Crime & Punishment in Distributed Algorithms

Pierre Civit (Sorbonne Université)
pierre.civit@lip6.fr

Crime & Punishment in Distributed Algorithms

Pierre Civit (Sorbonne Université)
pierre.civit@lip6.fr

Crime & Punishment in Distributed Algorithms

Pierre Civit
Sorbonne University, CNRS, LIP6

Seth Gilbert
NUS Singapore

Vincent Gramoli
University of Sydney

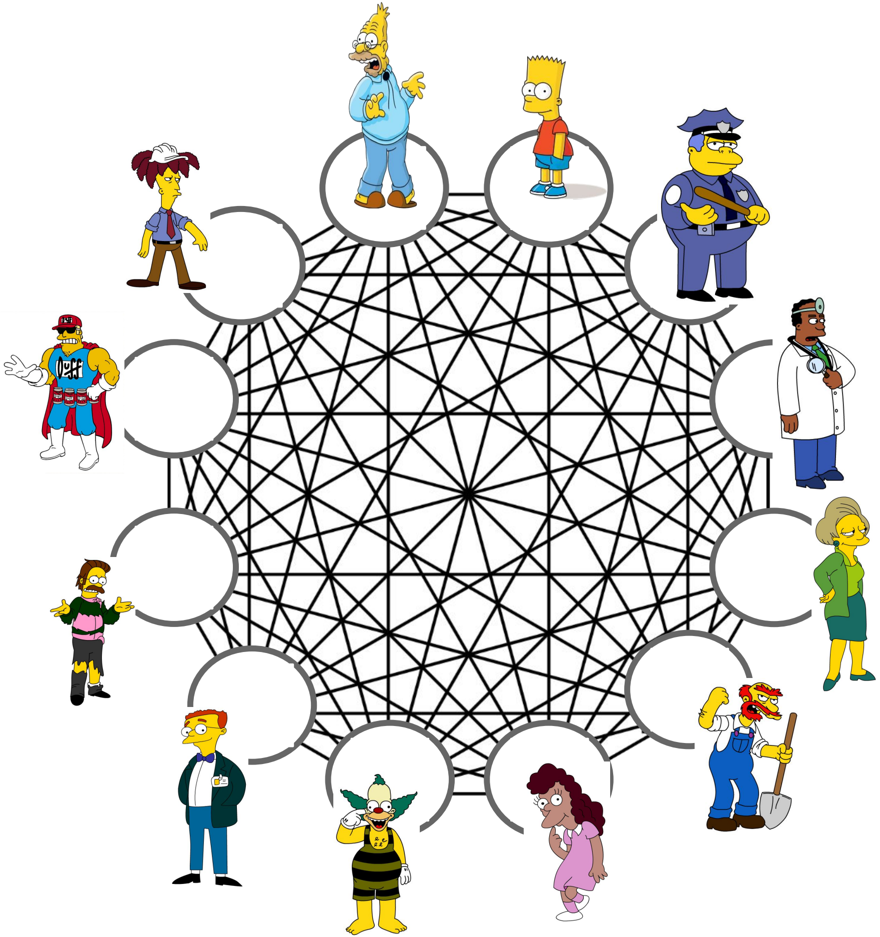
Rachid Guerraoui
Ecole Polytechnique Fédérale de Lausanne (EPFL)

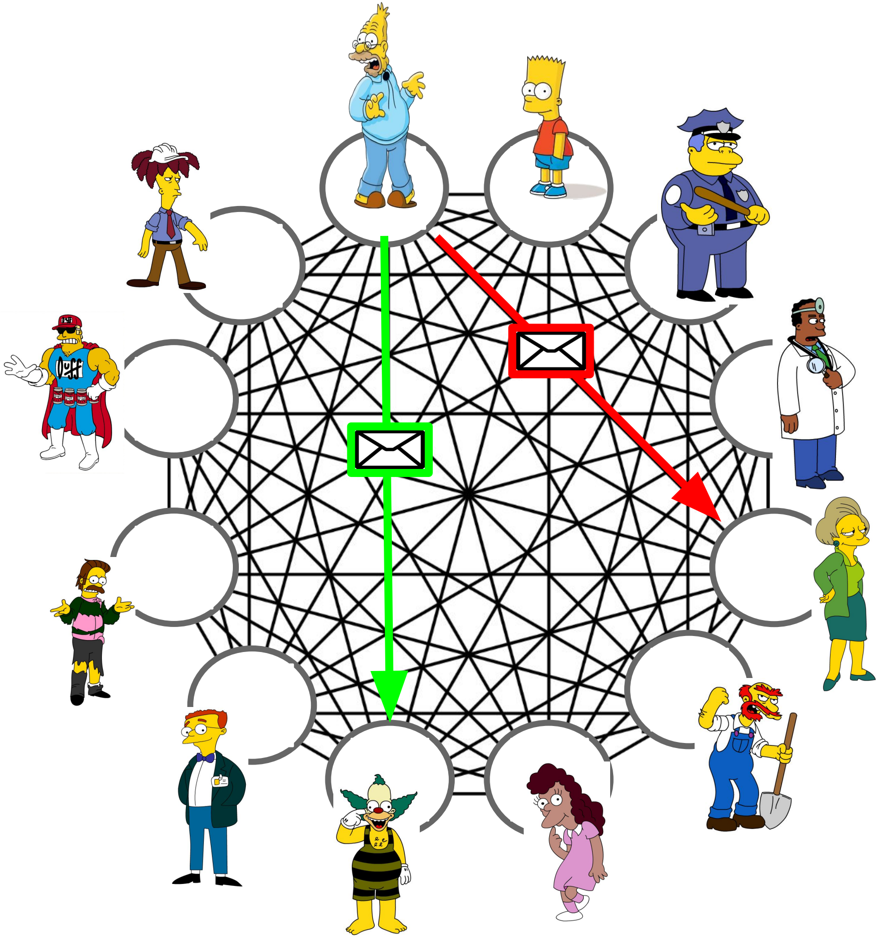
Jovan Komatovic
Ecole Polytechnique Fédérale de Lausanne (EPFL)

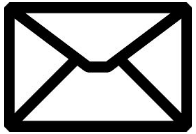
Zarko Milosevic
Informal Systems

Adi Serendinschi
Informal Systems

Motivation

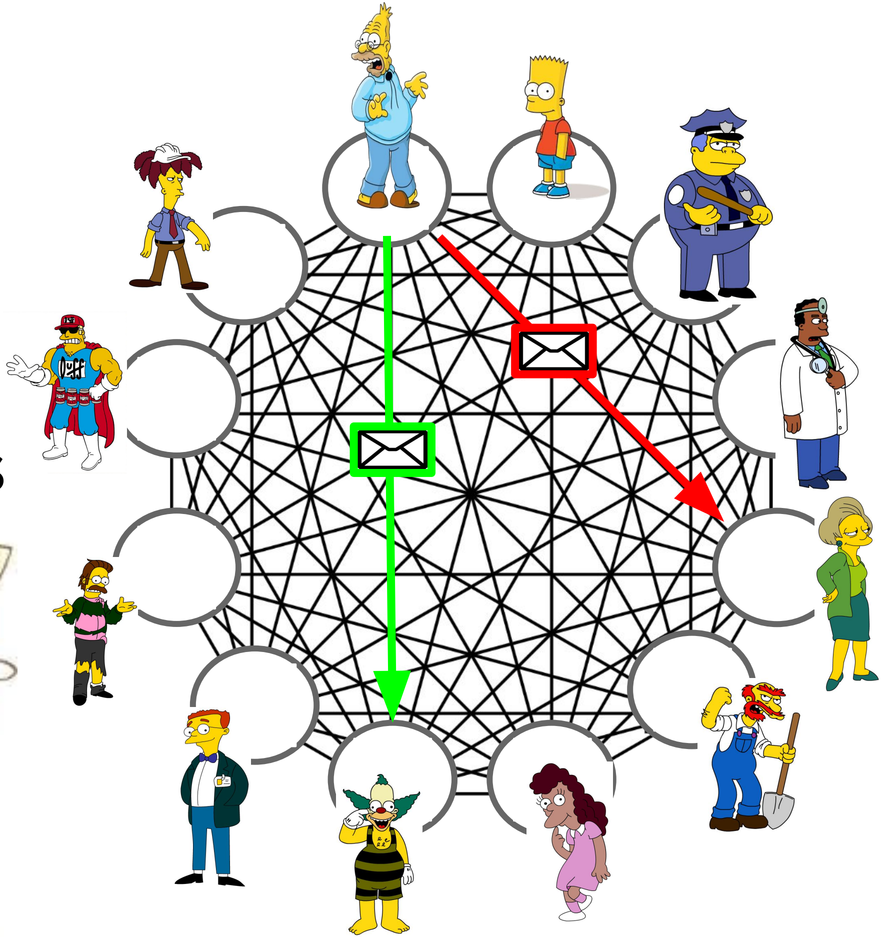






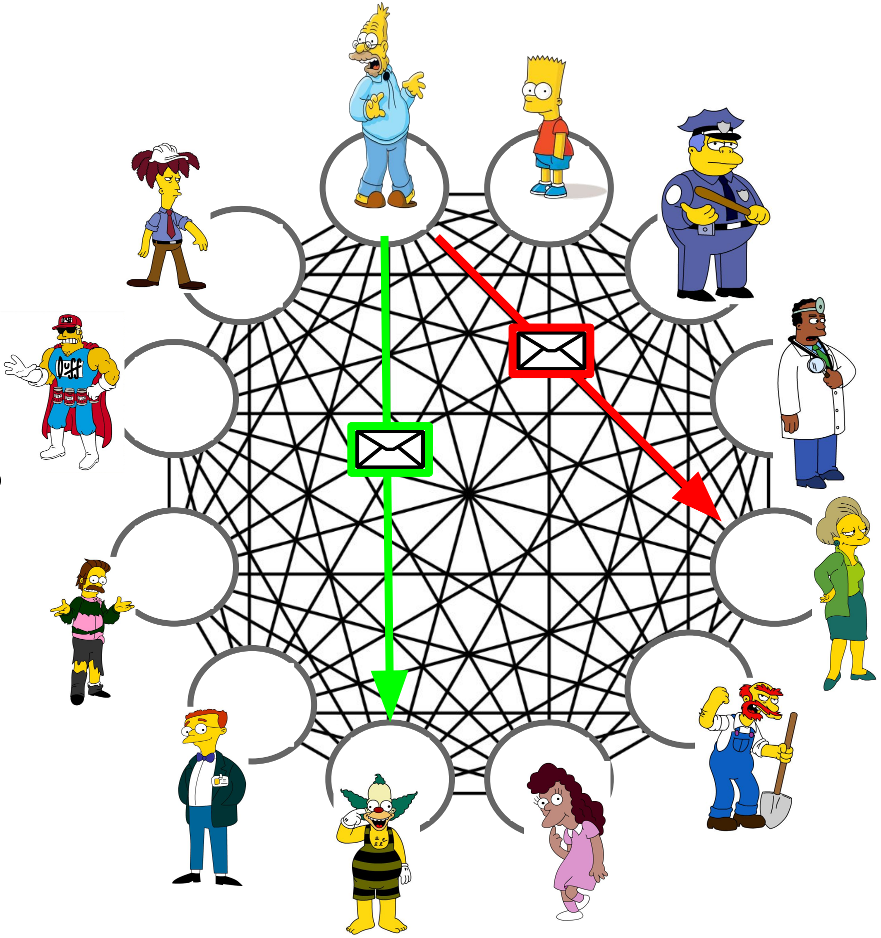
message-passing

message-passing



non synchronous

non
synchronous



message-passing

Authenticated

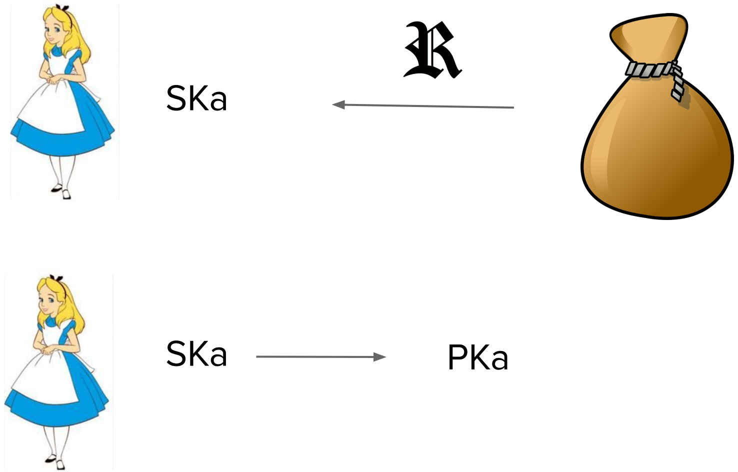
Digital Signature (Public Key Infrastructure)

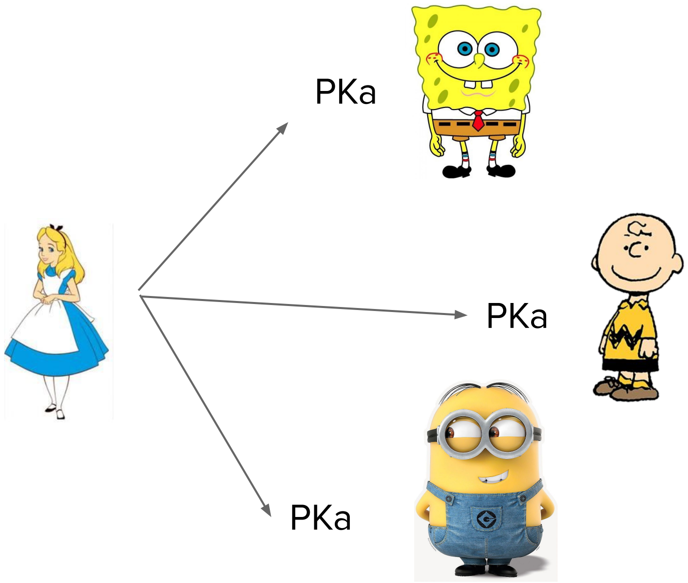
Authentication

Integrity

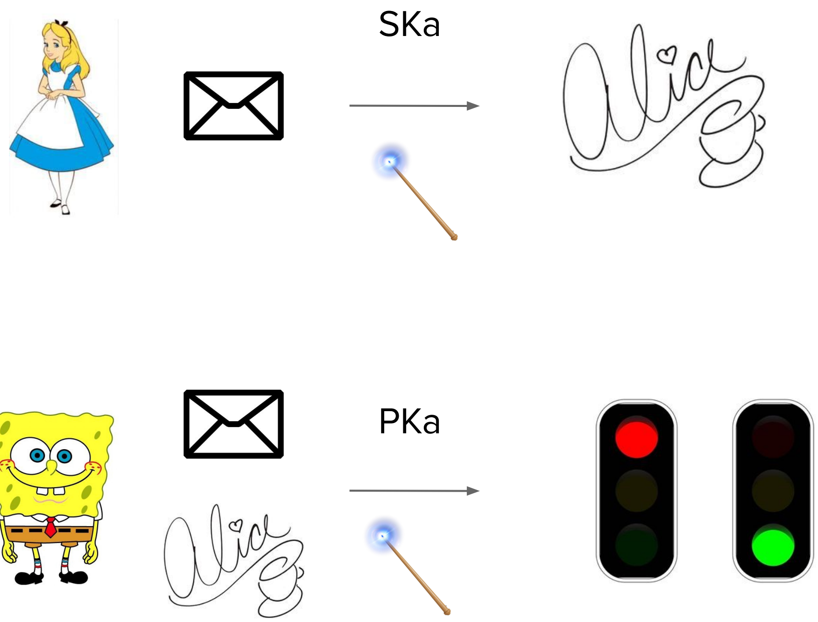
Non-Repudiation

Digital Signature

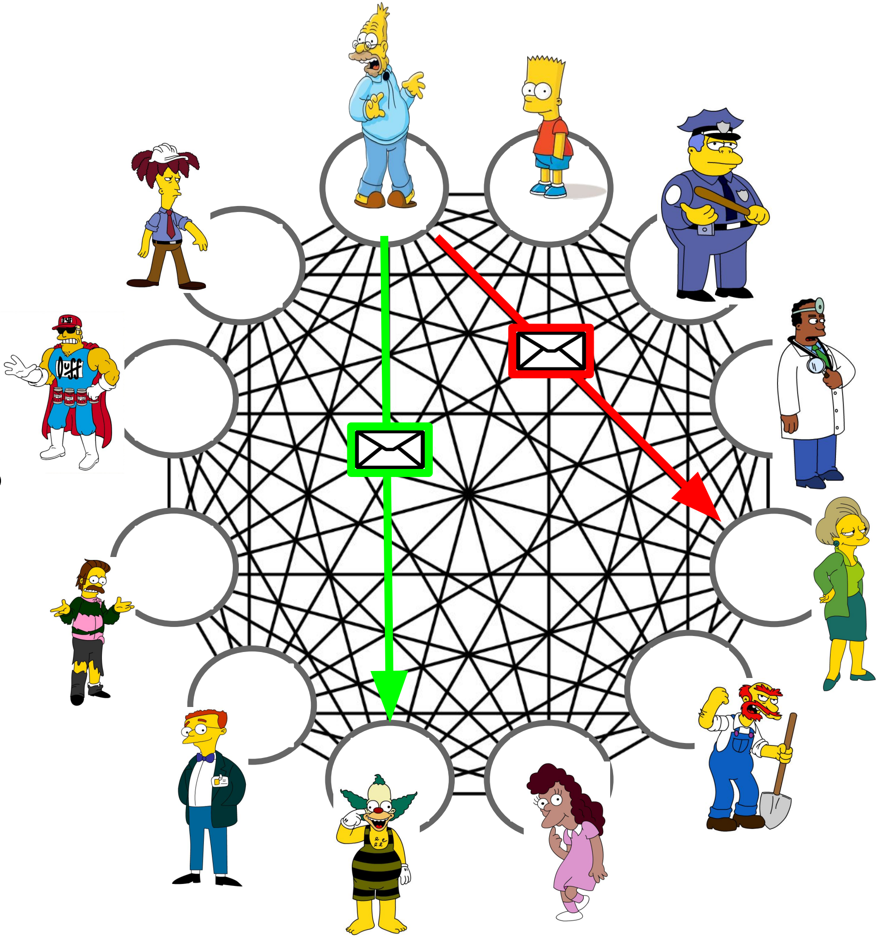




Signature

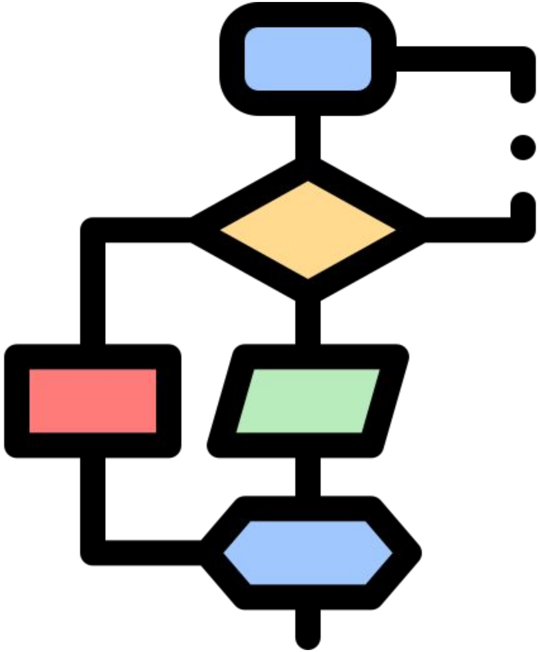


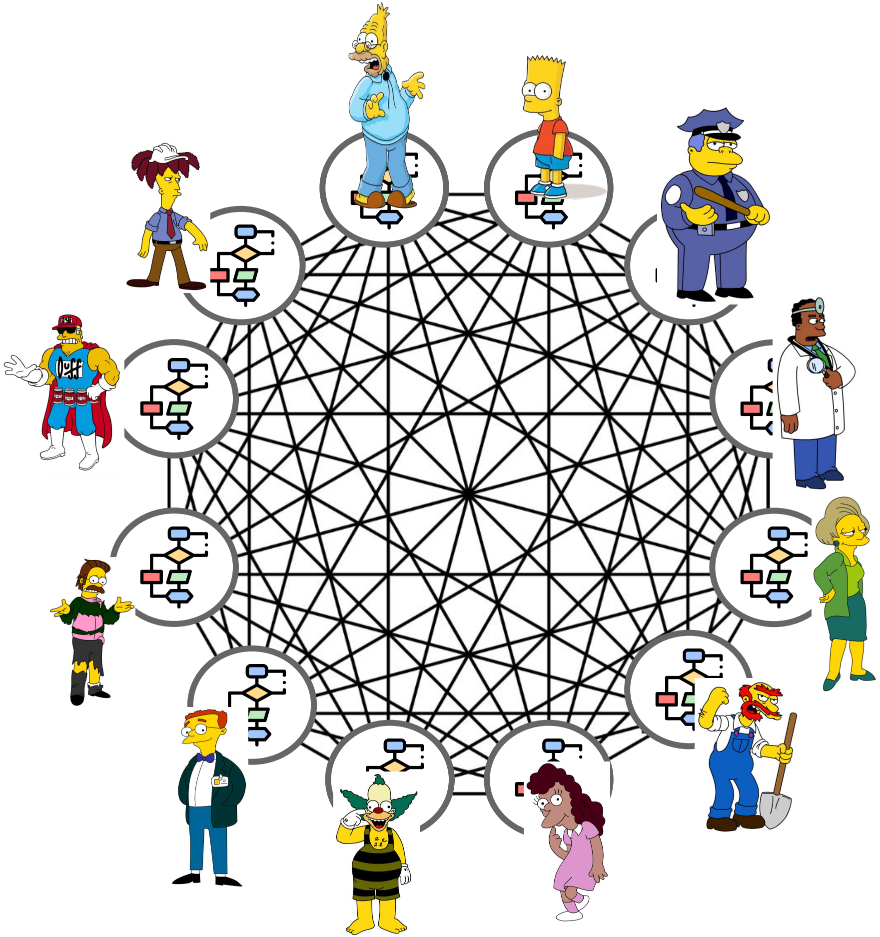
non
synchronous

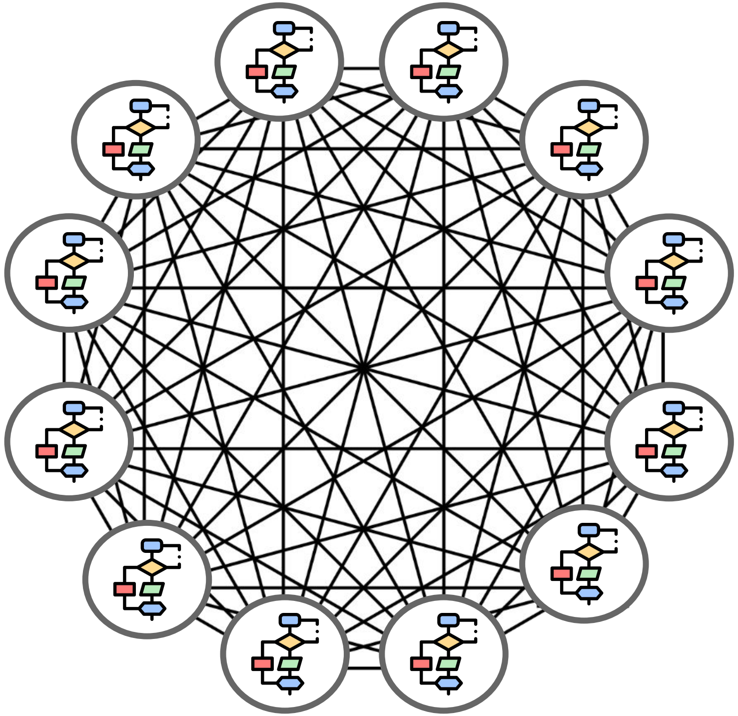


message-passing

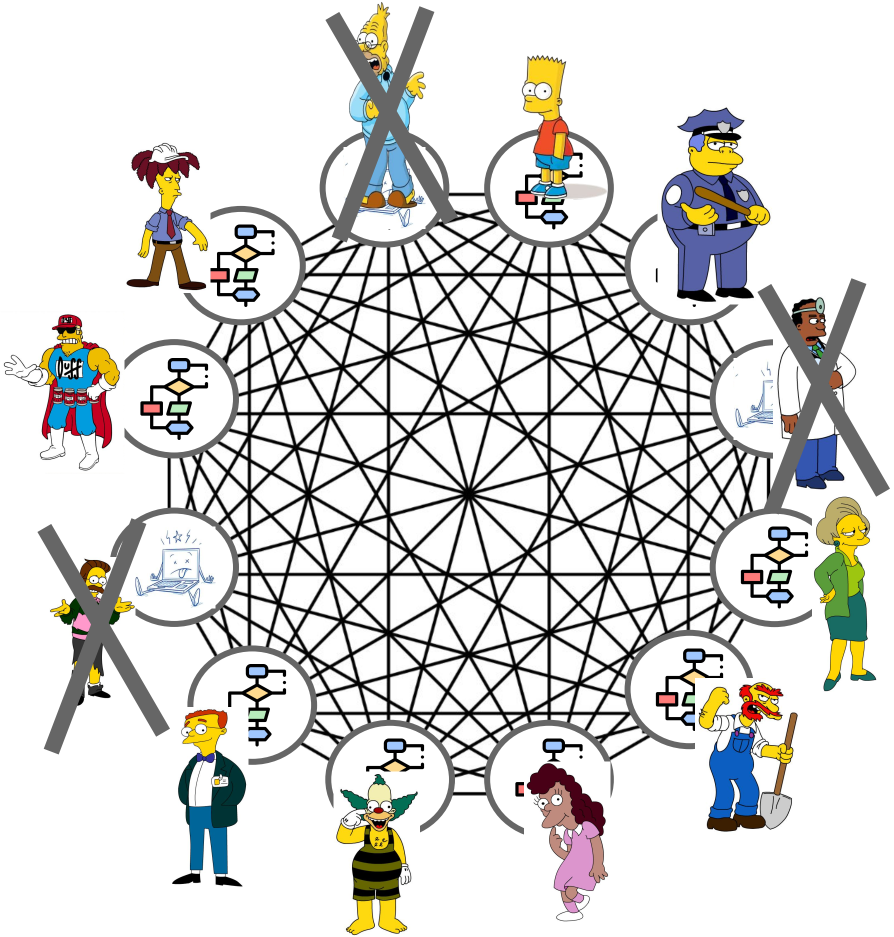
Authenticated



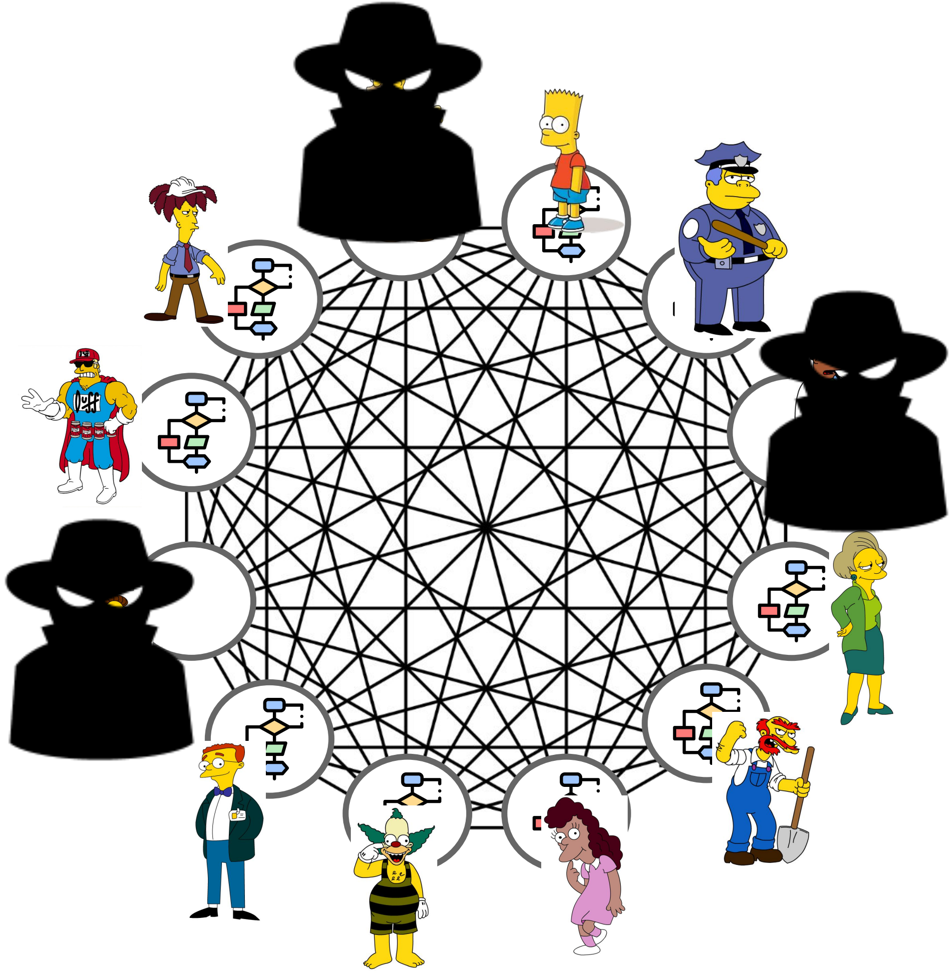





ALL WE REALLY NEED
(FORMALLY PROVEN)
SAFETY
NOTHING BAD WILL HAPPEN
LIVENESS
SOMETHING GOOD WILL DEFINITELY HAPPEN



resilient



resilient

How many  can we tolerate ?

$$f < n/2$$

How many  can we tolerate ?

$$t < n/3$$

Gracefully Degrading Task

$< f_o$ ➔ (Safety + Liveness)

$>= f_o$ ➔ Safety (~~Liveness~~)

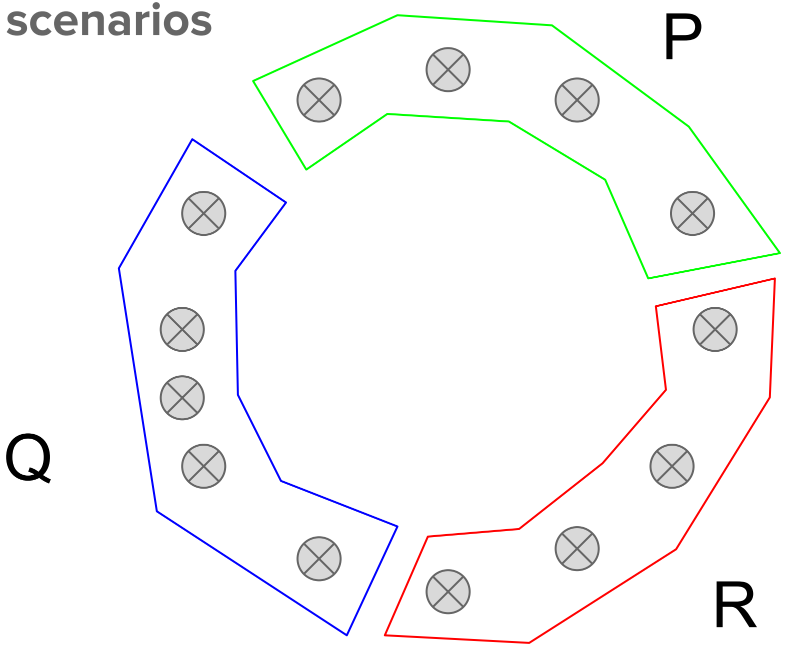
Gracefully Degrading Byzantine Task is generally impossible

 $< t_0 \rightarrow$ (Safety + Liveness)

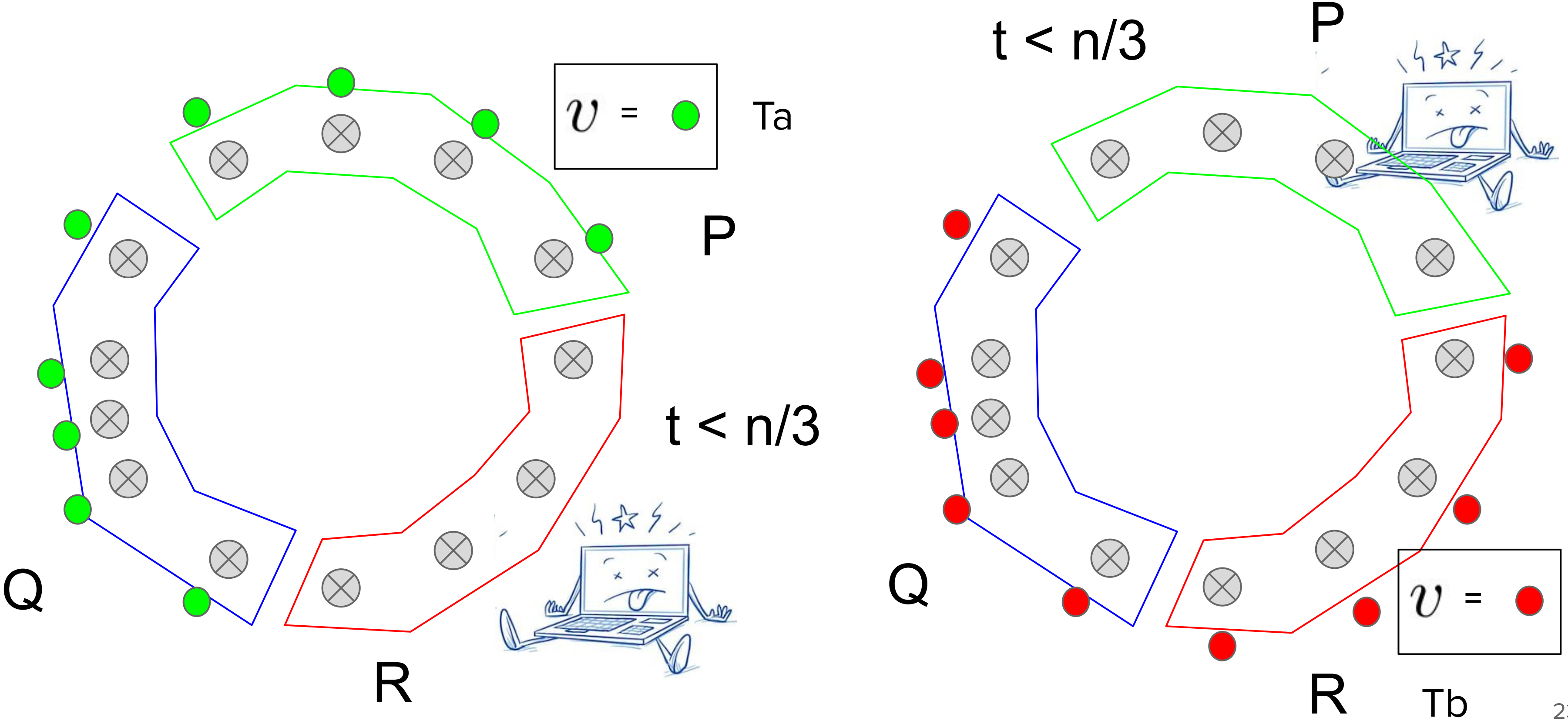
 $>= t_0 \rightarrow$ Safety (~~Liveness~~)

Impossibility of solving GDBC

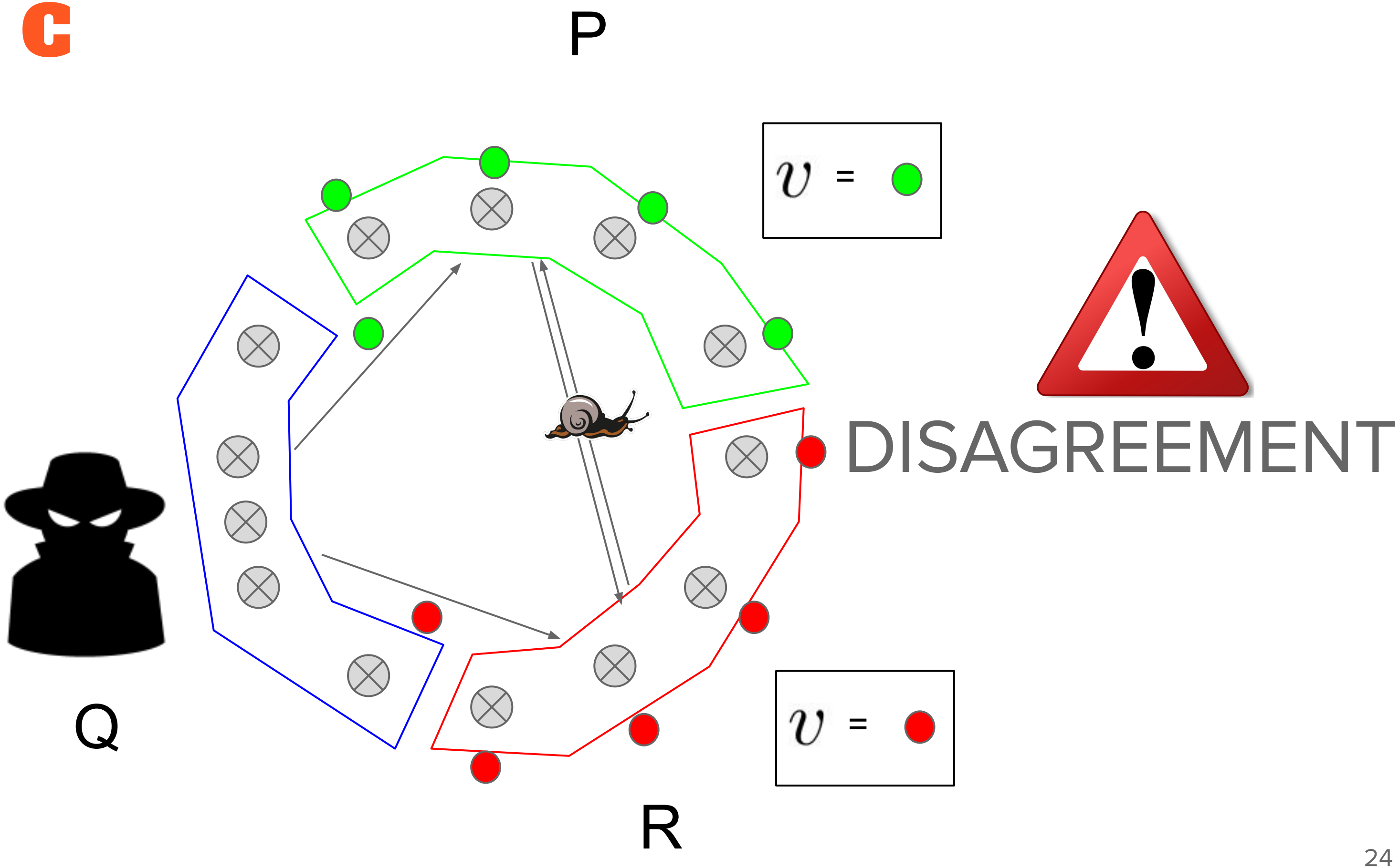
Undistinguishable scenarios

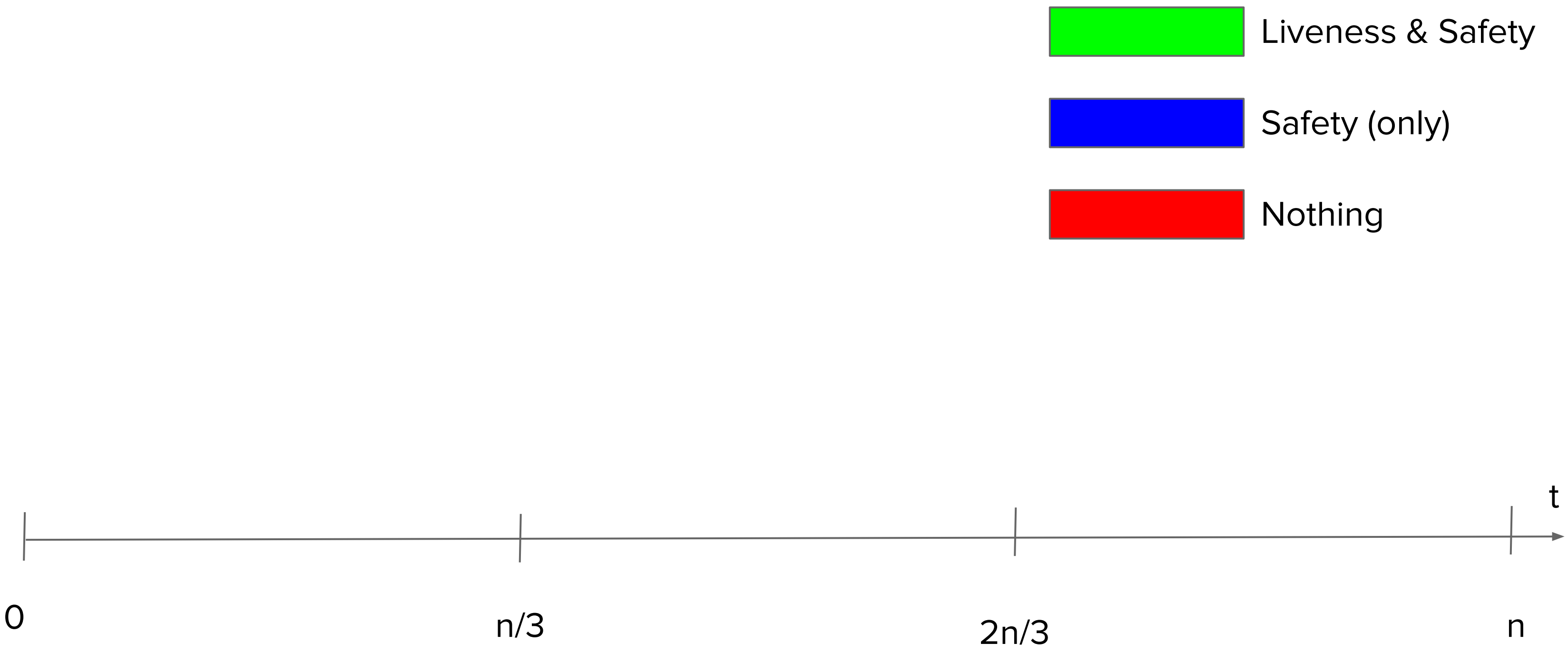


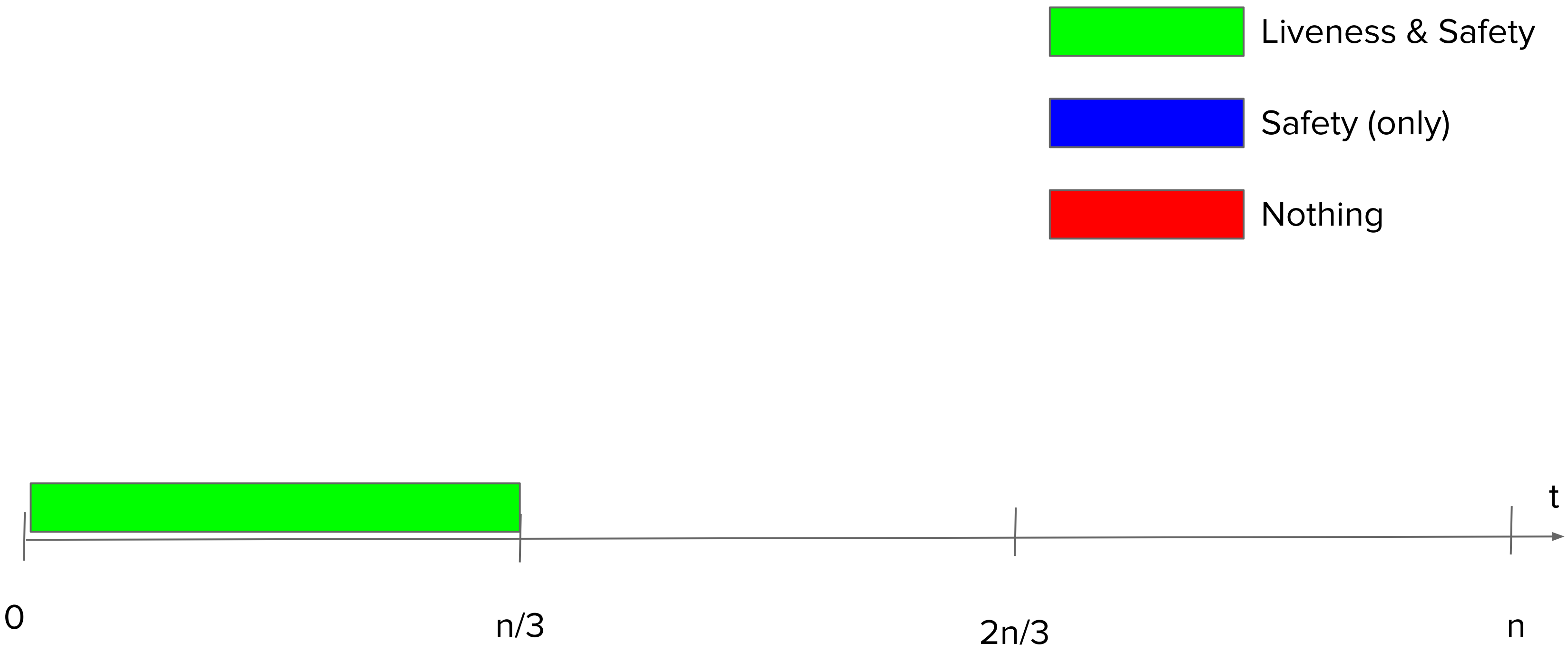
Scenarios A and B

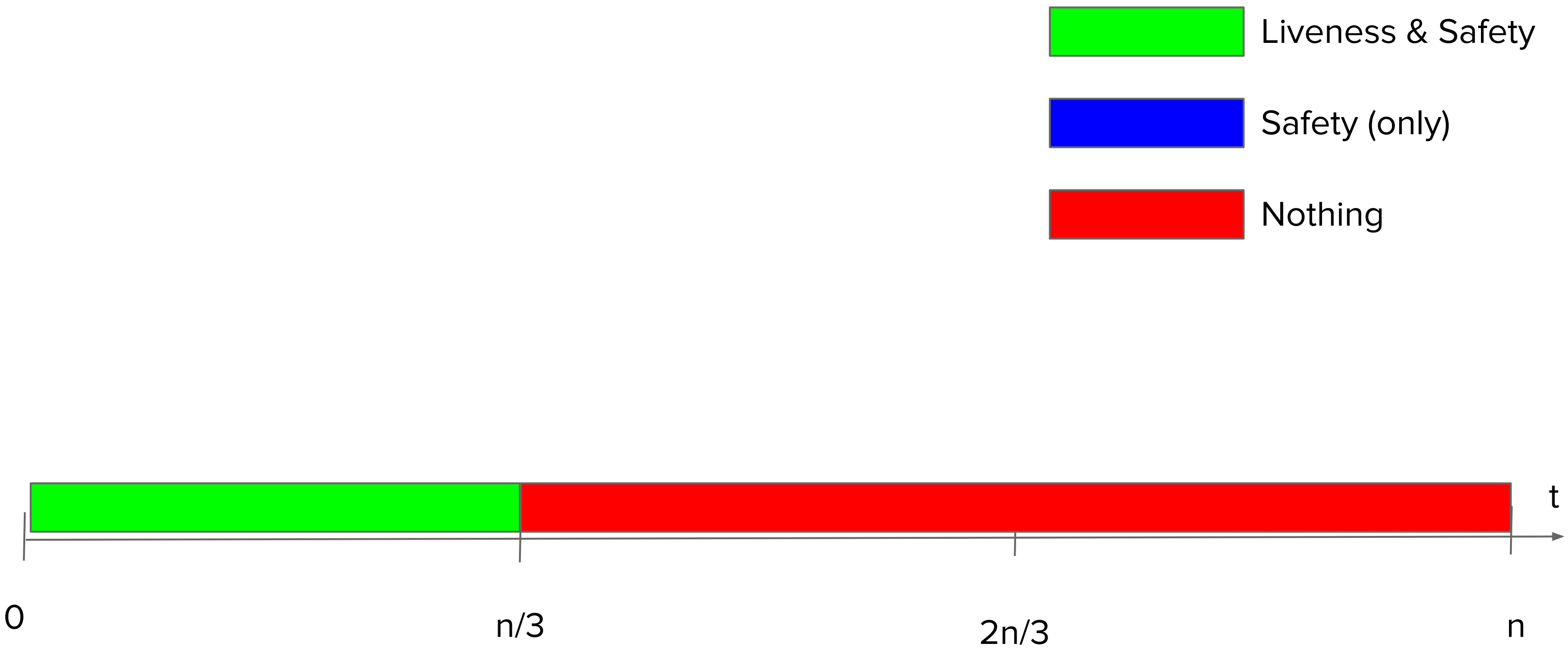


Scenario C

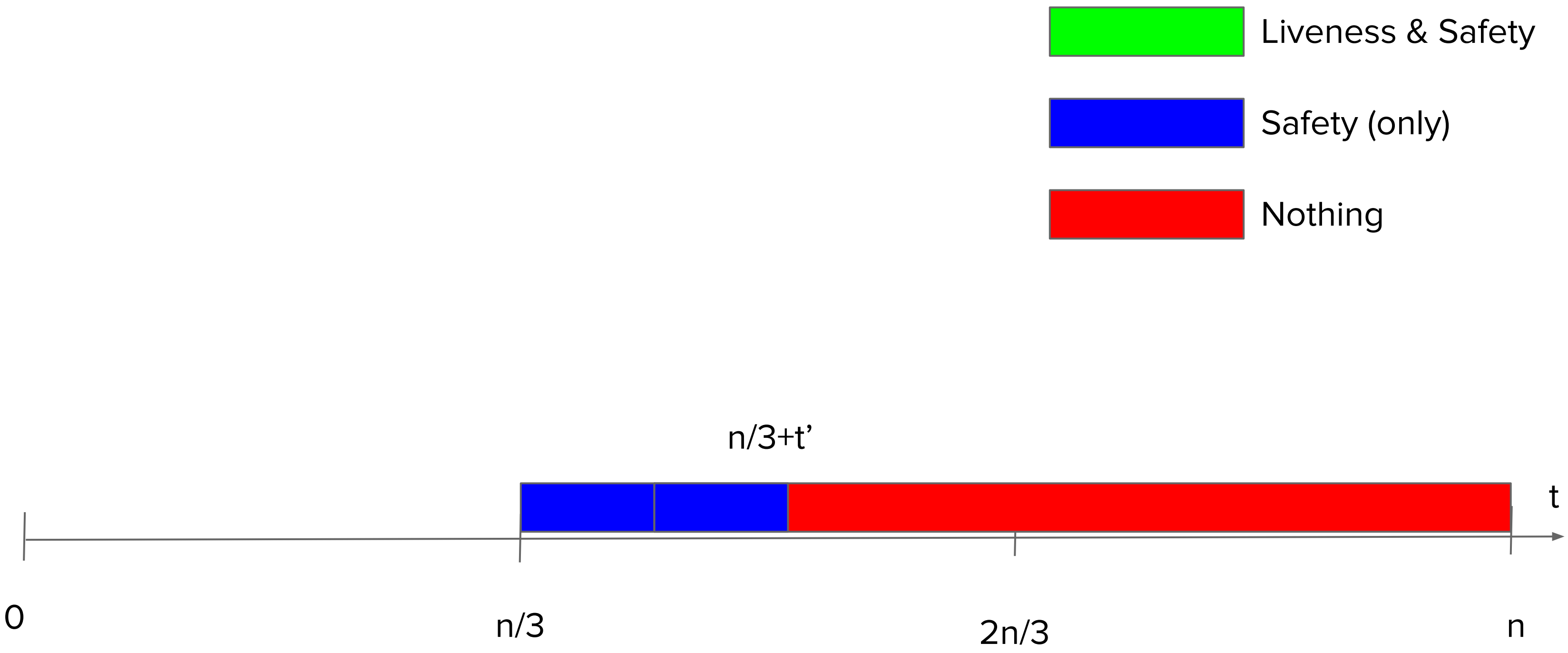


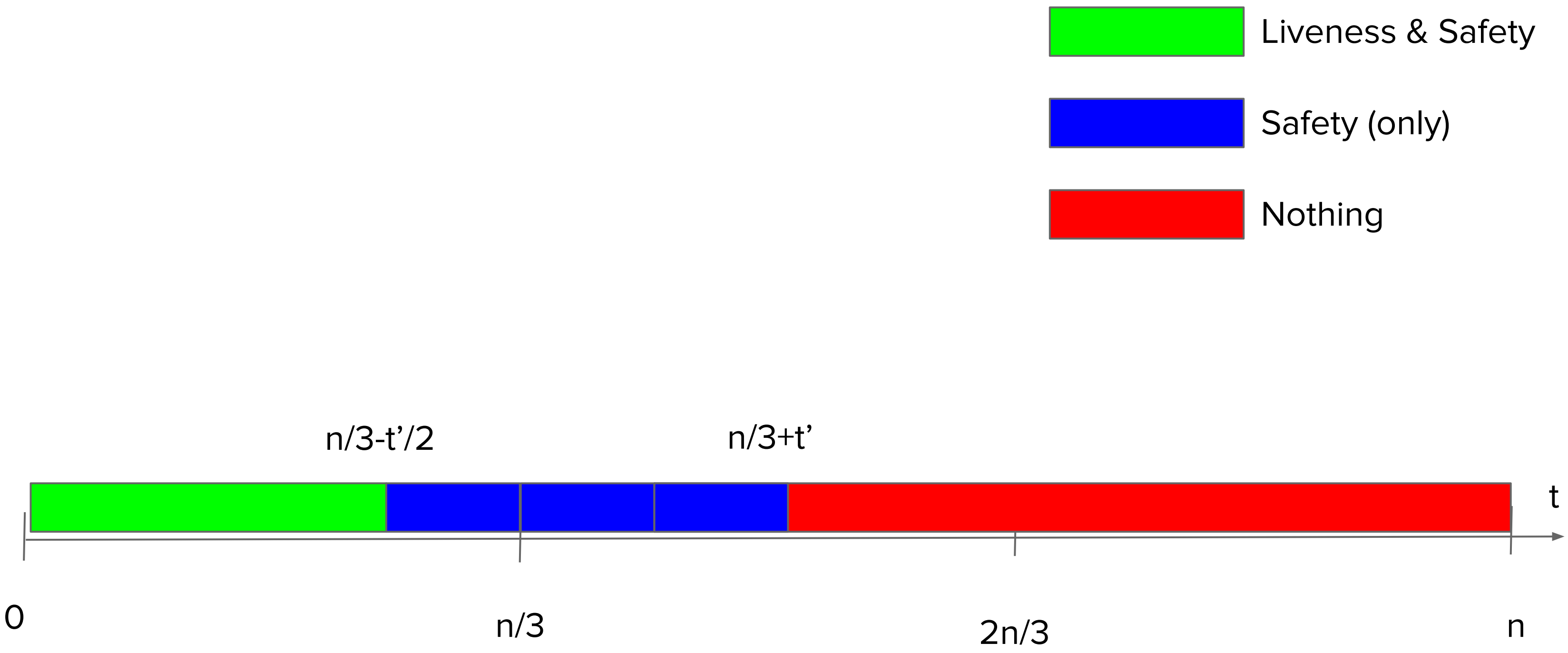

Liveness & Safety
Safety (only)
Nothing
t
0
n/3
2n/3
n

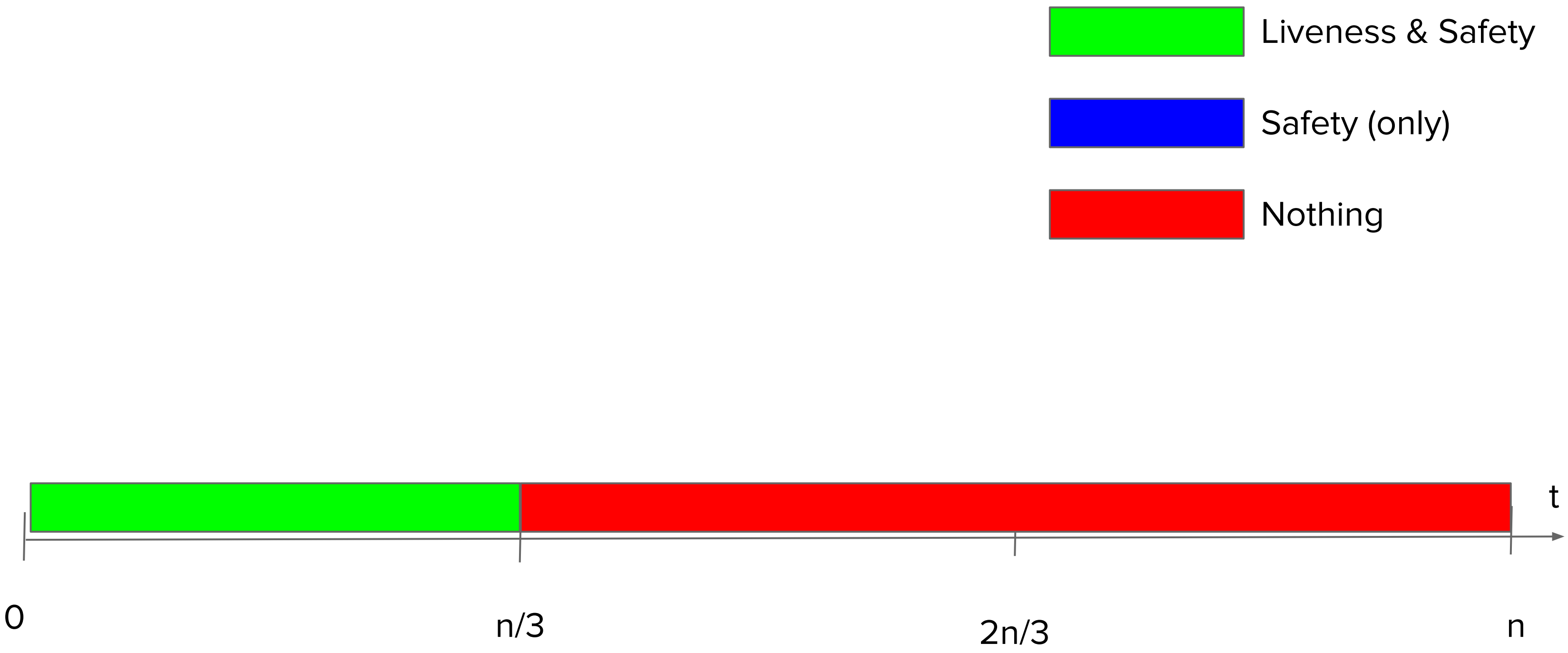

Liveness & Safety
Safety (only)
Nothing
t
0
n/3
2n/3
n



Liveness & Safety
Safety (only)
Nothing
t
0
n/3
2n/3
n


Liveness & Safety
Safety (only)
Nothing
n/3+t’
t
0
n/3
2n/3
n

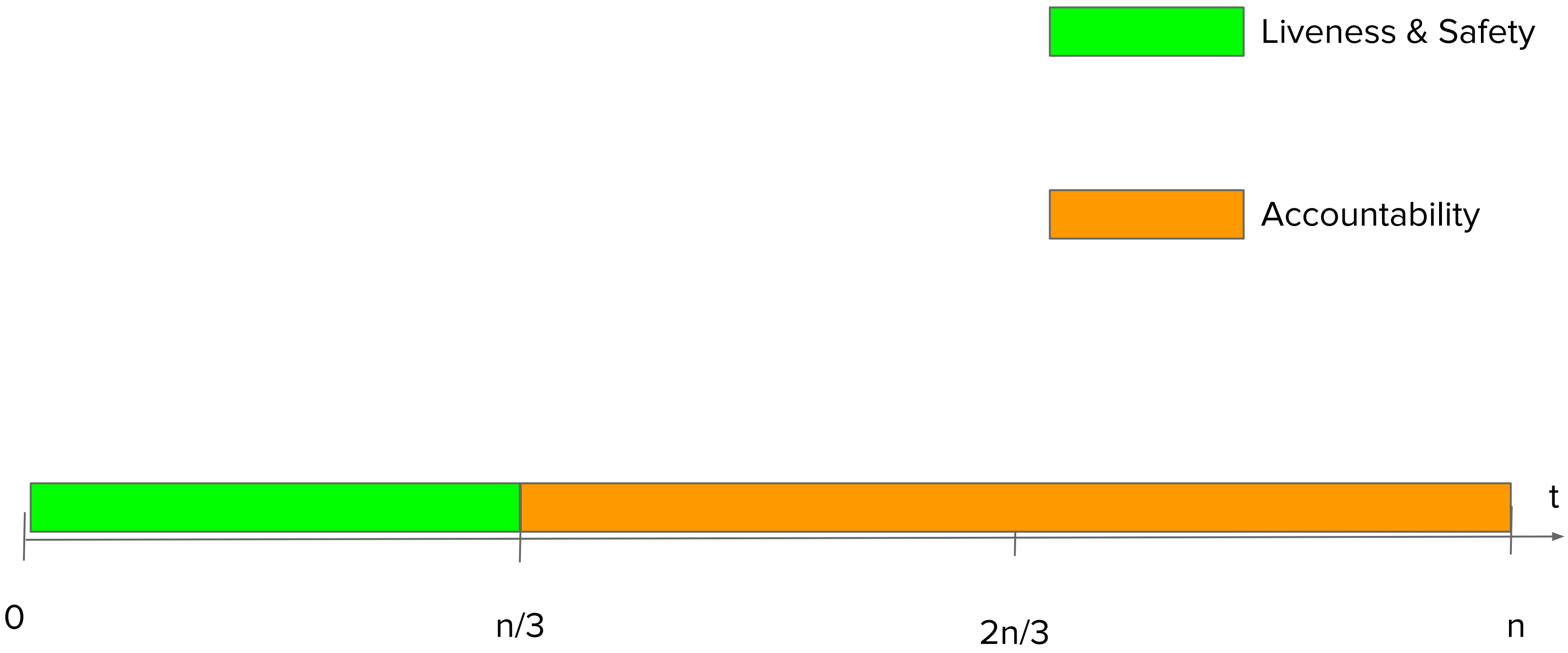

Liveness & Safety
Safety (only)
Nothing
n/3-t'/2
n/3+t'
t
0
n/3
2n/3
n

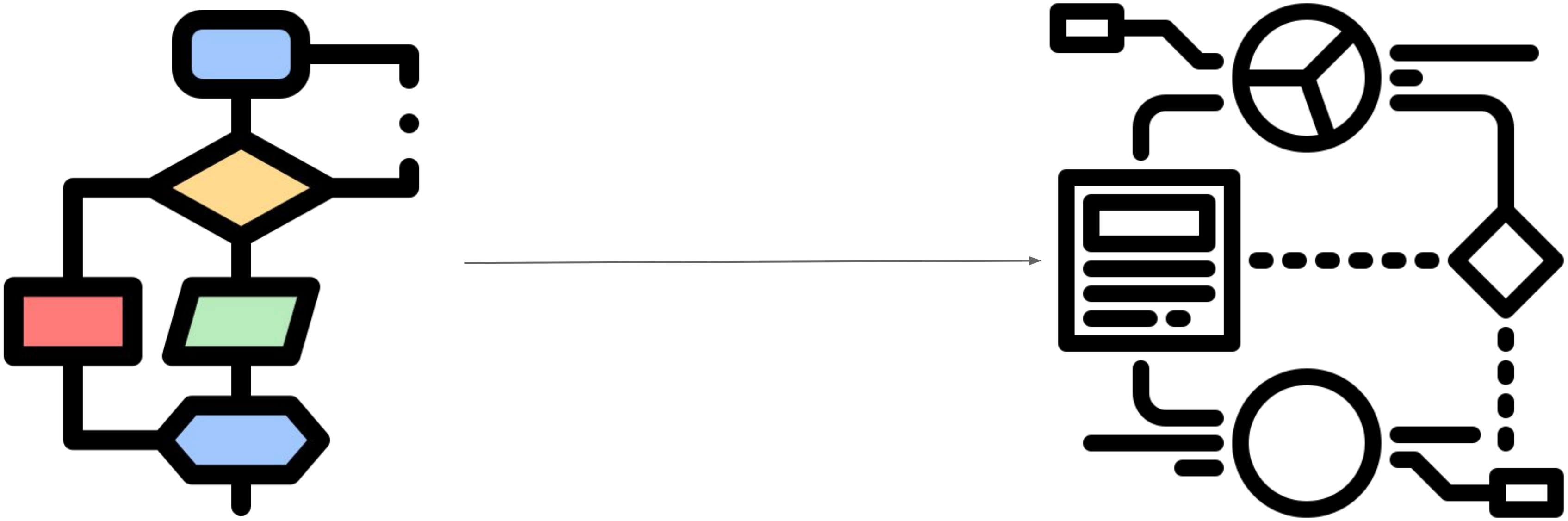

Liveness & Safety
Safety (only)
Nothing
t
0
n/3
2n/3
n

Accountable Algorithm

 ≤ t_0 → Safety + Liveness

 → Safety Violation → Detection → 


Liveness & Safety
Accountability
t
0
n/3
2n/3
n



- solves the same problem with same resiliency
- accountability in case of safety violation

Questions

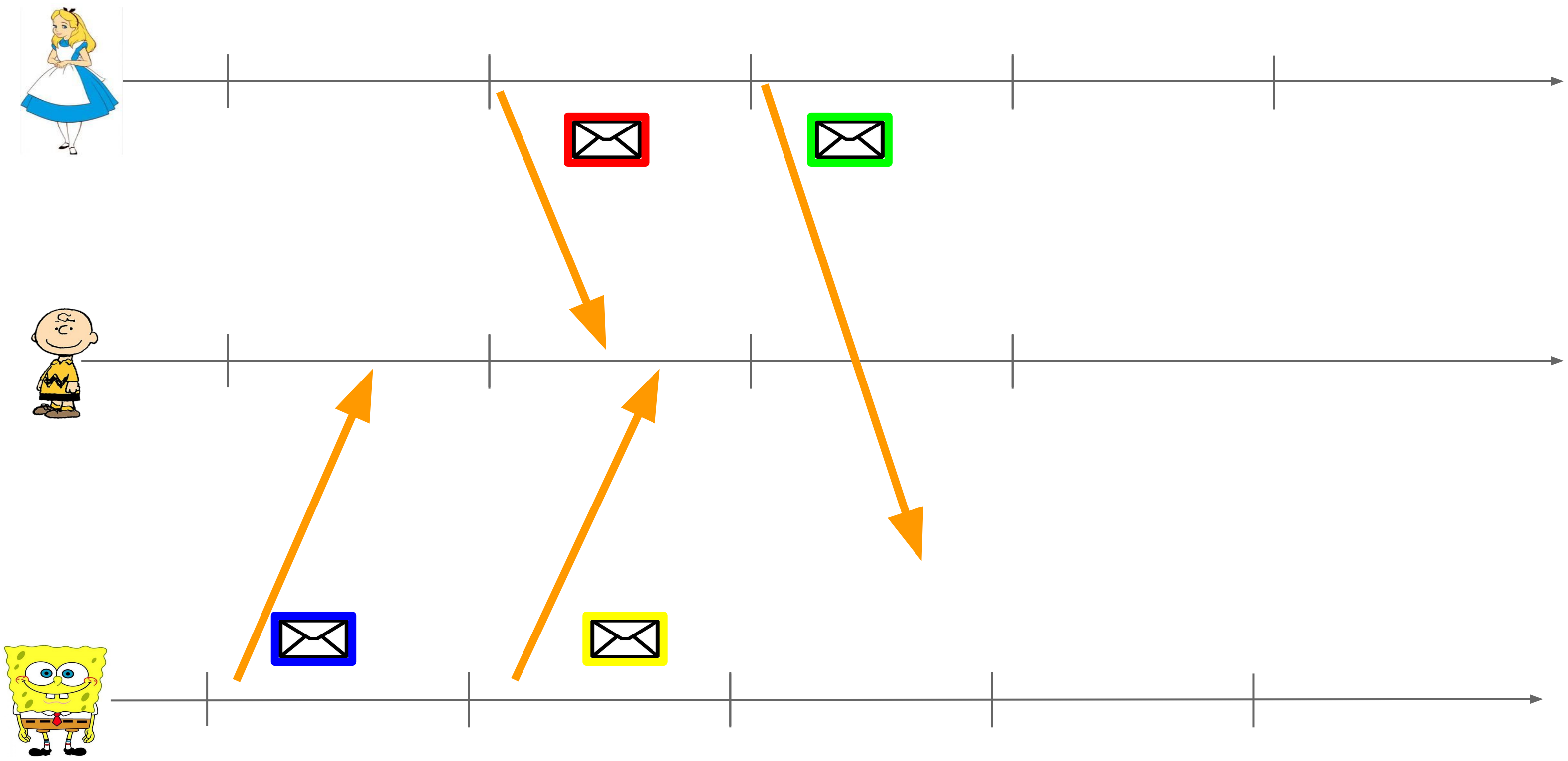
What are the Byzantine faults to detect/hide ?

Can they cause safety violation ?

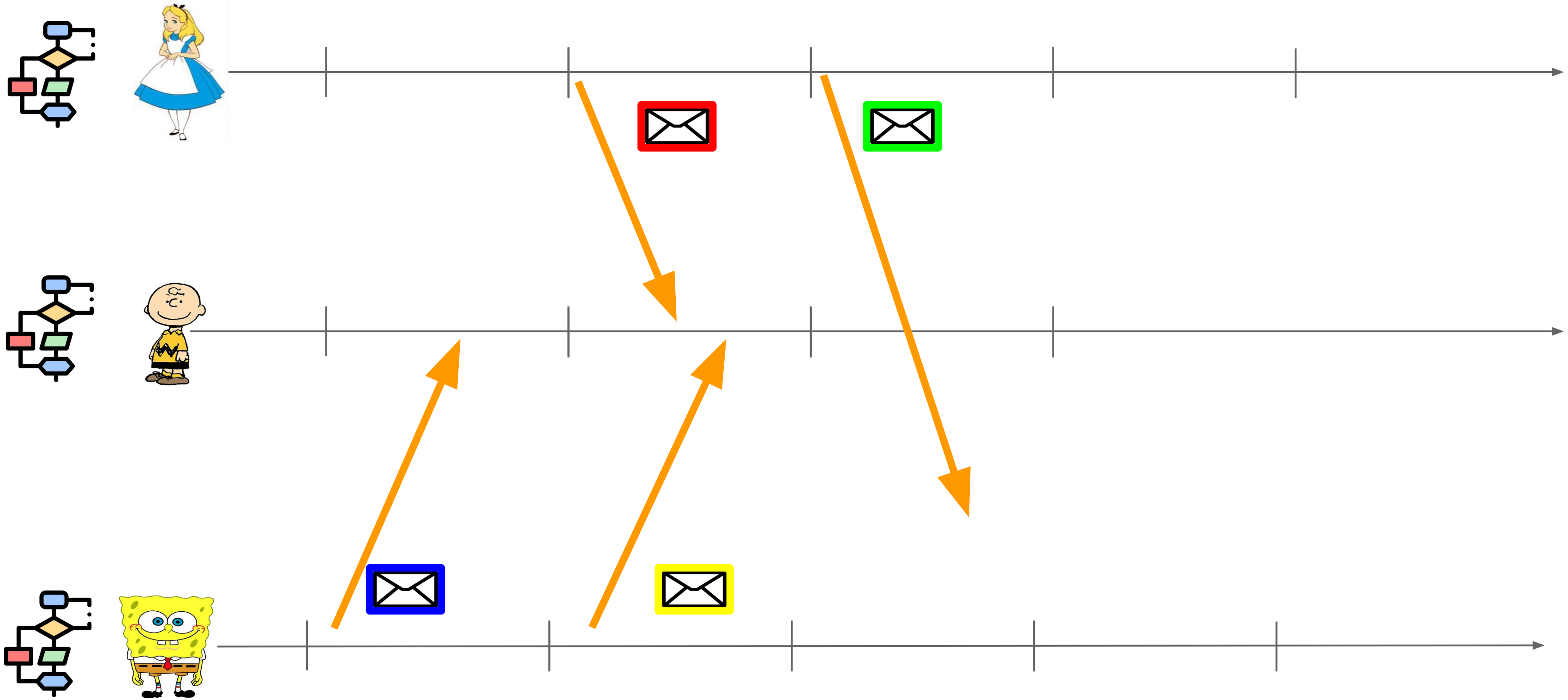
What is the cost to detect it ?

Fault Classification

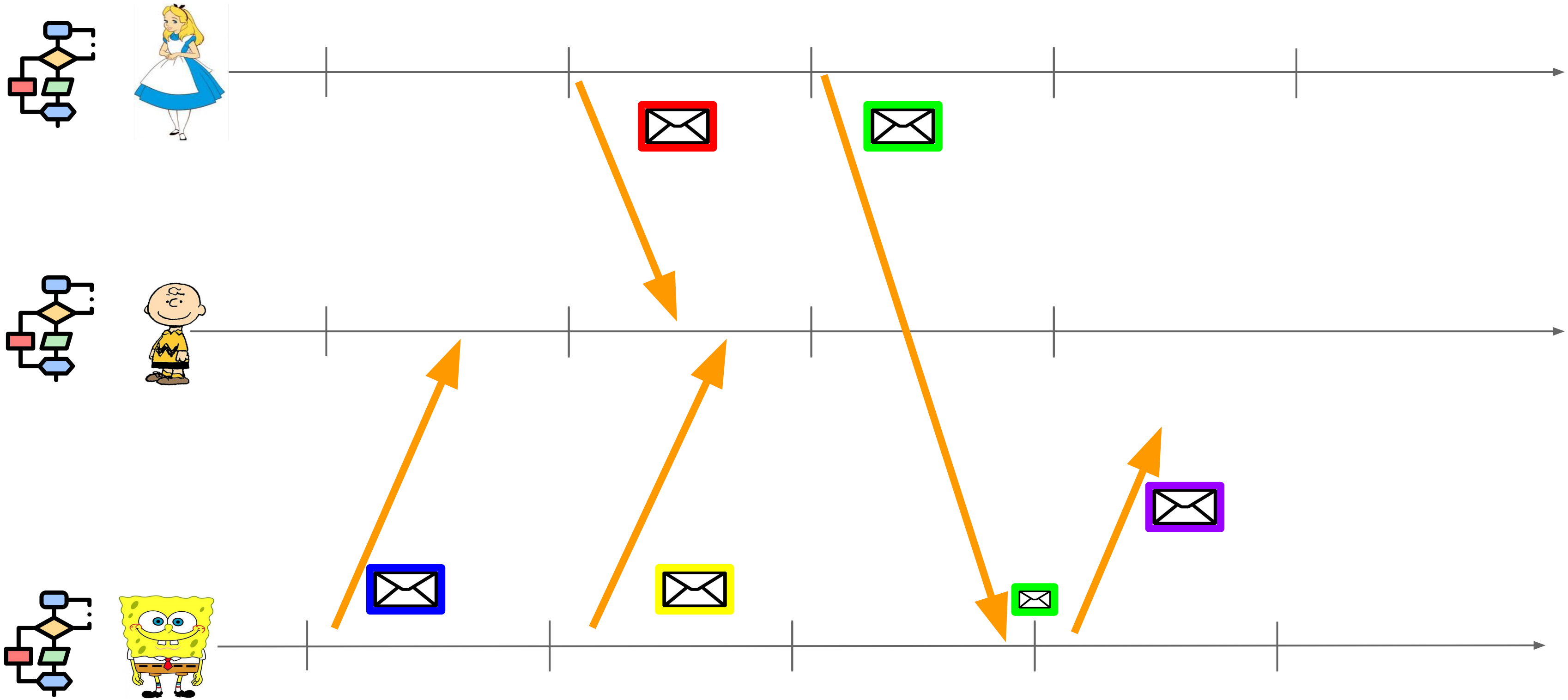
What is a fault ?



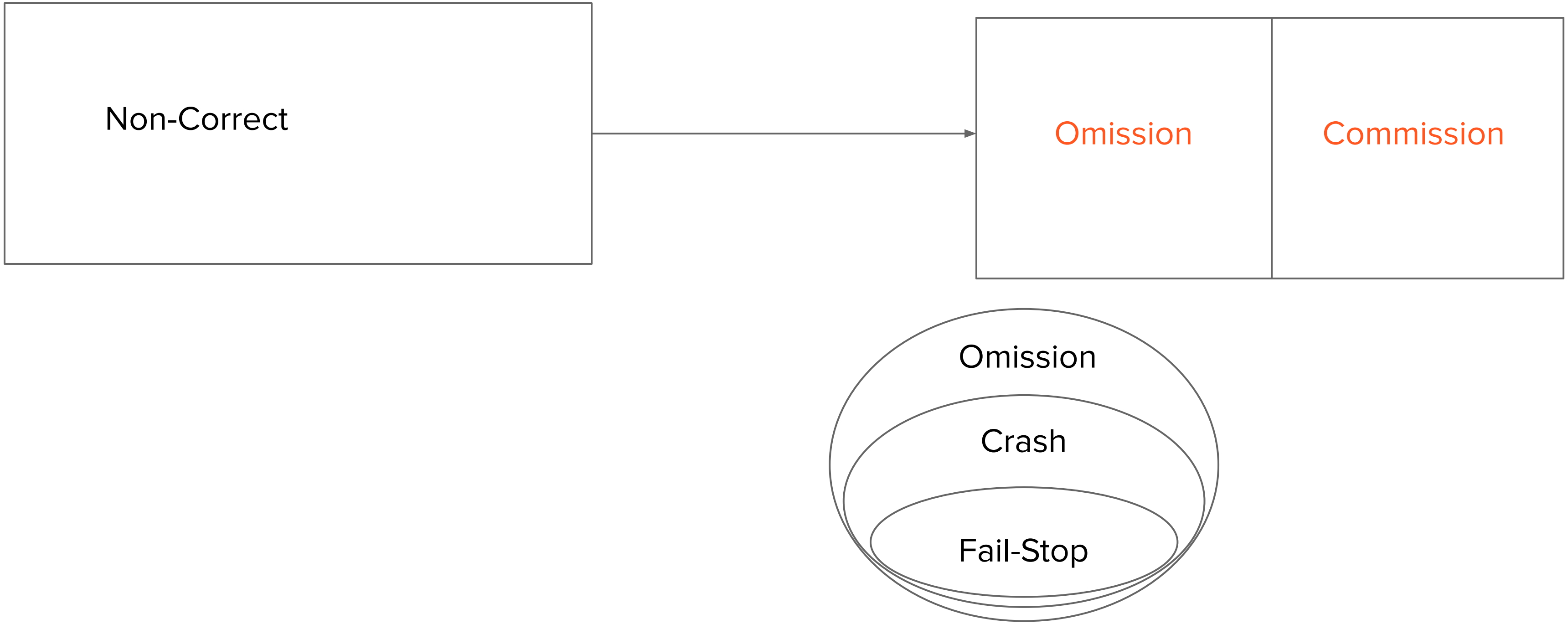
What is a fault ?

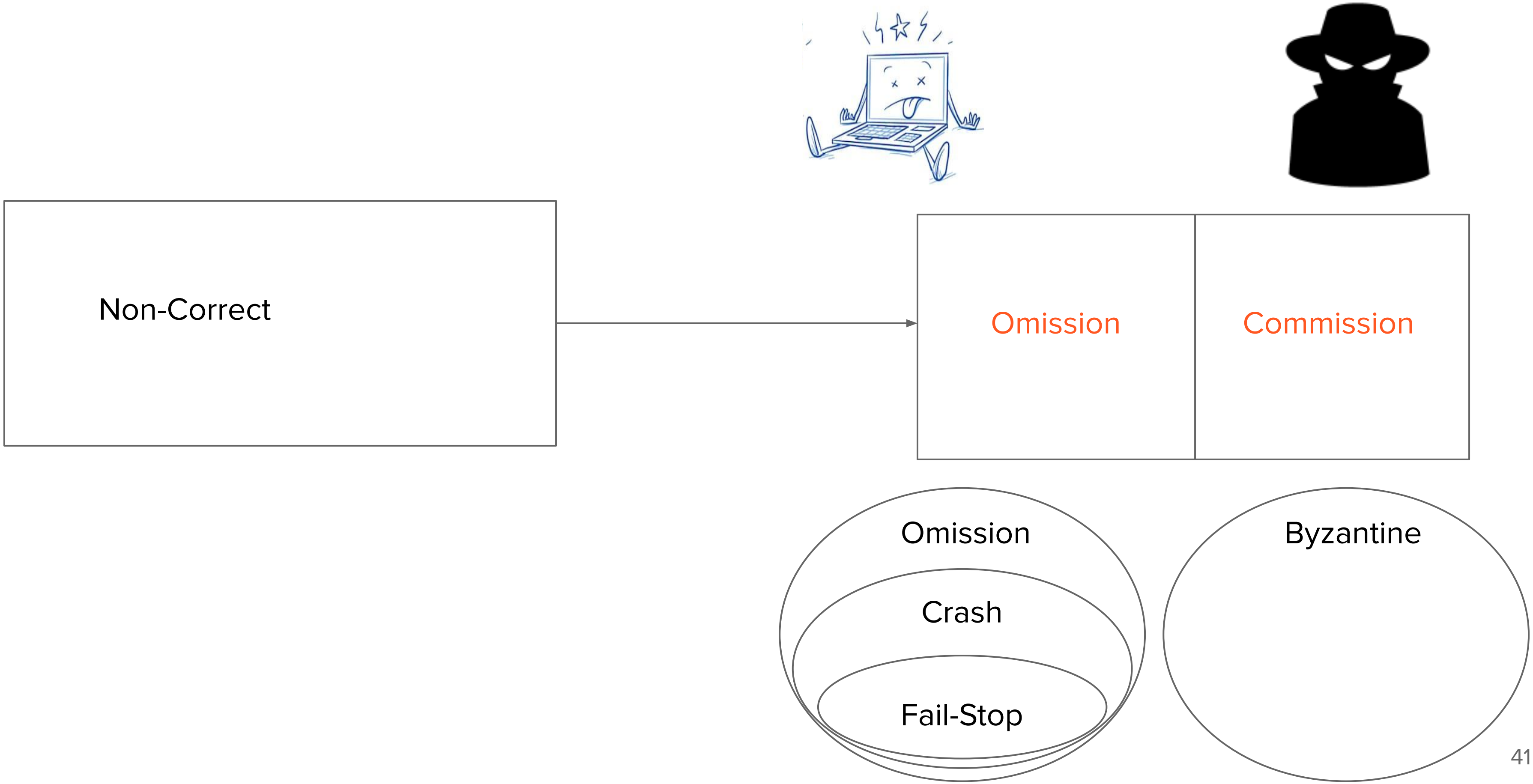


What is a fault ?

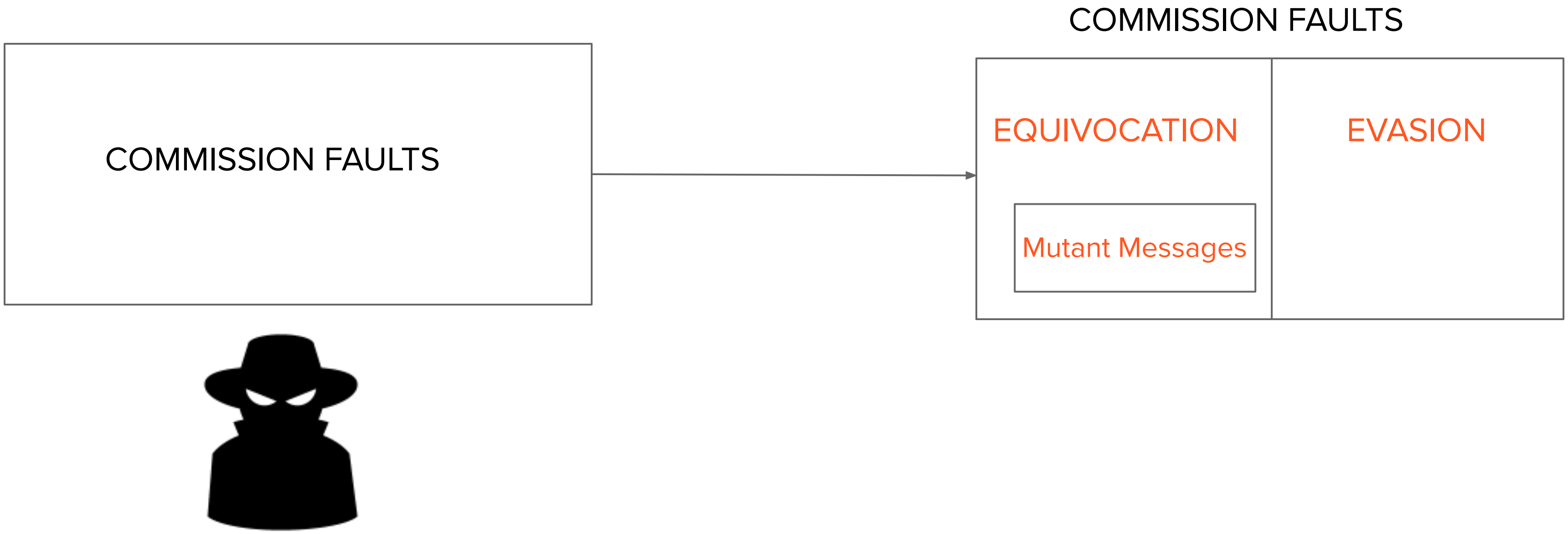



Processes
Correct
Non-Correct


Non-Correct
Omission
Commission
Omission
Crash
Fail-Stop


Non-Correct
Omission
Commission
Omission
Crash
Fail-Stop
Byzantine

1st minor contribution: formal partitioning



Faults & Accountability

1) Commission faults are necessary to violate safety

1) Commission faults are necessary to violate safety

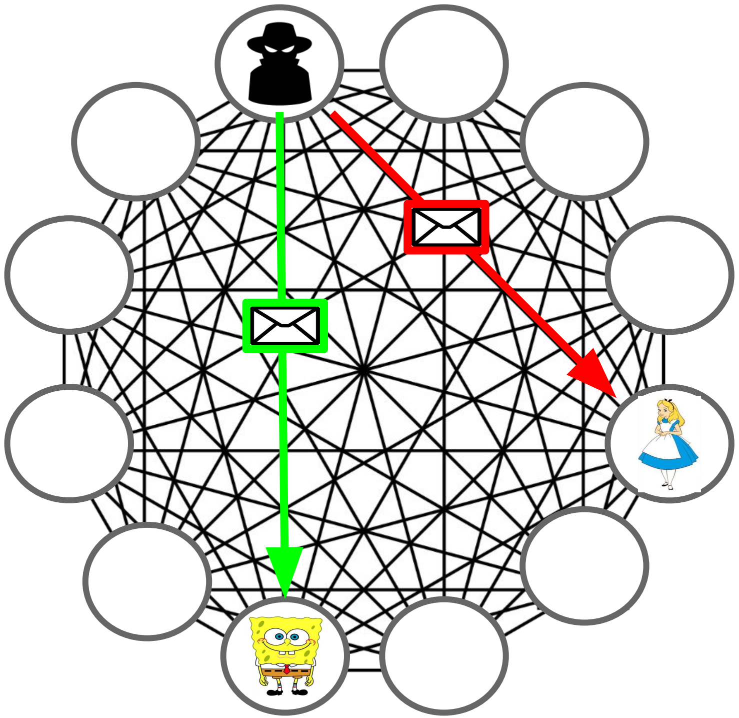
2) Only commission faults detection is possible

3) Commission faults detection is necessary* and sufficient** to provide accountability

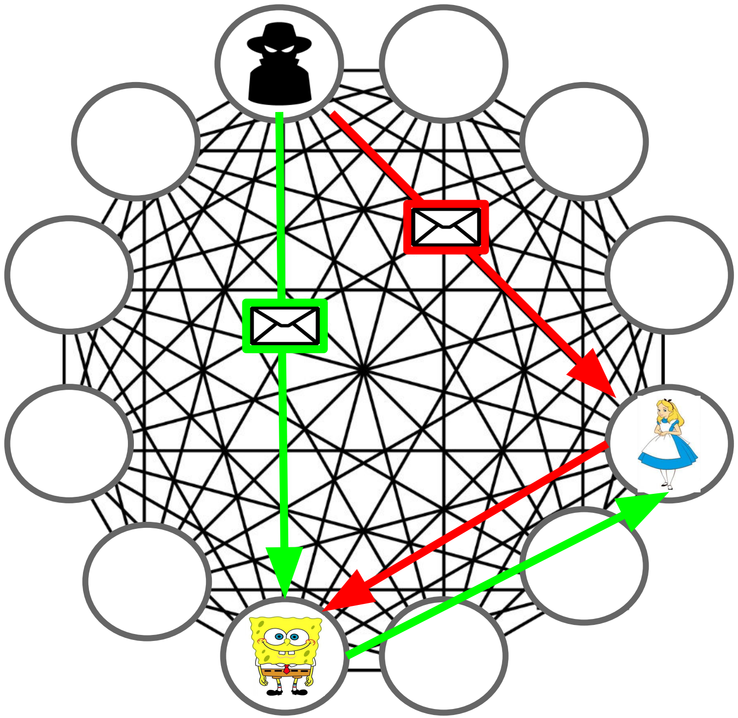
The cost of detection

Warm up: Equivocation

Warm up: Equivocation

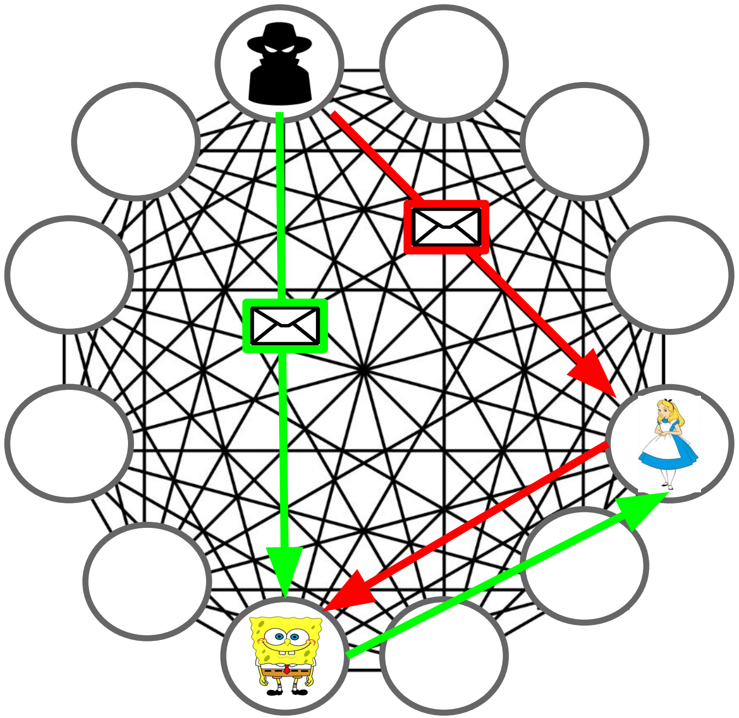


Warm up: Equivocation

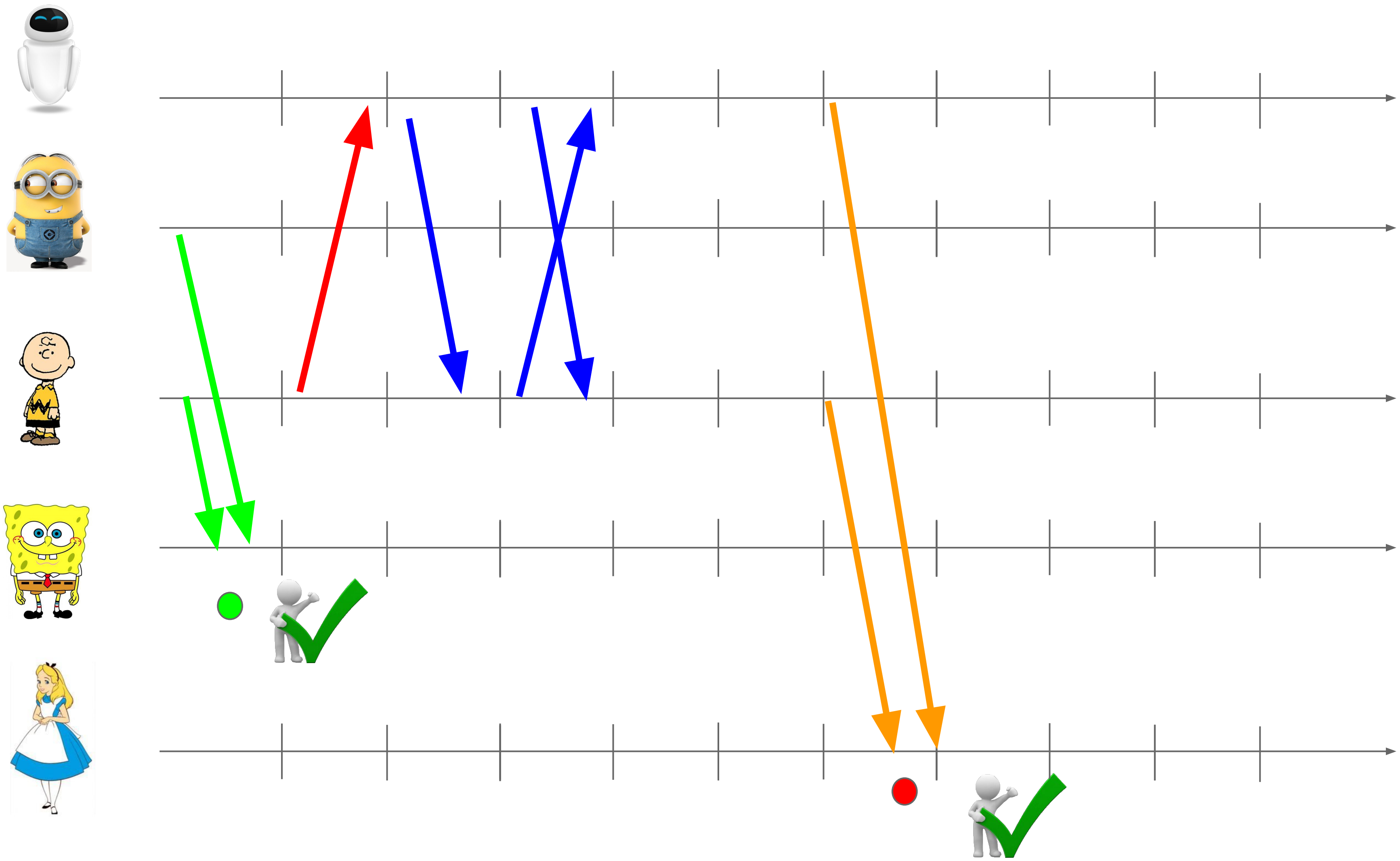


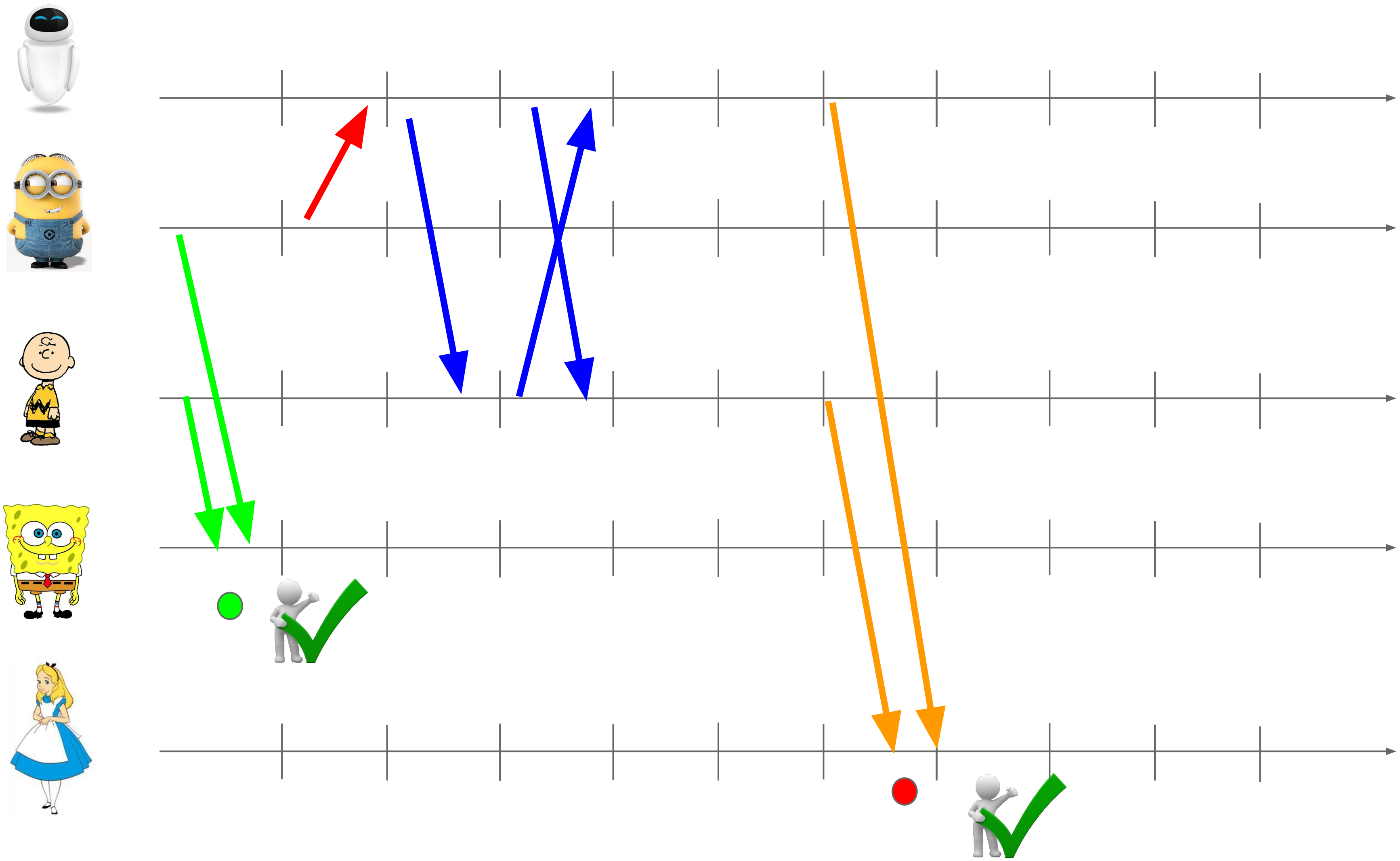
Warm up: Equivocation

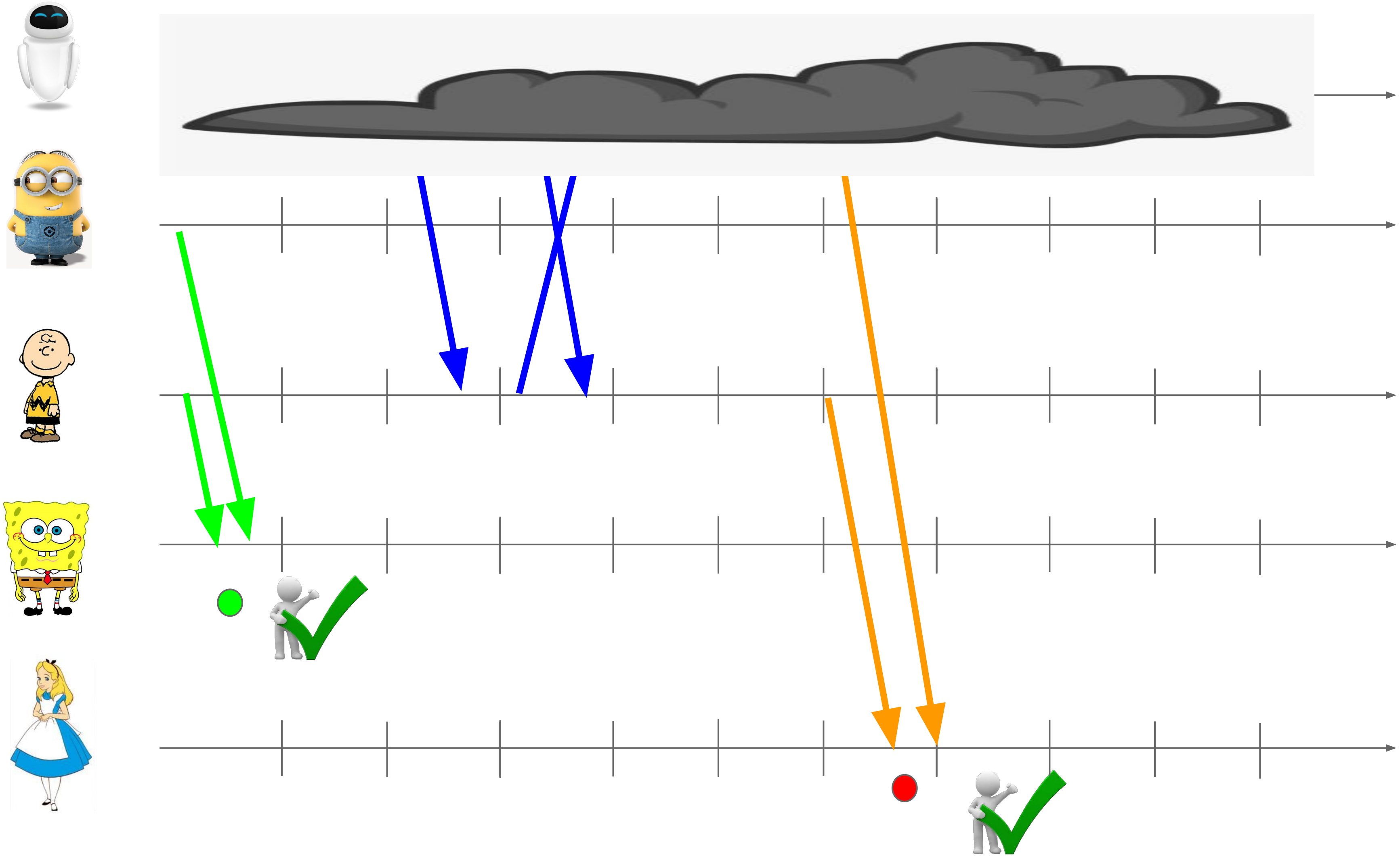
O(n^2)



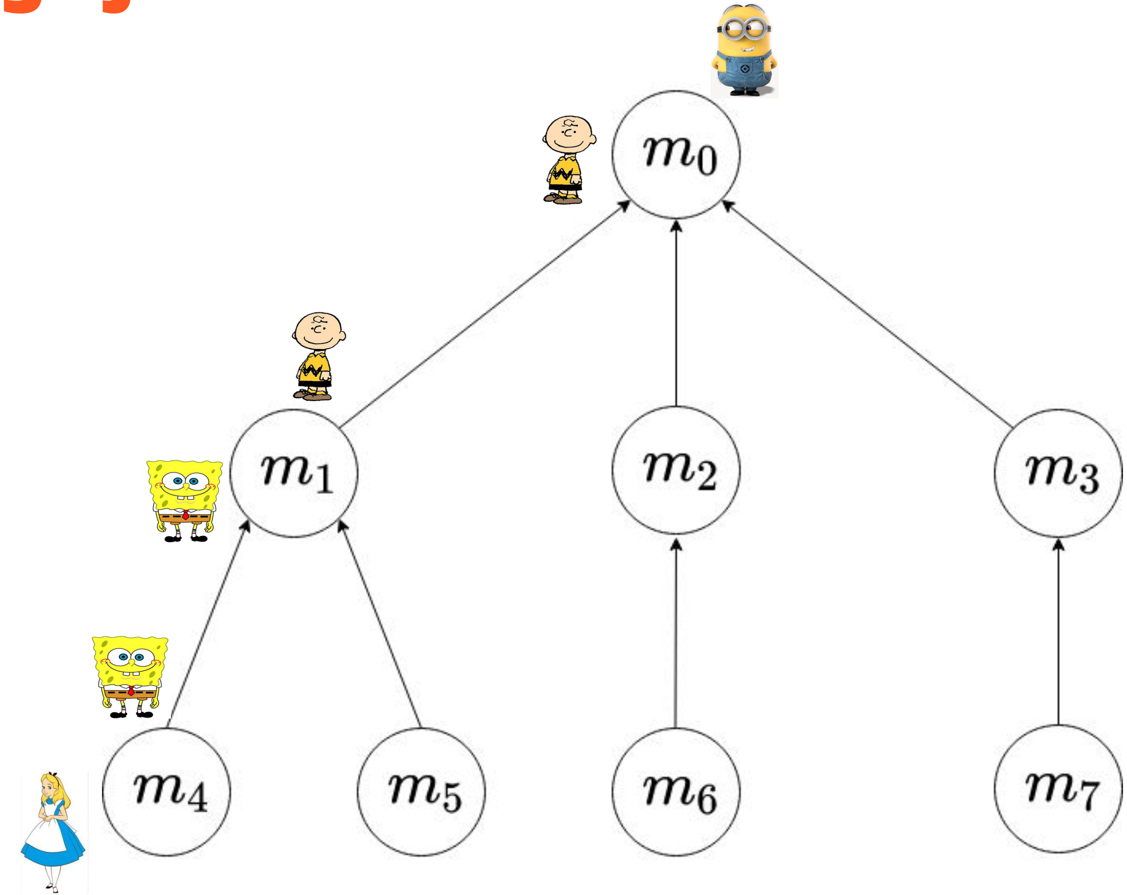
Chained Commission Faults



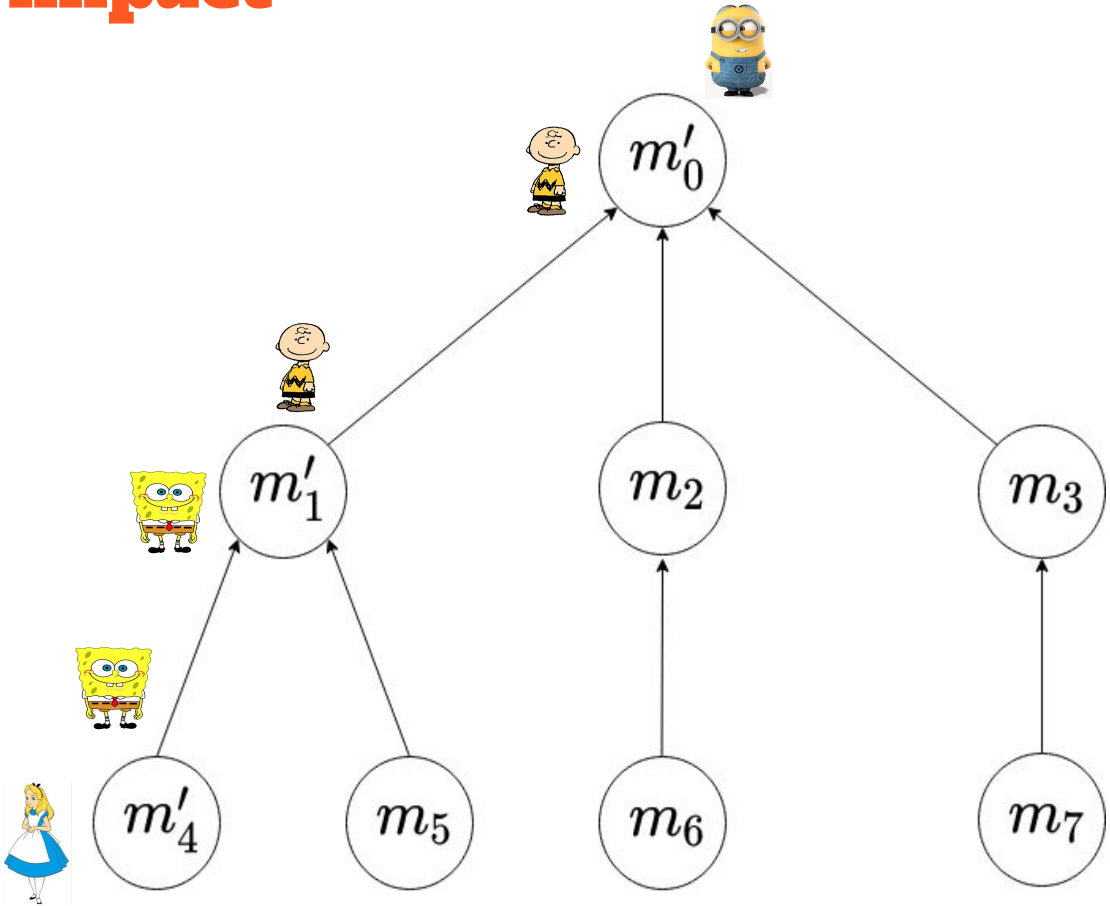




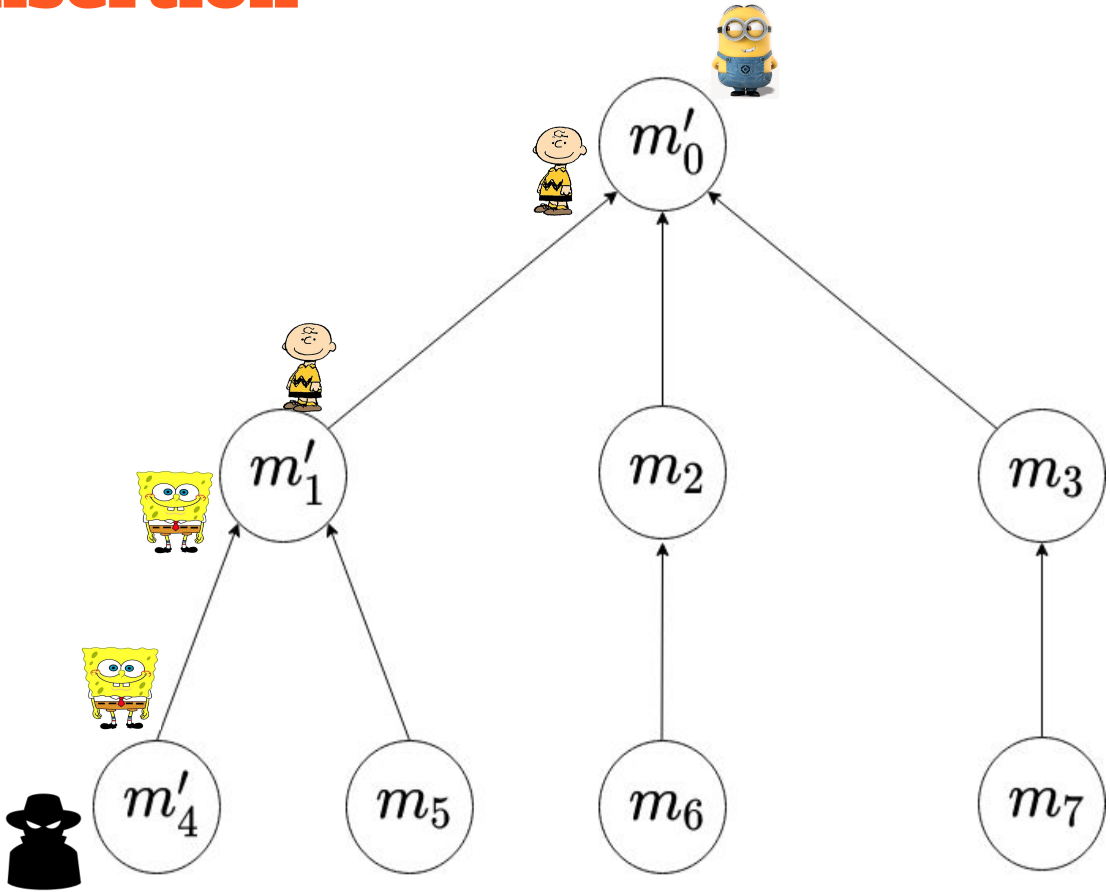
message justification



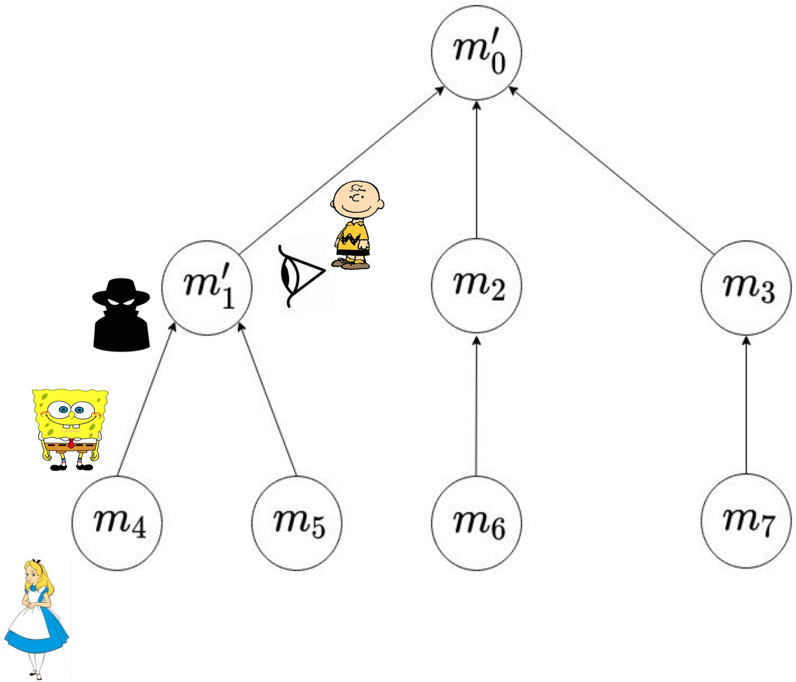
causal impact

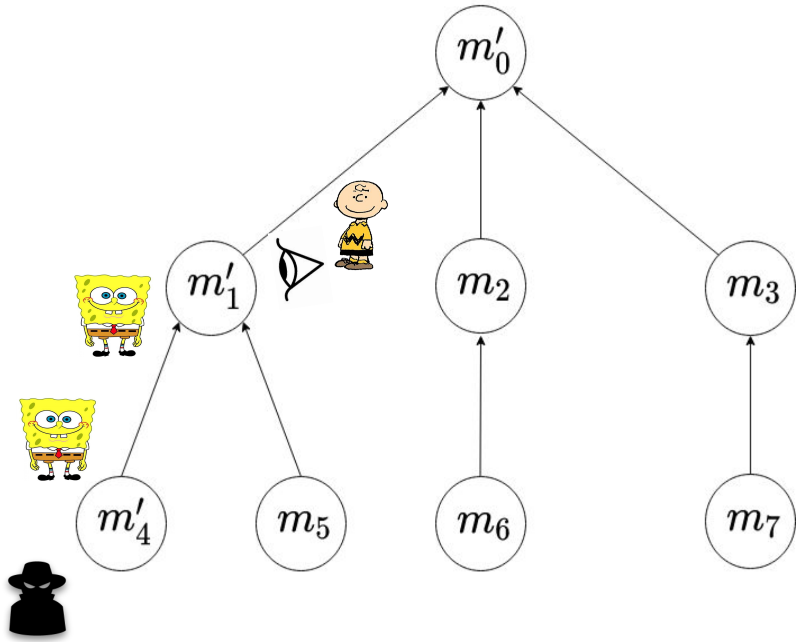


fault insertion

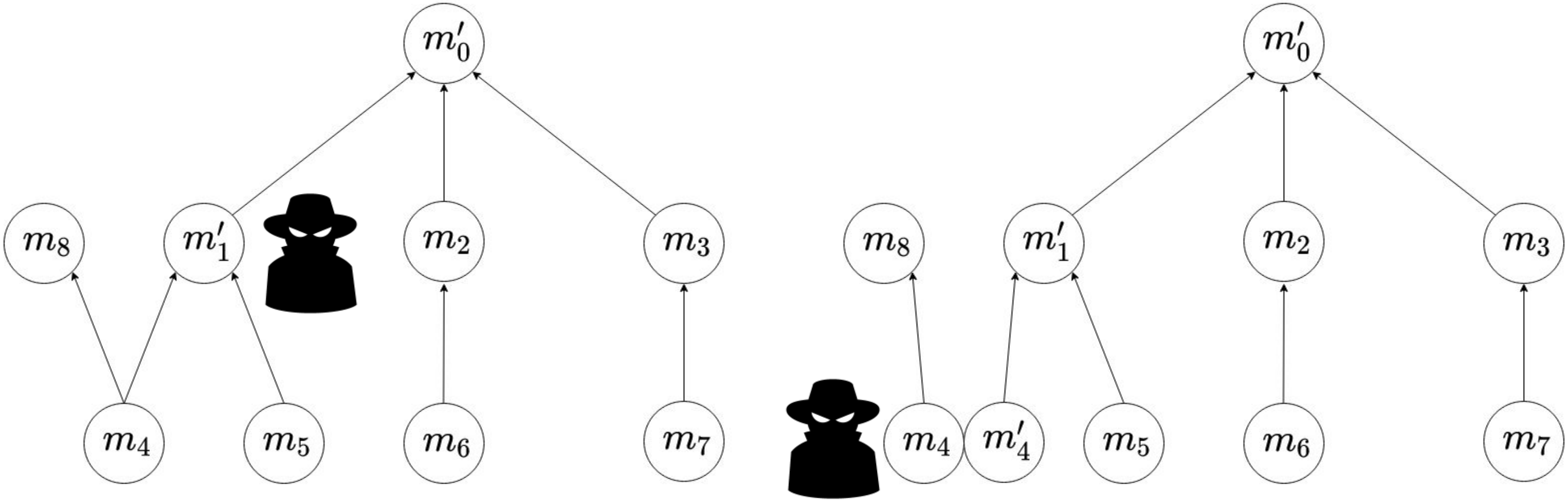


find the culprit

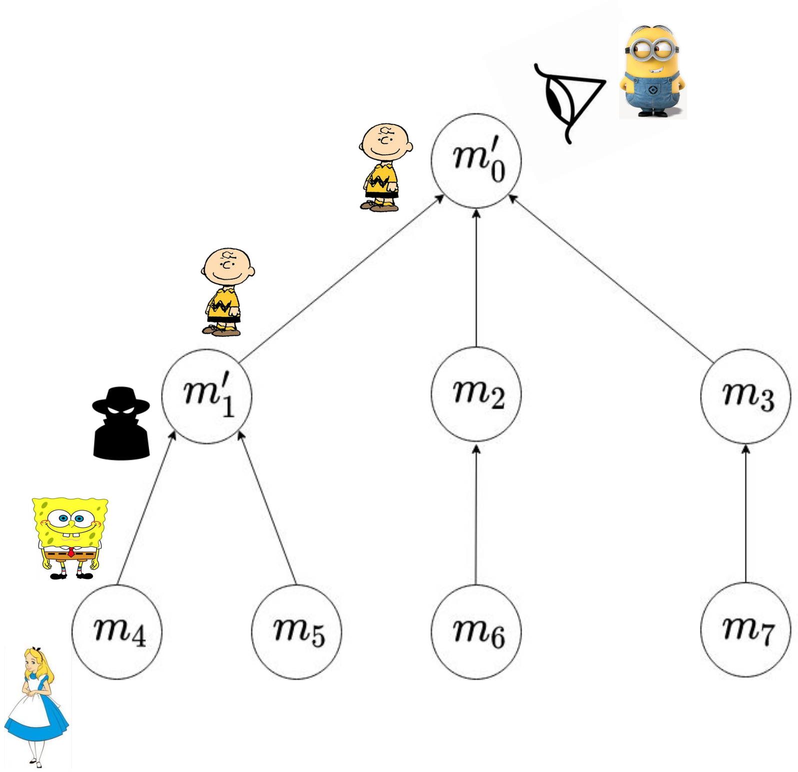


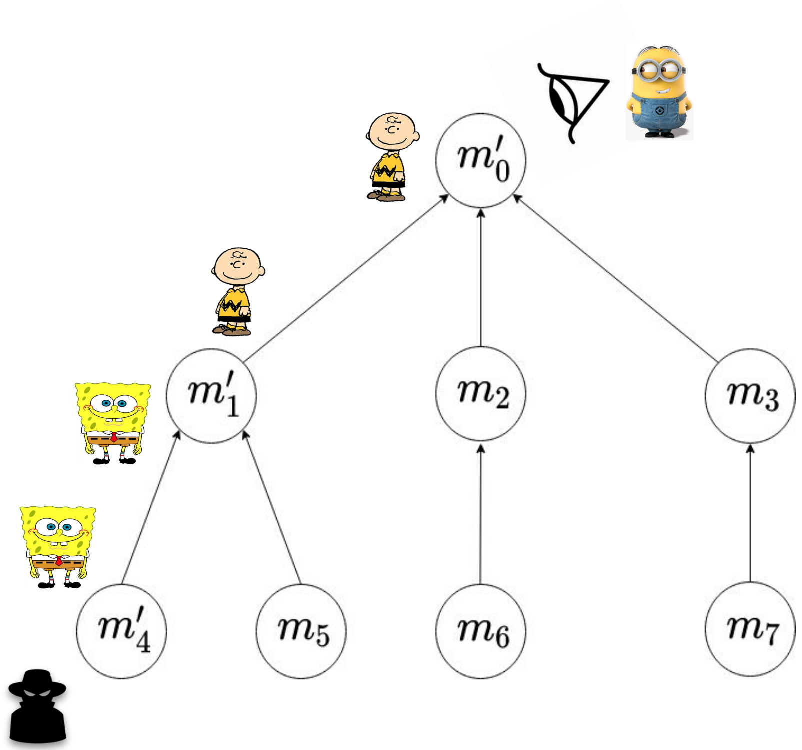


find the culprit with witnesses



find the culprit with justification with degree 1



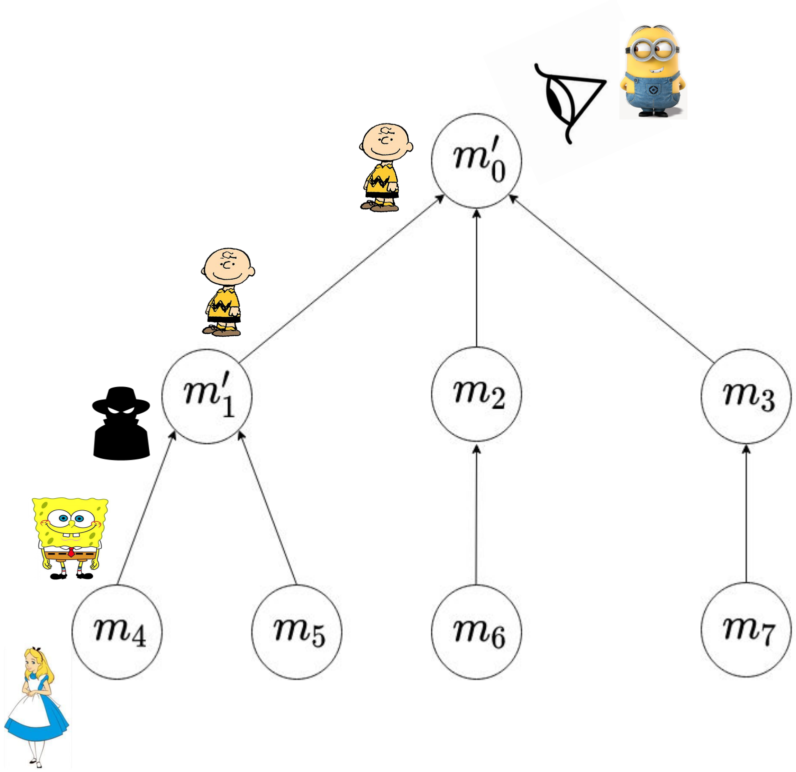


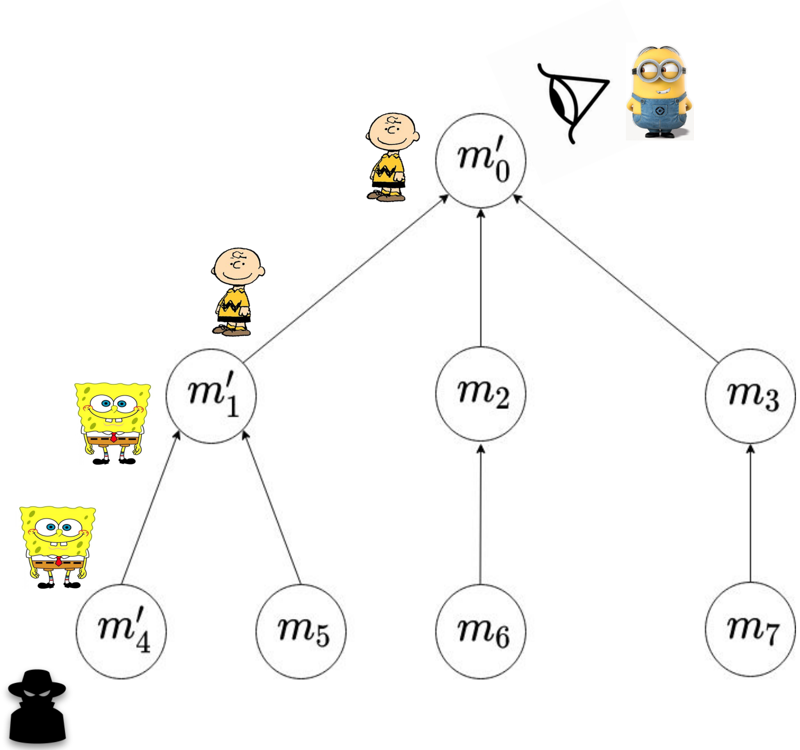
The Fault Detection Problem

Andreas Haeberlen[1] and Petr Kuznetsov[2]

[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] TU Berlin / Deutsche Telekom Laboratories

bit-complexity skyrockets !

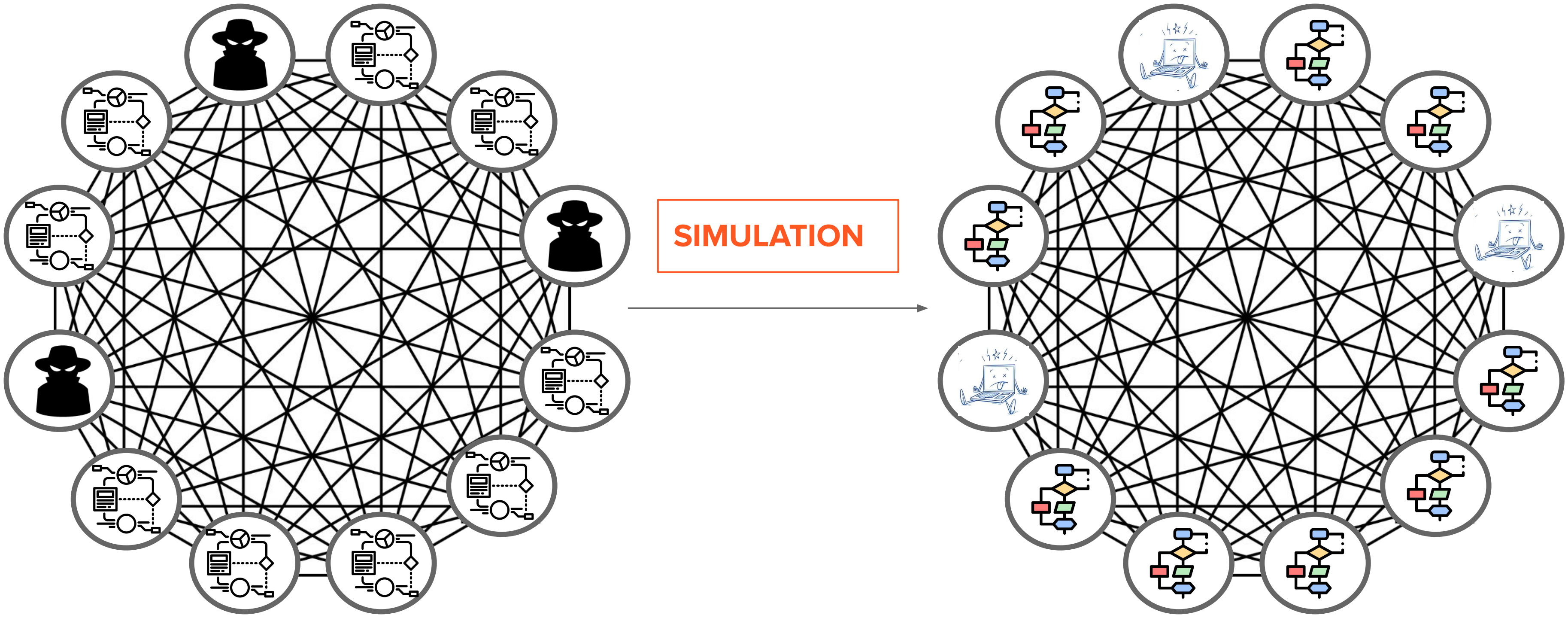




The end of accountability ?

No: Reduction to detection of directly observable equivocations

Simulation


SIMULATION

INFORMATION AND COMPUTATION 75, 130–143 (1987)

Asynchronous Byzantine Agreement Protocols

GABRIEL BRACHA

13Bart Street, Tel-Aviv 69104, Israel

INFORMATION AND COMPUTATION 75, 130–143 (1987)

Asynchronous Byzantine Agreement Protocols

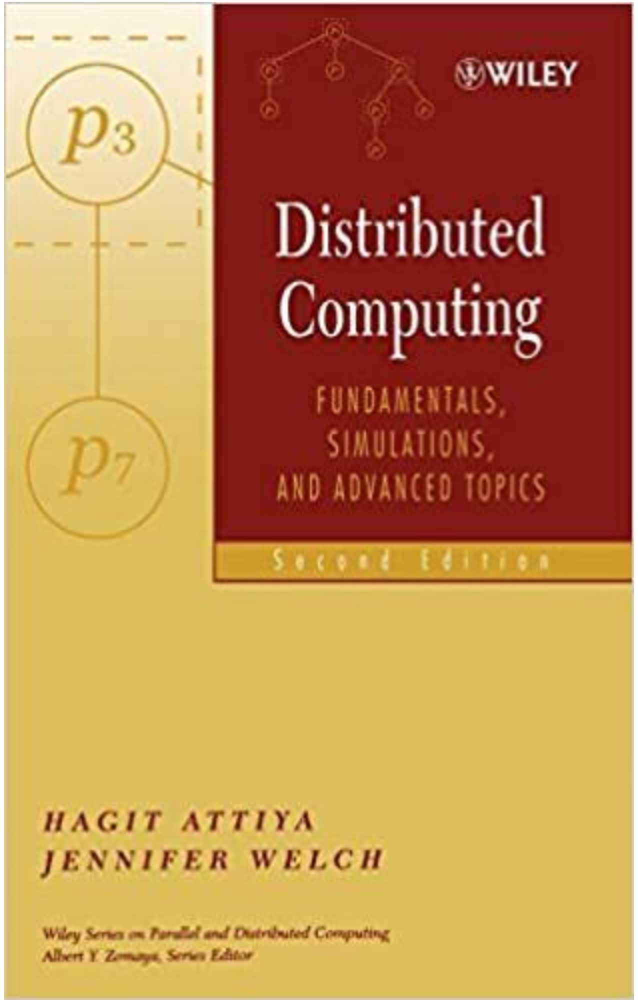
GABRIEL BRACHA

13Bart Street, Tel-Aviv 69104, Israel

IEEE TRANSACTIONS ON COMPUTERS, VOL. 37, NO. 12, DECEMBER 1988 1541

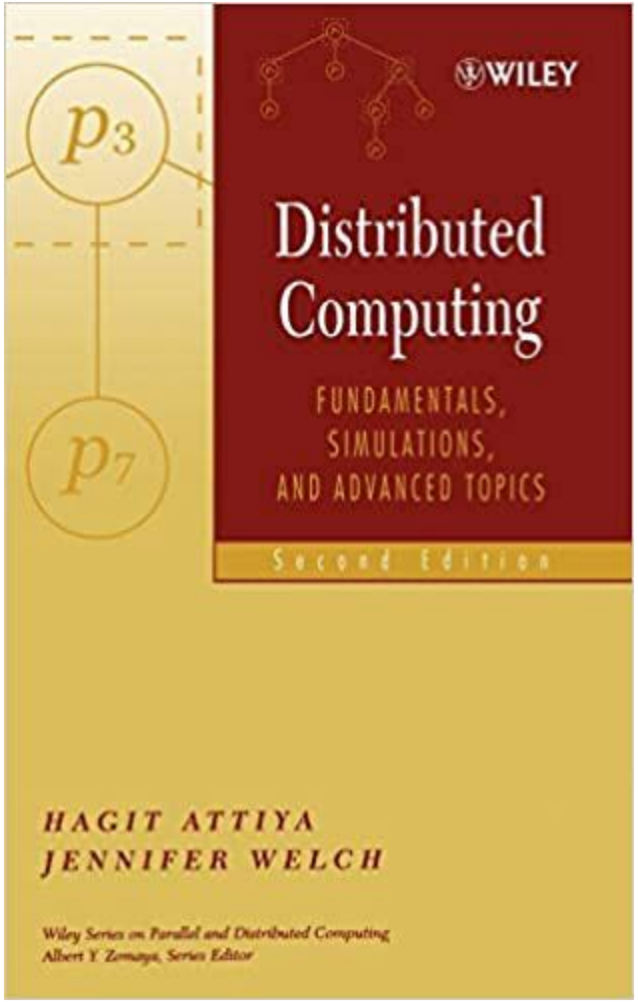
A Compiler that Increases the Fault Tolerance of Asynchronous Protocols

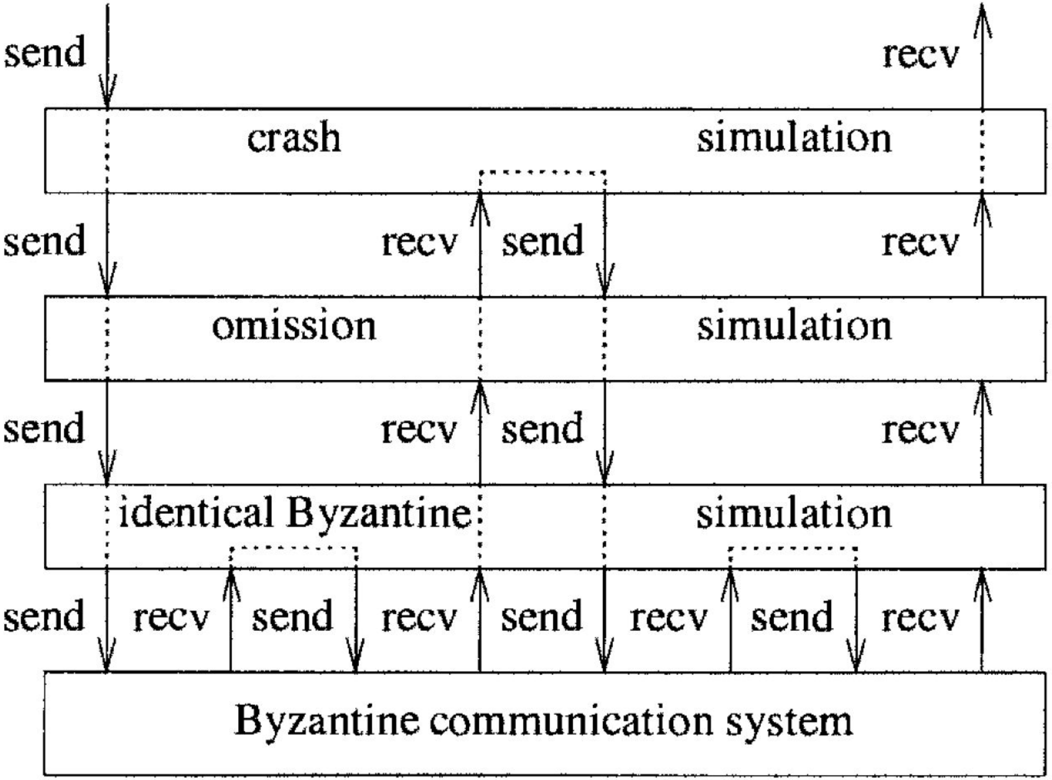
BRIAN A. COAN

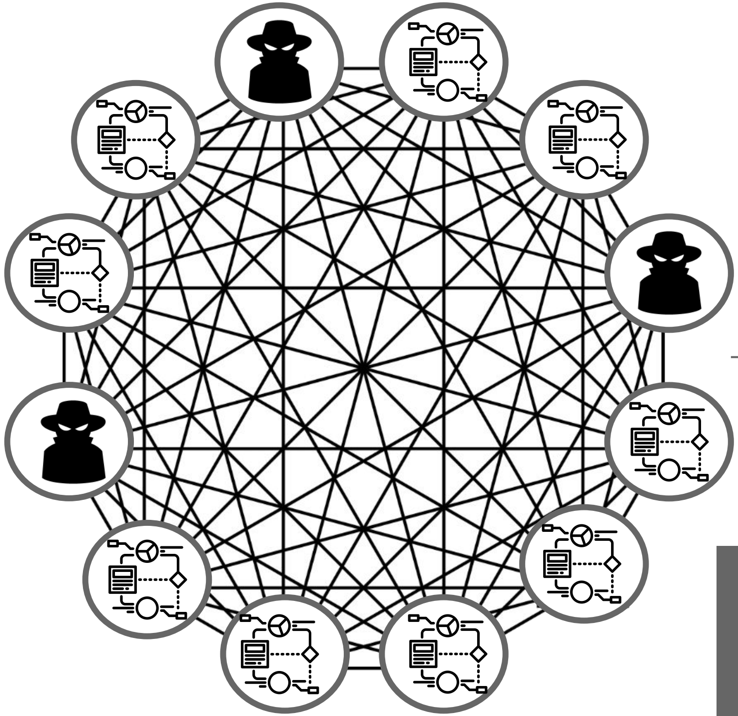


12

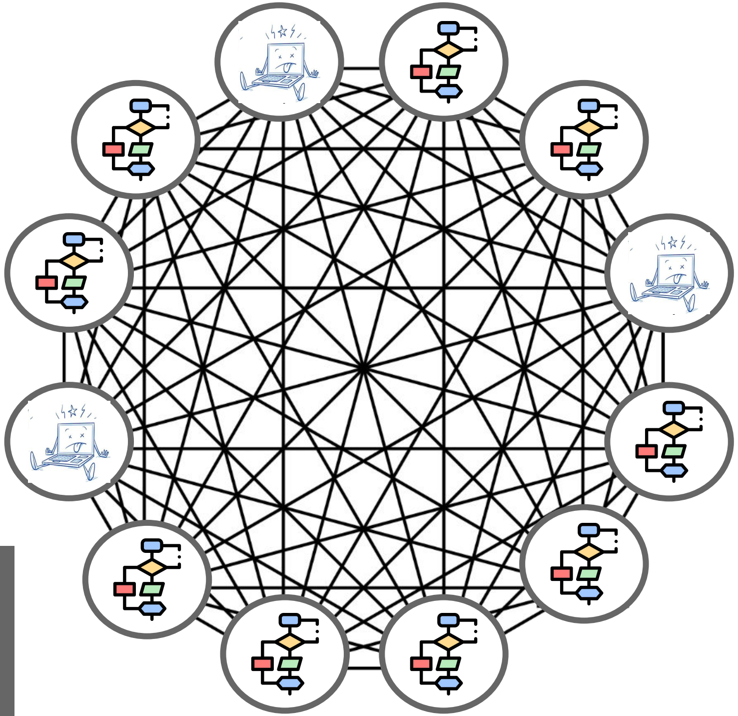
Improving the Fault Tolerance of Algorithms


WILEY
Distributed Computing
FUNDAMENTALS, SIMULATIONS, AND ADVANCED TOPICS
Second Edition
HAGIT ATTIYA
JENNIFER WELCH
Wiley Series on Parallel and Distributed Computing
Albert Y. Zomaya, Series Editor


send
recv
crash
simulation
send
recv
send
recv
omission
simulation
send
recv
send
recv
identical Byzantine
simulation
send
recv
send
recv
send
recv
send
recv
Byzantine communication system

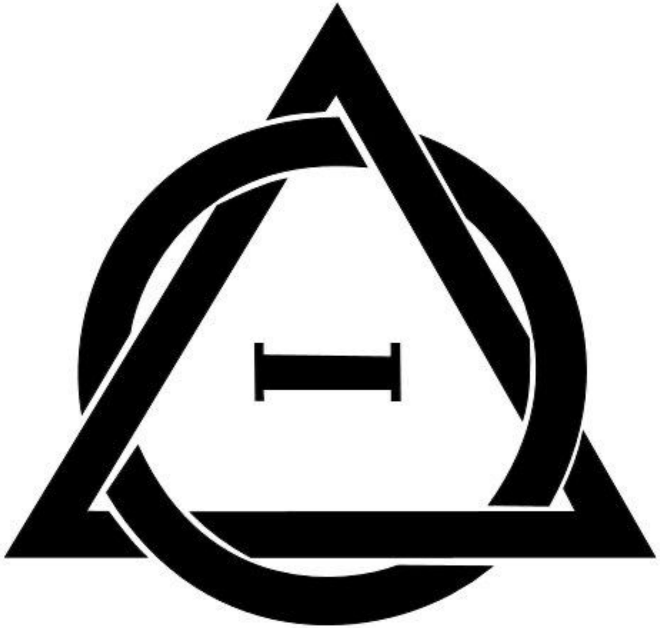


SIMULATION

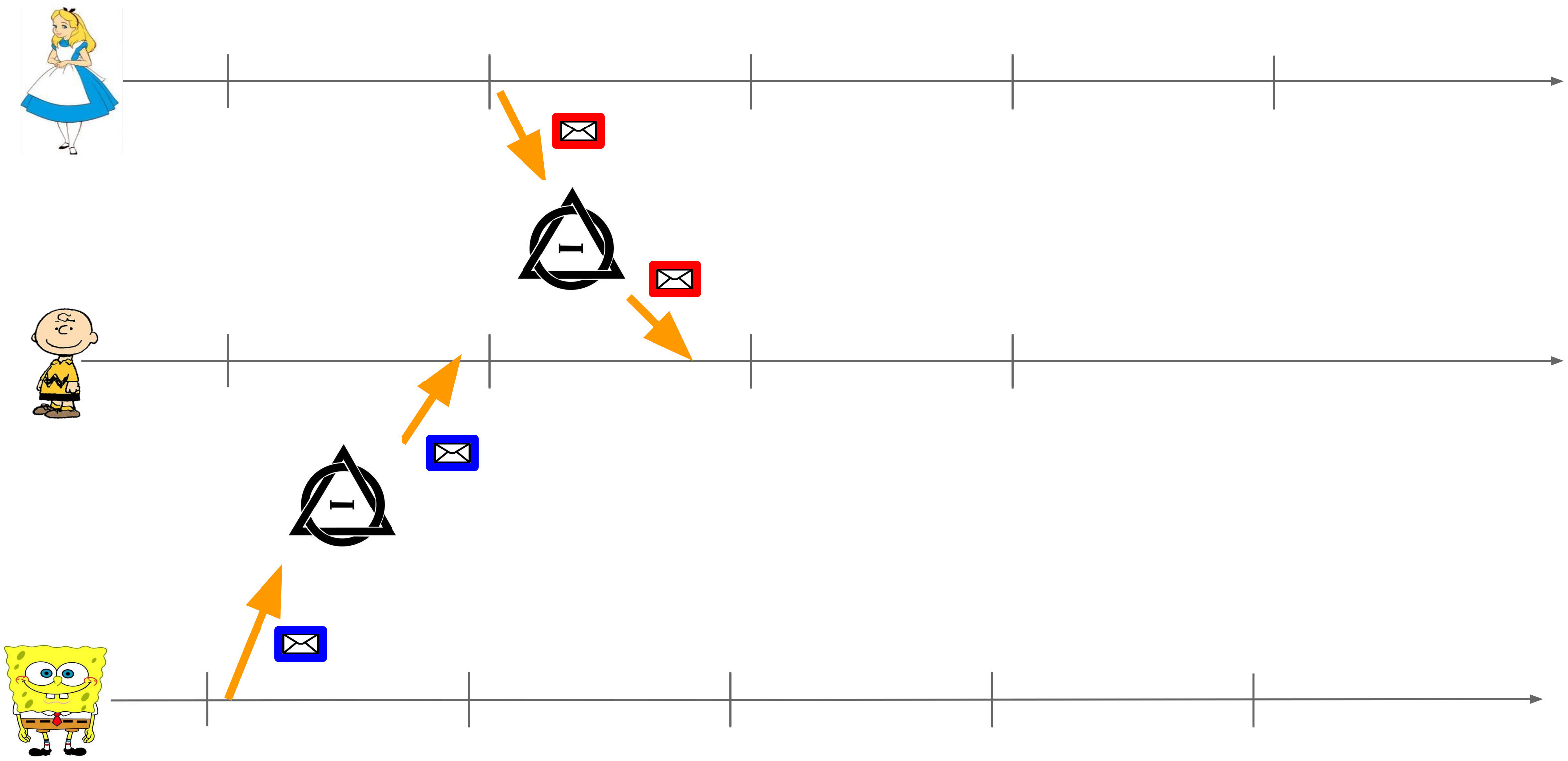


- Rb-bcast based simulation
- t<n/3
- O(n^2) bits

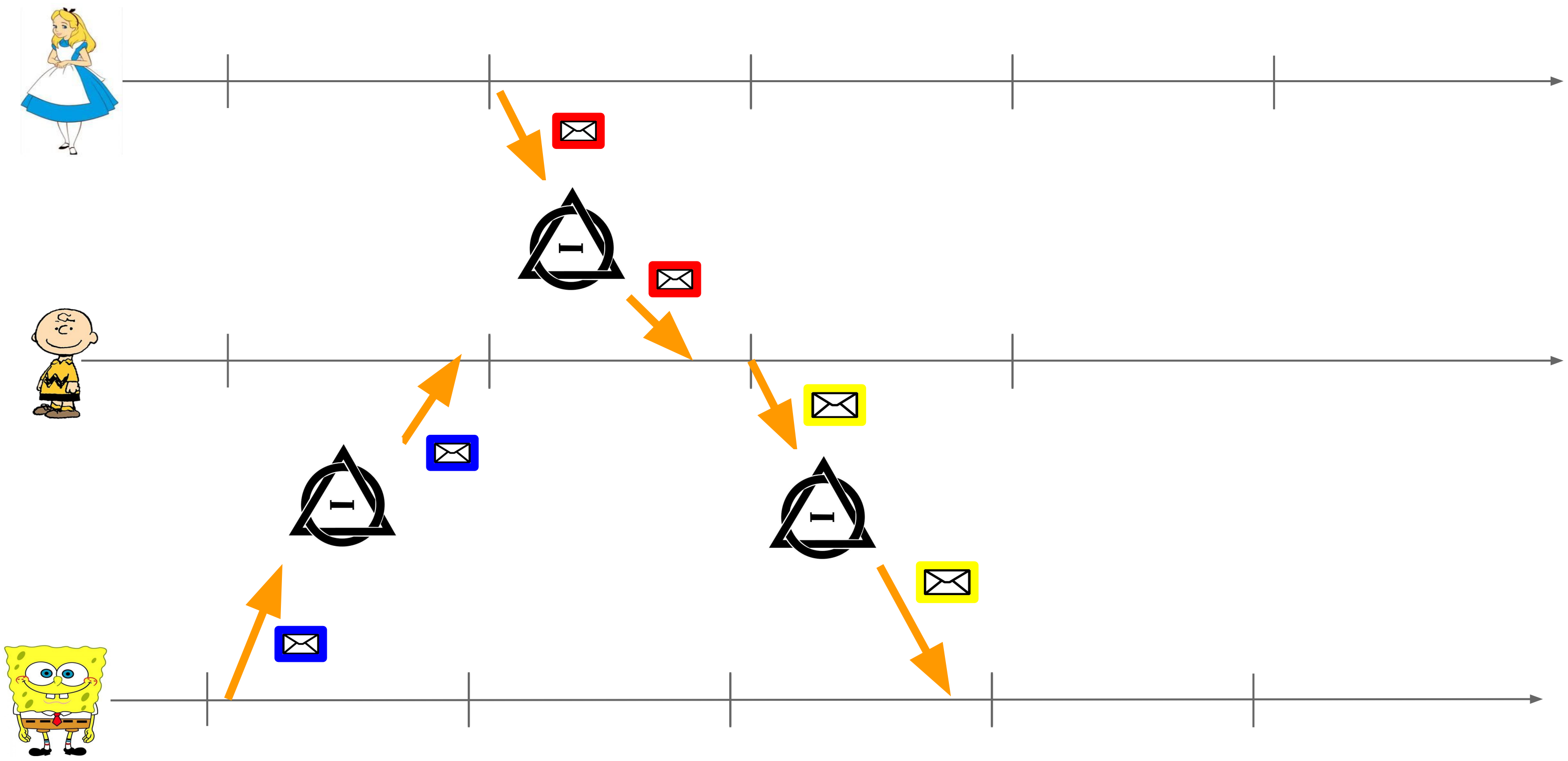
FullReview



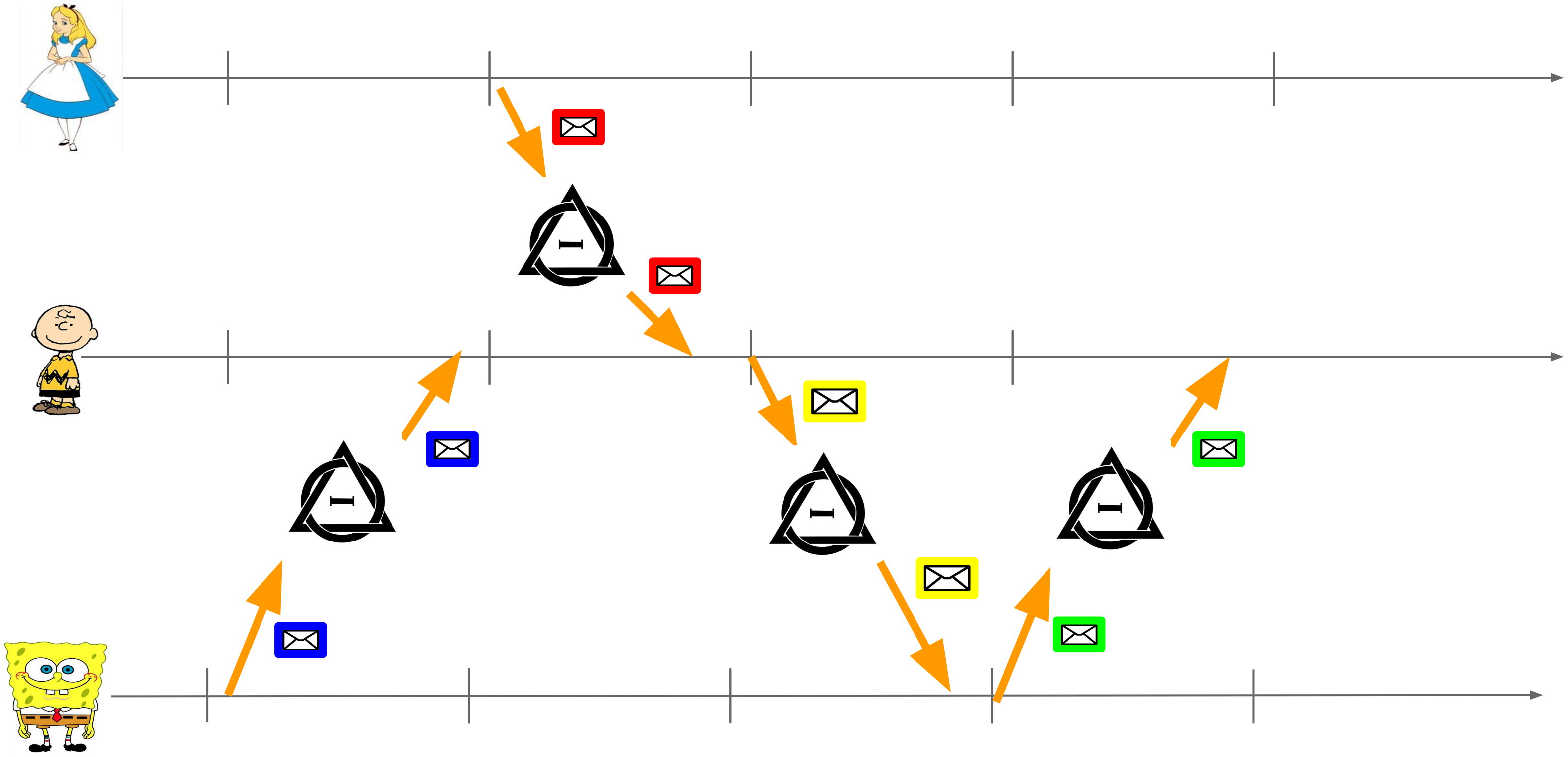
FullReview



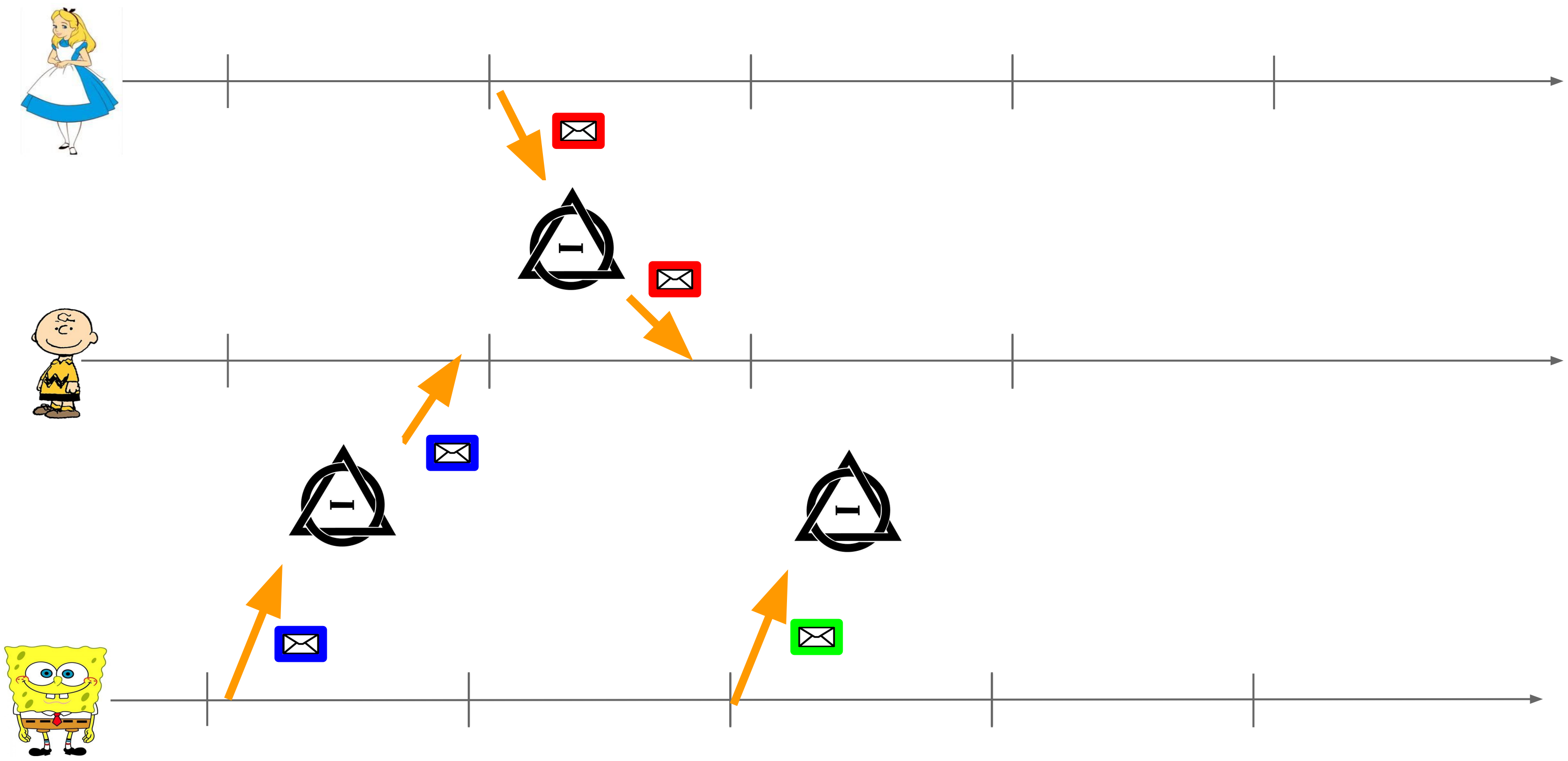
FullReview



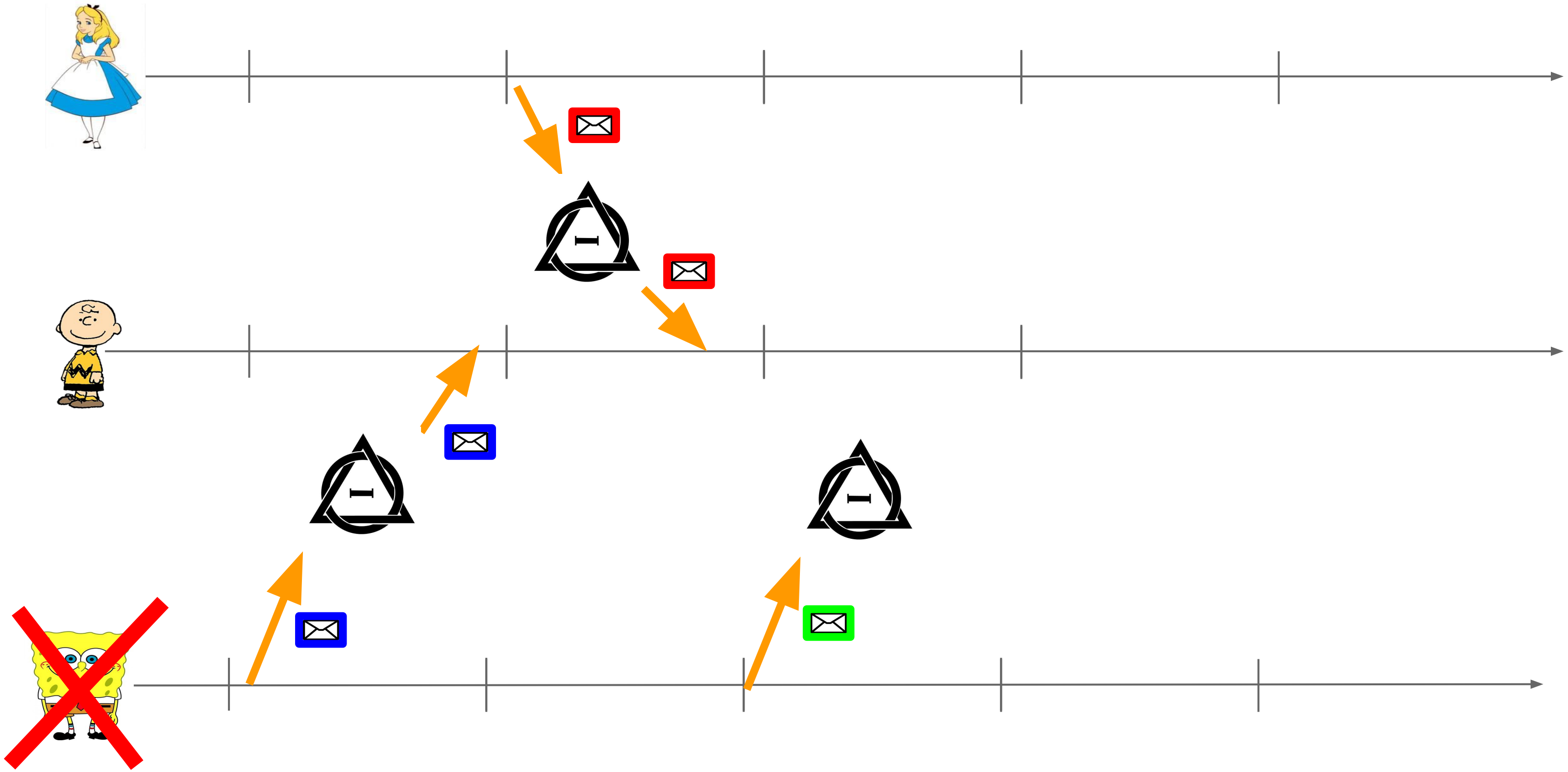
FullReview



FullReview



FullReview



FullReview Implementation

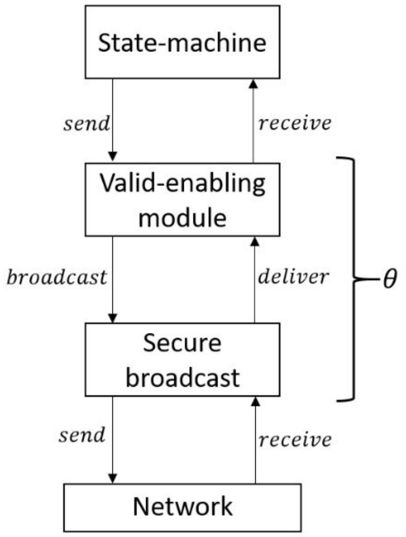


Fig. 1. Overview of the τ_{scr} transformation

Reliable-Broadcast

Interface

Broadcast
Operation

Deliver
Callback

Interface

Broadcast
Operation

Deliver
Callback

Interface

Broadcast
Operation

Deliver
Callback

Validity + Consistency + Totality

If the sender is correct, every correct process delivers its message. Either every correct process delivers the same message, or no correct process delivers any message.

Validity + Consistency + Totality

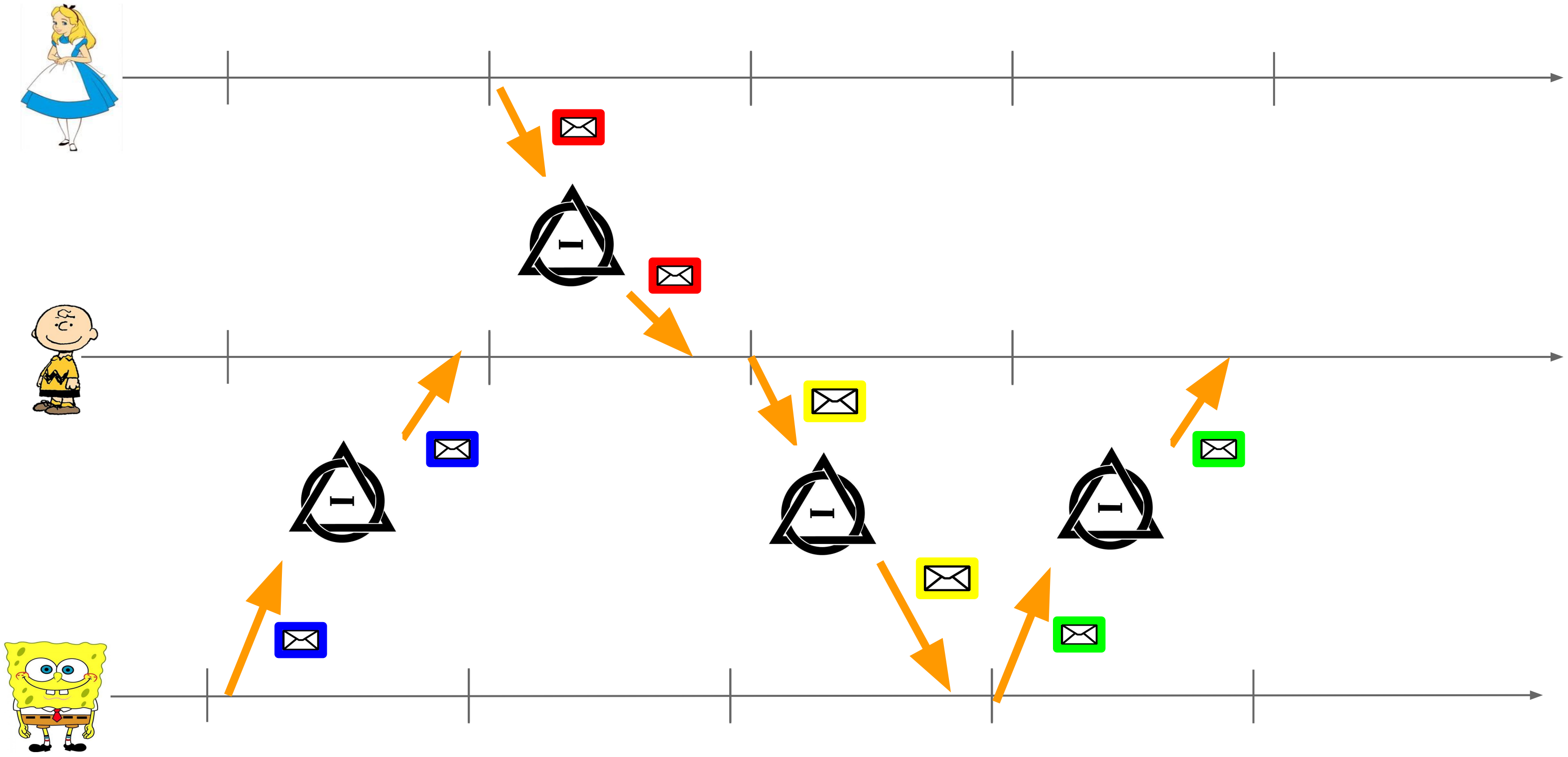
If the sender is correct, every correct process delivers its message. Either every correct process delivers the same message, or no correct process delivers any message.

Secure-Broadcast

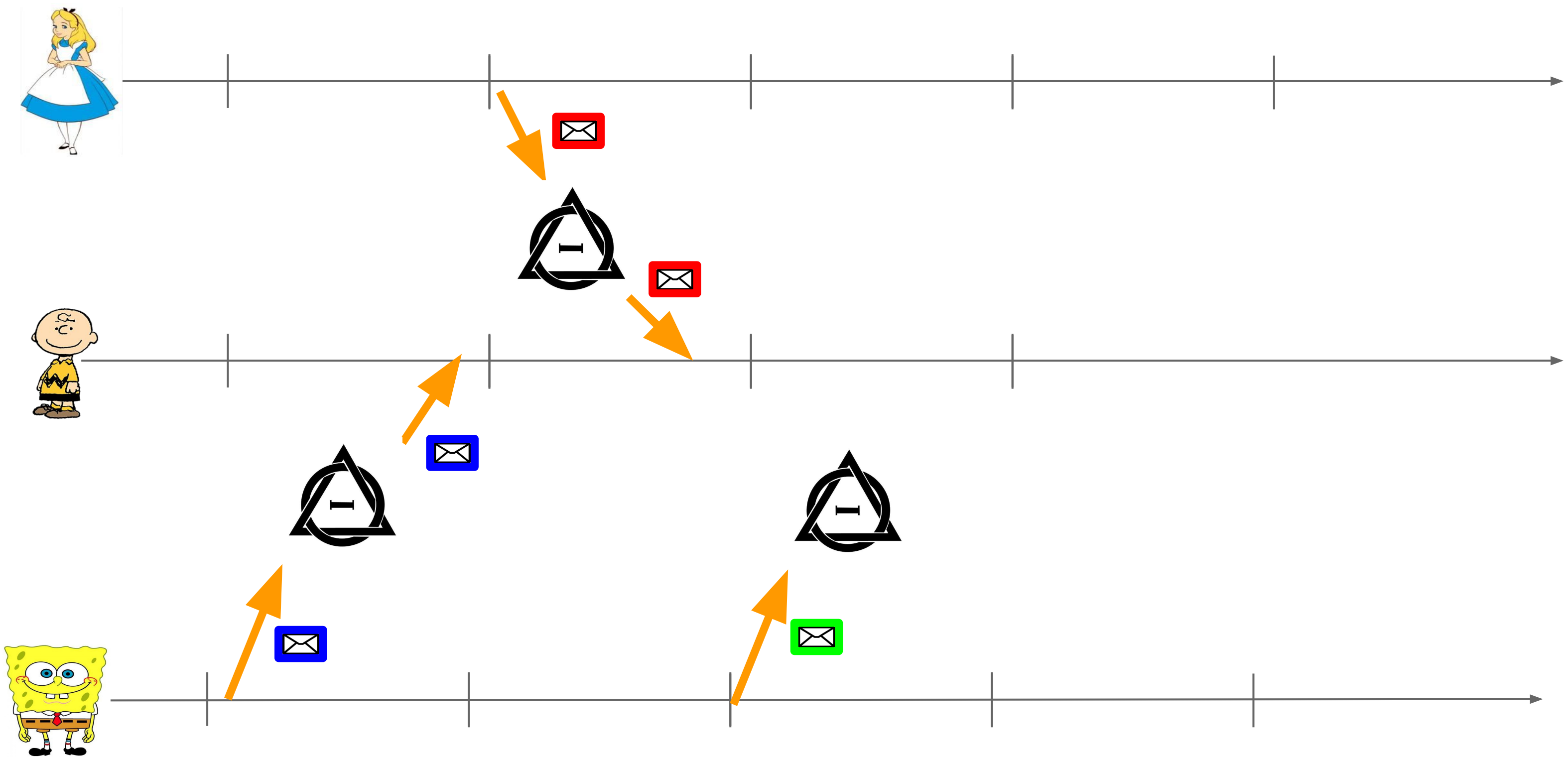
secure-broadcast = multishor RB-broadcast

- **Integrity:** a correct process executes *deliver*(p, m) at most once, and, in case the sender process p is benign, only if p called *broadcast*(m).
- **Agreement:** if p and q are correct and p executes *deliver*(r, m), then q eventually executes *deliver*(r, m).
- **Validity:** if p is correct and executes *broadcast*(m), then p eventually executes *deliver*(p, m).
- **Source Order:** if p and q are benign and p executes *deliver*(r, m) before *deliver*(r, m'), then q does not execute *deliver*(r, m') before executing *deliver*(r, m). Moreover, if r is benign and broadcasts m and afterwards broadcasts m', then no benign process delivers these two messages in the opposite order.

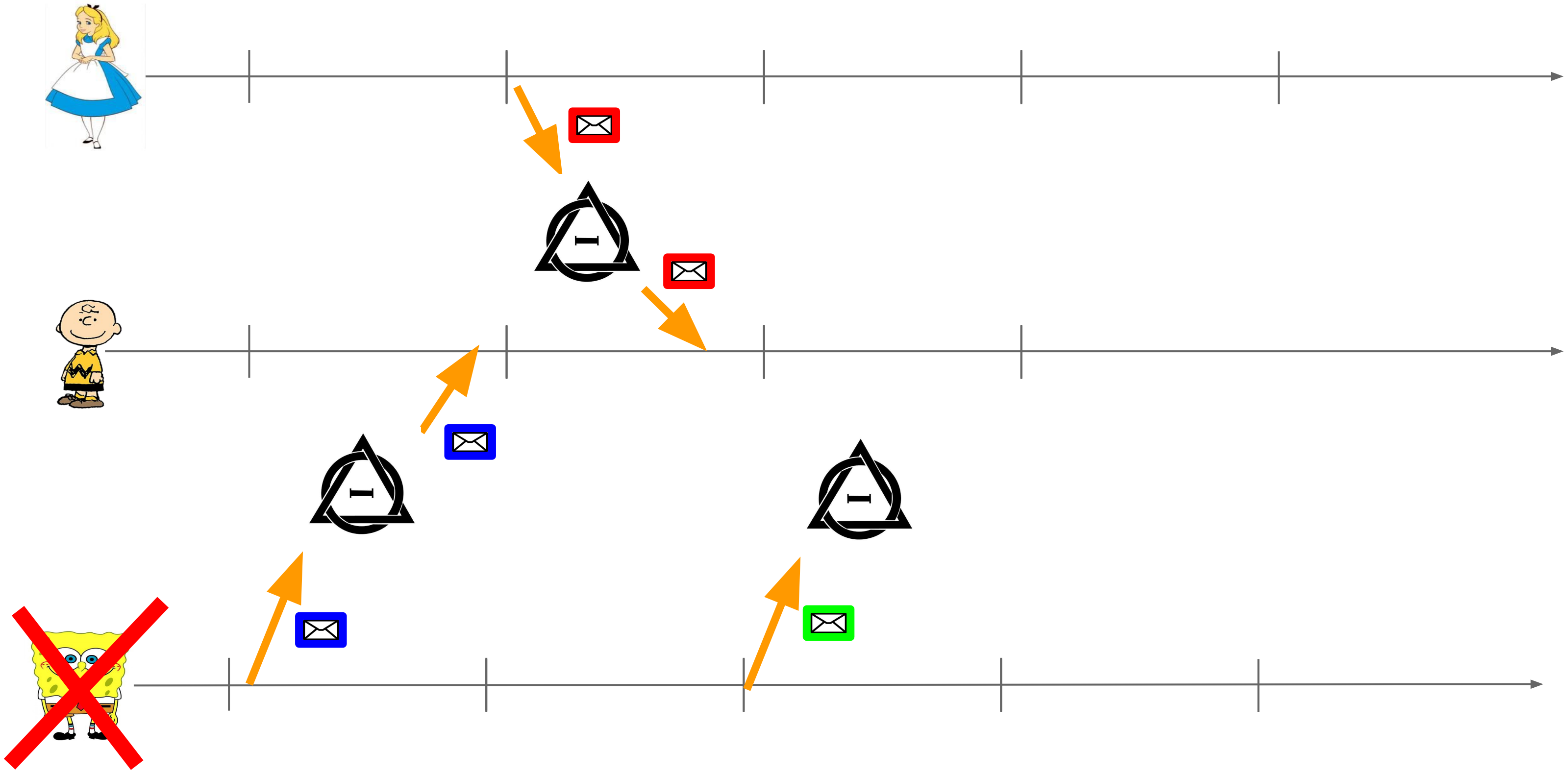
FullReview



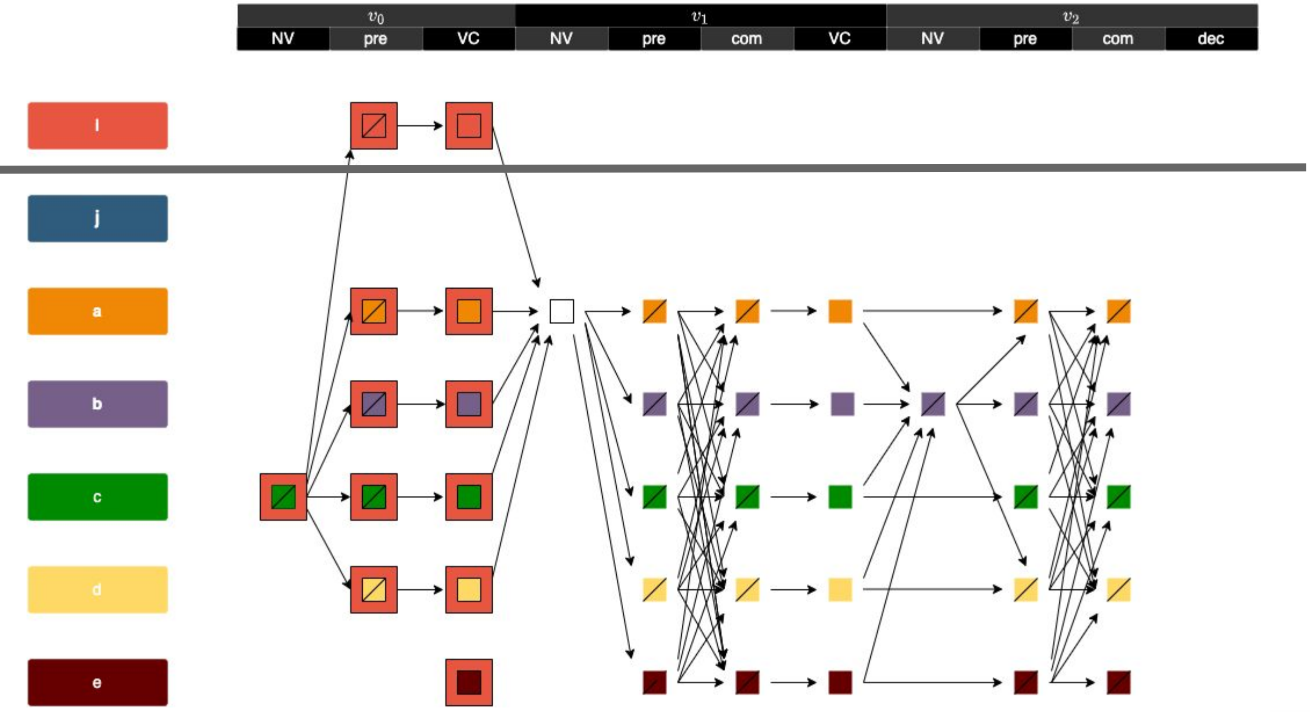
FullReview



FullReview



Returning to Accountability



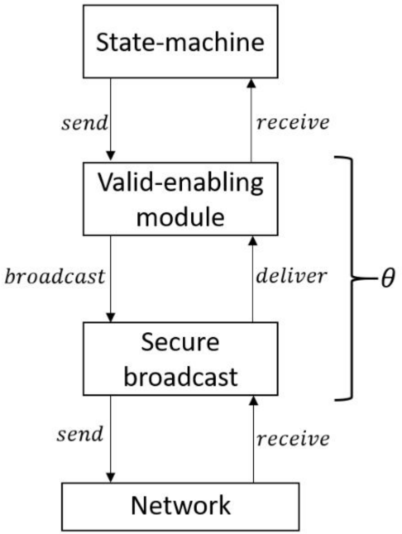
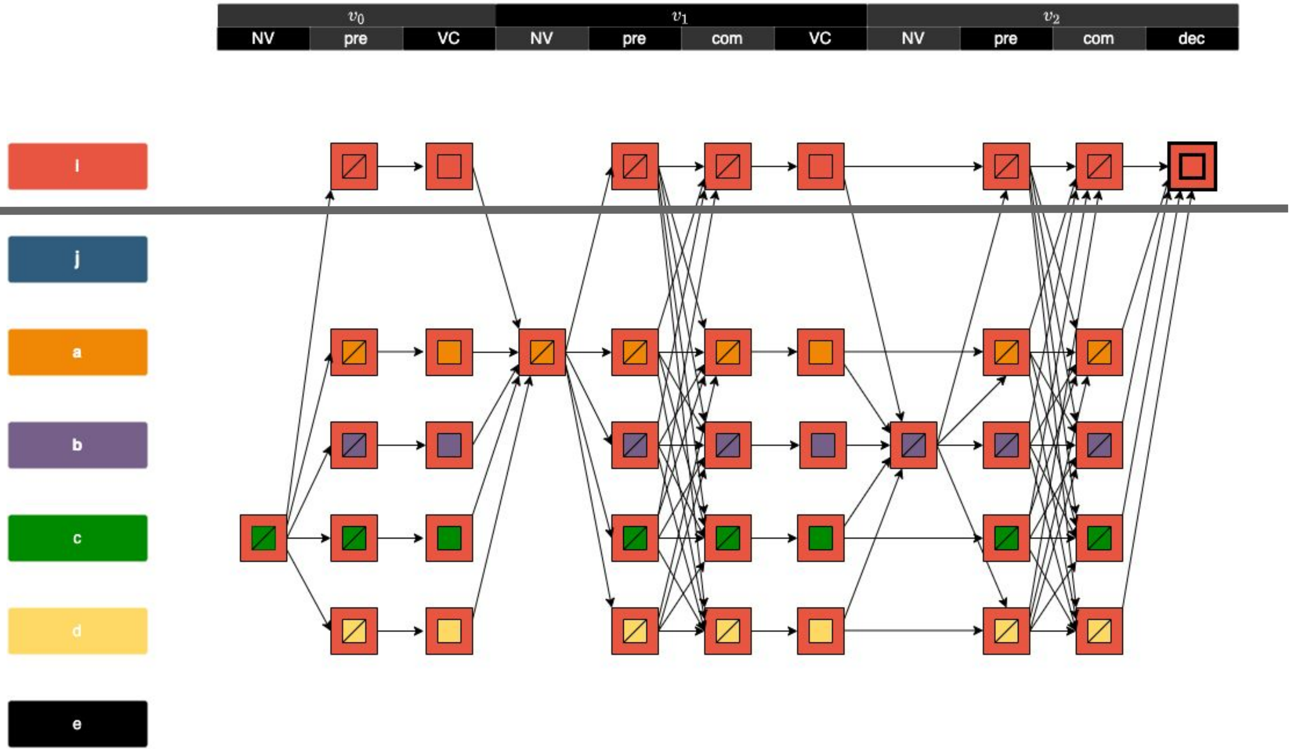


Fig. 1. Overview of the τ_{scr} transformation



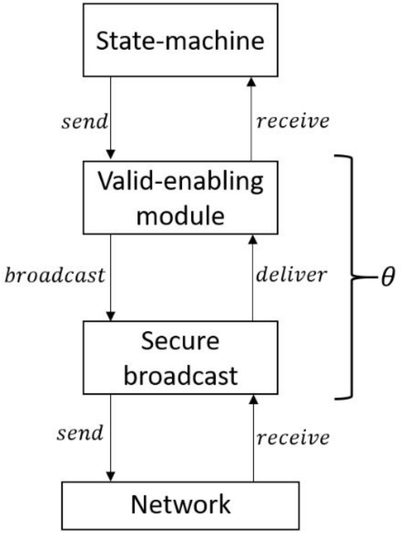
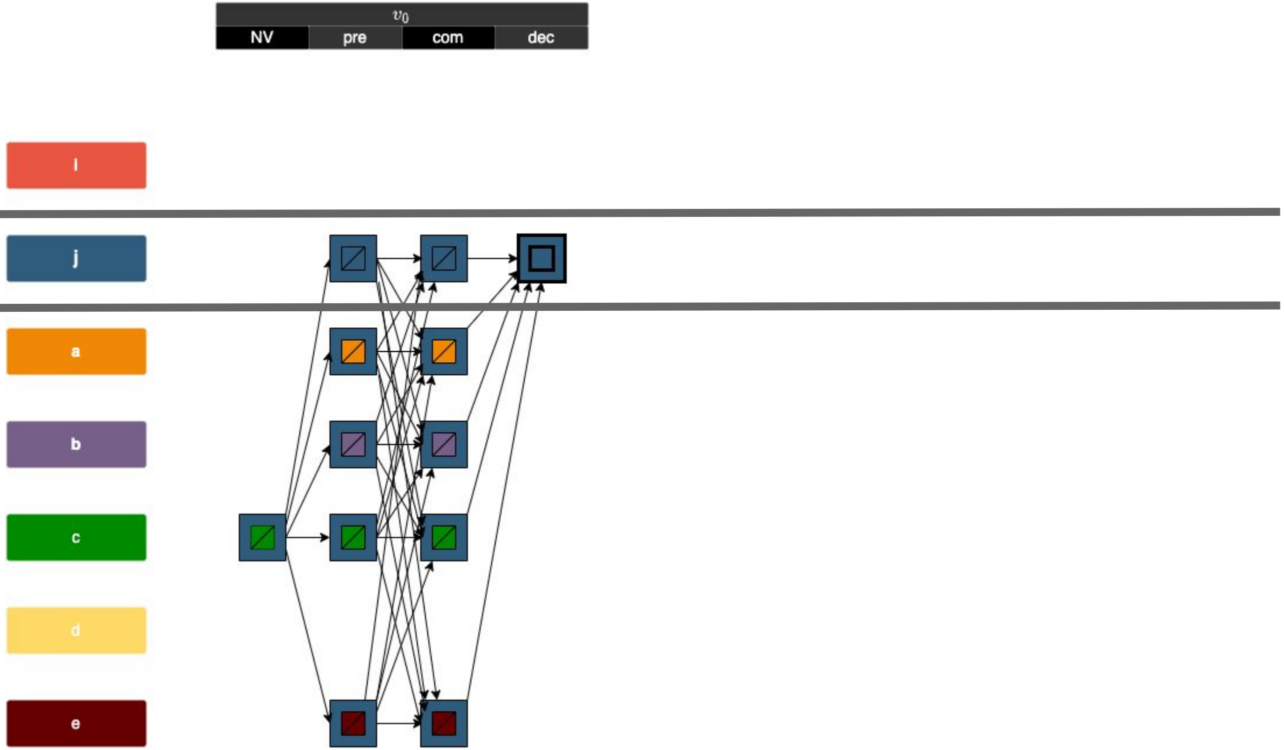


Fig. 1. Overview of the τ_{scr} transformation



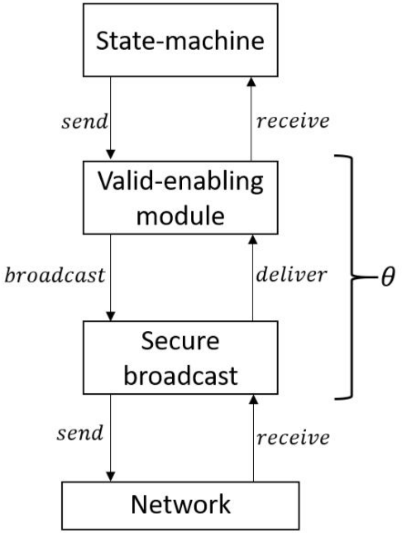
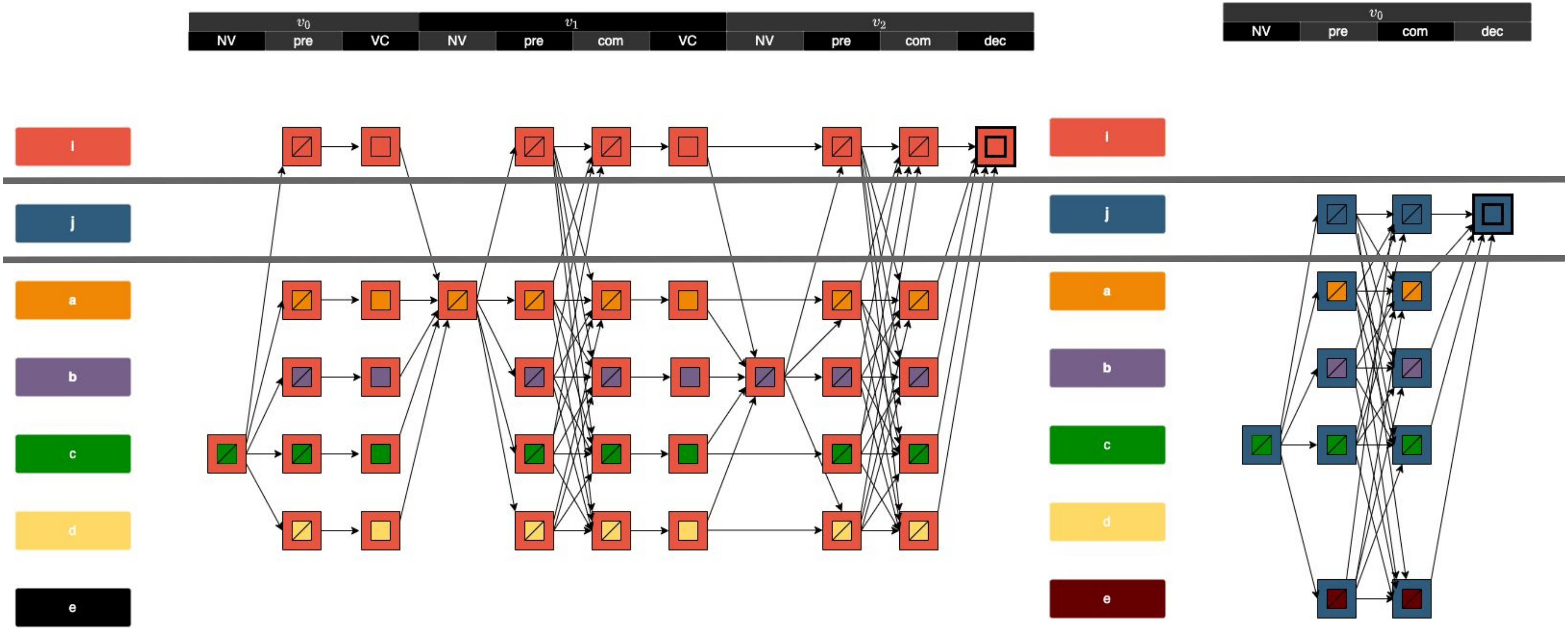
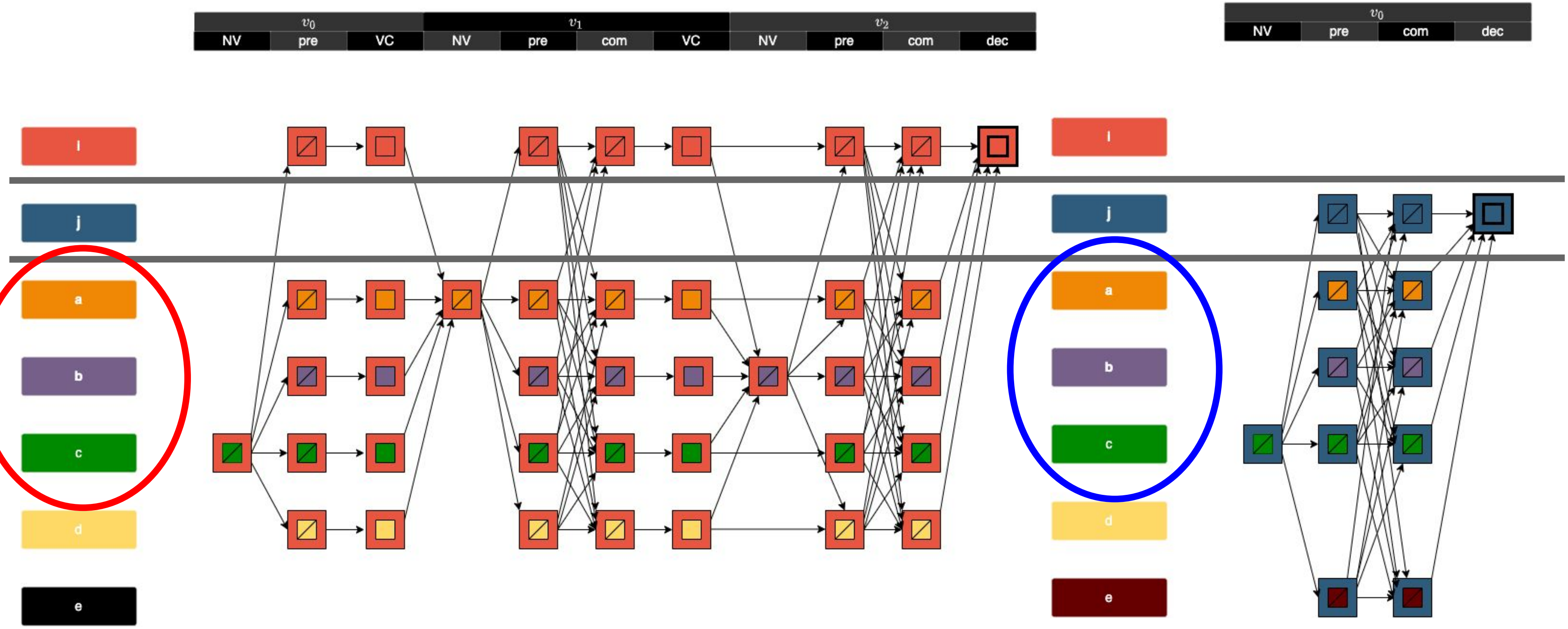


Fig. 1. Overview of the τ_{scr} transformation

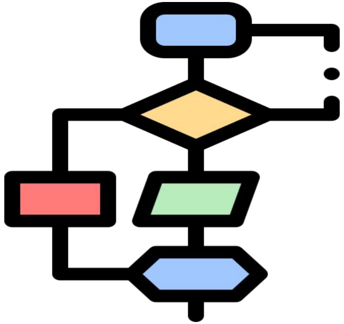


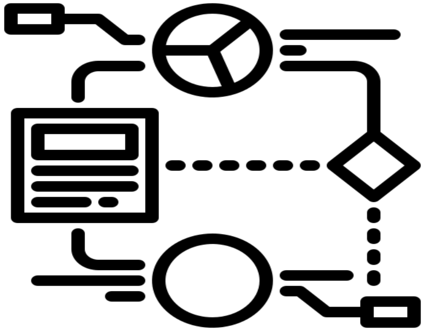
v_0
NV
pre
VC
v_1
NV
pre
com
VC
v_2
NV
pre
com
dec
l
j
a
b
c
d
e
v_0
NV
pre
com
dec
l
j
a
b
c
d
e



v_0
v_1
v_2
NV
pre
VC
NV
pre
com
VC
NV
pre
com
dec
l
j
a
b
c
d
e
l
j
a
b
c
d
e
v_0
NV
pre
com
dec

necessary and sufficient transformation





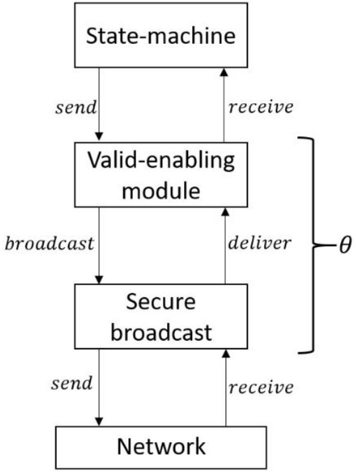


Fig. 1. Overview of the τ_{scr} transformation

Answers

What are the Byzantine faults to detect ? Commission = Equivocation + Evasion

Can they cause safety violation ? Commission are necessary

What is the cost to detect it ? Quadratic overhead in worst case

+

Can be applied to randomized protocols.

Can be applied to most of practical protocols that assume private channels

Can be applied to permissionless protocols

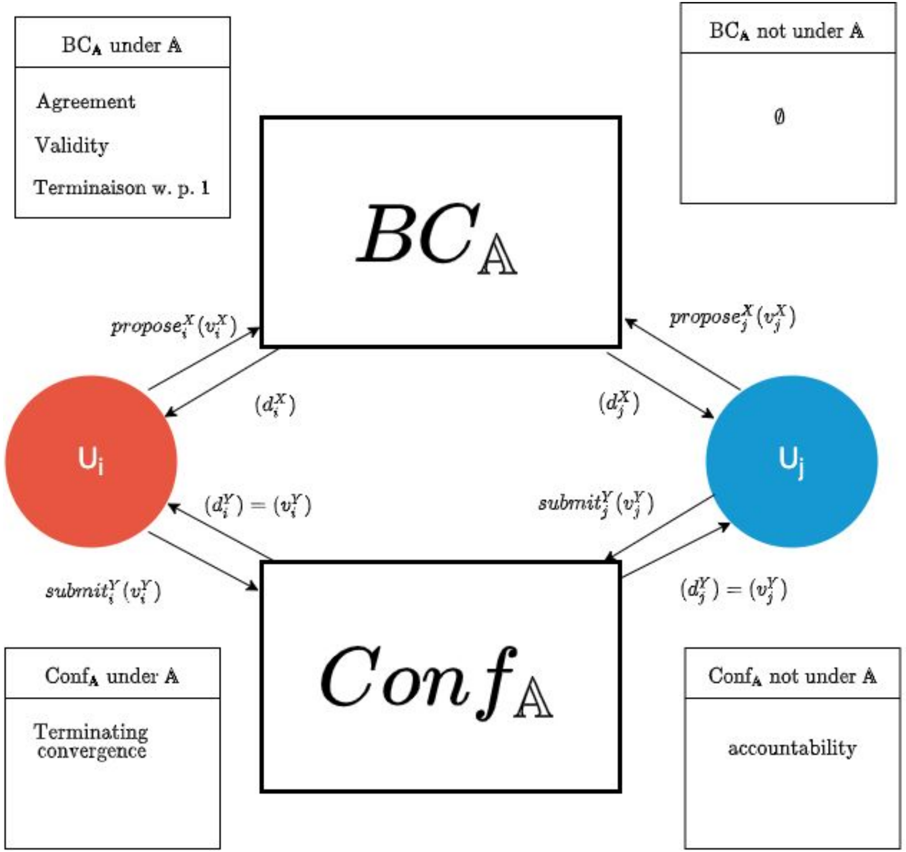
Can be applied to committee-based blockchains with fully corrupted committee

Cachin-Tessaro Optimization can be applied to heavy messages.

Conclusion

Particular Cases

Easy accountable task (consensus, RB-bcast, …)




```
function propose(v) do
    ▷ bc is any Byzantine consensus protocol
    v′ ← bc.propose(v)
    broadcast [CONFIRM, v′]
    wait for n − t0[CONFIRM, v′]
    return v′
```

particular cases

- Easy agreement tasks can be trivially made accountable (cf. “As easy as ABC (A)ccountable (B)yzantine (C)onsensus is easy!”).
- Only secure-broadcast critical sections.
- Use (randomized) scalable secure-broadcast with n.log(n) overhead and exchange scr-delivered messages with a certain probability only

Next ? Fully privacy-preserving accountability