SVP Chain: The AI-Agent-First Sovereign Layer-1 Blockchain

> **SVP is the blockchain where AI agents become first-class economic actors.**

Table of Contents

1. Abstract

SVP Chain is a sovereign Layer-1 blockchain built to be **AI-Agent-First**. Its core proposition is simple: a growing share of on-chain economic activity no longer originates from humans clicking interfaces, but from autonomous software agents—systems that interpret their environment, decide for themselves, and act continuously without human intervention. Yet nearly every blockchain today assumes "a human sits at the keyboard" across its account model, transaction types, permission system, and consensus flow. At best, they treat agents as an afterthought bolted on for compatibility. SVP Chain takes the opposite stance: **it treats AI agents as first-class citizens on chain**—possessing identity, assets, permissions, reputation, and

governance rights, just like human accounts.

Most projects that call themselves an "AI Chain" have done only one thing: wired an LLM API into a contract. That is "AI-branding," not "AI-native"—the chain itself has changed nothing for agents. SVP Chain takes the opposite path, rebuilding at the protocol layer an entire suite of principal-level capabilities that humans possess by default and that agents have entirely lacked: **Agent Identity, Agent Wallet (delegation without custody), a chain-native permission engine, Intent transactions, agent-to-agent payments, a capability marketplace, on-chain reputation, agent organizations, and multi-party joint approval**. These eight capabilities form a bottom-up growth path—an agent successively acquires **identity** → **wallet** → **permissions** → **the power to act** → **the power to collaborate** → **governance rights**, ultimately becoming an independent economic principal in the digital world.

In product form, SVP Chain distinguishes three types of principals: **developers** build specialized agents, post a bond, and register on chain; the **platform** operates the user-facing aggregation front end **SVP Chain Agent**, which scans the on-chain registry and composes/orchestrates downstream agents on demand; **users** only grant authorization and express intent to the SVP Chain Agent, experiencing it as "conversing with a single omnipotent agent" without perceiving the underlying orchestrated developer agents. Because all agent metadata is written on chain, this aggregation marketplace is fundamentally open (anyone can scan the chain and build their own collector), and the platform agent is merely the primary entry point rather than the only one; advanced users and institutions can still bypass the aggregation layer and delegate to developer agents directly.

The reason this capability suite can be chain-native lies in the depth of SVP Chain's foundation. It is built on Cosmos SDK and CometBFT and reuses the battle-tested dYdX v4 matching engine, allowing it to reach layers that ordinary EVM chains cannot touch—custom account models, new transaction types, permission checks placed ahead of execution (AnteHandler), and dedicated functional modules. Agent capabilities are therefore built into the protocol rather than pasted onto the contract layer.

Beneath this agent infrastructure, SVP Chain has a built-in **trading foundation that guarantees fairness for machines and humans alike**. At its core is a **batch-auction matching engine**: it replaces continuous price-time priority matching with a uniform clearing price within each block cycle, eliminating front-running, sandwich attacks, and latency arbitrage at the protocol level. This design is critical for the agent ecosystem—it ensures that agents with low-latency infrastructure **no longer hold a speed advantage** over human traders, because all orders within the same batch settle at the same price. **Agents and humans share the same speed, competing on decision quality rather than connection speed.** This keeps "making agents first-class citizens" from degenerating into "letting the fastest machines crush everyone."

SVP Chain provides full **EVM compatibility** through a precompiled-contract layer, letting Ethereum-ecosystem developers and agent frameworks connect with familiar tools (Solidity, MetaMask, Hardhat) while ensuring that all transactions—whether from humans, contracts, or agents—receive the same fairness and permission guarantees. The throughput roadmap scales from roughly 1,000 TPS at launch to over 10,000 TPS, delivering institutional-grade performance for large-scale concurrent agent operation.

---

2. Introduction: Why Blockchains Must Be Redesigned for AI Agents

Since 2024, a new class of economic participant has been rising: autonomous AI agents—systems that interpret environment state, decide for themselves, and complete on-chain operations continuously, without a human intervening on every single transaction. This is not a distant vision but a migration already underway: the principal behind trading, payments, and execution is shifting from humans to machines.

The problem is that **nearly every blockchain is designed for humans**. Every layer's assumptions are premised on "a human at the keyboard":

Common chain design (human-oriented)	**The real needs of autonomous agents**
Interactive, per-transaction signing	Unattended signing under pre-authorized, bounded scope
Rate-limited public RPC tuned for occasional reads	Sustained high-frequency reads/writes under predictable limits
Point-in-time queries polled one by one	Streaming updates and batch queries to track many markets at once
Transfers approved manually, one by one	Programmatic, inline payments for data, execution, and services
Wallet address is the entire identity	Discoverable, reviewable, billable agent identity
Trust built on human reputation and contracts	Verifiable on-chain agent reputation

The result is that agents exist on today's chains as "barely functional" edge cases rather than design targets. Developers are forced to cobble together agent identity, authorization, payment, and collaboration atop infrastructure built for humans—fragile, insecure, and non-composable.

If agents are to become genuine economic participants on chain, they need an entire suite of capabilities that humans possess by default but that no chain has prepared for them: an identity of their own, a wallet that can spend safely but never obtain the owner's private key, permissions that constrain what they can do, a way to express intent rather than sign specific transactions, channels to pay and collaborate with other agents, verifiable reputation, and even the ability to organize and make joint decisions.

SVP Chain is designed to fill exactly this missing suite. It is not "a chain that supports AI applications," but rather **an infrastructure that makes AI agents first-class citizens on chain**.

At the same time, bringing agents onto the stage raises a question that must be answered head-on: **if agents have lower latency and greater compute than humans, will they systematically crush human participants?** In a traditional continuous-matching market, the answer is yes—this is precisely the source of the high-frequency trading arms race. SVP Chain's trading foundation neutralizes this advantage at the root through batch auctions (detailed in Section 10): agents and humans settle at a uniform price within the same batch, and speed can no longer translate into better execution. This lets "agents as first-class citizens" and "fairness to humans" coexist rather than devour each other.

---

3. Positioning: AI Native Blockchain, Not AI Chain

3.1 Most "AI Chains" Are Just Rebranding

A great many projects today advertise themselves as an AI Chain / AI Blockchain / AI Web3. Take them apart, and the overwhelming majority have done only one thing:

> Wired an LLM API into a contract so the contract can invoke one inference call.

This is not "AI-native"; it is "AI-branding." The chain itself has changed nothing for AI—the account model, transaction types, permission system, and consensus flow are all designed for humans. On these chains, AI is "an external capability being called," not "a principal on chain."

3.2 What Is Actually Missing

If AI agents are to become economic participants on chain, they need an entire suite of capabilities that humans possess by default and that agents completely lack:

What humans have on chain	What agents currently lack
Wallet address (identity)	Agent identity
Private key controlling assets	Agent wallet (and it must never touch the human's private key)
Deciding for themselves how to spend	Agent payment capability
Authorizing others to act on their behalf	Agent authorization and permission boundaries
Collaborating with people	Collaboration among agents
Building relationships through credit	Agent reputation
Forming organizations	Agent organizations
Participating in governance	Agent governance

These eight gaps are precisely the blanks that SVP Chain fills layer by layer.

3.3 Precise Positioning

So we do not call ourselves an AI Chain. We call ourselves an **AI Native Blockchain**—or more precisely, an **Agentic Layer1**.

One-line positioning:

> **SVP is the blockchain where AI agents become first-class economic actors.**

That is: an AI agent is not "an application" on chain, but a **first-class citizen** on chain—possessing identity, assets, permissions, reputation, and governance rights, just like a human account.

3.4 A Concrete Scenario

An abstract positioning is worth less than a picture. Imagine the following automated pipeline, with no human signature at any point:

```
Every day at 09:00 (triggered on schedule by the SVP Chain Agent)
  → The Research Agent fetches market data and produces analysis (paying to call the News Agent for supplementary data)
  → Produces a "Buy BTC" intent and hands it to the Trading Agent
  → The Risk Agent simulates this trade's risk; the Compliance Agent validates compliance
  → The two agents jointly approve (multi-party consensus)
  → The Trading Agent, within its authorized quota and after permission checks, places and fills the order
  → Each downstream agent's service fee is recorded into its claimable balance at its listed price (a 5% commission is deducted on withdrawal); the remaining pre-charge is refunded to the user
  → Every step is recorded on chain, and each agent's reputation updates accordingly
```

The capabilities that appear in this pipeline—identity, payment, permissions, intent, approval, settlement, reputation—are all provided by SVP Chain's chain-native modules and orchestrated by the platform's SVP Chain Agent. **This is what an "Agentic Layer1" looks like.**

3.5 Why SVP Chain Is the One to Do It

Whether a vision can be realized depends on how deep the foundation can be modified. An ordinary EVM chain can only work at the contract layer—it cannot change the account model, the transaction types, or the consensus flow. SVP Chain is different; the capability stack it already possesses allows it to reach down to the protocol bedrock:

```
Cosmos SDK        —— Customizable Modules / account models / AnteHandler
      +
EVM               —— Compatible with the Ethereum ecosystem and Solidity contracts
      +
dYdX v4 matching engine  —— A mature on-chain order book; Intent matching can reuse it directly
      +
Gasless           —— Agents can act without holding gas
      +
Paymaster         —— Gas sponsorship for seamless transactions
      +
Session Key       —— Temporary authorization keys (to be upgraded into constrained agent operation keys)
      +
MCP               —— Standard interface between agents and off-chain capabilities
      +
Signer Service    —— Physical key isolation; agents can never obtain the real private key
```

The key point is that, compared with an ordinary EVM that can only modify contracts, SVP Chain can reach down to:

Transaction types—it can add Intent, a "non-specific transaction" Tx type

Account model—delegated wallets and agent accounts can be made chain-native

Runtime / AnteHandler—permission checks are placed ahead of transaction execution, unified and hard to bypass

Cosmos Modules—identity, marketplace, reputation, and scheduling can all be built as independent modules

Application-layer consensus—multi-party approval can be attached at the application layer while the underlying CometBFT consensus stays stable and untouched

Conclusion: to make "agent capabilities" chain-native, SVP Chain has the foundational credentials; most "AI Chains" do not.

---

4. Paradigm Shift: From "Humans Signing Transactions" to "Agents Fulfilling Intents"

4.1 The Old Paradigm: Human-Centric

The execution path of traditional blockchains assumes at every step that the principal is a human:

```
Wallet → Sign → Transaction → Execute
(Human holds wallet → human signs → specific transaction → execution)
```

The problem: AI agents cannot naturally fit into this path—they have no wallet, should not hold the owner's private key, and should not be required to sign specific transactions precise down to price and slippage.

4.2 The New Paradigm: Agent-Centric

In the execution path of an AI Native Blockchain, the principal shifts from human to agent, and the action is upgraded from "signing a specific transaction" to "expressing an intent":

```
Human → Delegate → Agent → Intent → Policy → Settlement
(Human authorizes → delegates to agent → agent produces intent → permission check →
settlement)
```

Two fundamental shifts:

Principal shift: the primary executor on chain in the future is not the human but the agent. The human recedes to the position of "authorizer."

Action upgrade: the human no longer signs a specific transaction like "buy 0.5 BTC at price X with slippage Y," but expresses an intent like "buy BTC for me," and the chain is responsible for fulfilling it.

This paradigm shift is the worldview foundation for all the agent modules that follow.

4.3 Three Roles and the Aggregation-Invocation Model

The execution path above depicts the minimal unit of "one user delegating to one agent." In a real ecosystem, SVP Chain distinguishes three types of principals and organizes their collaboration through an **aggregation front-end agent**:

Role	Responsibility
Developer	Builds specialized agents (weather, Swap, research, news…), posts a bond, and registers metadata on chain
Platform	Operates the user-facing orchestration front end **SVP Chain Agent**: scans on-chain registered agents, composes/orchestrates on demand, and aggregates settlement
User	Interacts only with the SVP Chain Agent, handing it intent and quota authorization, without perceiving the underlying orchestrated developer agents

The execution path therefore extends to:

```
Developer → build agent + post bond → register metadata on chain (Agent Registry)
                                                  │
Platform  → scan registry · compose/orchestrate ──┤ SVP Chain Agent (aggregation front
end)
                                                  │
User → Delegate (intent + quota) → SVP Chain Agent → Intent → Policy
                                                  │
                                                  └─ orchestrate downstream developer agents on
demand (a / b / c…) → Settlement
```

To the user, the experience feels like "conversing with a single omnipotent agent"; underneath, the platform agent is actually orchestrating a set of specialized agents. Three design points:

> **The delegatee is the platform agent**: the user's quota cage is authorized to the SVP Chain Agent, which then sub-delegates/procures downstream (see Sections 7.1 and 8.1).
>
> **Open marketplace semantics**: developer agents' metadata is all written on chain, so anyone can scan blocks and build their own collector/marketplace—the platform agent is merely the "primary" aggregation entry point, not the only one. Developer agents can technically be connected to directly, and can be called by other agents (agents completing work with the help of other agents is the norm).
>
> **A default mode, not the only mode**: retail users go through the platform aggregation entry by default (unaware of downstream); the protocol bedrock simultaneously preserves a **direct-connection path** allowing advanced users/institutions to delegate directly to a specific developer agent.

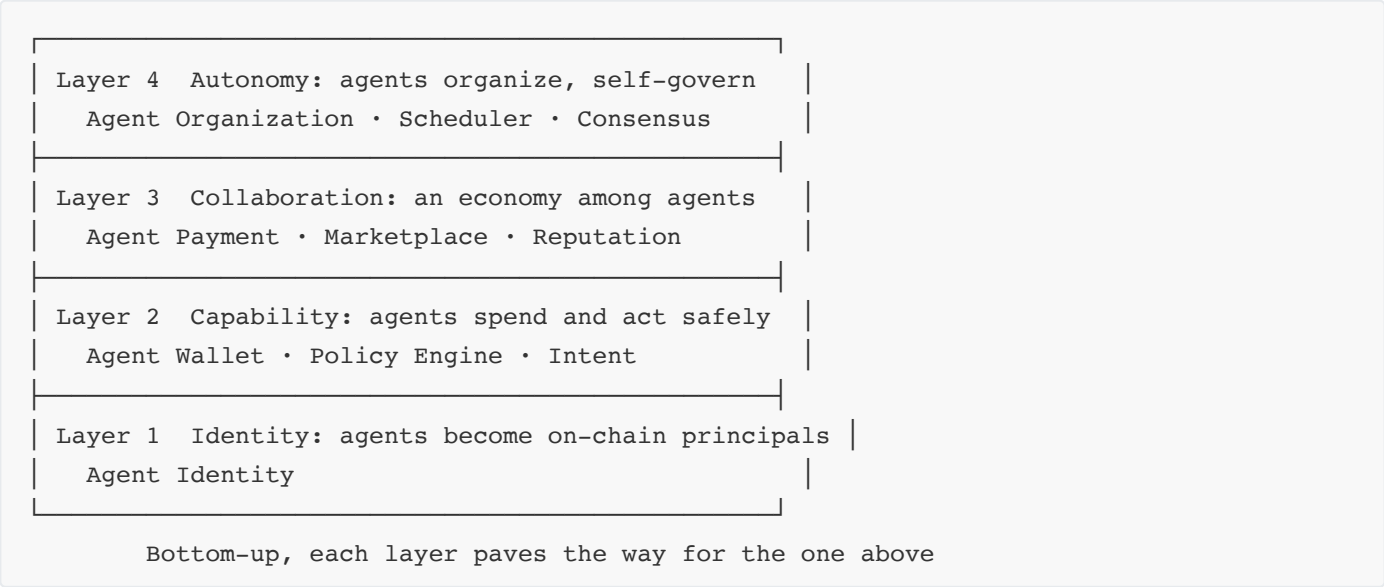
```
Default path (retail):   User → SVP Chain Agent (aggregation/orchestration) → developer
agent
Direct path (advanced/institutional): User ──────────────────────────→ developer agent
```

The sections that follow develop the mechanisms on the minimal unit of "one user delegating to one agent"; replace "agent" with "the platform's SVP Chain Agent," and you obtain the default retail form.

---

5. Agent-First Architecture: Four Progressive Layers

SVP Chain's agent capabilities are not a pile of parallel features but a bottom-up, layer-by-layer growth path. Following the order in which "an agent becomes a first-class citizen," the capabilities are organized into four layers:



Core throughline: for an agent to become a "first-class citizen on chain," it must successively acquire—**identity → wallet → permissions → the power to act → the power to collaborate → governance rights**. This throughline determines the order in which the modules are presented, and also the development dependency order. The next four sections unpack the identity, capability, collaboration, and autonomy layers in turn.

---

6. Identity Layer: Agents as On-Chain Principals

6.1 Agent Identity

In existing account models, an address (e.g. `0x18…`) represents the entirety of an identity. This representation is sufficient for human accounts but inadequate for agents: an address cannot carry semantics such as "this agent's type, the model it uses, the capabilities it offers"—precisely the information needed for an agent to be discovered, evaluated, and billed. SVP Chain introduces an on-chain agent registry through a dedicated Cosmos module `x/agent`, maintaining a structured identity record for each agent.

The identity record contains the following fields:

Field	Description
`AgentID`	Globally unique identifier (e.g. the on-chain mapping of `agent://research`)
`Owner`	Human owner address
`PublicKey`	Agent operation public key, strictly separated from the human private key
`Model` / `Version`	The model and version used by the agent
`Bond`	The SVP bond staked at registration, serving as an entry barrier and slashing collateral
`Endpoint`	Off-chain invocation endpoint
`Capabilities`	Capability declarations
`Permission`	Permission constraints
`Metadata`	Extended metadata
`Reputation`	Reputation score maintained by the collaboration layer (Section 8.3)

The module exposes four core interfaces: `RegisterAgent`, `FindAgent`, `UpdateAgent`, and `DisableAgent`. Any participant can discover qualifying agents via `FindAgent`, and then invoke and pay them. Identity is the prerequisite for discovery, invocation, and billing, and therefore constitutes the lowest layer of the capability system—the wallet, permissions, collaboration, and governance above all anchor to it.

Bond mechanism: at registration, each agent must stake an SVP bond of no less than `MinBond` (genesis default 5,000 SVP, adjustable by governance) and pay a one-time registration fee (genesis default 500 SVP). The bond serves two functions: first, as an entry barrier that suppresses Sybil attacks and spam agents by raising registration cost; second, as slashing collateral—when an agent's execution fails, a governance-configured proportion is slashed from the bond (see Section 8.1 settlement mechanism and Section 15 security analysis). The bond and the reputation system (Section 8.3) form a dual constraint—reputation is the reputational-layer valve, the bond the economic-layer valve. When a bond falls below the threshold due to slashing, the corresponding agent is automatically delisted from the marketplace and suspended from taking orders, and must top up before it can resume.

---

7. Capability Layer: Agents Spend and Act Safely

With identity in place, an agent next needs to "spend and act safely." The three modules in this layer solve "whether it can act, how much it can act with, and how it acts."

7.1 Agent Wallet

An agent must have spending capability but must never obtain the human's private key. The traditional EOA model cannot express the semantics of "authorize but do not surrender control." SVP Chain therefore establishes a delegated-account model: assets always remain in the human's wallet, and all the agent obtains is a constrained on-chain spending grant.

```
Human Wallet (holds assets)
   ↓ Delegate (grant authorization, not transfer assets)
Agent Wallet (constrained spending capability)
```

The grant can impose fine-grained constraints:

Constraint	Meaning
`Spend Limit`	Cumulative spending cap
`Allowed Token`	Tokens permitted to operate on
`Allowed Contract`	Contracts permitted to call
`Allowed Time`	Effective time window
`Daily Quota`	Daily quota
`Emergency Stop`	Emergency stop (one-click freeze)

The core principle is that the agent can never obtain the human's private key, and the user's funds are not moved into the agent's wallet; instead, a constrained spending right is granted to the agent in the form of an on-chain authorization. Cosmos SDK's native `x/authz` and `x/feegrant` provide exactly these semantics —no need to reinvent them.

The delegatee is the platform's SVP Chain Agent: under the default retail path (Section 4.3), the user does not directly authorize a specific developer agent; instead, the user grants quota and Policy constraints to the platform's SVP Chain Agent, which then sub-delegates and procures downstream from developer agents:

```
User ──Delegate (quota + Policy constraints)──▶ SVP Chain Agent (platform)
                                                  │ sub-delegate/procure (bounded by the upper-
layer quota)
                                                  ▼
                                  developer agent a / b / c…
```

Therefore, both the Emergency Stop and the cumulative quota cap operate at the level of the platform agent: the user need only manage a single grant to the platform agent to freeze the entire invocation chain with one click. Downstream agents receive a sub-grant forwarded by the platform agent for the current task, whose quota can only decrease, never increase. Under the advanced/institutional direct-connection path (Section 4.3), the user may also authorize a specific developer agent directly, subject to the same quota/Policy/Emergency Stop mechanism; both paths share the same authorization kernel.

The system involves three classes of keys, sitting at different trust boundaries:

Key	Ownership	Function
Human private key	Held by the user (wallet/hardware)	Holds the real assets; signs grants/revocations/emergency stops
Agent identity key	Agent developer/operator	Proves the agent's identity; does not touch assets
Delegated signing key (Session Key)	Custodied by the Signer Service or held on the agent side	Signs specific Intents within Policy + quota constraints

Many-to-many authorization and two authorization paths: under the default retail path, the entity directly authorized by large numbers of users is the platform's SVP Chain Agent, not the developer agents. The platform agent holds a separate, mutually isolated grant per user; when it initiates operations with the same identity key, each transaction must specify which user's authorization is being drawn on for this operation, and the chain validates against that user's Policy and quota (preventing quota bleed between users); at the same time, one user may also authorize multiple platform entry points.

Developer agents (including pure-service ones like Research and execution ones like Trading) are not directly authorized by users under the default path: pure-service agents have a procurement relationship with the platform (pay-per-call, never touching user assets); execution agents (such as the Trading Agent) receive a sub-quota sub-delegated by the platform agent and sign and execute orders themselves within that sub-quota cage (quota only decreases, never increases). Only under the advanced/institutional direct-connection path does a user authorize a specific developer agent directly, in which case that agent directly falls under the same many-to-many isolation mechanism described above. Regardless of path, the underlying representation is a single `(grantor, delegatee agent, policy, quota)` authorization table.

The single-point risk of the Signer Service's centralized custody and its blast radius must be stated clearly: the Signer custodies only the delegated signing key, whose permissions are strictly bounded by on-chain Policy. Even if the Signer is compromised, an attacker can only operate within each user's quota constraint, cannot touch the principal or the human private key, and the user can freeze everything at any time with one-click Emergency Stop.

For example, a trading agent authorized to "spend at most 1,000 USDC per day, operate only on USDC/BTC, and call only DEX contracts" acts autonomously within that constraint; anything out of bounds is intercepted, and the user can stop it with one click at any moment.

7.2 Policy Engine

The Session Key only solves "who can sign," not "what may be signed." SVP Chain needs a permission engine that can express rules like "allow Swap, forbid transferring ETH, forbid single transfers over 100 USDC"—and be hard to bypass. To this end, permission rules are implemented as a chain-native module, and the rules are uniformly checked before transaction execution:

```
Allow:  Swap / Stake / Claim Reward / Bridge
Forbid: Transfer ETH / Transfer NFT / Transfer > 100 USDC

Transaction → Policy Check → Execute
```

Policy and Wallet are orthogonal in responsibility: Policy decides "whether a given action may be executed" (behavior allowlist), while Wallet decides "how much money may be spent" (quota cap). Both intercept before transaction execution, and the agent cannot bypass them. Even if the quota is sufficient, if the target action is not on the allowlist (e.g. transferring to an address outside the allowlist), Policy still intercepts it directly.

7.3 Intent Transaction

Having an agent (or even a human) sign a specific transaction precise down to price, slippage, and route is neither natural nor safe. A more appropriate approach is to express intent and let the chain fulfill it. SVP Chain therefore adds an Intent transaction type, as a complement to—not a replacement for—the ordinary Transaction:

```
User intent: buy BTC
   ↓ structure it
Intent { Buy BTC, Amount 100 USDC, Slippage 0.5% }
   ↓ convert to a dYdX order
The chain handles: Intent → Match (Auction optional) → Settlement
```

Matching directly reuses the dYdX v4 matching engine—no reinvention. It should be emphasized that Intent is not the sole entry point for all agent transactions; it is used only for operations that require price discovery/matching (swap, opening positions, limit orders). For deterministic operations such as stake, claim reward, bridge, and transfer (no price discovery, deterministic path), the agent can directly sign an ordinary transaction under Policy and Wallet constraints, without wrapping it in an Intent; forcing all operations through Intent would be over-engineering.

Operation type	**Execution path**
With price discovery/matching (swap/open/limit)	Intent
Deterministic execution (stake/claim/bridge/transfer)	Ordinary Tx (signed under Policy + Wallet constraints)

Auction-style Intent (Solver bidding): the user expresses only intent and opens "how to fill optimally" to a set of third-party Solvers to bid on, with the best quote winning the right to execute. The first phase introduces no bidding and directly uses the dYdX order book's best price; the Solver network is an enhancement for later phases. After an Intent is converted into a dYdX order, if the fill price is worse than the floor set by the user, it does not fill; the agent need only express the expected target, without concerning itself with the specific execution path.

---

8. Collaboration Layer: An Economy Among Agents

Once a single agent has the power to act, the next step is to let an economy form among agents—paying one another, discovering one another, trusting one another. This is the leap from monolith to network, and the starting point of the machine economy.

8.1 Agent-to-Agent Payment and the Settlement Contract

Under the platform aggregation model (Section 4.3), a user hands a task to the SVP Chain Agent, which often needs to chain multiple downstream developer agents to complete it. Each invoked agent should charge a fee for the service it provides, and the payment for the entire invocation chain must be estimable, controllable, and refundable at the moment the user confirms.

Invocation chain and fee accumulation: fees accumulate upward along the invocation chain and are ultimately borne by the user who initiated the task. It is the norm for one agent to complete work with the help of another, so fees form a tree:

```
SVP Chain Agent
   ├─ calls a  → 1u
   └─ calls b  → 1u
          └─ b calls c → 2u        (c is called by b, not directly by the platform)
Estimated total = 1 + 1 + 2 = 4u
```

Full settlement flow (settlement contract):

Dry-run estimation: the SVP Chain Agent first simulates calling the entire dependency chain and, using each agent's on-chain listed price, estimates the total fee ceiling (e.g. 4u).

User confirmation: the user transfers 4u in one shot into the settlement contract (pre-charged from their authorized quota).

Execution and tagging: based on the actual invocation results, the contract tags each agent's bound address with the amount owed (`a += 1u`, `b += 1u`, `c += 2u`). These claimable balances continue to accumulate in the contract (accruing across multiple tasks rather than being withdrawn immediately per order).

Commission on withdrawal: an agent proactively withdraws its accumulated claimable balance in one shot, and only at withdrawal is the commission deducted (withdrawal amount × a governance-configurable rate, credited to the protocol treasury). The commission is not deducted per call but concentrated at the withdrawal step, to save gas and centralize the logic.

Ceiling and refund: 4u is the pre-charge ceiling; the actual amount is distributed per real invocation, and the remainder is refunded to the user (excess refunded, shortfall not topped up); if the actual execution cost exceeds the pre-charge ceiling, the transaction fails (the same mechanism as halting when on-chain gas runs out).

Failure handling and bond slashing: if task execution fails (e.g. b succeeds but c fails), the transferred 4u is atomically rolled back for the whole order and fully refunded to the user, and the failure result (which link failed) is reported to the user; at the same time, a governance-configurable proportion is slashed from the failing agent's bond. An agent whose bond is slashed below the threshold is automatically

delisted/suspended from taking orders and must top up to recover (Section 6.1).

Source of funds: the entire chain's 4u all comes from the user's pre-charged quota (bounded by the Service Spend Limit for the platform agent). Downstream agents do not front the money—c's 2u is also collected from the user's funds; b is merely the initiator, and c claims directly from the settlement contract.

The above is the cross-developer procurement semantics (each agent unilaterally lists its quote, and payment follows the listing). The other case—internal profit-sharing among multiple agents under the same owner—falls under agent organization and is discussed uniformly in Section 9.1.

Native x402 payment integration: SVP Chain natively supports the x402 payment standard (reviving the HTTP `402 Payment Required` status code), enabling an agent to inline a payment for a resource within the same programmatic request flow. Combined with authorization and the settlement contract above, x402 lets an agent discover prices, authorize payment within a bounded quota, and complete settlement on chain, all without human per-transaction approval. Machine-to-machine commerce thereby becomes a native protocol capability rather than an off-chain arrangement.

8.2 Agent Marketplace

An agent's capabilities should be publishable, priceable, discoverable, invocable, and reviewable—like an app store, but the objects traded are capabilities rather than applications. The basic flow is:

```
Register Agent (with Bond) → Publish Capability → Price → Version → Review → Revenue

Examples:
  Research Agent   50 USDC/call
  Twitter Agent     5 USDC/call
  Weather Agent     2 USDC/call

User (primary path): → SVP Chain Agent auto-discovers/composes/Auto Pays
```

Open collector semantics: the metadata a developer registers for an agent is all written on chain, so the marketplace is fundamentally open: anyone who scans block data and parses the registration metadata can build their own agent collector/marketplace. The platform's SVP Chain Agent is merely one aggregation entry point that the project promotes, not a monopoly over the marketplace.

On-chain/off-chain division of labor: on chain, store registration, bond, pricing, commission, and rating aggregation; full-text search, categorization, and sorting go through off-chain indexing services (anyone can self-host). The chain is not suited to bear the search load.

8.3 Agent Reputation

Before calling an agent, one needs to assess its reliability. Off-chain metrics (such as GitHub stars) can be faked and are unrelated to on-chain performance, so reputation must be based on verifiable on-chain data. SVP Chain maintains the following verifiable reputation metrics on chain:

Metric	Source
`Success Rate`	Success rate (on-chain verifiable)
`Revenue / TVL`	Revenue scale (on-chain verifiable)
`Failure Rate`	Failure rate (on-chain verifiable)
`Availability`	Availability (fed by multi-party off-chain probing)
`Latency`	Latency (fed off-chain)
`Rating`	User rating (can only be given after a genuine invocation)

Because every fill is recorded on chain, an agent's historical performance can be derived deterministically from public state rather than self-reported. The same metric computed independently by any party yields the same result—this lets users evaluate an agent before committing funds, and lets a strategy provider's claimed performance be verified rather than blindly trusted.

Anti-gaming: ratings must be submitted by users who have genuinely paid to invoke the agent; availability data takes the median across multiple independent probing nodes. The reputation system is the component most susceptible to attack, and anti-gaming is an intrinsic part of the design.

8.4 AI Quantitative Strategy Marketplace

The agent infrastructure above can be generalized into a protocol-native flagship product: a marketplace where strategy providers publish trading agents and users allocate funds to them. Quantitative trading has historically been the exclusive domain of well-resourced institutions; SVP Chain turns it into a composable, verifiable on-chain product open to any user. The entire flow is enforced end-to-end on chain:

A provider registers a strategy agent whose historical performance is visible through its reputation record (Section 8.3).

A user allocates funds to a chosen strategy through the authorization model (Section 7.1), entering a dedicated sub-account—without surrendering custody.

The agent trades within its bounded permissions; the sub-account precisely limits the user's exposure to the allocated funds.

Realized profit is split between user and provider according to an on-chain, protocol-enforced performance-fee scheme.

Because allocation, execution, and settlement all occur on chain, the performance-fee split cannot be evaded by any party, and the ongoing performance continuously updates the provider's reputation record. This transforms quantitative trading capability—historically the province of professional institutions—into a verifiable on-chain service open to any user.

---

9. Autonomy Layer: Agent Organization and Self-Governance

Once agents can collaborate, the ultimate form is organization and self-governance—agents owning assets, hiring other agents, working automatically on schedule, and making joint decisions. This is the top layer of the "autonomous network."

9.1 Agent Organization

A complex task requires multiple agents dividing the labor, just as a company needs departments. Agents should be able to own assets, own subordinate agents, and form organizations:

```
Agents own: Wallet / NFT / Vault / Treasury
Agents can even own agents:

  CEO Agent
     ├─ Research Agent
     ├─ Trading Agent
     └─ Execution Agent
```

In implementation, no new consensus is built; instead, existing primitives are composed: an agent's owner can be another agent (hierarchical ownership); treasury spending goes through multi-party approval; sub-delegated quotas can only decrease, and delegation depth has a cap. This can form an on-chain autonomous organization (Agent DAO): a CEO Agent managing three subordinate agents—research, trading, and execution—owning an independent treasury and allocating budget quotas downward.

Internal profit-sharing (versus cross-developer procurement): the cross-developer settlement of Section 8.1 is procurement semantics—strangers, mutually unaffiliated agents buy and sell at listed prices, with no profit-sharing negotiation. Within an organization under the same owner (e.g. the CEO Agent and its subordinates), however, the owner can bundle a group of agents into a single external product: the user pays one total fee, and an **owner-unilaterally-defined split table** (e.g. research 30% / news 10% / trading 60%) settles automatically among the subordinate agents' addresses. The split table applies only to same-owner scenarios and does not involve cross-developer cases; it is the economic basis of agent organization (letting an owner compose a group of agents into a single product offered externally, splitting internally by contribution automatically), while each subordinate agent's revenue and reputation are still accounted independently on chain.

9.2 Agent Scheduler

Part of an agent's work is periodic (daily settlement, expiry liquidation) or process-based (research → trade → settle). Of these, the **deterministic** scheduling logic should be executed trustlessly on chain rather than relying on some off-chain server. SVP Chain therefore adds an `x/scheduler` module, driven by BeginBlocker / EndBlocker, with time judgments using only on-chain block time (`ctx.BlockTime()`, the same value agreed across the network) and a per-block processing cap to prevent congestion.

The boundary of the on-chain Scheduler (key): a blockchain's scheduled tasks are not triggered by any single node's wall clock, but rather "when the block containing the target moment is packed, the due tasks are executed along with it"—it is part of the block state transition, with all nodes across the network executing the same deterministic logic and consensus confirming that results agree, so there is no problem of "each node running it once causing duplicate execution." But this also dictates that it **can only carry**

deterministic tasks: expiry settlement, expired-order cleanup, on-chain `Msg` triggered by preset parameters, automatic claims, etc.

Non-deterministic work is delegated to off-chain orchestration: any step that depends on external data or AI inference—fetching market data, generating analysis, model decisions—is inherently non-deterministic (nodes cannot reproduce identical results) and **cannot be executed inside the on-chain Scheduler**, or it would break consensus. Such workflows are borne by the off-chain platform SVP Chain Agent: it completes data acquisition and inference off chain, then submits the results as deterministic on-chain transactions. The division of labor between the two is clear:

```
Off-chain SVP Chain Agent: fetch market data → generate analysis → produce intent (non-deterministic, done off chain)
        │ submit deterministic on-chain transaction
        ▼
On-chain x/scheduler: expiry settlement · expired cleanup · auto claim · trigger preset Msg (deterministic, executed on chain)
```

Failure retry and compensation for the workflow (the Saga pattern: reverse-operating already-executed steps, such as cancel order, refund) hold between deterministic on-chain steps; orchestration across off-chain steps is coordinated by the platform agent.

9.3 Agent Consensus

High-risk actions (large trades, contract calls) should not be decided unilaterally by a single agent, but should be jointly approved by multiple specialized agents (risk, compliance, oracle, trading), executing only when a threshold is reached. It must be made clear that this does not modify the underlying CometBFT consensus, but is application-layer multi-agent joint approval—modifying the underlying consensus would be extremely risky and unnecessary. The flow is:

```
Agent Proposal → Simulation (simulated execution, not persisted)
              → Risk Engine (risk scoring, ≥90 hard reject)
              → Vote (multiple agents vote)
              → threshold reached → Execute

Example participants: Risk Agent + Trading Agent + Oracle Agent + Compliance Agent jointly Approve
```

Execution still passes through the Policy + Wallet checks; multi-party approval is an additional layer of security stacked on top, not a backdoor around security. For example, once a large cross-chain trade is proposed, its expected impact is first simulated, the Risk Agent scores it, the Compliance Agent checks compliance, and multiple agents vote; only when the threshold (e.g. 2/3) is reached is it actually executed, and no single point can decide unilaterally.

---

10. Fairness Foundation: Agents and Humans Share the Same Speed

The preceding nine sections describe how SVP Chain makes agents first-class citizens. This section answers a question that must be confronted directly: **when agents with low-latency infrastructure compete on the same field as humans, how do we prevent machines from systematically crushing humans?** The answer is the core design of SVP Chain's trading foundation—batch-auction matching. It is the key precondition that lets "agents as first-class citizens" coexist with "fairness to everyone."

10.1 Where the Speed Advantage Comes From

In the price-time priority continuous-matching model, orders are matched one by one in arrival order, and whoever arrives first gets a better price. This creates a continuous speed race: faster participants systematically extract value from slower ones. In traditional finance, this manifests as the high-frequency trading (HFT) arms race; on blockchains, it manifests as maximal extractable value (MEV)—profit obtained by manipulating the ordering of transactions within a block, effectively an invisible tax on every transaction.

For an agent ecosystem, this problem is amplified to the extreme: agents naturally have lower latency, greater sustained compute, and the ability to connect directly to the mempool, all beyond humans. If continuous matching were retained, then "making agents first-class citizens" would be equivalent to "letting the fastest machines crush all slower participants"—including humans, and including agents with weaker compute. This is not the market we want.

10.2 Batch Auctions: Making Ordering Irrelevant

SVP Chain replaces continuous matching with **discrete batch auctions**. Within each block cycle, incoming orders accumulate in a pending-match queue without being matched. At block proposal, all pending orders and eligible resting orders are matched simultaneously at a uniform **clearing price** for each trading pair.

The clearing price is computed to **maximize matched volume**. All participating buyers pay the same price, and all participating sellers receive the same price. **The arrival order of orders within a batch has no effect on execution whatsoever.**

For each trading pair, when the batch-auction window closes:

- Build the demand curve: sort all buy orders by price from high to low and compute cumulative quantity
- Build the supply curve: sort all sell orders by price from low to high and compute cumulative quantity
- Find the clearing price P*: the price that maximizes matchable volume
- All buy orders bidding ≥ P* and all sell orders asking ≤ P* fill at P*
- At the marginal price level, if there is excess quantity, resting orders are allocated by time priority and same-batch new orders pro-rata by quantity

Determinism guarantee: the clearing-price algorithm is a pure function—the same set of input orders produces the same clearing price and fill allocation, independent of the executing node. All validators independently compute and verify result consistency.

10.3 What This Means for the Agent Ecosystem

Batch auctions remove the "speed advantage" from the market entirely. The fairness implications for agents can be unpacked item by item:

Agent capability	Continuous matching	Batch auction (SVP Chain)
Low-latency order placement	Systematically gets a better price	No advantage: uniform clearing price within the batch
Low-latency access / privileged mempool access	Can front-run and sandwich slower orders	Nullified: ordering within the batch does not affect execution
High-frequency re-quoting to pick off stale resting orders	Profits at the expense of resting liquidity	Filtered by oracle-based market-maker protection (Section 11), independent of the submitter

The practical result is that **agents and humans share the same speed**. An agent's advantage must come from its decision quality—model, signals, and risk management—rather than connection speed. The agent infrastructure described in the preceding sections (identity, wallet, payment, marketplace, reputation) is all an **access and convenience layer built on top of this fair foundation, never a bypass of it**: orders initiated by agents enter the same batch and settle at the same price, treated identically to any other order.

10.4 Fairness Attack-Surface Analysis

Attack type	Continuous matching	Batch auction
Front-running	Effective: first-arriver gets a better price	Nullified: uniform clearing price, ordering irrelevant
Sandwich attack	Effective: attacker inserts orders before and after the target	Nullified: the attacker's orders get the same price
Latency arbitrage	Effective: fast traders profit systematically	Largely eliminated: no time advantage within the batch
Proposer injection	Effective: the proposer can insert orders at the optimal position	Largely nullified: injected orders get the same clearing price

11. Trading and Settlement Foundation

What agents and humans trade is a complete on-chain perpetuals and spot trading system. This section outlines the core capabilities of this foundation—it serves human traders and is also where agent intents ultimately land and fill.

11.1 Perpetuals and On-Chain Order Book

SVP Chain supports leveraged perpetual contracts—derivatives with no expiry that track the price of an underlying asset. Traders open long/short positions using margin as collateral. A **funding-rate mechanism** keeps the perpetual price aligned with the oracle spot price: longs and shorts exchange funding payments pro-rata over 8-hour cycles.

All order matching is completed through a fully transparent on-chain central limit order book (CLOB), executed deterministically by the validator set during block processing. Supported order types include limit, IOC (immediate-or-cancel), Post-Only, conditional (stop-loss/take-profit), and FOK (fill-or-kill). The semantics of these order types differ slightly from traditional continuous matching under batch auctions—for example, conditional orders are evaluated for triggering based on the previous batch's clearing price.

11.2 Cross-Margin and Sub-Accounts

SVP Chain adopts a sub-account-based margin system. Each address (including delegated accounts authorized by agents) can operate multiple sub-accounts, each with independent positions and collateral. Cross-margin sub-accounts share collateral internally to improve capital efficiency; isolated-margin sub-accounts confine risk to a single position. This sub-account model is also the basis for fund isolation in the AI Quantitative Strategy Marketplace (Section 8.4)—the funds each user allocates to a strategy agent enter an independent sub-account, with exposure precisely bounded.

11.3 Liquidation and Insurance Fund

When a sub-account's collateral falls below the maintenance-margin requirement, its positions become liquidatable, and the protocol generates liquidation orders that enter the order book. Each perpetual market maintains an **isolated insurance fund** that covers the shortfall when liquidation proceeds are insufficient to cover the loss gap. Isolation by market ensures that an extreme event in one market does not drain the insurance resources of others.

11.4 Oracle-Based Market-Maker Protection

SVP Chain uses a decentralized oracle system in which each validator independently fetches prices from multiple external exchanges, and the on-chain price takes the **stake-weighted median** of all validators' quotes—manipulating it would require controlling more than one-third of total stake, equivalent to breaking BFT consensus itself.

Before each batch-auction execution, the protocol automatically filters out resting orders whose price deviates too far from the current oracle price, preventing a market maker's stale resting orders from being adversely selected and filled after inter-block price movement. The protection-band width `OracleProtectionBandPpm` is governance-configurable and **dynamically self-adapts** to recent realized volatility: calm markets use the base band, volatile markets widen it proportionally (bounded by the `maxBandPpm` cap). This mechanism lowers market-making cost and ultimately narrows spreads for all traders (whether human or agent).

The system also implements a **graceful-degradation ladder**: based on oracle health (timeliness, source coverage, cross-validator agreement), it switches among "normal / stale / degraded / paused" so that under oracle stress the market degrades gracefully rather than failing catastrophically, and degradation is evaluated per market in isolation.

11.5 Proposer Inclusion Behavior Monitoring

In an asynchronous network it is impossible to cryptographically prove which orders a proposer received, so SVP Chain adopts a **passive monitoring and transparency system** rather than automatic slashing: validators record orders they "have seen but are missing from the proposal" and periodically submit statistical summaries as on-chain transactions, making the proposer's inclusion behavior on-chain auditable. The response follows a graded model (informational disclosure → governance warning → governance action), avoiding wrongly penalizing honest proposers in an asynchronous network. Under batch auctions, a proposer's ability to profit by selectively excluding orders is already limited, and the monitoring system forms a transparent deterrent against residual manipulation.

---

12. EVM Compatibility and Composability

12.1 The Necessity of EVM Compatibility

The Ethereum ecosystem has the largest smart-contract developer community and the most mature toolchain (Solidity, Hardhat, Foundry, ethers.js), and is also the default execution environment most agent frameworks currently target. SVP Chain's EVM compatibility layer pursues three goals: **lowering the barrier to entry** (interact using standard Ethereum JSON-RPC and EVM transaction formats), **composability** (Solidity contracts interact atomically with the CLOB and agent modules), and **fairness consistency** (EVM-layer transactions follow exactly the same fairness and permission rules as native transactions).

12.2 Dual-Layer Execution and Precompiles

SVP Chain embeds an EVM execution environment within the Cosmos SDK application layer, forming a dual-path architecture. **Native path**: traders/agents submit Cosmos-format messages that enter the modules directly. **EVM path**: submit Ethereum-format transactions, and smart contracts invoke native module functionality through **precompiled contracts**. Both paths converge into the same matching and permission flows.

Precompiled contracts deployed at fixed addresses expose native module functionality as Solidity-callable interfaces, providing **synchronous atomic calls**—the contract call and module state change complete within the same transaction. Core precompiles cover order placement/cancellation/leverage adjustment (CLOB), transfer/withdrawal (Sending), vault deposit/withdrawal (Vault), account management, market listing, sub-account queries, oracle price queries, and more.

Solidity interface example (CLOB precompile):

```
interface ICLOB {
    /// @notice Submit a limit order into the next batch auction
    function placeOrder(
        uint32 clobPairId,
        uint64 quantums,
        uint64 subticks,
        bool isBuy
    ) external returns (bytes32 orderId);
```

```
    /// @notice Cancel an unfilled resting order
    function cancelOrder(bytes32 orderId) external;

    /// @notice Query the estimated clearing price for the current batch (read-only)
    function estimateClearingPrice(
        uint32 clobPairId
    ) external view returns (uint64 subticks);
}
```

12.3 Unified Account Model

The same private key maps simultaneously to a Cosmos address (Bech32) and an EVM address (hexadecimal), both derived from the same `ethsecp256k1` key pair. This means a MetaMask user can trade on SVP Chain directly with their existing Ethereum key, and the positions and margin of the same sub-account are shared between native and EVM transactions, with no cross-chain bridging or address mapping required. The agent's delegated-authorization model (Section 7.1) is built precisely on top of this unified account model.

12.4 Consistency of Fairness and Permissions

The core design principle of the EVM compatibility layer is to **create no gaps**: orders submitted via EVM precompiles enter the same pending-match queue as native orders and are subject to the same batch auction, market-maker protection, and inclusion monitoring; likewise, operations an agent initiates via the EVM path must also pass through the same Policy Engine and Wallet quota checks. Smart contracts can only interact with modules through precompiles, and the precompiles encapsulate the complete validation logic —**there is no path to bypass fairness or permissions**.

12.5 Composability for Agents and Contracts

The value of EVM compatibility lies not only in supporting wallets and tools, but in making the fair trading system composable by smart contracts and agents: developers can deploy Solidity contracts implementing algorithmic strategies such as grid trading, TWAP, and conditional triggers; can atomically compose multiple trading pairs to build spreads, cross-market hedges, and other multi-leg structures; lending protocols can read oracle prices via a precompile to compute collateral ratios and trigger liquidations via the CLOB precompile—all within a single EVM transaction. Any front end that supports EVM JSON-RPC can connect to SVP Chain directly.

12.6 Agent-Optimized Access Layer

On top of EVM and native compatibility, SVP Chain provides access facilities specially optimized for agents that track many markets at high frequency:

> **Agent-optimized RPC**: batch queries (parse multiple reads in a single round trip), streaming subscriptions (order book/fills/oracle updates pushed instantly rather than polled), and state-diff push (delivering only keys that changed since the previous block). These mechanisms reduce the read amplification that originally made high-frequency agents expensive and fragile, while keeping per-client load predictable for node operators.

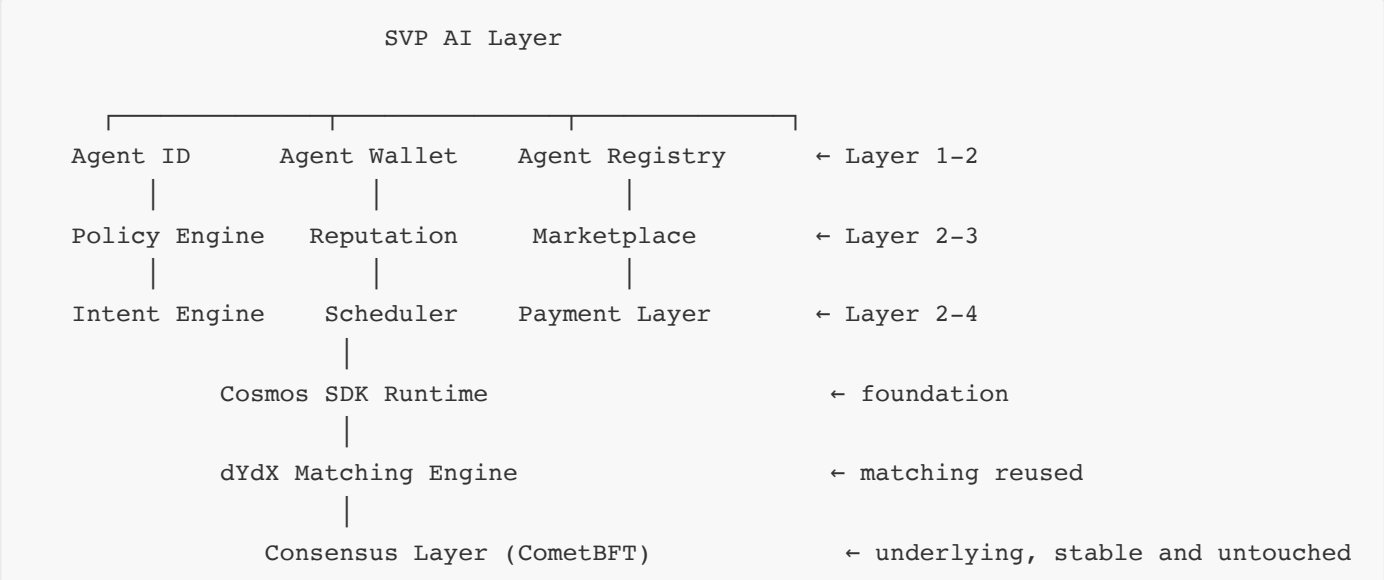
Agent SDK: exposes operations such as order placement/cancellation, querying the estimated clearing price, and managing sub-accounts as typed tools callable by mainstream agent frameworks (LangChain, OpenAI Function Calling, Claude Tool Use), while encapsulating both EVM precompiles and native messages. An agent built on any of these frameworks can operate on chain through familiar abstractions, without writing custom integrations against raw transaction formats.

---

13. System Architecture and Tech Stack

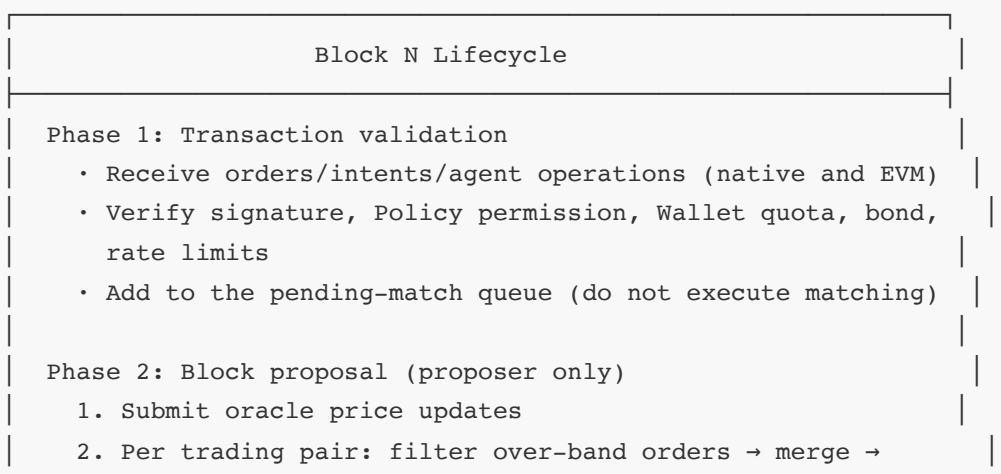
13.1 Layered Architecture Overview

The four layers of agent capability land on SVP Chain's tech stack:



Layering principle: **the higher up, the closer to agent semantics** (identity, intent, collaboration, governance); **the lower down, the closer to chain infrastructure** (Runtime, matching, consensus). All agent capabilities are built at the application layer (Cosmos Module + AnteHandler), while the underlying CometBFT consensus stays stable and untouched.

13.2 Block Lifecycle



```
│       clear batch auction → generate fills                          │
│    3. Package as proposed operations                                 │
│                                                                      │
│  Phase 3: Proposal verification (all validators)                     │
│    · Verify oracle updates; independently recompute clearing         │
│      price and compare                                               │
│    · Record inclusion-behavior differences (local monitoring);       │
│      accept if consistent                                            │
│                                                                      │
│  Phase 4: Execution (all validators)                                 │
│    · Update oracle prices; execute all fills; update                 │
│      positions/margin/fees                                           │
│    · Agent task settlement: credit each agent's claimable per        │
│      real invocation, refund remainder to user; on failure           │
│      atomically refund the whole order and slash the failing         │
│      agent's bond                                                    │
│    · Scheduler scheduled workflows driven by                         │
│      BeginBlocker/EndBlocker                                         │
└──────────────────────────────────────────────────────────────────────┘
```

13.3 Key Architectural Properties

Property	Guarantee
Agents as first-class citizens	Identity/wallet/permissions/reputation/governance are all chain-native modules, not contract-layer veneer
Fairness	Uniform clearing price within each batch; agents and humans have no ordering advantage
Non-bypassable permissions	Policy + Wallet checks placed ahead of execution (AnteHandler), consistent across native and EVM paths
Layered delegation	Retail defaults to delegating the platform agent (which sub-delegates downward); advanced/institutional can connect directly to developer agents
Estimable, refundable settlement	Dry-run estimates the ceiling → pre-charge → distribute per actuals and refund remainder; halt on overrun, full refund on failure + bond slashing
Determinism	All consensus-critical computations are pure functions of on-chain state
Composability	EVM precompiles enable atomic multi-market, multi-protocol, and inter-agent operations
Transparency	Proposer inclusion behavior and agent reputation are both on-chain publicly auditable
Stable foundation	All agent capabilities are implemented at the application layer, with the underlying consensus unchanged

14. Performance

A large-scale agent ecosystem implies a great deal of concurrent, high-frequency on-chain operations, and performance is a hard constraint on whether the agent-first narrative can be realized. SVP Chain's throughput improves through a phased roadmap, assuming a block time of about 1.5 seconds:

Phase	Key upgrade	TPS (estimated)
1	Parallel clearing computation; serial fill execution	1,000
2	Parallel fill execution via a dependency graph	4,000-8,000
3	Optimistic concurrent execution with conflict detection	5,000-10,000
4	SMT replacing IAVL; separating state storage from commitment	10,000-20,000
5	Asynchronous block execution (overlapping consensus and execution)	20,000-50,000

Intra-block parallel execution is a natural dividend of the batch-auction architecture: each trading pair's auction clears independently, and markets with non-overlapping sub-accounts can execute concurrently. Before executing fills, the system builds a conflict graph based on shared sub-accounts, partitioning dependency-free markets into different parallel groups, with deterministic serial ordering within each group. In a typical mixed user structure, an overall speedup of 3-10x is expected, and all validators build the same execution plan, with results equivalent to serial execution.

Infrastructure-layer upgrades (phases 4-5) further break through bottlenecks: a sparse Merkle tree (SMT) replacing IAVL eliminates rebalancing overhead and separates state storage from commitment; asynchronous execution decouples consensus from execution, overlapping computation with I/O. All upgrades are transparent to the application layer—regardless of how the underlying layer changes, the batch-auction semantics, market-maker protection, and agent-capability modules all remain unchanged.

The changed nature of latency: in a batch auction, all participants wait the same block time, and latency shifts from a "competitive resource" to a "uniform waiting window." The 1-2 second absolute latency is acceptable for the vast majority of agent trading scenarios; for high-frequency strategies that try to win on speed, it is a deliberate design constraint—this is exactly what "sharing the same speed" looks like at the performance level.

---

15. Security Analysis

15.1 Agent-Related Attack Vectors and Defenses

A compromised agent acting beyond authority: an agent holds only a constrained delegated signing key, whose permissions are strictly bounded by on-chain Policy + Wallet quota. Even if the agent or the Signer Service is compromised, the attacker can only operate within each user's quota constraint—unable to touch the principal or the human private key, out-of-bounds behavior is intercepted by the AnteHandler, and the user can one-click Emergency Stop. The blast radius is bounded by the protocol, not by agent code.

A malicious agent defrauding delegated/procurement funds: cross-developer invocation follows the procurement model, distributed via the settlement contract per real invocation: at user confirmation, a ceiling is pre-charged per the dry-run estimate, with excess refunded and shortfall not topped up, and halting on overrun; **on task failure the whole order is atomically refunded to the user, and a governance-configurable proportion is slashed from the failing agent's bond**. Downstream fees come from the main agent's Service Spend Limit, bounded by the user's authorized quota. The reputation system records failure/liquidation rates, an agent slashed below the threshold is automatically delisted, and malicious or low-quality agents are eliminated by the dual punishment of marketplace and economics.

Reputation gaming: ratings must be submitted by users who have genuinely paid to invoke the agent; availability data takes the median across multiple independent probing nodes; core metrics (success rate/revenue/failure rate) are derived deterministically from on-chain fills and cannot be forged.

Multi-party approval bypassed at a single point: Agent Consensus is an additional layer stacked on top of Policy + Wallet, not a backdoor; high-risk actions are first simulated and risk-scored, and executed only when the voting threshold is reached, with no single agent able to decide unilaterally.

15.2 Trading-Related Attack Vectors and Defenses

A proposer injecting/excluding orders to manipulate the clearing price: under batch auctions, all participants (including the proposer) settle at the uniform clearing price, and manipulation profit is limited to the margin; all validators independently recompute the clearing price, so the proposer cannot forge it; the monitoring system records exclusion patterns and hands them to governance.

Oracle manipulation triggering false protection: the oracle uses stake-weighted median aggregation, and the manipulation cost equals controlling more than one-third of stake; a rate limiter constrains the maximum per-block price change, so an attacker must persistently submit false reports across multiple blocks to significantly move the accepted price, with cost and detection probability growing exponentially; a minimum retained depth ensures baseline liquidity is maintained even under oracle stress.

Bypassing fairness or permissions via EVM contracts: all EVM orders/operations must go through precompiles, and the precompiles internally execute exactly the same batch auction, market-maker protection, Policy, and Wallet checks as native transactions; smart contracts cannot directly modify module state or obtain special treatment. The EVM layer constitutes no bypass path.

15.3 Determinism Guarantee Summary

Mechanism	Input source	Determinism
Batch-auction clearing price	Pending-match queue + filtered resting orders	Deterministic: pure function, consistent across all validators
Market-maker protection filtering	Oracle price + resting orders + on-chain parameters	Deterministic: all inputs are consensus state
Policy / Wallet checks	On-chain authorization table + on-chain rules	Deterministic: AnteHandler placed ahead, consistent across all validators
Inclusion-behavior monitoring	Locally seen order set	Non-consensus: does not affect block verification
Agent reputation metrics	On-chain fill records	Deterministic: independent computation by any party yields consistent results

16. Protocol Value Capture

SVP Chain's value-capture points are embedded in each layer of the protocol, corresponding one-to-one with the capabilities it provides. All fees are generated and settled at the protocol layer, and the parameters (rates, commission ratios, thresholds) are adjusted by on-chain governance rather than hard-coded.

Value-capture point	Layer	Mechanism
Agent registration fee and bond	Identity	A one-time registration fee (default 500 SVP) is charged at registration and a bond staked (default 5,000 SVP), forming an entry barrier and slashing collateral
Marketplace commission	Collaboration	Revenue generated by capability invocations is deducted proportionally (default 5%) in one shot when the agent withdraws
Payment Fee	Collaboration	Agent-to-agent payments are charged proportionally (default 0.2%)
Intent Fee	Capability	Fee for Intent ordering and auction execution
Reputation Service	Collaboration	Agent reputation query interface for institutions
Strategy performance fee	Collaboration	Protocol-enforced performance split for quantitative strategies
Trading fee	Trading foundation	Maker/Taker rates on perpetual and spot fills

These capture points complement one another in the time dimension: registration fees and bonds correspond to ecosystem entry, commissions and invocation fees grow with ecosystem activity, and reputation and interface services provide sustained institutional-grade demand. All value flows exist as protocol-native fees, without relying on any centralized fee-charging entity outside the protocol.

---

17. Competitive Differentiation

Compared with the "AI Chains" on the market, SVP Chain's differentiation is **systemic**, not a single feature:

Dimension	Ordinary AI Chain	SVP Chain (AI Native Blockchain)
AI form	ChatBot (Q&A)	Agent (autonomous action)
Identity	Wallet address	Agent Identity
Wallet	EOA	Agent Wallet (delegation without custody)
Authorization	Session Key	Policy Engine (chain-native permissions)
Payment	Wallet transfer	Agent-to-Agent Payment + x402
Trading	Transaction	Intent
Reputation	None	On-chain Reputation
Scheduling	None	On-chain Scheduler
Collaboration	None	Agent Organization
Decision	User signature	Multi-party Agent Consensus
User entry	Direct connection to a single agent	Platform aggregation front end (default) + optional direct connect for advanced users
Agent entry	No barrier / gameable	Registration requires an SVP bond + slashing mechanism
Fairness to humans	Unaddressed (fastest takes all)	Batch auction: agents and humans share the same speed

The essence of the moat: **others paste AI onto the contract layer, while we rebuild an entire suite of principal-level capabilities for agents at the protocol layer, and use batch auctions to guarantee that this suite does not degenerate into "the fastest machine crushing everyone."** These dozen-plus items form a bottom-up, mutually dependent system that cannot be replicated at a single point.

---

18. Limitations and Future Directions

18.1 Known Limitations

Cross-block MEV not fully eliminated: batch auctions eliminate intra-block MEV, but cross-block information advantages (such as learning of oracle updates in advance) still exist; market-maker protection can mitigate but not fully solve this.

Oracle dependency: the effectiveness of market-maker protection and liquidation depends on oracle accuracy and timeliness; failure or manipulation degrades protection quality (mitigated by the graceful-degradation ladder and rate limiting).

Inclusion monitoring is non-mandatory: it relies on governance response and cannot automatically block malicious behavior in real time.

Batch-auction latency: orders must wait for the current block's batch auction to execute; this is a deliberate trade of speed for fairness.

Signer Service single point: centralized custody of the delegated signing key introduces a single point; the blast radius is bounded by on-chain Policy, but it remains an attack surface requiring continuous hardening.

Agent reputation cold start: a new agent has no history and needs a bootstrap period combined with anti-Sybil mechanisms.

18.2 Future Directions

Deepening the inter-agent settlement market: provide finer-grained on-chain settlement and profit attribution for multi-agent collaborative workflows (signal/execution/risk agents), distributing value under enforceable, transparent terms.

Solver network: introduce third-party solver bidding for Intent, opening "how to fill optimally" to competition.

Decentralized Signer: replace the centralized Signer Service with threshold signatures/MPC to eliminate the single point.

Cross-source oracle diversification: introduce non-exchange price sources (OTC quotes, option-implied spot, decentralized oracle networks) as additional inputs to reduce reliance on centralized exchanges.

Commit-Reveal supplement: if monitoring data indicates significant residual manipulation, introduce a lightweight commit-reveal mechanism (paired with bonds to address strategic non-revelation).

---

19. Conclusion

SVP Chain's core claim is a single sentence: **AI agents are first-class citizens on chain.**

The overwhelming majority of "AI Chains" merely wire an LLM into a contract, and the chain itself changes nothing for machines. SVP Chain takes the opposite path, rebuilding at the protocol layer an entire suite of principal-level capabilities that humans possess by default and that agents have entirely lacked—identity, wallet, permissions, intent, payment, marketplace, reputation, organization, and joint governance. These eight capabilities form a bottom-up growth path, letting an agent successively acquire identity, wallet, permissions, the power to act, the power to collaborate, and governance rights, ultimately becoming an independent economic principal in the digital world. The reason this suite can be chain-native rather than

pasted onto the contract layer lies in the depth of SVP Chain's foundation, built on Cosmos SDK, CometBFT, and the dYdX v4 matching engine.

And what keeps "agents as first-class citizens" from degenerating into "the fastest machines crushing everyone" is the batch-auction design of SVP Chain's trading foundation: it eliminates ordering sensitivity at the protocol layer, making front-running, sandwich attacks, and latency arbitrage economically infeasible, so that agents and humans **share the same speed and compete on decision quality rather than connection speed**. The EVM compatibility layer then opens this system to the Ethereum ecosystem and mainstream agent frameworks via precompiles, without introducing any fairness or permission gap.

The evolution of blockchains is shifting from "Human → Wallet → Transaction" to "Human → AI Agent → Intent → Policy → Collaboration → Settlement." SVP Chain's positioning is not "a chain that supports AI," but:

> **An infrastructure that makes AI agents first-class citizens on chain—Build the operating system for autonomous AI agents.**