

Paxos

Zhaoguo Wang

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

Paxos Solves The Consensus Problem



The screenshot shows the ACM A.M. Turing Award website. At the top left is the ACM logo with the text "MORE ACM AWARDS". To its right is the "A.M. TURING AWARD" logo featuring a portrait of Alan Turing. Further right is a search bar with a home icon, a "Search" button, and a "TYPE HERE" input field. Below the search bar is a grid of 24 small portraits of past award winners. A navigation bar below the grid has three tabs: "ALPHABETICAL LISTING", "YEAR OF THE AWARD", and "RESEARCH SUBJECT". The "ALPHABETICAL LISTING" tab is selected. Below the tabs, a large portrait of Leslie Lamport is shown on the left. To his right, his name "LESLIE LAMPORT" is displayed in large blue letters, followed by a green "DL" icon. Below his name, it says "United States – 2013". Underneath that, the word "CITATION" is written in orange. The citation text reads: "For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency." At the bottom of the page, there are four small blue icons representing different social media or sharing options.

acm
MORE ACM AWARDS

A.M. TURING AWARD

Search TYPE HERE

A.M. TURING AWARD WINNERS BY...

ALPHABETICAL LISTING YEAR OF THE AWARD RESEARCH SUBJECT



LESLIE LAMPORT DL

United States – 2013

CITATION

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

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

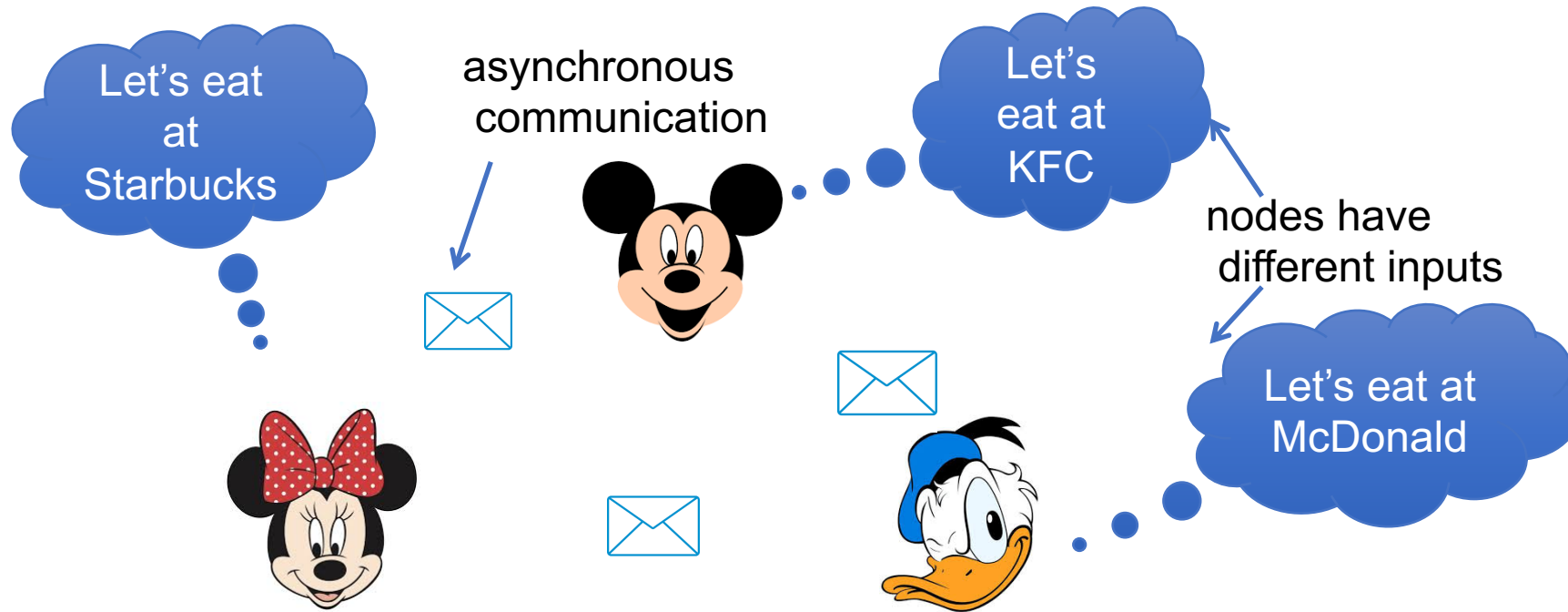
Problem setting

- 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

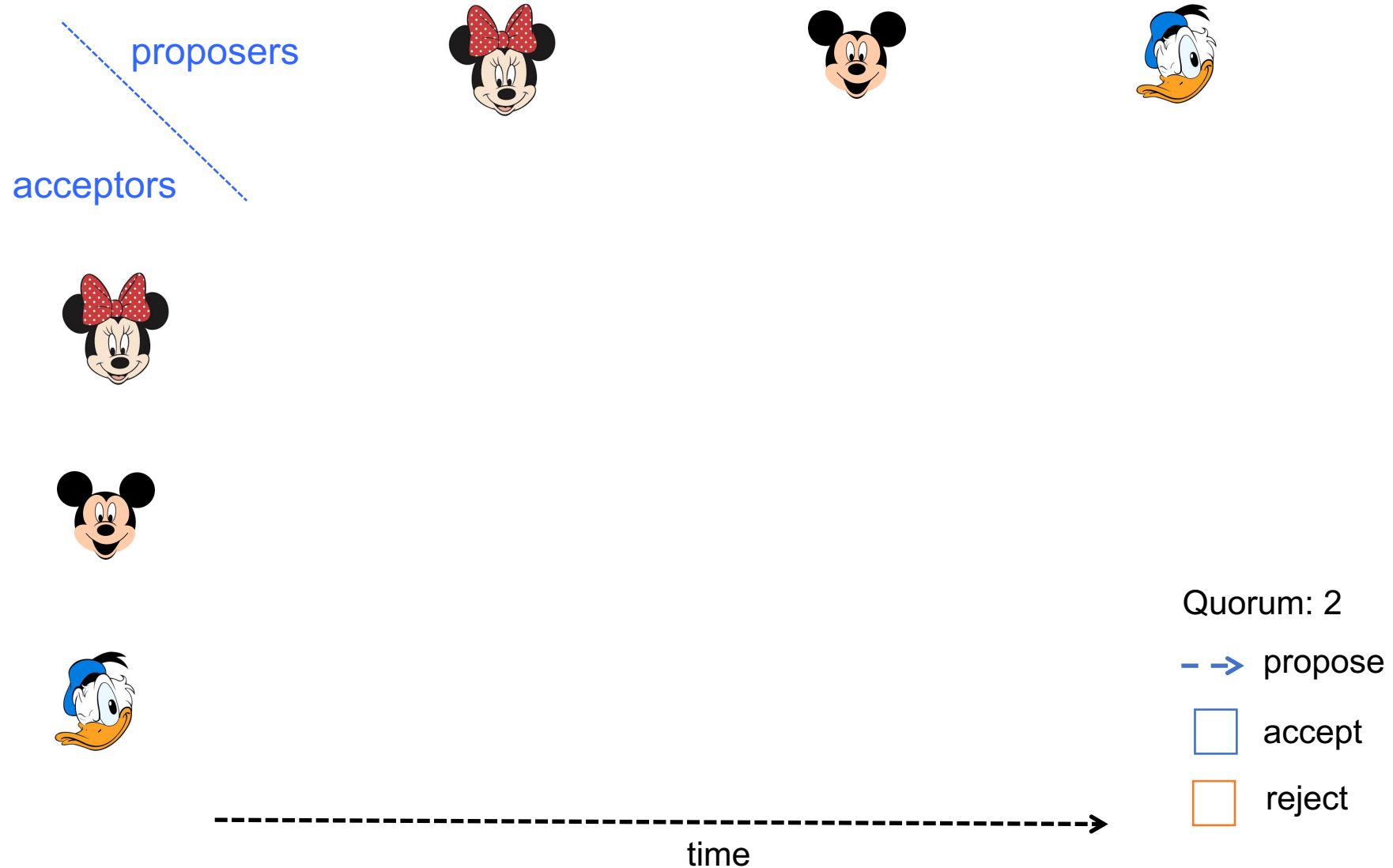
Roles

- 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

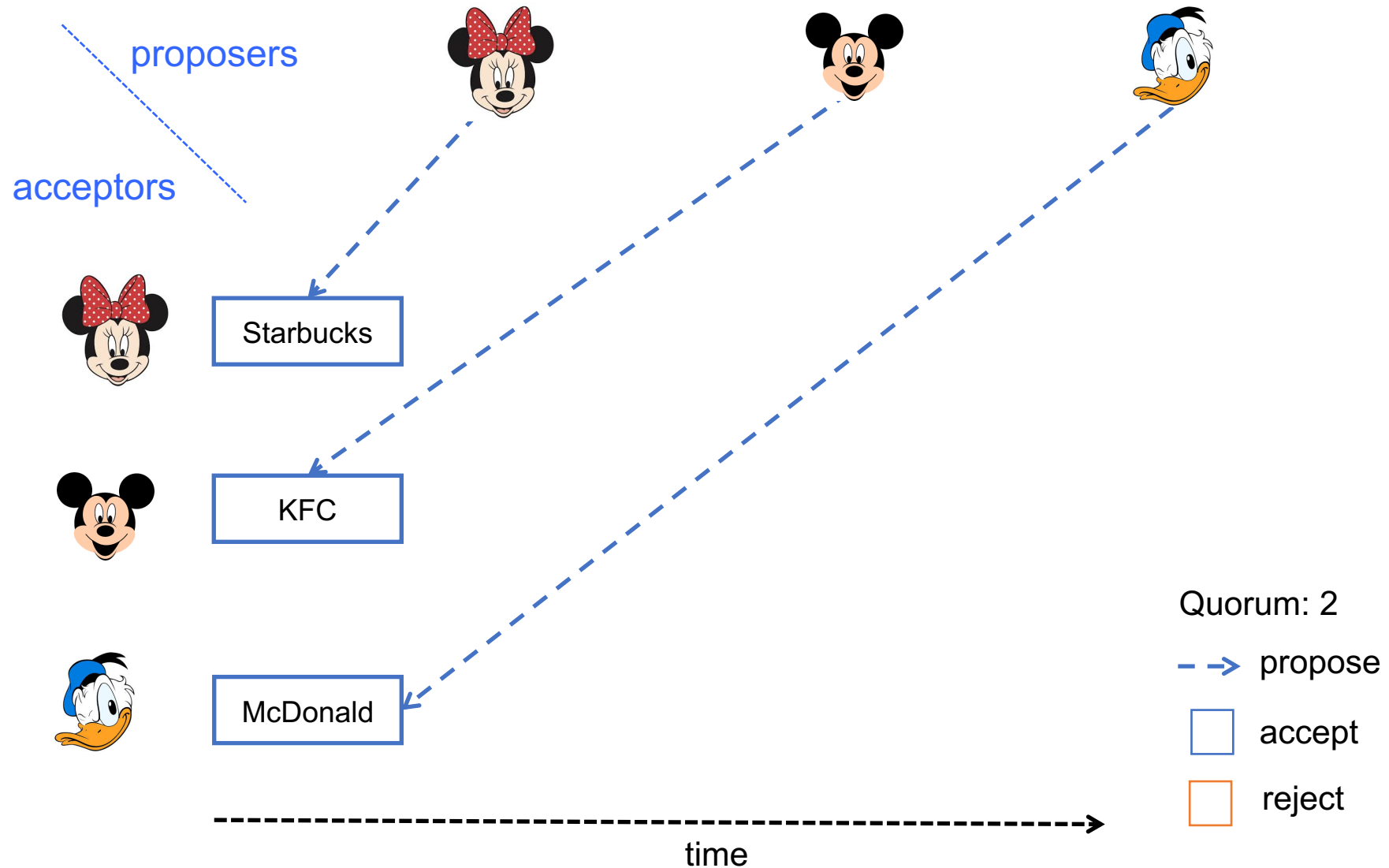
Chosen

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

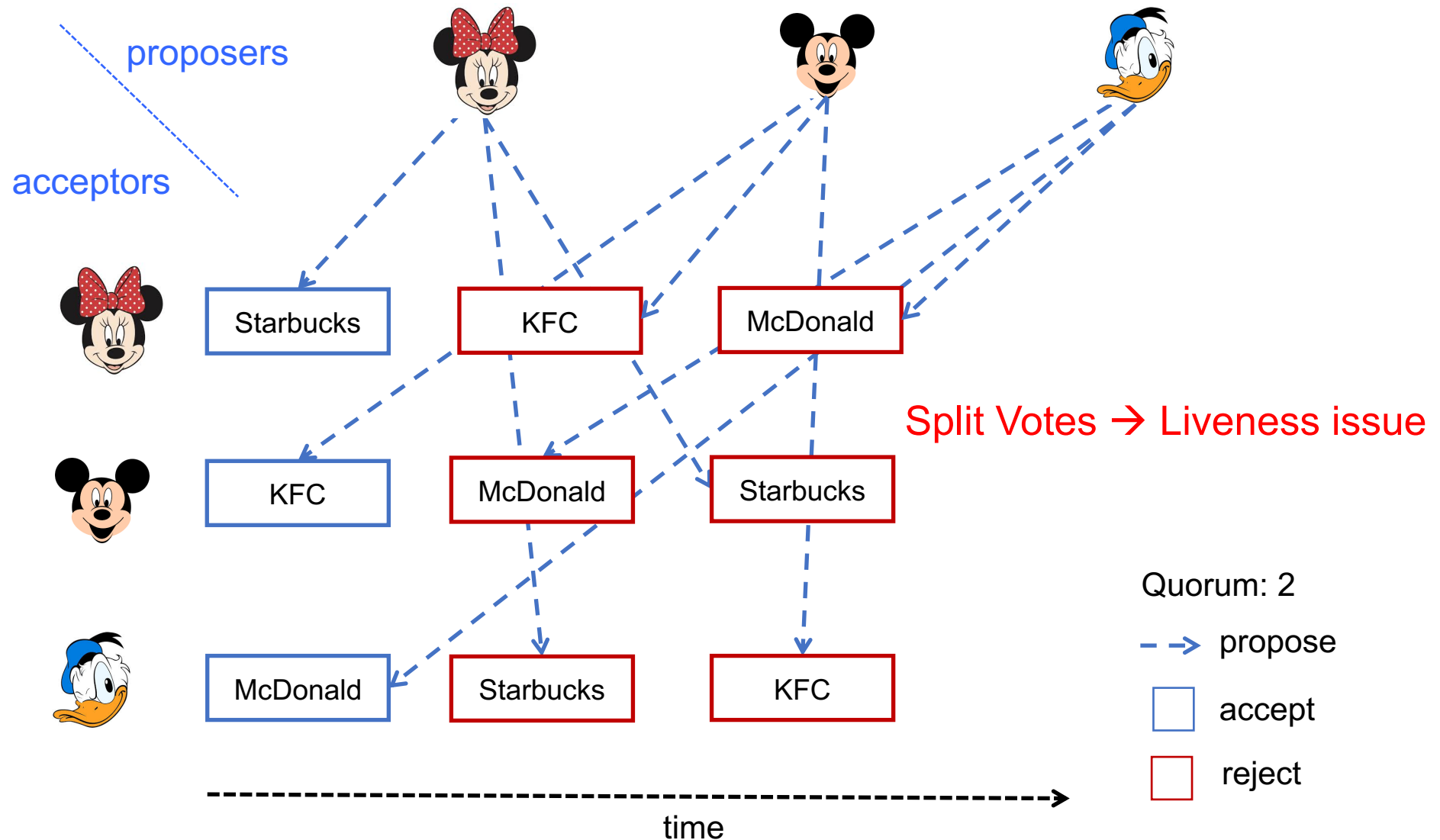
Method I – Only accept the first



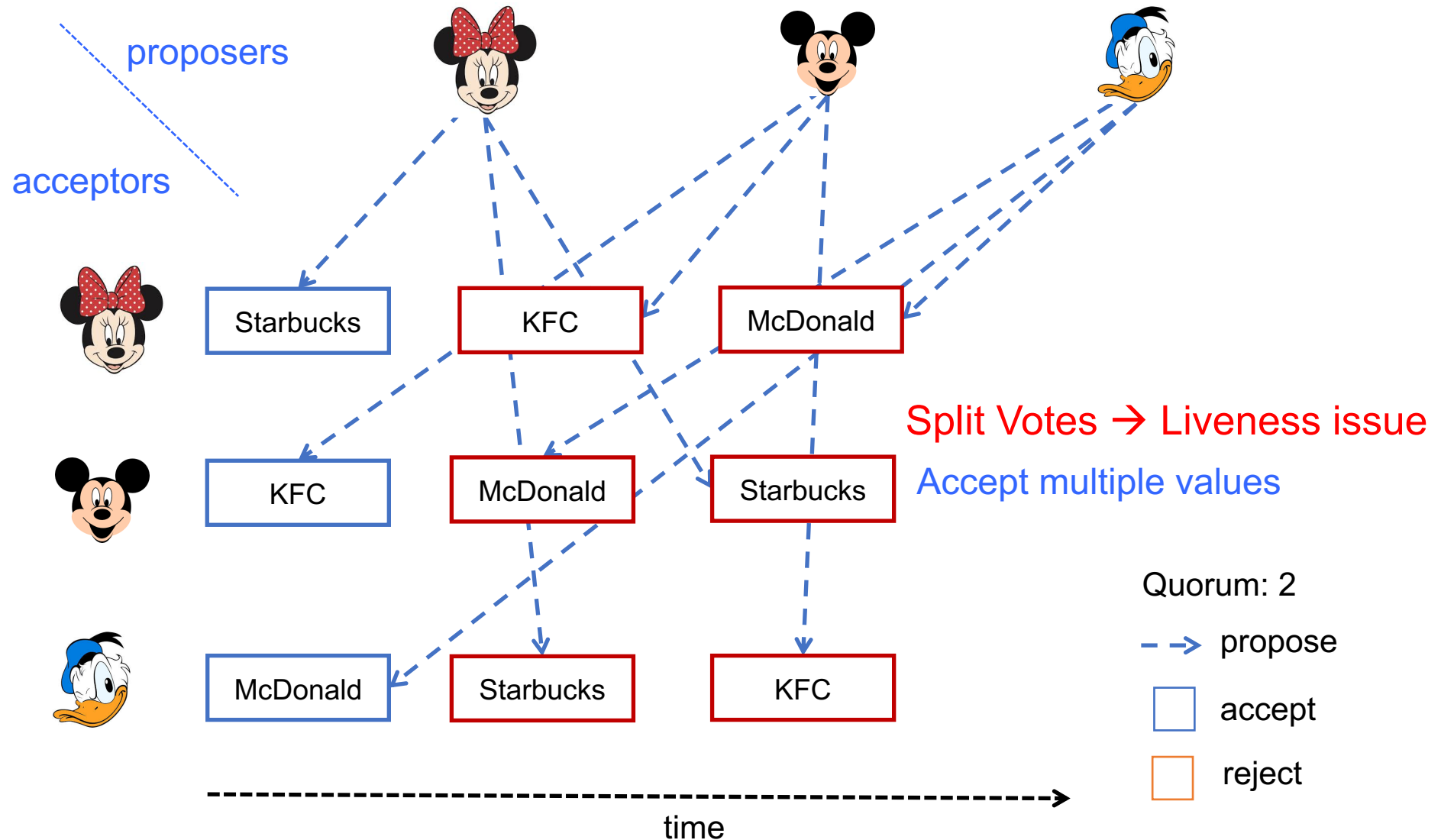
Method I – Only accept the first



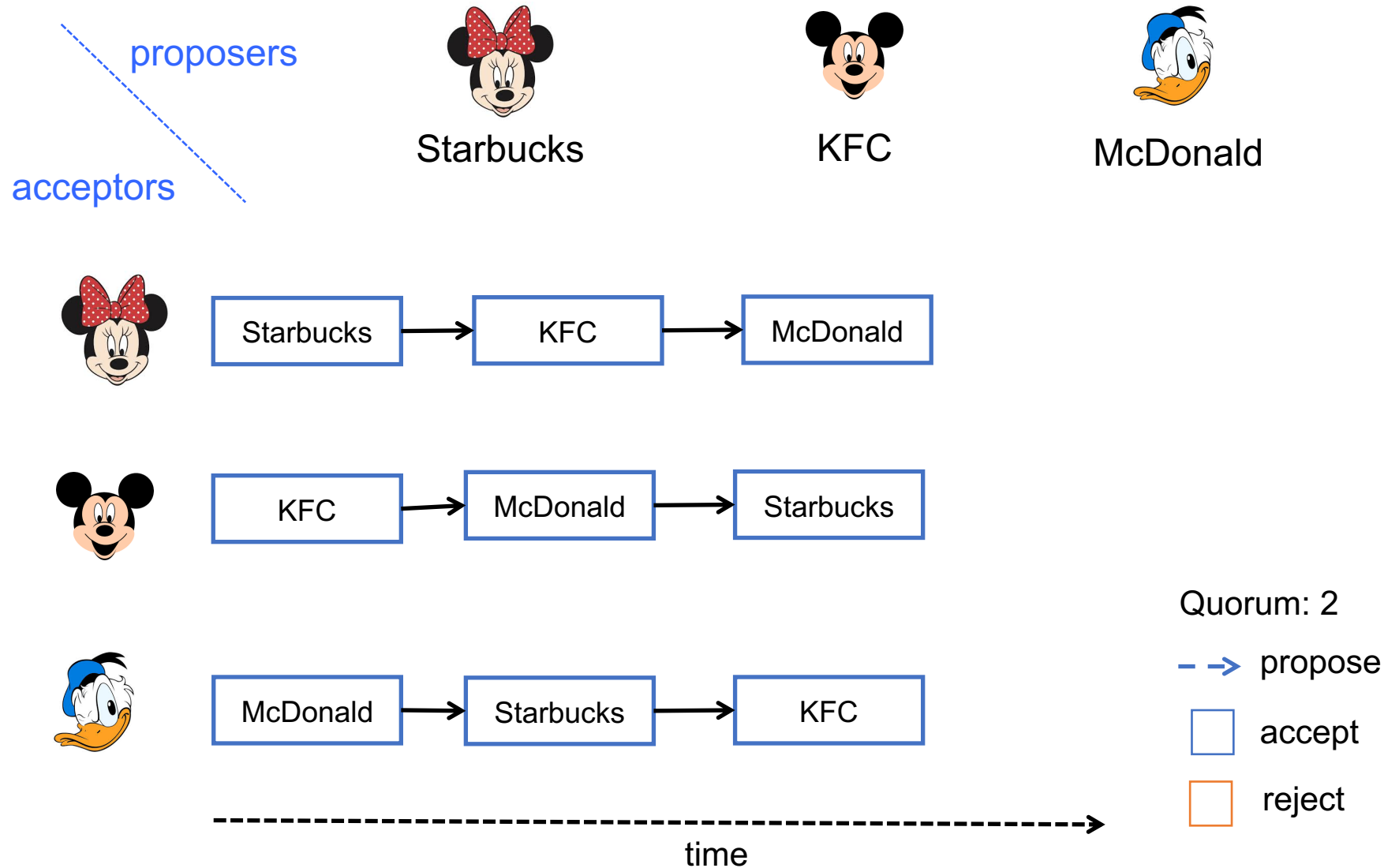
Method I – Only accept the first



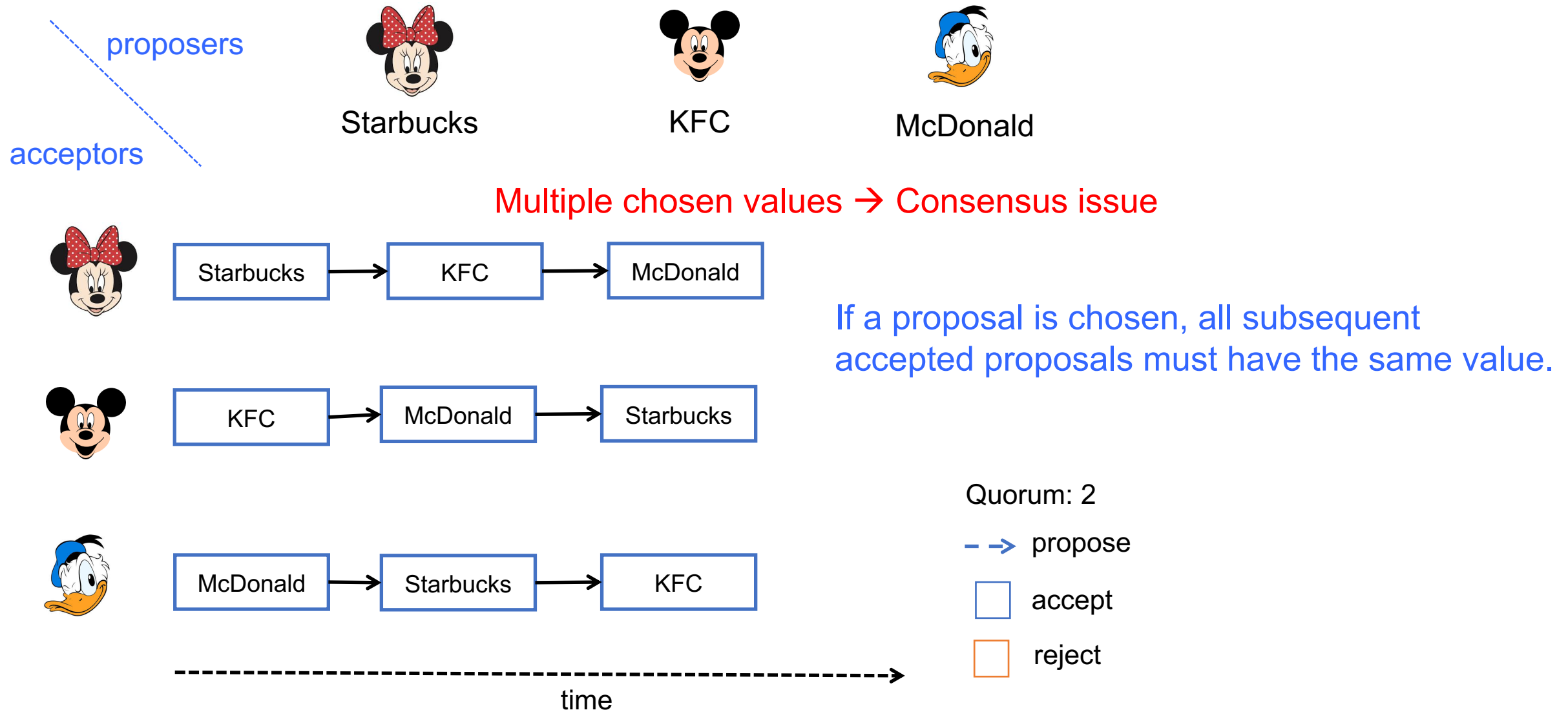
Method I – Only accept the first



Method II – Accept multiple proposed values



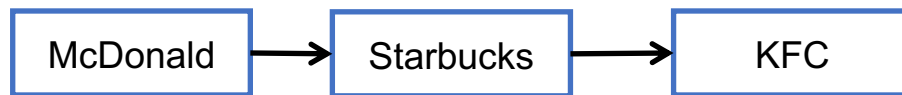
Method II – Accept multiple proposed values



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

time

Basic idea of Paxos

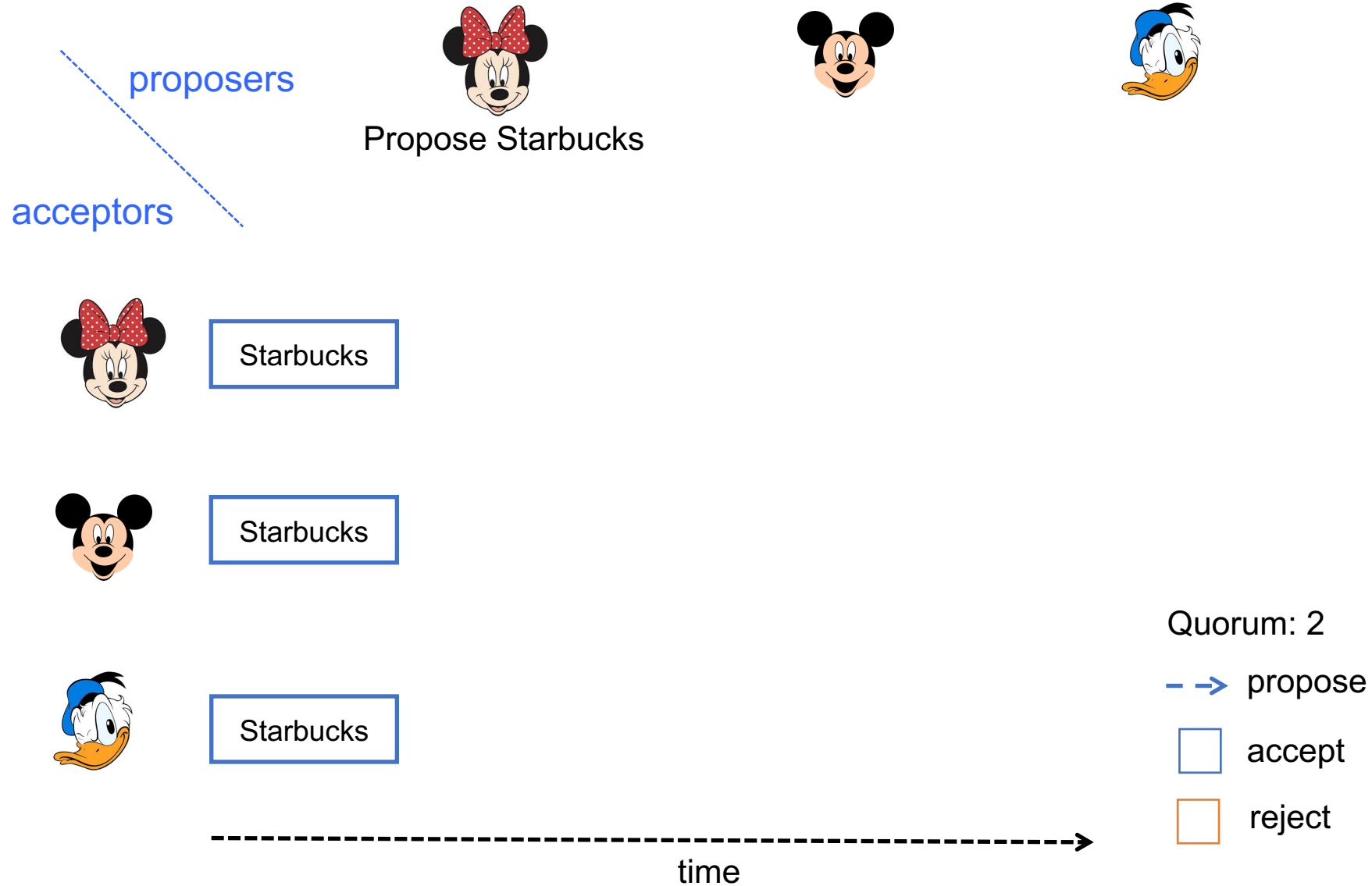
Liveness

- Acceptors can accept more than one proposals

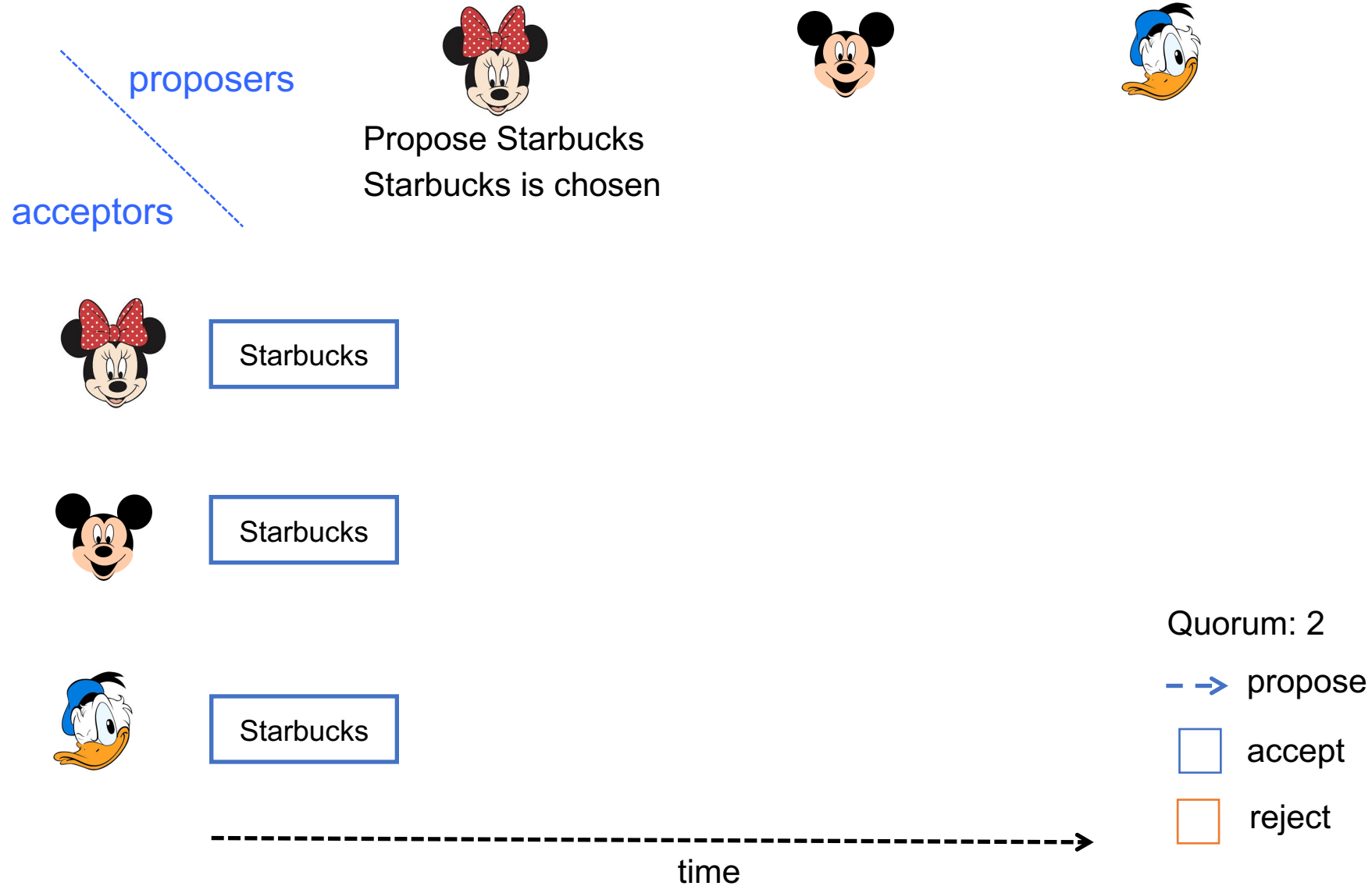
Consensus (Safety)

- 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.

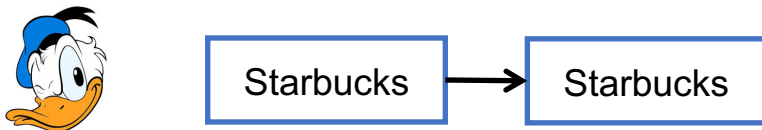
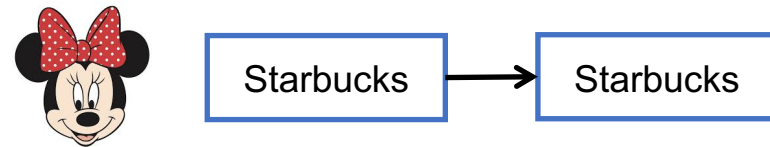
Review The Example



Review The Example



Review The Example



time →

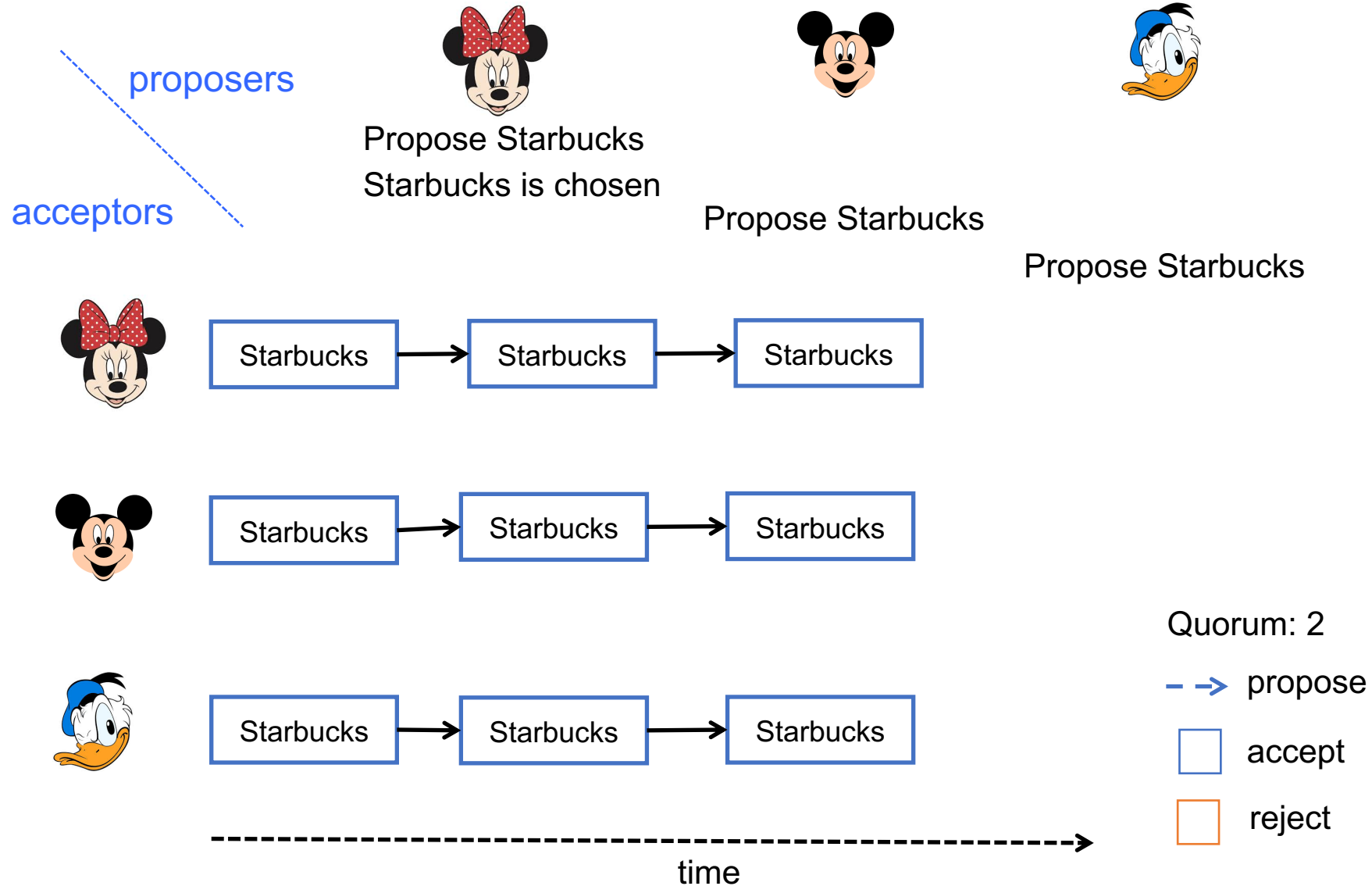
Quorum: 2

→ propose

□ accept

□ reject

Review The Example



Challenges

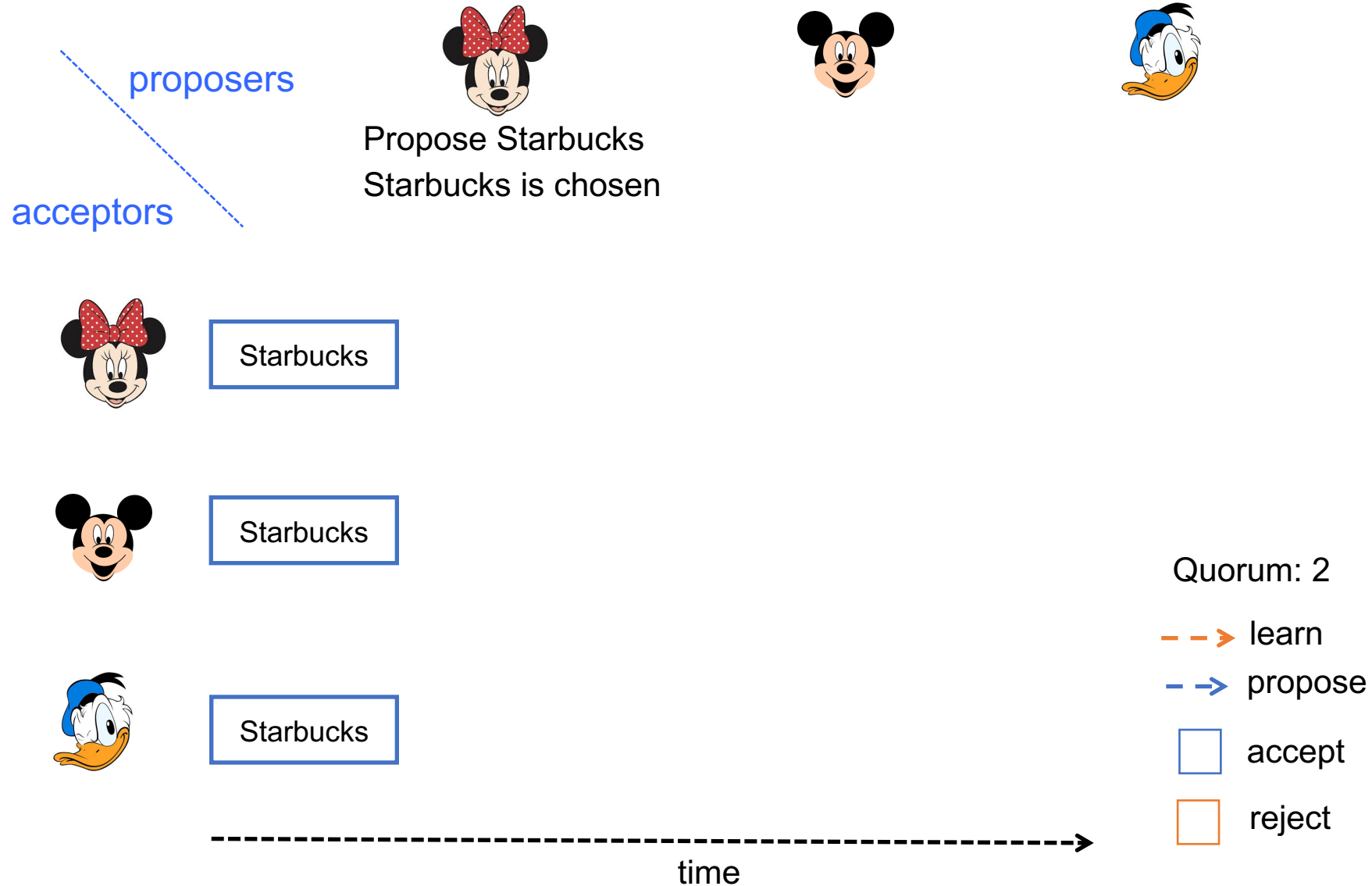
Discover the safe value

Challenges

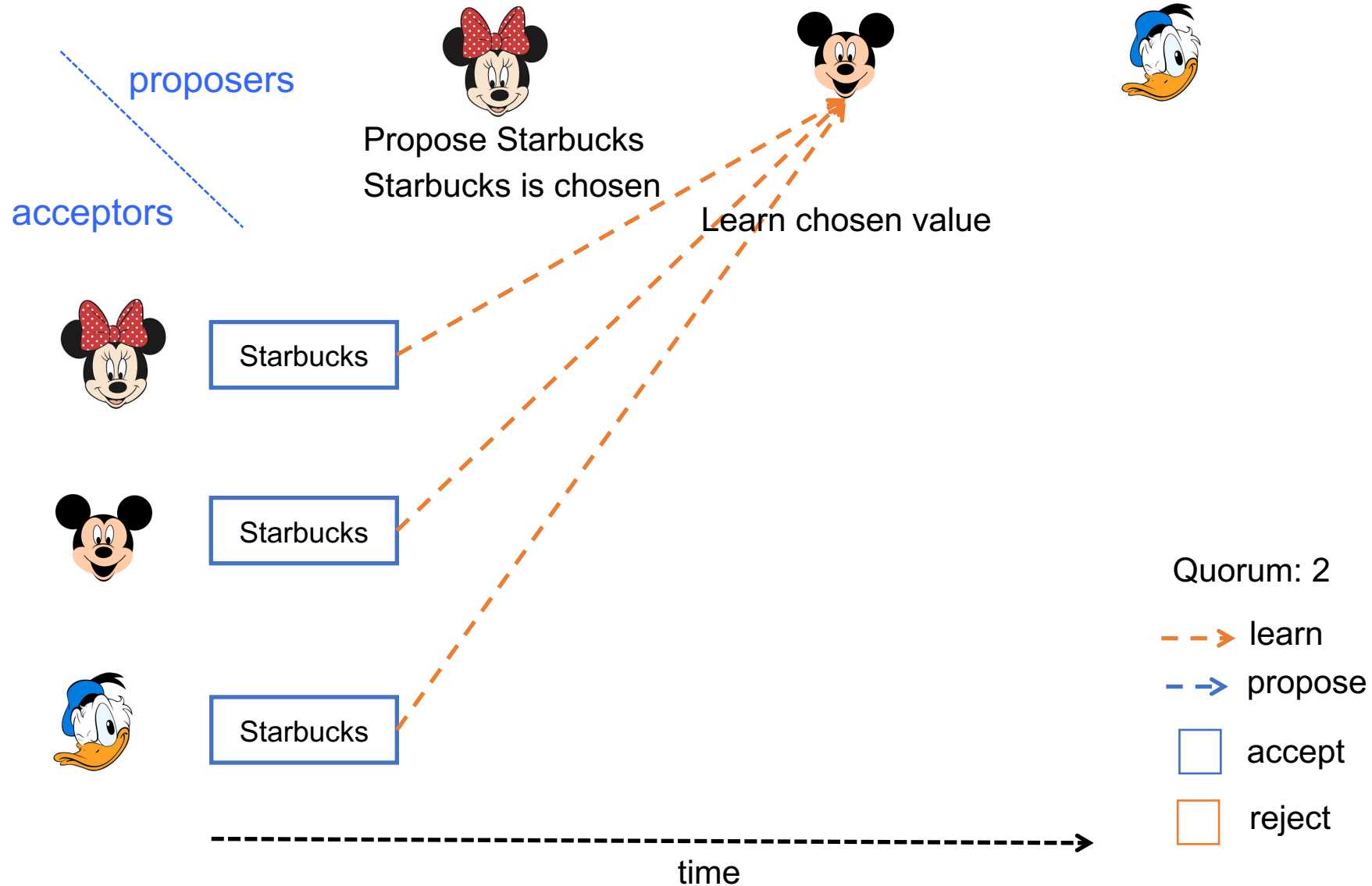
Discover the safe value

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

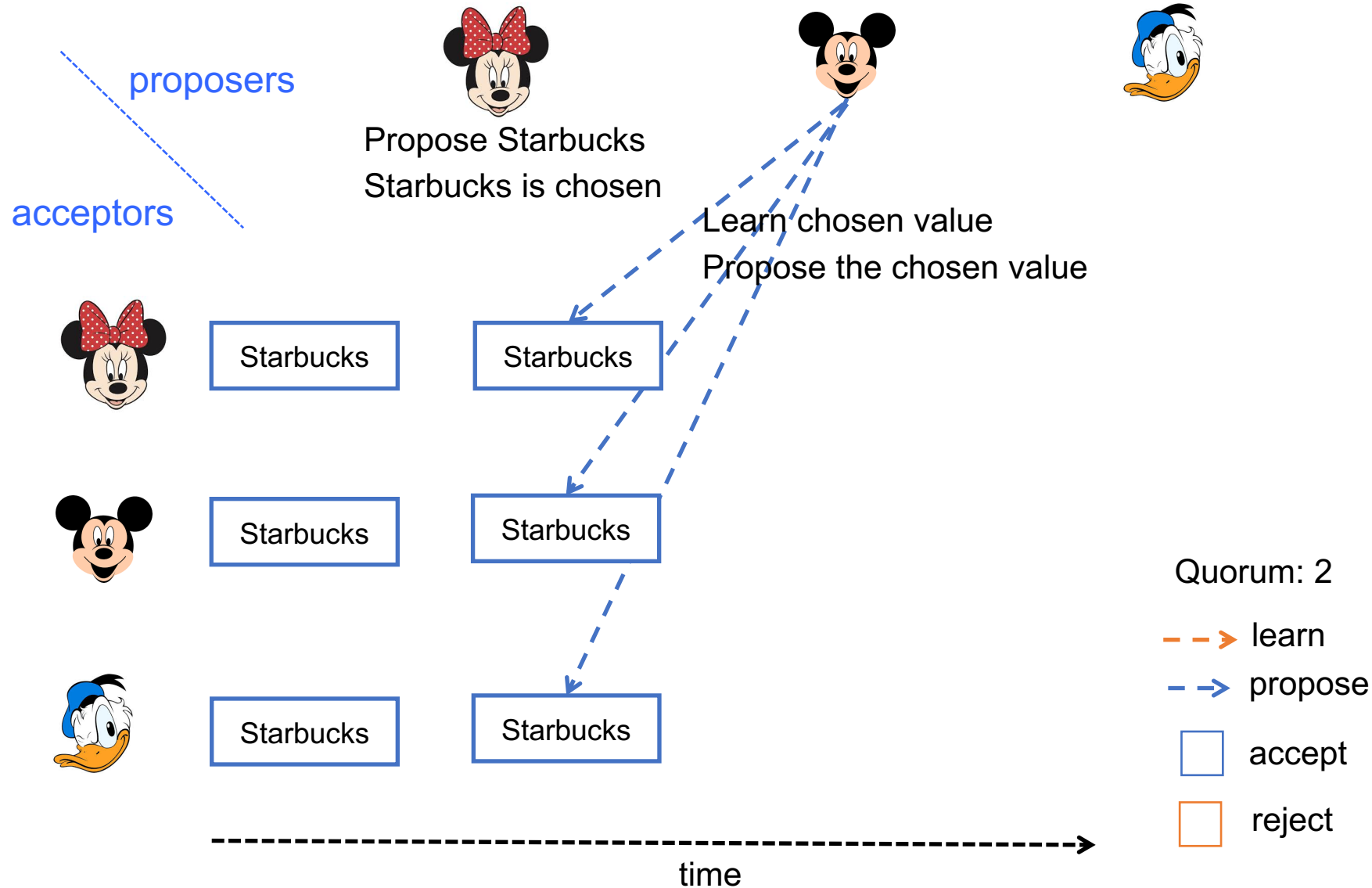
Discover the safe value with extra round trip



Discover the safe value with extra round trip



Discover the safe value with extra round trip



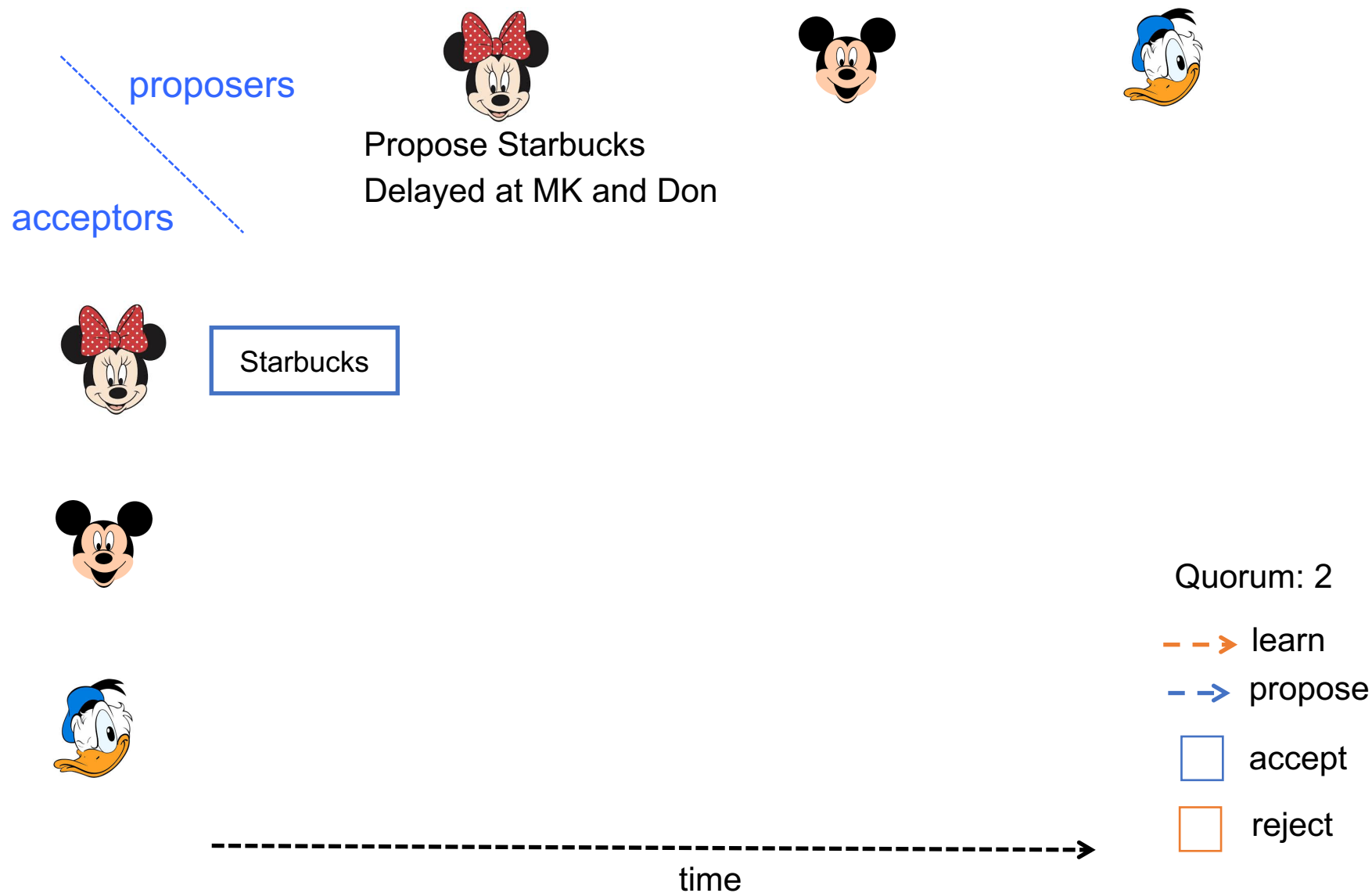
Challenges

Discover the safe value

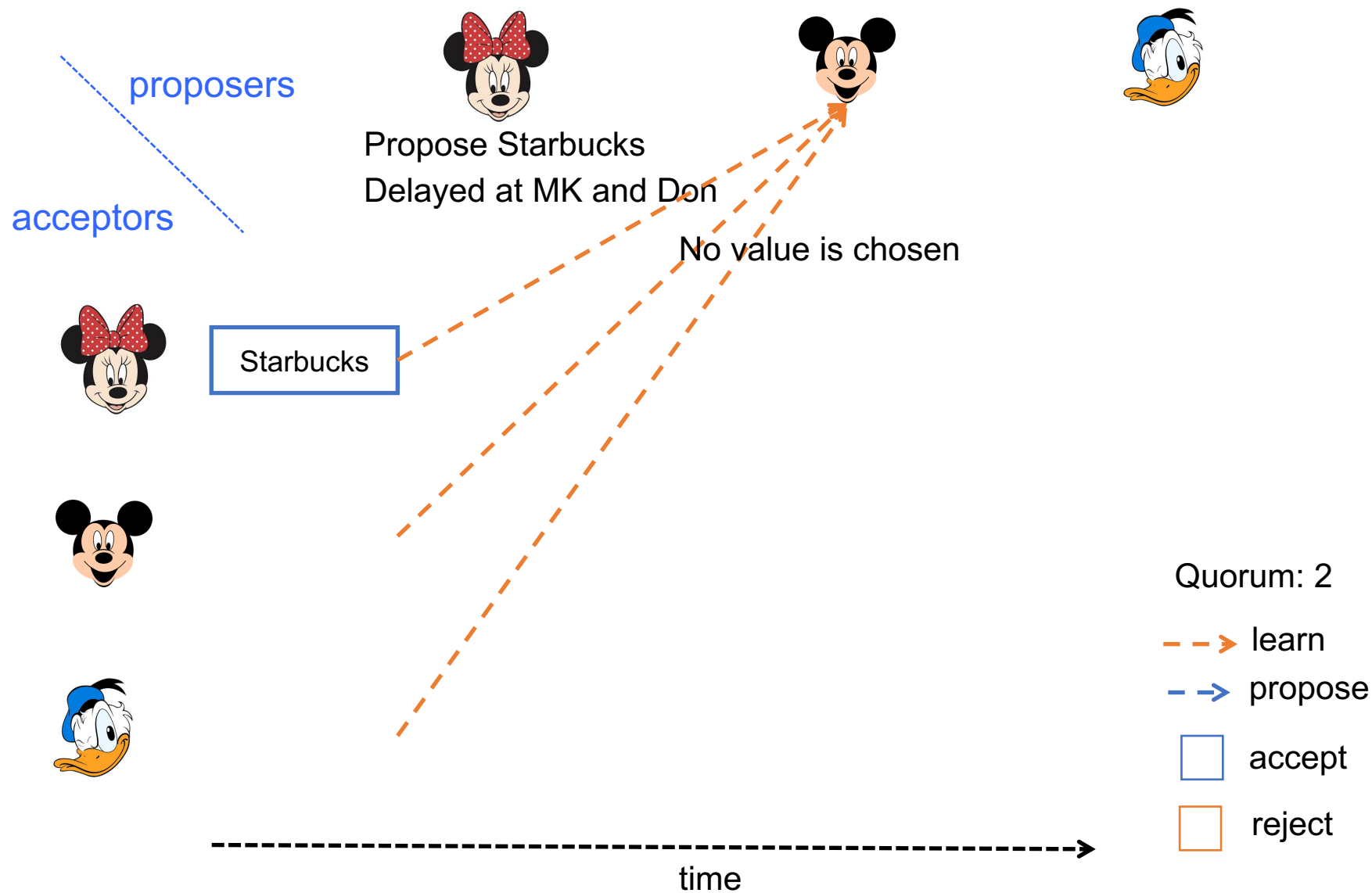
- Use an extra round of communication to find out the safe value

Asynchronous network delays/reorders packets

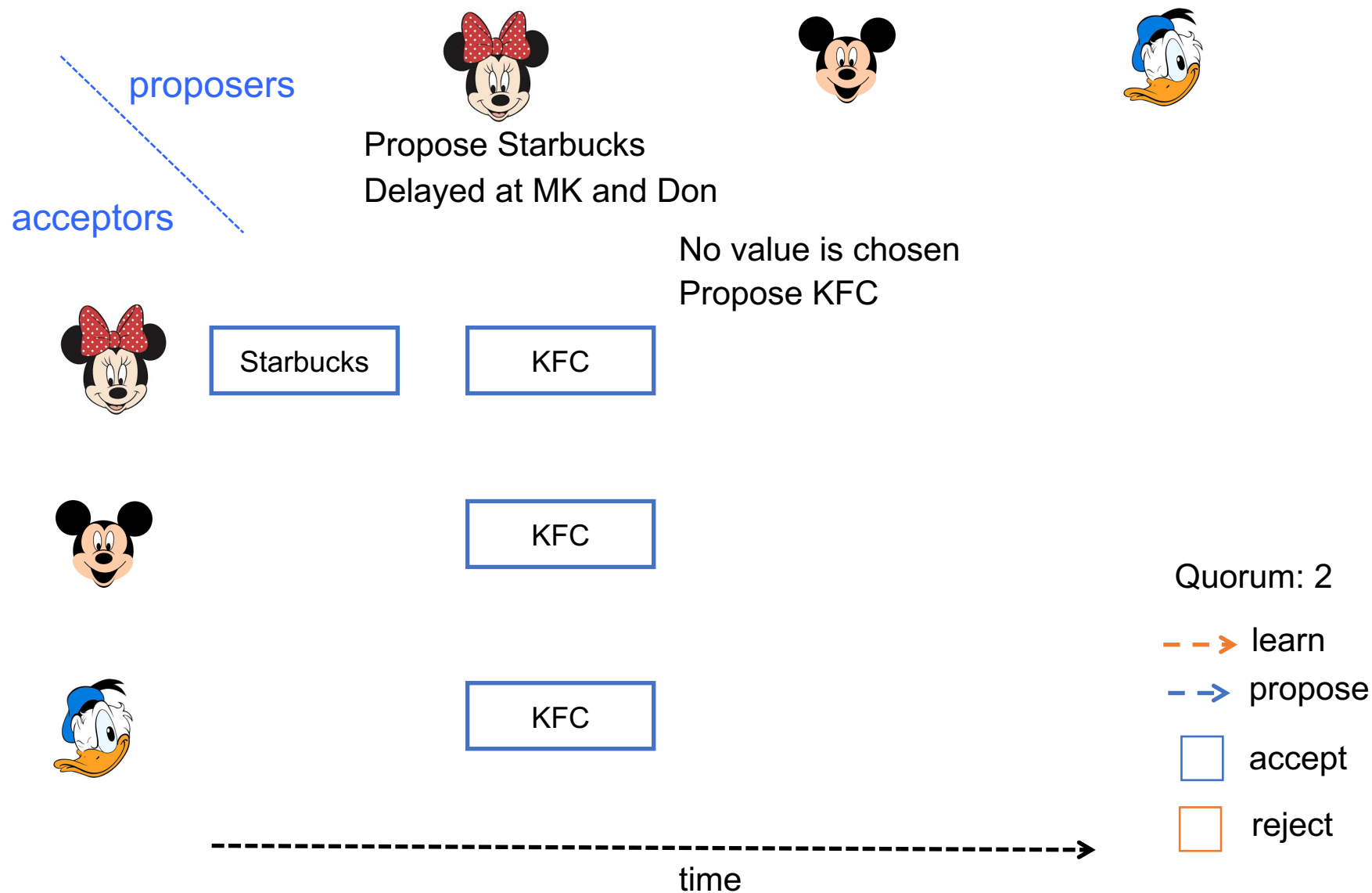
Asynchronous network delays/reorders packets



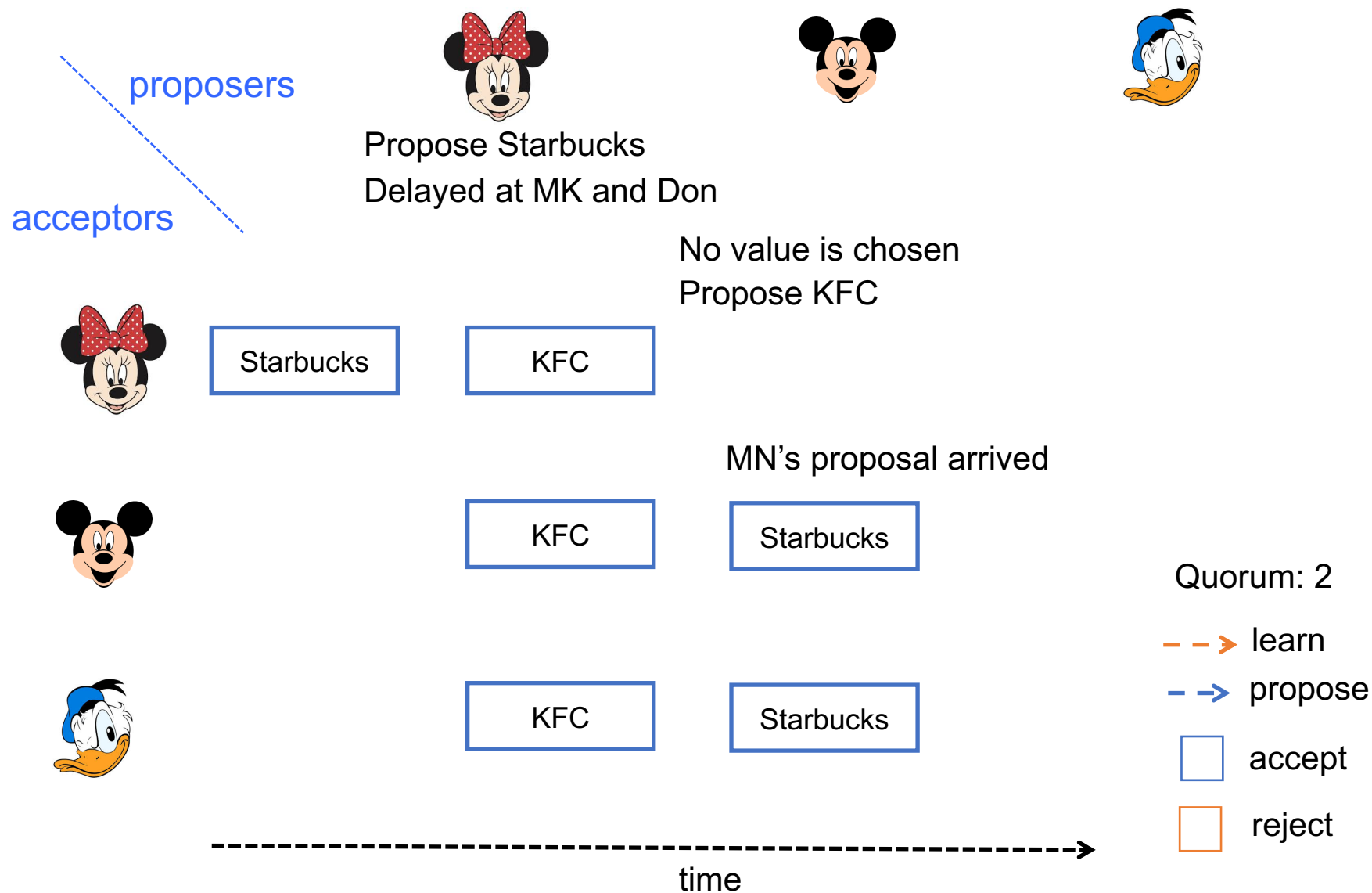
Asynchronous network delays/reorders packets



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

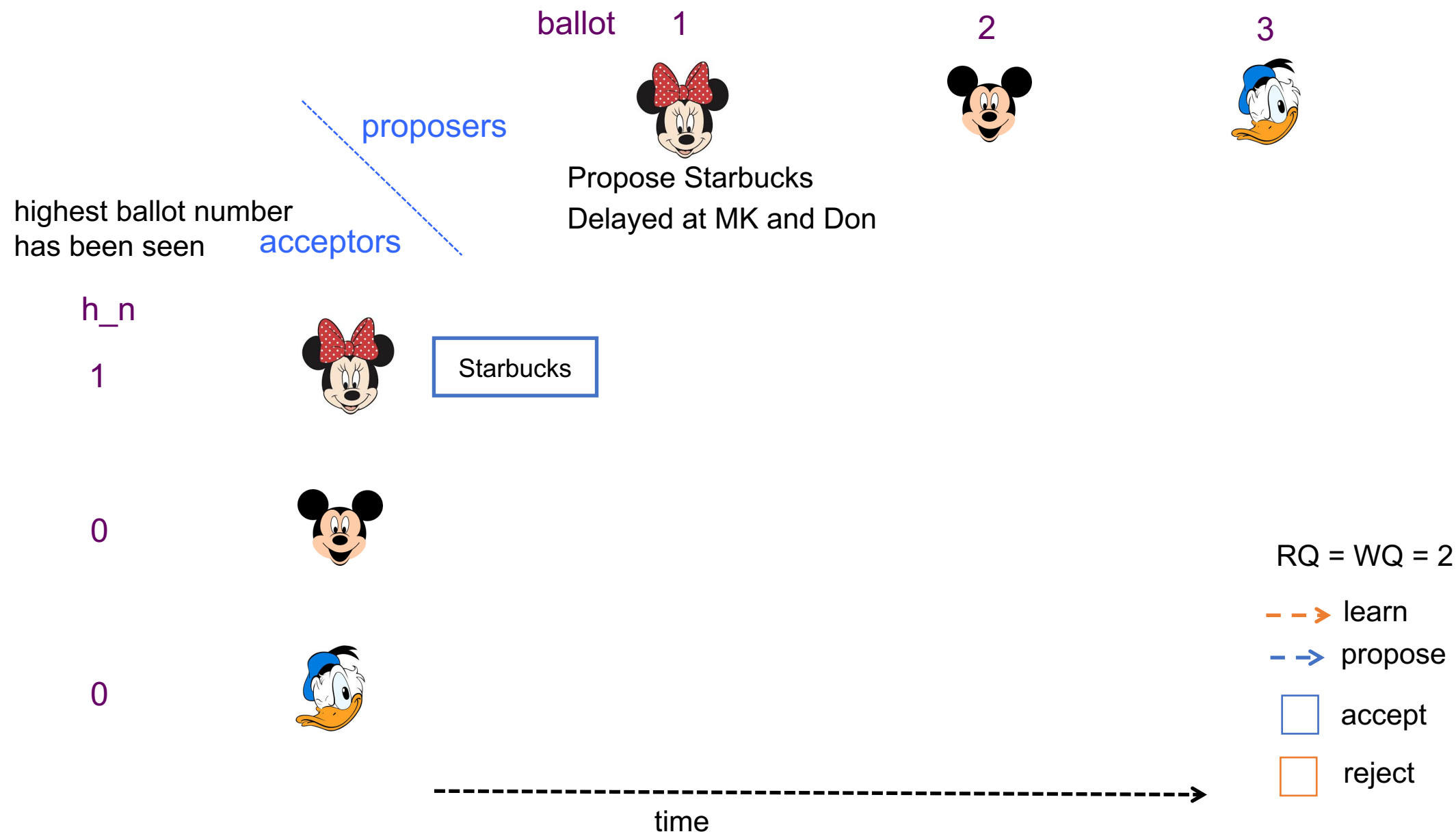
Phase 1 – Prepare

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

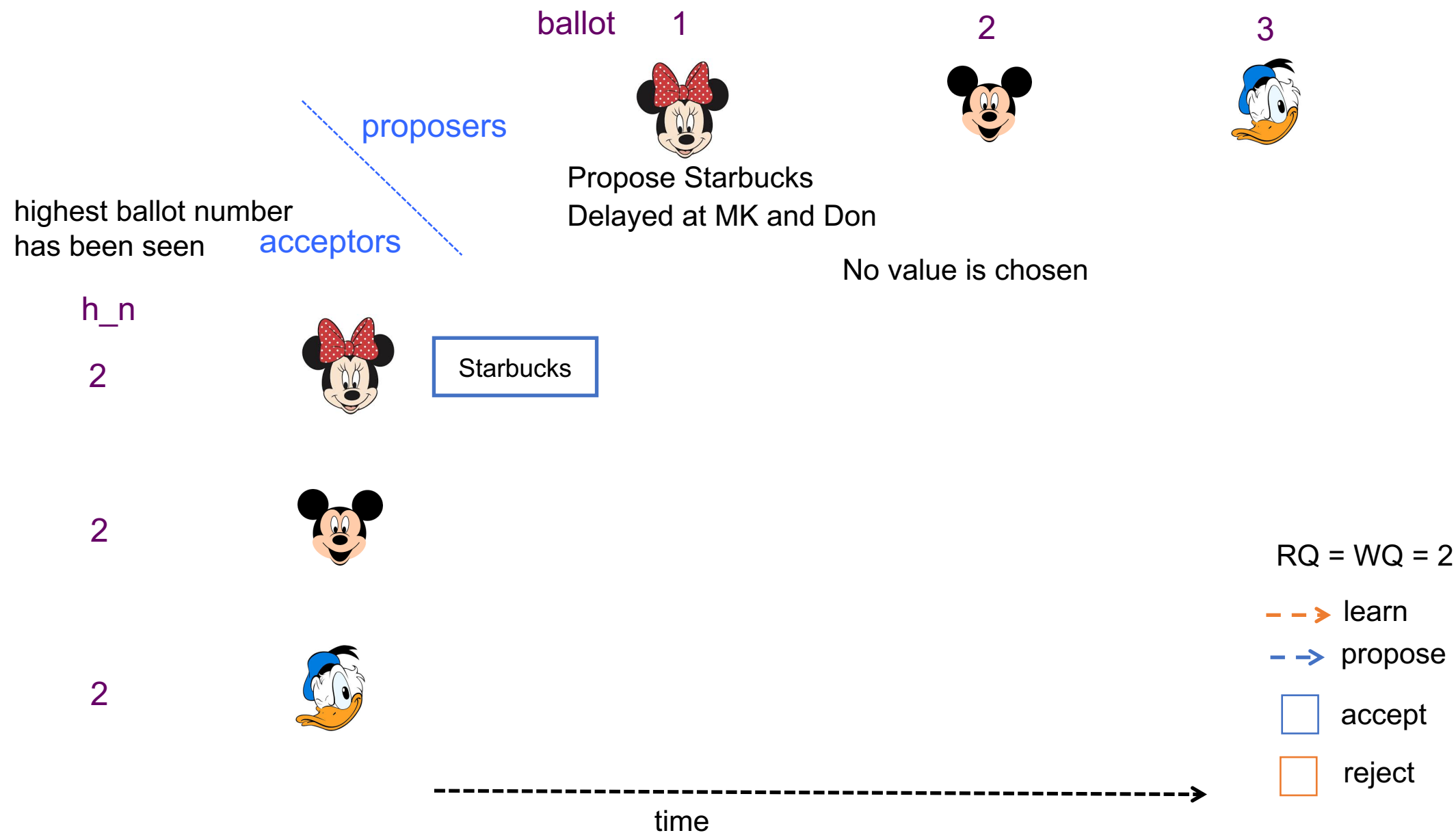
Phase 2 – Accept

- 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

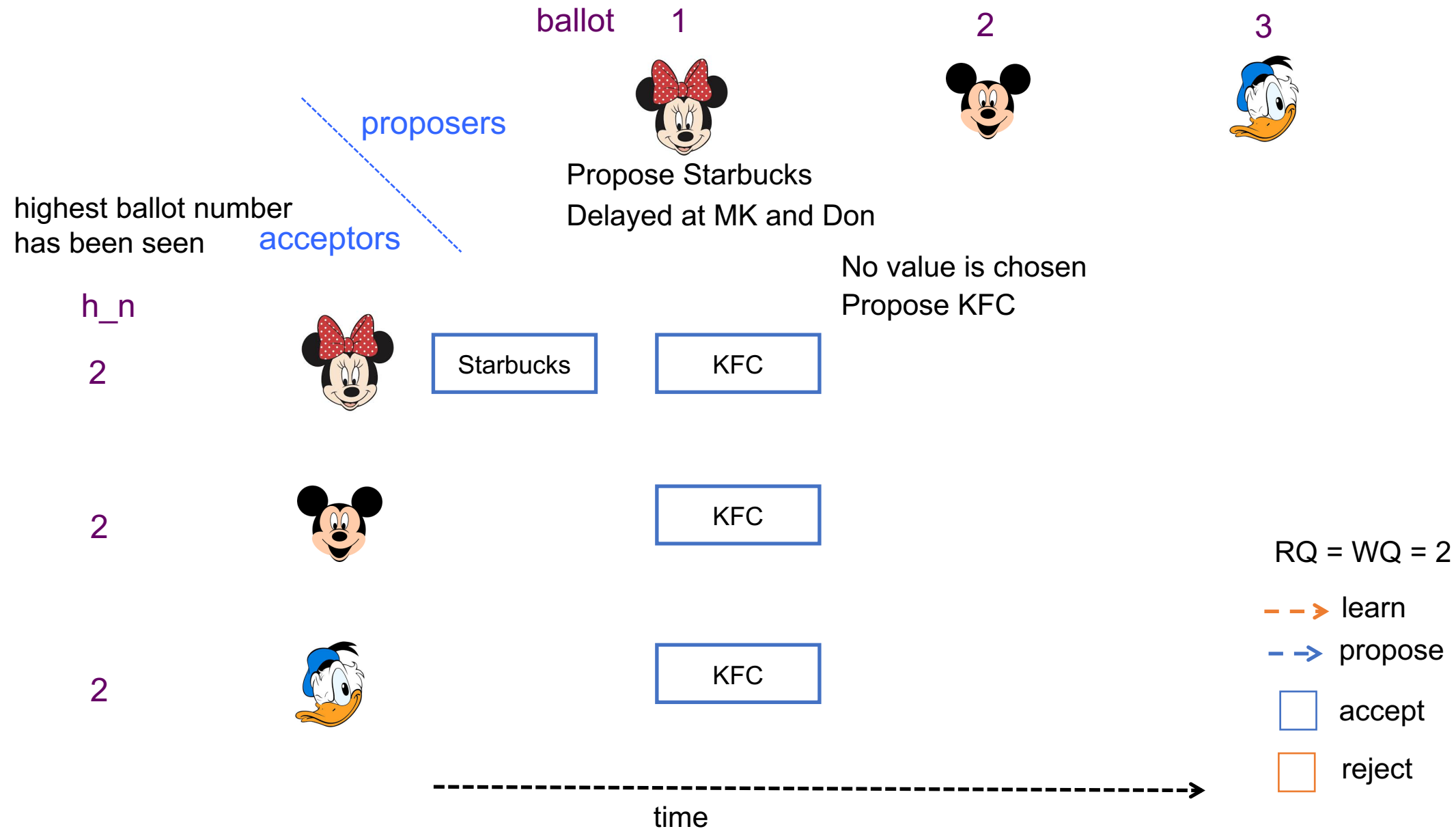
Asynchronous network delays/reorders packets



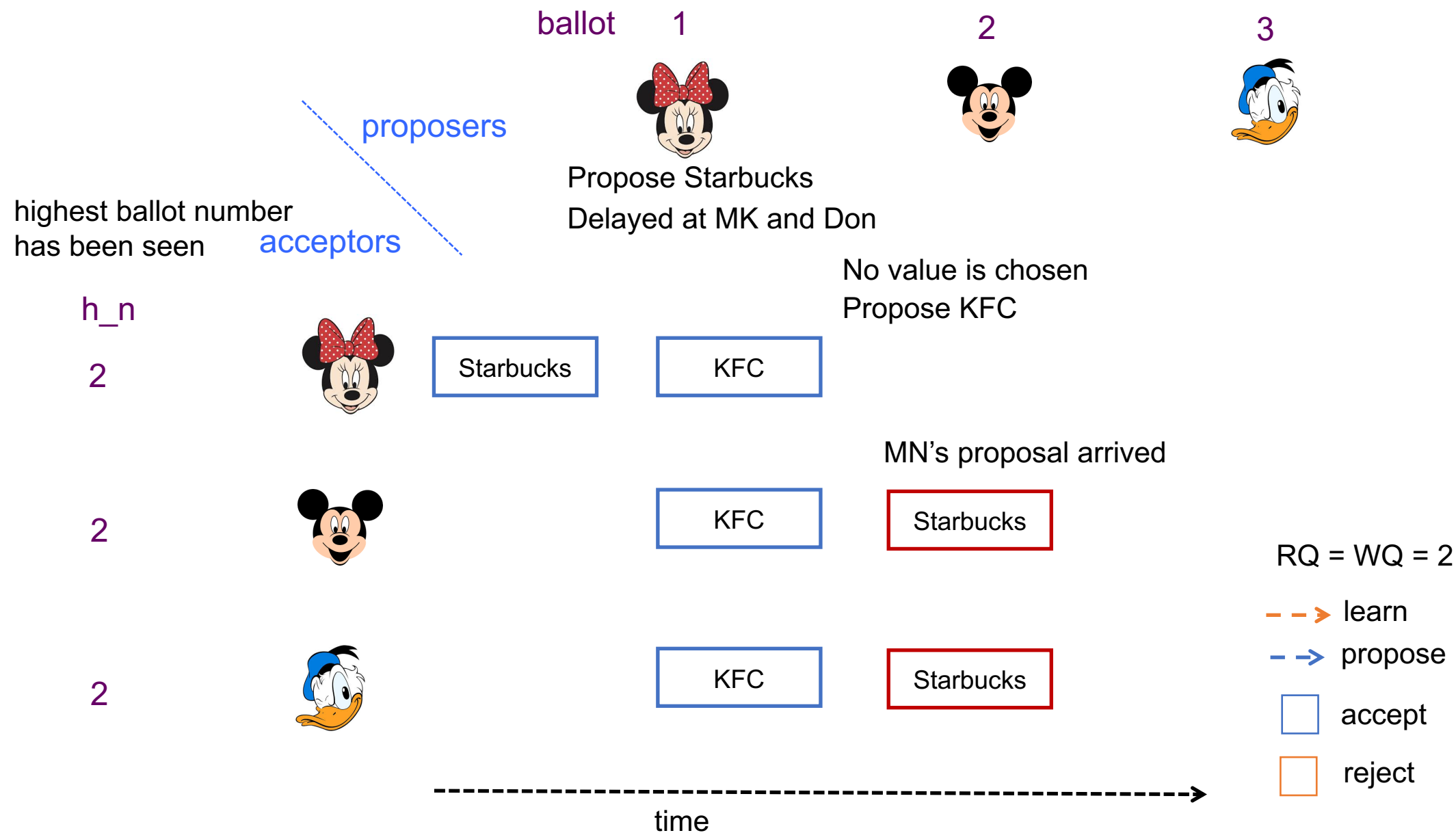
Asynchronous network delays/reorders packets



ballot	1	2	3
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	1
16	1	1	1
17	1	1	1
18	1	1	1
19	1	1	1
20	1	1	1
21	1	1	1
22	1	1	1
23	1	1	1
24	1	1	1
25	1	1	1
26	1	1	1
27	1	1	1
28	1	1	1
29	1	1	1
30	1	1	1
31	1	1	1
32	1	1	1
33	1	1	1
34	1	1	1
35	1	1	1
36	1	1	1
37	1	1	1
38	1	1	1
39	1	1	1
40	1	1	1
41	1	1	1
42	1	1	1
43	1	1	1
44	1	1	1
45	1	1	1
46	1	1	1
47	1	1	1
48	1	1	1
49	1	1	1
50	1	1	1
51	1	1	1
52	1	1	1
53	1	1	1
54	1	1	1
55	1	1	1
56	1	1	1
57	1	1	1
58	1	1	1
59	1	1	1
60	1	1	1
61	1	1	1
62	1	1	1
63	1	1	1
64	1	1	1
65	1	1	1
66	1	1	1
67	1	1	1
68	1	1	1
69	1	1	1
70	1	1	1
71	1	1	1
72	1	1	1
73	1	1	1
74	1	1	1
75	1	1	1
76	1	1	1
77	1	1	1
78	1	1	1
79	1	1	1
80	1	1	1
81	1	1	1
82	1	1	1
83	1	1	1
84	1	1	1
85	1	1	1
86	1	1	1
87	1	1	1
88	1	1	1
89	1	1	1
90	1	1	1
91	1	1	1
92	1	1	1
93	1	1	1
94	1	1	1
95	1	1	1
96	1	1	1
97	1	1	1
98	1	1	1
99	1	1	1
100	1	1	1



Asynchronous network delays/reorders packets

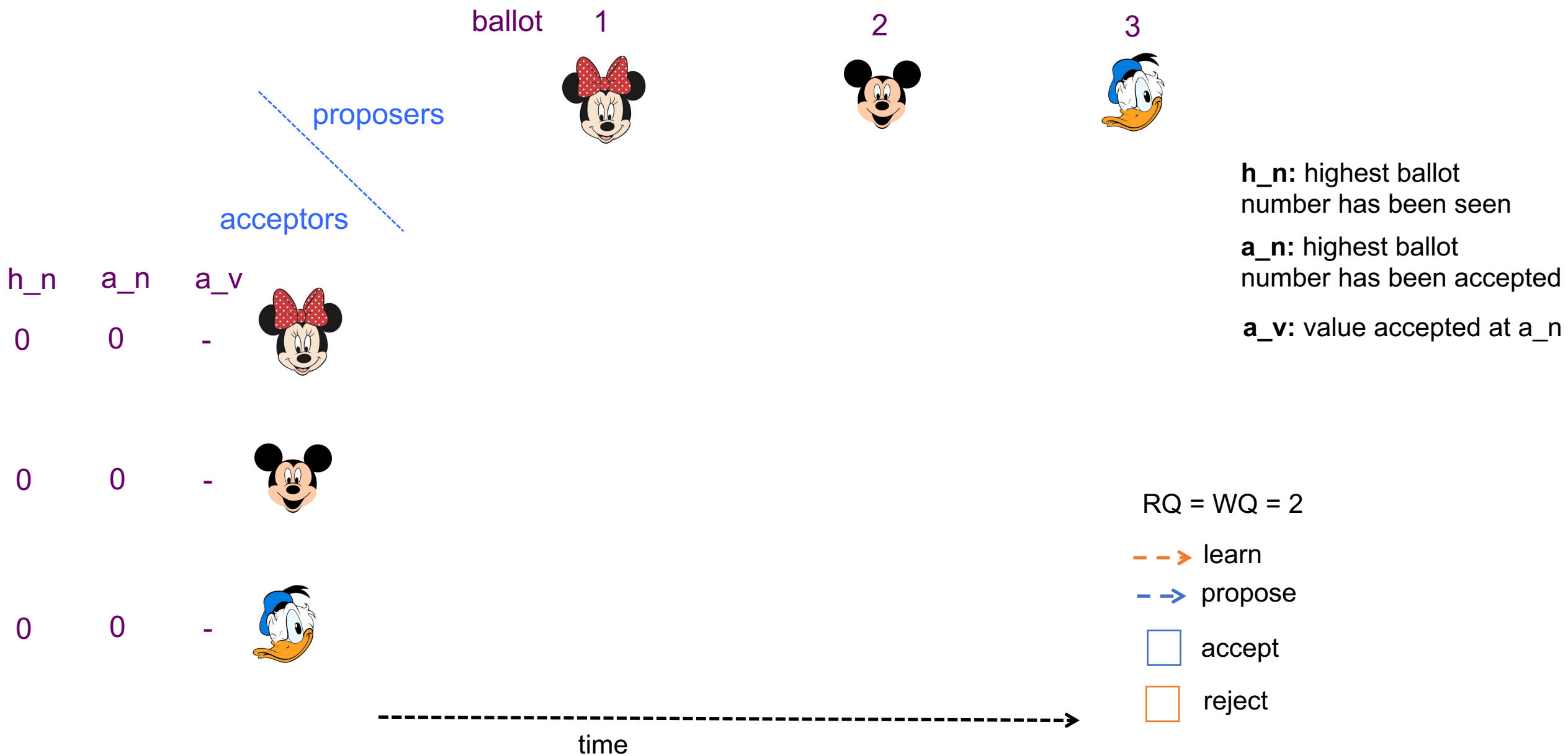


Find a safe value with a read quorum

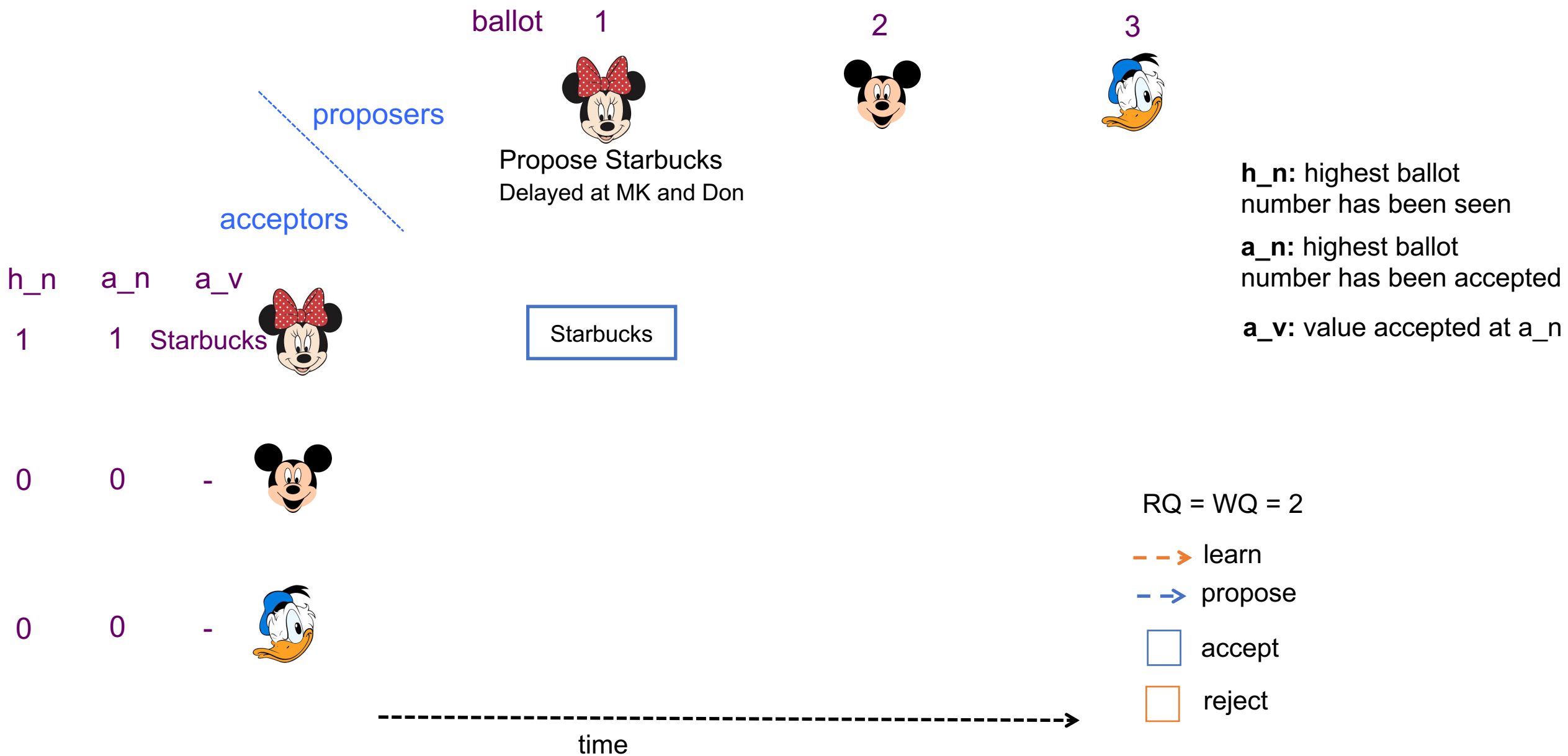
A value is safe if

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

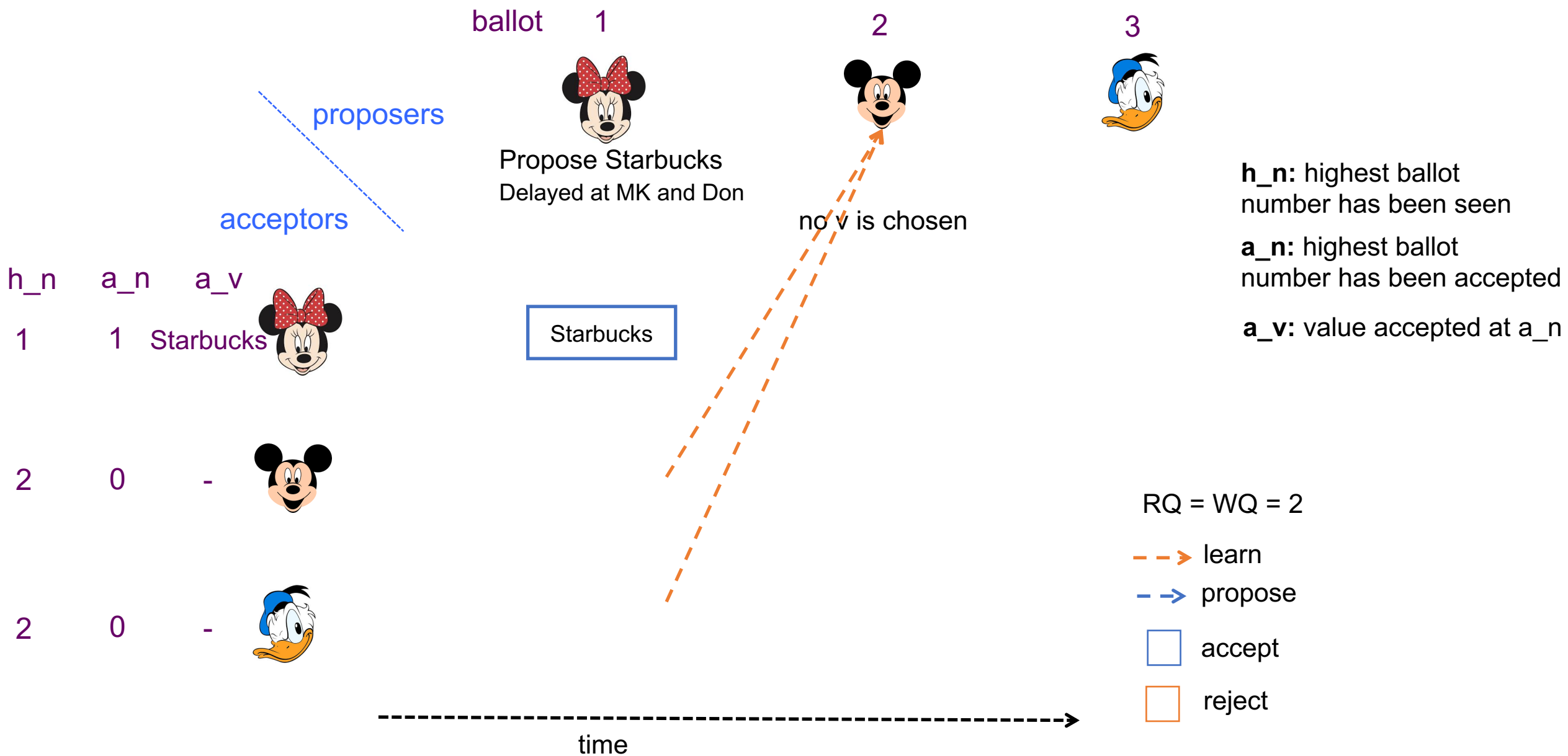
Asynchronous network delays/reorders packets



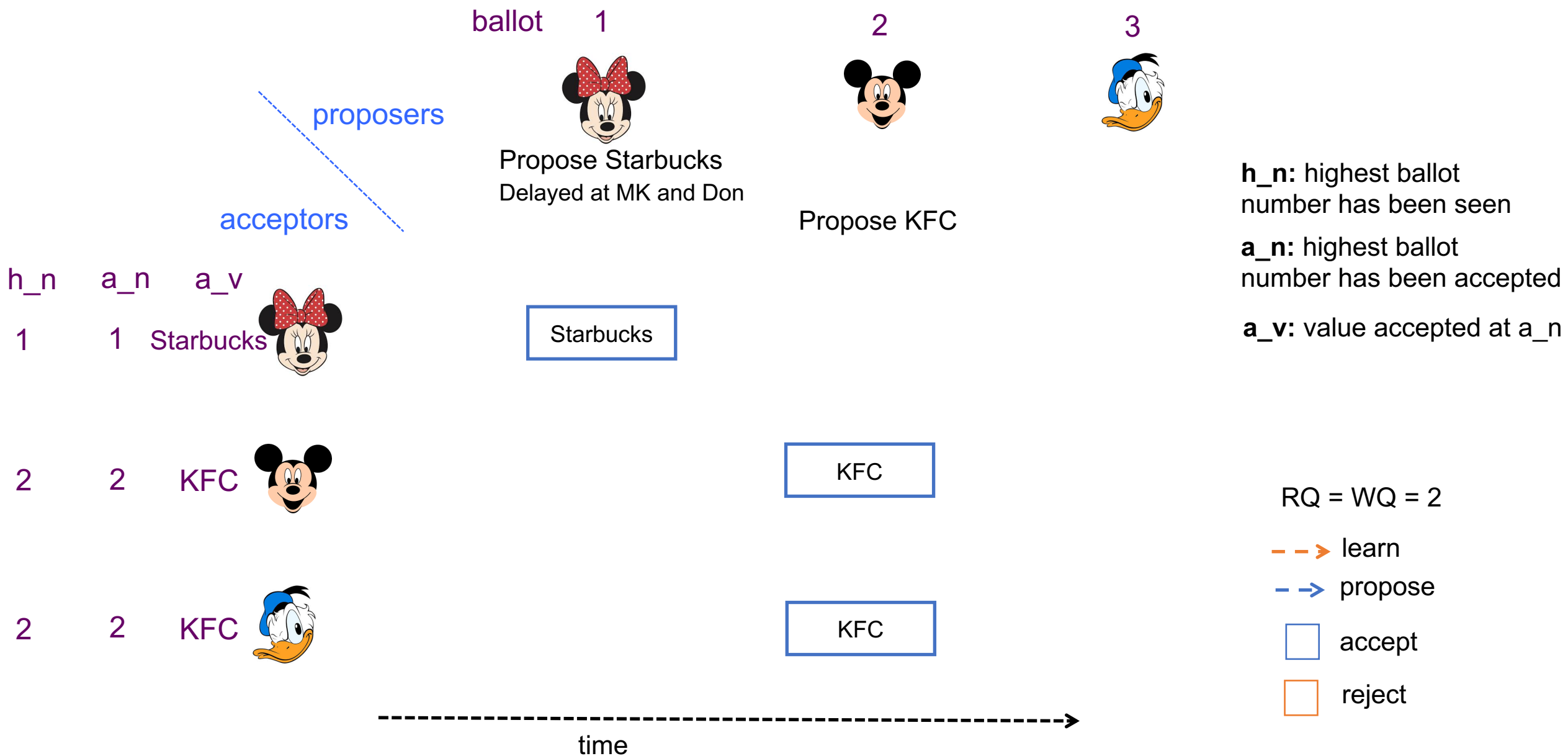
Asynchronous network delays/reorders packets



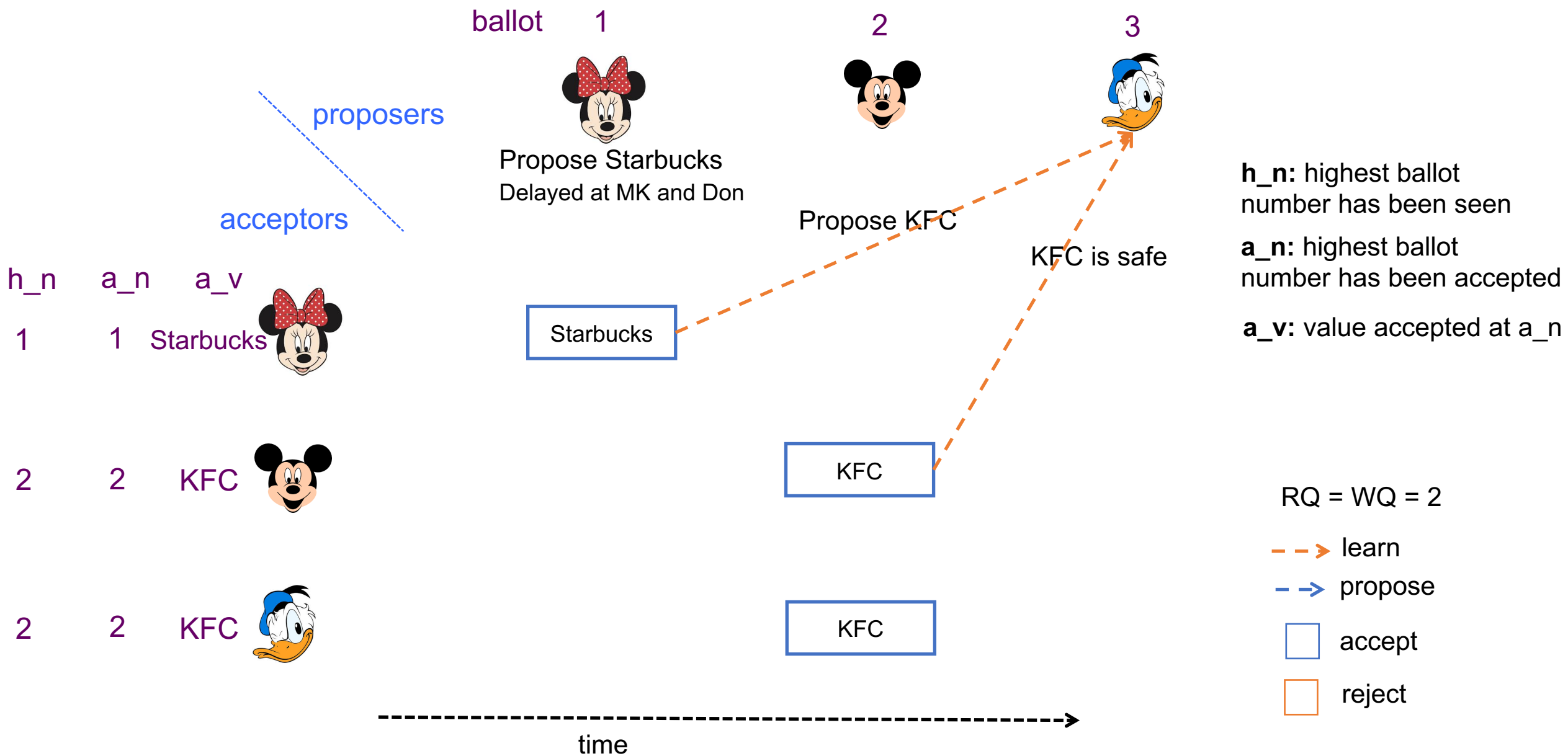
Asynchronous network delays/reorders packets



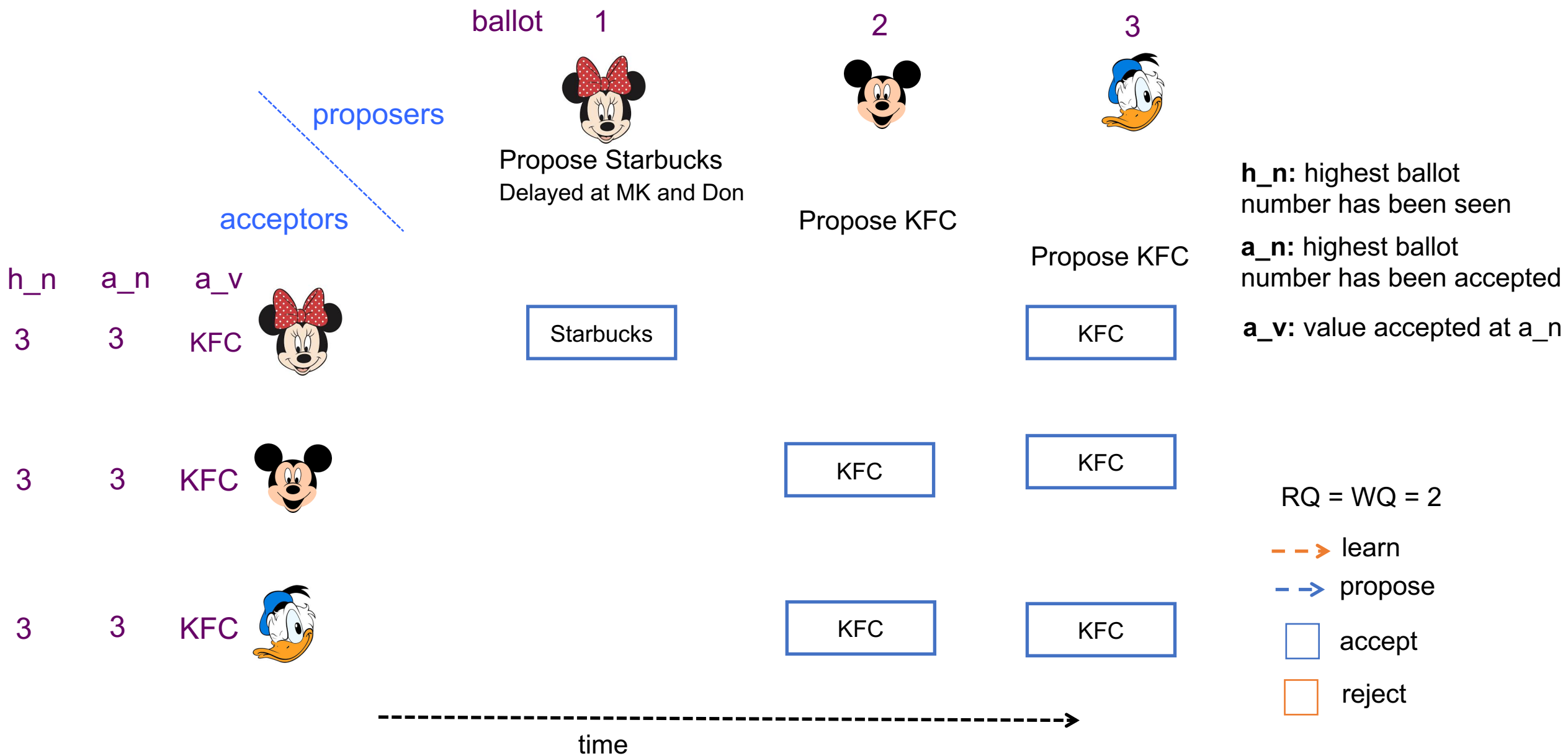
Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



Asynchronous network delays/reorders packets



Basic Algorithm



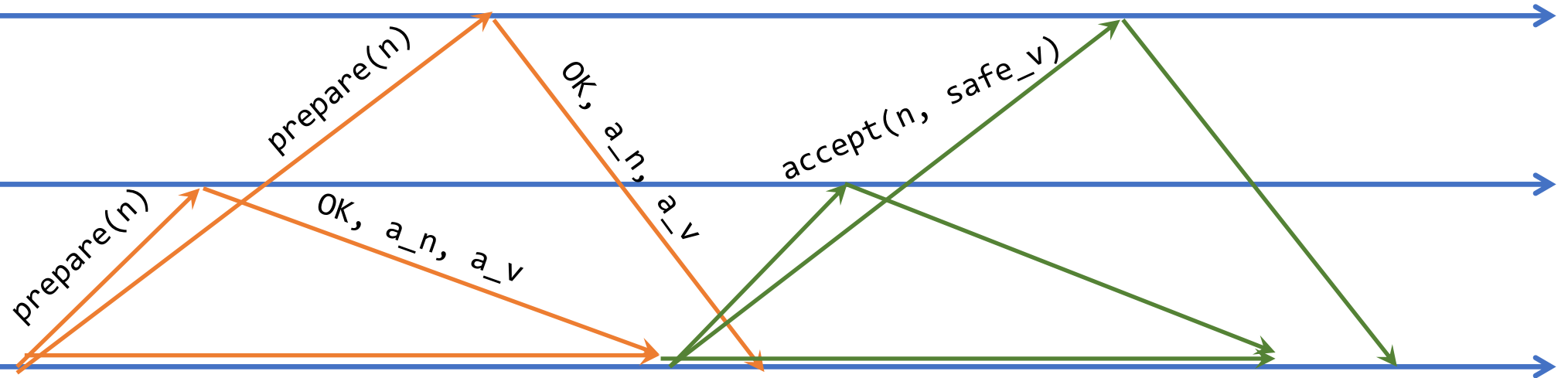
```
if n > h_n
  h_n = n
  return OK, a_n, a_v
else
  ignore
```

```
if n >= h_n
  h_n = n
  a_n = n
  a_v = safe_v
  return OK
else
  ignore
```

$n := \text{unique_ballot_number}()$

if receiving OK from RQ
safe_v = a_v with highest a_n
or safe_v = **any**

if receiving OK from WQ
safe_v is chosen

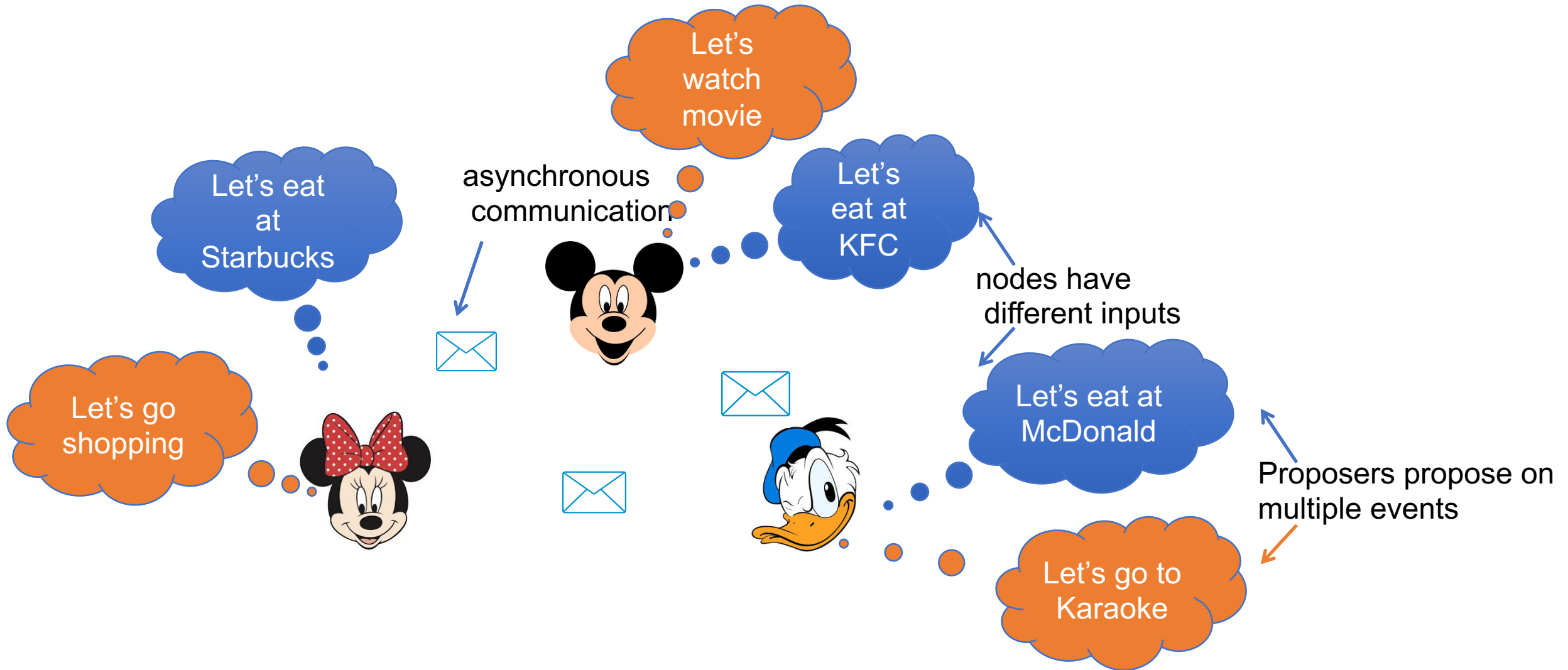


h_n – highest ballot number has been seen

a_n – highest ballot have been accepted

a_v – value accepted at a_n

Multi-Decree Consensus: from Paxos to MultiPaxos

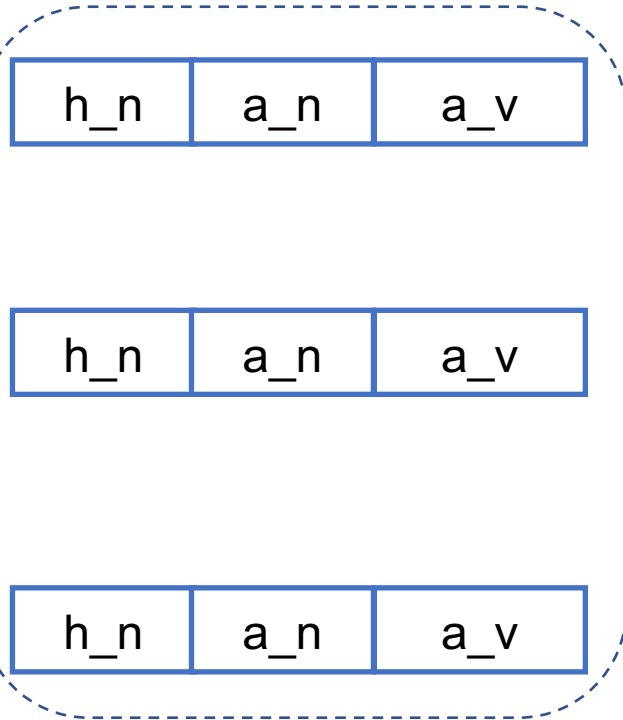


Multi-Decree Consensus: from Paxos to MultiPaxos

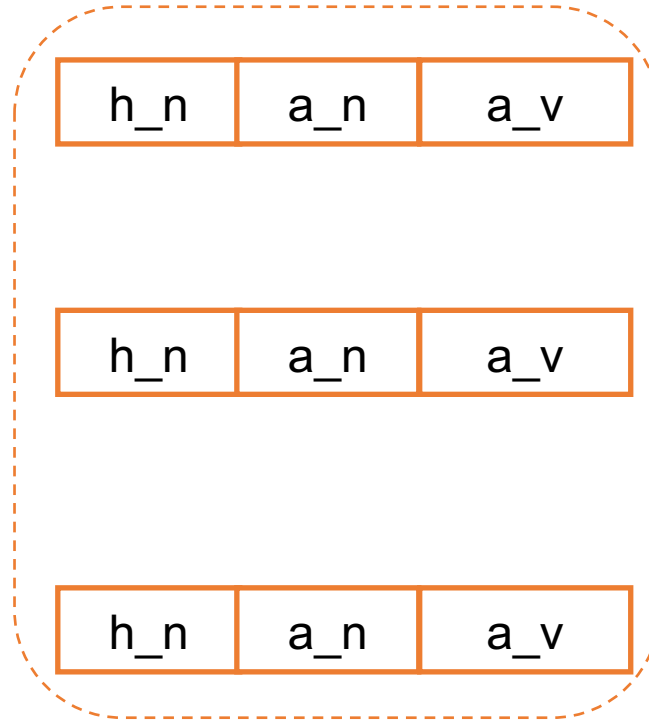
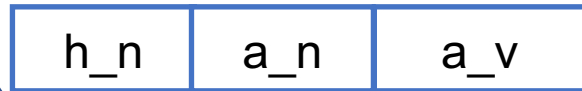
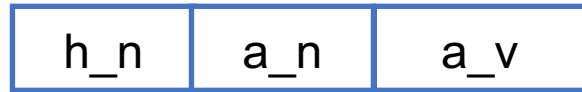
Events

Lunch

Entertainment



Single-decree Paxos



Single-decree Paxos

Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

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



1	0	-
---	---	---

h_n	a_n	a_v
-----	-----	-----



1	0	-
---	---	---

h_n	a_n	a_v
-----	-----	-----



1	0	-
---	---	---

h_n	a_n	a_v
-----	-----	-----

prepare

Single-decree Paxos

Single-decree Paxos

Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

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



1	0	-
---	---	---

h_n	a_n	a_v
-----	-----	-----



1	0	-
---	---	---

h_n	a_n	a_v
-----	-----	-----



1	0	-
---	---	---

h_n	a_n	a_v
-----	-----	-----

ok

Single-decree Paxos

Single-decree Paxos

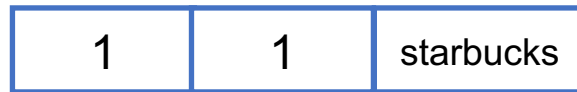
Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

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



accept

Single-decree Paxos

Single-decree Paxos

Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

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



1	1	starbucks
---	---	-----------

h_n	a_n	a_v
-----	-----	-----



1	1	starbucks
---	---	-----------

h_n	a_n	a_v
-----	-----	-----



1	1	starbucks
---	---	-----------

h_n	a_n	a_v
-----	-----	-----

ok

Single-decree Paxos

Single-decree Paxos

Multi-Decree Consensus: from Paxos to MultiPaxos

Events

Lunch

Entertainment

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



1	1	starbucks
---	---	-----------

h_n	a_n	a_v
-----	-----	-----



1	1	starbucks
---	---	-----------

h_n	a_n	a_v
-----	-----	-----



1	1	starbucks
---	---	-----------

h_n	a_n	a_v
-----	-----	-----

commit

Single-decree Paxos

Single-decree Paxos

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

3. Each node first has lunch then watches movie.



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

Log

0th entry

1st entry

Replicated State Machine

- Replicating a log of operations consistently



1	1	starbucks
---	---	-----------

1	1	movie
---	---	-------



1	1	starbucks
---	---	-----------

1	1	movie
---	---	-------



1	1	starbucks
---	---	-----------

1	1	movie
---	---	-------

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

Single-decree Paxos

Single-decree Paxos

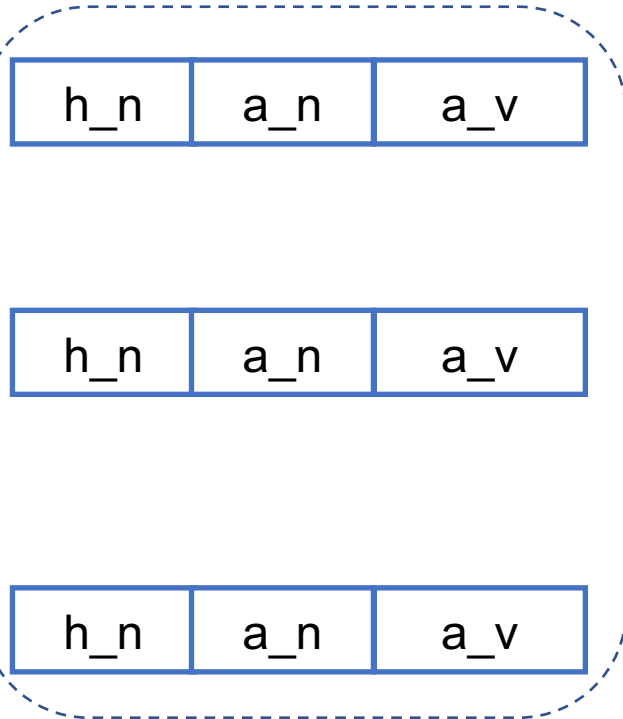
Multi-Decree Consensus: from Paxos to MultiPaxos

Log

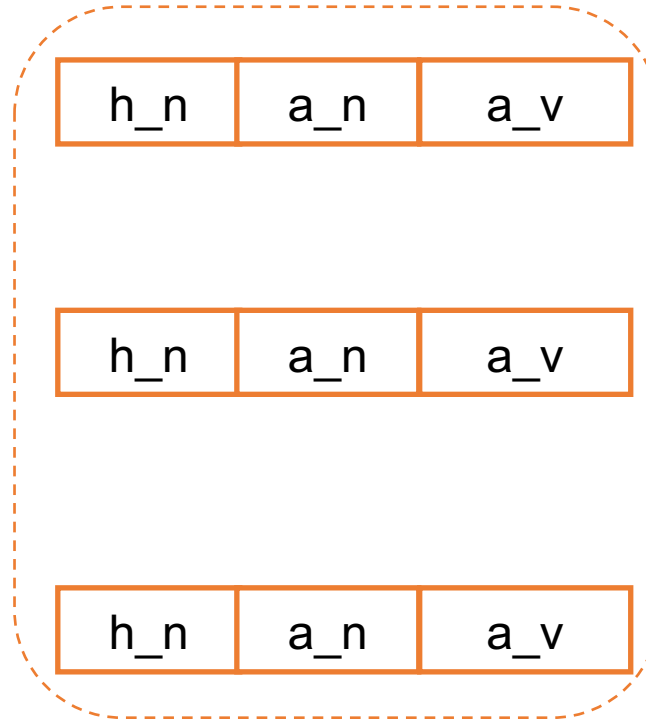
0th entry

1st entry

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



Single-decree Paxos



Single-decree Paxos



Multi-Decree Consensus: from Paxos to MultiPaxos

Log

h_n

0th entry

1st entry



0



0



0



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

Log

h_n

1

0

0

prepare(1)



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

Log

h_n



1



1

<OK, null>



1

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

Log

h_n



1



1

<OK, null>



1

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

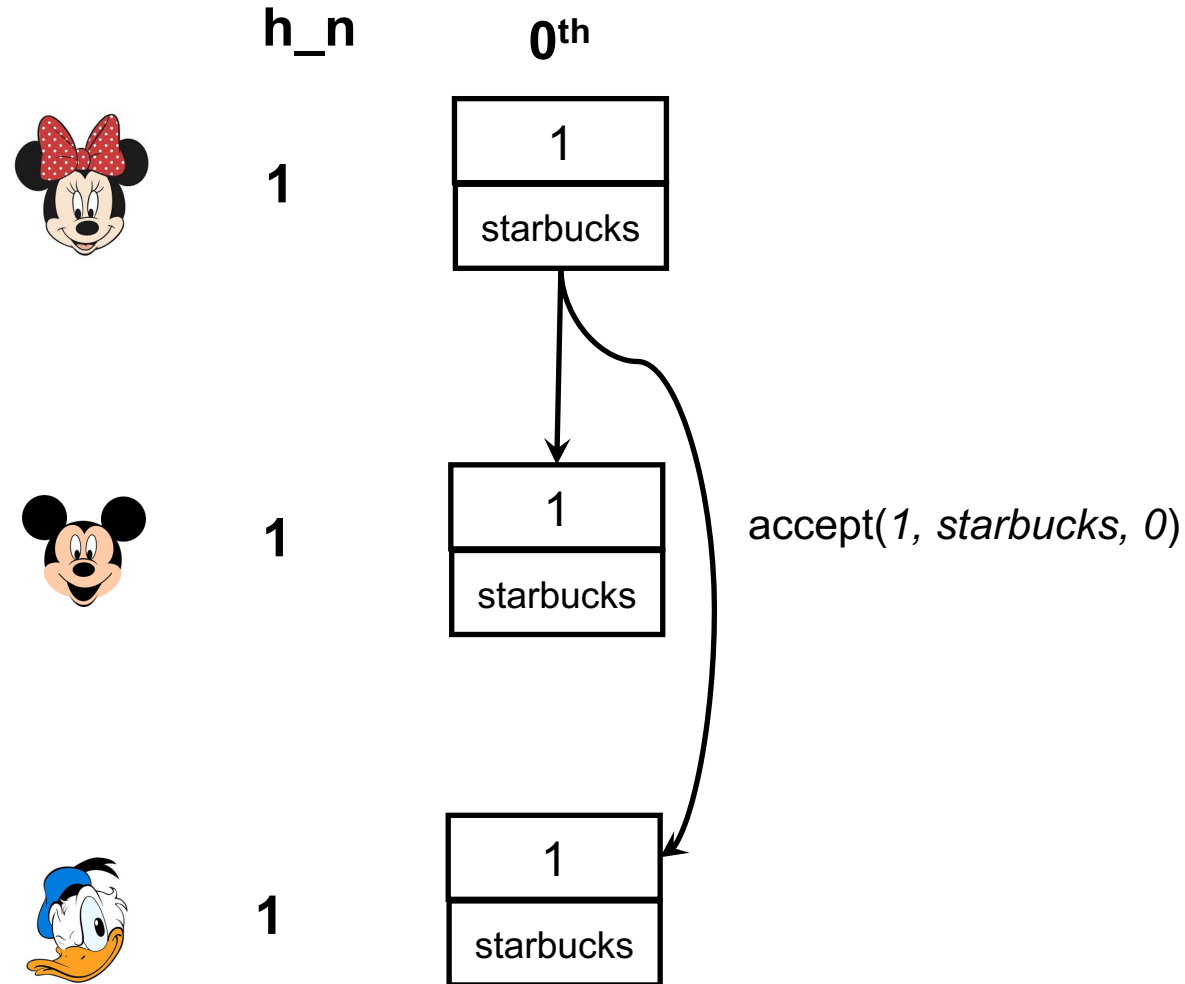


batch prepare

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 .

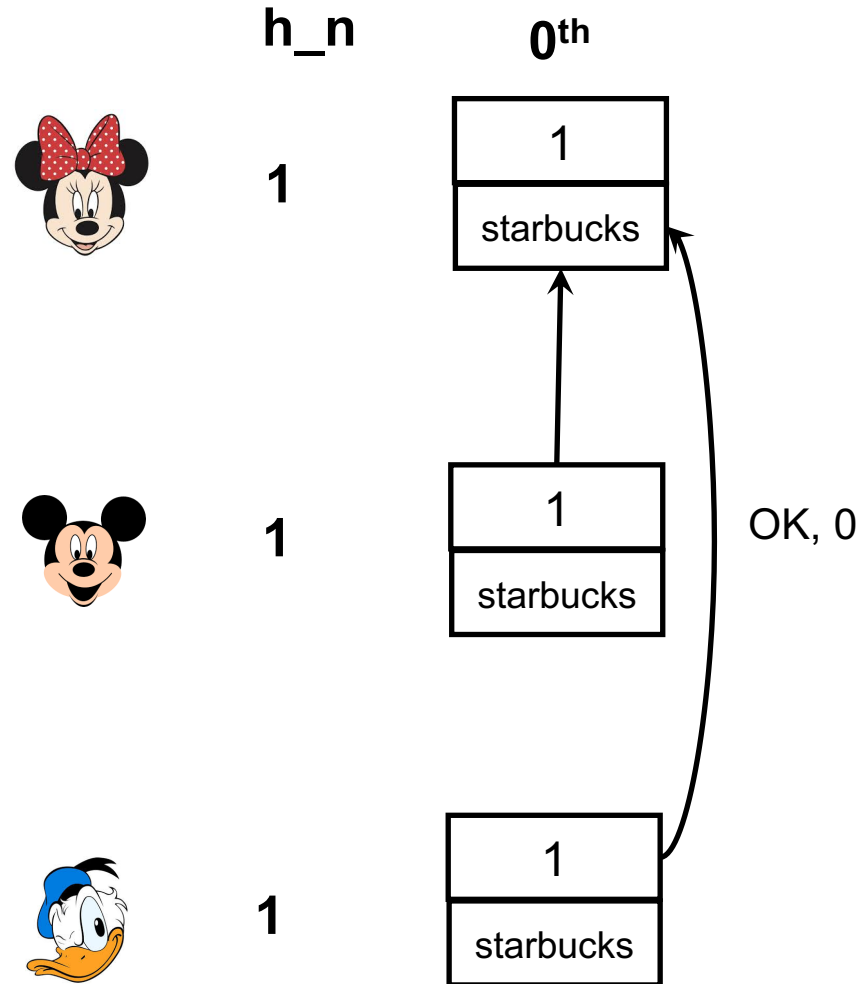


batch prepare

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

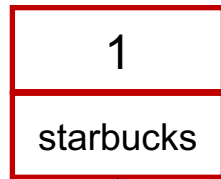
Log

h_n

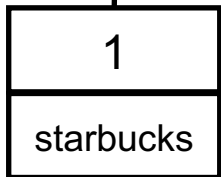


1

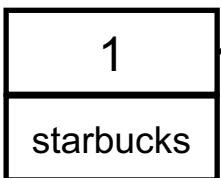
0th



1



1



OK, 0

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

0 th	1 st
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

Log

h_n



1

0 th	1 st
1	1
starbucks	Shopping



1

1	1
starbucks	Shopping



1

<table><tr><td>1</td></tr><tr><td>starbucks</td></tr></table>	1	starbucks
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

0 th	1 st
1	1
starbucks	Shopping



1

1	1
starbucks	Shopping



1

<table><tr><td>1</td></tr><tr><td>starbucks</td></tr></table>	1	starbucks
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

0 th	1 st	2 nd
1	1	1
starbucks	Shopping	Karaoke



1

0 th	1 st
1	1
starbucks	Shopping



1

0 th	1 st
1	1
starbucks	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



1

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

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.



1

1	1
starbucks	Shopping



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

Step 2.2. each acceptor replies OK



1

<table><tr><td>1</td></tr><tr><td>starbucks</td></tr></table>	1	starbucks
1		
starbucks		



<table><tr><td>1</td></tr><tr><td>Karaoke</td></tr></table>	1	Karaoke
1		
Karaoke		



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

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



2

1	1
starbucks	Shopping

prepare(2)



1

1	1
starbucks	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



1

0th

1st

2nd

3rd

1	1	1	1
starbucks	Shopping	Karaoke	Per Se

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.



2

1	1
starbucks	Shopping



log entry of 0 and 2



1

1
starbucks



1
Karaoke



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

2nd

3rd

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

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.



2

0 th	1 st	2 nd
1	1	1
starbucks	Shopping	Karaoke



log entry of 0 and 2



2

0 th	1 st	2 nd
1		1
starbucks		Karaoke



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

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

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.



2

1	1	1
starbucks	Shopping	Karaoke



accept 1 and 2 again



2

1
starbucks



1
Karaoke



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

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

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.



2

1	2	2
starbucks	Shopping	Karaoke



accept 1 and 2 again



2

1	2	2
starbucks	Shopping	Karaoke



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

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



2

1	2	2
starbucks	Shopping	Karaoke



2

1	2	2
starbucks	Shopping	Karaoke

OK

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

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



2

1	2	2	2
starbucks	Shopping	Karaoke	McDonald



2

1	2	2
starbucks	Shopping	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



1

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



2

1	2	2	2
starbucks	Shopping	Karaoke	McDonald



2

1	2	2	2
starbucks	Shopping	Karaoke	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



2

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

1	2	2	2
starbucks	Shopping	Karaoke	McDonald

1	2	2	2
starbucks	Shopping	Karaoke	McDonald

2

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

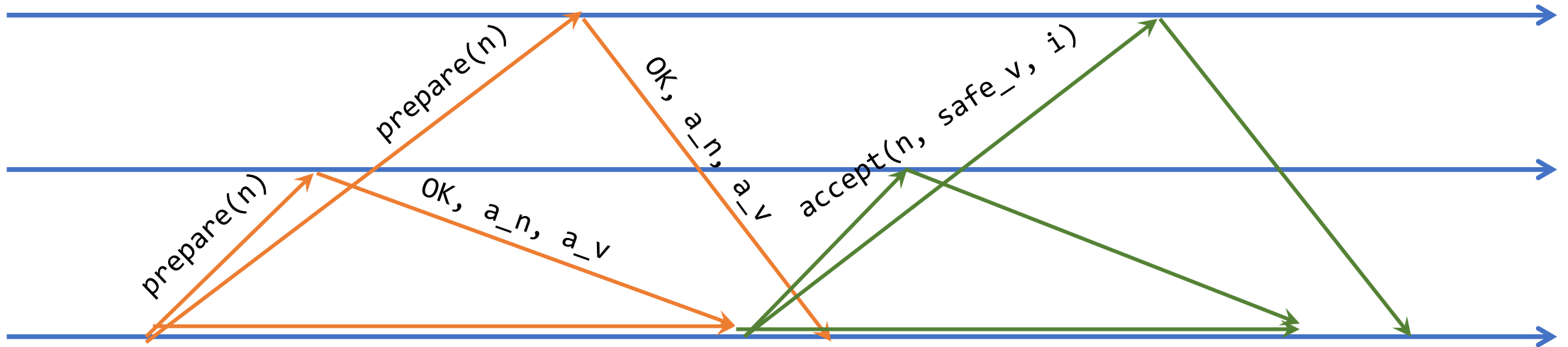
Multi-Paxos Algorithm

```

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

```

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
    
```



$n := \text{unique_ballot_number}()$

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

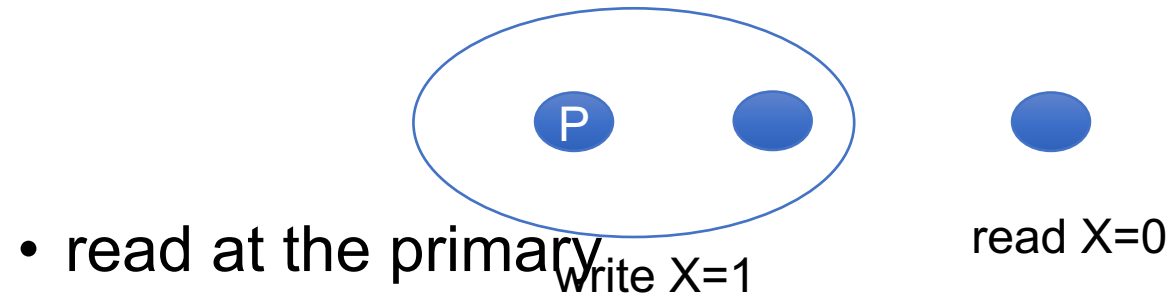
1	S1	<table><tr><td>OP1</td></tr><tr><td>1</td></tr></table>	OP1	1
OP1				
1				
1	S2	<table><tr><td>OP1</td></tr><tr><td>1</td></tr></table>	OP1	1
OP1				
1				
1	S3	<table><tr><td>OP1</td></tr><tr><td>1</td></tr></table>	OP1	1
OP1				
1				
1	S4	<table><tr><td>OP1</td></tr><tr><td>1</td></tr></table>	OP1	1
OP1				
1				
1	S5	<table><tr><td>OP1</td></tr><tr><td>1</td></tr></table>	OP1	1
OP1				
1				

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

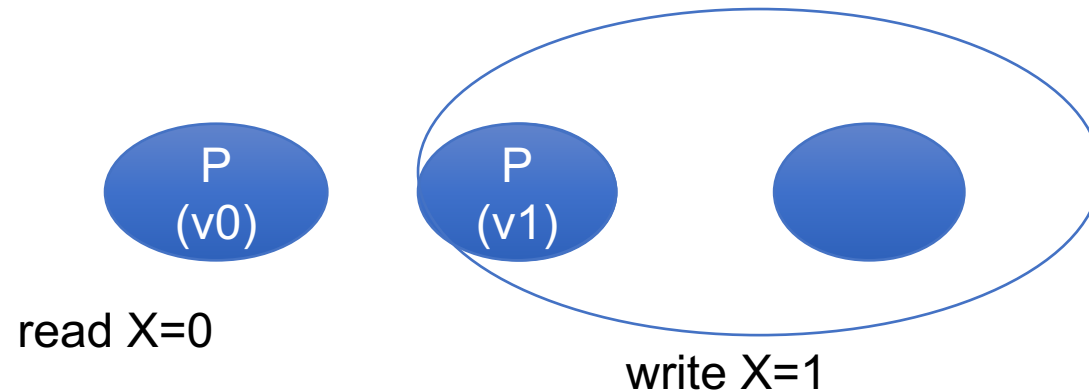
1. Network Partitions S1, S2 from Others
2. S1 receives OP2, OP3, OP4
3. OP5 and OP6 got chosen
4. S2 receives OP2, OP3, OP4 from S1
5. S3 crashes
6. Network heals
7. S1 tries to commit OP2, OP3, OP4
8. OP7 got chosen
9. S4 crashes
10. S3 recovers

How about this optimization

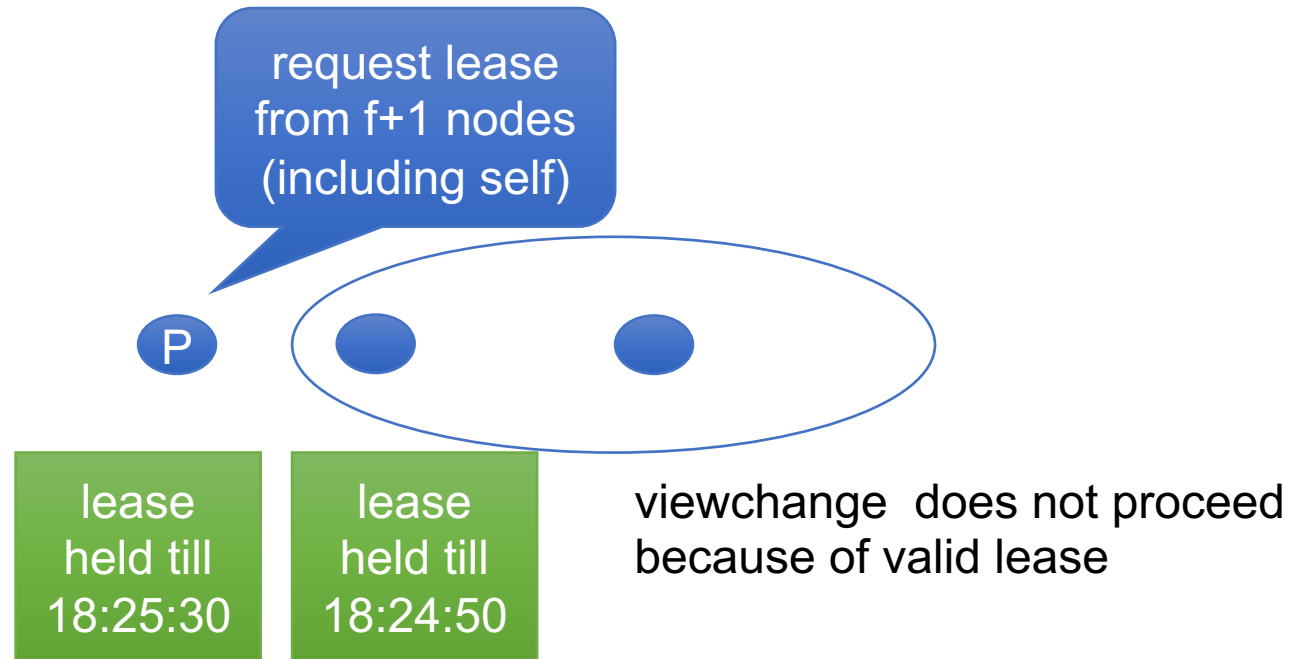
- Viewstamp with read optimization.
 - read at any arbitrary server



- 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