Please fill this in!

course-evaluations.berkeley.edu

If 80% fill it in, 1 EC point on MT3 post-curve

Pre-162

Post-162

CS162
Operating Systems and Systems Programming
Lecture 26

Coordination - Paxos

Professor Natacha Crooks
https://cs162.org/

Slides based on prior slide decks from David Culler, Ion Stoica, John Kubiatowicz, Alison Norman, Indy Gupta and Lorenzo Alvisi

Recall: General's Paradox

If the network is unreliable, it is impossible to guarantee two entities do something simultaneously

If nodes behave maliciously, impossible to get eventual agreement if there are less than 3f+1 parties present (of which f can misbehave)

Entire textbook on impossibility results in distributed computing ...

Recall: Eventual Agreement: Two-Phase Commit

Two or more machines agree to do something, or not do it, atomically

No constraints on time, just that it will eventually happen!

Used in most modern distributed systems!
Representative of other coordination protocols

Recall: 2PC Summary

Why is 2PC not subject to the General's paradox?

- Because 2PC is about *all nodes eventually coming to the same decision – not necessarily at the same time!*
- Allowing us to reboot and continue allows time for collecting and collating decisions

Biggest downside of 2PC: blocking

- A failed node can prevent the system from making progress
- Still one of the most popular coordination algorithms today

Failures

What types of failures can arise in distributed system?

- Crash
- Omission
- Arbitrary Failures (Byzantine)

Failure Model as a Contract

A system with N replicas will

1) Remain safe (produce a correct output)
2) Remain live (eventually produce correct output)

As long as there are no more than F failures.

What happens when there are more than f failures?
=> All bets are off.

Solving Consensus

How can we get a group of machines to agree on a single value when

1) There can be concurrent values proposed

2) Machines can fail!

2PC blocks in the presence of failures and requires explicit coordinator. How can we solve consensus in the presence of f failures?

Consensus

Given a set of processors, each with an initial value:

Termination: All non-faulty processes eventually decide on a value

Agreement: All processes that decide do so on the same value

Validity: Value decided must have proposed by some process

Consensus

Consensus is impossible in an asynchronous system!

Realisation 1:
Every *fun* thing in distributed systems is impossible

Realisation 2:
We build these systems anyways.

Paxos

Most popular consensus algorithm but doesn't (quite) solve consensus

Provides safety and eventual liveness

Safety: Consensus is not violated

Eventual Liveness: If things go well sometime in the future (messages, failures, etc.), there is a good chance consensus will be reached. But there is no guarantee.

Used (in some form) in most major distributed systems

Google's Chubby, Yahoo's Zookeeper, MultiPaxos in Spanner, Raft in Etcd/TiKV.

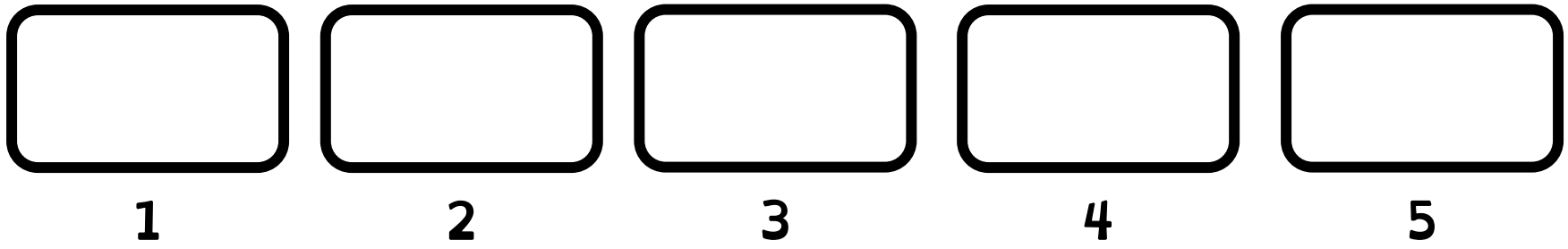
System Model

2f + 1 nodes, f of which may fail.

No upper-bound in message delivery *but* assume messages eventually arrive

No "special" coordinator node. Everyone is equal

Propose values (data/numbers, etc)



Greek Island Politics

the problem of governing with a part-time parliament bears a remarkable correspondence to the problem faced by today's fault-tolerant distributed systems, where legislators correspond to processes and leaving the Chamber corresponds to failing. The Paxons' solution may therefore be of some interest to computer scientists. I present here a short history of the Paxos Parliament's protocol, followed by an even shorter discussion of its relevance for distributed systems

No one really understood this version, so there's been many "translations" for Computer Scientists since

Paxos Made Simple
Paxos Made Moderately Complex
An Engineering Perspective on Paxos
Viewstamp Replication
Raft

Rounds

Paxos has rounds;
Each round has a unique ballot id

Rounds are asynchronous

Time synchronization not required
(but preferred for liveness)

If you're in round *j* and hear a message from round *j+1*, abort everything and move over to round *j+1*

Three Phases Per Round

Each round itself broken into phases
(which are also asynchronous)

Phase 1: A leader is elected
(Election)

Phase 2: Leader proposes a value, processes ack
(Bill)

Phase 3: Leader multicasts final value
(Law)

Phase 1 – Election

Potential leader (Proposer) chooses a ballot id.

Ballot id must be unique per proposer

Ballot id must be higher than any ballot id seen anything so far

Phase 1 – Election (Version 1)

Proposer sends PREPARE(ballot_id) to all participants.

If participant has already received a higher ballot id (b > ballot_id), do nothing.

Else:

1) Store b=ballot_id on disk
2) Send an PROMISE(ballot_id) to proposer.

Have I already agreed to ignore proposals with this proposal number?

Phase 1 – Election (Version 1)

If majority (i.e., quorum) respond PROMISE(ballot_id) then, proposer is the leader

Why a majority?

In what cases may the leader not receive a majority of votes?

Invariant: once have established a leader for ballot_id, no leader can be elected for a ballot smaller than ballot_id

Phase 2 – Proposal (Bill) (Version 1)

Leader sends proposed value v by sending PROPOSE(ballot_id,v) to all

If participant has already received a higher ballot id (b > ballot_id), do nothing.

Else:

1) Store b=ballot_id on disk
2) Send an ACCEPT(ballot_id, v) to proposer.

Have I already agreed to ignore proposals with this proposal number?

Phase 3 – Decision (Law) (Version 1)

If leader hears a majority of ACCEPT(ballot_id, v),

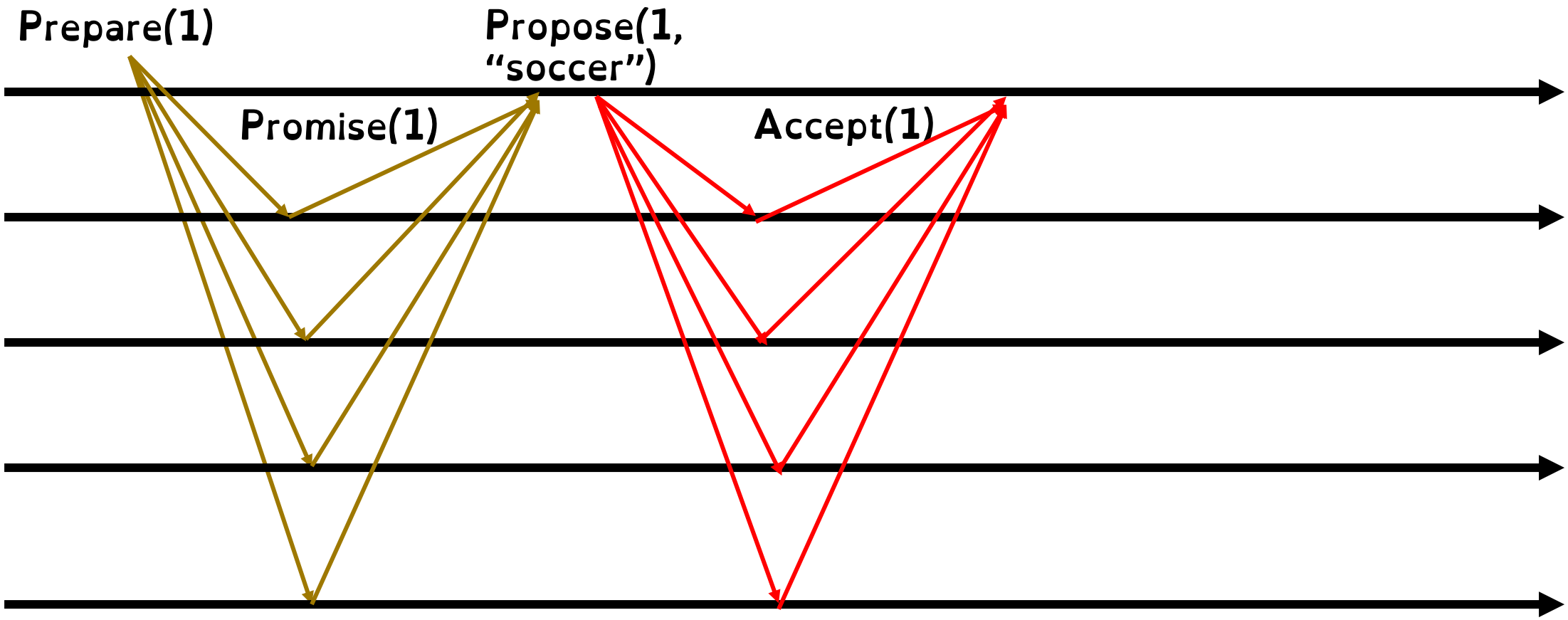
It lets everyone know of the decision.

Sends a COMMIT(ballot_id, v)

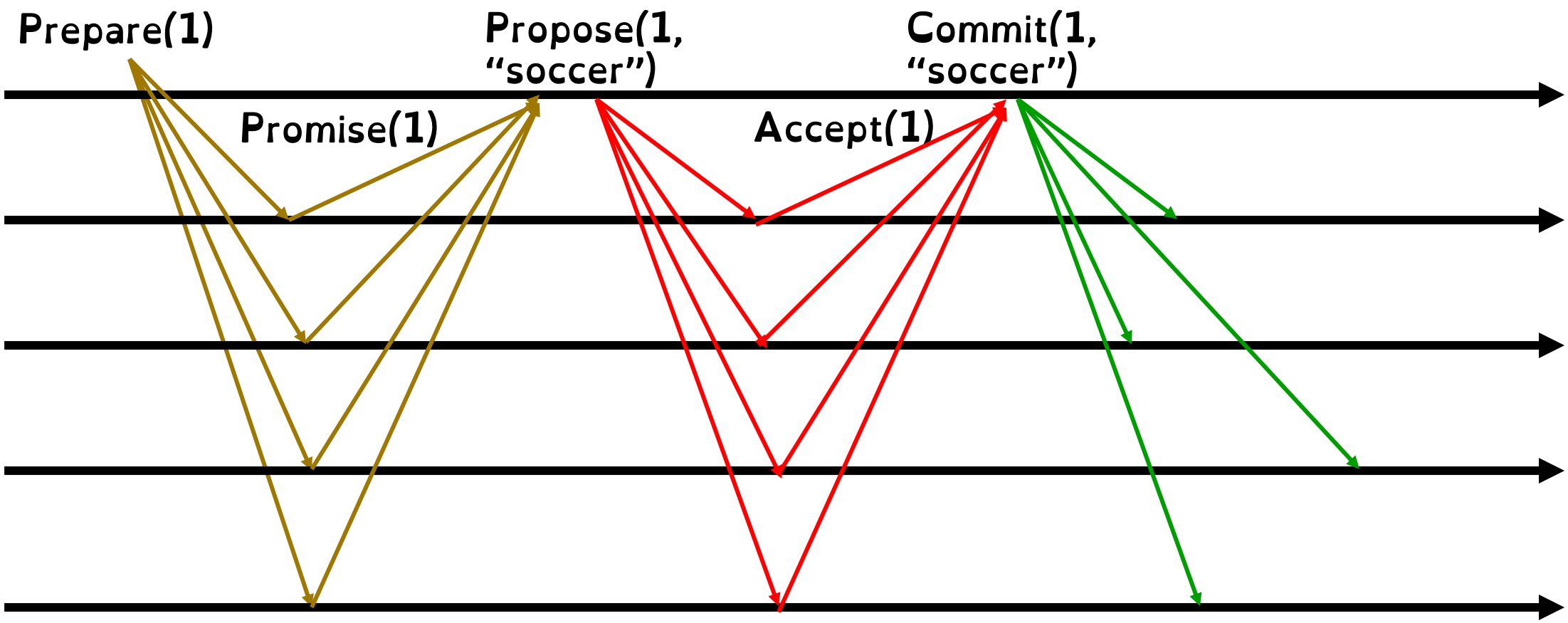
Participants can now execute v.

Easy Example

Easy Example

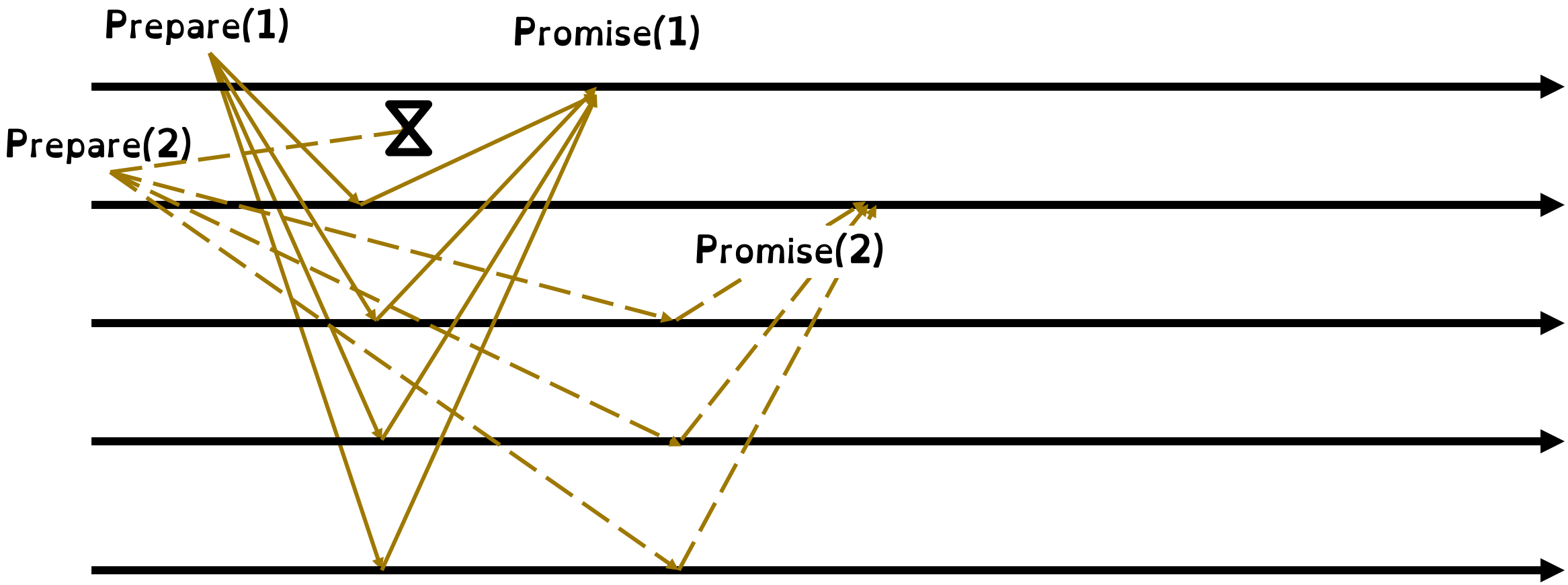


Easy Example

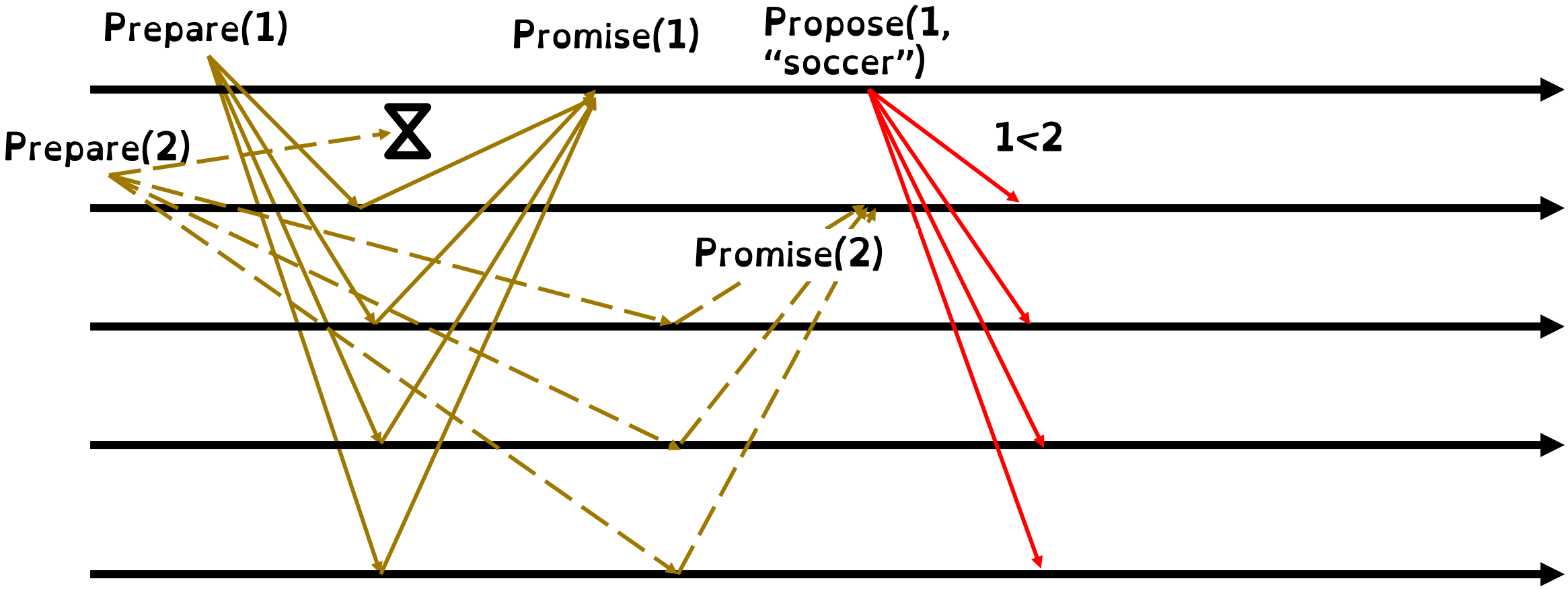


Moderately Easy Example

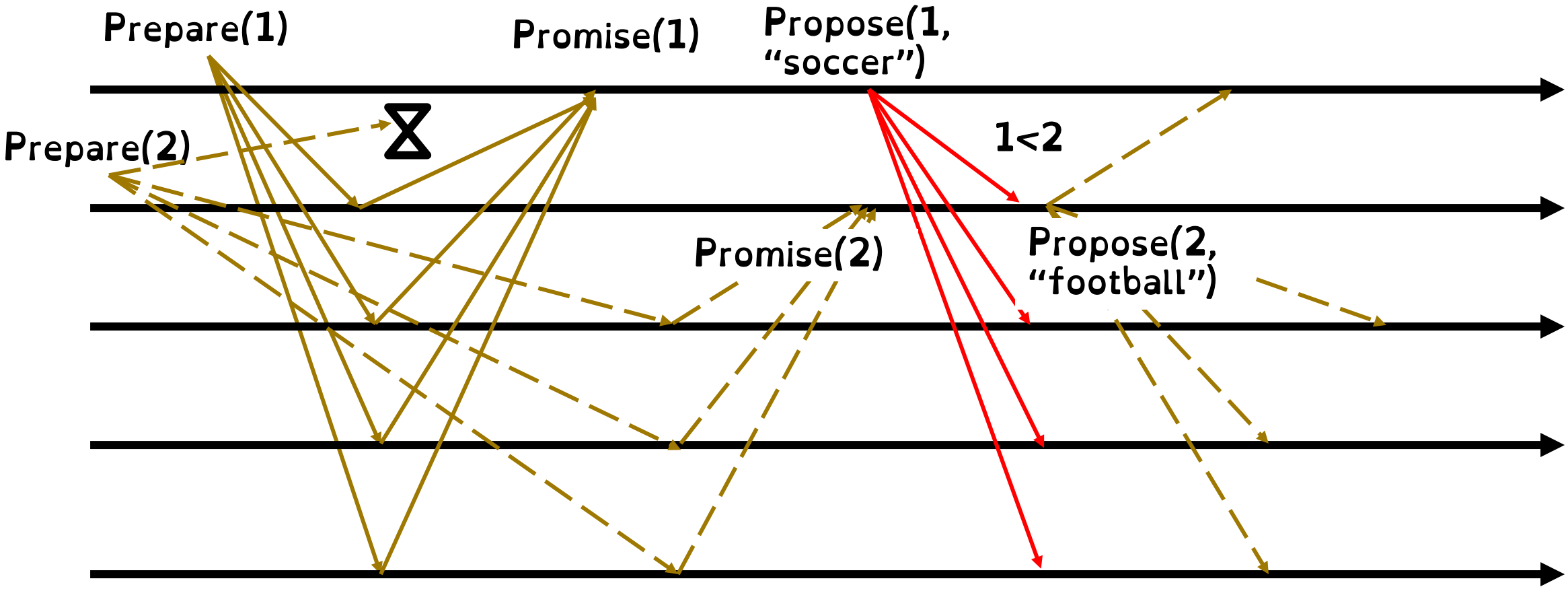
Moderately Easy Example



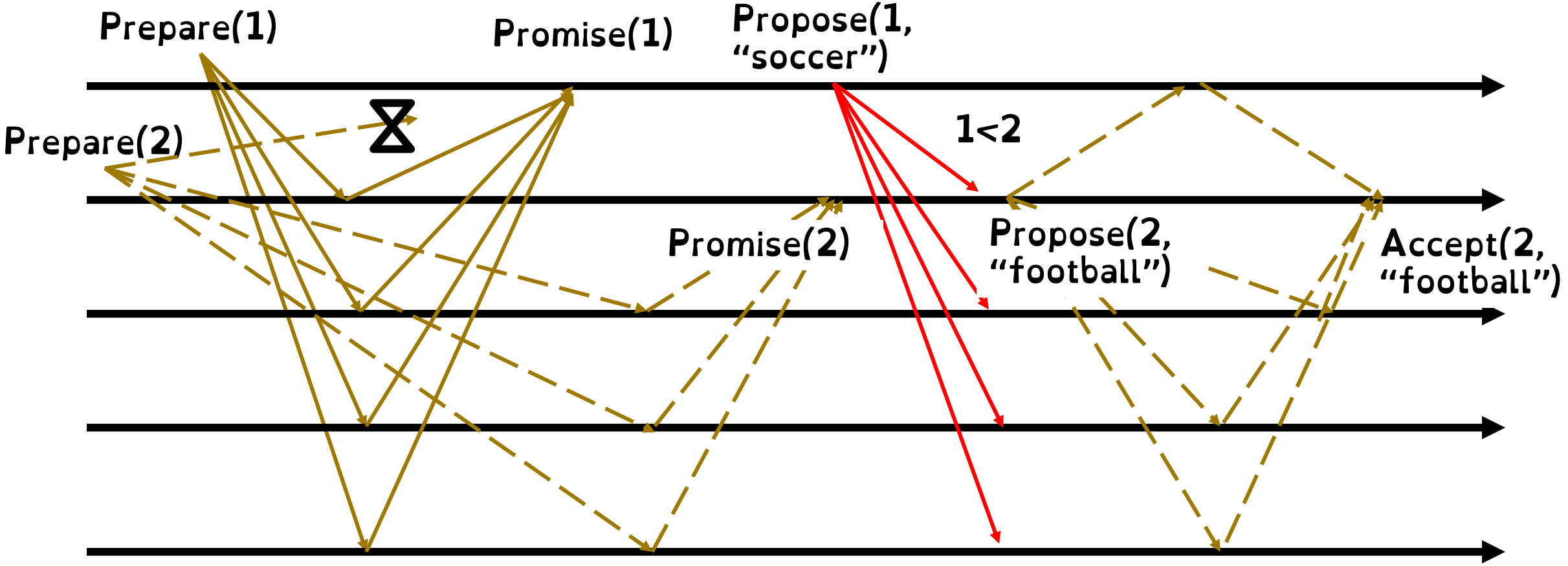
Moderately Easy Example



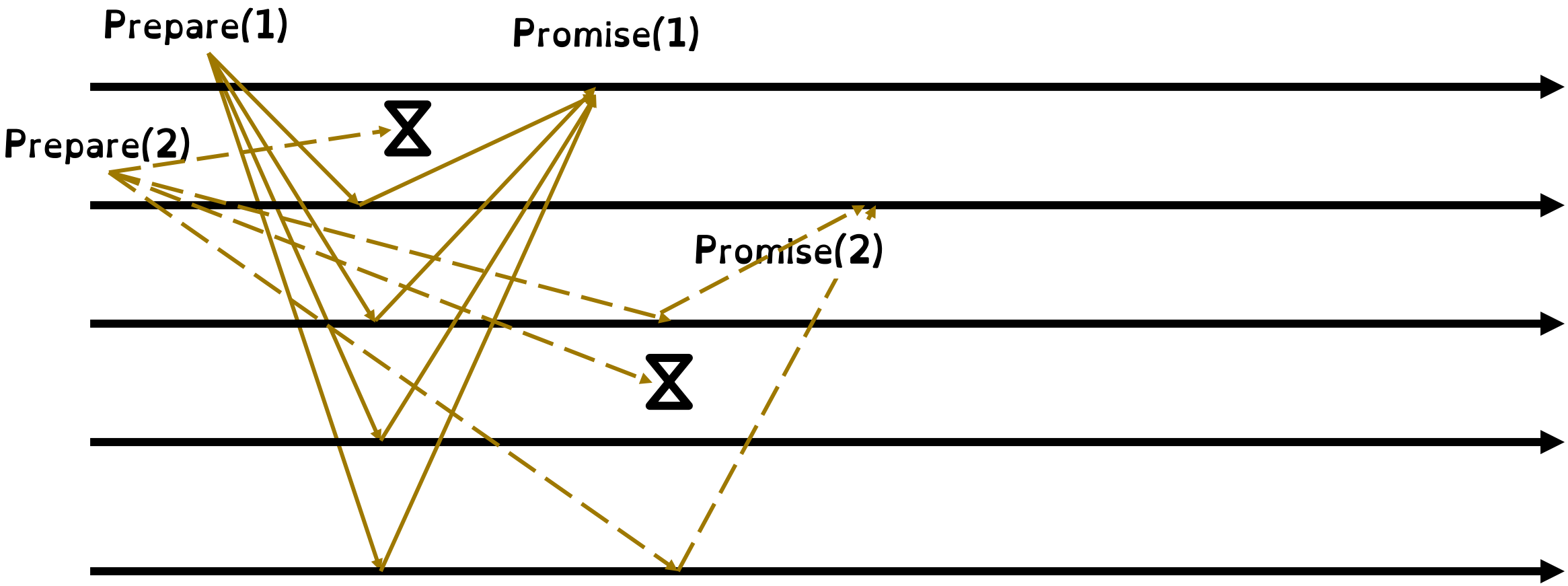
Moderately Easy Example



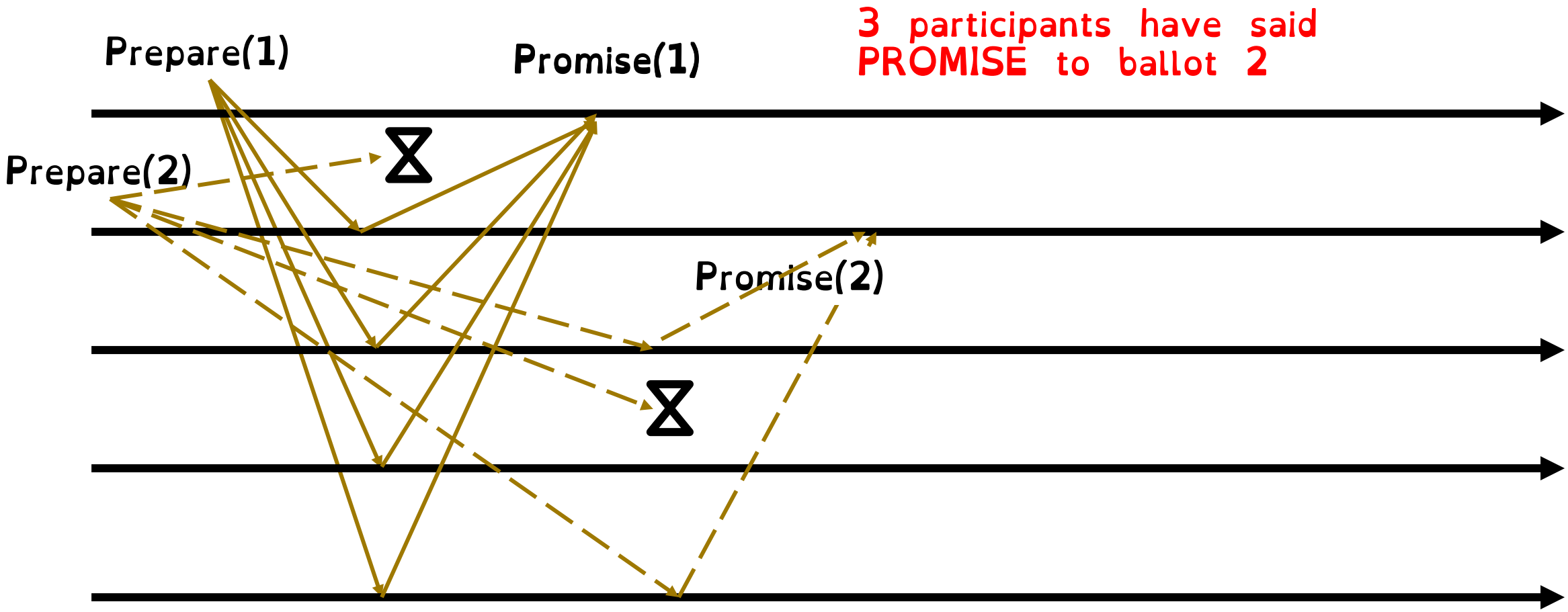
Moderately Easy Example



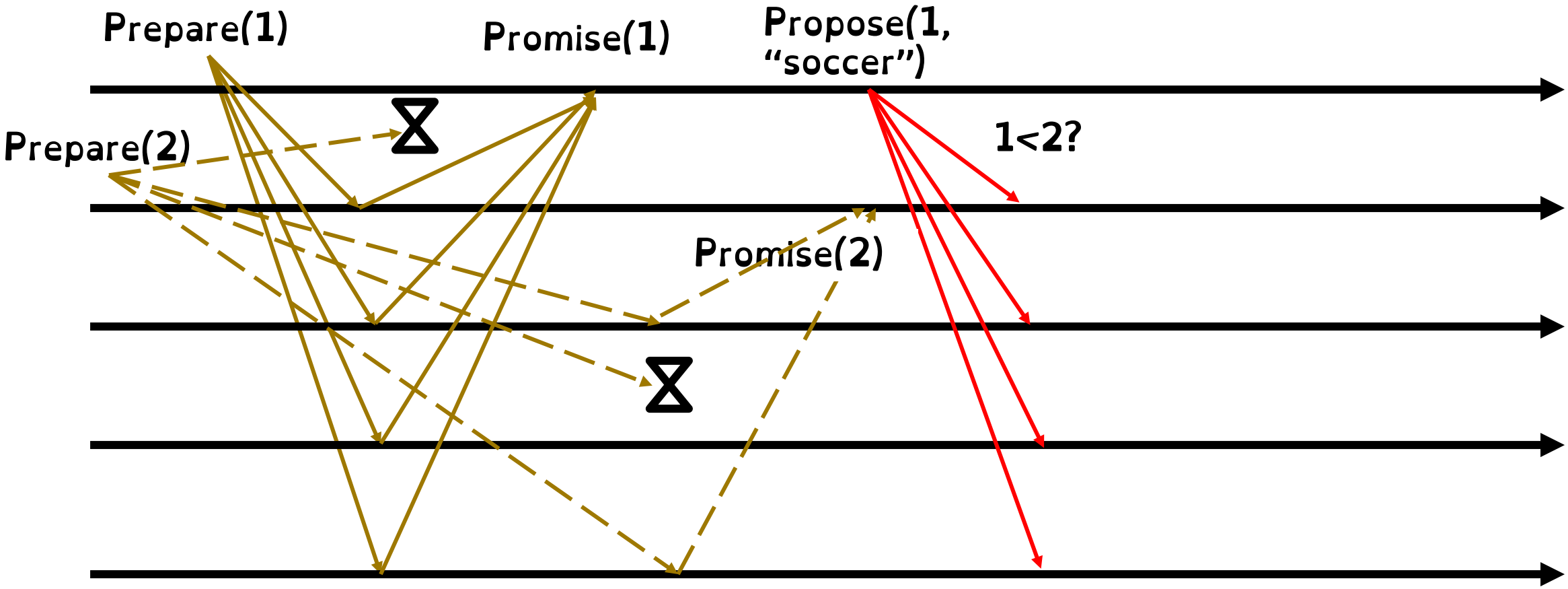
Moderately Easy Example: Quorums



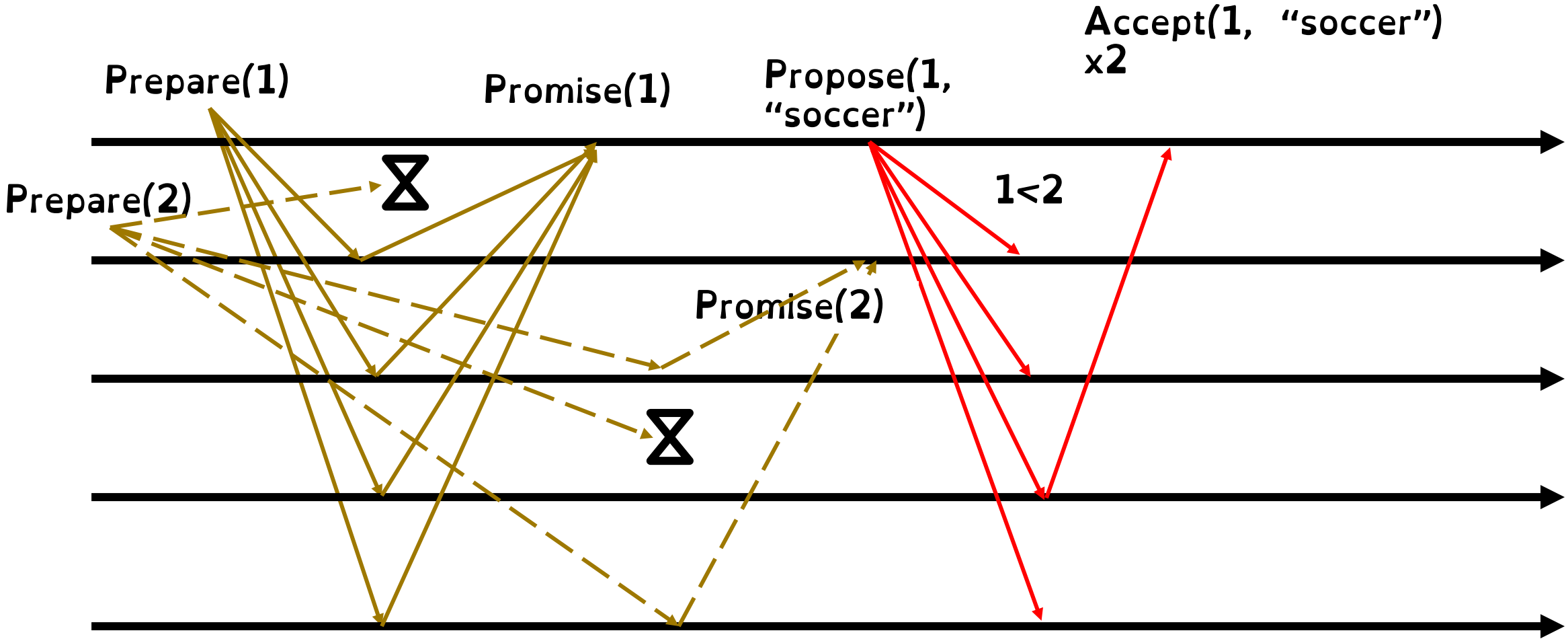
Moderately Easy Example: Quorums



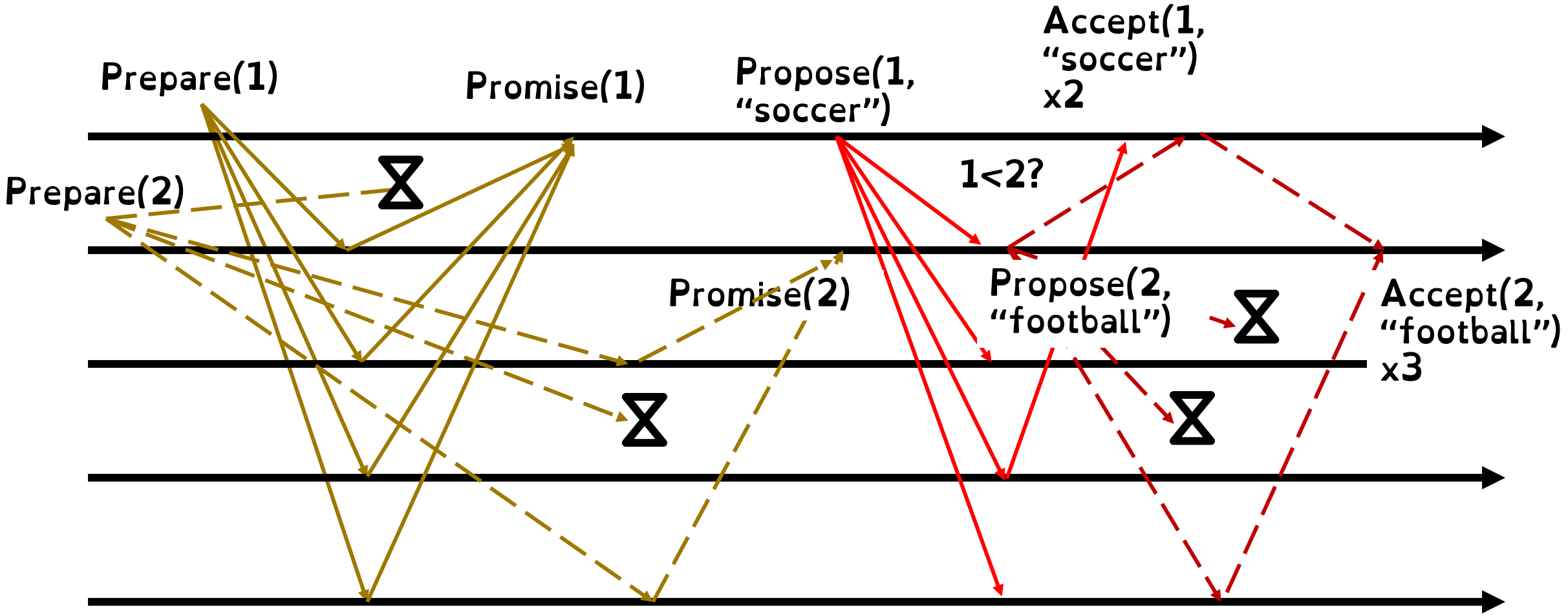
Moderately Easy Example: Quorums



Moderately Easy Example: Quorums



Moderately Easy Example: Quorums



Will never receive a majority of Accept(1,soccer) if a majority of nodes have already promised Promised(2)

Are we done?

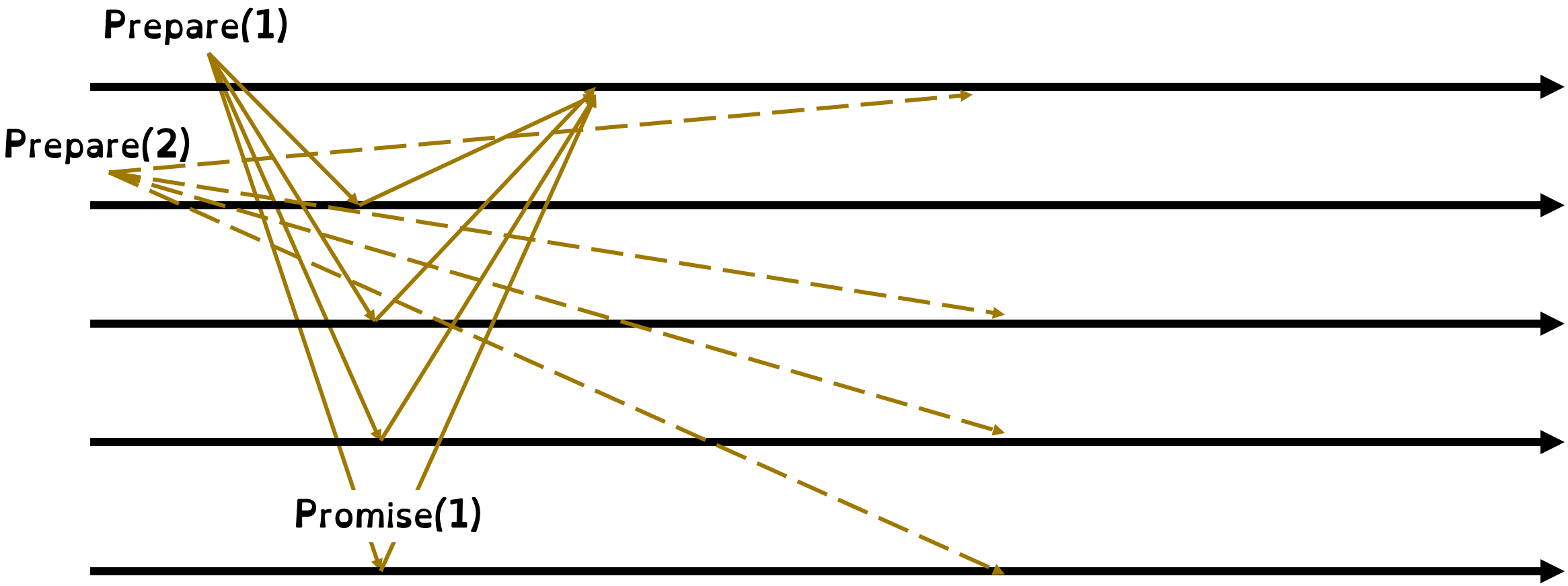
Given a set of processors, each with an initial value:

Termination: All non-faulty processes eventually decide on a value

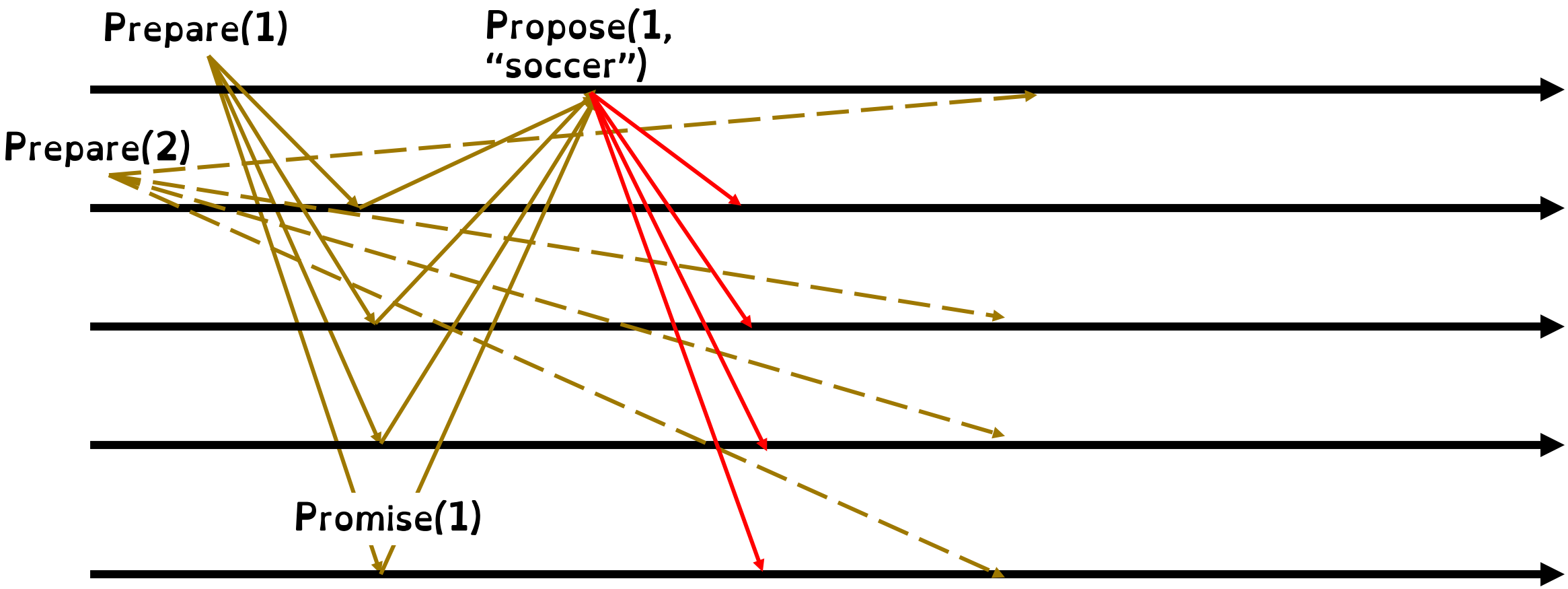
Agreement: All processes that decide do so on the same value

Validity: Value decided must have proposed by some process

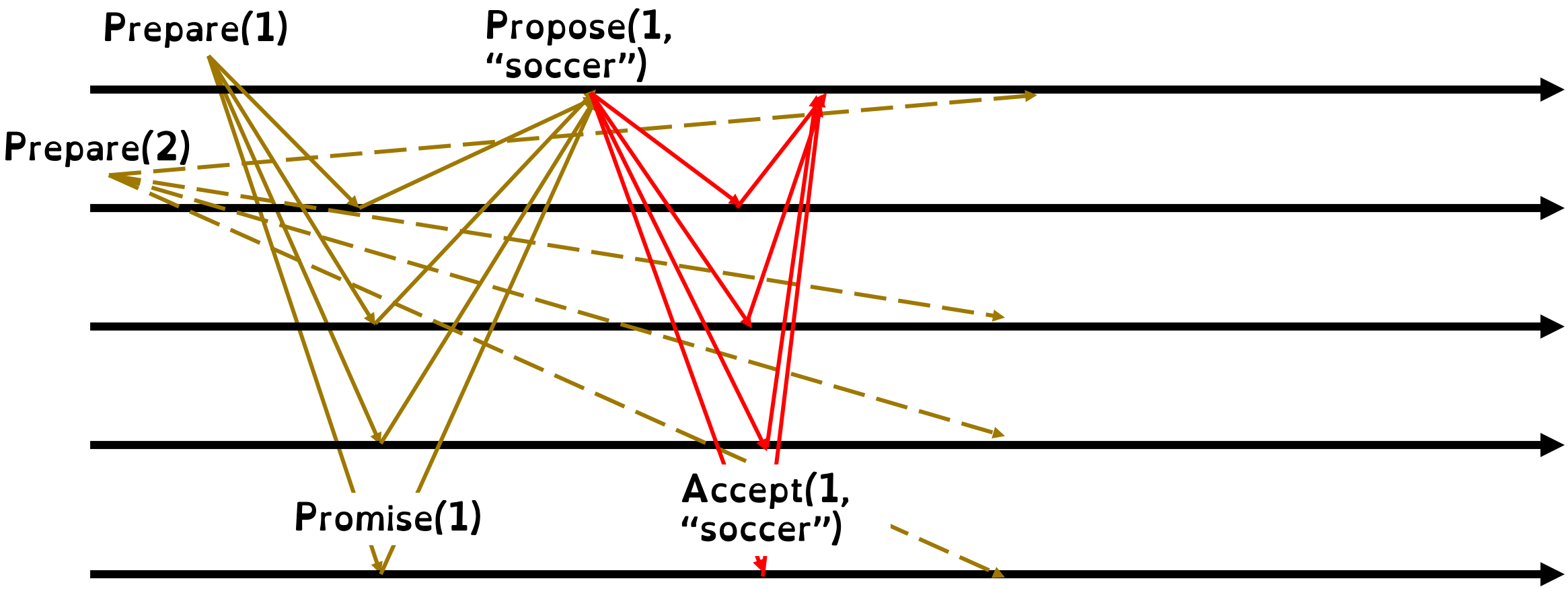
Harder Example



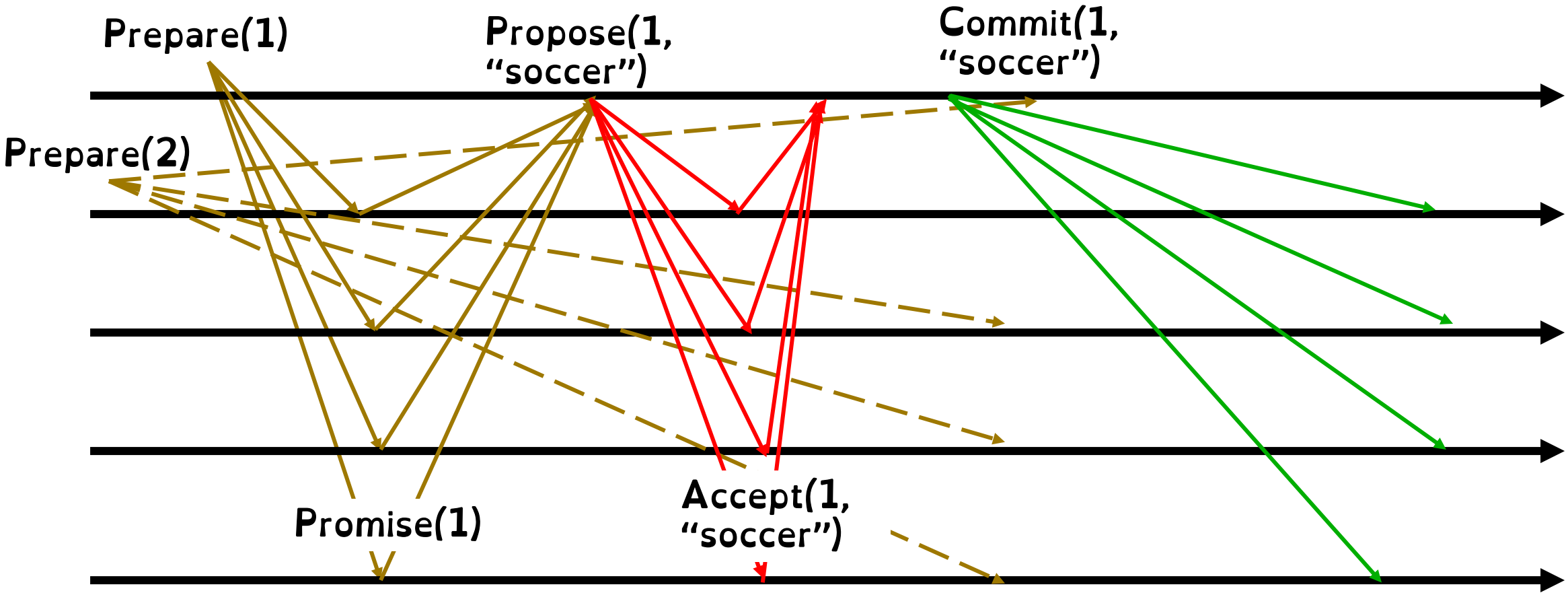
Harder Example



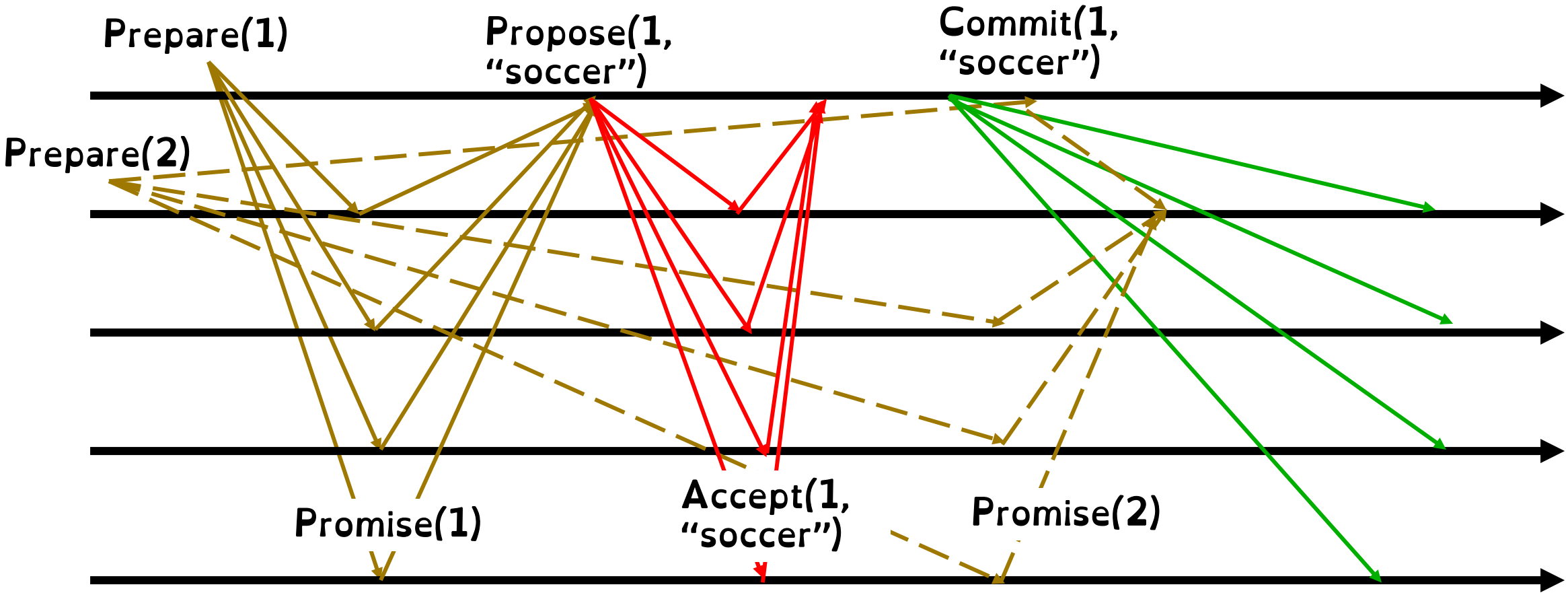
Harder Example



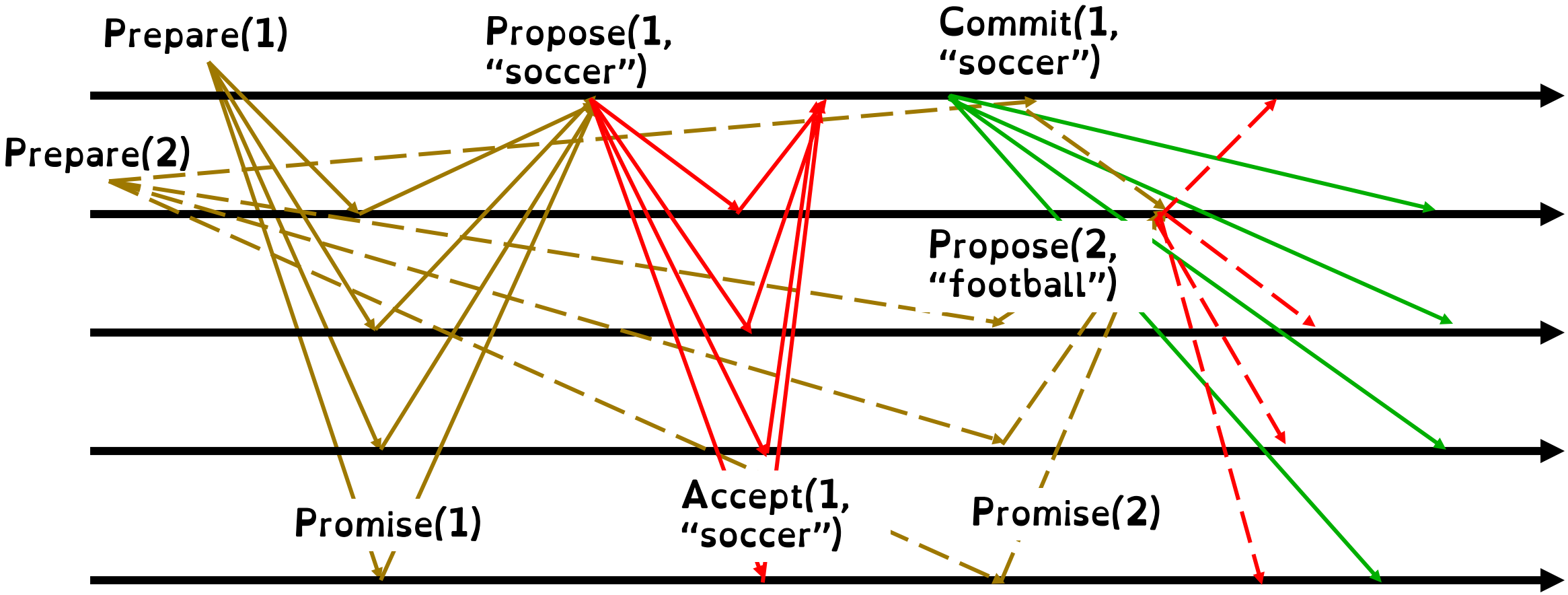
Harder Example



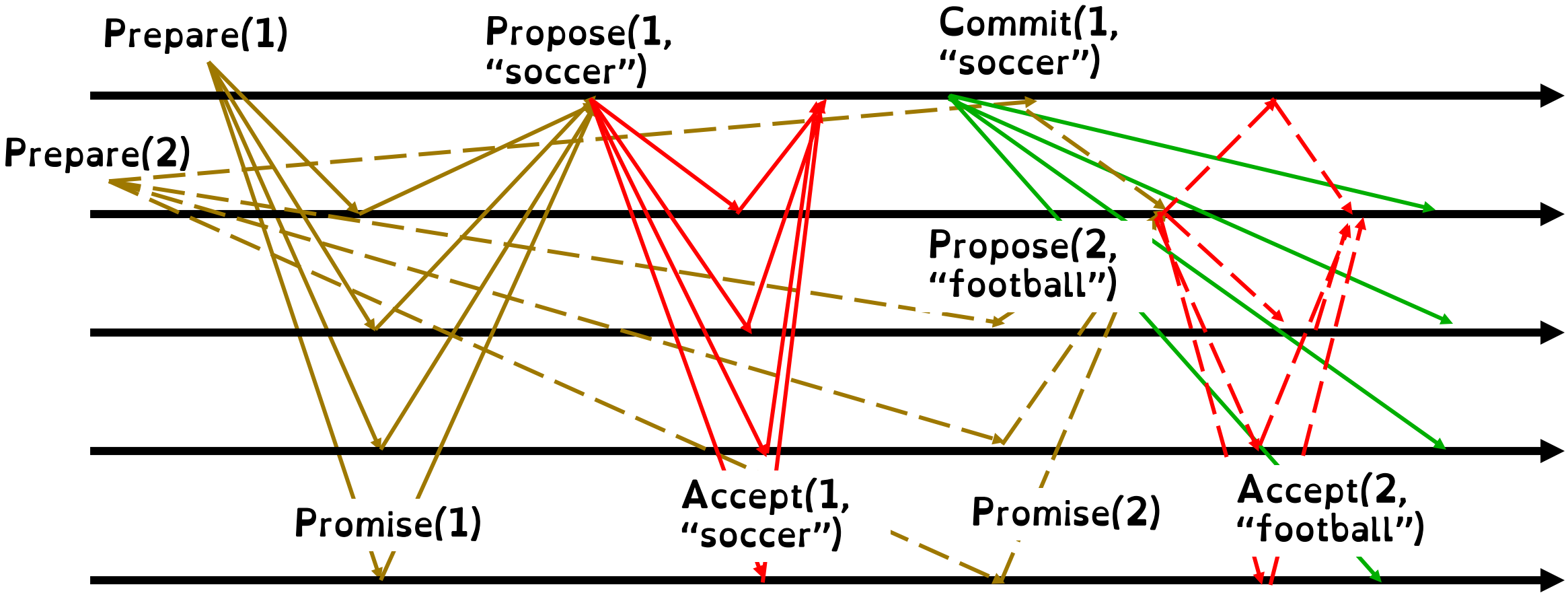
Harder Example



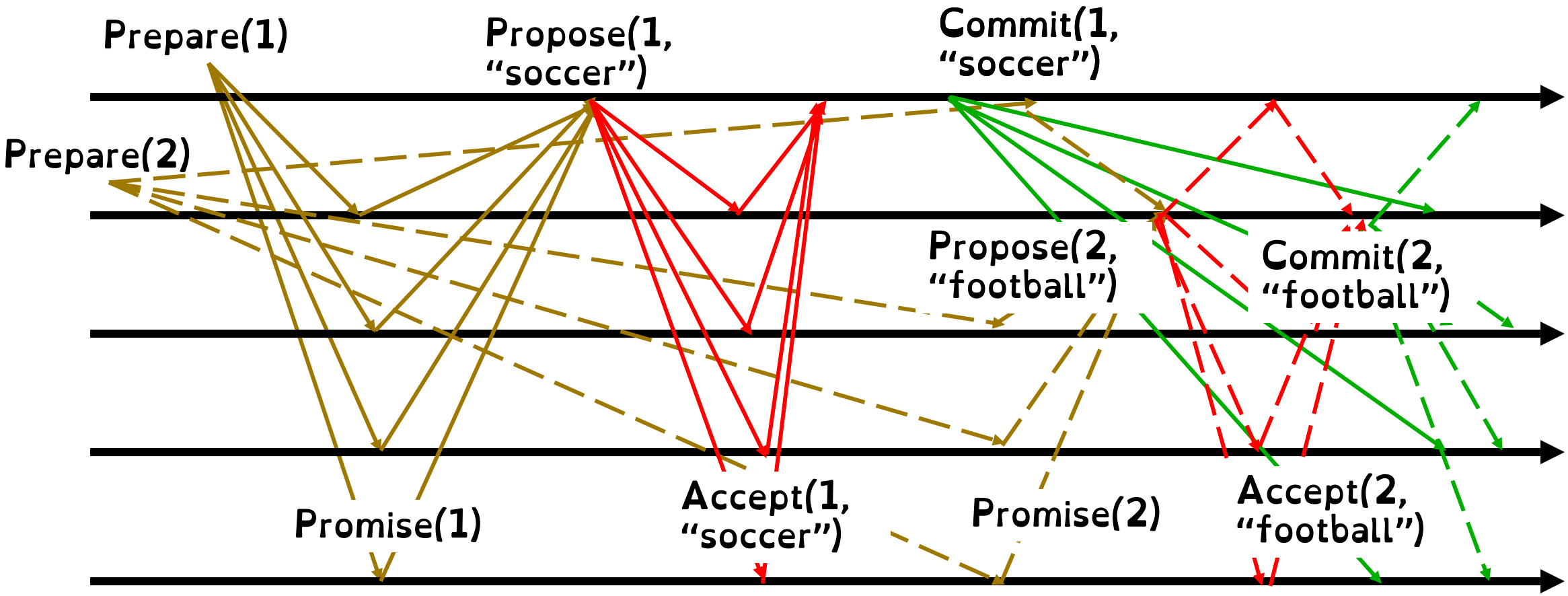
Harder Example



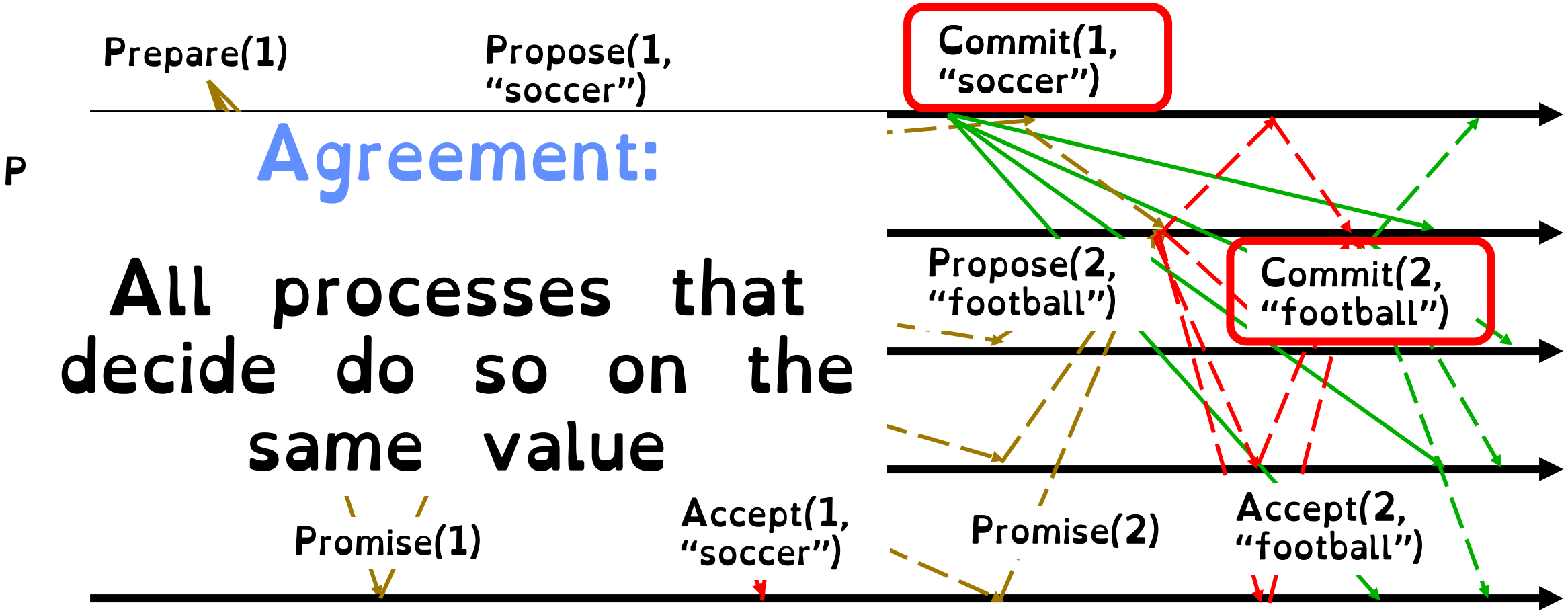
Harder Example



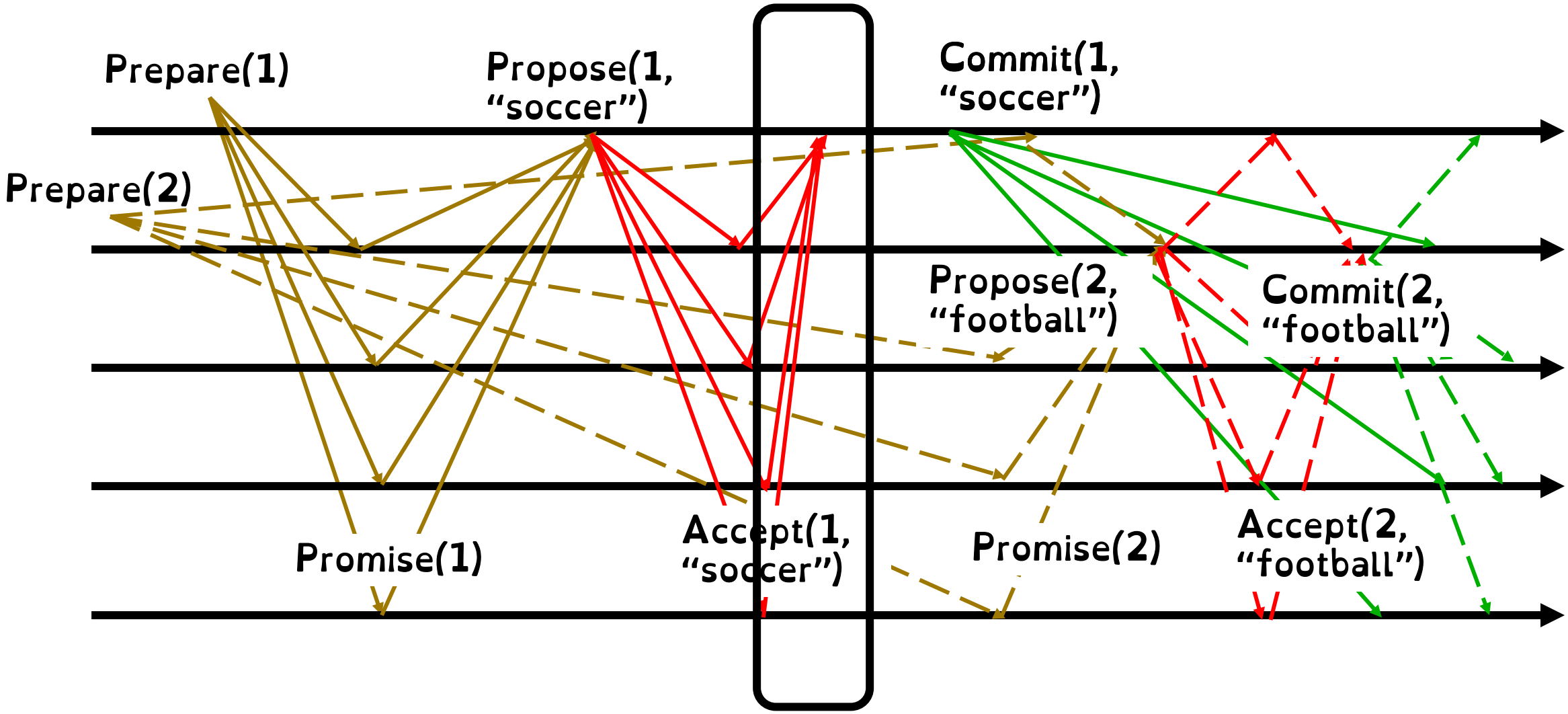
Harder Example



We don't have consensus!



What went wrong?



Consensus happens here! But participants don't know it yet

Phase 1 – Election (Version 1)

Proposer sends PREPARE(ballot_id) to all participants.

If participant has already received a higher ballot id (b > ballot_id), do nothing.

Else:

1) Store b=ballot_id on disk
2) Send an PROMISE(ballot_id) to proposer.

Have I already agreed to ignore proposals with this proposal number?

Phase 1 – Election (Version 2)

Proposer sends **PREPARE**(ballot_id) to all participants.

If highest ballot id received so far, send **PROMISE**(ballot_id)

If already sent an **ACCEPT**(old_ballot,value), send **PROMISE**(ballot_id, (old_ballot,value))

Otherwise do nothing

(Log Decision)

Have I already agreed to ignore proposals < ballot_id
Have I already potentially decided a value?

Phase 1 – Election (Version 2)

If majority (i.e., quorum) respond PROMISE(ballot_id) then, proposer is the leader.

Can propose any value it wants!

If majority (i.e., quorum) respond PROMISE(ballot_id, (old_ballot, v))

Then, select v with highest "old_ballot" value.
Must propose v

Leader not free to choose value as consensus may already have been reached!

Phase 2 – Proposal (Bill) (Version 1)

Leader sends proposed value v by sending PROPOSE(ballot_id,v) to all

If participant has already received a higher ballot id (b > ballot_id), do nothing.

Else:

1) Store b=ballot_id on disk
2) Send an ACCEPT(ballot_id, v) to proposer.

Have I already agreed to ignore proposals with this proposal number?

Phase 2 – Proposal (Bill) (Version 2)

Leader sends proposed value v by sending PROPOSE(ballot_id,v) to all

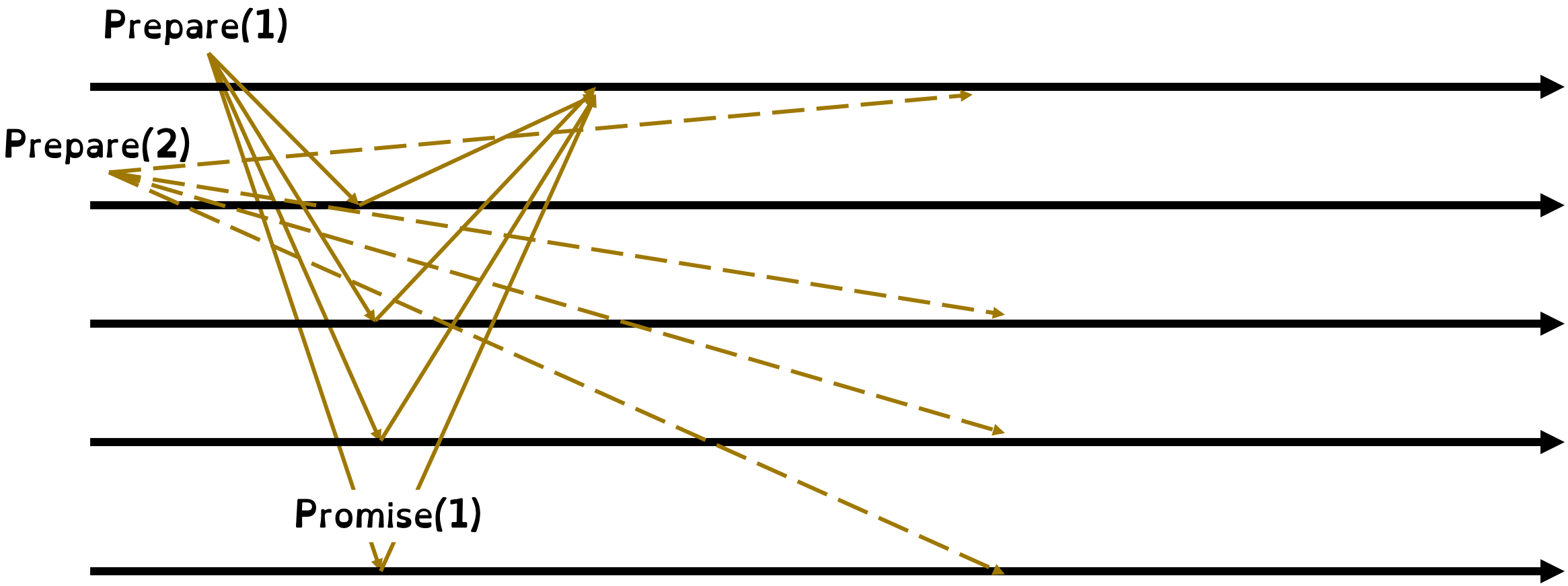
(where v is either leader's value or result of PROMISE message)

If participant has already received a higher ballot id (b > ballot_id), do nothing.

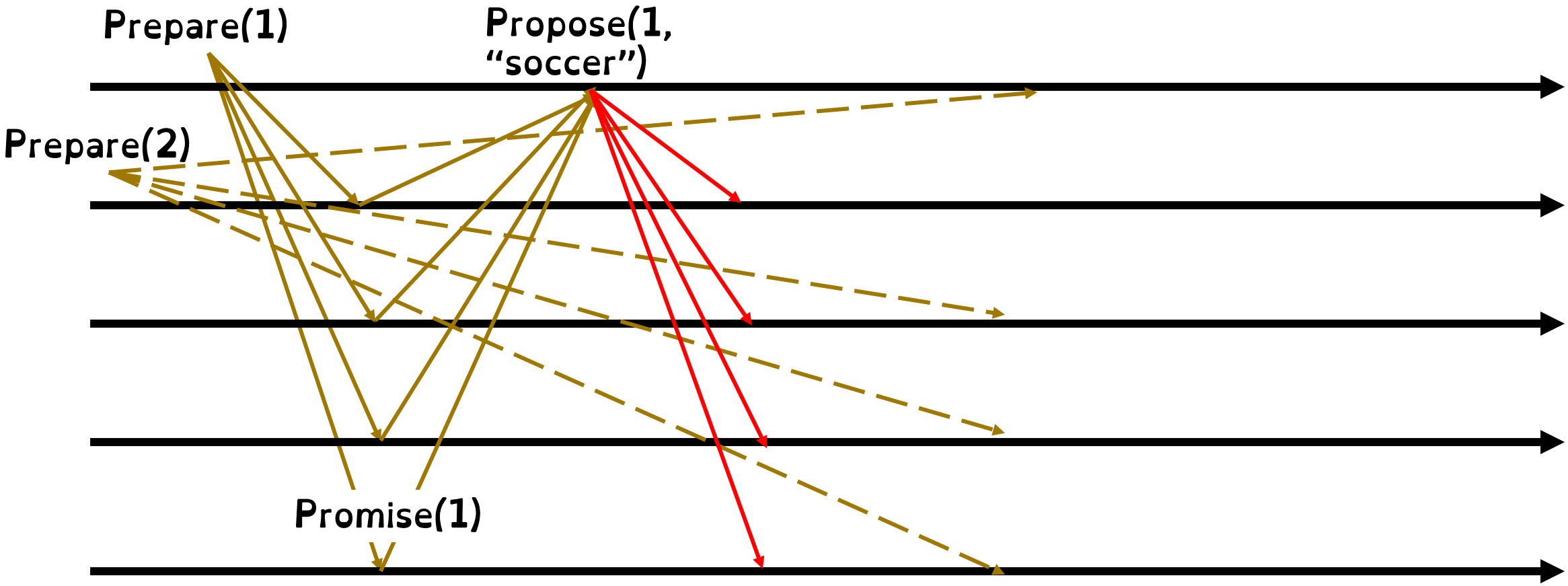
Else:

1) Store b=ACCEPT(ballot_id, v) on disk
2) Send an ACCEPT(ballot_id, v) to proposer.

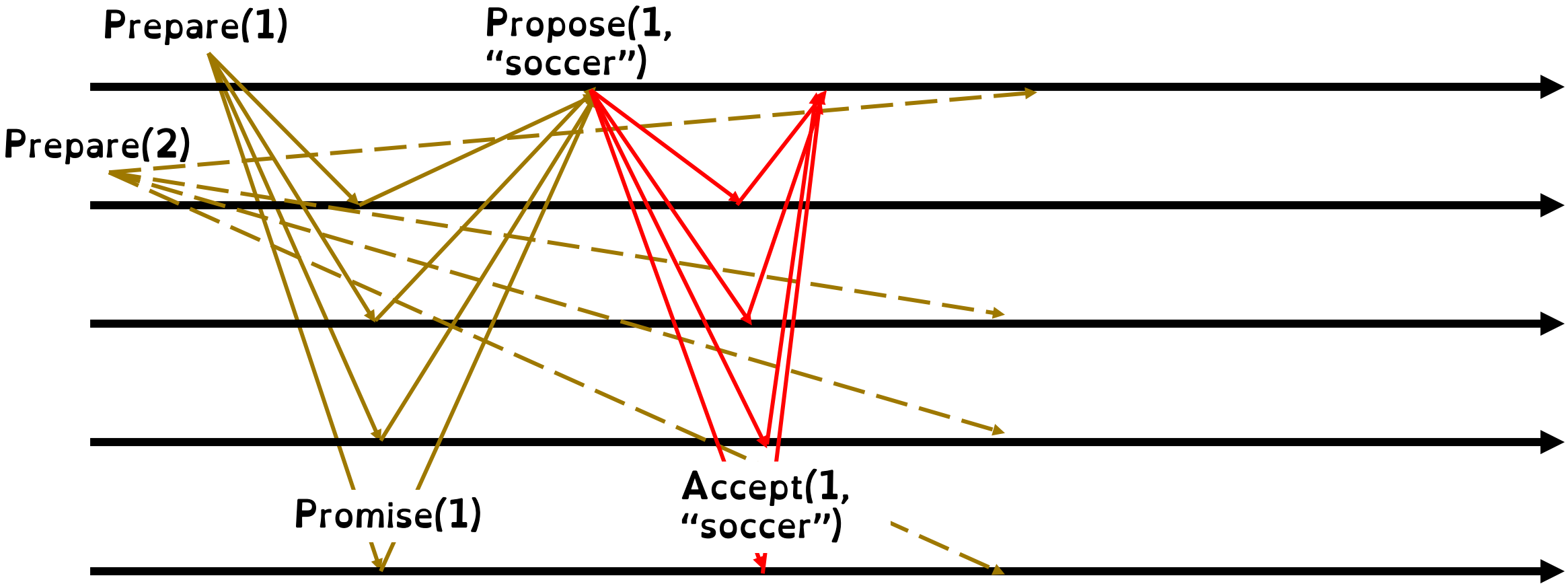
Harder Example (v2)



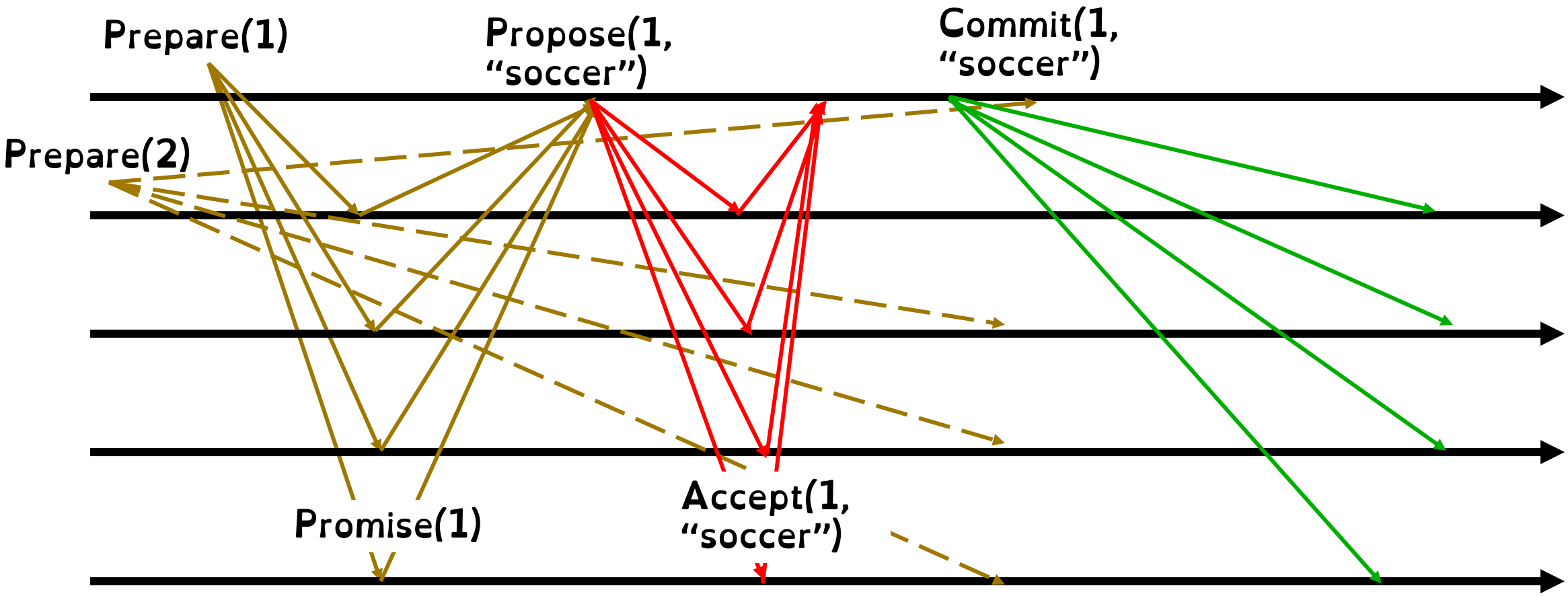
Harder Example (v2)



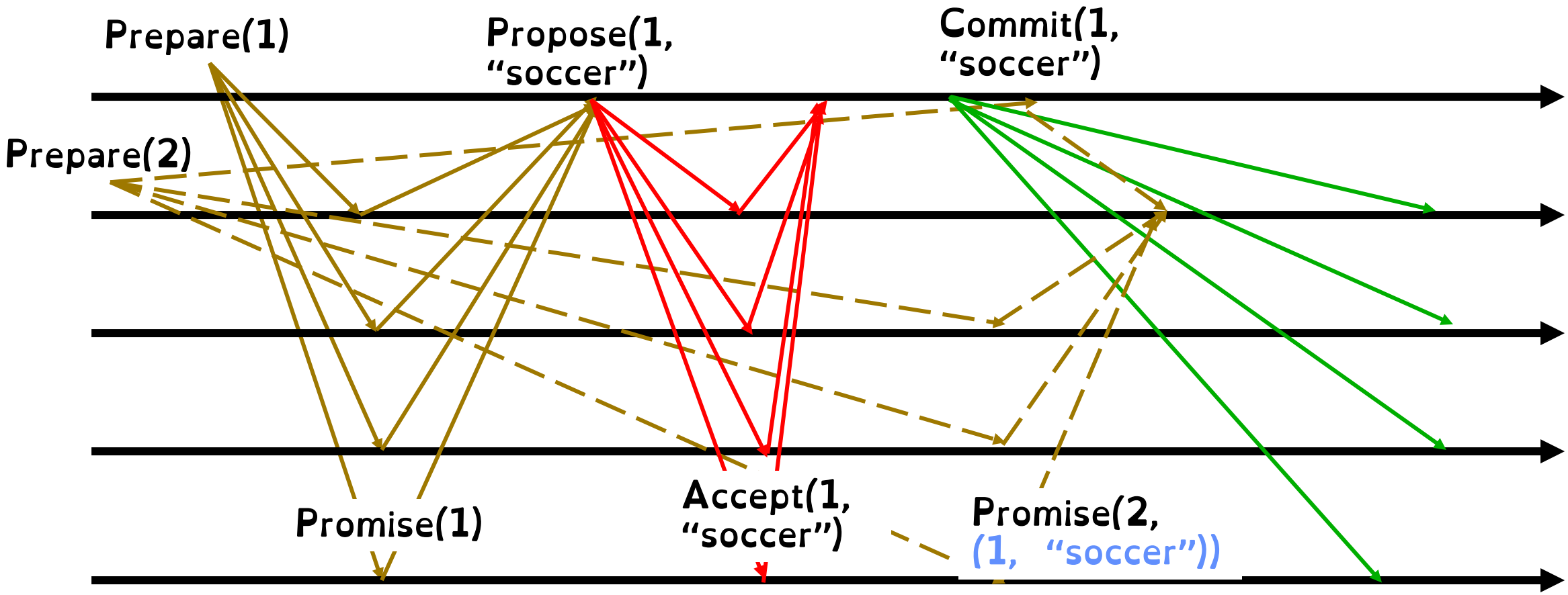
Harder Example (v2)



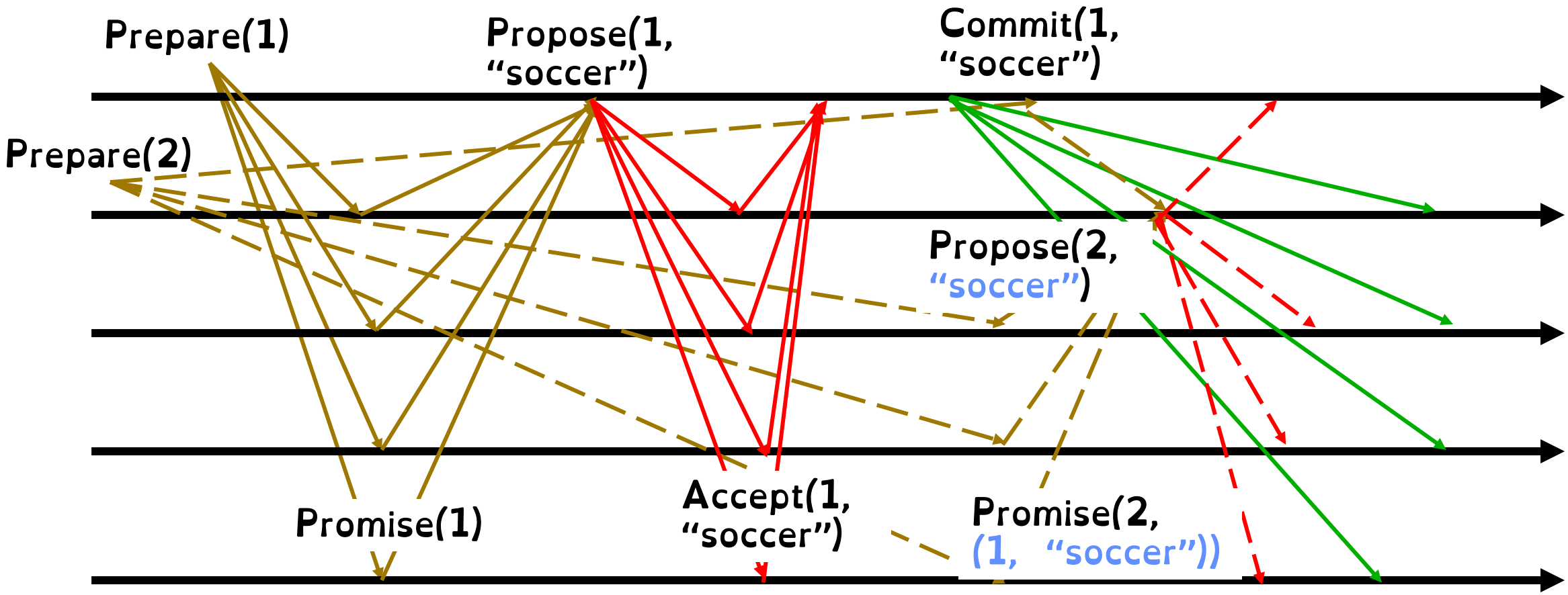
Harder Example (v2)



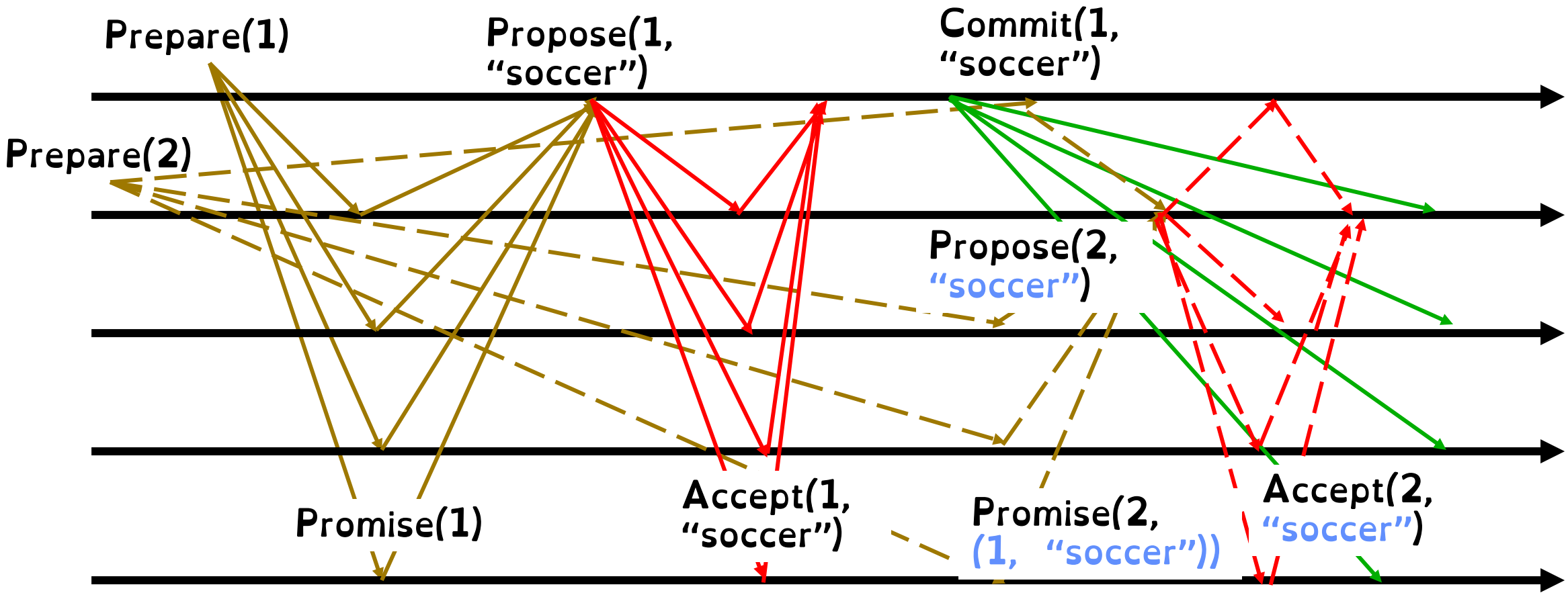
Harder Example (v2)



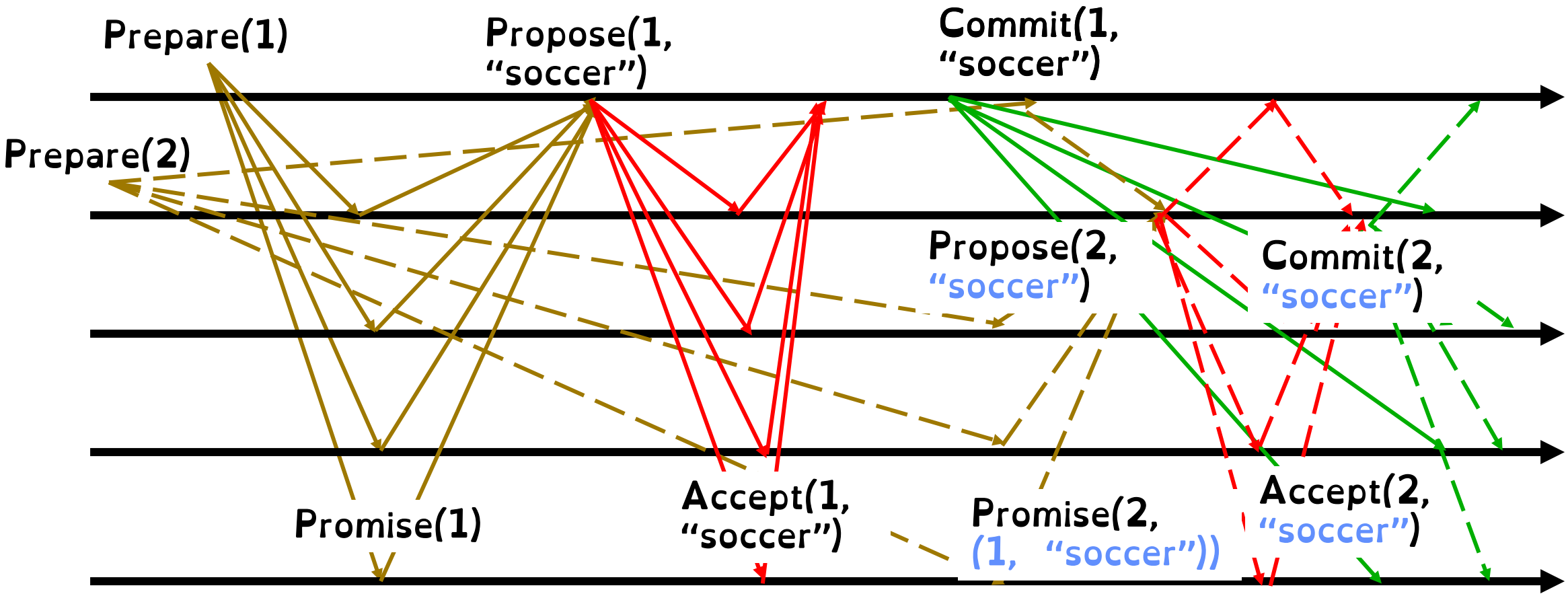
Harder Example (v2)



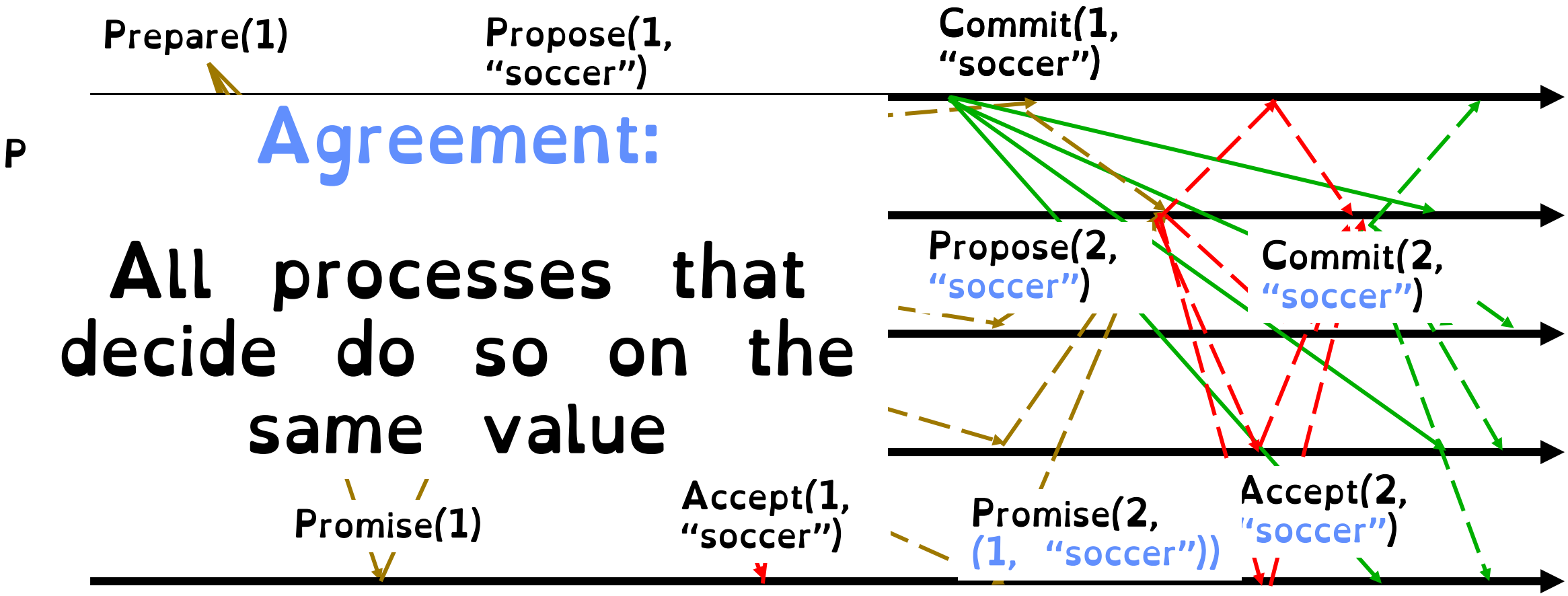
Harder Example (v2)



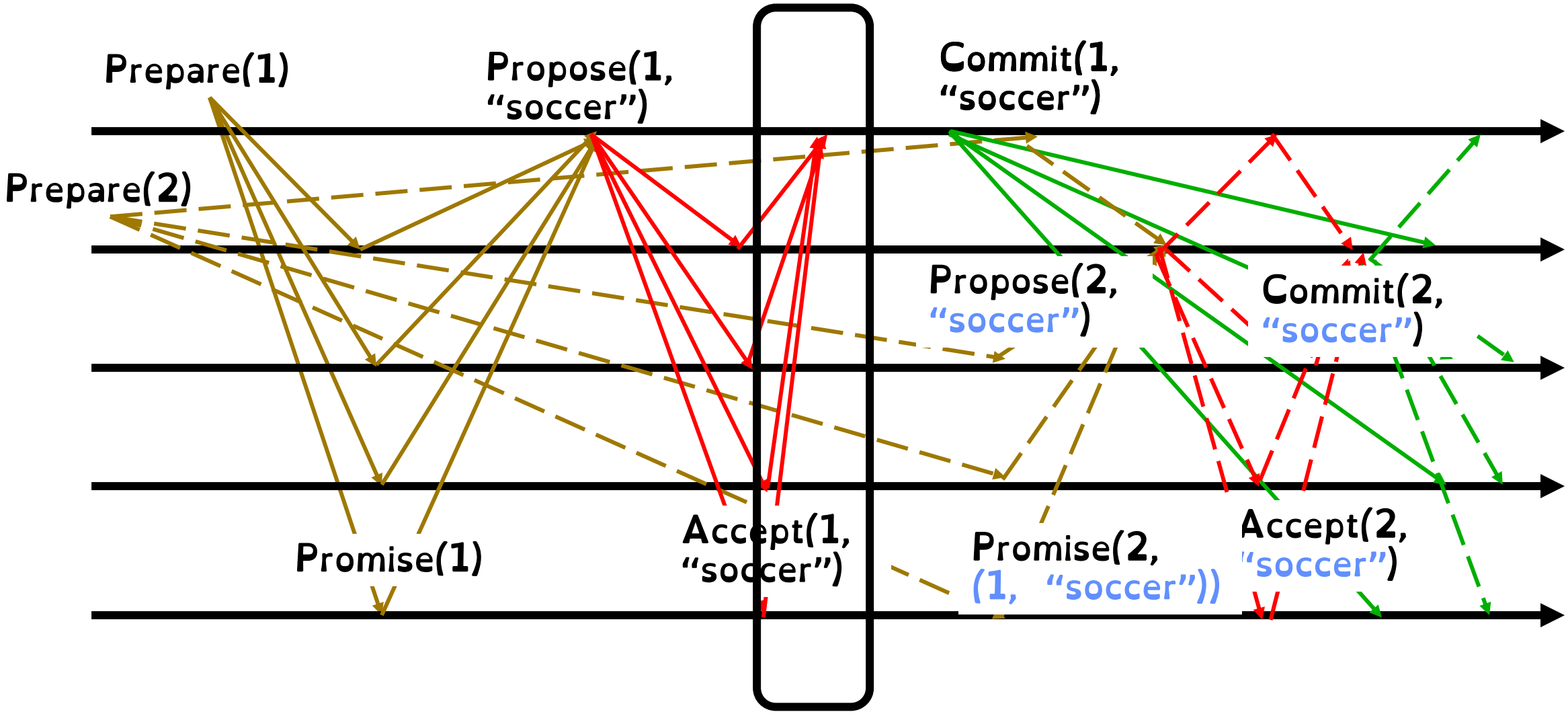
Harder Example (v2)



We do have consensus!

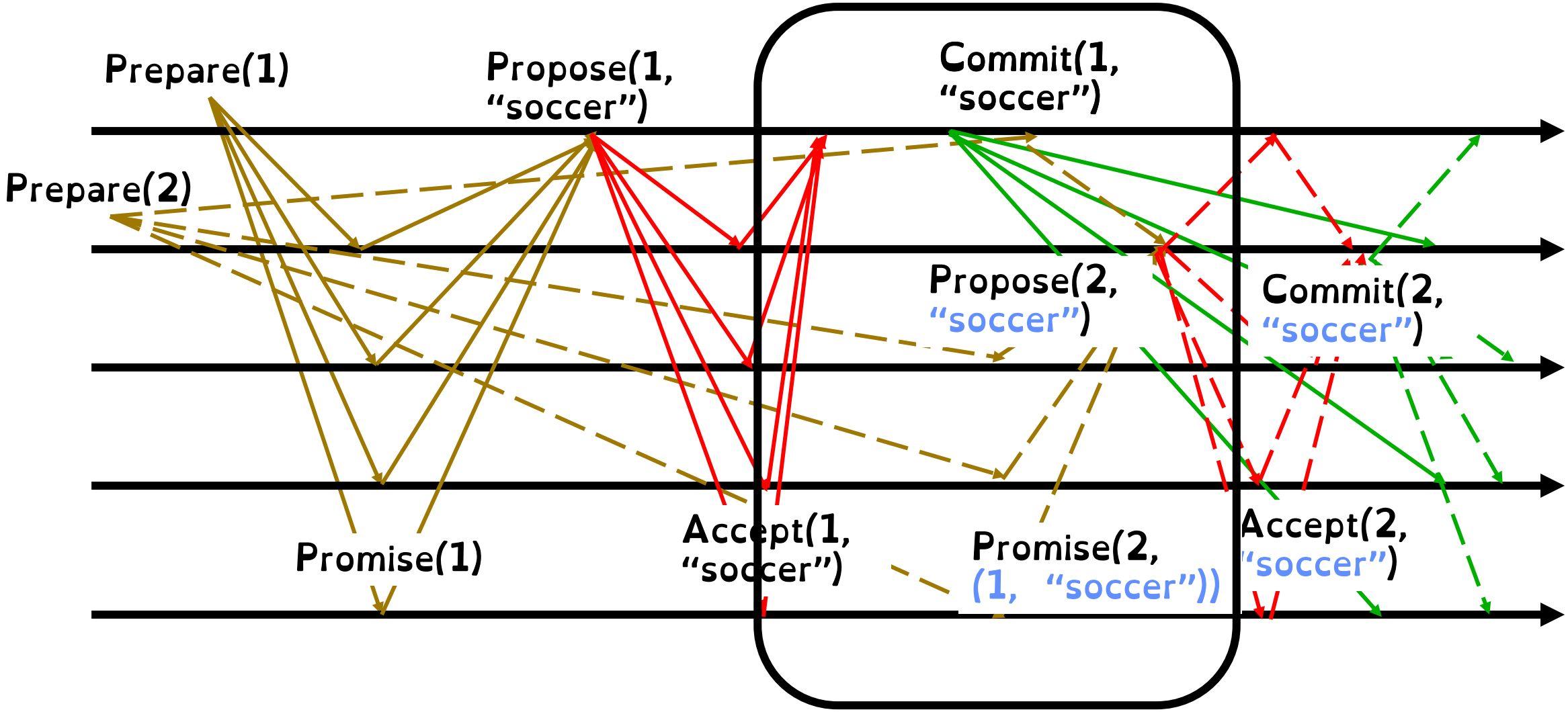


We do have consensus!



Consensus happens here! But participants don't know it yet

We do have consensus!



Because consensus *may* have happened on this value, proposer must re-propose it

Core Safety Theorem

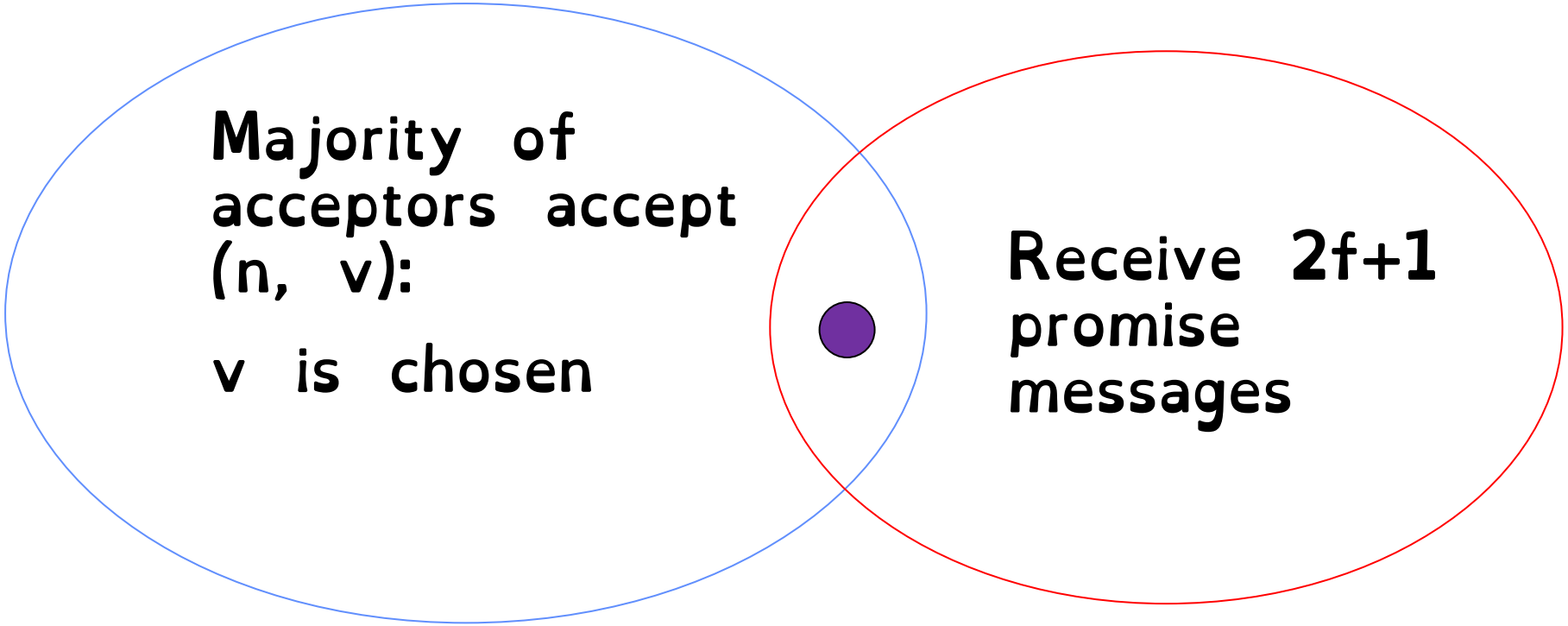
If some round has a majority (i.e., quorum) accepting value v, then subsequently at each round either:

1) the round chooses v as decision or

2) the round fails

Recall that cannot prove liveness!
(Think of proposers livelocking)

Core Safety Theorem Proof Intuition



If 2f+1 participants accepted v in round r

for all rounds r'>r, proposer will receive at least one PROMISE(r', (r,v))

Coordination – Paxos Summary

Decide a single value at once. Always safe, mostly live.

Three phases. Eventual (not simultaneous) agreement

Real implementations of Paxos decide on a "log" (MultiPaxos, Viewstamp Replication)

Topic roadmap

Distributed File Systems

Peer-To-Peer System:
The Internet

Distributed Data Processing

Coordination
(Atomic Commit and
Consensus)

Topic Breakdown

- Virtualizing the CPU
  - Process Abstraction and API
  - Threads and Concurrency
  - Scheduling
- Virtualizing Memory
  - Virtual Memory
  - Paging
- Persistence
  - IO devices
  - File Systems
- Distributed Systems
  - Challenges with distribution
  - Data Processing & Storage


"That's all Folks!"