Linearizability

Zhaoguo Wang

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

CSE – Distributed System Part

Part I: Fault Tolerance

Part II: Consistency

- Linearizability
- Paxos

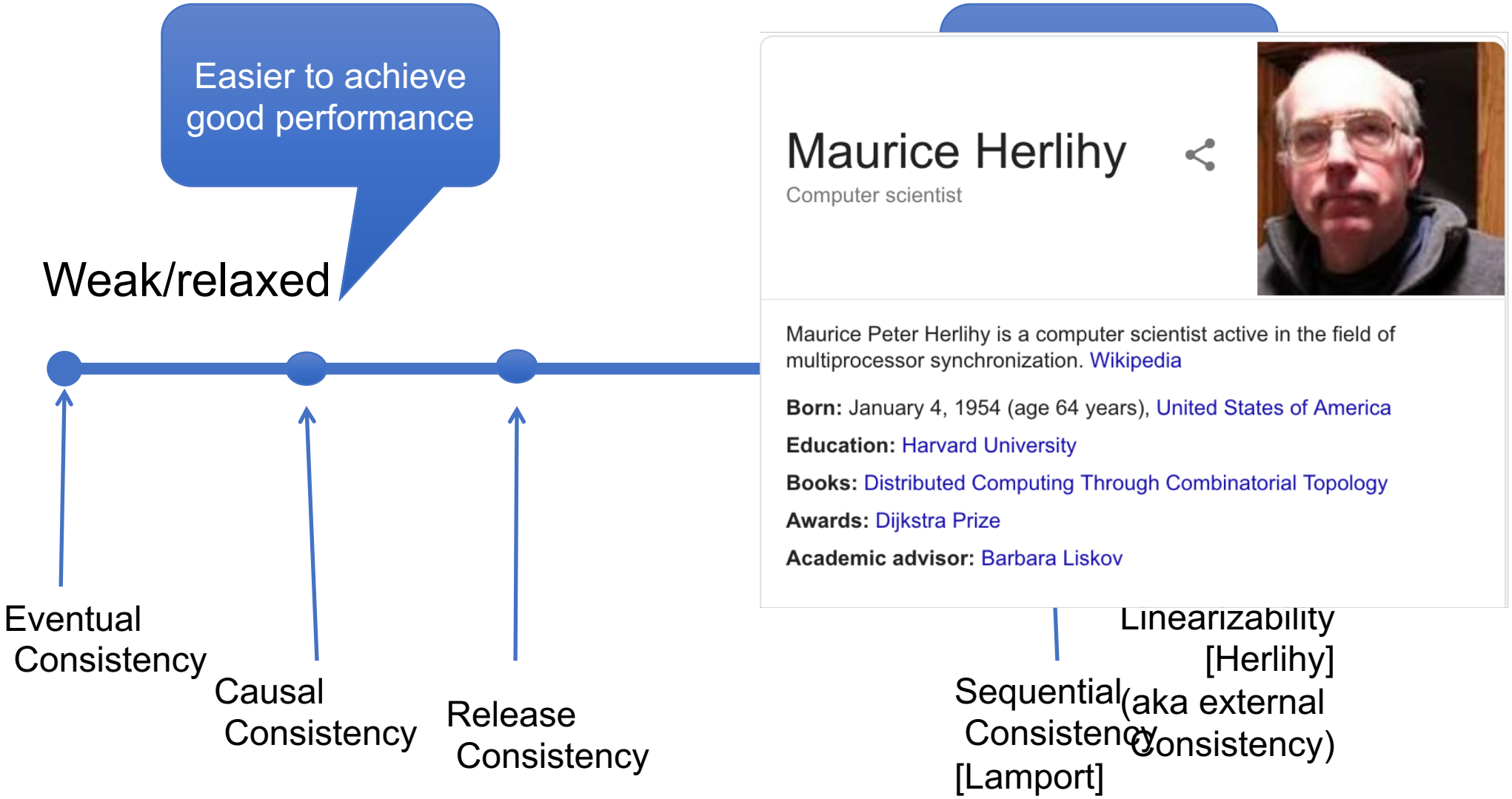
Part III: Distributed Transaction

Consistency model

Consistency model = meaning of operations in the face of concurrency and failure

- A contract between system designers and application programmers
- A set of rules dictating what allowed system behaviors are

Spectrum of Consistency Models



Consistency models are abstract

- Independent from any implementation
- Why specifying an abstract consistency model?
  - Application programmers do not have to understand the system implementation
  - System developers can reason correctness of implementation & optimizations
    - add caching
    - add replication
    - shard data across machines etc.

Application programmers' expectation

Thread-1
(running on client machine 1)

```
Put("x", 1);
Put("y", 1);
```

Thread-2
(running on client machine 2)

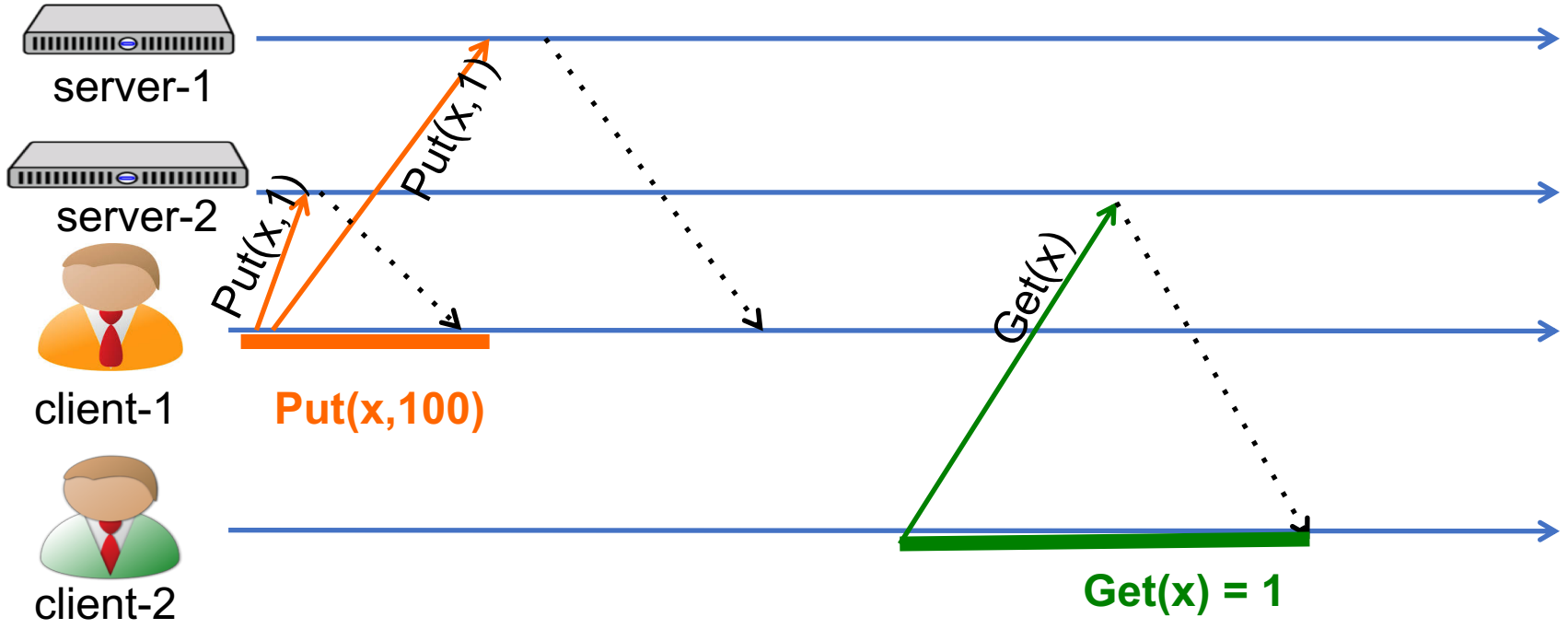
```
Get("Y")=?;
Get("X")=?;
```

What are expected values for Gets?

- Put(x,1), Put(y,1), Get(y)=1,Get(x)=1
- Put(x,1),Get(y)=0,Put(y,1), Get(x)=1
- ...
- Get(y)=0,Get(x)=0,Put(x,1),Put(y,1)

- 6 potential interleavings:
- 3 potential readings: (x=0,y=0) or (x=1 y=1) or (x=1 y=0)

A strawman key-value store

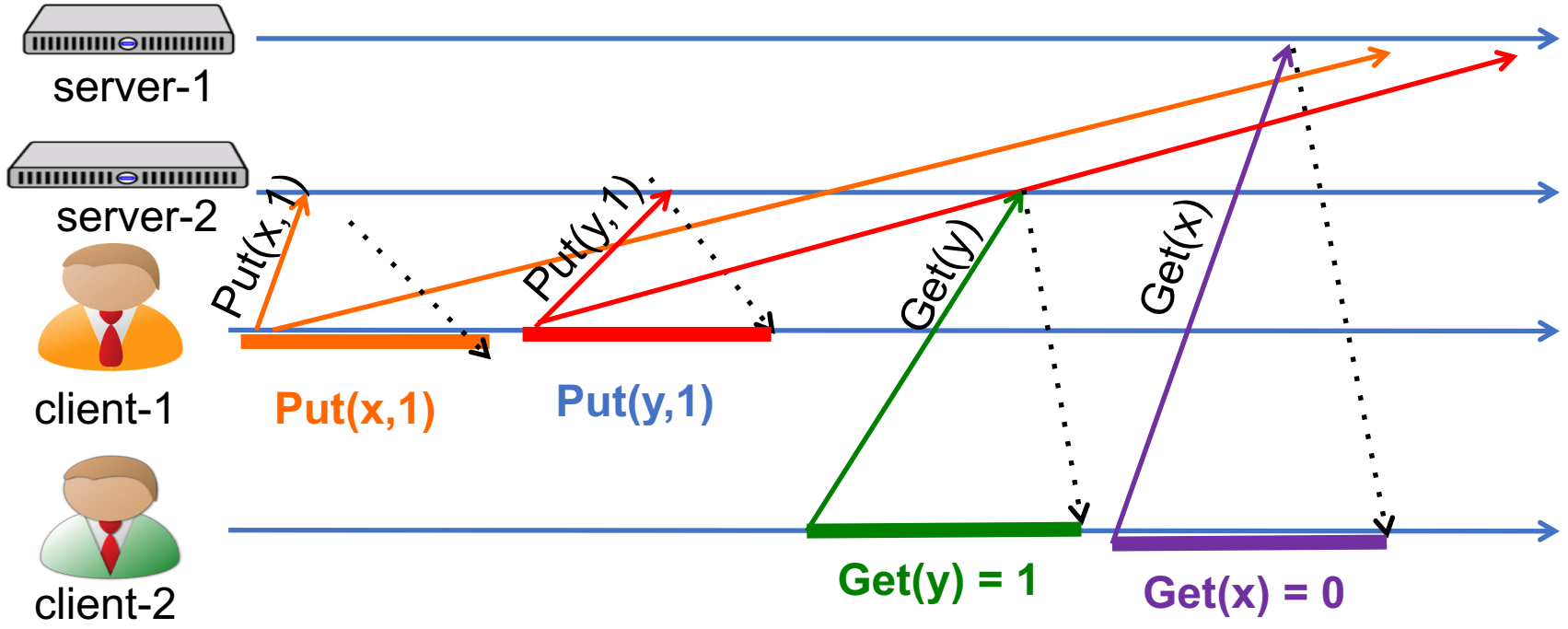


- System implementation:
  - Clients send writes to both replicas in parallel, wait for the fastest reply
  - Clients read from either replica

Does strawman satisfies programmers' expectation?

- Programmers' expectation: 3 read values
  - (x=0,y=0), or (x=1, y=1), or (x=1, y=0)
- Is the expectation fulfilled w/ a single server replica?
- Is the expectation fulfilled w/ 2 replicas?
  - Is x=0,y=1 possible under strawman?

A strawman key-value store



- System implementation:
  - Clients send writes to both replicas in parallel, wait for the fastest reply
  - Clients read from either replica

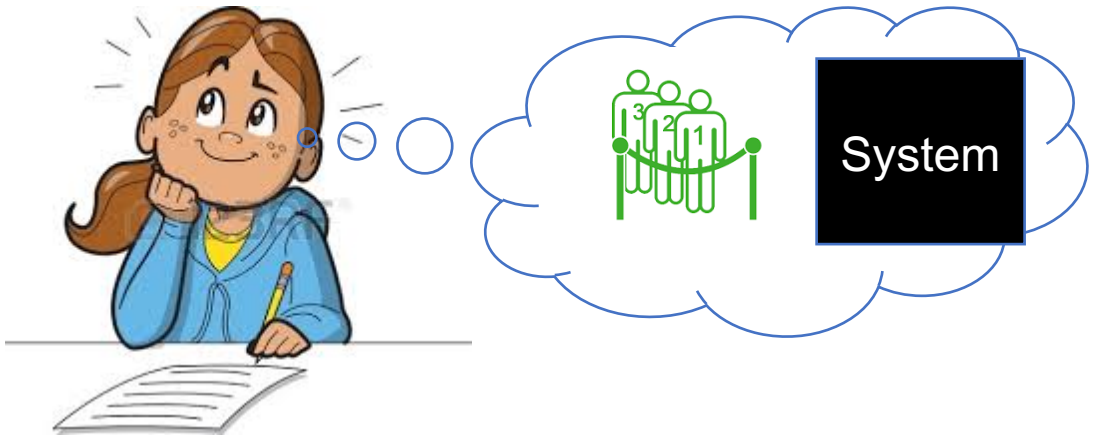
How to define a strong consistency model?

Reality

- Concurrent execution
- replicated data

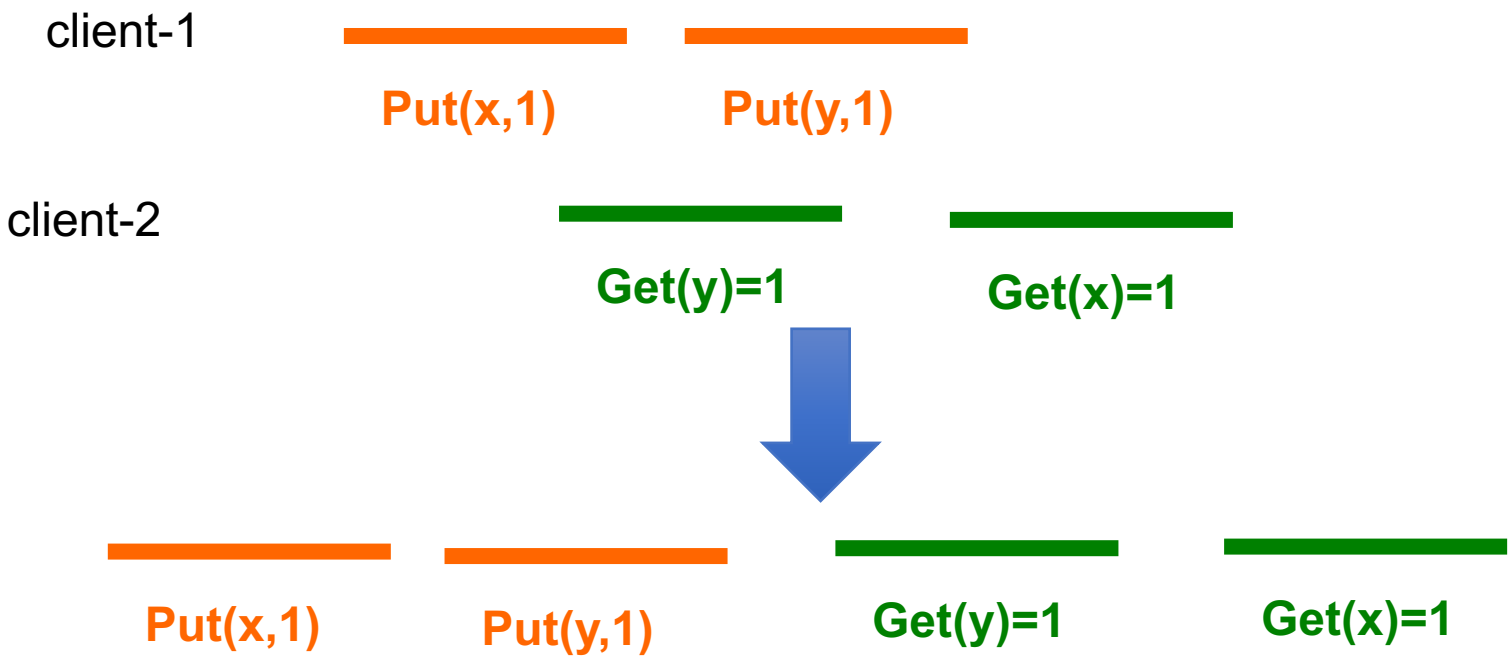
Simple abstract mental model

- “Sequential” behavior
- “One-copy” of data



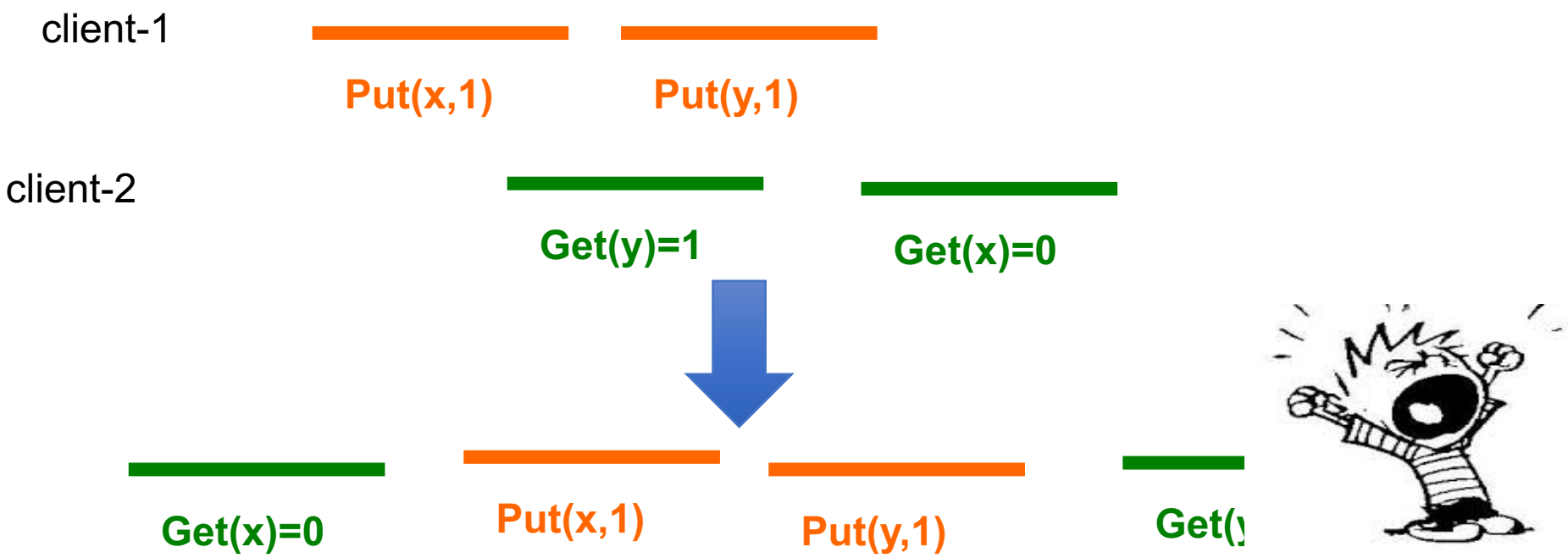
How to define a strong consistency model?

Reduce a concurrent history to an equivalent serial execution history



How to define a strong consistency model?

- Not all equivalent serial execution history “make sense”



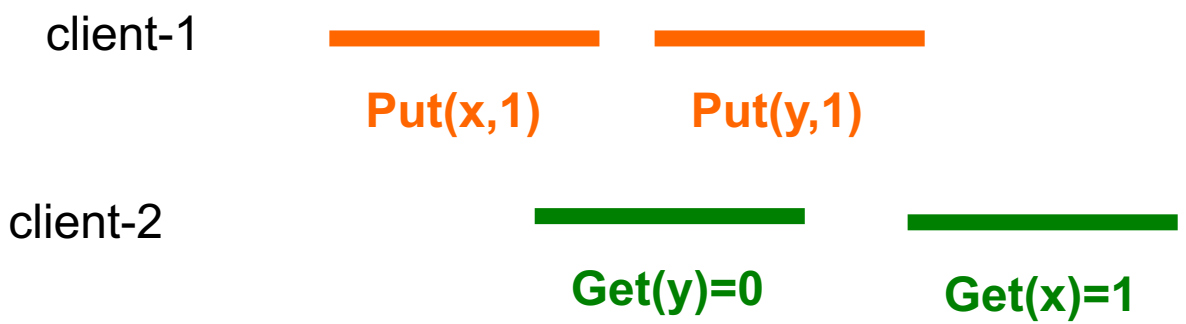
What are allowed equivalent serial history?

Certain orders in the original execution should be preserved in the equivalent history

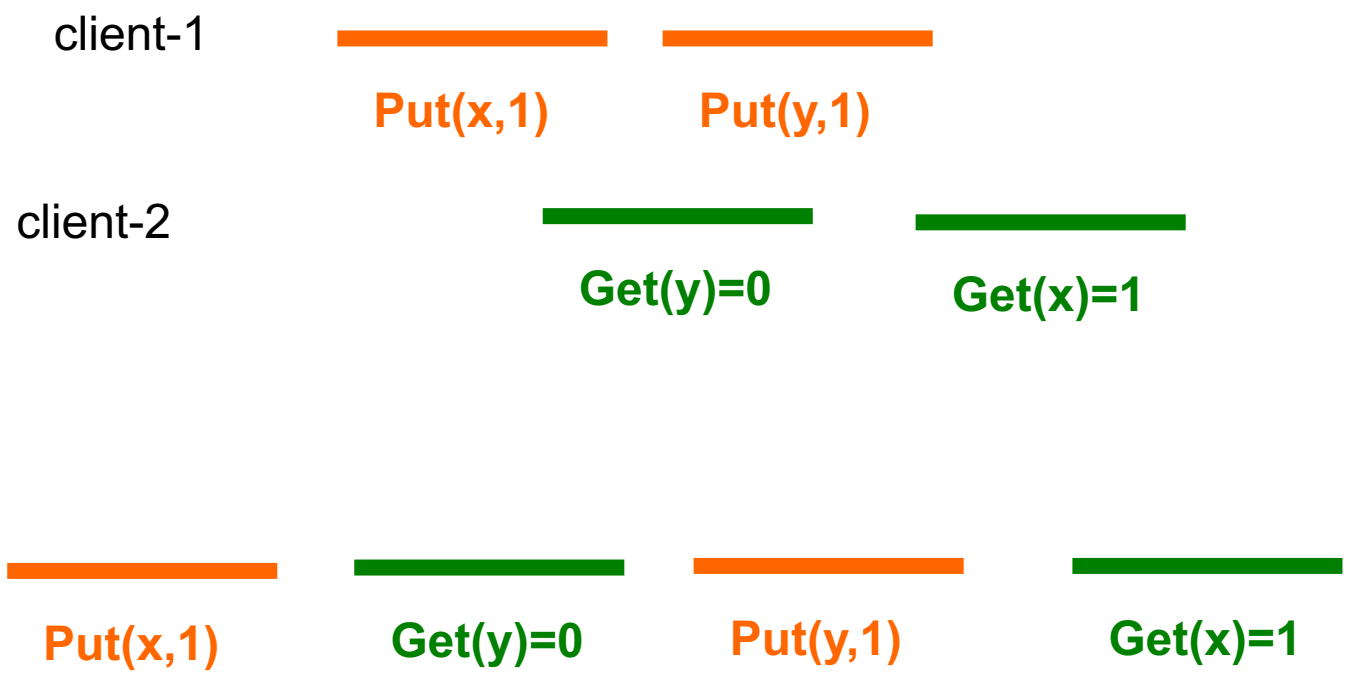
What are possibilities?

1. Global op issue order
2. Global op completion order
3. per-thread issue (and completion) order
4. global “completion-to-issue” order

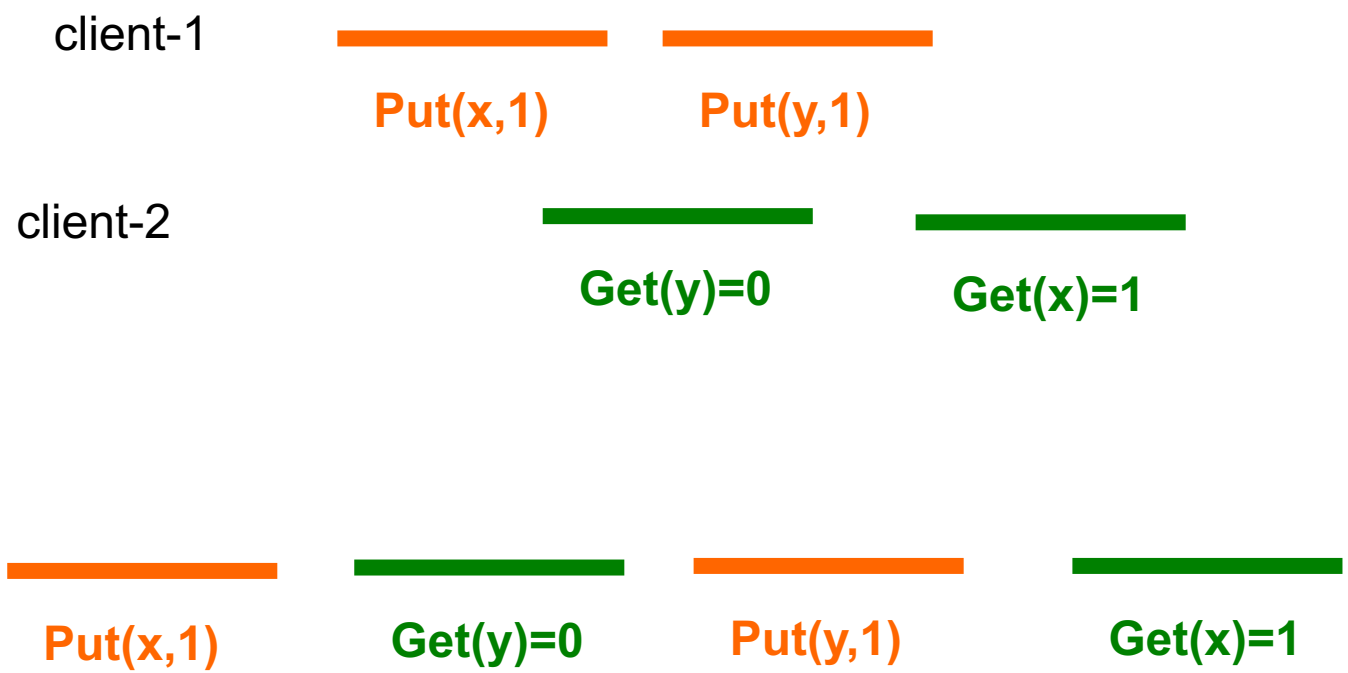
If op1 finishes before op2 starts, then op1 → op2 in equivalent history



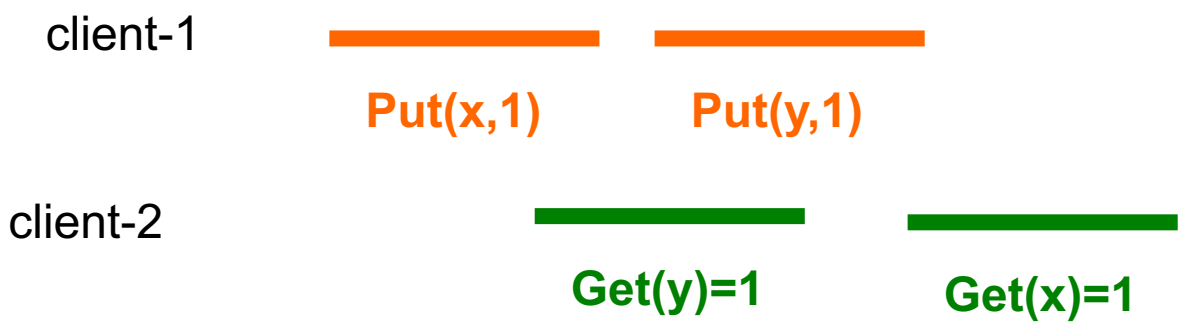
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



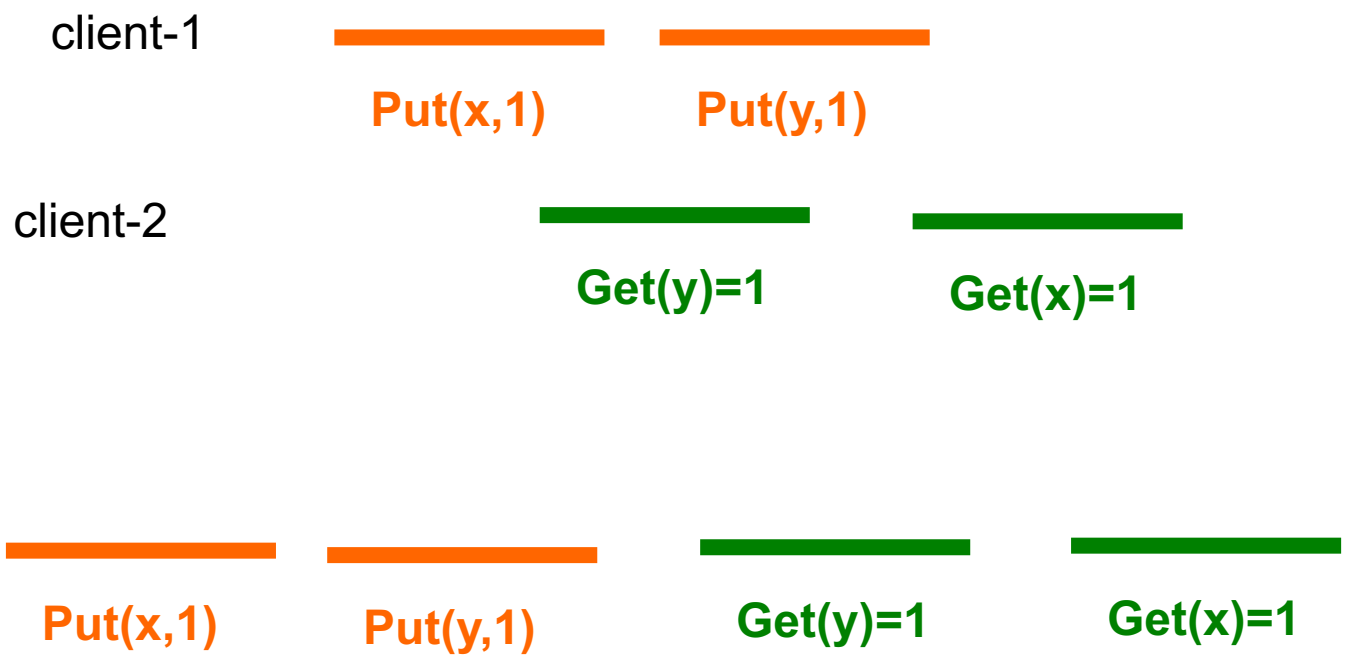
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



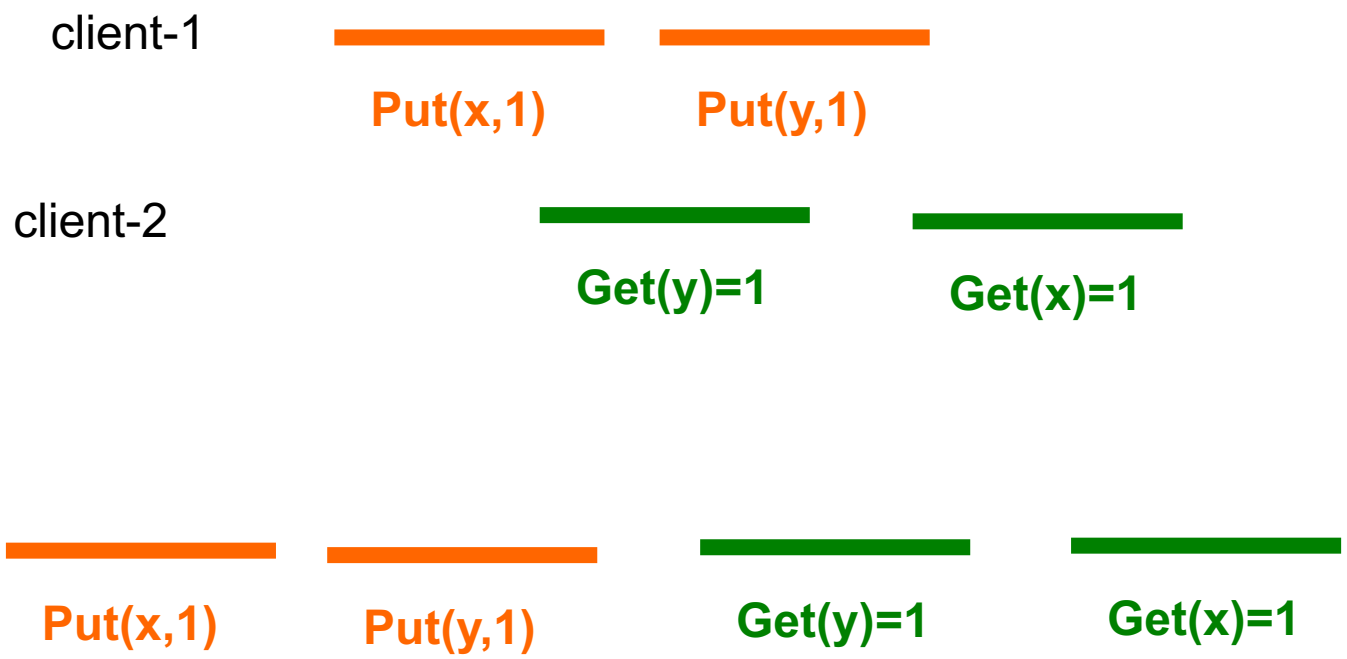
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



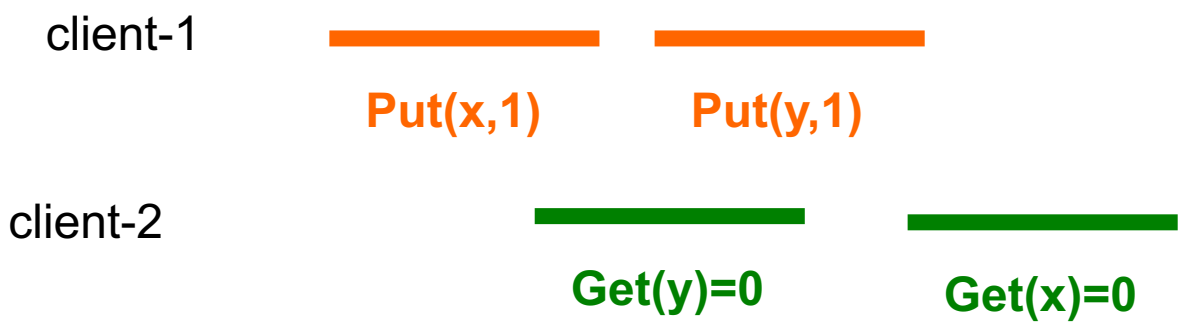
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



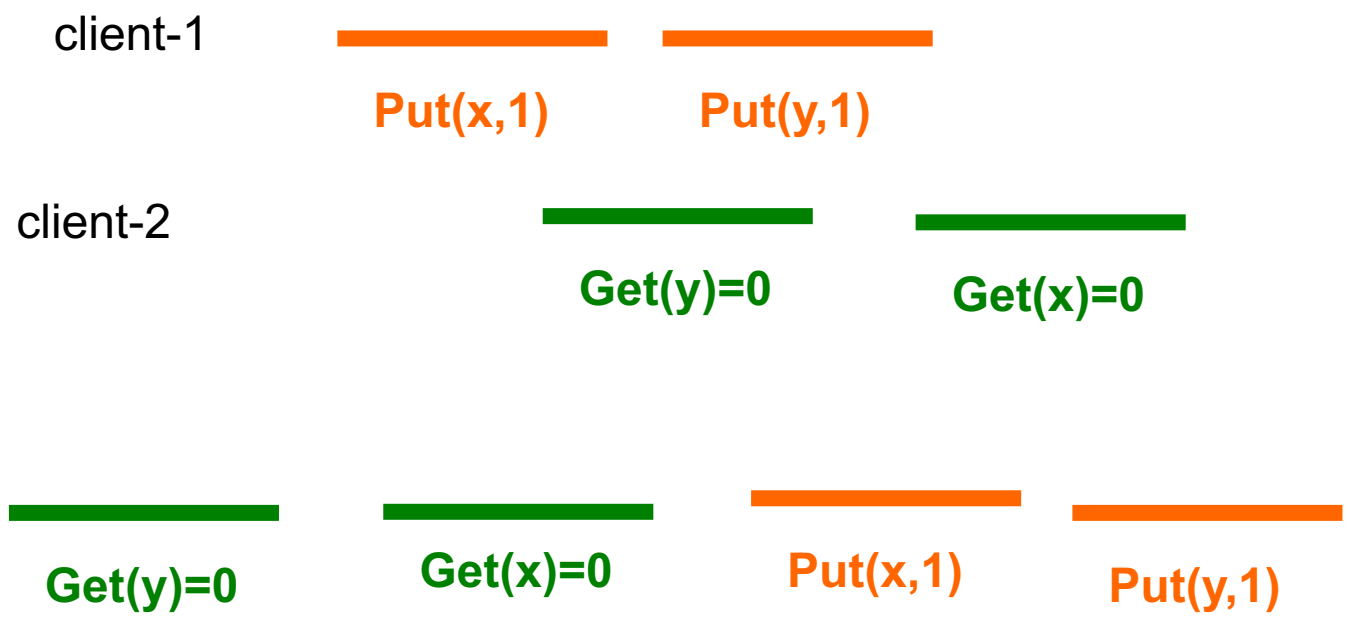
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



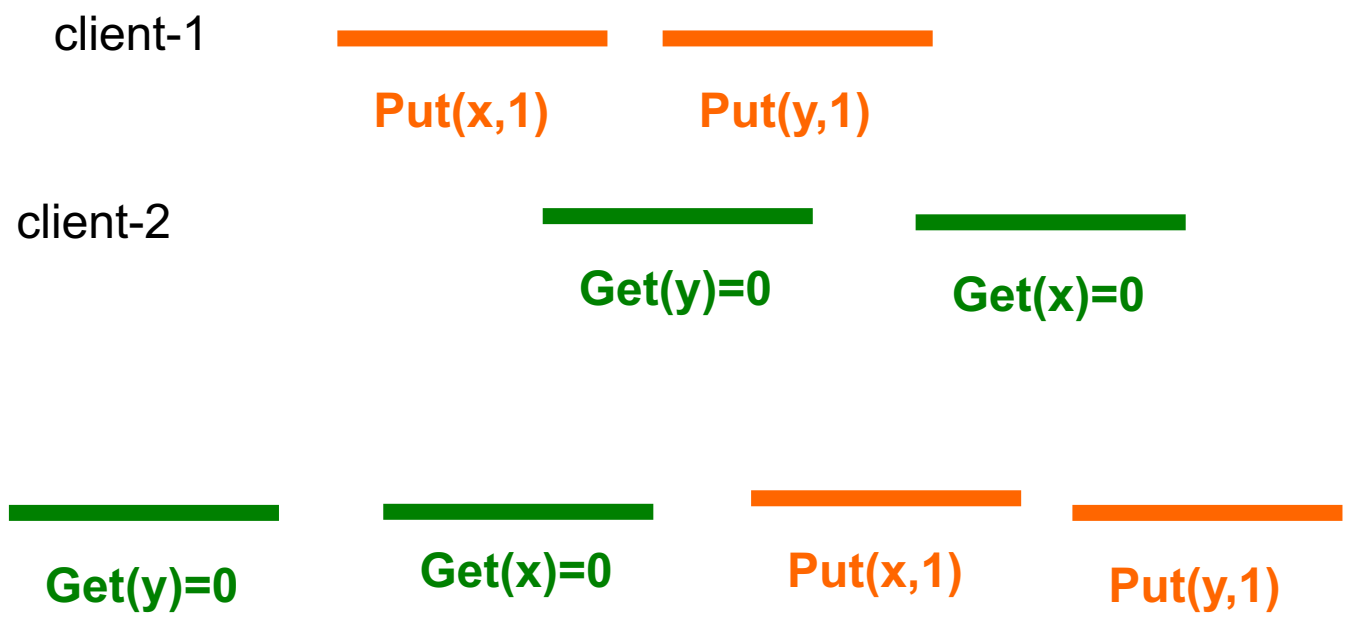
X 1. Global issue order
X 2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



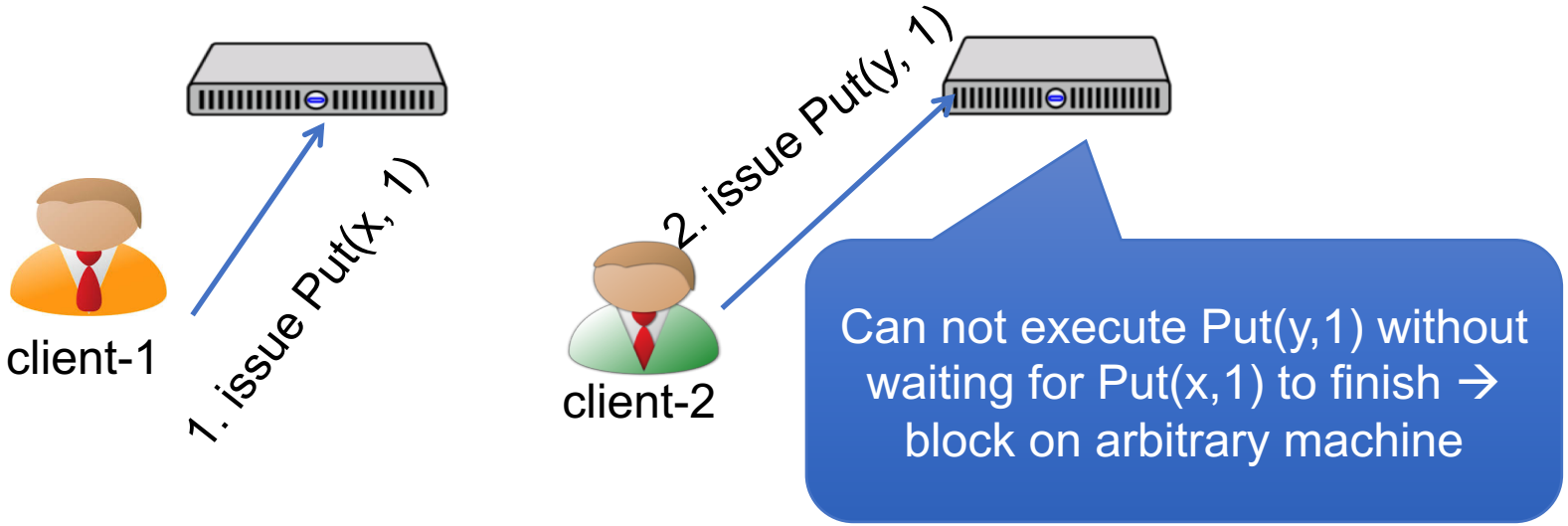
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order



✗ 1. Global issue order
✗ 2. Global completion order
3. Per-thread issue (and completion) order
✗ 4. Global completion-to-issue order

Strong consistency models

- Preserving global issue/completion order is impractical
  - → extreme blocking behavior



Strong consistency models

- Sequential consistency
  - equivalent serial history must obey per-process issue/completion order
- Linearizability
  - equivalent serial history must obey global “completion-to-issue” order
- Which one is stronger?

1. Global issue order
2. Global completion order

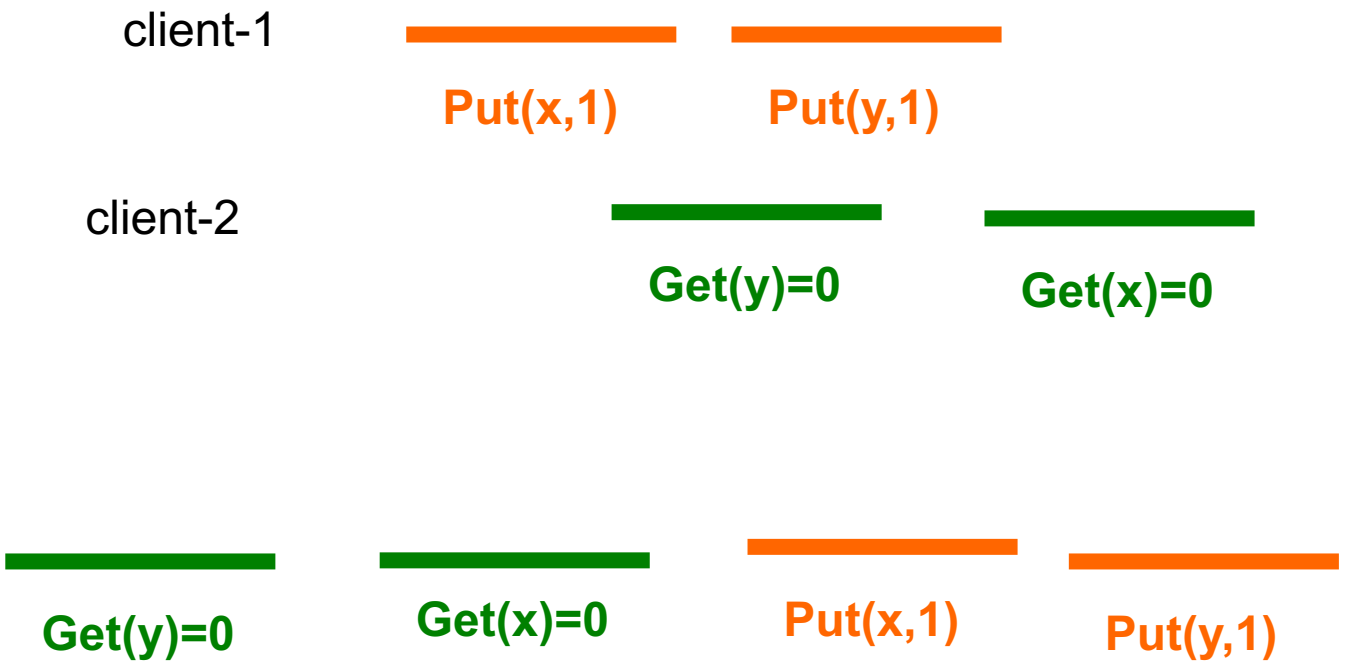
stronger

4. Global completion-to-issue order (Linearizability)

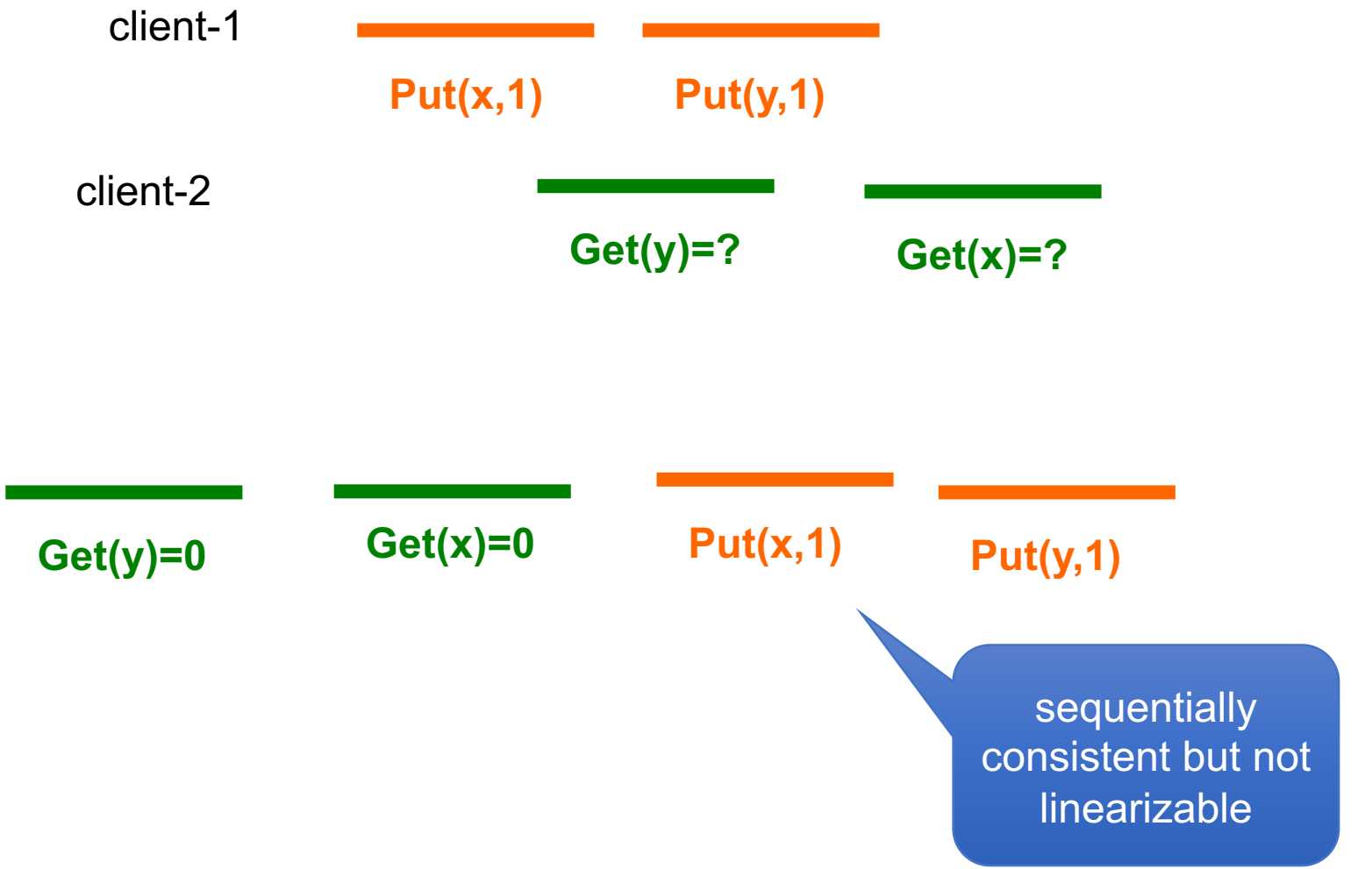
stronger

3. Per-thread issue (and completion) order (sequential consistency)

Sequential consistency vs. linearizability

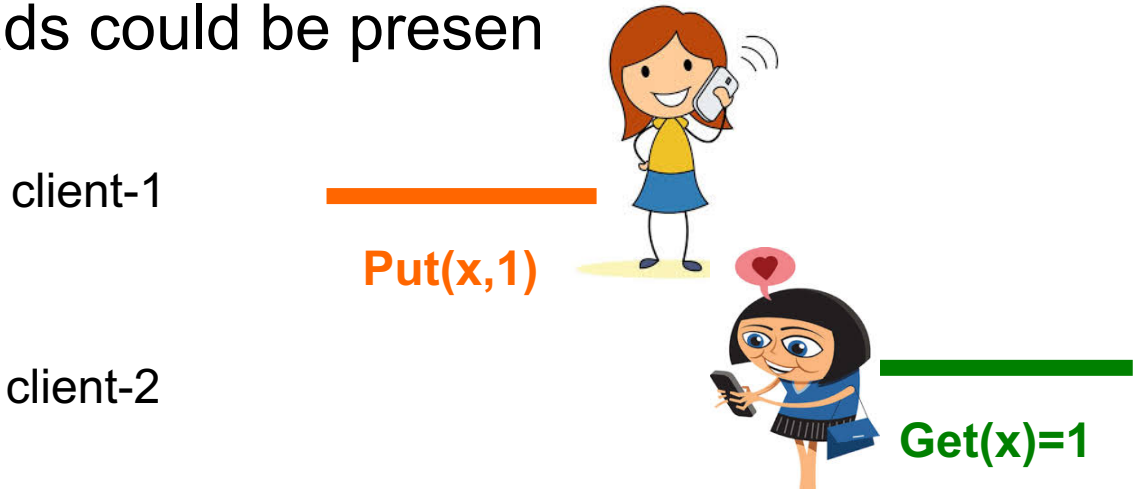


Sequential consistency vs. linearizability



Sequential consistency vs. linearizability

- Does the difference between the two matter?
  - Matters only when external communication between threads could be presen

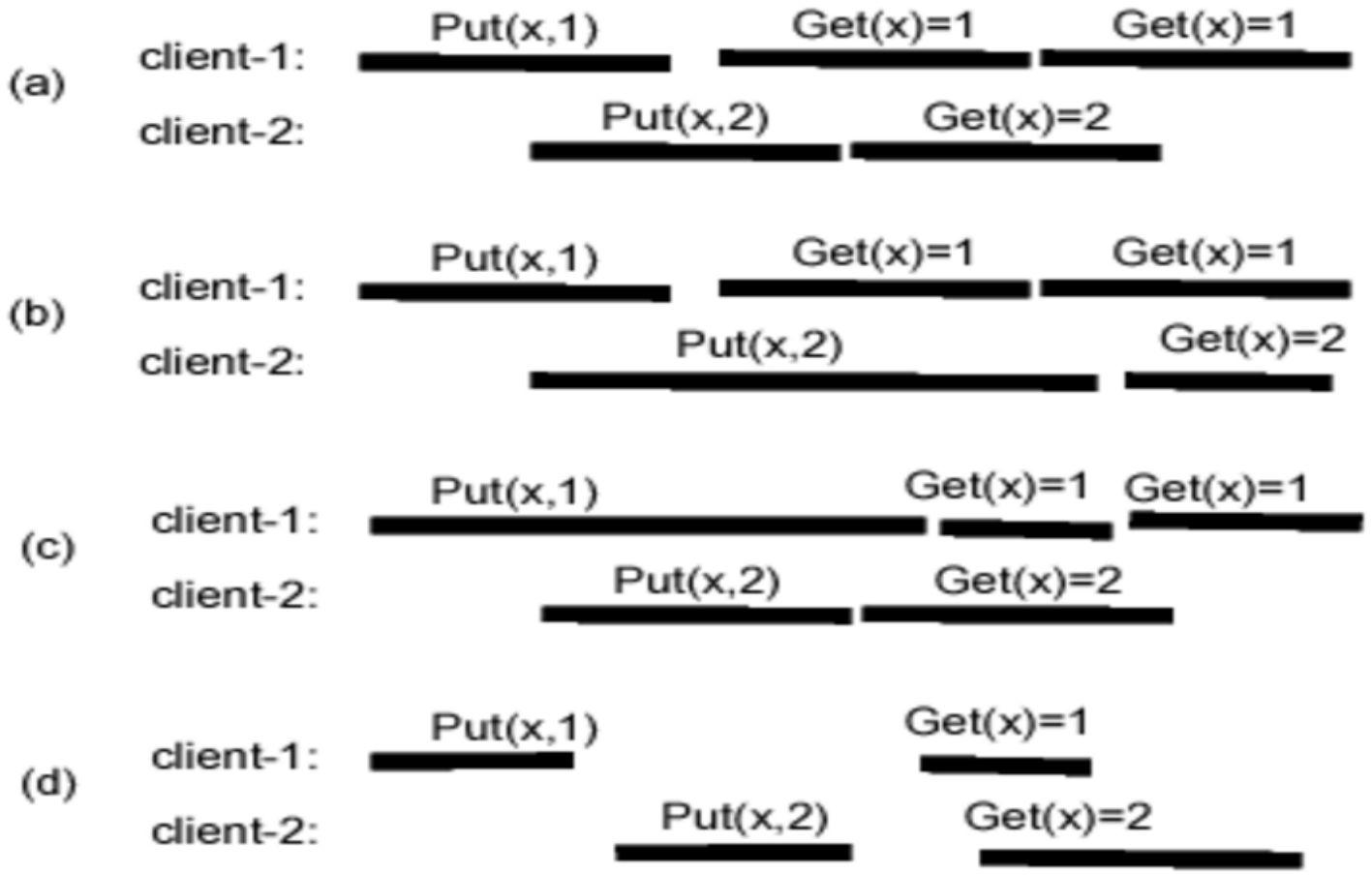


- Both are equally good w/o external communication
  - e.g. Multi-threaded programs sharing sequential consistent global memory

Why linearizability is a good strong model?

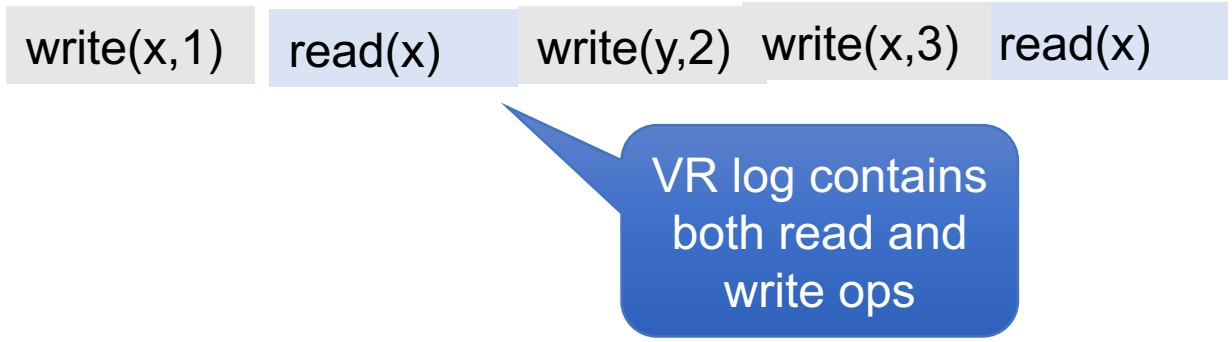
- It is strong
  - Strongest possible practical model
- It allows scalable system design
  - (the local property) if each object is linearizable, then the whole system is linearizable
  - Scale system → shard objects across servers

Recap exercise: which of the following histories are linearizable?



What are the implementations of linearizability

- Case-study: key-value store
  - Single server
  - N servers: server-i stores keys, k mod N = i
  - 3 servers replicating data using viewstamp replication



What we've learnt until now?

Strong consistency model: linearizability

- execution is equivalent to a serial history that preserves global completion-to-issue order

How to implement linearizability

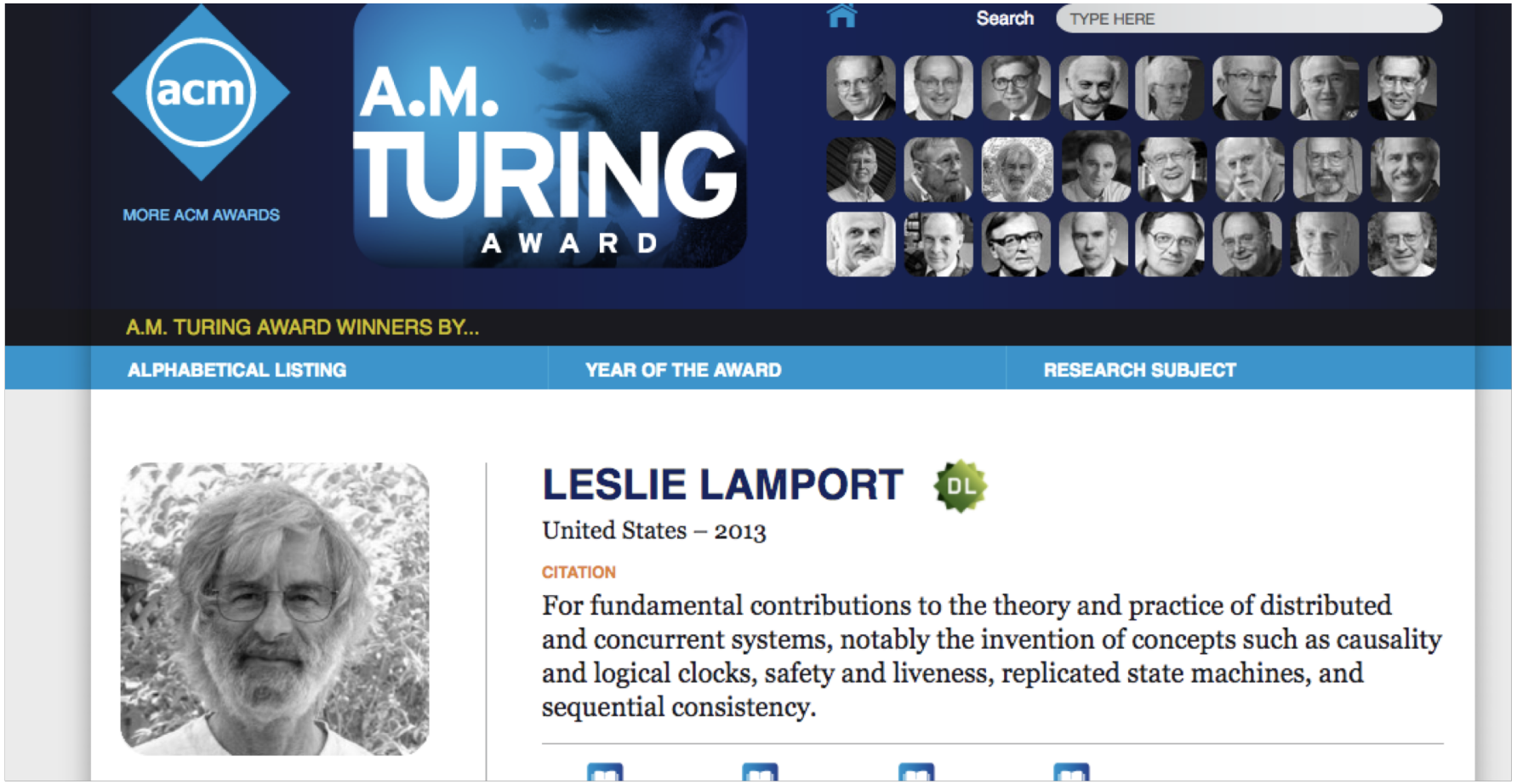
- no replication
- viewstamp replication

Consensus – Consistency Replication

Allow a set of nodes to agree on something in the presence of a number of failure.

- Replica servers agree on the same sequence of operations for execution
- Linearizable System (e.g., VR)

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

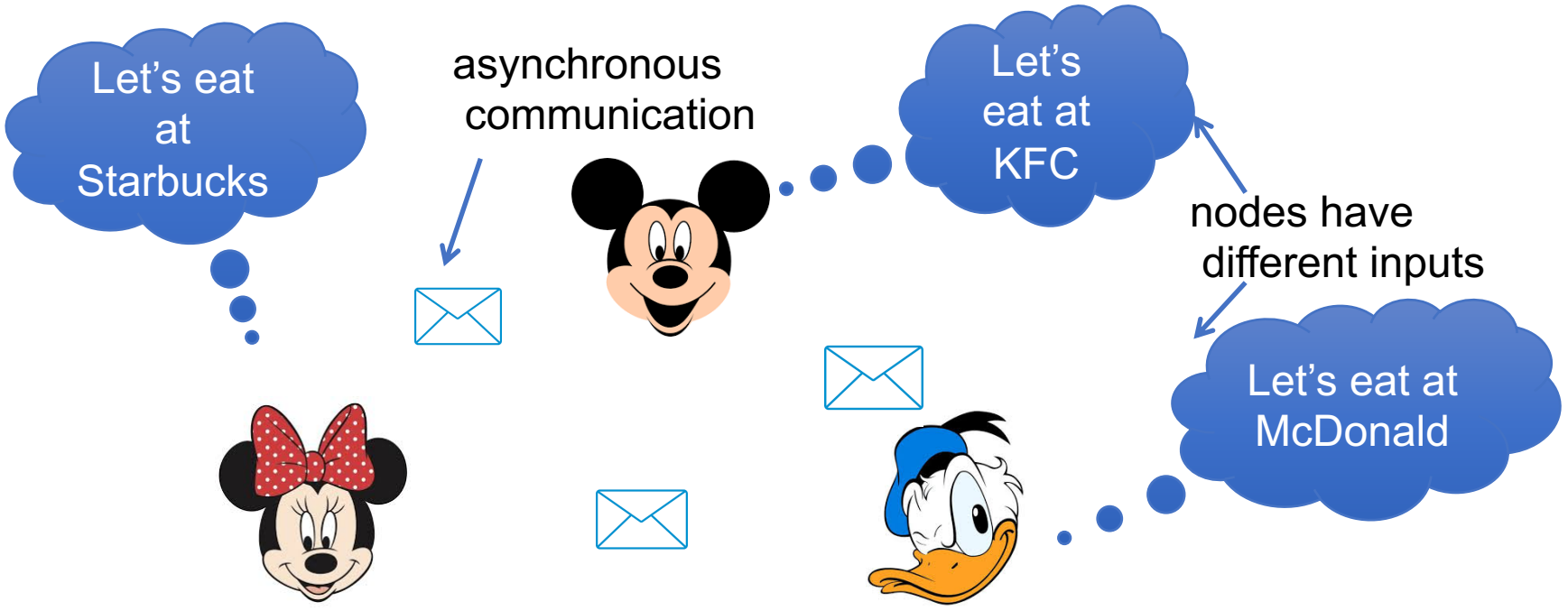
Problem setting

- 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

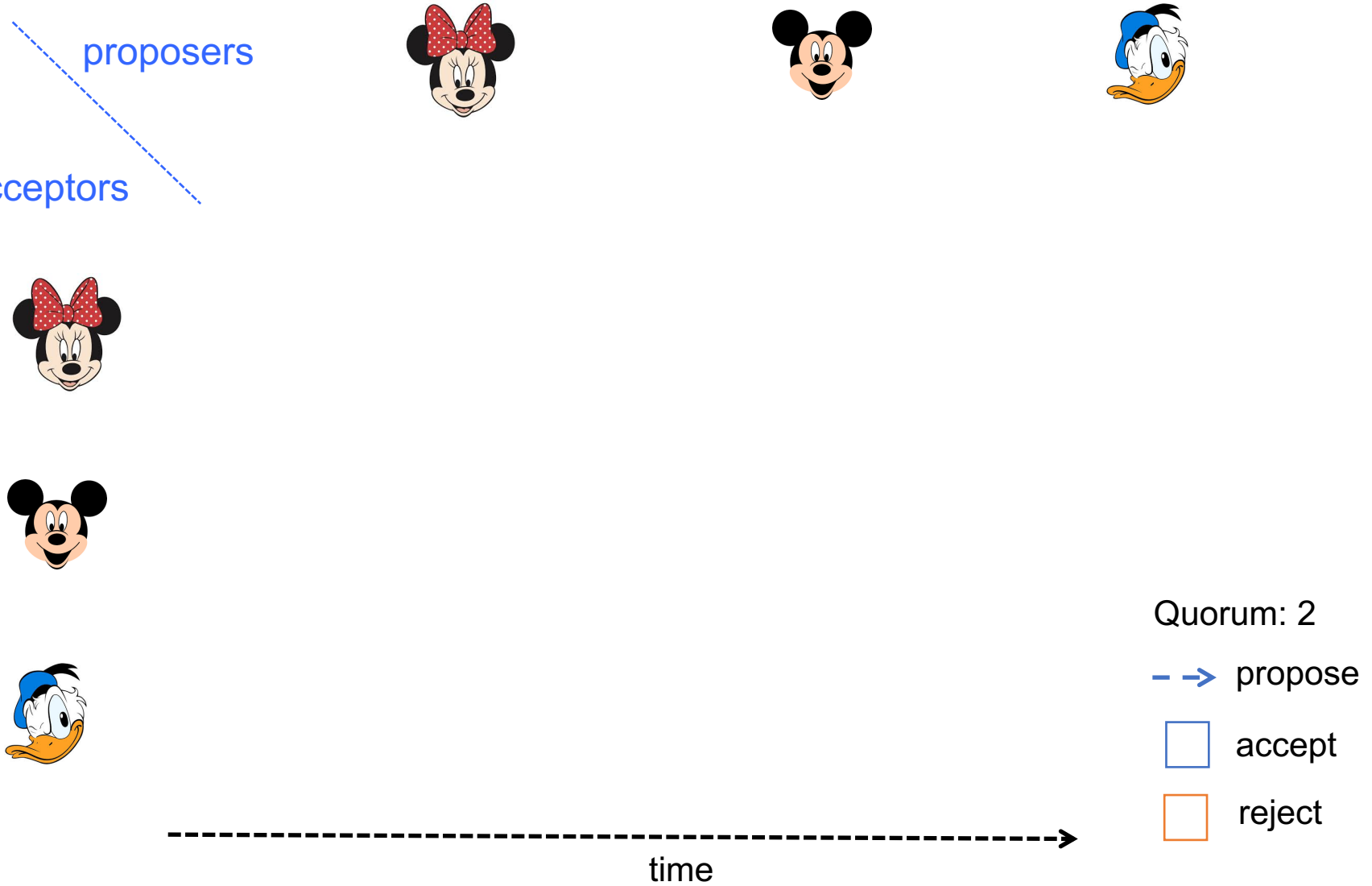
Roles

- 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

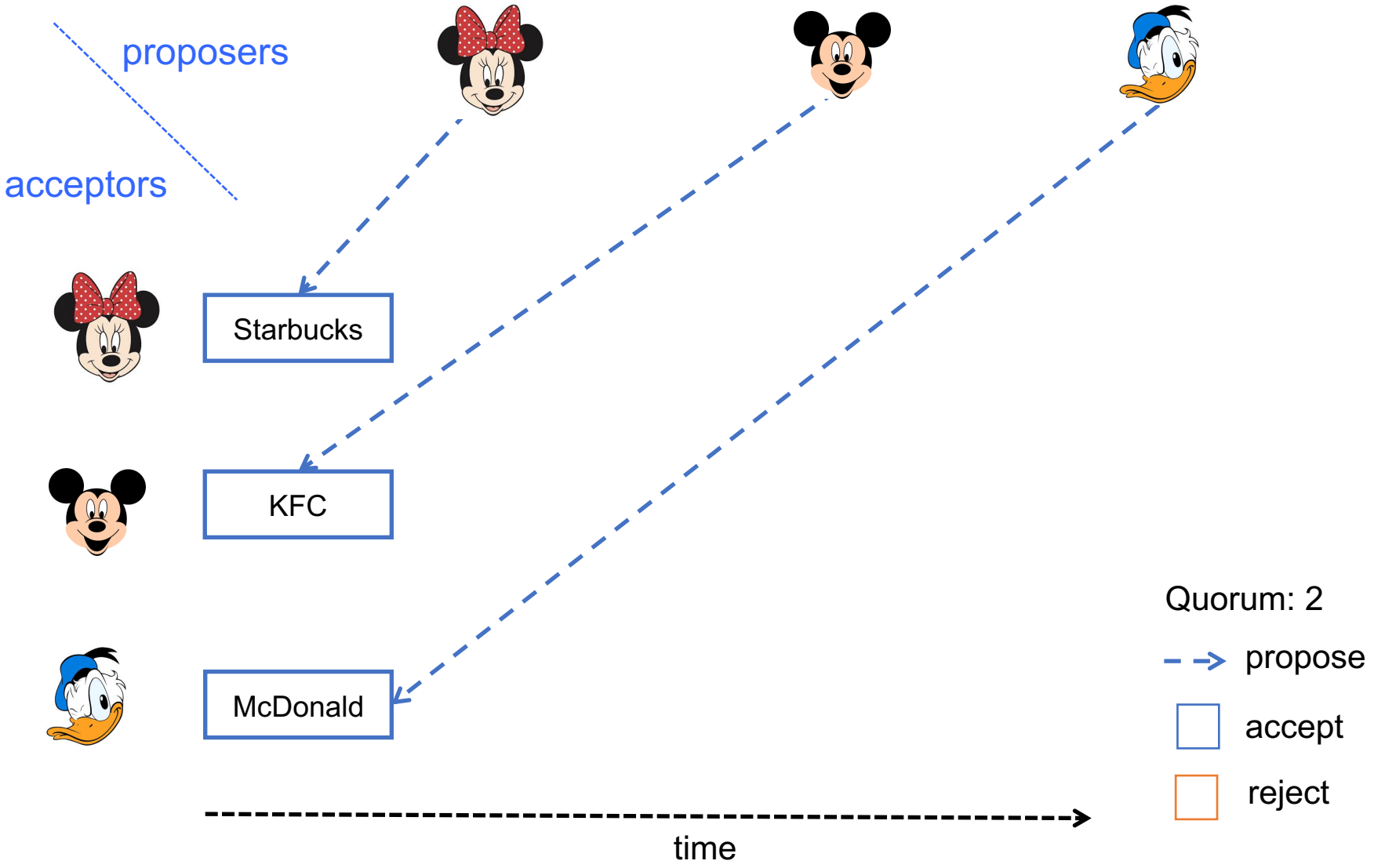
Chosen

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

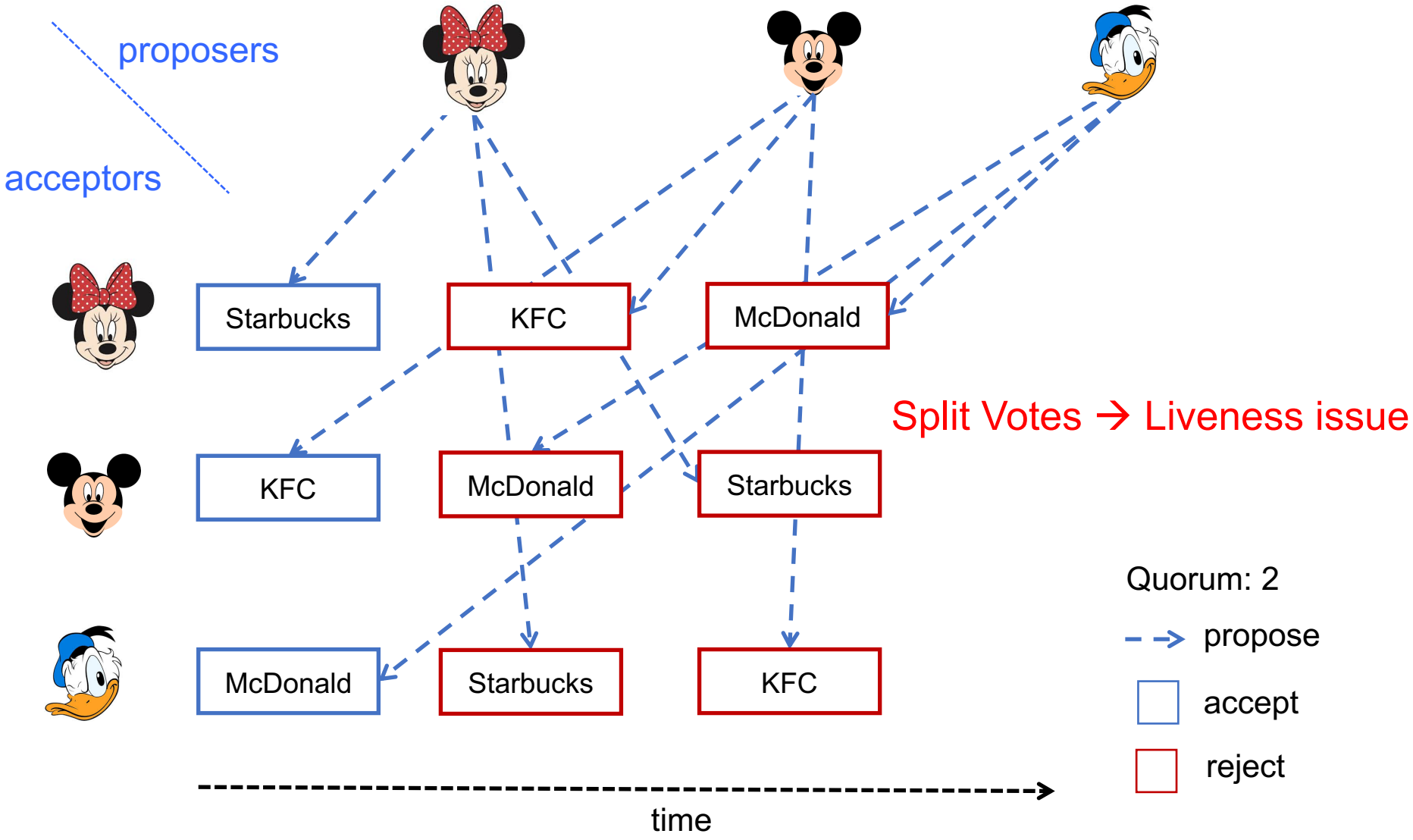
Method I – Only accept the first



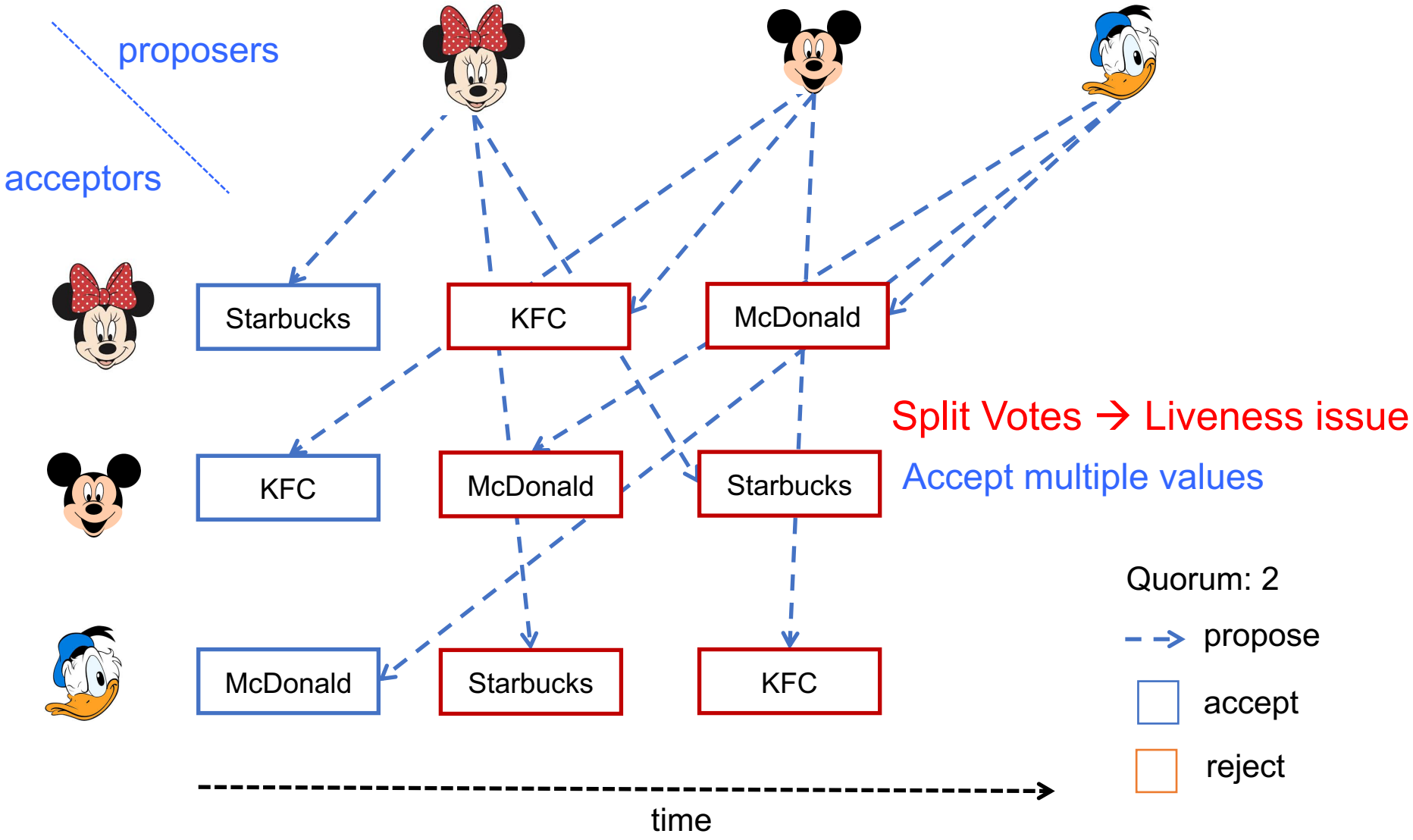
Method I – Only accept the first



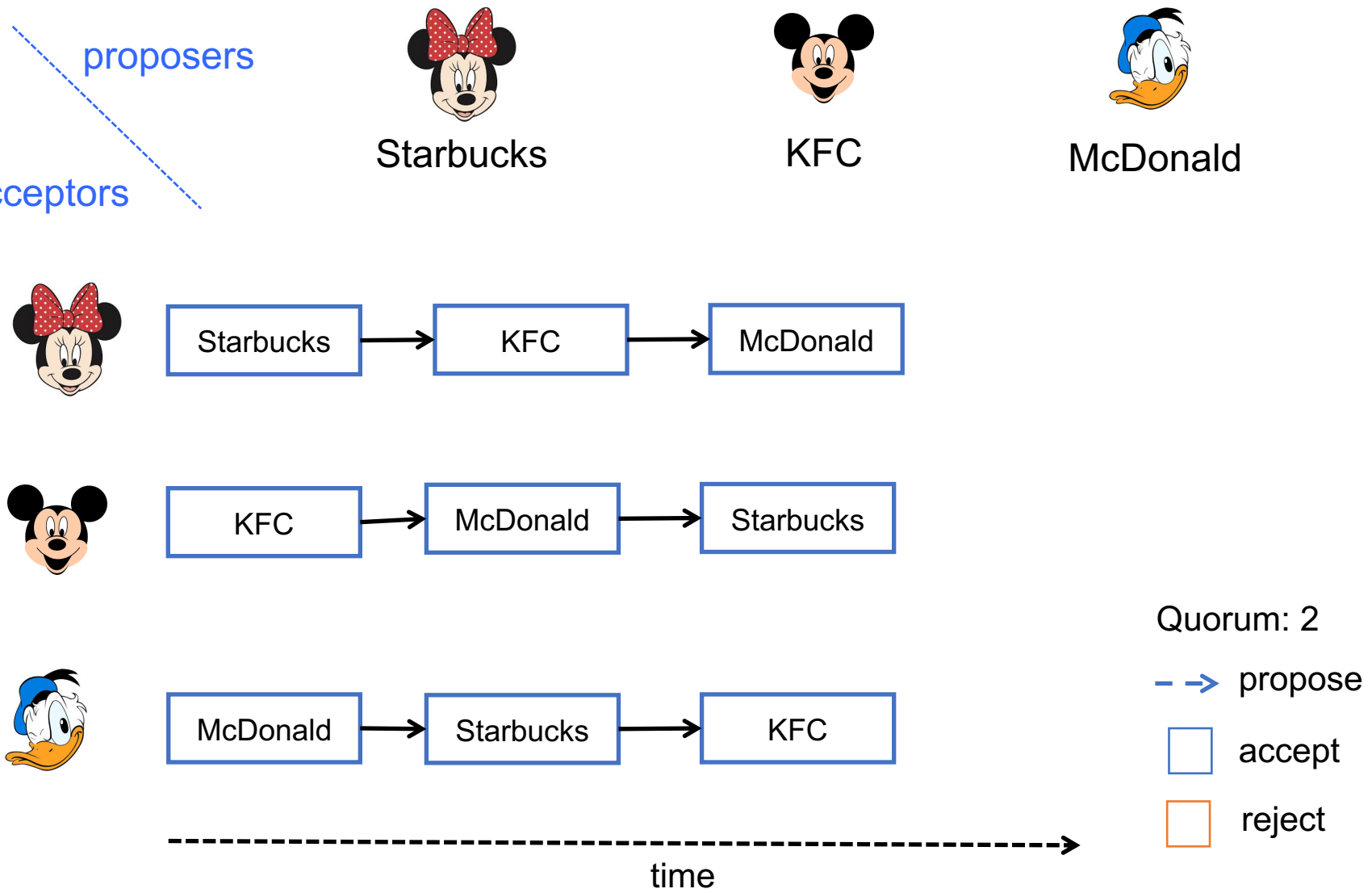
Method I – Only accept the first



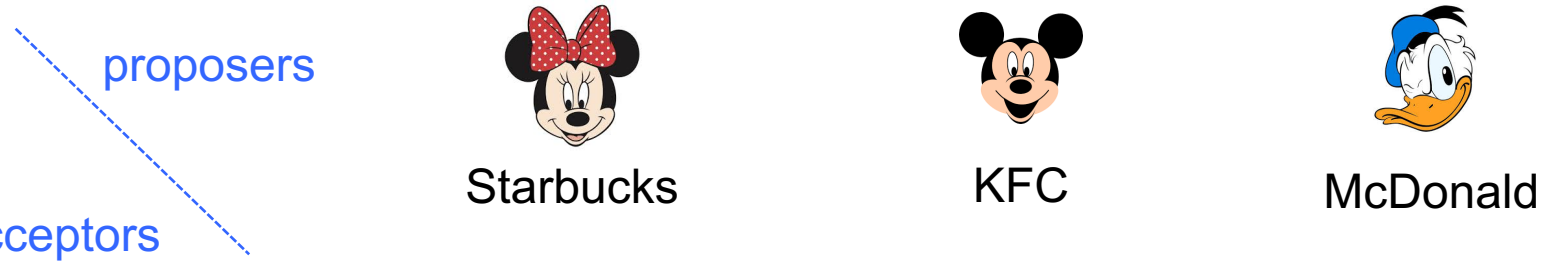
Method I – Only accept the first



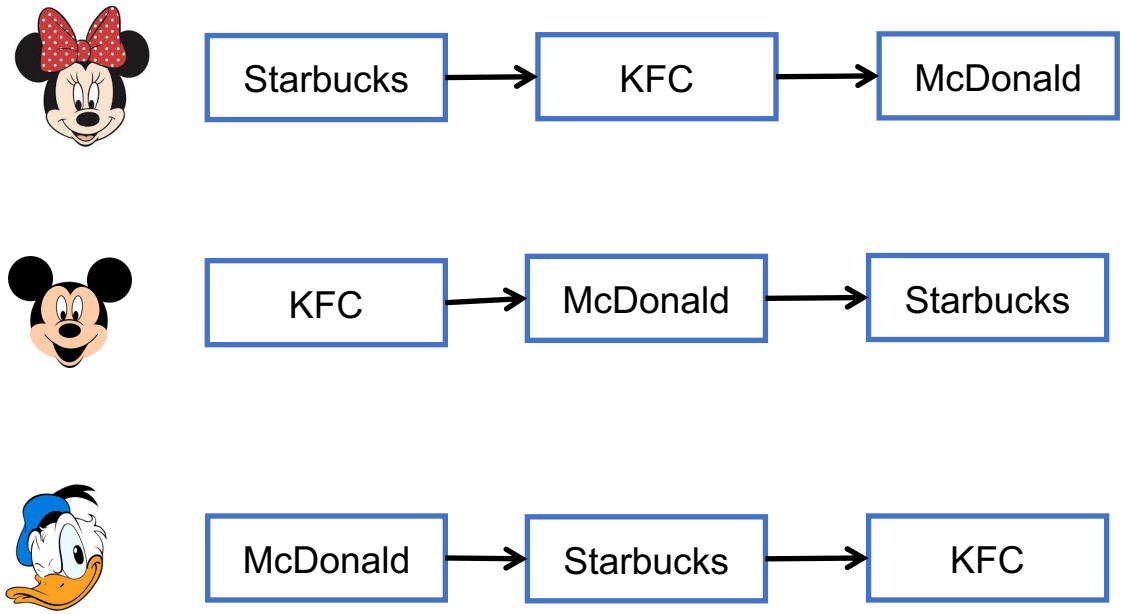
Method II – Accept multiple proposed values



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.

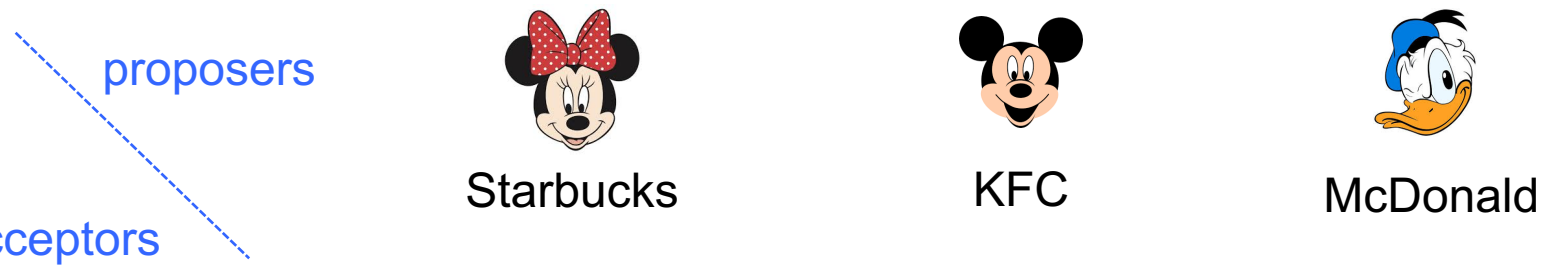
Quorum: 2

propose

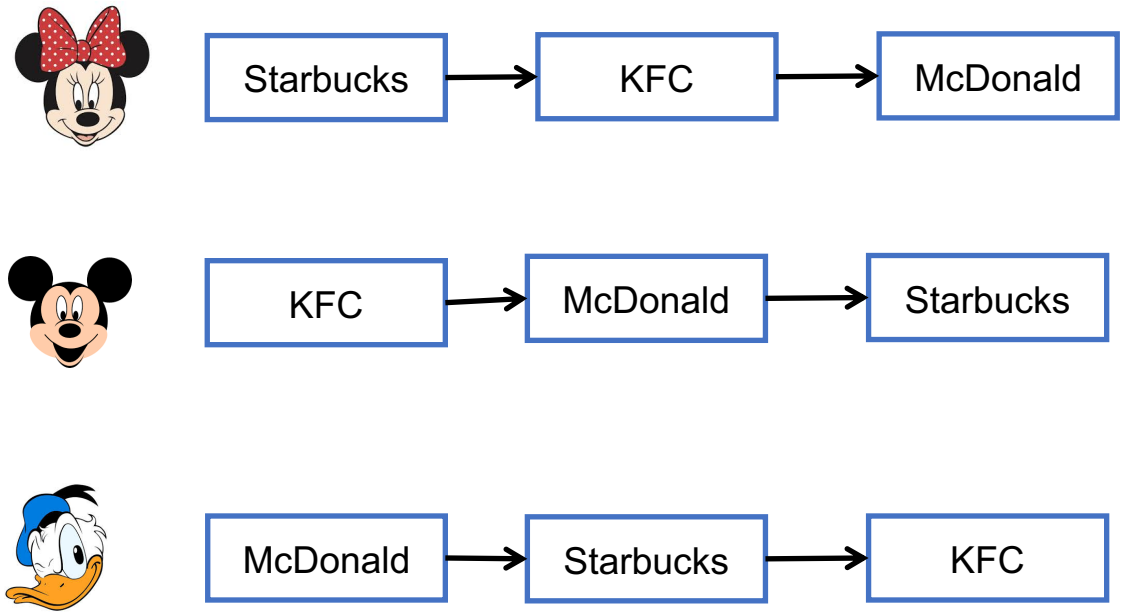
accept

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