EXONUM

Lecture 10 (2): Building Secure Blockchain Applications in Java on Exonum

Yury Yanovich

BITFURY

BITFURY

Blockchain transactions processing

We process Bitcoin Blockchain transactions and provide security to Bitcoin Blockchain

EXONUM

CRYSTAL

peach

Corporate level blockchain solutions

A framework for creating private blockchains, an analysis of blockchain data, a solution for micro-payments

Third-Party Infrastructure Solutions

Proprietary microelectronics BlockBox Air Cooled (mobile data centers with cooling system), managed hosting services.

/ Traditional Architecture of Web-Apps

A server or two

Admin

Admin is a King

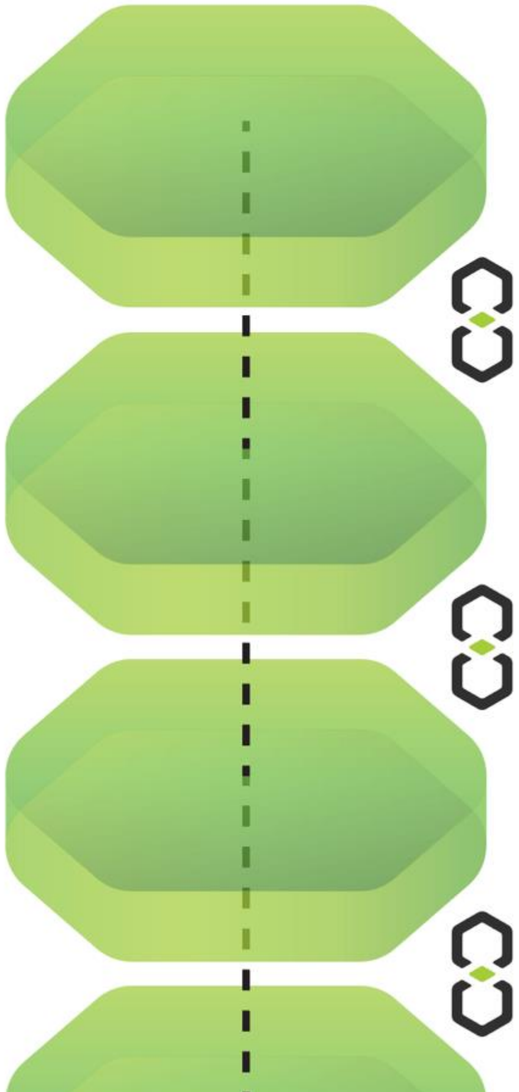
- Can change any data
- Can lose sensitive data
- Absolute faith

Plan

1. Blockchain intro
2. Exonum framework overview
3. Implementing smart-contracts in Java
4. Running a blockchain network

01

Blockchain Technology



Blockchain is

a decentralized database with tamper-resistant log and built-in auditability

- Assumes adversarial threat model
- A way of storing information: in atomic transactions, grouped in blocks
- Blocks joined in a chain with the aid of cryptography

Public blockchain

Anyone can become a ‘miner’:

- Built-in cryptocurrency with mining
- Single platform for everybody

Security: **High**

Performance: **Low**

Private blockchain

Only restricted set of nodes are validators:

- ‘Mining’ incentivizing is outside the solution
- Transaction creation and audit is regulated by blockchain maintainer

Security: **Low**

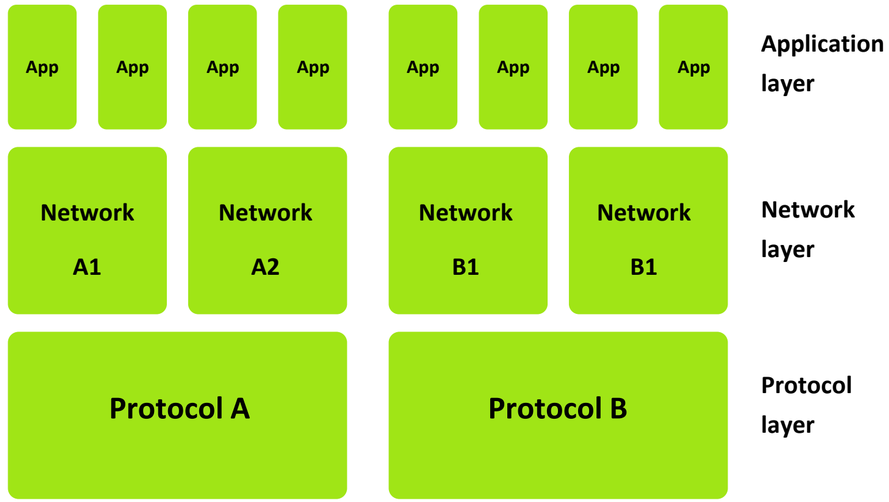
Performance: **High**

Architecture

P2P Decentralized Network

- Application layer
- Network layer
- Protocol layer

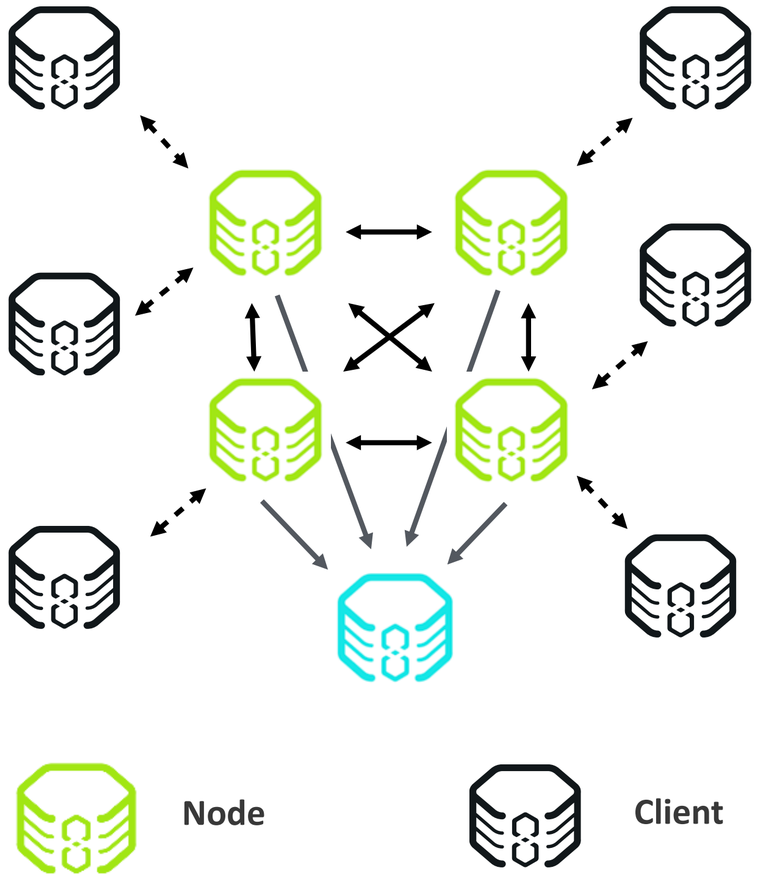
DLT system layers



Architecture

P2P Decentralized Network

- Application layer
- Network layer
- Protocol layer



Architecture

P2P Decentralized Network

- Application layer
- Network layer
- Protocol layer

Example of a DLT integrated ’stack’

citi®
CITICONNECT FOR BLOCKCHAIN

Nasdaq
LINQ BLOCKCHAIN

Chain

02

Exonum Framework

BFT, Rust, Light Client

EXONUM

is an open source platform for private blockchains

- A framework **without** any business logic (UTXOs, built-in cryptocurrency)
- Best in class architecture & algorithms solutions
- Smart contract functionality enabled

Public blockchain

Security: **High**

Performance: **Low**

Private blockchain

Security: **Low**

Performance: **High**

EXONUM

Security: **High**

Performance: **High**

Consensus

BFT Algorithm

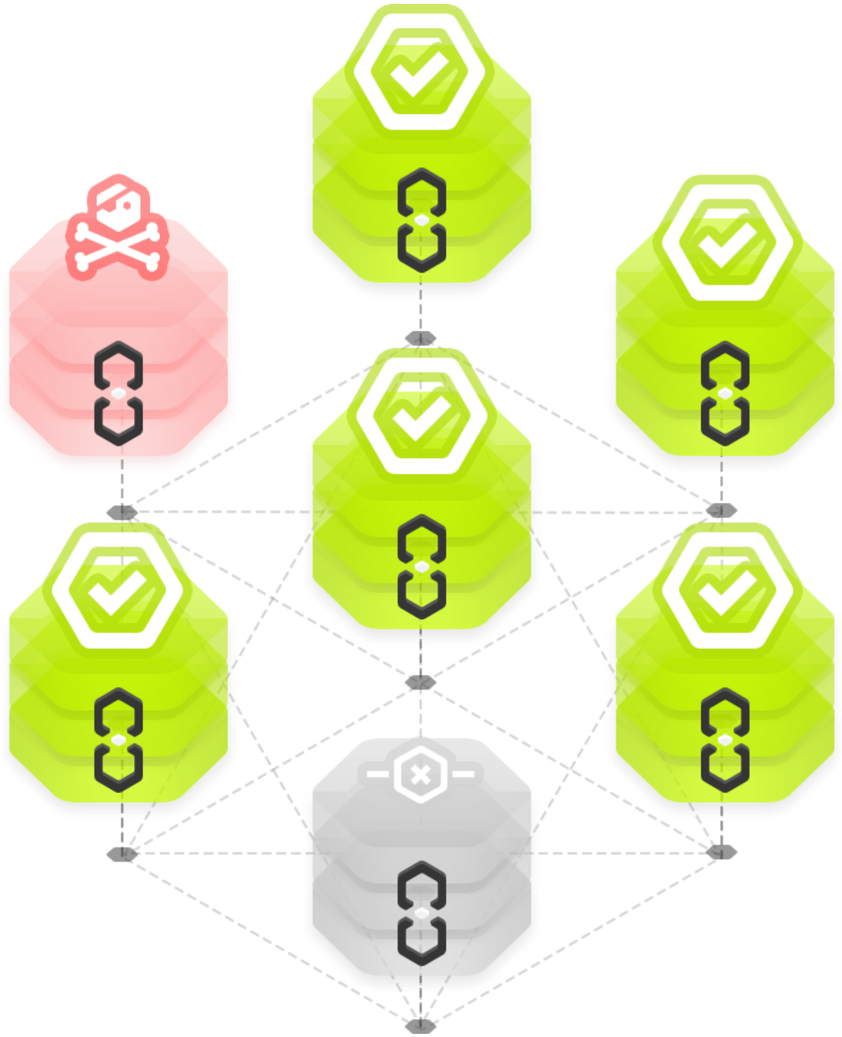
- High performance:

  5 000 tps,

  0.5 s latency

- Secure

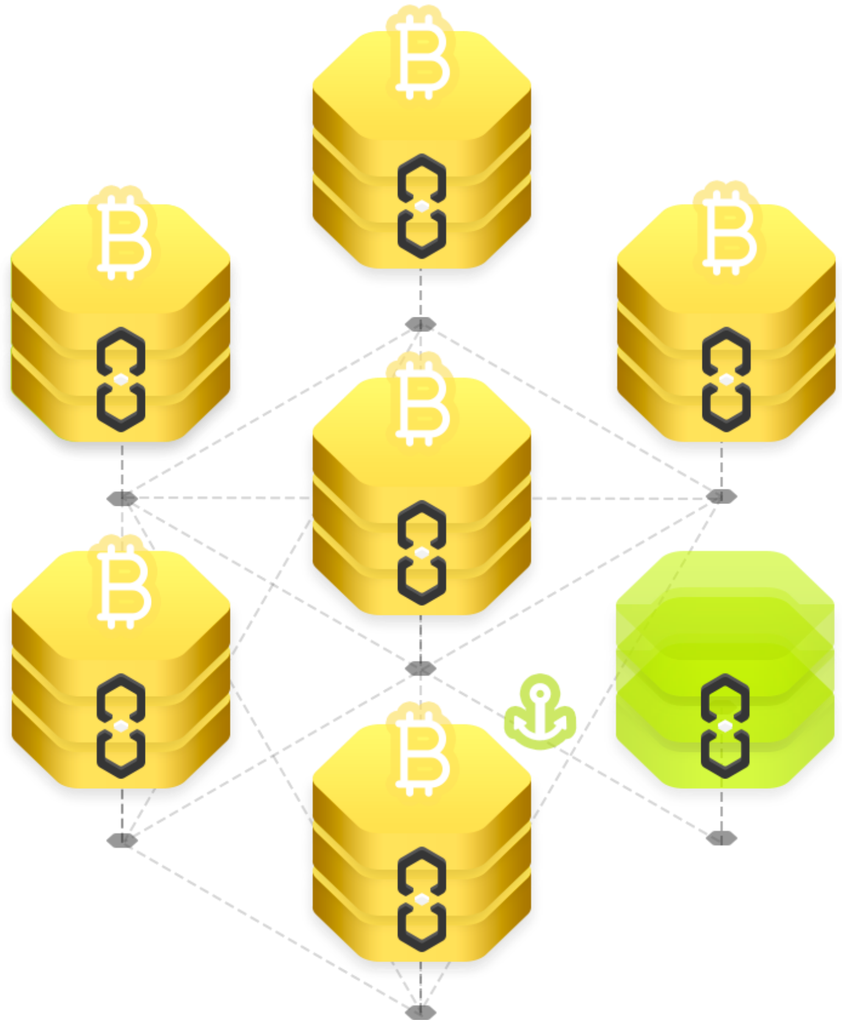
  (**ed25519, sha-256**)



Anchoring

to Bitcoin Blockchain

- Resistance to collision
- Raises security to the level of public blockchain
- Also Byzantine fault tolerant

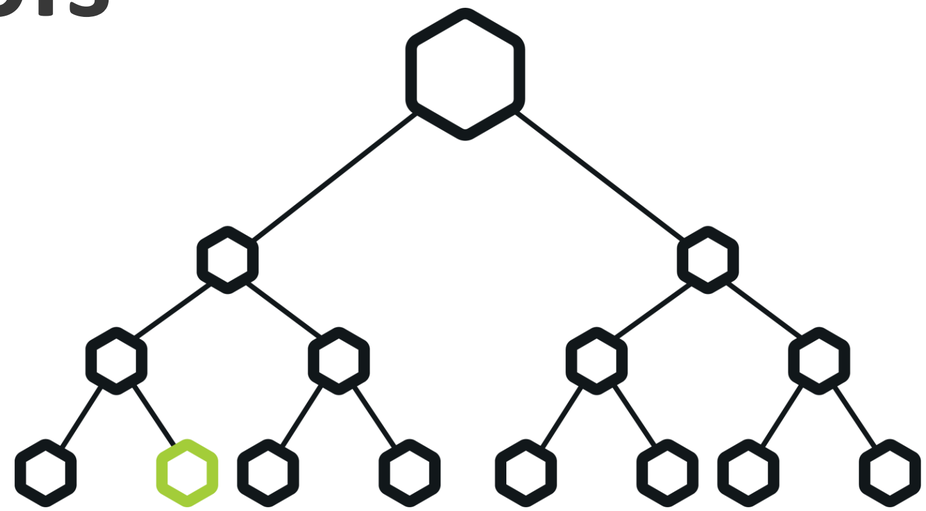


Cryptographic proofs

Exonum uses Merkle trees

for integrity control

- Availability guarantee
- Distributed trust



Rust

Programming Language

- The most secure language
- Safety
- Speed & Performance

Java Binding

Framework

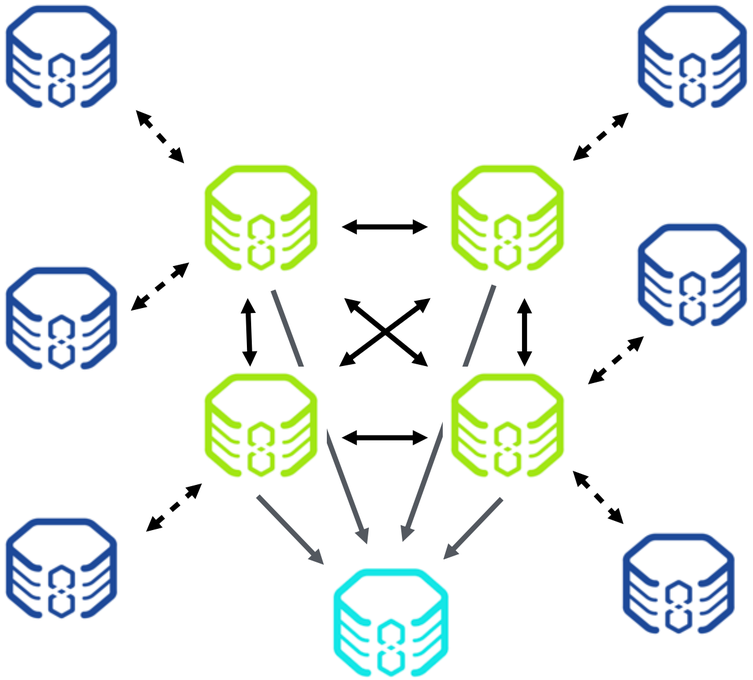
- A framework to define and run smart-contracts in Java
- Powered by Exonum core
- Open-source

Light Client

Software Library

- Client-side software
- Auditability and transparency
- Proofs verification

  + helper functions
- Anchoring verification



Architecture

P2P Decentralized Exonum Network

Validator Node

Stores a full copy of data. Provides network security. Participates in consensus. Accepts and commits records to registers.

Auditor Node

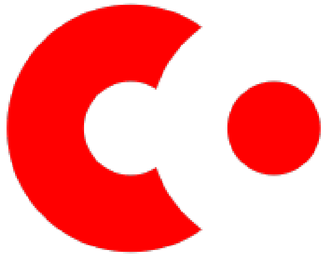
Stores a full copy of data. Does not participate in consensus. Fully checks correctness of data. Provides data on requests of a light client.

Light Client

Does not store a copy of register. Commits records to the register and fetches data from it through requests to validators. Can check a proof of existence of a record in the register.

Exonum is not alone

- Hyperledger
- Multichain
- R3 Corda
- …



Exonum
Permissioned Environment

VS

Ethereum
Permissionless Environment

- ✓ Well-defined responsibility scope for parties (platform providers, service providers, clients);
- ✓ SLA with platform guarantees transaction processing;
- ✓ Intuitive smart contract model; small attack surface thanks to limited contract interfaces;
- ✓ Rust for smart contracts, strong static analysis guaranty;
- ✓ Environment naturally scales to production;
- ✓ May use external cryptocurrencies;
- ✓ The project is open sourced.

- ✗ No responsibility (DAO hack and resulting Blockchain split);
- ✗ No transaction processing guarantees (see DDoS attacks);
- ✗ Complicated smart contract storage and persistence model with large attack surface;
- ✗ Untrustworthy toolchain (see Solidity security concerns);
- ✗ Much effort to adapt to the permissioned environment, resulting in suboptimal security;
- ✓ Popular cryptocurrency;
- ✓ Strong developer community.

Exonum

Blockchain Functionality

Full

VS

Narrow

Fabric

Part of Blockchains

- ✓ Own BFT Consensus algorithm with mathematically proven features. High-end performance (up to 9000 tps);
- ✓ Has all the necessary Blockchain features;
- ✓ Bitcoin anchoring makes system resistant to malicious behavior;
- ✓ Light client validates cryptographic proofs, provided with each Blockchain response;
- ✓ Medium system modularity, consensus is fixed;
- ✓ Written in Rust that guarantees thread safety;
- ✓ The project is open sourced

- ✗ Faulty proven BFT (was in 0.6, excluded in 1.0). Low to medium performance (500 tps);
- ✗ No difference vs. distributed databases or computing;
- ✗ Can be rewritten in case of maintainers collusion;
- ✗ There’s no way to automatically audit the system for a client;
- ✓ High system modularity, including consensus algorithm;
- ✗ Written in Go. Many modules are absent or insufficiently tested;
- ✓ A big company in SW development behind the project.

Exonum
Smart Contracts

- ✔ Use C bindings for SC execution (has Rust and Java API; **high** performance);
- ✔ Memory safety due to Rust;
- ✔ Dynamic addition of new SC using consensus algorithm with node restart process. Simplifies business logic enforcement process;
- ✔ Interoperability between SC via shared memory (blockchain state) and call-backs.

Fabric
Smart Contracts

- ✔ Uses Docker containers for SC execution (has Go and Java API; moderate performance);
- ✘ No memory safety;
- ✔ Dynamic addition of new SC using consensus algorithm on-the-fly;
- ✔ Interoperability between SC via additional query dispatch level.

Ethereum
Smart Contracts

- ✘ Designed to work in permissionless environment (**very slow** virtual machine, specific programming languages);
- ✘ No memory safety;
- ✔ Dynamic addition of new SC as byte code for virtual machine;
- ✔ Interoperability between SC via virtual machine API.

03

Building a blockchain application using Exonum in Java

A simple cryptocurrency application

- Users creating wallets
- Users transferring funds between wallets
- A network of nodes running the application

Service

A smart contract that:

- Specifies transaction processing rules
- Handles events in the ledger
- Declares its persistent data
- Defines an API

Exonum Storage

- Persistent collections

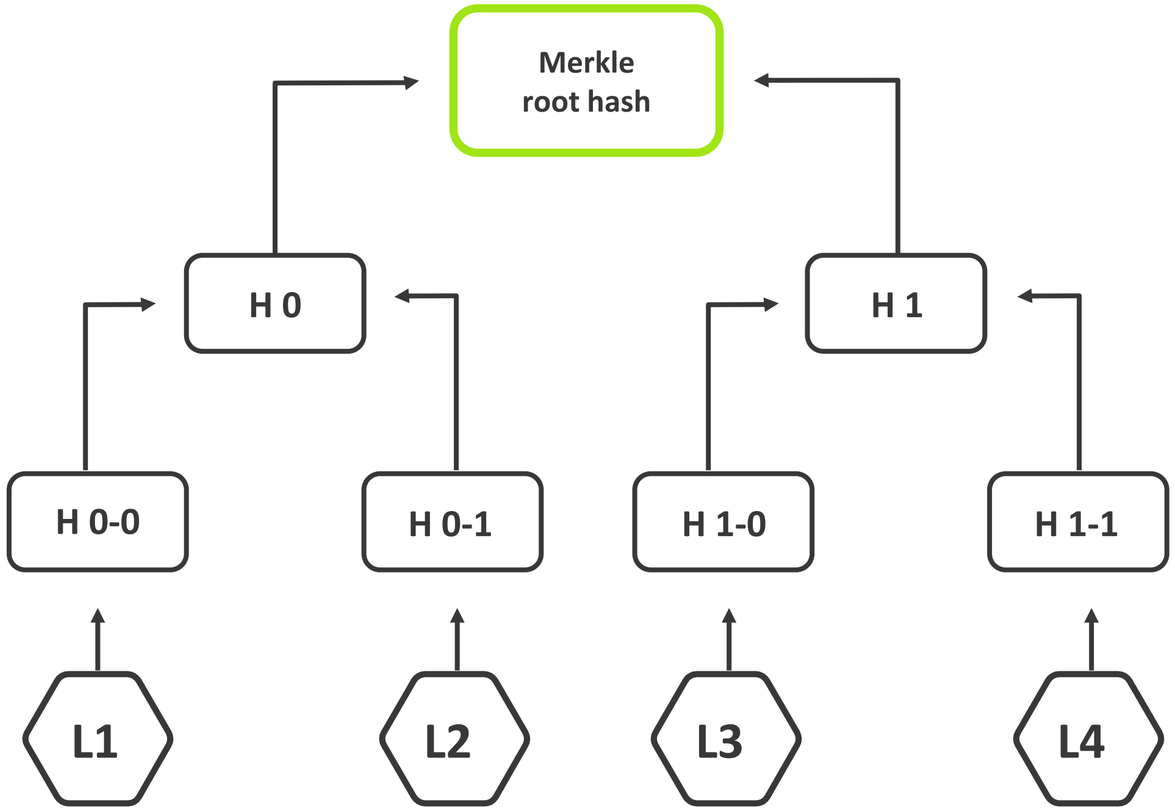
  List, Set, Map

- Merklized collections

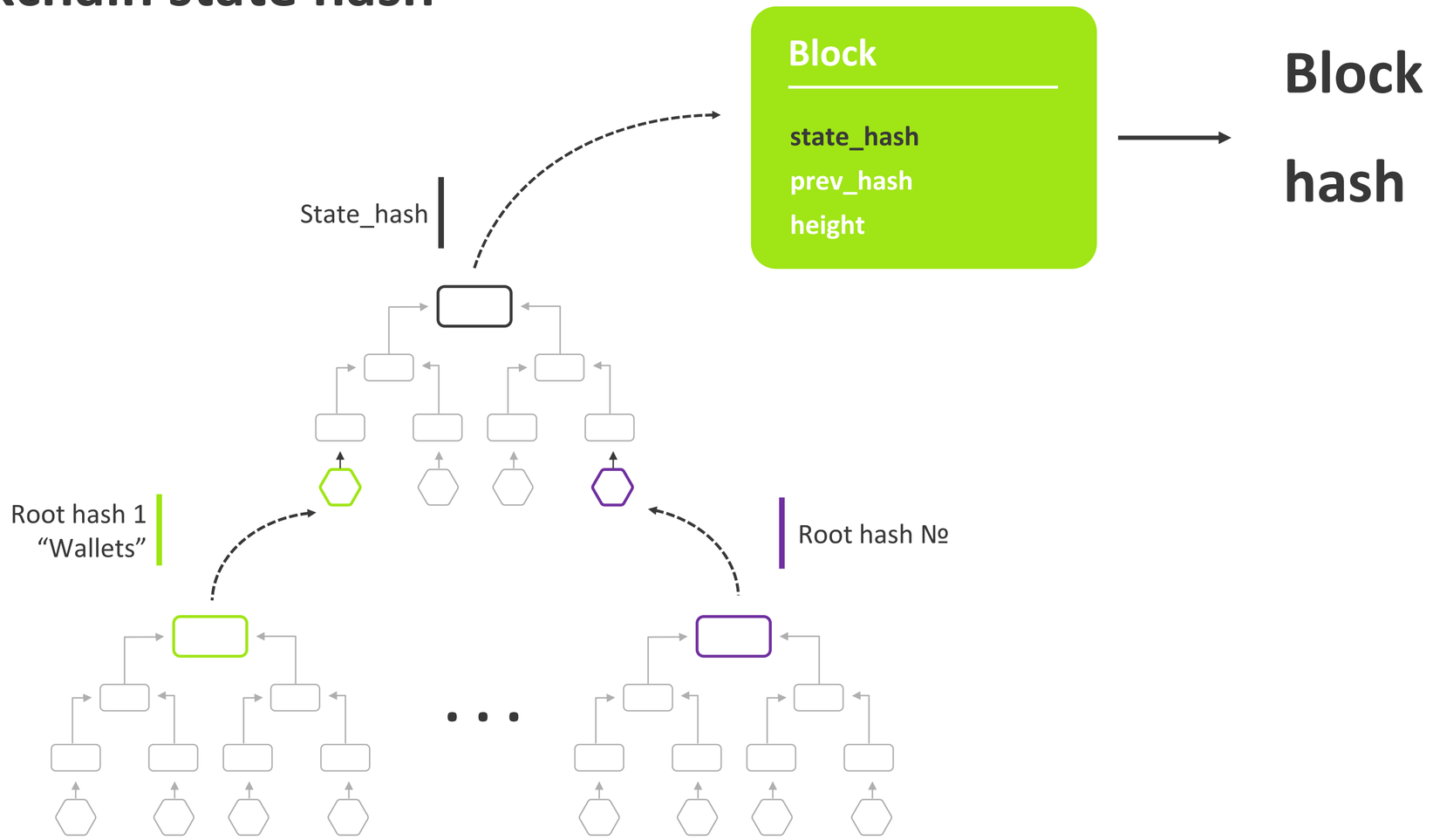
  List, Map

- Transaction log

Merklized collections



Blockchain state hash



Service Schema

- A set of named collections
- Elements serialized
- Storage accessed through a view

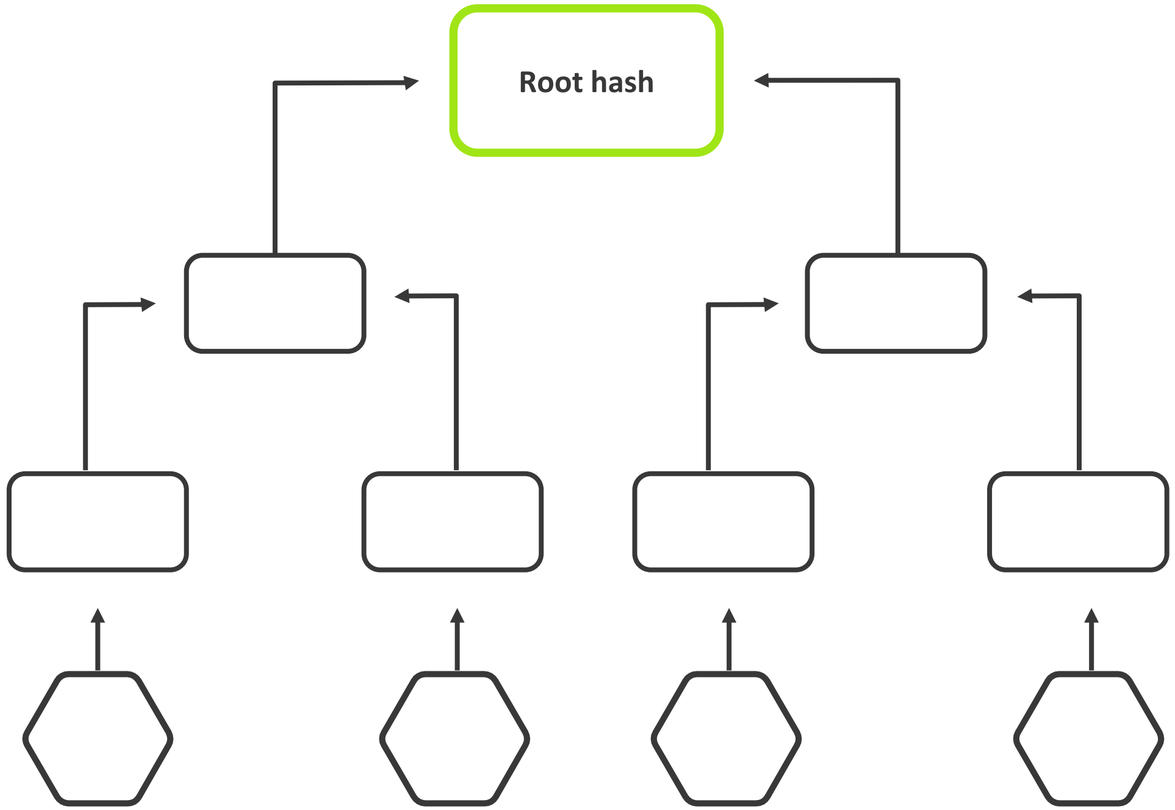
/ Java Binding / Cryptocurrency Schema

```
class CryptocurrencySchema implements Schema {

 private final View view;

 public CryptocurrencySchema(View view) {
   this.view = checkNotNull(view);
 }

 @Override public List<HashCode> getStateHashes() {
   return ImmutableList.of(wallets().getRootHash());
 }

 /** Returns a proof map of wallets. */
 public ProofMap<PublicKey, Wallet> wallets() {
   String name = "crypto.wallets";
   return ProofMapIndexProxy.newInstance(name, view, PublicKeySerializer.INSTANCE,
       WalletSerializer.INSTANCE);
 }
}
```

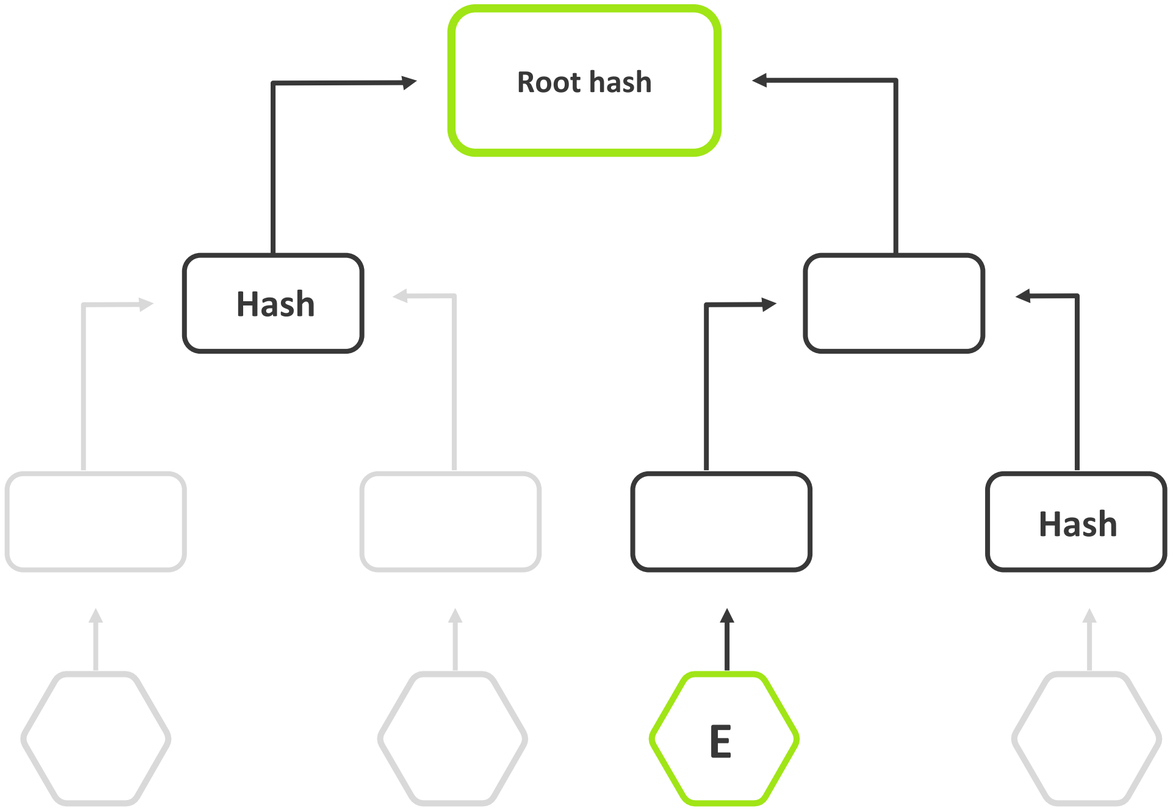
Exonum Proofs

- Available for Merklized lists and maps
- Do not reveal anything but the requested key and value
- Essential to the light clients

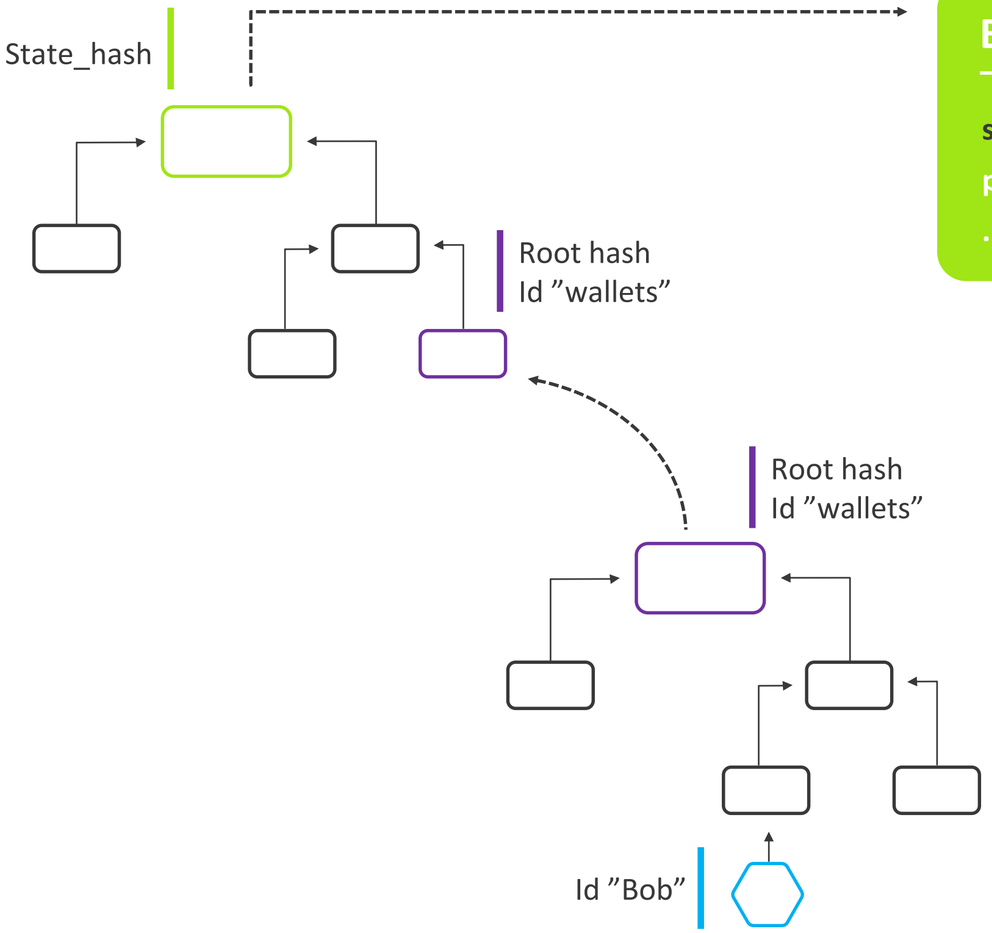
Single Collection Proof



Single Collection Proof



Full Collection Proof



From validators Nodes

Block

state_hash

prev_hash

. . .

+

Signed

Precommit

messages

A full proof, including a user-key proof, state-hash proof, precommits, a block.

/ Java Binding / Full Collection Proof

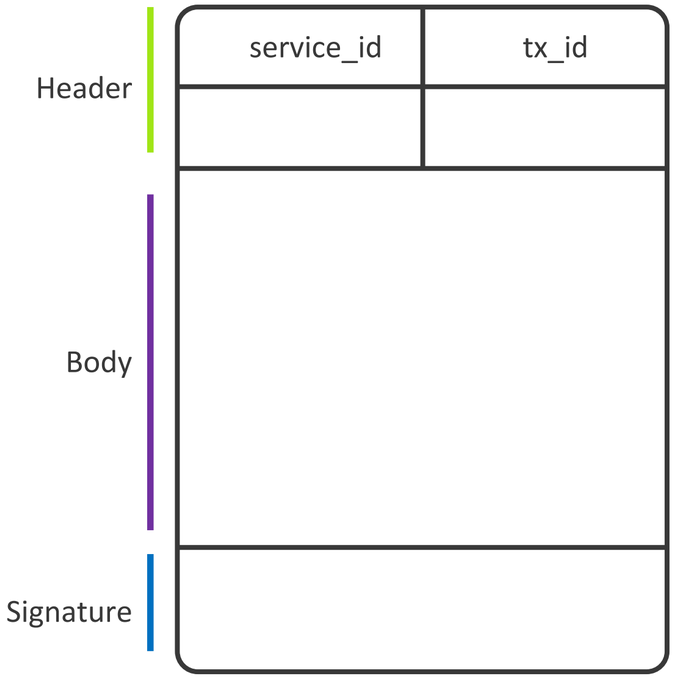
```
/* Not in alpha yet */
FullProof proof = ProofBuilder.newBuilder(storageSnapshot)
    .withMapProof("wallets", walletProof)
    .build();
```

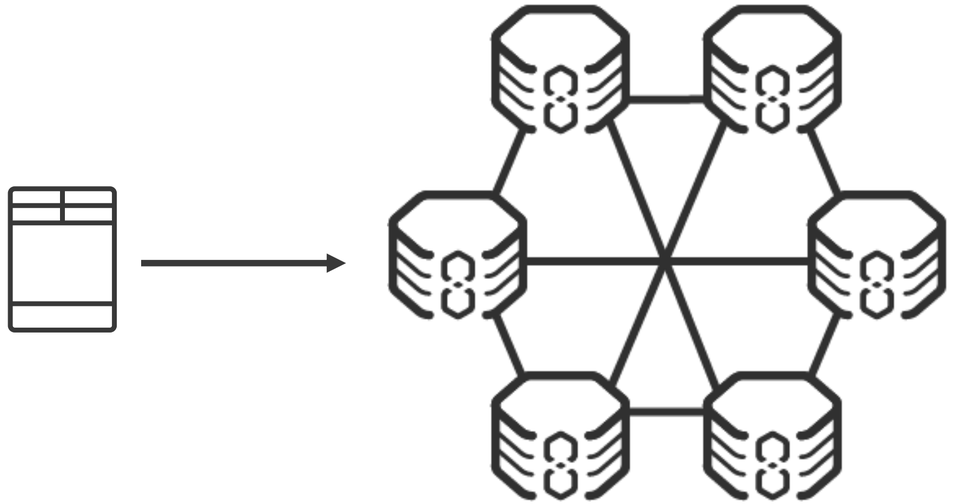
Transactions

- Determine most of the business logic
- Properties:

  Atomic, Authenticated, Ordered

Transaction Messages





/ Java Binding / Transaction Interface

```
public interface Transaction {
  boolean isValid();
  void execute(Fork view);
}
```

/ Java Binding / Create Wallet Transaction

```
class CreateWalletTx implements Transaction {
  CreateWalletTx(BinaryMessage message) { /* Parse a message, not shown */ }

  @Override public boolean isValid() {
    return getMessage().verify(CryptoFunctions.ed25519(), ownerPublicKey);
  }

  @Override public void execute(Fork view) {
    CryptocurrencySchema schema = new CryptocurrencySchema(view);
    MapIndex<PublicKey, Wallet> wallets = schema.wallets();
    if (wallets.containsKey(ownerPublicKey)) {
      throw new TransactionExecutionException(DUPLICATE_WALLET_ID);
    }
    Wallet wallet = new Wallet(initialBalance);
    wallets.put(ownerPublicKey, wallet);
  }
}
```

/ Java Binding / Create Wallet Transaction

```
class CreateWalletTx implements Transaction {
  CreateWalletTx(BinaryMessage message) { /* Parse a message, not shown */ }

  @Override public boolean isValid() {
    return getMessage().verify(CryptoFunctions.ed25519(), ownerPublicKey);
  }

  @Override public void execute(Fork view) {
    CryptocurrencySchema schema = new CryptocurrencySchema(view);
    MapIndex<PublicKey, Wallet> wallets = schema.wallets();
    if (wallets.containsKey(ownerPublicKey)) {
      throw new TransactionExecutionException(DUPLICATE_WALLET_ID);
    }
    Wallet wallet = new Wallet(initialBalance);
    wallets.put(ownerPublicKey, wallet);
  }
}
```

/ Java Binding / Create Wallet Transaction

```
class CreateWalletTx implements Transaction {
  CreateWalletTx(BinaryMessage message) { /* Parse a message, not shown */ }

  @Override public boolean isValid() {
    return getMessage().verify(CryptoFunctions.ed25519(), ownerPublicKey);
  }

  @Override public void execute(Fork view) {
    CryptocurrencySchema schema = new CryptocurrencySchema(view);
    MapIndex<PublicKey, Wallet> wallets = schema.wallets();
    if (wallets.containsKey(ownerPublicKey)) {
      throw new TransactionExecutionException(DUPLICATE_WALLET_ID);
    }
    Wallet wallet = new Wallet(initialBalance);
    wallets.put(ownerPublicKey, wallet);
  }
}
```

/ Java Binding / Create Wallet Transaction

```
class CreateWalletTx implements Transaction {
  CreateWalletTx(BinaryMessage message) { /* Parse a message, not shown */ }

  @Override public boolean isValid() {
    return getMessage().verify(CryptoFunctions.ed25519(), ownerPublicKey);
  }

  @Override public void execute(Fork view) {
    CryptocurrencySchema schema = new CryptocurrencySchema(view);
    MapIndex<PublicKey, Wallet> wallets = schema.wallets();
    if (wallets.containsKey(ownerPublicKey)) {
      throw new TransactionExecutionException(DUPLICATE_WALLET_ID);
    }
    Wallet wallet = new Wallet(initialBalance);
    wallets.put(ownerPublicKey, wallet);
  }
}
```

/ Java Binding / Transfer Funds Transaction

```
class TransferTx implements Transaction {
 TransferTx(BinaryMessage message) { /* Parse a message, not shown */ }

 @Override
 public boolean isValid() { /* Same as before */ }

 @Override
 public void execute(Fork view) { /* Next slide */ }

}
```

/ Java Binding / Transfer Funds Transaction

```
@Override
public void execute(Fork view) {
  CryptocurrencySchema schema = new CryptocurrencySchema(view);
  MapIndex<PublicKey, Wallet> wallets = schema.wallets();
  if (wallets.containsKey(sender) && wallets.containsKey(receiver)) {
    Wallet from = wallets.get(sender);
    Wallet to = wallets.get(receiver);
    if (from.getBalance() < sum) {
      throw new TransactionExecutionException("Insufficient balance");
    }
    if (sender.equals(receiver)) {
      throw new TransactionExecutionException("Sender and receiver are the same")
    }
    wallets.put(sender, new Wallet(from.getBalance() - sum));
    wallets.put(receiver, new Wallet(to.getBalance() + sum));
  }
}
```

/ Java Binding / Transfer Funds Transaction

```
@Override
public void execute(Fork view) {
  CryptocurrencySchema schema = new CryptocurrencySchema(view);
  MapIndex<PublicKey, Wallet> wallets = schema.wallets();
  if (wallets.containsKey(sender) && wallets.containsKey(receiver)) {
    Wallet from = wallets.get(sender);
    Wallet to = wallets.get(receiver);
    if (from.getBalance() < sum) {
      throw new TransactionExecutionException("Insufficient balance");
    }
    if (sender.equals(receiver)) {
      throw new TransactionExecutionException("Sender and receiver are the same")
    }
    wallets.put(sender, new Wallet(from.getBalance() - sum));
    wallets.put(receiver, new Wallet(to.getBalance() + sum));
  }
}
```

/ Java Binding / Transfer Funds Transaction

```
@Override
public void execute(Fork view) {
  CryptocurrencySchema schema = new CryptocurrencySchema(view);
  MapIndex<PublicKey, Wallet> wallets = schema.wallets();
  if (wallets.containsKey(sender) && wallets.containsKey(receiver)) {
    Wallet from = wallets.get(sender);
    Wallet to = wallets.get(receiver);
    if (from.getBalance() < sum) {
      throw new TransactionExecutionException("Insufficient balance");
    }
    if (sender.equals(receiver)) {
      throw new TransactionExecutionException("Sender and receiver are the same")
    }
    wallets.put(sender, new Wallet(from.getBalance() - sum));
    wallets.put(receiver, new Wallet(to.getBalance() + sum));
  }
}
```

/ Java Binding / Transfer Funds Transaction

```
@Override
public void execute(Fork view) {
  CryptocurrencySchema schema = new CryptocurrencySchema(view);
  MapIndex<PublicKey, Wallet> wallets = schema.wallets();
  if (wallets.containsKey(sender) && wallets.containsKey(receiver)) {
    Wallet from = wallets.get(sender);
    Wallet to = wallets.get(receiver);
    if (from.getBalance() < sum) {
      throw new TransactionExecutionException("Insufficient balance");
    }
    if (sender.equals(receiver)) {
      throw new TransactionExecutionException("Sender and receiver are the same")
    }
    wallets.put(sender, new Wallet(from.getBalance() - sum));
    wallets.put(receiver, new Wallet(to.getBalance() + sum));
  }
}
```

/ Java Binding / Transaction Converter

```
class CryptoTransactionConverter implements TransactionConverter {
 @Override public Transaction toTransaction(BinaryMessage message) {
   checkServiceId(message); // Verify it's a message of our service.
   short txId = message.getMessageType();
   switch (txId) {
     case CREATE_WALLET_TX_ID: return new CreateWalletTx(message);
     case TRANSFER_TX_ID: return new TransferTx(message);
     default: throw new IllegalArgumentException(
         "Unknown transaction id: " + txId);
   }
 }
}
```

Service API

- Submit transactions
- Query data from the blockchain

/ Java Binding / API Controller: submit transaction

```
class ApiController {
 void mountApi(Router router) {
   router.route("/submit-transaction")
           .handler(rc -> {
             // Get a binary message from the body.
             BinaryMessage message = messageFromBody(rc.getBody());
             // Create a transaction for the given binary message.
             Transaction tx = service.convertToTransaction(message);
             // Submit the transaction to the network.
             HashCode txHash = service.submitTransaction(tx);
             rc.response()
                     .putHeader("Content-Type", "text/plain")
                     .end(String.valueOf(txHash));
           });
   // …
 }
}
```

/ Java Binding / API Controller : get the balance

```
class ApiController {
  void mountApi(Router router) {
    // …
    router.route("/wallet/:ownerPublicKey").handler(rc -> {
          MultiMap requestParameters = rc.request().params();
          PublicKey ownerPublicKey = PublicKey.fromHexString(
              requestParameters.get("ownerPublicKey"));
          Optional<Wallet> wallet = service.getWallet(ownerPublicKey);
          if (wallet.isPresent()) {
            rc.response()
                .end(gson.toJson(wallet.get()));
          } else {
            rc.response()
                .setStatusCode(HTTP_NOT_FOUND)
                .end();
          }
        });
  }
}
```

/ Java Binding / Service interface

```
public interface Service {
  short getId();
  String getName();
  Optional<String> initialize(Fork fork);
  Transaction convertToTransaction(BinaryMessage message);
  List<HashCode> getStateHashes(Snapshot snapshot);
  void createPublicApiHandlers(Node node, Router router);
}
```

/ Java Binding / Service: get the balance

```
class CryptocurrencyService implements Service {
  short getId() { return 127; }

  String getName() { return "exonum-cryptocurrency"; }

  /** Submits the transaction to the network. */
  public HashCode submitTransaction(Transaction tx) {
    node.submitTransaction(tx);
    return tx.hash();
  }

  /** Returns the wallet of the given owner. */
  Optional<Wallet> getWallet(PublicKey ownerKey) {
    return node.withSnapshot(snapshot -> {
      CryptocurrencySchema schema = new CryptocurrencySchema(snapshot);
      return Optional.ofNullable(schema.wallets().get(ownerKey));
    })
  }
}
```

/ Java Binding / Service: get the balance

```
class CryptocurrencyService implements Service {
  short getId() { return 127; }

  String getName() { return "exonum-cryptocurrency"; }

  /** Submits the transaction to the network. */
  public HashCode submitTransaction(Transaction tx) {
    node.submitTransaction(tx);
    return tx.hash();
  }

  /** Returns the wallet of the given owner. */
  Optional<Wallet> getWallet(PublicKey ownerKey) {
    return node.withSnapshot(snapshot -> {
      CryptocurrencySchema schema = new CryptocurrencySchema(snapshot);
      return Optional.ofNullable(schema.wallets().get(ownerKey));
    })
  }
}
```

/ Java Binding / Service: get the balance

```
class CryptocurrencyService implements Service {
  short getId() { return 127; }

  String getName() { return "exonum-cryptocurrency"; }

  /** Submits the transaction to the network. */
  public HashCode submitTransaction(Transaction tx) {
    node.submitTransaction(tx);
    return tx.hash();
  }

  /** Returns the wallet of the given owner. */
  Optional<Wallet> getWallet(PublicKey ownerKey) {
    return node.withSnapshot(snapshot -> {
      CryptocurrencySchema schema = new CryptocurrencySchema(snapshot);
      return Optional.ofNullable(schema.wallets().get(ownerKey));
    })
  }
}
```

/ Java Binding / Service: get the balance

```
class CryptocurrencyService implements Service {
  short getId() { return 127; }

  String getName() { return "exonum-cryptocurrency"; }

  /** Submits the transaction to the network. */
  public HashCode submitTransaction(Transaction tx) {
    node.submitTransaction(tx);
    return tx.hash();
  }

  /** Returns the wallet of the given owner. */
  Optional<Wallet> getWallet(PublicKey ownerKey) {
    return node.withSnapshot(snapshot -> {
      CryptocurrencySchema schema = new CryptocurrencySchema(snapshot);
      return Optional.ofNullable(schema.wallets().get(ownerKey));
    })
  }
}
```

04

Exonum Network

Exonum Network

- No public platform — run yourself
- Use application:

  Runs Rust and Java services

  Bundles configuration and anchoring services
- Run the nodes!

05

Demo

Useful links

- https://exonum.com/: use cases, documentation, other links
- https://github.com/exonum: Exonum’s Github page
- https://gitter.im/exonum/exonum: chat for developers
- https://www.youtube.com/results?search_query=exonum: many webinars (mostly in Russian)
- Some demo projects
  - https://github.com/exonum/exonum/tree/master/examples
  - https://github.com/exonum/exonum-cryptoowls

What’s Next

- Dynamic Services
- Merkle Database
- Easier communication between services and clients
- Service Migration

See the full roadmap on our web site:

exonum.com/doc/roadmap

Thank You!

based on Dmitry Timofeev’s presentation

github.com/exonum

exonum.com

Join our
Telegram Channel

exonum_blockchain