Synchronisability and Communicating Session Automata

Amrita Suresh

University of Oxford

(Joint work with Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Etienne Lozes, and Nobuko Yoshida)

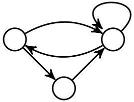
STARDUST Meeting
Glasgow 2023

Communicating Automata

Distributed processes

- each process is a finite state machine

P_i :

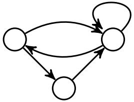


Communicating Automata

Distributed processes

- each process is a finite state machine
- fixed number

P_i :



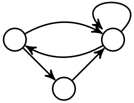
$P_1, \ldots, P_n$

Communicating Automata

Distributed processes

- each process is a finite state machine

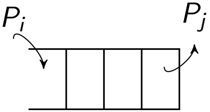
P_i :



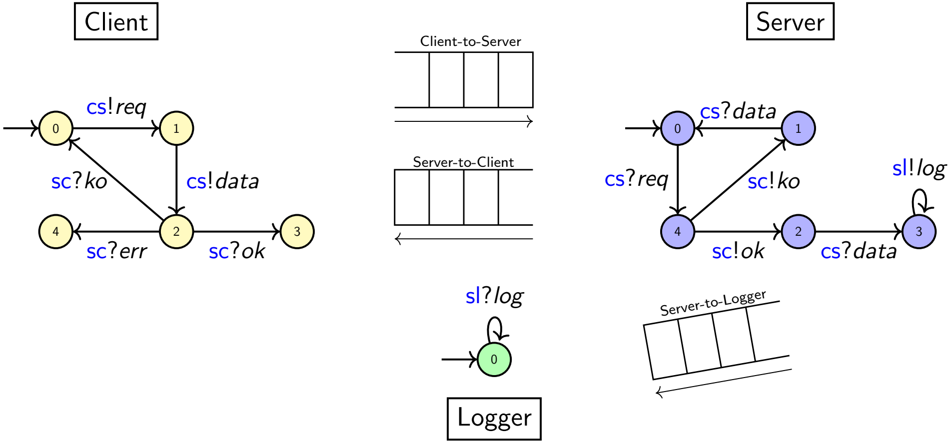
- fixed number

$P_1, \ldots, P_n$

- communicate using queues (perfect, peer-to-peer)

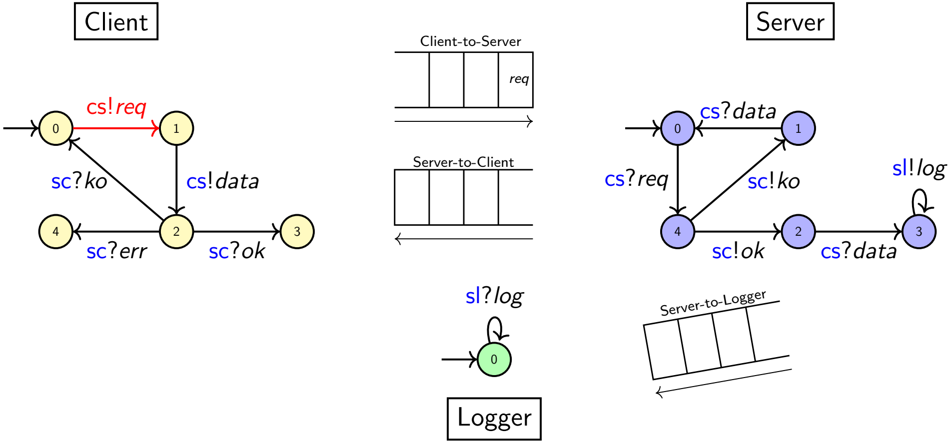


Client-Server-Logger protocol [1]



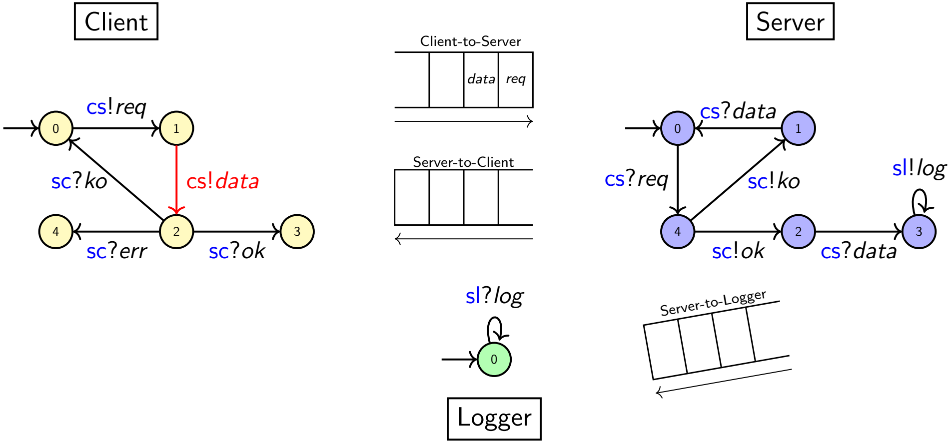
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



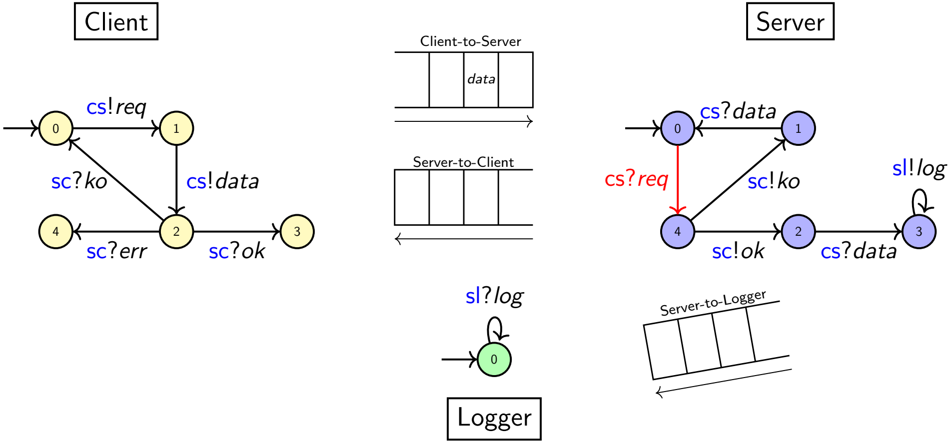
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



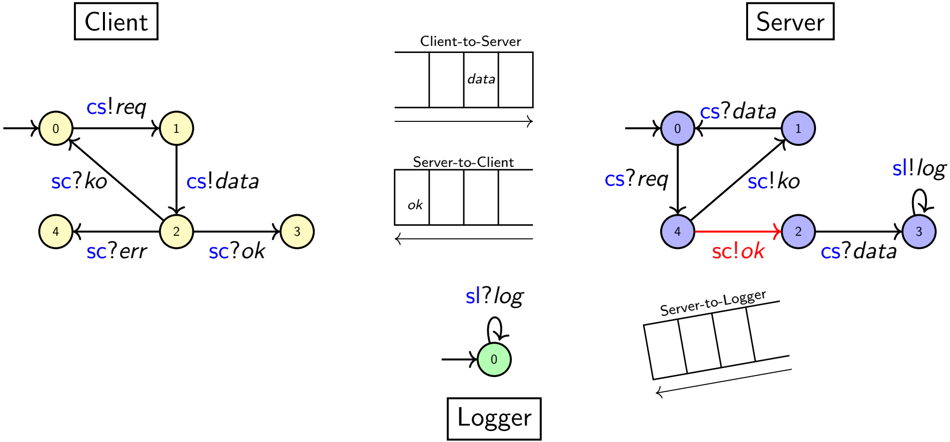
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



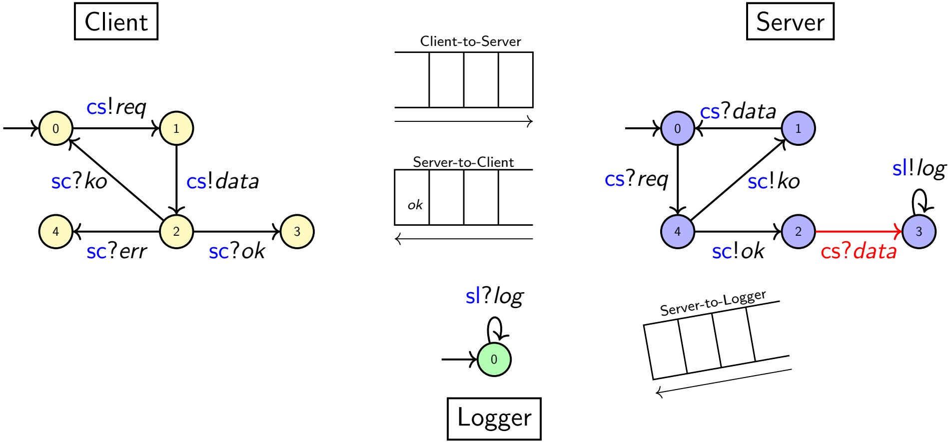
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



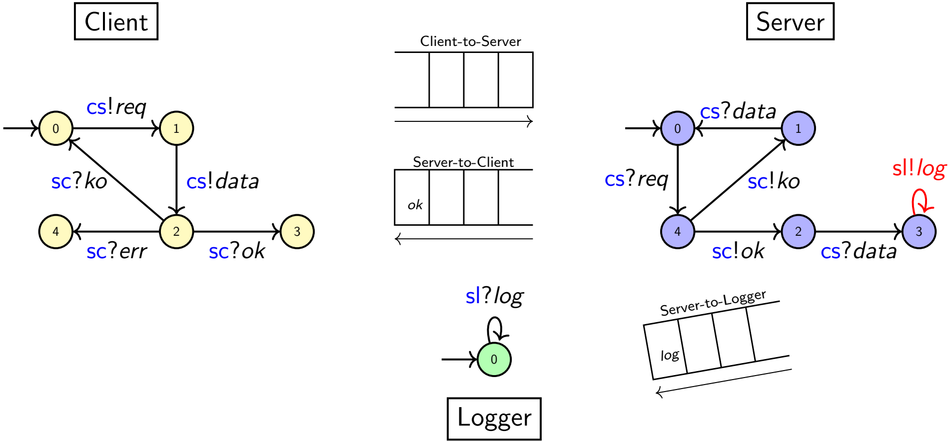
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



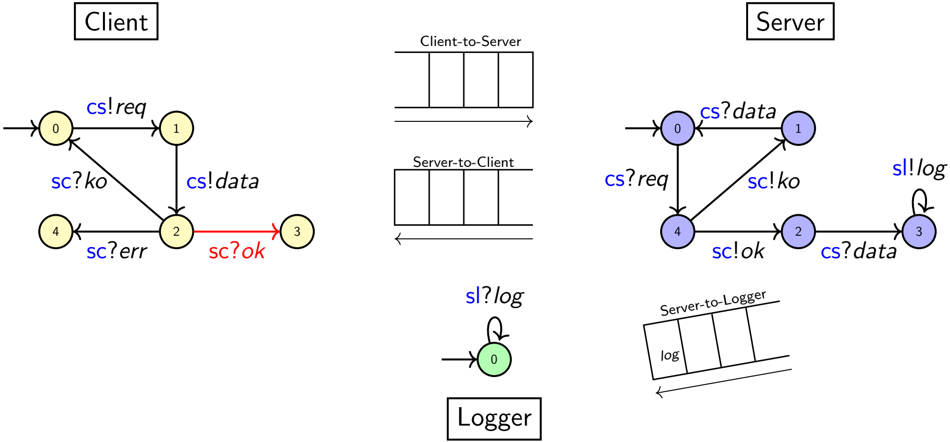
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



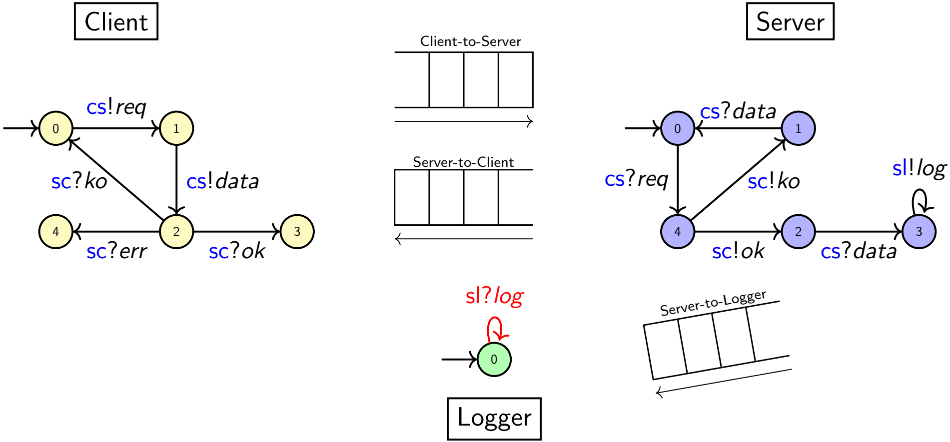
[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Client-Server-Logger protocol [1]



[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

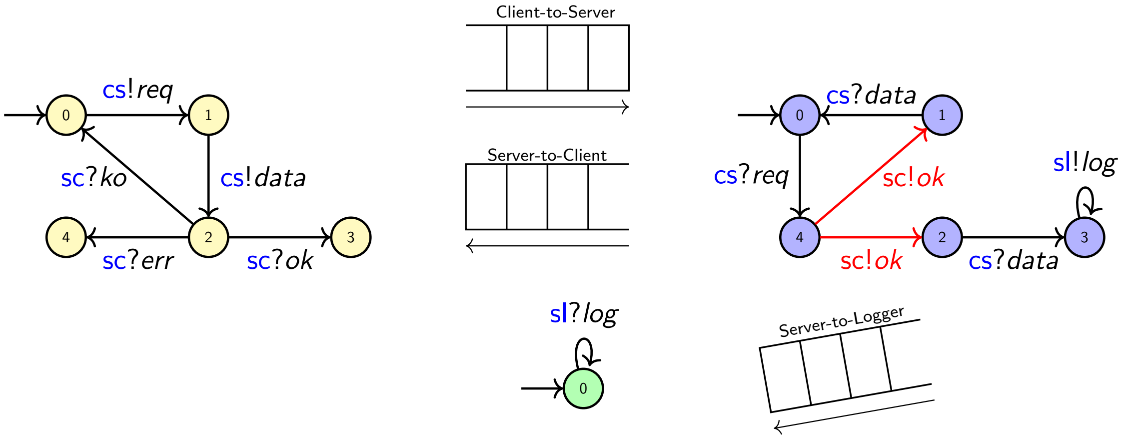
Client-Server-Logger protocol [1]



[1] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

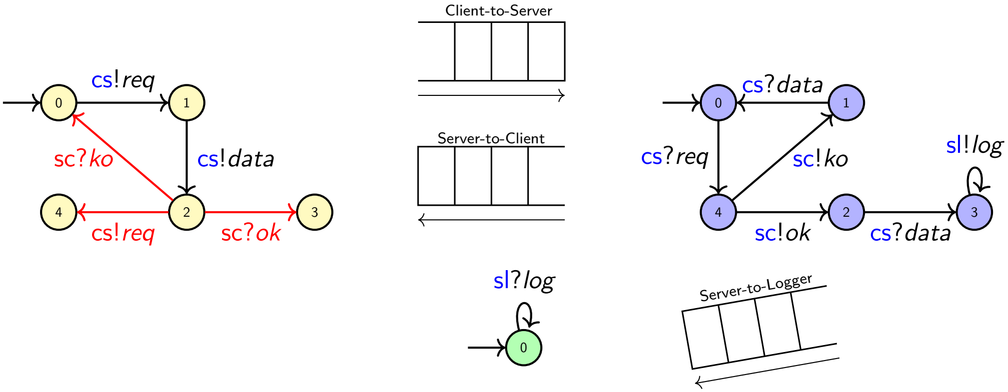
Communicating *Session* Automata

- Deterministic



Communicating *Session* Automata

- Deterministic
- No mixed states



Boundedness

Boundedness Problem

Is there a bound on the size of the queues for all runs?

Boundedness

Boundedness Problem

Is there a bound on the size of the queues for all runs?

UNDECIDABLE for general communicating automata [2]

[2] Brand and Zafiropulo, *On communicating finite-state machines*, 1983

Boundedness

Underapproximations

- Restrict to k-bounded channels.

Boundedness

Underapproximations

- Restrict to k-bounded channels. Too restricting!

Boundedness

Underapproximations

- Restrict to k-bounded channels. Too restricting!
- Every unbounded execution is equivalent to a bounded execution.

Message Sequence Charts

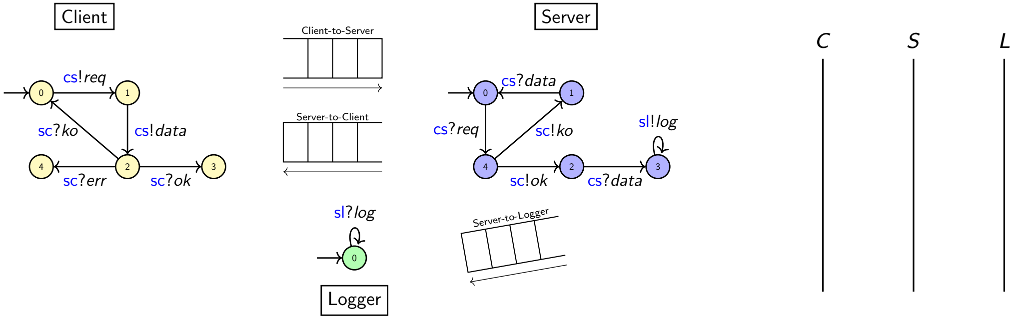
- A graphical way to represent executions

Message Sequence Charts

- A graphical way to represent executions
- Causally independent actions can be rescheduled

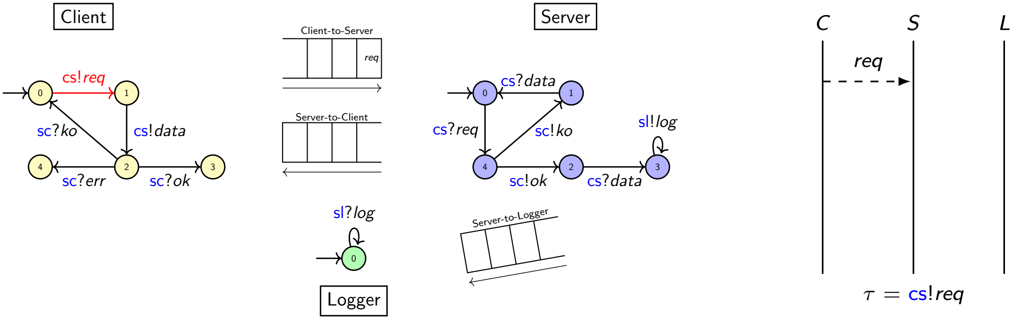
Message Sequence Charts

- A graphical way to represent executions
- Causally independent actions can be rescheduled



Message Sequence Charts

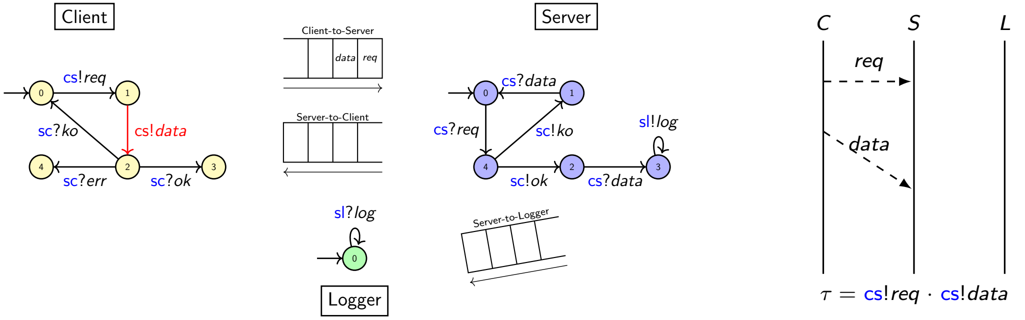
- A graphical way to represent executions
- Causally independent actions can be rescheduled



$\tau = \text{cs}!req$

Message Sequence Charts

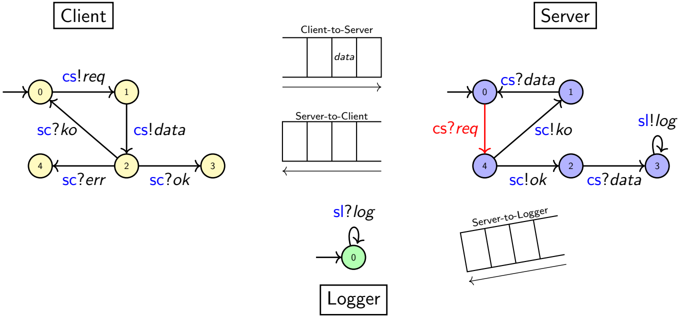
- A graphical way to represent executions
- Causally independent actions can be rescheduled

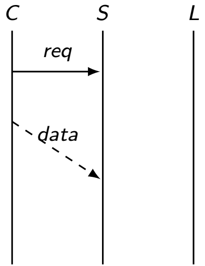


$\tau = \text{cs}!req \cdot \text{cs}!data$

Message Sequence Charts

- A graphical way to represent executions
- Causally independent actions can be rescheduled

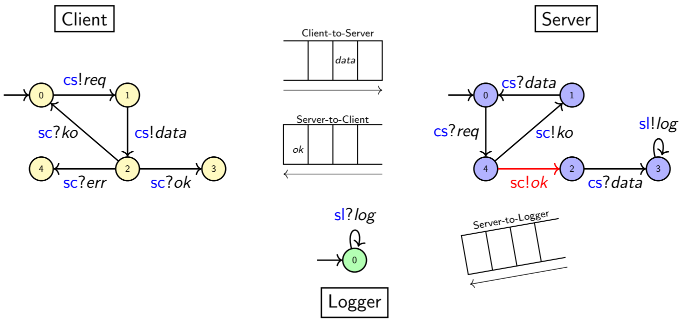


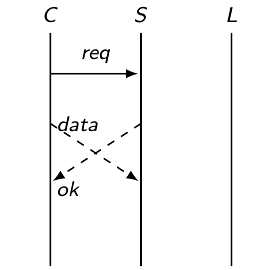


$\tau = \text{cs}!req \cdot \text{cs}!data \cdot \text{cs}?req$

Message Sequence Charts

- A graphical way to represent executions
- Causally independent actions can be rescheduled

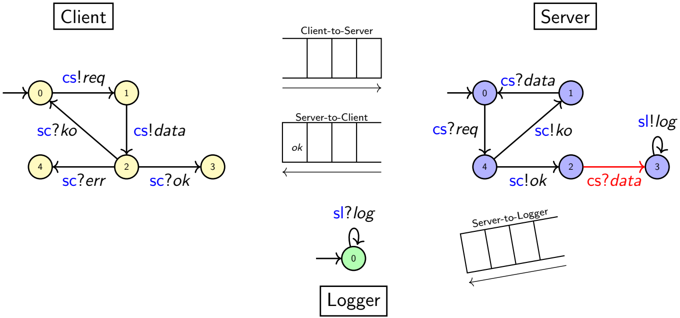


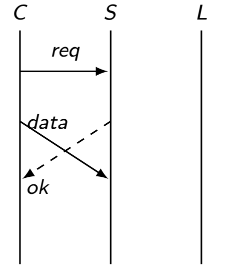


$\tau = \text{cs}!req \cdot \text{cs}!data \cdot \text{cs}?req \cdot \text{sc}!ok$

Message Sequence Charts

- A graphical way to represent executions
- Causally independent actions can be rescheduled

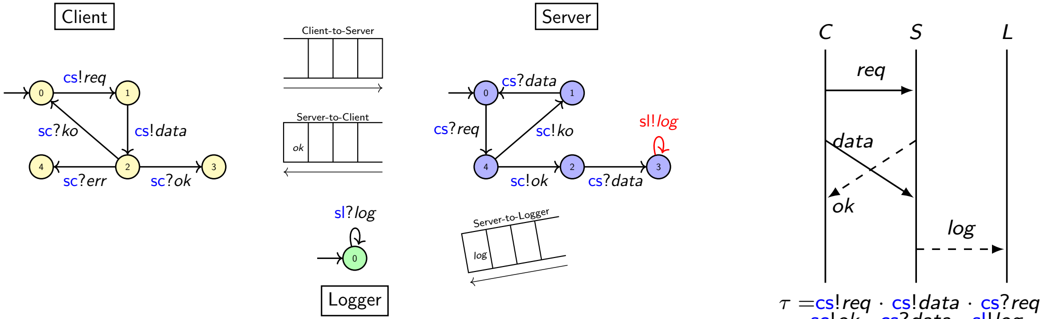




$\tau = \text{cs}!req \cdot \text{cs}!data \cdot \text{cs}?req \cdot \text{sc}!ok \cdot \text{cs}?data$

Message Sequence Charts

- A graphical way to represent executions
- Causally independent actions can be rescheduled



$\tau =$ cs!*req* · cs!*data* · cs?*req* · sc!*ok* · cs?*data* · sl!*log*

Synchronisability

- *existentially k-bounded* systems [2] [3] - all accepting executions re-ordered to a k-bounded execution.

[2] Lohrey and Muscholl, *Bounded MSC communication*, 2002

[3] Genest et al., *A Kleene theorem for a class of communicating automata with effective algorithms*, 2004

Synchronisability

- *existentially k-bounded* systems [2] [3]
- *synchronisable systems* [4] - send projection equivalent to rendezvous.

---

[2] Lohrey and Muscholl, *Bounded MSC communication*, 2002
[3] Genest et al., *A Kleene theorem for a class of communicating automata with effective algorithms*, 2004
[4] Basu and Bultan, *Choreography conformance via synchronisability*, 2011

Synchronisability

- *existentially k-bounded* systems [2] [3]
- *synchronisable systems* [4]
- *k-synchronisable systems* [5] - if every MSC admits a linearisation that can be divided into "blocks" of at most k messages.

---

[2] Lohrey and Muscholl, *Bounded MSC communication*, 2002

[3] Genest et al., *A Kleene theorem for a class of communicating automata with effective algorithms*, 2004

[4] Basu and Bultan, *Choreography conformance via synchronisability*, 2011

[5] Bouajjani et al., *On the completeness of verifying message passing programs under bounded asynchrony*, 2018

Synchronisability

- *existentially k-bounded* systems [2] [3]
- *synchronisable systems* [4]
- *k-synchronisable systems* [5]
- *k-exhaustive systems* [6] - whenever a send action is enabled, it can be fired within a k-bounded execution

[2] Lohrey and Muscholl, *Bounded MSC communication*, 2002
[3] Genest et al., *A Kleene theorem for a class of communicating automata with effective algorithms*, 2004
[4] Basu and Bultan, *Choreography conformance via synchronisability*, 2011
[5] Bouajjani et al., *On the completeness of verifying message passing programs under bounded asynchrony*, 2018
[6] Lange and Yoshida, *Verifying Asynchronous Interactions via Communicating Session Automata*, 2019

Weakly k-synchronous MSCs

A k-exchange is an MSC that allows one to schedule all sends before all receives, and there are at most k sends.

Weakly k-synchronous MSCs

A k-exchange is an MSC that allows one to schedule all sends before all receives, and there are at most k sends.

Definition

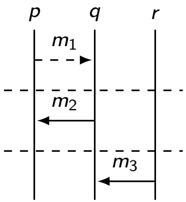
M is weakly k-synchronous if it is of the form $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is a k-exchange.

Weakly k-synchronous MSCs

A k-exchange is an MSC that allows one to schedule all sends before all receives, and there are at most k sends.

Definition

M is weakly k-synchronous if it is of the form $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is a k-exchange.



MSO definability

Condition 1

The set of MSCs are MSO-definable.

MSO definability

First-order variables

MSO Logic

- $x \to y$ — x precedes y in the process order
- $x \lhd y$ — x and y are matched send-receive events
- $\lambda(x) = a$ — x has the label a
- $x = y$ — Second-order variable
- $\exists x.\phi$ — there is an event x such that ϕ
- $\exists X.\phi$ — there is a unary relation X such that ϕ holds
- $\phi \vee \phi$, $\neg\phi$, $x \in X$, etc.

MSO definability

First-order variables

MSO Logic

- $x \rightarrow y$ — x precedes y in the process order
- $x \lhd y$ — x and y are matched send-receive events
- $\lambda(x) = a$ — x has the label a
- $x = y$
- $\exists x.\phi$ — there is an event x such that ϕ
- $\exists X.\phi$ — there is a unary relation X such that ϕ holds
- $\phi \vee \phi$, $\neg\phi$, $x \in X$, etc.

Second-order variable

$matched(x) = \exists y. x \lhd y$ indicates that x is a matched send.

Special tree width

Condition 2

The set of MSCs have bounded special tree-width.

Special tree width

Condition 2

The set of MSCs have bounded special tree-width.

- Adam-Eve play the *decomposition game*.

Special tree width

Condition 2

The set of MSCs have bounded special tree-width.

- Adam-Eve play the *decomposition game*.
- Eve "colours" some events on the MSC, removes edges between coloured events.

Special tree width

Condition 2

The set of MSCs have bounded special tree-width.

- Adam-Eve play the *decomposition game*.
- Eve "colours" some events on the MSC, removes edges between coloured events.
- Adam chooses one of the resulting connected components.

Special tree width

Condition 2

The set of MSCs have bounded special tree-width.

- Adam-Eve play the *decomposition game*.
- Eve "colours" some events on the MSC, removes edges between coloured events.
- Adam chooses one of the resulting connected components.
- Bounded special tree-width k if Eve can win (colour all vertices) with $k+1$ colours.

Crucial observation

Theorem

Let $\mathcal{C}$ be a class of MSCs. If $\mathcal{C}$ is MSO-definable and STW-bounded class, the following problem is decidable: Given a communicating system S, do we have $L(S) \subseteq \mathcal{C}$?

Crucial observation

Theorem

Let $\mathcal{C}$ be a class of MSCs. If $\mathcal{C}$ is MSO-definable and STW-bounded class, the following problem is decidable: Given a communicating system S, do we have $L(S) \subseteq \mathcal{C}$?

- Synchronisability for an STW-bounded class $\xrightarrow{\text{reduces to}}$ *bounded model-checking*
- Bounded model-checking $\rightarrow$ known to be decidable [7]

[7] Bollig and Gastin, *Non-sequential theory of distributed systems*, 2019

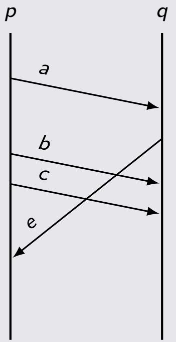
Applying the framework to k-weakly synchronous MSCs

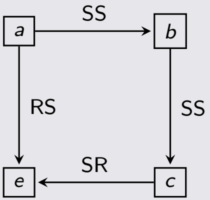
Result

The set of k-weakly synchronous MSCs are MSO-definable.

Applying the framework to k-weakly synchronous MSCs

Conflict graph





Applying the framework to k-weakly synchronous MSCs

Result

The set of weakly synchronous MSCs are MSO-definable.

Graphical characterisation of weakly synchronous MSCs

- No RS edge along any cycle
- At most k vertices in any SCC

MSO definable!

Applying the framework to k-weakly synchronous MSCs

Result

The set of weakly synchronous MSCs has bounded STW.

- Eve's strategy - isolate each exchange, then remove message pairs
- Uses at most $4n + 1$ colours

What if the channels are not perfect?

We can assume various sources of unreliability in the channels like:

- lossiness - some messages may be lost (while sending or in the channel)

What if the channels are not perfect?

We can assume various sources of unreliability in the channels like:

- lossiness - some messages may be lost (while sending or in the channel)
- corruption - some messages change to others

What if the channels are not perfect?

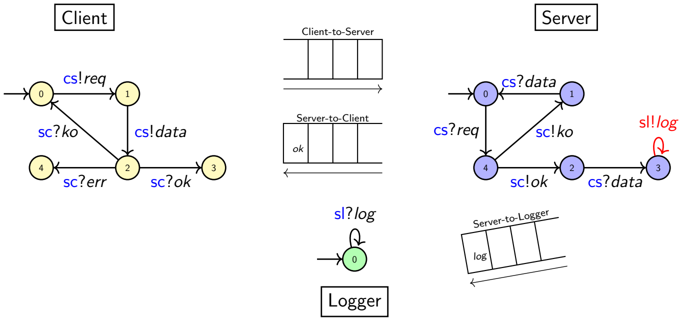
We can assume various sources of unreliability in the channels like:

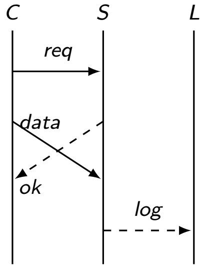
- lossiness - some messages may be lost (while sending or in the channel)
- corruption - some messages change to others
- out-of-order - the FIFO order is no longer maintained

What if the channels are not perfect?

We can assume various sources of unreliability in the channels like:

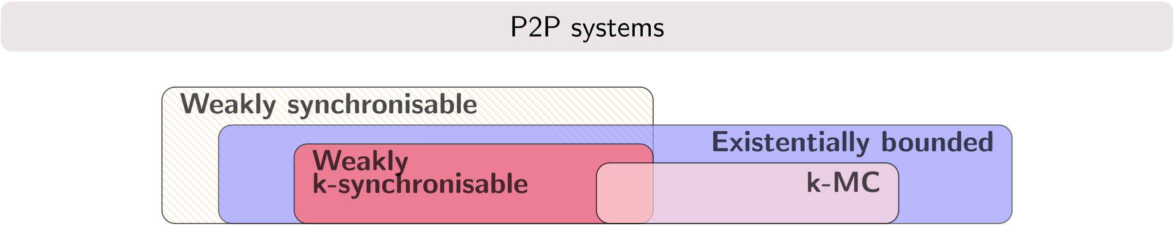
- lossiness - some messages may be lost (while sending or in the channel)
- corruption - some messages change to others
- out-of-order - the FIFO order is no longer maintained





τ = cs!*req* · cs!*data* · cs?*req* · sc!*ok* · cs?*data* · sl!*log*

Comparison of classes



Contributions and Perspectives

- Unifying framework for various notions of synchronisability.[8]
- Applicable to both mailbox and p2p communications.
- LCPDL for better complexity.

[8] Bollig et al. *A Unifying Framework for Deciding Synchronisability*, 2021

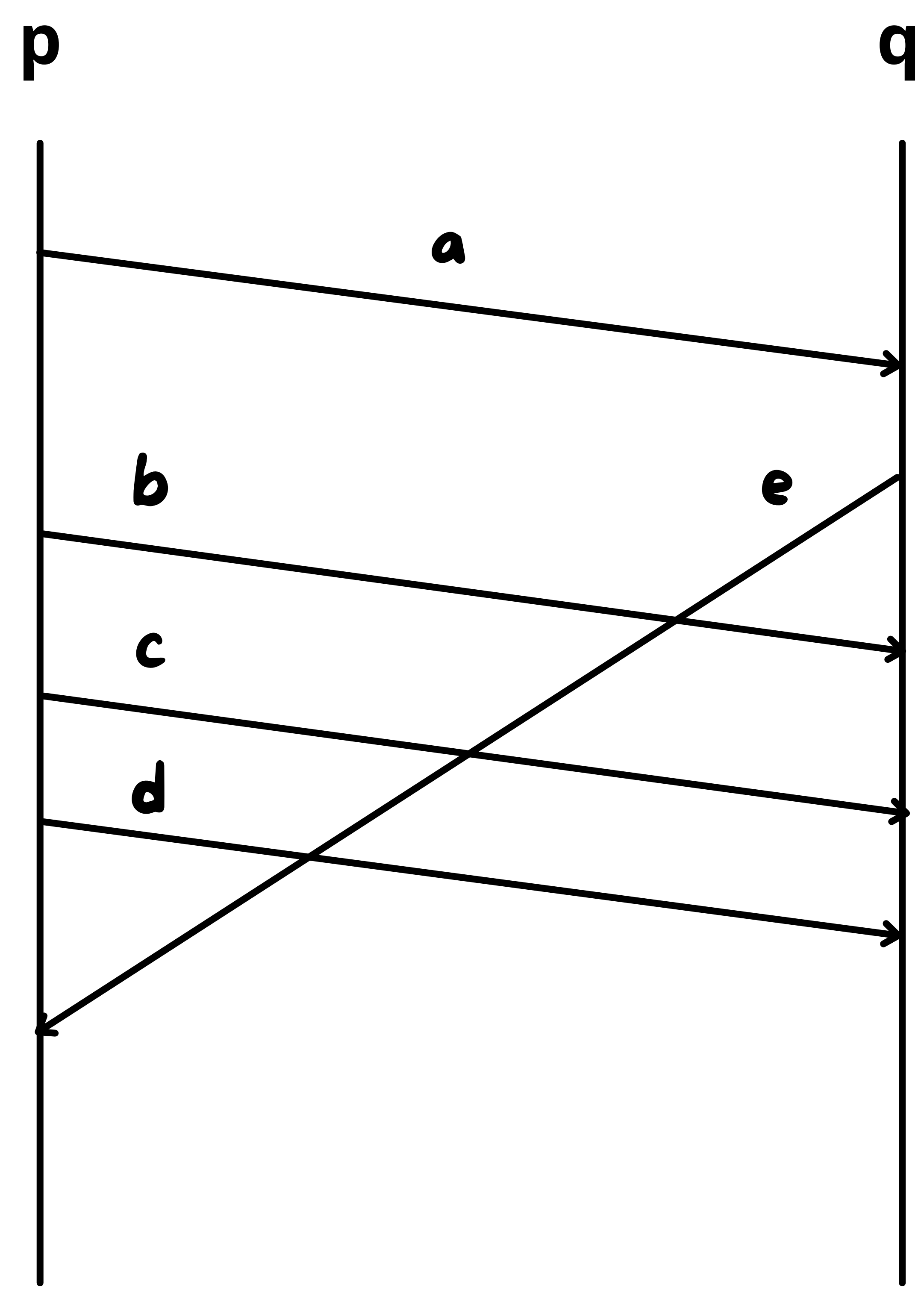
Contributions and Perspectives

- How can we modify these notions to retain their decidability in the presence of errors?
- Given an unreliable automaton, can we modify it to retain membership?
- Can we use ideas like completely specified protocols to always have information during errors?

Thank you!

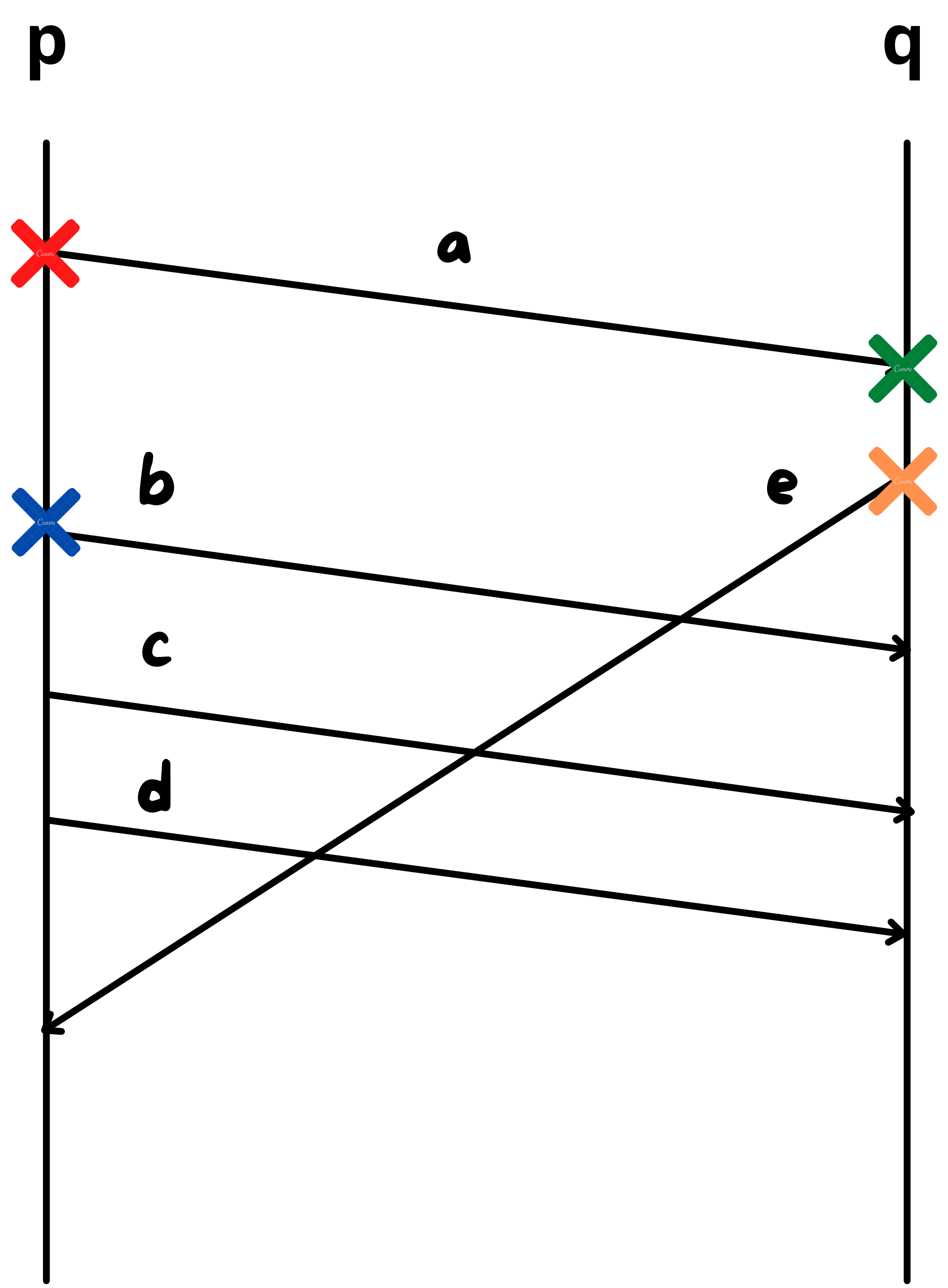
Questions?

Applying the framework to k-weakly synchronous MSCs



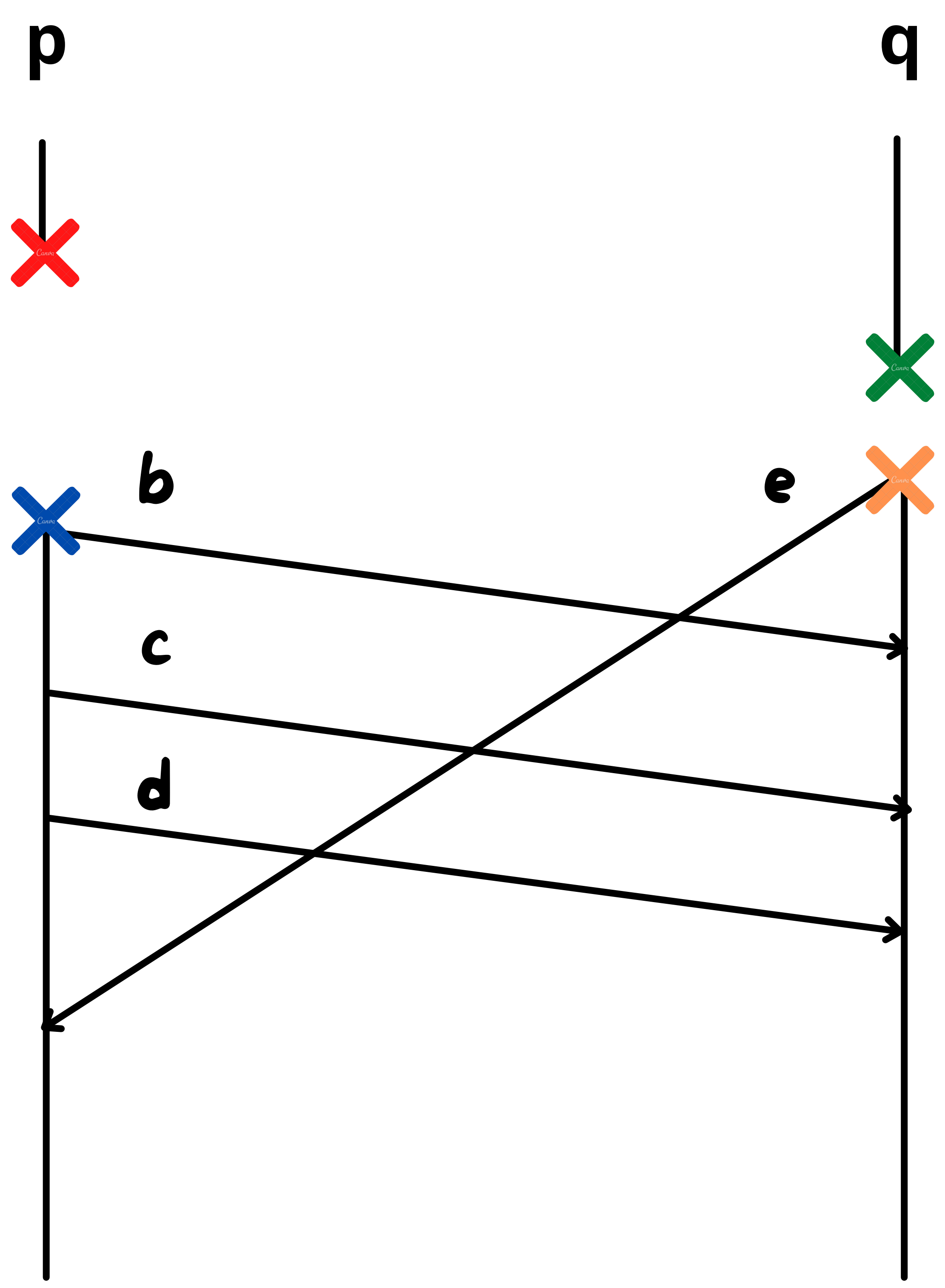
Applying the framework to k-weakly synchronous MSCs

Eve's turn



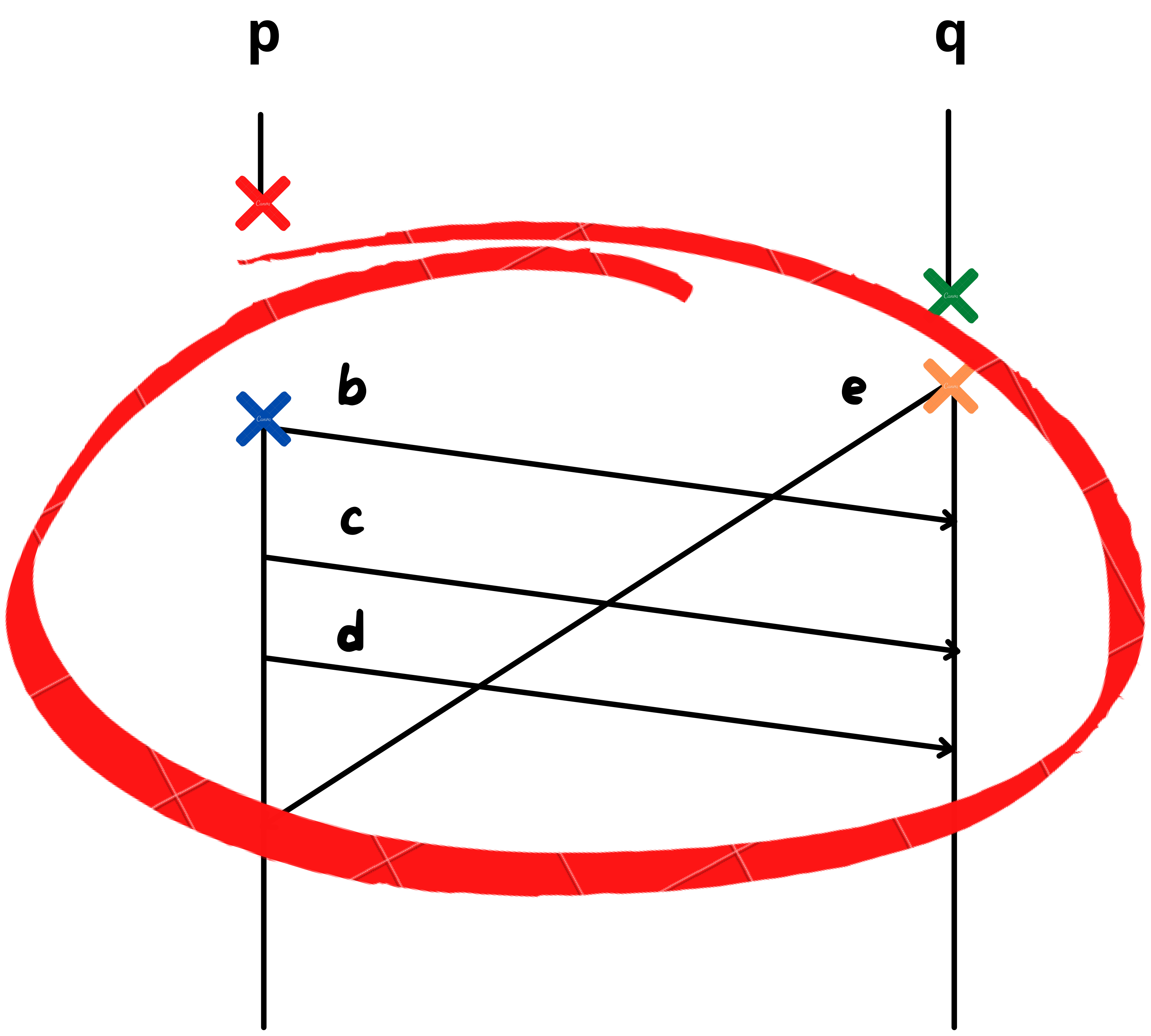
Applying the framework to k-weakly synchronous MSCs

Adam's turn



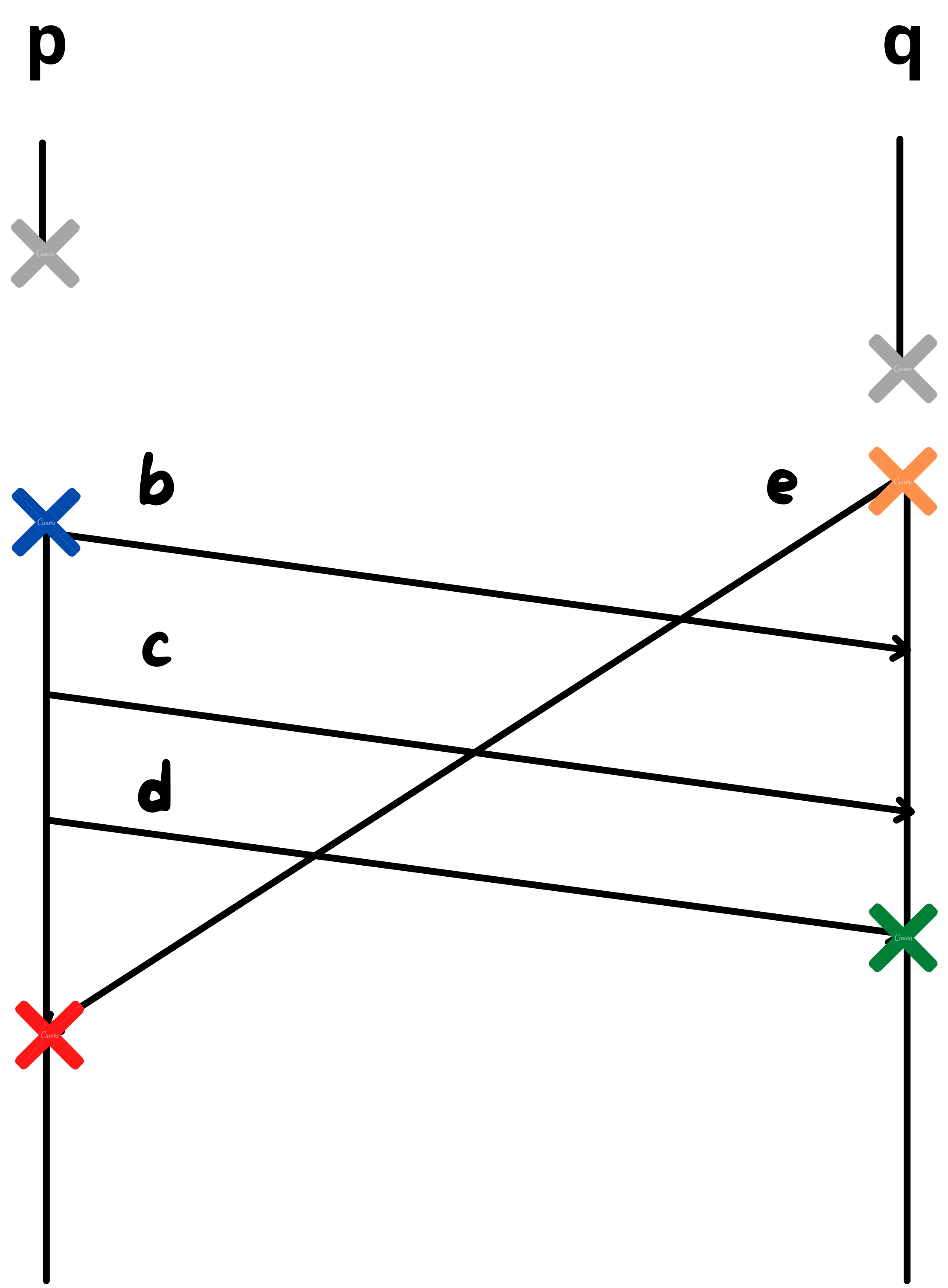
Applying the framework to k-weakly synchronous MSCs

Adam's turn



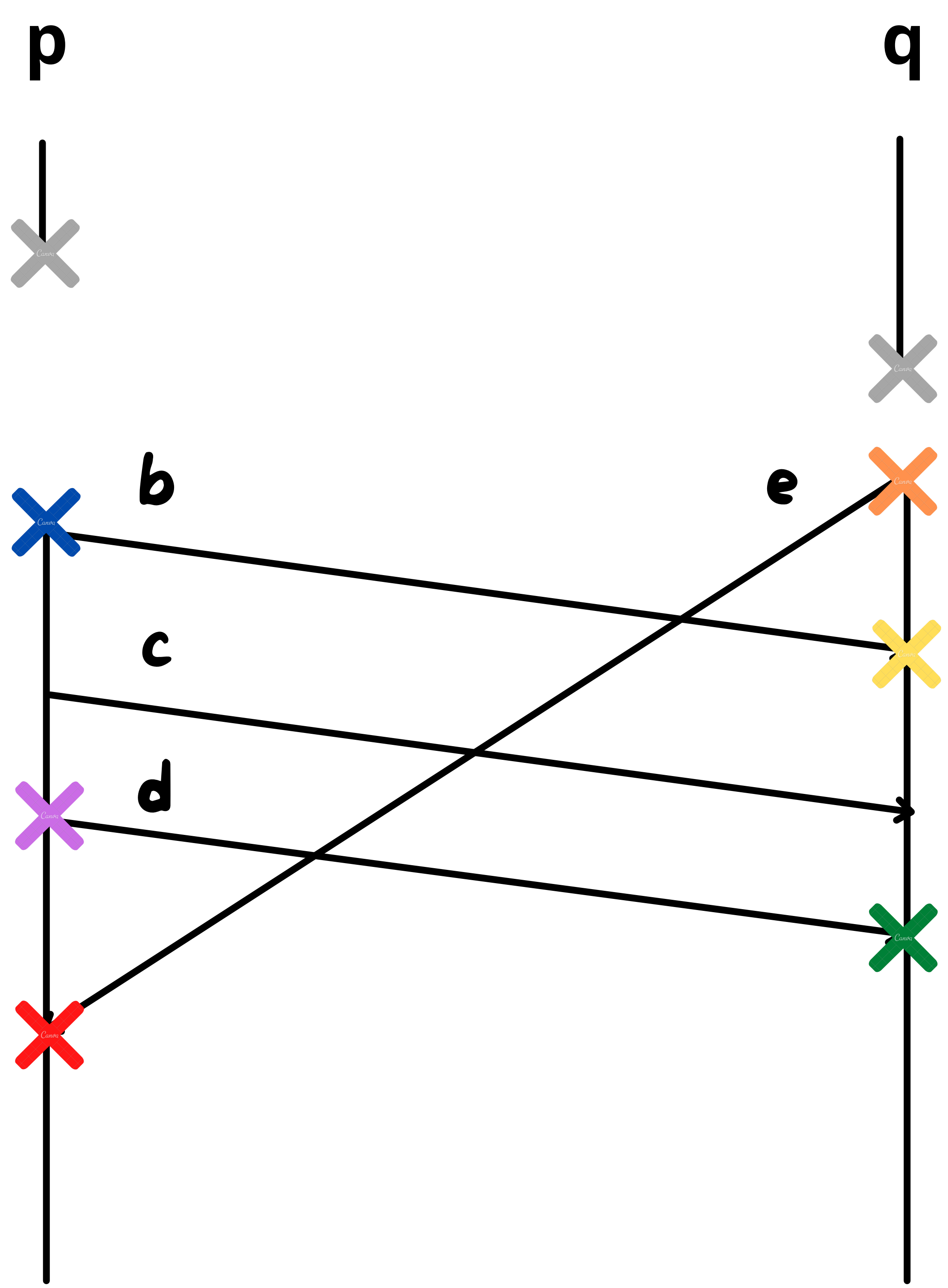
Applying the framework to k-weakly synchronous MSCs

Eve's turn



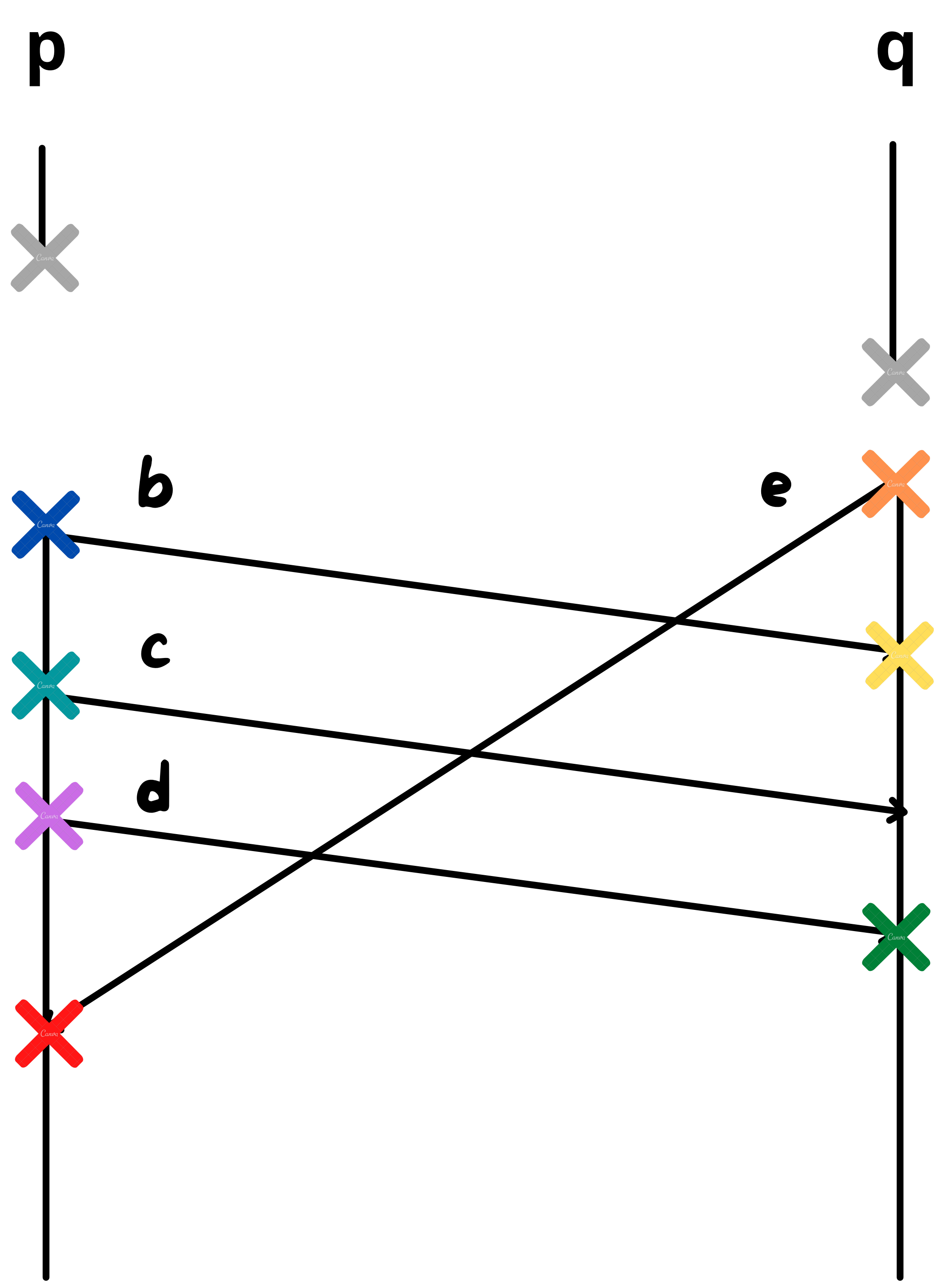
Applying the framework to k-weakly synchronous MSCs

Eve's turn



Applying the framework to k-weakly synchronous MSCs

Eve's turn



Applying the framework to k-weakly synchronous MSCs

Adam's turn