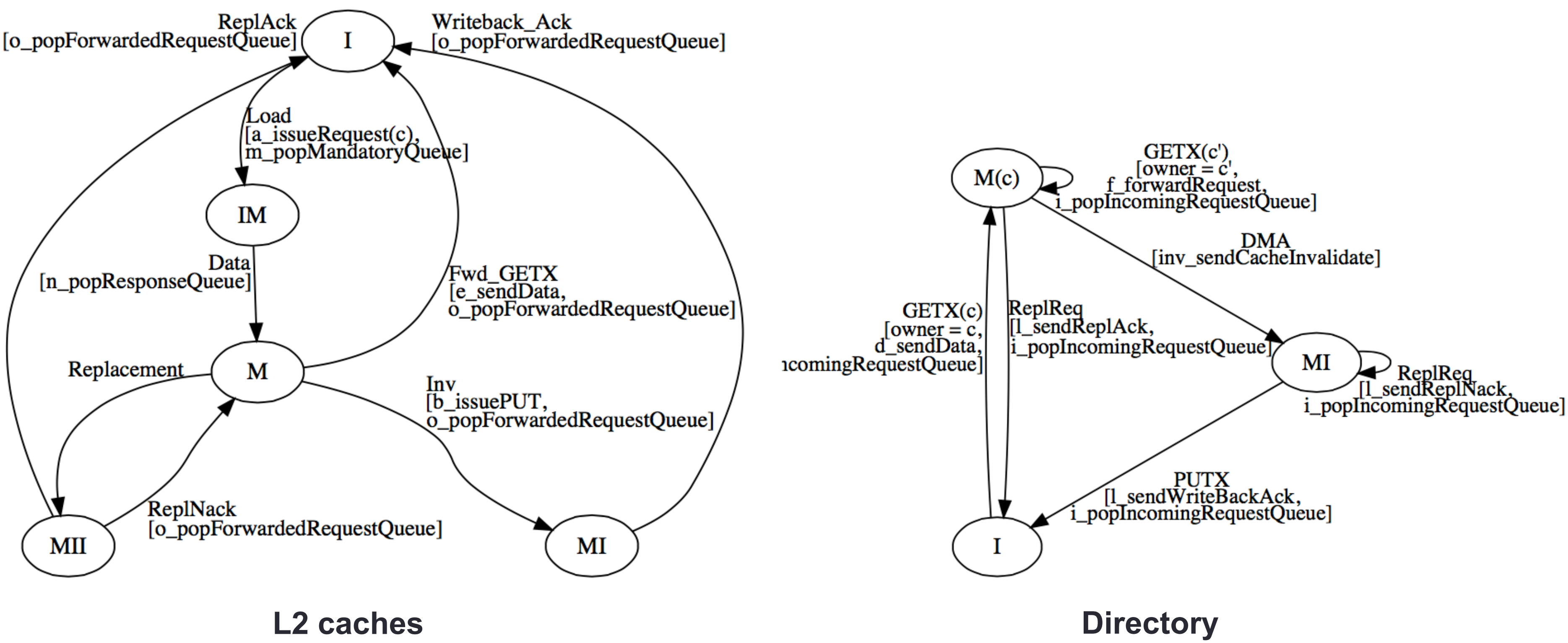
Case Study:
2D mesh, XY routing, MI cache coherence protocol

Interconnect:

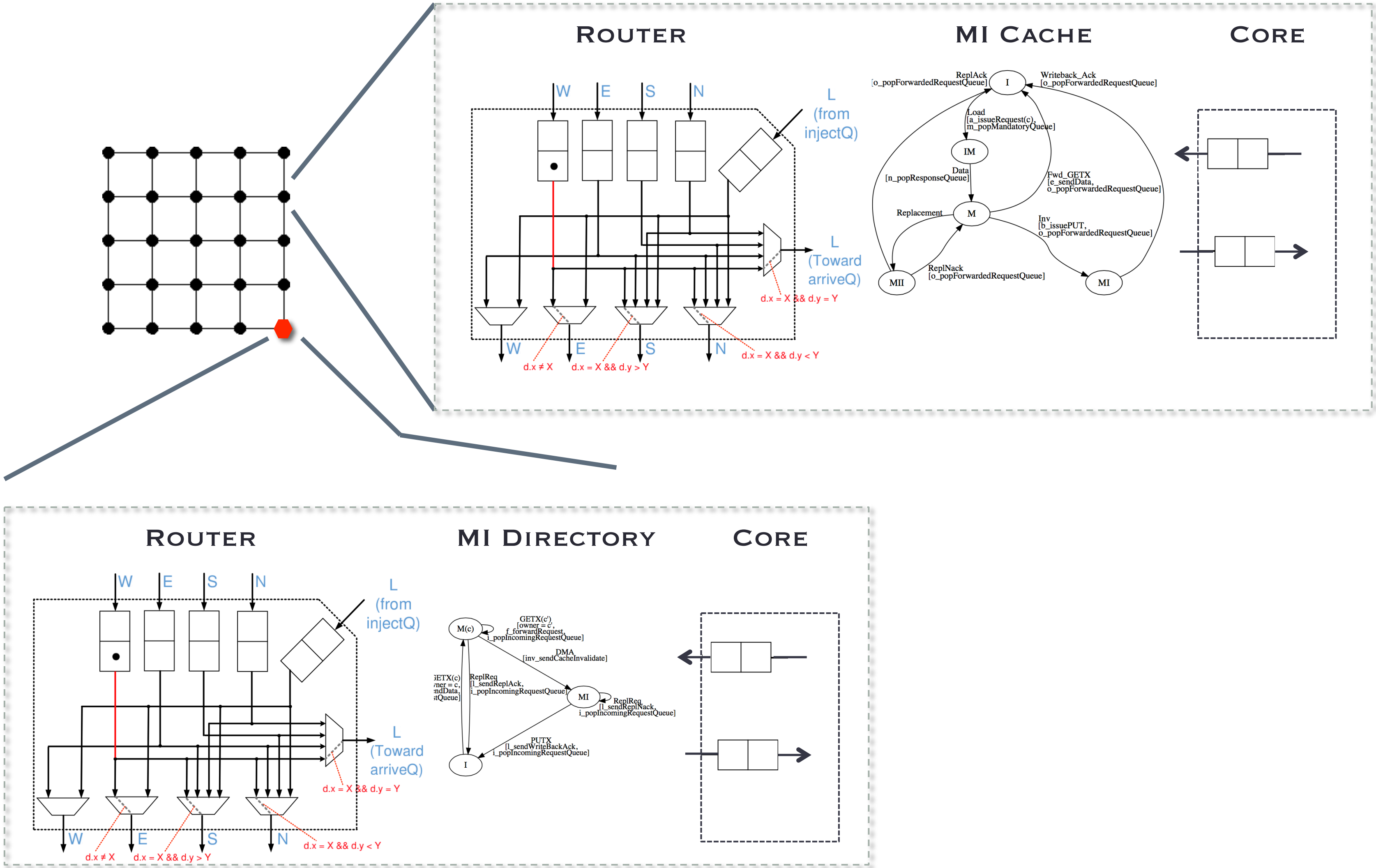


Case Study:
2D mesh, XY routing, MI cache coherence protocol

Protocol:



Case Study:
2D mesh, XY routing, MI cache coherence protocol

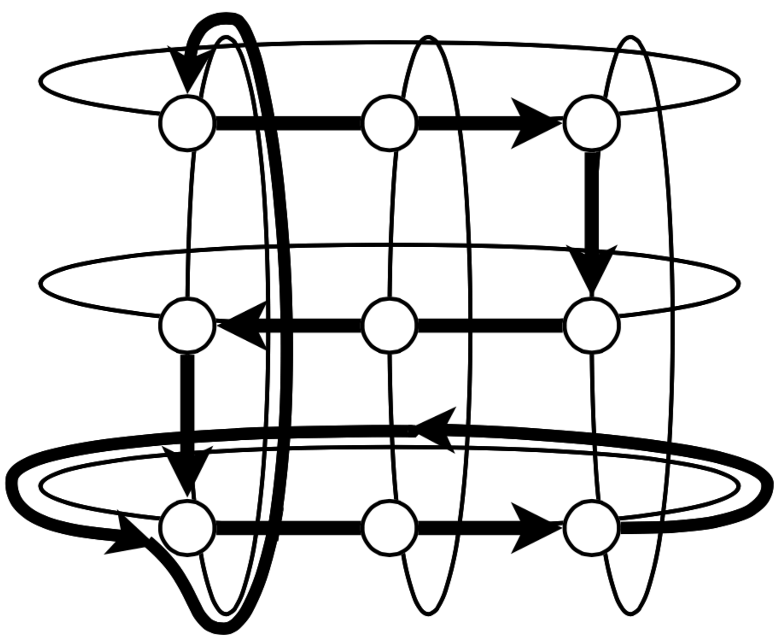


Experimental Results

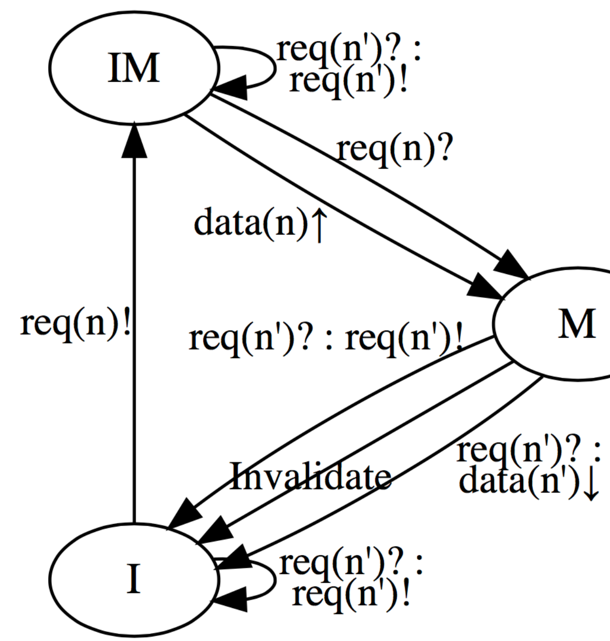
Size	DL	DLF	#primitives	#queues	#automata
2×2	1.7s	1.3s	100	24	4
3×3	23s	16s	225	54	9
4×4	3m52s	2m41s	400	96	16
5×5	33m18	23m5s	625	150	25
2×2	0.8s	0.7s	100	28	4
3×3	7.4s	5.0s	225	63	9
4×4	49.7s	25s	400	112	16
5×5	4m39s	1m48s	625	175	25

Case Study:
2D torus with ring, XY routing, snoopy cache coherence protocol

Interconnect:



Protocol:



Experimental Results

Size	DL	DLF	#primitives	#queues	#automata
2×2	1.7s	1.3s	100	24	4
3×3	23s	16s	225	54	9
4×4	3m52s	2m41s	400	96	16
5×5	33m18	23m5s	625	150	25
2×2	0.8s	0.7s	100	28	4
3×3	7.4s	5.0s	225	63	9
4×4	49.7s	25s	400	112	16
5×5	4m39s	1m48s	625	175	25

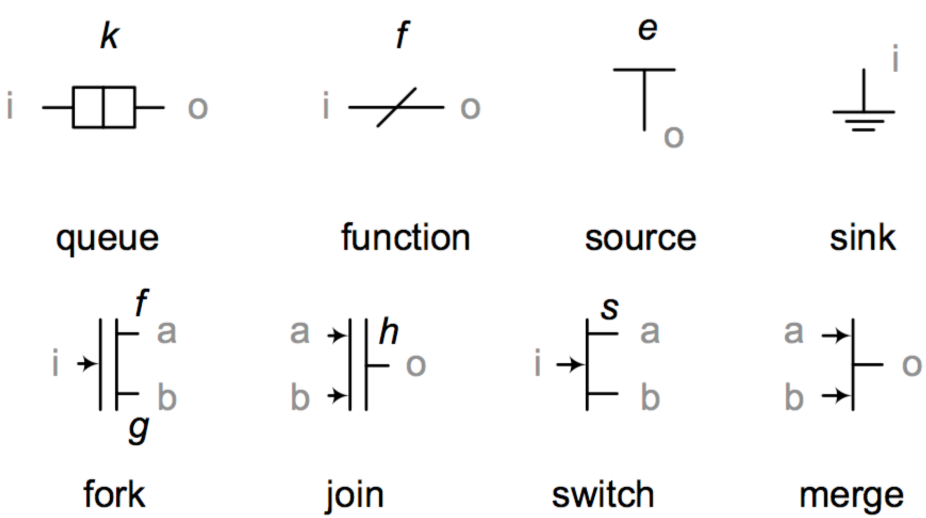
Deadlock Detection

1. Model **protocol + interconnect**
2. Overapproximate deadlocks
3. Use invariants to rule out unreachable deadlocks
4. Use SMT solver to prove deadlock-freedom or find possible deadlock

Modelling…

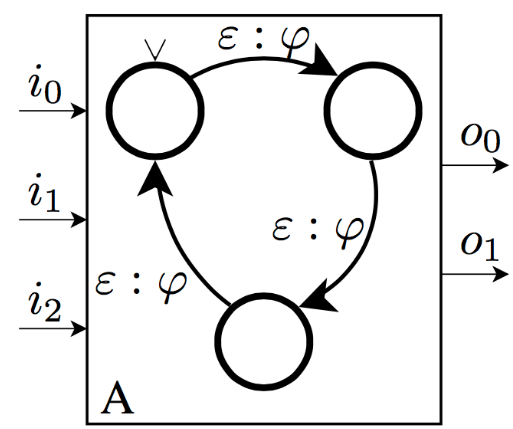
… the interconnect

- xMAS
- graphical language
- formal semantics:
  trdy/irdy/data signals



... the protocol

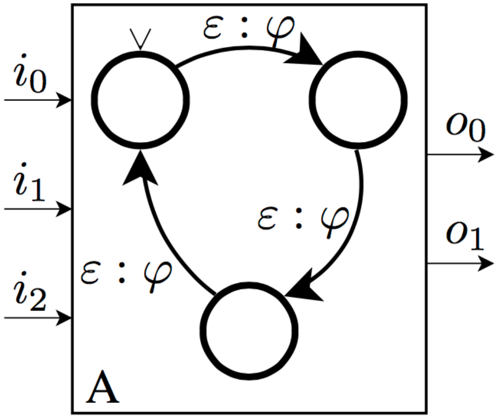
- IO automata
- events and transformations
- formal semantics:
  trdy/irdy/data signals



Semantics

***Definition* 2.** Let A be an XMAS automaton and let $A.s$ denote that automaton A is in state s. A transition $t = \langle s, s', \varepsilon, \varphi \rangle$ of XMAS automaton A *is enabled for in-channel* i, notation $\mathrm{enabled}(t, i)$, if and only if:

$$\begin{aligned} \mathrm{enabled}(t, i) &\stackrel{\mathrm{def}}{=} A.s \wedge i.\mathit{irdy} \wedge \\ &\quad \varepsilon(i, i.\mathit{data}) \wedge \mathrm{rdy}(\varphi(i, i.\mathit{data})) \\ \text{where } \mathrm{rdy}(\bot) &= \mathit{True} \\ \mathrm{rdy}(o, d') &= o.\mathit{trdy} \end{aligned}$$



Invariant Generation

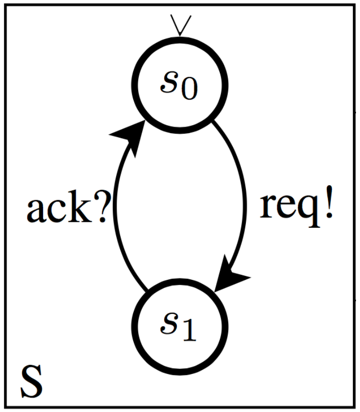
1. The sum of firing ingoing transitions equals the sum of firing outgoing transitions.

$$\sum_{t \in \langle \_, s, \_, \_ \rangle} \kappa_A^t = \left(\sum_{t \in \langle s, \_, \_, \_ \rangle} \kappa_A^t \right) + A.s - \mathrm{isInit}(s)$$

where

$$\mathrm{isInit}(s) = \text{if } s = s_0 \text{ then } 1 \text{ else } 0$$

Example:



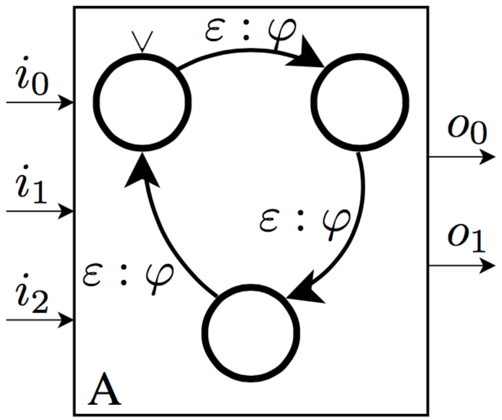
$$\#req! = \#ack? + S.s_1$$

$$\#ack? = \#req! + S.s_0 - 1$$

Invariant Generation

Let $\sim$ be a partitioning of all pairs (i, d) such that:

$$(\exists_{t=\langle s,s',\varepsilon,\varphi\rangle} \cdot \varepsilon(i,d) \wedge \varepsilon(i',d')) \implies (i,d) \sim (i',d')$$

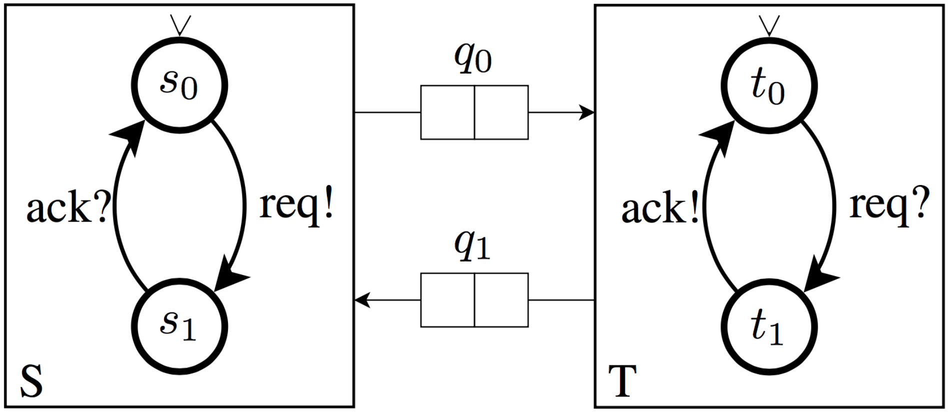


Invariant Generation

2. The sum of incoming packets equals the number of times a transition fired that consumes such a packet.

$$\sum_{(i,d)\in I} \lambda_i^d = \sum_{t\in T(I)} \kappa_A^t$$

Example:



$$\lambda_{q_1.\text{out}}^{\text{ack}} = \#ack?$$

Conclusion

- Methodology for finding *cross-layer* deadlocks
  - Makes use of cross-layer invariants
- Monolithic verification of protocol and interconnect
  - generic w.r.t. interconnect
  - generic w.r.t. protocol
- Fully *automated*
  - Haskell implementation of invariant generation, deadlock detection, and various required paraphernalia