The Paxos Consensus Protocol

Sakari Ellonen

Aalto University

1st December 2010

Outline

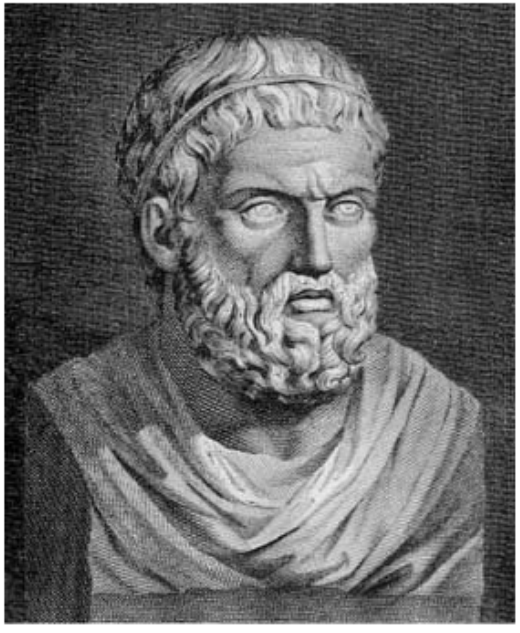
1. **Introduction**
   - Key Concepts
   - Relevance to Computer Science
2. **The Paxos Protocol**
   - Guarantees
   - Messages
   - Temporary information
   - The Protocol Formally
3. **Examples**

The Consensus Problem

How to reach a consensus (everybody agrees on something) in a chamber of priests, when

- Priests may enter and leave at random times
- Priests may forget everything during their absence
- Priests can communicate only via unreliable messengers (messages lost or repeated, but not corrupted)

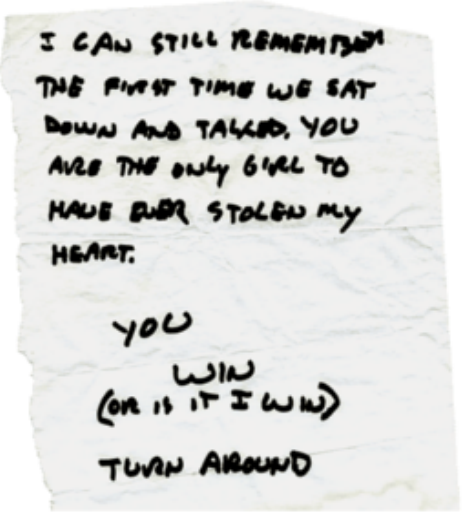
Key Concepts



priest



ledger,
persistent information



piece of paper,
modifiable,
may be lost

Key Concepts continued

Concepts continued

- *Decree* or law, e.g. “Painting on temple walls is forbidden”
- *Ballot*: a round of voting (b, d, S), where b is a ballot number, d a decree, S any majority of priests.
- The decree is *passed* iff every priest in S votes for it. Voting against is not possible!

What the priests do

The same decrees must appear in the ledger of every priest in the same order.

Relevance to Computer Science

In a distributed database

- servers are state machines
- a state transition corresponds to a decree passed
- unless the decrees are passed consistently, the state machines end up in different states

The Paxos Guarantees

Guarantees

The Paxos protocol guarantees that

- No two passed decrees have the same ballot number (consistency)
- If a majority of the priests were in the Chamber, and no one entered or left the Chamber for a sufficiently long period of time, then any decree proposed by a priest would be passed, and every decree that had been passed would appear in the ledger of every priest in the Chamber (progress)

The Paxos Protocol

Messages in the protocol

- *NextBallot(b)*, “prepare yourselves to vote in ballot *b*”
- *BeginBallot(b,d)*, “let’s have a ballot *b* on decree *d*”
- *LastVote(b’,q,d)*, “the last time *q* voted was in ballot b' on degree *d*”
- *Voted(b,q)*, “priest *q* voted for ballot *b*”
- *Success(d)*, “decree *d* was successfully passed”

The Promise

When a priest is preparing for a ballot *b*, he promises not to vote in any ballot numbered less than *b*. This is a key property of the Paxos protocol.

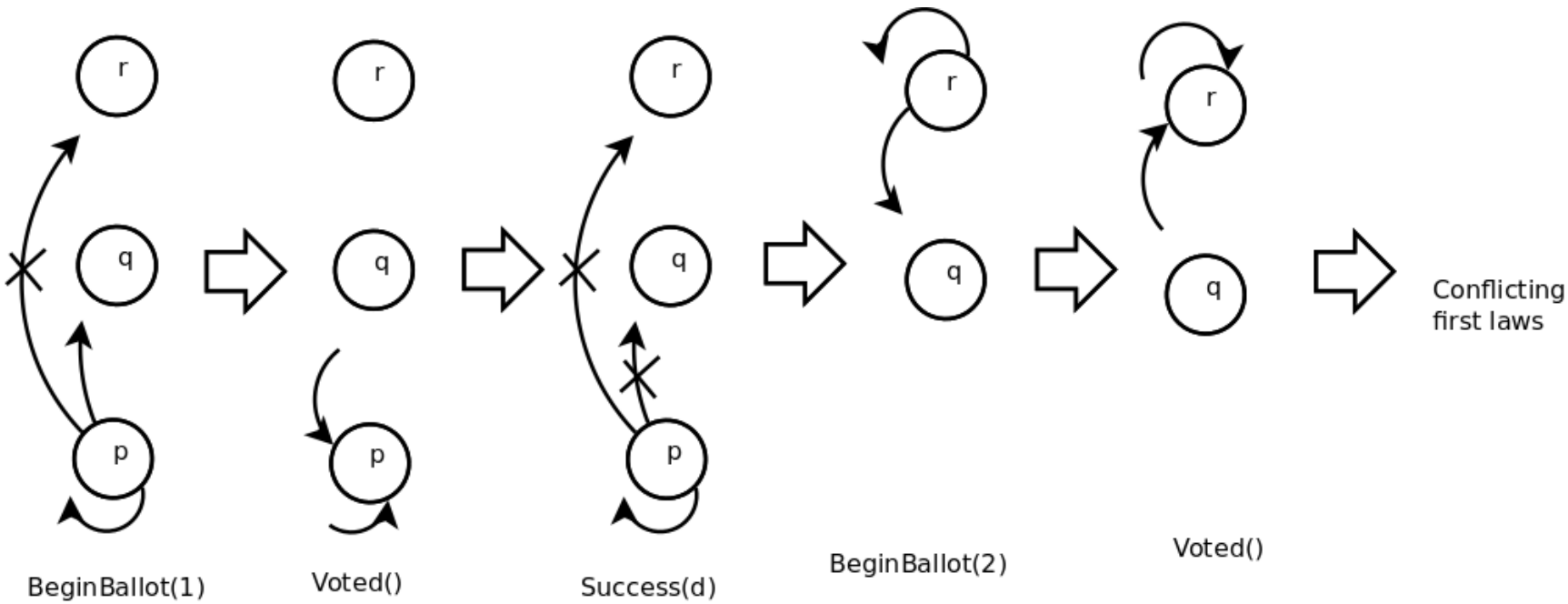
Two-stage initiation

Why do we need the *NextBallot(b)* message? Why not just send *BeginBallot(b,d)* messages and if a majority votes in favor, send a *Success(d)*? The problem is in the ordering.

- Suppose three priests p, q, r want to decide the olive tax
- Priest p sends *BeginBallot(1, tax 5%)* to everybody.
- p and q vote in favor, so it is passed.
- p sends a *Success(5%)* to everybody and leaves for holiday, but r and q never receive the confirmation
- Meanwhile, r sends *BeginBallot(2, tax 7%)* and q thinks the 5% tax never passed, so they agree. $\rightarrow$ immodifiable ledgers inconsistent on the first passed law!

Need some way of deciding what we are currently voting on.

Conflict


r
q
p
BeginBallot(1)
Voted()
Success(d)
BeginBallot(2)
Voted()
Conflicting
first laws

Temporary information

Priests maintain three records on the pieces of paper. They are initially set to $-\infty$.

- *lastTried[p]* The number of the last ballot that p tried to initiate
- *prevVote[p]* The vote (b', d) cast by p in ballot b' in which he voted, b' maximal
- *nextBal[p]* The ballot number b in which p is preparing to vote

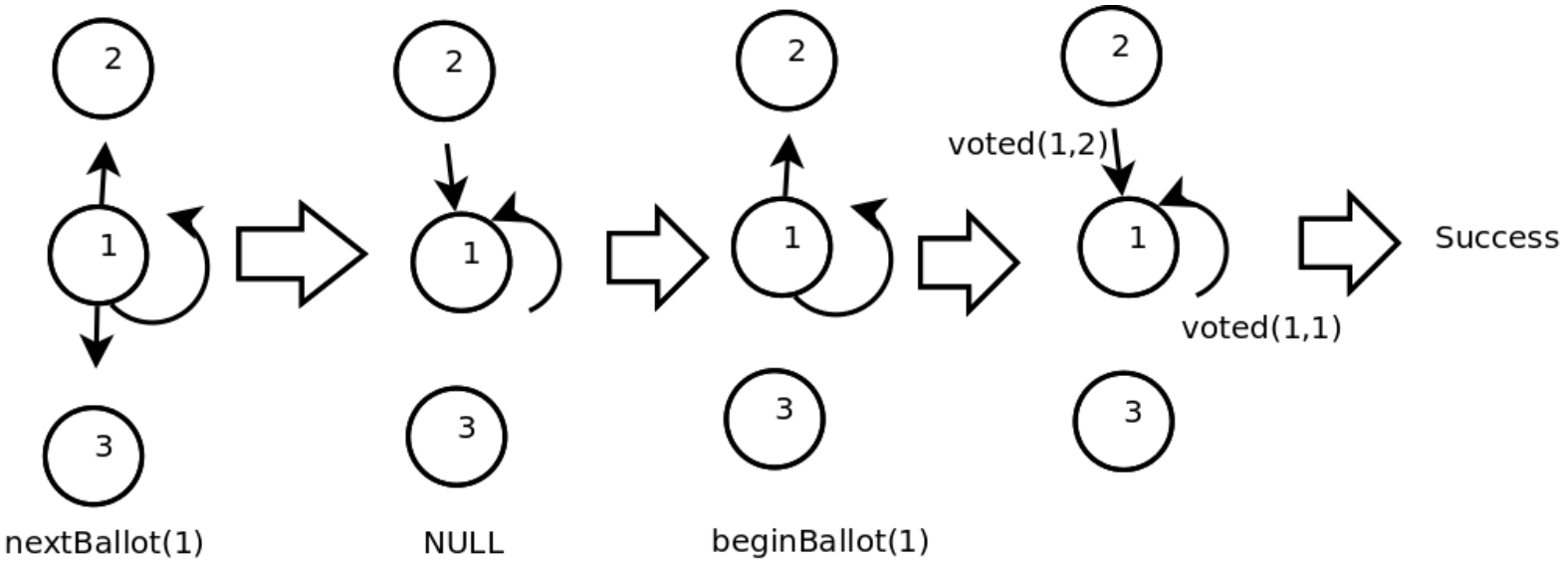
The Paxos protocol

1. Priest p chooses a new $b > lastTried[p]$, sets $lastTried[p] \leftarrow b$, and sends a *NextBallot(b)* to everybody
2. Priest q receives *NextBallot(b)* from p. If $b > nextBal[q]$, q sets $nextBal[q] \leftarrow b$, and sends *LastVote(b',q,d)* to p, where b' and d as in $prevVote[q]$.
3. After receiving a *LastVote()* from every priest in some majority set S, priest p chooses from them the decree d with the highest ballot number, and sends *BeginBallot(b,d)* to everyone in S. d is chosen freely if all replies are $-\infty$.
4. Priest q receives *BeginBallot(b,d)* from p. If $b = nextBal[p]$, priest q votes for b, $prevVote[p] \leftarrow b$, and sends *Voted(b,q)* to p.

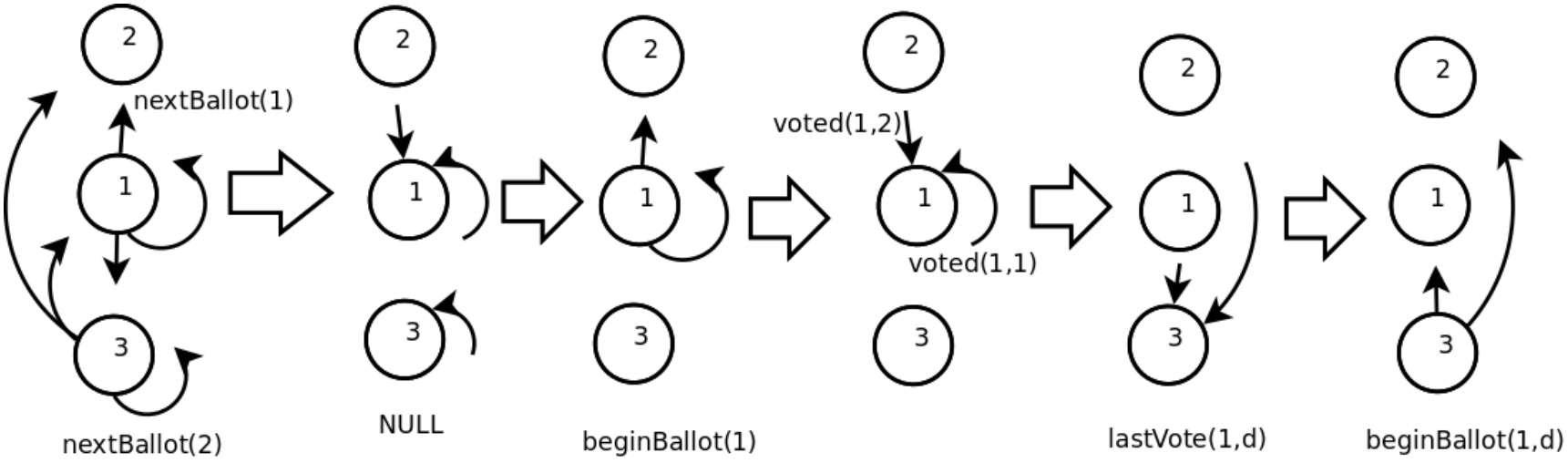
The Paxos protocol continued

5. If *p* has received a *Voted(b,q)* message from every priest in *S*, then he writes the decree *d* in his ledger and sends a *Success(d)* to every priest.
6. Upon receiving a *Success(d)*, a priest enters degree *d* in his ledger.

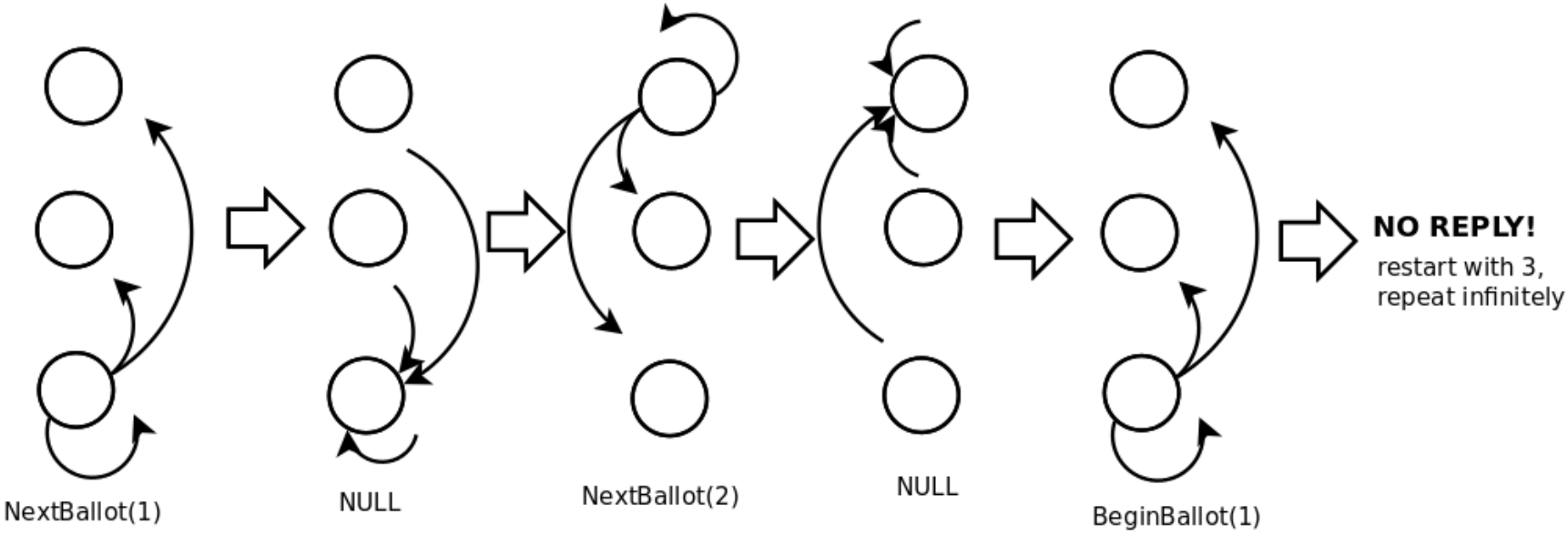
Example 1


2
1
3
nextBallot(1)
2
1
3
NULL
2
1
3
beginBallot(1)
2
voted(1,2)
1
voted(1,1)
3
Success

Example 2


2
nextBallot(1)
1
3
nextBallot(2)
2
1
3
NULL
2
1
3
beginBallot(1)
2
voted(1,2)
1
voted(1,1)
3
2
1
3
lastVote(1,d)
2
1
3
beginBallot(1,d)

Livelock



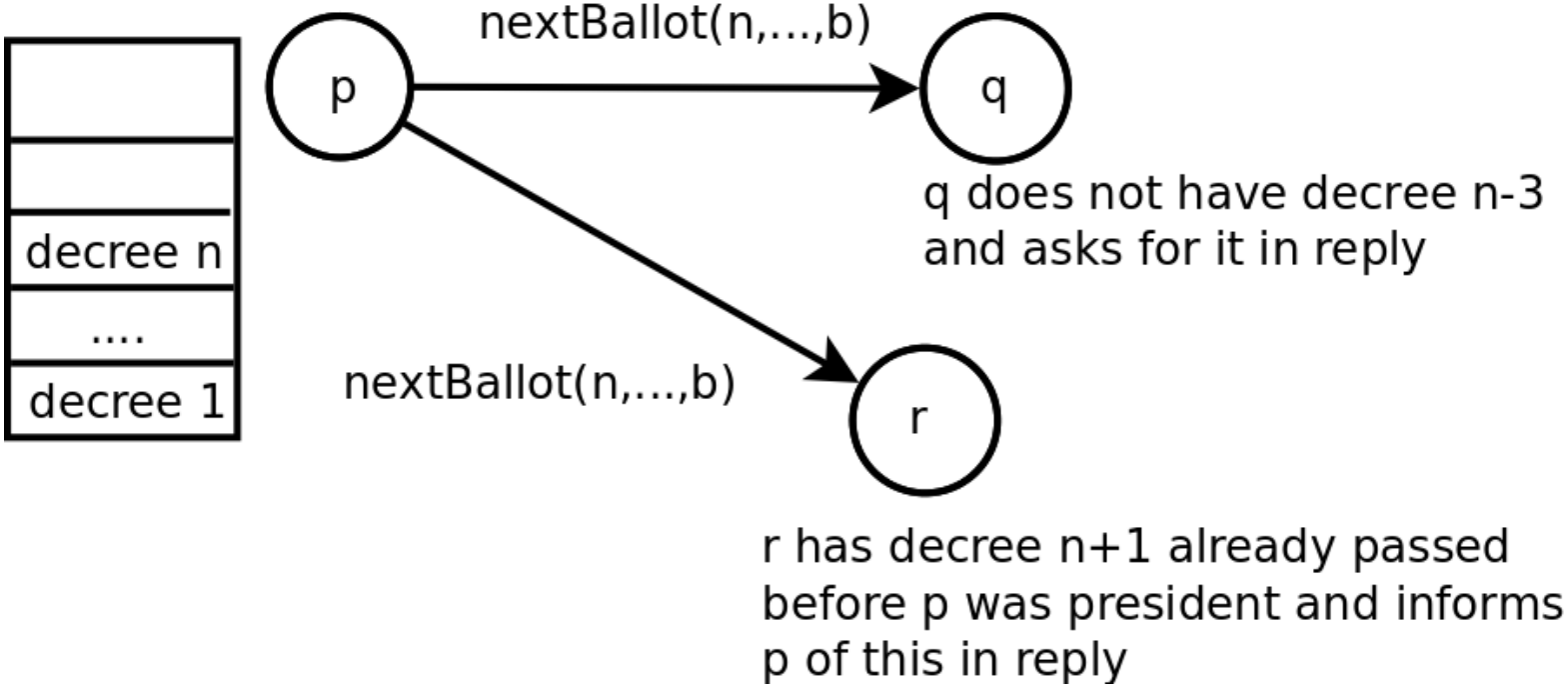
Single proposer

We should have only one priest initiating ballots. Let us call him president.

Single proposer

- Priests send their name to others regularly, and the one whose name is last in alphabetical order finally becomes president
- Anyone who wanted to pass a decree would inform the president

Multi-decree example



Conclusion

- Servers in a distributed system must agree in what order to make state transitions
- The Paxos protocol ensures that they never make inconsistent transitionsm, and as long as communication works, progress is made.
- The protocol is widely used in the industry.