×6.7

CACHE · ROUTE · DELEGATE · PLAN

Claude Code FinOps From Zero to Hero.

THE BRIEF, IN THREE BEATS

01 Caching is the lever — all else amplifies it.

02 Opus is a tool, not a default.

03 Plan once, execute cheap.

The $100 plan doesn't have a limit — your architecture does.

For Engineering Leaders & Platform Architects
Built on field data: 8.83B tokens / month / $100 plan

20 slides · 6 levers · 6 anti-patterns
Cost-modelled, not aspirational

SLIDE 02 · HOOK

A $100 plan, billed at API rates, is $18,200 of work.

The other ×182 isn't a discount — it's architecture.

Source: its-yosi.com, "Architecture for working with Claude Code" — 8.83B tokens / month, Max 5x plan, 95.7% cache rate.

FINOPS · ENGINEERING CLAIM

Claude Code cost is architectural, not arithmetical. The same volume of work runs at $18K or $136K depending on six engineering decisions — not on the price card.

01 The cost driver isn't requests — it's **re-processed context** on every turn.

02 Six levers cut that re-processing 80–95% — without doing less work.

03 Six anti-patterns silently undo every lever. You'll recognise them.

THE FIELD BENCHMARK

One developer, Max 5x ($100/mo), built a Rust migration + a game engine + 970 art assets in one month.

Tokens processed	8.83B
Cache hit rate	95.7%
Fresh-token leverage	×6.7
Off-Opus routing	54.6%
5-h limit hits	1 / month

Source: its-yosi.com, "Architecture for working with Claude Code" (2026) · methodology validated against Anthropic enterprise consumption norms ($150–250 / dev / month).

ANTHROPIC PRICE CARD · Q2 2026

Opus is 5× Sonnet on input, 5× on output.

MODEL	INPUT $ / MTOK	OUTPUT $ / MTOK	USE
Haiku 4.5	$1	$5	CRUD, scripts, mechanical scans
Sonnet 4.5 / 4.6 SWEET SPOT	$3	$15	Code, bug fix, refactor, ~80% of tasks
Opus 4.5 — 4.8	$5	$25	Architecture, multi-file reasoning only
Cached input (any model)	~× 0.1	–	Re-served at ~10% of fresh input price

SUBSCRIPTION TIERS

Pro	$20 / mo · 5-h window
Max 5x	$100 / mo · 5× Pro budget
Max 20x	$200 / mo · 20× Pro budget

Two limits, both rolling: **5-hour window** & **weekly cap** on active compute hours.

The price card is fixed. What you send through it isn't.

Sources: platform.claude.com/docs/en/about-claude/pricing · support.claude.com Max plan article (Q2 2026) · finout.io Claude Code pricing 2026.

Same workload, four cost worlds. Pick the one you're in.

FLOOR · OBSERVED

$18.2K

Real mix + caching

Cache: 95.7%

Routing: 54.6% off-Opus

Paid: $100

What good looks like — ×182 leverage.

OPUS + CACHE

$24.2K

Opus-only routing

Cache: 95.7%

Routing: 0% off-Opus

Penalty vs floor: +33%

Caching saves you. Routing would save more.

MIX · NO CACHE

$106K

Architectural failure

Cache: ~0%

Routing: kept

Penalty vs floor: ×5.8

Long uncleared chats land here.

CEILING · OPUS ONLY, NO CACHE

$136K

Worst-case API spend

Cache: 0%

Routing: 0%

Penalty vs floor: ×7.5

And you'd hit the 5-h limit by 10 AM.

Source: its-yosi.com, "Architecture for working with Claude Code" (2026) · same 8.83B-token workload priced under each scenario.

OFFICIAL ANTHROPIC MECHANISM

#01 Pay once, re-serve forever — cache the stable context.

WHY IT DOMINATES

95.7%

of input tokens served from cache in the field benchmark — **×6.7 effective leverage**. Every turn re-sends the whole history; cache means only the new turn pays full price.

THE MOVE

Stable on top. Put system prompt, `CLAUDE.md`, tool defs, and frozen reference docs at the start — they cache.

Volatile at bottom. User turn changes per call — that's the only fresh-cost token block.

5-min TTL. Cache lives ~5 min idle — keep sessions warm.

Plan files are gold. Long planning doc + 30 short execution turns = 84%/97% cache rate measured.

THE TRAP

Re-ordering invalidates. Move a single token before the cached suffix — the whole cache misses.

Long idle = cold start. Leave a session for an hour, pay full freight on resume.

Auto-compact rebuilds it. When compaction rewrites history, the new prefix is fresh until it warms up again.

No visible meter. You can't *feel* cache misses — they only show in the bill.

Sources: platform.claude.com/docs prompt-caching reference · its-yosi.com field measurement (95.7% cache, ×6.7 leverage on 8.83B tokens).

LEVER 2 OF 6 · ROUTE BY TASK — OPUS IS A TOOL, NOT A DEFAULT

5× ON INPUT · 5× ON OUTPUT

#02 Match the model to the verb — not the project.

VERB	MODEL	RELATIVE COST	EXAMPLE
Scan / lint	Haiku 4.5	× 1.0	grep, file listings, type imports, dead-code sweep
Write / fix code	Sonnet 4.5	× 3	~80% of all tool calls land here — the sweet spot
Architecture / plan	Opus 4.5+	× 15	migration design, multi-file refactor, hard debug
Mechanical exec	Local model	× 0	image gen, build runs, tests — no token cycle

THE MOVE

Default Sonnet. `claude --model sonnet` in shell alias.

Opus is opt-in. Switch only when you reach for architecture or hard reasoning — back to Sonnet after.

Sub-agent the noise. Push grep/scan/list to Haiku-backed sub-agents (lever #3).

Field result: 54.6% of tasks ran off-Opus on the $100 plan that built two systems.

Sources: Anthropic price card (Q2 2026) · its-yosi.com field benchmark (Haiku 6,757 mechanical scans, 54.6% off-Opus).

CONTEXT ISOLATION

#03 A sub-agent eats 1.2B tokens. Returns 0.4% to the main thread.

THE NUMBERS

0.4%

714 sub-agent invocations processed 1.2 **billion** input tokens last month — and returned only 0.4% of that volume to the main conversation as a distilled summary.

WHY IT SAVES

The main thread re-sends its history every turn. Anything in it is paid again, forever.

Sub-agent reads: 40 files, audits 600-line diff, scans a folder.

Main thread sees: "3 issues in auth.py: A, B, C."

Net effect: noise stays in the disposable context; the expensive cached context stays lean.

THE MOVE

Delegate the verb "explore". Anything that reads many files belongs in a sub-agent.

Force a typed return. Ask for "3-bullet summary" or "list of file:line refs" — not prose.

Cheap model inside. Sub-agent can run Haiku/Sonnet even when the main thread is Opus.

Sequential is fine. The win is context isolation; parallelism is a bonus.

Sources: code.claude.com/docs sub-agents reference · its-yosi.com (714 invocations, 1.2B in, 0.4% returned).

LEVER 4 OF 6 · PLAN ONCE WITH OPUS · EXECUTE CHEAP WITH SONNET

CACHE RATE 97% ON EXEC

#04 Front-load the thinking. The execution becomes cache hits.

THE PATTERN · TWO PHASES, TWO MODELS

Big task → one long planning doc on Opus, with everything decided. Then many short execution turns on Sonnet, against that doc.

```
Phase 1 — plan (Opus, /plan mode):
  Output: plan.md, 200–400 lines.
  Covers: contracts, edge cases, file list, tests.

Phase 2 — execute (Sonnet, fresh chat):
  Context: system + CLAUDE.md + plan.md = cached.
  Each turn: "Implement step N of plan.md".
```

Measured: planning chats ran 84% cache; execution chats against them ran 97% cache.

WHY IT COMPOUNDS

Hard thinking happens once, against the whole problem. Each execution turn rides cached plan tokens — cheap input, focused output, less rework. **Same dollar, less churn.**

THE TRAP

Skipping the plan doc and "just iterating." Every iteration re-discovers what a 30-minute plan would have decided once — on the expensive model, with no cache benefit.

Sources: Claude Code /plan mode reference · its-yosi.com (Rust migration: ~17,800-line plan doc, 165 compat scenarios, then execution at 97% cache).

LOADED ONCE PER SESSION

#05 Stop repeating yourself — codify it once in CLAUDE.md.

A MINIMAL CLAUDE.MD

```
# Stack
- Python 3.12, FastAPI, Postgres 16, pytest.
- Frontend: Next.js 15, TypeScript strict.

# Conventions
- Tests: pytest, no fixtures, parametrize.
- Imports: ruff, no star imports.
- Errors: typed, never bare except.

# What NOT to do
- No explanations unless I ask.
- No "here's the updated file" preambles.
- Return diffs, not whole files.
- No try/except unless real recoverable case.

# Commands
- Tests: `make test`.   Lint: `make lint`.
- Run dev: `make dev`.   Migrate: `make db`.
```

WHY IT SAVES

Sits at the top of context, gets cached on first turn, served at ~10% cost on every later turn. Replaces the “don't add comments / use pytest / no explanations” preamble you'd otherwise paste in every chat.

THE TRAP

A 5,000-line CLAUDE.md inflates every cached turn and pushes useful content out of context. Keep it under ~150 lines — rules and pointers, not the codebase itself. Files belong in `@filename` imports.

Sources: code.claude.com/docs memory reference · Anthropic engineering blog, "Claude Code best practices" (2025).

LEVER 6 OF 6 · SLASH COMMANDS — THE CHEAP PROMPT, IN A FILE

.CLAUDE/COMMANDS/

#06 ~80 tokens, not ~200 — over 1,000 calls that's 120K saved.

.CLAUDE/COMMANDS/FIX.MD

```
You are a senior engineer. Given the failing test below:

1. Locate the root cause — one file, one symbol.
2. Return a unified diff. No prose.
3. No try/except — only structural fixes.
4. No comments unless they explain a non-obvious choice.

Failing test: $ARGUMENTS

--- invoke: /fix tests/test_auth.py::test_expired_token
```

WHY IT SAVES

Engineer-style prompts are **2-3× shorter** than the natural-language version. Codified in a file, they get reused exactly — no token drift, no “please could you also” padding. Diff-only output saves another 60–70%.

THE TRAP

Loading 30 slash commands at session start — their descriptions enter context unconditionally. Keep the active set small; nest specialised ones under sub-agent prompts instead of the main thread.

Sources: code.claude.com/docs slash-commands reference · "Claude Code FinOps Hebrew guide" (80 vs 200 token measurement, 60–70% diff savings).

ANTI-PATTERN 1 OF 6 · SENDING WHOLE FILES WHEN YOU NEED 5 LINES

COST: VERY HIGH · EVERYWHERE

AP1 A 200-line file for a 5-line change — 195 lines of waste.

WHAT IT LOOKS LIKE

```
# bad — paste the whole file
"Here's auth.py, please add rate limiting:"
<200 lines>

# worse — let the model echo it back
"Return the updated file."
→ ~3,000 output tokens, mostly unchanged.

# even worse — both, per turn
Repeat for every iteration.
On 10 turns: ~60K tokens, ~$1.20 burned
on lines that didn't change.
```

THE FIX

Send the excerpt. Demand a diff.

1. `claude --file auth.py:42-58` — only the relevant range.
2. "Return only the changed lines as a unified diff."
3. Apply with `git apply` — the model never re-emits unchanged code.

Result: ~5 lines in, ~5 lines out — not 200 in, 200 out. 95% reduction on iterative work.

Source: "Claude Code FinOps Hebrew guide" — 200-line file with 5-line change = 195 wasted output lines per turn.

COST: COMPOUNDS PER TURN

AP2 The chat doesn't remember — it re-pays.

WHAT YOU'RE REALLY PAYING

Every turn re-sends the entire conversation history. A 50-turn chat sends turn 1 fifty times.

```
Turn 1:   2K tokens
Turn 10:  ~20K tokens (history+new)
Turn 30:  ~60K tokens
Turn 50:  ~120K tokens, auto-compact triggers
```

Cache helps on input — but only if topic stays stable. Topic drift = cache miss + 120K fresh tokens.

THE FIX

New task → new chat. The hotkey is `/clear`.

Rule: when the topic changes, the chat ends.

Bridge state: ask for a 5-bullet summary, paste it into the new chat — cheaper than carrying 30 turns.

Persist decisions: append them to CLAUDE.md — they survive every new session at cache prices.

Don't trust auto-compact — it silently rewrites history and breaks the cache (see Anti #5).

Sources: platform.claude.com/docs context-management cheatsheet · "Claude Code FinOps Hebrew guide" anti-pattern #2.

ANTI-PATTERN 3 OF 6 · OPUS FOR EVERYTHING — INCLUDING CRUD

COST: × 5 TO × 15

AP3 Using Opus for boilerplate is paying jet-fuel prices for a taxi ride.

THE ARITHMETIC

Same prompt, two models. Same answer.

```
Task: write a CRUD endpoint (~1K in / 600 out)

Sonnet 4.5:  (1K×$3 + 0.6K×$15) / 1M = $0.012
Opus 4.5:    (1K×$5 + 0.6K×$25) / 1M = $0.020

× 5,000 calls / mo:  $60 vs $100
```

$40/month for output Sonnet would have produced byte-identical. That's the rate-limit budget you didn't have left for the real work.

THE FIX

Default to Sonnet. Opt into Opus.

1. Shell alias: `claude --model sonnet` as default.
2. Use `/model opus` only when about to plan / design / debug-hard.
3. `/model sonnet` back the moment you're back to typing code.
4. Sub-agents inherit your default — route them to Haiku for scans.

Sources: Anthropic price card (Q2 2026, $3/$15 vs $5/$25) · "Claude Code FinOps Hebrew guide" anti-pattern #1.

ANTI-PATTERN 4 OF 6 · “BUILD IT AND EXPLAIN IT”

COST: 2–3× OUTPUT

AP4 The prose between the code is half your bill.

WHAT POLLUTION COSTS

“Here's what I did and why —” is **output** tokens. Output is ×5 input.

```
Default: "Write & explain"
→ ~600 tokens code + ~1,200 prose
→ 1,800 out × $15/M × 5K calls/mo
→ $135/mo of unread prose.

Code-only:
→ 600 out × $15/M × 5K calls/mo
→ $45/mo. Same answer.
```

Field-measured cuts: **“code only” −60% · “tests only, no fixtures” −70% · “bugs only, no style” −40%.**

THE FIX

Build OR explain — never both in one turn.

1. CLAUDE.md rule: *“Return code only. No preamble, no recap.”*
2. Want a rationale? Separate turn, separate question. Cheaper because cached context now includes the code.
3. For diffs: “`unified diff only`” — saves the unchanged 95% of the file.
4. Reviews: “`list bugs only, no style comments`” — −40%.

Source: "Claude Code FinOps Hebrew guide" measurements (60% / 70% / 40% reductions on three task families).

COST: INVISIBLE · RUINS CACHE

AP5 Auto-compact at 80% — quietly the most expensive feature.

WHAT AUTO-COMPACT ACTUALLY DOES

At 80% of context (~160K tokens), Claude Code summarises history and rebuilds the prompt — new prefix, cold cache.

```
Before compact:  160K cached, hot.
Auto-compact fires.
After compact:   ~30K fresh tokens.
Cache rate next turn: 0%.

And: useful context just got summarised —
so the agent re-fetches what it lost.
```

Two costs at once: full-price input on the new prefix, plus repeated work because the summary lost details.

THE FIX

Manual is cheaper than automatic.

1. Watch the context meter. At ~60% — choose: `/clear` or `/compact`.
2. `/clear` is almost always right — persist decisions to CLAUDE.md or a plan file.
3. `/compact <instructions>` when you must preserve mid-task state — tell it what to keep.
4. Some engineers cap context at 200K (vs 1M) to force earlier discipline.

Sources: platform.claude.com/docs Compaction reference · hyperdev.matsuoka.com "Protecting more context" (2026) · albertsikkema.com 200K-cap practice.

ANTI-PATTERN 6 OF 6 · THE --CONTINUE CHAIN

COST: STALE TOKENS, EVERY TURN

AP6 Yesterday's chat is today's cache miss.

WHY IT BITES

`claude --continue` resumes the prior session — full history, no cache (TTL ~5 min).

```
Yesterday's chat: 60K tokens of history,
mostly tangents you've moved past.

--continue → 60K fresh tokens billed,
0% cache hit on the first turn.

And every turn after re-sends those same
stale 60K + new content.
```

On a long-running project, the chain accumulates — you carry context you forgot about, paid for nightly.

THE FIX

Fresh chat. Persistent context lives on disk.

1. Start of day → new chat. `CLAUDE.md` + plan file load via cache anyway.
2. Reach for `--continue` only when the prior task is *actively unfinished* — not as a default.
3. Mid-task save: ask for a 5-bullet handoff. Paste into tomorrow's fresh chat.
4. Track open threads in code (TODOs / issues), not in chat history.

Source: "Claude Code FinOps Hebrew guide" · cache TTL behaviour from platform.claude.com/docs prompt-caching.

Token cost is not a billing problem. It's a context-engineering problem — the same architecture muscles that make agents safe also make them cheap.

THREE THINGS CHANGE WHEN THE ARCHITECTURE LANDS

01

Rate-limit hits stop being a daily event.

Same workload, fresh tokens drop by ×6.7. The 5-hour window becomes hard to fill — even on Max 5x.

02

Quality goes up with cost going down.

Plan-then-execute landed churn at 2.32% — below the pre-AI human baseline of 3.3%. Same cause: thinking happens before code.

03

Plan upgrades become deferrable.

Max 5x ($100) runs the workload Max 20x ($200) struggles with — if the architecture's there. The plan tier follows the architecture, not the other way round.

Field benchmark: its-yosi.com (Max 5x, 8.83B tokens / mo, 1 rate-limit hit / mo, churn 2.32%).

Pick the rung. Honestly.

STAGE 0 · ZERO

"Why's my bill so high?"

Default: Opus everywhere.

Cache rate: Unknown.

CLAUDE.md: None.

Pattern: One long chat per project.

Hits rate limit: Daily.

Effective price: ~×7 the floor.

STAGE 1 · REACTIVE

Sonnet by default

Default: Sonnet, opt-in to Opus.

Cache rate: ~50% (accidental).

CLAUDE.md: A starter file.

Pattern: `/clear` after a few hours.

Hits rate limit: Weekly.

Effective price: ~×3 the floor.

STAGE 2 · PROACTIVE

Cache-aware engineer

Routing: Haiku / Sonnet / Opus per verb.

Cache rate: 85–90% (intentional).

CLAUDE.md: Tight, <150 lines.

Pattern: Plan-then-execute on big tasks.

Hits rate limit: Rare.

Effective price: ~×1.5 the floor.

STAGE 3 · HERO

Agent team architecture

Routing: Custom sub-agent team delegates.

Cache rate: 95%+ (measured monthly).

CLAUDE.md: Plus per-agent system prompts.

Pattern: Plan docs + slash cmds + sub-agents.

Hits rate limit: <1 / month.

Effective price: floor — ×182 leverage on $100.

Composite ladder synthesised from its-yosi.com field benchmark + Anthropic enterprise consumption norms ($150–250 / dev / mo).

PR BY MONDAY

Which of the six levers is missing from your team today?

(a) · Lever #1 · Caching

No CLAUDE.md. No stable prefix. Every turn re-pays.

Highest-leverage fix. One PR, one file, ×6 savings on the cache-friendly slice.

(b) · Lever #2 · Model routing

Opus is the default. Sonnet would write the same code.

The bet of the deck — this is where most teams burn through their 5-hour window.

(c) · Lever #3 · Sub-agents

One context, polluted by every grep and audit.

The lever that turns lone-developer into agent-team. 0.4% return rate, measured.

My bet is (b). Yours doesn't have to match — but one of the three is true.