An introduction of BigchainDB

- Background
- Preliminaries
- Description
- Implementation Details
- Transaction Latency
- Private vs. Public

Background

- The development of Bitcoin has triggered a new wave of decentralization in computing. Bitcoin has a novel set of benefits:
  - **decentralized control**: “no one” owns or controls the network;
  - **Immutability**: written data is tamper-resistant (“forever”);
  - The ability to create & transfer assets on the network, without reliance on a central entity.

- As people learned more about the underlying technology of Bitcoin– **blockchain**, they extended the scope of the technology itself (e.g. smart contracts), as well as applications (e.g. intellectual property).

Background

- “blockchain” technologies are being re-framed and refactored into building blocks at four levels of the stack:
  - Applications;
  - Decentralized computing platforms (“blockchain platforms”);
  - Decentralized processing (“smart contracts”) and decentralized storage (file systems, databases), and decentralized communication;
  - Cryptographic primitives, consensus protocols, and other algorithms.

Background

- Can we frame a traditional blockchain (e.g., Bitcoin) as a database to provide a storage mechanism? – NO
  - Throughput a few transactions per second (tps); (**exceeding 1 million tps**)
  - Latency before a single confirmed write is 10 minutes; (**a fraction of a second**)
  - Capacity is a few dozen GB; (**petabytes and beyond**)
  - Adding nodes is terrible: with a doubling of nodes, network traffic quadruples with
  - no improvement in throughput, latency, or capacity; (**throughput and capacity that increases as nodes get added**)
  - Not support querying. (**rich abilities for insertion, queries, and access control in SQL or NoSQL flavors**)

Background

- Can we frame a traditional blockchain (e.g., Bitcoin) as a database to provide a storage mechanism? – NO
  - **Throughput** a few transactions per second (tps); (**exceeding 1 million tps**)
  - **Latency** before a single confirmed write is 10 minutes; (**a fraction of a second**)
  - **Capacity** is a few dozen GB; (**petabytes and beyond**)
  - **Scale**, adding nodes is terrible: with a doubling of nodes, network traffic quadruples with no improvement in throughput, latency, or capacity; (**throughput and capacity that increases as nodes get added**)
  - **Querying:** not support. (**rich abilities for insertion, queries, and access control in SQL or NoSQL flavors**)

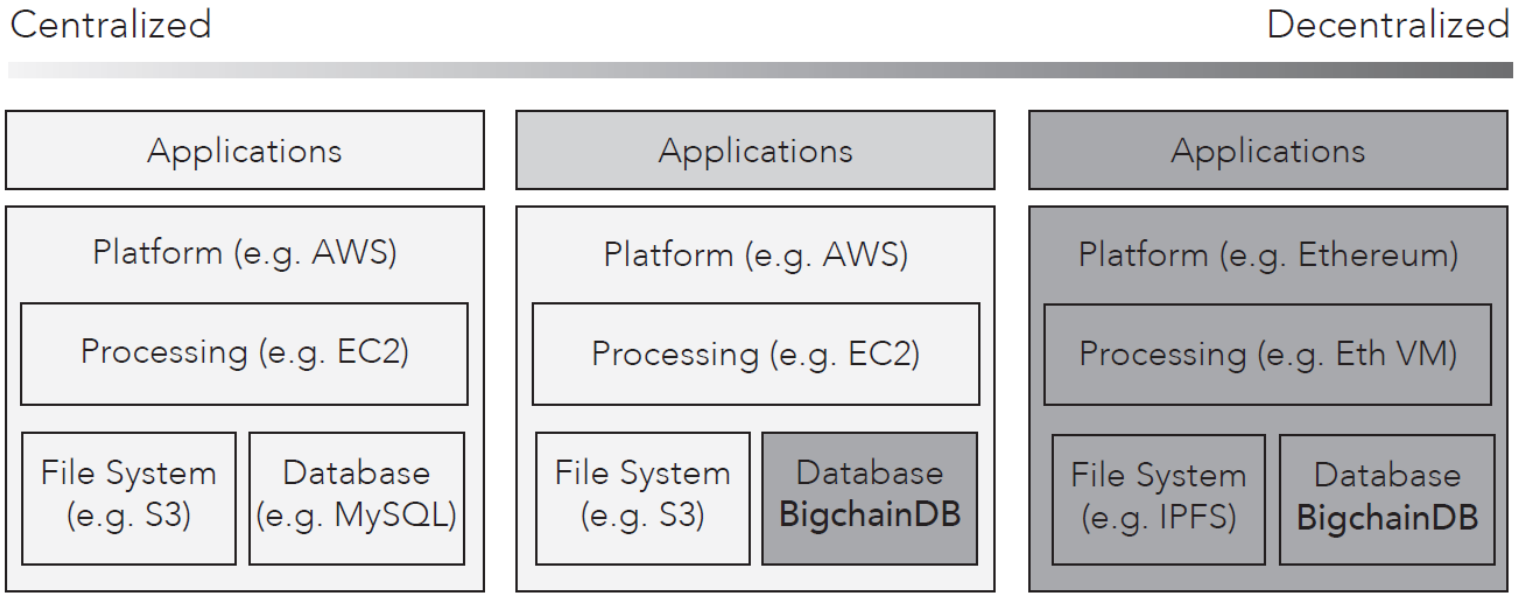
Background

- What is BigchainDB
  - It is for database-style decentralized storage: a blockchain database. BigchainDB combines the key benefits of distributed DBs and traditional blockchains, with an emphasis on scale.

	Traditional Blockchain	Traditional Distributed DB	BigchainDB
High Throughput; increases with nodes↑	-	✓	✓
Low Latency	-	✓	✓
High Capacity; increases with nodes↑	-	✓	✓
Rich querying	-	✓	✓
Rich permissioning	-	✓	✓
Decentralized control	✓	-	✓
Immutability	✓	-	✓
Creation & movement of digital assets	✓	-	✓
Event chain structure	Merkle Tree	-	Hash Chain

Background

- BigchainDB in the Decentralization Ecosystem



From a base context of a centralized cloud computing ecosystem (left), BigchainDB can be added as another database to gain some decentralization benefits (middle). It also fits into a full-blown decentralization ecosystem (right).

- Background
- Preliminaries
  - Traditional blockchain scalability
  - Distributed DBs
- Description
- Implementation Details
- Transaction Latency
- Private vs. Public

Preliminaries

Traditional blockchain scalability

- Blockchain is a distributed database (DB) that solves the “Strong Byzantine Generals” (SBG) problem, i.e., a combination of the **Byzantine Generals Problem** and the **Sybil Attack Problem**.
- **Byzantine Generals Problem** is the dependability of a fault-tolerant computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component has failed. In a "Byzantine failure", a component such as a server can inconsistently appear both failed and functioning to failure-detection systems, presenting different symptoms to different observers.
- **Sybil Attack Problem** is an attack wherein a reputation system is subverted by forging identities in peer-to-peer networks.

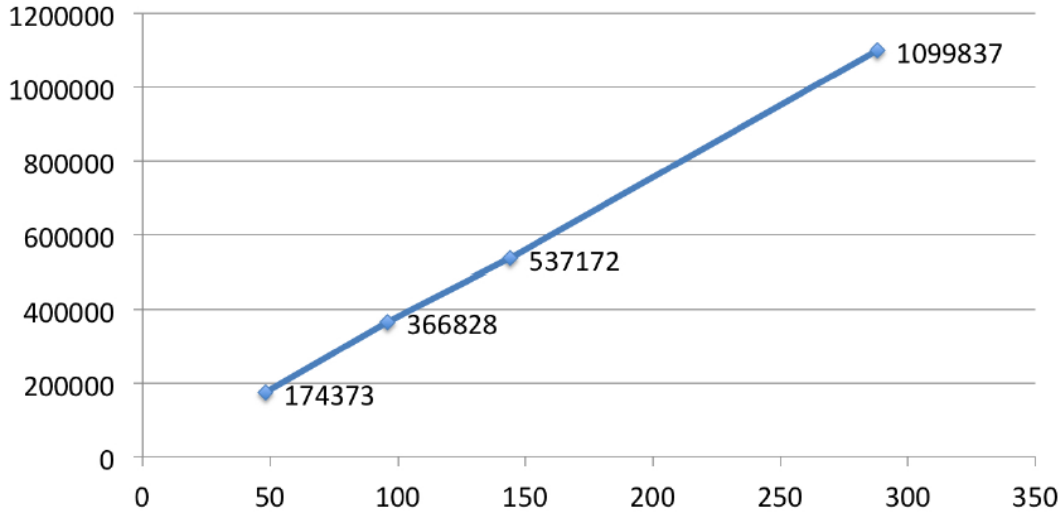
Preliminaries

- Traditional blockchain scalability
  - Throughput
  - Latency
  - Capacity
  - Network bandwidth.

- Why?
  - Consensus Algorithm: POW;
  - Replication: Full.

Preliminaries

- The scalability of distributed DBs



Netflix experimental data on throughput of its Cassandra database (Client writes/s by node count - Replication Factor=3). The x-axis is number of nodes; the y-axis is writes per second.

Preliminaries

- Consensus algorithms of distributed DBs
  - the consensus problem is the problem of figuring out how to get a bunch of isolated computing processes to agree on something, when some of them may be faulty, and they can only communicate by two-party messages.
  - Precise statements of the consensus problem: Byzantine fault tolerant (BFT)
  - The best-known fault-tolerant consensus algorithm is **Paxos**
  - Practical BFT Consensus Algorithms: Aardvark [42], RBFT [43] and Stellar [44] are examples of algorithms aimed at improving speed and reliability.

Preliminaries

- the strengths and weaknesses of DBs that use distributed consensus algorithms such as Paxos.
  - **Strengths**: Paxos is a field-proven consensus algorithm that tolerates benign faults, and can handle high throughput, low latency, high capacity, efficient network utilization, and any shape of data.
  - **Weaknesses**
    - Controlled by a single admin user
    - Mutable
    - Not usable by participants with divergent interests
    - Not designed to stop Sybil attacks
    - Traditionally without support for the creation and transfer of digital assets
    - Not typically open to the public

- Background
- Preliminaries
- Description
- Implementation Details
- Transaction Latency
- Private vs. Public

Description

Principles

- BigchainDB starts with a “big data” distributed database, and adds blockchain characteristics.
  - BigchainDB inherits high throughput, high capacity, a full-featured NoSQL query language, efficient querying, and permissioning. Nodes can be added to increase throughput and capacity.
  - Added blockchain characteristics
    - **Decentralized control**, where “no one” owns or controls a network;
    - **Immutability**, where written data is tamper-resistant (“forever”);
    - **The ability to create & transfer assets** on the network, without reliance on a central entity.

Description

Architecture

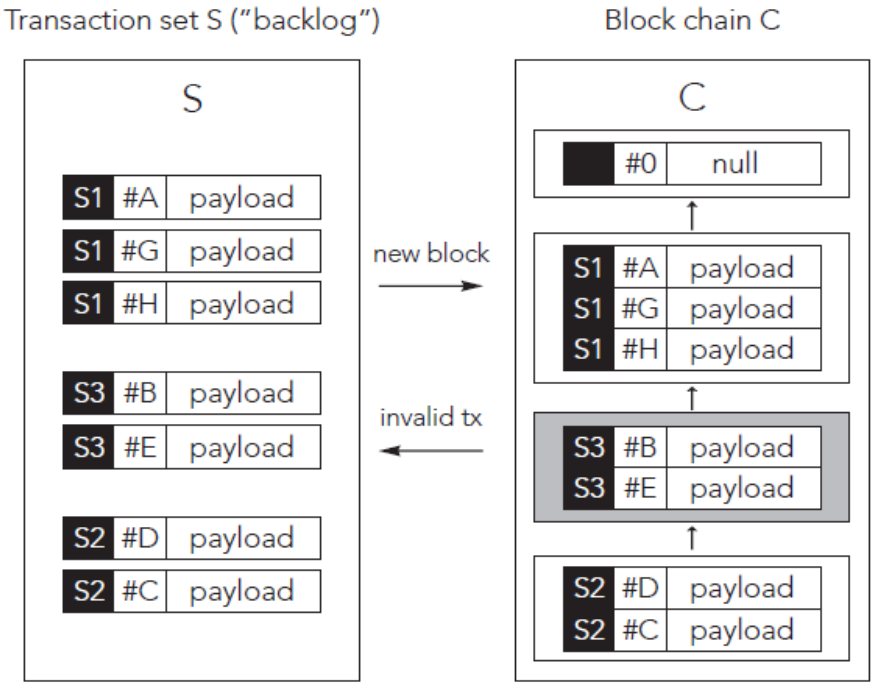


Figure 3: Architecture of BigchainDB system. There are two big data distributed databases: a Transaction Set **S** (left) to take in and assign incoming transactions, and a Blockchain **C** (right) holding ordered transactions that are "etched into stone". The signing nodes running the BigchainDB Consensus Algorithm update **S**, **C**, and the transactions (txs) between them.

Description

Behavioral Description

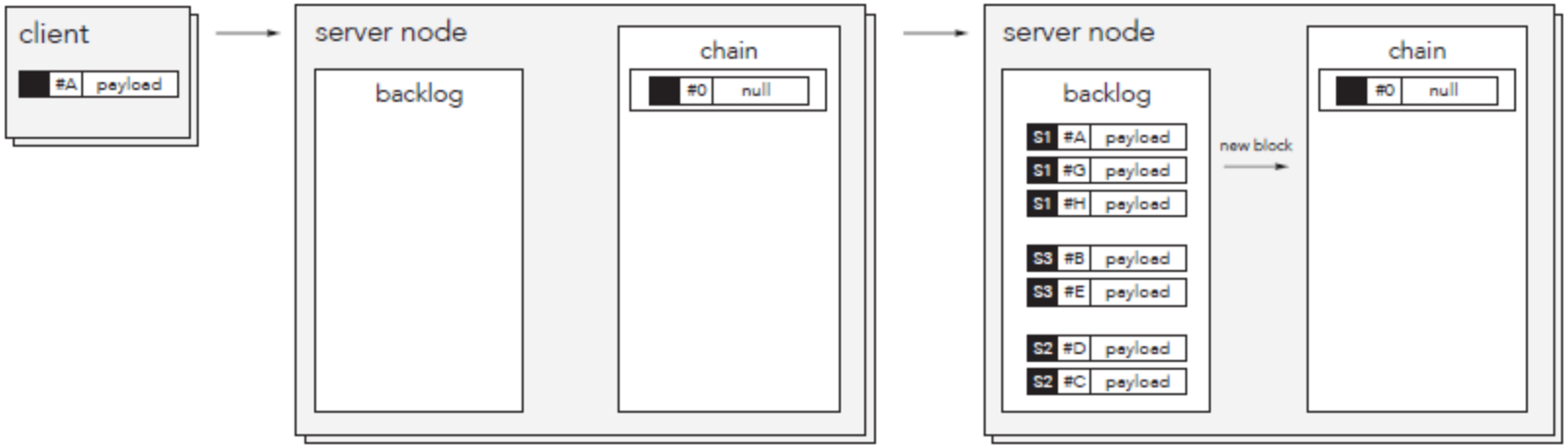


Figure 4: *Left:* The backlog **S** starts empty and the chain **C** starts with only a genesis block. *Right:* Clients have inserted transactions into backlog **S** and assigned to nodes 1, 3, and 2.

Description

Behavioral Description

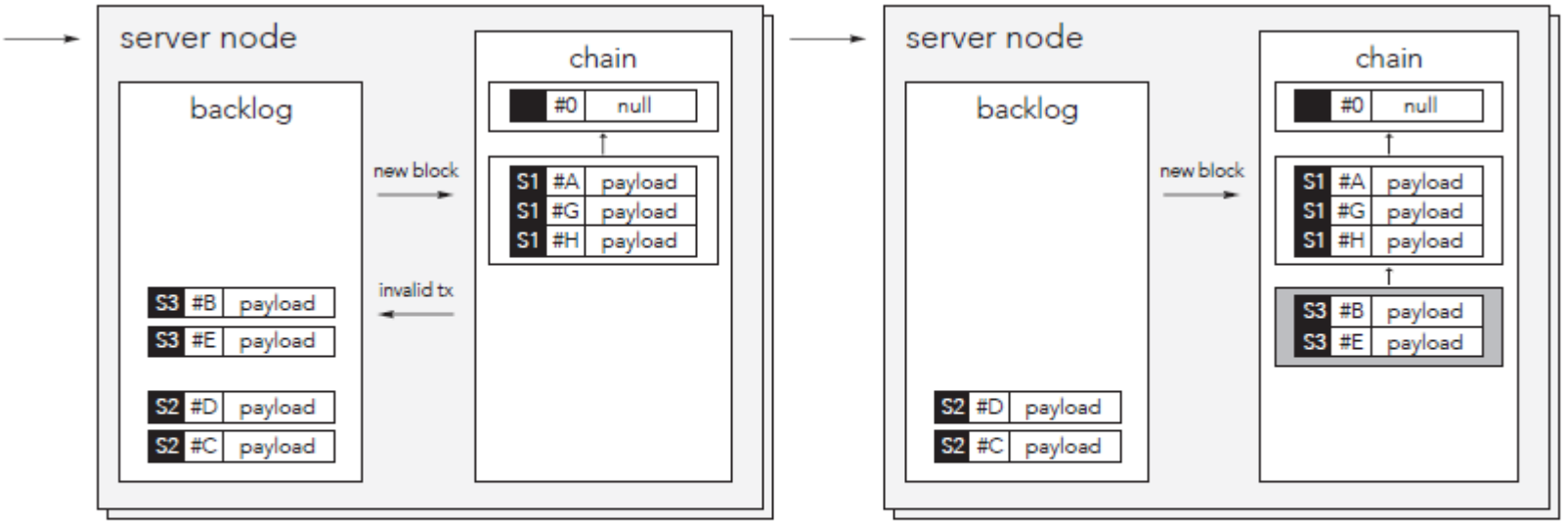


Figure 5: *Left:* Node 1 has moved its assigned transactions from backlog **S** to chain **C**. *Right:* Node 3 has processed its assigned transactions too.

Description

Behavioral Description

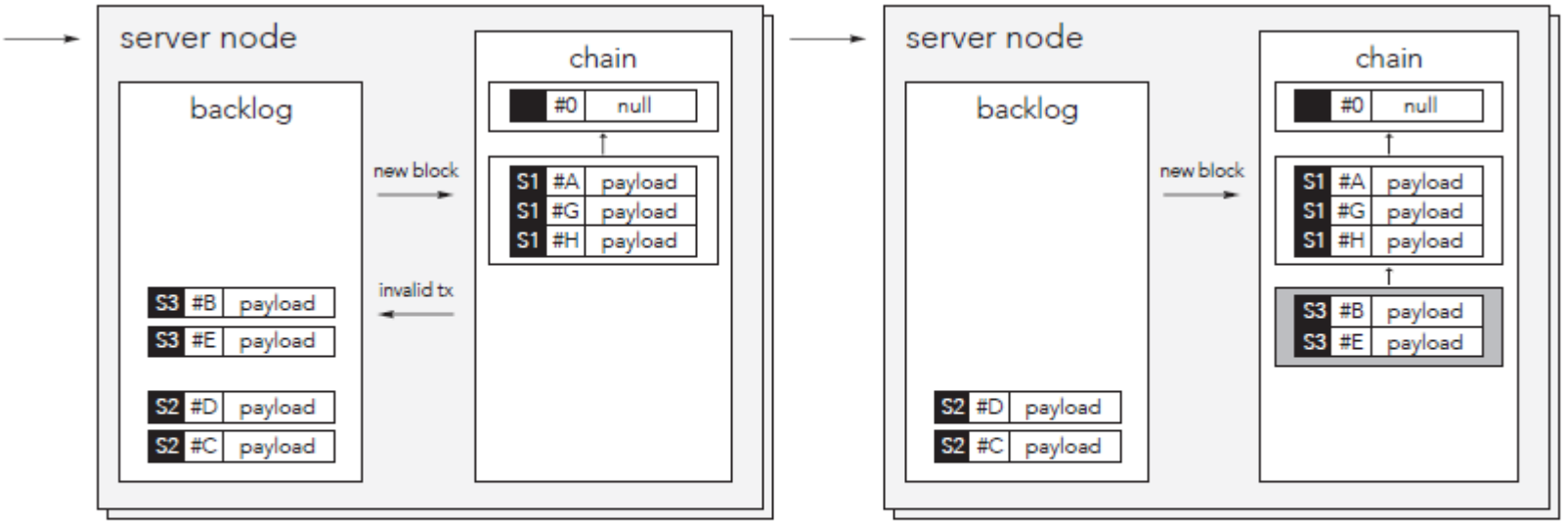


Figure 5: *Left:* Node 1 has moved its assigned transactions from backlog **S** to chain **C**. *Right:* Node 3 has processed its assigned transactions too.

Description

Behavioral Description

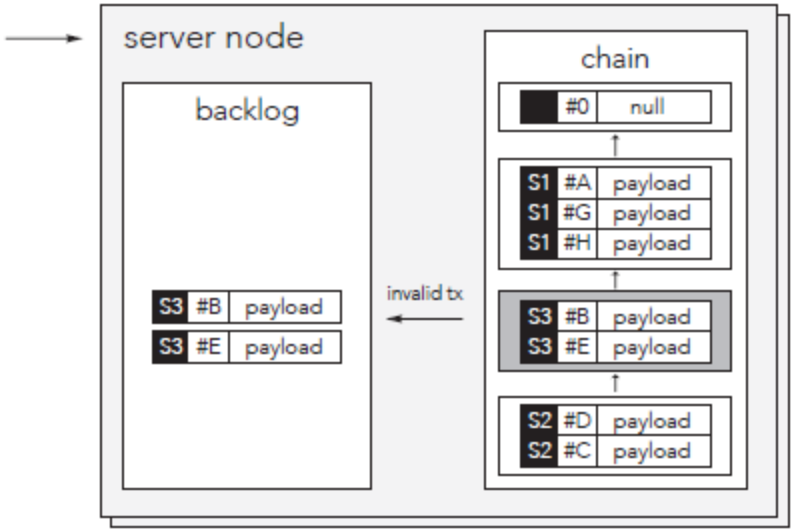


Figure 6: Transactions from an invalid block (on right, shaded) get re-inserted into backlog **S** for re-consideration.

Description

- Transaction Model
  - creation transactions and transfer transactions
- Block Model
- Vote Model

Description

Block Validity and Blockchain Pipelining

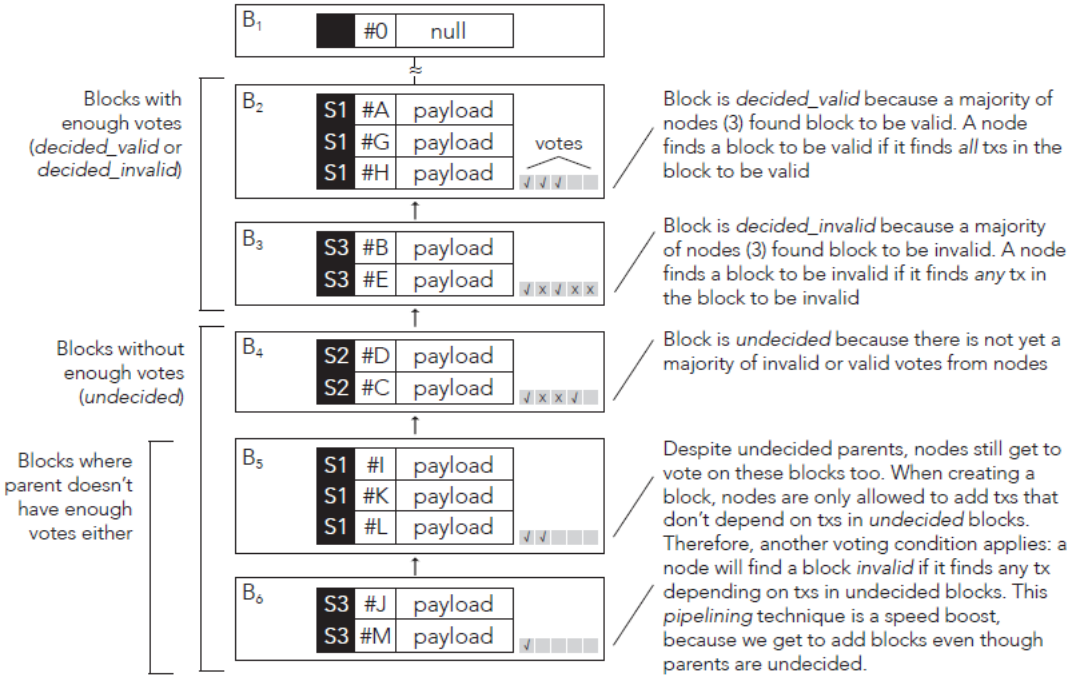


Figure 8: Pipelining in the BigchainDB blockchain C. Votes accumulate on each block. Blocks can continue to accumulate on the blockchain, even though their parents, grandparents, etc. may be undecided. The key is that when adding a new block, we can include transactions that do not depend on transactions in undecided blocks.

Description

BigchainDB Consensus Algorithm (BCA)

- The BigChainDB Consensus Algorithm (BCA) is a state machine that runs on each “signing” node (server).

Listing 1: BigchainDB Consensus Algorithm. This algorithm runs on every signing node.

```
def mainLoop(): # Pseudocode for signing node k
  # Assume S and C exist and are initialized,
  # and C contains a genesis block.
  global S, C # tx set and blockchain globally visible
  while True:
    S = assignTransactions(S, k)
    Sk, C = addBlock(Sk, C, k)
    C = voteOnBlocks(C, k)

```

Description

- Checklist
  - Block construction order
  - Hashing votes
  - Dropping transactions
  - Denial of service
  - Client transaction order
  - Database built-in communication vulnerability
  - Double spends
  - Malicious behavior
  - Admin becoming god
  - Offline nodes
  - Chaining blocks rather than transactions

Description

Native Assets

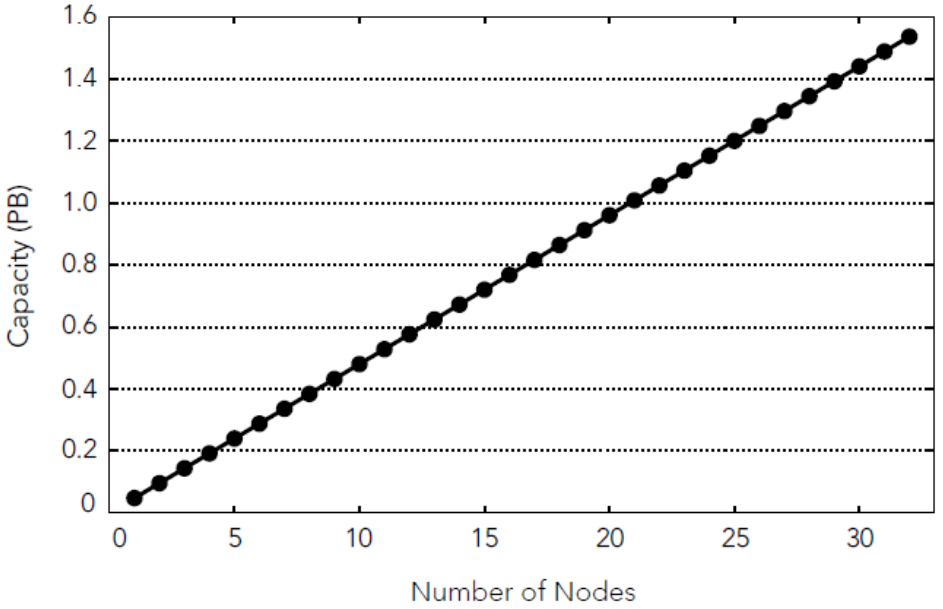
Table 2: Native Assets Validity & Incentivization

	Traditional Blockchain		BigchainDB	
	Native Asset	Overlay Assets	Native Asset	Overlay Assets
Asset Types	Y (one)	Y (multiple)	Y (multiple)	N
Validated by Blockchain Consensus	Y	N	Y	N/A
Incentivization Via	Native Asset (e.g. Mining Reward)		External Fees (e.g. Transactions, Storage)	
Payment for Transactions	Obtain Native Assets (e.g. Via an Exchange)		Fiat Currency (e.g. Traditional channels)	

- Background
- Preliminaries
- Description
- Implementation Details
- Transaction Latency
- Private vs. Public

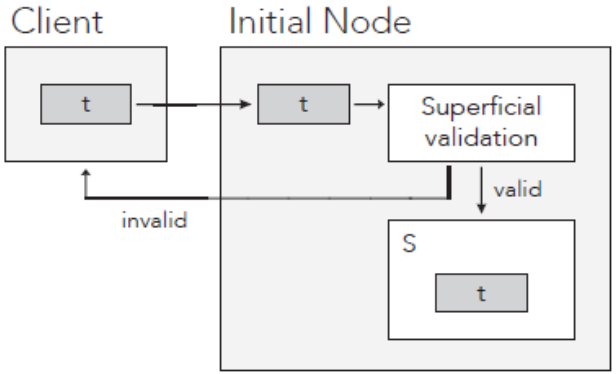
Implementation Details

- Choice of Distributed DB
  - Consistency
  - Automatic Change Notifications
- BigchainDB Capacity
- Serialization
- Cryptography
  - Cryptographic Hashes
  - Keys and Signatures



- Background
- Preliminaries
- Description
- Implementation Details
- Transaction Latency
- Private vs. Public

Transaction Latency



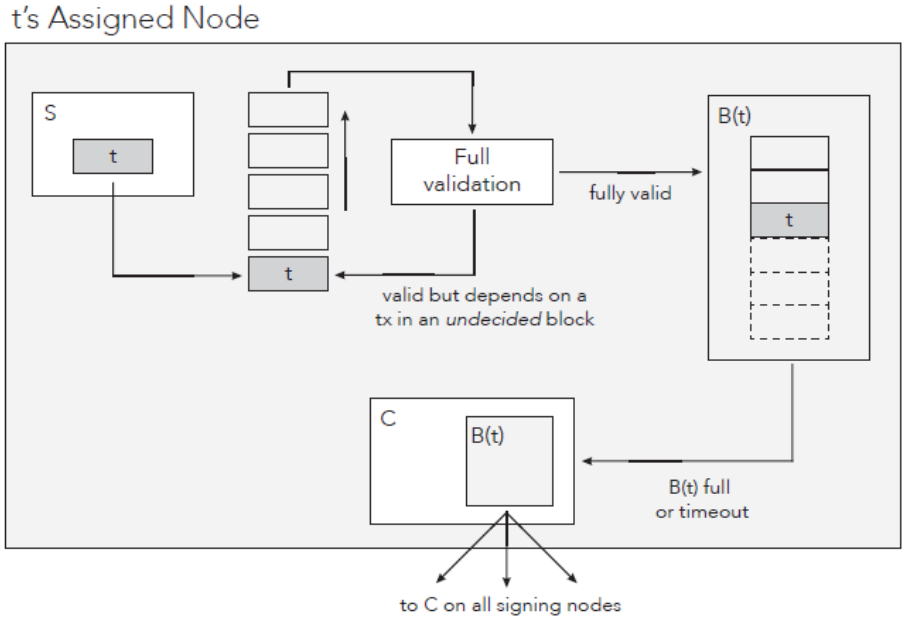


Figure 10: Life of a Transaction, Part 1/2

- Background
- Preliminaries
- Description
- Implementation Details
- Transaction Latency
- Private vs. Public

Transaction Latency

- Permissions, Identities, and Roles

Table 4: Example Permissioning / Roles in an Enterprise BigchainDB Instance

Action	Requires vote	Voting Node	Sys Admin	Issuer	Trader	Broker	Authenticator	Auditor	Core vs Overlay
Vote on Admin & Asset Actions		Y							Core
Admin actions									
Update Role or Permissions	Y	Y	Y						Core
Add/Remove Voting Node	Y	Y	Y[16]						Core
Update software	Y	Y	Y						Core
Asset actions									
Issue Asset	Y			Y					Core
Transfer Asset	Y			O	O	P			Core
Receive Asset	Y			Y	Y				Core
Grant Read Access on Asset	Y			O	O	P	P		Core
Consign Asset	Y			O	O				Overlay
Receive Asset Consignment	Y			Y	Y	Y			Overlay
Add Asset Information	Y			O	O	P			Overlay
Add Authentication Information	Y			O	O		P		Overlay
Create Certificate of Authenticity	N			O	O	P			Overlay
Read actions									
Read Asset Information	N	Y	Y	O	Y	P	P	P	Overlay
Read Certificate of Authenticity	N	Y	Y	O	Y	P	P	P	Overlay

Transaction Latency

- Permissions, Identities, and Roles

Table 5: Example Permissioning / Roles in an Public BigchainDB

Action	Requires vote	Voting Node	Sys Admin	Issuer	User	Authenticator
Vote on Admin & Asset Actions		Y				Core
Admin actions						
Update Role or Permissions	Y	Y	Y			Core
Add/Remove Voting Node	Y	Y	Y[16]			Core
Update software	Y	Y	Y			Core
Asset actions						
Issue Asset	Y			Y		Core
Transfer Asset	Y			O		Core
Receive Asset	Y			Y		Core
Grant Read Access on Asset	Y			N/A	N/A	Core
Consign Asset	Y			O		Overlay
Receive Asset Consignment	Y			Y		Overlay
Add Asset Information	Y			O		Overlay
Add Authentication Information	Y			O	P	Overlay
Create Certificate of Authenticity	N			O		Overlay
Read actions						
Read Asset Information	N	Y	Y	O	P	Overlay
Read Certificate of Authenticity	N	Y	Y	O	P	Overlay