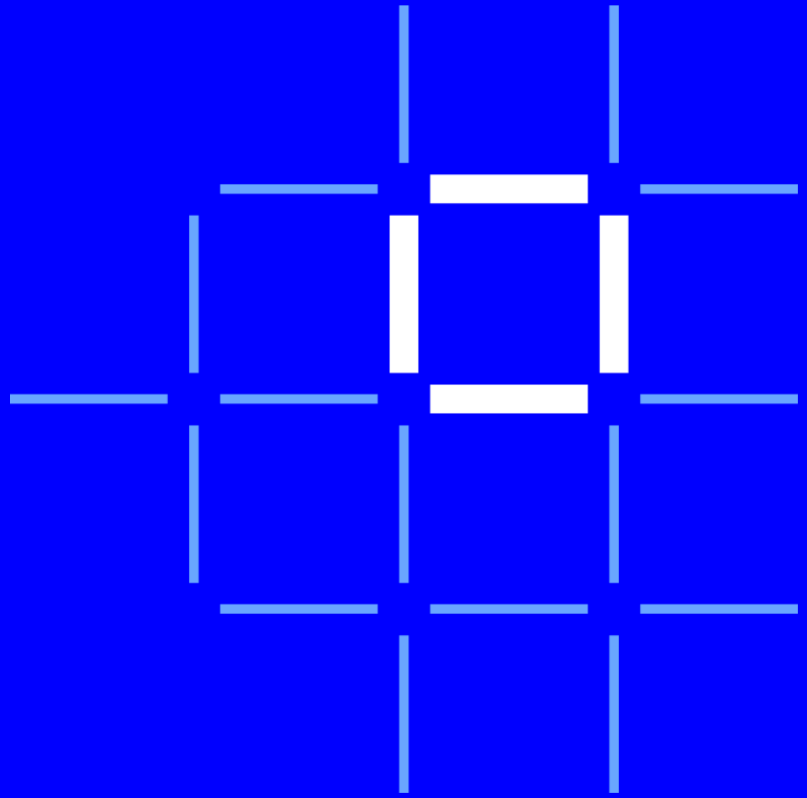
Blockchain for Insurance

March 2018

IBM **Blockchain**

Rachod C.
Software Client Architect

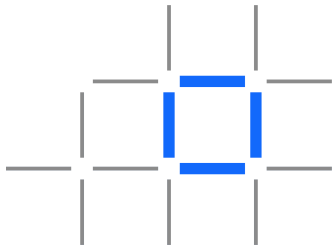
rachod@th.ibm.com



© 2017 IBM Corporation

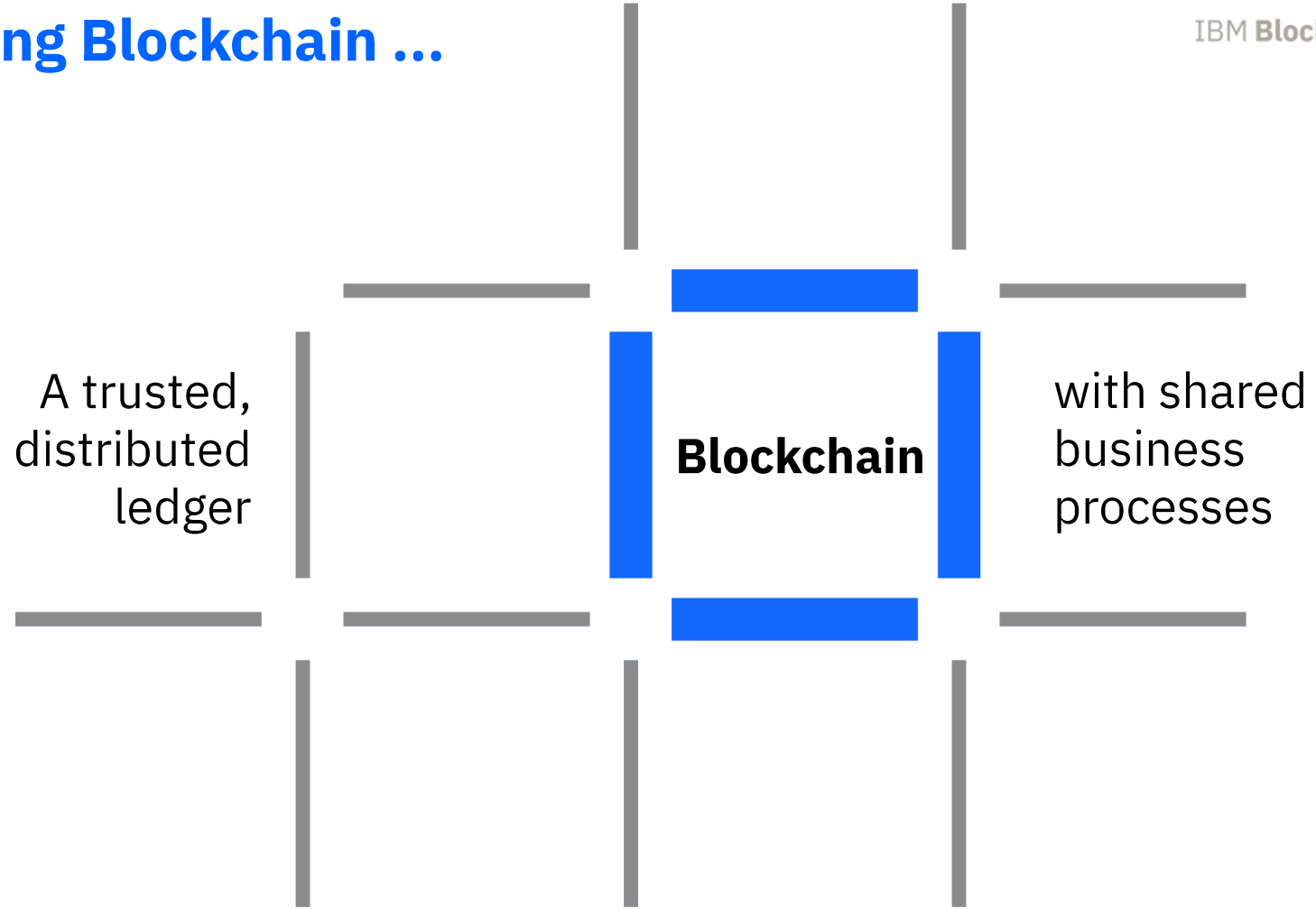
Agenda

- Blockchain overview
- Blockchain in Insurance
- Demo and technical deep dive



© 2017 IBM Corporation

Introducing Blockchain ...

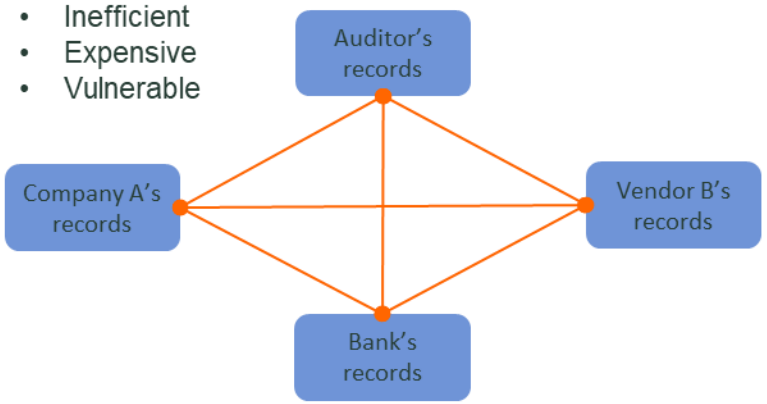


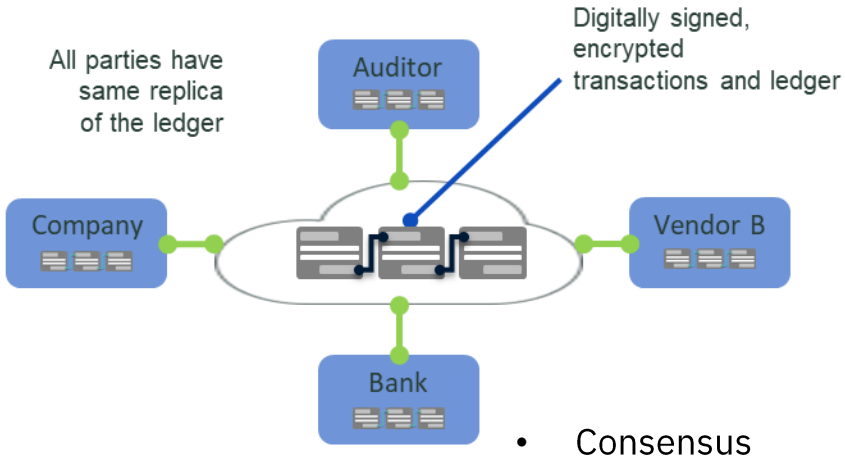
© 2017 IBM Corporation

Blockchain: Fundamentally changing business processes

IBM Blockchain

- Inefficient
- Expensive
- Vulnerable





- Consensus
- Provenance
- Finality
- Immutability

Shared Ledger

Smart Contracts

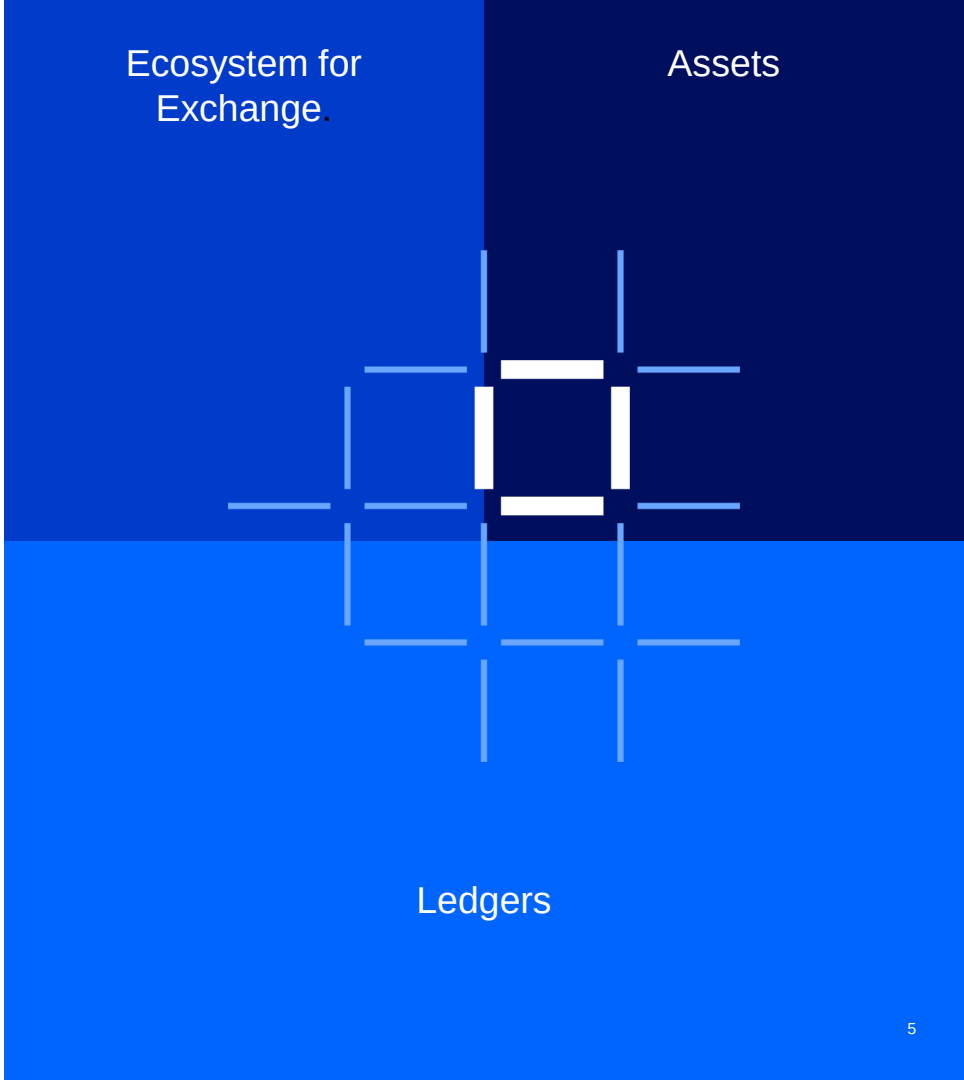
Trust

Privacy

© 2017 IBM Corporation

Why do we need Blockchain?

Blockchain will do for **transactions** what the Internet did for **information**



© 2017 IBM Corporation

What is driving Blockchain?

Create New Value

Exploit new business models and eliminate inefficiencies

Optimize Ecosystems

Streamline business processes and the exchange of value along your ecosystem

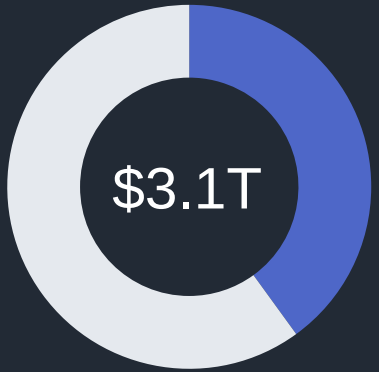
Increase Trust, Reduce Risk

Replace uncertainty with transparency and a trusted decentralized ledger

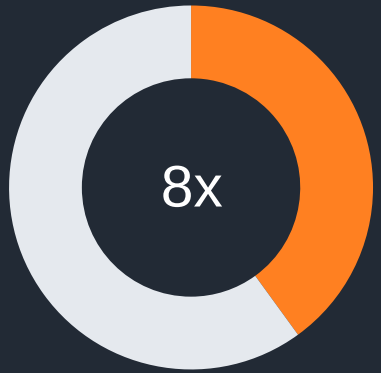
© 2017 IBM Corporation

Market Opportunity

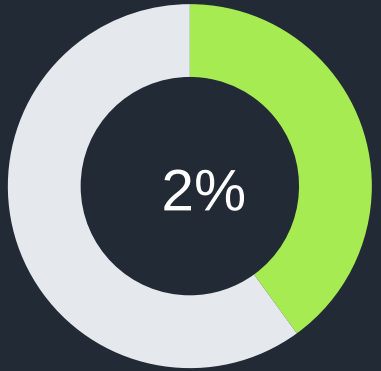
The race is on to build winning business models through digital ecosystems



Blockchain Business value created by 2030.[1]



Network orchestrators (Airbnb, Uber, Netflix, etc.) grow revenue faster with **8x** market value multiplier.[2]



Only about 2% of business uses new digital business models.[2]

1. Gartner (March 2017).
2. The Network Imperative: How to Survive and Grow in the Age of Digital Business Models by Barry Libert (2016)

© 2017 IBM Corporation

Blockchain for Business

bitcoin is an example of an **unpermissioned**, **public ledger**:

- The first blockchain application
- Defines an unregulated shadow-currency
- Resource intensive

- Blockchains for business are generally **permissioned and private**, and prioritize:
  - Identity over anonymity
  - Selective endorsement over proof of work
  - Assets over cryptocurrency

© 2017 IBM Corporation

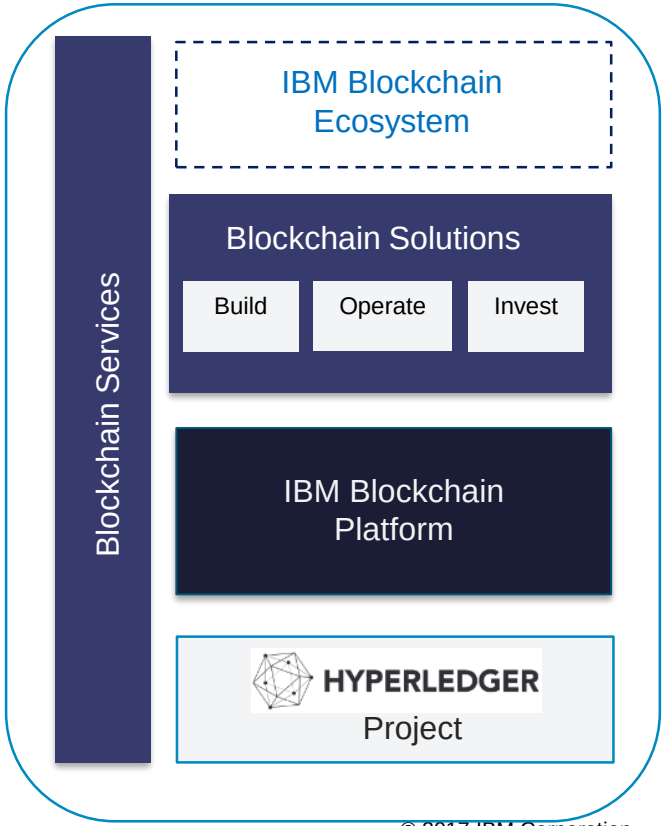
IBM Strategy:
Transforming Industries & Business Processes with Blockchain

Support — Education, support and partner services to create a thriving ecosystem for Blockchain

Provide — Solutions to transform business processes and industries by creating and operating Blockchain Business Networks with enterprise participants

Provide — Secure Blockchain Platform on the Cloud with developer services and production grade capabilities to enable the creation and management of Blockchain business networks

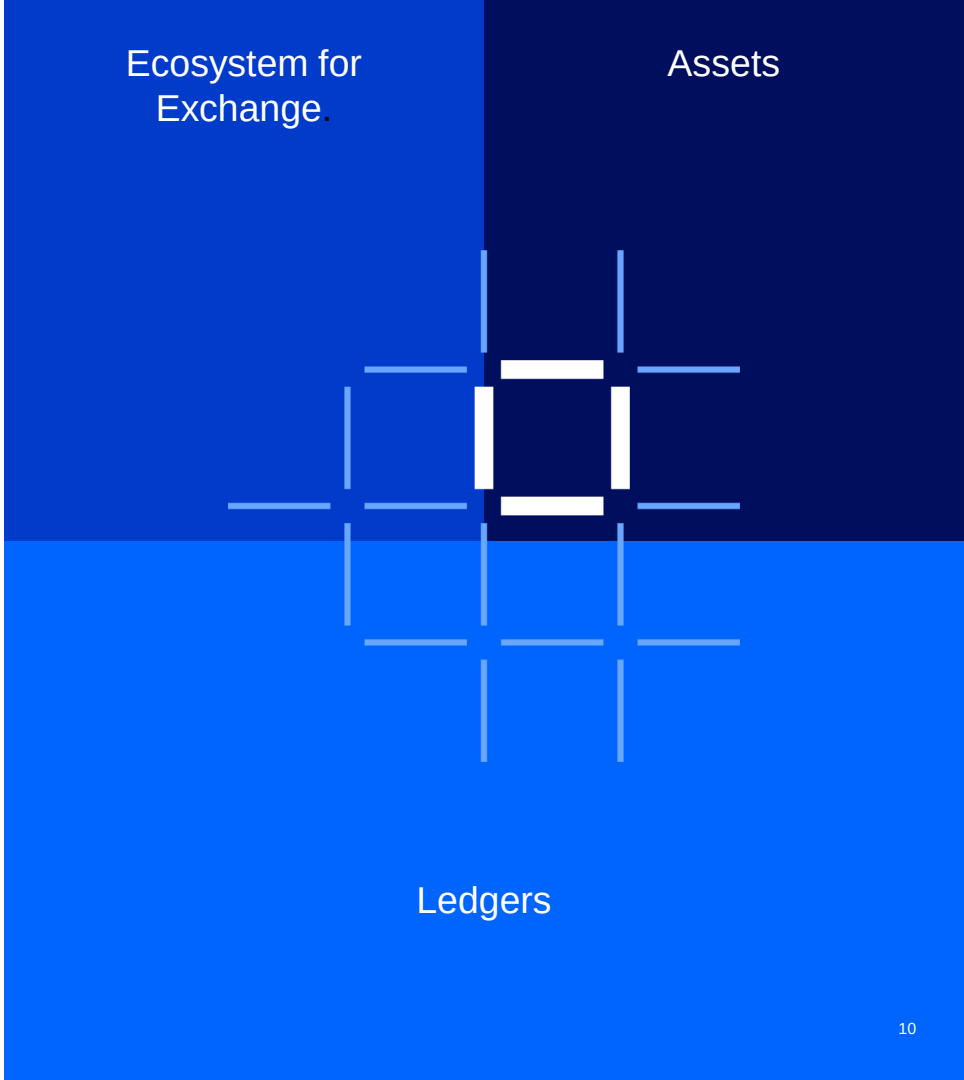
Invest — With The Linux Foundation community to ensure the best possible foundation for Blockchain in a commitment to open standards, open source and open governance



*Hyperledger Fabric is a blockchain framework implementation and Hyperledger Composer is a tooling environment for blockchain and two of the projects hosted by The Linux Foundation.

© 2017 IBM Corporation

Blockchain for Insurance



© 2017 IBM Corporation

Challenges in Insurance Industry

Significant Under-insurance
Across regions and classes of business

The Era of the “Customer”
Customer’s needs have changed

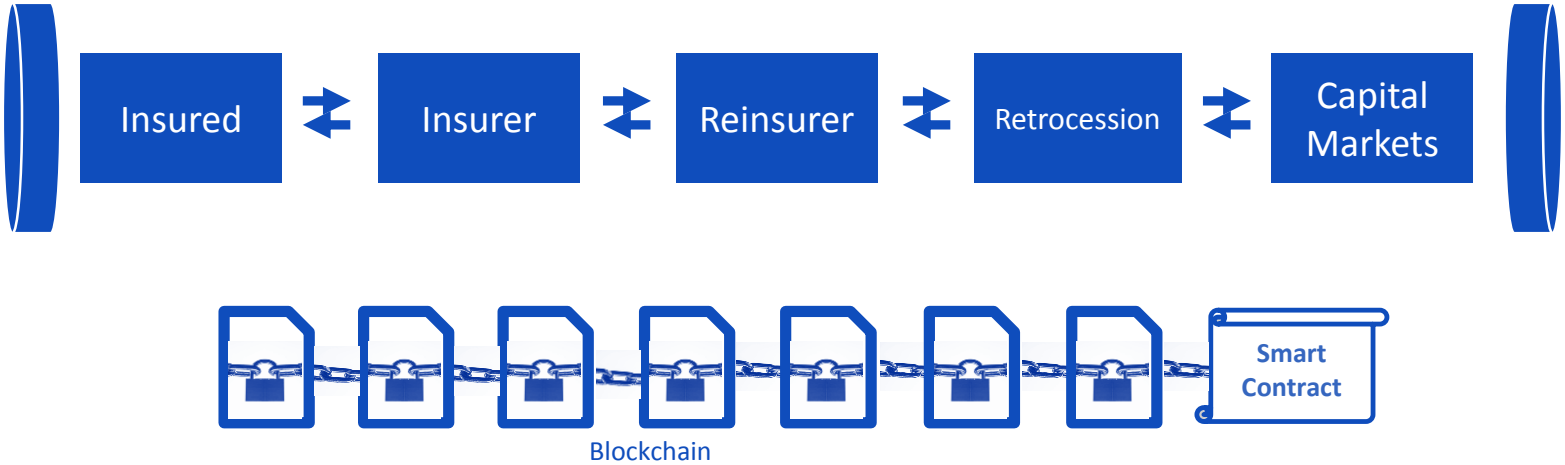
Fraud and Disputes
Multiple versions of truth

Frictionless Risk Transfer
Less frictions make healthier network

© 2017 IBM Corporation

Entire value chain is ripe for automation

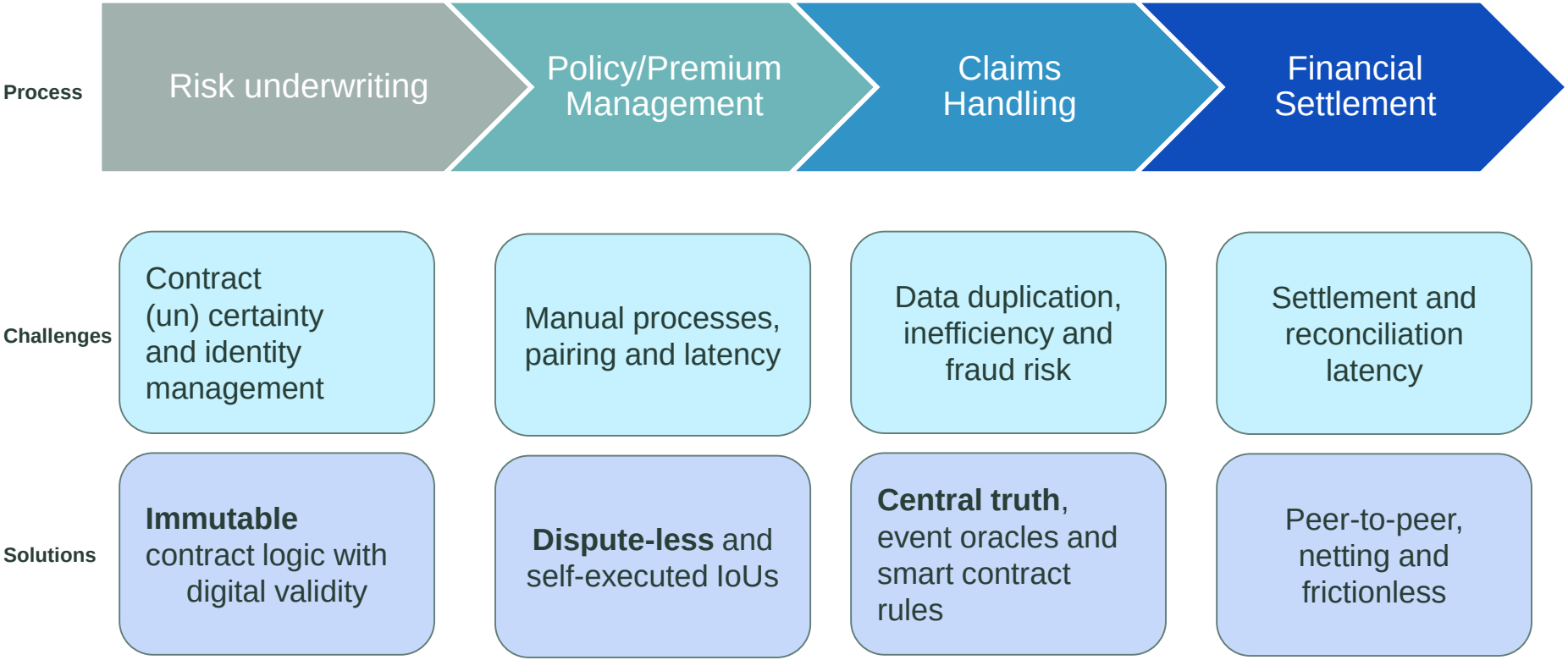
Transaction flow across multiple layers of counterparties within the "Digital Vault" enabled by Smart Contract & Blockchain



© 2017 IBM Corporation

A Closer Root Cause Analysis

Process Friction Points



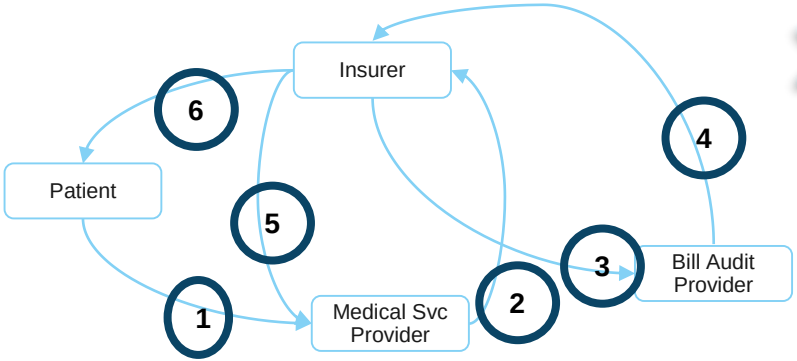
© 2017 IBM Corporation

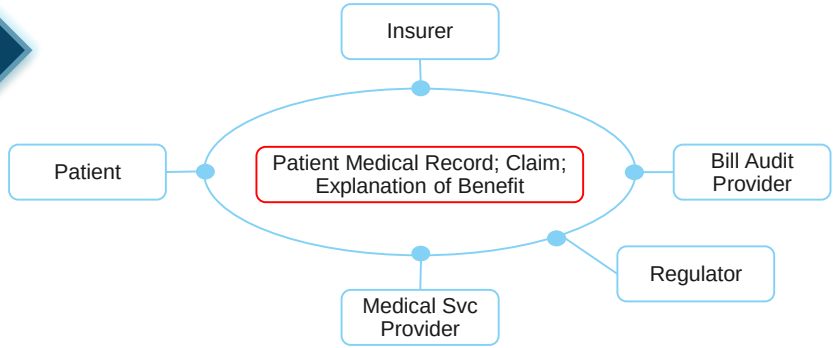
Blockchain in Insurance

- First Party Medical Claims Processing
- Scheduled Personal Property Claims Processing
- Property (Catastrophe) Claims Processing
- Property Re-insurance Contract Administration
- Property Re-insurance Claims Processing
- Surety Insurance
- Parametric Based Insurance

© 2017 IBM Corporation

Use Case 1: First Party Medical Claims Processing Summary



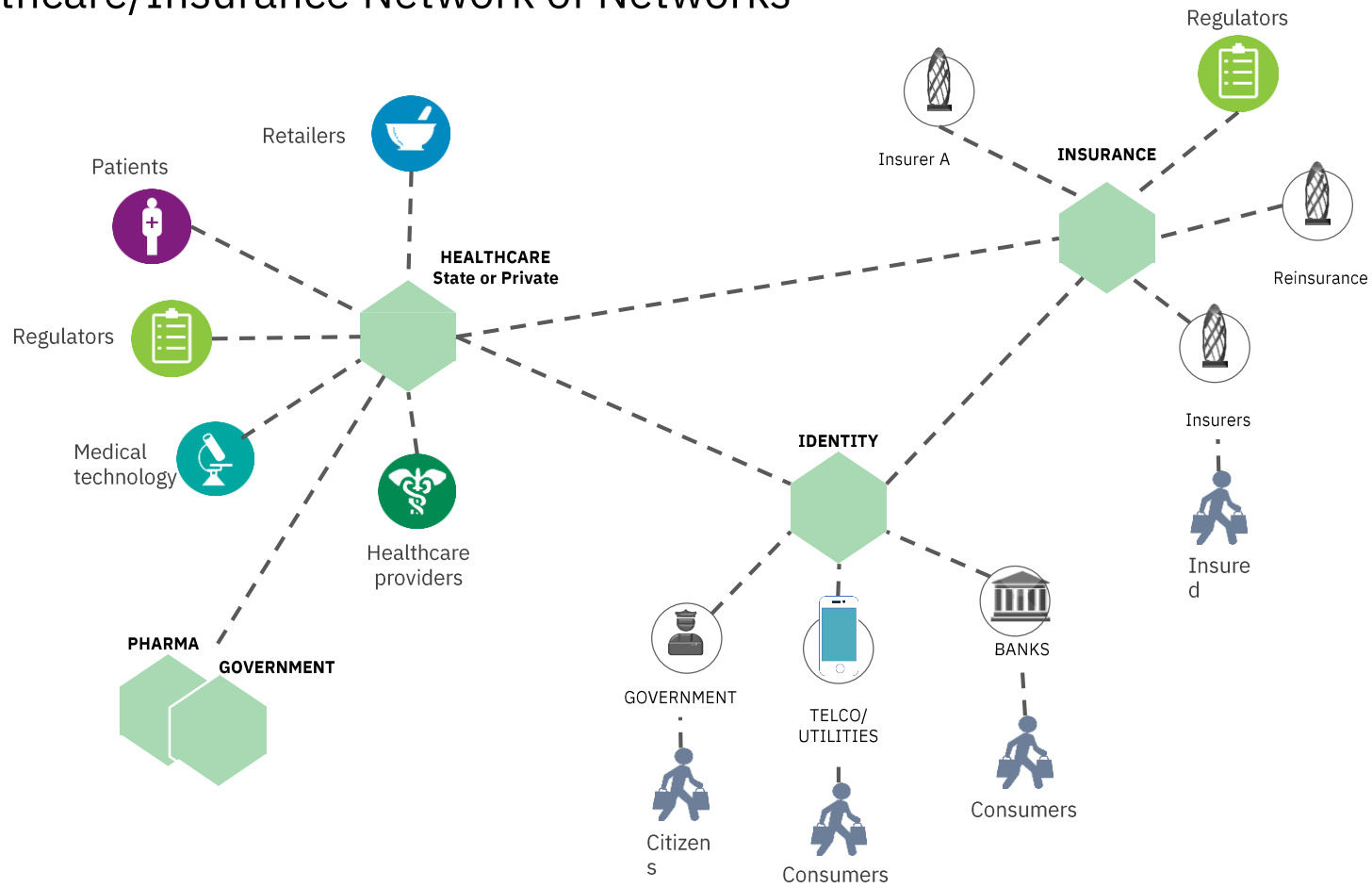


1	Insured seeks injury treatment
2	Medical svc provider registers claim with insurer
3	Insurer requests bill audit, initiates medical records release
4	Bill audit provider supplies bill audit
5	Insurer generates payment to medical svc provider
6	Insurer generates EOB to patient

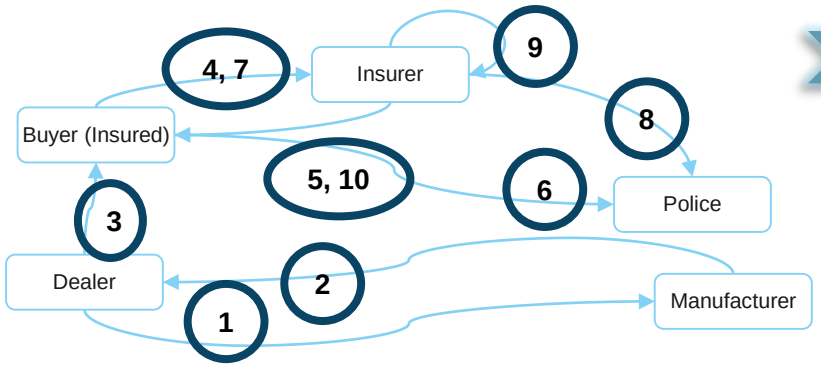
1	Insured seeks injury treatment
2	Claim registered on blockchain. Electronic billing triggers smart contract to initiate bill audit and medical records release
3	Insurer requests bill audit, initiates medical records release
4	Bill audit provider supplies bill audit
5	Insurer generates payment to medical svc provider
6	Insurer generates EOB to patient

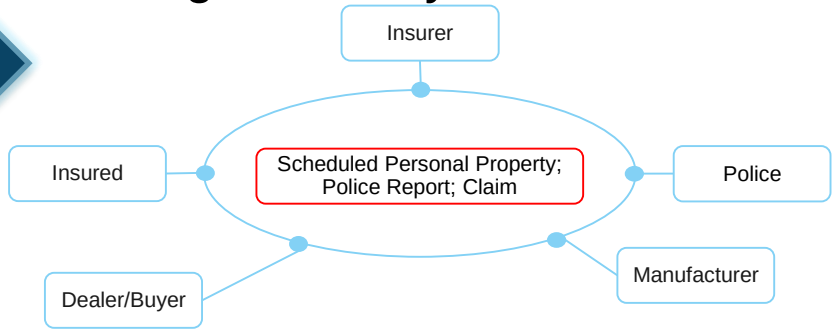
© 2016 IBM Corporation

Healthcare/Insurance Network of Networks



Use Case 2: Personal Property Claims Processing Summary



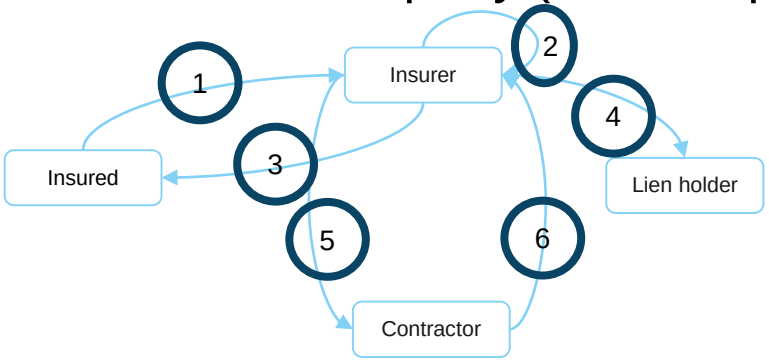


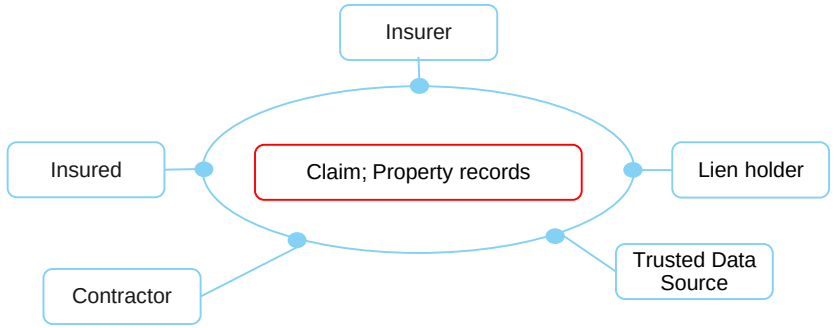
1	Dealer provides custom guitar specification
2	Manufacturer ships to dealer warehouse
3	Buyer purchases guitar online
4	Buyer obtains quote from insurer
5	Insurer adds scheduled item to homeowner policy
6	After 3 policy periods, guitar is stolen and police report filed
7	Buyer files claim
8	Insurer verifies police report
9	Insurer determines payout or replacement
10	Insurer communicates claim determination to insured

1	Serial number, description, photos, etc registered on blockchain when manufacturer builds guitar
2	Shipping info to dealer warehouse added to blockchain
3	Warehouse inventory control info added to blockchain
4	Electronic payment information is added when guitar is purchased
5	Smart contract initiates quote preparation based on market price to add scheduled item to homeowner policy
6	Quote accepted and new premium payment info recorded
7	In each policy period, current value of guitar recorded
8	Police report registered by insured on blockchain
9	Claim registered on blockchain, smart contract validates claim using police report evidence
10	Value determined by limit or tracked value; Payment or replacement provided; Guitar tracking information updated

© 2016 IBM Corporation

Use Case 3: Property (Catastrophe) Claims Processing Summary

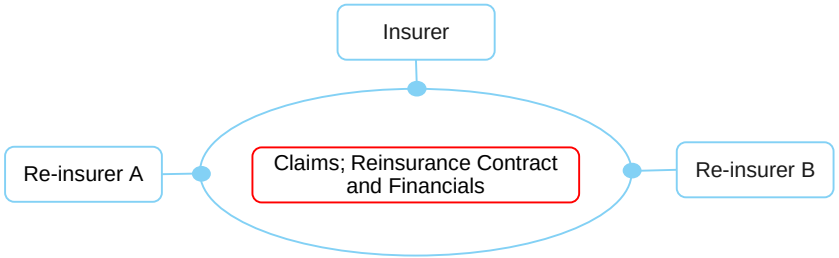




1	Insured discovers roof leaks after large hail storm and initiates claim for roof damage
2	Insurer uses policy and external data to evaluate roof area and develop estimate
3	Insurer provides claim determination of benefits and insured decides to proceed
4	Insurer notifies policy lien holder
5	Insurer exchanges claim resolution documentation with selected contractor and work proceeds to completion.
6	Contractor invoices upon completion of work and is paid by insurer using check as instrument requiring lien holder endorsement

1	Trusted data source provides weather data that can trigger smart contract to open a draft claim registration for roof damage automatically
2	Property records such as eagle view data/county auditor data/remote sensing images can be used to develop estimate to repair/replace and made available with the claim draft
3	Insured is notified within minutes of catastrophe of existence of claim draft
4	Lien holder provided visibility to claim if insured proceeds with the claim
5	Selected contractor has access to all claim detail including estimate of repair, makes challenges, provides validation and completes work with all transactions recorded on blockchain
6	Invoicing and payments recorded on blockchain upon validation of work by insured and lien holder

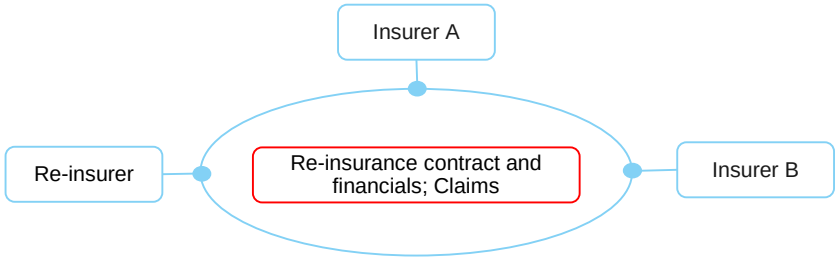
Use Case 4: Property Re-insurance Contract Administration Summary



1	Primary insurer insures property owner and desires to reinsure unwanted exposure (limits or coverages) with facultative reinsurer
2	Reinsurance is obtained through either the direct and/or broker markets and reinsurance contract is completed
3	Initial and subsequent payments are made during policy period
4	Premium audit triggers adjustment of premium and reinsurance with appropriate adjustment for reinsurance premiums and claims due / payable

© 2016 IBM Corporation

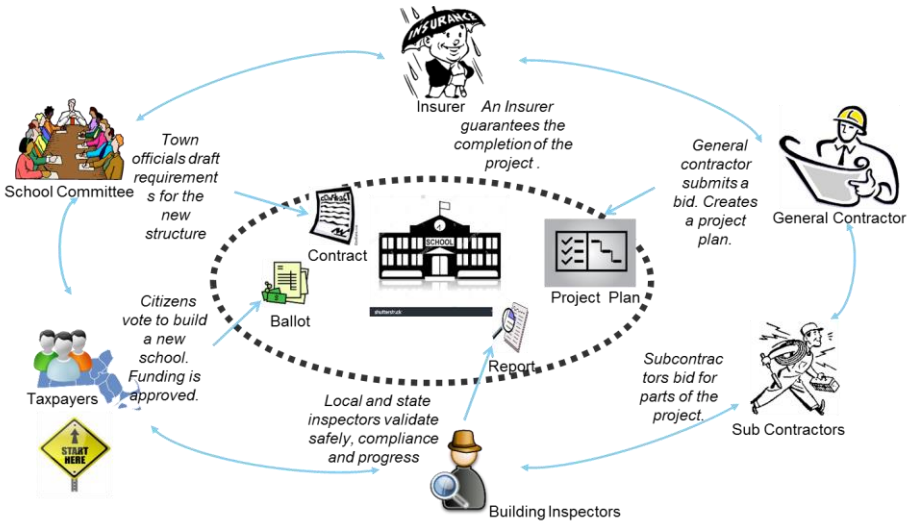
Use Case 5: Property Re-insurance Claims Processing Summary



1	Contract(s) for reinsurer is established with stated terms and conditions
2	Claims are reported and assignment to reinsurers is made based on the pre-established rules
3	Initial and periodic loss reports (paid, reserved, and covered loss adjustment expenses) are sent to reinsurers as agreed upon (e.g. death claims). Adjuster, legal and forensic information can be available
4	Claims payments, whether by insured or insurer or reinsurer that may impact premium payments, premiums and premium adjustments are created
5	Reserves or payments impacting the reinsurer trigger payment or collateral review initiated by one or more parties

© 2016 IBM Corporation

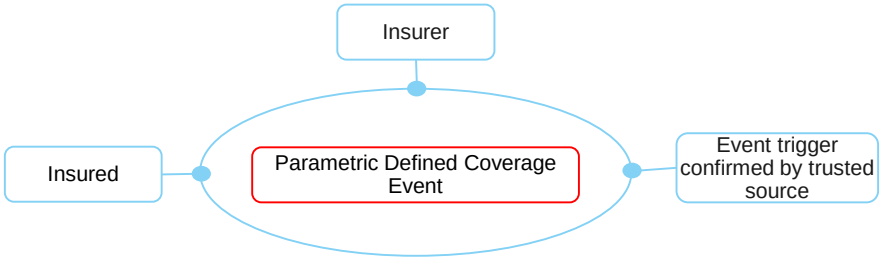
Use Case 6: Surety Insurance Summary

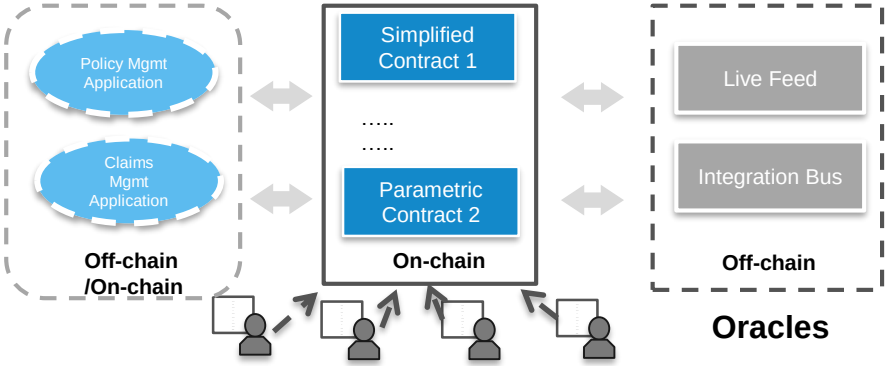


1	Today, Surety is a specialized form of insurance. There tend to be a small number of transactions compared to other insurance products.
2	There also tends to be a very high risk for each transaction. With Surety insurance, it is critical that the insurer make the right assessments. One error can be significant enough to be noted on the annual report.
3	Performance bonds can be extremely complicated and take months to define.
4	Use of a blockchain could streamline underwriting of performance bonds, reduce the cost of administering the bond, and improve outcomes.
5	Surety would be beneficial to all the participants in the blockchain making it a likely candidate for adoption.
6	There are several assets that need to be registered and managed in the blockchain. Some of those assets should be shared with all parties. Some of the data may need to be protected and only shared with a few.

© 2016 IBM Corporation

Use Case 7: Parametric Based Insurance



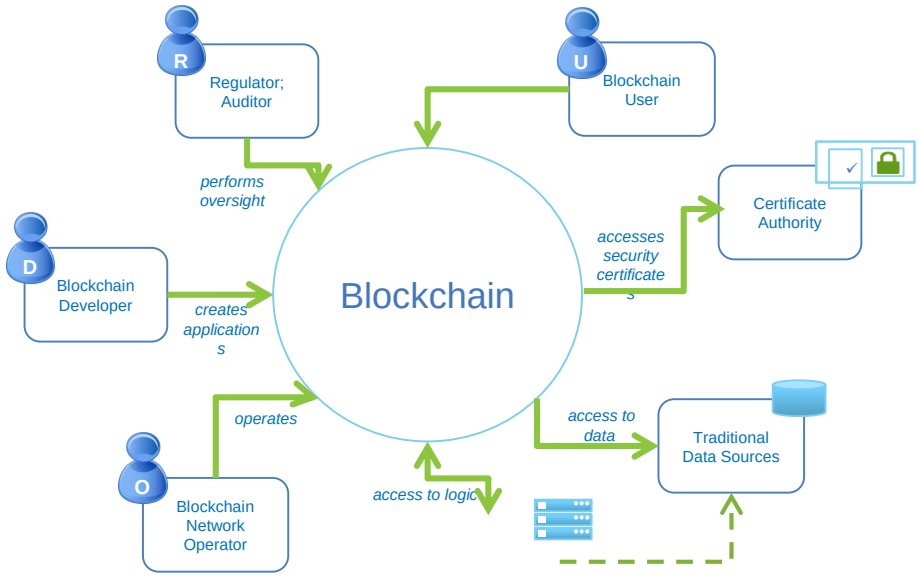


1	All clauses in the parametric contract can be encoded using smart contract language
2	Blockchain keeps track of all claims. Smart contracts keep track of experience account and its maintenance
3	Insurer payouts and account adjustments are automatically performed based on event trigger confirmation by trusted source

1	Parametric contracts are not related to actual loss, but to actual event occurrences or to certain index thresholds.
2	Smart contract, combined with event triggers from off-chain oracles (trusted and secure sources of information) can be used to partially or fully automate operation of these policies (eg. CAT swaps setup and payout at Allianz)
3	Off-chain oracles (such as IoT device events) are used to capture usage related to shared automobiles, homes, office spaces, etc and offer on-demand usage related policies. This can give rise to a whole new class of on-demand insurance products.

© 2016 IBM Corporation

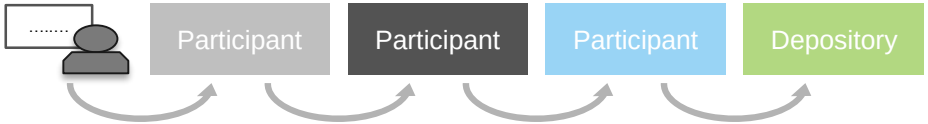
Use Case 8: Financial Audit and Reporting



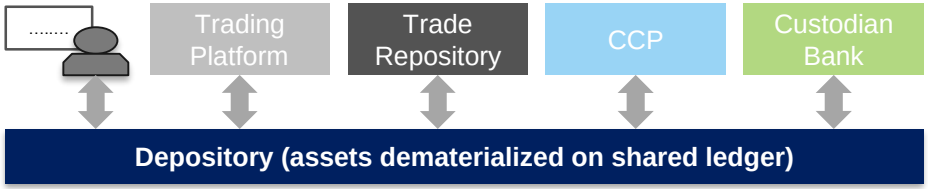
1	Financial audit and reporting is a side-benefit of holding shared assets such as contracts, claim detail, asset, liability and investment accounts within the insurance business network
2	By adding auditors as participants on an existing blockchain network, detailed reports and access to financial data can automatically be provided on a permissioned basis
3	Auditors have prescribed access to defined accounts and transactions with the ability to trace details of specifically targeted transactions in order to meet audit standards

© 2016 IBM Corporation

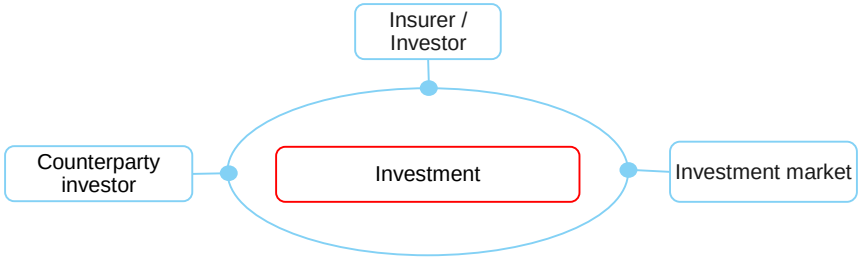
Use Case 9: Market Investments



Centralized Repository (today's system)



Shared Ledger



1	Securities and associated transactions are placed on blockchain and the approved counterparties provided access
2	Securities are dematerialized on a blockchain network so that all approved parties have direct access to that asset, allowing them to initiate trades and acquire information on an asset without going through layers of intermediaries.
3	Trades settled in near real time and all stakeholders able to access asset information in near real time. Stakeholder able to add business rules on any given asset type, as one example of using automation logic to further reduce operating costs.
4	Stakeholder able to add business rules on any given asset type, as one example of using automation logic to further reduce operating costs.

© 2016 IBM Corporation

Insurers are already using Blockchain technology within their operations

Manulife

Collaborating with ConseSys and Blockapps to develop multiple blockchain PoCs designed to show how distributed ledger technology could reinvent the established insurance industry.

Allianz

Partnered with Everledger to explore blockchain technology. Used blockchain based smart contracts to automate catastrophe swaps and bonds, improving transaction processing and accelerating settlement of funds between insurers and investors

LLOYD'S

Highlighted blockchain to insurance market participants as part of their modernization plan called Target Operating Model (TOM).

consuelo

Micro insurance company that allows migrant workers in the U.S. pay for health/life insurance for family in Mexico, using blockchain technology for fraud protection.

© 2016 IBM Corporation

Blockchain

Helping AIG Innovate on its multinational insurance programs with blockchain

"There is tremendous opportunity to apply advancements in blockchain technology to transform the insurance industry. By creatively leveraging smart contracts to help address tough regulatory requirements across different markets, we are seeing the enormous impact blockchain can have to improve efficiency and open up new business models."

Tweet this

The blockchain solution creates a new level of trust and transparency in the underwriting process, enabling AIG and Standard Chartered to execute multinational coverage more efficiently. Coordinating management and placement of multiple insurance policies across multiple countries is highly complex. The pilot solution was built by IBM and is based on Hyperledger Fabric – a blockchain framework and one of the Hyperledger projects hosted by The Linux Foundation.

Working together, AIG, Standard Chartered and IBM converted a multinational, controlled master policy written in the UK, and three local policies in the US, Singapore and Kenya, into a "smart contract" that provides a shared view of policy data and documentation in real-time. This also allows visibility into coverage and premium payment at the local and master level as well as automated notifications to network participants following payment events. The pilot also demonstrates the ability to include third parties in the network, such as brokers, auditors and other stakeholders, giving them a customized view of policy and payment data and documentation.

We're now bringing that same level of tracking and visibility to the insurance and reinsurance industries. Reinsurance serves, in part, to distribute risk across many insurance companies so that no one company is made insolvent by a claim. Forming these reinsurance contracts requires the participants to negotiate contracts, track contract amendments and agree on contract finality in a mutually transparent but secure manner.

Blockchain technology is a natural fit, providing these capabilities and enabling efficient digitization. Once the contracts have been formed and the risk sharing is in effect, blockchain smart contracts enable the calculation of intensive transactions that govern the flow of premium and claim monies between contract parties.

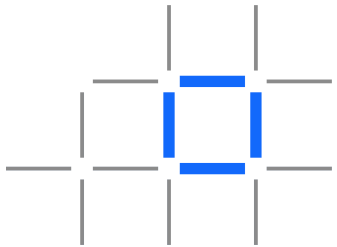
To best take advantage of this new technology, the Blockchain Insurance Industry Initiative (B3i) was formed in 2016 to jointly explore how blockchain could remove friction and lower costs across the insurance value chain. IBM is the technical partner for the initiative, which now counts 15 insurance and reinsurance member companies, and the Hyperledger Fabric is the core technology. After less than one year of starting the initiative, earlier this week the consortium announced the milestone of a market-ready prototype at 'Rendez-Vous de Septembre' in Monte Carlo.

Demo

© 2017 IBM Corporation

Blockchain Insurance Application

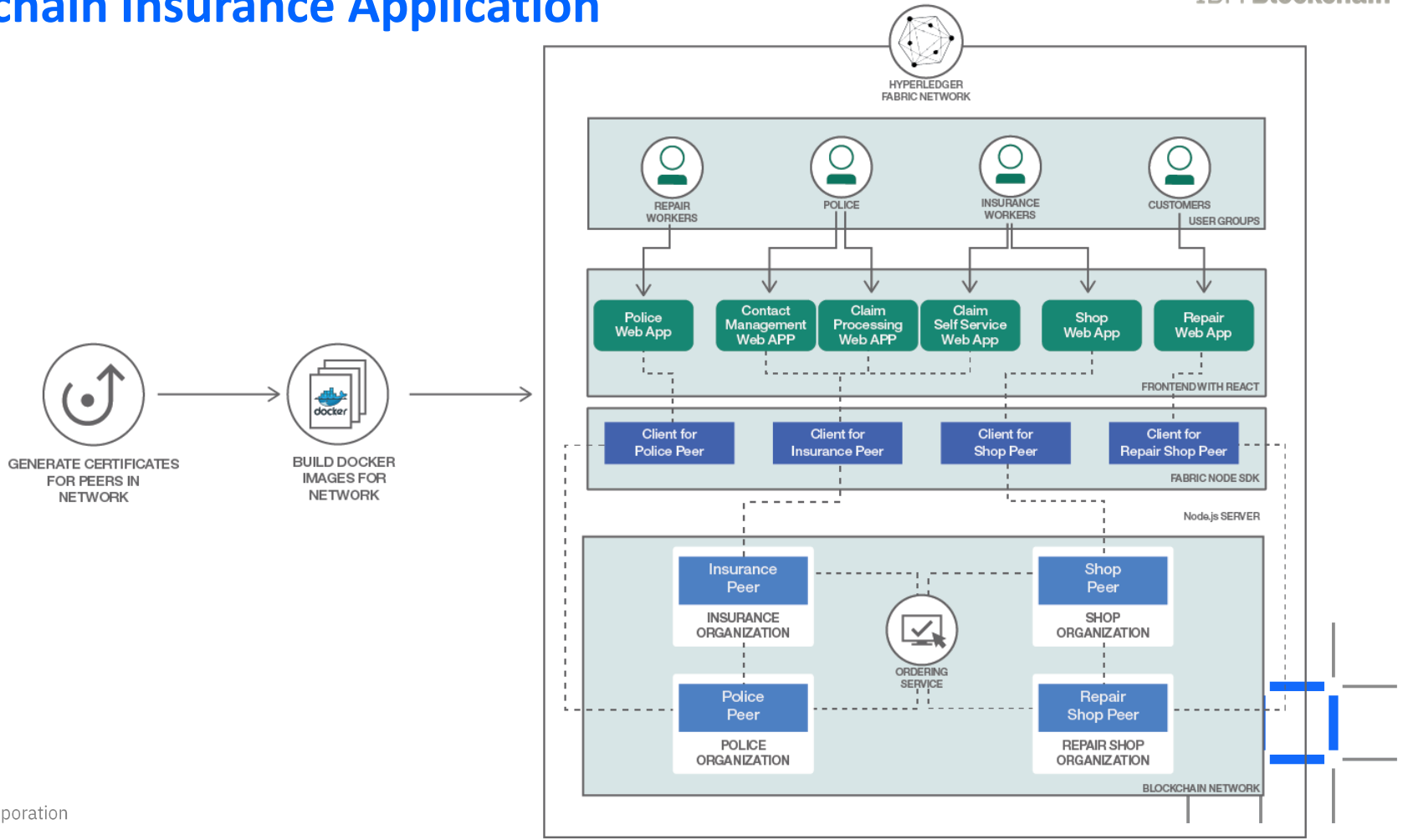
- This project showcases the use of blockchain in insurance domain for claim processing. In this application, we have four participants, namely insurance, police, repair shop and shop peer.
- **Insurance peer** is the insurance company providing the insurance for the products and it is responsible for processing the claims.
- **Police peer** is responsible for verifying the theft claims.
- **Repair shop** peer is responsible for repairs of the product while shop peer sells the products to consumer.



© 2017 IBM Corporation

Blockchain Insurance Application

IBM Blockchain



© 2017 IBM Corporation

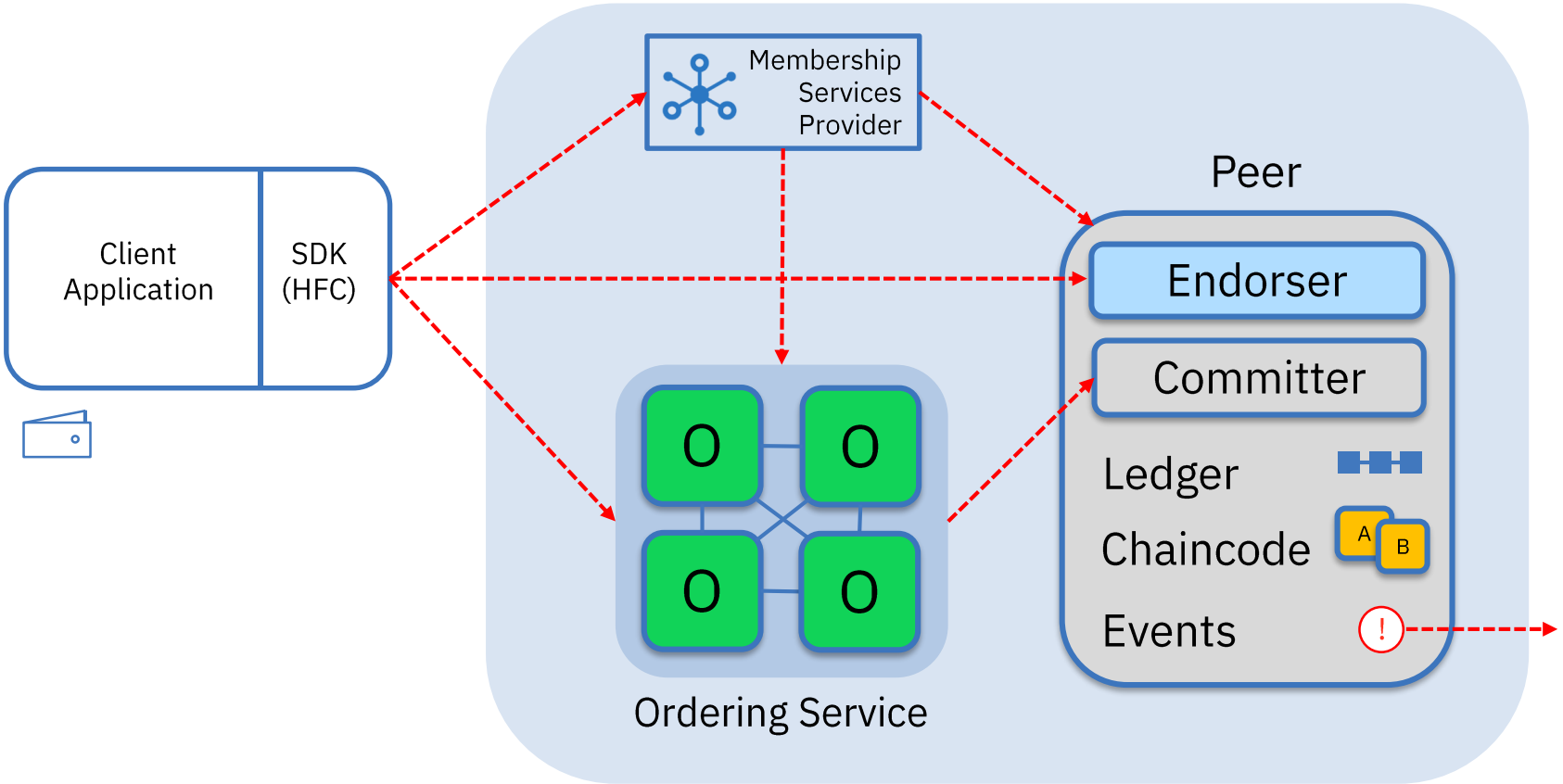
Technical Deep dive

© 2017 IBM Corporation

What is Hyperledger Fabric

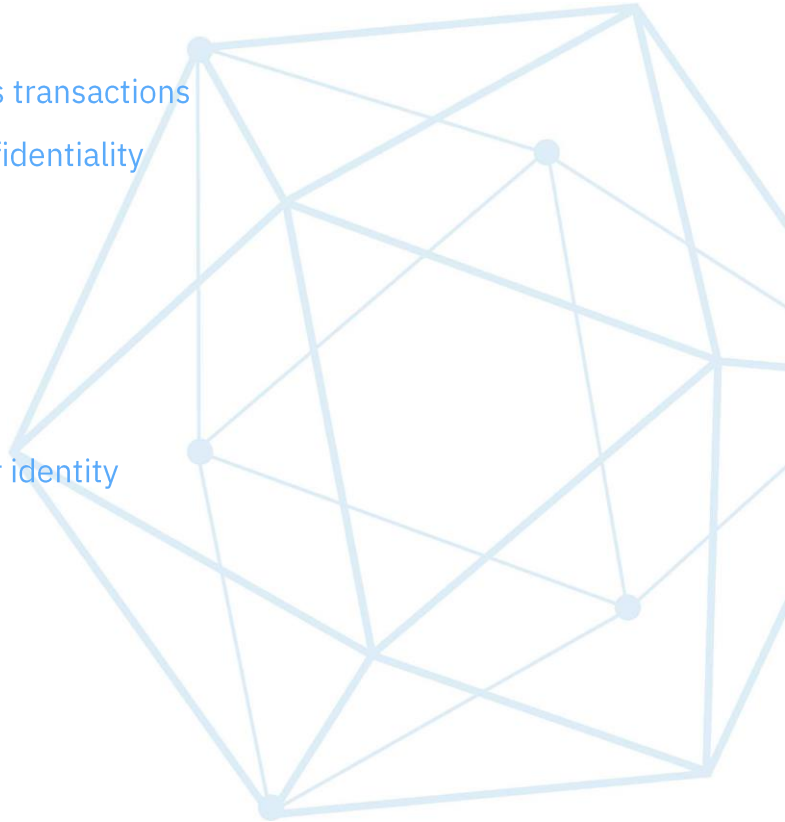
- Linux Foundation Hyperledger
  - A collaborative effort created to advance cross-industry blockchain technologies for business
- Hyperledger Fabric
  - An implementation of blockchain technology that is intended as a foundation for developing blockchain applications
  - Key technical features:
    - A shared ledger and smart contracts implemented as “chaincode”
    - Privacy and permissioning through membership services
    - Modular architecture and flexible hosting options
- V1.0 released July 2017: contributions by 159 engineers from 27 organizations
  - IBM is one contributor to Hyperledger Fabric

Hyperledger Fabric V1 Architecture

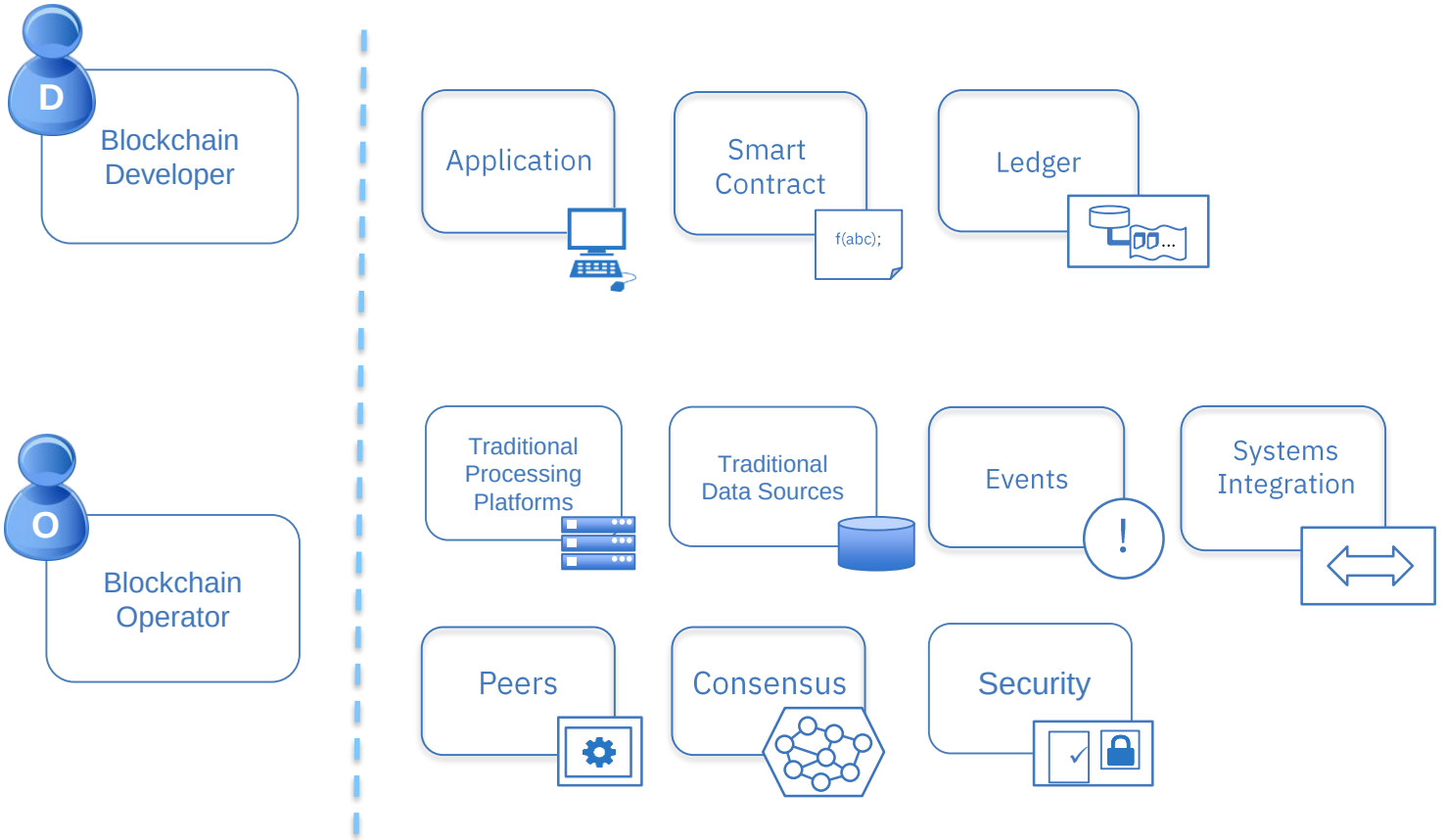


Overview of Hyperledger Fabric v1 – Design Goals

- Better reflect business processes by specifying who endorses transactions
- Support broader regulatory requirements for privacy and confidentiality
- Scale the number of participants and transaction throughput
- Eliminate non deterministic transactions
- Support rich data queries of the ledger
- Dynamically upgrade the network and chaincode
- Support for multiple credential and cryptographic services for identity
- Support for ”bring your own identity”



Recall key blockchain concepts



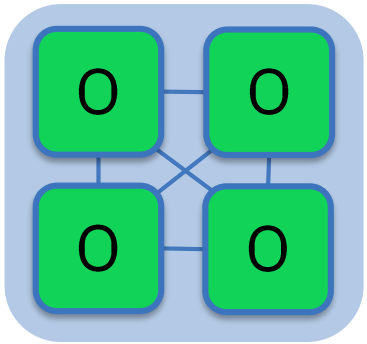
Hyperledger Fabric V1 - Deep Dive Topics

- Channels and Ordering Service
- Endorsement Policies
- Permissioned ledger access
- Pluggable world-state

Channels and Ordering Service

Ordering Service

The ordering service packages transactions into blocks to be delivered to peers. Communication with the service is via channels.



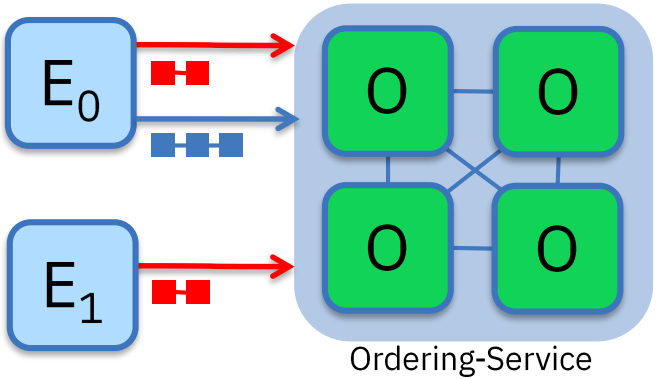
Ordering-Service

Different configuration options for the ordering service include:

– SOLO
  - Single node for development
– Kafka : Crash fault tolerant consensus
  - 3 nodes minimum
  - Odd number of nodes recommended

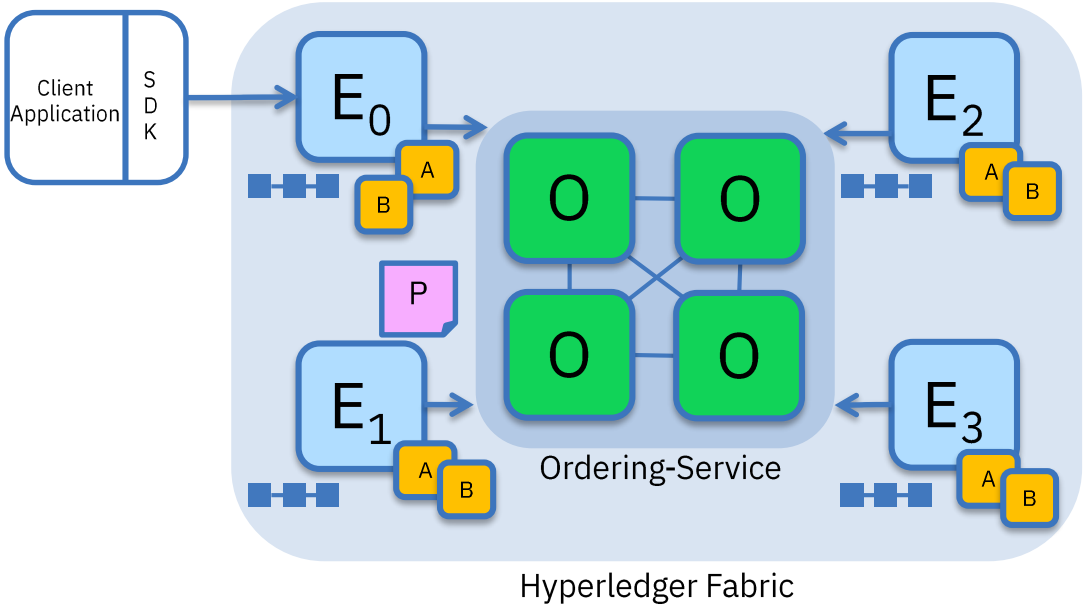
Channels

Separate channels isolate transactions on different ledgers

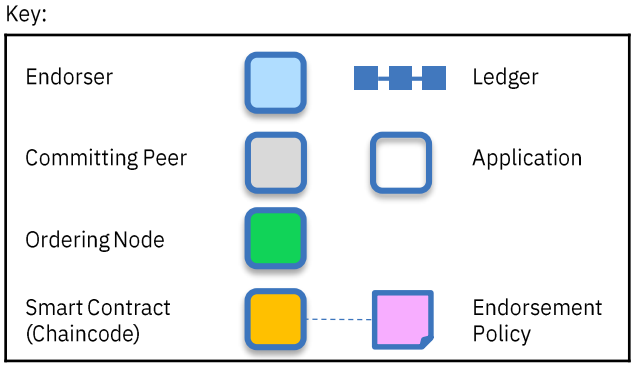


- Chaincode is installed on peers that need to access the worldstate
- Chaincode is instantiated on specific channels for specific peers
- Ledgers exist in the scope of a channel
  - Ledgers can be shared across an entire network of peers
  - Ledgers can be included only on a specific set of participants
- Peers can participate in multiple channels
- Concurrent execution for performance and scalability

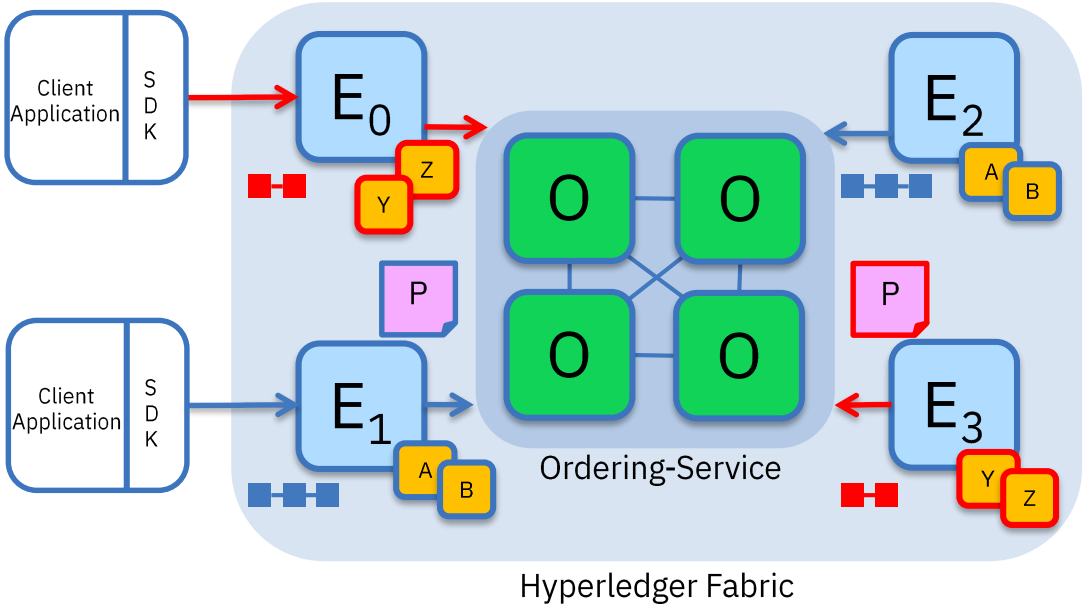
Single Channel Network



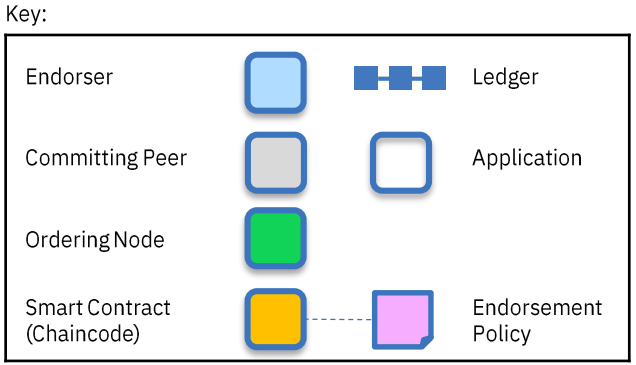
- Similar to v0.6 PBFT model
- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E_0, E_1, E_2 and E_3



Multi Channel Network



- Peers E_0 and E_3 connect to the red channel for chaincodes Y and Z
- Peers E_1 and E_2 connect to the blue channel for chaincodes A and B

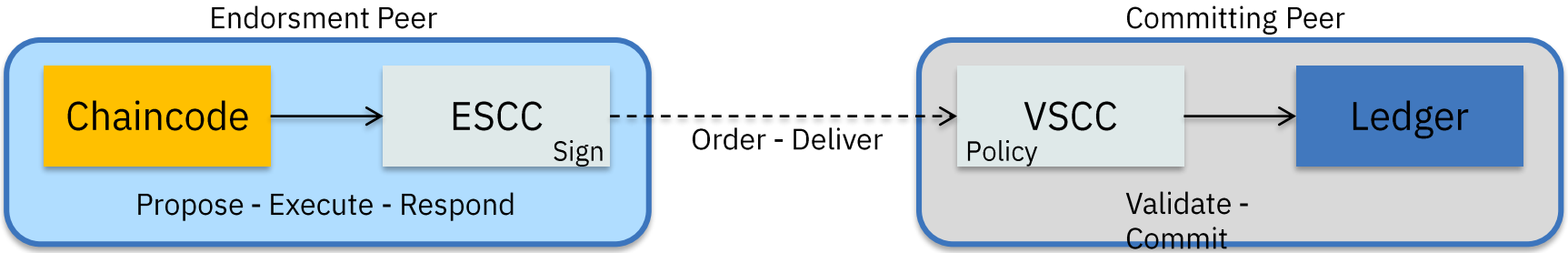


Endorsement Policies

Endorsement Policies

An endorsement policy describes the conditions by which a transaction can be endorsed. A transaction can only be considered valid if it has been endorsed according to its policy.

- Each chaincode is associated with an Endorsement Policy
- Default implementation: Simple declarative language for the policy
- ESCC (Endorsement System ChainCode) signs the proposal response on the endorsing peer
- VSCC (Validation System ChainCode) validates the endorsements



Endorsement Policy Syntax

```
$ peer chaincode instantiate
-C mychannel
-n mycc
-v 1.0
-p chaincode_example02
-c '{"Args":["init","a", "100",
"b","200"]}'
-P "AND('Org1MSP.member')“
```

This command instantiates the chaincode mycc on channel mychannel with the policy AND('Org1MSP.member')

Policy Syntax: EXPR(E[, E...])

Where EXPR is either AND or OR and E is either a principal or nested EXPR.

Principal Syntax: MSP.ROLE

Supported roles are: member and admin.

Where MSP is the MSP ID required, and ROLE is either “member” or “admin”.

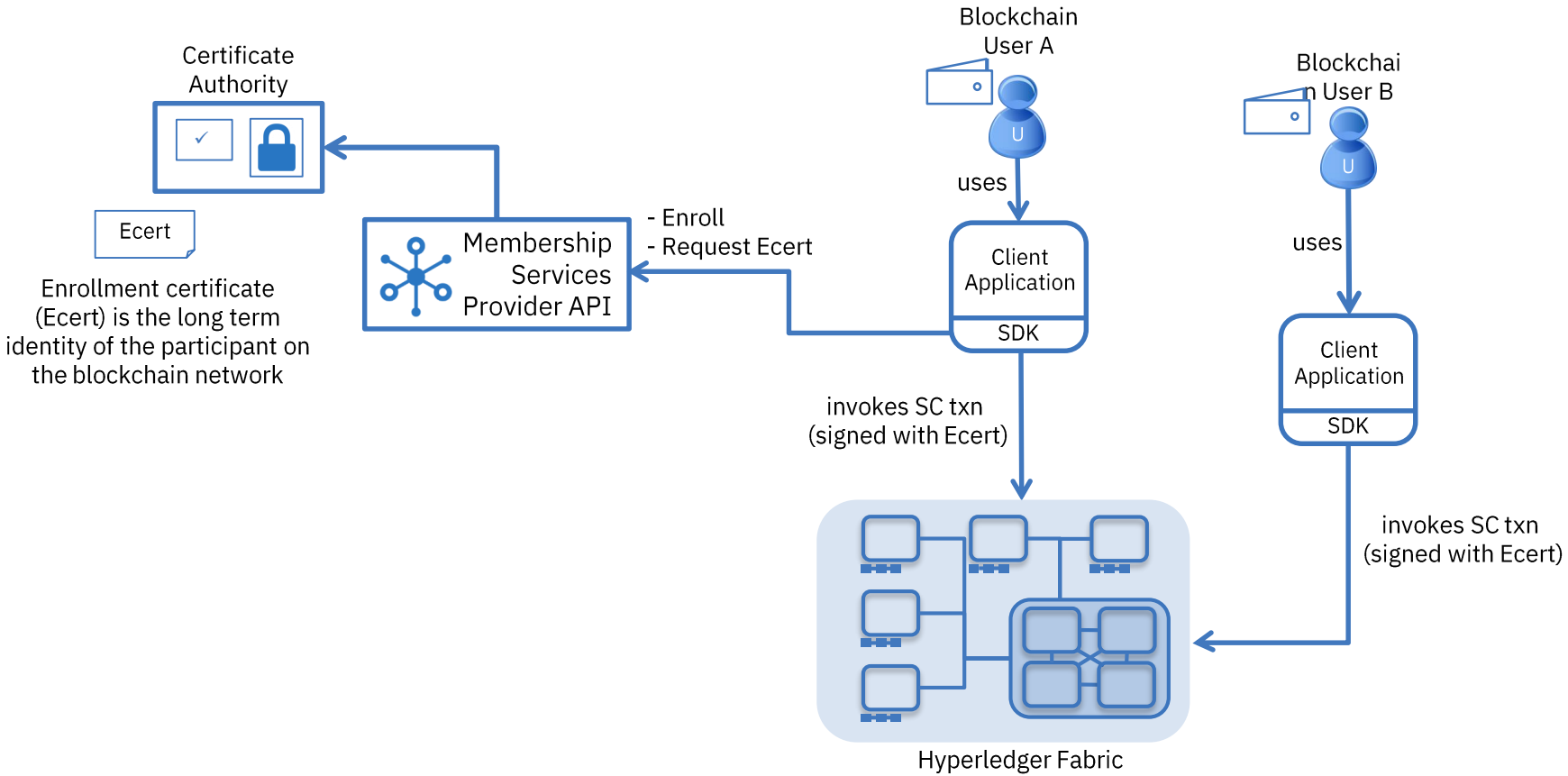
Endorsement Policy Examples

Examples of policies:

- Request 1 signature from all three principals
  - AND('Org1.member', 'Org2.member', 'Org3.member')
- Request 1 signature from either one of the two principals
  - OR('Org1.member', 'Org2.member')
- Request either one signature from a member of the Org1 MSP or (1 signature from a member of the Org2 MSP and 1 signature from a member of the Org3 MSP)
  - OR('Org1.member', AND('Org2.member', 'Org3.member'))

Permissioned Ledger Access

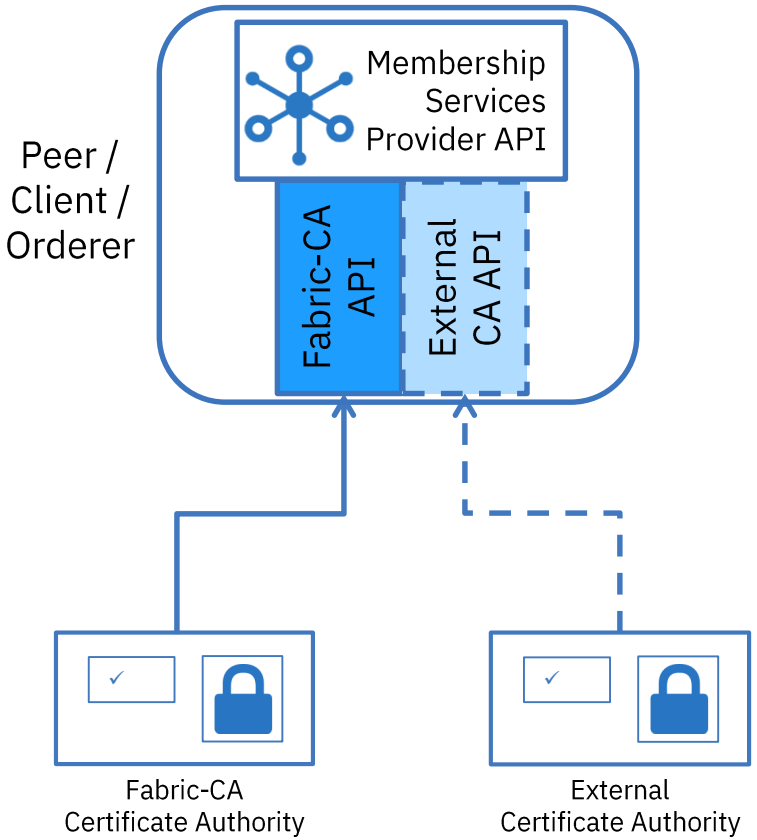
Membership Services Overview



Transaction and Identity Privacy

- Enrollment Certificates, Ecerts
  - Long term identity
  - Can be obtained offline, bring-your-own-identity
- Permissioned Interactions
  - Users sign with their Ecert
- Membership Services
  - Abstract layer to credential providers

Membership Services Provider API



Membership Services Provider API

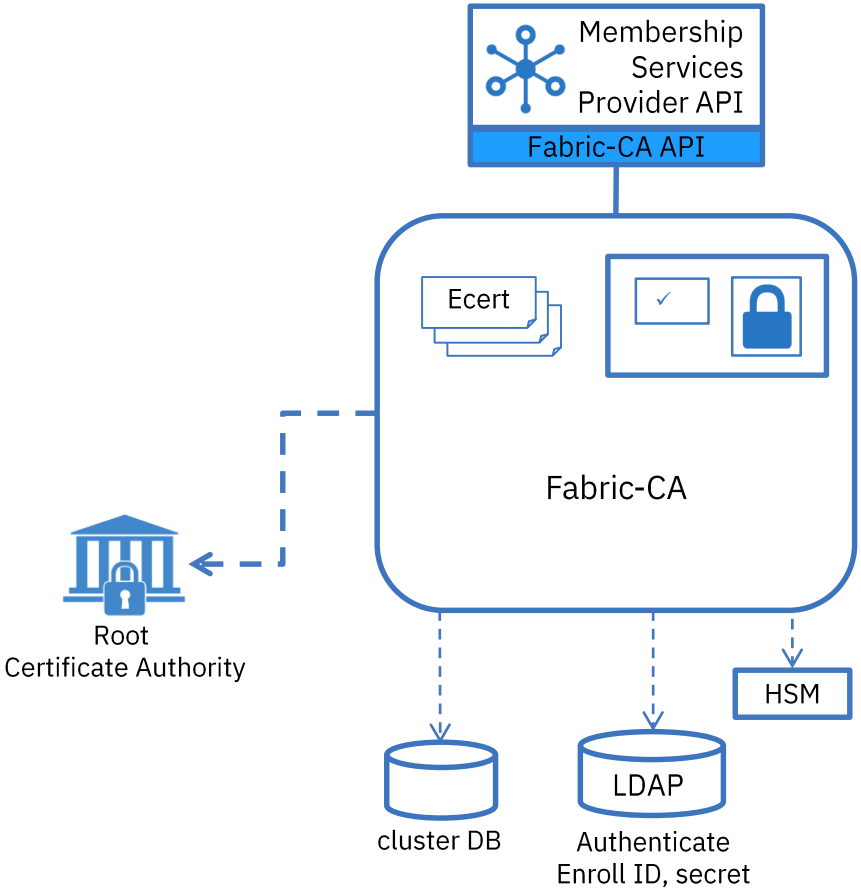
- Pluggable interface supporting a range of credential architectures
- Default implementation calls Fabric-CA.
- Governs identity for Peers and Users.
- Provides:
  - User authentication
  - User credential validation
  - Signature generation and verification
  - Optional credential issuance
- Additional offline enrollment options possible (eg File System).

Membership Services Provider (MSP)

- An abstraction to represent a membership authority and its operations on issuing and management of Hyperledger Fabric membership credentials in a modular & pluggable way
  - Allows for the co-existence of a variety of credential management architectures
  - Allows for easy organizational separation in credential management/administration operations according to business rules at a technical level
  - Potential to smoothly easily support different standards and membership implementations
  - Easy and straight-forward interface that the core can understand

- Described by a generic interface to cover:
  - User credential validation
  - User (anonymous but traceable) authentication: signature generation and verification
  - User attribute authentication: attribute ownership proof generation, and verification
  - (optionally) User credential issue

Fabric-CA Details



Fabric-CA

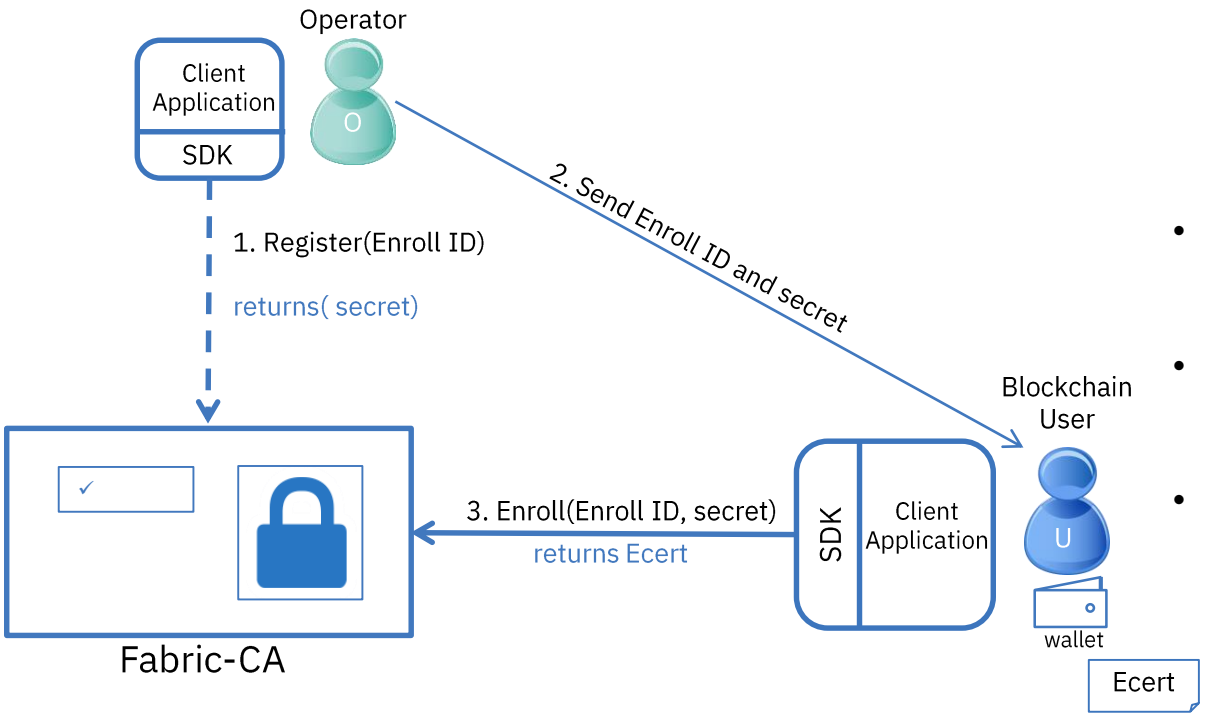
- Default implementation of the Membership Services Provider Interface.
- Issues Ecerts (long-term identity
- Supports clustering for HA characteristics
- Supports LDAP for user authentication
- Supports HSM

Fabric-CA

Certificate Authority

- Issues Ecerts and manages renewal and revocation
- Supports:
  - Clustering for HA characteristics
  - LDAP server for registration and enrollment
  - Hardware Security Modules

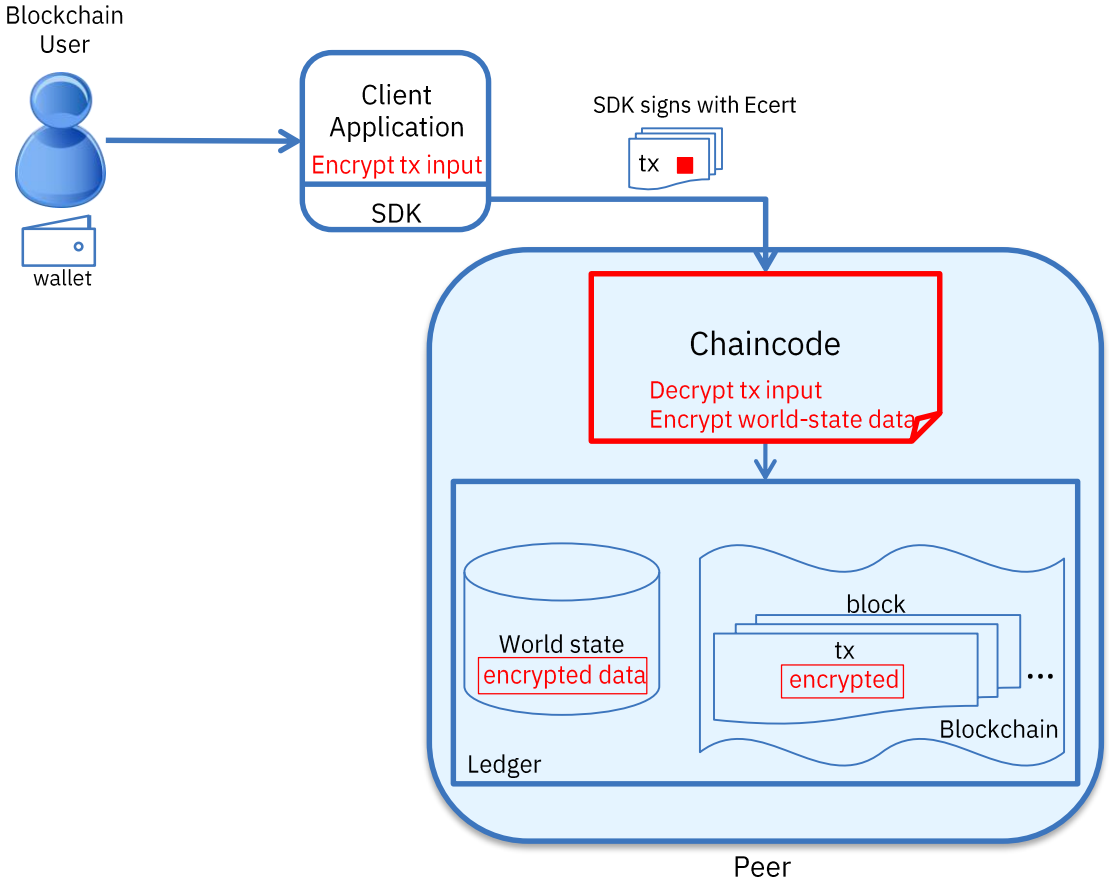
New User Registration and Enrollment



Registration and Enrollment

- Admin registers new user with Enroll ID
- User enrolls and receives credentials
- Additional offline registration and enrollment options available

Application Level Encryption



Data Encryption

Handled in the application domain.

Multiple options for encrypting:

- Transaction Data
- Chaincode*
- World-State data

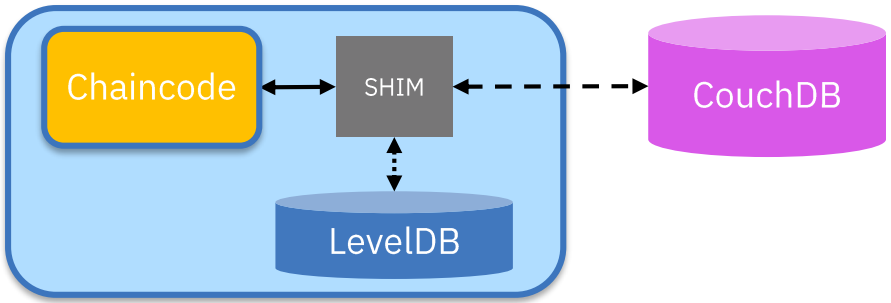
Chaincode optionally deployed with cryptographic material, or receive it in the transaction from the client application using the transient data field (not stored on the ledger).

*Encryption of application chaincode requires additional development of system chaincode.

Pluggable World State

WorldState Database

- Pluggable worldstate database
- Default embedded key/value implementation using LevelDB
  - Support for keyed queries, but cannot query on value
- Support for Apache CouchDB
  - Full query support on key and value (JSON documents)
  - Meets a large range of chaincode, auditing, and reporting requirements
  - Will support reporting and analytics via data replication to an analytics engine such as Spark (future)
  - Id/document data model compatible with existing chaincode key/value programming model



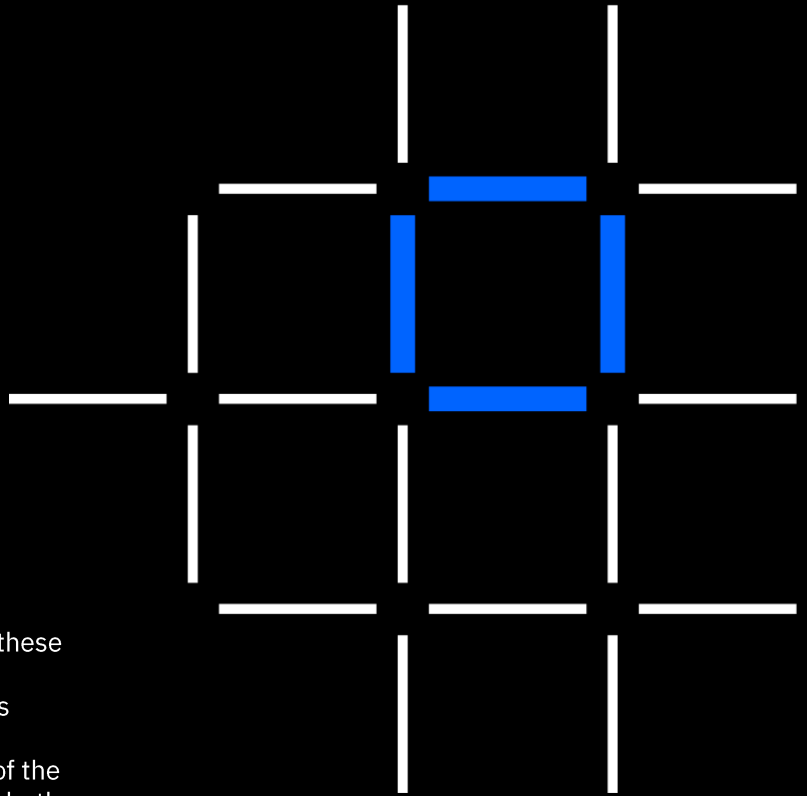
Thank you

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.

Hyperledger Technical deep dive

- Schema

```
type contract struct {
  Username         string    `json:"username"`
  Item             item      `json:"item"`
  StartDate        time.Time `json:"start_date"`
  EndDate          time.Time `json:"end_date"`
  Void             bool      `json:"void"`
  ContractTypeUUID string    `json:"contract_type_uuid"`
  ClaimIndex       []string  `json:"claim_index,omitempty"`
}

// Entity not persisted on its own
type item struct {
  ID          int32   `json:"id"`
  Brand       string  `json:"brand"`
  Model       string  `json:"model"`
  Price       float32 `json:"price"`
  Description string  `json:"description"`
  SerialNo    string  `json:"serial_no"`
}

// Key consists of prefix + UUID of the contract + UUID of the claim
type claim struct {
  ContractUUID  string      `json:"contract_uuid"`
  Date          time.Time   `json:"date"`
  Description   string      `json:"description"`
  IsTheft       bool        `json:"is_theft"`
  Status        ClaimStatus `json:"status"`
  Reimbursable  float32     `json:"reimbursable"`
  Repaired      bool        `json:"repaired"`
  FileReference string      `json:"file_reference"`
}
```

Hyperledger Technical deep dive

- Retrieve data from the ledger

```
type contract struct {
  Username         string    `json:"username"`
  Item             item      `json:"item"`
  StartDate        time.Time `json:"start_date"`
  EndDate          time.Time `json:"end_date"`
  Void             bool      `json:"void"`
  ContractTypeUUID string    `json:"contract_type_uuid"`
  ClaimIndex       []string  `json:"claim_index,omitempty"`
}
```

```
func (u *user) Contacts(stub shim.ChaincodeStubInterface) []contract {
  contracts := make([]contract, 0)

  // for each contractID in user.ContractIndex
  for _, contractID := range u.ContractIndex {

    c := &contract{}

    // get contract
    contractAsBytes, err := stub.GetState(contractID)
    if err != nil {
      //res := "Failed to get state for " + contractID
      return nil
    }

    // parse contract
    err = json.Unmarshal(contractAsBytes, c)
    if err != nil {
      //res := "Failed to parse contract"
      return nil
    }

    // append to the contracts array
    contracts = append(contracts, *c)
  }

  return contracts
}
```

Hyperledger Technical deep dive

- Store data into ledger

```
type contract struct {
  Username         string    `json:"username"`
  Item             item      `json:"item"`
  StartDate        time.Time `json:"start_date"`
  EndDate          time.Time `json:"end_date"`
  Void             bool      `json:"void"`
  ContractTypeUUID string    `json:"contract_type_uuid"`
  ClaimIndex       []string  `json:"claim_index,omitempty"`
}
```

```
} else {
  // Validate if the user with the provided username exists
  userAsBytes, _ := stub.GetState(userKey)
  if userAsBytes == nil {
    return shim.Error("User with this username does not exist.")
  }
}

contract := contract{
  Username:         dto.Username,
  ContractTypeUUID: dto.ContractTypeUUID,
  Item:             dto.Item,
  StartDate:        dto.StartDate,
  EndDate:          dto.EndDate,
  Void:             false,
  ClaimIndex:       []string{},
}

contractKey, err := stub.CreateCompositeKey(prefixContract, []string{dto.Username, dto.UUID})
if err != nil {
  return shim.Error(err.Error())
}
contractAsBytes, err := json.Marshal(contract)
if err != nil {
  return shim.Error(err.Error())
}
err = stub.PutState(contractKey, contractAsBytes)
if err != nil {
  return shim.Error(err.Error())
}
```