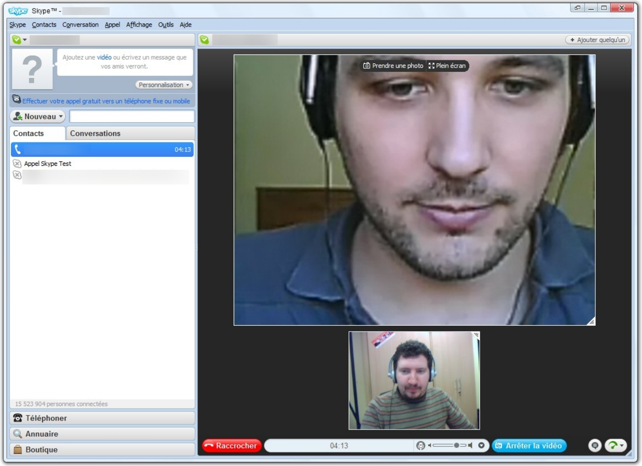
Synchronizability of Communicating Finite State Machines is not Decidable

A. Finkel E. Lozes
LSV, ENS Cachan

ICALP'2017 – Warsaw

Shared Memory

Synchronous Communications



Asynchronous Communication

friday

I think I found a bug in Lemma 36

saturday

...

10:00

I don't see how to fix that

10:01

Didn't you receive my fix?

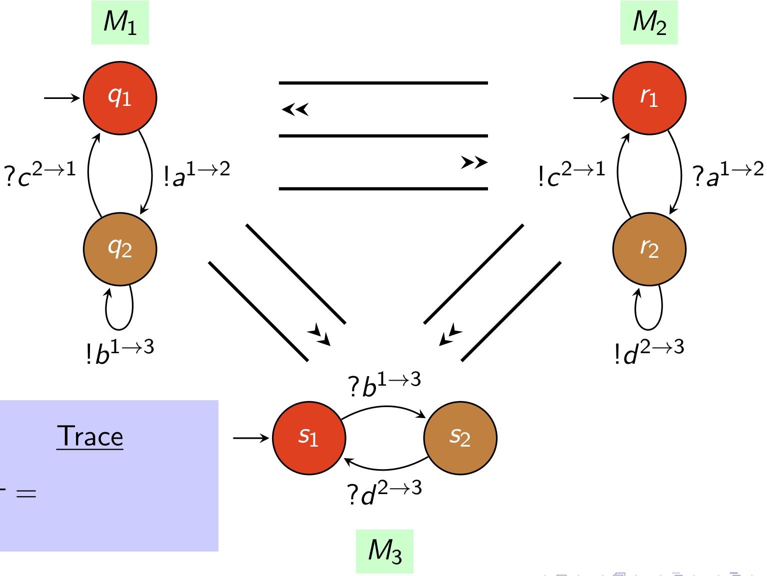
10:02

Do you know you ruined my week-end?

A model

- a network of machines that exchange messages asynchronously
- each machine M has a finite control state
- only one buffer for the messages sent by M_1 to M_2
- all buffers are independent of each others
- all buffers are FIFO queues

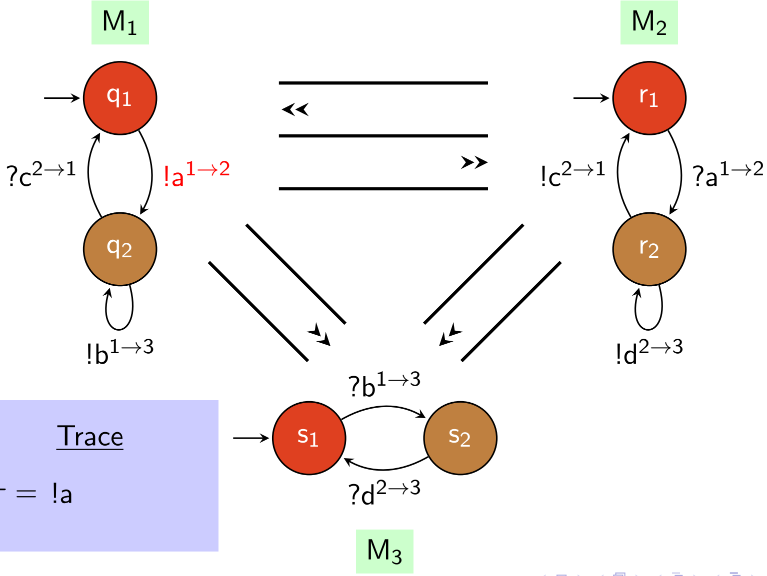
Example



Trace

$\tau =$

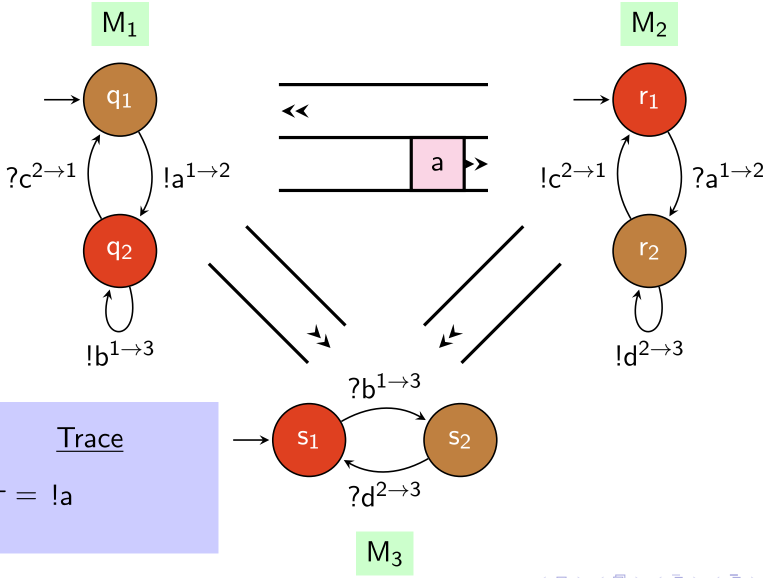
Example



<u>Trace</u>

$\tau =$!a

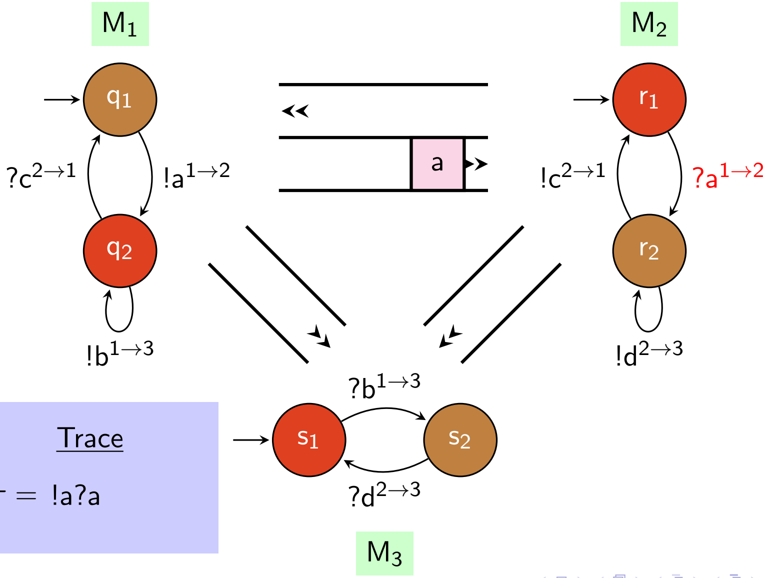
Example



Trace

$\tau = \; !a$

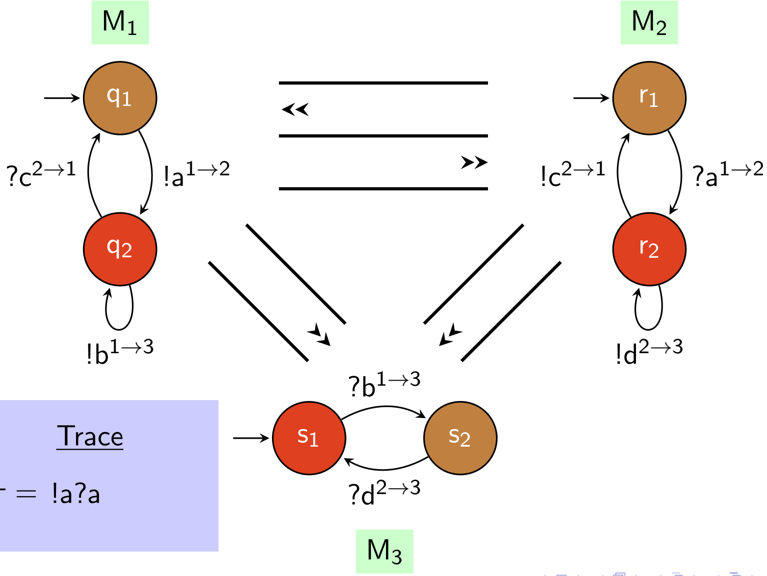
Example



Trace

$\tau =$!a?a

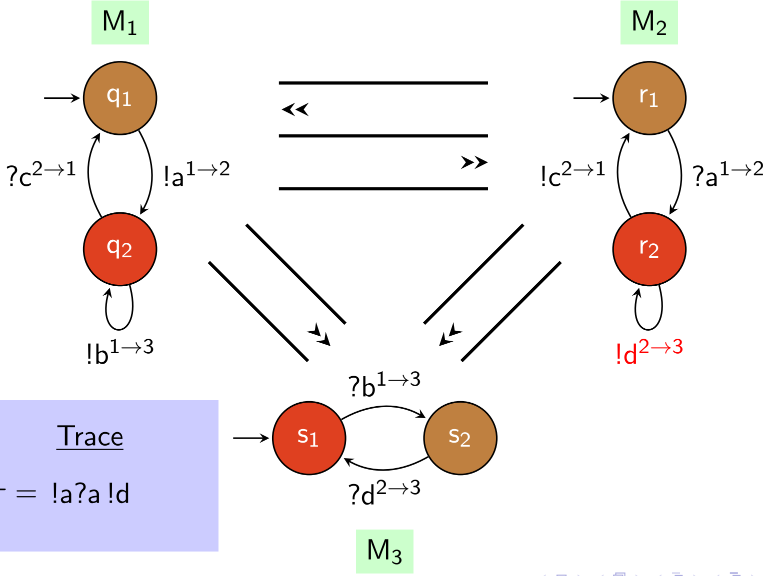
Example



Trace

$\tau =$!a?a

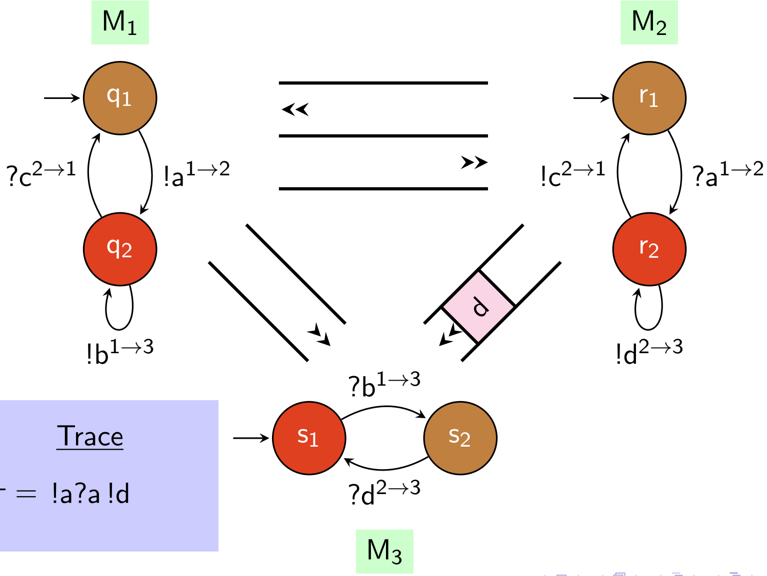
Example



Trace

$\tau =$!a?a !d

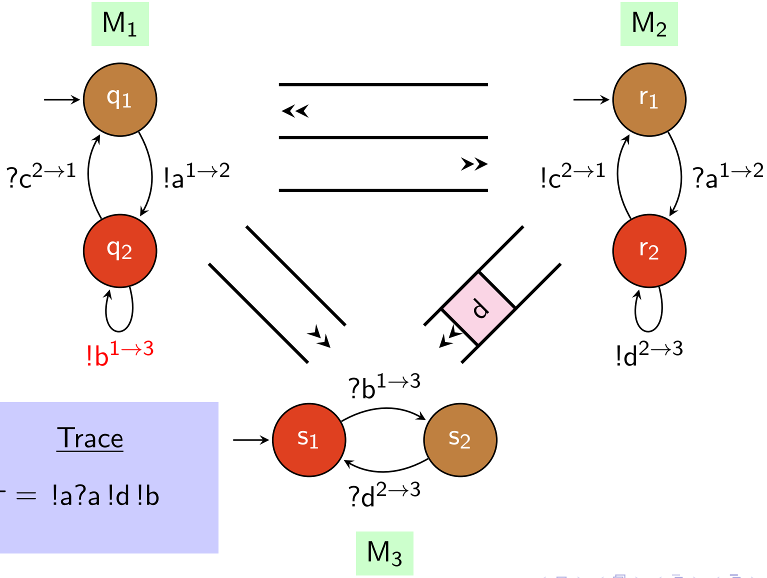
Example



Trace

$\tau = $!a?a !d

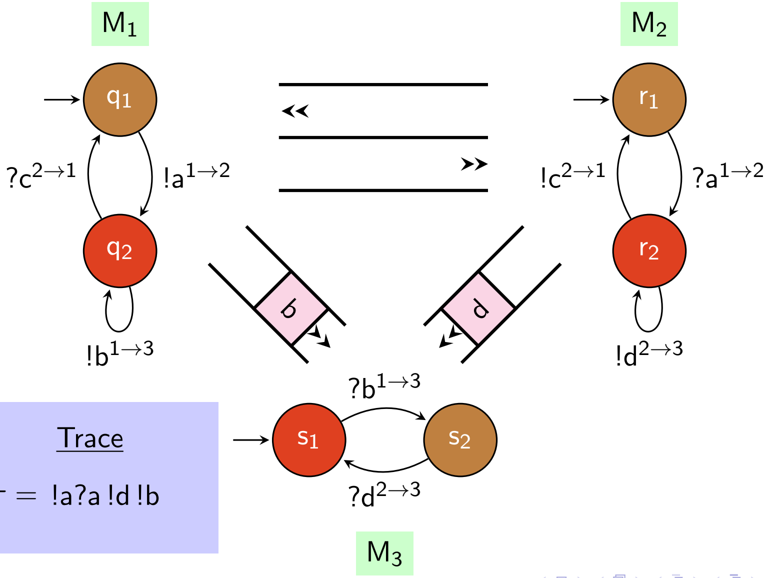
Example



Trace

$\tau =$!a?a !d !b

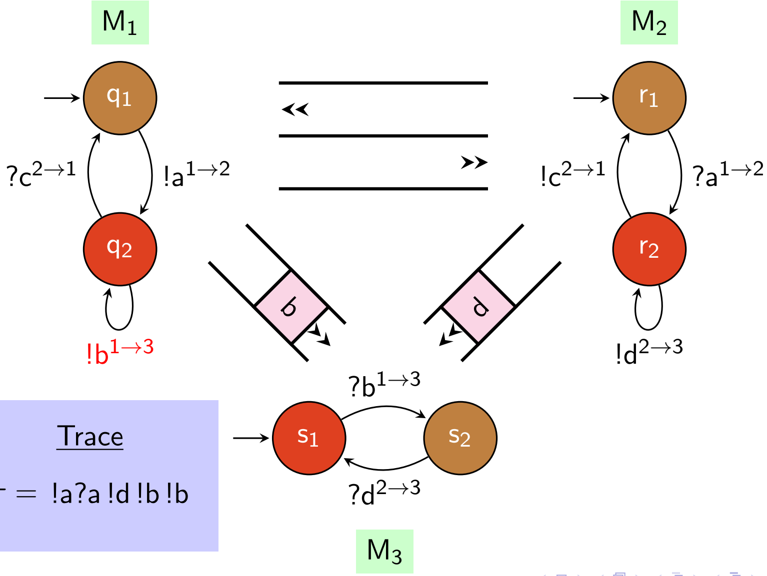
Example



<u>Trace</u>

$\tau =$!a?a !d !b

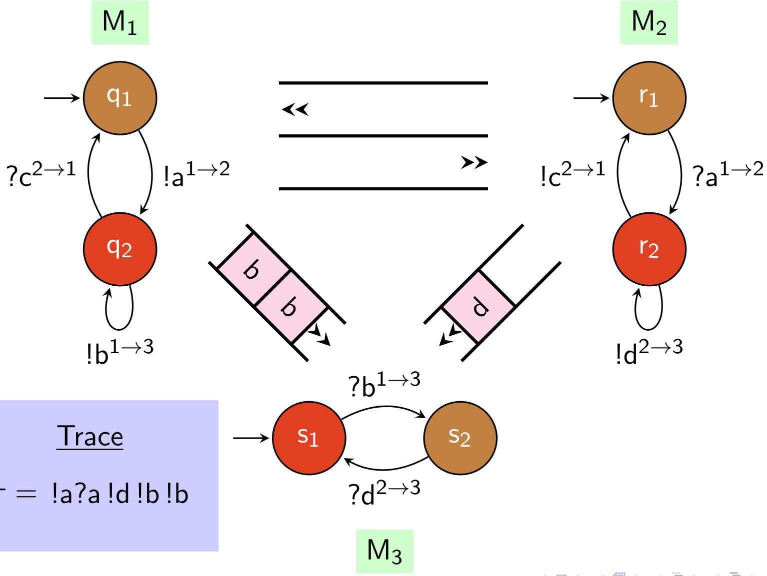
Example



Trace

$\tau =$!a?a !d !b !b

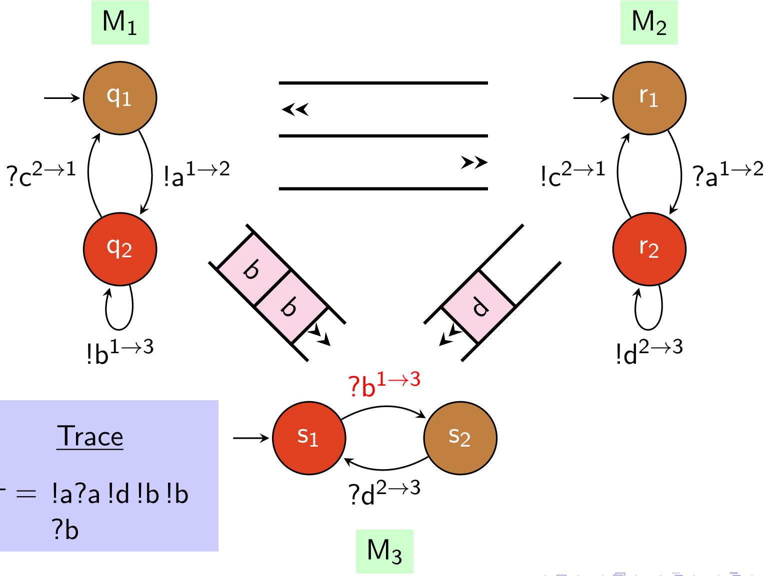
Example



Trace

$\tau =$!a?a !d !b !b

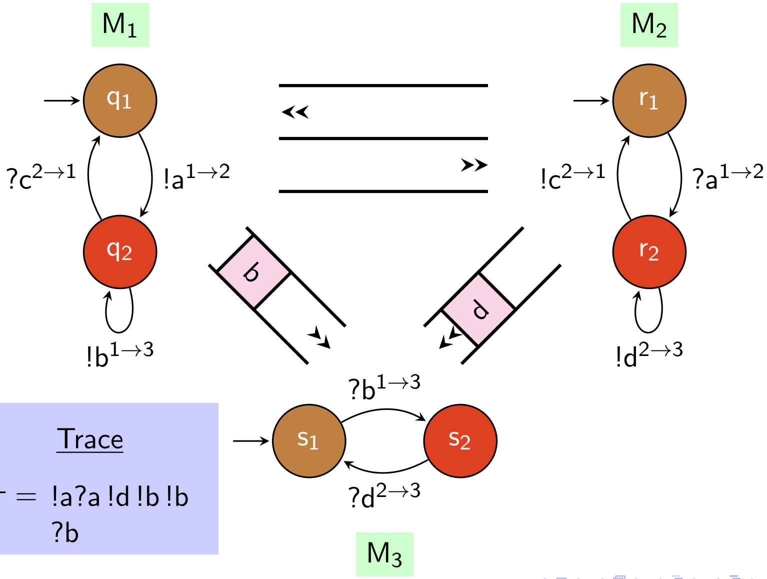
Example



<u>Trace</u>

$\tau =$!a?a !d !b !b ?b

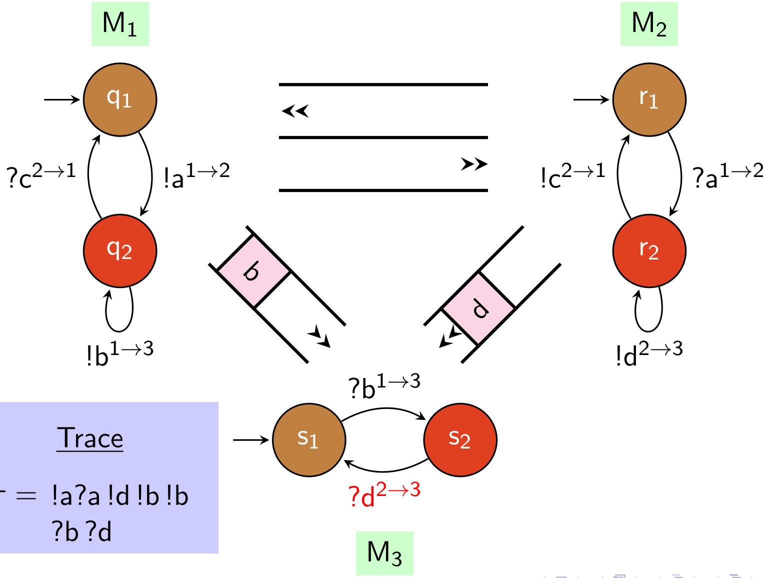
Example



Trace

$\tau =$!a?a !d !b !b ?b

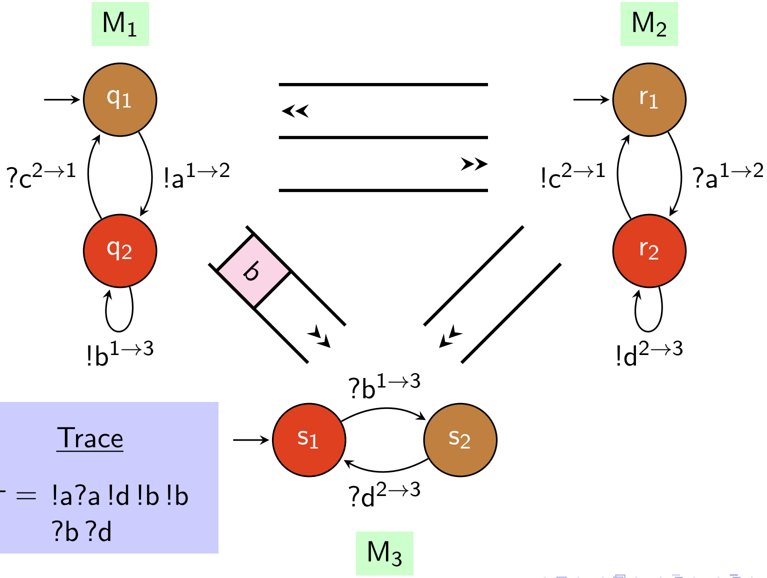
Example



Trace

$\tau =$!a?a !d !b !b ?b ?d

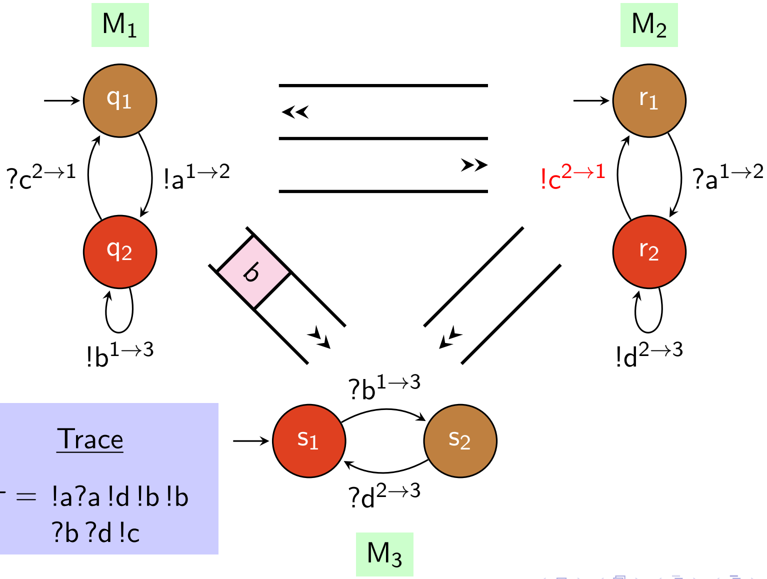
Example



Trace

$\tau =$!a?a !d !b !b ?b ?d

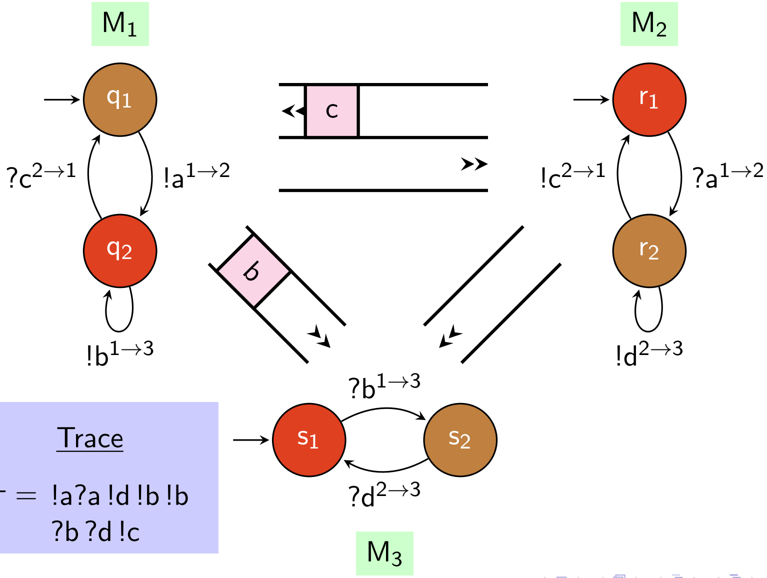
Example



Trace

$\tau =$!a?a !d !b !b ?b ?d !c

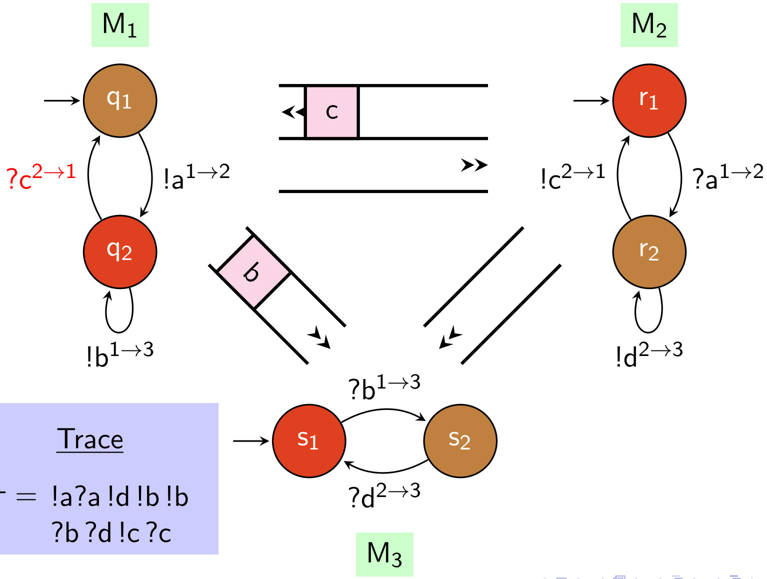
Example



Trace

$\tau =$!a ?a !d !b !b ?b ?d !c

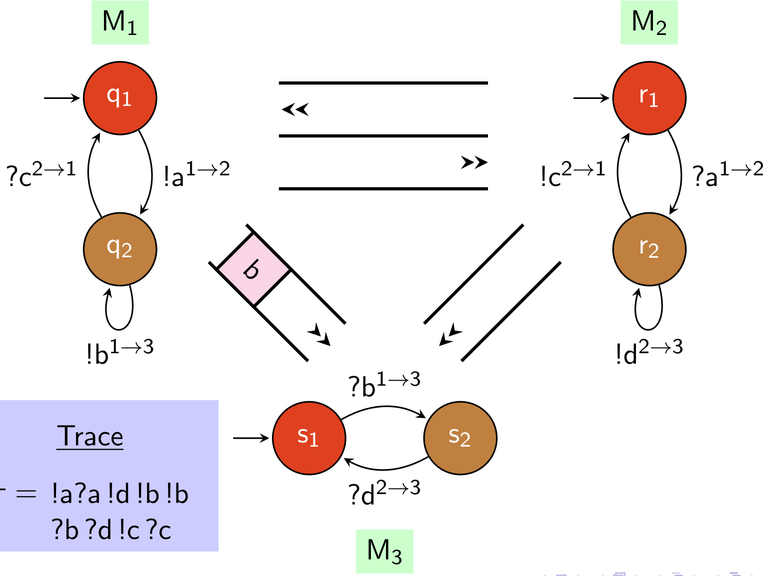
Example



<u>Trace</u>

$\tau =$!a ?a !d !b !b
?b ?d !c ?c

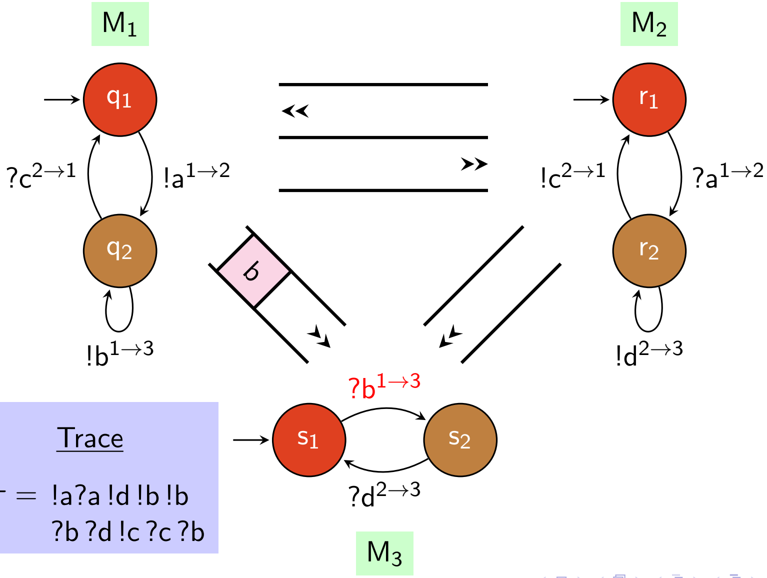
Example



Trace

$\tau =$!a?a !d !b !b
?b ?d !c ?c

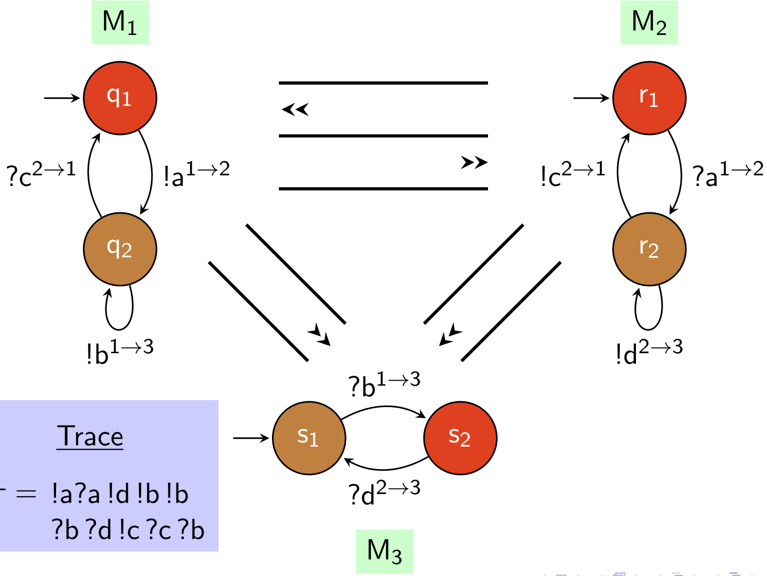
Example



<u>Trace</u>

$\tau =$!a?a !d !b !b ?b ?d !c ?c ?b

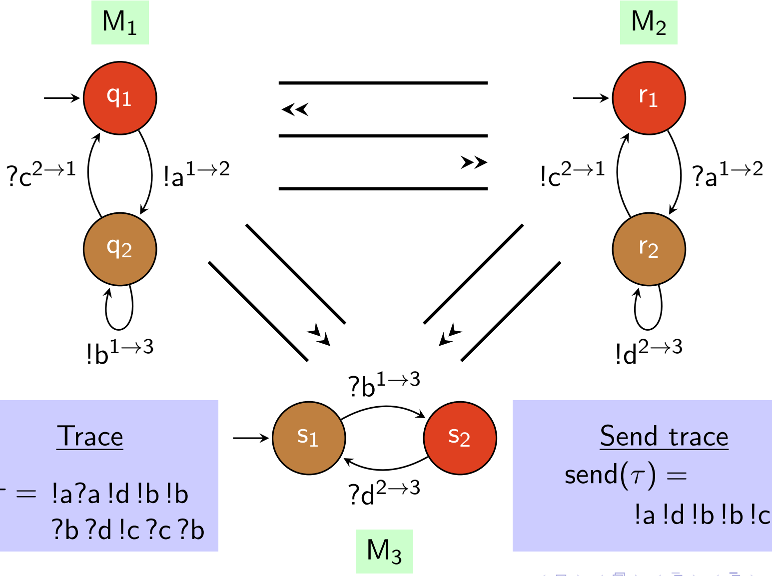
Example



Trace

$\tau =$!a ?a !d !b !b ?b ?d !c ?c ?b

Example



<u>Trace</u>

$\tau =$!a ?a !d !b !b ?b ?d !c ?c ?b

<u>Send trace</u>

$\text{send}(\tau) =$!a !d !b !b !c

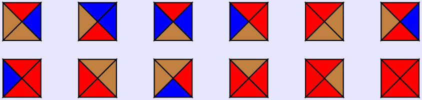
Verification Problems

- is there a bound on the size of the queues (for all runs) ?
- is there a run where a message is sent but never received?
- is there a run where a machine receives an unexpected message?
- is there a reachable configuration where all machines wait for messages but the queues are empty?
- ...

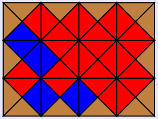
All these questions (and many others) are undecidable
[Brand Zafiropulo, JACM 1983]

CFSM and Tiling Problems

Set of tiles

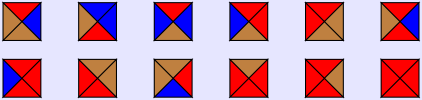


Solution

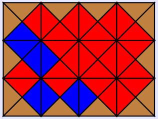


CFSM and Tiling Problems

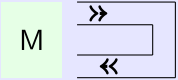
Set of tiles



Solution



CFSM

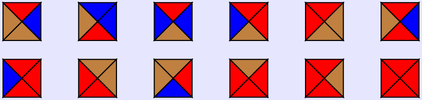


The machine M guesses the solution

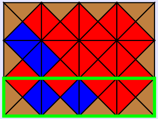
The other machine is a forwarder.

CFSM and Tiling Problems

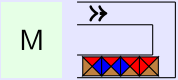
Set of tiles



Solution



CFSM



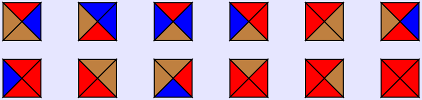
Guess first row and queue it

The machine M guesses the solution

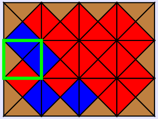
The other machine is a forwarder.

CFSM and Tiling Problems

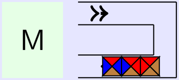
Set of tiles



Solution



CFSM



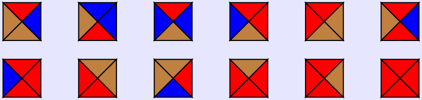
Guess tile above

The machine M guesses the solution

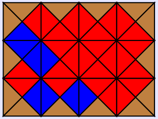
The other machine is a forwarder.

CFSM and Tiling Problems

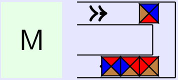
Set of tiles



Solution



CFSM



Guess tile above

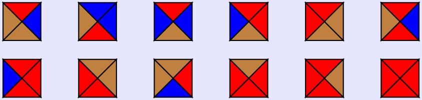
and queue it

The machine M guesses the solution

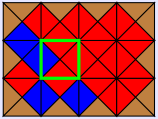
The other machine is a forwarder.

CFSM and Tiling Problems

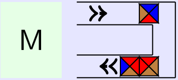
Set of tiles



Solution



CFSM



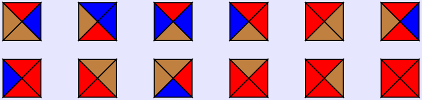
Guess tile above

The machine M guesses the solution

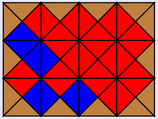
The other machine is a forwarder.

CFSM and Tiling Problems

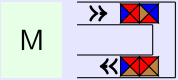
Set of tiles



Solution



CFSM



Guess tile above

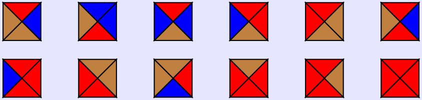
and queue it

The machine M guesses the solution

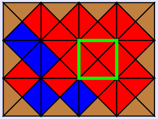
The other machine is a forwarder.

CFSM and Tiling Problems

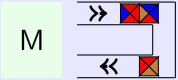
Set of tiles



Solution



CFSM



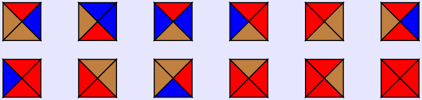
Guess tile above

The machine M guesses the solution

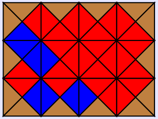
The other machine is a forwarder.

CFSM and Tiling Problems

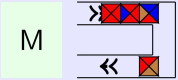
Set of tiles



Solution



CFSM



Guess tile above

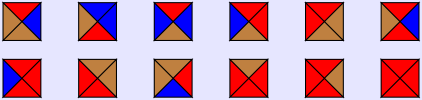
and queue it

The machine M guesses the solution

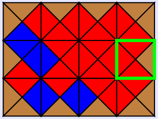
The other machine is a forwarder.

CFSM and Tiling Problems

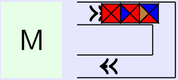
Set of tiles



Solution



CFSM



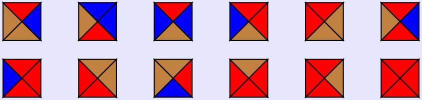
Guess tile above

The machine M guesses the solution

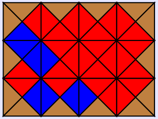
The other machine is a forwarder.

CFSM and Tiling Problems

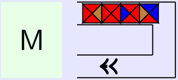
Set of tiles



Solution



CFSM



Guess tile above

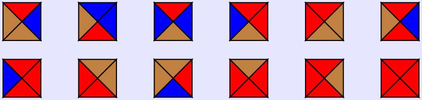
and queue it

The machine M guesses the solution

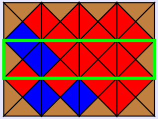
The other machine is a forwarder.

CFSM and Tiling Problems

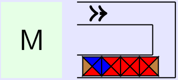
Set of tiles



Solution



CFSM



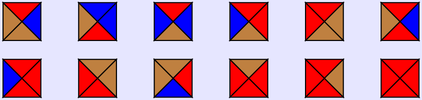
Start again
with the next row

The machine M guesses the solution

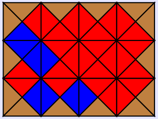
The other machine is a forwarder.

CFSM and Tiling Problems

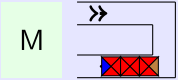
Set of tiles



Solution



CFSM

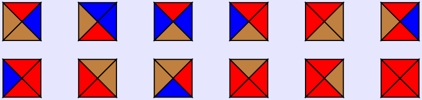


The machine M guesses the solution

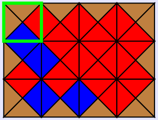
The other machine is a forwarder.

CFSM and Tiling Problems

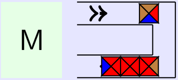
Set of tiles



Solution



CFSM

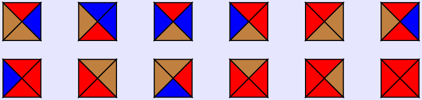


The machine M guesses the solution

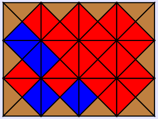
The other machine is a forwarder.

CFSM and Tiling Problems

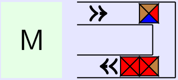
Set of tiles



Solution



CFSM

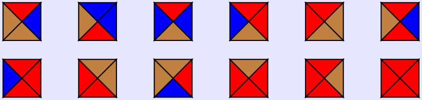


The machine M guesses the solution

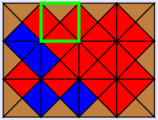
The other machine is a forwarder.

CFSM and Tiling Problems

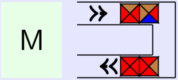
Set of tiles



Solution



CFSM

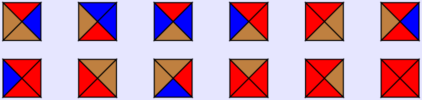


The machine M guesses the solution

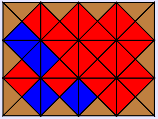
The other machine is a forwarder.

CFSM and Tiling Problems

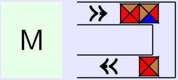
Set of tiles



Solution



CFSM

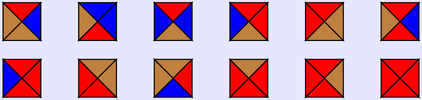


The machine M guesses the solution

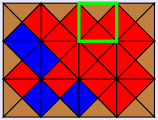
The other machine is a forwarder.

CFSM and Tiling Problems

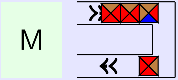
Set of tiles



Solution



CFSM

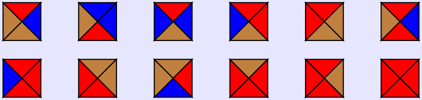


The machine M guesses the solution

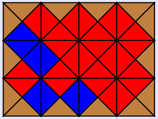
The other machine is a forwarder.

CFSM and Tiling Problems

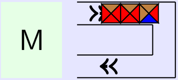
Set of tiles



Solution



CFSM

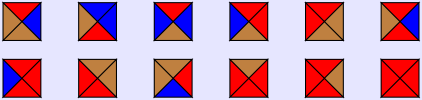


The machine M guesses the solution

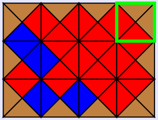
The other machine is a forwarder.

CFSM and Tiling Problems

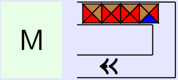
Set of tiles



Solution



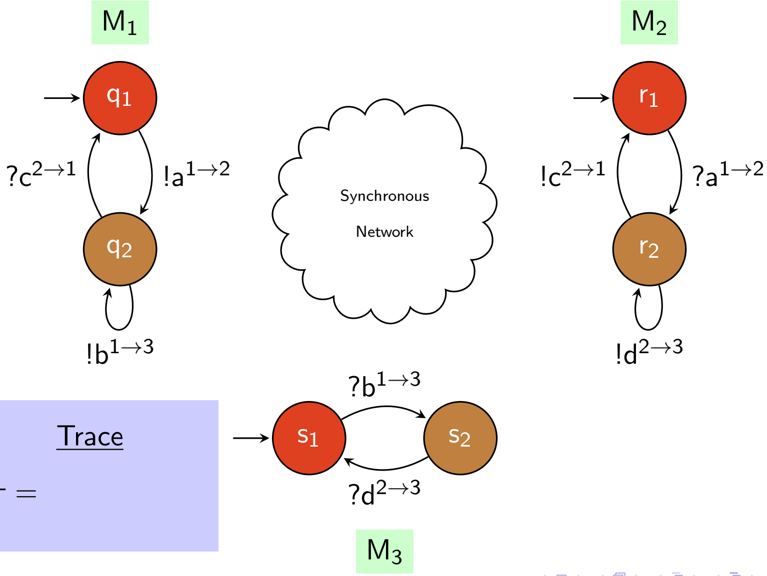
CFSM



The machine M guesses the solution

The other machine is a forwarder.

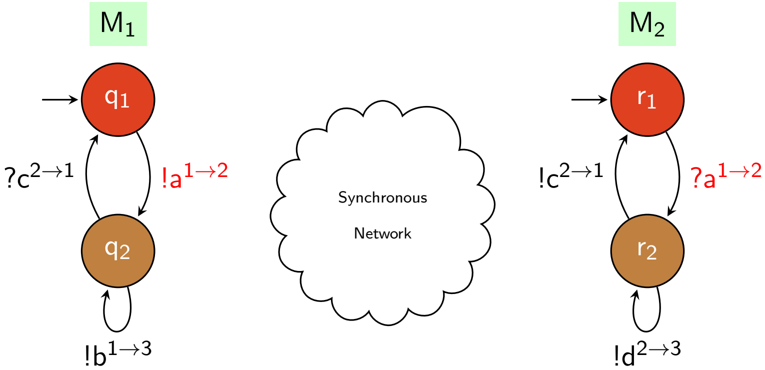
Synchronous Execution

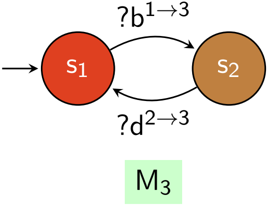


Trace

$\tau =$

Synchronous Execution

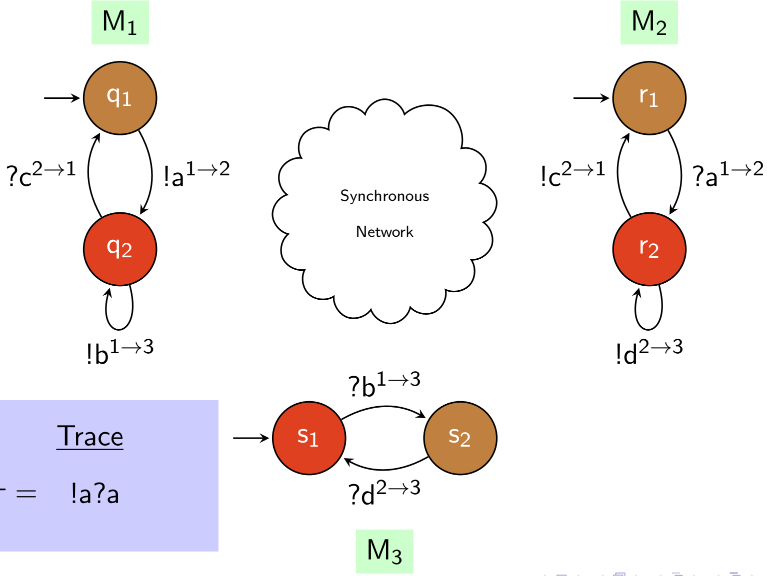




Trace

$\tau =$!a?a

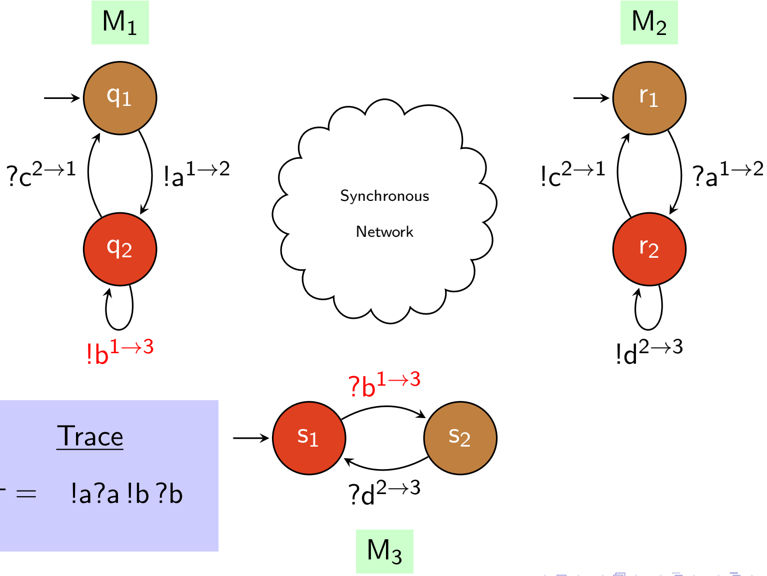
Synchronous Execution



Trace

$\tau = \;\; !a?a$

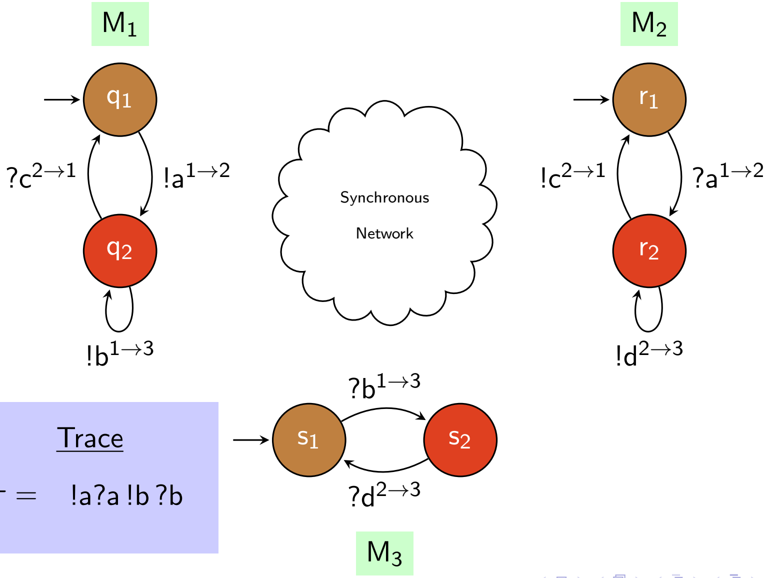
Synchronous Execution



Trace

$\tau =$!a?a !b ?b

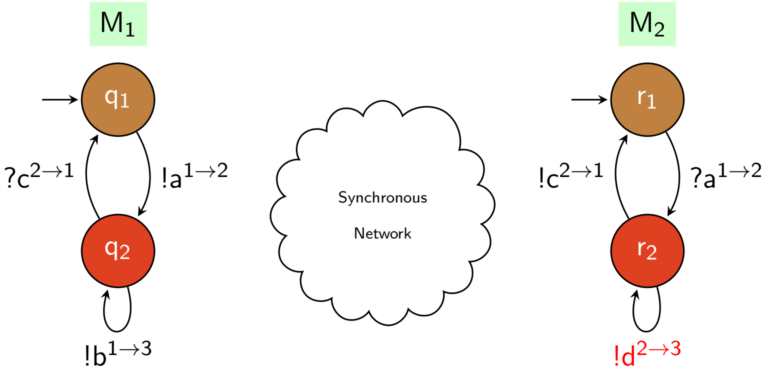
Synchronous Execution



<u>Trace</u>

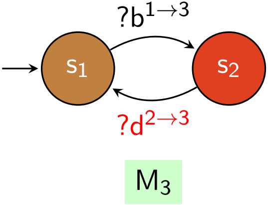
$\tau =$!a?a !b ?b

Synchronous Execution

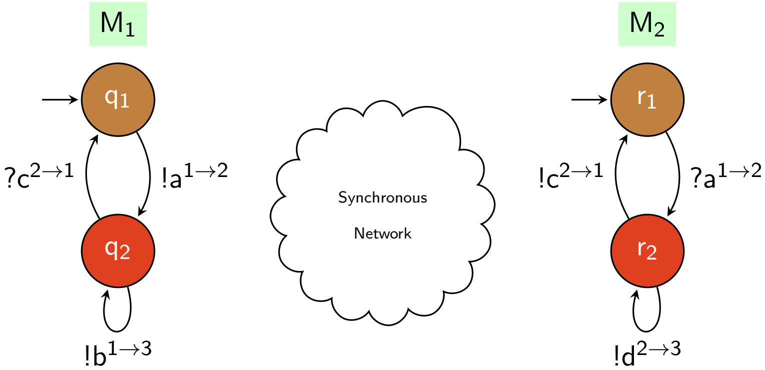


Trace

$\tau =$!a?a !b ?b
!d ?d

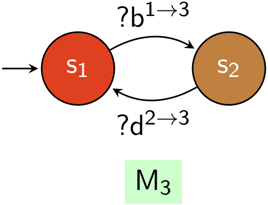


Synchronous Execution

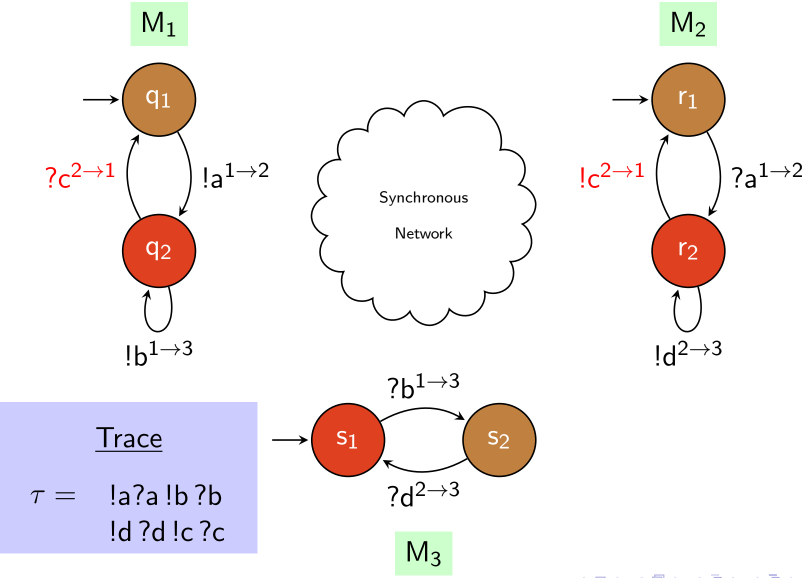


Trace

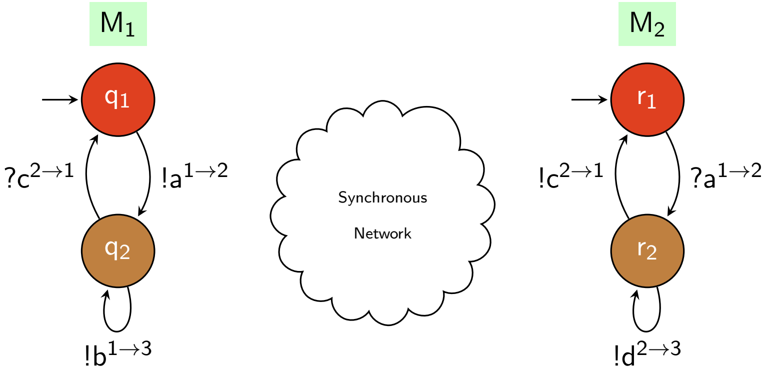
$\tau =$!a ?a !b ?b !d ?d



Synchronous Execution

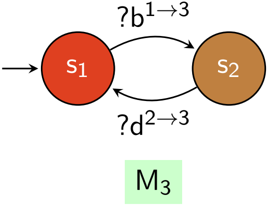


Synchronous Execution



Trace

$\tau =$!a ?a !b ?b !d ?d !c ?c



Send trace

$send(\tau) =$!a !b !d !c

Finite Transition System

- the transition system of a synchronous system is finite and effective
- this is also true for an asynchronous system with bounded buffers
- all previous verification problems become decidable
- but what can we do if the buffers are unbounded?

Slack Elasticity for CSP (1998)

Slack Elasticity in Concurrent Computing

Rajit Manohar and Alain J. Martin

California Institute of Technology, Pasadena CA 91125, USA

Abstract. We present conditions under which we can modify the slack of a channel in a distributed computation without changing its behavior. These results can be used to modify the degree of pipelining in an asynchronous system. The generality of the result shows the wide variety of pipelining alternatives presented to the designer of a concurrent system. We give examples of program transformations which can be used in the design of concurrent systems whose correctness depends on the conditions presented.

A system is slack elastic if it *behaves as if* it were synchronous.

Slack Elasticity for MPI (2010)

Precise Dynamic Analysis for Slack Elasticity: Adding Buffering without Adding Bugs*

Sarvani Vakkalanka, Anh Vo, Ganesh Gopalakrishnan, and Robert M. Kirby

School of Computing, Univ. of Utah, Salt Lake City, UT 84112, USA

Abstract. Increasing the amount of buffering for MPI sends is an effective way to improve the performance of MPI programs. However, for programs containing non-deterministic operations, this can result in *new* deadlocks or other safety assertion violations. Previous work did not provide any characterization of the space of *slack elastic* programs: those for which buffering can be safely added. In this paper, we offer a precise characterization of slack elasticity based on our formulation of MPI's *happens before* relation. We show how to efficiently locate *potential culprit sends* in such programs: MPI sends for which adding buffering can increase overall program non-determinism and cause new bugs. We present a procedure

Synchronizability for choreographies (2011)

Choreography Conformance via Synchronizability

Samik Basu
Department of Computer Science
Iowa State University
Ames, IA 50011, USA
sbasu@cs.iastate.edu

Tevfik Bultan
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
bultan@cs.ucsb.edu

ABSTRACT

Choreography analysis has been a crucial problem in service oriented computing. Interactions among services involve message exchanges across organizational boundaries in a distributed computing environment, and in order to build such systems in a reliable manner, it is necessary to develop techniques for analyzing such interactions. Choreography conformance involves verifying that a set of services behave according to a given choreography specification that characterizes their interactions. Unfortunately this is an undecidable problem when services interact with asynchronous communication. In this paper we present techniques that iden-

tify if the interaction behavior for a set of services remain the same when asynchronous communication is replaced with synchronous communication. This is called the synchronizability problem and determining the synchronizability of a set of services has been an open problem for several years. We solve this problem in this paper. Our results can be used to identify synchronizable services for which choreography conformance can be checked efficiently. Our results on synchronizability are applicable to any software infrastructure that supports message-based interactions.

Synchronizability : definition

Notations

- $\mathcal{I}_0$: set of send traces along any synchronous execution
- $\mathcal{I}_\omega$: set of send traces along any asynchronous execution
- $\mathcal{I}_\mathsf{k}$: set of send traces along any asynchronous execution with buffers bounded to size k

Observation

$$\mathcal{I}_0 \subseteq \mathcal{I}_1 \subseteq \mathcal{I}_2 \subseteq \cdots \subseteq \mathcal{I}_\omega$$

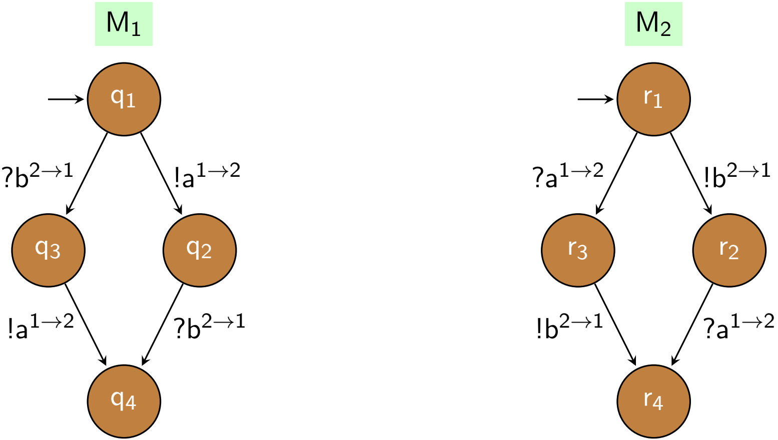
Definition

a system is synchronizable if $\mathcal{I}_0 = \mathcal{I}_\omega$

Example: not synchronizable

- $\mathcal{I}_0 = \{\epsilon\}$
- $\mathcal{I}_1 = \{\epsilon, \mathsf{a}, \mathsf{b}, \mathsf{ab}, \mathsf{ba}\}$

Example: synchronizable



$$\mathcal{I}_0 = \{\epsilon, \mathsf{a}, \mathsf{b}, \mathsf{ab}, \mathsf{ba}\} = \mathcal{I}_\omega$$

Why do we care about synchronizability?

a desirable property

- when the message-passing library gives no guarantee on (a)synchrony or buffer sizes
- when the system should run correctly in different networks

synchronizable systems are easier to verify?

- synchronizable systems are expected to be easy to verify
- it should not be too hard to check whether a system is synchronizable

Basu-Bultan conjecture

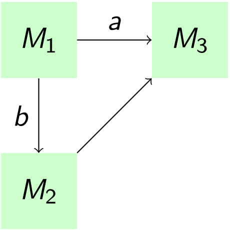
if $\mathcal{I}_0 = \mathcal{I}_1$, then $\mathcal{I}_0 = \mathcal{I}_\omega$

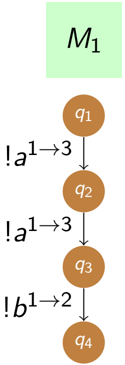
in particular, synchronizability would be decidable
(note that $\mathcal{I}_0$, $\mathcal{I}_1$ are regular)

several proof attempts
WWW'11, VMCAI'12, POPL'12, TCS'16,...

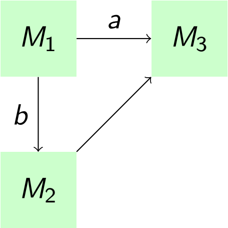
what about verification problems?
synchronizable ⇒ LTL model-checking <u>of send traces</u> is decidable

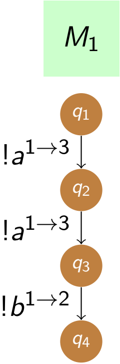
How to cook a counter-example





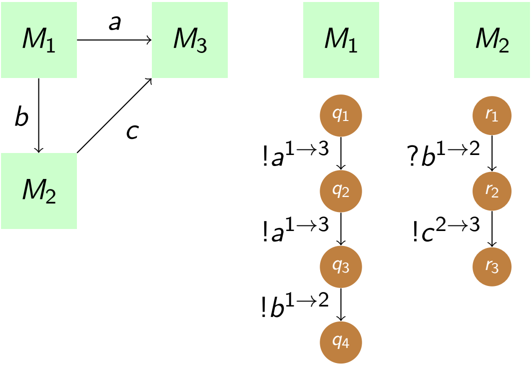
How to cook a counter-example



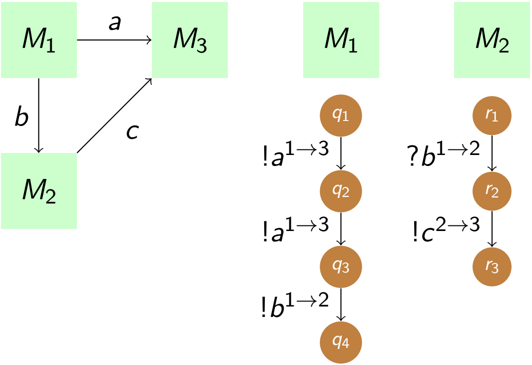


if buffer size ≥ 2 then $?b$ before $?a$ is possible

How to cook a counter-example

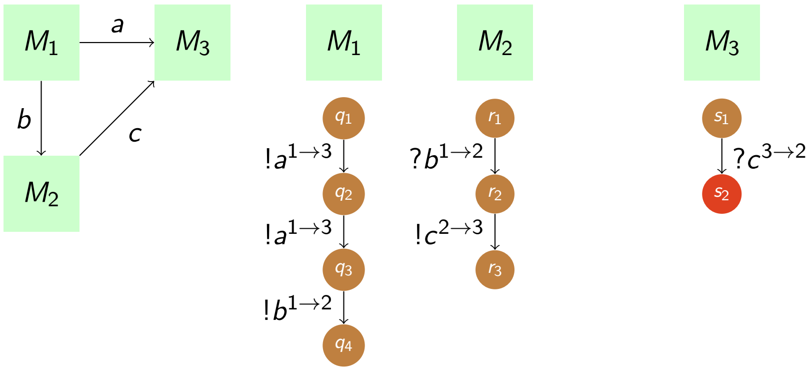


How to cook a counter-example

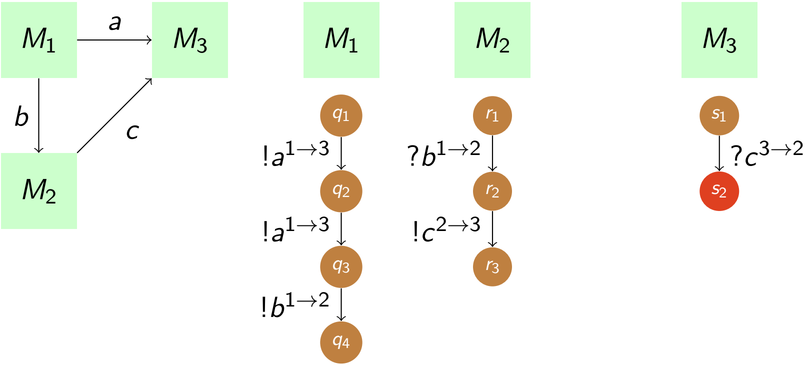


if buffer size ≥ 2 then $?c$ before $?a$ in M_3 is possible

How to cook a counter-example

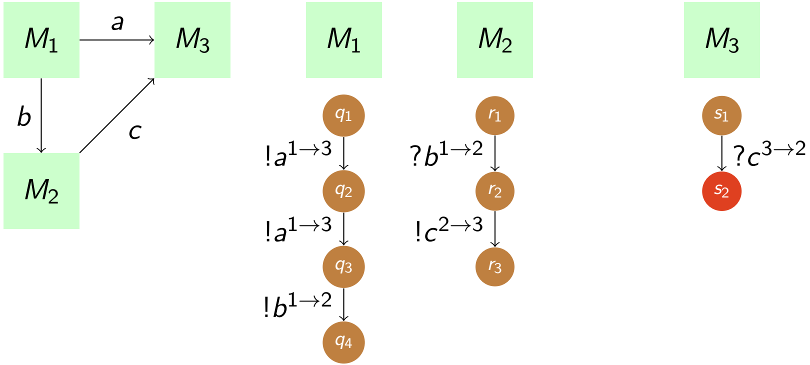


How to cook a counter-example



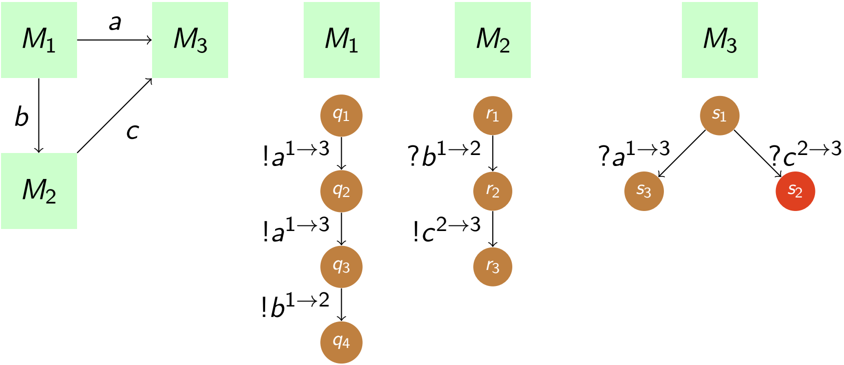
s_2 reachable iff buffer size ≥ 2

How to cook a counter-example

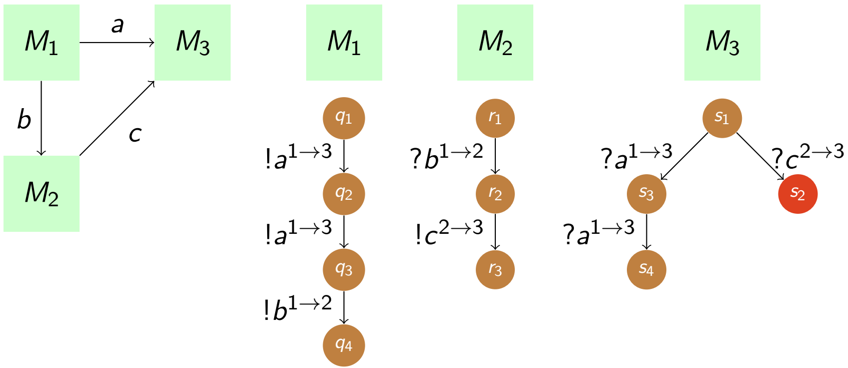


let's ensure $\mathcal{I}_0 = \mathcal{I}_1$

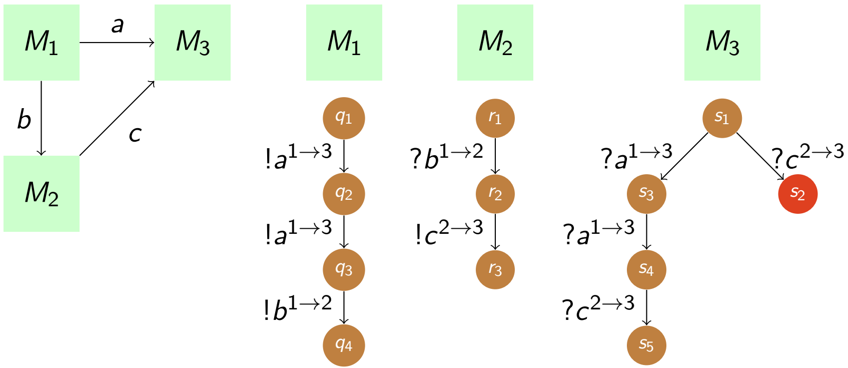
How to cook a counter-example



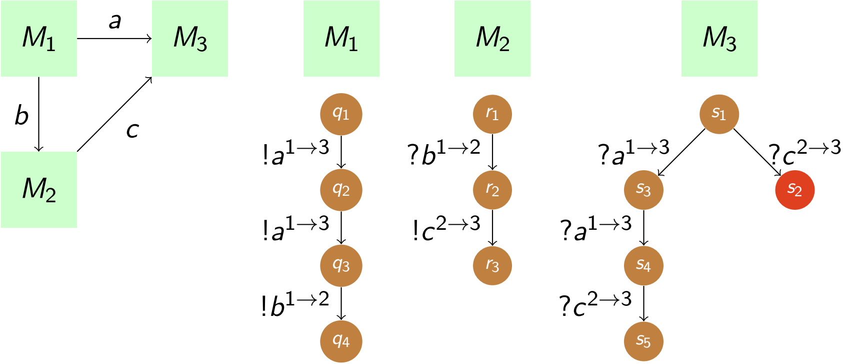
How to cook a counter-example



How to cook a counter-example

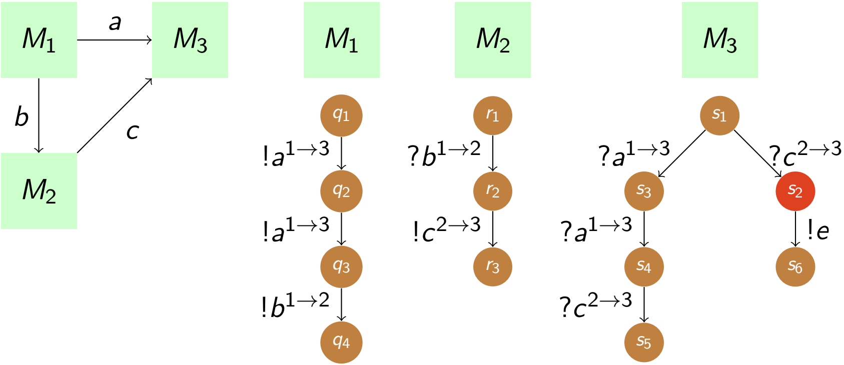


How to cook a counter-example

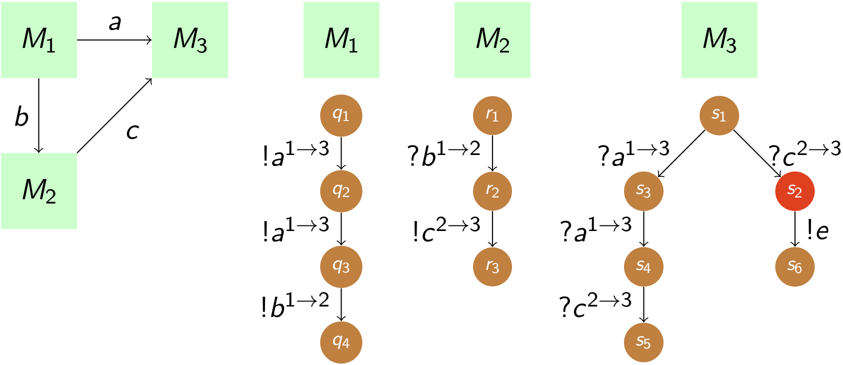


let's ensure $\mathcal{I}_0 \neq \mathcal{I}_2$

How to cook a counter-example



How to cook a counter-example



$aabce \in \mathcal{I}_2 \setminus \mathcal{I}_0$: not synchronizable, but $\mathcal{I}_0 = \mathcal{I}_1$

A limitation

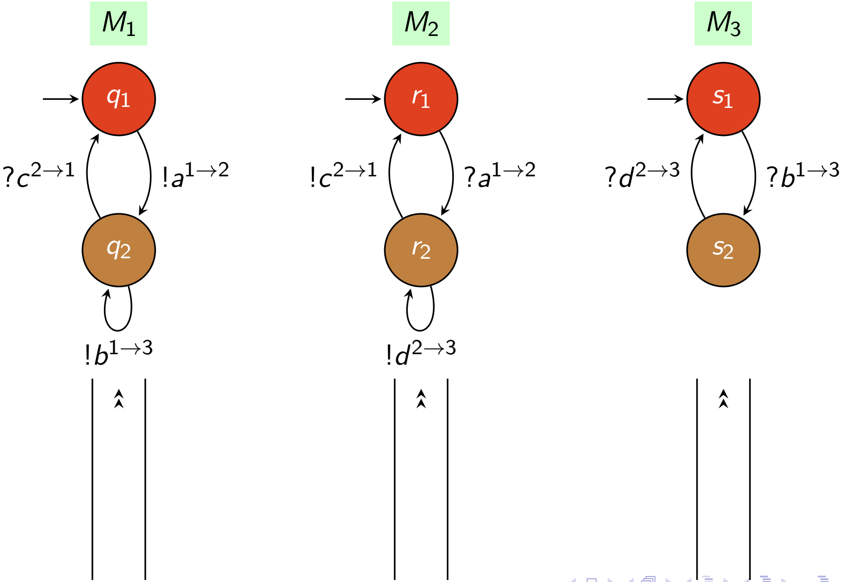
actually, this was just a counter-example for the conjecture for peer-to-peer communications

except in TCS'16, the communication model studied by Basu and Bultan was mailboxes

difference

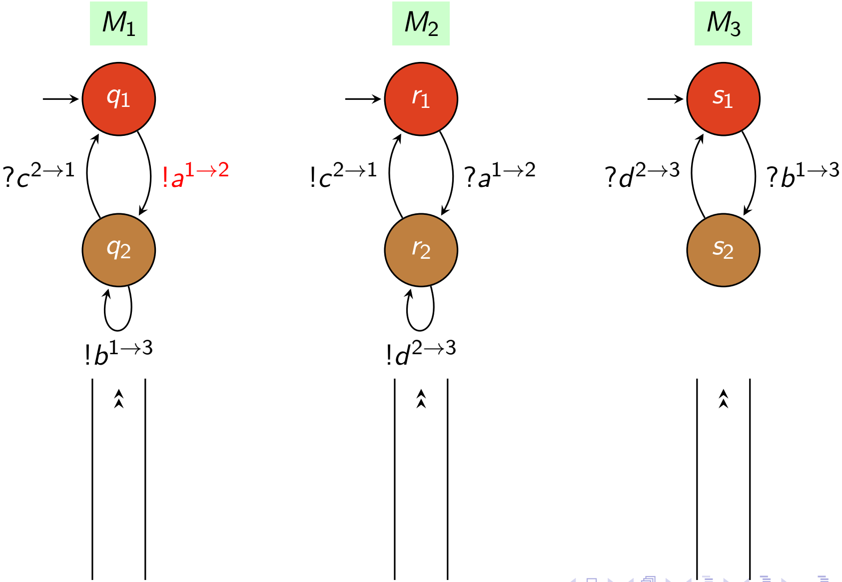
- every machine has exactly one mailbox
- all messages from other machines are merged in the mailbox
- mailboxes are still FIFO queues

Example

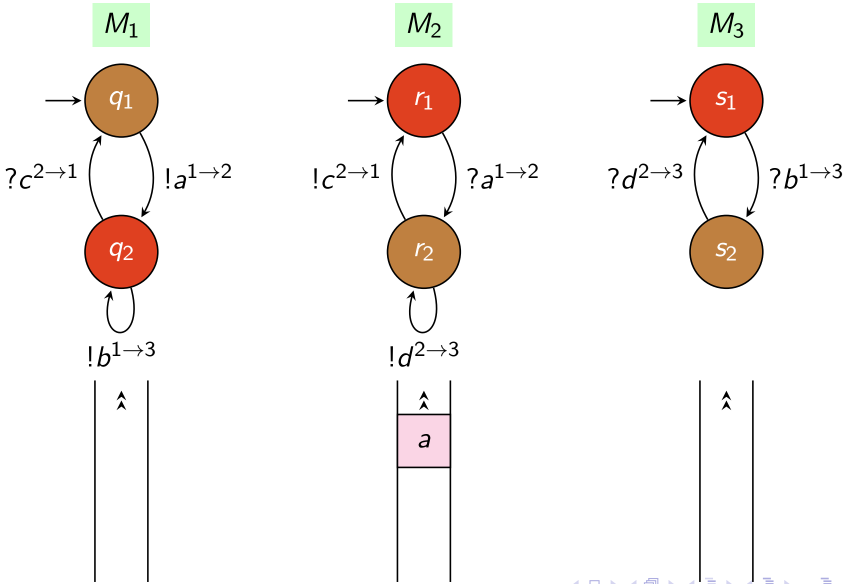


M_1
q_1
q_2
$?c^{2\to1}$
$!a^{1\to2}$
$!b^{1\to3}$
M_2
r_1
r_2
$!c^{2\to1}$
$?a^{1\to2}$
$!d^{2\to3}$
M_3
s_1
s_2
$?d^{2\to3}$
$?b^{1\to3}$

Example

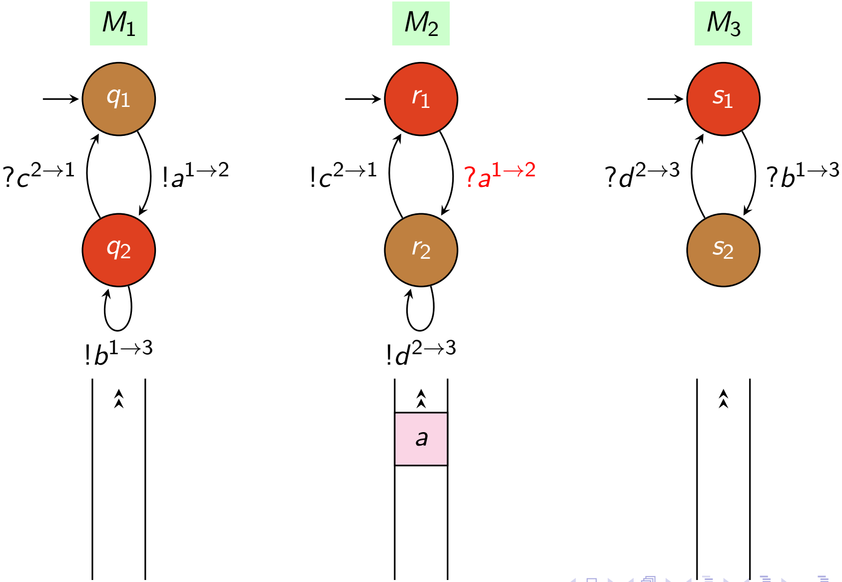

M_1
M_2
M_3
q_1
q_2
$?c^{2\to 1}$
$!a^{1\to 2}$
$!b^{1\to 3}$
r_1
r_2
$!c^{2\to 1}$
$?a^{1\to 2}$
$!d^{2\to 3}$
s_1
s_2
$?d^{2\to 3}$
$?b^{1\to 3}$

Example



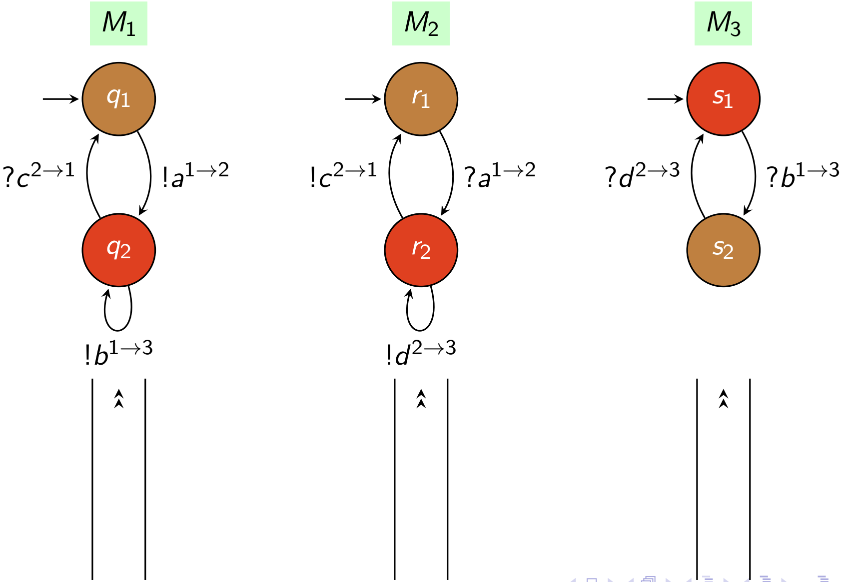
M_1
q_1
q_2
$?c^{2\to1}$
$!a^{1\to2}$
$!b^{1\to3}$
M_2
r_1
r_2
$!c^{2\to1}$
$?a^{1\to2}$
$!d^{2\to3}$
a
M_3
s_1
s_2
$?d^{2\to3}$
$?b^{1\to3}$

Example



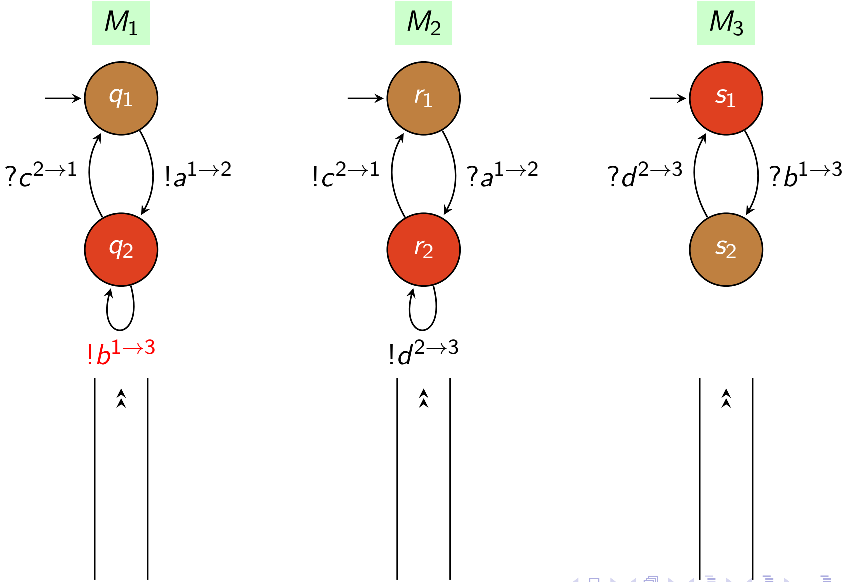
M_1
M_2
M_3
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
a
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2

Example

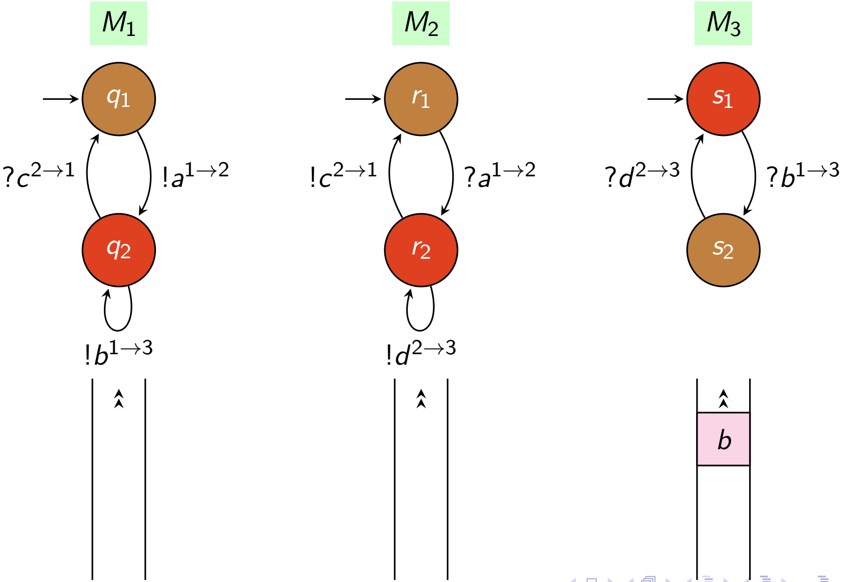


M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2

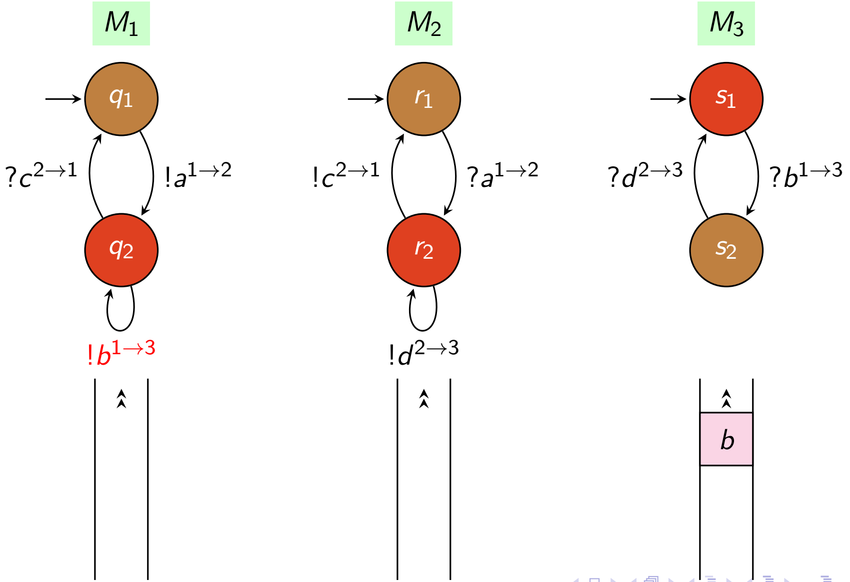
Example


M_1
q_1
q_2
$?c^{2\to1}$
$!a^{1\to2}$
$!b^{1\to3}$
M_2
r_1
r_2
$!c^{2\to1}$
$?a^{1\to2}$
$!d^{2\to3}$
M_3
s_1
s_2
$?d^{2\to3}$
$?b^{1\to3}$

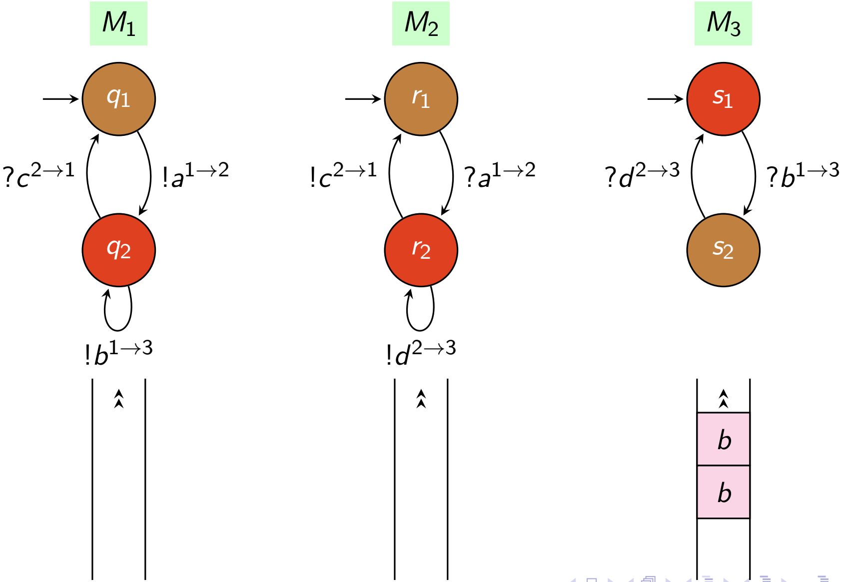
Example


M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2
b

Example

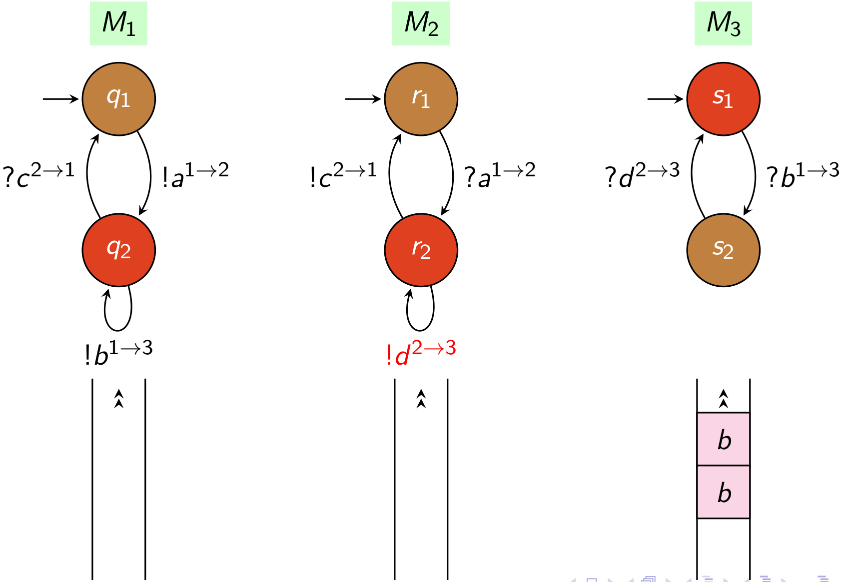

M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2
b

Example



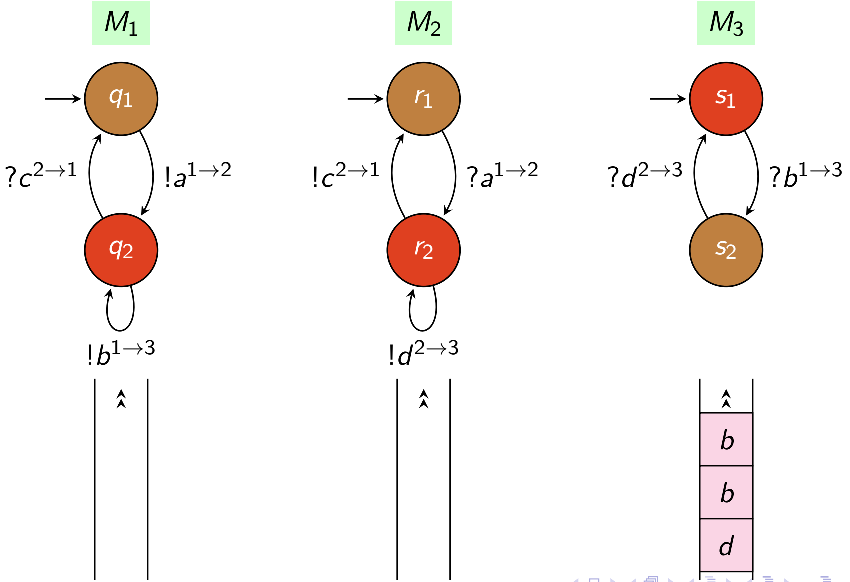
M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2
b
b

Example



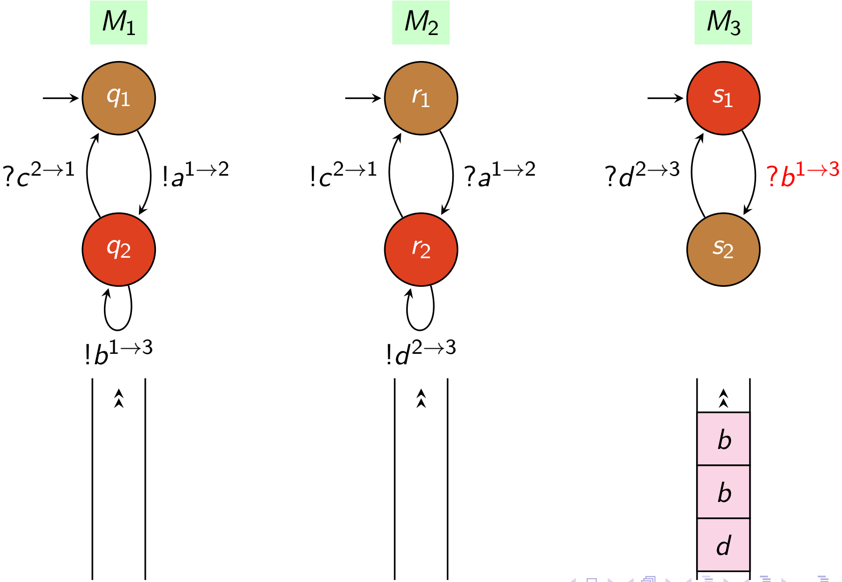
M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2
b
b

Example



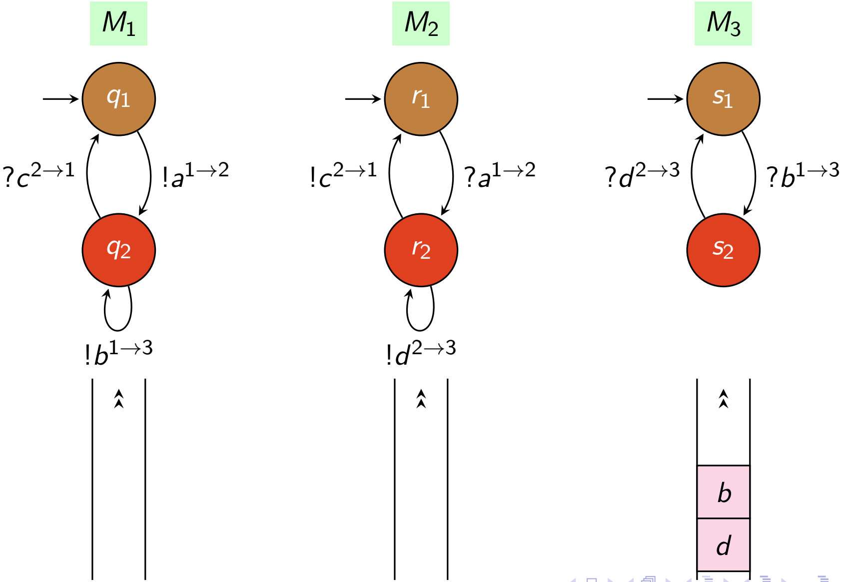
M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2
b
b
d

Example



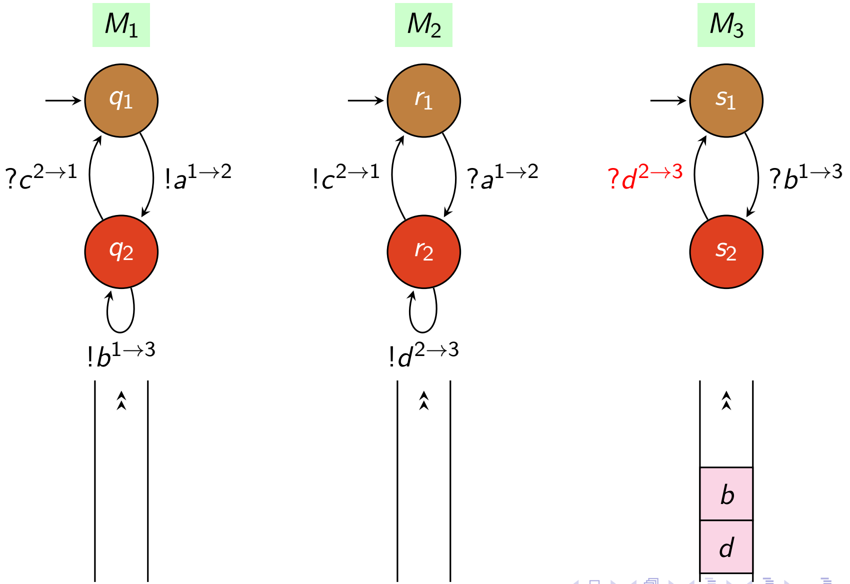
M_1
M_2
M_3
q_1
q_2
$?c^{2\to1}$
$!a^{1\to2}$
$!b^{1\to3}$
r_1
r_2
$!c^{2\to1}$
$?a^{1\to2}$
$!d^{2\to3}$
s_1
s_2
$?d^{2\to3}$
$?b^{1\to3}$
b
b
d

Example



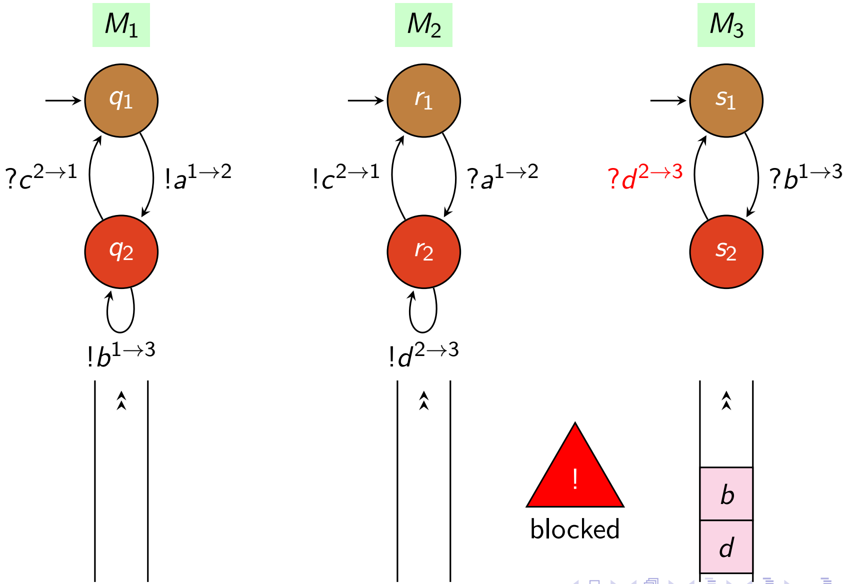
M_1
q_1
q_2
$?c^{2\to1}$
$!a^{1\to2}$
$!b^{1\to3}$
M_2
r_1
r_2
$!c^{2\to1}$
$?a^{1\to2}$
$!d^{2\to3}$
M_3
s_1
s_2
$?d^{2\to3}$
$?b^{1\to3}$
b
d

Example



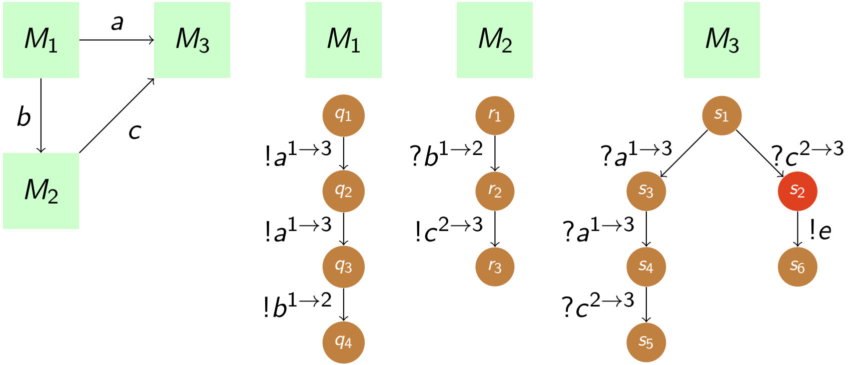
M_1
q_1
$?c^{2\to1}$
$!a^{1\to2}$
q_2
$!b^{1\to3}$
M_2
r_1
$!c^{2\to1}$
$?a^{1\to2}$
r_2
$!d^{2\to3}$
M_3
s_1
$?d^{2\to3}$
$?b^{1\to3}$
s_2
b
d

Example

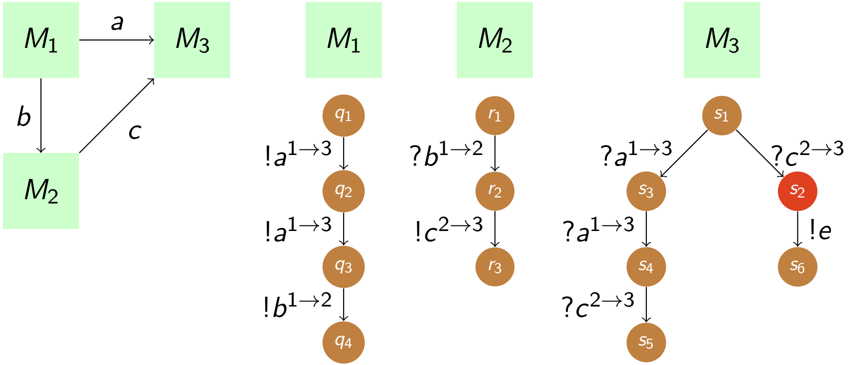


M_1
M_2
M_3
q_1
q_2
$?c^{2\to1}$
$!a^{1\to2}$
$!b^{1\to3}$
r_1
r_2
$!c^{2\to1}$
$?a^{1\to2}$
$!d^{2\to3}$
s_1
s_2
$?d^{2\to3}$
$?b^{1\to3}$
!
blocked
b
d

Back to the counter-example

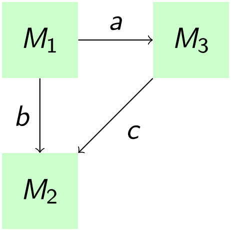


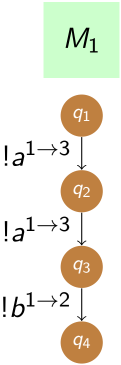
Back to the counter-example



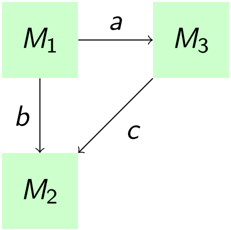
problem : now s_2 is not reachable!

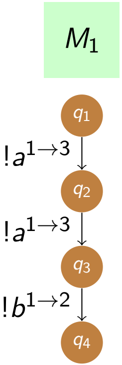
Adapting the counter-example to mailboxes





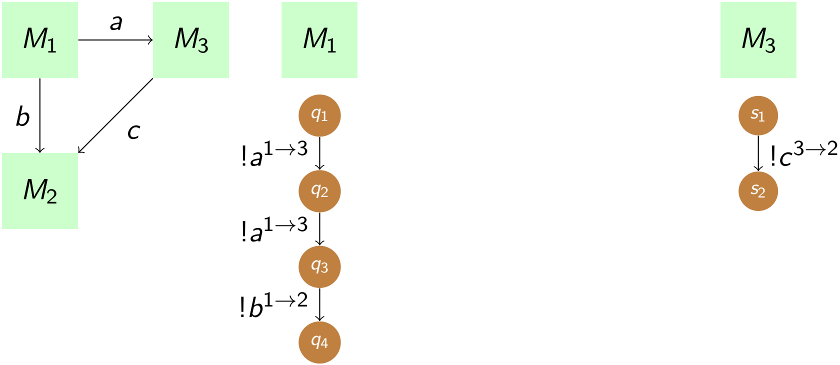
Adapting the counter-example to mailboxes



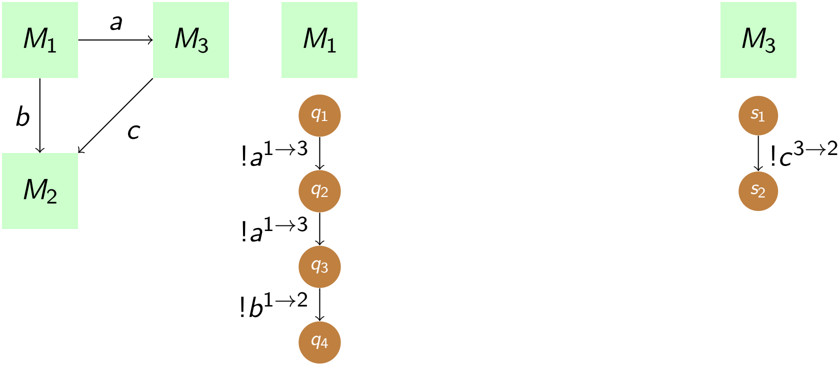


let's make M_3 compete with M_1

Adapting the counter-example to mailboxes

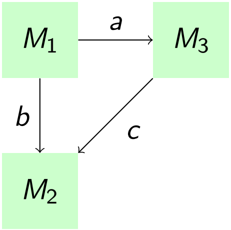


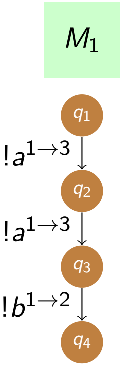
Adapting the counter-example to mailboxes

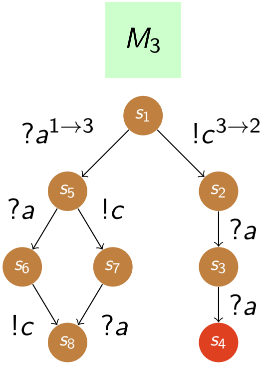


let's also prepare for $\mathcal{I}_0 = \mathcal{I}_1$

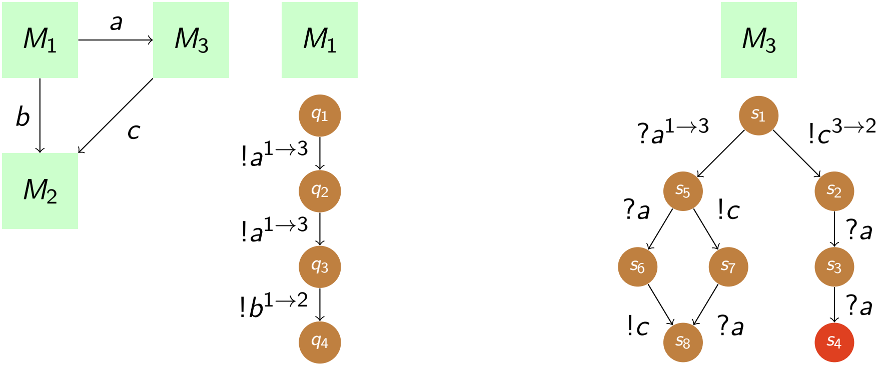
Adapting the counter-example to mailboxes





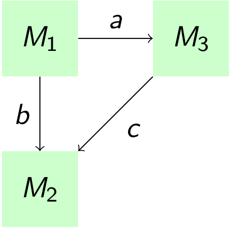


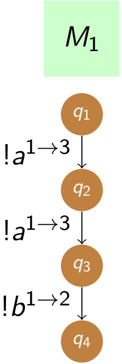
Adapting the counter-example to mailboxes

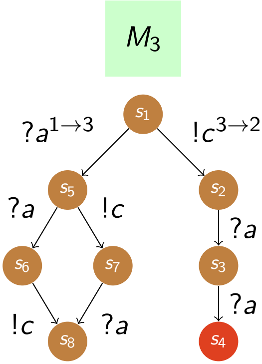


M_2 can receive b and c in any order

Adapting the counter-example to mailboxes

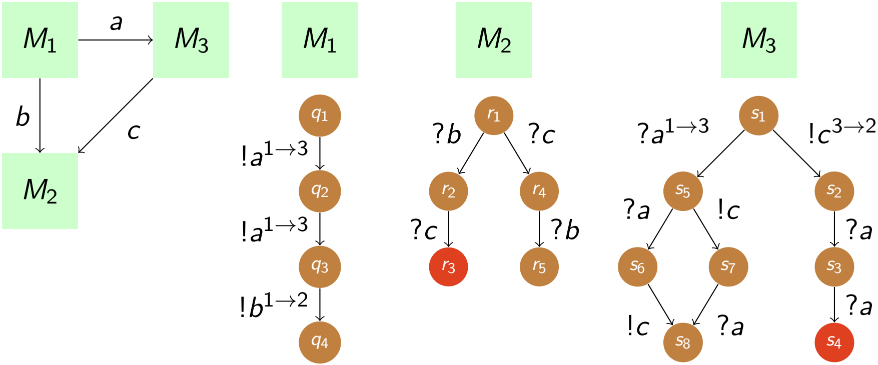




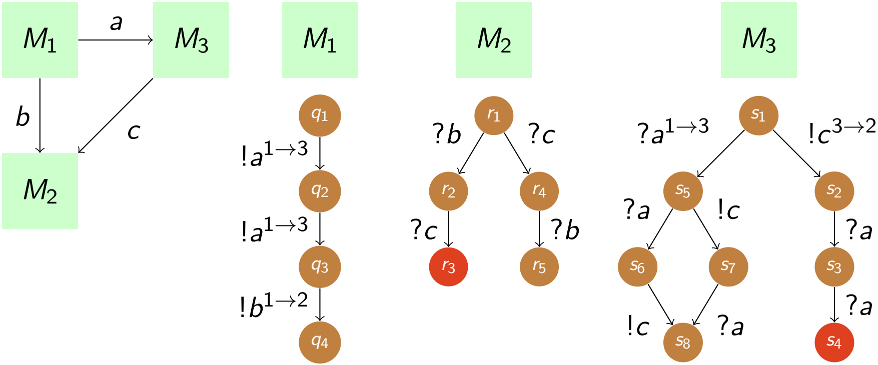


let's dig into that...

Adapting the counter-example to mailboxes

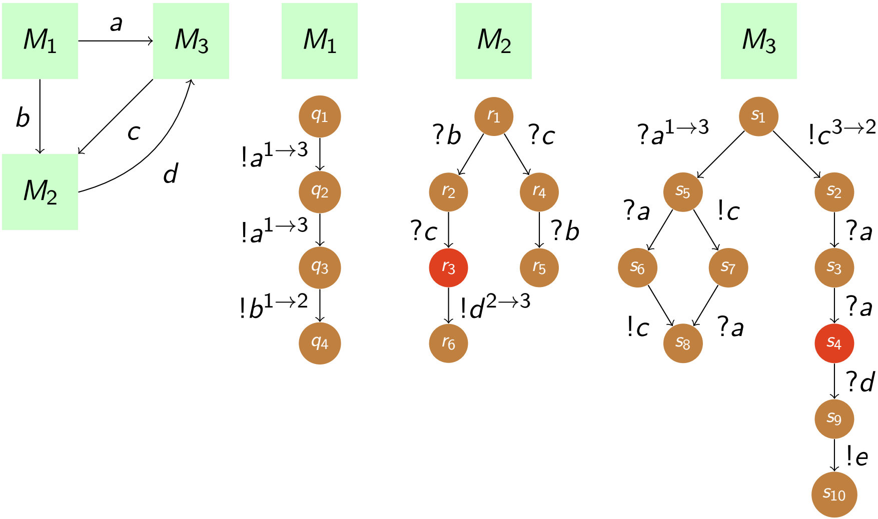


Adapting the counter-example to mailboxes

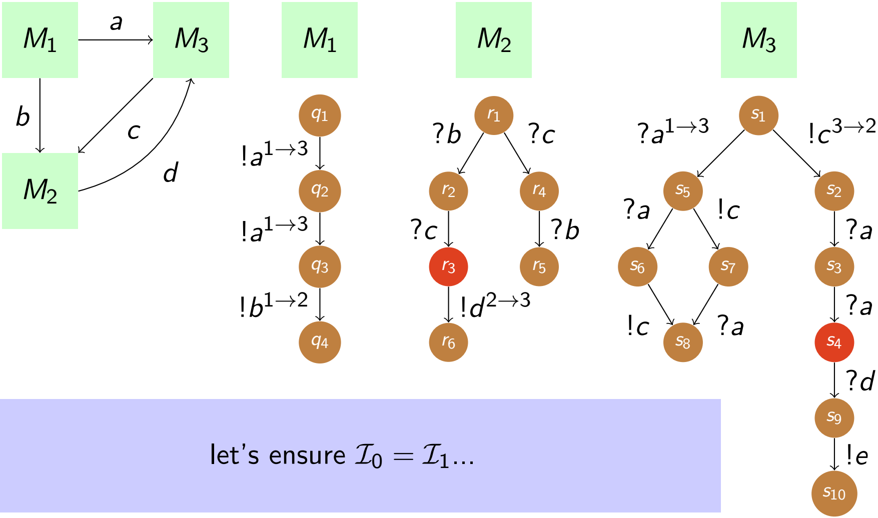


s_4 and r_3 are visited **in a same run** $\Leftrightarrow$ buffer size ≥ 2

Adapting the counter-example to mailboxes

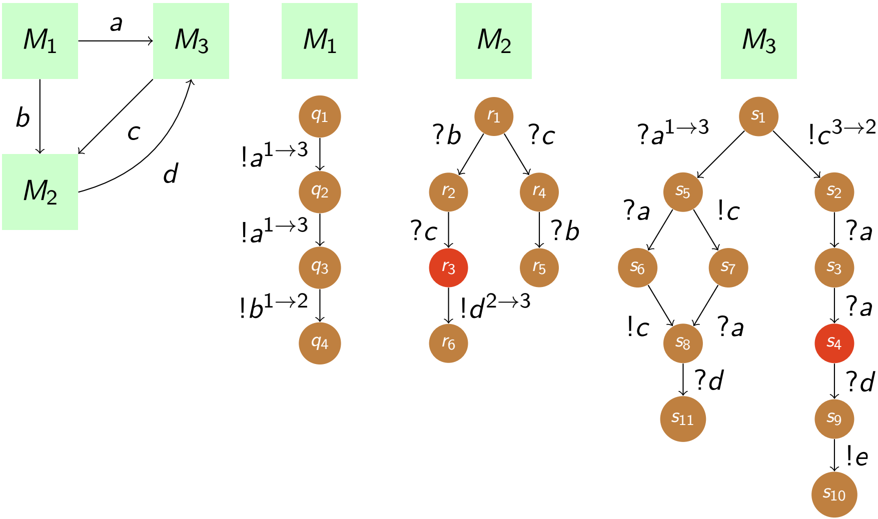


Adapting the counter-example to mailboxes

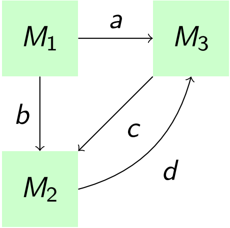


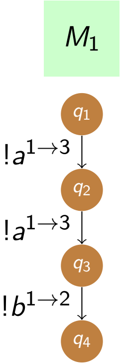
let's ensure $\mathcal{I}_0 = \mathcal{I}_1$...

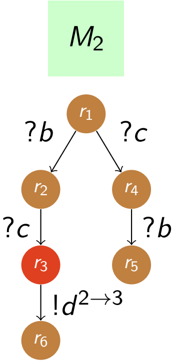
Adapting the counter-example to mailboxes

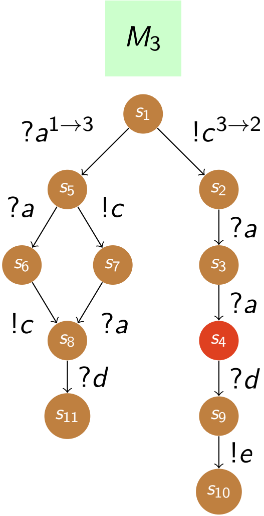


Adapting the counter-example to mailboxes









$\mathcal{I}_0 = \mathcal{I}_1 \neq \mathcal{I}_2$. Done!

Our main results (1)

Synchronizability is undecidable

Whether $\mathcal{I}_0 = \mathcal{I}_\omega$ for a peer-to-peer system is undecidable.

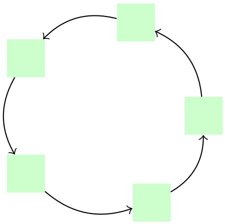
Construction
extension of the first counter-example, reduction from a tiling problem.

In particular
3 machines are needed

Our main results (2)

Oriented rings

- each machine receives from at most one other machine
- each machine sends to at most one other machine
- example: a system with two machines



Synchronizability is decidable for oriented rings

Whether $\mathcal{I}_0 = \mathcal{I}_\omega$ for a system with an oriented ring topology is decidable. Moreover, the set of reachable configurations is channel recognizable.

Open Problems

- what are the topologies for which $\mathcal{I}_0 = \mathcal{I}_\omega$ is decidable?
- is synchronizability for mailboxes *really* decidable?
- what would be a better definition of synchronizability?
  - for peer-to-peer, existentially 0-bounded seems promising [Genest et al]
  - what about mailboxes?