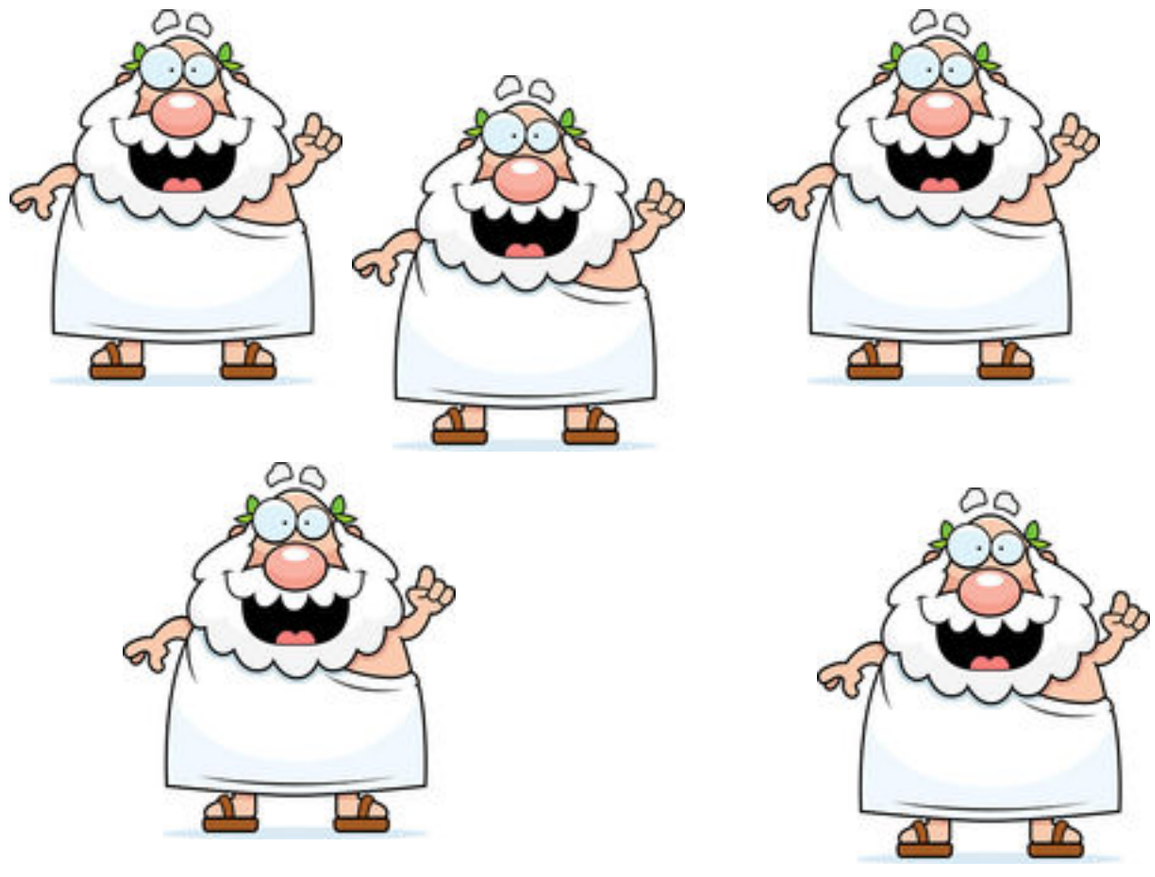
Paxos and Chubby



What and why

- Motivation for Paxos
- What exactly is Paxos
- How does it work?
- Chubby lock-service - the Google way…

The Island of Paxos

Part-Time Parliament

- Determine the *law* of the land
- the Law: *a sequence of decrees*
- the Law was determined in the *Chamber*
- *Priests* wandered in and out of the Chamber
- **Can the Law be consistent???**

Some assumptions

- Each priest had *their own ledger*
- An entry in the list of decrees was *never changed*
- Acoustics in the chamber were very poor (*messengers* were used to deliver messages)
- When in the Chamber, they all devoted themselves to the business of the parliament

The Single-Decree Synod

- Chosen through a series of *ballots*
- A ballot, *B*, was a referendum on *a single decree*
- Priest life was simple (*voting*, or *not voting*)
- *Quorum* - was a set of priests (for the ballot)
- Ballot successful!! - if and only if every priest in the quorum voted for the decree

This is paxos…

B1(**B**) - Each ballot in **B** has a unique ballot number

B2(**B**) - The quorums of any two ballots in **B** have at least one priest in common

B3(**B**) - For every ballot *B* in **B**, if any priest in *B*'s quorum voted in an earlier ballot in **B**, then the decree of *B* equals the decree of the latest of those earlier ballots

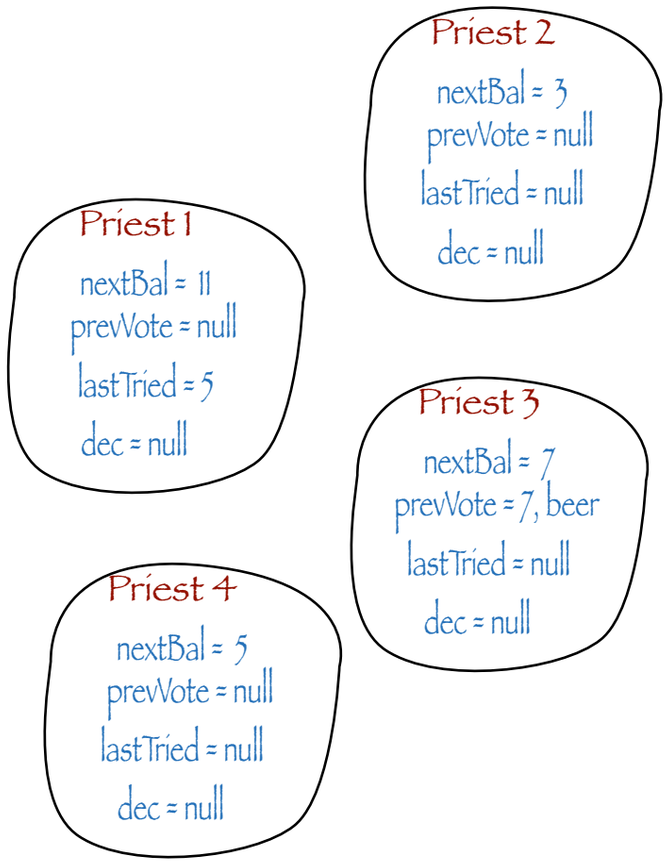
* **B** - the set of all ballots

#	decree	quorum and voters			
2	α	A	B	Γ	**Δ**
5	β	A	B	**Γ**	E
14	α	**B**		Δ	**E**
27	β	**A**		**Γ**	**Δ**
29	β		**B**	Γ	Δ

some theorems

1. If B1 - B3 hold, then any two successful ballots are for the same decree
2. If there are enough priests in the Chamber, then it is possible to conduct a successful ballot while preserving B1 - B3. (does not guarantee progress though…)

- 3 states
- 5 possible messages



Messages:

NextBallot(b)
LastVote(b, v)
BeginBallot(b, d)
Voted(b, q)
Success(d)

1. Priest *p* choose a new ballot number *b*, sends a *NextBallot(b)* message to some priests

2. Upon receipt of a *NextBallot(b)* message from *p* with *b>nextBal[q]*, priest *q* sets *nextBal[q]* to *b* and sends a *LastVote(b, v)* message to *p,* where *v* equals *prevVote[q]*. (Ignored if *b≤nextBal[q].)* ***Promise to ignore ballots smaller than b***

3. After receiving a *LastVote(b, v)* message from every priest in some majority set *Q*, where *b=lastTried[p]*, priest *p* initiates a new ballot with number *b*, quorum *Q*, and decree *d,* where *d* is chosen to satisfy B3. He then sends a *BeginBallot(b, d)* message to every priest in *Q.*

4. Upon receipt of a *BeginBallot(b, d)* message with *b=nextBal[q],* priest *q* casts his vote in ballot number *b,* sets *prevVote[q]* to this vote, and sends a *Voted(b, q)* message to *p*. (Ignored if *b =/ nextBal[q]*)

5. If *p* has received a *Voted(b, q)* message from every priest *q* in *Q* (the quorum for ballot number *b*), where *b=lastTried[p]*, then he writes *d* (the decree of the ballot) in his ledger and sends a *Success(d)* message to every priest.

6. Upon receiving a *Success(d)* message, a priest enters decree *d* in his ledger.

(1) Choose a new ballot number
b = 12

Priest 1
nextBal = 11
prevVote = null
lastTried = 5
dec = null

Priest 2
nextBal = 3
prevVote = null
lastTried = null
dec = null

Priest 3
nextBal = 7
prevVote = 7, beer
lastTried = null
dec = null

Priest 4
nextBal = 5
prevVote = null
lastTried = null
dec = null

Priest 5
nextBal = 4
prevVote = null
lastTried = null
dec = null

Priest 6
On Vacation

(1) Choose a new ballot number
b = 12

Priest 1
nextBal = 12
prevVote = null
lastTried = 5
dec = null

Priest 2
nextBal = 3
prevVote = null
lastTried = null
dec = null

Priest 3
nextBal = 7
prevVote = 7, beer
lastTried = null
dec = null

Priest 4
nextBal = 5
prevVote = null
lastTried = null
dec = null

Priest 5
nextBal = 4
prevVote = null
lastTried = null
dec = null

Priest 6
On Vacation

(1) Set
lastTried[p]
to *b (=12)*

Priest 1
nextBal = 12
prevVote = null
lastTried = 5
dec = null

Priest 2
nextBal = 3
prevVote = null
lastTried = null
dec = null

Priest 3
nextBal = 7
prevVote = 7, beer
lastTried = null
dec = null

Priest 4
nextBal = 5
prevVote = null
lastTried = null
dec = null

Priest 5
nextBal = 4
prevVote = null
lastTried = null
dec = null

Priest 6
On Vacation

(1) Set
lastTried[p]
to *b (=12)*

Priest 1
nextBal = 12
prevVote = null
lastTried = 12
dec = null

Priest 2
nextBal = 3
prevVote = null
lastTried = null
dec = null

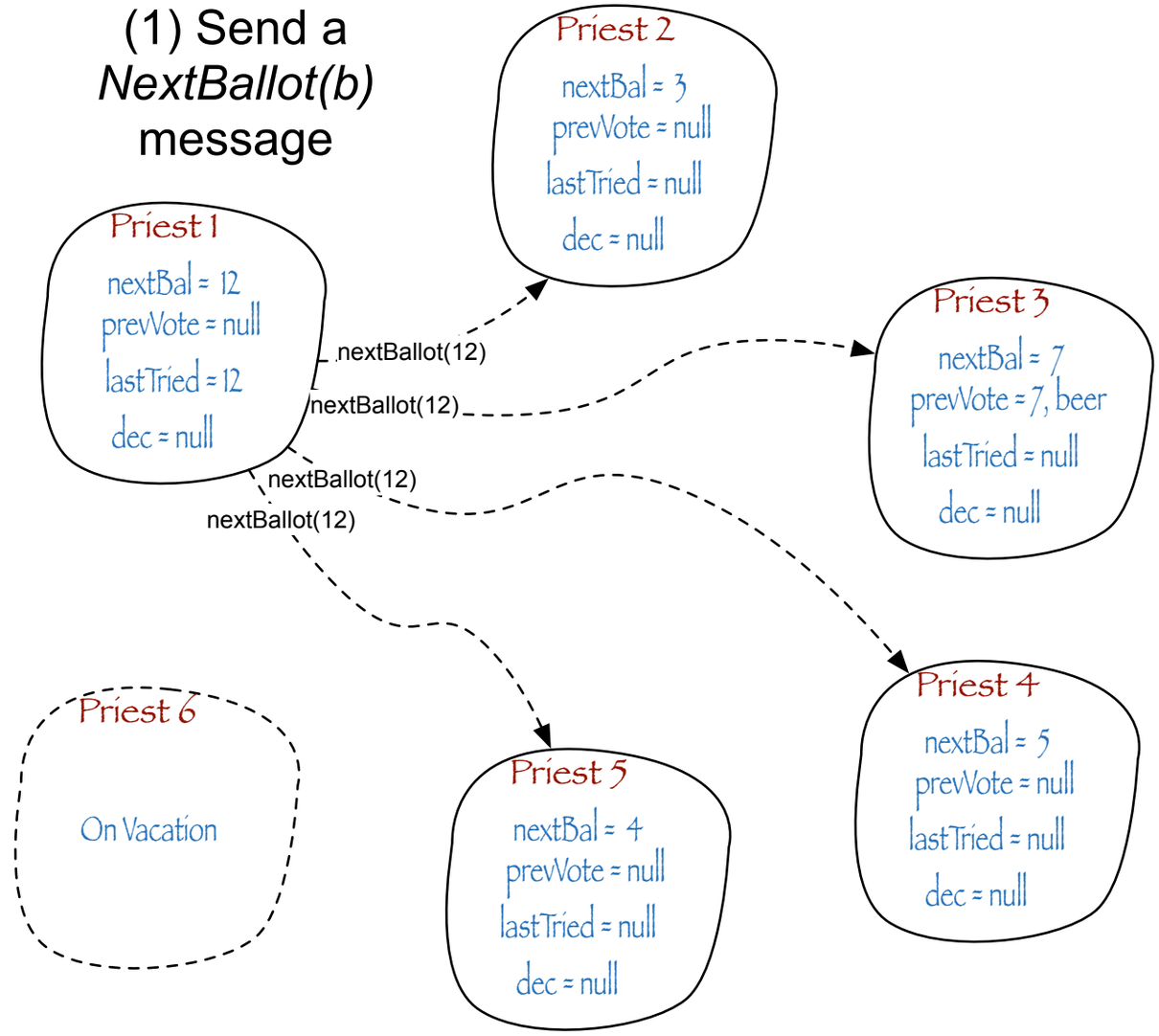
Priest 3
nextBal = 7
prevVote = 7, beer
lastTried = null
dec = null

Priest 4
nextBal = 5
prevVote = null
lastTried = null
dec = null

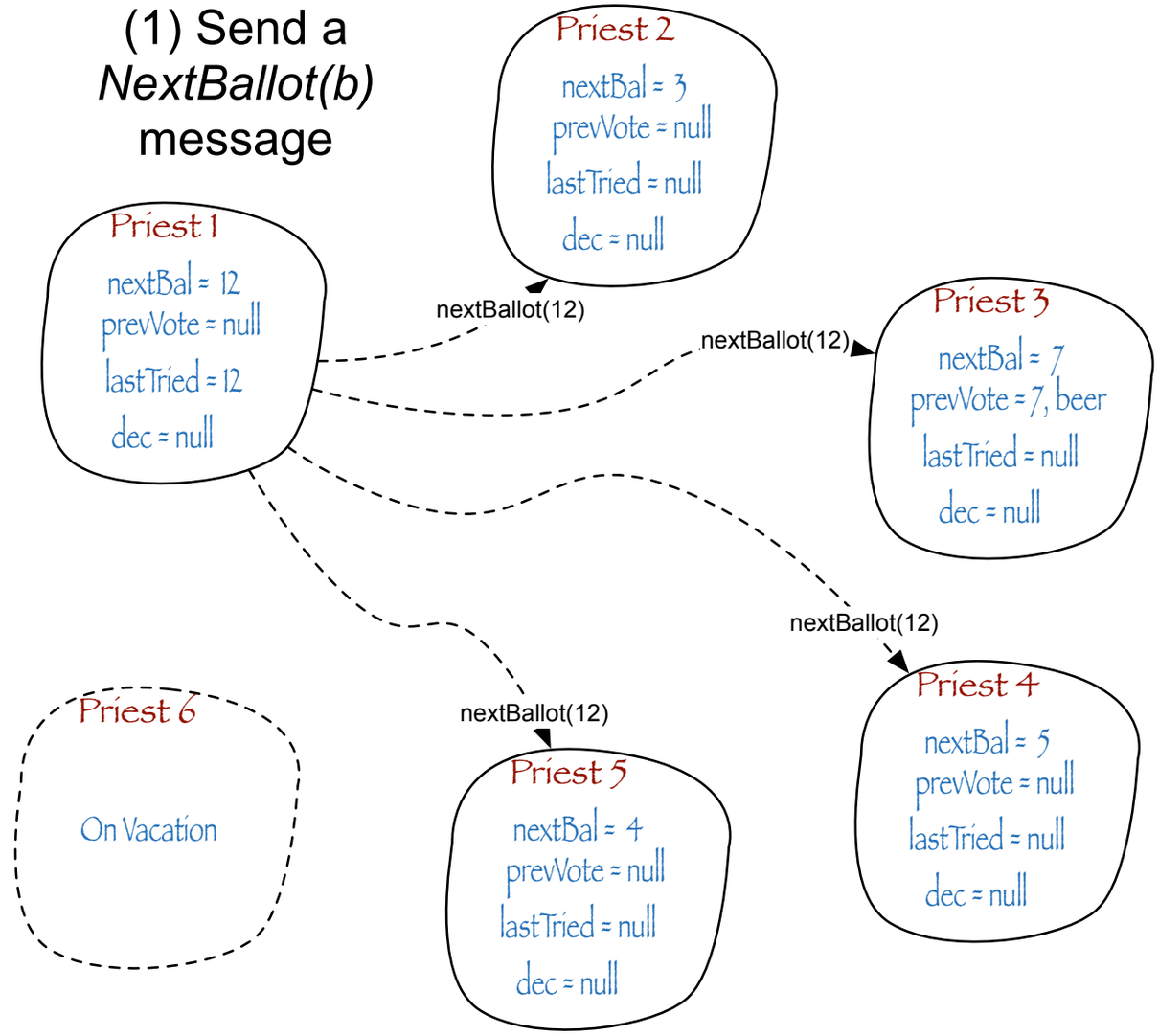
Priest 5
nextBal = 4
prevVote = null
lastTried = null
dec = null

Priest 6
On Vacation

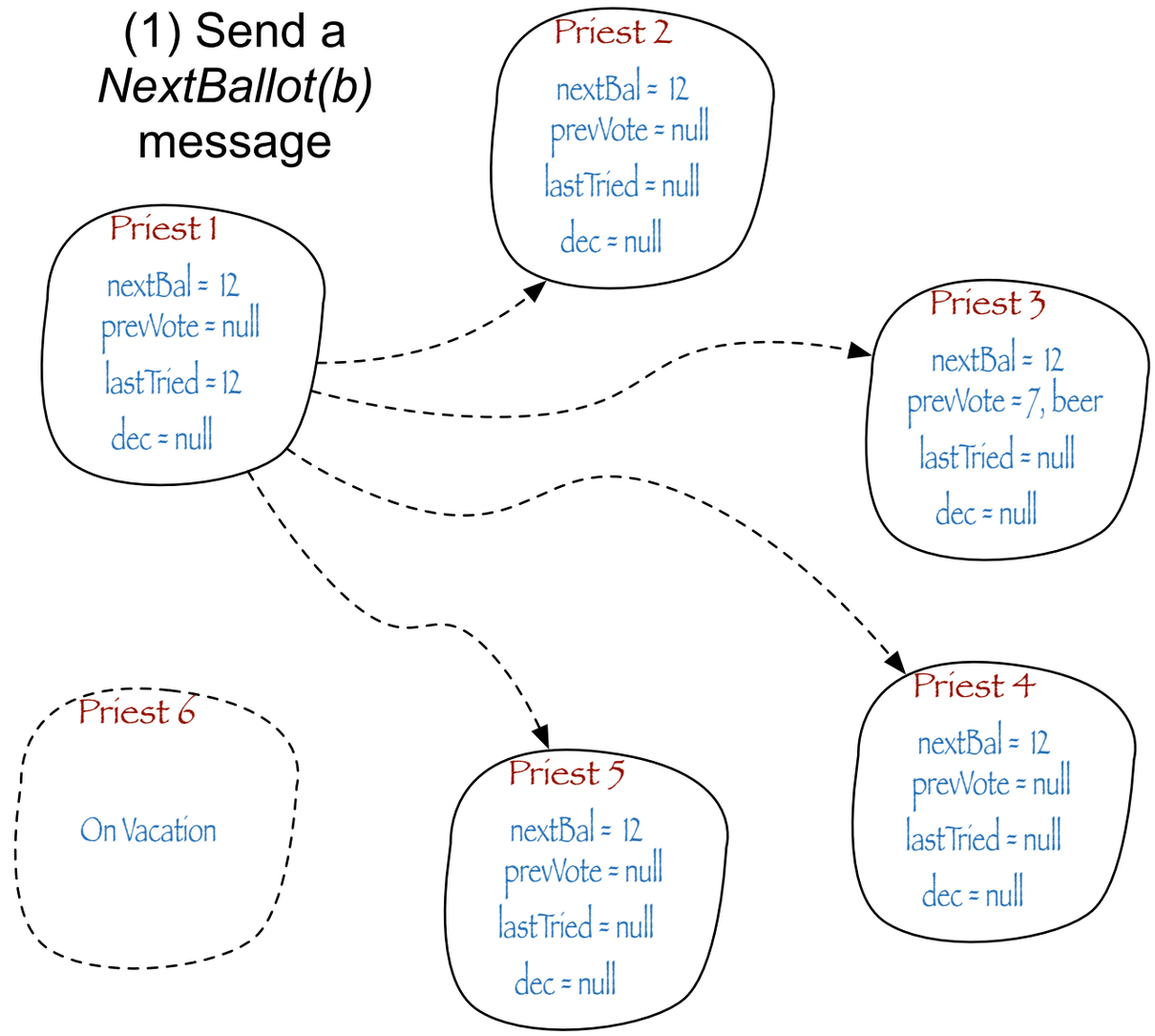
(1) Send a *NextBallot(b)* message



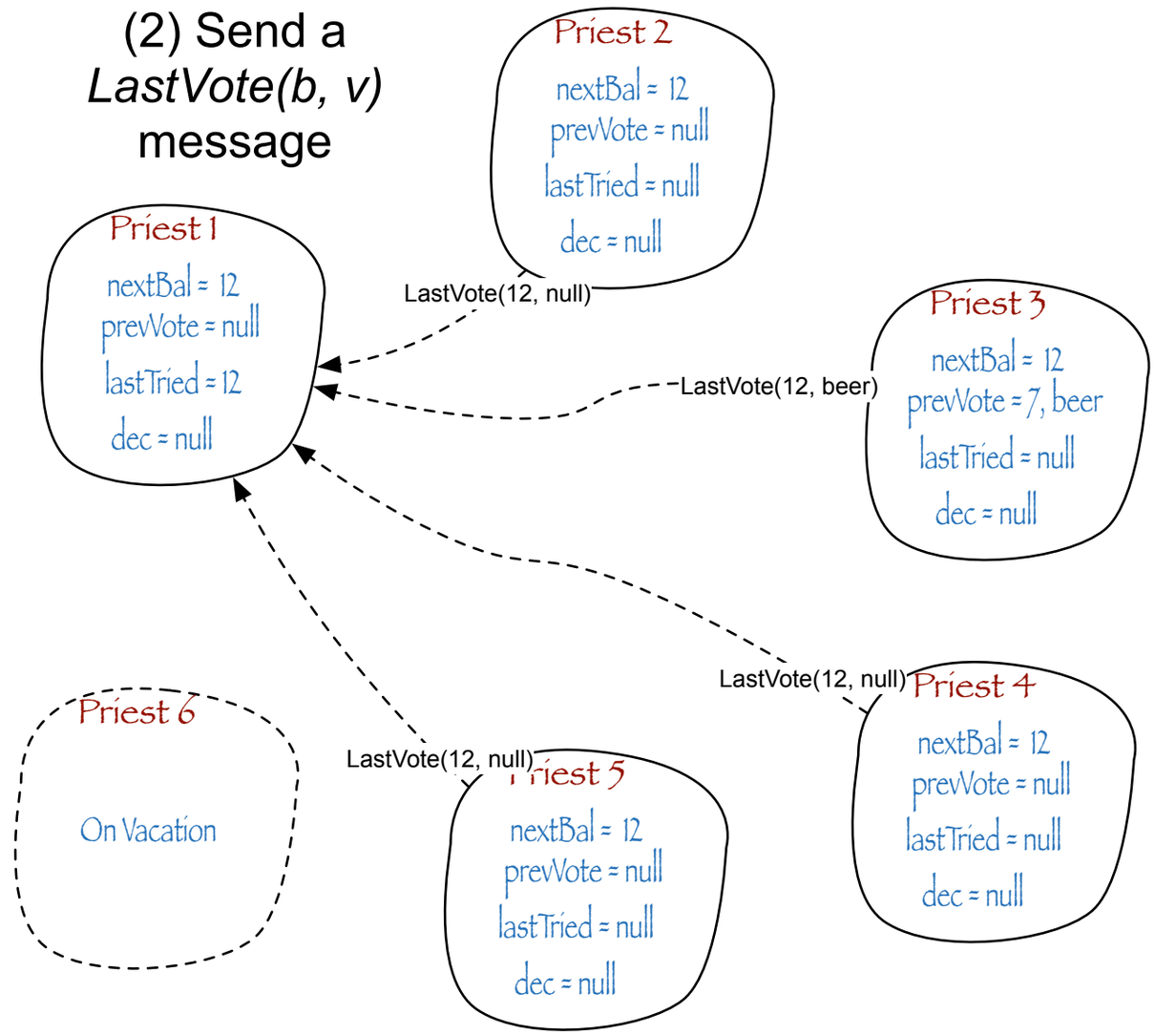
(1) Send a *NextBallot(b)* message



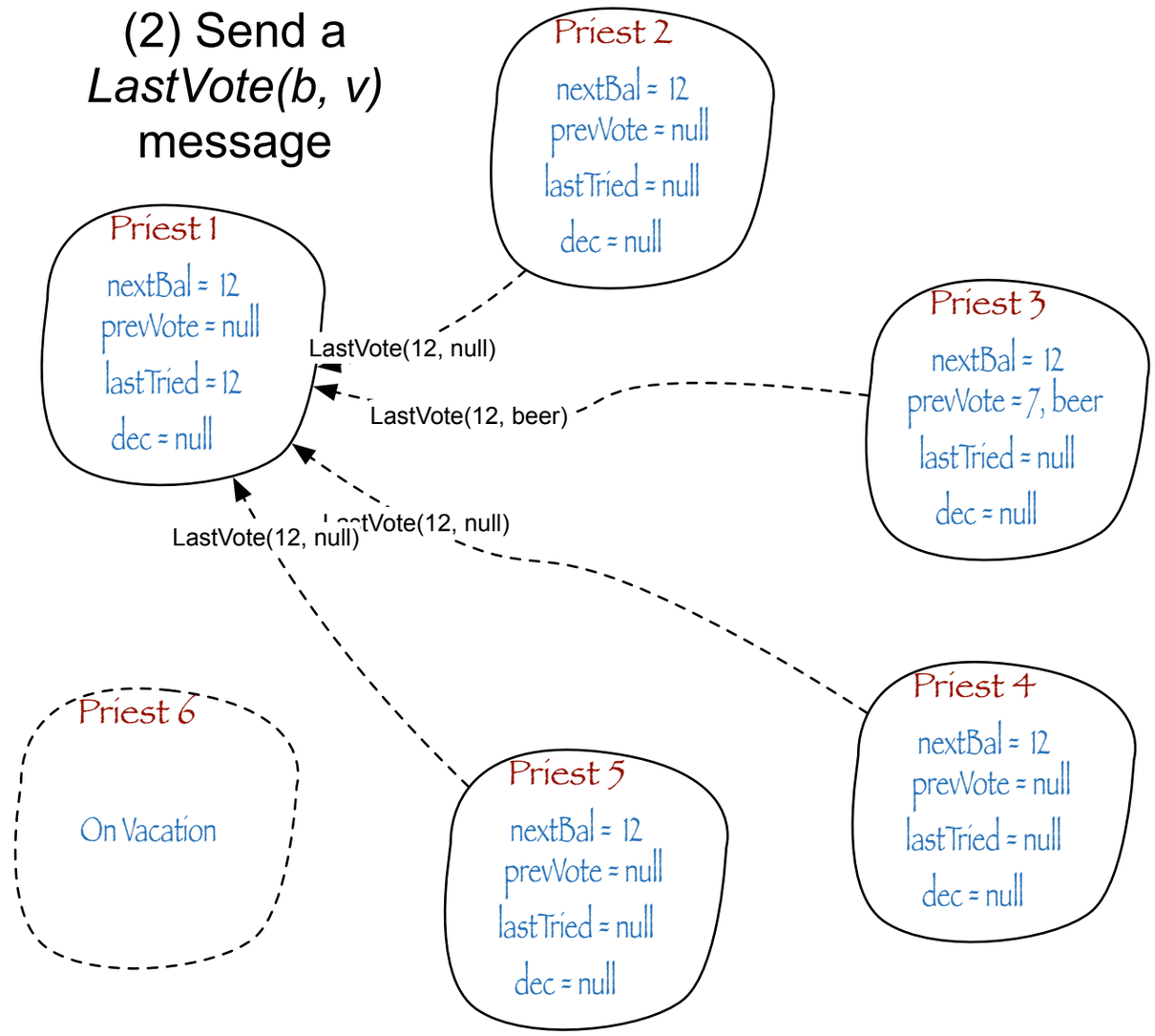
(1) Send a *NextBallot(b)* message



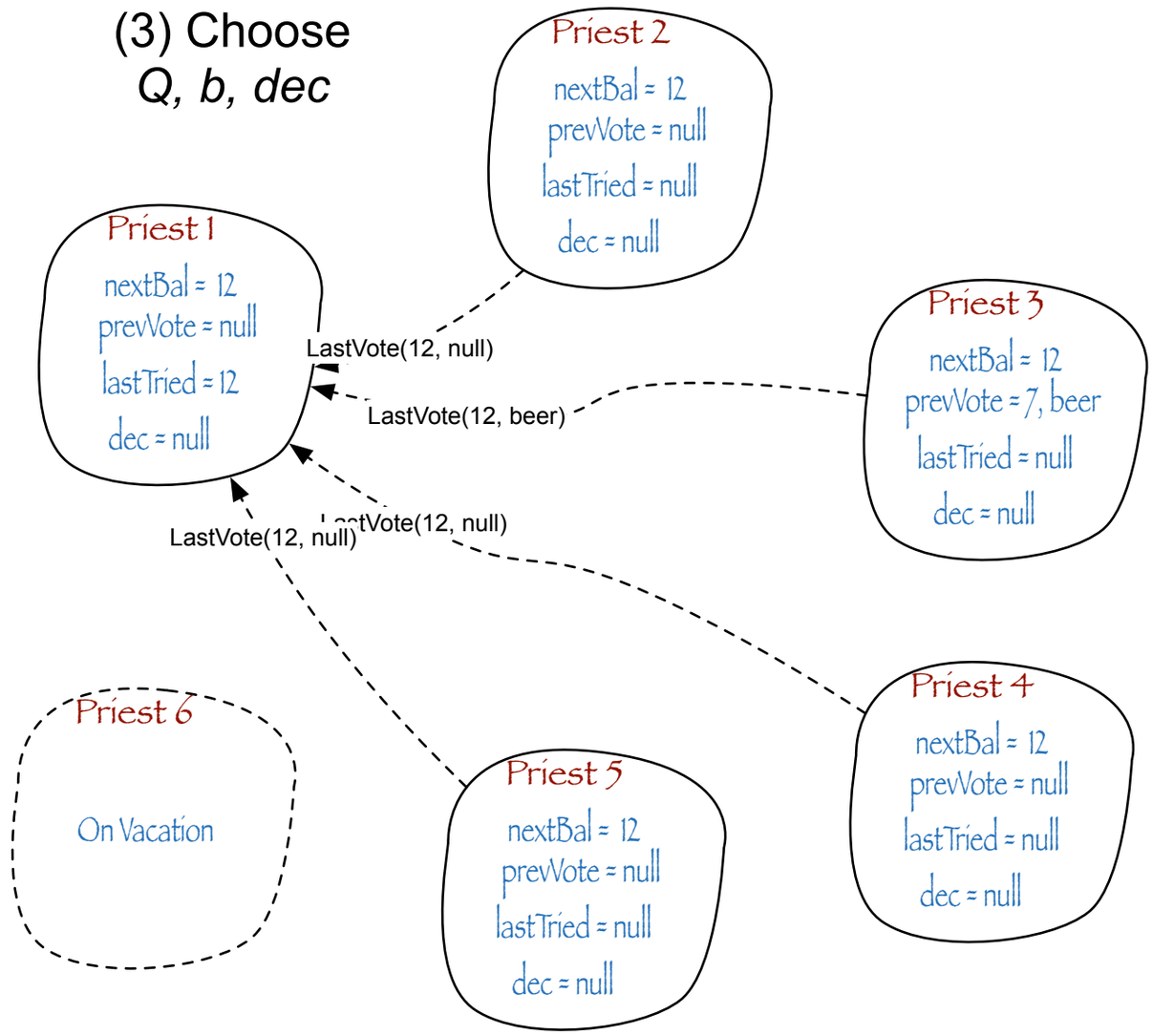
(2) Send a *LastVote(b, v)* message



(2) Send a *LastVote(b, v)* message

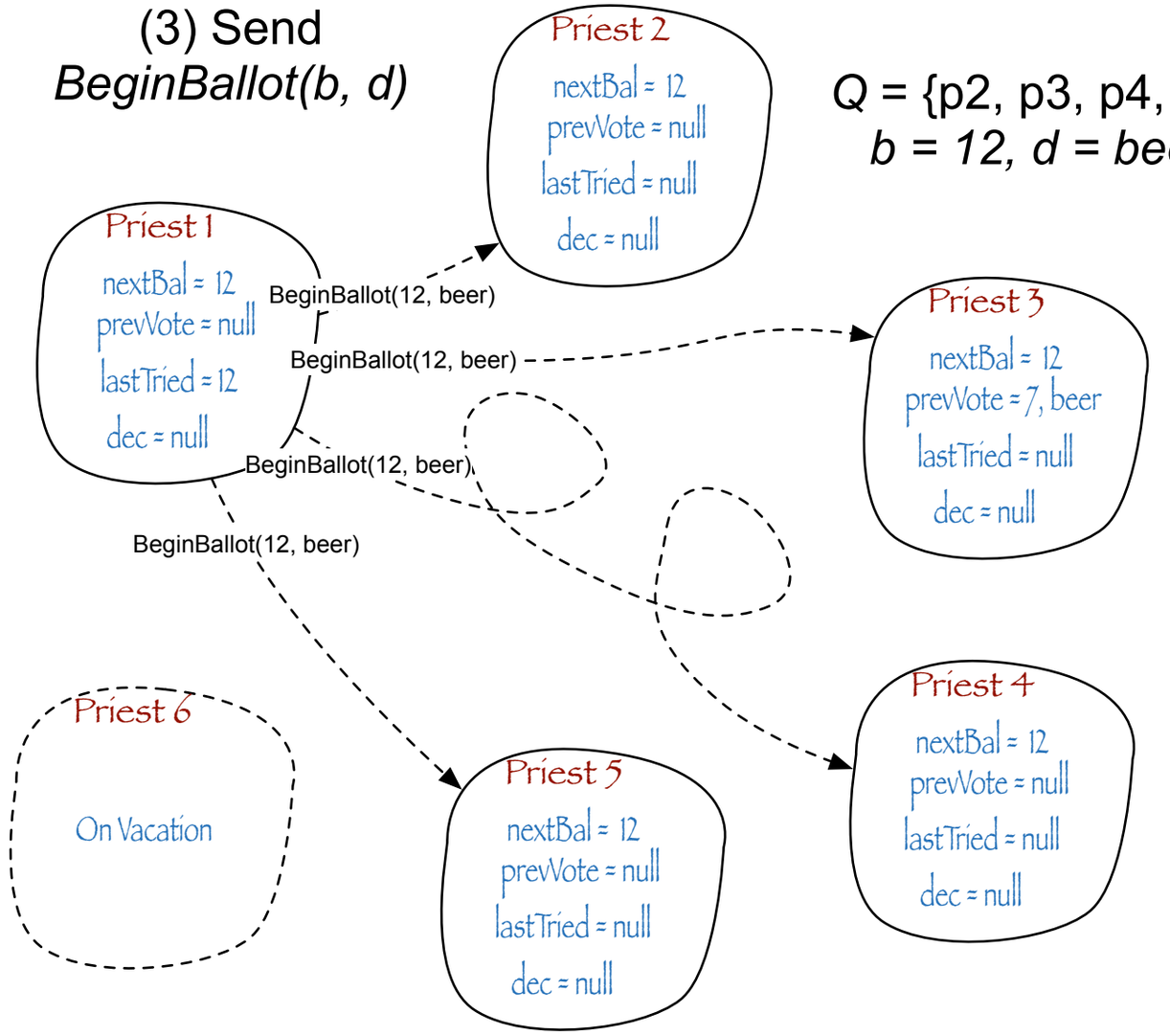


(3) Choose
Q, b, dec

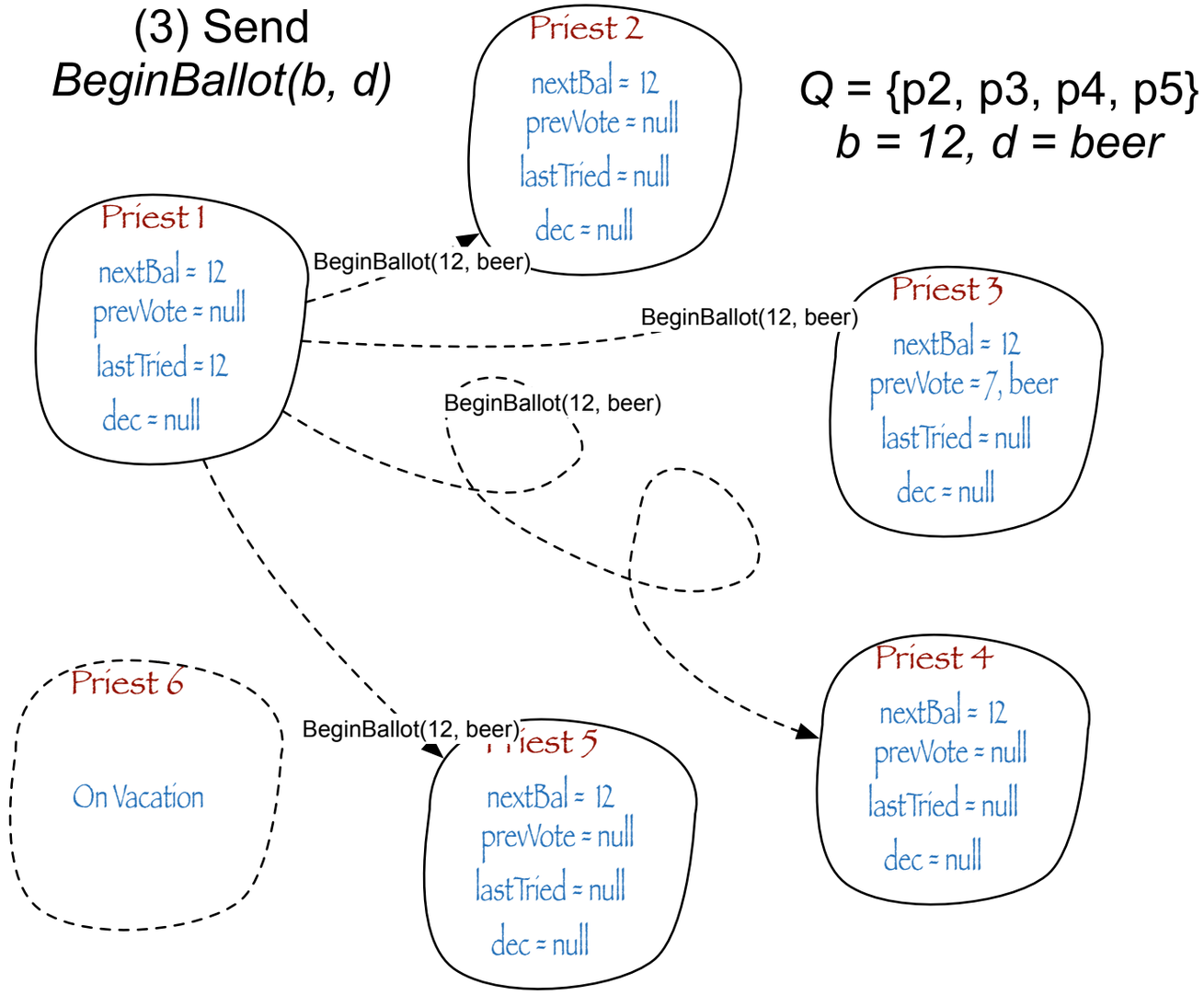


(3) Send
BeginBallot(b, d)

Q = {p2, p3, p4, p5}
b = 12, d = beer



(3) Send
BeginBallot(b, d)



Q = {p2, p3, p4, p5}
b = 12, d = beer

(4) Set
prevVote[p] to *d*

Priest 2
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

Q = {p2, p3, p4, p5}
b = 12, d = beer

Priest 1
nextBal = 12
prevVote = null
lastTried = 12
dec = null

BeginBallot(12, beer)

Priest 3
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

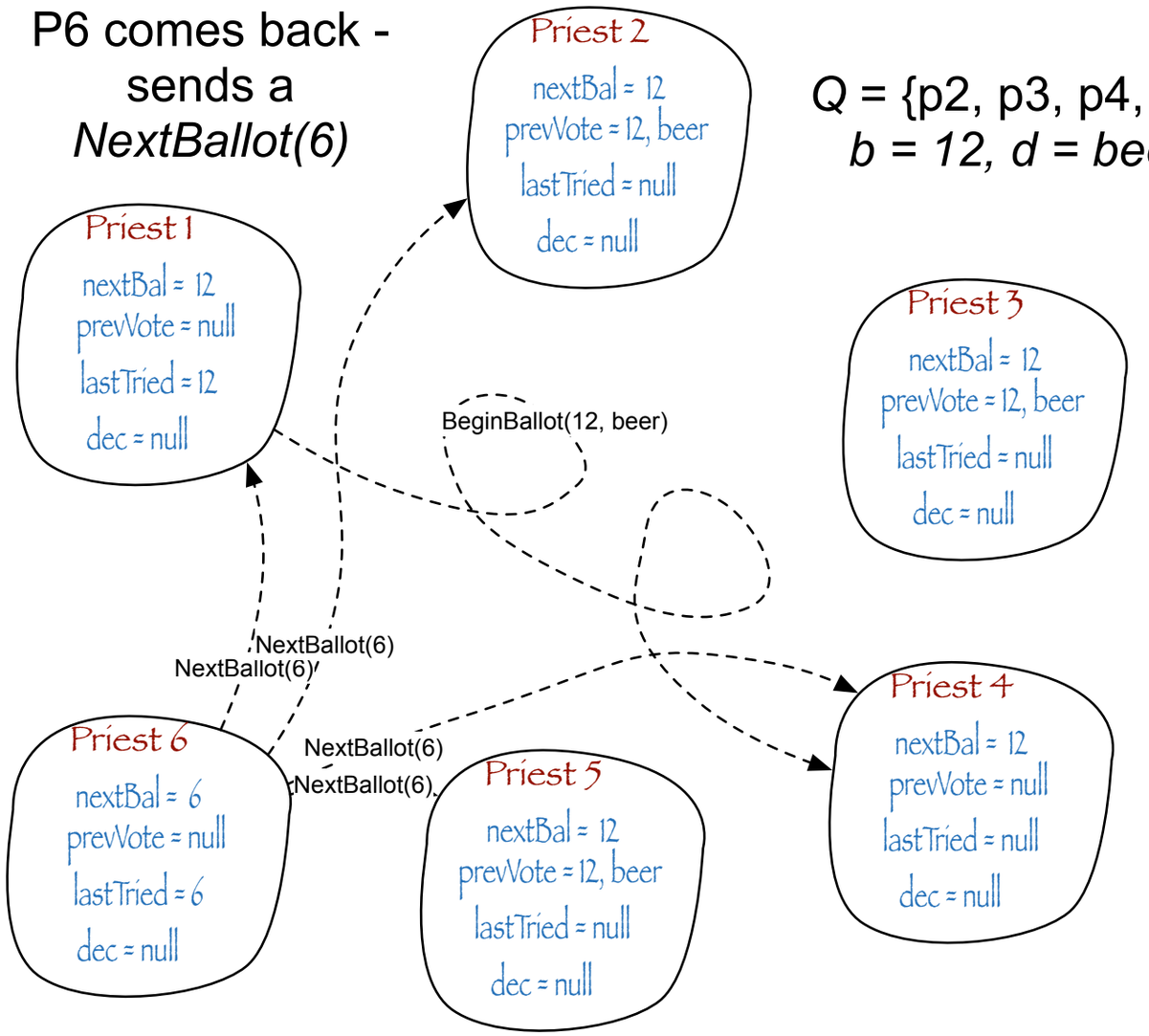
Priest 6
On Vacation

Priest 5
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

Priest 4
nextBal = 12
prevVote = null
lastTried = null
dec = null

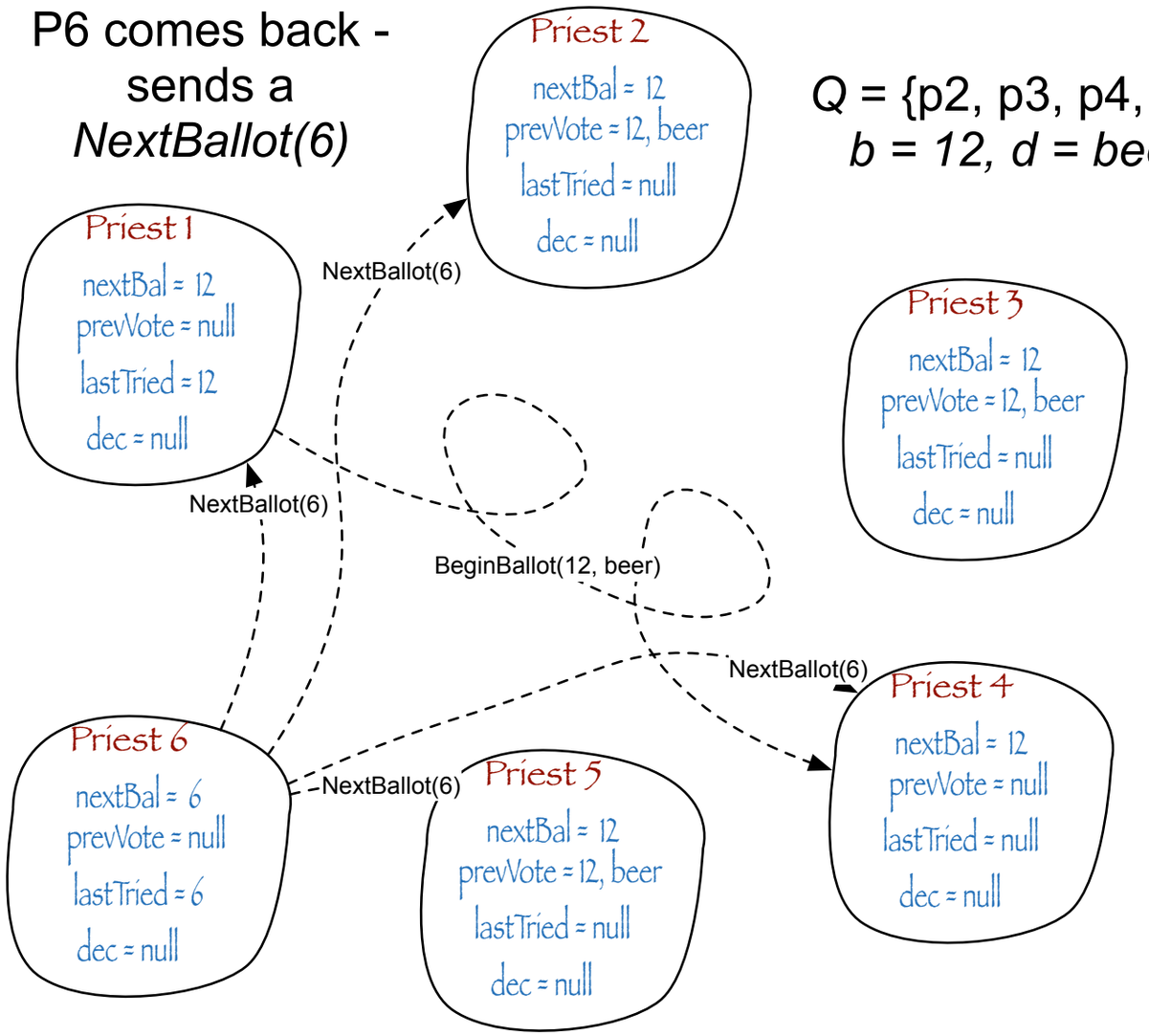
P6 comes back - sends a *NextBallot(6)*

Q = {p2, p3, p4, p5}
b = 12, d = beer



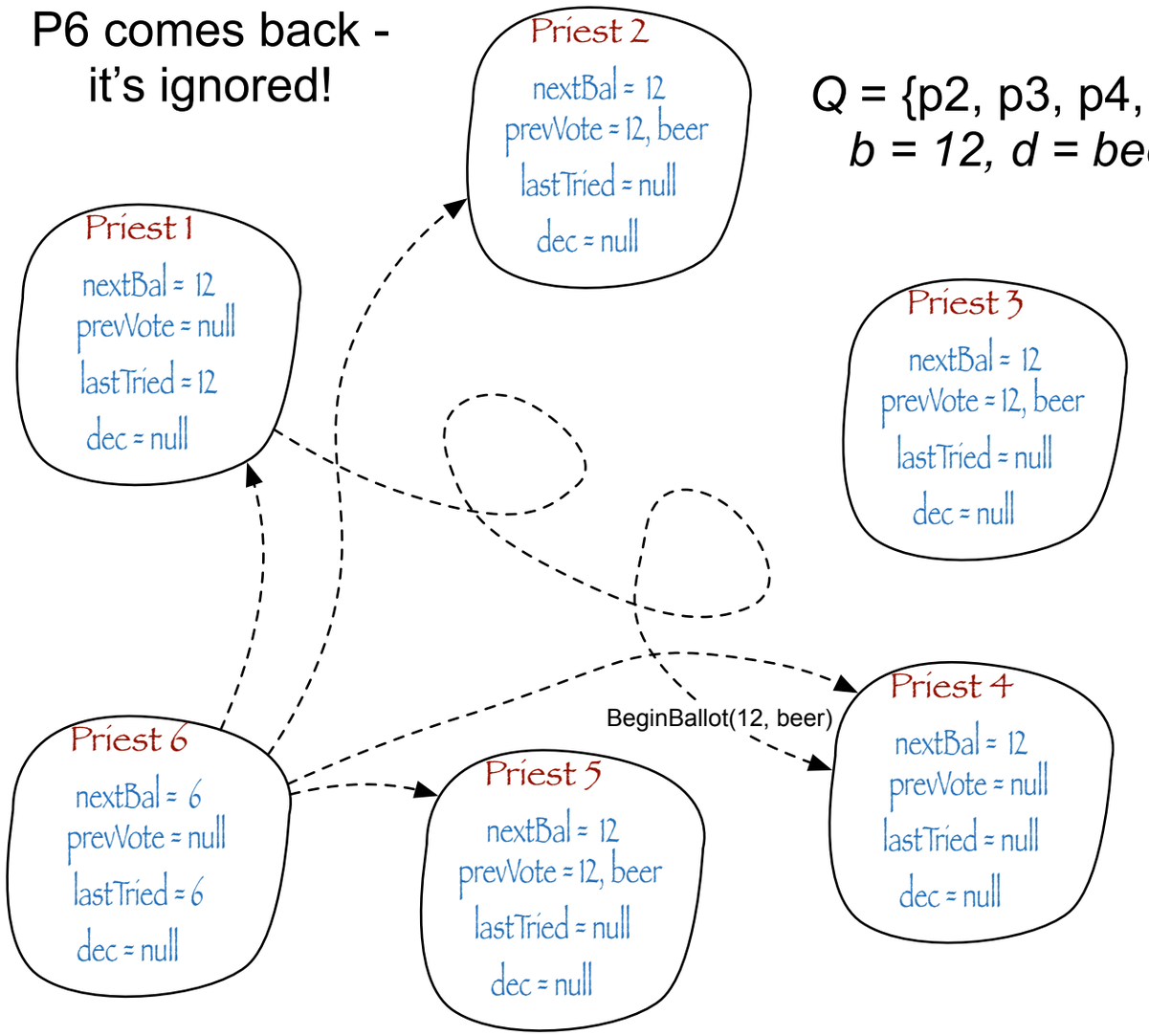
P6 comes back - sends a *NextBallot(6)*

Q = {p2, p3, p4, p5}
b = 12, d = beer



P6 comes back - it's ignored!

Q = {p2, p3, p4, p5}
b = 12, d = beer



P6 comes back - it's ignored!

Priest 2
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

Q = {p2, p3, p4, p5}
b = 12, d = beer

Priest 1
nextBal = 12
prevVote = null
lastTried = 12
dec = null

Priest 3
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

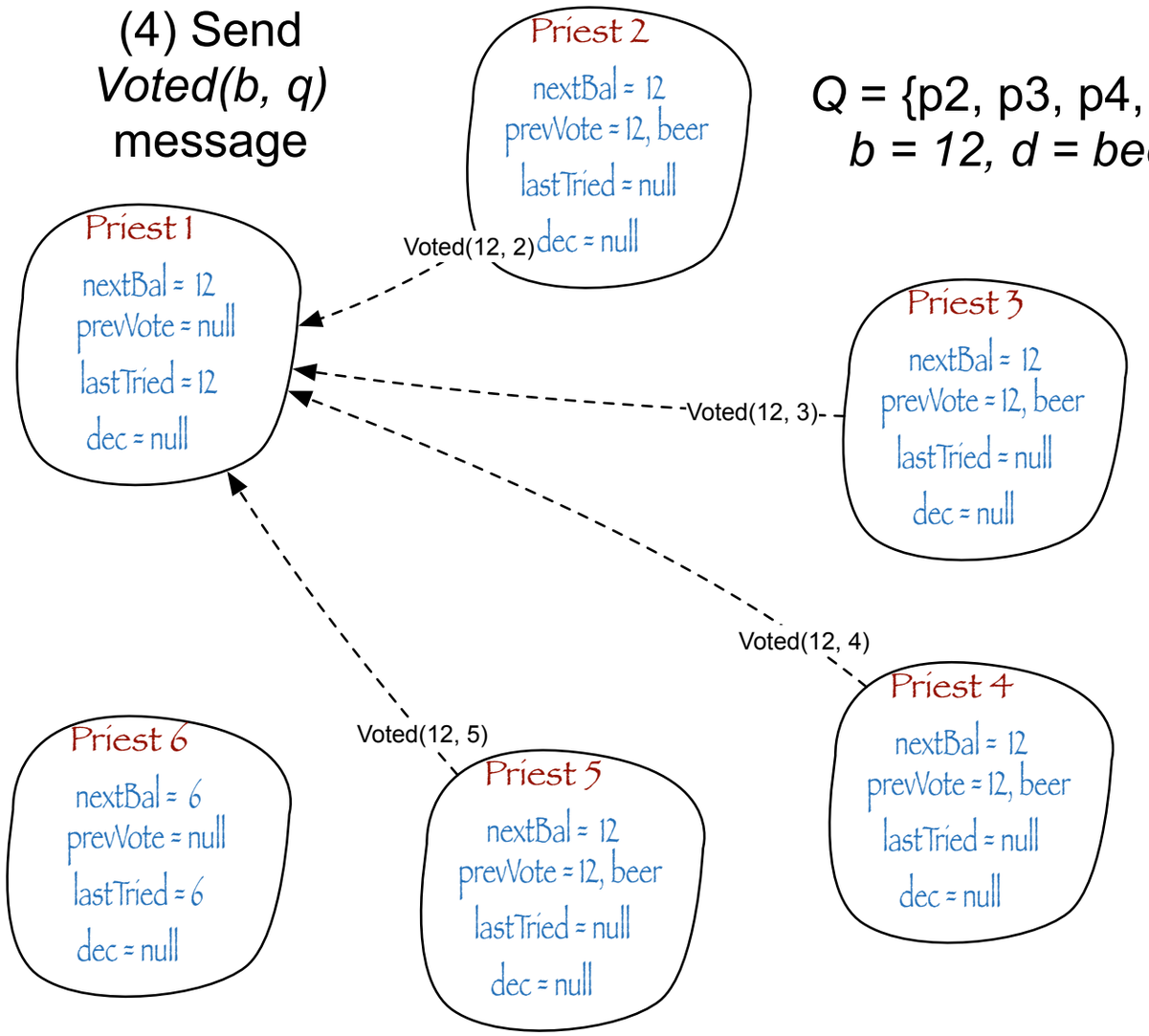
Priest 6
nextBal = 6
prevVote = null
lastTried = 6
dec = null

Priest 5
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

Priest 4
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

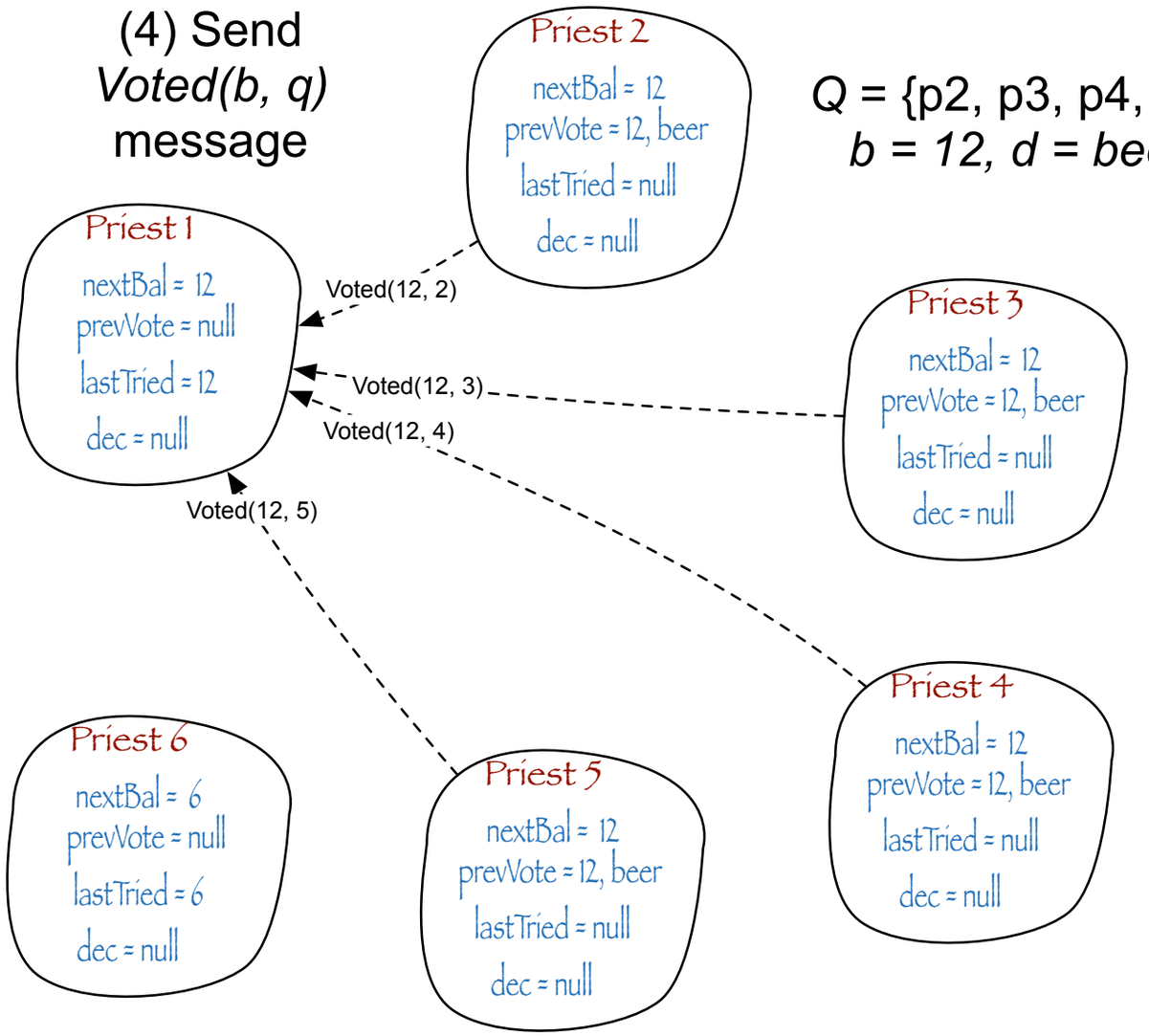
(4) Send
Voted(b, q)
message

Q = {p2, p3, p4, p5}
b = 12, d = beer



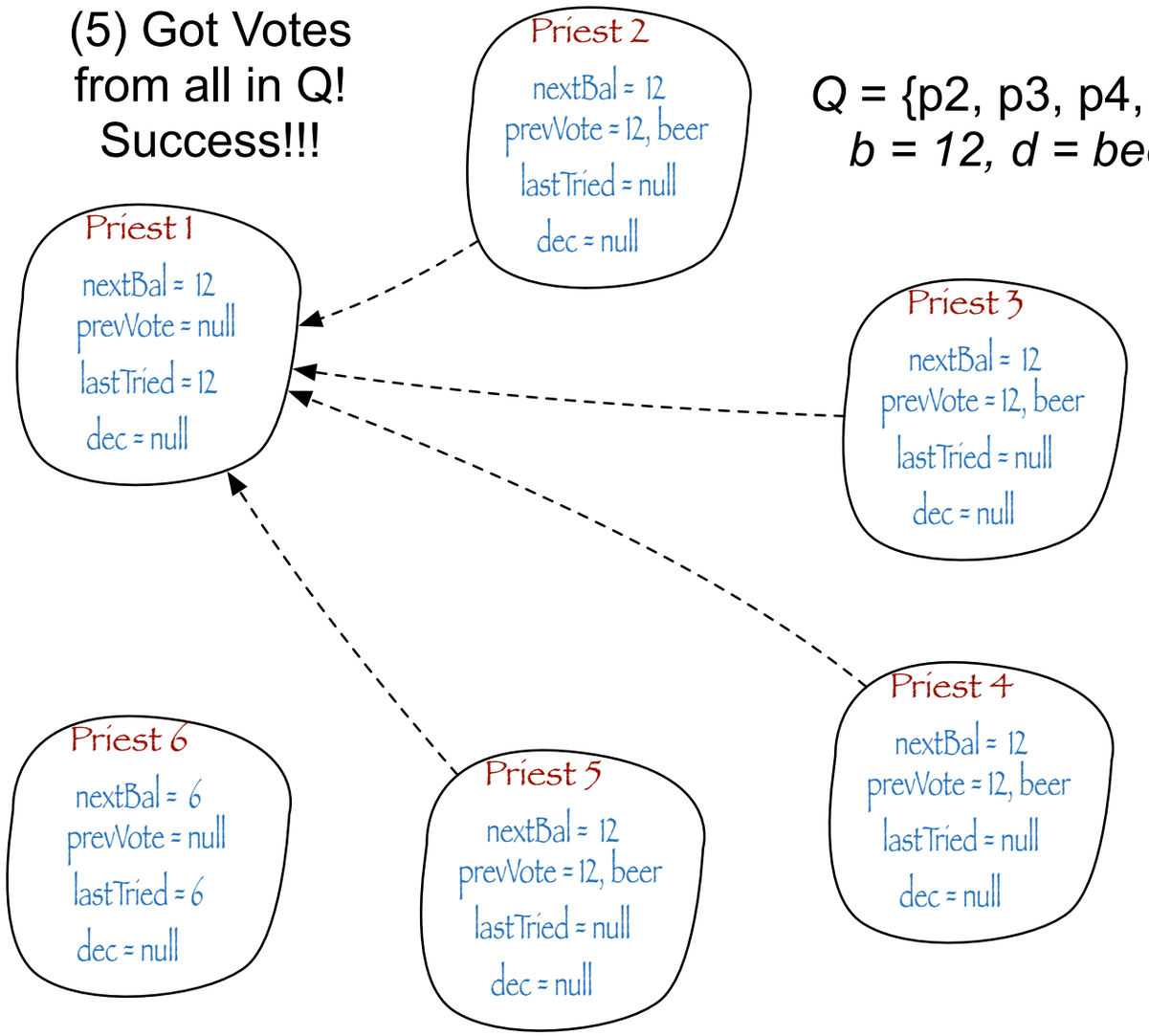
(4) Send
Voted(b, q)
message

Q = {p2, p3, p4, p5}
b = 12, d = beer



(5) Got Votes from all in Q! Success!!!

Q = {p2, p3, p4, p5}
b = 12, d = beer



(5) Enter *d* in the ledger

Priest 2
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

Q = {p2, p3, p4, p5}
b = 12, d = beer

Priest 1
nextBal = 12
prevVote = null
lastTried = 12
dec = beer

Priest 3
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

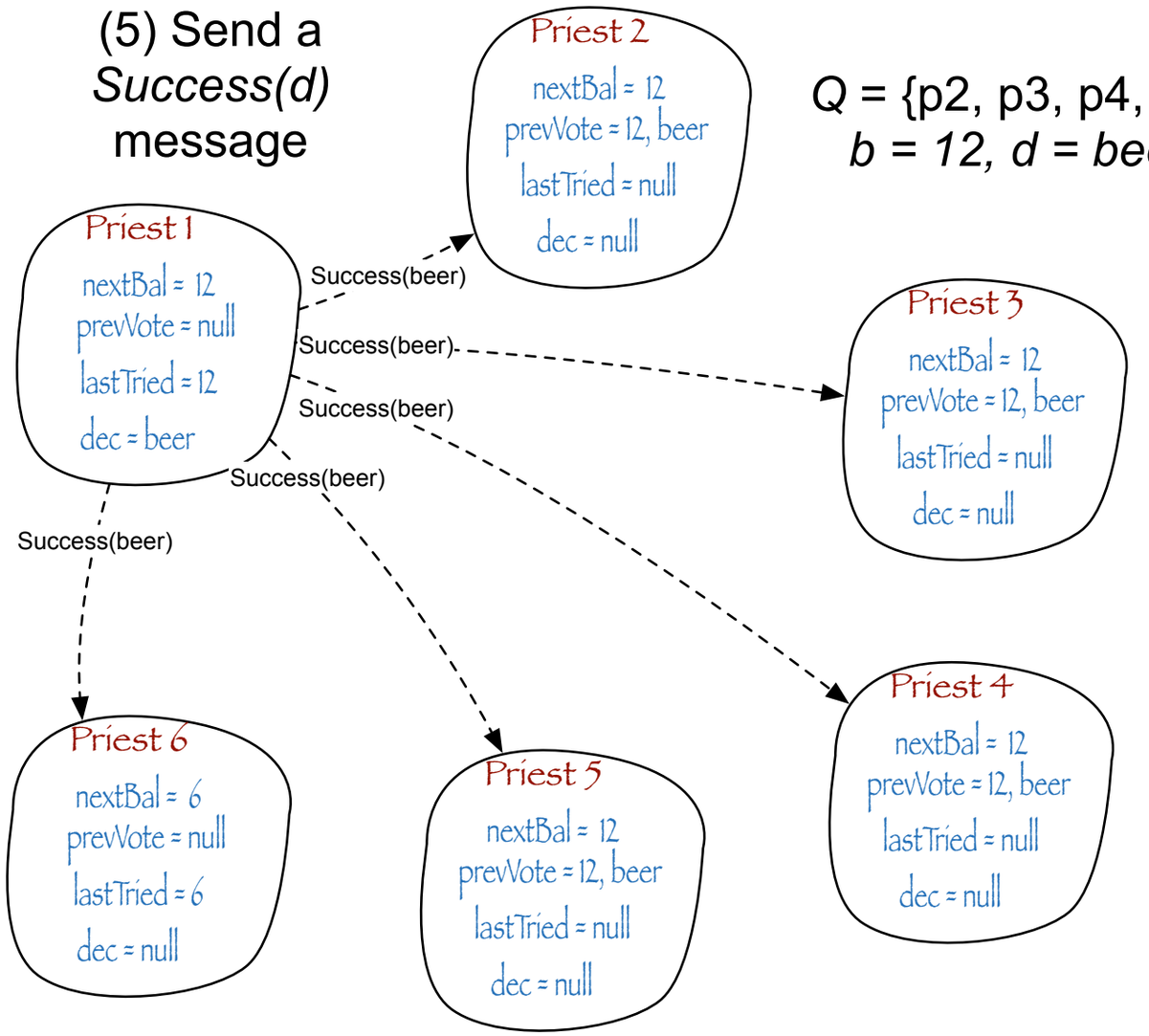
Priest 6
nextBal = 6
prevVote = null
lastTried = 6
dec = null

Priest 5
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

Priest 4
nextBal = 12
prevVote = 12, beer
lastTried = null
dec = null

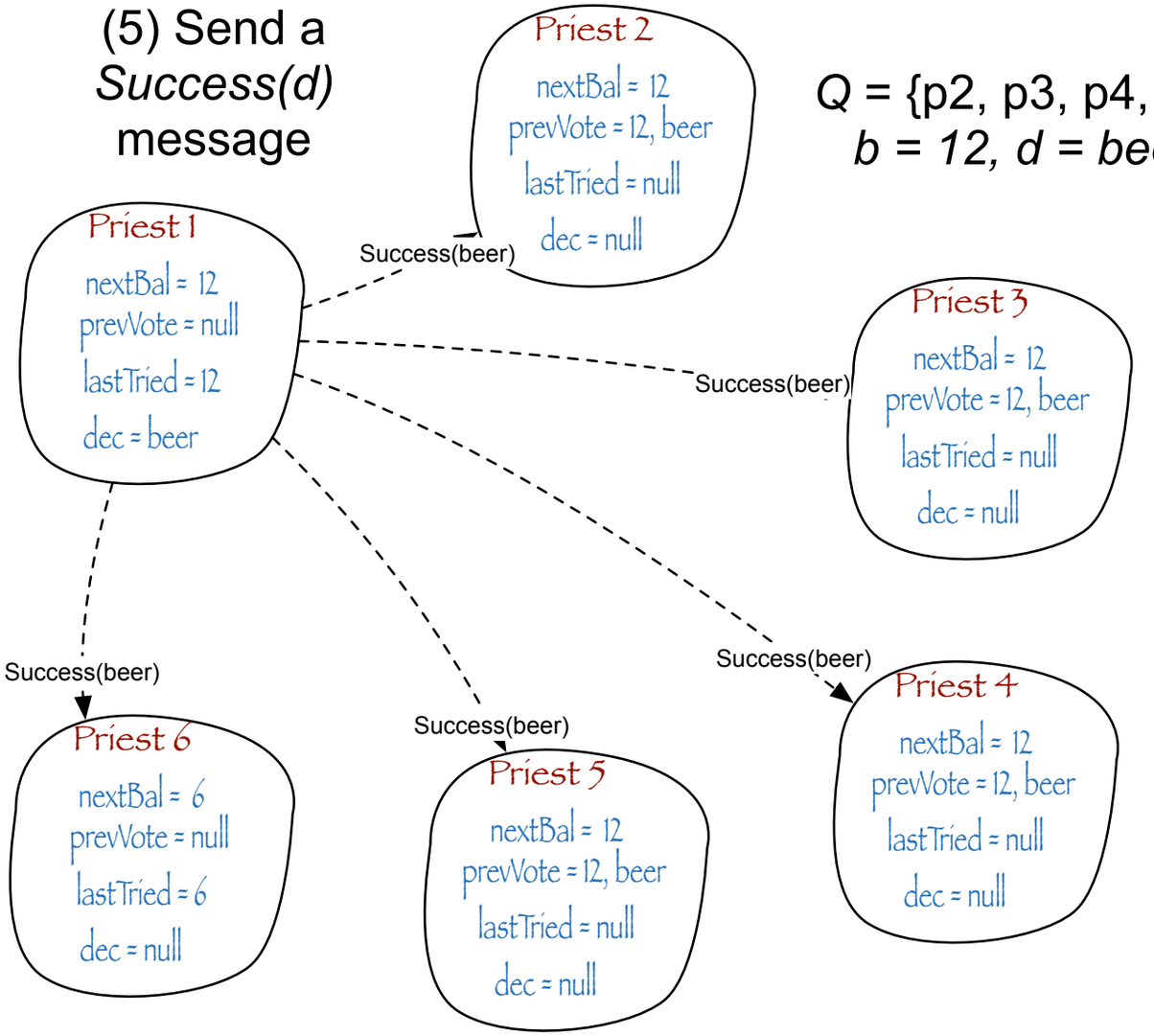
(5) Send a *Success(d)* message

Q = {p2, p3, p4, p5}
b = 12, d = beer



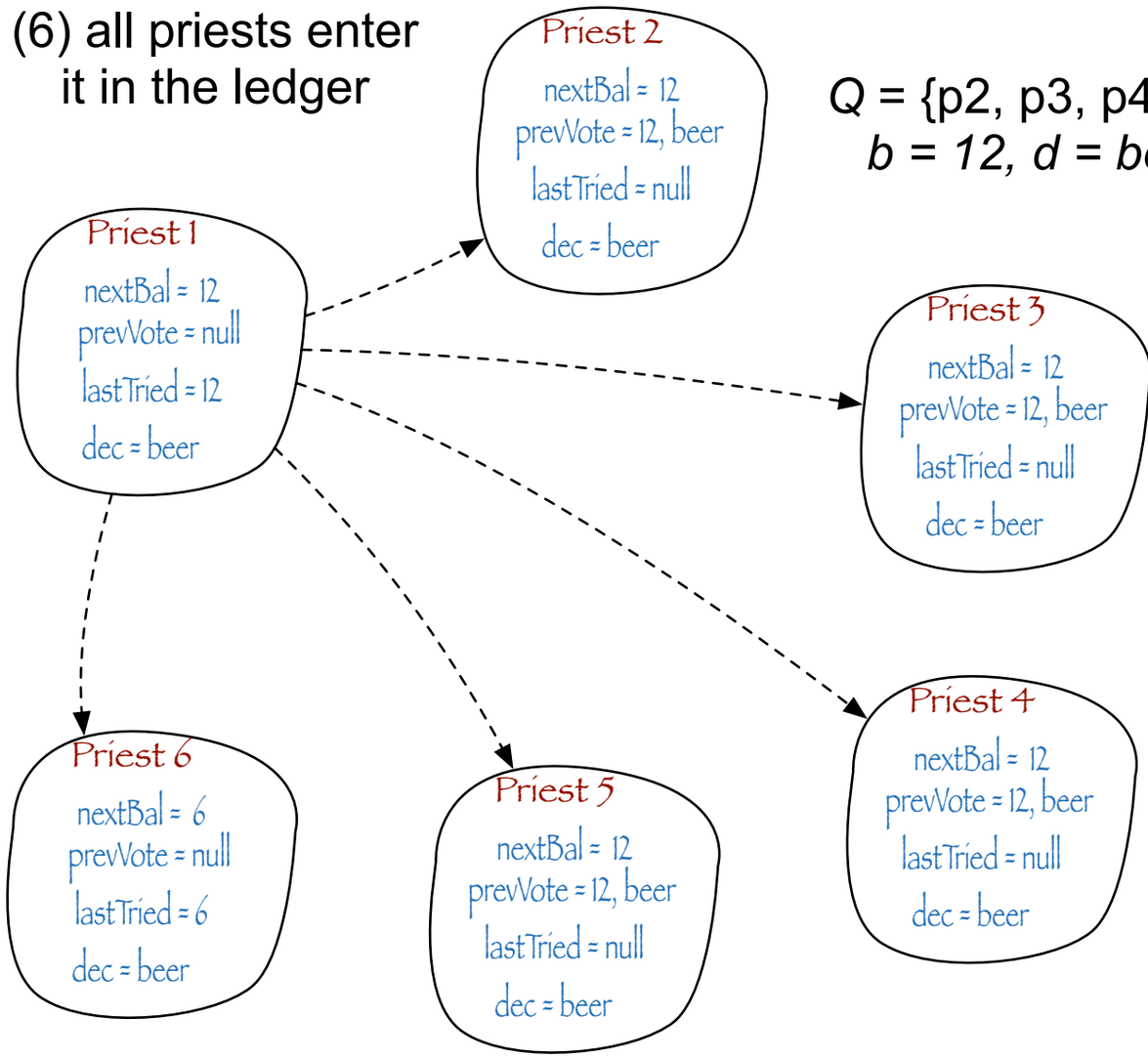
(5) Send a *Success(d)* message

Q = {p2, p3, p4, p5}
b = 12, d = beer



(6) all priests enter it in the ledger

Q = {p2, p3, p4, p5}
b = 12, d = beer



Some more reading

- The Part-time Parliament, 1998 - Lamport
- Paxos made simple, 2001 - Lamppost
- Paxos made live, 2007 - Chandra et al (Google)

Chubby

Mike Burrows, *Google Inc.*

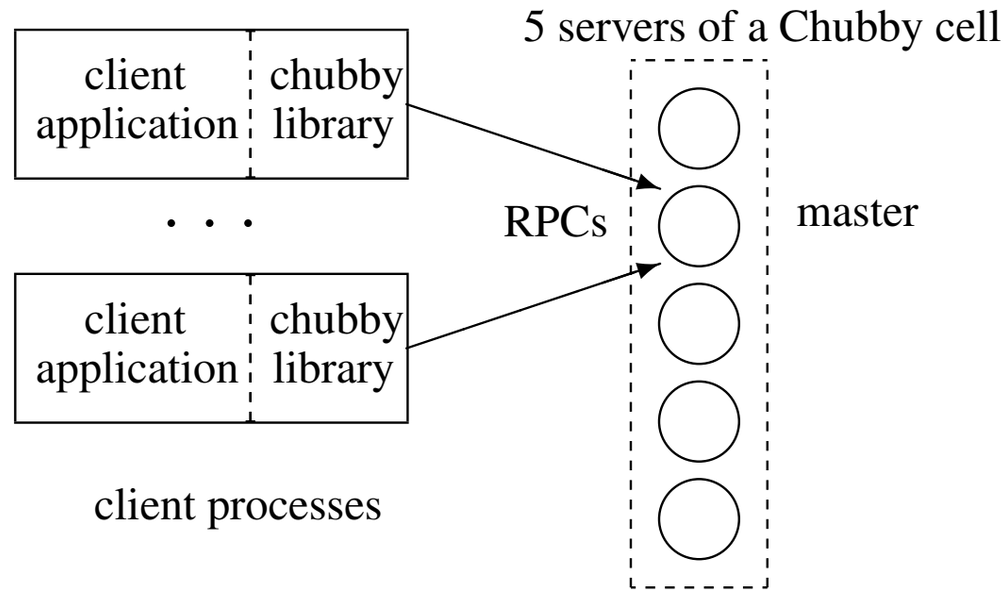
Not saying that this is Google…
Nor Mike Burrows…

What is Chubby?

- A lock service
- Storage for loosely-coupled distributed systems
- “The purpose of the lock service is to allow its clients to synchronise their activities and to agree on basic information about their environment”
- Google File System (GFS) and Bigtable use Chubby as the root of their dist. structure

System Structure

- Chubby cell run Paxos
- Client directs ALL requests to the master
- Master is elected for some time - *master lease*



Files and namespace

- `ls/foo/wombat/pouch`
- ls - stands for lock service
- foo - the name of the chubby cell
- only files and directories (nodes)
- Any node can act as an advisory reader/writer lock
- Meta Data @ node - e.g. access control lists (ACLs)

API

- open() - opens a handle to a node
- close() - guess what this does…
- acquire(), release() - takes and releases a lock

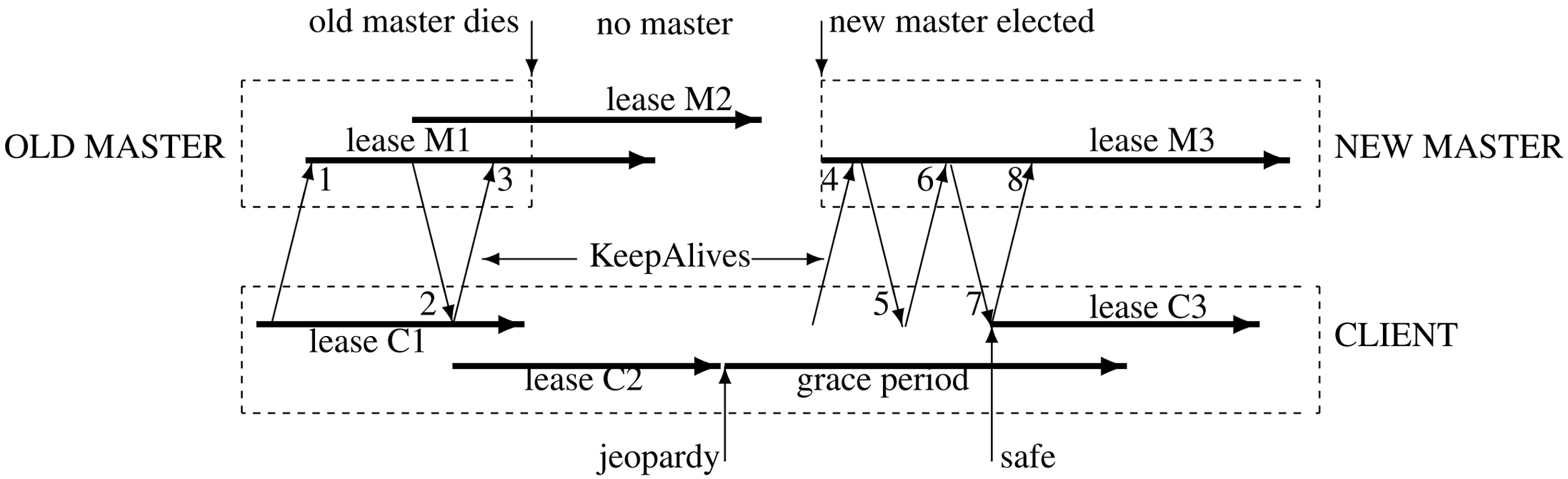
Locks

- reader-writer advisory locks (like mutexes)
- acquiring either lock requires write permissions
- Lock-delay - you kinda keep the lock after you die

Sessions and KeepAlives

- a relationship between Chubby and the Client
- handshakes keep it alive
- each session has a lease
- Why? - expensive to connect to a new master

Sessions and KeepAlives



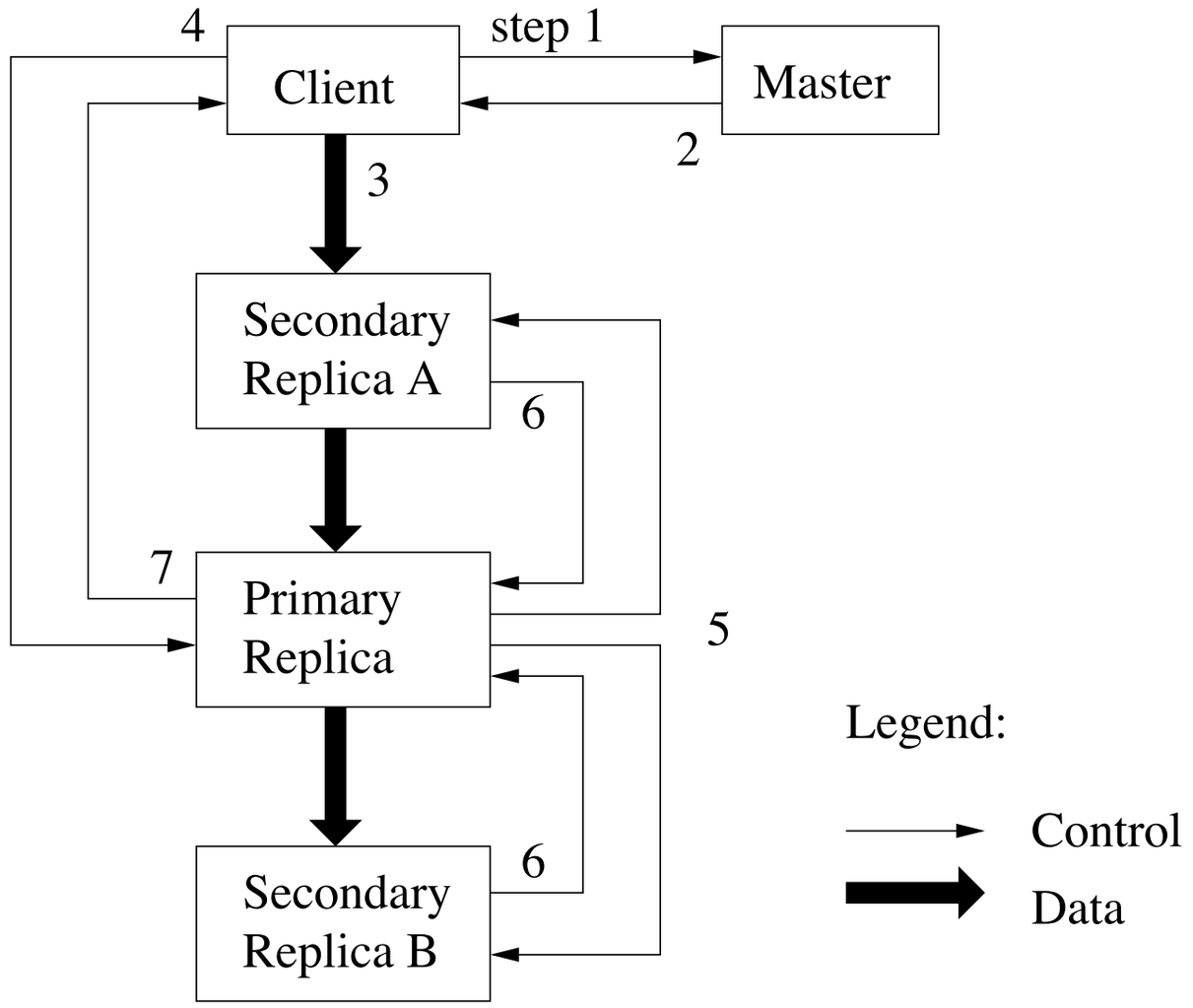
Design Rationale

- lock service, instead of a paxos library

- small-files to permit the election of a primary

Intended use

- Whenever a bunch of clients need to elect a primary among themselves
- GFS is a great example



Actual use

- Name service
- **Google's primary internal name service!**
- GFS, MapReduce, Bigtable all use Chubby

More reading

- the Google file system, 2003 - Ghemawat et al
- Paxos made live, 2001- Chandra et al
- The Chubby lock service for loosely-coupled distributed systems, 2006 - Mike Burrows
- https://www.hakkalabs.co/articles/chubby-lock-service-loosely-coupled-distributed-systems