Beyond the Hype: Claude Code in a Real Insurance Project



Frank Baier


helvetia
KARAKUN
Kanton Basel-Stadt
ERNI
INNOFUSE
+++Ba
sel++
++Te+
+++ch
Basel
basel.ch
JAVA
USER
GROUP
CH
BLKB
optravis
company of .msg
cyon
CS2
CSS
IRIX
swiss made
software
ICT
Scouts
Wir fördern Zukunft
SD:>_
[SwissDevJobs.ch]
#BaselOne26
baselone.ch

helvetia

Four things.

01 What is Claude Code?

02 How we applied Claude Code.

03 Observations … and learnings.

04 How we now started to use it: Aurora.

helvetia

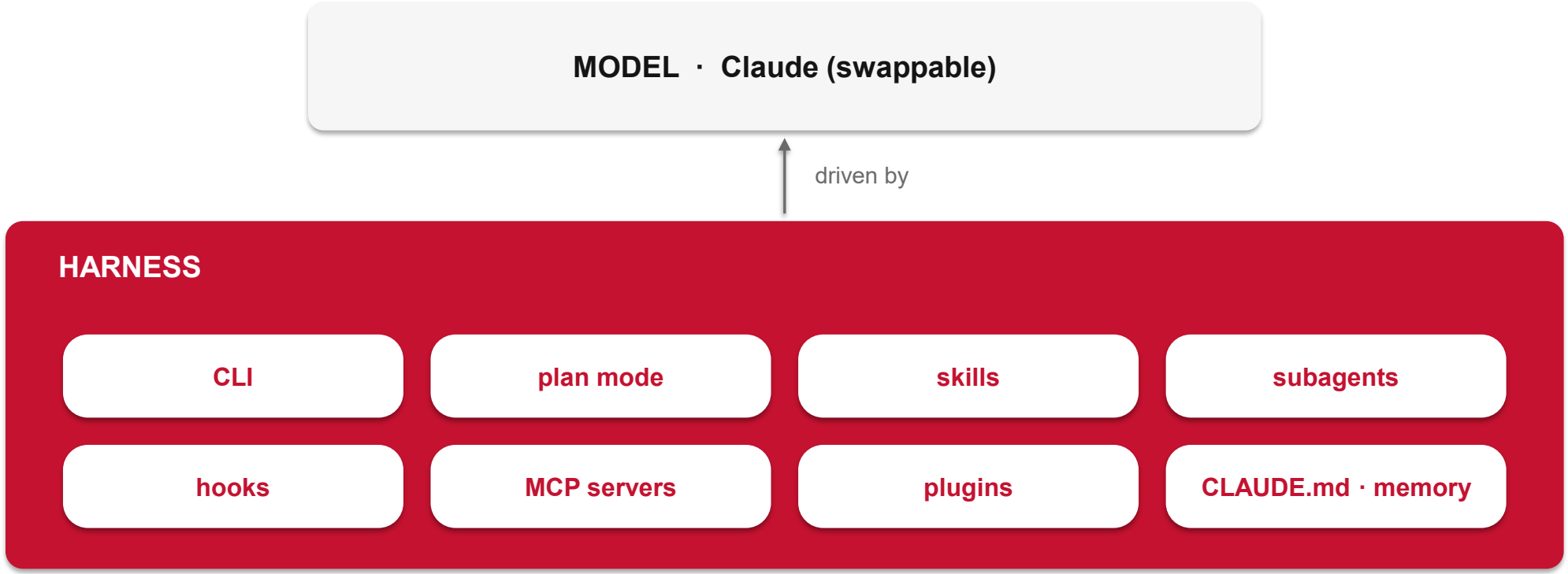
01

What is Claude Code?

helvetia

The cockpit.

The model is the engine. The harness is what you actually steer with.



The model is interchangeable. The harness is what you optimise.

helvetia

Same task. Same way. Every time.

WHAT IS A SKILL?

A Markdown file with instructions, examples and checklists.

Loaded on demand when the task matches the skill description.

Lives in the repo. Versioned. Travels with the project.

Effect: same task done the same way by everyone, every time.

helvetia

One protocol. Many tools.

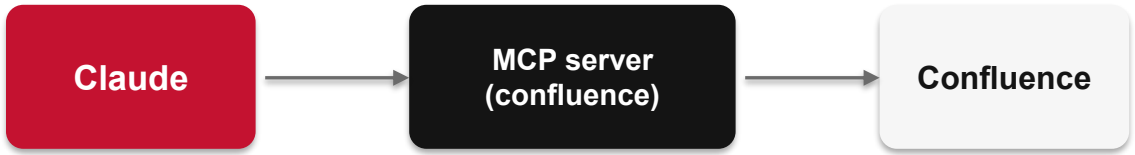
MCP = Model Context Protocol. One standard plug for external systems.

Without MCP



Every team builds their own glue. Nothing reusable.

With MCP



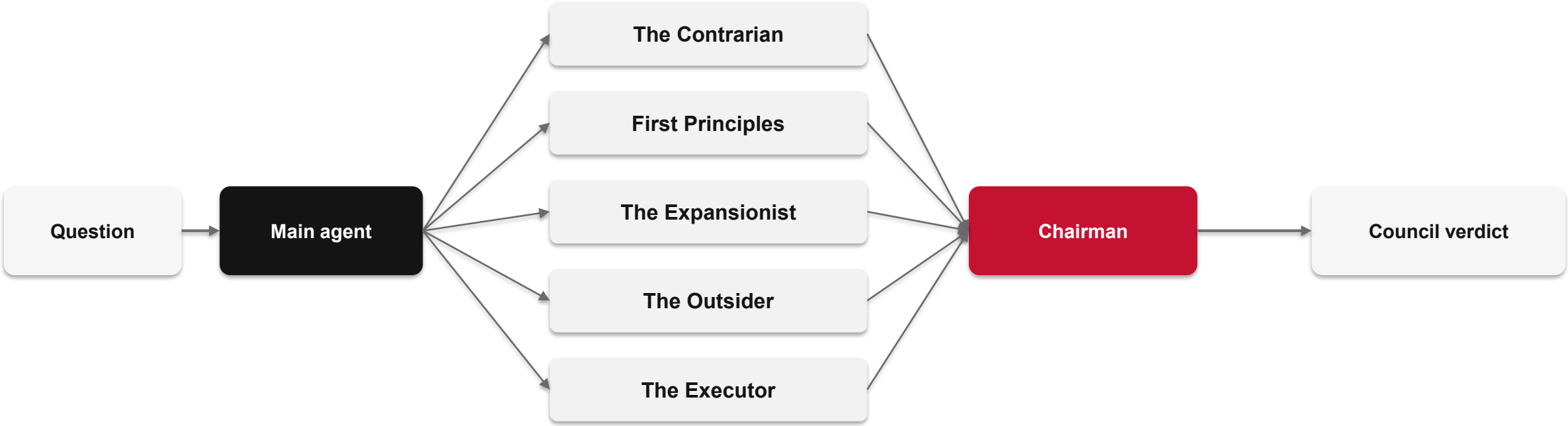
One protocol. Reused by Jira, GitHub, Bitbucket, databases, …

Write the connector once — any agent (Claude Code, Cursor, ChatGPT) can use it.

helvetia

Specialists. On demand.

One question. Five advisors in parallel. A chairman synthesises the verdict.

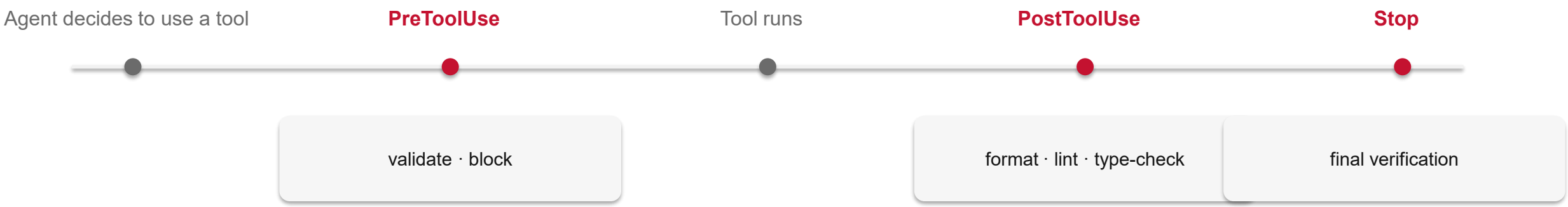


parallel · anonymous peer-review

helvetia

Guardrails. Automatic.

Shell commands the harness runs at lifecycle events. The agent only sees the result.



Less prompt babysitting — fewer 'you forgot to format' moments.

helvetia

02

How we applied Claude Code

helvetia

Salomon. Live, then Claude.

Introducing Guidewire ClaimCenter for Helvetia nonlife claims — three years to go-live.

3 years
building the platform

Nov 2025
ClaimCenter go-live

Feb 2026
Claude Code for the first developers

Today
stabilisation of a running system

GUIDEWIRE CLOUD

ClaimCenter · ContactManager
claims handling and contacts — run by Guidewire

Proprietary platform · Gosu
Guidewire's own JVM language

HELVETIA LANDSCAPE · OPENSHIFT

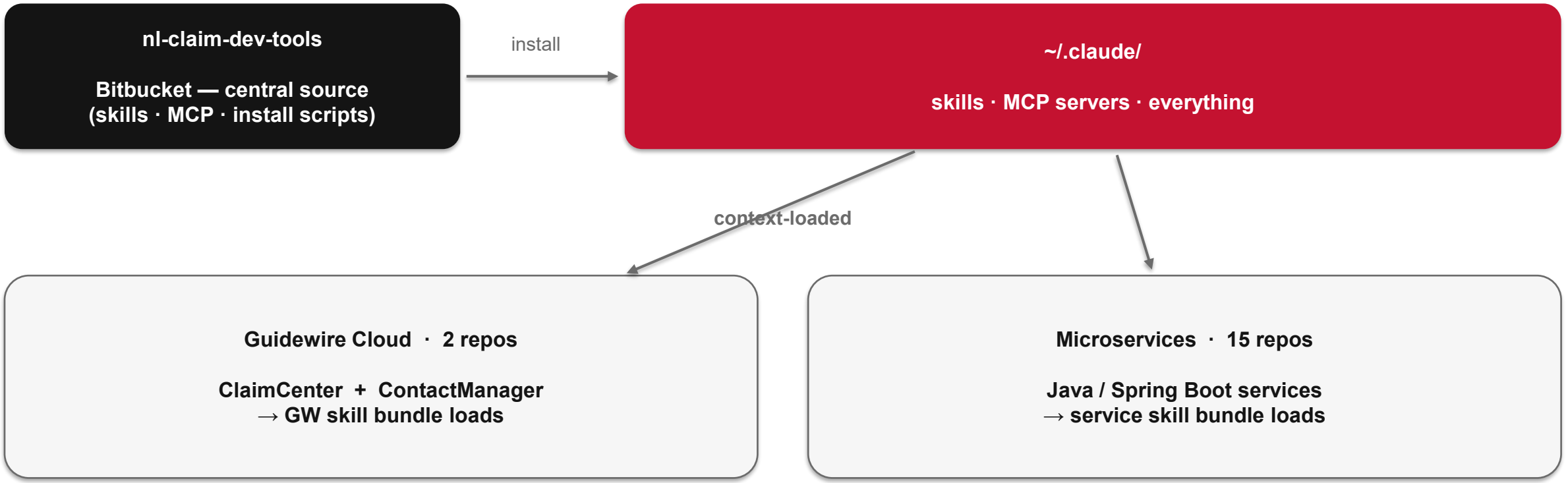
15 microservices · nl-claim-*
Java · Spring Boot

REST · Kafka
wired into the Helvetia landscape

Claude Code arrived after go-live — in a running system, not on a greenfield.

One source. Two contexts.

All skills live in ~/.claude/. The right bundle loads by repo context.



One central source. One install. Context picks the right bundle for the repo you opened.

helvetia

/init, then ADRs.

One Markdown file the agent reads every turn. /init scaffolds it; we extend it.

CLAUDE.md

```
# nl-claim — ClaimCenter project

## Modules
…

## Architecture decisions

The reasoning behind each module
lives in ADRs.

See `docs/architecture/adrs/` —
every enhancement must respect or
update the relevant ADR.
```

1. **/init** scanned the repo and produced the baseline CLAUDE.md.

2. **We added** an Architecture decisions section pointing to docs/architecture/adrs/.

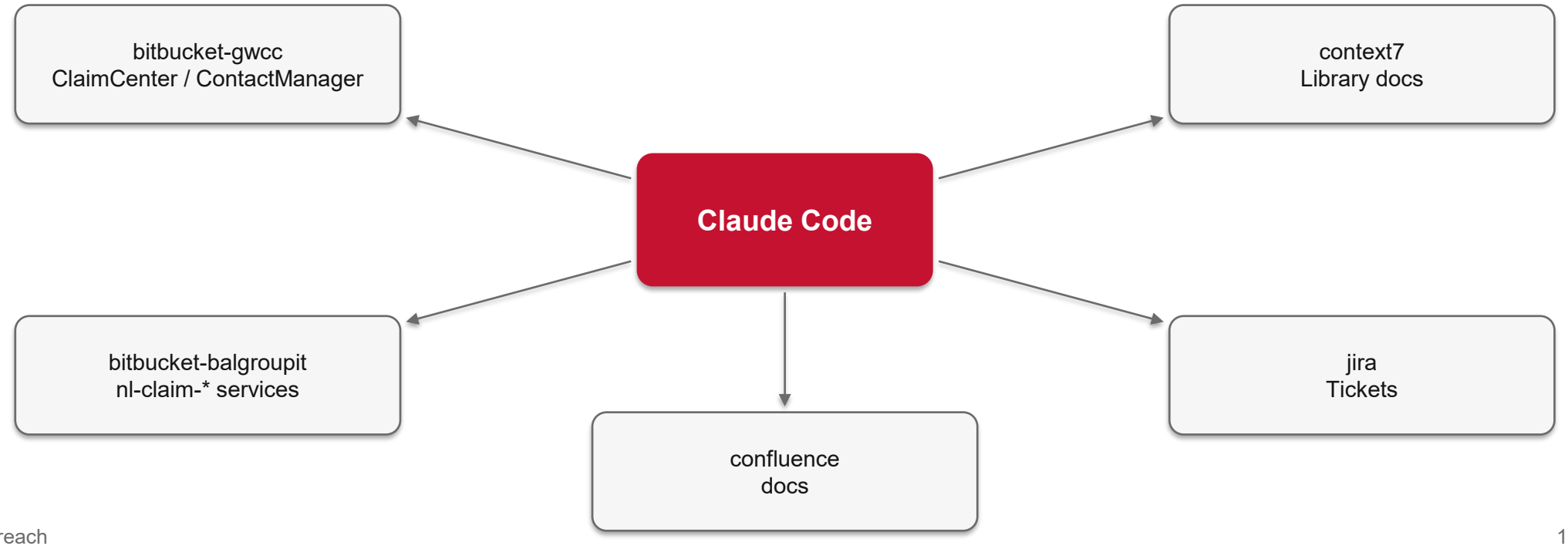
3. **Every proposal** now references the relevant ADR — or proposes updating it.

ADRs are the reasoning behind the code. The agent follows the reasoning, not just the patterns.

§ /init plus an ADR pointer

What it can reach.

Five servers wired up. The agent works on real artefacts, not transcripts of them.



helvetia

Workflows we ship.

CLAIMCENTER / GOSU

- nl-claim-gw-bugfix
- nl-claim-gw-git
- nl-claim-gw-arch-review
- nl-claim-arc42
- gosu

MICROSERVICES / JAVA

- nl-claim-svc-bugfix
- nl-claim-svc-git
- nl-claim-svc-code-review
- nl-claim-svc-arch-review

MAINTAINED CENTRALLY · applies to both

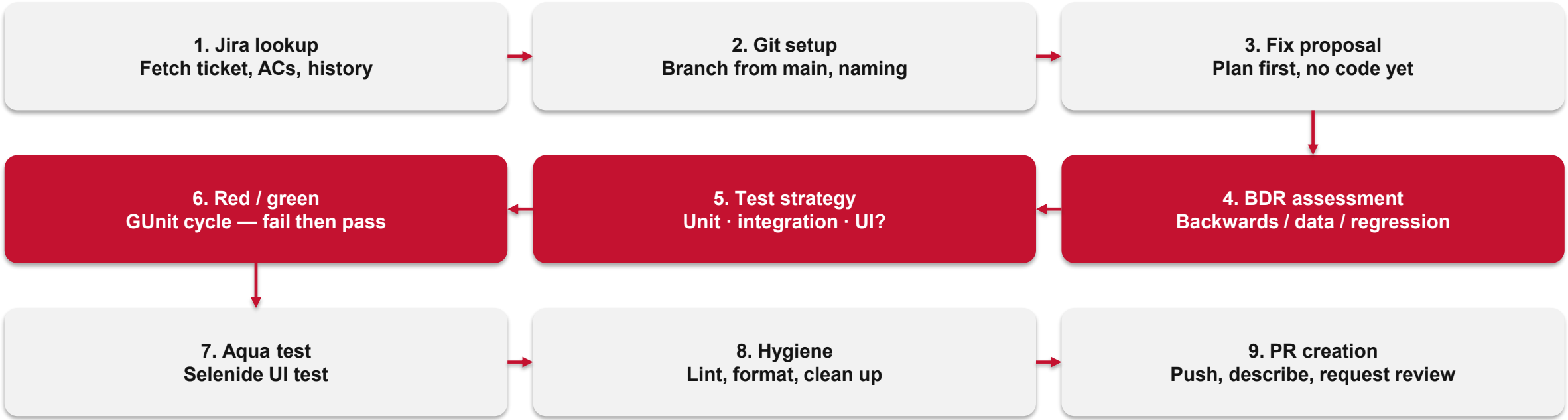
Shared utility skills: nl-claim-jenkins · jira-integration

Installed via nl-claim-dev-tools — one script, same skills & MCP servers across every nl-claim project.

helvetia

Nine steps. Every time.

One Markdown file turns a Jira ticket into a merged PR. The agent never skips a step.



Same nine steps every time — that is what makes it reviewable.

helvetia

03

Observations … and learnings

helvetia

Who does what.

Power tool, not colleague. The line stays bright.

AGENT DOES	HUMAN OWNS
→ Code drafts and refactors	✓ PR review and approval
→ Test scaffolding (red / green)	✓ BDR decision and sign-off
→ Confluence page drafts	✓ Test strategy choice
→ Jira analysis	✓ Stakeholder communication
→ Boilerplate — Jenkins, configs, glue	✓ Production consequences

Generated artefacts cross the line only after a human says so.

helvetia

Risk of AI Slop
Your name on the PR.

THE RULES

Whoever runs the agent owns the output. PR, commit, accountability.

Generated code is reviewed like any other code. No shortcuts.

Generated Confluence pages are read and approved before publishing.

Tests, BDR, sign-off: still human-decided.

If something breaks in prod, "Claude wrote it" is not an answer.

helvetia

Faster devs. New bottleneck.

AI gives developers more output. The bottleneck moves upstream — to BA / SME.



Three options:

- **Scale BA / SME —** hire more, train more
- **Slow developers down —** throttle AI use
- **Engage developers as BAs —** if willing and able — not everyone is both

helvetia

Coder → instructor.

What you're paid for shifts. The agent knows the patterns; you know the business.

OLD ASSET	NEW ASSET
− Java streams details	+ Understanding the business
− Uncle Bob's rules of thumb	+ Knowing the architecture and the responsibilities of the components
− Framework quirks by heart	+ Configuring the harness
− Boilerplate from memory	+ Prompting precisely
	+ Knowing what to review deeply

Do you still need to be a developer to be a ClaudeCoder? Yes.

Otherwise — who reviews the critical code (DB changes, security, business logic)?



helvetia

Coder → instructor.

What you're paid for shifts. The agent knows the patterns; you know the business.

OLD ASSET	NEW ASSET
− Java streams details	+ Understanding the business
− Uncle Bob's rules of thumb	+ Knowing the architecture and the responsibilities of the components
− Framework quirks by heart	+ Configuring the harness
− Boilerplate from memory	+ Prompting precisely
	+ Knowing what to review deeply

Do you still need to be a developer to be a ClaudeCoder? Yes.

Otherwise — who reviews the critical code (DB changes, security, business logic)?

helvetia

Why devs hesitate.

Three concerns we hear — and one answer to all of them.

01

QUALITY

"Will the code quality really be there?"

02

ROLE

"I'm a dev — not a reviewer or a BA."

03

CAREER

"Am I working myself out of a job?"

The train has started.

Jump on, or stay behind.

helvetia

What works.

- ✚ Feature development — e.g. EC2 — 50–70% acceleration
- ✚ Bugfix skill — same reproducible flow per developer
- ✚ Automatic code review — scheduled every 30 minutes
- ✚ Confluence pages — BDRs, ADRs in minutes
- ✚ Jenkins boilerplate — nobody enjoys writing it
- ✚ Jira triage — refinement, duplicate detection

helvetia

What to watch.

- Velocity measurement - how can we prove that we are faster – and how much?
- Code and documentation volume can outrun review capacity
- Polished prose is not the same as accurate prose
- Jira complexity analysis — did not work as expected
- Skill drift — skills age if nobody maintains them
- Knowledge concentration — who actually learns the system?
- Cost discipline — long sessions and parallel agents add up

helvetia

Fast. But untraceable.

Phase one was tuned to stabilise a running system. Aurora has to build a new one.

TRACEABILITY

- **No spec — just the diff**
  intent lived in the prompt, not in Git
- **Review on the diff only**
  the reviewer reverse-engineers the intent
- **No proof per acceptance criterion**
  “the suite is green” was the evidence
- **Knowledge lived in chat sessions**
  gone when the session closed

PROCESS

- **Prompt-driven feature work**
  result depended on session and person
- **Tuned for bugfix and stabilisation**
  new development possible — unstructured
- **No defined human gates**
  accountability was a rule, not a step
- **Skills grew where it hurt**
  no process tying them together

END-TO-END · BUSINESS

- **Not wired into the project flow**
  Jira, business and code side by side
- **Business artefacts unstructured**
  only BDRs and ADRs carried IDs
- **BA / SME role undefined**
  who delivers what — and when?

Good enough to fix a system. Not good enough to build one. → **Part four: Aurora**

helvetia

04

How we now started to use it: Aurora.

helvetia

Two homes. One rule.

What the business decided lives in Confluence. How a system realises it lives as Markdown in Git.

BUSINESS ARTEFACTS · CONFLUENCE

BR — **Business Rule · BR-NNNN**
one sentence, one owner, a permanent ID

BC — **Business Concept · BC-NNNN**
a shared definition — used verbatim, never paraphrased

BDR — **Business Decision Record · BDR-NNNN**
what the business decided, and who decided it

Owned by the BA · permanent IDs · cited by ID, never restated

ARCHITECTURE DOCS · MARKDOWN IN GITHUB

arc42 — **12 sections per component**
one Markdown file per section, next to the code

ADR — **ADR-<CODE>-### · §09 decisions**
how a system realises it — one .md per decision

NFR — **NFR-<CODE>-### · §10**
the only home for a non-functional requirement

Changed by pull request · approved via CODEOWNERS · versioned with the code

Business wants it → a BDR in Confluence · A system realises it → an ADR in Git · Nobody had a choice → an arc42 §02 constraint

helvetia

From Intent to green tests.

Five artefacts, four human gates — one at every transition.



FOUR HUMAN ACTS — THE SKILLS SIMULATE NONE OF THEM

1 **The BA accepts the Intent**
— before anyone designs

2 **BA + Developer accept the spec**
SA signs off if architecture-relevant

3 **The Developer’s plan self-review**
— its first row is a hard stop

4 **The Code Owner approves the PR**
on the proof per criterion

Every artefact is reviewable before the next one exists — the Intent names no file, the spec names no change target, the plan re-decides nothing.

Not invented here.

Three inputs. We assembled them — we did not invent them.

ANTHROPIC
Agentic engineering guidance

- Plan before you build
- Verify everything, then iterate
- Optimise the harness, not the model

SIMON MARTINELLI
Spec-Driven Development

- The spec is the primary artefact
- Argue with the document, not the diff
- Acceptance criteria become named tests

OUR OWN PHASE ONE
What the hand tools taught us

- Skills decay if nobody owns them
- The bottleneck moves to BA / SME
- Volume can outrun review capacity

WHAT WE PUT TOGETHER

Intent · spec.md · plan.md — three artefacts, four human gates, and the registers every artefact cites.

None of the parts is novel. The work was deciding which of them survive contact with an insurance project.

helvetia

Early days. Strong pull.

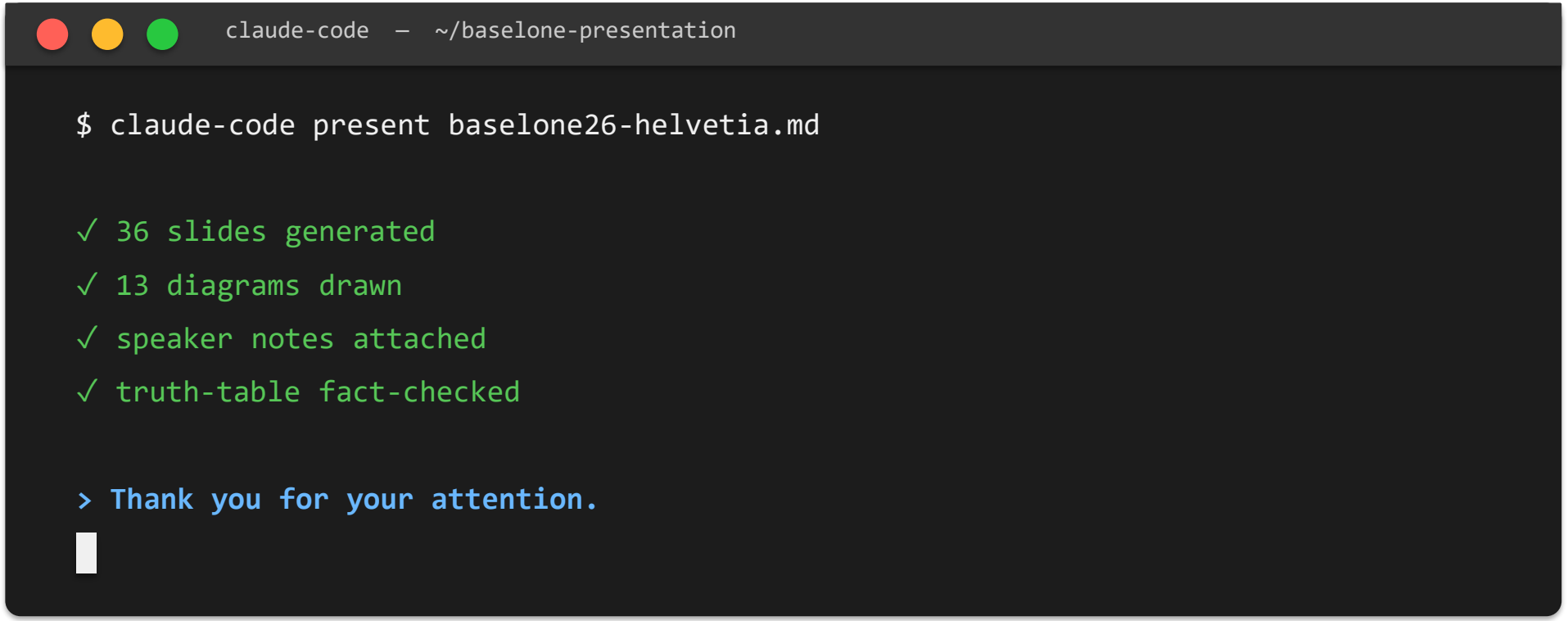
5 teams, 49 people — 18 Devs · 10 BAs · 8 SAs · 5 POs · 3 SMEs. What they tell us so far.

- ✚ Developers value the process
- ✚ BAs and POs dive into Claude Code — highly motivated, even in a terminal
- ✚ Good feedback from the teams overall
- ✚ Broad buy-in across the board
- → Fine-tuning still needed — the process is not finished

The bottleneck moved upstream to BA / SME — now they are picking up the tool themselves.

Thank you.

This deck was written end-to-end by Claude Code. Yes, including this slide.



FEEDBACK

helvetia

Your feedback.

Scan the code and rate the session — honest beats kind.



Beyond the Hype: Claude Code in a Real Insurance Project

Frank Baier · BaselOne26 · October 2026

helvetia

Aurora in action.

Three short recordings — one per skill. Click a card to play.

`/aurora:intent`

01 Intent

The Jira Story becomes an Intent — written by the Originator, co-defined with SME + BA.

▶ 1_AuroraIntent_Demo.mp4

`/aurora:spec`

02 spec.md

The spec lands in git — co-authored by BA + Developer.

▶ 2_AuroraSpec_Demo.mp4

`/aurora:plan`

03 plan.md

The plan is written and self-reviewed by the Developer.

▶ 3_AuroraPlan_Demo.mp4

Intent → spec.md → plan.md · the three artefacts from the SDLC slide, live