# Substrate, Proof of Work and Consensus Engines

Wei Tang

17 Sep, 2019

# Outline

Section 1

# Introduction

# Introduction

- Wei Tang
- Rust developer at Parity Technologies
- Ethereum, Shasper
- Substrate

# Substrate

- More details in the next talk!
- Substrate is a framework for building blockchains.
- Designed to work with Polkadot (and Polkadot is built on top of it!).

# Substrate

Everything that a blockchain project needs:

- Database backend, state backend, runtime executor
- Transaction pool, block proposer
- RPC, CLI, keyring
- Networking

Composible libraries that you can use to build your blockchains:

- Consensus protocols
- Many runtime modules!

# Substrate

Make building blockchains easy, and the goal is to make it possible to build (nearly) all blockchain ideas on top of it.

Section 2

# Proof of Work

# Blockchain

There are several primary goals for a blockchain:

- For participants to reach consensus on *something*.
- Make it possible for anyone to audit how the consensus is reached, and convince themselves that the consensus is reached.

Blockchain (chain of blocks) is one way to accomplish this:

- Define a state, and how to transit from one state to another.
- Group state transitions into blocks. A sequence of blocks form a chain.
- Define a way to figure out which chain is canonical that everyone can agree.

# Blockchain

- State execution (runtime)
- Consensus

# Consensus

- Validation
- Fork choice
- Finalization

# Proof of Work

- Find solutions to given problem (defined by `(pre_hash, nonce)`, where it should only be able to be solved by ”brute-force”.

```
type Hash = H256;

/// Given the block pre-hash, and a random nonce, find a seal.
fn mine(pre_hash: &Hash, nonce: &Hash) -> Seal;
/// Given the pre-hash, nonce and seal, verify its validity.
fn verify(pre_hash: &Hash, nonce: &Hash, seal: &Seal) -> bool;
```

# Proof of Work

- Hash functions (Hashcash PoW). The cost of verification is the same as one proof attempt.
- Graph, prime numbers.

Section 3

# Proof of Work on Substrate

# Substrate Consensus Engines

- Aura, BABE
- Grandpa (Finality)
- Casper
- RHD (BFT consensus)
- PoW

# Substrate Consensus Engines

What PoW?

# Substrate Proof of Work

Generic PoW:

```
pub trait PowAlgorithm<B: BlockT> {
  fn difficulty(&self, parent: &BlockId<B>) -> Result<Difficulty, String>;
  fn verify(
    &self, parent: &BlockId<B>, pre_hash: &H256, seal: &Seal, difficulty:
        Difficulty,
  ) -> Result<bool, String>;
  fn mine(
    &self, parent: &BlockId<B>, pre_hash: &H256, seed: &H256, difficulty:
        Difficulty, round: u32,
  ) -> Result<Option<Seal>, String>;
}
```

# Consensus and Runtime

- Runtime: executing states
- Consensus: choosing chains of blocks

# Consensus and Runtime

Observation:

- We can gain flexibility and online upgradability by allowing consensus to use data from runtime.
- In other words, allowing consensus engine to access data from state.

# Consensus and Runtime

For Proof of Work, we can implement flexible difficulty adjustment algorithm, by deferring the definition to runtime.

```
fn difficulty(&self, parent: &BlockId<B>) -> Result<Difficulty, String> {
  self.client.runtime_api().difficulty(parent)
    .map_err(|e| format!("Fetching difficulty from runtime failed: {:?}", e));
}
```

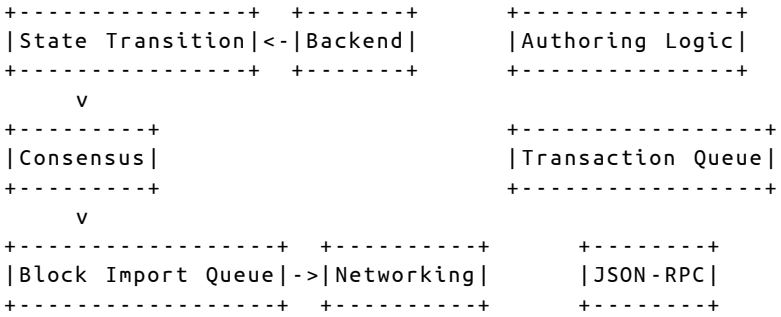
# Consensus and Runtime

However, it's not safe to defer the seal verification to runtime in current Substrate design!

Section 4

# Writing an Engine for Substrate

# Overview


State Transition
Backend
Authoring Logic
Consensus
Transaction Queue
Block Import Queue
Networking
JSON-RPC

# Writing an Engine

Block verification:

- Define an import queue, or use the `BasicQueue` with a `Verifier`.

Block proposal:

- Any function that takes `BlockImport`, and import blocks continuously.
- You usually want to make it a `Future`.

Section 5

# Conclusion

# Conclusion

- We're hiring!
- jobs@parity.io