Unleash the Cores

@gavofyork


Polkadot
JAM

JAM is a Rollup Reactor

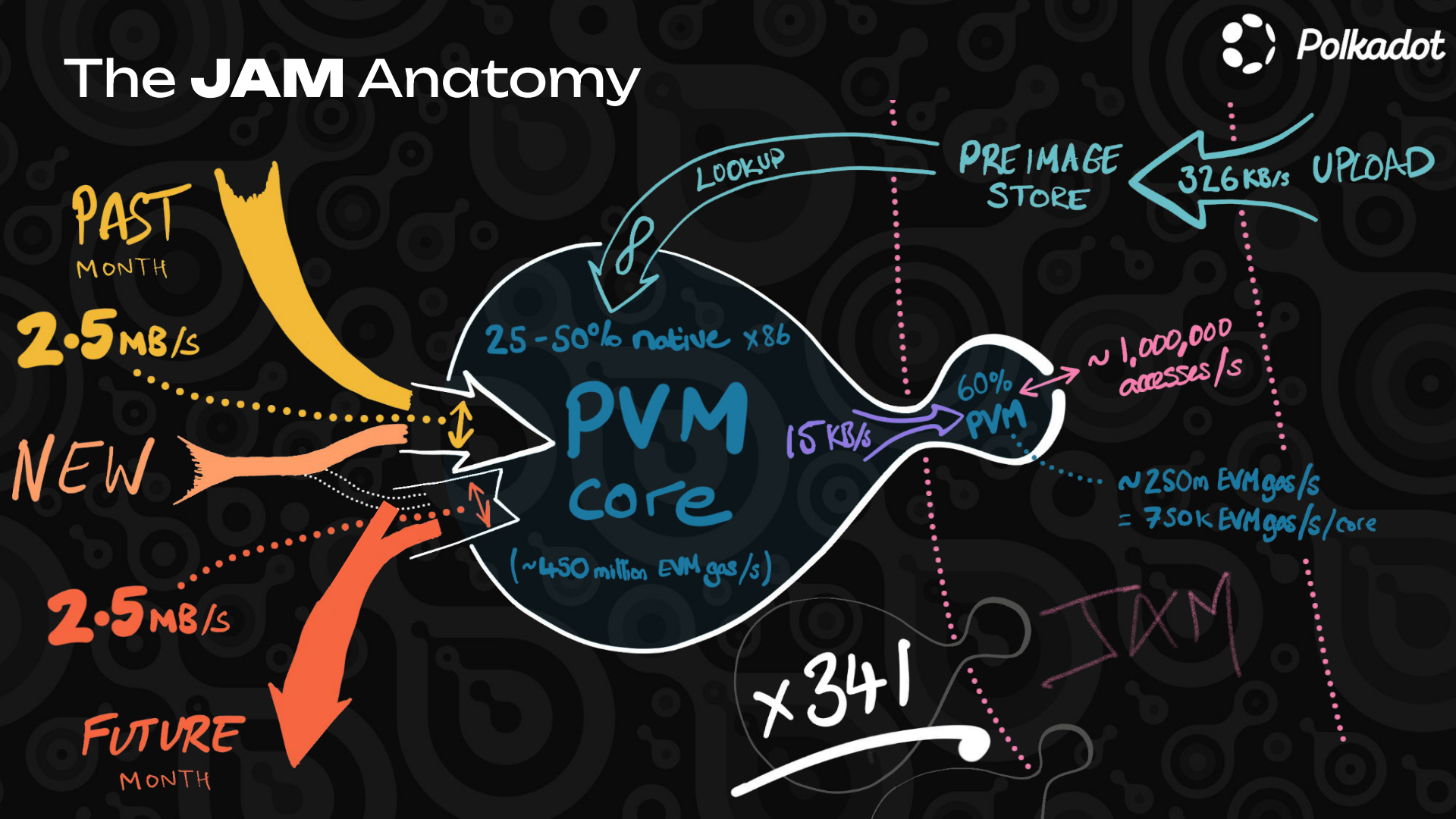
- Secure Audited Roll-Ups (High-throughput, Low-latency)
- Multi-Modal High-Resilience Data Availability
- PVM-based:
  - Metered
  - X86 recompilable [O(n), ***fast***]
  - LLVM/RISC-V tooling
- Transactionless with Agile Coretime

The **JAM** Data Topology

Polkadot

The **JAM** Anatomy

Polkadot



JAM: Back of the Envelope

Current performance targets for initial rollout:

- **2.06 PB** Total Multi-Modal HRDA **28d** @ **850 MB/s** I&O
- **150,000,000,000+** EVM equiv gas/sec in-core
- **5MB/s** coherence bandwidth
- **250,000,000+** EVM equiv gas/sec (fully-coherent)

Watch this space for updates and empirical results

* Not taking into account L2 tech or pre-compiles

Polkadot

The JAM Approach

Protocol-oriented and decentralized

GRAY PAPER

JAM PRIZE

JAM TOASTER

The **JAM** Approach: The **GRAY** PAPER

accept as the canonical "version" of the world of Ether
The state can include such information as account
ances, reputations, trust arrangements, data pertai
to information of the physical world; in short, anyt
that can currently be represented by a computer is ac
sible. Transactions thus represent a valid arc between
states; the 'valid' part is important—there exist far
invalid state changes than valid state changes. Invalid
changes might, e.g., be things such as reducing an acc
balance without an equal and opposite increase elsew
A valid state transition is one which comes about thr
a transaction. Formally:

$$\boldsymbol{\sigma}_{t+1} \equiv \Upsilon(\boldsymbol{\sigma}_t, T) \tag{1}$$

where Υ is the Ethereum state transition functio
Ethereum, Υ, together with $\boldsymbol{\sigma}$ are considerably more
erful than any existing comparable system; Υ allows
ponents to carry out arbitrary computation, while $\boldsymbol{\sigma}$ al
components to store arbitrary state between transact
Transactions are collated into blocks; blocks are cha

4. Overview

As in the Yellow Paper, we begin our formalisms by recalling that a blockchain may be defined as a pairing of some initial state together with a block-level state-transition function. The latter defines the posterior state given a pairing of some prior state and a block of data applied to it. Formally, we say:

$$\sigma' \equiv \Upsilon(\sigma, \mathbf{B}) \tag{11}$$

Where σ is the prior state, σ' is the posterior state, B is some valid block and Υ is our block-level state-transition function.

Broadly speaking, JAM (and indeed blockchains in general) may be defined simply by specifying Υ and some *genesis state* σ^0.[7] We also make several additional assumptions of agreed knowledge: a universally known clock, and the practical means of sharing data with other systems operating under the same consensus rules. The latter two were both assumptions silently made in the *YP*.

The GRAY PAPER

JAM is protocol, not software

The formal specification precedes implementation, helping give equal opportunities to implementation teams and avoiding the pit of centralization around a single code repository and team.

The GRAY PAPER

JAM is protocol, not software

For more information, visit:

https://graypaper.com/

GRAY PAPER Tour

A global tour of lectures to understand **JAM**.

I'll be appearing at universities around the world in a series of lectures to take viewers through the **GRAY** PAPER and explain each part in depth.

GRAY PAPER Tour

A global tour of lectures to understand **JAM**.

Details on dates, places and topics coming soon!

The JAM PRIZE

A resilient, decentralized protocol requires multiple viable implementations

The **JAM** prize is a fund and contest launched and underwritten by the Web3 Foundation to both help ensure many viable impls of **JAM** and to maximize protocol expertise in and around the community.

The JAM PRIZE

A resilient, decentralized protocol requires multiple viable implementations

Prize fund: **10 million DOT**

Split between several language sets, with several implementation tiers.

More details coming soon.

The JAM TOASTER

Facilitating a data-driven approach for an optimally parameterized protocol

The **JAM** TOASTER is a development facility which can closely approximate a production network and allow for deep analysis in a controlled environment.

Open for use for all implementation teams.

The JAM TOASTER

Facilitating a data-driven approach for an optimally parameterized protocol

16,384 AMD TR 5 GHz cores

16 GB total L2 cache

32 TB RAM, 8 channel

20 PB secondary storage

10 Tbps ethernet switching

JAM Rollout Proposal

1. Gray-Paper Publication
2. In-Principle Ratification of Approach and Scope
3. Activation of Implementation Prize
4. Optimization & Parameterization of Protocol
5. Final Ratification of JAM Protocol
6. Three Months after Three Viable Validator Impls
7. Migration, Transition and Prizes

The POLKA**JAM** Governance Proposal

The scope and direction of **JAM** as specified in the GRAYPAPER describe a protocol which shall replace the Polkadot Relay-chain.

The Polkadot Fellows shall be responsible for ratifying, and thereby defining conformance and performance tests for, the Polkadot **JAM** Protocol.

The Polkadot Fellowship shall be responsible for the development and deployment of a replacement service for the Polkadot Relay-chain on the **JAM** protocol.

The Polkadot Relay-chain shall be upgraded to halt on signal from the Polkadot Fellows once it is determined that the Polkadot **JAM** chain, the Polkadot Relay-chain replacement service, all required system parachains and all requisite tooling are ready for deployment.

Readiness shall include a minimum of: a successful independent professional audit of the protocol specification and three viable, independent and audited validator-node implementations which pass the published conformance and performance tests for the Polkadot **JAM** Protocol.

Polkadot

Let's **JAM!**

Thankyou!

– @gavofyork


Polkadot

Polkadot

JAM Proposal: Stage 1

- Relay-chain becomes JAM chain (stop-migrate-start)
- Formal and complete protocol specification first
- Minimal complexity, solve only one problem
- Implementation prize (W3F-backed prize-fund of $100m)
- Create **CoreChains** service to host parachains on JAM
- Ratify protocol alteration with DOT holders
-
- Create **CorePlay** service for a multi-linguistic actors env.

Polkadot

The JAM Concept

- Input is arranged into **Work Packages**
- A Work Package is anchored to a recent block;
- and contains one or more Work Items.
- Work Items are `refine()`-ed into Work Results
- Thus a Work Package uses one core for one timeslot to be transformed into a **Work Report** (containing the corresponding Work Results)

JAM: Join-Accumulate Machine

- From CoreJam (Collect-Refine-Join-Accumulate)
- JAM is a candidate design for a future iteration of Polkadot Relay
- A new blockchain concept:
  - (What Ethereum folks might call) A “secure-rollup domain-specific chain”
  - Transactionless (no transaction distribution/pool 🎉)
  - Permissionless code- & data-hosting
  - PVM-based (a refinement of RISC-V ISA)
  - Safrole-based (a refinement of Sassafras)
  - Parachains- & Agile Coretime capable
  - Optimized network architecture & pipelined
  - Fixed parameters, non-upgradable
  - Token-hosting (u64), Staking & Coretime-sales agnostic

Polkadot

The JAM Concept

Polkadot

The JAM Concept

- Domain-specific chain (a **DSC**? 😬)
- Accepts outputs of “rollups”
- Guarantees the output is correct
- Integrates output into some shared state

Polkadot

The JAM Concept

- Shared state is arranged into **Services**
- Creation of a service is permissionless
- Services have a single piece of code with 3 entry points:
  - `refine`: Do the rollup (mostly stateless)
  - `accumulate`: Integrate the output into shared state
  - `on_transfer`: Accept some memo/tokens from some other service

Transactionless

- No transactions.
- Still extrinsic information; five types:
  - Guarantees (reports of the output from `refine` computation)
  - Assurances (attestations of the availability of Work Packages)
  - Judgements (dispute outcome)
  - Preimages (lookup data which a service has requested)
  - Tickets (anonymous entries into future block production lottery)
- Minimal opinionation

Polkadot

Permissionless

- JAM-chain hosts services' code, data and state
- Introducing a new service is permissionless
- Services are limited in their code, data and state only by deposit requirements
- Example service: "Parachains" (i.e. "Polkadot 1.0")

Service model

- Refine
  - Accepts up to 6MB of data
  - Yields up to 90KB of data
  - May execute for up to 6 seconds of PVM gas
  - Gets contextual information on other contents of Work Package
  - May make preimage lookups, invoke PVMs
- Accumulate
  - Accepts multiple outputs from Refine
  - May execute for up to 10ms of PVM gas per output
  - Stateful: may transfer funds, alter state, read other services' state
  - May create services and upgrade its code and request preimage availability
- OnTransfer
  - Stateful: may alter state and read other services' state

PVM-based

A refinement of the RISC-V ISA

PVM is designed to be a simple, secure, metered, deterministic ISA capable of hosting arbitrary code.

It is optimized to be **blazingly fast** on real hardware.

Safrole -based

A refinement of the Sassafras consensus algorithm

Safrole is a refinement of Sassafras, a SNARK-based consensus algorithm providing fork-resistant, constant-time anonymised block production.

Polkadot 1.0 capable

JAM can host Polkadot 1.0 parachains

Chain PVF would be retargeted to PVM (which we expect to be an official target before launch).

Parachains ++

JAM brings key features to parachains

JAM ushers in several key features in the parachains SDK including:

No-Benchmark pallets

Accords

Full XCMP

Agile Coretime

JAM is fully compatible with Agile Coretime

JAM makes full use of the Agile Coretime sales paradigm.

The Authorization layer allows for non-parachain assignments of Coretime.

Optimized

JAM does away with gossip

Quik underpins **JAM** allowing an efficient m2m topology.

Gossip is avoided within the validator set, replaced by robust grid-diffusal and targeted publication enabled by Safrole.

Pipelined

JAM enables temporal parallelization

Unlike Ethereum and Polkadot, JAM headers include the prior state-root, not posterior. Furthermore, the final 95% of JAM's state transition is both parallelisable and infallible.

This allows for pipelining in block production, enabling block-compute times much closer to block period.

Fixed

Permissionless services allow JAM to fix key qualities without sacrificing flexibility

Unlike Polkadot, JAM is a fixed-function protocol. However, the fixed-function pipeline is itself Turing-complete with PVM-invocation capability.

This allows for a much simpler core protocol and provides substantial opportunities for optimization.

Coretime Sales, Staking and Parachains logic are still 100% upgradable.

Token-hosting, otherwise agnostic

JAM provides underlying reserve functionality for DOT, allowing its economics to be credibly fixed

Since JAM is a fixed-function protocol, it now becomes possible to introduce economic guardrails to the DOT token which a majority of stakeholders cannot arbitrarily alter.

It is entirely agnostic to other protocol elements previously hosted on the Relay-chain such as governance, staking and parachain-service logic.

JAM TOASTER

A super-computer capable of hosting JAM in its entirety: a rig for Testing, Observation, Analysis, Scaled-Trials, Experimentation and Research

The Polkadot Palace, an ecosystem facility under development in Lisbon, will play host to a 12,276 core, 16TB RAM super-computer capable of hosting the full JAM network in its entirely.

It will be used to do in-depth profile, usage-simulation and monitoring to help ensure that the deployed JAM-chain performs to expectations.

Polkadot

JAM + Substrate

Benchmarks optional

JAM averts the need for benchmarking code in most circumstances

Since JAM's PVM provides metering for PVFs, benchmarking becomes much less of an issue.

Benchmarks may still be useful for performance or where functions may take longer than a block of time.

XCMP

JAM requires full XCMP

Since JAM normalizes the execution platform on which the PVF is based, there is no option to fudge limits.

XCMP, which keeps PoV sizes within sensible limits, therefore becomes a requirement.

Accords

JAM enables functionality

The JAM model applies categorically less opinionation to what cores can be doing.

Mixed-code functionality, as required by Accords, becomes not only possible, but near-trivial to implement.

DA/ZK ready

JAM enables mixed-resource consumption models

While JAM validators perform a highly normalized task, work packages may be formulated entirely at the builder discretion.

Different services may have different resource envelopes and be combined in a package to maximize overall value of the JAM service profile.

The Road to JAM

The Road to JAM

- ✅ Preliminary RFCs (Sassafras, CoreJam)
- ⏳ Draft of proposal [soon!]
- ⏳ Initial implementations [soon!]
- ⏳ Open RFC for discussion and further evolution [soon!]
- 📅 Final draft and RFC [3-12 mo?]
- 📅 Ratification (Fellowship/Governance) [3-12 mo?]
- 📅 Implementations (min. 3), Substrate tooling [6-18 mo?]
- 📅 Deployment [8-20 mo?]