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

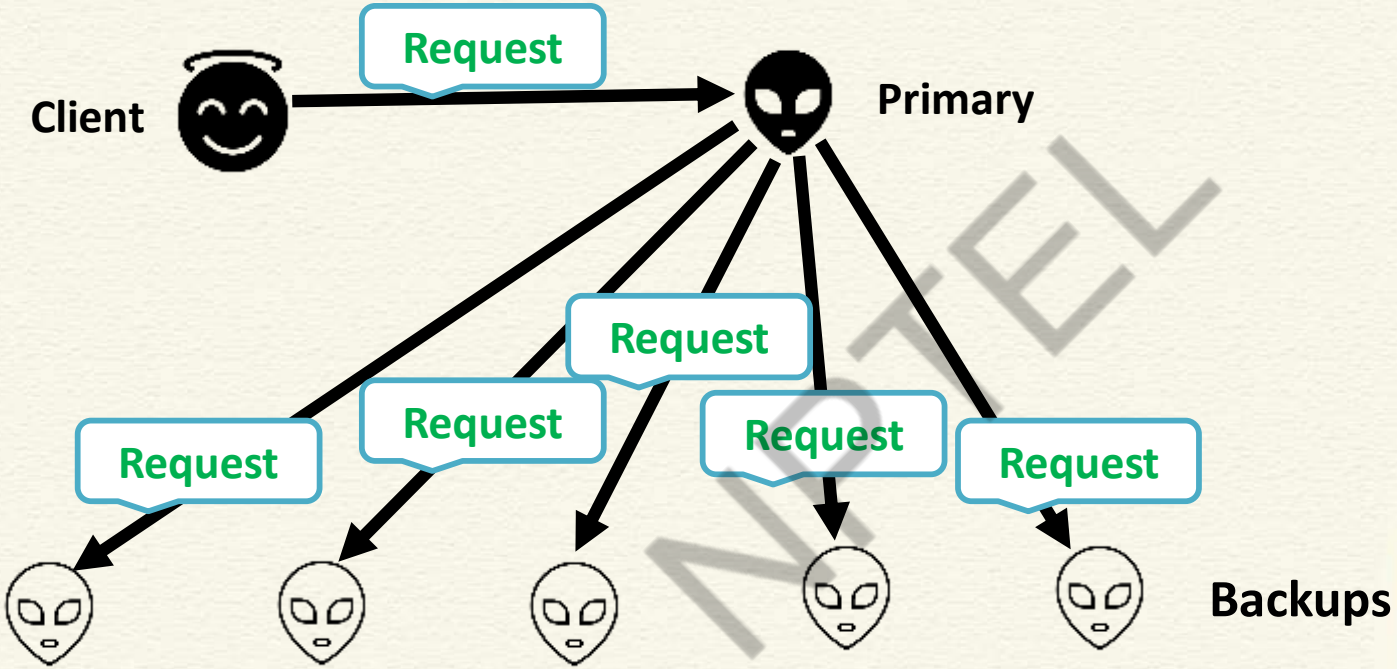
- The replicas move through a succession of configurations, known as ***views***
  - One replica in a view is considered as the **primary** (works like a leader), and others are considered **backups**
  - The primary proposes a value (similar to the Proposers in Paxos), and the backups accept the value (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

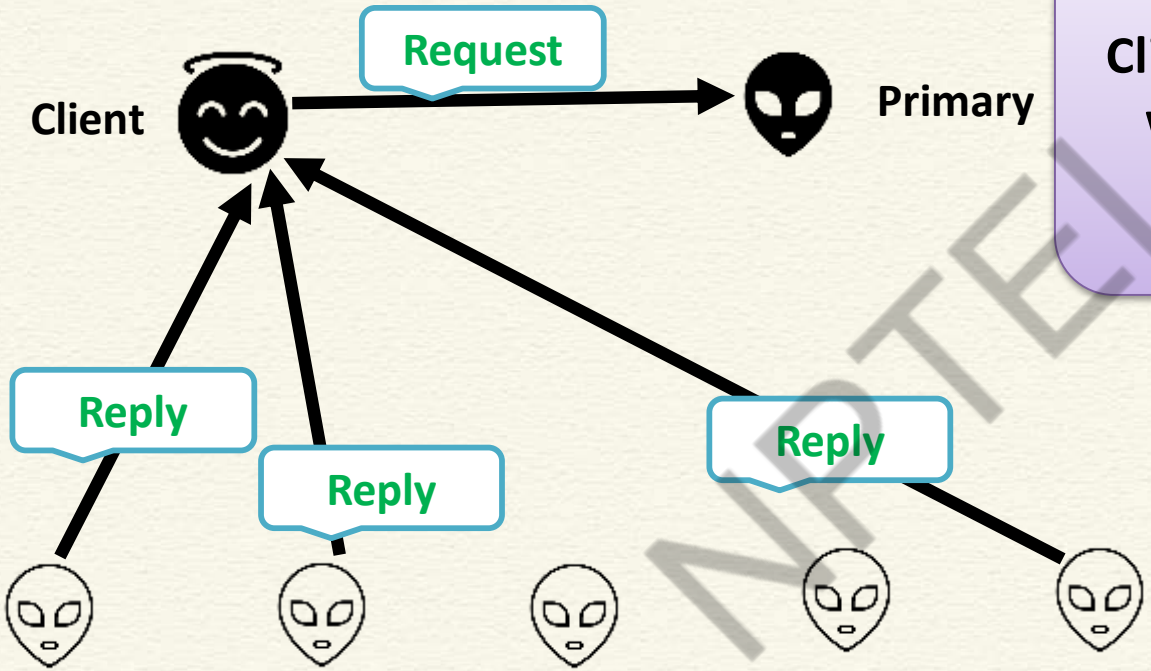
- The replicas move through a succession of configurations, known as ***views***
  - One replica in a view is considered as the **primary** (works like a leader), and others are considered **backups**
  - The primary proposes a value (similar to the Proposers in Paxos), and the backups accept the value (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

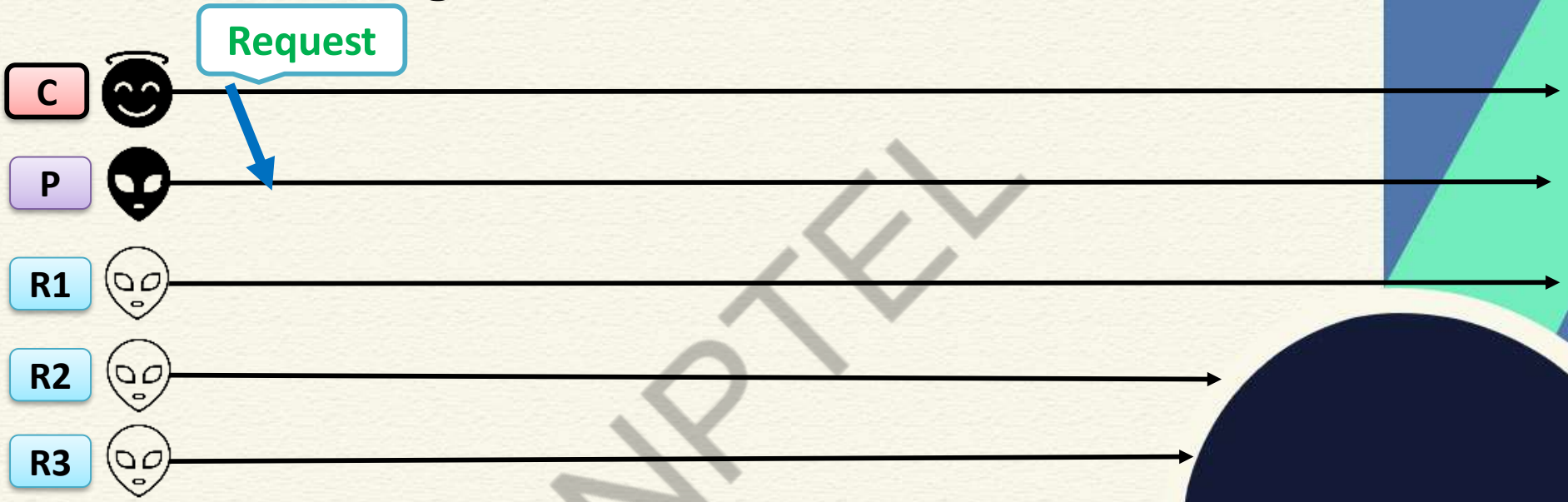


PBFT – Broad Idea



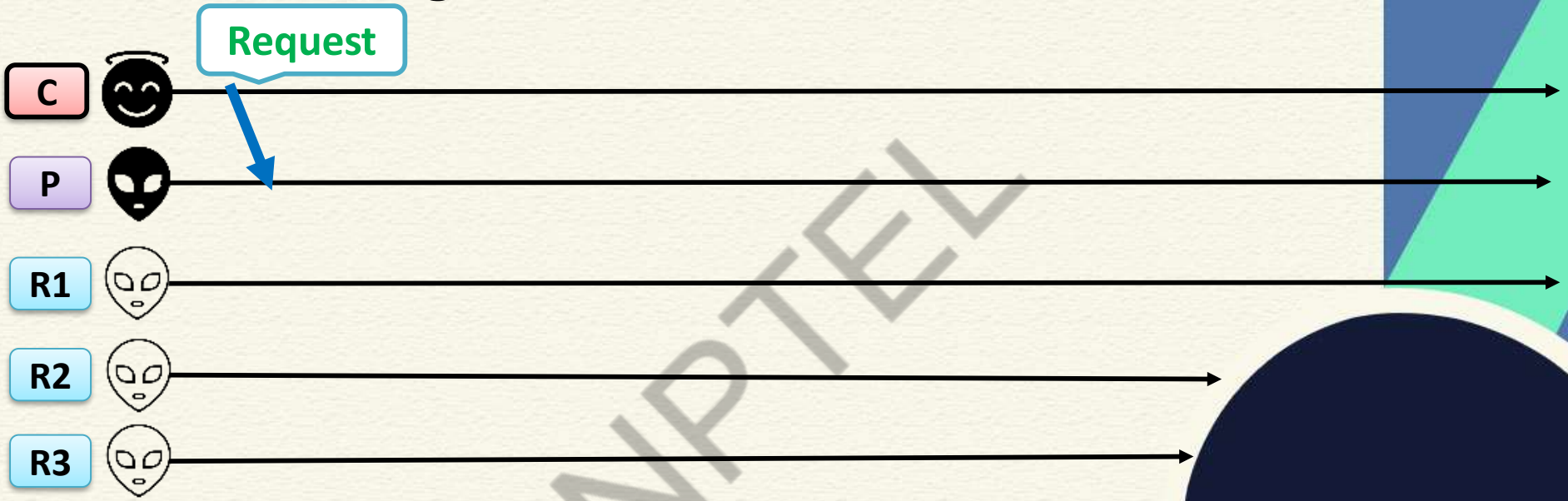
Client waits for 2F+1 replies, where F is the maximum number of faulty nodes

PBFT – The Algorithm



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

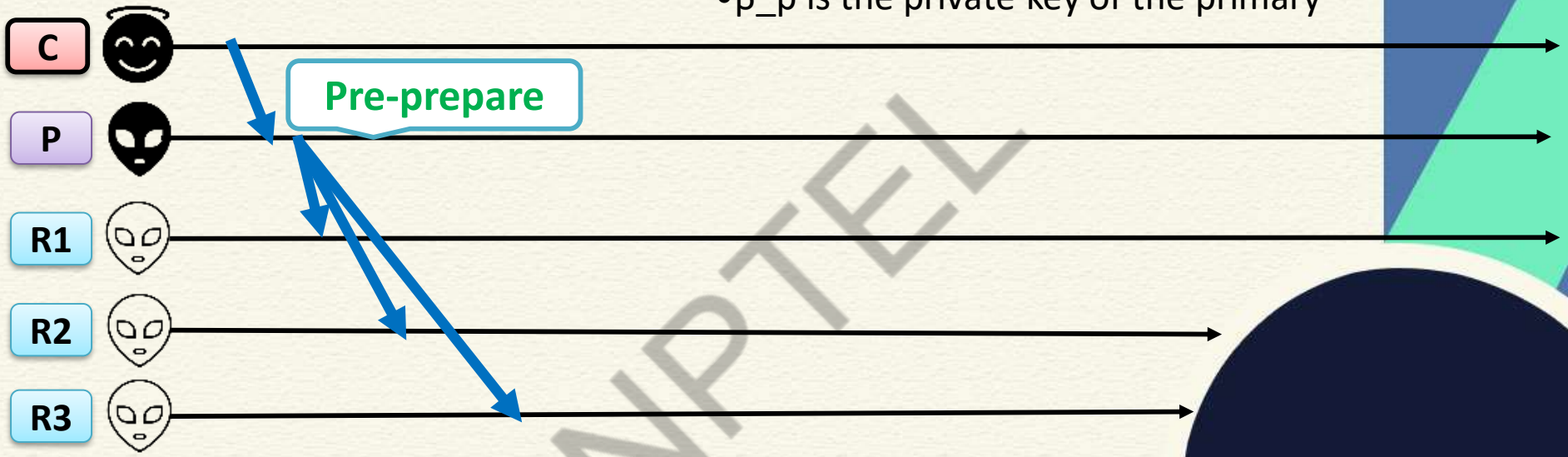
PBFT – The Algorithm



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

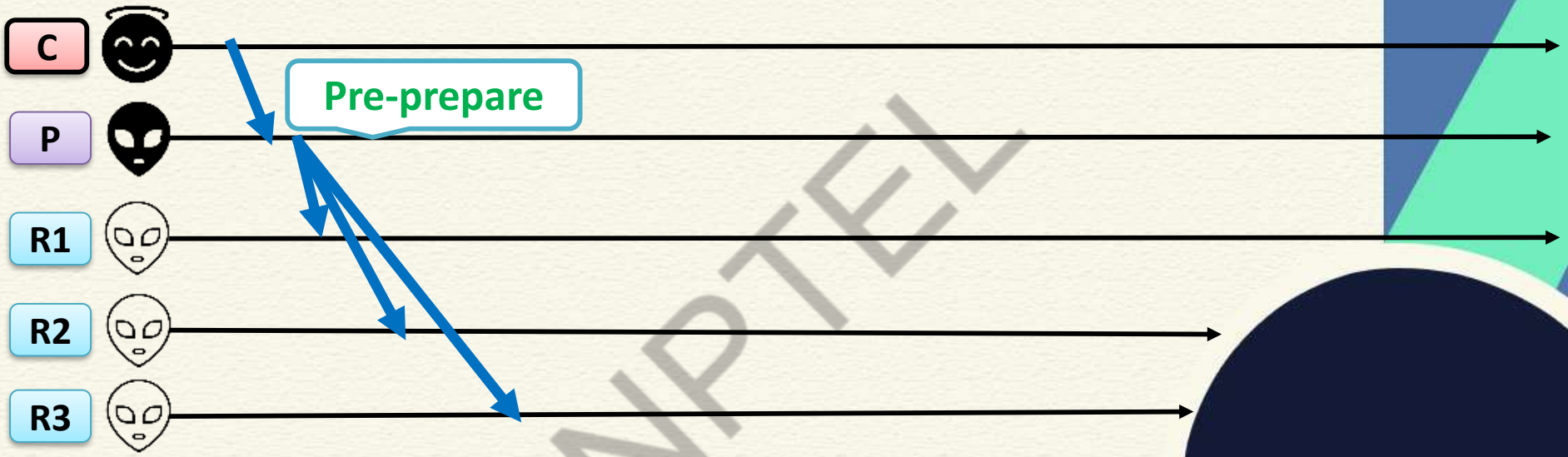
PBFT – The Algorithm

- *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

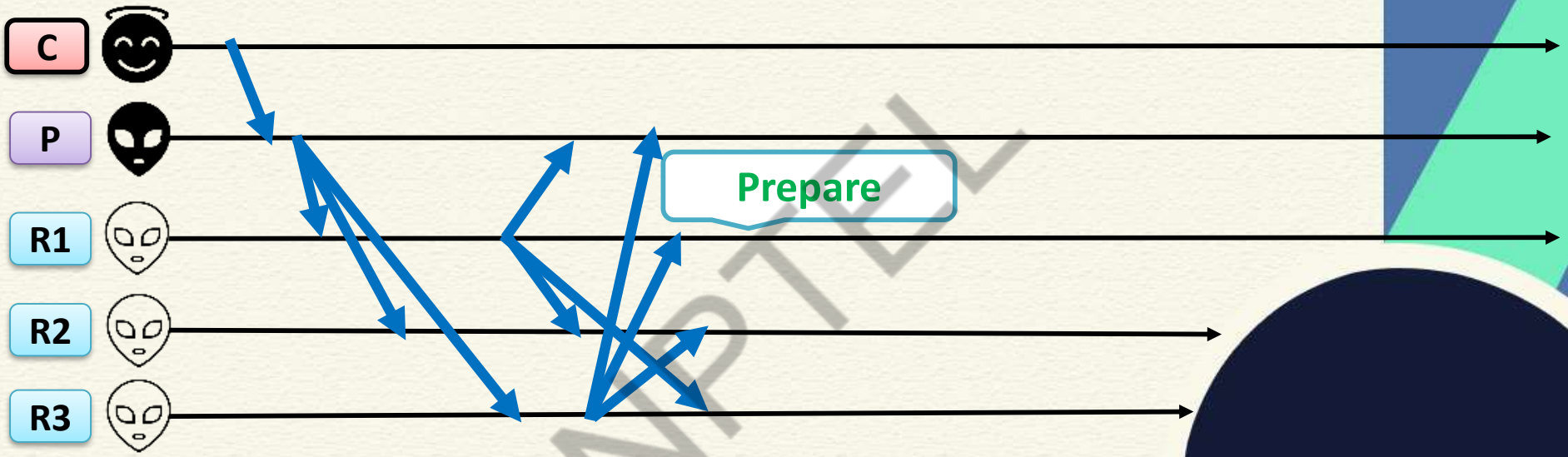


- Pre-prepare works as a proof that the Request was assigned a sequence number *n* for the view *v*

Accepting Pre-Prepare

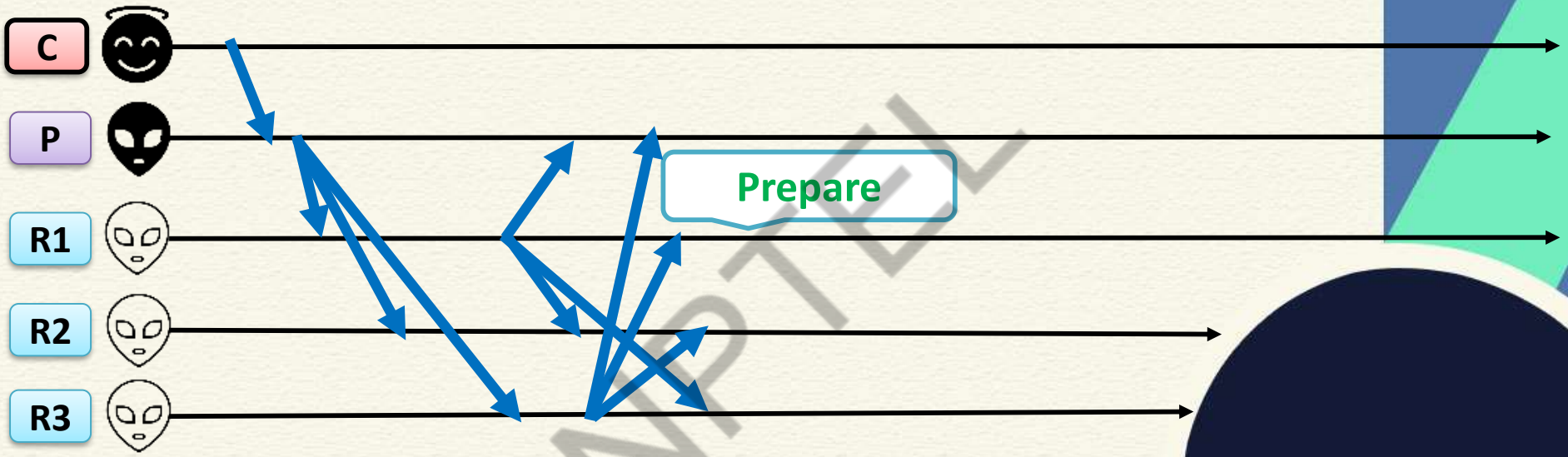
- A backup accepts the Pre-prepare message, if
  - The signature is correct and *d* is the digest of the message *m*
  - The backup is in view *v*
  - It has not received a different Pre-Prepare message with sequence *n* and view *v* with a different message digest
  - The sequence number is within a 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

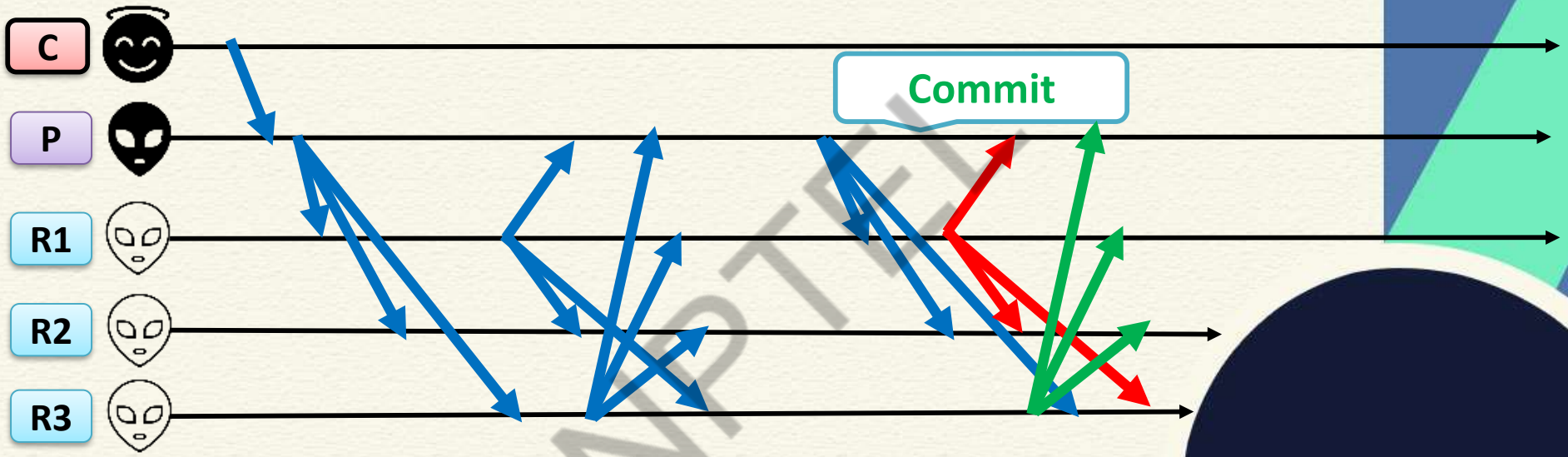
- Remember, you are on an **asynchronous channel** – messages get delayed and can be received out of order

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

s) -- ***f*** are

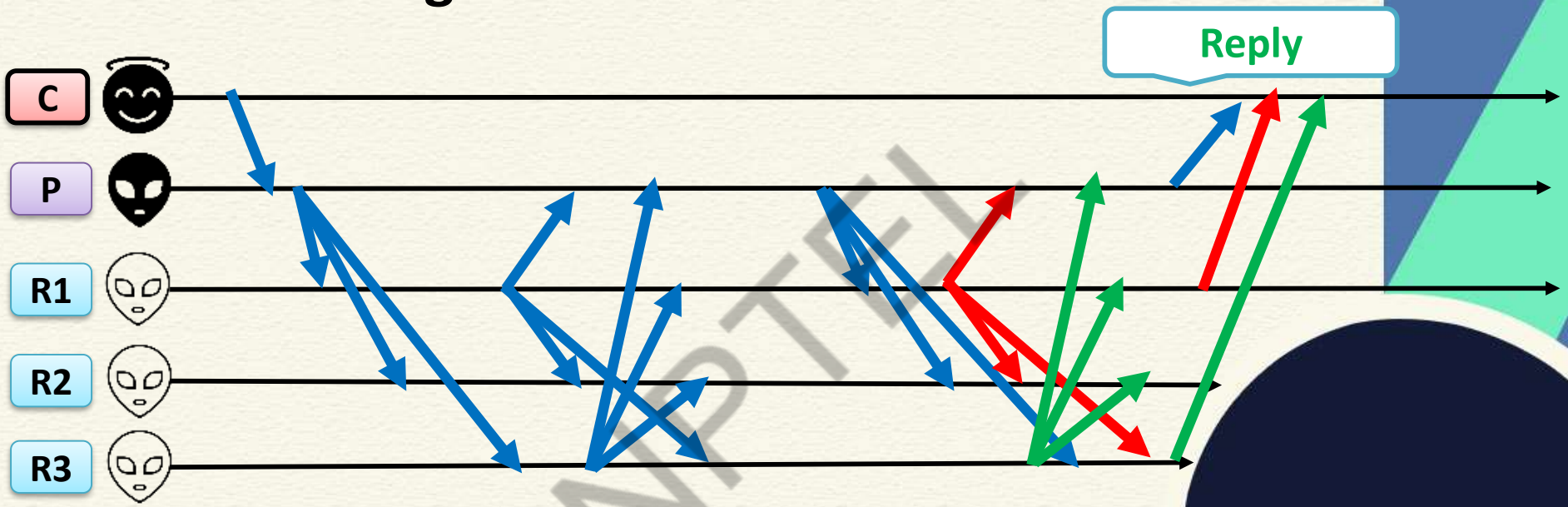
are

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

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 32: Safety and Liveness of PBFT

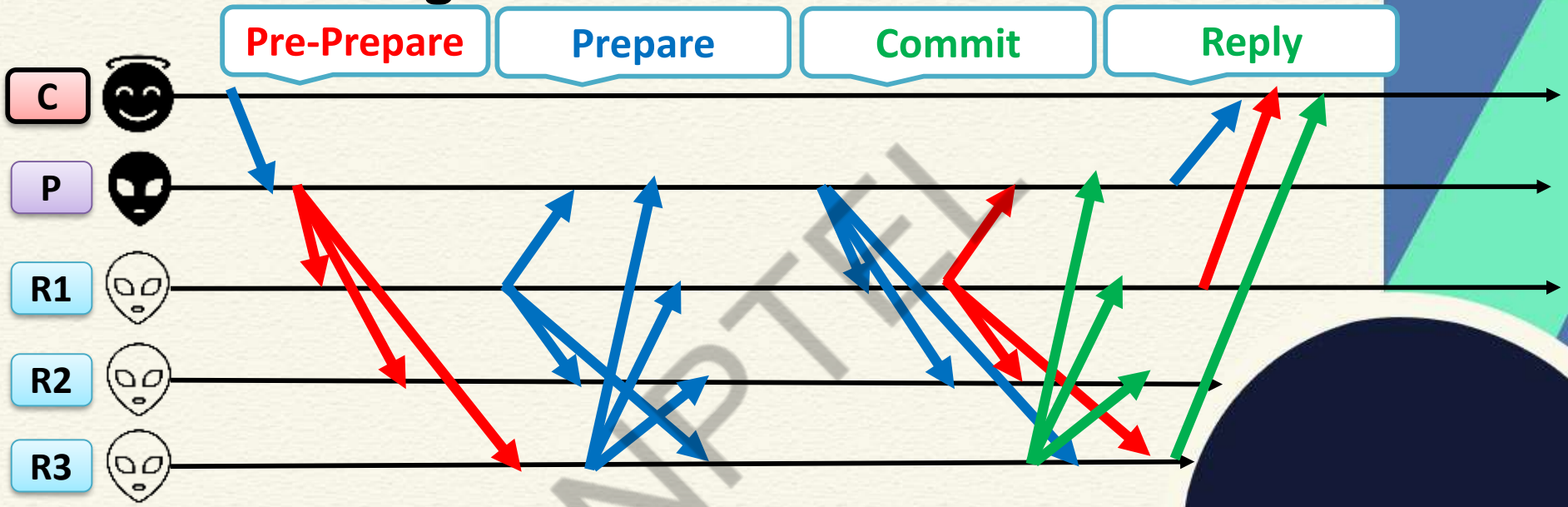
CONCEPTS COVERED

- **Safety and Liveness of PBFT**
- **PBFT View Change**

KEYWORDS

- **Weak Synchrony assumptions**
- **The view change protocol**

PBFT – The Algorithm



Safety in PBFT

- Unlike multiple Paxos proposers, **PBFT works with a single Primary**
  - Ping-pong does not arise from the proposals from multiple replicas
  - However, a replica needs to wait for ***2f + 1*** votes (Prepare and Commit messages)

Safety in PBFT

- PBFT is safe with **2f+1** quorum
  - The leader can always have the majority votes to support its proposal

- The leader can reach to the consensus even when it does not receive messages from some of the replicas due to asynchronous nature of the channel

Liveness in PBFT

- However, a primary may fail – the liveness gets hampered as the protocol cannot progress any further
  - Primary failure cannot be handled in a pure asynchronous system – you do not know whether it is a message delay from the primary, or a primary failure

Weak Synchrony Assumption

- **Weak Synchrony:**
  - (1) Both sender and the receiver is correct,
  - (2) Sender keeps retransmitting the messages until it is received,
  - (3) There is an **asymptotic upper bound** on the message transmission delay

The View Change Protocol

- What if the **primary** is **faulty** ?
  - Non-faulty replicas detect the fault
  - Replicas together start view change operation

- View-change protocol provides **eventual liveness** -- Allows the system to make progress when primary fails

The View Change Protocol

- If the primary fails, backups will not receive any message or will receive faulty messages from the primary

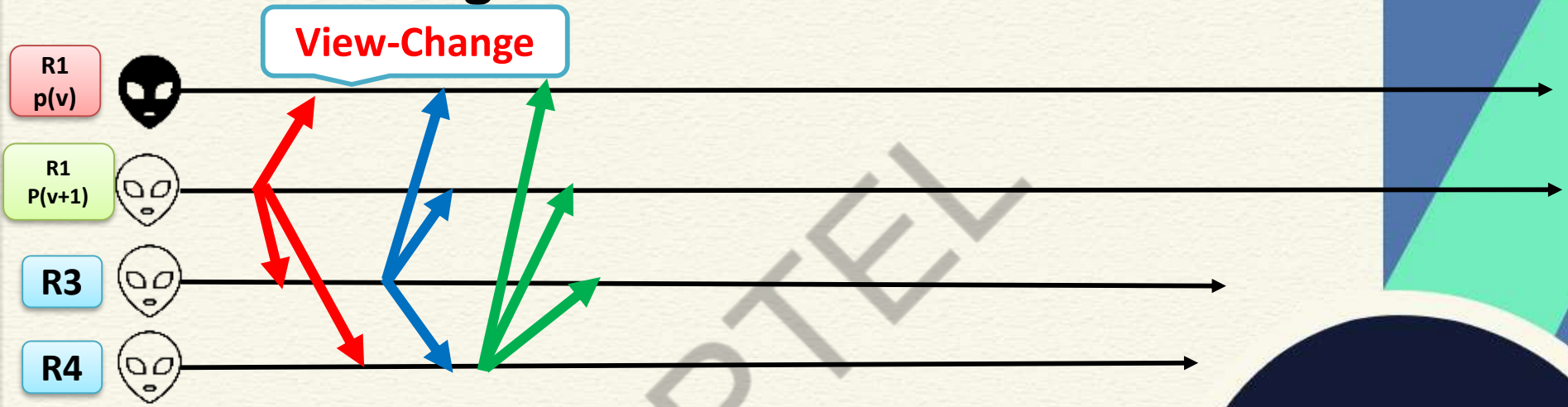
- View changes are triggered by timeouts (weak synchrony assumption)
  - Prevent backups from waiting indefinitely for requests to execute

The View Change Protocol

- Backup starts a timer when it receives a request, and the timer is not already running
  - The timer is stopped when the request is executed
  - Restarts when some new request comes

- If the timer expires at view *v*, backup starts a **View Change** to move to the view *v + 1*

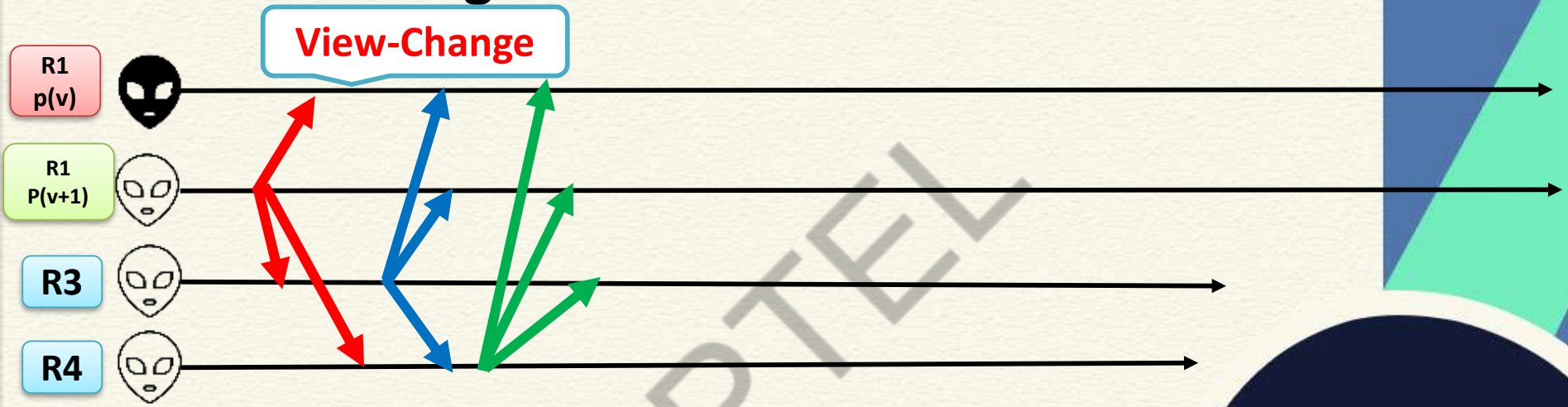
The View Change Protocol



Multicast the View Change message <VIEW-CHANGE, *v+1*, *n*, *C*, *P*, *k*>β_k

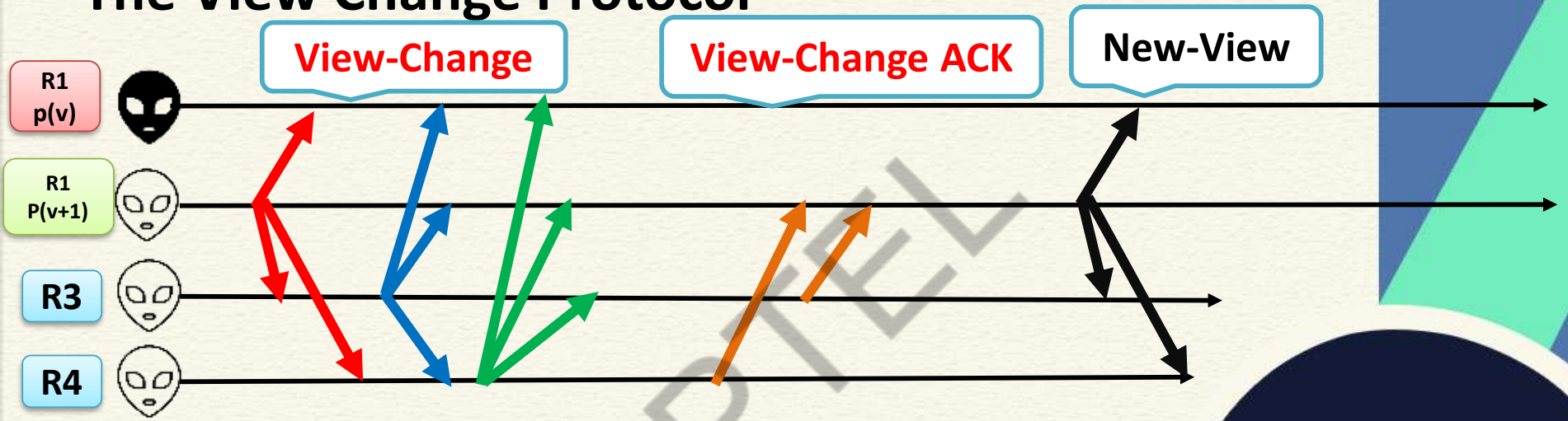
- *n* is the sequence number of last stable checkpoint *s* known to *k*
- *C* is a set of *2f + 1* valid checkpoint messages corresponding to *s*
- *P* is a set containing a set *Pm* for each request *m* that prepared at *k* with a sequence number higher than *n*

The View Change Protocol



- The new view is initiated after receiving *2f + 1* View Change messages
- Next primary selection
  - Round Robin (Hyperledger Sawtooth)
  - Leader election (Hyperledger Fabric)

The View Change Protocol



- Replicas send a View Change ACK – quorum is formed on these messages
- New View message to initiate a new view

Conclusion

- PBFT is safe under 2f+1 quorum over an asynchronous environment
- Liveness if affected when the primary is faulty
- View change to elect a new primary when the primary is detected as faulty
  - Weak synchrony assumption

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 33: Enterprise Blockchains

CONCEPTS COVERED

- **Enterprise blockchains**

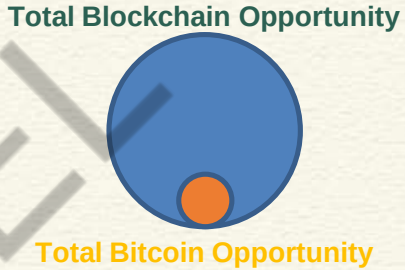
KEYWORDS

- Enterprise blockchain applications
- The Hyperledger greenhouse
- Hyperldger Fabric

Blockchain – The Application Space

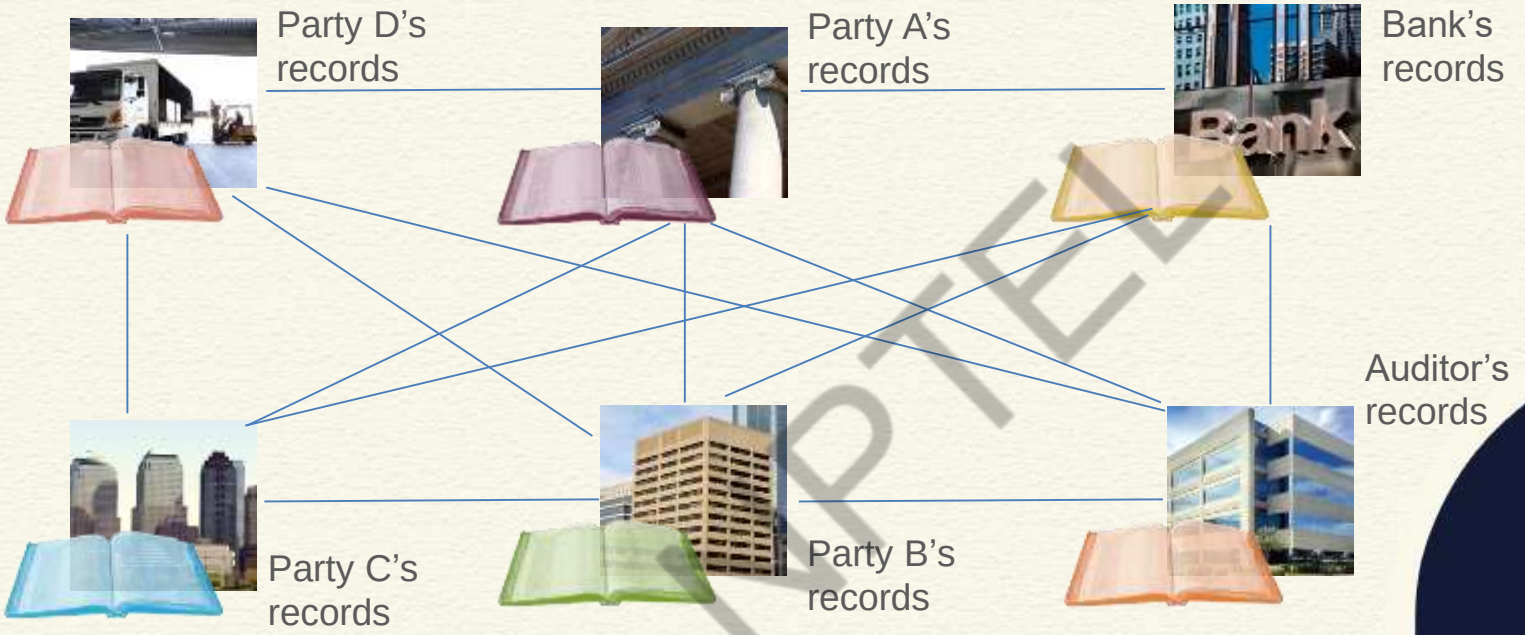
Blockchain is a design pattern made famous by its use in Bitcoin. But its uses go *far beyond*.

Blockchain can reimagine the world's most fundamental business interactions and open the door to invent new styles of digital interactions.



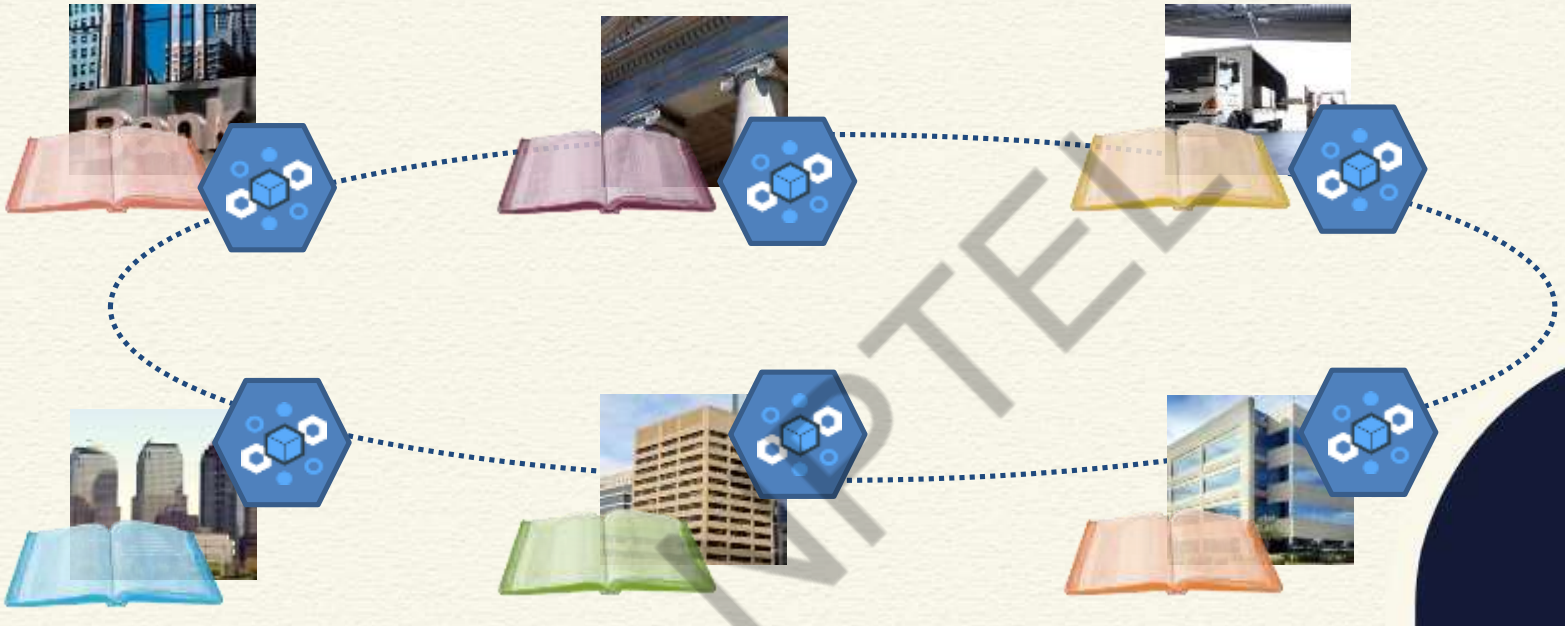
Enterprises are adopting Blockchain to a very broad range of business applications

Asset Transfer in a Business Network



… Inefficient, expensive, vulnerable

Asset Transfer in a Business Network



… Consensus, provenance, immutability, finality

Benefits of Blockchain for Business

Append-only distributed system of record shared across business network

Shared Ledger

Security

Ensuring appropriate visibility; transactions are secure, authenticated & verifiable

Business terms embedded in transaction database & executed with transactions

Smart Contracts

Consensus

All parties agree to network verified transaction

ReducesTime

Transaction time from days to near instantaneous

Removes Cost

Overheads and cost of intermediaries

ReducesRisk

Tampering, fraud & cyber crime

Enables NewBusiness Models

IoT Integration into supply chain

Degree of Centralization

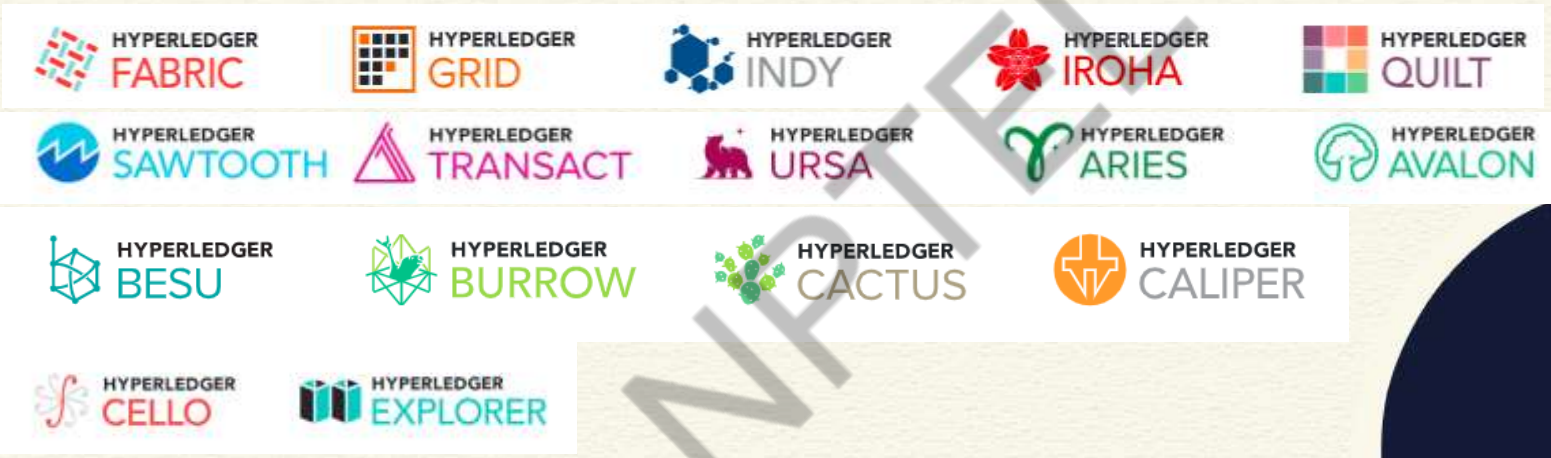
Figure source: "Distributed Ledger Technology: Beyond Blockchain", A report by UK Govt Chief Scientific Adviser

Permissionless vs Permissioned Blockchains

	Permissionless	**Permissioned**
Access	Open read/write access to database	Permissioned read/write access to database
Scale	Scale to a large number of nodes, but not in transaction throughput	Scale in terms of transaction throughput, but not to a large number of nodes
Consensus	Proof of work/ proof of stake	Closed membership consensus algorithms
Identity	Anonymous/pseudonymous	Identities of nodes are known, but transaction identities can be private/anonymous/pseudonymous
Asset	Native assets	Any asset/data/state

The Linux Foundation Hyperledger Project

A collaborative effort created to advance blockchain technology by identifying and addressing important features for a cross-industry open standard for distributed ledgers that can transform the way business transactions are conducted globally.



https://www.hyperledger.org/

Hyperledger Members

https://www.hyperledger.org/about/members

Premier

General (Partial list)

Associates (Partial list)

Hyperledger Fabric – Distributed Ledger Platform

https://www.hyperledger.org/use/fabric

- An implementation of blockchain technology that is a foundation for developing blockchain applications
- Emphasis on ledger, smart contracts, consensus, confidentiality, resiliency and scalability.
- V1.0 released July 2017
  - 159 developers from 27 organizations

Conclusion

- Enterprise blockchains have a wide spectrum of applications and use cases that can be developed over permissioned models

- We'll next explore Hyperledger Fabric to develop our first DLT application

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Bishakh Chandra Ghosh

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 34: Hyperledger Fabric 1

CONCEPTS COVERED

- Hyperledger Foundation
- Hyperledger Fabric Introduction
- Fabric Installation

KEYWORDS

- Hyperledger
- Fabric
- Permissioned Network

Hyperledger Foundation

https://www.hyperledger.org/

- Open source community - focused on **enterprise-grade blockchain deployments**.
- Home for various distributed ledger frameworks including: Hyperledger Fabric, Sawtooth, Indy, etc.

Hyperledger Foundation

HYPERLEDGER FOUNDATION

https://www.hyperledger.org/

- Open source community - focused on **enterprise-grade blockchain deployments**.
- Home for various distributed ledger frameworks including: Hyperledger Fabric, Sawtooth, Indy, etc.

  - Different companies / organizations want to collaborate
  - Closed group: members know each other
  - Do not fully trust each other
  - Distributed shared ledger – based on permissioned consensus

Hyperledger Foundation Projects

Tooling to serve as operational dashboard for Blockchains

Tooling to invoke, deploy or query blocks

Permissioned Enterprise Blockchain

Permissioned, EVM Based, BFT Consensus

Identity Management

Hyperledger Fabric

- **Open source,** enterprise-grade
- **Permissioned** DLT platform

- **Modular blockchain framework**
  - Designed for developing blockchain-based products, solutions, and applications using plug-and-play components that are aimed for use within private enterprises.

- Pluggable Components: Including **consensus and membership services**.

- **Smart contracts in general purpose languages** such as **Java, Go and Node.js**.

https://hyperledger-fabric.readthedocs.io/

Install Hyperledger Fabric - Prerequisites

Install Prerequisites

- **Git**
  - https://git-scm.com/downloads
- **cURL**
  - https://curl.se/download.html
- **Docker** (Docker version 17.06.2-ce or greater is required)
  - https://docs.docker.com/engine/install/
- **Go**
  - https://golang.org/doc/install
- **Docker Compose** (Docker Compose version 1.14.0 or greater installed)
  - https://docs.docker.com/compose/install/

https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html

Install Hyperledger Fabric - Prerequisites

Install Prerequisites

- **Git**
  - `sudo apt install git`
- **cURL**
  - `sudo apt install curl`
- **Go**
  - wget https://golang.org/dl/go1.17.3.linux-amd64.tar.gz
  - sudo rm -rf /usr/local/go
  - sudo tar -C /usr/local -xzvf go1.17.3.linux-amd64.tar.gz

https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html

Install Hyperledger Fabric - Prerequisites

Install Prerequisites

Docker

```
sudo apt install ca-certificates gnupg lsb-release

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io

sudo groupadd docker
sudo usermod -aG docker $USER
```

https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html

Install Hyperledger Fabric - Prerequisites

Install Prerequisites

Docker Compose

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose
```


https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html

Install Hyperledger Fabric 2.2

Install Samples, Binaries, and Docker images

Determine a location on your machine where you want to place the *fabric-samples* repository and enter that directory in a terminal window.

```
mkdir fabric
cd fabric
curl -sSL https://bit.ly/2ysbOFE | bash -s -- <fabric_version> <fabric-ca_version>
```

Example:

```
curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.2.4 1.5.2
```

```
export PATH=<path to download location>/bin:$PATH
```

https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html

Conclusion

- Hyperledger Foundation
- Enterprise blockchain – Fabric
- Installation of Hyperledger Fabric

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Bishakh Chandra Ghosh

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 35: Hyperledger Fabric 2

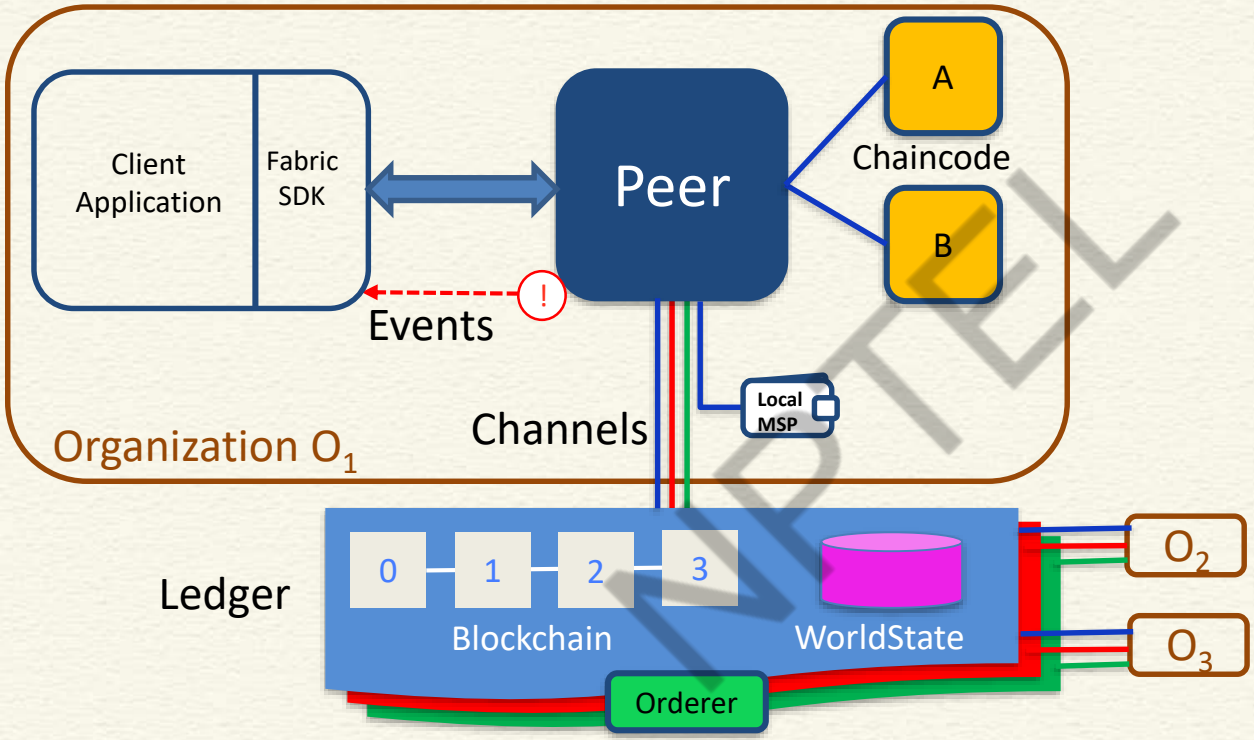
CONCEPTS COVERED

- Fabric Architecture
- Fabric Test Network
- Sample Chaincode
- Invoke and Query Chaincode

KEYWORDS

- **Fabric Test Network**
- **Chaincode Transactions**

Fabric Architecture



Fabric Test Network

- Real network consists of multiple organizations. Each maintain their own set of:
  - Peers
  - Client Applications
  - Optionally Orderers
  - MSP

- **Test Network**:
  - All organizations in a single system
  - Development and testing purposes
  - 2 Orgs, each having 1 peer and optionally one CA
  - 1 orderer
  - All components are containerized

Start Test Network

Navigate to the directory where you have installed fabric samples.

```
cd fabric-samples
cd test-network
./network.sh up
```

```
~/fabric/fabric-samples/test-network  main  ./network.sh up
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb' with crypto from 'cryptogen'
LOCAL_VERSION=2.2.4
DOCKER_IMAGE_VERSION=2.2.4
/home/bishakh/fabric/fabric-samples/test-network/../bin/cryptogen
Generating certificates using cryptogen tool
Creating Org1 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organizations
org1.example.com
+ res=0
Creating Org2 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organizations
org2.example.com
+ res=0
Creating Orderer Org Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organizations
+ res=0
Generating CCP files for Org1 and Org2
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "fabric_test" with the default driver
Creating volume "docker_orderer.example.com" with default driver
Creating volume "docker_peer0.org1.example.com" with default driver
Creating volume "docker_peer0.org2.example.com" with default driver
```

Monitor Containers

`docker ps`

```
~/fabric/fabric-samples/test-network  main  docker ps
CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS              PORTS
                                                                                                          NAMES
9f60342c20ac   hyperledger/fabric-tools:latest     "/bin/bash"         2 minutes ago   Up About a minute
                                                                                                          cli
9828aff6e8f1   hyperledger/fabric-peer:latest      "peer node start"   2 minutes ago   Up 2 minutes        0.0.0.0:7051->7051/tcp, :::7051-
>7051/tcp, 0.0.0.0:17051->17051/tcp, :::17051->17051/tcp                                                  peer0.org1.example.com
a7cf98ab34c0   hyperledger/fabric-orderer:latest   "orderer"           2 minutes ago   Up 2 minutes        0.0.0.0:7050->7050/tcp, :::7050-
>7050/tcp, 0.0.0.0:7053->7053/tcp, :::7053->7053/tcp, 0.0.0.0:17050->17050/tcp, :::17050->17050/tcp   orderer.example.com
85eb38ae9ea4   hyperledger/fabric-peer:latest      "peer node start"   2 minutes ago   Up 2 minutes        0.0.0.0:9051->9051/tcp, :::9051-
>9051/tcp, 7051/tcp, 0.0.0.0:19051->19051/tcp, :::19051->19051/tcp                                        peer0.org2.example.com
```

- 2 fabric-peer containers, - 1 per organization.
- 1 fabric-orderer container
- 1 fabric-tools container

Create Channel

```
./network.sh createChannel
```

- Creates a channel with name "**mychannel**"

```
./network.sh createChannel -c <channel name>
```

```
~/fabric/fabric-samples/test-network  5b8c439  ./network.sh createChannel

Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveld
b'
Generating channel create transaction 'mychannel.tx'
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/mychannel.tx -channelID mychannel
2021-11-16 03:42:06.986 IST [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-11-16 03:42:07.008 IST [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/bishakh/fa
bric/fabric-samples/test-network/configtx/configtx.yaml
2021-11-16 03:42:07.008 IST [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2021-11-16 03:42:07.010 IST [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
+ res=0
Creating channel mychannel
Using organization 1
+ peer channel create -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.example.com -f ./channel-artifac
ts/mychannel.tx --outputBlock ./channel-artifacts/mychannel.block --tls --cafile /home/bishakh/fabric/fabric-samples/test-
network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.
pem
+ res=0
2021-11-16 03:42:10.087 IST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
```

Install Chaincode

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go

```
~/fabric/fabric-samples/test-network  5b8c439  ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chai
ncode-go -ccl go
deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: basic
- CC_SRC_PATH: ../asset-transfer-basic/chaincode-go
- CC_SRC_LANGUAGE: go
- CC_VERSION: 1.0
- CC_SEQUENCE: 1
- CC_END_POLICY: NA
- CC_COLL_CONFIG: NA
- CC_INIT_FCN: NA
- DELAY: 3
- MAX_RETRY: 5
- VERBOSE: false
Vendoring Go dependencies at ../asset-transfer-basic/chaincode-go
~/fabric/fabric-samples/asset-transfer-basic/chaincode-go ~/fabric/fabric-samples/test-network
~/fabric/fabric-samples/test-network
Finished vendoring Go dependencies
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go --lang golang --label basic_1.
0
+ res=0
Chaincode is packaged
```

Invoke Chaincode – Configure Peer

```
export FABRIC_CFG_PATH=$PWD/../config/


# Set environment variables for Org1


export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"


export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.exa
mple.com/peers/peer0.org1.example.com/tls/ca.crt

export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.exampl
e.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=localhost:7051
```

Invoke Chaincode

```
peer chaincode invoke -o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp
/tlscacerts/tlsca.example.com-cert.pem \
-C mychannel \
-n basic \
--peerAddresses localhost:7051 \
--tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/t
ls/ca.crt \
--peerAddresses localhost:9051 \
--tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/t
ls/ca.crt -c '{"function":"InitLedger","Args":[]}'
```

Invoke Chaincode

```
~/fabric/fabric-samples/test-network  ➦ 006942b  peer chaincode invoke -o localhost:7050 \
--ordererTLSHostnameOverride orderer.example.com \
--tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \
-C mychannel \
-n basic \
--peerAddresses localhost:7051 \
--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \
--peerAddresses localhost:9051 \
--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"function":"InitLedger","Args":[]}'

2021-11-16 04:39:55.375 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

```
~/fabric/fabric-samples/test-network  ➦ 006942b  docker ps
CONTAINER ID   IMAGE
                                    COMMAND                  CREATED              STATUS                PORTS                                        NAMES
1e7c0c143823   dev-peer0.org1.example.com-basic_1.0-4ec191e793b27e953ff2ede5a8bcc63152cecb1e4c3f301a26e22692c61967ad-42f57faac8360472e47cbbbf3948e81bba8343970
2d085878d148089a1b213ca             "chaincode -peer.add…"   About a minute ago   Up About a minute                                                  dev-pe
er0.org1.example.com-basic_1.0-4ec191e793b27e953ff2ede5a8bcc63152cecb1e4c3f301a26e22692c61967ad
56af428daf24   dev-peer0.org2.example.com-basic_1.0-4ec191e793b27e953ff2ede5a8bcc63152cecb1e4c3f301a26e22692c61967ad-6c0d5b0755cb92ed5555bd2e8a8765a6f425d1ed5
ed9a90e625e01939e2113be             "chaincode -peer.add…"   About a minute ago   Up About a minute                                                  dev-pe
er0.org2.example.com-basic_1.0-4ec191e793b27e953ff2ede5a8bcc63152cecb1e4c3f301a26e22692c61967ad
3e162f246443   dev-peer0.org1.example.com-mychaincode_1.0-29c02707eeac8b0ea3a398bda48ecb8365216e2d2ce3d327f9a645465219cf2b-dd78b6c684a11434dea96b4dfebadaba8ff
1bd392da3dcf9629357c2501e1183       "chaincode -peer.add…"   8 minutes ago        Up 8 minutes                                                       dev-pe
er0.org1.example.com-mychaincode_1.0-29c02707eeac8b0ea3a398bda48ecb8365216e2d2ce3d327f9a645465219cf2b
047728ffad54   dev-peer0.org2.example.com-mychaincode_1.0-29c02707eeac8b0ea3a398bda48ecb8365216e2d2ce3d327f9a645465219cf2b-ffd1881bf7c1c540e095927a973925e0b2e
4fc289fa83d92e143e6617d6863cf       "chaincode -peer.add…"   8 minutes ago        Up 8 minutes                                                       dev-pe
er0.org2.example.com-mychaincode_1.0-29c02707eeac8b0ea3a398bda48ecb8365216e2d2ce3d327f9a645465219cf2b
166d8b0b77f6   hyperledger/fabric-tools:latest
                                    "/bin/bash"              9 minutes ago        Up 9 minutes                                                       cli
c63a321e7de6   hyperledger/fabric-peer:latest
                                    "peer node start"        9 minutes ago        Up 9 minutes          7051/tcp, 0.0.0.0:9051->9051/tcp, :::9051->9051/tcp   peer0.
org2.example.com
d33441ab6305   hyperledger/fabric-peer:latest
```

Query Chaincode

peer chaincode query -C mychannel -n basic -c '**{"Args":["GetAllAssets"]}**'

```
~/fabric/fabric-samples/test-network ➦ 006942b peer chaincode query -C mychannel -n basic -
c '{"Args":["GetAllAssets"]}'
[{"ID":"asset1","color":"blue","size":5,"owner":"Tomoko","appraisedValue":300},{"ID":"asset2","c
olor":"red","size":5,"owner":"Brad","appraisedValue":400},{"ID":"asset3","color":"green","size":
10,"owner":"Jin Soo","appraisedValue":500},{"ID":"asset4","color":"yellow","size":10,"owner":"Ma
x","appraisedValue":600},{"ID":"asset5","color":"black","size":15,"owner":"Adriana","appraisedVa
lue":700},{"ID":"asset6","color":"white","size":15,"owner":"Michel","appraisedValue":800}]
```

Conclusion

- Fabric Test Network
- Query and Invoke Transactions