Byzantine Agreement: Applications

- Diego Puppin

Real-world applications

- Software errors and attacks are more and more common
- Byzantine Agreement can bring reliability in case of faults, attacks etc.
- Faulty/malicious nodes can exhibit byzantine behavior (wrong, missing, late messages)
- We have seen an exponential algorithm, let’s try something better

Assumptions

- Network can delay, lose, duplicate, re-order messages freely
- Faulty nodes may behave arbitrarily
- Independent failures
  - Diversified code!!
- Cryptographic protection to messages

Safety

- A replicated service is SAFE if is satisfies linearizability:
- It behaves like a centralized system that executes operations atomically, one at a time
- Safety is not enough: bad clients can destroy data on FS
  - Access control is needed

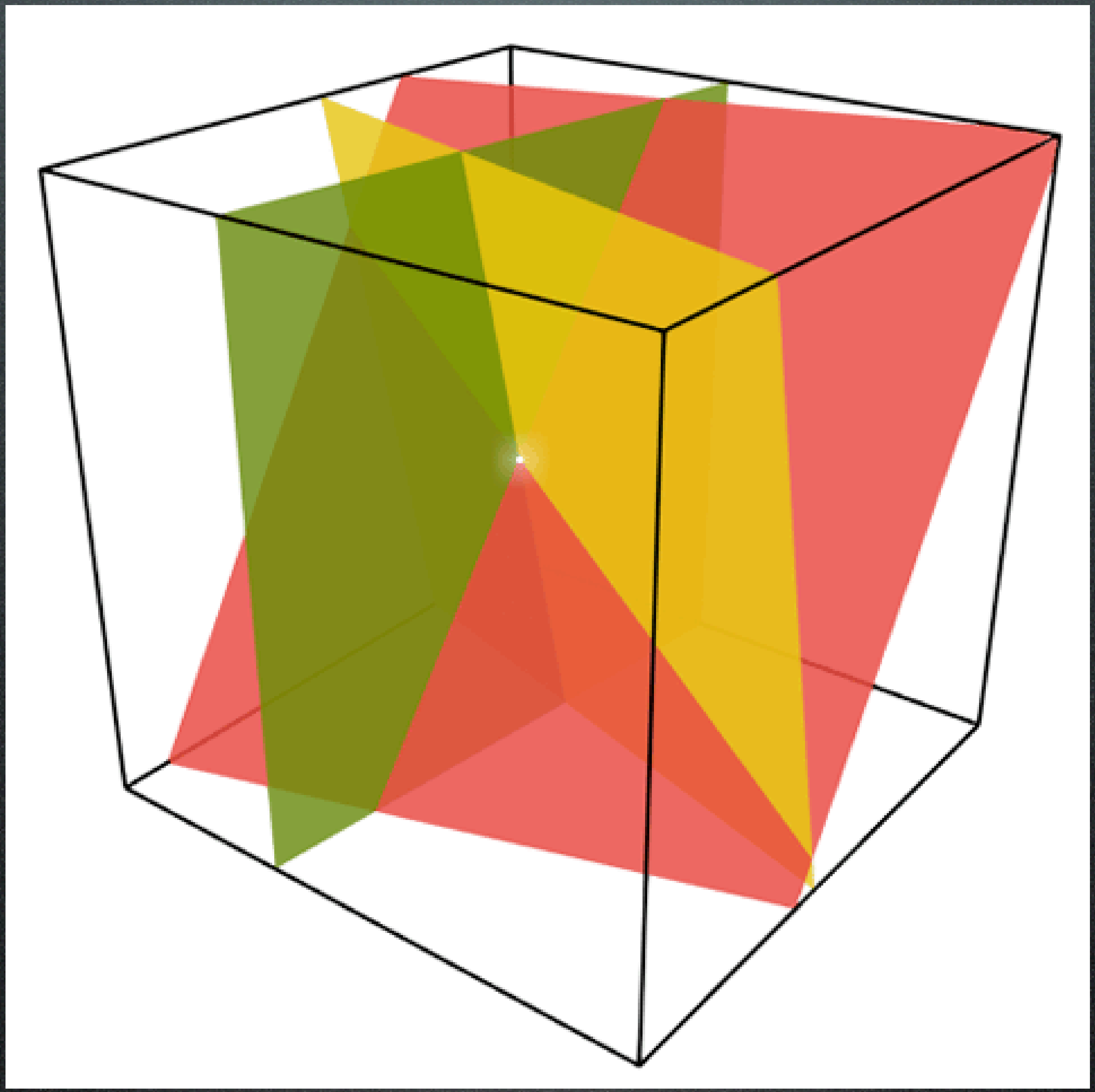
Liveness

- Clients eventually receive replies, if at most are faulty, and delay(t) does not grow
- Synchrony is needed to guarantee liveness

Fault tolerance

- 3f+1 copies are needed to survive f faults
- Privacy is not guaranteed:
  - A faulty process could share data
- Some solutions available using secret sharing schemes

Secret sharing



Algorithm

- Let's have a set R of replicas, $|R| = 3f+1$
- The replicas go through views
- In view v, replica v mod $|R|$ is considered primary (the other backups)
- View is changed when the primary replica fails (appears to fail)

Algorithm

- A client send a request to invoke an operation to the primary
- The primary multicasts the req to backups
- Replicas execute and reply to client
- The client waits for f+1 identical results

- Replicas are deterministic
- They start from the same state
- All non-faulty replicas agree on a total order of requests

Client (1)

- C sends a request <REQ, op, time, c>c to primary
- Primary broadcasts
- A replica i replies <REPLY, view, time, c, i, res>i
- Clients wait for f+1 results with the same time and res

Client (2)

- If no results (before timeout), REQ is broadcast to all replicas
- Replicas elaborates (or re-send) REPLY and then relies the message to primary
- If primary doesn't broadcast, it may be faulty

Primary's role

- When p receives REQ, there is an atomic three-phase broadcast
- pre-prepare, prepare, commit

pre-prepare

- p gives an ID n to REQ
- m = <REQ, op, time, c>c   - dm is the digest
- p multicasts <<PRE-PREPARE, v, n, dm>p, m>
- Backups accept if:
  - signature is ok
  - v number is ok
  - <v, n> is new

prepare

- If backup accepts, it multicasts
  - <PREPARE, v, n, dm, i>i
  - PREPREPARE and PREPARE msgs are logged
- If not, NOP

prepared()

- prepared(m, v, n, i) TRUE if replica i has logged: one pre-prepare msg and 2f prepare msgs
- Non faulty replicas agree on an order (given by n)
- There cannot be prepared(m1, v, n, i) and prepared(m2, v, n, i)

commit

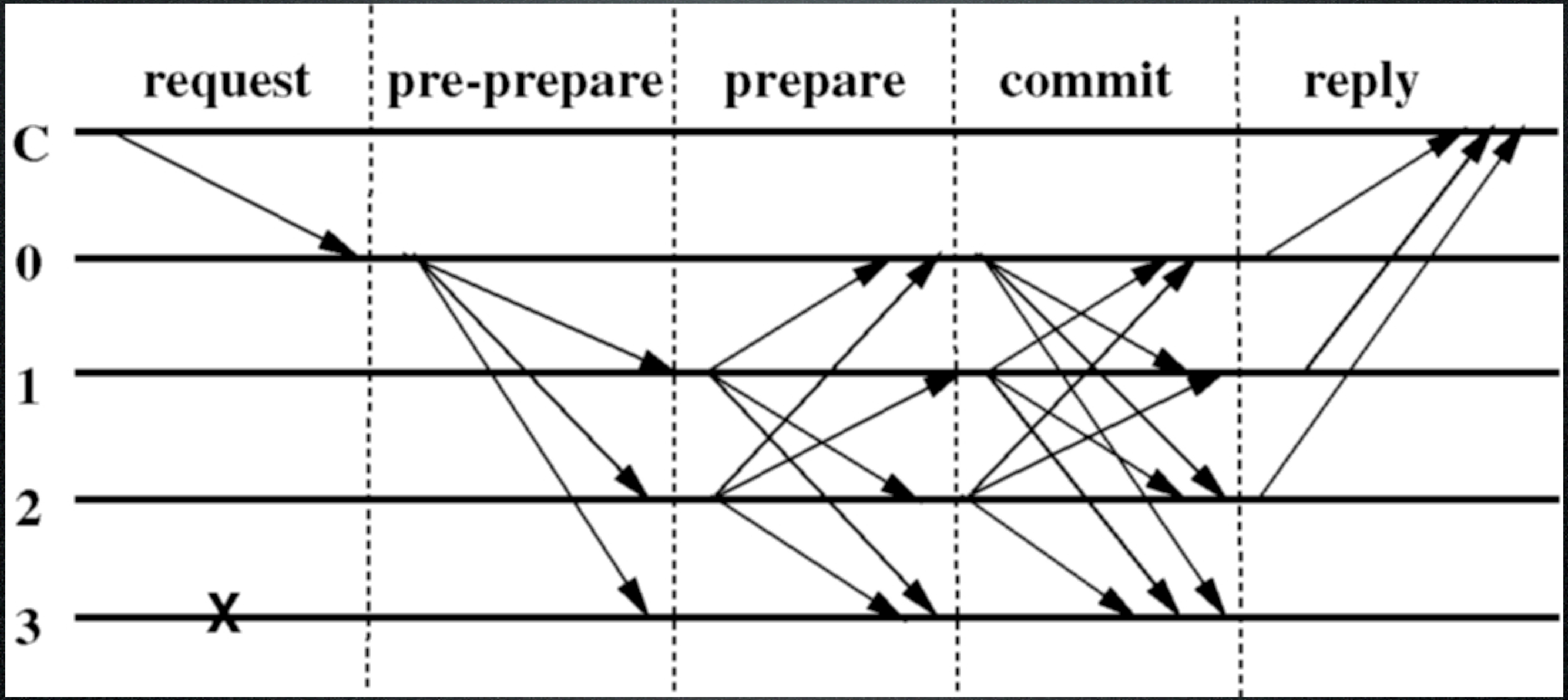
- When prepared(m, v, n, i) replica i broadcasts <COMMIT, v, n, dm, i>i   to replicas
- Replicas accept COMMIT and log it if v,n and signatures are ok

committed()

- committed(m, v, n) TRUE if prepared(m, v, n, i) is valid for f+1 non faulty replicas
- committed-local(m, v, n, i) TRUE if prepared (m, v, n, i) and i accepted 2f+1 COMMIT
- ∃i non faulty.committed-local i => committed
- Replicas agree on n even if they are in different views v
- Also, if there is one committed-local, at least f +1 non faulty will also commit-local

Last round

- Those i committed-local will reply to client
- Client will accept results when f+1 replies agree



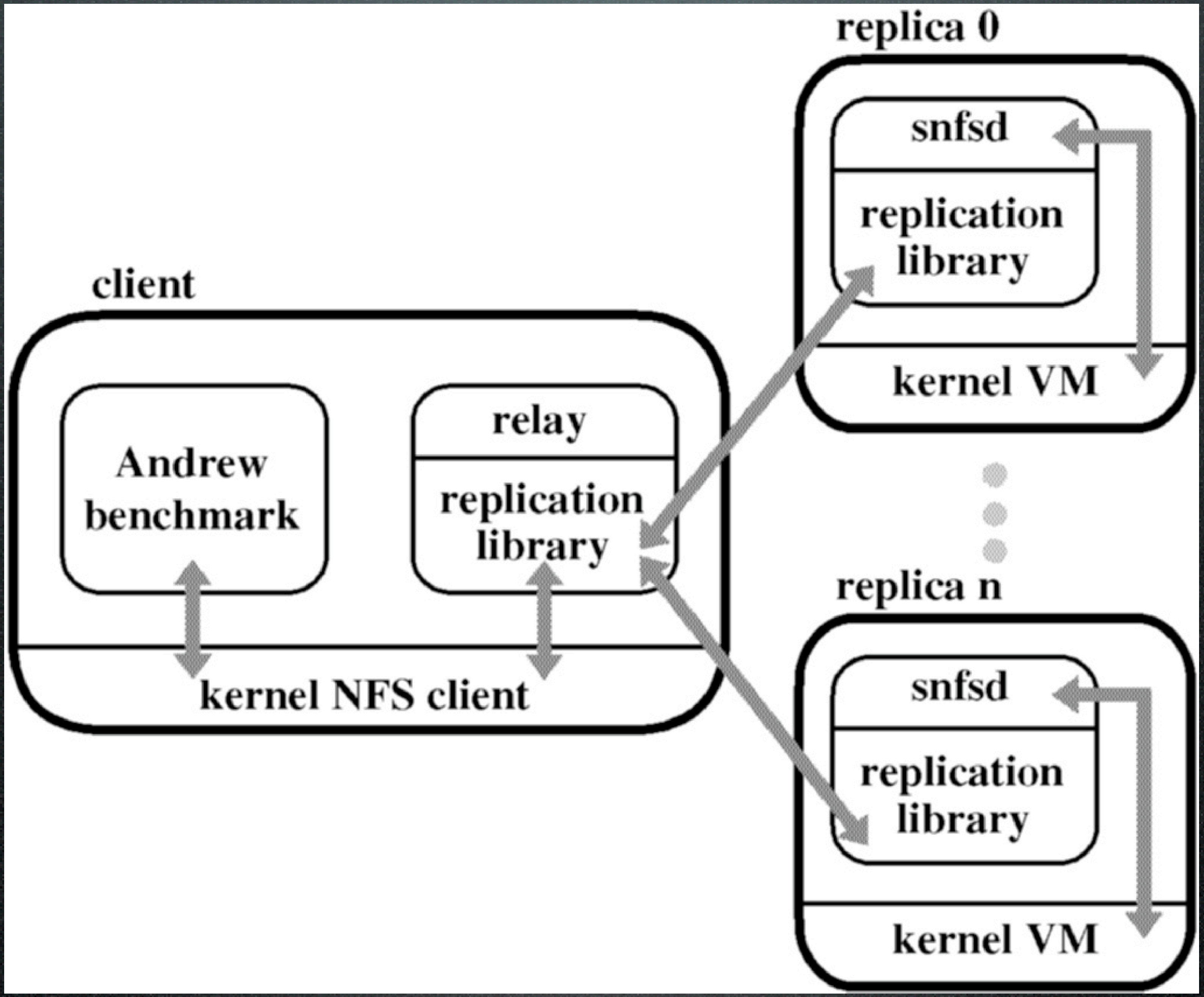
View change

- View can change to ensure liveness if primary fails
- A timeout starts when pre-prepare is received
- If commit is NOT executed within timeout, replica i sends <VIEW, v+1, n, C, P, i>i
- C and P are checkpoints and outstanding messages (see papers)

New primary

- When the new primary p1 = v+1 mod |R| receives 2f valid view changes
- It broadcasts <NEW, v+1, V, O>p1
- V is the set of view-change requests
- O is again related to checkpoint
- Backup verifies NEW

Byzantine NFS



Implementation

- Unmodified NFS server and clients
- At user level, the application uses a replication library to manage the protocol
- File system is implemented in memory in replicas
- Optimization: R/O requests are broadcast directly to all replicas

arg./res. (KB)	replicated		without replication
	read-write	read-only	
0/0	3.35 (309%)	1.62 (98%)	0.82
4/0	14.19 (207%)	6.98 (51%)	4.62
0/4	8.01 (72%)	5.94 (27%)	4.66

phase	BFS		BFS-nr	NFS-std
	strict	r/o lookup		
1	0.55 (57%)	0.47 (34%)	0.35	1.75
2	9.24 (82%)	7.91 (56%)	5.08	9.46
3	7.24 (18%)	6.45 (6%)	6.11	5.36
4	8.77 (18%)	7.87 (6%)	7.41	6.60
5	38.68 (20%)	38.38 (19%)	32.12	39.35
total	64.48 (26%)	61.07 (20%)	51.07	62.52