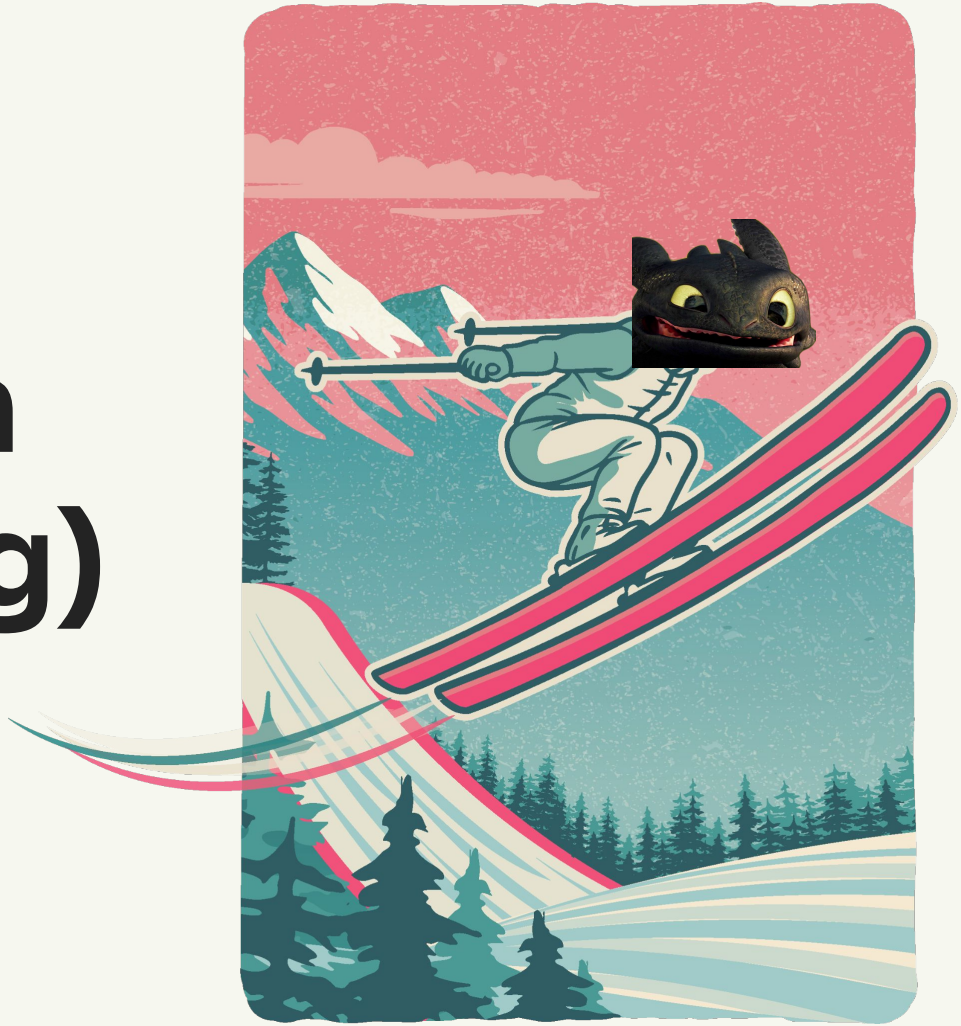
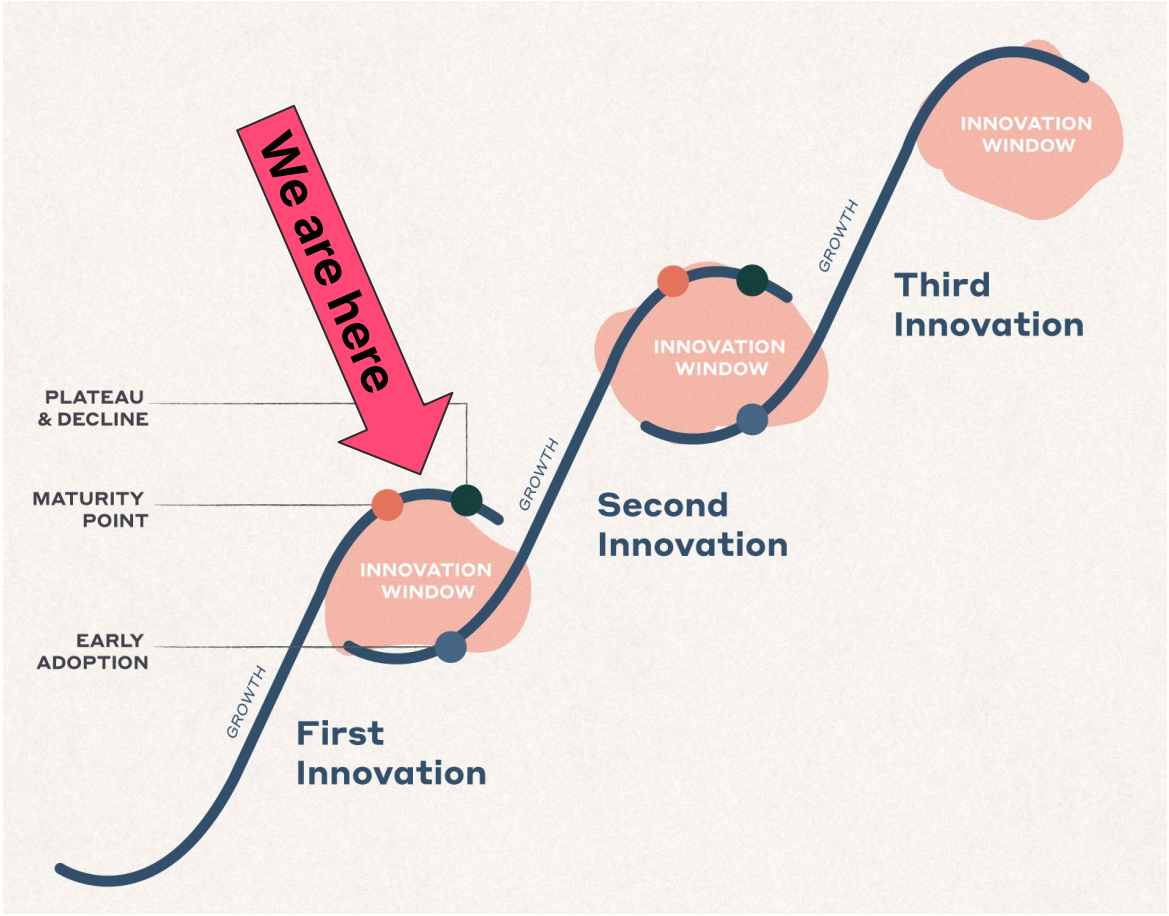
CKO 26
MAKE THE LEAP

How to Train Your (Coding) Agent



Using a coding agents be like



Raw model intelligence growth has slowed

Diminishing returns from more data & compute

Next breakthrough unknowable, so focus on better use of existing models

1

Model Quality

"Intelligence":
limits quality of output

2

Context Window

"Working memory":
limits size of task

3

Generation Rate

"Tokens per second":
limits coding speed

These limits will **always constrain**
New models **only loosen, not remove them**

CKO26
MAKE THE LEAP

Limit #1: Model Quality

Limit #1 — Model Quality

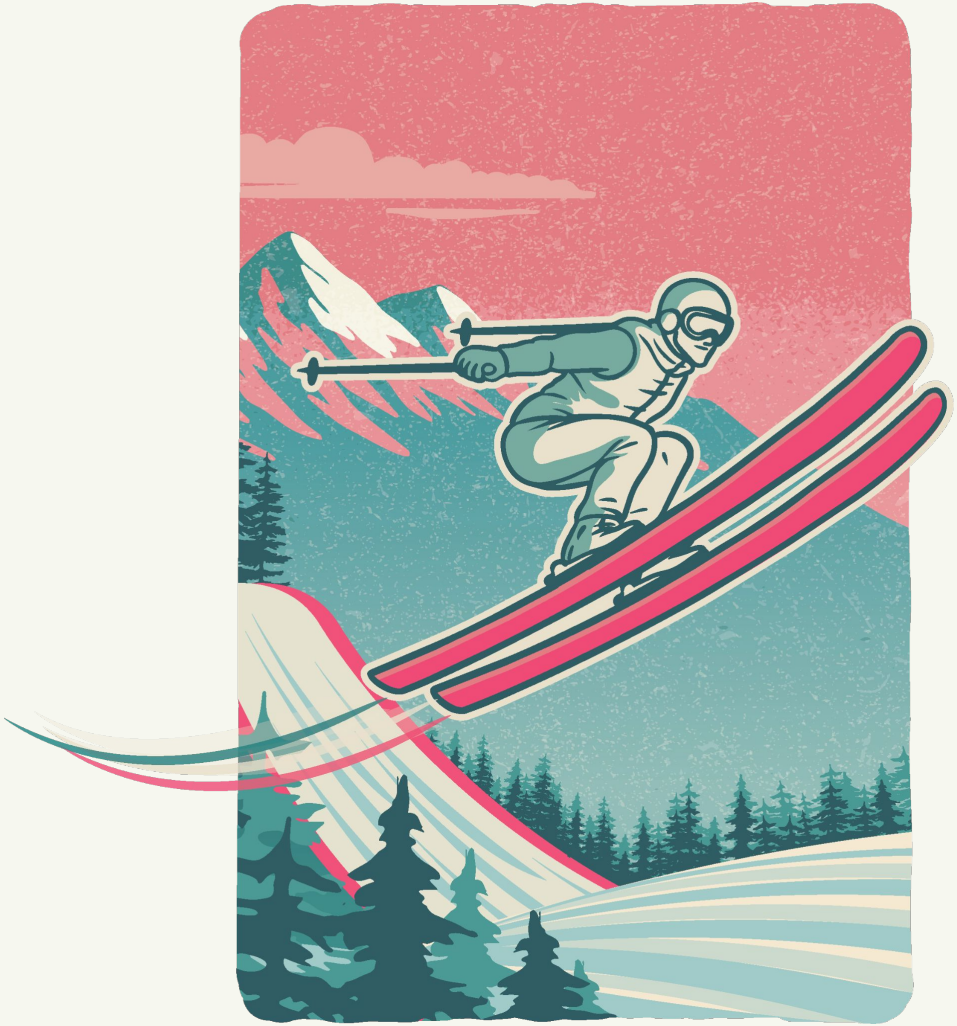
Model Quality = **"depth" of pattern** the model can handle, **"intelligence"**

How to manage: choose a "smarter" model to improve results (a bit)

Future: we may be at a qualitative "plateau"; don't hold your breath

CKO26
MAKE THE LEAP

Limit #2: Context Window

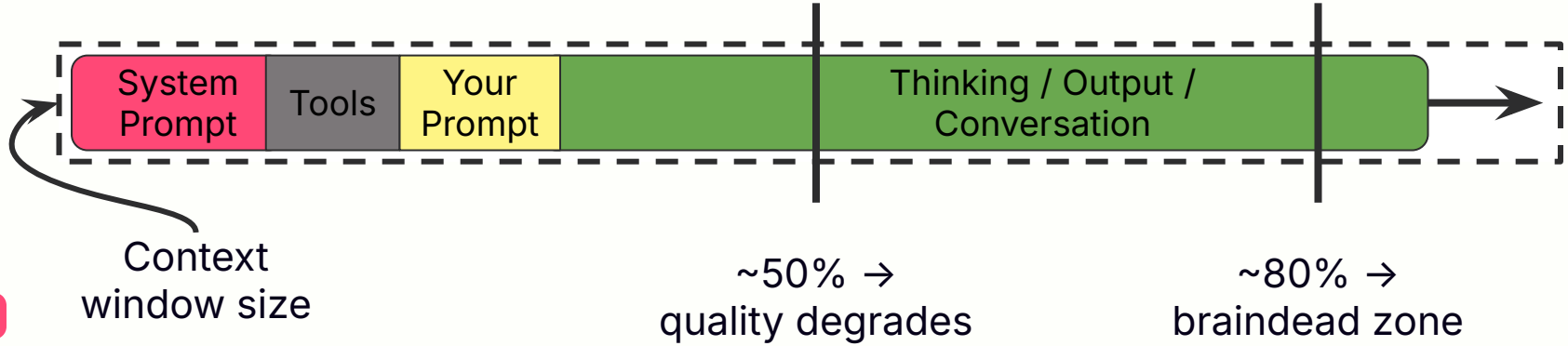


Limit #2 — Context Window

Context Window = **"Working Memory":** all input/output uses this up

Running low = **LLM amnesia**

Most common reason for a failed LLM task



Limit #2 — Context Window

How to manage:

- ➢ **Tools/MCP:** offload work outside the LLM
- ➢ **"Skills":** delay loading context until needed
- ➢ **Subagents:** separate context window to offload from main
- ➢ **"Ralph Wiggum" loop:** periodic context wipes with persistent progress

How NOT to manage: "compaction" — just dilutes/delays the amnesia

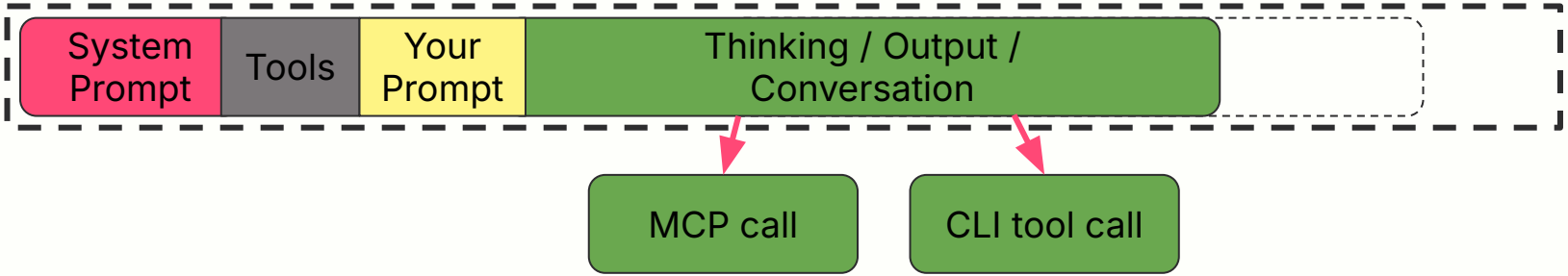
Future: new model releases increase this, but always limited

Tools/MCP

Idea: LLM invokes tools to do work, **minimizing input/output tokens**

Examples: (see appendix for how-to)

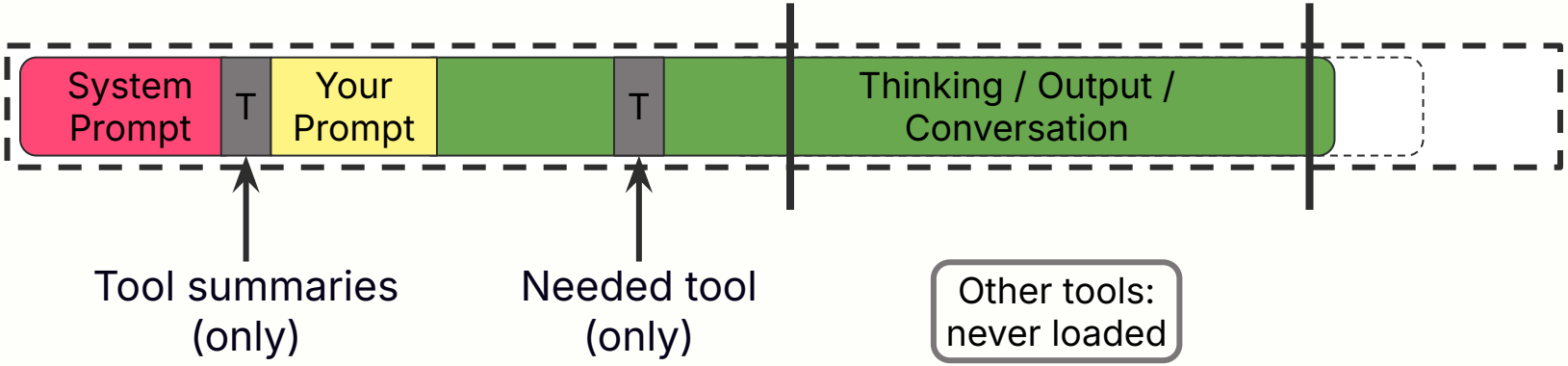
- ➢ **gopls MCP:** code exploration (e.g. find usages, implementations, etc.)
- ➢ **gorename CLI:** rename symbols in one command (spanning many files)



"Skills"

Idea: load **summary only** of tool capabilities, full instructions only if needed

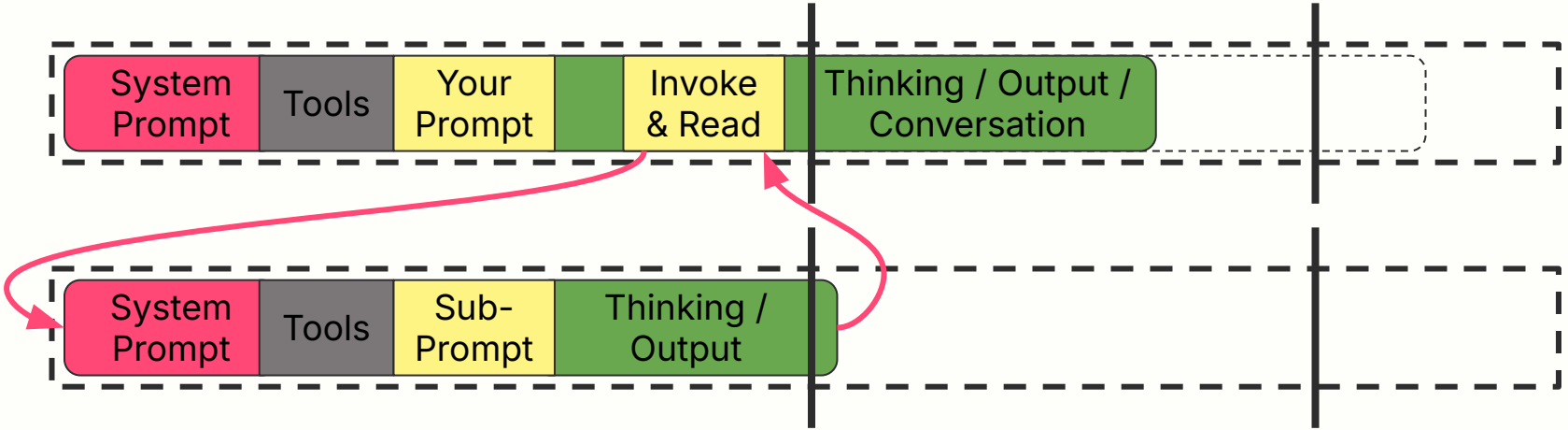
Example: Claude "Skills" load "frontmatter" only, then main .MD file if needed, then auxiliary .MD files based on that



Subagents

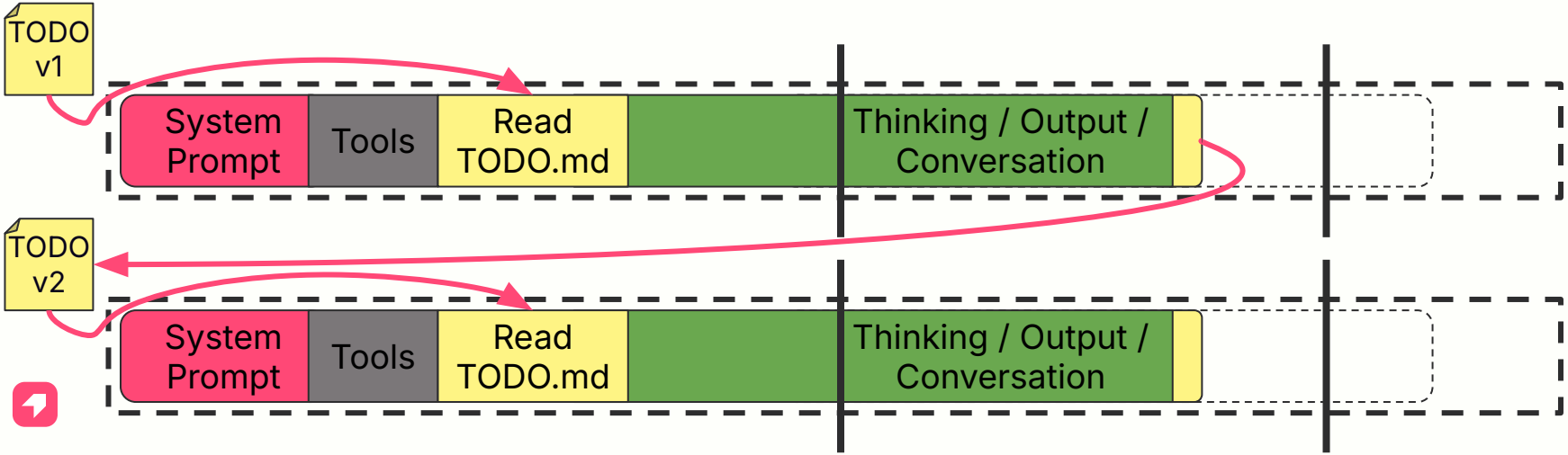
Idea: main agent **prompts 2nd agent** with a task, only processes result (Costs **more total tokens** due to duplicative prompts, etc.)

Example: "run a subagent to run tests and analyze results"



"Ralph Wiggum" loop

Idea: break **large tasks** into **context-sized chunks**, each **with fresh context**
(Costs **more total tokens** due to duplicative prompts, etc.)



"Ralph Wiggum" loop

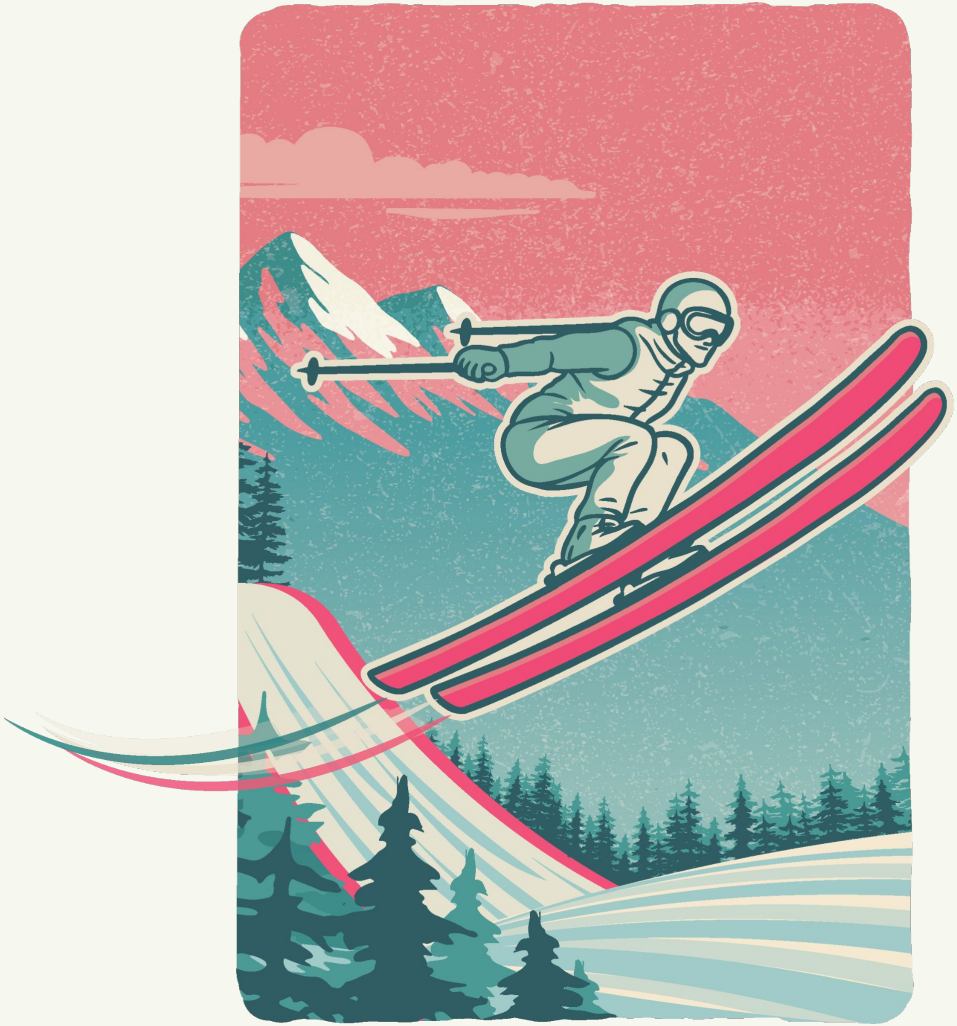
Example:

1. Write specs to **GOAL.md** (or ask agent to do it)
2. Prompt: "Read **GOAL.md**, break down into tasks, write to **TASKS.md**"
3. /clear
4. Prompt: "Read **GOAL.md** and **TASKS.md**, do 1 task, update **TASKS.md**"
5. Repeat from #3 until done

Advanced: can be **automated** (detect "DONE" in LLM output in step #5)

CKO26
MAKE THE LEAP

Limit #3: Generation Rate



Limit #3 — Generation Rate

Generation Rate = **"tokens per second":** how fast the LLM can think/code

How to manage:

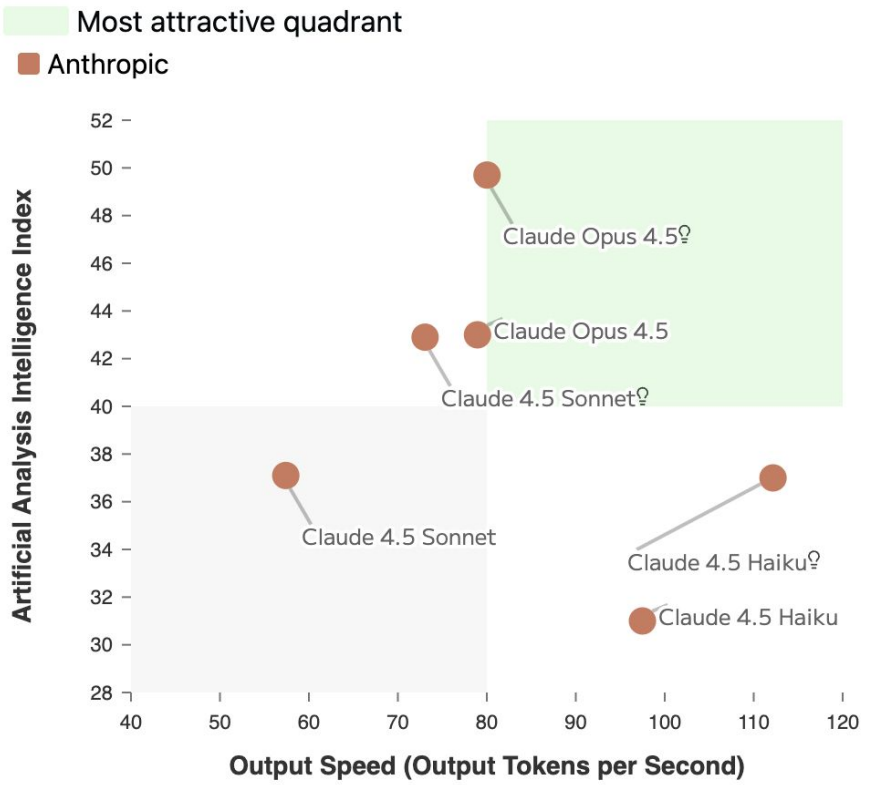
- ➢ Use a faster (though often "dumber") model
- ➢ Parallel agents

Future: faster hardware and algorithms will boost baseline speed

Faster Model

Idea: "dumber" models may generate faster

Example: use Haiku over Sonnet/Opus in Claude Code



Parallel Agents

Idea: run multiple subagents in parallel

Example: "run the following tasks in parallel subagents: X, Y, Z"

WARNINGS:

- **Expensive:** subagent cost overhead, now multiplied with Parallelism™
- **Best with: read-only tasks** (e.g. research, create TODO list)
- **Best with:** cheaper/faster models
- **Terrible with: intersecting tasks** (subagents conflict and "thrash")

CKO26
MAKE THE LEAP

Story Time: Once Upon a Task

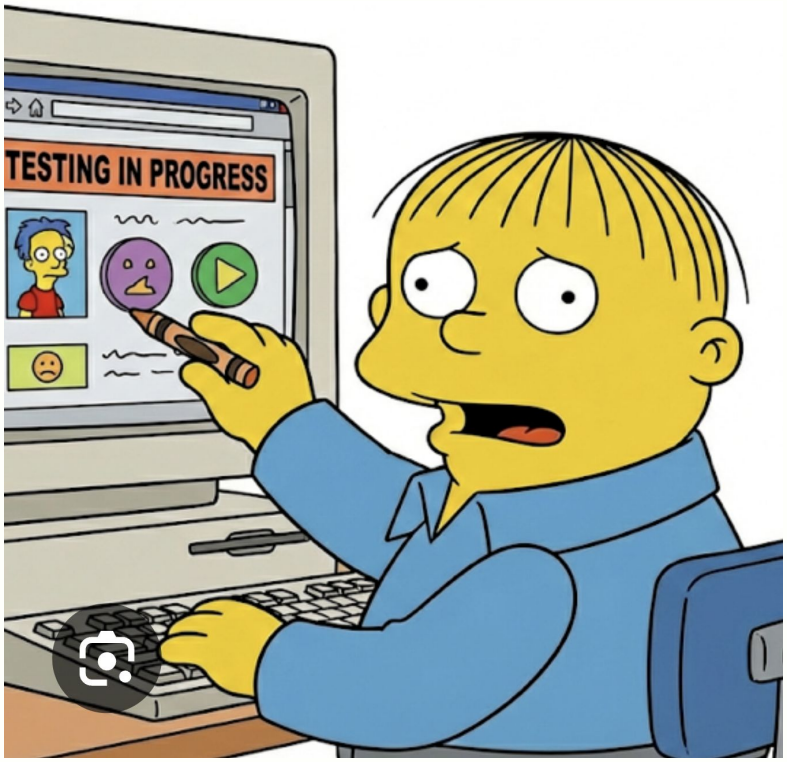
Part 1: Planning

Sam gets assigned a task in Jira and starts working on it.
He pulls the details into his favorite coding agent (Claude Code) using Jira MCP
He tells the agent to create **GOAL.md** and **TASKS.md**
They go back and forth until they're on the same page. Then he tells the agent to start on the first task and tick the checkbox in **TASKS.md** when it's done.

```
# Tasks

- [ ] Initialize project
- [ ] Render static todo list
- [ ] Add in-memory CRUD (add / toggle / delete)
```

Part 2: Creating a Skill



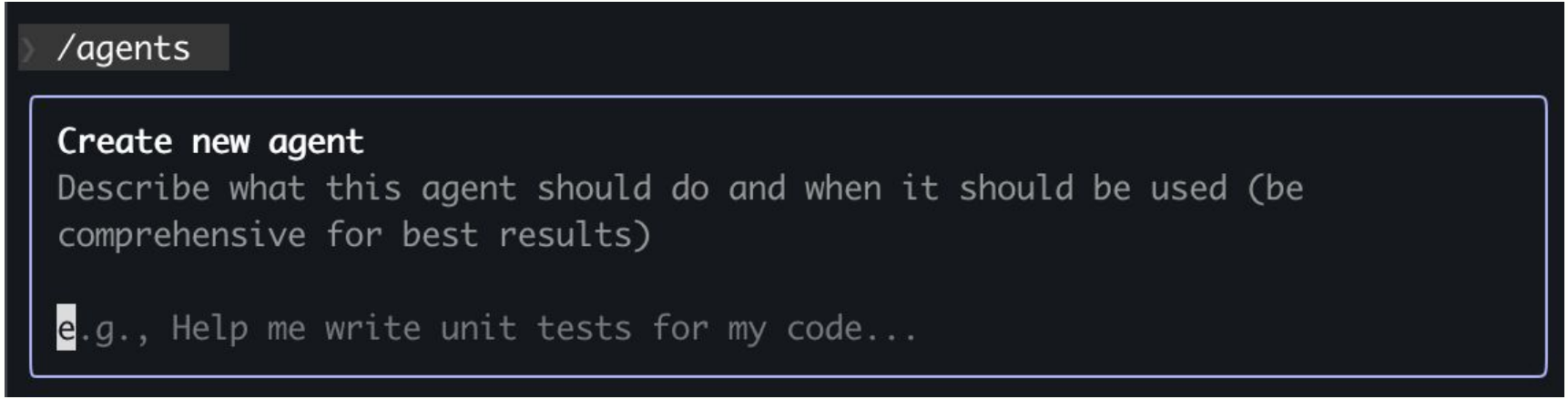
The agent finishes the first task. Sam wants to make sure it actually works, so he gives the agent browser access and asks it to verify the app end-to-end.

The agent struggles with the browser workflow at first; Sam jumps in a few times to guide it.

Once the agent can reliably run the checks, Sam gets the agent to save the process as a reusable skill: **browser-testing**, so he doesn't have to repeat the same instructions next time.

Part 3: Creating a Subagent

Sam doesn't want to tell the agent to run browser-testing **after every task**. He also doesn't want testing details **filling up the main context**. So he sets up a **tester subagent**. It runs the browser-testing skill, tests the app thoroughly and reports any problems back to the main agent.



Part 4: Creating PROMPT.md

Sam notices the chat context is over 50% full and the output is getting worse. He starts a fresh chat and tells the agent to read GOAL.md and TASKS.md, figure out where it left off and tick the checkbox in TASKS.md when it's done. But doing this manually after every task is inefficient. So he **creates a PROMPT.md:**

```
# Prompt

1. Study `GOAL.md` thoroughly
2. Study `TASKS.md` thoroughly
3. Pick the next unchecked task
4. Complete that task fully
5. Run the tester agent to verify the work
6. Fix anything that's broken until the task passes
7. Update `TASKS.md` (`- [x]` the completed task)
```

Part 5: Creating a Bash Loop

Sam wraps PROMPT.md in a bash loop so each time it runs, it starts a brand new chat with clean context. He writes a simple script checks if there are any boxes left to tick:

```
#!/bin/bash
while grep -q '\- \[ \]' TASKS.md; do
    cat PROMPT.md | claude -p
done
```

A Map of Sam's Workflow