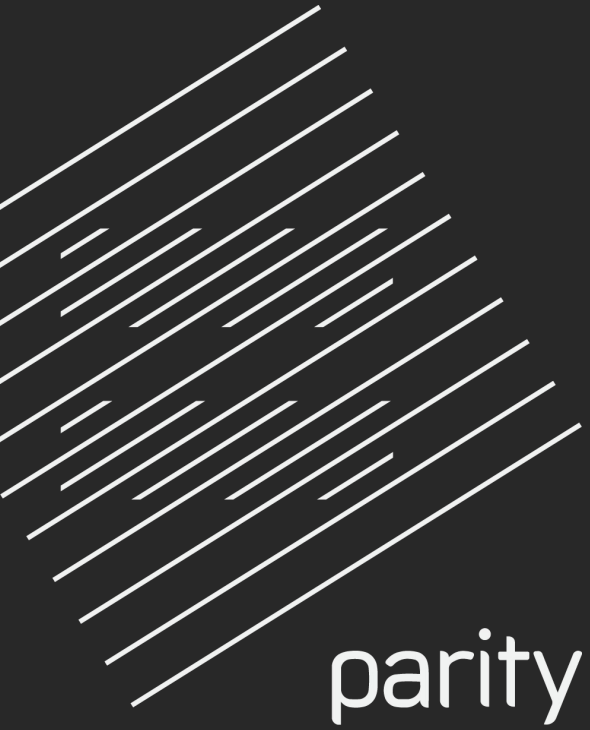


Substrate: A blockchain builder for Polkadot

Deploying and upgrading your own blockchain network
部署和升级自定义区块链网络

Nicole Zhu
nicole@parity.io | @nczhu

Hanwen Cheng
hanwen@parity.io

Get help: tiny.cc/substrate-technical

We will talk about 内容一览

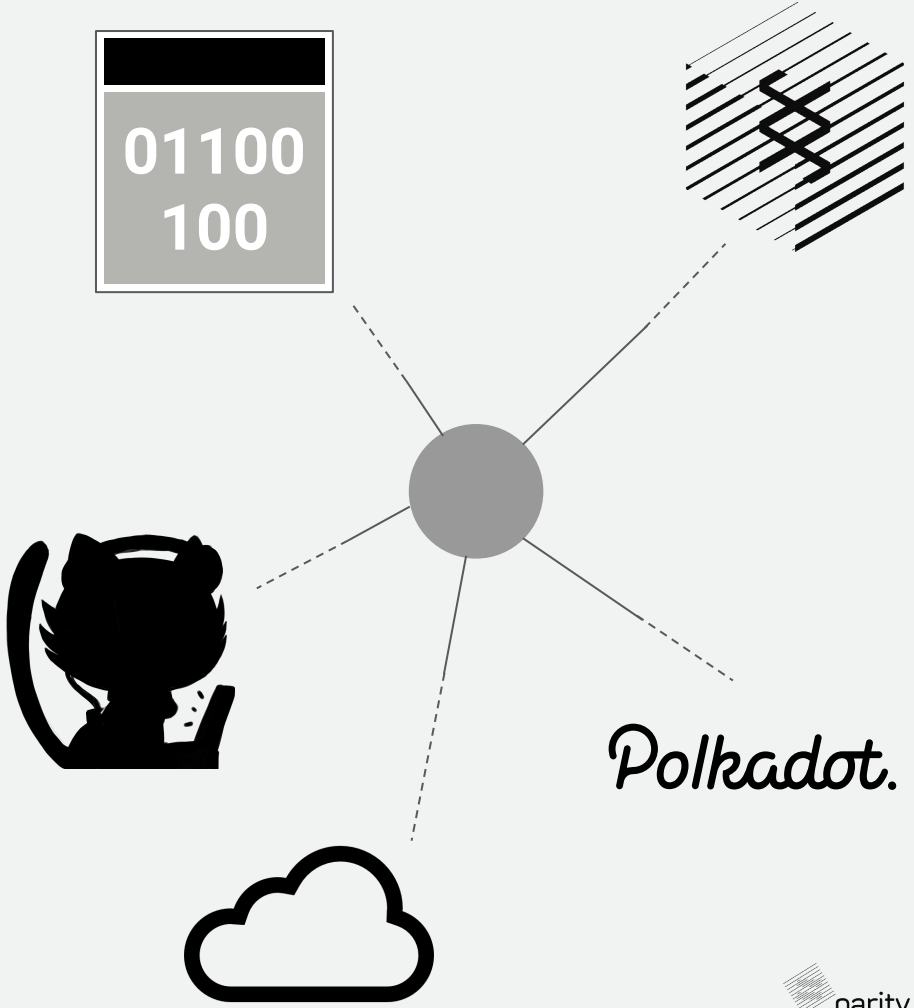
- Substrate & Polkadot Overview

  Substrate & Polkadot 概览
- Deploy a private blockchain network together

  部署一个私人区块链网络
- Basics of Substrate runtime development

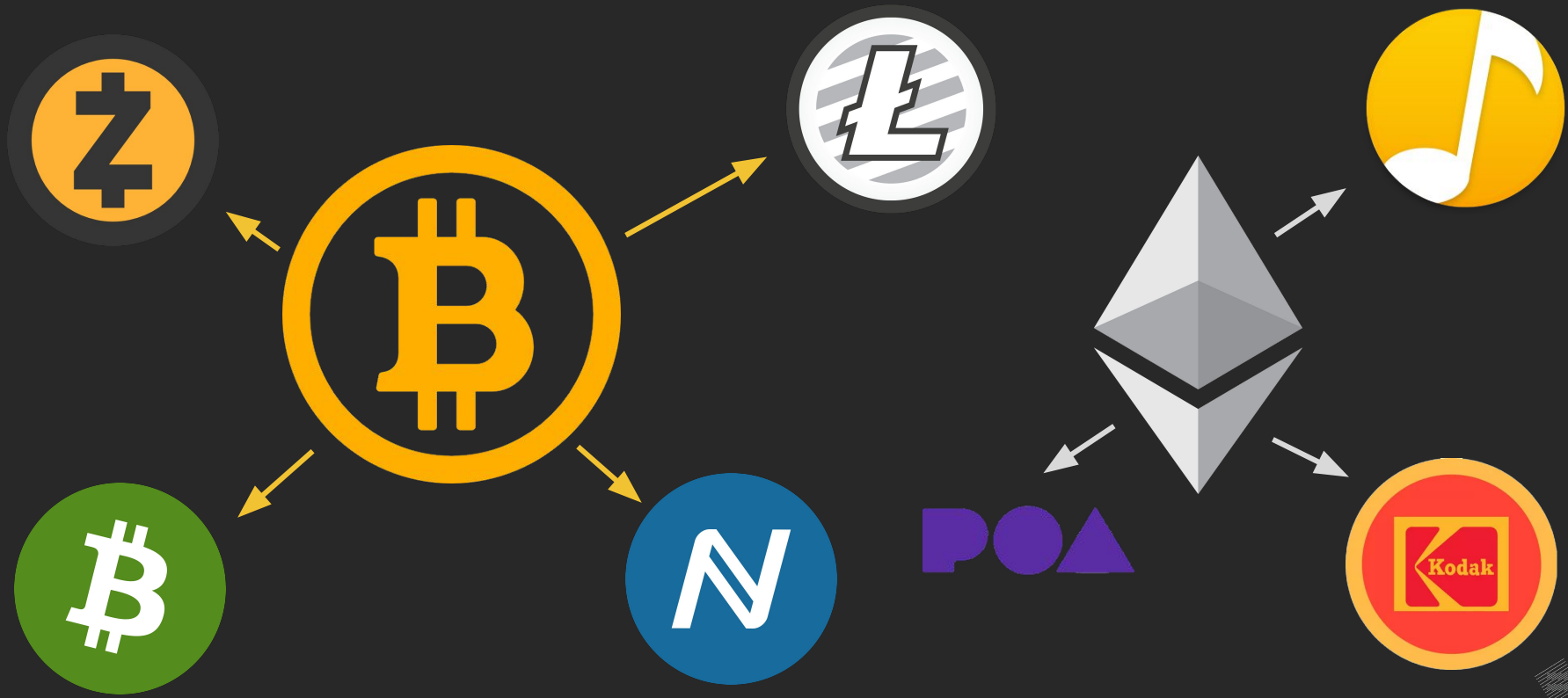
  Runtime开发基础
- Change & deploy blockchain code without forking!

  更改 & 部署区块链代码(无分叉！)



parity

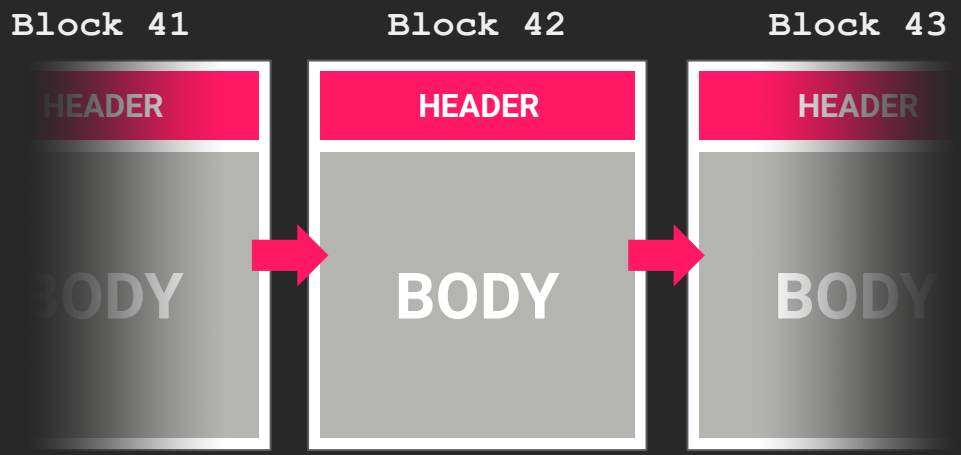
How does blockchain development happen today? 区块链开发现状



An oversimplified blockchain stack 简化后的区块链技术栈

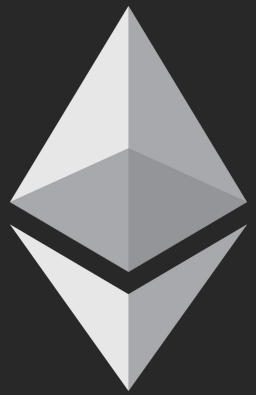
Blockchain nodes need
区块链节点组成

- Database
  数据库
- P2P Network
  点对点网络
- Consensus Algorithm
  共识算法
- Transaction Handling
  交易处理
- State Transition Function (Runtime)
  状态转移函数 (Runtime)
- Some special functionality: zk, sharding, etc?
  特殊功能如：zk，sharding，等等



parity

Parity has a lot of blockchain building experience...
Parity 拥有多个区块链网络的构建经验...

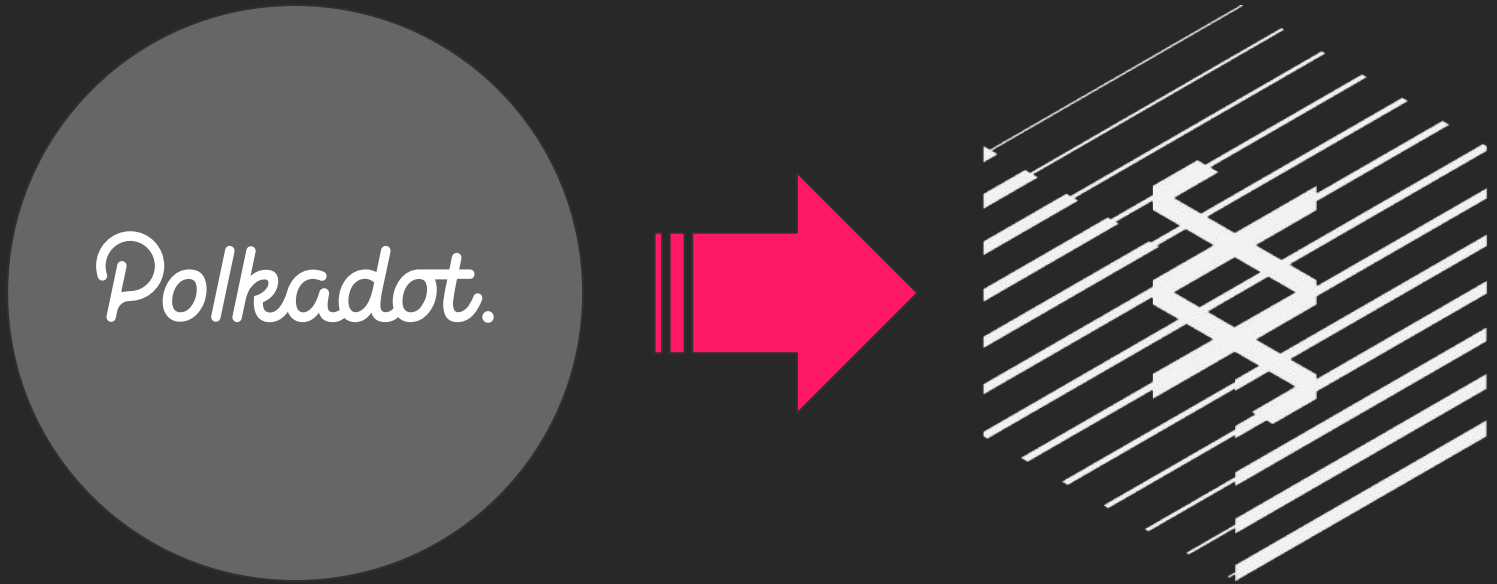


github.com/paritytech/
parity-ethereum

github.com/paritytech/
parity-bitcoin

github.com/paritytech/
polkadot

From Polkadot, came Substrate.
在 Polkadot 开发之中，诞生了 Substrate.



What is Substrate?

Substrate is an **open source**, **modular**, and **extensible** framework for building blockchains.

Substrate 是一个开源、模块化和可扩展的区块链构建框架。

What is Substrate?

Substrate provides all the core components of a Blockchain:

Substrate网络提供了构成区块链的所有核心组件

- Database Layer 数据层
- Networking Layer 网络层
- Transaction Queue 通信队列
- Consensus Engine 共识模型
- Library of Runtime Modules runtime模块库

Each of which can be customized and extended. 每一个组件都可以自定义和扩展

parity

The Substrate Runtime

The runtime is the **block execution logic** of the blockchain, a.k.a. the State Transition Function.

It is composed of **Runtime Modules**

Substrate Runtime Module Library (SRML)			
assets	babe	balances	collective
contract	democracy	elections	grandpa
indices	grandpa	indices	membership
offences	session	staking	sudo
system	timestamp	treasury	and more...

Runtime 是区块链网络的**区块执行逻辑**，也被称为状态转化函数。

由多个 **Runtime 模块组成**。

What is Polkadot?

Polkadot is a network that **connects blockchains**.

A **vision** toward a multi-chain future.

Polkadot 是在多链世界连接不同区块链的网络

What is Polkadot?

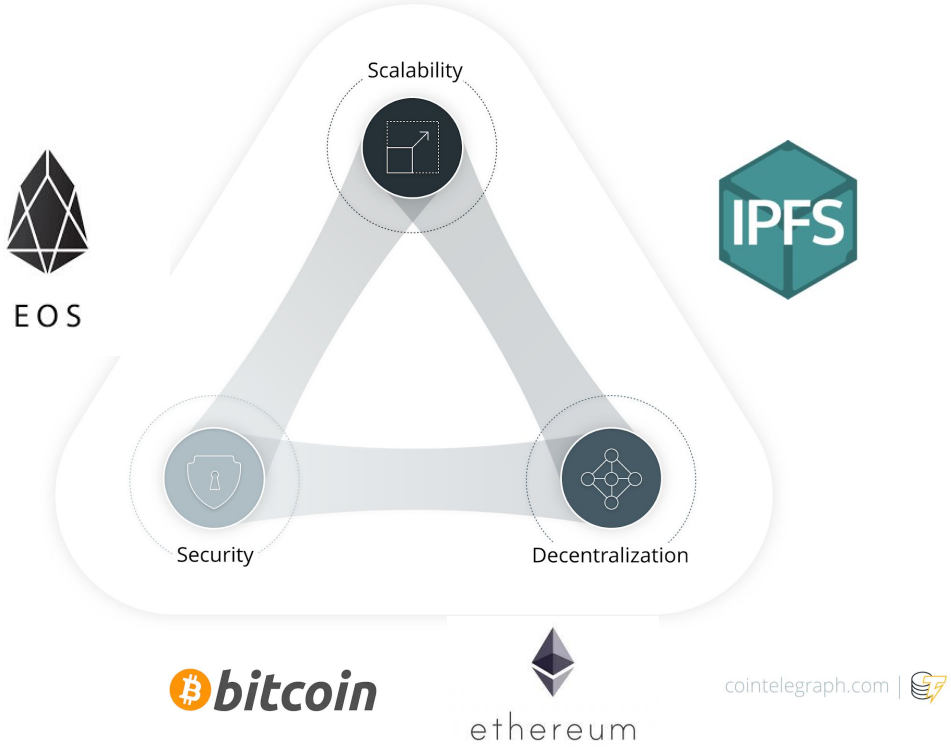
Polkadot attempts to solve three problems:

1. **Interoperability**
2. **Scalability**
3. **Shared Security**

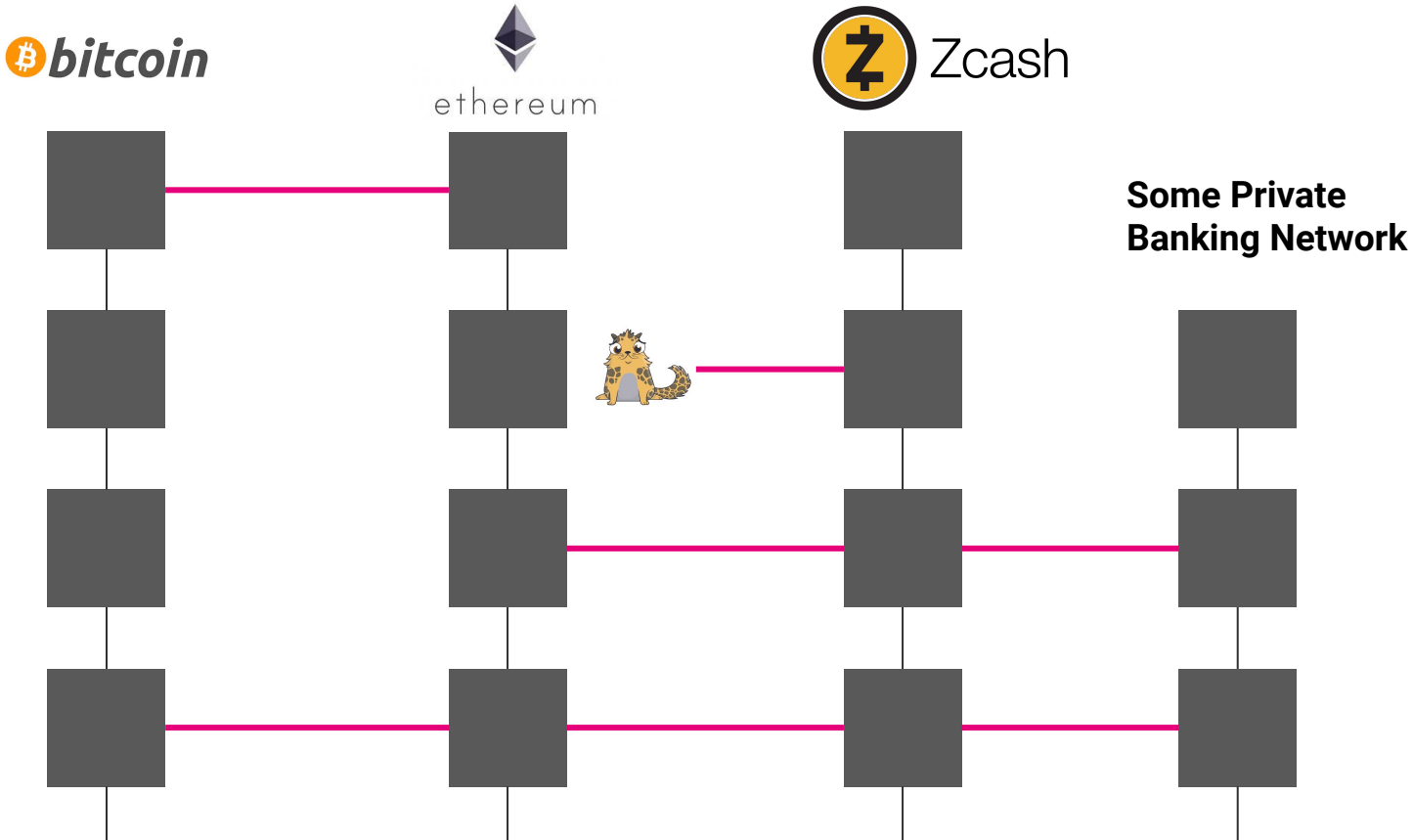
Polkadot 试图解决以下三个问题：

1. **互操作性**
2. **扩展性**
3. **共享安全**

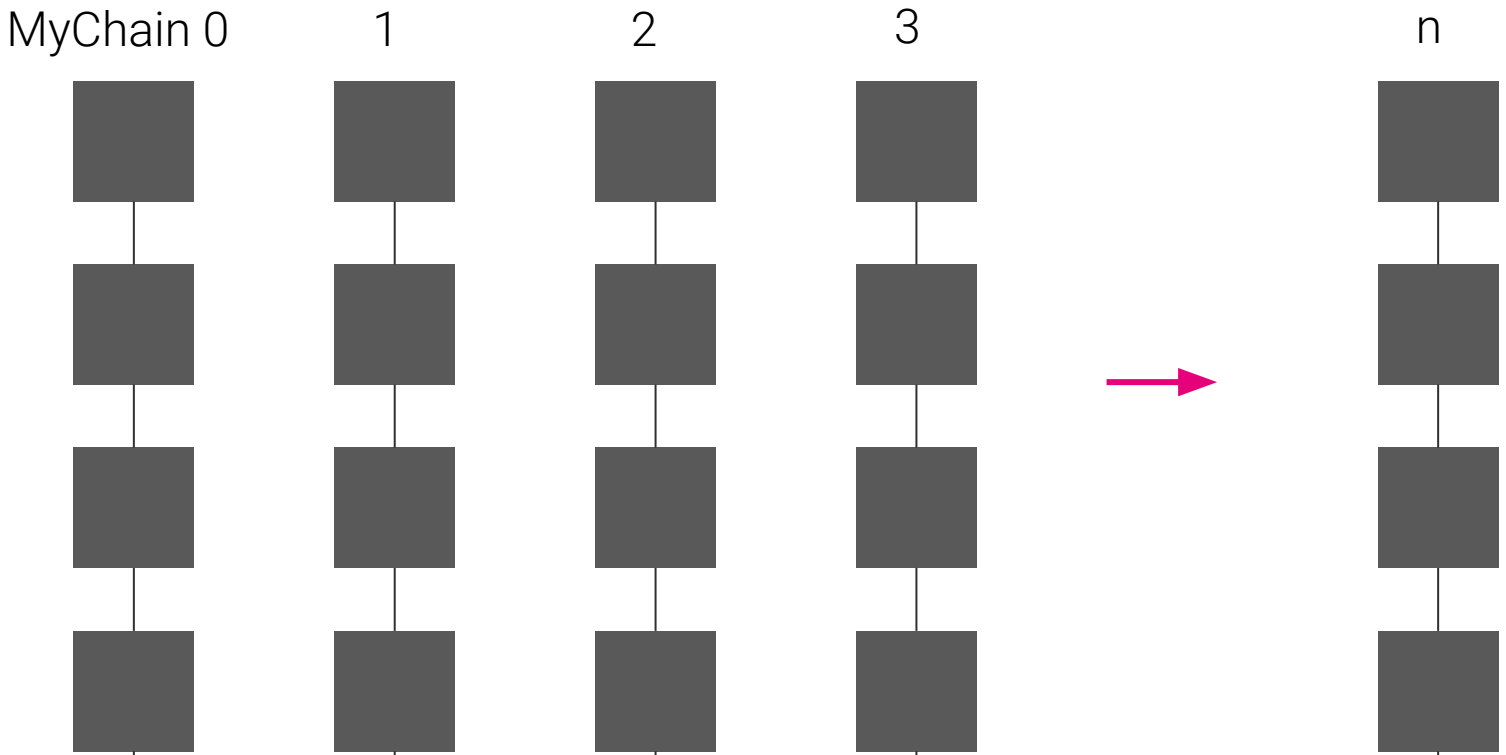
Traditional blockchains makes a distinct trade-off based on its use cases
一些传统区块链不得不为一些特殊的应用场景作出牺牲



Polkadot aims to scale the entire ecosystem
Polkadot的目标是让整个生态结合并扩大



As a result of interoperability, Polkadot also enables infinite scalability
在实现互通性的同时，Polkadot也解决了扩容问题



Takeaway:

Substrate builds blockchains

Polkadot is the network that connects these blockchains

要点:

Substrate 用于构建区块链

Polkadot 是连接这些区块链的网络

Specifically, what is inside Substrate? Substrate 具体包含了些什么？

Core Blockchain System
区块链核心

- Database
- Transaction Queue
- CLI
- Keyring
- RPC

Basic Blockchain Logic
区块链基础

- Primitives
- Balances
- Timestamp

Networking
网络通信

- Peer-Set Management
- Gossip Protocol Aggregator
- Distributed Hash Table

Consensus
共识算法

- Staking
- Babe
- Grandpa
- Finality Tracking

Governance
治理机制

- Democracy
- Elections
- Collectives
- Treasury

What does Polkadot add on top of Substrate?
在 Substrate 之上，Polkadot 添加了什么？

Parachains
平行链

Collators
采集人

Inter Chain Message Pass
跨链通信

Gossip Protocol
流言协议

Persistent Availability Store
数据存储

Parathreads
平行线程

Crowd Funding
众筹

Claims
确权

Auctions
拍卖

Registrar
注册

Substrate

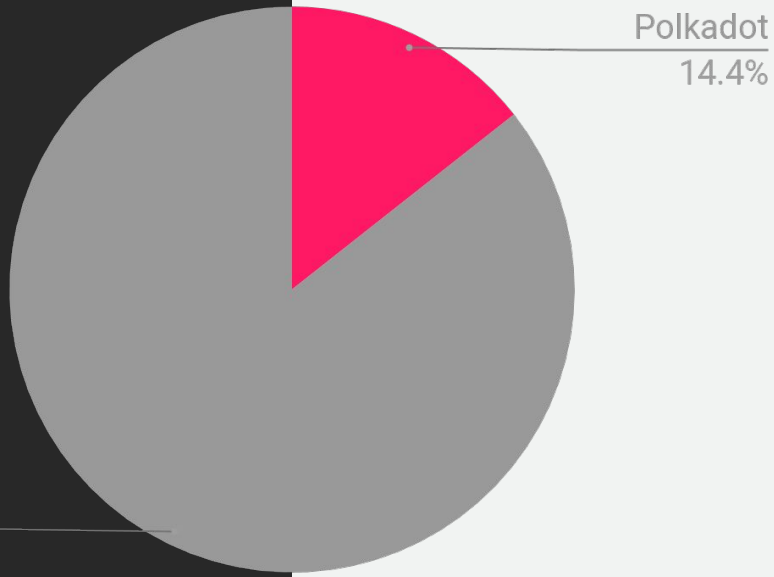
188,700

Lines of Code

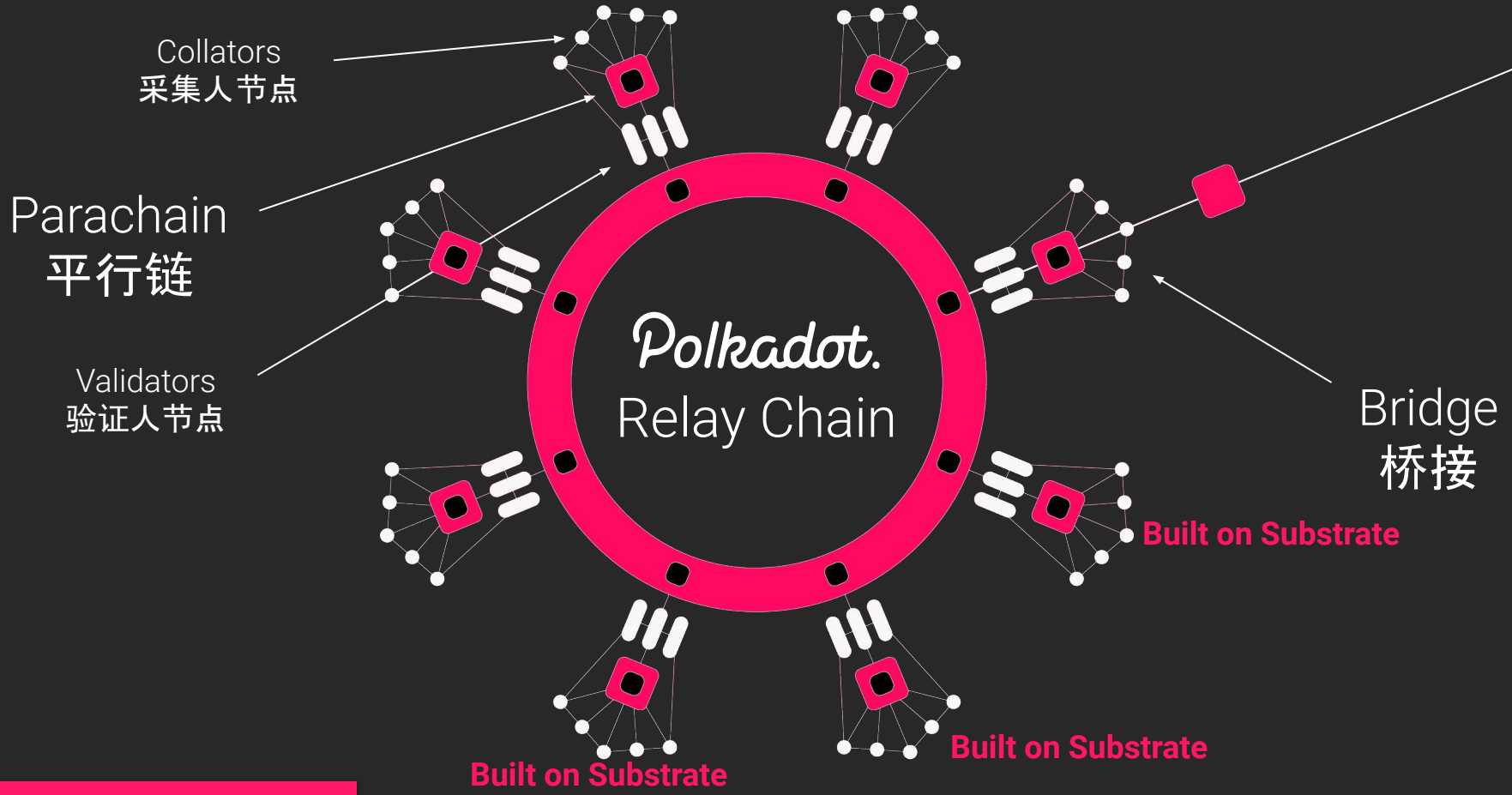
代码行数

Polkadot

31,700



parity



Components of Polkadot

parity

Exercise 1: Deploying a Substrate blockchain to a private network

练习 1: 部署一个 Substrate 区块链 到私人网络

Workshop Related Material & Code 相关的代码和连接

http://bit.ly/workshop666

parity

Step 1: Start as a boot node “Alice”创建初始节点

1. git clone -b v1.0 https://github.com/paritytech/substrate
2. cargo install --force --path subkey subkey
3. cd node-template && ./scripts/init.sh && ./scripts/build.sh
4. cargo build --release

Step 1: Start as a boot node “Alice”创建初始节点

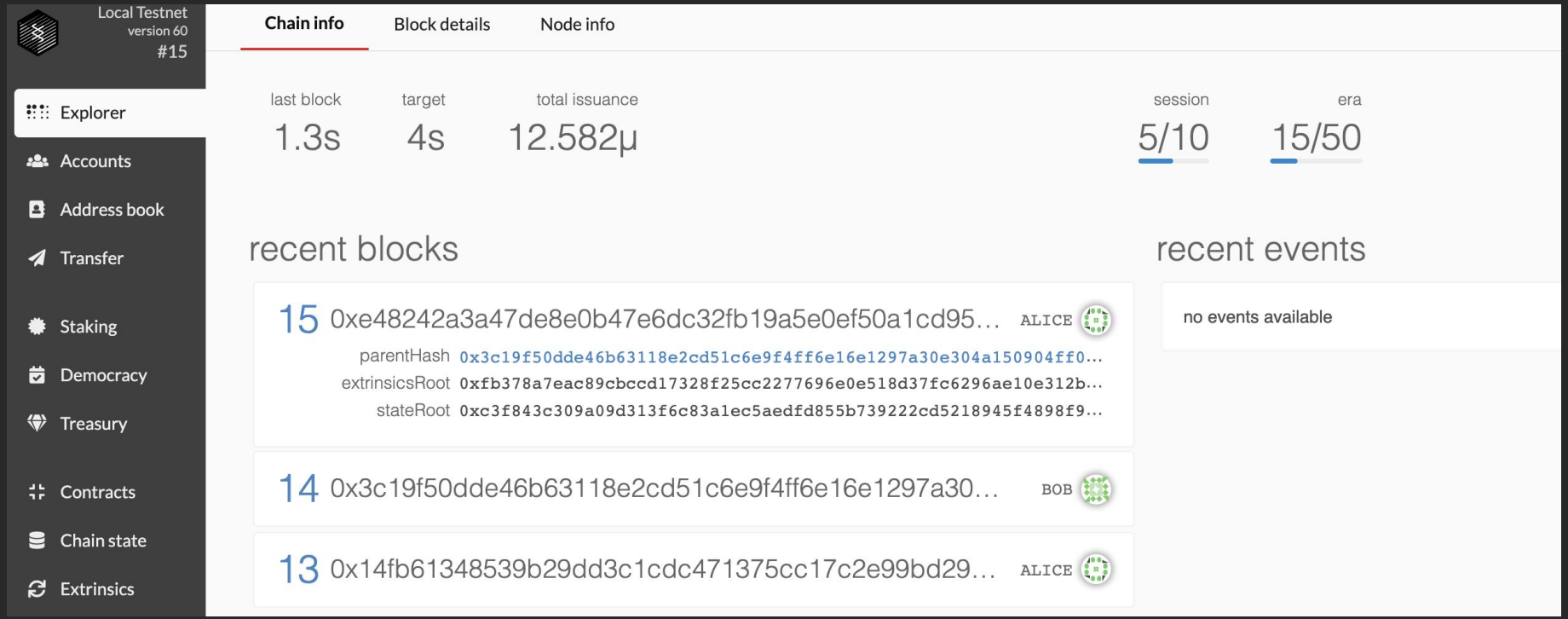
```
# If you're still in node-template/
cd ..

# Start the node
./target/release/node-template \
  --base-path /tmp/alice \
  --chain=local \
  --key //Alice \
  --port 30333 \
  --telemetry-url ws://telemetry.polkadot.io:1024 \
  --validator \
  --name AlicesNode
```

Step 2. Second node “Bob” joins in 第二个节点加入

```
./target/release/node-template \
  --base-path /tmp/bob \
  --chain=local \
  --key //Bob \
  --port 30333 \
  --telemetry-url ws://telemetry.polkadot.io:1024 \
  --validator \
  --name BobsNode \
  --bootnodes /ip4/<Alices IP Address>/tcp/<Alices
Port>/p2p/<Alices Node ID>
```

Step 3. Check the State of your blockchain 检查出块状态



parity

Takeaway: Substrate, out of the box, is a working blockchain!

要点: Substrate 是一个开箱即用、可运行的区块链！

Basics of Runtime Development

Runtime 开发基础知识

Verify First, Write Last 先验证，再写入

- A “bad transaction” does not work the same as Ethereum
- Ethereum: State is reverted, storage is untouched, and a fee is paid
- Substrate: State changes will persist if an `Err` is returned
- Needed for situations like:
  - Increasing Account transaction nonce, even with failed transactions
  - Charging transaction fees even when “out of gas”
- Need to be conscious of this pattern when making “sub-functions”

- “bad transaction” 的处理和 Ethereum 不同
- Ethereum: 状态被回滚，存储未修改，并支出交易费用。
- Substrate: 如果返回了 `Err`，状态的改变将会持久化
- 对于某些情况需要 这样做，如：
  - 出现失败的交易，也会增加 Account 交易随机数
  - 尽管 “out of gas”，也要收取交易 费用
- 编写“子函数”时，需要格外注意 这种模式

parity

Skeleton of a Module 框架

```
use support::{decl_module, decl_storage, decl_event,...};
pub trait Trait: system::Trait {...}

decl_storage! {...}
decl_module! {...}
decl_event! {...}
impl<T: Trait> Module<T> {...}
```

Importing and Defining Generic Types 导入和定义类型

```
pub trait Trait: system::Trait {
    type Event: From<Event<Self>> + Into<<Self as system::Trait>::Event>;
}
```

...and it inherits from `system::Trait`:

```
// From `system` SRML Module
pub trait Trait: 'static + Eq + Clone {
    type Origin: ...
    type BlockNumber: ...
    type Hash: ...
    type AccountId: ...
    … and more
}
```

parity

Declaring Storage 声明状态

```
decl_storage! {
    trait Store for Module<T: Trait> as TemplateModule {
        // Here we are declaring a StorageValue, `SomeValue` as a u32
        // `get(some_value)` defines a getter function
        // Getter called with `Self::some_value()`
        SomeValue get(some_value): u32;
        // Here we are declaring a StorageMap from an AccountId to a Hash
        // Getter called with `Self::some_map(account_id)`
        SomeMap get(some_map): map T::AccountId => u32;
    }
}
```

parity

Declaring Events 声明事件

```
decl_event!(
    pub enum Event<T>
    where
        <T as system::Trait>::AccountId
    {
        // Event `ValueStored` deposits values of type `AccountId` and `u32`
        ValueStored(AccountId, u32),
    }
);
```

Declaring Dispatchable Functions 声明可调用方法

```
decl_module! {
  pub struct Module<T: Trait> for enum Call  where origin: T::Origin {
    fn deposit_event<T>() = default;  // The default deposit_event definition

    pub fn store_value(origin, input: u32) -> Result {
      let sender = ensure_signed(origin)?; // Check for transaction
      <SomeValue<T>>::put(input); // Put a value into a StorageValue
      <SomeMap<T>>::insert(sender, input); // Insert key/value in StorageMap
      Self::deposit_event(RawEvent::ValueStored(sender, input)); // Emit
Event
      Ok(()) // Return Ok at the end of a function
}}}
```

parity

Declaring Public and Private Functions 声明外部内部函数

```
impl<T: Trait> Module<T> {
  fn mint(to: T::AccountId, id: T::Hash) -> Result {}
  fn transfer(from: T::AccountId, to: T::AccountId, id: T::Hash) -> Result
{}
}
```

Note: These can also be called from other Rust modules if made `pub`. It is not a Substrate “public” function. Substrate public functions are “dispatchable” fns

一旦声明作为外部函数之后，他们甚至可以从其他的Rust module来调用

Defining a Custom Struct 创建自定义struct

```
use parity_codec::{Encode, Decode};

#[derive(Encode, Decode, Default, Clone, PartialEq)]
#[cfg_attr(feature = "std", derive(Debug))]
pub struct Kitty<Hash, Balance> {
    id: Hash,
    price: Balance,
    gen: u64,
}
```

Common question: What is the difference between a runtime and a smart contract?

FAQ: 智能合约和runtime的区别？

Exercise 2: Build a simple proof of existence blockchain

练习2: 创建一个简单的存在性证明区块链

PoE: online service that verifies existence of files 证明文件是否存在的在线服务

Proof of Existence is an online service that verifies the existence of computer files as of a specific time via timestamped transactions in the bitcoin blockchain.

As mentioned, the Proof of Existence service was originally created on top of the Bitcoin blockchain. However, this functionality is not properly supported by the Bitcoin protocol, and the existing service uses more of a hack on top of other Bitcoin functions.

Introducing new functionality to a blockchain like Bitcoin is incredibly difficult since it is designed for a single purpose

存在性证明是一个通过在比特币网络上发送一个加入时间戳的交易，来证明电脑上文件在特定时间是否存在的线上服务。

POE最初是基于比特币网络的，但是，他的功能不能很好的被比特币网络支持，主要依托应用层用不同的hack方法来实现。

而给像比特币这样的区块链引入新的功能又很困难，因为这些功能太专，不适合于修改基础协议。

parity

Now we're ready to code the custom module!
现在让我们开始创建自定义module！

```
// 1. Imports
use support::{decl_module, decl_storage, decl_event,...};

// 2. Module Configuration
pub trait Trait: system::Trait {...}

// 3. Module Events
decl_event! {...}

// 4. Module Storage Items
decl_storage! {...}

// 5. Callable Module Functions
decl_module! {...}
```

parity

1. Import necessary dependencies 导入必要的依赖模块

You can start by copying this at the top of your empty poe.rs file:

第一步可以在新建的poe文件开头加入：

```
use support::{decl_module, decl_storage, decl_event, ensure, StorageMap};
use rstd::vec::Vec;
use system::ensure_signed;
```

2. Configure your module to emit events
设置使module可以发送事件

For now, the only thing we will configure about our module is that it will emit some Events.

现在，我们只需要添加一个event模块

```
/// The module's configuration trait.
pub trait Trait: system::Trait {
    /// The overarching event type.
    type Event: From<Event<Self>> + Into<<Self as system::Trait>::Event>;
}
```

3. Define your module's events
定义module中的事件

```
// This module's events.
decl_event!(
    pub enum Event<T> where AccountId = <T as system::Trait>::AccountId {
        // Event emitted when a proof has been stored into chain storage
        ProofStored(AccountId, Vec<u8>),
        // Event emitted when a proof has been erased from chain storage
        ProofErased(AccountId, Vec<u8>),
    }
);
```

4. Add the storage/state items
添加状态存储

```
// This module's storage items.
decl_storage! {
        trait Store for Module<T: Trait> as PoeStorage {
        // Define a 'Proofs' storage item for a map with
        // the proof digest as the key, and associated AccountId as value.
        // The 'get(proofs)' is the default getter.
                Proofs get(proofs): map Vec<u8> => T::AccountId;
        }
}
```

5. Add your callable [public] module functions
添加可调用的公共方法/函数

```
// The module's dispatchable functions.
decl_module! {
    /// The module declaration.
    pub struct Module<T: Trait> for enum Call where origin: T::Origin {
        // A default function for depositing events
        fn deposit_event() = default;

        // Allow a user to store an unclaimed proof
        fn store_proof(origin, digest: Vec<u8>) {
            // Verify that the incoming transaction is signed
            let sender = ensure_signed(origin)?;

            // Verify that the specified proof has not been claimed yet
            ensure!(!Proofs::<T>::exists(&digest), "This proof has already been claimed");

            // Store the proof and the claim owner
            Proofs::<T>::insert(&digest, sender.clone());

            // Emit an event that the claim was stored
            Self::deposit_event(RawEvent::ProofStored(sender, digest));
        }

        // Allow the owner of a proof to erase their claim
        fn erase_proof(origin, digest: Vec<u8>) {
            // Determine who is calling the function
            let sender = ensure_signed(origin)?;

            // Verify that the specified proof has been claimed
            ensure!(Proofs::<T>::exists(&digest), "This proof has not been stored yet");

            // Get owner of the claim
            let owner = Self::proofs(&digest);

            // Verify that sender of the current call is the claim owner
            ensure!(sender == owner, "You must own this proof to erase it");
```

parity

6. Add module into Runtime
把module添加到runtime

```
# Step 1: Add module into runtime
mod poe;
```

```
# Step 2: Add the trait implementation
impl poe::Trait for Runtime {
        type Event = Event;
}
```

```
# Step 3: Update `construct_runtime`
construct_runtime!(
  pub enum Runtime where
  Block = Block,
  NodeBlock = opaque::Block,
  UncheckedExtrinsic = UncheckedExtrinsic
  {
    ...
    POEModule: poe::{Module, Call, Storage, Event<T>},
  }
);
```

parity

7. Compile! 编译！

If you were able to copy all of the parts of this module correctly into your `poe.rs` file and update `lib.rs`, you should be able to recompile your node successfully!

如果把以上的模块都放入poe.rs文件中，并正确的修改lib.rs，那么这个时候就可以编译成功了

```
cargo build --release
```

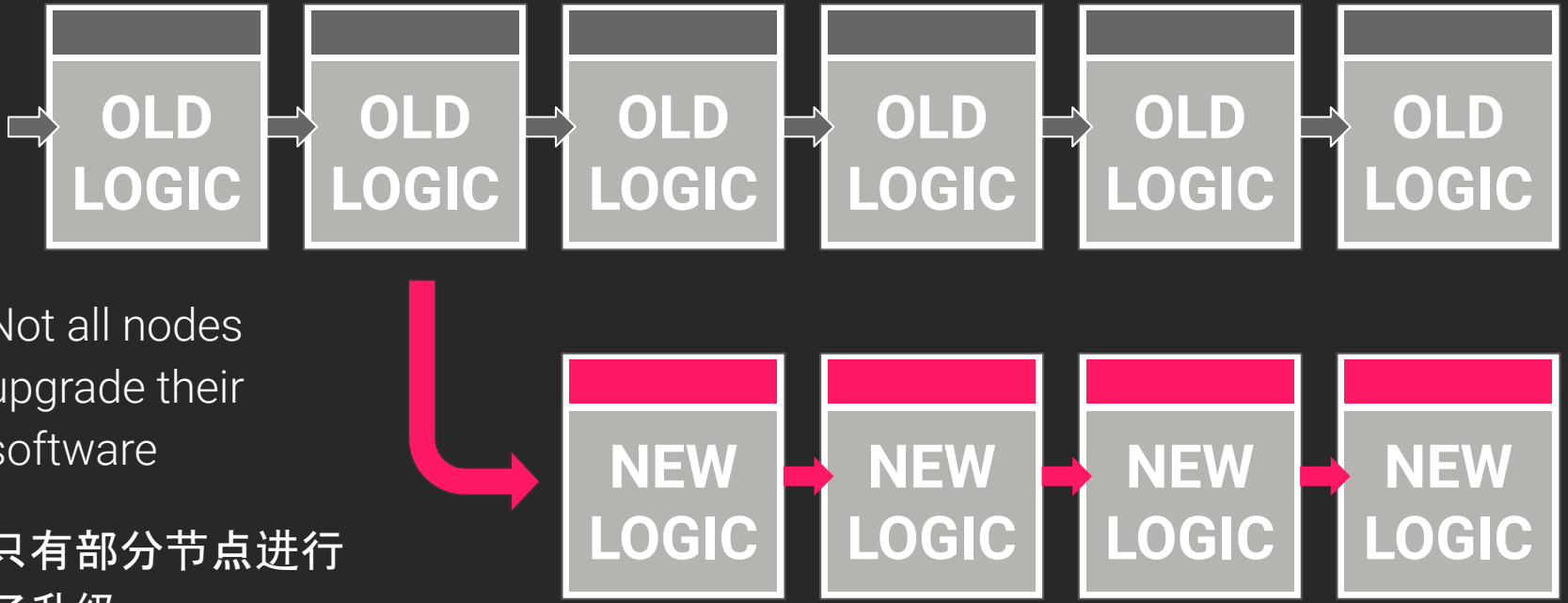
parity

Exercise 3: Perform a forkless upgrade to our blockchain!

Exercise 3: 进行区块链网络无分叉升级

parity

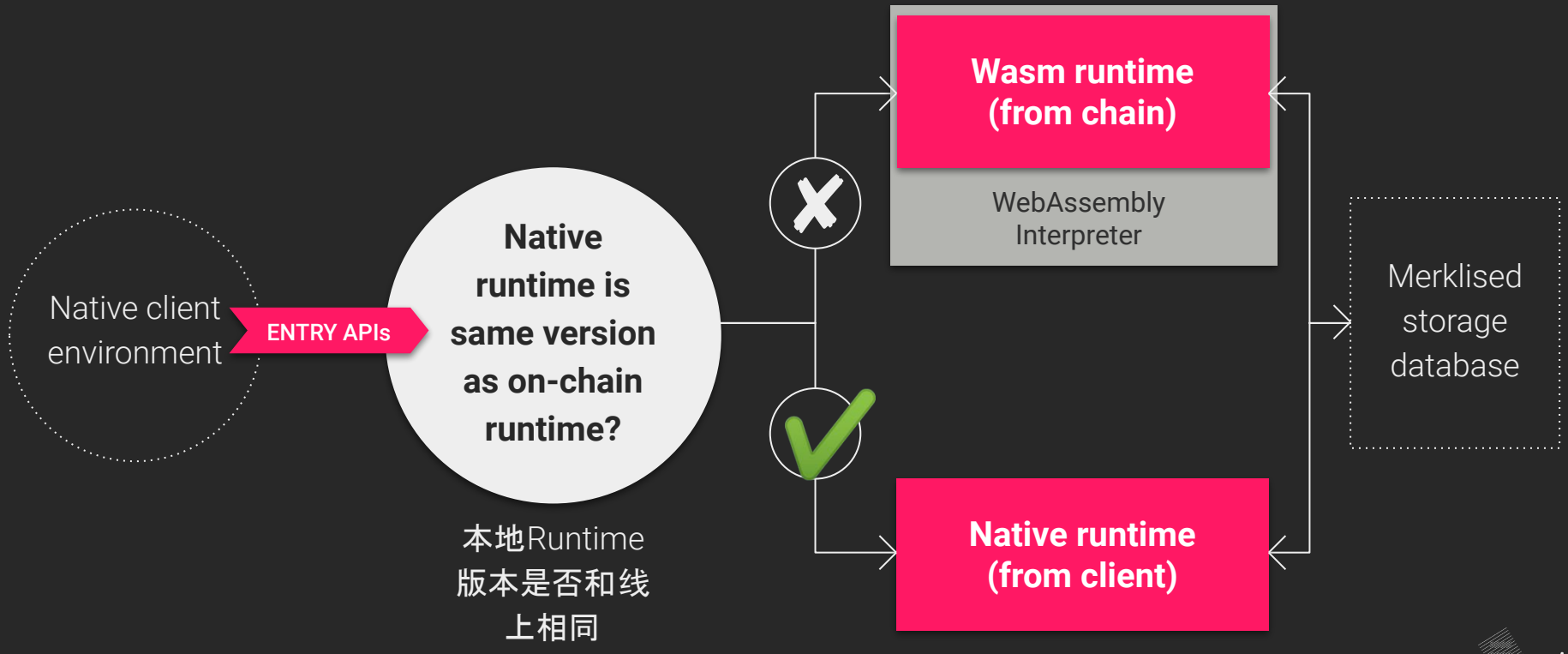
Hard Fork Upgrades 硬分叉升级



Not all nodes upgrade their software

只有部分节点进行了升级

Forkless Runtime Upgrades 无分叉 Runtime 升级



A Need for Upgrades 升级的必要性

- Fix important security vulnerabilities
- Change core rules in the protocol
- Add new functionality
- Repair chain state

➢ Hard fork upgrades require a lot of coordination

➢ Unclear governance/signaling around upgrades

➢ 修复重要的安全漏洞

➢ 更改协议中的核心规则

➢ 添加新功能

➢ 修复链上状态

➢ 硬分叉升级需要大量的协作

➢ 关于升级的治理和信号不明确

parity

Governing Runtime Upgrades Runtime 升级治理

- Runtime code is accessible through on-chain governance
- Sudo Module
- Democracy Module
- Your own module and logic
- **Runtime Upgrades are Optional**

- 可以通过 链上治理 访问Runtime 代码
- Sudo 模块
- Democracy 模块
- 自定义的模块和功能
- **Runtime 升级是可选的**

parity

Important Note: Wasm, if one chain upgrades interface, the rest do not need to upgrade. If not wasm, if one chain upgrades interface, the entire network needs to perform an upgrade.

Let’s do an upgrade...

Live Demo

parity

Takeaway: Substrate forklessly upgrades your blockchain!

要点: Substrate 无分叉地升级你的区块链！

Questions or feedback?

tiny.cc/substrate-technical

parity

Next Steps For You!

- ✓ Clone and follow instruction from the Substrate Package
  - ➢ tiny.cc/substrate-package
- ✓ Join and ask questions in the Substrate Technical channel on Riot
  - ➢ tiny.cc/substrate-technical
- ✓ Explore and read the Substrate Runtime Module Library
  - ➢ tiny.cc/substrate-srml
- ✓ BUILD ON SUBSTRATE!

Helpful Links

- tiny.cc/substrate-getting-started
- tiny.cc/substrate-package
- tiny.cc/substrate-docs
- tiny.cc/substrate-workshop
- tiny.cc/substrate-technical

We are hiring!

nicole@parity.io

parity