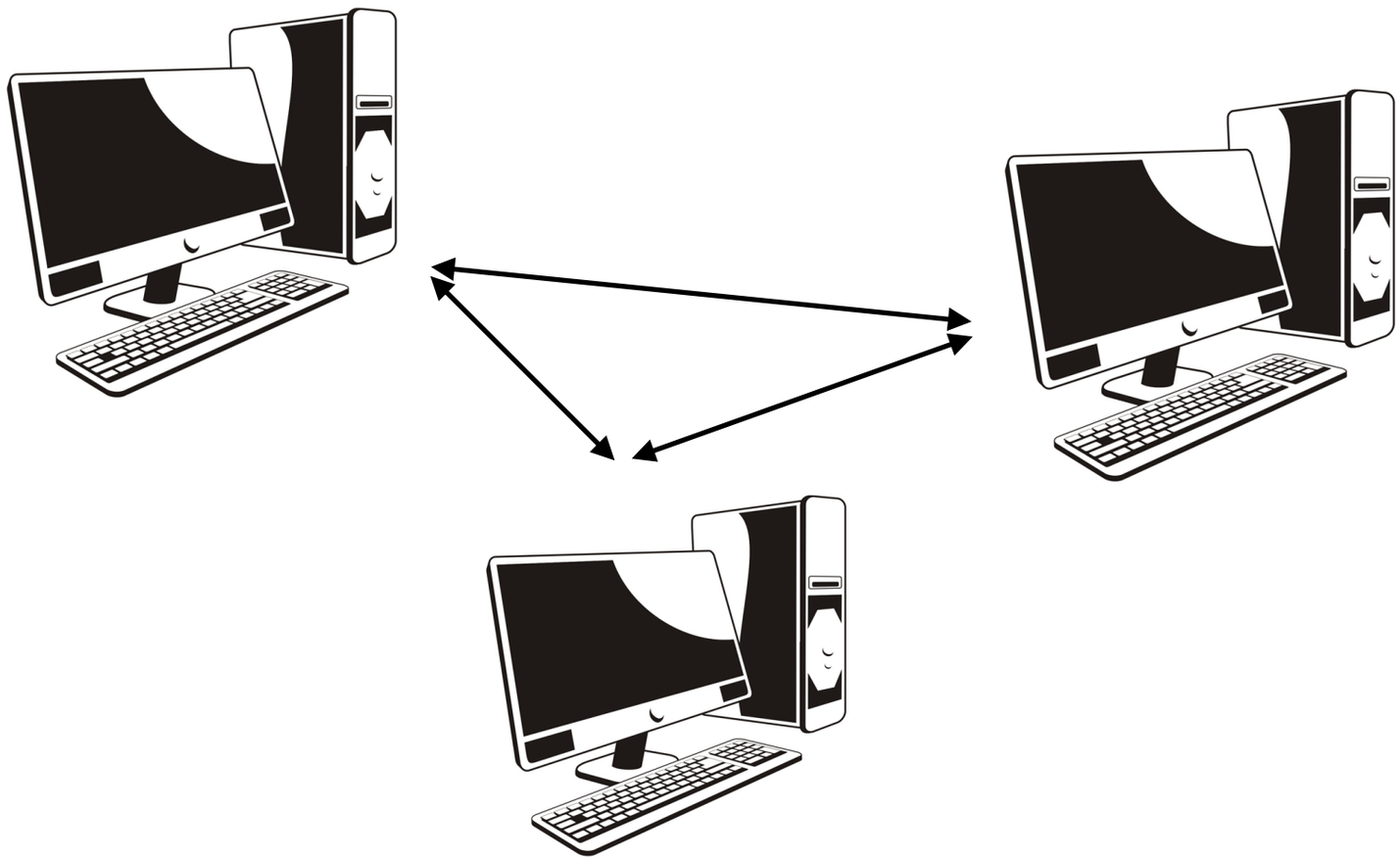
Distributed Systems

Byzantine Fault Tolerance and Consensus

Dragos-Adrian Seredinschi
lpd.epfl.ch

Problem

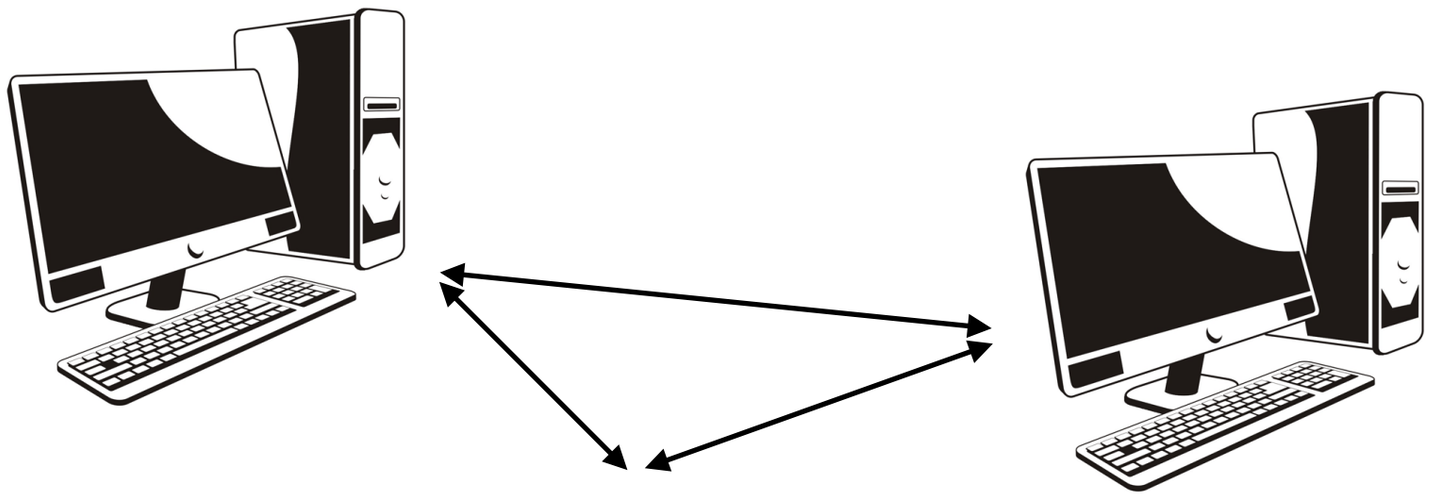
Correct process



General goal:
Run a distributed algorithm

Problem

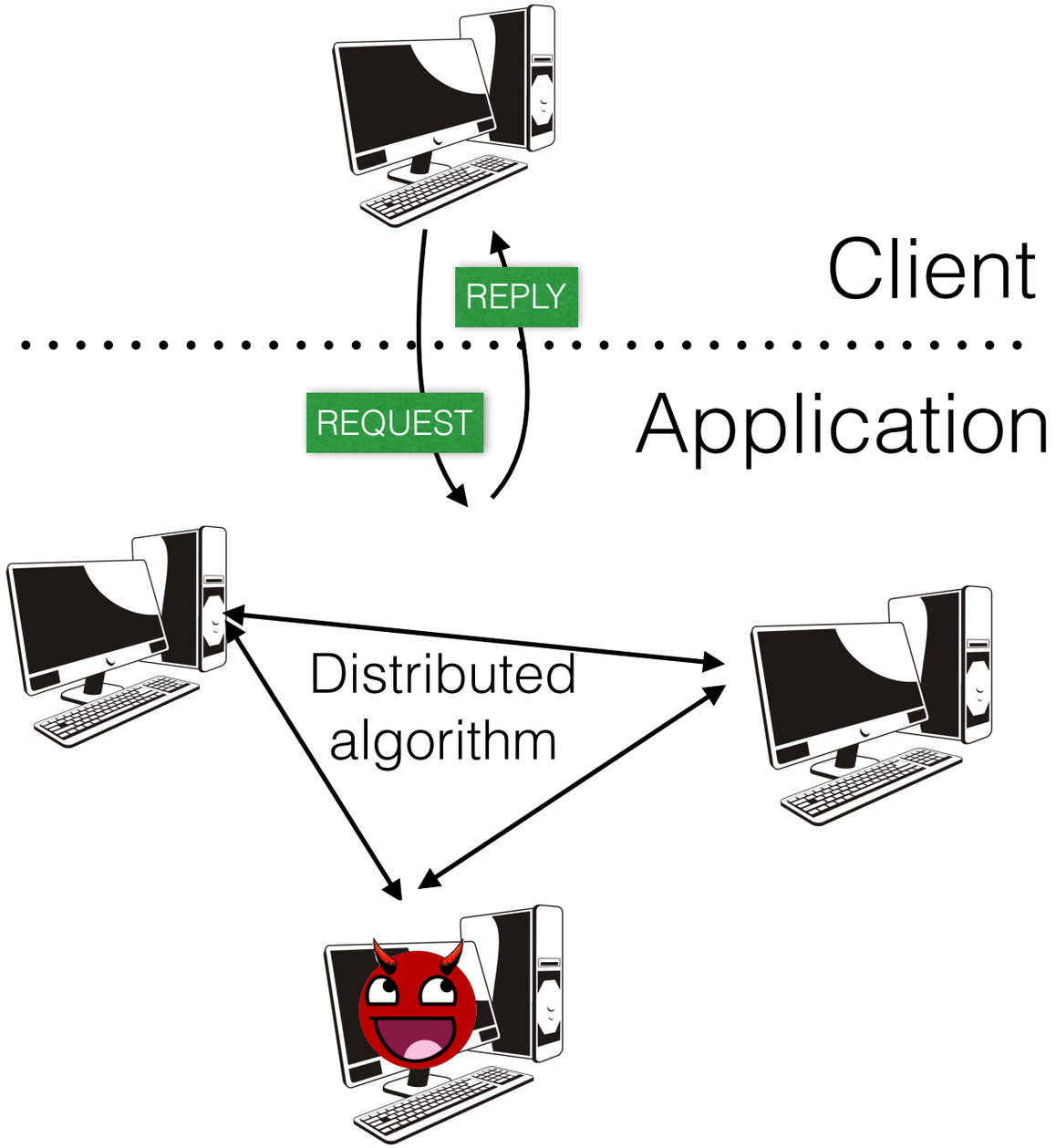
Correct process



Arbitrary failure

General goal:
Run a distributed algorithm

Recasting the problem

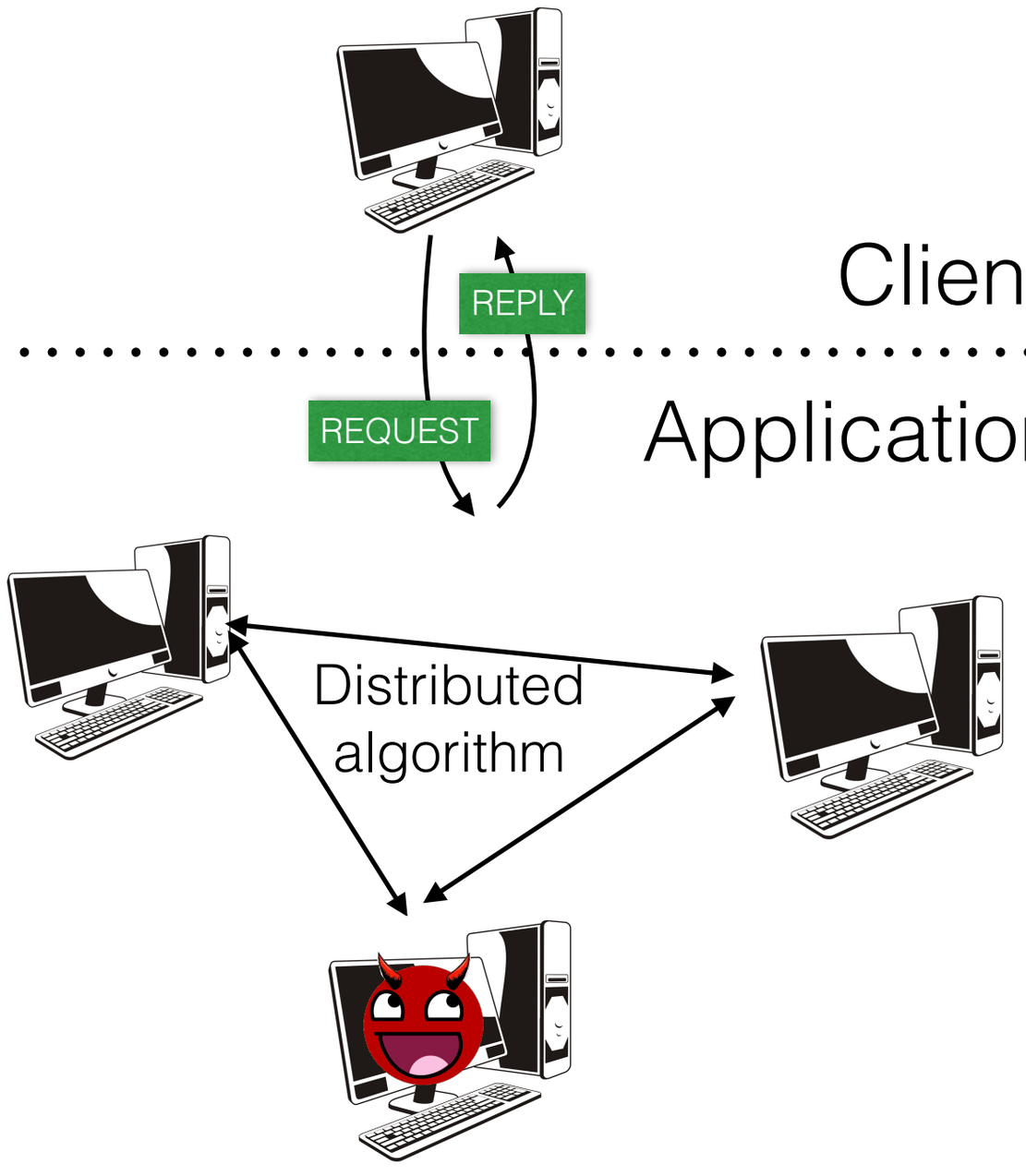


Recasting the problem

Application requirements:

- High-Availability
  (give a reply to a request)
- Reliability
  (give correct replies)

Boils down to fault-tolerance



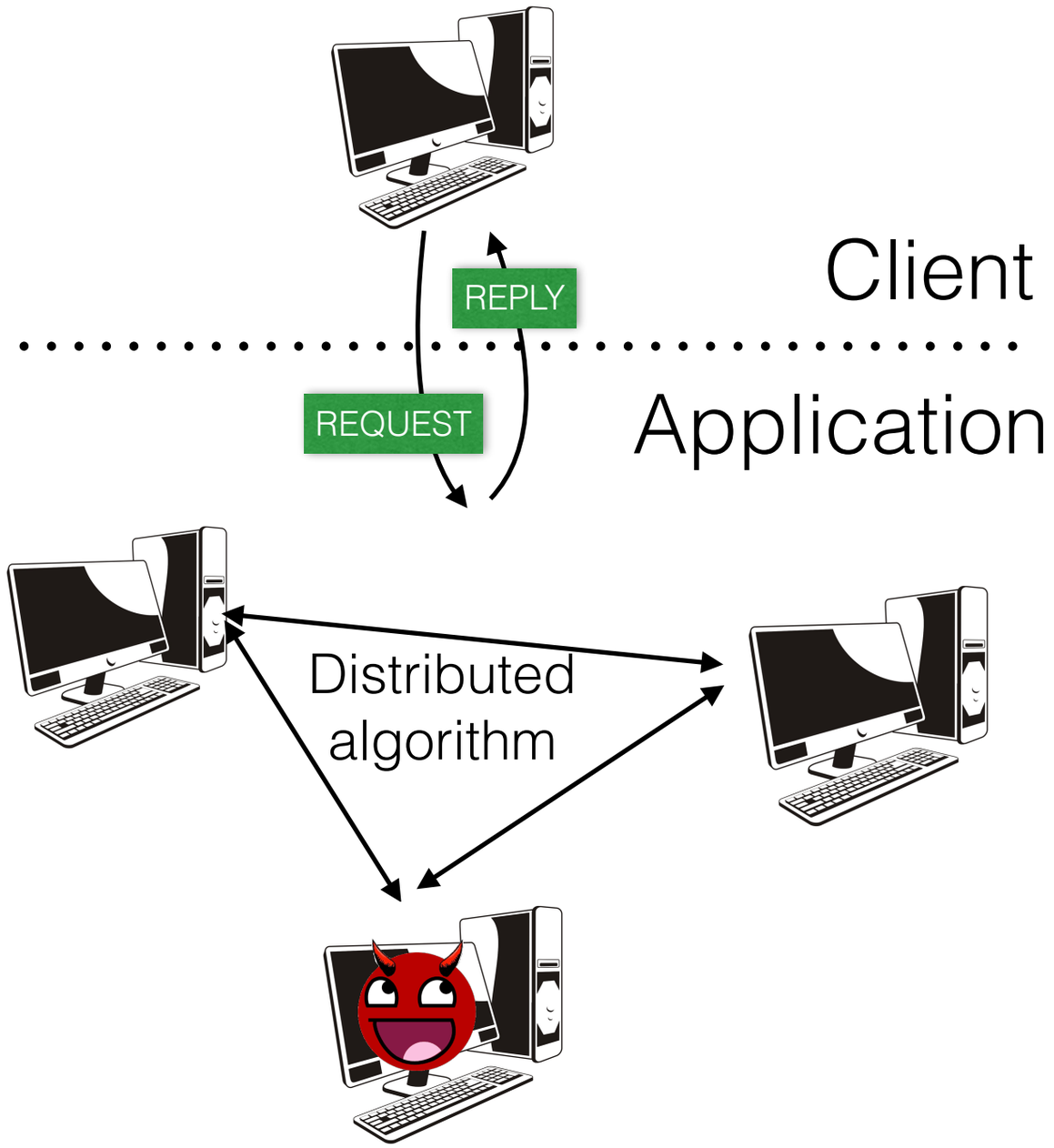
Solution

Fault-tolerance basic techniques:

- Agreement = Consensus
- Replication =
  State Machine Replication

In the following we will see…

- PBFT
- Seminal algorithm for
  Byzantine Fault Tolerance



PBFT

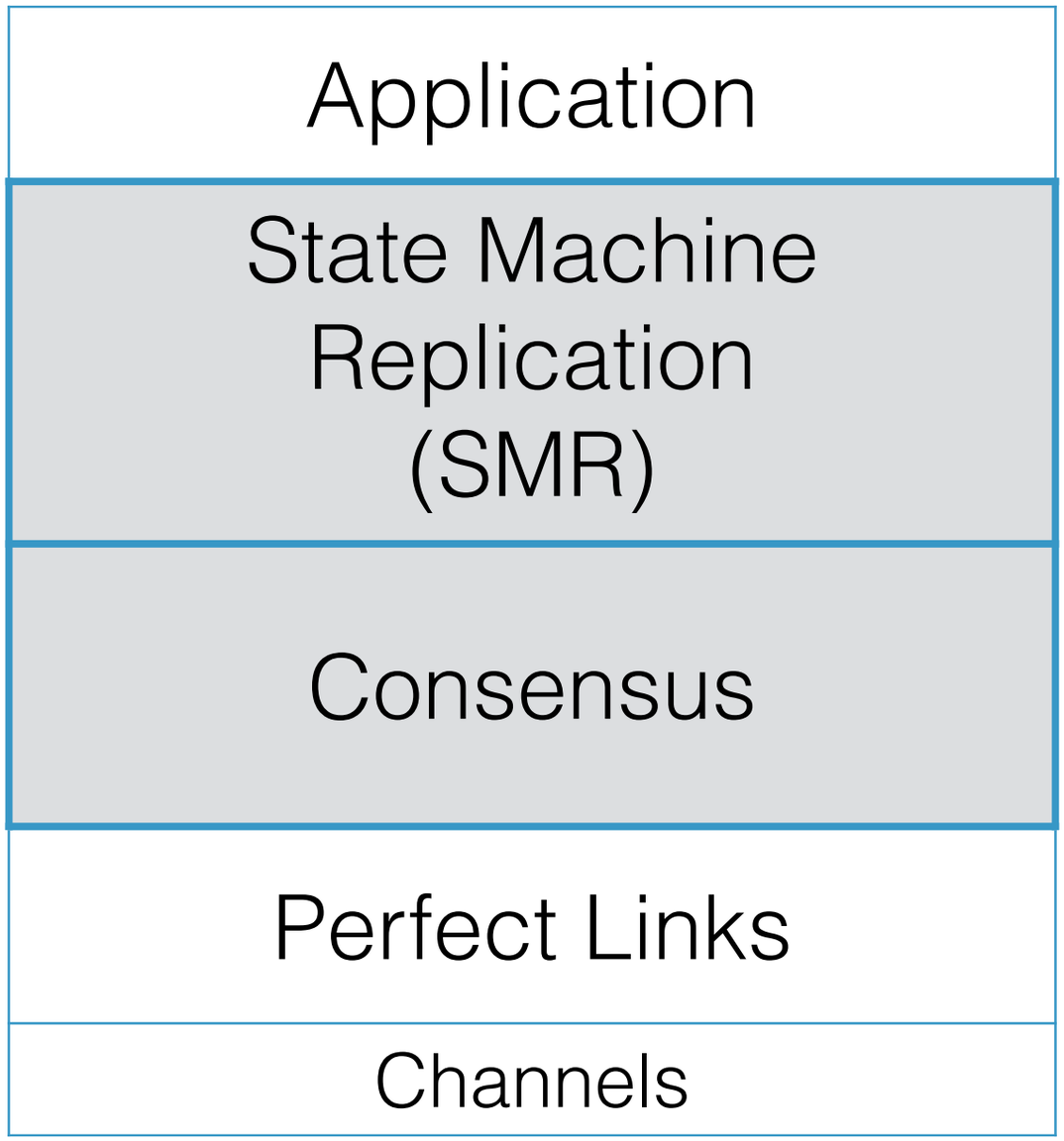
Practical Byzantine Fault Tolerance

OSDI'99

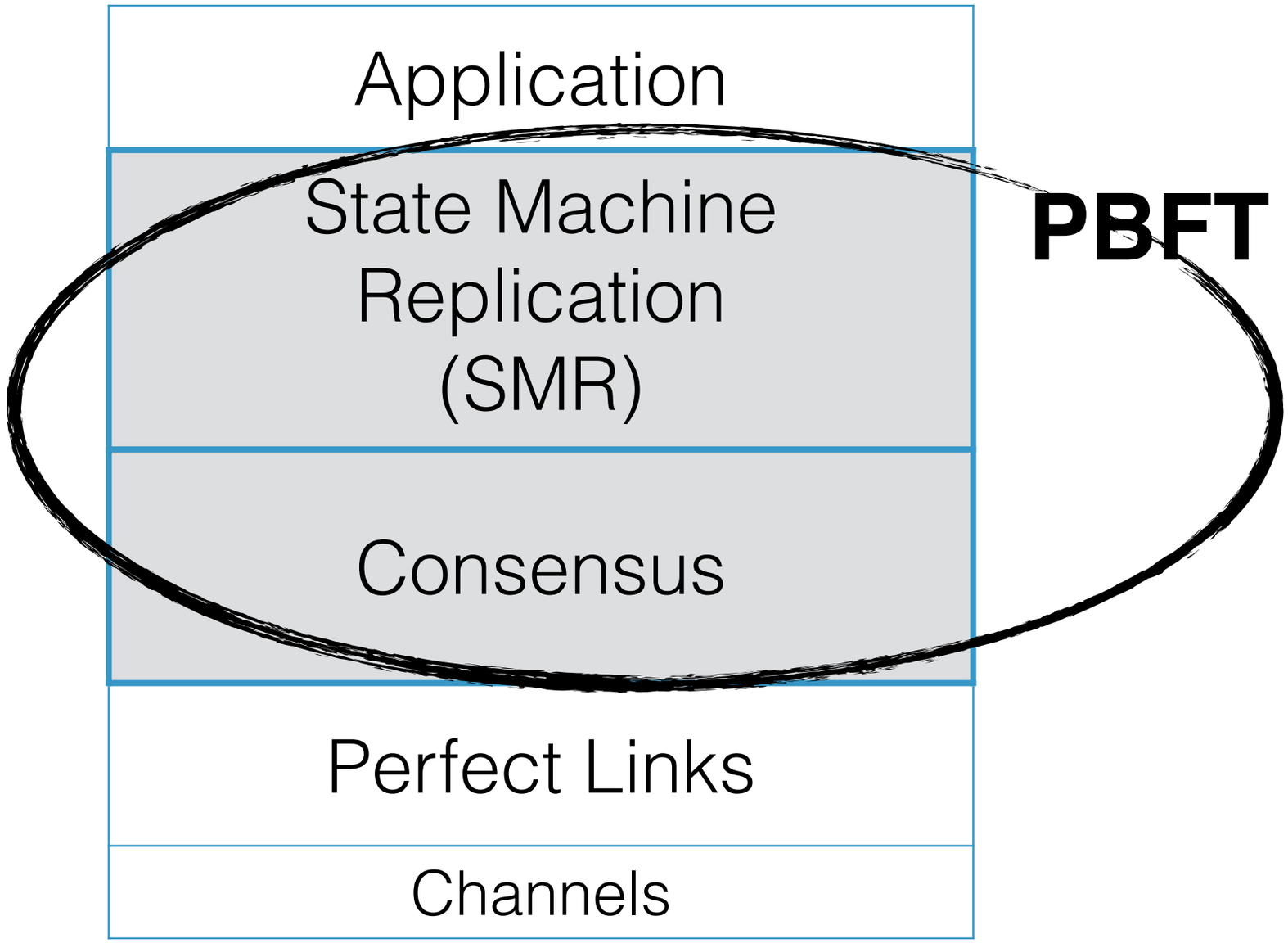
Miguel Castro

Barbara Liskov

Modules


Application
State Machine Replication (SMR)
Consensus
Perfect Links
Channels

Modules


Application
State Machine Replication (SMR)
Consensus
Perfect Links
Channels
PBFT

Overview

PBFT:

- **System model**
  — slightly different from what we've seen so far

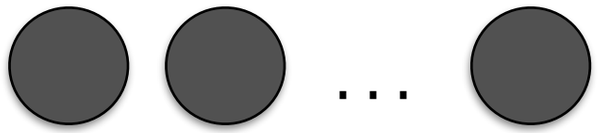
- SMR

- Consensus

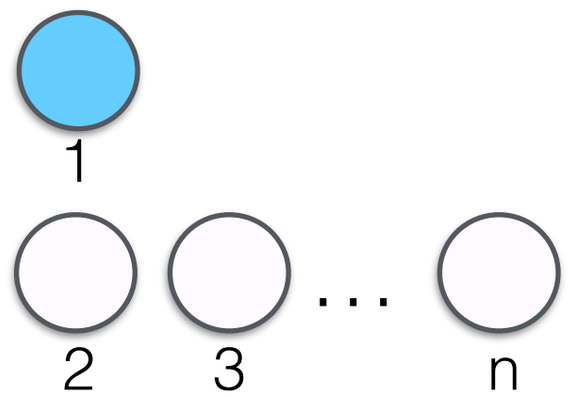
System model
Processes

Three types of processes in this algorithm:

- Clients



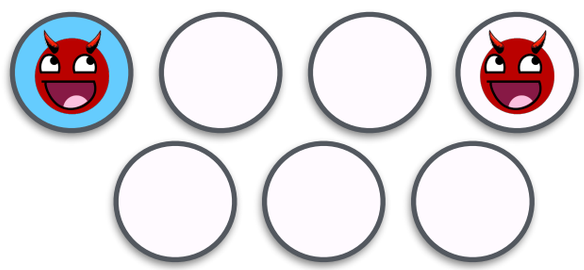
- n replicas
  - one of them is <u>primary</u>
  - others are <u>backup</u>



System model

Failure model

- Arbitrary (Byzantine) faults
- Clients:
  - Any client can be faulty
- Replicas:
  - $n = 3f + 1$
  - f faulty (upper bound)



System model

Network & crypto

- Assume perfect links
- Direct links between any two processes
- For messages:
  - Public-key signatures, message authentication codes
  - Avoid spoofing, replays, corruption
- Clients are authenticated
  - Can revoke access to faulty clients

Overview

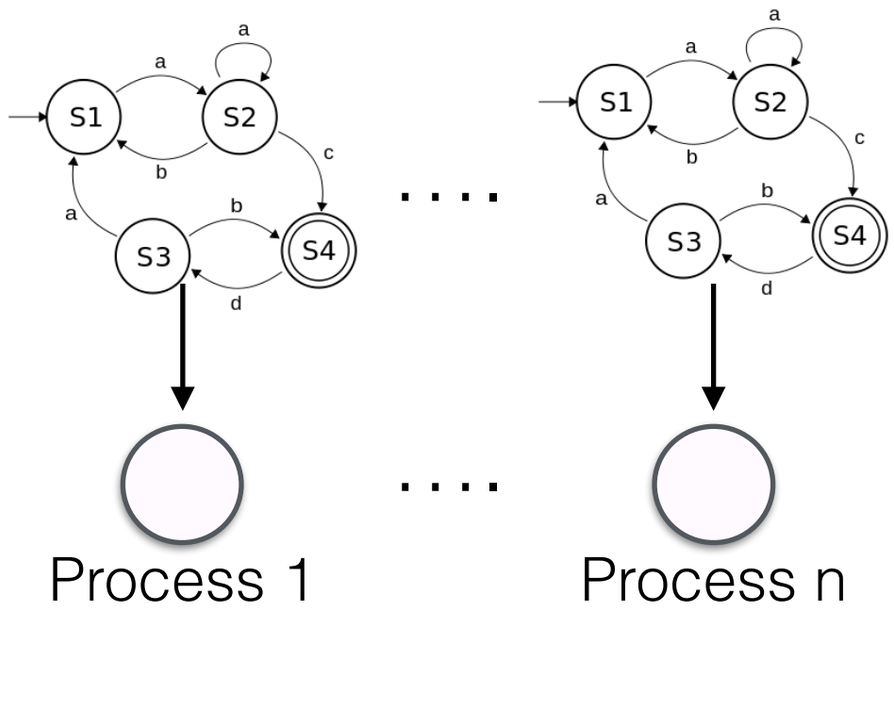
PBFT:

- System model
  — slightly different from what we've seen so far

- **SMR**

- Consensus

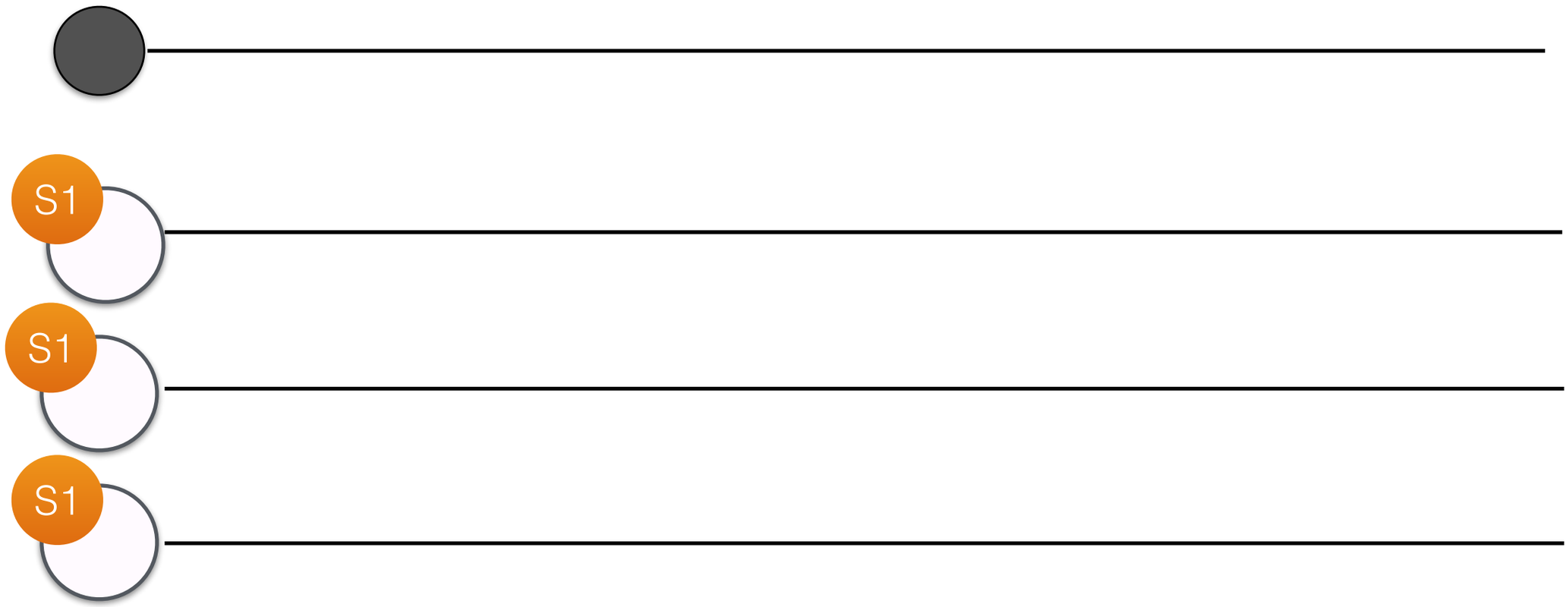
State Machine Replication (SMR)



- A fault-tolerance technique
- Basic ideas:
  - Application = state machine
    - Run the application on multiple processes
    - Each processes is a faithful replica of the application
- Note: We can ignore the primary/backup distinction in this example

SMR

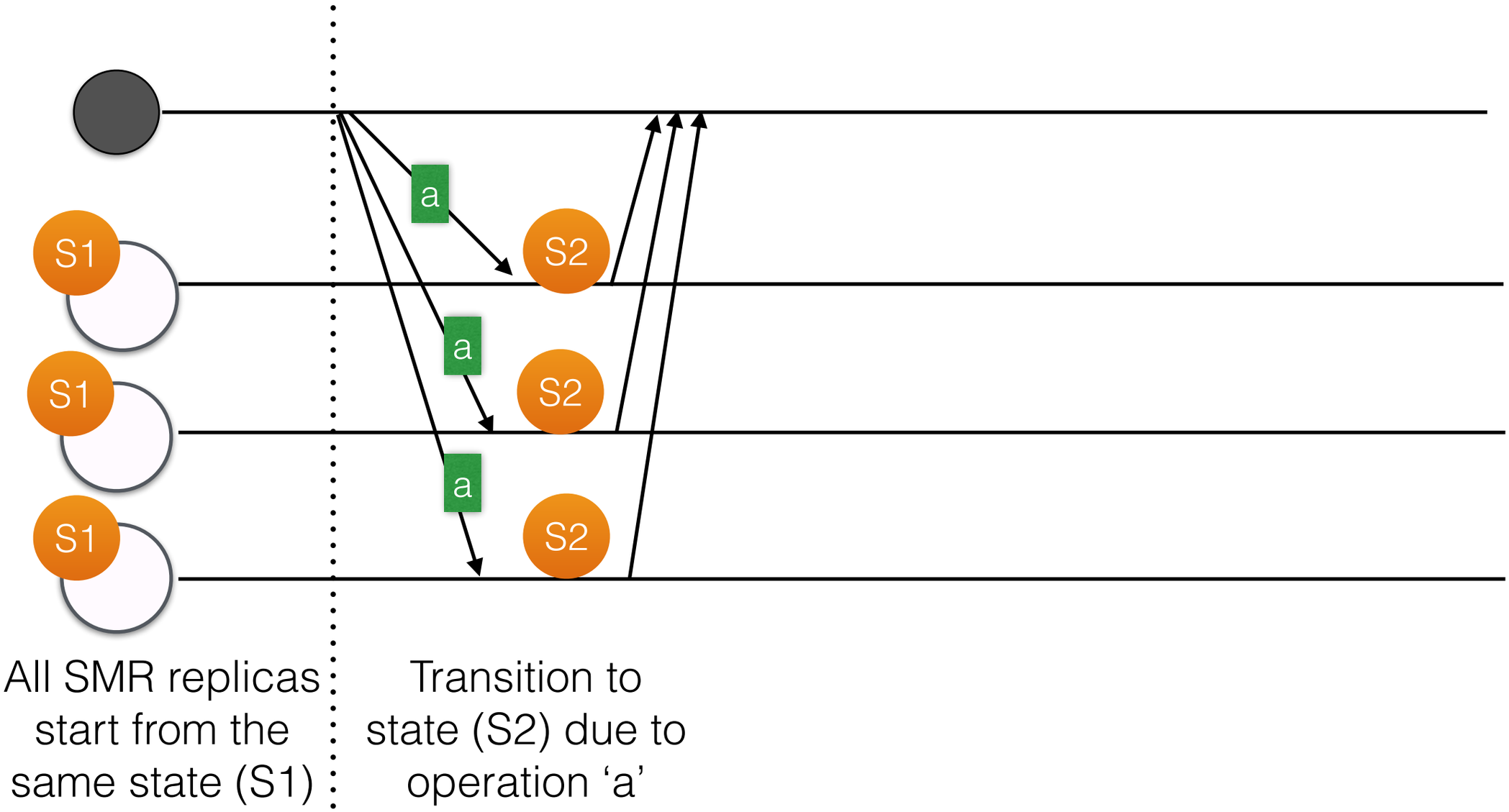
in a nutshell



All SMR replicas start from the same state (S1)

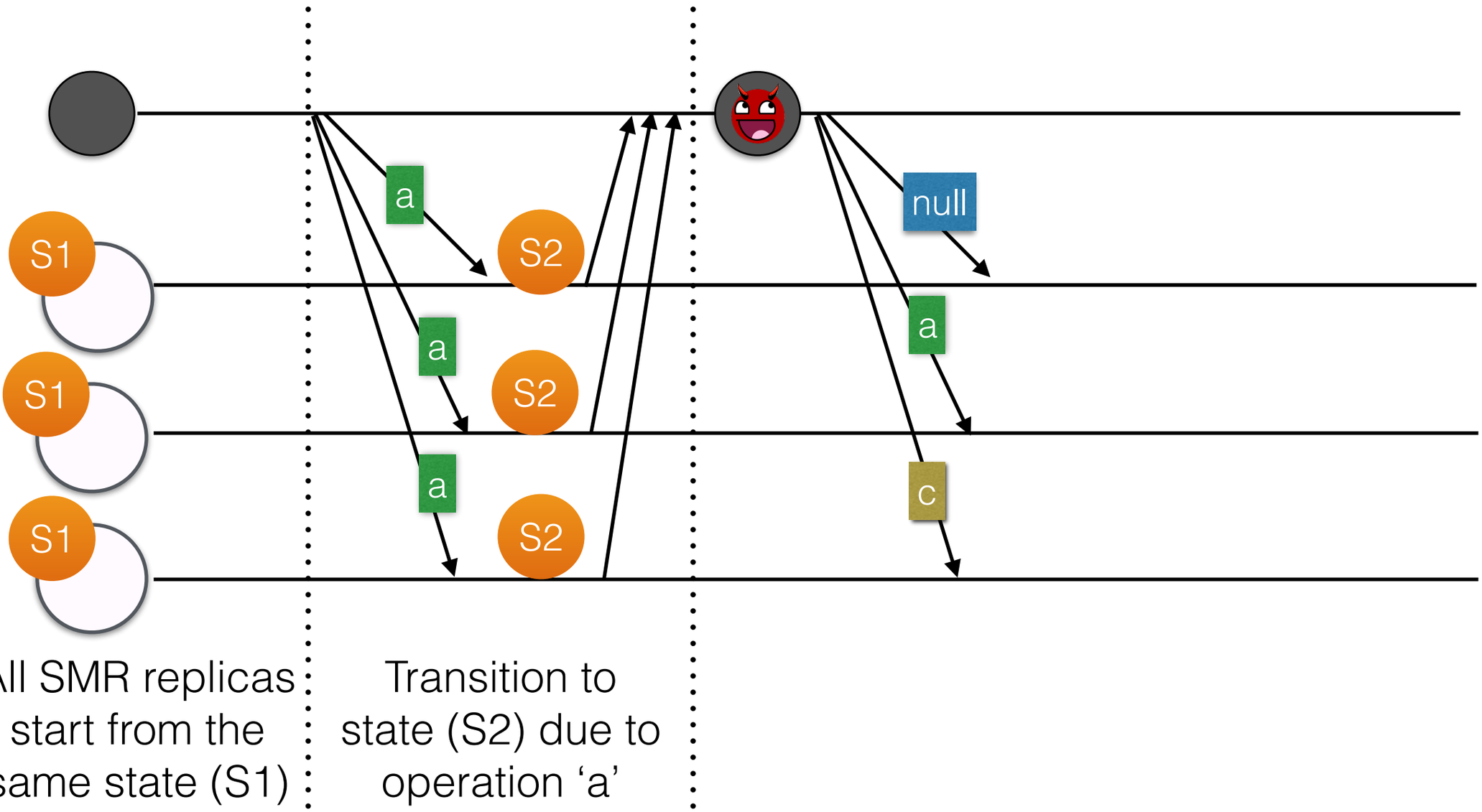
SMR

in a nutshell



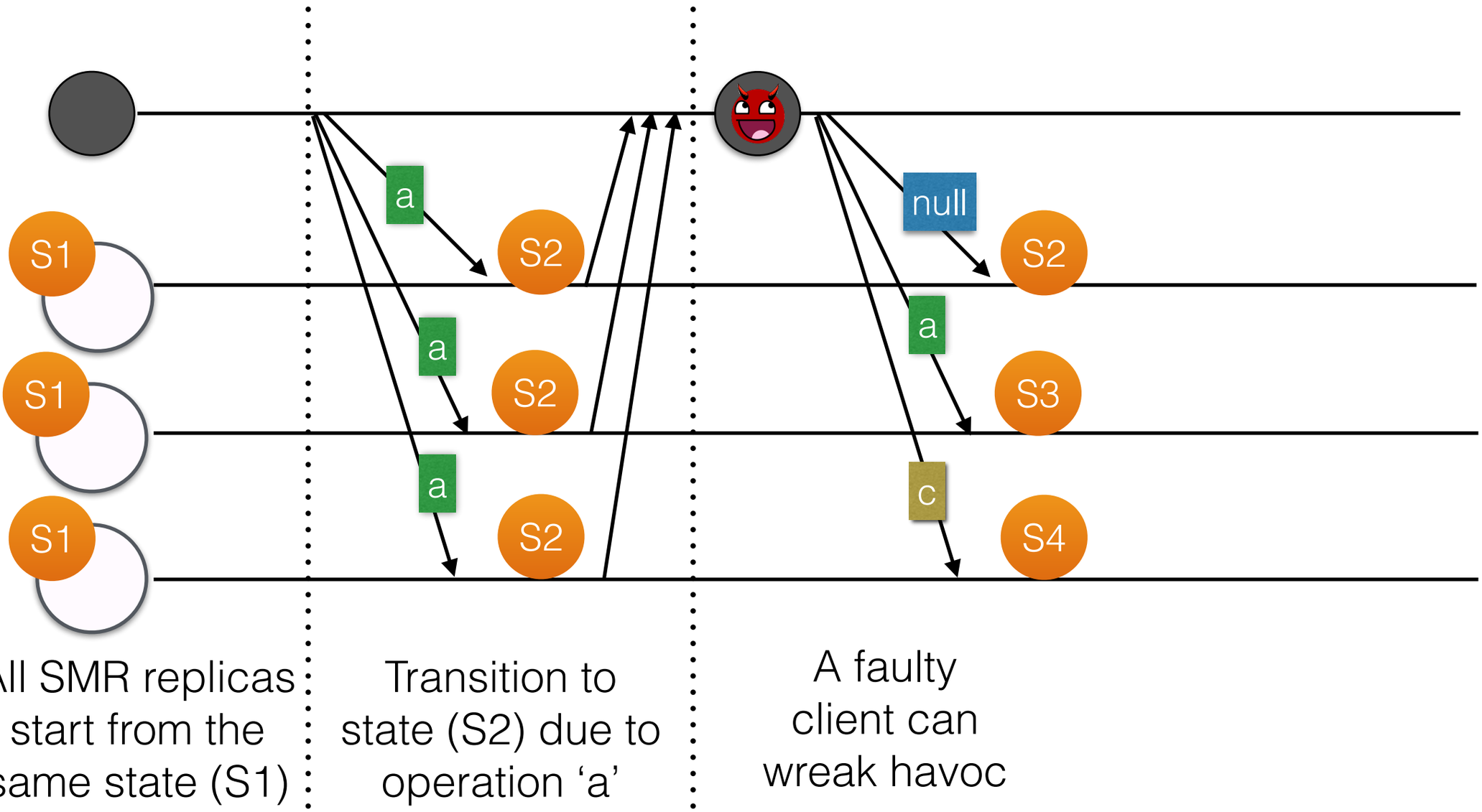
SMR

in a nutshell



SMR

in a nutshell



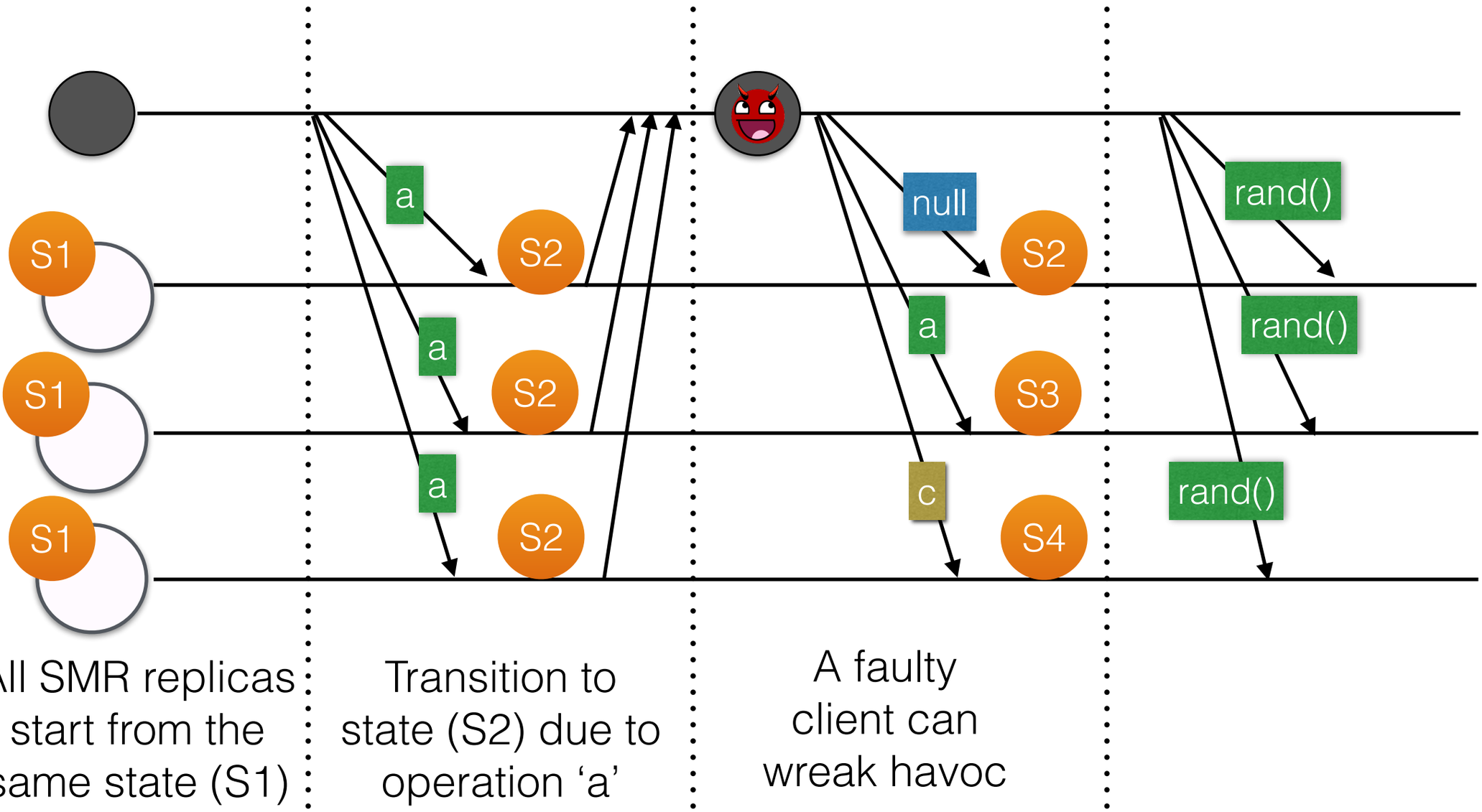
All SMR replicas start from the same state (S1)

Transition to state (S2) due to operation ‘a’

A faulty client can wreak havoc

SMR

in a nutshell



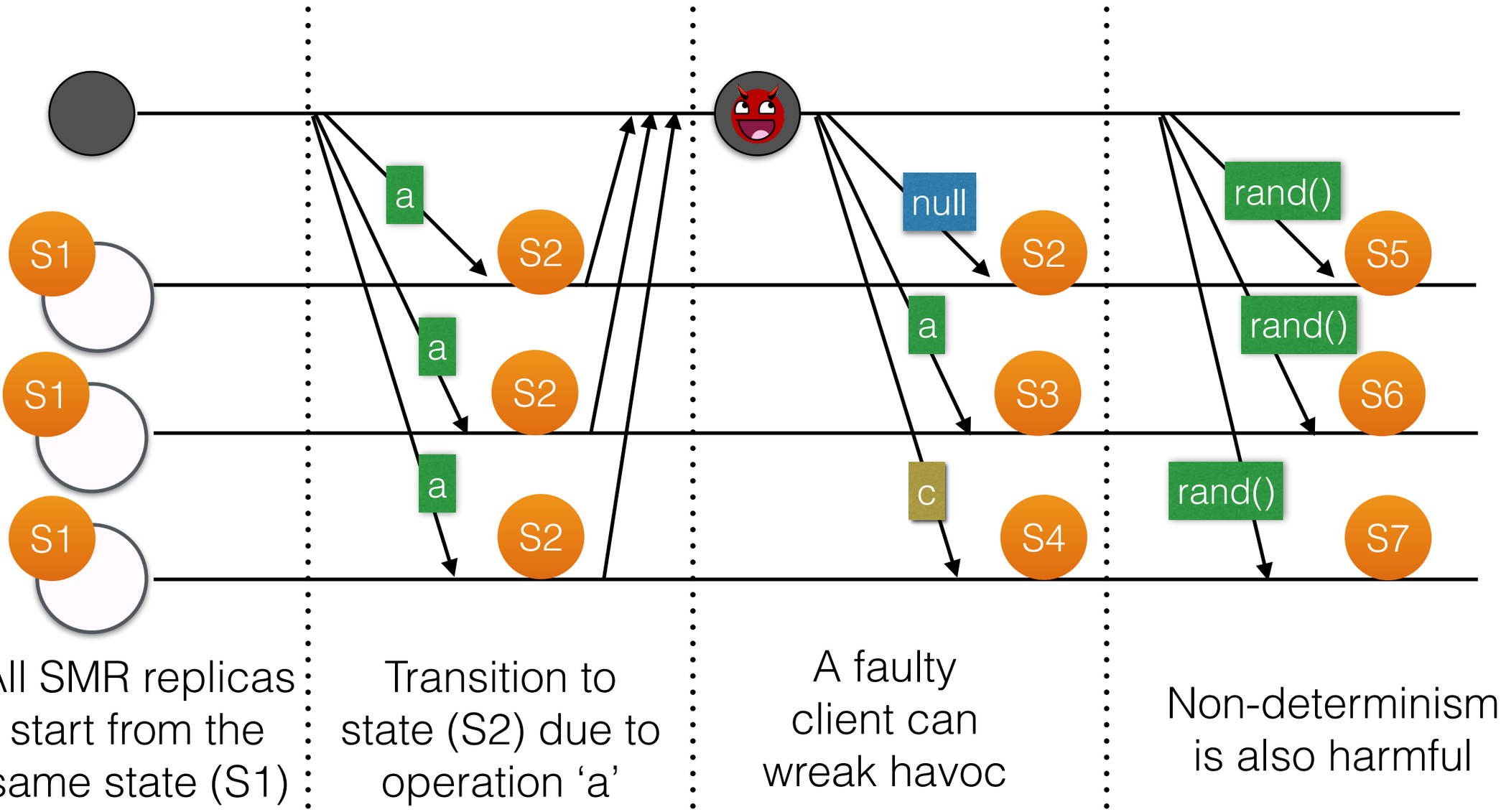
All SMR replicas start from the same state (S1)

Transition to state (S2) due to operation ‘a’

A faulty client can wreak havoc

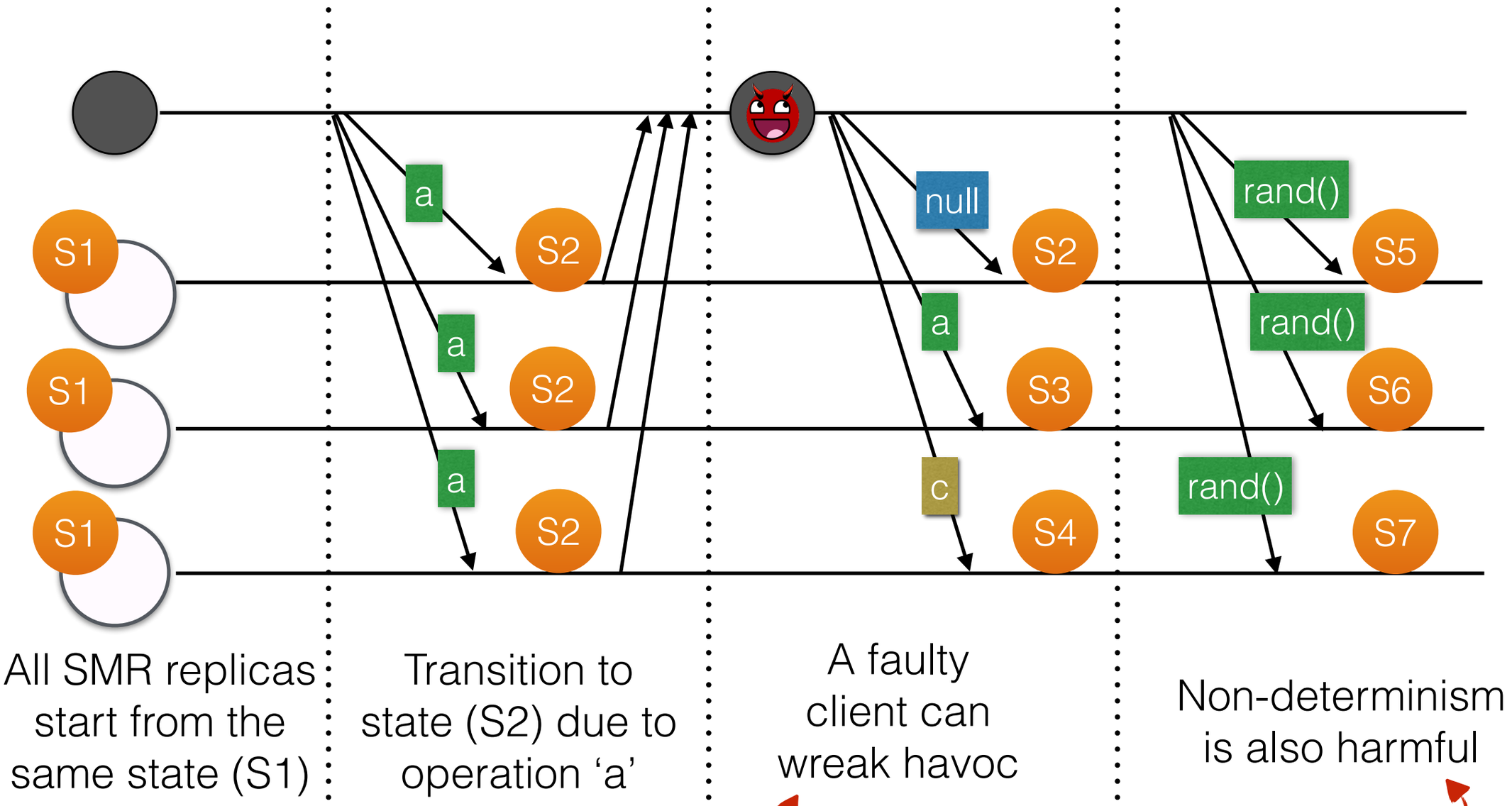
SMR

in a nutshell



SMR

in a nutshell



All SMR replicas start from the same state (S1)

Transition to state (S2) due to operation ‘a’

A faulty client can wreak havoc

Non-determinism is also harmful

Diverging states!

SMR
Requirements

- Avoid diverging states

- All replicas must:

1. Start in the same state

2. Execute the same sequence of operations

3. Use only provided operation (+parameters), thus avoid non-determinism

SMR
Requirements

- Avoid diverging states

- All replicas must:

1. Start in the same state
2. Execute the same sequence of operations
3. Use only provided operation (+parameters), thus avoid non-determinism

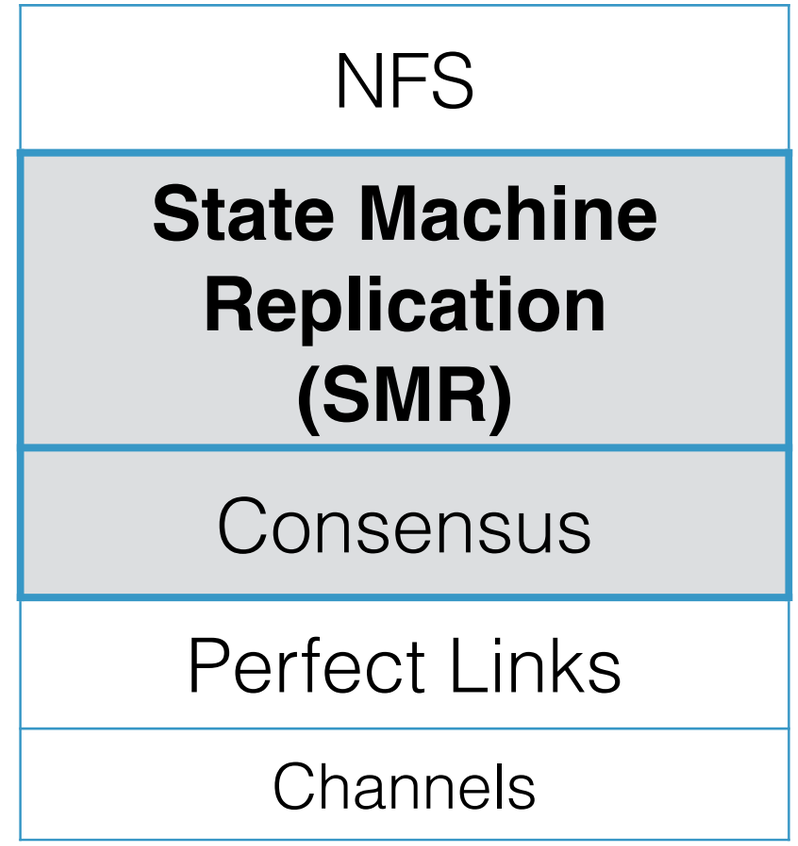
Simple

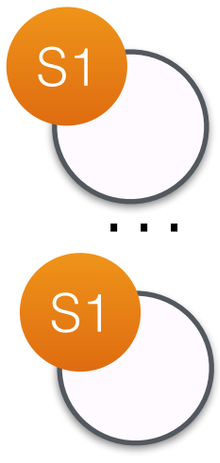
Consensus

Depends on application

SMR
in PBFT

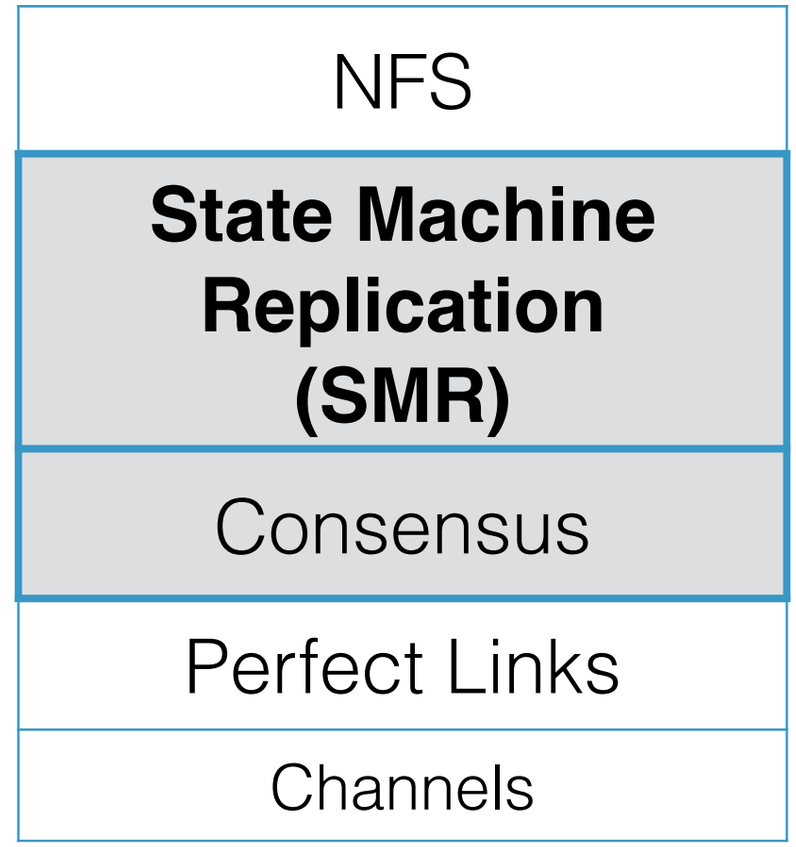
- Application = a distributed file system — Network File System (NFS)
- Operations = write to a file, delete, etc.
- Primary/backup distinction is relevant

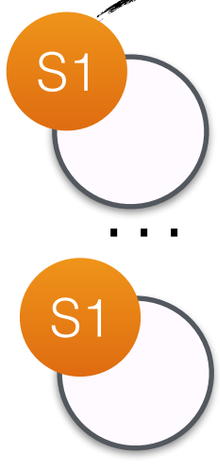




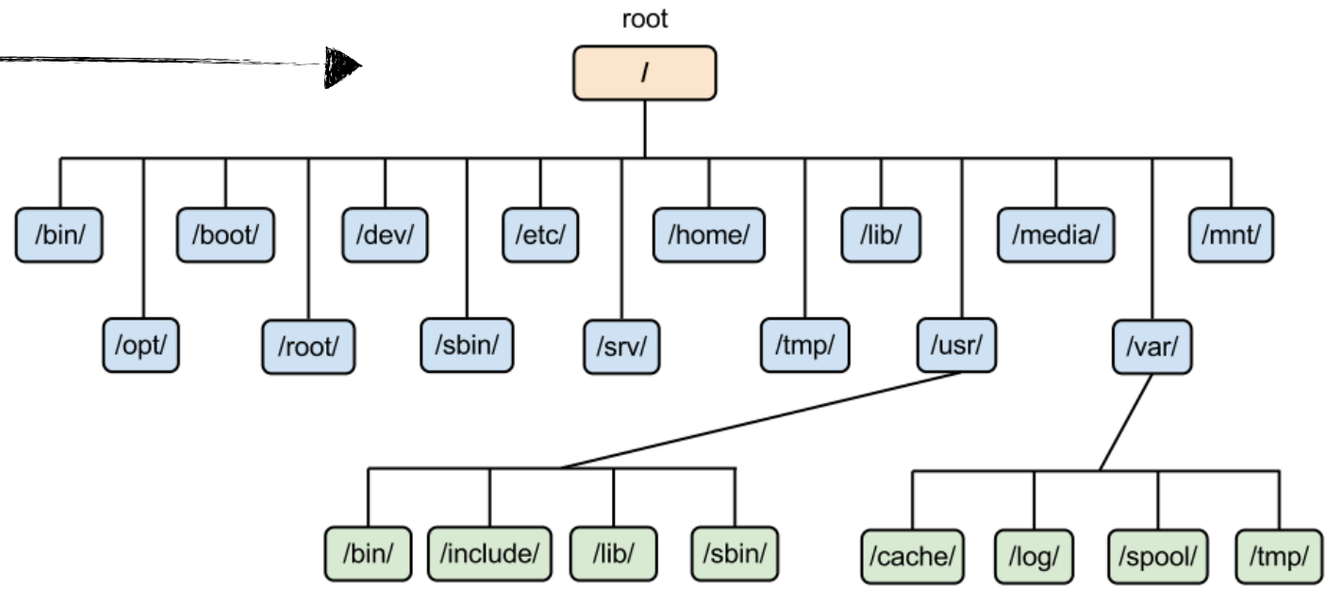
SMR
in PBFT

- Application = a distributed file system — Network File System (NFS)
- Operations = write to a file, delete, etc.
- Primary/backup distinction is relevant



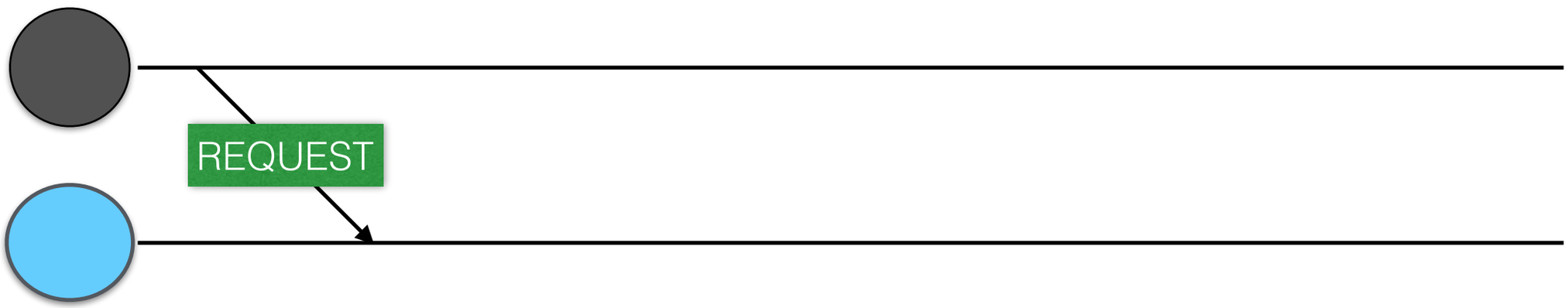


The replicated state is a file system



SMR

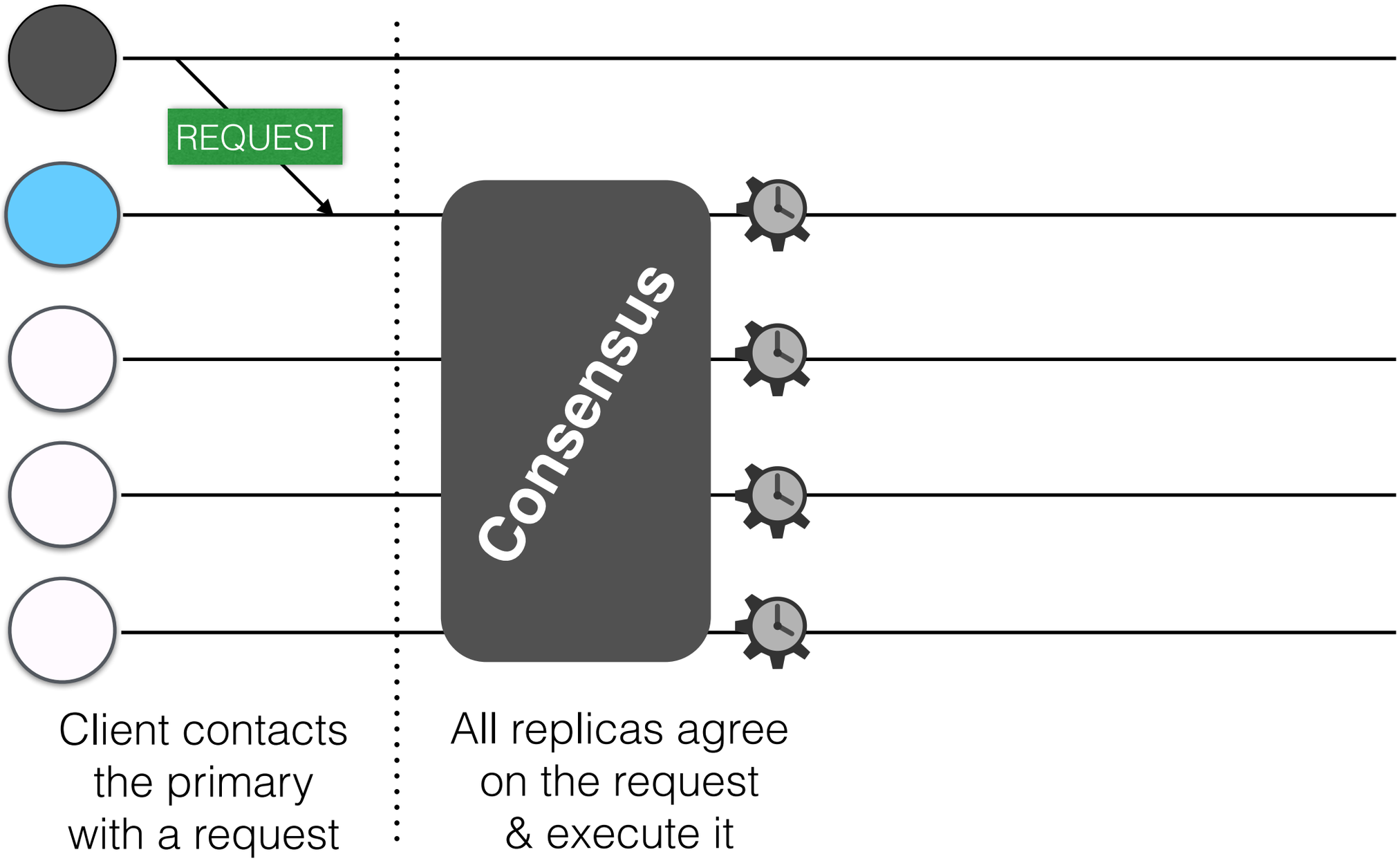
in PBFT



Client contacts
the primary
with a request

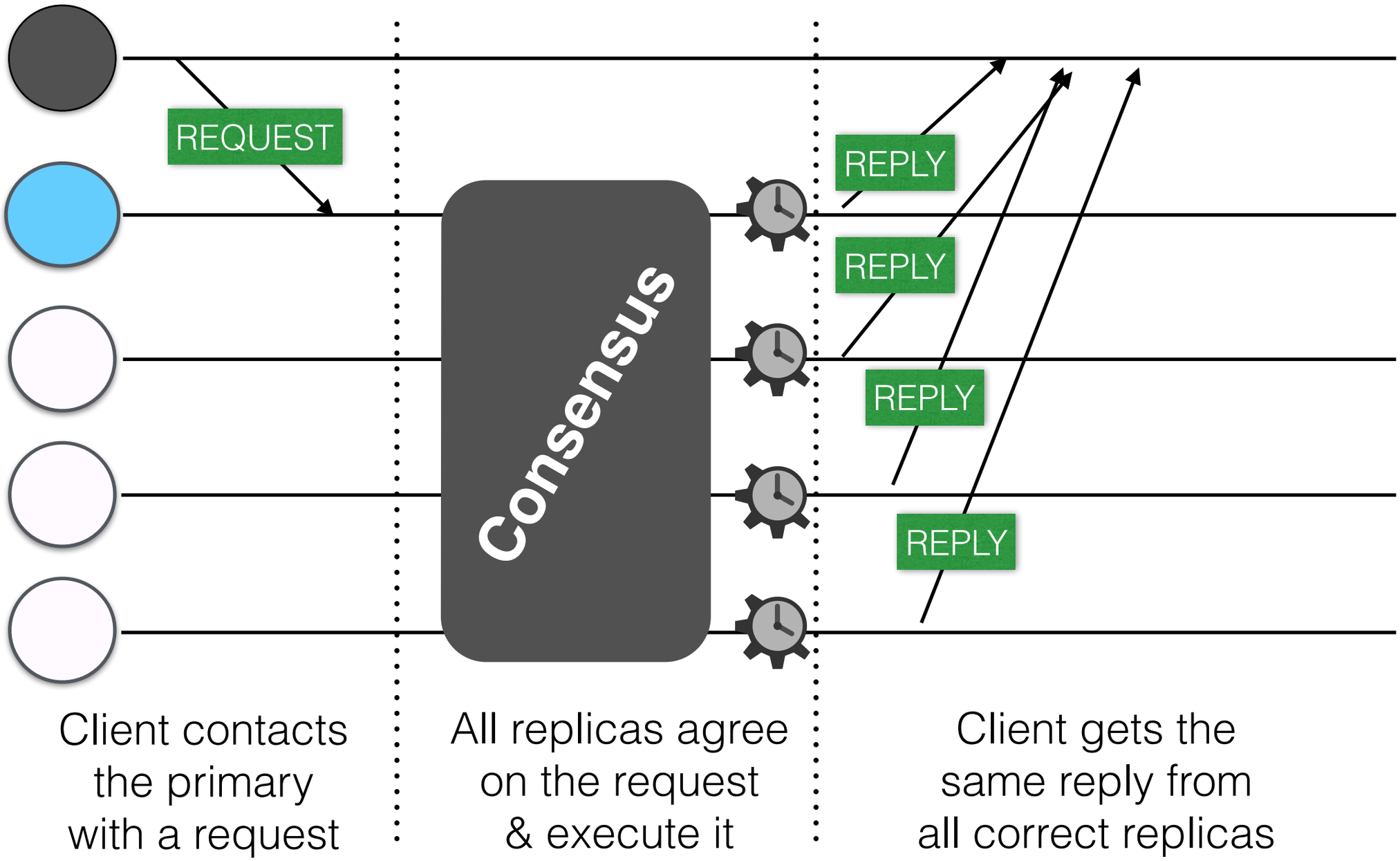
SMR

in PBFT


REQUEST
Consensus
Client contacts the primary with a request
All replicas agree on the request & execute it

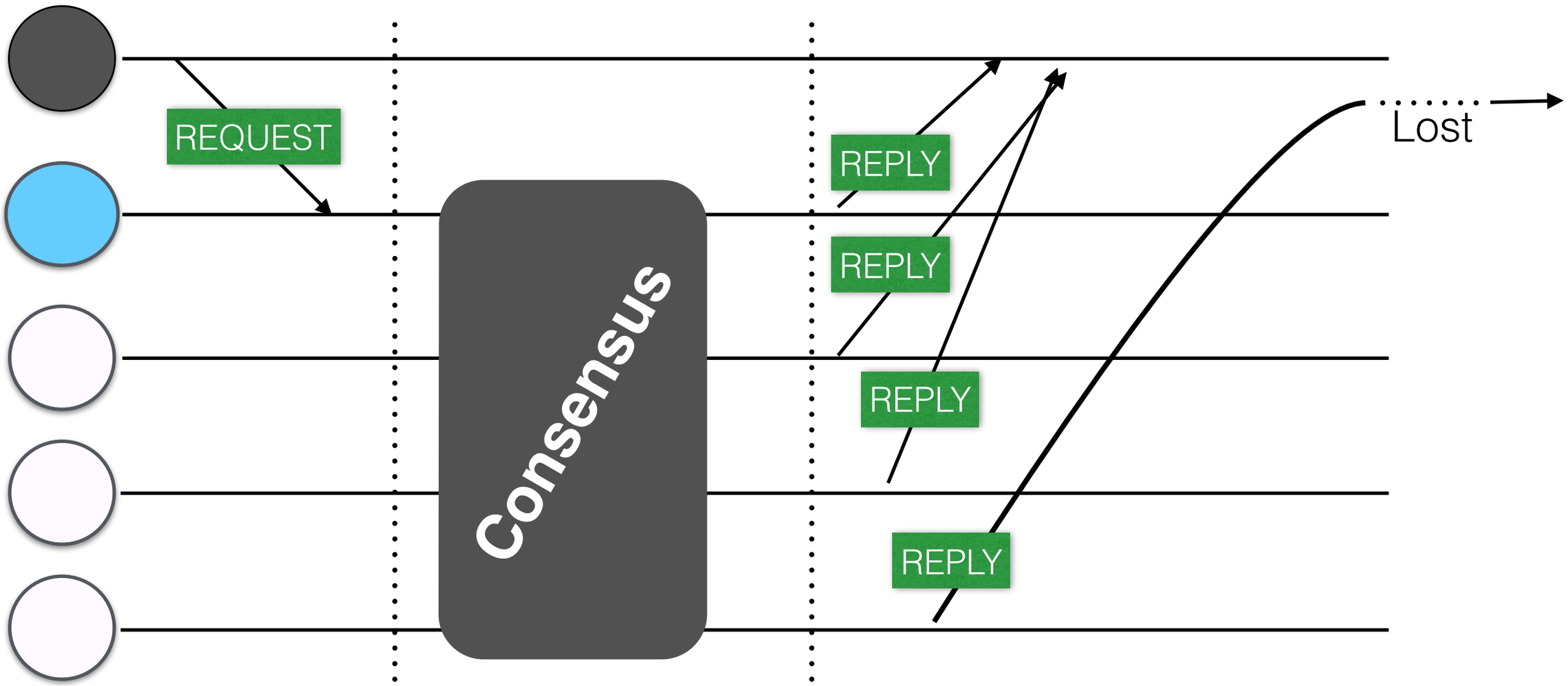
SMR

in PBFT


REQUEST
Consensus
REPLY
REPLY
REPLY
REPLY
Client contacts the primary with a request
All replicas agree on the request & execute it
Client gets the same reply from all correct replicas

SMR

in PBFT



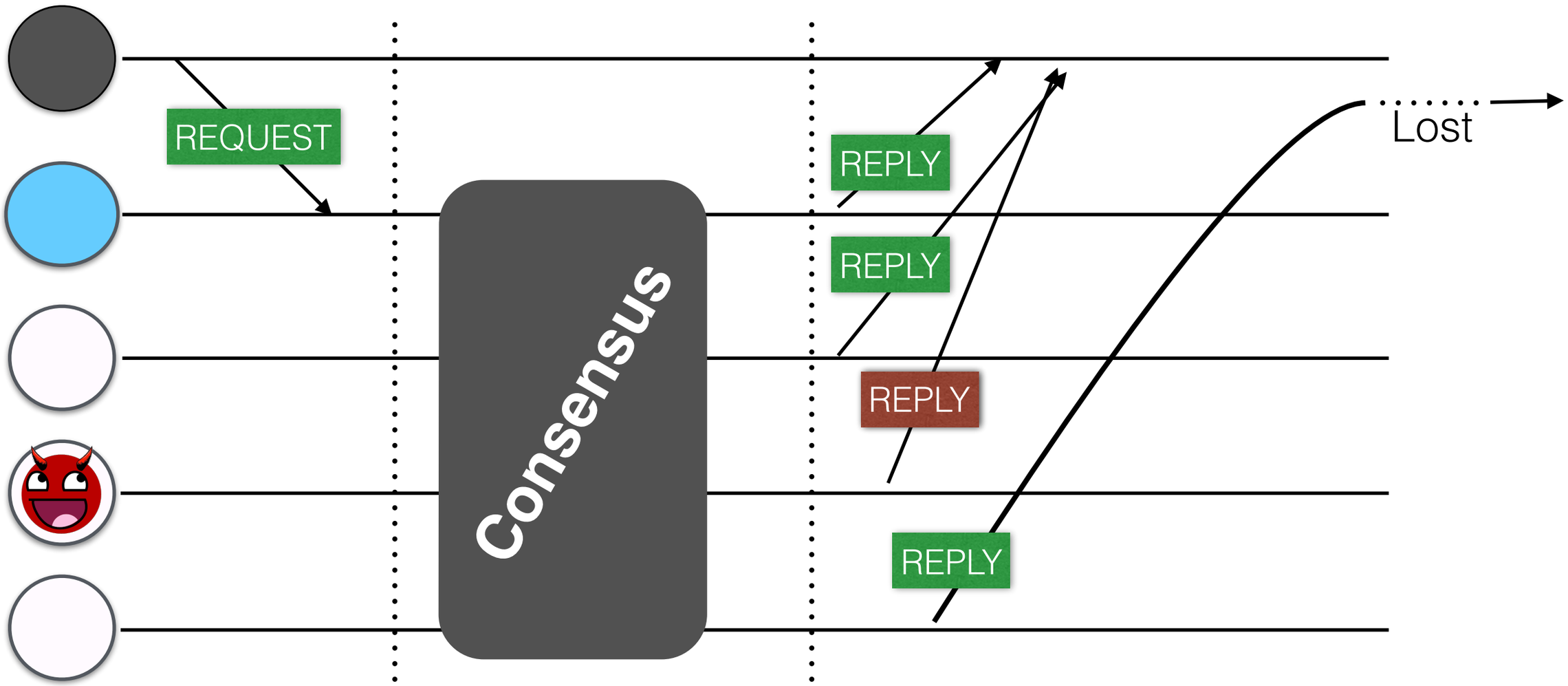
Client contacts the primary with a request

All replicas agree on the request & execute it

Redundancy in Replies = Cope with failures

SMR

in PBFT



Client contacts the primary with a request

All replicas agree on the request & execute it

Redundancy in Replies = Cope with failures

Overview

PBFT:

- System model
  — slightly different from what we've seen so far

- SMR

- **Consensus**

Consensus

- The core for many algorithms, including:
- TRB, Group membership, View synchronous b-cast, State machine replication

Traditionally

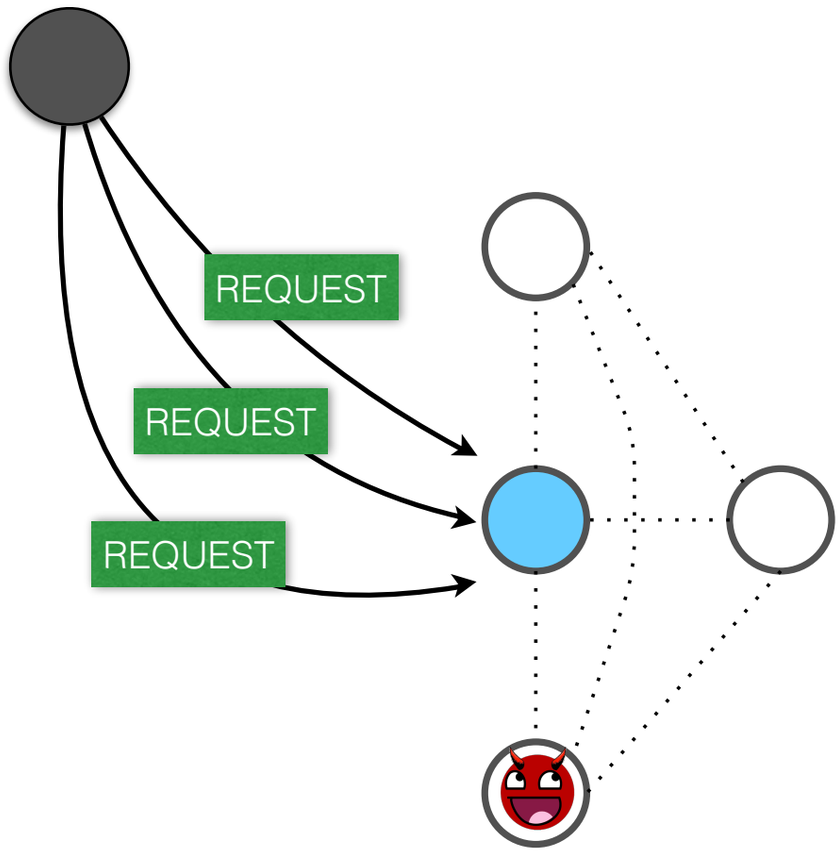
Instance
- Processes propose <u>values</u>
- Agree on a proposed value

In PBFT:

Instance
- Clients propose <u>request</u>
- Primary multicasts one request to backup replicas
- Replicas accept the request

Consensus
in PBFT

- We'll assume one client
  - Proposals = requests for application operations
- Assume:
  - n = 4, f = 1
  - The faulty replica does not cooperate
- Concurrent requests:
  - Consensus to agree on a sequential execution of requests



Consensus
in PBFT

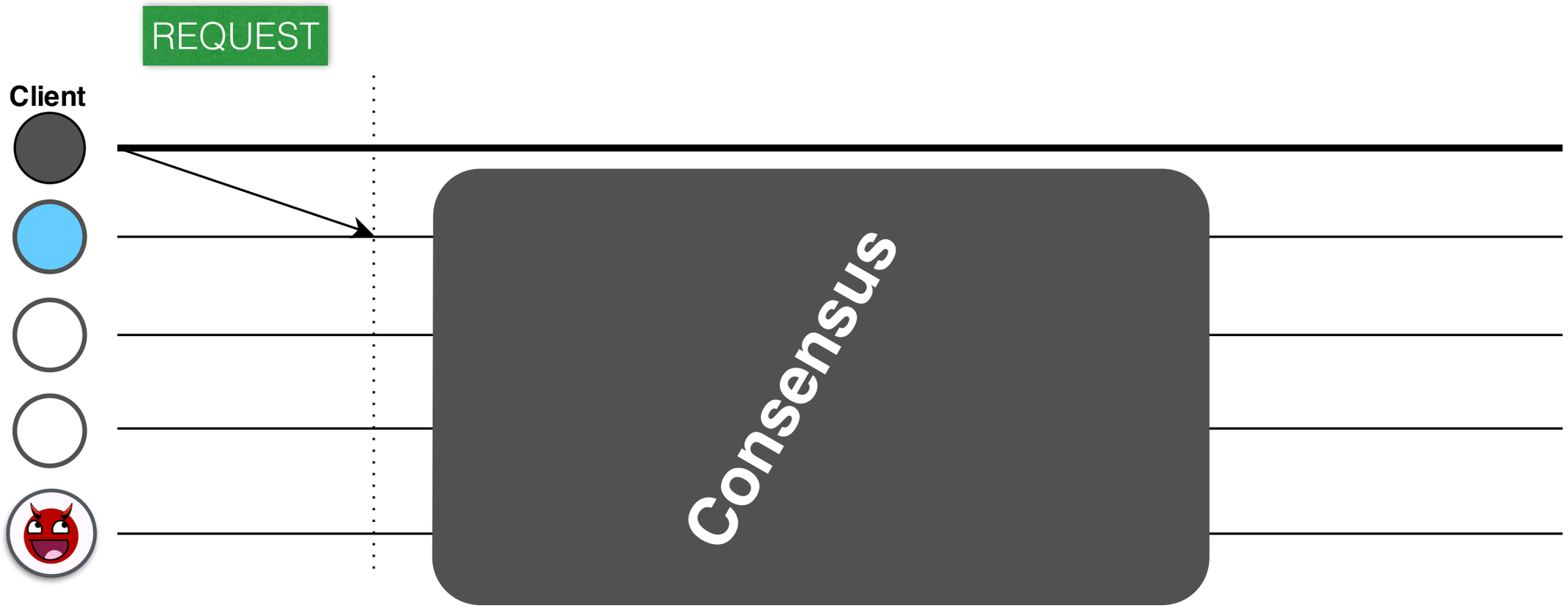
Algorithm ideas:

- Client sends requests to the primary replica
- Execute a sequence of consensus instances:
    - Each instance is dedicated to a request
    - Instances (and therefore requests) are sequentially ordered by the primary
    - Backup replicas adopt requests from the primary in the imposed order

Properties: **Validity, Agreement, Termination, Integrity**

Consensus

in PBFT



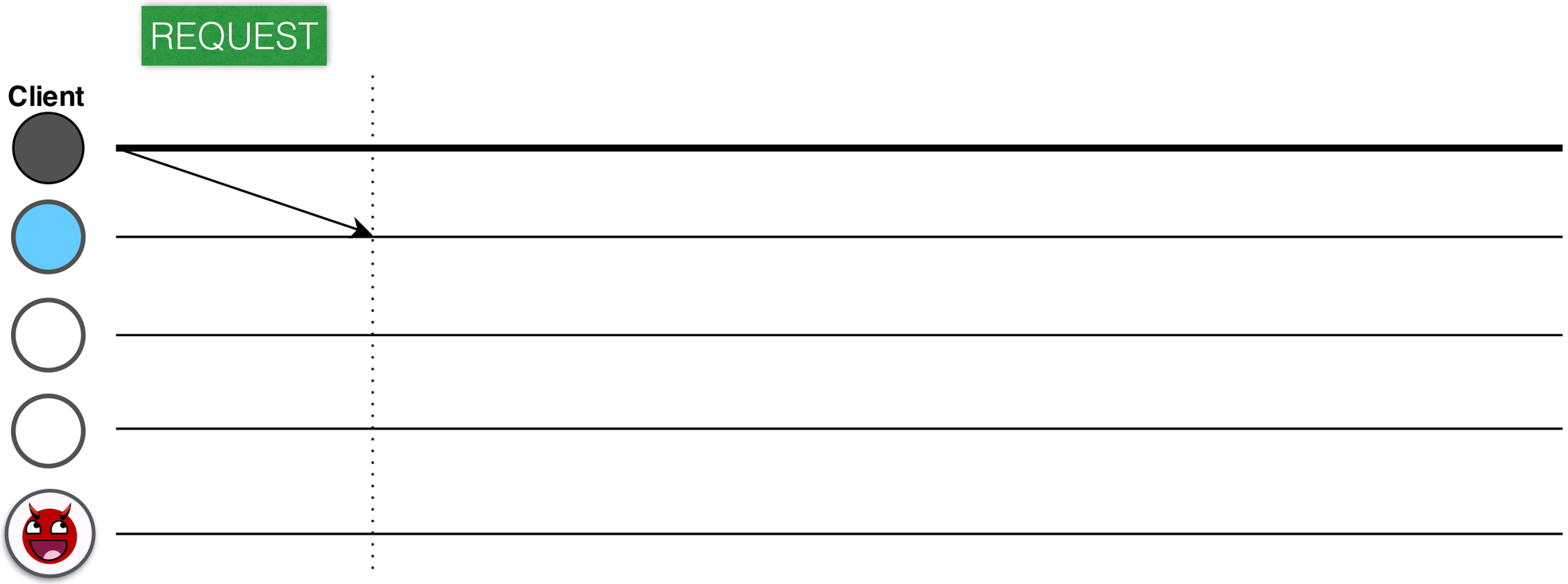
Consensus

in PBFT

A three-phase protocol

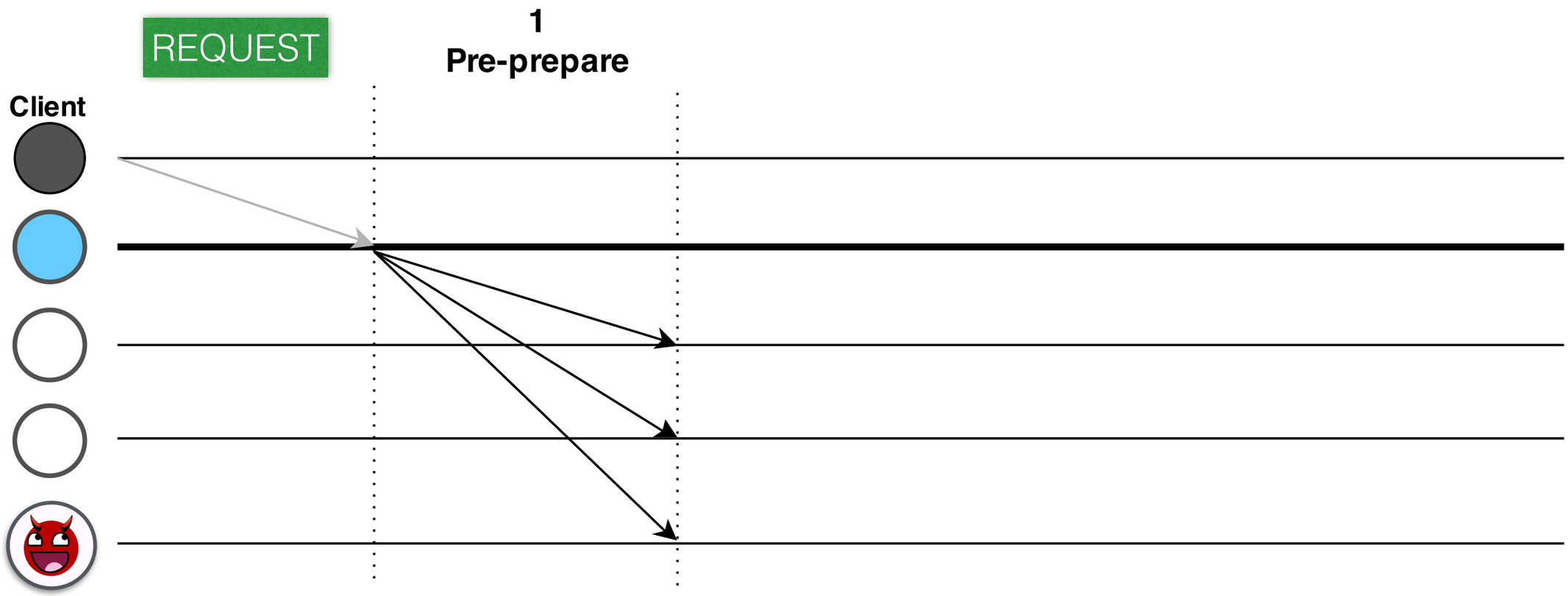
Consensus instance

= three-phase protocol



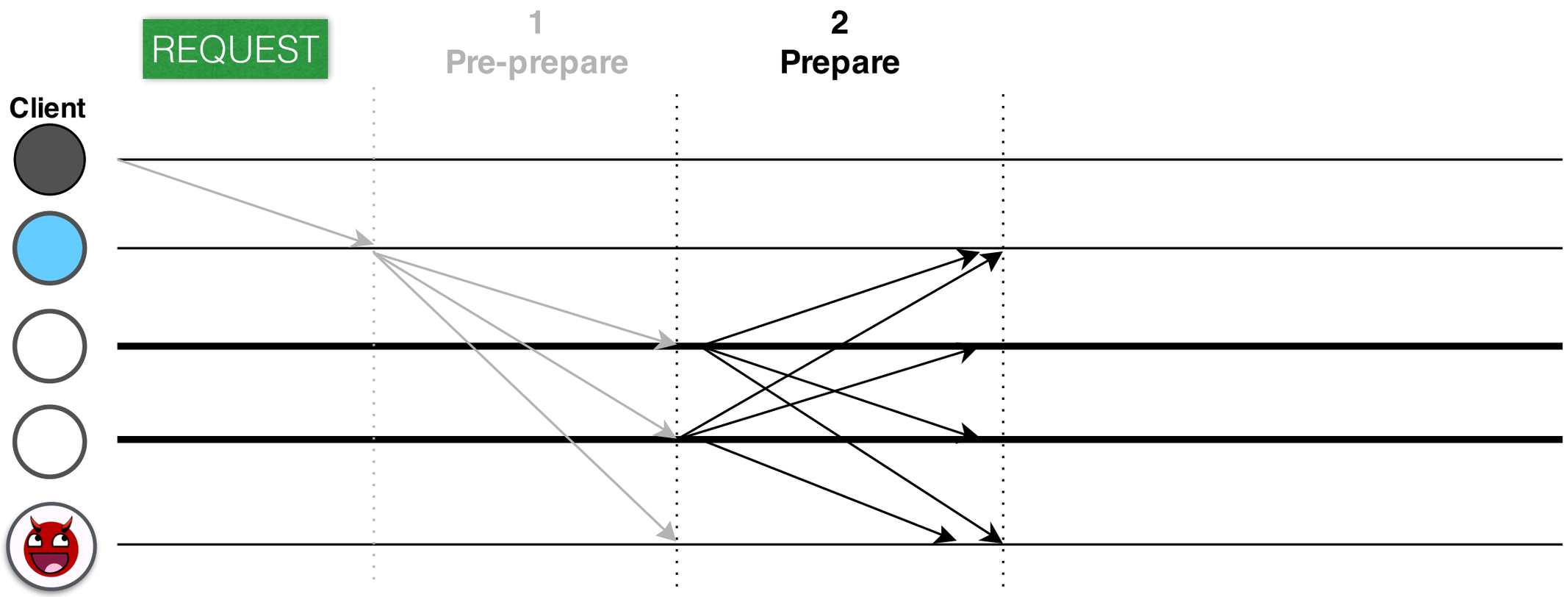
Consensus instance

= three-phase protocol



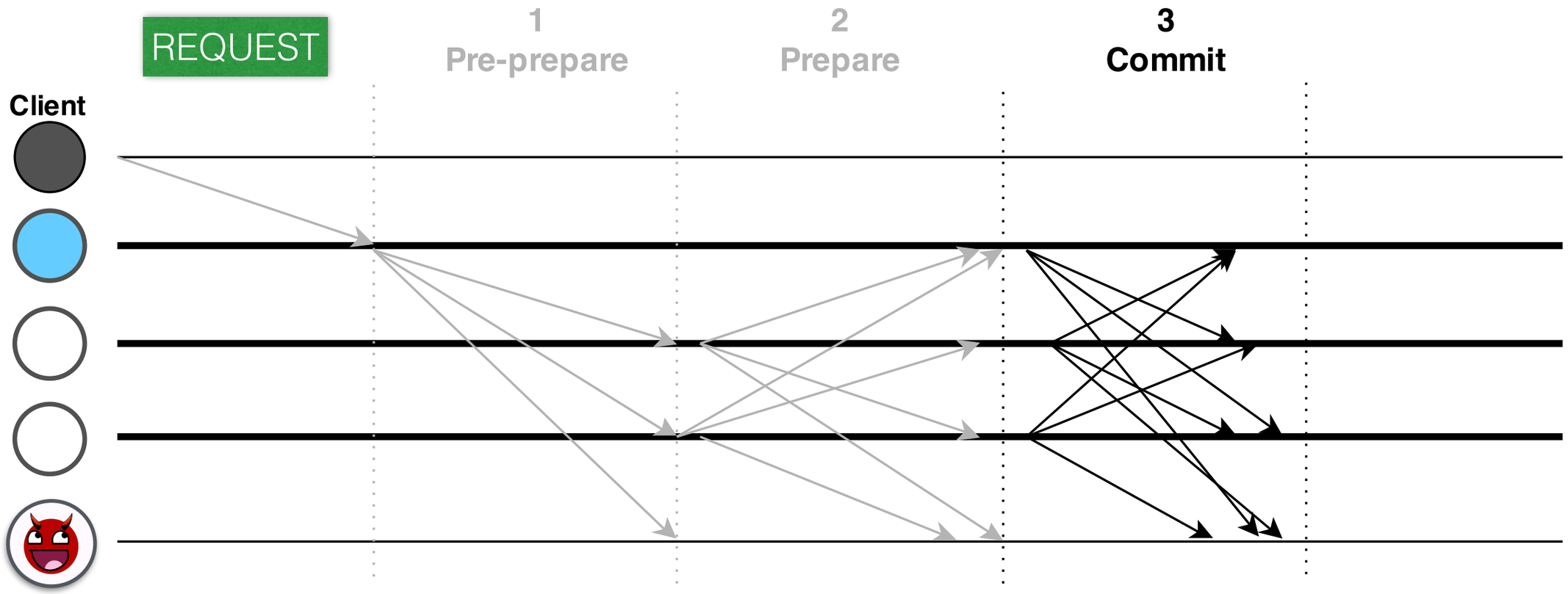
Consensus instance

= three-phase protocol



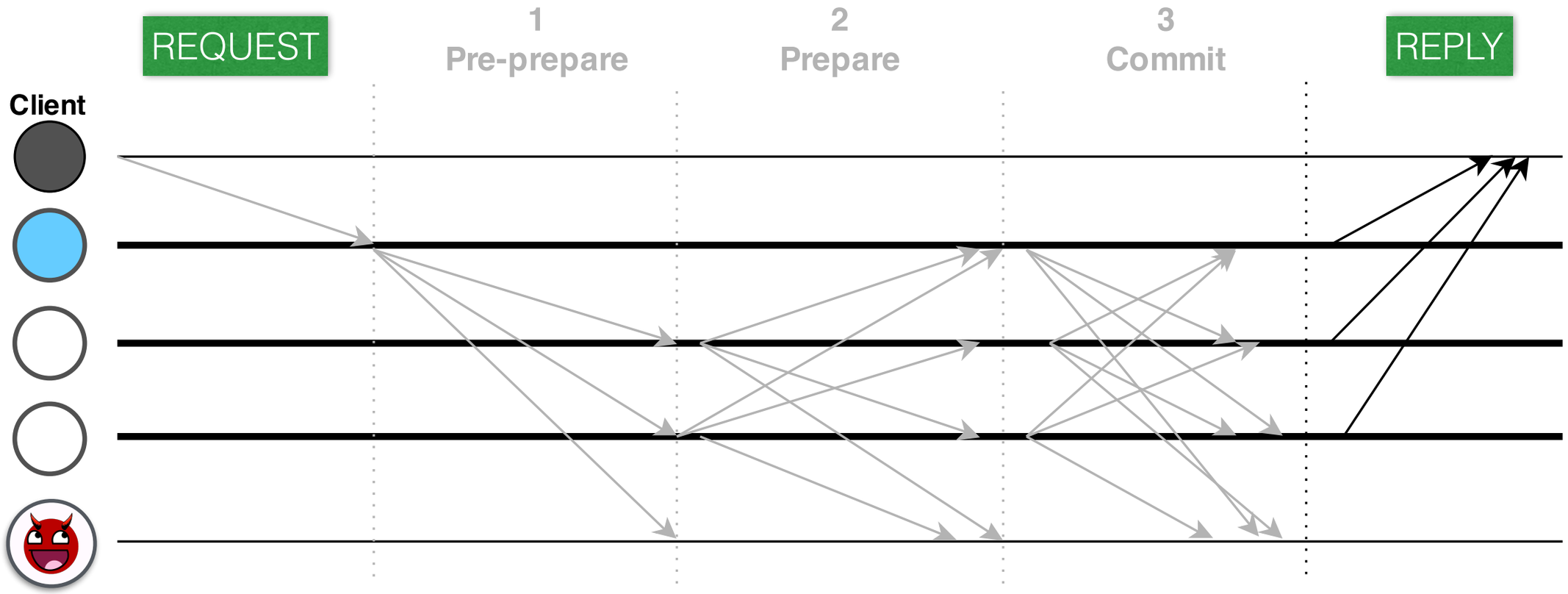
Consensus instance

= three-phase protocol



Consensus instance

= three-phase protocol



Consensus

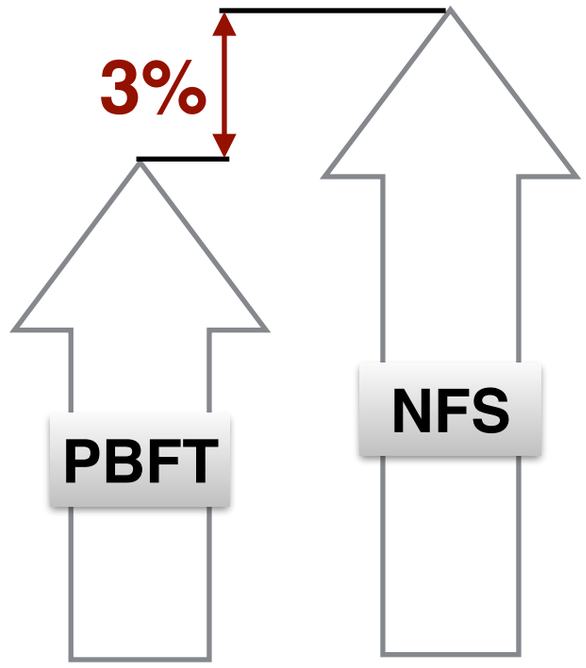
Corner case

What if the primary is faulty, e.g.
does not multicast the request to the backups?

- View change protocol: primary replaced by one of the backups
- Idea:
  - Replicas are numbered 1 … n
  - In view *v*, the replica *p* is the primary, where
    p = v mod n

Practical BFT

"Reasonable overhead"



- Does not assume synchrony
- Some clever optimizations:
  - MD5 replaces digital signatures
  - Message digests
  - Read-only requests, tentative execution

Further reading

- Castro, M., & Liskov, B. (1999). Practical Byzantine fault tolerance. OSDI, (February), 1–14. Available at: http://dl.acm.org/citation.cfm?id=296806.296824

- Castro, M. (2011). Practical Consensus. Microsoft Research Cambridge. Available at: http://msrvideo.vo.msecnd.net/rmcvideos/167097/dl/167097.pdf