Ethereum: Concepts, Theory, Approach

Define-first 90-minute teaching lecture

Prof. Dr. Jörg Osterrieder

BSc Blockchain, Crypto Economy & NFTs

Spring 2026

Today's question. What is Ethereum, mechanically? Six sections, define-first. Every term gets a one-sentence definition before it appears in any application. The companion Colab notebook lets you compute every concept yourself.

Roadmap (six sections, 90 minutes)

Sections

1. Crypto primitives (hash, public-key, signature)
2. State machine (state, account, MPT, transaction)
3. EVM and gas (VM, opcode, EIP-1559)
4. Consensus and finality (validator, slot, epoch, finality)
5. Smart contracts (contract, ABI, event, ERC-20, ERC-721)
6. Appendix: Layer 2 (rollup, optimistic, zk)

Pedagogy

- **Define-first.** If a word is on a slide, it was defined on an earlier slide. Raise your hand if not.
- **Apply-after.** Each Define is followed by an Apply slide showing how Ethereum uses the concept.
- **Compute-yourself.** The Colab notebook reproduces every Apply slide in Python.

Five advanced sidebars (ADV1 to ADV5) wait at the end for the questions you will have.

In this section we will define:

hash function → **public-key cryptography** → **digital signature**

Each Define slide is followed by an Apply slide showing how Ethereum uses the concept.

Goal: by the end of this section, you can explain what `0x1a4...c92d` actually is.

Define: Hash function

Hash function (a digital fingerprint). A deterministic function $H : \{0,1\}^* \to \{0,1\}^n$ that maps any input to a fixed-length output of n bits. Three required properties:

- **Deterministic.** Same input always gives the same output.
- **Pre-image resistant.** Given $H(x)$, finding any x that hashes to it is computationally infeasible.
- **Collision resistant.** Finding two different inputs with the same output is computationally infeasible.

Why it matters in Ethereum. keccak256 (a 256-bit hash) is the workhorse: addresses, transaction IDs, block IDs, and all Merkle commitments are keccak256 of something. Every identifier on the chain is either an address or a hash.

Idiom gloss. "Computationally infeasible" = no known algorithm finishes the search before the heat death of the universe. keccak256 is the original SHA-3 candidate, slightly different from the standardised SHA3-256.

Apply: Hash builds block linkage

Each block stores the keccak256 of its parent. Tampering with block N changes its hash, which breaks the link from N+1, which breaks N+2, and so on. The chain is *tamper-evident*, not tamper-proof; consensus (Section 4) decides which chain is canonical.

Concrete. Add a single comma to a transaction in block N and the stateRoot changes; block N+1's parentHash now points at a fingerprint nobody else has, so honest nodes drop the tampered chain.

Vocab. Tamper-evident = changes are detectable. Canonical chain = the one all honest nodes agree on.

Define: Public-key cryptography

Public-key cryptography (a postbox plus its key). A scheme where each user has TWO keys.

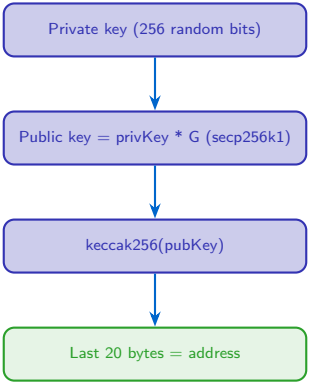
- **Private key** (256-bit secret number). Never shared.
- **Public key** (derived from the private key via a one-way function). Shared freely.

Mathematical guarantee: deriving public from private is easy, the reverse is computationally infeasible. Ethereum uses the secp256k1 elliptic curve, the same curve as Bitcoin.

Why it matters in Ethereum. The keypair lets you prove identity (you signed something only the private-key holder could sign) without revealing the secret. The Ethereum address is the last 20 bytes of keccak256 of your public key (next slide).

Vocab. secp256k1 = a specific elliptic curve standardised by SECG. The curve choice affects security, not the protocol shape.

Apply: How an Ethereum address is born



Concrete. An address like `0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045` is the hex of the last 20 bytes of keccak256 of someone's public key. Cell 4 of the Colab notebook computes this end-to-end.

Check. 40 hex characters = 160 bits = 20 bytes. So an address is `0x` + 40 hex chars after the prefix.

Define: Digital signature

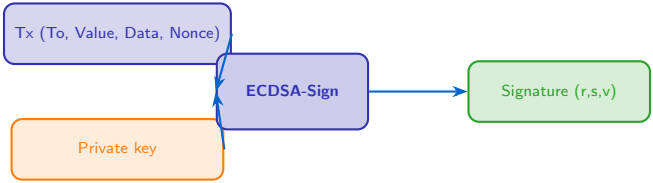
Digital signature (a wax seal that fits only THIS letter). A value $\sigma = \mathrm{Sign}(\mathrm{privKey}, m)$ such that anyone with the matching public key can run $\mathrm{Verify}(\mathrm{pubKey}, m, \sigma)$ and confirm the signature was produced by the holder of the private key.

- Only the private-key holder can produce a valid σ.
- Anyone can verify with the public key.
- σ is bound to the exact message; flipping one bit invalidates it.

Why it matters in Ethereum. Every transaction is signed by the sender. Without a valid signature, the network rejects the transaction. The signature is what makes "send 1 ETH from Alice to Bob" not forgeable.

Vocab. ECDSA = Elliptic Curve Digital Signature Algorithm. Ethereum uses ECDSA over secp256k1.

Apply: Signing a transaction



The validator that includes your transaction recovers your public key from the signature, derives your address (S07), and checks that this address has the funds to pay. No signature means no inclusion. Cell 5 of the Colab notebook signs and verifies a message; flipping one bit in the message invalidates the signature.

Vocab. (r, s, v) are the three components of an Ethereum signature; v is the "recovery id" used to recover the public key from the signature alone.

We will define:

state machine → account model (EOA vs contract)

→ Merkle Patricia Trie → transaction

Each Define is followed by an Apply showing how Ethereum implements it.

Goal: by the end, you can answer "what changes between two consecutive Ethereum blocks?".

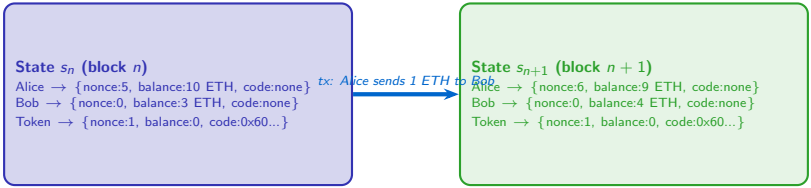
Define: State machine

State machine (a vending machine: drop coin, press button, receive can). A tuple $(\mathcal{S}, \delta, s_0)$ where $\mathcal{S}$ is a set of states, $\delta : \mathcal{S} \times I \to \mathcal{S}$ is a deterministic transition function from a (state, input) pair to a new state, and $s_0 \in \mathcal{S}$ is the starting state. Apply δ over a sequence of inputs and you trace the state's evolution.

Why it matters in Ethereum. Ethereum is a state machine where (1) the state is the world state of every account, (2) the transition function is "process one transaction", (3) the starting state is the genesis block. Every node, given the same starting state and the same sequence of transactions, computes the same ending state.

Vocab. World state = the global mapping from address to account record at one point in time.

Apply: Ethereum's world state, two consecutive blocks



A block is a batch of transactions that, applied together, transitions $s_n \to s_{n+1}$. Deterministic: every node arrives at the same s_{n+1}.

In practice $\mathcal{S}$ has hundreds of millions of accounts; the diagram shows three.

Define: EOA vs contract account

Externally Owned Account, EOA (your personal email account). An account whose authority is a private key. Has nonce, balance, no code, no storage. A human user with MetaMask owns an EOA.

Contract account (an automated email reply rule). An account whose authority is its own code. Has nonce, balance, code (immutable bytecode), and storage (a key-value map specific to this contract). A deployed smart contract is a contract account.

Why it matters. EOAs initiate transactions; contracts react to them. A transaction always starts with an EOA signature.

Vocab. ERC-4337 (account abstraction, ADV4) is starting to relax this distinction.

Apply: An account is four fields

Field	Type	Meaning
`nonce`	uint64	For EOAs: number of transactions sent. For contracts: number of contracts created from this contract. Prevents replay attacks.
`balance`	uint256 (wei)	ETH owned by this account, in wei (10^{18} wei = 1 ETH).
`codeHash`	bytes32	For EOAs: hash of empty bytes. For contracts: keccak256 of the deployed bytecode. Bytecode itself is stored separately.
`storageRoot`	bytes32	For EOAs: empty trie root. For contracts: root of a Merkle Patricia Trie holding this contract's storage slots.

Vocab. wei = the smallest unit of ETH. Replay attack = re-sending a previous transaction; the nonce blocks this.

Define: Merkle Patricia Trie (MPT)

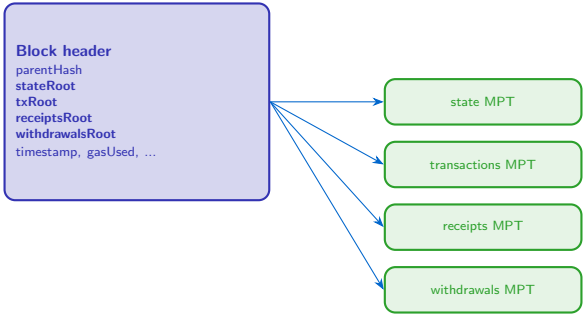
Merkle Patricia Trie (a library catalogue that gets a new fingerprint when any single book is replaced). A tree data structure that maps keys to values AND produces a single hash (the "root") committing to the entire mapping. Two properties:

- Lookup by key takes $O(\log n)$ time and produces a proof of inclusion (a small Merkle path).
- Any change to any value changes the root.

Why it matters in Ethereum. The world state, each contract's storage, and the transaction list are all MPTs. The block header carries the roots, not the data, so a header is small (a few hundred bytes) regardless of how big the state is.

Vocab. Commitment = a short value uniquely identifying a larger dataset. Patricia is a 1968 trie variant; Merkle is a 1979 hash tree.

Apply: One block header, four roots



To prove "Alice has 9 ETH at block N+1" you produce a Merkle proof from Alice's leaf up to the stateRoot in the header. The proof is hundreds of bytes; the state itself is gigabytes.

This is what makes light clients possible: download headers, verify proofs on demand. Verkle trees (ADV1) shrink proofs further.

Define: Transaction (post-EIP-1559)

Transaction (a signed bank cheque; the data field is the memo line). A signed tuple:

$$(\texttt{nonce}, \texttt{to}, \texttt{value}, \texttt{data}, \texttt{gasLimit}, \texttt{maxFeePerGas}, \texttt{maxPriorityFeePerGas}, \mathrm{signature}).$$

The signature is over a hash of the other fields. To create a contract, set `to` to the empty address; the contract's address is computed from sender + nonce.

Why it matters. Every state change on Ethereum starts with one of these. There is nothing else. The chain is the chronologically ordered log of valid transactions.

Vocab. calldata = the `data` field. For a token transfer, it encodes "transfer(to, amount)" using the contract's ABI (S29).

We will define:

virtual machine + opcode → gas → EIP-1559 fee market

Concept density rises here. Definitions are tight; applications carry the worked numbers.

Goal: you can compute the dollar cost of a contract call from gas units, base fee, and priority tip.

Define: Virtual machine, opcode, stack

Virtual machine, VM (a recipe translator every cook reads identically). A program that emulates a computer with memory, instructions, and a deterministic execution model.

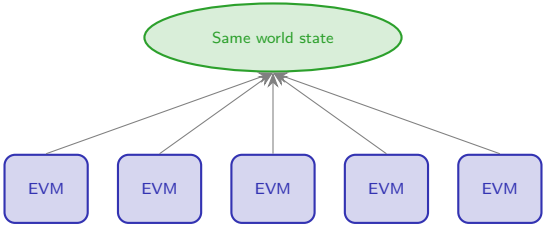
Opcode. A single instruction the VM understands, identified by a one-byte code. Examples on the EVM: ADD (0x01), MSTORE (0x52), SLOAD (0x54), SSTORE (0x55), CALL (0xF1).

Stack-based execution. Most opcodes pop their inputs from the top of a stack and push the result back. The EVM stack holds 256-bit words and is at most 1024 deep.

Why it matters. The EVM is THE program every node runs. Different node implementations (Geth in Go, Nethermind in C#, Reth in Rust) all execute the EVM identically because the spec is exact.

Vocab. Bytecode = a sequence of opcodes. Solidity compiles to bytecode.

Apply: EVM determinism means consensus



deterministic + replicated = consensus

What determinism rules out. The EVM has no `rand()`, no `time.now()`, no file or network I/O. Block timestamp (set by the proposer) and block hash are the only "external" inputs available to a contract. Any non-determinism would split consensus.

This is why on-chain randomness is hard. The fix uses commit-reveal schemes or VRFs (verifiable random functions); out of scope today.

Define: Gas, gas limit, EIP-1559 fees

Gas (a taxi fuel meter: distance times rate; the city sets the rate). A unit of computational work. Each EVM opcode costs a fixed amount of gas. ADD costs 3, SLOAD costs 2100 (cold) or 100 (warm), SSTORE up to 22100. Total gas of a transaction is the sum over all opcodes executed.

Gas limit. The maximum gas the sender allows for this transaction. If execution would exceed it, the transaction reverts but the sender still pays for the gas consumed up to the limit.

EIP-1559 fee market. Each block has a **base fee** (set by protocol, burned). Senders also offer a **priority tip** (paid to the proposer). Effective gas price = base fee + min(priority tip, maxFeePerGas − base fee).

Vocab. Burn = ETH is removed from circulation forever. Tip also called priority fee.

Apply: Gas in dollars (worked example)

Scenario. You send 1 ETH from Alice's EOA to Bob's EOA on mainnet. Base fee 20 gwei. Priority tip 1 gwei. ETH price $3,000.

- Gas used by a simple transfer: 21,000 (a protocol minimum).
- Effective gas price: $20 + 1 = 21$ gwei.
- Total fee in gwei: $21{,}000 \times 21 = 441{,}000$ gwei.
- Convert to ETH: $441{,}000 \times 10^{-9} = 0.000441$ ETH.
- Convert to dollars: $0.000441 \times 3000 = \$1.32$.

Of which: $21{,}000 \times 20$ gwei $= 0.00042$ ETH burned (gone forever); $21{,}000 \times 1$ gwei $= 0.000021$ ETH tip to the proposer.

Contract calls (e.g. token transfer) typically use 50,000 to 100,000 gas, so 2 to 5 dollars at these prices. Cell 9 of the Colab notebook lets you compute any scenario.

Vocab. gwei $= 10^{9}$ wei $= 10^{-9}$ ETH. Numbers are illustrative; live mainnet base fees vary by orders of magnitude.

Apply: Why base fees move (EIP-1559 dynamics)

Block fullness	Base-fee response
At target (50% of gas limit)	Base fee unchanged
Above target (more demand)	Base fee rises (up to $+12.5\%$ per block)
Below target (less demand)	Base fee falls (up to -12.5% per block)

Property. The base fee tracks demand. Periods of low congestion bring it toward zero; spikes bring it up. The 12.5% bound prevents overshoot.

Why this design? Pre-EIP-1559 (2021), users had to guess a "good" gas price; auctions were brittle. EIP-1559 gives a predictable base fee plus a tiny tip, which makes wallet UX simple ("set max fee at 2x base fee, set tip to 1 gwei").

Analogy. Surge pricing on Uber: high demand pushes the base price up; a small tip to the driver stays small. The protocol burns the surge so no validator can manipulate it upward.

Vocab. Gas target = the size at which base fee is unchanged. Today on mainnet this is 15M gas per block; the absolute limit is 30M.

We will define five dense terms together first, then apply them:

validator | **slot** | **epoch** | **attestation** | **finality**

One combined Define slide (next), then Apply slides.

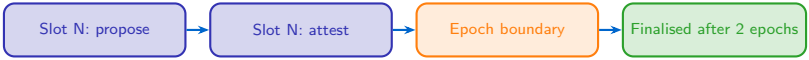
This is the densest section. Slow down; do not hesitate to ask.

Define: The five consensus terms (one slide)

Term	One-sentence definition
Validator	A staker who has locked 32 ETH into the deposit contract; in exchange they get to propose and attest to blocks.
Slot	A 12-second window in which exactly one validator is randomly chosen to propose a block.
Epoch	A sequence of 32 slots (so 6.4 minutes). Reward and penalty accounting happens per epoch.
Attestation	A signed vote by a validator about which block is the head of the chain at the current slot. Most validators attest to most slots.
Finality	A block is finalised when 2/3 of all staked ETH has attested to it across two consecutive epochs. Reverting requires destroying 1/3 of all staked ETH (slashing).

Vocab. Slashing = a penalty that destroys part of a validator's stake for misbehavior (double signing, etc.).

Apply: How a block is confirmed



A block proposed at slot N is attested by 32,000 validators in the next 12 seconds. After two more epochs (12.8 minutes total), the block is FINAL: reverting requires destroying at least 1/3 of all staked ETH (currently worth tens of billions of dollars).

This is why Ethereum has slow finality but strong finality. Bitcoin has instant inclusion but no notion of "this block is final"; you wait for confirmations as a probabilistic measure.

Vocab. Confirmations = number of blocks built on top. Final = a stronger property; reverting becomes economically catastrophic.

Apply: Justified vs finalised (Casper FFG)

- **Justified.** A checkpoint (the first block of an epoch) is "justified" once 2/3 of staked ETH has attested to it.
- **Finalised.** A justified checkpoint becomes "finalised" when the NEXT checkpoint is also justified.
- **Property.** An attacker who wants to revert a finalised block must attest contradictorily to TWO checkpoints; this is the "1/3 slashable" condition. Honest validators never sign contradictions.

In one image: justification is a vote; finalisation is a vote of confidence in the previous vote.

Analogy. A jury votes once to indict (justified). A second jury then confirms the indictment (finalised). Reversing the verdict requires both juries to retract under threat of contempt charges (slashing).

Vocab. Casper FFG = Casper the Friendly Finality Gadget, the protocol Ethereum uses to add finality on top of attestations. Designed by Vitalik Buterin and Virgil Griffith (2017).

We will define:

smart contract → ABI + function selector

→ event/log → ERC-20 → ERC-721

Apply slides show a worked HelloWorld and a worked ERC-20 Transfer.

Goal: read a Solidity contract and predict its on-chain effect.

Define: Smart contract, ABI, function selector

Smart contract (a vending machine; bolted to the wall on deploy day, never edited again). A contract account whose code (bytecode) is deployed once and executed by the EVM whenever someone sends a transaction whose to matches the contract address. Once deployed, the bytecode is immutable.

ABI, Application Binary Interface (the menu card on the front of the vending machine). A JSON description of a contract's external functions: their names, parameter types, return types. Without the ABI you cannot encode a function call.

Function selector (the button code on the vending machine: "B7" picks the right snack). The first 4 bytes of `keccak256("functionName(paramTypes)")`. Calldata for a function call is: 4-byte selector plus ABI-encoded arguments. The contract dispatches by reading the first 4 bytes.

Vocab. Immutable = cannot be changed after deployment. Upgradability is a pattern (proxies), not a built-in.

Apply: HelloWorld in three lines

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;
contract HelloWorld {
    string public greeting = "Hello, Class";
}
```

What is happening. The keyword `public` on a state variable auto-generates a getter function. The Solidity compiler emits bytecode plus an ABI fragment: `{type:"function", name:"greeting", inputs:[], outputs:[{type:"string"}], stateMutability:"view"}`.

Calling it costs zero gas because `view` functions are read-only and answered locally by the node, not via a transaction. Only state-changing calls cost gas.

Vocab. view = read-only declaration; the EVM rejects state writes inside a view function.

Define: Event and log

Event (a receipt printer at a checkout). A Solidity declaration like `event Transfer(address indexed from, address indexed to, uint256 value);` that, when emitted from a contract, writes a structured record into the transaction's "logs" field.

Log. The on-chain trace of an emitted event: up to 4 indexed topics (the function selector + indexed args) plus arbitrary unindexed data. Logs are cheap to write and easy for off-chain code (Etherscan, The Graph) to filter and index.

Why it matters. Logs are how you observe what happened on chain. A contract's "history" is the set of logs it emitted. ERC-20 token balances at any point in time are computed by replaying Transfer events.

Vocab. Indexed = stored in the searchable topics field. The Graph = a popular indexer; out of scope today.

Apply: The anatomy of an ERC-20 transfer

A user calls `token.transfer(0xBob, 100)`. The five things that happen:

1. Wallet ABI-encodes: `0xa9059cbb` (selector for `transfer(address,uint256)`) + 32 bytes of Bob's address + 32 bytes for the value 100.
2. User signs the transaction (S09), with `to` = token contract address, `data` = the encoded payload above.
3. Validators include the tx; the EVM runs the token's bytecode; it reads Alice's balance from storage, decrements by 100, increments Bob's balance by 100.
4. Contract emits `Transfer(Alice, Bob, 100)` log.
5. Block is included; logs become visible on Etherscan; off-chain indexers update.

Note. ERC-20 is just a Solidity **interface** (function names and event signatures). Any contract that implements those names is "an ERC-20 token". USDC, USDT, DAI all implement this same interface.

Vocab. Interface = a contract-shaped declaration without implementation. ERC-721 (NFTs) is a different interface for unique tokens.

Appendix: Define rollup, optimistic vs zk

Rollup (an express train on its own track, posting its passenger list back to the main station). A separate chain (the "L2") that processes transactions off the Ethereum mainnet (the "L1") and periodically commits a compressed batch back to L1. Inherits L1 security at L1 settlement cost.

Optimistic rollup. Assumes batches are valid; allows a 7-day fraud-proof window in which anyone can challenge an invalid batch. Examples: Arbitrum, Optimism, Base.

Zero-knowledge (zk) rollup. Each batch is accompanied by a cryptographic validity proof that the L1 verifies. No fraud window; finality is fast. Examples: zkSync, StarkNet, Polygon zkEVM.

Why it matters in 2026. Most real Ethereum activity in 2026 happens on L2s, not on mainnet. Same Solidity, same accounts, much cheaper gas.

Vocab. Settlement = where transaction results become final. L2s settle on L1.

Recap, in one line per concept

1. **Hash function:** deterministic fixed-output map; pre-image and collision resistant.
2. **Public-key crypto:** keypair lets you prove identity without revealing the secret.
3. **Signature:** σ from (privKey, msg); anyone with pubKey verifies; flipping a bit invalidates.
4. **State machine:** $(\mathcal{S}, \delta, s_0)$; Ethereum is one.
5. **EOA vs contract:** humans vs code as authority.
6. **MPT:** tree structure that is map AND commitment.
7. **Transaction:** signed (nonce, to, value, data, gas, fees) tuple.
8. **EVM:** stack-based deterministic VM running everywhere identically.
9. **Gas + EIP-1559:** base fee (burned) + tip (to proposer); demand-tracking.
10. **Validator, slot, epoch, attestation, finality:** 32-ETH stakers, 12-sec slots, 32-slot epochs, 2/3-stake votes, 2-epoch finality.
11. **Smart contract:** deployed-once immutable bytecode at a contract account.
12. **ABI + selector:** function names plus 4-byte calldata prefix.
13. **Event + log:** emitted structured record; off-chain indexable.
14. **ERC-20 / ERC-721:** standardised contract interfaces.
15. **Rollup (optimistic / zk):** L2 chain anchoring batches to L1.

If any line surprises you on second read, that is the term to revisit in the Colab notebook.

References and the Colab notebook

Companion Colab notebook (compute every concept yourself).
https://colab.research.google.com/github/Digital-AI-Finance/Cryptoeconomics-Blockchain/blob/master/lectures/ethereum_teaching/notebooks/ethereum_concepts.ipynb

Primary references.

- Ethereum Yellow Paper (formal spec): https://ethereum.github.io/yellowpaper/paper.pdf
- EIP-1559 spec: https://eips.ethereum.org/EIPS/eip-1559
- Casper FFG paper (Buterin and Griffith, 2017): https://arxiv.org/abs/1710.09437
- EIP-4844 spec (blob transactions, ADV2): https://eips.ethereum.org/EIPS/eip-4844
- ERC-4337 spec (account abstraction, ADV4): https://eips.ethereum.org/EIPS/eip-4337
- ethereum.org Learn hub (curated, beginner): https://ethereum.org/en/learn/

Five advanced sidebars (ADV1 to ADV5) follow.

ADV1: Verkle trees and stateless Ethereum

The problem. Today's MPT proofs are large (1KB per account access). A future where every node only holds the current state and proves state on demand needs much smaller proofs.

Verkle trees use vector commitments based on KZG polynomial commitments instead of Merkle hashing. Result: proofs are tens of bytes, not hundreds. The tradeoff: trusted setup for the KZG ceremony.

Status (2026). Active research. The Pectra and subsequent forks are evaluating partial transitions. *Stateless clients* (validators that download only headers + proofs) become viable once Verkle ships.

Vocab. Trusted setup = a one-time ceremony to generate cryptographic parameters; if at least one participant is honest, the parameters are safe.

ADV2: Blob transactions and proto-danksharding (EIP-4844)

The problem. L2 rollups post compressed transaction batches back to L1. Before EIP-4844 these batches sat in regular calldata at full gas price. L2 fees were dominated by L1 calldata cost.

Blob transactions introduce a new transaction type carrying "blobs" (125 KB chunks) attached to the block but NOT executed by the EVM. Blobs are committed via KZG and pruned after 18 days. Cost: a separate blob-fee market in EIP-1559 style.

Result. L2 fees dropped 10x to 100x after Dencun (March 2024). Most of today's user fees on Arbitrum / Base / Optimism are blob costs, not L1 calldata costs.

Vocab. Calldata = the data field of a transaction. Pruned = deleted after a fixed window because rollups already have what they need from the commitment.

ADV3: Gas refunds, cold vs warm storage

Cold vs warm. EIP-2929 split storage access into two tiers. The first SLOAD of a slot in a transaction is "cold" and costs 2100 gas; subsequent reads of the same slot in the same transaction are "warm" and cost 100 gas. Same for accounts.

SSTORE refund. Setting a non-zero slot to zero used to refund up to 15,000 gas. EIP-3529 (London, 2021) capped refunds at 20% of the transaction's gas used and reduced the per-slot refund to 4,800. *Reason.* Refund-driven gas-token contracts (CHI, GST2) were inflating block usage artificially.

Practical impact. Optimised contracts pack multiple values into one storage slot to amortise the 20,000-gas first-write cost.

Vocab. Gas token = a contract that "stored" gas during low-fee periods to release during high-fee periods; killed by EIP-3529.

ADV4: Account abstraction internals (ERC-4337)

The goal. Relax the EOA-only-signs rule. Let smart contracts be first-class accounts with arbitrary authorisation logic (multi-sig, social recovery, biometrics, sponsored gas).

The five pieces.

- **UserOperation.** A pseudo-transaction the smart-account wants executed. Lives in a separate mempool.
- **EntryPoint.** A singleton contract that validates and executes UserOperations.
- **Smart account.** The contract that owns assets; implements an `validateUserOp` function defining authorisation.
- **Bundler.** A new actor that batches UserOperations into a regular transaction sent to the EntryPoint.
- **Paymaster.** An optional contract that pays the bundler so users do not need ETH for gas.

ERC-4337 ships at the application layer (no consensus change). EIP-7702 adds a thinner protocol-level path for EOAs to "act like" smart accounts.

ADV5: MEV and proposer-builder separation (PBS)

Define MEV. Maximal Extractable Value. The maximum value a block proposer can extract from ordering, including, or excluding transactions in their block, beyond standard block reward and gas fees. Sources: arbitrage, liquidations, front-running.

The problem. If solo validators ran their own MEV searchers, well-resourced players would dominate validator economics, centralising staking.

Proposer-Builder Separation (PBS).

- **Searchers** find MEV opportunities and submit bundles to builders.
- **Builders** construct full blocks and bid for inclusion via a relay.
- **Proposers** (validators) just sign whatever block won the auction; they do not see the contents until commitment.

Today. 90% of mainnet blocks are built via mev-boost relays. Enshrined PBS is on the roadmap.

Vocab. Front-running = inserting your tx before someone else's known pending tx to capture value (e.g. arbitrage on a known DEX swap). Relay = trusted intermediary between builders and proposers.