Serenity on Substrate

Inside Shasper, Parity's current Serenity codebase

Wei Tang

24 January, 2019

Outline

1. Overview
2. Runtime
3. Consensus
4. The rest

Introduction

- Wei Tang, Rust developer at Parity Technologies.
- Shasper, a Serenity beacon chain client built on top of Substrate.

Disclaimer

- Only about beacon chain. Not (yet) sharding!
- Our code base is still 80% spec 2.1, but this presentation is based on the current newest spec!
- Not about beacon chain spec, but Shasper's implementation details.
- "Very technical", but not.
- 90% correct.

Introduction

- Serenity: "Ethereum 2.0", Proof of Stake, scale via sharding, beacon chain.
- Substrate: "General blockchain framework", Rust, WebAssembly.

Serenity

- Beacon chain as phrase 0.
- Bare minimal things needed to get a proof of stake chain going, and lay out foundations for shards.

Serenity

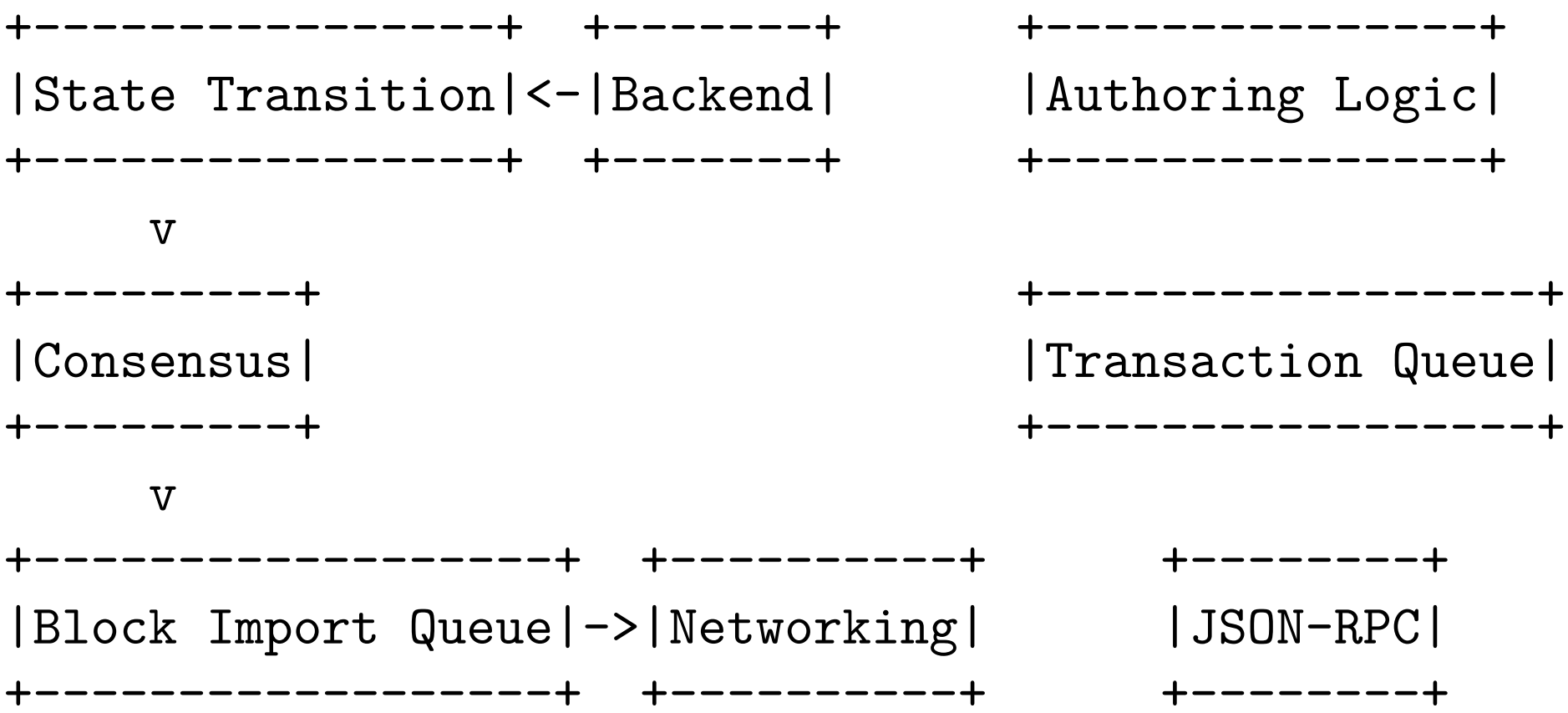
- One-way transfer ("burn") from eth1 to beacon chain.
- Store validator states and balances, handle rewards.
- Validator making attestations, so as to justify and finalize blocks.
- Slashing conditions for proposers and validators.
- Casper fork choice rule.
- RANDAO for selecting committes and proposers.

Serenity

- Beacon chain, no sharding.
- No one makes transactions yet!
- A new blockchain!

Blockchain framework

We need to write a new blockchain client, now what?



Blockchain framework

Wait! We have wrote them already!

- In Parity Ethereum.
- Re-using the code is not easy.
- The data structure is vastly different and nearly nothing is in common (except the hashing algorithm).
- Adding additional complexity into a not-so-simple codebase.

Substrate

We can start from scratch, or...

- Build it in Substrate!

Substrate

What we get:

- Light client and WebAssembly by default!
- All the hard parts that people "just want a good implementation": storage backend, networking, transaction queue, basic chain sync, basic JSON-RPC.
- General[1] blockchain framework, it has a lot more!

[1] Conditions apply.

Our current Shasper codebase

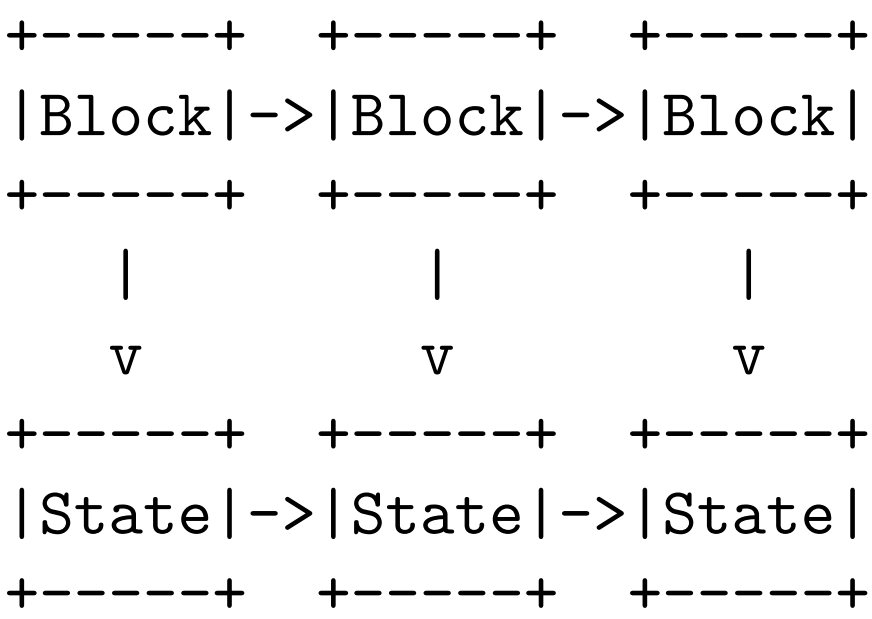
An experimental project:

- Located at github.com/paritytech/shasper
- Implemented three things: runtime, consensus, block authoring.
- To be implemented: validator logic.
- Other things can be handled in Substrate[2].

[2] Conditions apply.

Overview

Runtime, also known as state transition:



Overview

In Substrate, a runtime:

- Exposes several function that can be queryed.
- Talk with backend through externs.
- Runs both natively and through WebAssembly.
- WebAssembly byte code is written into the blockchain's state on genesis.
- State transition are deterministic[3].

[3] If you don't intentionally breaks the guarantees!

Overview

```
impl_runtime_apis! {
  impl Core<Block> for Runtime {
    fn version() -> RuntimeVersion { }
    fn authorities() -> Vec<ValidatorId> { }
    fn execute_block(Block) { }
    fn initialise_block(&Header) { }
  }

  impl BlockBuilder<Block> for Runtime {
    fn apply_extrinsic(Extrinsic) -> ApplyResult { }
    fn finalise_block() -> Header { }
    fn inherent_extrinsics(Data) -> Vec<Extrinsic> { }
  }
}
```

Assumptions in Substrate

Structure a block as follows:[4]

- Header
- Extrinsic

So that:

- Header can always be generated from previous block, and when the state transition function is finalized.
- Extrinsic contains all external outputs.

[4] There are more fields related to consensus and light client.

Serenity block structure

From eth2.0 spec:

```
{
    ## Header ##
    'slot': 'uint64',
    'parent_root': 'bytes32',
    'state_root': 'bytes32',
    'randao_reveal': 'bytes32',
    'eth1_data': Eth1Data,
    'signature': 'bytes96',

    ## Body ##
    'body': BeaconBlockBody,
}
```

Serenity block structure

From eth2.0 spec:

```
{
    'proposer_slashings': [ProposerSlashing],
    'casper_slashings': [CasperSlashing],
    'attestations': [Attestation],
    'custody_reseeds': [CustodyReseed],
    'custody_challenges': [CustodyChallenge],
    'custody_responses': [CustodyResponse],
    'deposits': [Deposit],
    'exits': [Exit],
}
```

Assumptions in Substrate

Problem: Header contains external outputs. Solution: Move them to extrinsic.

Extrinsic

Split extrinsic into two parts:

- Inherent extrinsic: usually only one per type. Can only be generated within the client. Slot, randao reveal, eth1 data.
- Transaction: can be received from the network. Attestations, slashings, deposits, etc.

Extrinsic

```
impl_runtime_apis! {
  impl BlockBuilder<Block> for Runtime {
    fn inherent_extrinsics(Data) -> Vec<Extrinsic> {
      extrinsics.push(slot_from_data(inherent));
    }
    fn apply_extrinsic(Extrinsic) -> ApplyResult {
      if extrinsic_index == 0 {
        assert!(validate_slot_extrinsic(e));
      }
    }
  }
}
```

Adapter design

When the block structure we store in database does not match the block structure in specification:

- Create adapter method that converts internal block structure into specification block structure.
- As long as hashes can be computed cheaply.
- Multi-trie[5].

[5] https://github.com/paritytech/substrate/issues/872

Casper

Brief overview of Casper FFG:

- Validators attest blocks to create supermajority links.
- Mark a block as justified if we find a supermajority link from another justified block.
- Mark a block as finalized if it's justified, and we have a supermajority link from this block to a direct child.
- Two properties that 2/3 validators must not violate. Slashing conditions.

Consensus in Shasper

- Slashing, attestation validation can all be validated by runtime.
- Additional information needed by consensus, but not defined by state transition can as well be runtime. This includes validators' latest attestations.

Fork Choice

Defined on top of state transition and backend.
Consensus layer chooses what to do completely

```
/// Block import trait.
pub trait BlockImport<B: BlockT> {
    type Error: ::std::error::Error + Send + 'static;
    fn import_block(&self,
        block: ImportBlock<B>,
        new_authorities: Option<Vec<AuthorityIdFor<B>>>
    ) -> Result<ImportResult, Self::Error>;
}
```

How to. . .

Validate eth1 data?[6]

- Use inherent extrinsic.
- Runtime won't validate whether the eth1 data is correct.
- Consensus layer handles the validation, and uses whatever sources it wants.
- May bundle the new client with Parity Ethereum, but still two underlying clients.

[6] Not yet implemented.

How to. . .

Deploy fork for Shasper?

- WebAssembly byte code written in state from genesis.
- Upgrade by inherent extrinsic!
- Consensus layer can require a particular fork byte code to be included at a particular block, otherwise, refuse to import it.

How to. . .

Implement networking?

- No known specification yet.
- Either through plugging in Substrate's network layer or creating a new network layer implementation.

Thank you

- Wei Tang, Rust developer at Parity Technologies.
- Reach me @sorpaas, hi@that.world
- Shasper codebase:
  `https://github.com/paritytech/shasper`
- Slides:
  `https://git.that.world/talks/20190124-shasper.git`