From crash to Byzantine fault tolerance (and back, to secure TEE storage)

Rodrigo Rodrigues
Instituto Superior Técnico (ULisboa) and INESC-ID

Lecture at SPTDC 2020
July 9, 2020

PART 1 – CRASH AND BYZANTINE FAULT MODELS

Motivation: tolerating faults

- Suppose you manage a data center with 75,000 machines, 4 disks per machine
- According to recent studies, 3% of disks in data centers require replacement in a given year
- How often do disks suffer such permanent faults?
  - Once per week?
  - Once per day?
  - Once per hour?
  - Once per minute?

- 75 000 X 4 = 300 000 disks
- 300 000 X .03 = 9 000 disk faults per year
- 365 days X 24 hours = 8 760 hours per year

- These were just permanent disk faults, need to account for everything else!

The first published replication protocol

A PRINCIPLE FOR RESILIENT SHARING OF DISTRIBUTED RESOURCES*

Peter A. Alsberg and John D. Day
Center for Advanced Computation
University of Illinois at Urbana-Champaign

ICSE'76

Keywords: resilient protocols, resource sharing, distributed control, distributed computer systems, resilient resource sharing

A technique is described which permits distributed resources to be shared (services to be offered) in a resilient manner. The essence of the technique is to a priori declare one of the server hosts primary and the others backups. Any of the servers can perform the primary duties. Thus the role of primary can migrate around the set of servers. The concept of n-host resiliency is introduced and the error detection and recov-

be completely dissimilar (e.g., weather data may be stored on the ARPANET datacomputer and processed on the ILLIAC IV). Between these two extremes lie the resource sharing concerns of interest to most users.

The user expects a tolerable, as well as tolerant, resource sharing environment. The user we are interested in wants a maximum degree of automation and transparency in his resource sharing. He wishes the resource sharing to be resilient to host failures and, when catastropic failures occur, he would like a "best effort" recovery to be automatically initiated by the

Protocols require: assumptions, assumptions, assumptions

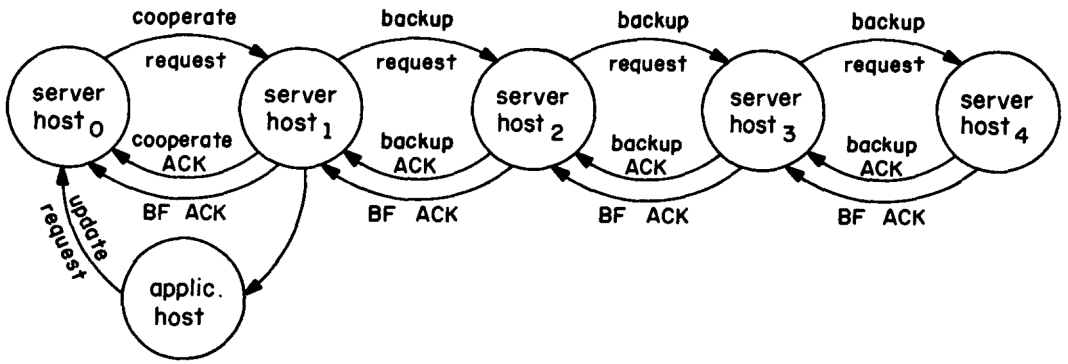


Figure 3

Summary of the message flow for the resiliency scheme.

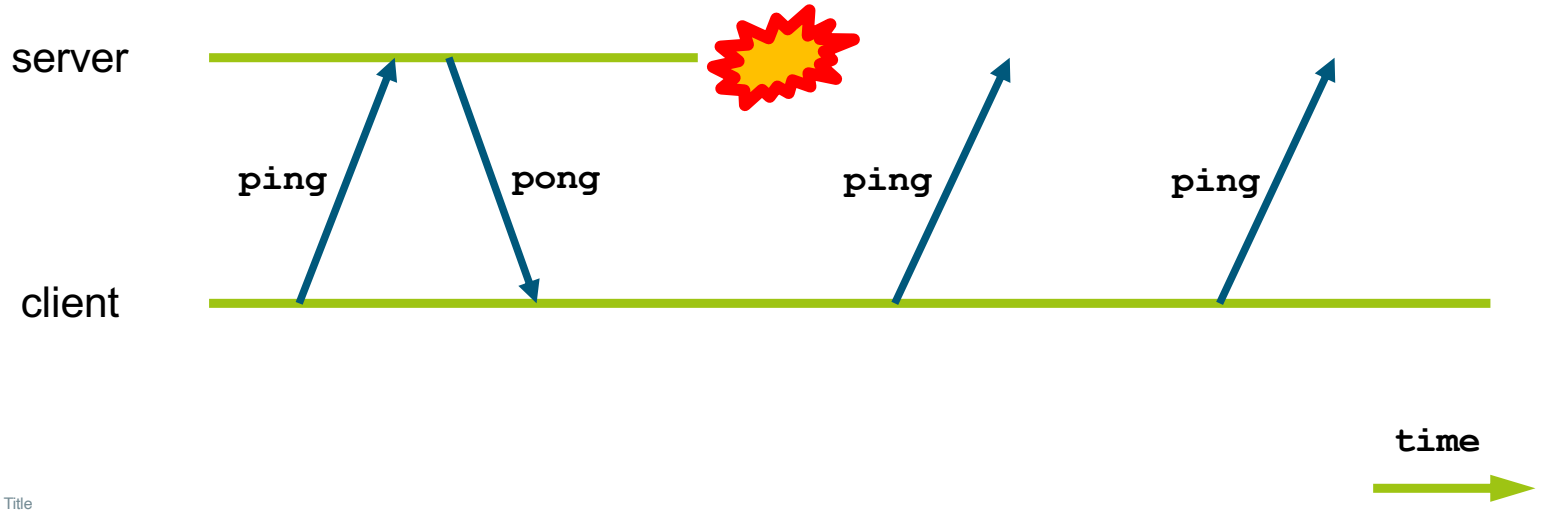
Today's Lecture

- Failure behavior:
  - Stop executing protocol steps and sending messages
  - Emit wrong outputs
- Timing behavior:
  - Upper bound on message transmission and processing time (faults are detectable)
  - No such upper bound (can't distinguish slow vs. faulty participant)

Asynchronous model

Crash fault tolerance (CFT)

- Assumes a crash fault model → nodes fail by silently halting
- Matches many real-world faults → used in most data center systems
- Some other faults can be turned into crash faults (e.g., by halting when exception is caught)



Protocols: specification versus implementation

- The specification describes what the protocol is supposed to do
- Implementation is the protocol logic that should enforce that specification

- Focus on a storage system with read/write operations on a single object
  - For object store, create multiple instances of the same protocol

- What are the possible specifications for a system with multiple clients and a read/write interface?

On Interprocess Communication

Leslie Lamport

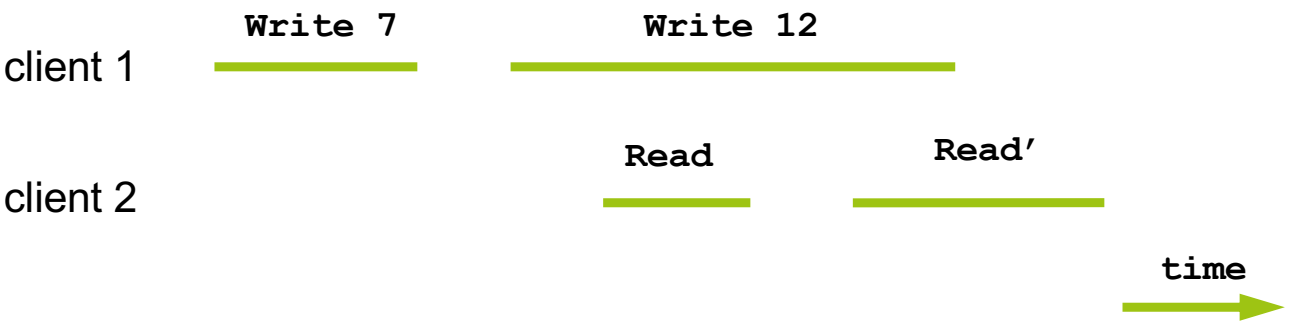
December 25, 1985

Distributed Computing 1, 2 (1986), 77-101.
Also as DEC SRC Research Report 8.

Specifications for a read/write object

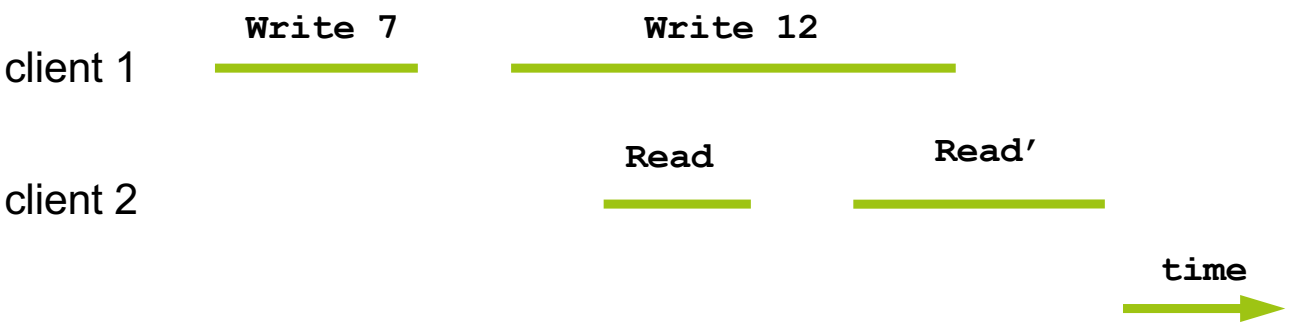
- Lamport's paper specifies semantics for single-writer, multiple-reader
  - Therefore, writes are not concurrent with one another
- Safe semantics
  - Reads that are not concurrent with any writes return the most recently written value
  - Nothing is required from reads that are concurrent with writes
- Regular semantics
  - Reads that are not concurrent with any writes return the most recently written value
  - Reads that are concurrent with writes return either the old or the new value
- Atomic semantics (also applies to multiple writers)
  - There is a total order of all operations, consistent with real-time order, such that reads return the most recently written value according to that order
- Simple generalization of safe/regular to multiple writers:
  - There is a total order of writes consistent with real-time order, which is used to define "most recent" above

Examples



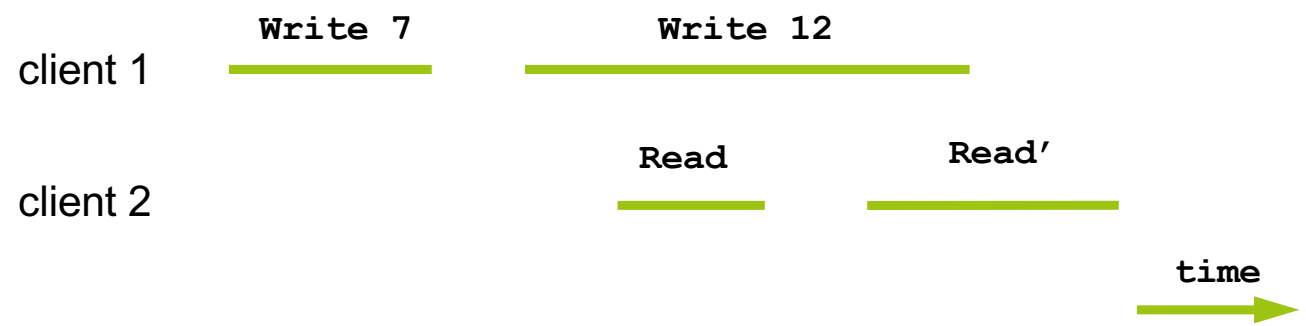
- Read returns 12
- Read' returns 7
- Safe?
- Regular?
- Atomic?

Examples



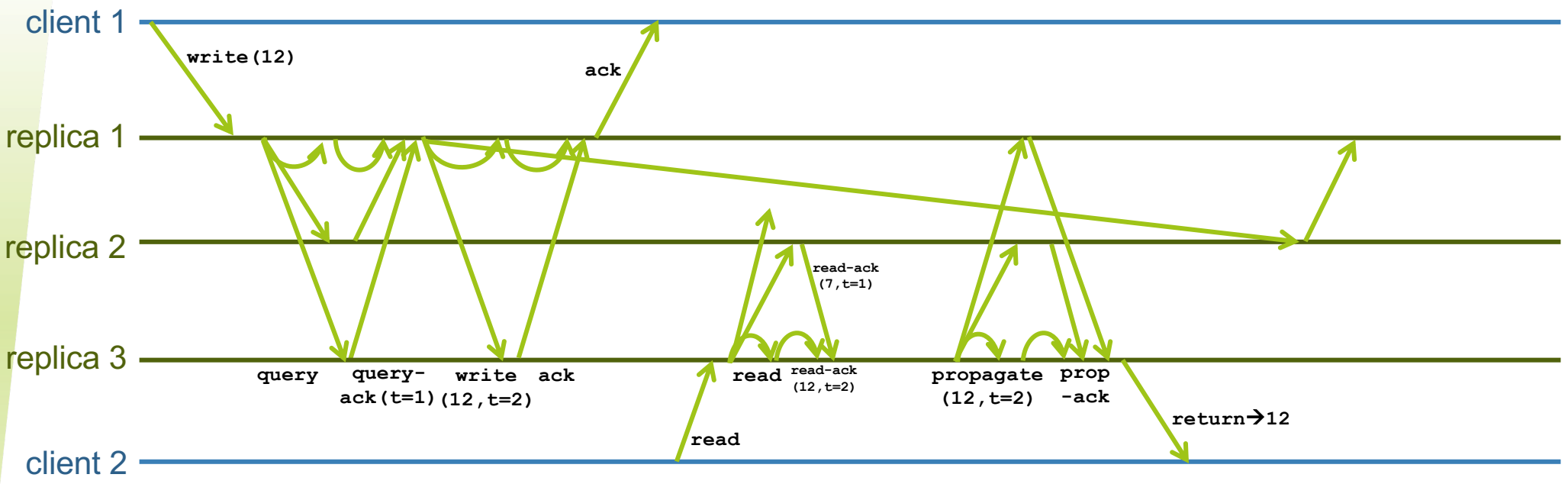
- Read returns 7
- Read' returns 7
- Safe?
- Regular?
- Atomic?

Examples

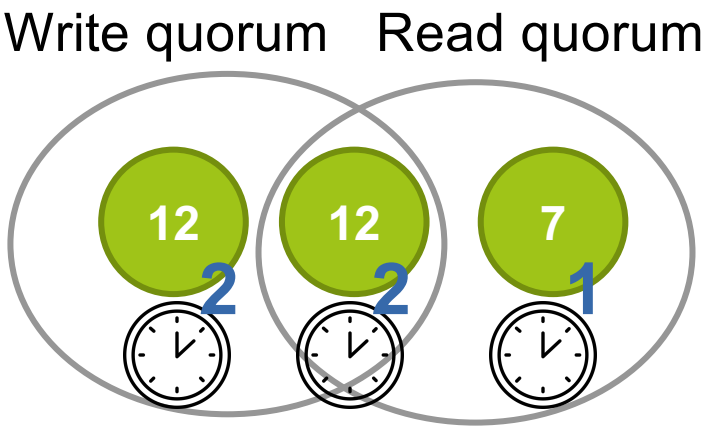


- Read returns 12
- Read' returns
- Safe?
- Regular?
- Atomic?

Crash fault-tolerant (CFT) implementation: ABD protocol (seen yesterday in Prof. Keidar's lecture)



Quorum intersection in CFT



Byzantine Fault Tolerance (BFT)

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of generals and possible traitors. Applications of the solutions to reliable computer systems are then discussed.

Byzantine fault model

- Faulty processes may deviate arbitrarily from the algorithm assigned to it:
  - due to both benign/unintentional faults, e.g., bugs
  - …as well as malicious faults: attacks, collusions
  - …and of course crashing or omitting steps
- Maximum number of Byzantine faults: f
- More expensive to tolerate than crashes:
  - larger number of replicas
  - cryptographic primitives

Digital signatures vs. authenticators

- Digital signature
  - Creator of message m has <PubKey,PrivKey> pair
  - Sign: m → { Hash(m) }$_{PrivKey}$
  - Verify: Decrypt with PubKey corresponding to PrivKey, compare hashes
  - Anyone who has access to PubKey can verify that the message was generated by its creator

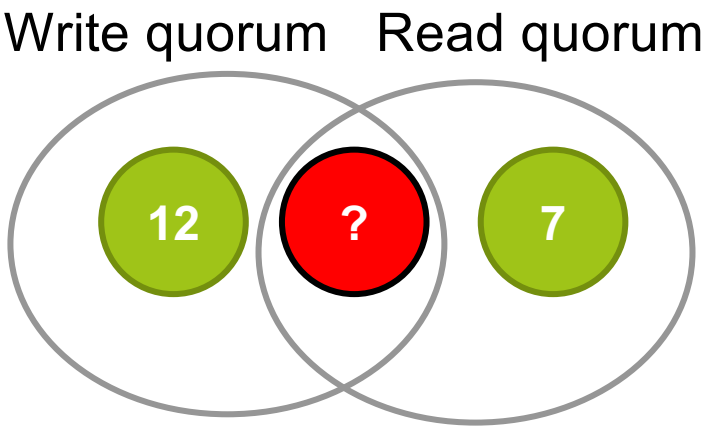
- MAC (Message Authentication Code)
  - Communicating processes i and j share a common secret K
  - Create authenticator: m → H(K||m)
    - Vulnerable to length extension attack, in practice uses H((K' xor *opad*)||H(K' xor *ipad*)||m))
  - Verify: same as create, compare values

- Unlike MACs, digital signatures are transferable (offer non-repudiation)
- But signatures are computationally much more expensive

Authenticated channels

- In the Byzantine model, we assume channels are authenticated
  - e.g., by appendng HMACs to every message that is sent
- Digital signatures will appear explicitly in the protocol description

Previous quorums no longer work

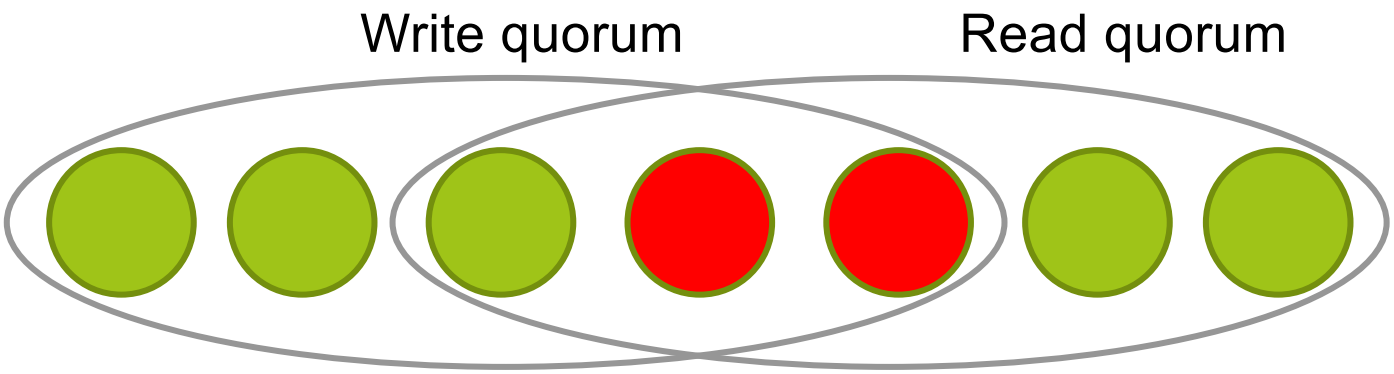


Deriving Byzantine Quorum Systems (Called Dissemination Quorum System in the paper)

- Need **at least** one non-faulty replica in the intersection between any two quorums
- How to achieve this?
- Q → quorum size
- What is size of intersection?
- $N - (N - Q) - (N - Q) = 2Q - N$
- To ensure that it contains at least one non-faulty replica: $2Q - N > f \rightarrow Q > (N + f) / 2$
- Furthermore, we need to ensure there is always a quorum available despite f unreachable replicas:
- $Q <= N - f$

- These two equations imply:
- $N \geq 3f + 1$
- Or, with minimal replication costs: N=3f+1, Q=2f+1

Byzantine (dissemination) quorum, f=2 → N=7

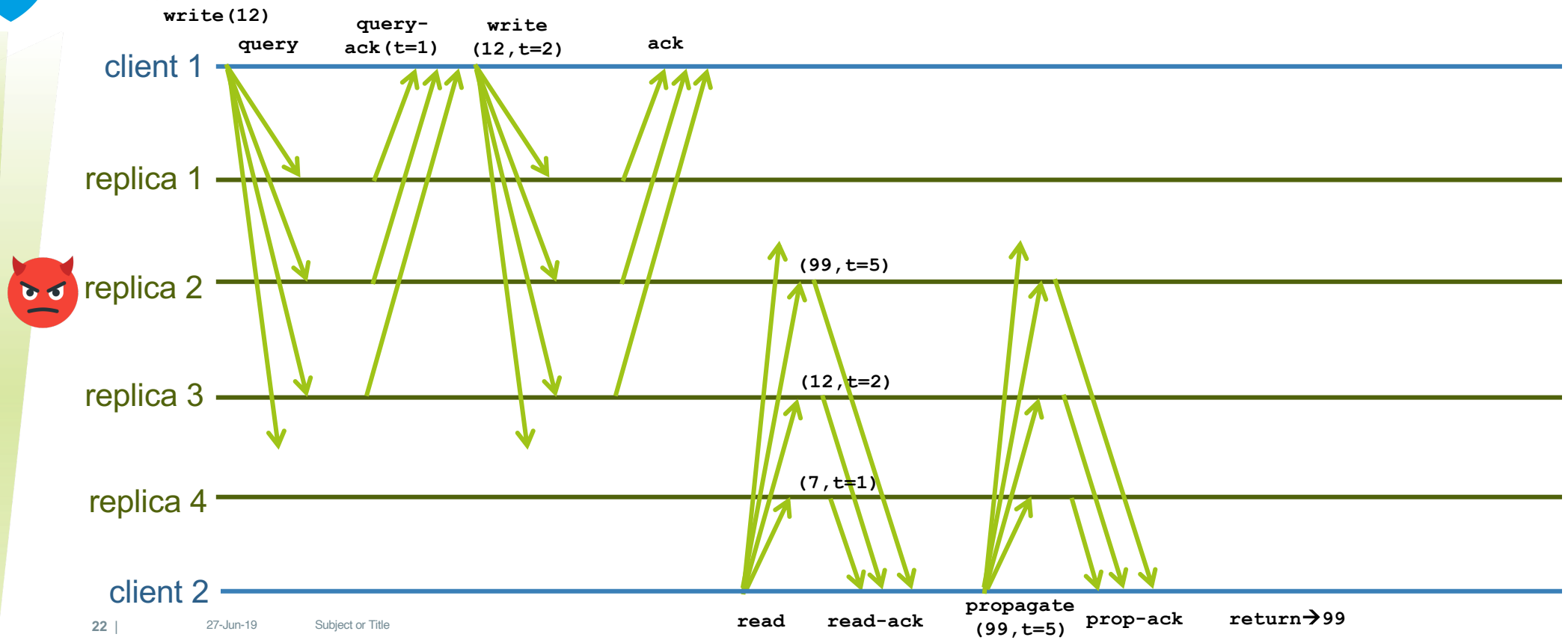


For the same number of replicas (N=7), what happened if quorums were smaller? And what if the quorums were larger?

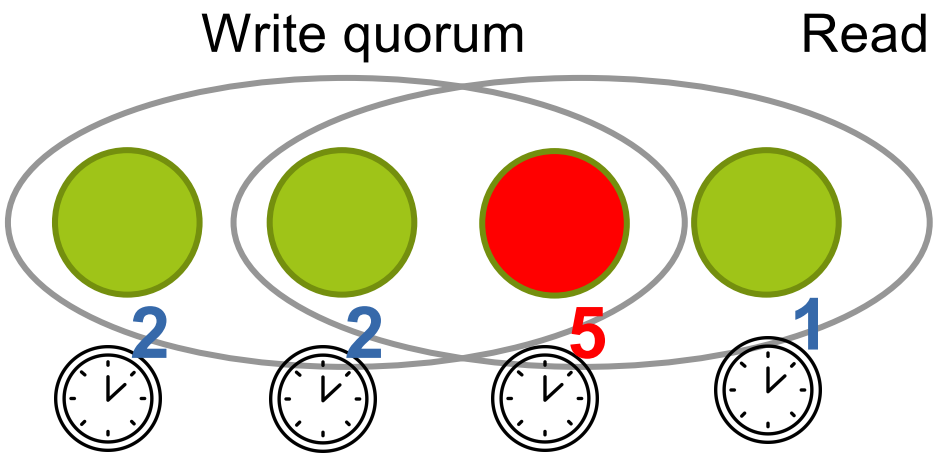
ABD Protocol: CFT → BFT

- Processes communicate through authenticated channels
- Protocol should be driven by client, not through a proxy replica
  - Client cannot trust a single replica's reply
- If no other changed are made to the ABD protocol, what is the required quorum / replica group size?
- Is the previous change to quorums sufficient?

Problematic execution



Problem: intersection was insufficient



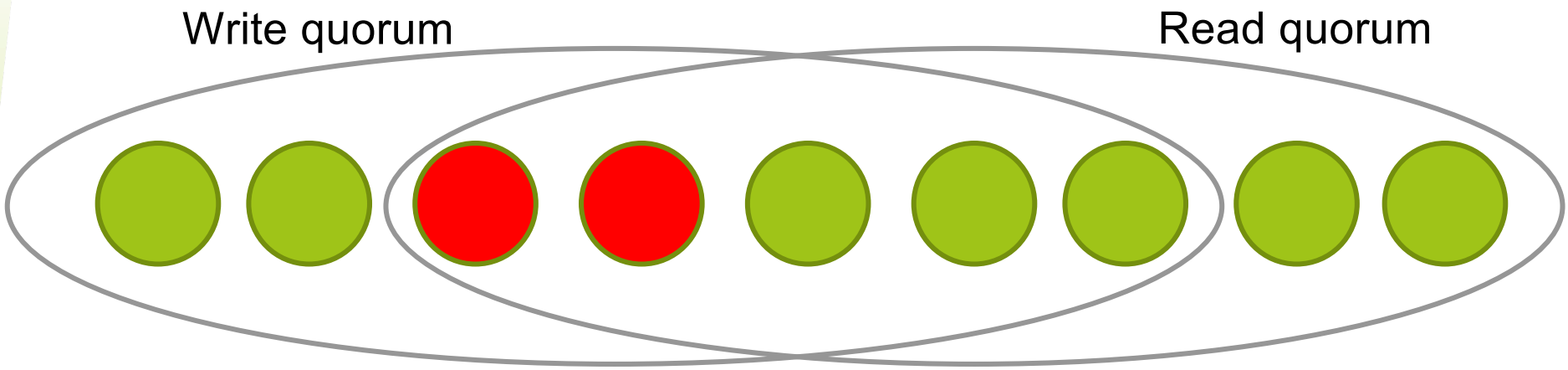
what is the required intersection condition, without further protocol changes?

Masking quorum systems

- Need **a majority** of non-faulty replicas in the intersection between any two quorums
- How to achieve this?
- Q → quorum size
- What is size of intersection?
- N – (N – Q) – (N – Q) = 2Q – N
- To ensure that it contains at least one non-faulty replica: 2Q – N > **2f** → Q > (N + 2f) / 2
- Furthermore, we need to ensure there is always a quorum available despite f unreachable replicas:
- Q <= N – f

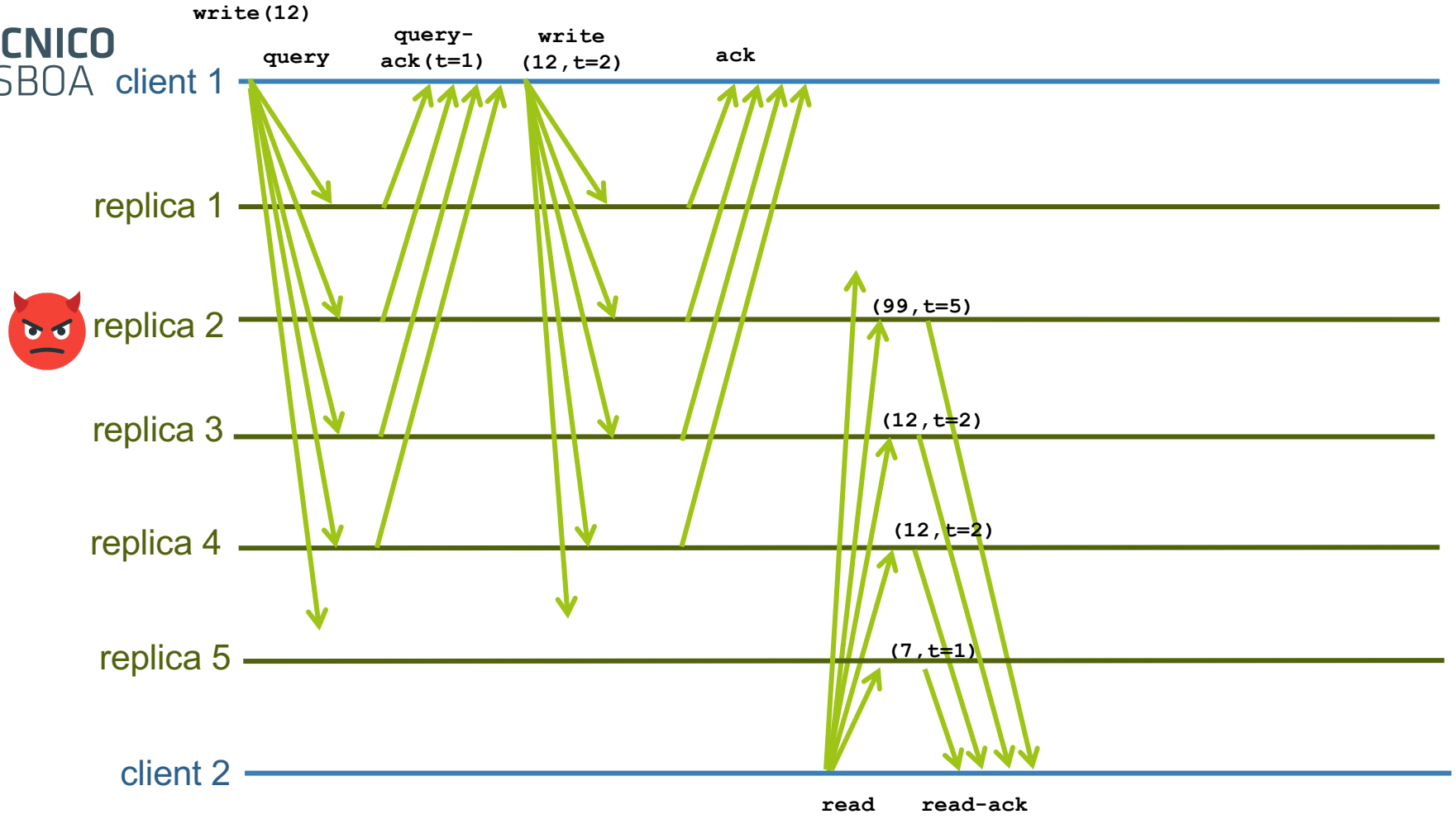
- These two equations imply:
- N ≥ 4f + 1
- Or, with minimal replication costs: N=4f+1, Q=3f+1

Masking quorum system, f=2 → N=9



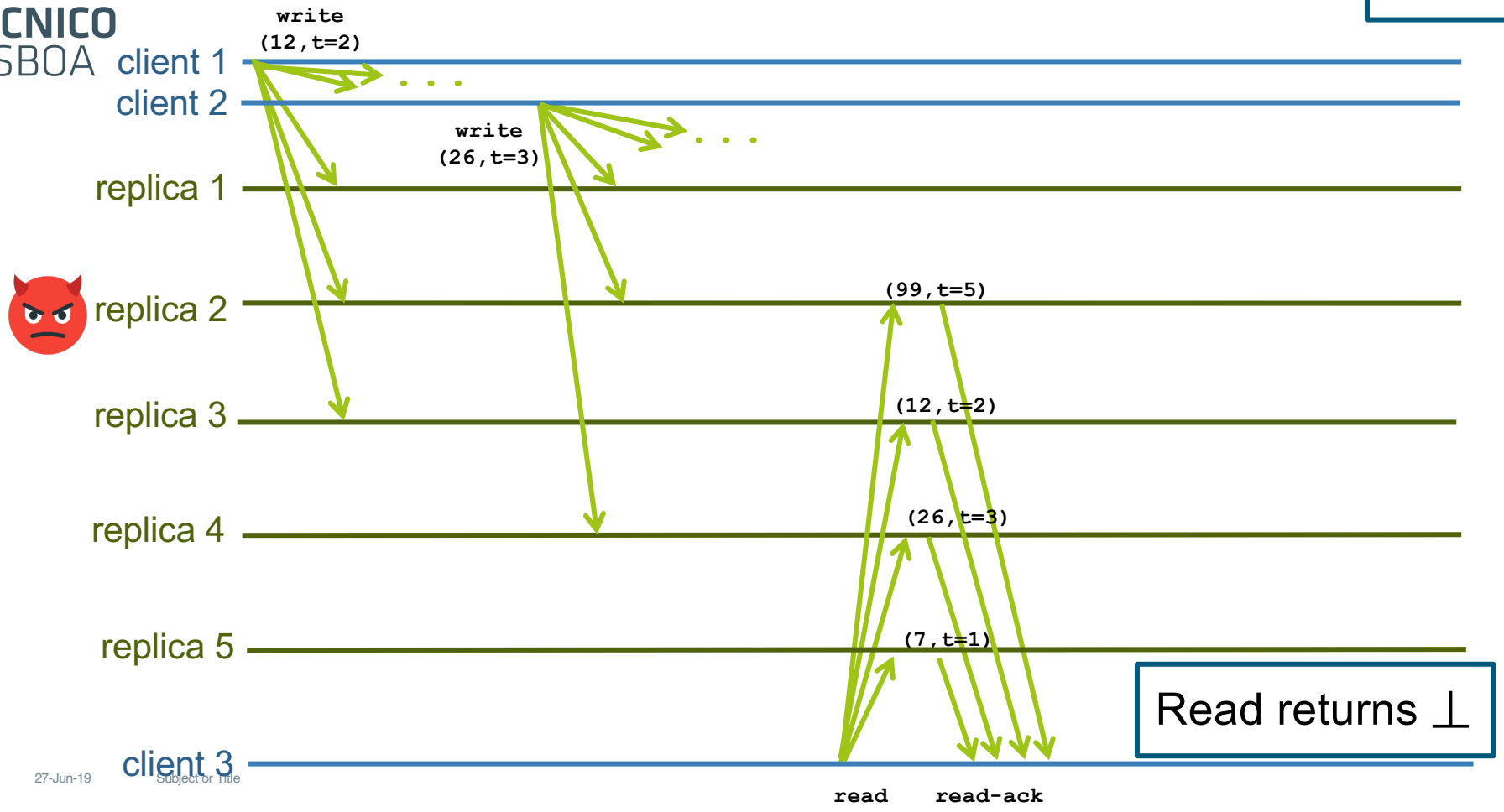
Reads accept only values returned by at least f+1 replicas

Does this solve the prior problematic execution?



What is the resulting semantics for this protocol?

Safe semantics



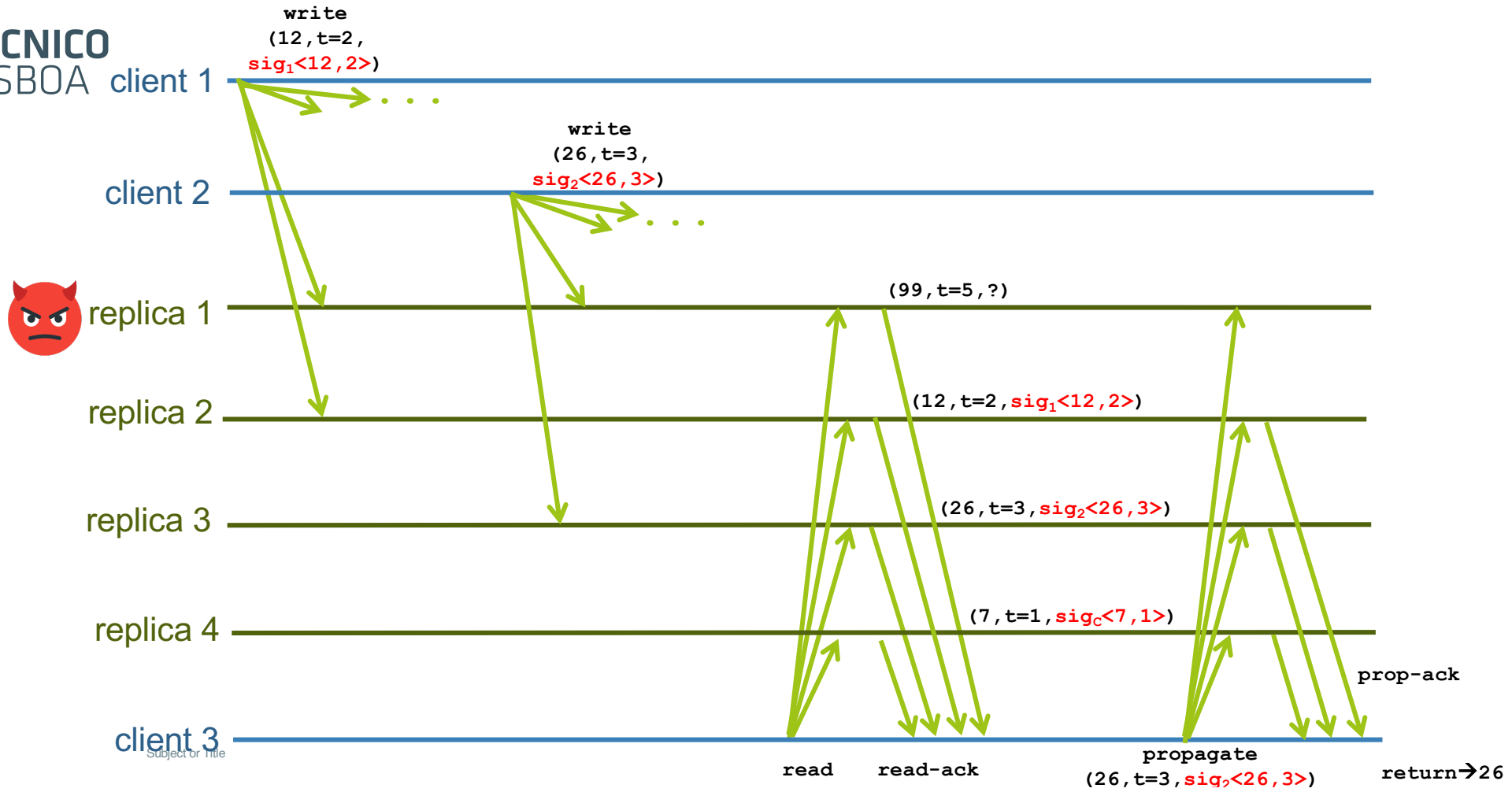
Read returns ⊥

How to regain atomic semantics?

- Problem in previous execution comes from being hard to filter out responses made up by Byzantine replicas
- Solution: make it impossible to make up such responses. How?
- Self-verifying information
- Extend protocol with the following:
  - Each client has <PubKey, PrivKey> pair
  - Each process know the Public Keys of all clients
  - Clients sign new <value,timestamp> pairs before writing to replicas
  - Values that are not correctly signed are discarded (both by replicas and other clients when reading)

- This allows us to use Dissemination Quorum Systems again (N=3f+1, Q=2f+1)

BFT Atomic Read/Write Protocol (ABD-inspired)



Other classes of specifications

- Move beyond the Read/Write model
- We will not look at consensus, but to the Broadcast problem

- Credit for the slides on Byzantine broadcast: Rachid Guerraoui, EPFL

Byzantine consistent broadcast: specification

- **BCB1:** *Validity:* If a correct process p broadcasts a msg m, then every correct process eventually delivers m.
- **BCB2:** *No duplication:* Every correct process delivers at most one message.
- **BCB3:** *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.
- **BCB4:** *Consistency:* If some correct process delivers a message m and another correct process delivers a message m′, then m = m′.

Note: “Correct” means non-Byzantine-faulty

Byzantine Consistent Broadcast: Algorithms

- Byzantine Consistent Broadcast
  - If the sender s is correct then every correct process should (at some point in time) deliver m.
  - If s is faulty, then every correct process delivers the same message, if it delivers one at all.
    - i.e., correct processes are not guaranteed to deliver the same set of messages
- Two algorithms:
  - Authenticated Echo Broadcast
    - exchanges a quadratic number of messages, uses MACs
  - Signed Echo Broadcast
    - linear no. of messages but uses digital signatures (costly)

Authenticated Echo Broadcast

- exploits Byzantine (dissemination) quorums (N=3f+1, Q=2f+1)
  - two rounds of message exchanges.
  - 1st round: sender disseminates msg to all processes.
  - 2nd round:
    - every process acts as a witness for m
    - resends m in an ECHO message to all others.
    - deliver only when received 2f+1 ECHOs
      - Byzantine processes cannot cause a correct process to *deliver* a message m′ ≠ m.
- relies on authenticated links
  - prevent injection of forged messages

Authenticated Echo Broadcast

- **Implements:**

  ByzantineConsistentBroadcast, with sender s.

- **Uses:**

  AuthPerfectPointToPointLinks, **instance** *al*.

- **upon event** ⟨ *bcb*, Init ⟩ **do**

  sentecho := FALSE;

  delivered := FALSE;

  echos := $[\perp]^N$;

Authenticated Echo Broadcast

- **upon event** ⟨ *bcb*, Broadcast | m ⟩ **do** *// only process s*
  forall q ∈ Π **do**
  trigger ⟨ *al*, Send | q, [SEND, m] ⟩;

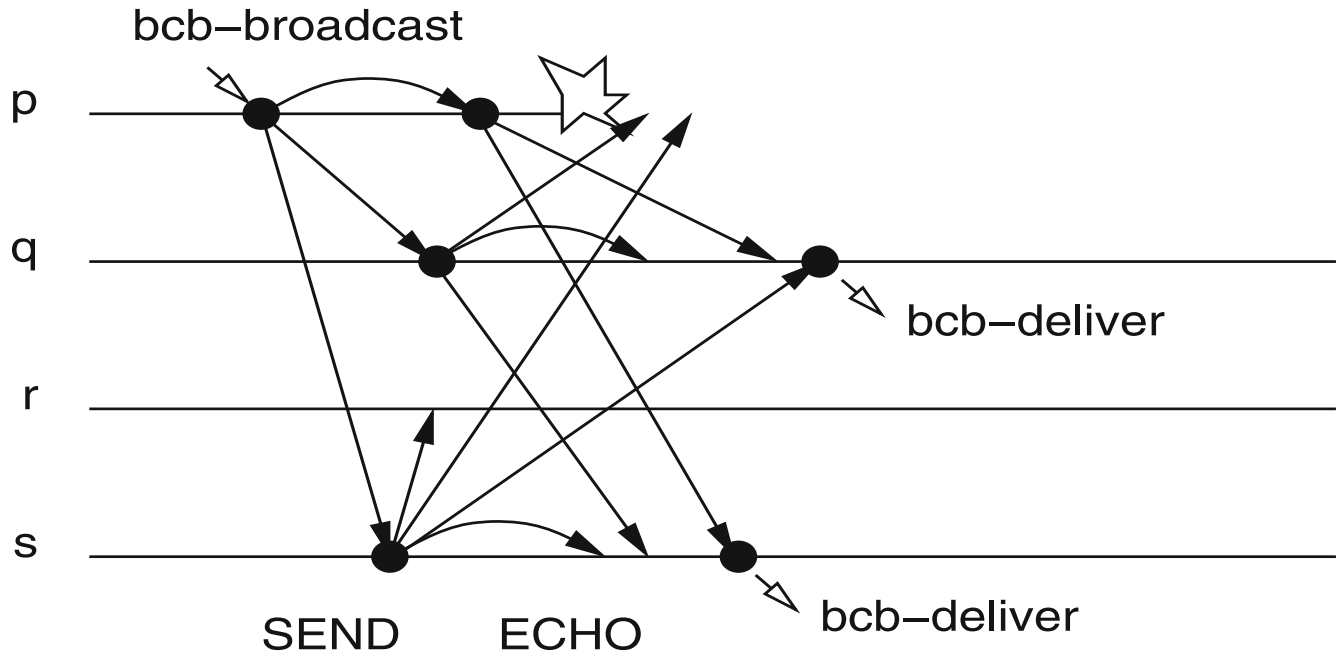
- **upon event** ⟨ *al*, Deliver | p, [SEND, m] ⟩
  such that p = s **and** *sentecho* = FALSE **do**
  sentecho := TRUE;
  forall q ∈ Π **do**
  trigger ⟨ *al*, Send | q, [ECHO, m] ⟩;

Authenticated Echo Broadcast

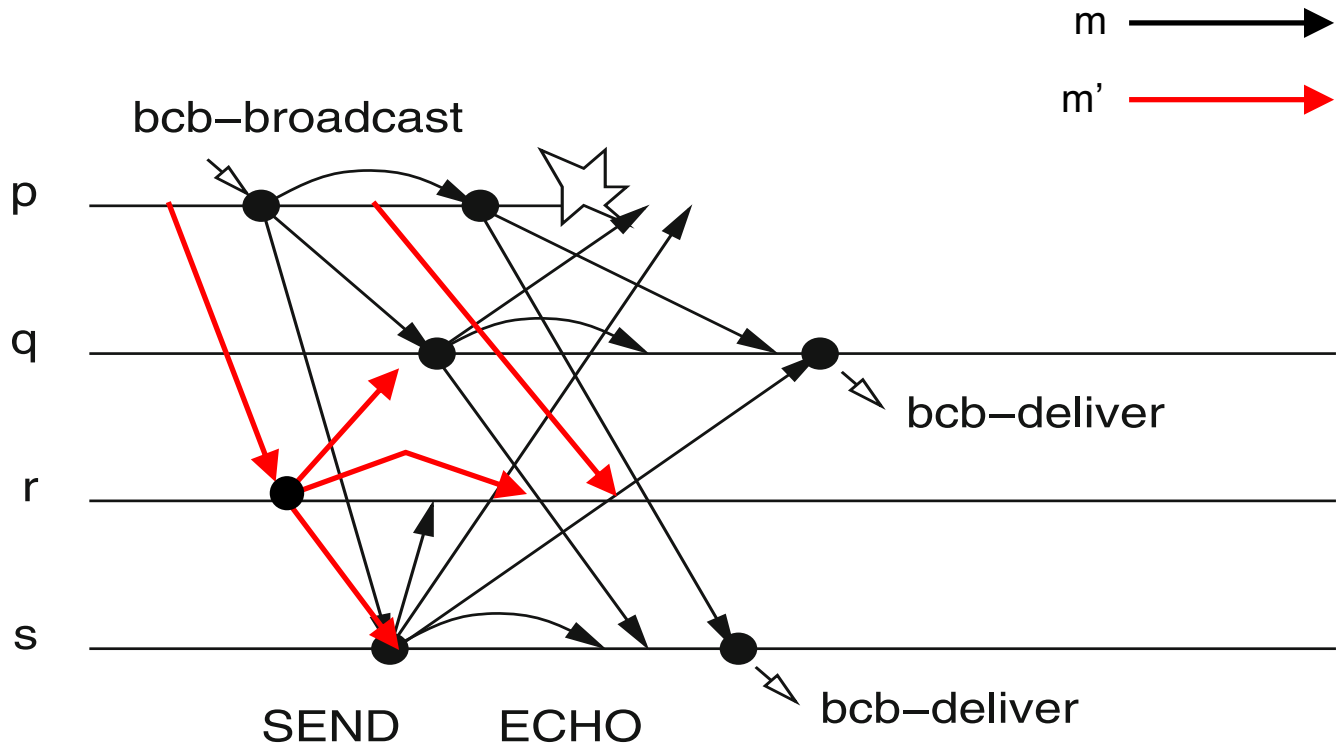
- **upon event** ⟨ *al*, Deliver | p, [ECHO, m] ⟩ **do**
  if *echos*[p] = ⊥ **then**
  echos[p] := m;

- **upon exists** m != ⊥ such that :
  |{p ∈ Π |*echos*[p] = m}| >= 2f+1 **and** *delivered* = FALSE **do**
  delivered := TRUE;
  trigger ⟨ *bcb*, Deliver | s, m ⟩;

Authenticated Echo Broadcast



Authenticated Echo Broadcast



r can never deliver m', if q and s deliver m
(r can get at most 2 ECHOes for m', including its own)

Correctness

BCB1: *Validity:* If a correct process p broadcasts a msg m, then every correct process eventually delivers m.

Proof (sketch)

- if the sender is correct, then every correct process sends an ECHO message
- every correct process delivers at least N − f of them.
- N − f >= 2f+1 under the assumption that N = 3f+1

  ➔ every correct process also delivers the message m contained in the ECHO messages

Correctness

- **BCB2:** *No duplication:* Every correct process delivers at most one message.
- **Proof (sketch):** enforced by the *delivered* variable in the algorithm

Correctness

- **BCB3:** *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.
- **Proof (sketch):** by algorithm construction and the properties of Authenticated Perfect Links

Correctness

- **BCB4:** *Consistency:* If some correct process delivers a message m and another correct process delivers a message m′, then m = m′.
- For a correct process p to deliver some m, it needs to receive ECHO messages for m from a Byzantine quorum.
- Two Byzantine quorums overlap in at least one correct process.
- Assume, by contradiction, that a different correct process p′ that delivers some m′.
  - p′ has received a Byzantine quorum of ECHO messages for m′
  - the correct process in the intersection of the two Byzantine quorums sent different ECHO messages to p and to p′

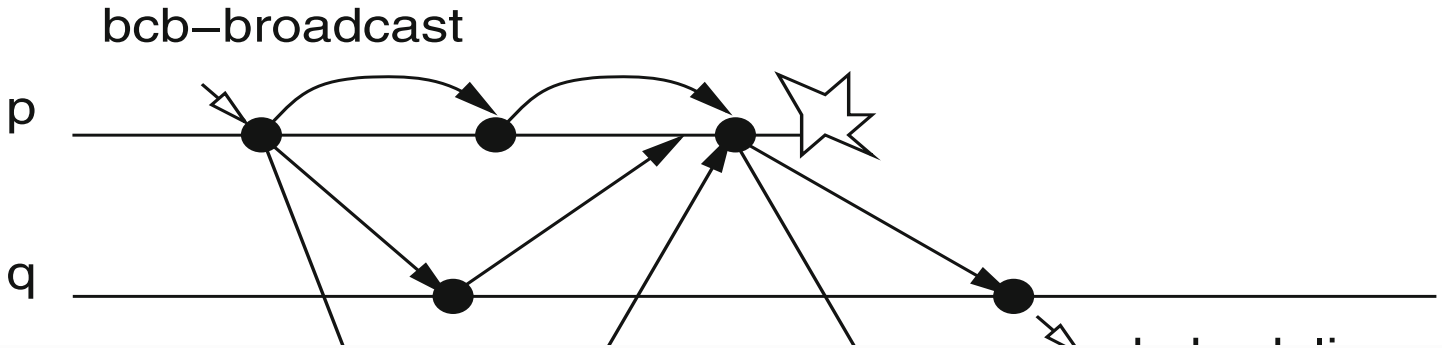
Contradiction found!

Signed echo broadcast

- Exploits digital signatures:
  - more powerful than MACs
  - allow a third process to verify the authenticity of a message sent from a 1st process s to a 2nd process r
  - avoid quadratic number of message exchanges of prev. algorithm

- Basic idea:
  - Witnesses do not authenticate a request by sending an ECHO message to all processes
  - instead, sign a statement that they return to the sender
  - Sender collects a Byzantine quorum of these signed statements and relays them in a third communication step to all processes.

Signed Echo Broadcast



International Symposium on Distributed Computing

DISC 2008: Distributed Computing pp 16-31 | Cite as

Matrix Signatures: From MACs to Digital Signatures in Distributed Systems

Authors | Authors and affiliations

Amitanand S. Aiyer, Lorenzo Alvisi, Rida A. Bazzi, Allen Clement

Conclusions

- Fault models matter
- Provide a tradeoff between correctness and performance
- Can provide strong security despite adversarial behavior
- Traditionally the world was split between Crash and Byzantine faults
- Intermediate models may be useful for addressing real-world problems

Questions?