Reachability of Communicating Timed Processes

Lorenzo Clemente (1), Frédéric Herbreteau (1),
Amélie Stainer (2), and Grégoire Sutre (1)

(1) University of Bordeaux/LaBRI
(2) University of Rennes I

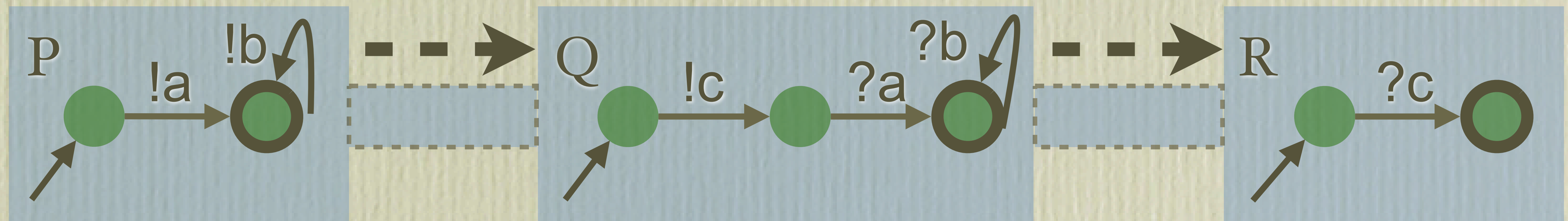
Plouzané, Tuesday 16th April 2013

At a glance

Model:	Communicating Finite State Machines + *Time*
Problem:	Decidability and complexity of reachability w.r.t. the communication *topology*
Results:	Adding time does not change decidability, but the complexity worsens

Communicating Finite State Machines

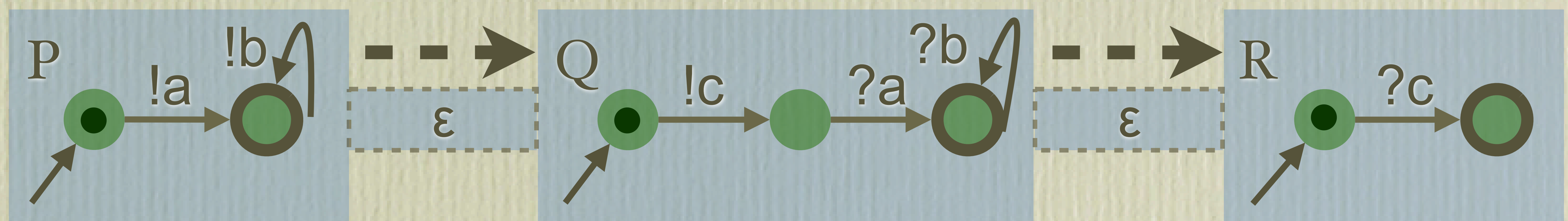
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

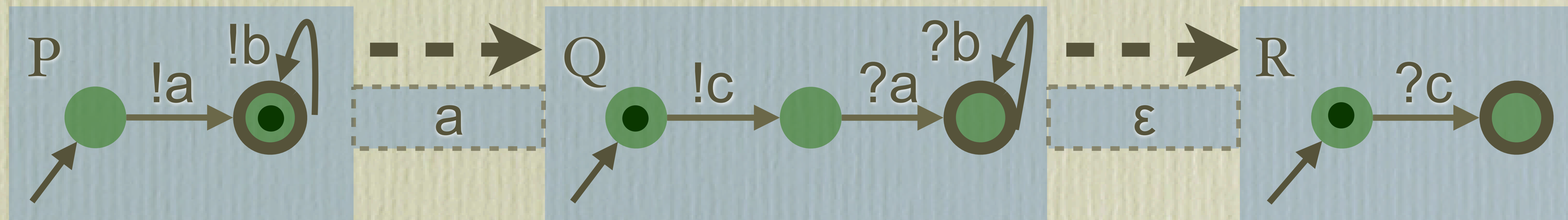
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

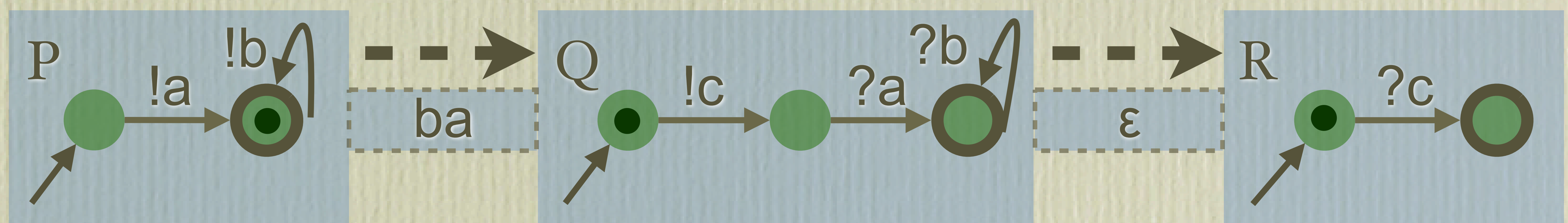
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

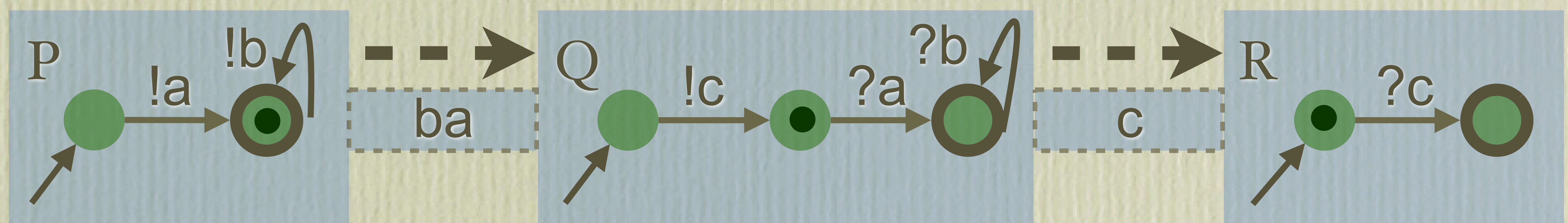
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

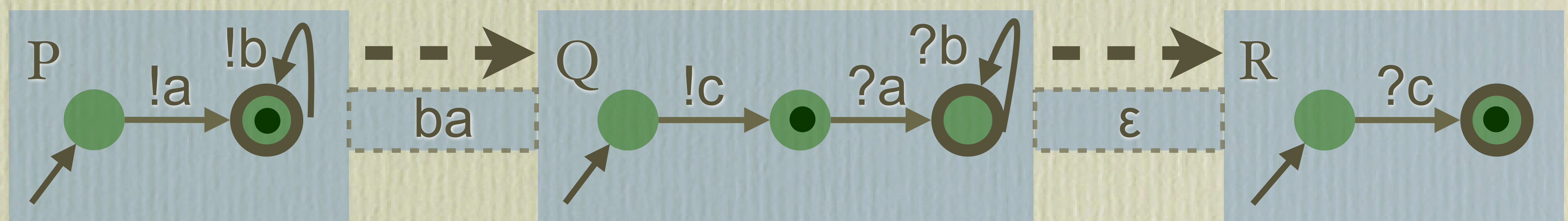
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

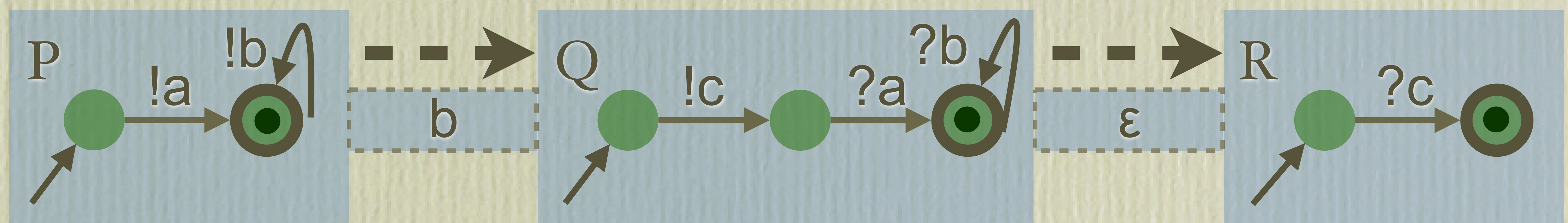
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

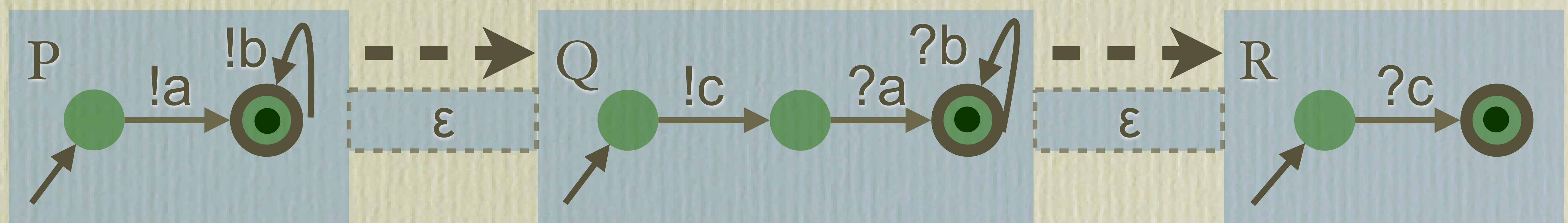
Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

Network of finite automata communicating over *unbounded* FIFO channels.



Reachability: Every process is in a final state and channels are empty

Communicating Finite State Machines

Reachability is in general *undecidable* [Brand&Zafiropulo].
Decidabile restrictions:

- Bound the channel (→ finite state system).
- Restrict the message alphabet to a singleton (→ Petri nets) [Karp&Miller'69].
- Make the channel lossy [Abdulla&Jonsson'96, Cece&Finkel&Iyer'96].
- Restrict to mutex communication [Heussner&Leroux&Muscholl&Sutre'10].

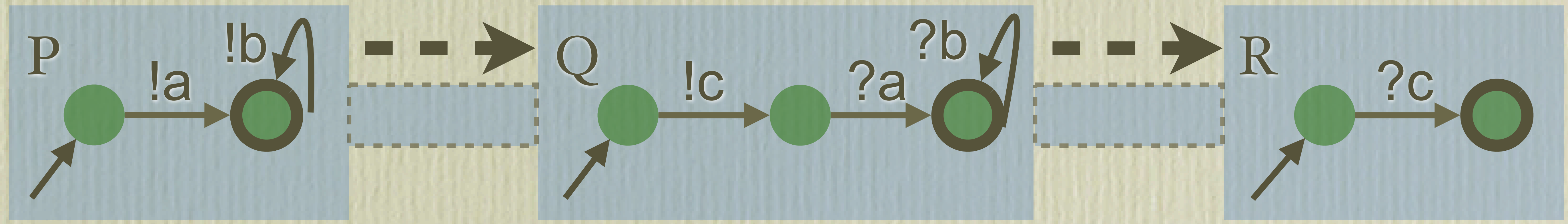
Communicating Finite State Machines

Reachability is in general *undecidable* [Brand&Zafiropulo].
Decidabile restrictions:

- Bound the channel ($\rightarrow$ finite state system).
- Restrict the message alphabet to a singleton ($\rightarrow$ Petri nets) [Karp&Miller'69].
- Make the channel lossy [Abdulla&Jonsson'96, Cece&Finkel&Iyer'96].
- Restrict to mutex communication [Heussner&Leroux&Muscholl&Sutre'10].
- **Restrict the communication topology** [La Torre&Madhusudan&Parlato'08].

Communicating Finite State Machines

Decidability for restricted topologies. Topology: $P \to Q \to R$



Communicating Finite State Machines

Decidability for restricted topologies.

Theorem [La Torre&Madhusudan&Parlato'08].
Reachability for CFSMs is decidable iff the topology is a **polyforest**.
(no undirected cycle)

Examples of polyforest:

$P \to Q \to R$

$$\begin{tikzcd} & & R \\ P \arrow[r] & Q \arrow[ur] & S \arrow[l] \end{tikzcd}$$

Non-example:

$$\begin{tikzcd} & Q \arrow[dr] & \\ P \arrow[ur] \arrow[rr] & & R \end{tikzcd}$$

Theorem [La Torre&Madhusudan&Parlato'08].
Reachability is PSPACE-complete on polyforest topologies.

Our aim: Extend this results to models with **time**.

Communicating FSMs + Time

Discrete time

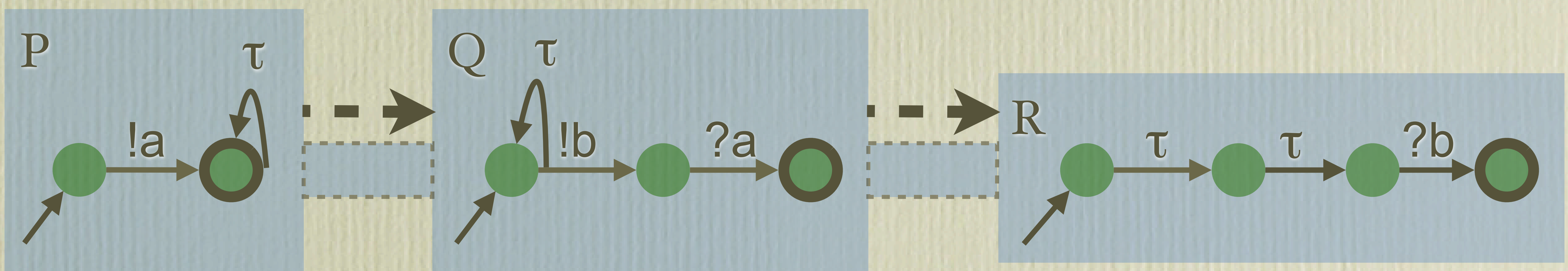
- Each process is a *tick automaton* [Gruber&Holzer&Kiehn&Koenig]
- Add a synchronising action τ
- All processes perform τ at the same time

Dense time

- Each process is a *timed automaton* [Alur&Dill'94]
- Clocks are local to each process
- All clocks evolve at the same rate

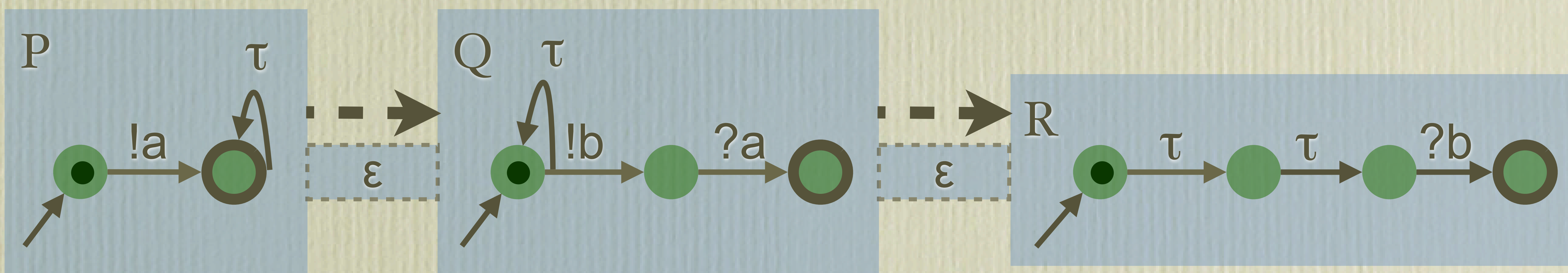
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



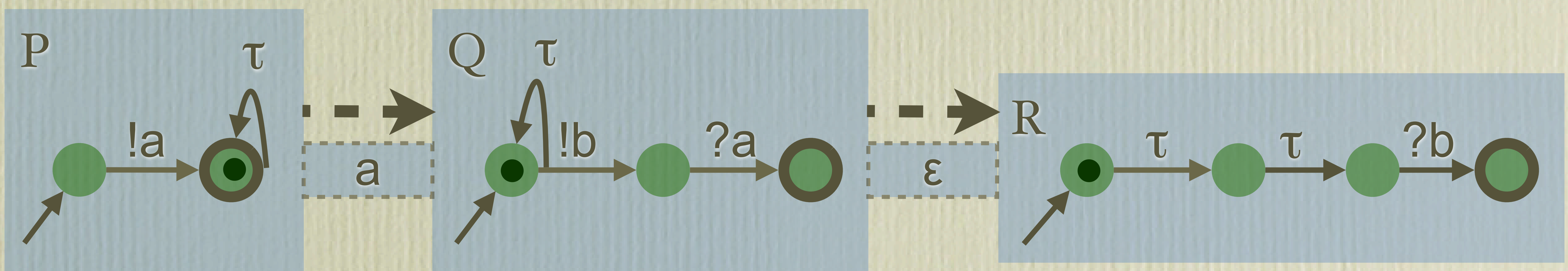
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



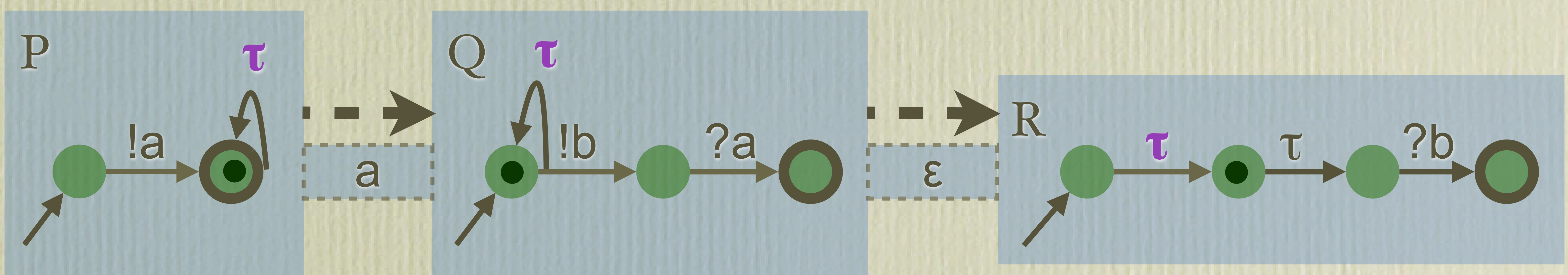
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



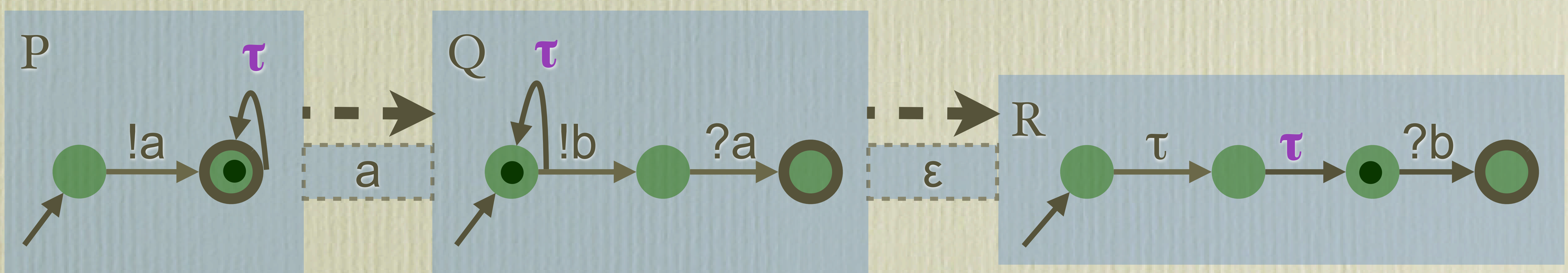
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



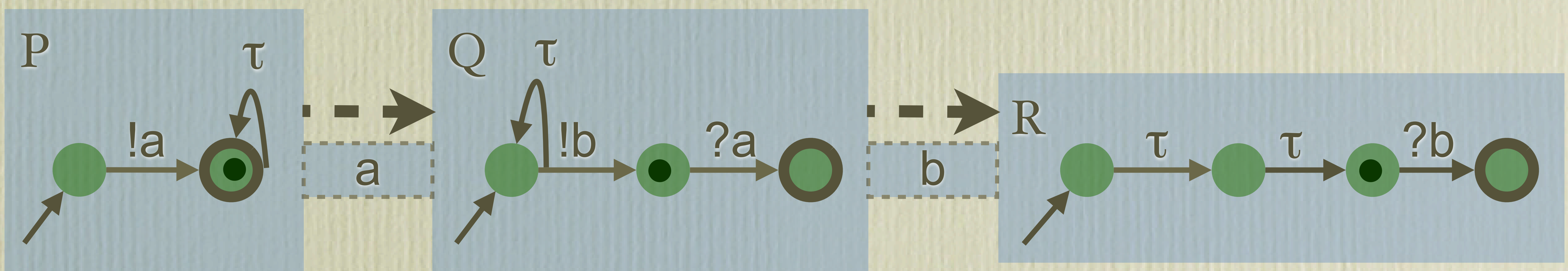
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



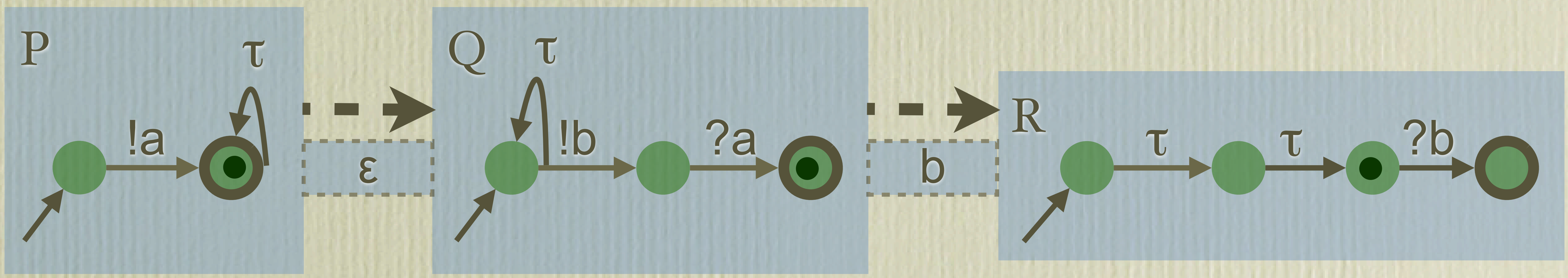
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



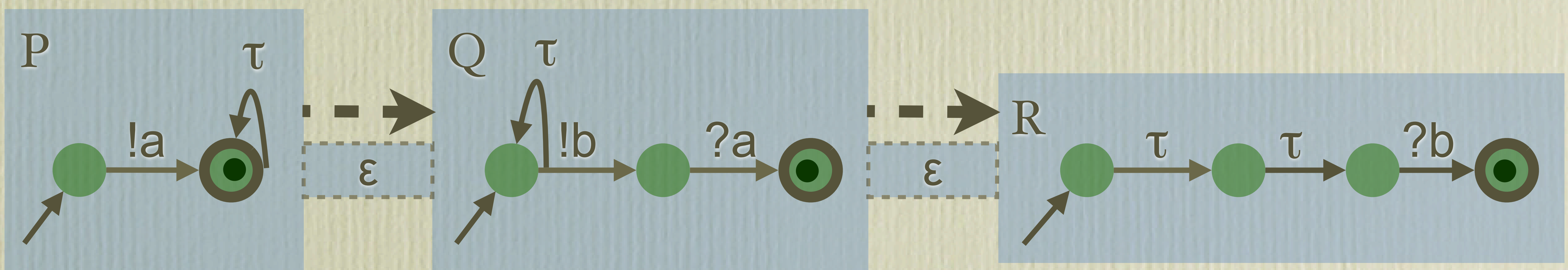
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



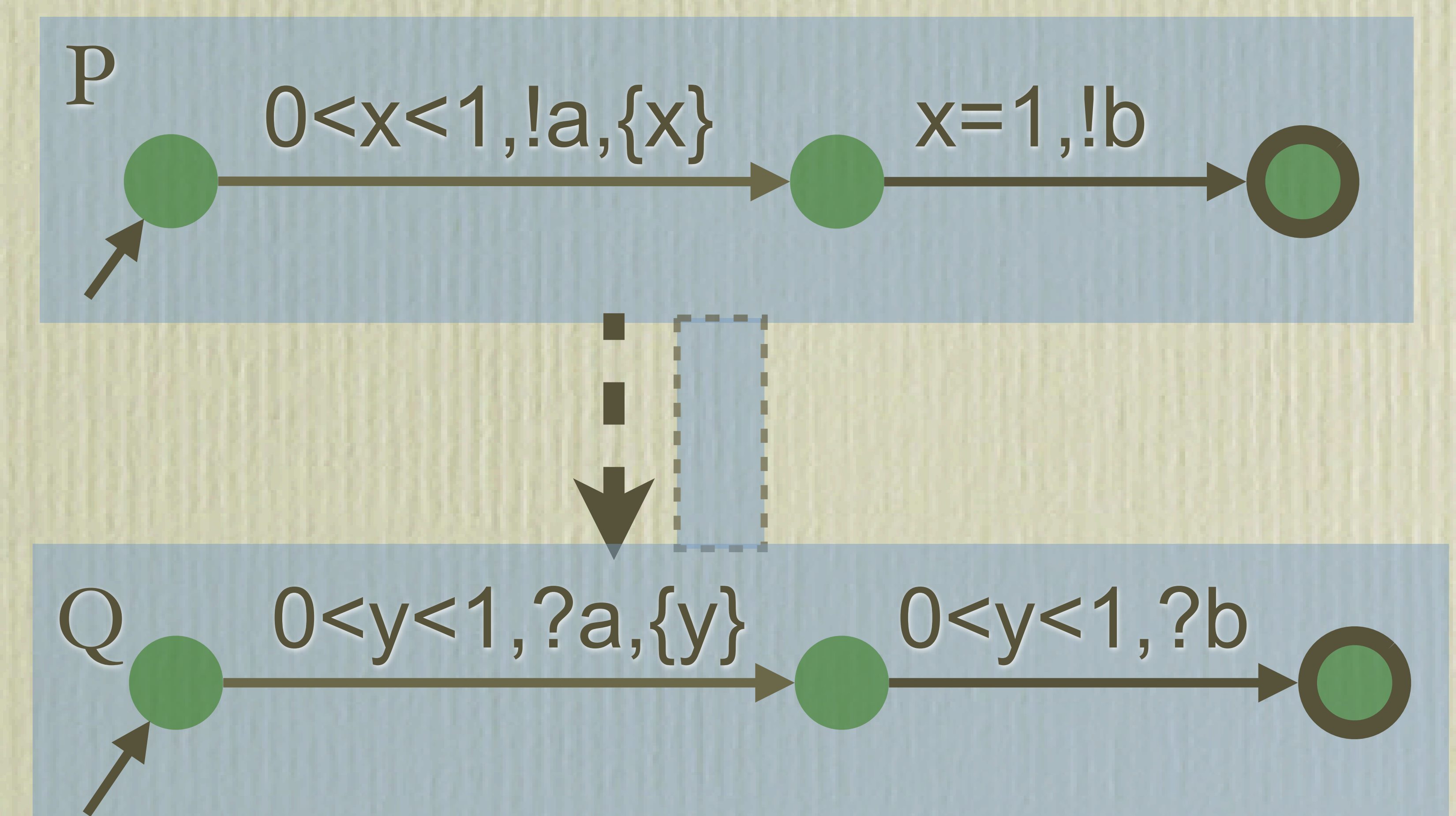
Discrete time

- Add a synchronising action τ
- All processes perform τ at the same time



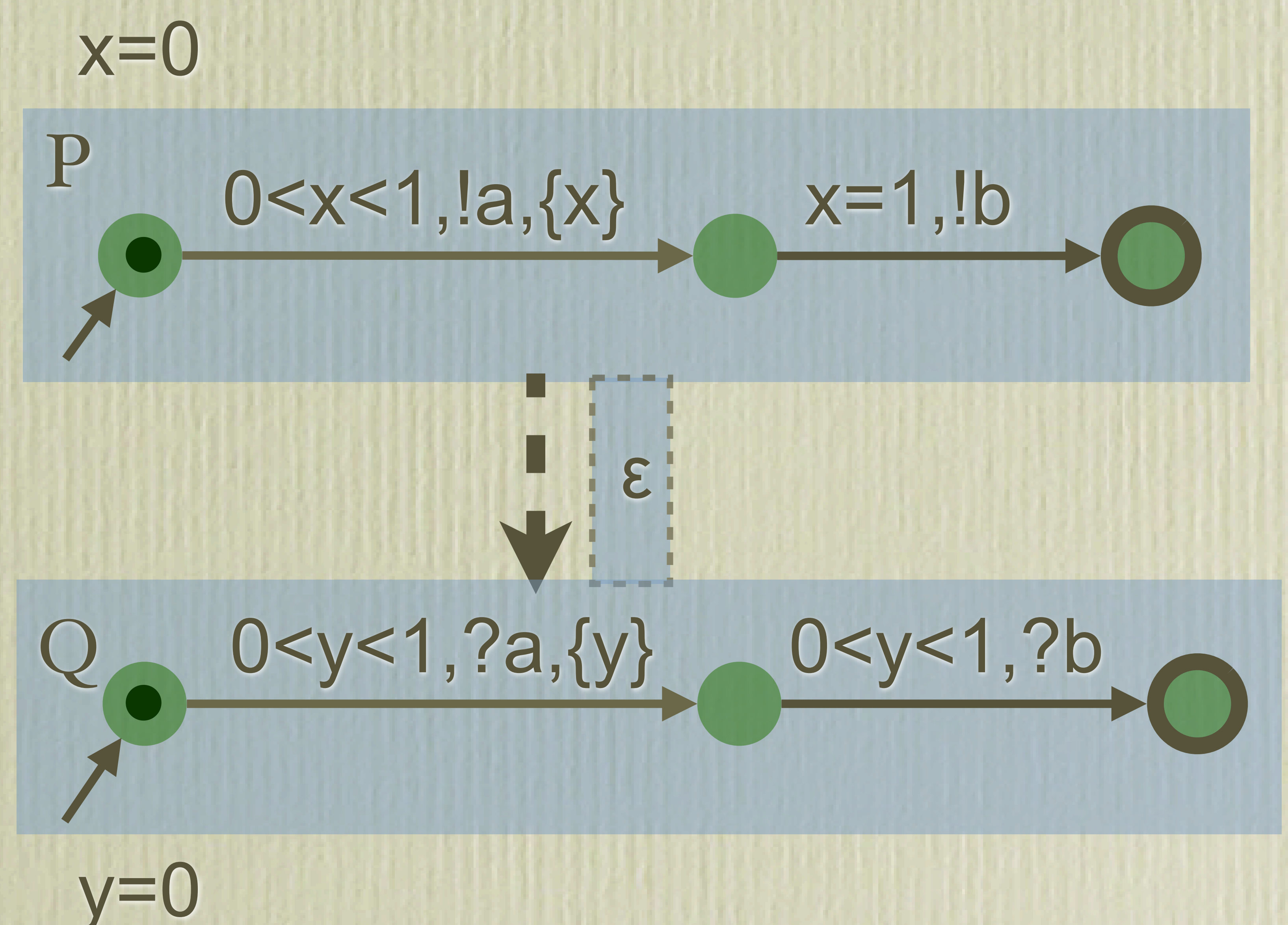
Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



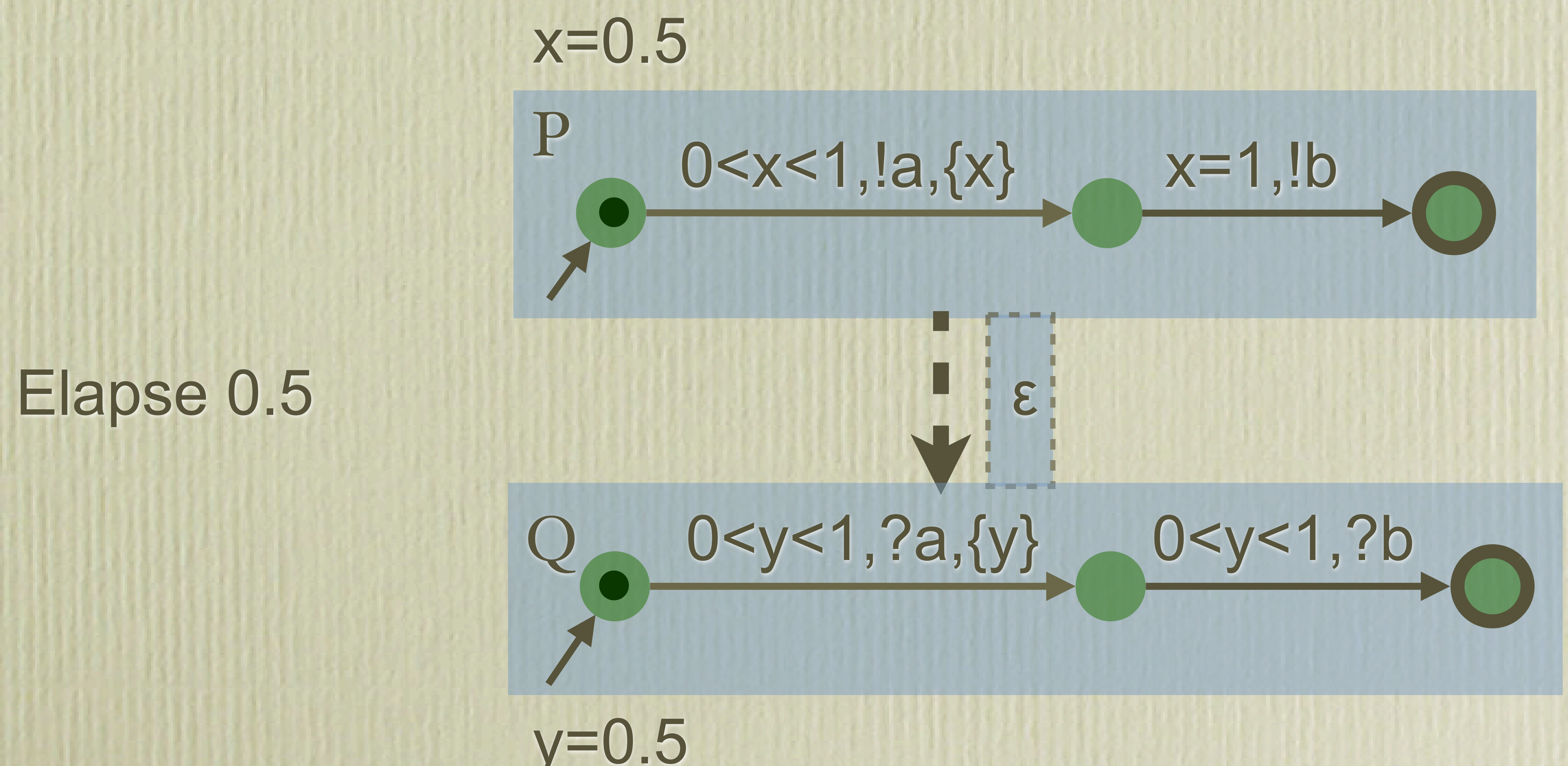
Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



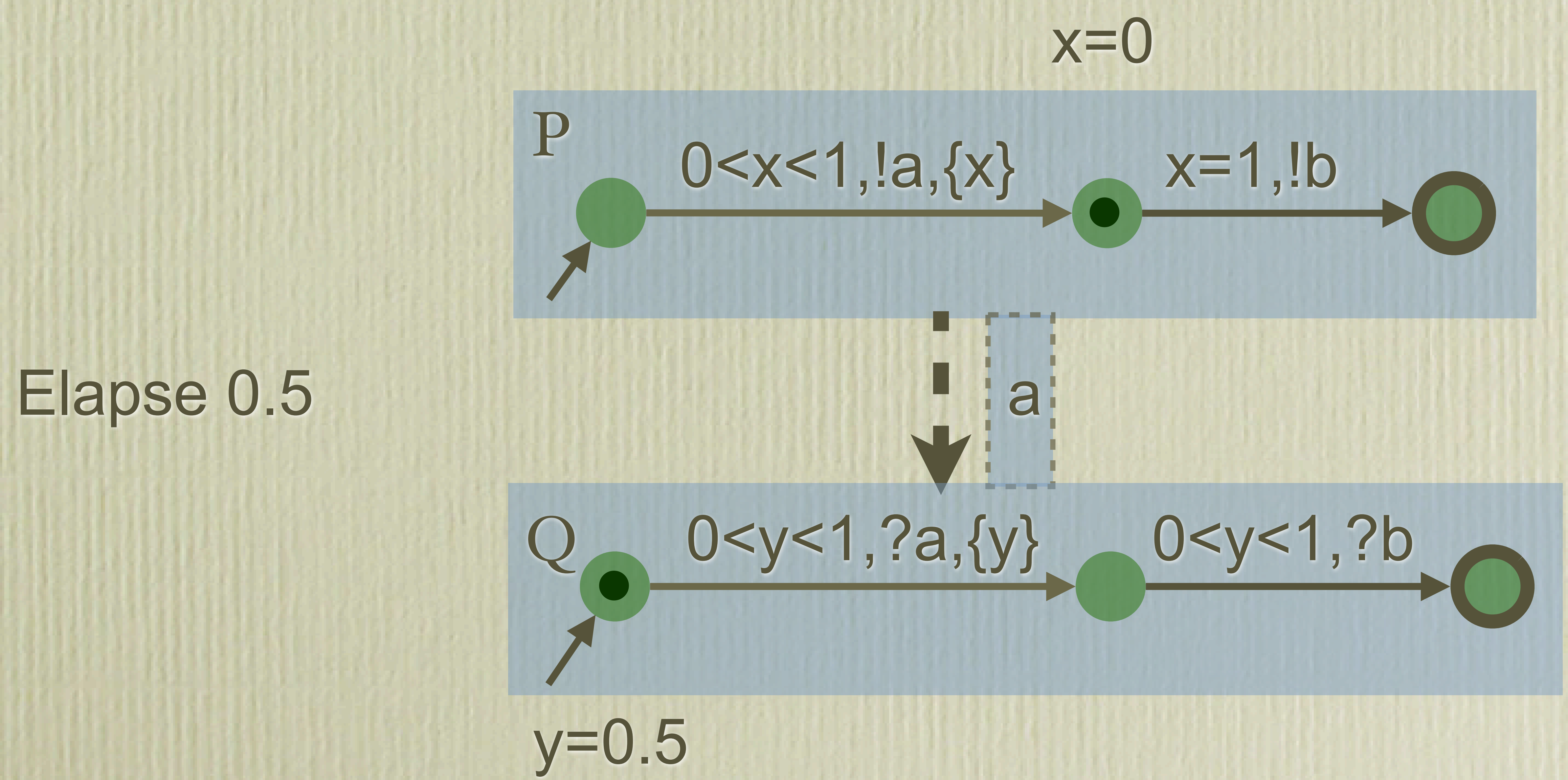
Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



Dense time

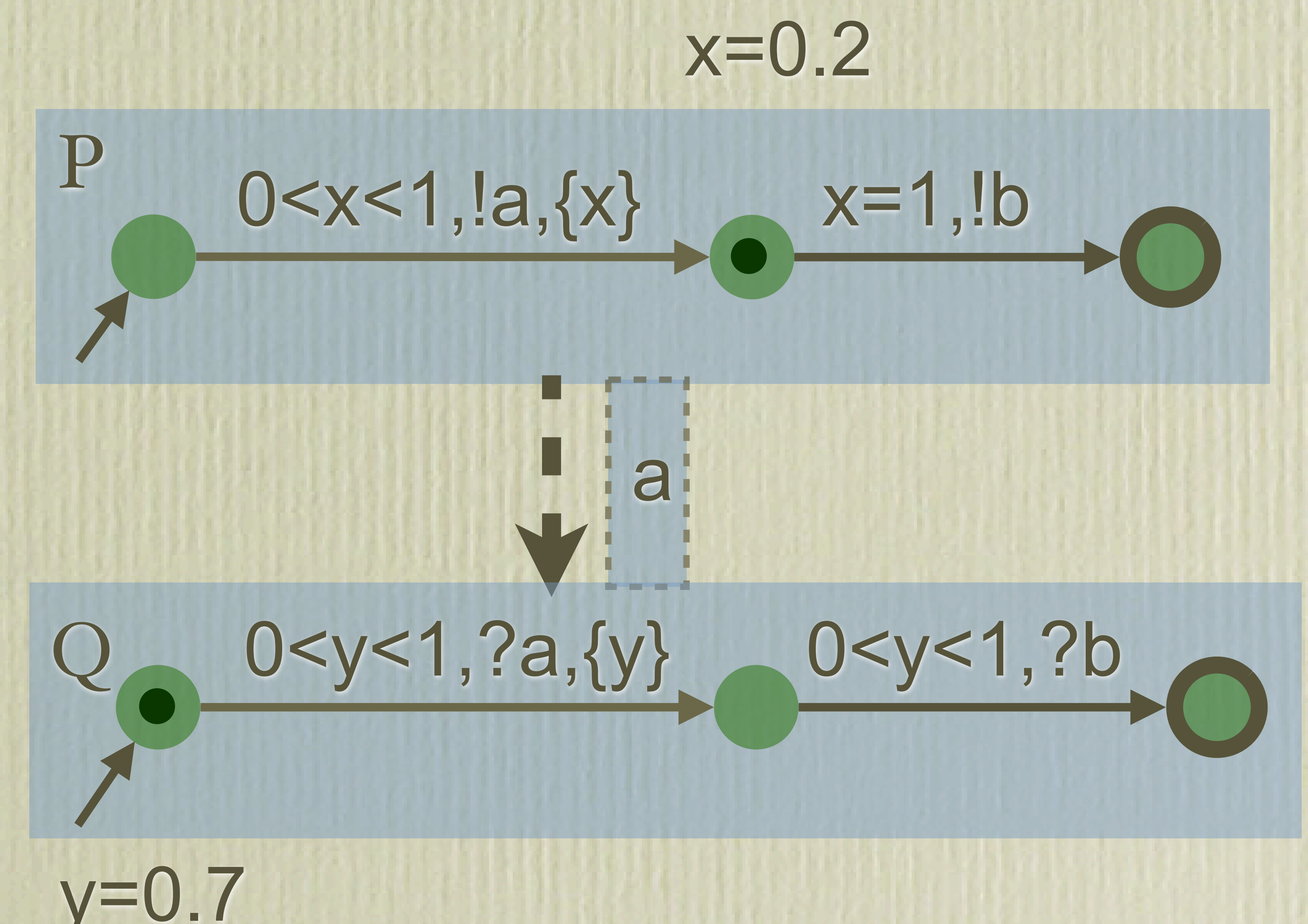
- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



Dense time

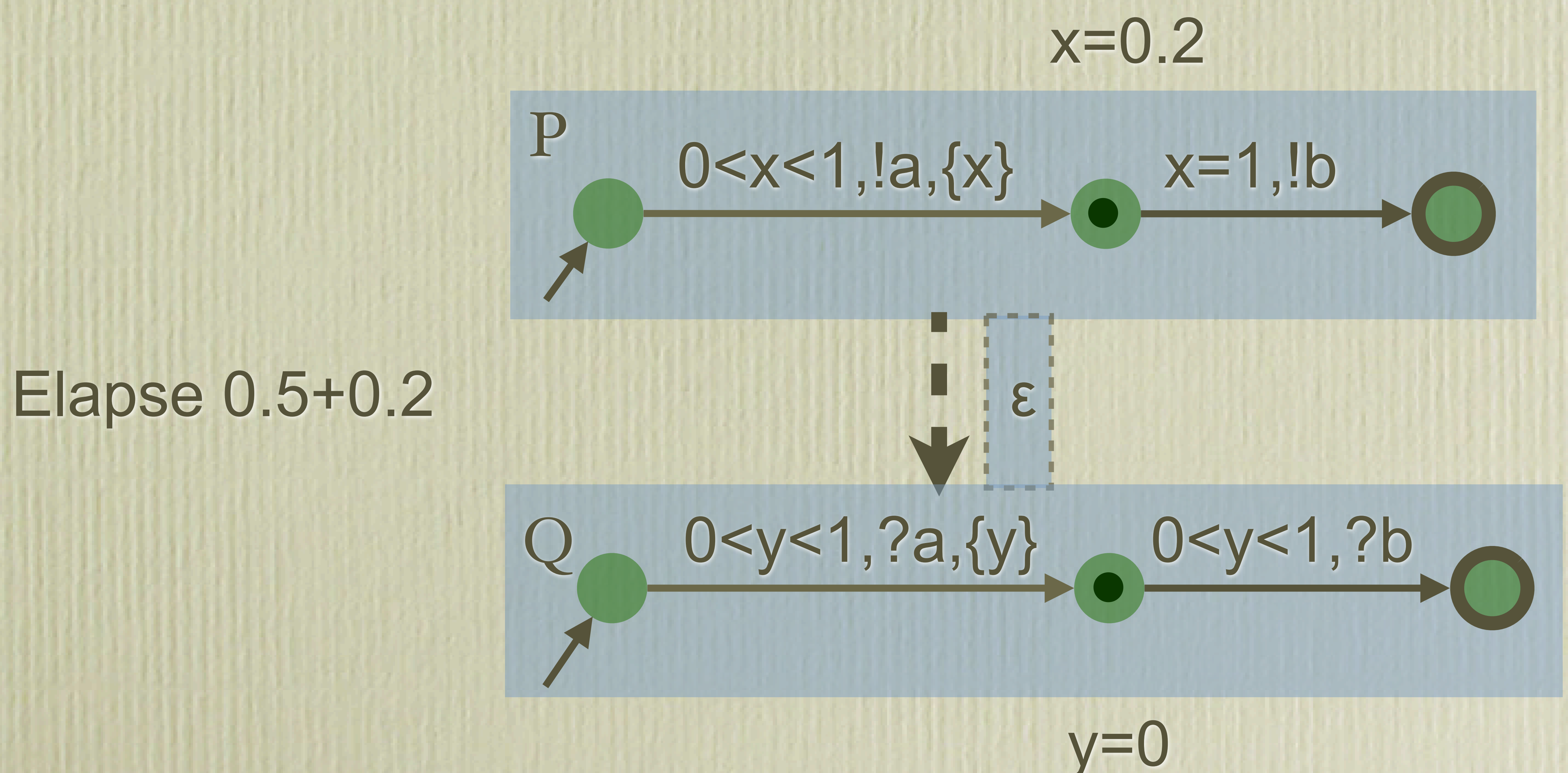
- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate

Elapse 0.5+0.2



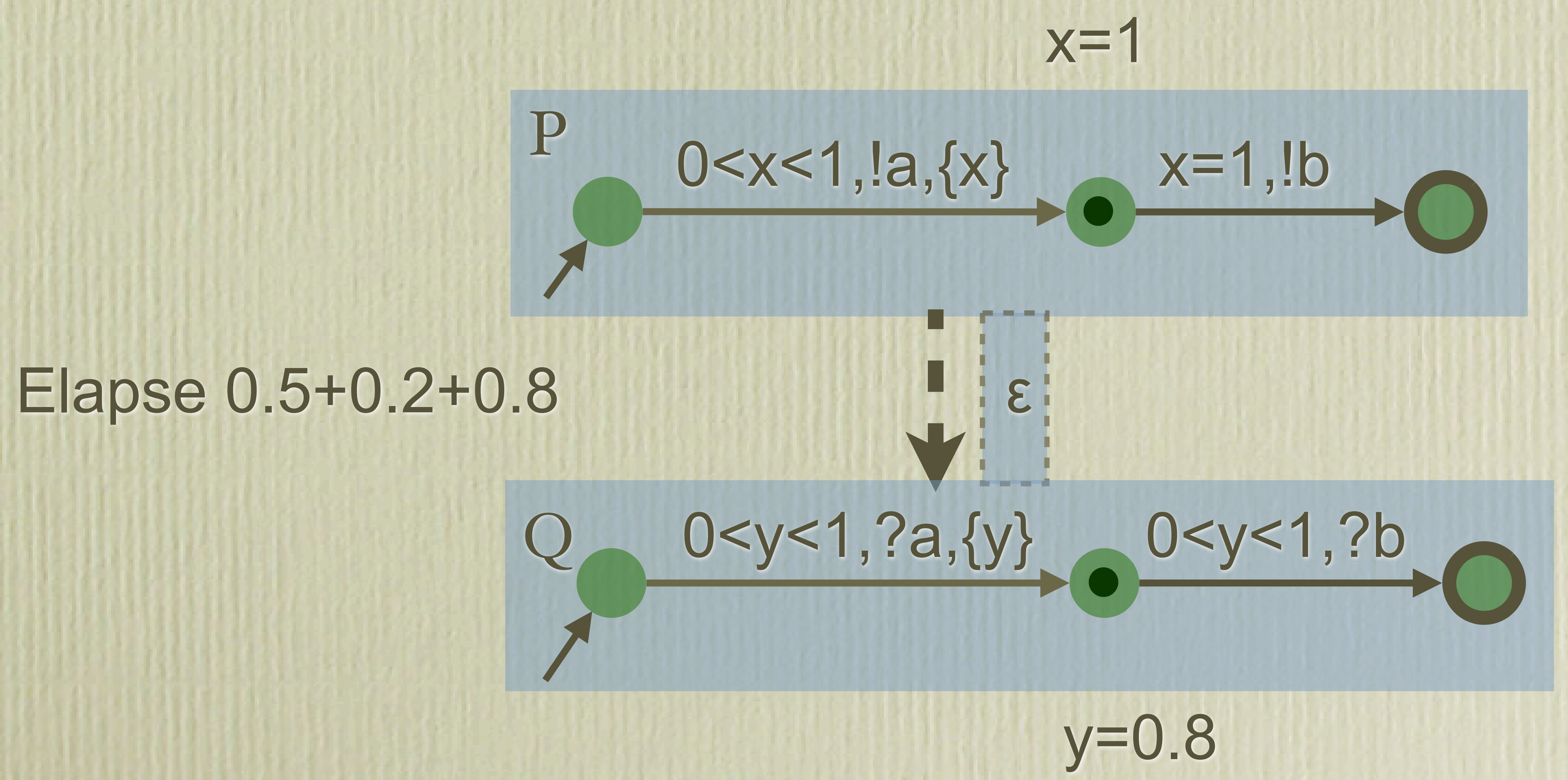
Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



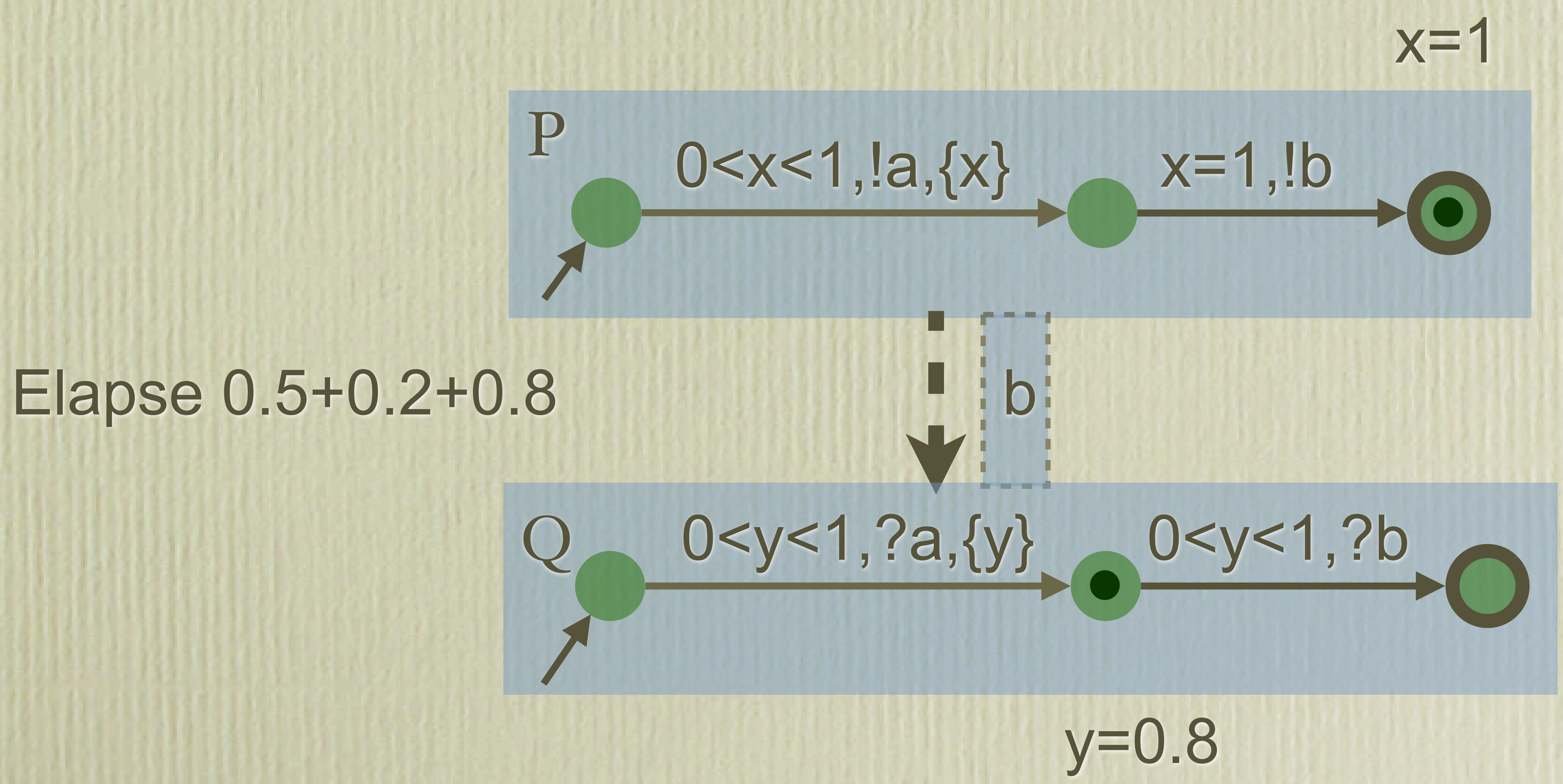
Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



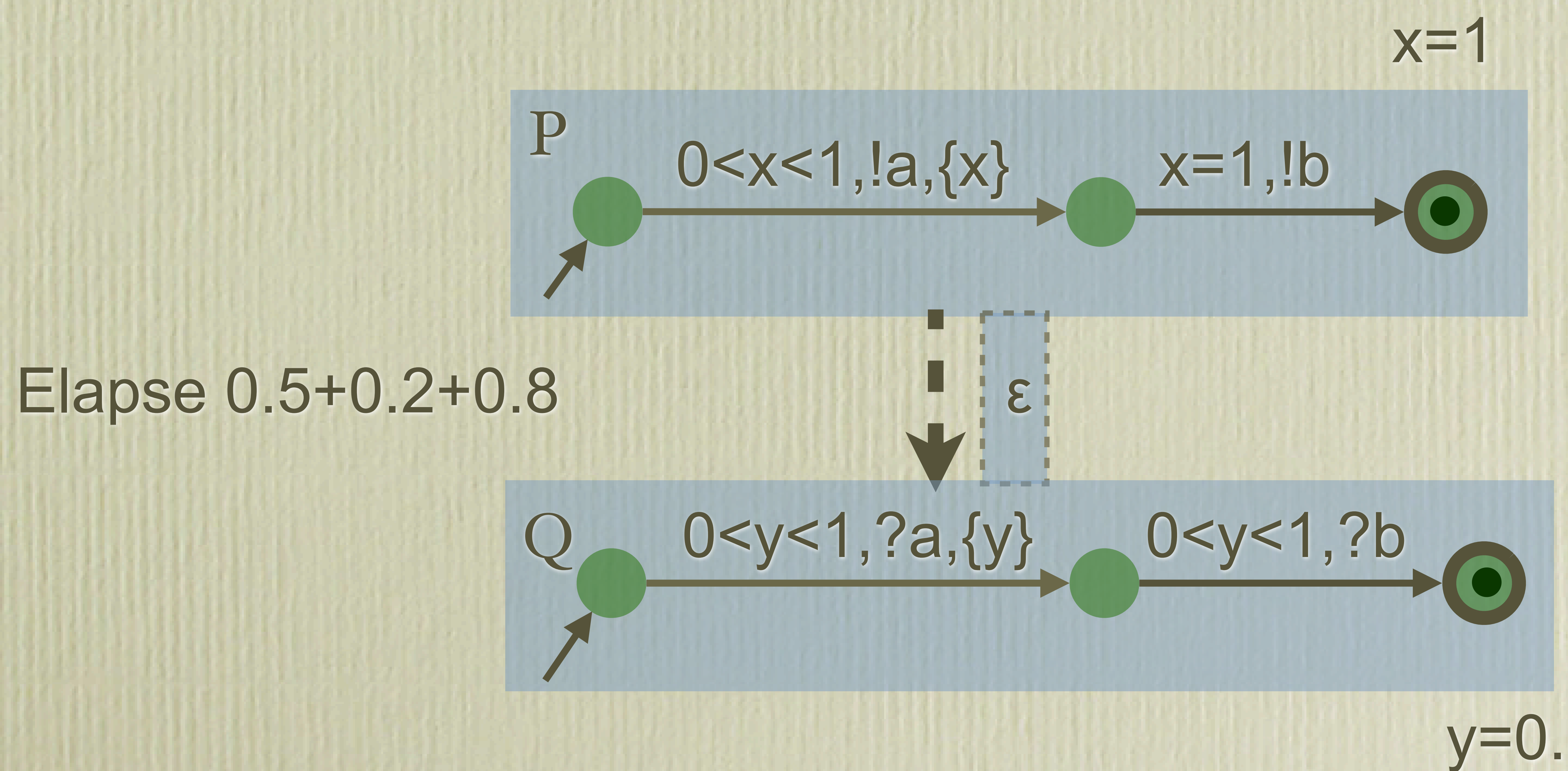
Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



Dense time

- Each process is a timed automaton with local clocks
- All clocks evolve at the same rate



Communicating FSMs + Time

Discrete time

- Each process is a *tick automaton* [Gruber&Holzer&Kiehn&Koenig]
- Add a synchronising action τ
- All processes perform τ at the same time

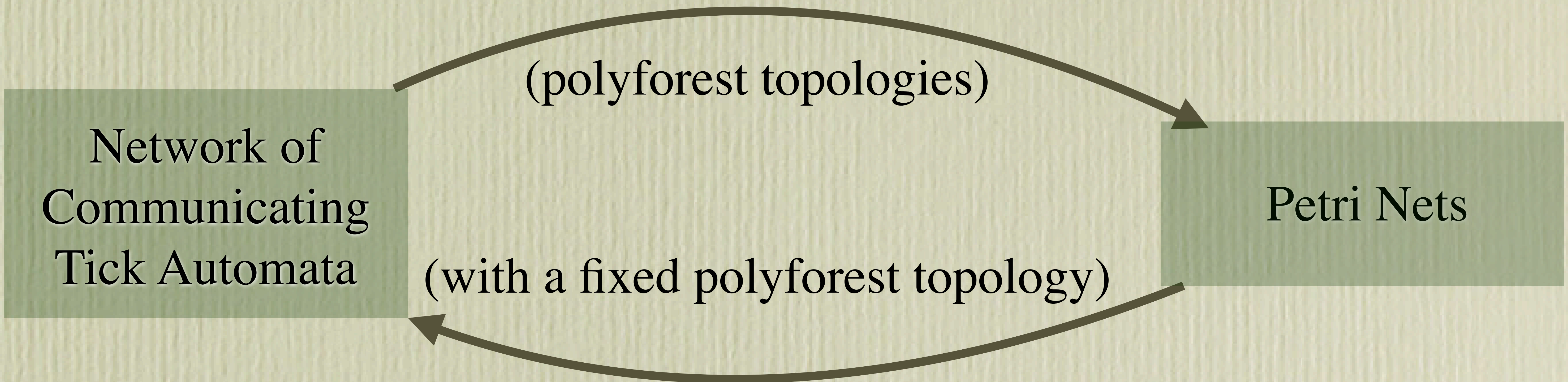
Dense time

- Each process is a *timed automaton* [Alur&Dill'94]
- Clocks are local to each process
- All clocks evolve at the same rate

We study discrete time first

and then extend to dense time

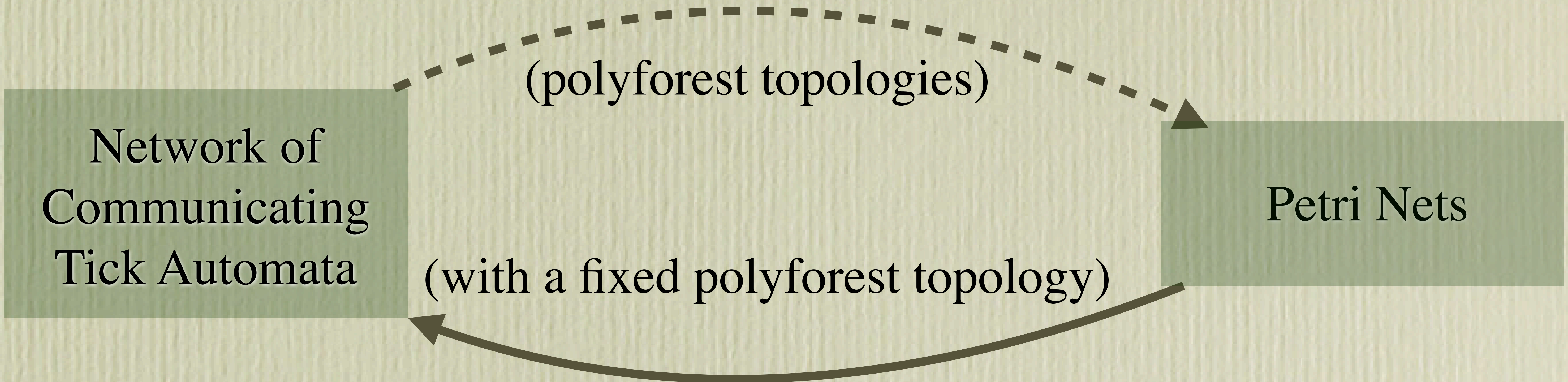
Discrete time



Theorem: Reachability decidable iff polyforest topology

Theorem: The complexity is the same as Petri nets

Discrete time



Theorem: Reachability decidable iff polyforest topology

Theorem: The complexity is the same as Petri nets

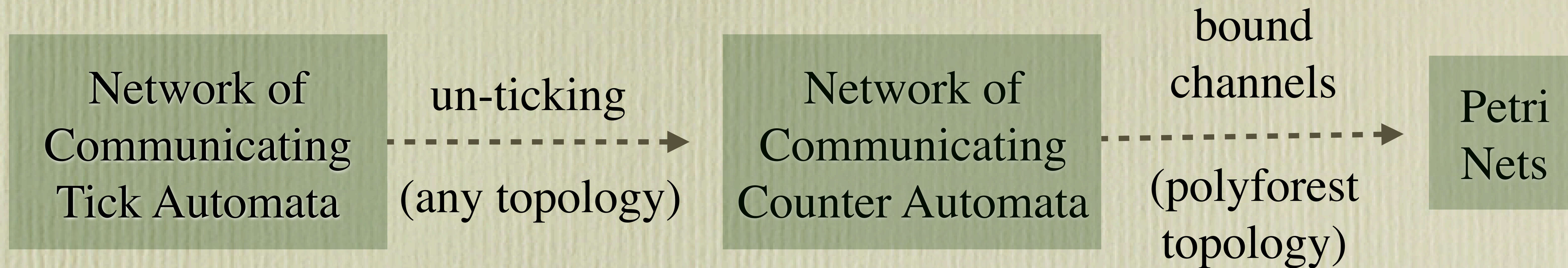
Discrete time

Network of Communicating Tick Automata

(polyforest topologies)

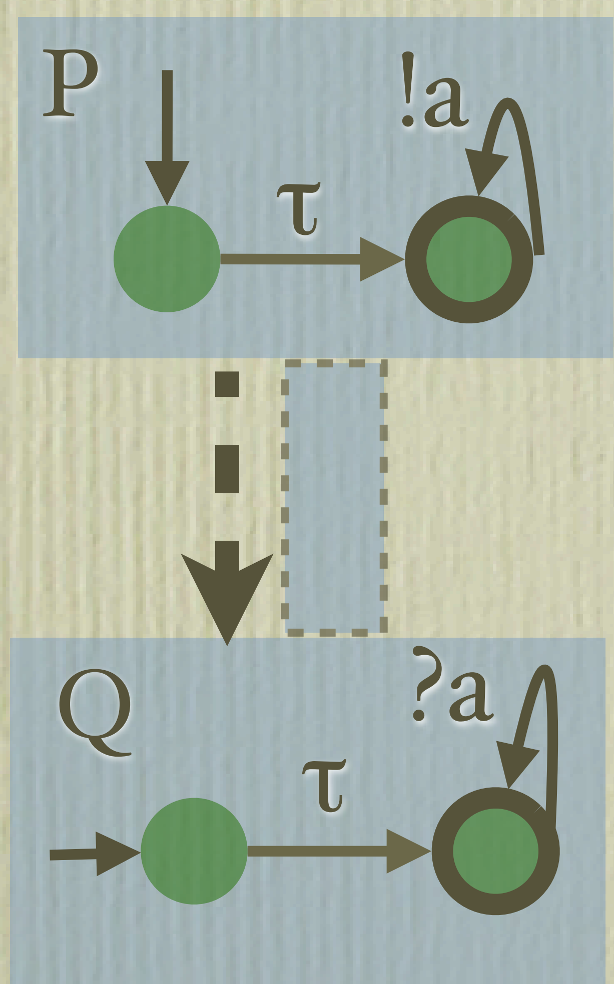
Petri Nets

Discrete time

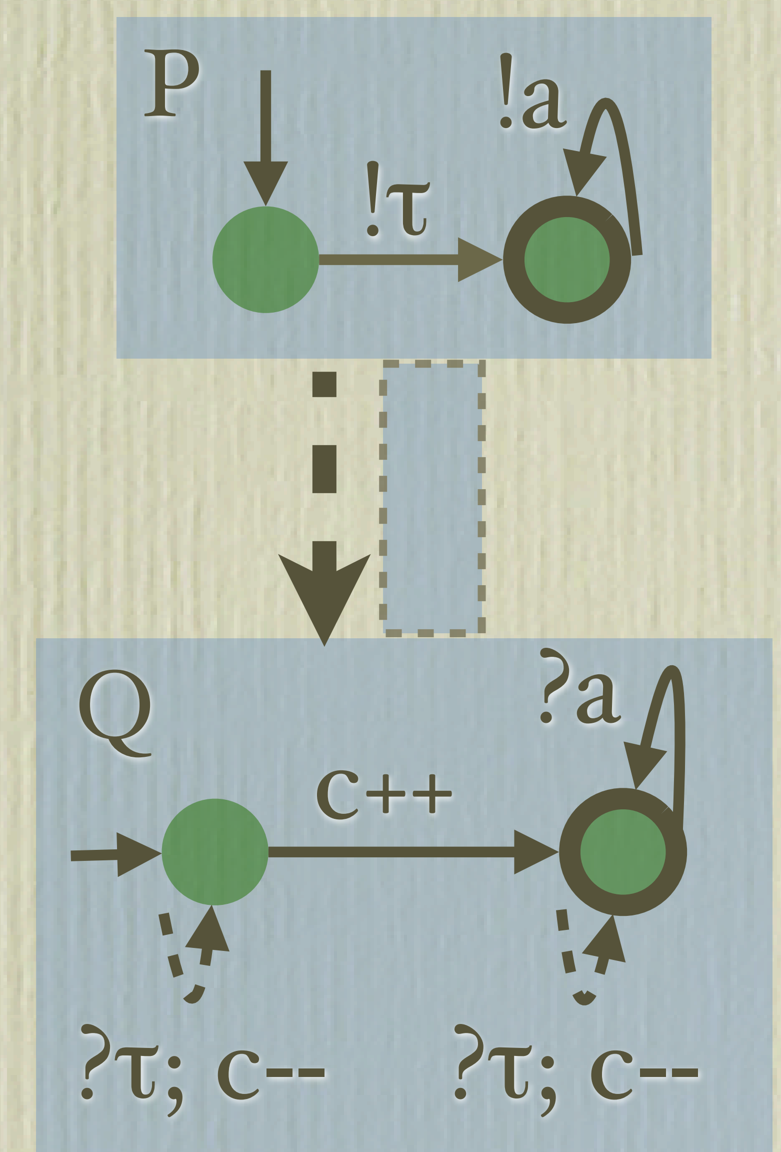


Discrete time

Network of Communicating Tick Automata → un-ticking (any topology) → Network of Communicating Counter Automata → bound channels (polyforest topology) → Petri Nets



un-ticking

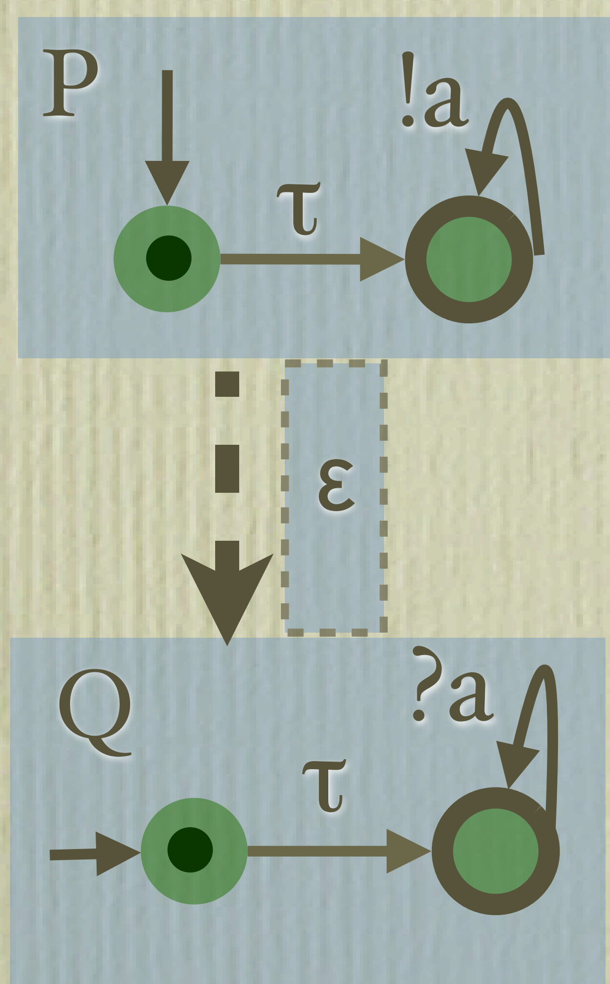


- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

Discrete time

`P:`

`Q:`

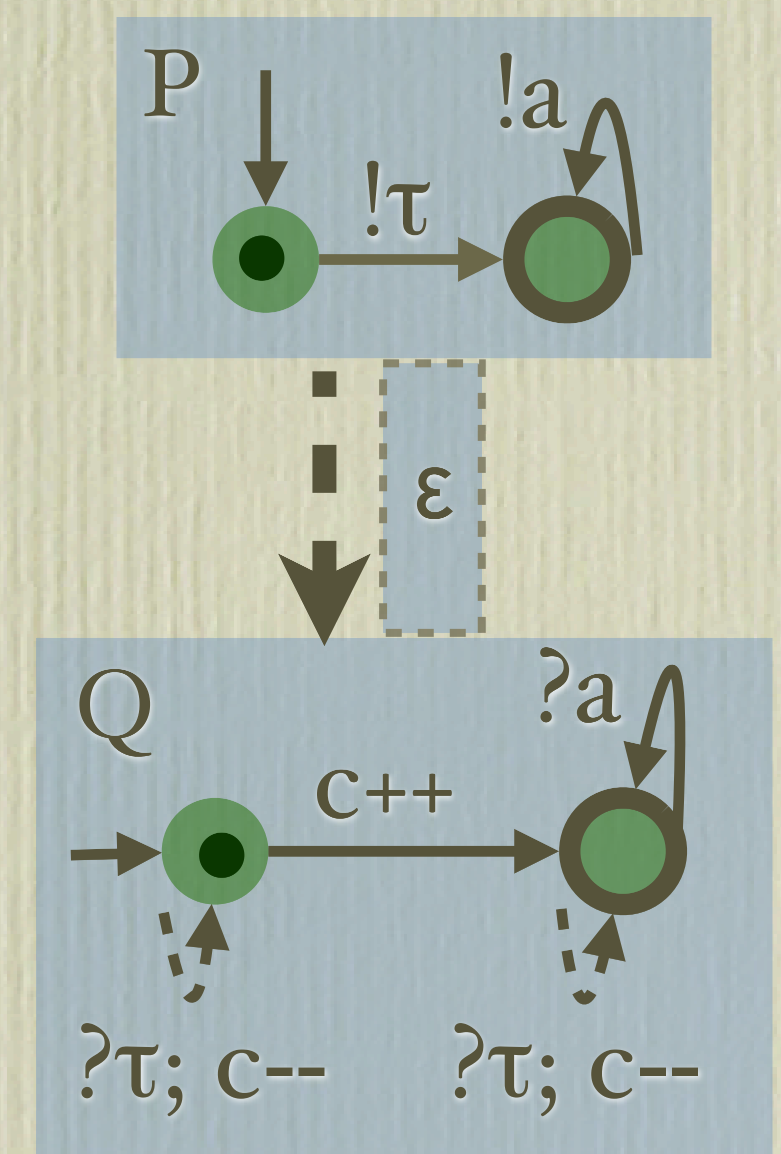


un-ticking

`P:`

`Q:`

`c=0`



- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

Discrete time

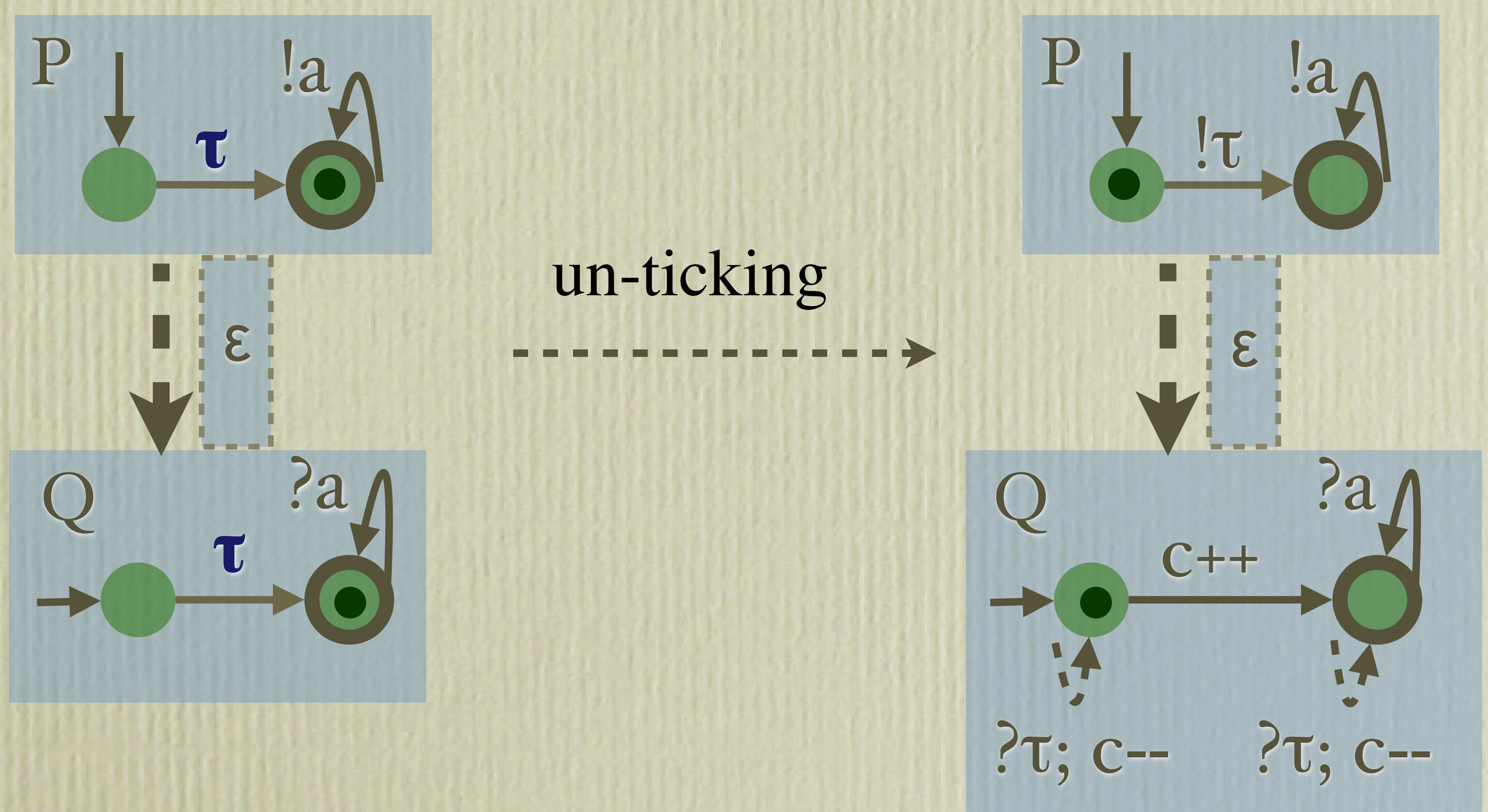
P: τ

Q: τ

P:

Q:

c=0



- P *sends* its ticks
- Q increments a <u>counter</u> instead of doing a tick
- In any state, Q can *receive* a tick

Discrete time

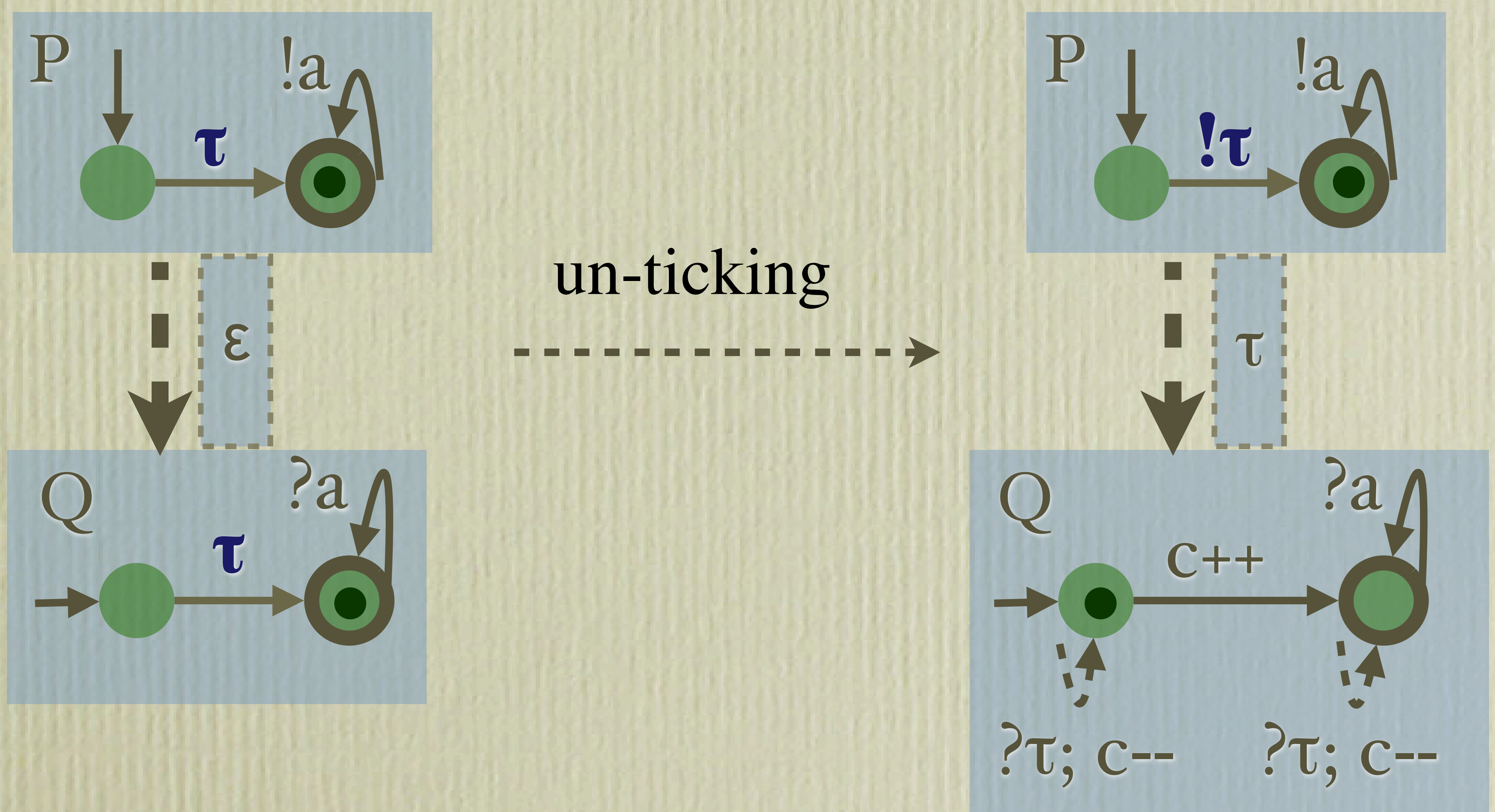
`P: τ`

`Q: τ`

`P: !τ`

`Q:`

`c=0`

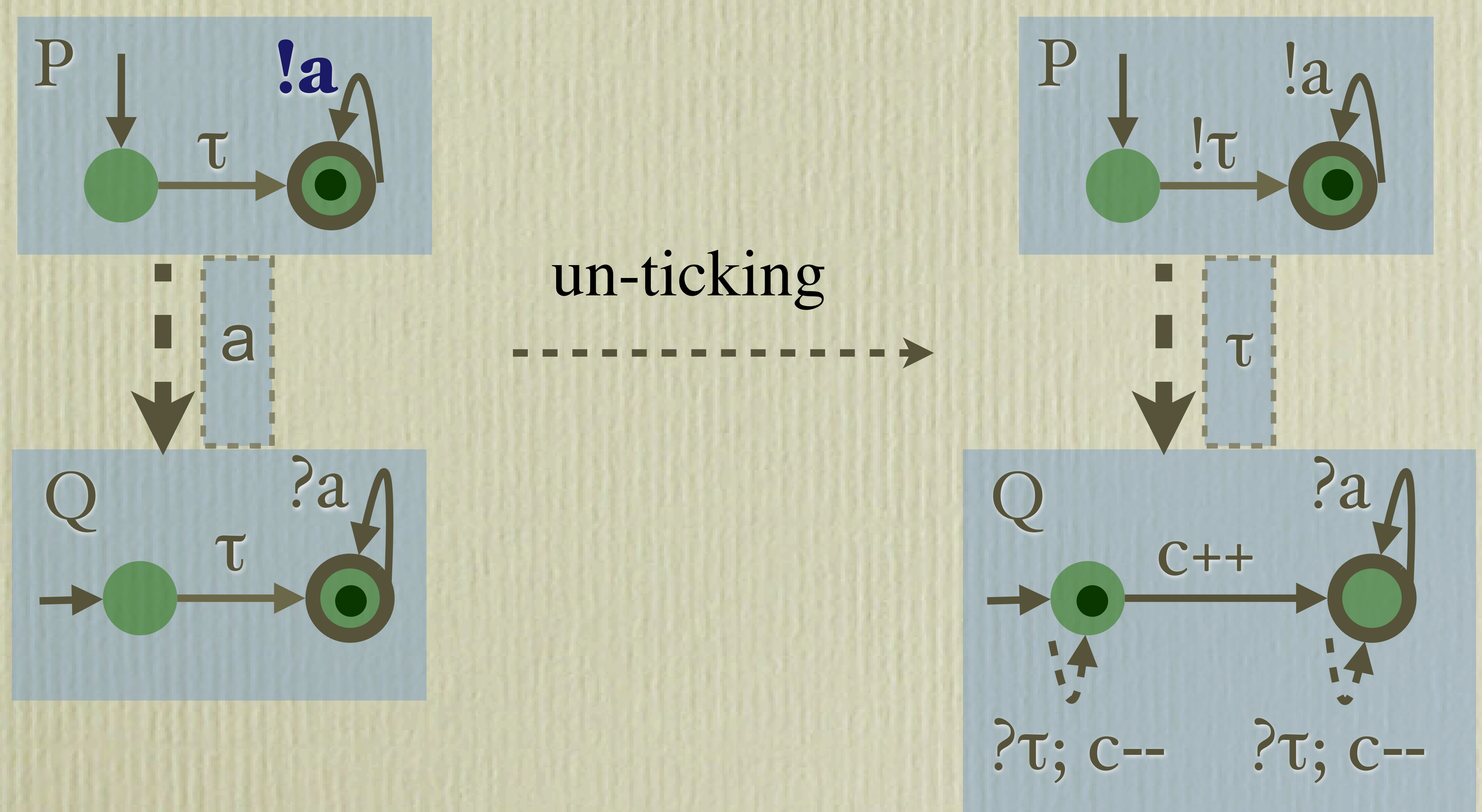


- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

Discrete time

```
P: τ !a
Q: τ
```

```
P: !τ
Q:
c=0
```

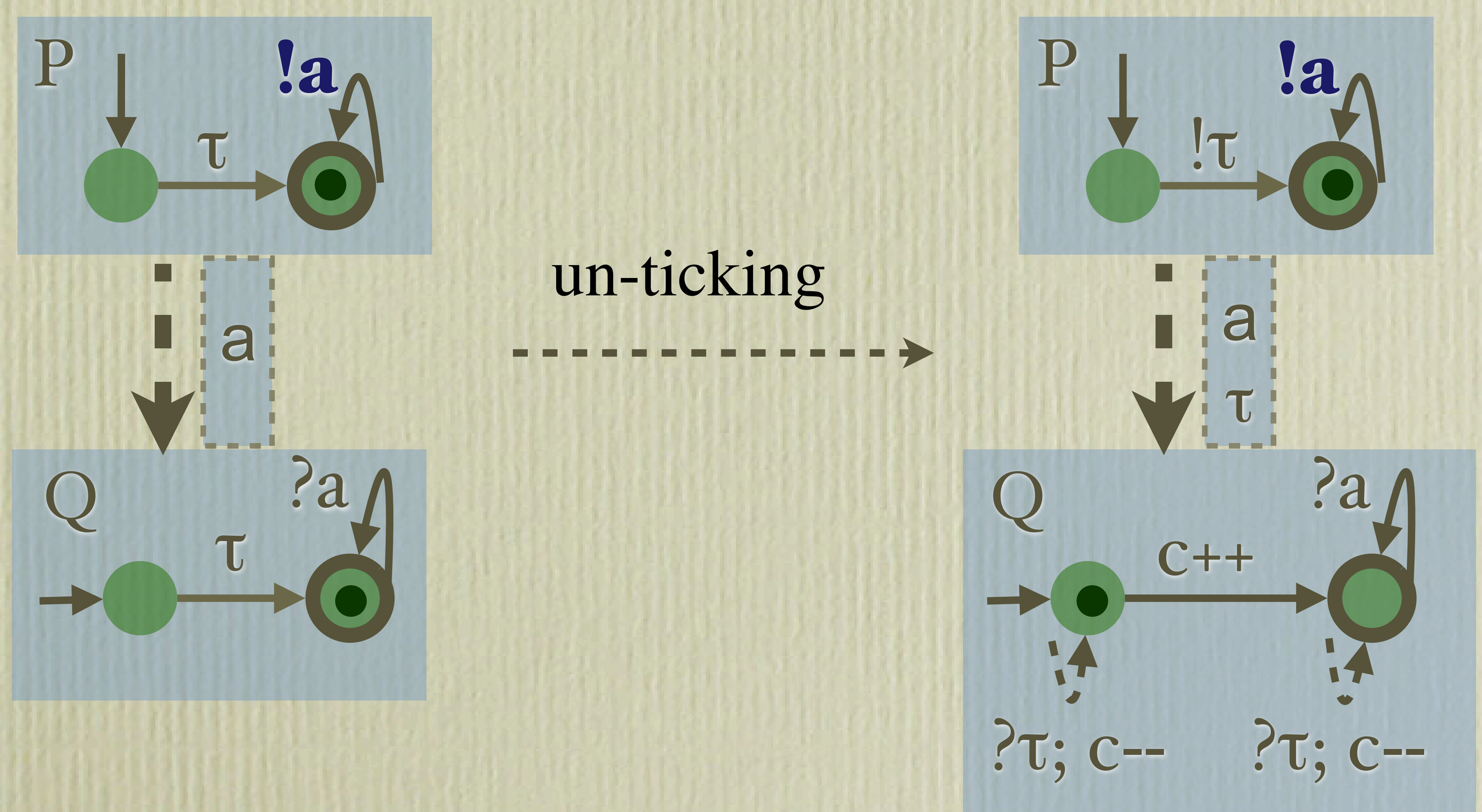


- P *sends* its ticks
- Q increments a <u>counter</u> instead of doing a tick
- In any state, Q can *receive* a tick

Discrete time

```
P: τ !a
Q: τ
```

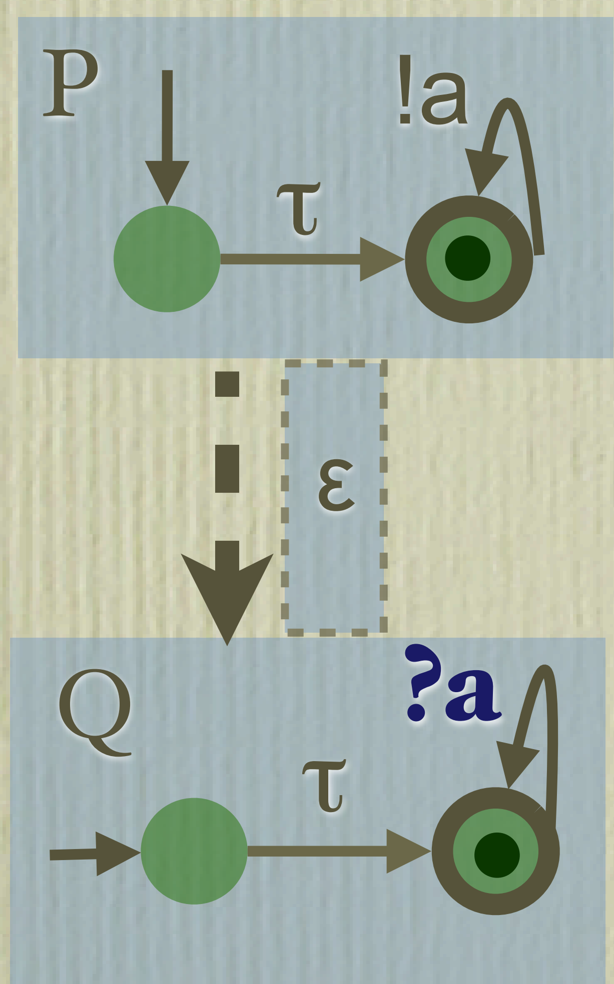
```
P: !τ !a
Q:
c=0
```



- P *sends* its ticks
- Q increments a <u>counter</u> instead of doing a tick
- In any state, Q can *receive* a tick

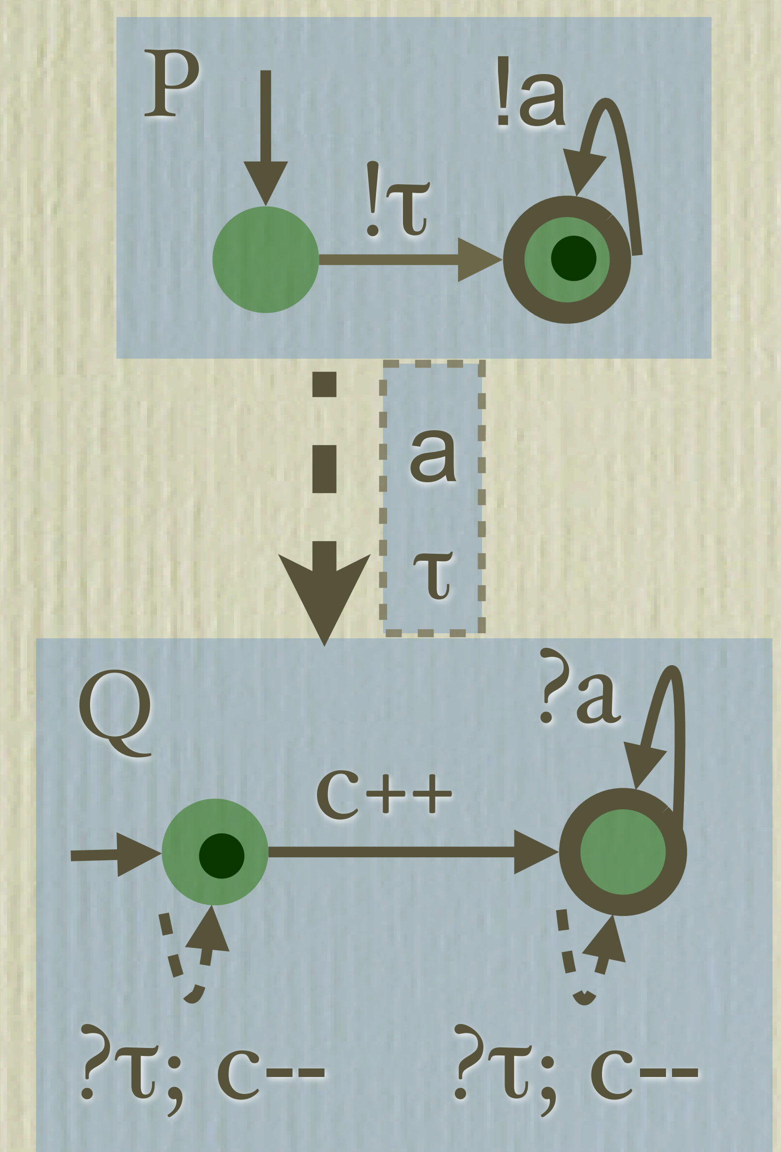
Discrete time

```
P: τ !a
Q: τ    ?a
```



un-ticking

```
P: !τ !a
Q:
c=0
```

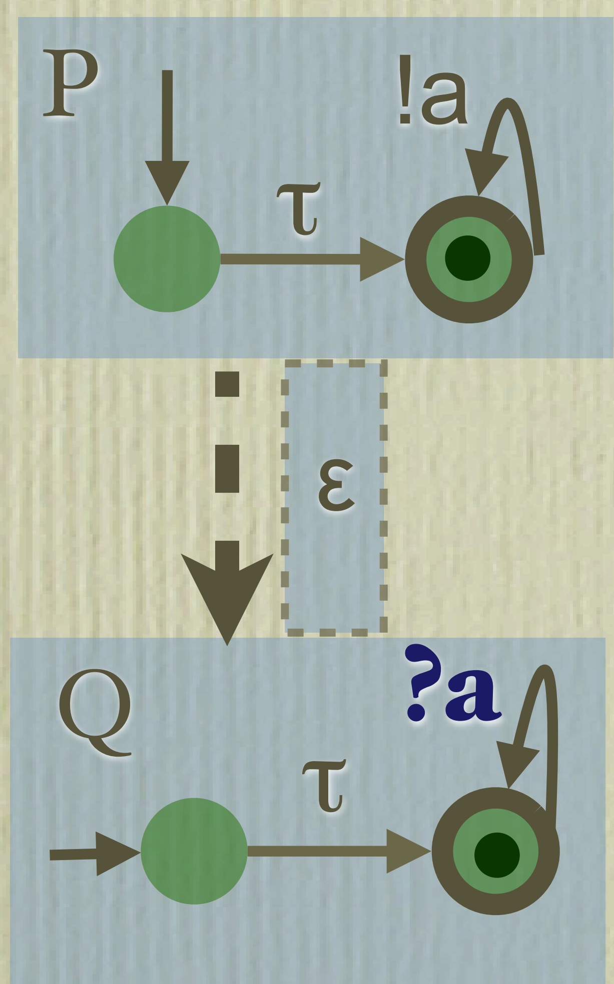


- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

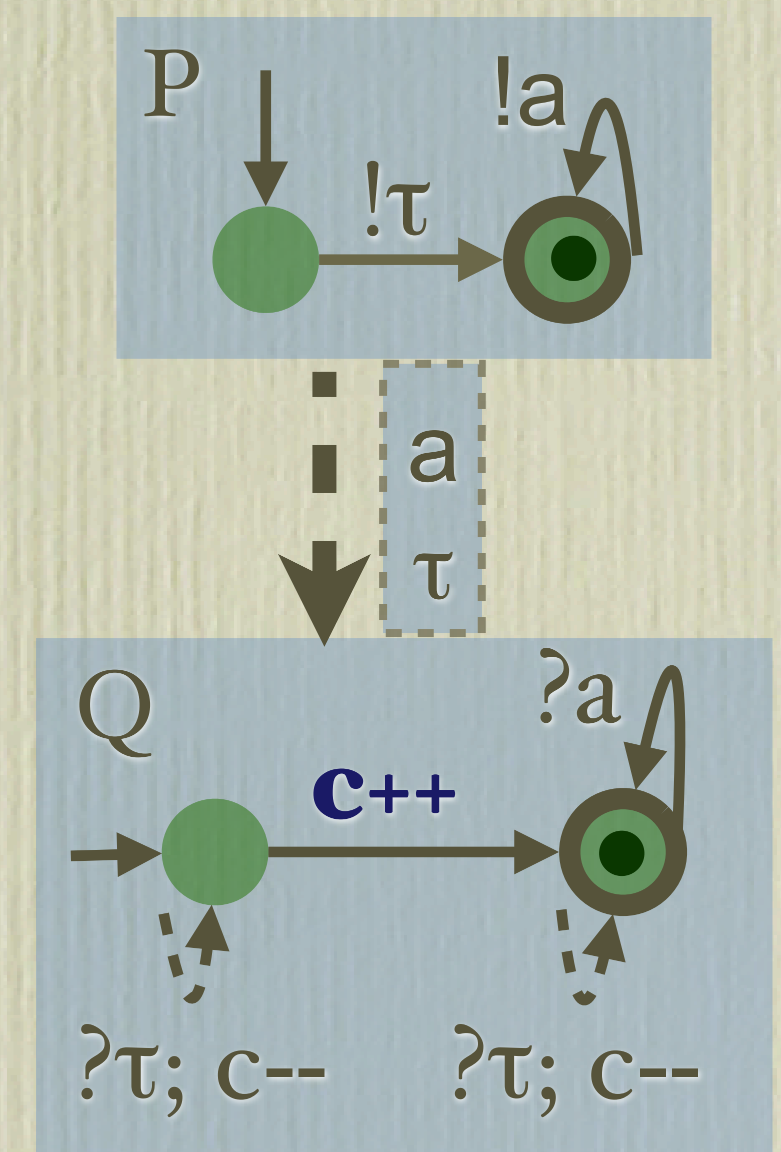
Discrete time

```
P: τ !a
Q: τ    ?a
```

```
P: !τ !a
Q:          c++
c=1
```



un-ticking

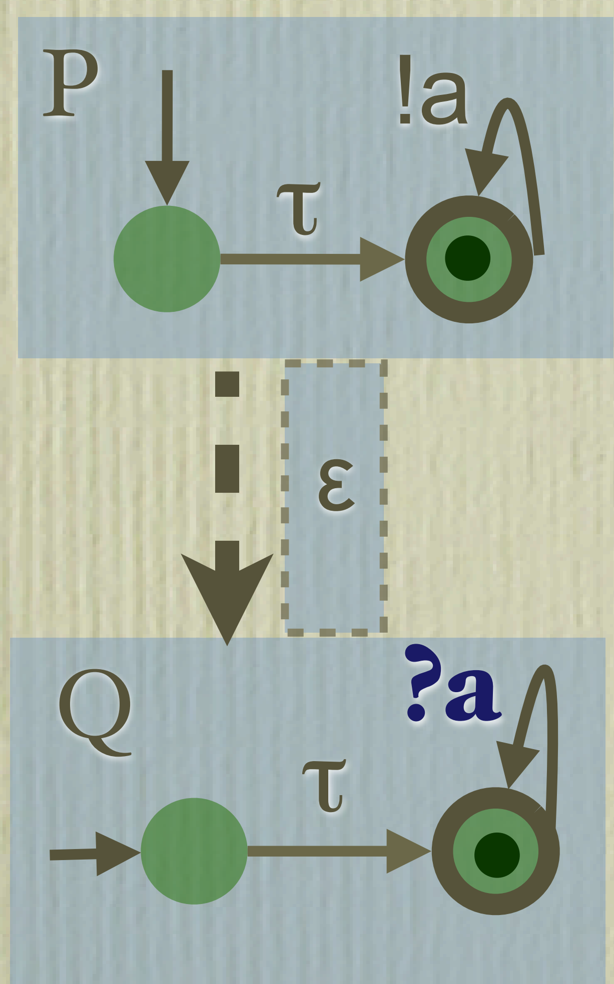


- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

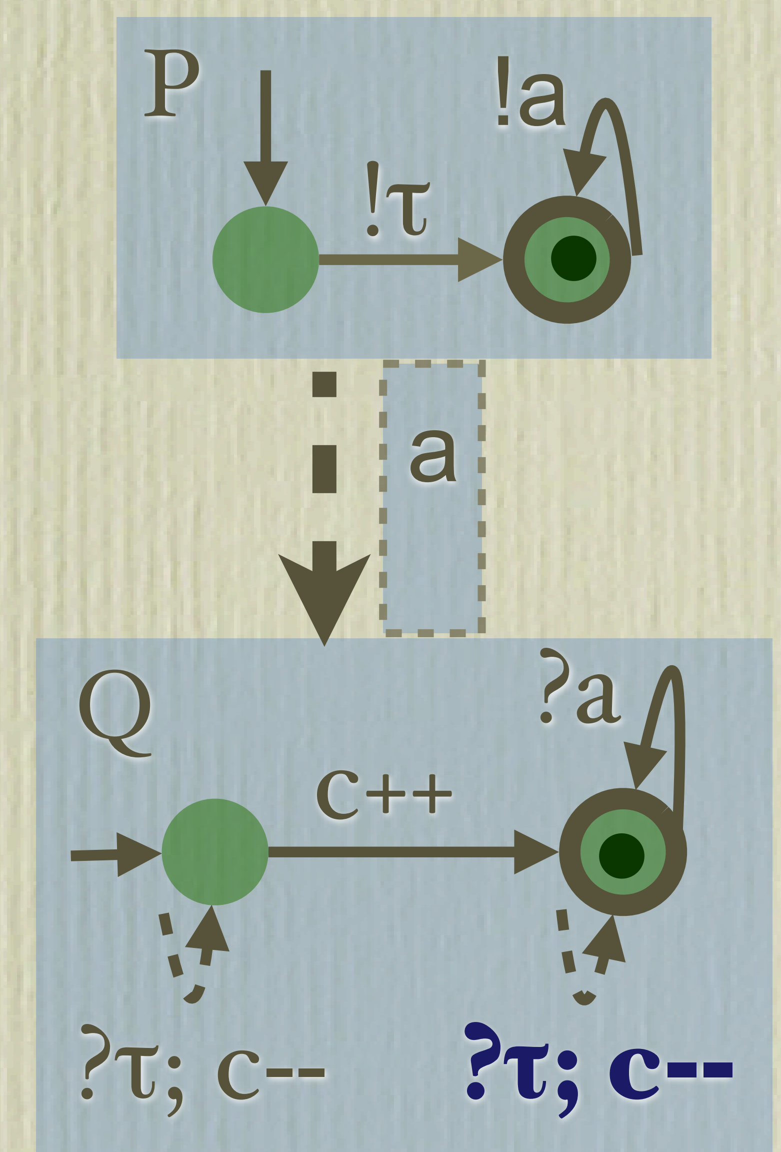
Discrete time

```
P: τ !a
Q: τ     ?a
```

```
P: !τ !a
Q:          c++ ?τ c--
c=0
```



un-ticking

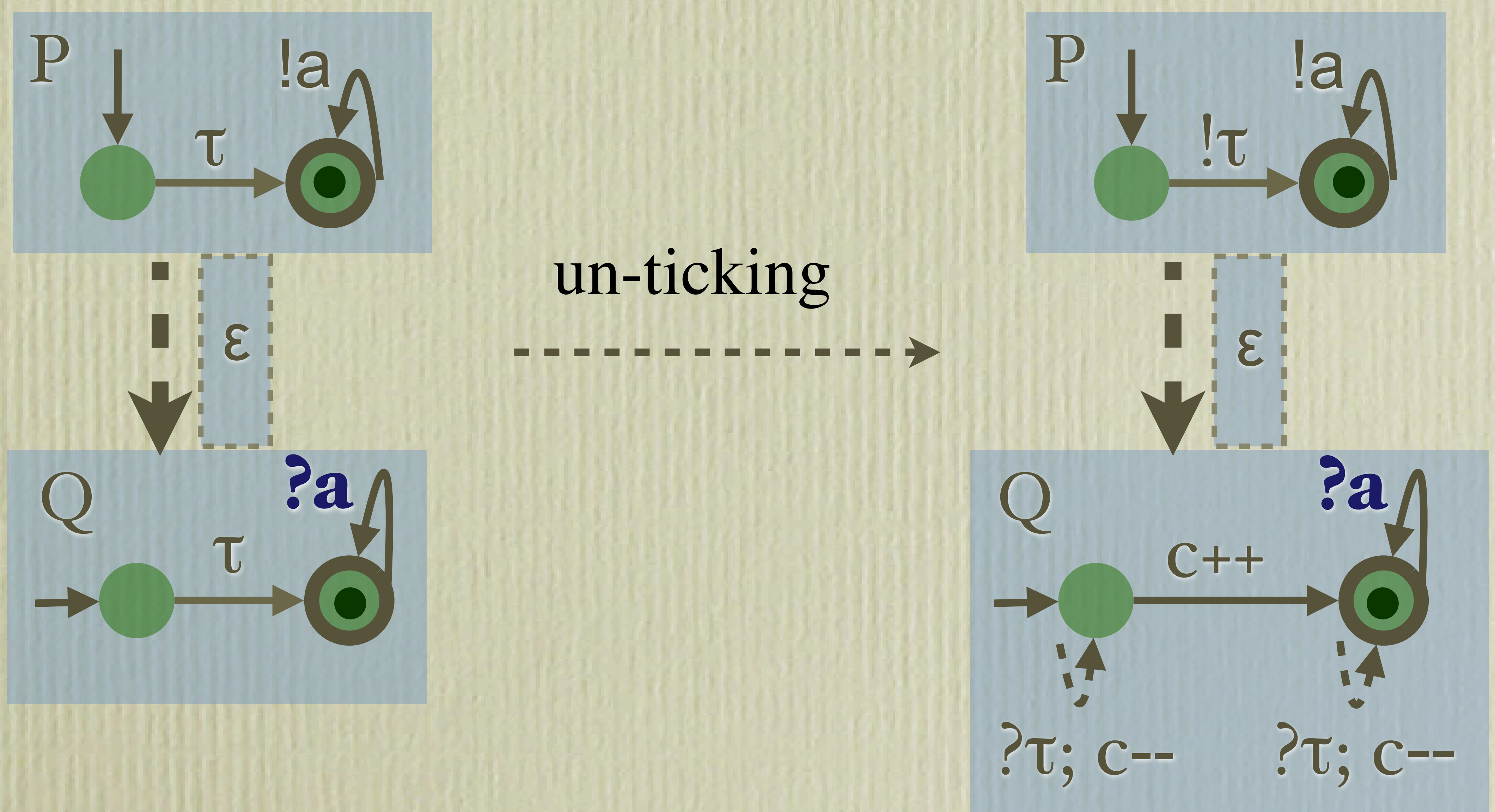


- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

Discrete time

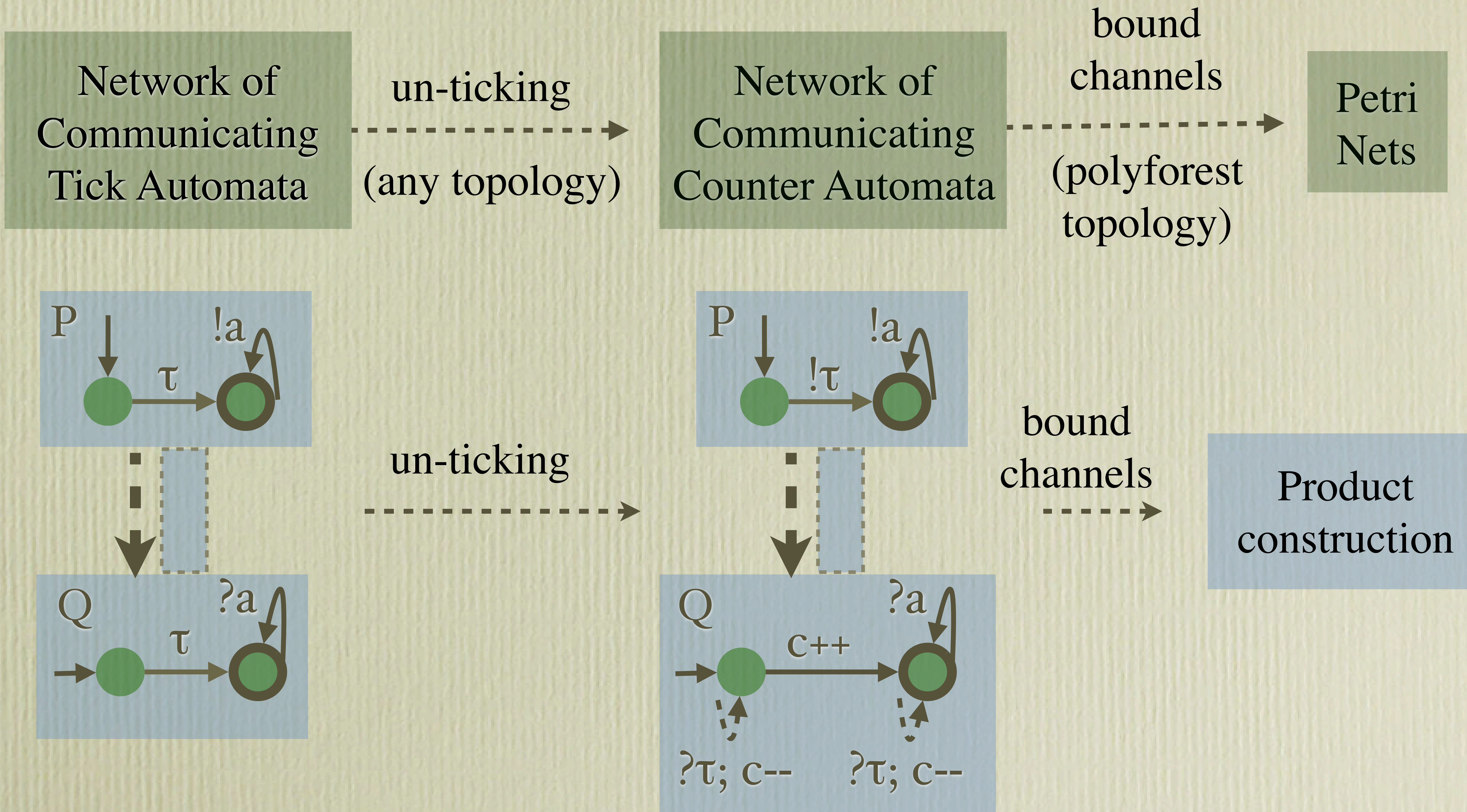
```
P: τ !a
Q: τ     ?a
```

```
P: !τ !a
Q:          c++ ?τ c-- ?a
c=0
```

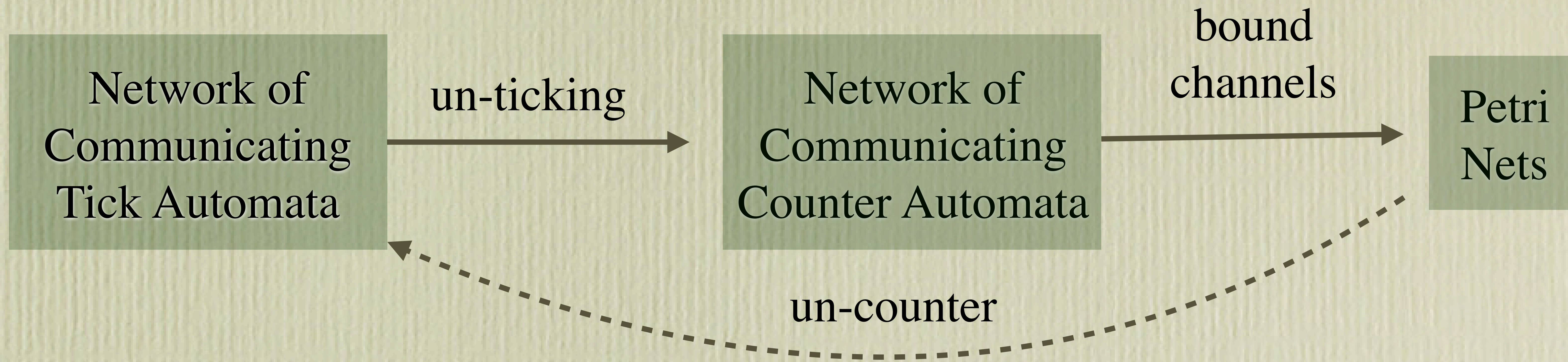


- P *sends* its ticks
- Q increments a counter instead of doing a tick
- In any state, Q can *receive* a tick

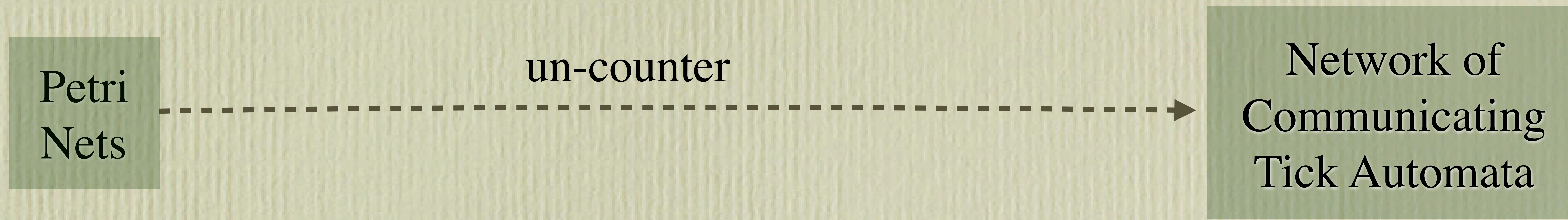
Discrete time

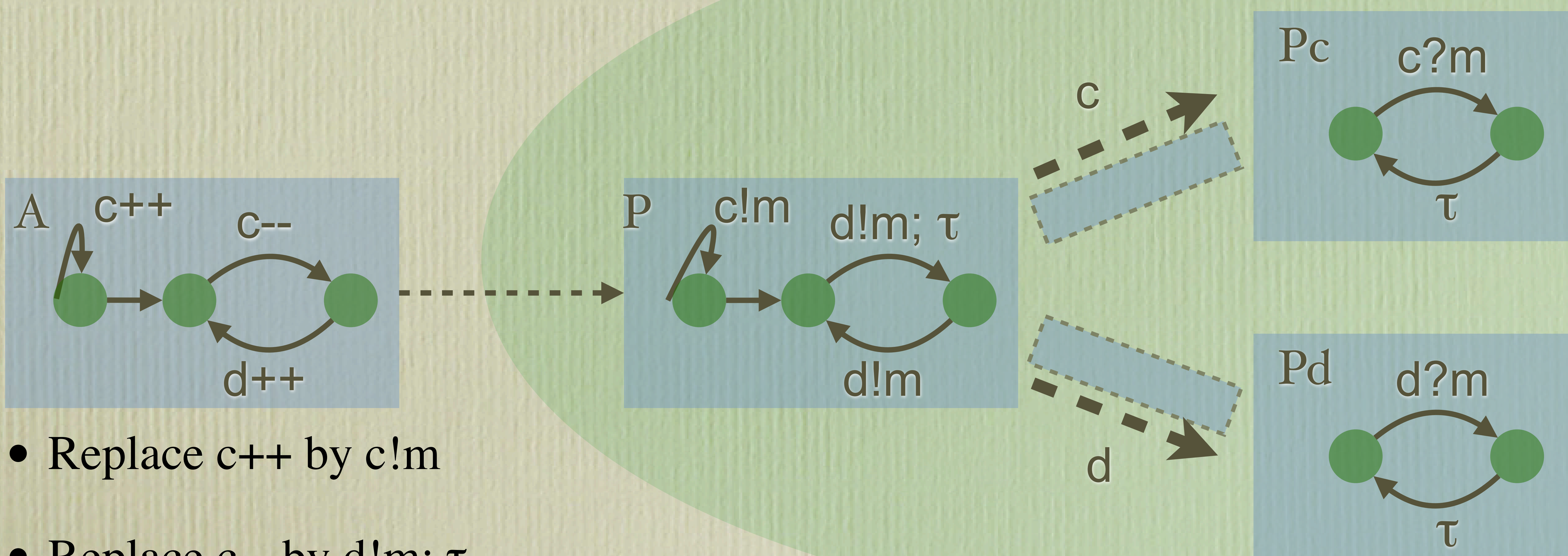


Discrete time



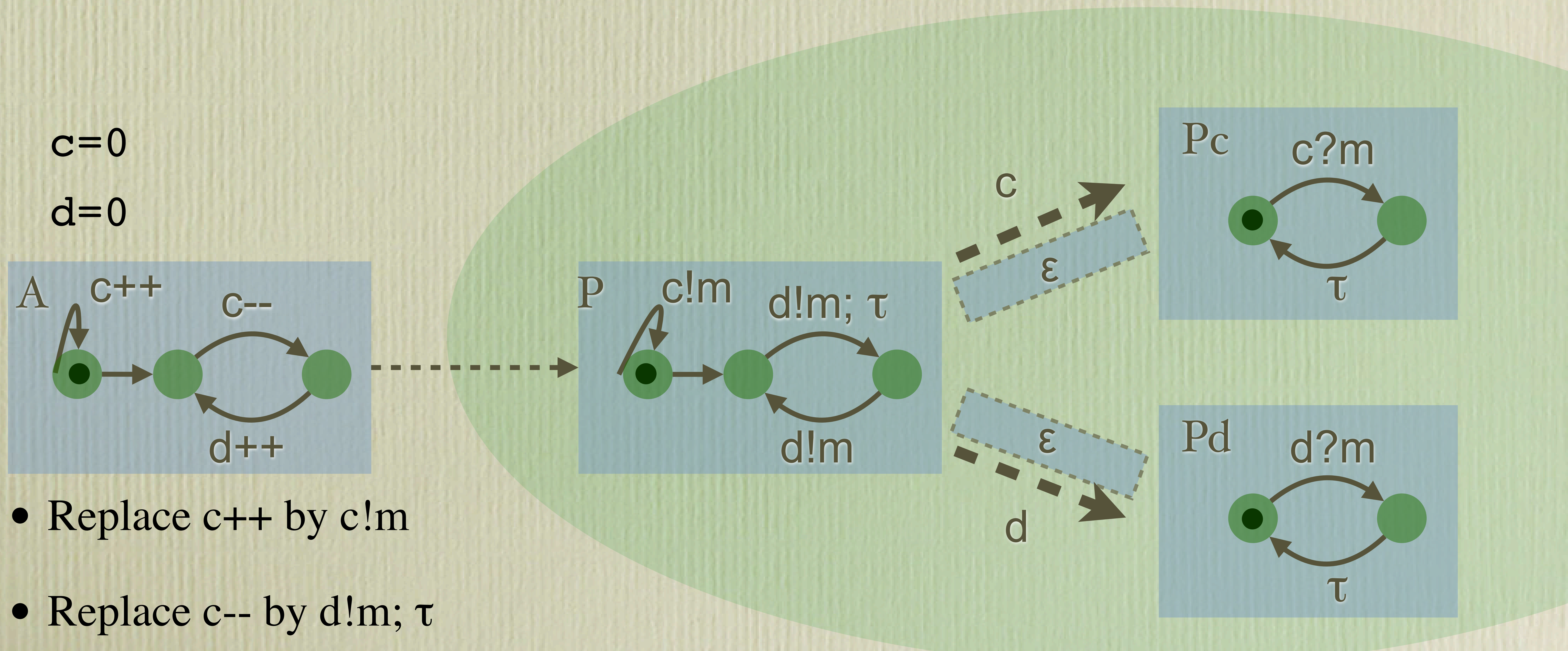
Discrete time





- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

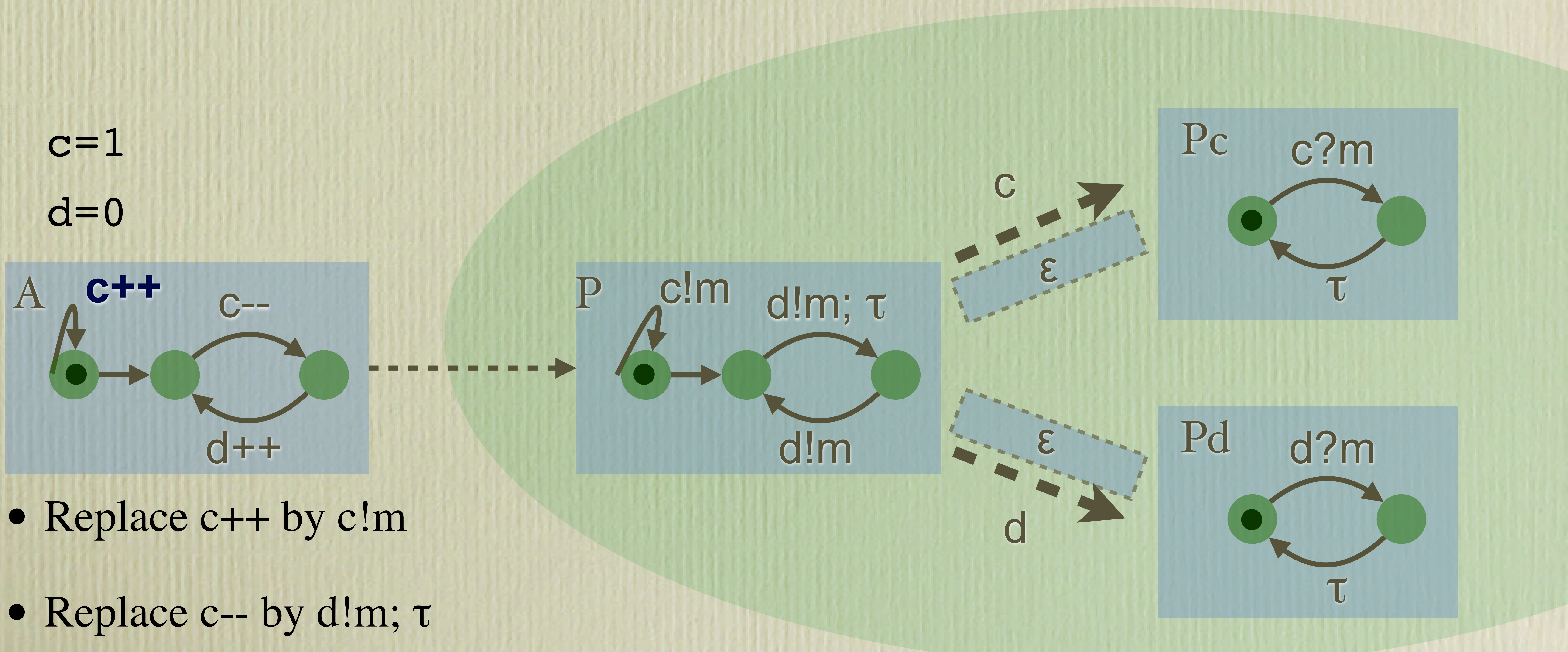


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

d=0

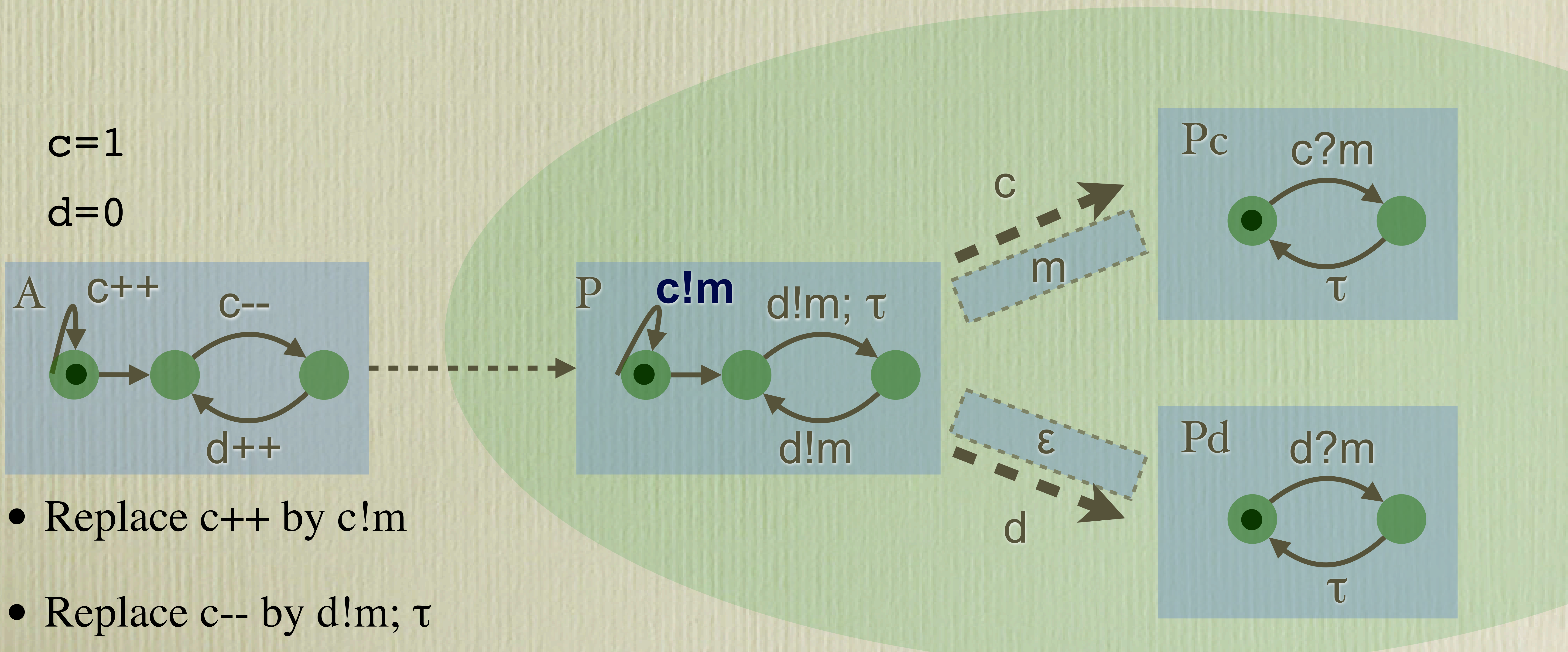


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

d=0

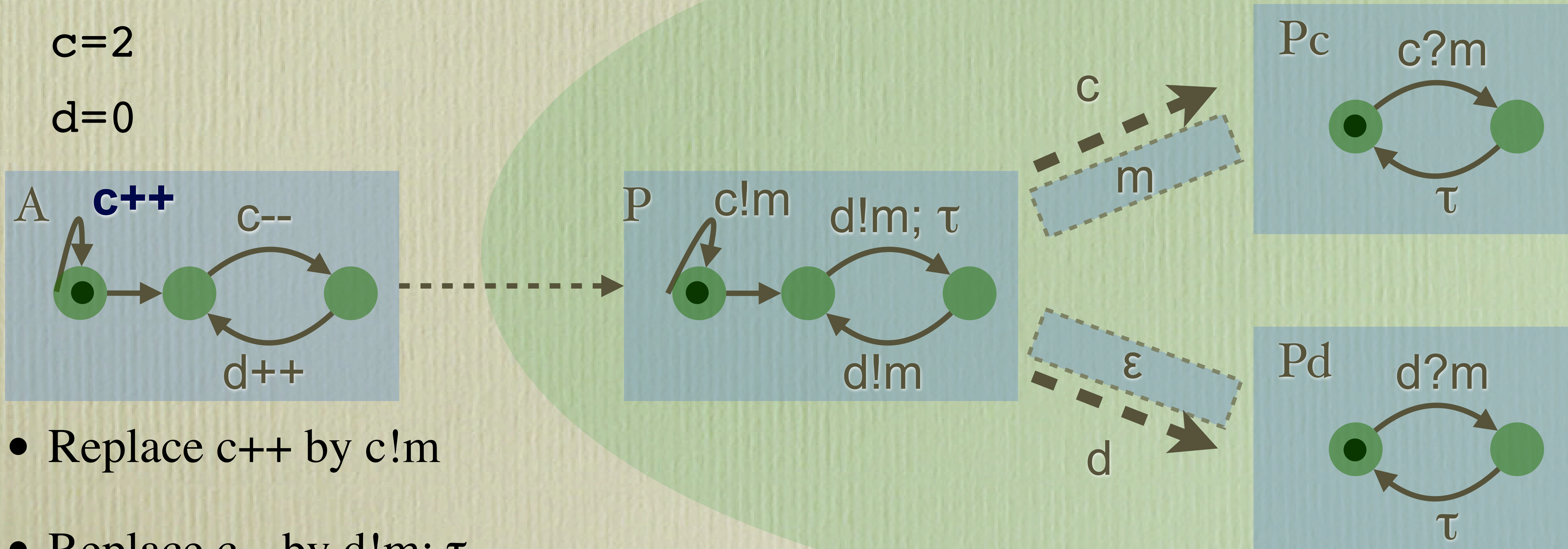


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

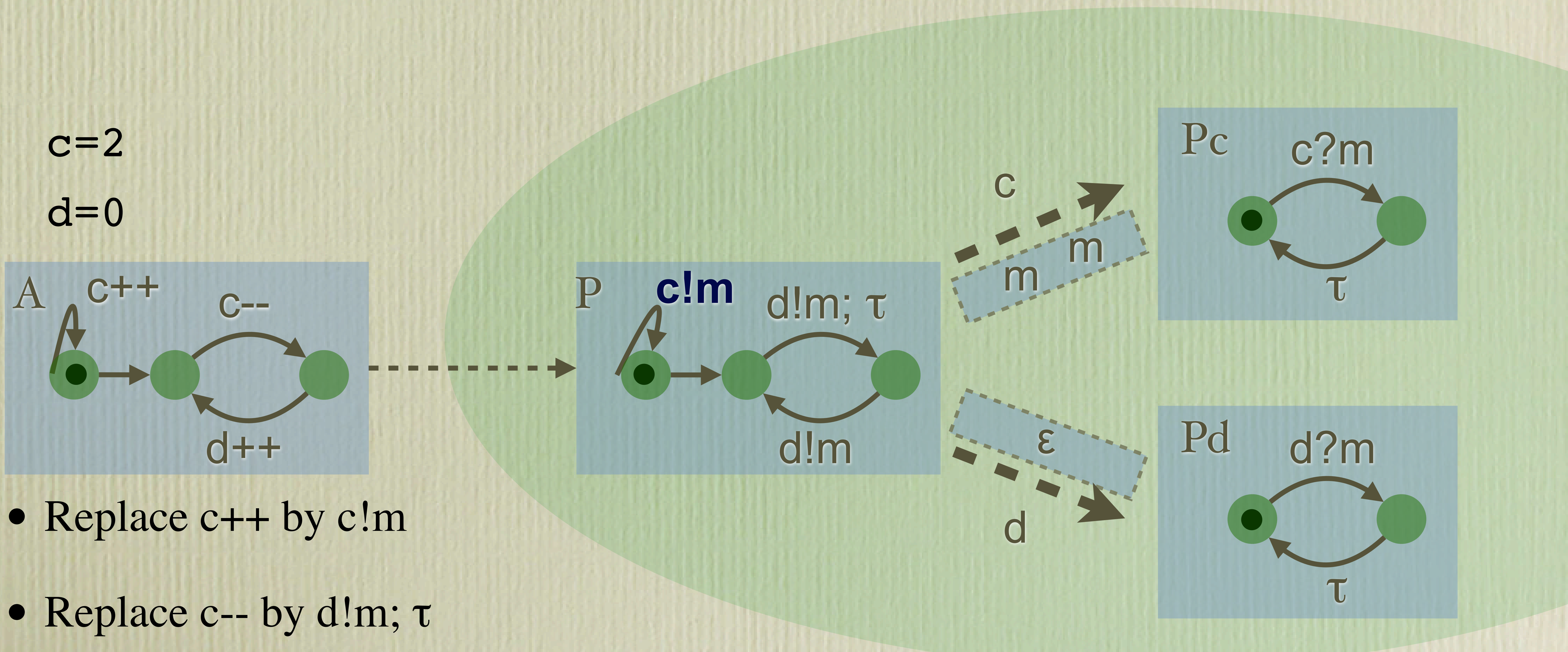
c=2

d=0



- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

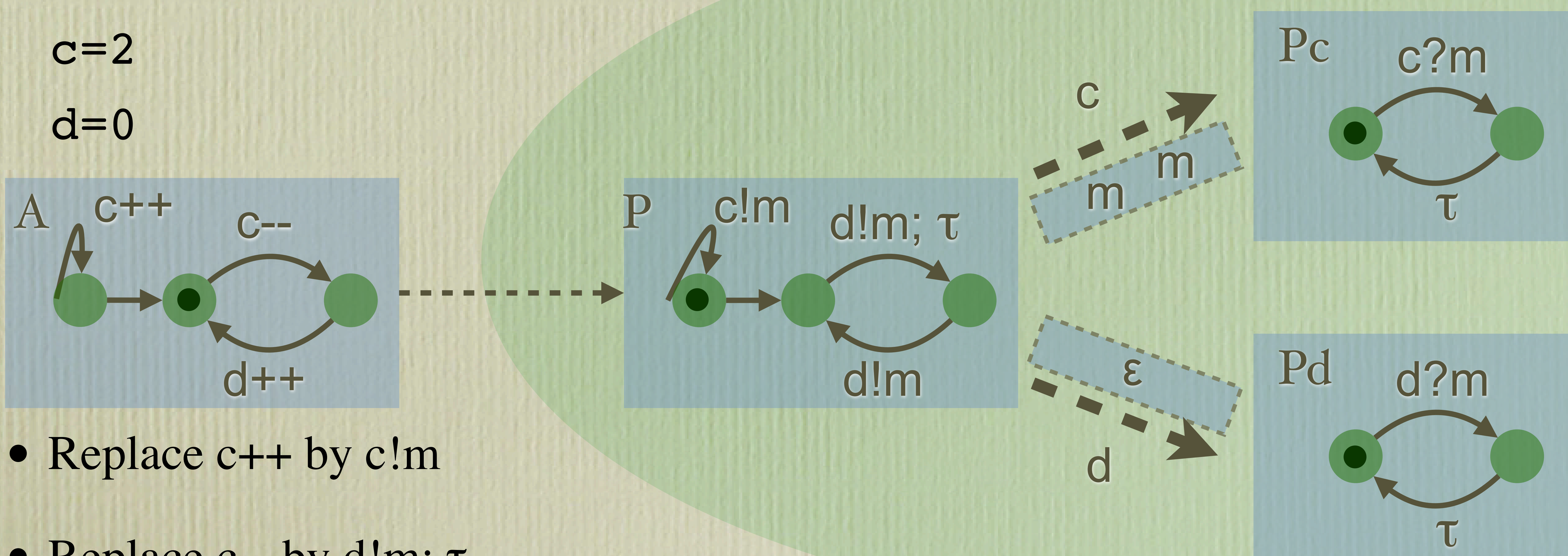


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=2

d=0

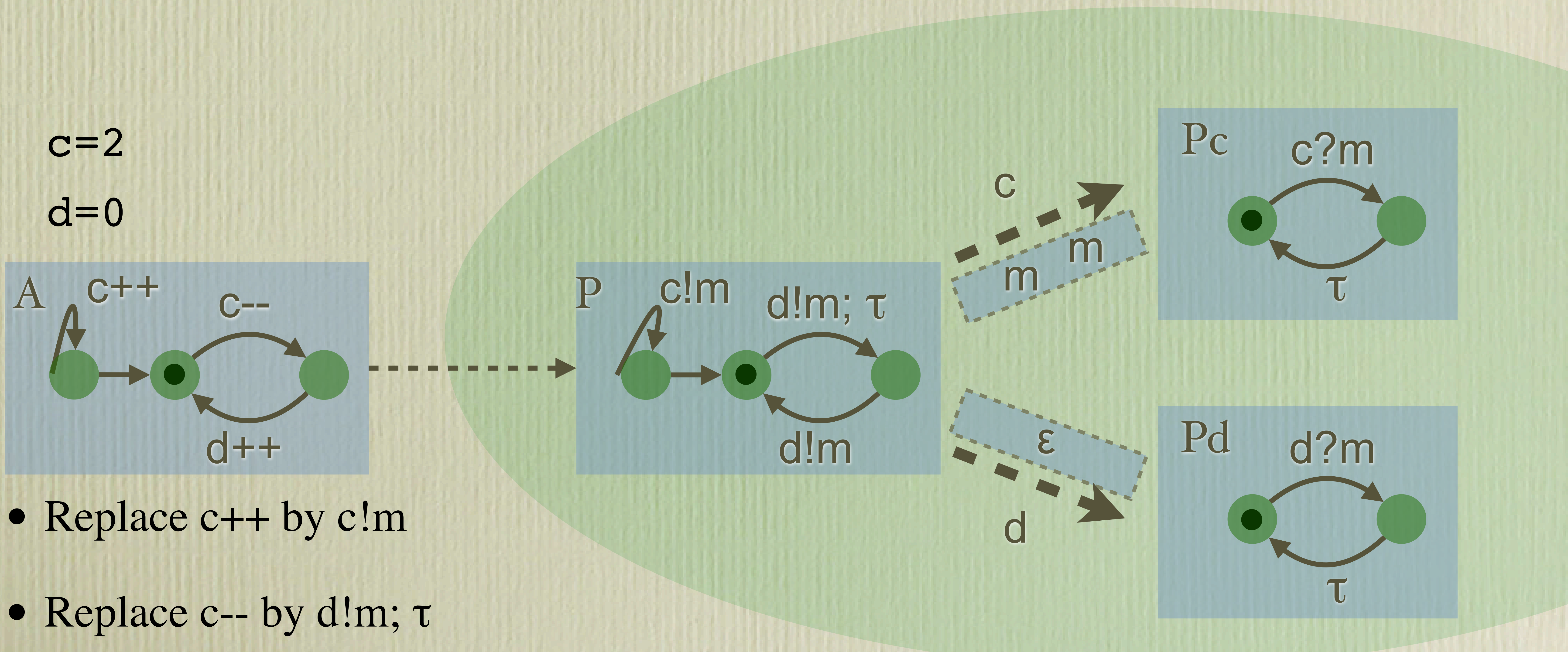


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=2

d=0

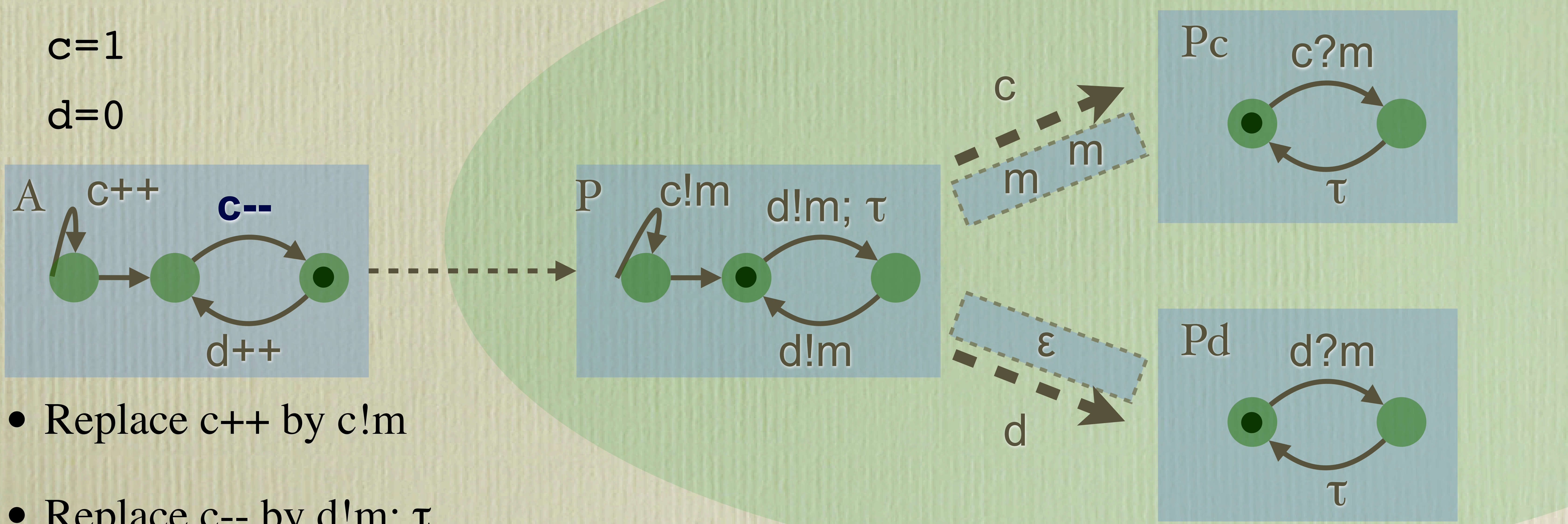


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

d=0

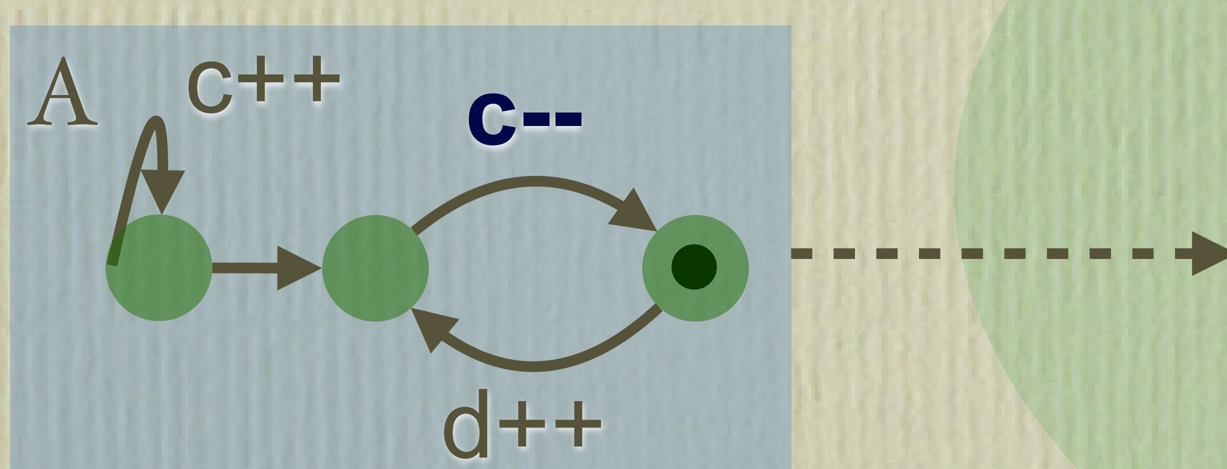


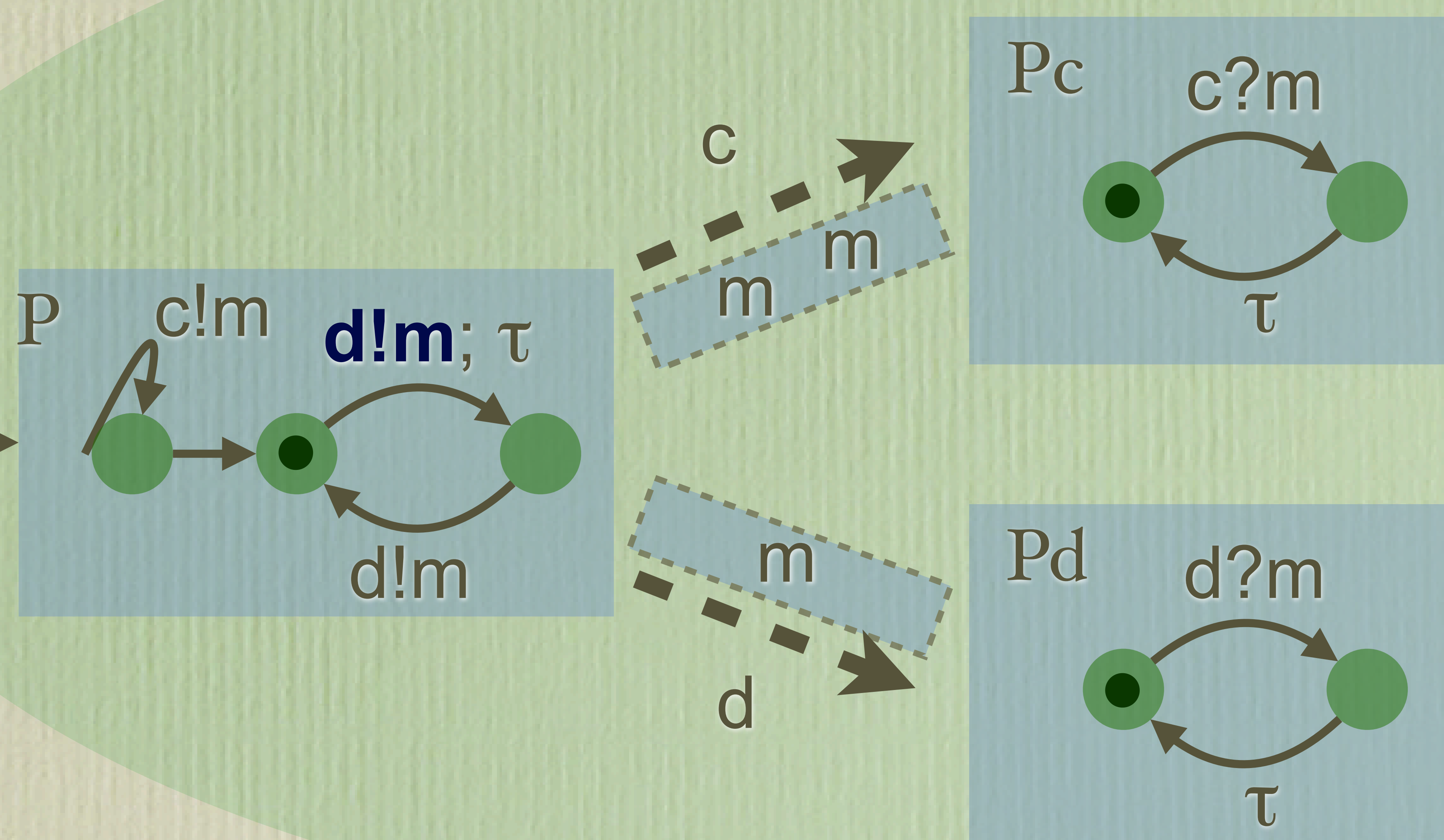
- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

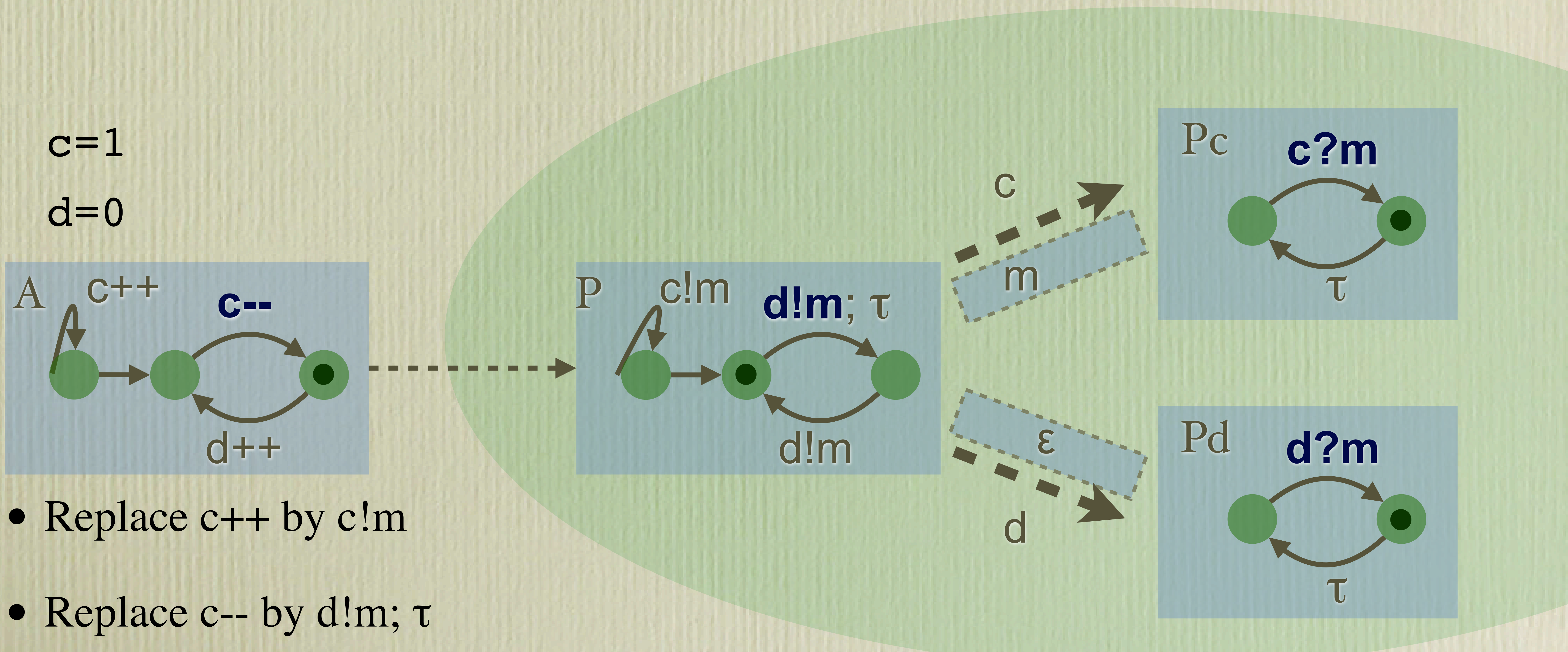
d=0





- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

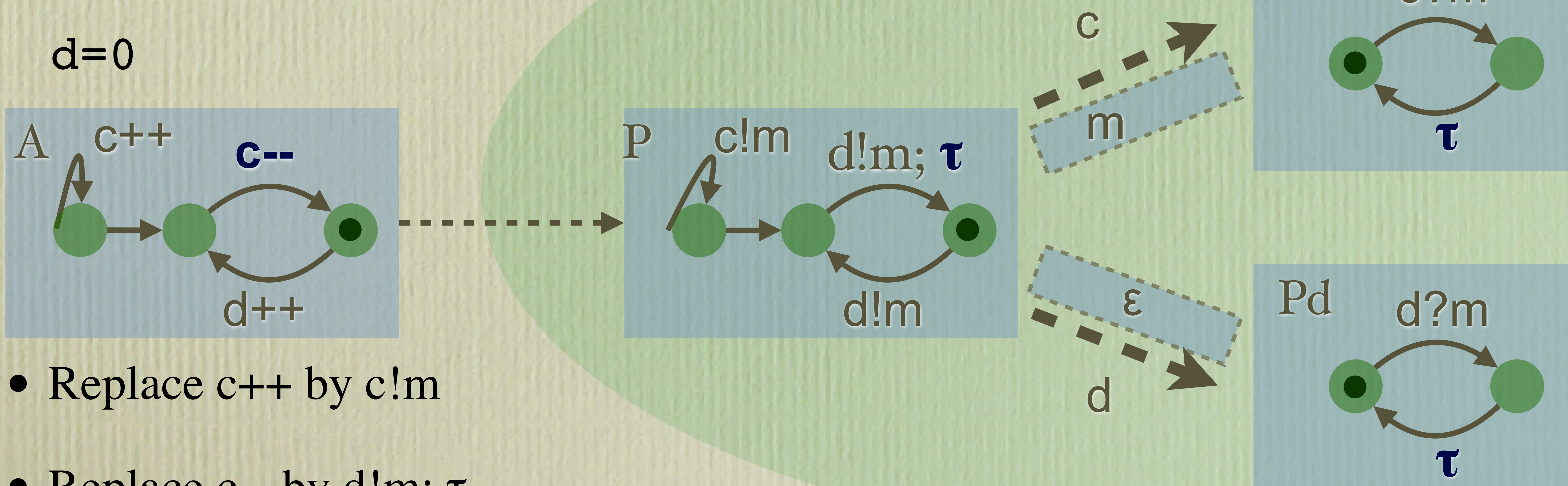


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

d=0

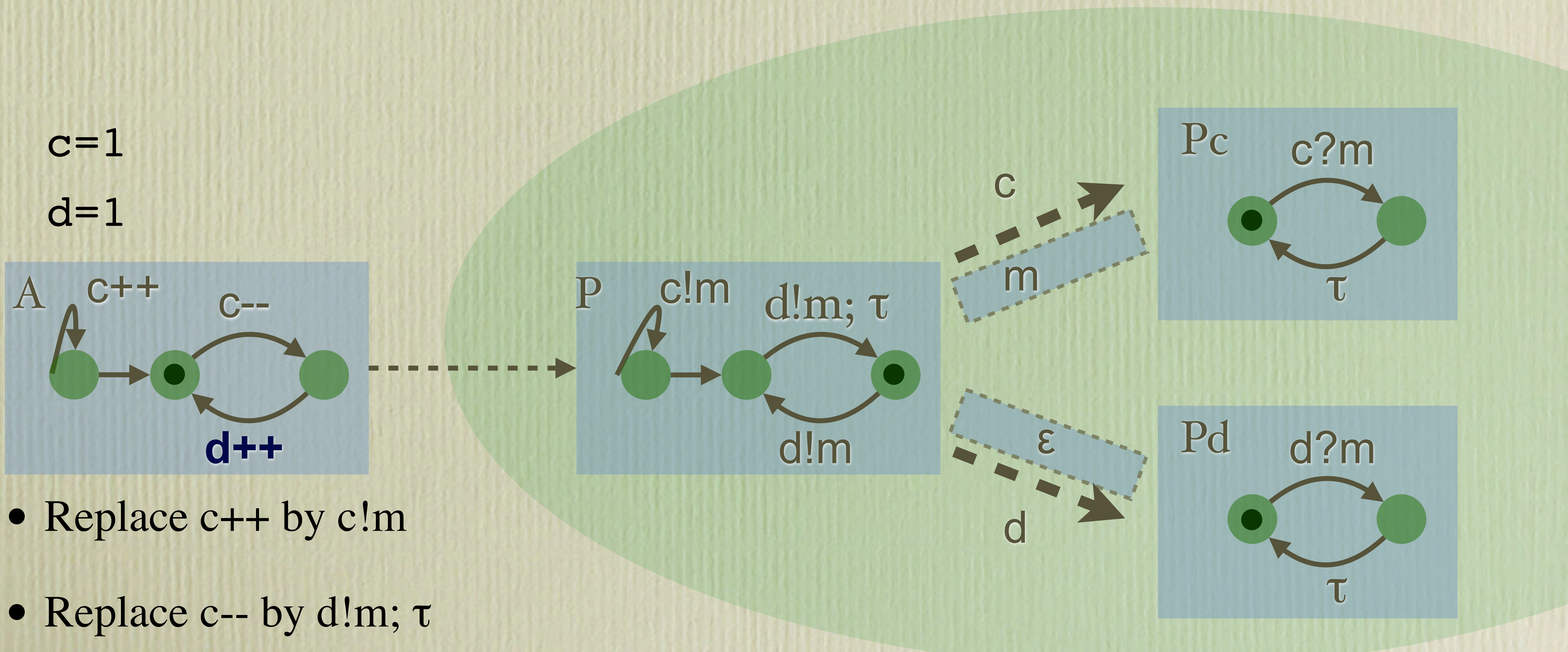


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

d=1

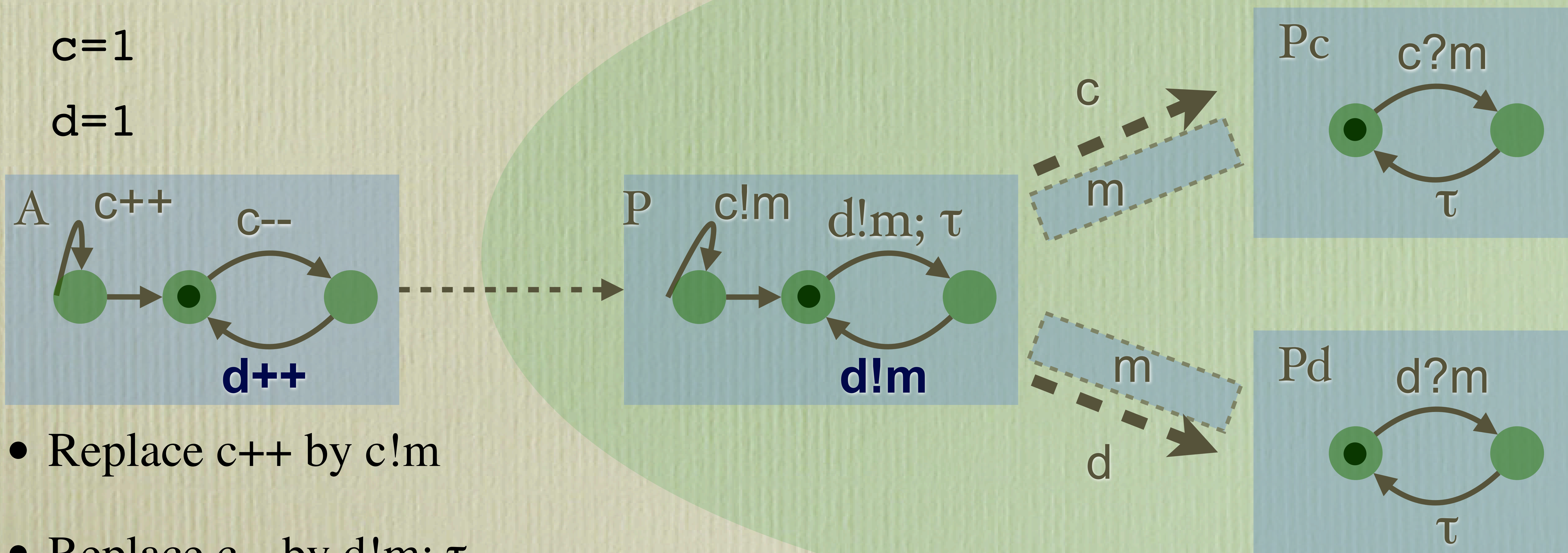


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=1

d=1

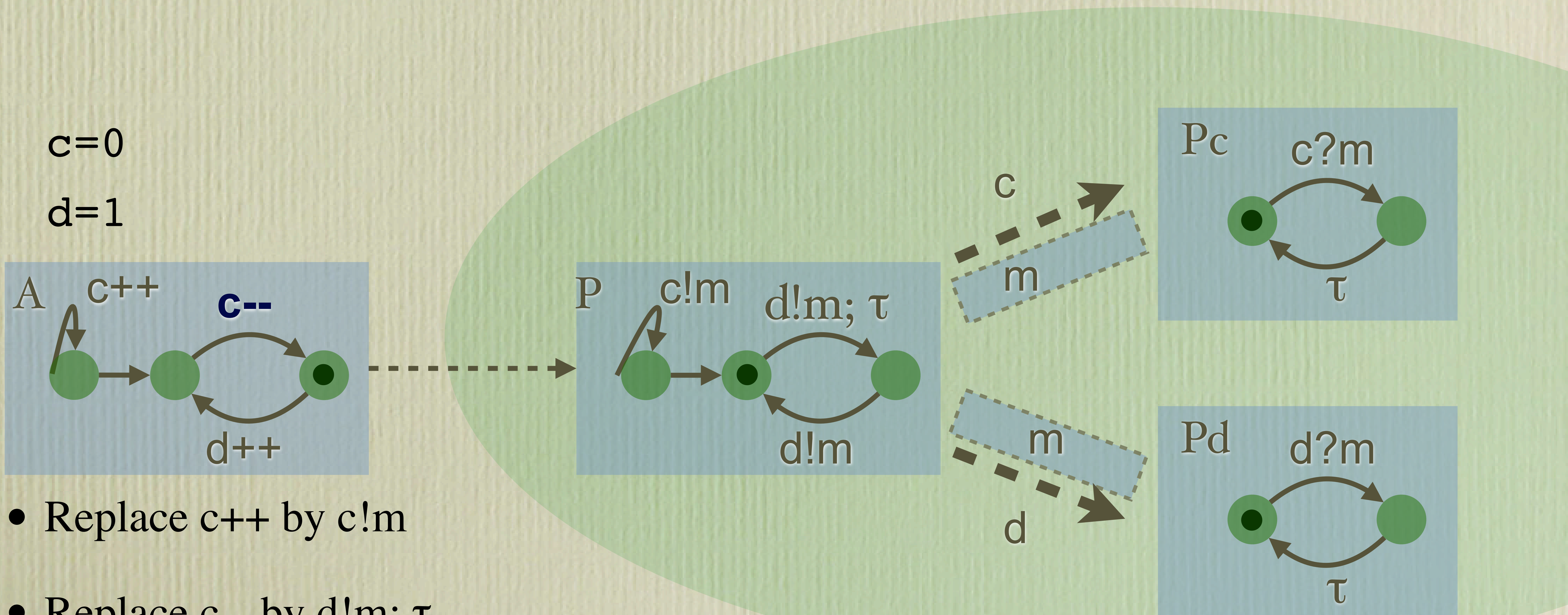


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=0

d=1

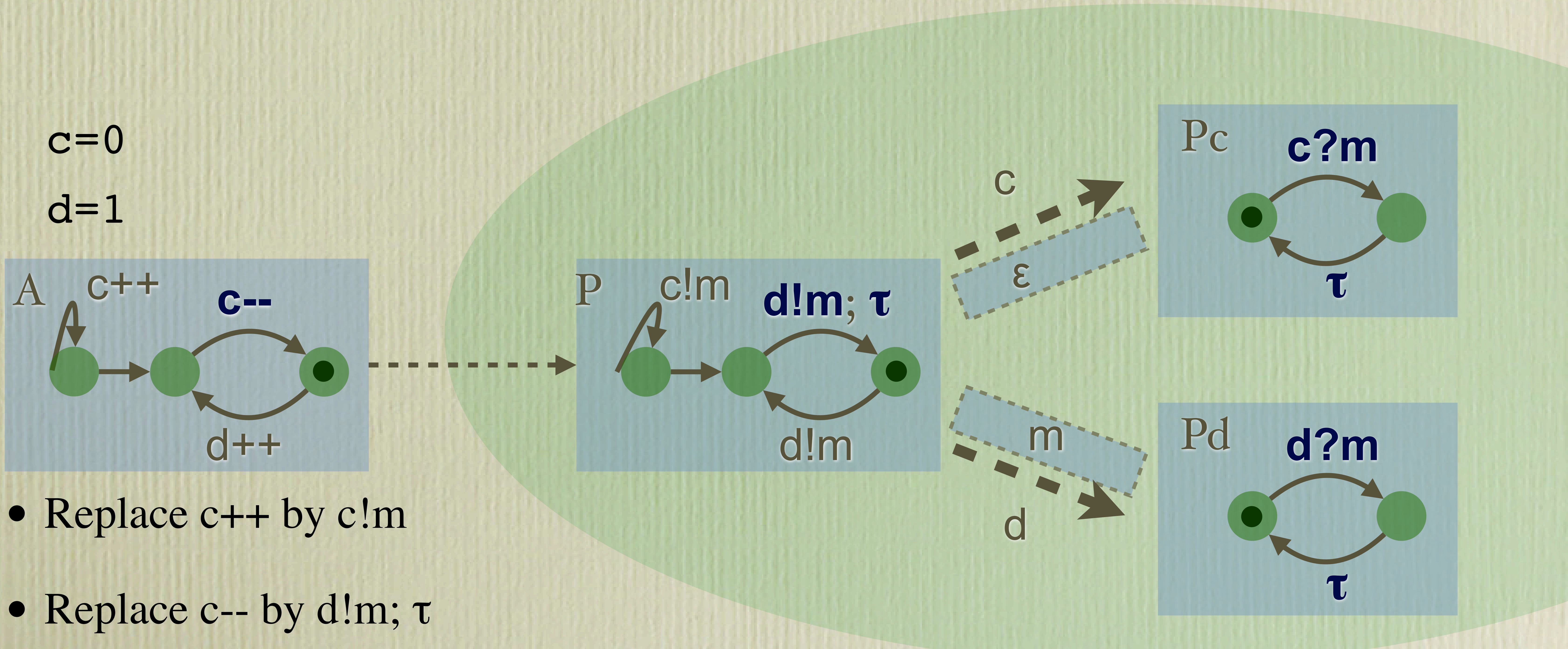


- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=0

d=1

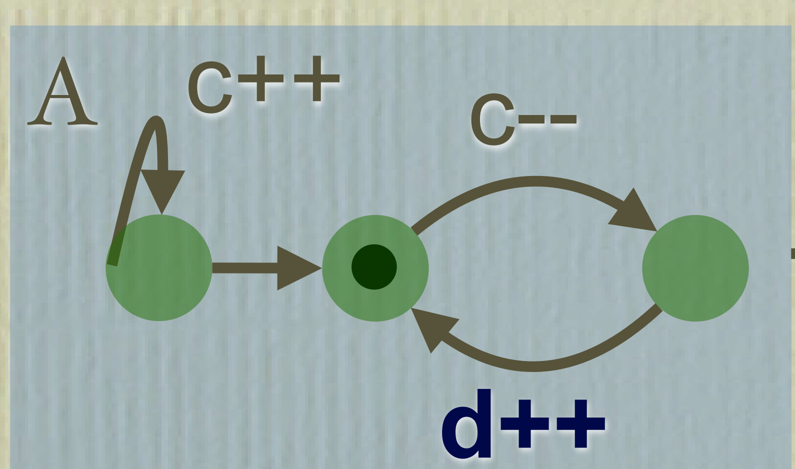


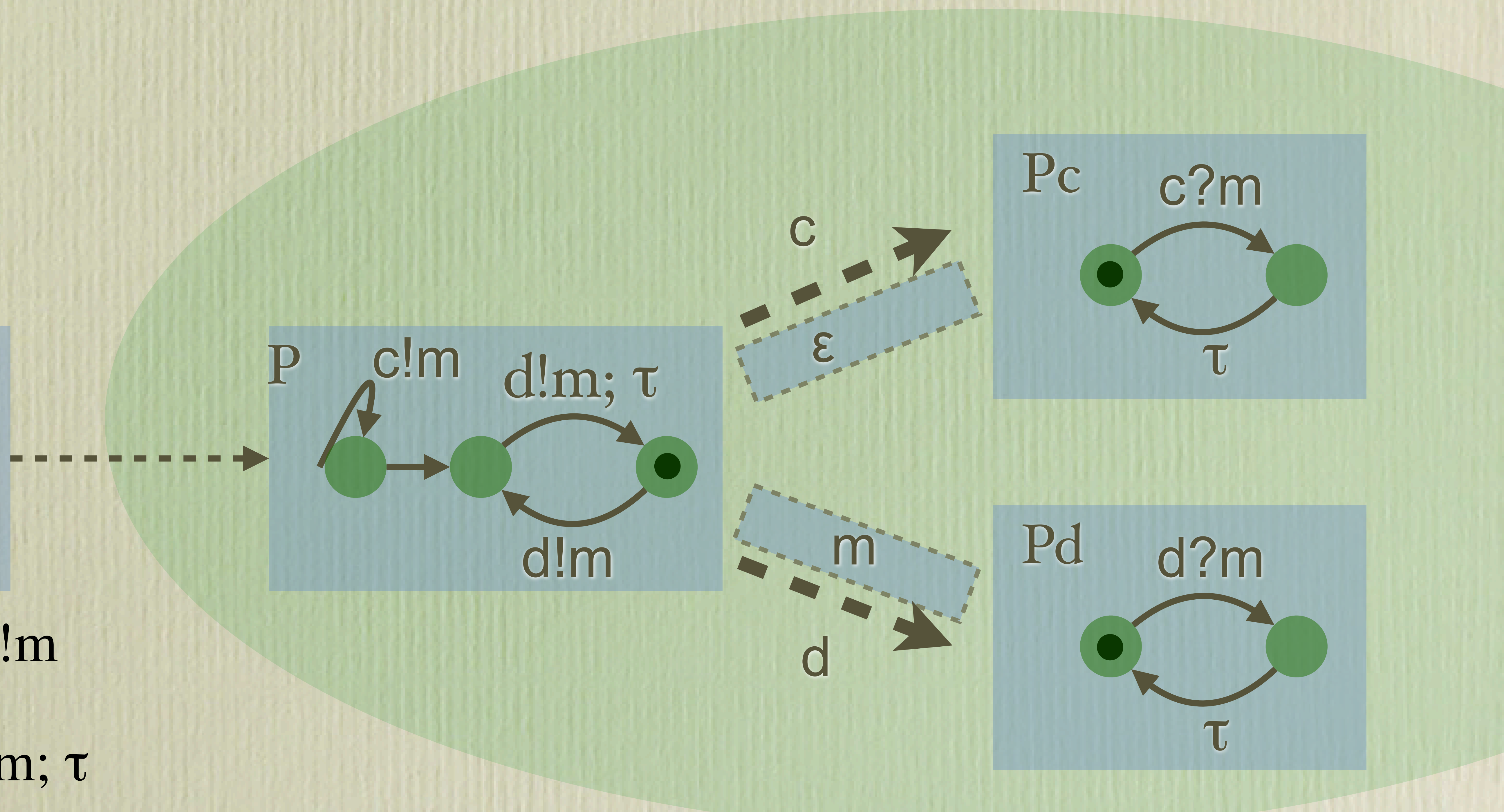
- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

c=0

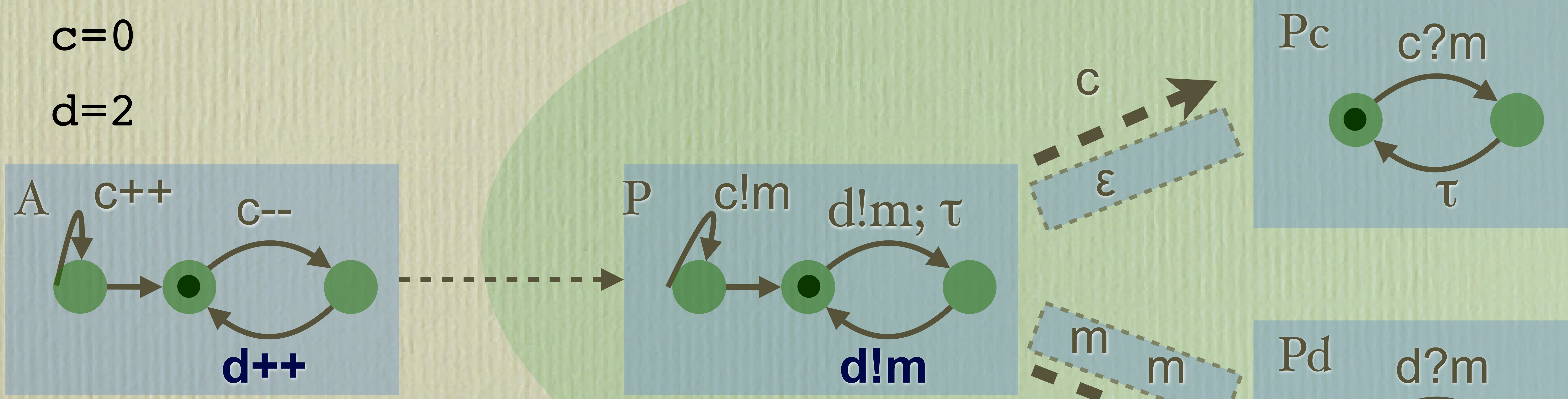
d=2





- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time



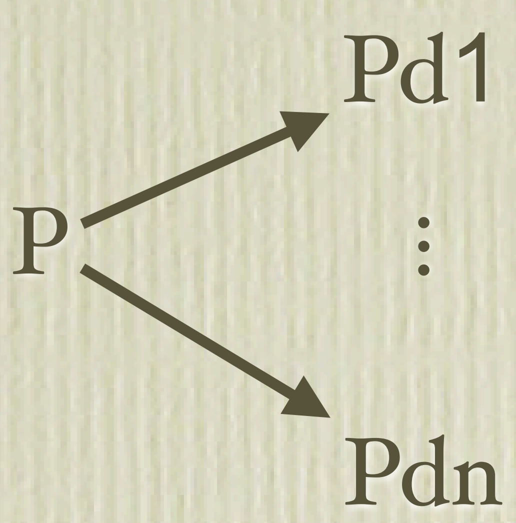
- Replace c++ by c!m
- Replace c-- by d!m; τ

Discrete time

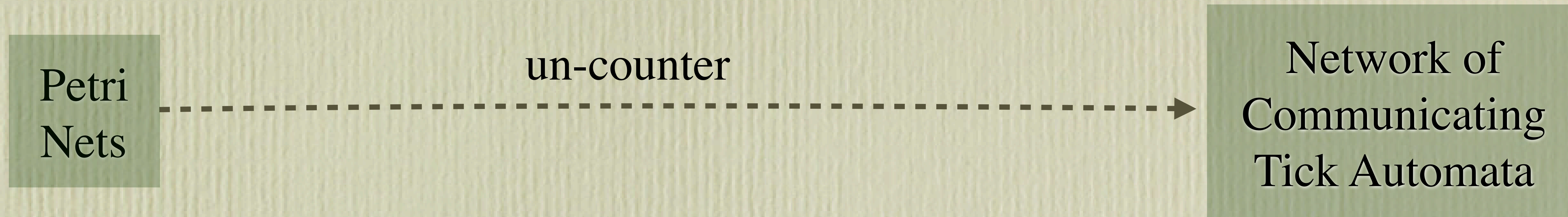
Petri Nets $\xrightarrow{\text{un-counter}}$ Network of Communicating Tick Automata

The construction can be adapted to work for any given *star* topology

given counters
d_1...d_n

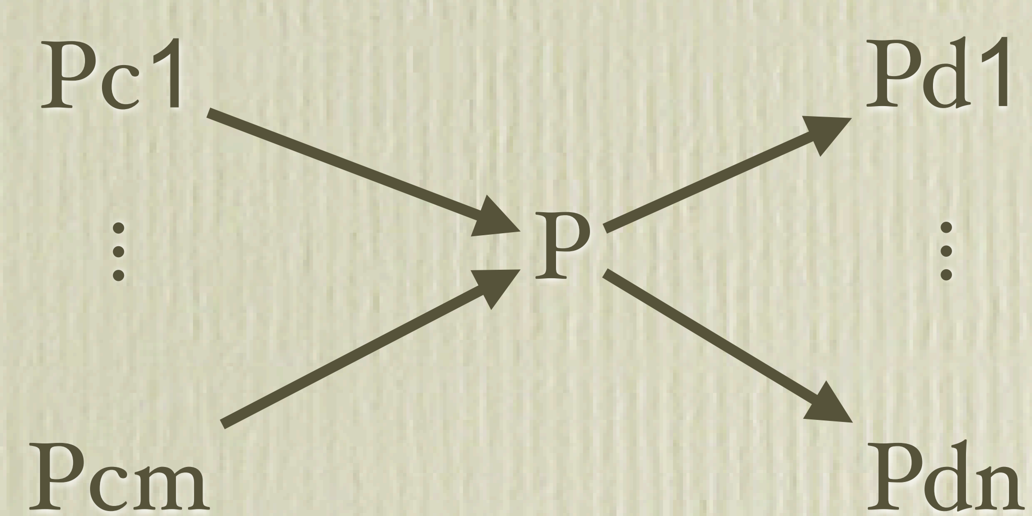


Discrete time

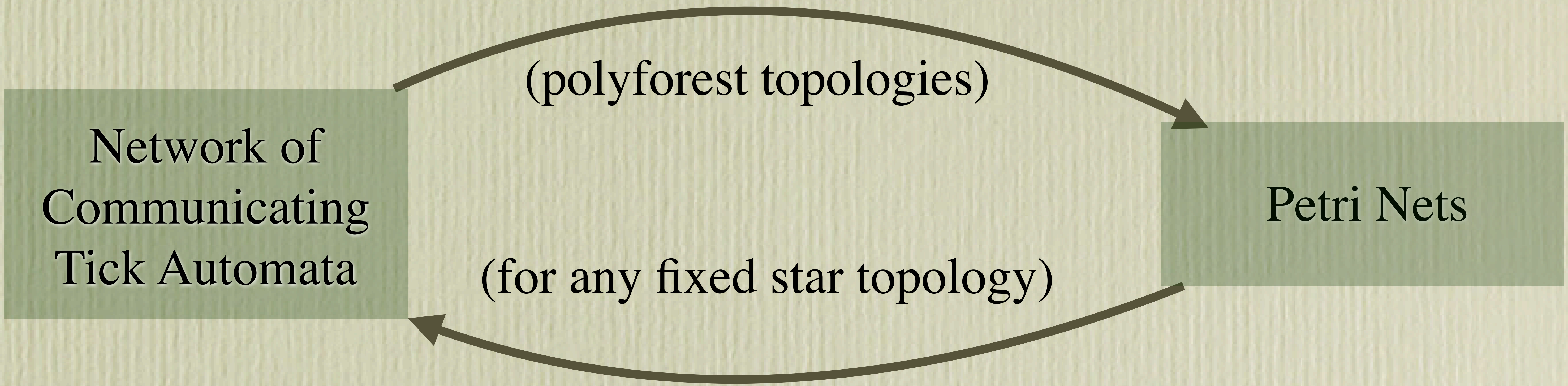


The construction can be adapted to work for any given *star* topology

given counters c_1...c_m and d_1...d_n



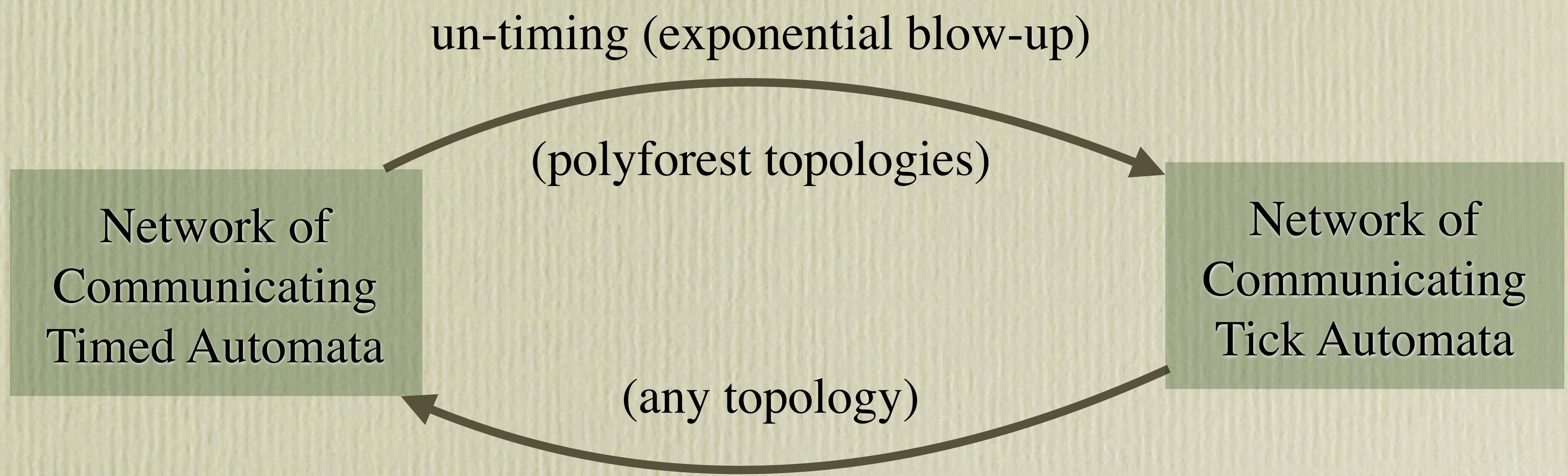
Discrete time (summary)



Theorem: Reachability decidable iff polyforest topology

Theorem: The complexity is the same as Petri nets

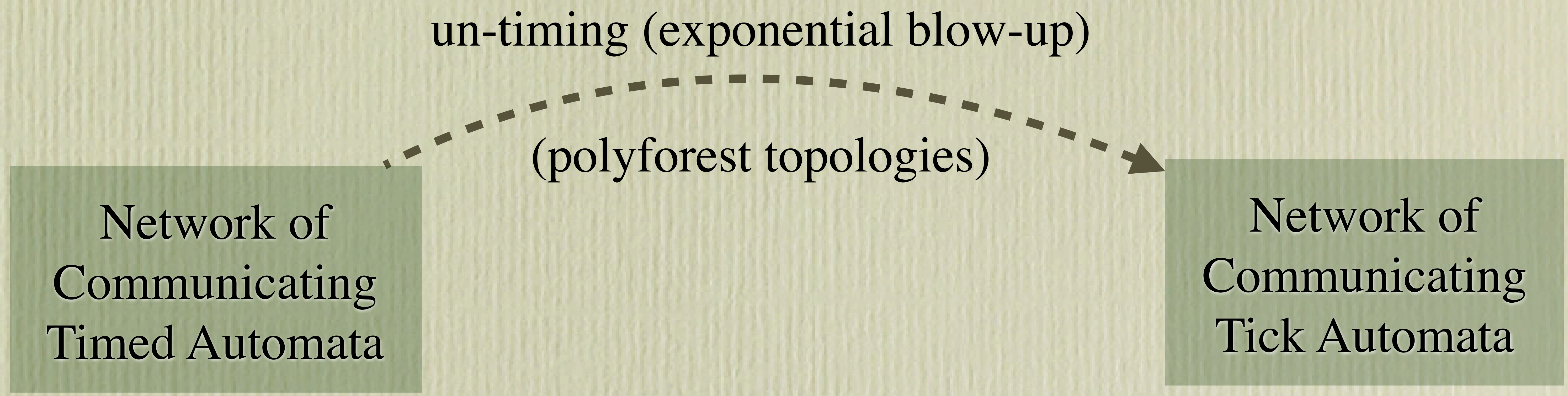
Dense time



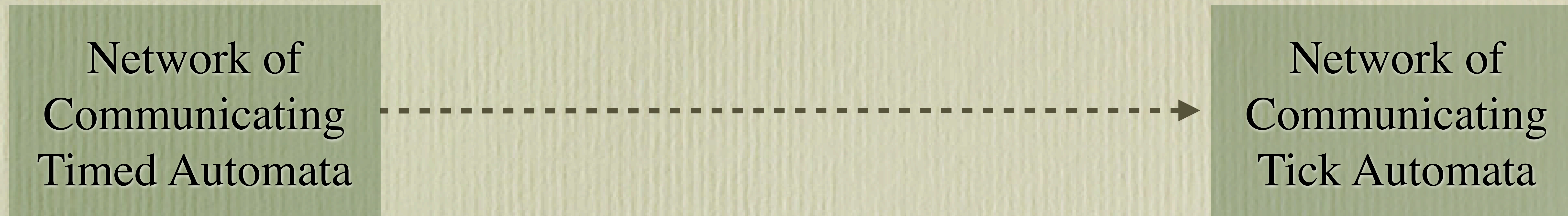
Theorem: Reachability decidable iff polyforest topology

Theorem: The complexity is Petri net-hard and in Exp-Petri net

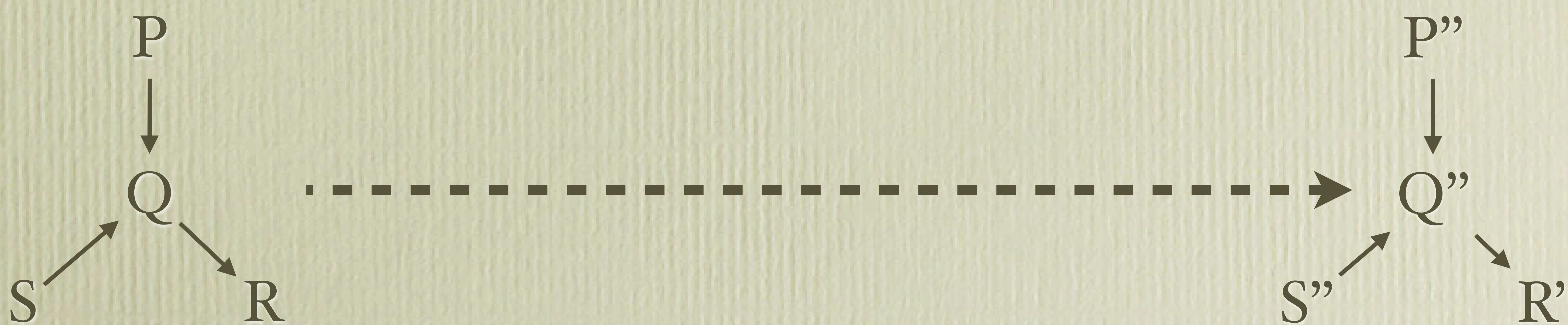
Dense time


un-timing (exponential blow-up)
(polyforest topologies)
Network of Communicating Timed Automata
Network of Communicating Tick Automata

Dense time



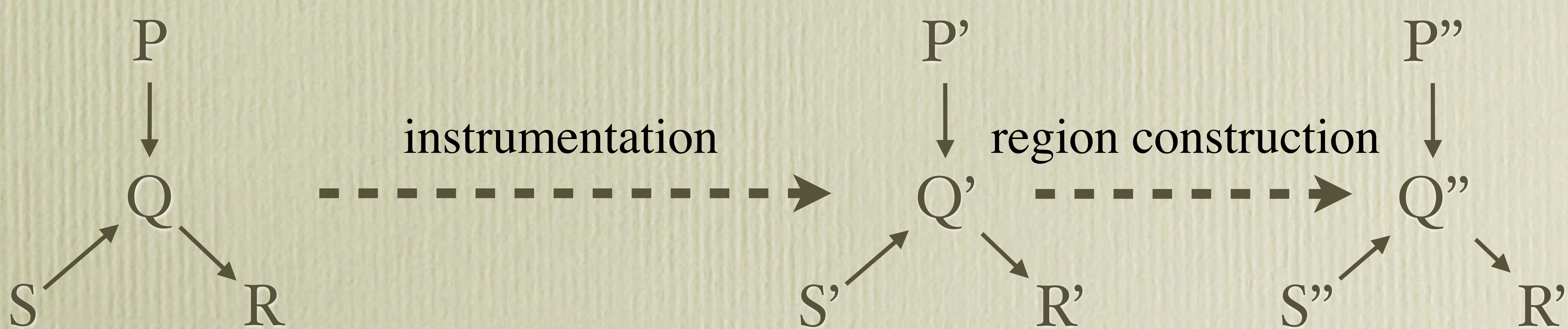
In order to preserve the topology, we would like to apply the region construction *locally* to each automaton. But this is incorrect.



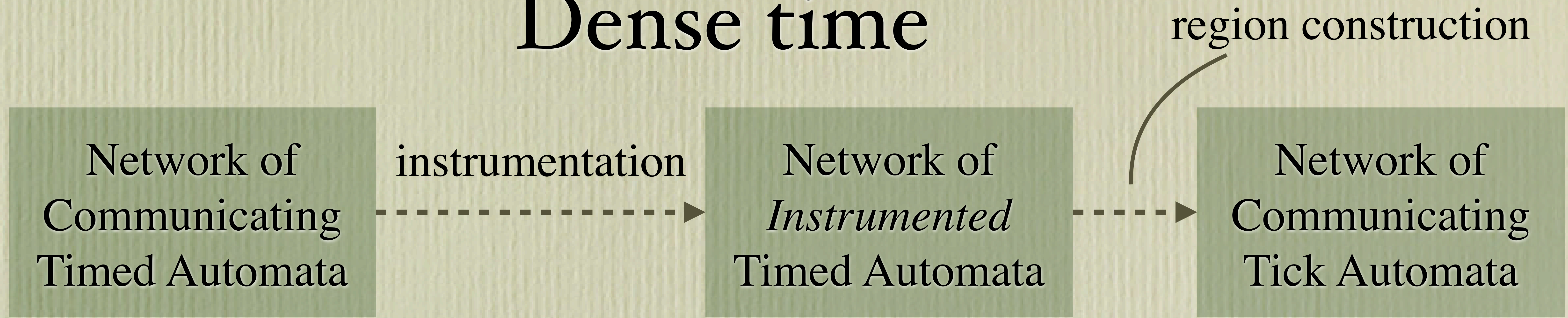
Dense time

Network of Communicating Timed Automata → instrumentation → Network of *Instrumented* Timed Automata → region construction → Network of Communicating Tick Automata

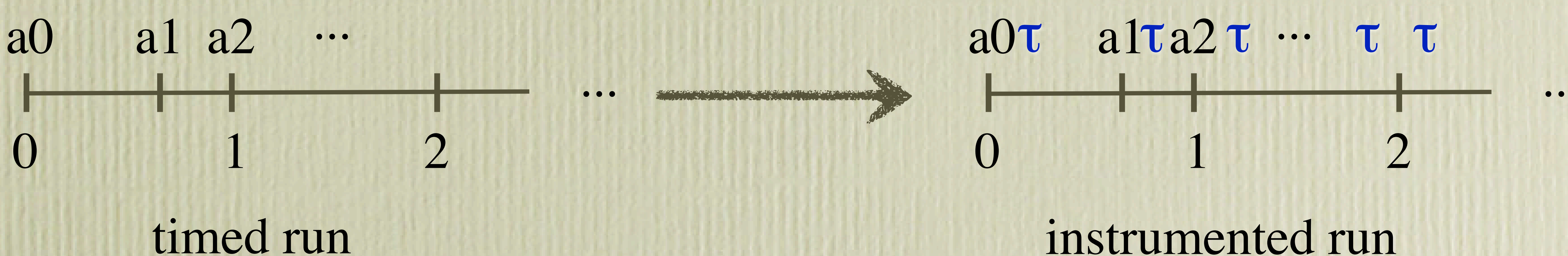
Instrumented automaton: just a
timed automaton with tick actions



Dense time

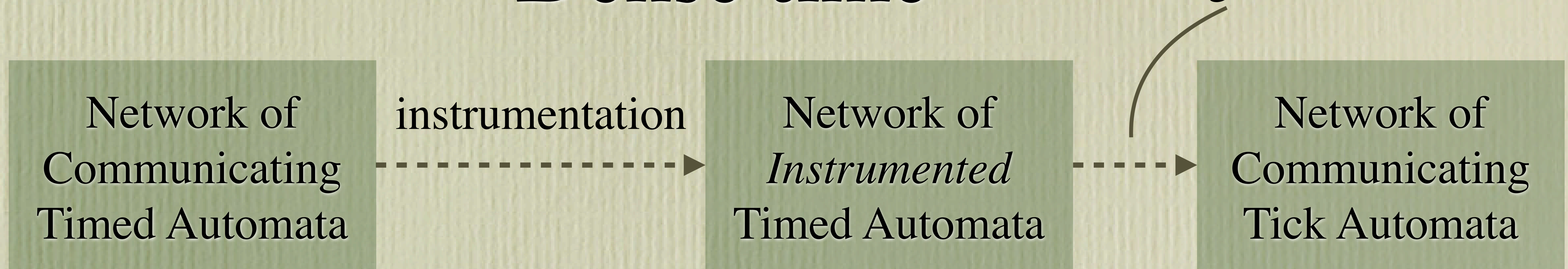


Intuition for instrumentation:
add *ticks* to mark the beginning and end of integer time points

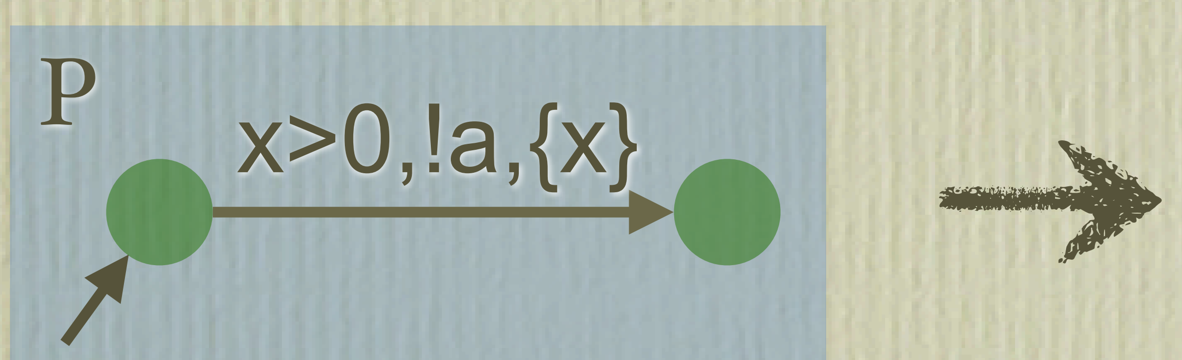


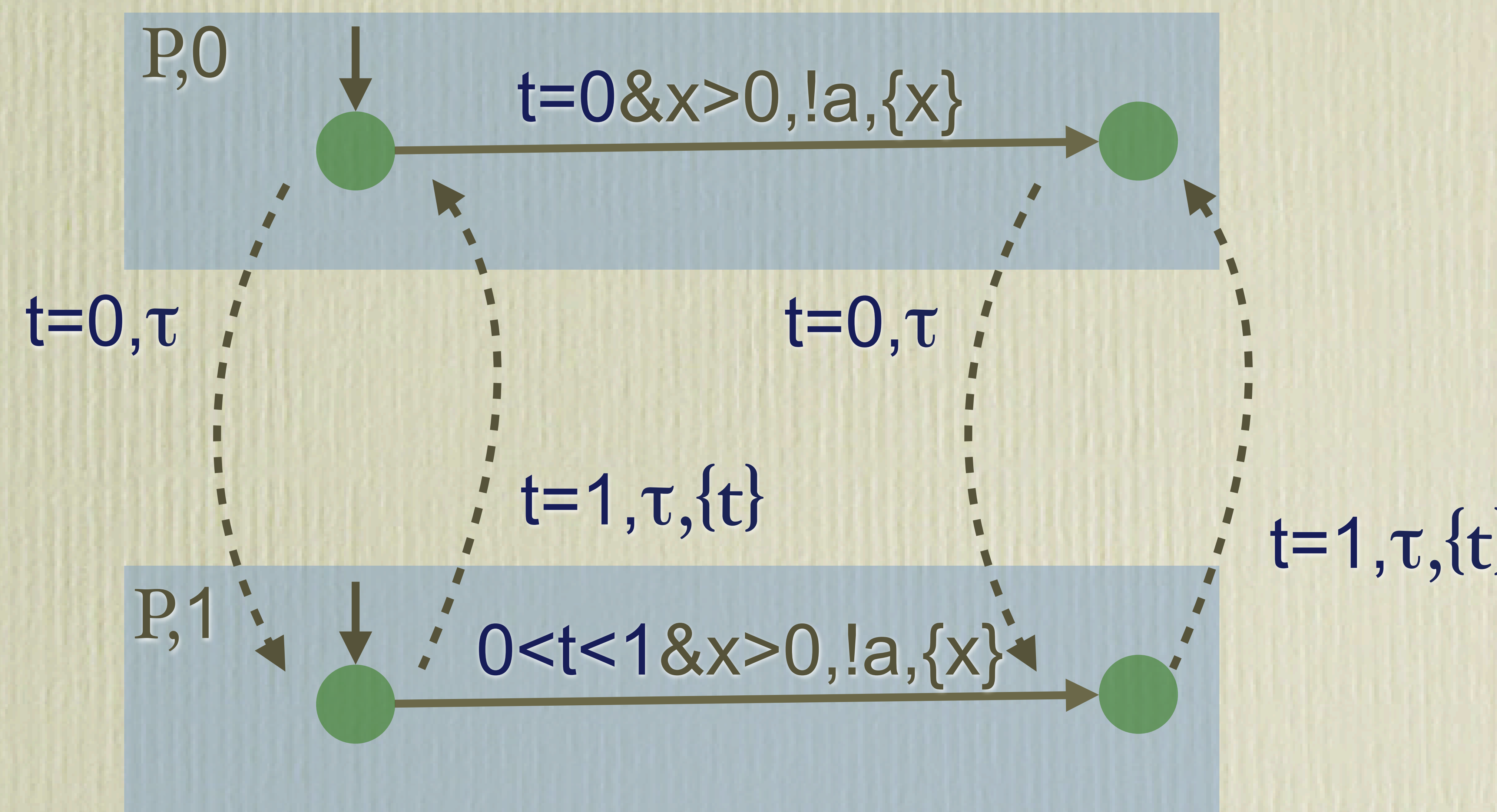
This can be achieved by a simple construction on each timed automaton

Dense time

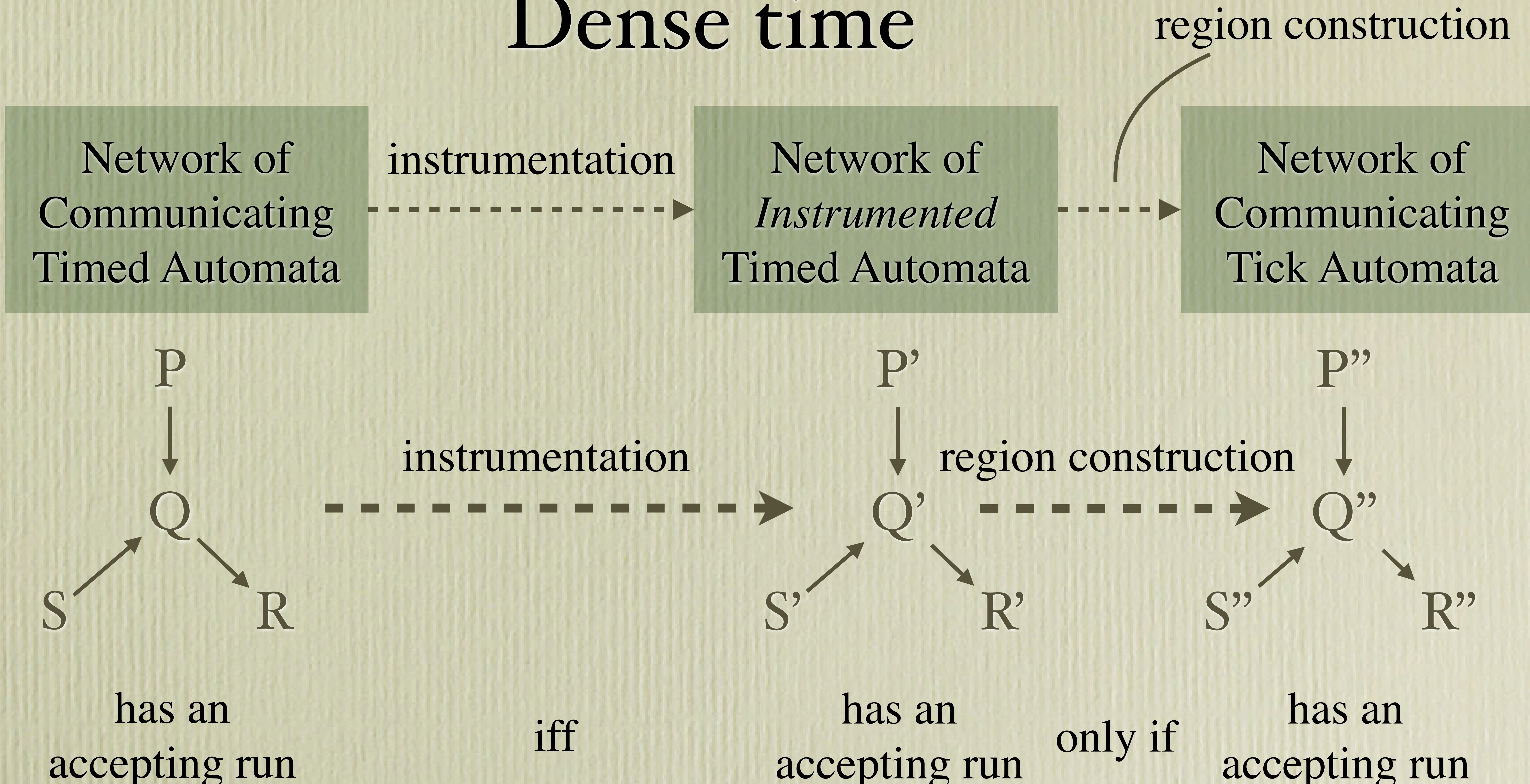


Realising instrumentation:
add a new clock t





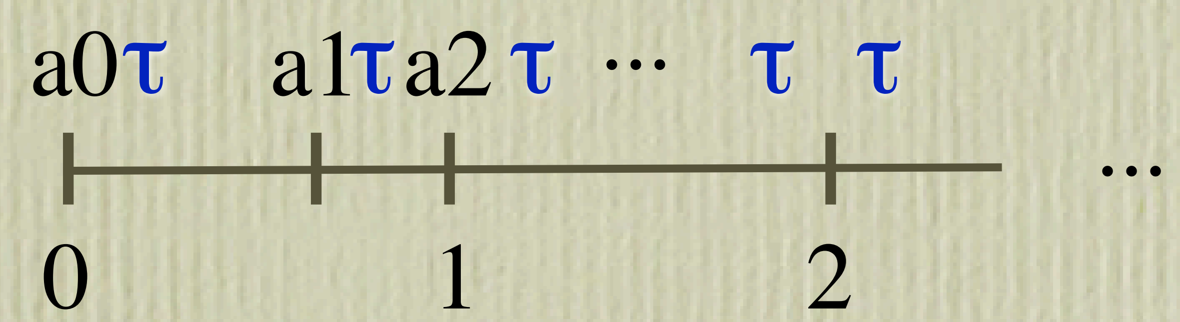
Dense time



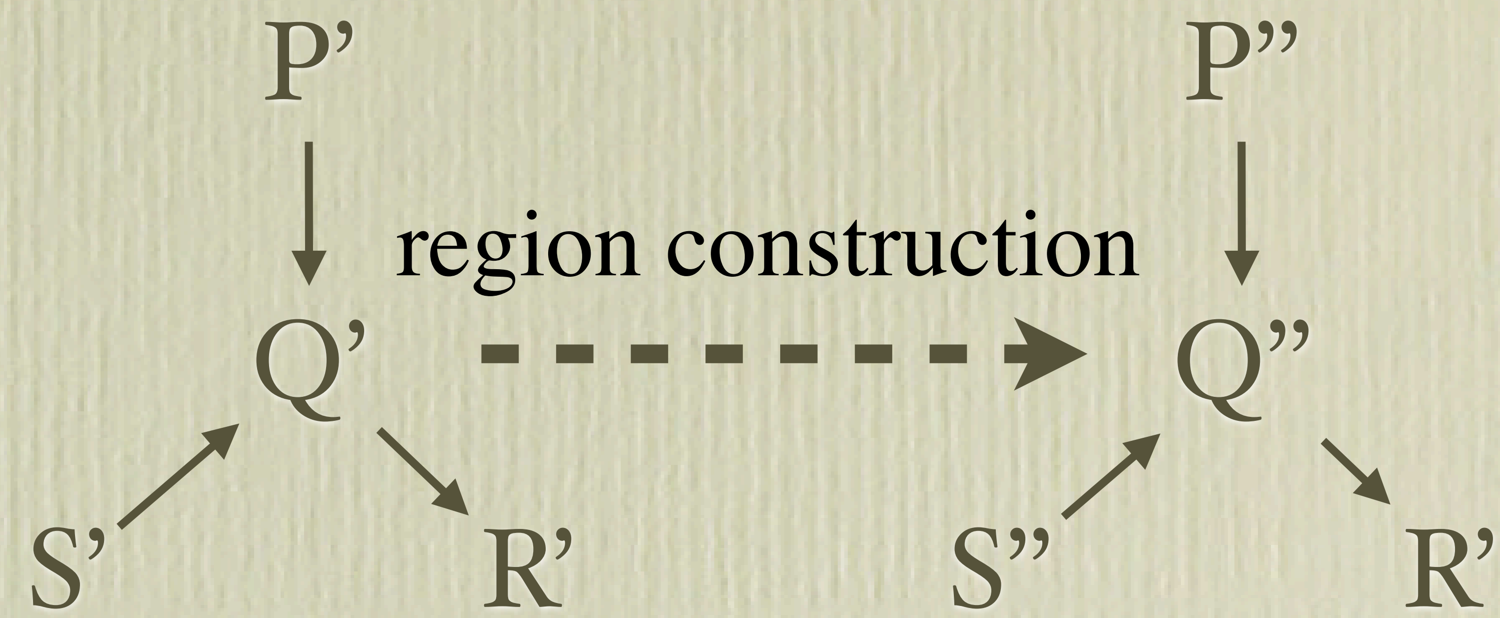
The other direction follows from a Rescheduling Lemma for timed automata

Dense time

- An accepting run on the left induces a local accepting run in each P”, Q”, etc…
- Instrumentation guarantees that integral timestamps agree.



- What about non-integral ones? Instrumentation just ensures that they are in the same interval (k, k+1)

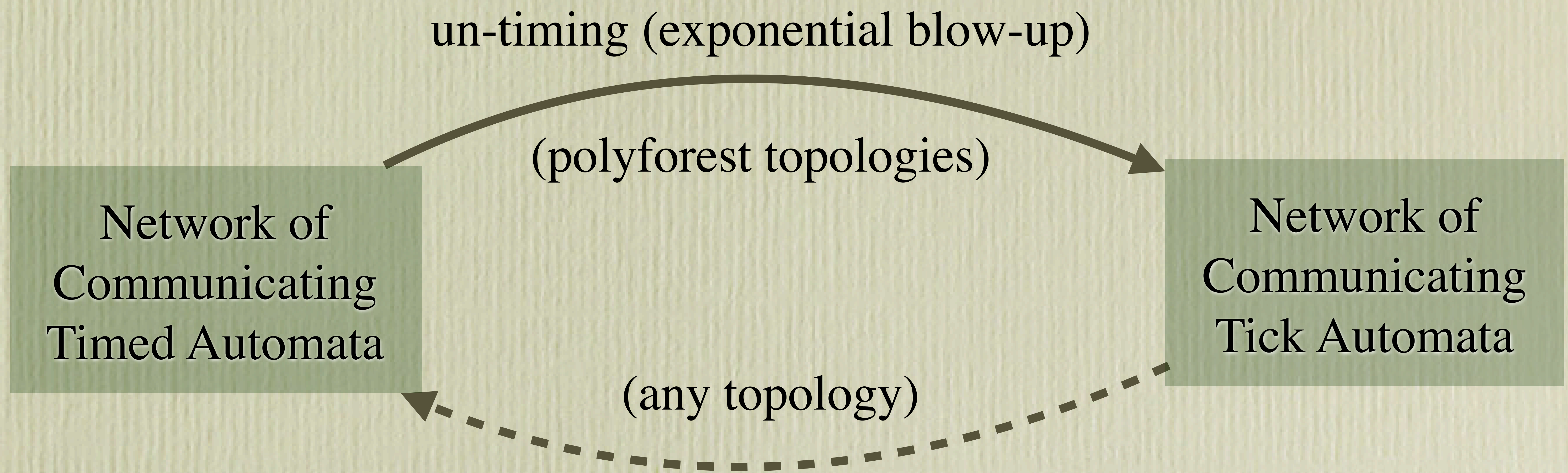


has an accepting run if has an accepting run

Need to preserve send/receive causality ordering!

A Rescheduling Lemma ensures that *on polyforest topologies* timestamps can be chosen to preserve causality

Dense time

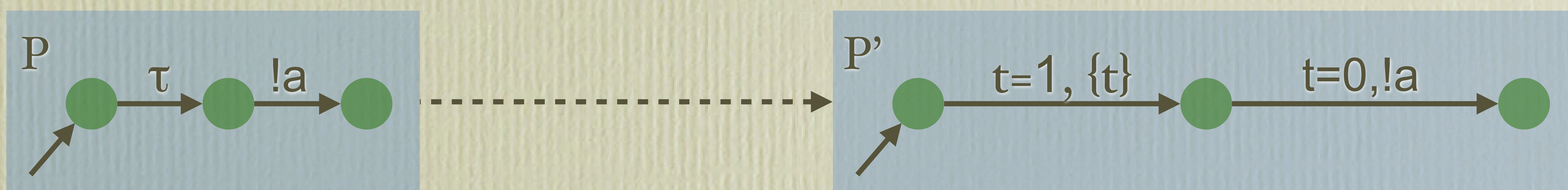

un-timing (exponential blow-up)
(polyforest topologies)
Network of Communicating Timed Automata
Network of Communicating Tick Automata
(any topology)

Dense time

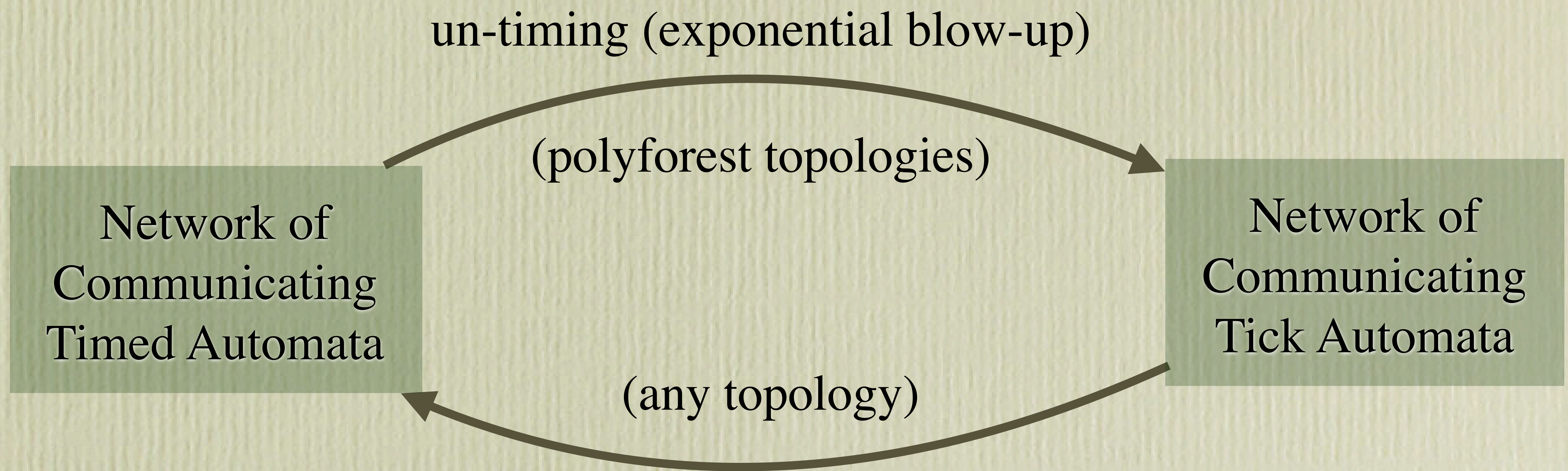
Network of Communicating Tick Automata → Network of Communicating Timed Automata

(any topology)

Simulate ticks locally by interpreting τ as one time unit



Dense time (summary)



Theorem: Reachability decidable iff polyforest topology

Theorem: The complexity is Petri net-hard and in Exp-Petri net

Extension: Urgency

Not shown in this presentation: **Urgent channels**

- Urgency: receptions have priority over internal actions.
- We show that urgent channel are equivalent to zero tests on counters.

Theorem (Discrete time): Rechability decidable iff at most 1 urgent channel for each weakly connected component.

$P \rightarrow Q \rightarrow R$

- [Krcal&Yi'06]: Reachability is undecidable in the urgent pipeline of length 2.

<u>Extension</u> (Dense time): Reachabiliy undecidable if any weakly connected component contains at least 2 urgent channels.

Future directions

- Our generic topology-preserving transformation dense --> discrete time does not work in the presence of urgency.
- Obtain a complete characterisation in the presence of urgency also for *dense time*.
- Study richer models where timestamps can be sent along the channel.