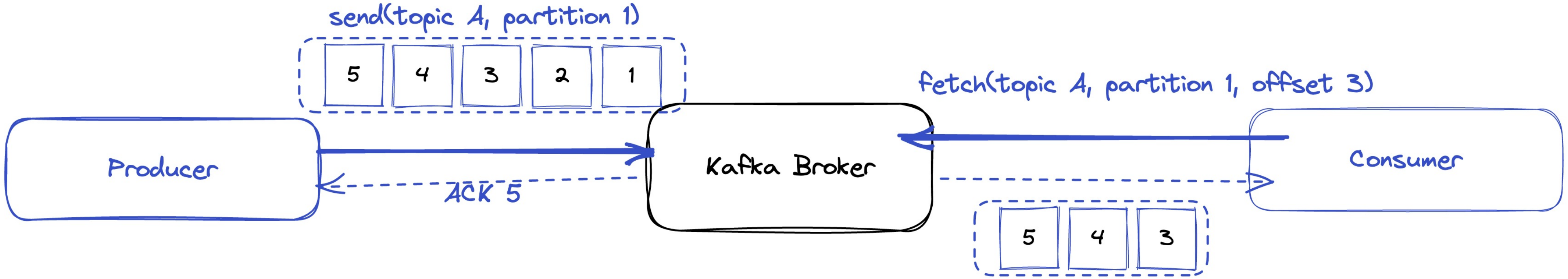
CRITEO

Introduction to Consensus

Ilyas Toumlilt
i.toumlilt@criteo.com

Previously on Kafka

How can Kafka scale ? (clients-side)


send(topic A, partition 1)
5
4
3
2
1
Producer
ACK 5
Kafka Broker
fetch(topic A, partition 1, offset 3)
Consumer
5
4
3

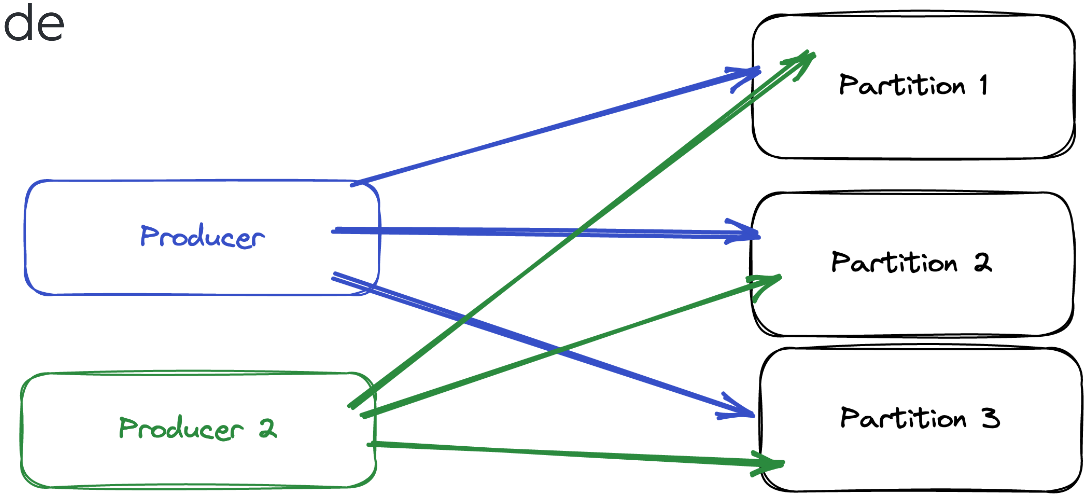
- Multiple producers
- Multiple consumer with different consumer group doesn’t share workload(partitions)
- Multiple consumer with the same consumer group
  - Each consumer in same group can assign at least one partitions

    N(consumer_in_same_group) < N(partition_topic)

How can Kafka scale ? (clients-side)

- Many producers yes but how do they decide which partition(in same topic) to write ?

Problem: how do we balance each partition ?



- By default round-robin fashion.
- Key partitioning:

  partition = hash(key) % number_of_partition

  Key could be something like country=France

  - What happens if partition increased ?
- Not good enough: customise your own partition(by extending the partition class).

Kafka producer-side configs

- Producer ACK: The number of acknowledgments the producer requires the leader to have received before considering a request complete. This affects durability. (Options: acks=0, acks=1, acks=-1(all))

- Producer Idempotency(enable.idempotence = true):

  When set to 'true', the producer will ensure that exactly one copy of each message is written in the stream.

  If 'false’, producer retries due to broker failures, etc., may write duplicates of the retried message in the stream

Fallacies of distributed computing

Super easy to do mistake, things will fail eventually:

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. Network is secure
5. Network topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Consensus

One of the most important abstractions for distributed systems is **consensus**:

- Getting all of the nodes to agree on something (the leadership, lock etc).

Consensus can be used to define a leader across all nodes.

If the leader dies, the remaining nodes can use consensus to elect a new leader.

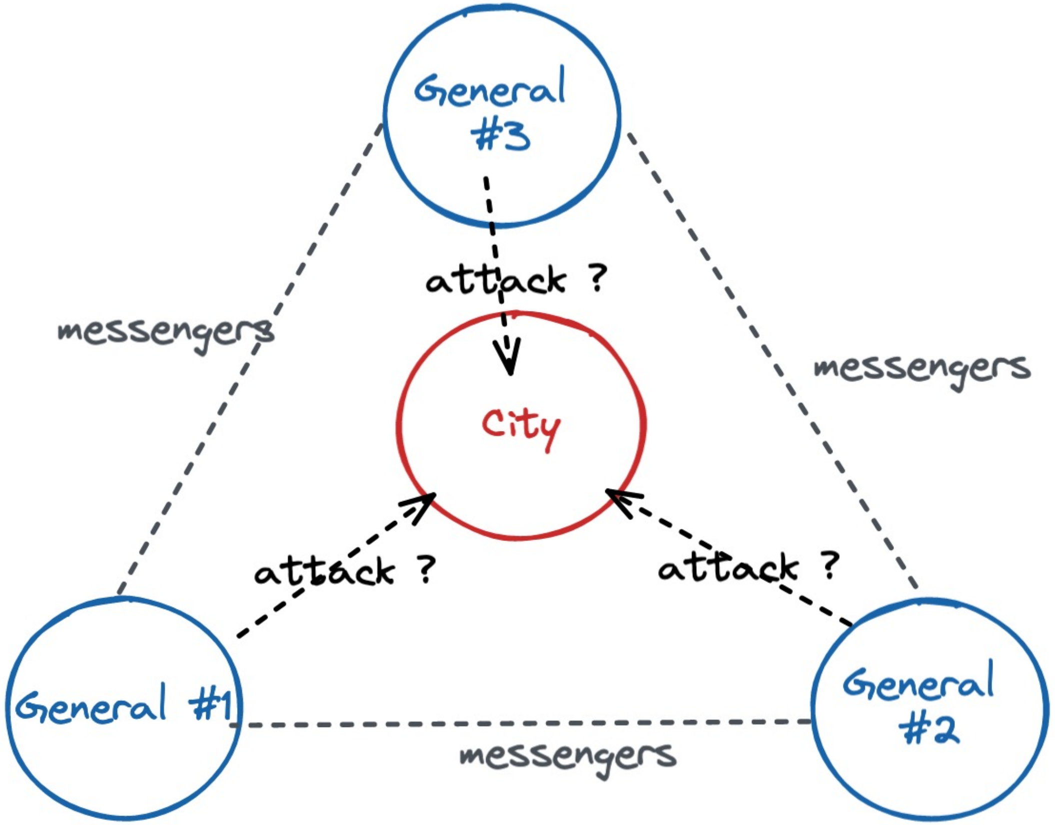
If two nodes both believe that they are the leader, that situation is called **split brain**, and it often leads to data loss.

Reliably reaching consensus in spite of network faults and process failures is a tricky problem.

Byzantine generals problem

- Generals will decide attack/retrait
- Assuming messaging reliable
- Some general/s could be traitors(lying)
- Published on ACM TOPLAS in 1982 by Lamport et al.
- Super expensive

- Real life example:
  * Boeing 777 & 787 flights control



CRITEO

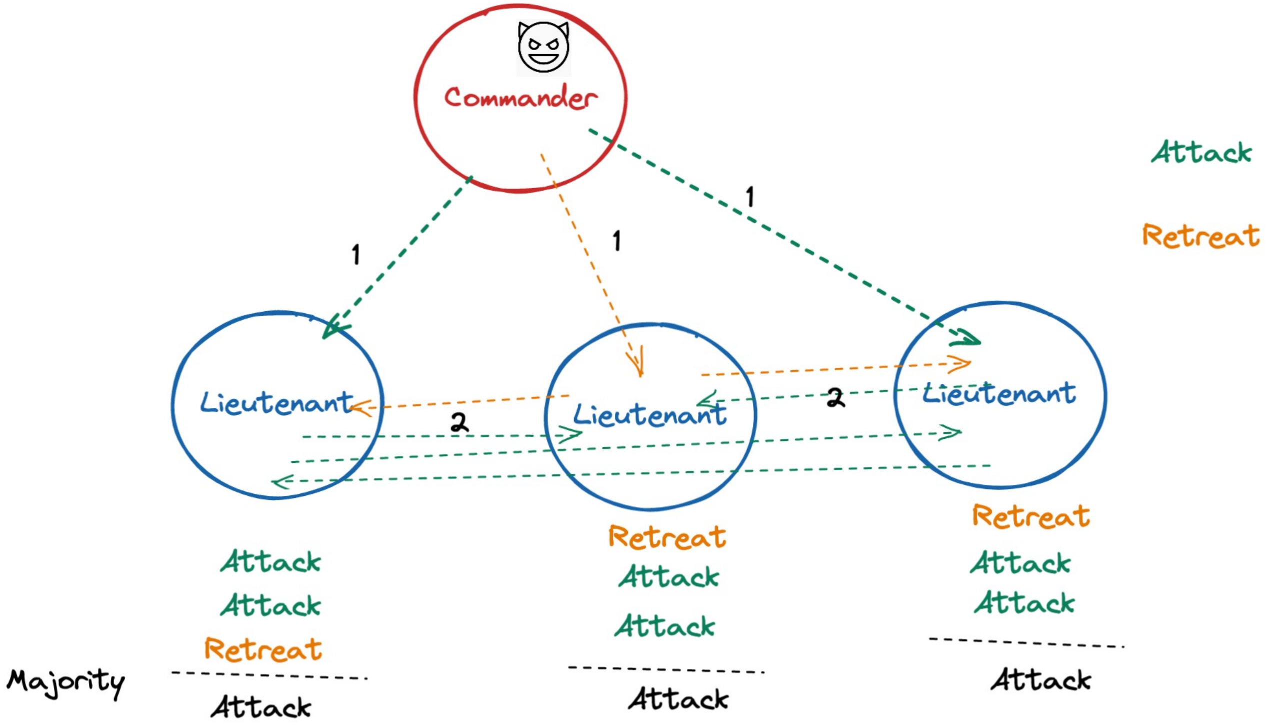
Byzantine generals problem

- Honest generals don’t know who the malicious ones are.
- Malicious generals may know other malicious generals.

- If you have *f* g traitor generals, you need at least *3f+1* generals to if

Byzantine generals problem: CT

- n = 4 (let's solve inductively) : Commander traitors



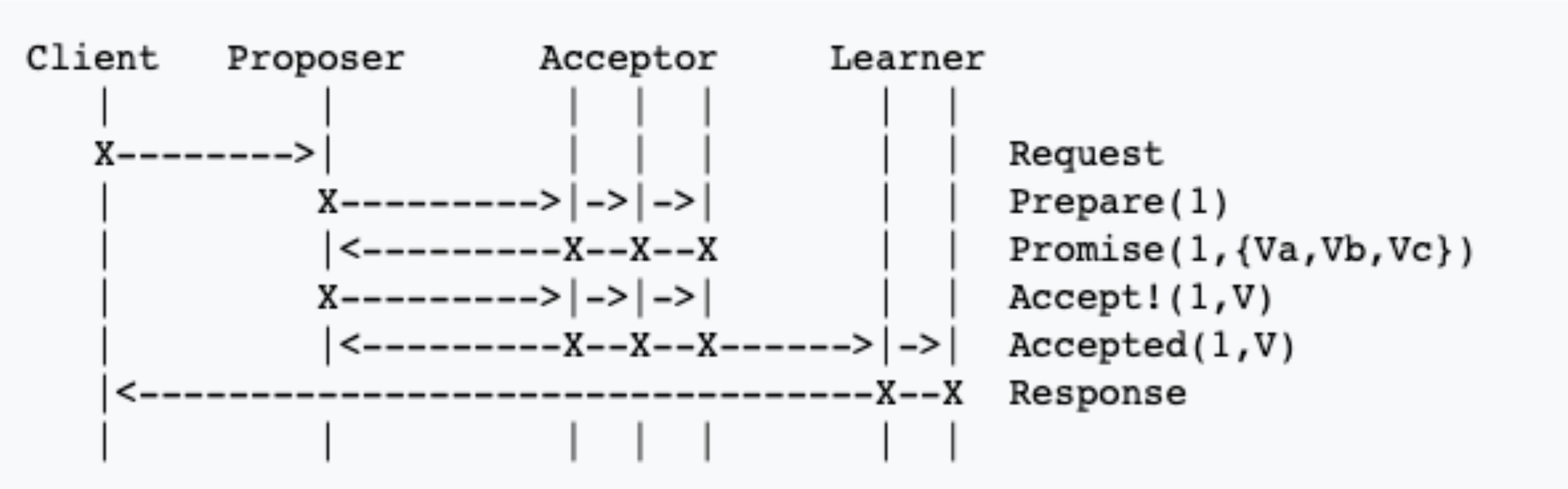
- *SLOW (Expensive): How many message exchange required ?*
- *Prove in the same way when N = 7 max traitor 2*

CRITEO

Byzantine failure

- Byzantine fault is a condition such a component in distributed environment may misbehave inconsistently (failed and functioning in failure-detection).
- It is a difficult problem because first nodes should have consensus on failing one then act accordingly.

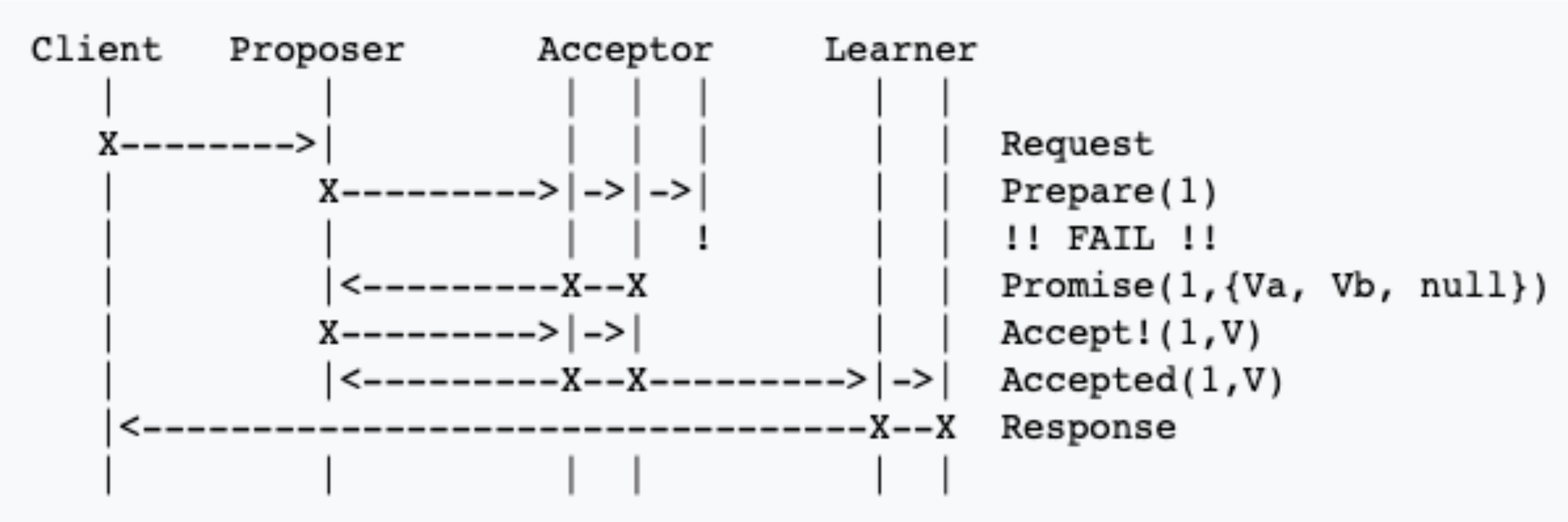
How does Paxos work ? (no failures)



```
Client   Proposer      Acceptor     Learner
   |         |          |  |  |       |  |
   X-------->|          |  |  |       |  |  Request
   |         X--------->|->|->|       |  |  Prepare(1)
   |         |<---------X--X--X       |  |  Promise(1,{Va,Vb,Vc})
   |         X--------->|->|->|       |  |  Accept!(1,V)
   |         |<---------X--X--X------>|->|  Accepted(1,V)
   |<---------------------------------X--X  Response
   |         |          |  |  |       |  |
```

CRITEO

How does Paxos work ? Failure on Acceptor



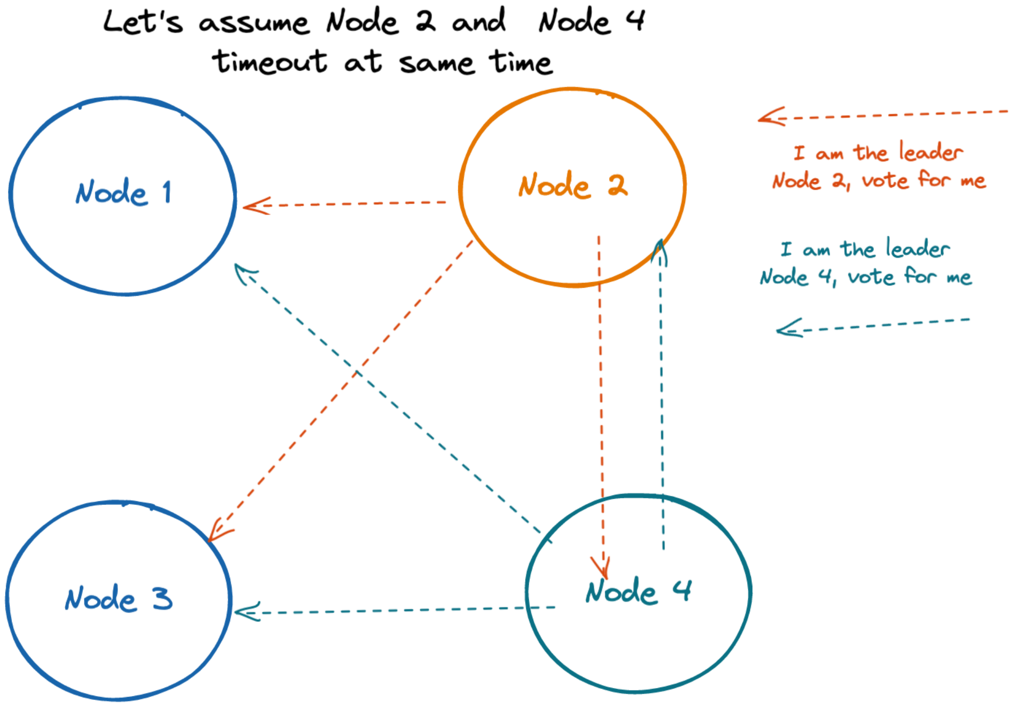
```
Client    Proposer       Acceptor      Learner
   |         |          |  |  |       |  |
   X-------->|          |  |  |       |  |  Request
   |         X--------->|->|->|       |  |  Prepare(1)
   |         |          |  |  !       |  |  !! FAIL !!
   |         |<---------X--X          |  |  Promise(1,{Va, Vb, null})
   |         X--------->|->|          |  |  Accept!(1,V)
   |         |<---------X--X--------->|->|  Accepted(1,V)
   |<---------------------------------X--X  Response
   |         |          |  |          |  |
```

Two scenarios : - Prepare -> Do I have majority ?
- Accept(Propose) Yes, it is fine; No, let's retry again

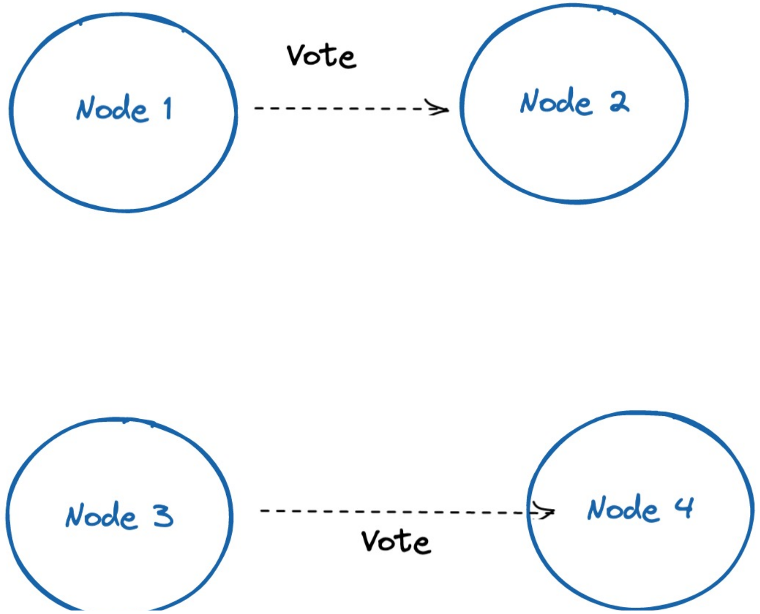
Raft: Leader Election – Split Vote

for instance:

i) time outing and
election voting start phase



ii) voting phase



iii) result

No majority, let's try again