Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea Magistrale in Informatica

ZERONETHERMIND: ENGINEERING A STATELESS EXECUTION LAYER FOR ETHEREUM USING ZERO-KNOWLEDGE PROOFS

Relatore:
Chiar.ma Prof.
Marco Di Felice

Presentata da:
Manuel Arto

Correlatore:
Dott. Luca Sciullo

Sessione Marzo 2026
Anno Accademico 2024/2025

Abstract

La velocità complessiva di Ethereum è vincolata dalle capacità computazionali del suo nodo più lento. Per preservare un ambiente sicuro e decentralizzato, il protocollo impone attualmente a ciascun validatore la ri-esecuzione di ogni transazione e il mantenimento di un database di stato nell'ordine dei terabyte. Questo approccio genera un collo di bottiglia con complessità $O(N)$ rispetto al volume delle transazion e innesca la criticità architetturale nota come State Bloat, allontanando progressivamente l'hardware consumer dalla partecipazione attiva alla rete. Questa tesi presenta ZeroNethermind, un proof-of-concept che trasforma il client Execution Layer Nethermind in un validatore stateless, introducendo un bridge FFI cross-language tra C#/.NET e Rust che bypassa l'esecuzione EVM nativa attraverso la verifica di prove crittografiche generate da cinque zkVM indipendenti. Le campagne sperimentali, condotte in tempo reale sulla Mainnet Ethereum e in ambiente devnet controllato, confermano un tempo di verifica quasi costante e indipendente dalla complessità del blocco, con riduzioni significative nell'utilizzo delle risorse. Delegando il carico di esecuzione a prover specializzati e consentendo la verifica su dispositivi consumer, ZeroNethermind dimostra la fattibilità di un nuovo modello di validazione, contribuendo alle fondamenta ingegneristiche per scalare il Layer 1 senza compromettere la decentralizzazione della rete.

Indice

1 Introduzione **1**

2 Stato dell'Arte **5**
2.1 Architettura di Ethereum 5
2.2 Scalabilità: L2 vs. L1 13
2.3 Stateless clients 16
2.4 Zero Knowledge 22
2.5 Lavori Correlati 28

3 ZeroNethermind **33**
3.1 Il Paradigma "Double Zero" 33
3.2 Architettura a Tre Strati 34
3.3 Il Nuovo Flusso di Validazione 36

4 Implementazione **39**
4.1 Fase 1: The Foundation 39
4.2 Fase 2: Nethermind Integration 46
4.3 Fase 3: The ZK Validation Service 49
4.4 Profilo di Configurazione 51

5 Validazione **53**
5.1 Metodologia 54
5.2 Demo 1: Validazione sulla Mainnet 58
5.3 Demo 2: Analisi in Ambiente Controllato 61

5.4 Discussione dei Risultati 67

6 Conclusione **71**
6.1 Riepilogo 71
6.2 Limitazioni 73
6.3 Lavori Futuri 74
6.4 Considerazioni finali 76

Riferimenti bibliografici **79**

Capitolo 1

Introduzione

La promessa fondamentale delle reti blockchain decentralizzate risiede nella democratizzazione della fiducia e nella resistenza sistemica alla censura. Questa *libertà* algoritmica, tuttavia, si scontra con rigorose limitazioni fisiche e architetturali. Nell'ecosistema Ethereum, il mantenimento dell'integrità del network in un ambiente trustless richiede attualmente che ogni nodo validatore ri-esegua nativamente ogni singola transazione. Di conseguenza, il throughput complessivo della rete è vincolato dalle capacità computazionali del nodo più lento all'interno del sistema.

Questo collo di bottiglia architetturale impone un costo di ri-esecuzione pari a $O(N_{gas})$, proporzionale alla complessità aggregata delle transazioni di un blocco. Aumentare il Gas Limit per processare un volume maggiore di transazioni comporterebbe l'inevitabile esclusione dell'hardware di livello consumer. A questa problematica si sovrappone la criticità architetturale nota come State Bloat, l'espansione incontrollata del database di stato globale.

È fondamentale precisare che il principale collo di bottiglia nell'elaborazione e validazione dei blocchi su Ethereum non è di natura computazionale (CPU-bound), bensì legato all'accesso ai dati (I/O-bound). L'esecuzione di una singola transazione all'interno dell'EVM richiede

continue e frammentate operazioni di lettura e scrittura sul database di stato, il quale deve tracciare accuratamente la totalità dei saldi degli account e degli slot di memoria degli smart contract.

Attualmente, questo strato dati impone ai nodi il mantenimento locale di archivi nell'ordine dei terabyte. Con la crescita ininterrotta dello stato, i requisiti hardware si inaspriscono: il limite non è più dettato dalla capacità di archiviazione, ma dalla necessità di disporre di memorie con prestazioni elevatissime in termini di IOPS (Input/Output Operations Per Second) per sostenere il ritmo di esecuzione. Tale deriva spingerebbe la validazione verso una rete dominata da data center centralizzati, minacciando il fondamento stesso della decentralizzazione e della resistenza alla censura.

Con la transizione dalla fase architetturale di "The Merge" alla roadmap futura di "The Verge" [2], l'ecosistema Ethereum sta attivamente puntando a mitigare questa problematica. Il suo obiettivo a lungo termine è raggiungere la *statelessness*, abbassando i requisiti hardware al punto da permettere la validazione completa della chain su dispositivi a bassissimo consumo, come smartphone o smartwatch.

Storicamente, l'evoluzione della crittografia ha tracciato la roadmap per la sicurezza delle infrastrutture digitali mondiali: l'introduzione delle funzioni di hash, della crittografia a chiave asimmetrica e delle firme digitali ha posto le fondamenta per l'autenticazione e il trasferimento di valore senza intermediari. L'adozione della crittografia Zero-Knowledge (ZK) rappresenta il passo evolutivo successivo.

Le architetture blockchain tradizionali sono intrinsecamente "simmetriche": ogni nodo è costretto a duplicare lo stesso identico sforzo computazionale del nodo che ha prodotto il blocco. L'applicazione delle Zero-Knowledge Proofs (ZKPs) impone un cambio di paradigma radicale: lo sforzo per generare il blocco e la relativa prova crittografica resta oneroso, ma la successiva verifica diviene computazionalmente trascurabile. In un ecosistema basato su ZKPs, la validazione com-

putazionale di un blocco contenente diecimila transazioni richiede il medesimo tempo necessario per verificare un blocco contenente una singola transazione.

Questo principio si concretizza nel modello topologico "Beast vs. Fort" [6]. Da un lato, le infrastrutture "Beast Mode" (Provers/Builders) centralizzate e dotate di cluster hardware ad altissime prestazioni generano i blocchi e le relative prove crittografiche. Dall'altro lato, i validatori in "Fort Mode" operano su hardware di fascia consumer, rimuovendo il peso dell'esecuzione e limitandosi a certificare la validità matematica della prova e degli header dei blocchi, garantendo una sicurezza crittografica a difesa della rete.

L'obiettivo primario di questo lavoro di tesi è l'ingegnerizzazione di ZeroNethermind: un Proof-of-Concept ideato per trasformare il client Execution Layer Nethermind in un nodo "Fort Mode", finalizzato a validare i blocchi della Mainnet (L1) utilizzando ZKPs, svincolandosi totalmente dalla necessità di mantenere un database di stato locale.

Il presente elaborato apporta i seguenti contributi al panorama dei client Ethereum:

- **Implementazione del Paradigma "Double Zero":** Dimostrazione pratica della fattibilità di un *validatore stateless* (zero-state) che integra zero-knowledge per sostituire la ri-esecuzione all'interno di un client production-ready.
- **Interoperabilità Cross-Language:** Sviluppo di un bridge FFI (Foreign Function Interface) ad alte prestazioni tra l'ambiente C# (.NET) e le librerie crittografiche in Rust.
- **Validazione Live L1:** Intercettazione del Data Flow dell'Engine API per validare in tempo reale i blocchi in cima alla Mainnet demandando la generazione delle prove a infrastrutture esterne.
- **Analisi Quantitativa e Prestazionale:** Valutazione empirica dei tempi di verifica crittografica rispetto all'esecuzione nativa

EVM e analisi dell'abbattimento del resource footprint in una Ethereum devnet privata.

Il perimetro della ricerca si focalizza sull'Execution Layer (EL), sulle dinamiche di intercettazione dell'Engine API e sulla fattibilità architetturale dell'approccio stateless. Non rientra negli scopi di questo lavoro l'ingegnerizzazione dell'infrastruttura di proving (Beast Mode), né la generazione locale delle prove crittografiche. Inoltre, la formulazione teorica estesa dei circuiti aritmetici sottostanti i protocolli ZK viene trattata solo in funzione della loro applicazione pratica per la verifica.

Il documento è strutturato come segue. Il `capitolo 2` presenta lo Stato dell'Arte, analizzando l'architettura di Ethereum, la dicotomia di scalabilità tra Layer 1 e Layer 2, il paradigma dei client stateless e i fondamenti della crittografia Zero-Knowledge, concludendo con una rassegna dei lavori correlati e delle proposte di aggiornamento del protocollo. Il `capitolo 3` descrive la progettazione di ZeroNethermind: il paradigma "Double Zero", l'architettura a tre strati e il flusso di dati modificato rispetto alla validazione standard. Il `capitolo 4` documenta l'implementazione condotta attraverso tre distinte fasi incrementali. Il `capitolo 5` presenta la validazione sperimentale su due fronti: una Demo live sulla Mainnet Ethereum e un ambiente controllato basato su devnet Kurtosis per l'analisi comparativa. Il `capitolo 6` chiude l'elaborato con le conclusioni, le limitazioni del PoC e le prospettive di sviluppo futuro nel contesto della roadmap di Ethereum.

Capitolo 2

Stato dell'Arte

Il presente capitolo ha l'obiettivo di delineare il contesto tecnologico e scientifico in cui si inserisce questo lavoro di tesi, fornendo le basi necessarie per comprendere le sfide attuali e le soluzioni emergenti nell'ecosistema blockchain. Il capitolo è strutturato in quattro sezioni principali: la prima analizza l'Architettura di Ethereum, definendone i principi di base, i limiti prestazionali intrinseci e il funzionamento dei nodi; la seconda esplora la dicotomia della scalabilità (L1 vs L2); la terza analizza l'introduzione di Client Stateless all'interno del protocollo di Ethereum; la quarta sezione è dedicata alla crittografia Zero-Knowledge, il motore del cambio di paradigma da computazione a verifica; infine, l'ultima sezione passa in rassegna i lavori correlati e il panorama della ricerca corrente.

2.1 Architettura di Ethereum

Ethereum è la piattaforma blockchain *general-purpose* più adottata a livello globale per l'esecuzione di *smart contract*. Più che una semplice rete per il trasferimento di valore, Ethereum è stato concepito come un'infrastruttura su cui chiunque può costruire applicazioni decentralizzate (dApp) senza richiedere permessi (*permissionless*). I tre pilastri fondamentali su cui si basa questa architettura sono:

- **Trasparenza:** Tutto il codice e lo storico delle transazioni sono pubblici e verificabili crittograficamente in modo indipendente.
- **Resistenza alla censura:** Nessuna entità centrale, governo o azienda ha l'autorità o la capacità tecnica di bloccare una transazione valida o impedire l'accesso alla rete.
- **Controllo da parte degli utenti:** Tramite l'uso della crittografia a chiave asimmetrica, gli utenti mantengono la piena e unica sovranità sui propri fondi e sui propri dati (*self-custody*).

L'innovazione architetturale introdotta da Ethereum è stata la creazione del primo "computer globale decentralizzato". A differenza dei paradigmi informatici tradizionali centralizzati (es. servizi cloud gestiti da entità corporative), l'infrastruttura di Ethereum non è localizzata in un singolo luogo fisico. È una macchina virtuale distribuita che non può essere spenta, resettata o distrutta. L'accesso richiede unicamente una connessione a Internet, e l'infrastruttura crittografica garantisce uno spazio di indirizzamento enorme: le chiavi private derivano da uno spazio di probabilità pari a 2^{256}, da cui vengono estratti indirizzi a 160 bit, garantendo virtualmente spazio inesauribile per l'intera popolazione globale senza alcun rischio pratico di collisioni.

2.1.1 Blockchain Trilemma

Sebbene Ethereum funga da computer globale, esso è intrinsecamente più lento e costoso rispetto all'informatica centralizzata. Questa non è un'inefficienza ingegneristica casuale, ma una precisa scelta architetturale legata al "Blockchain Trilemma". Tale postulato afferma che una rete decentralizzata deve bilanciare tre proprietà — Scalabilità, Sicurezza e Decentralizzazione — potendone storicamente massimizzare solo due simultaneamente.

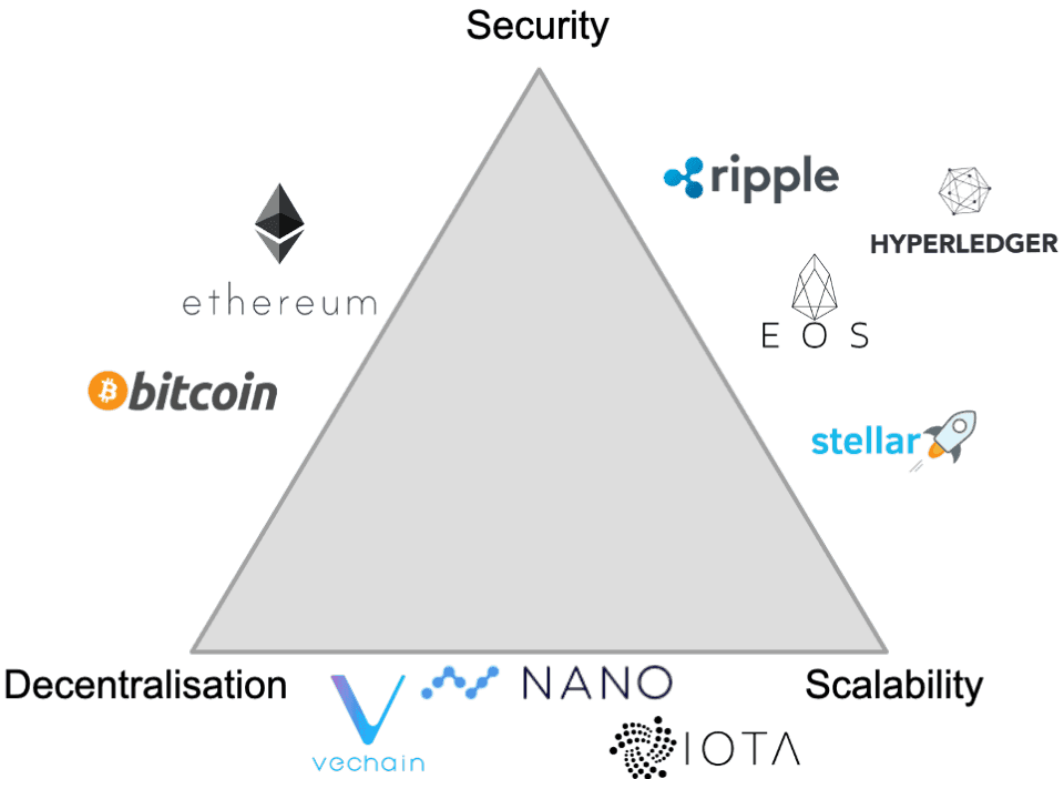

Figura 2.1: Blockchain Trilemma [21]

Ethereum ha sistematicamente privilegiato Sicurezza e Decentralizzazione. I colli di bottiglia primari sono riconducibili a due fattori:

1. **Ri-esecuzione:** Per garantire che il sistema sia *trustless*, ogni singolo nodo validatore della rete deve ri-eseguire in modo indipendente ogni singola transazione per verificarne la validità. Questo lavoro ridondante impone un costo asintotico $O(N_{gas})$ sull'esecuzione, rendendo l'intero network veloce unicamente quanto il suo "nodo più lento".
2. **Latenza di Rete e Consenso:** Ethereum è una rete P2P distribuita globalmente. Per permettere a migliaia di nodi di ricevere i dati, validare i blocchi e raggiungere un accordo probabilistico o deterministico sullo stato della rete, il protocollo impone una divisione del tempo in *Slot* fissi di 12 secondi.

2.1.2 Lo Stato in Ethereum e la Funzione di Transizione

Ethereum opera fondamentalmente come una complessa macchina a stati finiti deterministica basata su transazioni. Il concetto di "Stato" (*State*) è vitale: esso è il database globale che contiene il quadro aggiornato della rete.

La letteratura distingue tra *World State* (Stato Globale) e *Account State* (Stato dell'Account). Il *World State* mappa ogni indirizzo a 160 bit al suo rispettivo *Account State*. Un *Account State* è composto da quattro elementi: il *nonce* (contatore delle transazioni), il *balance* (saldo in ETH), il *codeHash* (l'hash del bytecode del contratto, se presente) e lo *storageRoot* (la radice dell'albero che mappa le variabili di stato interne al contratto).

La transizione da uno stato valido al successivo è governata dalla *State Transition Function* (STF) della Ethereum Virtual Machine (EVM). Come definito formalmente nello Yellow Paper[27], l'applicazione di una transazione T allo stato pre-esistente σ_t genera un nuovo stato valido σ_{t+1}:

$$\sigma_{t+1} \equiv \Upsilon(\sigma_t, T) \tag{2.1}$$

Per garantire l'integrità crittografica di questo immenso database (che oggi pesa svariate centinaia di gigabyte, superando il terabyte per i nodi *Archive*), Ethereum non utilizza un database relazionale standard, bensì un albero crittografico chiamato Merkle Patricia Trie (MPT). L'MPT funge da database autenticato composto da quattro tipologie di nodi: *Branch* (nodi di diramazione a 16 vie), *Extension* (nodi che comprimono percorsi comuni), *Leaf* (nodi foglia contenenti il valore finale) e *Empty* (nodi nulli). I dati grezzi serializzati di questo albero vengono fisicamente persistiti su disco tramite motori di database chiave-valore ad alte prestazioni come LevelDB o RocksDB. Tuttavia, il costo computazionale e di I/O (Input/Output) di questa architettura è estremamente gravoso: per ogni operazione di persistenza e aggiornamento, l'EVM deve ricalcolare crittograficamente gli hash

partendo dal nodo foglia modificato e risalendo l'intero albero fino alla radice. Man mano che l'MPT cresce in dimensione e profondità, questo meccanismo di autenticazione genera una severa amplificazione delle operazioni di lettura e scrittura su disco (*read/write amplification*), arrivando a pesare fino a 64 accessi al database per una singola operazione nel caso peggiore. Questo rende la ri-esecuzione $O(N_{gas})$ delle transazioni un problema prevalentemente *I/O-bound* (limitato dalla latenza e dall'accesso sequenziale/casuale allo storage) e non puramente *CPU-bound*. L'hash della radice del nuovo albero globale (*State Root*) viene infine calcolato e inserito nell'intestazione di ogni nuovo blocco per sigillarne la validità.

2.1.3 Gas Limit and Bottlenecks

Il *Gas* è l'unità di misura dello sforzo computazionale richiesto per eseguire un'operazione nell'EVM. Poiché Ethereum impone la ri-esecuzione su tutti i nodi, la rete deve autoproteggersi dal problema dell'arresto (Halting Problem) e dal consumo illimitato di risorse. Questo avviene tramite il *Block Gas Limit*, un tetto massimo assoluto alla quantità di calcoli effettuabili in un singolo blocco (attualmente intorno ai 36 milioni di gas, in lenta evoluzione e oggetto di intensi dibattiti per aumenti fino a 60 milioni).

Aumentare "semplicemente" il Gas Limit comporterebbe un incremento drastico del carico computazionale, della larghezza di banda e dei requisiti di RAM e storage (SSD NVMe). L'aggravarsi dello *State Bloat* — l'accumulo permanente di dati di stato — e l'aumento dell'onere di validazione spingerebbero inevitabilmente l'hardware di livello consumer fuori dalla rete, delegando il consenso a data center professionali e distruggendo la neutralità credibile dell'infrastruttura.

2.1.4 Execution Layer (EL) vs Consensus Layer (CL)

A seguito dell'aggiornamento "The Merge" (transizione al meccanismo Proof-of-Stake), l'architettura dei client Ethereum ha subìto una profonda refattorizzazione, separando le responsabilità in due moduli software eseguiti in tandem:

- **Execution Layer (EL):** Questo strato (es. Nethermind, Geth) è il "motore" computazionale. L'EL mantiene l'MPT del *World State*, gestisce la *mempool* locale delle transazioni non confermate ed esegue nativamente l'EVM per applicare la funzione di transizione di stato.
- **Consensus Layer (CL):** Noto anche come *Beacon Node* (es. Lighthouse, Prysm), questo strato è agnostico rispetto all'esecuzione degli *smart contract*. Il suo compito è governare il protocollo di consenso Proof-of-Stake: coordina i turni dei validatori, propaga i blocchi tramite rete gossip, aggrega le firme (*attestations*) e applica la regola di scelta del fork (*Fork Choice Rule*) per stabilire la catena canonica

Questi due livelli comunicano in modo bidirezionale, sicuro e standardizzato tramite una suite di interfacce JSON-RPC nota come Engine API.

2.1.5 Nethermind Client

Nethermind è uno dei principali client dell'Execution Layer per Ethereum, sviluppato interamente in C#/.NET. Il progetto è entrato in produzione nel 2019 e rappresenta, insieme a Geth (Go Ethereum) e Reth (Rust Ethereum), uno dei tre client EL più diffusi nella rete mainnet, contribuendo attivamente alla diversità del protocollo.

I requisiti hardware[20] per eseguire un nodo Nethermind in piena validazione sono significativi e in costante crescita:

- **CPU:** 4 o più core.

- **RAM:** Almeno 16 GB (32 GB consigliati per prestazioni ottimali).
- **Storage:** Oltre 2 TB di NVMe SSD ad alte prestazioni (min 10.000 IOPS).

Nethermind impiega RocksDB[19], un motore di database chiave-valore ad alte prestazioni, per la persistenza su disco. I dati sono organizzati in circa sei categorie (State, Receipts, Blocks, Headers, Code, Metadata), con il solo database di stato che supera i 150 GB. L'intera base dati, comprensiva dello storico dei blocchi e delle ricevute, raggiunge e supera 1 TB. Questa massiccia impronta di storage, unita alla necessità di IOPS elevati, rappresenta la barriera primaria che rischia di escludere l'hardware di livello consumer dalla partecipazione attiva alla rete in futuro.

2.1.6 Flusso di Validazione di un Blocco e l'Engine API

Il processo di validazione di un nuovo blocco rappresenta il momento in cui l'onere computazionale ricade fisicamente sul nodo. Le dinamiche di comunicazione tra EL e CL sono il fulcro su cui opera ZeroNethermind. Il ciclo di vita segue questi step architetturali:

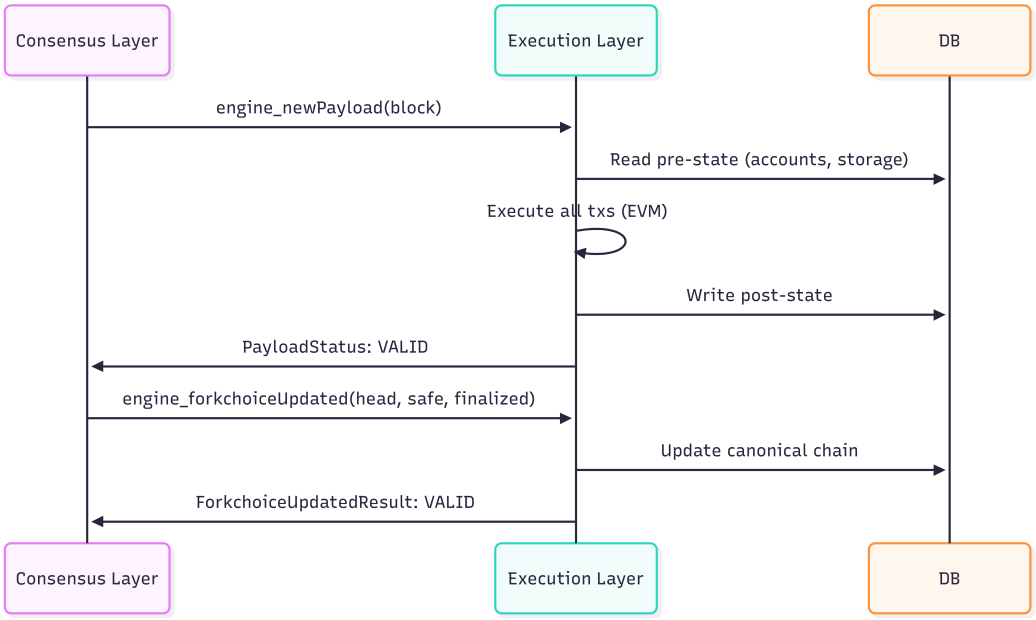

Figura 2.2: Flusso di Validazione Standard

1. **Mempool e Costruzione:** Le transazioni inviate dagli utenti popolano la *mempool* dell'Execution Layer. Il nodo selezionato come *Proposer* per lo slot corrente aggrega le transazioni in un *ExecutionPayload* e lo incapsula in un *Beacon Block*.
2. **Propagazione (CL):** Il blocco Beacon viene diffuso attraverso la rete P2P del Consensus Layer.
3. **Ricezione e Interrogazione (Engine API):** Quando un nodo remoto riceve il blocco, il suo CL estrae il payload di esecuzione e lo invia al proprio EL locale richiamando il metodo RPC `engine_newPayload`.
4. **Ri-esecuzione Nativa (EL):** L'EL deve verificare rigorosamente il blocco. Carica lo stato antecedente dal proprio disco, controlla la validità del base fee, la disponibilità di gas, i nonce, i saldi iniziali ed esegue sequenzialmente ogni transazione all'interno dell'EVM. Infine, ricalcola la radice dell'albero MPT.

5. **Esito della Validazione:** Se il ricalcolo dello *State Root* coincide con quello dichiarato dal Proposer, l'EL risponde al CL con lo status `VALID`.
6. **Aggiornamento Fork Choice:** Avuta conferma della validità esecutiva, il CL invia il comando `engine_forkchoiceUpdated`, istruendo l'EL a designare tale blocco come nuova *head* della catena canonica.

2.2 Scalabilità: L2 vs. L1

Storicamente, in risposta al severo limite intrinseco di transazioni al secondo dettato dal modello di ri-esecuzione N-of-N, la comunità di ricerca ha delegato il carico computazionale a reti secondarie. Questo approccio ha dato vita alla cosiddetta roadmap "Rollup-Centric". In questo paradigma, il Layer 1 (L1) sacrifica la scalabilità dell'esecuzione per mantenere un ruolo esclusivo come livello di massima sicurezza e garanzia della disponibilità dei dati. L'elaborazione massiva delle transazioni viene invece spostata *off-chain* sui Layer 2 (L2), noti come *Rollup*.

Esistono due approcci principali per garantire che l'esecuzione avvenuta su L2 sia valida quando viene finalizzata su L1:

- **Optimistic Rollups:** Assumono per default che tutte le transazioni processate siano corrette. La sicurezza è garantita da un meccanismo di risoluzione delle dispute (*Fraud Proofs*), che introduce tuttavia una latenza significativa (tipicamente di 7 giorni) prima che le transazioni raggiungano la finalità inoppugnabile sul L1.
- **ZK-Rollups:** Sfruttano le *Validity Proofs* (prove crittografiche come SNARK o STARK) generate *off-chain* per certificare l'integrità matematica di ogni batch di transazioni. Questo approccio garantisce la correttezza a priori e consente un settlement sul

L1 vincolato unicamente dal tempo di generazione della prova crittografica

2.2.1 Criticità degli L2 attuali

Nonostante il proliferare delle soluzioni L2, la loro architettura corrente presenta criticità sistemiche che contraddicono l'etica decentralizzata del L1. L'assunto di ereditare passivamente la sicurezza di Ethereum nasconde spesso l'impiego di *Training Wheels*: meccanismi di salvaguardia centralizzati che conferiscono a pochi attori il potere di alterare il protocollo.

Analizzando il framework di classificazione *L2Beat* [18], si osserva che la quasi totalità degli L2 attuali si trova allo "Stage 0" o "Stage 1" di maturità decentralizzata. Questi network si affidano a *Sequencer* operati da singole entità corporative e a *Security Council* governati da chiavi *multisig* capaci di sovrascrivere l'esecuzione o imporre aggiornamenti forzati degli smart contract di convalida.

Questa situazione ha portato a una profonda ricalibrazione teorica: un L2 che non offre solide garanzie crittografiche *trustless* non sta realmente scalando Ethereum, ma sta operando come un L1 indipendente mascherato da L2 tramite un bridge centralizzato. Per formalizzare questo concetto, è stato introdotto l'*Ethereum Settlement Score* (ESS), una metrica che quantifica il grado in cui un L2 utilizza in modo inoppugnabile il livello di base per finalizzare le transazioni.

A livello di ecosistema, questa proliferazione orizzontale di Rollup imperfetti ha inoltre frammentato drasticamente la liquidità e distrutto la *Synchronous Composability* — la capacità degli smart contract di interagire atomicamente tra loro all'interno del medesimo blocco, una proprietà fondamentale che aveva alimentato il successo della Finanza Decentralizzata (DeFi) sul Layer 1.

2.2.2 Il Cambio di Paradigma: Scalare L1 tramite ZK

Il superamento delle vulnerabilità degli L2 richiede un ritorno al potenziamento del Layer 1, abbandonando il paradigma in cui "tutti calcolano tutto" in favore di un modello in cui "uno calcola, tutti verificano". Questo rappresenta il salto architetturale dal calcolo ridondante (N-of-N) alla verifica crittografica (1-of-N).

Questo principio è stato formalizzato dal ricercatore della Ethereum Foundation Justin Drake[6] attraverso la dicotomia architetturale nota come Beast Mode vs. Fort Mode:

- **Beast Mode:** Prevede l'esternalizzazione dell'esecuzione massiva e della generazione delle prove crittografiche a *Provers* e *Builders* operanti su infrastrutture di livello data-center. L'obiettivo prestazionale è spingere il limite di elaborazione del L1 verso la frontiera del *GigaGas/s* (fino a 10.000 TPS), superando i limiti dei server general-purpose.
- **Fort Mode:** Rappresenta il baluardo della decentralizzazione. I validatori cessano di eseguire l'EVM nativamente per diventare validatori "ultra-leggeri". Grazie alla asimmetria computazionale fornita dalla crittografia ZK, questi nodi possono operare su hardware di consumo (laptop, smartphone o persino smartwatch) limitandosi a certificare matematicamente le prove generate dalle "Beast".

Caratteristica	**Beast Mode**	**Fort Mode** (ZeroNethermind / Validators)
Ruolo	Esegue computazioni pesanti e genera prove crittografiche.	Mette in sicurezza la chain verificando le prove
Hardware	Specialized Hardware.	Laptop consumer, tablet, smartphone
Storage (Stato L1)	Terabyte	0 GB (MemDB)
Complessità Computazionale	$O(N_{gas})$ - proporzionale al volume delle transazioni	$O(1)$
Obiettivo di Rete	Scalare Ethereum a 10k+ TPS (GigaGas).	Resistenza incompromissibile alla censura e sicurezza

Tabella 2.1: Confronto architetturale tra il paradigma Beast Mode e Fort Mode

La transizione verso questa architettura è prevista in due fasi: una prima fase di *optional proofs* [10], in cui i nodi possono scegliere volontariamente di verificare i blocchi tramite prove ZK anziché ri-eseguirli, seguita da una fase di *mandatory proofs*, in cui la verifica crittografica diviene il meccanismo predefinito e la ri-esecuzione un'opzione per ruoli specializzati (RPC, block building). ZeroNethermind anticipa la prima fase, dimostrando che un client EL può già oggi operare in modalità proof-only.

2.3 Stateless clients

Una delle proprietà più potenti di una rete blockchain è la capacità, per chiunque, di eseguire un nodo sul proprio computer e verificare in modo indipendente la correttezza della chain. Anche se la quasi totalità dei nodi in consenso dovesse accordarsi su regole malevole, un nodo in piena validazione rifiuterebbe categoricamente la chain compromessa. Tuttavia, affinché questa dinamica di sicurezza rimanga valida, l'esecuzione di un nodo deve rimanere accessibile a una massa critica di utenti.

Oggi, eseguire un nodo su hardware consumer è possibile, ma sempre più complesso a causa della mole di dati da archiviare. La fase della roadmap di Ethereum denominata "The Verge" si pone l'obiettivo di rivoluzionare questo aspetto, rendendo la validazione talmente leggera da poter essere eseguita di default su dispositivi mobili, wallet browser o perfino smartwatch.

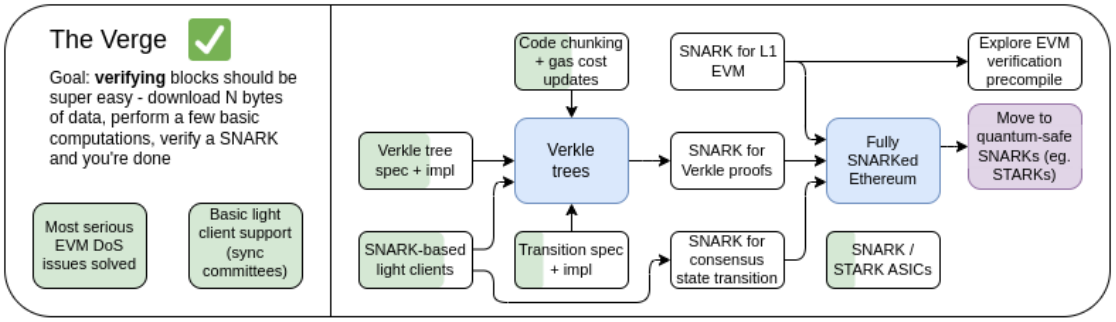

Figura 2.3: The Verge, 2023 roadmap [2]

Il problema fondamentale che The Verge cerca di risolvere è la crescita incontrollata dei dati di stato. Attualmente, un client Ethereum deve archiviare centinaia di gigabyte di dati per poter validare i blocchi, con un incremento di circa 30 GB di dati grezzi all'anno, a cui si aggiungono i metadati per aggiornare efficientemente il database. Questo attrito tecnologico allunga i tempi di sincronizzazione a giorni interi e disincentiva la partecipazione dei nodi indipendenti.

Originariamente, The Verge mirava a sostituire i Merkle Patricia Trie con i Verkle Trees, strutture dati che consentono prove molto più compatte. Oggi, la visione si è ampliata per includere la verifica dell'intera esecuzione di Ethereum tramite prove crittografiche SNARK, puntando a due obiettivi chiave:

- **Client Stateless:** I client in piena validazione e i nodi di staking non dovranno richiedere più di una manciata di gigabyte di archiviazione.

- **Verifica su dispositivi ultra-leggeri:** Nel lungo termine, la convalida della catena consisterà unicamente nel download di una minima porzione di dati e nella verifica matematica di una SNARK.

2.3.1 Stateless Ethereum

"Stateless Ethereum" rappresenta un aggiornamento radicale del protocollo in cui i blocchi diventano unità di esecuzione self-contained. Il nodo non è più obbligato a scaricare l'intero stato della rete, poiché tutte le informazioni necessarie per l'esecuzione e la validazione vengono impacchettate direttamente all'interno del blocco stesso tramite prove crittografiche (i cosiddetti *witnesses*).

I benefici principali di questa architettura si diramano su molteplici fronti:

- **Scalabilità:** Rimuovendo il collo di bottiglia dell'I/O legato all'accesso al database su disco, i validatori possono processare molte più transazioni, permettendo un innalzamento sicuro del Gas Limit e quindi un TPS (Transaction Per Second) più elevato.
- **Decentralizzazione:** L'abbattimento dei requisiti hardware permette l'ingresso di nuovi attori nella sicurezza della rete, aprendo la strada a client ultra-leggeri *trustless* e a dinamiche flessibili come il *rainbow staking* o la creazione di pool di validazione privati.
- **Sincronizzazione Immediata:** Non necessitando del database storico, un nodo necessita soltanto del blocco corrente per unirsi attivamente al network.

2.3.2 Relazione tra Verifica dei Blocchi e Stato

Per comprendere appieno la relazione tra lo stato di Ethereum e la verifica dei blocchi, si consideri uno scenario in cui si debba validare

un blocco contenente un'unica transazione: il trasferimento di 10 ETH dall'account A all'account B. Omettendo dettagli secondari, la logica di base impone di verificare che l'account A possieda un saldo sufficiente per coprire sia l'importo trasferito che le commissioni di rete (*gas fees*). In caso contrario, la transazione deve fallire.

È fondamentale notare che, al momento della ricezione di un blocco, un client Execution Layer non può prevedere a priori quali account saranno coinvolti. Di conseguenza, in un'architettura tradizionale, il client è costretto a mantenere in memoria le informazioni relative a tutti gli account Ethereum esistenti. L'assenza di tali dati renderebbe impossibile la verifica del blocco per un client non equipaggiato, penalizzando in particolar modo le architetture *stateless*. Le transazioni più complesse, come quelle che invocano *smart contract*, seguono la medesima logica: i client devono disporre del bytecode dei contratti e del relativo stato di archiviazione (*storage state*), poiché i blocchi possono teoricamente contenere transazioni che interagiscono con qualsiasi contratto distribuito sulla rete.

L'intuizione principale alla base del paradigma *stateless* è che la porzione di stato effettivamente necessaria per verificare un singolo blocco è di ordini di grandezza inferiore rispetto all'intero *World State*. Esistono porzioni dello stato di Ethereum che non vengono interrogate da anni, ma che devono comunque essere archiviate per l'eventualità che una nuova transazione possa richiederne l'accesso. Questa problematica fa emergere tangenzialmente la necessità di implementare meccanismi di *State Expiry* (scadenza dello stato) a livello di protocollo.

2.3.3 Validatori Stateless

Affinché un nodo possa validare tradizionalmente un blocco, deve possedere lo stato antecedente (pre-state). In un modello fully-SNARKified, l'Execution Layer Client non deve eseguire alcun codice EVM: deve

semplicemente validare una prova crittografica che attesta la corretta transizione dal `pre_state_root` al `post_state_root` applicando l'hash del blocco corrente.

Questa dinamica abilita la creazione dei **Validatori Stateless**. Poiché non dispongono del database completo, questi nodi non possono operare come *Block Builders*, un compito che richiede l'accesso completo allo stato per simulare transazioni ed estrarre valore. Tuttavia, grazie a meccanismi già esistenti come la *Proposer-Builder Separation* (PBS) o MEV-boost, un client stateless può delegare la costruzione a terze parti, ricevere il blocco candidato con la relativa prova, verificarne l'integrità in tempo costante e proporlo alla rete.

2.3.4 Architettura del Client Stateless

Il vantaggio ingegneristico di un client stateless risiede nella sua estrema semplicità architetturale rispetto a un full node. Passando al paradigma di pura verifica, interi moduli del client diventano obsoleti o superflui.

Osservando la struttura di un nodo stateless, notiamo una drastica riduzione delle componenti attive:

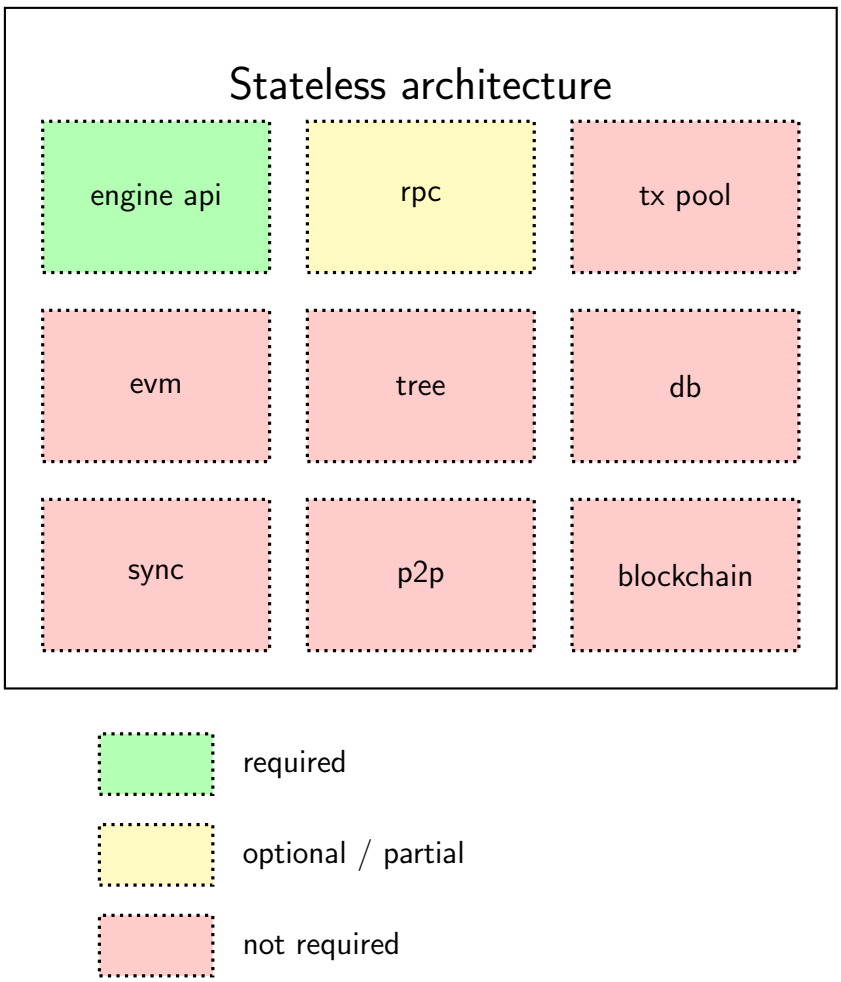

Figura 2.4: Stateless Client Architecture

- **Componenti eliminate (Not Required):** Il client viene snellito rimuovendo l'EVM (non essendovi esecuzione nativa), la Mempool delle transazioni (*Tx Pool*), le logiche di aggiornamento dell'albero di stato (*Tree*), il motore di persistenza su disco (*DB*), la sincronizzazione storica (*Sync*) e l'infrastruttura di propagazione Gossip (*P2P* e *Blockchain historical data*).
- **Componenti essenziali (Required):** L'interfaccia di comunicazione *Engine API* rimane il cuore pulsante per interagire con il Consensus Layer.

- **Componenti opzionali:** Il servizio *RPC* può essere mantenuto per interrogazioni limitate, sebbene non possa rispondere a query dipendenti dallo stato globale.

Per l'integrazione di questi client, stanno emergendo due pattern principali:

1. **Separate Clients:** Il client stateless (CL+EL) si connette a peer *stateful* tramite un sotto-protocollo di rete. L'esecuzione e la verifica sono disaccoppiate ma rimangono interattive.
2. **Embedded Clients:** Un singolo client combina la logica di consenso (CL) con un Verificatore EL leggero incorporato (es. una zkVM o un esecutore stateless). Questo client riceve blocchi e prove tramite la rete gossip e li verifica interamente in locale, un modello che rispecchia l'architettura implementata in questa tesi.

2.4 Zero Knowledge

L'evoluzione della crittografia informatica può essere suddivisa in due generazioni distinte. La "Prima Generazione" era caratterizzata da algoritmi *special-purpose*, realizzati artigianalmente per assolvere compiti altamente specifici, come le funzioni di hash per l'integrità o le firme digitali per l'autenticazione delle transazioni. L'avvento della "Seconda Generazione" segna invece il passaggio alla crittografia *general-purpose*. Introdotta formalmente alla fine degli anni Ottanta dai crittografi Goldwasser, Micali e Rackoff, la crittografia Zero-Knowledge[26] (ZK) ne rappresenta un'espressione particolarmente rilevante, conferendo l'abilità inedita di formulare asserti crittografici su computazioni del tutto arbitrarie e Turing-complete. Una dimostrazione a conoscenza zero (Zero-Knowledge Proof, ZKP) permette a un'entità di provare matematicamente la veridicità di un'affermazione senza rivelare alcuna informazione sottostante o dato segreto. Questa tecnologia detiene il potere di trasferire il modello di fiducia dal-

le istituzioni centralizzate o dall'hardware specializzato direttamente all'infallibilità della matematica pura.

A livello applicativo, i protocolli ZK abilitano una vasta gamma di casi d'uso trasversali. In ambito di identità digitale (ZK-ID e ZK-KYC), permettono a un utente di dimostrare il superamento di controlli anagrafici (ad esempio, l'essere maggiorenne) o il possesso di fondi sufficienti per una transazione, senza esporre la propria data di nascita o il saldo bancario reale. Nel settore finanziario istituzionale, le ZKPs garantiscono la conformità alle normative antiriciclaggio (AML) e alle sanzioni, mantenendo contemporaneamente private le posizioni dei clienti e le strategie di trading. Nel panorama del calcolo distribuito in cloud, consentono inoltre di delegare carichi di lavoro computazionalmente onerosi a server non fidati, i quali restituiscono il risultato accompagnato da un certificato matematico di corretta esecuzione verificabile in tempo reale.

È proprio su quest'ultimo aspetto che si basa la transizione di Ethereum verso la *statelessness* e la scalabilità massiva, attraverso l'adozione di prove crittografiche succinte e non interattive (SNARG/-SNARK). Il principio matematico fondante di questi protocolli in ottica di scalabilità risiede nell'introduzione di una netta asimmetria computazionale tra le entità partecipanti: da un lato figura il *Prover*, una macchina dotata di altissima capacità di calcolo incaricata di eseguire la computazione e generare la prova crittografica; dall'altro opera il *Verifier*, un nodo con risorse hardware limitate il cui unico onere è controllare rigorosamente la validità della prova ricevuta.

In ambito blockchain, e in particolare nell'ecosistema Ethereum, è tuttavia doverosa una precisazione accademica formale: l'utilizzo del termine "Zero-Knowledge" (ZK) rappresenta spesso un *misnomer* (un'inesattezza terminologica). La proprietà fondamentale che le cosiddette "zkEVM" sfruttano per abbattere il collo di bottiglia dell'esecuzione non è la *privacy* (ovvero l'occultamento dei dati transazionali), bensì

la succintezza (*succinctness*). Di conseguenza, a livello ingegneristico, architetture come ZeroNethermind andrebbero più correttamente definite come *Succinct EVMs* o *SNARG-EVMs*.

2.4.1 L'Interfaccia di una zkVM

Dal punto di vista dell'integrazione di sistema, una zkVM opera come un costrutto crittografico che espone un'interfaccia standardizzata definita da tre funzioni principali:

- `Prepare(f)` $\rightarrow$ `vk`: Durante la fase di setup, l'algoritmo riceve in input la descrizione formale del programma `f` (come la funzione di transizione di stato della EVM) e genera una chiave di verifica pubblica (*verification key*, `vk`) di dimensioni estremamente ridotte.
- `Prove(f, x)` $\rightarrow$ `(y, proof)`: Funzione eseguita dal *Prover*. Essa calcola l'output `y` a partire dallo stato di input `x` e produce contestualmente la prova crittografica (`proof`) che attesta la corretta esecuzione di `f(x)`.
- `Verify(vk, x, y, proof)` $\rightarrow$ `0, 1`: Funzione eseguita dal *Verifier*. Presa in carico la chiave di verifica, gli input/output pubblici e la prova, l'algoritmo restituisce un valore booleano che certifica o rigetta la validità della transizione in un tempo sublineare.

2.4.2 Le Tre Proprietà Fondamentali

Affinché la rete possa delegare in sicurezza l'esecuzione *trustless*, il sistema di proving sottostante deve garantire tre proprietà matematiche inderogabili:

1. **Completeness (Completezza):** Se l'affermazione computazionale è vera e il *Prover* si comporta in modo onesto, il *Verifier* accetterà sempre la prova. Questa proprietà è cruciale per pre-

servare la *liveness* della blockchain, evitando falsi negativi che bloccherebbero il progresso della catena.

2. **Soundness (Solidità/Robustezza):** Se l'affermazione è falsa, nessun avversario computazionalmente limitato può generare una prova valida, se non con probabilità trascurabile. Nelle zkVM, si fa ricorso alla *Knowledge Soundness*, la quale impone che il *Prover* non solo calcoli il risultato, ma conosca effettivamente l'intera traccia dell'esecuzione (*witness*) per poter costruire la prova.
3. **Succinctness (Succintezza):** Rappresenta la chiave di volta per risolvere il trilemma della scalabilità. La dimensione della prova e, soprattutto, il tempo di verifica computazionale risultano sublineari rispetto al tempo necessario per la ri-esecuzione nativa del programma, assestandosi su tempi di risoluzione costanti ($O(1)$) o polilogaritmici.

2.4.3 Problemi Architetturali Risolti dalle zkEVM

La scalabilità di qualsiasi architettura blockchain si scontra universalmente con tre vincoli fondamentali: *Compute*, *I/O* e *Bandwidth*. L'introduzione delle zkEVM, progettata specificamente per dimostrare l'esecuzione corretta della state transition function di Ethereum, interviene per mitigare congiuntamente queste limitazioni architetturali. In primo luogo, il vincolo del calcolo viene risolto direttamente dalla natura stessa delle prove ZK, che comprimono l'onere computazionale in una verifica crittografica pressoché istantanea. In secondo luogo, il vincolo dell'I/O viene risolto indirettamente: non dovendo ri-eseguire le transazioni nell'EVM, i nodi sono sollevati dall'obbligo di caricare e aggiornare costantemente l'enorme database di stato locale, abilitando di fatto operazioni parzialmente o totalmente *stateless*. Infine, il problema della larghezza di banda viene affrontato accoppiando il paradigma ZK con meccanismi di Data Availability Sampling (DAS), permettendo alla rete di attestare la disponibilità dei dati e dei bloc-

chi compressi senza forzare i nodi al download integrale di tutte le transazioni storiche.

Nel protocollo attuale, il throughput è strettamente vincolato dal *Block Gas Limit*. Un suo incremento richiederebbe ai validatori hardware sempre più costoso per sostenere la ri-esecuzione completa, accentrando il consenso. Nelle zkEVM, invece, un singolo *builder* esegue l'intero blocco e un *prover* ne genera la prova crittografica. I validatori si limitano al *proof checking*, un'operazione asintoticamente più economica. Questo paradigma 1-of-N permette di innalzare il limite di gas in totale sicurezza: espandendo la capacità dei blocchi si spalma l'alta domanda su una maggiore disponibilità di spazio, stabilizzando e abbattendo i prezzi delle commissioni.

Parallelamente, delegare il carico computazionale ai *prover* previene i ritardi nella *finality*. Poiché la verifica crittografica richiede un tempo costante e indipendente dalla complessità del blocco, i validatori non rischiano più di perdere gli slot a causa di ri-esecuzioni troppo lente. La barriera hardware viene così abbattuta radicalmente, permettendo persino a dispositivi modesti (come un Raspberry Pi) di partecipare alla messa in sicurezza della rete e preservandone l'assoluta diversità.

Separando nettamente la fase di esecuzione massiva da quella di validazione, le zkEVM trasformano radicalmente il modello di scalabilità di Ethereum verso un'architettura unificata. Questa dicotomia genera un mercato dell'esecuzione altamente competitivo che incentiva l'ottimizzazione dell'hardware specializzato per i *prover*, garantendo parallelamente una massiccia diversità dei validatori tramite l'empowerment di nodi leggeri. Il risultato finale è un throughput elastico e sicuro, che consente alla capacità del Layer 1 di crescere senza mai compromettere la decentralizzazione e l'accessibilità della rete.

2.4.4 Ethproofs.org

Ethproofs.org[12] è un *Block Proof Explorer* per Ethereum: una piattaforma pubblica che aggrega prove crittografiche ZK generate da molteplici team di proving indipendenti per i blocchi della Mainnet. Lanciato da parte dell'Ethereum Foundation, ethproofs.org svolge un ruolo analogo a quello di un block explorer tradizionale, ma specializzato nel tracciamento delle ZK proofs.

La piattaforma oltre a presentare un sito web interattivo, espone un'API REST pubblica che consente di interrogare i metadati delle prove disponibili per ciascun blocco e di scaricare sia le prove binarie sia le relative chiavi di verifica (*Verification Keys*, VK).

L'Ecosistema dei Prover

Una delle caratteristiche più significative di ethproofs.org è la sua natura di aggregatore multi-prover. La piattaforma raccoglie prove generate da diversi cluster di proving, ciascuno basato su una zkVM (*Zero-Knowledge Virtual Machine*) distinta. La tabella 2.2 riassume i principali prover attualmente operativi sulla piattaforma.

Prover	**Famiglia zkVM**	**Sistema di Prova**
Zisk	Polygon Hermez	pil2-proofman
OpenVM	Axiom	STARK (verify-stark)
Pico	Pico Prism	KoalaBear Poseidon2
SP1 Hypercube	Succinct Labs	Compressed STARK
Airbender	Matter Labs (zkSync)	Mersenne31 + Unified Recursion

Tabella 2.2: Principali prover operativi su ethproofs.org.

Questa client diversity è intenzionale. Molteplici team competono per generare prove valide nel minor tempo possibile, prefigurando la nascita di un vero e proprio *Prover Market*[16].

2.5 Lavori Correlati

L'architettura e i principi di design alla base del progetto presentano radici in un filone di ricerca ampio e in rapida evoluzione, che spazia dalla teoria dei sistemi distribuiti fino alle più recenti proposte di aggiornamento del protocollo Ethereum (EIP). Questa sezione analizza la letteratura e i progetti fondamentali che fanno da preludio all'implementazione di un nodo validatore Layer 1 (L1) basato su zkEVM, inserendo il prototipo *ZeroNethermind* nel contesto della roadmap ufficiale.

2.5.1 Il Trilemma della Blockchain e i Client Leggeri

Storicamente, il design dei registri distribuiti è stato dominato dal concetto del *Blockchain Trilemma*, un'euristica che postula l'impossibilità di massimizzare simultaneamente scalabilità, sicurezza e decentralizzazione nelle architetture a validazione simmetrica[23]. Tuttavia, la letteratura accademica più recente si è concentrata sul superamento pratico di queste limitazioni attraverso l'adozione di prove crittografiche a conoscenza zero (ZKP)[3]. L'introduzione di protocolli basati su zk-SNARK altera radicalmente le regole d'ingaggio del trilemma introducendo una forte asimmetria computazionale: in reti come Ethereum, la sicurezza può essere garantita non più dall'onerosa ri-esecuzione ridondante di tutti i nodi, ma dalla snella verifica matematica di prove crittografiche. Di conseguenza, la scalabilità non è più un compromesso obbligato contro la decentralizzazione, ma diviene un risultato ingegneristico raggiungibile disaccoppiando la complessa produzione delle prove dalla loro immediata verifica.

Questa necessità di disaccoppiamento richiama il concetto storico di *Light Client*. Come sistematizzato nel documento *SoK: Blockchain Light Clients*[4], i client leggeri permettono di interagire con la blockchain senza memorizzarne l'intera cronologia. Mentre i light client tradizionali si affidano ad assunzioni di fiducia, la ricerca moderna si

sta orientando sull'utilizzo di accumulatori crittografici e zk-SNARKs per permettere a dispositivi con risorse limitate di verificare matematicamente l'integrità del sistema in modalità pienamente *trustless*.

2.5.2 The Verge e la "Statelessness" di Ethereum

Il superamento dei limiti computazionali dei nodi è il nucleo della fase di sviluppo della roadmap di Ethereum nota come *The Verge*[2]. L'obiettivo primario è democratizzare la validazione, rendendola così efficiente da poter essere eseguita su dispositivi di largo consumo (smartphone o smartwatch).

Attualmente, un nodo Ethereum richiede la memorizzazione di un database di stato in continua crescita, creando un collo di bottiglia I/O che innalza le barriere all'ingresso e spinge verso la centralizzazione. Il paradigma *Stateless Ethereum*[24] risolve questo problema trasformando i blocchi in unità di esecuzione auto-contenute: anziché interrogare un database locale, il nodo stateless riceve un blocco accompagnato da un *witness* (testimone) e da una prova crittografica che ne attesta la validità.

A supporto di questa profonda transizione strutturale, è stata proposta l'EIP-4762 (*Statelessness gas cost changes*)[8]. Questa proposta introduce una rimodulazione dei costi del gas per riflettere accuratamente l'onere crittografico della generazione dei witness, disincentivando operazioni di stato che renderebbero le prove crittografiche troppo pesanti o complesse da generare.

2.5.3 Il Panorama dei Layer 2 e l'Analisi dei Rischi

Fino ad oggi, l'onere della scalabilità è stato delegato ai Layer 2 (L2) tramite la *rollup-centric roadmap*. Piattaforme analitiche come L2BEAT[18] svolgono un ruolo cruciale nel valutare i compromessi di sicurezza di queste architetture, dimostrando come i rollup attuali si collochino su uno "spettro di fiducia". Molte soluzioni operano anco-

ra in *Stage 1*, basandosi su sequencer centralizzati, bridge controllati da multisig e *Security Councils* autorizzati a forzare l'aggiornamento dello stato.

Per mitigare questi vettori di centralizzazione e i rischi derivanti da un ecosistema frammentato, la ricerca sta virando verso l'integrazione nativa delle prove ZK nel Layer 1 (*Enshrined zkEVM*). Questa mossa mira a unificare le garanzie di sicurezza dell'ecosistema, dimostrando che la tecnologia zkVM, maturata sui Layer 2, è pronta per il livello base del protocollo.

2.5.4 Lean Ethereum: Il Cambio di Paradigma

Il cambio di paradigma di Lean Ethereuem introdotto da Justin Drake, che passa dal modello *"Trust but Verify"* al modello *"Verify the Math"*, è delineato nella L1-zkEVM Roadmap 2026[14]. Tale documento definisce il flusso end-to-end che parte da un *Execution Client* (che genera l'*ExecutionWitness*), passa per un programma ospite zkVM e un Prover, per finire al nodo validatore che ne asserisce la validità.

In questa direzione, l'EIP-8025 (*Optional Execution Proofs*)[10] rappresenta un passo fondamentale: introduce una modifica opzionale a livello di consenso che permette ai nodi Beacon di verificare la validità dei payload di esecuzione tramite prove ZK, senza dover interrogare un client di esecuzione locale.

2.5.5 L'Ecosistema zkVM: Implementazioni e Competizione

L'implementazione pratica del *real-time proving* poggia su macchine virtuali a conoscenza zero (zkVM) capaci di interpretare set di istruzioni, monitorate attivamente da portali come *zkevm.fyi*[7]. Tra i progetti di punta figurano:

- **SP1**[25] (Succinct Labs): Una zkVM *general-purpose* ad alte prestazioni basata sull'architettura RISC-V, che permette di generare prove per programmi compilati in Rust (o tramite LLVM).
- **ZisK**[1] (Polygon): Un'architettura zkVM altamente specializzata, progettata specificamente per l'esecuzione rapida e la generazione di prove per transazioni compatibili con l'EVM.

La coesistenza e la competizione tra questi *prover* sono vitali per garantire la *client diversity* anche a livello crittografico, scongiurando singoli punti di fallimento (es. bug in un circuito ZK) richiedendo *multi-proofs* per l'accettazione di un blocco.

2.5.6 Prototipi e L1 Strawmap

La validità di questo approccio è stata dimostrata sul campo durante conferenze di settore come il Devconnect. In tale occasione, è stata presentata zkLighthouse[5], una versione modificata del client di consenso Lighthouse. Operando su hardware di largo consumo, questo nodo ha bypassato completamente l'uso di un *Execution Client* classico (come Geth o Nethermind), validando i blocchi della Mainnet in tempo reale. Il sistema si limitava ad ascoltare un canale P2P dedicato, verificando in millisecondi le prove prodotte da cluster di prover esterni.

L'intero sforzo architetturale si condensa infine nella L1 Strawmap (*strawmap.org*)[13], una bozza di roadmap mantenuta dal team della Ethereum Foundation. Questa mappa unifica gli aggiornamenti in tre filoni (Consenso, Dati, Esecuzione) e fissa le pietre miliari del protocollo, tra cui la transizione a un *Gigagas L1* (10.000 TPS) resa possibile proprio dall'integrazione delle prove zkEVM e del *real-time proving*.

Lo sviluppo del presente lavoro di tesi si inserisce esattamente all'incrocio di queste tecnologie, proponendo la re-ingegnerizzazione di un *Execution Layer stateless* in grado di agire come puro validatore di

prove all'interno del framework *Fort Mode* delineato dal futuro del protocollo Ethereum.

Capitolo 3

ZeroNethermind

Questo capitolo presenta la progettazione ad alto livello di ZeroNethermind, un Proof-of-Concept sviluppato per dimostrare la fattibilità di un client Execution Layer operante in modalità completamente *stateless* tramite la verifica di prove crittografiche Zero-Knowledge.

L'esposizione è strutturata per illustrare il radicale cambio di approccio rispetto all'esecuzione tradizionale. Inizialmente verrà introdotto il concetto teorico alla base del sistema, definito come paradigma "Double Zero". Successivamente, verrà analizzata l'architettura a tre strati del nuovo validatore, per poi dettagliare il flusso di dati modificato all'interno dell'Engine API. Le logiche di basso livello e l'interoperabilità del codice saranno trattate nel `capitolo 4`, mentre la validazione sperimentale delle prestazioni nel `capitolo 5`.

3.1 Il Paradigma "Double Zero"

Il design di ZeroNethermind si regge su un approccio architetturale sintetizzato nel paradigma **"Double Zero"**, il quale combina due meccanismi complementari volti a eliminare i colli di bottiglia del nodo:

- **Zero State**: il client non mantiene alcun database di stato locale. Operando in modalità `MemDb`, tutta la persistenza è effimera e limitata ai soli header dei blocchi validati. L'impronta su disco è pertanto prossima allo zero.
- **Zero Knowledge**: la validità dei blocchi non è asserita tramite ri-esecuzione delle transazioni nell'EVM, bensì tramite la verifica crittografica di ZKPs generate da prover esterni.

L'adozione di questo paradigma comporta uno spostamento di complessità radicale: dalla ri-esecuzione nativa con costo $O(N_{gas})$, proporzionale al volume di gas consumato nel blocco, alla verifica crittografica con costo $O(1)$, indipendente dalla complessità del blocco. In questo modello, un blocco contenente una singola transazione richiede il medesimo tempo di verifica di un blocco contenente migliaia di transazioni complesse.

3.2 Architettura a Tre Strati

L'architettura di ZeroNethermind è organizzata in tre strati con responsabilità distinte, ciascuno isolato e comunicante attraverso interfacce ben definite. Questa stratificazione riflette sia le scelte ingegneristiche sia le necessità tecnologiche (interoperabilità cross-language tra C# e Rust).

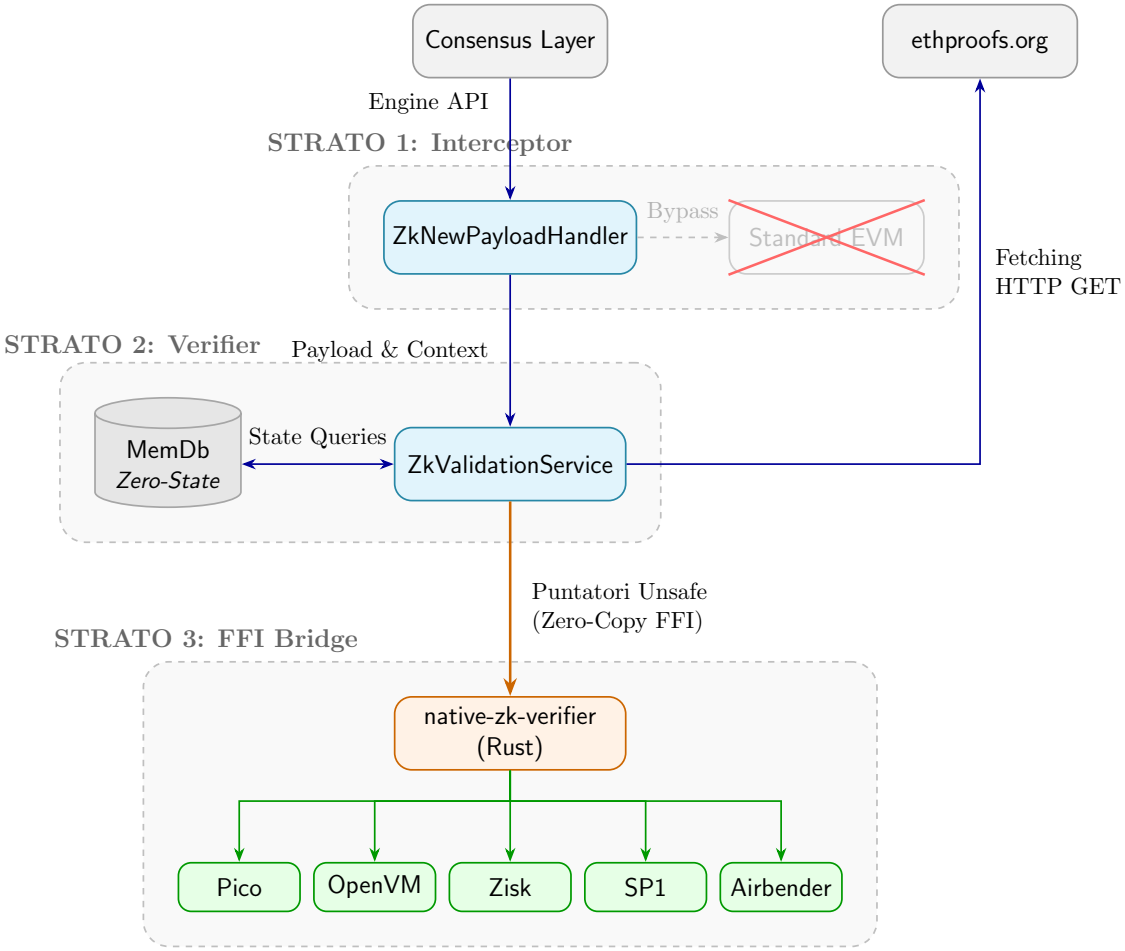

Figura 3.1: Architettura core di ZeroNethermind

La Figura 3.1 illustra l'architettura logica e il flusso di esecuzione del sistema, scomposto in tre macro-livelli:

3.2.1 Strato 1 (Interceptor)

Rappresenta il punto di contatto con l'Engine API. Il gestore custom (`ZkNewPayloadHandler`) intercetta i payload in ingresso dal Consensus Layer, bypassando integralmente la tradizionale macchina a stati (`Standard EVM`) e abbattendo radicalmente l'onere computazionale nativo.

3.2.2 Strato 2 (Verifier)

Costituisce il modulo di orchestrazione principale, implementato in `C#`. Il servizio orchestratore coordina il recupero parallelo delle prove tramite chiamate HTTP all'oracolo esterno (`ethproofs.org`). Inoltre, gestisce la persistenza effimera della catena: gli *header* validati vengono salvati nel `MemDb`, permettendo al client di operare in uno stato di totale dipendenza dalla memoria RAM (*Zero-State*), senza ricorrere ad alcun database su disco.

3.2.3 Strato 3 (FFI Bridge)

Rappresenta il confine di interoperabilità ad alte prestazioni tra l'ambiente *managed* di .NET e le librerie crittografiche native. Per evitare colli di bottiglia causati dalla serializzazione e dal Garbage Collector, i payload binari (prove e chiavi di verifica) vengono passati allo strato `Rust` tramite puntatori bloccati in memoria (*Zero-Copy FFI*). Questo strato agisce come dispatcher verso gli algoritmi di verifica specifici delle singole zkVM (Zisk, OpenVM, Pico, SP1, Airbender).

3.2.4 Ethproofs.org - Oracolo Esterno

Nell'architettura di ZeroNethermind, ethproofs.org riveste il ruolo di *single source of truth* per le ZK proofs, eliminando così la necessità per il PoC di operare una propria infrastruttura di proving, la quale richiederebbe cluster GPU di alto profilo e competenze specialistiche nella generazione delle prove. ZeroNethermind si concentra esclusivamente sulla verifica, delegando interamente la generazione all'ecosistema esterno.

3.3 Il Nuovo Flusso di Validazione

Il flusso di validazione in ZeroNethermind differisce radicalmente da quello standard descritto nella sezione 2.1.6. Il processo segue questi

step architetturali:

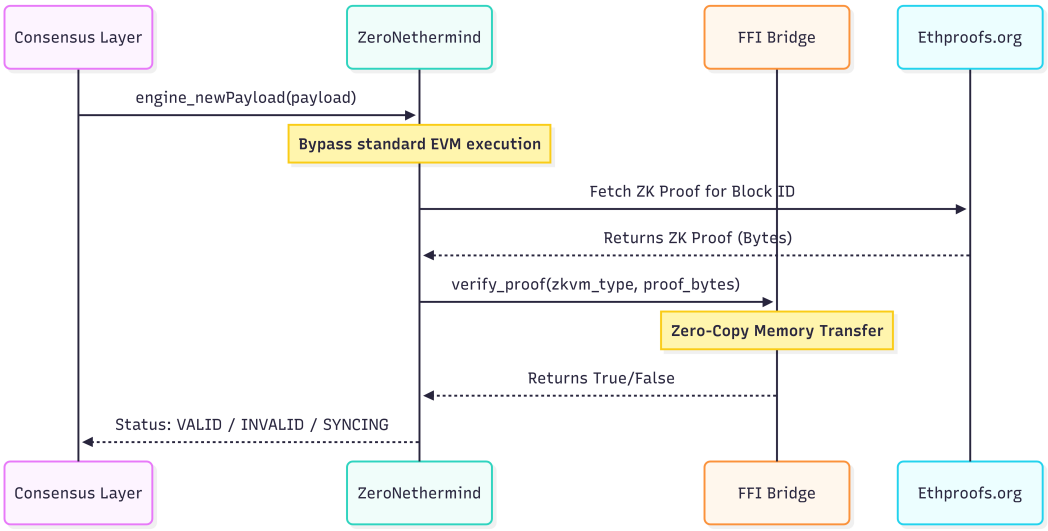

Figura 3.2: Flusso di Validazione di ZeroNethermind

1. **Ricezione del blocco:** il CL riceve un blocco dalla rete gossip ed invoca `engine_newPayload` verso ZeroNethermind, trasmettendo l'`ExecutionPayload`.
2. **Pre-processing minimale:** ZeroNethermind decodifica il blocco e ne valida l'hash dell'header. A differenza del flusso standard, non vengono eseguiti i controlli che dipendono dal database di stato locale (verifica parent header, controllo pre-pivot, etc.).
3. **Recupero delle prove:** il sistema interroga l'API di ethproofs.org per ottenere i metadati delle prove disponibili per il numero di blocco corrente.
4. **Verifica crittografica:** per ciascuna prova disponibile, ZeroNethermind scarica i byte binari della prova e della chiave di verifica, quindi invoca il verificatore nativo (Rust) tramite FFI.
5. **Maggioranza multi-prover:** il blocco è considerato valido se almeno il 50% delle prove verificate risulta valido. Questo approccio mitiga il rischio di bug in un singolo circuito zkVM.

6. **Risposta al CL:** ZeroNethermind restituisce lo status `VALID`, `INVALID` o `SYNCING` al Consensus Layer.
7. **Aggiornamento della catena:** alla ricezione di `engine_forkchoiceUpdated`, il sistema aggiorna la testa canonica.

È importante sottolineare che il nodo opera con un profilo di configurazione dedicato (`zk-mainnet`) che disabilita esplicitamente le componenti non necessarie — P2P, sincronizzazione, TxPool — e attiva la modalità MemDb. I dettagli di questa ottimizzazione saranno approfonditi nella sezione 4.4.

La Tabella 3.1 sintetizza le differenze architetturali tra un nodo Nethermind standard e ZeroNethermind. Il confronto evidenzia come il PoC ribalti il profilo delle risorse richieste: dallo storage massivo e dall'I/O intensivo di un full node, si passa a un modello dominato dalla latenza di rete.

Caratteristica	**Nethermind Standard**	**ZeroNethermind**
Input	Blocco + Stato Completo	Blocco + Prova ZK
Azione	Ri-esecuzione Tx (EVM)	Verifica Locale
Storage di Stato	∼2 TB (RocksDB)	∼0 GB (MemDb, solo header)
Risorse	Alta CPU, Alto I/O SSD	Bassa CPU, Alta Rete
Bottleneck	Velocità Disco (IOPS)	Disponibilità Prove (Latenza)
Cold Start	Ore (Snap Sync)	Secondi

Tabella 3.1: Confronto architetturale tra Nethermind Standard e ZeroNethermind.

Capitolo 4

Implementazione

Questo capitolo documenta il percorso ingegneristico dalla prima sperimentazione alla realizzazione del prototipo ZeroNethermind funzionante. L'implementazione è stata condotta in tre fasi incrementali, ciascuna costruita sulle fondamenta della precedente: la creazione di un verificatore ZK standalone (sezione 4.1), la sua integrazione nel client Nethermind (sezione 4.2), e l'orchestrazione del servizio di validazione completo (sezione 4.3). Il capitolo si conclude con l'analisi del profilo di configurazione che trasforma il client in un validatore stateless (sezione 4.4).

4.1 Fase 1: The Foundation

L'obiettivo della prima fase è stato costruire un verificatore ZK multi-prover standalone, denominato `EthProofValidator`, per dimostrare la fattibilità dell'interoperabilità .NET $\leftrightarrow$ Rust, indipendentemente dal client Nethermind.

4.1.1 L'Ecosistema dei Verifier ZK

Come descritto nel `capitolo 1`, l'ecosistema delle zkEVM è composto da molteplici team indipendenti, ciascuno dei quali ha sviluppato la

propria zkVM e il relativo sistema di proving. Per ZeroNethermind, questi verifier sono trattati come *black-box*: il PoC si interfaccia esclusivamente con le loro API di verifica, senza intervenire sulla logica interna.

Un aspetto tecnico rilevante è che tutti i principali framework di proving sono implementati in Rust, il linguaggio dominante per la computazione crittografica ad alte prestazioni. Questo crea una sfida d'integrazione significativa con il client Nethermind, che è interamente basato su C#/.NET: è necessario un bridge affidabile ed efficiente tra i due runtime.

Le cinque zkVM attualmente integrate nel PoC sono: Zisk (Polygon Hermez), OpenVM (OpenVM.org), Pico (Brevis), SP1 Hypercube (Succinct) e Airbender (zkSync/Matter Labs). Ciascun team espone un'interfaccia di verifica concettualmente analoga — una funzione che accetta una prova e una chiave di verifica, e ritorna un risultato booleano — ma con differenze nei formati di serializzazione e nelle dipendenze interne. Queste differenze rendono necessaria la creazione di un `crate` Rust unificato (`native-zk-verifier`) che funge da adattatore comune.

4.1.2 WASM vs Native FFI

Per realizzare il bridge C# $\leftrightarrow$ Rust, sono state valutate due strategie architetturali: il caricamento dei verifier come moduli WebAssembly (WASM), come fatto da ethproofs.org per l'*in-browser* validation, tramite il runtime `Wasmtime.NET`, e l'invocazione diretta tramite Foreign Function Interface (FFI) nativa con `P/Invoke`. La tabella 4.1 sintetizza il confronto.

Aspetto	WASM	Native FFI (P/Invoke)
Prestazioni	Buone (VM overhead).	Massime (esecuzione diretta su CPU).
Portabilità	Cross-platform nativo.	Compilazione per OS specifico.
Sicurezza	Sandboxed.	`unsafe`: un errore Rust può causare il crush del processo .NET.
Complessità	Richiede il pacchetto NuGet `Wasmtime`.	Nessuna dipendenza esterna.

Tabella 4.1: Confronto tra le strategie di bridge WASM e Native FFI.

La scelta è ricaduta sull'approccio FFI nativo per la priorità assegnata alle prestazioni massimali. In un contesto dove la verifica delle prove deve avvenire entro la finestra temporale di 12 secondi di uno slot Ethereum, ogni overhead è significativo. L'approccio WASM, pur offrendo un migliore isolamento, introduce un livello di indirezione (la VM WebAssembly) che penalizza il throughput in scenari di verifica intensiva. In cambio della perdita del sandboxing, il bridge nativo offre zero overhead di marshalling ed esecuzione diretta delle istruzioni sulla CPU del nodo.

4.1.3 Il Bridge FFI: `native-zk-verifier`

Il ponte tra i due runtime è realizzato dal crate Rust `native-zk-verifier`, configurato come `cdylib` (libreria condivisa C-compatibile) e compilato con Rust nightly, come richiesto dalle dipendenze crittografiche.

Lato Rust

Il crate espone una singola funzione C-ABI che funge da dispatcher verso il verifier appropriato:

```rust
#[no_mangle]
pub extern "C" fn verify(
    zk_type: u32,
    proof_ptr: *const u8, proof_len: usize,
    vk_ptr: *const u8,    vk_len: usize
) -> i32
```

Listing 4.1: Dispatcher FFI in Rust

Il parametro `zk_type` seleziona il verifier tramite un enum intero. L'intera chiamata è avvolta in `panic::catch_unwind`, un costrutto Rust che cattura eventuali panic nel codice dei verifier impedendo che si propaghino al processo .NET host, il quale crasherebbe irreversibilmente. I valori di ritorno seguono una convenzione semplice: `1` (Valid), `0` (Invalid), `-1` (Error).

Internamente, ciascun modulo verifier (`zisk.rs`, `openvm.rs`, `pico.rs`, `sp1_hypercube.rs`, `airbender.rs`) implementa un trait comune `Verifier` con una funzione `verify(proof, vk) -> Result<bool>`, astraendo le differenze di serializzazione e di librerie sottostanti.

Lato C#

Sul versante .NET, la dichiarazione P/Invoke utilizza il source-generated marshalling di .NET 8+:

```csharp
[LibraryImport("native_zk_verifier")]
public static partial int verify(
    int zk_type,
    byte[] proof, nuint proof_len,
    nint vk_ptr,  nuint vk_len);
```

Listing 4.2: Dichiarazione P/Invoke in C#

La scelta di `LibraryImport` rispetto al più tradizionale `DllImport` elimina l'overhead del marshalling a runtime, generando il codice di interoperabilità staticamente in fase di compilazione.

Gestione della Memoria

Il modello di ownership della memoria tra i due runtime è un aspetto critico del bridge:

- **Verification Key (VK):** Allocata in memoria unmanaged tramite `Marshal.AllocHGlobal`, al di fuori del Garbage Collector .NET. Questo evita che il GC sposti o liberi la VK durante la chiamata FFI. Il wrapper `ZkProofVerifier` implementa `IDisposable` per il rilascio deterministico della memoria unmanaged.
- **Proof data:** `Owned` dal GC di C# e passata come array `byte[]`. Il runtime .NET effettua il `pinning` automatico durante la chiamata P/Invoke.
- **Rust:** Agisce come un `guest` puramente funzionale: riceve puntatori read-only, esegue la verifica, e ritorna un intero. Nessuna allocazione o liberazione di memoria avviene sul lato Rust.

Build Automatica

L'integrazione tra i due sistemi di build è gestita da un target MSBuild custom (`BuildRustDependency`) nel file `.csproj`:

1. Esegue `cargo build --release` nella directory `native-zk-verifier/`.
2. Copia la libreria nativa risultante (`.so`/`.dll`/`.dylib`) nella directory di output del progetto .NET.

Questo garantisce che un singolo comando `dotnet build` compili automaticamente sia il codice C# che il codice Rust, semplificando notevolmente il workflow di sviluppo.

4.1.4 L'Applicazione Console Multi-Verifier

Con il bridge FFI operativo, la prima fase culmina nella creazione di `EthProofValidator`: un'applicazione console standalone C# che valida blocchi mainnet reali senza alcun overhead del client Nethermind.

Architettura

L'applicazione è strutturata in quattro componenti:

- `Program.cs`: Entry point CLI che accetta `<LatestBlockId> <BlockCount>` e itera sui blocchi invocando il validatore.
- `BlockValidator`: Orchestratore che per ciascun blocco recupera le prove da ethproofs.org, le distribuisce ai verifier in parallelo (`Task.WhenAll`), e applica la logica di majority voting.
- `VerifierRegistry`: Registry pattern implementato con `ConcurrentDictionary<clusterId, ZkProofVerifier>`. I verifier vengono creati *lazily* al primo incontro con un nuovo cluster, scaricando la relativa VK dall'API di ethproofs.org.
- `EthProofsApiClient`: Client HTTP per le API REST di ethproofs.org (fetch prove, download binari, download VK).

Majority Voting

La logica di accettazione di un blocco segue un algoritmo di *majority voting* multi-prover: un blocco è considerato valido se almeno il 50% delle prove non-skipped risulta valido (`validCount * 2 >= totalCount`). Questa soglia rappresenta un compromesso tra sicurezza e tolleranza ai guasti: un singolo verifier con errori non invalida l'intero blocco, purché la maggioranza converga sullo stesso risultato.

Risultati della Demo Standalone

L'applicazione è stata eseguita su una sequenza di 50 blocchi consecutivi della mainnet Ethereum (#24199951–#24200000), con 5 prove da 5 zkVM distinte per ciascun blocco. I risultati chiave:

- **Tutti i 50 blocchi validati con successo** (5/5 prove valide per blocco).

- **Tempi di verifica per singola prova:** ∼5–50 ms (Zisk, Airbender), ∼90–140 ms (OpenVM, Pico).
- **Tempo totale per blocco:** ∼1.6–2.5 s, dominato dal download HTTP delle prove e non dalla verifica crittografica.

Di seguito è riportato un estratto dei log di esecuzione che illustra l'elaborazione di una sottosequenza di 3 blocchi:

```
Processing Block #24199996
   Proof 5218300 - Zisk             : Valid (43 ms)
   Proof 5218298 - Zisk             : Valid (17 ms)
   Proof 5218303 - OpenVM           : Valid (114 ms)
   Proof 5218306 - Airbender        : Valid (13 ms)
   Proof 5218301 - Pico             : Valid (89 ms)
   --------------------------------
BLOCK #24199996 ACCEPTED (5/5)
Total Time: 2059 ms

Processing Block #24199997
   Proof 5218316 - Zisk             : Valid (40 ms)
   Proof 5218315 - Zisk             : Valid (15 ms)
   Proof 5218322 - OpenVM           : Valid (107 ms)
   Proof 5218326 - Airbender        : Valid (12 ms)
   Proof 5218319 - Pico             : Valid (112 ms)
   --------------------------------
BLOCK #24199997 ACCEPTED (5/5)
Total Time: 1868 ms

Processing Block #24199998
   Proof 5218337 - Zisk             : Valid (35 ms)
   Proof 5218340 - OpenVM           : Valid (111 ms)
   Proof 5218335 - Zisk             : Valid (6 ms)
   Proof 5218346 - Airbender        : Valid (33 ms)
   Proof 5218339 - Pico             : Valid (115 ms)
   --------------------------------
BLOCK #24199998 ACCEPTED (5/5)
Total Time: 1857 ms
```

Listing 4.3: Estratto log di esecuzione EthProofValidator

Questi risultati dimostrano la fattibilità del modello multi-verifier e

confermano che la verifica crittografica è trascurabile rispetto alla latenza di rete, validando empiricamente la scelta del bridge FFI nativo.

4.2 Fase 2: Nethermind Integration

Con il verificatore standalone operativo, la seconda fase affronta l'integrazione della logica di validazione ZK nel ciclo di vita del client Nethermind, senza eseguire un fork del repository upstream.

4.2.1 Alternative Considerate

Prima di giungere alla soluzione finale, sono state valutate e scartate diverse strategie di integrazione:

1. **Strategy Pattern nel `NewPayloadHandler`:** Definire un'interfaccia `IPayloadExecutionStrategy` con due implementazioni (`LocalExecutionStrategy` per l'EVM standard, `ZkValidationStrategy` per la verifica ZK). Scartata perché richiedeva la modifica del codice core del gestore standard, violando il principio di non-invasività.
2. **Modifica del `BlockchainProcessor`:** Come suggerito dal team Nethermind, intervenire a livello del processore di blocchi per bypassare l'esecuzione EVM. Scartata perché il `BlockchainProcessor` è fortemente accoppiato alla macchina a stati standard e avrebbe richiesto modifiche troppo invasive.
3. **Override diretto del `MergePlugin`:** Estendere il plugin esistente che gestisce il consenso post-Merge. Scartata perché avrebbe creato una dipendenza fragile dalla struttura interna del `MergePlugin`, soggetta a rotture ad ogni aggiornamento del client.

4.2.2 L'Architettura a Plugin

La soluzione adottata sfrutta il sistema di plugin modulare di Nethermind, basato sul container di Dependency Injection **Autofac**. L'interfaccia chiave è `IConsensusWrapperPlugin`, che permette di:

- **Registrare tipi custom** nel container DI tramite un `Autofac.Module`.
- **Sovrascrivere i gestori** dell'Engine API senza modificare il codice core di Nethermind.

Questo approccio è il medesimo utilizzato dal `MergePlugin` (che gestisce il consenso post-Merge) e dallo `ShutterPlugin` (che estende la logica di consenso per il protocollo Shutter), garantendo compatibilità architetturale con il design del client.

4.2.3 `ZkValidationPlugin`: Il Punto di Ingresso

Il plugin `ZkValidationPlugin` implementa `IConsensusWrapperPlugin` ed è il punto di ingresso dell'intero PoC. La sua attivazione è condizionale: la flag `ZkValidation.Enabled` è impostata a `true` nel file di configurazione, e il modulo `Merge` è attivo (requisito per il funzionamento dell'Engine API post-Merge).

Il `ZkValidationPluginModule` (Autofac Module) registra nel container DI tutti i servizi necessari:

- `ZkValidationService` (orchestratore condiviso)
- `ZkNewPayloadHandler`
- `ZkForkchoiceUpdatedHandler`
- `BlockValidator`
- `EthProofsApiClient`

4.2.4 Override dell'Engine API

`ZkNewPayloadHandler`

Il gestore ZK di `engine_newPayload` sostituisce il `NewPayloadHandler` standard e semplifica radicalmente il pre-processing. La tabella 4.2 sintetizza i controlli mantenuti e quelli eliminati.

Controllo	**ZK**	**Motivazione**
Decodifica Blocco	✓	Necessario per ottenere la struttura dati
Validazione Hash	✓	Integrità crittografica dell'header
Invalid Chain Tracker	✓	Propagazione di invalidità
Controllo Pre-Pivot	×	Irrilevante senza sincronizzazione
Ricerca Header Genitore	×	Nessun database locale
Inserimento Blocchi Orfani	×	Nessun sync in corso
Controllo ShouldProcessBlock	×	Si vuole sempre la validazione ZK
Controlli TTD	×	Solo blocchi post-merge
Esecuzione EVM	×	**Sostituito dalla verifica ZK**

Tabella 4.2: Confronto dei controlli di pre-processing: gestore standard vs. gestore ZK.

Dopo il pre-processing semplificato, il gestore delega la validazione al `ZkValidationService`, il quale verifica in un tempo massimo la disponibilità delle prove, le scarica ed esegue la validazione locamente restituendo `VALID` o `INVALID` di conseguenza. Se le prove non sono disponibili l'EL ritornerà lo stato di `SYNCING`, aspettando nuovamente la chiamata di `engine_newPayload` dal CL.

`ZkForkchoiceUpdatedHandler`

Il gestore ZK di `engine_forkchoiceUpdated` bypassa il controllo `WasProcessed`, una flag normalmente impostata a `true` solo dopo l'e-

secuzione EVM e quindi sempre `false` in modalità ZK. Le operazioni residue sono:

1. Verifica che il blocco non appartenga a una catena invalidata (`IsOnInvalidChain`).
2. Ricerca del blocco nel `blockTree`.
3. Aggiornamento del `blockTree.ForkChoiceUpdated(...)`.
4. Restituzione di `VALID` con il `headBlockHash` aggiornato.

4.3 Fase 3: The ZK Validation Service

La terza fase integra la logica di verifica nel contesto operativo completo del client, affrontando le sfide di latenza dei prover, gestione della cache, e interazione con il protocollo di consenso.

4.3.1 `ZkValidationService`: L'Orchestratore

Il `ZkValidationService` è il servizio condiviso tra i due gestori dell'Engine API. Le sue responsabilità includono:

- **Cache dei blocchi validati:** Prima di ogni verifica, il servizio controlla se il blocco è già presente nel `blockTree` tramite `FindBlock(hash)`. In caso affermativo, ritorna immediatamente `VALID` senza re-verifica, evitando lavoro ridondante quando il CL invia lo stesso blocco sia via `newPayload` che via `forkchoiceUpdated`.
- **Inserimento nel BlockTree:** Su risultato `VALID`, il blocco viene inserito con `blockTree.Insert(block, SaveHeader, BeaconBlockInsert)`, persistendo solo l'header nel `MemDb` senza scrivere alcun dato di stato.

- **Gestione dell'invalidità:** Su risultato `INVALID`, il servizio invoca `invalidChainTracker.OnInvalidBlock(hash, parentHash)`, marcando l'intera catena discendente come invalida e propagando l'informazione ai successivi blocchi figli.

4.3.2 La Gestione del Protocollo SYNCING

La sfida ingegneristica principale della Demo 1 (sezione 5.1.2) è stata la gestione della latenza dei prover. Quando un blocco appena prodotto dalla rete raggiunge il nodo, la prova crittografica potrebbe non essere ancora disponibile su ethproofs.org (il tempo medio di proving è di circa 5–15 secondi). Senza un meccanismo di gestione adeguato, il nodo entrerebbe in uno stato `SYNCING` permanente, poiché il CL continuerebbe a inviare nuovi blocchi prima che i precedenti siano verificati.

La soluzione implementata è un **pattern Active-Wait**: il `ZkValidationService` esegue un retry loop configurabile che blocca il thread RPC durante l'attesa della prova. Il flusso operativo è il seguente:

1. Il `BlockValidator` interroga ethproofs.org e non trova prove disponibili → ritorna `SYNCING`.
2. Il servizio attende `RetryDelayMs` e ritenta la query fino al numero massimo di tentativi.
3. Il CL (Lighthouse) riceve `SYNCING` e passa in modalità "optimistic" (assume provvisoriamente il blocco come valido in attesa della conferma dell'EL).
4. Una volta che la prova è disponibile e verificata, il servizio ritorna `VALID`.
5. Lighthouse transita a `INFO Synced ... (verified)`.

Questo approccio rappresenta un **workaround funzionale** per il PoC: il blocco del thread RPC è accettabile in un contesto di dimostrazione, ma non sarebbe sostenibile in un ambiente di produzione dove la reattività dell'Engine API è critica. Un'implementazione production-ready richiede prima un addattamento e modifica del protocollo attuale.

4.3.3 Adattamento del BlockValidator

Il `BlockValidator` sviluppato nella sezione 4.1 (standalone) è stato adattato al contesto del plugin Nethermind con le seguenti modifiche:

- Integrazione con il sistema di logging di Nethermind (`ILogManager`) per una diagnostica unificata.
- Condivisione del `VerifierRegistry` e dell'`EthProofsApiClient` tramite il container DI di Autofac, anziché istanziazione diretta.
- Medesima logica di majority voting, con gestione degli errori adattata: un errore su un singolo verifier non interrompe il processo, ma viene registrato e conteggiato nella votazione.

4.4 Profilo di Configurazione

Il file `zk-mainnet.json` definisce il profilo operativo che trasforma Nethermind in un validatore "Fort Mode". La 4.3 riassume le componenti disabilitate e le relative motivazioni.

Namespace	Parametro	Valore	Motivazione
Init	DiagnosticMode	MemDb	Tutte le store in memoria, zero I/O su disco
Init	DiscoveryEnabled	false	Nessun peer discovery P2P
Init	PeerManagerEnabled	false	Nessuna gestione connessioni P2P
Init	ProcessingEnabled	false	Disabilita il `BlockchainProcessor`
Sync	SynchronizationEnabled	false	Nessuna sincronizzazione
TxPool	Size	0	Nessun gossip di transazioni, nessun mempool

Tabella 4.3: Moduli disabilitati nel profilo `zk-mainnet`.

I moduli che restano attivi sono esclusivamente quelli necessari al funzionamento del validatore stateless: `JsonRpc` (con i soli moduli `Net`, `Web3`, `Engine` per la comunicazione con il CL), `Merge` (prerequisito per l'Engine API post-Merge), `ZkValidation` (attivazione del plugin), `HealthChecks` e `Metrics` (monitoraggio e telemetria).

Il risultato è un nodo che non ha database su disco (MemDb al posto dei $\sim$2 TB di RocksDB), non scopre né gestisce peer (riceve blocchi esclusivamente dal CL adiacente), non esegue transazioni (il `BlockchainProcessor` è disattivato), non partecipa al gossip delle transazioni, e valida esclusivamente tramite prove ZK. Questo profilo materializza concretamente il concetto di "Fort Mode" descritto nel `capitolo 1`: un validatore leggero, deployabile su hardware consumer, che contribuisce alla sicurezza della rete delegando la generazione delle prove ai "Beast" e concentrandosi sulla sola verifica crittografica.

Capitolo 5

Validazione

La validazione del PoC ZeroNethermind si articola attorno a due domande di ricerca complementari:

1. **Fattibilità operativa:** Un verificatore stateless può mantenere la sincronizzazione con la chain L1 Mainnet in tempo reale, operando esclusivamente tramite prove crittografiche?
2. **Impatto sulle risorse:** Quali sono le differenze quantitative in termini di velocità di verifica, consumo di CPU, RAM, I/O su disco e storage tra un nodo standard a ri-esecuzione completa e un nodo ZeroNethermind?

Per rispondere a queste domande, sono state progettate due campagne sperimentali distinte: la **Demo 1** (Sezione 5.1.2), che effettua una validazione live sulla Mainnet Ethereum tramite prove fornite da `ethproofs.org`, e la **Demo 2** (Sezione 5.1.3, che utilizza un ambiente controllato basato su Kurtosis[17] per isolare e confrontare le metriche hardware in condizioni deterministiche.

5.1 Metodologia

5.1.1 Ambiente di Test

Tutti gli esperimenti sono stati eseguiti sulla medesima macchina fisica, utilizzando container Docker per garantire l'isolamento e la riproducibilità. Le specifiche hardware sono riportate nella tabella 5.1.

Componente	**Specifica**
Sistema Operativo	Manjaro Linux, kernel 6.12.73 (64-bit)
CPU	4 x Intel Core i5-7400 @ 3.00 GHz
RAM	16 GiB DDR4
GPU	NVIDIA GeForce GTX 1060 6 GB
Rete	Connessione domestica (FTTC)
Containerizzazione	Docker Engine con `docker-compose`

Tabella 5.1: Specifiche hardware della macchina di test.

La scelta di hardware consumer di fascia media è intenzionale: l'obiettivo del PoC è dimostrare che la validazione ZK è realizzabile su dispositivi comuni, senza necessità di server specializzati.

5.1.2 Demo 1: Validazione in Tempo Reale sulla Mainnet

Setup

La Demo 1 consiste nel deployment di ZeroNethermind accoppiato con un Consensus Layer Lighthouse, entrambi eseguiti come container Docker sulla macchina locale.

- **Execution Layer:** ZeroNethermind con il profilo `zk-mainnet` (sezione 4.4), configurato in modalità `MemDb` (zero storage su disco), con discovery P2P, peer manager, sincronizzazione e `BlockchainProcessor` disabilitati.

- **Consensus Layer:** Lighthouse in modalità `checkpoint-sync`, sincronizzato al tip della Beacon Chain tramite il checkpoint pubblico `mainnet.checkpoint.sigp.io`.
- **Fonte delle prove:** `ethproofs.org` via HTTP API, interrogato automaticamente dal `BlockValidator` per ogni blocco ricevuto tramite `engine_newPayload`.
- **Monitoraggio:** Prometheus[22] + cAdvisor[15] per il campionamento delle metriche container (CPU, RAM, I/O, bandwidth) con frequenza di 15 secondi.

Obiettivo

Dimostrare la fattibilità operativa del paradigma "Fort Mode": il nodo, privo di database e di capacità di esecuzione EVM, deve riuscire a validare i blocchi della Mainnet in tempo reale, mantenendo la sincronizzazione con il CL e rispondendo a ciascuna chiamata `engine_newPayload` con uno status `VALID` entro i vincoli temporali dello slot.

Procedura

Il test è stato condotto su una finestra di circa 17 minuti, durante i quali il nodo ha ricevuto e validato **121 blocchi Mainnet** (nel range `#24628200` – `#24628397`), corrispondenti a circa 3 epoche della Beacon Chain. Per ciascun blocco, il `BlockValidator` ha:

1. Interrogato `ethproofs.org` per ottenere l'elenco delle prove disponibili.
2. Scaricato in parallelo le prove binarie generate da diversi zkVM indipendenti: **Zisk** (tipicamente 2 istanze), **OpenVM**, **Pico** e **Airbender**.
3. Invocato il bridge FFI nativo (`libnative_zk_verifier.so`) per la verifica crittografica di ciascuna prova.

4. Eseguito il majority voting: il blocco è accettato se almeno il 50% delle prove verificate risulta valido.
5. Inserito il blocco (*header-only*) nel `BlockTree` in-memory e restituito `VALID` al CL.

Per ogni blocco validato, il sistema ha registrato nel log: il tempo totale di validazione, il tempo di download massimo (bottleneck del parallelismo HTTP), il tempo di verifica FFI massimo (bottleneck del parallelismo crittografico), e il conteggio prove valide/totali. I dati sono stati estratti dai log del container e aggregati in formato CSV.

5.1.3 Demo 2: Ambiente Controllato (Kurtosis Lab)

Setup

La Demo 2 utilizza un'enclave Kurtosis per creare una devnet Ethereum locale completamente isolata, configurata per confrontare direttamente il footprint hardware di un nodo Nethermind standard e di un nodo ZeroNethermind.

- **Nodi standard:** 2 istanze Nethermind con configurazione di default, equipaggiate con validatori e MEV-Boost.
- **Nodo ZeroNethermind:** 1 istanza con il profilo `zk-mainnet` e il plugin `ZkValidation` attivo, connessa al **Mock Prover** locale.
- **Consensus Layer:** Lighthouse per tutti i nodi.
- **Mock Prover:** Un servizio containerizzato (`zero-prover`) che replica l'API di `ethproofs.org`. Per ogni blocco restituisce **5 istanze** della medesima prova crittografica Airbender pre-generata e valida, simulando il carico di lavoro reale in cui il nodo scarica e verifica prove da 5 zkVM indipendenti in parallelo. Questa scelta consente di misurare il tempo di verifica FFI in condizioni realistiche, eliminando al contempo la variabilità della latenza di rete verso un servizio esterno.

- **Carico di lavoro:** `spamoor` genera un flusso continuo di transazioni per saturare i blocchi al Gas Limit corrente.
- **Monitoraggio:** Prometheus, Grafana e `ethereum_metrics_exporter` per la raccolta integrata delle metriche di tutti i nodi.

L'utilizzo di MEV-Boost per entrambe le tipologie di nodo è intenzionale: delegando la costruzione dei blocchi a un builder esterno, tutti i nodi si limitano alla sola fase di validazione, che è esattamente il componente sotto test.

Obiettivo

Quantificare il divario di risorse hardware tra validazione per ri-esecuzione EVM ($O(N_{gas})$) e validazione per verifica crittografica ($O(1)$), in un ambiente dove le condizioni di rete, il carico transazionale e l'hardware sono identici per tutti i nodi.

Procedura

La Demo 2 adotta un protocollo a **carico progressivo** della durata di ∼45 minuti, articolato in due fasi:

1. **Fase Baseline** (primi ∼15 minuti): la devnet opera con un throughput transazionale minimo (1 transazione per tipo di spammer), al fine di stabilire i valori di riferimento per tutte le metriche di interesse.
2. **Fase sotto carico** (ultimi ∼30 minuti): il throughput di `spamoor` viene incrementato manualmente in 4 step successivi, aumentando gradualmente il numero di transazioni per tipo (`storagespam`, `gasburnertx`, `erc20tx`, `eoatx`) fino a saturare i blocchi al Gas Limit di 500 M.

Durante l'intero esperimento, le metriche vengono raccolte tramite due fonti complementari: (i) le metriche native di Nethermind esposte via Prometheus (in particolare `nethermind_new_payload_execution_time`

per il tempo di validazione del `newPayload`), e (ii) cAdvisor per le metriche di risorse a livello di container (CPU, RAM, I/O, bandwidth, storage).

I risultati della Demo 2 sono presentati nella sezione 5.3.

5.2 Demo 1: Validazione sulla Mainnet

I risultati della Demo 1 sono presentati attraverso due visualizzazioni complementari:

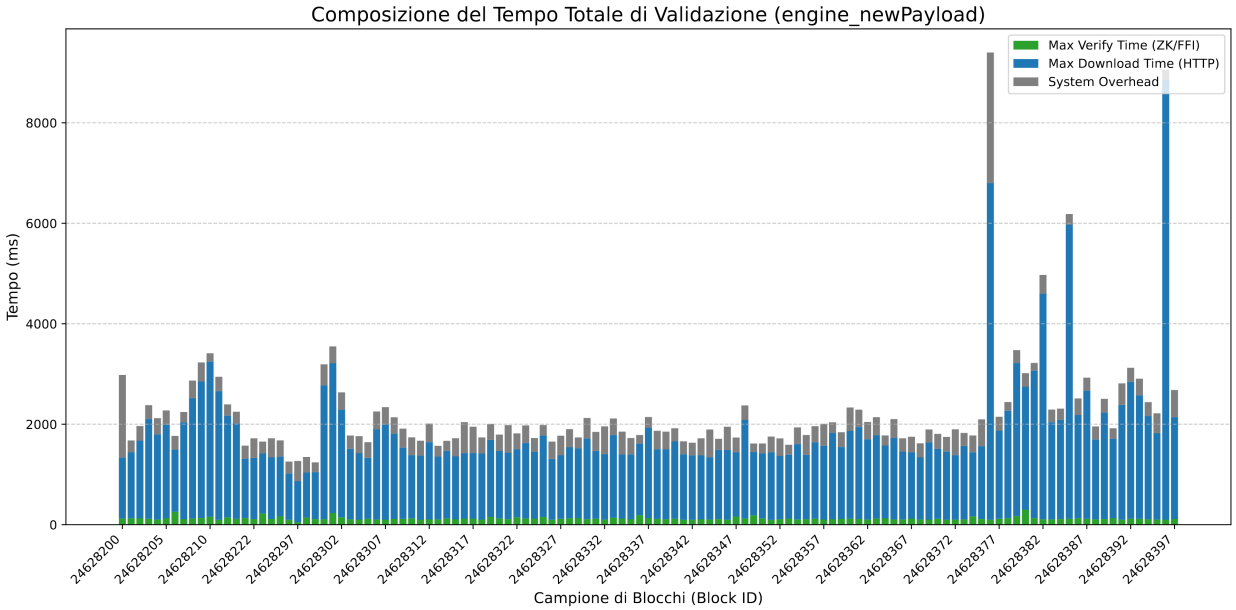

Figura 5.1: Composizione del Tempo Totale di Validazione

Il **Grafico 5.1** illustra la scomposizione del tempo totale per validare ciascun blocco all'interno del gestore `engine_newPayload`. Come si evince chiaramente dalla predominanza della componente blu rispetto a quella verde, il *Max Download Time* (il tempo impiegato per recuperare le prove tramite HTTP da ethproofs.org) costituisce il vero e unico collo di bottiglia del processo, assorbendo regolarmente oltre il 90% del tempo totale. Al contrario, il *Max Verify Time* (in verde),

ovvero il tempo di calcolo effettivo speso dalla CPU per la verifica crittografica tramite il bridge FFI, si mantiene costante e irrisorio (nell'ordine dei 100-150 ms), indipendentemente dalla complessità del blocco. I picchi anomali visibili (fino a ∼9000 ms) sono imputabili esclusivamente a latenze di rete transitorie del provider esterno.

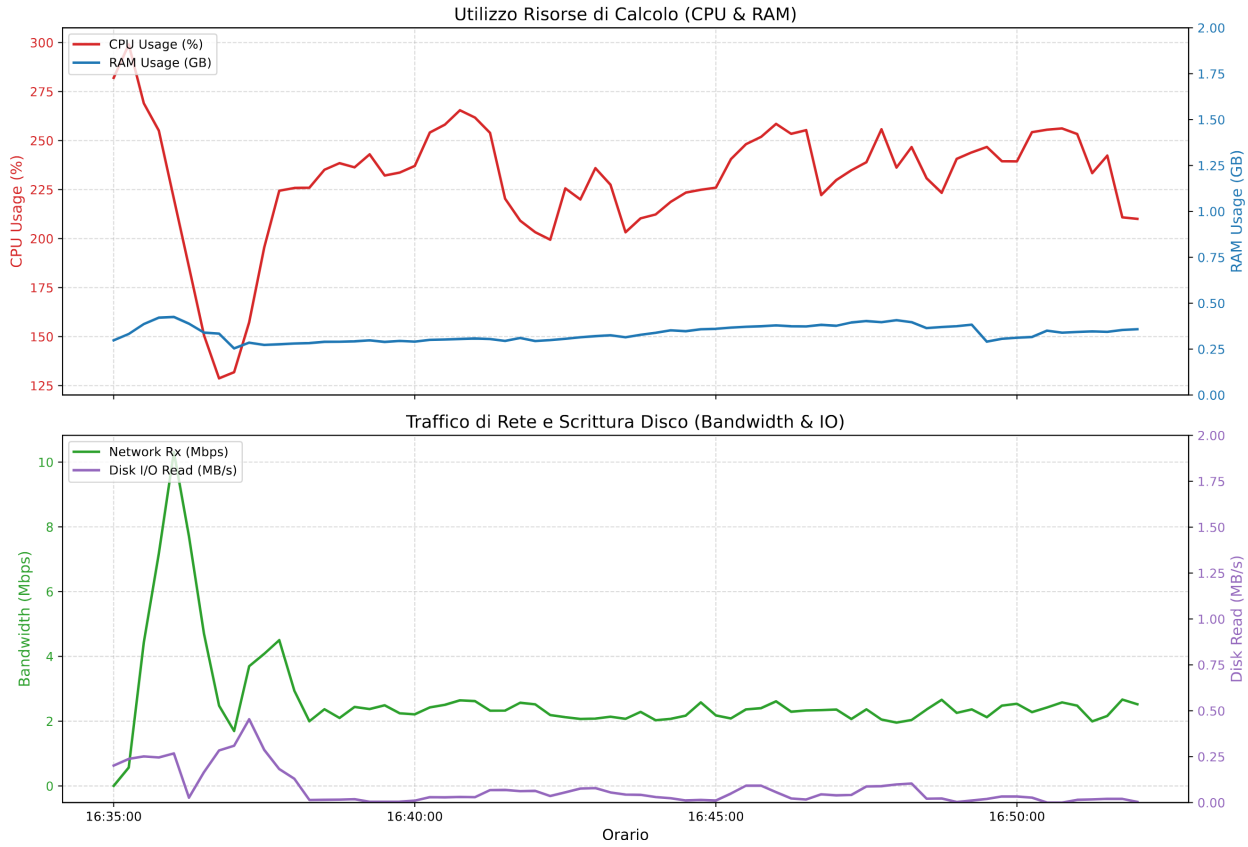

Figura 5.2: Analisi Prestazionale: Nodo Stateless ZeroNethermind

Il **Grafico 5.2** conferma empiricamente il successo del paradigma "Zero State" attraverso l'analisi temporale dell'utilizzo delle risorse hardware. Nel pannello superiore, l'utilizzo della memoria RAM (linea blu) si mantiene straordinariamente basso e piatto, assestandosi costantemente al di sotto dei 400 MB. Questo dimostra l'efficacia dell'eliminazione del database di stato persistente e della Mempool. Nel pannello inferiore, la metrica più significativa è il traffico I/O su disco

(linea viola), che, dopo un minimo assorbimento in fase di avvio, si azzera del tutto (0 MB/s), confermando l'assenza totale delle pesanti operazioni di lettura/scrittura sull'albero MPT tipiche dell'EVM. Contestualmente, la banda di rete in ricezione (linea verde) mostra pattern regolari associati al download continuo delle prove crittografiche per ogni nuovo slot, confermando che il sistema si è spostato da un limite *I/O-bound* a un limite *Bandwidth-bound*.

La tabella 5.2 riporta le statistiche aggregate della sessione.

Metrica	**Media**	**Mediana**	**Min**	**Max**
Tempo totale di validazione (ms)	2 237	1 950	1 241	9 400
Max download time (ms)	1 768	1 447	817	8 755
Max verify time (ms)	122	116	48	301
RAM container (MB)	360	335	272	457
CPU container (%, per-core)	57,7	—	32,2	74,8
Bandwidth RX (KB/s)	333	295	<1	1 289
I/O read (KB/s)	77	34	<1	476

Tabella 5.2: Statistiche aggregate della Demo 1 (121 blocchi Mainnet).

Il risultato più significativo è la sproporzione tra il tempo di download e il tempo di verifica: il tempo medio di verifica crittografica FFI è di soli **122 ms** (mediana 116 ms), rappresentando appena il **5,5%** del tempo totale medio di validazione. Il restante 94,5% è dominato dalla latenza di rete per il download delle prove da `ethproofs.org`. Questo dato conferma che la verifica ZK è computazionalmente trascurabile rispetto al trasporto dei dati, e anticipa una proprietà fondamentale: quando le prove saranno incluse direttamente nel payload del blocco, il tempo di validazione si ridurrà esclusivamente alla componente di verifica, nell'ordine di $\sim$100 ms.

5.3 Demo 2: Analisi in Ambiente Controllato

I risultati della Demo 2 confermano quantitativamente le ipotesi formulate nel `capitolo 1`: il paradigma di validazione per verifica crittografica ($O(1)$) presenta vantaggi strutturali rispetto alla ri-esecuzione EVM ($O(N_{gas})$), sia in termini di tempo di elaborazione sia di consumo di risorse hardware.

5.3.1 Velocità di Verifica

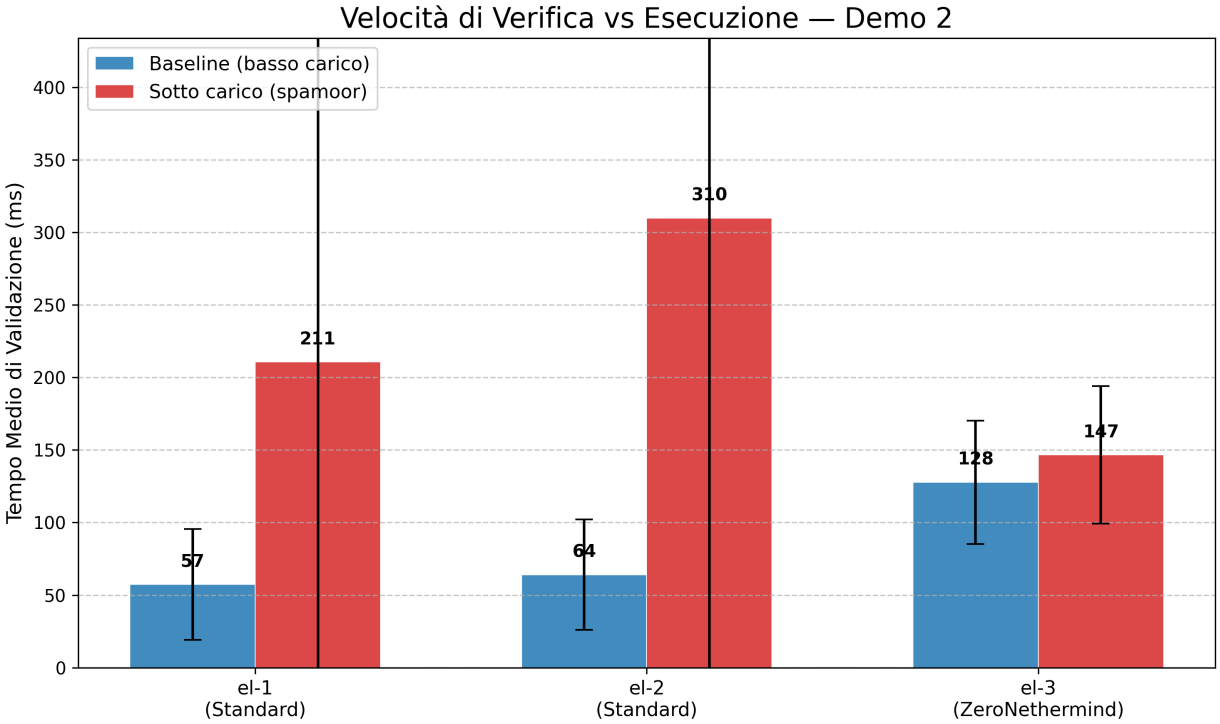

Figura 5.3: Velocità di Verifica vs Esecuzione: confronto dei tempi medi di validazione nelle fasi Baseline e sotto carico.

Il **Grafico 5.3** rivela un'asimmetria fondamentale tra i due approcci di validazione. La Tabella 5.3 riporta le statistiche dettagliate per ciascun nodo nelle due fasi sperimentali.

Nodo	Baseline		Sotto carico		Complessivo	
	Media	Max	Media	Max	Media	Max
el-1 Standard	57	264	211	1 129	162	1 129
el-2 Standard	64	203	310	1 487	226	1 487
el-3 ZeroNethermind	128	268	147	334	141	334

Tabella 5.3: Tempo di validazione `engine_newPayload` (ms) per fase.

In condizioni di basso carico (*Baseline*), i nodi standard risultano più veloci (∼57–64 ms) rispetto a ZeroNethermind (∼128 ms), poiché la ri-esecuzione di transazioni leggere è computazionalmente meno costosa del download e della verifica di 5 prove crittografiche in parallelo. Tuttavia, sotto carico transazionale crescente, il rapporto si inverte drasticamente: i nodi standard rallentano a ∼211–310 ms con picchi fino a ∼1 500 ms, mentre ZeroNethermind resta sostanzialmente invariato a ∼147 ms. L'incremento medio per i nodi standard è di $\sim 3{,}7\times$ rispetto alla baseline, contro un aumento di appena il 15% per ZeroNethermind. Questa è la manifestazione empirica della complessità $O(N_{gas})$ vs $O(1)$: il tempo di verifica crittografica è indipendente dalla complessità computazionale del blocco.

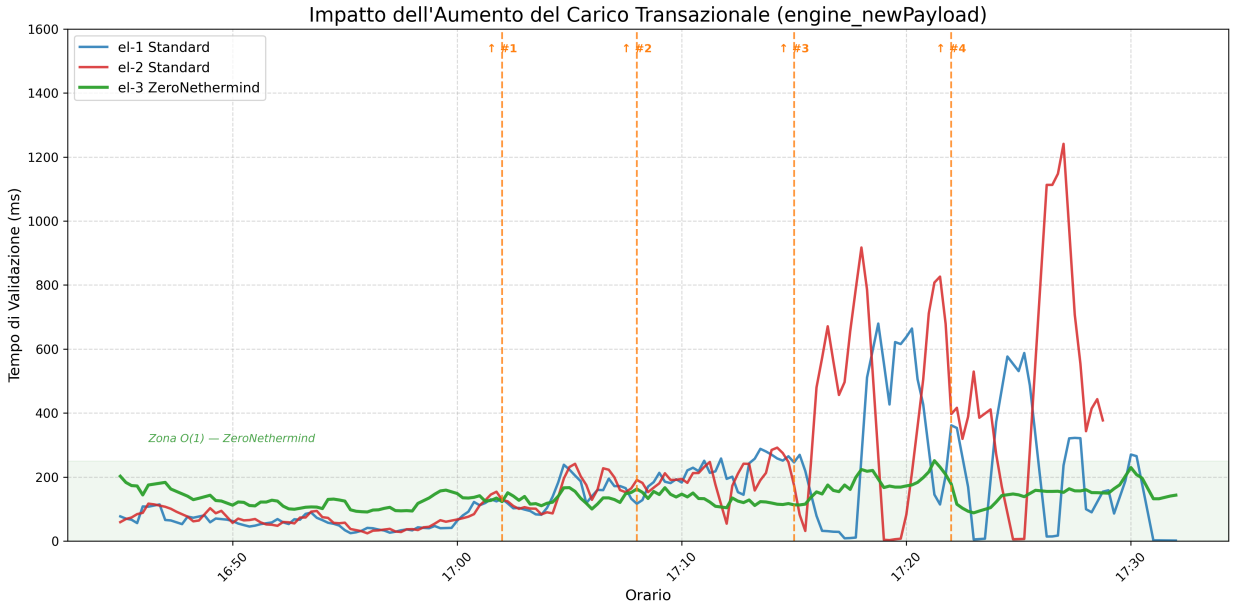

Figura 5.4: Impatto dell'aumento del carico transazionale sul tempo di validazione (`engine_newPayload`).

Il **Grafico 5.4** fornisce la dimensione temporale del fenomeno. Le 4 rette verticali tratteggiate indicano i momenti in cui il throughput degli spammer è stato incrementato manualmente. Per quantificare la divergenza progressiva, la Tabella 5.4 riporta le medie per finestre di 5 minuti.

Finestra	**el-1 (ms)**	**el-3 (ms)**	**Rapporto el-1/el-3**
16:45–16:50	76	153	0.50×
16:50–16:55	60	116	0.52×
16:55–17:00	36	114	0.31×
17:00–17:05	**118**	**132**	**0.90×**
17:05–17:10	172	139	1.24×
17:10–17:15	228	124	1.84×
17:15–17:20	295	168	1.76×
17:20–17:25	293	152	1.93×

Tabella 5.4: Andamento del tempo medio di validazione per finestre di 5 minuti (UTC). La riga in grassetto indica il crossover.

Durante la fase Baseline (prime 3 finestre), il nodo standard è fino a 3× più veloce. Il crossover si manifesta nella finestra 17:00–17:05 UTC, corrispondente al primo incremento di throughput (↑#1), dove i due approcci si equivalgono (0.90×). Da quel punto in poi, il nodo standard diventa progressivamente più lento: dopo il ↑#3, el-1 impiega quasi il doppio del tempo di el-3 (rapporto 1.93×). Le curve dei nodi standard (blu e rossa) divergono verso l'alto con pendenza crescente, un comportamento coerente con la complessità lineare della ri-esecuzione EVM. Al contrario, la curva di ZeroNethermind (verde) resta piatta all'interno della fascia ∼100–200 ms indipendentemente dagli incrementi di carico, confermando un tempo di verifica empiricamente costante, indipendente dal volume di gas consumato.

5.3.2 Footprint delle Risorse

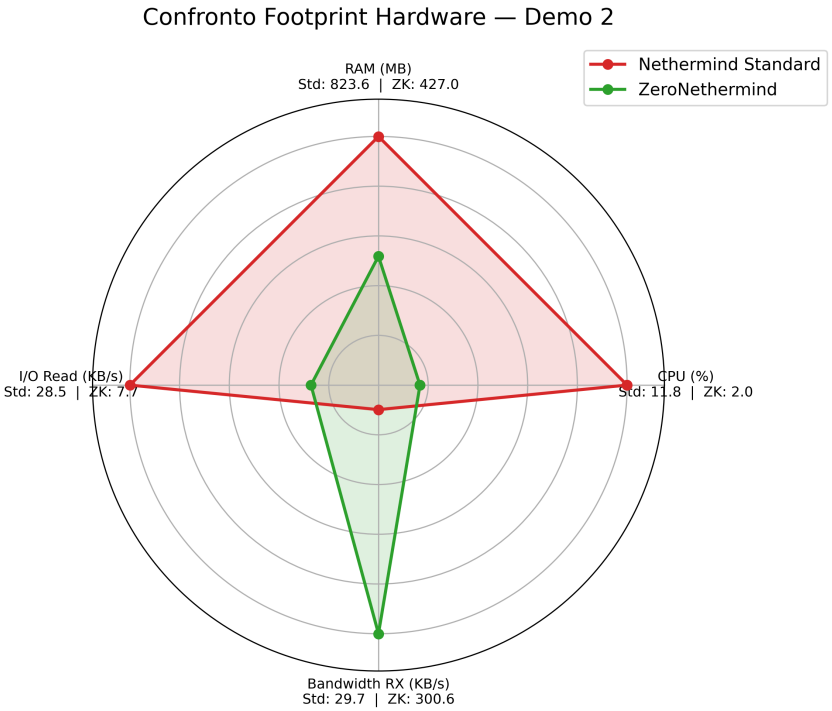

Figura 5.5: Confronto del footprint hardware tra Nethermind Standard (media dei 2 nodi) e ZeroNethermind.

Il **Grafico 5.5** mostra il profilo di risorse dei due approcci architetturali su 4 dimensioni, normalizzate rispetto al valore massimo di ciascun asse. Si evidenziano due profili complementari che riflettono architetture di validazione radicalmente diverse:

- **Nethermind Standard:** Domina su CPU ($\sim$11,8% vs $\sim$2,0%, un fattore $\sim$6$\times$) e RAM ($\sim$823 MB vs $\sim$427 MB), riflettendo il pesante onere computazionale della ri-esecuzione EVM e della gestione in memoria dell'albero di Merkle Patricia (MPT). Si nota inoltre un maggiore I/O Read dal disco ($\sim$28,5 KB/s vs

$\sim$7,7 KB/s) necessario per caricare i rami dello stato richiesti dalle transazioni.

- **ZeroNethermind:** Ribalta il paradigma, spostando il carico predominante sulla Bandwidth in ricezione ($\sim$300,6 KB/s vs $\sim$29,7 KB/s, un fattore $\sim$10$\times$). Le risorse locali (CPU, RAM, Disco) rimangono compresse vicino al centro del grafico.

Questo radar quantifica visivamente un principio architetturale chiave: il passaggio alla validazione crittografica non "azzera" magicamente il costo del network, ma ne trasla la natura, sostituendo risorse locali costose e rigidamente limitate (CPU, IOPS del disco) con una risorsa esterna, elastica e naturalmente scalabile (la larghezza di banda).

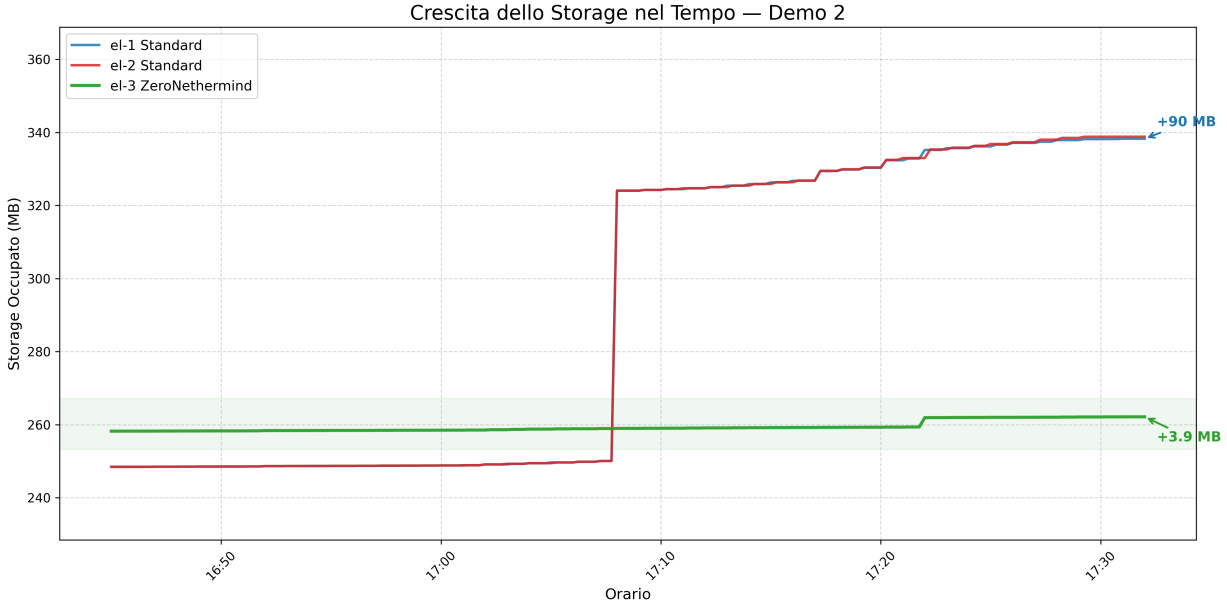

Figura 5.6: Crescita dello storage nel tempo: confronto dell'accumulo di stato su disco.

Il **Grafico 5.6** dimostra empiricamente il paradigma *Zero State* introdotto nel `capitolo 1`. I due nodi standard partono da $\sim$248 MB e raggiungono $\sim$338 MB in $\sim$45 minuti (crescita di $\sim$90 MB ciascuno). ZeroNethermind, partendo da $\sim$258 MB, raggiunge soli $\sim$262 MB, pari al **4,3%** di quella standard. La crescita residua di ZeroNethermind è

attribuibile ai soli metadati operativi del client (log interni, cache del Consensus Layer), non all'accumulo di stato della blockchain.

Questo risultato dimostra che l'eliminazione del database di stato persistente non è solo un vantaggio teorico, ma produce un impatto misurabile e significativo sulla sostenibilità operativa a lungo termine di un nodo validatore.

5.4 Discussione dei Risultati

I risultati delle due campagne sperimentali, pur condotte in contesti operativi distinti, convergono verso un quadro coerente che può essere sintetizzato lungo tre assi tematici.

1. Paradigma computazionale: da $O(N_{gas})$ a $O(1)$. La Demo 1 ha dimostrato che la verifica crittografica FFI richiede mediamente $\sim$122 ms per blocco Mainnet, indipendentemente dalla complessità del blocco stesso (fino a 30M Gas). La Demo 2 ha confermato questa proprietà sotto carico controllato e crescente: mentre i nodi standard vedono il tempo di validazione aumentare di $\sim$3,7$\times$ passando dalla fase Baseline a quella sotto carico (con picchi fino a $\sim$1 500 ms), ZeroNethermind resta nell'intervallo 128–147 ms, mostrando un incremento del solo 15%. Il crossover tra i due approcci avviene già con carichi transazionali moderati (Tabella 5.4), suggerendo che il vantaggio della verifica ZK si manifesta *prima* della saturazione dei blocchi.

2. Spostamento del collo di bottiglia architetturale. Sia la Demo 1 sia la Demo 2 confermano lo spostamento del bottleneck da *I/O-bound* a *bandwidth-bound*. Nella Demo 1, il download delle prove da `ethproofs.org` assorbe il 94,5% del tempo totale di validazione, relegando la verifica FFI al 5,5%. Nella Demo 2, il radar chart (Grafico 5.5) quantifica questo spostamento: il consumo di bandwidth di ZeroNethermind è $\sim$10$\times$ quello dei nodi standard, mentre CPU e

RAM sono ridotti di $\sim$6$\times$ e $\sim$2$\times$ rispettivamente. Questo risultato ha un'implicazione progettuale importante: una volta che le prove saranno incluse direttamente nel payload del blocco (come previsto dalla roadmap di Ethereum), il collo di bottiglia della bandwidth si azzererà, e il tempo di validazione coinciderà con la sola componente FFI ($\sim$120 ms).

3. Stabilità operativa e azzeramento dello State Bloat. Durante la Demo 1, ZeroNethermind ha validato 121 blocchi Mainnet consecutivi in sincronizzazione con il Consensus Layer Lighthouse, senza mai perdere un attestation slot e mantenendo la RAM stabilmente sotto i 400 MB. La Demo 2 ha esteso questa osservazione a 45 minuti di carico sostenuto, confermando che la memoria e la CPU restano piatte anche sotto stress. Il dato forse più rilevante riguarda lo storage: la crescita del disco è pari al 4,3% di quella standard (Grafico 5.6), dimostrando che il paradigma *Zero State* elimina il problema dello *State Bloat* che costituisce uno degli ostacoli principali alla decentralizzazione di Ethereum.

La Tabella 5.5 sintetizza le differenze architetturali e prestazionali emerse dalle due campagne tra un nodo Nethermind Full Node e il proof-of-concept ZeroNethermind.

Parametro	Nethermind Full Node	ZeroNethermind
Modello di Fiducia	Ri-esecuzione locale N-of-N	Verifica matematica 1-of-N
Storage di Stato	$\sim$1 TB (Crescita continua)	$\sim$0 GB (MemDb in RAM)
RAM Richiesta	16–32 GB	$<$500 MB (Plateau)
Impatto CPU	Saturazione multi-core (EVM)	Picchi minimi transitori ($\sim$120 ms)
I/O Disco (IOPS)	Elevato (Accesso casuale MPT)	Nullo
Collo di Bottiglia	Limiti Hardware (I/O-bound)	Latenza di Rete (Bandwidth-bound)
Complessità	$O(N_{gas})$ — Lineare al Gas Limit	$O(1)$ — Costante crittografica
Sincronizzazione	Ore / Giorni (State Sync)	Secondi (Checkpoint Sync)
Servizi Estesi	Full RPC, Block Building	Solo Attestation (Fort Mode)

Tabella 5.5: Sintesi del salto architetturale tra un nodo Stateful e un validatore Stateless.

Capitolo 6

Conclusione

Questo capitolo conclusivo sintetizza i contributi del lavoro svolto, ne analizza le limitazioni ingegneristiche, delinea le direzioni di sviluppo futuro e inquadra il proof-of-concept nella traiettoria evolutiva del protocollo Ethereum.

6.1 Riepilogo

Questa tesi ha dimostrato, attraverso la progettazione, l'implementazione e la validazione sperimentale del proof-of-concept ZeroNethermind, che un client dell'Execution Layer di Ethereum può operare in modalità completamente *stateless*, affidandosi alla verifica matematica delle prove crittografiche Zero-Knowledge anziché alla ri-esecuzione computazionale delle transazioni.

Il PoC ha introdotto tre innovazioni architetturali principali:

1. **Il paradigma "Double Zero":** L'eliminazione simultanea del database di stato locale (Zero State, tramite MemDb) e della ri-esecuzione EVM (Zero Knowledge, tramite verifica crittografica), realizzando per la prima volta un validatore L1 che non richiede né terabyte di storage né potenza di calcolo multi-core.

2. **Il bridge FFI cross-language:** Un'interfaccia di interoperabilità ad alte prestazioni tra il runtime managed C#/.NET di Nethermind e le librerie crittografiche native in Rust, capace di verificare prove generate da 5 zkVM indipendenti (Zisk, OpenVM, Pico, SP1, Airbender) con trasferimento Zero-Copy dei payload binari.
3. **L'architettura a plugin:** L'integrazione non invasiva nel client Nethermind tramite il sistema di plugin, che consente di intercettare e sovrascrivere i gestori dell'Engine API senza modificare il codice core, preservando la compatibilità con gli aggiornamenti futuri del client.

La validazione sperimentale, condotta su due fronti complementari, ha prodotto risultati quantitativi significativi:

- La **Demo 1 (Mainnet)** ha validato 121 blocchi in sincronizzazione con il Consensus Layer Lighthouse. Il tempo medio di verifica crittografica FFI è risultato di soli **122 ms** (mediana 116 ms), rappresentando appena il 5,5% del tempo totale, con il restante 94,5% dominato dal download delle prove. La RAM si è mantenuta stabilmente sotto i 400 MB e l'I/O su disco si è azzerato dopo la fase di avvio.
- La **Demo 2 (devnet Kurtosis)** ha confermato empiricamente la proprietà $O(1)$ della verifica crittografica sotto carico crescente: mentre i nodi standard hanno visto il tempo di validazione aumentare di $\sim$3,7$\times$ con picchi fino a $\sim$1 500 ms, ZeroNethermind è rimasto nell'intervallo 128–147 ms (incremento del solo 15%). L'utilizzo di CPU è risultato $\sim$6$\times$ inferiore, la RAM $\sim$2$\times$ inferiore, e la crescita del disco è stata pari al **4,3%** di quella standard, dimostrando una riduzione del 95,7% nell'accumulo locale di stato.

Questi risultati rispondono positivamente alle due domande di ricerca poste in fase di progettazione: (i) la validazione per verifica crittografi-

ca è *operativamente fattibile* all'interno di un client production-ready, operando entro i vincoli temporali dello slot di 12 secondi; (ii) il passaggio dalla ri-esecuzione alla verifica ZK produce un *impatto misurabile e significativo* sul profilo di risorse, spostando il collo di bottiglia dall'I/O su disco alla latenza di rete.

Il supporto nativo a 5 zkVM indipendenti costituisce inoltre una prima forma di *multi-prover diversity*: qualora una singola zkVM presentasse un bug nei circuiti aritmetici, la logica di majority voting distribuirebbe il rischio su molteplici implementazioni indipendenti, offrendo una mitigazione strutturale contro le vulnerabilità crittografiche.

6.2 Limitazioni

Nonostante i risultati incoraggianti, il PoC presenta tre limitazioni ingegneristiche che ne impediscono l'adozione diretta in un ambiente di produzione.

1. La latenza dell'oracolo HTTP esterno. ZeroNethermind dipende dalle API REST di `ethproofs.org` per il recupero delle prove crittografiche. Questa dipendenza da un oracolo HTTP centralizzato introduce una latenza di rete inaccettabile per i vincoli di timing del protocollo: nella Demo 1, il download delle prove ha assorbito il 94,5% del tempo totale di validazione, lasciando alla verifica FFI effettiva soli $\sim$120 ms. In uno scenario ideale, dove le prove fossero immediatamente disponibili, il tempo di validazione crollerebbe di un ordine di grandezza. Inoltre, la dipendenza da un singolo endpoint centralizzato rappresenta un *single point of failure*, incompatibile con i principi di decentralizzazione del protocollo.

2. Il workaround del protocollo `SYNCING`. Per gestire il caso in cui la prova per un blocco non sia ancora disponibile al momento della ricezione del `newPayload`, il PoC adotta una strategia di *Active-Wait*: il

gestore custom risponde al CL con lo status `SYNCING` e blocca il thread RPC in un ciclo di polling finché la prova non diventa disponibile su ethproofs.org. Questo approccio, sebbene funzionale per la dimostrazione, non è scalabile in produzione: il blocco del thread introduce una latenza non deterministica e potrebbe causare timeout del Consensus Layer in caso di ritardi prolungati nella generazione delle prove.

3. Il gap della Data Availability. Un client stateless, per definizione, non possiede il database di stato della blockchain. Questo implica l'impossibilità di rispondere alle query RPC standard degli utenti, come `eth_getBalance`, `eth_call` o `eth_getStorageAt`, che richiedono l'accesso diretto allo stato corrente o storico. Il PoC è pertanto strettamente definito come un nodo **attester-only** ("Fort Mode"): valida la catena con garanzie crittografiche complete, ma non è in grado di fungere da endpoint RPC *full-service* per applicazioni decentralizzate o wallet. Questa è una scelta progettuale intenzionale che sacrifica la funzionalità RPC estesa in favore dell'estrema efficienza delle risorse.

6.3 Lavori Futuri

Le limitazioni descritte nella sezione precedente non sono intrinseche al paradigma della validazione stateless, ma riflettono vincoli dello stato attuale dell'ecosistema. La roadmap di Ethereum prevede diverse evoluzioni che risolverebbero ciascuna di esse.

1. ePBS e l'espansione della finestra di proving. Il protocollo attuale concede ai validatori circa 4 secondi per ricevere e validare un blocco all'interno dello slot di 12 secondi. L'adozione dell'*Enshrined Proposer-Builder Separation* (ePBS, EIP-7732)[9], prevista per l'hard-fork Glamsterdam (stimato per la seconda metà del 2026), espanderebbe la finestra temporale disponibile per gli attestatori da ∼2 secondi a ∼6–9 secondi[14]. Questo incremento è cruciale per la fattibilità del

proving in tempo reale: con una finestra di 9 secondi, anche i prover attuali potrebbero generare prove entro i vincoli temporali dello slot.

2. Gossip P2P delle prove e inclusione nel protocollo. La dipendenza da un oracolo HTTP centralizzato verrebbe eliminata attraverso due evoluzioni progressive:

- **P2P Proof Gossip:** Le prove verrebbero propagate attraverso la rete peer-to-peer del Consensus Layer, analogamente a come avviene per i blocchi e le attestazioni.
- **Inclusione nel protocollo (EIP-8025):** La proposta EIP-8025[10] introduce due nuove modalità per i validatori: *zkEVM Proof generating* e *Stateless validation*. I nodi generatori propagherebbero le prove su subnet dedicate del CL, mentre i nodi stateless le consumerebbero senza necessità di alcun endpoint esterno. Essendo le prove opzionali, l'EIP consente una transizione graduale senza impatti sul consenso, fungendo da banco di prova per un futuro obbligo a livello di protocollo.

3. Data Availability tramite Portal Network. Il gap della Data Availability del nodo attester-only potrebbe essere colmato dall'integrazione con la Portal Network[11], una rete decentralizzata di tipo DHT progettata specificamente per servire dati storici e di stato a client leggeri. La Portal Network distribuisce lo stato globale in frammenti tra i nodi partecipanti, consentendo query RPC come `eth_getBalance` senza che alcun singolo nodo debba mantenere l'intero database. L'integrazione di un client ZeroNethermind con un nodo Portal consentirebbe di combinare la validazione stateless con la capacità di rispondere a query di stato, superando la limitazione attester-only senza reintrodurre i requisiti di storage di un full node.

4. Deployment su dispositivi mobili. La combinazione della verifica $O(1)$ ($\sim$120 ms), del consumo di RAM inferiore ai 500 MB e dell'as-

senza di I/O su disco rende concepibile il deployment di un validatore stateless su dispositivi mobili. Il bridge FFI in Rust è già oggi compilabile per architetture ARM64, e l'impronta di risorse misurata nella Demo 2 è compatibile con smartphone di fascia media. Un validatore mobile contribuirebbe alla decentralizzazione della rete in modo senza precedenti, abbattendo la barriera d'ingresso alla partecipazione attiva al consenso.

6.4 Considerazioni finali

Lo spostamento dal paradigma $O(N_{gas})$ al paradigma $O(1)$ modifica la natura del trade-off nel Blockchain Trilemma: anziché sacrificare la scalabilità per preservare la decentralizzazione, il peso computazionale viene delegato a infrastrutture specializzate (i prover), distribuendo la verifica su una base di validatori più ampia e accessibile. Il requisito hardware per un validatore passerebbe da 2 TB di SSD NVMe e 32 GB di RAM a un dispositivo con poche centinaia di megabyte di memoria e una connessione a banda larga. Con l'adozione parallela di PeerDAS per la disponibilità dei dati e delle zkEVM per la validazione, Ethereum potrebbe raggiungere simultaneamente un throughput elevato, un consenso decentralizzato e una validazione universalmente accessibile.

Questa tesi ha dimostrato che il componente Fort Mode di questa visione è tecnicamente realizzabile oggi, all'interno di un client di produzione, con prestazioni compatibili con i vincoli temporali del protocollo. Il lavoro rimanente risiede nella maturazione dell'infrastruttura di proving, nella riduzione della latenza di generazione delle prove e nella loro integrazione nativa nel layer di consenso.

Il principio architetturale alla base di questo lavoro è la trasformazione della validazione da un problema di potenza di calcolo a un problema di matematica: la crittografia Zero-Knowledge introduce un'asimmetria computazionale tale per cui verificare un blocco con diecimila

transazioni richiede le medesime risorse di verificare un blocco con una sola transazione. Quando questa proprietà sarà integrata a livello di protocollo, i requisiti hardware cesseranno di costituire una barriera all'ingresso e la partecipazione al consenso potrà estendersi a una classe di dispositivi oggi esclusa dalla validazione attiva.

Riferimenti bibliografici

[1] Jordi Baylina et al. ZisK: A fast zkVM architecture. [Repository GitHub], 2025.

[2] Vitalik Buterin. Possible futures of the Ethereum protocol, part 4: The Verge. [Articolo online], 2024.

[3] Stefanos Chaliasos, Denis Firsov, and Benjamin Livshits. Towards a formal foundation for blockchain zk rollups. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 2714–2728, 2025.

[4] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. Sok: Blockchain light clients. In *International Conference on Financial Cryptography and Data Security*, pages 615–641. Springer, 2022.

[5] Justin Drake. Keynote + demo: zkEVM attesting. [Video], 2025. Ethereum Foundation, Devconnect.

[6] Justin Drake. Lean Ethereum. [Articolo online], 2025. Ethereum Foundation Blog.

[7] ETH Applied Cryptography Team. Ethereum zkEVM book. [Sito web], 2026.

[8] Ethereum Community. EIP-4762: Statelessness gas cost changes. [Specifica online], 2022.

[9] Ethereum Community. EIP-7732: Enshrined proposer-builder separation (ePBS). [Specifica online], 2024.

[10] Ethereum Community. EIP-8025: Optional execution proofs. [Specifica online], 2026.

[11] Ethereum Community. Portal network. [Documentazione online], 2026.

[12] Ethereum Foundation. Ethproofs — learn. [Sito web], 2023.

[13] Ethereum Foundation. L1 strawmap — Ethereum draft roadmap. [Sito web], 2026.

[14] Ethereum Foundation zkEVM Team. L1-zkevm roadmap 2026: Integrating zkevm proofs into ethereum's core protocol. [Specifica online], 2026.

[15] Google. cAdvisor: Container resource usage and performance analysis. [Repository GitHub], 2021.

[16] Julian. An Ethereum prover market proposal. [Specifica online], 2025.

[17] Kurtosis Technologies. Kurtosis — ethereum network testing. [Sito web], 2024.

[18] L2BEAT Team. L2BEAT stages: A framework to evaluate rollup decentralization. [Sito web], 2024.

[19] Nethermind. Database — Nethermind documentation. [Documentazione online], 2026.

[20] Nethermind. System requirements — Nethermind documentation. [Documentazione online], 2026.

[21] Orochi Network. Blockchain trilemma: Balancing decentralization, security, and scalability. [Articolo online], 2025.

[22] Prometheus Authors. Prometheus — monitoring system and time series database. [Sito web], 2025.

[23] Giovanni Quattrocchi, Filippo Scaramuzza, and Damian A Tamburri. The blockchain trilemma: an evaluation framework. *IEEE Software*, 41(6):101–110, 2024.

[24] Stateless Consensus. Ethereum stateless book. [Sito web], 2025.

[25] Succinct Labs. SP1: A performant, 100% open-source zkVM. [Repository GitHub], 2024.

[26] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends in Privacy and Security*, 4(2–4):117–660, 2022.

[27] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.