Week 5
Spyros Mastorakis

Outline

- Paxos

Implementing RSMs

- Logical clock based ordering of requests
  - Cannot serve requests if any one replica is down
- Primary-backup replication
  - Replace primary/backup upon failure

Availability of primary-backup RSM

- When is RSM unavailable to serve requests?
  - View service is down
  - Primary failed before syncing with backups
  - Backup is up and connected to view service but not primary
  - Summary: replica is down but we may not be able to move to new view

- How to…
  - make RSM tolerant to network partitions?
  - ensure that operations don’t block even if some machines are unavailable?

Analogy

- US Senate needs to pass laws

- Senators are often on travel
  - Common case: not all senators present

- How to pass laws successfully?
  - Goal: ensure that, despite only a subset showing up each day, that if law gets passed, it won’t be overwritten

RSM via Consensus

- Key idea: apply update only if majority of replicas commit to it
  - If 2f + 1 replicas, we need f+1 to commit
  - Tolerate more failures → have more replicas

Paxos Context

- Assumption: replicas start in sync with one another
- First challenge: among several concurrent new updates, how to pick next update to apply?
- Next challenge: how to apply all updates in a consistent order all replicas

Desirable Properties

- Safety
  - “No bad things happen”
  - System never reaches an undesirable state
- Liveness
  - “Good things eventually happen”
  - System makes progress eventually
- Tradeoff between consistency and latency

Roles of a Process

- Three conceptual roles
  - Proposers propose values
  - Acceptors accept values; chosen if majority accept
  - Learners learn the outcome (chosen value)

- In reality, a process can play any/all roles

Paxos: Key Ideas

- May not be able to come to consensus in a single round
  - Protocol runs over multiple rounds

- Once a value is accepted by a majority, a different value cannot be accepted in a later round

Paxos Overview

- Three phases in each round

- Prepare Phase
  - Proposer sends a unique proposal number to acceptors
  - Waits to get commitment from majority of acceptors

- Accept Phase
  - Proposer sends proposed value to acceptors
  - Waits to get proposal accepted by majority

- Learn Phase
  - Learners discover value accepted by majority

Paxos State

- Every acceptor maintains three values:
  - n_p → highest proposal number promised to accept
  - n_a → highest proposal number accepted
  - v_a → value accepted

- This state must persist across restarts
- Learners can rediscover accepted value (if any) from acceptors

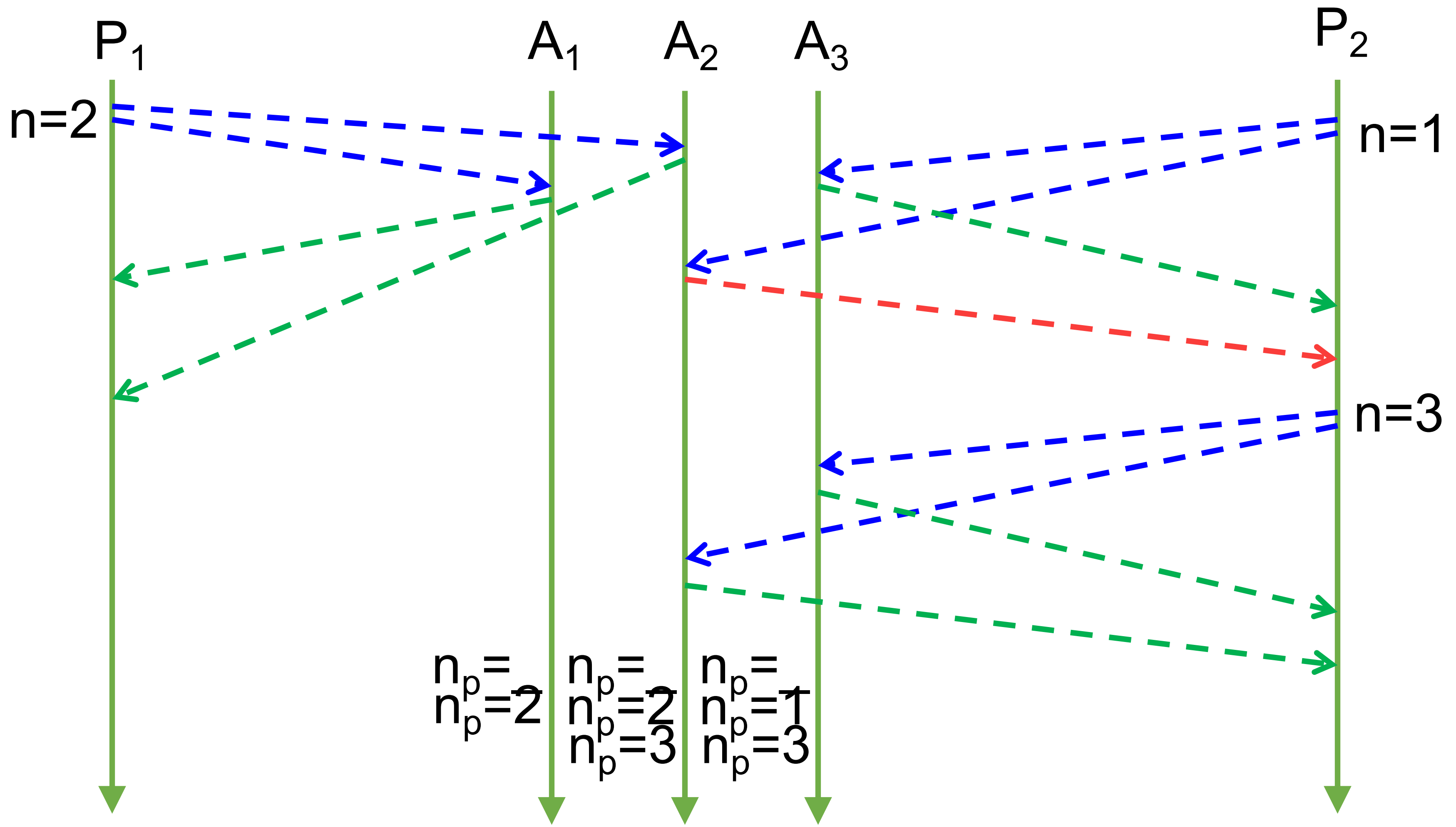
Paxos: Phase 1 (Prepare)

- Proposer
  - Choose unique proposal number n
  - Send <prepare, n> to all acceptors

- Acceptors
  - If $n > n_p$
    - $n_p = n$ ← promise not to accept any new proposal where $n' < n$
    - If no prior proposal accepted → reply with <promise, n, Ø>
    - Else → reply with <promise, n, (n_a, v_a)>

  - Else
    - Reply with <prepare-failed>

Example: Prepare Phase

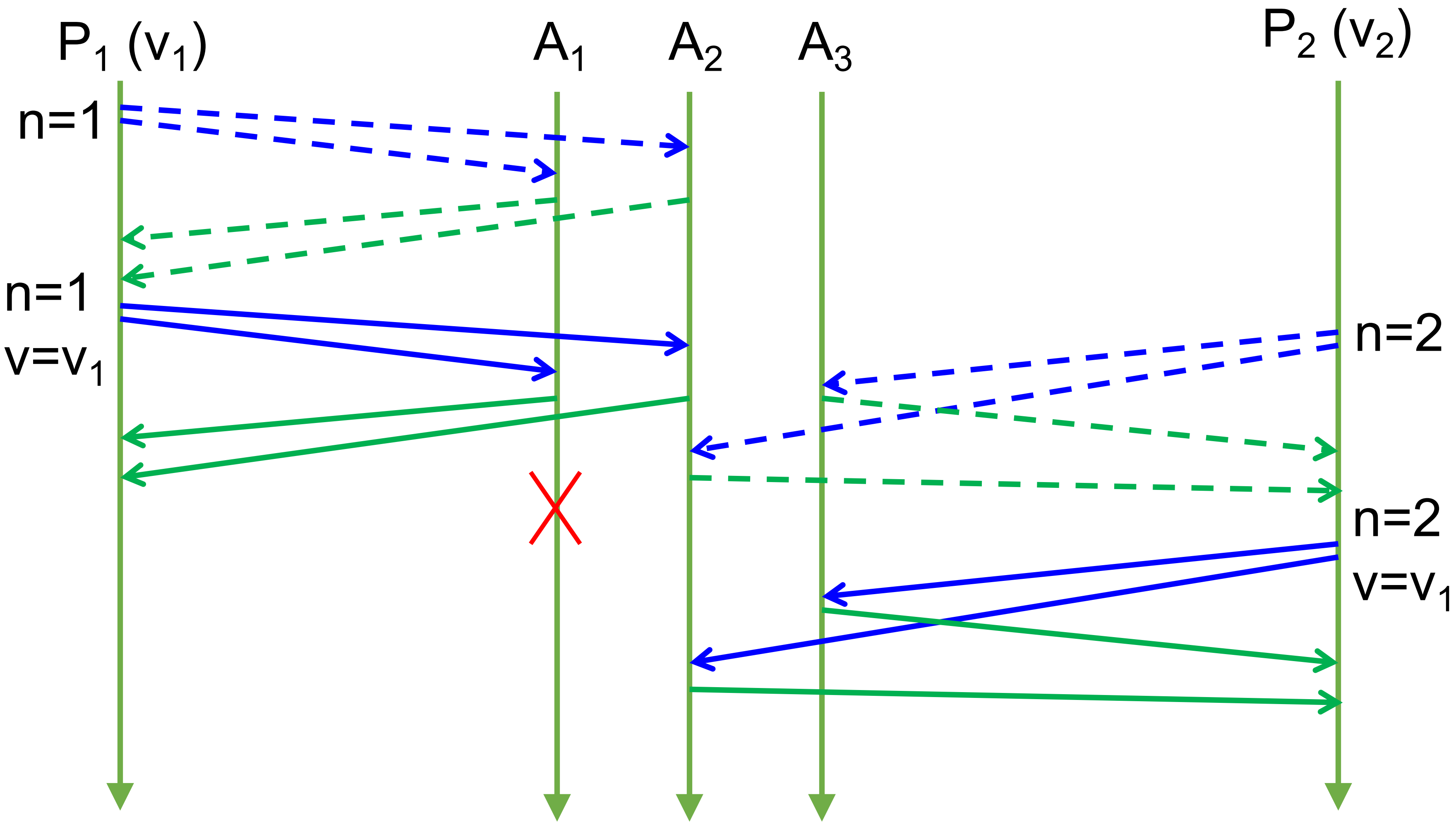


Paxos: Phase 2 (Accept)

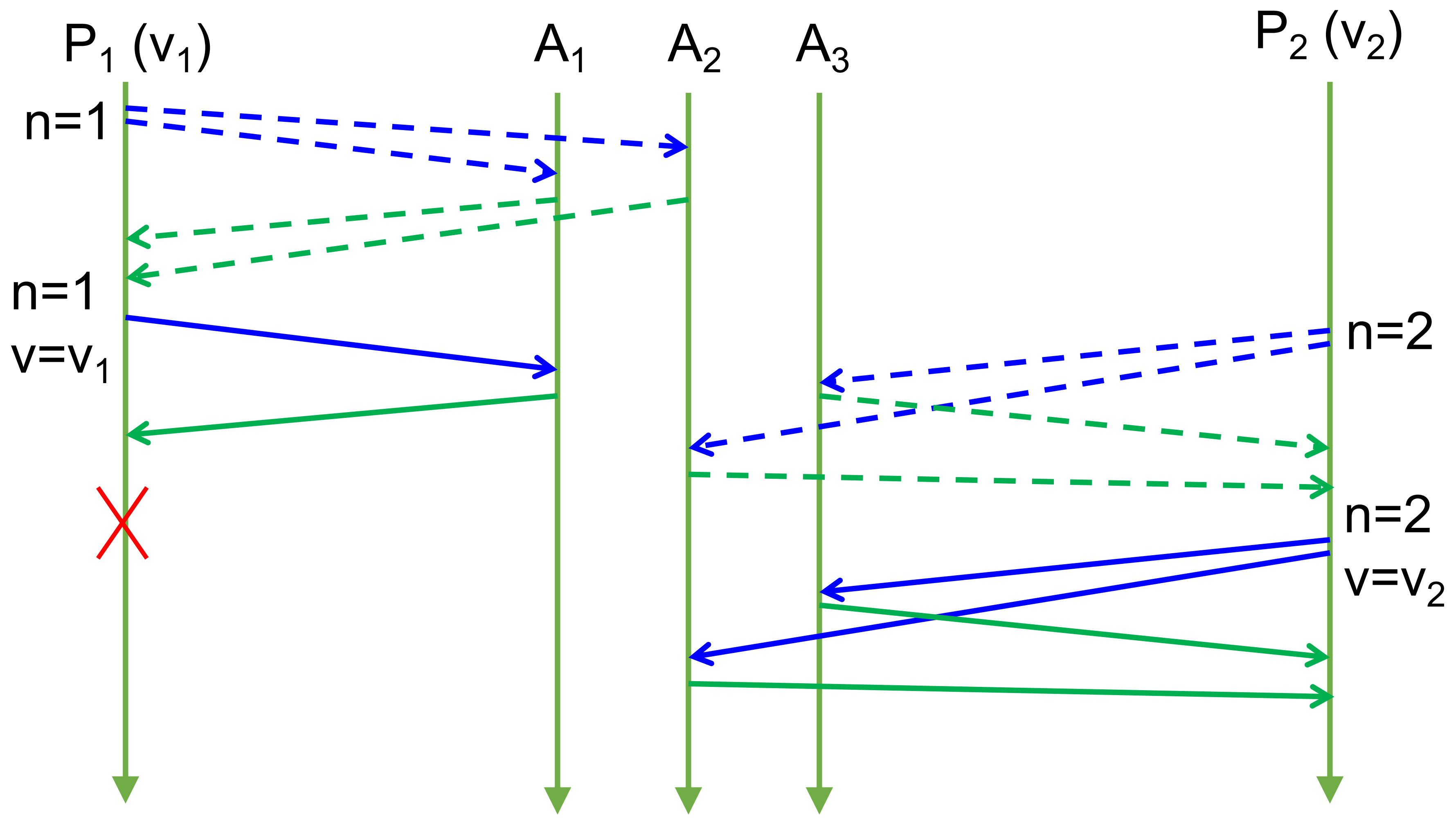
- Proposer
  - Once it has received promises from a majority of acceptors:
    - $v' = v_a$ returned with highest n_a, if exists, else own v
    - Send <accept, (n, v')> to acceptors

- Acceptors
  - Upon receiving (n, v), if $n \geq n_p$
    - Accept proposal and notify learner(s)
    - $n_a = n_p = n$
    - $v_a = v$

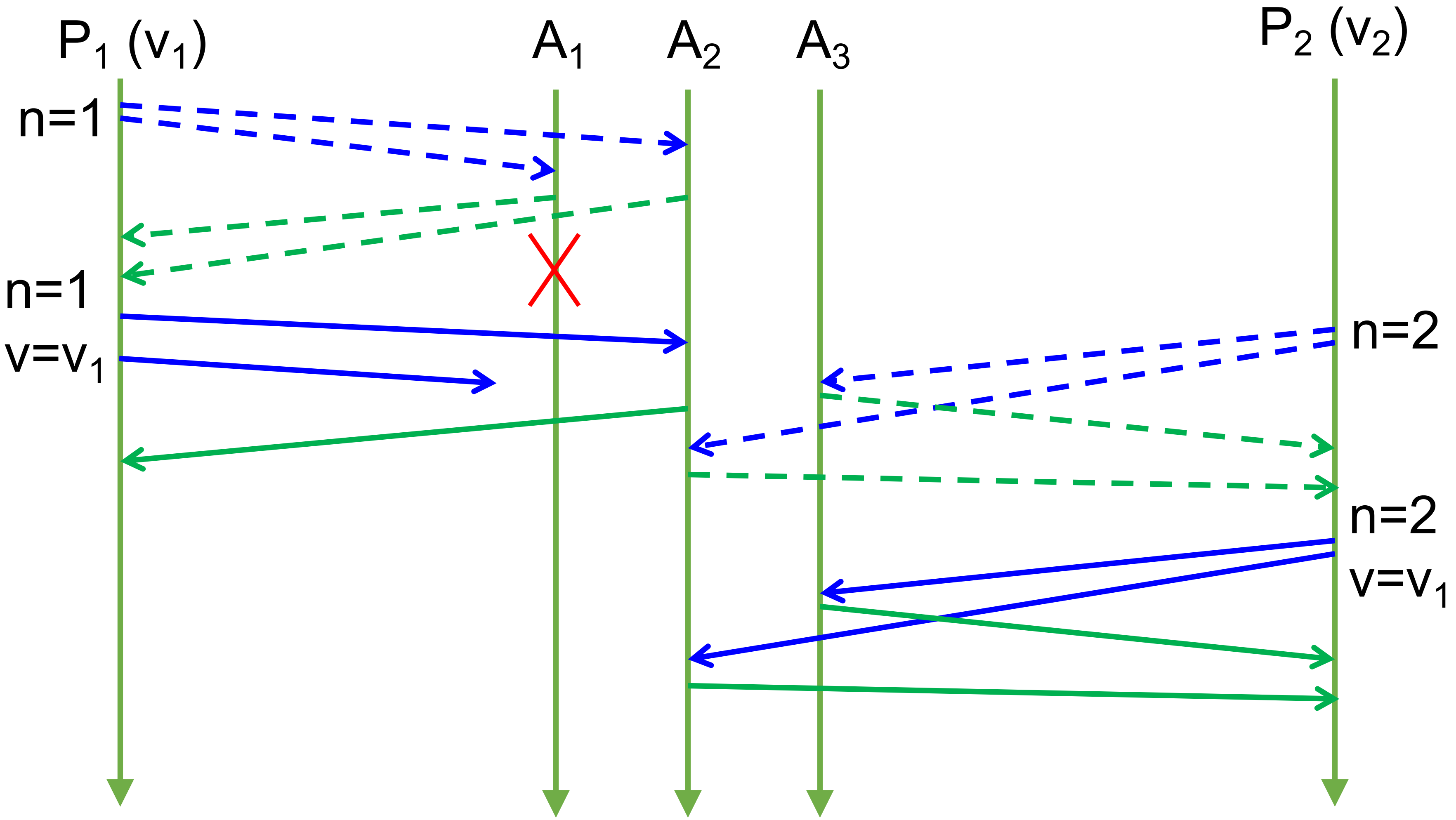
Example: Accept Phase



Example 2: Accept Phase



Example 3: Accept Phase



Paxos: sample execution

- Acceptor1: P1 A1-X P2
- Acceptor2: P1 A1-X P2 P3
- Acceptor3: P1 A1-X P3

Paxos: sample execution

- Acceptor1: P1 A1-X P3 A3-X
- Acceptor2: P1 P2 A1-X P3 A2-Y A3-X
- Acceptor3: P2 A2-Y

Paxos: sample execution

- Acceptor1: P1 A1-X
- Acceptor2: P1 A1-X P2
- Acceptor3: P2

Paxos: sample execution

• Acceptor1:	P1	A1-X		
• Acceptor2:	P1	A1-X	P2	A2-X
• Acceptor3:			P2	A2-X

Paxos: sample execution

- Acceptor1: P1
- Acceptor2: P1 A1-X P2
- Acceptor3: P2

Paxos: sample execution

- Acceptor1: P1
- Acceptor2: P1 A1-X P2 A2-X
- Acceptor3: P2 A2-X

Paxos: sample execution

- Acceptor1: P1 A1-X
- Acceptor2: P1 P2
- Acceptor3: P2

Paxos: sample execution

- Acceptor1: P1 A1-X
- Acceptor2: P1 P2 A2-Y
- Acceptor3: P2 A2-Y

Race Condition Prevents Liveness

Proposer 1 | **Proposer 2**

Completes phase 1 with proposal n0

Starts and completes phase 1 with proposal n1 > n0

Performs phase 2, acceptors reject

Retries and completes phase 1 with proposal n2 > n1

Performs phase 2, acceptors reject

… can go on indefinitely …

Paxos: Race Condition

- Acceptor1: P1 A1-X P3
- Acceptor2: P1 P2 A1-X P3 A2-Y P4
- Acceptor3: P2 A2-Y P4

How to fix this?

Liveness Solutions

- When proposal fails, back off for a random period of time before retrying
- Proposers pick some random value within some interval and back off for this period of time
  - Why not having every proposer pick the same back off interval?

Paxos: Phase 3 (Learn)

- Goal
  - Have all learners discover if any value was accepted by majority

- Potential approaches
  - Proposer who has proposal accepted by majority of acceptors informs all learners
  - Acceptor broadcasts to all learners whenever it accepts any value
  - Acceptors notify distinguished learner, which informs others