



Consensus in Substrate

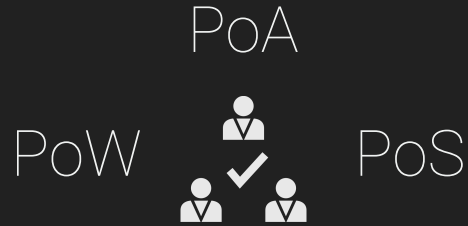
Robert Habermeier

robert@polkadot.io | @rphmeier

WHAT ARE BLOCKCHAINS COMPOSED OF?

CONSENSUS

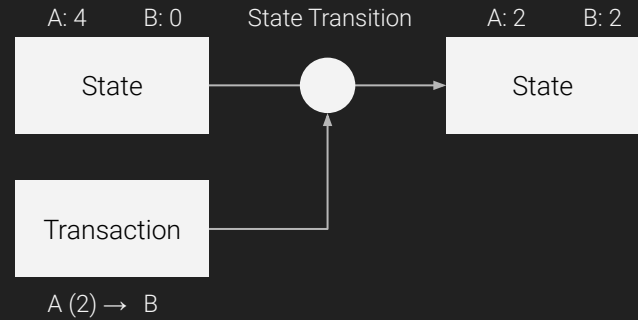
- How do we agree on what changes to include? In what order? What is “final”?
- PoW or PoS provide incentives for consensus



WHAT ARE BLOCKCHAINS COMPOSED OF?

STATE MACHINE

- What are the changes that are actually agreed upon?
- Transactions, balances, etc. all abstracted



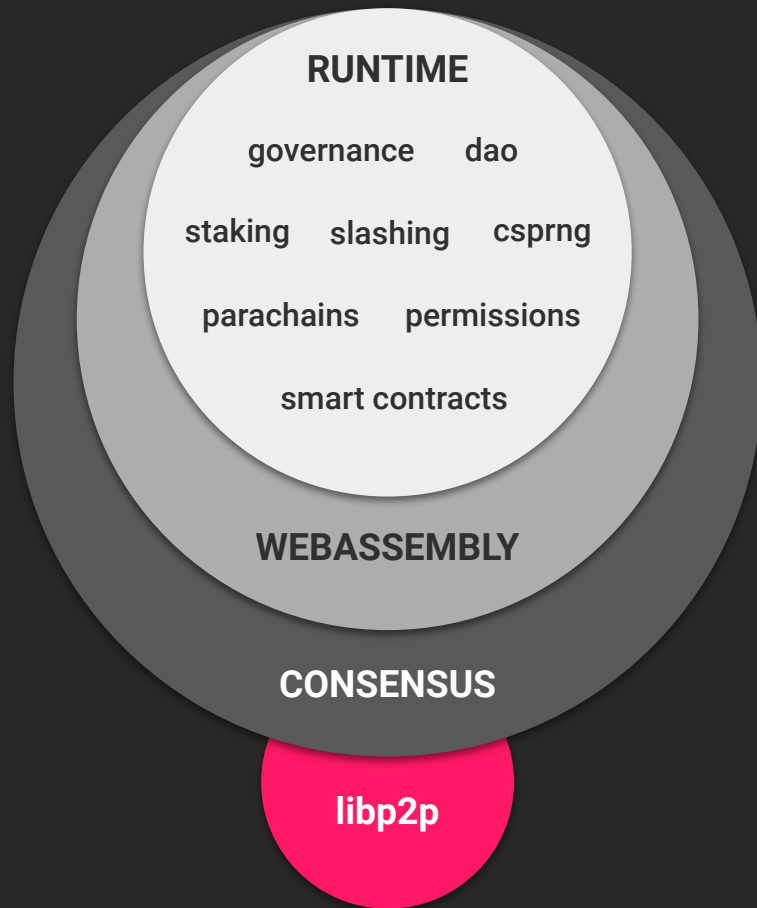
Written in Rust

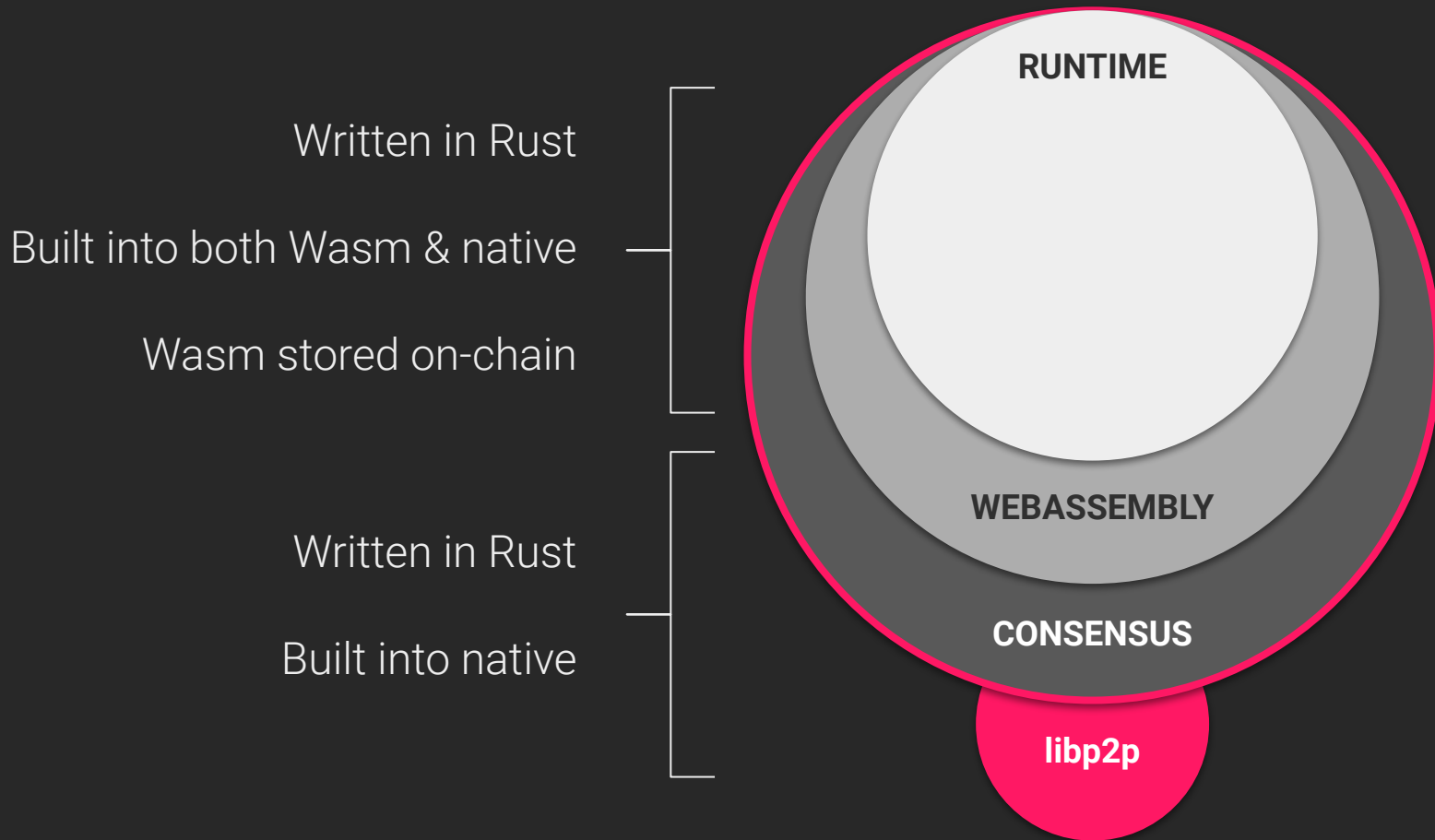
Built into both Wasm & native

Wasm stored on-chain

Written in Rust

Built into native





GRANDPA: The Grandfather of Finality Gadgets

(GHOST-based Recursive ANcestor Deriving Prefix Agreement)

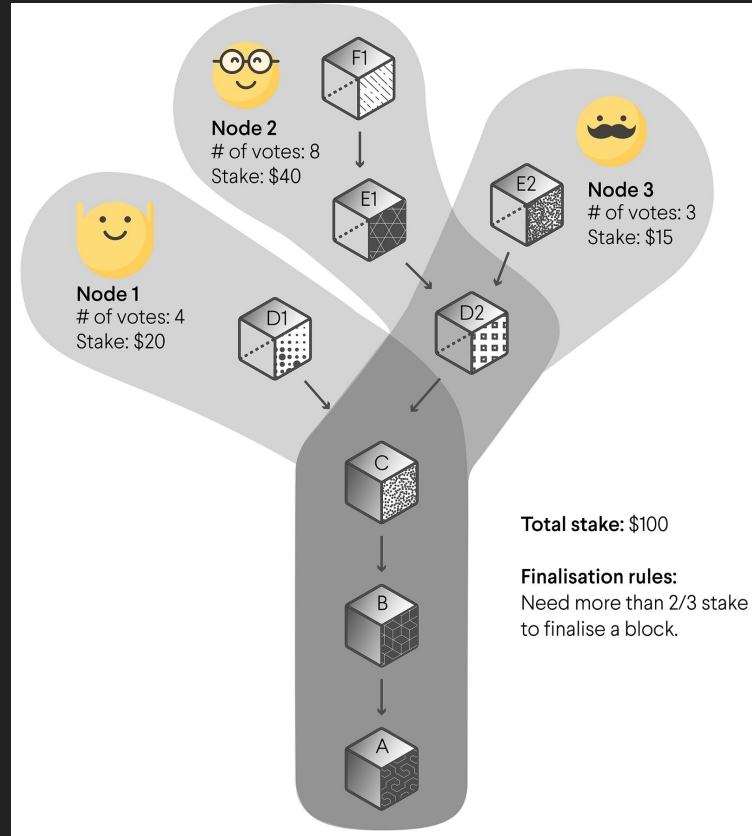


GRANDPA

GRANDPA is a round-based two-phase-commit finality gadget where voters vote on chains and not blocks.

A local view of a round may only be concluded when an estimate of the maximum finalized block starts to move backwards.

GRANDPA

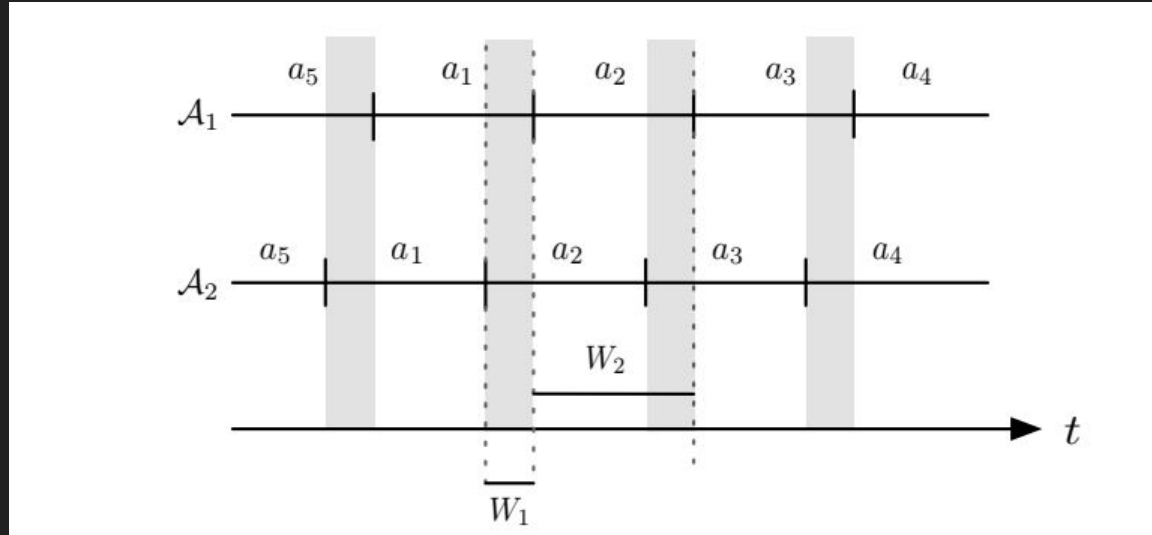


BABE: Blind Assignment for Blockchain Selection

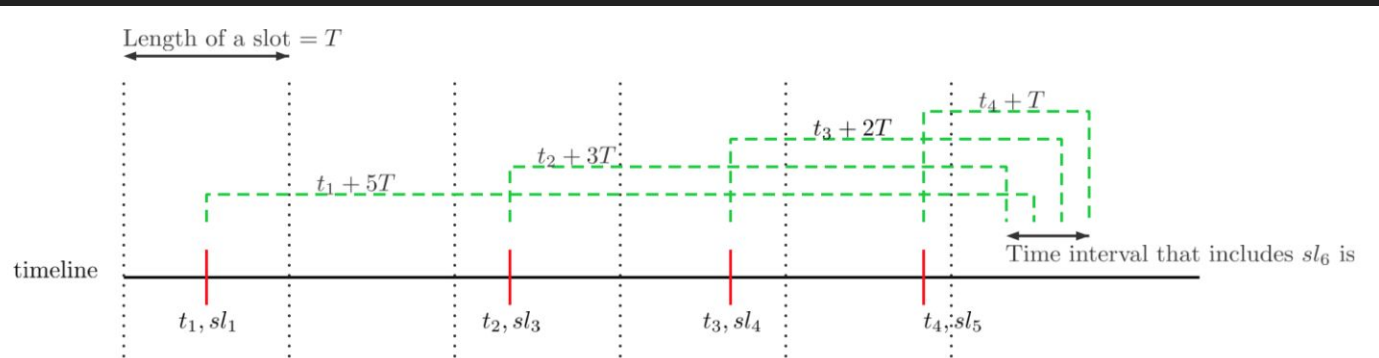
Inspired by Ouroboros Praos...

...but addresses practical issues

Clock drift is a huge practical problem for
slot-based block production.



De Angelis et al. Applying the CAP Theorem to Permission Blockchain



The figure shows how a new party P_j deduces an interval including sl_6 given the timeline of P_j , arrival time t_u of blocks for slots sl_v

Building Blocks

ImportQueue

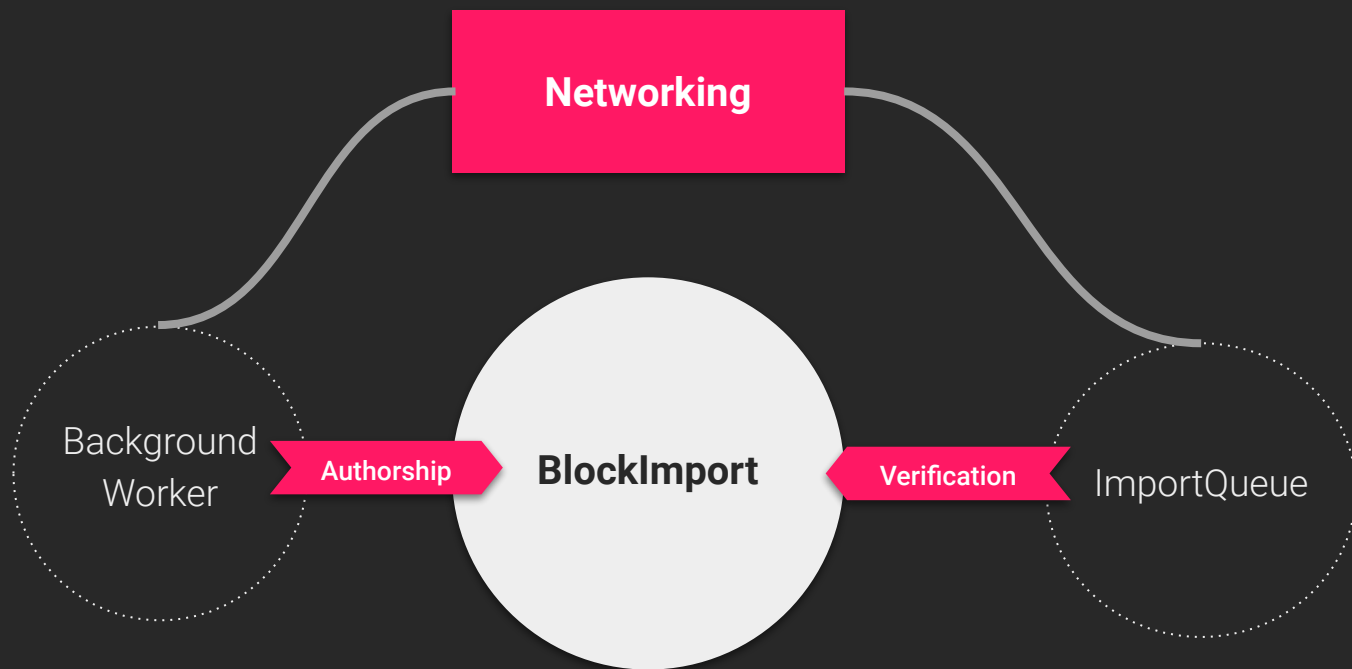
Does potentially-parallel first-stage, unordered-style verification of the blocks coming from the network.

BlockImport

Synchronous import of all blocks,
from network or locally Authored.
The basic BlockImport is a Client,
and others will wrap it.

Background Worker

An asynchronous background task or set of tasks that performs work such as authoring blocks or casting votes.



Cumulus



Cumulus is a set of libraries providing a “Polkadot Consensus” engine for Substrate and modules for turning a Substrate runtime into a Polkadot validation function.



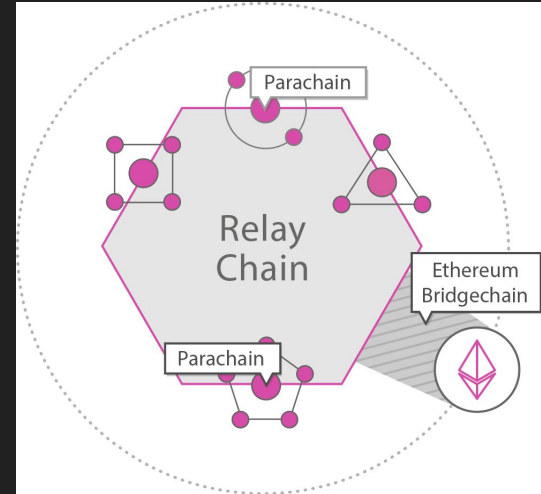
Cumulus can be used to turn any already-written Substrate codebase into a parachain variant with minimal code changes.

A node using Cumulus embeds a Polkadot Light Client as well.



A 10,000 Foot View

- **Relay Chain.** Coordinates consensus and transaction delivery between chains
- **Parachains.** Constituent blockchains which gather and process transactions
- **Bridges.** Link to blockchains with their own consensus




The background of the slide is a dark, monochromatic version of Leonardo da Vinci's Vitruvian Man drawing. The figure is centered, with arms and legs extended to touch the boundaries of a square and a circle. The text is overlaid on the lower half of the figure.

Anatomy of a Parachain

The Validation Function

This is a piece of code registered on the Polkadot relay chain. This is what Polkadot validators run to see if collations are valid.

Validation functions are deterministic and self-contained.

```
1   fn validate(  
2      last_header: &[u8],  
3      polkadot_info: PolkadotStatus,  
4      block_data: &[u8],  
5  ) -> Result<Vec<u8>>  
6
```

```
1  [-] fn validate(  
2      last_header: &[u8],  
3      polkadot_info: PolkadotStatus,  
4      block_data: &[u8],  
5  ) -> Result<Vec<u8>>  
6
```

??

Collator Nodes

These are nodes who have the job to compute block data for a parachain such that the validation function for that parachain returns successfully.

What that block data is depends on the parachain, so this is something that needs to be specialized.

Collators need to follow the relay chain and the parachain that they collate for.

Cumulus-Collator

Cumulus-collator is a special block-authorship pipeline for Substrate that creates blocks along with their State Trie Block Data.

The key observation is that Cumulus needs to include trie nodes necessary for executing transactions, but the messages should not be included in the block data, as that would be redundant with the argument of `validate(...)`

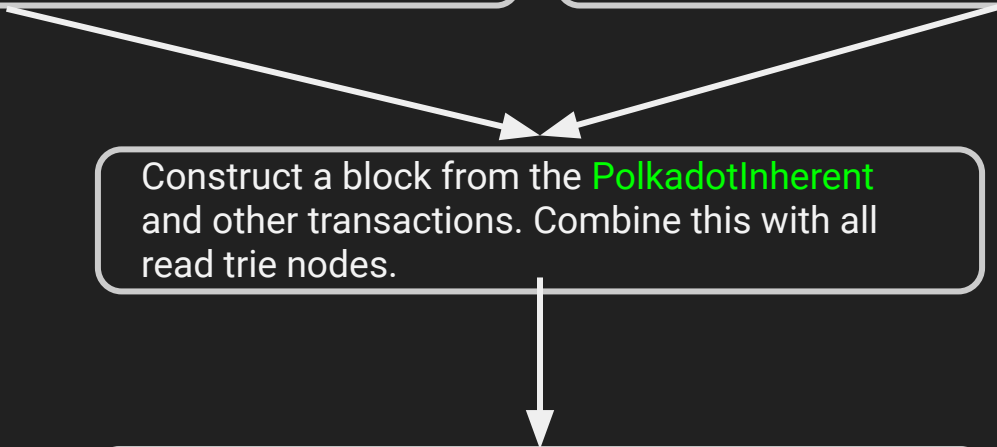
Cumulus Collation Pipeline

Compute incoming messages and other relay chain data. Form a “**PolkadotInherent**” from them.

Select a set of other transactions and invariants to include in the block.

Construct a block from the **PolkadotInherent** and other transactions. Combine this with all read trie nodes.

Combine the transactions, state trie proof, and messages into the **PoVBlock**.



How are Parachains different than sovereign-chain Substrate?

Part 2: Chain Synchronization

Following The Relay Chain

The Polkadot Relay Chain is finalized with GRANDPA consensus. Parachains are finalized via the same process.

That means that we can always find the latest finalized header for a given parachain.

For light clients, our work is done once we've found that.

However, full nodes need to download all the blocks. Downloading in reverse means we can't import blocks as we download.

Downloading forward poses its own challenges.

Following The Relay Chain

Finding the most recently finalized parachain
header is easy...

...but finding the canonical hash of an older header
isn't.

Solution: Log2 Ancestors

A parachain will contain in its state a table with N entries. The i 'th entry will contain the block hash and number of the last parachain block whose number was divisible by 2^i .

With a special-case rule for state-pruning, we can easily find canonical block hashes once every few thousand blocks easily.

Solution: Log2 Ancestors

	Number	Hash
1	1,000,123	0x16ab...
2	1,000,120	0x963e
4	1,000,112	0x5cc3
16	983,040	0x13bd
18	786,432	0x1712

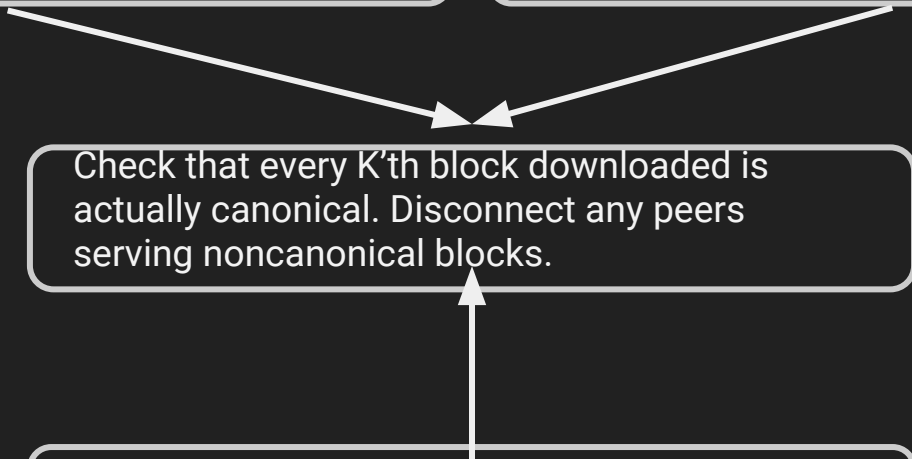
Parachain Synchronization Strategy

Follow the latest finalized parachain header by tracking the relay chain's finalization.

Use \log_2 ancestors to find canonical block hashes K (≈ 1000) blocks apart.

Check that every K 'th block downloaded is actually canonical. Disconnect any peers serving noncanonical blocks.

Download blocks sequentially, attempting to follow the best chain.



Tying It All Together

What does Cumulus Give You?

- A **consensus** and chain **synchronization** implementation for Substrate that follows a Relay chain.
- A **Collator node** for producing parachain blocks.
- Tools for writing Substrate runtimes that **send and receive messages** from other parachains.

How to get started with Cumulus?

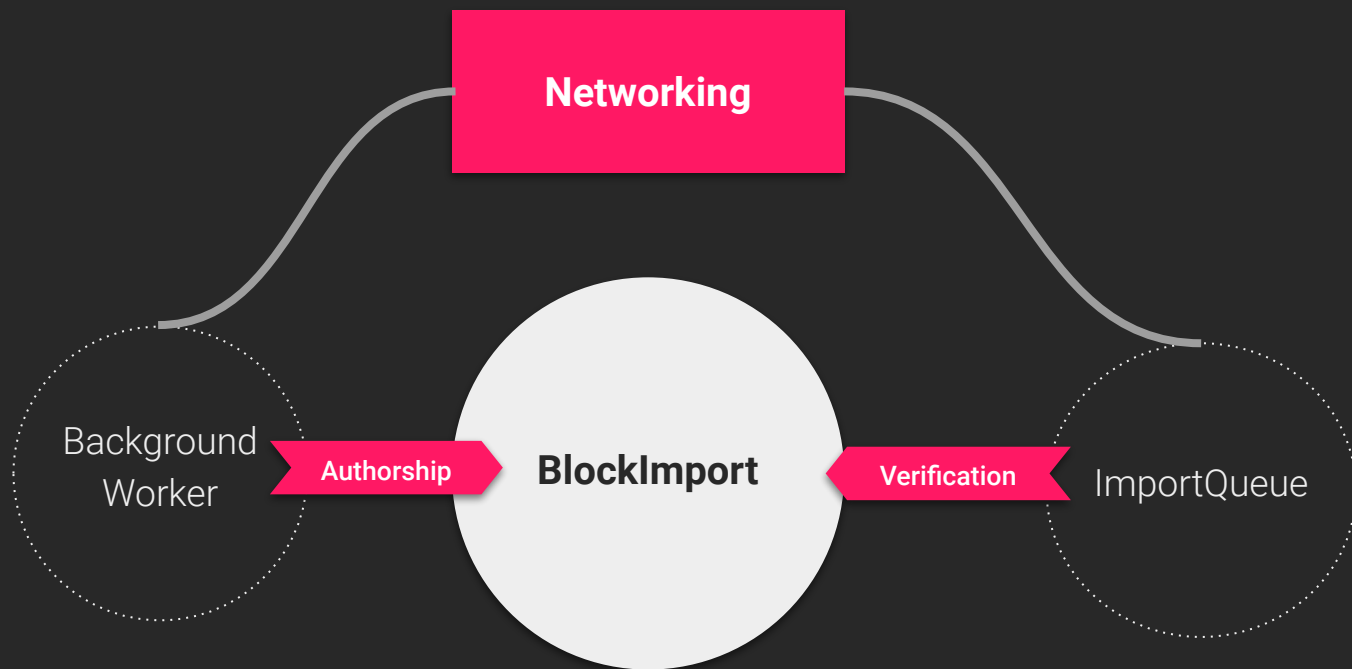
Cumulus itself is in the very early stages of development. But it is designed to require minimal codebase changes to an implementation of a sovereign chain in Substrate.

This means that you can write your chain in Substrate as a sovereign chain and tweak the codebase slightly to transform it into a parachain later on.

What if you want to start a chain as a sovereign
chain...

...and later turn it into a parachain?

Swappable Consensus



Kinds of Consensus Engines

- Finality Providers
- Blockchain Extenders

```
/// Consensus engine unique ID.  
pub type ConsensusEngineId = [u8; 4];
```

Consensus Overseer

The Consensus Overseer is a Consensus engine that allows seamless switching between a number of registered consensus engines. Each of these has its own ID.

Some, like BABE, GRANDPA, and Parachain, will come pre-registered by Substrate. You will be able to add your own as well.

Switching Engines

Only one finality provider or blockchain extender may be registered at any time.
Some engines may be both at once.

Switching Engines

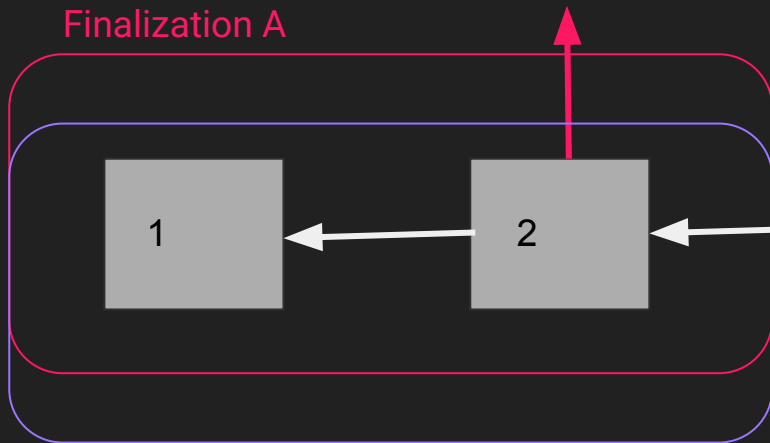
Changing engines is done by issuing a digest in the header.

If the engine ID is the OVERSEER_ID, the message may contain a signal to transfer engine.

When do engine switches happen?

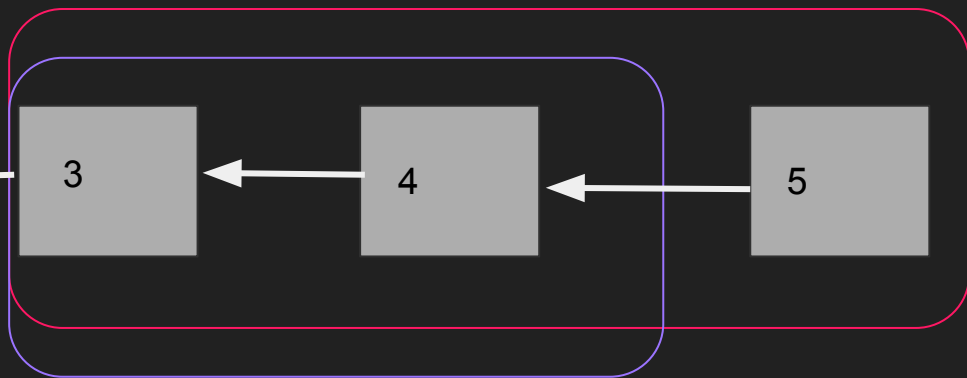
```
Consensus(ConsensusEngineId, Vec<u8>),
```

Finalization A



Chain Extension A

Finalization B starts when 2 is **Finalized**

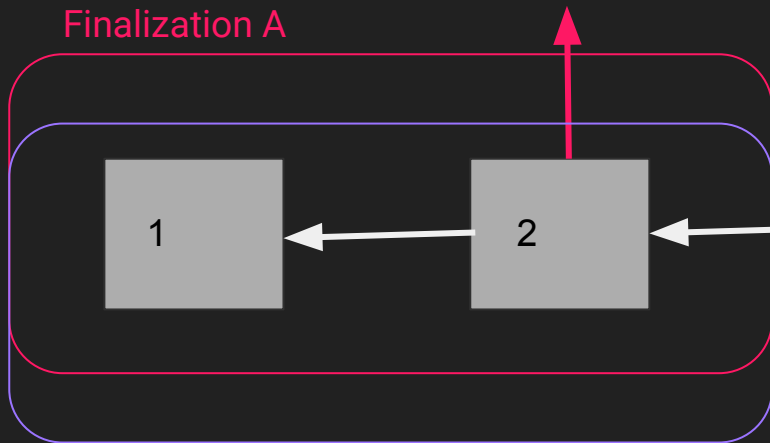


Chain Extension B starts when 2 is **Authored**

Fork Choice when changing Extension

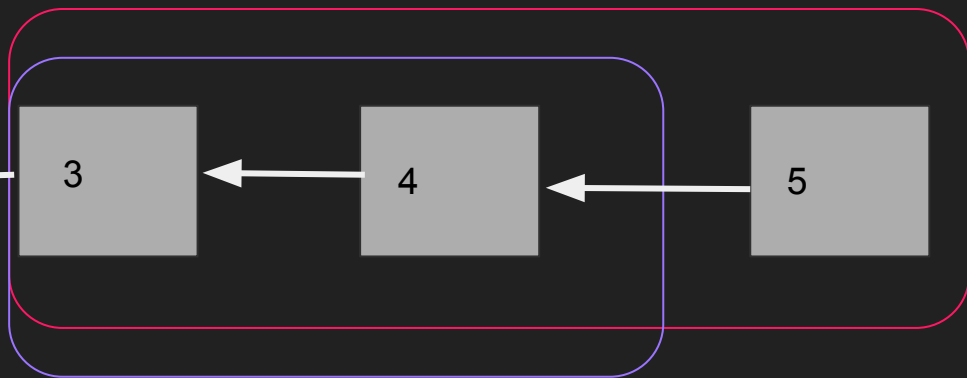
```
Consensus(ConsensusEngineId, Vec<u8>),
```

Finalization A



Chain Extension A

Finalization B starts when 2 is **Finalized**



Chain Extension B starts when 2 is **Authored**

Thank you

Robert | robert@polkadot.io | [@rphmeier](https://twitter.com/rphmeier)