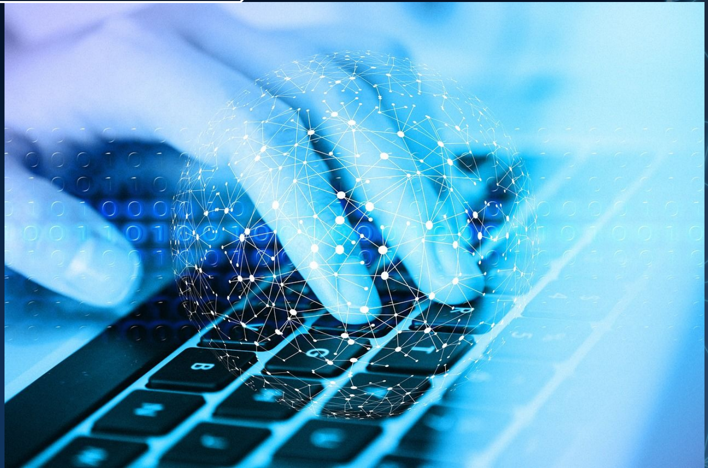
Building Blockchain Applications using Java



Overview of Blockchain

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network.

What Makes Blockchain Different?

01

Security and Immutability

02

Transparency

03

Decentralization

04

Consensus Mechanism

When to Use It?

Decision-making Process

1. Is it necessary to track the lifecycle of assets?
2. Do you need a sole custodian or extensive chain of custodians?
3. Who is your target audience?
4. What types of transactions will occur?
5. What transaction speed is required?
6. How large are the transactions?

When to Use It?

Decision-making Process

7

How will the data be maintained?

8

What level of trust will be built in the network?

Use Cases

Supply Chain Management

Digital Identity

Digital Payments

Types of Blockchain

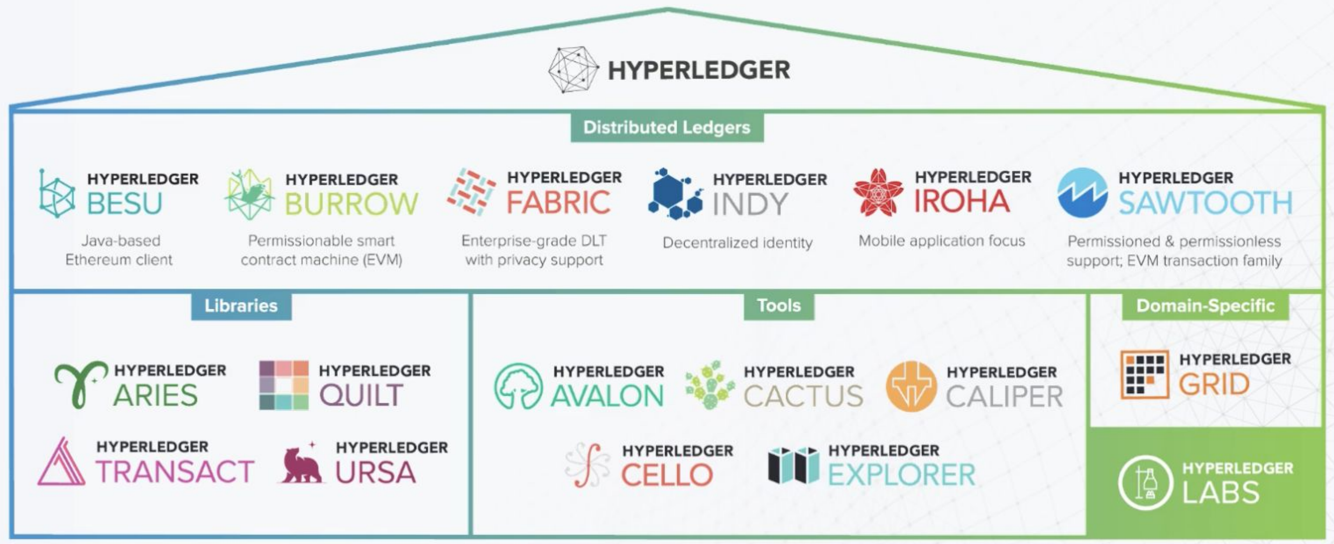
	Public Blockchains	Private Blockchains
ACCESSIBILITY	Open to anyone	Restricted to authorized entities
DECENTRALIZATION	Fully decentralized	Varying degrees of decentralization
TRANSACTION SPEED	Slower due to complex consensus	Faster due to simpler consensus
SECURITY	Robust security with transparency	Controlled security with privacy
CONSENSUS MECHANISM	PoW, PoS, etc.	PBFT, PoA

Types of Blockchain

	Public Blockchains	Private Blockchains
GOVERNANCE	Decentralized governance with decisions made by the community	Controlled governance by designated entities
INTEROPERABILITY	Polkadot, Cosmos, Hyperledger Cacti	

Overview of Hyperledger

Primary Goal: Construct enterprise-grade blockchain networks



Hyperledger Fabric Features and Components

Privacy and Confidentiality

Scalability and Performance

Chaincode

Pluggable Consensus

Hyperledger Fabric Features and Components

Peer Nodes

Ordering Service

Membership Services Provider

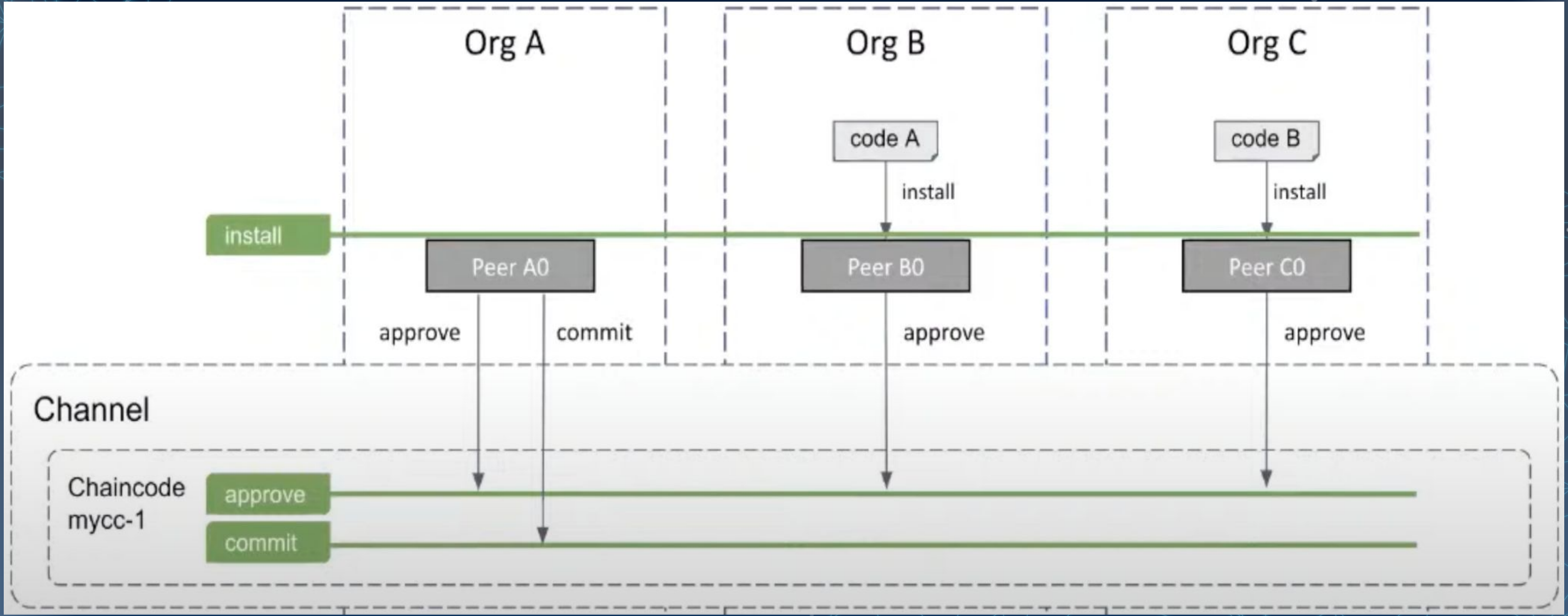
Channels

SDK

Chaincode Lifecycle

1. Package the chaincode.
2. Install the chaincode on peers.
3. Approve the chaincode.
4. Commit the chaincode definition to the channel.

Deploying Chaincode



Writing Chaincode

Define the contract's metadata.

```
@Contract(
        name = "basic",
        info = @Info(
                title = "Asset Transfer",
                description = "The hyperlegendary asset transfer",
                version = "0.0.1-SNAPSHOT",
                license = @License(
                        name = "Apache 2.0 License",
                        url = "http://www.apache.org/licenses/LICENSE-2.0.html"),
                contact = @Contact(
                        email = "a.transfer@example.com",
                        name = "Asset Transfer",
                        url = "https://hyperledger.example.com")))
```

Writing Chaincode

Create initial assets on the ledger.

```
@Transaction(intent = Transaction.TYPE.SUBMIT)
public void InitLedger(final Context ctx) {
    ChaincodeStub stub = ctx.getStub();

    CreateAsset(ctx, assetID: "asset1", color: "blue", size: 5, owner: "Tomoko", appraisedValue: 300);
    CreateAsset(ctx, assetID: "asset2", color: "red", size: 5, owner: "Brad", appraisedValue: 400);
    CreateAsset(ctx, assetID: "asset3", color: "green", size: 10, owner: "Jin Soo", appraisedValue: 500);
    CreateAsset(ctx, assetID: "asset4", color: "yellow", size: 10, owner: "Max", appraisedValue: 600);
    CreateAsset(ctx, assetID: "asset5", color: "black", size: 15, owner: "Adrian", appraisedValue: 700);
    CreateAsset(ctx, assetID: "asset6", color: "white", size: 15, owner: "Michel", appraisedValue: 700);

}
```

Writing Chaincode

Create a new asset on the ledger.

```java
@Transaction(intent = Transaction.TYPE.SUBMIT)
public Asset CreateAsset(final Context ctx, final String assetID, final String color, final int size,
    final String owner, final int appraisedValue) {
    ChaincodeStub stub = ctx.getStub();

    if (AssetExists(ctx, assetID)) {
        String errorMessage = String.format("Asset %s already exists", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_ALREADY_EXISTS.toString());
    }

    Asset asset = new Asset(assetID, color, size, owner, appraisedValue);
    // Use Genson to convert the Asset into string, sort it alphabetically and serialize it into a json string
    String sortedJson = genson.serialize(asset);
    stub.putStringState(assetID, sortedJson);

    return asset;
}
```

Writing Chaincode

Retrieves an asset with the specified ID from the ledger.

```java
@Transaction(intent = Transaction.TYPE.EVALUATE)
public Asset ReadAsset(final Context ctx, final String assetID) {
    ChaincodeStub stub = ctx.getStub();
    String assetJSON = stub.getStringState(assetID);

    if (assetJSON == null || assetJSON.isEmpty()) {
        String errorMessage = String.format("Asset %s does not exist", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_NOT_FOUND.toString());
    }

    Asset asset = genson.deserialize(assetJSON, Asset.class);
    return asset;
}
```

Writing Chaincode

Changes the owner of an asset on the ledger.

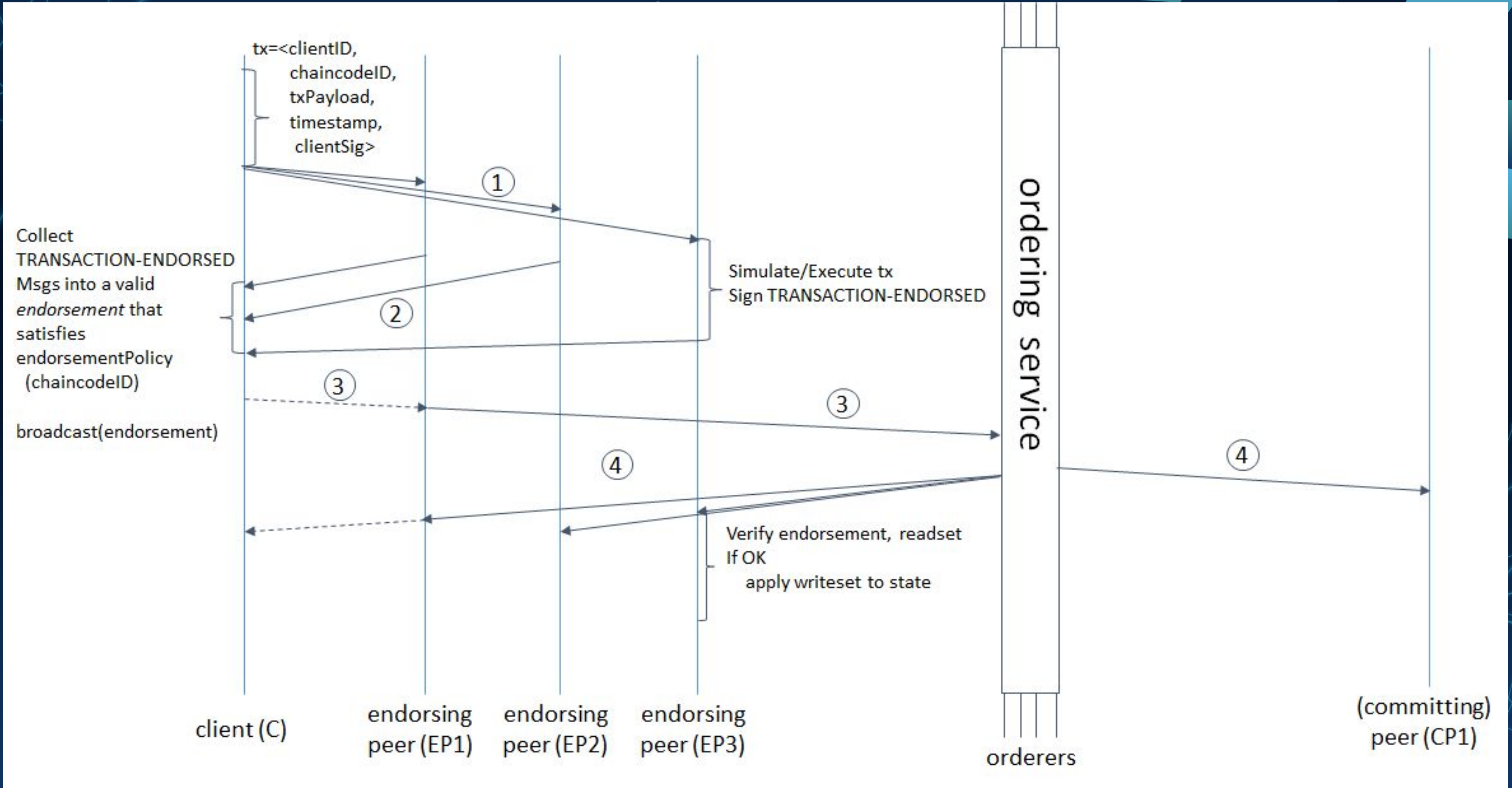
```java
@Transaction(intent = Transaction.TYPE.SUBMIT)
public String TransferAsset(final Context ctx, final String assetID, final String newOwner) {
    ChaincodeStub stub = ctx.getStub();
    String assetJSON = stub.getStringState(assetID);

    if (assetJSON == null || assetJSON.isEmpty()) {
        String errorMessage = String.format("Asset %s does not exist", assetID);
        System.out.println(errorMessage);
        throw new ChaincodeException(errorMessage, AssetTransferErrors.ASSET_NOT_FOUND.toString());
    }

    Asset asset = genson.deserialize(assetJSON, Asset.class);

    Asset newAsset = new Asset(asset.getAssetID(), asset.getColor(), asset.getSize(), newOwner, asset.getAppraisedValue());
    // Use a Genson to conver the Asset into string, sort it alphabetically and serialize it into a json string
    String sortedJson = genson.serialize(newAsset);
    stub.putStringState(assetID, sortedJson);

    return asset.getOwner();
}
```

Writing Chaincode

Retrieves all the assets from the ledger.

```
@Transaction(intent = Transaction.TYPE.EVALUATE)
public String GetAllAssets(final Context ctx) {
    ChaincodeStub stub = ctx.getStub();

    List<Asset> queryResults = new ArrayList<~>();

    // To retrieve all assets from the ledger use getStateByRange with empty startKey & endKey.
    // Giving empty startKey & endKey is interpreted as all the keys from beginning to end.
    // As another example, if you use startKey = 'asset0', endKey = 'asset9' ,
    // then getStateByRange will retrieve asset with keys between asset0 (inclusive) and asset9 (exclusive) in lexical order.
    QueryResultsIterator<KeyValue> results = stub.getStateByRange("", "");

    for (KeyValue result: results) {
        Asset asset = genson.deserialize(result.getStringValue(), Asset.class);
        System.out.println(asset);
        queryResults.add(asset);
    }

    final String response = genson.serialize(queryResults);

    return response;
}
```

Transaction Flow



Hyperledger Fabric Demo

Steps:

(1) Build the chaincode.

```
./gradlew clean build
```

(2)
```
./network.sh down
```

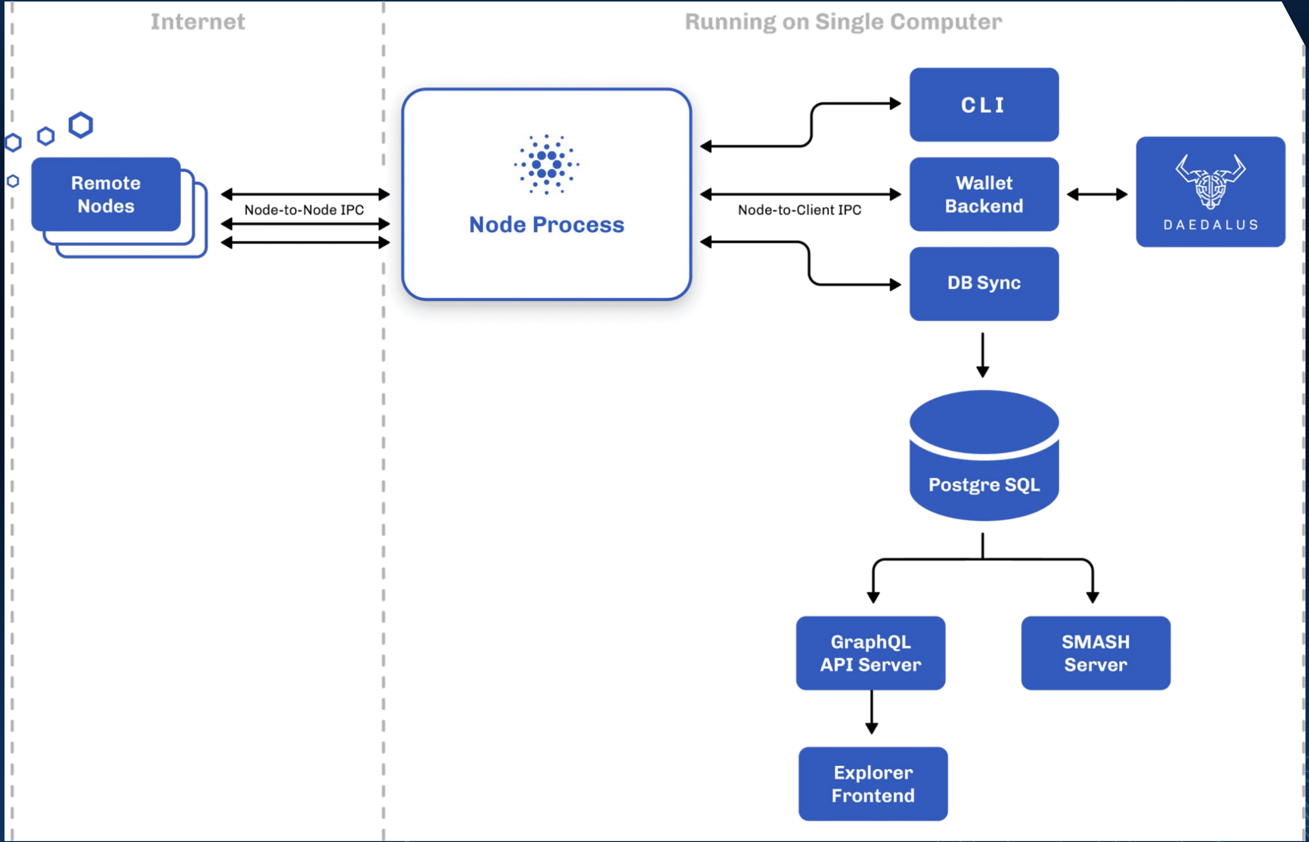
(3)
```
./network.sh up createChannel -ca -c mychannel -s couchdb
```

(4)
```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-java/ -ccl java
```

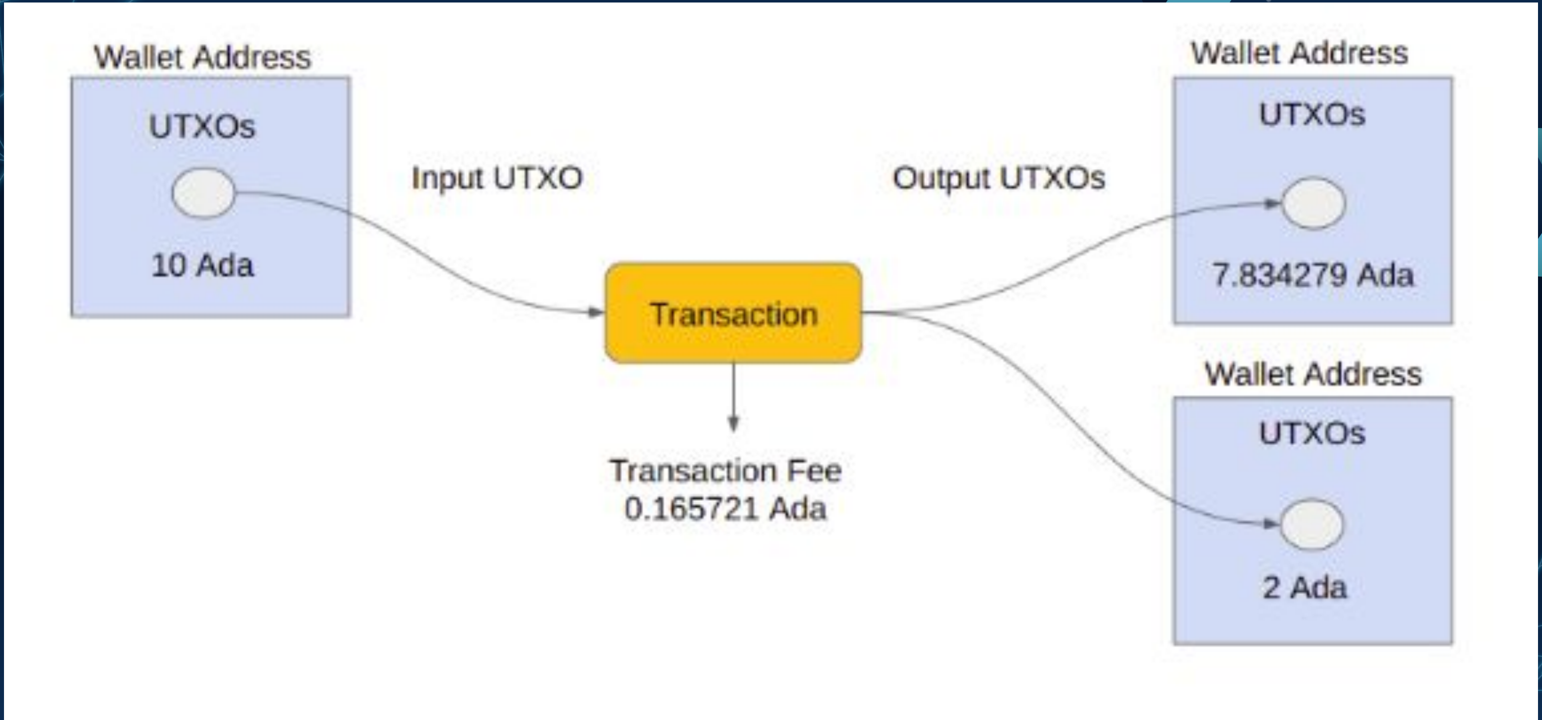
(5) Run the unit tests with mocked values.

Overview of Cardano

Cardano Network



eUTXO Transaction Model



Cardano Java Examples GitHub URL:

https://github.com/lley154/cardano-java-examples

Thank you

Do you have any questions?