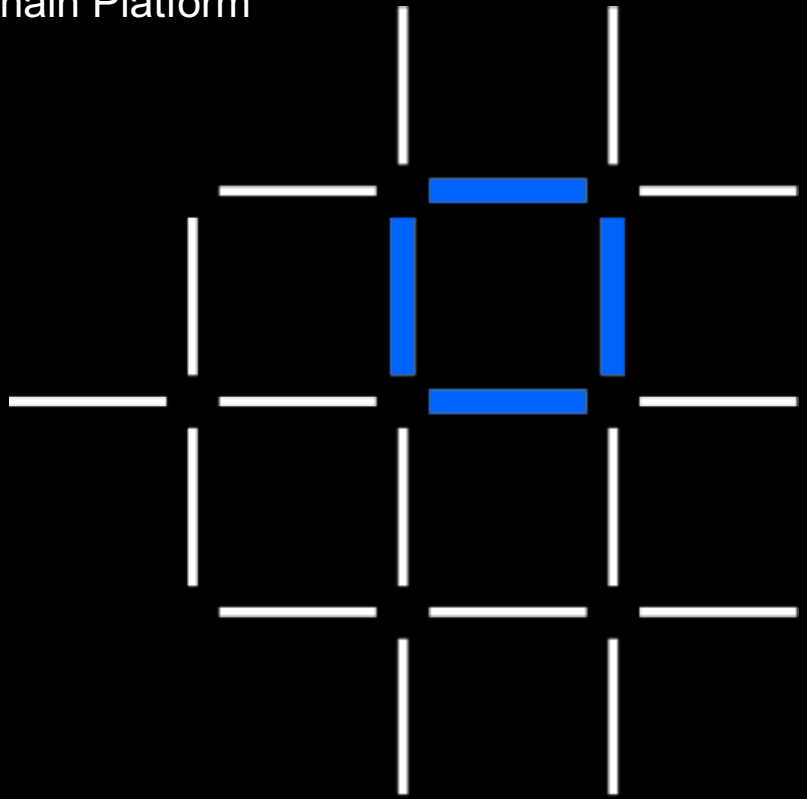
Hyperledger Fabric Deep Dive

Understand the technology that underpins IBM Blockchain Platform



Barry Silliman
Blockchain Enablement on IBM Z and LinuxONE
IBM North America Technical Sales
silliman@us.ibm.com

V1.4, 24 January 2020

IBM **Blockchain**

IBM

Overview

The very basics of Hyperledger Fabric

Hyperledger Fabric Concepts and Components

Understand the technology that underpins IBM Blockchain Platform

Blockchain concepts

- Consider the business concepts that comprise blockchain solutions:
  - **Assets** modeled as data structures
  - **Contracts** modeled as algorithms
  - **Transactions** modeled as invocations of algorithms
  - **Business Networks** modeled as peer-to-peer networks
  - **Participants** modeled as nodes on these networks

- IBM Blockchain Platform is based around **Hyperledger Fabric**
  - We will now look at how Hyperledger Fabric exposes these business concepts to the blockchain developer

A simple **consensus** example (1/4) – a puzzle!

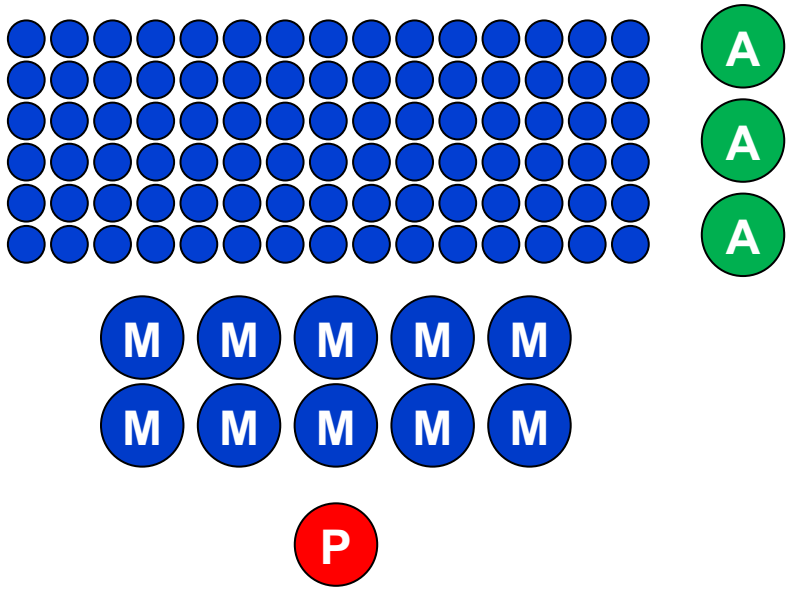
- **Situation:** A presenter in a room with 100 people, 10 of whom are mathematicians!
- **Problem:** Answer the question: **What is the square root of 2401?**
- **Question:** Can everyone agree on an answer?

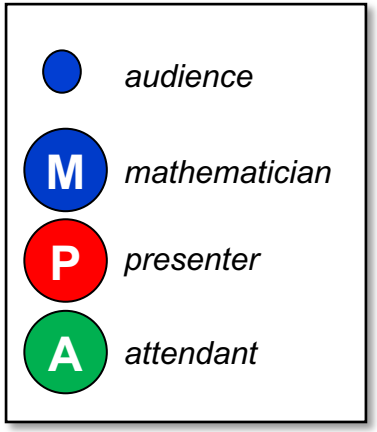
a puzzle
a process
a problem
a policy

You are here! →

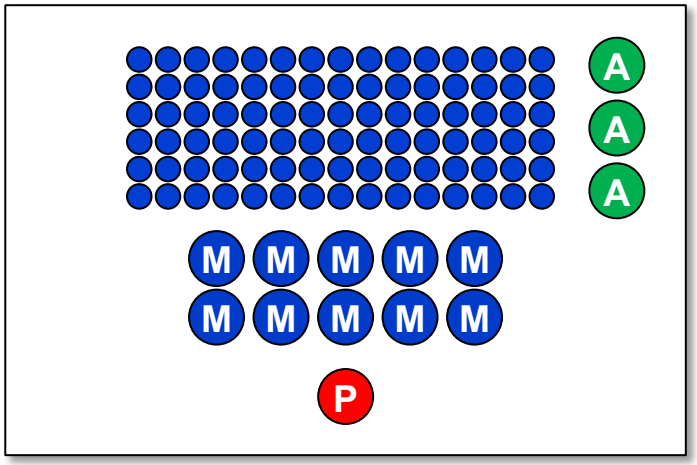
Or here! →

But not here ;) →





A simple **consensus** example (2/4) – a process

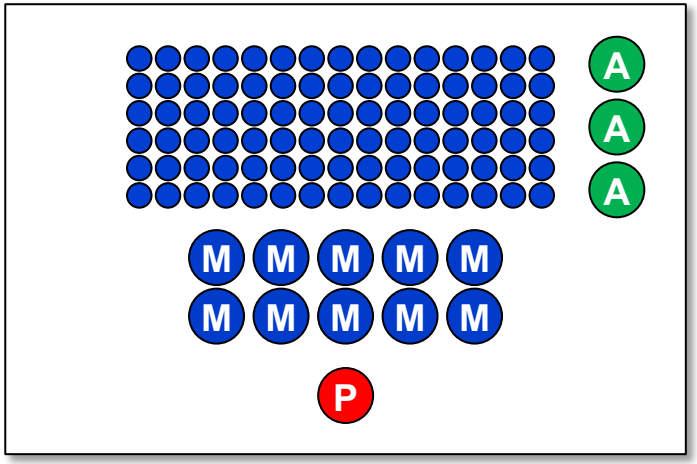


a puzzle
a process
a problem
a policy

Solution approach

1. The presenter asks every mathematician to calculate √2401
2. Every mathematician writes their answer on a piece of paper, signs with their (digital) signature
3. The presenter collects every mathematician's signed response
4. The room attendants distribute copies of the signed answer to every member of the audience
5. Every audience member checks that every mathematician agrees the answer, or not!
6. Everyone is happy; there is **consensus!**

A simple **consensus** example (3/4) – a problem



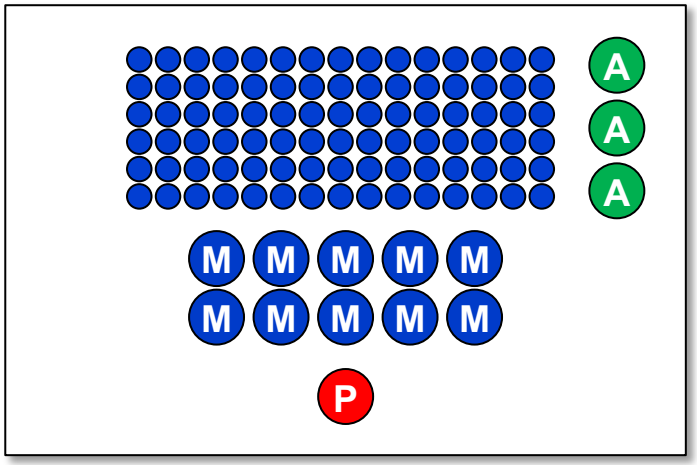
a puzzle
a process
a problem
a policy

Issues

1. How many mathematicians should the presenter ask?
2. How many mathematicians need to agree for an answer to be correct?
3. What happens if the mathematicians disagree on the answer, or the presenter selects a bad answer?

→ Everyone needs to agree what constitutes a **good** answer

A simple **consensus** example (4/4) – a policy



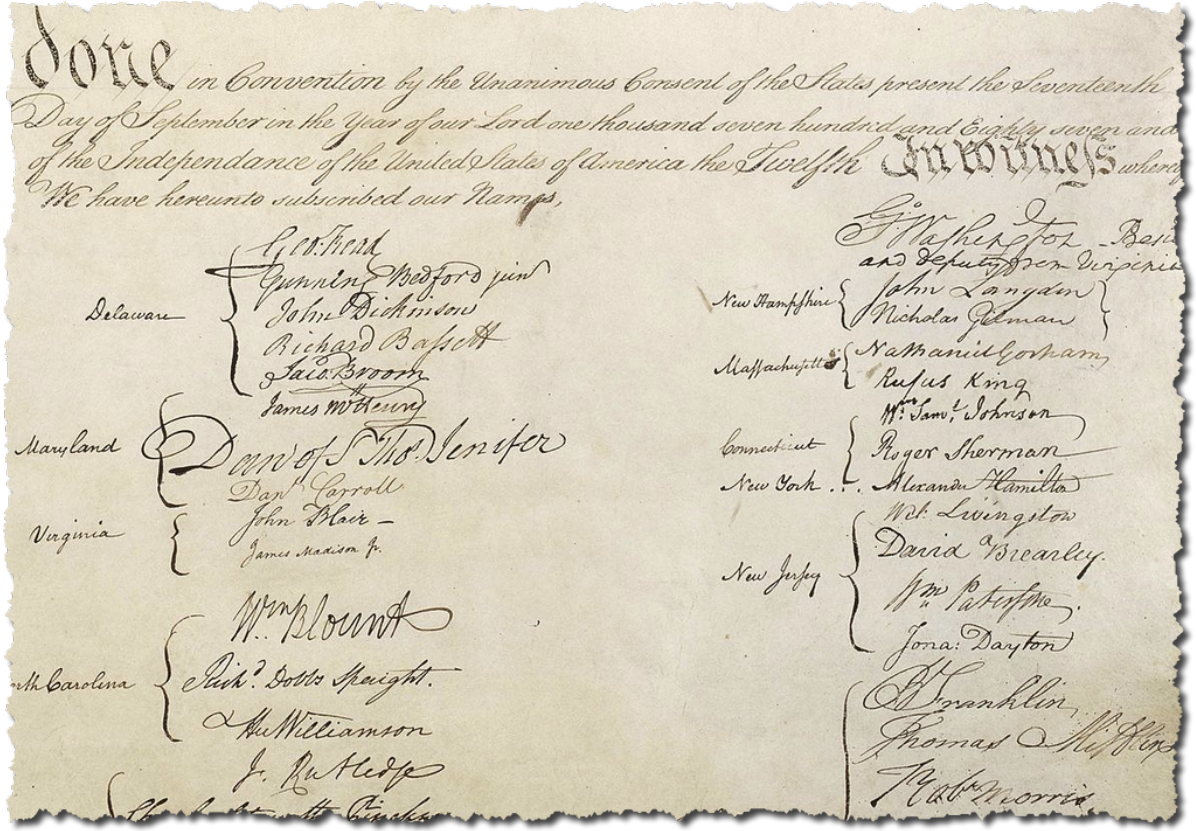
a puzzle
a process
a problem
a policy

The importance of policy

1. The audience agrees on a **policy for good answers**
2. Example policies:
   - **all mathematicians must agree**
   - **the majority of mathematicians must agree**

→ Using this **policy** everyone can now agree on what makes a **good** answer

Transaction Endorsement



Hyperledger Fabric **multi-party transactions**

- A transaction describes a **change to a system**

  Examples:
  - a change in bank balance
  - a student receives an academic qualification
  - a parcel is delivered

- Traditionally transactions are signed by a **single organization**
  - a bank signs payment transactions
  - a university signs graduation transactions
  - a logistics company signs parcel receipt transactions

- **Multi-party transactions** are **the heart** of Hyperledger Fabric
  - e.g. buyer and seller both sign car transfer transaction
  - e.g. logistics and delivery companies both sign parcel transaction

- **Understand multi-party transactions and you will understand Fabric!**

```
car transfer transaction:

 identifier: 1234567890

 proposal:
  input: {CAR1, seller, buyer}
  signature: input*seller

 response:
  output:
    {CAR1.oldOwner=seller,
     CAR1.newOwner=buyer}
  signatures:
    output*seller
    output*buyer
```

*A transaction to transfer a car is signed by both the **buyer** and the **seller***

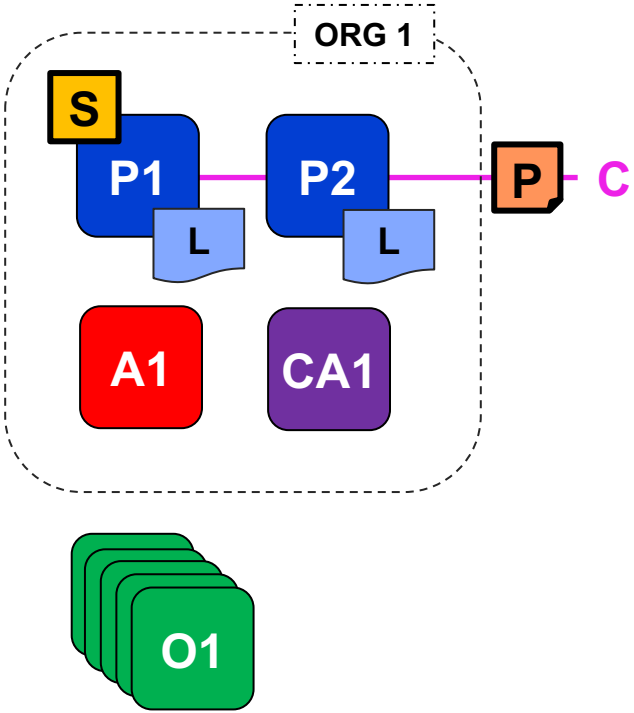
Overview

The very basics of Hyperledger Fabric

Hyperledger Fabric Concepts and Components

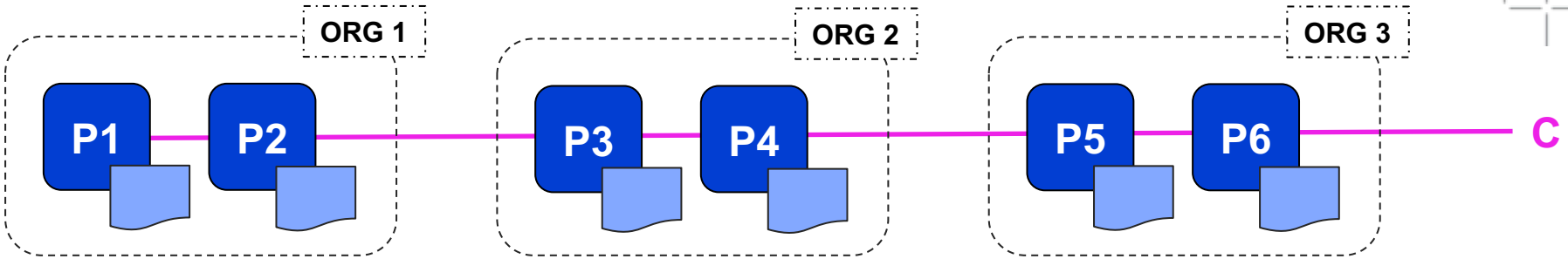
Understand the technology that underpins IBM Blockchain Platform

Hyperledger Fabric Components and Concepts



- Peers
- Ledgers
- Channels
- Applications
- Policies
- Smart Contracts
- Certificate Authorities
- Ordering Service

A Hyperledger Fabric **network**

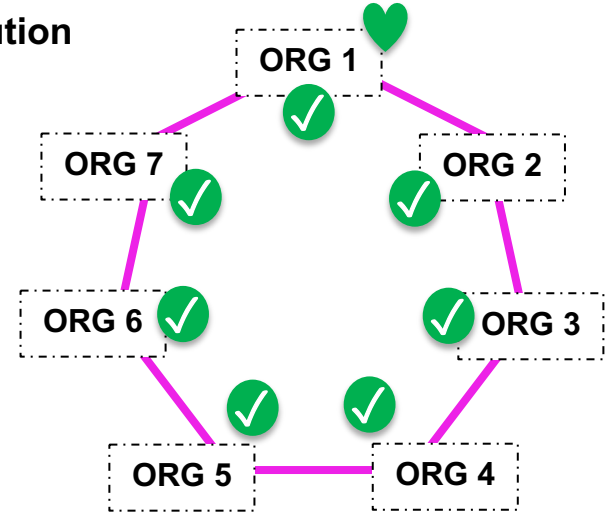


- A Hyperledger Fabric network comprises a **set of nodes**
- The most common type of node is a **peer**
  - Peers are owned by **different organizations**
  - Organizations can own **any number of nodes** (2 or 3 is common)
- Peers are connected using **channels**
  - The channel defines the scope of your **network**
  - Each channel has a **single ledger** that is replicated to every peer that is a member of it

The network is governed using policies

- Organizations in a consortium agree to form a network using a set of **policies**
- A network's policies encode **each organization's rights**, like a **constitution**

  Examples:
  - Every organization has the same authority over the network
  - All organizations must agree on any change to the network
  - A majority of organizations must agree on any change to the network
  - A majority of organizations including a particular organization must agree a change
- Transition between constitutions possible with agreement
- Hyperledger Fabric uses **identity** and **policy** throughout
  - Just like real world relationships between organizations



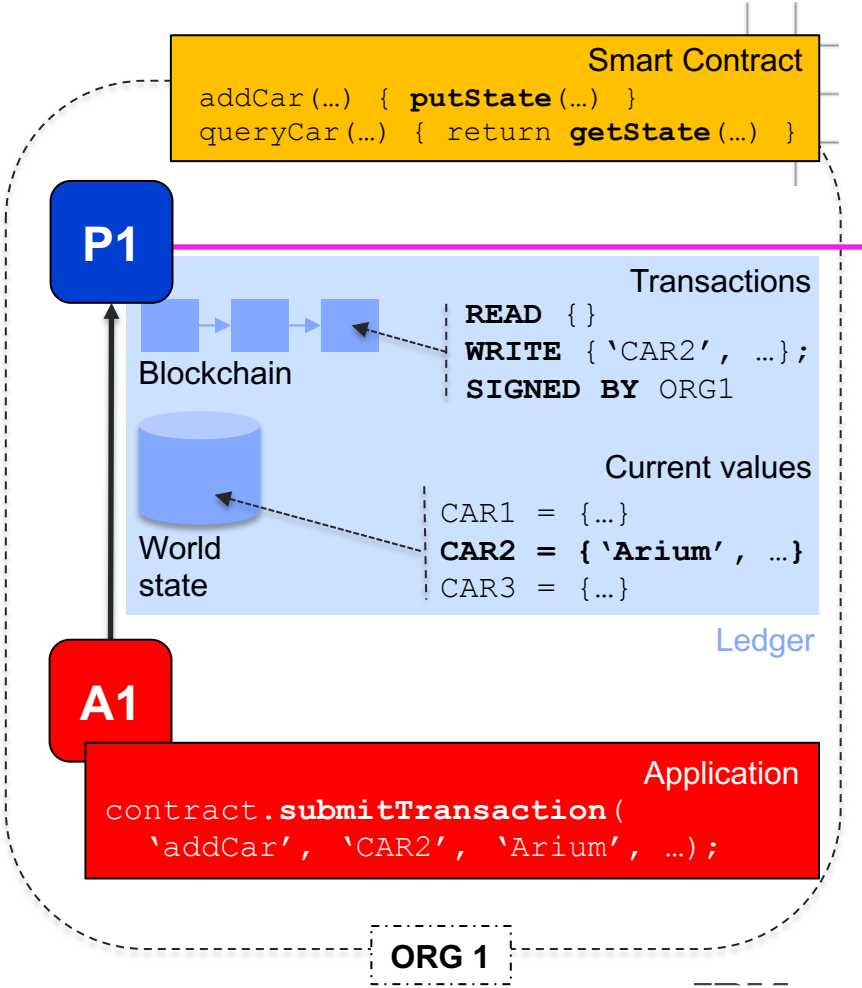
Understanding the ledger

- The ledger comprises two data structures
  - A **blockchain**: an immutable log of transactions shared across the peers in the network
  - A **world state**: a database of current asset values, derived from the blockchain

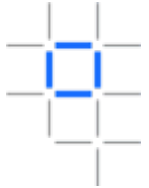
- Smart contracts read to and write from the world state (they are like database stored procedures)

- Applications submit transactions, each of which asks the network to invoke a smart contract with a set of parameters. Transaction order & output is agreed on by the network through **consensus.**

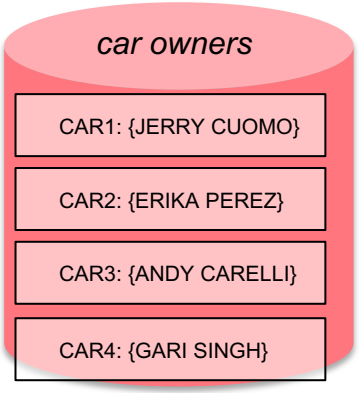
- World state may be complemented by **private data collections**



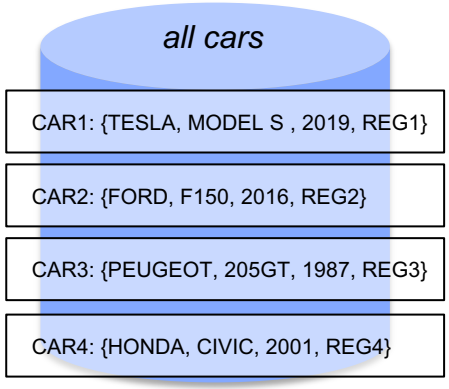
An example ledger showing **private** and **public** state

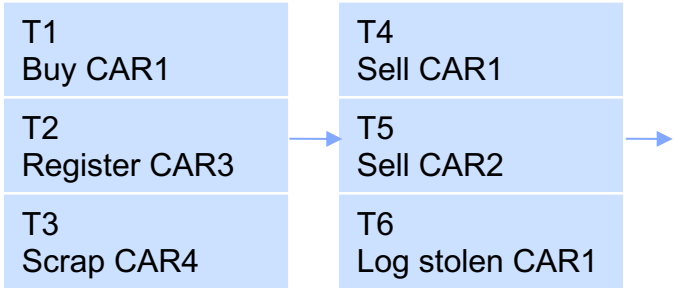


PRIVATE to DMV, Police



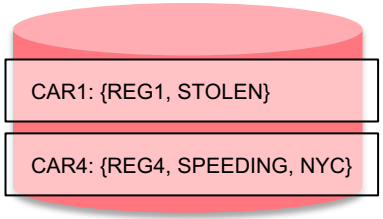
PUBLIC to all orgs



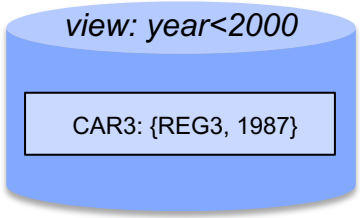


Blockchain

PRIVATE to Police



PUBLIC to all orgs

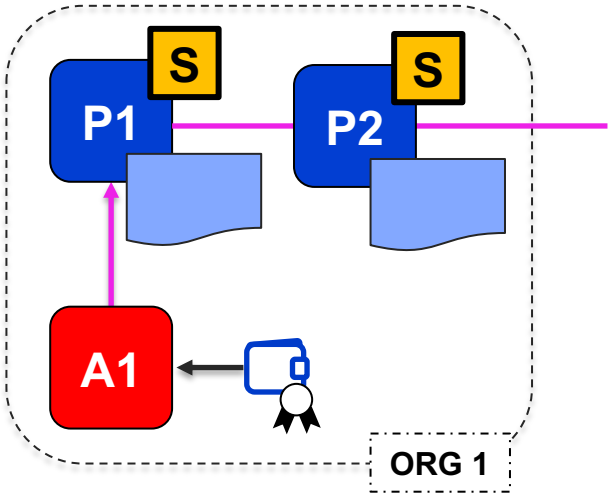


IBM

The basic structure of the **client application**

- Building an application that interacts with a Hyperledger Fabric network is extremely simple:

    1. **Connect** to a gateway using an identity from a wallet
    2. **Select** an available network and smart contract
    3. **Submit** transactions and check their responses
    4. **Disconnect** from the gateway

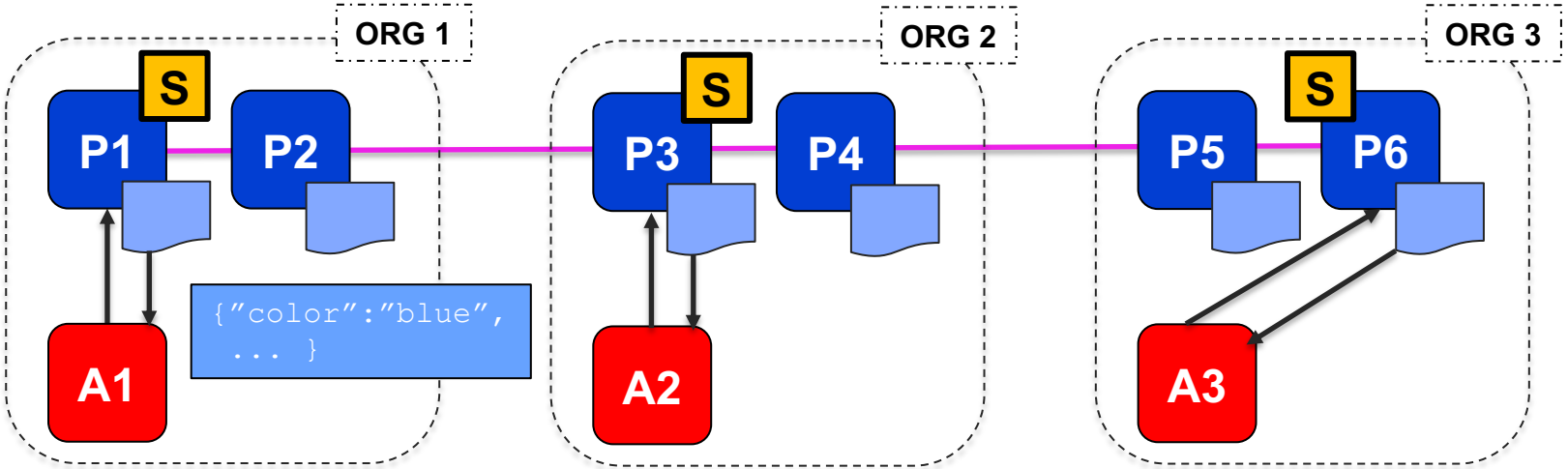


- The complexity of dealing with the network (e.g., consensus) is hidden from the application programmer
    - All interactions are through a Software Development Kit (SDK)

IBM

How applications **query** the ledger

- **Querying** the ledger is the simplest operation an application can perform!
  - They can query any peer's ledger to determine the value of an object; every instance is the same!
  - Applications typically query their **own organization's peers**

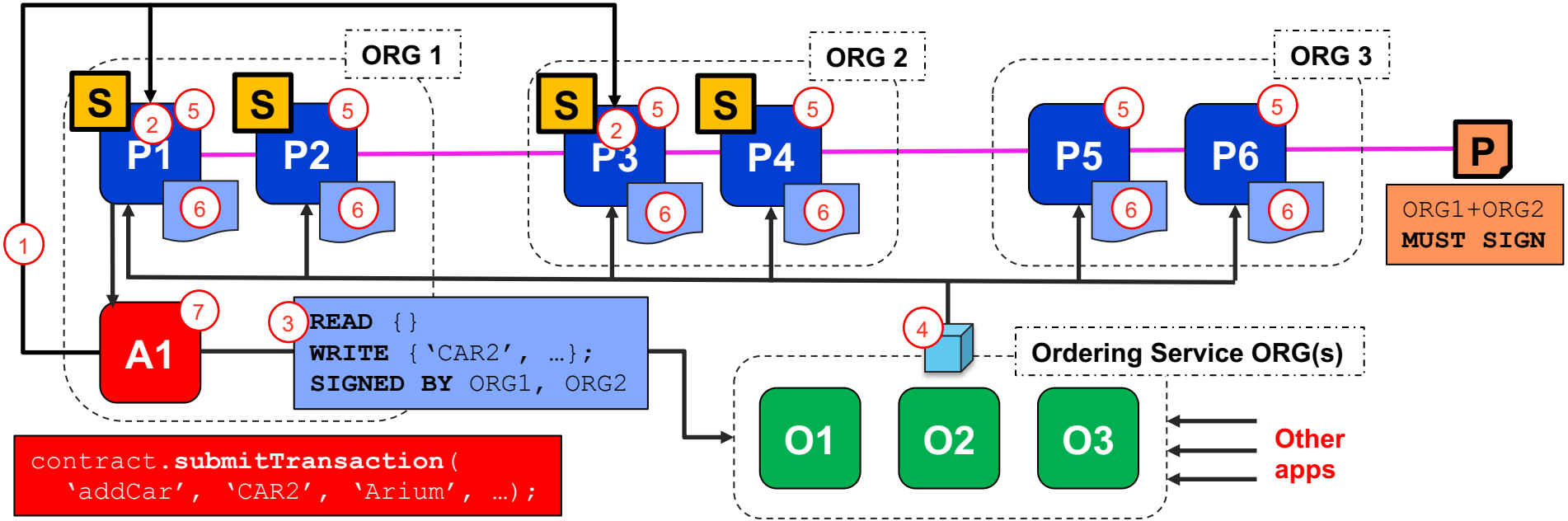



```
car1 = contract.evaluateTransaction(
  'queryCar', 'CAR1');
```

How applications **submit** transactions to the network

1. Propose
2. Execute
3. Submit
4. Create block
5. Validate
6. Update
7. Notify

- Applications use **smart contracts** installed on endorsing **peers** to create **multi-party txs**
- **Ordering Service** groups transactions into **blocks** which are **distributed to every node**
- **Every peer validates every transaction** using the smart contract's **endorsement policy**



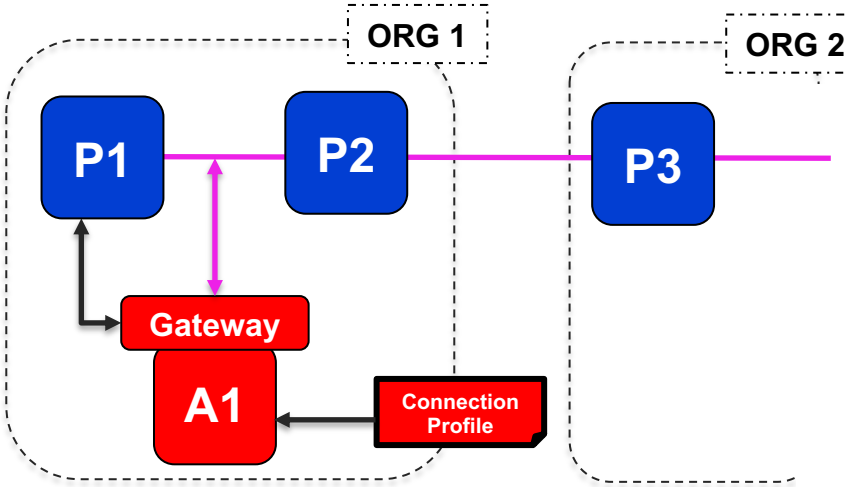
IBM

Understanding application **gateways**

- Network topology constantly changing
  - Peers, Orderers, CAs, smart contracts, ledgers

- **Gateways** isolate business logic from topology
  - Allows client applications to focus on the **WHAT** not the **HOW**
  - A conceptual part of the client SDK

- Gateways handle all interaction with the network
  1. Client SDK uses a connection profile to specify the location of a peer
  2. Peer returns available network services (e.g., peers, smart contracts) to gateway
  3. Through the gateway object, the client SDK interacts with the channel's services

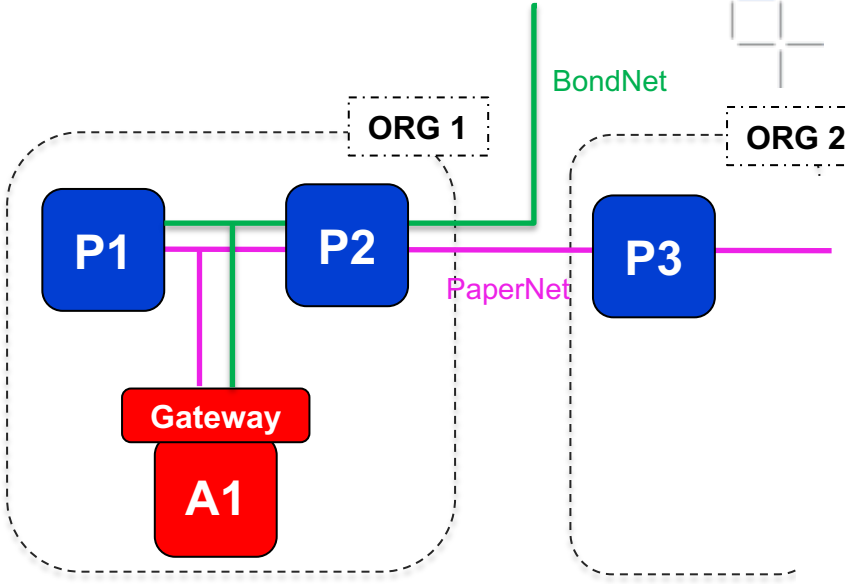



```
const connectionProfile = yaml.safeLoad('../gateway/driveNet.yaml');
const connectionOptions = {..., commitTimeout: 100,... };

const gateway = new Gateway();
await gateway.connect(connectionProfile, connectionOptions);
```

Connecting to **multiple networks**

- **Applications** can work with multiple networks at once
  - Each gateway instance can connect to multiple **channels**
  - All instantiated **smart contracts** accessible on the channel
  - Submit and evaluate multiple transactions
  - Can also connect to multiple **gateways** from within the same application

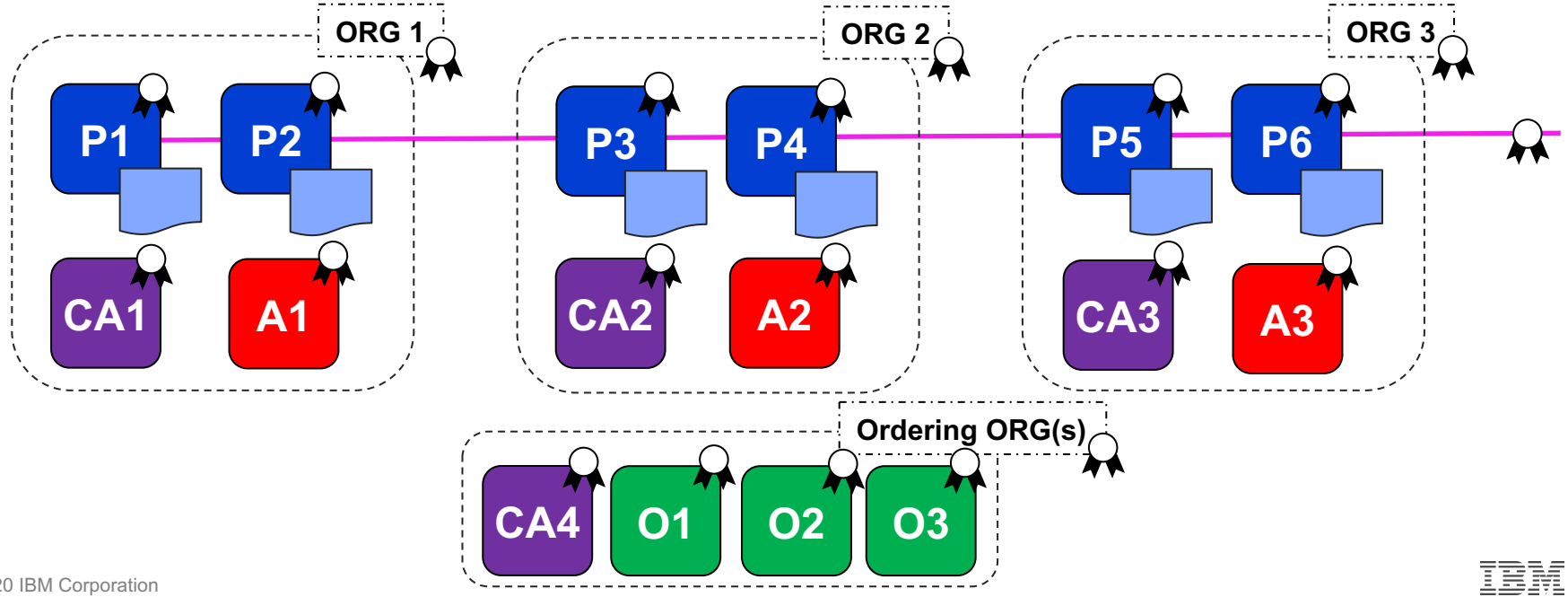
- Each network instance can use a separate **identity**




```
const network1 = await gateway.getNetwork('PaperNet');
const network2 = await gateway.getNetwork('BondNet');

const euroContract = await network1.getContract('EuroCommercialPaperContract');
const bondContract = await network2.getContract('BondContract');

const issueResponse = await euroContract.submitTransaction('issue', 'MagnetoCorp'...);
const sellResponse = await bondContract.submitTransaction('sell', 'BOND007'...)
```

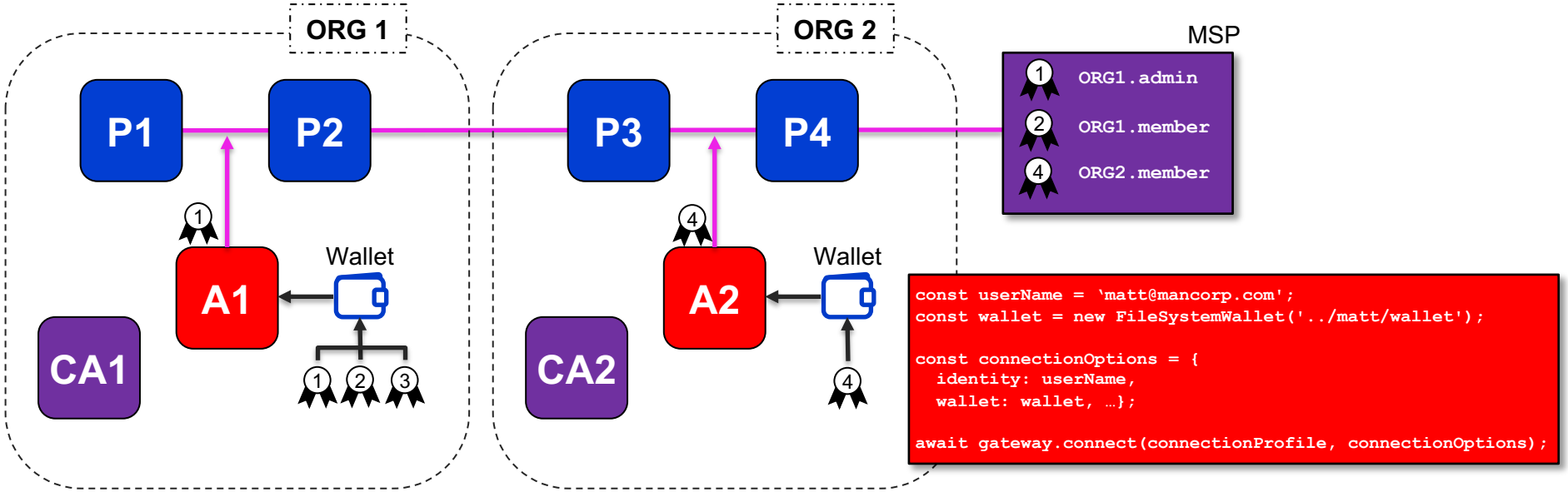
The importance of **identity**

- **Every component** has an associated X.509 identity issued by its organization's **Certificate Authority**
- The component uses its identity to determine its **organizational role**
- This **role determines** the level of **access** it has to network resources, e.g., read/write the ledger



Understanding **wallets**

- Wallets store the identities available to a component
- Applications can use multiple wallets & identities
- Applications connect to gateway with identity, as part of gateway connection options
- **Membership Services Provider** (MSP) uniquely identifies the identities and roles for a channel

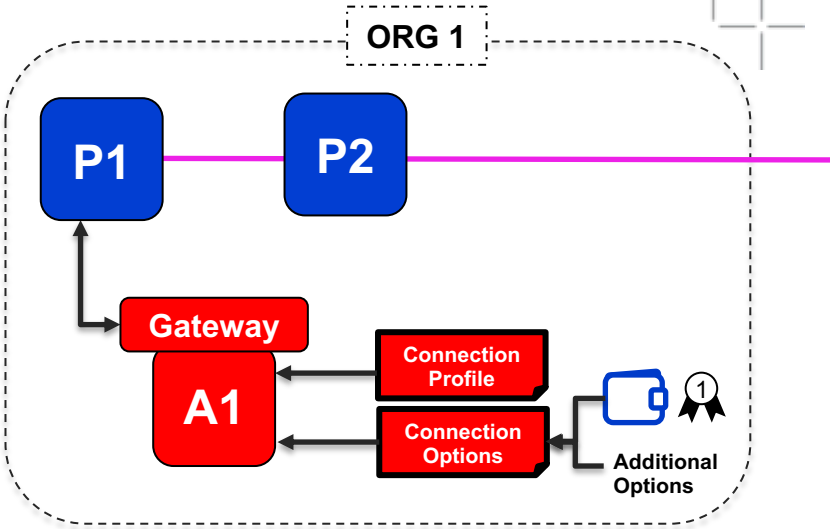


Fine-tuning **connection options**

- **Gateway** connections can be customized using connection options
  - **wallet** and **identity** must be set by app
  - All other options have sensible defaults

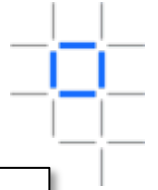
- Strategies codify most popular behaviour
  - e.g., "wait for any/all peers in my organization"
    - EventStrategies.MSPID_SCOPE_**ANY**FORTX
    - EventStrategies.MSPID_SCOPE_**ALL**FORTX

- Handlers for **programmable interactions**
  - User function gets control at appropriate point in transaction lifecycle with relevant topology - i.e., peers
    - e.g. startListening(), waitForEvents(), cancelListening()...
  - Keeps handler logic separate from business logic
  - Advanced feature; only required for special scenarios



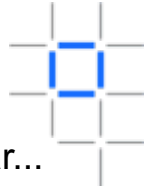

```
Gateway.Builder builder = Gateway.createBuilder();

builder.commitHandler(
   DefaultCommitHandlers.MSPID_SCOPE_ANYFORTX);

builder.identity(wallet,
   "admin").networkConfig(networkConfigPath);

Gateway gateway = builder.connect();
```

Bringing it all together: a **client application example**



```
Path walletPath = Paths.get("wallet");
Wallet wallet = Wallet.createFileSystemWallet(walletPath);          // Access file wallet

Path networkConfigPath = Paths.get("..", "..", "paper-network", "connection.json");
                                                                    // Identify a network using a CCP

Gateway.Builder builder = Gateway.createBuilder();                  // Create gateway to the network
builder.commitHandler(DefaultCommitHandlers.MSPID_SCOPE_ANYFORTX); // One peer from my organization to reply
builder.identity(wallet, "admin").networkConfig(networkConfigPath);

try (Gateway gateway = builder.connect()) {                         // connect to the gateway

  Network network = gateway.getNetwork("market1234");               // get the network
  Contract contract = network.getContract("commercial-paper");      // get contract in network

  String paper = contract.submitTransaction("issue", "IBM", "1000000", "2019-10-31")
                                                                    // issue paper by IBM
  contract.submitTransaction("move", paper, "ACME", "900000"))      // sell paper to ACME

} catch (Exception ex) {
  ex.printStackTrace();
}
```

Programming **smart contracts**



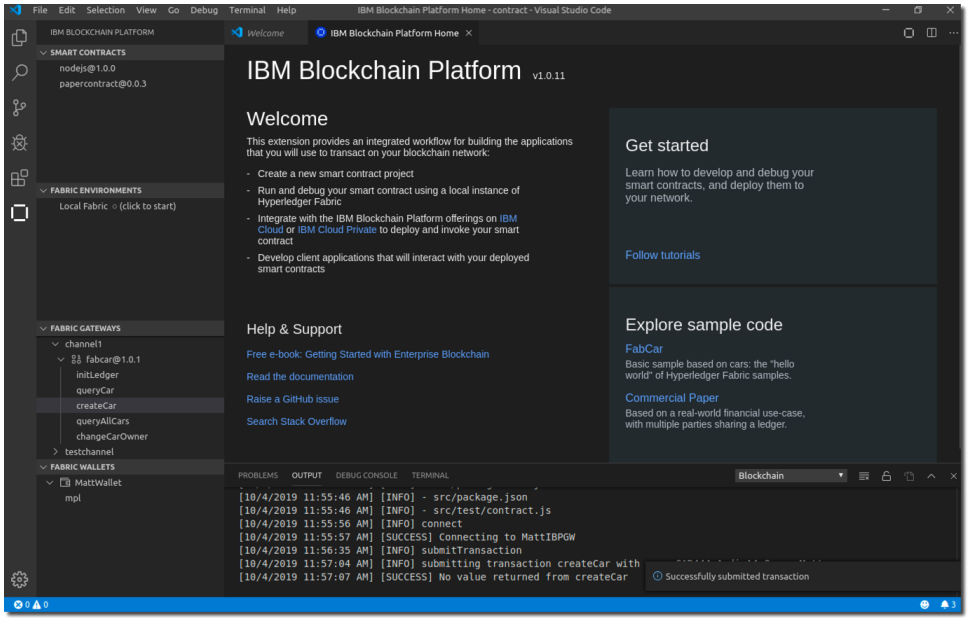
- A **smart contract** governs the **lifecycle of an object**. e.g., createCar, queryCar, updateCar, scrapCar...
- Implemented as a class; method contains relevant transaction logic in JavaScript, TypeScript, Java & Go*
- Installed on relevant organizations' **peer**(s). Execution results in a digitally signed transaction response.

```
@Contract(name="org.papernet.commercialpaper")
@Default
class CommercialPaper implements ContractInterface {

  public Context createContext(ChaincodeStub stub) {
    return new CommercialPaperContext(stub);
  }

  @Transaction
  public CommercialPaper issue(
                CommercialPaperContext ctx,
                String issuer,
                String value,
                String date,
                int faceValue) {
 // Interact with the ledger
     getState(existingPaper);
     putState(newPaper);
     ...
```


** Go under development*

Getting Started with Hyperledger Fabric

- The IBM Blockchain Platform makes it easy to get started with Hyperledger Fabric
- Download VSCode developer tool from VSCode marketplace. Linux, Windows and MacOS)
- Create local networks, smart contracts and applications. Move to multi-cloud when you're ready!



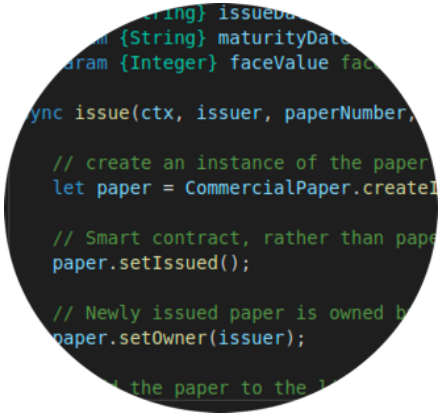
https://marketplace.visualstudio.com/items?itemName=IBMBlockchain.ibm-blockchain-platform

Summary

- The **Hyperledger Fabric transaction**
  - Structure drives system design and application architecture

- **Smart contract**
  - Contains transaction definitions for entire lifecycle of business object(s) stored in a decentralized ledger
  - built-in contract class makes programming easy

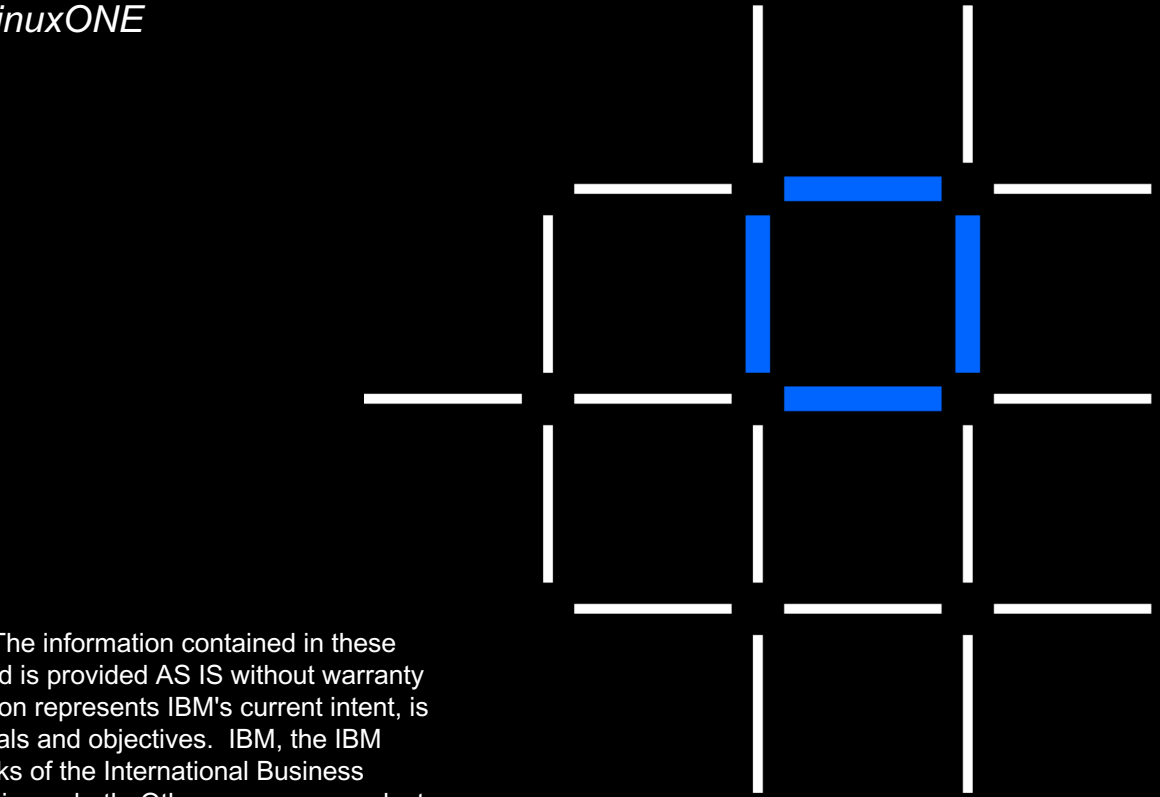
- **Application**
  - Consensus is complex, but the SDK makes it easy for applications
  - submitTransaction(), evaluateTransaction(), addListener()
  - gateway connectionOptions for ultimate customizability



Thank you

Barry Silliman
Blockchain Enablement on IBM Z and LinuxONE
IBM North America Technical Sales
silliman@us.ibm.com

IBM **Blockchain**



www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org

IBM


IBM