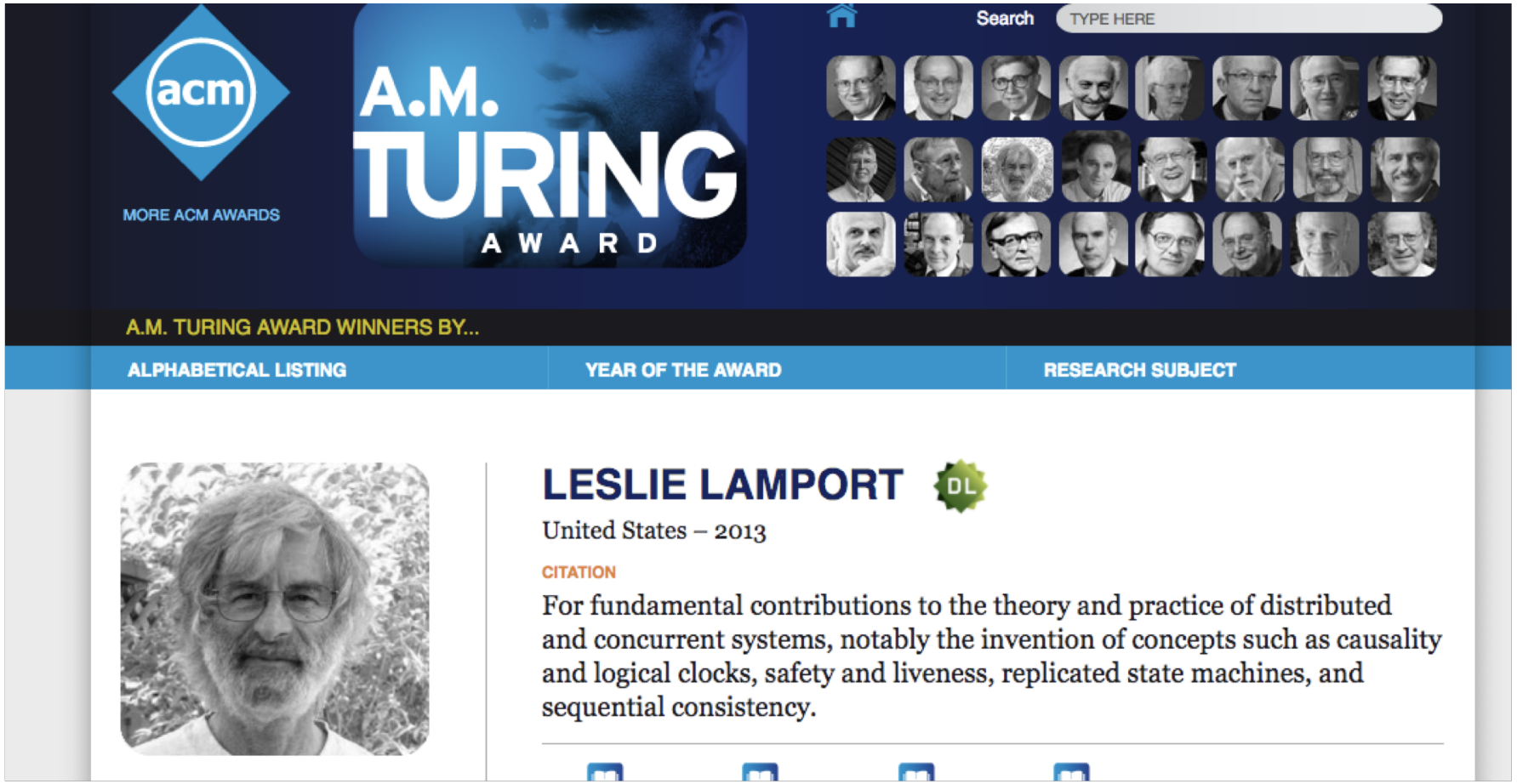
Paxos

Zhaoguo Wang

Some slides are adapted from Prof. Jinyang Li’s DS class

Paxos Solves The Consensus Problem



Correctness: servers execute the same sequence of log

No two different ops are committed at the same log position

All committed ops in view V - 1 must be known to primary in view V

Correctness: servers execute the same sequence of log

No two different ops are committed at the same log position

All committed ops in view V - 1 must be known to primary in view V

Single Decree Consensus

Single Decree Consensus

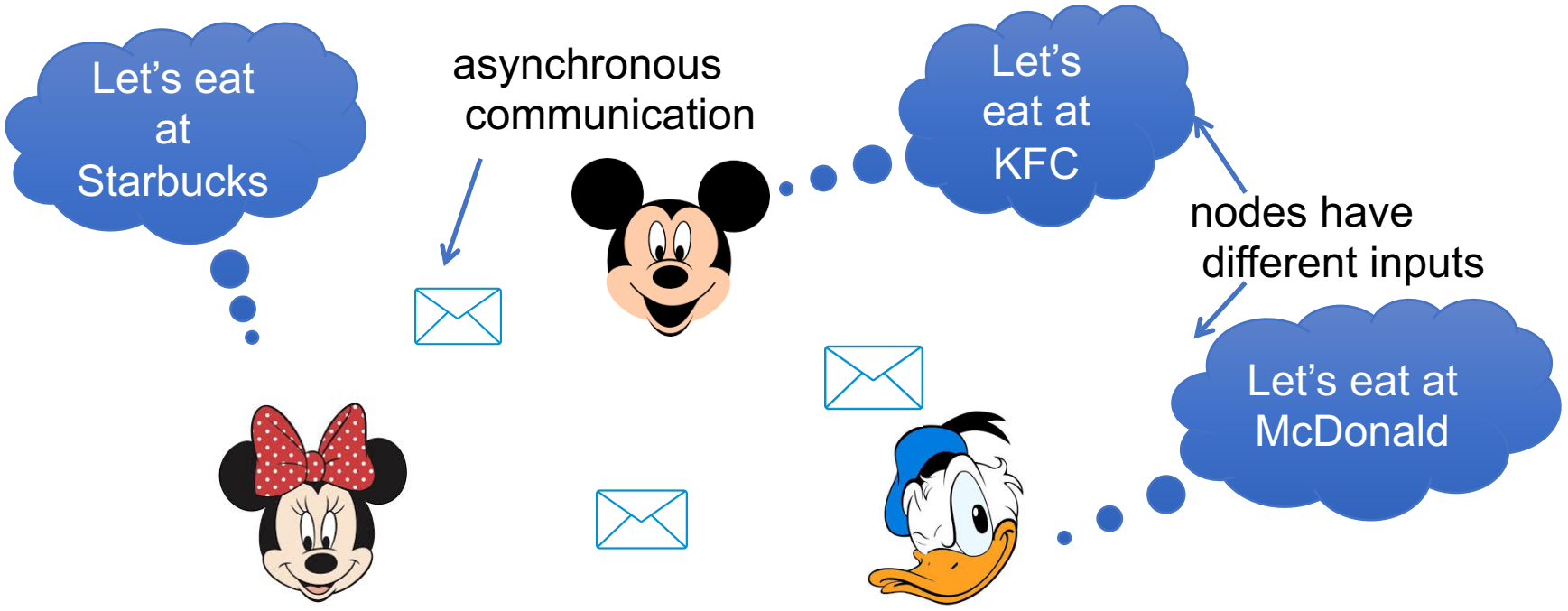
Problem setting

- N nodes, each with a (potentially) different input
- One or more may have non-byzantine fail
- Network is asynchronous: cannot tell node crash from slow communication

Goal

- Safety: chosen values by different nodes are the same
- Liveness: all live nodes eventually choose some value

Single-decree consensus



Paxos Components

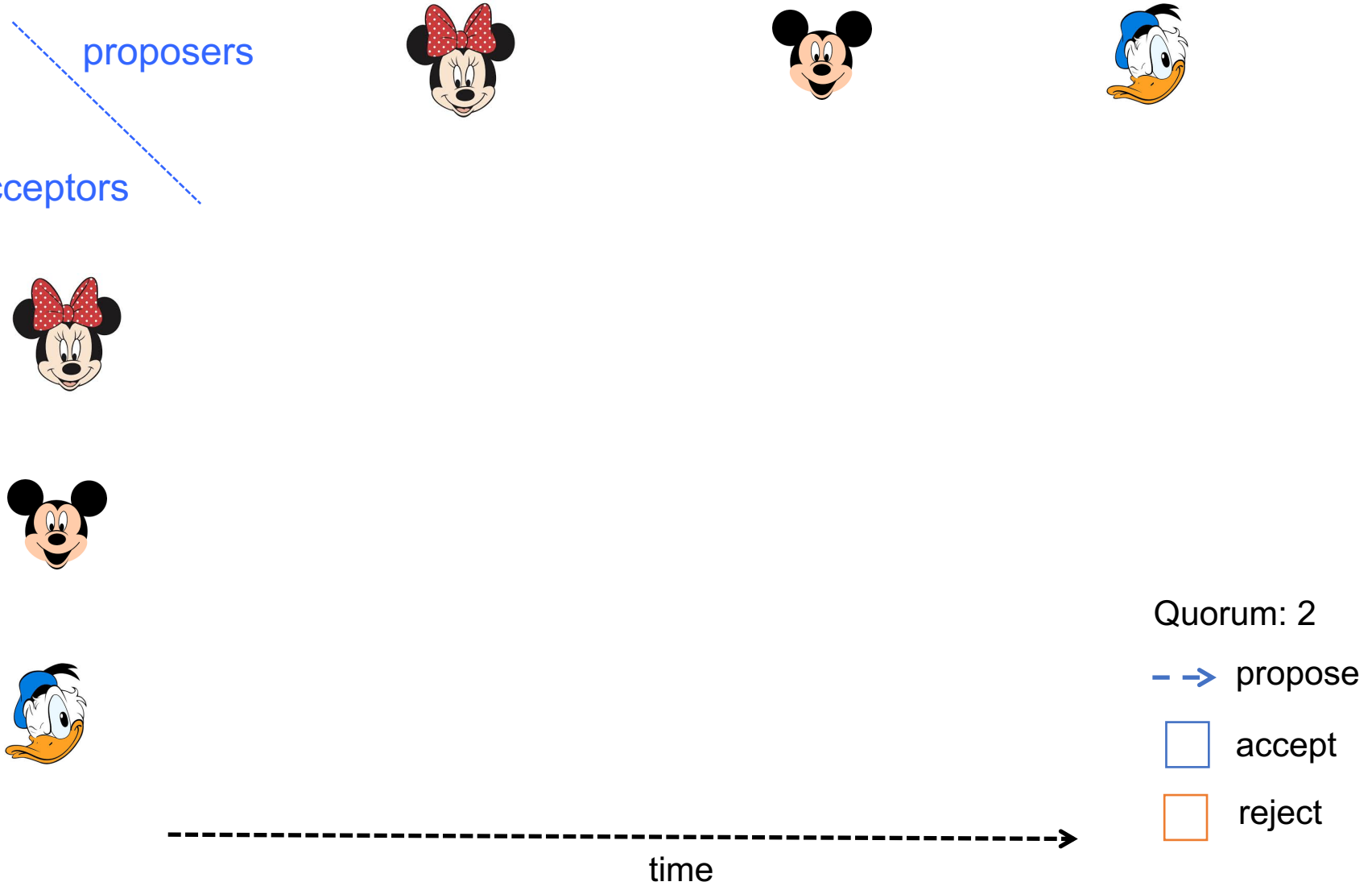
Roles

- Proposers: send proposed values to acceptors
- Acceptors: accept/reject proposed values from proposer
- Leaners: learn about outcome of consensus (chosen value)
- For simplicity, every node plays all the roles

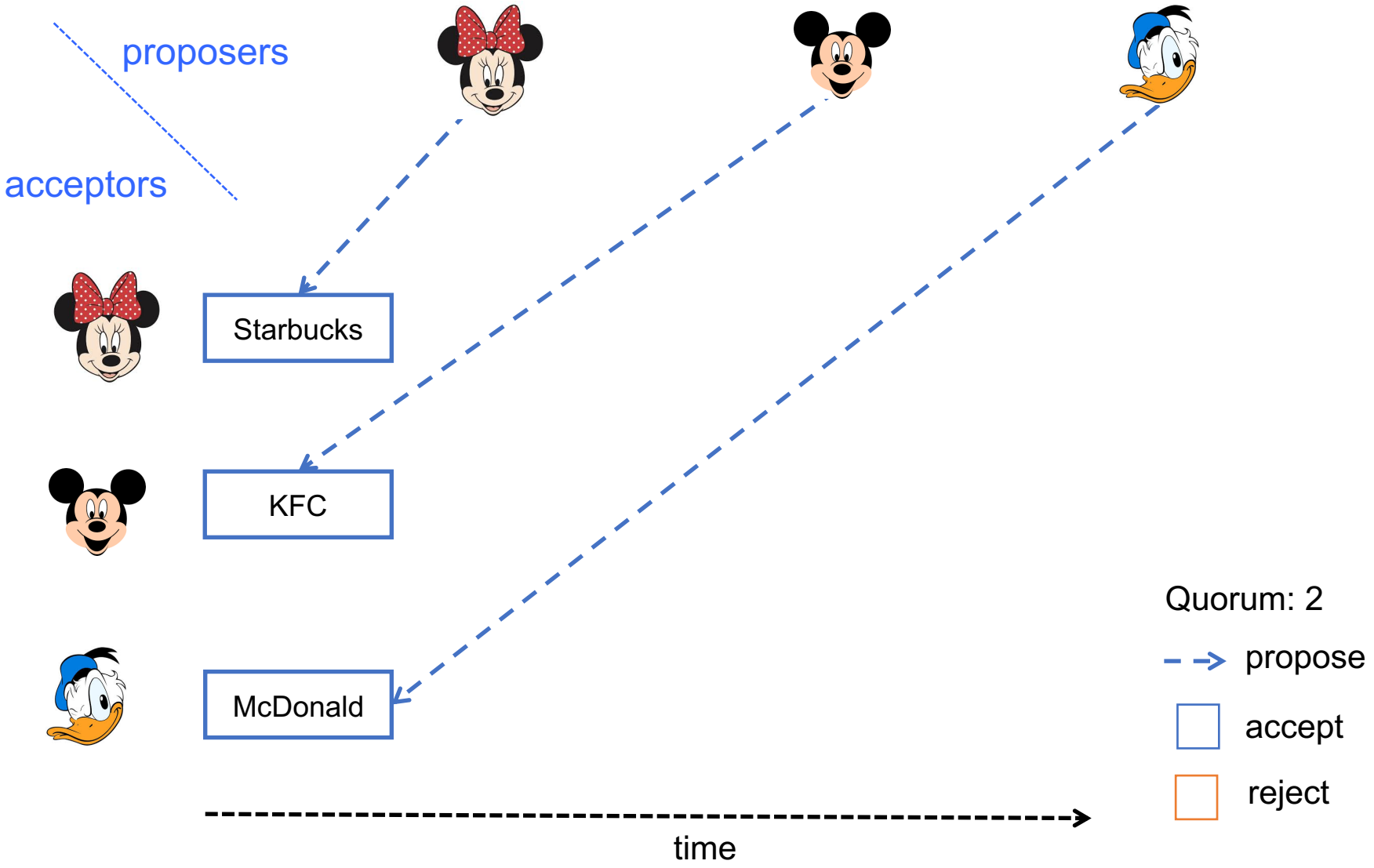
Chosen

- A value is **chosen (decided)** if it is accepted by a quorum of nodes

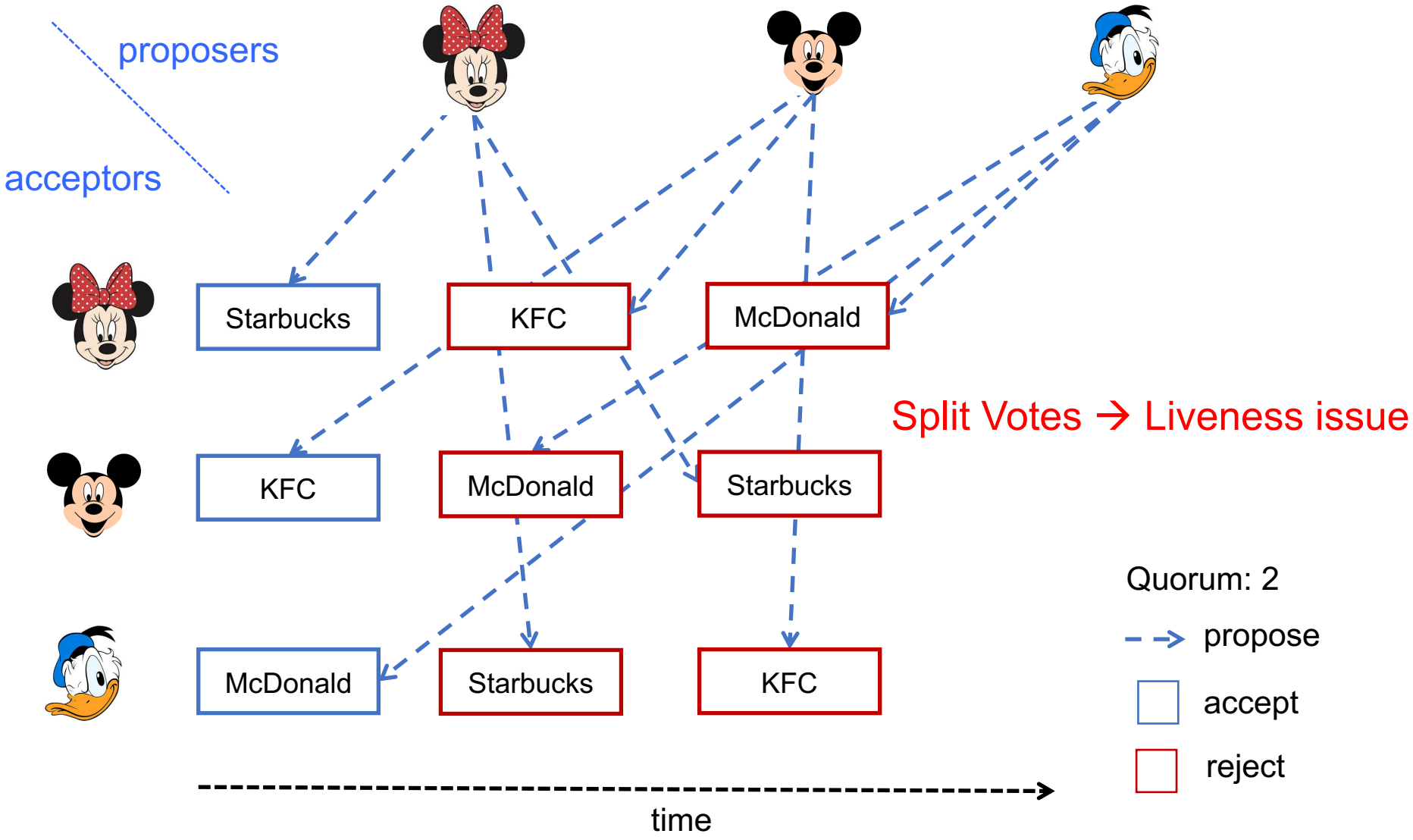
Method I – Only accept the first



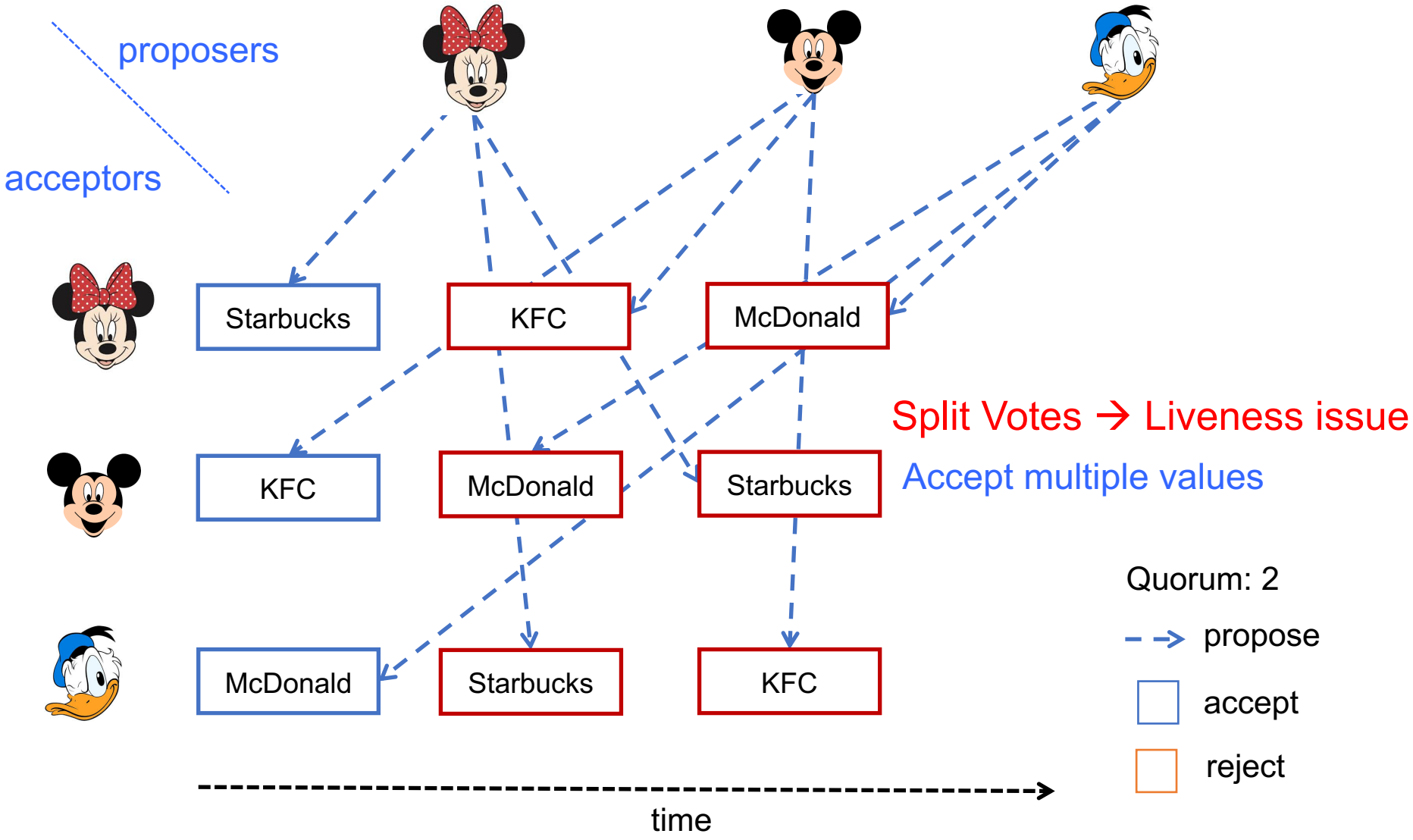
Method I – Only accept the first



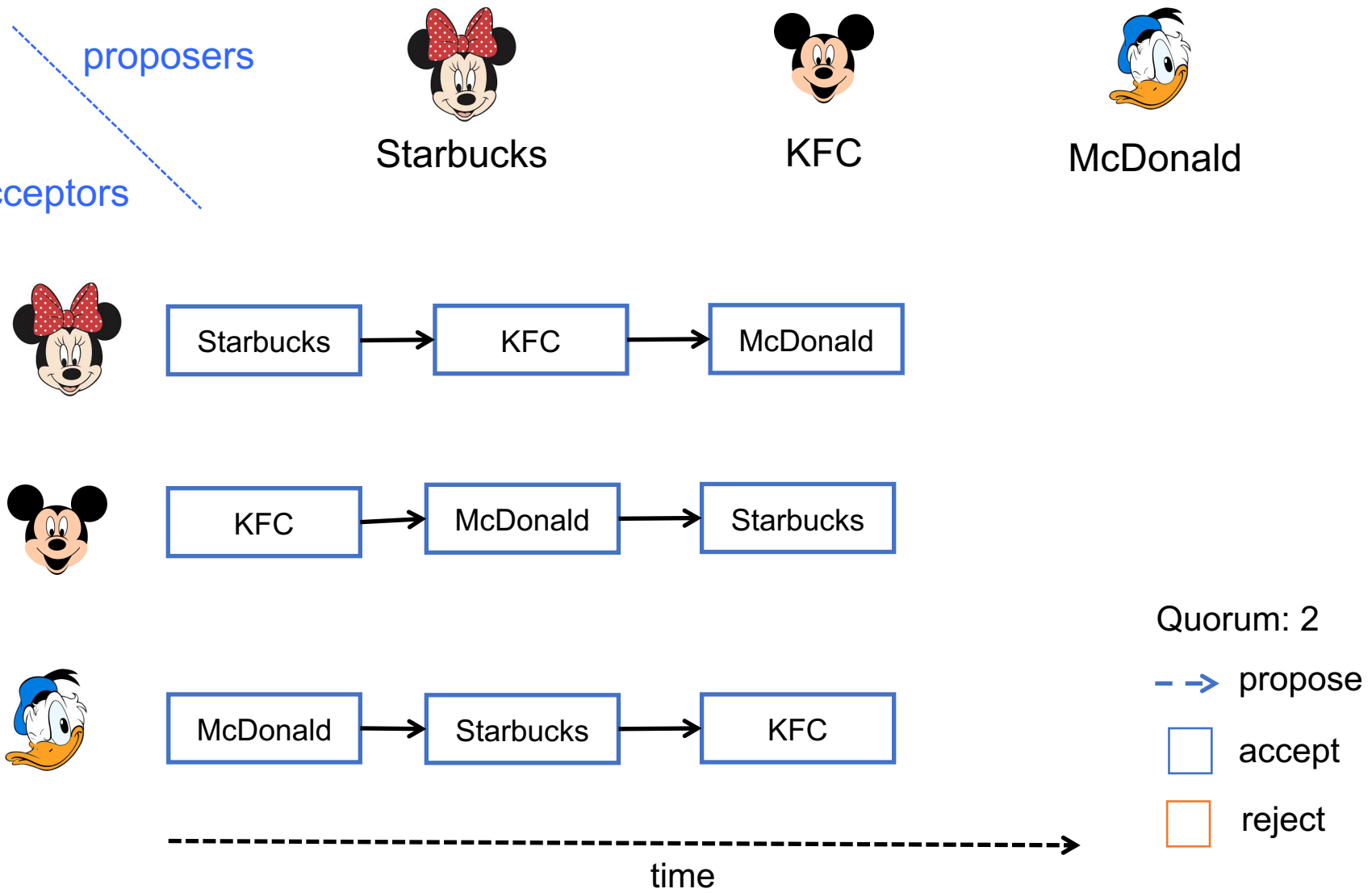
Method I – Only accept the first



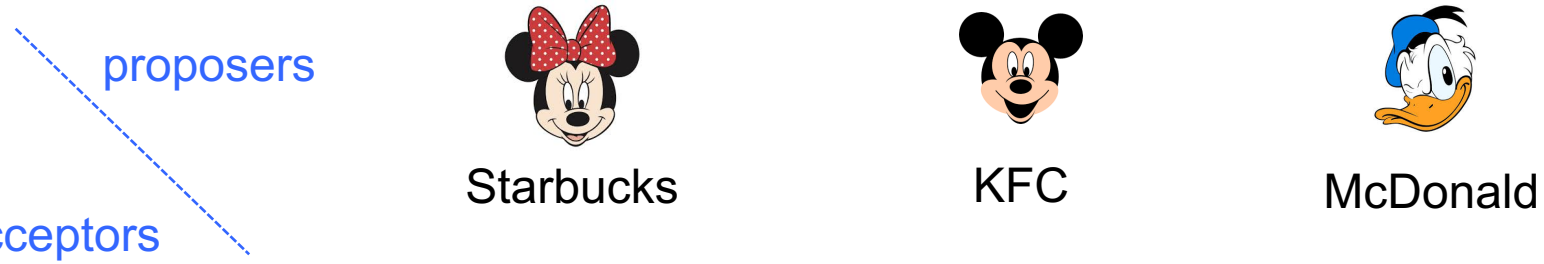
Method I – Only accept the first



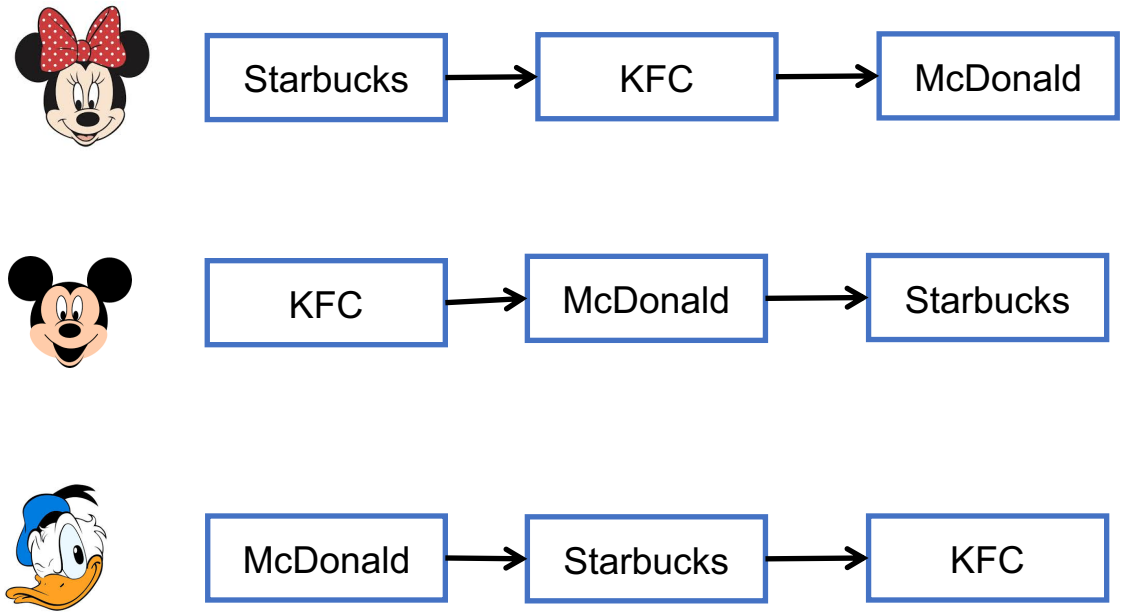
Method II – Accept multiple proposed values



Method II – Accept multiple proposed values



Multiple chosen values → Consensus issue



If a proposal is chosen, all subsequent accepted proposals must have the same value.

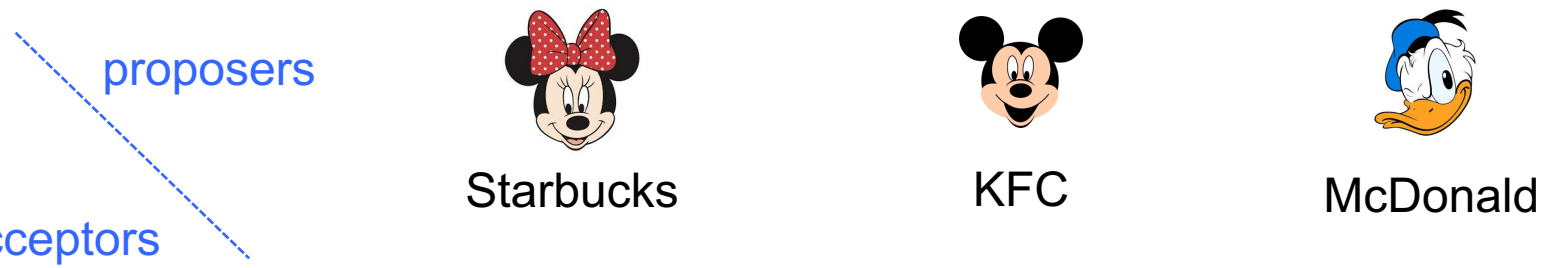
Quorum: 2

propose

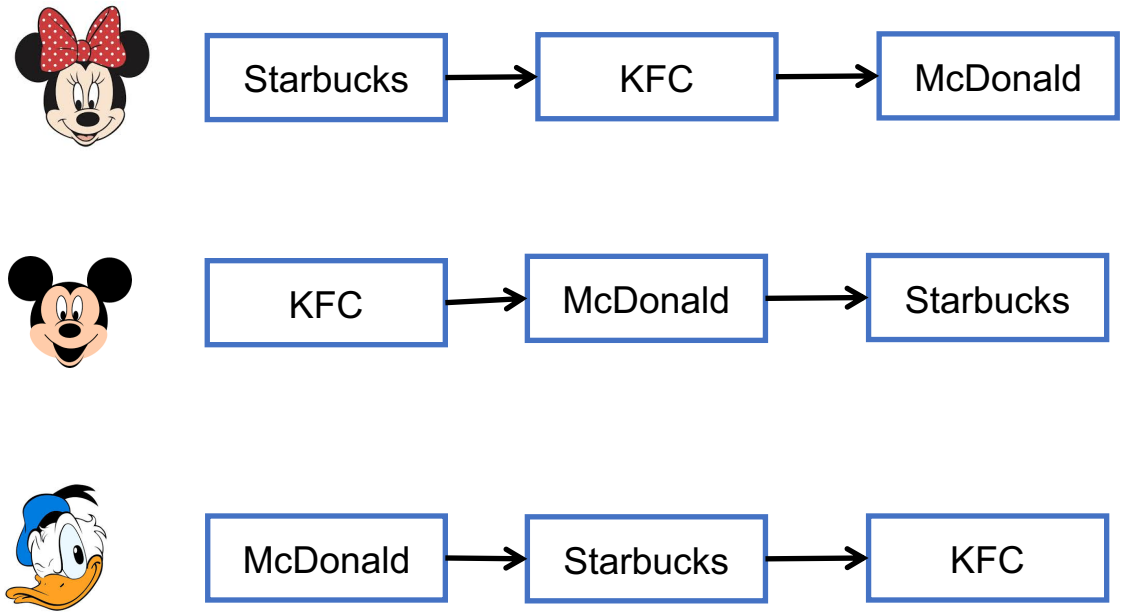
accept

reject

Method II – Accept multiple proposed values



Multiple chosen values → Consensus issue



If a proposal is chosen, all subsequent accepted proposals must have the same value.
→ Proposer should always propose the **safe value.**

Quorum: 2

propose

accept

reject

Basic idea of Paxos

Liveness

- Acceptors can accept more than one proposals

Consensus (Safety)

- Proposers only propose safe value
  - If a proposal with value p is chosen, then p is the safe value.
  - Otherwise, any value is the safe value.

Review The Example

proposers

acceptors

Propose Starbucks

Starbucks

Starbucks

Starbucks

time

Quorum: 2

propose

accept

reject

Review The Example

proposers

acceptors

Propose Starbucks

Starbucks is chosen

Starbucks

Starbucks

Starbucks

time

Quorum: 2

propose

accept

reject

Review The Example

proposers

acceptors

Propose Starbucks

Starbucks is chosen

Propose Starbucks

Starbucks → Starbucks

Starbucks → Starbucks

Starbucks → Starbucks

time

Quorum: 2

propose

accept

reject

Review The Example

proposers

acceptors

Propose Starbucks

Starbucks is chosen

Propose Starbucks

Propose Starbucks

Starbucks → Starbucks → Starbucks

Starbucks → Starbucks → Starbucks

Starbucks → Starbucks → Starbucks

time

Quorum: 2

propose

accept

reject

Challenges

Discover the safe value

Challenges

Discover the safe value

- Use an extra round of communication to find out the safe value

Discover the safe value with extra round trip

proposers

acceptors

Propose Starbucks

Starbucks is chosen

Starbucks

Starbucks

Starbucks

time

Quorum: 2

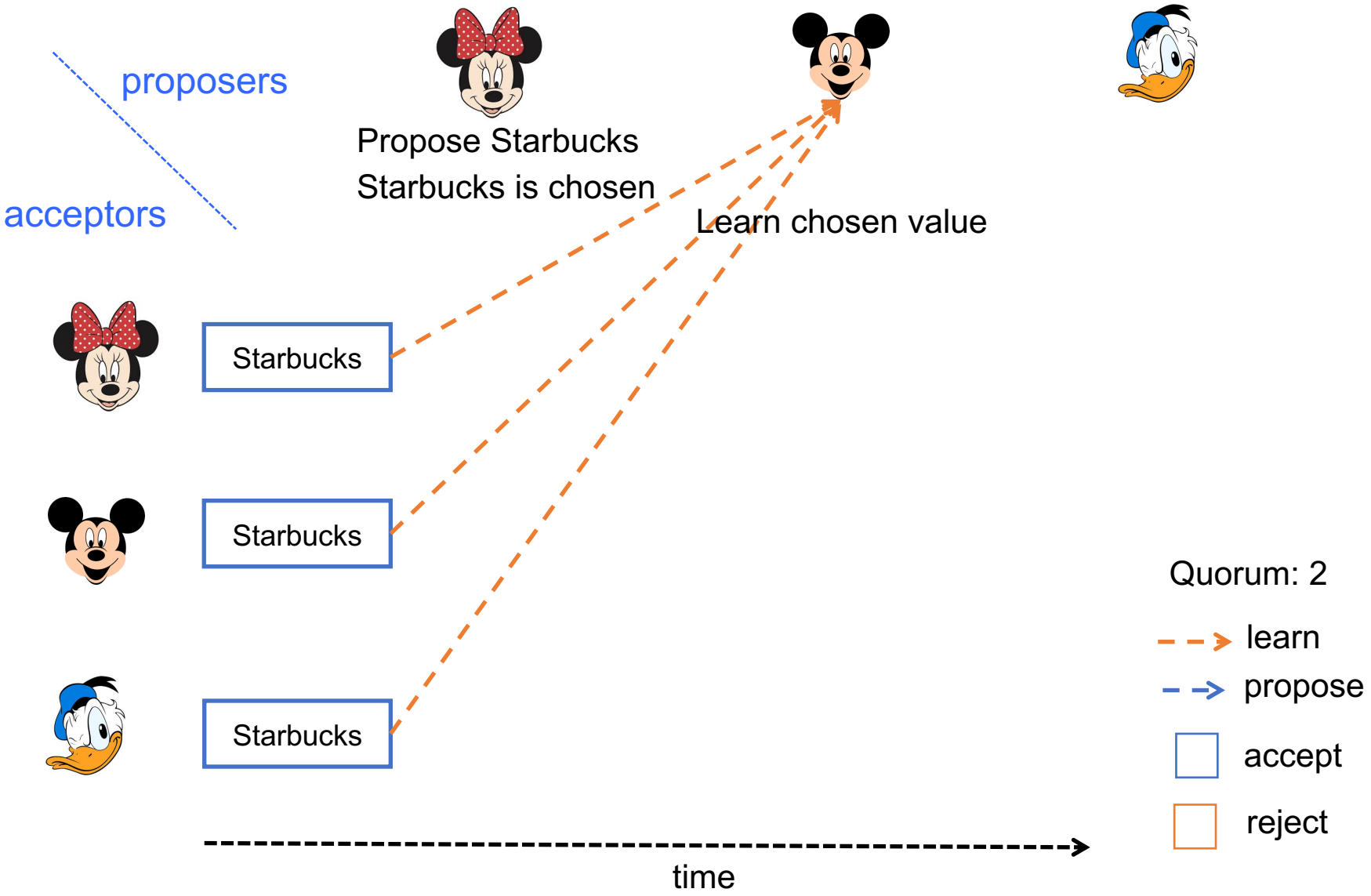
learn

propose

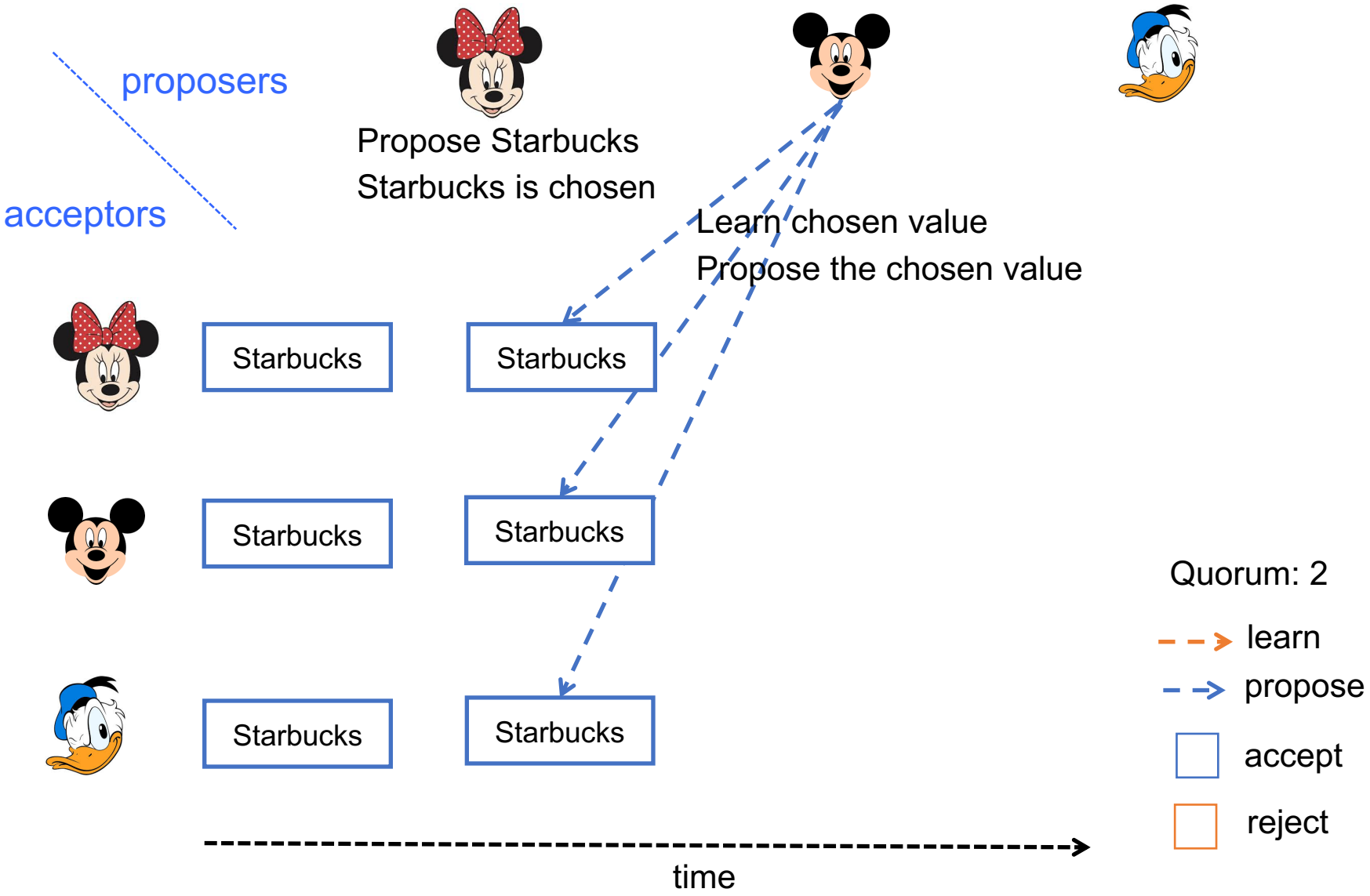
accept

reject

Discover the safe value with extra round trip



Discover the safe value with extra round trip



Challenges

Discover the safe value

- Use an extra round of communication to find out the safe value

Asynchronous network delays/reorders packets

Asynchronous network delays/reorders packets

proposers

acceptors

Propose Starbucks

Delayed at MK and Don

Starbucks

Quorum: 2

learn

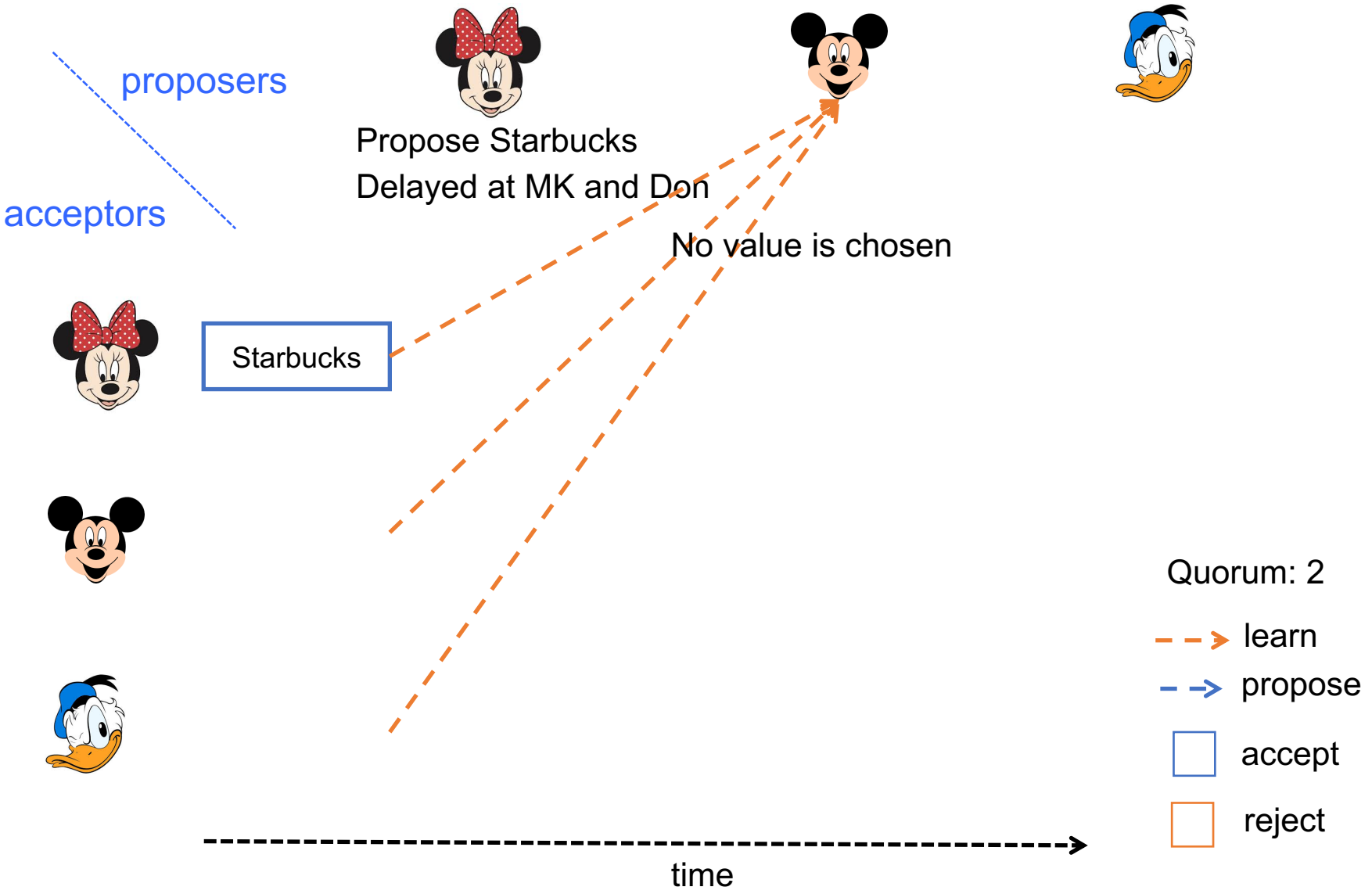
propose

accept

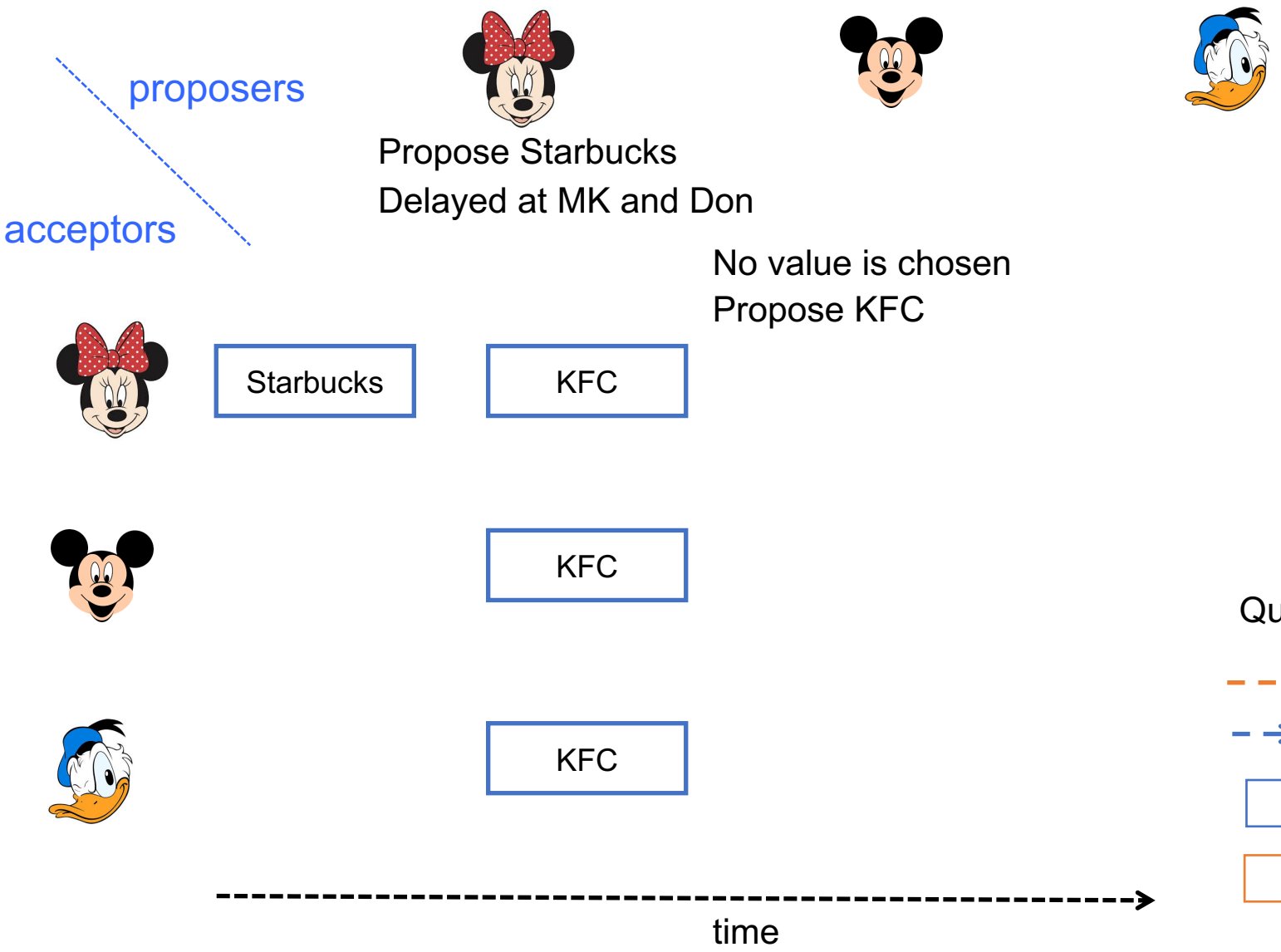
reject

time

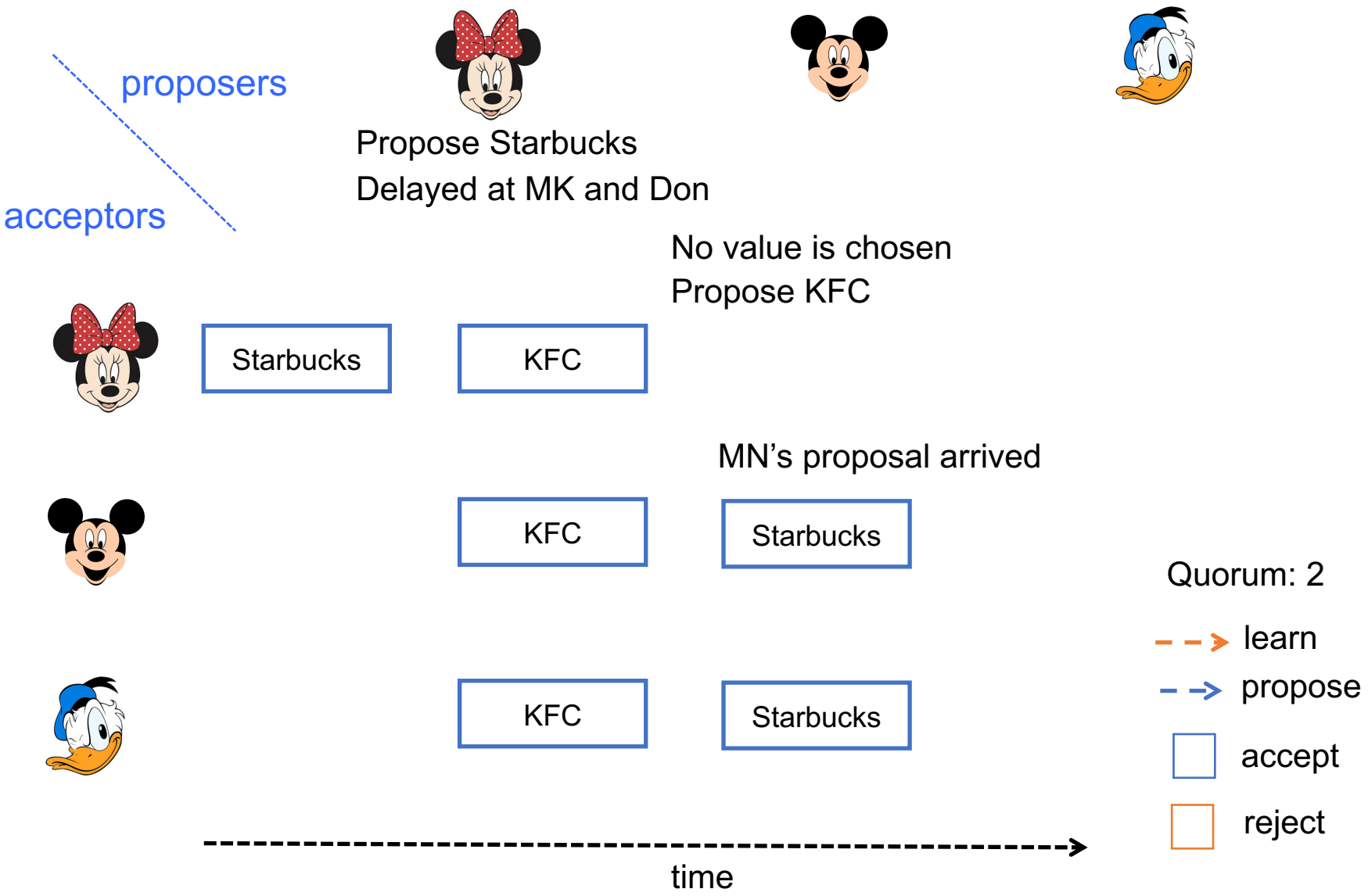
Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



Challenges

Discover the safe value

- Use an extra round of communication to find out the safe value

Asynchronous network delays/reorders packets

- Totally ordered proposals

Totally ordered proposals

Each proposal is identified by a globally unique number (proposal / ballot number)

- *cascade(*local counter, server id)

If proposal p is chosen, then for all p’ > p, p’.value = p.value

- Guarantee that if p is chosen, all subsequent accepted proposals must have the same values

Acceptor rejects proposal p’, if p’ < p and p is the latest/highest proposal have been seen

- Tolerate the asynchronous network

Basic Paxos – two phase protocol

Phase 1 – Prepare

- Proposer: pick a proposal number and discover the safe value from a Read Quorum
- Acceptor: record the highest proposal number have been seen

Phase 2 – Accept

- Proposer: send the safe value to all acceptors, commit if a Write Quorum of nodes say yes
- Acceptor: accept if the proposal number is equal or higher than the remembered one

Asynchronous network delays/reorders packets

ballot 1 2 3

proposers

Propose Starbucks

Delayed at MK and Don

highest ballot number has been seen

acceptors

h_n

1

Starbucks

0

0

time

RQ = WQ = 2

learn

propose

accept

reject

Asynchronous network delays/reorders packets

ballot 1 2 3

proposers

Propose Starbucks
Delayed at MK and Don

highest ballot number has been seen

acceptors

No value is chosen

h_n

2

Starbucks

2

2

time

RQ = WQ = 2

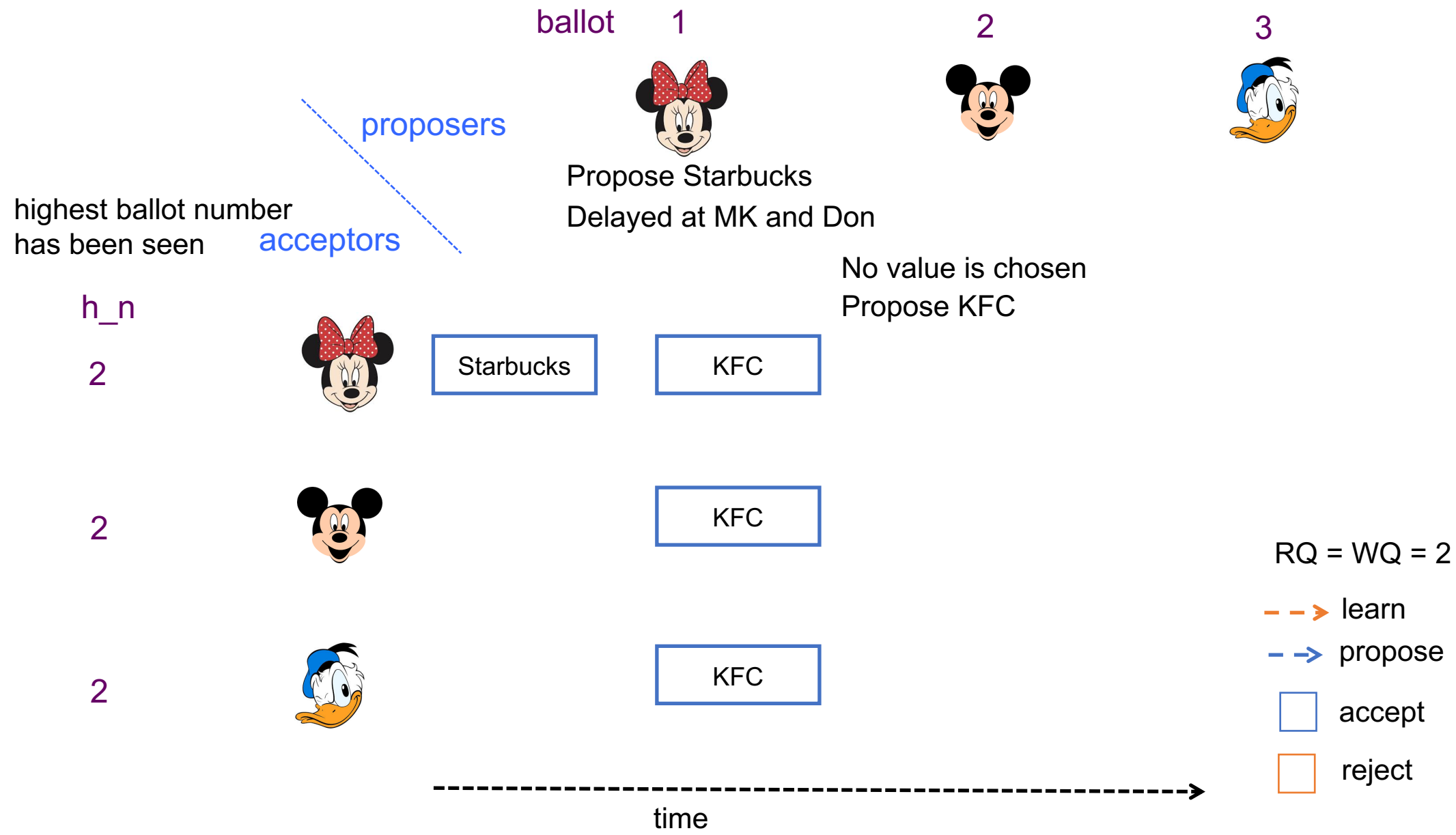
learn

propose

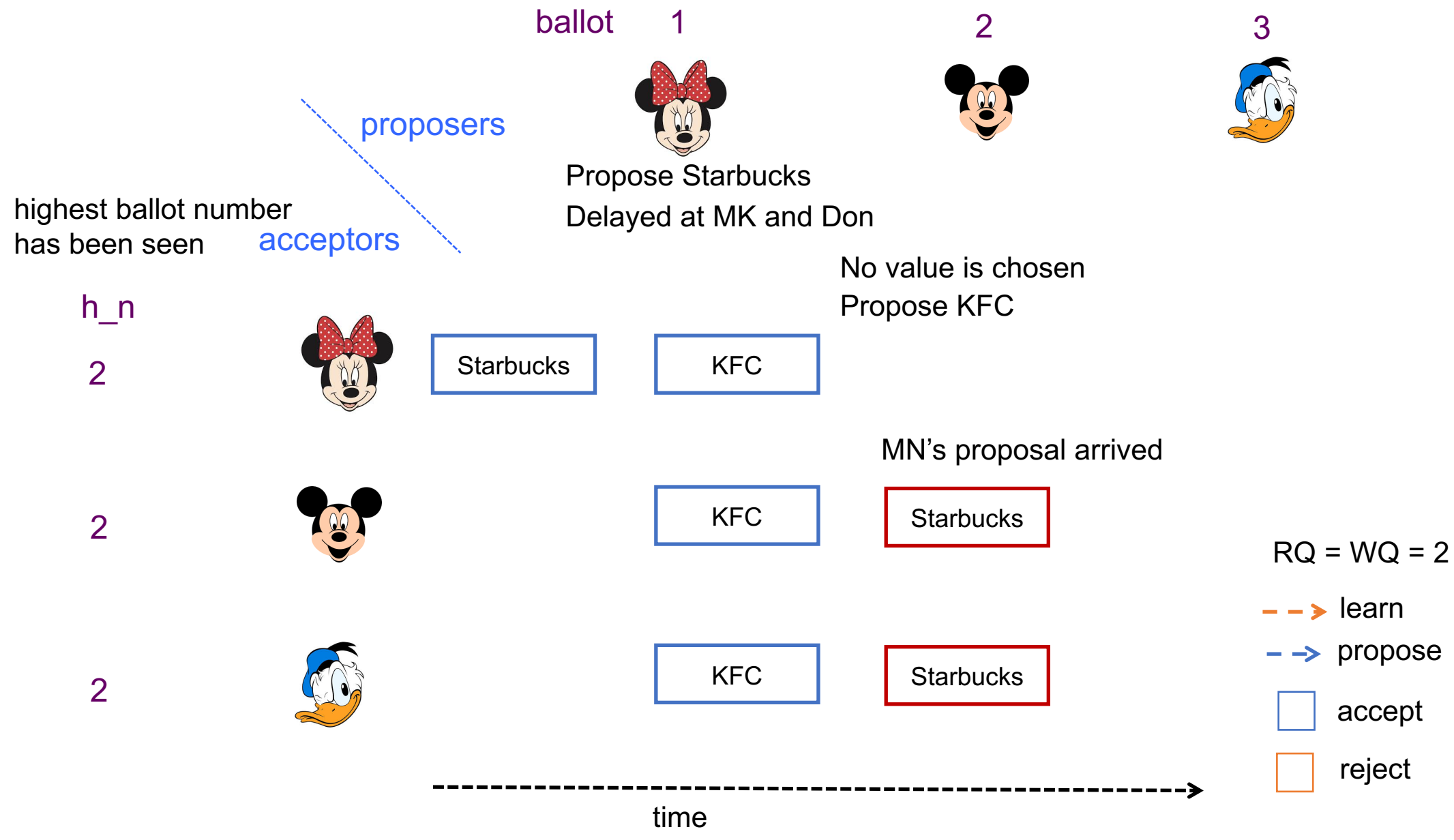
accept

reject

Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets

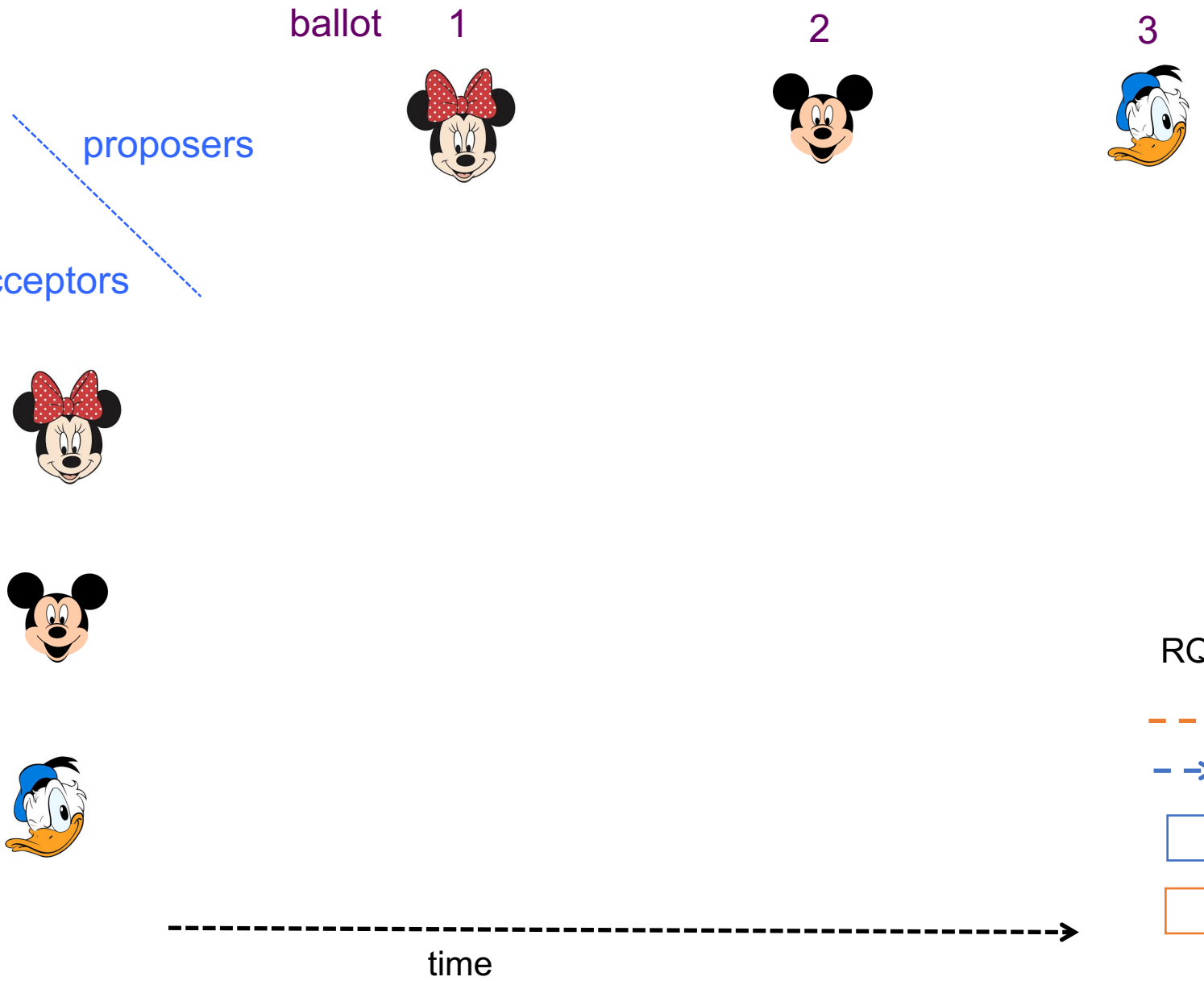


Find a safe value with a read quorum

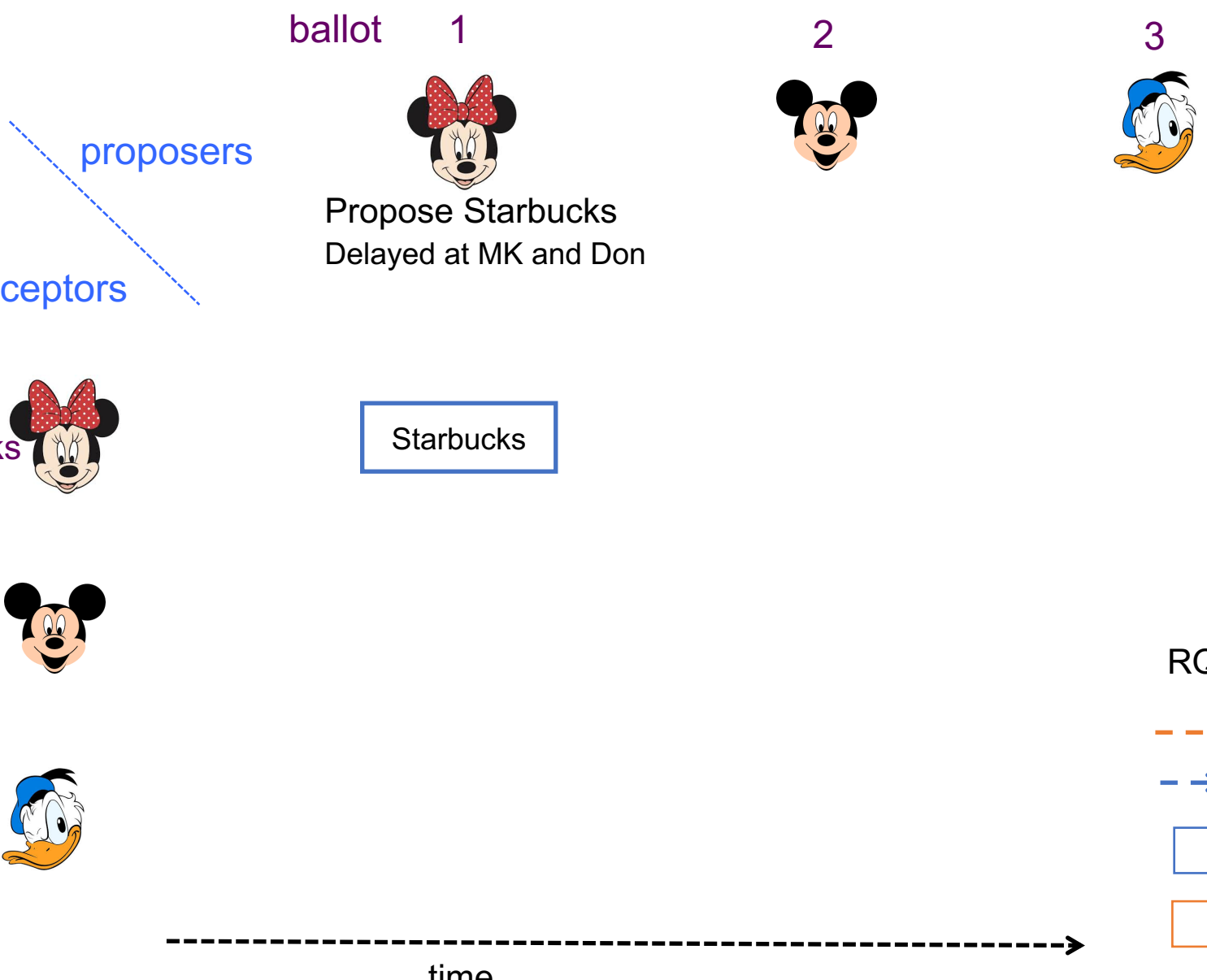
A value is safe if

- it has highest accepted ballot number among a read quorum
- the intersection of read quorum and write quorum is not empty
  - E.g., RQ = WQ = majority

Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



h_n: highest ballot number has been seen

a_n: highest ballot number has been accepted

a_v: value accepted at a_n

RQ = WQ = 2

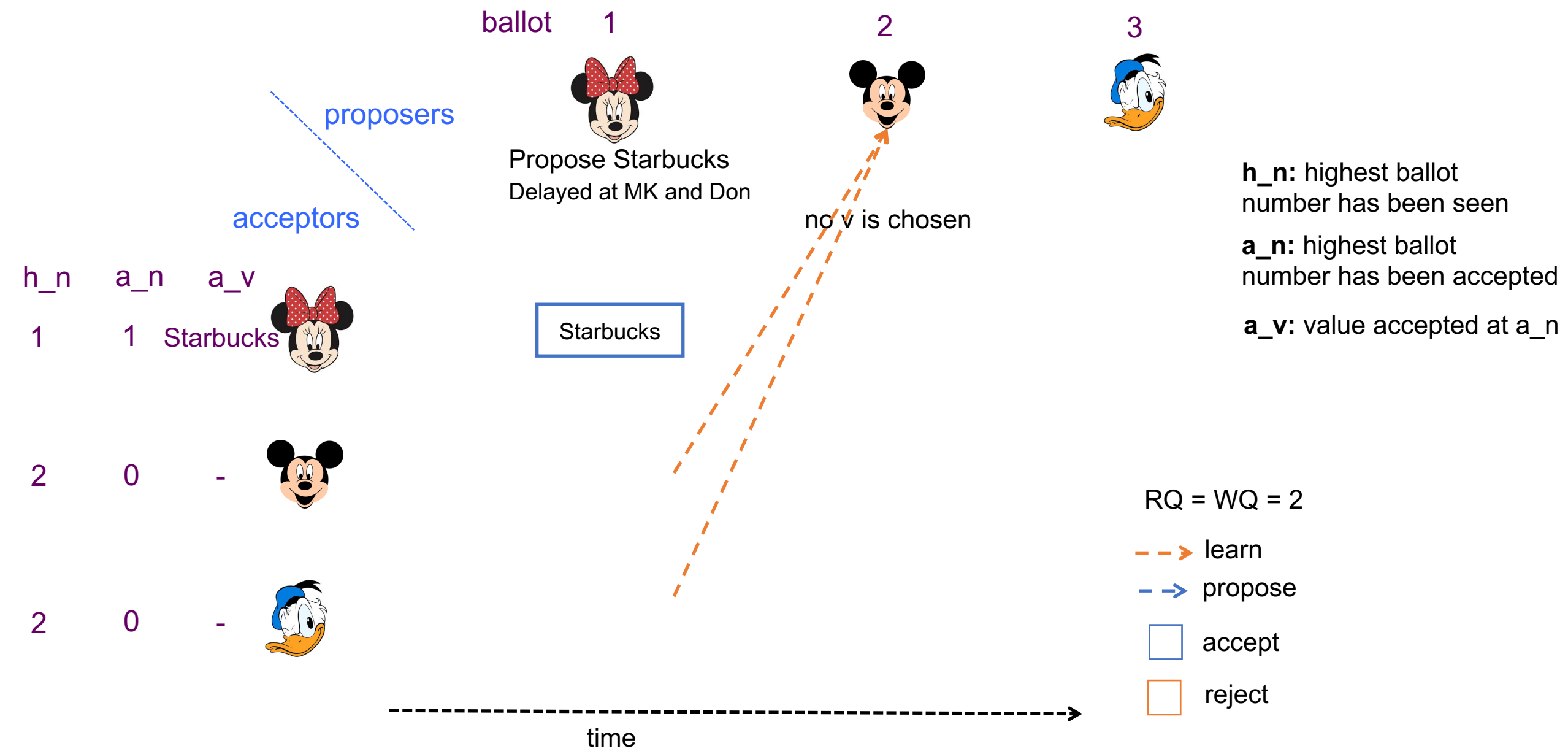
learn

propose

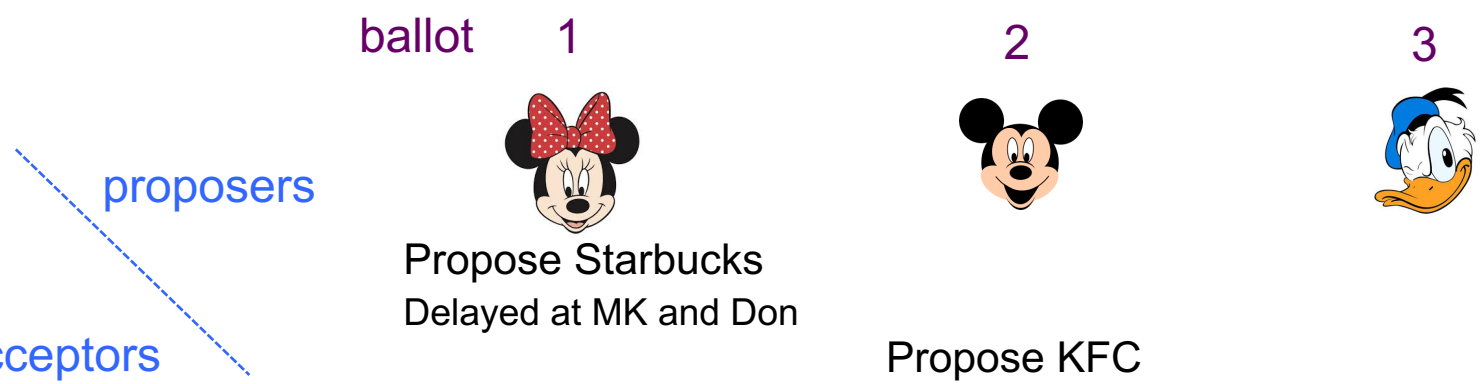
accept

reject

Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



h_n: highest ballot number has been seen

a_n: highest ballot number has been accepted

a_v: value accepted at a_n

RQ = WQ = 2

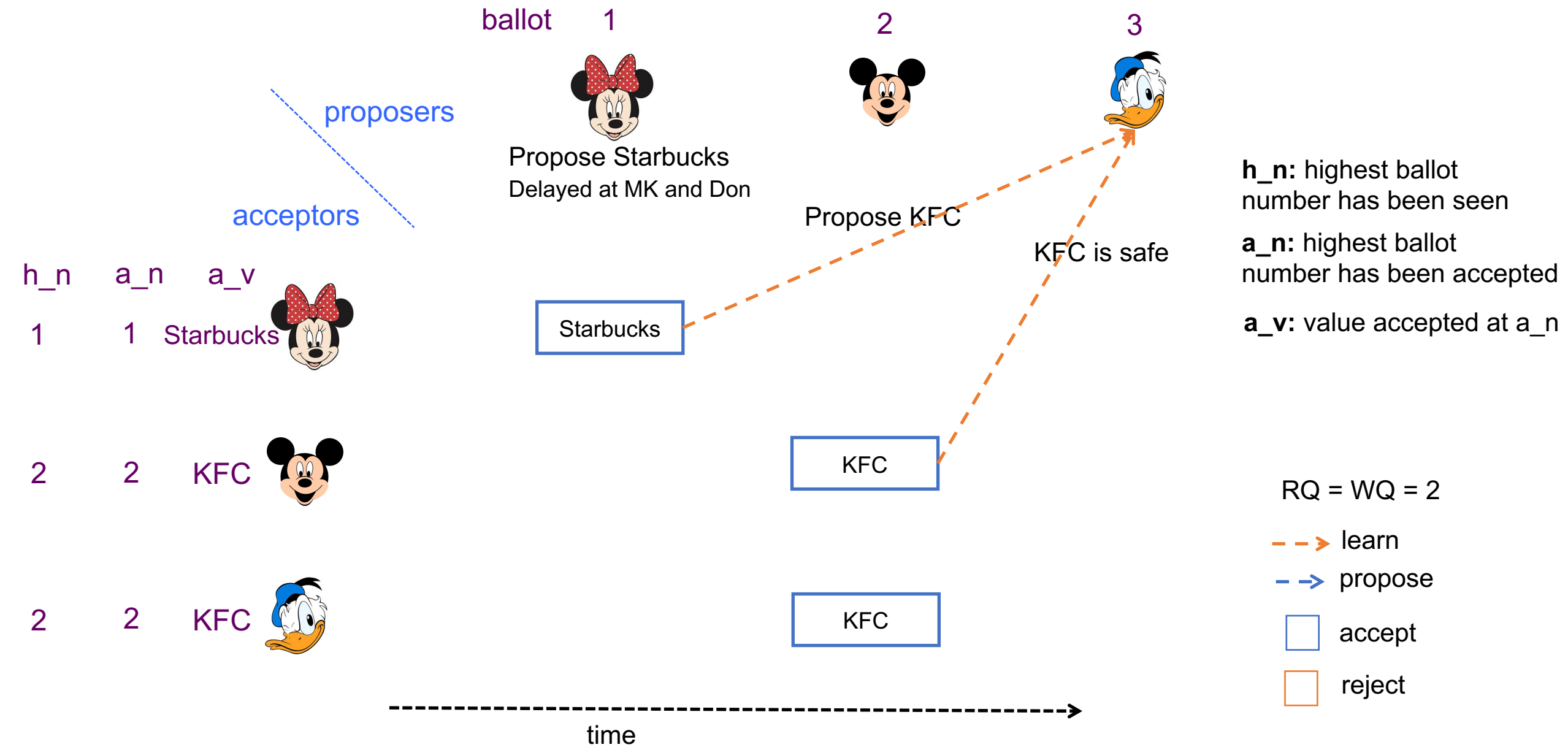
learn

propose

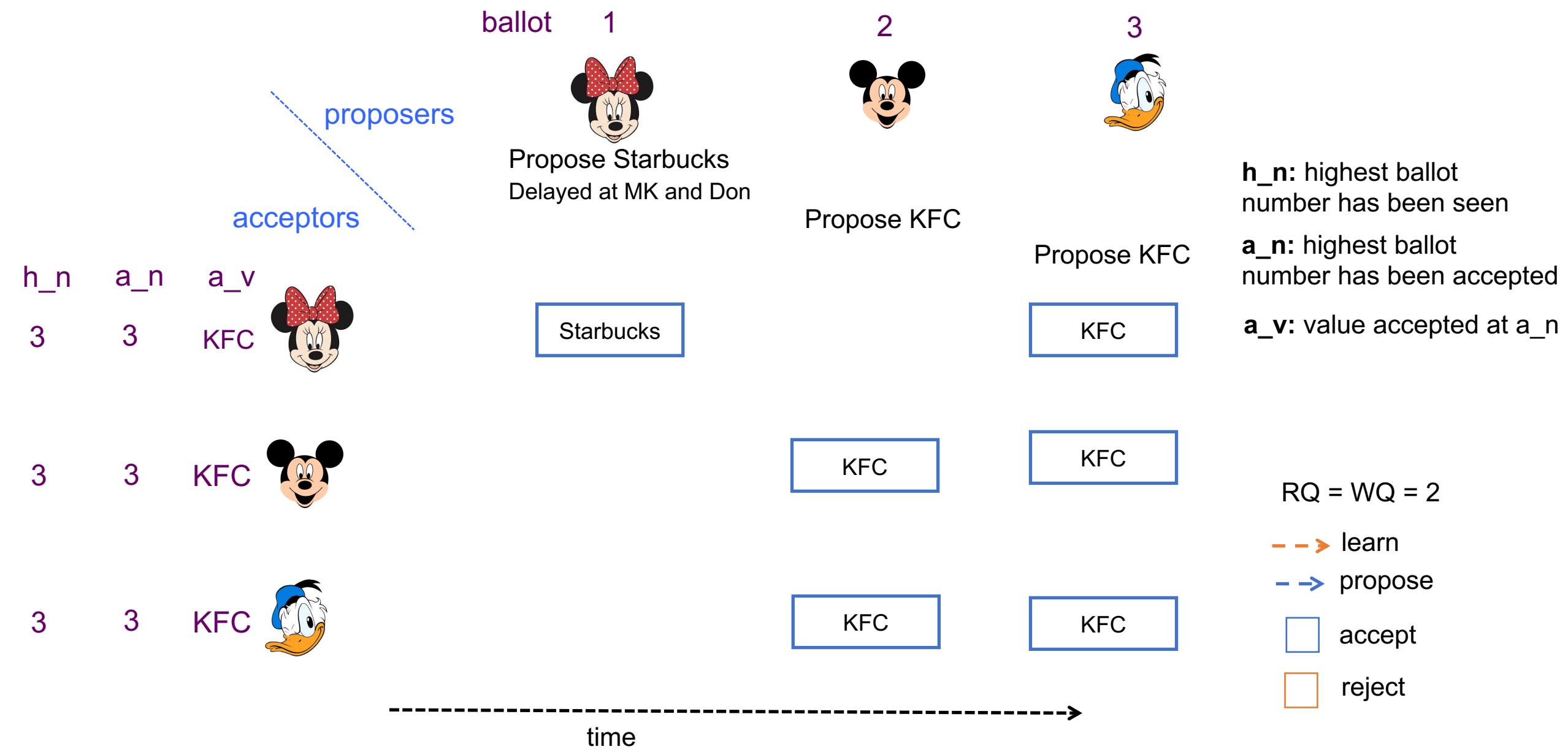
accept

reject

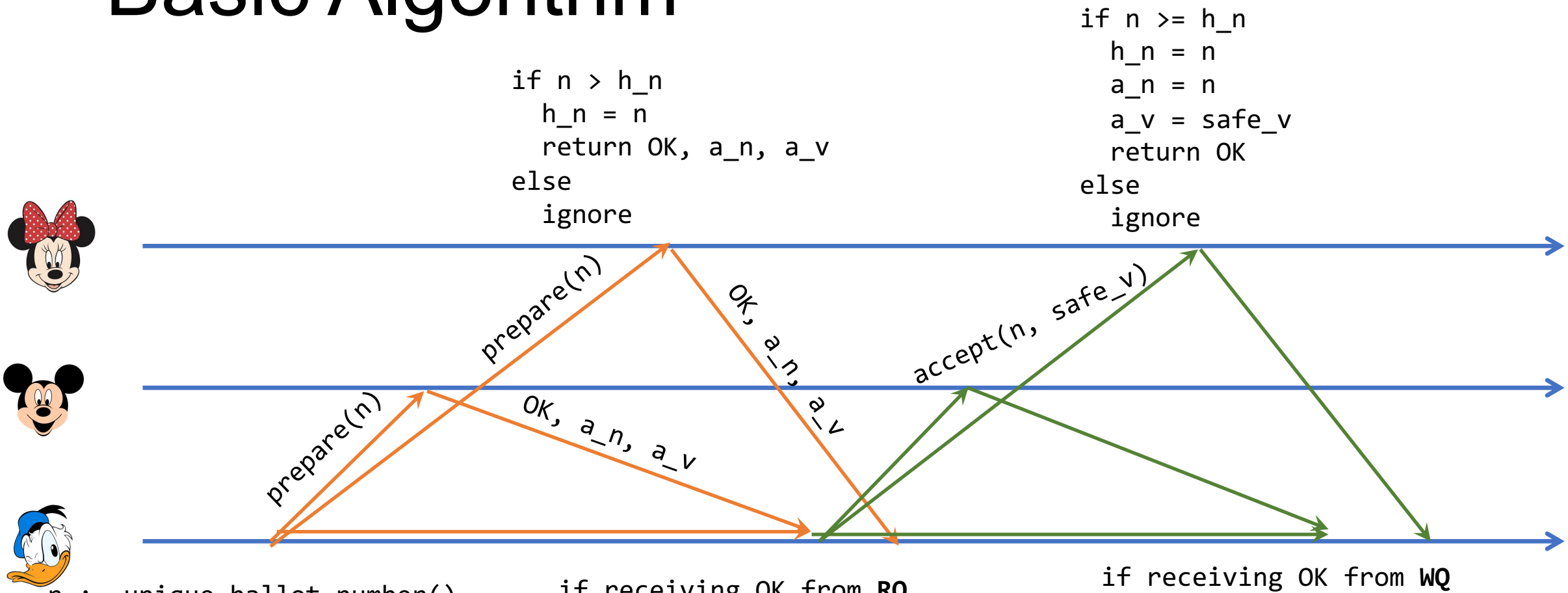
Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



Basic Algorithm

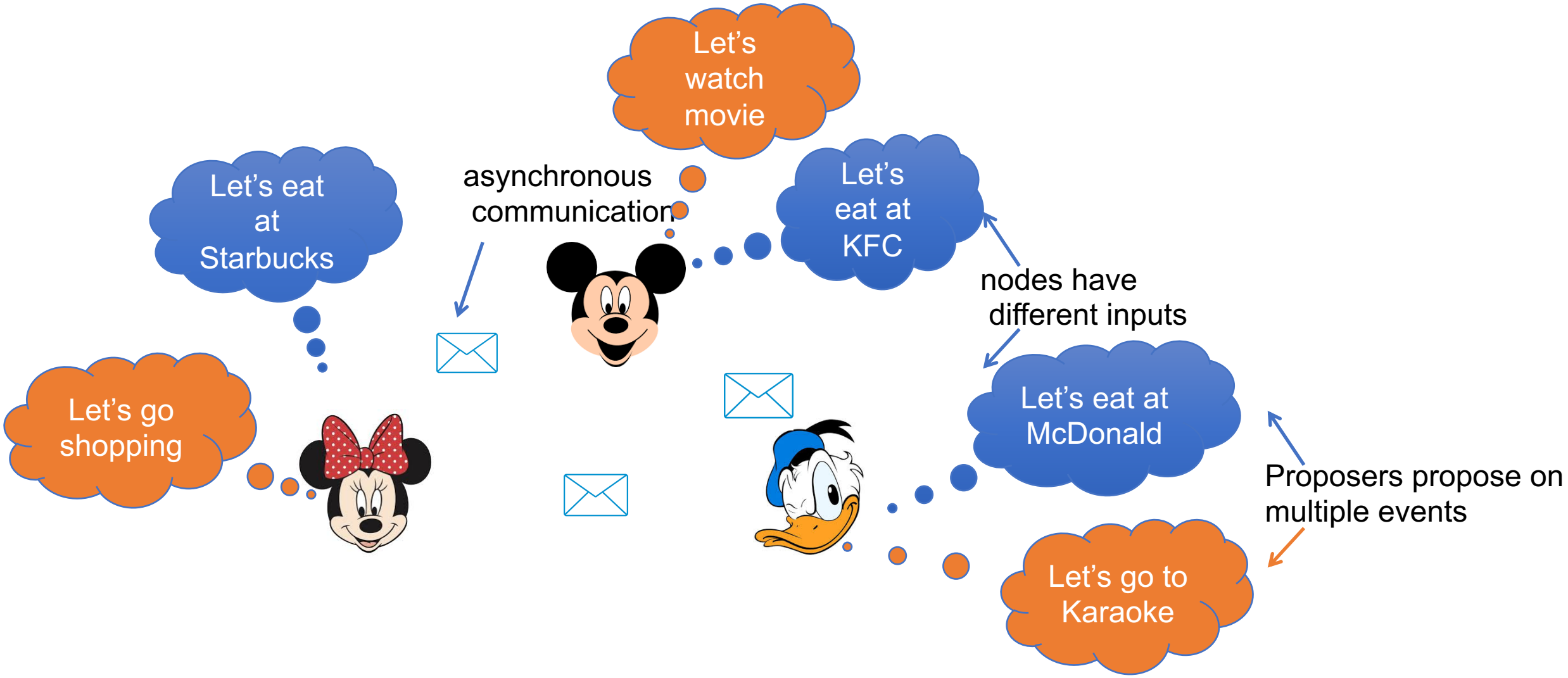


h_n – highest ballot number has been seen

a_n – highest ballot have been accepted

a_v – value accepted at a_n

Multi-Decree Consensus: from Paxos to MultiPaxos



Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

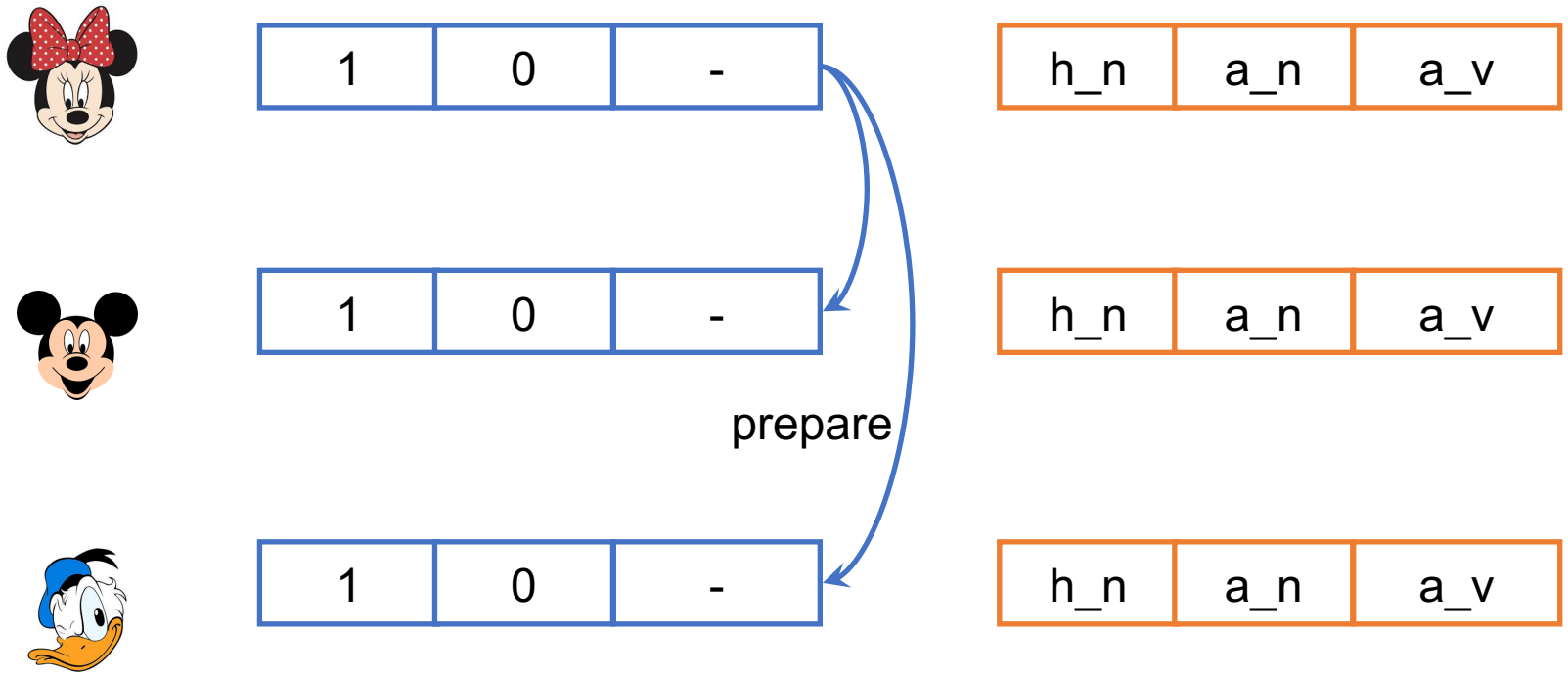
h_n	a_n	a_v
h_n	a_n	a_v
h_n	a_n	a_v

Single-decree Paxos

h_n	a_n	a_v
h_n	a_n	a_v
h_n	a_n	a_v

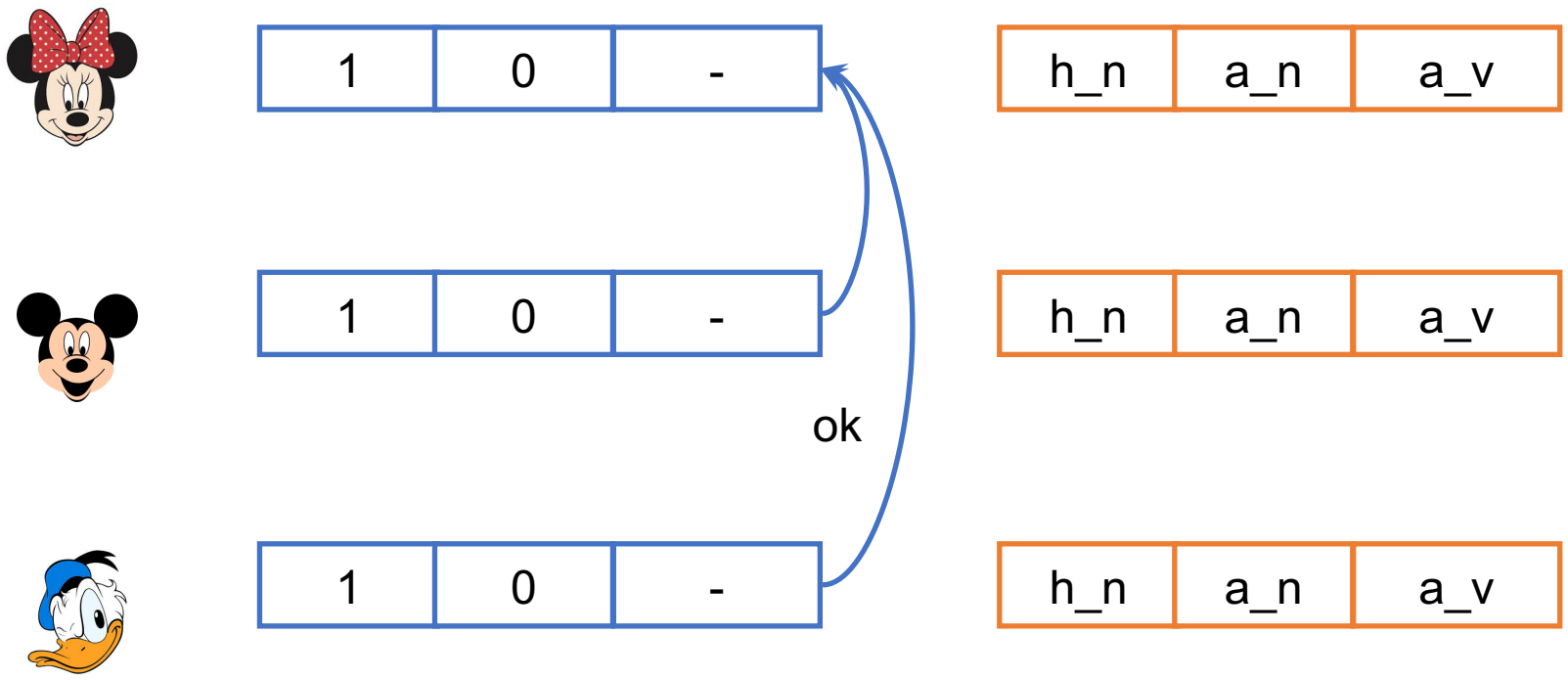
Single-decree Paxos

Multi-Decree Consensus: from Paxos to MultiPaxos



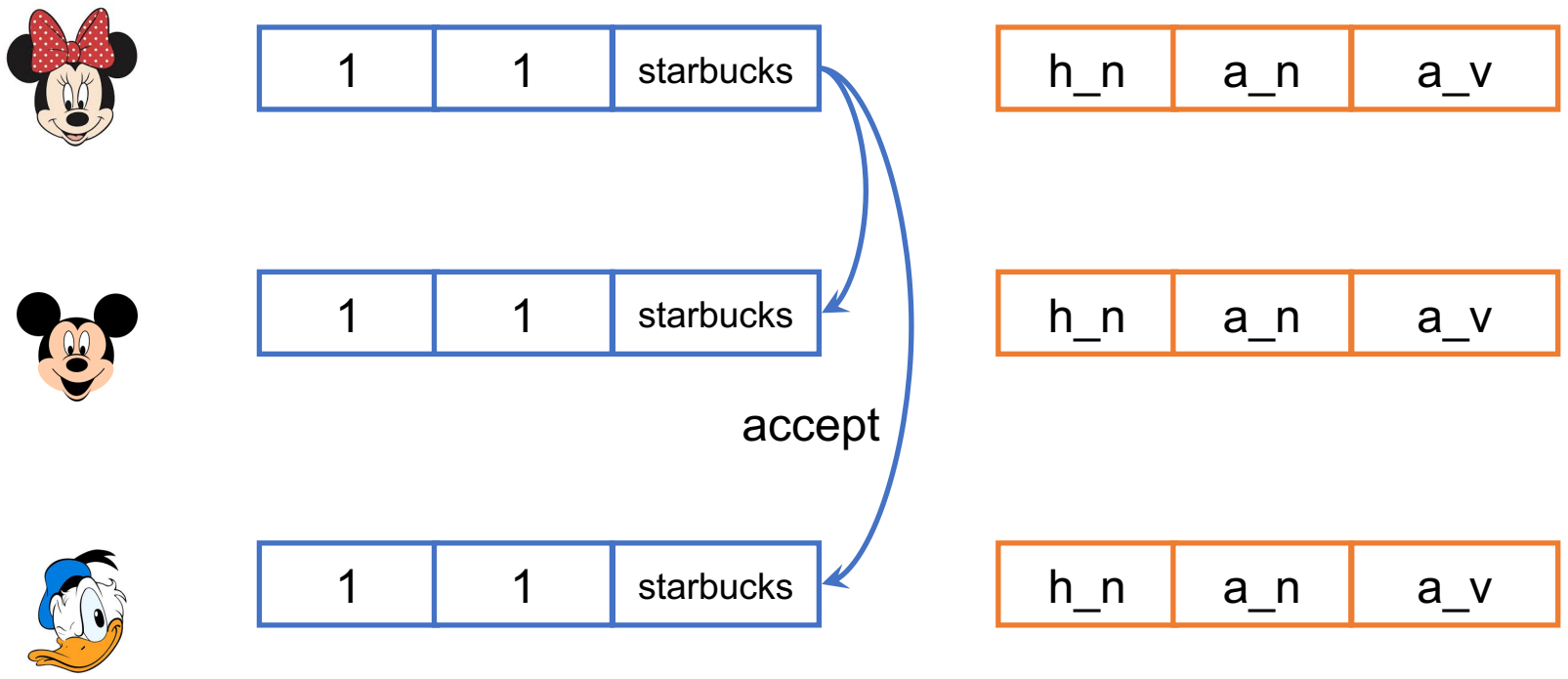
1. Use Paxos to achieve consensus on **Lunch**

Multi-Decree Consensus: from Paxos to MultiPaxos



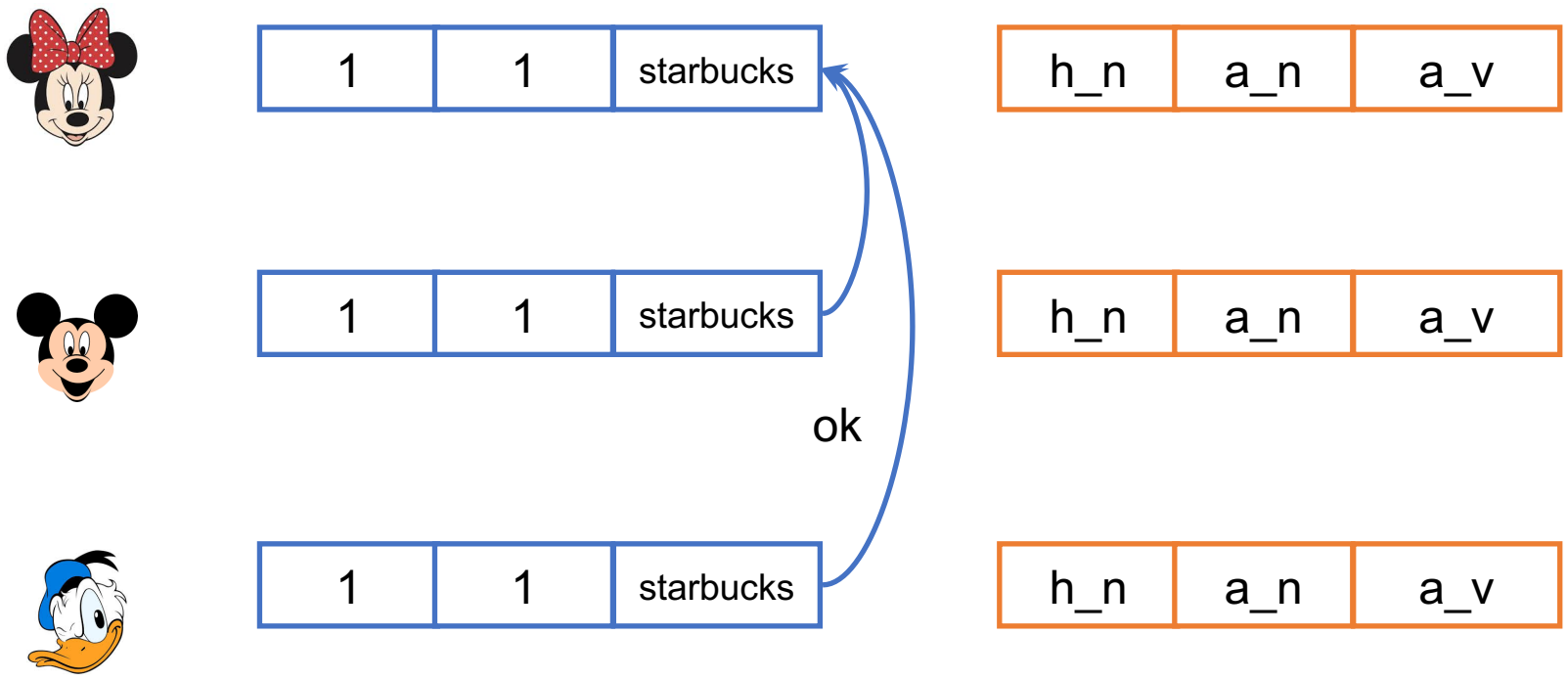
1. Use Paxos to achieve consensus on **Lunch**

Multi-Decree Consensus: from Paxos to MultiPaxos



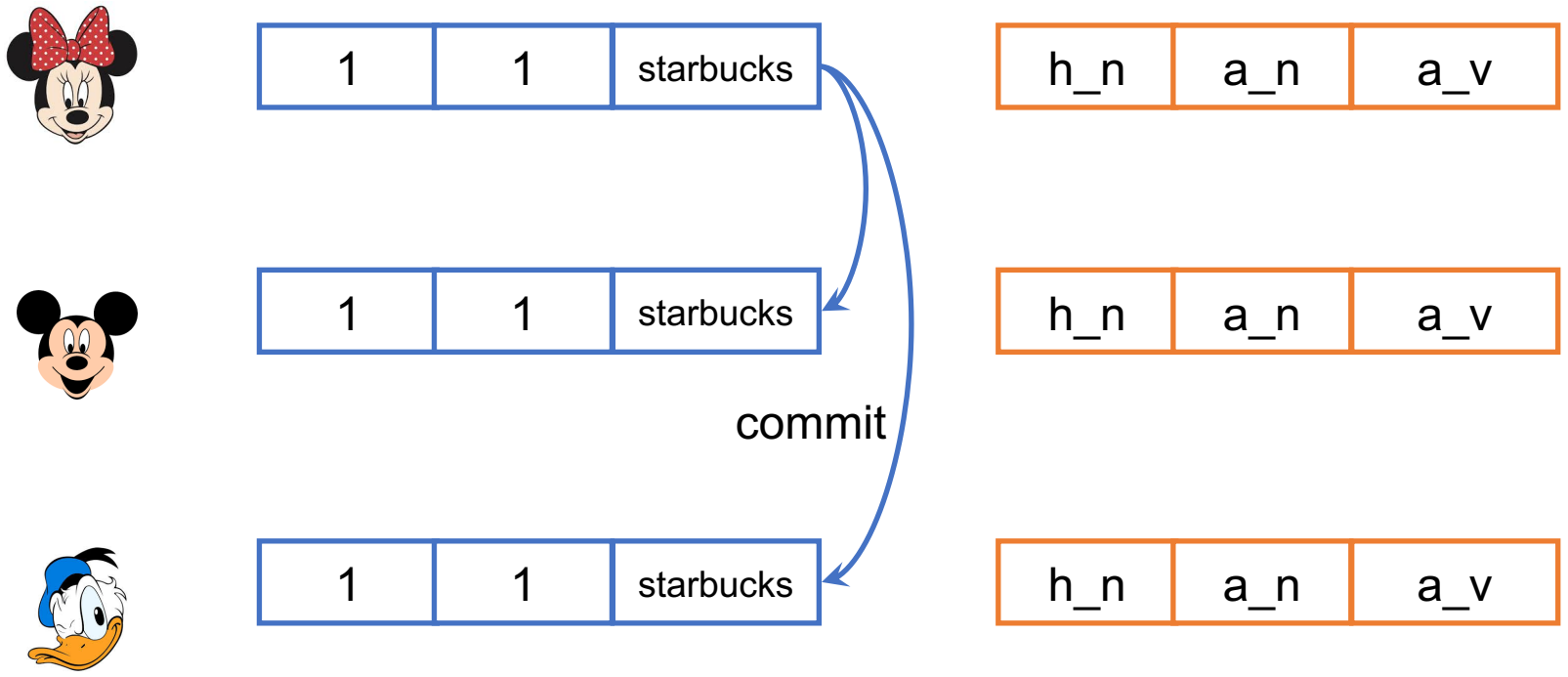
1. Use Paxos to achieve consensus on **Lunch**

Multi-Decree Consensus: from Paxos to MultiPaxos



1. Use Paxos to achieve consensus on **Lunch**

Multi-Decree Consensus: from Paxos to MultiPaxos



1. Use Paxos to achieve consensus on **Lunch**

Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

2. Use Paxos to achieve consensus on **Entertainment**

1	1	**starbucks**

1	1	**movie**

1	1	**starbucks**

1	1	**movie**

1	1	**starbucks**

1	1	**movie**

Single-decree Paxos

Single-decree Paxos

Multi-Decree Consensus: from Paxos to MultiPaxos

Events	Lunch	Entertainment
	1 | 1 | **starbucks**	1 | 1 | **movie**
	1 | 1 | **starbucks**	1 | 1 | **movie**
	1 | 1 | **starbucks**	1 | 1 | **movie**
	Single-decree Paxos	Single-decree Paxos

3. Each node first has lunch then watches movie.

Multi-Decree Consensus: from Paxos to MultiPaxos

Log

0th entry

1	1	**starbucks**

1	1	**starbucks**

1	1	**starbucks**

Single-decree Paxos

1st entry

1	1	**movie**

1	1	**movie**

1	1	**movie**

Single-decree Paxos

Replicated State Machine

- Replicating a log of operations consistently

The naïve approach –

- Run a Paxos instance to agree on the value for each log entry
- Each Paxos instance has its own state: h_n, a_n, a_v
- The i-th op is able to be executed only if i-th Paxos instance has chosen a value, i-1-th op has been executed

Multi-Decree Consensus: from Paxos to MultiPaxos

Log

0th entry

1st entry

h_n	a_n	a_v
h_n	a_n	a_v
h_n	a_n	a_v

Single-decree Paxos

h_n	a_n	a_v
h_n	a_n	a_v
h_n	a_n	a_v

Single-decree Paxos

Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

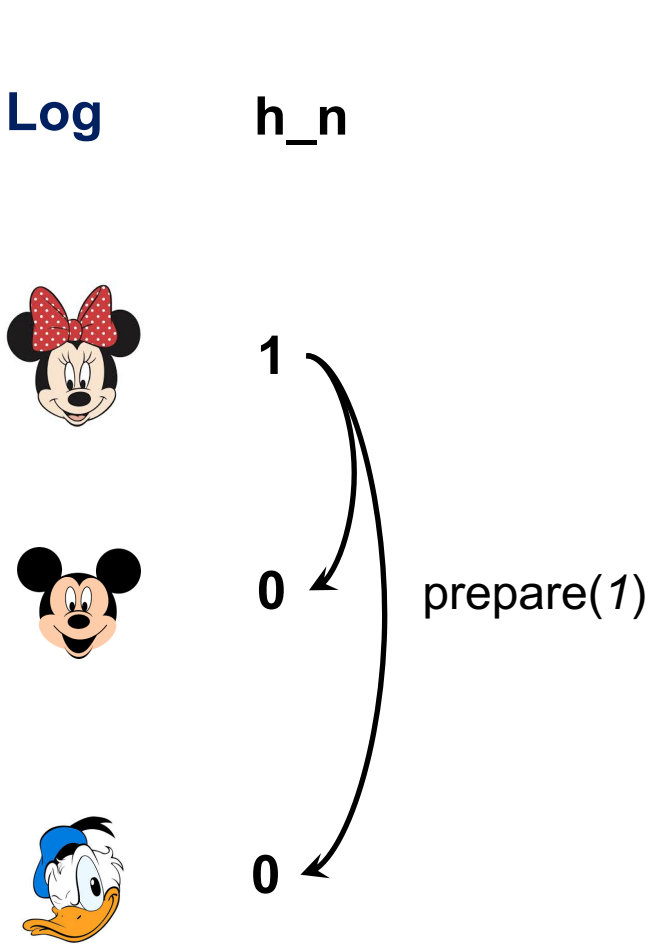
Multi-Decree Consensus: from Paxos to MultiPaxos

Log	h_n	0th entry		1st entry	
	**0**	a_n	a_v	a_n	a_v
	**0**	a_n	a_v	a_n	a_v
	**0**	a_n	a_v	a_n	a_v

Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Multi-Decree Consensus: from Paxos to MultiPaxos

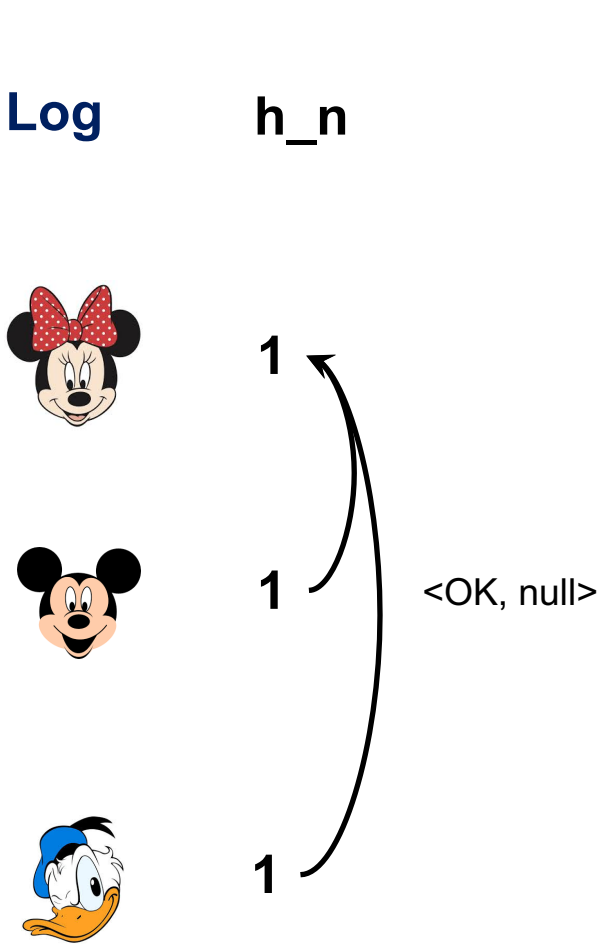


Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Multi-Decree Consensus: from Paxos to MultiPaxos

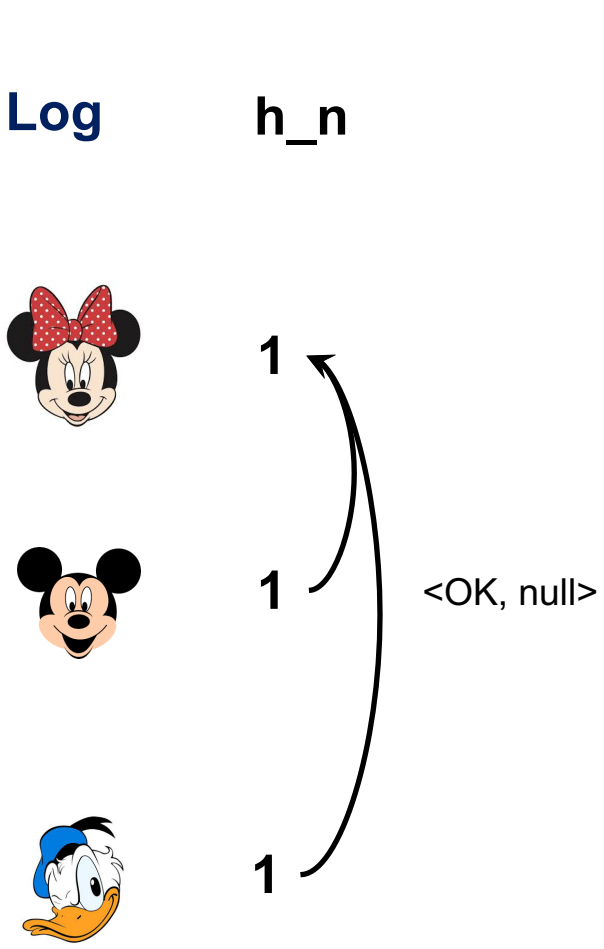


Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Step 1.2. each acceptor returns its non-empty log entries.

Multi-Decree Consensus: from Paxos to MultiPaxos



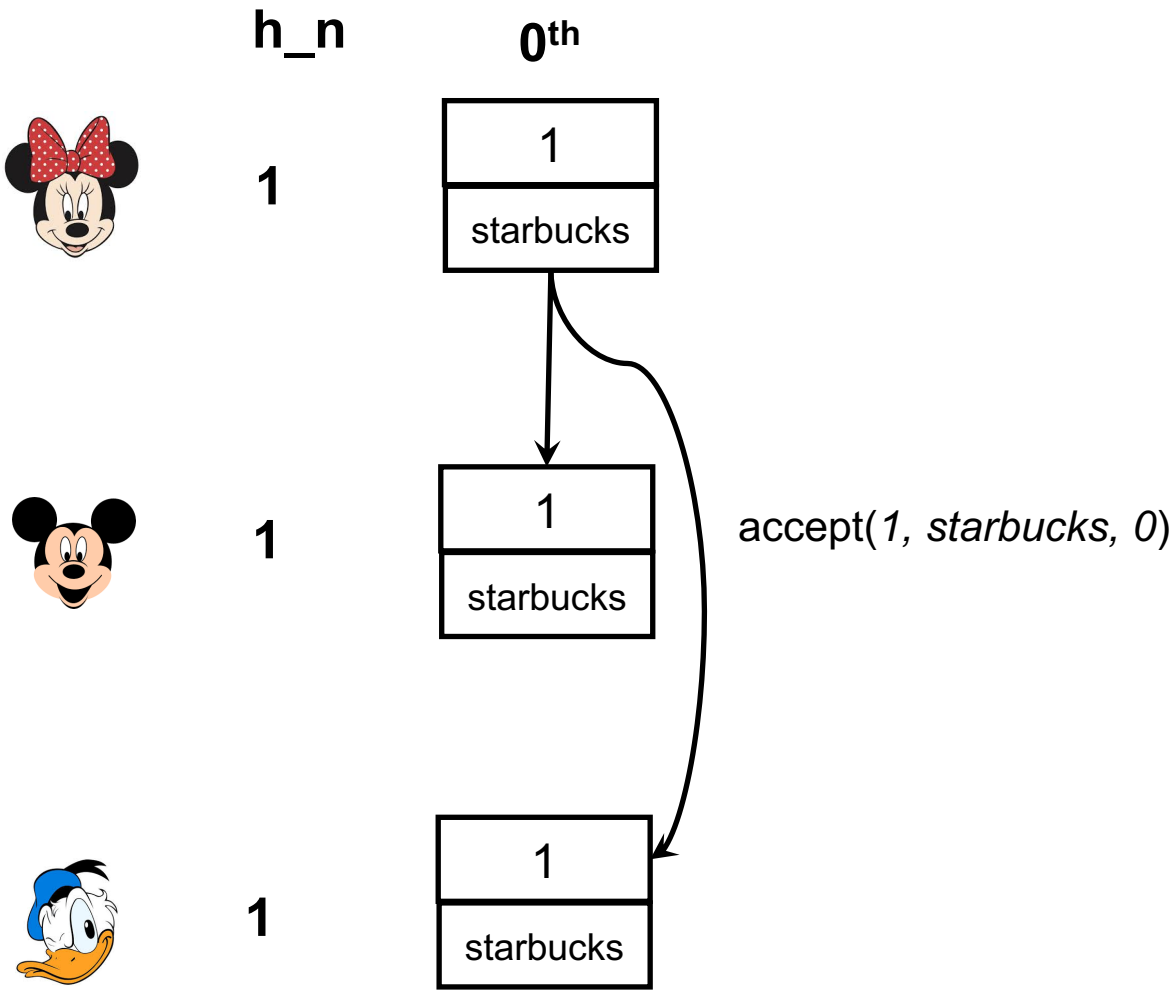
Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Step 1.3. the distinguished leader updates its log with the replied logs.

Multi-Decree Consensus: from Paxos to MultiPaxos

Log



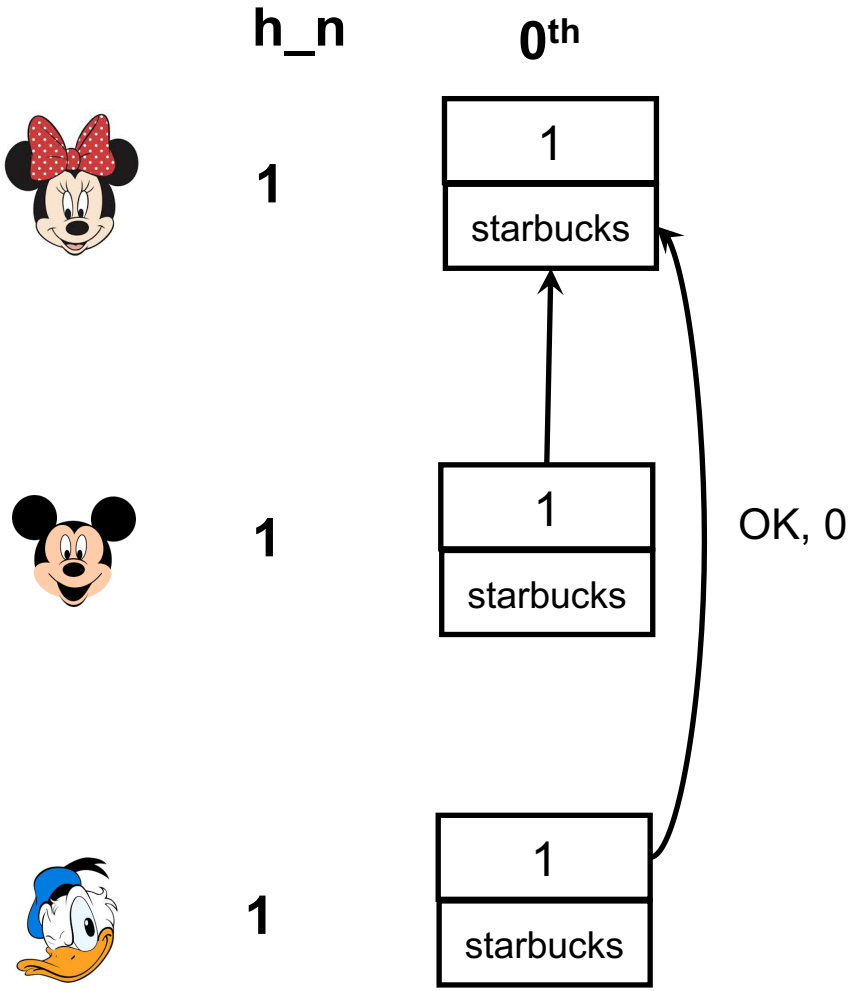
Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Multi-Decree Consensus: from Paxos to MultiPaxos

Log

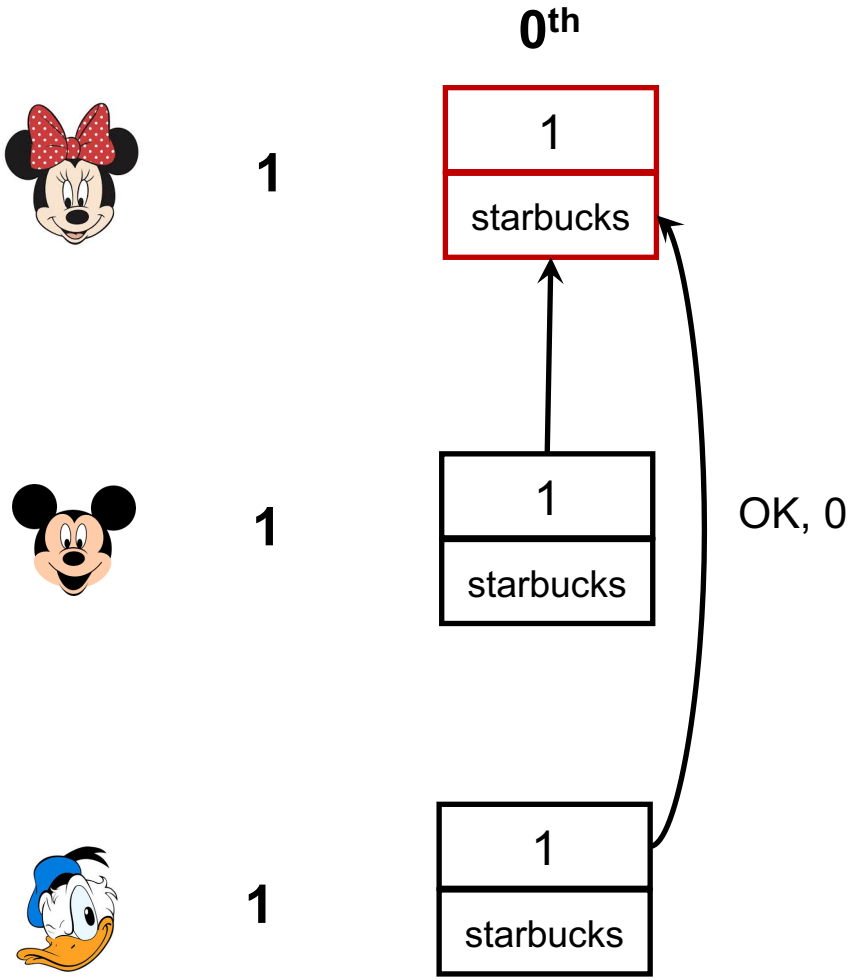


Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Step 2.2. each acceptor replies OK

Multi-Decree Consensus: from Paxos to MultiPaxos



Observation: if we ask one server to be the default/distinguished proposer, all instances share the same h_n.

batch prepare

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos

Log **h_n**

0th

1

1
starbucks

1

1
starbucks

1

1
starbucks

Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos

Log **h_n**

1

	0th	**1st**
	1	1
	starbucks	Shopping

1

1
starbucks

1

1
starbucks

Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

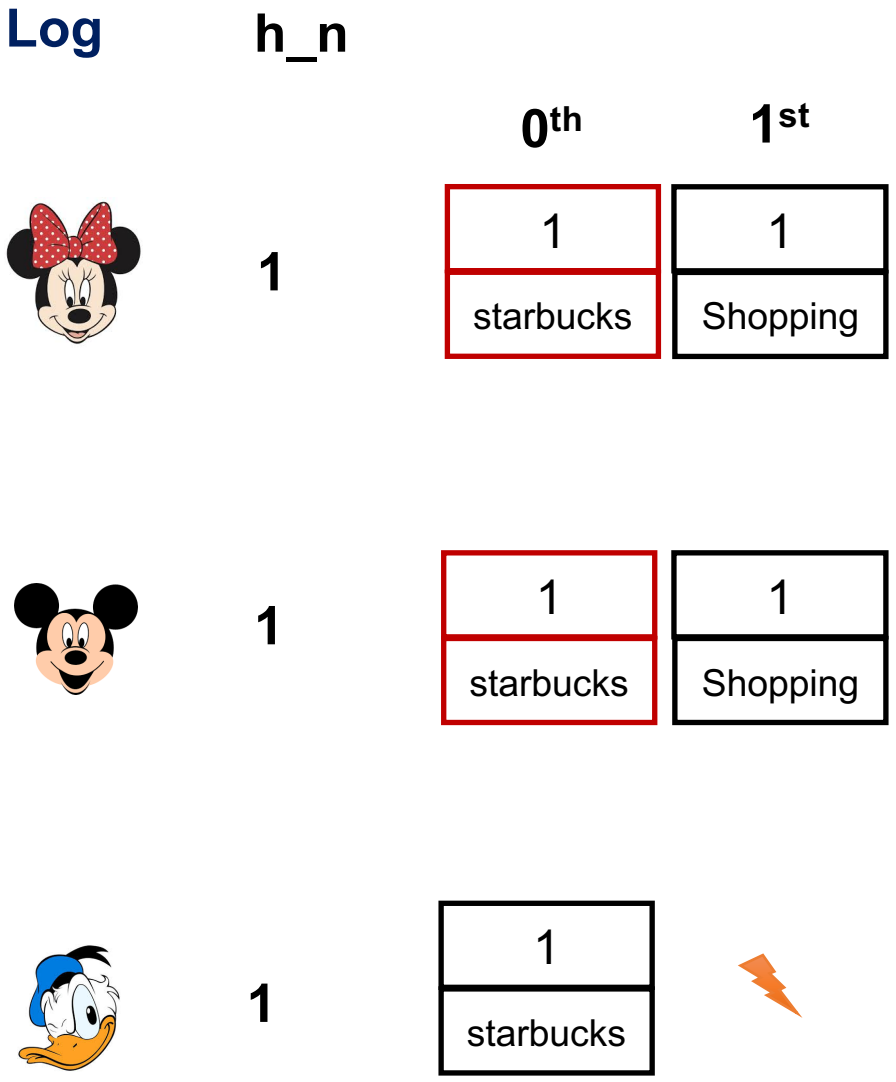
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

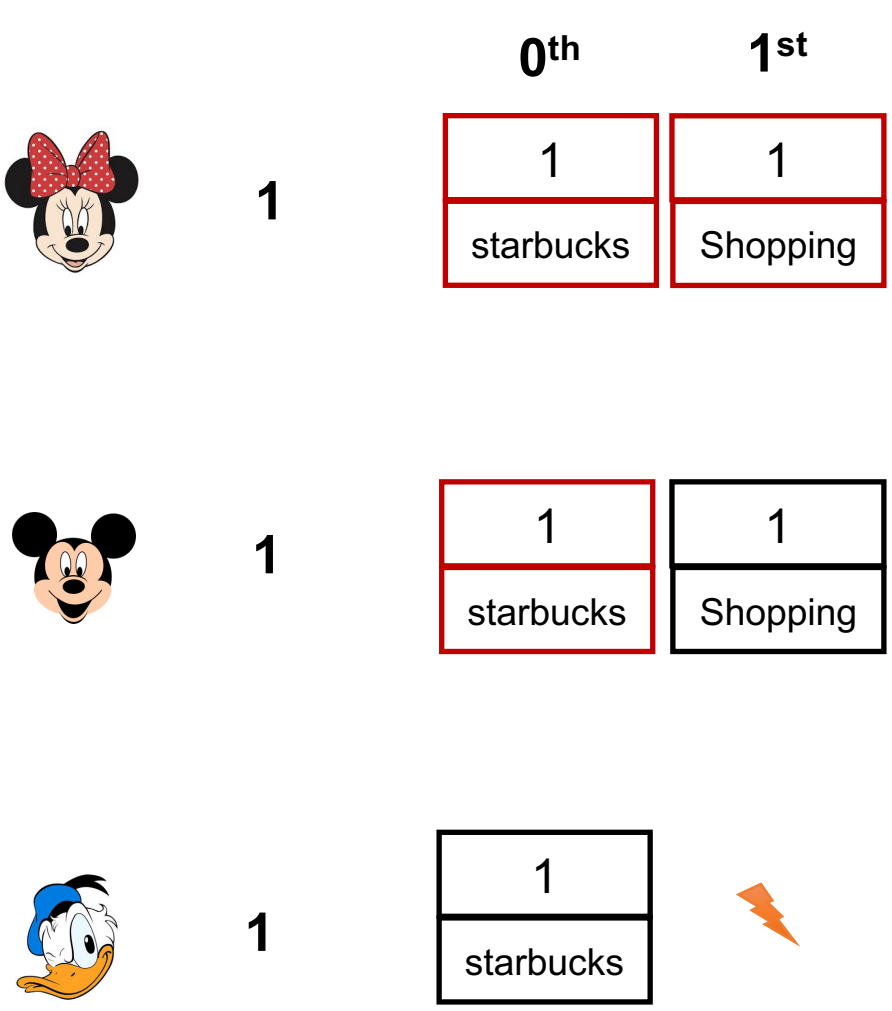
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

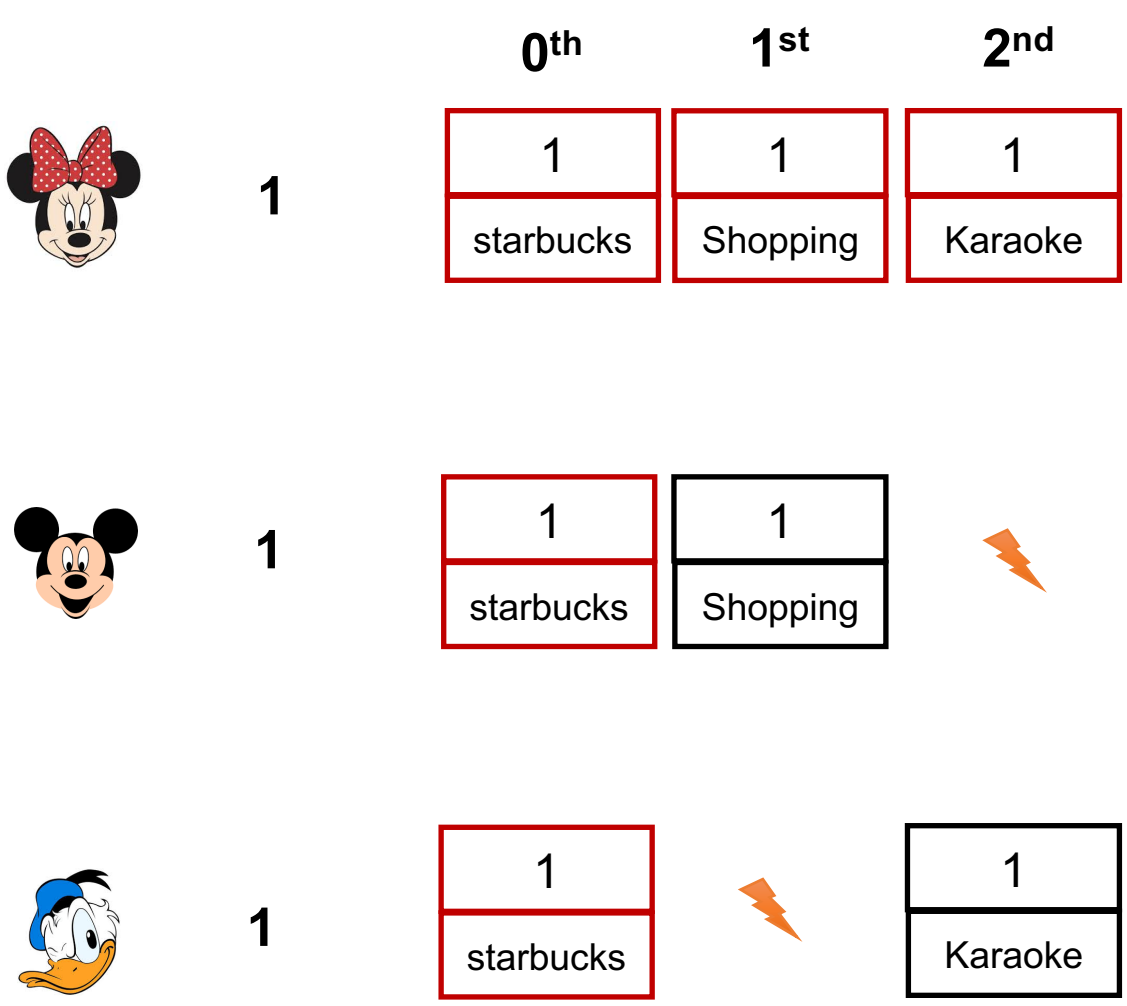
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

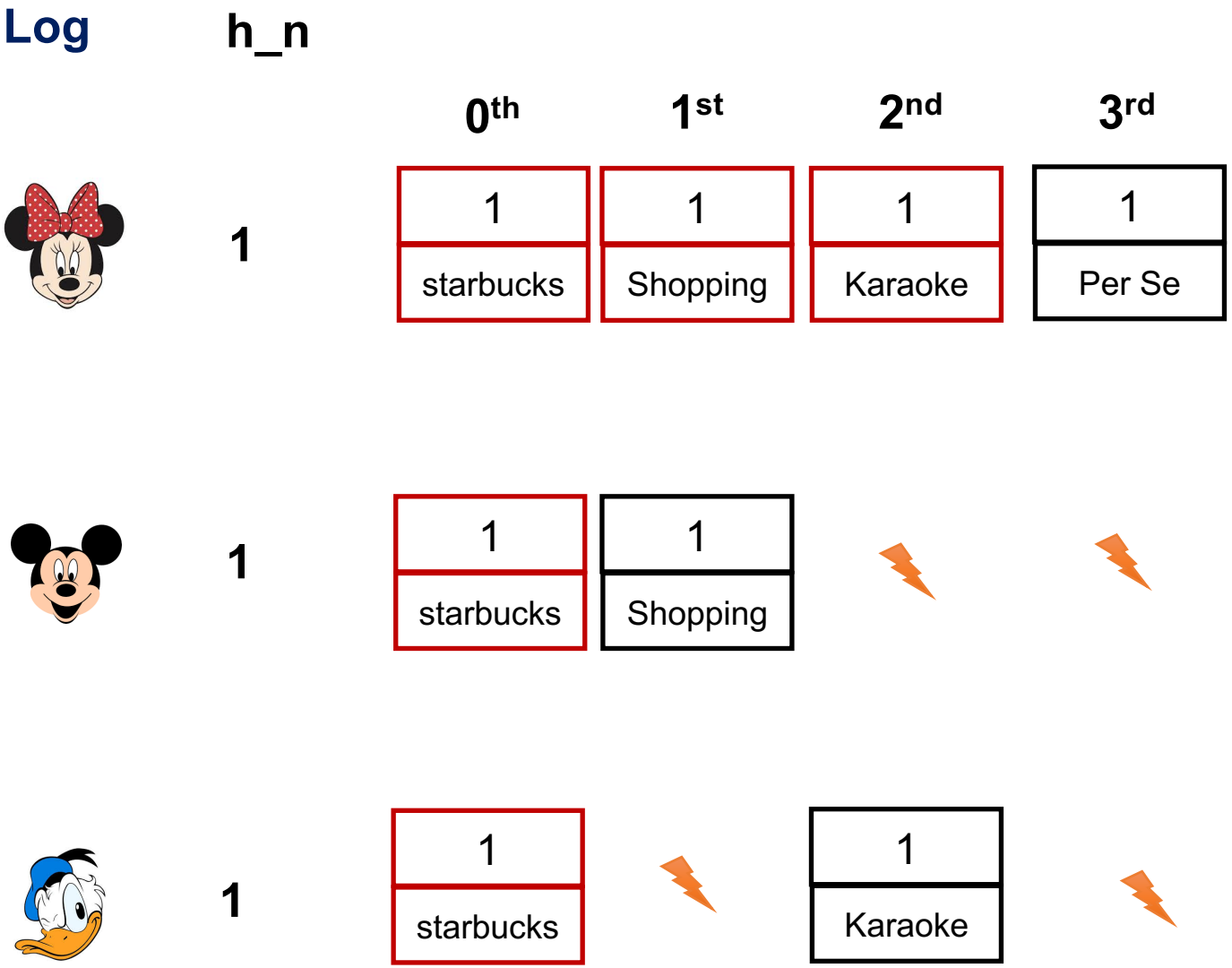
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

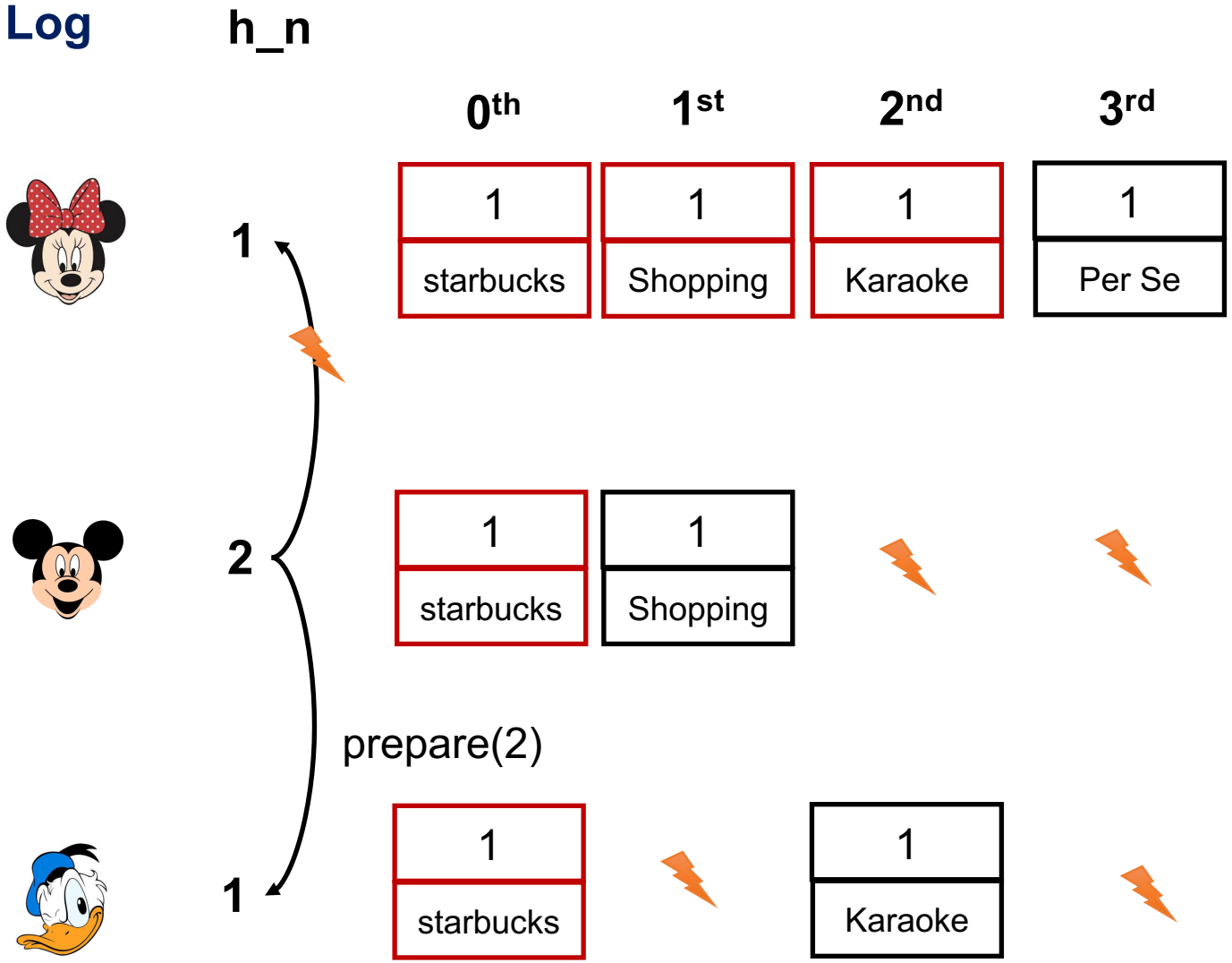
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

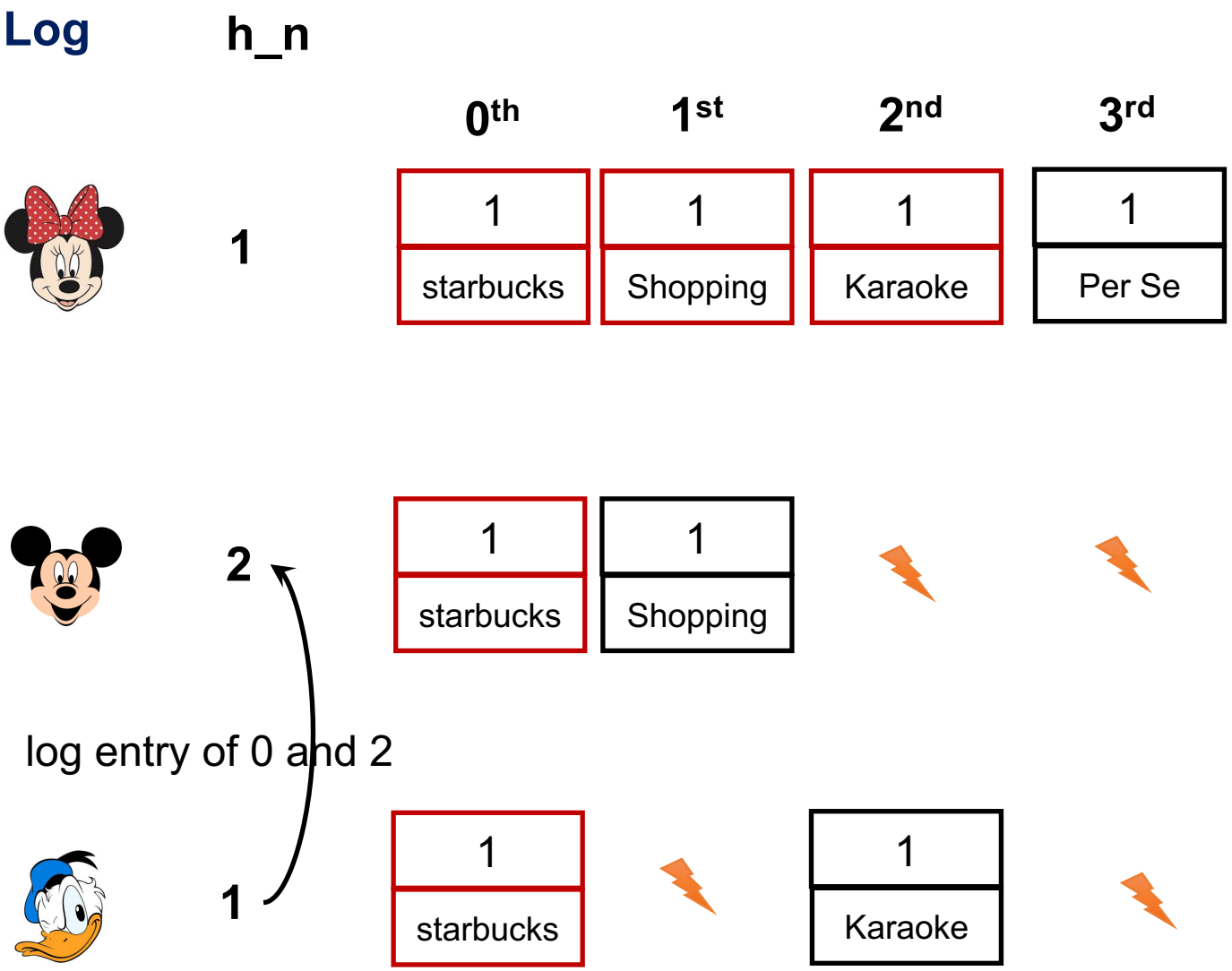
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

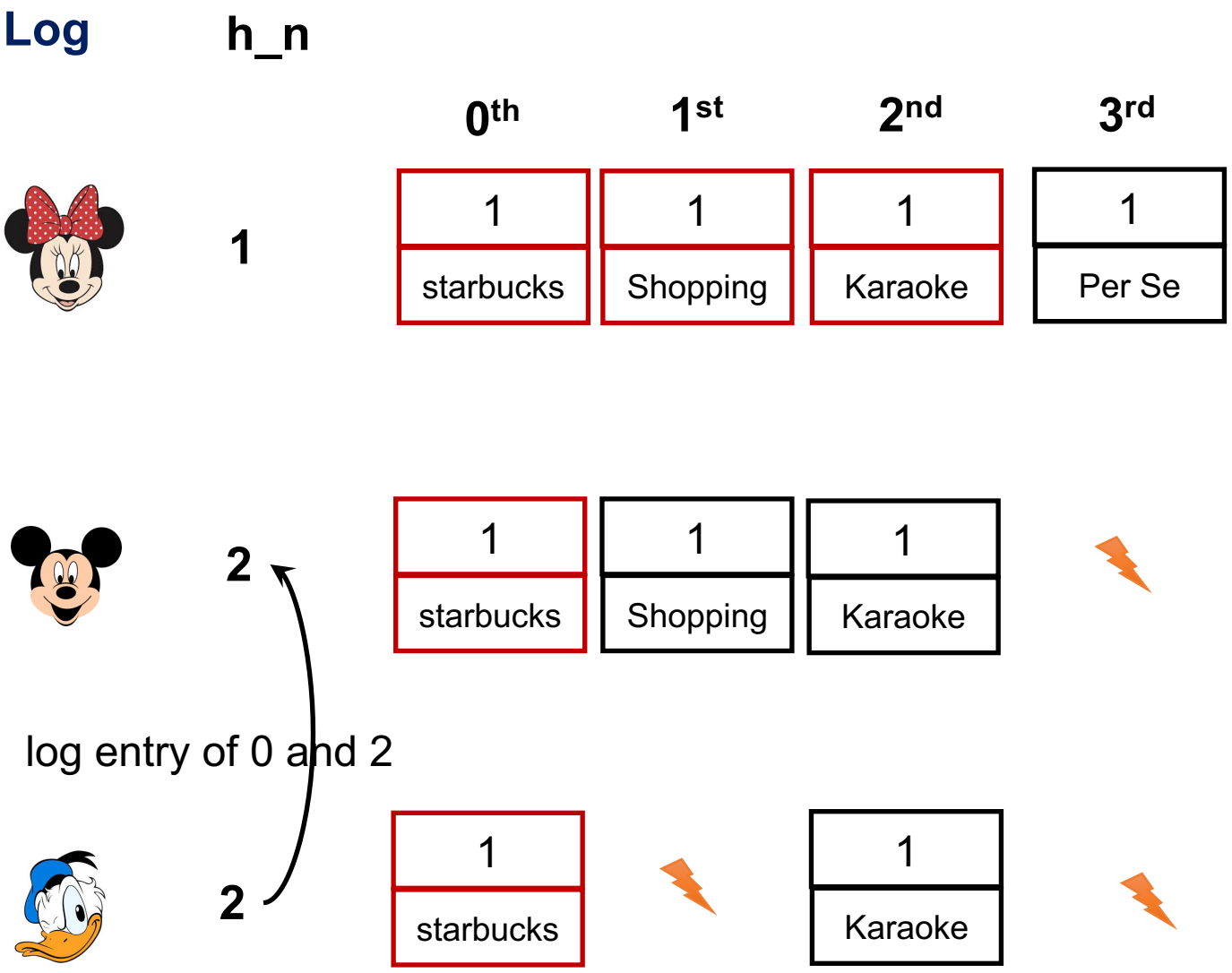
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

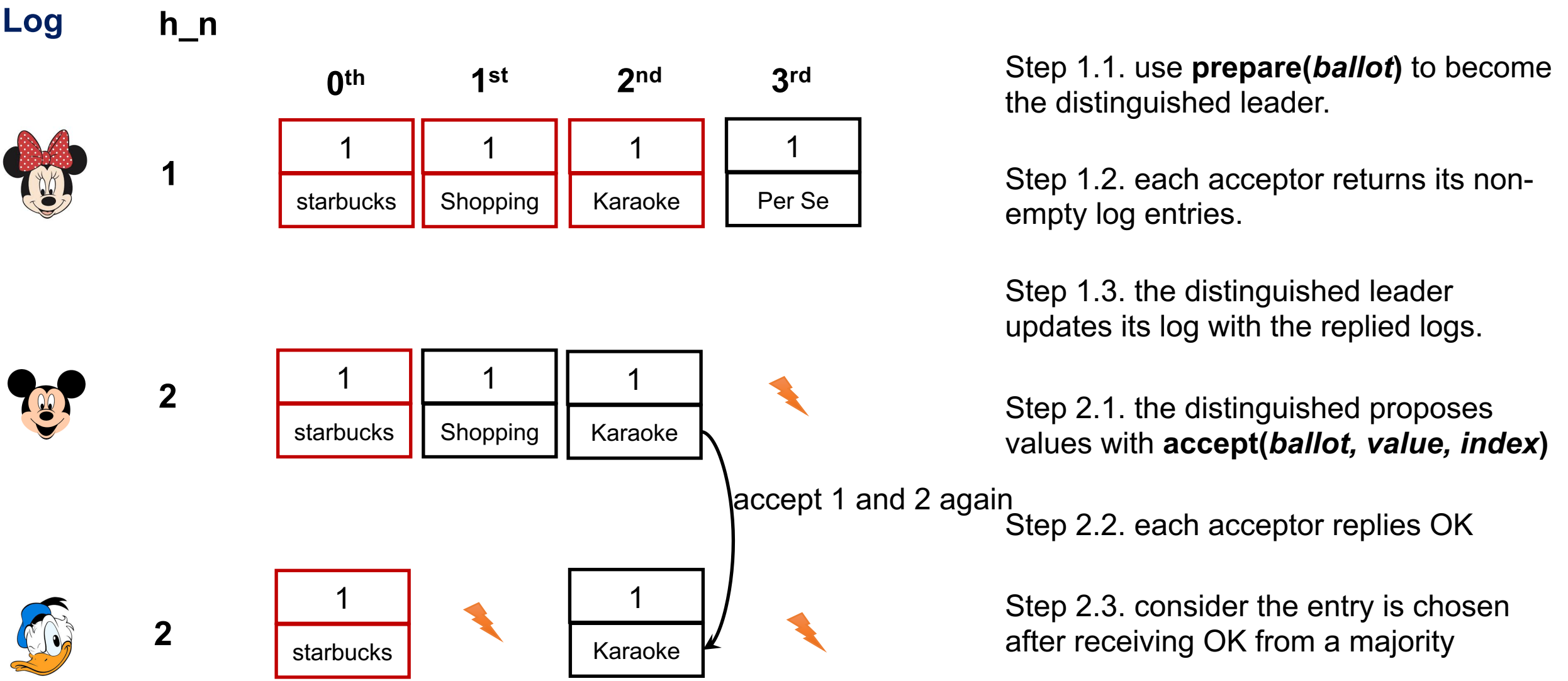
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

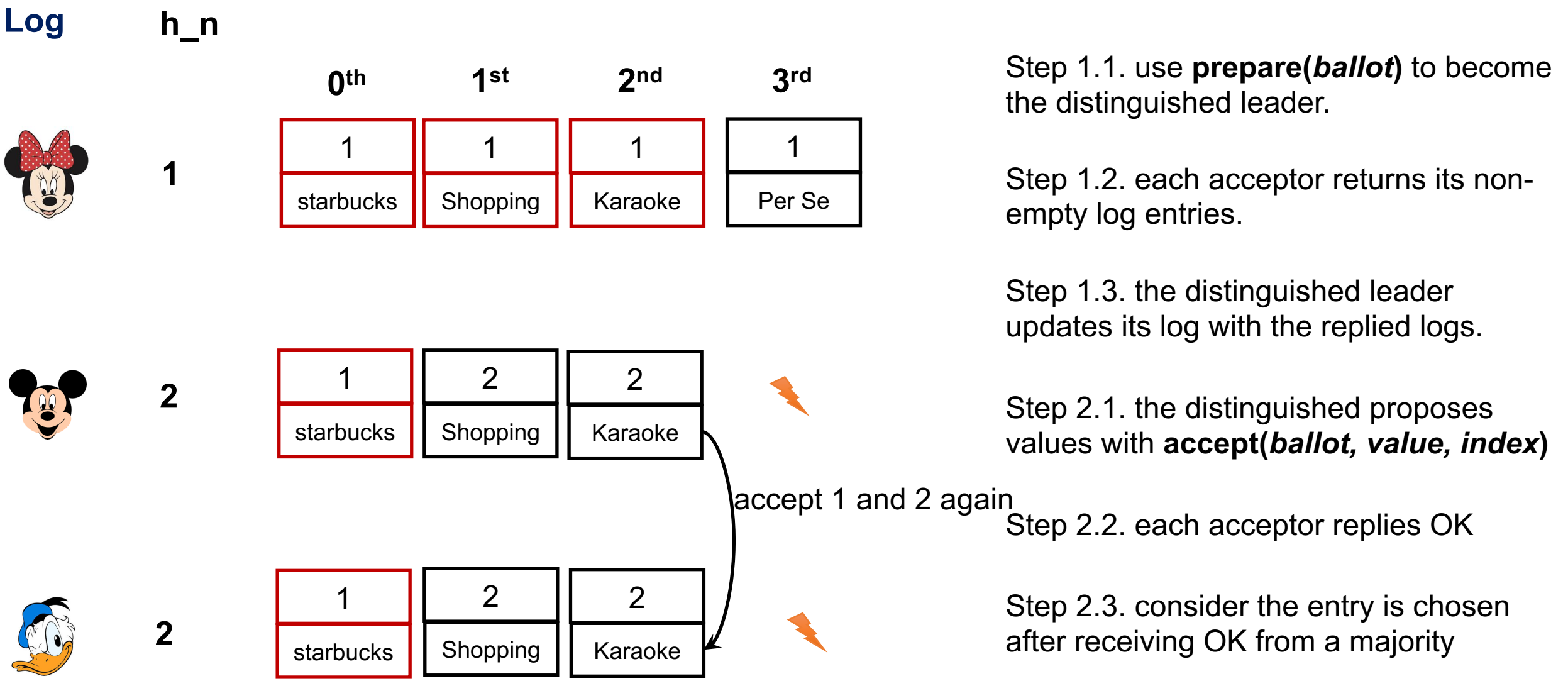
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

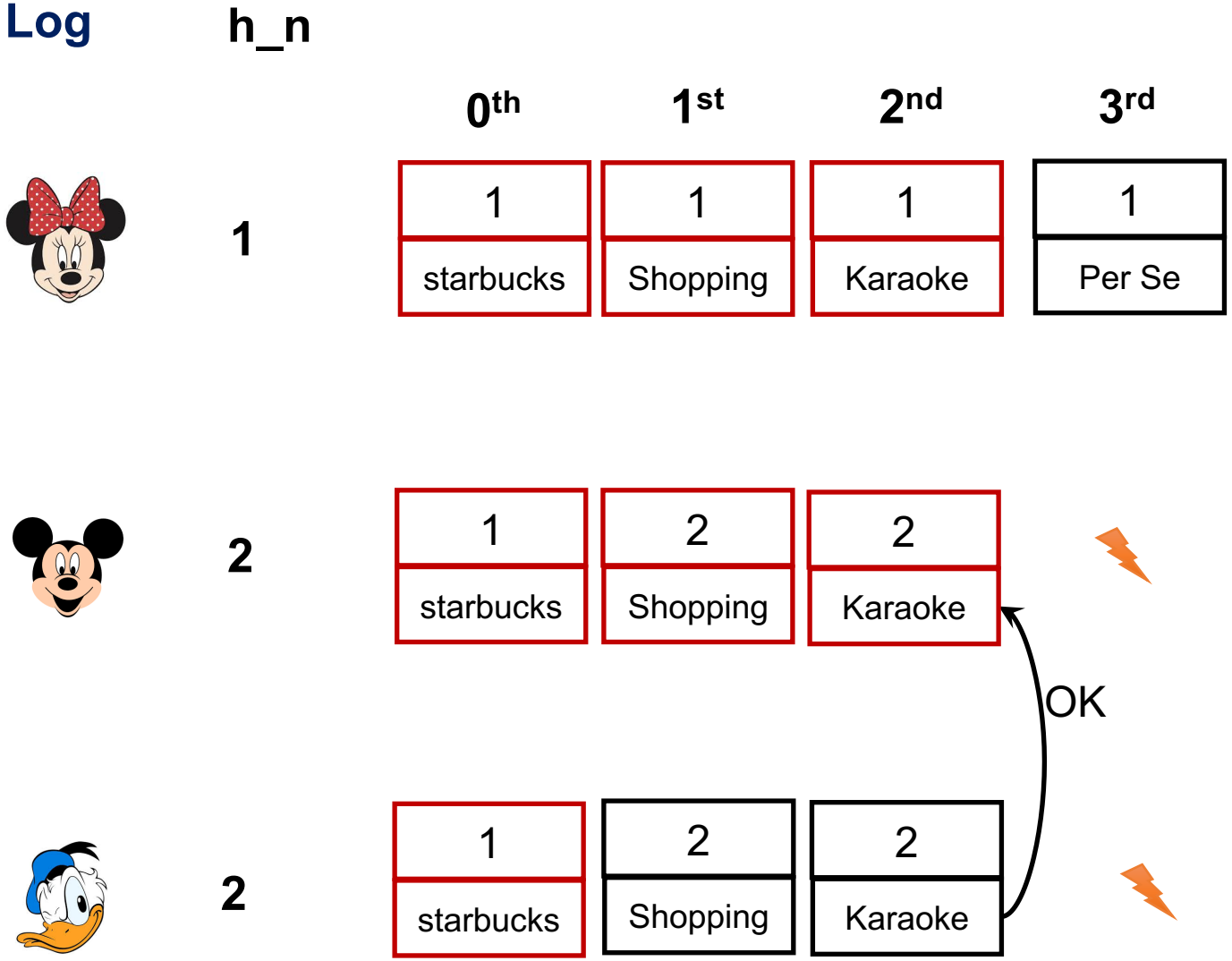
Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos



Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos

Log	h_n	0th	1st	2nd	3rd
	1	1 starbucks	1 Shopping	1 Karaoke	1 Per Se
	2	1 starbucks	2 Shopping	2 Karaoke	2 McDonald
	2	1 starbucks	2 Shopping	2 Karaoke	

Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos

Log	h_n	0th	1st	2nd	3rd
	1	1 starbucks	1 Shopping	1 Karaoke	1 Per Se
	2	1 starbucks	2 Shopping	2 Karaoke	2 McDonald
	2	1 starbucks	2 Shopping	2 Karaoke	2 McDonald

Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Decree Consensus: from Paxos to MultiPaxos

Log	h_n	0th	1st	2nd	3rd
	2	1 starbucks	1 Shopping	1 Karaoke	2 McDonald
	2	1 starbucks	2 Shopping	2 Karaoke	2 McDonald
	2	1 starbucks	2 Shopping	2 Karaoke	2 McDonald

Step 1.1. use **prepare(*ballot*)** to become the distinguished leader.

Step 1.2. each acceptor returns its non-empty log entries.

Step 1.3. the distinguished leader updates its log with the replied logs.

Step 2.1. the distinguished proposes values with **accept(*ballot, value, index*)**

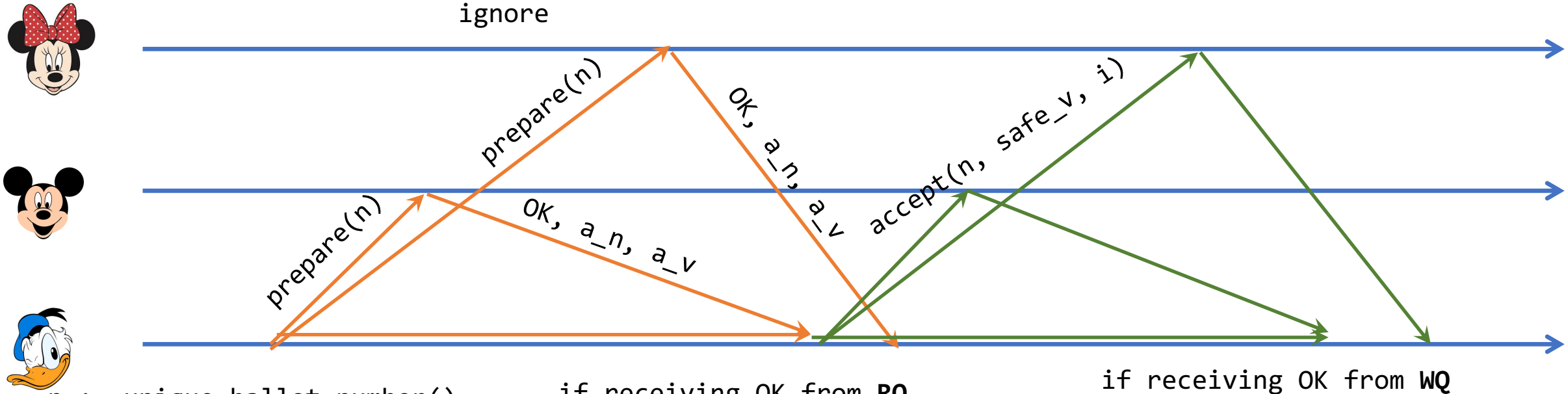
Step 2.2. each acceptor replies OK

Step 2.3. consider the entry is chosen after receiving OK from a majority

Multi-Paxos Algorithm

```
if n >= h_n
  h_n = n
  log[i].a_n = log[i].n
  log[i].a_v = log[i].safe_v
  return OK
else
  ignore
```

```
if n > h_n
  h_n = n
  return <OK, non-empty log entries>
else
  ignore
```




```
n := unique_ballot_number()
```

```
if receiving OK from RQ
  for each log entry
    calculate the safe value
      with the a_n
```

```
if receiving OK from WQ
  log[i] is chosen
```

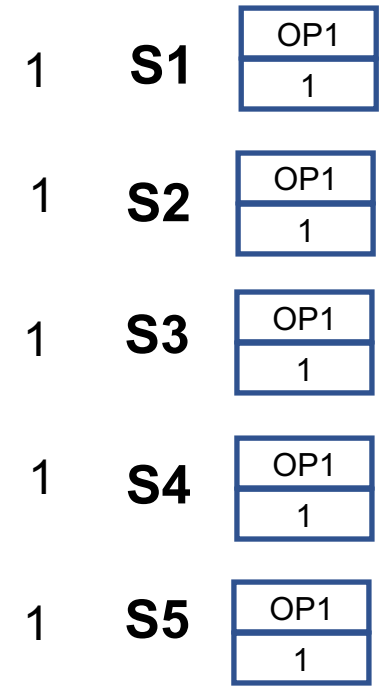
h_n – highest ballot number has been seen

a_n – highest ballot have been accepted

a_v – value accepted at a_n

Exercise I

h_n

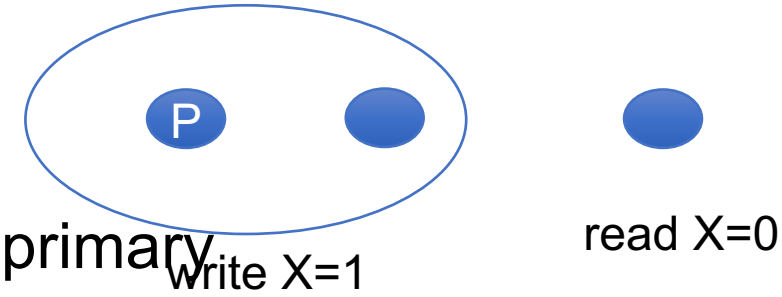


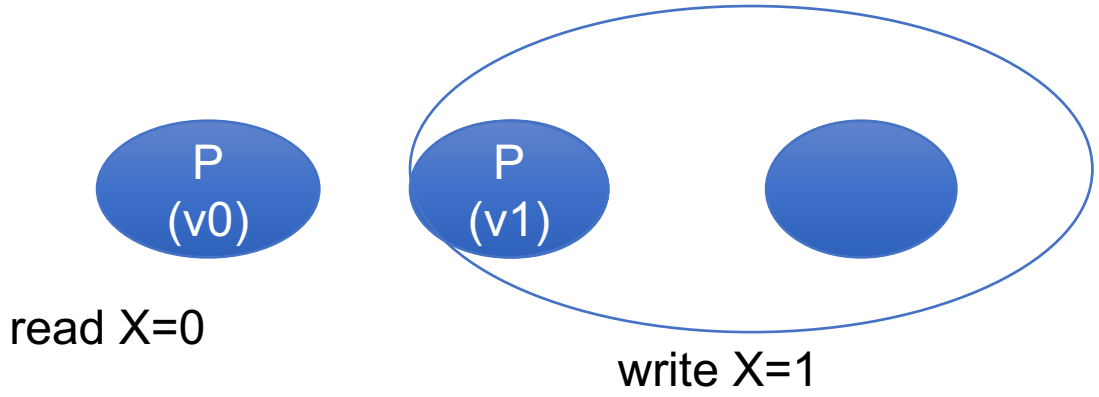
What is the eventual view number?
What is the content of the leader's log?

1. Network Partitions S1, S2 from Others 2. S1 receives OP2, OP3, OP4 3. OP5 and OP6 got chosen
4. S2 receives OP2, OP3, OP4 from S1 5. S3 crashes 6. Network heals 7. S1 tries to commit OP2, OP3, OP4
8. OP7 got chosen 9. S4 crashes 10. S3 recovers

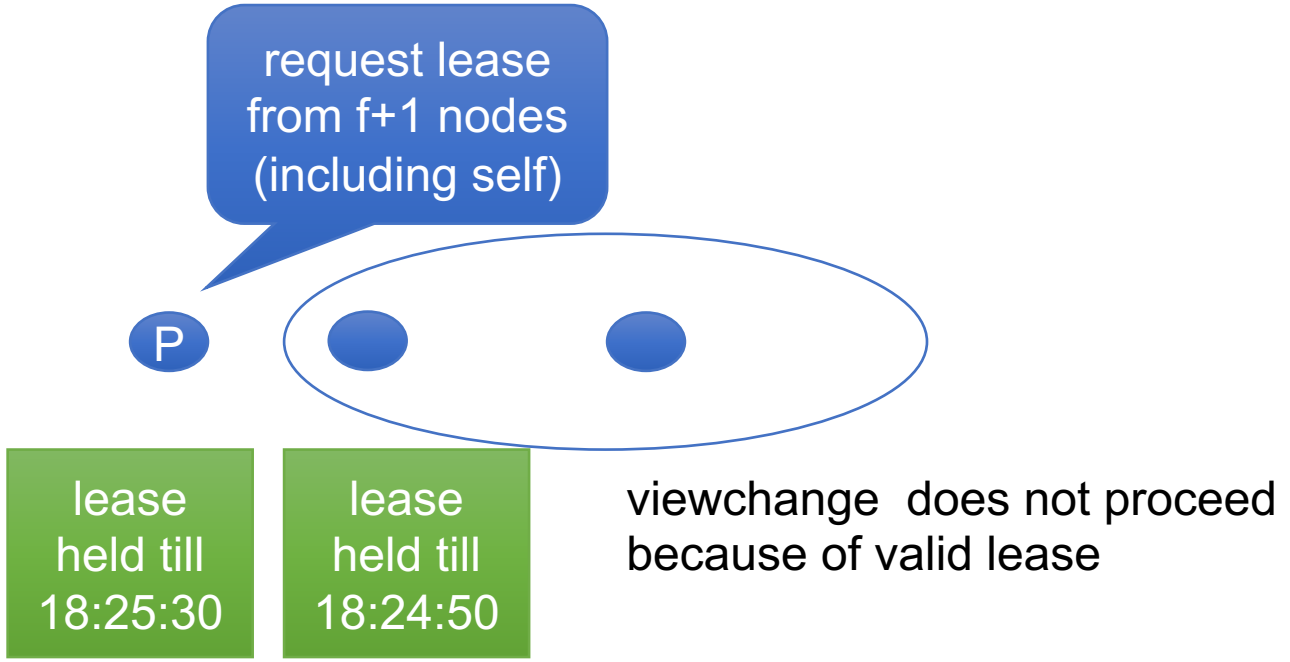
How about this optimization

- Viewstamp with read optimization.
  - read at any arbitrary server
  - read at the primary





The “correct” VR read optimization



Strong consistency models in practice

- The memory model of Intel CPU?
- Popular key-value stores
  - S3
  - Dynamo
  - MySQL with asynchronous replication