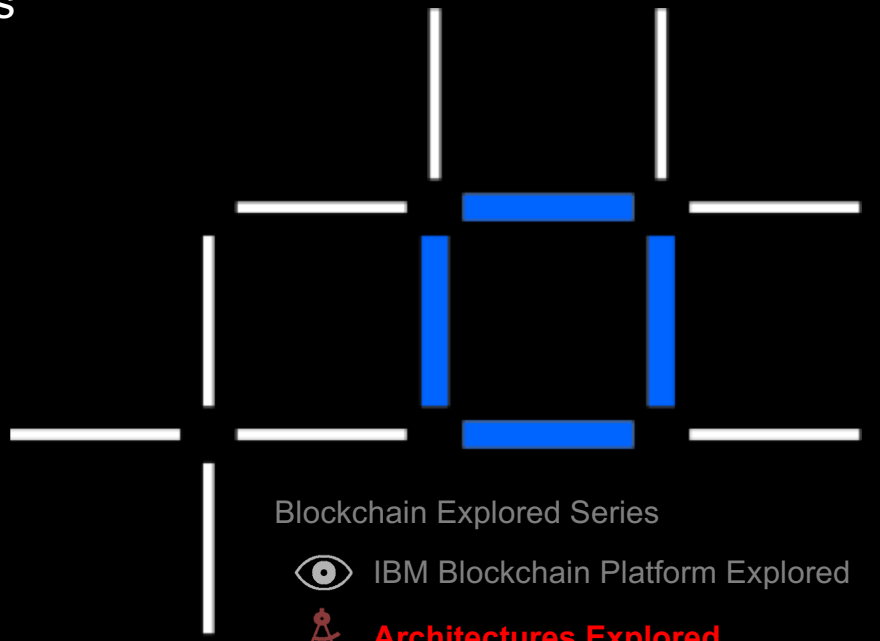
Architectures Explored

The systems architectures behind blockchain solutions



Blockchain Explored Series

- IBM Blockchain Platform Explored
- **Architectures Explored**
- Fabric Explored
- Composer Explored
- What's New

V2.10, 23 May 2018

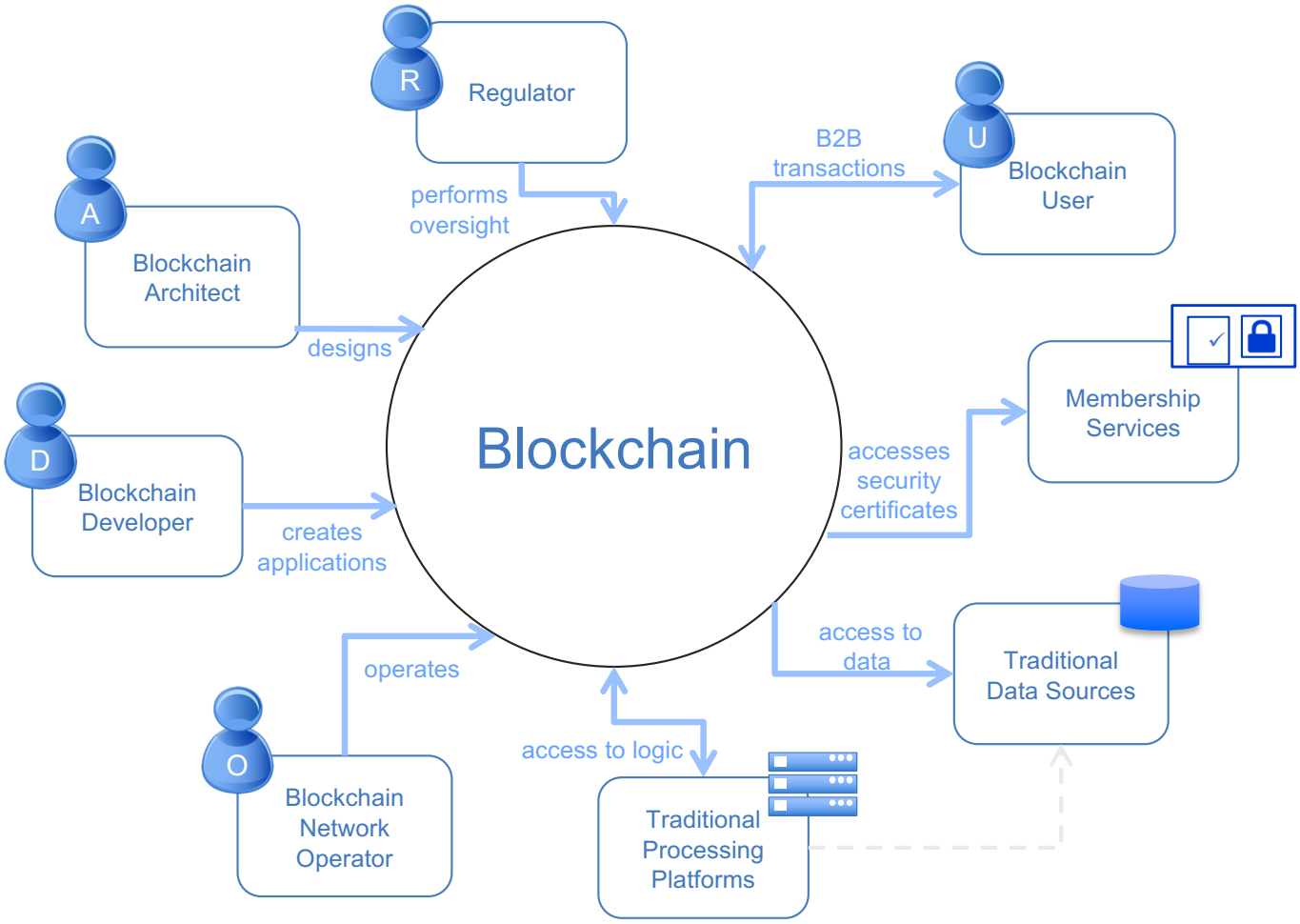
IBM **Blockchain**

IBM

Concepts and Components

Considerations for the Developer, Operator and Architect

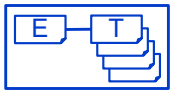
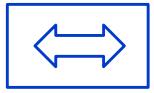
Actors in a blockchain solution



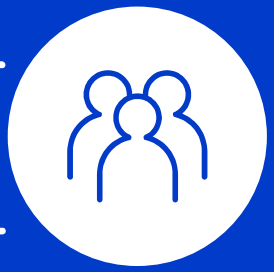
Actors in a blockchain solution

Blockchain Architect		Responsible for the architecture and design of the blockchain solution
Blockchain User		The business user, operating in a business network. This role interacts with the Blockchain using an application. They are not aware of the Blockchain.
Blockchain Regulator		The overall authority in a business network. Specifically, regulators may require broad access to the ledger's contents.
Blockchain Developer		The developer of applications and smart contracts that interact with the Blockchain and are used by Blockchain users.
Blockchain Operator		Manages and monitors the Blockchain network. Each business in the network has a Blockchain Network operator.
Membership Services		Manages the different types of certificates required to run a permissioned Blockchain.
Traditional Processing Platform		An existing computer system which may be used by the Blockchain to augment processing. This system may also need to initiate requests into the Blockchain.
Traditional Data Sources		An existing data system which may provide data to influence the behavior of smart contracts.

Components in a blockchain solution

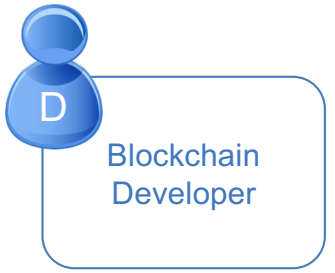
Ledger		A ledger is a channel's chain and current state data which is maintained by each peer on the channel.
Smart Contract		Software running on a ledger, to encode assets and the transaction instructions (business logic) for modifying the assets.
Peer Network		A broader term overarching the entire transactional flow, which serves to generate an agreement on the order and to confirm the correctness of the set of transactions constituting a block.
Membership		Membership Services authenticates, authorizes, and manages identities on a permissioned blockchain network.
Events		Creates notifications of significant operations on the blockchain (e.g. a new block), as well as notifications related to smart contracts.
Systems Management		Provides the ability to create, change and monitor blockchain components
Wallet		Securely manages a user's security credentials
Systems Integration		Responsible for integrating Blockchain bi-directionally with external systems. Not part of blockchain, but used with it.

Concepts and Components

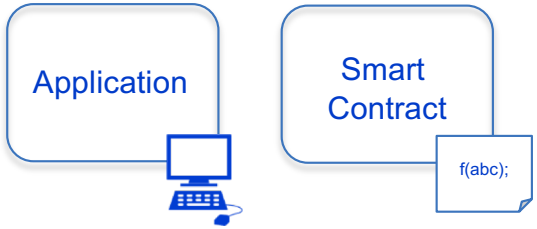


Considerations for the Developer, Operator and Architect

The blockchain developer



Blockchain developers' primary interests are…

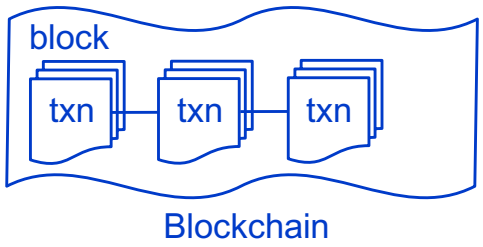


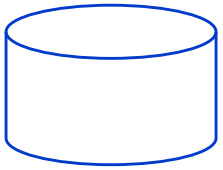
…and how they interact with the ledger and other systems of record:

They should NOT have to care about operational concerns, such as:

X Peers Consensus Security

A ledger often consists of two data structures

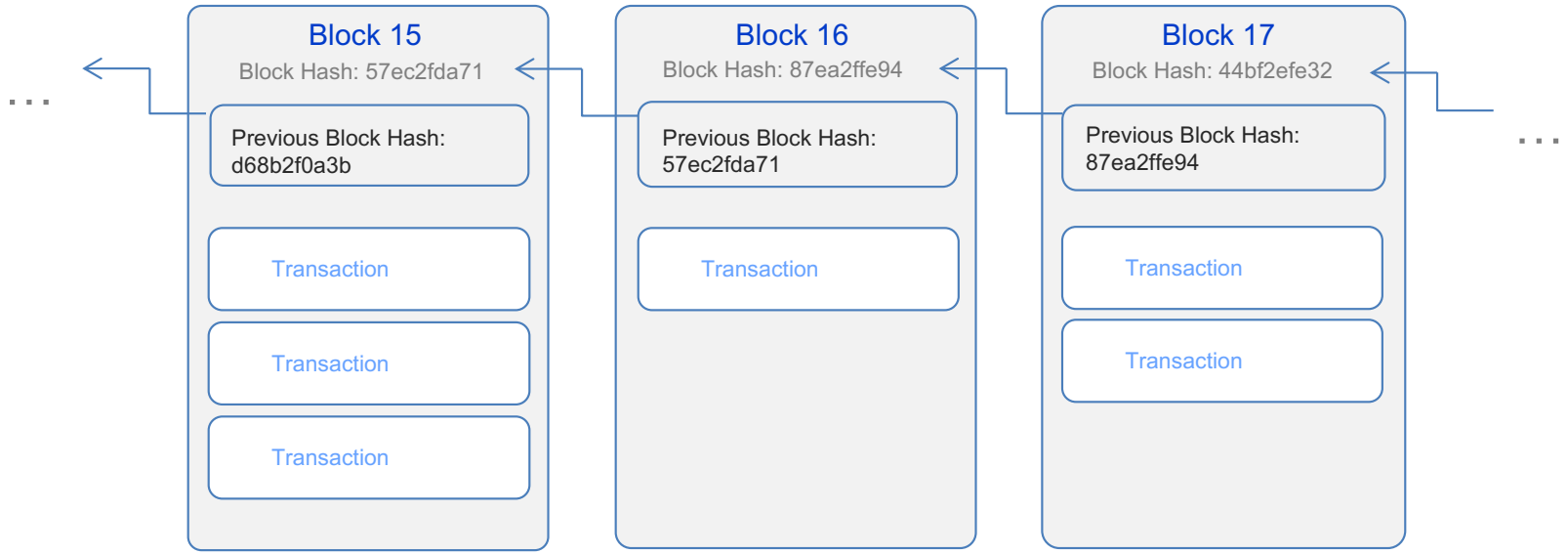




- Blockchain
  - A linked list of blocks
  - Each block describes a set of transactions (e.g. the inputs to a smart contract invocation)
  - Immutable – blocks cannot be tampered

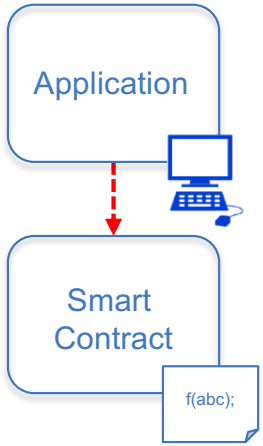
- World State
  - An ordinary database (e.g. key/value store)
  - Stores the combined outputs of all transactions
  - Not usually immutable

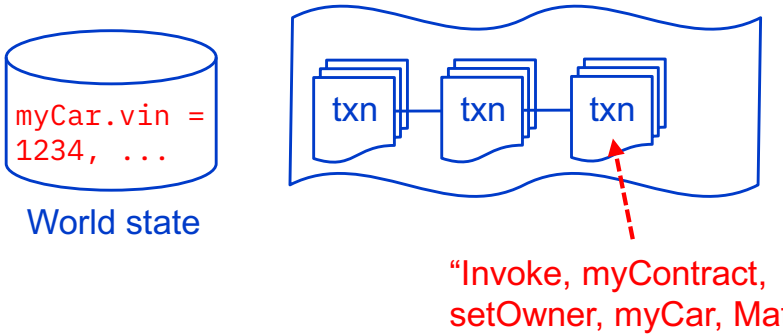
Block detail (simplified)



- A blockchain is made up of a series of blocks with new blocks always added to the end
- Each block contains zero or more transactions and some additional metadata
- Blocks achieve immutability by including the result of a hash function of the previous block
- The first block is known as the “genesis” block

Working with the ledger example: a change of ownership transaction





Transaction input - sent from application

```
invoke(myContract, setOwner,
       myCar, Matt)
…
```

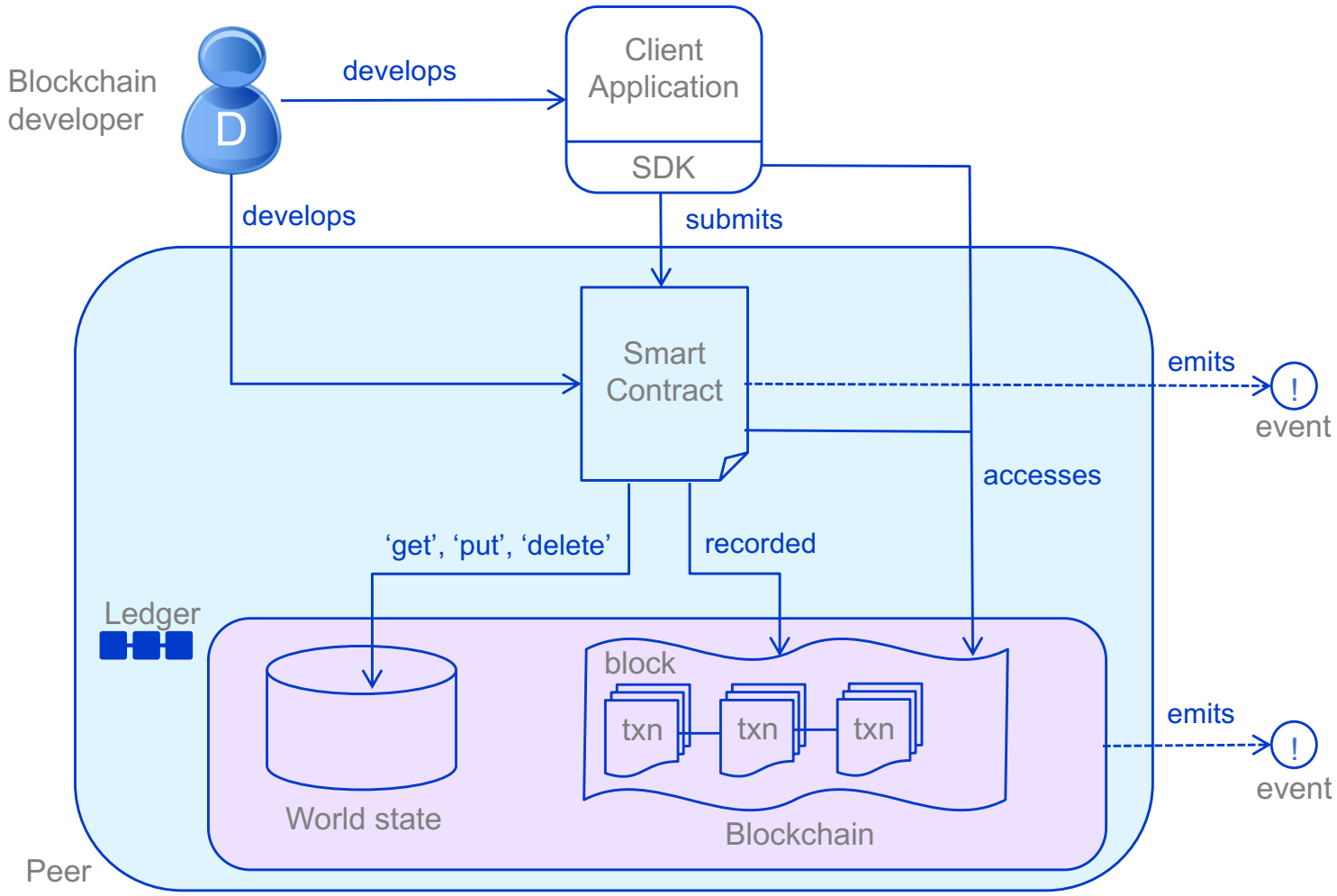
Smart contract implementation

```
setOwner(Car, newOwner) {
   set Car.owner = newOwner
}
```

World state: new contents

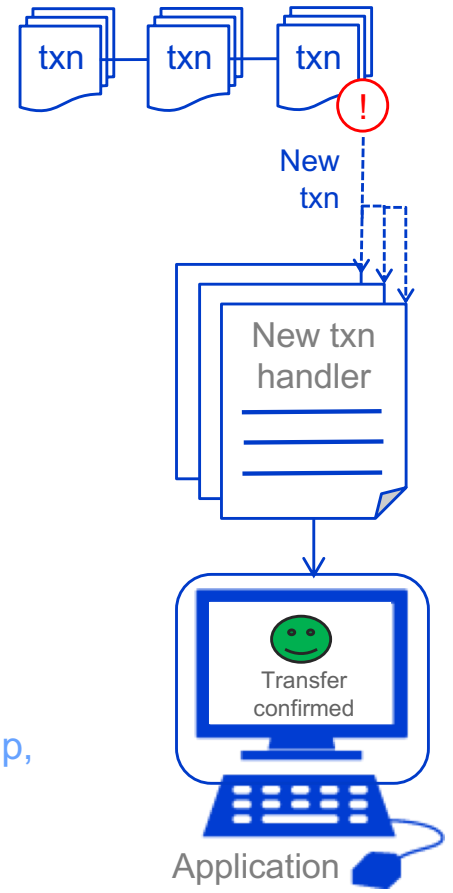
```
myCar.vin = 1234
myCar.owner = Matt
myCar.make = Audi
…
```

How applications interact with the ledger

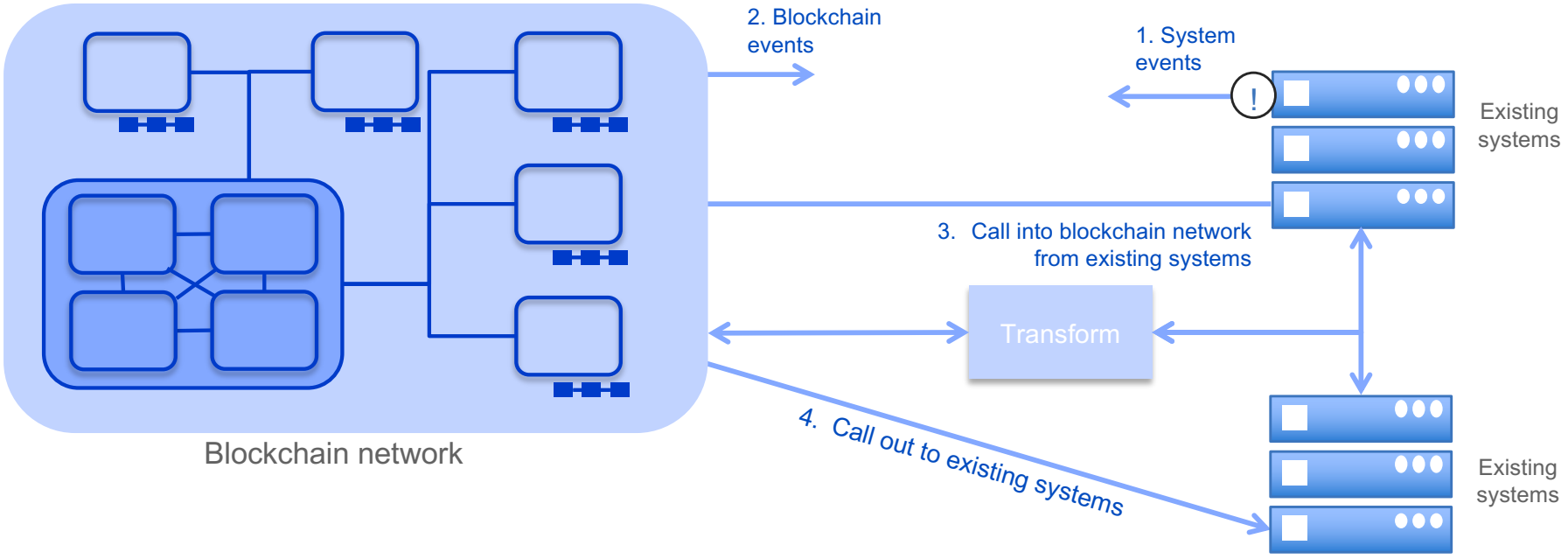


How events are used in blockchain

- In computing, an event is an occurrence that can trigger handlers
  - e.g. disk full, fail transfer completed, mouse clicked, message received, temperature too hot…
- Events are important in asynchronous processing systems like blockchain
- The blockchain can emit events that are useful to application programmers
  - e.g. Transaction has been validated or rejected, block has been added…
- Events from external systems might also trigger blockchain activity
  - e.g. exchange rate has gone below a threshold, the temperature has gone up, a time period has elapsed…

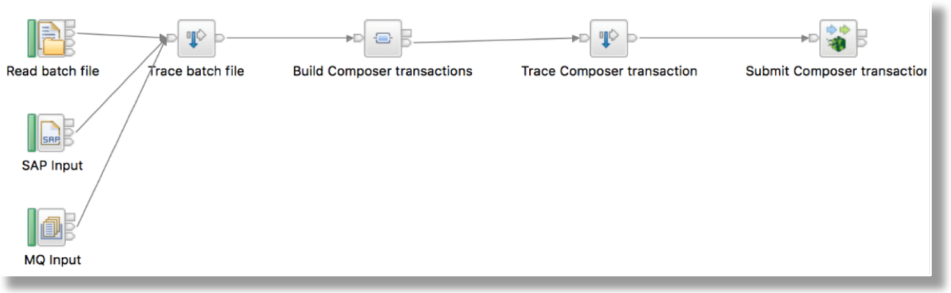


Integrating with existing systems – possibilities



Integrating with existing systems – using middleware

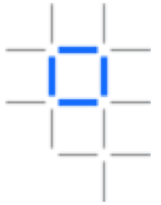
- Blockchain is a network system of record
- Two-way exchange
  - Events from blockchain network create actions in existing systems
  - Cumulative actions in existing systems result in Blockchain interaction



- Transformation between blockchain and existing systems' formats
  - GBO, ASBO is most likely approach
  - Standard approach will be for gateway products to bridge these formats
  - Gateway connects to peer in blockchain network and existing systems

- Smart contracts can call out to existing systems
  - Query is most likely interaction for smart decisions
    - e.g. all payments made before asset transfer?
    - Warning: Take care over predictability… transaction must provide same outputs each time it executes

Non-determinism in blockchain



- Blockchain is a distributed processing system
  - Smart contracts are run multiple times and in multiple places
  - As we will see, smart contracts need to run deterministically in order for consensus to work
    - Particularly when updating the world state
- It's particularly difficult to achieve determinism with off-chain processing
  - Implement services that are guaranteed to be consistent for a given transaction, or
  - Detect duplicates for a transaction in the blockchain, middleware or external system

random()

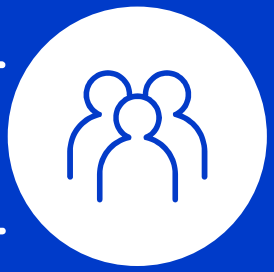
getExchangeRate()

getDateTime()

getTemperature()

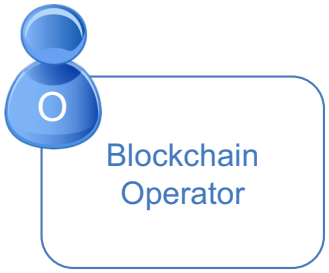
incrementValue inExternalSystem(…)

Concepts and Components

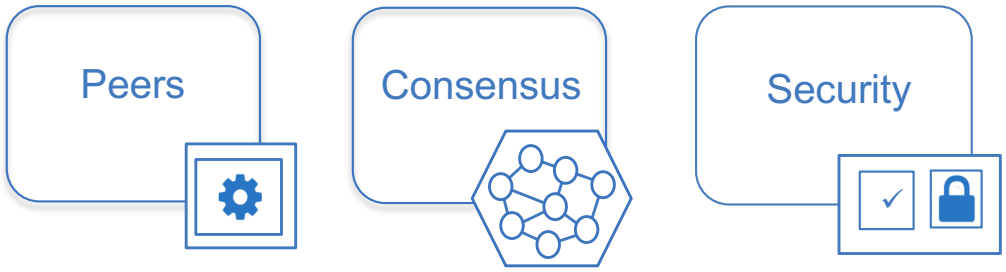


Considerations for the Developer, Operator and Architect

The blockchain operator



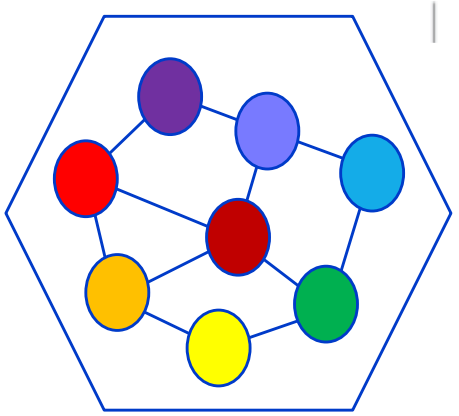
Blockchain operators' primarily interests are in the deployment and operation of part of the blockchain:



They should NOT have to care about development concerns, such as:

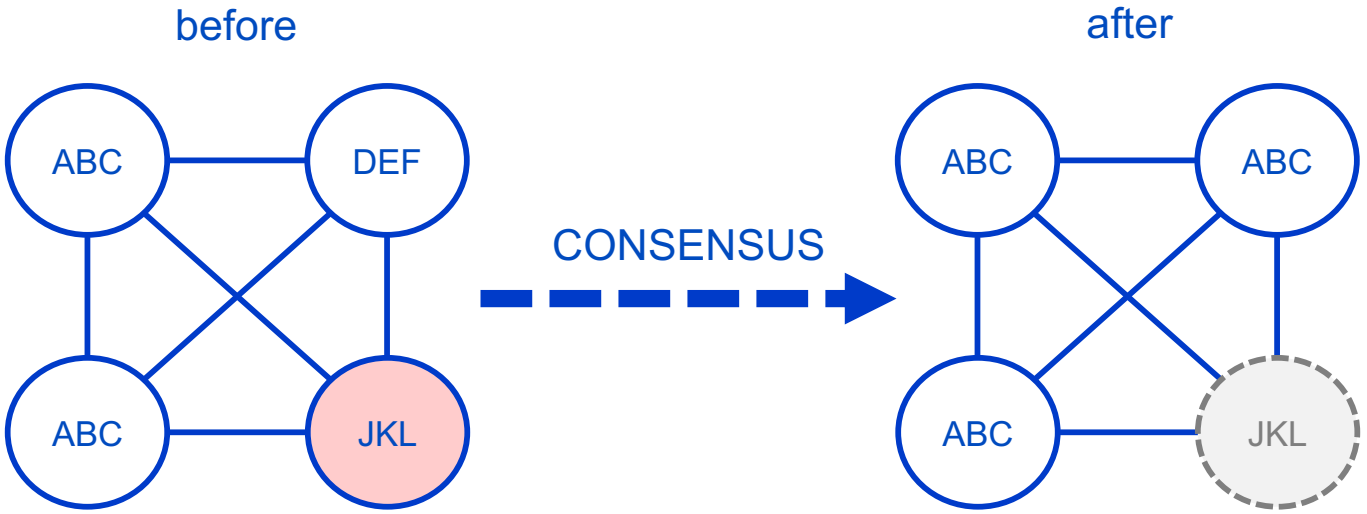
X Application code Smart contract code

Peers



- Peers are the technical services that a blockchain requires in order to work
  - Peers hold and maintain the ledger
  - They receive transactions from applications (and other peers)
  - Peers can validate transactions
  - They notify applications about the outcome of submitted transactions
- Peers are implemented as an operating system process
  - …to which applications and other peers can connect
  - Very similar to web servers!
- Peers connect to other peers to form nodes on a peer-to-peer blockchain network
  - Peers can be run wherever makes sense; allows for heterogeneous technology choices
  - Some blockchains are worldwide, others are private to a business network
  - However, peers from one blockchain implementation cannot talk with others (yet!)
    - For example, an Ethereum blockchain cannot transfer assets to a Hyperledger Fabric blockchain
  - It might make sense to have one peer per business network participant, but this is not necessarily so

Consensus: The process of maintaining a consistent ledger



peer

LEDGE
R

Keep all peers up-to-date
Fix any peers in error
Ignoring all malicious nodes

Some examples of consensus algorithms

Proof of work

Proof of stake

Solo /
No-ops

Kafka /
Zookeeper

Proof of
Elapsed Time

PBFT
based

Consensus algorithms have different strengths and weaknesses

Proof of work

Require validators to solve difficult cryptographic puzzles
PROs: Works in untrusted networks
CONS: Relies on energy use; slow to confirm transactions
Example usage: Bitcoin, Ethereum

Proof of stake

Require validators to hold currency in escrow
PROs: Works in untrusted networks
CONS: Requires intrinsic (crypto)currency, ”Nothing at stake” problem
Example usage: Nxt

Proof of Elapsed Time

Wait time in a trusted execution environment randomizes block generation
PROs: Efficient
CONS: Requires processor extensions
Example usage: Hyperledger Sawtooth

Consensus algorithms have different strengths and weaknesses

Solo / No-ops

Validators apply received transactions without consensus
PROs: Very quick; suited to development
CONS: No consensus; can lead to divergent chains
Example usage: Hyperledger Fabric V1

PBFT-based

Practical Byzantine Fault Tolerance implementations
PROs: Reasonably efficient and tolerant against malicious peers
CONS: Validators are known and totally connected
Example usage: Hyperledger Fabric V0.6

Kafka / Zookeeper

Ordering service distributes blocks to peers
PROs: Efficient and fault tolerant
CONS: Does not guard against malicious activity
Example usage: Hyperledger Fabric V1

Security: Public vs. private blockchains

Public blockchains

- For example, Bitcoin
- Transactions are viewable by anyone
- Participant identity is more difficult to control

Private blockchains

- For example, Hyperledger Fabric
- Network members are known but transactions are secret

- Some use-cases require anonymity, others require privacy
  - Some may require a mixture of the two, depending on the characteristics of each participant

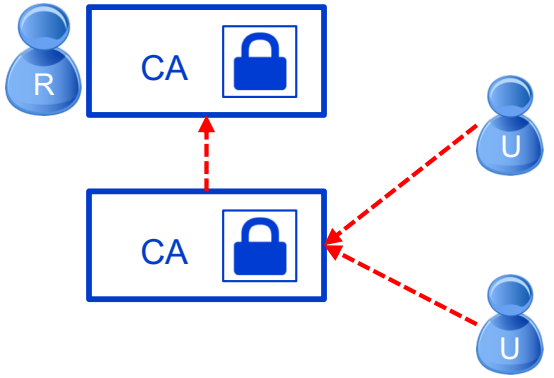
- Most business use-cases require private, permissioned blockchains
  - Network members know who they're dealing with (required for KYC, AML etc.)
  - Transactions are (usually) confidential between the participants concerned
  - Membership is controlled

Security: Real-world vs. digital identity

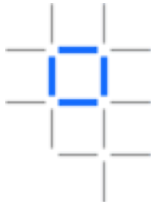
- Consider real-world identity documents…
  - The issuers of the identity documents are trusted third parties (e.g. passport office)
  - There is usually a chain of trust (e.g. to get a bank card you need a drivers license or passport)
  - Identity documents are often stored in wallets

- In the digital world, identities consist of public/private key pairs known as certificates
  - Identity documents are issued by trusted third parties known as Certificate Authorities (CAs)

- Private blockchain networks also require CAs
  - So network members know who they're dealing with
  - May sit with a regulatory body or a trusted subset of participants



Security: Encryption and Signing

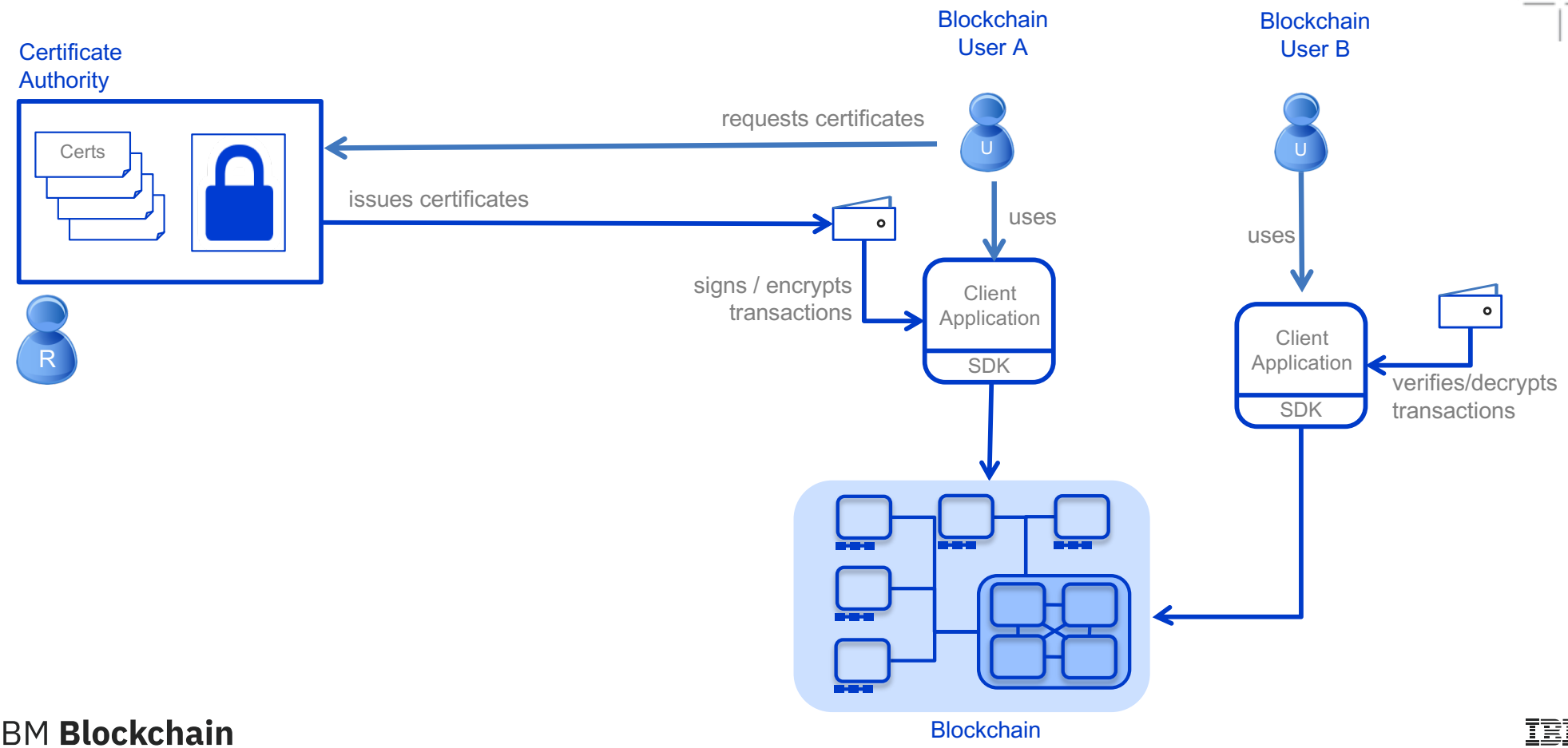


- Cryptography basics
  - Every member of the network has (at least) one public key and one private key
  - Assume that every member of the network knows all public keys and only their own private keys
  - Encryption is the process applying a transformation function to data such that it can only be decrypted by the other key in the public/private key pair
  - Users can sign data with a private key; others can verify that it was signed by that user

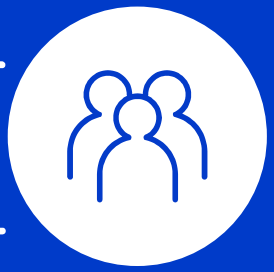
- For example
  - Alice can sign a transaction with her private key such that anyone can verify it came from her
  - Anyone can encrypt a transaction with Bob's public key; only Bob's private key can decrypt it

- In private, permissioned blockchains
  - Transactions and smart contracts can be signed to verify where they originated
  - Transactions and their payloads can be encrypted such that only authorized participants can decrypt

Certificate Authorities and Blockchain

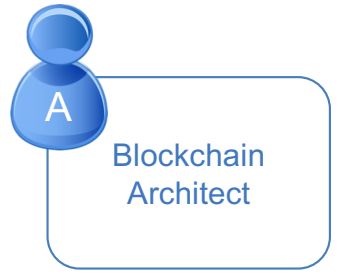


Concepts and Components



Considerations for the Developer, Operator and Architect

The blockchain architect

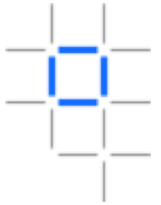


For a successful solution, blockchain architects need a good understanding of many development and operational concerns discussed in this session:

- Applications
- Smart contracts
- Events and Integration
- Peers
- Consensus
- Security

However there are additional considerations for architects to bear in mind from the outset. For example:

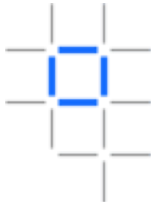
Business considerations



- As a B2B system, blockchain adds a number of aspects that are not typical in other projects:
  - Who pays for the development and operation of the network?
  - Where are the blockchain peers hosted?
  - When and how do new participants join the network?
  - What are the rules of confidentiality in the network?
  - Who is liable for bugs in (for example) shared smart contracts?
  - For private networks, what are the trusted forms of identity?

- Remember that each business network participant may have different requirements (e.g. trust)
  - Evaluate the incentives of potential participants to work out a viable business model
    - Mutual benefit → shared cost (e.g. sharing reference information)
    - Asymmetric benefit → money as leveler (e.g. pay for access to KYC)

Trade-offs between non-functional requirements



Performance

- The amount of data being shared
- Number and location of peers
- Latency and throughput
- Batching characteristics

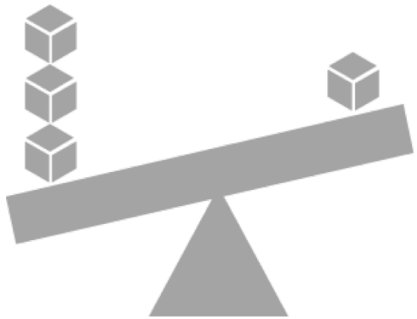
Security

- Type of data being shared, and with whom
- How is identity achieved
- Confidentiality of transaction queries
- Who verifies (endorses) transactions

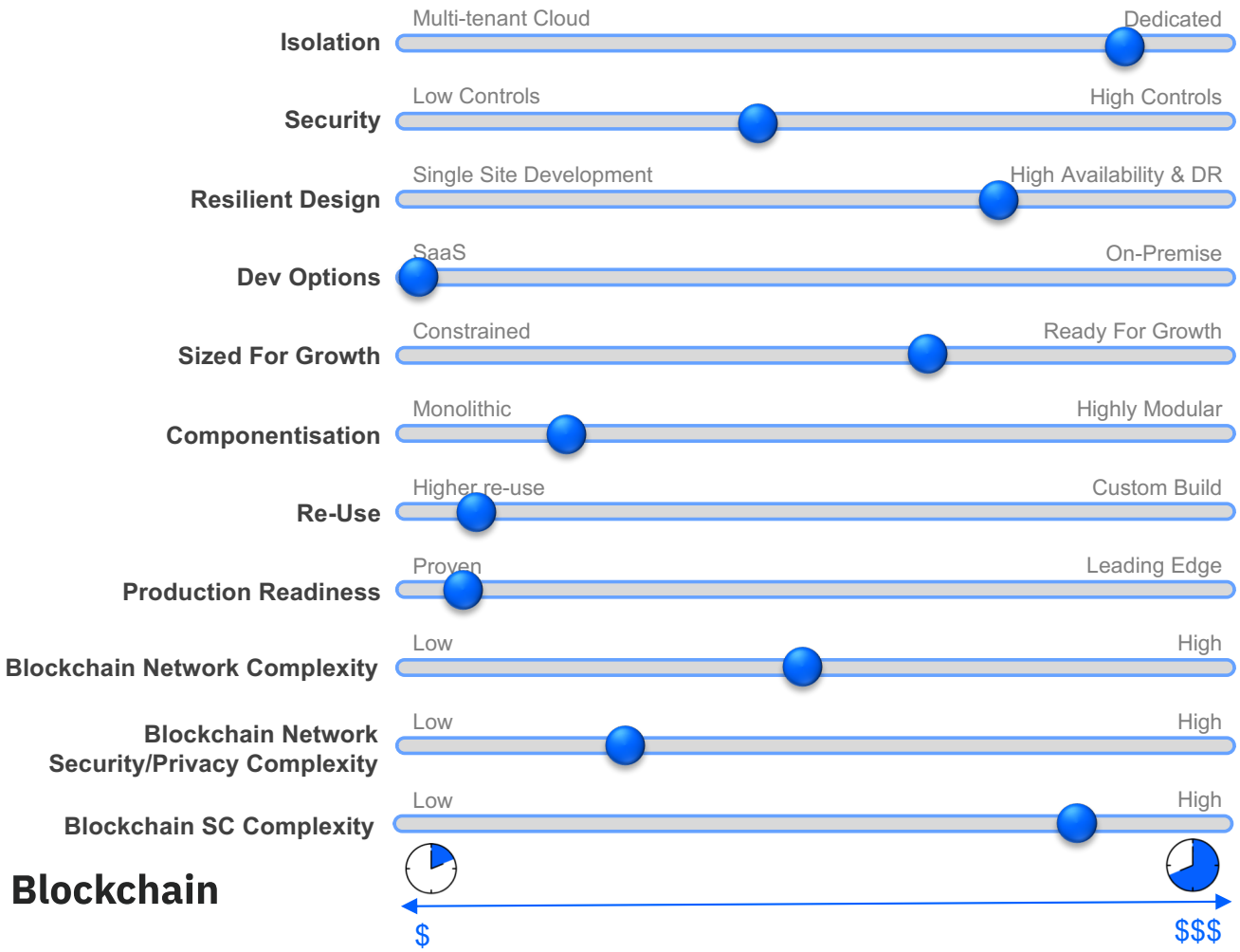
Resiliency

- Resource failure
- Malicious activity
- Non-determinism

Consider the trade-offs between performance, security and resiliency!

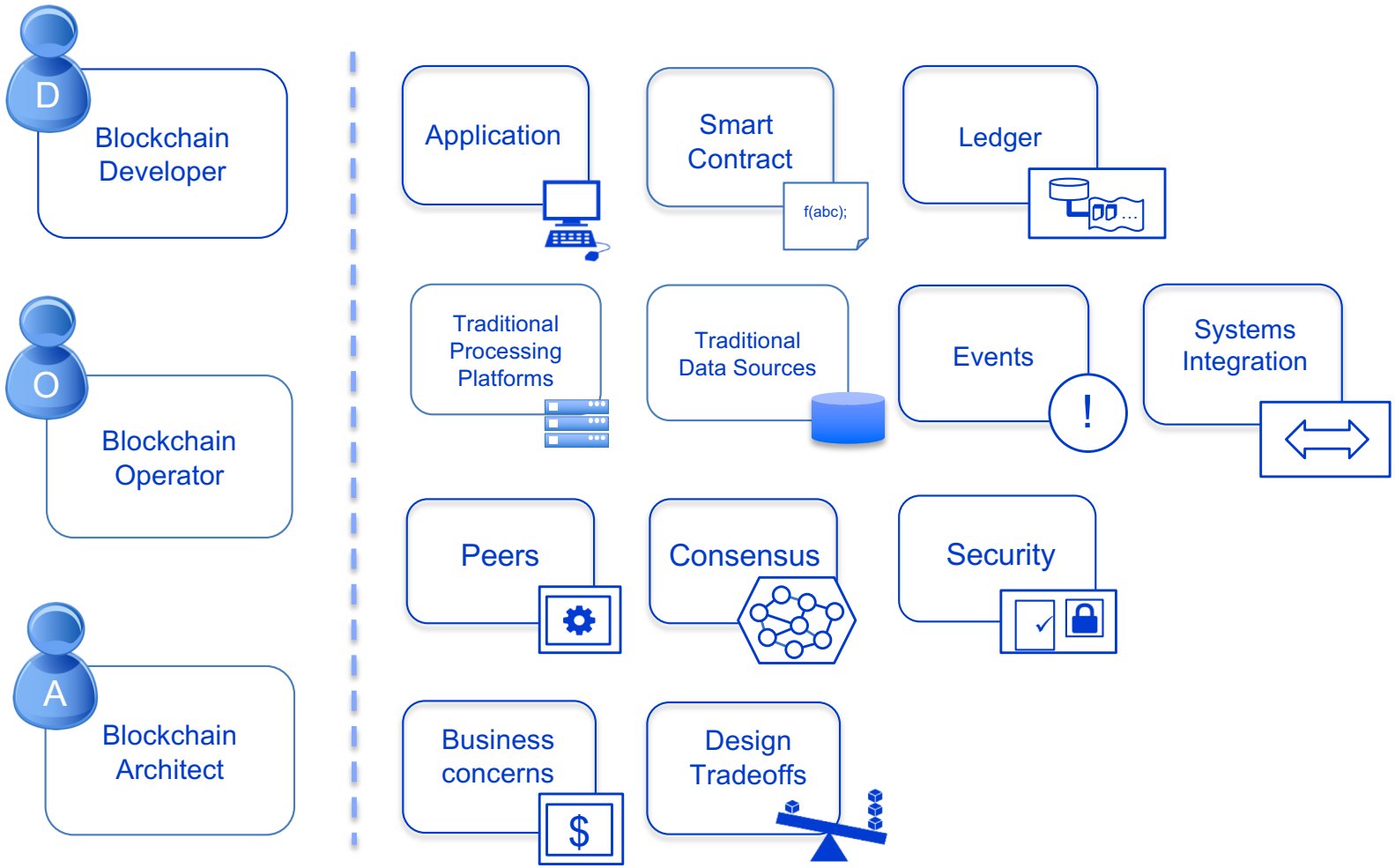


Non-Functional Requirements



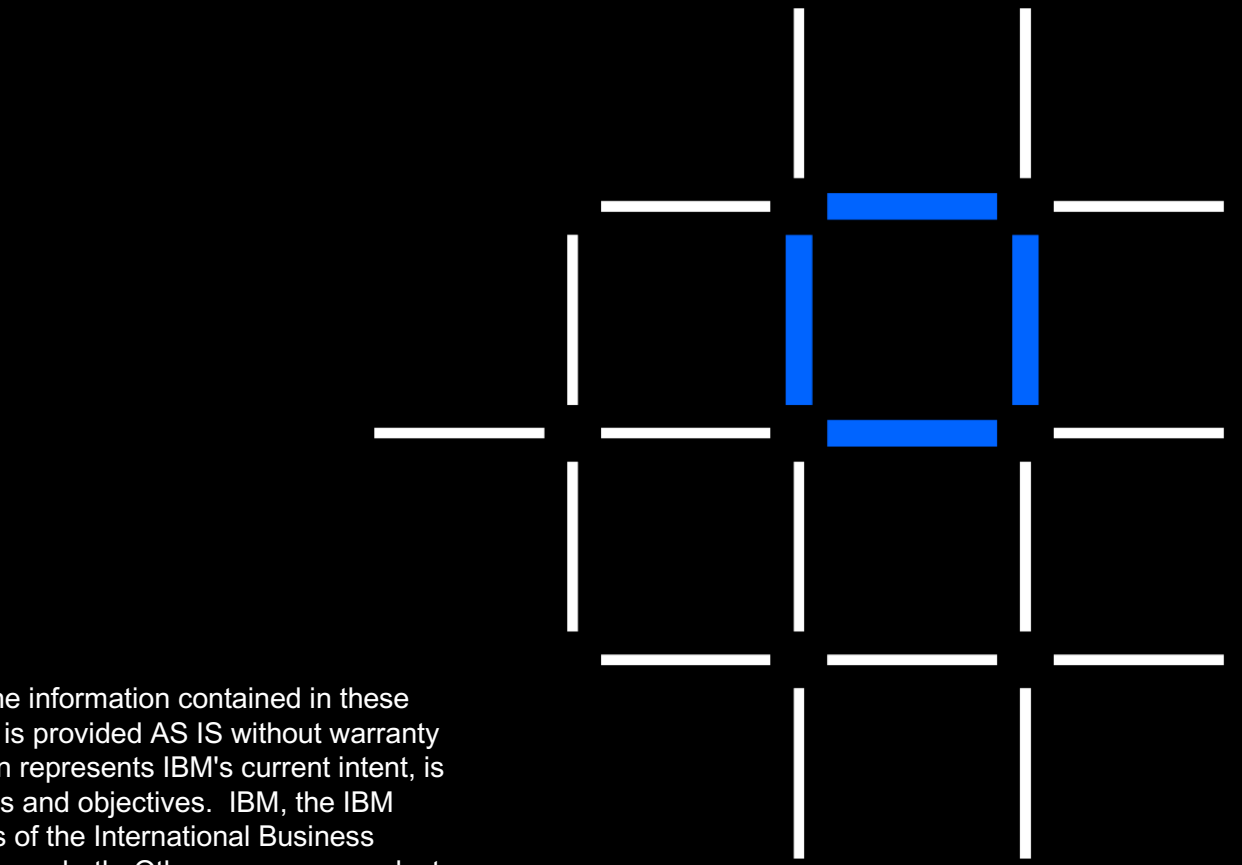
Adjust the sliders with the client early in the project so all parties are aligned on the expectations of robustness, isolation, security controls etc. as all these factors have material impact on the cost and complexity of the solution.

Summary of Key Concepts



Thank you

IBM **Blockchain**



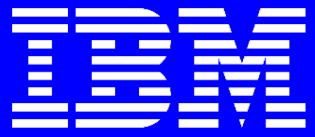
www.ibm.com/blockchain

developer.ibm.com/blockchain

www.hyperledger.org

© Copyright IBM Corporation 2017. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or withdrawal, and represents only goals and objectives. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

IBM


IBM