



# COMP6452

## Software Architecture for Blockchain Applications: Introduction

Ingo Weber | Principal Research Scientist & Team Leader  
Architecture & Analytics Platforms (AAP) team  
[ingo.weber@data61.csiro.au](mailto:ingo.weber@data61.csiro.au)

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

[www.data61.csiro.au](http://www.data61.csiro.au)

## Agenda:

- Part 1: Course Summary

- Lecturers and Tutor
- Learning Outcomes, Course Outline, Assessments

- Part 2: Topic Overview

- What is Blockchain, and Why Does it Matter?
- Blockchain-based Applications
- Blockchain Functionality
- Blockchain Non-functional Properties
- Blockchain Architecture Design

## Agenda:

- Part 1: Course Summary

- Lecturers and Tutor
- Learning Outcomes, Course Outline, Assessments

- Part 2: Topic Overview

- What is Blockchain, and Why Does it Matter?
- Blockchain-based Applications
- Blockchain Functionality
- Blockchain Non-functional Properties
- Blockchain Architecture Design

# Who are we?



# Lecturers and Tutor



- Dr Ingo Weber:

- Principal Research Scientist & Team Leader @ Data61, CSIRO; Conjoint Assoc. Prof. @ CSE, UNSW; Adjunct Assoc. Prof. @ Swinburne University.
- PhD from University of Karlsruhe (TH); MSc from the University of Massachusetts, Amherst, USA.



- Dr Xiwei (Sherry) Xu:

- Senior Research Scientist @ Data61, CSIRO & Conjoint Lecturer @ CSE, UNSW
- PhD from UNSW
- Working on blockchain since 2015



- Dr Mark Staples:

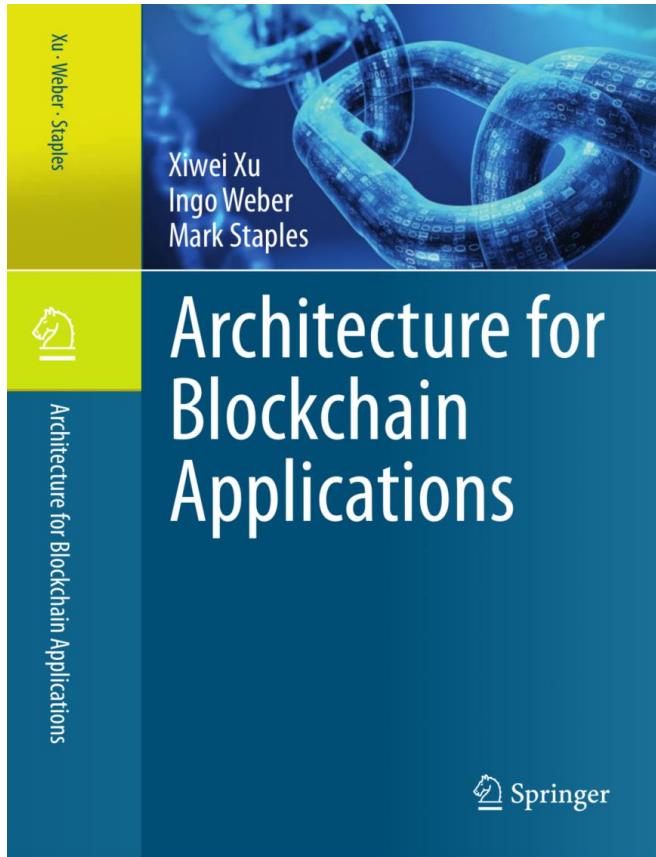
- Senior Principal Research Scientist & Team Leader @ Data61, CSIRO; Conjoint Assoc. Prof. @ CSE, UNSW
- PhD from University of Cambridge; BSc and BInfTech (Hons) from University of Queensland
- Leading member of ISO standardization for Blockchain (ISO/TC 307).



- Sin Kuang Lo:

- PhD student @ Data61, CSIRO and UNSW
- Tutor

# Book: Architecture for Blockchain Applications



*Xiwei Xu, Ingo Weber, Mark Staples.*  
*Architecture for Blockchain*  
*Applications.*  
*Springer, 2019.*

# CSIRO: Australia's National Science Agency

TOP 10  
applied research  
agencies globally

WIFI

WASP

TOP 1%  
of global research  
institutions in 14 of  
22 research fields

TOP 0.1%  
of global research  
institutions in 4 of 22  
research fields

ZEBEDEE

# Data61: Australia's Digital Innovation Powerhouse

1100+  
employees  
[including  
students]

31  
Government  
partners

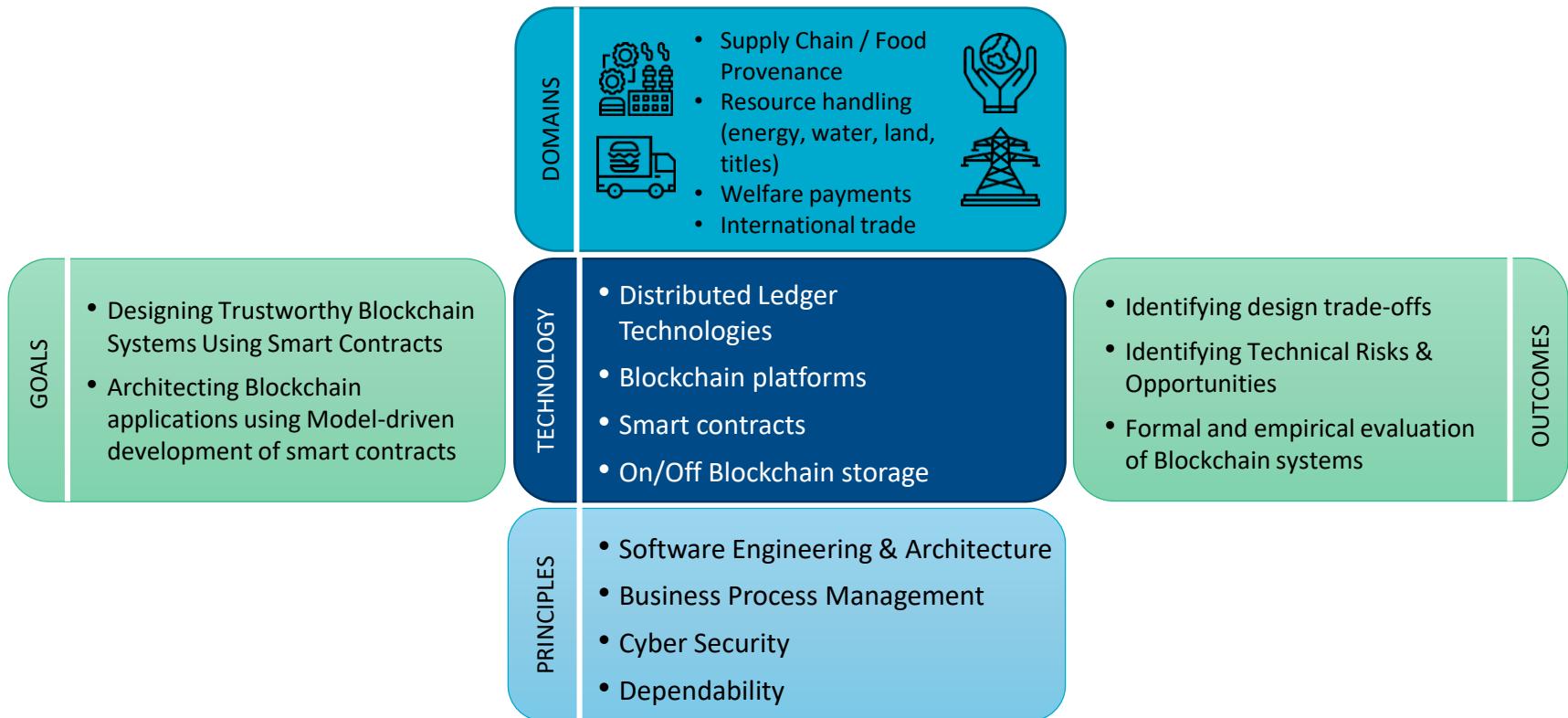
91  
Corporate  
partners

38  
University  
partners

190+  
data-driven  
projects

172  
patents

# Blockchain Research at Data61 – Overview



# Course Structure



# Learning Outcomes

After the course, you will be able to:

- Explain the principles of blockchain and which roles it can play in an application architecture
- Decide on the suitability of blockchains and how to design applications on them
- Make functional and non-functional trade-offs for blockchain-based applications
- Build small applications on blockchain

# Course Outline (1)

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
1st	18 Feb	Ingo Weber	Introduction	1. Introduction	
2nd	25 Feb	Ingo Weber	Existing Blockchain Platforms	4. Example use cases 2. Existing Blockchain Platforms (1h on smart contract dev)	Assignment 1 out (Monday before lecture)
3rd	4 Mar	Sherry Xu	Blockchain in Software Architecture 1	3. Varieties of blockchain 5. Blockchain in Software Architecture (including software architecture basics) 1/2	
4th	11 Mar	Mark Staples	Blockchain in Software Architecture 2	5. Blockchain in Software Architecture (Non-functional properties and trade-offs) 2/2	Pitching session Assignment 1 due (Wednesday)
5th	18 Mar	Sherry Xu	NFPs 1	6. Design Process for Applications on Blockchain (DevOps of blockchain) 9. Cost	
6th	25 Mar	Mark Staples	NFPs 2	10. Performance	Mid-term Exam (1 hour)

# Course Outline (2)

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
7th	1 Apr	Mark Staples	NFPs 3	11. Dependability and Security	Assignment 2 out (Monday before lecture)
8th	8 Apr	Sherry Xu	Design Patterns for Blockchain Applications	7. Blockchain Patterns (design pattern basics, design pattern language)	
9th	15 Apr	Ingo Weber	Model-Driven Engineering	8. Model-driven Engineering for Applications on Blockchains (model-driven basics, UML)	Assignment 2 due (After tutorial)
10th	22 Apr			Easter Holiday	
11th	29 Apr	Guest Lecturer + Mark Staples	Guest Lecture and Summary	Case study chapters Summary (including Epilogue content) Discuss disruptive potential, high-level opportunities and risks	Guest Lecture 2-hour and Summary of the Lecture

# Assessments

Assessment Title	Assessment Type	Marks
Assignments (2)	Assignment  Two assignments where you will analyse and implement blockchain scenarios.	2x 12.5
Mid-term quiz	Quiz Test  Mid-term quiz conducted in class on material covered up to that point.	25
Final Exam	Examination (central)  A written examination, testing all course content, with a focus on material from the second half of the course. Students must obtain a <b>passing grade on the exam</b> in order to pass the course (i.e. < 50% of the exam points means failing the course).	50

# Marking & Course Website

- To pass the course, your **overall mark must be 50 or higher**, and your **mark in the final exam must be 25 or higher**. The overall final mark will be the sum of your marks for each component if you pass the course.
- Marking will be done according to this formula:

```
if (final >= 25)
    then total = ass1 + ass2 + mid_exam + final_exam;
else
    total = final_exam * 2;
```
- Course website:  
<https://webcms3.cse.unsw.edu.au/COMP6452/19T1/>

## Agenda:

- Part 1: Course Summary

- Lecturers and Tutor
- Learning Outcomes, Course Outline, Assessments

- Part 2: Topic Overview

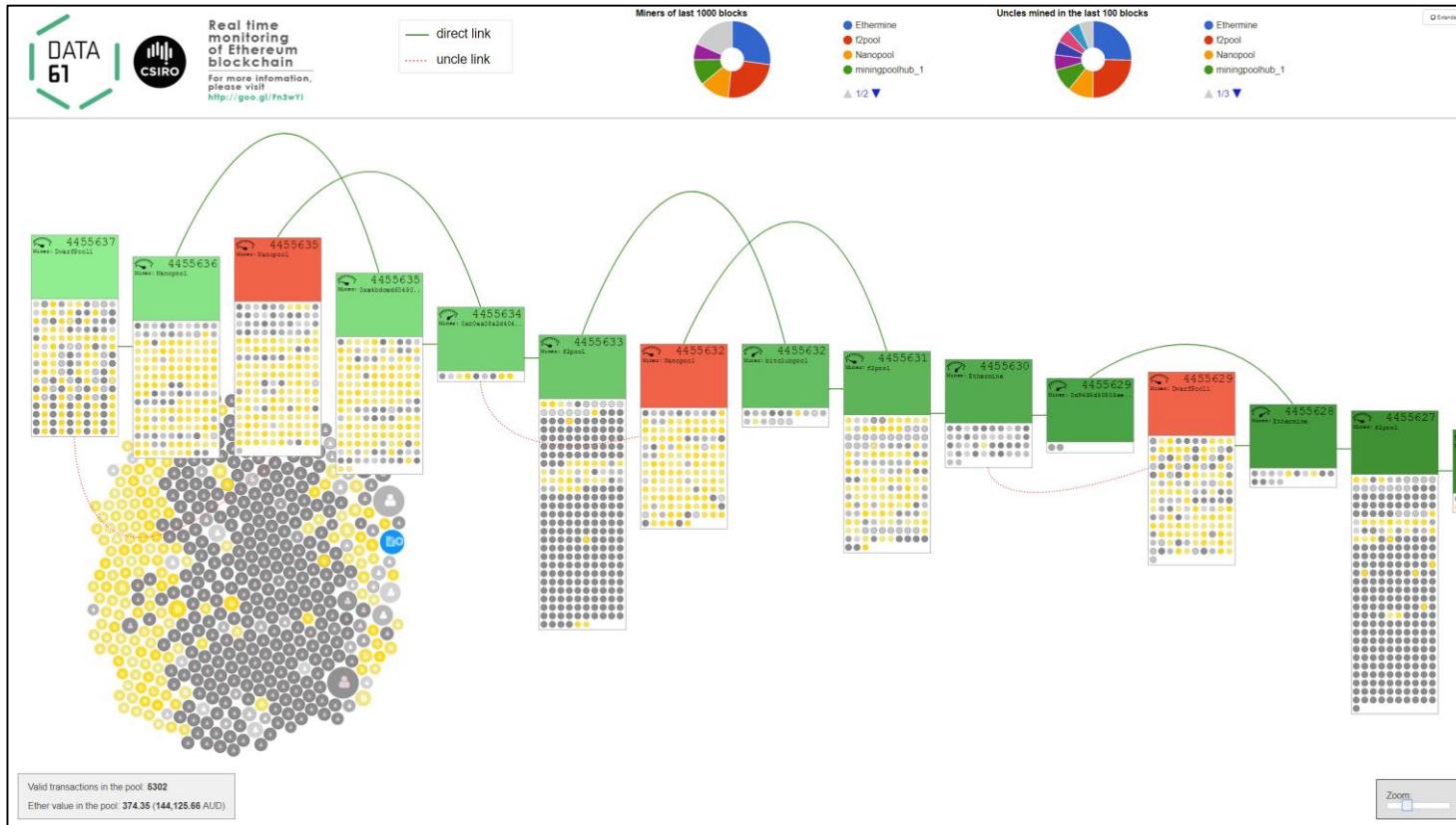
- What is Blockchain, and Why Does it Matter?
- Blockchain-based Applications
- Blockchain Functionality
- Blockchain Non-functional Properties
- Blockchain Architecture Design

# What is Blockchain, and Why Does it Matter?



# What is a Blockchain?

Visualization of a Blockchain: <http://ethviewer.live>



# What is the Beef about Blockchain?

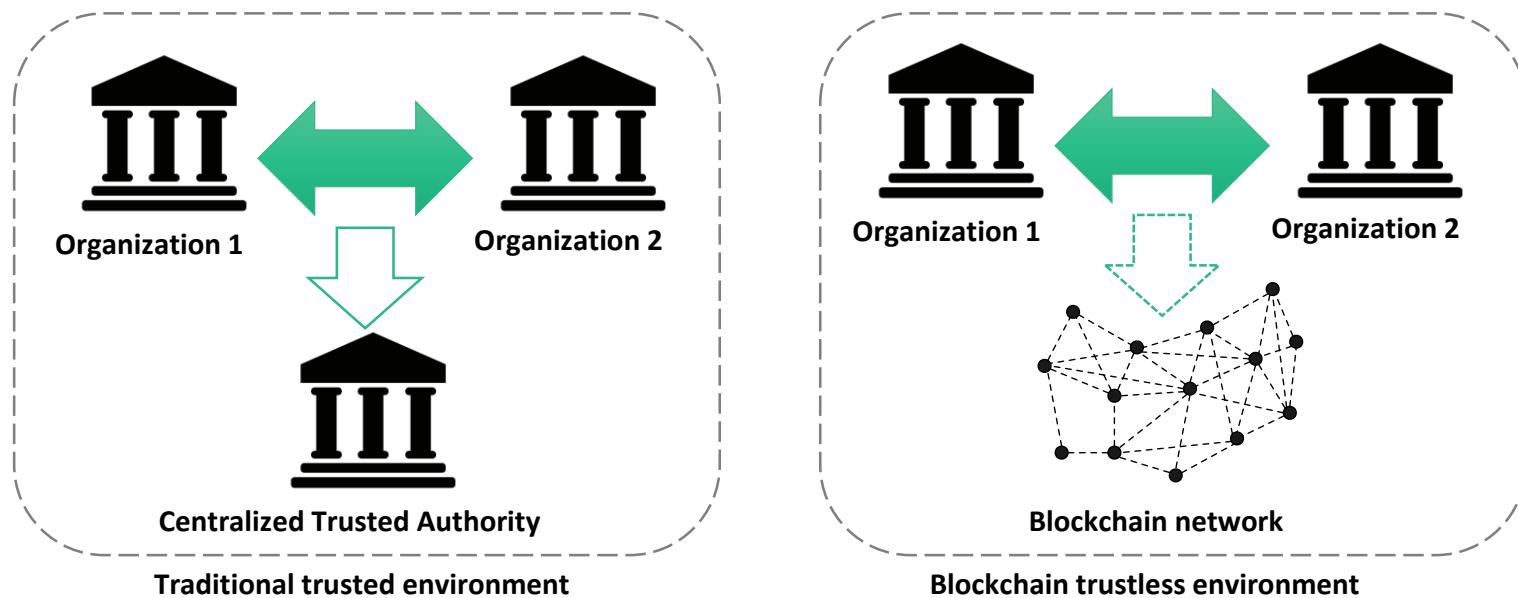


Applications coming soon:

- NSW electronic driver licences
- ASX system for settlement of trades

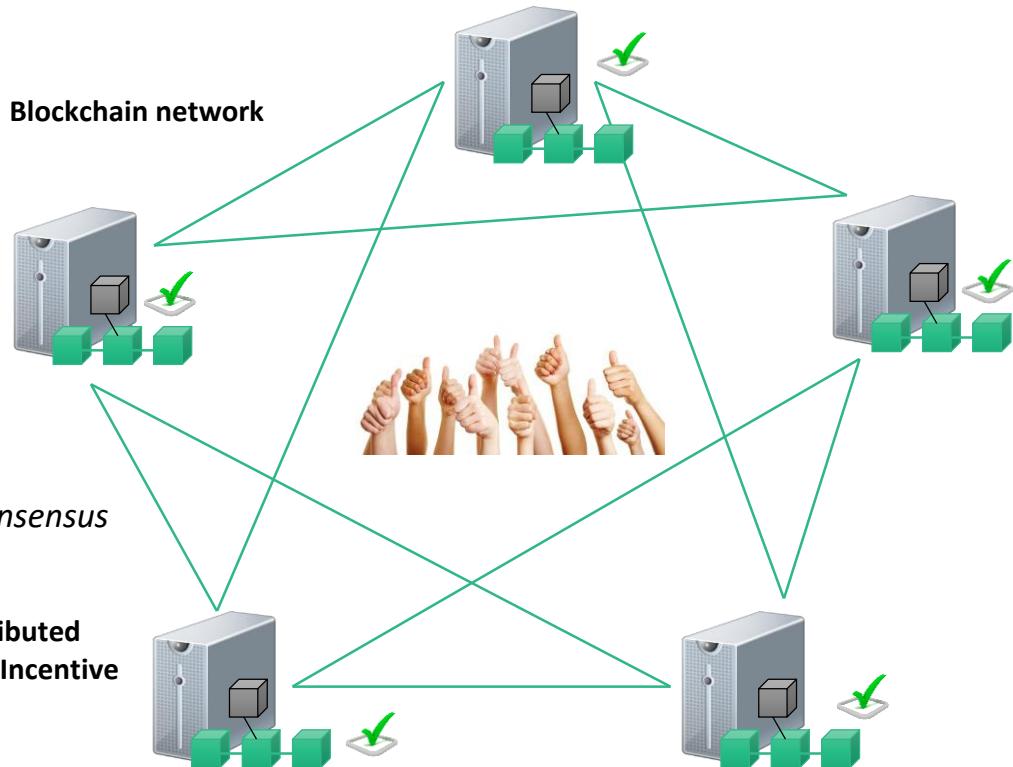
Video & reports: <https://www.data61.csiro.au/blockchain>

# Blockchain – replacing centralized trusted authority

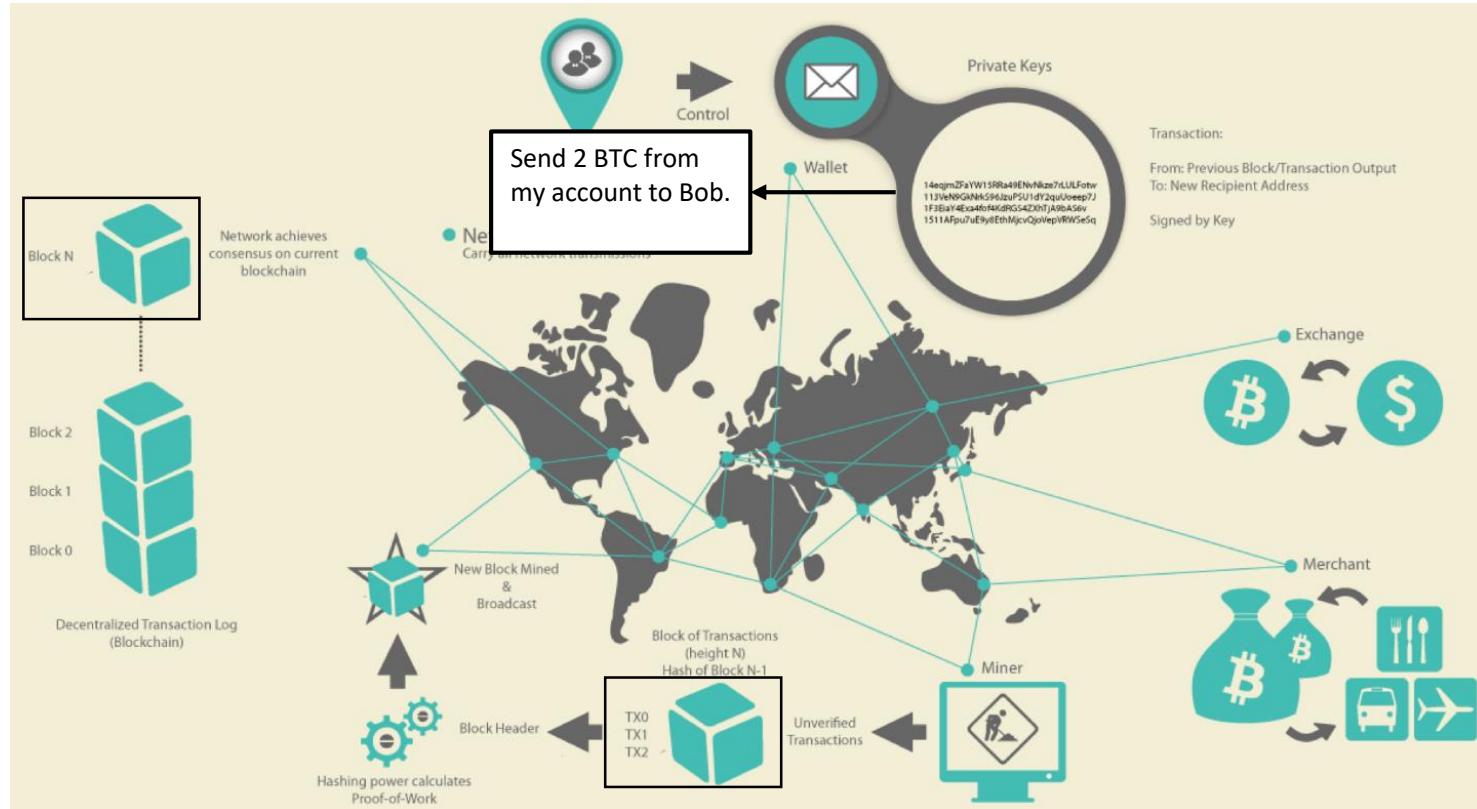


# How?

- **Immutable data base**
  - Public ledger
- **Every node hosts a replica**
  - Distributed consensus
    - *No central owner of consensus*
- **Transaction is verified by every node**
- **Combination of knowledge from Distributed Systems, Peer-to-Peer, Cryptography, Incentive Systems and Game Theory**



# Blockchain 1<sup>st</sup> gen — Cryptocurrency



Users:

- create transactions,
- sign them, and
- announce them to network

Miners:

- receive transactions
- include them in a new block,
- (try to) append the new block to the data structure

When a transaction is part of the data structure, it has taken place (though it's a bit more complicated – more later).

# Blockchain 2<sup>nd</sup> gen – Smart Contracts

- 1<sup>st</sup> gen blockchains: transactions are financial transfers
- Now Blockchain ledger can do that, and store/transact any kind of data
- Blockchain can deploy and execute programs: Smart Contracts
  - User-defined code, deployed on and executed by whole network
  - Can enact decisions on complex business conditions
  - Can hold and transfer assets, managed by the contract itself
  - Ethereum: pay per assembler-level instruction



Approved!



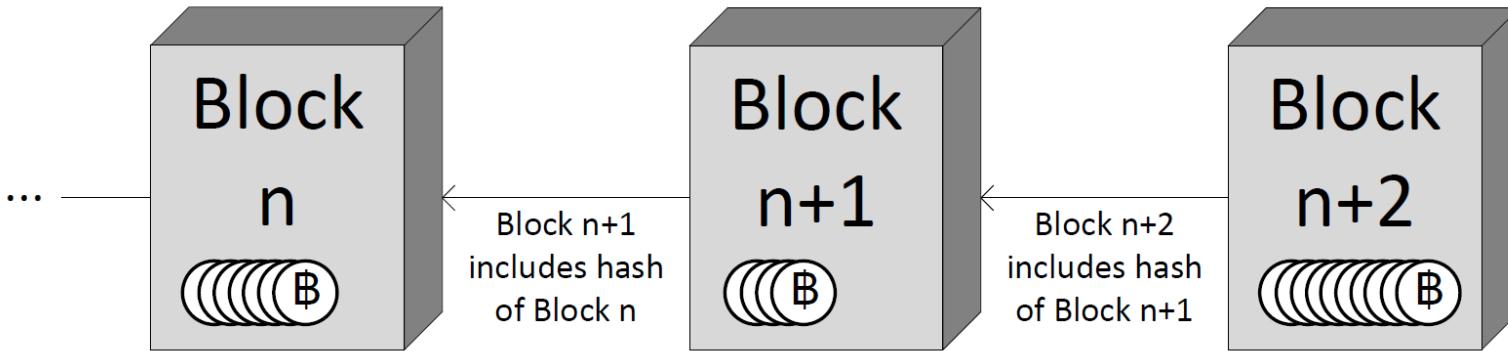
# So what?

- Well, blockchains are exciting because they can be used as a new foundation for re-imagining systems:
  - a neutral infrastructure for processing transactions and executing programs
  - potentially interesting for innovation at **all touch-points** between organizations or individuals
- **blockchain applications have the potential to disrupt the fabric of society, industry, and government**
- Blockchains can also be used as a technology platform to handle hard issues of data replication and system state synchronization with high integrity.



# Defining Blockchain (1)

- **Distributed Ledger**
  - An “append-only” transaction store distributed across machines
  - A new transaction might reverse a previous transaction, but both remain part of the ledger
- **Blockchain**
  - A distributed ledger structured into a linked list of blocks.
  - Each block contains an ordered set of transactions
  - Use cryptographic hashes to secure the link from a block to its predecessor.



# Defining Blockchain (2)

- A **Blockchain System** consists of
  - A blockchain network of nodes
  - A blockchain data structure
    - For the ledger replicated across the blockchain network
    - Full nodes hold a full replica of the ledger
  - A network protocol
    - Defines rights, responsibilities, and means of communication, verification, validation, and consensus across the nodes in the blockchain network
    - Includes ensuring authorisation and authentication of new transactions, mechanisms for appending new blocks, incentive mechanisms

# Defining Blockchain (3)

- A **Public Blockchain** is a blockchain system with the following characteristics:
  - Has an open network
    - Nodes can join and leave without requiring permission from anyone
    - All full nodes can verify new transactions and blocks
    - Incentive mechanism to ensure the correct operation
      - Valid transactions are processed and included in the ledger and invalid transactions are rejected
- A **Blockchain Platform** is the technology needed to operate a blockchain
  - Blockchain client software for processing nodes
  - The local data store
  - Alternative clients to access the blockchain network

# Decentralised Applications and Smart Contracts

- **Smart contracts**

- Programs deployed as data and executed in transactions on the blockchain
- Blockchain can be a computational platform (more than a simple distributed database)
- Code is deterministic and immutable once deployed
- Can invoke other smart contracts
- Can hold and transfer digital assets

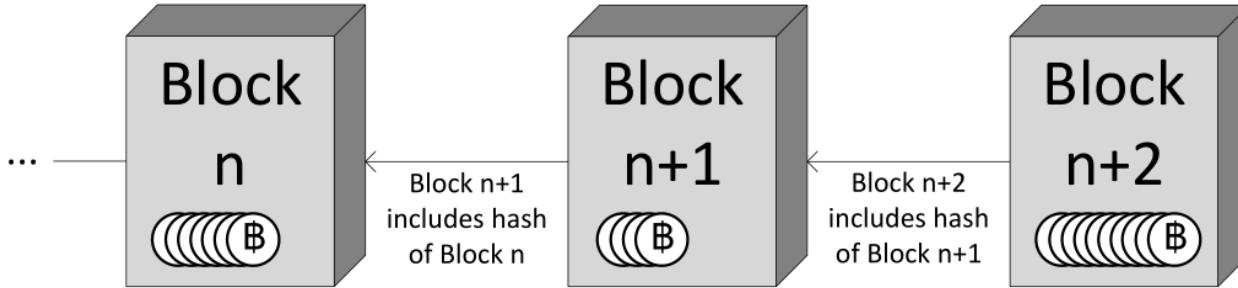
- **Decentralized applications or dapps**

- Main functionality is implemented through smart contracts
- Backend is executed in a decentralized environment
- Frontend can be hosted as a web site on a centralized server
  - Interact with its backend through an API
- Could use decentralized data storage such as IPFS
- “State of the dapps” is a directory recorded on blockchain: <https://www.stateofthedapps.com/>

# Blockchain defined (1/4)

## Verbatim from the Book

- **Definition 1 (Distributed Ledger).** A Distributed Ledger is an *append-only store of transactions* which is distributed across many machines.
- **Definition 2 (Blockchain (Concept)).** A Blockchain is a *distributed ledger* that is structured into a *linked list of blocks*. Each block contains an ordered set of transactions. Typical solutions use cryptographic hashes to secure the link from a block to its predecessor.



# Blockchain defined (2/4)

## Verbatim from the Book

- **Definition 3 (Blockchain System).** A Blockchain System consists of:
  - a *blockchain network* of machines, also called *nodes*;
  - a *blockchain data structure*, for the ledger that is replicated across the blockchain network. Nodes that hold a full replica of this ledger are referred to as *full nodes*;
  - a network *protocol* that defines rights, responsibilities, and means of communication, verification, validation, and consensus across the nodes in the network. This includes ensuring *authorization and authentication* of new transactions, mechanisms for appending new blocks, incentive mechanisms (if needed), and similar aspects.

# Blockchain defined (3/4)

## Verbatim from the Book

- **Definition 4 (Public Blockchain).** A Public Blockchain is a *blockchain system* that has the following characteristics:
  - it has an *open network* where nodes can join and leave as they please without requiring permission from anyone;
  - all full nodes in the network can *verify each new piece of data* added to the data structure, including blocks, transactions, and effects of transactions; and
  - its protocol includes an *incentive mechanism* that aims to ensure the correct operation of the blockchain system including that valid transactions are processed and included in the ledger, and that invalid transactions are rejected.

# Blockchain defined (4/4)

## Verbatim from the Book

- **Definition 5 (Blockchain Platform).** A blockchain platform is the *technology needed to operate a blockchain*. This comprises the blockchain client software for processing nodes, the local data store for nodes, and any alternative clients to access the blockchain network.
- **Definition 6 (Smart Contract).** Smart contracts are *programs* deployed as data in the blockchain ledger, and executed in transactions on the blockchain. Smart contracts can *hold and transfer digital assets* managed by the blockchain, and can invoke other smart contracts stored on the blockchain. Smart contract code is *deterministic and immutable* once deployed.
- **Definition 7 (dapp).** A decentralized application or dapp is a software system that is designed to provide its main functionality through smart contracts.

# Cryptocurrencies and Tokens

- **Cryptocurrencies**

- ‘Baked in’ to the core platform of public blockchains -base currency of blockchains
- Symbiotic relationship
  - Blockchain keeps track of the ownership of portions of that currency, e.g. Alice owned 2Ether, transferred 1 Ether to Bob, offered 0.01Ether to miner
  - Cryptocurrency enables the incentive mechanism for blockchain operations

- **Digital tokens**

- Created and exchanged using smart contracts
- Represent assets
  - Fungible asset: individual units are interchangeable, e.g. company share, gold
  - Non-fungible asset: represents a unique asset, e.g. cryptokitties, car title

- **Not all applications are the same:**

- Transferring coins / tokens vs. tracking movement of physical goods
- Core difference: where is the default version of the truth, on or off-chain?

# Fungible and Non-fungible Tokens

- Fungible tokens: interchangeable
  - E.g., \$2 coin, \$10 note
  - Main concern: how many?
  - Ethereum: ERC20 standard
    - [https://theethereum.wiki/w/index.php/ERC20\\_Token\\_Standard](https://theethereum.wiki/w/index.php/ERC20_Token_Standard)
  - Example: OmiseGO (OMG).

“The OmiseGO blockchain comprises a decentralized exchange, liquidity provider mechanism, clearinghouse messaging network, and asset-backed blockchain gateway. ... It uses the mechanism of a protocol token to create a proof-of-stake blockchain to enable enforcement of market activity amongst participants. Owning OMG tokens buys the right to validate this blockchain, within its consensus rules.”
- Non-fungible tokens
  - E.g., houses, cars, patents
  - Main concern: which ones?
  - Ethereum: ERC721
    - <https://github.com/ethereum/EIPs/issues/721>
  - Example: cryptokitties  
<https://www.cryptokitties.co/>



- Kitties are non-fungible, individual, and their appearance depends on the individual features

# Blockchain Applications



# Blockchain-based Application

- A **blockchain-based application** (or just **blockchain application**) makes significant use of blockchain
  - dapps are an example, but the concept is far broader
  - significant portions of such applications can be based on traditional systems.
- Globally, many financial services companies, enterprises, startups, and governments are exploring suitable applications
- Areas include supply chain, electronic health records, voting, energy supply, ownership management, and protecting critical civil infrastructure
- By now, most if not all industry sectors have explored blockchain use
- The course (and the book) are about what you need to know to design and build blockchain-based applications

# Application Areas – Enterprises and Industry

- **Supply chain:** store key events to ensure goods provenance and logistic visibility
- **IoT:** device access control and software/configuration updates
- **Utility resources and services:** monitoring and payment of usage
- **Digital rights and IP management:** A trusted media asset registry to store hashes, meta-data or other identifier on blockchain and manage access and right information
- **Data management:** A metadata layer for decentralized data sharing and analytics - to discover and integrate large datasets and data analytics services
- **Proof of existence:** store a timestamped record of a hash of the document
- **Inter-divisional accounting:** A shared distributed ledger of inter-divisional accounts
- **Corporate affairs:** Board and shareholder voting and registration

# Application Areas – Financial Services

- **Digital currency**
  - New form of money transferred without 3rd parties
  - Programmable money: attach policies to specific parcels of currency
- **(International) payments**
  - Via digital currency with local exchanges between the digital currency and FIAT currencies
  - Pseudonymous: but usually have regulatory requirements to have identity, e.g. Anti-Money Laundering (AML)
- **Reconciliation for correspondent banking**
  - Banks can create a single shared ledger of accounts maintained in real-time
- **Securities settlement**
  - The exchanged assets are represented by tokens using smart contracts
  - Payment are made using tokens or native cryptocurrency
- **Markets**
  - Provides a platform for making and accepting offers to trade assets or services
  - Record the status of the offers
- **Trade finance**
  - Evidence trade-related documents
  - Automate payments

# Application Areas – Government Services

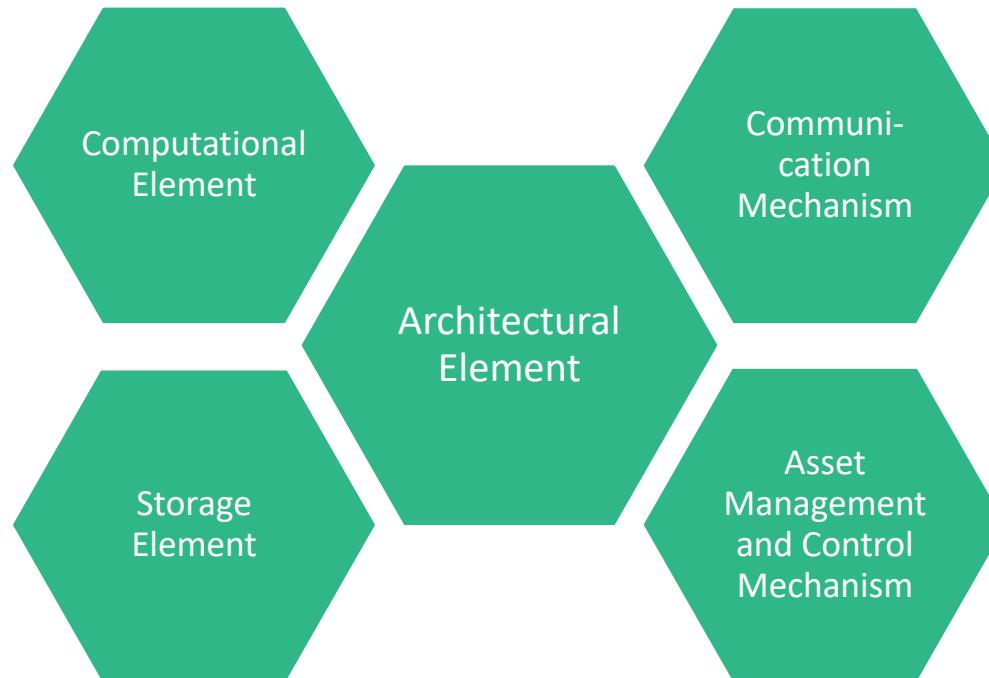
- **Registries and identity**
  - Including identities of persons, companies or devices; licensing; qualification; and certification
- **Grants and social security**
  - Automate the process coordination for application, decision making, and payment distribution
  - Allow conditional payments through programmable money – e.g. NDIS – where the money checks the conditions for spending it when attempting to do so
- **Resource quota management**
  - Government granted quotas, allocations, rights to physical resources could be awarded and tracked through tokens
  - E.g. water access licenses can provide rights to take a certain volume of water from specific sources during specific time frames
- **Taxation**
  - Automate tax collection using smart contract

# Blockchain Functionality



# Functions Blockchain can provide in a dapp

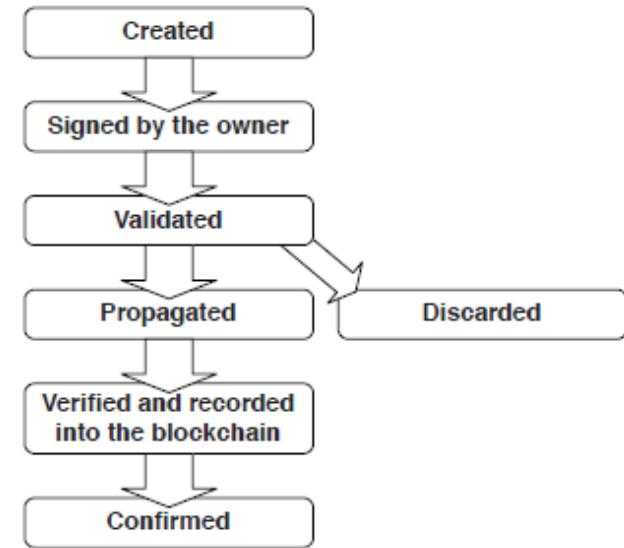
- Blockchain as...



# Blockchain Functionality

## Blockchain as Storage Element

- Append only data store – prevent tampering of data
  - An ordered list of blocks, where each block contains a small (possibly empty) list of transactions.
  - Each block in is “chained” back to the previous block, by containing a hash of the previous block
- Transactions
  - Information is recorded within the transactions
  - Update blockchain states
  - Store code, variables, and results of function call results
  - Public key and digital signatures are normally used to identify accounts and to ensure integrity and authorisation of transactions



# Blockchain Functionality

## Blockchain as a Computational Infrastructure

- Smart contracts: programs, typically small in size, on the blockchain (e.g. escrow)
- Architectural decision
  - Which parts of data and computation should be placed on-chain or kept off-chain since the amount computation power, data storage space and control of read accesses on a blockchain can be limited
  - A common practice: store hashed data, meta-data and small-sized public data on-chain and keep large and private data off-chain
- There are existing platforms (e.g. IPFS) that can be used for providing a data layer on top of blockchain
- Oracles: interacting with the external world

# Blockchain Functionality

## Blockchain as a Communication Mechanism

- Transactions that are announced get broadcast across the network
- Committing a transaction to the data structure may be required before regarding the transaction as final
  - ...or the receiver can already act based on the announcement

## Blockchain as an Asset Management and Control Mechanism

- Cryptocurrency and tokens are virtual assets
- Typically, only the owner can control them
- More complex models can be implemented, e.g.:
  - Multi-signature or threshold voting (more than one account controls an asset)
  - Escrow: smart contract code controls an asset
- Very customizable through smart contracts

# Blockchain Non-functional Properties



# Blockchain Non-functional Properties

- Immutability
  - If data is contained in a committed transaction, it will eventually become immutable
- Non-repudiation
  - The immutable chain of cryptographically-signed historical transactions provides non-repudiation of the stored data
- Integrity: cryptographic tools support data integrity
- Transparency: public access (on permission-less blockchains)
- Equal rights
  - Allows every participant the same ability to access and manipulate the blockchain
  - Trust: Achieved from the interactions between nodes within the network
  - Censorship resistance

# Blockchain Non-functional Properties

## Limitations

- Data privacy
  - No privileged users
  - Tradeoff between data privacy and transparency
- Scalability
  - Size of the data: due to the global replication of all data across the network
  - Transaction throughput: 3-20 transactions per second (tps)
    - VISA network handles 1700 tps on average
  - Latency of data transmission
    - Write latency caused by propagation
    - Number of transactions included in each block (1MB for bitcoin)
    - Latency between submission and confirmation caused by consensus protocol
- All addressed through various research and development
  - But there is no one-size-fits-all solution (yet?)

# Decentralization

## Deployment

---

Deployment Option	Fundamental properties	Impact			Flexibility
		Cost efficiency	Performance		
Public blockchain	⊕⊕⊕	⊕	⊕	⊕	⊕
Consortium/community blockchain	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Private blockchain	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

# Decentralization

## Permissions

- A blockchain may be permissioned: having one or more authorities act as a gate for participation
  - Permission to join the network, to initiate transactions, to mine, to create an asset etc.
  - Suitable in regulated industries. E.g. banks are required to establish real-world identity of transacting parties
  - Depends on the size of the network
  - Tradeoffs between permissioned and permission-less blockchain
    - Transaction processing rate, cost, flexibility, reversibility, etc.
  - Permission management mechanism may become a single point failure

# Anonymity

- Zero-knowledge proofs
  - Maintain a secure ledger
  - Private payment without disclosing the parties or amounts involved
  - E.g. Zcash encrypts the payment information in the transactions and uses a cryptographic method to allow verifying the encrypted transaction
- Mixing services
  - Groups several transactions together
    - Payment contains multiple input addresses and multiple output addresses
  - Examples: Monero, CoinJoin, Blindcoin, CoinSwap

# Core components of different types of blockchain

Public

## Permission-less

**Consensus:** Proof-of-X

**Permission management**

- Blockchain layer
- Application layer (optional)

**Incentive:** Blockchain layer

## Permissioned

**Consensus**

- Proof-of-X
- PBFT, Federated consensus, Round Robin etc.

**Permission management**

- Blockchain layer
- Application layer (optional)

**Incentive:**

- Blockchain layer
- Governance around permissions

Private

**Consensus**

- Proof-of-X
- PBFT, Federated consensus, Round Robin etc.

**Permission management:**

- Blockchain layer
- Network layer
- Application layer (optional)

**Incentive:** Governance around access

**Consensus**

- Proof-of-X
- PBFT , Federated consensus, Round Robin etc.

**Permission management:**

- Blockchain layer
- Network layer
- Application layer (optional)

**Incentive:** Governance around access permissions

# Non-functional properties of different types of blockchain

	Permission-less		Permissioned	
	Public	Private	Public	Private
Immutability	+++ (#Nodes, Consensus, Topology)		Immutability	++
Integrity	+++ (#Nodes, Consensus, Topology)		Integrity	++
Transparency	++ (Access control)		Transparency	++
Availability	+++ (#Nodes, Topology)		Availability	++
Performance	+ (Consensus, latency)		Performance	++
Cost Efficiency	+		Cost Efficiency	++
<hr/>				
Immutability	+		Immutability	+
Integrity	+		Integrity	+
Transparency	+		Transparency	+
Availability	+		Availability	+
Performance	+++		Performance	+++
Cost Efficiency	+++		Cost Efficiency	+++

# Blockchain Architecture Design



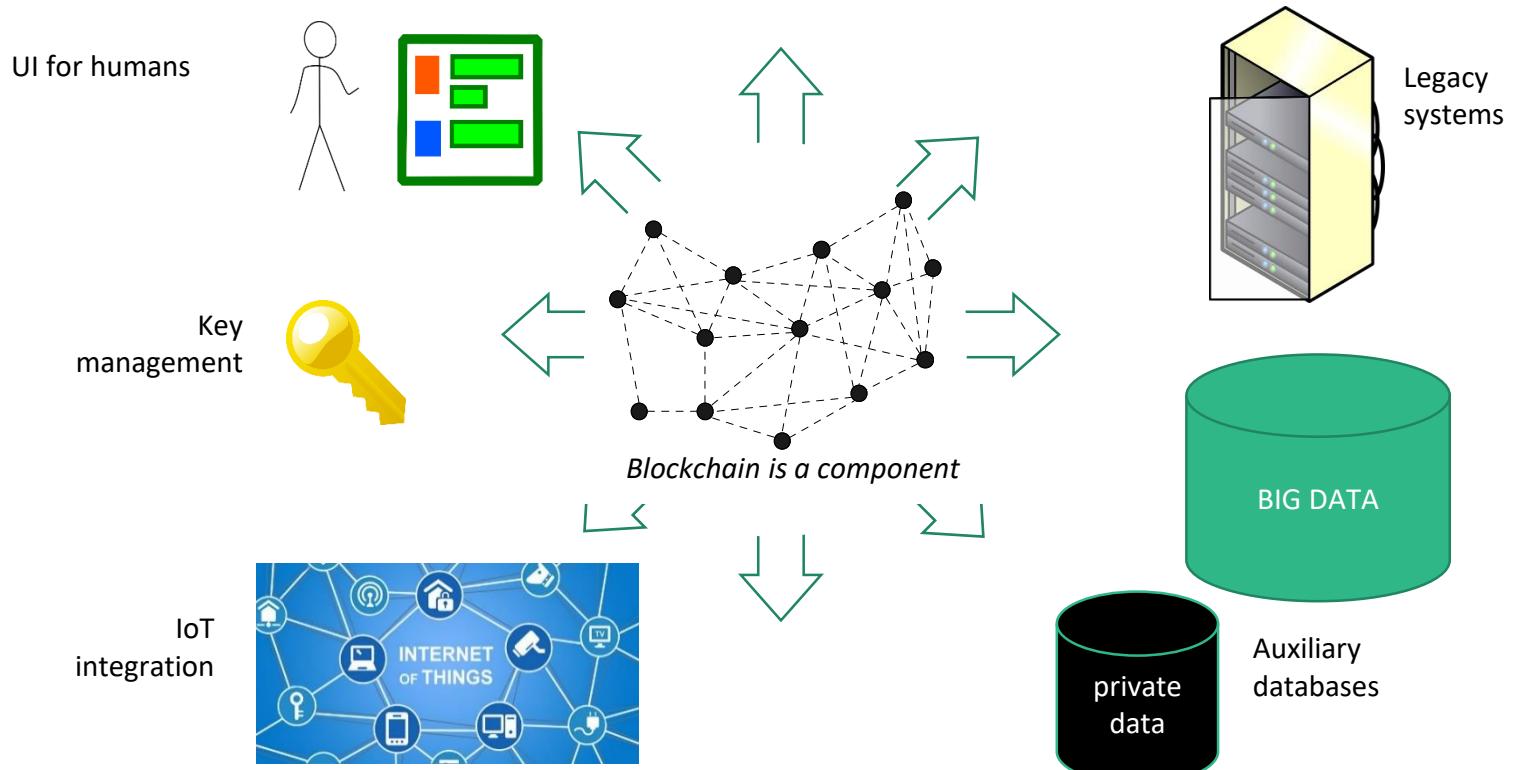
# Overview

- Many **interesting applications** for Blockchain
  - Basically of interest in most lack-of-trust settings where a distributed application can coordinate multiple parties
  - Examples:
    - Supply chains
    - Handling of titles, e.g., land, water, vehicles
    - Identity
  - Many startups and initiatives from enterprises / governments
- ... but also many **challenges**
  - When to use blockchain
  - Trade-offs in architecture
    - Downsides: cost, latency, confidentiality
    - What to handle on-chain, what off-chain?

# Relevant Topics in the Course

- Architecting applications on Blockchain:
  - Taxonomy and design process
  - Blockchain as an element in an architecture
    - Functional and non-functional properties of blockchain
  - Blockchain Patterns
  - Cost: how much will using blockchain cost?
  - Latency: simulation under changes
  - Security and Dependability
- Model-driven development of smart contracts
  - Business process execution
  - Model-based generation of registries and UIs

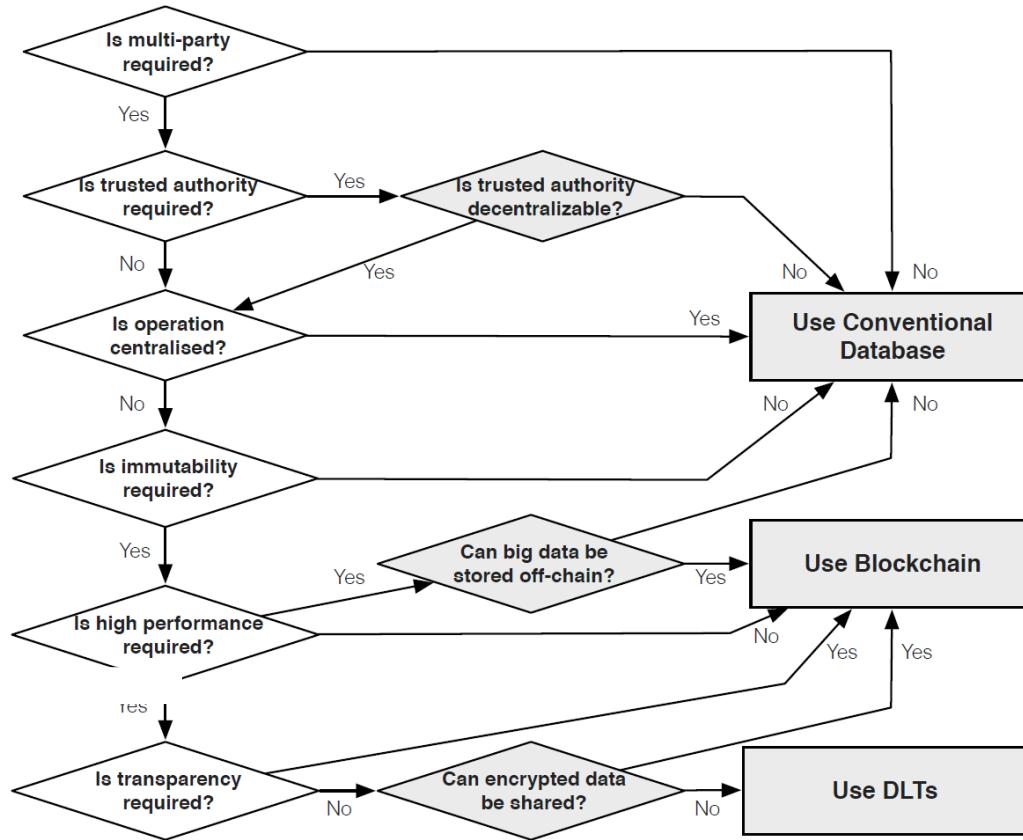
# Blockchains are Not Stand-Alone Systems



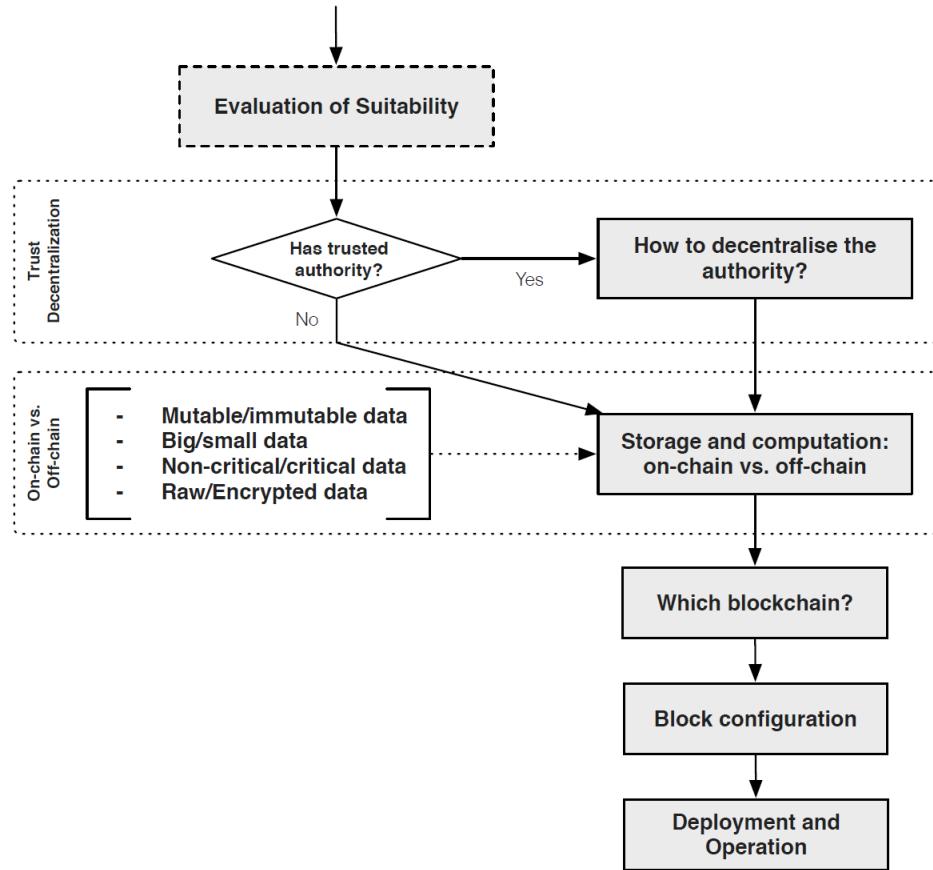
# Non-Functional Trade-Offs

- Compared to conventional database & script engines, blockchains have:
    - (-) Confidentiality, Privacy
    - (+) Integrity, Non-repudiation
    - (+ read/ - write) Availability
    - (-) Modifiability
    - (-) Throughput / Scalability / Big Data
    - (+ read/ - write) Latency
- Security: combination of CIA properties

# Evaluation of Suitability



# Design Process for Blockchain-Based Systems



# Taxonomy

Blockchain-related design decisions regarding (de)centralisation, with an indication of their relative impact on quality properties

Legend:  $\oplus$ : Less favourable,  $\oplus\oplus$ : Neutral,  $\oplus\oplus\oplus$ : More favourable

Design Decision	Option	Impact				#Failure points
		Fundamental properties	Cost efficiency	Performance		
Fully Centralised	Services with a single provider (e.g., governments, courts)	$\oplus$	$\oplus\oplus\oplus$	$\oplus\oplus\oplus$	1	
	Services with alternative providers (e.g., banking, online payments, cloud services)					
Partially Centralised & Partially Decentralised	Permissioned blockchain with permissions for fine-grained operations on the transaction level (e.g., permission to create assets)	$\oplus\oplus$	$\oplus\oplus$	$\oplus\oplus$	*	
	Permissioned blockchain with permissioned miners (write), but permission-less normal nodes (read)					
Fully Decentralised	Permission-less blockchain	$\oplus\oplus\oplus$	$\oplus$	$\oplus$	Majority (nodes, power, stake)	
Verifier		Fundamental properties	Cost efficiency	Performance	#Failure points	
		$\oplus\oplus$	$\oplus\oplus$	$\oplus\oplus$	1	
		$\oplus\oplus\oplus$	$\oplus$	$\oplus$	M	
	Ad hoc verifier trusted by the participants involved	$\oplus$	$\oplus\oplus\oplus$	$\oplus\oplus$	1 (per ad hoc choice)	

Also consider skills available for a specific platform

# Taxonomy

Blockchain-related design decisions regarding storage and computation, with an indication of their relative impact on quality properties

Design Decision		Option	Fundamental properties	Impact		
				Cost efficiency	Performance	Flexibility
Item data	On-chain	Embedded in transaction (Bitcoin)	⊕⊕⊕⊕	⊕	⊕	⊕⊕
		Embedded in transaction (Public Ethereum)		⊕⊕⊕⊕	⊕	⊕⊕⊕
		Smart contract variable (Public Ethereum)		⊕⊕	⊕⊕⊕	⊕
		Smart contract log event (Public Ethereum)		⊕⊕⊕	⊕⊕	⊕⊕
	Off-chain	Private / Third party cloud	⊕	~KB Negligible	⊕⊕⊕⊕	⊕⊕⊕⊕
		Peer-to-Peer system		⊕⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
Item collection	On-chain	Smart contract	⊕⊕⊕⊕	⊕⊕⊕⊕ (public)	⊕⊕⊕⊕	⊕
		Separate chain		⊕ (public)	⊕	⊕⊕⊕⊕
Computation	On-chain	Transaction constraints	⊕⊕⊕⊕	⊕	⊕	⊕
		Smart contract				
	Off-chain	Private / Third party cloud	⊕	⊕⊕⊕⊕	⊕⊕⊕⊕	⊕⊕⊕⊕

# Taxonomy

Blockchain-related design decisions regarding blockchain configuration

Design Decision	Option	Impact			
		Fundamental properties	Cost efficiency	Performance	Flexibility
Blockchain scope	Public blockchain	⊕⊕⊕	⊕	⊕	⊕
	Consortium/community blockchain	⊕⊕	⊕⊕	⊕⊕	⊕⊕
	Private blockchain	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
Data structure	Blockchain	⊕⊕⊕	⊕	⊕	⊕
	GHOST	⊕⊕	⊕⊕	⊕⊕	⊕
	BlockDAG	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
	Segregated witness	⊕⊕⊕	⊕⊕	⊕	⊕
Consensus Protocol	Proof-of-work	⊕⊕⊕	⊕	⊕	⊕
	Proof-of-retrievability	⊕⊕⊕	⊕	⊕	⊕
	Proof-of-stake	⊕⊕	⊕⊕	⊕⊕	⊕⊕⊕
	BFT (Byzantine Fault Tolerance)	⊕	⊕⊕⊕	⊕⊕⊕	⊕
Protocol Configuration	Bitcoin-NG	⊕⊕⊕	⊕	⊕	⊕
	Off-chain transaction protocol	⊕	⊕⊕⊕	⊕⊕	⊕⊕⊕
	Mini-blockchain	⊕⊕	⊕⊕	⊕	⊕⊕
Protocol Configuration	Security-wise	X-block confirmation	⊕	⊕	⊕⊕⊕
	Scalability-wise	Checkpointing	⊕⊕⊕	⊕⊕⊕	⊕
	Scalability-wise	Original block size and frequency	⊕⊕⊕	n/a	⊕
New blockchain	Scalability-wise	Increase block size / Decrease mining time	⊕	n/a	n/a
	Security-wise	Merged mining	⊕⊕⊕	⊕⊕	⊕
	Security-wise	Hook popular blockchain at transaction level	⊕⊕	⊕	⊕⊕⊕
	Scalability-wise	Proof-of-burn	⊕	⊕	⊕⊕
	Scalability-wise	Side-chains	⊕⊕⊕	⊕	⊕
	Scalability-wise	Multiple private blockchains	⊕	⊕⊕⊕	⊕⊕⊕

## Summary of today:

- Part 1: Course Summary

- Lecturers and Tutor
- Learning Outcomes, Course Outline, Assessments

- Part 2: Topic Overview

- What is Blockchain, and Why Does it Matter?
- Blockchain-based Applications
- Blockchain Functionality
- Blockchain Non-functional Properties
- Blockchain Architecture Design

# Course Outline – next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
1st	18 Feb	Ingo Weber	Introduction	1. Introduction	
2nd	25 Feb	Ingo Weber	Existing Blockchain Platforms	4. Example use cases 2. Existing Blockchain Platforms (1h on smart contract dev)	Assignment 1 out (Monday before lecture)
3rd	4 Mar	Sherry Xu	Blockchain in Software Architecture 1	3. Varieties of blockchain 5. Blockchain in Software Architecture (including software architecture basics) 1/2	

Note: **tutorials** run from week 2 to week 9.  
They are not mandatory, but helpful,  
especially for completing the assignments.



# End of Lecture / Consultation

Ingo Weber | Principal Research Scientist & Team Leader

Architecture & Analytics Platforms (AAP) team

[ingo.weber@data61.csiro.au](mailto:ingo.weber@data61.csiro.au)

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

[www.data61.csiro.au](http://www.data61.csiro.au)



# COMP6452 Lecture 2: Existing Blockchain Platforms

Ingo Weber | Principal Research Scientist & Team Leader  
Architecture & Analytics Platforms (AAP) team  
[ingo.weber@data61.csiro.au](mailto:ingo.weber@data61.csiro.au)

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

[www.data61.csiro.au](http://www.data61.csiro.au)

# Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development

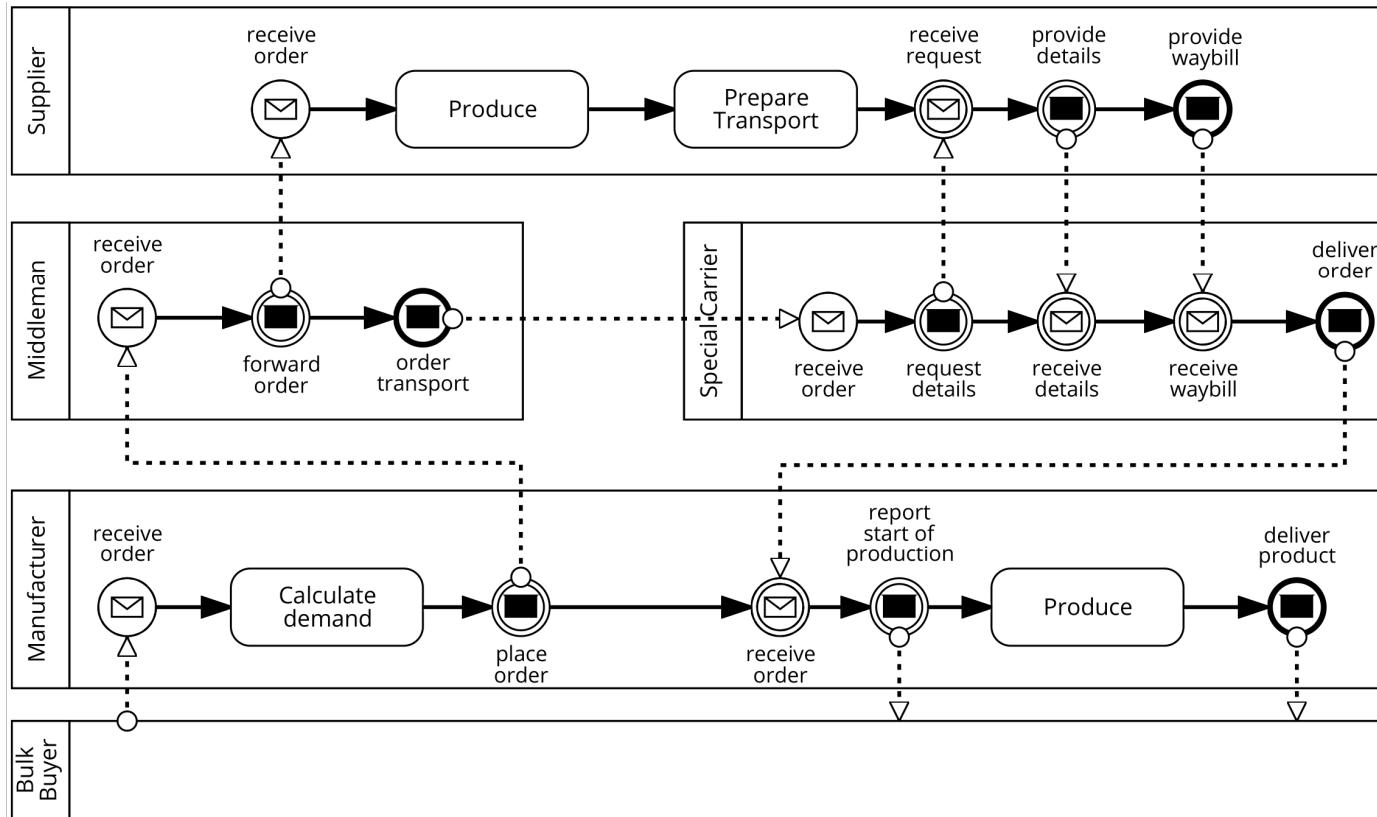


# Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development



# Motivation: example



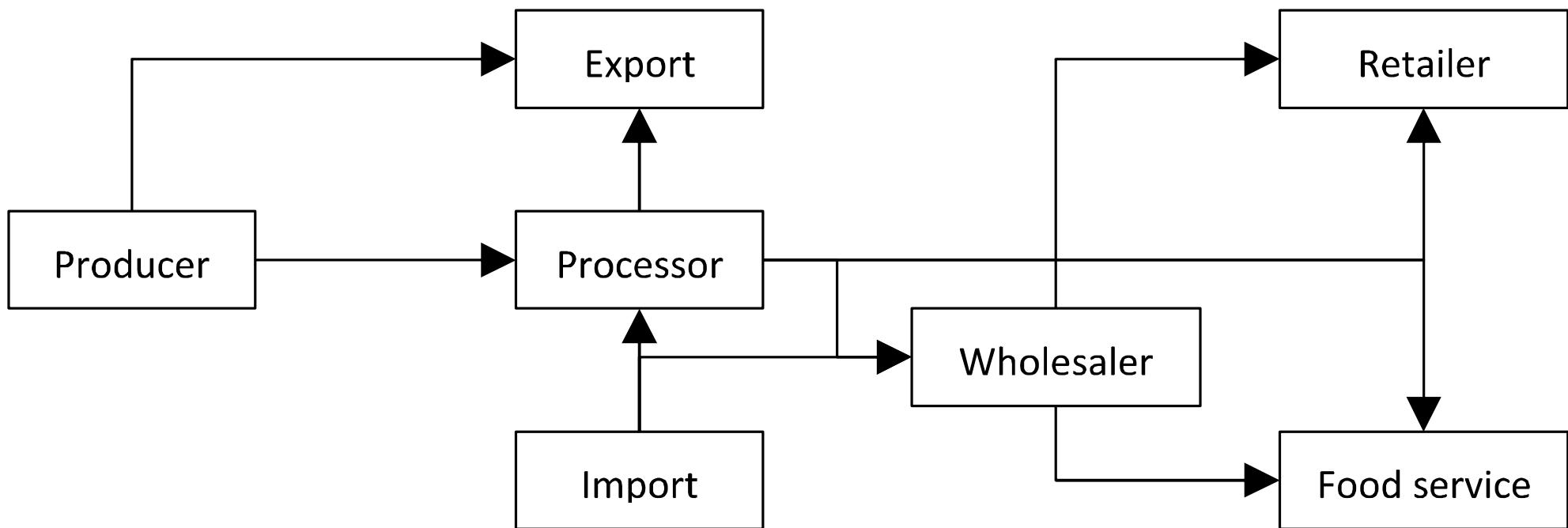
Issues:

- Knowing the status, tracking correct execution
- Handling payments
- Resolving conflicts

→ Trusted 3rd party?  
→ Blockchain!

# Use Case: Supply Chains

## Sample Agricultural Supply Chain Network

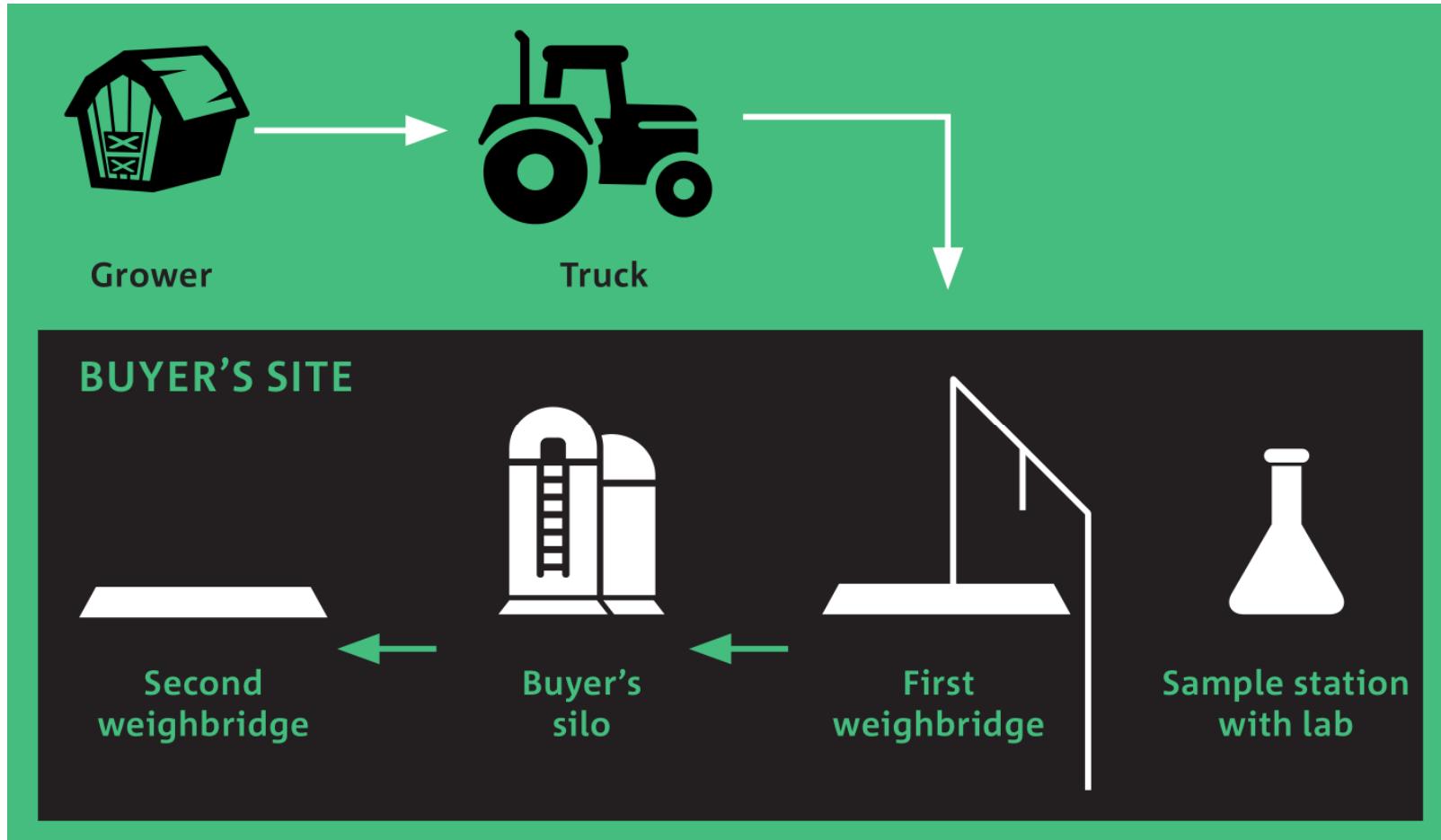


# Using Blockchain for Supply Chains

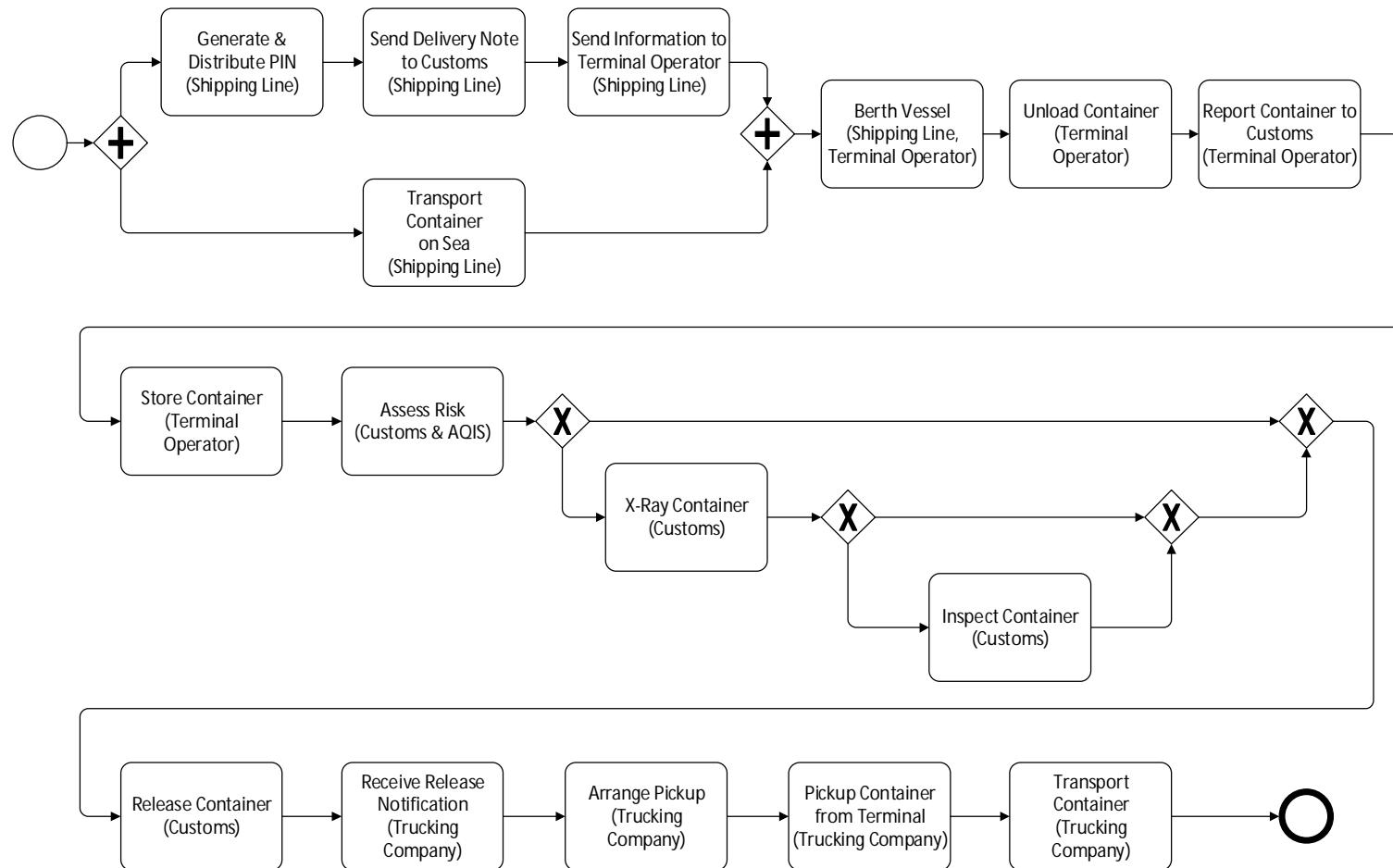
- Why?
  - Irrefutable, tamper-proof data store
    - Prevent or detect counterfeiting
  - Smart contracts can check integrity and authorization / authentication
  - Can solve other problems:
    - Counter-party risks
    - Lack of trust, e.g., in competition
    - Supply chain transparency
  - ...
- How?
  - Record supply chain events on blockchain, e.g.:
    - GS1 EPCIS events
    - Other tag scans
    - Phytosanitary certificates
  - Check that event sequences are correct, e.g. through
    1. Process conformance
    2. Business rules adherence
    - Can be on-chain or off-chain
    - Regulatory compliance

# Example: AgriDigital's first pilot

see <https://www.data61.csiro.au/blockchain> / Chapter 12 in the book



# Example: Sea Import to Australia



# Some Benefits of using Blockchain in Supply Chains

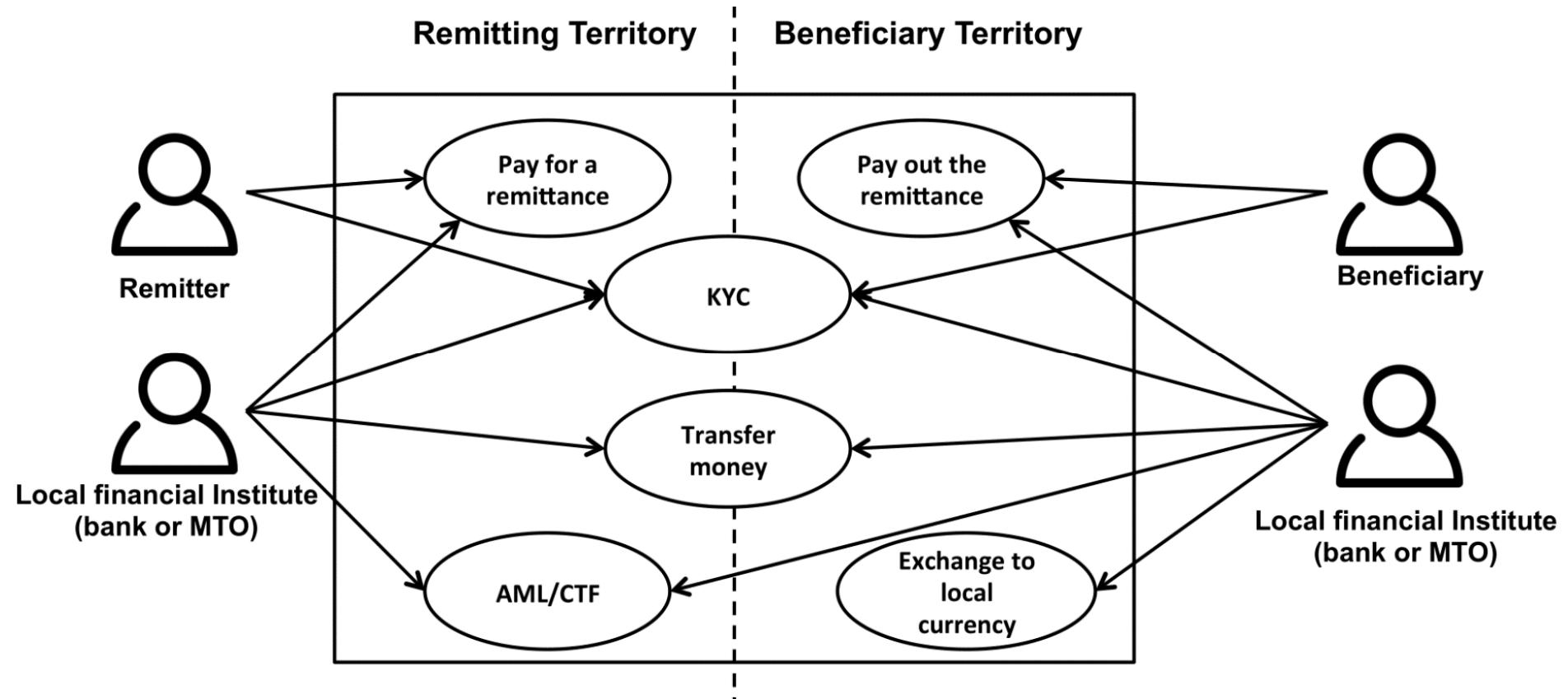
- Electronic titles to supply chain goods
  - Ensure ownership, right to sell, etc are handled correctly
  - Reduce financial risk – e.g.: if a buyer goes bankrupt before paying for the goods, the seller still owns them
- Establish identity and authenticity for:
  - Requester
  - Other relevant supply chain participants
- Check financial record / trustworthiness
- Ensure correctness of specific supply chain documents
  - E.g., invoice, purchase order, ...

# Use Case: International Money Transfers

- Many workers in Australia regularly send money back to their families overseas
  - Up to 10% of GDP in some developing countries (and even 27% in Tonga and 20% in Samoa)
  - High remittance costs have serious effects in these countries
- Remittance costs in Pacific Island countries are among the highest in the world
  - For example, to send \$200 from Australia to Vanuatu costs \$33.20 and \$28.60 to Samoa
- Issues:
  - Many parties involved, sometimes little transparency
  - Difficulties of satisfying AML/CTF (Anti-Money Laundering/Counter-Terrorism Financing) regulation, especially where the receiving party may not have a bank account.
  - High latency, with transaction times up to 5 days.

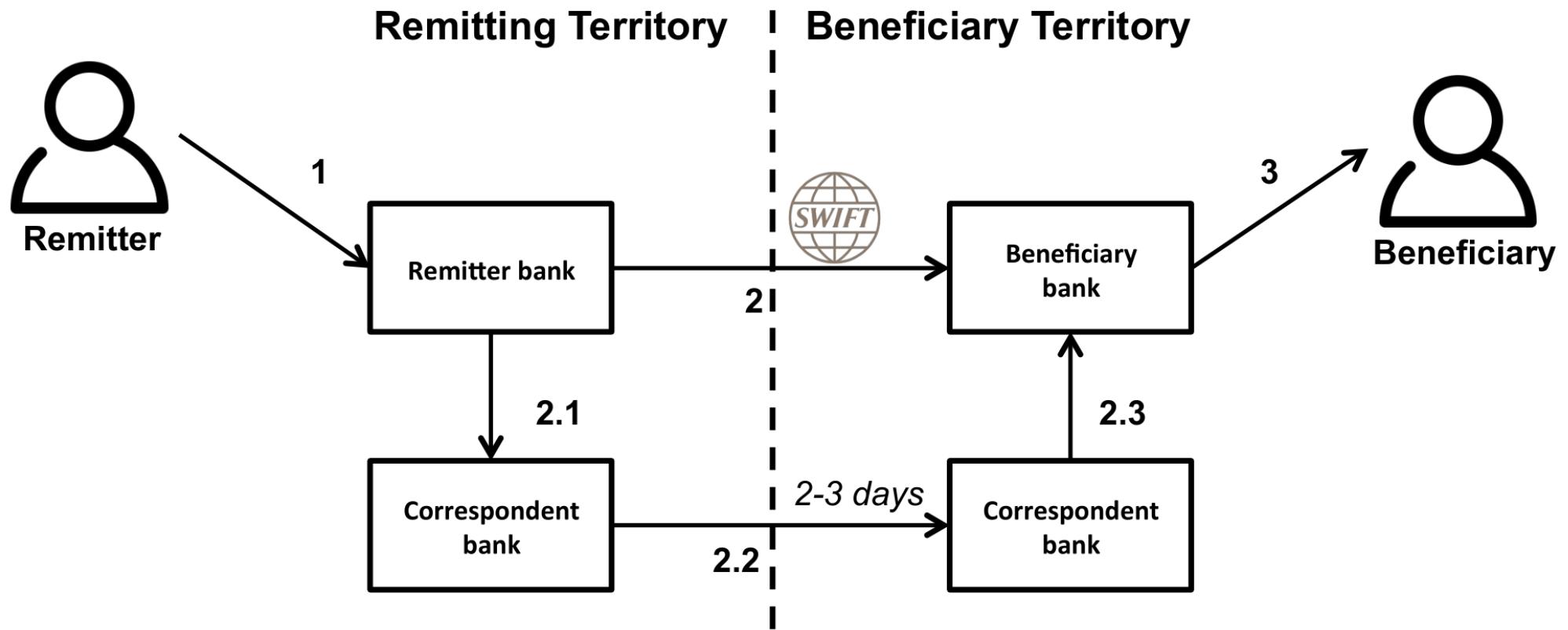
# Use Case: International Money Transfers

## Stakeholders and Functions



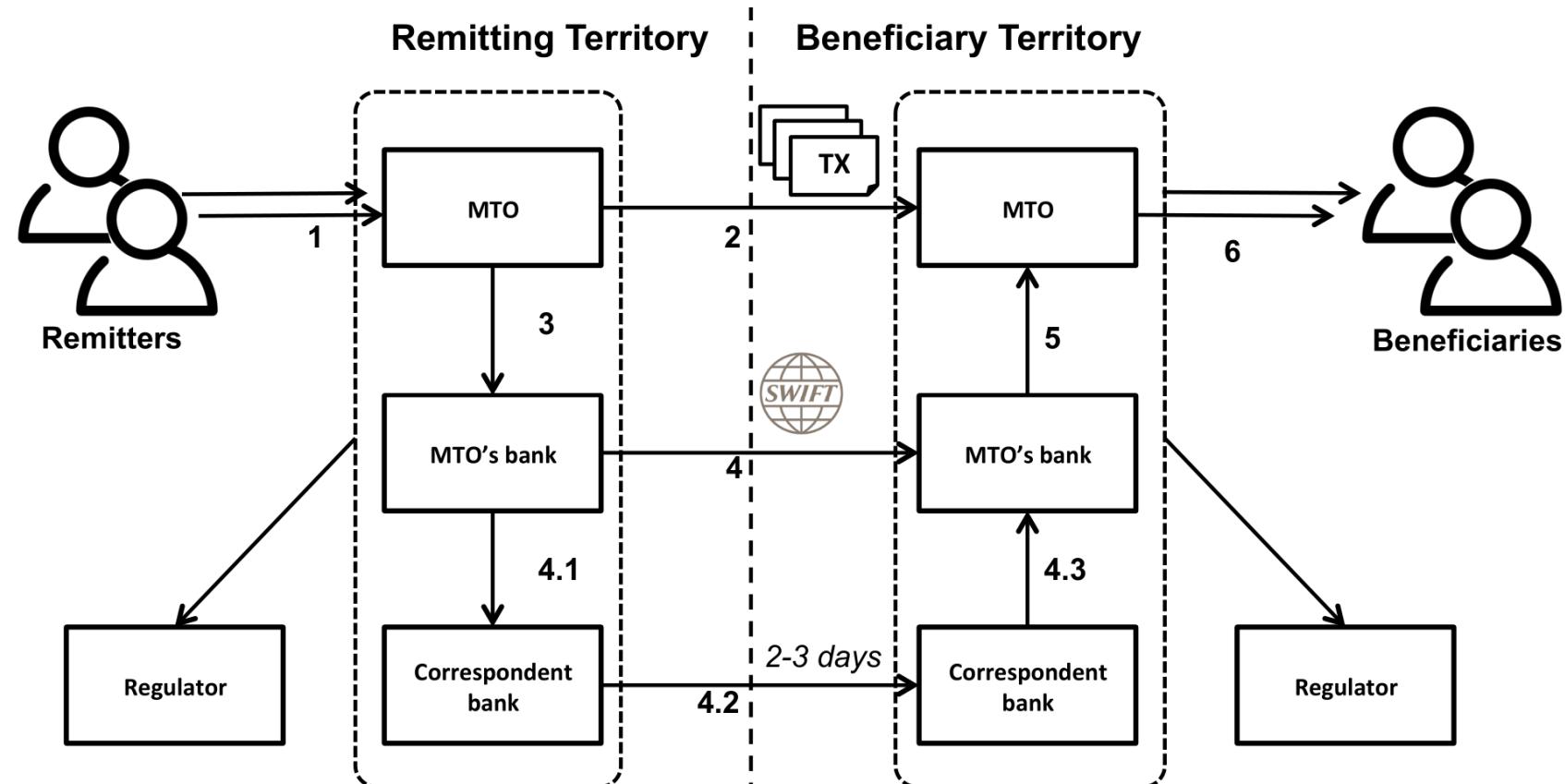
# Use Case: International Money Transfers

## Remittance through banks



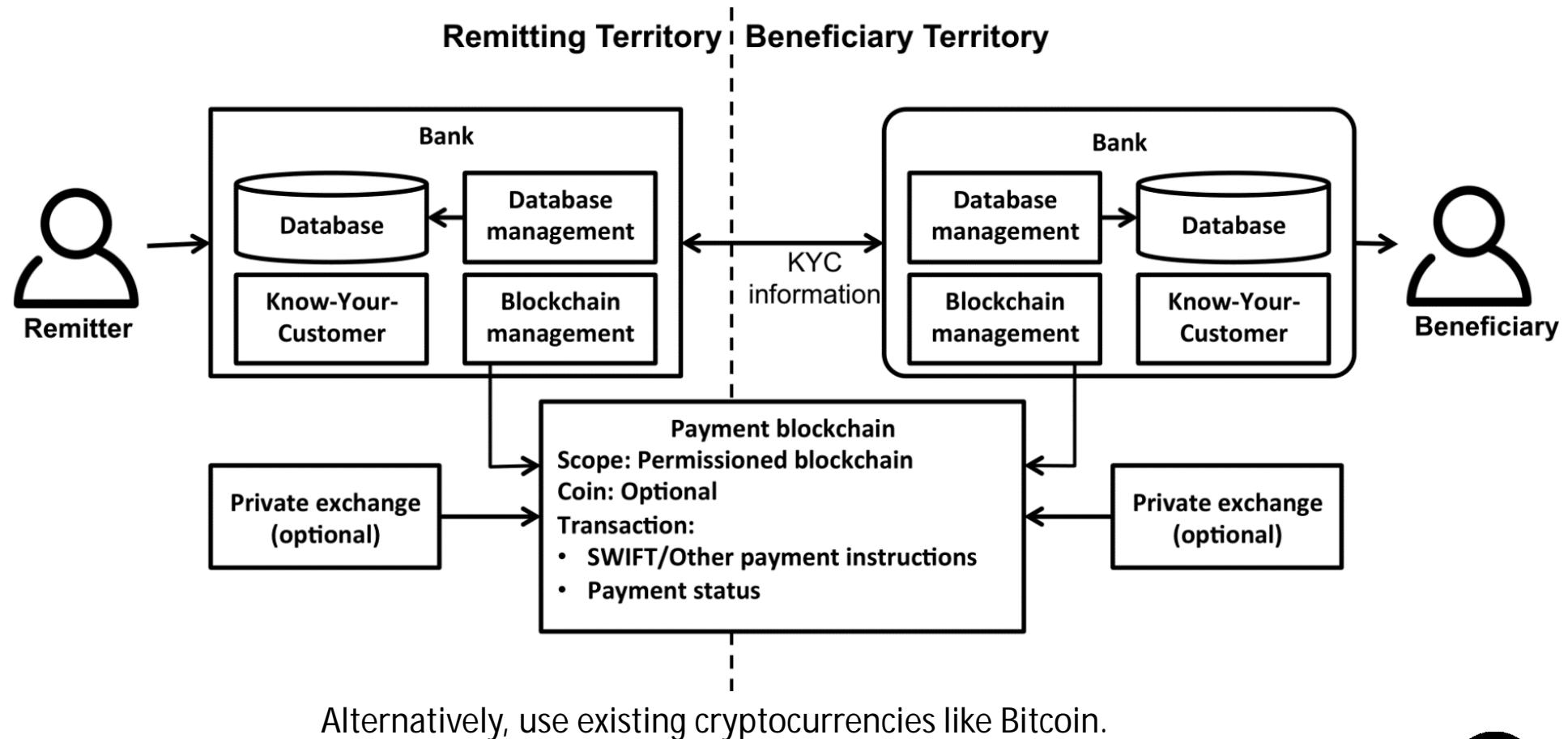
# Use Case: International Money Transfers

## Remittance through a Money Transfer Operator (MTO)



# Use Case: International Money Transfers

## Remittance through Blockchain



# Intl. Money Transfers Non-functional Properties

- Transaction Latency:
  - From days (conventional) to hours or minutes (with blockchain)
- Cost:
  - Depends on the fees charged by various parties – but more parties are involved in the conventional designs
- Transparency:
  - Greater in the blockchain setting; foreign exchange rates might still be unfavourable for the customers
- Barriers to Entry
  - Conventional design requires participants to have banking / financial services licenses, and business relationships with correspondent banks
  - Public blockchains have low barriers to entry, but local regulation still applies to end-points within countries

# Agenda:

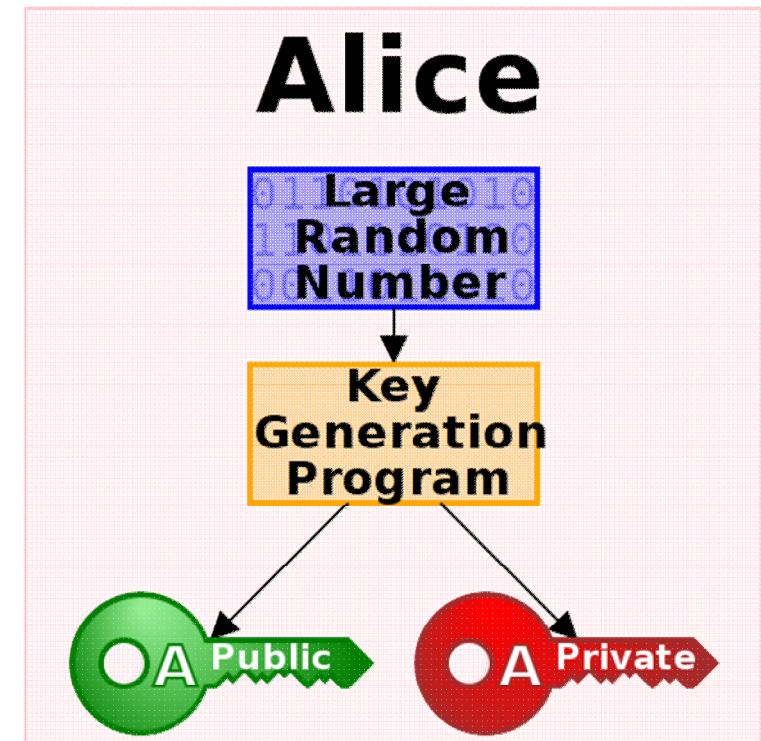
- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development



# Cryptography basics: public-key cryptography (1)

## Overview

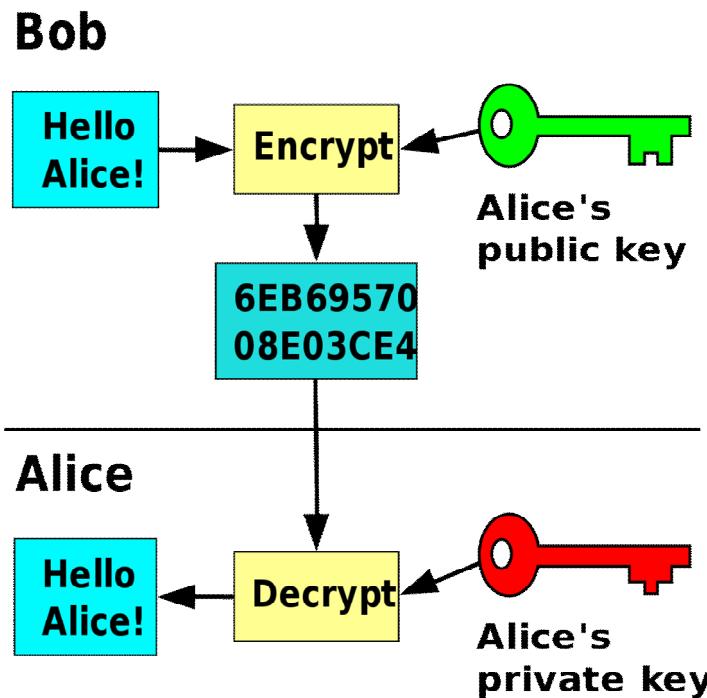
- Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses *pairs of keys*:
  - public keys which may be disseminated widely;
  - private keys which are known only to the owner.
- Effective security only requires keeping the private key private
- Easy to create new key pairs
- Used heavily in blockchain
  - Losing your private key can mean loss of assets
  - If hackers can get your private key, they can steal your assets



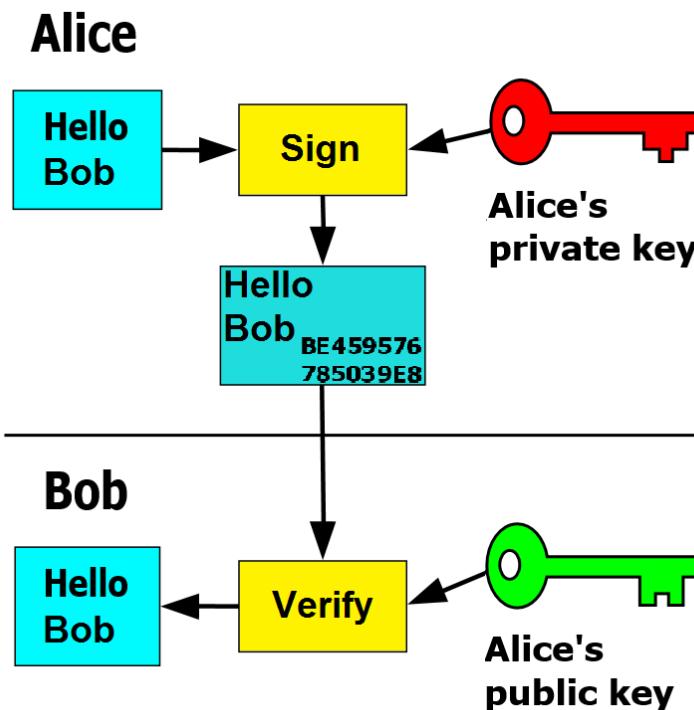
Some content from [https://en.wikipedia.org/wiki/Public-key\\_cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography)

# Cryptography basics: public-key cryptography (2)

## Encryption and digital signatures



Only Alice can decrypt the message



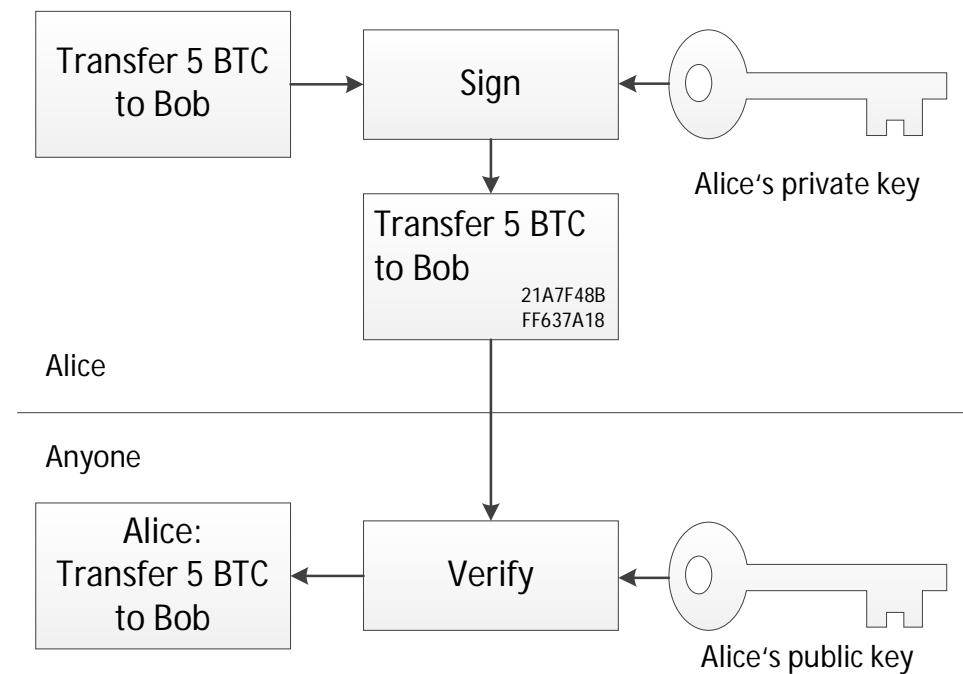
No-one can change the message without breaking Alice's signature

Some content from [https://en.wikipedia.org/wiki/Public-key\\_cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography)

# Cryptography basics: public-key cryptography (3)

## Use in Blockchain

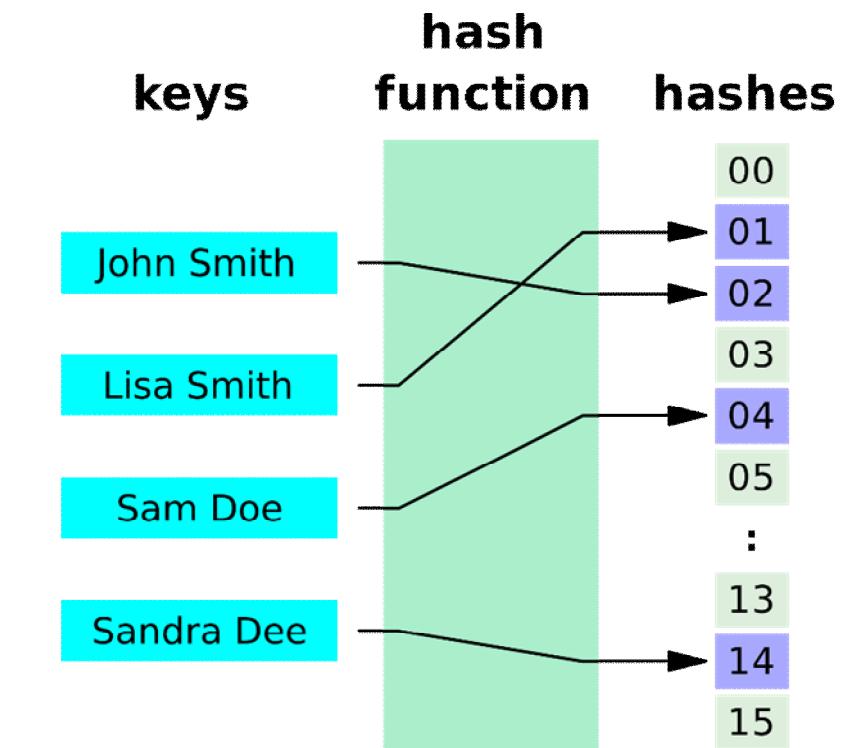
- Control over accounts: by private key
  - Control means the ability to act on behalf of the account, like spending the assets it owns
- In fact: each account is known by its public key
  - "Alice" here is really 0x7a2f16dab8b5c2cf99c35e4c6a5beb45c7df8f87
  - For some accounts, we may know the person / organization owning it
  - But by default, we don't



Some content from [https://en.wikipedia.org/wiki/Public-key\\_cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography)

# Cryptography basics: hash functions (1)

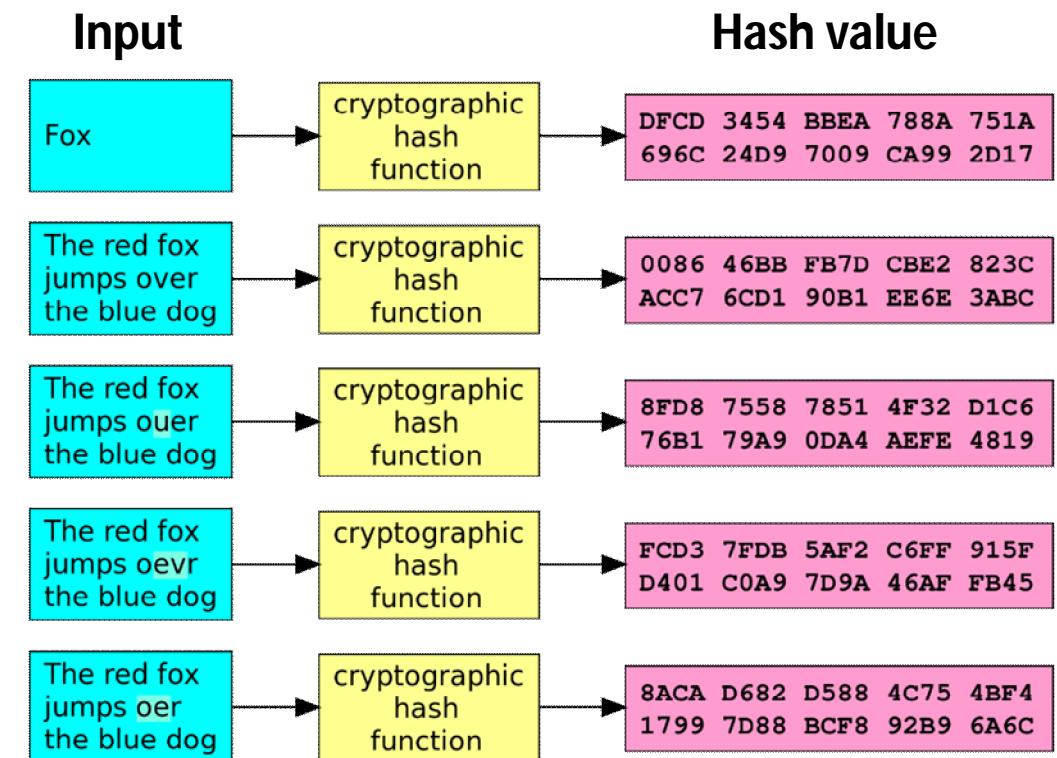
- A hash function is any function that can be used to map data of arbitrary size to data of a fixed size.
  - Values returned by a hash function are called hash values, hash codes, digests, or simply hashes.



Some content from [https://en.wikipedia.org/wiki/Hash\\_function](https://en.wikipedia.org/wiki/Hash_function) and [https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)

# Cryptography basics: hash functions (2)

- A **cryptographic hash function** is a special class of hash function with certain properties:
  - a one-way function: a function which is infeasible to invert.
  - deterministic: the same message always results in the same hash
  - quick to compute hash value for any message
  - it is infeasible to generate a message from its hash value except by trying all possible messages
  - a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
  - it is infeasible to find two different messages with the same hash value



Some content from [https://en.wikipedia.org/wiki/Hash\\_function](https://en.wikipedia.org/wiki/Hash_function) and [https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)

# Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development



# Bitcoin

## First Cryptocurrency

- A cryptocurrency operated on a p2p blockchain network
- 3 main types of participants
  - Users with wallets which contain keys to authenticate the transaction sent by the user using digital signature
  - Miners for producing blocks which store the validated transactions
  - Exchanges where users can trade bitcoin with other currencies

*Inventor: Satoshi Nakamoto (American? European? Denis Wright? Hal Finney?)*



*Source: Andreas M. Antonopoulos, Mastering Bitcoin-Unlocking Digital Cryptocurrencies*

# Bitcoin

## Network Distribution

### GLOBAL BITCOIN NODES

#### DISTRIBUTION

Reachable nodes as of Thu Nov 22 2018  
11:36:15 GMT+1100 (Australian Eastern Daylight Time).

**10168 NODES**

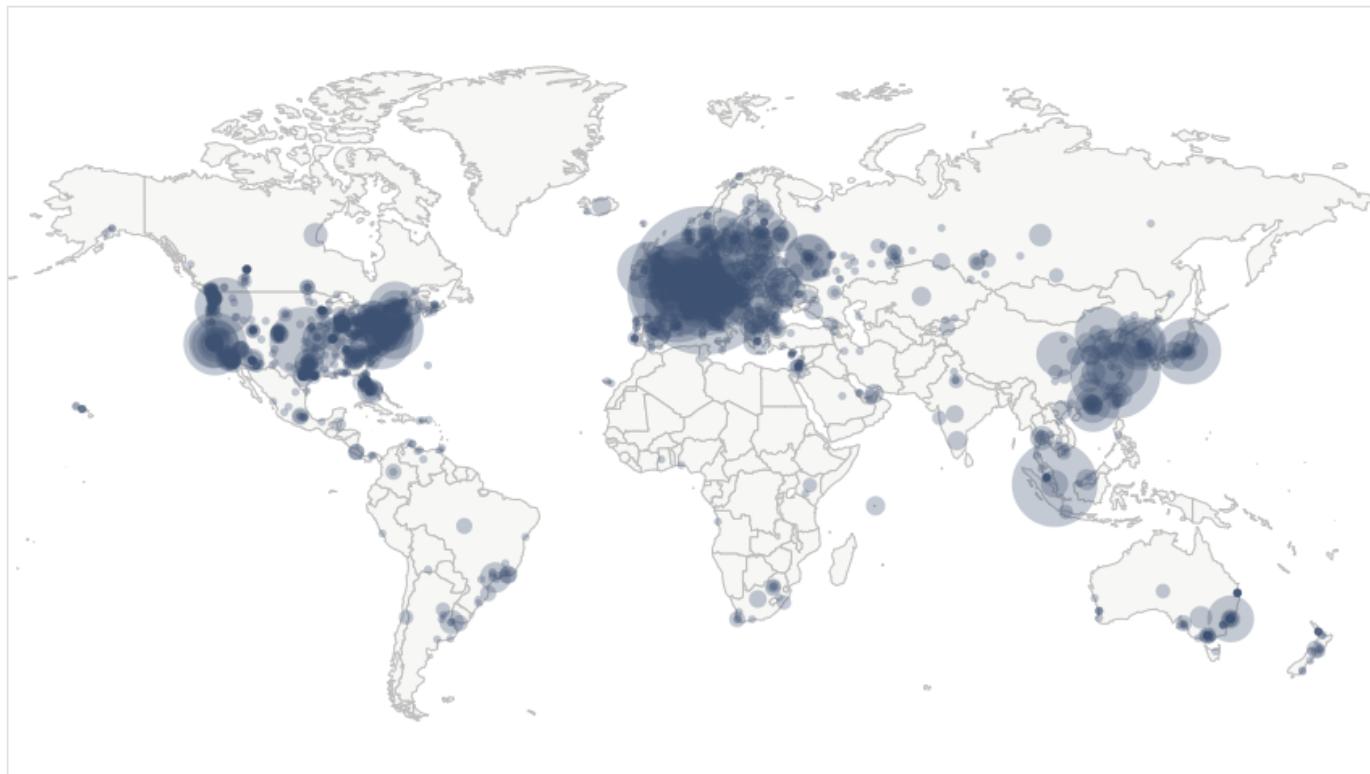
[24-hour charts »](#)

Top 10 countries with their respective number of reachable nodes are as follow.

RANK	COUNTRY	NODES
1	United States	2375 (23.36%)
2	Germany	1936 (19.04%)
3	France	667 (6.56%)
4	China	630 (6.20%)
5	Netherlands	493 (4.85%)
6	n/a	482 (4.74%)
7	Canada	367 (3.61%)
8	United Kingdom	331 (3.26%)
9	Singapore	266 (2.62%)
10	Russian Federation	259 (2.55%)

[More \(102\) »](#)

Source: [bitnodes.21.co/](http://bitnodes.21.co/)



# Bitcoin

## Exchange rate to US\$

- Lowest: \$0.05
- Highest: ~\$20,000
- Now: below  
~\$4000
- Not stable



Source: [99bitcoins.com/price-chart-history/](https://99bitcoins.com/price-chart-history/)

# Bitcoin

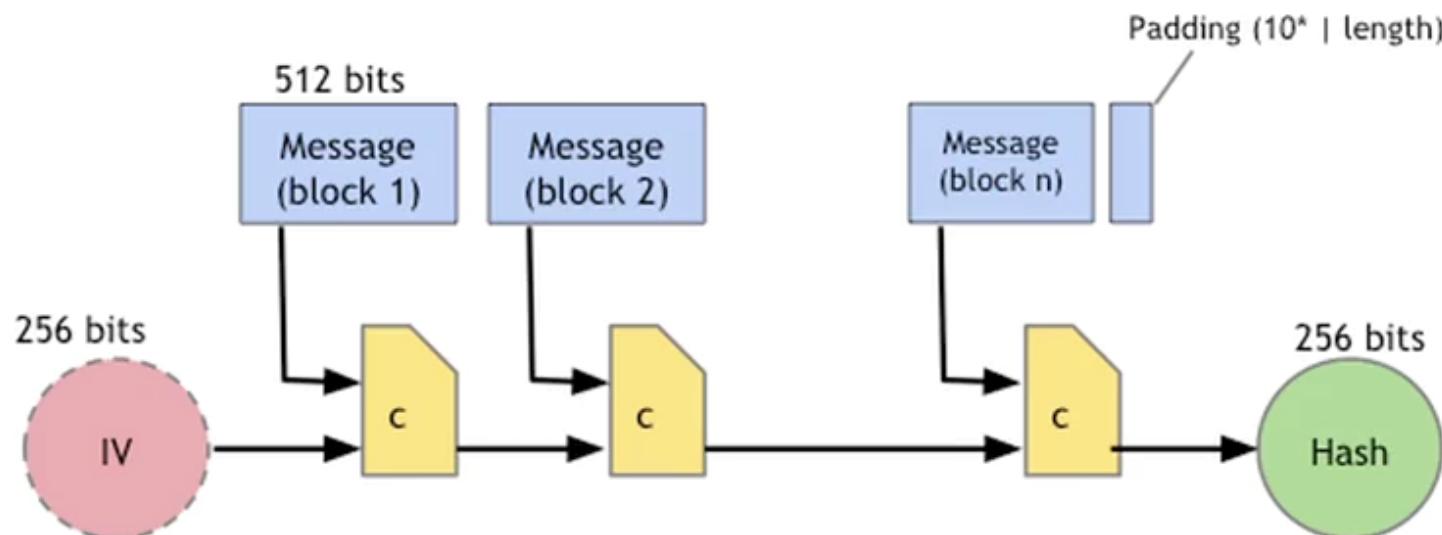
## Deflationary Cryptocurrency

- Total supply: 21 million
- New BTCs are issued during the mining process
  - New BTCs are rewarded to the miner who create the valid block.
  - The reward is halving every 210,000 blocks.
  - Reward (since 2016): 12.5 BTC (initially, 50 BTC)
  - Minting in Bitcoin will run out in 2140, when no more new BTC being issued.

# Bitcoin

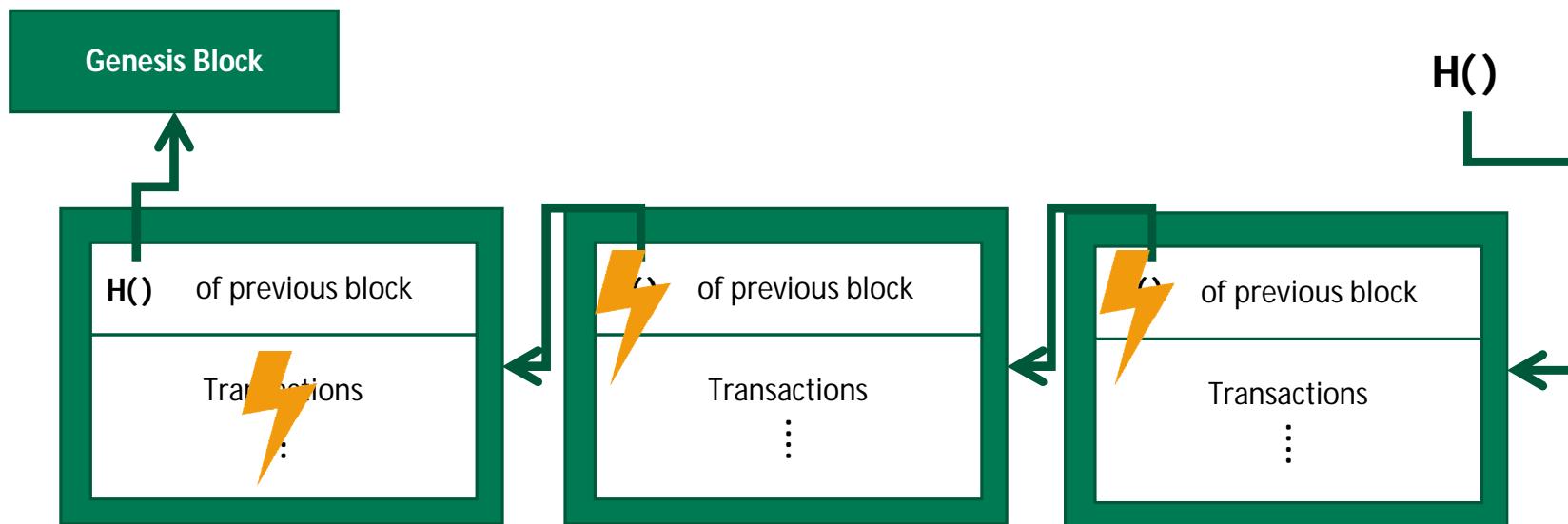
## SHA-256 Hash Function

- Hash function
  - Map data of arbitrary size to data of fixed size
  - One-way function –easy to compute but hard to invert –collision free



# Bitcoin

## Linked list with hash pointer

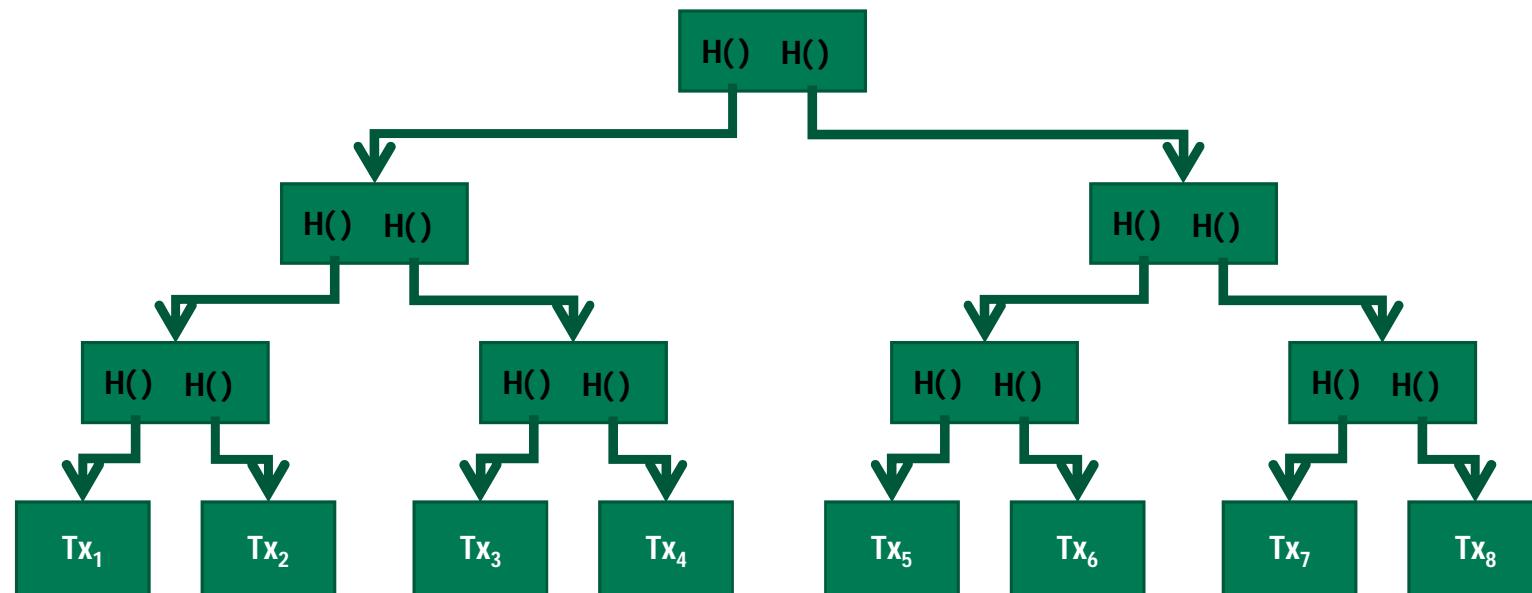


- Time between blocks, called *inter-block time*, is on average 10 minutes
  - But variation of times is high

# Bitcoin

## Block – Container of Transactions

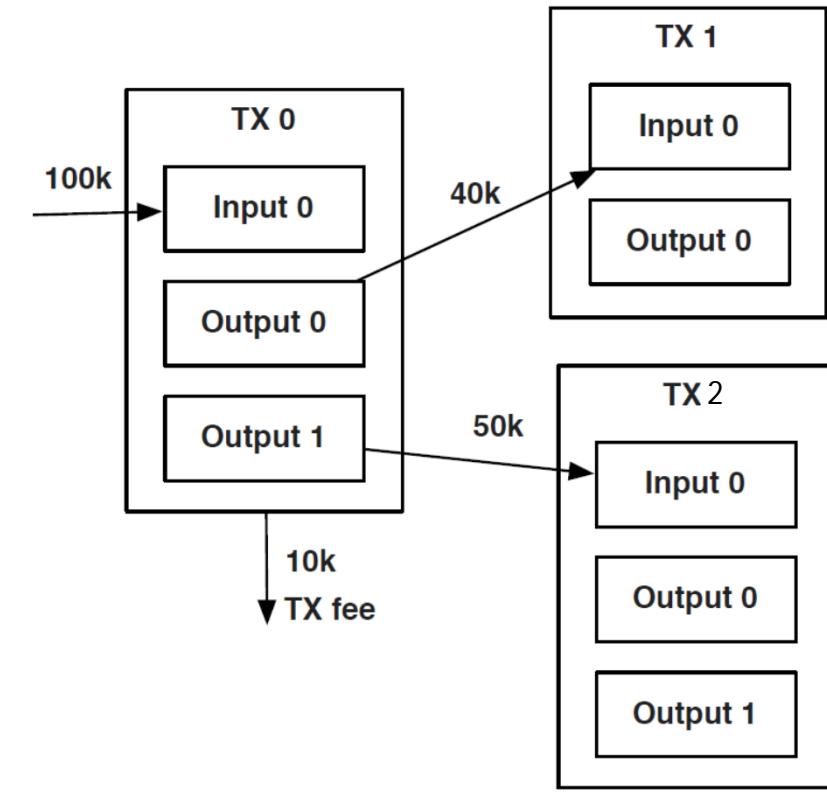
- Merkle Tree: Binary tree built using the hash function



# Bitcoin

## Transactions

- Transfer currency from source addresses to destination addresses
- Contains one or more inputs and one or more outputs
  - If the sum of the outputs is less than the sum of the inputs, the difference is a fee to the miner
    - Transaction fee is an incentive for miners to contribute computing power
    - The only variable that client software ask users to choose when create a transaction
- Contains proof of ownership for each input, in the form of a digital signature from the owner

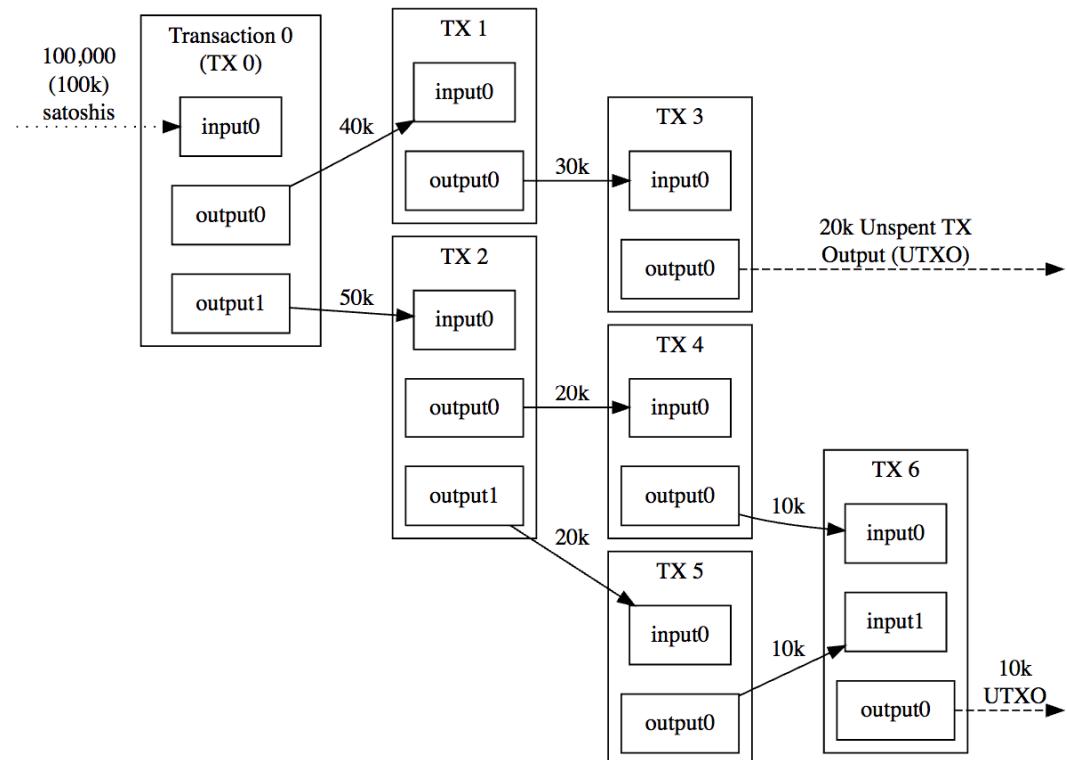


How transactions are linked

# Bitcoin

## Transactions

- Linked Transactions
  - Outputs of transactions become inputs of new transactions
  - Bitcoin addresses don't contain "coins"
    - Store unspent transaction outputs (UTXO)
  - "Address balance": the sum of all of the values of UTXOs associated with the address
    - Bit"coin" is misleading, because fractional ownership (e.g., 1.64 BTC) is the norm



# Bitcoin

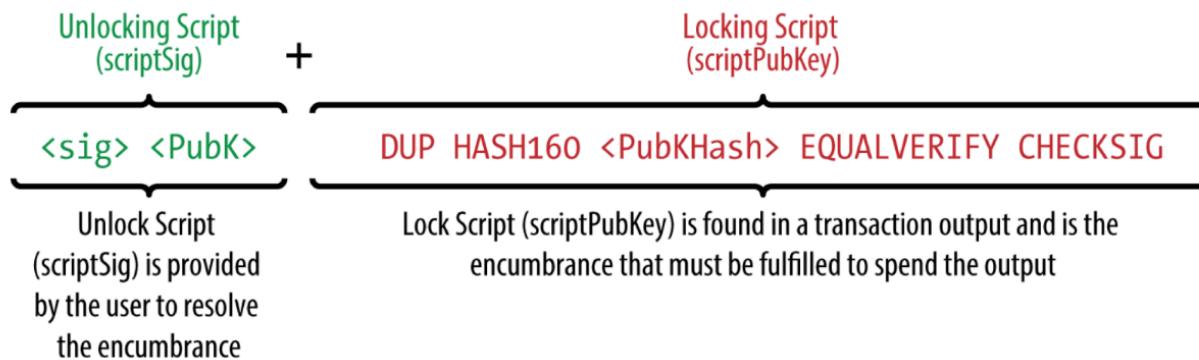
## Transactions

- Transactions delay factors
  - Transaction fee
    - Miners tend to optimize block creation by preferring transactions with higher fees
  - Transaction announcement happens during mining of next block
    - Mining will not be restarted, so new transactions can only be included in the block after the next one → theoretical inclusion delay is 1.5x inter-block time
  - Orphan
    - Transactions must arrive in order for a miner to process them fast
    - If the referenced input transactions, called parents, are as-yet unknown, a miner will delay the inclusion of the new transaction — it is then a so-called orphan
    - Miners may choose to keep orphans in the mempool while waiting for the parent transactions to arrive, but they also remove orphans after a time-out they choose
  - Locktimes
    - A transaction can contain a parameter declaring it invalid until the block with a certain sequence number has been mined

# Bitcoin

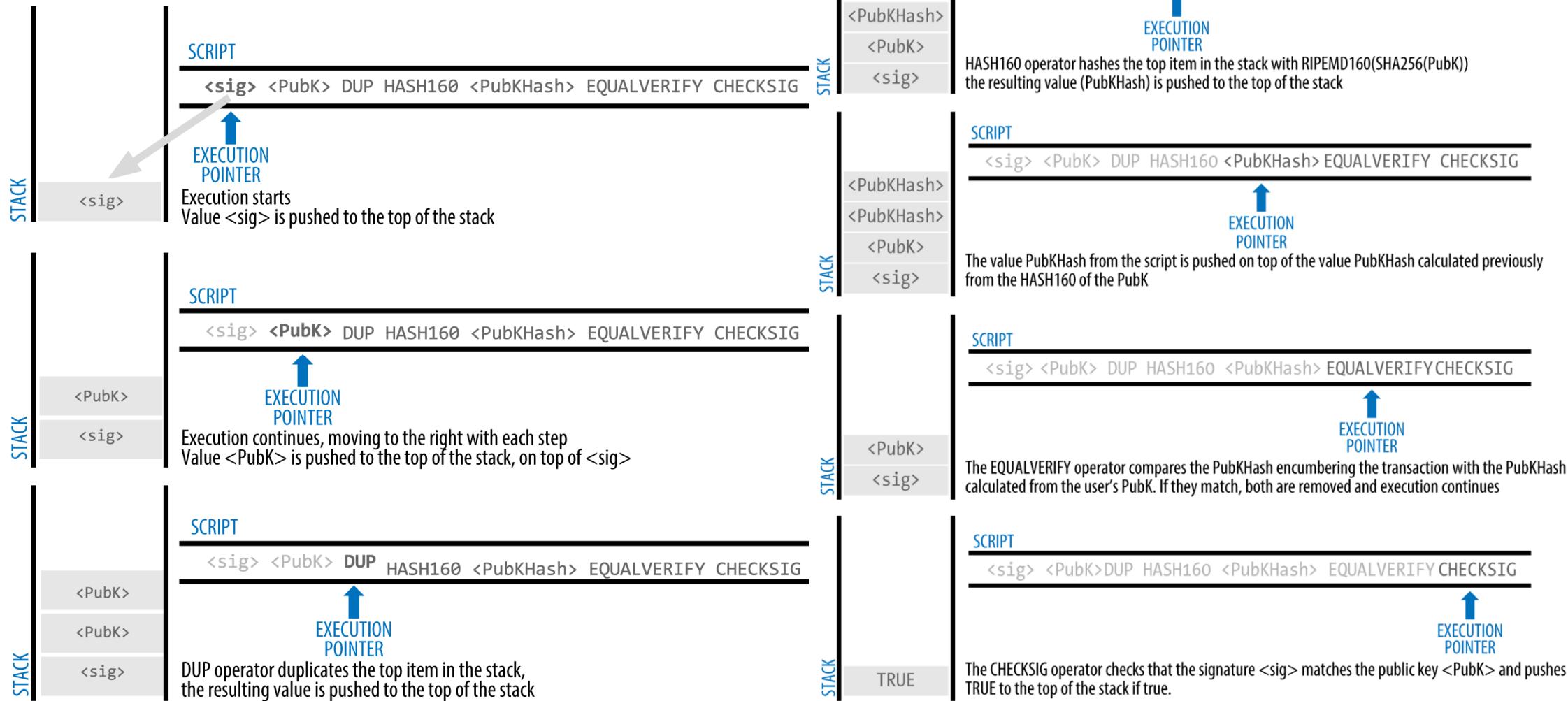
## Script

- Bitcoin uses a scripting system for transactions
  - Transaction validation relies on two types of script
    - A locking script in an output specifies the conditions for spending the BTC
    - An unlocking script in an input satisfies the conditions of the locking script
    - The unlocking script and the locking script are combined and executed
      - If the result is true, the transaction is valid
    - Script keywords are called *opcode*
    - Pay-to-PubKey-Hash(P2PKH): the most common opcode implements a simple transfer



# Bitcoin

## Script – Stack-based Execution



# Bitcoin

## Script

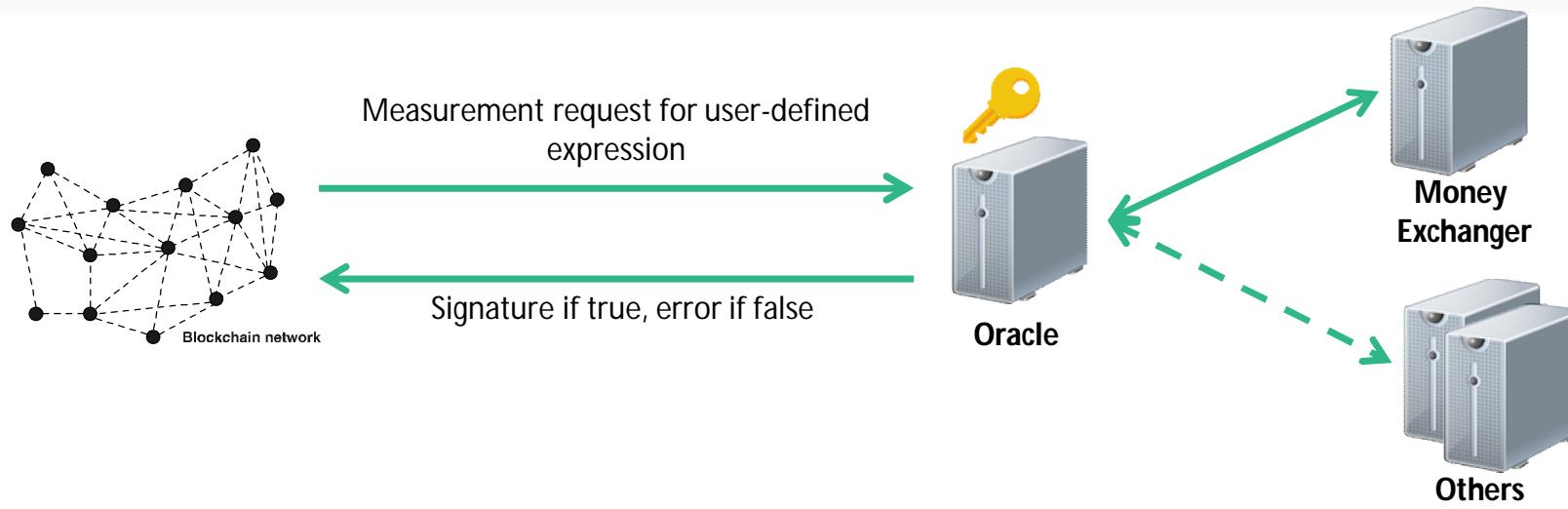
- Provides certain flexibility to change the parameters of the conditions to spend the transferred BTC
  - Multi-signature: A transaction can require multiple keys and signatures
  - OP\_RETURN
    - An *opcode* used to mark a transaction output as invalid
    - Has been used as a standard way to embed arbitrary data to the Bitcoin blockchain for various purposes, like representing assets, e.g. diamonds

# Bitcoin

## Script

- Script programs are pure functions, which cannot import any external state
- An oracle can be used to include external state into the blockchain execution environment

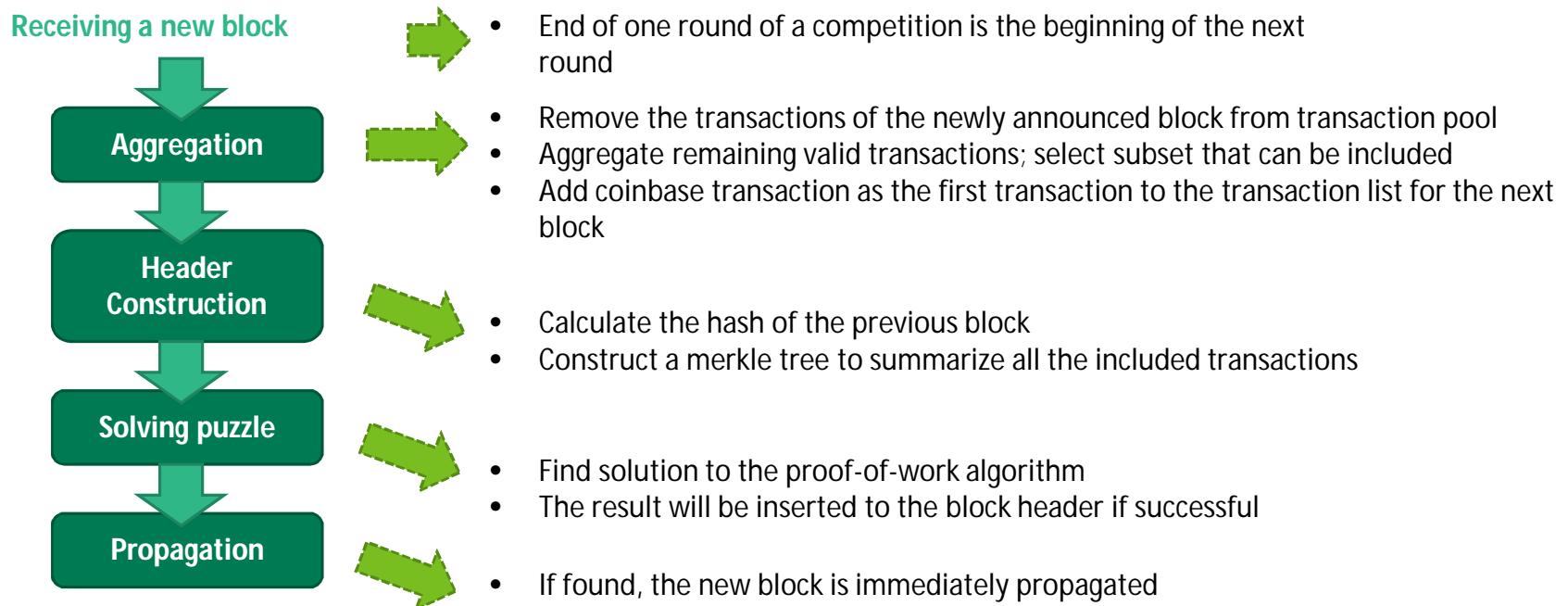
```
today() == 2011/09/25 && exchange_rate(mtgoxUSD) >= 12.5 && exchange_rate(mtgoxUSD) <= 13.5  
Require exchange rate to be between two values on a given date
```



# Bitcoin

## Mining

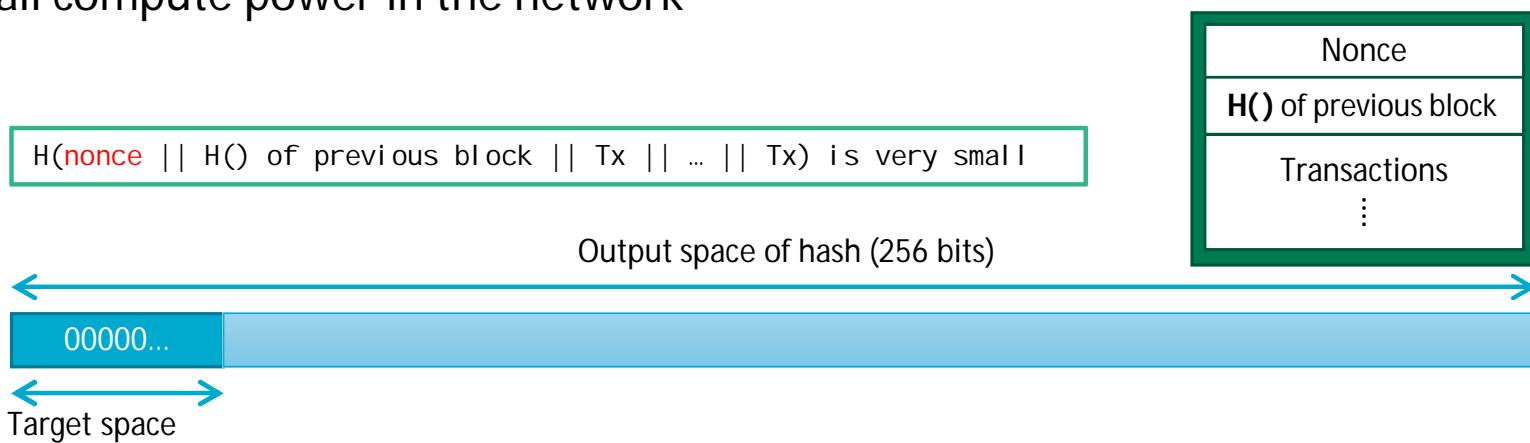
- Miners compete to create new blocks by solving hash puzzles
- Bitcoin proof-of-work: hashcash



# Bitcoin

## Mining – Proof-of-Work

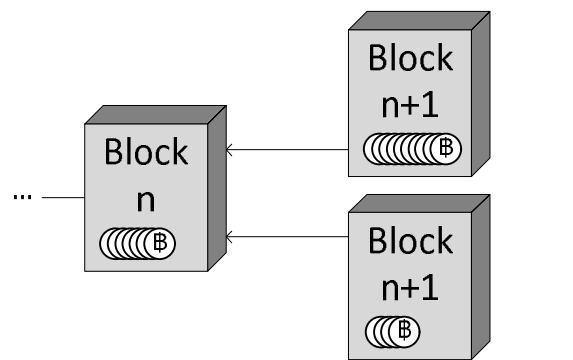
- Solve a hash puzzle - finding a value for a field in the block header, the nonce, which leads to the block hash being smaller than a given threshold
  - Requires a lot trials and errors – try values until you get luck
  - The threshold is adjusted over time to ensure that the average inter-block time is 10 minutes
  - The likelihood to solve the puzzle is proportional to the compute power invested relative to all compute power in the network



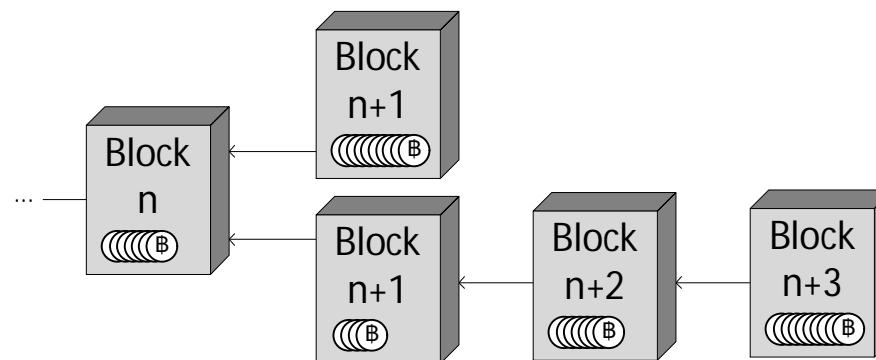
# Bitcoin

## Nakamoto Consensus

- Miners might find and announce the next block at the same time
- Treat the longest history of blocks as the *main chain*
  - The one that received most computation



Fork in the blockchain



Fork decided: longer chain wins

# Bitcoin

## Nakamoto Consensus

- To determine with high probability that a transaction is permanently included:
  - wait for several blocks (5 blocks by default) to be added after first inclusion of the transaction
  - Each of these subsequent blocks is called a *confirmation block*
  - Once sufficiently many confirmations occurred after the transaction block inclusion, then the transaction is considered *committed*
- Unlike many traditional transaction commit semantics:
  - Commit only has a probabilistic guarantee
  - A longer chain *could* appear – although it may be very, very unlikely

# Bitcoin

## Accounts and States

- An account is associated with a cryptographic key pair
  - Public key is used to create the account address
  - Private key: sign transactions sent from the account
- The account balance is the sum of UTXO that an account has control over
- The state of the blockchain, and the account balances of all users, result from the genesis block (very first block of the blockchain) and the set of transactions included since
  - Some accounts might be pre-loaded with an initial account balance from the beginning
- As transactions are grouped into blocks, the entire system moves from one discrete state to another through adding a new block

# Bitcoin

## Wallets and Exchanges

- Wallets
  - Software wallet: Manage a collection of private keys corresponding to their accounts, and to create and sign transactions
  - Hardware wallets are devices that store private keys in chips
  - Cold storage backups: to avoid key loss/stores a representation of the keys independent of the user's current hardware wallets
- Exchanges
  - Places to trade Bitcoin with other currencies
  - Holds currency on behalf of users
  - Clients may choose to ask the exchange to transfer purchased Bitcoin to an address under their control
  - If the exchange's system fails, their users may lose control of "their" Bitcoin.
  - Key stakeholders for public blockchain
    - Provide liquidity for cryptocurrency, which supports its real-world value
    - Underpin the incentive mechanism

# Bitcoin

## A bit of fun: Bitcoin Obituaries

- Declared dead over 300 times, and yet... still alive and operating!
- <https://99bitcoins.com/bitcoin-obituaries/>

## Bitcoin Obituaries



Bitcoin has died 346 times

[Submit an Obituary](#)

### Bitcoin Obituary Stats

#### Most Recent Death:

February 14, 2019 - [JPMorgan Just Killed the Bitcoin Dream](#)

#### Oldest Death:

December 15, 2010 - [Why Bitcoin can't be a currency](#)



# Agenda:

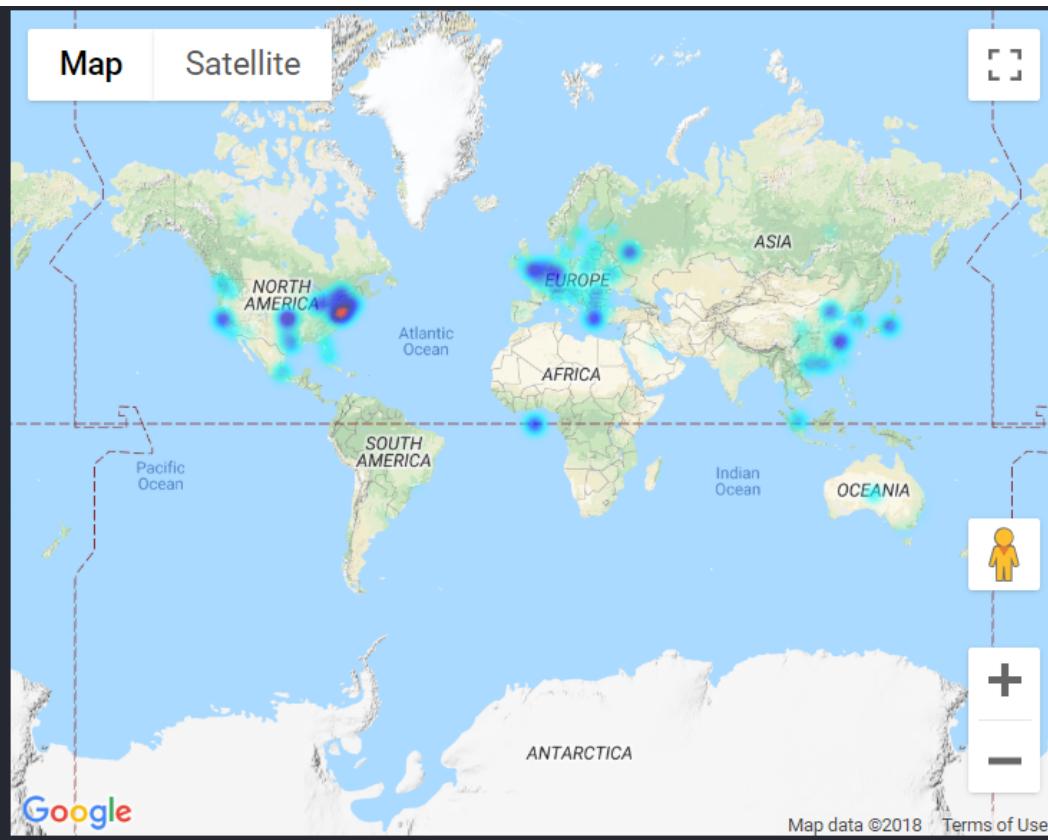
- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development



# Ethereum

## Network distribution

Total	13355 (100%)
United States	5612 (42.02%)
China	1752 (13.12%)
Canada	974 (7.29%)
Germany	557 (4.17%)
Russian Federation	462 (3.46%)
United Kingdom	436 (3.26%)
Netherlands	305 (2.28%)
Korea, Republic of	271 (2.03%)
France	240 (1.80%)
Ukraine	228 (1.71%)



# Ethereum

## Exchange rate

Lowest price: \$0.05  
Highest price: \$1400



Source: [etherscan.io/chart/etherprice](https://etherscan.io/chart/etherprice)

# Ethereum

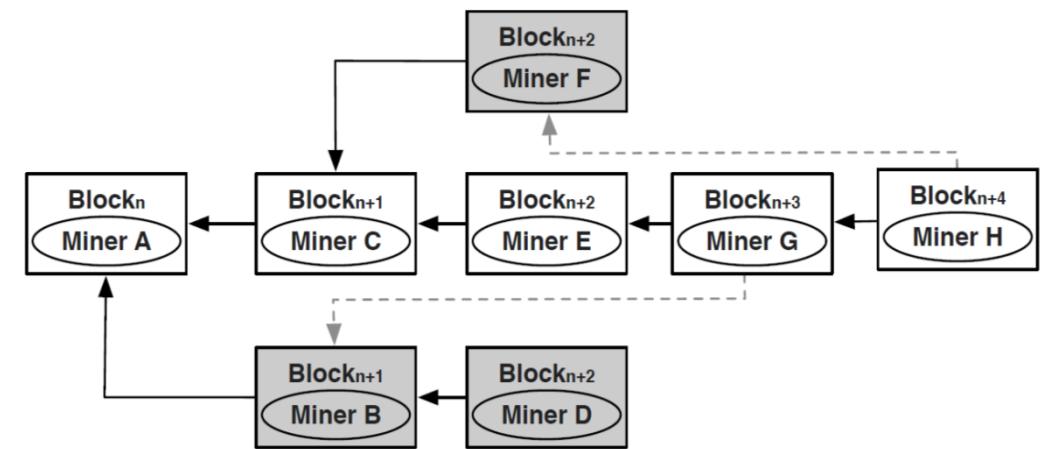
## Proof of Work - Ethash

- Shorter inter-block time: 13-15 seconds (Bitcoin: 10 mins)
  - The Ethereum inter-block time is significantly shorter than Bitcoin's, typically meaning transactions are recorded and executed a lot faster on the Ethereum blockchain
  - According to median values, Ethereum appends a new block to its data structure every 13-15 seconds
- Smaller blocks
  - Gas limit per block - block size limits complexity of transactions
  - At most 380 transactions in a block (Bitcoin: 1,500 txs/block)
    - Currently maximum block size is around 8,000,000 gas
    - Basic transactions have a complexity of 21,000 gas
  - Most blocks are under 2KB (Bitcoin: 1 MB)

# Ethereum

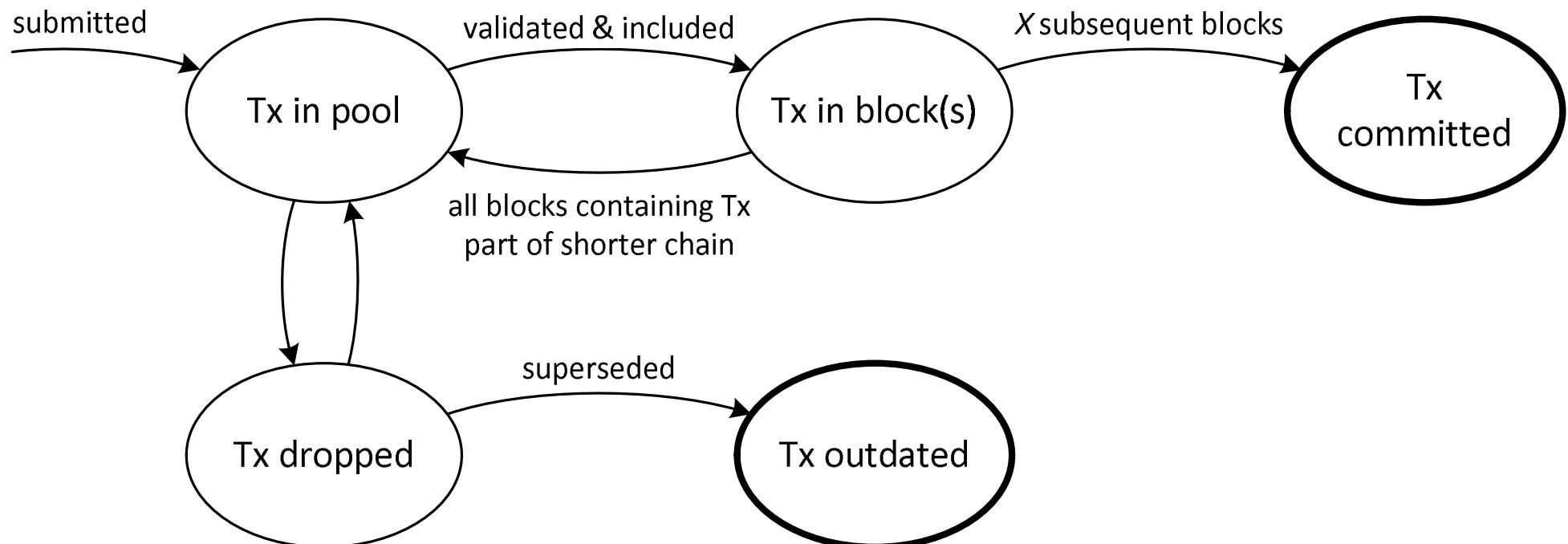
# Ethereum Protocol

- Ethereum's inter-block time is not hugely longer than block propagation time around the globe
    - Much more likely that multiple competing blocks are created at a similar time
  - GHOST protocol (Greedy Heaviest Observed Subtree)
    - The heaviest chain wins and uncles contribute to the weight
    - Miners of uncle blocks receive 87.5% of a standard block reward
    - For every uncle included in the block, the miner gains an additional 3.125% and increases the weight of the chain including the block
  - Ethereum uses 3 Merkle trees, one each for integrity of:
    - State (account balances, data stored, etc.)
    - Transactions
    - Receipts (effects of transactions)



# Ethereum

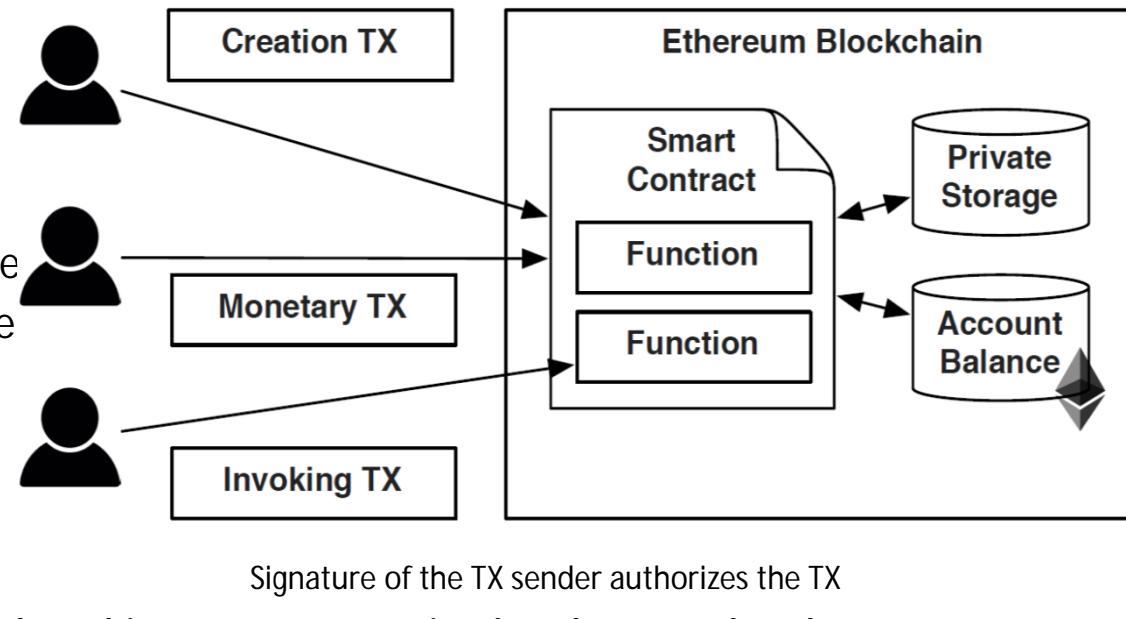
## Transactions Lifecycle



# Ethereum

## Smart Contract (SC) in Solidity

- SC creation TX: deployment
  - After included, identified by a SC address
  - A smart contract account contains
    - A piece of executable code
    - An internal storage to store internal state
    - An amount of Ether, i.e. contract balance
- Monetary TX
  - Users can transfer Ether to the SC
- Invoking TX contains
  - The interface of the function being invoked and its parameters in the data payload
  - An amount of Ether as gas for the TX execution



# Ethereum

## Paying Fees in “Gas”

- Gas: a fee to limit the resource usage
- Gas cost
  - A fixed gas cost per transaction (base cost)
  - Variable gas cost for data (dependent on its size) and execution of a smart contract method (charged per bytecode instruction)
  - Additional gas cost for the deployment of new contracts
- Gas cost is converted to Ether according to a user defined gas price –how much ether per gas the TX creator is willing to pay
  - By default, clients may set the gas price to a market price
  - Users set higher gas price if inclusion of transaction is urgent, or lower gas price if transaction inclusion is not (time-) critical
  - Gas price recommendations available e.g. from ETH Gas Station  
<https://ethgasstation.info/>

# Ethereum

## Data structure of a transaction

- An Ethereum transaction has the following fields
  - From – source address (regular account)
  - To – destination address (regular account or contract address)
  - Value – Ether (in unit “wei”) to transfer to the destination (can be 0)
  - Nonce – transaction sequence number for the sending account
  - Gas price – price you are offering to pay (Ether per gas)
  - Gas limit – maximum amount of gas allowed for the transaction (called “startgas”)
  - Data – payload data, e.g., description of transaction, binary code to create smart contract, or function invocation
  - Digital signature (field names: v, r, s)
- Once included, additional information (block number, actual gas used, actual fee, etc.) is available
- See e.g. <https://etherscan.io/tx/0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060>

# Agenda:

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development



# Recap: Dapps and Smart Contracts

- Smart contracts
  - Programs deployed as data and executed in transactions on the blockchain
  - Blockchain can be a computational platform (more than a simple distributed database)
  - Code is deterministic and immutable once deployed
  - Can invoke other smart contracts
  - Can hold and transfer digital assets
- Decentralized applications or dapps
  - Main functionality is implemented through smart contracts
  - Backend is executed in a decentralized environment
  - Frontend can be hosted as a web site on a centralized server
    - Interact with its backend through an API
  - Could use decentralized data storage such as IPFS
  - “State of the dapps” is a directory recorded on blockchain:  
<https://www.stateofthedapps.com/>

# Smart Contracts in general

- Analogy: Java program
  - Smart contract code: a class
  - Deployed contract: an object
- Many software design patterns still apply, e.g., Factory
  - Code is deterministic, i.e., same state/inputs result in the same state changes/outputs
    - However, state may not be deterministic from viewpoint of the caller (e.g. block number)
- Why can smart contract execution be trustworthy?
  - A contract is deployed as data in a transaction and thus immutable
  - All their inputs are through transactions and the current state
  - Their code is deterministic
  - The results of transactions (including function calls) are captured in the state and receipt trees, which are part of the consensus
    - If other nodes get a different result, they would reject a proposed block

# Smart Contract Development in Ethereum

- Smart contracts in Ethereum are deployed and executed as byte code
  - Runs on the Ethereum Virtual Machine (EVM)
- Typically, byte code results from compiling code in a high-level language
- Most popular language for Ethereum: Solidity
  - High-level, object-oriented language; syntax is similar to that of JavaScript
  - Statically typed, supports inheritance, libraries and complex user-defined types
- Useful links:
  - In-browser IDE: <https://remix.ethereum.org/>
  - Solidity official documentation page: <https://solidity.readthedocs.io/en/latest/>
  - Official tutorials / examples: <https://solidity.readthedocs.io/en/latest/solidity-by-example.html>
  - Other tutorials, e.g.: [https://ethereumbuilders.gitbooks.io/guide/content/en/solidity\\_tutorials.html](https://ethereumbuilders.gitbooks.io/guide/content/en/solidity_tutorials.html)
  - Smart contract best practices from Consensys (a company)  
<https://consensys.github.io/smart-contract-best-practices/>

# Solidity

## Example

```
pragma solidity >=0.4.0 <0.6.0; // compiler instruction: language and version

contract SimpleStorage {
    uint storedData;           // variable declaration

    // getter, callable by anyone on the network
    function get() constant returns (uint retVal) {
        return storedData;
    }

    // setter, callable by anyone on the network
    function set(uint x) {
        storedData = x;
    }
}
```

# Solidity

## Features

- Object-oriented, high-level language
- Statically typed, supports inheritance, libraries and complex user-defined types
  - Multi-inheritance is supported
- Designed for smart contracts to run on the Ethereum Virtual Machine (EVM)
- Syntax is closest to JavaScript, with some similarity to Java
  - But no support for Lambda expressions (anonymous methods without a declaration)
  - Also influenced by C++, Python
- Each deployed contract is assigned an address (like a regular account)
  - Can receive, hold, and spend assets, like Ether
  - Assets held by the account address are controlled by the program code
    - Bugs / vulnerabilities / omissions can lead to permanent loss of assets
    - But: untrusting parties can use smart contract as neutral, trustworthy middle ground
- Follow best practices, especially around security – there is no “safety net”

# Solidity

## Features

- Deploy contract by sending a transaction with recipient “to” set to “null”
- Can specify interfaces, by having at least one un-implemented function

```
contract base { function foo(); }           // an interface  
contract derived is base { function foo() {} } // implements the interface
```

- Can overload functions (same function name but different parameters)
  - But might not be a good idea – hard to understand the code
- Arrays work in the usual, Java/JavaScript-like way
- Constructor is executed when a contract is created is executed once
  - Function name: constructor

```
uint[3] public data; data[0] = 0; ...  
contract Base {  
    constructor(uint i) public {...}  
}
```

# Solidity

## Remix example

```
pragma solidity >=0.4.22 <0.6.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    /// Create a new ballot with _numProposals different proposals.
    constructor(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }
}

// From https://remix.ethereum.org/
```



# Solidity

## Remix test example

```
import "remix_tests.sol"; // this import is automatically injected by Remix.
import "./Ballot.sol";    // the file to test

contract test3 {

    Ballot ballotToTest;
    function beforeAll () public {
        ballotToTest = new Ballot(2);
    }

    function checkWinningProposal () public {
        ballotToTest.vote(1);
        Assert.equal(ballotToTest.winningProposal(), uint(1),
                    "1 should be the winning proposal");
    }

    function checkWinningProposalWithReturnValue () public view returns (bool) {
        return ballotToTest.winningProposal() == 1;
    }
}

// From https://remix.ethereum.org/
```

# Solidity

## Visibility of data / functions

- Solidity provides two types of function calls: internal and external
  - Internal call: local call from the same contract
  - External: by message call
- Functions can be specified to be *external*, *public*, *internal* or *private*
  - External: only from outside (transactions or other contracts); part of the contract interface
    - If function `f` is external, calling it internally is impossible (for example, `f()` will not work, however, `this.f()` works).
  - Public: can be called both by message calls and internally; part of the contract interface
    - Used to be the default, but compiler now enforces explicit declaration
  - Internal: only from other code in the same contract (including by inheritance)
  - Private: only be visible for the contract that they are declared in only, and not even in derived contracts
    - No data is ever really private in smart contracts – anyone on the network can extract it
- For state variables, external is not possible and the default is internal
  - Compiler automatically creates getter functions for public variables

# Solidity

## Visibility of data / functions - example

```
pragma solidity ^0.5.1;

contract cont1 {
    uint private data;
    function func(uint x) private returns(uint y) { return x + 1; }
    function dataSet(uint x) public { data = x; }
    function dataGet() public returns(uint) { return data; }
    function compute(uint x, uint y) internal returns (uint) { return x+y; }
}

contract cont2 {
    function dataRead() public {
        cont1 z = new cont1();
        uint local = z.func(7); // error: member "func" is not visible
        z.dataSet(3);
        local = z.dataGet();
        local = z.compute(3, 5); // error: member "compute" is not visible
    }
}

contract cont3 is cont1 {
    function g() public {
        cont1 z = new cont1();
        uint val = compute(3, 5); // access to internal member (from derived to parent contract)
    }
}

// Adapted from https://www.btdegree.org/learn/solidity-visibility-and-getters
```

# General principles for smart contracts

- Follow the KISS principle: keep it simple, stupid
  - ... and readable / understandable, so other developers/technical users can start trusting into your code
- Follow best practices, especially around security – there is no “safety net”
- Interface of a smart contract is NOT visible from the deployed code
  - No requirement of the binary code to have a particular structure
    - But Solidity compiler establishes a particular structure
    - This structure needs to be known to the caller – i.e., the transaction invoking a smart contract method must be aware of this structure and be constructed accordingly
  - Signatures can be guess-extracted (but not necessarily the function names), if the binary code was written in Solidity and compiled with the standard compiler
    - Cannot rely on it not becoming known
- Make the code available to the potential users (e.g., open source), or at least the interface
  - Many interface standards proposed, e.g., ERC-20 for fungible tokens
- Always look at the online documentation: Solidity is under very active development and changes are frequent

# Summary

- Use cases
- Cryptography basics
- Bitcoin
- Ethereum
- Smart Contract Development



# Course Outline – next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
2nd	25 Feb	Ingo Weber	Existing Blockchain Platforms	4. Example use cases 2. Existing Blockchain Platforms (1h on smart contract dev)	Assignment 1 out (Monday before lecture)
3rd	4 Mar	Sherry Xu	Blockchain in Software Architecture 1	3. Varieties of blockchain 5. Blockchain in Software Architecture (including software architecture basics) 1/2	
4th	11 Mar	Mark Staples	Blockchain in Software Architecture 2	5. Blockchain in Software Architecture (Non-functional properties and trade-offs) 2/2	Pitching session Assignment 1 due (Wednesday)



# End of Lecture / Consultation

Ingo Weber | Principal Research Scientist & Team Leader

Architecture & Analytics Platforms (AAP) team

[ingo.weber@data61.csiro.au](mailto:ingo.weber@data61.csiro.au)

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

[www.data61.csiro.au](http://www.data61.csiro.au)



# COMP6452 Lecture 3: Blockchain in Software Architecture 1

Xiwei (Sherry) Xu (xiwei.xu@data61.csiro.au)

4<sup>th</sup> of March, 2019

[www.data61.csiro.au](http://www.data61.csiro.au)

# Outline

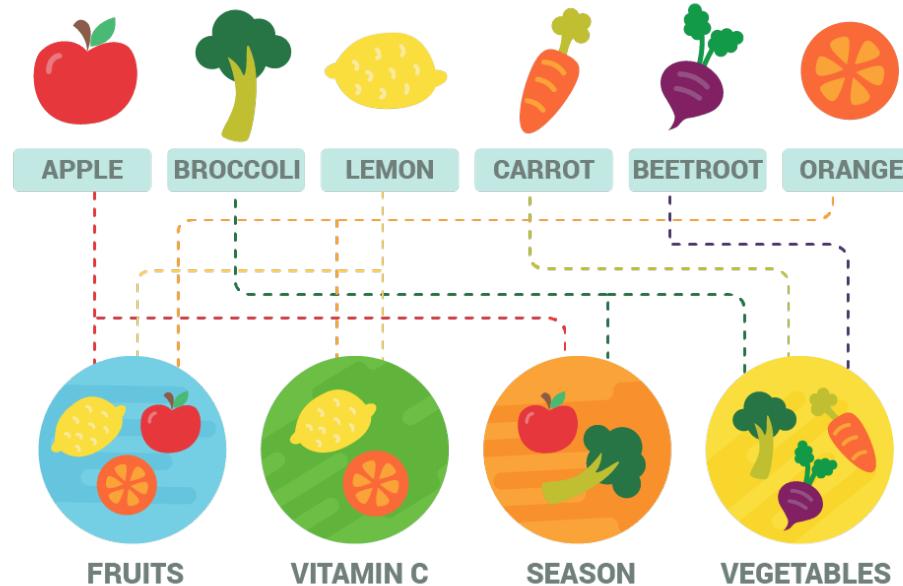
- What is Taxonomy?
- Varieties of Blockchain – A Taxonomy
  - (De)centralization
  - Deployment
  - Ledger Structure
  - Consensus Protocol
  - Block Configuration and Data Structure
  - Auxiliary Blockchain
  - Anonymity
  - Incentive
- Summary



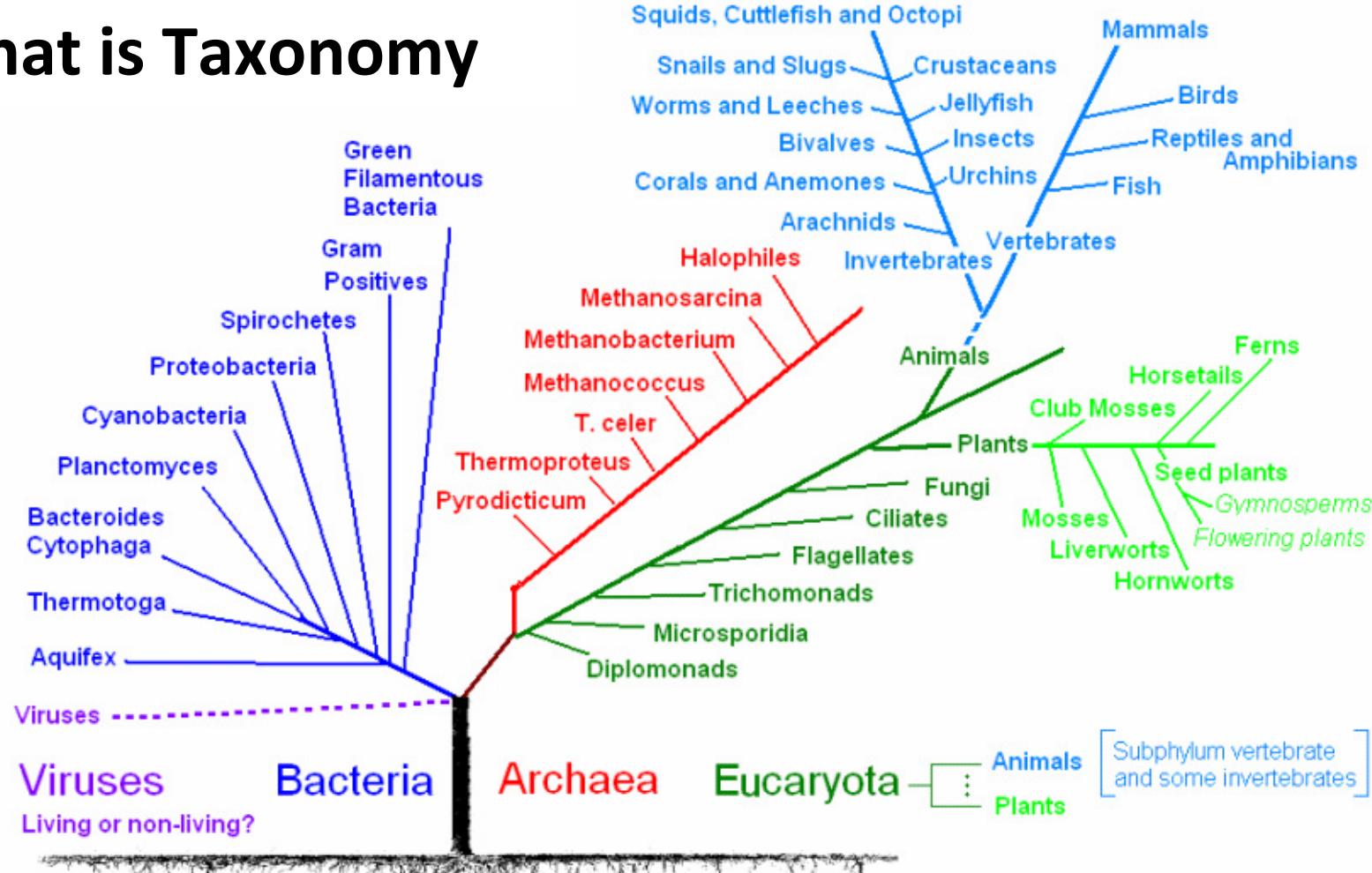
# What is Taxonomy?

# What is Taxonomy?

- *Taxonomy is the practice and science of classification of things or concepts, including the principles that underlie such classification*



# What is Taxonomy



# Why Taxonomy?

- Diverse range of blockchain has emerged since the advent of Bitcoin in 2008
  - Complex internal structure and many configurations and variants
- Comparison of different blockchain is difficult
  - Lack of product data and technology evaluation resources
- Blockchain Taxonomy
  - Dimensions and categories for classifying blockchains
  - Understanding blockchain technology
- Benefits
  - Systematically consider the features and configurations of blockchain
  - Explore the conceptual design space
  - Compare and evaluate design options
    - Assess their impact on **quality attributes**



# Properties of Blockchain

- Blockchain cannot meet requirements for all usage scenarios
  - E.g. those that require real-time processing
- Fundamental Properties
  - Immutability *from committed transaction*
  - Integrity *from cryptographic tool*
  - Transparency *from public access*
  - Equal rights *from consensus*
    - Weighted by the compute power or stake owned by the miner
- Limitation
  - Data privacy
    - No privileged users
  - Scalability
    - Size of the data on blockchain
    - Transaction processing rate
    - Latency of data transmission

# Taxonomy: A Glimpse 1/2

## Classification

	Permission-less	Permissioned
Public	<p><b>Consensus:</b> Proof-of-X</p> <p><b>Permission management</b></p> <ul style="list-style-type: none"><li>• Blockchain layer</li><li>• Application layer (optional)</li></ul> <p><b>Incentive:</b> Blockchain layer</p>	<p><b>Consensus</b></p> <ul style="list-style-type: none"><li>• Proof-of-X</li><li>• PBFT, Federated consensus, Round Robin etc.</li></ul> <p><b>Permission management</b></p> <ul style="list-style-type: none"><li>• Blockchain layer</li><li>• Application layer (optional)</li></ul> <p><b>Incentive:</b></p> <ul style="list-style-type: none"><li>• Blockchain layer</li><li>• Governance around permissions</li></ul>
Private	<p><b>Consensus</b></p> <ul style="list-style-type: none"><li>• Proof-of-X</li><li>• PBFT, Federated consensus, Round Robin etc.</li></ul> <p><b>Permission management:</b></p> <ul style="list-style-type: none"><li>• Blockchain layer</li><li>• Network layer</li><li>• Application layer (optional)</li></ul> <p><b>Incentive:</b> Governance around access</p>	<p><b>Consensus</b></p> <ul style="list-style-type: none"><li>• Proof-of-X</li><li>• PBFT , Federated consensus, Round Robin etc.</li></ul> <p><b>Permission management:</b></p> <ul style="list-style-type: none"><li>• Blockchain layer</li><li>• Network layer</li><li>• Application layer (optional)</li></ul> <p><b>Incentive:</b> Governance around access permission</p>

# Taxonomy: A Glimpse 2/2

## Quality Tradeoffs

	Permission-less	Permissioned																														
Public	<table><tr><td>Immutability</td><td>+++ (#Nodes, Consensus, Topology)</td><td>++</td></tr><tr><td>Integrity</td><td>+++ (#Nodes, Consensus, Topology)</td><td>++</td></tr><tr><td>Transparency</td><td>++ (Access control)</td><td>++</td></tr><tr><td>Availability</td><td>+++ (#Nodes, Topology)</td><td>++</td></tr><tr><td>Performance</td><td>+ (Consensus, latency)</td><td>++</td></tr><tr><td>Cost Efficiency</td><td>+</td><td>++</td></tr></table>	Immutability	+++ (#Nodes, Consensus, Topology)	++	Integrity	+++ (#Nodes, Consensus, Topology)	++	Transparency	++ (Access control)	++	Availability	+++ (#Nodes, Topology)	++	Performance	+ (Consensus, latency)	++	Cost Efficiency	+	++	<table><tr><td>Immutability</td><td>++</td></tr><tr><td>Integrity</td><td>++</td></tr><tr><td>Transparency</td><td>++</td></tr><tr><td>Availability</td><td>++</td></tr><tr><td>Performance</td><td>++</td></tr><tr><td>Cost Efficiency</td><td>++</td></tr></table>	Immutability	++	Integrity	++	Transparency	++	Availability	++	Performance	++	Cost Efficiency	++
Immutability	+++ (#Nodes, Consensus, Topology)	++																														
Integrity	+++ (#Nodes, Consensus, Topology)	++																														
Transparency	++ (Access control)	++																														
Availability	+++ (#Nodes, Topology)	++																														
Performance	+ (Consensus, latency)	++																														
Cost Efficiency	+	++																														
Immutability	++																															
Integrity	++																															
Transparency	++																															
Availability	++																															
Performance	++																															
Cost Efficiency	++																															
Private	<table><tr><td>Immutability</td><td>+</td><td>+</td></tr><tr><td>Integrity</td><td>+</td><td>+</td></tr><tr><td>Transparency</td><td>+</td><td>+</td></tr><tr><td>Availability</td><td>+</td><td>+</td></tr><tr><td>Performance</td><td>+++</td><td>+++</td></tr><tr><td>Cost Efficiency</td><td>+++</td><td>+++</td></tr></table>	Immutability	+	+	Integrity	+	+	Transparency	+	+	Availability	+	+	Performance	+++	+++	Cost Efficiency	+++	+++	<table><tr><td>Immutability</td><td>+</td></tr><tr><td>Integrity</td><td>+</td></tr><tr><td>Transparency</td><td>+</td></tr><tr><td>Availability</td><td>+</td></tr><tr><td>Performance</td><td>+++</td></tr><tr><td>Cost Efficiency</td><td>+++</td></tr></table>	Immutability	+	Integrity	+	Transparency	+	Availability	+	Performance	+++	Cost Efficiency	+++
Immutability	+	+																														
Integrity	+	+																														
Transparency	+	+																														
Availability	+	+																														
Performance	+++	+++																														
Cost Efficiency	+++	+++																														
Immutability	+																															
Integrity	+																															
Transparency	+																															
Availability	+																															
Performance	+++																															
Cost Efficiency	+++																															

# Taxonomy Dimensions

(De)centralization

Deployment

Ledger Structure

Consensus Protocol

Block Configuration and Data Structure

Auxiliary Blockchain

Anonymity

Incentive



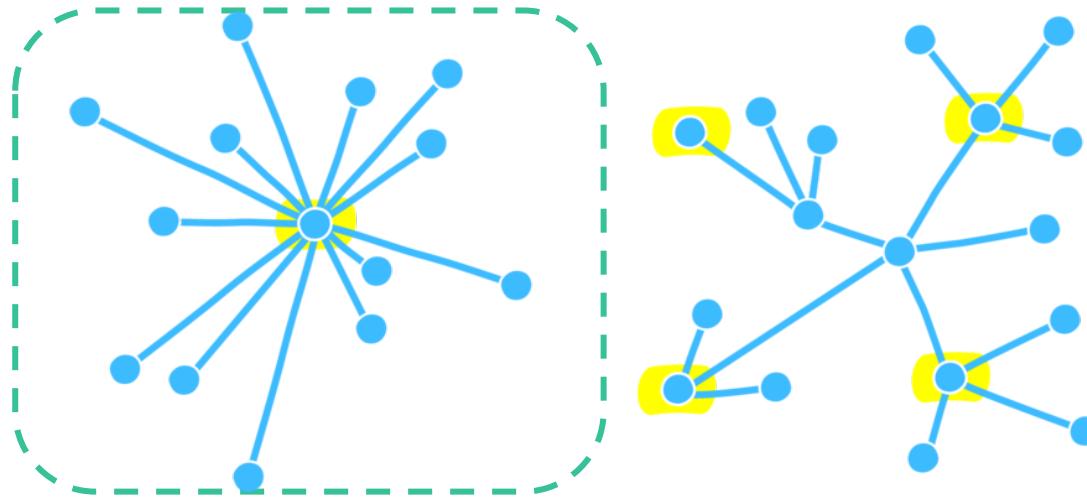
# Centralization – Decentralization

Design Decision	Option	Impact				#Failure points
		Fundamental properties	Cost efficiency	Performance		
Fully Centralised	Services with a single provider ( <i>e.g.</i> , governments, courts)	⊕	⊕⊕⊕	⊕⊕⊕	1	
	Services with alternative providers ( <i>e.g.</i> , banking, online payments, cloud services)					
Partially Centralised & Partially Decentralised	Permissioned blockchain with permissions for fine-grained operations on the transaction level ( <i>e.g.</i> , permission to create assets)	⊕⊕	⊕⊕	⊕⊕	*	
	Permissioned blockchain with permissioned miners (write), but permission-less normal nodes (read)					
Fully Decentralised	Permission-less blockchain	⊕⊕⊕	⊕	⊕	Majority (nodes, power, stake)	



# Full Centralization

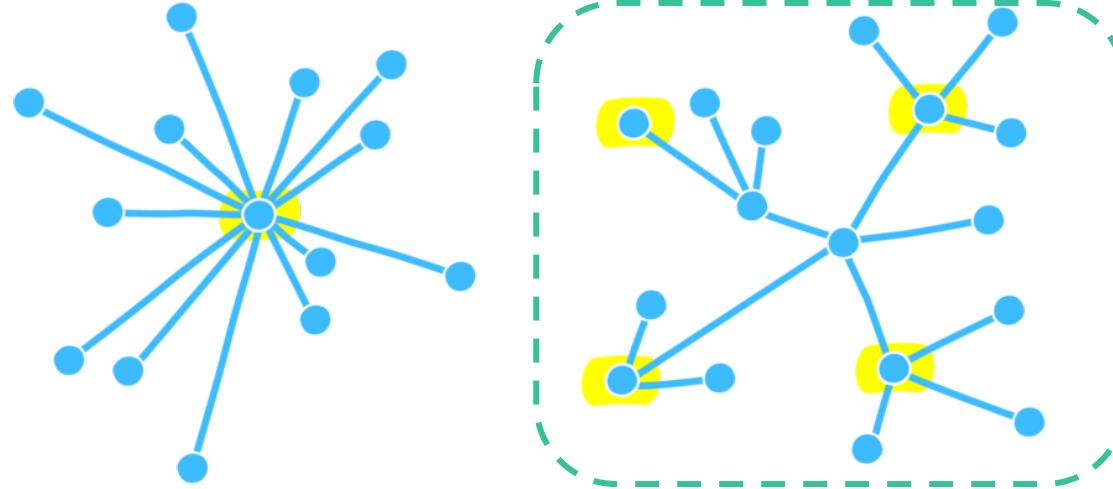
- Services with a single provider
  - E.g., governments, courts, business monopolies
  - Single point of failure
- Services with alternative providers
  - E.g., banks, online payments, cloud services
  - Failure of a single service provider affects its users



# Full Decentralization



- Permission-less public blockchains
- Completely open
  - New users can join, validate transaction or mine block at any time
- Protect against Sybil attack
  - Pseudonymity: blockchain account represents a user
  - PoW: total amount of computational power rather than the number of nodes is important for integrity



# Partial (De)centralization 1/2

- Permissioned blockchain requires authorities act as a gate for participation
  - Permission to join the network (read)
  - Permission to send transaction
  - Permission to mine (write)
- Permissioned blockchain with permissioned miners (write), and permission-less normal nodes (read)
- Permissioned blockchain with permissions for fine-grained operations
  - Permission to create assets



# Partial (De)centralization 2/2

- More suitable in regulated industries
  - Banks establish the real-world identity of transacting parties
    - Know-Your-Customer (KYC) regulation
  - Transactions on permission-less blockchain
    - Across jurisdictional boundaries
    - Undermine regulatory controls
- Better control access to off-chain information about real-world assets

## Tradeoffs between permissioned and permission-less blockchains

Transaction processing rate, cost, flexibility in changing the network rules, reversibility and finality

# Taxonomy Dimensions

(De)centralization

**Deployment**

Ledger Structure

Consensus Protocol

Block Configuration and Data Structure

Auxiliary Blockchain

Anonymity

Incentive



# Deployment Overview

Deployment Option	Fundamental properties	Impact		
		Cost efficiency	Performance	Flexibility
Public blockchain	⊕⊕⊕	⊕	⊕	⊕
Consortium/community blockchain	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Private blockchain	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

- Public blockchain: Most cryptocurrencies
  - Anyone on the internet can access
  - Better information transparency and auditability
  - Sacrifice performance
  - Data privacy relies on encryption or cryptographic hashes
  - Different cost model (*Lecture 5*)

# Deployment Overview

Deployment Option	Fundamental properties	Impact		
		Cost efficiency	Performance	Flexibility
Public blockchain	⊕⊕⊕	⊕	⊕	⊕
Consortium/community blockchain	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Private blockchain	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

- Consortium/Community blockchain
  - Across multiple organizations
  - Consensus process controlled by pre-authorized nodes
  - Read permission can be public or restricted to specific participants
- Private blockchain
  - Write permission kept within one organization
  - Governed and hosted by a single organization - most flexible

Consortium/private instantiation of public blockchain

- Blockchain platform is open source
- Network layer access control – firewall

# Taxonomy Dimensions

(De)centralization

Deployment

**Ledger Structure**

Consensus Protocol

Block Configuration and Data Structure

Auxiliary Blockchain

Anonymity

Incentive



DATA  
61

# List

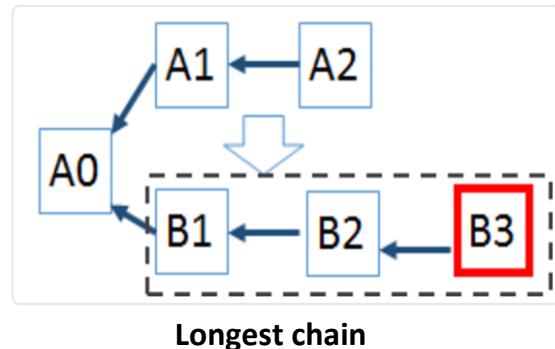
Option	Fundamental properties	Cost efficiency	Impact		Flexibility
			Performance		
Global list of blocks (Bitcoin)	⊕⊕⊕	⊕	⊕	⊕	⊕
Global DAG of blocks (Hashgraph)	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Global DAG of transactions (IOTA)	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Restricted shared ledgers (Corda)	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

- Global list of blocks
  - Bitcoin/Ethereum
  - Nodes record blockchain as tree of blocks
    - Shorter branches attached to the main chain represent alternative competing histories
    - Used for operating blockchain and determining consensus
  - Blockchain is a list of blocks under the logical view from a user's perspective

# Tree



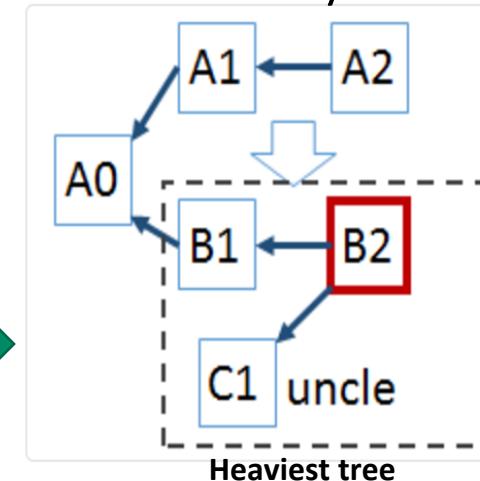
- Orphan/Stale
- Two nodes find a block at same time
- Propagated, verified, but eventually being cast off
- Fast block time suffer from a high number of stale blocks
- Ghost (Greedy Heaviest-Observed Sub-Tree)
- Add stale blocks (uncles) into calculation of cumulative difficulty



Bitcoin

Ethereum

- Block time interval: 10 minutes
- Block propagation (50% of the network): ~12 seconds



Heaviest tree



# Directed Acyclic Graph (DAG)

Option	Fundamental properties	Cost efficiency	Impact		Flexibility
			Performance		
Global list of blocks (Bitcoin)	⊕⊕⊕	⊕	⊕	⊕	⊕
Global DAG of blocks (Hashgraph)	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Global DAG of transactions (IOTA)	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Restricted shared ledgers (Corda)	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

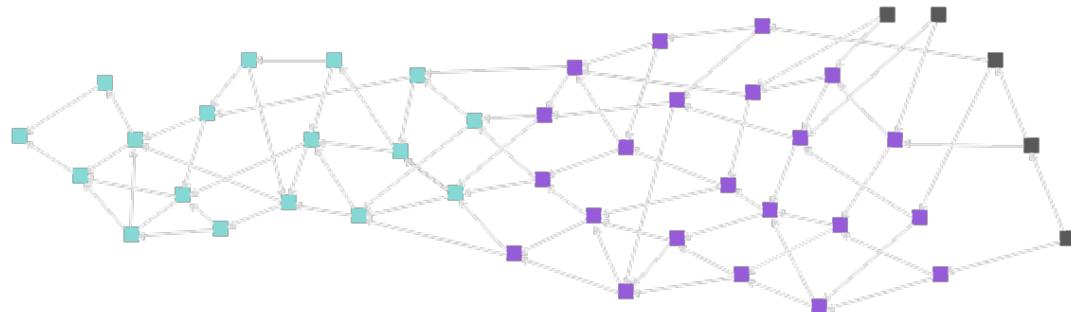
- Global DAG of blocks
  - Hashgraph
  - Logical view of transactions is based on a directed acyclic graph of blocks
    - Rather than a list

# Graph

Blockchain



Tangle (DAG/ Directed Acyclic Graph)



	Blockchain (Bitcoin)	IoTA
Byzantine Toleration	51%	34%
Confirmation Time	60 min (6-Confirmation)	Unstable

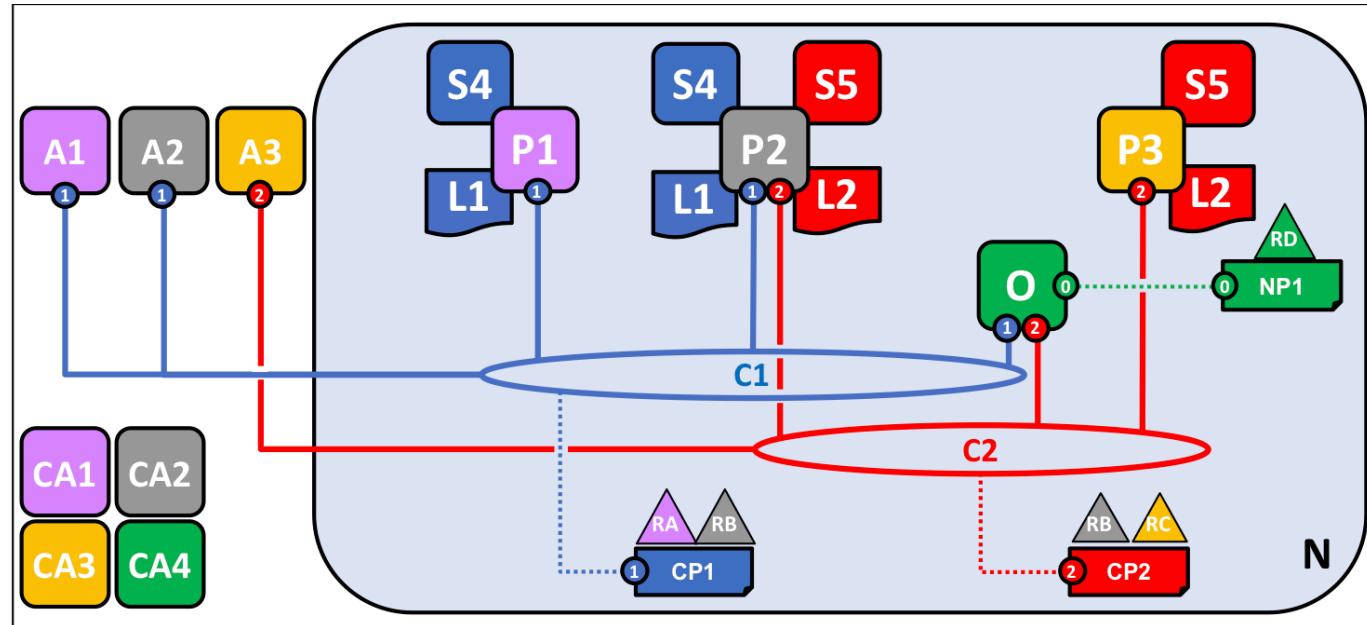
# Tradeoff Overview

Option	Fundamental properties	Cost efficiency	Impact		Flexibility
			Performance	Flexibility	
Global list of blocks (Bitcoin)	⊕⊕⊕	⊕	⊕	⊕	⊕
Global DAG of blocks (Hashgraph)	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Global DAG of transactions (IOTA)	⊕⊕	⊕⊕	⊕⊕	⊕⊕	⊕⊕ transaction history
Restricted shared ledgers (Corda)	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕ Multiple small ledgers

- Multiple small ledgers shared between parties of interest
  - Hyperledger Fabric, Corda
  - Parties of interest are authorized to view the transactions recorded in the ledgers

# Peer-to-Peer Ledger

- Hyperledger Fabric
- A collection of small ledgers
  - Channel
- More rigid transaction distribution policy
  - Isolating transactions within the channels



- Corda
- Abstract logic view is a global graph of transactions
  - View most parties see is a collection of small ledgers shared with other business contacts
- Notaries are used to further limit transaction distribution
  - Special agents
  - Attest to the integrity of unseen parts of the global transaction graph

c·rda



# Taxonomy Dimensions

(De)centralization

Deployment

Ledger Structure

## Consensus Protocol

Block Configuration and Data Structure

Auxiliary Blockchain

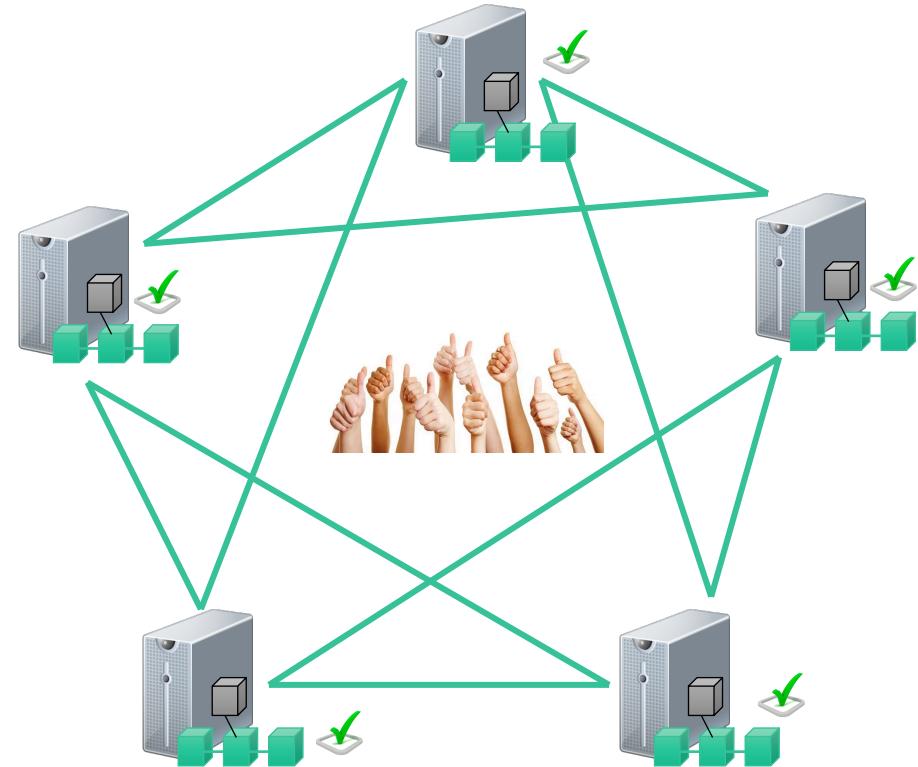
Anonymity

Incentive



# Consensus Protocol

- Miners generate new blocks
- Miners propagate the blocks to the peers in the blockchain network
- Miners encounter different competing new blocks
- Miners resolve this using consensus mechanism



# Consensus Overview

	Option	Fundamental properties	Cost efficiency	Impact	
				Performance	Flexibility
Security-wise	Proof-of-work	⊕⊕⊕	⊕	⊕	⊕
	Proof-of-retrievability	⊕⊕⊕	⊕	⊕	⊕
	Proof-of-stake	⊕⊕	⊕⊕	⊕⊕	⊕⊕⊕
	Practical Byzantine Fault Tolerance (PBFT)	⊕	⊕⊕⊕	⊕⊕⊕	⊕
Scalability-wise	Bitcoin-NG	⊕⊕⊕	⊕	⊕	⊕
	RBBC	⊕⊕	⊕⊕⊕	⊕⊕⊕	⊕

# Proof-of-Work 1/2



- Miners compete for right to write block
- Solve a hash puzzle
  - Easy to verify, difficult to solve
  - Takes effectively random time

$H(\text{nonce} \mid\mid H(\text{ }) \text{ of previous block} \mid\mid \text{Tx} \mid\mid \dots \mid\mid \text{Tx})$  is very small

Nonce
$H(\text{ })$ of previous block
Transactions
⋮

Output space of hash (256 bits)



- *If the Hash function is secure: The only way to succeed is to try enough number of values*
- *Prob (winning next block) = Fraction of global hash power the miner controls*

# Proof-of-Work 2/2

- Not energy-efficient
  - Electricity consumption of Bitcoin is close to Turkmenistan
- Proof-of-work for good use
  - Primecoin
    - Generates prime number chains which are of interest to mathematical research



# Proof-of-Stake

- Select the next mining node based on the control of the native digital currency
- Align the incentive of digital currency holders with the good operation
- Does not necessarily select the next miner with the largest stakeholding
- More Cost-efficient, shorter latency

## Peercoin

- Prove the ownership of a certain amount of peercoin
- Combines randomization and coin age



peercoin



NXT  
Cryptocurrency



Tendermint



casper

## Delegated Proof-of-Stake

- Account delegate their stake to other accounts rather than participating in the transaction validation
- Bitshares
  - Representative take turns in a round-robin manner



bitshares™



# Practical Byzantine Fault Tolerance (PBFT)

- Ensure consensus despite arbitrary behaviors from fraction of participants
- More conventional approach within distributed systems
- Stronger consistency and lower latency
- Smaller number of participants
- Used in permissioned blockchains
- All participants agree on the list of participants in the network



# Other Alternatives

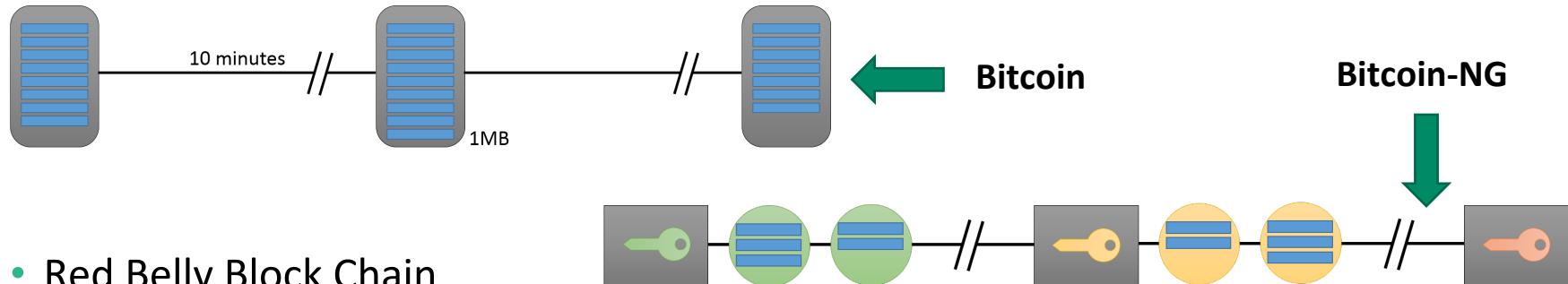
- Proof-of-Retrievability (Permacoin)
  - In proportion to distributed storage of archival data
- Proof-of-Elapsed Time
  - Miner waiting a random time to write the next block
  - Using Intel SGX (Software Guard Extension)
    - Run trusted code in trusted environment
    - Ensure the wait times are created fairly

# Consensus Overview

	Option	Fundamental properties	Cost efficiency	Impact	
	Performance	Flexibility			
Security-wise	Proof-of-work	⊕⊕⊕	⊕	⊕	⊕
	Proof-of-retrievability	⊕⊕⊕	⊕	⊕	⊕
	Proof-of-stake	⊕⊕	⊕⊕	⊕⊕	⊕⊕⊕
Scalability-wise	Practical Byzantine Fault Tolerance (PBFT)	⊕	⊕⊕⊕	⊕⊕⊕	⊕
	Bitcoin-NG	⊕⊕⊕	⊕	⊕	⊕
	RBBC	⊕⊕	⊕⊕⊕	⊕⊕⊕	⊕

# Scalability-wise

- Bitcoin-NG
  - Decouple Bitcoin's operation into two planes: Leader election and transaction serialisation
  - Selected leader is entitled to serialize transactions until the next leader is selected



- Red Belly Block Chain
  - Democratic Byzantine consensus without leader nodes
  - Transactions being collected by a set of proposers
  - Proposers collectively decide on a proposed set of transaction to send to a verifier
  - Verifier enforces consensus using hashes exchanged for the proposed sets of transactions

# Taxonomy Dimensions

(De)centralization

Deployment

Ledger Structure

Consensus Protocol

## **Block Configuration and Data Structure**

Auxiliary Blockchain

Anonymity

Incentive



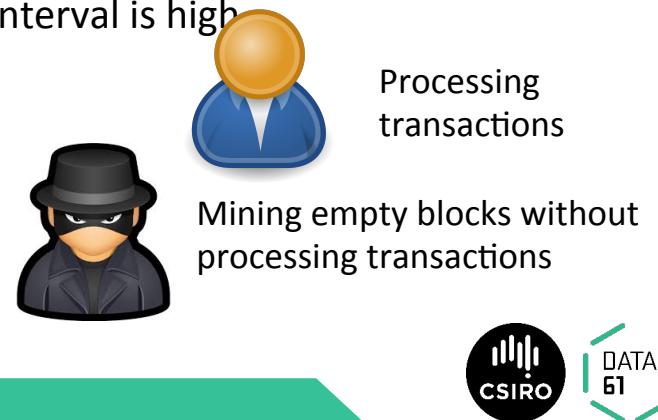
# Block Configuration 1/2

Option	Impact			
	Fundamental properties	Cost efficiency	Performance	Flexibility
Original block size and frequency	⊕⊕	n/a	⊕	n/a
Increase block size / Decrease mining time	⊕	n/a	⊕⊕	n/a

- Adjust mining difficulty to shorten the block time interval
  - Reducing latency
  - Increasing throughput
  - Increased frequency of forks
    - Ethereum has shorter block time interval (10-20 seconds) than Bitcoin (10 minutes)
    - Ethereum needs more confirmation blocks than Bitcoin

# Block Configuration 2/2

- Block size limit
  - Data size in MB (Bitcoin): Proposal for Bitcoin to increase block size from 1MB to 8 MB
  - Gas limit (Ethereum): Limit the complexity of the contained transactions
- Decision on block size is subject to tradeoffs
  - Speed of replication, block time interval and throughput
  - Mining new block can not start before observing the latest block
    - State changes as a result of the new block
- Unlimited block size causes DoS
  - Flooding system with transactions such that the block time interval is high
- High block size increase the risk of empty blocks
  - It is economical to mine as many empty blocks as possible
    - If block limit and block mining reward is high
  - Deteriorates the value of the network
    - Not processing new transaction anymore



# Block Data Structure 1/3

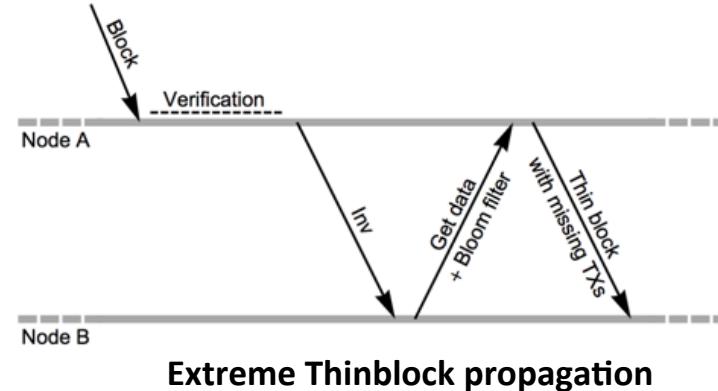
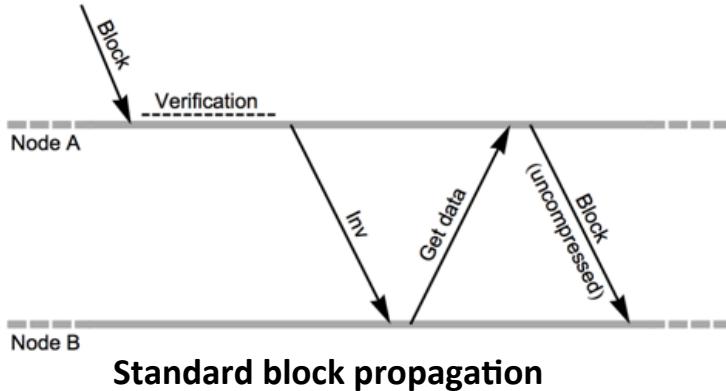


- Block size limit might not be a consensus rule in the first place
  - A hard limit on the block size chokes the growth of transaction
  - Changing the limit requires a change of the consensus rules
    - Developers decide a new block size limit
    - Merge code and hope the ecosystem follows (can be rejected by miners)
- Bitcoin unlimited
  - Removed block size from the consensus rules
  - The maximum size of a block is freely adjustable by the miners
  - New Bitcoin client that helps the system to scale

# Block Data Structure 2/3



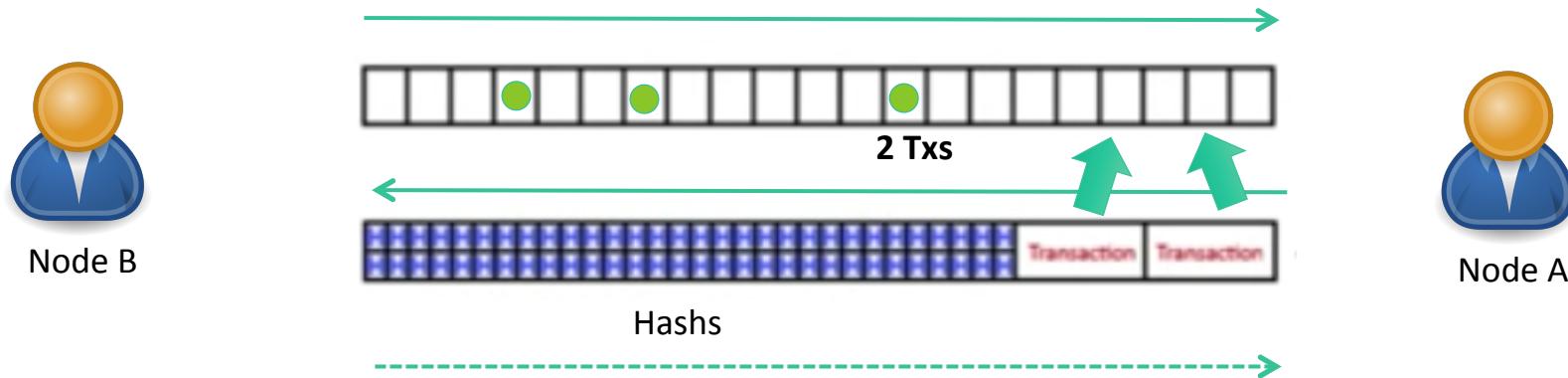
- Bitcoin Nodes
  - Keeping a list of unconfirmed transaction in memory (*mempool*)
  - When a new block is minded, its transactions must be relayed between nodes
- Bitcoin unlimited Nodes with Xtreme Thinblock
  - Avoid sending transactions again
  - Using a Bloom filter to reduce transmission sizes
    - Images the *mempool* onto a Bloom filter



# Block Data Structure 3/3



- Bloom filter
  - An array of Boolean
  - Designed to indicate whether an element is present in a set
  - Probabilistic data structure
    - Either definitely is not in the set or may be in the set



# Taxonomy Dimensions

(De)centralization  
Deployment  
Ledger Structure  
Consensus Protocol  
Block Configuration and Data Structure  
**Auxiliary Blockchain**  
Anonymity  
Incentive



# Tradeoff Overview

	Option	Fundamental properties	Cost efficiency	Impact	
				Performance	Flexibility
Security-wise	Merged mining	⊕⊕⊕	⊕⊕	⊕	⊕
	Hook into popular blockchain at transaction level	⊕⊕	⊕	⊕⊕	⊕⊕⊕
	Proof-of-burn	⊕	⊕	⊕⊕⊕	⊕⊕
Scalability-wise	Sidechains	⊕⊕⊕	⊕	⊕	⊕
	Multiple private blockchains	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

# Merged Mining 1/5

- Mining is ordinarily exclusive
- Each attempt either has a chance to be a parent chain block or has a chance to be a new chain block
- Obstacle to bootstrapping
- Miner splits mining resource to mine a new blockchain
- Miner switch between parent chain and new chain

**Hash( prev || merkle\_root || nonce) < TARGET**

Parent chain (Bitcoin)

**Hash( prev || merkle\_root || nonce) < TARGET**

New chain

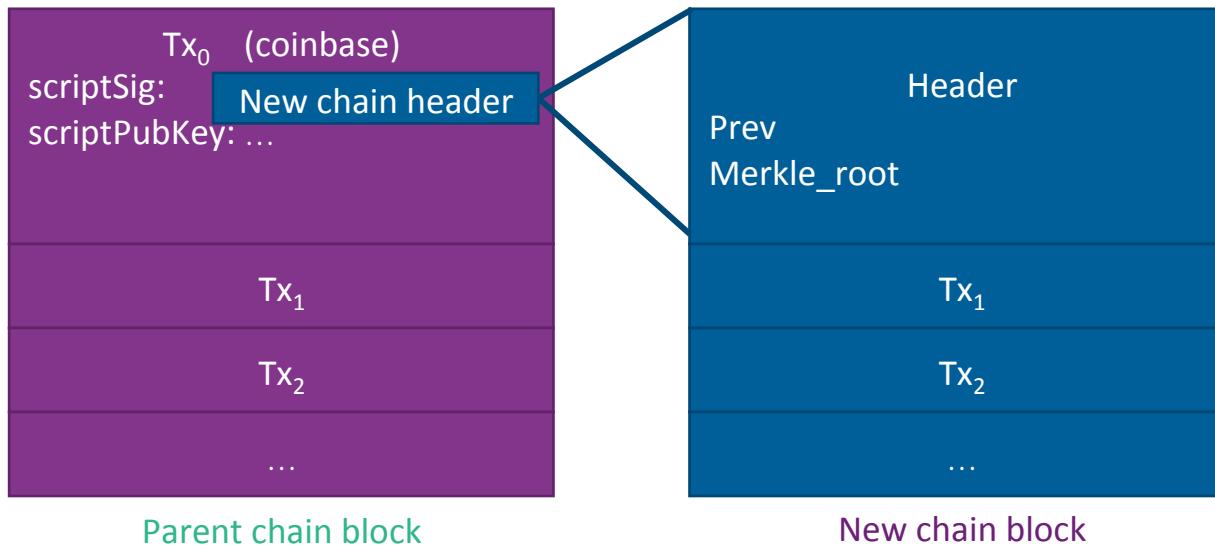
# Merged Mining 2/5

$\text{Hash}(\text{ prev } \parallel \text{ merkle\_root } \parallel \text{ nonce}) < \text{TARGET}$

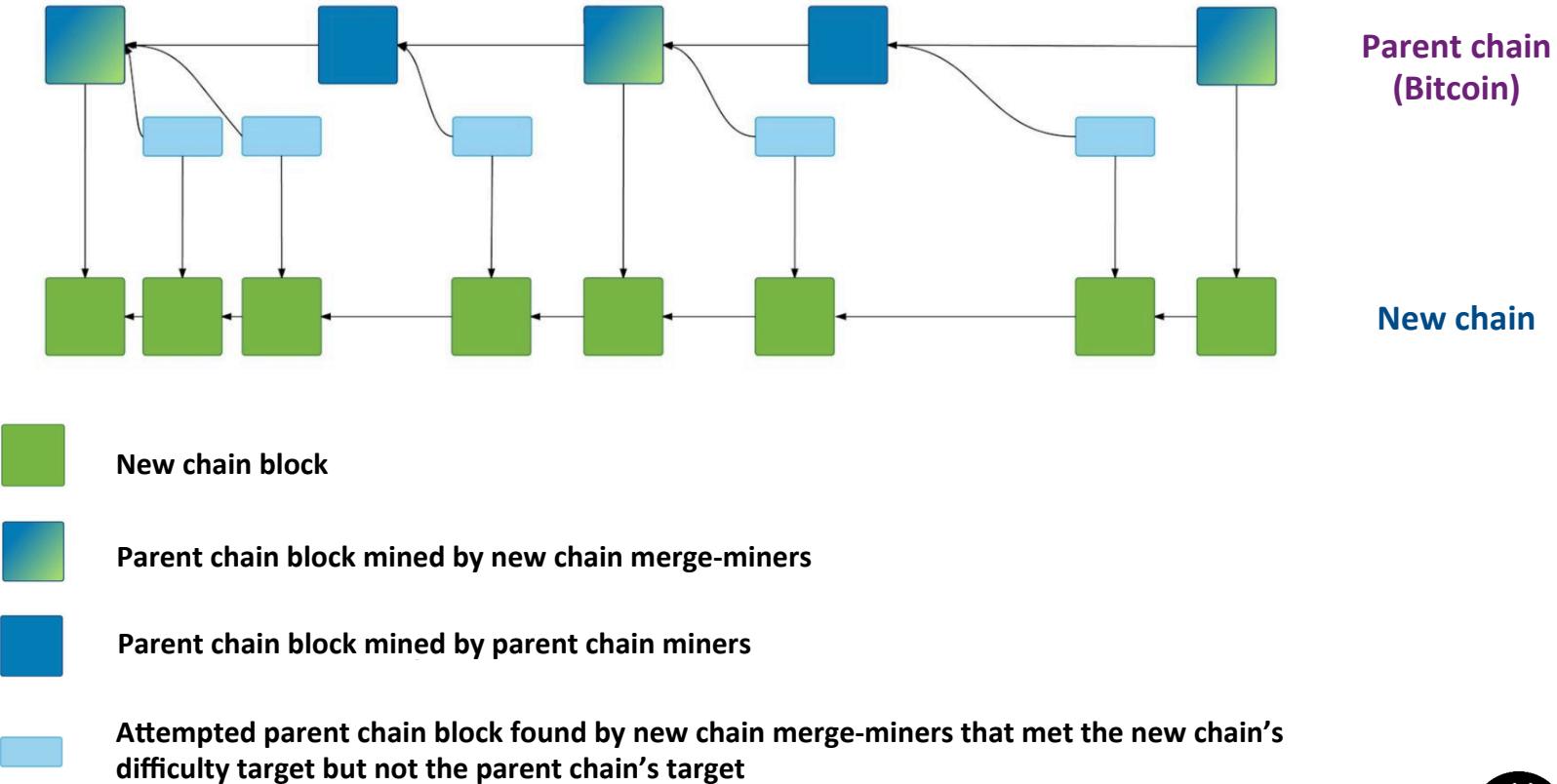
Parent chain (bitcoin)

$\text{Hash}(\text{ prev } \parallel \text{ merkle\_root } \parallel \text{ nonce}) < \text{TARGET}$

New chain



# Merged Mining 3/5



# Merged Mining 4/5

- Parent chain client
  - Block with encoded new chain block looks like any other Bitcoin block
  - Hash in *scriptSig* is ignored
  - No change
- New chain client
  - Interpret new chain block
    - Ignore parent chain transactions
  - Understand parent chain block
    - Be able to verify the “work”

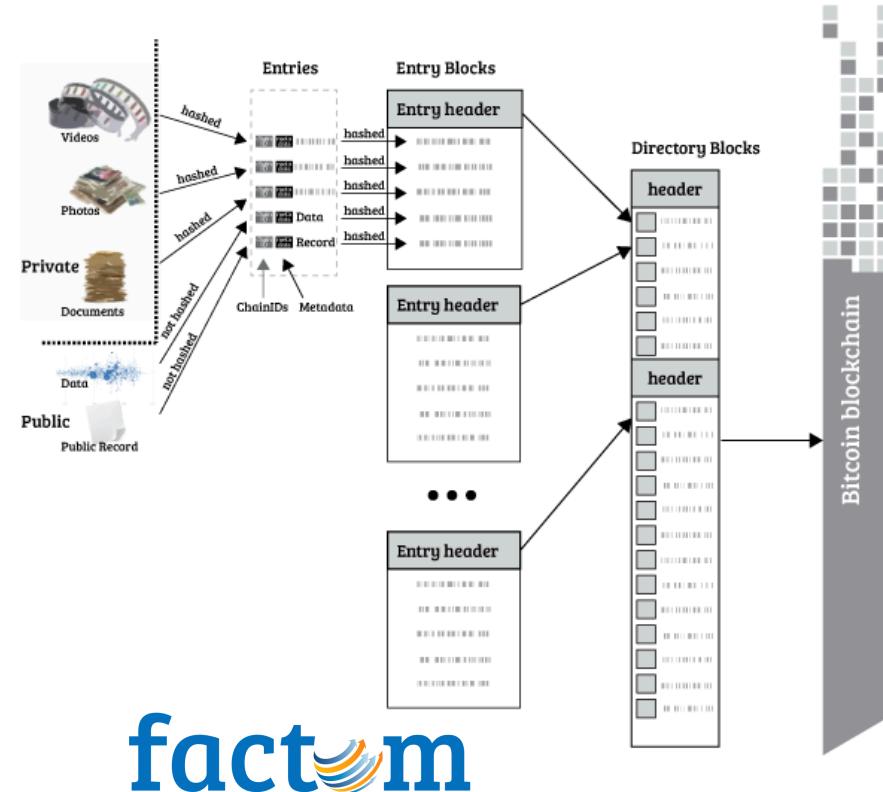
# Merged Mining 5/5



- Namecoin is the first blockchain uses merged mining with Bitcoin
- A small miner (or mining pool) on a large network can demolish a merged mining blockchain
  - Jan 2012: CoiledCoin – Eligius pool (large mining pool)
    - Eligius decided that CoiledCoin is a scam
    - Launch an attack to mine a lot of blocks that reversed days of transaction history of CoiledCoin
    - A long chain with empty blocks containing no transaction to make DoS
    - Cheaper for attackers
  - Jul 2013: TerraCoin – unknown
  - Nov 2013: WorldCoin – unknown
- Many mining pools merge-mine several sidechains
  - Ghash.IO: Bitcoin, Namecoin, IXCoin, Devcoin

# Hooking into Public Blockchain at Transaction Level

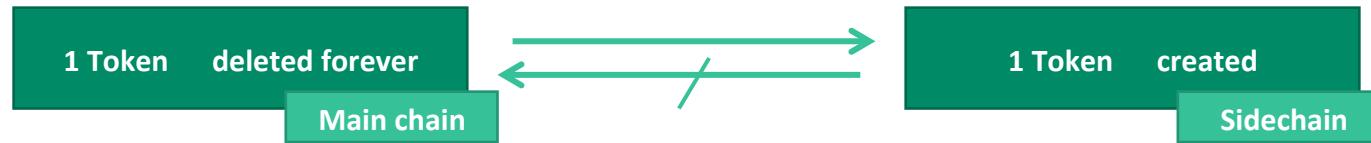
- Hash of new chain is embedded in transaction of parent blockchain (*OP\_RETURN*)



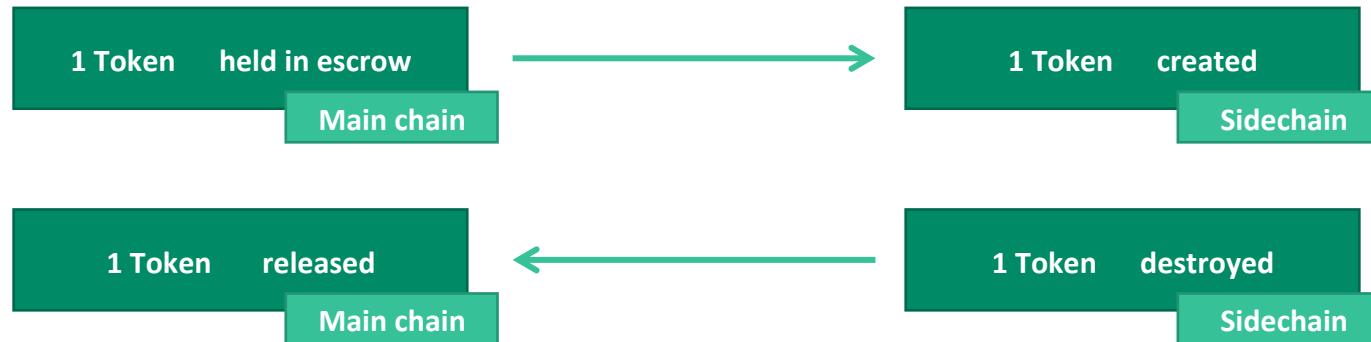
factom

# Sidechain 1/5

- Unilateral peg



- Bilateral peg



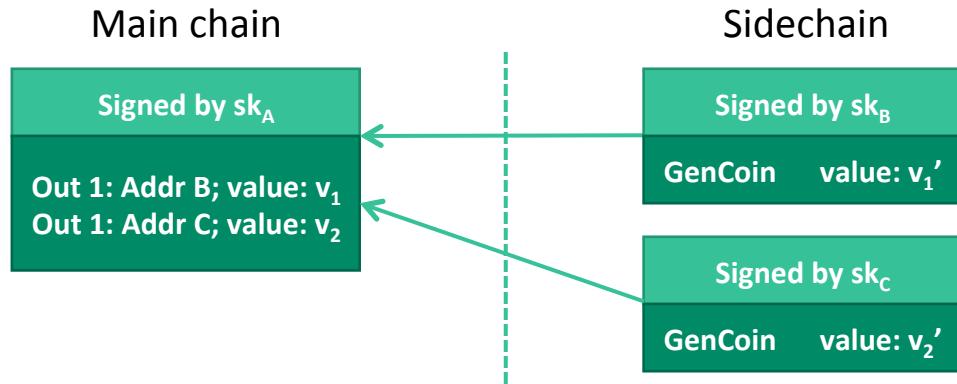
# Sidechain 2/5

- Proof-of-Burn



- Signed by the same private key
- Same signature scheme

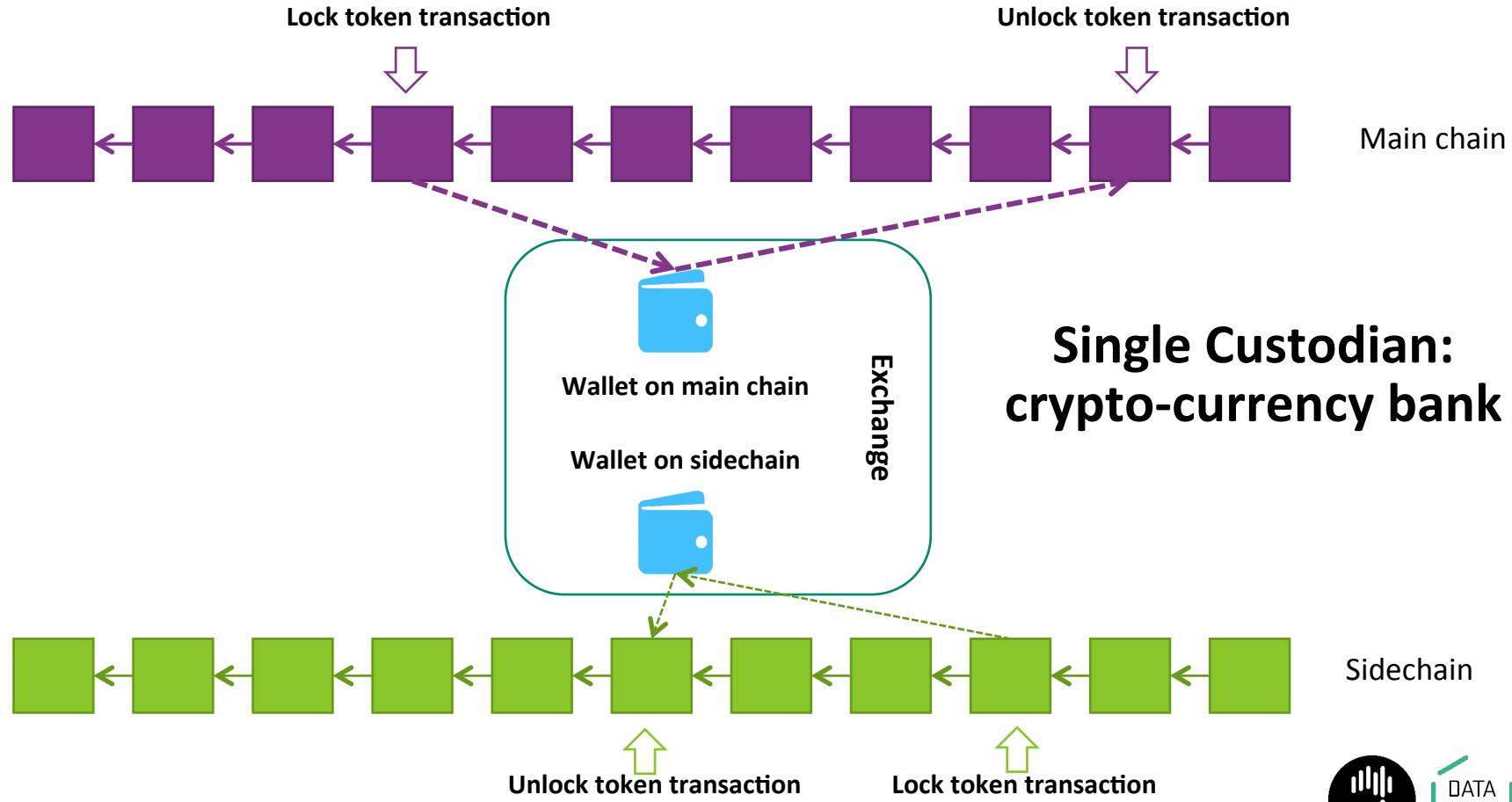
- Snapshot



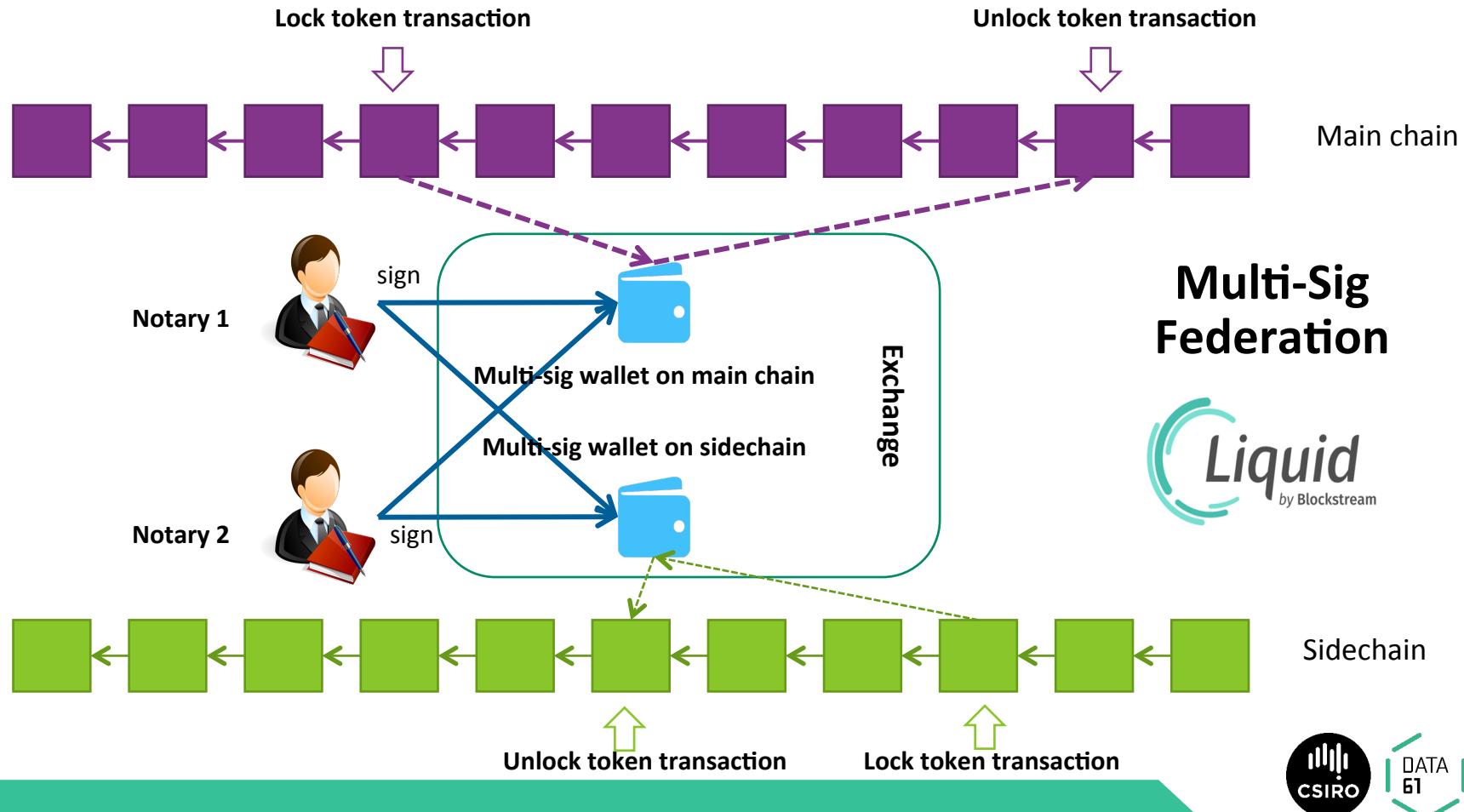
# Sidechain 3/5

- *Bilateral pegged sidechain is a voting system*
- A group of custodians cast vote on
  - When to unlock tokens
  - Where to send the unlocked tokens to
- Single Custodian
- Multi-sig Federation

# Sidechain 4/5



# Sidechain 5/5



# Problems of Sidechain

- Public blockchains do not have settlement finality
  - Main chain can never be 100% sure if a sidechain transaction has been accepted by the network
  - Neither does sidechain
- Bitcoin-backed sidechains require a soft-fork or hard-fork for Bitcoin to add new complex *opcode*

# Entangled Blockchains

- Entangle both chains to overcome the lack of transaction finality
- Reversal of the lock transaction in main chain → Reversal of the unlock transaction in side chain
- Sidechain transaction is embedded in transaction of parent blockchain (*OP\_RETURN*)
- Sidechain block has extra parent in the parent blockchain
  - Sidechain nodes verify that parents
- Sidechain client maintains a copy of parent blockchain

# Tradeoff Overview

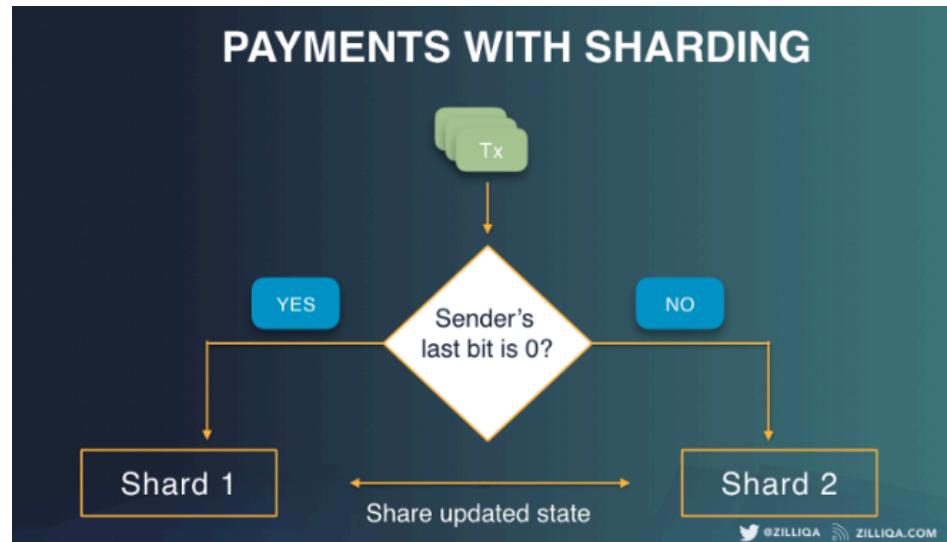
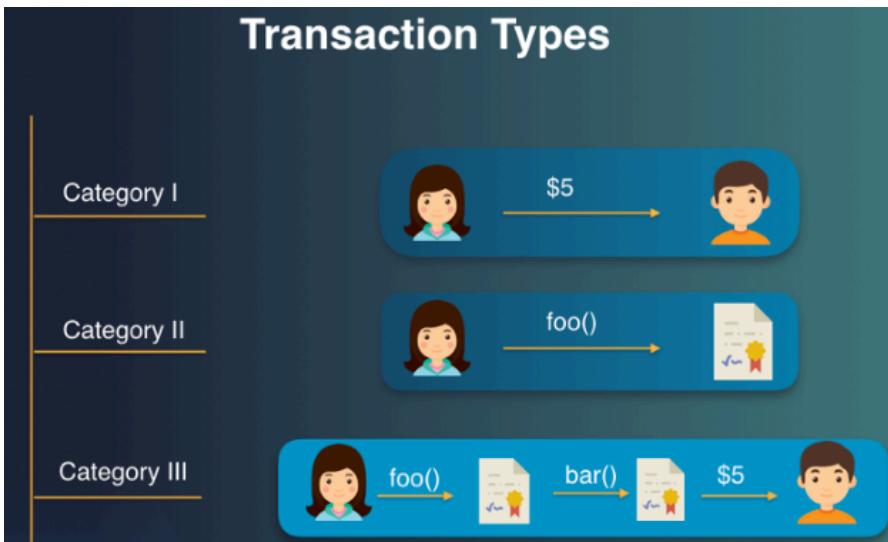
	Option	Fundamental properties	Cost efficiency	Impact	
	Performance	Flexibility			
Security-wise	Merged mining	⊕⊕⊕	⊕⊕	⊕	⊕
	Hook into popular blockchain at transaction level	⊕⊕	⊕	⊕⊕	⊕⊕⊕
Scalability-wise	Proof-of-burn	⊕	⊕	⊕⊕⊕	⊕⊕
	Sidechains	⊕⊕⊕	⊕	⊕	⊕
	Multiple private blockchains	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

# Sharding 1/5

- Full nodes require sizeable storage space
  - Bitcoin > 200GB
  - Ethereum > 600GB
  - Keep growing
- Sharding
  - Divide the blockchain state into pieces
  - Blockchain nodes only hold some shards
  - Transaction sharding vs. State sharding

# Sharding 2/5

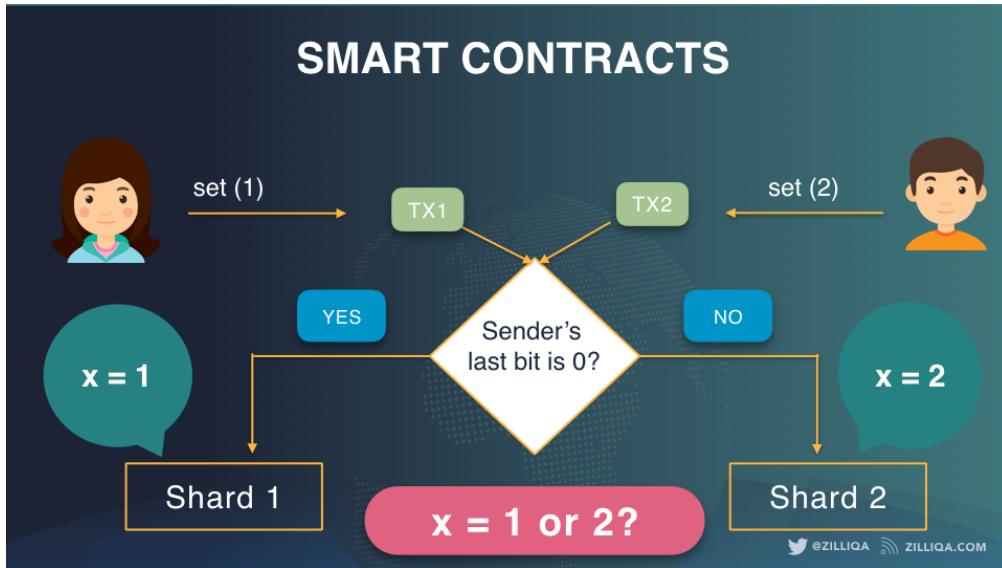
- Transaction Sharding



# Sharding 3/5



- Conflicting state
- Using the recipient address instead
- Smart contract address



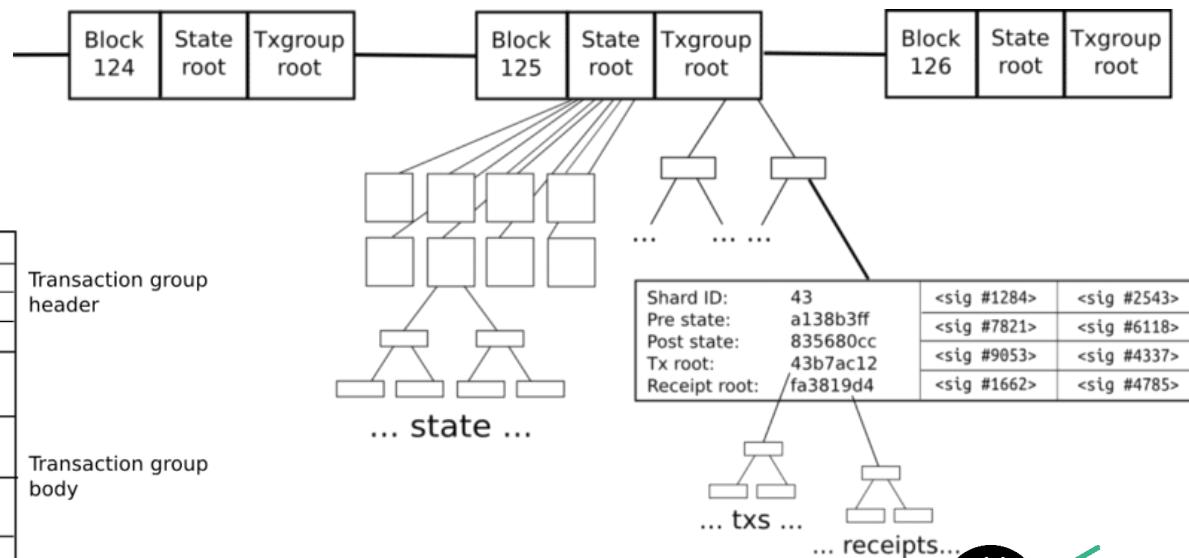
# Sharding 4/5



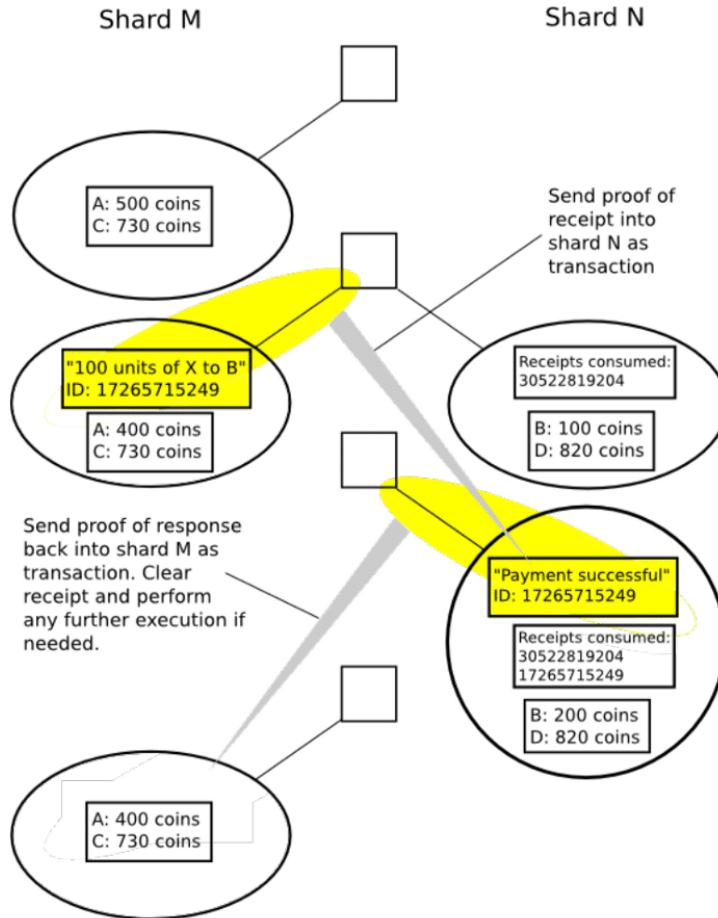
- After sharding is activated
    - State sharding
    - Each unique account is in one shard
    - Accounts can only transact with other accounts within the same shard
  - Two levels of interaction

Random validator who need to verify the transactions in the shard

Shard ID:	43	<sig #1284>	<sig #2543>
Pre state:	a138b3ff	<sig #7821>	<sig #6118>
Post state:	835680cc	<sig #9053>	<sig #4337>
Receipt root:	fa3819d4	<sig #1662>	<sig #4785>
Tx a142	Tx a558	Tx eca6	
Tx a35f	Tx e25a	Tx 34ac	
Tx 2308	Tx 6987	Tx f260	
Tx 9f14	Tx ec30	Tx 5fc3	



# Sharding 5/5



- **Transaction**
  - Change the state of shard
  - Generate a receipt
- **Receipts**
  - Stored in a distributed shared memory
  - Seen by other shards
  - Not modified
- **Receipts enable cross-shard communication**
  - Accessed via multiple Merkle trees from the the transaction group Merkle root

# Taxonomy Dimensions

(De)centralization  
Deployment  
Ledger Structure  
Consensus Protocol  
Block Configuration and Data Structure  
Auxiliary Blockchain  
**Anonymity**  
Incentive



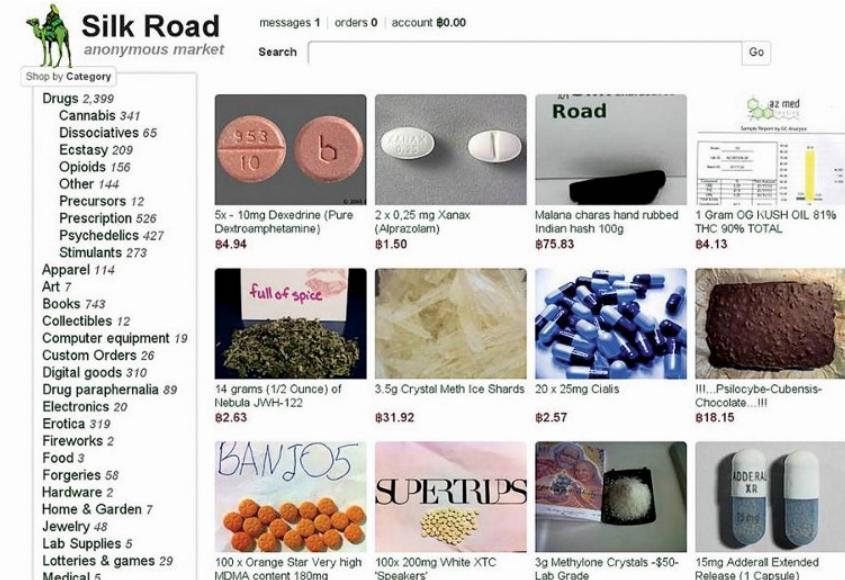
# Pseudonymity

- Bitcoin is pseudonymous (pseudo-anonymous)
  - Physical world evidence can be connected to compromise the anonymity of Bitcoin users
- Silk Road
  - “eBay for drugs”
  - Cash flow of at least USD213 million
  - Combination Tor and Bitcoin
- Linking visual world and physical world
  - Exposed physical IP address of the Silk Road server
    - Iceland
  - Linked Ross Ulbricht with “Dread Pirate Roberts”
  - FBI became the second largest owner of Bitcoins
    - 144, 000 Bitcoins



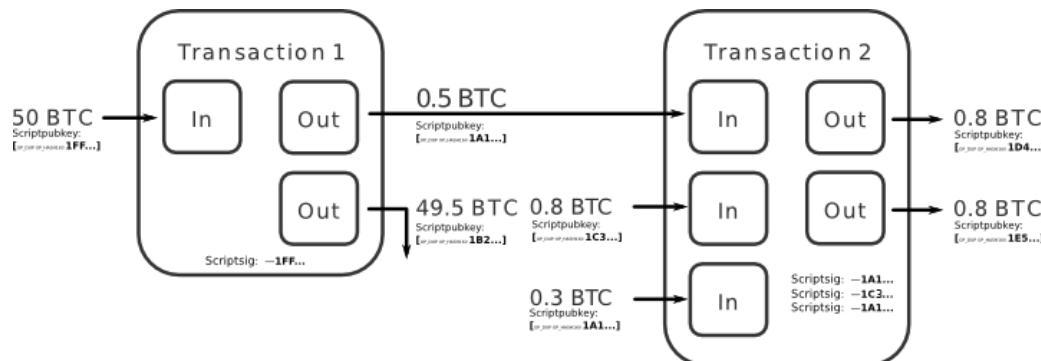
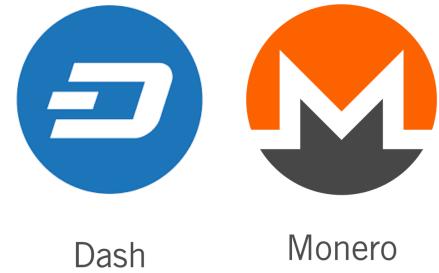
Tor is a intermediary server

- Hidden IP address
- Hidden users surfing on it



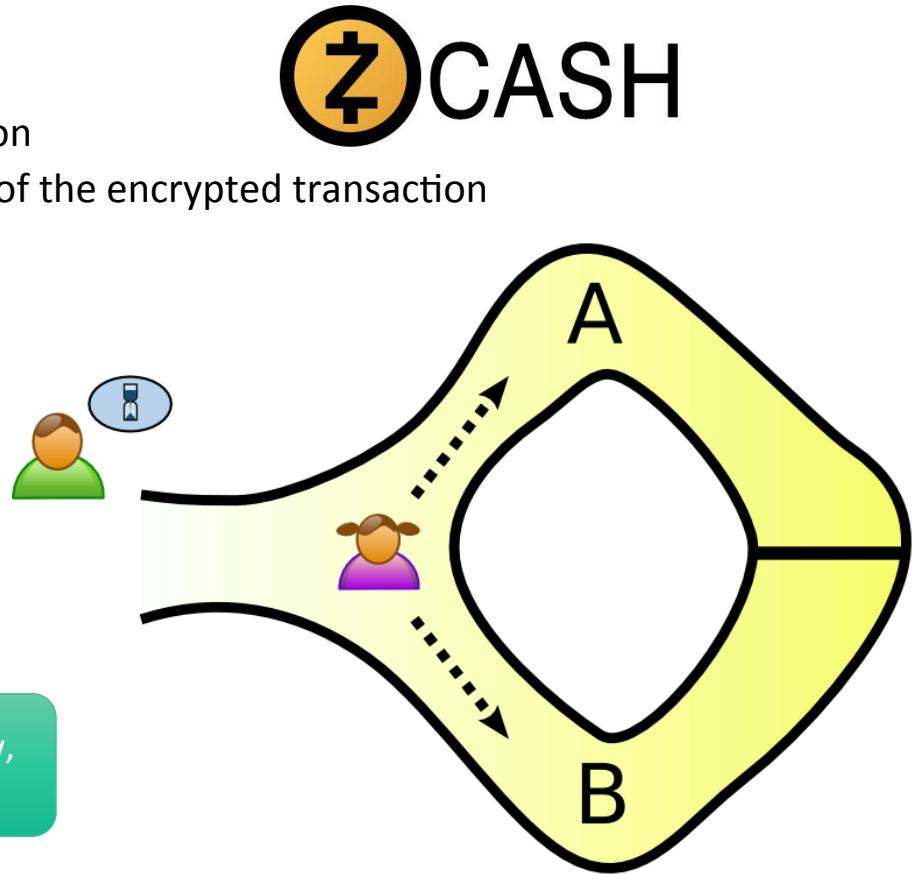
# Mixing Services

- Mixing service
  - Group transactions together with multiple *input* addresses and *output* addresses
  - Hard to track which output address is paid by which input address
  - A series of mixing services can be linked sequentially
  - Mixed transactions are uniformed in value
- Dash: Pre-anonymize funds through mixing rounds
- Monero: Ring signature



# Advanced Cryptographic Technology

- Zcash
  - Encrypted payment information in transaction
  - Cryptographical method to verify the validity of the encrypted transaction
    - Zero-knowledge proof
- Zero-knowledge proof
  - Allow the blockchain network to maintain a secure ledger
  - Enable private payment without disclosing the parties or amounts involved



You have a secret that you don't want others to know, but you want to proof you have the secret

# Taxonomy Dimensions

- (De)centralization
- Deployment
- Ledger Structure
- Consensus Protocol
- Block Configuration and Data Structure
- Auxiliary Blockchain
- Anonymity
- Incentive**



DATA  
61

# Incentive

- Financial incentive in the cryptocurrencies
  - Join the network
  - Validate transactions
  - Generate blocks
  - Execute smart contract
- Bitcoin
  - Reward for generating new blocks and fees associated with transactions
- Ethereum
  - Fee to execute smart contract
- Enigma
  - Fixed price for storage, data retrieval and computation within the network
  - Security deposit
    - Deposit of dishonest node will be split among the honest nodes



# Summary

- Blockchain Taxonomy
- (De)centralization
- Deployment
- Ledger Structure
- Consensus Protocol
- Block Configuration and Data Structure
- Auxiliary Blockchain
- Anonymity
- Incentive

# Course Outline – next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
3rd	4 Mar	Sherry Xu	Blockchain in Software Architecture 1	3. Varieties of blockchain 5. Blockchain in Software Architecture (including software architecture basics) 1/2	
4th	11 Mar	Mark Staples	Blockchain in Software Architecture 2	5. Blockchain in Software Architecture (Non-functional properties and trade-offs) 2/2	Pitching session
5th	18 Mar	Sherry Xu	NPFs 1	6. Design Process for Applications on Blockchain 9. Cost	Assignment 1 due 24 March



# Consultation Time

**Xiwei Xu** | Senior Research Scientist

Architecture & Analytics Platforms (AAP) team

t +61 2 9490 5664

e xiwei.xu@data61.csiro.au

w www.data61.csiro.au/

[www.data61.csiro.au](http://www.data61.csiro.au)



# Blockchain in Software Architecture 2: NFPs and design trade-offs

Mark Staples

Email: [mark.staples@data61.csiro.au](mailto:mark.staples@data61.csiro.au)

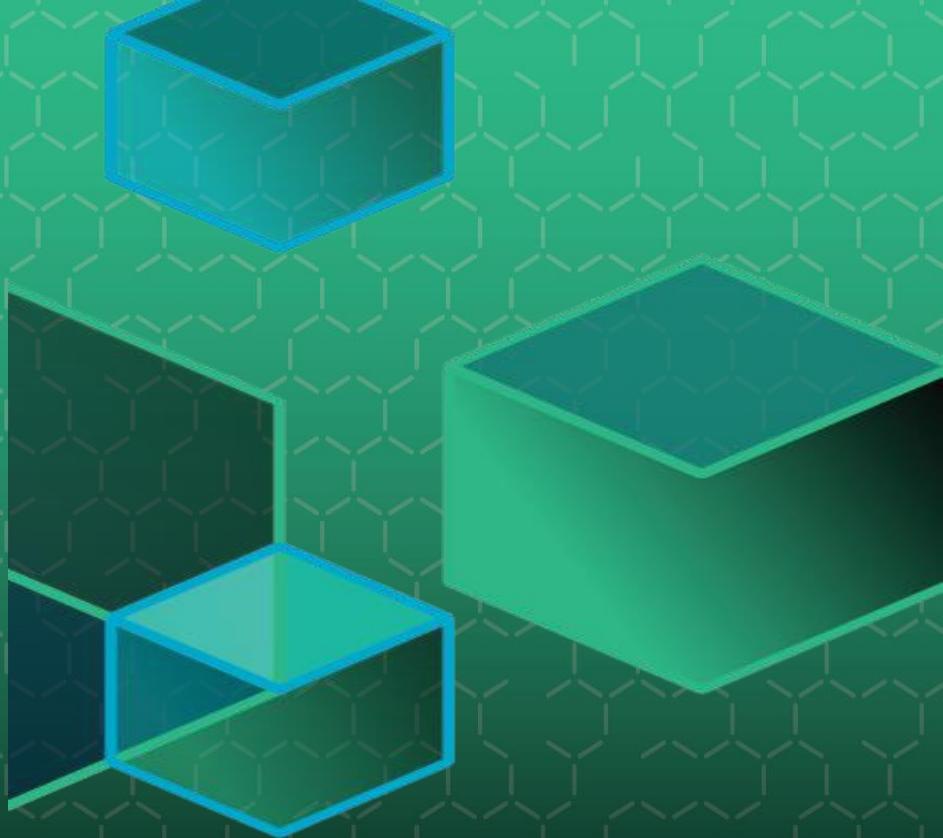
[www.data61.csiro.au](http://www.data61.csiro.au)

# Outline

- PITCH SESSION!
- What is Software Architecture?
  - Components, Connectors, Configuration
  - Non-Functional Properties (NFPs)
  - Models: Views and Viewpoints
  - NFP Analysis and Trade-offs
- Blockchain in Software Architecture
  - Blockchain as Component, Connector, Configuration
  - Blockchain Component Services
- Design and Trade-offs in Blockchain-based Applications
- Summary



# What is Software Architecture?



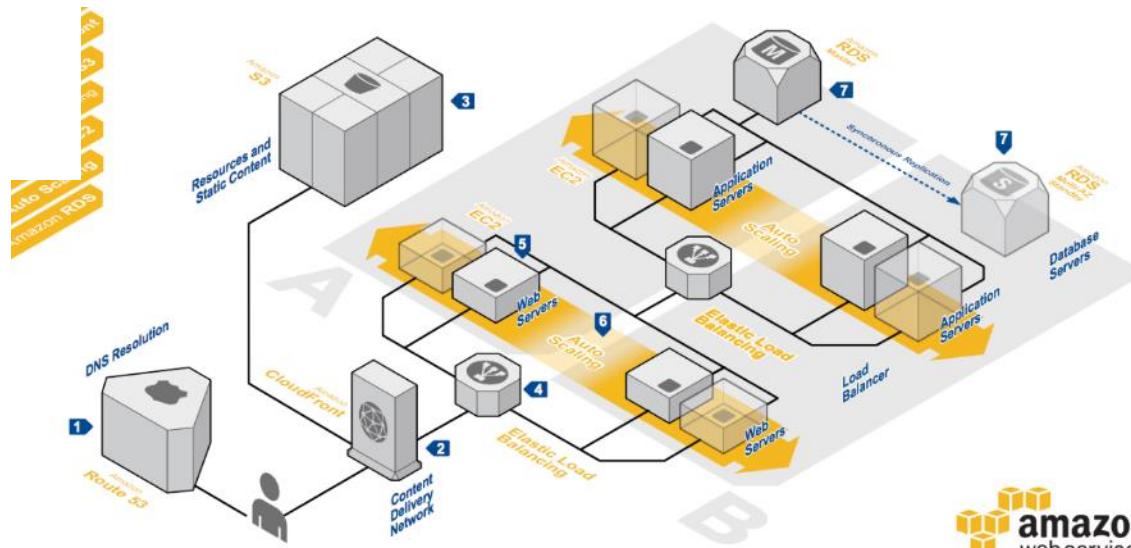
# Every software system has a software architecture

- “*The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both*”
- Diagrams/ Documentation
- Designing
  - (But not all design is architecture)
- Discipline/ Practice/ Profession
- Discovery (Research Area)
  - Starting in 1960s, increasing interest since 1990s



# Software Architecture Elements

- A software system's architecture is typically not a uniform monolith
  - Component, Connector, and Configuration



# Software Component

- Software components are the fundamental building blocks for software architecture
- A software component is an architectural entity that
  - Encapsulates a subset of the system's functionality and/or data
  - Restricts access to that subset via an explicitly defined interface
  - Has explicitly defined dependencies on its required execution context
- Components typically provide application-specific services

# Software Connectors

- In complex systems interaction may become more important and challenging than the functionality of the individual components
- Architectural building block tasked with effecting and regulating interactions among components
- Connectors typically provide application-independent interaction facilities
  - **Communication:** transfer data
  - **Coordination:** transfer control
  - **Facilitation:** enable and optimise component's interactions
  - **Conversion:** adjust the interactions between incompatible interfaces

# Example Software Connectors



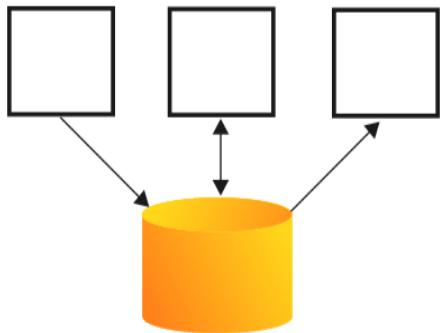
File Transfer



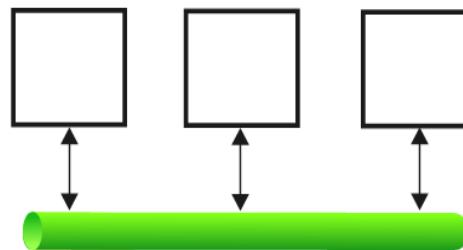
Stream



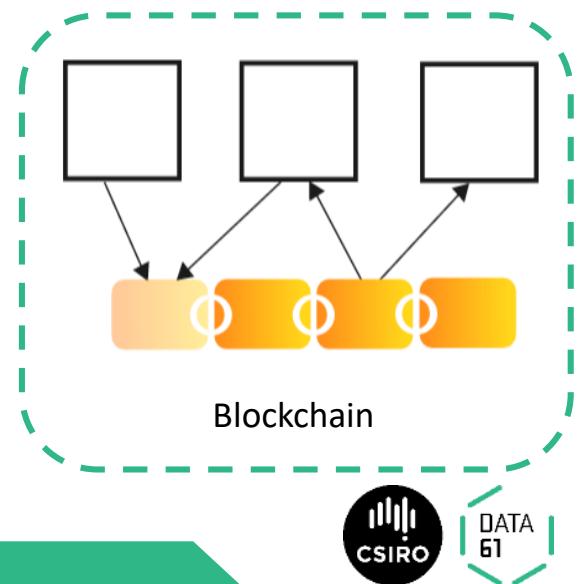
Remote Procedure Call



Shared Database



Message Bus



Blockchain

# Architecture defines system structure

- Decomposition of system into components/modules/subsystems
- Architecture defines:
  - Component interfaces
    - What a component can do
  - Component responsibilities
    - Precisely what a component will do when you ask it
  - Component connections and dependencies
    - How components are related to each other through their interfaces
- Can we infer system properties from element properties?

# Architecture specifies component communication

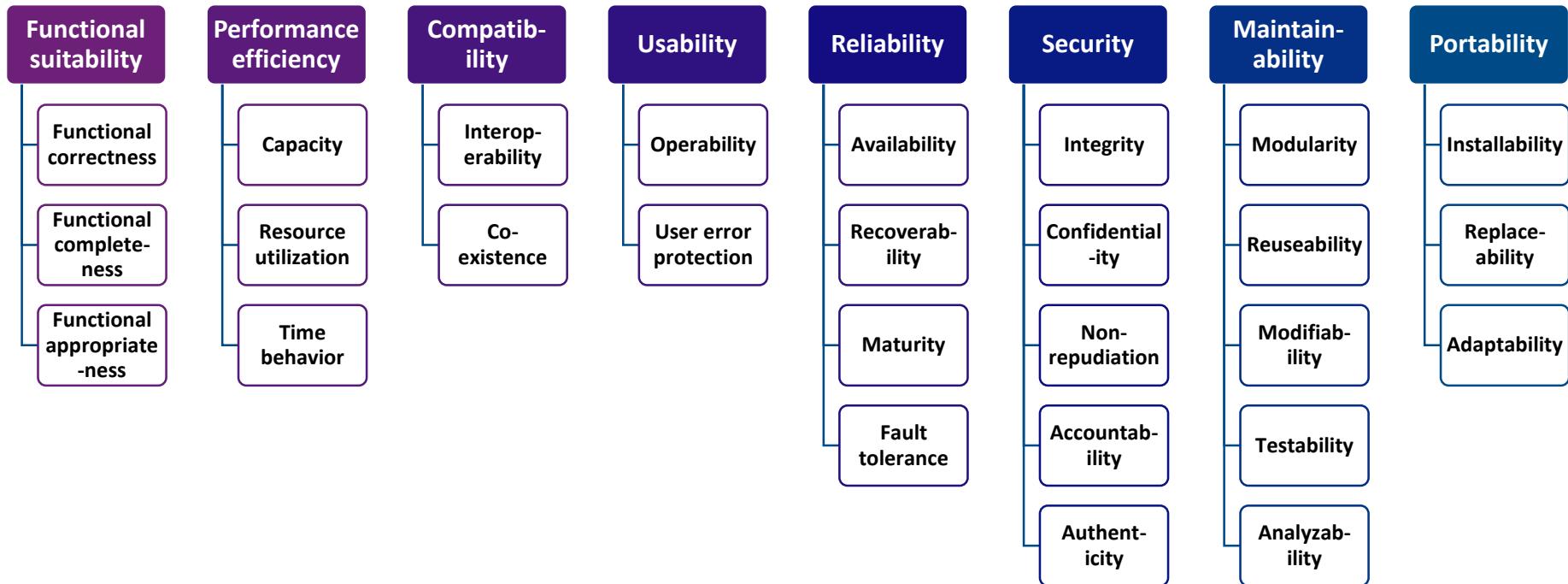
- Data passing mechanisms, e.g.:
  - Function call
  - Remote method invocation
  - Asynchronous message
- Control flow
  - Flow of messages between components
  - Sequential
  - Concurrent/parallel
  - Synchronization



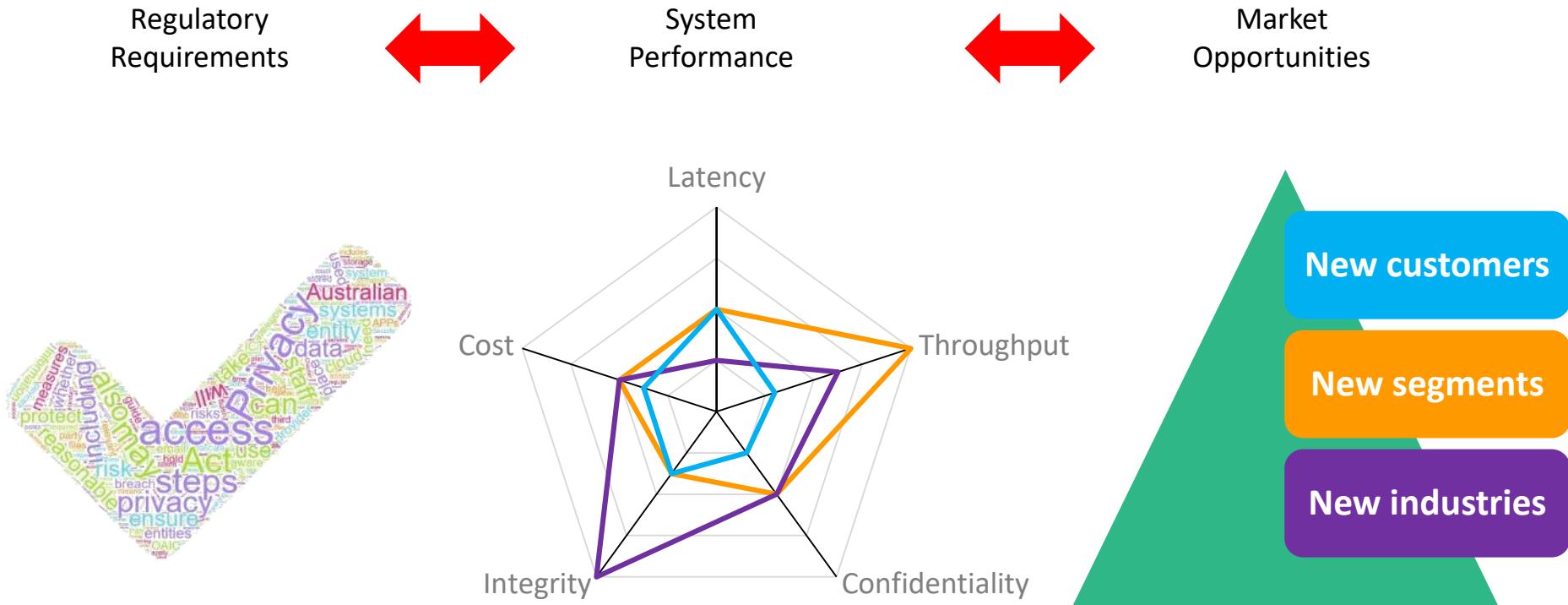
# Non-Functional Properties and Requirements

- There are two kinds of requirements:
  1. Functional Requirements (i.e. what are the inputs and outputs)
  2. Non-Functional Requirements (a.k.a. *Qualities*, or *-ilities*)
    - e.g. “Performance” (latency, throughput, ... )
    - e.g. “Security” (confidentiality, integrity, availability, privacy, ...)
    - e.g. Usability, Reliability, Modifiability, ...
    - Cost
    - Technical and business constraints

# ISO/IEC 25010:2011 Quality Model

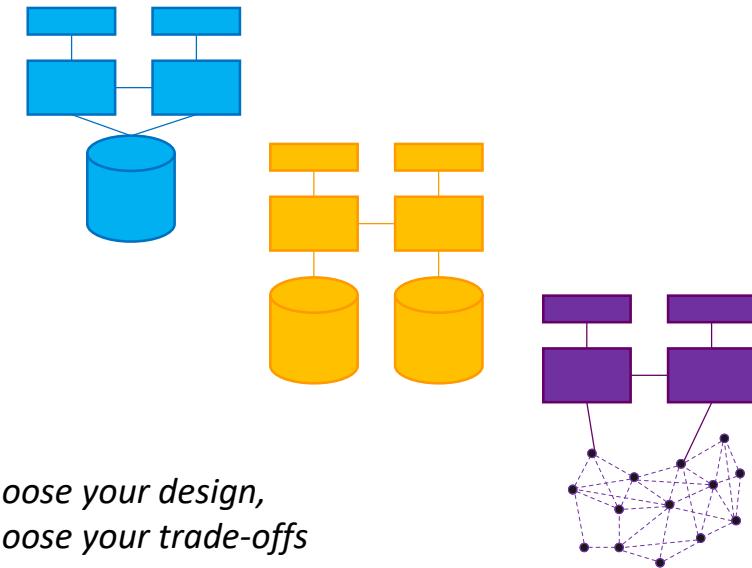
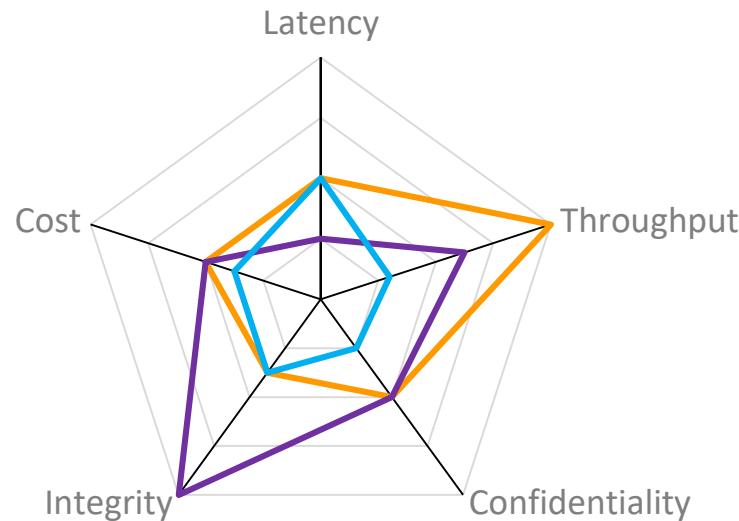


# Why Non-Functional Properties Matter

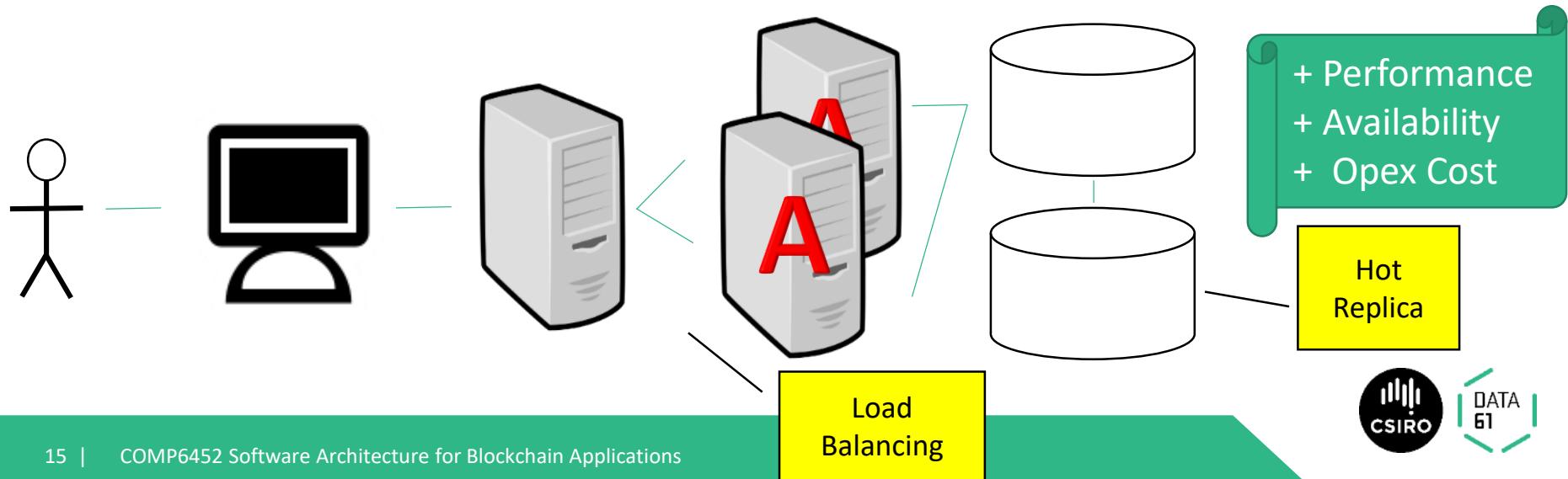
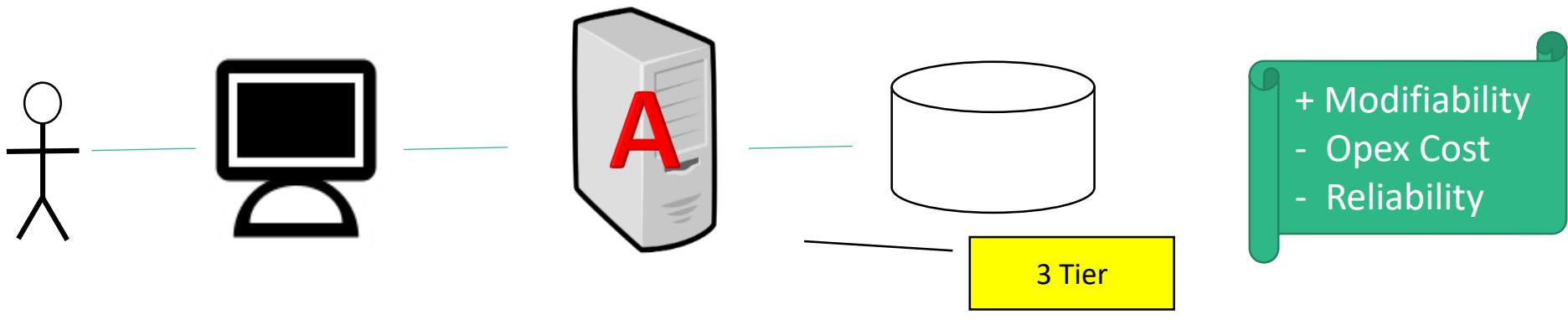


# Software Architecture

Non-Functional Properties arise from Architectural Design Choices



# For Example...

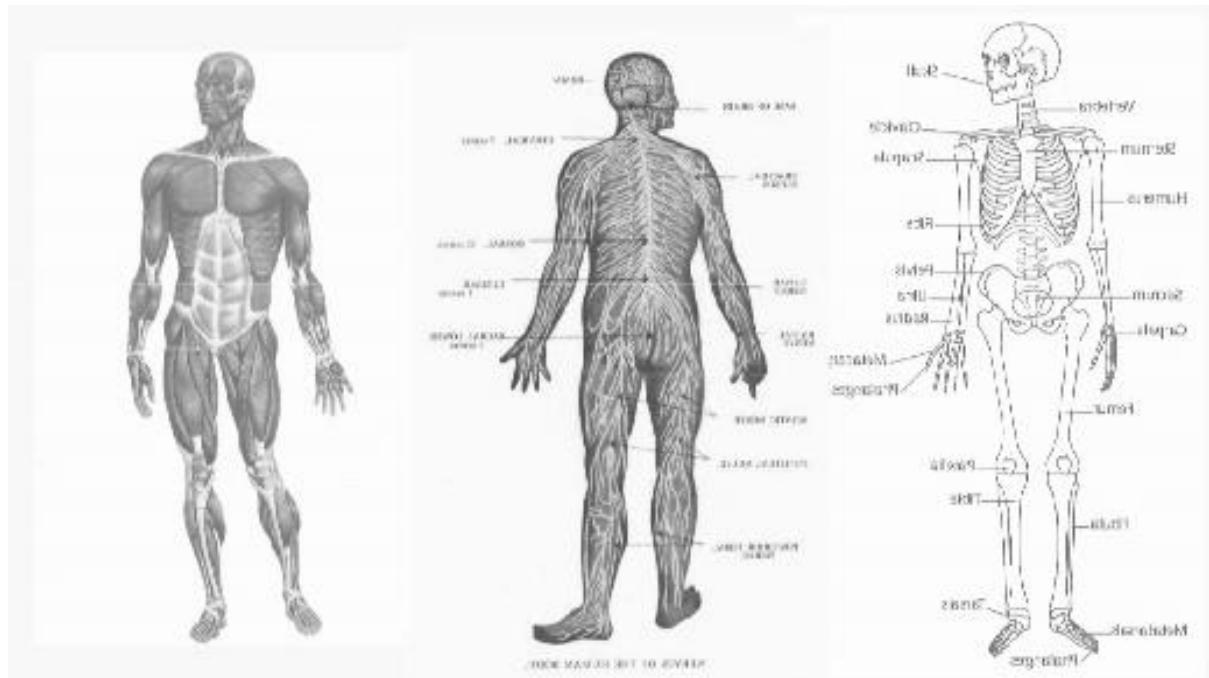


# Architecture is an Abstraction

- Architecture provides an abstract view of a design
  - Hides complexity of design
  - May or may not be a direct mapping between architecture elements and software elements
    - e.g. “marketecture”
- “All models are wrong, but some models are useful”
  - Box
- Discussion: Why Abstraction?

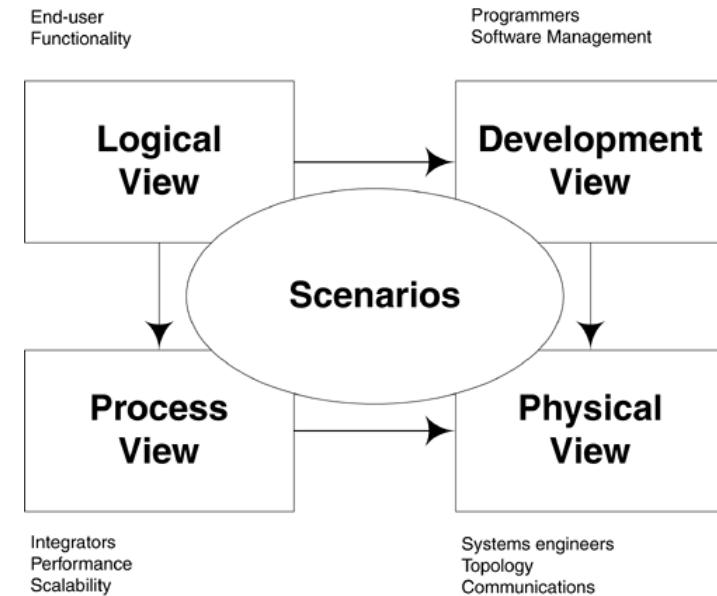


# Viewpoints and Views

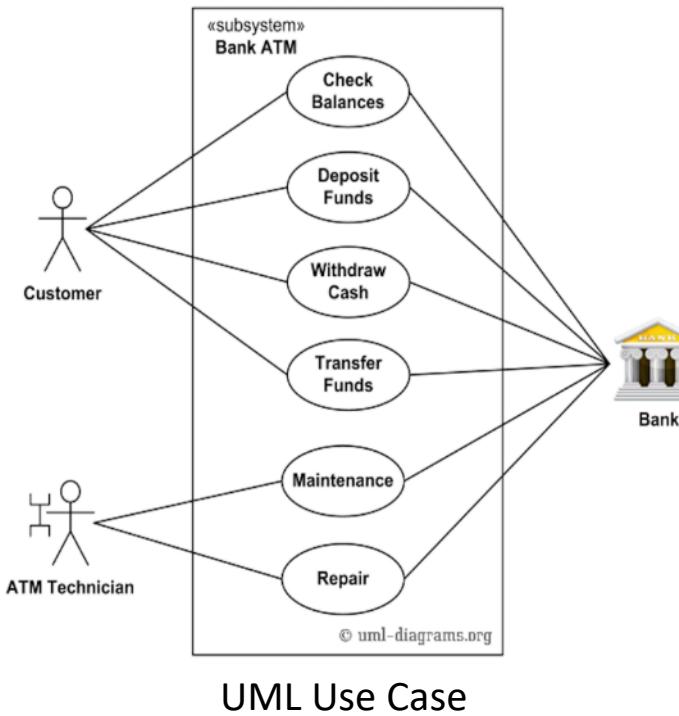


# Krutchens 4+1 View Model

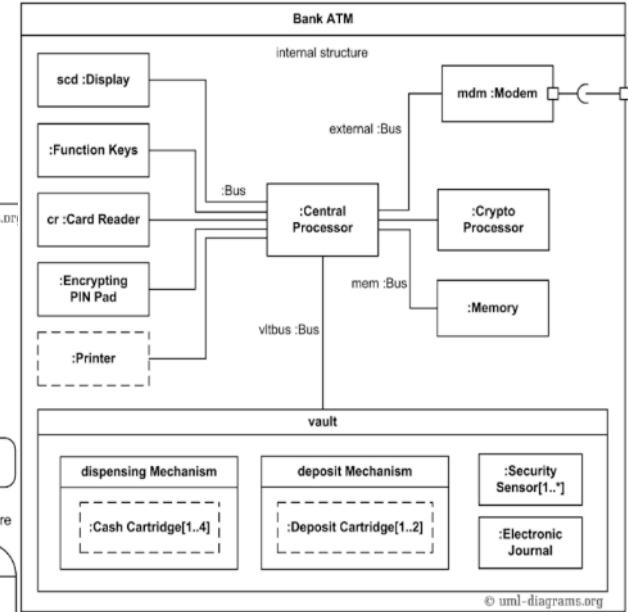
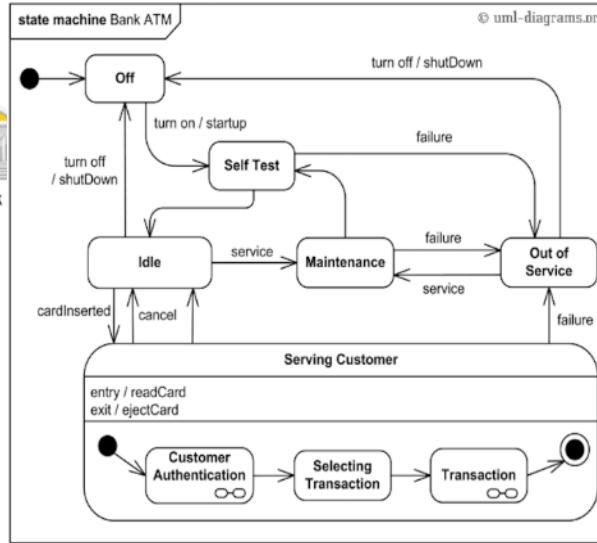
- **Logical:** architecturally-significant elements and the relationships between them.
- **Process:** concurrency and communications elements
- **Physical:** how the major processes and components are mapped to applications hardware
- **Development:** internal organization of the software components as held in e.g. a configuration management tool
- **Use cases:** requirements for the architecture; related to more than one particular view



# UML as Viewpoints and Views



UML Statechart



UML Composite Structure

# Architecture Analysis Methods

- Analysis is one of the important uses of models
- e.g. does this design support service availability?
  - (Quantitative) fault-tree statistical analysis
  - (Qualitative) failure scenarios
- e.g. What is the transaction latency in this design?
  - (Quantitative) Simulation-based prediction of latency distributions
  - (Quantitative) Formula-based calculation of average latency
- e.g. Will this design ensure confidentiality?
- e.g. Is this design easily modifiable?
- Different kinds of models allow different kinds of analyses

# Design Trade-offs

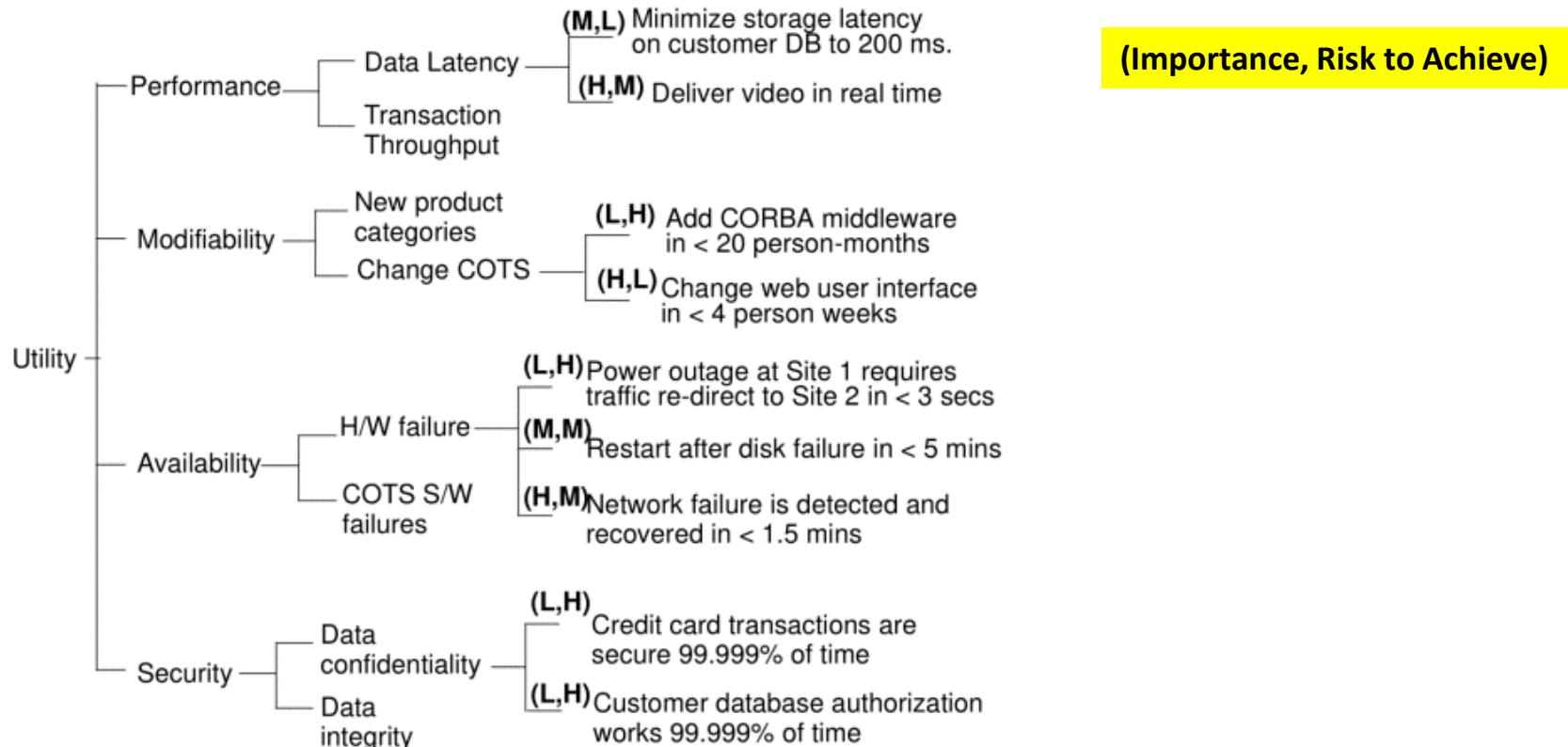
- Often, improving one quality will hurt another one
  - e.g. Use a bigger server
    - faster, but more expensive
  - e.g. Use redundant servers
    - better availability (hot fail-over)? Or better performance (load balancing)? Worse write latency? more expensive
- What are the trade-offs? How should you choose?
- Specific methods exist to help
  - ATAM: Architecture Trade-off Analysis Method
  - Multi-criteria decision-making methods

# ATAM

- Presentation
  1. Present ATAM
  2. Present Business Drivers (NFPs and other business goals)
  3. Present Architecture
- Investigation & Analysis
  4. Identify Architectural Approaches
  5. Generate Quality Attribute Utility Tree (& initial use case scenarios)
  6. Analyse Architectural Approaches
  7. Brainstorm & Prioritise Scenarios
  8. Analyse Architectural Approaches (using scenarios as test cases)
- Reporting
  9. Present Results (summary, risks, sensitivities/trade-offs, ...)



# ATAM Quality Attribute Utility Tree



Example from “ATAM: Method for Architecture Evaluation” (Kazman et al., 2000)

[https://resources.sei.cmu.edu/asset\\_files/TechnicalReport/2000\\_005\\_001\\_13706.pdf](https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf)

# ATAM Scenarios

**Scenario:** S12 (Detect and recover from HW failure of main switch.)

**Attribute:** Availability

**Environment:** normal operations

**Stimulus:** CPU failure

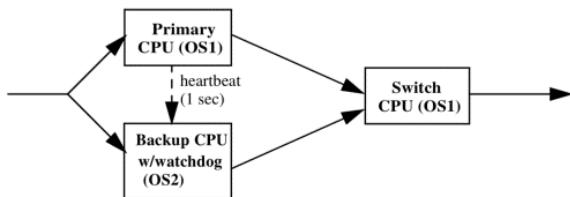
**Response:** 0.999999 availability of switch

Architectural decisions	Risk	Sensitivity	Tradeoff
Backup CPU(s)	R8	S2	
No backup Data Channel	R9	S3	T3
Watchdog		S4	
Heartbeat		S5	
Failover routing		S6	

**Reasoning:**

- ensures no common mode failure by using different hardware and operating system (see Risk 8)
- worst-case rollover is accomplished in 4 seconds as computing state takes ...
- guaranteed to detect failure with 2 seconds based on rates of heartbeat and watchdog ...
- watchdog is simple and proven reliable
- availability requirement might be at risk due to lack of backup data channel ... (see Risk 9)

Architecture diagram:



- Scenarios are commonly used in Software Architecture

- In ATAM, links attributes, risks, trade-offs, & reasoning about architecture design

- Qualitative – identify and evaluate risks

- Methodical, but not exhaustive

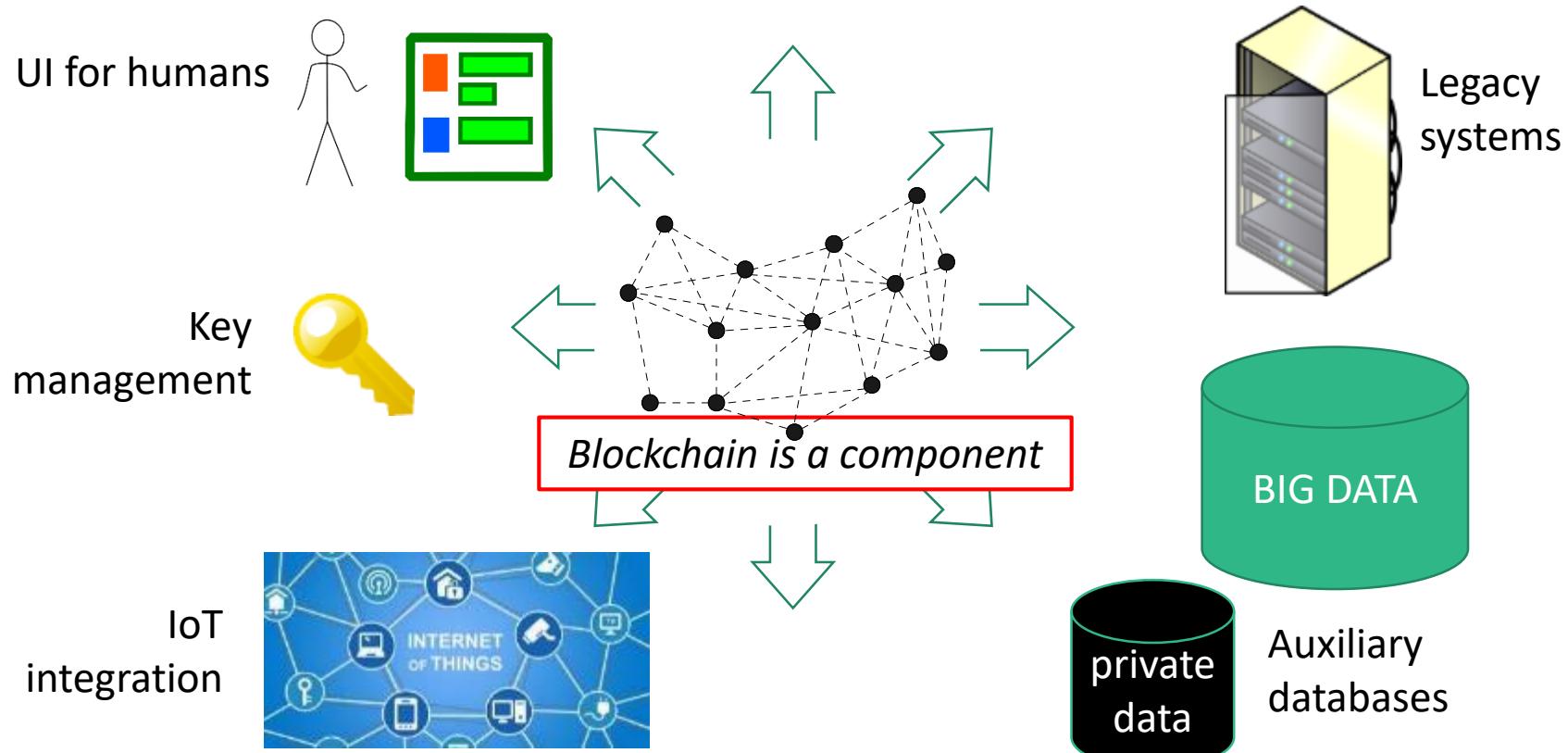
Example from “ATAM: Method for Architecture Evaluation” (Kazman et al., 2000)

[https://resources.sei.cmu.edu/asset\\_files/TechnicalReport/2000\\_005\\_001\\_13706.pdf](https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf)

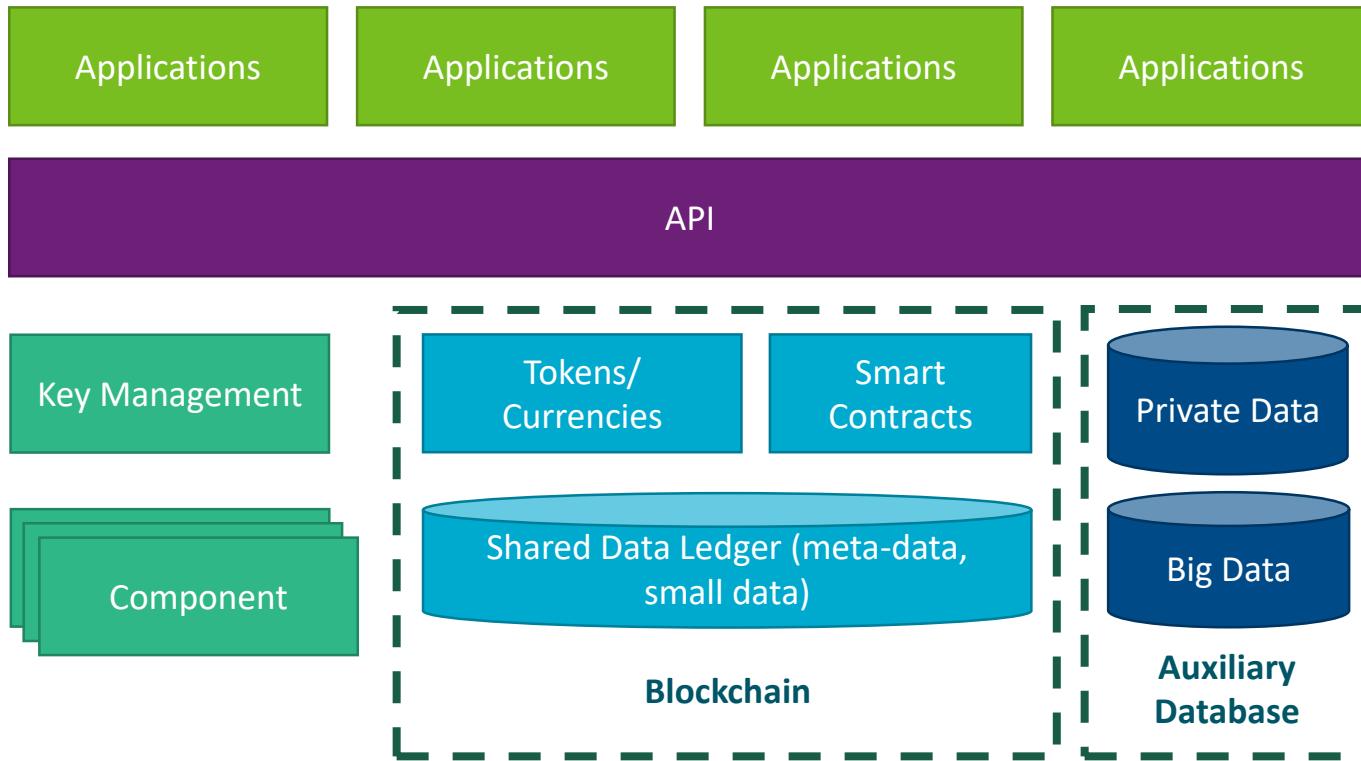
# Blockchain in Software Architecture



# Blockchains are Not Stand-Alone Systems



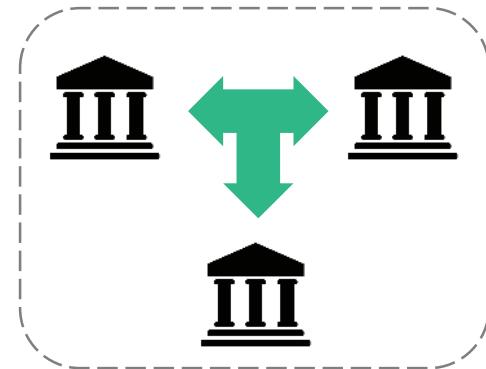
# Blockchain in Larger Software System



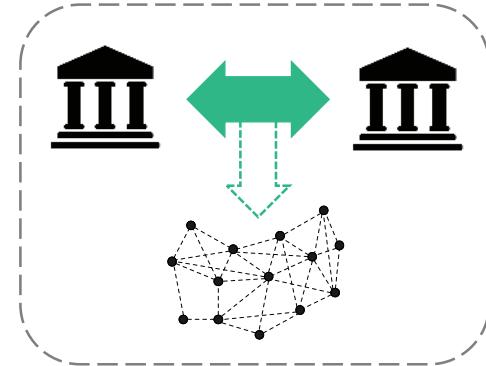
# What Does a Blockchain Do?

- Functionally, blockchains are...
- A database (ledger)
  - Record of transactions
- A compute platform
  - “Smart contracts”
- Distributed, and no central owner

Centralised Trust  
using a  
Third-Party



Distributed Trust  
using a  
Blockchain

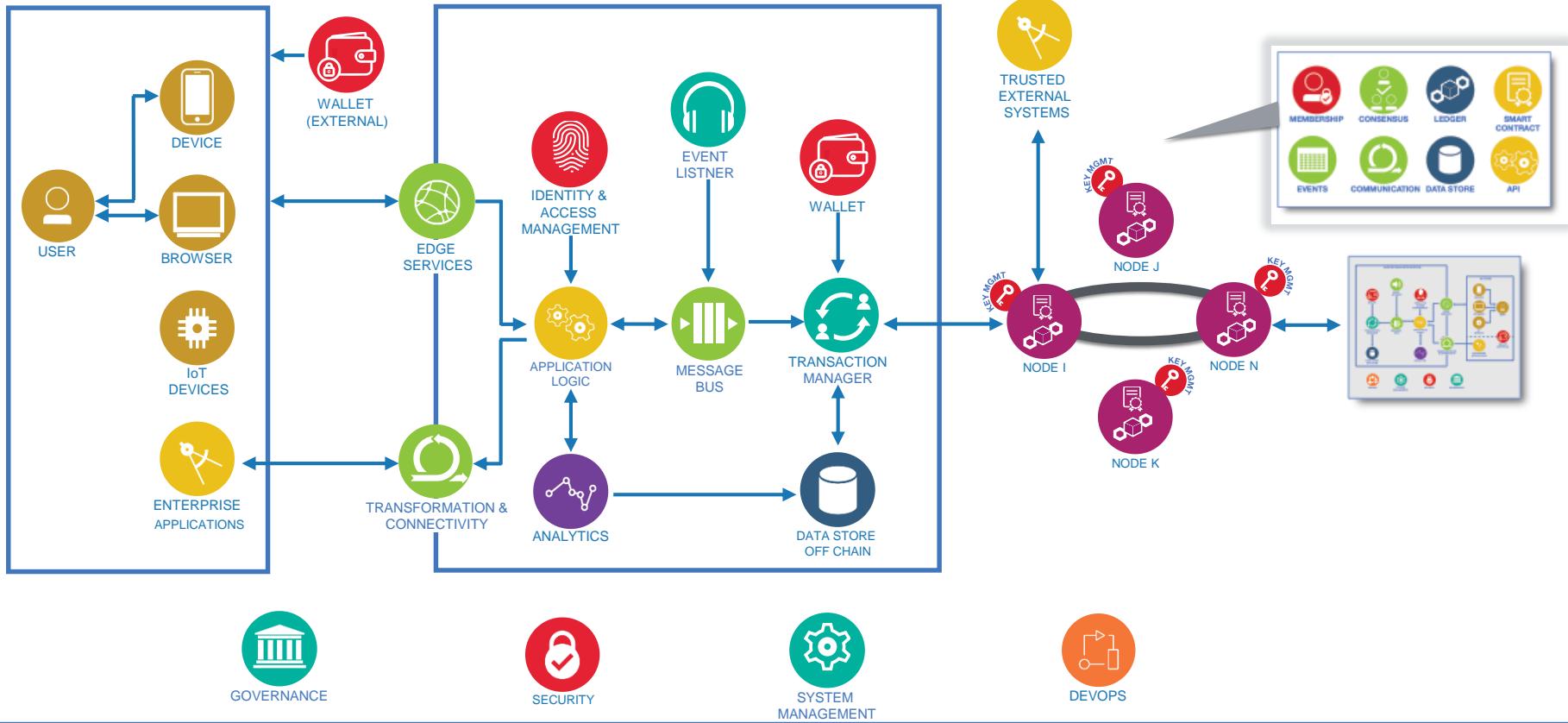


# Blockchain as a Software Component

- Complex, network-based software component
- Providing services
  - Data storage
  - Computation services
  - Communication services
  - Asset management
- Features
  - Cryptographically-secure payment
  - Mining
  - Transaction Validation
  - Incentive mechanism
  - Permission management



# An Example Blockchain Reference Architecture



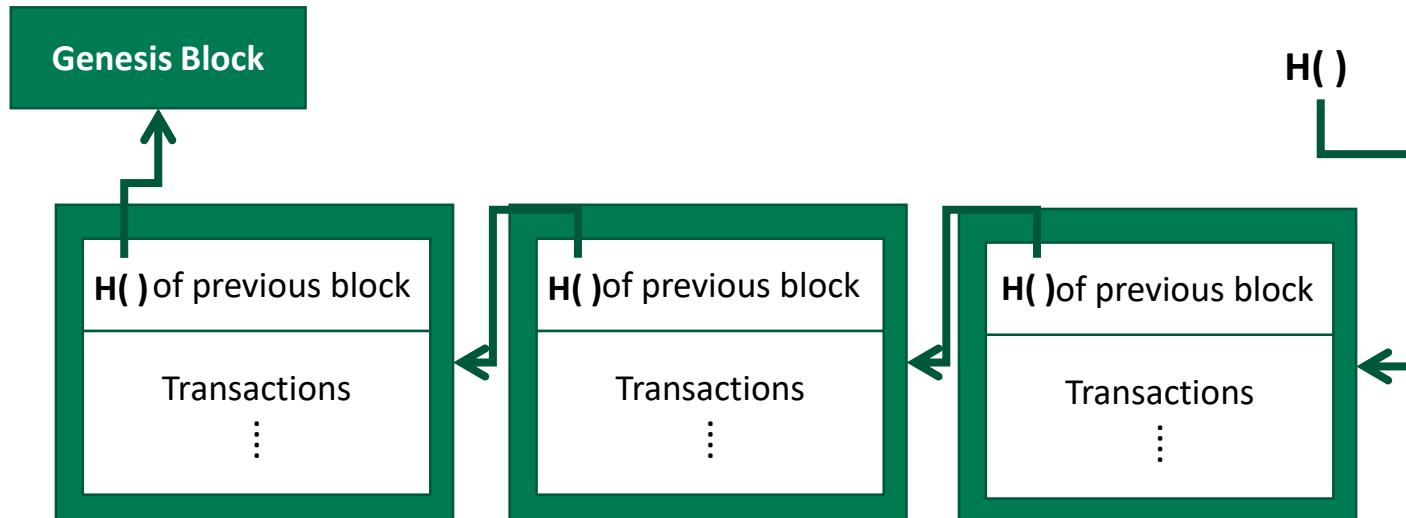
# Blockchain Component Service Options

- Data Storage
- Computation Service
- Communication Service
- Asset Management



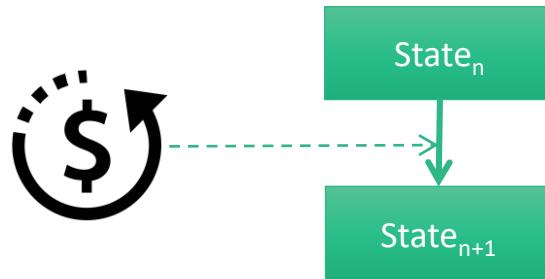
# Blockchain as Storage Element

- Blockchain data structure
  - Linked list with hash pointers
- Tamper-proof
  - Computational constraints
  - Incentive scheme



# Blockchain as Storage Element

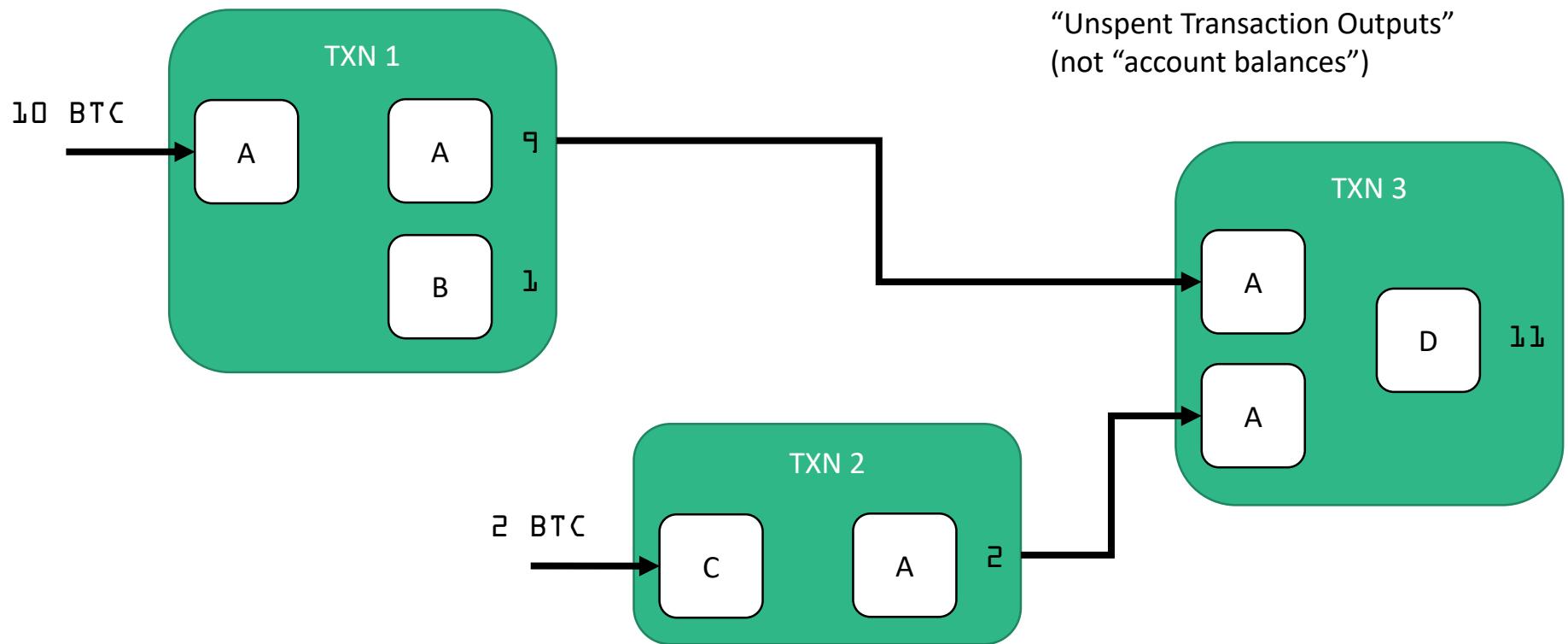
- Transaction Integrity
  - Transactions represent authorized state transitions
  - Transactions can record data
  - Transaction can transfer control of digital assets among participants
    - **Crypto-currencies**
    - **Smart contract-based digital assets**
  - Public key cryptography and digital signature are used to identify accounts
    - **Ensure integrity and authorization of transactions initiated on a blockchain**



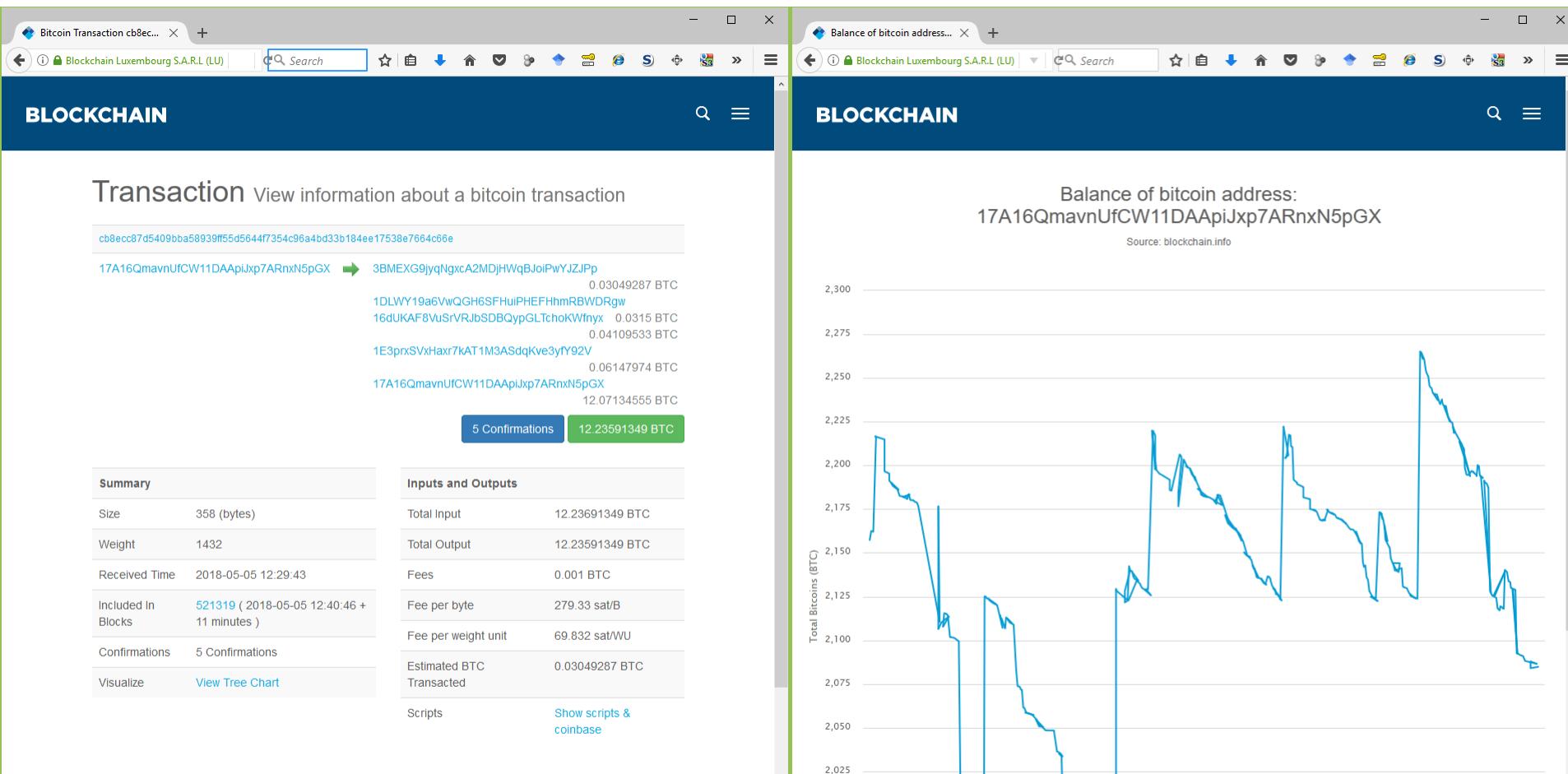
# Representation of Digital Assets

- Bitcoin Cryptocurrency
  - Collection of unspent transaction outputs (UTXO) from all previous transactions to that address
  - Tokens represented using conventions (e.g. tracking “color”)
- Ethereum
  - Cryptocurrency account balances in global system state
  - Tokens represented by smart contracts using storage

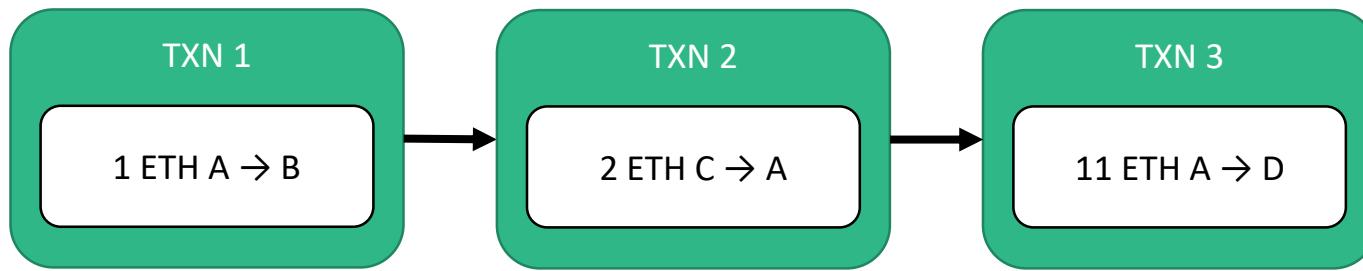
# Bitcoin Holdings – UTXO



# What Do Bitcoin Transactions/ Holdings Look Like?



# Ethereum Holdings – Accounts



Address	Balance
A	10
B	1
C	4

Address	Balance
A	9
B	2
C	4

Address	Balance
A	12
B	0
C	2

Address	Balance
A	1
B	0
C	4
D	11



# What Do Ethereum Transactions/ Holdings Look Like?

The image displays two side-by-side screenshots of the Etherscan blockchain explorer interface, illustrating the details of an Ethereum transaction and the holdings of a specific address.

**Left Screenshot (Transaction Details):**

- Header:** Ethereum Transaction 0xa14dee4df20ef4616cd3193d52b8c4abd38b1af38c63470545898ae47766d35a
- Breadcrumbs:** Home / Transactions / Transaction Information
- Sponsored Link:** Share Wi-Fi - earn cryptocurrency. Decentralized free Wi-Fi network. Get your bonus now. World Wi-Fi
- Overview Tab:** Selected. Shows the transaction information table below.
- Table (Transaction Information):**

Transaction Information		Tools & Utilities
TxHash:	0xa14dee4df20ef4616cd3193d52b8c4abd38b1af38c63470545898ae47766d35a	
TxReceipt Status:	Success	
Block Height:	5560602 (6 block confirmations)	
TimeStamp:	1 min ago (May-05-2018 01:04:01 PM +UTC)	
From:	0x6b0b7ecc2263aa562014b93b383e83111e6c84b	
To:	0x4c60fc9cdf2b334dc0c1ba821727c08d3da13db4	
Value:	5.076 Ether (\$4,175.21)	
Gas Limit:	31000	
Gas Used By Txn:	21000	
Gas Price:	0.000000005 Ether (5 Gwei)	
Actual Tx Cost/Fee:	0.000105 Ether (\$0.09)	
Nonce:	7	

**Right Screenshot (Address Details):**

- Header:** Ethereum Accounts, Addr... (0x4c60fc9cdf2b334dc0c1ba821727c08d3da13db4)
- Breadcrumbs:** Home / Accounts / Address
- Sponsored Link:** DocTailor - Legal Self Customisable Smart Contract Platform - Bridging the Gap Between Business & Cryptocurrency Holders - Join Now!
- Overview Tab:** Selected. Shows the address balance and transaction history.
- Table (Address Overview):**

Balance:	5.207054917052558449 Ether
USD Value:	\$4,281.14 (@ \$822.18/ETH)
Transactions:	121 txns
- Misc Tab:** Shows Address Watch and Token Balances.
- Table (Transactions):**

TxHash	Age	From	To	Value	[TxFee]
0xa14dee4df20ef461...	2 mins ago	0x6b0b7ecc2263aa...	IN	0x4c60fc9cdf2b334d...	5.076 Ether 0.000105
0xd155ec4c8c7bfbc...	4 hrs ago	0x4c60fc9cdf2b334d...	OUT	0x6b0b7ecc2263aa...	0.1 Ether 0.0001029

# Storing Arbitrary Data on Blockchain

- Two ways of storing data on blockchain
  - Adding data into transactions
    - Bitcoin
    - Ethereum
  - Adding data into contract storage
    - Smart contract on Ethereum
    - Smart contracts have an address, which is used to invoke the contract
    - Smart contract can only update its own storage
    - “Update” is only to the latest view of state (remember, append-only txns)
- Both ways store data through submitting transactions
  - Contain information of money transfer
  - Together with optional other data



# Comparison with Other Data Storage

- Comparison with Shared Centralised Database
- Comparison with Cloud Storage
- Comparison with Peer-to-Peer Data Storage
- Comparison with Replicated State Machines

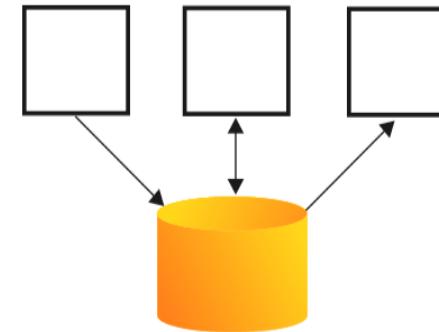
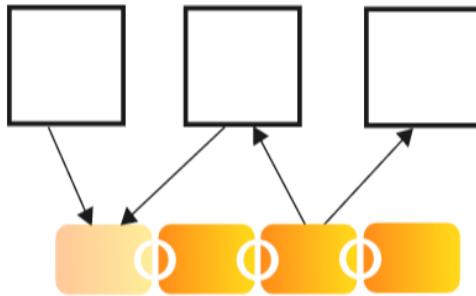


# NFPs: Blockchain vs Conventional Technology

- Non-Functional Property Trade-offs
  - (+) Integrity, Non-repudiation
  - (-) Confidentiality, Privacy
  - (-) Modifiability
  - (-) Throughput/ Scalability/ Big Data
  - (+ read/ - write) Availability/ Latency
- Many “limitations” are only for older public blockchains
  - e.g. hugely inefficient (more electricity than Ireland?)
  - e.g. very low performance (1 hour latency? max 3 tps?)



# Blockchain vs. Shared Centralised DB



- Transactions are Append-Only
  - Mimic “CRUD” only as a view of the blockchain (transaction log)
- Consensus by majority of peers agree on transactions
  - Usually no master, no trusted nodes

- Create, Read, Update, Delete
  - Log present, but only for admins
- Distributed Transactions (2 Phase Commit, Paxos)
  - Usually a master node, usually only a few trusted nodes

# Blockchain vs. Cloud

Blockchain	Cloud
No trusted single party	Cloud provider is trusted
Users can monitor or participate themselves as nodes that store the blockchain	Cloud provider store user data and provide access to the data
Data integrity is guaranteed (probabilistically)	Data integrity and access availability may not be guaranteed
No defined SLAs (service-level agreement) provided by public blockchain	Clearly defined SLAs

# Blockchain vs. Peer-to-Peer Data Storage

- Peer-to-peer Data Storage allow users to access data that is stored in other computers connected to the same peer-to-peer network
  - BitTorrent, IPFS (InterPlanetary File System)

Blockchain	Peer-to-Peer Data Storage
Very high availability <ul style="list-style-type: none"><li>• All nodes have the same shared copy of the blockchain data</li></ul>	Low availability for unpopular data
Not suitable for storing large data	Only store or distribute content the user wants to store or distribute
Strong data integrity	Hash pointer, checksum

# Blockchain vs. Replicated State Machines

	Blockchain	Replicated State Machines
Fault Tolerance	Use distribution to not depend on any single entity	<ul style="list-style-type: none"><li>• Replicate state at multiple servers</li><li>• Coordinate service requests from clients</li></ul>
Consensus	Ensure only one among multiple conflicting proposed transaction is included	<p>Decide upon receiving update requests from components</p> <ul style="list-style-type: none"><li>• Ensure only one client acquires a lock</li></ul>
Voting	large portion of the community to agree to achieve consensus	A quorum of voters with weighted votes
Communication	Transactions are replicated and persisted after it is included	<ul style="list-style-type: none"><li>• Transmitting state update data among components</li><li>• Information is replicated</li></ul>
Facilitation	Blocks and transactions are totally ordered	Requests are ordered

# Blockchain Component Service Options

- Data Storage
- **Computation Service**
- Communication Service
- Asset Management



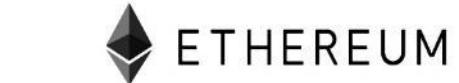
# Data and Computational Platform

- **Blockchain 1.0 – Cryptocurrency**



- **Blockchain 2.0 – Cryptoeconomic State Machine**

- Smart contract: Event-driven program (with state) that runs on a blockchain
  - Can realize more complex business logic
    - *More than simple currency/token-based value transfer*
    - Can represent digital assets on the ledger
  - Computational results stored on the public ledger

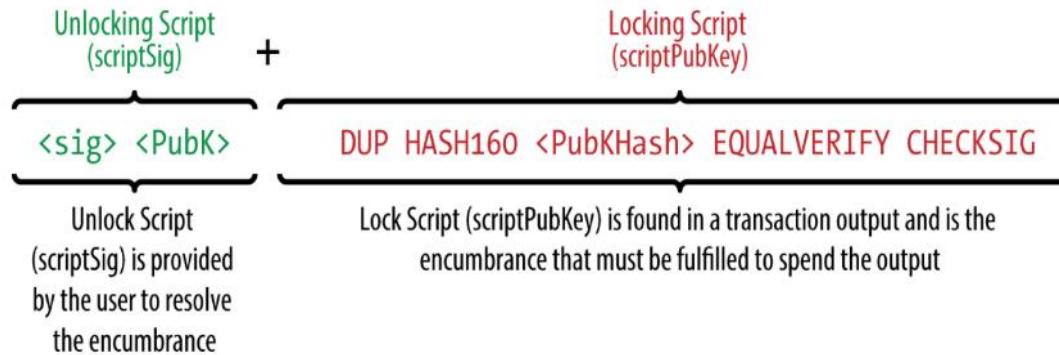


corda



# Limited Computational Power of 1<sup>st</sup> Generation

- Native smart contracts on Bitcoin do not support complex control flow definition



- External services allow end users to build self-executing contracts on Bitcoin
  - Executed by external Oracle
  - Integrity of execution is not guaranteed

# Turing-Complete 2<sup>nd</sup> Generation

- Ethereum is a general computational platform
  - Turing-complete programming language
  - In principle, as expressive as every other general purpose programming language (via Church-Turing thesis)
  - In practice, limitations on computational complexity
    - Gas limit
- Hyperledger Fabric, R3 Corda allow Java, etc
  - Can limit computation to nodes for parties of interest
  - Be careful to make your smart contracts deterministic

# 3<sup>rd</sup> Generation? Sub-Turing-Complete Again

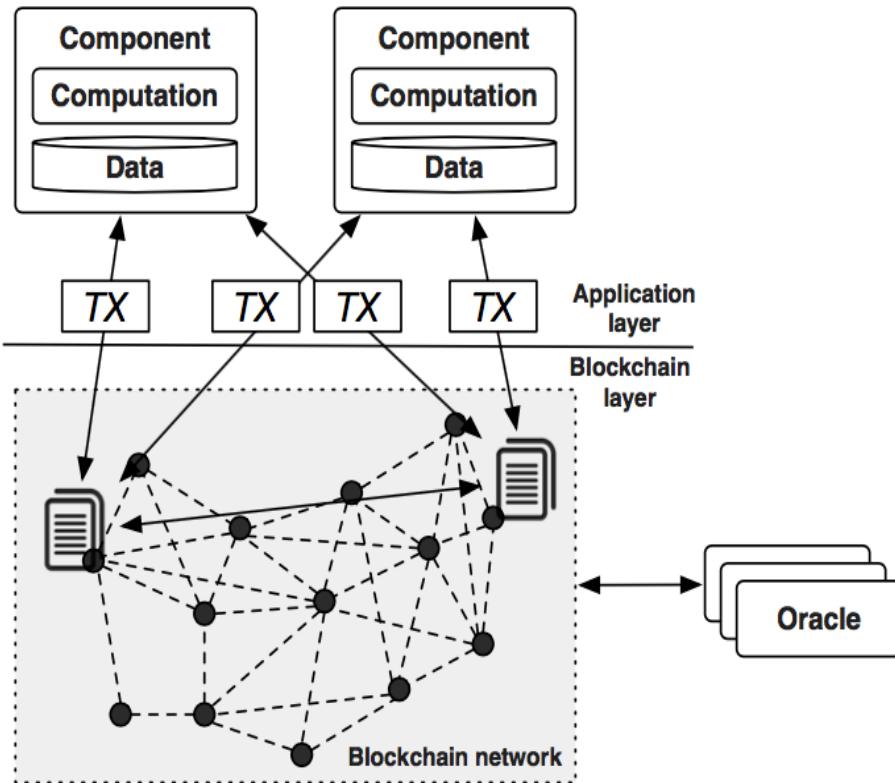
- A middle way – a somewhat expressive language, but still allowing automated formal verification
- e.g. Kadena's Pact language
  - Lists, datatypes
  - Functional: Conditionals, Fold, Map, Filter, ...
    - But not arbitrary recursion
  - Automatic verification using model checkers (Z3)
- e.g. DAML (Digital Asset)
  - Functional, non-Turing complete
  - Focus on workflow for exchange of rights & obligations
  - Tool support for formal verification

# Blockchain Component Service Options

- Data Storage
- Computation Service
- **Communication Service**
- Asset Management



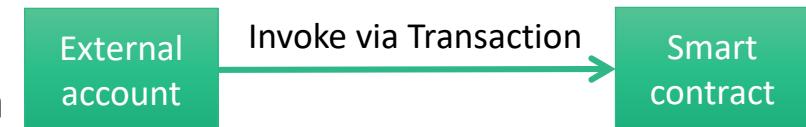
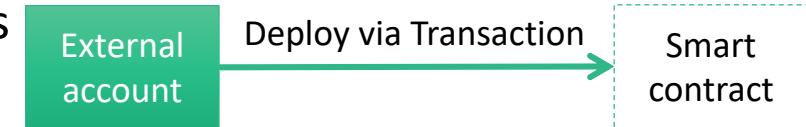
# Data Communication



- Components at application layer use blockchain as a mediator to transfer data
  - Sending data to blockchain using transactions
  - Query blockchain to retrieve data
- Blockchain provides data API
  - Access historical transactions
  - Filter historical transactions
- Local component monitoring update from new blocks
  - Relevant data in a local database

# Computation Communication 1/2

- Components use blockchain to coordinate computation
  - Submitting transactions to smart contracts to invoke their function
  - Using oracle to sign transactions depending on external state
- Typical control flow
  - Initiated from externally owned accounts
  - Transferred among contract accounts
- Contract termination
  - Cannot respond to transactions
  - Contract code remains on the blockchain
    - Permanently stored
    - In the creation transaction



# Computation Communication 2/2

- Oracle facilitates component coordination with external state
  - Some direct support for oracles
  - Others use oracles as external services
    - Interacting with blockchain through normal transactions
- Validation of transactions depends on external state
  - Platform-supported oracle can validate and sign transaction
    - Block transaction progress until the oracle completes
  - Oracle can be an external service injecting data into blockchain
    - Other smart contracts use that data to validate transaction
    - Delay is between the external state changes and time when those are recorded into blockchain
- Automated vs. human



# Blockchain Component Service Options

- Data Storage
- Computation Service
- Communication Service
- **Asset Management**



# Asset Management and Control Mechanism

- Native token of the 1<sup>st</sup> generation of blockchain
  - Cryptocurrency is the native asset
  - Identity of the cryptocurrency or associated data can represent other assets
    - Track claims of title over physical assets
    - Transactions record the transfer of title from one user to another
  - Limited due to small size of arbitrary data
    - **Bitcoin overlay network: colored coin**
      - Taint a subset of Bitcoin to represent and manage real-world assets
      - Few attributes can be recorded and few conditions can be checked within blockchain
- Smart contract of the 2<sup>nd</sup> generation of blockchain
  - Enable more expressive data structure
  - Flexibility for tokenizing a wider variety of assets



# Tokens as Digital Assets

- What is a Token?
  - Like plastic discs/boarding pass/ticket/ in the physical world
  - Digital asset on the blockchain – current “owner” etc can be checked by looking at ledger
- A token represents a bundle of rights and obligations
  - Can represent digital assets or physical assets
    - Fungible: interchangeable, like cryptocurrencies, gasoline
    - Non-fungible: Unique and cannot be interchanged, like cryptokitties, artwork and land
  - Transferrable (or not); Reusable (or not); Exclusive (or not); ...
- Tokens for “ownership” of assets?
  - “Property” is a collection of rights
  - Can the blockchain legally bind ownership or change of ownership?
    - You could self-impose contractual conditions on token and underlying property, to try to ensure the blockchain record is consistent with legal ownership
    - But, contracts might not survive bankruptcy; and Courts can order change of ownership without the blockchain

# Digital Assets and Blockchains: Symbiosis

- The blockchain ensures digital assets are not replicated
  - For digital information, you can make identical copies
  - For money and other assets, don't make identical copies!
  - The global public ledger means everyone can check



- Digital assets (esp. cryptocurrency for public blockchains) ensure the blockchain is operated
  - Provide incentives for nodes to operate the blockchain
    - Mining reward claimed under a convention, by the node that creates a block
    - Transaction fees – offered by the transacting party to the node which includes the transaction in a block

# Token Standards on Ethereum

- ERC20 for fungible tokens
- ERC841 for non-fungible tokens

```
1 // -----
2 // ERC Token Standard #20 Interface
3 // https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
4 //
5 contract ERC20Interface {
6     function totalSupply() public view returns (uint);
7     function balanceOf(address tokenOwner) public view returns (uint balance);
8     function allowance(address tokenOwner, address spender) public view returns (uint remaining);
9     function transfer(address to, uint tokens) public returns (bool success);
10    function approve(address spender, uint tokens) public returns (bool success);
11    function transferFrom(address from, address to, uint tokens) public returns (bool success);
12
13    event Transfer(address indexed from, address indexed to, uint tokens);
14    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
15 }
```

# Design and Trade-offs in Blockchain Applications

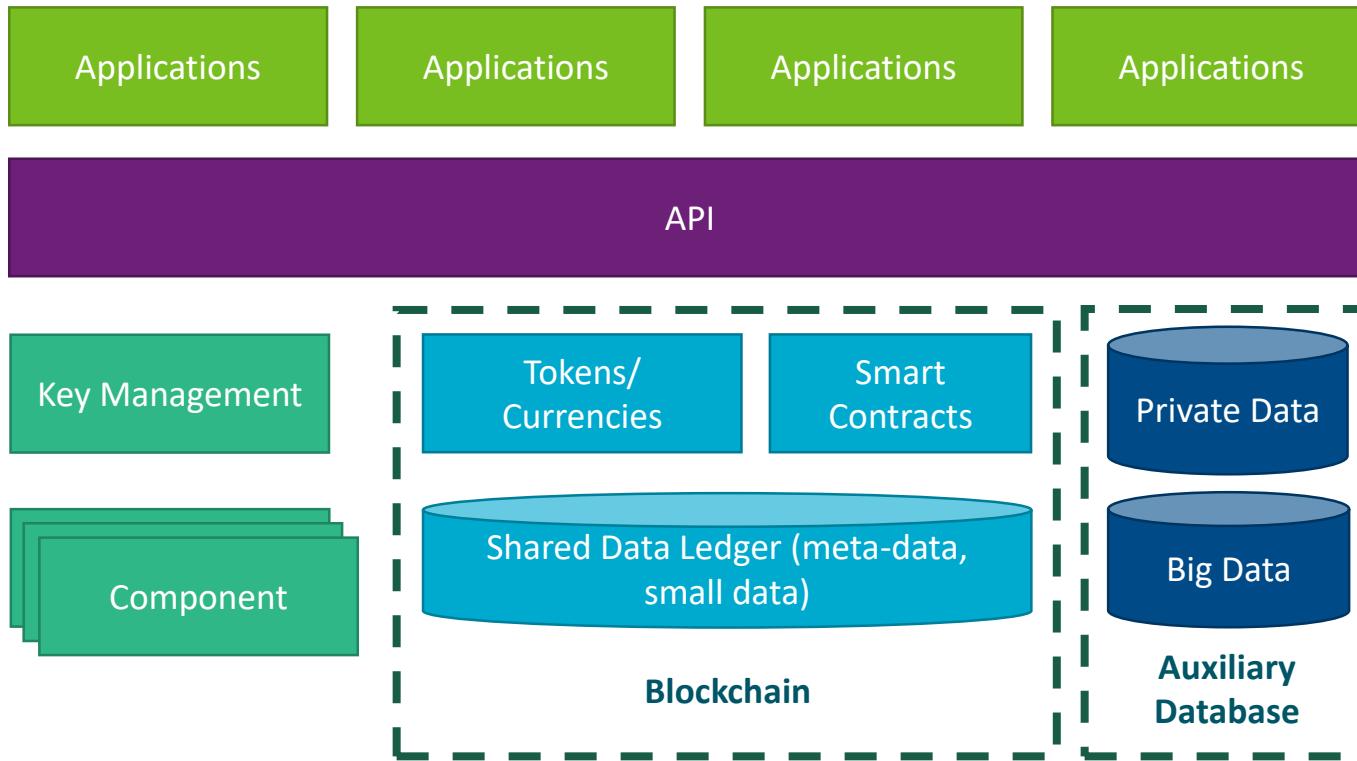


# Use Blockchain for What It's Good For

- Trustworthy and efficient ways to work together
  - Focus on spaces between individuals, organisations
    - Data integrity for information sharing
    - Neutral ground for process coordination
  - Logically centralises information
  - Administratively decentralises control
- Digital Assets and Tokens
  - Authorised by issuer, transferrable between cryptographic IDs
  - Representing physical assets, digital assets, services, rights, ...
  - Be careful about legal title for property: blockchain is not magic



# Recall: Blockchain in Larger Software System



# How to use blockchains in a design?

- Later lectures covering topics in design process, blockchain design patterns, etc
- Some themes
  - Choice and configuration of blockchain
    - Private (Consortium) vs. Public
    - Single logical chain vs. network of distributed ledgers
  - Use programming design patterns for smart contracts
  - Combine on-chain with off-chain components
    - Hashed content on-chain (content off-chain)
    - Signed or encrypted content on-chain (keys off-chain)
    - “State channels” judges on-chain (computation & comms off-chain)
- Today, some illustrative examples

# 3 Examples from Data61/Treasury Reports

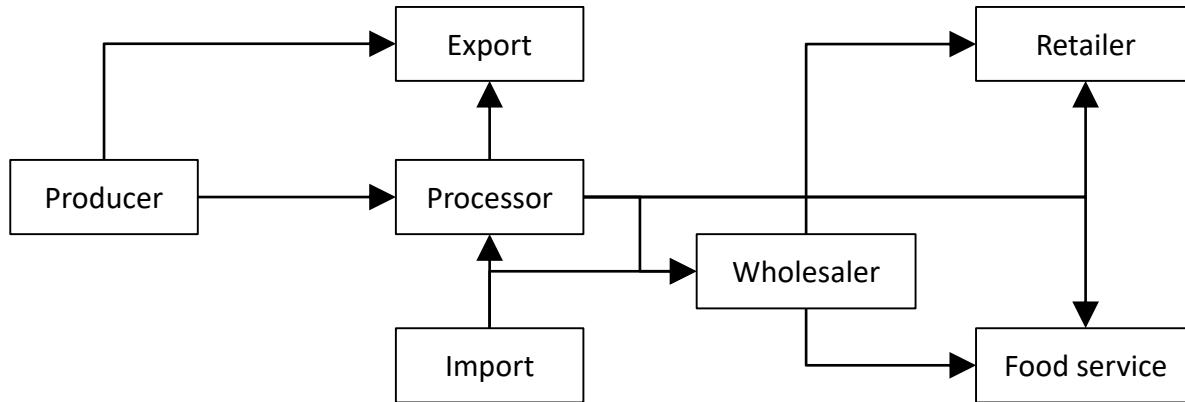


- <http://www.data61.csiro.au/blockchain>
- *Risks and Opportunities for Systems Using Blockchain and Smart Contracts*
- What are technical risks & opportunities for use cases?

# Potential Use Cases

- Financial Services
  - Digital currency
  - (International) payments
  - Reconciliation
  - Settlement
  - Markets
  - Trade finance
- Government Services
  - Registry & Identity
  - Grants & Social Security
  - Quota management
  - Taxation
- Enterprise and Industry
  - Supply chain
  - IoT
  - Metered access
  - Digital rights & IP
  - Data management
  - Attestation
  - Inter-divisional accounting
  - Corporate Affairs
- Three Illustrative Cases Selected
  1. Agricultural supply chain
  2. Open data registry
  3. Remittance payments

# Agricultural Supply Chain – Use Case

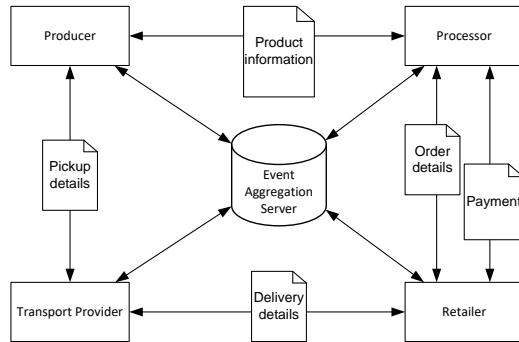


Interoperability  
Latency  
Integrity  
Confidentiality  
Scalability

# Agricultural Supply Chain – Designs

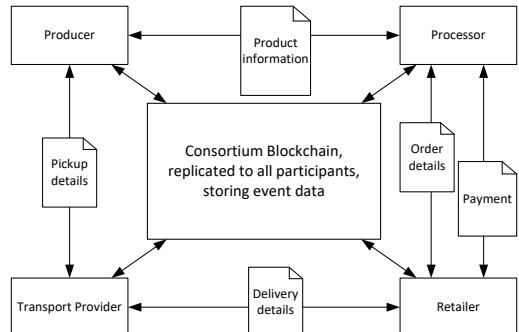
## 1. Conventional

Point-to-point messaging and event aggregation server

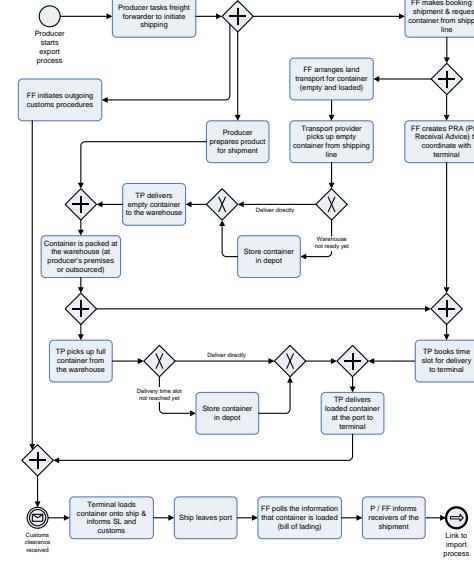


## 2. Event Tracking on Blockchain

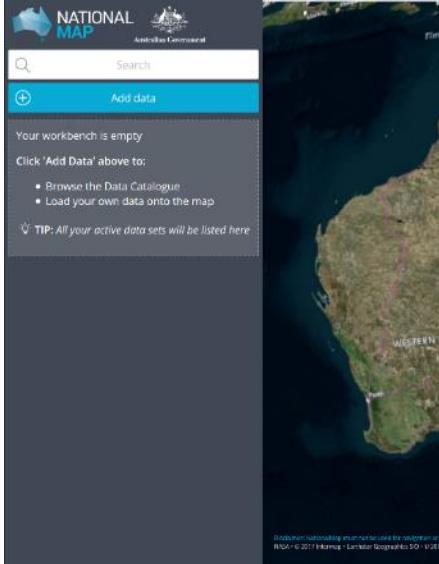
Point-to-point messaging and event aggregation on blockchain



## 3. Supply chain process coordination on blockchain as smart contracts



# Open Data Registries – Use Case



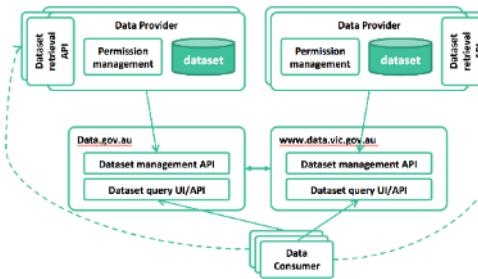
A screenshot of the data.gov.au website. The header features the Australian Government logo and the data.gov.au logo. The main navigation menu includes Datasets, Organisations, About, Site Statistics, Request Data, Use Cases, National Map, and Toolkit. Below the menu is a search bar with the placeholder 'E.g. environment'. A sidebar on the left shows a message about an empty workbench and instructions to add data. The central content area contains a section titled 'Search data' with a search bar, popular tags like Earth Sciences, GA Publication, and Oceans, and a statistics box showing 35.8k datasets, 7k API enabled resources, 24.7k openly licenced datasets, and 25 unpublished datasets. There are also sections for 'Latest data.gov.au News' and 'Guest Post: Help to shape ASIC Registry's public datasets'.

Integrity  
Availability  
Read Latency  
Interoperability  
Barriers to access

# Open Data Registries – Designs

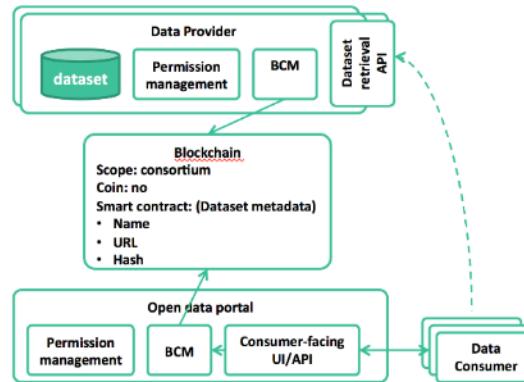
## 1. Conventional

Registry operated by single agency



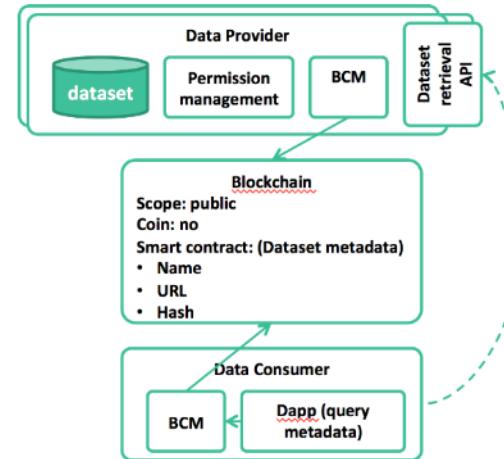
## 2. Consortium across data providers

Public access still controlled through a portal

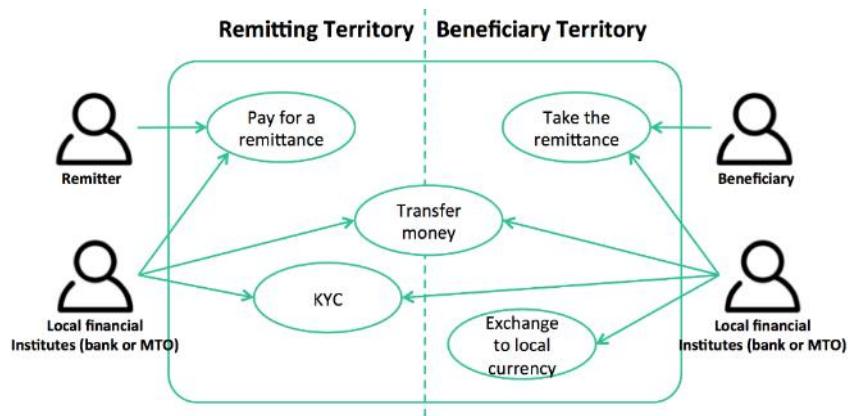


## 3. Registry on public blockchain

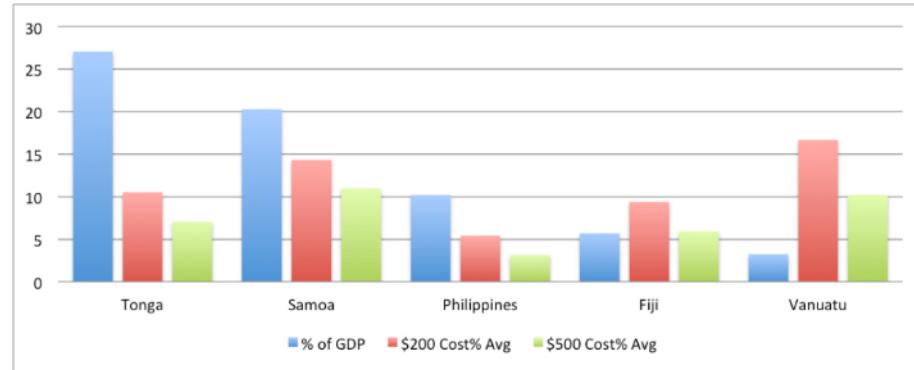
Agency only controls entries included on official index



# Remittance Payments – Use Case



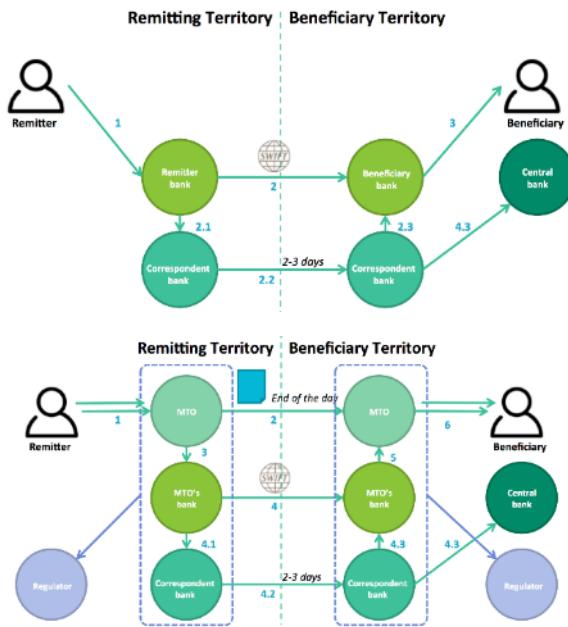
Write Latency  
Cost  
Cost transparency  
Controlled confidentiality  
Low barriers to entry



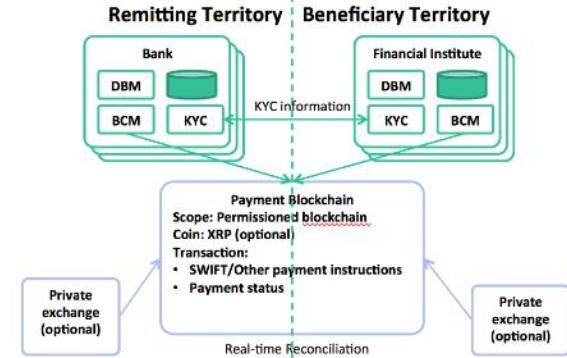
# Remittance Payments – Designs

## 1. Conventional

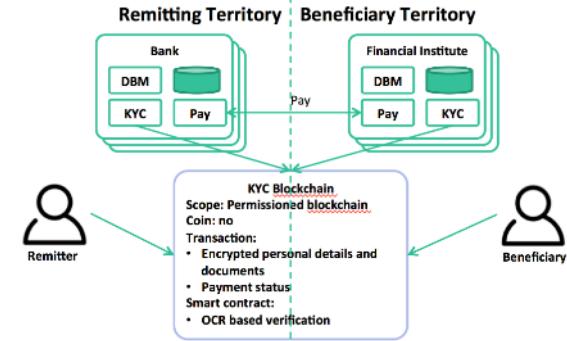
Through bank or MTO



## 2. Payment through blockchain



## 3. KYC through blockchain



# Summary



# Summary

- What is Software Architecture?
  - Architectural design choices impact Non-Functional Properties (NFPs)
  - Use Architectural Models (Views and Viewpoints)
    - To communicate and analyse systems
  - Analyse NFPs and consider NFP Trade-offs
- Blockchain in Software Architecture
  - Blockchain as Component, Connector, Configuration
  - Blockchain Component Services
    - Data Storage
    - Computation Service
    - Communication Service
    - Asset Management
- Design and Trade-offs in Blockchain-based Applications





# THANK YOU

[www.data61.csiro.au](http://www.data61.csiro.au)

- Design Process for Applications on Blockchain
- Cost



# COMP6452 Lecture 5.1: Design Process for Applications on Blockchain

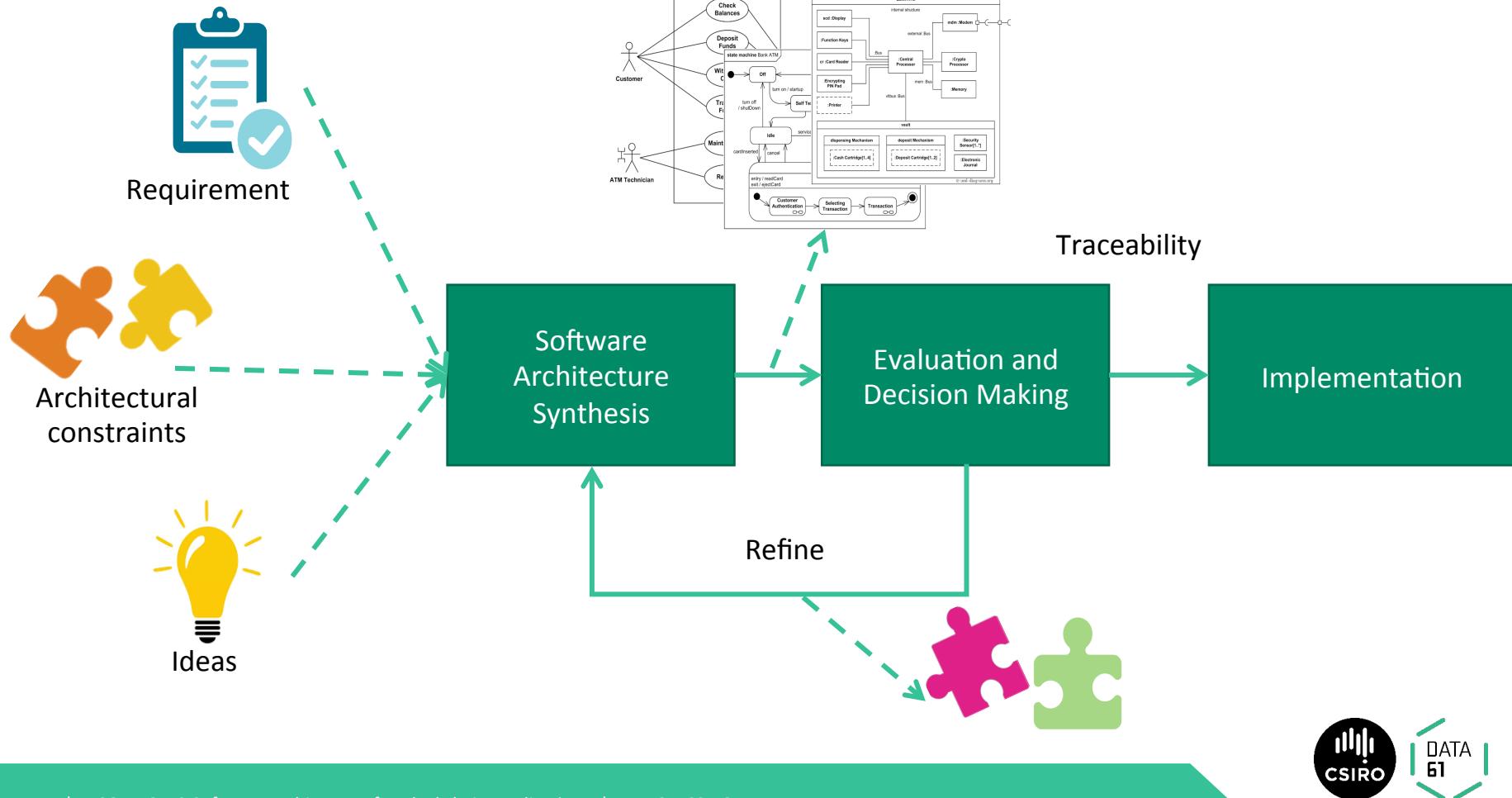
Xiwei (Sherry) Xu (xiwei.xu@data61.csiro.au)

18<sup>th</sup> of March, 2019

# Outline

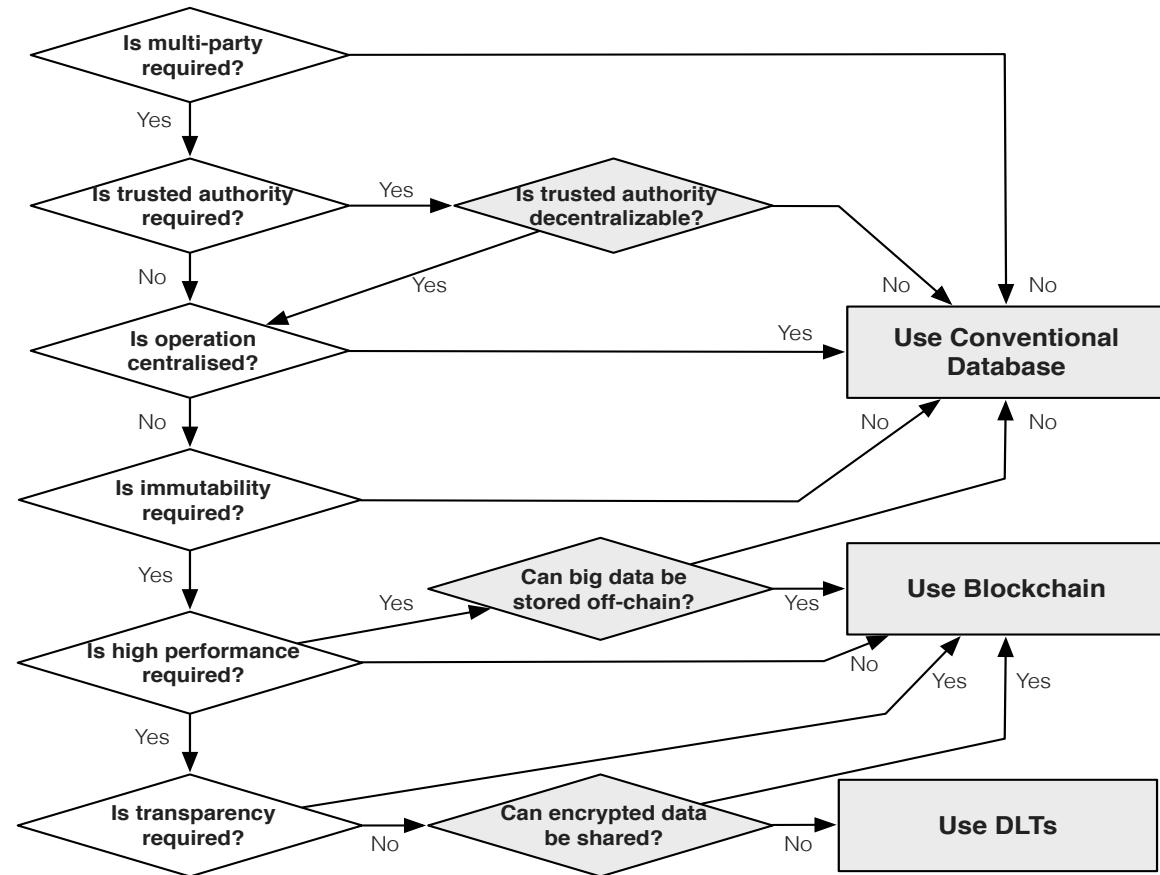
- Evaluation of Suitability
- Example Use Cases for Suitability Evaluation
- Design Process for Blockchain-based Systems
  - Subsidiary Design Choices
    - Private blockchain vs. public blockchain
    - What consensus protocol
    - What the block frequency
  - What's on-chain and What's off-chain

# Design Process

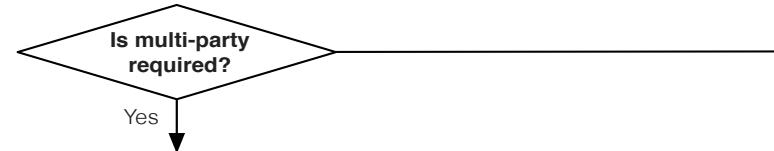


# Evaluation of Suitability

# Evaluation Framework



# Is Multi-party Required? 1/2

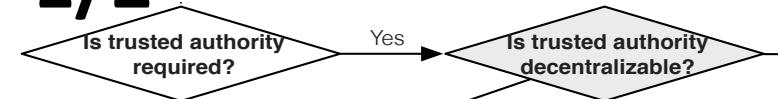


- Blockchain is **NOT SUITABLE** for systems that only serve individual isolated users
- Conventional database is simpler and more efficient
- Different legally distinct parties
- Supply chain
  - Complex, dynamic, multi-party arrangements with regulatory and logistical constraints spanning jurisdictional boundaries
    - Manufacturers, shipping companies, transport infrastructure organizations, financial service firms, or regulators
    - Information exchange can be as important and difficult as the physical exchange of goods
- Inter-bank payments and reconciliation
  - Two different banks
  - Account holders are organizations or individuals

# Is Multi-party Required? 2/2

- Large enterprise or government
  - Different functional or geographic divisions or departments
  - Informational or administrative “silos” as multiple parties
- Blockchain can be **SUITABLE** for supporting multi-party systems
  - Physically distributed
  - Logically centralized
  - Infrastructure providing a single view of truth across those parties

# Is Trusted Authority Required? 1/2

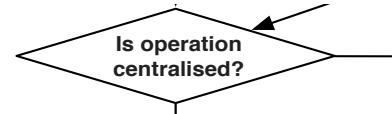


- An entity that is relied upon to perform a function (operating a system)
  - Blockchain is **NOT SUITABLE** if a single party can or must be relied upon as a trusted authority
  - Trusted authority implements a traditional centralized solution with conventional technologies
    - Banks
    - Government departments
- Scope of the system is important in deciding the question
  - Bank is a trusted authority for payment transfer among bank accounts
  - Central bank is a trusted authority for inter-bank payments
    - The two banks collective rely upon central bank
- Trusted authority is a single point of failure
  - Technical single points of failure can be mitigated by using redundancy
  - Single points of organizational or business failure remain present
    - Business failures, service interruptions, data loss or fraud

# Is Trusted Authority Required? 2/2

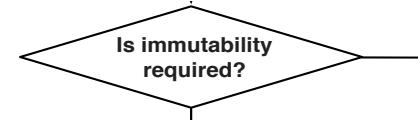
- Trusted authority is a monopoly or oligopoly service provider
  - “rent-seeking” behavior
    - Increase one’s share of existing wealth without creating new wealth
- A natural trusted authority might be difficult for everyone to accept reliance on that party
  - Organization could define a central agency to provide services for coordinated operations across the whole organization
  - Centralization of services can be perceived as a loss of control or power
- Blockchain is **SUITABLE** for system with no single party that is acceptable for operating the system
  - Operated jointly by a collective of nodes
  - Not remove trust
    - Users are exposed to risk in use of blockchain technology
    - What is trusted is the software, the incentive mechanism, and “oracles”
  - Distributed Trust
    - Remove the need to trust a single third-party to maintain a ledger

# Is operation centralized?



- System with multiple parties, but no party is suitable as a trusted authority for administering the system
  - Group of parties form a joint venture to operate a conventional centralized system
  - Credit card associations, like Visa and MasterCard
    - Joint venture between banks
- Centralized operation of the system lead to the administering party becoming a trusted authority
  - Unacceptable to the parties within the system
  - Forming a new entity like a joint venture is too costly
  - Centralized administration may cause single point of business failure
- Blockchain-based systems do not need a single system operator
  - Better system reliability and availability

# Is Immutability Required? 1/3



- *Data immutability* means data can not be changed or altered after its creation
  - Immutability supports non-repudiation
    - Assurance that a party cannot deny the authenticity of their signature
- Conventional technologies naturally support mutable data
- Blockchain naturally supports data immutability in the ledger
  - Linking of blocks in a chain of cryptographic hashes supports immutability
  - Data continually replicated across many locations and organizations
    - Attempts to change it in one location will be interpreted as an attack on integrity
  - Strong evidence that the transactions were performed by someone with control over their cryptographic keys
- Transaction history is immutable
  - Transaction changes the latest view of the current state

# Is Immutability Required? 2/3

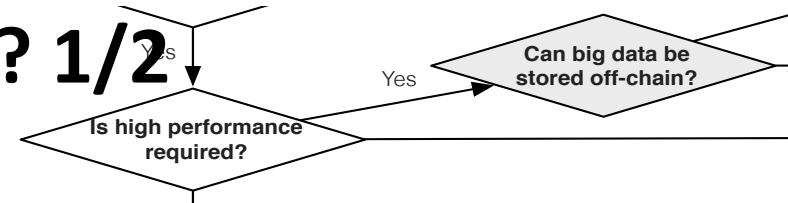
- Impossible to change the transaction history in most blockchains
  - Cause problems if blockchain contains illegal content
  - Court orders content to be removed from the blockchain
- Immutability makes it less adaptable for the following issues
  - Disputed transactions
  - Incorrect addresses
  - Exposure or loss of private keys
  - Data-entry errors
  - Unexpected changes to assets tokenized on blockchain
- Other cheaper mechanisms available to prove the originality of data
  - Hashing technology
  - Cryptographically signed data



# Is Immutability Required? 3/3

- Immutability of PoW-based blockchain is a long-run probabilistic durability
  - Conventional database supports ACID (Atomicity, Consistency, Isolation and Durability)
  - A transaction initially thought by a participant to be committed may later turn out to have been on a shorter chain
  - A transaction is in practice be immutable if it has been committed to a blockchain for a sufficiently long time (number of blocks)
    - X-confirmation ([Lecture 7](#))
- Blockchain using other consensus mechanism can offer stronger and more conventional immutability
  - E.g. PBFT(Practical Byzantine Fault Tolerance)
  - Small number of known nodes participating in the operation

# Is High Performance Required? 1/2



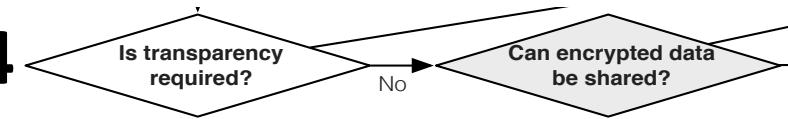
- Blockchain is ***NOT SUITABLE*** if the system need to support high performance
  - Extremely short response time (Latency)
  - Process very large amounts of data (Throughput)
- Bitcoin and Ethereum cannot currently match the maximum throughput of conventional transaction processing system
  - Visa payments network
- New mechanisms to improve performance (*Lecture 3*)
  - Sharding
  - State channels
  - Reduced inter-block time
- Consortium and private blockchains with careful design and performance tuning have much better performance

# Is High Performance Required? 2/2

- Read latency can be much faster than conventional technologies
  - Response time for accessing historical data from a blockchain client
  - Clients keep a full local copy of the database
  - No network delays
- Write latency is probabilistic with several sources of uncertainty
  - Network delay of transaction propagation
  - Consensus process delay
  - Confirmation blocks on PoW-based blockchain increases write latency
- Blockchain is **NOT SUITABLE** for storing Big Data
  - Large volumes or high velocity
  - Massive redundancy



# Is Transparency Required? 1/4



- *Data transparency* means data is available and accessible to by other parties
  - Facebook public newsfeed posts
  - Twitter public tweets
  - Facebook/Twitter support confidentiality
    - Choose what content publish to the public or to specific audience
- Blockchain provides a neutral platform where all participants can see and audit the published data
  - Validation of cryptocurrency transfers
    - From addresses with enough cryptocurrency
    - Signed with an authorized private key
  - Validation of smart contract execution is correctly recorded on the blockchain

# Is Transparency Required? 2/4

- Blockchain **MAY BE SUITABLE** if data transparency is required or acceptable
  - Confidentiality is harder to establish in blockchain-based systems
  - Information is visible to all participants
- Amount of interactions between parties is confidentiality concern
  - Very often customer relationships, pricing, or aggregated transaction volume are commercially-sensitive information
  - Use pseudonymity
    - Contents of a transaction are publicly visible
  - Create a new address for each transaction
    - Flow of assets may be used to infer relationships between addresses
  - Reidentification
    - Reuse of addresses and their connection via transfers of cryptocurrency



# Is Transparency Required? 3/4

- Public blockchain **MAY BE SUITABLE**
  - Public advertising or fully open government registries in highly regulated industries
    - Banks advertise on television
    - Television is not a highly-regulated banking transaction system
    - Data integrity and publicity is required rather than privacy or confidentiality
  - Secure software package management
  - IoT device configuration updates
- Blockchain can be used to share encrypted data
  - Asymmetrical with a party's public key
  - Symmetrical with a shared secret key
    - Requires a secure means of exchanging the secret key
  - Increase confidentiality, but reduce performance
  - Encrypted data makes it difficult to use smart contracts with the data
    - Embedding keys within a smart contract would reveal the keys

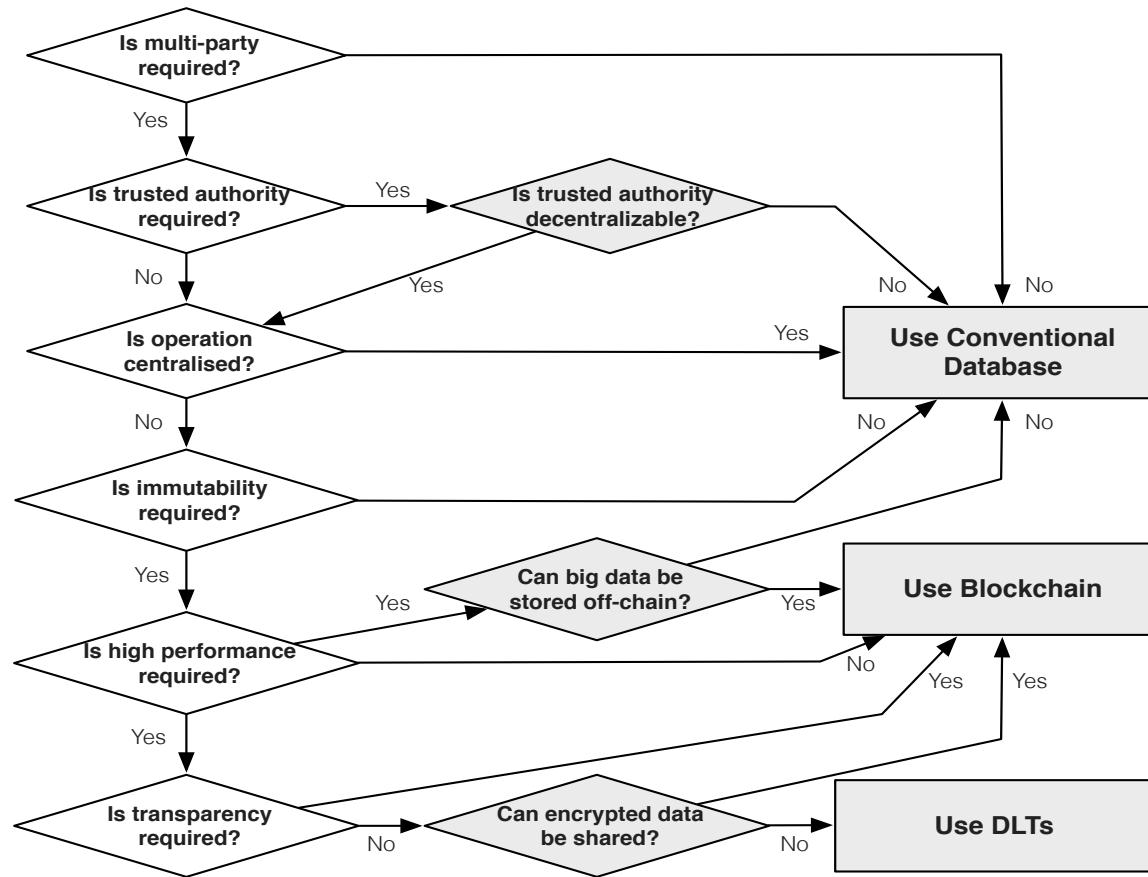


# Is Transparency Required? 4/4

- Sometimes encryption is not acceptable
  - Concerns about successful encryption key management
  - Future technological developments in decryption (e.g. quantum computing)
  - Reveal information as meta-data
- Consortium and private blockchains provide read access controls
  - Data is not commercial confidentiality between competitors
  - Trade-off is between the benefits of sharing data within the group of collaborators and retaining confidentiality towards competitors where needed
- More controlled data sharing can be enabled by distributed ledger technology
  - Corda or Hyperledger Fabric
    - Small ledgers shared between parties of interest to store each transaction



# Evaluation Framework



# Example Use Cases for Suitability Evaluation

	Supply chain	Electronic health records	Identity	Stock market
<b>Multi-party</b>	Required	Required	Required	Required
<b>Trusted authority</b>	Not required	Decentralized	Not required	Not required
<b>Centralized operation</b>	Not required	Not required	Not required	Not required
<b>Data immutability &amp; Non-repudiation</b>	Required	Required	Required	Required
<b>High performance</b>	Not required	Not required	Not required	Required
<b>Data transparency &amp; Confidentiality</b>	Transparent (but not fully public)	Confidential	Transparent	Confidential
<b>Sample Result</b>	DLT	Conventional System	Blockchain	Conventional System

# Use Case 1: Supply Chain 1/2

- A collection of processes involved in creating and distributing goods, from raw materials to completed products, through to consumers
- Highly complex **multi-party system**
  - Farmers, factories, transport providers, and retailers
- **Operations** are distributed and loosely coupled
- **Data transparency** is desired by participants
  - Support logistics planning, and to identify and respond to problems
  - Controlled confidentiality is required
    - Small ledgers between parties in interest
    - Conventional information exchange + hashed information on blockchain
- Transaction history and **data immutability** are desired to enable traceability back to the origin of goods
  - Control fraud and substitution



# Use Case 1: Supply Chain 2/2

Supply chain	
<b>Multi-party</b>	Required
<b>Trusted authority</b>	Not required
<b>Centralized operation</b>	Not required
<b>Data immutability &amp; Non-repudiation</b>	Required
<b>High performance</b>	Not required
<b>Data transparency &amp; Confidentiality</b>	Transparent (but not fully public)
<b>Sample Result</b>	DLT

- The time taken in a supply chain is dominated by physical transportation and storage, which moderates demand for **performance**
- Reasonably short latency is required at key points of hand-over of goods
- Dynamic structure of business relationship and **operation** can be accommodated by blockchain network
- Logically-centralized view of information supports demands for transparency

# Use Case 2: Electronic Health Records 1/4

- Collections of patient medical records
  - Blood type, vital signs, past medical records, medication, and radiology report
  - Maintained by specific healthcare providers in siloed systems
- **Multiple parties** from different medical jurisdictions are involved
  - Patients, professionals and organizations
- Healthcare service providers are **decentralized trusted authorities**
  - Each has access to patient data and authority to make changes
- **Operation is distributed** across healthcare service providers
- **Data transparency** is the main issue
  - Patient privacy
  - Shared with patient consent
  - Exceptions: emergency situations; access to anonymised data for approved medical research
- Health records cannot be inappropriately created or updated



# Use Case 2: Electronic Health Records 2/4

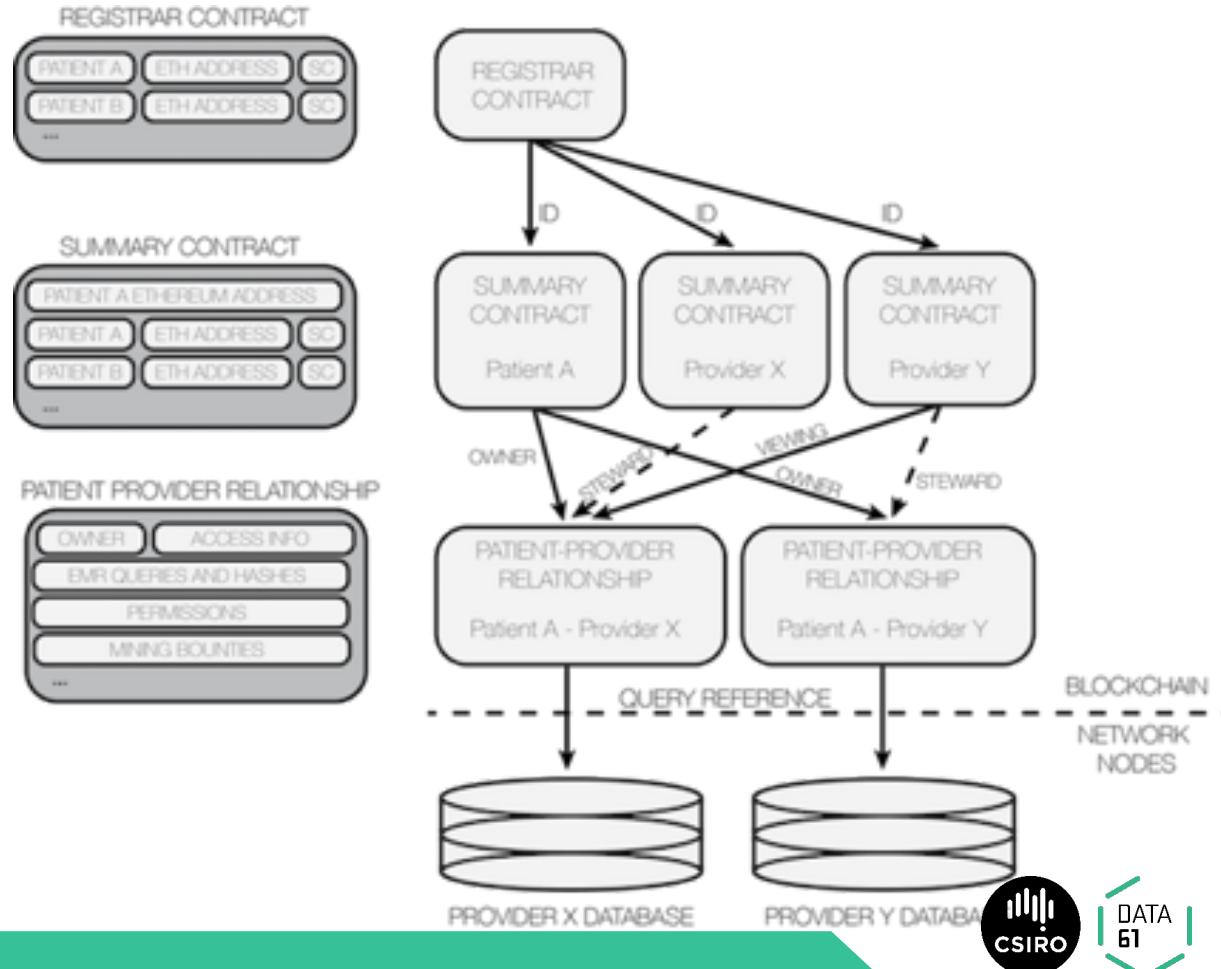
<b>Electronic health records</b>	
<b>Multi-party</b>	Required
<b>Trusted authority</b>	Decentralized
<b>Centralized operation</b>	Not required
<b>Data immutability &amp; Non-repudiation</b>	Required
<b>High performance</b>	Not required
<b>Data transparency &amp; Confidentiality</b>	Confidential
<b>Sample Result</b>	Conventional System

- **No low latency updates**
  - Most records do not change often
  - Large diagnostic image needs to be managed
- Due to privacy constraints, blockchain can not be used to store patient records, even in encrypted form
- Conventional systems are used
  - Blockchain provides auxiliary service
    - Keep audit logs of accesses made to EHR
    - Ensure data integrity

# Use Case 2: Electronic Health Records 3/4



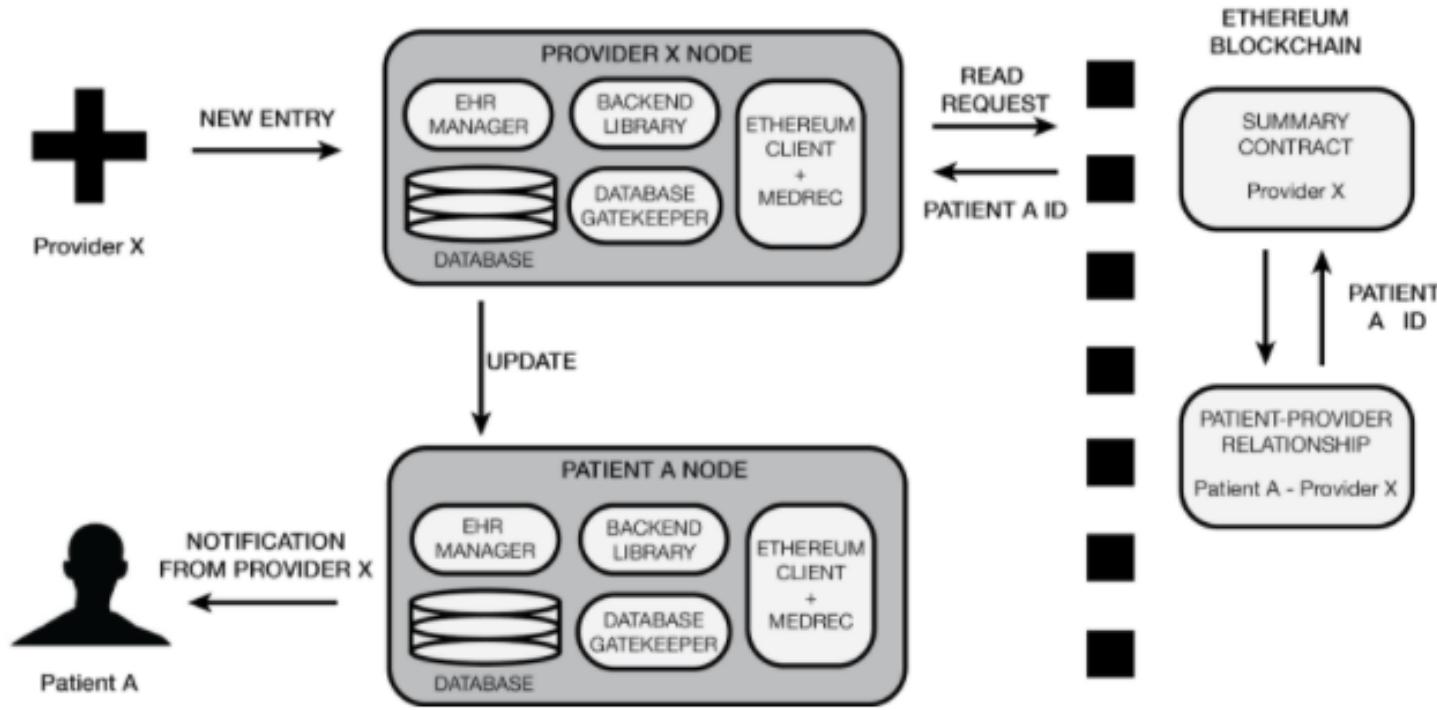
- Initiative to explore on blockchain architecture in contributing to secure and interoperable EHRs system



# Use Case 2: Electronic Health Records 4/4

## MedRec 2.0

- Improve scalability
  - Bypass the blockchain for patient notification
  - Restrict blockchain storage to creation and modification of identities and relationships



# Use Case 3: Identity Management 1/2

- Individuals, organizations, devices and assets can be identified by many schemes
  - Passport, wedding certificates, serial number, registration certificate
- Conventionally, the **operations are centralized**
- Managed by a **trusted authority**
  - Set permissions and role for users to ensure they only access parts of the system relevant to them
  - Integrity is critical
- Complicated authorization
  - Requirement for delegated authorization
  - Requirement for dynamic revocation of authorizations
- Logs of system accesses are required to be able to audit
- **Read accesses** can be frequent, **updates to information** are less frequent
  - Some delay in propagating updates is acceptable

# Use Case 3: Identity Management 2/2

Identity	
<b>Multi-party</b>	Required
<b>Trusted authority</b>	Not required
<b>Centralized operation</b>	Not required
<b>Data immutability &amp; Non-repudiation</b>	Required
<b>High performance</b>	Not required
<b>Data transparency &amp; Confidentiality</b>	Transparent
<b>Sample Result</b>	Blockchain

- Blockchain allows the roles, permission and privilege of users to be verified by the distributed peers
  - Remove the centralized administrator
  - Remove the centralized database
  - Ensure integrity of user identities, roles, and authorization
- Privacy is critical
  - Plaintext identity information is kept off-chain or encrypted on-chain

# Use Case 4: Stock Market 1/2

- A place where stocks, bonds and securities are traded
- Inherently involves **multiple entities** to issue and trade stocks
- Conventionally implemented by a **centrally-controlled and maintained** register
  - Regulatory approval is required for the operation of stock market
  - Regulatory approval may be required for the trading of specific stocks
- Stock market is a natural **trusted authority**
- **Integrity, immutability and non-repudiation** is critical
  - Ensure high-value trades cannot be undone by any party
- **Transaction history** is important in providing evidence for trades and current stock holdings
- Typically have a **high-volume, extremely low-latency** price-setting mechanism
  - Settlement can have **high throughput** but **does not have extreme latency**

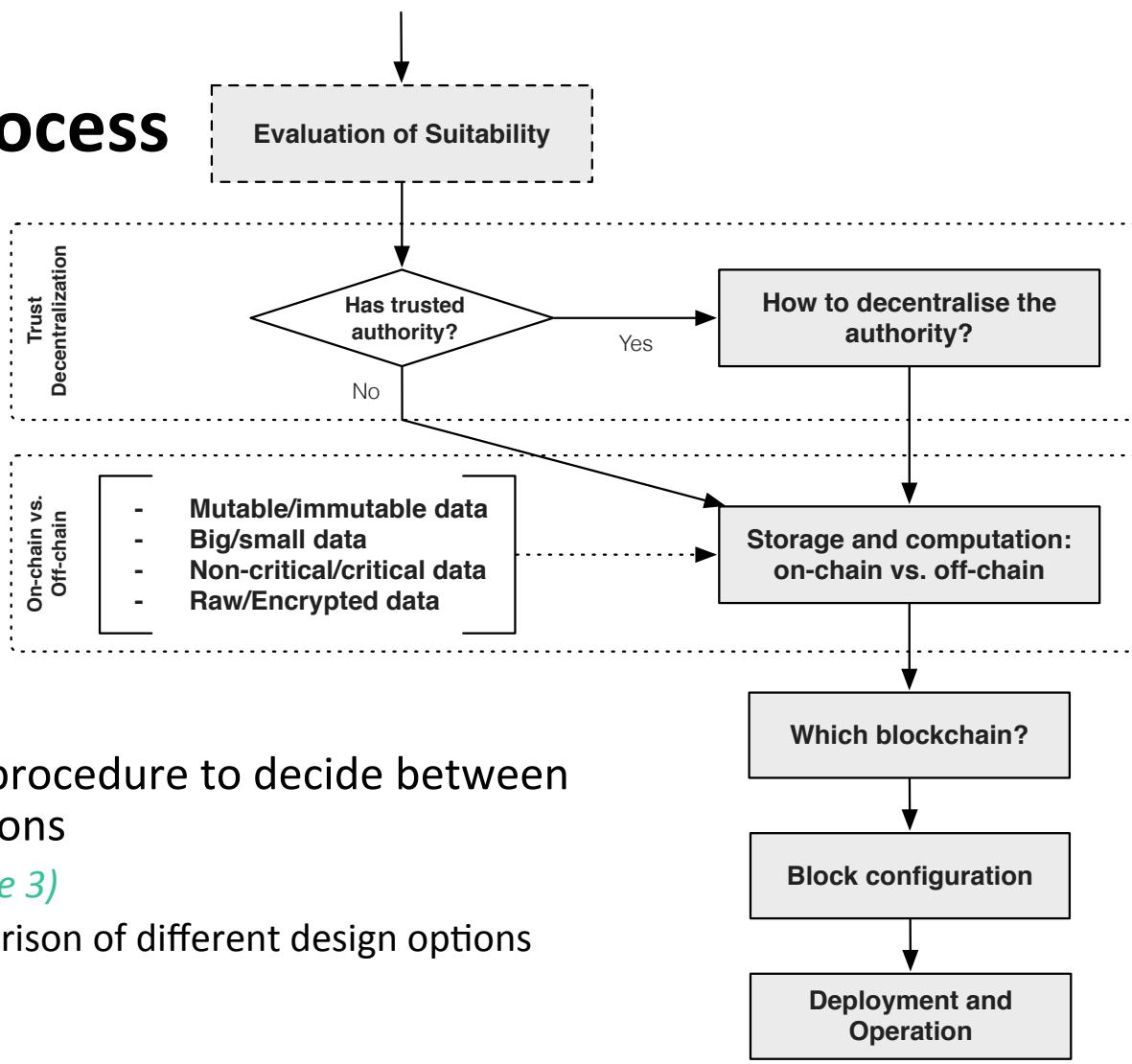
# Use Case 4: Stock Market 2/2

Stock market	
Multi-party	Required
Trusted authority	Not required
Centralized operation	Not required
Data immutability & Non-repudiation	Required
High performance	Required
Data transparency & Confidentiality	Confidential
Sample Result	Conventional System

- Blockchain allows settlement using peer confirmation
- Remove the centralized operation and authority
- **Data transparency is an issue**
  - All investors and market participants are exposed to blockchain participants
- Data immutability is crucial
  - Ensures no successful transaction can be tampered with by anyone
- Blockchain is **not highly suitable** for the operation of conventional regulated stock markets
  - Scalability issue
- NASDAQ offers Linq ledger for registration and settlement of securities

# Design Process for Blockchain-based Systems

# Design Process



- Every step is a procedure to decide between alternative options
  - Taxonomy (*Lecture 3*)
  - Systematic comparison of different design options

# Trade-off Analysis 1/2

Design decision	Quality
Block size, Block frequency	Scalability
Consensus protocol	Security
Public/Consortium/Private blockchain	Cost efficiency
Data structure	Performance

- Design decision
  - Improve the performance of one quality attribute
  - May harm the performance of other quality attributes

# Trade-off Analysis 2/2

## Encrypting data before storing it on a blockchain

- Increase confidentiality
- Reduce performance, may harm transparency or independent auditability

## Storing only a hash of data on-chain and keeping the contents off-chain

- Improve confidentiality and performance
- Partly undermine the benefit of blockchains in providing distributed trust
- Single point of failure reduces system availability and reliability

## Using private blockchain instead of public blockchain

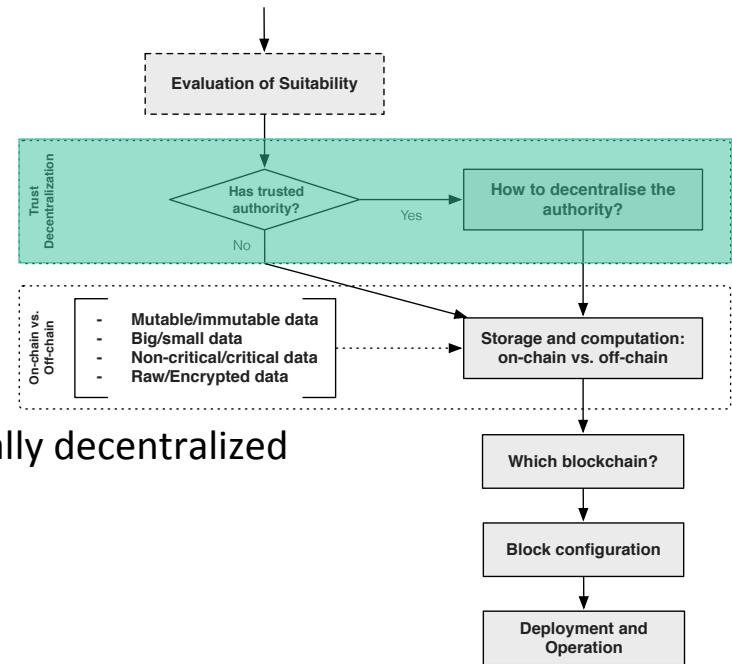
- Allow greater control over the admittance of processing nodes and transaction into the system
- Increase barriers to entry for participation, thus partly reduce some benefit of using blockchain

## Higher number of confirmation blocks

- Increase confidence in integrity and durability of transaction
- Harm latency

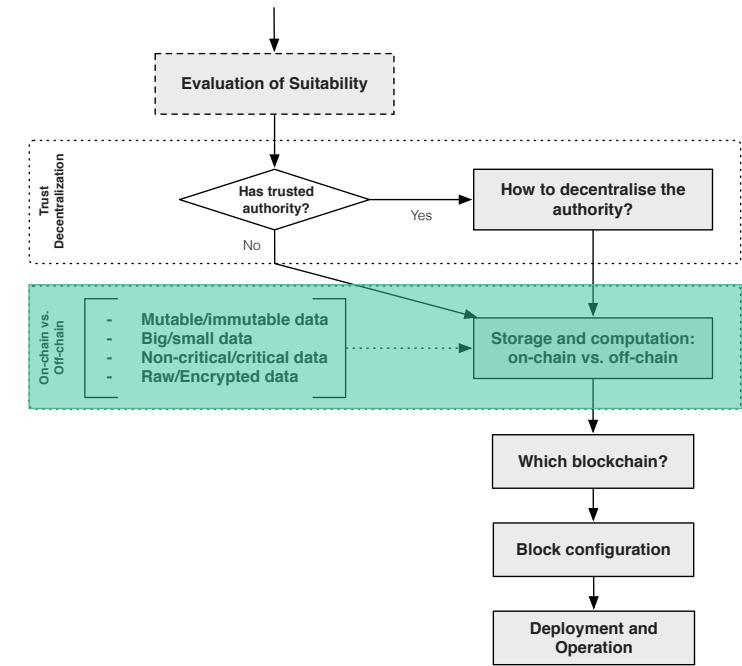
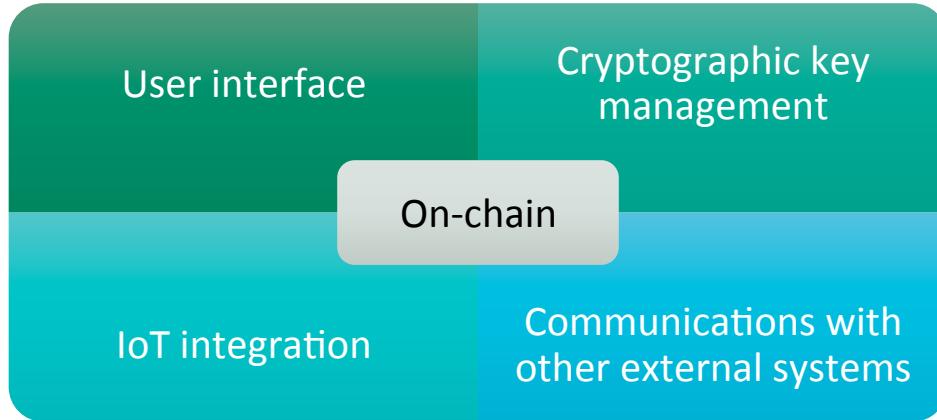
# Decentralisation

- Blockchain is used in scenarios
  - Where no single trusted authority is required
  - Where the trusted authorities can be decentralized or partially decentralized
- Deployment and operation of system spectrum



- Some components or functions are decentralized while others are centralized

# On-chain vs. Off-chain



- Many kinds of data are better stored off-chain
- Scalability reason—“Big” data
  - “Not tiny” data may be too large to store on blockchain
- Confidentiality reason – private data
- Dealing with legacy database

# Design Decisions and Their Impact

Design Decision	Option	Fundamental properties	Impact		
			Cost efficiency	Performance	Flexibility
Data	On-chain	Embedded in transaction (Bitcoin)	⊕	⊕	⊕⊕
		Embedded in transaction (Public Ethereum)	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕
		Smart contract variable (Public Ethereum)	⊕⊕	⊕⊕⊕	⊕
		Smart contract log event (Public Ethereum)	⊕⊕⊕	⊕⊕	⊕⊕
	Off-chain	Private / Third party cloud	⊕	~KB Negligible	⊕⊕⊕⊕
		Peer-to-Peer system		⊕⊕⊕	⊕⊕⊕
Computation	On-chain	Transaction constraints	⊕⊕⊕	⊕	⊕
		Smart contract			
	Off-chain	Private / Third party cloud	⊕	⊕⊕⊕	⊕⊕⊕⊕

# On-chain Data

- A common practice of storing item data
  - Raw data off-chain
  - On-chain just meta-data, small critical data and hashes of the raw data
  - On-chain data not just for integration with external data
- Store data in Bitcoin
  - *OP\_RETURN* (limited to 40 bytes)
  - Slow and costly
- Store data in Ethereum
  - Storing arbitrary data in transaction
    - Transaction size is limited by the maximum size of a block
    - Practically smaller transactions accepted by other users
  - Storing data in smart contract
    - As variable in a smart contract
    - As log event of smart contract
    - Variable is more efficient to manipulate, but less flexible due to the constraints of language

Colour coin as On-chain auxiliary data

- Overlays on Bitcoin to represent real world assets



# Cost Model

- Monetary cost of using public blockchains follows a different cost model than conventional software systems (*Second half of this lecture*)
  - More expensive
  - One-time cost for permanent storage
    - Partial refund of Ethereum

# Off-chain Data Storage

- Concern the interaction between blockchain and data storage facilities
- Cloud storage
  - Private cloud on the client's infrastructure
  - Public storage provided by a third party
  - Data replication is managed by the system or consumer
- Peer-to-peer data storage
  - IPFS (InterPlanetary File System)
    - Free, but availability depends on IPFS server that hosts the data
  - StorJ
  - Data is replicated automatically, or based on the user behavior



**STORJ**



# Design Decisions and Their Impact

Design Decision	Option	Fundamental properties	Impact		
			Cost efficiency	Performance	Flexibility
Data	On-chain	Embedded in transaction (Bitcoin)	⊕	⊕	⊕⊕
		Embedded in transaction (Public Ethereum)	⊕⊕⊕	⊕⊕⊕⊕	⊕
		Smart contract variable (Public Ethereum)	⊕⊕	⊕⊕⊕	⊕
		Smart contract log event (Public Ethereum)	⊕⊕⊕	⊕⊕	⊕⊕
	Off-chain	Private / Third party cloud	⊕	~KB Negligible	⊕⊕⊕⊕
Computation	On-chain	Peer-to-Peer system	⊕	⊕⊕⊕	⊕⊕⊕⊕
		Transaction constraints	⊕⊕⊕⊕	⊕	⊕
	Off-chain	Smart contract	⊕	⊕	⊕
		Private / Third party cloud	⊕	⊕⊕⊕⊕	⊕⊕⊕⊕

# Computation

- Different levels of expressiveness of on-chain computation
  - Bitcoin only allows simple scripts and conditions
    - Satisfied to transfer Bitcoin
  - Ethereum provides a Turing complete programming language
    - Not only perform conditional payments
    - Make modification to the working data in smart contract variables
  - DAML (Digital Asset Modelling Language)
    - More expressive than Bitcoin Script
    - Purposefully not Turing-complete: codify financial rights and obligations
      - In order to facilitate static analysis
- Benefit of on-chain computation
  - Inherent interoperability among the systems built on the same blockchain network
  - Neutrality of execution environment
  - Immutability of the program code once deployed



Digital Asset

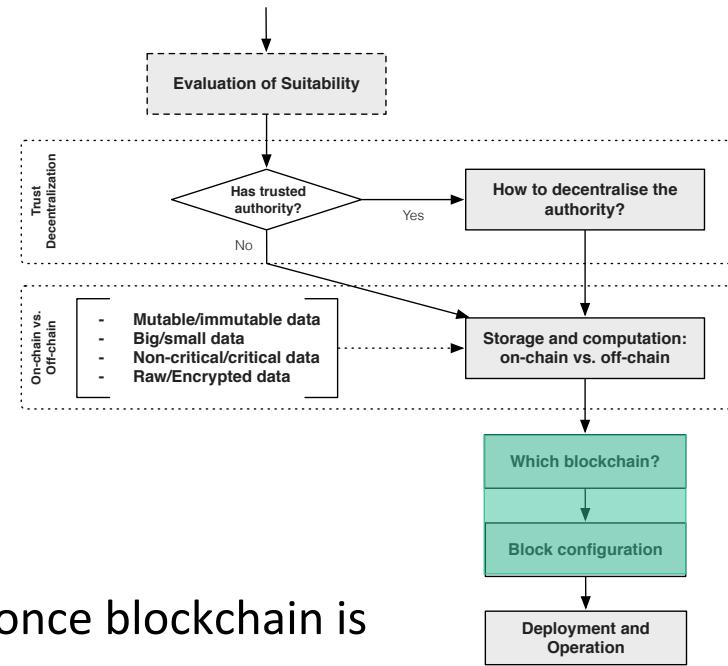
- Non-determinism
- Blocks impose an order on transactions Resolving non-determinism
  - Otherwise might affect the execution results

# Impact of Other Components

- Other components in the broader system has impact on design decisions
  - What's on-chain vs. what's off-chain
- Identity management (*One of the example use cases*)
  - Supports systems where has a requirement to know individual human or system involved in transactions
  - International payments have regulatory requirement to establish identity of participants
    - AML (Anti-Money Laundering) and CTF (Counter-Terrorism Financing)
  - Real-world identity is not required from technical perspective
    - Bitcoin transacting agents are only cryptographically identified
    - International exchange can be performed without establishing real-world identity
  - AML/CTF requirements are not obviated by the use of blockchain
  - Off-chain protocol might be used to store identity information
    - Privacy and confidentiality can be a challenge when storing on-chain

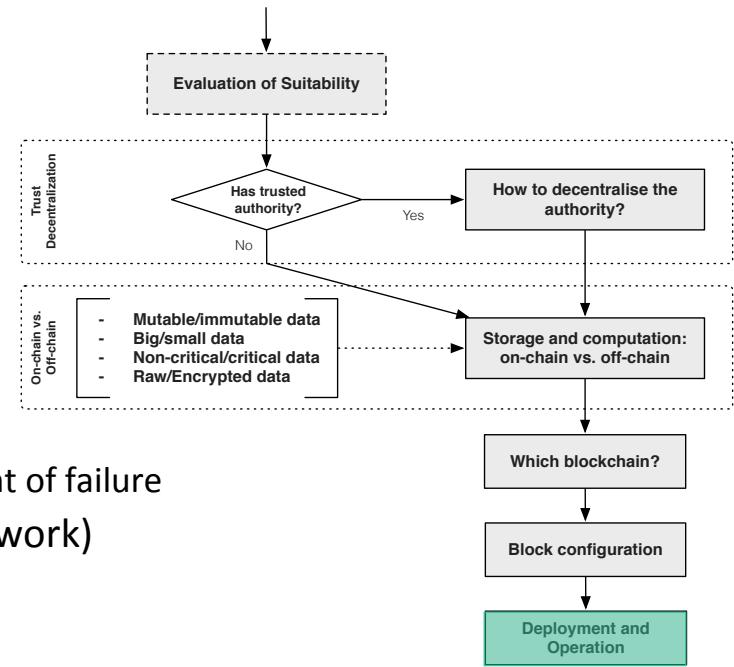
# Blockchain Selection

- Blockchain platform is selected
  - Requirement of the use case
  - Features of blockchain platform
  - Trade-off analysis
- Consensus protocol and other decisions are fixed once blockchain is selected
  - Hyperledger Fabric is an exception
    - Support pluggable implementations of various consensus protocols
  - Inter-block time is configurable for Proof-of-Work protocol
    - Through adjustments to the difficulty of mining



# Deployment

- Deployment has impact on quality attributes
- Deploying on cloud or using BaaS (Blockchain-as-a-Service)
  - Introduces the uncertainty of cloud infrastructure
  - Cloud provider becomes a trusted third-party and a single point of failure
- Deploying a public blockchain on a VPN (virtual private network)
  - Becomes a private blockchain
  - Permissioned access controls provided at the network level
  - VPN introduces additional network latency overhead



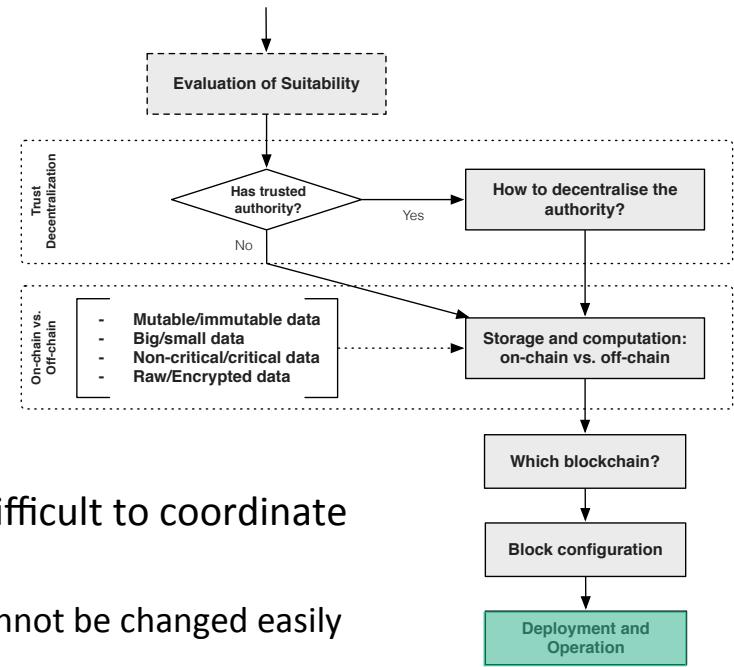
# Operation

- Operation challenges

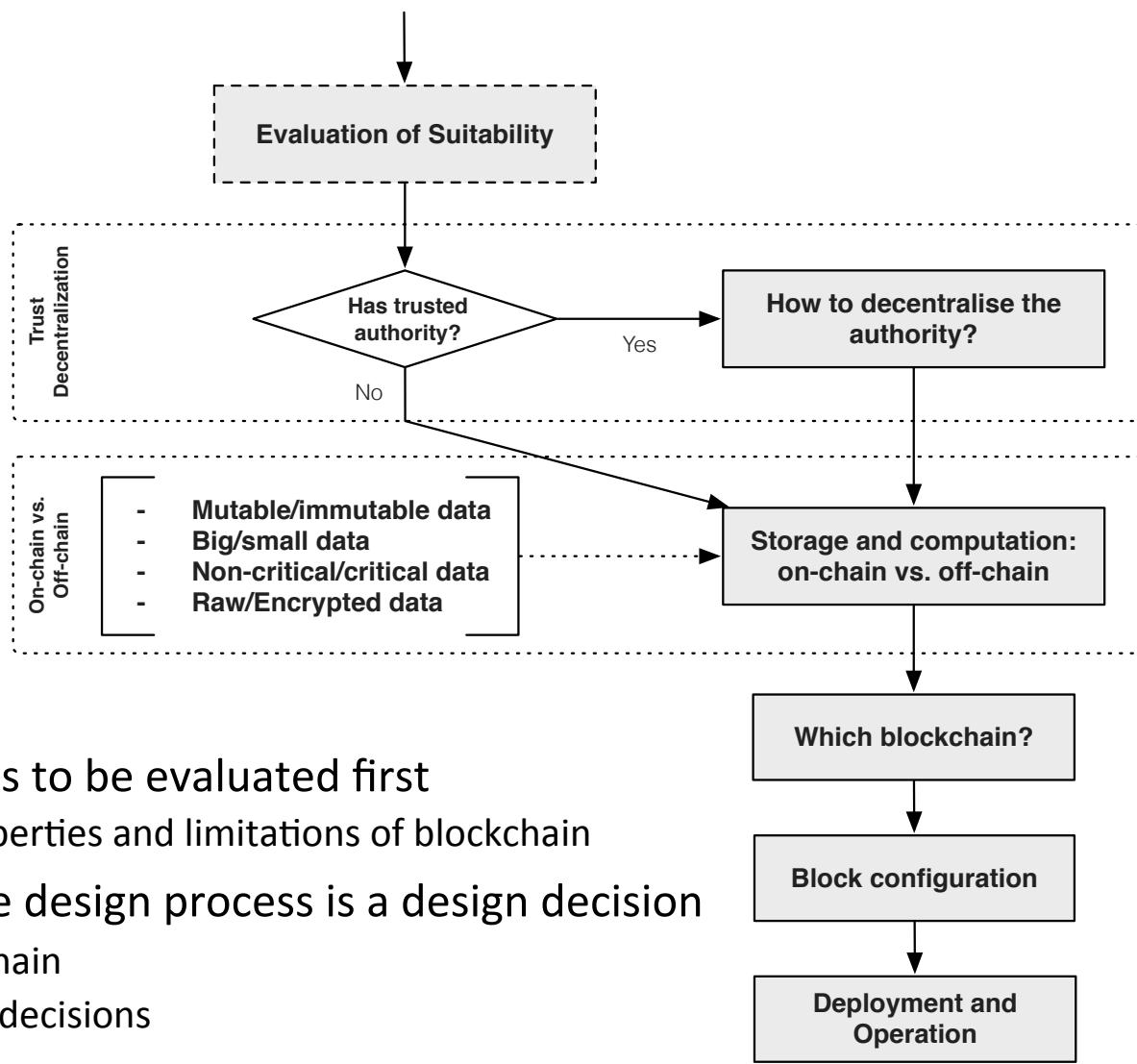
- Blockchain-based systems can be harder to modify than conventional systems
- Blockchain platform software running on multiple independently-operating nodes
- Updating software can be physically and administratively difficult to coordinate
- Blockchain is immutable by design, so cannot be updated
  - Trust is derived partly from the fact that the smart contract cannot be changed easily

- Governance

- Changes may be made to correct defects, add features or migrate to new IT context
- More like diplomacy in multi-party system with no single owner
- Blockchain is **NOT SUITABLE** to a system that needs to change frequently
- Learn from governance in open-source software
- Governance concerns software development, deployment and operation
- Immutable smart contract also simplify governance to some degree
  - Smart contract is available for execution while the blockchain operates normally



# Summary





# COMP6452 Lecture 5.2:

## Cost

# Outline

- On-chain Data Cost
- Smart Contract Cost
- Cost Models
- Using Cost Model

# Cost is important

- Monetary cost is as important for blockchain technologies as they are for conventional technologies
- Blockchain systems have different cost models
- Cost for storing too much data on-chain can explode quickly
- Blockchains enable distributed trust, but bring tradeoffs against execution cost and latency and throughput

# Arbitrary Bytes in Unspendable Bitcoin Tx

## Coinbase Transaction in Block

- Coinbase transaction mints new coins received by the miner who generates the block
- *Coinbase* parameter can contain arbitrary data
- Only the miner has access to this parameter

## *n*Sequence of a transaction

- Distinguish transactions from other Bitcoin transactions
- Presenting assets other than BTCs
- Every participants with the permission to submit transaction can set the value

## Fake account address

- Sending a small amount of coins to the fake account
- The coin is lost forever
- Use 1-to-n transaction
- Minimal amount of funds to avoid denial of service attack

## Conditional statements

- *OP\_IF*, *OP\_ELSE*, *OP\_ENDIF*
- Clauses cannot be reached under any condition
- Extra overhead

# ***OP\_RETURN***

- Official way to embed arbitrary data in a Bitcoin transaction
  - Returns immediately with an error
    - Included data is not interpreted as a script
  - Default Bitcoin client only relayed *OP\_RETURN* transactions up to 80 bytes
    - Reduced to 40 bytes in 2014
- Storing 80 bytes of arbitrary data on the Bitcoin costs roughly US\$0.459
  - Assuming a typical Bitcoin transaction with one input and one output (220 bytes)
  - The default transaction fee rate is  $2 \times 10^{-4}$  BTC/KB
- It is debatable whether Bitcoin should be used to record arbitrary data.

Exchange rates of US\$7650 / BTC from 2 August 2018 ([https://poloniex.com/exchange#usdt\\_btc](https://poloniex.com/exchange#usdt_btc))

# Storing Data on Ethereum Transaction

- Theoretically allows storing arbitrary data of any size
- Storing 80 bytes of arbitrary data on Ethereum costs roughly US\$0.22
  - Every transaction has a fixed cost of 21,000 gas
    - Gas is the internal pricing for executing a transaction of storing data
  - Every non-zero byte of data costs additional 68 gas
  - Total cost of storing 80 bytes via transaction is 26,440 gas
    - Assuming all bytes are non-zero

80 bytes on Bitcoin  
costs US\$0.459

Exchange rate of US\$420 / ETH from 2 August 2018 ([https://poloniex.com/exchange#usdt\\_eth](https://poloniex.com/exchange#usdt_eth))  
gas price of  $2 \times 10^{-9}$  ETH (2 Gwei) on Ethereum



# Storing Data in Smart Contract

- Storing data as a variable in a smart contract
  - Cost is based on the number of *SSTORE* operations
  - 1 *SSTORE* operation that changes the data from zero to non-zero (20,000 gas)
  - Transaction as the carrier costs a base 21,000 gas
  - Data payload costs extra gas
    - Function signature and the actual data
  - Cost for creating the smart contract depending on its complexity
  - Total cost is > US\$0.036 ( $20,000 + 21,000 + 32 \times 68$  gas )
  - Subsequent transactions to **update** data costs 5,000 gas (keeping the data as non-zero)
  - Cost of subsequent transactions is ~ US\$0.024 ( $5000 + 21,000 + 32 \times 68$  gas)
  - Less flexible due to the constraints of Solidity on the value types and length
- Storing data as a log event in a smart contract
  - 1 log topic costs 375 gas
  - Every byte of data costs an extra 8 gas
  - Transaction as the carrier costs a base 21,000 gas
  - Total cost is ~US\$0.018 ( $21,000 + 375 + 32 \times 8$  gas)

Storing 32 bytes of data  
(simple types of Solidity are 32 bytes)

# Smart Contract Cost

- Cost charged on transactions in relation to their complexity
  - Base cost for any transaction (21,000 gas)
  - Variable components
    - Data attachments
    - Contract execution is charged per *bytecode* instruction
    - Additional cost for contract deployment
- *Gas*
  - All cost follows a fixed [pricing table specified](#) in the unit *gas*
  - Gas cost is converted to Ether
  - User-defined *gas price* factor
    - How much Ether-per-gas is the transaction creator willing to pay
    - Default value is the current market rate
      - average over previously included transactions

# Gas Limit

- Block gas limit
  - Sum of gas used by the set of transactions included in a given block cannot exceed this limit
  - Set by the miners
  - Defined in terms of gas usage
    - Cannot be influenced by variations the user has power over
      - For example, underbidding the market price
  - Making it a limit of complexity for new blocks
  - An upper bound to throughput scalability
    - Cost of transactions vary
    - Non-trivial to understand how the bound relates to transaction throughput
- <https://ethgasstation.info/index.php>

# Cost Model

- **Cost Model of Blockchain Infrastructure (Ethereum)**
  - Turing complete language to implement business logic
- **Cost Model of Amazon Web Services (Amazon SWF)**
  - Dedicated to process execution
    - Commonly-used workflow patterns and messaging patterns
  - AWS is a leading commercial cloud computing provider  
*(Use the execution of an instance of a business process model as sample application)*



# Ethereum Transaction 1/2

- 3 types of transactions
  - Financial transfer, message call and contract creation
  - Basic elements: `from`, `to`, `gasLimit`, `value` and `data`
- Financial transaction
  - From/to sender/recipient of the transaction
  - Value the amount transferred
  - Data (optional) data in arbitrary form
    - E.g. XML, pictures
    - Fee for a transaction covers the cost for storing the data permanently

# Ethereum Transaction 2/2

- Message call transaction
  - Invoke a function of a contract
  - From/to sender/recipient of the transaction
  - Value (optional)
  - Data the method to be invoked and the parameters
  - **gasLimit** maximum gas can be used in this transaction
    - Gas is paid for each executed bytecode instruction
- Contract creation transaction
  - to NULL
  - Data the contract bytecode
  - Value (optional)

Cost of executing business process

Cost of deploying business process

# Contract Creation Cost 1/2

- Contract creation transaction
- Data compiled bytecode
  - Permanent storage incurs cost
- Value (optional)
  - Endowment upon initialization
- Ethereum address is assigned to it
  - Calculated with a deterministic function depending only on the creator's Ethereum account

$$C_{pload} = \text{payload (in bytes)} \times C_{gas/byte}$$

- Cost of payload for contract bytecode is 200 gas per byte
- Cost of payload for data in a financial transaction/message call is 68 gas per non-zero byte and 4 gas per zero byte

$$C_{create} = C_{tx} + C_{addr} + C_{pload} + C_{fndef}$$

- $C_{tx}$ : 21,000 base gas for transaction itself
- $C_{addr}$ : 32,000 for allocating address
- $C_{fndef}$ : consumed by the function definition

# Contract Creation Cost 2/2

- Contract can be created by another contract
  - Not via transaction
  - Cheaper Without  $C_{tx}$

$$C_{create}' = C_{addr} + C_{pload} + C_{fndef}$$

# Contract Execution Cost

- Function call cost

$$C_{execute} = C_{tx} + C_{pload} + C_{fnexe}$$

- $C_{tx}$ : 21,000 base gas for transaction itself
- $C_{pload}$ : cost of data payload
- $C_{fnexe}$ : consumed by the opcode present during the function execution

# Gas Cost → Ether → Another Currency

$$C_{\$} = C_{Gas} \times \text{gasPrice} \times 10^{-18} \times \text{EXC}_{ETH2CUR}$$

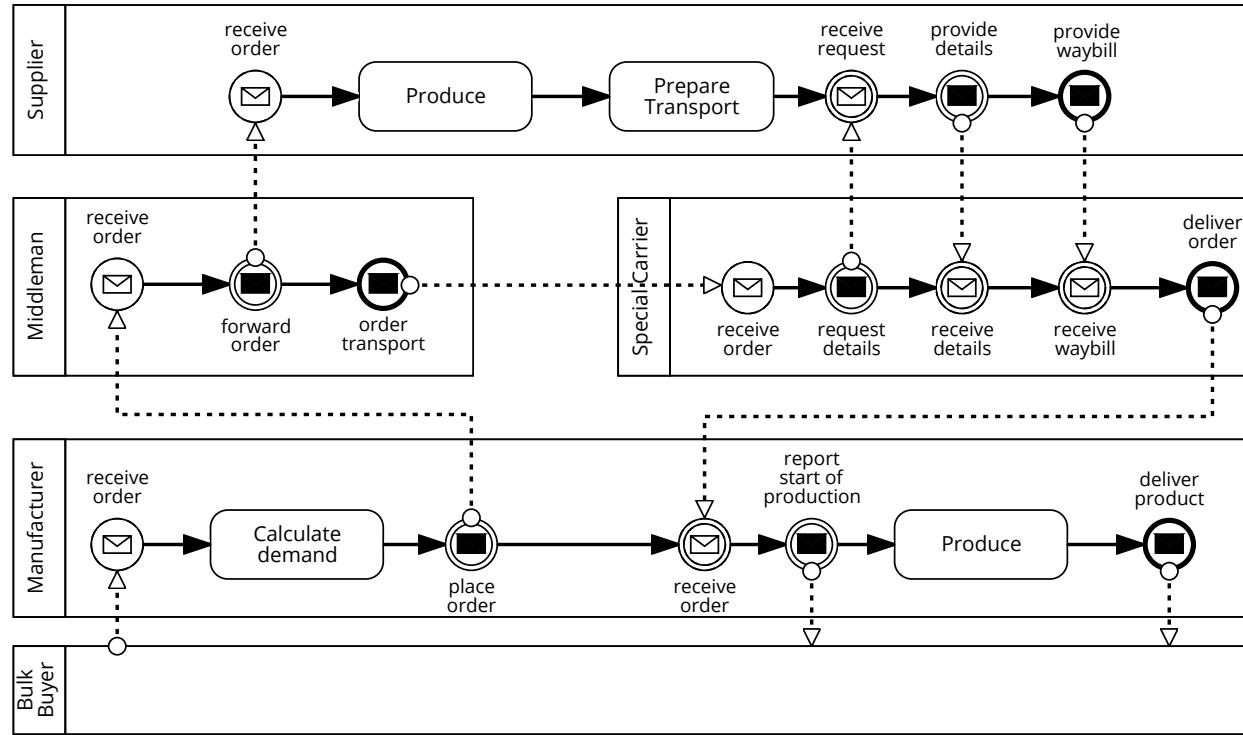
- $C_{Gas}$  : cost in gas
- Gas price in *wei* ( $10^{-18}$  Ether)
- EXC<sub>ETH2CUR</sub> : Exchange Rate

<https://coinmarketcap.com/currencies/ethereum/>

<https://fx-rate.net/ETH/USD/>

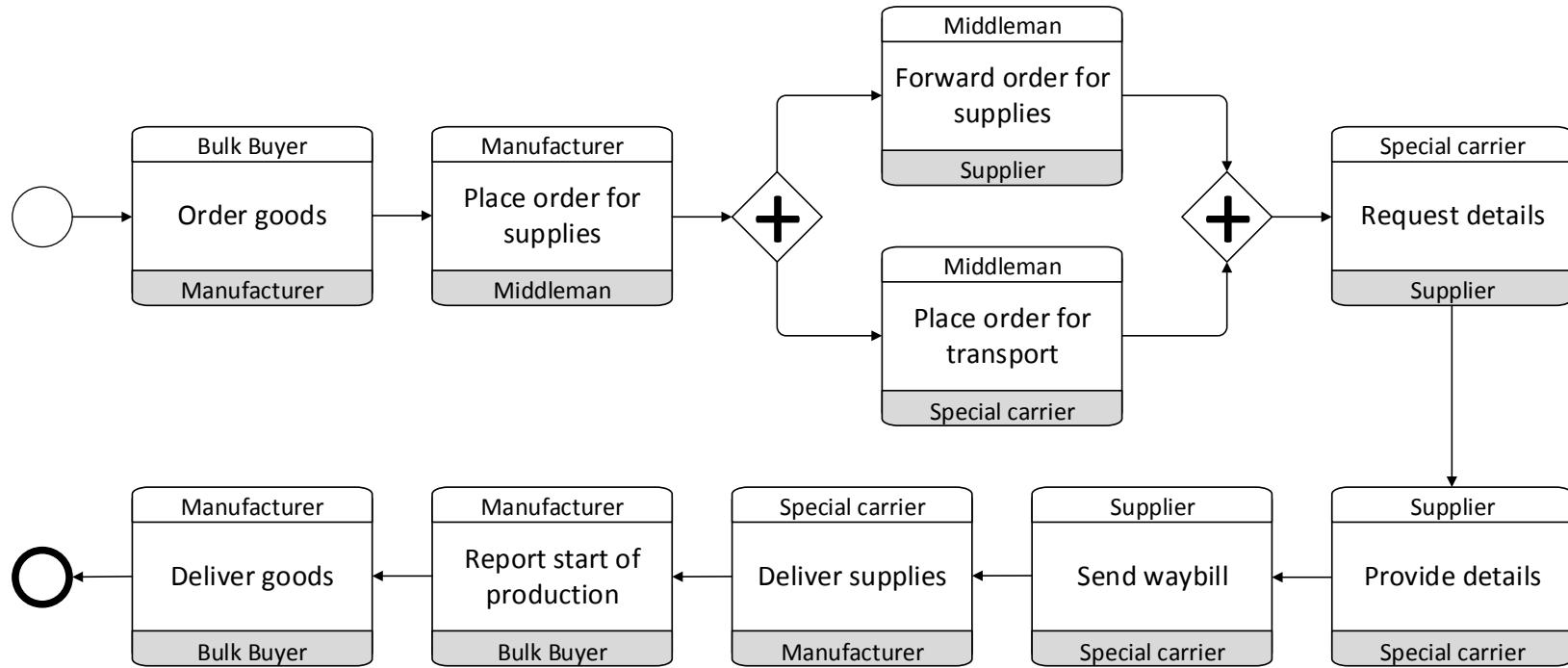
# Cost of Interaction Component 1/5

## Supply Chain Process in BPMN Orchestration



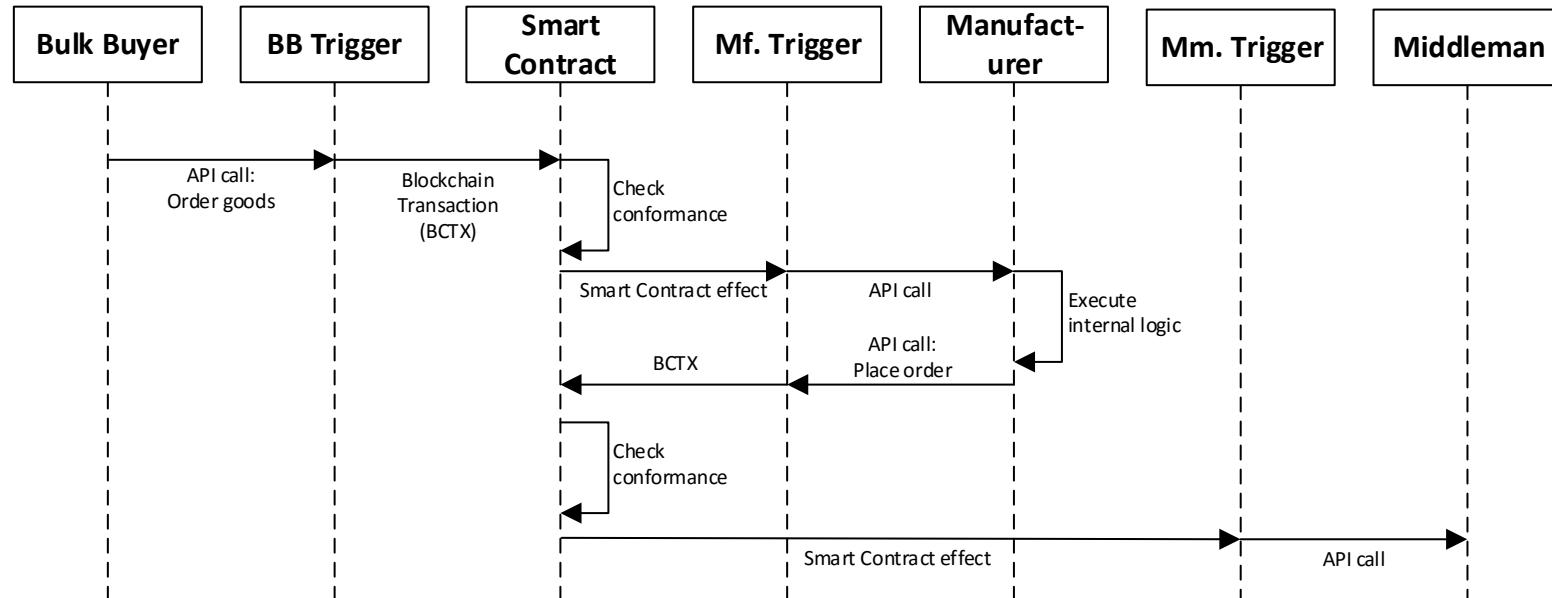
# Cost of Interaction Component 2/5

## Supply Chain Process In BPMN Choreography



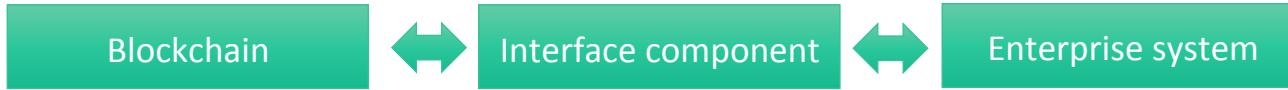
# Cost of Interaction Component 3/5

## Sequence Diagram of the First Two Tasks



- Interaction between internal process implementation, triggers, and process instance smart contract

# Cost of Interaction Component 4/5



- Assuming the interface component is running on AWS

$$C_{comp} = EC2_{price}(ec2_t) \times time$$

- $EC2_t$  : the set of all available VM types in AWS  $ec2_t \in EC2_t$
- Capacity of VM
- $TP_{bc} : EC2_t \rightarrow R$
- Determines VM type based on workload
- $f_{bc} : (TP_{bc}, WL_{bc}) \rightarrow EC2_t$
- Cost of running a VM of this type per billing time unit (BTU)
- $EC2_{price} : EC2_t \rightarrow R$

# Cost of Interaction Component 5/5



- Interface component operates a full node
  - Synchronize the blockchain if the VM is not constantly online
  - Required duration includes the time of synchronization
- Ethereum clients allows fast synchronization (“fast” flag)
  - Downloading transaction receipts instead of the full set of known blocks
    - Shows that the transactions happened but
    - NOT show the results of the smart contract function execution
    - Less evidence for integrity
    - Only be done when downloading the blockchain from scratch
      - Takes on the order of hours to days for public Ethereum blockchain

# Cost Model

- Cost Model of Blockchain Infrastructure
- **Cost Model of Amazon Web Services (SWF)**

(Use the execution of an instance of a business process model as sample application)



# Amazon SWF



- Simple Workflow Service (SWF) provided by AWS
- Representative for cloud-based business process execution
- Clear mapping to the process model
- Tiered pricing model
  - More usage results in cheaper cost per unit

## Workflow Executions

A workflow is a set of tasks executed in a certain order (sometimes with a set of conditional flows or loops). Each time that a workflow is executed, it is considered a distinct workflow execution. You pay for workflow executions when you start them (i.e. their first task becomes available for application hosts to execute) and for each 24-hour period until they are completed. The first 24 hours of workflow execution are free.

### Start a Workflow Execution

Region: US East (Ohio) ▾

- \$0.00 for first 1,000 workflow executions
- \$0.0001 per workflow execution above the free tier

### Data Transfer \*\*

Region: US East (Ohio) ▾

	Data Transferred	Pricing
<strong>Data Transfer IN</strong>		
All data transfer in		\$0.00 per GB
<strong>Data Transfer OUT***</strong>		
Up to 1 GB / Month		\$0.00 per GB
Next 9.999 TB / Month		\$0.09 per GB
Next 40 TB / Month		\$0.085 per GB
Next 100 TB / Month		\$0.07 per GB
Greater than 150 TB / Month		\$0.05 per GB

DATA  
61

CSIRO

# SWF Cost Model

- Main elements: workflow, actor, task, and signal
- Workflow: A collection of activities in a specified sequence
  - An instance of business process
- Actor: Play participant roles from the business process
- Activity (task): Schedule a notification to the appropriate actor to proceed with the next activity
- Decision (task):
  - Determine whether the current state of execution conforms to the workflow
  - Determine which activity to execute next
- Signal: External triggered event to a currently executing workflow

# Element Mapping

Business Process	Blockchain	Amazon SWF
Process instance	Instance of Smart Contract	Workflow
Conformance checking	Contract execution	Decision task
Activity	Contract execution	Activity task
Incoming message	Transaction	Signal
Outgoing message	Entry in contract event log	Notification

- Conformance checking is a technique in process mining
  - compare an existing process model with an event log produced by the process model
  - check if what happened in reality conforms to the process model and can be used at runtime

# Base Cost of Workflow Instance

$$C_{wf} = \#wf \times SWF_{wf}$$

- $\#wf$  : Number of instances
- $SWF_{wf}$  SWF cost of starting a workflow execution

# Cost of Scheduling Tasks

$$C_{task} = (\#actTask + \#decTask) \times SWF_{task}$$

- $\#actTask$  : number of activity tasks
- $\#decTask$  : number of decision tasks
- $SWF_{task}$  : price per task
- No. SWF activity tasks = No. activities in a business process
- No. SWF decision tasks = No. activities in a business process
- Schedule a decision task every time receiving a signal (completion of a activity)
- Process initiation creates a decision task to instruct the workflow to wait for the first signal

# Cost of Signals

$$C_{sig} = \#signals \times SWF_{signal}$$

- $\#signals$  : number of signals
- $SWF_{signal}$  : price per signal

# Cost of Data Retention and Transfer

$$C_{ret} = (execT + retT) \times SWF_{ret}$$

$$C_{dat} = payload \times SWF_{data}$$

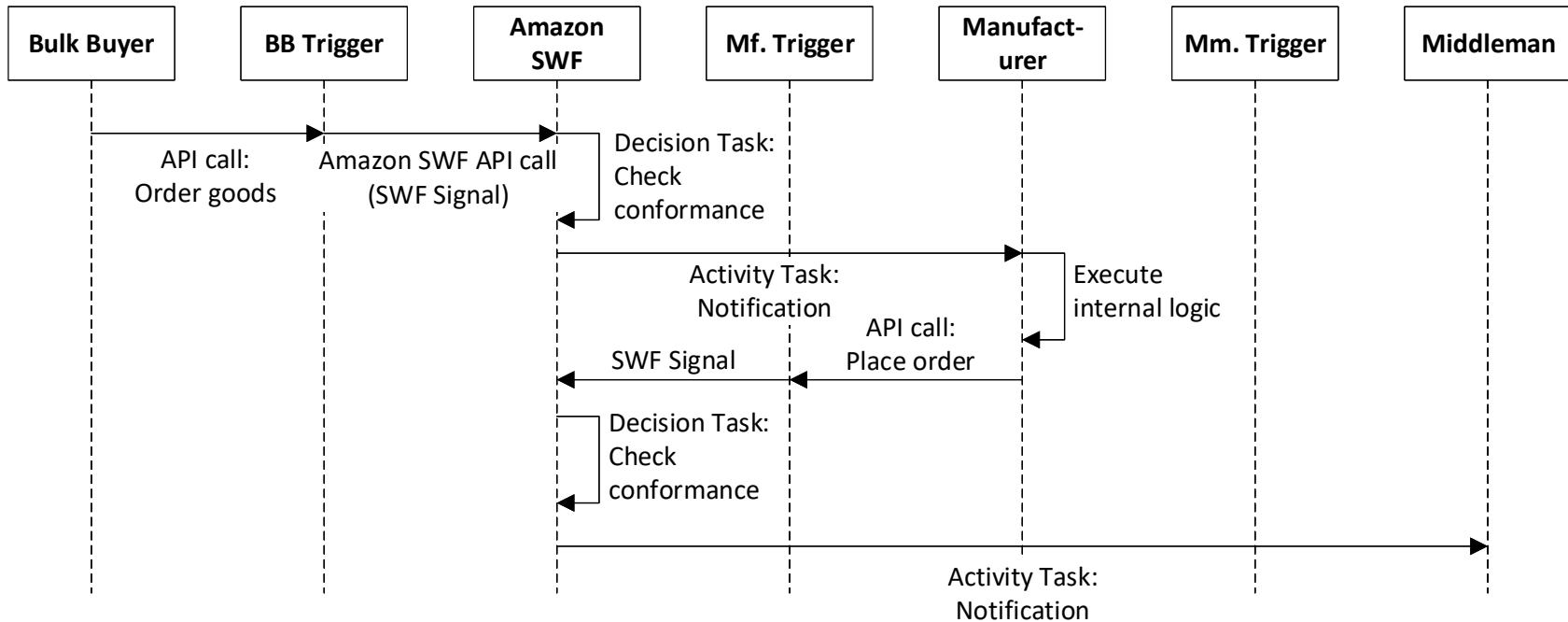
- $retT$ : user-specified duration for generated data being retained
  - Charged for storage per 24 hours
- $execT$ : Workflow execution time
  - Charged per 24 hours at the same rate as data retention cost
- $SWF_{ret}$ : SWF cost rate
- $Payload$ : inwards and outwards data size
- $SWF_{data}$ : price per data unit

# Coordination Cost

$$C_{swf} = C_{wf} + C_{task} + C_{sig} + C_{ret} + C_{dat}$$

# Cost of Interaction Component 1/2

## Using Amazon SWF



- Task execution requires actor running a *Amazon SWF worker* module
  - On AWS EC2 or internal infrastructure
  - Execute both decision task and activity task

# Cost of Interaction Component 2/2

- Cost of VMs
  - Triggers and Amazon SWF workers
- Throughput per VM type

$$TP_{swf} : EC2_t \rightarrow \text{IR}$$

- VM type depends on capacity of VM types and the workload
  - The throughput values are different from the ones for blockchain triggers

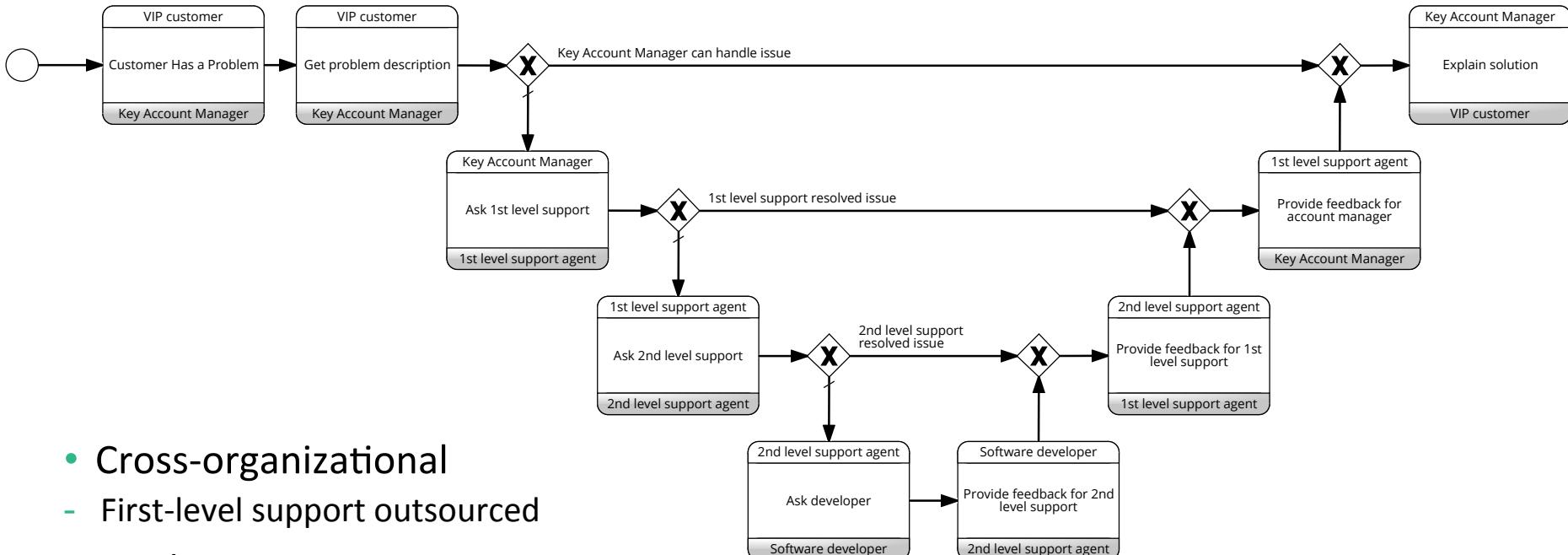
$$f_{swf} : (TP_{swf}, WL_{swf}) \rightarrow EC2_t$$

- Cost of running the VMs for the time needed

$$C_{comp} = EC2_{price}(ec2_t) \times time$$

- Minimum requirement is one VM to host the trigger and SWF worker
- Preferable setup is that each participant at least one VM
  - Host their own trigger and worker

# Evaluating Cost Model



- Cross-organizational
  - First-level support outsourced
- Incident Management Process
  - 9 tasks
  - 6 gateways
  - 4 conforming traces

# AWS VM Throughput Benchmark

VM Types	vCPU Specifications	Memory (GiB)	Decision limits		
t2.small	1 Intel Xeon E5-2676 2.40 Ghz v3 w/ Turbo up to 3.3 Ghz	2	Decision	Bucket size	Refill rate / s
m3.medium	1 Intel Xeon E5-2670 2.50 GHz v2 (Ivy Bridge) Processors	3.75	RequestCancelExternalWorkflowExecution	100	10
m3.large	2 Intel Xeon E5-2670 2.50 GHz v2 (Ivy Bridge) Processors	7.5	ScheduleActivityTask	500	100
m3.xlarge	4 Intel Xeon E5-2670 2.60 GHz v2 (Ivy Bridge) Processors	15	SignalExternalWorkflowExecution	500	10
			StartChildWorkflowExecution	500	2
			StartTimer	1000	142



- AWS EC2 VM types and specification
- Throughput

Metrics	Blockchain			Amazon SWF	
	t2.small	m3.medium	m3.large	m3.medium (default)	m3.medium (incr. limit)
Transactions or Signals	13,580	7,336	20,104	73,871	152,404
Network In (MB)	102	114	128	138	168
Network Out (MB)	195	131	278	353	376
Duration (sec)	3,610	3,605	3,604	3,605	3,605
Average Tx/sec or Average signal/sec	3.8	2.0	5.6	20	42

# Incident Management Blockchain Cost

- 32 process instances with a total 256 transactions
- Deployment of factory contract costs 0.032 Ether (One-time cost)
- Each run with data transformation costs 0.035 Ether
  
- Total cost is approx. US\$1.34
  - Exchange rate is US\$420 / ETH
  - Gas price of 2 Gwei

# Incident Management Amazon SWF Cost

- EC2 *t2.micro* VM for trigger and SWF task worker
- Process instances executed in sequence
- US\$0.92 for 1,000 process instances
- US\$0.0009 per instance
  - Data retention is 1 day
- US\$0.0027 per instance
  - Data retention is 365 days
- Cost breakdown 

Element	elements in experiment	Unit cost (US\$)	Total cost (US\$)
Decision Task	15,000	0.000025	0.375
Activity Task	7,000	0.000025	0.175
Signal	7,000	0.000025	0.175
Workflow	1,000	0.0001	0.1
Retention (24h)	1,000	0.000005	0.005
Execution Time (24h)	1,000	0.000005	0.005
Data Transfer	1	0.09	0.09

# Comparison

- Cost on blockchain is three orders of magnitude higher than on Amazon SWF
  - Excluding the one-time factory contract deployment
- Blockchain stores the result permanently
  - As long as the blockchain is in existence
- Ongoing cost for data storage on Amazon SWF
  - Store for 243,863 days (approx. 668 years) to reach break-even

# Volatility of Cryptocurrency

- Sensitive to the volatility of the exchange rate

Costs	Ethereum (in Ether)	Exchange Rate (in US\$)				
	0.10	1.00	10.00	100.00	1,000.00	
Incident Management (contract deployment)	0.0032	0.00032	0.0032	0.032	0.320	3.20
Incident Management (per process instance)	0.00347	0.000347	0.00347	0.0347	0.347	3.47

- Comparison
  - Exchange rate
  - Retention rate

Costs	SWF cost (in US\$)	SWF vs Blockchain cost comparison					Break-even rate (in US\$)
		\$0.10	\$1.00	\$10.00	\$100.00	\$1,000.00	
Incident (24 hours)	0.000925	0.375 (0)	3.751 (+1)	37.51 (+2)	375.14 (+3)	3,751.35 (+4)	0.27
Incident (99 years)	0.181595	0.002 (-3)	0.019 (-2)	0.191 (-1)	1.91 (0)	19.11 (+1)	52.33

# Why Blockchain

- Blockchain provides trusted storage and execution environment
  - No trust in any single third-party
- Conventionally participants need to jointly agree on a mutually-trusted third party
  - E.g. AWS (for confidentiality and truthful execution)
  - The party controlling the Amazon SWF account
- Public blockchain inherently supports payment and escrow
  - Sending cryptocurrency with existing messages would not incur additional cost
    - Due to a flat fee structure
  - Offset the premium cost of distrust
    - Commercial escrow service charge 0.5% to 3.25%
    - Lower the cost of process executions involving monetary transaction

## Co-opetition

- Organizations cooperate for cases to achieve business goals
- Compete in other cases

# Cost vs. Maintainability

- Deployment methods impact cost and non-functional properties
  - (1) One smart contract with two functions
  - (2) Two smaller contracts, each implementing one function
    - One contract acts as an entry point
- The first has lower deployment cost
  - (2) needs to pay  $C_{tx}$  and  $C_{adr}$  twice
  - The payload of contracts in (2) is higher, as there are header bytes in the payload
- (1) is not as maintainable as (2)
  - One function needs to be modified
    - (1): updated contract needs to be redeployed as a whole
    - (2): only one contract is redeployed
    - (1): the triggers need to be updated with the new address
    - (2): might be avoided

# Cost vs. Scalability of Triggers

- Additional resources are needed to accommodate increasing workload
  - Vertical scaling (bigger VM)
  - Horizontal scaling (more VMs)
- Blockchain nodes can scale vertically
  - Horizontal scaling has complication
    - Easy to add additional VMs into the network
    - Using one account from multiple VMs may lead to double-spending
    - Using different accounts on different VMs
      - Maintainability issues and increase storage costs
- SWF can scale both horizontally and vertically
  - Vertical scaling by choosing larger VM
  - Horizontal scaling by adding new VMs (registering it with SWF)
    - SWF then acts as load balancer
      - Distributing requests to multiple VMs

# Summary

- Cost of basic compute and storage on public blockchain have different cost model than conventional cloud
- Public Ethereum and Amazon SWF are compared using business process
  - Construct and benchmark cost model for both infrastructures
  - Cost on public Ethereum blockchain is three orders of magnitude higher than on Amazon SWF
- Cost model incorporates exchange rate is important
  - Given the high volatility of the exchange rate
- Cost is often in tradeoff with other non-functional properties
  - Maintainability and Scalability

# Course Outline – next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
5th	18 Mar	Sherry Xu	NPFs 1	6. Design Process for Applications on Blockchain 9. Cost	
6th	25 Mar	Mark Stapes	NPFs 2	10. Performance	Mid-term Exam (1 hour)
7th	1 Apr	Mark Stapes	NPFs 3	11. Dependability and security	Assignment 2 out (Monday before lecture)



# THANK YOU

**Xiwei Xu** | Senior Research Scientist

Architecture & Analytics Platforms (AAP) team

t +61 2 9490 5664

e [xiwei.xu@data61.csiro.au](mailto:xiwei.xu@data61.csiro.au)

w [www.data61.csiro.au/](http://www.data61.csiro.au/)

[www.data61.csiro.au](http://www.data61.csiro.au)



# NFPs 2: Performance

Mark Staples

Email: [mark.staples@data61.csiro.au](mailto:mark.staples@data61.csiro.au)

[www.data61.csiro.au](http://www.data61.csiro.au)

# Outline

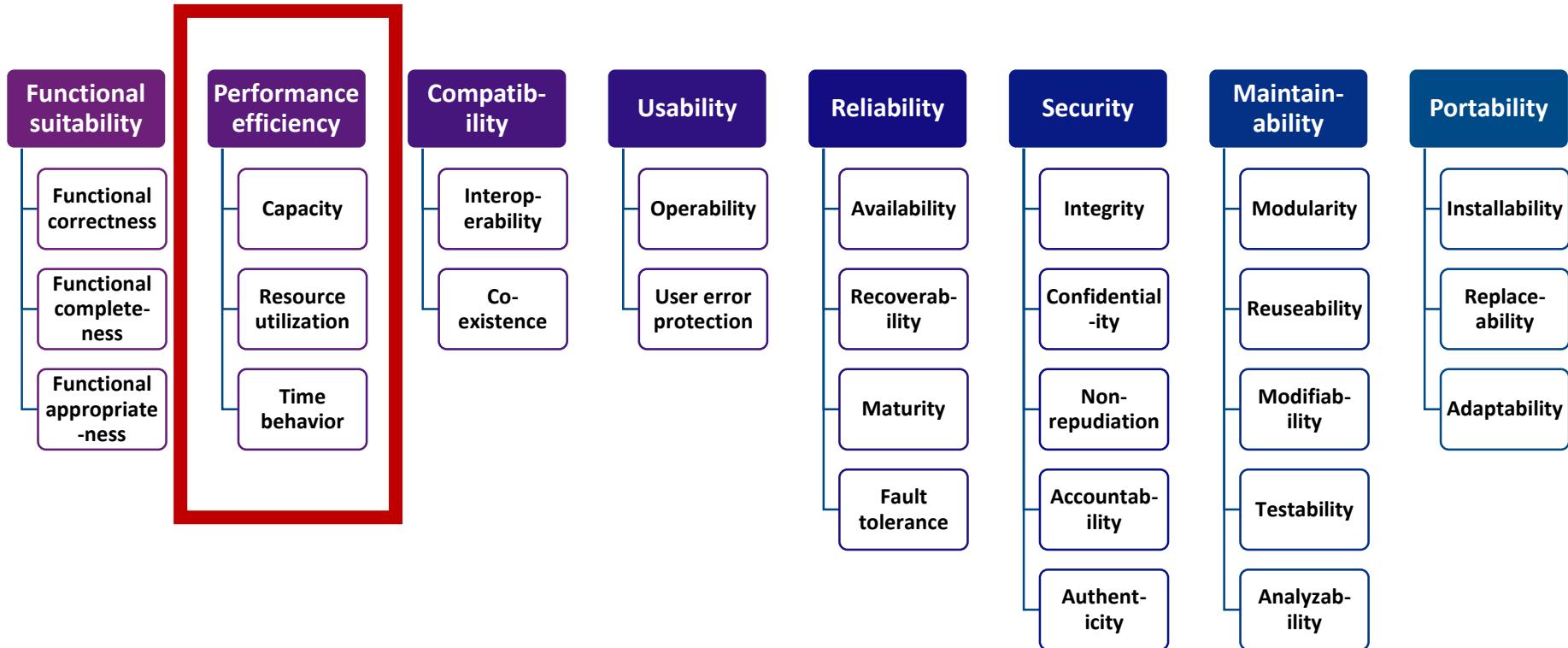
- Performance
- Latency
  - Transaction Inclusion in Public Blockchain
  - Automating Process Execution on Public Blockchain
- Throughput
- Summary



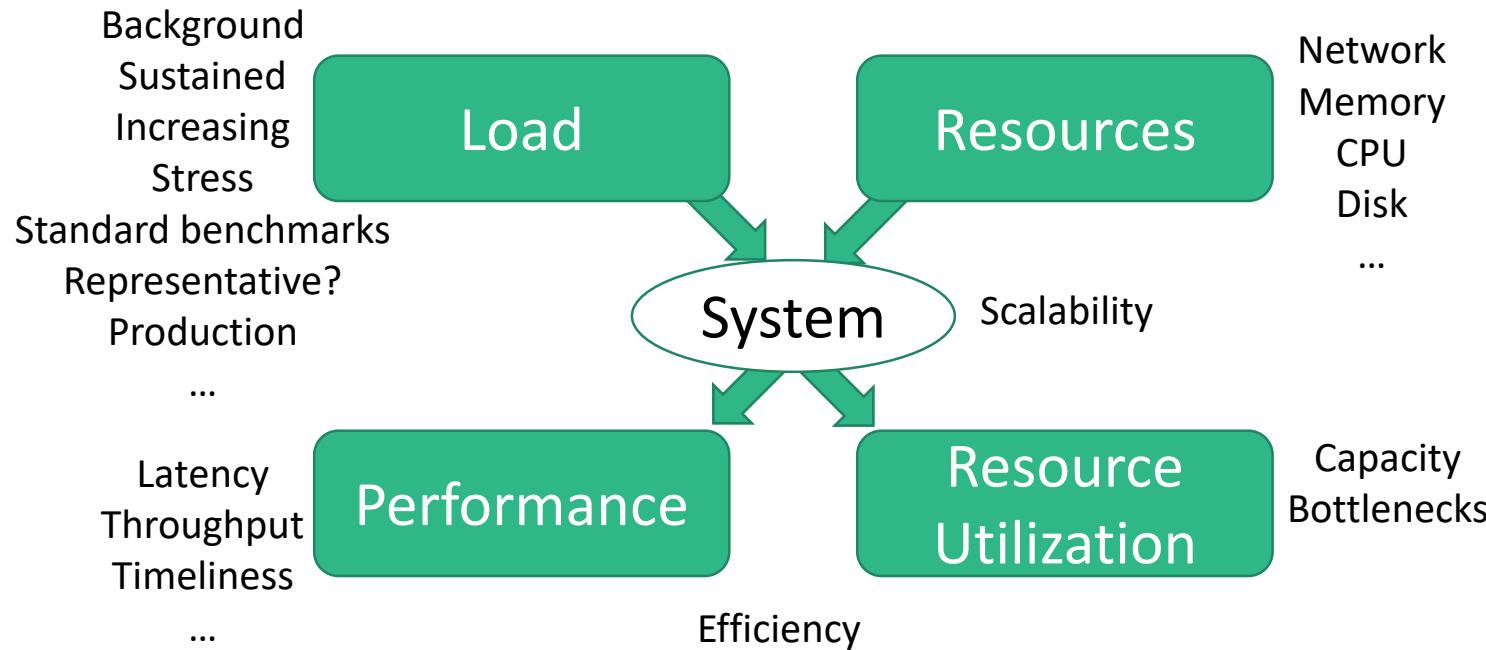
# Performance



# ISO/IEC 25010:2011 Quality Model



# Performance-Related Considerations



# Some Important Varieties of Performance

- Latency
  - How quickly does the system respond to a request?
- Throughput
  - How many requests can the system process in a fixed unit of time?
- Timeliness
  - Will the system always process a request in a specified time window?
- Scalability is an orthogonal issue
  - How does *<performance>* change under increasing load & system resources?
    - (Can also be interested in response when decreasing resources, under variable load)

# To Understand Performance, You Need Measurements

- Application-Level measurements
  - Overall performance; check requirements met; validate results from detailed models
- Component-Level measurements/ Micro-benchmarking
  - Piece-wise drivers of performance; observe interactions; identify bottlenecks
- Load testing – check system response & utilisation under increasing load
- Stress Testing – check maximum capacity; find resource bottlenecks
- Soak Testing – check system response under long-duration load
- “Benchmarking crimes”
  - <https://www.cse.unsw.edu.au/~gernot/benchmarking-crimes.html>

# Predicting Performance?

- Analytic formulae
  - Highly abstract/simplified models
  - Quick & dirty
  - Only needs design & component benchmarks
- Simulation
  - Uses a model; many abstractions/simplifications
  - Only needs a design & benchmarks, but takes work
  - Starts to explore variation & interactions
- Monitoring
  - Needs real deployed system
  - Needs a monitoring framework
  - Most valid results; e.g. actual effects!

Only needs a design  
& benchmarks

Needs a real  
deployed system

Highly approximate



# Some Resources in Blockchains

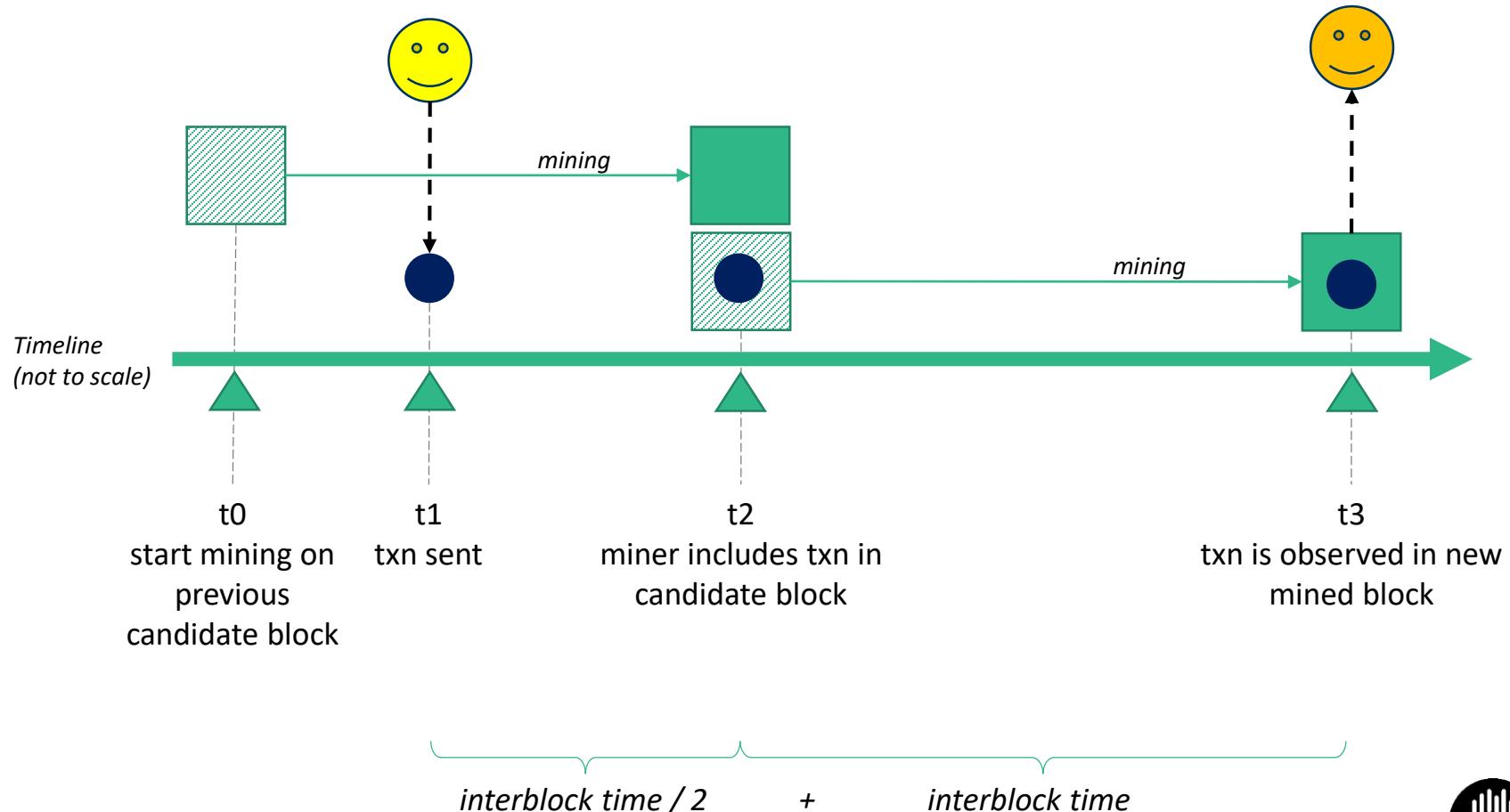
- For public blockchains
  - Cryptocurrency for Transaction Fees
    - Can affect probability of transaction inclusion (impacts transaction latency)
  - Mining difficulty for Proof of Work
  - Gas & Block Gas Limits (e.g. in Ethereum)
    - More of an Availability problem as discussed next week
  - Network capacity including speed of light for global transaction & block propagation
    - Impacts consensus for transaction inclusion latency
  - Transaction throughput capacity
    - Latency will be impacted when system is overloaded beyond capacity
- For private blockchains
  - Normal cloud/enterprise server & network resource considerations



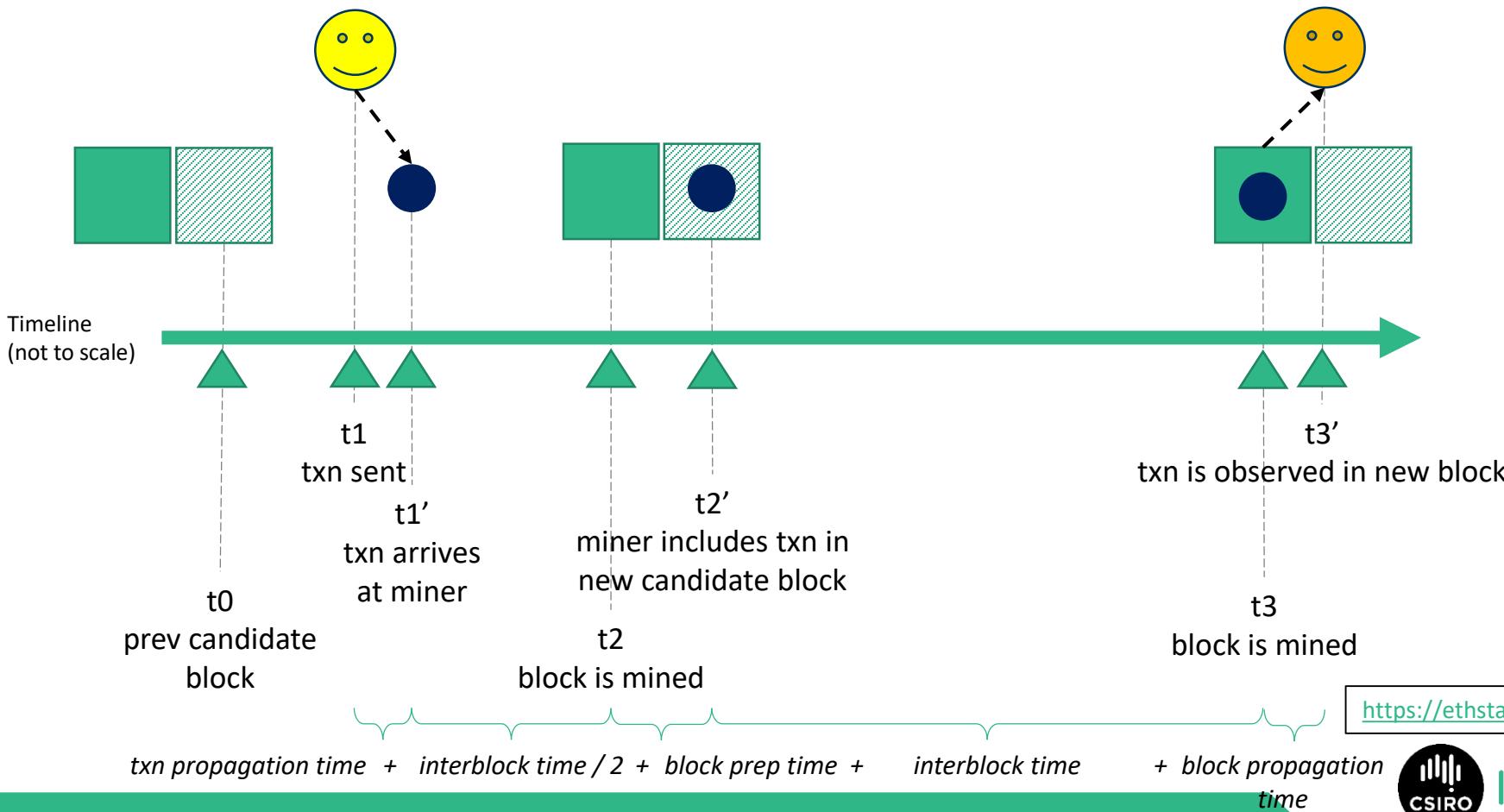
# Latency



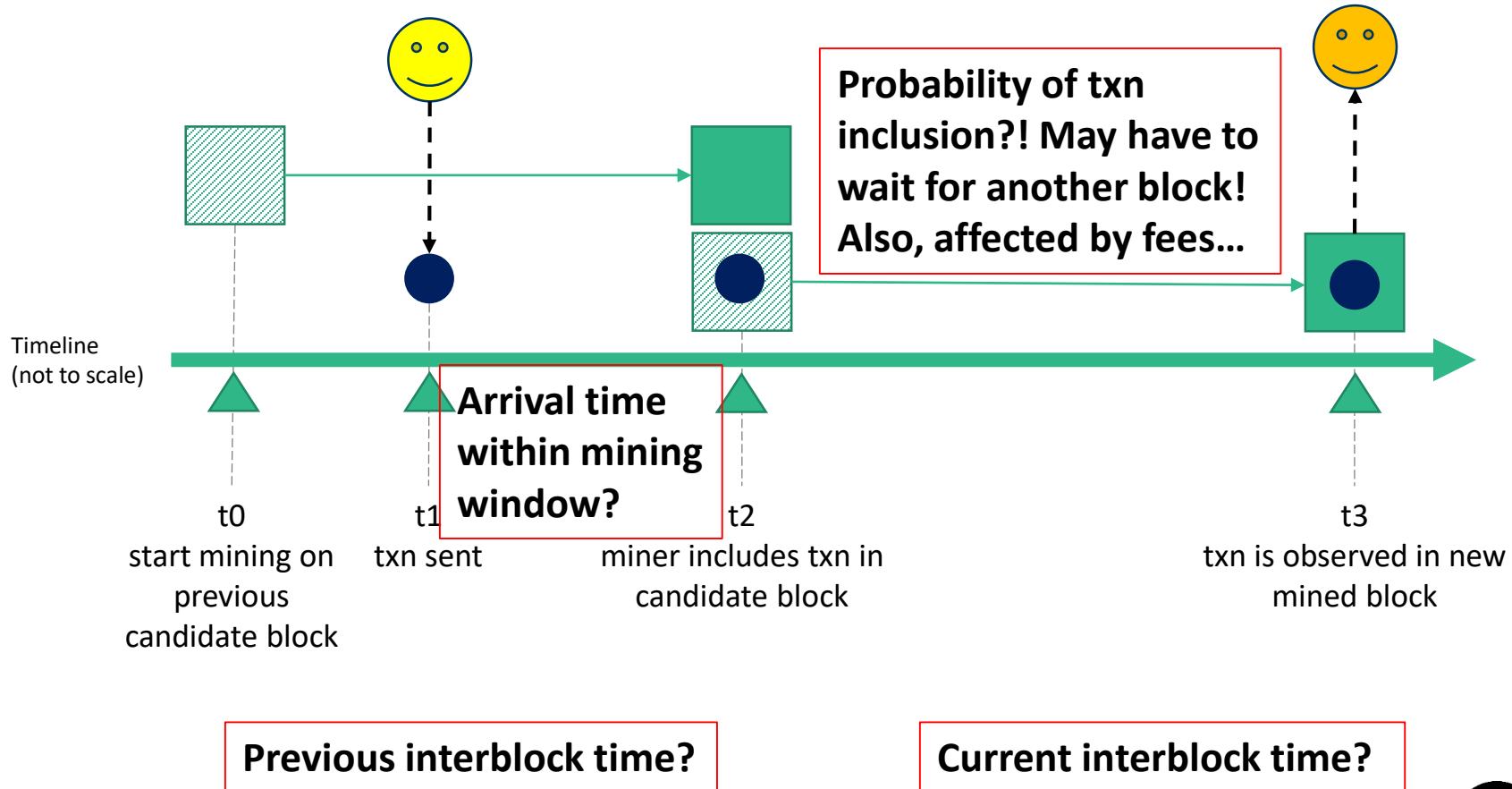
# Transaction Inclusion Time in Public Blockchain...



# ... Plus Extra Small Bits of Time



# ... Also, Major Sources of Variation in Latency



# Upshot is High Variation of Latency

- Not 1.5 x interblock time; in practice more like ~1.9 x inter-block time?
- But with a lot of variation! Some caused by txn fee, gas fee, etc
- You need to design your system to cope with this to meet requirements

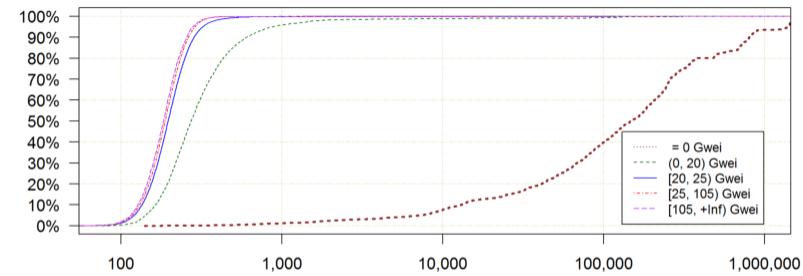
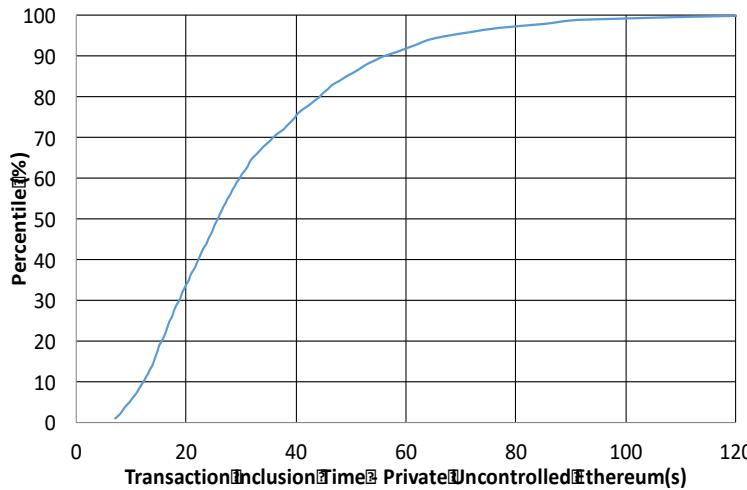
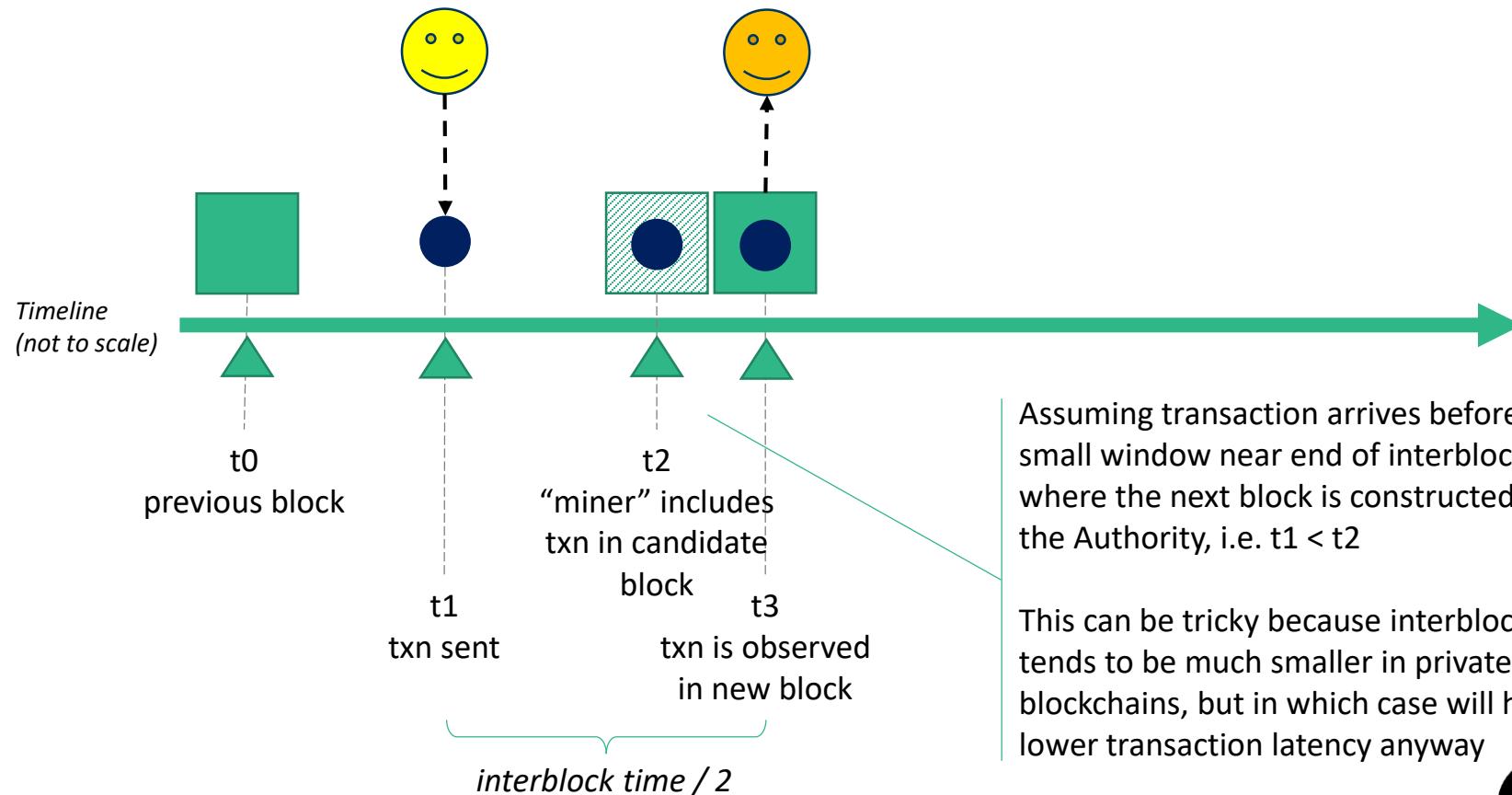


Figure 6: Commit delay (sec) for transactions based on gas price. Note logarithmic  $x$  axis.

N.B.: with 11 “confirmation blocks” adding a large “constant”-ish factor on the main cause of difference in latency variation



# An Aside: Transaction Inclusion with Proof of Authority in Private Blockchain



# In Public Blockchains, Also Wait for Confirmation Blocks!?

- Nakamoto consensus can create “uncle” blocks
  - Blockchains using Proof-of-Work, Proof-of-Stake, ...
  - Transaction you saw in a block turns out not to have been officially included in the blockchain
  - Only probabilistic, long-run, transaction inclusion
- Can increase confidence your transaction is really included, by waiting for subsequent “confirmation blocks”
  - For Bitcoin, often people say wait 6 blocks
  - For Ethereum, often people say wait 12 blocks
    - (Lower interblock time can increase likelihood of uncles)
- Waiting for confirmation blocks increases latency, and increases latency variability

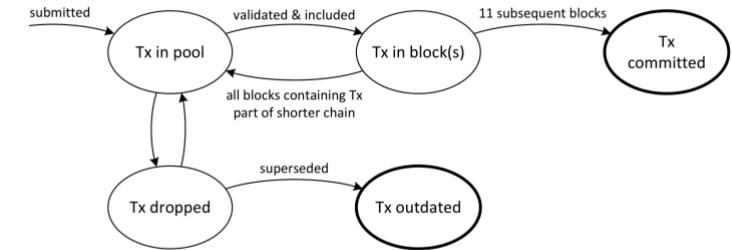


Figure 4: State machine for an individual transaction

# Confirmations: Inclusion Confidence vs Latency

- Should be risk-based: how much confidence? vs. how much latency?

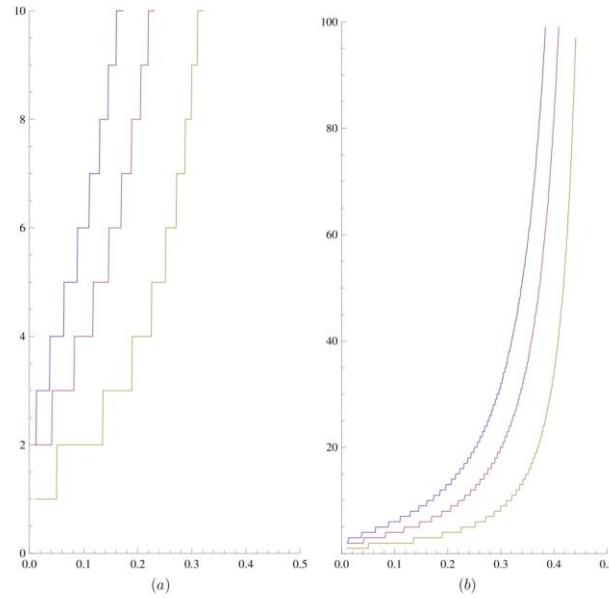


Figure 5: The number of confirmations required to keep the probability of success low, as a function of the attacker's hashrate, for various values of the target probability: 10% (yellow), 1% (purple) and 0.1% (blue). The graph is shown in two different vertical scales.



“Analysis of hashrate-based double spending”

<https://arxiv.org/abs/1402.2009>

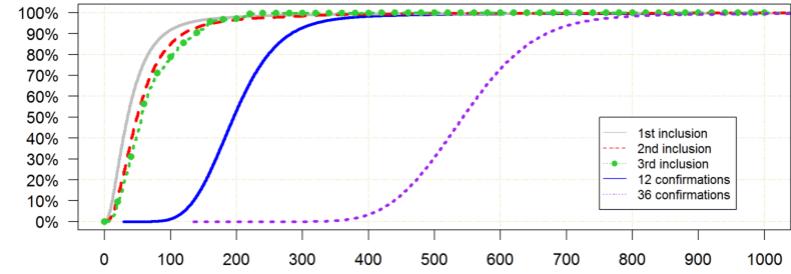


Figure 5: Time (sec) for first inclusion and commit (12 or 36 confirmations), as well as second and third inclusion of transactions that were previously not included in main chain.



“On Availability for Blockchain-Based Systems”

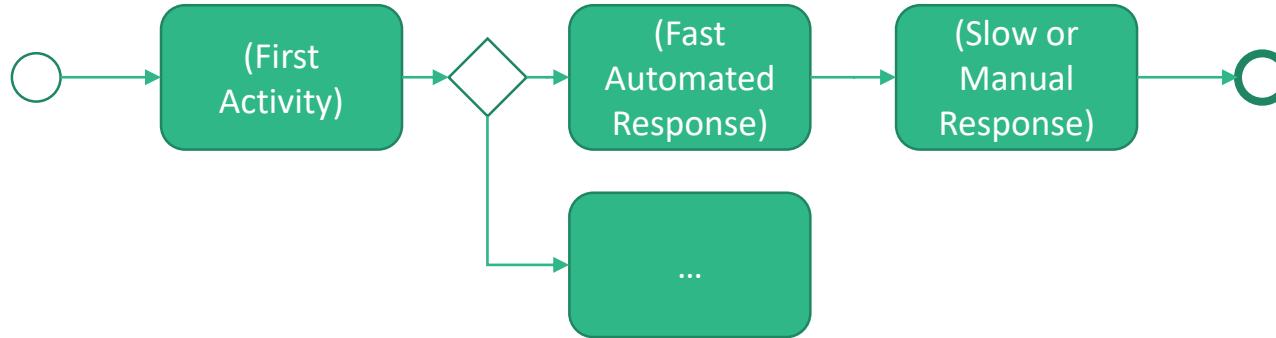
<https://research.csiro.au/data61/wp-content/uploads/sites/85/2016/08/OnAvailabilityForBlockchain-BasedSystems-SRDS2017-authors-copy.pdf>



# How Architects Can Influence Transaction Latency

- Offer a Transaction fee that meets/exceeds miners' expectations
  - Ethereum "fee" is gas price \* gas used
- Choose smallest number of confirmation blocks that is "safe" for your application
- Use another blockchain (private? or a public blockchain that works "better"?)
  - Configure blockchain to use other consensus algorithms (PBFT, PoA, ...)
    - Avoid Nakamoto consensus & possibility of uncles, so you can avoid confirmation blocks
  - Configure blockchain to target a small interblock time
- Work off-chain, e.g. using "state channel" design pattern, to be discussed in later lectures

# An Example: Process Execution on Public Blockchain

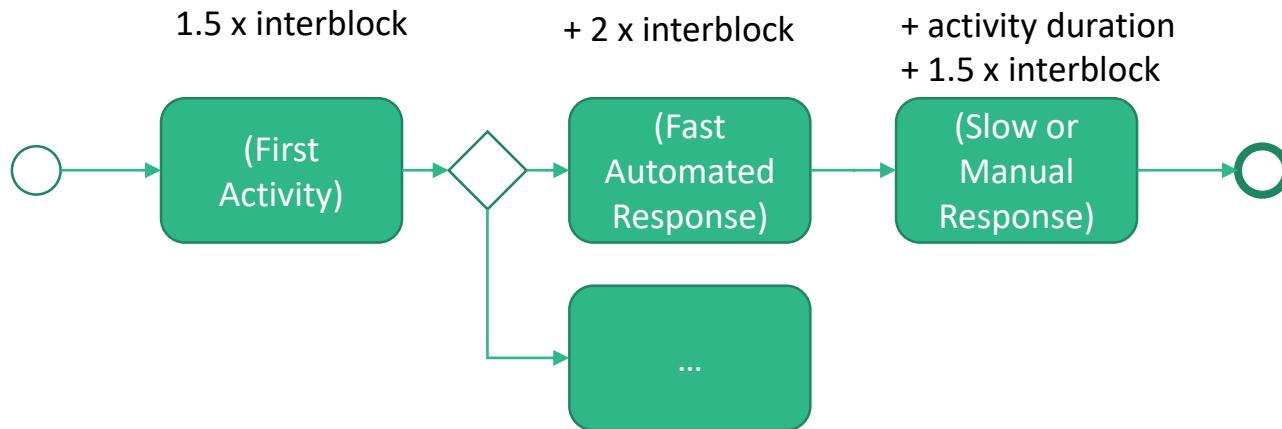


A process instance can be represented as a smart contract on a blockchain.

Each activity in the process can be a different function in the smart contract API.

An off-chain “trigger component” on a cloud or enterprise server watches a blockchain node and invokes the next activity in the process. (Immediately if activity is automated, otherwise e.g. in response to some manual activity.)

# Latency of Process Execution on Public Blockchain



Here, the design does not use confirmation blocks.

Quick & dirty!  
Assumes averages (median or mean)!

- Transactions arrive in the middle of an interblock time window
- Uniform average interblock time

The first activity will arrive in “the middle” of some interblock time window, so takes  $1.5 \times$  interblock times.

If activity response is very fast (in the same interblock time window), it will still be too late to include in the next block, so will take  $2 \times$  interblock times.

If the activity response is very slow (longer than the interblock time window), it will arrive in “the middle” of some future interblock time window

# Simulation for Latency Estimation

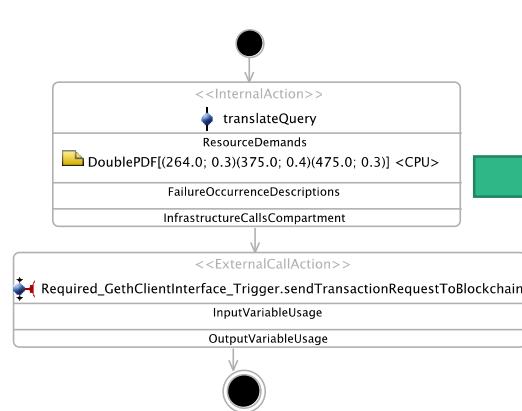
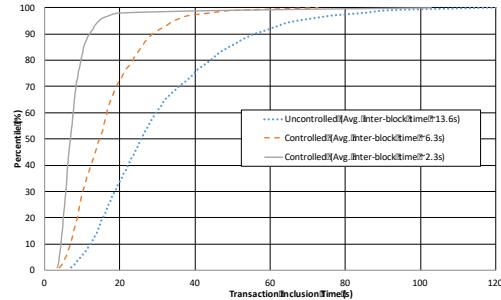
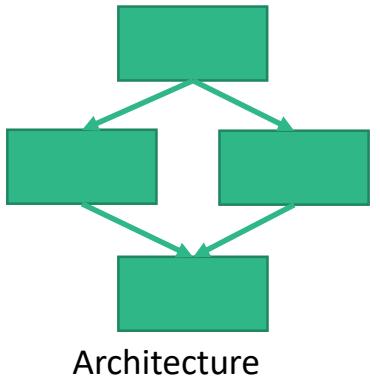
- e.g. using Palladio
  - Open source, Component-Based
    - <https://www.palladio-simulator.com/home/>
  - Widely used for performance modelling and simulating
  - UML-like notation
- Need to Benchmark Transaction Inclusion on Blockchain
  - Treat the blockchain as a black box – txns in/txns out
  - Measure time between submitting a transaction and receiving verification by a given number of confirmation blocks
    - Not just averages; measure the distribution of times
  - Also benchmark trigger implementation that invokes txns

“Predicting latency of blockchain-based systems using architectural modelling and simulation”

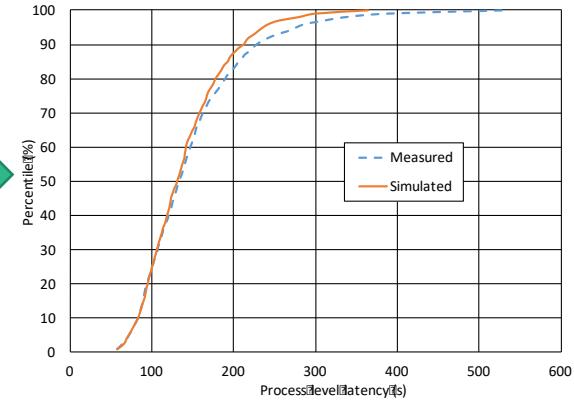
[https://research.csiro.au/data61/wp-content/uploads/sites/85/2016/08/2017-ICSA-Blockchain-Latency-Sim-authors\\_copy.pdf](https://research.csiro.au/data61/wp-content/uploads/sites/85/2016/08/2017-ICSA-Blockchain-Latency-Sim-authors_copy.pdf)



# Simulation for Predicting Latency

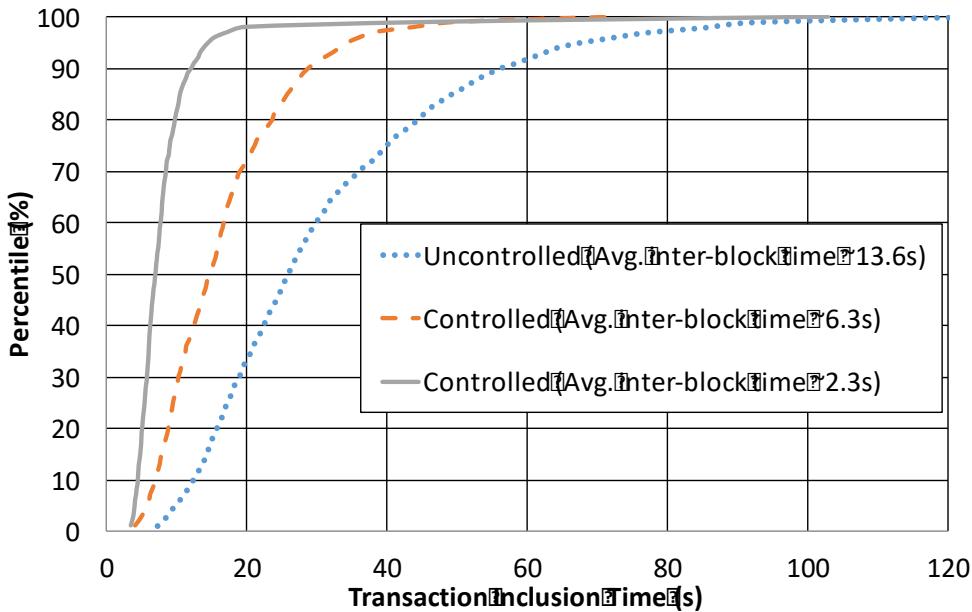


Model Construction

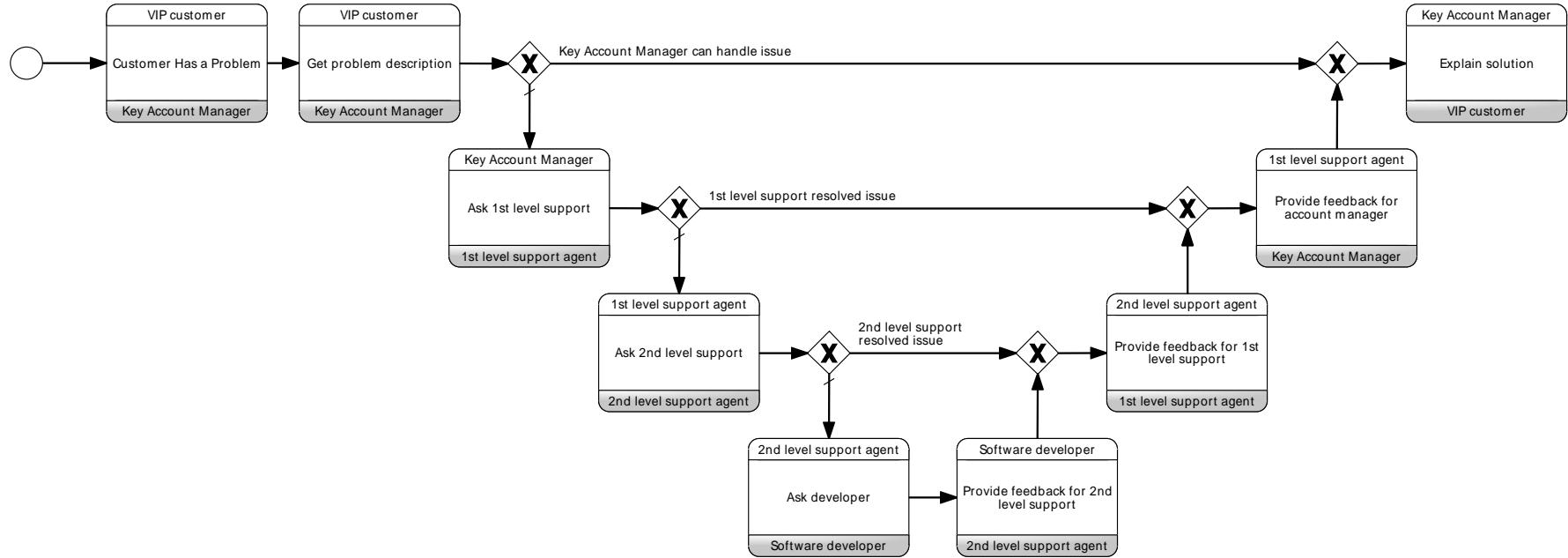


# Benchmarking Transaction Commit Time

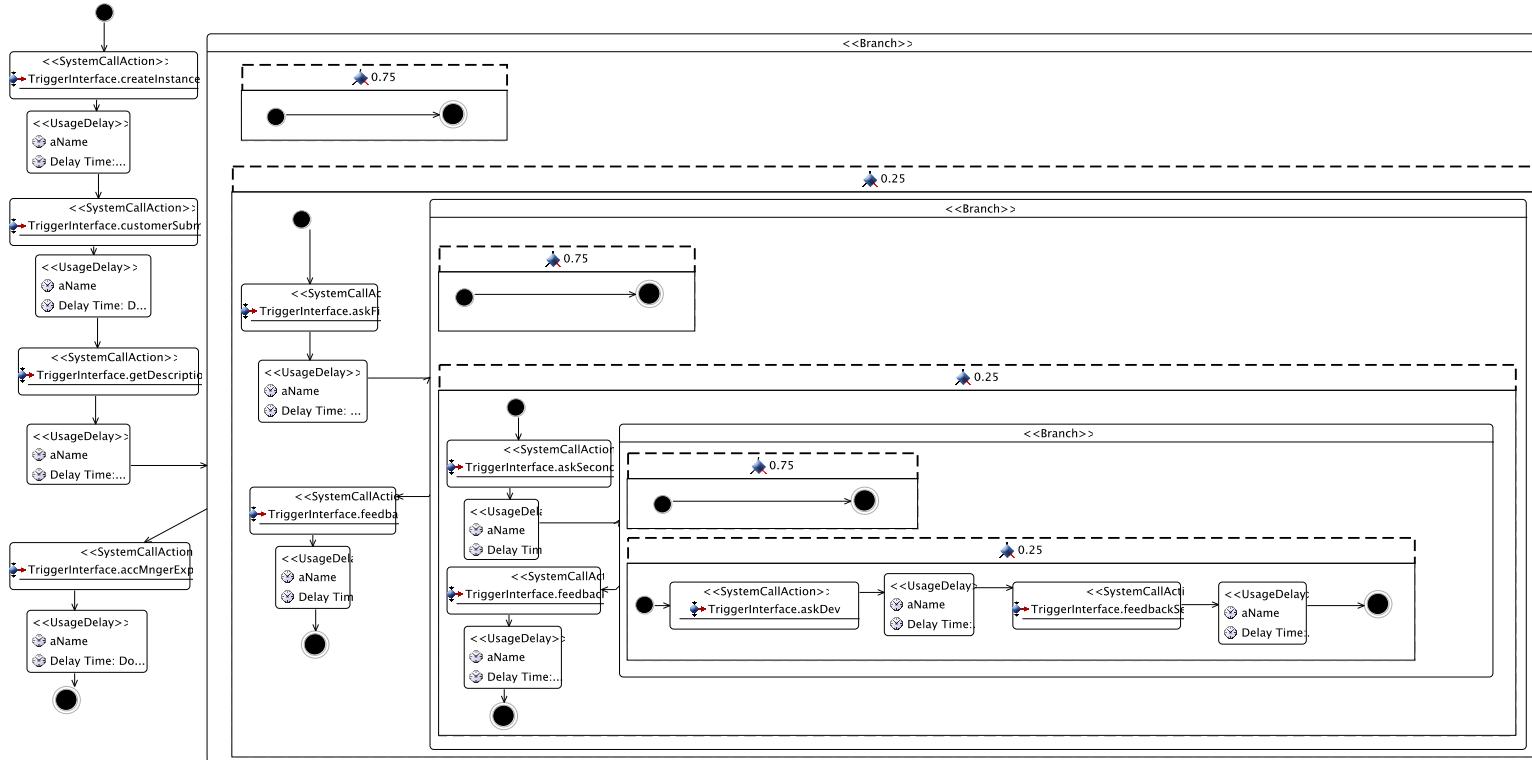
- Transaction Inclusion Time: Time Taken from Submitting transaction until transaction get included in blockchain



# Example Incident Management Business Process



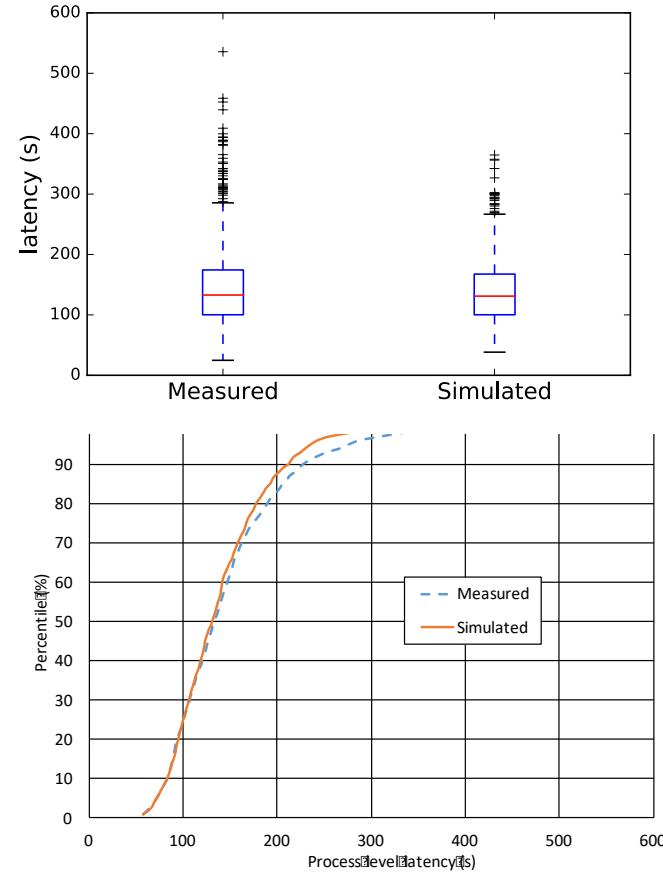
# Example Usage Model



Usage diagram

# Example Results & Checking vs. Real System

	Measured	Simulated	Relative error
Median latency	136s	134s	1.6%
Standard error of median (SEM)	1.27	1.07	-
95 <sup>th</sup> Percentile	274s	248s	9.4%
99 <sup>th</sup> Percentile	373s	329	11.5%



# Throughput



# Throughput

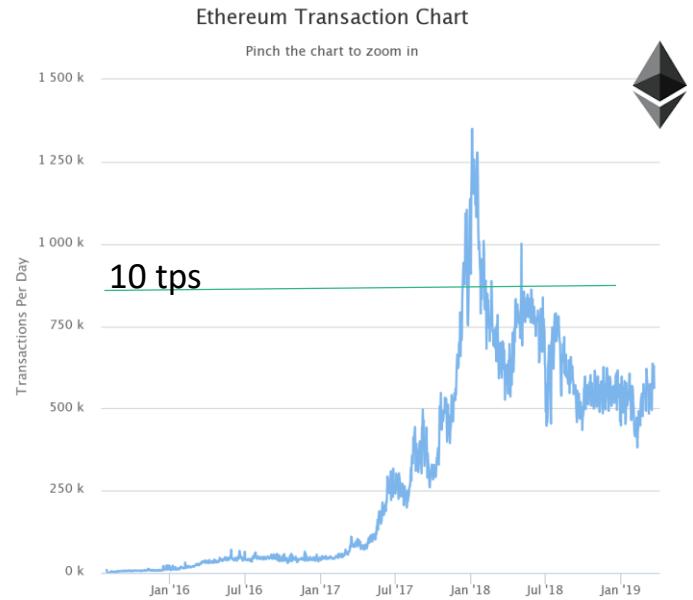
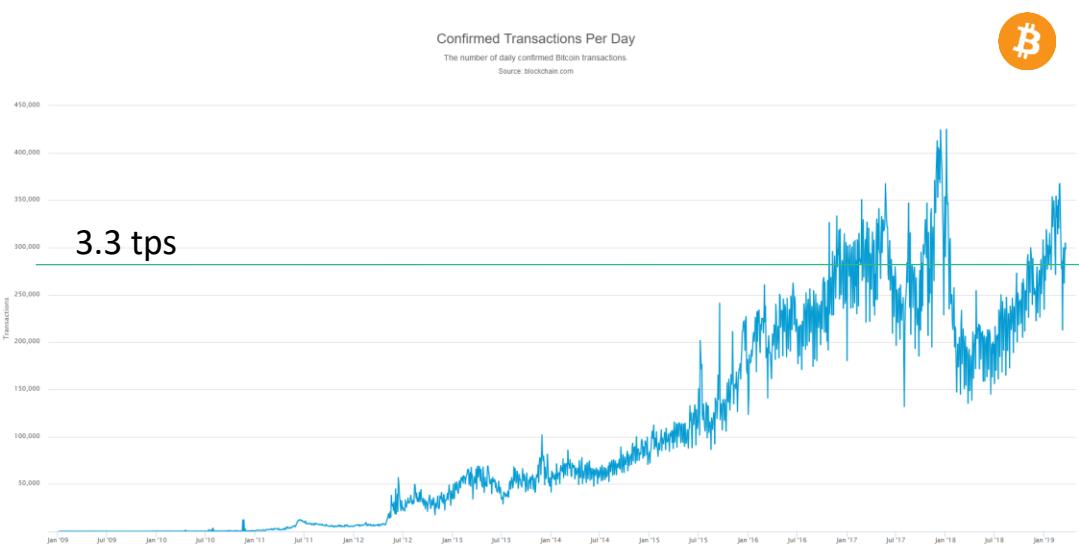
- How many transactions can you process in a unit of time?
- e.g. Visa card processing, 2000 tps daily average, peak capacity ~60000 tps
  - <https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html>
- Bitcoin has poor throughput capacity (3 to 7 tps)
- Ethereum has poor throughput capacity (7 to 25 tps)
  - <https://medium.com/coinmonks/understanding-cryptocurrency-transaction-speeds-f9731fd93cb3>
- You as an individual user are unlikely to create system overload!
  - But, you need to understand trends in ecosystem utilisation and bottlenecks
    - e.g. At one point in time, about a third of transactions on Ethereum were for cryptokitties



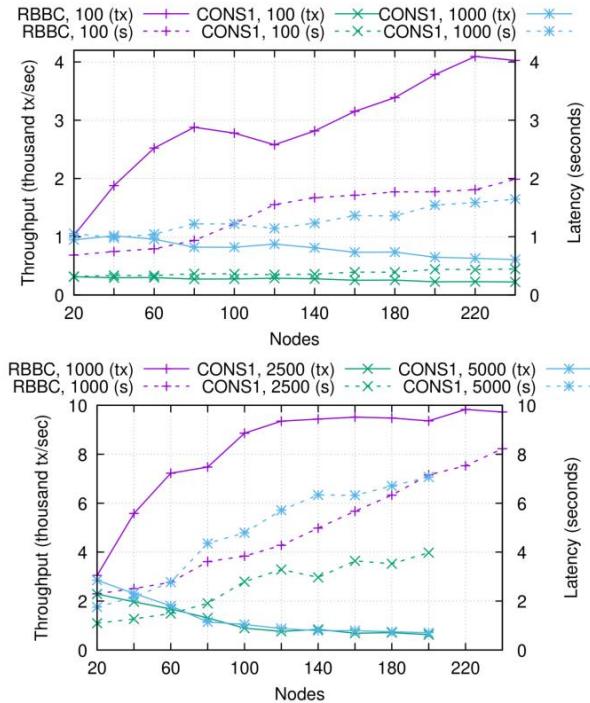
# Throughput Limits

- Transactions per block \* blocks per second = Transactions per second
- Bitcoin limited by block size, and average transaction size
  - $1000000 \text{ Bytes} / 495 \text{ Bytes} / 600 \text{ seconds} = 3.37 \text{ tps}$ 
    - (growth in use of segwit and other txn size optimisation can double? throughput)
- Ethereum limited by block gas limits, and average transaction gas
  - $8000000 \text{ gas block limit} / 76000 \text{ gas} / 15 \text{ seconds} = 7 \text{ tps}$
- Limits are there mainly to control block propagation times
  - To control likelihood of uncles, DDoS, and control centralisation of mining

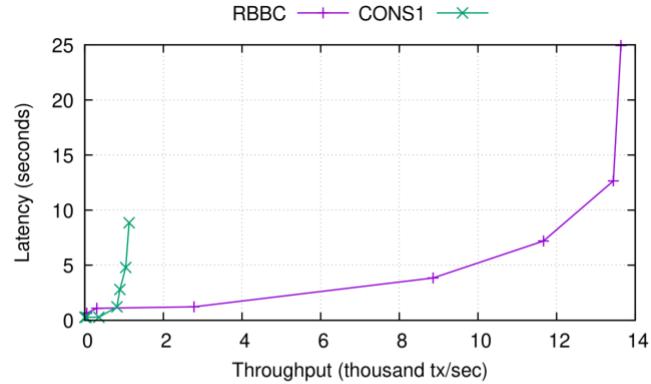
# Growth in Throughput



# Red Belly Blockchain Throughput



**Figure 3.** The performance of CONS1 and RBBC with  $t + 1$  proposer nodes; the number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency



**Figure 4.** Comparing throughput and latency of CONS1 and RBBC with  $t + 1$  proposer nodes on 100 geodistributed nodes; each point represents the number of transactions in the proposals, either 10, 100, 1000, 2500, 5000 or 10000

“Evaluating the Red Belly Blockchain”  
<http://arxiv.org/abs/1812.11747>

# DAGs, Sharding, etc

- Blockchains enforce a strictly ordered single list of transactions
  - Critical for integrity of transfer of digital assets (no “double spend”)
  - But creates a throughput bottleneck
- DAGs (“tangles”), e.g., IOTA, ...
  - Not a list, but a directed acyclic graph
  - Multiple heads (“tips”), each processed mostly concurrently, repeatedly diverging/merging
  - Hard to maintain data integrity across the heads – transactions can conflict!
    - Can, e.g. resolve a bit like like Nakamoto consensus – tentatively included until merged
- Sharding, e.g. future Ethereum?
  - Split chain into smaller chunks (horizontal partitioning), each processed mostly concurrently
    - Might increase Ethereum throughput by 1000x
  - Hard to maintain data integrity across the shards
    - Shards interact with a protocol only when necessary, to maintain global integrity
- Networks or hierarchies of interrelated blockchains
  - e.g. geographically-sharded IoT blockchain networks

# How Application Architects Can Influence Transaction Throughput

- Use another blockchain (private? or a public blockchain that works “better”?)
  - Configure blockchain to use other consensus algorithms (PBFT, PoA, ...)
  - Sharding, DAGs
  - Configure blockchain to use a larger block size, or smaller interblock time
  - If you don’t need exchange of digital property, consider networks of blockchains, DAGs, ...
- If you have to use a specific public blockchain, then focus on how you use the blockchain, and focus on other areas of the application architecture
  - Reduce the data you put on-chain (sampling, digests, hashes, ...)
  - Do more work off-chain, e.g. using “state channel” design pattern, to be discussed in later lectures

# Summary



# Summary: General

- $(\text{Load} \times \text{Resources}) \rightarrow (\text{Performance} \times \text{Resource Utilisation})$ 
  - Measurement is critical to understand all this in practice
- Kinds of Performance: Latency vs Throughput (vs Timeliness)
- Blockchains have high latency and high variability of latency
  - Variability is significantly affected by choice of consensus mechanism
  - Number of confirmation blocks are a risk-based decision
    - Can avoid them if you don't use Nakamoto consensus/probabilistic commits
- Blockchains have poor throughput
  - Affected by ledger structure and consensus mechanism
  - Do you need exchange of digital assets?
    - If you don't need a global integrity property, you can avoid major bottlenecks



# Summary: Predicting Latency

- For a single transaction on public blockchain  
 $\text{latency} = 1.5 * \text{interblock time}$ 
  - Assuming transaction is included ASAP
  - Ignoring transaction/block propagation times, and block preparation time
  - On average (ignoring variation in interblock time, arrival time)
- For sequence of  $n > 1$  transactions on public blockchain  
 $\text{latency} = 1.5 * \text{interblock time} + (n-1)*2 \text{ interblock time}$ 
  - Assuming next transaction is injected immediately into the interblock window
- Can use simulation for more precise prediction of application-level latency
  - Can explore latency variation, interactions between components, and bottlenecks
- Or, use measurement and test of real(istic) system under real(istic) load!





# THANK YOU

[www.data61.csiro.au](http://www.data61.csiro.au)



# NFPs 3: Dependability and Security

Mark Staples

Email: [mark.staples@data61.csiro.au](mailto:mark.staples@data61.csiro.au)

[www.data61.csiro.au](http://www.data61.csiro.au)

# Outline

- Dependability and Security
- Trust and the Need for Trustworthy Blockchain Applications
- Functionality
- Security
- Reliability
- Summary
- Assignment 2

# Dependability and Security

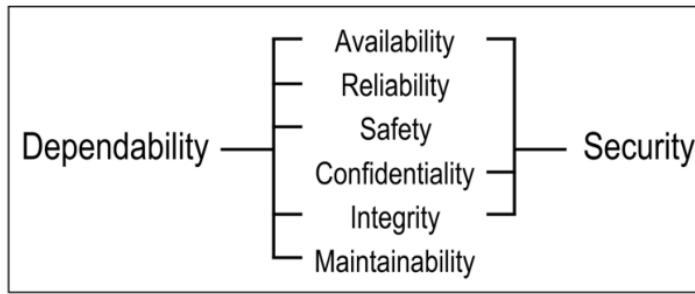


Fig. 1. Dependability and security attributes.

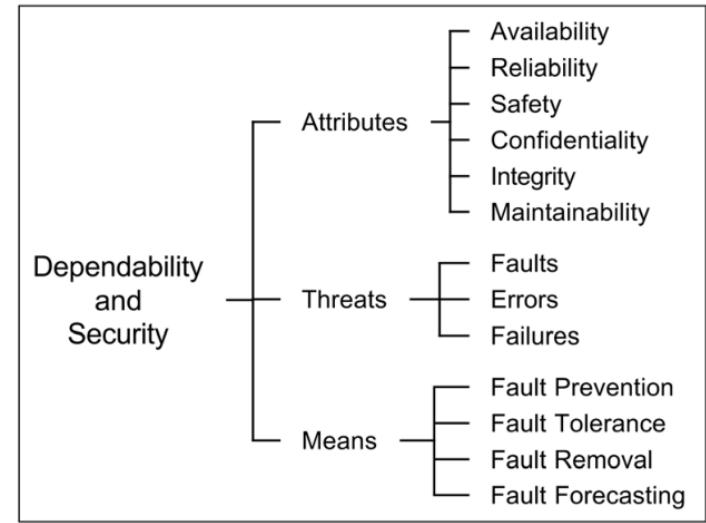


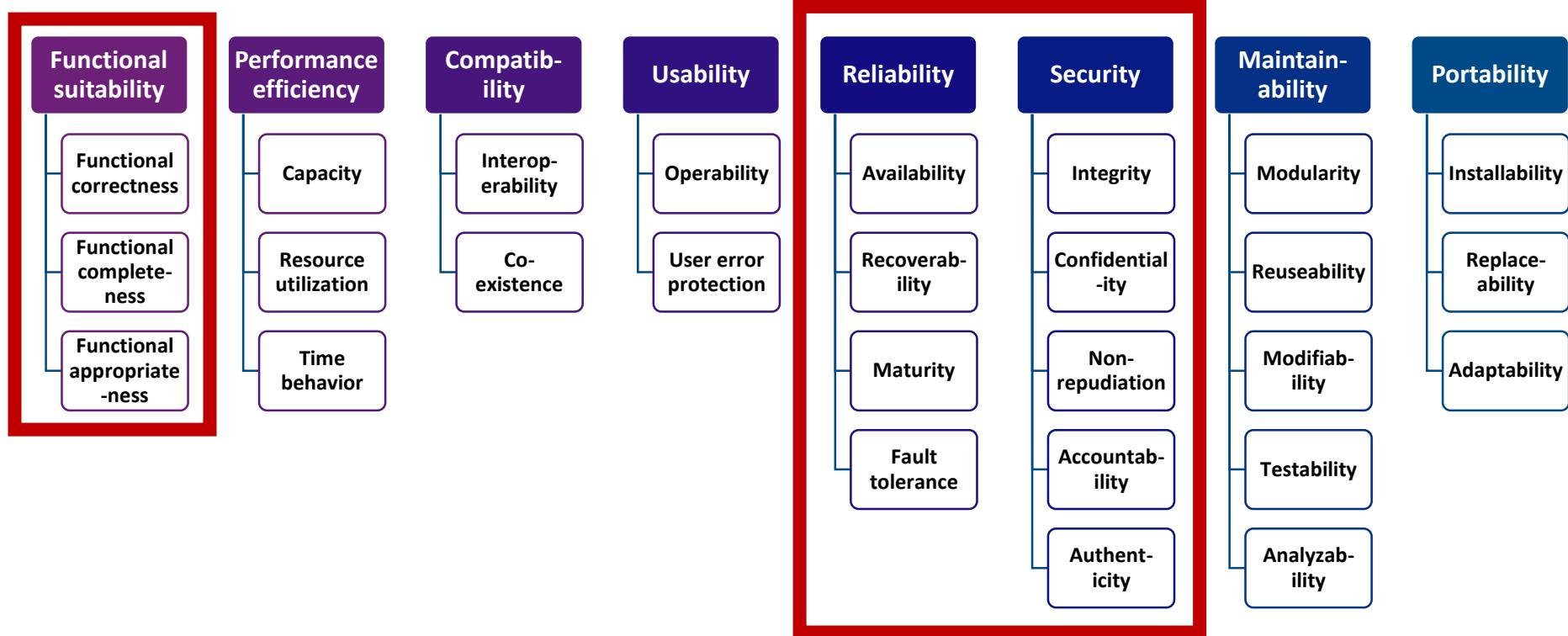
Fig. 2. The dependability and security tree.

"Basic concepts and taxonomy of dependable and secure computing"  
[https://www.nasa.gov/pdf/636745main\\_day\\_3-algirdas\\_avizienis.pdf](https://www.nasa.gov/pdf/636745main_day_3-algirdas_avizienis.pdf)

# Faults, Errors, Failures

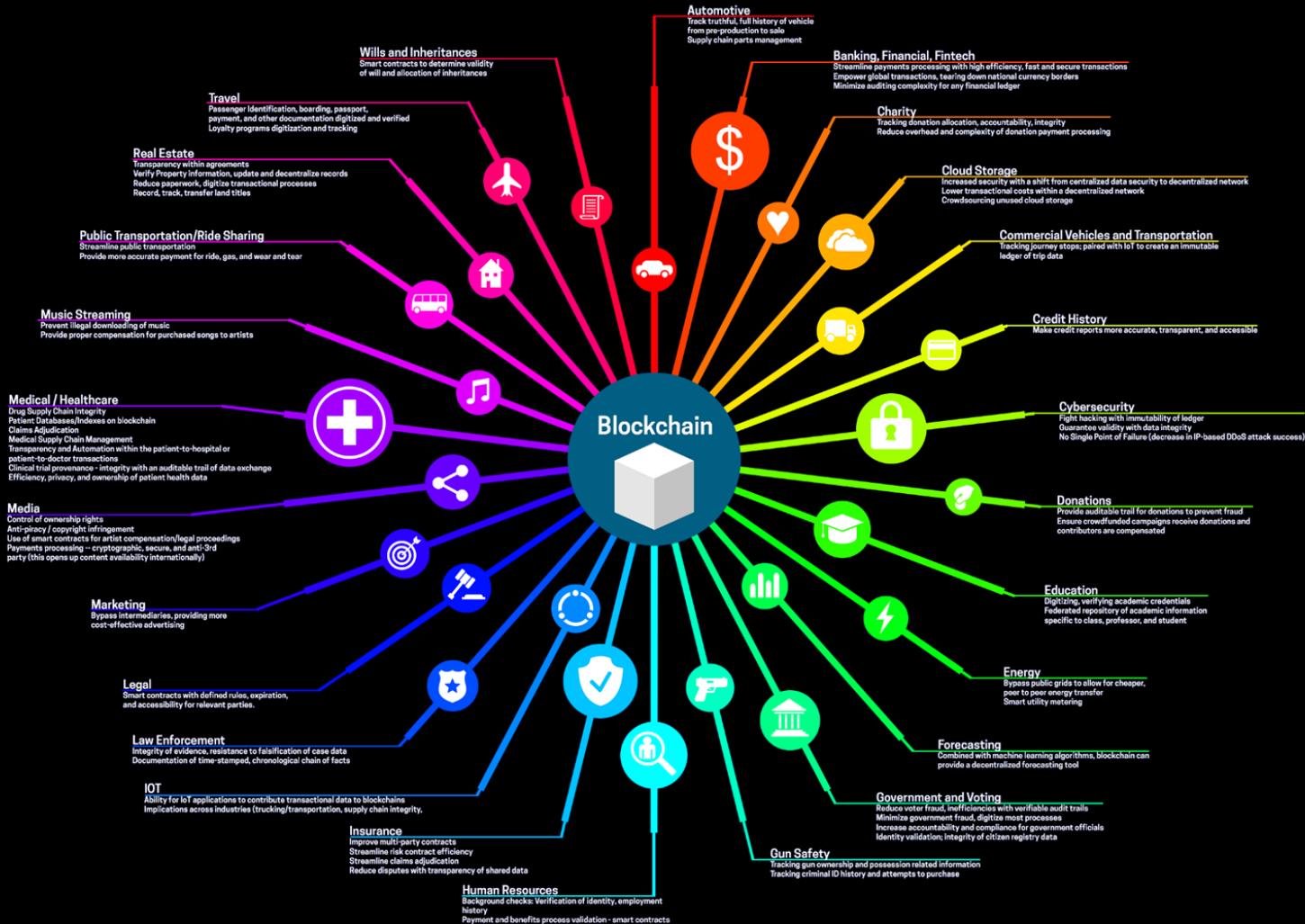
- **Correct service** is delivered when the service implements the system function.
- A **(service) failure**, is an event that occurs when the delivered service deviates from correct service.
- An **error** is a system state that could lead to a service failure.
- A **fault** is the adjudged or hypothesized cause of an error.
- **Fault prevention** means to prevent the occurrence or introduction of faults.
- **Fault tolerance** means to avoid service failures in the presence of faults.
- **Fault removal** means to reduce the number and severity of faults.
- **Fault forecasting** means to estimate the present number, the future incidence, and the likely consequences of faults.

# ISO/IEC 25010:2011 Quality Model



# Trust, and the Need for Trustworthy Blockchain Applications





# We (Will) Rely on Blockchain-Based Systems



- Smart contract bugs: DAO (\$60M); Parity (\$280M)
- Cryptographic key loss
  - Hacking: Mt Gox (\$450M), Bitfinex (\$72M), total Jan-Sep 2018 (\$927M); UK rubbish tip (\$146M); guns e.g. NYC (\$1.8M)



- Huge future economic value (the main point!)
  - e.g. supply chain, asset registries, settlement, energy, ...



- Security-critical and Safety-critical use cases
  - e.g. e-health records, food safety, pharma supply chain, IoT management, cybersecurity, law enforcement, ...

# About “Trust”

- Dependable Software Systems says...
  - Trusted System
    - A system you have chosen to rely on to fulfil a goal
    - When it fails, you suffer harm or loss
  - Trustworthy System
    - A system where you have evidence it will not fail
- “Trustless” Blockchains? **MYTH!**
  - Can shift trust within a wider system
  - Blockchains themselves become trusted components
  - Other components are also trusted
    - e.g. Oracles, Key Management Systems, ...



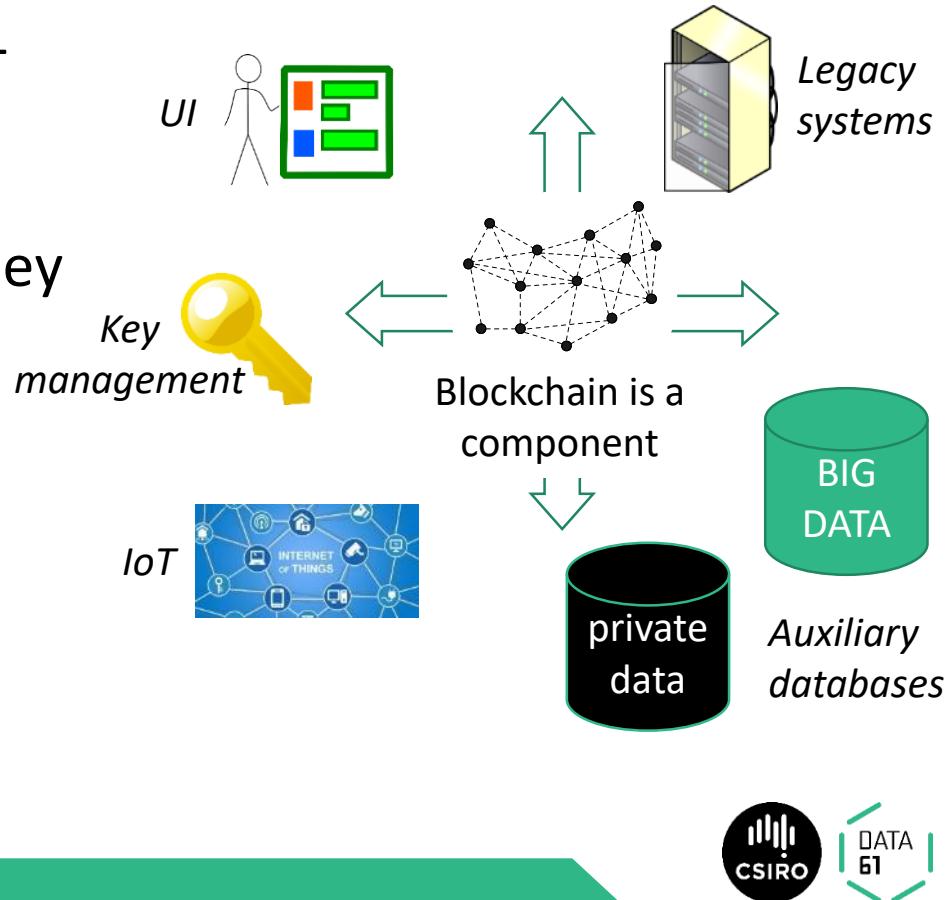
# Engineering: Assurance is Key

- Distinguishes engineering from e.g. architecture, building
- Assurance is...
  - A claim that an **artefact** meets key **requirements**
  - Made & evaluated by recognised experts
  - On the basis of a rationale
- Assurance rationale must be backed by empirical **engineering theories** & evidence



# Trustworthy Blockchain-Based Systems?

- How do we design blockchain-based systems that work?
- What is good evidence that they will work?
  - Functional correctness
  - Non-Functional properties
- How do we get acceptance?
  - Individual, Enterprise, Regulatory, Societal



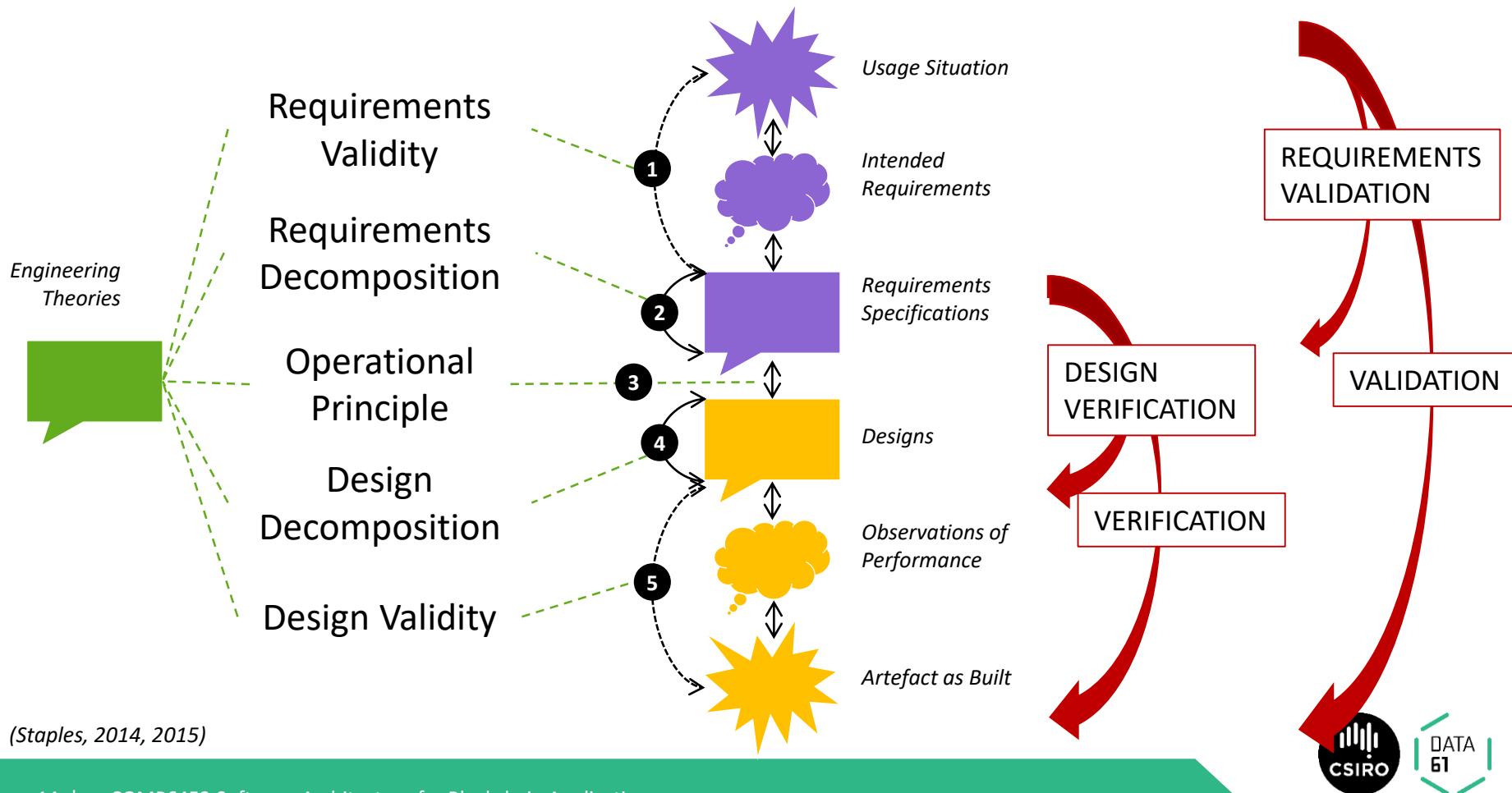
# Functionality



# Functional Suitability from ISO/IEC 25010:2011

- Functional Correctness
  - degree to which a product or system provides the correct results with the needed degree of precision
- Functional Completeness
  - degree to which the set of functions covers all the specified tasks and user objectives
- Functional Appropriateness
  - degree to which the functions facilitate the accomplishment of specified tasks and objectives

# Verification and Validation



# What Function? Is it true that “Code is Law?

- Lawrence Lessig famous quote “Code is Law”

<https://harvardmagazine.com/2000/01/code-is-law-html>

- Some people think this means “The power of code can take over from the law”
  - Lessig meant: “Code requires choices about values, has social implications, and should be governed like the law”

- “The DAO” failure on Ethereum in 2016

- An instance of the “Distributed Autonomous Organisation” concept
  - “The DAO” was for decentralised control of investment funds, raised > \$150M
  - But the code was used in an unexpected way to siphon funds away (~ \$50M?)
  - Led to a hard fork in Ethereum vs. Ethereum Classic to reverse the effects of that

## The DAO terms

# Explanation of Terms and Disclaimer

The terms of The DAO Creation are set forth in the smart contract code existing on the Ethereum blockchain at 0xbb9bc244d798123fde783fcc1c72d3bb8c189413. Nothing in this explanation of terms or in any other document or communication may modify or add any additional obligations or guarantees beyond those set forth in The DAO's code. Any and all explanatory terms or descriptions are merely offered for educational purposes and do not supercede or modify the express terms of The DAO's code set forth on the blockchain; to the extent you believe there to be any conflict or discrepancy between the descriptions offered here and the functionality of The DAO's code at 0xbb9bc244d798123fde783fcc1c72d3bb8c189413, The DAO's code controls and sets forth all terms of The DAO Creation.

# What Should Your Specification Be?

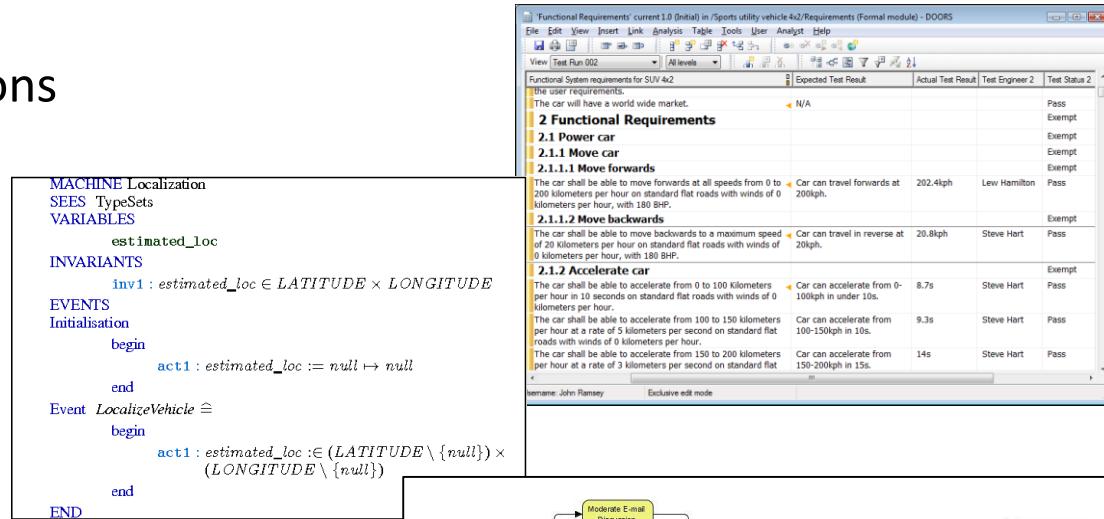
- There was no problem with Ethereum virtual machine; the code functioned exactly as it was written
- Question: Was this a “bug” in the DAO code?
- Lessons from the DAO failure
  - Lesson: Code is not the law
  - Lesson: Code is not a good way of specifying smart contracts
- Open question: how should we specify smart contracts?

# Specifications are Models of Requirements

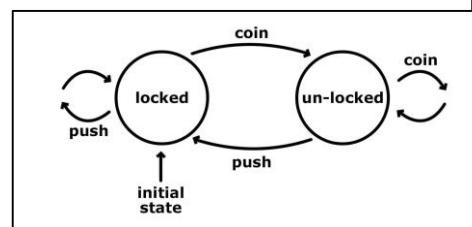
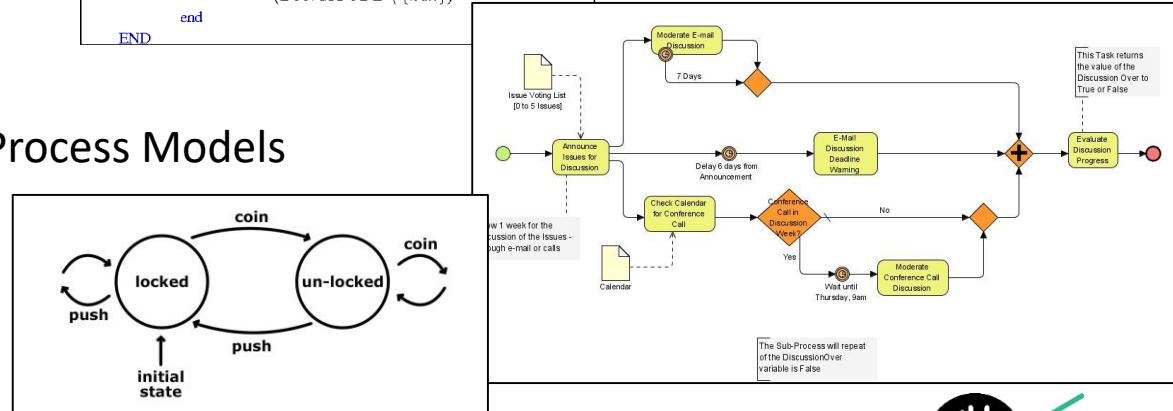
- Natural Language Specifications
  - Lists of Requirements
  - Use Cases and Scenarios

- Formal Specifications
  - Z, Object-Z, B, Event-B, HOL

- Models
  - State Machines, Business Process Models



```
MACHINE Localization
SEES TypeSets
VARIABLES
    estimated_loc
INVARIANTS
    inv1 : estimated_loc ∈ LATITUDE × LONGITUDE
EVENTS
Initialisation
begin
    act1 : estimated_loc := null ↪ null
end
Event LocalizeVehicle ≡
begin
    act1 : estimated_loc :∈ (LATITUDE \ {null}) ×
        (LONGITUDE \ {null})
end
END
```

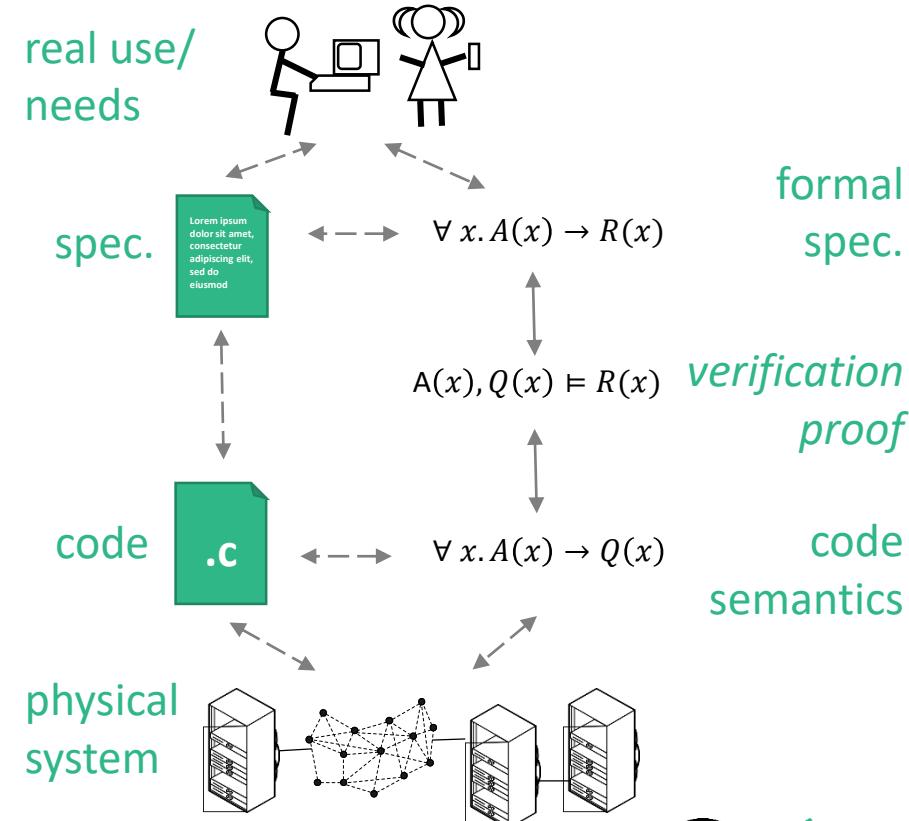


# Formal Specification & Verification

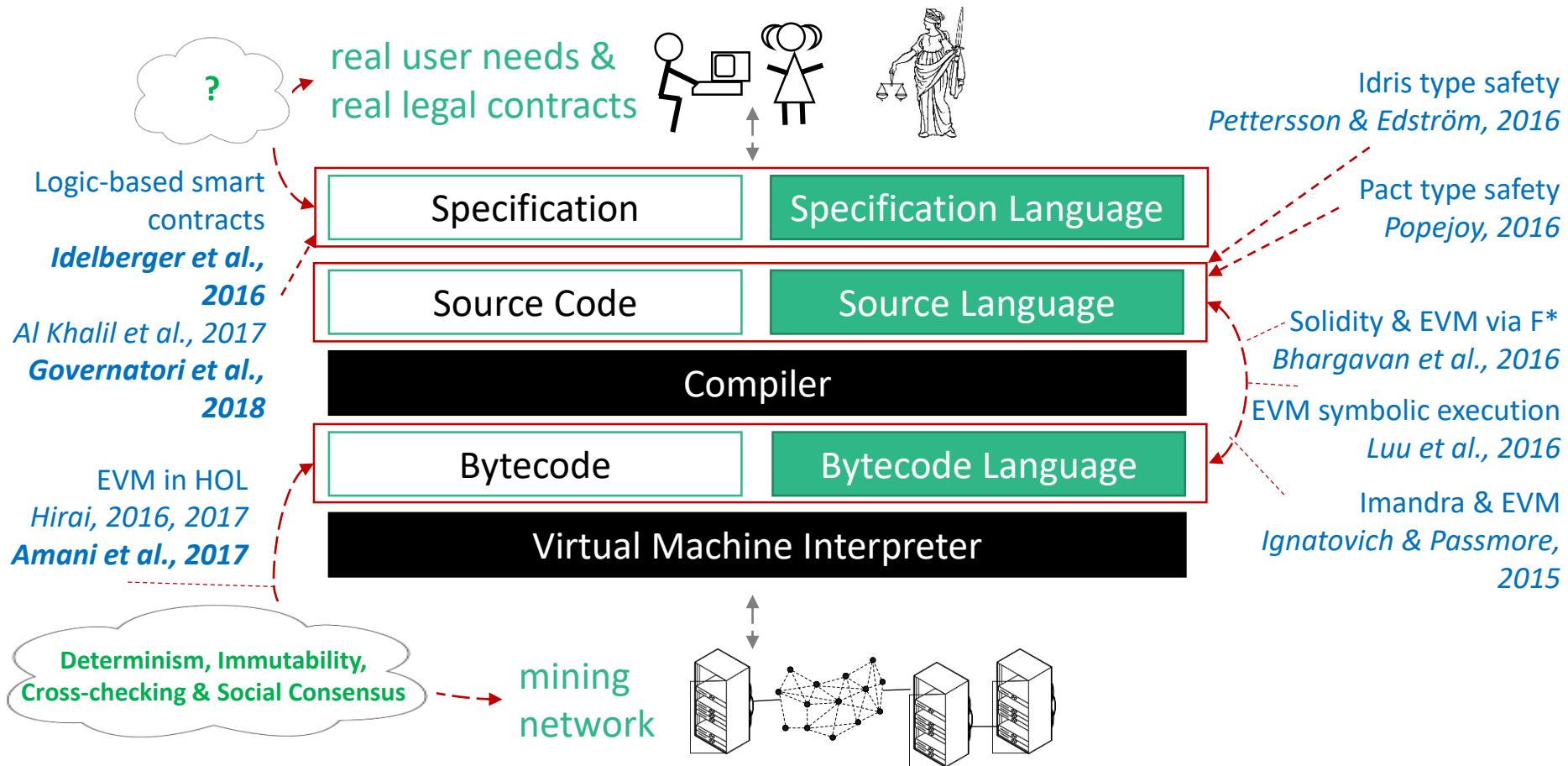
*Program testing can be used to show the presence of bugs, but never to show their absence!*

- Dijkstra, '60s/'70s

- Formal methods is logical reasoning about all possible behaviours of software
- Is the strongest kind of evidence that software is correct



# Formalising Smart Legal Contracts



# Security



# ISO/IEC 25010:2011 Security Characteristics

- Integrity
    - degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data
  - Confidentiality
    - degree to which a product or system ensures that data are accessible only to those authorized to have access
  - Non-repudiation
    - degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later
  - Accountability
    - degree to which the actions of an entity can be traced uniquely to the entity
  - Authenticity
    - degree to which the identity of a subject or resource can be proved to be the one claimed
- Only good writes & deletes
- Only good reads
- Undeniable
- You did it!
- Not fake

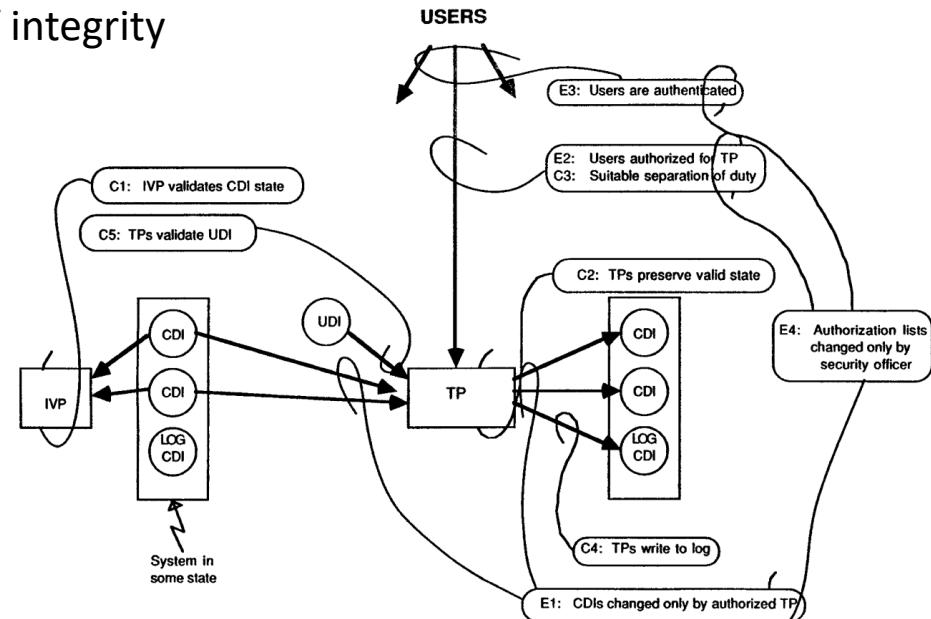
(Privacy is controlling access to yourself; and not just information)

(ISO/IEC 5010:2011 treats Availability as a “Reliability” characteristic)



# Integrity

- ISO/IEC 25010:2011: “degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data”
- Clark-Wilson policy model is classic model of integrity
  - Based on double-entry bookkeeping
- Start with a well-formed initial state
- “Transformation Procedures” preserve state well-formedness
  - “Integrity Validation Procedures” check conditions for “Constrained Data Items”
- Only authenticated & authorised users can make changes
  - Also ensure appropriate “separation of duty”
- Audit logs, controls on admin changes



“A Comparison of Commercial and Military Computer Security Policies”

<https://groups.csail.mit.edu/ana/Publications/PubPDFs/A%20Comparison%20of%20Commercial%20and%20Military%20Computer%20Security%20Policies.pdf>

# Integrity for Blockchain Platforms

- Clark-Wilson perspective on blockchain integrity
  - Blockchain ledger is the system state and the audit log
    - Blocks and their transactions are the “Constrained Data Items”
  - Initial state: genesis block is well-formed & cross-checked by all miners
  - Mining and cross-checking other miners’ blocks is the “Transformation Procedure”
  - Miners’ block validation checks are the “Integrity Validation Procedure”
  - Authorisation is checked using signatures from public-key cryptography
  - No authentication on public blockchains! Often important on private blockchains
  - No separation of duty enforced?
  - “Admin changes” are by the mining collective, e.g. hard forks
- Example integrity conditions
  - Were transactions against an account address signed by corresponding private key?
  - Does the sending address/account have enough cryptocurrency?
    - Some platforms support other crypto-assets with different integrity conditions
  - Did a miner give themselves the right mining reward?
  - Did the execution recorded for a smart contract give the same result I get?

# “Blockchain anomaly”

- Blockchain transactions are in a strict order, and are immutable... or are they?
- Nakamoto consensus can give us alternative/replacement histories
  1. Issue a transaction
  2. Wait to you see it
  3. Issue second transaction that depends on the first one
  4. Possible anomaly!

Both need to be independently “valid” from the perspective of a blockchain, but invalid to the application when reordered

Solutions: wait for more confirmation blocks, try to rely on Ethereum “transaction nonce”, or use a smart contract to enforce/check the ordering

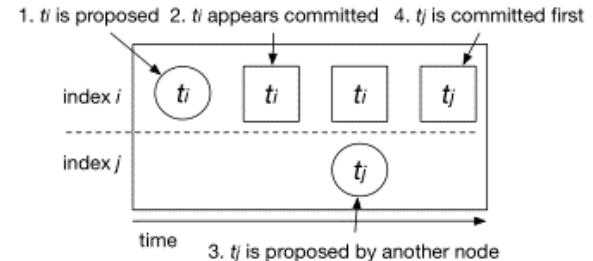


Figure 3: The Blockchain anomaly: a first client issues  $t_i$  that gets successfully mined and committed then a second client issues  $t_j$ , with  $t_j$  being conditional to the commit of  $t_i$  (note that  $j \geq i + k$  for  $t_i$  to be committed before  $t_j$  gets issued), but the transaction  $t_j$  gets finally reorganized and successfully committed before  $t_i$ , hence violating the dependency between  $t_i$  and  $t_j$

“The Blockchain Anomaly”  
<https://arxiv.org/pdf/1605.05438>

# Integrity for Component Services in Blockchain Applications

- Integrity of Data Storage and Communication
  - On-chain data in transactions: “enforce” using off-chain code & conventions
  - On-chain data in smart contract variables: enforce using smart contract code
  - Off-chain data: enforce by cross-checking data hash with on-chain hash values & also check authorisation off-chain using the signatures for those transactions
  - Off-chain data: enforce by checking data using rules stores on-chain
- Integrity of Computation and Asset Management
  - On-chain: smart contracts themselves being correct! Or to cross-check others
  - Off-chain: smart contracts cross-check off-chain computation (e.g. “state channel” pattern)
- Smart contracts can encode & enforce application-level integrity checks and transformations on-chain – code needs to be correct!
  - e.g. were the application-level preconditions true when an API call was made?
  - e.g. was the API call made by an authorised party?

# Confidentiality

- ISO/IEC 25010:2011: “degree to which a product or system ensures that data are accessible only to those authorized to have access”
- Blockchain platforms are not good for confidentiality, because mining nodes cross-check contents of all transactions
- Not just the plain data, also need to be concerned with re-identification attacks, patterns from transaction analytics, and graph datamining
  - Identity of parties?
  - Transaction volume?
  - Transaction meta-data?
    - e.g. characteristic patterns in time-of-day for transactions, geolocation of source IP addresses of submitted transactions



# Confidentiality for Blockchain Applications

Might you use...	But...
pseudonymous addresses	<ul style="list-style-type: none"><li>if addresses are re-used, can start to recover identity information</li></ul>
private blockchain	<ul style="list-style-type: none"><li>check all nodes are authorised to see all the data!</li></ul>
distributed ledger where ledgers and transactions are shared only with parties of interest (e.g. Corda, Hyperledger Fabric)	<ul style="list-style-type: none"><li>can be harder to ensure property rights for digital assets (“double spending” issues)</li></ul>
encryption for on-chain confidential data	<ul style="list-style-type: none"><li>more complex key management</li><li>key loss reveals all old data too</li><li>quantum? or other breaks to the crypto schemes</li></ul>
keep data off-chain, use conventional technology for access control	<ul style="list-style-type: none"><li>more complex system design</li><li>less value directly from blockchain for sharing data</li></ul>

# Privacy

- Similar but different to confidentiality
- Privacy is controlling access to yourself; and not just information
  - e.g. physical privacy in your house
  - 1948 Universal Declaration of Human Rights:  
*No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honor and reputation. Everyone has the right to the protection of the law against such interference or attacks.*
- Even for information, privacy has different regulation than confidentiality
  - General Data Protection Regulation (GDPR) in Europe
    - Including “right to erasure (right to be forgotten)”, but can’t delete on blockchain?
- For privacy in blockchain-based applications, use same general approaches as for confidentiality, targeting privacy policies



# Non-Repudiation

- ISO/IEC 25010:2011: “degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later”
- Main support is from blockchain’s immutable ledger
  - But be careful of probabilistic immutability in Nakamoto consensus; use the right number of confirmation blocks for your application’s risk profile
- Some support from public key signatures for transactions
- Just because data is recorded on-chain, doesn’t mean it’s true!
  - Someone might have stolen my private key?
  - Sender might have fraudulently recorded false data in a transaction
- Only have non-repudiation that the transaction happened
- If using hashes on-chain for off-chain data, need to retain original data

# Accountability

- ISO/IEC 25010:2011: “degree to which the actions of an entity can be traced uniquely to the entity”
- Main support in blockchain platforms is from cryptographically-signed transactions
  - Does a single entity control the signing key? Good key management is critical!
- Most public blockchains try to directly stop this
  - Pseudonymous IDs are too weak to
  - New privacy coins (Zcash, Dash, Monero, ...) try to provide anonymity like cash
- KYC/AML-CTF
  - “Know Your Customer”/ “Anti-Money Laundering, Counter-Terrorism Financing”
  - e.g. Australian AUSTRAC regulation requires KYC/AML by crypto-currency exchanges
- Private blockchains are normally permissioned, and know who users are
- Authentication requires off-chain mechanisms to establish identity
- Blockchain can be used to communicate or ensure integrity of KYC information

# Authenticity

- ISO/IEC 25010:2011: “degree to which the identity of a subject or resource can be proved to be the one claimed”
- Major issue in supply chain, e.g. is this real Australian baby formula?
- Many blockchain applications are trying to improve supply chain quality
- For physical assets, hard to guarantee, unless a unique physical “fingerprint”
  - e.g. diamonds, DNA?
- Attacks include fake duplicate labels, substitution in (or reuse of) original packaging
- Need a way of comparing blockchain record to the physical item
- For digital assets, much easier on blockchain!
  - Put authoritative register on blockchain, or have an off-chain authority sign the asset

# Reliability



# ISO/IEC 25010:2011 Reliability Characteristics

- Reliability
  - degree to which a system, product or component performs specified functions under specified conditions for a specified period of time
- Availability
  - degree to which a system, product or component is operational and accessible when required for use
- Recoverability
  - degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system
- Maturity
  - degree to which a system, product or component meets needs for reliability under normal operation
- Fault-Tolerance
  - degree to which a system, product or component operates as intended despite the presence of hardware or software faults

# Availability

- ISO/IEC 25010:2011: “degree to which a system, product or component is operational and accessible when required for use”
  - A measure could be something like “probability of being able to provide service at any given time” or “
  - Affected by the other reliability characteristics, such as probability of failure, fault tolerance, time to recovery, etc
- Blockchain platform is highly redundant (many nodes)
- Easy to subscribe to updates to get replicas of the ledger
- An application can run many redundant blockchain nodes locally  
→ high read availability
- Write availability is a different story...

# Latency Variability Can Limit Write Availability

- Scenario
  - your application sends a transaction to update blockchain ledger
  - waiting... waiting...
  - ? should I keep waiting? or was there a service failure?  
(i.e. an availability problem)
- You will define some limit on your waiting time
- Long tail of variability in latency means sometimes it will take even longer
- But, by then your application will have decided a failure has happened
- Complex trade-off:
  - Availability vs. Cost (gas/fee) vs. Latency (# confirmations) vs. Risk (# confirmations)
- Mainly a problem with public blockchains?
  - (Depending on consensus mechanism, mining node policies, etc)

“On availability for blockchain-based systems”  
<https://ieeexplore.ieee.org/document/8069069>

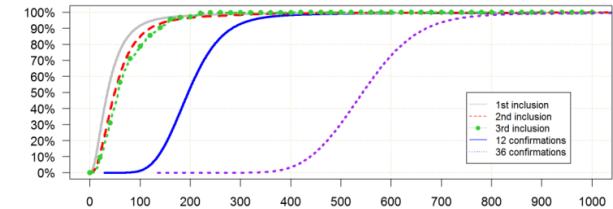


Figure 5: Time (sec) for first inclusion and commit (12 or 36 confirmations), as well as second and third inclusion of transactions that were previously not included in main chain.

# Availability for Blockchain-Based Applications

- A blockchain-based application has many components
  - Even if the blockchain platform works, your other components may fail
- Use normal availability-increasing design strategies for the architecture, for example...
  - Increase quality of each component & connector
    - High quality software and hardware (!)
  - Eliminate single points of failure/ increase redundancy
    - Load balancing/failover monitoring and routing
    - Stateless server components
      - Blockchain can help enable “stateless” server component (use blockchain to store the state)
  - Detect and recover from failures
    - Hot backup/failover servers

# Recoverability

- ISO/IEC 25010:2011: “degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system”
- Blockchain platform is likely to recover – consensus mechanisms are designed to autonomically recover nodes and ledger consistency
  - In worst case in a public blockchain, there might be a hard fork in the ledger
  - Normally, recoverability will mostly be a concern for other parts of the application architecture
- But, can you abort a blockchain transaction, to safely retry?
  - Most blockchains do not explicitly provide a support for this!



# How to Abort a Transaction in Ethereum?

- If you send a transaction into the pool, and it's not yet included, can you abort it?
    - You've changed your mind, or something is wrong, or taking too long to commit
  - Ethereum transactions have a nonce, normally increment per transaction
    - Miners will consider an old or reused nonce as being “outdated”
1. Issue a competing null transaction with same address & transaction nonce, with a higher fee
    - e.g. send 0 Ether to yourself, or invoke a smart contract to raise an exception
  2. Reissue the “same” transaction from same address & nonce, with a higher fee
    - Higher fee means it has different hash value, so will be seen as “different”

# Does Transaction Aborting in Ethereum Work?

- It's not guaranteed to work
- This trick creates a race condition; either transaction in the pool might get included
  - Depends on not just the fee for the new transaction, but also the fee for the initial transaction
- Also unclear if transaction nonce is always honoured by miners (c.f. “Blockchain anomaly” discussed earlier)

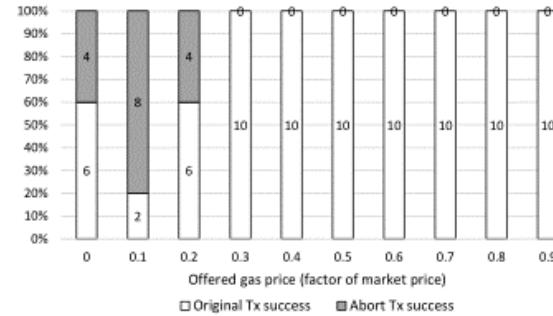


Figure 11: Underbidding market fee and automatic abort after 10 minutes if the original Tx was not included

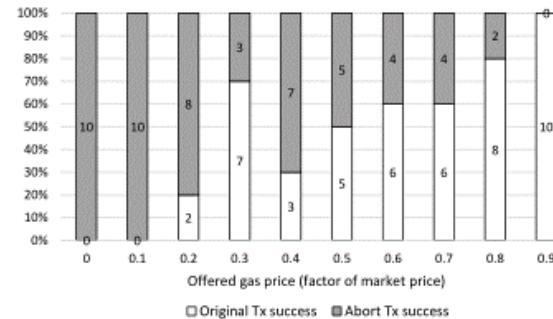


Figure 12: Underbidding market fee and automatic abort after 3 minutes if the original Tx was not included

“On availability for blockchain-based systems”  
<https://ieeexplore.ieee.org/document/8069069>

# Maturity (“Reliability”)

- ISO/IEC 25010:2011: “degree to which a system, product or component meets needs for reliability under normal operation”
    - *My opinion: not a great name for this...*
  - Availability is about readiness for correct service
- vs.
- Reliability is about continuity of correct service
  - Previous discussion about availability for blockchain-based applications applies here too

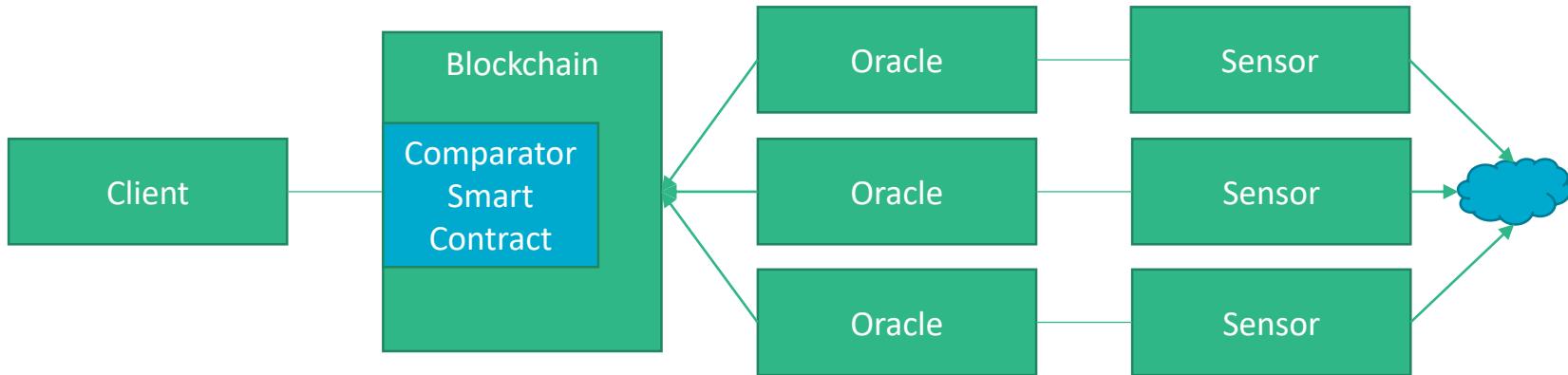
# Fault-Tolerance

- ISO/IEC 25010:2011: “degree to which a system, product or component operates as intended despite the presence of hardware or software faults”
- Blockchain platform tolerates faults
  - Hardware faults in individual miners: other miners will keep working
  - Software faults too, if there is enough diversity of mining software
    - In worst case, might get a hard fork
- For blockchain applications
  - Use normal design strategies for fault tolerance (redundancy, monitor, detect, isolate, recover) for other parts of the architecture
  - Smart contracts are not normally fault-tolerant for software faults
  - Smart contracts can be used to do fault tolerance logic for off-chain components...



# Oracle Faults

- Can use on-chain M-of-N voting for redundant oracles



# Summary



# Blockchains as Dependable Systems

- Trust – accepted dependence
- Trustworthy – justified Trust
- Need Evidence to justify Engineering Assurances
- Validation
  - Sort-of-but-not-quite: “am I solving the right problem?”
  - Does my requirements/specification/system/design address/solve the user problem?
- Verification
  - Sort-of-but-not-quite: “am I solving the problem right?”
  - Does my system/design meet the requirements specification?
- Can provide evidence based on architectural analysis, early in the lifecycle
- Blockchains are increasingly relied on; need evidence they are trustworthy

# ISO/IEC 25010:2011 Dependability & Security

- Functional Suitability
  - Correctness
  - Completeness
  - Appropriateness
    - Code is Not Law
    - How to specify smart contracts?
- Security
  - Integrity & Clark-Wilson policy model
    - Blockchain “anomaly” in Nakamoto consensus: possible transaction reordering
  - Confidentiality
    - And its difference to Privacy
  - Non-Repudiation
  - Accountability
    - Avoid or support? KYC/AML-CTF
  - Authenticity
- Reliability
  - Availability: readiness for service
    - Affected by latency variability
  - Recoverability
    - Aborting transactions
  - “Maturity”
    - Reliability: continuous service
  - Fault Tolerance
    - Use of smart contracts for redundant oracles

# Assignment 2

# Design & Evaluation of a Blockchain-Based System

- THIS IS NOT A PROGRAMMING ASSIGNMENT: DO NOT WRITE CODE
- WE GIVE YOU OUTLINE OF A SYSTEM AND 4 NFPS/REQUIREMENTS
  - If you need more assumptions or context, then make and describe them
  - Totally OK to research more info about the problem domain – provide citations/references
- SHOW US 2 DESIGNS
  - One using private blockchain, one using public blockchain
- HIGH-LEVEL EVALUATION OF THE TWO DESIGNS
  - Using ATAM “Scenarios” & “Utility Tree” structures, but in a spreadsheet not a picture of a tree
- SUMMARY CHOICE & RATIONALE BETWEEN 3 DESIGNS
  - Conventional technology, private blockchain, public blockchain



# THANK YOU

[www.data61.csiro.au](http://www.data61.csiro.au)



# COMP6452 Lecture 8: Architectural Patterns for Blockchain Applications

Xiwei (Sherry) Xu ([xiwei.xu@data61.csiro.au](mailto:xiwei.xu@data61.csiro.au))

8<sup>th</sup> of April, 2019

[www.data61.csiro.au](http://www.data61.csiro.au)

# Outline

- Design Pattern Essential
  - What are Design Patterns?
  - What are Architectural Patterns?
  - Pattern Template
- Architectural Patterns for Blockchain-based Applications
  - Overview
  - Interaction with External World (5 patterns)
  - Data Management (4 patterns)
  - Security (4 patterns)
  - Structural Patterns of Contract (5 patterns)
  - Deployment (2 patterns)
- Summary



# Design Pattern Essential

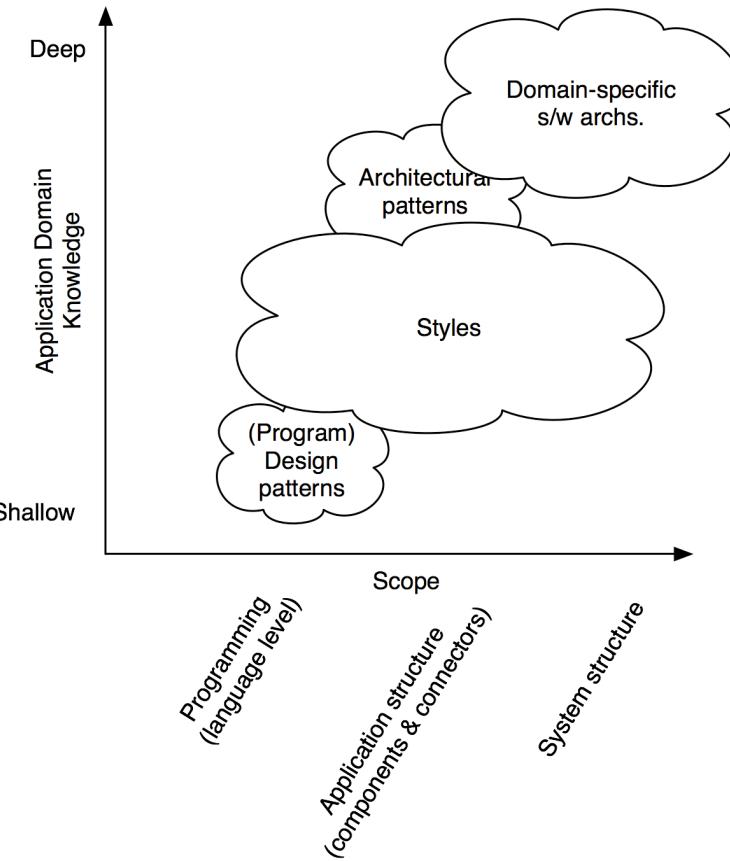
# What is Design Pattern?

- To solve a recurring problem in software development
- Not a finished design that can be transformed directly into code
- A description or template for how to solve a problem
  - Define constraints that restrict the roles of architectural elements
    - Processing
    - Connectors
    - Data
  - Define constraints that restrict the interaction among these elements
- Cause trade-offs among quality attributes



# Architecture Design

- From scratch
  - Unexpected solutions can be found
  - Labour-intensive and error-prone
- Apply a generic solution/strategy (Architectural style/ Design pattern) and adapt it to the problem
  - Reuse, less work and less errors
  - Generic solution might be ill-fitting or too generic



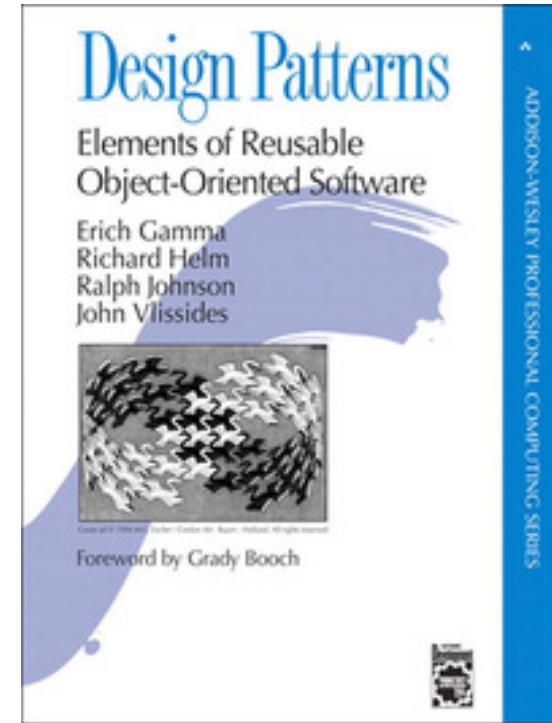
# Advantages of Patterns

- Speed up the development process by providing tested, proven development paradigms
  - Design patterns document the efforts of the experts
  - Design patterns concern with a flexible software architecture
- Effective software design requires considering issues that may not become visible until later in the implementation
  - Reuse can prevent subtle issues that can cause major problems
  - No need to reinvent the wheel
- Better code readability for programmers and architects familiar with the patterns

# Gang of Four (GoF)

## Classic Object Oriented Design Patterns

- GoF are Erich Gamma, Richard Helm, Ralph Johnson and John Vissides
- GoF document 23 classic software design patterns in their book
  - Design Patterns: Elements of Reusable Object-Oriented Software
- The GoF book published at October 1994 and documented design patterns already exist but not documented before



# Four Essential Elements

## Pattern Name

- Describe a design problem, its solutions and consequences in a word or two

## Problem

- Explain the problem and its context
- Conditions must be met

## Solution

- Describe the elements that make up the design
- Relationship, responsibilities, and collaborations

## Consequence

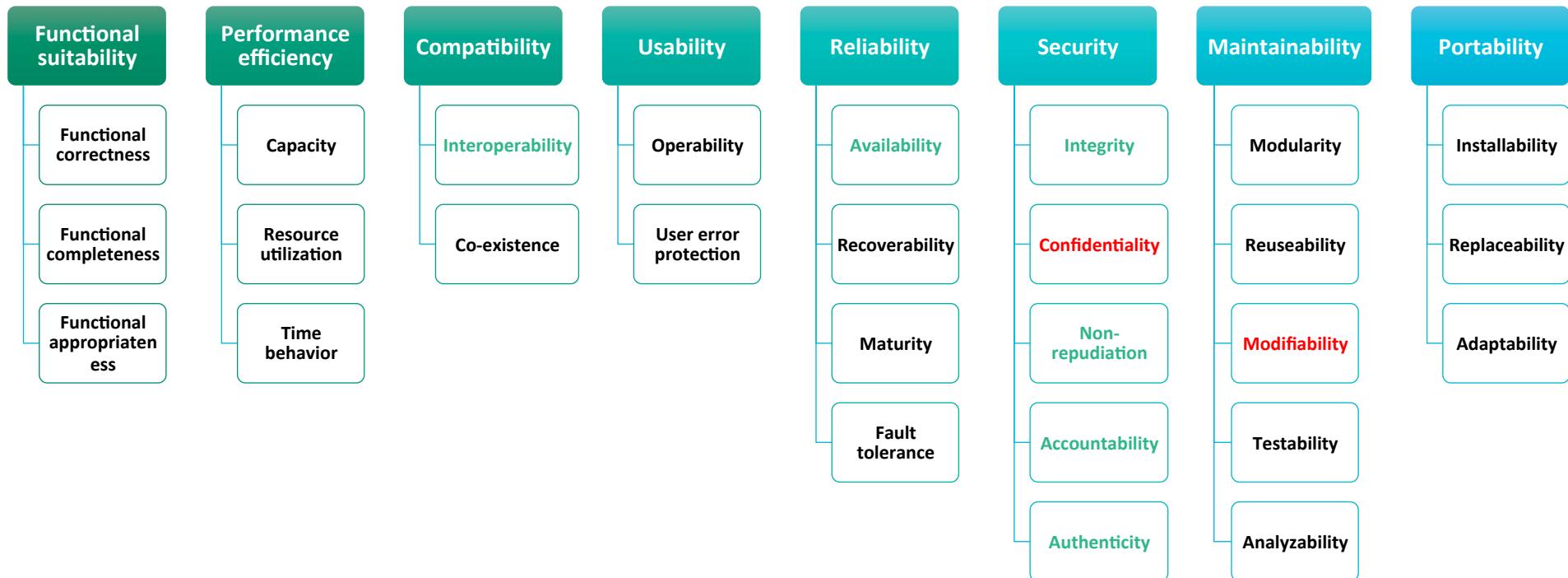
- Results and trade-offs of applying the pattern
- Critical for understanding the costs/benefits

# Non-Functional Properties

Non-Functional Properties arise from Architectural Design Choices

- There are two kinds of requirements:
  - Functional Requirements (i.e. what are the inputs and outputs)
  - Non-Functional Requirements (a.k.a. *Qualities*, or *-ilities*)
    - e.g. “Performance” (latency, throughput, ... )
    - e.g. “Security” (confidentiality, integrity, availability, privacy, ...)
    - e.g. Usability, Reliability, Modifiability, ...

# ISO/IEC 25010:2011 Quality Model



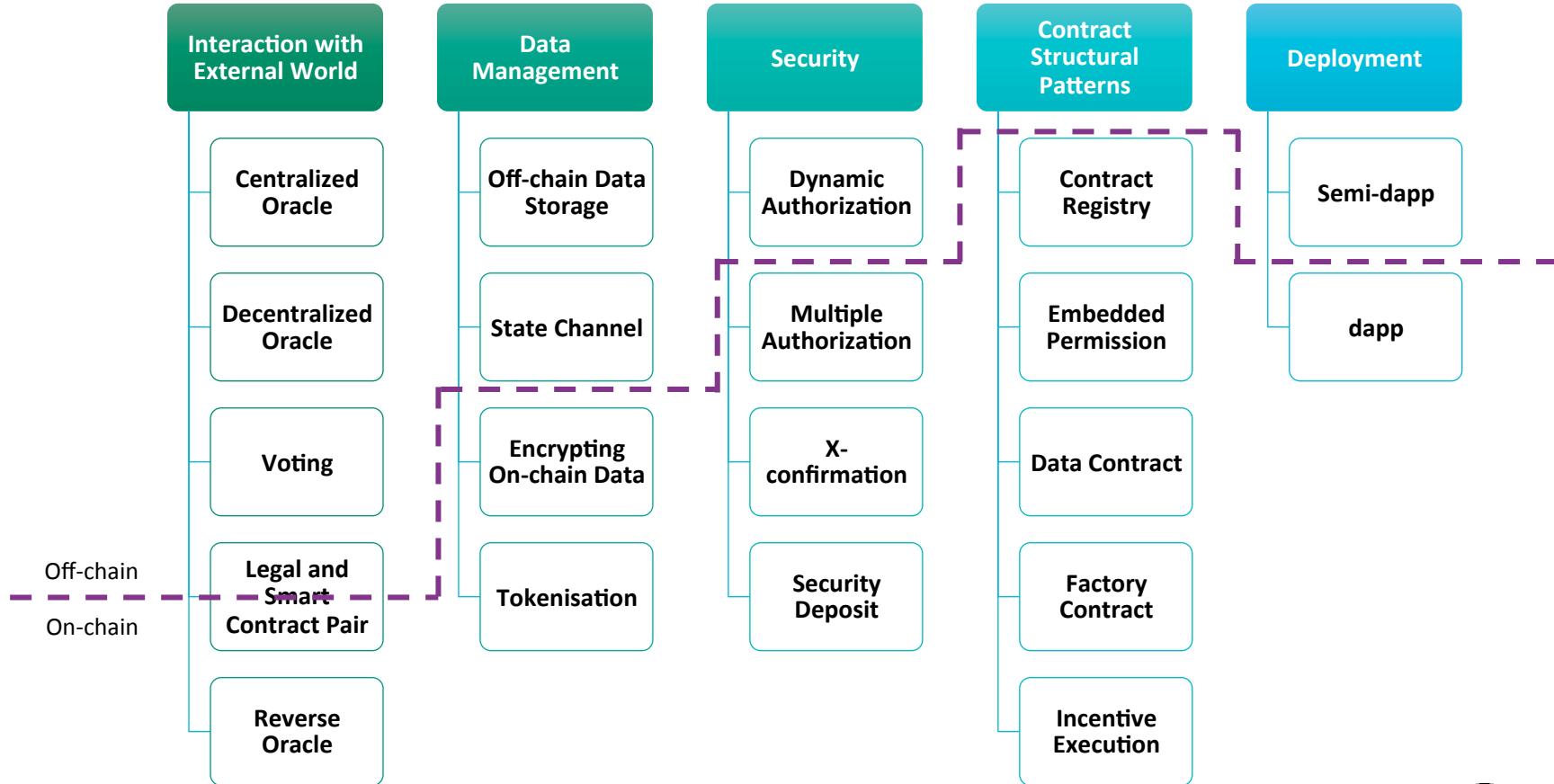
Adopting a design pattern causes trade-offs among quality attributes



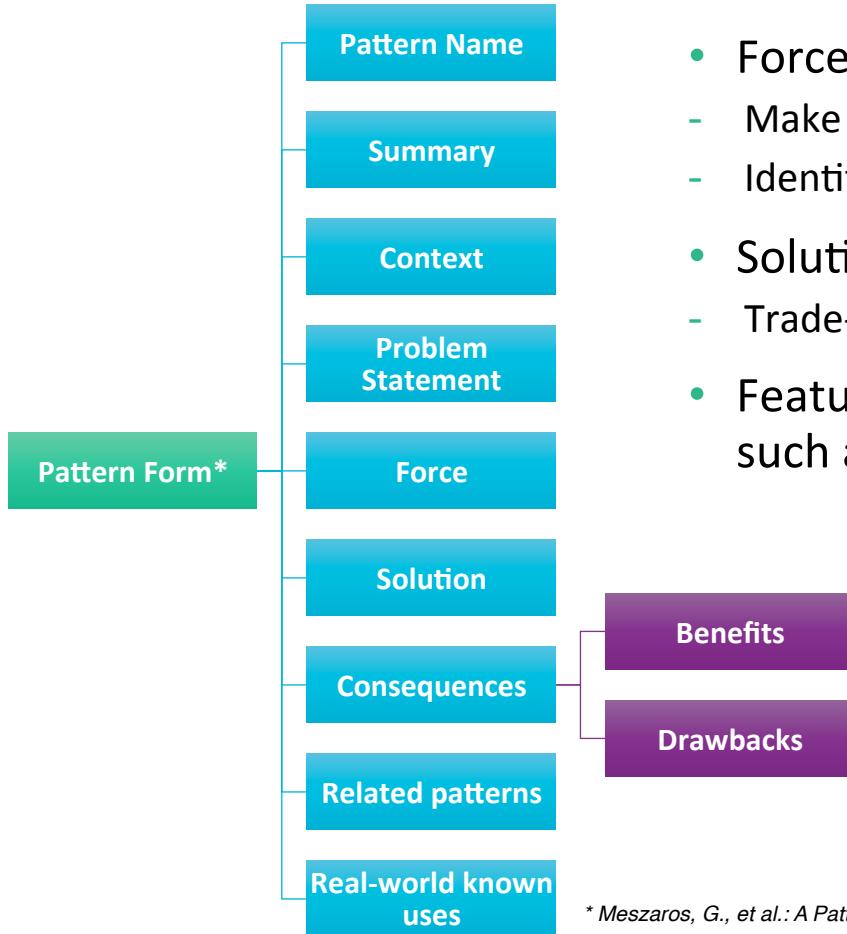
# Blockchain-based Application Pattern Collection



# Pattern Collection



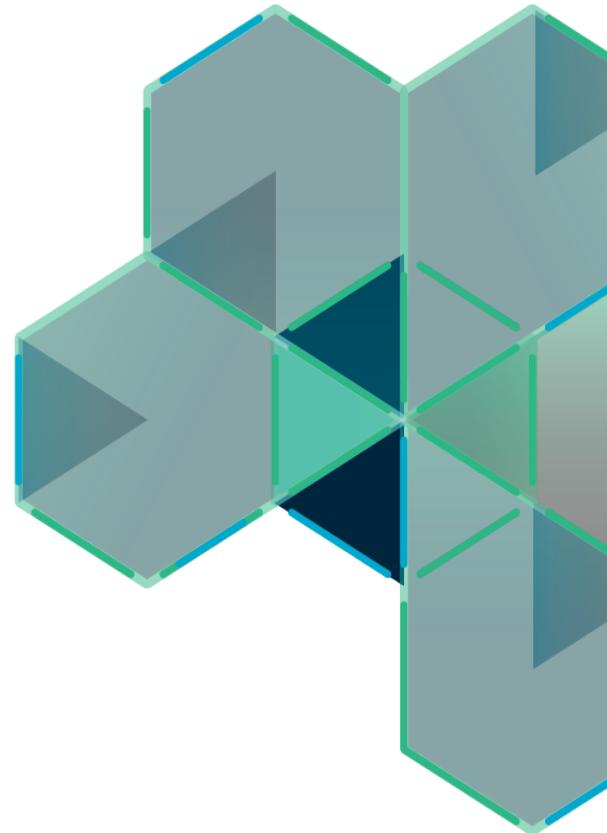
# Pattern Form



\* Meszaros, G., et al.: *A Pattern Language for Pattern Writing. Pattern languages of program design* (1998)

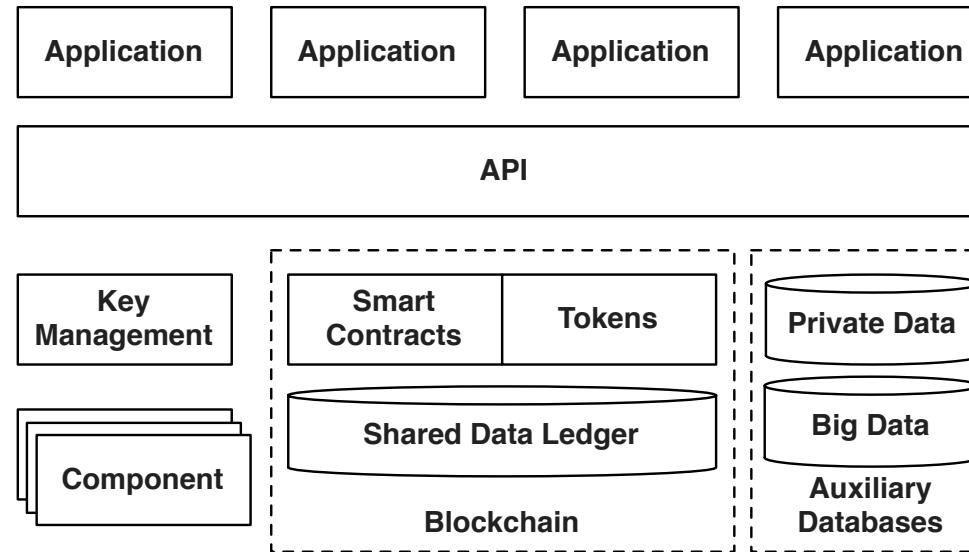
# Pattern Collection

- **Interaction with External World (5)**
- Data Management (4)
- Security (4)
- Contract (5)
- Deployment (2)



# Overview

- Blockchain can be a component of a big software system
- Communicate with other components within the software system



# Pattern 1: Centralized Oracle 1/3

- **Summary**

- Introduce the state of external systems into the closed blockchain execution environment through a single centralized oracle

- **Context**

- Blockchain-based applications might need to interact with other external systems
- Validation of transactions might depend on external state

- **Problem**

- Blockchain is a self-contained execution environment
- Smart contracts are pure functions that can't access external systems

- **Forces**

- Closed environment
  - Secure, isolated execution environment
- Connectivity
  - General-purpose applications might require information from external systems
- Long-term availability and validity
  - External state used to validate a transaction may change or even disappear



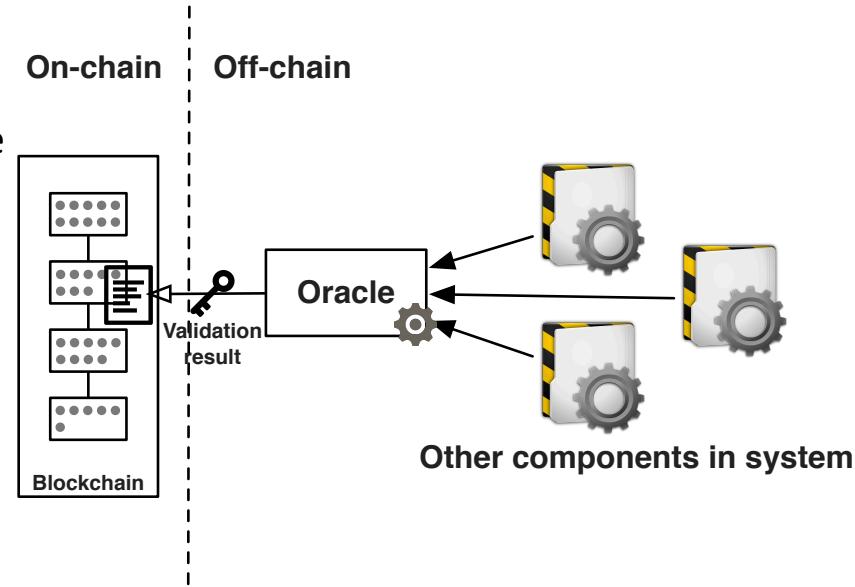
# Pattern 1: Centralized Oracle 2/3

- **Solution**

- Oracle assists in evaluating conditions that cannot be expressed in a smart contract
- Oracle injects the result to the blockchain in a transaction signed using its own key pair
- Validation of transactions is based on the authentication of the oracle

- **Consequences**

- Benefits
  - Connectivity: Closed environment of blockchain is connected with external world through Oracle
- Drawbacks
  - Trust: Oracle is trusted by all the participants
  - Validity: External states injected into the transactions can not be fully validated by miners
  - Long-term availability and validity: External state used to validate transaction changes after the transaction was originally appended to the blockchain

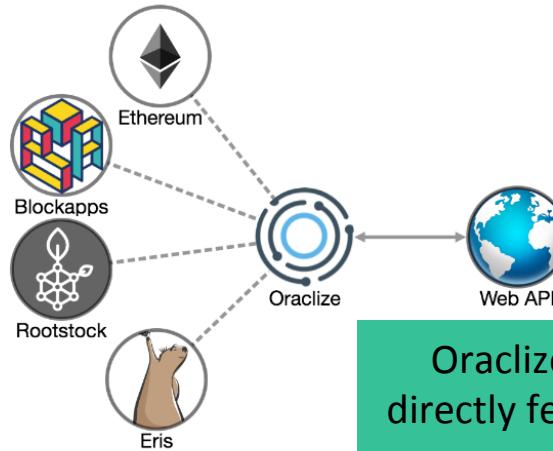


# Pattern 1: Centralized Oracle 3/3

- **Related Patterns**

- *Pattern 2. Decentralized Oracle*
- *Pattern 4. Reverse Oracle*

- **Known uses**



Oraclize uses trusted hardware to directly fetch information from trusted execution environment (TEE)



Oracle in Bitcoin evaluates user-defined expressions based on the external state



Corda has a embedded oracle mechanism using Intel Software Guard Extension (SGX) for hardware attestation to prevent unauthorized access outside of the SGX environment

# Pattern 2: Decentralized Oracle 1/3

- **Summary**

- Introduce the state of external systems into the closed blockchain execution environment through *decentralized* oracle

- **Context**

- Blockchain-based applications might need to interact with other external systems
- Validation of transactions relies on *oracle* to inject the external state

- **Problem**

- A centralized oracle introduces a single trusted third party

- **Forces**

- Reliability
  - Centralized oracle is a single point of failure
- Variety of data sources
  - Static web page, physical sensor, input from a human
  - Multiple sources might come from a single authorized source



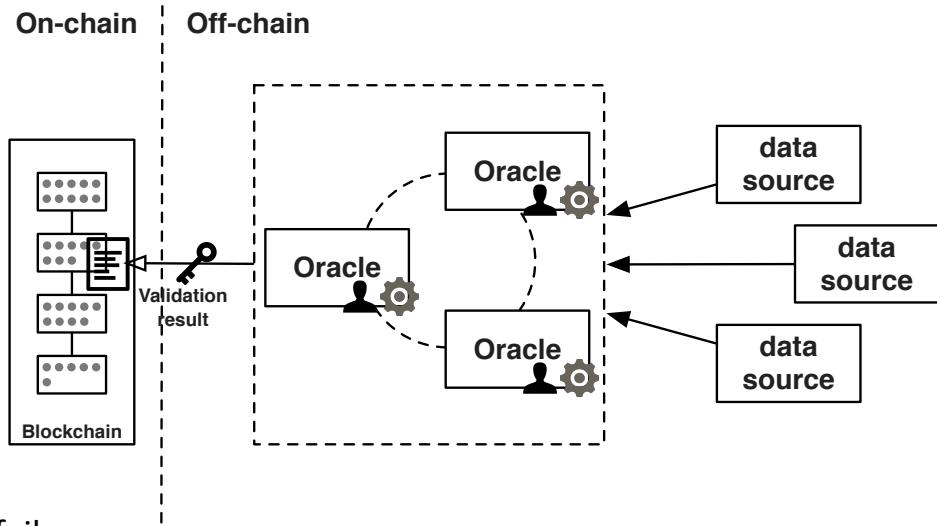
# Pattern 2: Decentralized Oracle 2/3

## • Solution

- Decentralized oracle based on multiple servers and multiple data sources
- Consensus on the external status
  - K-out-of-M threshold signature

## • Consequences

- Benefits
  - Reliability
    - Risk is reduced from a single point of failure
    - Improves the likelihood of getting accurate external data
- Drawbacks
  - Trust: All the oracles that verify the external state are trusted by all participants involved in transactions
  - Time: Get required information from multiple data sources and reach a consensus for the final result
  - Cost: increase with the number of oracles being used

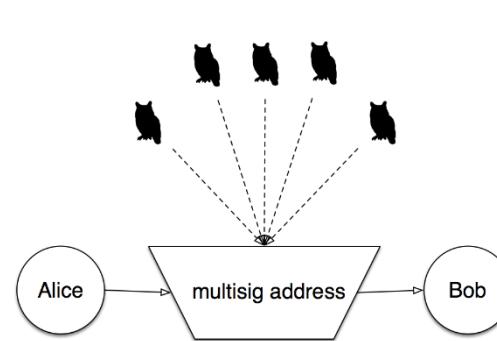


# Pattern 2: Decentralized Oracle 3/3

- **Related Patterns**

- *Pattern 1. Centralized Oracle*
- *Pattern 3. Voting*
- *Pattern 4. Reverse Oracle*

- **Known uses**



Orisi on Bitcoin allows participants involved in a transaction to select a set of independent oracles



Augur is a prediction market that use human oracles



Gnosis is a prediction market allows users to choose oracles they trust

# Pattern 3: Voting 1/3

- **Summary**

- Voting is a method for a group of blockchain users of a decentralized oracle to make a collective decision or to achieve a consensus

- **Context**

- Public access of blockchain provides equal rights
- Participant has the same ability to access and manipulate the blockchain

- **Problem**

- Participants have different preference

- **Forces**

- Decentralization
  - Devolves responsibility and capability from a central location to all the participants
- Consensus
  - Participants need to reach an agreement to make decision



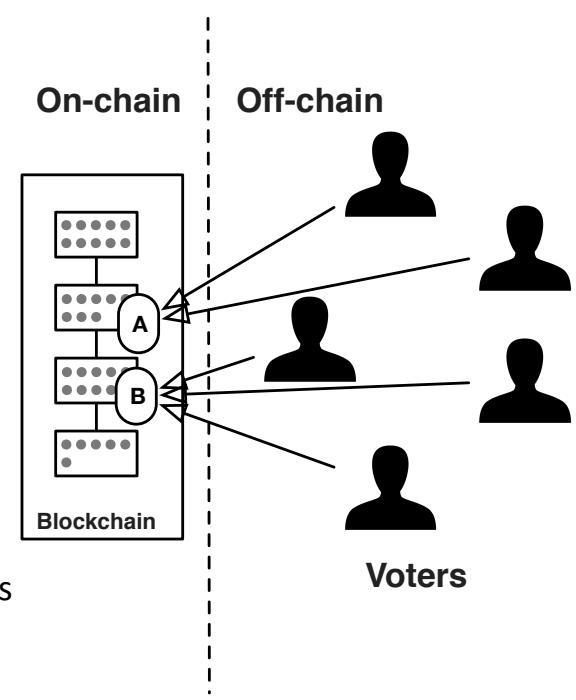
# Pattern 3: Voting 2/3

## • Solution

- Vote through sending transaction through blockchain account
- Voting transaction is signed by the private key
  - Represent the right to make decision
  - Might be weighted by the owned resource

## • Consequences

- Benefits
  - Equality: Participants use their right to make decision
  - Consensus: participants with different preferences can reach consensus
- Drawbacks
  - Collusion: Collude during voting to gain benefit
  - Permission grant: Pseudonymity allows participant to gain additional voting power
    - Through owning multiple blockchain addresses
  - Time: Long voting/dispute time window



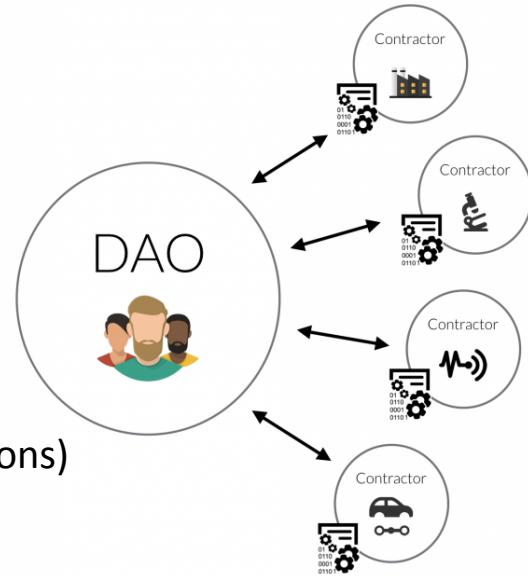
# Pattern 3: Voting 3/3

- **Related Patterns**

- *Pattern 2. Decentralized Oracle*
- *Pattern 10. Multiple Authorization*
- *Pattern 13. Security Deposit*

- **Known uses**

- Voting is used in DAOs (Decentralized Autonomous Organizations)
- Gnosis
  - Voting is used to challenge the reported outcomes
- Augur
  - Similarly, voting is used to resolve dispute



# Pattern 4: Reverse Oracle 1/3

- **Summary**

- The reverse oracle of an existing system relies on smart contracts to validate requested data and check required status

- **Context**

- Off-chain components might need to use the data stored on the blockchain
- Off-chain components might need to the smart contracts to check certain conditions

- **Problem**

- Some domains use very large and mature systems, which comply with existing standards
- Leverage the existing complex systems with blockchain without changing the core of the existing systems

- **Forces**

- Connectivity
  - Integrate blockchain to leverage the unique properties of blockchain
- Simplicity
  - Introduce minimal changes to the existing system



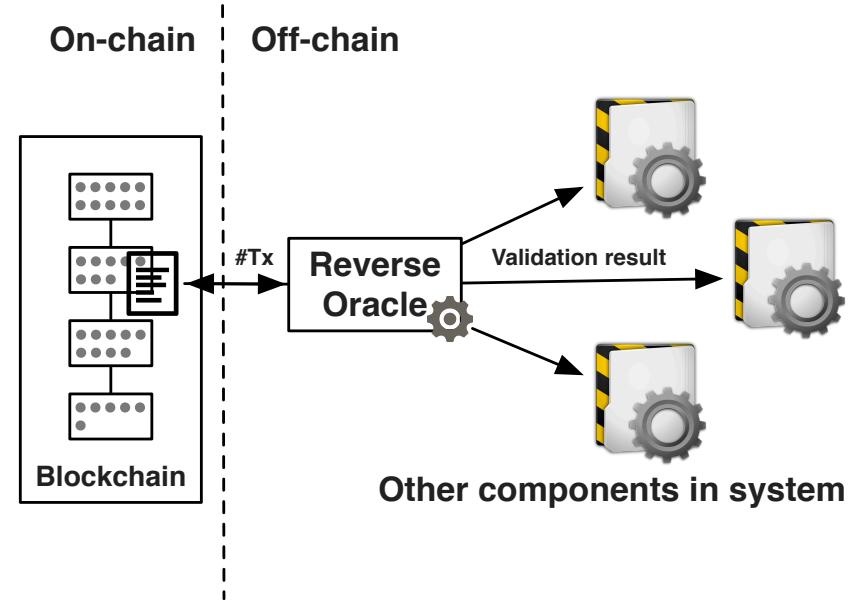
# Pattern 4: Reverse Oracle 2/3

## • Solution

- Transaction ID and Block ID can be integrated into existing system
- Validation is on blockchain using smart contract
- An off-chain component is required to query blockchain

## • Consequences

- Benefits
  - Connectivity:
    - Blockchain is integrated into a system through adding IDs of transaction as a piece of data into the system
- Drawbacks
  - Non-intrusive
    - Writing and reading blockchain might need changes to the existing system



# Pattern 4: Reverse Oracle 3/3

- **Related Patterns**

- *Pattern 1. Centralized Oracle*
- *Pattern 2. Decentralized Oracle*

- **Known uses**

- Identitii
  - Enrich payment in banking systems with documents and attributes using blockchain
  - Identity token exchanged between the banks through SWIFT protocol
- Slock.it
  - Autonomous objects and universal sharing network
    - Devices sell or rent themselves, and pay for services provided by others
  - Availability information is stored on blockchain
    - Validity checking is on blockchain



Slock.it

# Pattern 5: Legal and Smart Contract Pair 1/3

- **Summary**

- A bidirectional binding is established between a legal agreement and the corresponding smart contract

- **Context**

- Legal industry is digitized
  - Digital signature is a valid way to sign legal agreements
- Richardian contract (Mid 1990s)
  - interpret legal contracts digitally without losing the value of the legal prose
- An independent trustworthy execution platform is needed to execute the digital legal agreement

- **Problem**

- Bind a legal agreement to the corresponding smart contract to ensure 1-to-1 mapping

- **Forces**

- Authoritative source: 1-to-1 mapping makes SC the authoritative source of legal contract
- Secure storage: Blockchain is a trustworthy data storage
- Secure execution: Blockchain provides a trustworthy computational platform

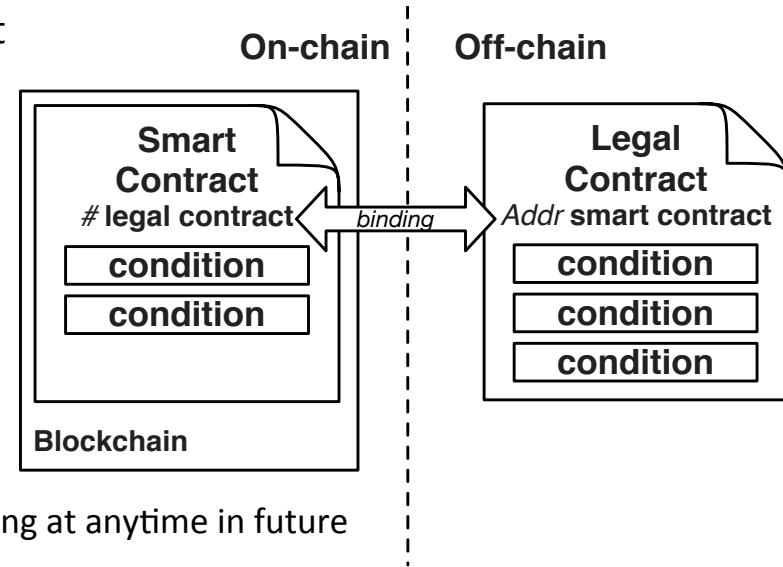
# Pattern 5: Legal and Smart Contract Pair 2/3

## • Solution

- SC implements conditions defined in the legal agreement
  - Checked and enforced by the smart contract
- SC has a blank variable to store hash of legal contract
- SC address included in the legal agreement
- Legal agreement hash is added to the SC variable

## • Consequences

- Benefits
  - Automation: SCs are programs running on blockchain
  - Audit trail: immutable historical transactions enable auditing at anytime in future
- Drawbacks
  - Expressiveness: Smart contract language might have limited expressiveness to express contractual terms
  - Enforceability: No central authority to decide a dispute or perform the enforcement of a court judgment
  - Interpretation: Ambiguity of natural language is a challenge to accurately digitize a certain legal term

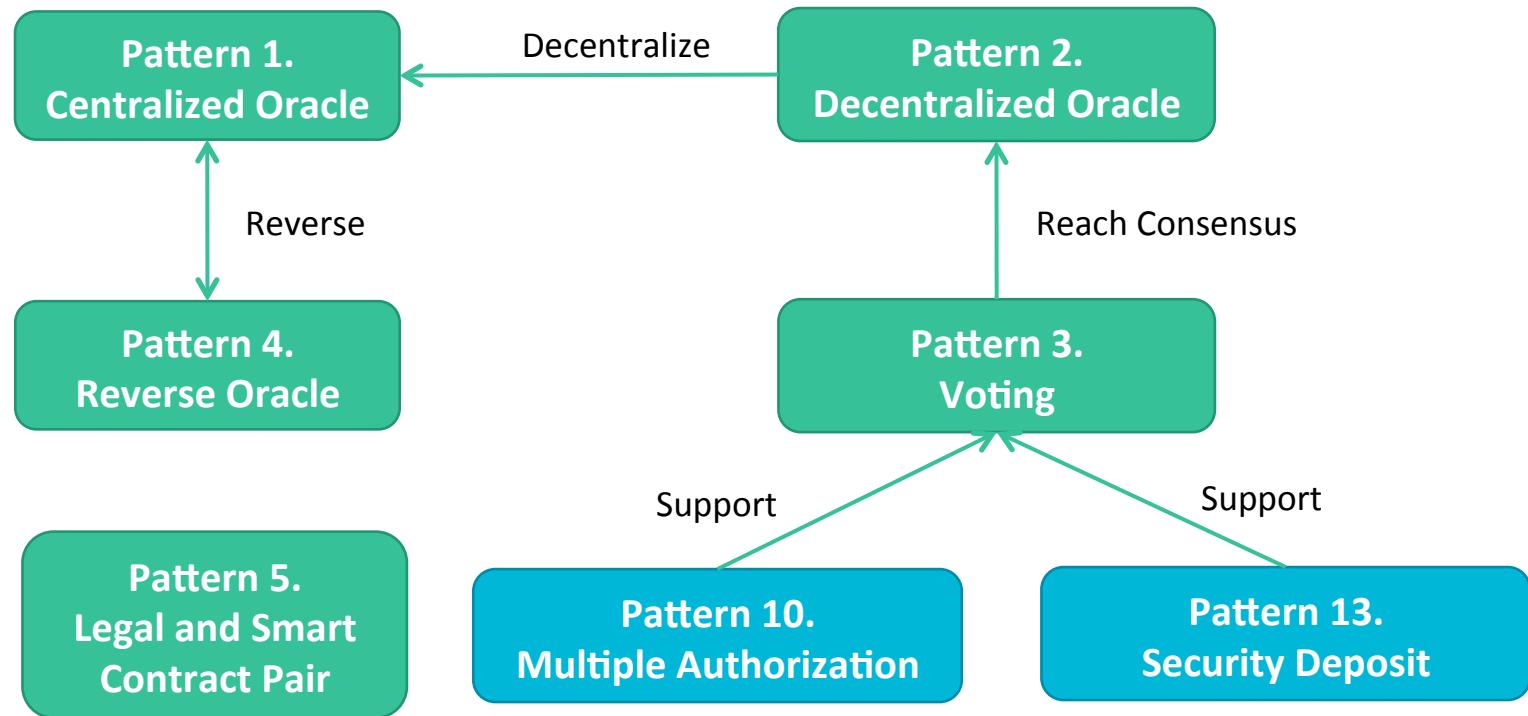


# Pattern 5: Legal and Smart Contract Pair 3/3

- **Related Patterns** N/A
- **Known uses**
  - OpenLaw
    - Legally binding and self-executable agreements on the Ethereum blockchain
    - The legal agreement templates are stored on a decentralized data storage, IPFS
  - Smart Contract Template proposed by Barclays uses legal document templates to facilitate smart contracts running on Corda
  - Accord Project explored the representation of machine-interpretable legal terms

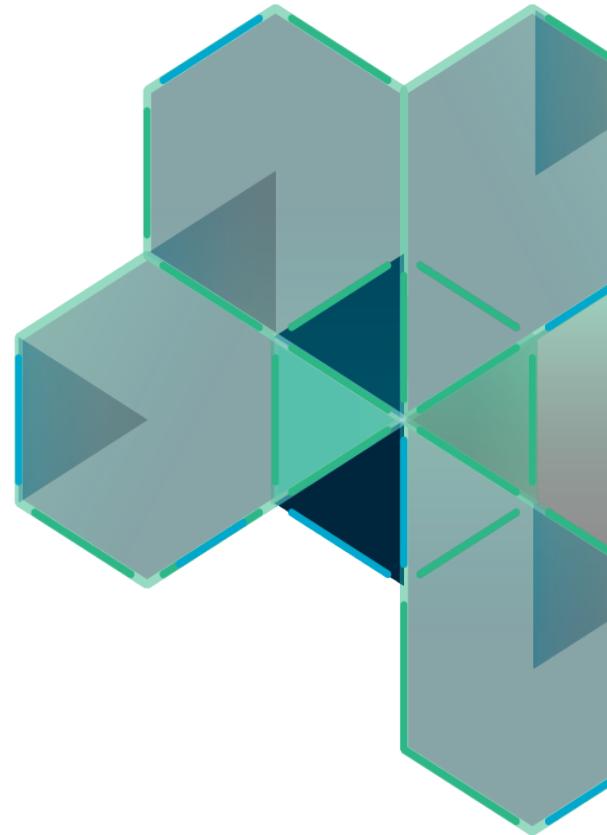


# Related Patterns



# Pattern Collection

- Interaction with External World (5)
- **Data Management (4)**
- Security (4)
- Contract (5)
- Deployment (2)



# Pattern 6: Encrypting On-chain Data 1/3

- **Summary**

- Ensure confidentiality of the data stored on blockchain by encrypting it

- **Context**

- Commercially critical data that is only accessible to the involved participants
  - Special discount price offered by a service provider to a subset of its users

- **Problem**

- Data privacy is a limitation of blockchain
  - All information on blockchain is publicly available to all participants
  - No privileged user: no matter public/consortium/private blockchain

- **Forces**

- Transparency
  - Historical transactions are publically accessible to enable validation of previous transactions
  - Transactions on public blockchain are accessible to anyone with access to internet
- Lack of confidentiality
  - Commercially sensitive data should not be stored on blockchain



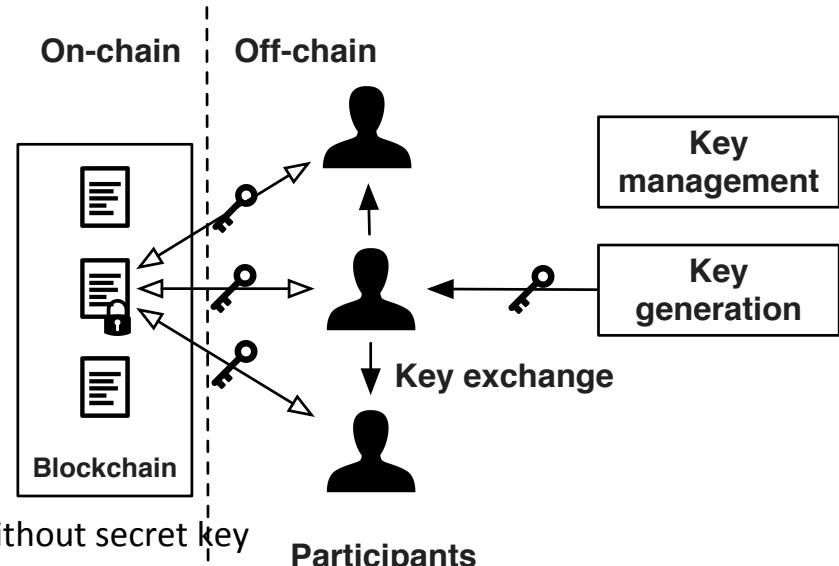
# Pattern 6: Encrypting On-chain Data 2/3

## • Solution

- Data is encrypted before being inserted into blockchain
  - Symmetric encryption
  - Asymmetric encryption

## • Consequences

- Benefits
  - Confidentiality: Encrypted data is useless to anyone without secret key
- Drawbacks
  - Compromised key: Encryption mechanism does not guarantee the confidentiality or integrity with a compromised or disclosed key
  - Access revocation: Access of Encrypted data is forever because of immutability
  - Immutable data: Subject to brute force decryption attack
    - Quantum computing
  - Key sharing: Off-chain key exchange otherwise accessible to all blockchain participants



# Pattern 6: Encrypting On-chain Data 3/3

- **Related Patterns**

- *Pattern 8. Off-Chain Data Storage*

- **Known uses**

- Encrypted queries from Oraclize
  - Developers encrypt the parameters of their queries before passing them to a smart contract
  - Oraclize can decrypt the call parameters
- Crypto digital signature from MLGBlockchain
  - Encrypting data before sharing data between the parties
- Hawk\* stores transactions as encrypted data on blockchain to retain the privacy of the transactions
  - Automatically generate a cryptographic protocol for a smart contract
  - Involved participants interact with the blockchain following the cryptographic protocol



\*Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 37th IEEE Symposium on Security and Privacy (S&P2016)

# Pattern 7: Tokenization 1/3

- **Summary**

- Using tokens to represent fungible goods for easier distribution

- **Context**

- Reduce risk in handling high value financial instruments by replacing them with equivalents
  - Tokens used in casino
- Tokens represent transferable and fungible goods
  - Shares or tickets

- **Problem**

- Tokens representing assets should be the authoritative source of the corresponding assets

- **Forces**

- Risk
  - Handling fungible financial instruments with high value is risky
- Authority
  - Tokens are the authoritative source of the assets

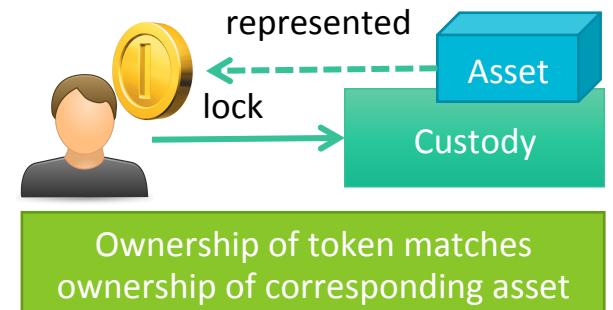
# Pattern 7: Tokenization 2/3

- **Solution**

- Native tokens on blockchain used to represent digital or physical assets
  - Transactions record the verifiable title transfer from one user to another
  - With limited condition checking
- Smart contract based data structure used to represent physical assets
- On-chain token is the authoritative source of the physical asset

- **Consequences**

- Benefits
  - Risk: Replacing high value financial instruments with equivalents
  - Authority
- Drawbacks
  - Integrity: Authenticity of the physical asset is not guaranteed automatically
  - Legal process for ownership:
    - Owner of an asset may be entitled to sell the asset without being required to create a transaction on the blockchain



# Pattern 7: Tokenization 3/3

- Related Patterns N/A

- Known uses

- Coloredcoin
  - Open source protocol for tokenizing digital assets on Bitcoin blockchain
- Digix
  - Use tokens to track the ownership of gold as a physical property



# Pattern 8: Off-chain Data Storage 1/3

- **Summary**

- Using hashing to ensure the integrity of arbitrarily large datasets

- **Context**

- Using blockchain to guarantee the integrity of large amounts of data

- **Problem**

- Limited storage capacity: full replication across all participants of the blockchain network
- Limited size of the block: Storing large amounts of data within a transaction is impossible
  - Block gas limit in Ethereum
- Data cannot take advantage of immutability or integrity guarantees without being stored on-chain

- **Forces**

- Scalability: Data is replicated permanently across all nodes
- Cost: Public blockchain charges real money: One-time cost
- Size: Limits of transaction size or block size
  - Bitcoin relays *OP\_RETURN* transactions up to 80 bytes



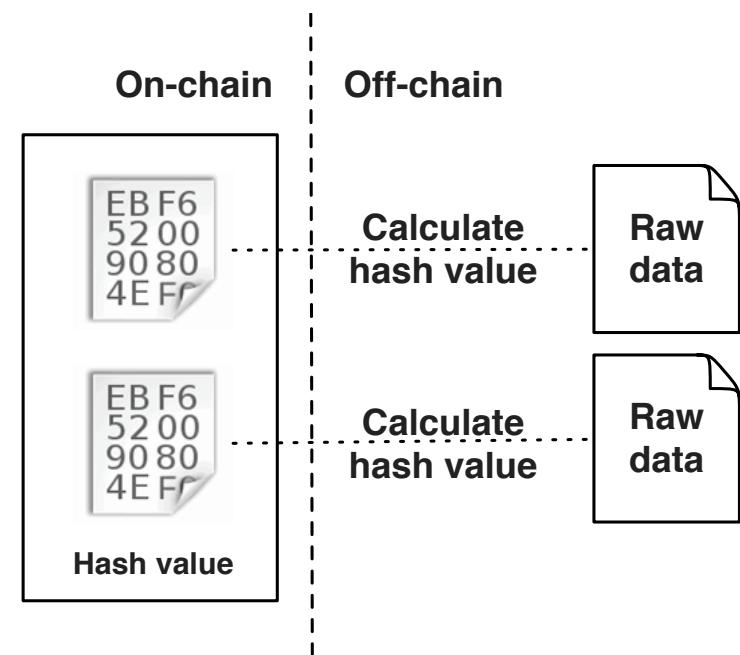
# Pattern 8: Off-chain Data Storage 2/3

## • Solution

- Data of big size
  - Data that is bigger than its hash value
- Hash value of the data is stored on blockchain
  - With other small sized metadata: a URI pointing to it

## • Consequences

- Benefits
  - Integrity:
    - Blockchain guarantees integrity of the hash value
    - Hash value guarantees integrity of the raw data
  - Cost: Fixed low cost for integrity of data with arbitrary size
- Drawbacks
  - Integrity
    - Raw data might be changed without authorization
    - Detectable but unrecoverable
  - Data loss: Off-chain raw data may be deleted or lost



# Pattern 8: Off-chain Data Storage 3/3

- **Related Patterns**

- *Pattern 9. State Channel*

- **Known uses**

- Proof-of-Existence (POEX.IO).

- Entering an SHA-256 cryptographic hash of a document into the Bitcoin blockchain
  - A “proof- of-existence” of the document at a certain time



- Chainy

- Smart contract running on Ethereum blockchain
  - Stores a short link to an off-chain file and its corresponding hash value in one place



# Pattern 9: State Channel 1/3

- **Summary**

- Micro-payments exchanged off-chain and periodically recording settlements for larger amounts on chain
- Can be generalized for arbitrary state updates

- **Context**

- Micro-payments are payments that can be as small as a few cents
  - E.g., payment of a very small amount of money to a WiFi hot-spot for every 10 KB of data usage

- **Problem**

- Decentralized design has limited performance: Long commit time
- High transaction fees on a public blockchain: Largely independent of the transacted amount

- **Forces**

- Latency
  - Long commitment time on blockchain
  - Micro-payment is expected to happen instantaneously
- Scalability: Data replicated permanently across all nodes
- Cost: Transaction fee might be higher than the monetary value associated with micro-payment transaction



# Pattern 9: State Channel 2/3

## • Solution

- Establish a payment channel between two participants
- Deposit from one or both sides locked up
- Payment channel keeps the intermediate states
- Only the finalized payment is on chain
- Frequency of settlement depends on use cases

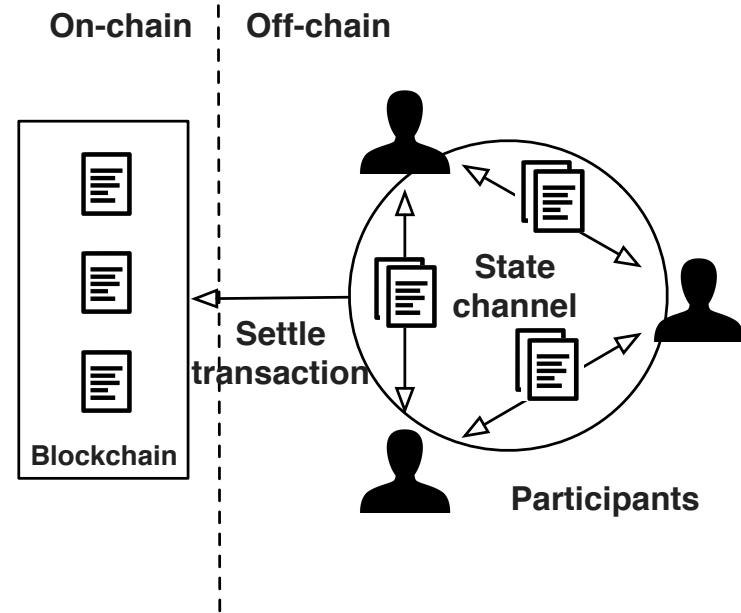
## • Consequences

### - Benefits

- Speed: off-chain transaction settled without waiting for blockchain network to include the transaction
- Throughput: off-chain transaction throughput is not limited by blockchain configuration
- Privacy: intermediate off-chain transactions do not show up in the public ledger
- Cost: only the final settlement transaction costs fee to be stored on blockchain

### - Drawbacks

- Trustworthiness: Micro-payment transactions are not immutable and can be lost after the channel is closed
- Reduced liquidity: Locked up security deposit reduces liquidity of channel participants



# Pattern 9: State Channel 3/3

- Related Patterns

- *Pattern 8. Off-Chain Data Storage*

- Known uses



BITCOIN LIGHTNING NETWORK

- Hashed Timelock Contracts (HTLCs)
  - *Hashlocks* and *timelocks* of Script
  - Receiver acknowledges receiving the payment before deadline by proof
- Bi-directional payment channel



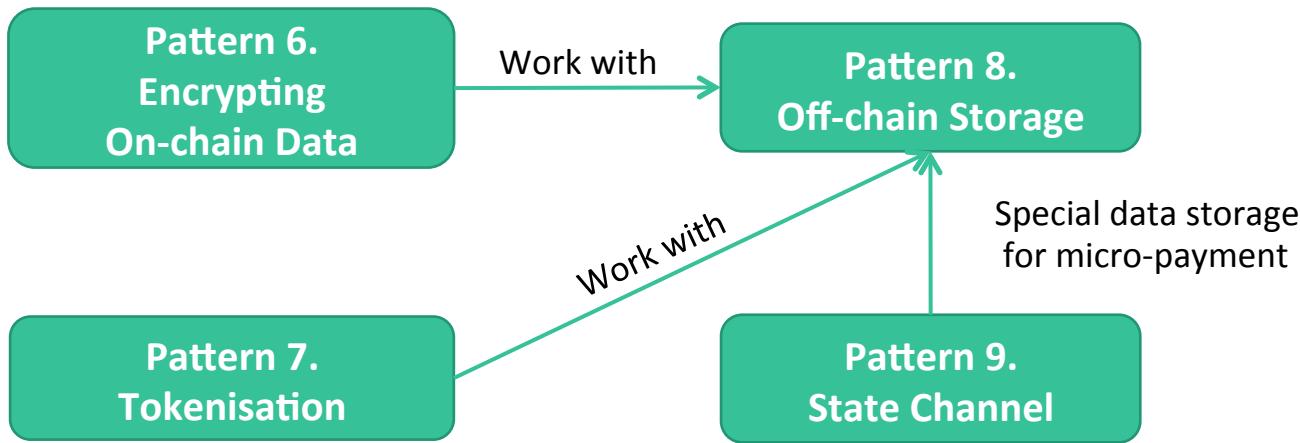
Raiden network on Ethereum



Orinoco on Ethereum is a payment hub for payment channel management



# Related Patterns



# Pattern Collection

- Interaction with External World (5)
- Data Management (4)
- **Security (4)**
- Contract (5)
- Deployment (2)



# Pattern 10: Multiple Authorization 1/3

- **Summary**

- A set of blockchain addresses which can authorize a transaction is pre-defined
- Only a subset of the addresses is required to authorize transactions

- **Context**

- Activities might need to be authorized by multiple blockchain addresses
  - A monetary transaction may require authorization from multiple participants

- **Problem**

- The actual addresses that authorize an activity might not be able to be decided due to availability

- **Forces**

- **Flexibility**
  - The actual authorities can be from a set of pre-defined authorities
- **Tolerance of compromised or lost private key**
  - Blockchain does not offer any mechanism to recover a lost or a compromised private key
  - Losing a key results in permanent loss of control over an account and smart contracts



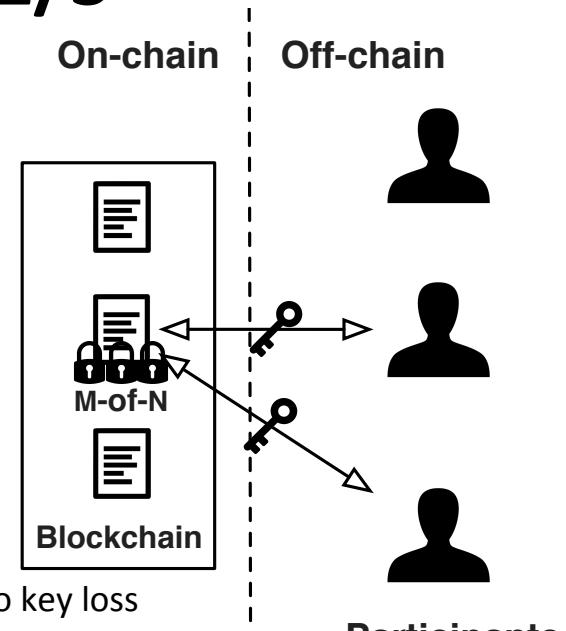
# Pattern 10: Multiple Authorization 2/3

## • Solution

- The set of blockchain addresses for authorization are not decided before the transaction being submitted to blockchain network
- Multiple signature mechanism (M-of-N) is used to require more than one address to authorize a transaction

## • Consequences

- Benefits
  - Flexibility: Enable flexible binding of authorities based on availability
  - Lost key tolerance
    - Owning multiple addresses to reduce the risk of losing control due to key loss
    - Threshold-based authorized update
- Drawbacks
  - Pre-defined authorities: All the possible authorities need to be known in advance
  - Lost key: At least M among N private keys should be safely kept to avoid losing control
  - Cost of dynamism: Extra logic and extra addresses cost extra money as does deploying the logic for multiple authorities



Participants

# Pattern 10: Multiple Authorization 3/3

- **Related Patterns**

- *Pattern 3. Voting*
- *Pattern 11. Off-Chain Secret Enabled Dynamic Authorization*

- **Known uses**

- MultiSignature mechanism provided by Bitcoin  **bitcoin**
- Multisignature wallet, written in Solidity, running on Ethereum blockchain  **ethereum**
  - Available in the Ethereum DApp browser Mist

# Pattern 11: Dynamic Authorization 1/3

- **Summary**
  - Using a hash created off-chain to dynamically bind authority for a transaction
    - *Hashlock*
- **Context**
  - Activities might need to be authorized by multiple blockchain addresses
    - These participants are unknown when a first transaction is submitted to blockchain
- **Problem**
  - The authority who can authorize a given activity is unknown
  - No dynamic binding with an address of a participant
    - All authorities for a second transaction are required to be defined in the first transaction
- **Forces**
  - Dynamism: Dynamic binding multiple unknown authorities
  - Pre-defined authorities: All the possible authorities are required to be defined beforehand if on-chain mechanism is used ([Pattern 10](#))

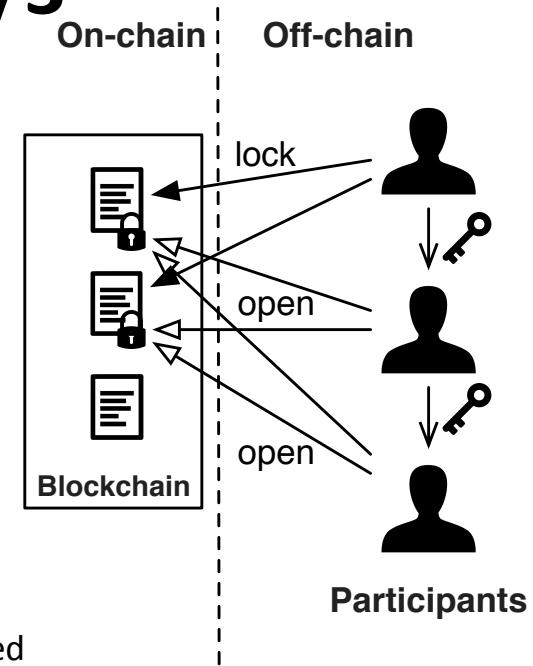
# Pattern 11: Dynamic Authorization 2/3

- **Solution**

- Off-chain secret used to enable dynamic authorization
- Transaction is “locked” by hash of an off-chain secret
  - E.g. a random string, called pre-image
- Whoever receives the secret off-chain can authorize the transaction

- **Consequences**

- Benefits
  - Dynamism: Enable dynamic binding of unknown authorities
  - Lost key tolerance: No specific key is required to authorize transaction
  - Routability: Enable multi-hop transfer since all payment transactions secured using the same secret can open at same time
  - Interoperability: Enable interaction between other systems and blockchain
- Drawbacks
  - One-off secret: Secret is not reusable after being revealed
  - Lost secret: Transaction is “locked” forever if the secret is lost



# Pattern 11: Dynamic Authorization 3/3

- **Related Patterns**

- *Pattern 10. Multiple Authorization*

- **Known uses**

- Raiden network

- Multi-hop transfer mechanism
  - *hashlocked* transactions securely router payment through a middleman



- Atomic cross-chain trading in the Bitcoin ecosystem
  - Trading Bitcoin for tokens on a Bitcoin sidechain



# Pattern 12: X-Confirmation 1/3

- **Summary**

- Waiting for enough number of blocks as confirmations to ensure that a transaction added into blockchain is immutable with high probability

- **Context**

- Proof-of-work (Nakamoto) consensus enables probabilistic immutability
  - Most recent few blocks are replaced by a competing chain fork
  - Transactions included in the discarded branches go back to the transaction pool

- **Problem**

- Fork: No certainty as to which branch will be permanently kept in blockchain

- **Forces**

- Chain fork: Occurs on a blockchain using proof-of-work consensus
- Frequency of chain fork
  - Shorter inter-block time would lead to an increased frequency of forks

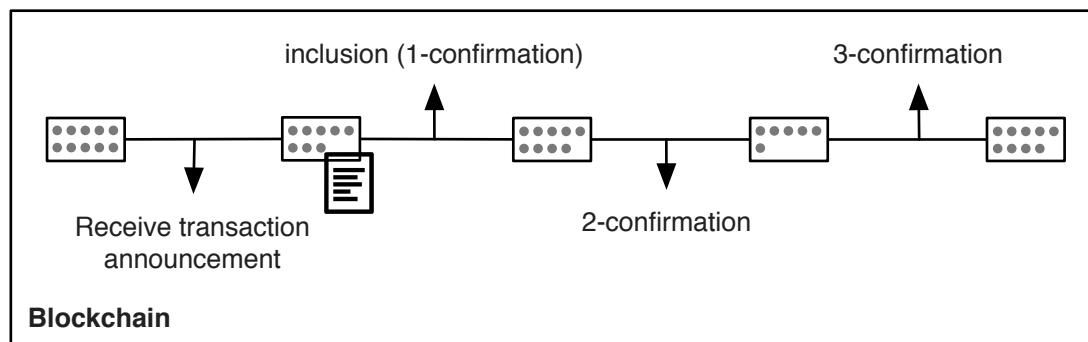
# Pattern 12: X-Confirmation 2/3

- **Solution**

- Wait for a certain number (X) of blocks to be generated after the transaction is included into one block
- Transaction is considered committed
- X is blockchain specific

- **Consequences**

- Benefits
  - Immutability: The more blocks generated after the block with the transaction, the higher probability of the immutability
- Drawbacks
  - Latency
    - Latency between submission and confirmation of a transaction is affected by consensus protocol and X
    - The larger value of X, the longer latency



# Pattern 12: X-Confirmation 3/3

- Related Patterns N/A

- Known uses

- Bitcoin uses 6-confirmation  **bitcoin**
  - An attacker is unlikely to amass more than 10% of the total amount of computing power
  - A negligible risk of less than 0.1% is acceptable
- Ethereum uses 12-confirmation  **ethereum**

# Pattern 13: Security Deposit 1/3

- **Summary**

- A user puts aside a certain amount of money, which will be paid back to the user for her honesty or given to the other parties to compensate them for the dishonesty of the user

- **Context**

- Trust is achieved from the interactions between participants within the network
- Blockchain-based applications relying on all the users to facilitate transactions

- **Problem**

- Equal rights of blockchain allows every participant the same ability to manipulate the blockchain
- How to prove honesty?

- **Forces**

- Security: Security of the system relies on the behavior of all the participants
- Incentive: Participants in a decentralized application can be incentivized to behave honestly

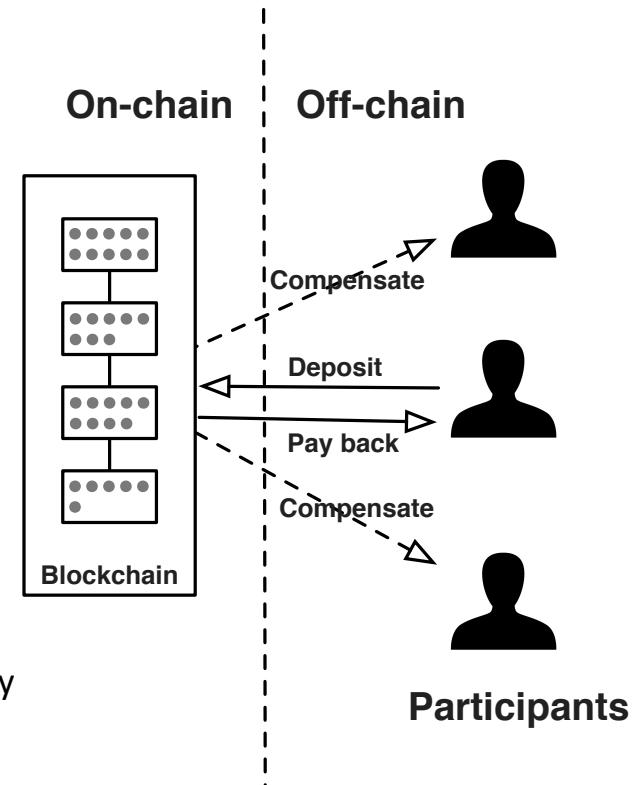
# Pattern 13: Security Deposit 2/3

## • Solution

- Participant is required to put aside amount of tokens
  - Temporarily sacrificing stake
  - Recorded on blockchain
- Paid back if the participant behaves honestly
- Or compensate others for their lost due to dishonesty of the participant

## • Consequences

- Benefits
  - Security: Deposit is paid back only if the participant behaves honestly
    - Reduce the risk of participants misbehave
- Drawbacks
  - Access
    - Security deposit is normally larger than the potential profit gain from dishonesty
    - Large security deposit restricts access to the application



# Pattern 13: Security Deposit 3/3

- **Related Patterns**

- *Pattern 18. Incentive Execution*

- **Known uses**

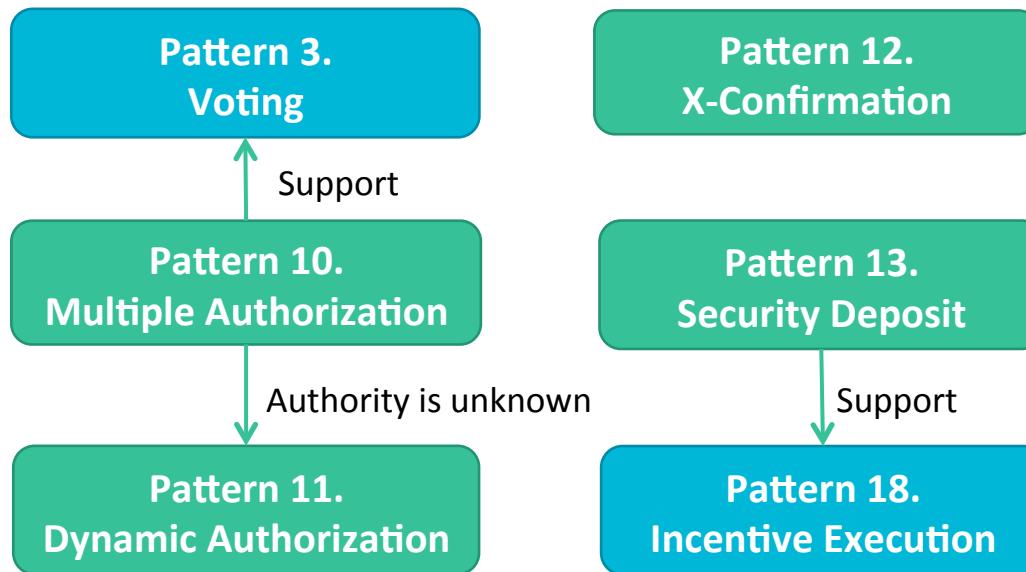
- Deposit used in Bitcoin contract
    - Party with no reputation buys deposit as a proof of trust
  - Slock.it requires servers to pay a deposit
    - Lost in case of a wrong response
    - Incentivize cross-checking as watchdog
  - Ethereum alarm clock enables scheduling of transactions for delayed execution in the future
    - *Claim window*: Deposit is required to claim a request
    - Return if the claimer fulfills the commitment to execute the request
    - Given to someone else that executes the request as an additional reward



Slock.it

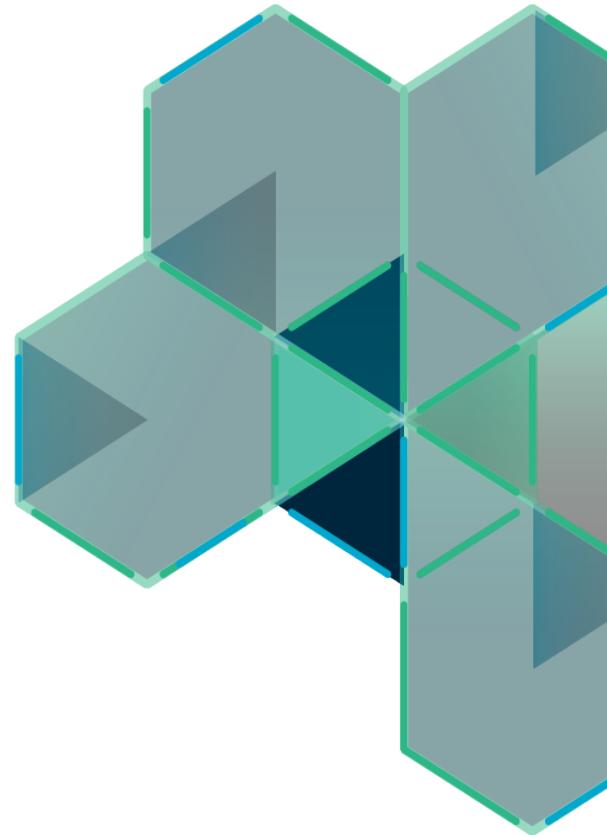


# Related Patterns



# Pattern Collection

- Interaction with External World (5)
- Data Management (4)
- Security (4)
- **Contract (5)**
- Deployment (2)



# Overview

- Smart contracts are programs running on blockchain
  - Design patterns and programming principles for conventional software are applicable
- Structural design of the smart contract has large impact on its execution cost
  - Monetary cost on public blockchain
  - Cost of data storage proportional to the size of the smart contract
    - Applicable to both public and consortium blockchain
- Structural designs of smart contracts may affect performance
  - More or less transactions may be required

# Pattern 14: Contract Registry 1/3

- **Summary**

- Before invoking it, the address of the latest version of a smart contract is located by looking up its name on a contract registry

- **Context**

- Blockchain-based applications need to be upgraded to new versions
  - Fix bugs or fulfill new requirements

- **Problem**

- Smart contract cannot be upgraded because of the immutable code stored on blockchain

- **Forces**

- Immutability
  - On-chain contract code is immutable
- Upgradability
  - Fundamental need to upgrade smart contract over time
- Human-readable contract identifier
  - Hexadecimal address is not human-readable



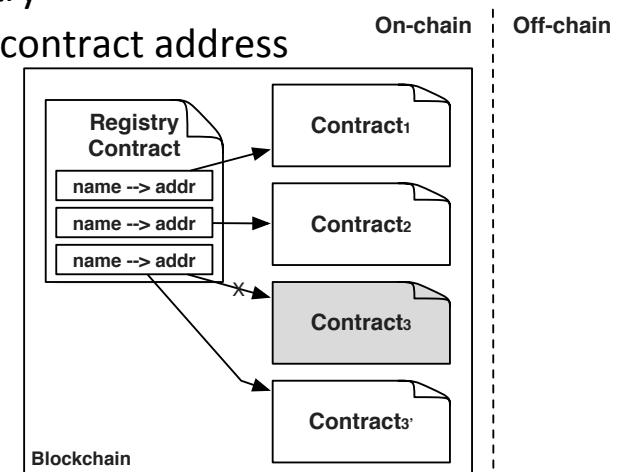
# Pattern 14: Contract Registry 2/3

## • Solution

- On-chain registry contract maintains a mapping between user-defined names and contract addresses
- Registry contract address is advertised off-chain
- Contract creator register the name and address of the new contract to the registry contract
- Invoker retrieves the latest version of the contract from the registry
- Upgrade contract by replacing the old contract address with new contract address

## • Consequences

- Benefits
  - Human-readable contract name
  - Constant contract name
  - Transparent upgradability
  - Version control: Look-up based on name and version
- Drawbacks
  - Limited upgradability: Interface cannot be modified if the functions are called by others
  - Cost: Additional cost to maintain the registry and registry look-up for latest version



# Pattern 14: Contract Registry 3/3

- **Related Patterns**

- *Pattern 15. Data Contract*
- *Pattern 16. Embedded Permission*

- **Known uses**

- ENS (Ethereum Name Service)
  - Support registering both smart contract and off-chain resources
  - Contract registry accessible to everyone
    - Blockchain-based application can maintain a registry for the application
- Regis
  - In-browser application for registries deployment and management
  - Allows user-defined key-value pairs for creating a contract registry



# Pattern 15: Data Contract 1/3

- **Summary**

- Store data in a separate smart contract

- **Context**

- The need to upgrade application and smart contract over time is ultimately necessary
- Logic and data change at different times and with different frequencies

- **Problem**

- Upgrading transactions might need a large data storage for copying data from old to new contract
- Porting data to new version might require multiple transactions
  - E.g. Ethereum gas limit prevents overly complex data migration transaction

- **Forces**

- Coupling: The data stored in a deactivated contract is not accessible through SC functions
- Upgradability: The need to upgrade application and smart contract is ultimately necessary
- Cost: Copying data from old contract to new contract has extra cost



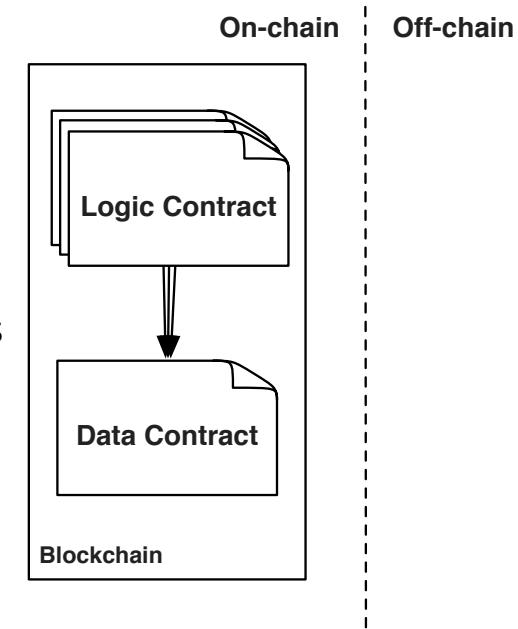
# Pattern 15: Data Contract 2/3

## • Solution

- Data store is isolated from the rest of the code
  - Avoid moving data during upgrades of smart contracts
- Strict definition or a loosely typed flat store
  - Depends on data store size and change frequency
- More generic and flexible data structure can be used by other contracts
  - Less likely to require changes: Mapping between SHA3 key and value pair

## • Consequences

- Benefits
  - Upgradability: Application logic can be upgraded without affecting the data
  - Cost: No cost for migrating data when the logic is upgraded
  - Generality: Generic separated data contract can be used by multiple smart contracts
- Drawbacks
  - Cost of generality: Generic data structure might cost more than a strictly defined data structure
  - Querying data with generic data structure is less straightforward



# Pattern 15: Data Contract 3/3

- **Related Patterns**

- *Pattern 14. Contract Registry*

- **Known uses**

- Chronobank

- Tokenize labor
    - Market for professionals to trade labor time with businesses
    - Generic data store uses a mapping of SHA3 key and value pairs



**Chronobank.io**

- Colony

- Ethereum-based platform for open organizations
    - Generic data store uses a mapping of SHA3 key and value pairs



# Pattern 16: Embedded Permission 1/3

- **Summary**

- Smart contracts use an embedded permission control to restrict access to the invocation of their functions

- **Context**

- All the smart contracts can be accessed and called by all the participants and other smart contracts
- No privileged users

- **Problem**

- Permission-less function can be triggered by unauthorized users accidentally
- Permission-less function becomes vulnerability
  - E.g., A permission-less function used by Parity multi-sig wallet caused freezing 500K Ethers

- **Forces**

- Security

- Smart contract is publically available for everyone to invoke
  - Internal logic in conventional software system is normally invisible to end users
  - API/Interface is possible to enforce access control policies



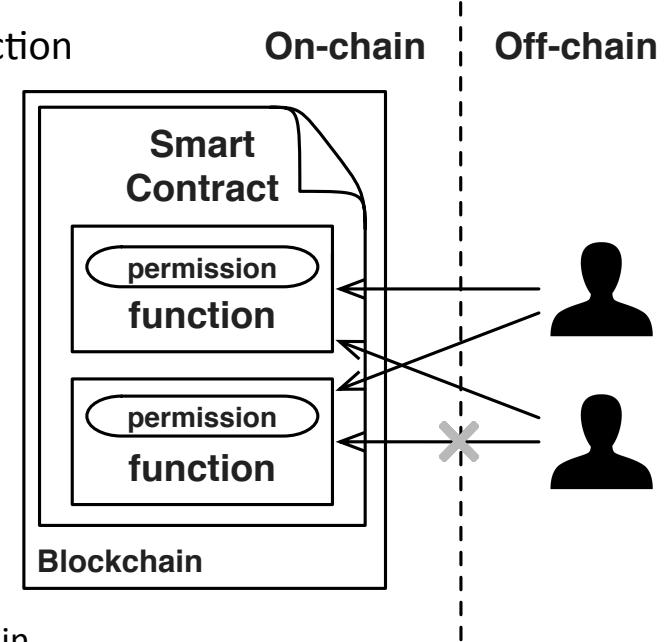
# Pattern 16: Embedded Permission 2/3

## • Solution

- Adding permission control before every smart contract function
  - Check authorization of the caller before executing the function
  - Check permission based on blockchain addresses
- Unauthorized calls are rejected

## • Consequences

- Benefits
  - Security
    - Only authorized participants can successfully call functions
  - Secure authorization
    - Authorization leverages blockchain properties
- Drawbacks
  - Cost
    - Additional deployment and run-time cost on public blockchain
  - Lack of flexibility
    - Permissions are defined before deployment and difficult to change
    - Mechanism needed to support dynamic granting and removal of permissions



# Pattern 16: Embedded Permission 3/3

- **Related Patterns**

- *Pattern 10. Multiple Authorization*
- *Pattern 11. Dynamic Authorization*

- **Known uses**

- Mortal contract on Ethereum
  - Restricts the permission of invoking the self-destruct function to the contract owner
  - *Owner* is a variable defined in the contract
- Restrict access pattern on Ethereum
  - Uses *modifier* to restrict who can call the functions
  - *Modifier* add a piece of code before the function to check certain conditions
    - Modifier makes such restrictions highly readable

```
contract owned {
    constructor() public { owner = msg.sender; }
    address owner;
}

contract mortal is owned {
    function kill() public {
        if (msg.sender == owner) selfdestruct(owner);
    }
}
```

```
modifier onlyBy(address _account)
{
    require(
        msg.sender == _account,
        "Sender not authorized."
    );
    // Do not forget the "_"! It will
    // be replaced by the actual function
    // body when the modifier is used.
    _;

    /// Make `_newOwner` the new owner of this
    /// contract.
    function changeOwner(address _newOwner)
        public
        onlyBy(owner)
    {
        owner = _newOwner;
    }
}
```

# Pattern 17: Factory Contract 1/3

- **Summary**

- On-chain template contract is used as a factory that generates contract instances from the template

- **Context**

- Application might need to use multiple instances of a standard contract with customization
- Contract instance is created by instantiating a contract template
  - E.g. business process instances
- Stored off-chain in a code repository or on-chain within its own smart contract

- **Problem**

- Off-chain contract template cannot guarantee consistency between different SC instances

- **Forces**

- Dependency management: Storing smart contract off-chain introduces more components
- Secure code sharing: Blockchain is a secure platform for sharing contract code
- Deployment: Extra effort is needed to deploy a smart contract from off-chain source code

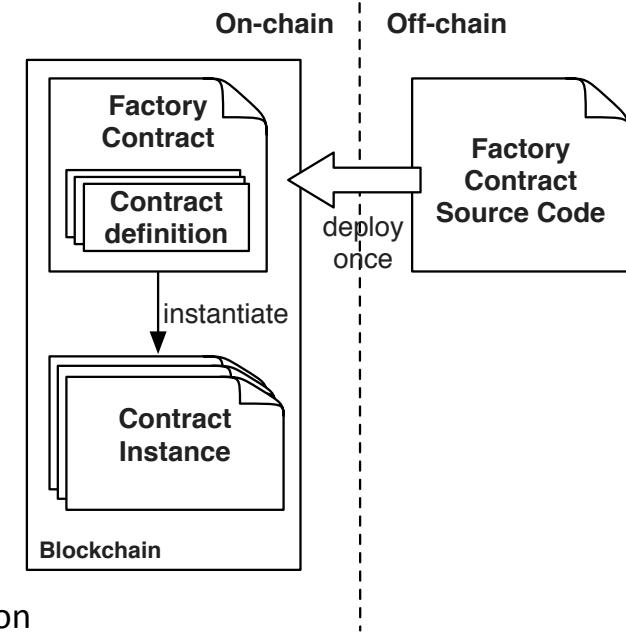
# Pattern 17: Factory Contract 2/3

## • Solution

- Smart contract are created from a contract factory on blockchain
- Factory contract is deployed once from off-chain source code
- Smart contract instances are generated by passing parameters to the contract factory to instantiate customized instances
- SC instance maintain its properties independently of the others
  - Class vs. Object

## • Consequences

- Benefits
  - Security: On-chain factory contract guarantees consistency
  - Efficiency: Smart contract instances are generated by calling a function
- Drawbacks
  - Cost on public blockchain
    - Deployment
    - Function call for smart contract instance creation



# Pattern 17: Factory Contract 3/3

- **Related Patterns**
  - *Pattern 14. Contract registry*
- **Known uses**
  - Tutorial from Ethereum developer
    - Create a contract factory
  - \*Being applied in a real-world blockchain-based health care application
  - \*\*A business process management system uses a contract factory to generate process instances



\*Zhang, P., White, J., Schmidt, D.C., Lenz, G.: Applying Software Patterns to Address Interoperability in Blockchain-based Healthcare Apps (Jun 2017)

\*\*Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain.

In: BPM. pp. 329–347. Springer, Rio de Janeiro, Brazil (Sep 2016)

# Pattern 18: Incentive Execution 1/3

- **Summary**

- Reward is provided to the caller of the contract function for invoking the execution

- **Context**

- Smart contracts are event-driven
  - Cannot execute autonomously
- Accessorial functions need to run asynchronously from regular user interaction
  - Clean up the expired records or make dividend payouts
  - Start after a time period

- **Problem**

- Users have no direct benefit from calling accessorial functions
- Extra monetary cost to call accessorial functions

- **Forces**

- Completeness: Regular services are supported by accessorial functions
- Cost: Execution of accessorial functions causes extra cost from users



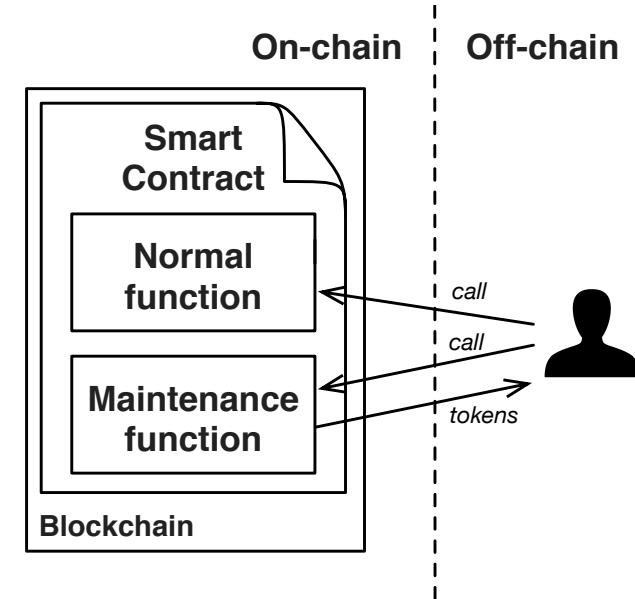
# Pattern 18: Incentive Execution 2/3

- **Solution**

- Reward the caller of a function defined in a smart contract for invoking the execution
  - E.g., sending back a percentage of payout to the caller to reimburse execution cost

- **Consequences**

- Benefits
  - Completeness: Execution of accessorial functions helps to complete the regular services
  - Cost: The caller is compensated by the reward associated with the execution
- Drawbacks
  - Unguaranteed execution
    - Execution cannot be guaranteed even with incentive
    - Embed the logic of accessorial functions into other regular functions
      - Users have to call to use the services



# Pattern 18: Incentive Execution 3/3

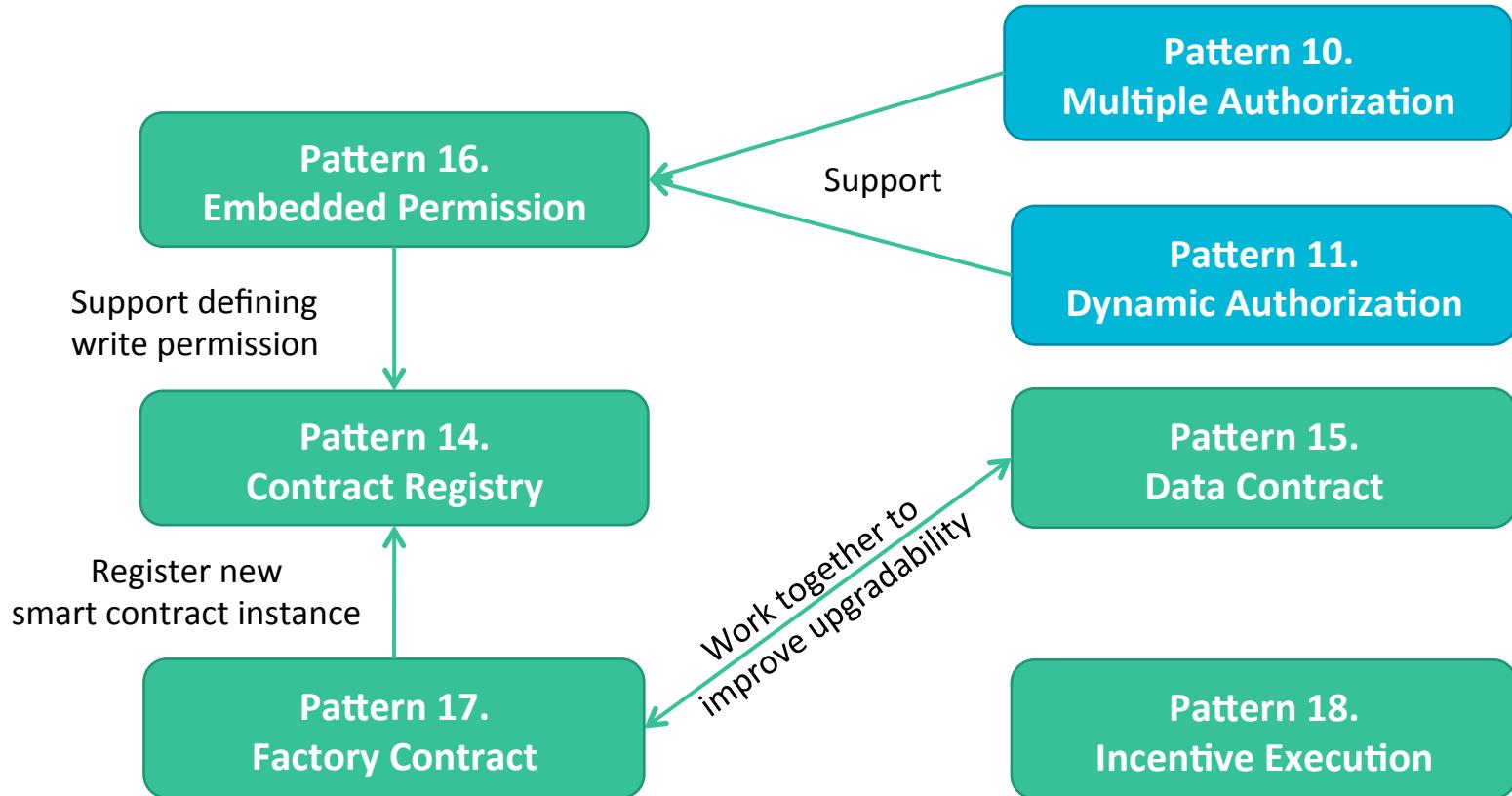
- **Related Patterns** N/A

- **Known uses**

- Regis
  - In-browser tool to create smart contract registries
  - Incentivize users to execute functions that clean up the expired records
- Ethereum alarm clock facilitates scheduling function calls for a specified block in the future
  - Provide incentive for users to execute the scheduled functions



# Related Patterns



# Pattern Collection

- Interaction with External World (5)
- Data Management (4)
- Security (4)
- Contract (5)
- **Deployment (2)**



# Pattern 19: dapp 1/3

- **Summary**

- Decentralized applications(dapps) are applications running on P2P network
- Dapps are blockchain-based websites that allow users to interact with smart contracts

- **Context**

- Users interacting with smart contracts through sending transactions to call smart contract
  - Source code of the smart contract should be open source
  - ABI (application binary interface) should be publicly accessible

- **Problem**

- Strong technical understanding of blockchain and smart contract is required
- Error-prone process with a bad user experience

- **Forces**

- Learning Curve: Read source code → understand smart contract → interact with smart contract
- Convenience: Manually generating transaction is a error-prone process



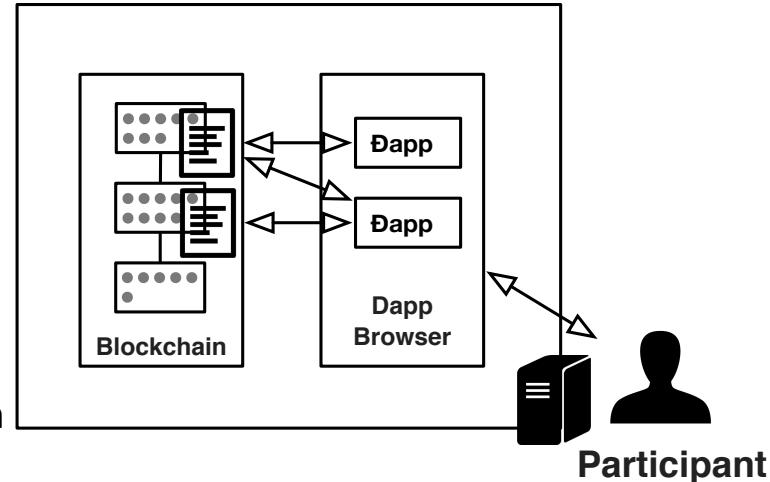
# Pattern 19: dapp 2/3

- **Solution**

- Front-end for users to interact with smart contract
- Hosted on a decentralized storage
  - E.g., IPFS
- Rendered by dapp browsers or plug-ins to web browser
  - E.g., Ethereum Mist or MetaMask
- Transactions calling SC are generated by dapp
- User verifies transactions before being sent to blockchain

- **Consequences**

- Benefits
  - Convenience: User experience of using front-end is much better than manually generating transactions
- Drawbacks
  - Trust
    - Requires certain trust in the dapp provider
    - Impact of the transaction execution is not explicit without understanding the smart contract
  - Learning Curve: Require basic technical knowledge regarding transactions and smart contracts



# Pattern 19: dapp 3/3

- Related Patterns
- *Pattern 20. Semi-dapp*

- Known uses
- *State of the dapp*
  - Directory of Dapps on Ethereum
  - 1800+ Dapps with different levels of maturity

STATE OF THE DAPPS Home All DApps Rankings Stats

Submit a DApp

Search by DApp name or tag

Showing 50 of 188 results

PLATFORM: Ethereum

CATEGORY: Finance

STATUS: All statuses

SORT BY: Newest

Platform	Name	Date	Category	Description
Ethereum	Set	NOV 27, 2018	FINANCE	The standard for tokenized baskets
Ethereum	Connect Network	NOV 27, 2018	FINANCE	Open source micropayment infrastructure
Ethereum	DIA data	NOV 27, 2018	FINANCE	Crowd-validated, open-source data.
Ethereum	Abacus	NOV 27, 2018	FINANCE	The identity and compliance protocol for permissioned tokens
Ethereum	WalletConnect	NOV 27, 2018	FINANCE	An open source standard to connect desktop DApps to mobile Wallets
Ethereum	μRaiden	NOV 25, 2018	FINANCE	A payment channel framework for fast and free off-chain ERC20 token transfers
Ethereum	Most Expensive Artwork	NOV 18, 2018	FINANCE	Crypto-artwork by Frederik F. Mettjes
Ethereum	Eholders	NOV 17, 2018	FINANCE	Tokenized smart contract high income in the medium

<https://www.stateofthedapps.com/>

# Pattern 20: Semi-dapp 1/3

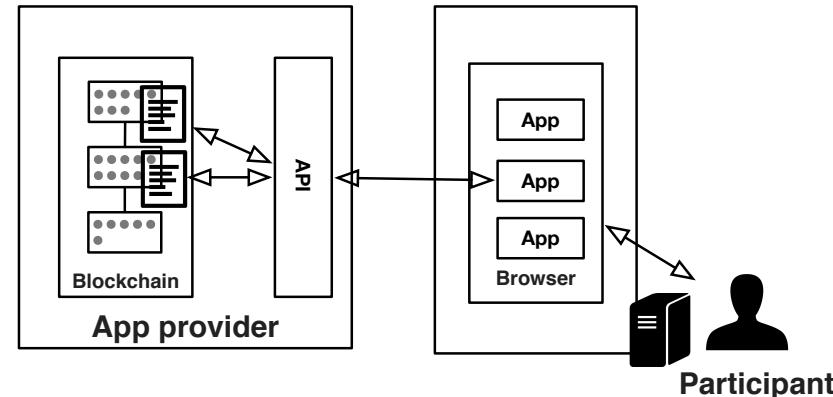
- **Summary**
  - dapp provider offers a website that can be browsed using a conventional web browser
- **Context**
  - Interacting with smart contracts through sending transactions to invoke execution
    - Source code of the smart contract should be open source
    - ABI (application binary interface) should be publicly accessible
- **Problem**
  - Even with a front-end assisting the user to interact with smart contracts
    - Basic knowledge regarding transactions and gas prices are required to use dapp and verify the transactions generated by dapp
- **Forces**
  - Learning Curve: Users need basic knowledge of smart contracts in order to use dapp
  - User Experience
    - Front-end of most dapps is quite simple
    - Exposes some technical detail of the underneath smart contract



# Pattern 20: Semi-dapp 2/3

- **Solution**

- Front-end for users to interact with smart contract
- Rendered by Web browsers
  - Underneath SCs are invisible to the users
- Website communicates with the dapp backend through RESTful API calls
- Backend is responsible for interacting with the smart contracts on behalf of the user



- **Consequences**

- Benefits
  - Convenience: User experience is as same as conventional web applications
- Drawbacks
  - Trust: Requires complete trust in the dapp provider
    - dapp provider manages the private keys of users
      - Mt. Gox lost 850,000 BTC due to a compromised internal computer
      - *Check the execution of the transactions sent by the dapp through an official blockchain explorer*

# Pattern 20: Semi-dapp 3/3

- **Related Patterns**

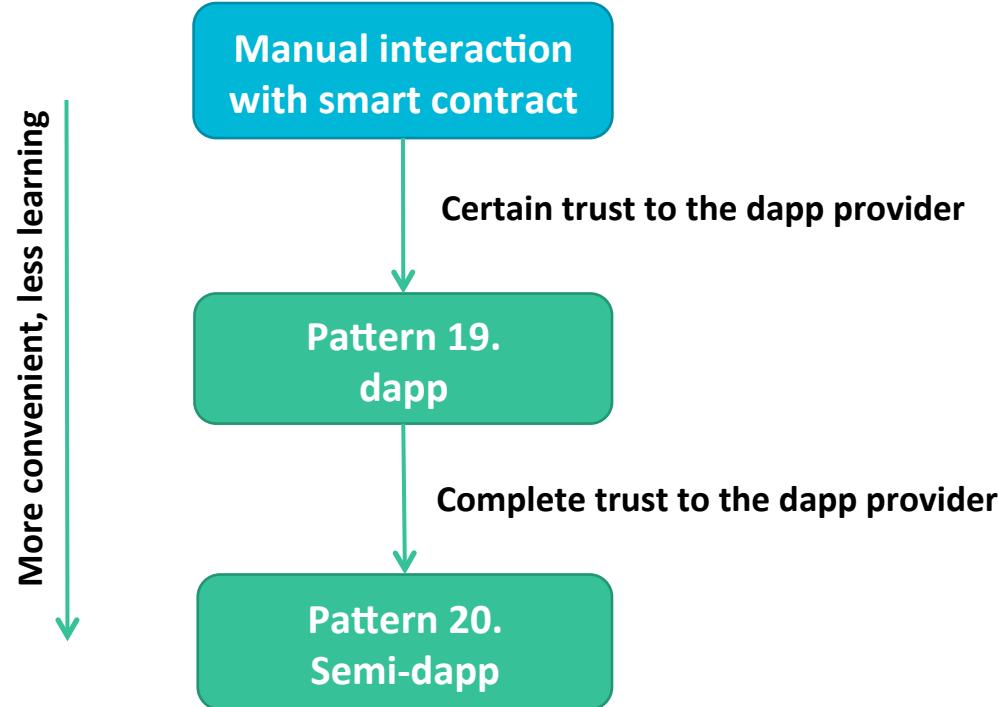
- *Pattern 19. dapp*

- **Known uses**

- Cryptocurrency Exchanges
    - Kraken
      - Francisco-based Bitcoin exchange
    - Binance
      - China-based cryptocurrency exchange



# Related Patterns



# Blockchain-based Application Pattern Collection



# Summary 1/2

## Interaction with External World

### Centralized Oracle

- Introducing external state into the blockchain environment through a centralized oracle

### Decentralized Oracles

- Introducing external state into the blockchain environment through decentralized oracles

### Voting

- A method for a group of blockchain users to make a collective decision

### Reverse Oracle

- Reverse oracle relies on smart contracts to validate requested data and check status

### Legal and smart contract pair

- A bidirectional binding between a legal agreement and the corresponding smart contract that codifies the legal agreement

## Data Management

### Encrypting On-chain Data

- Ensuring confidentiality of the data stored on blockchain by encrypting it

### Tokenisation

- Using tokens on blockchain to represent transferable digital or physical assets or services

### Off-chain Data Storage

- Using hashing to ensure the integrity of arbitrarily large datasets which may not fit directly on the blockchain

### State Channel

- Transactions that are too small in value or that require much shorter latency, are performed off-chain with periodic recording of net transaction settlements on-chain

## Security

### Multiple Authorization

- Transactions are required to be authorized by a subset of the pre-defined addresses

### Dynamic authorization

- Using a hash created off-chain to dynamically bind authority for a transaction

### X-Confirmation

- Waiting for enough number of blocks as confirmation to ensure that a transaction added into blockchain is immutable with high probability

### Security Deposit

- A deposit from a user, which will be paid back to the user for her honesty or given to others to compensate them for the dishonesty of the user



# Summary 2/2

## Structural Patterns of Contract

### Contract Registry

- The address, and the version of the smart contract is stored in a contract registry

### Embedded Permission

- Embedded permission control is used to restrict access to the invocation of the functions defined in the smart contracts

### Data Contract

- Storing data in a separate smart contract

### Factory Contract

- An on-chain template contract used as a factory that generates contract instances from the template

### Incentive Execution

- A reward to the caller of a contract function for invocation

## Deployment

### dapp

- Blockchain-based application hosted on P2P network, with a website that allows users to interact with smart contracts

### Semi-dapp

- Blockchain-based application with a website that can be browsed using a conventional web browser without any dapp plugin

# Course Outline – Next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
8th	8 Apr	Sherry Xu	Design Patterns for Blockchain Applications	7. Blockchain Patterns	
9th	15 Apr	Ingo Weber	Model-Driven Engineering	8. Model-driven Engineering for Applications on Blockchain	Assignment 2 due on 17 Apr (Wednesday)
10th	22 Apr		Easter break		
11th	29 Apr	Mark Staples + Guest Lecturer	Guest Lecture + Summary		



# THANK YOU

**Xiwei Xu** | Senior Research Scientist

Architecture & Analytics Platforms (AAP) team

t +61 2 9490 5664

e [xiwei.xu@data61.csiro.au](mailto:xiwei.xu@data61.csiro.au)

w [www.data61.csiro.au/](http://www.data61.csiro.au/)

[www.data61.csiro.au](http://www.data61.csiro.au)



# COMP6452 Lecture 9: Model-Driven Engineering

Ingo Weber | Principal Research Scientist & Team Leader

Architecture & Analytics Platforms (AAP) team

[ingo.weber@data61.csiro.au](mailto:ingo.weber@data61.csiro.au)

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

[www.data61.csiro.au](http://www.data61.csiro.au)

# Agenda:

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models

# Agenda:

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models

# Motivation for MDE

- Blockchain is still a relatively new technology with limited tooling and documentation, skill set is rare
- According to a survey by Gartner, “23% of [relevant surveyed] CIOs said that blockchain requires the newest skills for implementation at various application areas, while 18% said that blockchain technique itself is one of the most difficult.”<sup>1</sup>
- Mistakes in smart contracts have led to massive economic loss, such as the DAO exploit.
- According to an ACS / Data61 report from 2019: 14 jobs per blockchain developer<sup>2</sup>

[1] Gartner Press Release. Gartner Survey Reveals the Scarcity of Current Blockchain Deployments. (3 May 2018).  
<https://www.gartner.com/newsroom/id/3873790>

[2] Bratanova, A., Devaraj, D., Horton, J., Naughtin, C., Kloester, B., Trinh, K., Weber, I., Dawson, D. (2019) Blockchain 2030: A Look at the Future of Blockchain in Australia. CSIRO Data61: Brisbane, Australia.  
[https://www.researchgate.net/publication/332298704\\_Blockchain\\_2030\\_A\\_Look\\_at\\_the\\_Future\\_of\\_Blockchain\\_in\\_Australia](https://www.researchgate.net/publication/332298704_Blockchain_2030_A_Look_at_the_Future_of_Blockchain_in_Australia)



# Model-driven Engineering

- A methodology for using models at various levels of abstraction and for different purposes during software development
  - Often concern-specific or domain-specific modelling languages
  - “Architecture-centric MDE”: architecture is the domain
  - Text-based or graphical modelling notations can be used (i.e., not *necessarily* graphical)
  - Can cover static structures (like data models) or dynamic behaviour (like activity sequences)
- Requires you to formalize the architecture, can be an “architectural catalyst”
  - Helps to get a usable blueprint of the system
  - Makes architecture models immediately useful & motivates up-to-dateness
  - Architecture diagrams become **technical artefacts** of the system
- MDE can increase code quality and, even when starting from scratch, pay off within weeks



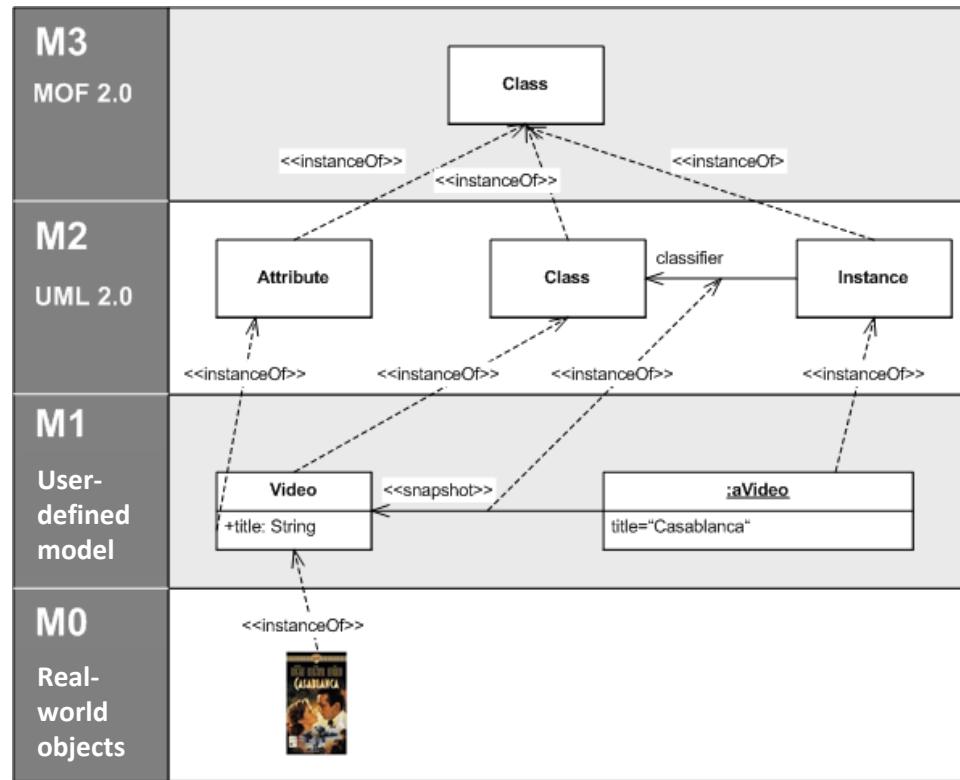
# Model-driven Engineering – modelling levels

- Models are abstractions; what to model and what to abstract depends on the purpose
  - “[A] model which took account of all the variegation of reality would be of no more use than a map at the scale of one to one.” Joan Robinson, 1962
- Low-level models: production code can be directly derived from the models
- High-level models: means of communication between business owners and developers implementing a system
- Intermediate levels can support model-based system analysis or system management tools
- Any level: can generate some of the code:
  - Code skeleton or early version of the code
  - Glue code and boilerplate code

# MDE vs. MDD vs. MDA

- Model-driven development (MDD) vs. MDE:
  - The parts of MDE that focus on code generation
  - Think of software engineering vs. software development
- Model-driven architecture (MDA) vs. MDD/MDE:
  - OMG's standard for MDE, using OMG modelling languages – primarily Unified Modeling Language (UML)
    - <https://www.omg.org/mda/>
  - Does not cover all aspects of an architecture, but architecture-centric models
  - Foundation standard: Meta-Object Facility (MOF) – see next slide
  - Other prominent concepts:
    - Platform-Independent Model (PIM)
    - Platform-Specific Model (PSM)
    - Transformations: model-to-model or model-to-code

# Meta-Object Facility (MOF)



Adapted from [https://en.wikipedia.org/wiki/Meta-Object\\_Facility](https://en.wikipedia.org/wiki/Meta-Object_Facility)

# Advantages of MDE in the blockchain context

- Code generation can implement best practices and well-tested building blocks
  - Code can adhere to blockchain “standards” (like ERC-20, ERC-721, ...)
- Improved productivity, especially for novices in a particular technology
  - However: someone should understand the code and review it
- Models can be independent of specific blockchain technologies or platforms
  - Avoid lock-in to specific blockchain technologies
- Models are often easier to understand than code – particularly useful in communicating with business partners about smart contracts
  - Facilitates building trust
  - Domain experts can understand how their ideas are represented in the system

# Agenda:

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models

# Fungible and Non-fungible Tokens

## Recap from Lecture 1

- Fungible tokens: interchangeable
  - E.g., \$2 coin, \$10 note
  - Main concern: how many?
  - Ethereum: ERC20 standard
    - [https://theethereum.wiki/w/index.php/ERC20\\_Token\\_Standard](https://theethereum.wiki/w/index.php/ERC20_Token_Standard)
    - Example: OmiseGO (OMG).

“The OmiseGO blockchain comprises a decentralized exchange, liquidity provider mechanism, clearinghouse messaging network, and asset-backed blockchain gateway. ... It uses the mechanism of a protocol token to create a proof-of-stake blockchain to enable enforcement of market activity amongst participants. Owning OMG tokens buys the right to validate this blockchain, within its consensus rules.”
- Non-fungible tokens
  - E.g., houses, cars, patents
  - Main concern: which ones?
  - Ethereum: ERC721
    - <https://github.com/ethereum/EIPs/issues/721>
    - Example: cryptokitties  
<https://www.cryptokitties.co/>



- Kitties are non-fungible, individual, and their appearance depends on the individual features

# MDE for data structures and tokens

- Approach:
  - Enter high-level information
  - Model data structure (variables, types) – not for fungible tokens
  - Model relationships to other types / tokens
  - Select features

→ Code is generated– deploy directly or customize
- Feature examples:
  - Fungible tokens:
    - Can be minted? Burnt? By whom?
  - Non-fungible tokens
    - Include standard method(s) for sale
    - One contract for all tokens or one per token?
- Code generated is compliant with standards

→ interface syntax and semantics

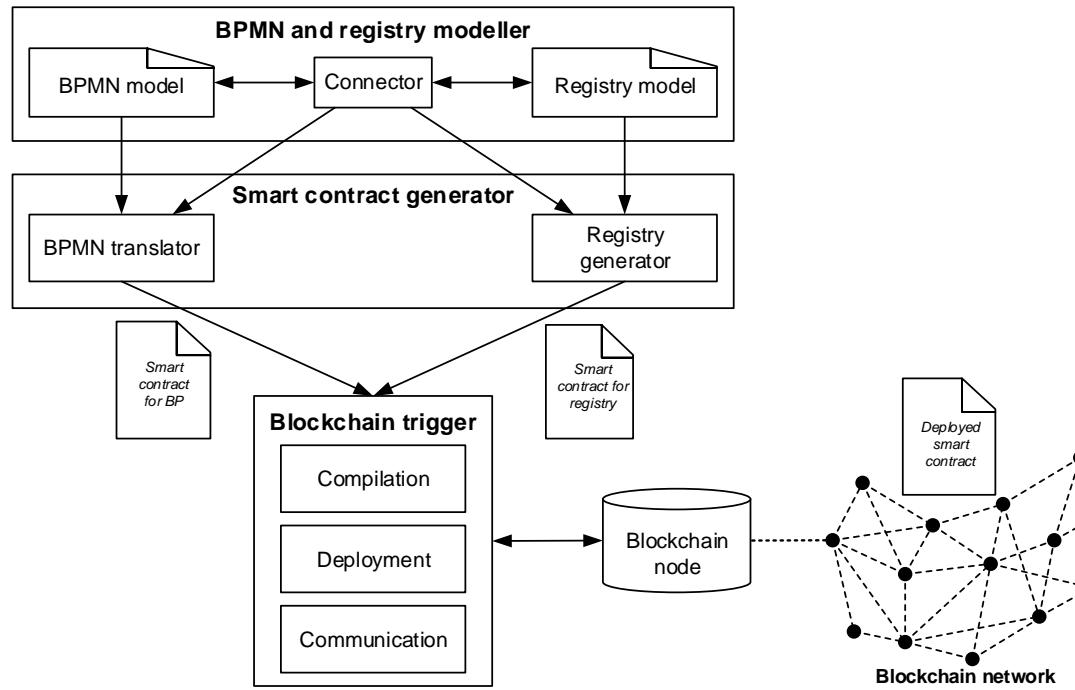


# Task

- In the next 3 minutes:
  - Pick a neighbour for later discussion
  - Open a new page / entry in your note pad / note-taking app
  - Think about what data to model for students of this class
  - Write down your top 4-10 data fields
  - Discuss with your neighbour
  - Think about any class-specific functions that the tool won't support & write it down

# Data61 tool: Lorikeet

- Lorikeet: automatic generate smart contracts from BPMN models/registry data schema



# Design time for data / non-fungible tokens

The screenshot shows the Lorikeet platform interface for designing a blockchain registry. On the left, the 'Edit Registry Design' tab is active, displaying fields for the digital asset name (COMP6452-students), registry type (Single selected), record ID data type (address), and record attributes (StudentName, uint attribute z-ID). On the right, the 'Smart Contract Output' tab shows the generated Solidity code for the registry contract.

```
pragma solidity ^0.4.21;

// COMP6452-students registry smart contract.
contract COMP6452 - studentsRegistry {

    enum RecordStates {
        NON_EXISTING,
        ACTIVE
    }

    // Data structure containing basic fields/attributes of a registry record.
    struct RecordAttributes {
        // Define record attributes here...
        uint z - ID;
    }

    // Data structure representing a single registry record.
    struct Record {
        // Record owner can be an external Ethereum account,
        // or a smart contract representing an user group or Organisation.
        address owner;

        RecordStates state;
        RecordAttributes attrs;
    }

    mapping(address => Record) records;
    address[] all_record_ids;

    // Admin of registry.
    address public registry_admin;

    // Registry constructor.
    function COMP6452 - studentsRegistry() public {
        // The user deploying the registry contract will be the registry admin.
        registry_admin = msg.sender;
    }

    // ====== CONTRACT EVENTS ======
}
```

# Design time for fungible tokens

The screenshot shows the Lorikeet blockchain development interface. On the left, the "Edit ERC20 Token Design" panel contains fields for Name (COMP6452-Token), Symbol (C6452), and Decimals (2). It also includes sections for mintability (Not Mintable selected) and burnability (Not Burnable selected). Below these, an "Allocation" section shows an account (e.g. 0x1434e8f21A0d8A66024E06a45637664b9349A691) with an allocation of 0. A "Save & Close" button is at the top right of this panel.

On the right, the "Smart Contract Output" panel displays the generated Solidity code:

```
pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
library SafeMath {

    /**
     * @dev Multiplies two numbers, reverts on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns(uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
     * @dev Integer division of two numbers truncating the quotient, reverts on division by
     */
    function div(uint256 a, uint256 b) internal pure returns(uint256) {
        require(b > 0); // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend)
     */
    function sub(uint256 a, uint256 b) internal pure returns(uint256) {
        ...
    }
}
```

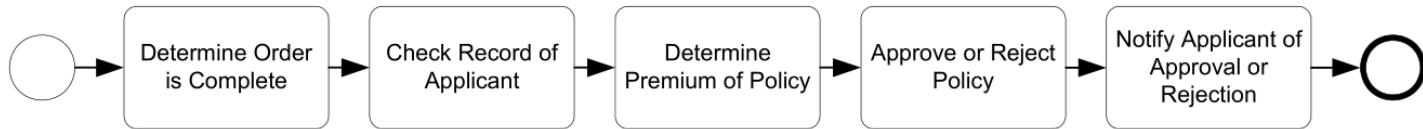
Build 61. Commit SHA: 9172290fb80f4ba57d521932bbfae1730dbbcde

# Agenda:

- Model-Driven Engineering basics
- MDE for data and token model
- **Business Process Model and Notation (BPMN)**
- MDE for process models

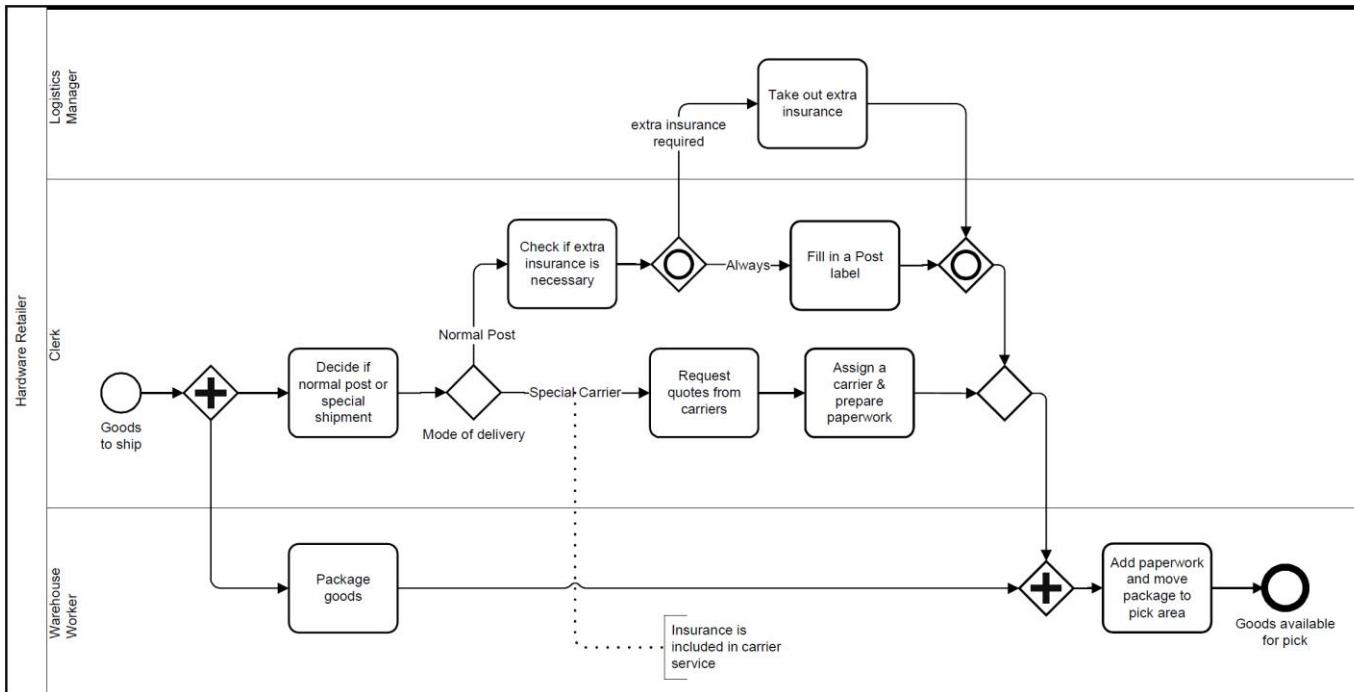
# Business Processing Modelling: basic terminology

- BPMN example: insurance application processing



- Tasks: an activity that cannot be subdivided (unit of work)
- Sequence: some tasks need to be performed in a strict order
- Choice: certain tasks do not always need to be carried out
- Parallel: some tasks can be performed in parallel
- Synchronisation: some tasks need to wait for the result of previous tasks
- Iteration: some tasks need to be repeated

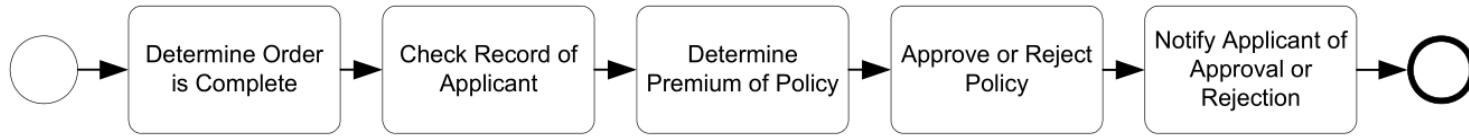
# BPMN example: shipment process of a hardware retailer



Source: BPMN 2.0 by Example, OMG Document Number dtc/2010-06-02

# Business Process Model and Notation (BPMN): Motivation

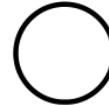
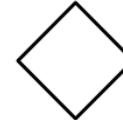
- Focus of earlier workflow languages: clear execution semantics (e.g., PetriNets), expressivity (e.g., YAWL), interoperability (e.g., BPEL)
- Separate set of languages: conceptual process modelling notations without clear execution semantics, etc.
- Before BPMN, no notation existed that appealed to business users and has (somewhat) clear execution semantics
  - BPMN addresses this "niche"
- OMG Spec 2.0.2 in 2013: <http://www.omg.org/spec/BPMN/2.0.2/PDF>
  - From p.56 on, the elements are explained
  - (v1.0 released in 2006, v2.0 in 2011)
- BPMN example: private process



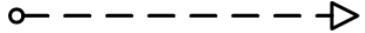
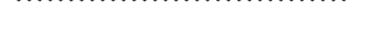
# BPMN diagram elements

- Flow Objects
  - Events
  - Activities
  - Gateways
- Connecting Objects
  - Sequence Flow
  - Message Flow
  - Association
- Swimlanes
  - Pools
  - Lanes
- Artefacts
  - Data Object
  - Group
  - Annotation

# Core flow objects (from BPMN Spec)

Element	Description	Notation
Event	An Event is something that “happens” during the course of a Process (see page 235) or a Choreography (see page 339). These Events affect the flow of the model and usually have a cause ( <i>trigger</i> ) or an impact ( <i>result</i> ). Events are circles with open centers to allow internal markers to differentiate different <i>triggers</i> or <i>results</i> . There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.	
Activity	An Activity is a generic term for work that company performs (see page 149) in a Process. An Activity can be atomic or non-atomic (compound). The types of Activities that are a part of a Process Model are: Sub-Process and Task, which are rounded rectangles. Activities are used in both standard Processes and in Choreographies.	
Gateway	A Gateway is used to control the divergence and convergence of Sequence Flows in a Process (see page 147) and in a Choreography (see page 335). Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control.	

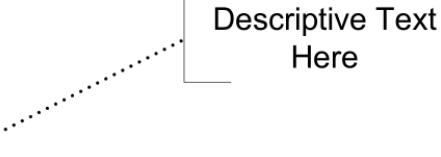
# Connecting objects (from BPMN Spec)

Sequence Flow	A Sequence Flow is used to show the order that Activities will be performed in a Process (see page 95) and in a Choreography (see page 320).	
Message Flow	A Message Flow is used to show the flow of Messages between two <i>Participants</i> that are prepared to send and receive them (see page 113). In BPMN, two separate Pools in a Collaboration Diagram will represent the two <i>Participants</i> (e.g., PartnerEntities and/or PartnerRoles).	
Association	An Association is used to link information and Artifacts with BPMN graphical elements (see page 65). Text Annotations (see page 69) and other Artifacts (see page 64) can be Associated with the graphical elements. An arrowhead on the Association indicates a direction of flow (e.g., data), when appropriate.	

# Swimlanes (from BPMN Spec)

Pool	A Pool is the graphical representation of a <i>Participant</i> in a Collaboration (see page 113). It also acts as a “swimlane” and a graphical container for partitioning a set of Activities from other Pools, usually in the context of B2B situations. A Pool MAY have internal details, in the form of the Process that will be executed. Or a Pool MAY have no internal details, i.e., it can be a “black box.”	<table border="1"><tr><td>Name</td><td></td></tr></table>	Name	
Name				
Lane	A Lane is a sub-partition within a Process, sometimes within a Pool, and will extend the entire length of the Process, either vertically or horizontally (see on page 304). Lanes are used to organize and categorize Activities.	<table border="1"><tr><td>Name</td><td>Name</td></tr></table>	Name	Name
Name	Name			

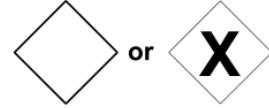
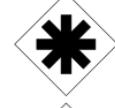
# Artefacts (from BPMN Spec)

Data Object	Data Objects provide information about what Activities require to be performed and/or what they produce (see page 204). Data Objects can represent a singular object or a collection of objects. Data Input and Data Output provide the same information for Processes.	
Message	A Message is used to depict the contents of a communication between two <i>Participants</i> (as defined by a business PartnerRole or a business PartnerEntity—see on page 91).	
Group (a box around a group of objects within the same category)	A Group is a grouping of graphical elements that are within the same Category (see page 68). This type of grouping does not affect the Sequence Flows within the Group. The Category name appears on the diagram as the group label. Categories can be used for documentation or analysis purposes. Groups are one way in which Categories of objects can be visually displayed on the diagram.	
Text Annotation (attached with an Association)	Text Annotations are a mechanism for a modeler to provide additional text information for the reader of a BPMN Diagram (see page 69).	 Descriptive Text Here

# Start, intermediate, end events (from BPMN Spec)

Start	As the name implies, the Start Event indicates where a particular Process (see page 235) or Choreography (see page 339) will start.	Start
Intermediate	Intermediate Events occur between a Start Event and an End Event. They will affect the flow of the Process (see page 237) or Choreography (see page 340), but will not start or (directly) terminate the Process.	Intermediate
End	As the name implies, the End Event indicates where a Process (see page 245) or Choreography (see page 343) will end.	End

# Gateways (from BPMN Spec)

Gateway Control Types	<p>Icons within the diamond shape of the Gateway will indicate the type of flow control behavior. The types of control include:</p> <ul style="list-style-type: none"><li>• Exclusive decision and merging. Both Exclusive (see page 286) and Event-Based (see page 296) perform exclusive decisions and merging. Exclusive can be shown with or without the "X" marker.</li><li>• Event-Based and Parallel Event-based gateways can start a new instance of the Process.</li><li>• Inclusive Gateway decision and merging (see page 291).</li><li>• Complex Gateway -- complex conditions and situations (e.g., 3 out of 5; page 294).</li><li>• Parallel Gateway forking and joining (see page 292).</li></ul> <p>Each type of control affects both the incoming and outgoing flow.</p>	<p><b>Exclusive</b></p>  <p>or</p>  <p><b>Event-Based</b></p>   <p><b>Parallel Event-Based</b></p>  <p><b>Inclusive</b></p>  <p><b>Complex</b></p>  <p><b>Parallel</b></p> 
-----------------------	---	---

# Task

- In the next 5 minutes, model the process of taking this class:
  - What are the activities?
  - In which order do they occur?
  - What are decision points in the process, what can happen?
  - Draw a model & discuss with your neighbour

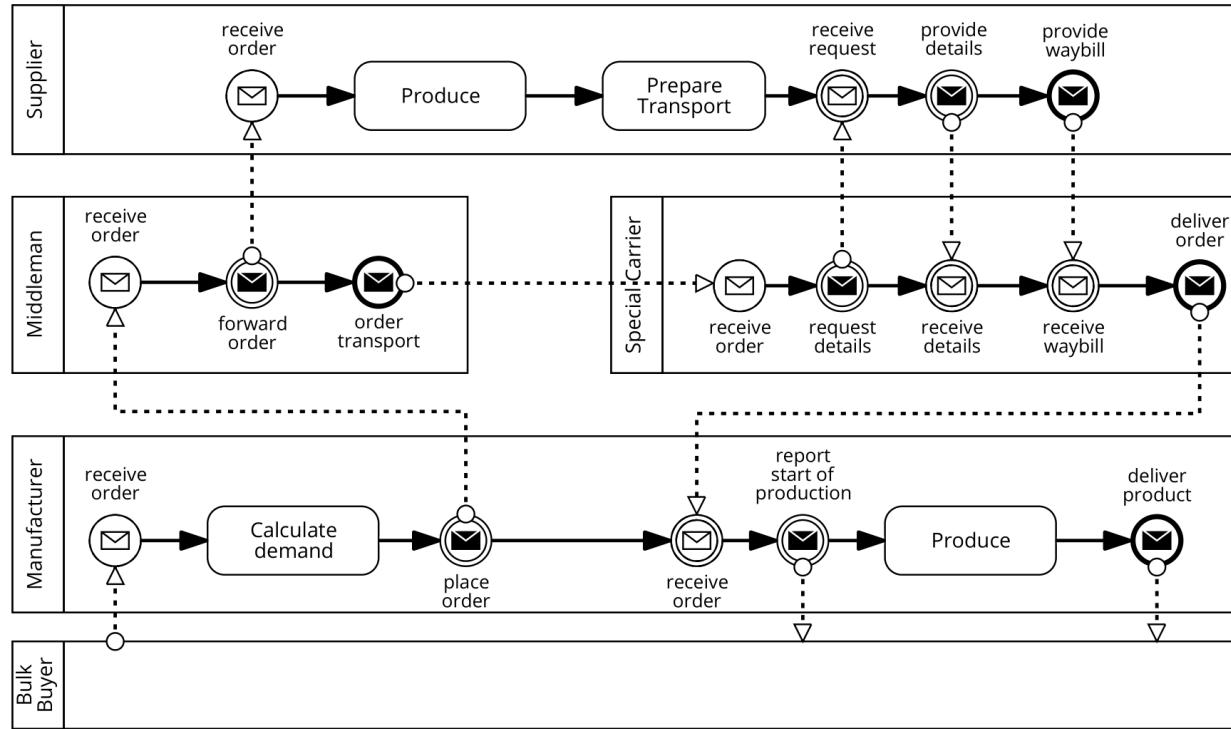
# Agenda:

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models

# MDE for Processes – Motivation

- Integration of business processes across organizations:  
a key driver of productivity gains
- Collaborative process execution
  - Doable when there is trust – supply chains can be tightly integrated
  - Problematic when involved organizations have a **lack of trust** in each other  
→ if 3+ parties should collaborate, where to execute the process that ties them together?
  - Common situation in “coopetition”

# Motivation: example



## Issues:

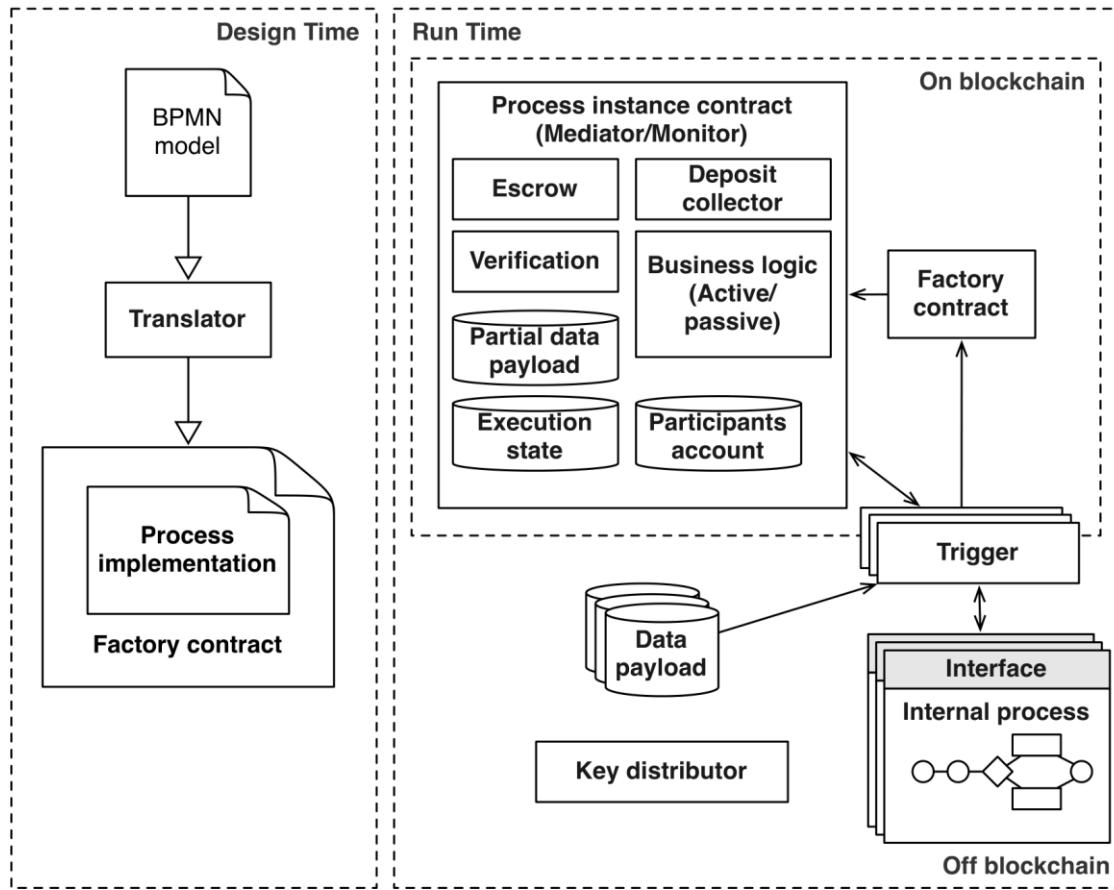
- Knowing the status, tracking correct execution
- Handling payments
- Resolving conflicts

# Approach in a nutshell

- Goal: implement collaborative business processes as smart contracts
  - Translate (enriched) BPMN to smart contract code
  - Triggers act as bridge between Enterprise world and blockchain
  - Smart contract provides:
    - Independent, global process monitoring
    - Conformance checking and process enforcement: only expected messages are accepted, only from the respective role
    - Automatic payments & escrow
    - Data transformation
    - Encryption
- Processes can make use of data / token contracts
  - Process activity to hand over title to a car / shipment / grain / ..., e.g., in exchange for fungible tokens



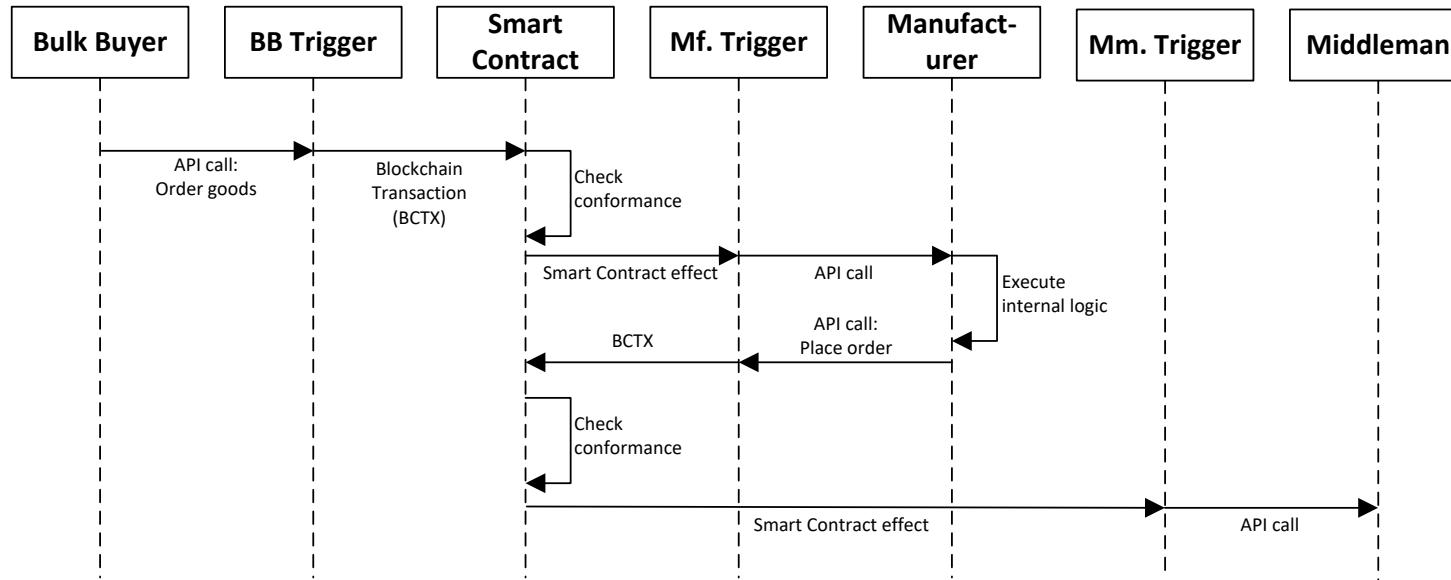
# Architecture overview



# Runtime

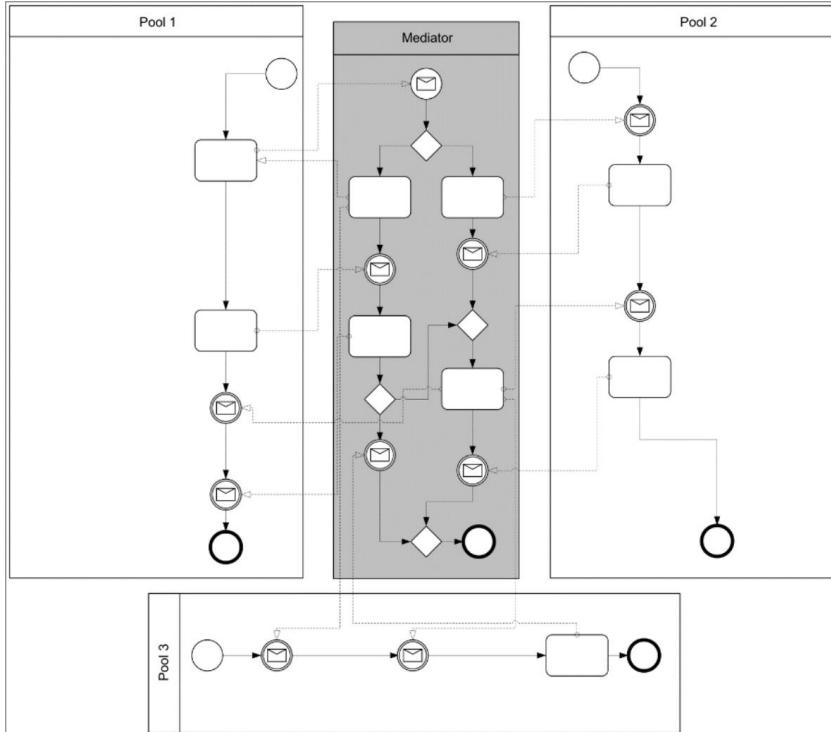
- Instantiation:
  - New *instance contract* per process instance
  - Assign accounts to roles during initialization
  - Exchange keys and create secret key for the instance
- Messaging:
  - Instead of sending direct WS calls: send through triggers & smart contract
  - Instance contract handles:
    - Global monitoring
    - Conformance checking
    - Automated payments\*
    - Data transformation\*

# Runtime – UML Sequence Diagram

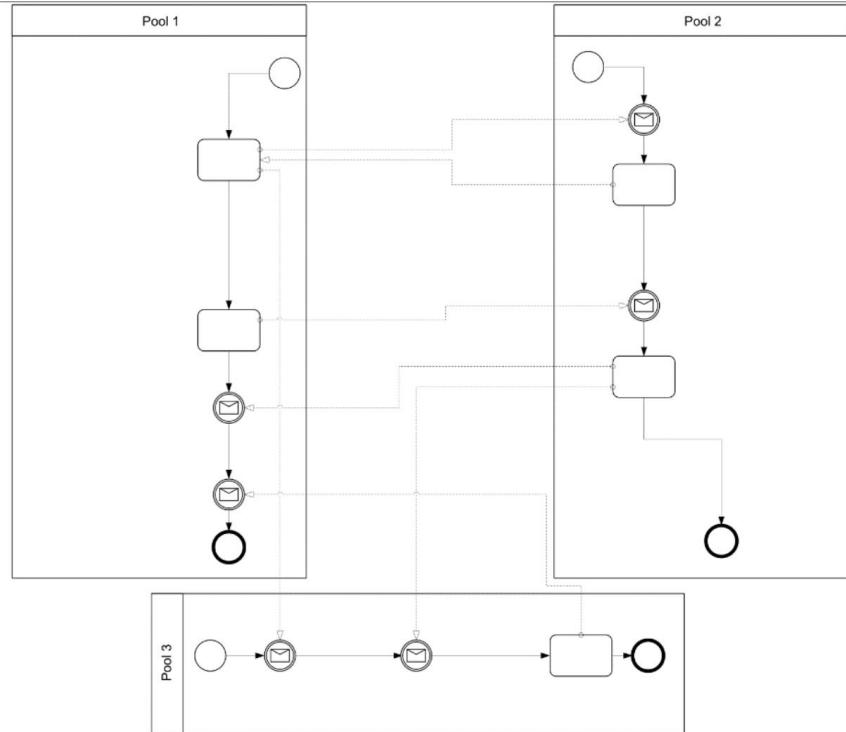


# Collaborative processes: variants

Mediator (orchestration)

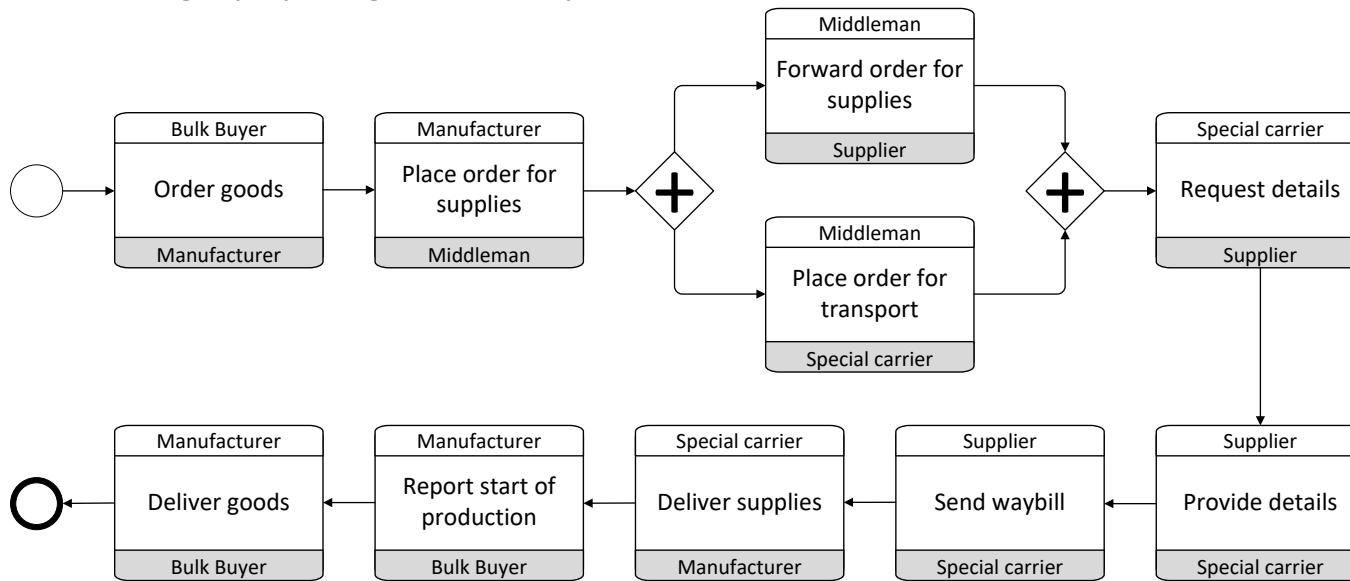


Choreography → C-Monitor

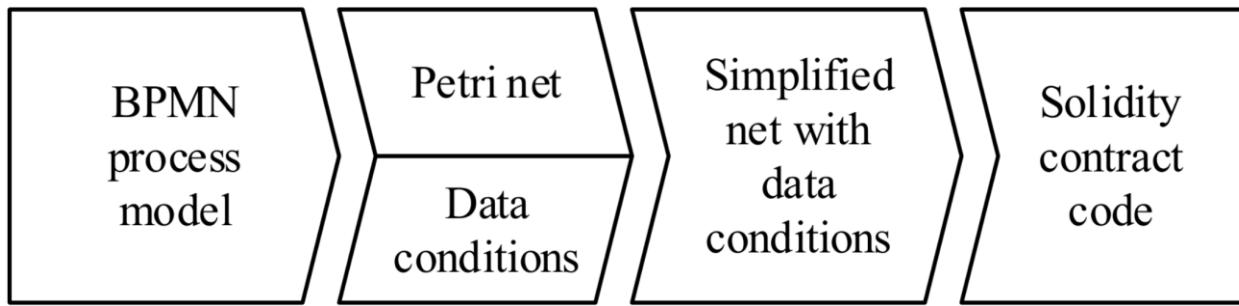


# Translator

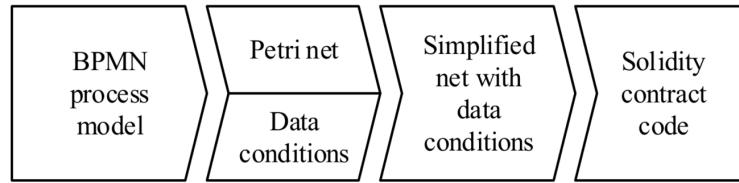
- Translate subset of BPMN elements to Solidity
  - BPMN Choreography diagrams or regular BPMN models with pools
  - BPMN Choreography diagram example:



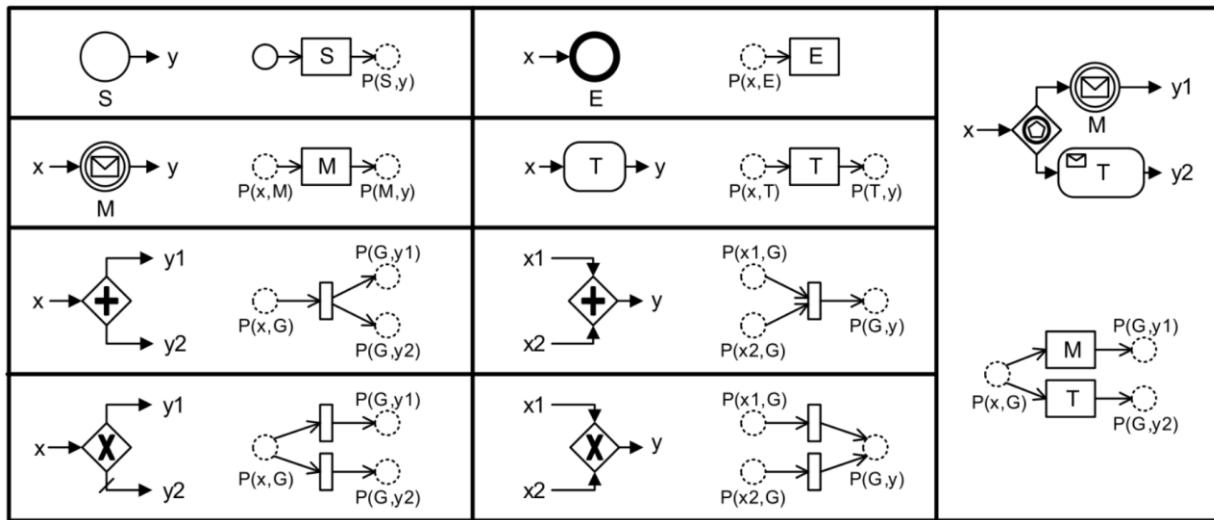
# Translator



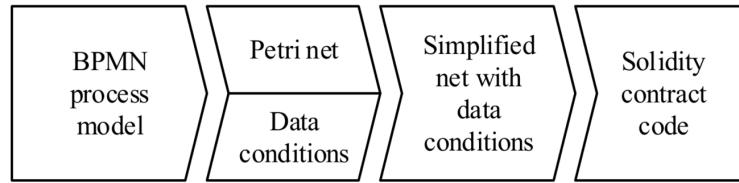
# Translator



## 1. Translate BPMN control flow to WFnet (proven to be safe)

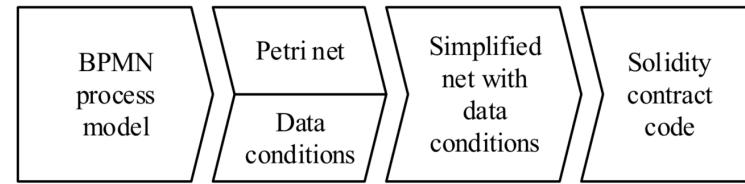


# Translator

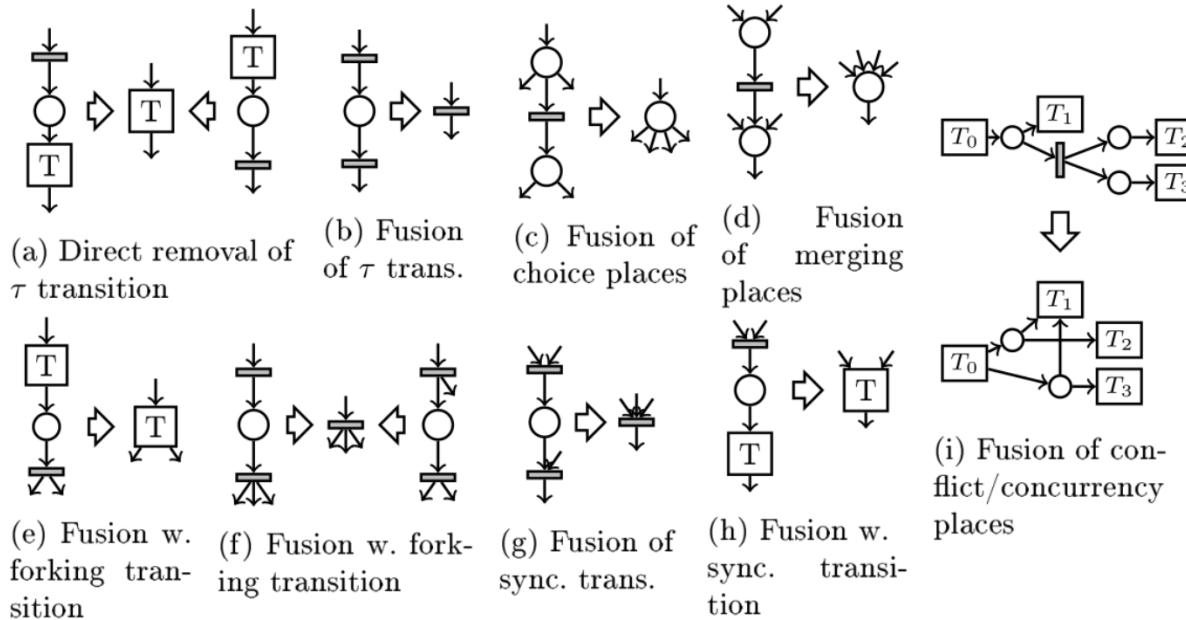


1. Translate BPMN control flow to WFnet (proven to be safe)
2. Capture dataflow and conditions

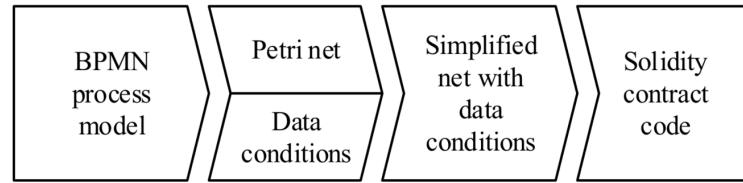
# Translator



1. Translate BPMN control flow to WFnet (proven to be safe)
2. Capture dataflow and conditions
3. Reduce WFnet and annotate dataflow

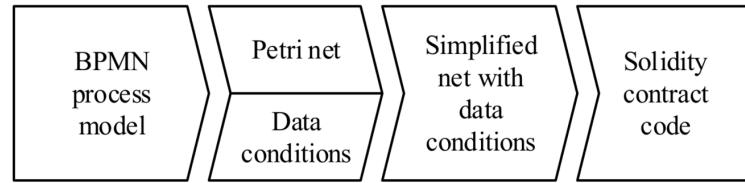


# Translator



1. Translate BPMN control flow to WFnet (proven to be safe)
2. Capture dataflow and conditions
3. Reduce WFnet and annotate dataflow
4. Translate into Solidity code
  - Status of the process is kept in a bit vector
  - Updates are bit-wise operations

# Translator



1. Translate BPMN control flow to WFnet (proven to be safe)

2. Capture data

3. Reduce WFne

4. Translate into

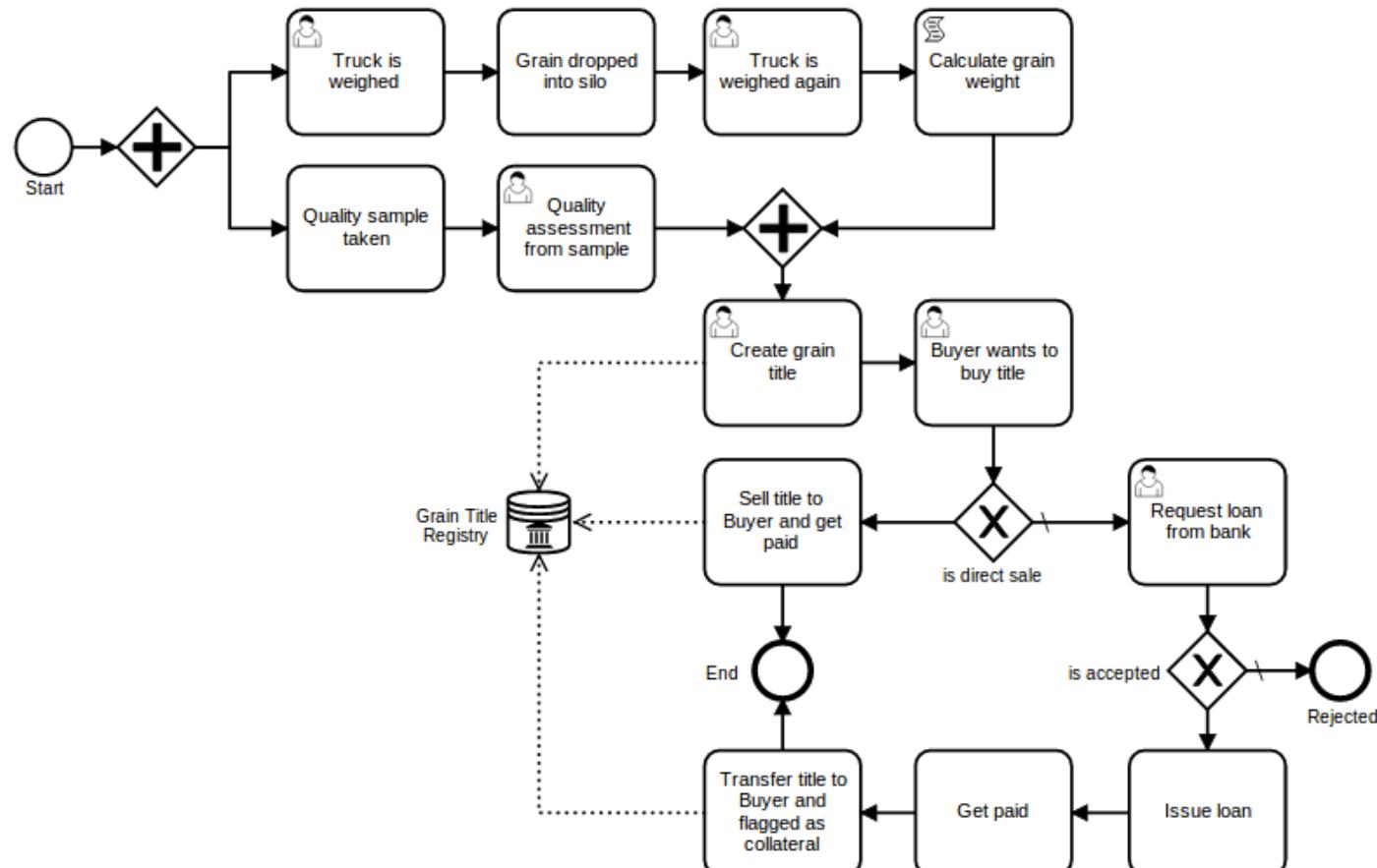
- Status of the
- Updates are b

```
1 contract BPMNContract {
2     uint marking = 1;
3     uint predicates = 0;
4
5     function CheckApplication( - input params - ) returns (bool) {
6         if (marking & 2 == 2) { // is there a token in place p1?
7             // Task B's script goes here, e.g. copy value of input params to contract variables
8             uint tmpPreds = 0;
9             if ( - eval P - ) tmpPreds |= 1; // is loan application complete?
10            if ( - eval Q - ) tmpPreds |= 2; // is the property pledged?
11            step (
12                marking & uint(~2) | 12, // New marking
13                predicates & uint(~3) | tmpPreds // New evaluation for "predicates"
14            );
15            return true;
16        }
17        return false;
18    }
```

Bit vector update: set pos 1 to "0"

set pos 2 and 3 to "1"

# Combining process and data/token models



# Tooling: Design time

DATA 61 | LORIKEET | Home | Design | Manage | Welcome 0x180d34b876DAa90057B2Ec345E82E2B1E9a4A082 | Log Out

Edit BPMN Design

Business Process Name: GrainSupplyChain

Use Petri-Net Method:

Save Model | Source XML

BPMN Diagram:

```
graph LR; Start((Start)) --> TW1[Truck is weighed]; TW1 --> GDS[Grain dropped into silo]; GDS --> TW2[Truck is weighed again]; TW2 --> QTS[Quality sample taken]; QTS --> QAS[Quality assessment from sample]; QAS --> P1{+}; P1 --> CGT[Create grain title]; CGT --> BBWBT[Buyer wants to buy title]; BBWBT --> SSBG[Sell title to Buyer and get paid]; SSBG --> End((End)); SSBG -- "is direct sale" --> RLB[Request loan from bank]; RLB -- "is accepted" --> End; RLB -- "is rejected" --> P2{X}; P2 --> GTR[Grain Title Registry]; GTR --> End;
```

RegistryReference\_03z5ksr

General

Id: RegistryReference\_03z5ksr

Smart Contract Output

Solidity Code:

```
// ----- REGISTRY INTERFACES
contract GrainTitleRegistry {
    function record_create(uint weight) returns(uint record_id);

    function record_transfer_ownership(uint record_id, address buyer_address, bool is_collateral);
}

// ----- PROCESS MONITOR CONTRACT

contract ProcessMonitor {
    //uint preconditions = 0x800;

    function getPreconditions(uint instanceID) internal returns(uint);
    function setPreconditions(uint instanceID, uint preconditions) internal;
    event taskCompleted(uint indexed instanceID, string taskName);

    // -----
    //bool isLoanAccepted;
    //bool isDirectSale;
    //bytes32 titleID;
    //address buyerAddress;
    //

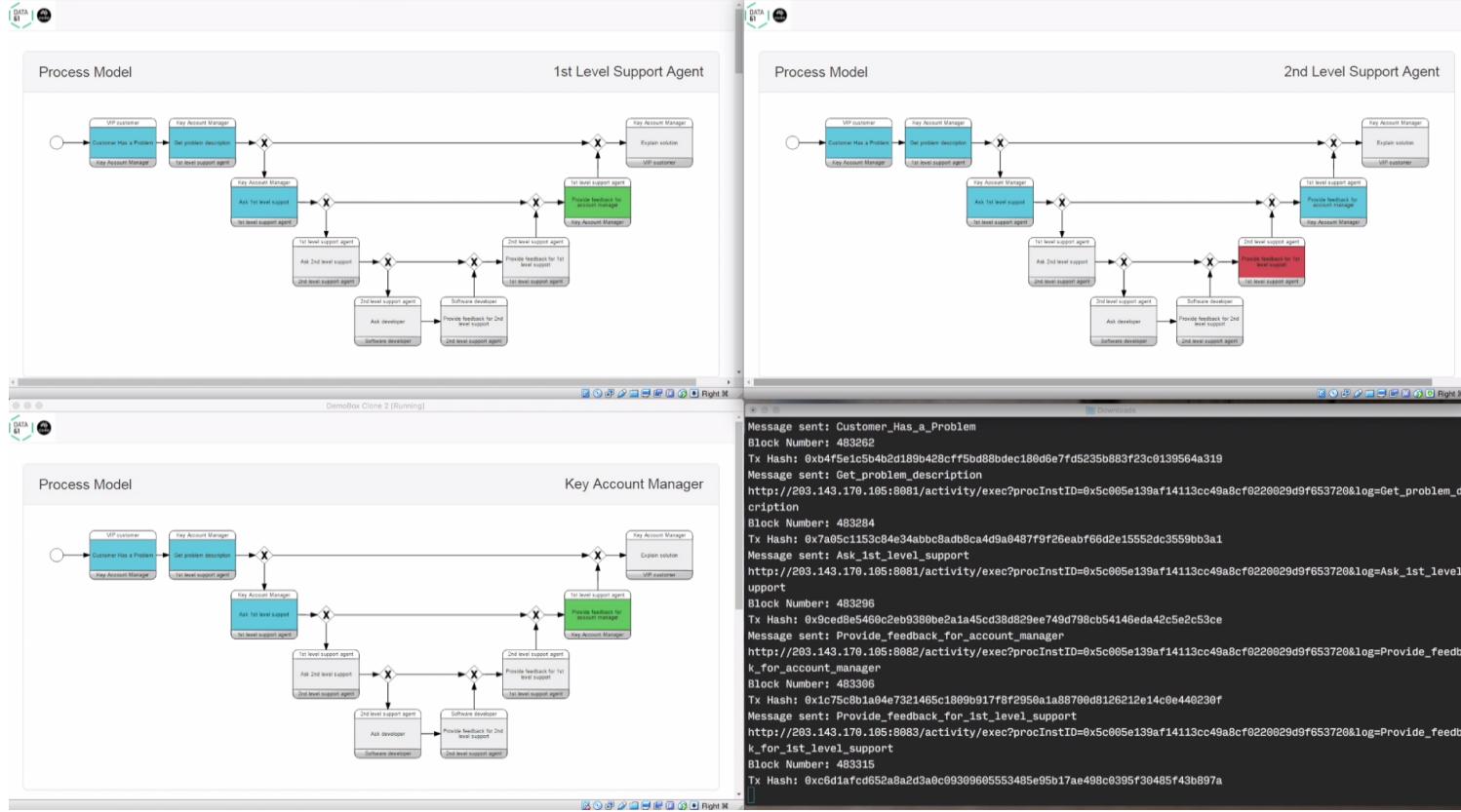
    // ----- REGISTRY CONTRACT ADDRESSES
    address addressForGrainTitleRegistry = 0x1D6fd252049f869349C4df4E2df3E17c8539B70;
    //

    function ProcessMonitor() {
        //
        //
        //
    }

    function Issue_loan(uint256 instanceID) returns(bool) {
        uint preconditions = getPreconditions(instanceID);

        if ((preconditions & (0x2 | 0x1000) == (0x2 | 0x1000))) {
            step(instanceID, preconditions & uint(-0x2) | 0x40);
            taskCompleted(instanceID, "Issue_loan");
            return true;
        }
    }
}
```

# Tooling: Runtime



# Summary:

- Model-Driven Engineering basics
- MDE for data and token model
- Business Process Model and Notation (BPMN)
- MDE for process models

# MDE & related tools for blockchain

- Lorikeet (and its predecessor Regerator & BPM work)
  - Tools are not publicly available, but lots of information/papers on our website:  
<https://research.csiro.au/data61/blockchain/>
- Caterpillar – open-source BPMN execution engine (Uni Tartu & Data61)
  - <https://github.com/orlenyslp/Caterpillar>
- Regis – registry generator
  - <https://regis.nu/>
- Hyperledger Composer – for HLF business networks
  - <https://www.hyperledger.org/projects/composer>
- Other organisations (including Consensys) are working on tools, but most are not publicly available as yet

# Course Outline – next two weeks

Week	Date	Lecturer	Lecture Topic	Relevant Book Chapters	Notes
10th	22 Apr			Easter Holiday	
11th	29 Apr	Guest Lecturer + Mark Staples	Guest Lecture and Summary	Case study chapters Foreword, Epilogue	



# End of Lecture / Consultation

Ingo Weber | Principal Research Scientist & Team Leader

Architecture & Analytics Platforms (AAP) team

[ingo.weber@data61.csiro.au](mailto:ingo.weber@data61.csiro.au)

Conj. Assoc. Professor, UNSW Australia | Adj. Assoc. Professor, Swinburne University

[www.data61.csiro.au](http://www.data61.csiro.au)