NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 31: Byzantine Agreement Protocols

CONCEPTS COVERED

- **Practical Byzantine Fault Tolerance (PBFT)**

KEYWORDS

- PBFT Algorithm
- Quorum in PBFT

BFT Consensus

- **Lamport-Shostak-Peas Algorithm***
    - Synchronous environment
    - Reliable communication channel
    - Fully Connected Network
    - Receivers always know the identity of the Senders

* LAMPORT, LESLIE, ROBERT SHOSTAK, and MARSHALL PEASE. "The Byzantine Generals Problem." *ACM Transactions on Programming Languages and Systems* 4.3 (1982): 382-401.

BFT Consensus

- **Lamport-Shostak-Peas Algorithm***
  - Synchronous environment
  - Reliable communication channel
  - Fully Connected Network
  - Receivers always know the identity of the Senders

Unrealistic assumptions for real networks

* LAMPORT, LESLIE, ROBERT SHOSTAK, and MARSHALL PEASE. "The Byzantine Generals Problem." *ACM Transactions on Programming Languages and Systems* 4.3 (1982): 382-401.

BFT Consensus

- Many different variants of BFT Consensus have emerged

- **Practical Byzantine Fault Tolerance (PBFT)****
  - Use cryptographic techniques to release the **unrealistic** assumptions

** Castro, Miguel, and Barbara Liskov. "Practical byzantine fault tolerance." *USENIX OSDI*. Vol. 99. No. 1999. 1999.

Practical Byzantine Fault Tolerance

- **Why Practical?**
    - Considers an asynchronous environment (Gives priority to Safety over Liveness)
    - Utilizes digital signature to validate the identity of the senders
    - Low overhead

Practical Byzantine Fault Tolerance

- Incorporated in a large number of distributed applications including blockchain
  - Tendermint
  - Hyperledger Fabric

- Uses cryptographic techniques to make the messages tamper-proof

PBFT Overview

- Based on State Machine Replication
  - Considers 3F + 1 replicas where F can be the maximum number of faulty replicas

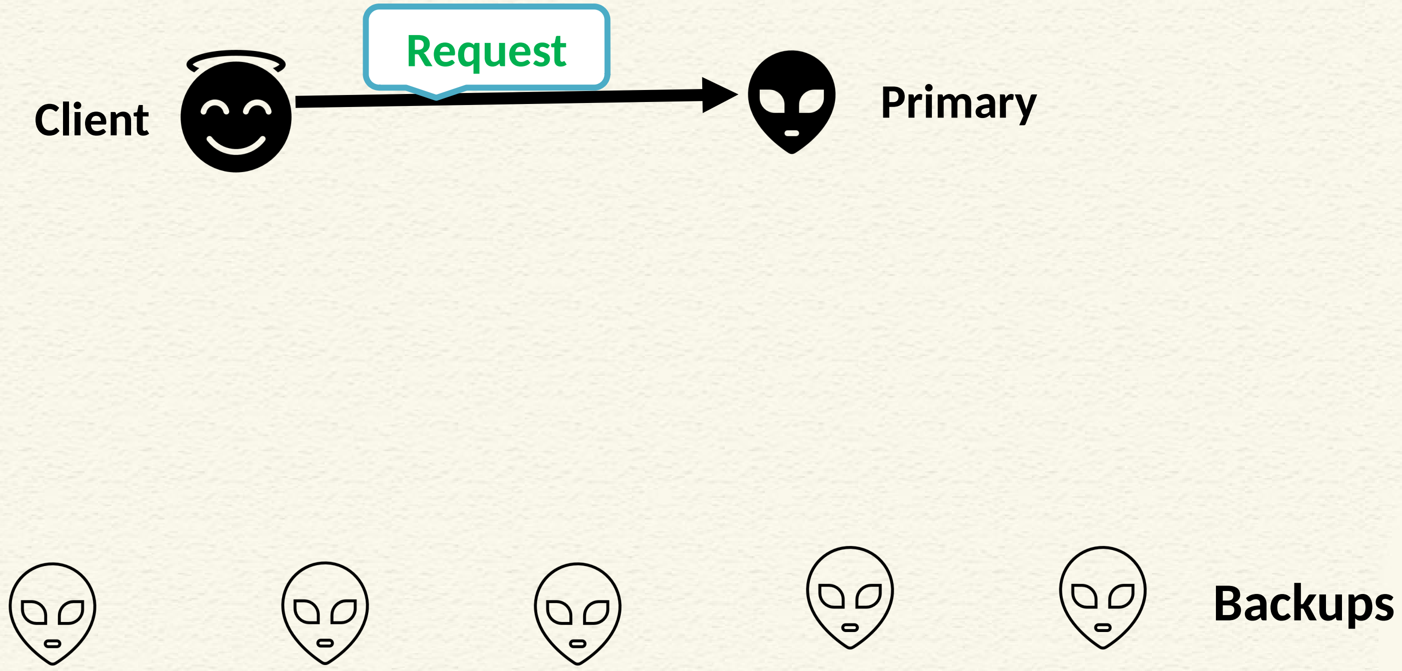
PBFT Overview

- The replicas move through a succession of configurations, known as ***views***
  - One replica in a view is considered as the **primary** (works like a leader), and others are considered **backups**
  - The primary proposes a value (similar to the Proposers in Paxos), and the backups accept the value (similar to the Paxos Acceptors)
  - When the primary is detected as faulty, the view is changed – PBFT elects a new primary and a new view is initiated
  - Every view is identified by a unique integer *v*
  - Only the messages from the current view is accepted

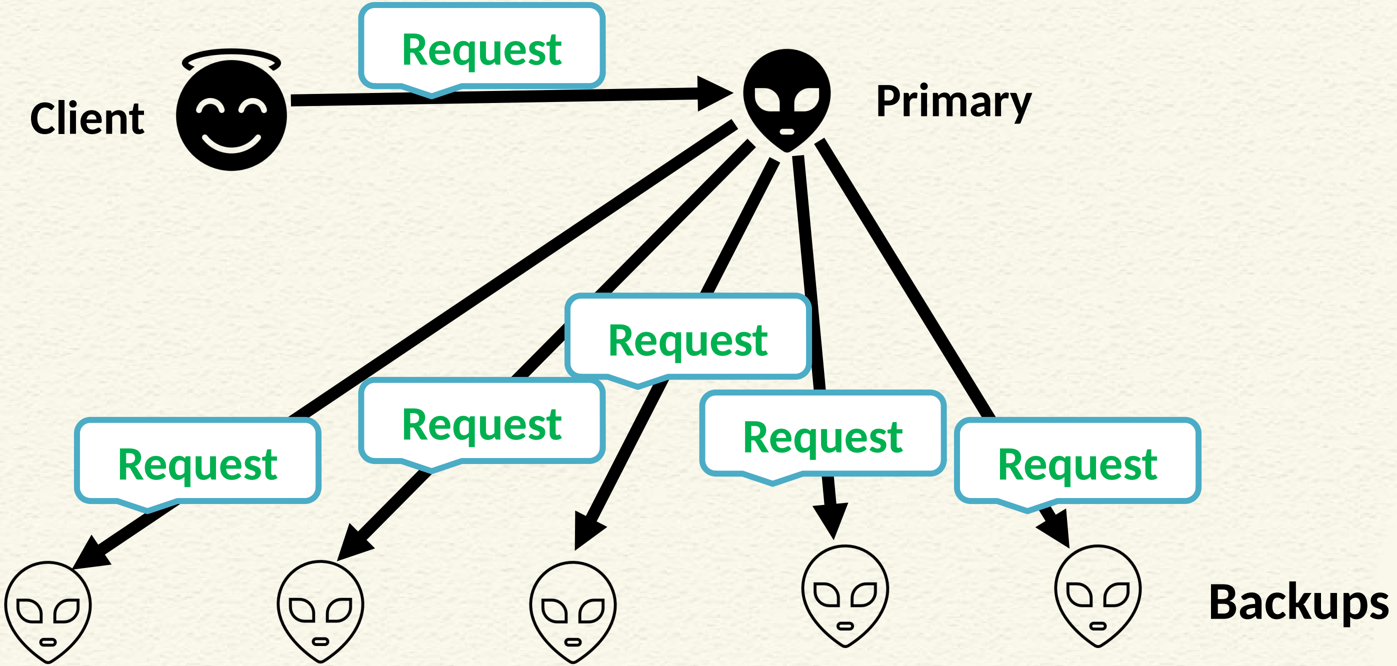
PBFT Overview

- The replicas move through a succession of configurations, known as ***views***
  - One replica in a view is considered as the **primary** (works like a leader), and others are considered **backups**
  - The primary proposes a value (similar to the Proposers in Paxos), and the backups accept the value (similar to the Paxos Acceptors)
  - When the primary is detected as faulty, the view is changed – PBFT elects a new primary and a new view is initiated
  - Every view is identified by a unique integer *v*
  - Only the messages from the current view is accepted

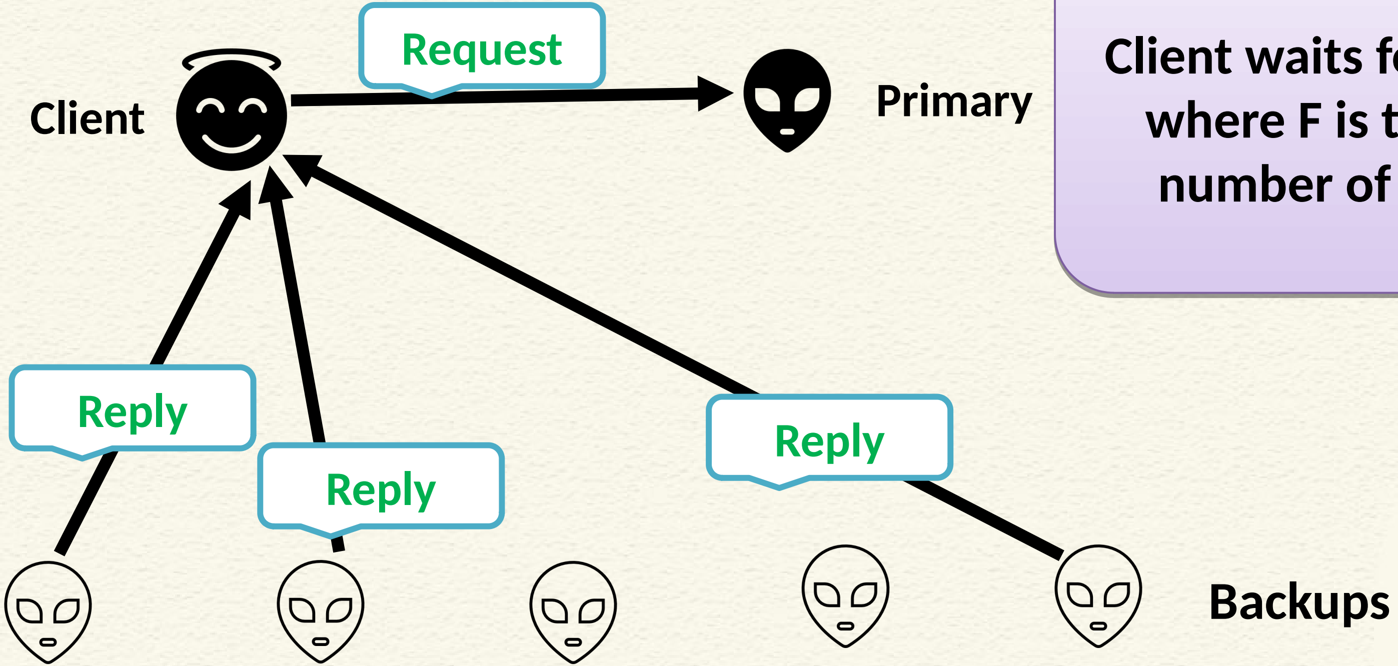
PBFT – Broad Idea



PBFT – Broad Idea

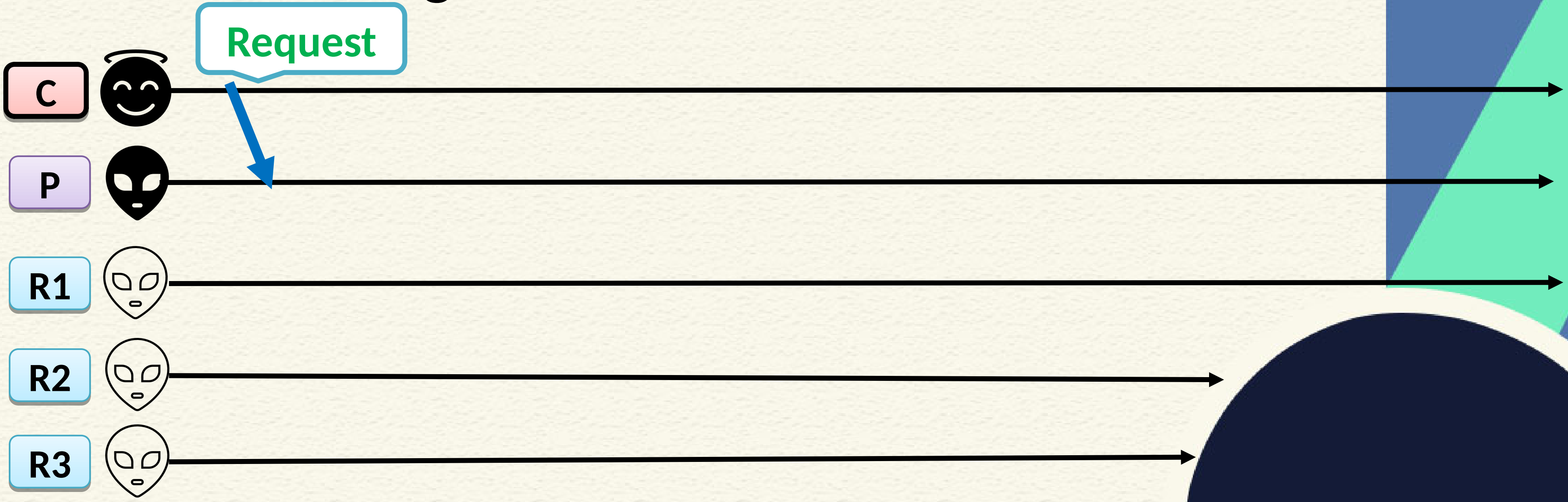


PBFT – Broad Idea



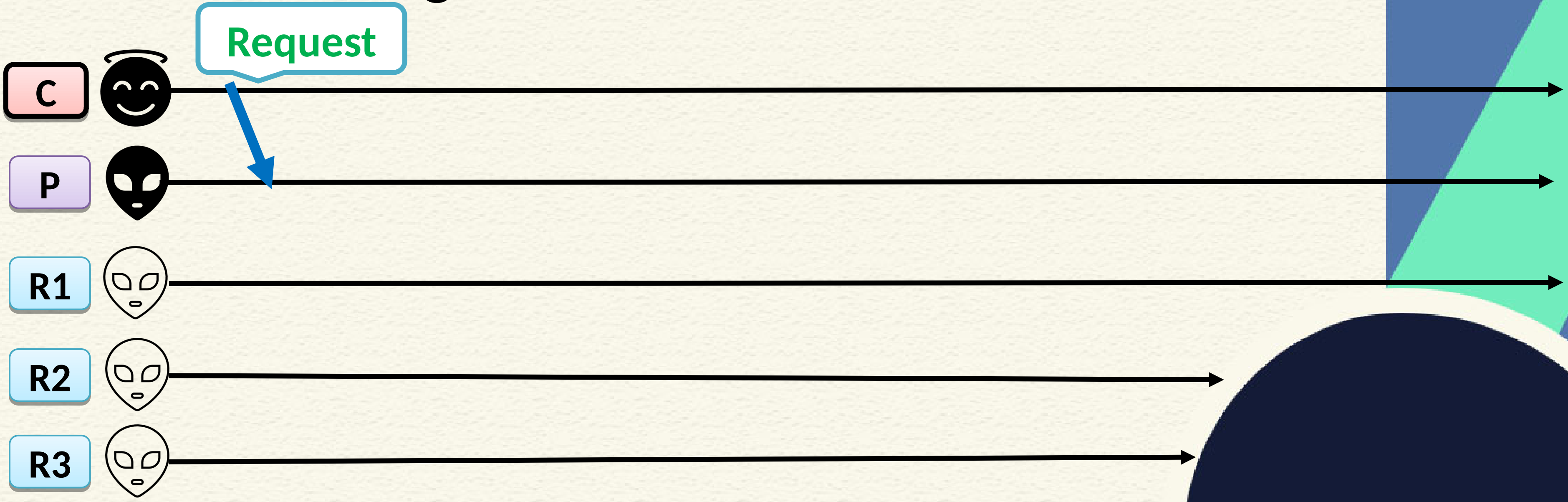
Client waits for 2F+1 replies, where F is the maximum number of faulty nodes

PBFT – The Algorithm



- The protocol starts by the client sending a Request message to the primary

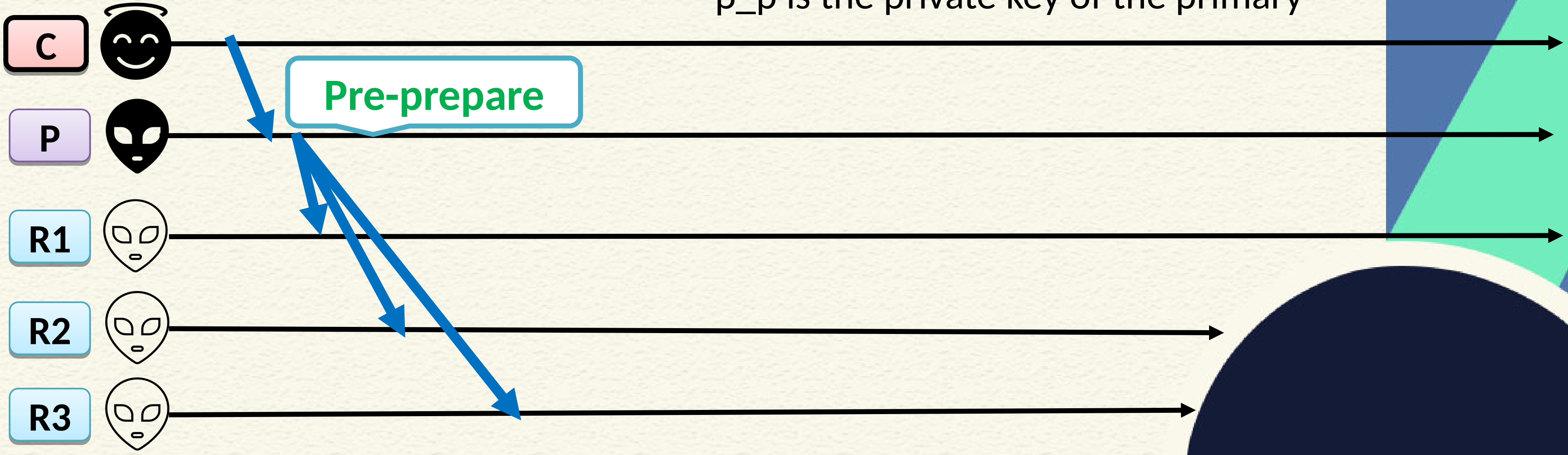
PBFT – The Algorithm



- The primary collects all the Request messages from different clients and order them based on certain pre-defined logic

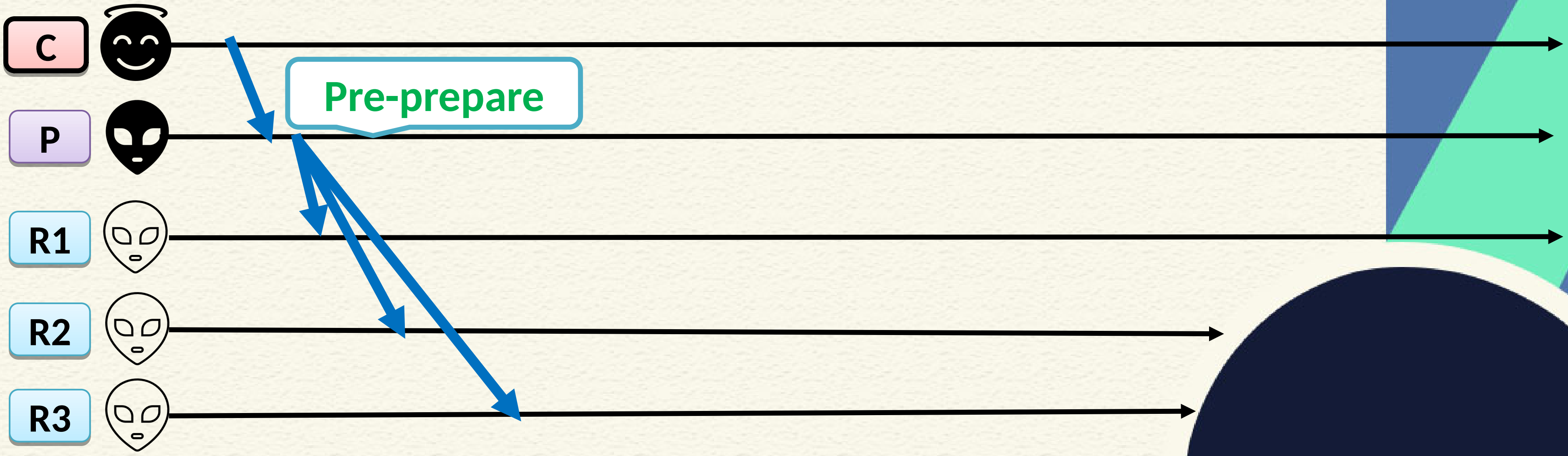
PBFT – The Algorithm

- *v* is the current view number, *d* is the message digest, m is the message
- β_p is the private key of the primary



- Primary assigns a sequence number *n* to the Request (or a set of Requests) and multicast a message **<<PRE-PREPARE, *v*, *n*, *d*>β_p, *m*>** to all the backups

PBFT – The Algorithm

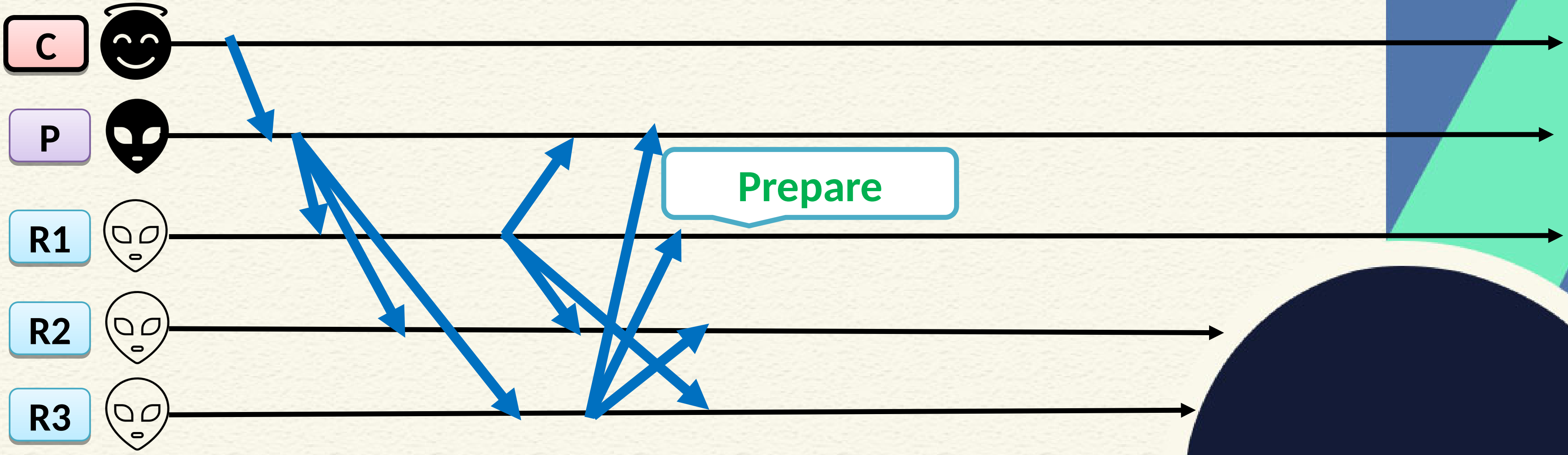


- Pre-prepare works as a proof that the Request was assigned a sequence number *n* for the view *v*

Accepting Pre-Prepare

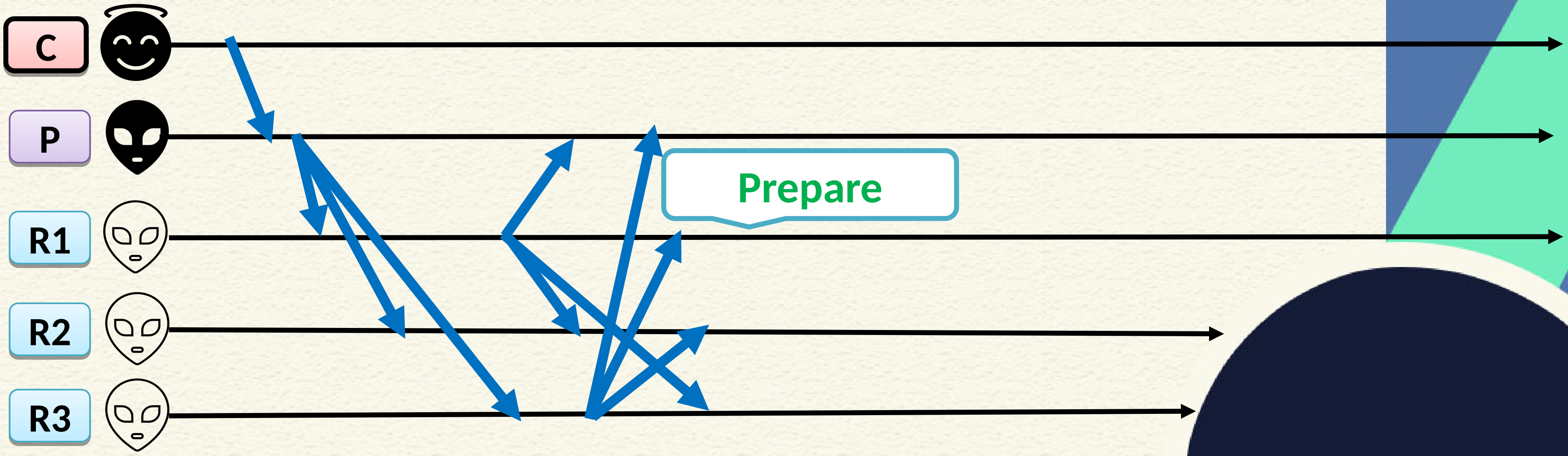
- A backup accepts the Pre-prepare message, if
  - The signature is correct and *d* is the digest of the message *m*
  - The backup is in view *v*
  - It has not received a different Pre-Prepare message with sequence *n* and view *v* with a different message digest
  - The sequence number is within a threshold (the message is not too old – prevents a reply attack)

PBFT – The Algorithm



- The correct backups send a Prepare message to all other backups including the primary – works as proof that the backups agree on the message with the sequence number *n* under view *v*

PBFT – The Algorithm



- Message format for backup *k* : <PREPARE, *v*, *n*, *d*, *k*>β_k

Accepting Prepare Message

- Primary and backups accepts the Prepare message, if
  - The signatures are correct
  - View number is equal to the current view
  - Sequence number is within a threshold (note that messages may be received **out of order** – so a backup may receive the Prepare message before the corresponding Pre-prepare message – so it needs to keep track of all the messages received)

Three Phase Commit

- Pre-prepare and Prepare ensure that non-faulty replicas guarantee on a **total order** for the requests within a view

Three Phase Commit

- Pre-prepare and Prepare ensure that non-faulty replicas guarantee on a **total order** for the requests within a view

- Assumptions for Commit:
  - Primary is non-faulty
  - You may have a maximum of ***f*** faults including Crash + Network + Byzantine

Three Phase Commit

- A message is committed if
  - ***2f*** Prepare from different backups matches with the corresponding Pre-prepare
  - You have total ***2f + 1*** votes (one from the primary that you already have!) from the non-faulty replicas

Three Phase Commit

- A message is committed if
  - ***2f*** Prepare from different backups matches with the corresponding Pre-prepare
  - You have total ***2f + 1*** votes (one from the primary that you already have!) from the non-faulty replicas

- Note that all ***2f + 1*** votes may not be same
  - You have votes from Byzantine faulty replicas as well

Quorum – Why 2f+1 Votes?

- **Quorum:** Minimum number of votes a distributed transaction needs to obtained to get committed
  - Proposed by David Gifford in 1979 (Gifford, David K. (1979). *Weighted voting for replicated data*. SOSP '79)
  - Widely used in Commit protocols and Replica management

Quorum – Why 2f+1 Votes?

- **Byzantine Dissemination Quorum:**
  - **Intersection:** Any two quorums have at least one correct replica in common
  - **Availability:** There is always a quorum available with no faulty replicas

Quorum – Why 2f+1 Votes?

- **Byzantine Dissemination Quorum:**
  - **Intersection:** Any two quorums have at least one correct replica in common
  - **Availability:** There is always a quorum available with no faulty replicas

- PBFT uses Byzantine Dissemination Quorum with $2f + 1$ replicas

Quorum in PBFT

- You have *f* number of faulty nodes – you need at least ***3f + 1*** replicas to reach consensus
  - But you do not know whether those are Crash faults, Network faults, or Byzantine Faults

Quorum in PBFT

- **Case 1: All *f* are Crash or Network faulty – You'll not receive messages from them!**
  - You'll receive ***2f + 1*** Prepare messages from non-faulty nodes
  - All these ***2f + 1*** are non-faulty votes – you can reach to an agreement

Quorum in PBFT

- **Case 2: All *f* are Byzantine faulty – they send messages!**
  - You may receive at most ***3f + 1*** Prepare messages (votes) -- ***f*** are from Byzantine nodes
  - Sufficient to wait till ***2f + 1*** Prepare messages – even if ***f*** are faulty, you still have ***f+1*** non-faulty votes
  - You cannot wait for ***f+1***, the first ***f*** might be all faulty

Quorum in PBFT

•

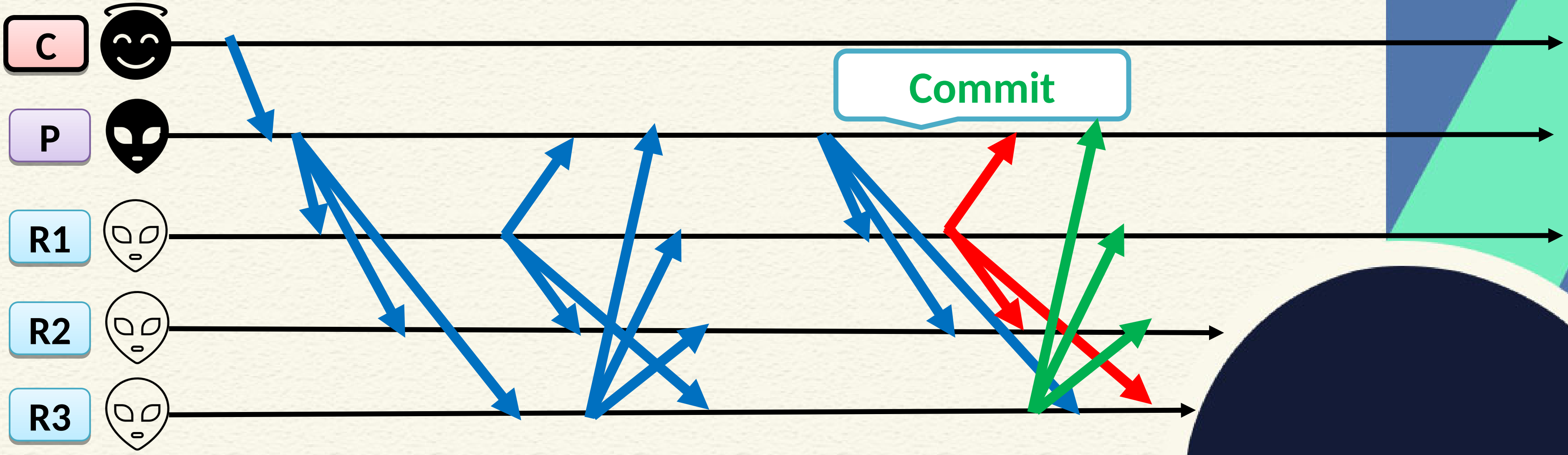
s) -- *f* are

are

Remember, you are on an **asynchronous channel** – messages get delayed and can be received out of order

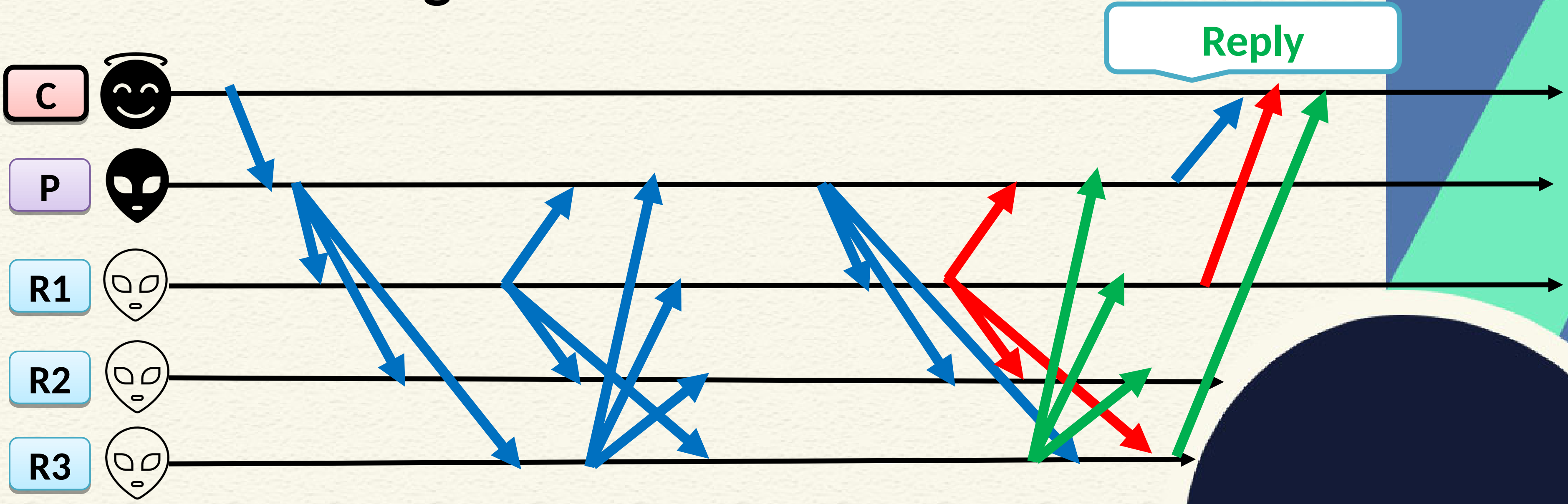
Wait untill you receive *2f + 1* Prepare messages – once you received *2f + 1* votes, you can safely take a decision based on majority voting

PBFT – The Algorithm



- Message format for replica *k* : <COMMIT, *v*, *n*, *d*, *k*>β_k

PBFT – The Algorithm



- The protocol is committed for a replica when
  - It has sent the Commit message
  - It has received *2f* Commit messages from other replicas

Conclusion

- PBFT works with 3f+1 replicas over 2f+1 quorum
- Next, we'll see the safety and liveness properties of PBFT


Thank you