Eth2

Beyond the Beacon Chain

Dr. Robert Drost
Consensys R&D
@rjdrost
robert.drost@consensys.net

Eth 1.0

- Hard coded transaction/execution framework. To participate in Consensus (PoW mining) you have to maintain full state and execute all transactions yourself.
- Any changes to Eth1’s consensus rules or state transition function require a ...

Eth 2.0:

- Composes blockchain from Fixed and EE layers:

  Fixed layer (forks to change) defines and runs

  The Beacon and Shards POS protocol to validate+order blocks

  Eth protocol to execute Eth transfers (EE↔BC, EE1→EE2)

  New contract construct defining Execution Environments (EEs)

  EE layer (no forks to change) defines for each EE:

  What is a transaction (e.g. Eth1, UTXO, ZEXE etc. models)

  What is the state (if any)

  How to execute+validate a block-to-block state transition

Eth 2.0 development

Phase 0: Beacon Chain

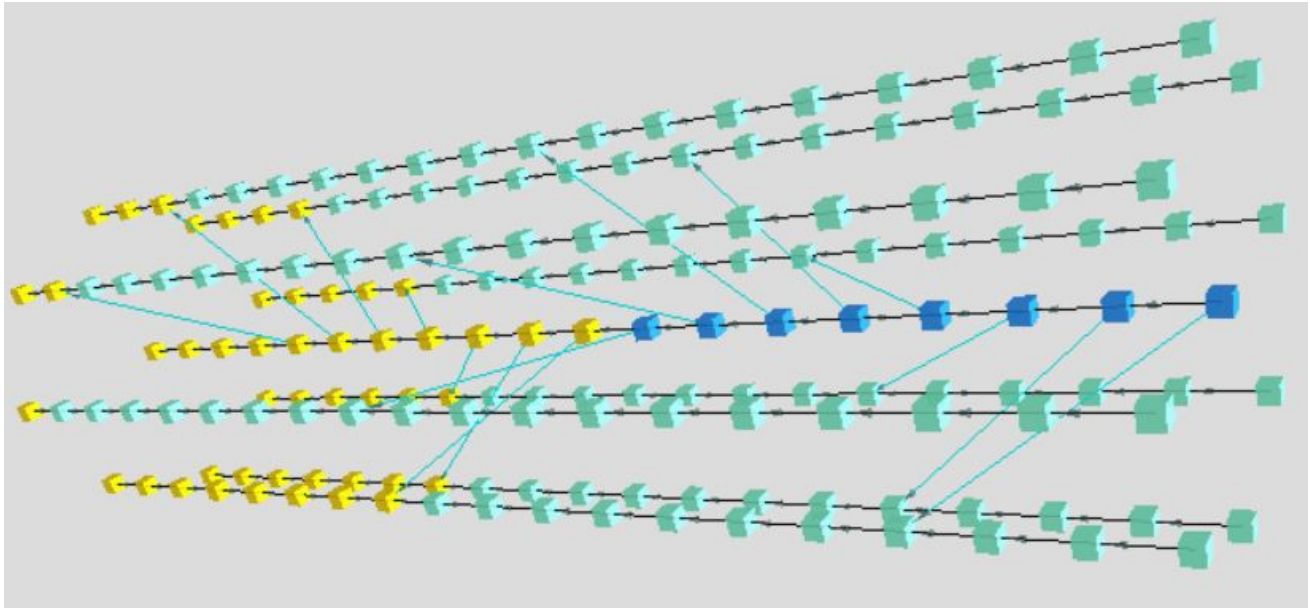
Phase 1: Shards

Phase 2: Execution

Phase 0 - Beacon Chain

- 1 Ring to rule them all and in the darkness bind them
- Randomness
- Checkins
- Finality = Trust

Phase 1 - Shards



Phase 2 - State Execution

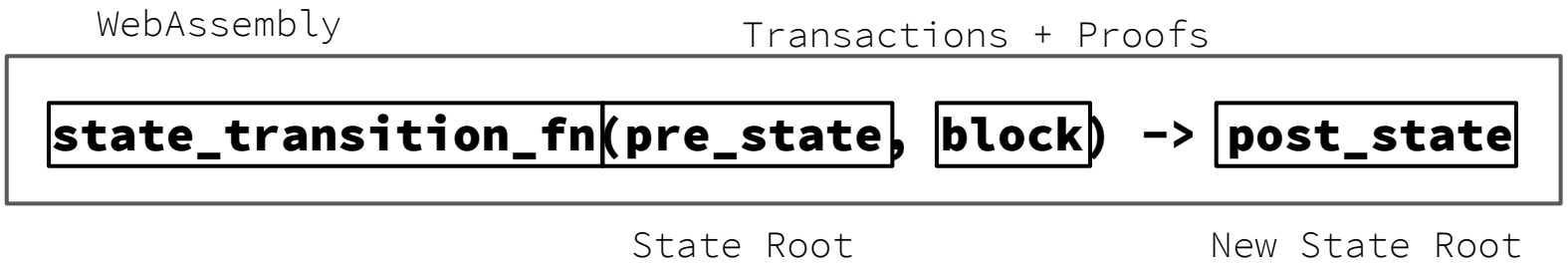
- Utility

Execution Environments in Eth 2.0

What’s an Execution Environment?

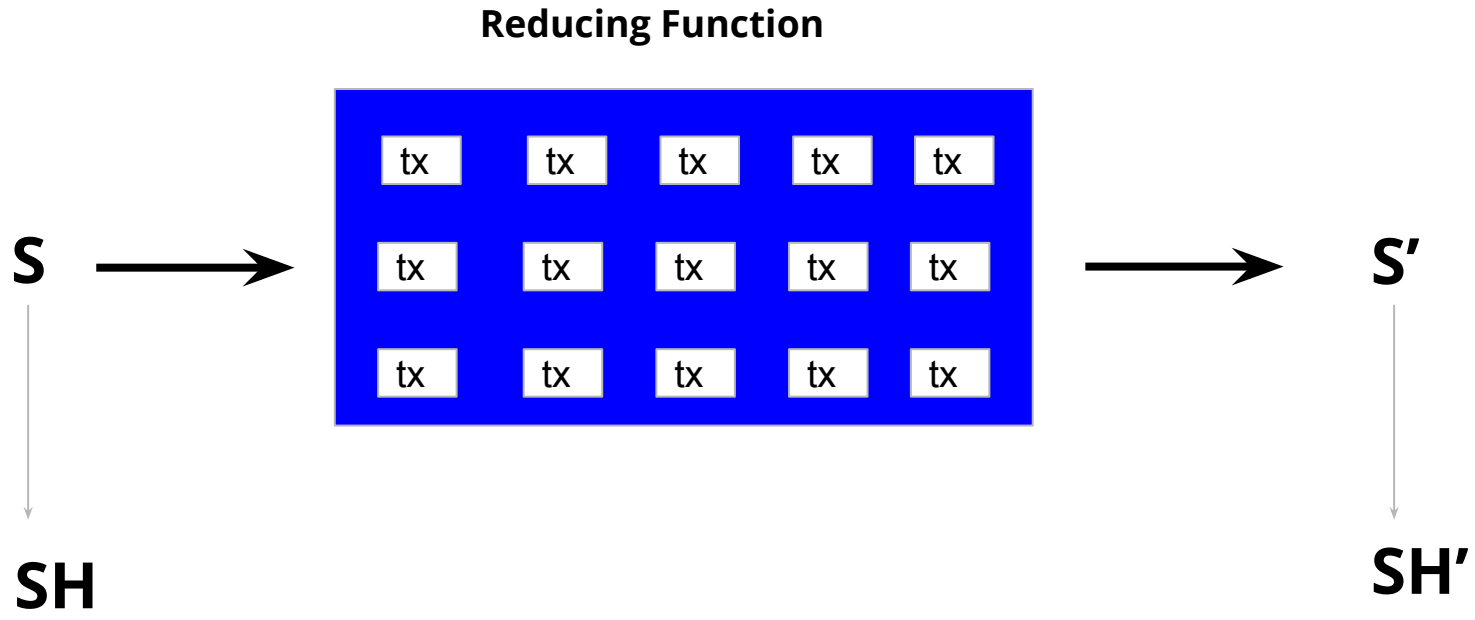
- Accepts Consensus rules of Beacon chain and shards as they are--changing those rules requires a fork
- But defines a state transition function (in WASM)
  - Pure function that accepts a pre-state root and input block data (transactions and data witnesses), and returns the a post-state root after executing the block (e.g. stateless)
- Expensive to deploy--not surprising as the state function can capture the execution protocol of a full blockchain: Hyperledger Fabric, Bitcoin, Eth1, Sovrin(Hyperledger Indy), and most others

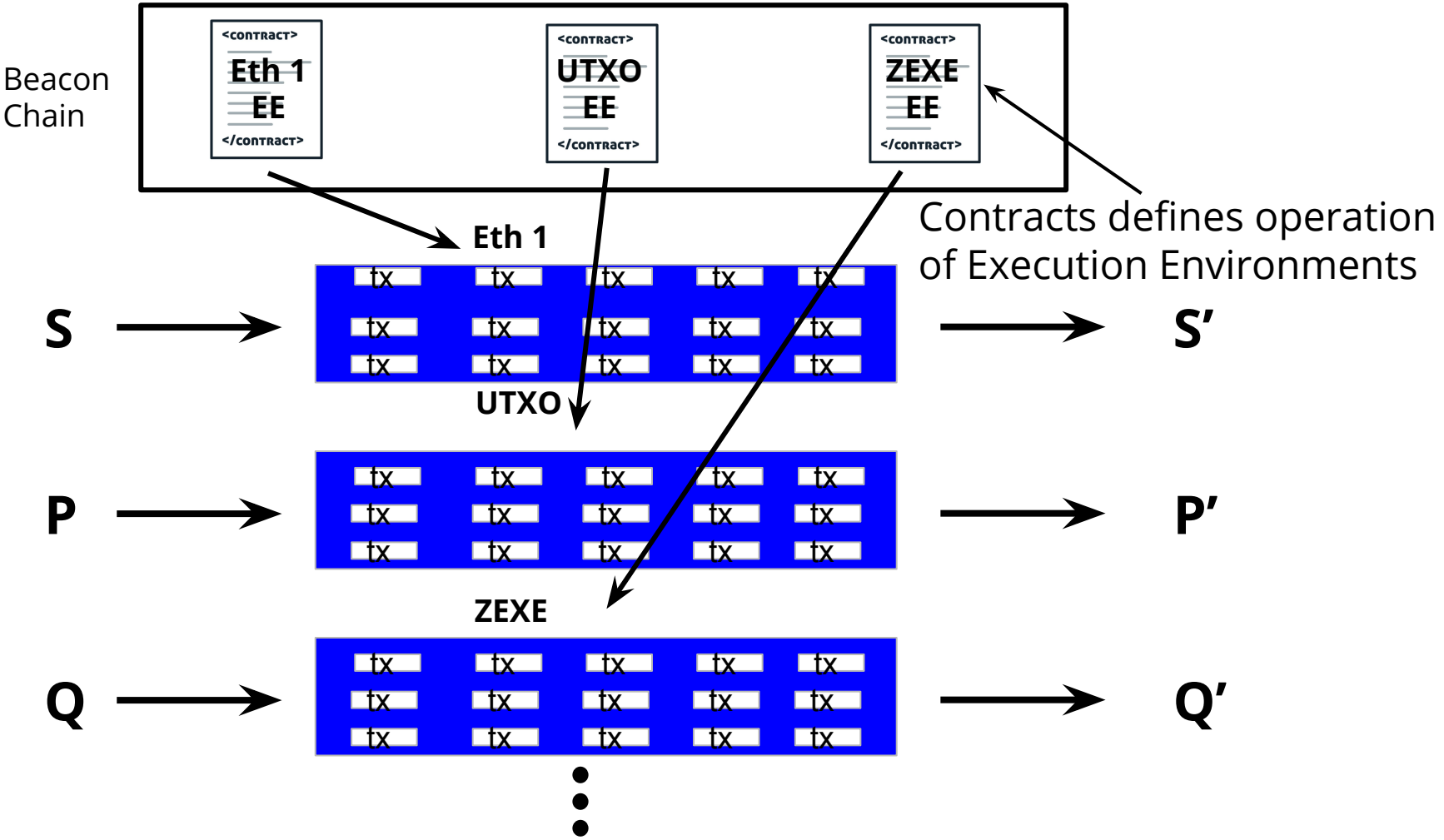
Pure Function



merkle partials for proofs: https://hackmd.io/@matt/SJE_v6Qer

State Update Hardcoded in Eth 1




Beacon Chain
<contract>
Eth 1 EE
</contract>
<contract>
UTXO EE
</contract>
<contract>
ZEXE EE
</contract>
Contracts defines operation of Execution Environments
Eth 1
UTXO
ZEXE
S
P
Q
S'
P'
Q'
tx

EE defined via Beacon chain contracts, but

EE != Eth1 Smart Contract

rather

EE == Executable* definition of a Blockchain: What are transactions, state, transition function

** But accepting the fixed layer operations of Eth2.0*

Why are Execution Environments valuable?

- In a sense,

  With EEs, Eth2 abstracts the fixed execution semantics of almost any conceivable programmable blockchain …

  in a similar way to how Eth1 abstracted the fixed semantics of digital tokens / currency in the early Blockchain/Bitcoin era

Why are Execution Environments valuable?

- The Beacon Chain plus Sharding alone “just” yields ten of thousands (to start) of secure and decentralized TPS
- EEs enable those TPS to be shared among programmed execution layers of any multitude of blockchains

Do I need to build an EE?

Questions you should answer:

- Are you a core developer / wish to contribute to the public infrastructure of Ethereum?
- Do you have an idea which only makes sense as an EE?
- Will my application benefit from a two-level contract hierarchy (with few modification over time of the EE level!)
  - Beacon chain EE contracts to define state transitions +
  - Transactions (that could include Eth-style Contracts+State) that run within that definition

What if I only build dapps

- Your world shouldn’t change too much if you run in an Eth1 or Eth2 EE (Eth2 EE adds cross-shard state xfers to an Eth1 account model)--other than that...
- Dapps will be faster and execution cost less due to POS and sharding (yeah!)

- But you may be left behind if you don’t learn to leverage the new infrastructure to your advantage

Example EEs

- Eth 1.x EE
  - EVM interpreter written in WASM
  - Defines how to interact with other EEs
  - Legacy smart contracts will continue happily humming along
- Eth 2.0 EE
  - Pure wasm--no need for interpreter as Eth2 clients have one already
  - Focus on supporting new cross-EE and cross-shard support
  - Implement things that are too difficult to get into 1.x
- Generic Asset EE
  - Like an ERC-20, but supports ~any interface + isn’t redeployed
  - Facilitates cross-EE transactions
- UTXO EE
  - Implements the Bitcoin protocol
- MOVE EE
  - From Facebook Libra

More Example EEs

- Delayed State Execution
  - The EE orders transactions and accepts state root commitments
  - Can allow for instant / synchronous cross-shard communication
- L2 Checkin
  - By checking in data onchain, it ensures data availability
  - Similar to delayed state execution
- Rollups
  - Zero-knowledge schemes which only commit a root
- Verified Claims EE
  - Implements a Sovrin-like protocol: Global Self-sovereign Identity base layer
- You tell us!
  - Eth 1.0 also benefited from fun experiments and pushing the protocol’s potential to its limits

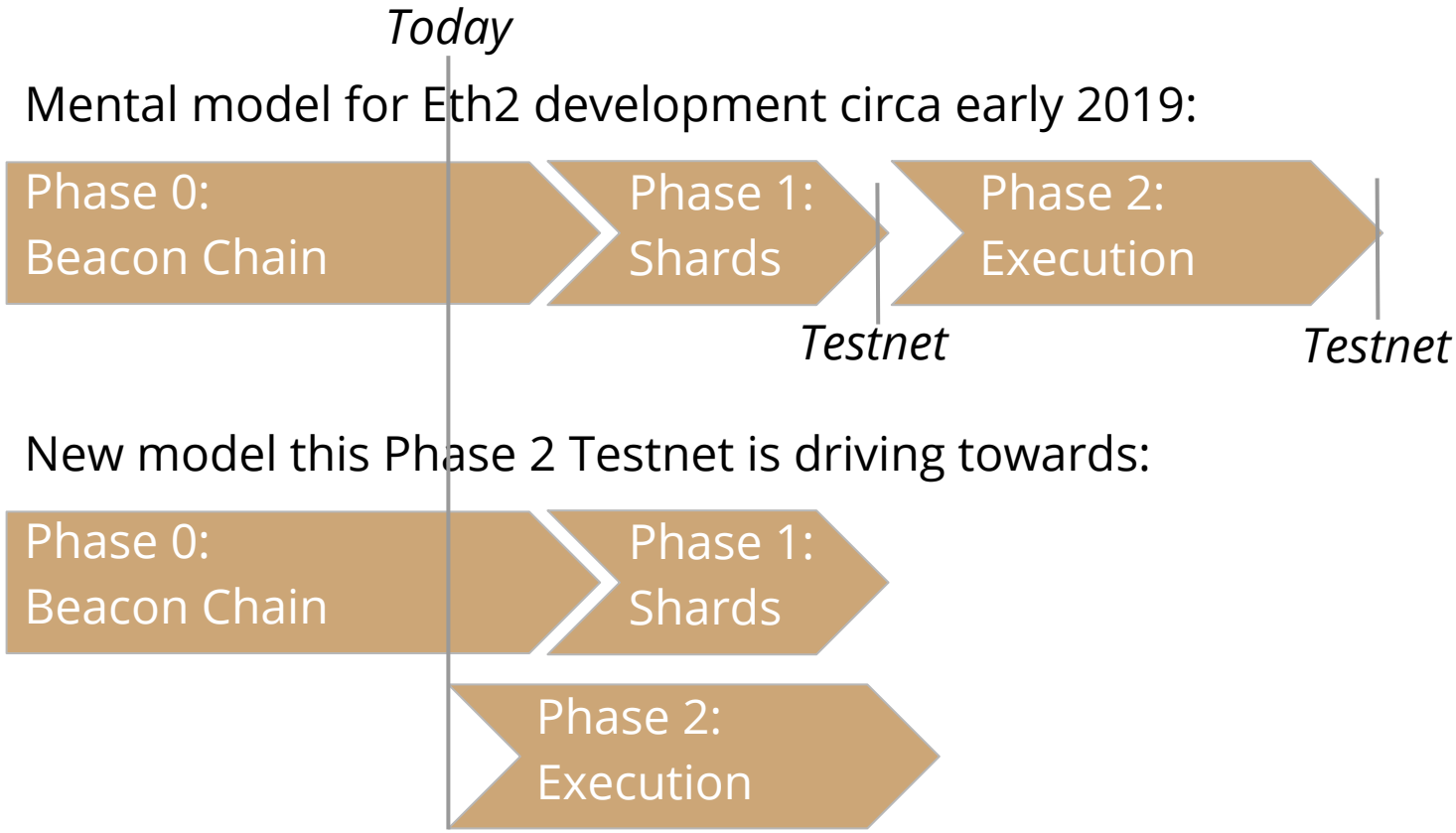
Testnet - Phase 2 (simulated Beacon chain, 8-16 Shards)

- Collaboration of Consensys R&D Quilt team efforts with
  - Scout (EF)
    - Eth2.0 execution engine
    - https://github.com/ewasm/scout
  - Lighthouse (SigmaPrime)
    - Eth2.0 Client in Phase 0/1 development currently
    - Written in Rust
- Supports Dapp eWasm contract deployments
- Begins simulating contracts/execution environments in action
- Begins simulating cross shard behaviors
- Provide target platform for research in Relayers and Fee Markets

Learn+contribute... https://hackmd.io/UzysWse1Th240HELswKqVA?view

Quilt/Phase2 testnet telegram channel... https://t.me/eth2quilt

Why this Phase 2 Testnet is exciting!



Acknowledgements

- Quilt related team
  - At Consensys
    - Will Villanueva*, Matt Garnett*, John Adler, Joseph Chow
- Community effort for the Phase2 Testnet
  - EF: Axic, Casey Detrio
  - Sigma Prime: Paul Hauner and the Lighthouse dev team

* This deck is mashup of presentations by Will and Matt (+ a bit of my spin :)

Thanks!

robert.drost@consensys.net

@rjdrost

Learn+contribute https://hackmd.io/UzysWse1Th240HELswKqVA?view

Quilt/Phase2 testnet telegram channel https://t.me/eth2quilt

EE Developer Toolbox

- WebAssembly
- Host functions
- Dynamic host functions (DHFs)

WebAssembly (wasm)

“WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.” — WebAssembly docs

Host functions

- Written in the language of the client, linked into the wasm runtime
- semi-finalized
  - eth2_loadPreStateRoot(offset: *const u32)
  - eth2_blockDataSize() -> u32
  - eth2_blockDataCopy(outOffset: *const u32, offset: u32, length: u32)
  - eth2_savePostStateRoot(offset: *const u32)
  - eth2_pushNewDeposit(offset: *const u32, length: u32)
- to-be-determined
  - eth2_execCode(code_ptr: *const u32, code_length: u32, calldata_ptr: *const u32, calldata_length: u32)
  - eth2_execEE(ee: u64, calldata_ptr: *const u32, calldata_length: u32)
  - ???

Dynamic Host functions

- Defined per EE
- Written in wasm, linked into the wasm runtime
- Allows for control of privileges during execution
- Not formalized yet