TECHCAMP HAMBURG · 25 AUG 2026

What Happens When You Hit Enter

The Anatomy of a Coding Agent

JAN-HENDRIK KOCH · HUB.KI

YOU TYPE A PROMPT. YOU HIT ENTER.

Then the agent hands you a pull request.

What happened in between?

An LLM cannot do anything.

No filesystem · No shell · No memory between calls

How does tool calling work?

1 · HARNESS → MODEL

Tool definition

```
{
  "name": "read_file",
  "description": "Read a file",
  "parameters": {
    "type": "object",
    "properties": {
      "path": { "type": "string" }
    },
    "required": ["path"]
  }
}
```

2 · MODEL → HARNESS

Tool call

```
{
  "name": "read_file",
  "arguments": {
    "path": "src/auth.ts"
  }
}
```

The model only asks. Your code does the work.

Then you put it in a loop.

```
messages = [systemPrompt, userPrompt]

while true:
  response = llm(messages, tools)
  messages.append(response)
  if response has no tool calls: break
  for each call in response.toolCalls:
    messages.append(execute(call))
```

That's a coding agent.

Agent = LLM + tools + loop

Pi is a minimal coding agent.

At a high level, the default tools expose four operations:

`read(path)`	Read file contents into the context.	`write(path, content)`	Create or overwrite a file.
`edit(path, oldText, newText)`	Replace one specific block of text.	`bash(command)`	Run a shell command and return its output.

That's the core: file access + shell execution. Orchestration is optional.

`@earendil-works/pi-coding-agent` · pi.dev

Pi's base prompt is still short.

CURRENT BASE · ABRIDGED

```
You are an expert coding assistant operating inside pi,
a coding agent harness. You help users by reading files,
executing commands, editing code, and writing new files.

Available tools:
- read
- bash
- edit
- write

Guidelines:
- Use bash for file operations like ls, rg, find
- Be concise in your responses
- Show file paths clearly when working with files
```

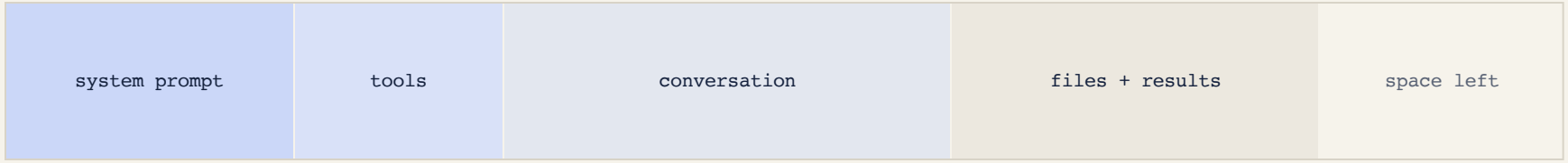
Assembled at runtime: tool descriptions · documentation · project instructions · skills · working directory

The loop is simple. What the model sees is the hard part.

The context window is everything the model can see right now.

For each model call, the harness assembles one finite token budget.

ONE MODEL CALL · FIXED TOKEN BUDGET



Turn 1 → rebuild + resend → Turn 2 → rebuild + resend → Turn 3

Context is a budget: every instruction, file and result competes for the same space.

Skills load expertise only when the task needs it.

A skill packages reusable instructions, references, and scripts for a particular kind of work.

EVERY TASK

`AGENTS.md`

Project conventions · commands · boundaries

\+

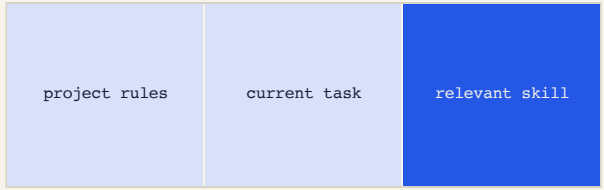
MATCHING TASK

Relevant skill

Instructions · references · scripts

→

AGENT CONTEXT



`AGENTS.md` is persistent context. **Skills are context on demand.**

Good tools protect the context window.

LIMIT WHAT COMES BACK

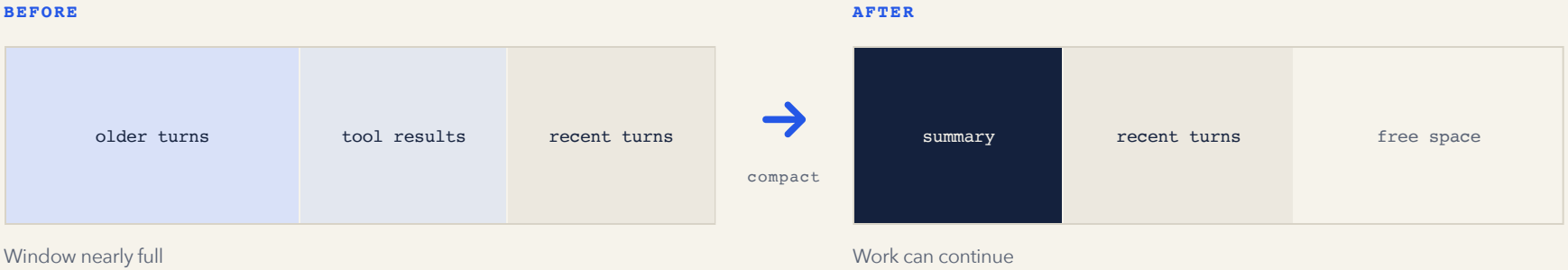
`read`	**First 2,000 lines or 50 KB** Page through the rest with `offset` and `limit`.
`bash`	**Keep the useful tail; truncate the rest** Pi saves the full output separately when needed.

One careless `cat` can flood the window.

Tool limits are a harness design decision.

Compaction makes room by summarizing the past.

When the context window fills, the harness replaces older turns with a shorter summary.

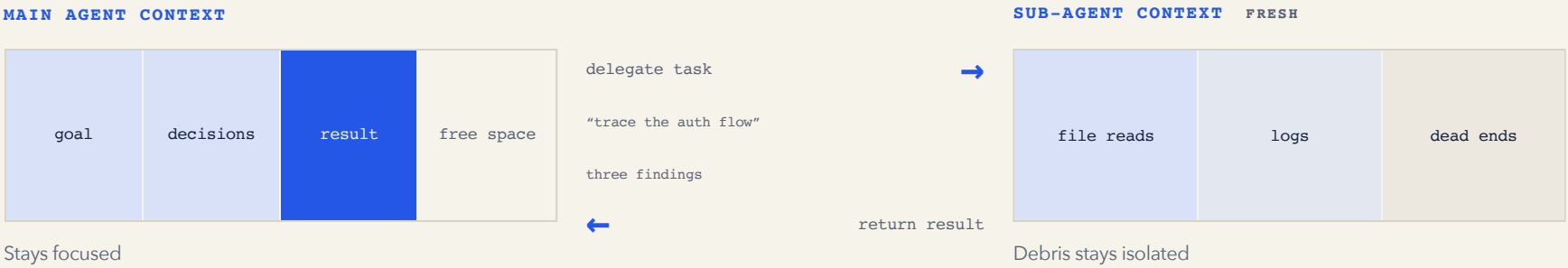


The trade-off: more room, less detail. The summary decides what survives.

`Preserve decisions · constraints · failed approaches`

A sub-agent gives noisy work its own context.

The main agent chooses a bounded task and a small amount of context to send.



Context engineering: only the useful result returns to the main agent.

A sub-agent isn’t smarter. It’s just somewhere else.

What else is in there

Memory
What survives a session, and what doesn't

To-dos & task state
Externalise progress instead of relying on conversation history

Ordering & prompt caching
Where things sit is a performance decision

MCP & tool discovery
When schemas enter context and how to avoid loading all of them upfront

Retrieval
Pull in only what's relevant, with all the usual retrieval problems attached

Each of these is its own talk. Ask me in the break.

What we have and what's still missing.

01

Build the core

LLM + tools + loop

02

Make it work well

Context engineering

03

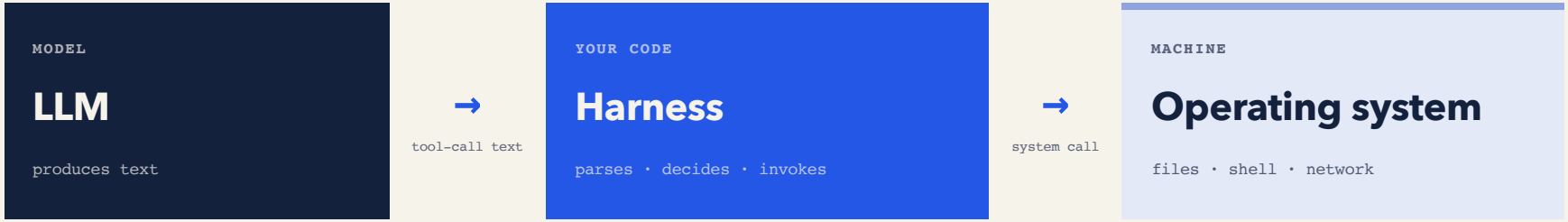
Run it safely

Sandboxing + guardrails

How do we give an agent autonomy without giving it everything?

The LLM only asks.

The model produces a tool call. The harness chooses what happens next.

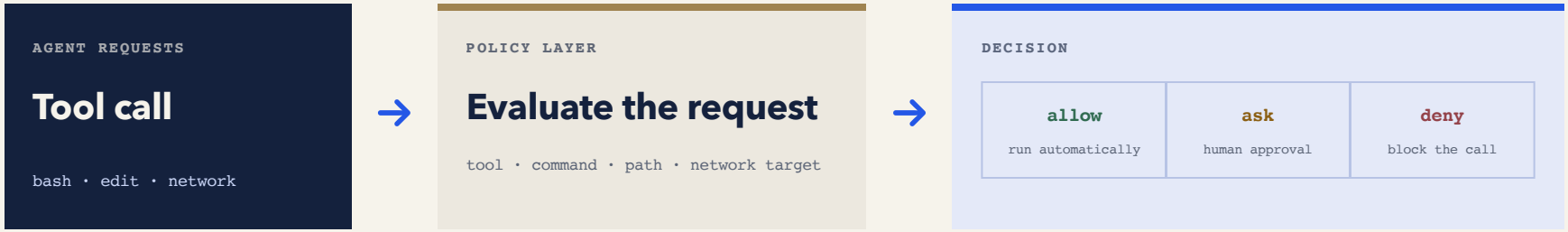


Pi by default: no built-in permission system or sandbox; tools run with the permissions of its process.

Control lives in the harness.

The harness can allow, ask, or deny.

Before a tool runs, policy decides whether the request may proceed.



Approvals bring a human into the harness decision.

Common setups: auto mode · allowlists & deny rules · scoped permissions · LLM approval · human approval

Even approved code needs a boundary.

Approval answers “should this run?” Isolation answers “what can it reach?”

APPROVED TOOL CALL

Execute

bash · edit · network

runs inside →

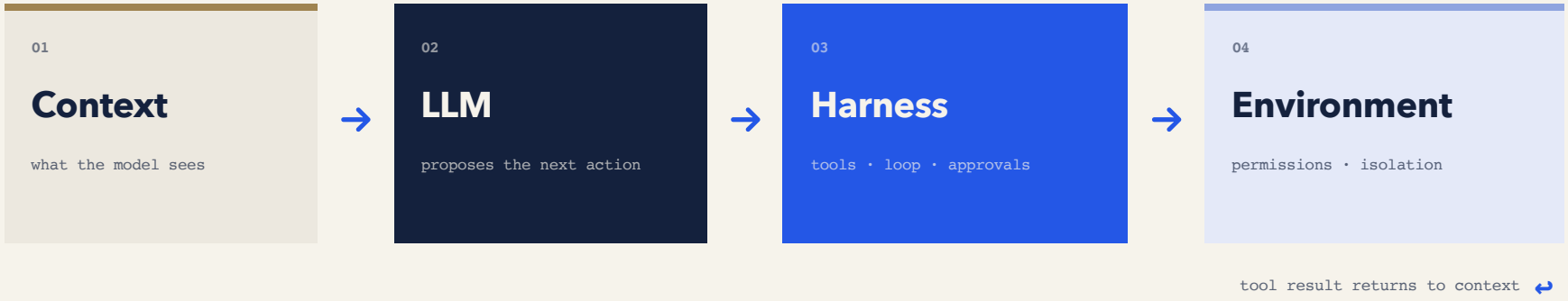
EXECUTION BOUNDARY

Container · sandbox · micro-VM

filesystem	network
credentials	processes

Approvals reduce bad decisions. Isolation limits the blast radius.

Most of the engineering happens around the model.



General-purpose agents look much the same: connect the harness to browsers, email, APIs or databases.

A QUICK SHOW OF HANDS

Are you using AI for coding?

You should be!

Engineering has always moved.

ALWAYS NEW

languages

ALWAYS NEW

frameworks

ALWAYS NEW

patterns

ALWAYS NEW

tools

AND NOW

AI is the newest tool. A very powerful one.

Use it – or fall behind.

Is this still fun?

For me, yes — just somewhere else.

The fun didn't disappear. It shifted.

"AI robbed the fun of coding."

BEFORE

Writing every line

→

NOW

Designing the setup

I'm always asking: **How can I automate more of my own work?**

CONNECTORS · SKILLS · PIPELINES · WORKFLOWS

How many ideas never made it past “someday”?

There’s so much to build.

- + a website
- + an app
- + a game
- + a better dev setup
- + custom CLI tools

It’s an amazing time to be alive.

It’s an amazing time to be an engineer.

THANK YOU

Now go build one.

Questions?

Jan-Hendrik Koch

FOUNDING ENGINEER HUB.KI

CONNECT ON LINKEDIN

READ A SMALL AGENT: pi.dev →