PROMPT CACHE IS ARCHITECTCTURE

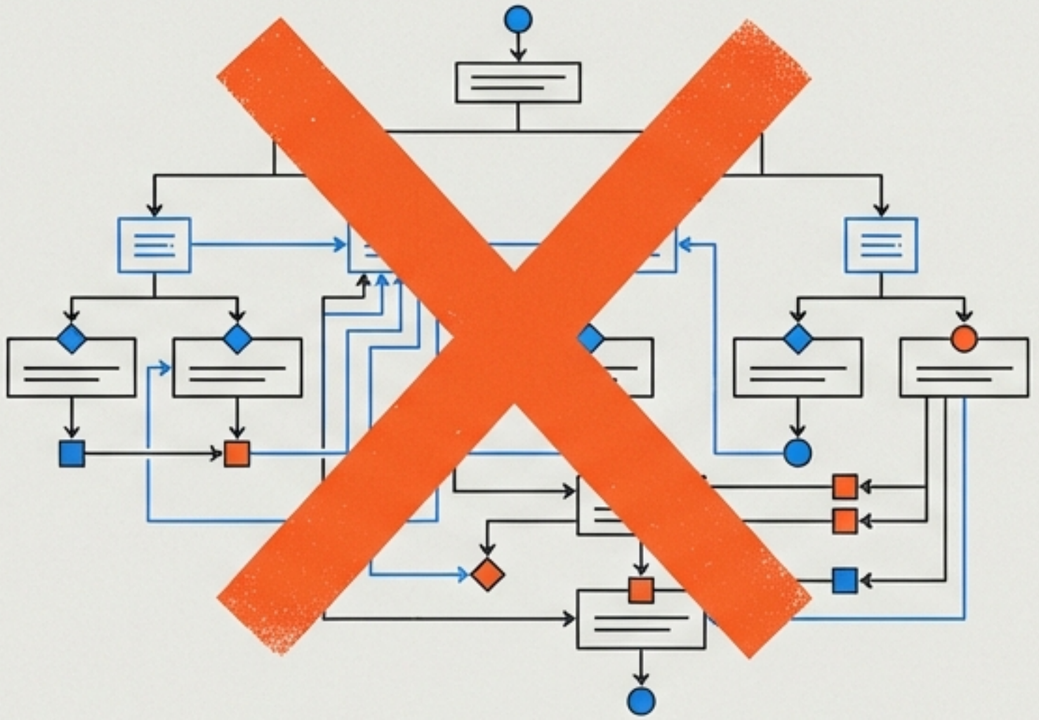
Why the most critical component of modern agent runtimes is disguised as a cost optimization.

Most builders treat prompt cache like a CDN

THE DEFAULT MINDSET

Helpful when it hits. Nice for cost. Faster loads.

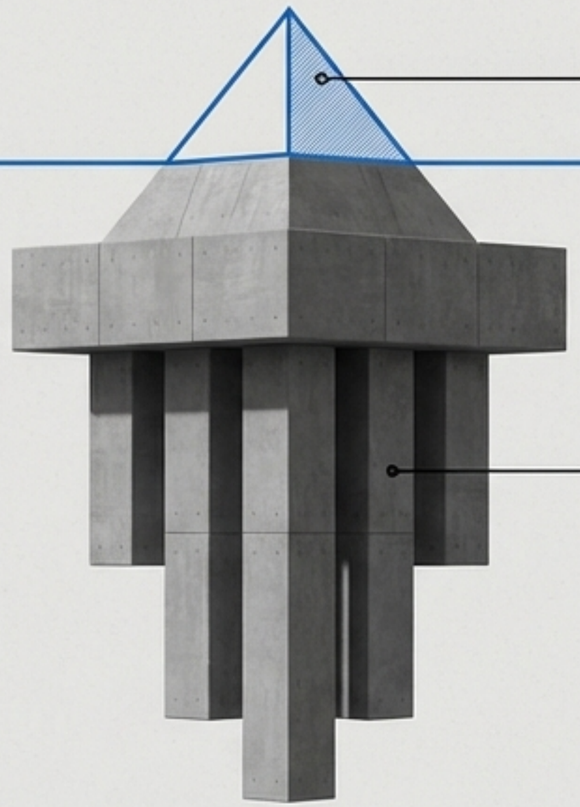
CDN OPTIMIZATION



This intuition works for short, isolated tasks.
It fundamentally breaks the moment your system requires:

Long sessions	Repeated context reuse
Background agents	Subagents & multi-agent fan-out

The true cost is dragging context back into the room



10%

90%

Model Output / Final Answer

The Prefix: The repeated, expensive context dragged into every request.

Agent cost isn't mostly about how many tokens the model consumed.
It's about how much repeated expensive prefix you force the system to rebuild.

A tear-down of the Claude Code snapshot reveals a structural obsession with cache

Not the permission system.

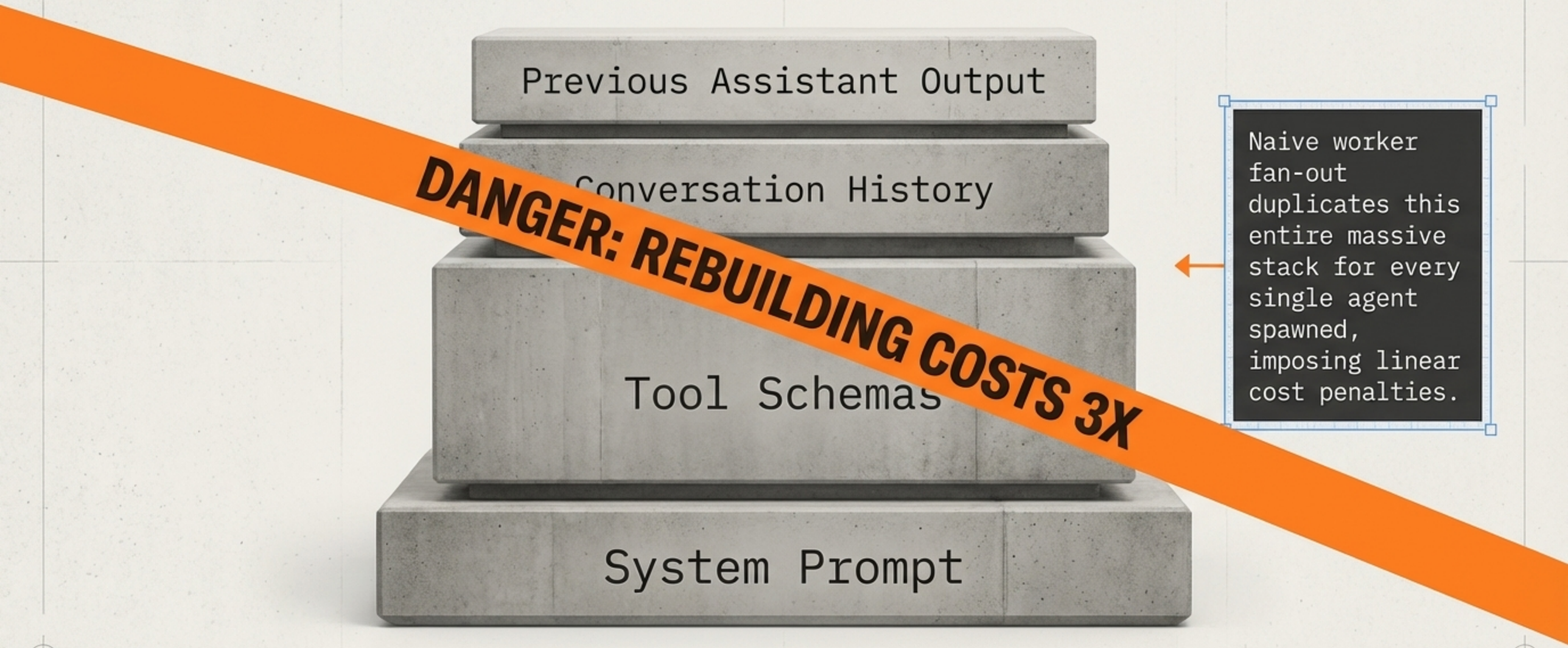
```
async fn handle_request() { ... }
    let context = build_context(); ...

    // Permission check sequence
    // Fork Worker Routine
    if (!check_permissions()) return; ...
    // Tool pipeline execution
    let tools = resolve_tools(); ...
    // Tool workers_execuration | ...
```

Not the tool pipeline.

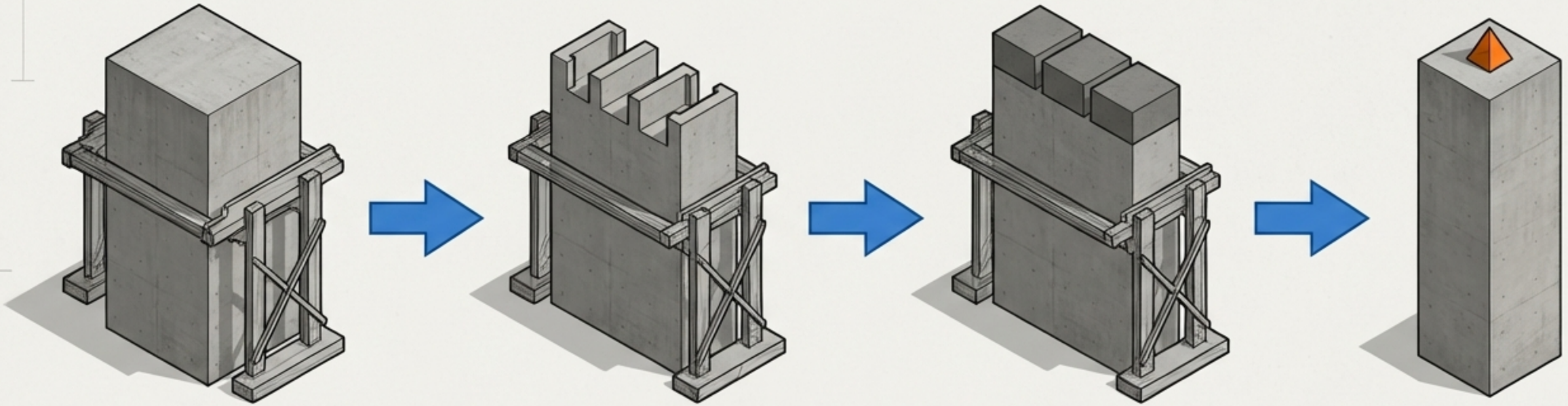
It was the forking logic.

The runtime goes to extraordinary lengths to ensure forked workers share a **byte-identical request** prefix with the parent. This isn't a late-stage optimization—**it's the core architecture.**

Anatomy of an Expensive Context Prefix



The Weird Trick for byte-identical forking



1. Keep full parent message.

2. Construct tool_result placeholders.

3. Inject exact same text.

4. Append child-specific directive at the end.

Forked children don't get fresh instructions.
They get identical histories with tiny, localized variations.

Equivalent is not equivalent enough

Semantically Similar

CACHE MISS
(Rebuilds Entire Stack)

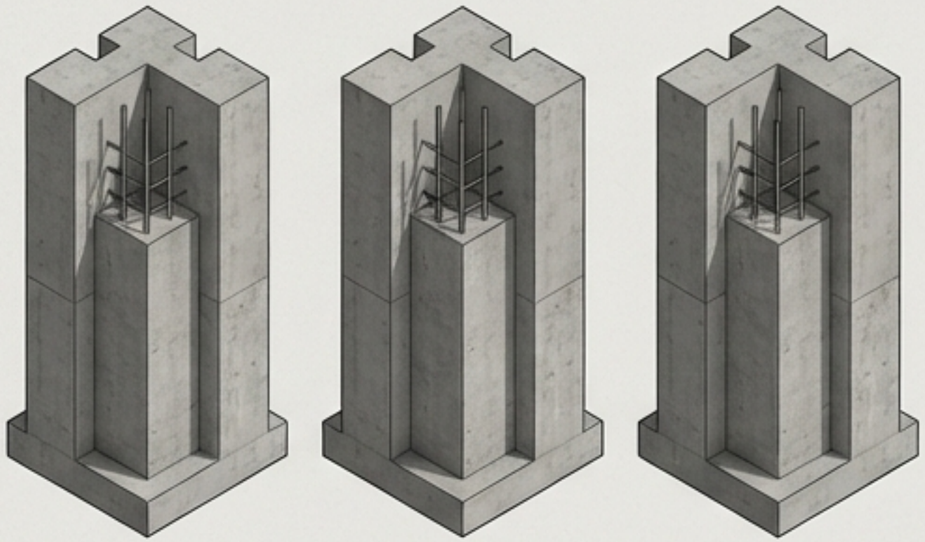
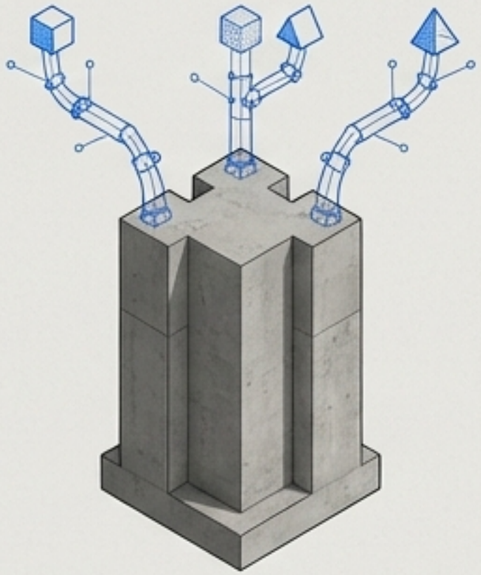
Byte-Identical

CACHE HIT
(Massive Resource Leverage)

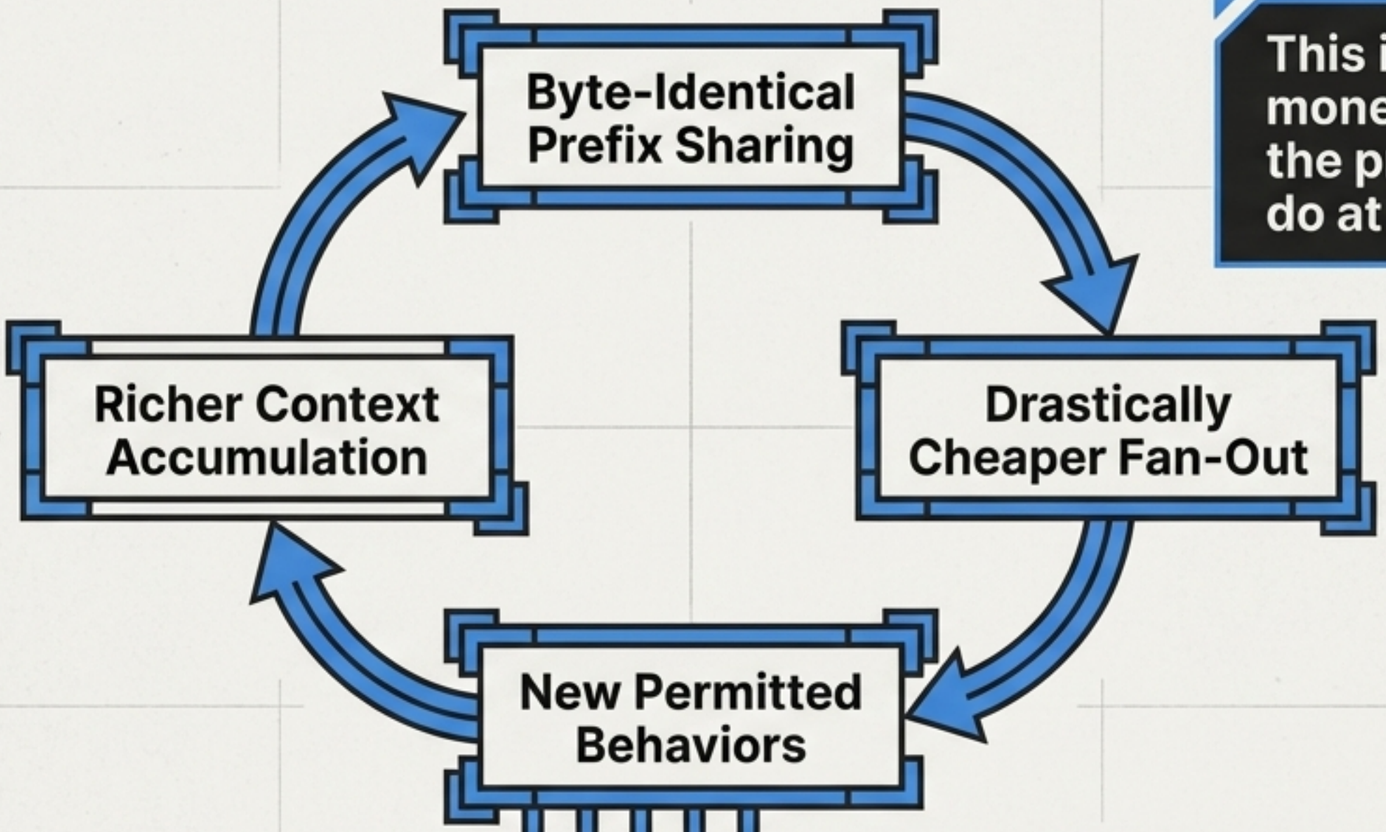
The runtime demands byte-identical parity, not semantic similarity.
The exact wire format must be preserved across multi-agent forks.

The wire economics of delegation

Deploying 3 Sub-Agents

Naive “Spawn a Worker”	Cache-Aware Forking
	
Linear Cost (3 agents = 3x massive prefix cost)	**Marginal Cost** (3 agents = 1x prefix cost + tiny suffix delta)
Economically prohibitive for complex workflows	**Enables extreme multi-agent scale**

Cache economics dictate product capabilities



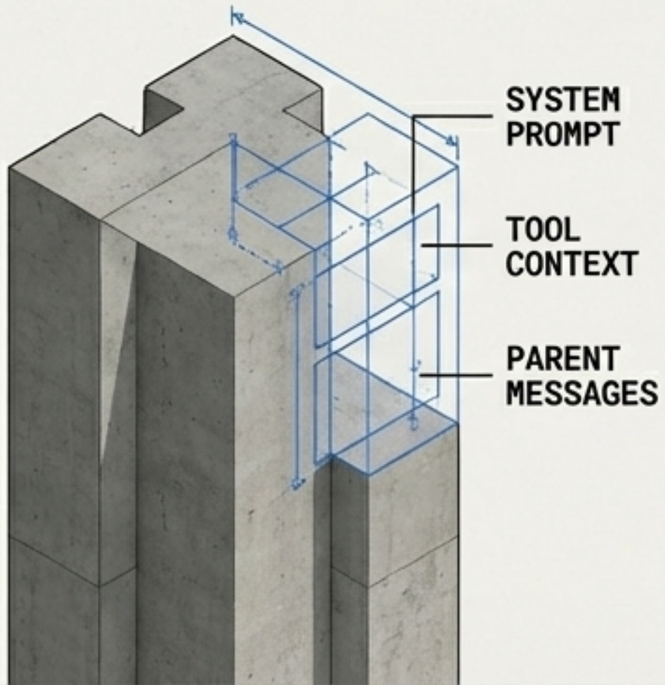
This isn’t about saving money. It’s about what the product is allowed to do at all.

- Spawn sidecar agents
- Summarize background workers
- Extract memory continuously
- Compact via summarizers
- Long-lived branching workflows

Cache identity is a system-wide architectural discipline

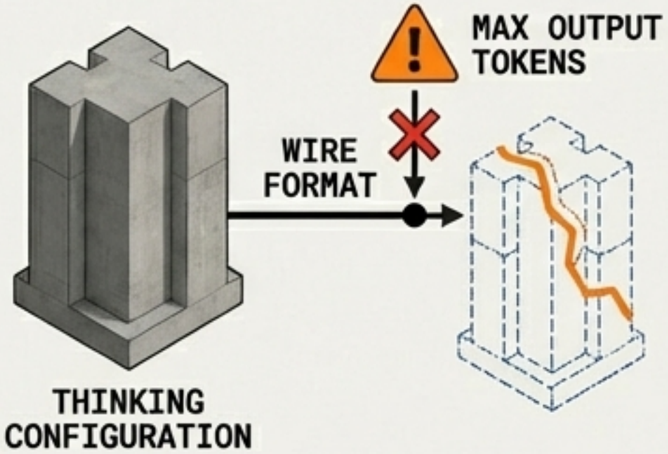
CacheSafeParams

Explicit typing formally models the cache-critical portion of requests (system prompt, tool context, parent messages).



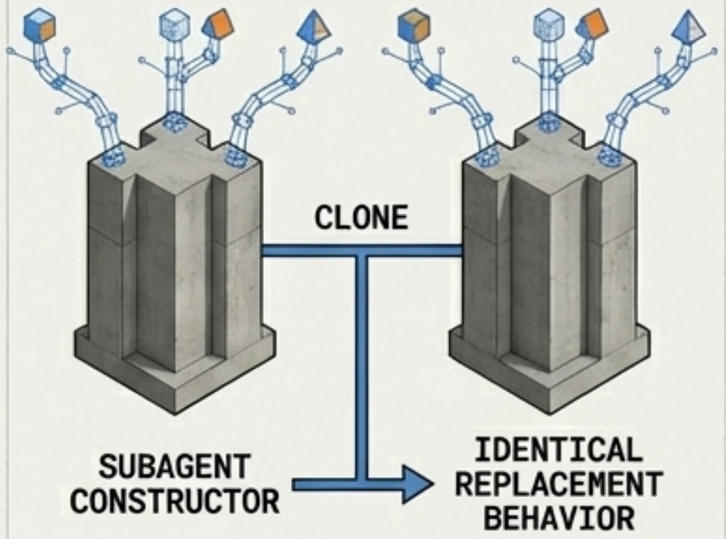
Compaction Logic

Code explicitly warns against setting max output tokens here, as altering the thinking configuration alters the wire format and destroys cache identity.

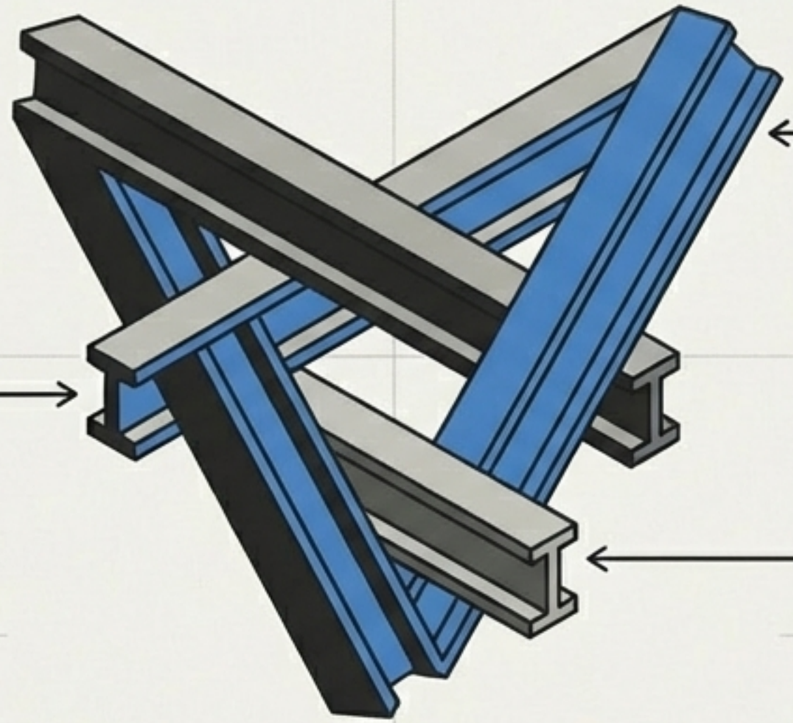


Subagent Context Cloning

Subagent constructors are engineered specifically to preserve identical replacement behavior for tool results.



Worker topology is downstream from cache strategy



Orchestration Design
(The worker topology)

Cache Design
(The identity preservation)

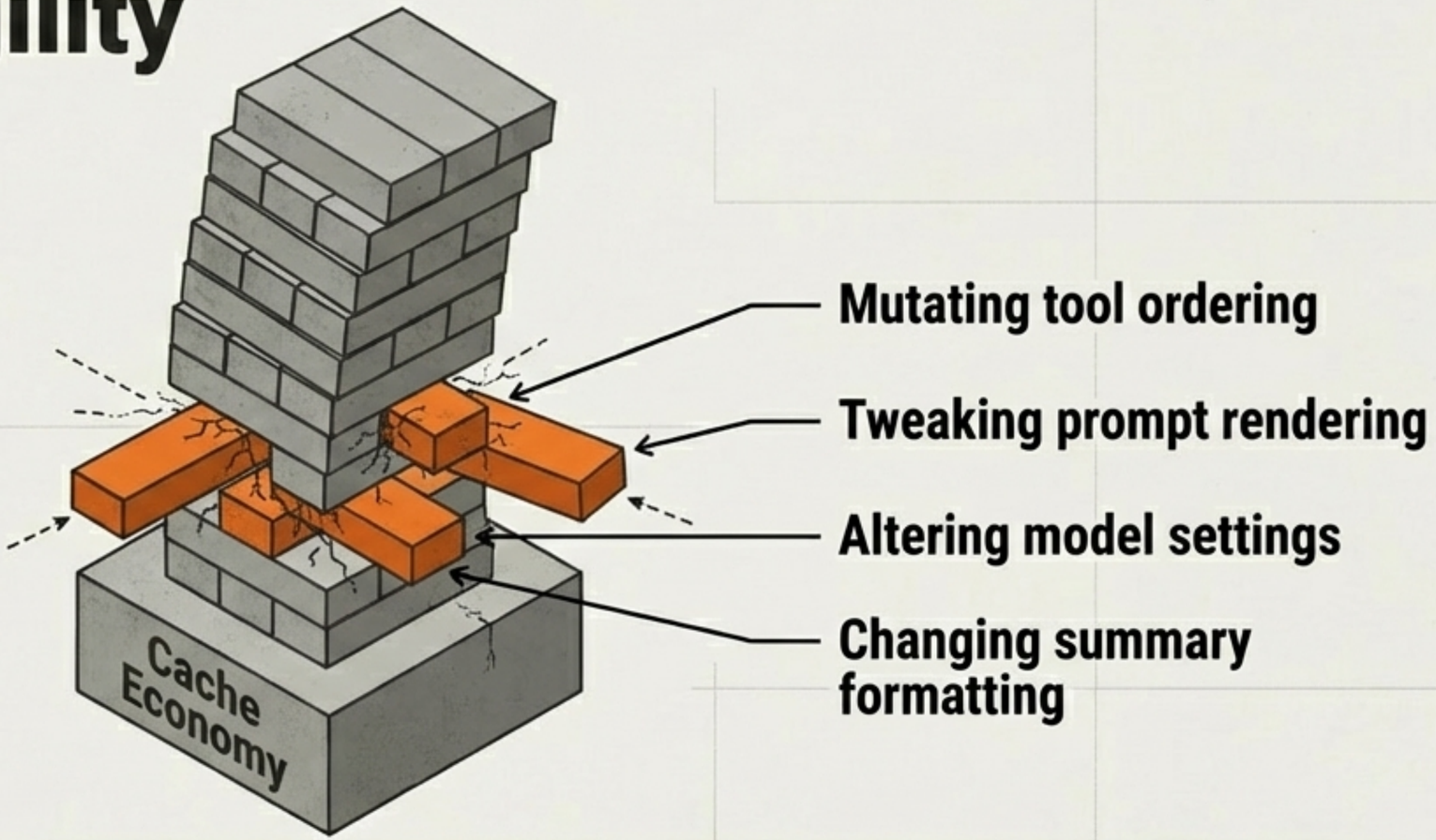
Context Design
(The wire format)

You cannot separate these three layers. The moment you add `subagents`, your `orchestration design` physically alters your `context footprint`.
The three gears turn as one single system.

A structural shift in mental models

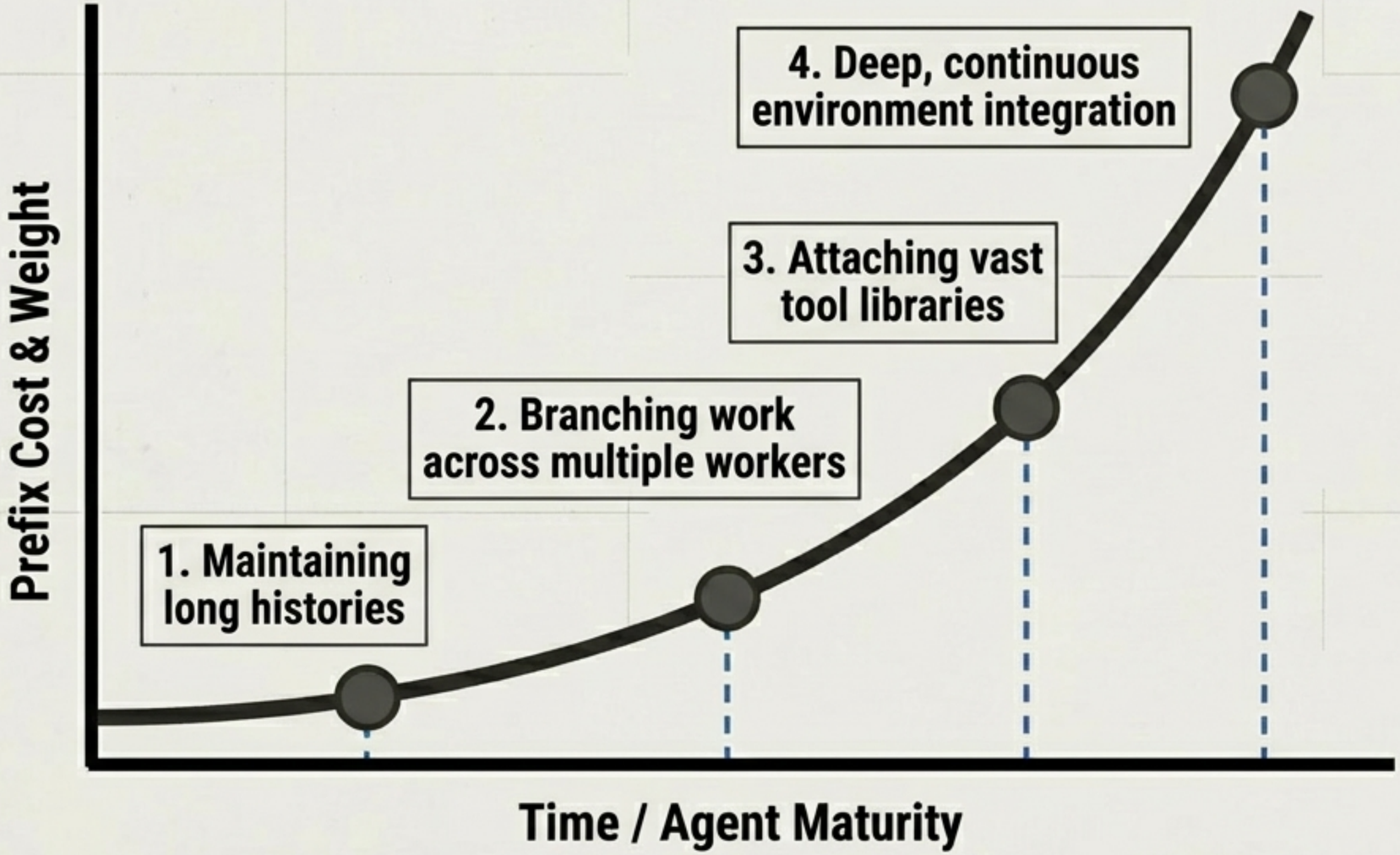
Dimension	Old Intuition	New Intuition
Analogy	CDN	**Foundation**
Role	Incidental bonus	**Structural resource**
Primary Goal	Save money, speed up loads	**Enable multi-agent fan-out**
Impact of a Miss	Wasted money	**Destroys viable workflows**
When to Design	Late-stage optimization	**Day-one architecture**

The Trade-Off: High leverage brings high fragility



Seemingly harmless refactors can quietly `destroy` the `cache` behavior.
The runtime acts like a `distributed system`: small mismatches matter,
and “equivalentrr” is never equivalent enough.

Why this pattern will dominate the future of AI

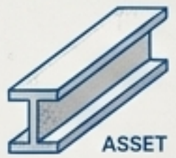


As prefix costs rise, systems will look less like “prompt engineering products” and more like **cache-preserving execution engines** with a model inside.

The playbook for agent architects

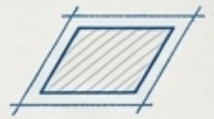
1

Treat expensive prefixes as **reusable assets**, not ephemeral strings.



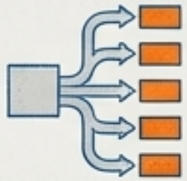
2

Identify **invariant request zones** that must remain identical for reuse.



3

Design **worker fan-out** so only the smallest suffix varies.

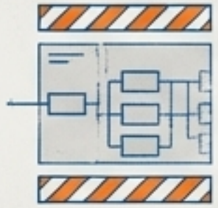


4

Assume **extreme fragility**: build testing around cache-breaking mutations.

3

Design **worker fan-out** so only the smallest suffix varies.



5

Expose **cost structure** to the runtime, not just to finance dashboards.

Not glamorous. Just necessary.

From the outside, prompt cache looks like an optimization. From the inside of a serious runtime, it is a budget. It dictates exactly how much branching, memory maintenance, and background work your product is allowed to do. It determines if your system scales from a neat demo to something you can actually afford to run.