IF3130 – Sistem Paralel dan Terdistribusi

Paxos

Achmad Imam Kistijantoro (imam@itb.ac.id)

Consensus Problem

- setiap proses dapat mengajukan sebuah nilai, dan harus sepakat pada nilai akhir hasil konsensus
- banyak kasus pada sistem terdistribusi yang dapat dilihat sebagai masalah consensus:
  - total order broadcast, atomic commit, terminating reliable broadcast

Consensus property

C1. Validity: Any value decided is a value proposed

C2. Agreement: No two correct processes decide differently

C3. Termination: Every correct process eventually decides

C4. Integrity: No process decides twice

Paxos

- Protokol consensus yang diusulkan oleh Lamport (1989)
- published di JACM (1998)
- Asynchronous model: message dapat sampai dalam waktu lama, proses dapat berjalan sangat lambat
- Majority rules: asumsi majority process correct

Konsep Paxos

- Process dibagi menjadi
  - Proposer: mengusulkan value yang akan diputuskan (decided) system
  - Acceptor: memilih di antara nilai yang diusulkan
    - Jika sebagian besar acceptor memilih nilai yang sama => decided
  - Learner: mempelajari hasil decision

- Safety
  - only values proposed may be decided
  - only a single value is decided
  - only decided values may be learnt by correct learners
- Liveness
  - some proposed value is eventually decided
  - if a value has been decided, all correct learners can eventually learn it

FLP Impossibility

- No asynchronous consensus algorithm can guarantee both safety and liveness

- However

- asynchrony is overly pessimistic;
- partial synchrony is the common case

- *safety affects algorithm correctness;*
- *liveness affects algorithm performance;*
- compromise on liveness;
- only when the system becomes asynchronous

Paxos

- Paxos merupakan algoritma untuk asynchronous consensus

- Menjamin:
  - Safety: always
  - Liveness: saat sistem menjamin
    - Partially synchronous for long enough period of time
    - No more than N/2 crashes

System Model

- Processor
  - Fixed set of processes
  - Asynchronous computation
  - Failures are independent and random
  - Processes may rejoin the protocol after failure
  - No byzantine behavior

System Model

- Network:
- asynchronous communication (unbounded delivery time);
- all processors can communicate with one another;
- messages can be lost, reordered, or duplicated;
- messages, if delivered, are not corrupted

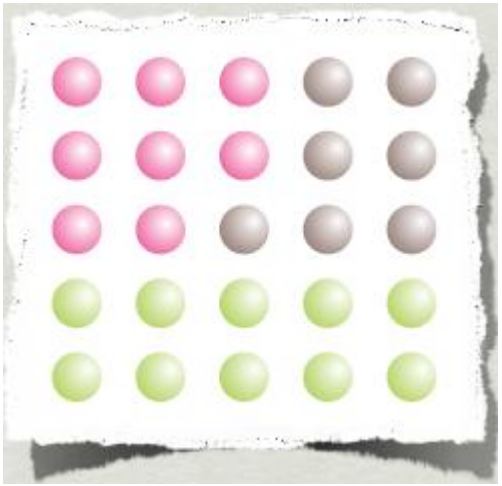
Asynchronous Consensus

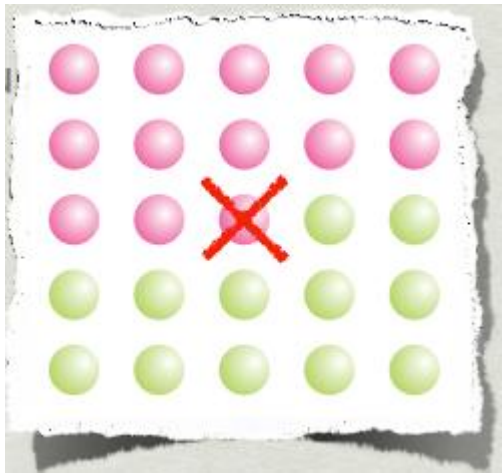
- Solusi 1:
- Ada 1 acceptor
  - Collect all proposals
  - Decides value and tells everyone else

- Approach 2: one acceptor is not enough.
- Let's have multiple *acceptors;*
  - from there, we must reach a decision. How?
- decision = value chosen by the majority;
  - (don't care how this is made known to learners);
- potentially:
  - can take up to N/2 acceptor failures;
  - and still reach consensus.

Problem

- Ada lebih dari 2 value yang harus dipilih
- Unlucky failure
- Tidak bisa mengabaikan yang hilang
  - Asynchronous: mungkin lambat
  - Tidak bisa decide





Prinsip Paxos

- Berbasis algoritma 2
- Multiple acceptor
- value decided = value accepted by a majority;
- to work around the problems:
  - uses a sequence of *views or rounds;*
  - a majority in any view **decides;**
  - if view doesn't work out, later view can supersede it.

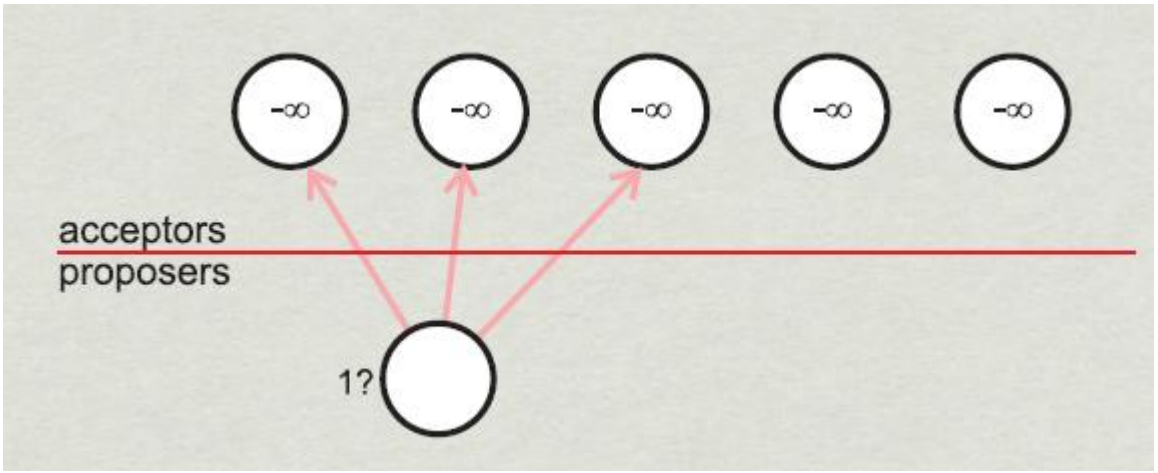
Prinsip Paxos

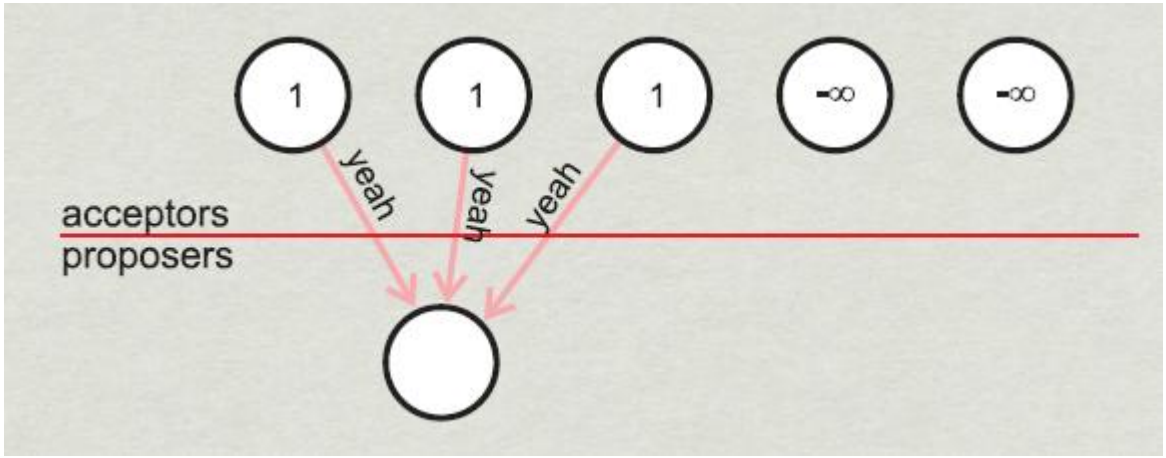
- Problem:
  - decisions in all views must agree.
- Key property ensured by Paxos:
  - once a view reaches a decision, every *view that* supersedes it will either:
  - reach no decision;
  - reach the same decision.

Paxos – 1st phase

- Paxos is a 2-phase protocol (but it's asynchronous).
- A proposer, in the first phase:
  - proposes a **view number to a majority of acceptors;**
  - view numbers are unique across the whole system;
  - view numbers are monotonically increasing.
- An acceptor, in the first phase;
  - **accepts: if it's the largest number it's ever seen;**
  - **rejects it otherwise.**
  - On acceptance, replies with:
    - the largest view number in which it has accepted a value;
    - the last value it has accepted (will make sense in a second);
  - On rejection, might send the largest view value it committed to.

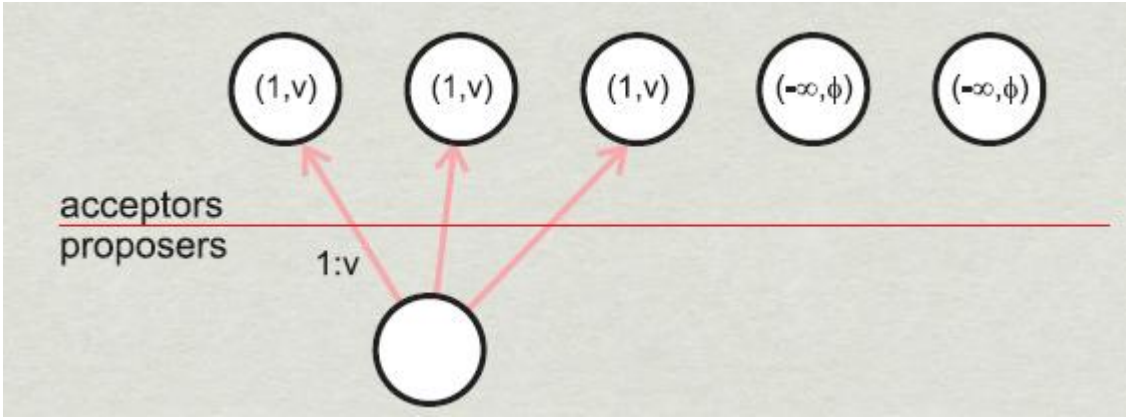
Paxos 1st phase





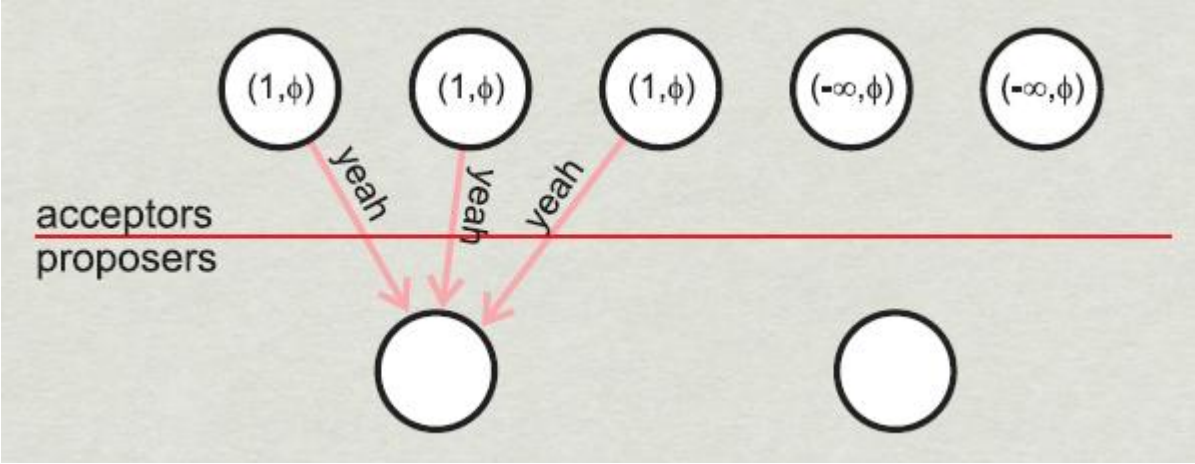
Paxos 2nd phase

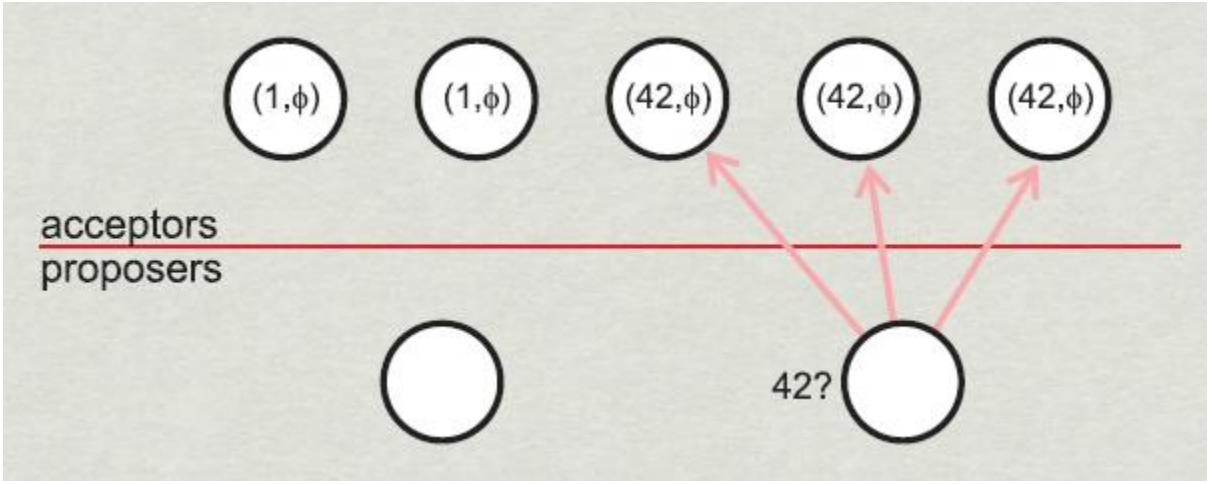
- If a majority of acceptors answers "yes";
- proposer proposes a value to a majority of acceptors;
- if a majority accepts it, the value is **decided**

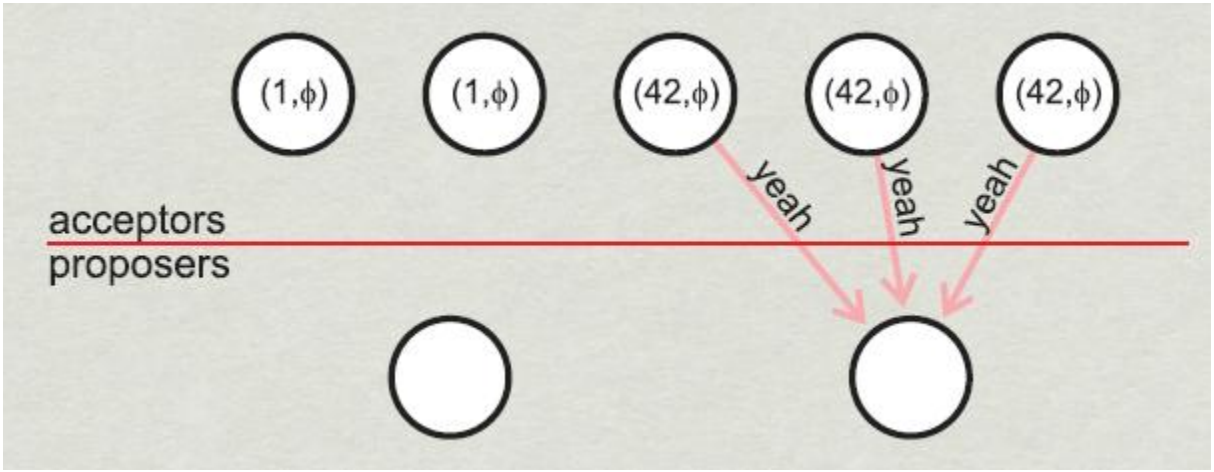


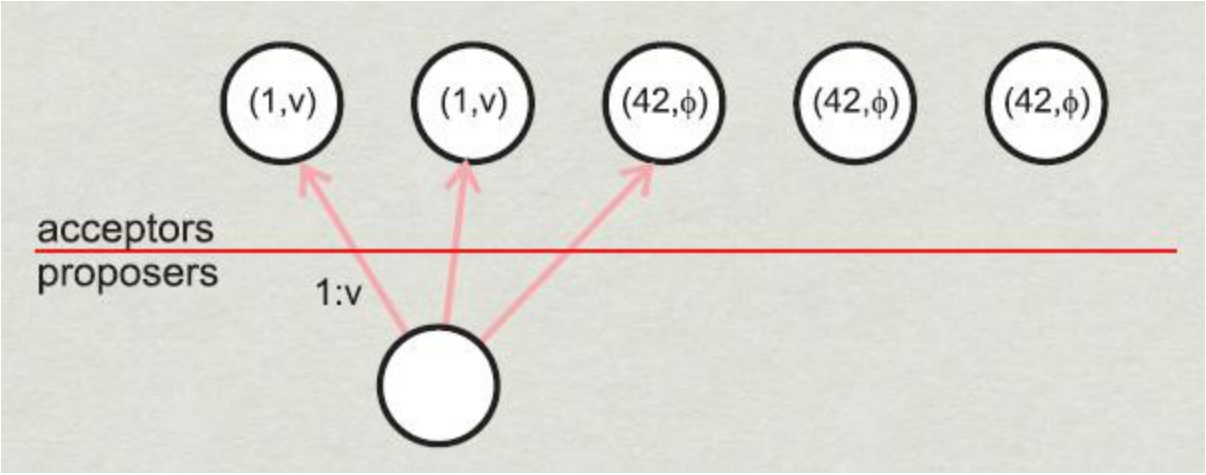
Paxos 2nd phase

- Acceptors only accept a value if no **larger view** number has been proposed in the meantime.



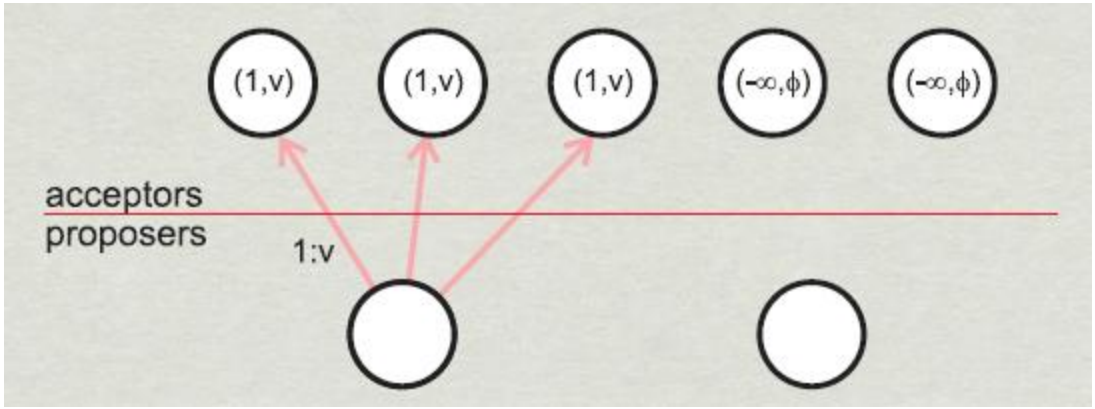

(1,ϕ)
(1,ϕ)
(42,ϕ)
(42,ϕ)
(42,ϕ)
acceptors
proposers
42?

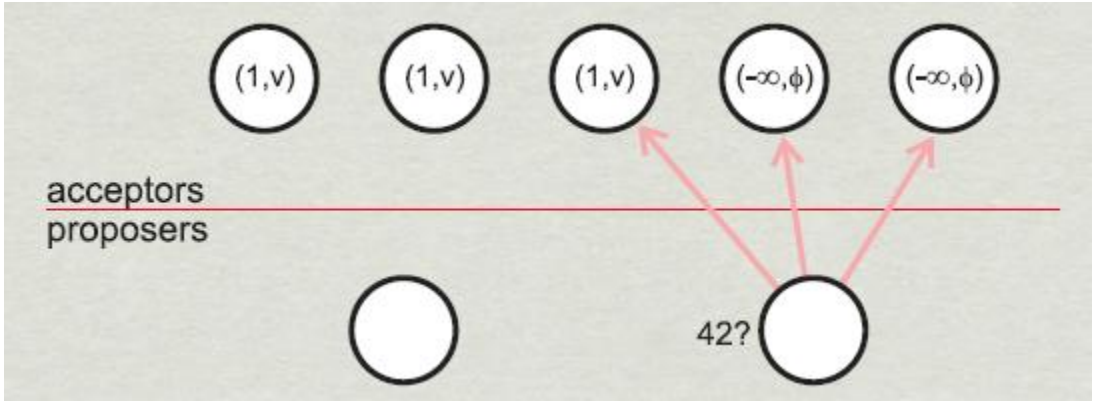

(1,ϕ)
(1,ϕ)
(42,ϕ)
(42,ϕ)
(42,ϕ)
yeah
yeah
yeah
acceptors
proposers

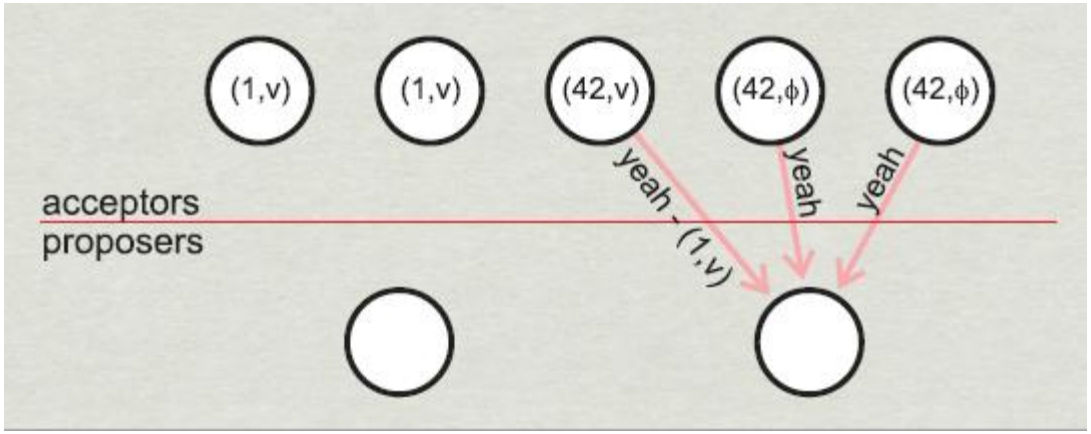

(1,v)
(1,v)
(42,ϕ)
(42,ϕ)
(42,ϕ)
acceptors
proposers
1:v

Paxos 2nd phase

- In Paxos, there are **rules for proposing values.**
- If no values are returned in accept messages from phase 1;
  - proposer is free to pick any proposal value it wants.
- Otherwise;
  - proposer **has to pick a previously proposed value;**
  - the one ACCEPTED in the proposal with the highest number, in a previous round.


(1,v)
(1,v)
(1,v)
(-∞,ϕ)
(-∞,ϕ)
acceptors
proposers
1:v


(1,v)
(1,v)
(1,v)
(-∞,ϕ)
(-∞,ϕ)
acceptors
proposers
42?


(1,v)
(1,v)
(42,v)
(42,ϕ)
(42,ϕ)
yeah
yeah
yeah
(1,v)
acceptors
proposers

Paxos Protocols

- **Phase 1a: Proposer (PREPARE)**
  - A proposer initiates a PREPARE message, picking a unique, ever-incrementing value.
    - `ID = cnt++;`
    - `send PREPARE(ID)`

- **Phase 1b: Acceptor (PROMISE)**
- An acceptor receives a PREPARE(ID) message:

```
if (ID <= max_id)
      do not respond (or respond with a "fail" message)
  else
      max_id = ID      // save highest ID we've seen so far
      if (proposal_accepted == true)
          respond: PROMISE(ID, accepted_ID, accepted_VALUE)
      else
          respond: PROMISE(ID)
```

- **Phase 2a: Proposer (PROPOSE)**
- The proposer now checks to see if it can use its proposal or if it has to use the highest-numbered one it received from among all responses:

```
did I receive PROMISE responses from a majority of acceptors?
if yes
    do any responses contain accepted values (from other proposals)?
    if yes
        val = accepted_VALUE // value from PROMISE message with the
highest accepted ID
    if no
        val = VALUE     // we can use our proposed value
    send PROPOSE(ID, val) to at least a majority of acceptors
```

- **Phase 2b: Acceptor (ACCEPT)**
- Each acceptor receives a PROPOSE(ID, VALUE) message from a proposer. If the ID is the highest number it has processed then accept the proposal and propagate the value to the proposer and to all the learners.

```
if (ID == max_id) // is the ID the largest I have seen so far?
    proposal_accepted = true     // note that we accepted a proposal
    accepted_ID = ID             // save the accepted proposal number
    accepted_VALUE = VALUE       // save the accepted proposal data
    respond: ACCEPTED(ID, VALUE) to the proposer and all learners
else
    do not respond (or respond with a "fail" message)
```

Paxos Safety

- These simple rules bring the following property:

- *Once a value v has been decided in view r, every proposal with number n > r made by ANY PROPOSER will either:*
  - *be for value v;*
  - *fail.*

Why?

- Part of it is somewhat obvious;
- decision = majority;
- any two majorities share at least one element;
- therefore:
- after the first **decision;**
- any subsequent view, either:
- does not contain n/2 + 1 acceptors;
- contains one element that accepted *v*.

Why?

- But, that's not the end of it;
- to ensure that *v is proposed, we must ensure that v was always the* highest-numbered value to be accepted in any majority.
- Now, if value *v got accepted by a majority;*
  - then, it had the largest view number across the whole majority;
  - obvious, or it wouldn't have been accepted;
  - **less obvious: immediately after acceptance, it has the highest** number among all accepted values;

Failure

- What happens after success?
- Safety: the answer is in the network;
- learners might learn it when the system becomes
- stable again.
- Acceptors or proposers may fail, responses may be lost;
- only delays progress or knowledge of consensus;
- does not affect safety

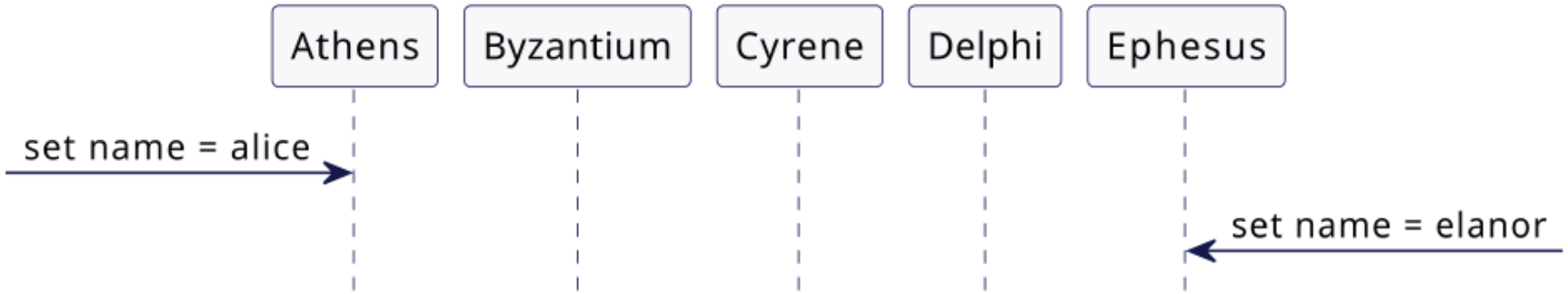
Liveness

- Note that:
  - multiple proposers may delay progress;
  - proposer 1: completes phase 1 by proposing number n1;
  - proposer 2: completes phase 1 by proposing n2 > n1;
  - proposer 1: completes phase 1 by proposing n3 > n2;
  - *ad infinitum.*

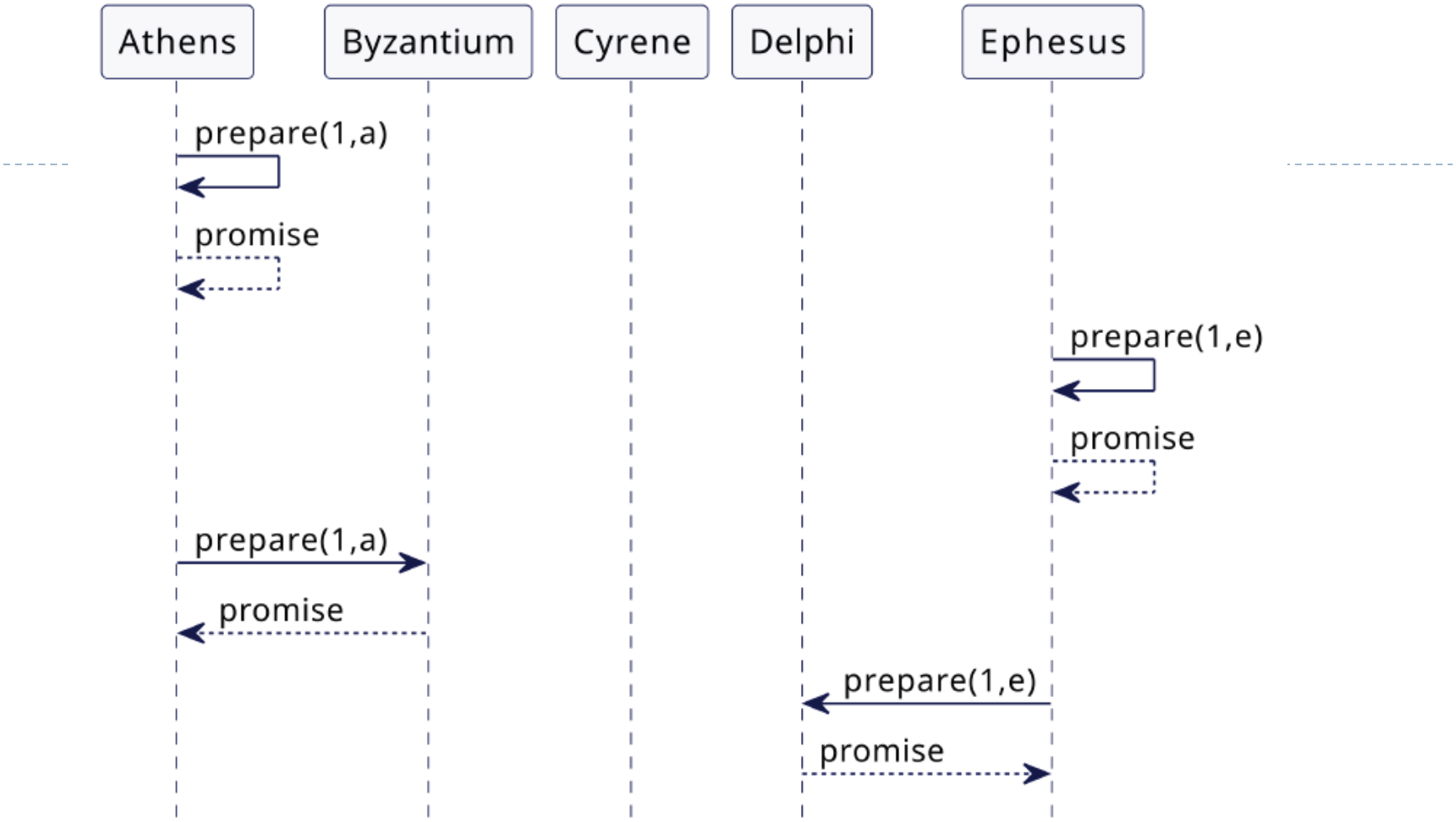
Liveness

- In practice:
- people try to maintain a single proposer at any time;
- leader election, leases;
  - again, if it fails, it's okay. Algorithm does not block, though progress may be impaired;
  - if two nodes think they're leaders, it's okay as well.
- Multiple roles may be played by each node (e.g., all acceptors are learners).

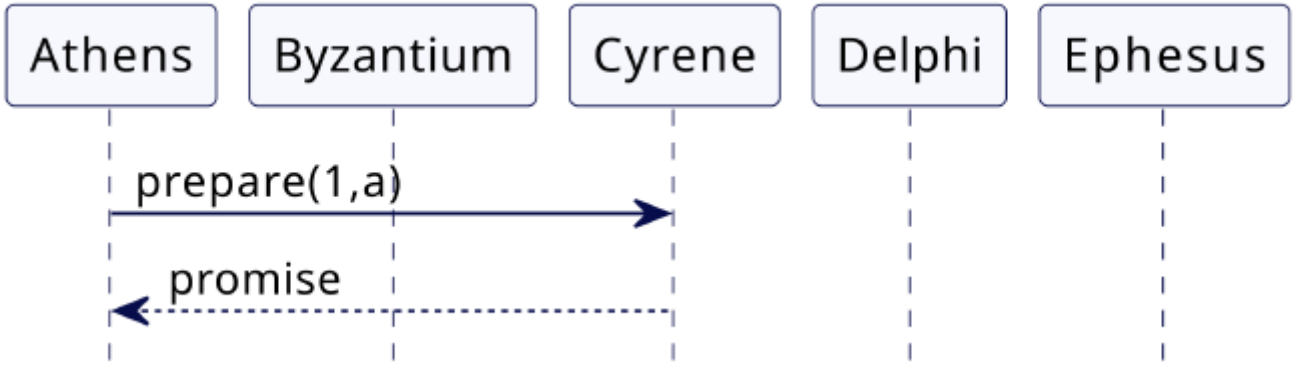
Ilustrasi Paxos



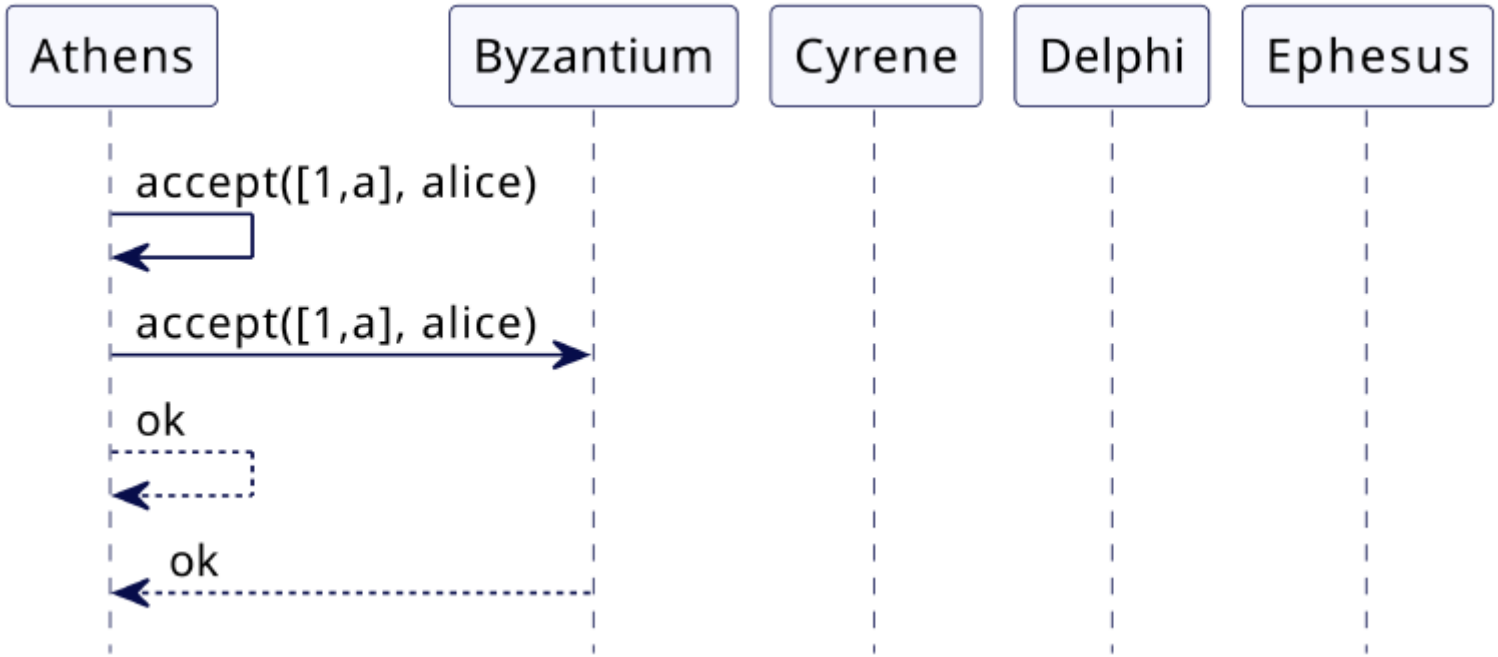
Joshi, Unmesh. *Patterns of distributed systems*. Addison-Wesley Professional, 2023



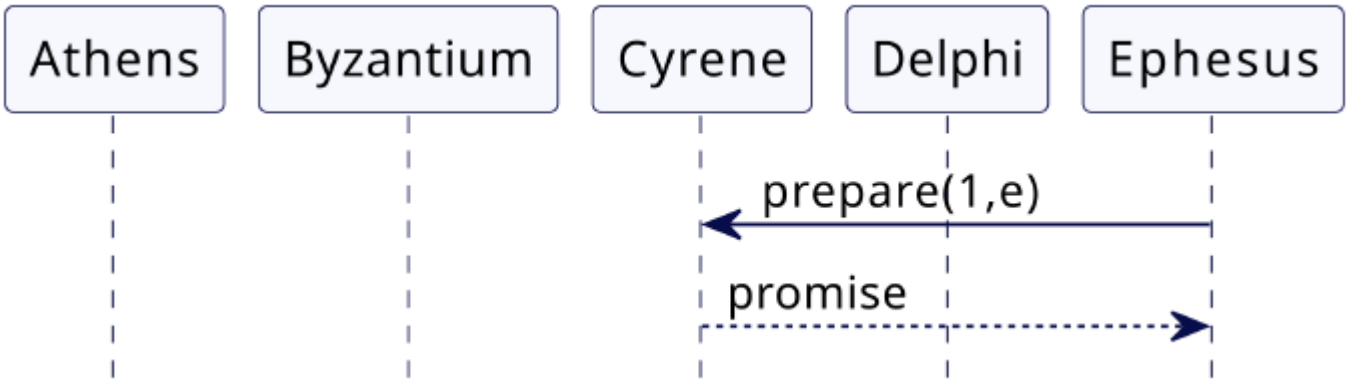
Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	1,a	1,a	0	1,e	1,e
Accepted value	none	none	none	none	none

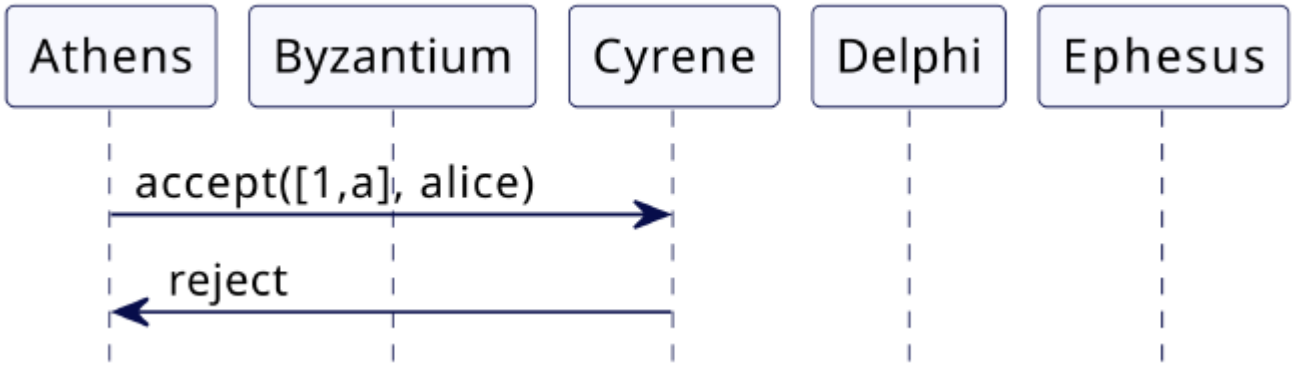


Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	1,a	1,a	1,a	1,e	1,e
Accepted value	none	none	none	none	none

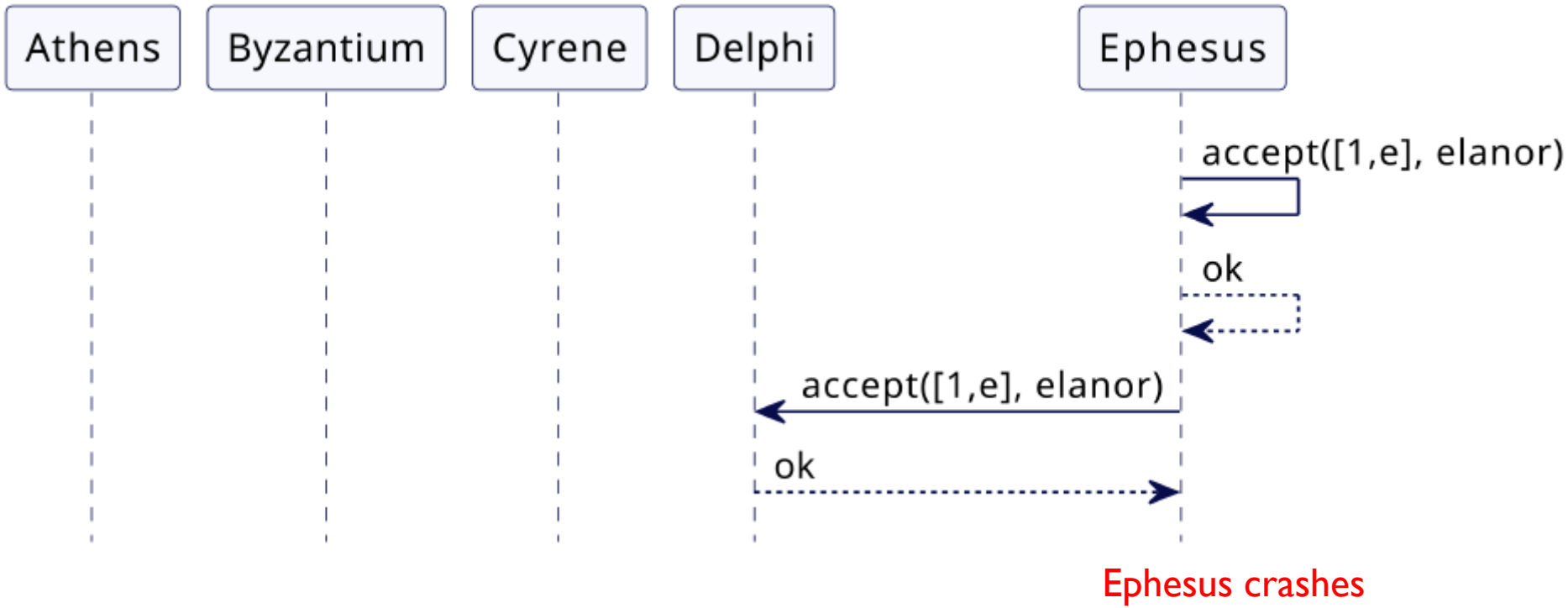


Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	1,a	1,a	1,a	1,e	1,e
Accepted value	alice	alice	none	none	none

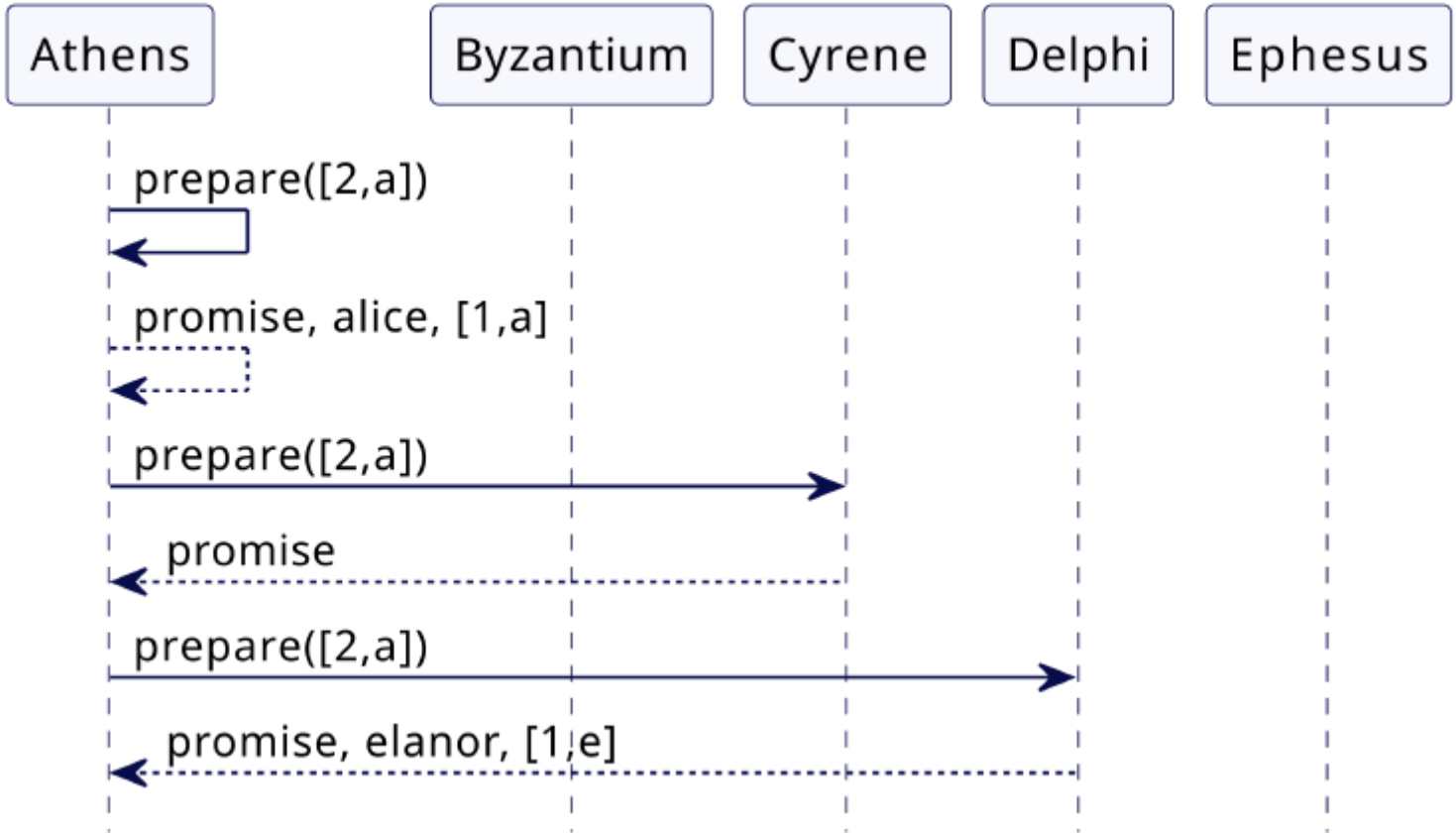




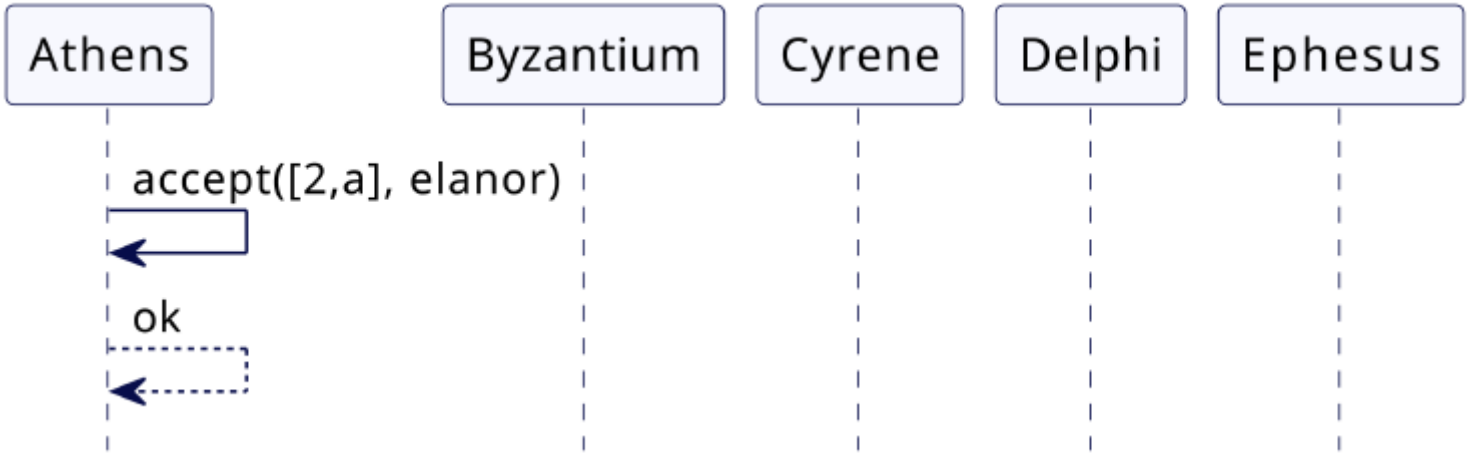
Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	1,a	1,a	1,e	1,e	1,e
Accepted value	alice	alice	none	none	none



Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	1,a	1,a	1,e	1,e	1,e
Accepted value	alice	alice	none	elanor	elanor



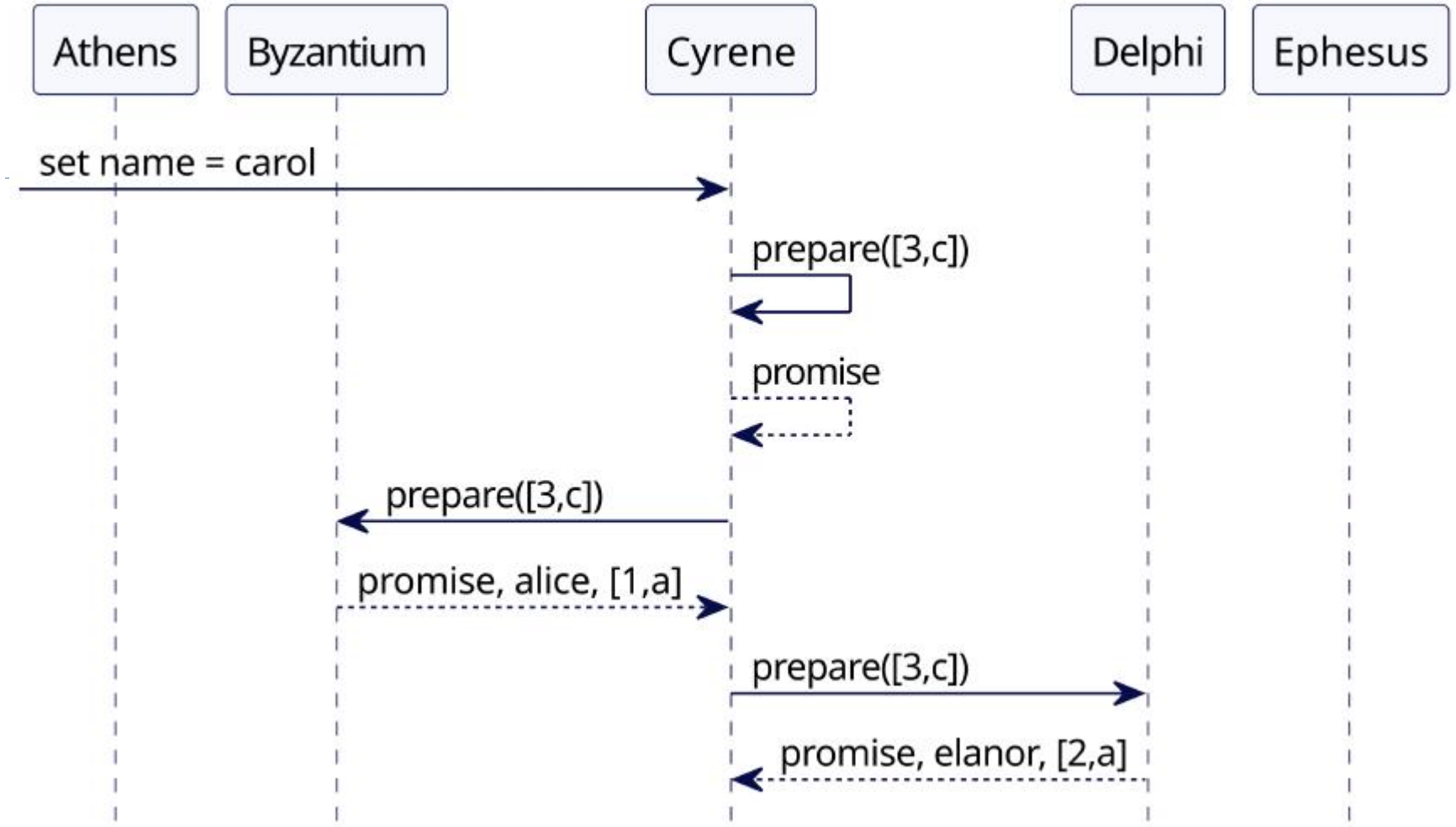
Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	2,a	1,a	2,a	2,a	1,e
Accepted value	alice	alice	none	elanor	elanor



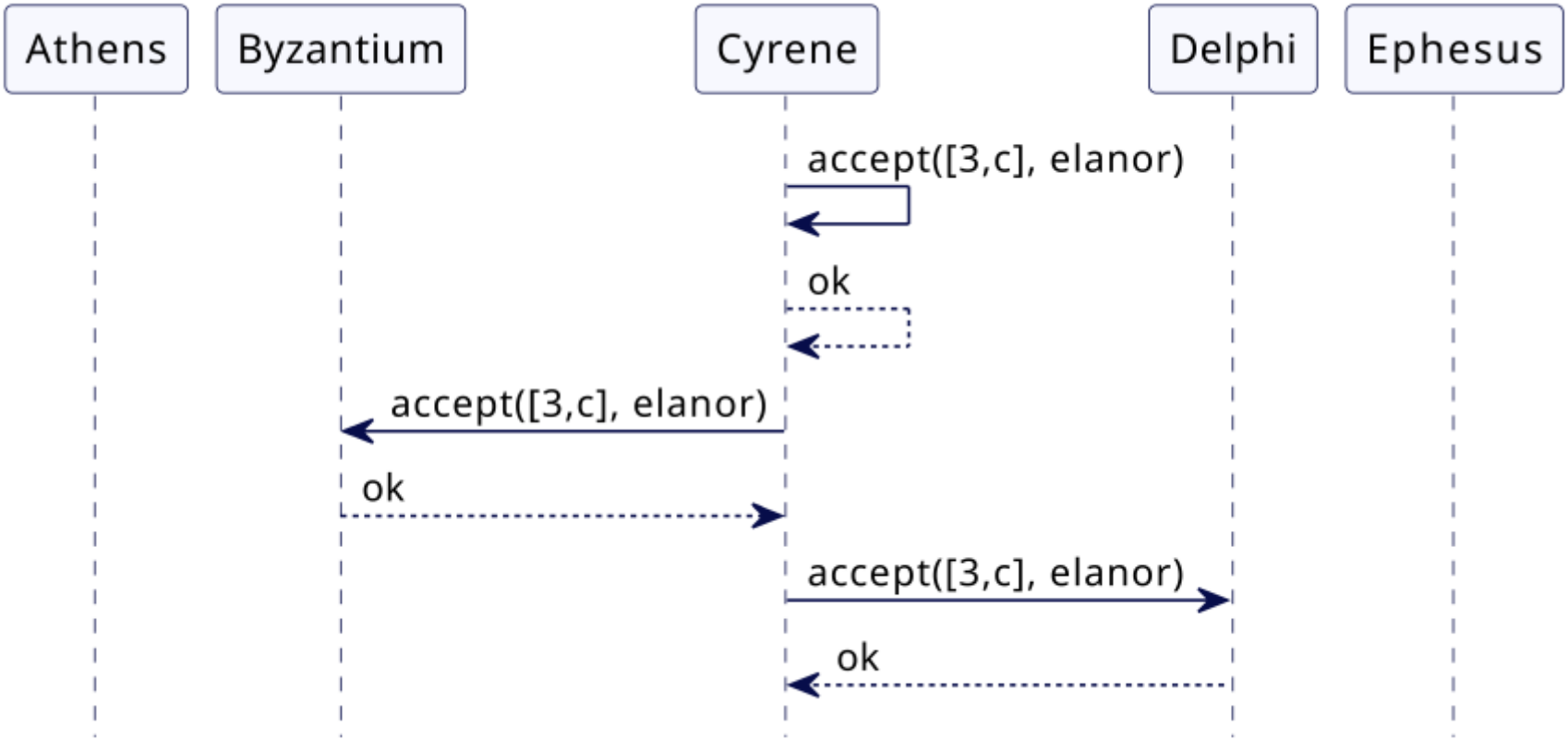
athens crash

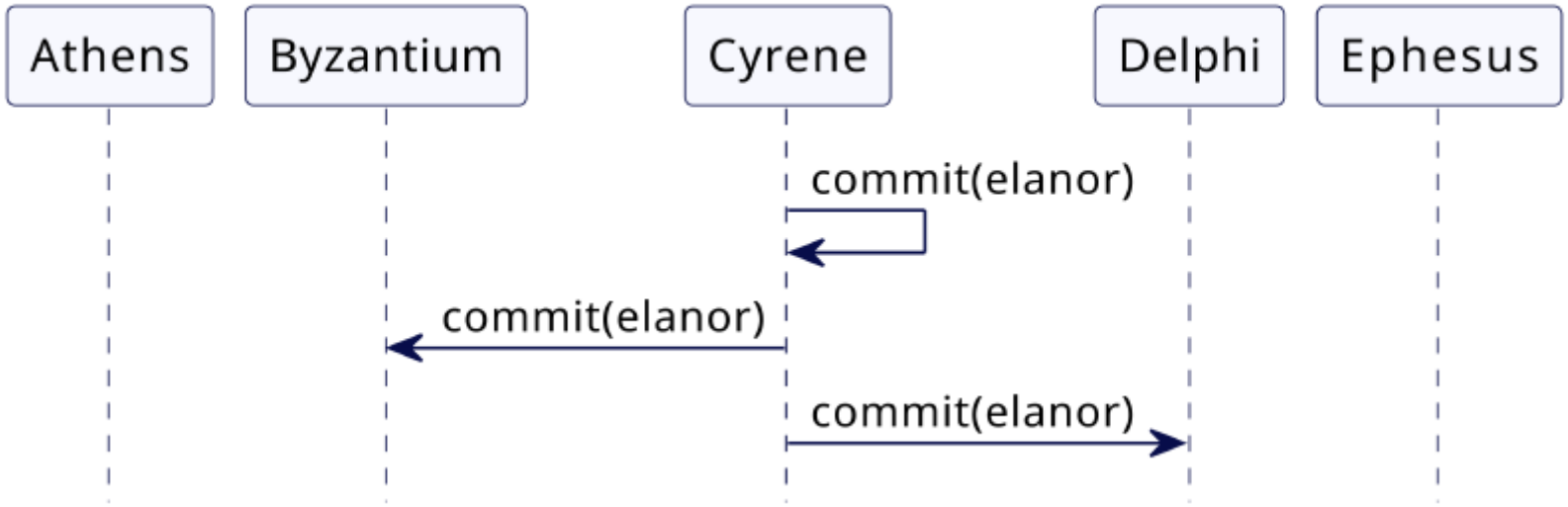
Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	2,a	1,a	2,a	2,a	1,e
Accepted value	elanor	alice	none	elanor	elanor

elanor accepted by majority, but nobody knows yet



Node	Athens	Byzantium	Cyrene	Delphi	Ephesus
Promised generation	2,a	3,c	3,c	3,c	1,e
Accepted value	elanor	alice	none	elanor	elanor


Athens
Byzantium
Cyrene
Delphi
Ephesus
accept([3,c], elanor)
ok
accept([3,c], elanor)
ok
accept([3,c], elanor)
ok


Athens
Byzantium
Cyrene
Delphi
Ephesus
commit(elanor)
commit(elanor)
commit(elanor)

Sumber

- Lamport, Leslie. "Paxos made simple." *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001): 51-58.
- Joshi, Unmesh. *Patterns of distributed systems*. Addison-Wesley Professional, 2023