Flow State, Metered

August 2026

12 min read • 2,795 words

I remember what programming used to feel like in my body.

There was a file open in Sublime Text, a terminal beside it, and one problem occupying the whole available field of consciousness. I would change a line, run the program, read the failure, and change another line. Hours disappeared. Hunger became theoretical. The work was difficult, but the difficulty held me there. Every small discovery made the next question more interesting.

That was flow state in its natural habitat: a demanding problem, immediate feedback, and just enough skill to keep the problem from becoming either boring or impossible. The reward was not merely that software existed at the end. The process itself was rewarding. Attention entered the loop and came back as understanding.

I still make software for hours at a time. I make more of it than I ever have. But increasingly, I am not moving through code one line at a time. I am describing what should happen, dispatching an agent, reviewing what came back, redirecting it, and starting something else while it works. I wrote about shipping the largest PyTheory release in its history from my phone in roller-coaster lines. It did not feel like coding. It felt like conducting.

Conducting can be wonderful. It is also a different cognitive activity from playing the instrument.

The more seriously I use agentic coding tools, the more I notice a strange inversion. In traditional programming, my attention was the scarce resource. I could keep working for as long as I had patience, curiosity, and physical stamina. With AI-assisted programming, execution is abundant and attention is comparatively underloaded. To recreate the old density of engagement, I have to keep more work moving at once. I have to use more inference. I have to blow through the tokens.

And then the meter appears.

The Old Loop

Old-school programming was full of friction, but most of the friction belonged to the thing itself. The compiler rejected your program because your program was wrong. The test failed because your model of the system was incomplete. The segfault was not trying to upsell you. It was a fact about memory.

That kind of resistance is unusually good at holding attention. The system gives immediate, specific feedback, and the feedback points back into the work. You are not waiting for somebody else to do the thinking. You are thinking through the material directly. The hands, the editor, the runtime, and the mind become one continuous circuit.

> This is close to the hands-on imperative I wrote about in The Hacker Ethic and the Vibe Coder. Understanding came from touching the system repeatedly, including all the places where it resisted you.

There were interruptions, of course. Builds took time. Documentation was missing. Dependencies broke. But the governing rhythm remained local. I could decide to spend twenty minutes on a name, two hours on an abstraction, or the entire night on one impossible bug. Nothing outside the problem told me that my allocation of thought had expired.

This mattered more than I understood at the time. Programming was not only a way to produce software. It was a structure for sustained attention. The machine created a narrow channel through which curiosity could run for hours without spilling everywhere. For a certain kind of mind, mine included, this was not merely productive. It was regulating. The work organized consciousness.

I have called programming a spiritual practice because it could produce this quality of presence. You did not have to force yourself to concentrate once the loop took hold. The interestingness of the problem supplied the willpower. Effort and reward were braided together.

The New Scarce Resource

Agentic coding changes the loop because it changes what the human is doing inside it.

When I ask an agent to implement a feature, the machine may read twenty files, form a plan, edit the code, run the tests, notice a failure, repair it, and report back. These are exactly the actions that once sustained my engagement. Now they happen behind a status indicator. My work moves upward: define intent, provide context, judge the result, decide what matters next.

This is real work. Judgment is not a ceremonial step added after the machine does the important part. Judgment is the important part. Taste, scope, architecture, and responsibility remain human problems. My argument in Write It First, Then Let AI Drive was that a strong human foundation gives the model something worth extending. The model supplies throughput; the person supplies direction.

But direction arrives in bursts. I can give an agent a minute of concentrated intent and then spend five minutes waiting for the consequence. If I watch it work, I am under-stimulated. If I leave, I break context. If I open another agent, I recover momentum.

So I open another agent.

Then another.

Soon I am not working on a problem. I am operating a small portfolio of problems. One task is implementing, one is testing, one needs review, one is blocked on a choice, and another has wandered into a part of the codebase it should not be touching. The cognitive challenge is no longer how deeply I can descend into one system. It is how many partially delegated systems I can keep coherent at once.

The old loop rewarded depth. The new loop rewards concurrency.

This is the inversion I keep feeling. To remain as cognitively engaged as I was while writing code by hand, I have to increase throughput until the orchestration itself becomes difficult enough to absorb me. One agent is often less engaging than one editor used to be. Five agents can approach the old intensity, but now the intensity comes from routing, remembering, reviewing, and deciding. I am manufacturing enough traffic to keep the human control plane busy.

> This is not necessarily worse. A conductor can enter flow as completely as a violinist. But the challenge-skill balance has moved. The new skill is maintaining a coherent vision across parallel work, and the new failure mode is confusing motion with meaning.

That traffic costs tokens.

Flow Becomes Throughput

This changes the psychology of a productive day.

In the old loop, I stopped because I was tired, the problem was solved, or life required me somewhere else. In the agentic loop, another stopping condition enters the room: the plan limit. The tool tells me I have consumed a percentage of a rolling window, a weekly allocation, or a shared pool. A resource that was invisible during traditional programming now sits beside the work and measures how much continuation remains.

The meter is not just measuring output. It is measuring access to a particular cognitive rhythm.

That distinction matters. If Claude Code were merely a faster compiler, reaching a limit would be an ordinary inconvenience. But an agent can become part of how a person initiates tasks, sustains attention, explores unfamiliar systems, and crosses the distance between intention and action. For some people, especially people whose executive function is uneven, the model is not simply labor-saving software. It is scaffolding for thought. Interrupting it can feel less like a tool running out of credits and more like part of the working mind being put behind glass.

> I mean this functionally, not mystically. A notebook, a calendar, an IDE, and another person can all become parts of a cognitive system. I made the broader case in Your Phone Is Part of Your Mind.

The natural response is to become efficient about consumption. Use a cheaper model for easy work. Compact context. Keep prompts precise. Do not let agents wander. These are sensible practices. But notice what has happened: the programmer is now optimizing not only the software, but the rate at which cognition passes through a commercial allowance.

The plan begins to set the tempo.

There is another, stranger pressure too. Once I have paid for a large allowance, unused capacity can feel wasted. A subscription quietly turns a ceiling into a target. If I am nowhere near the limit, perhaps I am not getting the value I purchased. If I reach it, perhaps I am exactly the kind of serious user who needs the next tier.

Either interpretation points upward.

The Token Thirst Trap

This is where the pricing starts to feel like a thirst trap.

I want to be precise about the accusation, because there are several honest reasons for usage limits. Inference costs money. Capacity is finite. Heavy users can consume radically more compute than ordinary users. A flat subscription with no boundary would either be unprofitable, unreliable, or subsidized by people who barely use it. Rate limits can protect the service and make pricing legible.

But a limit can be economically necessary and psychologically powerful at the same time.

Claude's individual plans make the structure unusually visible. Pro is currently $20 per month. Max offers five times or twenty times the Pro capacity for $100 or $200 per month. Usage runs across rolling five-hour sessions, with weekly

limits layered on top, and activity across Claude and Claude Code draws from the same pool. When the pool runs out, the documented options include waiting, enabling metered usage credits, or moving to a higher plan.

> These details come from Anthropic's current pricing page and Max plan documentation, consulted in August 2026. The numbers will change. The structure is the important part.

Anthropic does not hide the emotional benefit it is selling. Its Max documentation describes higher limits as "No more interruptions" and invites users to "stay in flow when it matters most."

So yes, I think they know.

That sentence does not mean I know a secret product strategy. I do not know whether anyone at Anthropic sat in a room and said, "Let us interrupt programmers at the moment of maximum engagement so they will upgrade." I would not claim that without evidence. The mundane explanation is probably a mixture of capacity planning, cost control, market segmentation, and a desire to offer predictable subscriptions.

But the company plainly knows that interruption hurts, that flow has value, and that higher tiers can sell relief from that hurt. The upgrade is not presented only as more computation. It is presented as continuity of mind.

That is the thirst trap: the product increases the rate at which you can turn intention into reality, which increases your appetite for continuation, then meters the continuation. It teaches you to work at a cognitive velocity that the lower plan cannot sustain. The more completely you adapt your process to the tool, the more painful the boundary becomes.

Social platforms learned to sell access to our attention. Agentic tools may have found something even more intimate to sell back to us: uninterrupted access to our own momentum.

Intent Is the Least Interesting Question

I keep returning to the principle that the metrics you expose are the values you endorse. A meter changes behavior whether or not its designer intends every consequence.

Display steps, and people walk in circles before midnight. Display a streak, and rest becomes failure. Display tokens, session percentages, and reset times beside creative work, and people begin to experience thought as a reservoir with a refill schedule.

This is why I am less interested in proving manipulation than in describing the incentive gradient. Anthropic benefits when a deeply engaged Pro user becomes a Max user. It benefits again when a Max 5x user becomes a Max 20x user. The user benefits too, at least when the additional capacity produces work worth more than the price. There is no requirement that one side be deceived for the system to exert psychological pressure.

The most effective commercial mechanisms often align genuine value with escalating appetite. Coffee works. Faster computers work. Better musical instruments work. Once a tool expands what you can do, returning to the previous constraint feels like losing part of yourself. The value is real, which is precisely why the lever moves.

Agentic coding adds a peculiar wrinkle: the tool can consume its allowance faster when you become more skilled at using it. As your prompts improve, your projects multiply. As your review practice sharpens, you can keep more agents in flight. Mastery does not necessarily conserve the resource. It can increase the rate at which meaningful uses become visible.

The better I get at collaborating with the machine, the thirstier I become.

The Risk of Plan-Paced Cognition

None of this means we should retreat to typing every line by hand. I do not want the old friction back merely because it was familiar. AI has helped me cross creative blocks that lasted years. It has made difficult work possible on days when my own cognitive machinery was unreliable. It has let me turn ideas into working systems with a speed that still feels miraculous.

I do not want less capability. I want to notice what the capability is training me to optimize for.

The obvious danger is confusing token consumption with engagement, and engagement with meaning. If I need five concurrent agents to feel occupied, I can always invent more work. Every codebase contains infinite improvements. Every project can grow another feature, another abstraction, another test, another platform. Throughput has no natural stopping point.

The old craft contained its own temptation toward obsession, but at least the body eventually slowed the hands. Agentic work can borrow stamina from the machine and keep offering completed tasks faster than judgment can metabolize them. The bottleneck moves from implementation to discernment. That is a dangerous place to increase speed, because discernment is the part we most need to remain slow.

There is also a loss hiding inside the gain. If all difficult implementation is delegated, we may lose one of the structures that taught us how to attend. The cognitive architecture we lost was already being eroded by notifications and algorithmic feeds. Programming remained one of the places where sustained attention could still be practiced. We should be careful about converting that refuge into another dashboard of parallel activity, percentages, and reset clocks.

The goal is not to preserve difficulty for moral reasons. Suffering is not a credential. The goal is to preserve contact with the work.

Keeping the Rhythm Human

I am still learning what a healthy agentic practice looks like. A few principles are becoming clear.

First, the unit of a good day should remain meaningful work, not consumed capacity. A token allowance is a ceiling, not a challenge. Leaving some of it unused is not waste if the thing is already done.

Second, not every task needs to become a delegation. There are parts of a system I want to touch directly because the touching teaches me what the system is. I still want to write the foundation, name the important things, hear the architecture click into place, and understand where the sharp edges live. Hands-on work is not inefficient when understanding is one of the outputs.

Third, concurrency should serve continuity. Starting another agent because the first one is waiting can preserve flow. Starting six because silence feels intolerable creates a different kind of fragmentation. The right number is the number I can still review with care.

Finally, the meter should not be allowed to narrate my appetite back to me. Reaching a limit does not prove seriousness. Staying below it does not prove timidity. An upgrade can be a rational purchase without becoming a referendum on whether I am making enough.

The companies building these tools have choices too. They can make limits predictable, explain the economics clearly, and avoid turning usage displays into achievement systems. They can offer soft continuations at transparent prices instead of theatrical cliffs. They can let people set budgets and pacing preferences that protect them from both surprise bills and compulsive overuse. Most of all, they can remember that they are not merely allocating compute. They are shaping the rhythm in which people think.

This is the recursive loop again. We build tools to increase human agency. The tools reorganize the way agency feels. Then we adapt ourselves to the business model wrapped around the tools.

Old-school programming asked how long I could stay with a problem.

Agentic programming increasingly asks how much problem I can keep moving before the window resets.

Those are not the same question. The first is about attention. The second is about throughput. Both can produce extraordinary software, and both can produce genuine flow, but only one arrives with a price ladder attached to the continuation of the feeling.

The scarce resource was once my willingness to remain present. Now the tool can provide more momentum than one mind knows how to inhabit, then charge according to how continuously I want to inhabit it.

That is powerful. It is useful. It may even be fairly priced.

But I want the rhythm of my thought to remain mine.