DELPHI CREATIVE

Fantom 2.0

Designed by: Miguel Mendes

CONTENTS

Introduction

As blockchains scale, “the modular vs monolithic” dichotomy has become a clear differentiating factor in how blockchain architectures are designed at a fundamental level.

Fantom, which pursues the monolithic approach, **offers an alternative to the modular architecture pursued by Ethereum L2s**. While both architectures have their tradeoffs, a clear advantage of monolithic solutions is their simplicity. As a monolithic chain, Fantom sidesteps the intricate complexities inherent in multi-layered modular designs and focuses purely on optimizing the scalability of a single chain with a shared state.

To this end, **Fantom has developed a new innovative Sonic client that pushes the limits of monolithic design**. In this report, we will walk you through the Fantom tech stack as it transitions from its current version, Fantom Opera, to its next iteration Fantom Sonic.

Sonic is the result of a multi-year research and engineering effort from the Fantom team which drastically improves the performance of the Fantom chain by better utilizing hardware and the network resources available to nodes. Sonic has demonstrated impressive demonstrates proven performance results, showcasing Fantom’s commitment to simplicity and scalability.

Fantom Sonic (aka. Fantom 2.0)

Development of Sonic has involved performance engineering; an evolutionary, iterative cycle of:

I. **Observation/experimentation**
II. **Identification of immediate bottlenecks**
III. **Elimination of immediate bottlenecks**

Which was repeated over and over again in search for peak performance - i.e., what is the best possible performance if the software is highly tuned. As a result, Sonic encompasses several upgrades to the Fantom tech stack, introducing the new virtual machine FVM, the new database storage Carmen, and optimizations on the transaction pool that is part of the consensus protocol of Fantom. Importantly, the implementation of these upgrades doesn't require a hard fork on the network.

Sonic is currently being evaluated under two testnets

CLOSED & OPEN

Sonic is currently being evaluated under two testnets; closed & open. The closed testnet is meant to test Sonic's maximum theoretical performance limits, while the open testnet is for the dev and user experience. Both have led to promising results.

During closed testnet Sonic was put under load with synthetic transactions whose goal is to mimic a real-life scenario based on historical Fantom data. The transaction breakdown was **10% token transfers, 64% erc20 mints & transfers** with **the rest being Uniswap style swaps**. Under this load, **Sonic managed to process around 2,048 TPS or 400 million gas per second with a 1.1 second time to finality.**

This is a 65x increase in throughput compared to the current Fantom Opera client which caps at around 30 tps under similar transaction types, primarily due to limitations of the EVM.

Furthermore, archival and validator nodes under Sonic were able to reduce their storage burden compared to Opera radically.

The outcomes of the closed testnet are shown below:

Sonic and Opera Comparison

General	Sonic	Opera
Minimum Consensus Quorum	15	14
Validators Stake Share	Equal	Natural
Peak Gas Per Day	34,560	283.96
Peak Gas Per Second	405	3.28
Peak Tx Per Second	2100	21
Archive Node DB Size	180	2106
Archive Node DB Size At 518 Tx	1000	10893
Live Pruned DB Size	351	1904
Offline Pruned DB Size	N/A	1194

Validator	Sonic	Opera
Live Pruning Support	Yes	No
Minimum Pruning Downtime	None	30 Min
Full Offline Pruning Downtime	None	3-8 Hours
Validator Hardware Cost	Low	Higher
Validator Operating Cost	Low	Higher
Validator Operating Risk	Low	Higher
Avg. Offline Pruning Period	None	166
DB Size Growth Rate	.74	17.78
Time To Full Sync	<=2 Days	<=4 Days

DELPHI DIGITAL

The open testnet was recently launched and will continue to operate for a few months until Sonic can launch in mainnet.

Mainnet Launch
SPRING 2024

Fantom Virtual Machine (FVM)

When it comes to throughput, chains are typically not immediately bottlenecked by their consensus protocols but by the VM they run. Historically, this has been EVM for Fantom but that is now about to change. EVM is not a virtual machine optimized for scalability, and was a restraining factor for Fantom's performance, leading to the development of FVM; a custom VM optimized for performance.

While FVM is a radical new design one of the main goals of FVM is to maintain the same userX/devX for the app layer. To this end, FVM maintains a good level of compatibility with the Solidity compiler and all the existing dev tooling. **Devs can write their code just as they were before, using Solidity, Vyper, etc**. Furthermore, **apps will continue to be deployed and stored on-chain in EVM-compatible bytecode**. As such, what works on Fantom Opera will continue to work on Fantom Sonic.

The EVM → FVM translation happens entirely on the client side on the fly through a technique known as “dynamic translation”. While the translation is seamless for the end users, FVM uses an entirely different instruction set and encoding scheme amenable to fast execution. Once translated, the application logic can be executed at a much faster pace using the FVM instruction set and encoding scheme.

FVM also has the notion of “super instructions”: bundles of instructions amalgamated to another instruction. Super instructions can be executed in a single dispatch achieving much faster speeds compared to multiple instructions being executed one by one. Super instructions won’t be available for the first Sonic release but will be released in a later performance upgrade.

Carmen Database Storage

Carmen is Sonic’s new database storage for Fantom’s chain state (account balances, nonces, smart contract storage and code, etc.). At a high level Carmen’s design stems from the recognition that archival nodes and validator nodes have different access patterns that require different implementation techniques. To this end, Carmen introduces two different databases: **liveDB** and **archiveDB**.

LiveDB is for validators to run. It stores the current world state and is optimized to process blocks as fast as possible. **ArchiveDB, on the other hand, contains historical blocks and states, using data versioning**. It can best be be thought as a time machine to go back and query a certain state in the past for a block number. Archive nodes run the archiveDB (as well as liveDB to stay in sync) and mainly aim to respond to read-only RPC requests.

In Sonic FVM interacts with both liveDB and archiveDB. Separation and specialization of the two databases has been a critical factor behind Sonic’s performance which has shown to surpass the traditional Opera solution where validators and archival nodes share the same single database design.

Amongst other things, the separation of the two databases has paved the way for live pruning; validators maintaining liveDB can prune historical data while they continue to validate new blocks. This drastically alleviates their storage cost by reducing their storage requirements by up to 90%; as per the observed testnet results. All else equal, this should encourage wider participation in the network, as it signifcaintly reduces operational costs to run validator nodes.

Five different schemas are being explored/implemented as a data structure for the liveDB and archiveDB as we speak. **Schema 3**, which has been the subject of recent open/closed testnets, **uses a flat-storage data structure instead of the Merkle Patricia Trie data structure of the EVM.**

In EVM the state of the chain's state is stored in a Merkle Patricia Trie (MPT)

A key-value store with parent-child relationships. Values in MPT are represented in leaf nodes at the bottom. They are then recursively hashed in pairs to compute parent nodes and, eventually the single Merkle at the top representing the whole trie.

In such a trie-like structure, **each read and write into the database requires traversing the tree, which in turn requires several random accesses to the disk**. Random disk accesses are notoriously intensive and slow operations. What's more problematic is that, as the state grows and the tree becomes larger, the number of disk accesses required for each read-and-write operation ratchets up, resulting in even slower transaction processing times. Finally, changes in leaf node values must be reflected on the rest of the trie. This requires many expensive hashing operations to compute the new parent nodes including the final root. Put simply, **as the cumulative activity on the chain grows, so does the time it takes to verify it**.

Schema 3 makes drastic changes to the storage system to flip this dynamic. It replaces Opera's MPT with an array-based flat storage schema and instead of storing state indirectly as a key-value store, it implements a file-based StateDB. This reduces the need for random disk accesses and expensive hashing operations to update the trie and thereby simplifies data retrievals. More precisely, **StateDB allows constant-size reads and writes regardless of the size of the state**.

On the flip side, Schema 3 design isn't free of tradeoffs. While Merkle Trees are slow and expensive data structures to maintain and update, they are very handy for light nodes. Given a relevant Merkle inclusion proof (which is very succinct thanks to the trie-like structure), any light nodes can directly verify any individual account or storage data. This is important from an end-user verifiability perspective and moves users away from a world where most users trust a centralized RPC provider to read their blockchain data. While LiveDB can be perceived as a lightweight node given its low resource usage, as **Schema 3 doesn't adopt a trie-like structure, witness proofs for a single value won't be as concise as it'd be otherwise**.

Lachesis, DAG-based aBFT Consensus

Lachesis is Fantom's DAG-based aBFT consensus mechanism. Lachesis pre-dates Sonic and has always been a fundamental part of the Fantom tech stack. While consensus wasn't the bottleneck under Fantom Opera, it could become the new bottleneck under Sonic. While Sonic doesn't make any consensus-related changes to Lachesis, it enhances it by optimizing the transaction pool.

Now let's take a closer look at what Lachesis is.

Lachesis is a finalizing, asynchronous, leaderless, DAG-based consensus protocol. Below we elaborate on these properties to see how Lachesis compares to other consensus algorithms.

Lachesis Is Asynchronous & Finalizing

According to the CAP theorem, **consensus protocols can't remain both live and safe during network partitions.** They must favor one or the other.

Based on this, we can split consensus protocols into two classes:

Dynamically Available

Dynamically available protocols are liveness-favoring. They can make progress even when only a small fraction of consensus nodes are present.

The typical example here is Bitcoin's Nakamoto/ PoW consensus that follows the longest chain rule; the chain continues to grow regardless of how many miners (hash rate) there are. There are also dynamically available protocols in the PoS world such as Cardano's Ouroboros or Polkadot's BABE.

Finalizing

Finalizing protocols, on the other hand, are safety-favoring. Cosmos' Tendermint, Solana's pBFT, Flux's Hotstuff, and Avalanche are all finalizing protocols.

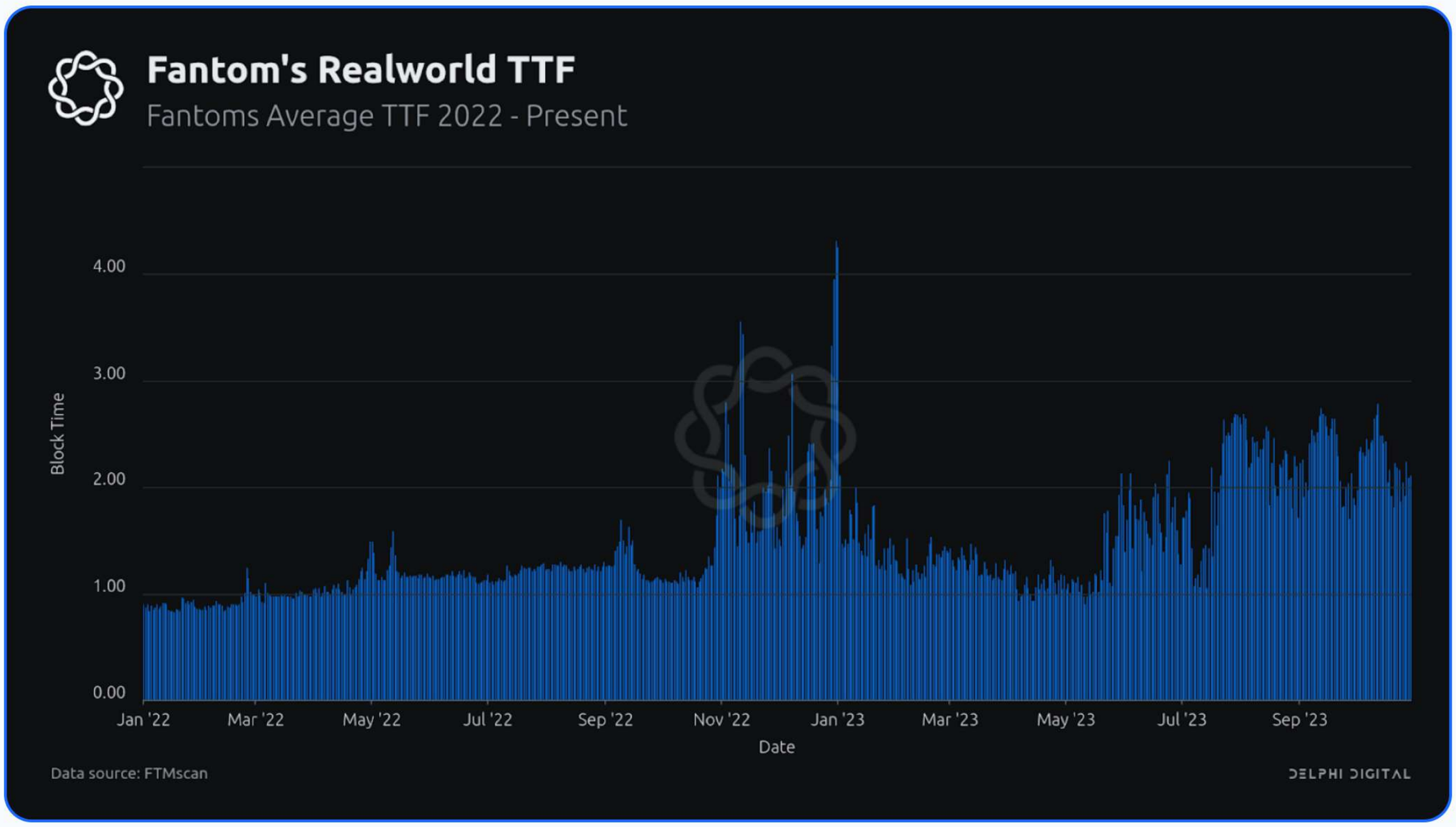
These protocols are all PoS-based protocols and all have a notion of a quorum, where blocks voted by the quorum (usually defined as ⅔ of the total stake) are deemed final.

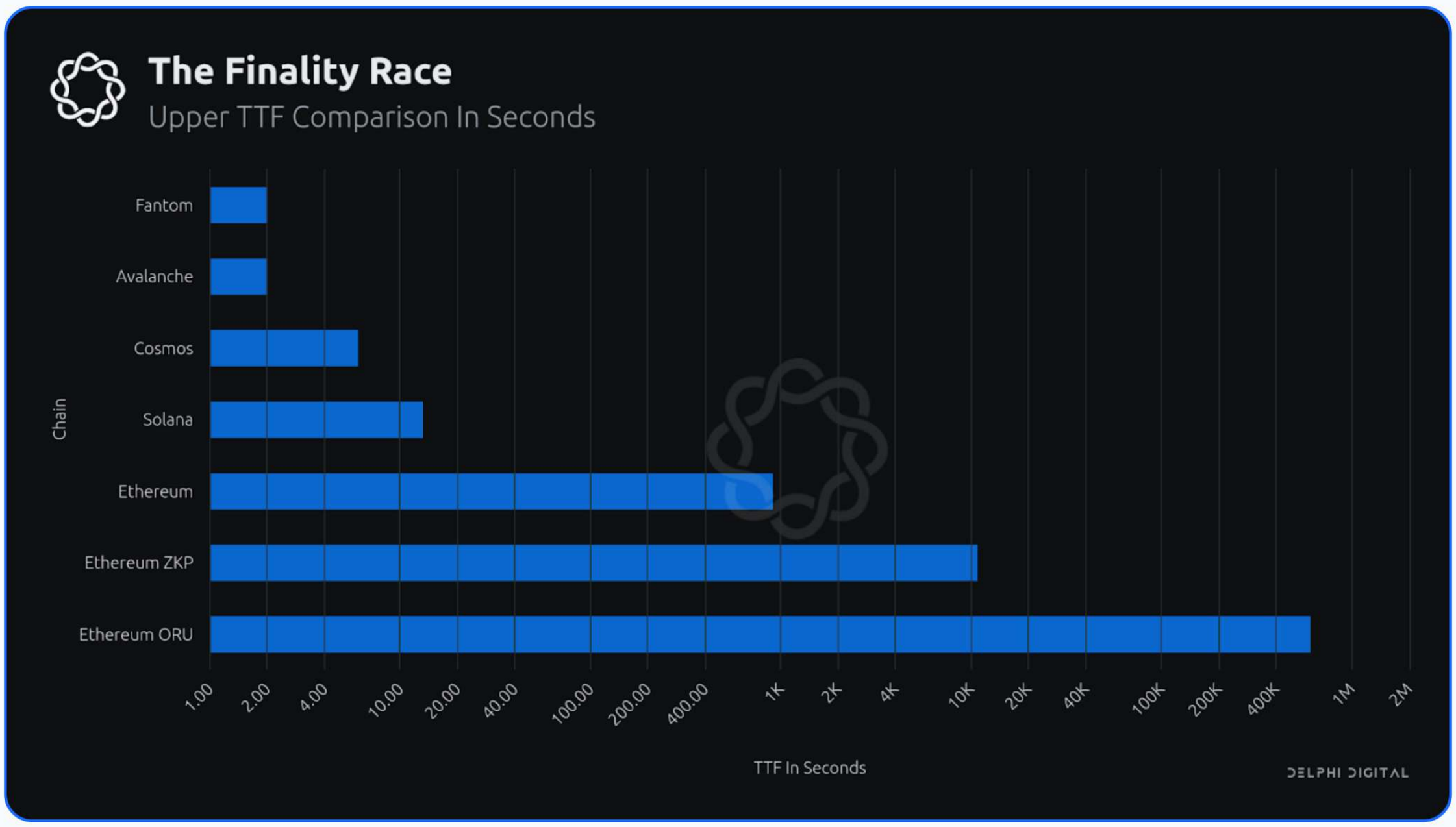
Typically dynamically available protocols assume synchronous networks where all messages are expected to arrive at their destination within a predefined bounded time. **Synchronous consensus protocols can't tolerate (remain safe during) large network latencies exceeding the expected time-bound**. For example, Ethereum post merge has experienced a 7-block reorg, in part, due to unmet synchrony conditions. Similarly, in Bitcoin, if *msgs* don't propagate across the network between the target block time (10 mins), the chain may experience detrimental forks.

Finalizing protocols can remain safe (have no forks) even under asynchrony. Asynchrony is a weaker assumption because asynchronous networks guarantee *msgs* will **eventually** arrive but otherwise don't assume a delivery time. Fantom's Lachesis uses aBFT consensus, which is finalizing and safety favoring. More precisely, **Lachesis can remain safe even under asynchrony.**

Unsurprisingly, if the network latencies are higher than expected it may take more time for finality to be reached. Here there is a fundamental tradeoff between throughput and finality; **more tps can lead to longer processing times and slow down time to finality**. However, Lachesis won't experience any forks even when messages in the network are arbitrarily delayed for whatever reason.

In practice, Fantom (Opera or Sonic) is one of the fastest chains in terms of time to finality. We can see this in the below charts. Sonic enables Fantom to process more transactions, while maintaining the same time to finality that Opera has been operating under.





Finality can become a heated debate in crypto. This is partly because finality can be seen as a confirmation rule at the discretion of users that varies from one user to another, rather than an inherent property of a chain.

The typical case study here is a user buying a coffee vs a user transferring millions of dollars. For example, in the case of Bitcoin, the merchant selling the coffee can deem the transaction to be final as soon as they see it in the mempool, whereas the user receiving millions would wait multiple blocks for finality.

Similarly, **a full node of an optimistic rollup can be 100% sure that a transaction won't revert as soon as its data is posted on Ethereum and Ethereum finalizes (15 mins).** This is because they execute and verify every rollup transaction on their client. On the other hand, a smart contract on Ethereum doesn't execute rollup transactions. Therefore it can't act on a withdrawal transaction from an optimistic rollup until the dispute challenge period is over (typically configured as 7 days).

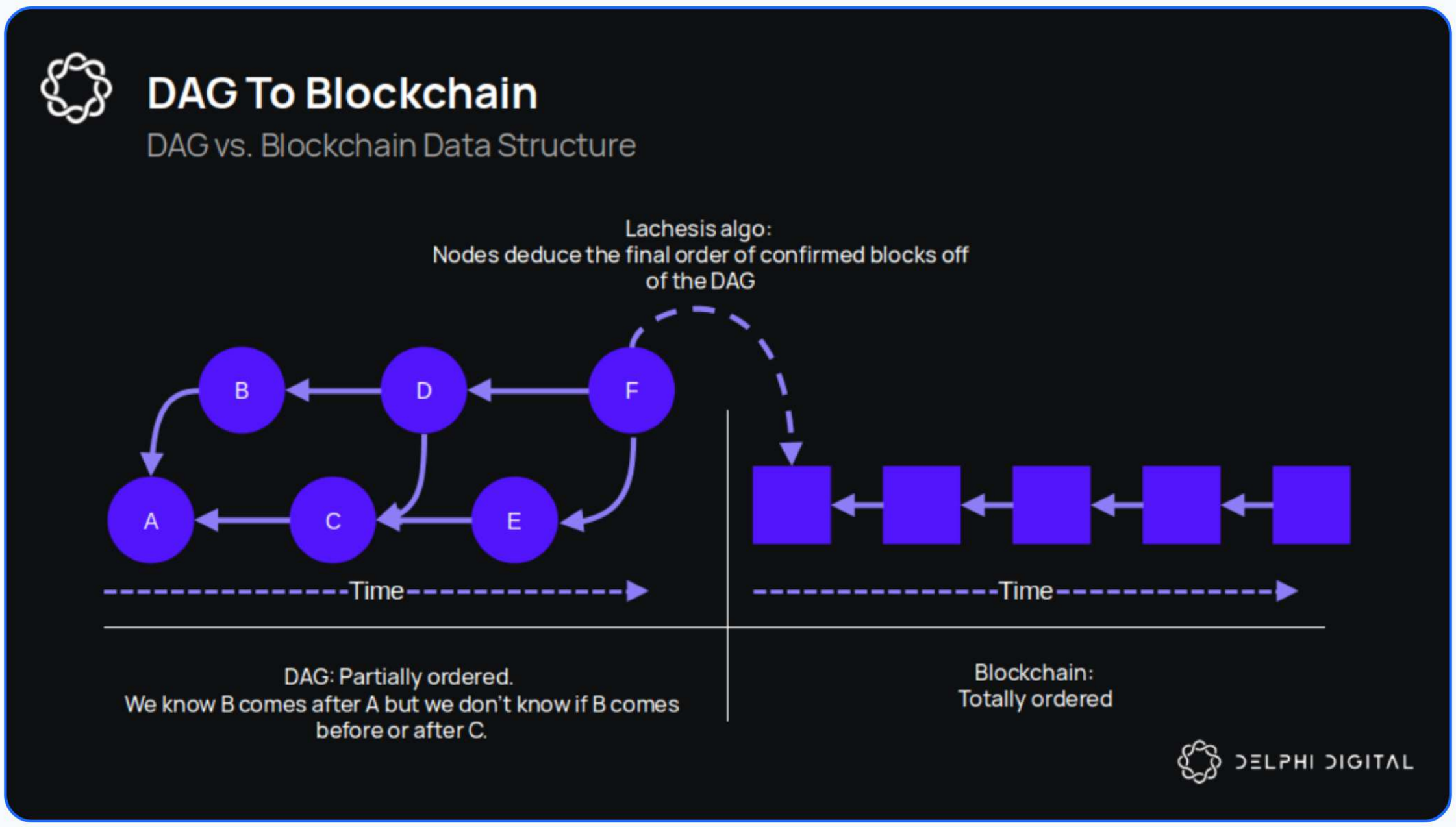
Nonetheless, there is something beautiful about a fast time to **full** finality, where everyone can be mathematically certain that ⅔ of validators are backing a particular state of the chain with their full stake. It's unarguably a neat feature that Fantom, as one of the fastest monolithic chains, possesses.

Lachesis Is DAG based & Leaderless

Perhaps the most differentiated aspect of Lachesis is that **it combines aBFT** with **a directed acyclic graph** (DAG) which lays the foundation for its leaderlessness.

Almost all consensus protocols widely used in blockchains need a leader election mechanism to achieve a total order of blocks. This could be a simple round-robin in PoS or mining in PoW. The leader typically proposes a block that gets voted on by the rest of the consensus nodes. Fantom's Lachesis on the other hand is leaderless. **Any node can propose a new transaction or event block.**

The DAG-based data structure is an integral part of this leaderless consensus. A DAG is a data structure that captures happens-before relationships between events. In Lachesis, nodes store a DAG of blocks (known as event blocks), each of which contains transactions. This gives a casual (partial) order between blocks, which is unlike a blockchain where all blocks are in total order.



Each node maintains a local DAG, which grows as nodes periodically exchange new blocks with each other. Since Fantom is a distributed network there is no global clock that can be relied on. The DAG establishes a partial order between blocks via lamport clocks which help encode "happened-before" relationships between blocks. That is; **each block in the DAG references a previous block implying it comes after it, and that it implicitly votes for it**. Just like in other BFT protocols, blocks that collect ⅔ of votes are confirmed.

Unlike your typical BFT protocol, Lachesis is leaderless. Each node independently applies the same Lachesis Consensus Algorithm to their local DAG to derive a consistent total order of confirmed blocks off of it, forming the exact final blockchain. Importantly, they do so w/o needing to further communicate with each other. For example, there is no need for finalized blocks to be propagated across the network. This removes the need for an extra communication overhead in the network for consensus. Lachesis therefore has a smaller communication overhead than synchronous BFT consensus mechanisms. It also saves the protocol from any leader-related bottlenecks and leader-related censorship / MEV (note that this is not a panacea for all sorts of MEV).

Closing Thoughts

Fantom network is on the verge of radical performance upgrades. Sonic, the next iteration of the Fantom network is designed to push the limits of a monolithic chain through a number of innovations including the Fantom Virtual Machine (FVM), Carmen storage solution, and an optimized transaction pool for Lachesis consensus.

Sonic has shown remarkable performance results in closed and open testnets **processing ~2k defi type tps with ~1-sec finality, all while reducing storage costs of nodes by up to 90%. The Sonic upgrade**, scheduled for spring 2024, **will not require a hard fork on the network and will retain full compatibility with the current Opera client running EVM.**