**Mothership:** https://github.com/ethereum/wiki/wiki

This is really gory: https://ethereum.github.io/yellowpaper/paper.pdf section 10 describes how canonical blockchain state is maintained. Genesis block described in annex.
Ethereum, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some final state. It is this final state which we accept as the canonical “version” of the world of Ethereum. The state can include such information as account balances, reputations, trust arrangements, data pertaining to information of the physical world; in short, anything that can currently be represented by a computer is admissible. Transactions thus represent a valid arc between two states; the ‘valid’ part is important—there exist far more invalid state changes than valid state changes. Invalid state changes might, e.g., be things such as reducing an account balance without an equal and opposite increase elsewhere
Transactions are collated into blocks; blocks are chained together using a cryptographic hash as a means of reference. Blocks function as a journal, recording a series of transactions together with the previous block and an identifier for the final state (though do not store the final state itself—that would be far too big). They also punctuate the transaction series with incentives for nodes to mine. This incentivisation takes place as a state-transition function, adding value to a nominated account. Mining is the process of dedicating effort (working) to bolster one series of transactions (a block) over any other potential competitor block. It is achieved thanks to a cryptographically secure proof. This scheme is known as a **proof-of-work**.

**P2P protocol is DEVP2P:**

https://github.com/ethereum/wiki/wiki/Ethereum-Wire-Protocol

https://github.com/ethereum/wiki/wiki/%C3%90%CE%9EVp2p-Wire-Protocol

Recursive Length Prefix is used to serialize objects
https://github.com/ethereum/wiki/wiki/RLP

RLPx is a Cryptographic Network & Transport Protocol
https://github.com/ethereum/devp2p/blob/master/rlpx.md

RPLx itself is adopts Kademlia for Node discovery (w/ some changes): https://en.wikipedia.org/wiki/Kademlia#Joining_the_network - follows the same bootstrap process.

Geth - Go implementation of Ethereum client-
https://github.com/ethereum/go-ethereum/wiki/Connecting-to-the-network
https://github.com/ethereum/go-ethereum/blob/master/p2p/server.go

---
**Data structures of Ethereum**

Storage:

https://blog.ethereum.org/2015/06/26/state-tree-pruning/

https://github.com/ethereum/wiki/wiki/Patricia-Tree
---

WOW diagrams: https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture

==========

Setting up ETH genesis blockchain:

https://media.consensys.net/how-to-build-a-private-ethereum-blockchain-fbf3904f337
https://medium.com/mercuryprotocol/how-to-create-your-own-private-ethereum-blockchain-dad6af82fc9f

==========
Proof of work: ethash: https://www.vijaypradeep.com/blog/2017-04-28-ethereums-memory-hardness-explained/
Fundamental concept is same:
Guess a nounce - hash some stuff and it has to be less than pre-specified target.

DAG: pseudorandom dataset initialized by current blockchain length. Regenerated every 30K blocks (~5days).

1. The **Preprocessed Header** (derived from the latest block) and the **Current Nonce** (the current guess), are combined using a SHA3-like algorithm to create our initial 128 byte mix, called **Mix 0** here.
2. The **Mix** is used to compute which 128 byte page from the DAG to retrieve, represented by the **Get DAG Page** block.
3. The **Mix** is combined with the retrieved DAG page. This is done using a ethereum-specific mixing function to generate the next mix, called **Mix 1** here.
4. Steps 2 & 3 are repeated 64 times, finally yielding **Mix 64**.
5. **Mix 64** is post processed, yielding a shorter, 32 byte **Mix Digest**.
6. **Mix Digest** is compared against the predefined 32 byte **Target Threshold**. If **Mix Digest** is less than or equal to **Target Threshold**, then the **Current Nonce** is considered successful, and will be broadcast to the ethereum network. Otherwise, **Current Nonce** is considered invalid, and the algorithm is rerun with a different nonce (either by incrementing the current nonce, or picking a new one at random).

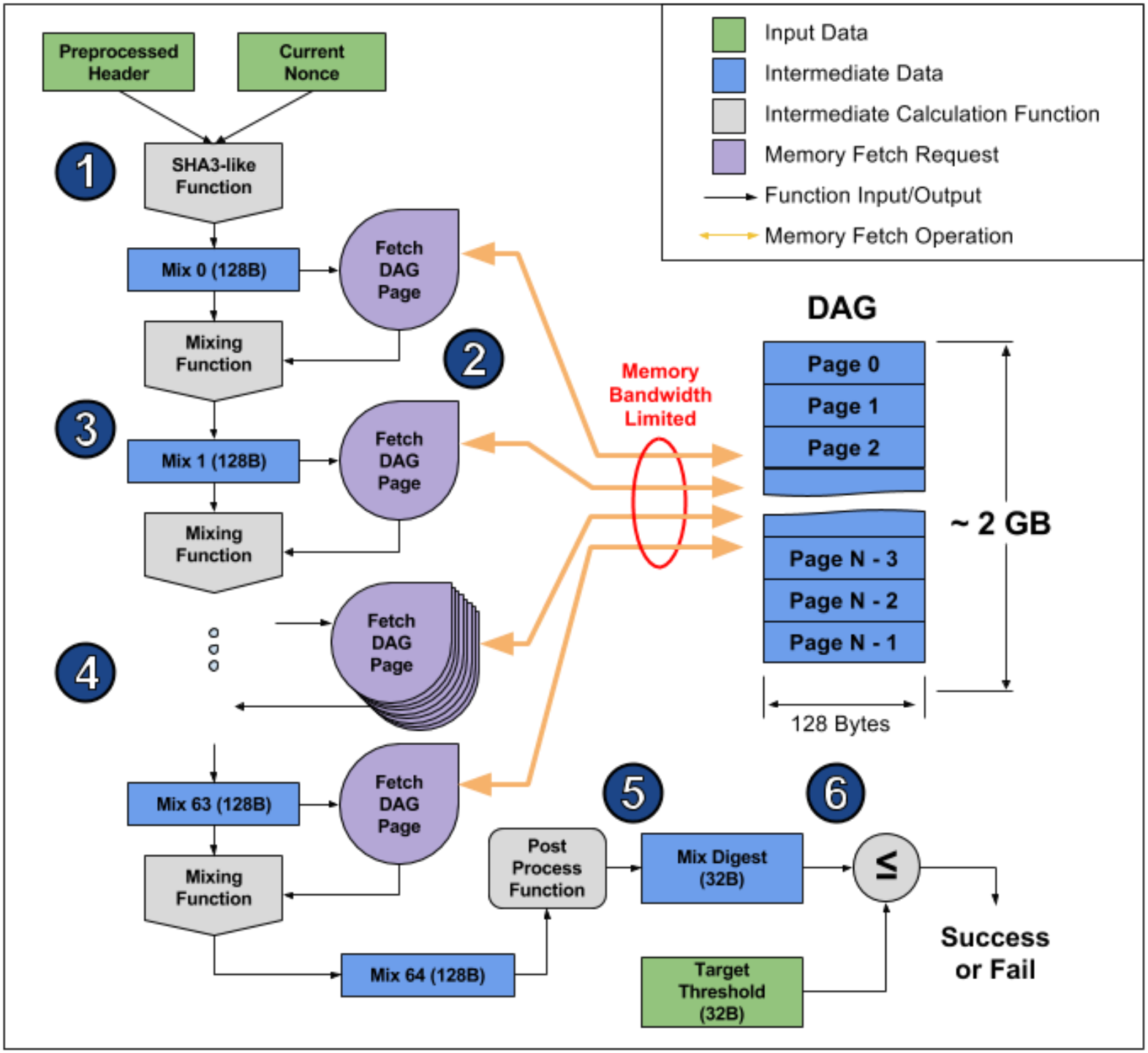
Ethash Hashing Algorithm
Input Data
Intermediate Data
Intermediate Calculation Function
Memory Fetch Request
Function Input/Output
Memory Fetch Operation
Preprocessed Header
Current Nonce
1
SHA3-like Function
Mix 0 (128B)
Fetch DAG Page
Mixing Function
2
3
Mix 1 (128B)
Fetch DAG Page
Mixing Function
4
Fetch DAG Page
Mix 63 (128B)
Fetch DAG Page
Mixing Function
Mix 64 (128B)
Memory Bandwidth Limited
DAG
Page 0
Page 1
Page 2
Page N - 3
Page N - 2
Page N - 1
~ 2 GB
128 Bytes
5
Post Process Function
Mix Digest (32B)
6
≤
Target Threshold (32B)
Success or Fail

**Figure 1 - Algorithmic flow for how Ethereum's ethash hashing algorithm works (Vijay Pradeep - Source)**

==========

Gas business: https://blog.softwaremill.com/ethereum-everything-you-want-to-know-about-the-gas-b7c8f5c17e7c

https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas

https://bitfalls.com/2017/12/05/ethereum-gas-and-transaction-fees-explained/

TOKENS (tokens): https://blockgeeks.com/guides/ethereum-token/