zkdeal: A Bounded Execution Mode for Ethereum

Temporary and rolling execution rooms with validity checkpoints

Oleg Iakushkin
Founder zkDeal
contact@zkdeal.org

August 2026

Abstract. Ethereum supports atomic transactions and persistent rollups, but some workflows need several ordered steps without a permanent chain. A zkdeal room is a temporary EVM instance governed by fixed execution and closure rules. Its state becomes canonical only when Ethereum accepts a validity checkpoint. Users sign intents, an operator orders them, and a proof re-executes the resulting blocks. A public journal binds continuity, data, authorization, accounting, withdrawals, and closure. The proof establishes correct execution, while admission, fair ordering, data availability, and Ethereum finality remain separate concerns.

Prototype runs execute 3–10 local blocks in 1.5–5 seconds at a 0.5-second cadence. Recorded B200 tests returned Ethereum-verifiable Groth16 proofs, but the small sample lacks stage, cycle, inclusion, finality, and L1-gas measurements. The contribution is a protocol composition and prototype, not a new cryptographic primitive or a cost claim.

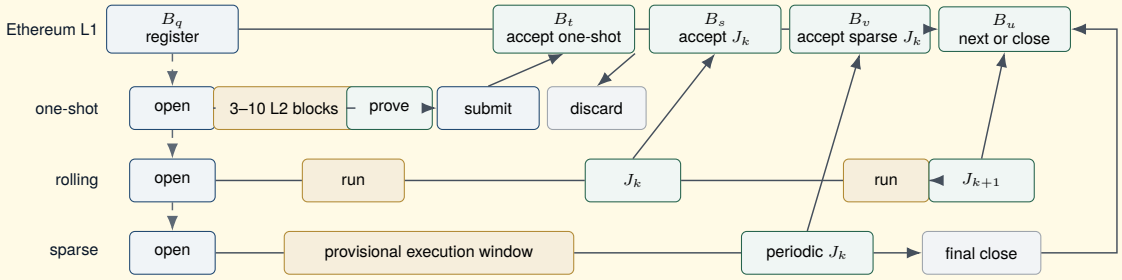

Figure 1: Room execution against Ethereum block time. Horizontal edges represent local processing. Upward edges submit journals. Later room state becomes canonical only when the connected L1 block accepts the submitted journal. Spacing is schematic.

1 Problem and thesis

An Ethereum transaction gives one atomic state transition. A persistent rollup supplies continuing execution, but also requires a sequencer, prover, bridge, data service, and long-lived operations [1]. State channels support temporary interaction, usually through participant-signed updates and an on-chain dispute path [2].

Some workflows sit between these choices. Examples include delivery-versus-payment, an asynchronous vault round, sealed market clearing, and payment authorization followed by fulfilment. They need several ordered steps, but they do not always need a permanent chain.

A room provides a temporary EVM environment for one such workflow. Local execution is provisional. Its result becomes canonical only after Ethereum accepts a checkpoint journal and validity proof.

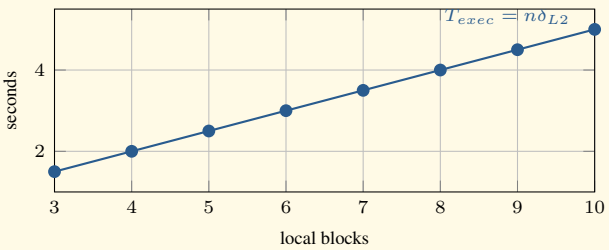

Figure 2: IMPLEMENTATION Local execution window. At the configured 0.5-second cadence, 3–10 blocks take 1.5–5 seconds. Proof and settlement are separate.

1.1 Premise

Persistent L2 infrastructure can outlive a bounded task. A one-shot room opens, runs the workflow, proves a closing checkpoint, settles on Ethereum, and decommissions its runtime. Rolling and sparse rooms reuse the same acceptance rule with different checkpoint schedules.

This design rolls up a workflow, not an application estate. No cost or latency advantage follows without measurements for proving, data publication, and L1 acceptance.

1.2 Design requirements

The protocol has five requirements:

1. **Authenticated opening.** Ethereum binds the template, initial root, participants, and accounting frame before room execution.
2. **Certified execution.** Every checkpoint follows one deterministic EVM transition policy.
3. **Continuity.** Every accepted checkpoint extends the latest L1-accepted root.
4. **Conservation.** Custodied assets reconcile with liabilities and withdrawal claims.
5. **Recovery.** Canonical data supports reconstruction, claims, and safe runtime disposal.

Claim labels. MEASURED is recorded; IMPLEMENTATION is observed in code or tests; MODELED uses stated assumptions; DESIGN TARGET requires future qualification.

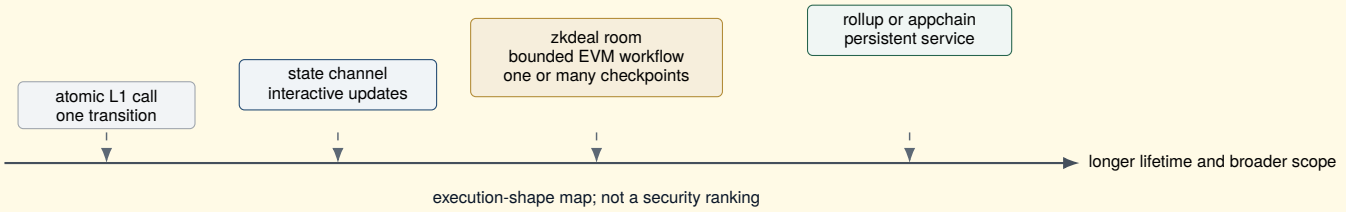

Figure 3: zkdeal occupies the lifecycle gap between one-call atomicity and persistent L2 infrastructure.

2 Protocol contribution

Temporary runtimes, off-chain ordering, EVM execution, validity proofs, storage proofs, and L1 custody are established techniques. zkdeal combines them around one Ethereum checkpoint for a bounded workflow.

The claimed combination has four parts:

(i) **Authenticated room.** Ethereum binds the opening state and the rules that govern execution and closure.

(ii) **Intent-authorized order.** Users sign actions, the operator orders them, and the proof certifies the resulting EVM transition.

(iii) **Ethereum validity journal.** The checkpoint binds execution, public data, authorization, and accounting to one proof; fig. 7 and table 5 give the field classes.

(iv) **Explicit terminal state.** One-shot, rolling, and sparse rooms use one acceptance rule, while an accepted close preserves custody and claims after the runtime ends.

This is a systems-composition claim, not a new cryptographic primitive. Section 10 narrows the claim against channels, Ephemeral Rollups, Plasma, validium, coprocessors, and rollup variants.

3 System model and notation

3.1 Actors and trust

Users authorize intents. The operator chooses their order, and the executor computes EVM blocks. A prover certifies the transition. The room manager contract verifies checkpoints, updates accounting, and records the canonical room state on Ethereum.

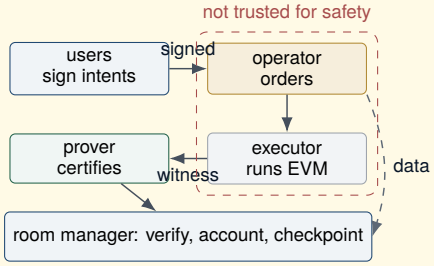

Figure 4: The proof and room manager protect state safety. Ordering, liveness, and provisional data still depend on the operating mode.

The operator need not be trusted for execution safety, but it controls admission and order unless the execution policy constrains them. Proof soundness removes the need to trust the prover's result; the prover may still learn its witness. Recovery also requires the canonical data named by the accepted journal.

Guarantee boundary.

Proof and L1 contract establish	They do not establish
intent authorization; deterministic execution; checkpoint continuity; public accounting; closure rules	admission; fair ordering; censorship resistance; witness secrecy; availability of provisional sparse state; L1 inclusion or finality

3.2 Notation

Table 1: Core notation.

Symbol	Meaning
D, r, e	L1 chain and room-manager domain, room identifier, immutable instance epoch
T	reusable cold template
t	registered cold-template commitment
P, A, Γ	execution, authorization, and checkpoint policies
χ_P, χ_{proof}	commitments to P and the proof program
S_j	complete room state after step j
ι_i	signed intent
a_i	signer recovered from ι_i
π	operator-selected intent order
B_b	ordered room block at height b
J_k, c_k	journal for checkpoint k and its hash
ρ_k	state root accepted after checkpoint k
Π_k, vk	validity proof and verification key
δ_{L2}	configured local block cadence
Δ_c	interval between checkpoints

The deployment domain D contains the L1 chain identifier and room-manager address. Bytes are length-delimited before hashing. Addresses and integers use Ethereum ABI encoding unless a field states otherwise. Every signature and room-specific commitment binds its message type and (D, r, e). The reusable template binds its type but not a room instance.

3.3 Room

Definition 1 (room). A room is

$$R = (D, r, e, T, P, A, \Gamma, S_0), \tag{1}$$

where S_0 is its authenticated opening state.

The template commits to the opening root, execution policy, and proof program:

$$t = \text{Keccak256}(\text{ABI}(\mathsf{RoomTemplate}, \rho_0, \chi_P, \chi_{proof})). \tag{2}$$

The room manager registers t before it admits participant assets.

3.4 Signed intent and order

An intent contains a signer, nonce, deadline, call payload, value, and fee bounds. Let $\mathcal{H}$ be the domain-separated signing hash. The signature rule is

$$\begin{aligned} \iota_i &= (m_i, \sigma_i), \\ \text{Recover}(\mathcal{H}(D, r, e, \mathsf{SignedIntent}, m_i), \sigma_i) &= a_i. \end{aligned} \tag{3}$$

The operator chooses a permutation π over admitted intents. The proof checks each intent at its selected position under policy P.

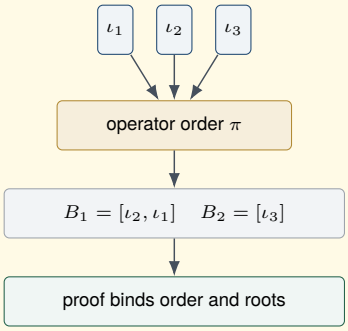

Figure 5: Signed intents become an ordered batch whose execution can be proved.

4 Protocol lifecycle and modes

All rooms use the lifecycle in fig. 1. They share one transition rule and one Ethereum acceptance predicate. Their checkpoint schedules differ.

4.1 Cold-template proof and opening

A cold-template proof certifies construction of an allowed opening state. It binds code, initialized storage, execution limits, participant roots, the verifier, and the initial accounting frame. The proof prepares a reusable template; it does not execute later user actions.

Opening registers one room instance against that template. The registration binds (D, r, e), the opening root, all three policies, the verifier, participant limits, custody, and closing conditions. The room manager requires the service bond before participant assets enter the room.

4.2 Execute

Users submit signed intents at the admission boundary. The operator checks the signature, nonce, domain, resource bounds, and available value before admitting an intent. Admitted intents remain provisional until an L1-accepted checkpoint includes them.

The production rule is drop and continue. The operator scans a bounded queue, records why an envelope is structurally invalid, drops it, and continues. A bad signature or domain is structural invalidity; an ordinary EVM revert is not. A reverting intent remains in its selected block and consumes its nonce and gas.

Each block is atomic. If block construction fails, the executor restores state, headers, address discovery, and queue effects to their pre-block values.

4.3 Checkpoint chain and modes

Checkpoint k begins at the latest L1-accepted root:

$$J_k.\mathit{preRoot} = \rho_{k-1}, \qquad J_k.\mathit{index} = k. \tag{4}$$

Its first L2 block follows the accepted height, and its block range is contiguous. Ethereum orders and finalizes checkpoint transactions; it does not choose the room's intent order.

Every checkpoint must extend the latest L1-accepted root. An operator may prove several local blocks at once, but it may not checkpoint a branch that bypasses that root. The post-state becomes ρ_k only after L1 acceptance.

The three modes are:

1. **One-shot.** One checkpoint proves the workflow and closes the room.
2. **Rolling.** Successive checkpoints extend the room without a fixed short lifetime.
3. **Sparse.** A longer provisional window precedes a periodic or closing checkpoint.

Sparse operation changes the rollback and availability boundary; section 7.5 states that boundary once.

4.4 Close and decommission

A closing journal proves the terminal allocation and sets the close flag. The room manager rejects later checkpoints, while claims and withdrawals remain usable.

After L1 acceptance, the operator may decommission executor instances and delete caches and prover workspaces. Canonical batch data, custody, liabilities, withdrawal roots, and accepted commitments remain available under the recovery rules in section 7.4.

5 Certified execution and proof

5.1 Transition rule

Let E_j contain the constrained block number, timestamp, gas limit, chain identifier, and environment fields. For execution policy P, one selected intent advances the state by

$$S_{j+1} = \Upsilon_P(S_j, \iota_{\pi(j)}, E_j). \tag{5}$$

Policy P bounds block and transaction counts, gas, memory, code, contract creation, call edges, storage namespaces, and authenticated imports. The proof rejects any transition outside those bounds.

Production authorization uses the non-broadcast SignedIntent format from eq. (3). It binds the user instruction to the L1 chain, room manager, deployment, room, epoch, nonce, deadline, and payload.

The inner EVM transaction is witness data rather than a reusable authorization for another chain. The registry must never reuse the tuple (D, r, e).

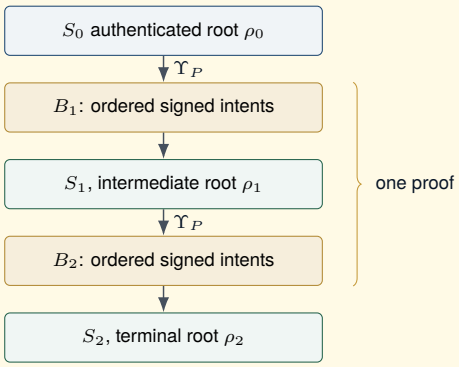

Figure 6: One proof folds a contiguous block sequence and binds every intermediate state root.

The proved block list is nonempty and no larger than the policy maximum. A policy may permit a one-block rolling or closing checkpoint.

Let τ_Q be the latest L2 timestamp accepted in room state Q. Let τ_{L1} be the checkpoint's L1 acceptance time. For batch timestamps $\tau_1, \ldots, \tau_z$ and earliest deadline d_{min}, require

$$\tau_Q \le \tau_1 \le \cdots \le \tau_z \le \tau_{L1} \le d_{min}. \tag{6}$$

The proof checks the L2 sequence. The room manager checks the L1 bounds, so a batch cannot assign an earlier timestamp to an expired intent.

5.2 Authorization modes

Both authorization modes use the same transition proof:

- **Journal-unanimous mode.** Every active approver signs the exact checkpoint journal.
- **Per-intent mode.** Each caller signs its own intent, and the proof verifies those signatures inside the selected batch.

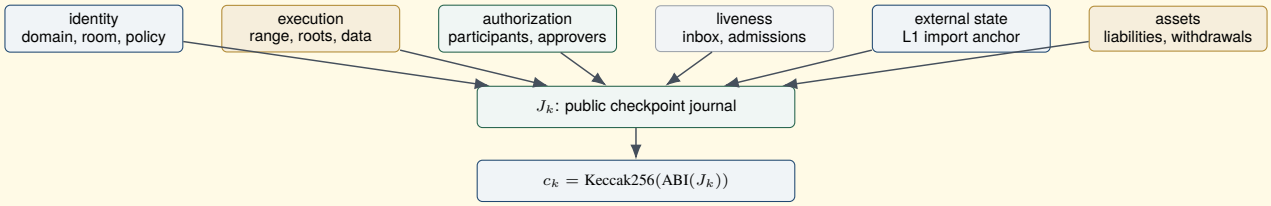

Figure 7: The journal binds identity, execution, authorization, liveness, imports, and asset accounting into one commitment.

5.3 Checkpoint journal

The journal J_k is the public statement for checkpoint k. It groups settlement fields into the six classes shown in fig. 7; table 5 is normative.

Ethereum computes

$$c_k = \text{Keccak256}(\text{ABI}(J_k)). \tag{7}$$

The proof uses c_k as its public input. Canonical, length-prefixed batch bytes are also public when the mode requires calldata publication, and their hash must match the journal.

The journal binds an L1 inclusion range $[b_{min}, b_{max}]$. The room manager accepts only when the current block lies in that range. Changing the submission block inside the range therefore needs no new proof.

5.4 Witness and validity proof

The private witness contains the pre-state and the data needed to derive each journal class: execution blocks, authorization records, liveness records, authenticated imports, and asset transitions. The public inputs are the journal commitment and any canonical bytes required by the data policy.

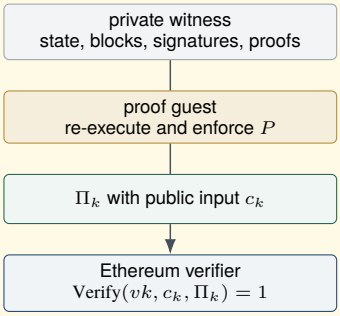

Figure 8: The guest re-executes the ordered batch and produces an Ethereum-verifiable proof for the journal commitment.

Validity is

$$\text{Verify}(vk, c_k, \Pi_k) = 1. \tag{8}$$

This equation certifies the selected transition and its public journal commitment. The broader guarantee boundary remains the one in section 3.1.

6 Ethereum acceptance and accounting

6.1 Acceptance predicate

The room manager accepts a checkpoint only when it matches the current room state and its proved inclusion range. Define

$$\begin{aligned}\text{Accept}_{L1}(J_k, \Pi_k) = \ &\mathsf{ProofValid} \wedge \mathsf{ChainContinuous} \\ &\wedge \mathsf{Authorized} \wedge \mathsf{Accounted} \\ &\wedge \mathsf{BeforeDeadline} \wedge \mathsf{DataAvailable}.\end{aligned} \tag{9}$$

Each condition fails closed. A valid proof for a stale root, wrong room, expired range, incomplete accounting frame, or mismatched batch data is rejected. After acceptance, the room manager advances all roots and cursors atomically. Closing prevents later checkpoints but preserves claims and withdrawals.

6.2 Asset entry, liabilities, and withdrawal

Funds enter through L1 custody, a deposit inbox, or balances authenticated in the opening state. The room may spend them only after the proof binds their amount and owner to its accounting state.

For each base asset a, the accounting frame has four buckets: pending deposits p_a, room-controlled assets c_a, claimable withdrawals q_a, and amounts already paid d_a. Write

$$L_a = (p_a, c_a, q_a, d_a). \tag{10}$$

A transition must satisfy

$$\mathit{custody}_a = p_a + c_a + q_a, \qquad d'_a \geq d_a. \tag{11}$$

Execution may move backed value among the first three buckets. An accepted withdrawal moves claimable value to paid value and releases the corresponding L1 asset. Room-native shares use a separate supply invariant; they are claims on underlying custody and are not counted again as base assets.

A closing transition must allocate all controlled assets. Its execution policy must define a recipient, refund, or permitted burn for rounding dust in every reachable terminal state. This makes closure feasibility an application-policy obligation.

Physical delivery, bank settlement, and external exchange balances remain application obligations unless an authenticated adapter brings them into the proof.

6.3 Authenticated L1 imports

A room may read selected Ethereum state through an authenticated import. The witness carries account and storage proofs rooted in a confirmed L1 header [3].

The journal binds the source block, header and state roots, adapter, allowed keys, payload, and replay cursor. The execution policy limits which contracts and storage namespaces the adapter may expose.

6.4 Admission and forced outcomes

An admission acknowledgment records when the operator accepted responsibility for an intent. Two forms provide different guarantees:

Admission guarantees.

Form	Promise	Required commitment
Classification	succeeded, reverted, or rejected by a stated boundary	intent and terminal-record boundary
Success or position	specified success or ordered position	sealed prefix, batch index, and position

A classification acknowledgment reserves no position, so later intents may precede it. A stronger promise freezes the named prefix; otherwise later ordering could invalidate the promise.

A forced queue lets a user bypass normal admission after a policy delay. An omission proof may slash the service bond

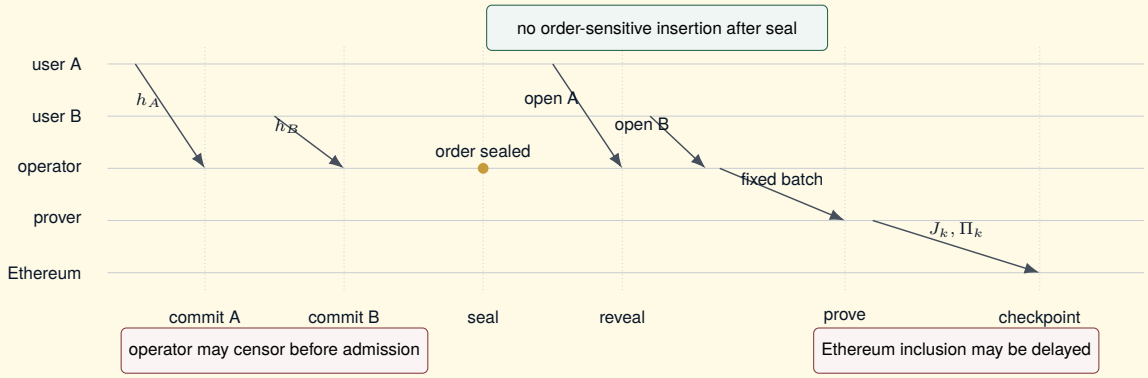

Figure 9: Commit–reveal blocks reveal-reactive insertion after the order is sealed. Pre-admission censorship and delayed Ethereum inclusion remain possible.

when the operator misses its committed boundary. These are liveness controls; ordering rules remain part of the execution policy.

6.5 Data requirement

The DataAvailable check requires the canonical bytes named by the room's data policy. Rollup-like checkpoints publish them on Ethereum. Recovery and the weaker sparse boundary are specified in sections 7.4 and 7.5.

7 Security, availability, and recovery

7.1 Safety invariants

Assume sound proofs, collision-resistant commitments, unforgeable signatures, correct Ethereum execution, and available canonical data. Every L1-accepted checkpoint then satisfies these invariants:

1. room identity and policy remain fixed within the checkpoint chain;
2. the pre-state root equals the latest accepted post-state root;
3. every caller authorized its exact proved intent;
4. intermediate roots match the ordered canonical batch;
5. liabilities and withdrawals reconcile with custody; and
6. closure preserves every outstanding claim.

7.2 Ordering and MEV

Maximal extractable value includes gains from inclusion and ordering choices [4]. A room narrows this surface when its execution policy hides information or constrains the admissible order.

The prototype uses first-in, first-out order after observation. Network arrival is not a fair-order rule. Applications that need fairness must encode a checkable rule, such as commit–reveal, a deterministic auction, or signed price and time bounds.

Calling a plaintext intent "sealed" does not hide it from the operator. Commit–reveal hides the preimage until reveal, while stronger operator privacy requires user-held reveals, threshold encryption, or another confidentiality system.

7.3 Market exposure

A shorter workflow window reduces modeled spot movement only slightly at second-scale horizons. MODELED At 80% annualized volatility, the model in eq. (17) gives 3.59 basis points over ten seconds and 3.94 over twelve seconds. The main benefit is therefore multi-step atomicity and less public interleaving, not a sub-slot spot-price claim.

7.4 Data availability and recovery

An accepted checkpoint is recoverable when its canonical batch data, cold template, journal, and imports remain available. A replacement executor or prover can reconstruct the accepted state from those inputs.

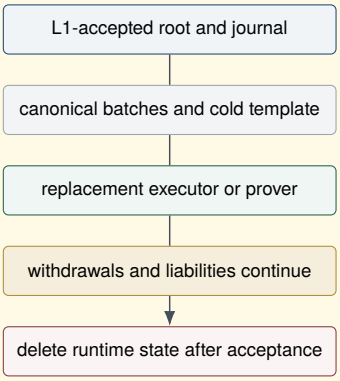

Figure 10: Runtime disposal preserves the accepted data, custody, and claims needed for recovery.

A rollup-grade deployment also needs a timeout exit for claims in the accepted accounting state and a refund path for unused L1 deposits. No exit can promise an outcome that never reached an accepted checkpoint. Private keys and unpublished application secrets remain outside protocol recovery.

7.5 Sparse-mode boundaries

Sparse mode treats uncheckpointed execution as operator-served provisional state. The protocol provides no canonical data availability, forced progress, or accepted proof for that interval. If the operator disappears, users return to the latest L1-accepted root unless the exact local data was published elsewhere.

Longer checkpoint intervals therefore lengthen the rollback window and increase dependence on operator availability. They may also increase each proof's work. Sparse mode amortizes fixed submission overhead, but it does not by itself reduce proof work, calldata, or total cost.

8 Latency, UX status, and measurements

8.1 Four intervals and finality

Room execution exposes four operational intervals before L1 inclusion. Ethereum finality is a later event.

The four user-visible statuses are:

- **Provisional.** The room accepted the local result; rollback remains possible.
- **Proof-ready.** An Ethereum-verifiable proof exists, but Ethereum has not accepted it.
- **Submitted.** The checkpoint transaction is waiting for inclusion.
- **L1-included.** The room manager accepted the checkpoint; consensus finality follows separately.

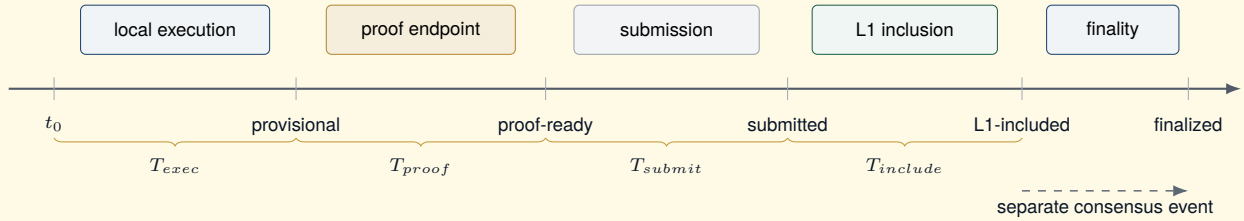

Figure 11: Local completion, proof readiness, submission, L1 inclusion, and finality are distinct events. The recorded proof measurements stop at proof readiness.

Table 2: MEASURED Recorded B200 timing, room gas, and paired H100 comparison. Room gas excludes cold preparation and L1 settlement.

Demo	Cold-template proof	Final Groth16 endpoint	Room-EVM gas	B200 faster than H100	N_{obs}
Auction	5.130 s	8.917 s	113,808	21.7–30.2%	1
Shop	5.139 s	8.924 s	113,803	23.5–25.5%	1
ERC-7540	5.968 s	29.641 s	781,992	20.2–21.3%	1
DvP	5.138 s	8.917 s	69,524	22.5–23.2%	1
ERC-4626	5.158 s	8.910 s	312,412	21.4–22.4%	1
AMM	5.398 s	8.917 s	824,865	18.7–21.5%	1
Cards	7.309 s	28.866, 25.596, 26.426, 25.608 s	480,575	18.1–23.4%	4

A tap-to-pay interface may show the provisional status immediately when the merchant accepts its rollback and operator-availability risk. Irreversible delivery can instead require the proof-ready, L1-included, or finalized status chosen by the application.

The page-one plot defines $T_{exec}(n) = n\delta_{L2}$ for the configured cadence. Let $T_{proof}(n)$ be the proof-endpoint latency. It includes composition, recursion, identity work, Groth16 wrapping, self-verification, serialization, and host overhead. Here identity work means RISC Zero's identity_p254 recursion step, which prepares a BN254-friendly proof before Groth16 wrapping [5]. Then

$$T_{ready}(n) = T_{exec}(n) + T_{proof}(n). \tag{12}$$

MEASURED The recorded timer starts before the proof request and stops after the endpoint returns a final Groth16 proof suitable for Ethereum verification. The timing includes wrapping, but the record does not retain a separate wrap time.

Adding a nominal two-block cadence to one 8.917-second endpoint observation gives 9.917 seconds. That value is arithmetic, not a jointly timed run, percentile, submission result, or inclusion result. We do not claim completion within one Ethereum slot.

8.2 Recorded B200 observations

MEASURED Each cold-template value records reusable opening-state preparation. Each checkpoint value records the proof endpoint through its final Ethereum-verifiable Groth16 proof. The symbol N_{obs} counts recorded endpoint calls. The table reports individual demonstrations, not production percentiles.

The record did not retain Groth16 wrap splits, zkVM cycles, accelerator utilization, or L1 gas. Five workloads cluster between 8.910 and 8.924 seconds despite a twelvefold room-gas range. This suggests a proof-system granularity or recursion floor, not equal proof cost.

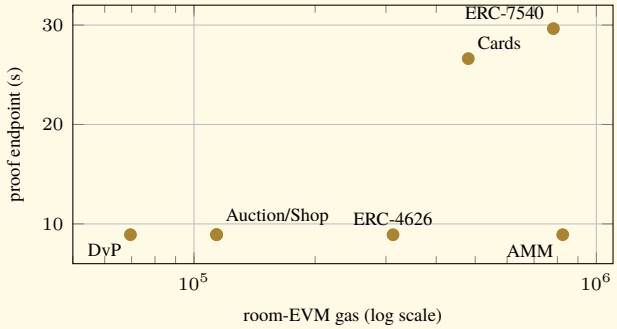

Figure 12: MEASURED EVM gas is not a proof-complexity proxy. Similar gas can produce very different endpoint times.

The recorded B200 advantage over H100 is modest for a hardware-generation comparison. Without cycle, memory, queue, and utilization telemetry, the record cannot identify the bottleneck.

8.3 Checkpoint resource model

Let Δ_c be the checkpoint interval, λ the arrival rate, w the work per arrival, and W_0 fixed checkpoint work. For checkpoint q,

$$W_q = W_0 + \lambda\Delta_c w, \qquad T_{proof,q} = F(W_q), \tag{13}$$

where F is the unmeasured proof-latency function.

Let T_{cold} be reusable template-proof time, m the checkpoint count, and T_R the room duration. Accelerator duty is

$$U_{acc} = \frac{T_{cold} + \sum_{q=1}^{m} F(W_q)}{T_R}. \tag{14}$$

Let B_{day} be canonical bytes published per day. Let G_{fixed} denote fixed submission gas. The functions G_{data}, G_{hash}, and G_{store} cover data, hashing, and storage:

$$G_{day} = mG_{fixed} + G_{data}(B_{day}) + G_{hash}(B_{day}) + G_{store}(m). \tag{15}$$

Sparse checkpointing amortizes G_{fixed} but leaves throughput-dependent terms. The missing F, L1-gas, and accelerator-price records prevent a cost comparison with L1 or a persistent L2.

9 Prototype status and examples

The prototype contains an EVM engine, a transition-policy guest, a GPU proof path, Ethereum settlement contracts, and seven examples [6].

Table 3: Prototype examples and the property exercised by each example.

Example	Bounded workflow	Checkpoint shape	Property exercised	Rights or assets out
Auction	commit, seal, reveal, clear	one closing checkpoint	no reveal-reactive insertion after sealing	winner allocation; seller proceeds
Shop	authorize purchase, disconnect, fulfil	rolling	fulfilment while customer is offline	item right; merchant proceeds
ERC-7540	request, fulfil, claim deposit or redemption	two blocks, one checkpoint	asynchronous vault lifecycle [7]	accepted deposit or redemption claim
DvP	propose, affirm, exchange both legs	one closing checkpoint	all-or-nothing terminal allocation	atomic asset allocation
ERC-4626	deposit, donate, redeem	rolling checkpoints	share accounting and rounding [8]	shares, assets, or withdrawal claim
AMM	commit two trades, reveal, swap	two blocks, one checkpoint	reserve conservation under sealed order	reserve-backed swap output
Cards	private hand proofs and public moves	four outer checkpoints	inner proofs composed into room settlement	stake or prize claim

The Cards template fixes the duel, token, verifier adapter, verifying keys, funded seats, and participant root. Each browser later proves a hidden-deck action. The outer room proof verifies those public inner proofs and the ordered public moves.

The AMM is a two-trader demonstration. For pool reserves R_x, R_y, input x, and fee factor f, it computes

$$y = \frac{fxR_y}{R_x + fx}. \tag{16}$$

The proof checks reserve conservation and minimum output; the admission policy still governs censorship.

The ERC-7540 manager and exchange rate are demonstration-scoped. The ERC-4626 example implements caps, virtual assets and shares, donations, and explicit rounding. Neither example is a production vault review.

9.1 Execution compatibility

IMPLEMENTATION The recorded Osaka corpus contains 17,902 applicable cases [9]. Every case passed in both engines: revm through the transition core and EthereumJS VM. The qualified set excludes 1,615 blob and EIP-7702 delegated-code envelope cases.

All 256 opcode byte values are classified and tested. The room engine enforces gas limits, chain identifiers, block numbers, and intermediate roots. Zero base fee is an explicit room rule, so admission fees, queue caps, sender limits, and bond coverage carry the spam-control load.

The EIP-7702 exclusion also removes a natural smart-account route for signed intents. The production intent adapter described in eq. (3) remains required before delegated accounts are supported.

9.2 Proof reliability

MEASURED The repaired CUDA path completed 20 proof artifacts across five workload shapes without retry. A run passed only when its final Groth16 proof verified locally against the expected public statement. The workload alternated witness shapes to exercise the two repaired cross-stream dependencies.

MEASURED A separate RTX 4090 capacity sweep completed 120 artifacts plus a 20-artifact mirror follow-up without retry under reduced published limits. These completion runs test selected conditions; they do not estimate a production failure rate.

10 Related work and novelty boundary

Nearby systems differ in state lifetime, order authorization, correctness enforcement, data availability, and exit guarantees. The same short runtime can therefore sit inside very different security models.

10.1 Virtual and multi-party channels

Perun virtual payment channels avoid intermediary participation in each payment [10]. Multi-party channels extend the construction to changing participant subsets [11]. Nitro supports ledger and virtual channels with an on-chain force-move path [12].

Channel participants normally authorize successive states together. zkdeal users authorize individual intents, while an operator selects their order and a validity proof certifies execution. This supports asynchronous participation but leaves admission and unconstrained ordering with the operator.

10.2 Ephemeral Rollups

Picco and Fortugno describe Ephemeral Rollups for games on the Solana Virtual Machine (SVM) [13]. Accounts are delegated to a temporary runtime, periodically committed, and later undelegated. The documented system uses sequencer control and fraud-proof finalization [14].

zkdeal instead re-executes ordered EVM blocks in a validity proof. Its Ethereum journal also records custody liabilities, authenticated deposits and imports, withdrawal claims, and terminal closure. This is an architectural comparison; it is not a claim that one proof family is always superior.

10.3 Plasma and validium

Plasma uses child-chain commitments, fraud proofs, and exit games so asset safety need not depend on operator cooperation [15]. Its central lesson is that data withholding must not make accepted assets unrecoverable.

Validium combines validity proofs with off-chain data availability. For example, StarkEx can require a data-availability committee before it accepts an update [16]. By contrast, uncheckpointed sparse-room state has neither an accepted proof nor committee-backed availability.

10.4 Authenticated imports and coprocessors

Storage-proof coprocessors authenticate historical or cross-chain state and prove computations over it. Herodotus, for example, verifies storage proofs in a zkVM before returning a result on-chain [17].

The import mechanism in section 6.3 uses this pattern inside a room transition. Its contribution is the binding between the source proof, adapter policy, replay cursor, execution, and checkpoint journal.

10.5 Based and booster rollups

Based rollups let the next L1 proposer drive sequencing and inherit L1 ordering liveness [18]. zkdeal is not based: the operator orders room intents, while Ethereum orders checkpoint transactions.

Booster rollups expose the L1 environment while retaining separate L2 storage [19]. They focus on state access across a continuing rollup. zkdeal imports selected state through certified adapters and keeps the room lifecycle explicit [19].

10.6 Comparative inference

Prior work supplies every individual ingredient. The claimed combination is an Ethereum-accepted journal that ties ordered EVM execution to public accounting, recovery data, and an explicit close. An indefinitely rolling room approaches a validity rollup; the remaining distinction is its certified workflow policy, room-scoped liabilities, and terminal-state rule.

Table 4: Execution design space. Cells describe representative defaults; deployments may vary.

Family	Runtime and state	Order and correctness	Settlement and recovery	Asset exit and closure
L1 transaction	canonical EVM; one atomic call	Ethereum ordering and native execution	L1 block and native data	direct contract custody; no room close
Managed service	private service workflow	operator trust; L1 checks submitted calls	no protocol data recovery	application-specific custody and shutdown
Virtual channel [10, 12]	escrowed application session	joint state signatures; dispute or force move	latest signed state and participant-held data	terminal escrow allocation
Ephemeral Rollup [13, 14]	delegated SVM accounts	sequencer; commitments and fraud proof	periodic/final commit; revert and undelegation	base-layer accounts return on close
zkdeal	**authenticated EVM room**	**signed intents; operator order; validity proof**	**journal plus canonical checkpoint data**	**L1 liabilities and claims survive close**
Plasma / validium [15, 16]	continuing off-chain state	operator; fraud or validity proof	exit data or off-chain availability committee	escrow exit; no inherent bounded close
Persistent L2	continuing chain and bridge	sequencer, committee, or L1; proof or dispute	periodic roots and system-specific data path	bridge withdrawal; no task-scoped close
Based / booster rollup [18, 19]	persistent rollup or L1-reflecting state	L1-driven or host-rollup order and proof	host-rollup data and recovery	host bridge; no inherent terminal close
Storage coprocessor [17]	authenticated query or computation	request selects data and program; storage proof	returns a verified fact, not a state chain	no custody lifecycle by itself

11 Conclusion

zkdeal gives a bounded EVM workflow an authenticated opening state, user-authorized intents, an operator-selected order, and one Ethereum checkpoint path. The validity proof certifies execution; the room manager enforces continuity, accounting, and closure.

The prototype demonstrates the composition, not production latency or cost. The intended niche is a multi-step task that benefits from fast provisional interaction and explicit termination, while Ethereum remains the source of canonical settlement.

Appendices

Normative field definitions, the L1 acceptance algorithm, qualification records, and the illustrative exposure model are collected below.

A Journal field classes

Table 5: Normative checkpoint-journal field classes. Integer widths and ABI order follow the settlement interface.

Class	Bound fields
Identity	deployment, room, epoch, authorization mode, template, proof program, proof system, execution-policy commitment
Execution	batch and block ranges, timestamps, minimum deadline, pre/post roots, ordered-data and canonical-data commitments
Participants	pre/post participant and approver roots, epochs, counts, capacity, change cursors
Liveness	deposit, admission, and forced-queue cursors; record and outcome commitments
Imports	source L1 block, header and state roots, adapter commitment, import cursor
Assets and close	liability commitments, withdrawal epoch and root, inclusion range, close flag

Proof-work telemetry is not part of the settlement statement. The generic timestamp and inclusion-range fields are required protocol fields; the current prototype does not yet include them.

B Acceptance algorithm

Let Q be the accepted room state and J the submitted journal. Let Π be its validity proof, C canonical batch data, X auxiliary records, and vk the registered verification key.

The journal supplies block range $[b_{min}, b_{max}]$, batch times τ_1, τ_z, and minimum intent deadline d_{min}. The current L1 block and timestamp are b_{L1} and τ_{L1}.

The room manager performs:

1. Reject a closed room and require $b_{min} \le b_{L1} \le b_{max}$.
2. Match the identity, policy, checkpoint index, pre-state, participant state, and cursors in J against Q and table 5.
3. Require $Q.\tau \le \tau_1 \le \tau_z \le \tau_{L1} \le d_{min}$.
4. Compute $c = \text{Keccak256}(\text{ABI}(J))$ and require $\text{Verify}(vk, c, \Pi) = 1$.
5. In journal-unanimous mode, verify every active approver's signature on J. In per-intent mode, rely on the proved intent signatures.
6. Recompute the commitment to C and validate the liveness, participant-change, and import records in X.
7. Apply the liability transition, require custody conservation, and bind the next withdrawal root.
8. Replace Q atomically with the accepted post-state and cursors. If J closes, mark the room closed.

Any failure reverts the full submission.

C Required measurement record

Table 6: Fields required for future qualification records.

Category	Required fields
Timing	request boundary, composite, recursion, identity, Groth16 wrap, verification, serialization, end-to-end time
Proof complexity	user and total cycles, segments, witness bytes, canonical bytes
Accelerator	device, utilization, memory, power, billed seconds, software and asset versions
Queue and retry	queue delay, workload identity, attempt count, retry reason, terminal status
L1 acceptance	verifier, journal, data, hashing, and storage gas; inclusion delay; finality rule

Hardware cost comparisons must state the billed accelerator-seconds and price source. For an accepted checkpoint, record L1 costs separately from proof-endpoint measurements.

D Illustrative market-exposure model

For a driftless log price with annualized volatility σ_a, the expected absolute movement over t seconds is

$$\mathbb{E}|\Delta \log P_t| = \sigma_a \sqrt{\frac{2t}{\pi Y}}, \tag{17}$$

where Y is seconds per year.

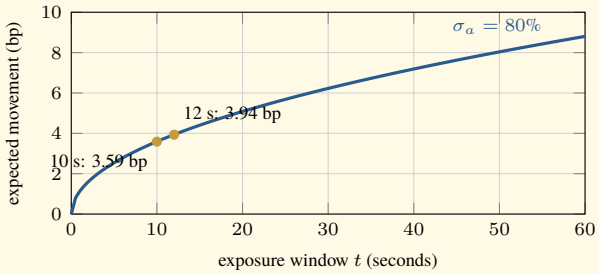

Figure 13: MODELED Expected absolute movement under the illustrative volatility model. The 10-to-12-second difference is 0.35 basis points.

This model omits spreads, jumps, and market impact; it supports only the narrow comparison in section 7.3.

References

[1] Ethereum Foundation. *Zero-Knowledge Rollups*. 2026. (Visited on August 5, 2026).

[2] Ethereum Foundation. *State Channels*. 2026. (Visited on August 5, 2026).

[3] Felix Lange. *EIP-1186: RPC Method to Get Merkle Proofs*. 2018. (Visited on August 5, 2026).

[4] Philip Daian et al. "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability". In: *2020 IEEE Symposium on Security and Privacy*. 2020, pp. 910–927. DOI: 10.1109/SP40000.2020.00040.

[5] RISC Zero. *RISC Zero zkVM Documentation*. 2026. (Visited on August 5, 2026).

[6] zkdeal contributors. *zkdeal Reference Implementation*. 2026. (Visited on August 5, 2026).

[7] Martin Baer et al. *ERC-7540: Asynchronous ERC-4626 Tokenized Vaults*. 2023. (Visited on August 5, 2026).

[8] Joey Santoro, transmitted-light, and jet-lab. *ERC-4626: Tokenized Vaults*. 2021. (Visited on August 5, 2026).

[9] Ethereum Foundation. *Ethereum Execution Specification Tests*. 2026. (Visited on August 5, 2026).

[10] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. "Perun: Virtual Payment Hubs over Cryptocurrencies". In: *2019 IEEE Symposium on Security and Privacy*. 2019, pp. 106–123. DOI: 10.1109/SP.2019.00020.

[11] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. "Multi-Party Virtual State Channels". In: *Advances in Cryptology – EUROCRYPT 2019*. 2019, pp. 625–656. DOI: 10.1007/978-3-030-17653-2_21.

[12] Tom Close. *Nitro Protocol*. Cryptology ePrint Archive, Paper 2019/219. 2019. (Visited on August 5, 2026).

[13] Gabriele Picco and Andrea Fortugno. *Ephemeral Rollups Are All You Need*. arXiv:2311.02650. 2023. DOI: 10.48550/arXiv.2311.02650. (Visited on August 5, 2026).

[14] MagicBlock. *Ephemeral Rollup*. 2026. (Visited on August 5, 2026).

[15] Joseph Poon and Vitalik Buterin. *Plasma: Scalable Autonomous Smart Contracts*. Working draft. 2017. (Visited on August 5, 2026).

[16] StarkWare. *StarkEx Data Availability*. 2026. (Visited on August 5, 2026).

[17] Herodotus. *Data Processor: Verifiable Compute over Authenticated On-Chain Data*. 2026. (Visited on August 5, 2026).

[18] Justin Drake. *Based Rollups—Superpowers from L1 Sequencing*. 2023. (Visited on August 5, 2026).

[19] Brecht Devos. *Booster Rollups—Scaling L1 Directly*. 2023. (Visited on August 5, 2026).