Hyperledger Fabric

Introduction to Hyperledger Fabric.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Hyperledger Fabric

Hyperledger Fabric is a **permissioned** distributed ledger developed by IBM under the umbrella of **Linux Foundation's Hyperledger project**.

- Fabric is an **enterprise-grade** platform that offers **modularity** and **versatility** for a broad set of industry use cases.
- Being **open-source**, the project is supported by a **large community** of developers.

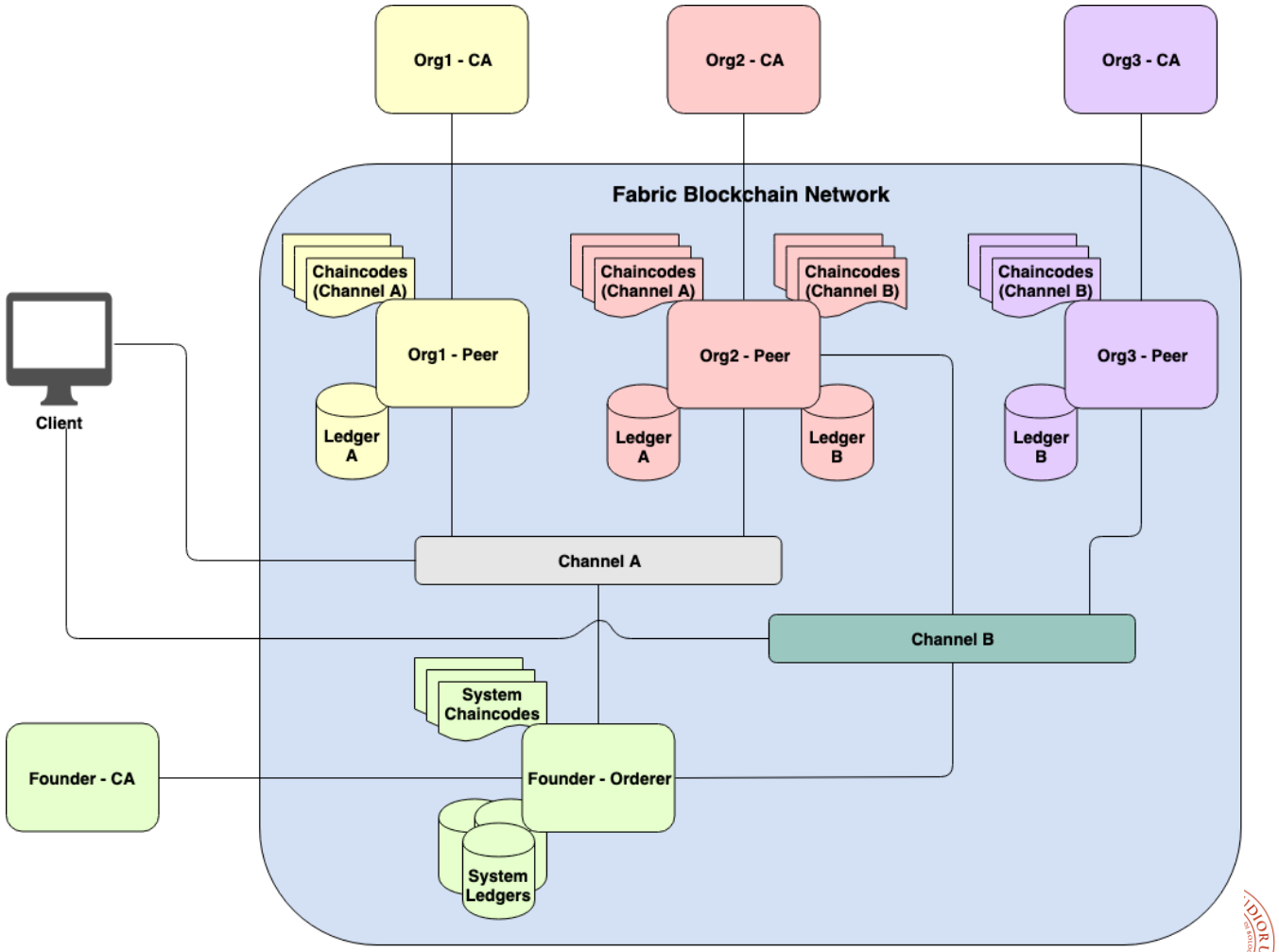
Hyperledger Fabric Key Features

- **Permissioned** architecture.
- **Highly modular**.
- **Pluggable consensus**.
- Smart Contracts (**multi-language support**: Go, Java, Nodejs).
- **Low latency**.
- **Flexible endorsement model** for achieving consensus across required organizations.

Hyperledger Fabric Architecture

Key elements:

- Channel.
- Client.
- Peer.
- Orderer.
- Certificate Authority.
- Chaincode.
- Ledger.



*** Actually, the system chaincodes and ledgers are also deployed on peers but they were not put into the figure for the sake of simplicity ***

Channel

A **Hyperledger Fabric channel** is a private **"subnet"** of communication between two or more specific network members, for the purpose of conducting **private** and **confidential** transactions.

Each transaction on the network **is executed on a chan**nel, where each party must be **authenticated** and **authorized** to transact on that channel.

One organization can take part in **multiple channels** at the same time.

Actors

The main actors are:

- **Client**: is considered to be an application that interacts with Fabric blockchain network.
- **Peer**: is a node that **commits transactions** and **maintains** the state and a copy of the ledger.
- **Orderer**: is a service responsible for **ordering transactions**, **creating** a new block of ordered transactions, and **distributing** a newly created block to all peers on a relevant channel.
- **Certificate Authority**: is responsible for managing user certificates.

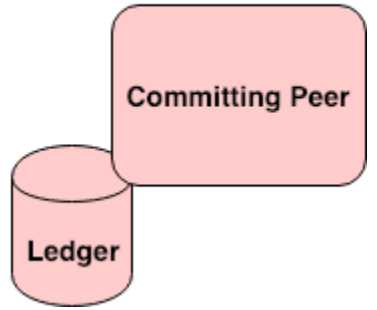
Peers

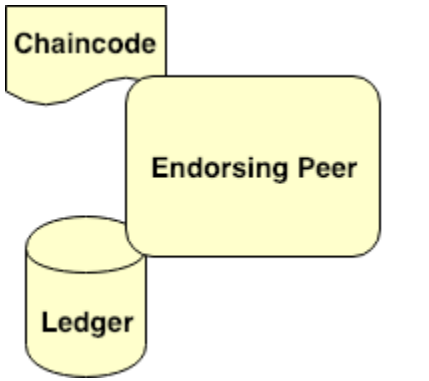
Committing Peer

- Stores only a local ledger on it.

Endorsing Peer

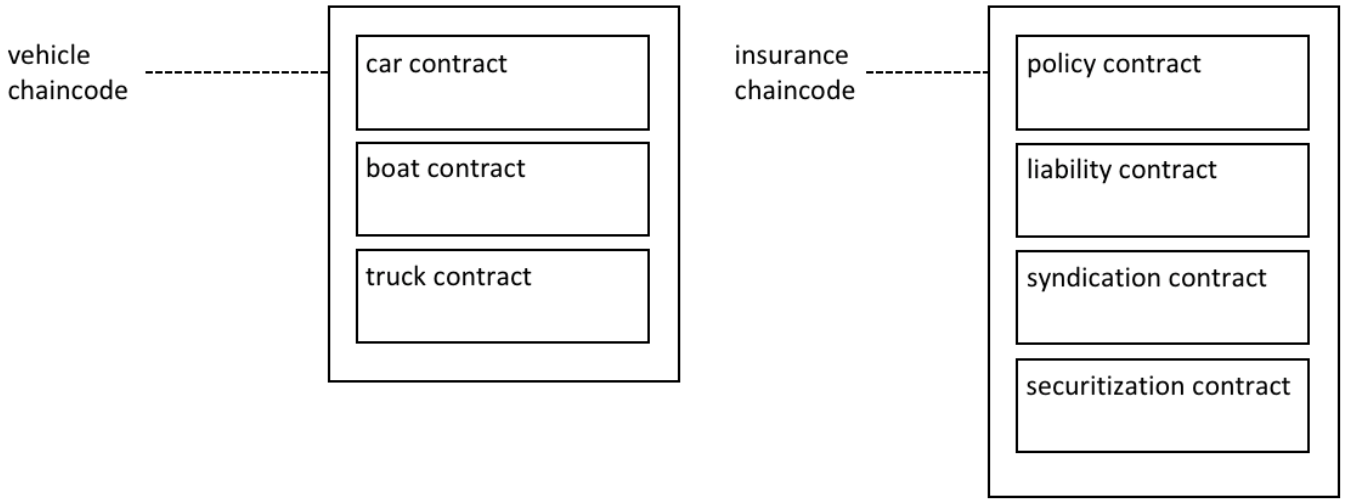
- The special function of an **endorsing** peer occurs with respect to a **particular chaincode** and consists in **endorsing a transaction** before it is committed.
- Every chaincode may specify an **endorsement policy** that may refer to a set of endorsing peers.
- Endorsement policy defines **which peers need to agree** on the results of a transaction before the transaction can be added onto ledgers of all peers on the channel.





Chaincode

In Hyperledger Fabric, a **Chaincode** is typically used by administrators to **group related smart contracts** for deployment.

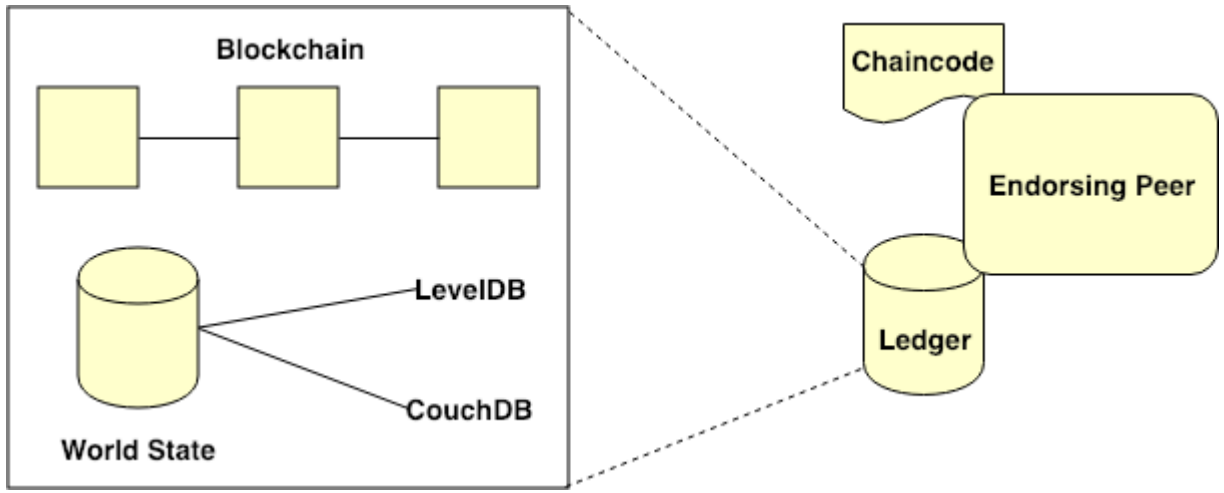


While deploying the chaincode, an admin can define an **endorsement policy** to the chaincode.

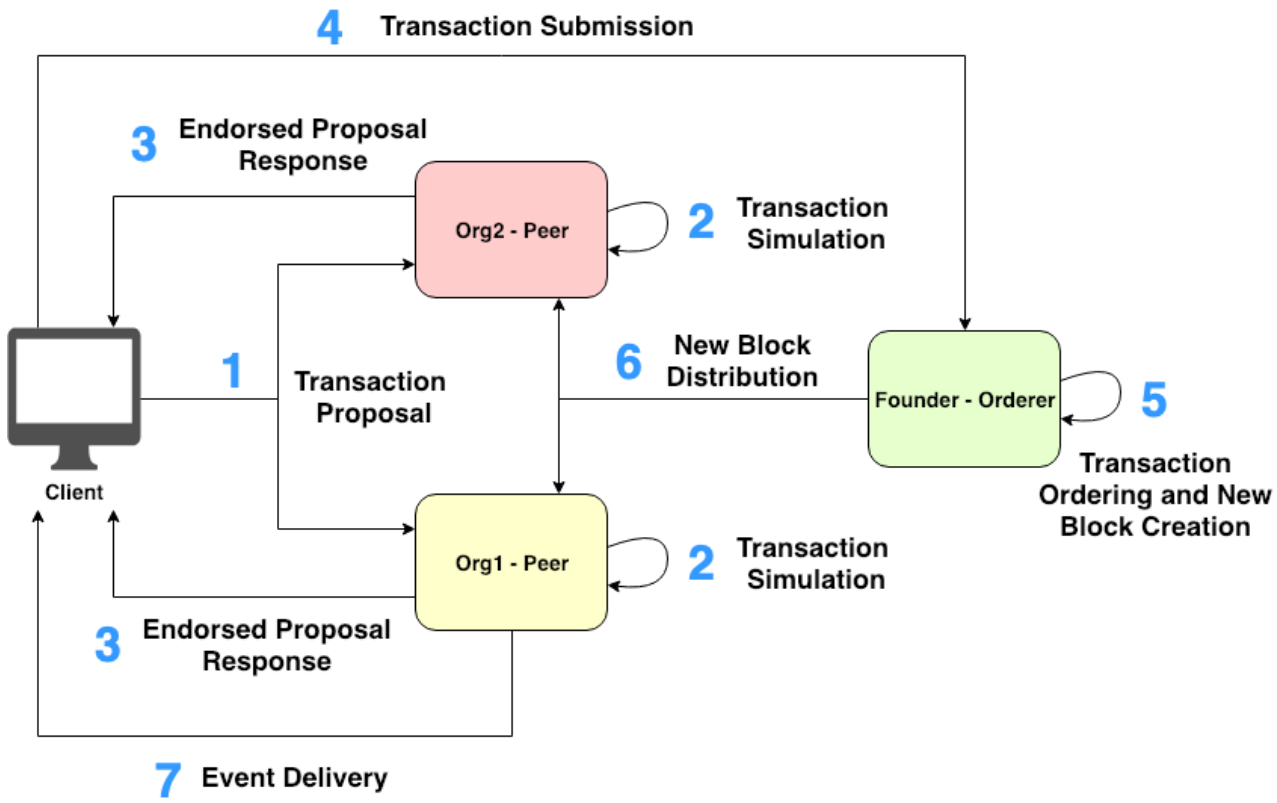
Ledger

The interior components inside the Peer's ledger include **Blockchain** and **World State**:

- **Blockchain** holds the history of all transactions for every chaincode on a particular channel.
- **World State** maintains the current state of variables for each specific chaincode.



Transaction Flow



Consensus

Hyperledger Fabric's design relies on ***deterministic* consensus algorithms**, any block validated by the peer is guaranteed to be final and correct.

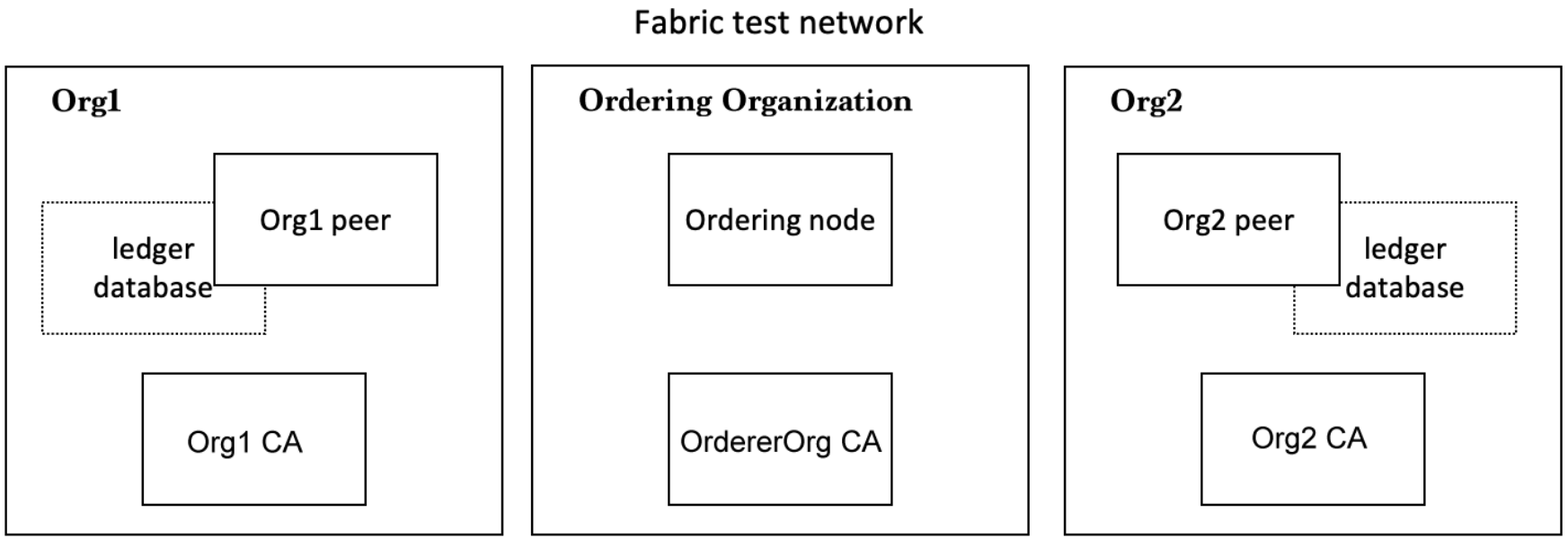
Determinism is guaranteed by the **ordering service**. The recommended ordering service is **Raft**, a crash fault tolerant (**CFT**) ordering service based on an implementation of Raft protocol.

Have we solved consensus?!

Deterministic agreement, but probabilistic termination!

Hands on

Fabric test-network



Starting the Network

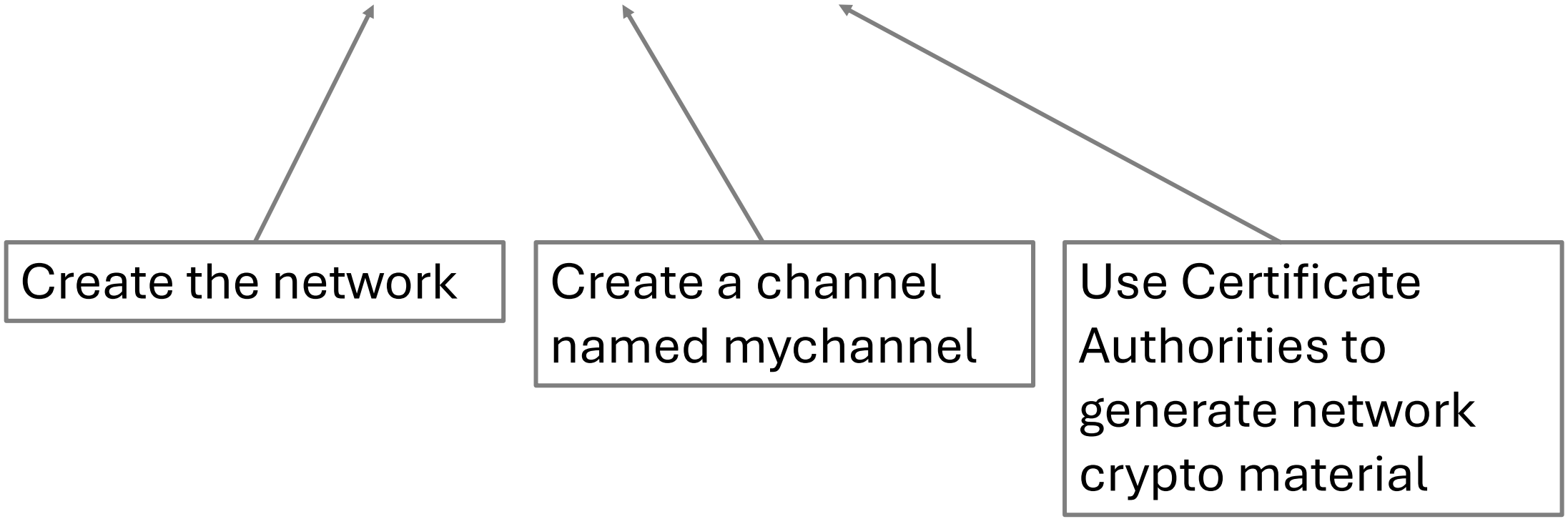
Under fabric-samples/test-network execute:

./network.sh up createChannel -ca

Starting the Network

Under fabric-samples/test-network execute:

./network.sh up createChannel -ca



Starting the Network

Verify the actual deployment of containers using the command:

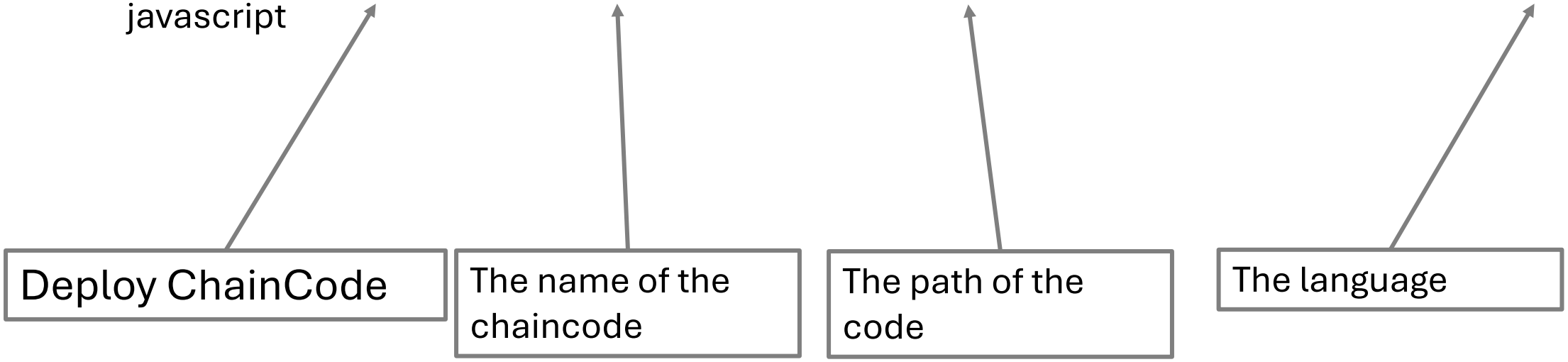
docker ps

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Deploy a Chaincode

Now we can deploy a chaincode on channel mychannel:

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



Asset Transfer Basic

```
async InitLedger(ctx)
async CreateAsset(ctx,...)
async ReadAsset(ctx,...)
async UpdateAsset(ctx,...)
async DeleteAsset(ctx,...)
async AssetExists(ctx,...)
async TransferAsset(ctx,...)
async GetAllAssets(ctx)
```

Execute the Chaincode

Under fabric-samples/asset-transfer-basic/application-javascript run:

npm install

node app.js

It will create a **wallet** directory specifically for that network. If you recreate the network, you will have to delete the directory.

submitTransaction()	**change the state of the blockchain.**
evaluateTransaction()	**simply read the state.** There will be no
trace	of this transaction on the ledger because it will not be sent to the orderer.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Exercise

Re-use the code to:

- Create a new asset.
- Check that the asset exists.
- Delete it.

Comment all the other transactions as they have already been done.

Example

Create a new smart contract or modify asset-transfer-basic:

- Create a method that return «hello world».

World State

To interact with the world state there are the two methods of ctx.stub:

- getState

<async> getState(key)

Retrieves the current value of the state variable key

Parameters:

Name	Type	Description
key	string	State variable key to retrieve from the state store

Returns:

Promise for the current value of the state variable

Type

Promise.<Array.<byte>>

World State

To interact with the world state there are the two methods of ctx.stub:

- putState

<async> putState(key, value)

Writes the state variable `key` of value `value` to the state store. If the variable already exists, the value will be overwritten.

Parameters:

Name	Type	Description
key	string	State variable key to set the value for
value	Array.<byte> | string	State variable value

Returns:

Promise will be resolved when the peer has successfully handled the state update request or rejected if any errors

Type

Promise

Input and Output Parameters

Remember that all parameters (input and output of methods) are either strings or bytes.

Generally, it is best to use the **JSON serializer** and **deserializer**.

JSON.stringify() and JSON.parse()

In this way, the type of the value is maintained.

Example

Use Math.random() to return a random number.

Exercise

Create in the previous smart contract:

- A method that sums two values given in input.
- Create a method to store a JSON object in input.
- Create a method to read the object.