AGENTIC MEETUP/ HAMBURG/ 2026

Agentic Coding

in the wild

Ole Wendland

👋

Hi

Ole Wendland

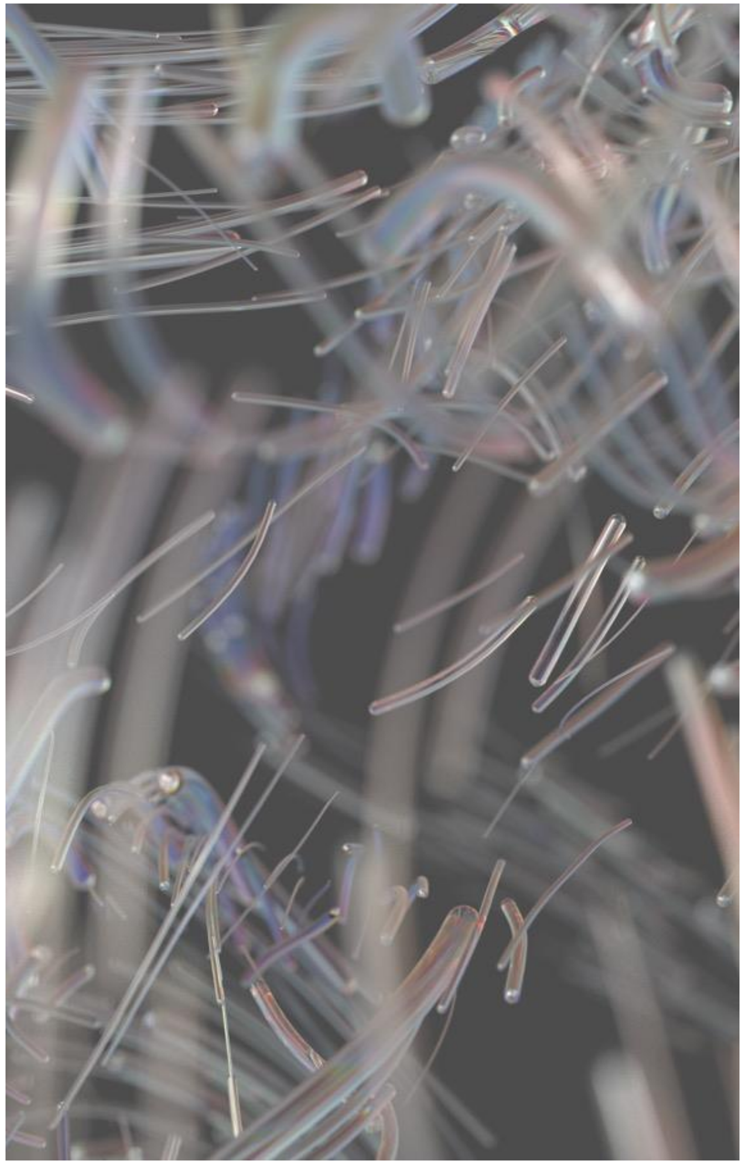
Agenda

Motivation

Learnings and Advice

Challenges

Wrap-up

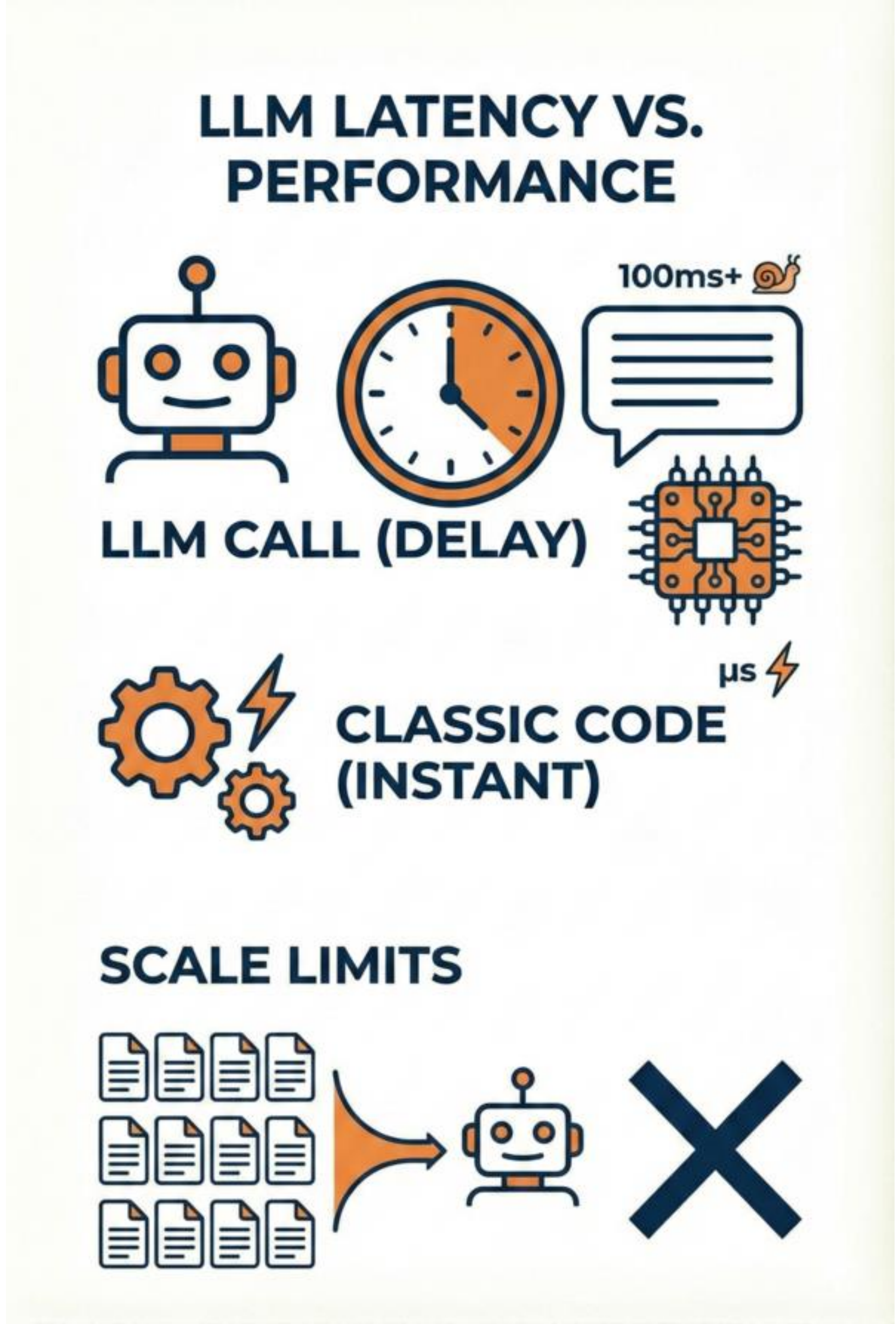


AI is perfect! It will solve everything!

AI is perfect!

Except for:

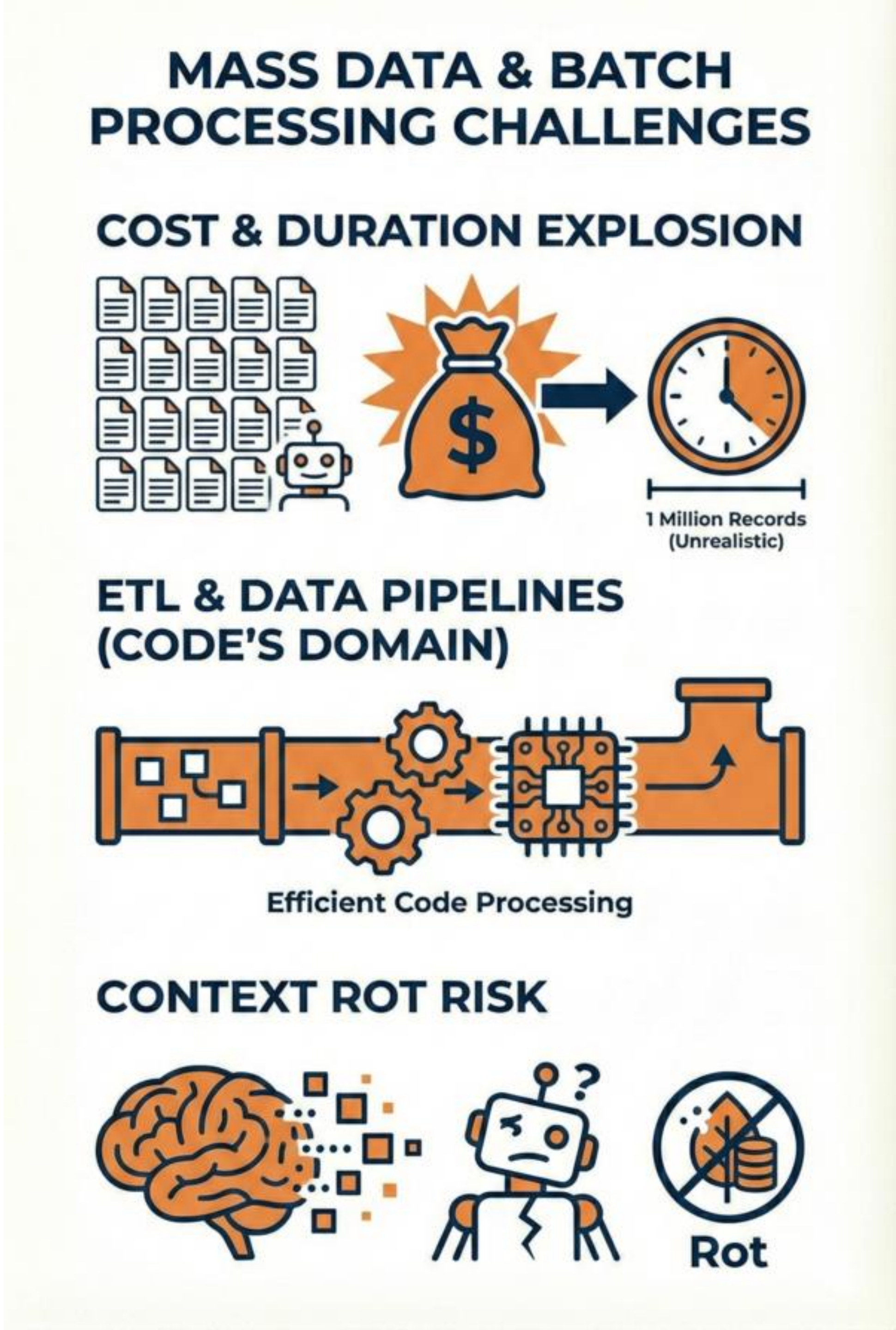
- Speed



AI is perfect!

Except for:

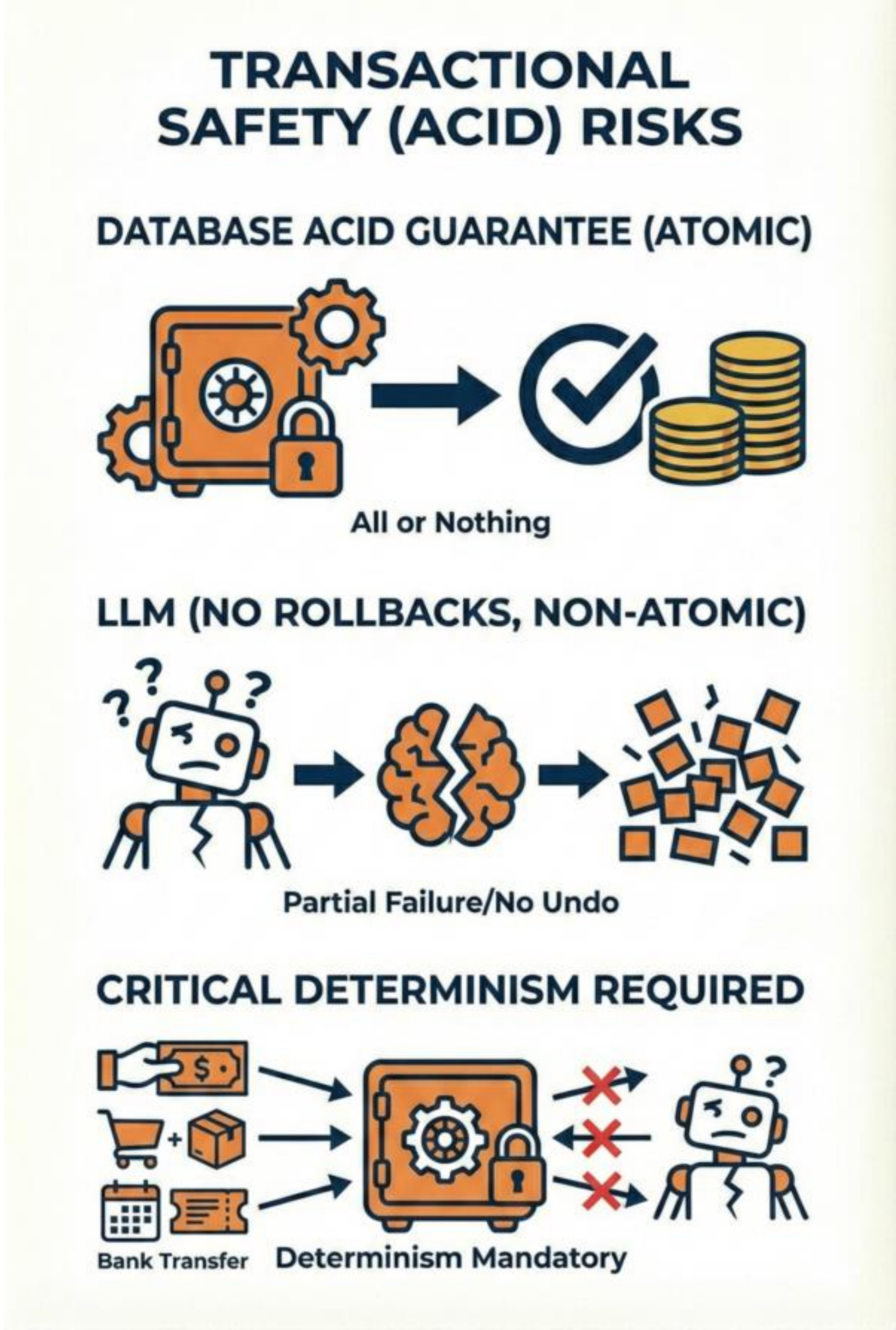
- Speed
- Massdata


MASS DATA & BATCH PROCESSING CHALLENGES
COST & DURATION EXPLOSION
$
1 Million Records (Unrealistic)
ETL & DATA PIPELINES (CODE'S DOMAIN)
Efficient Code Processing
CONTEXT ROT RISK
Rot

AI is perfect!

Except for:

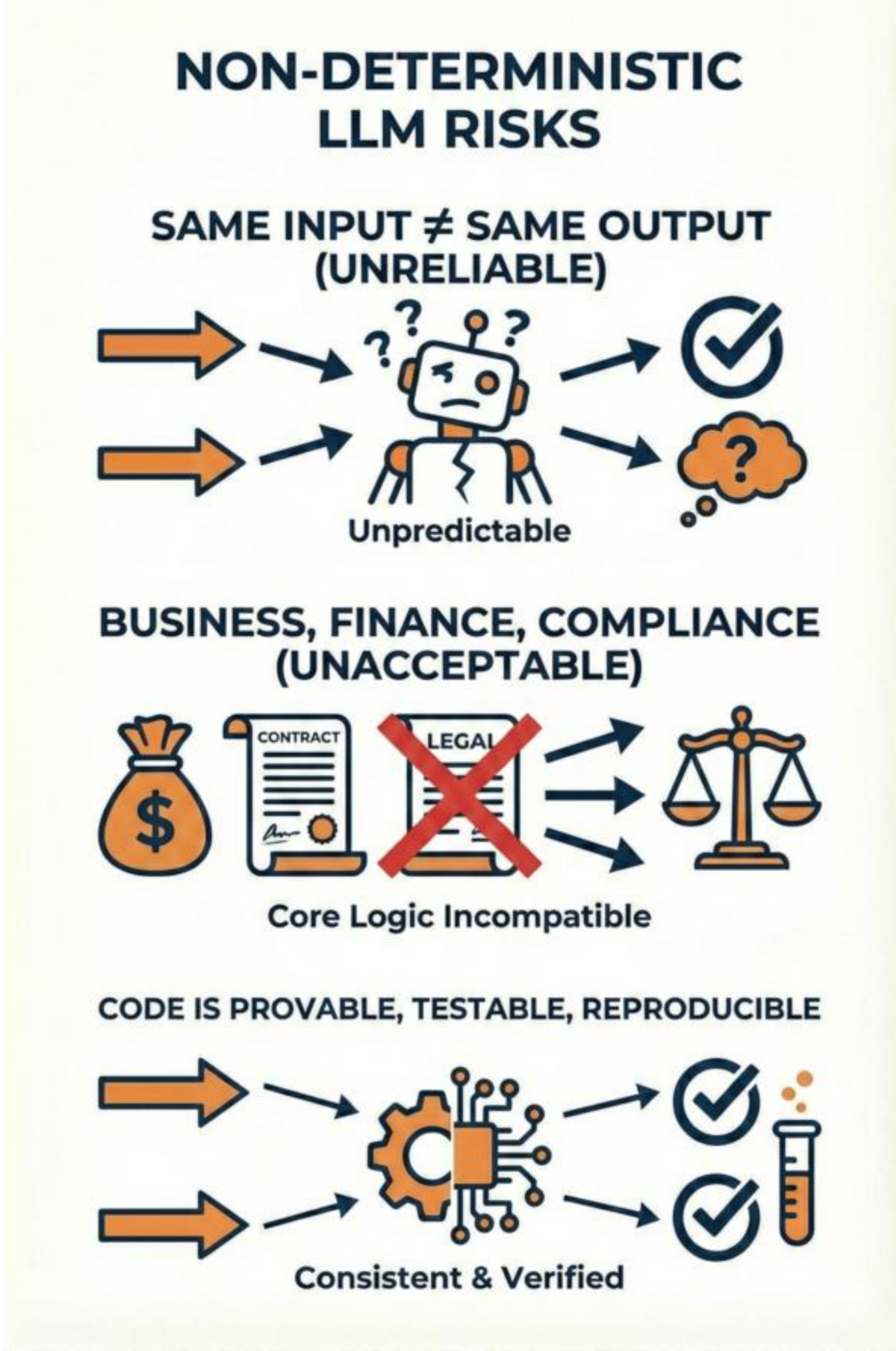
- Speed
- Massdata
- Transactionional Security

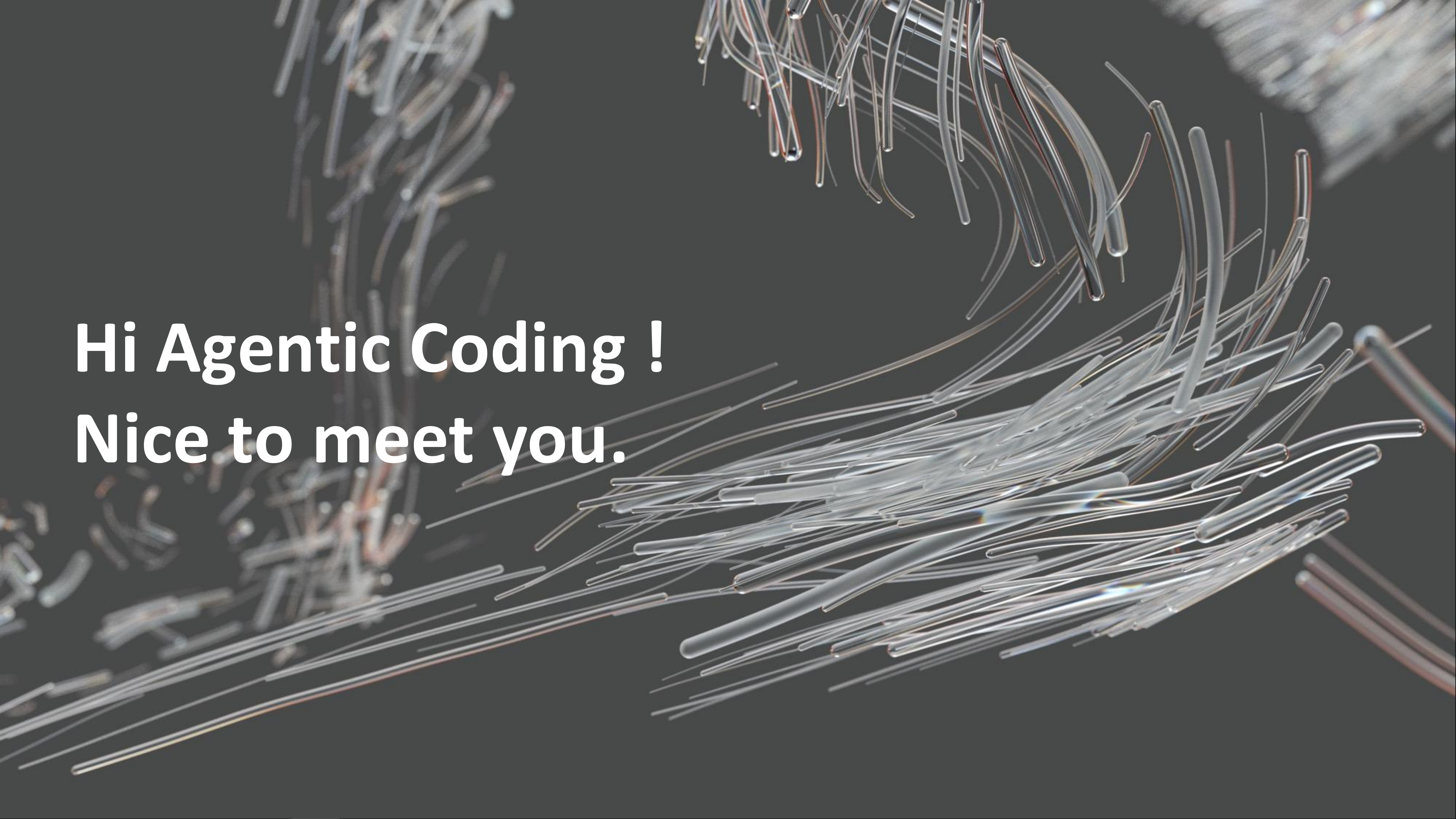

TRANSACTIONAL SAFETY (ACID) RISKS
DATABASE ACID GUARANTEE (ATOMIC)
All or Nothing
LLM (NO ROLLBACKS, NON-ATOMIC)
Partial Failure/No Undo
CRITICAL DETERMINISM REQUIRED
Bank Transfer
Determinism Mandatory

AI is perfect!

Except for:

- Speed
- Massdata
- Transactional Security
- Reliability

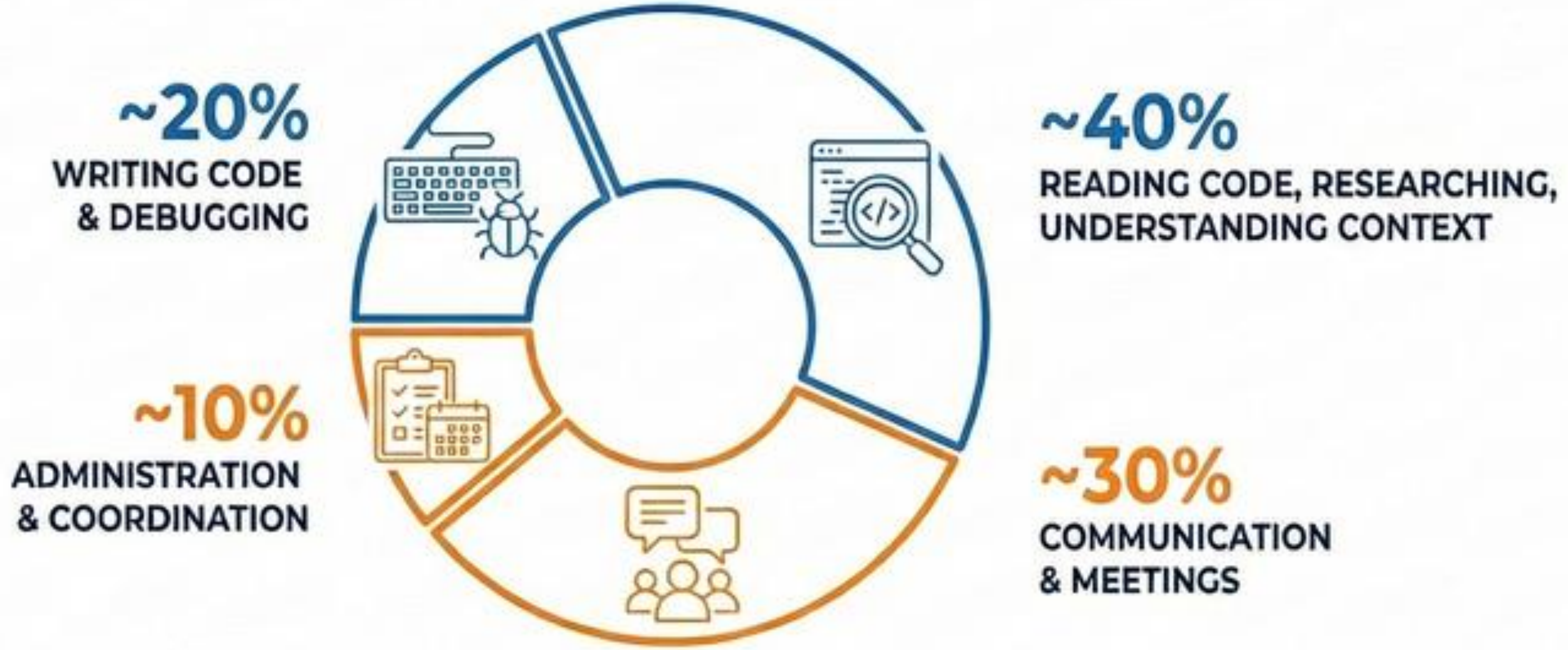

NON-DETERMINISTIC LLM RISKS
SAME INPUT ≠ SAME OUTPUT (UNRELIABLE)
Unpredictable
BUSINESS, FINANCE, COMPLIANCE (UNACCEPTABLE)
CONTRACT
LEGAL
Core Logic Incompatible
CODE IS PROVABLE, TESTABLE, REPRODUCIBLE
Consistent & Verified



Hi Agentic Coding !
Nice to meet you.

THE MYTH: SOFTWARE DEVELOPMENT = CODING

MOST DEVELOPER TIME IS **NOT** SPENT WRITING CODE



Writing

- Boilerplate: fully delegatable
- Tests: mostly delegatable
- Glue code: mostly delegatable
- Business logic: … maybe

- Less noise, more focus

WRITING ≠ TYPING

VALUE COMES FROM DELEGATION

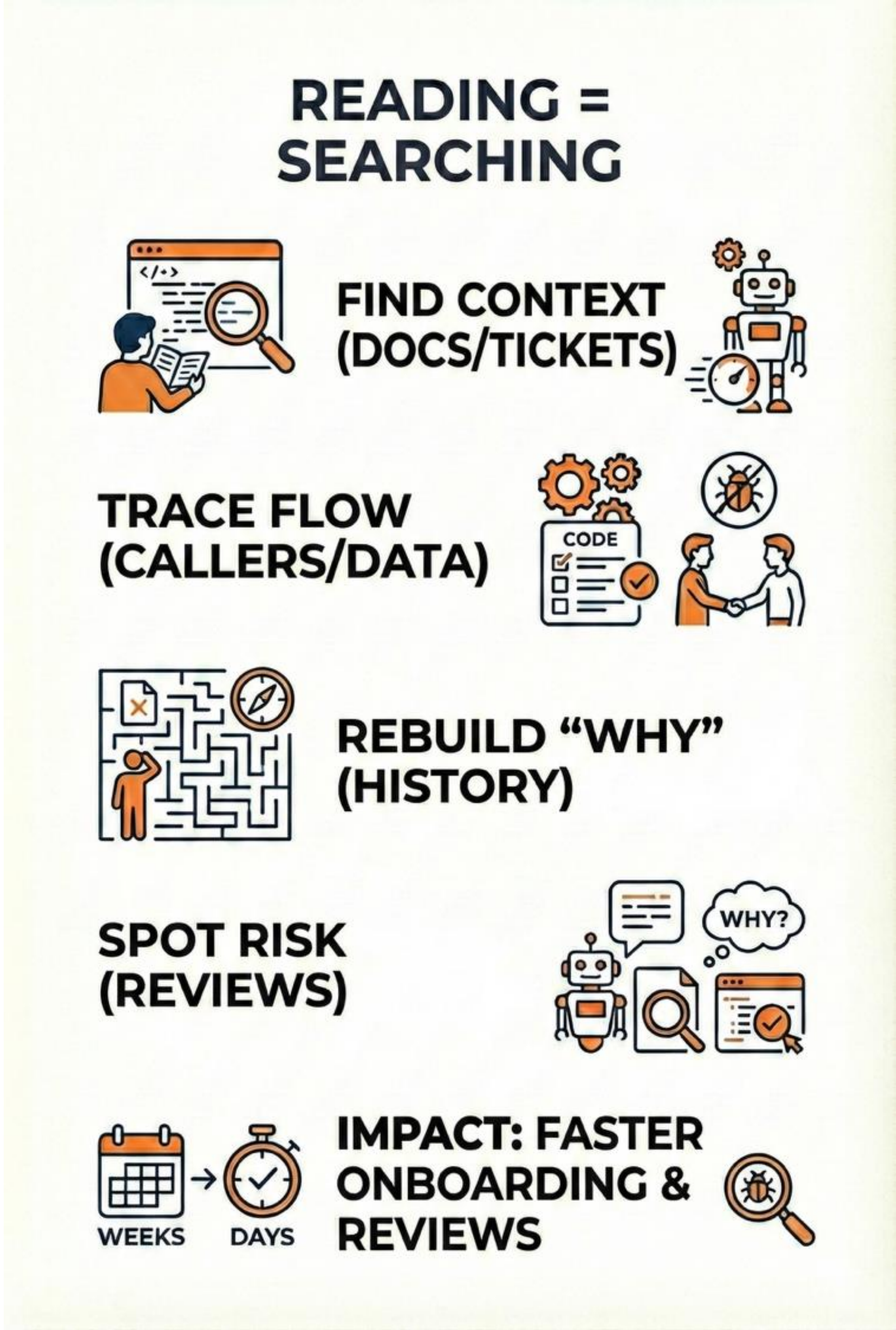
BOILERPLATE. 100% DELEGATION. Clear Patterns.

TESTS & GLUE. HIGH DELEGATION. Standard Practices.

BUSINESS LOGIC. PARTIAL DELEGATION. Human Decisions, Agent Drafts.

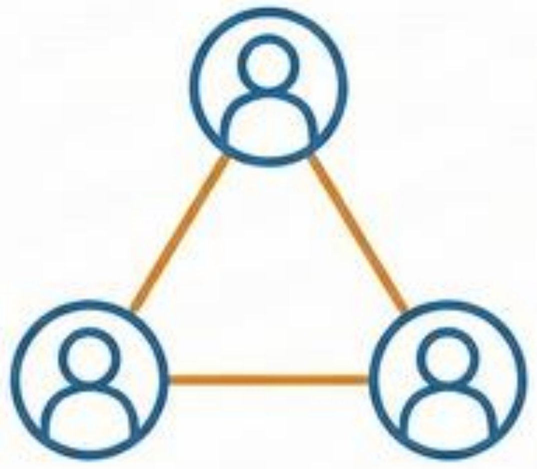
Reading

- Reading = Searching
- Find context (docs/tickets)
- Trace flow (callers/data)
- Rebuild “why” (history)
- Spot risk (reviews)

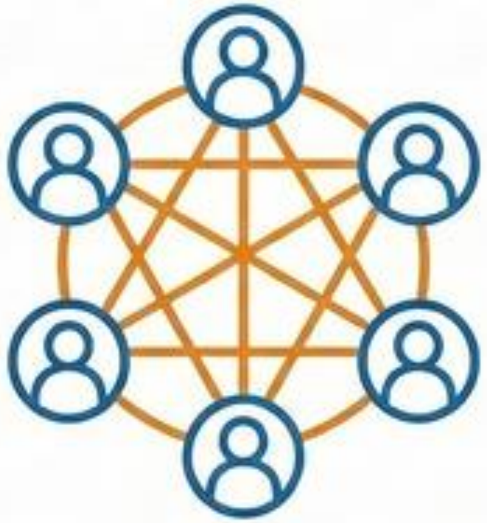

READING = SEARCHING
FIND CONTEXT (DOCS/TICKETS)
TRACE FLOW (CALLERS/DATA)
CODE
REBUILD “WHY” (HISTORY)
SPOT RISK (REVIEWS)
WHY?
WEEKS
DAYS
IMPACT: FASTER ONBOARDING & REVIEWS

BROOKS' LAW: WHY SO MUCH COMMUNICATION?

COMMUNICATION OVERHEAD GROWS FASTER THAN THE TEAM



3-PERSON TEAM → 3 LINKS



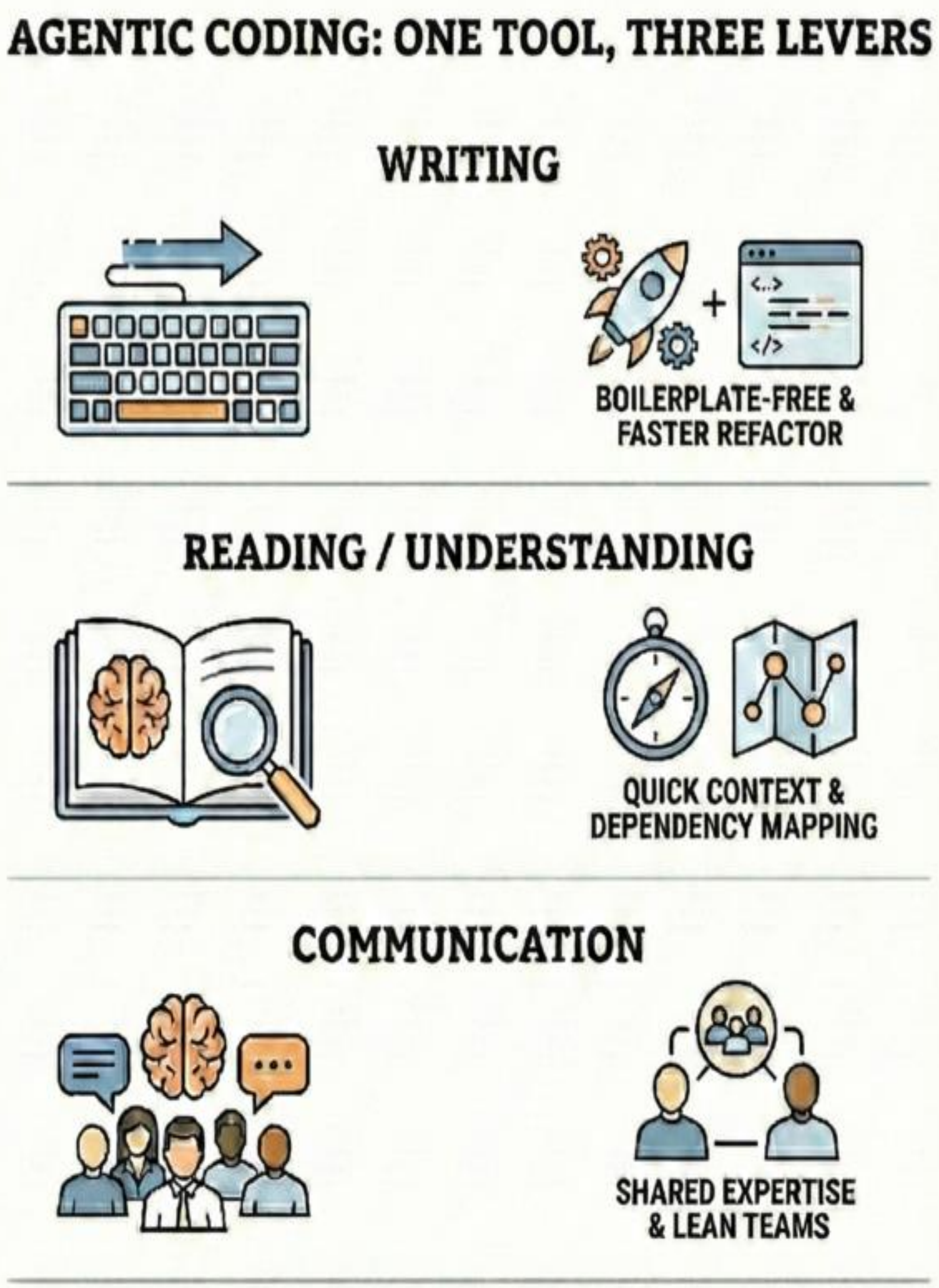
6-PERSON TEAM → 15 LINKS

COMMUNICATION PATHS: $\frac{n \times (n-1)}{2}$

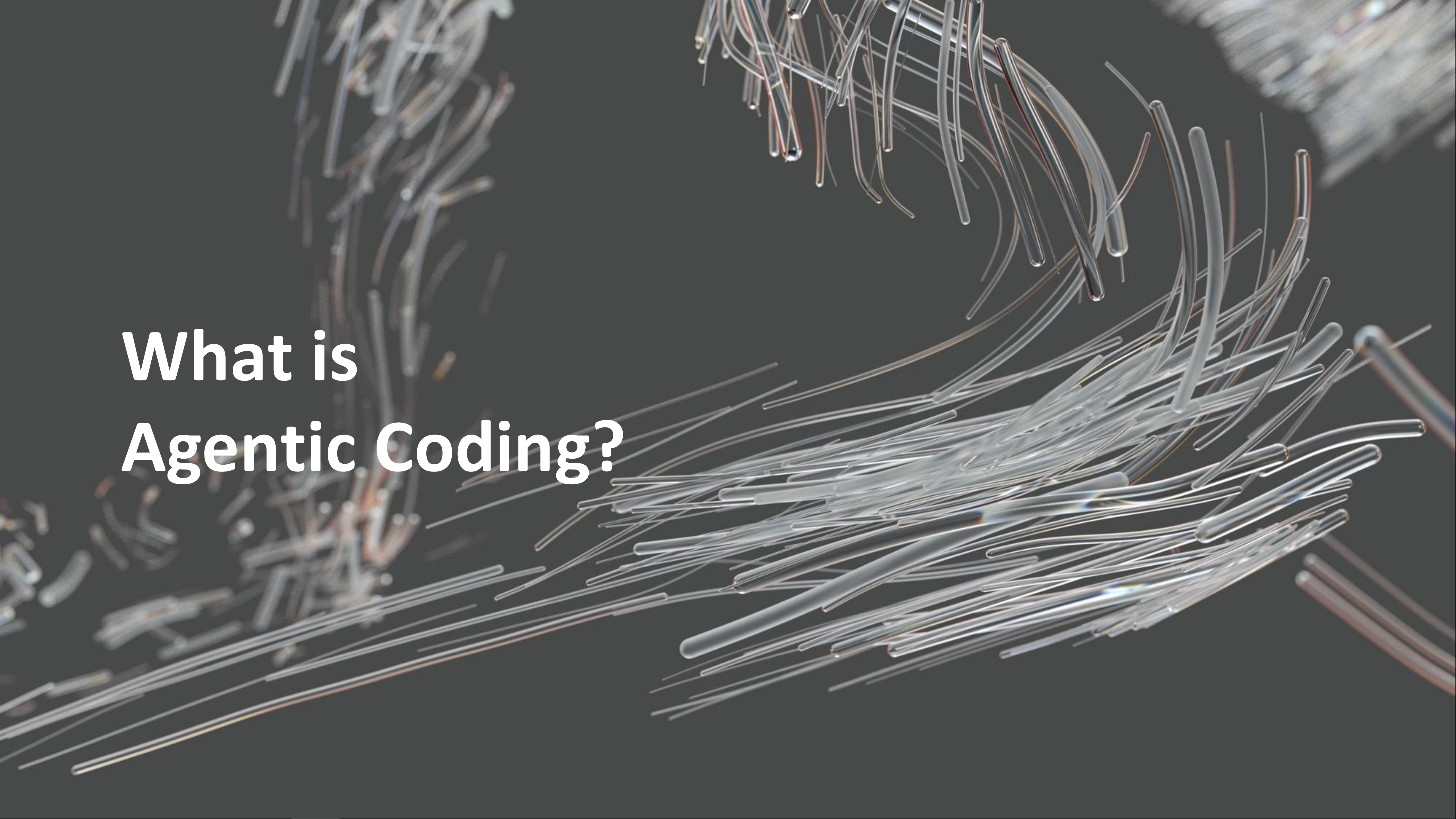
SPECIALISTS INTRODUCE:
NEW DEPENDENCIES, MORE COORDINATION, MORE HANDOVERS

Software is staying

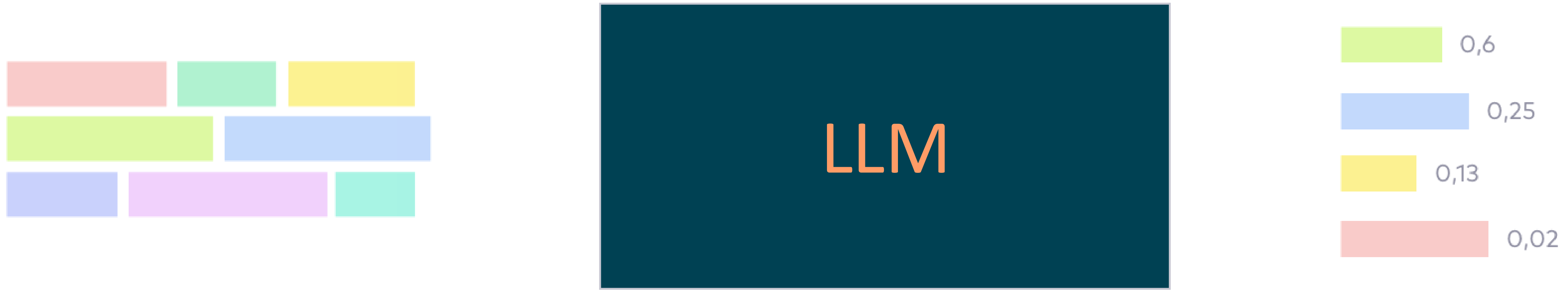
- AI can't replace 80% of the tasks
- Agentic Coding increases efficiency on multiple levels
- Cheaper Software = less AI


AGENTIC CODING: ONE TOOL, THREE LEVERS
WRITING
BOILERPLATE-FREE & FASTER REFACTOR
READING / UNDERSTANDING
QUICK CONTEXT & DEPENDENCY MAPPING
COMMUNICATION
SHARED EXPERTISE & LEAN TEAMS

What is Agentic Coding?



LLMs are ***Calculators*** for the next best Token



Input Sequence → Linear Algebra → Output Probability

Why context ***matters***

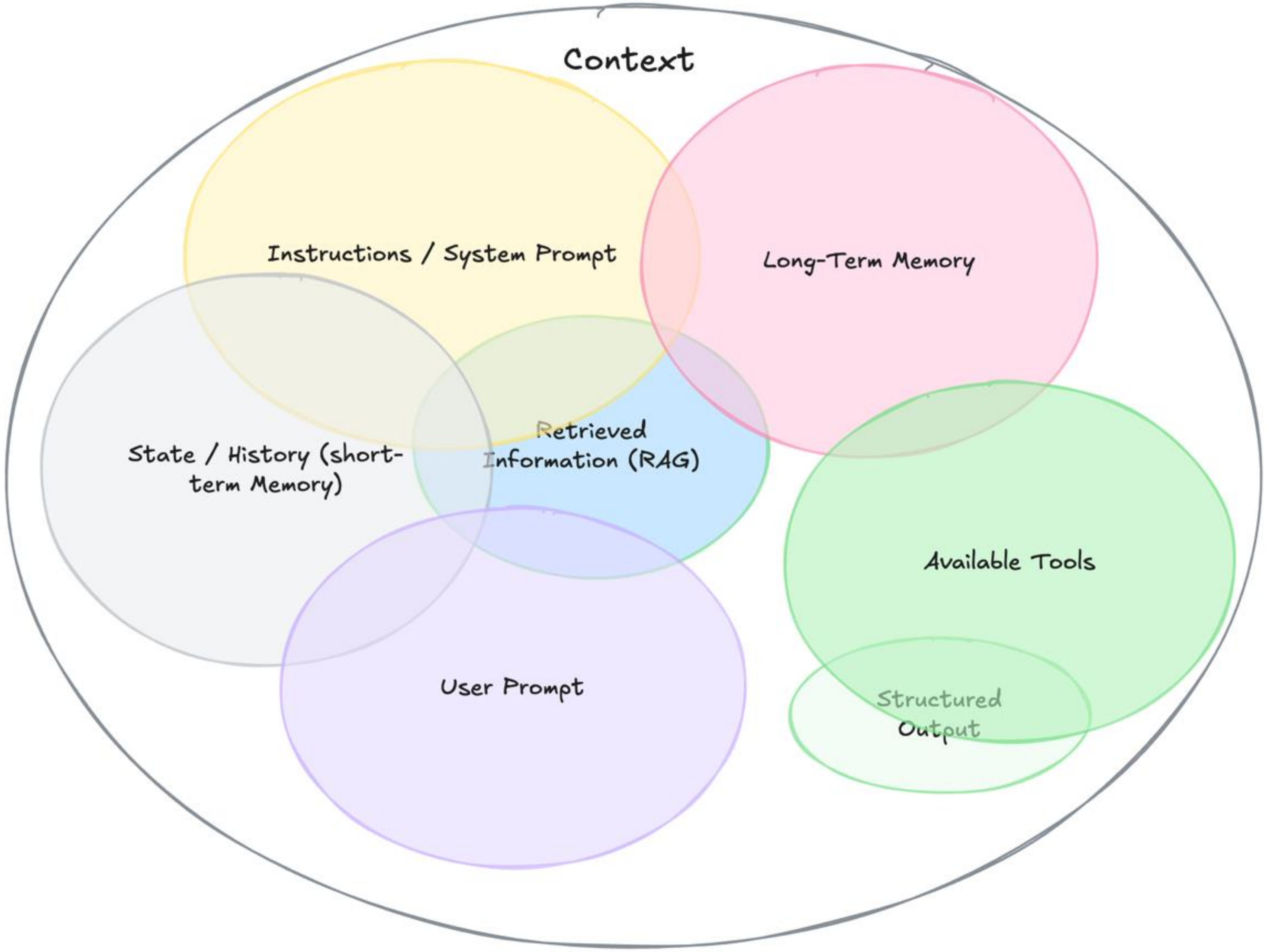
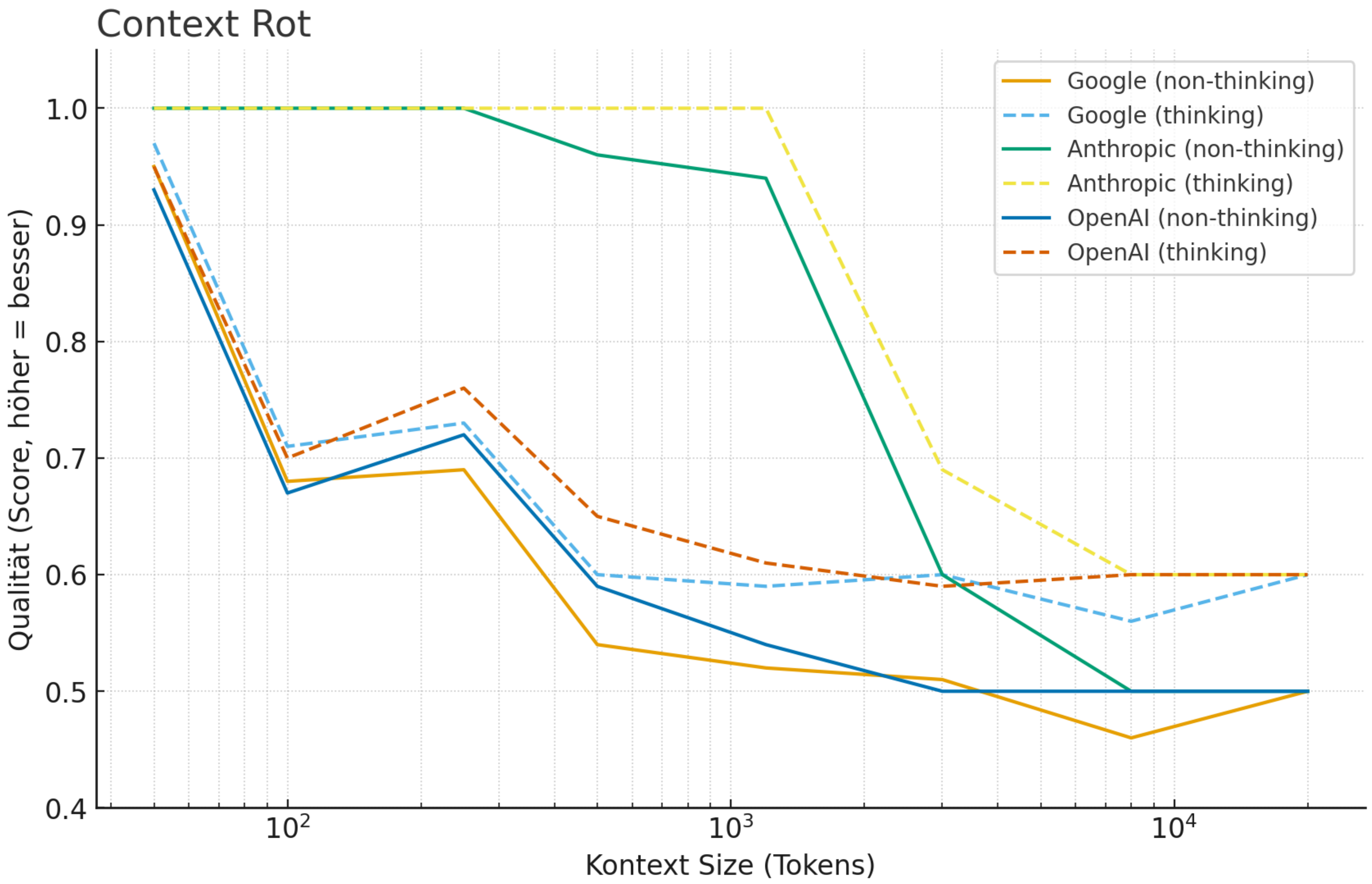

Context
Instructions / System Prompt
Long-Term Memory
State / History (short-term Memory)
Retrieved Information (RAG)
Available Tools
User Prompt
Structured Output

Image by https://www.philschmid.de/context-engineering

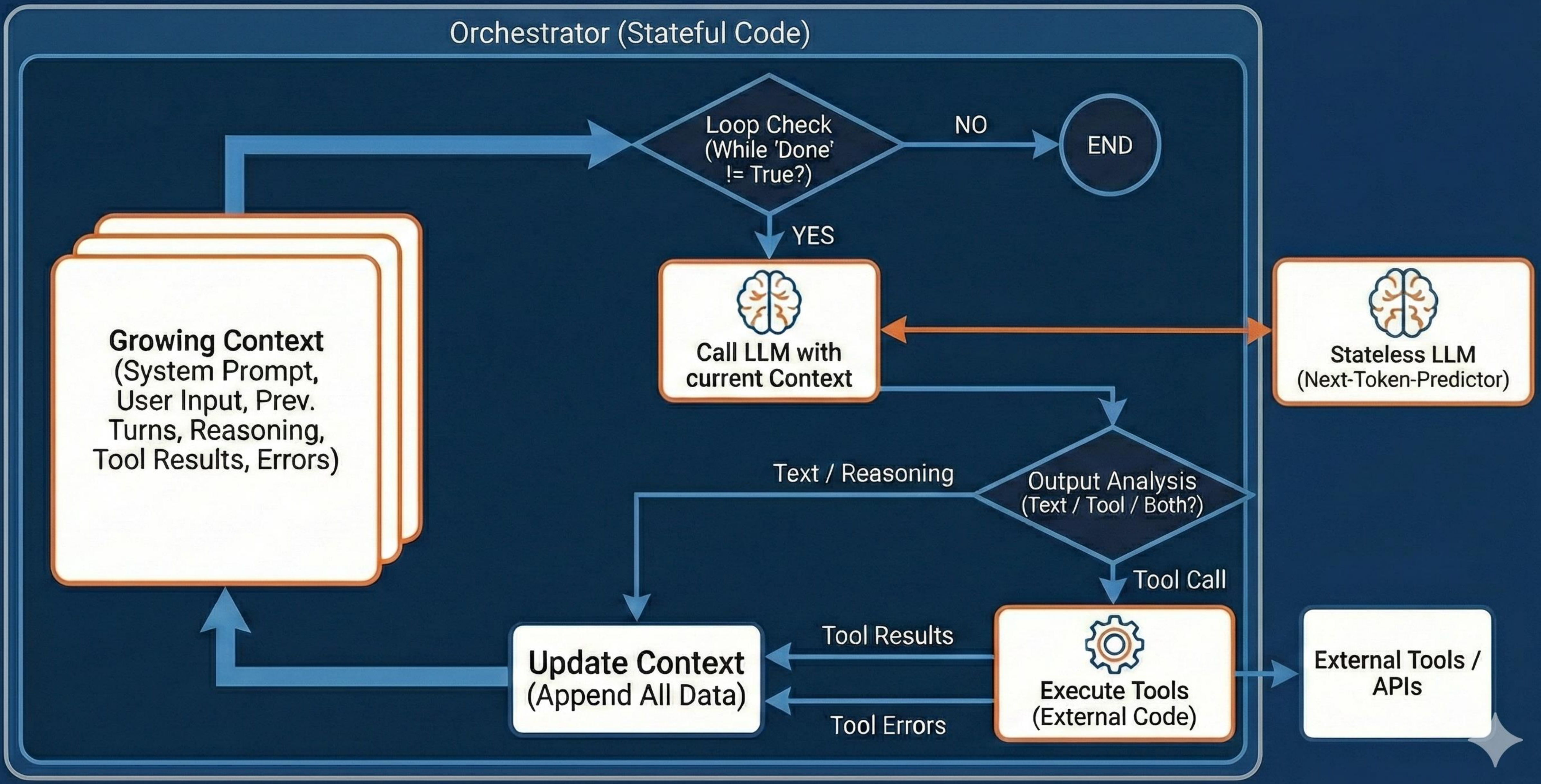
Dealing with context is *hard*

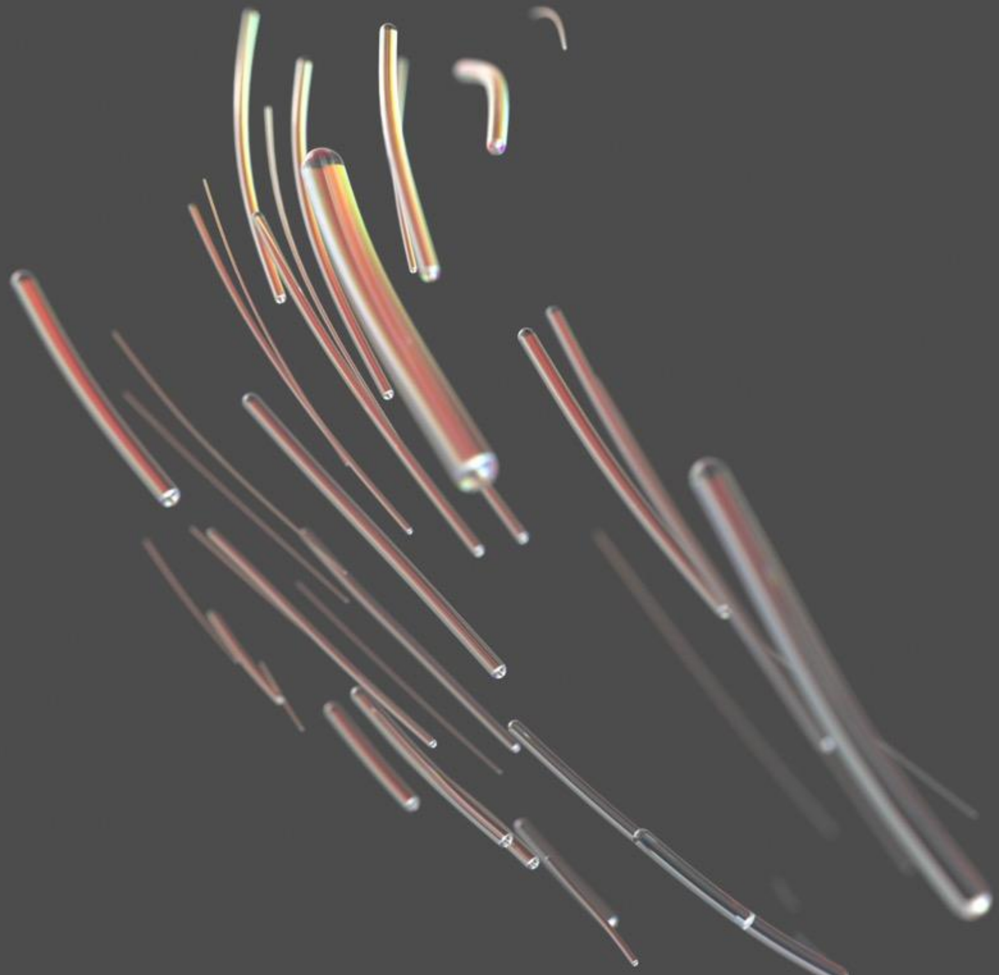
Overflow / Distraction / Poisoning / Confusion / Clash / Rot



https://research.trychroma.com/context-rot

Agentic Loop


Orchestrator (Stateful Code)
Loop Check (While 'Done' != True?)
NO
END
YES
Growing Context (System Prompt, User Input, Prev. Turns, Reasoning, Tool Results, Errors)
Call LLM with current Context
Stateless LLM (Next-Token-Predictor)
Text / Reasoning
Output Analysis (Text / Tool / Both?)
Tool Call
Update Context (Append All Data)
Tool Results
Tool Errors
Execute Tools (External Code)
External Tools / APIs



Agent Features

Memory /

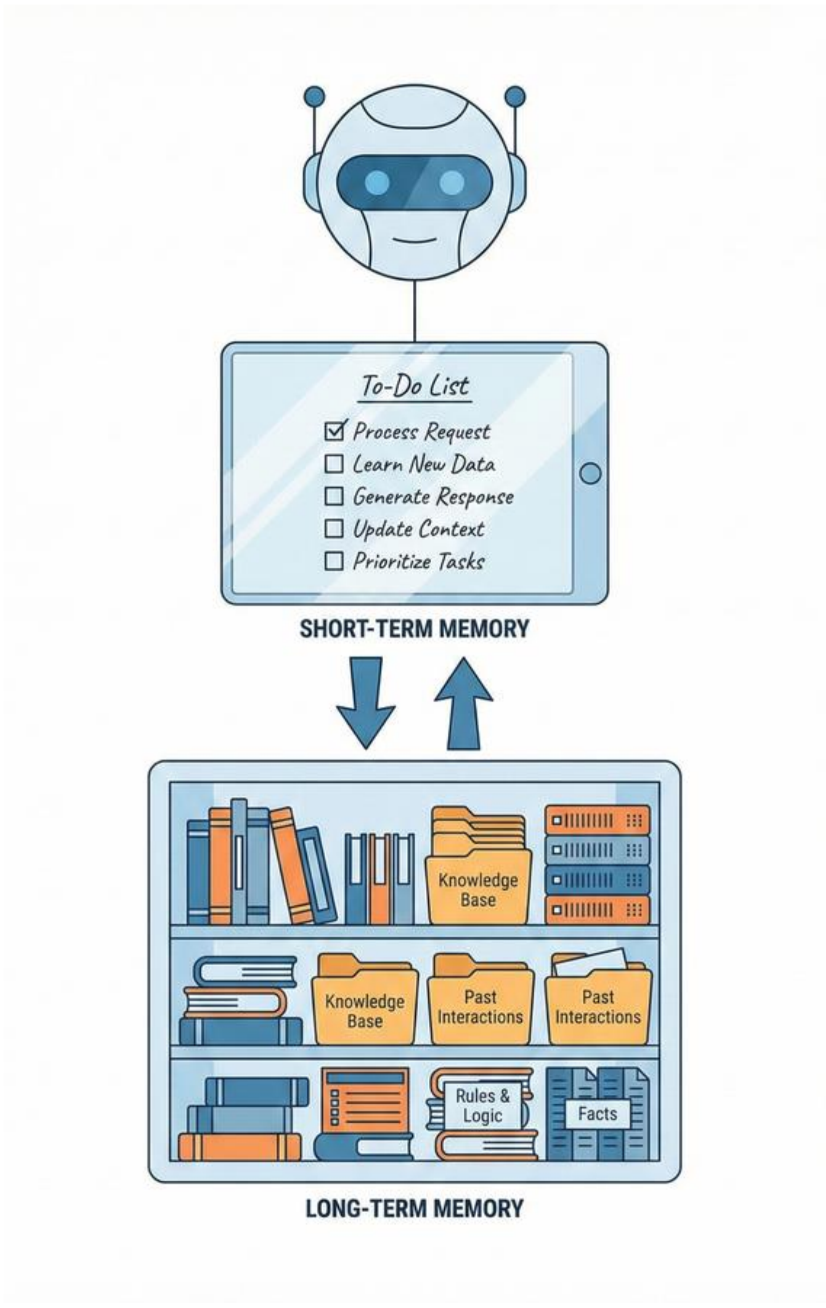
Tools /

Hooks /

Subagents

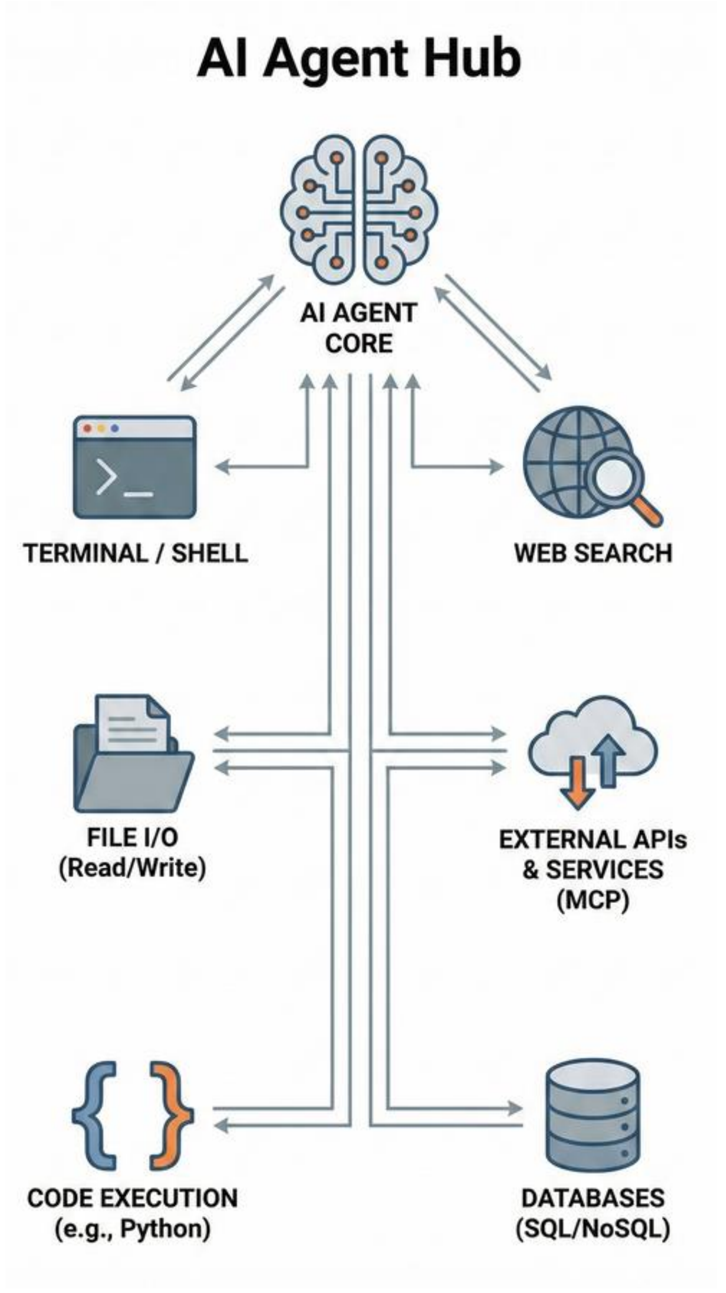
Memory

- **Context window:** Short-term memory
- **Project context:** CLAUDE.md
- **Persistent memory:** Diagrams, Glossar
- **Tip:** Maintain your agent memory like documentation



Tools

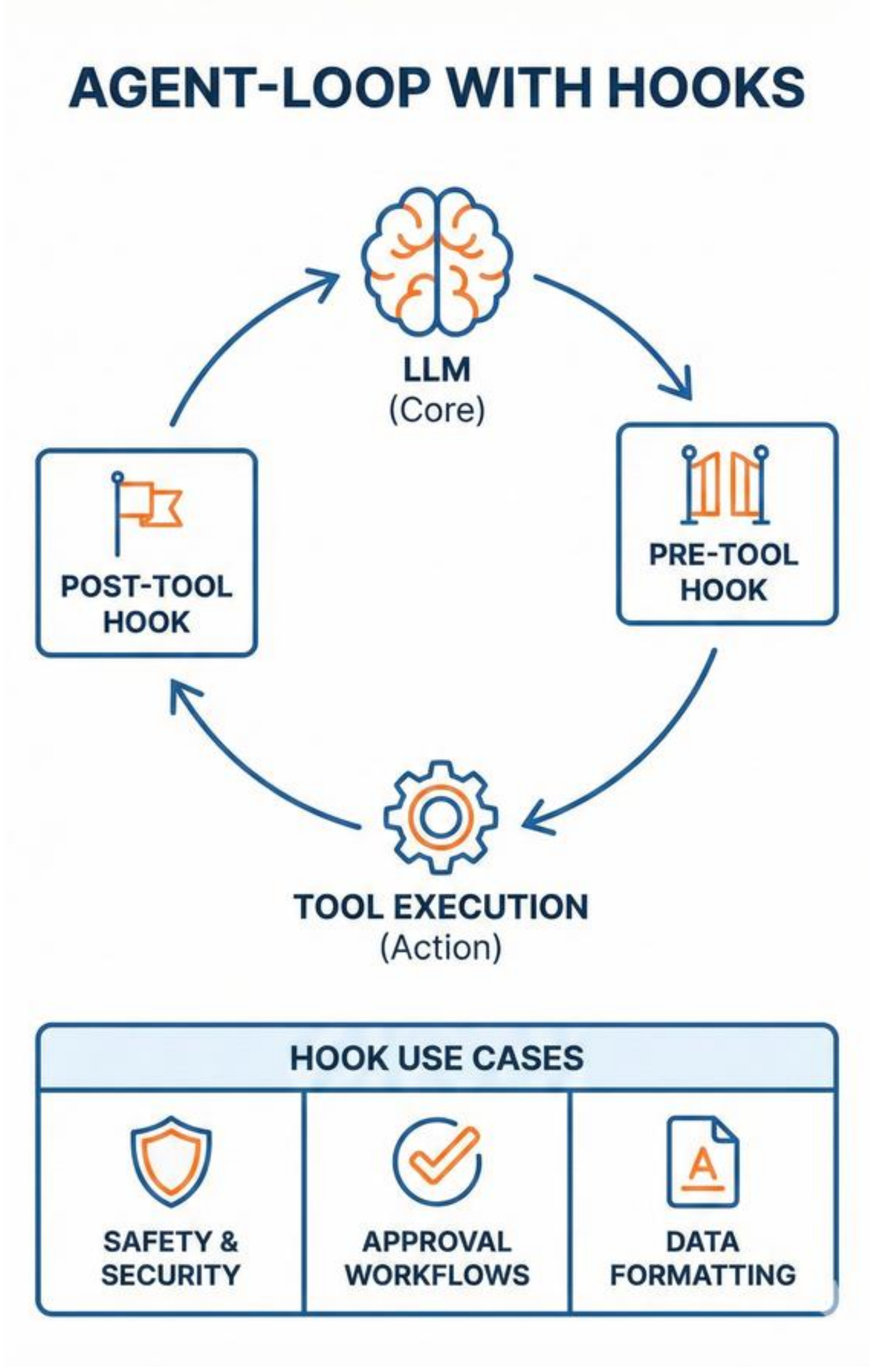
- **Built-in tools:** files, terminal, search
- **Agent doesn’t “see” — it reads**
- **Each tool call costs time & context**
- **MCP:** standard tool interface
- **Examples:** read Jira, search docs, trigger deploys



Hooks

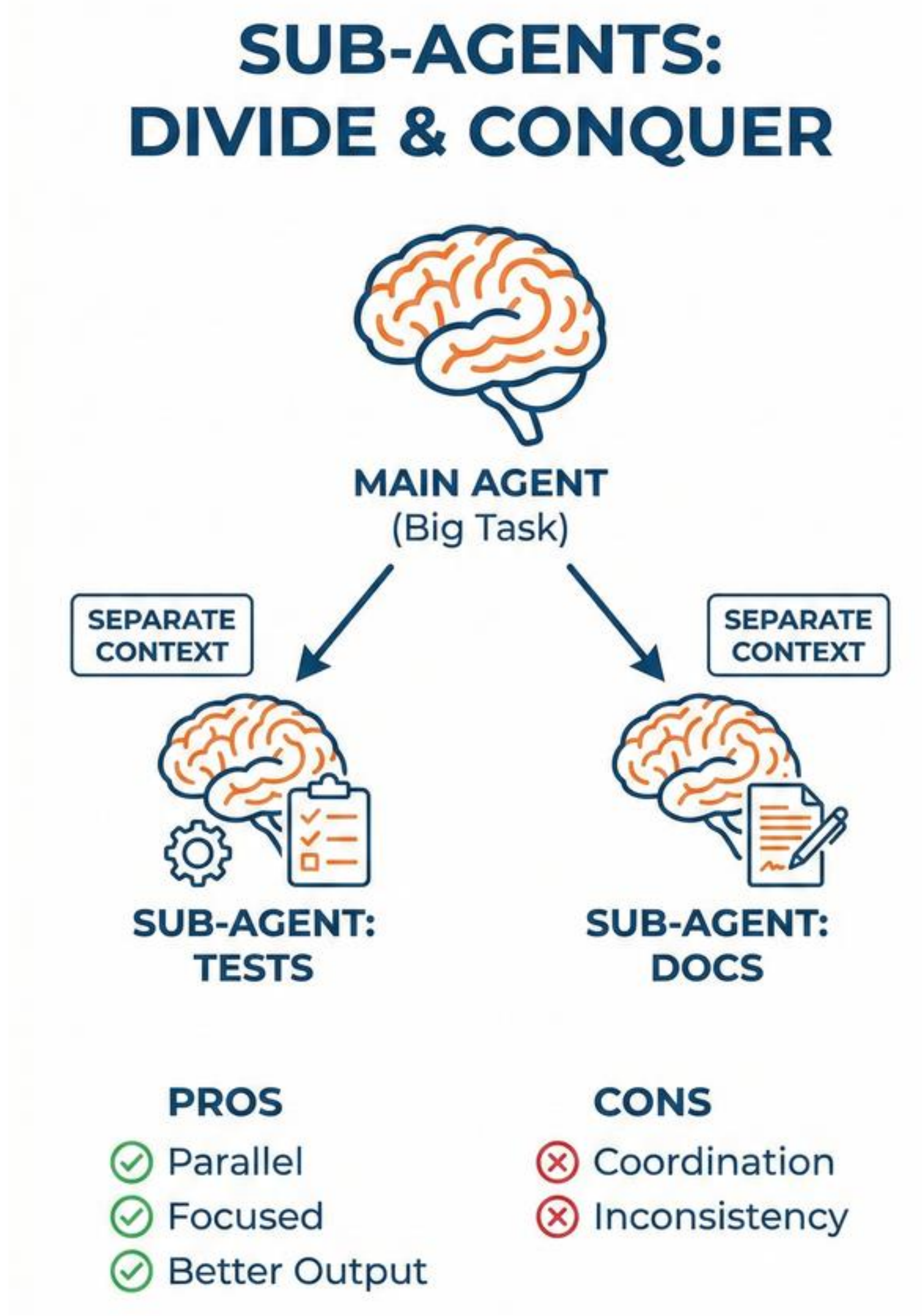
- **Hooks = intercepter** in the agent loop
- **PreToolCall:** before a tool runs
- **PostToolCall:** after a tool runs
- **Use cases:** formatting, approvals,

  safety



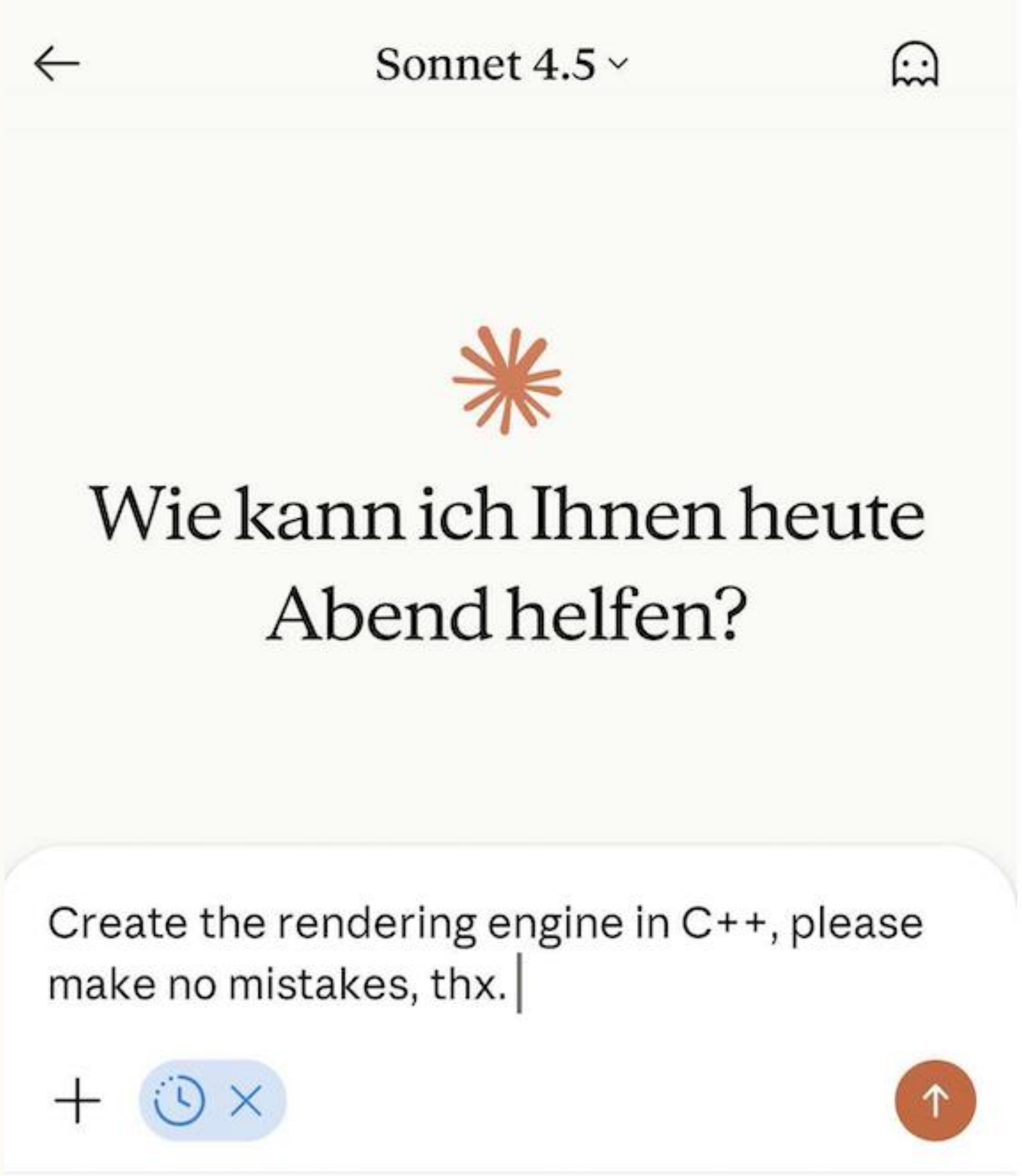
Subagents

- **Spawn to save context**
- **Example**: code → tests + docs
- **Separate context per subagent**
- **Pros:** parallel, focused, better output
- **Cons**: coordination, inconsistency



Best Practices

What could *possibly* go wrong?



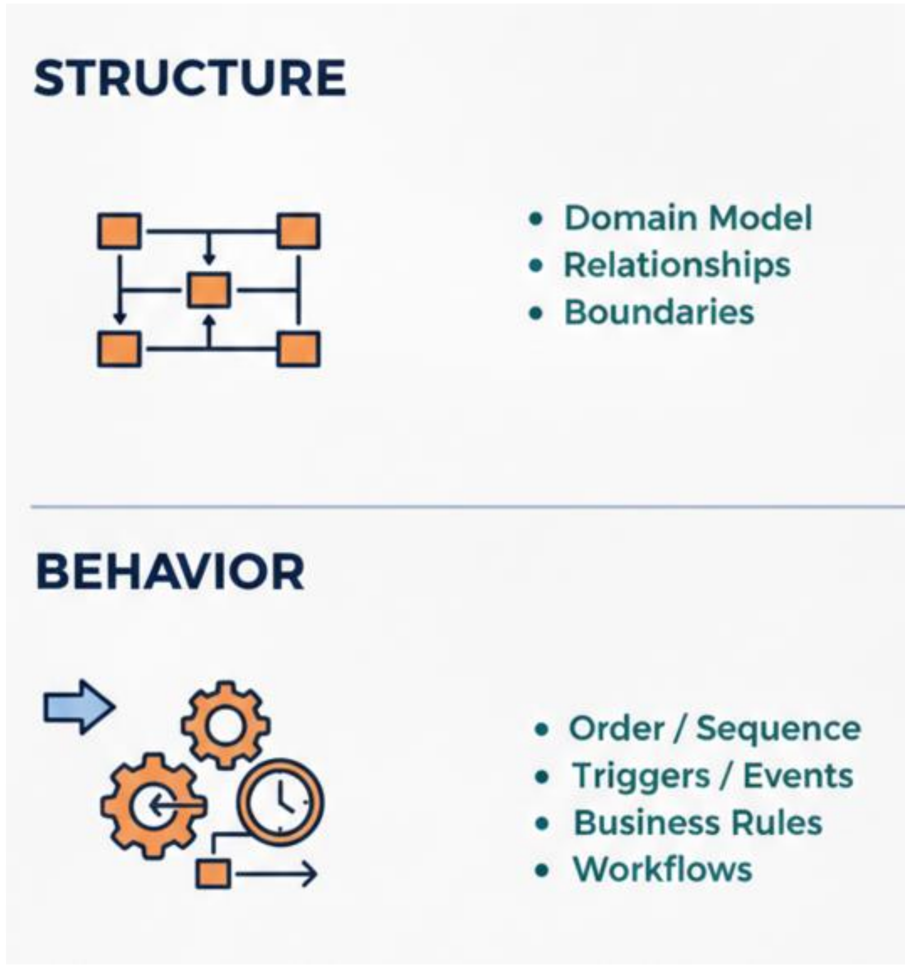
Wie kann ich Ihnen heute Abend helfen?

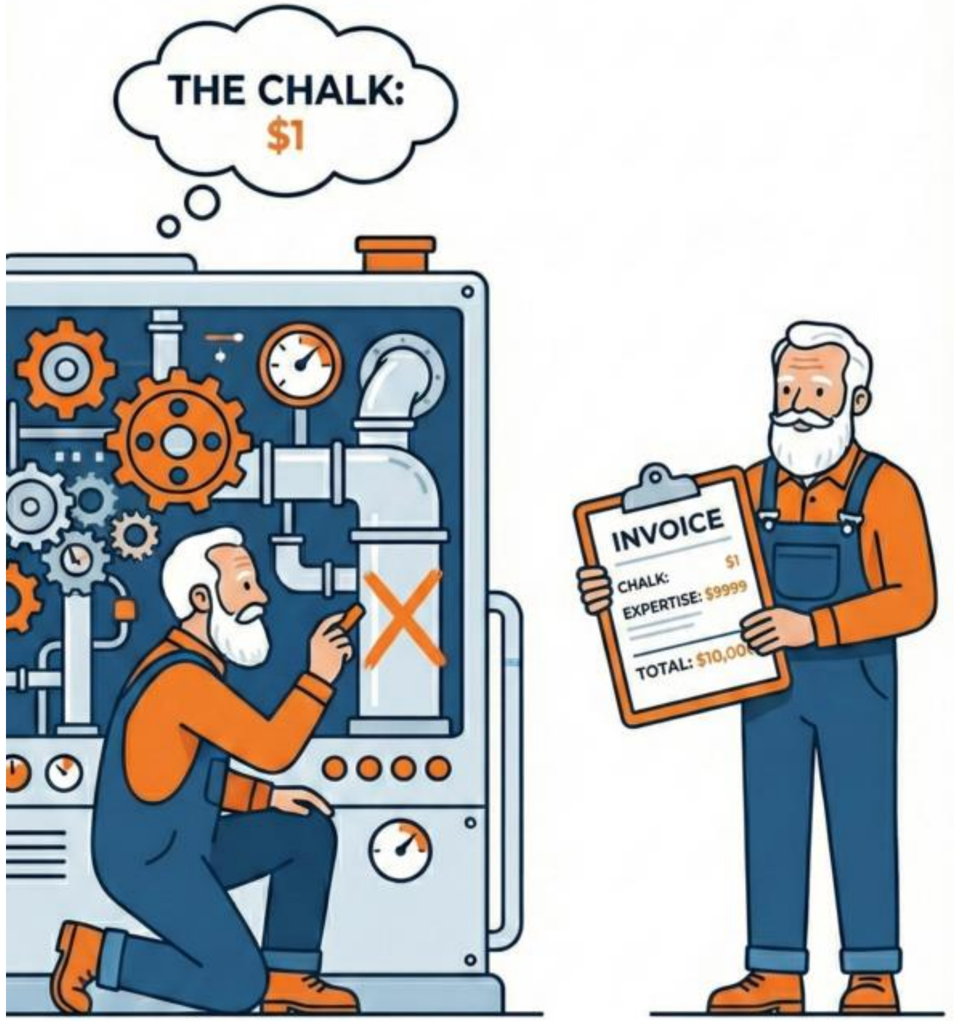
Create the rendering engine in C++, please make no mistakes, thx.

What should I control?
Who much can I delegate?

Keep control:

- Delegate implementation — not decisions
- **Structure:** Classes, Model, Schema, DTO
- **Behavior:** triggers, business rules, workflows
- Claude.md!




THE CHALK: $1
INVOICE
CHALK: $1
EXPERTISE: $9999
TOTAL: $10,00

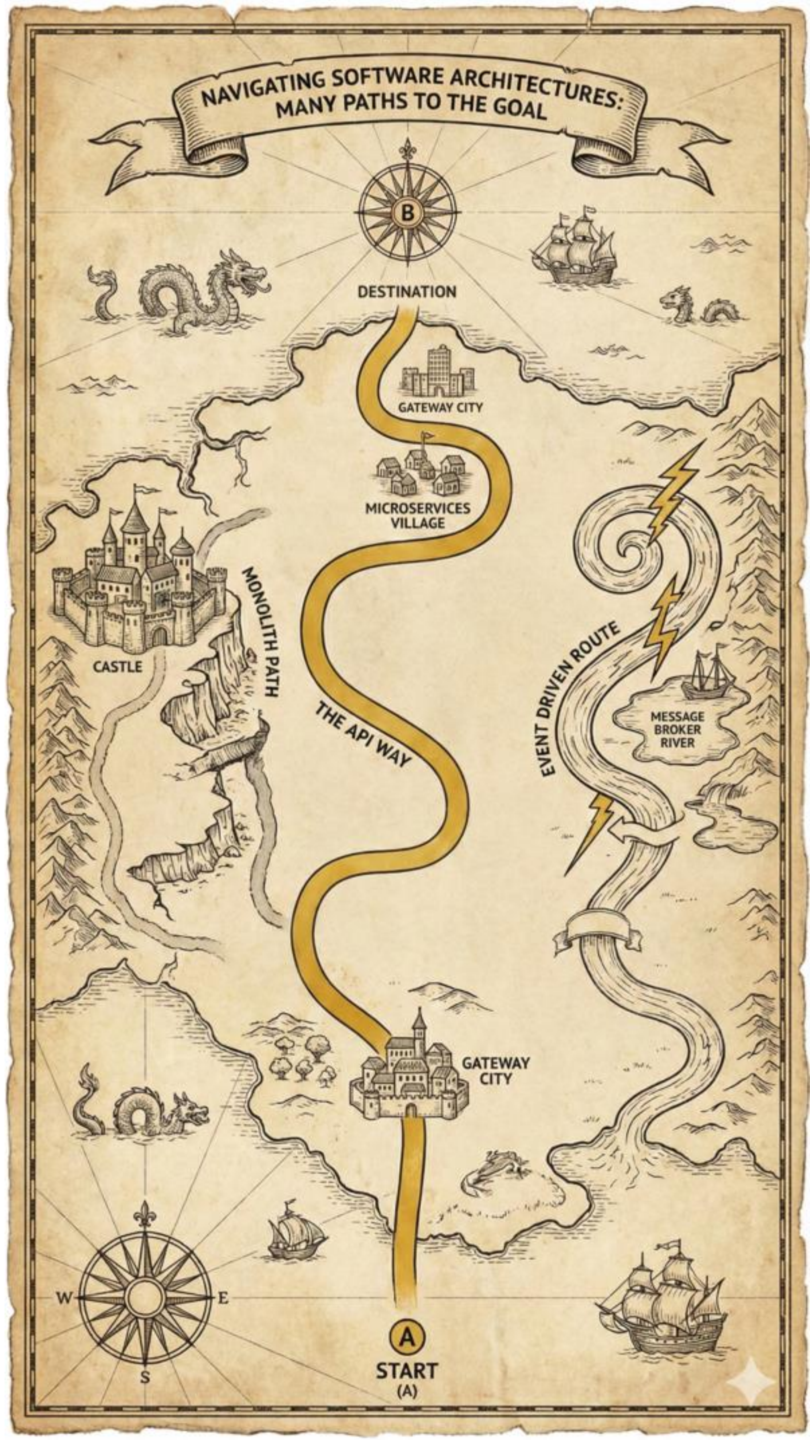
THE CHALK: $1
INVOICE
CHALK: $1
EXPERTISE: $9999
TOTAL: $10,00

THE CHALK: $1
Sequence
Class
- class
- class
+ create()
+ snring()
+ genRulls()
+ sequences()
+ sptish()
+ system()
Component
...
...
Class
ClassA
ClassB
system()
Client
ergnt()
Component
RECHNUNG
KREIDE: $1
EXPERTISE: $9999
SUMME: $10,00

Production errors are deceptive

(Don’t become your manager)

- Symptom ≠ root cause
- Common traps: **race conditions**, **upstream error**, **stale data**, **wrong exceptions**
- Stacked errors give false signals
- You must drive: **diagnosis → hypothesis → test → verify**



Developer as Foreman

- Master of its craft
- Delegates execution, **reviews and guides**
- Keeps **system overview + history**, not every detail
- Bridge between **production and stakeholders**
- Key skills: **judgement**, **systems thinking**, **domain insight**

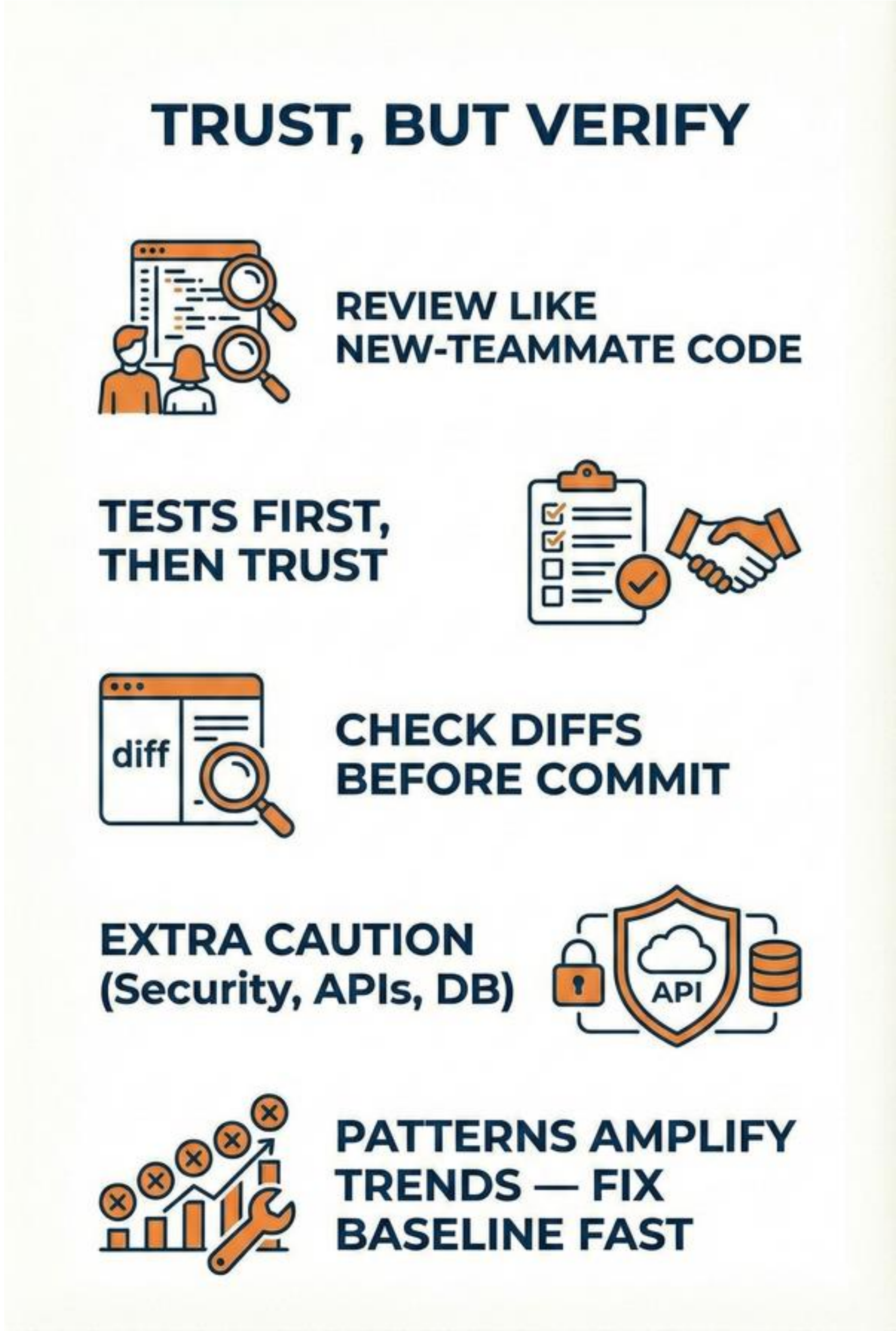
The Art of Briefing

- Give context: project, architecture, conventions
- Define the goal, not the path
- Define constraints early (e.g., no deps, Java 17)
- Work in small, reviewable stories
- Prefer questions over guesses (Plan Mode)

EFFECTIVE AI AGENT COLLABORATION
GIVE CONTEXT
(Goals, Arch, Conventions)
STATE THE TARGET, NOT THE ROUTE
No new deps
Java 17
NAME CONSTRAINTS UPFRONT
WORK IN SMALL, REVIEWABLE STORIES
?
LET THE AGENT ASK WHEN UNSURE
...

Trust, but Verify

- Review like junior code
- Tests first, then trust
- Check diffs before commit
- Extra caution: security, APIs, DB changes
- Patterns amplify trends—fix baseline fast
- Vibe-Coding is expensive!!


TRUST, BUT VERIFY
REVIEW LIKE NEW-TEAMMATE CODE
TESTS FIRST, THEN TRUST
diff
CHECK DIFFS BEFORE COMMIT
EXTRA CAUTION (Security, APIs, DB)
API
PATTERNS AMPLIFY TRENDS — FIX BASELINE FAST

„You push it, you own it.“

Developer Responsibility for AI Output

Developer Skills

- **Communication:** brief precisely, ask good questions, share context
- **Decision-making**
- **Judgment:** is it good, safe, and aligned with the architecture?
- **deep domain expertise**

NEW SKILLS:
THE DEVELOPER AS PRINCIPAL

COMMUNICATION
Brief precisely
Ask good questions
Share context

DECISION-MAKING
Agent offers options
You choose & commit

JUDGMENT
Is it good?
Is it safe?
Aligned with architecture?

SENIORITY

GOOD DOMAIN KNOWLEDGE

Developer Skills

- **Orientation**
- **Filtering**
- **Quality mindset:** more ≠ better
- **Balance:** let go enough for speed, control enough for quality

KEEPING THE BIG PICTURE

ORIENTATION
Where are we?
What's the goal?
What's missing?

FILTERING
Agent outputs a lot — what matters vs. noise?

QUALITY MINDSET
More code ≠ better code

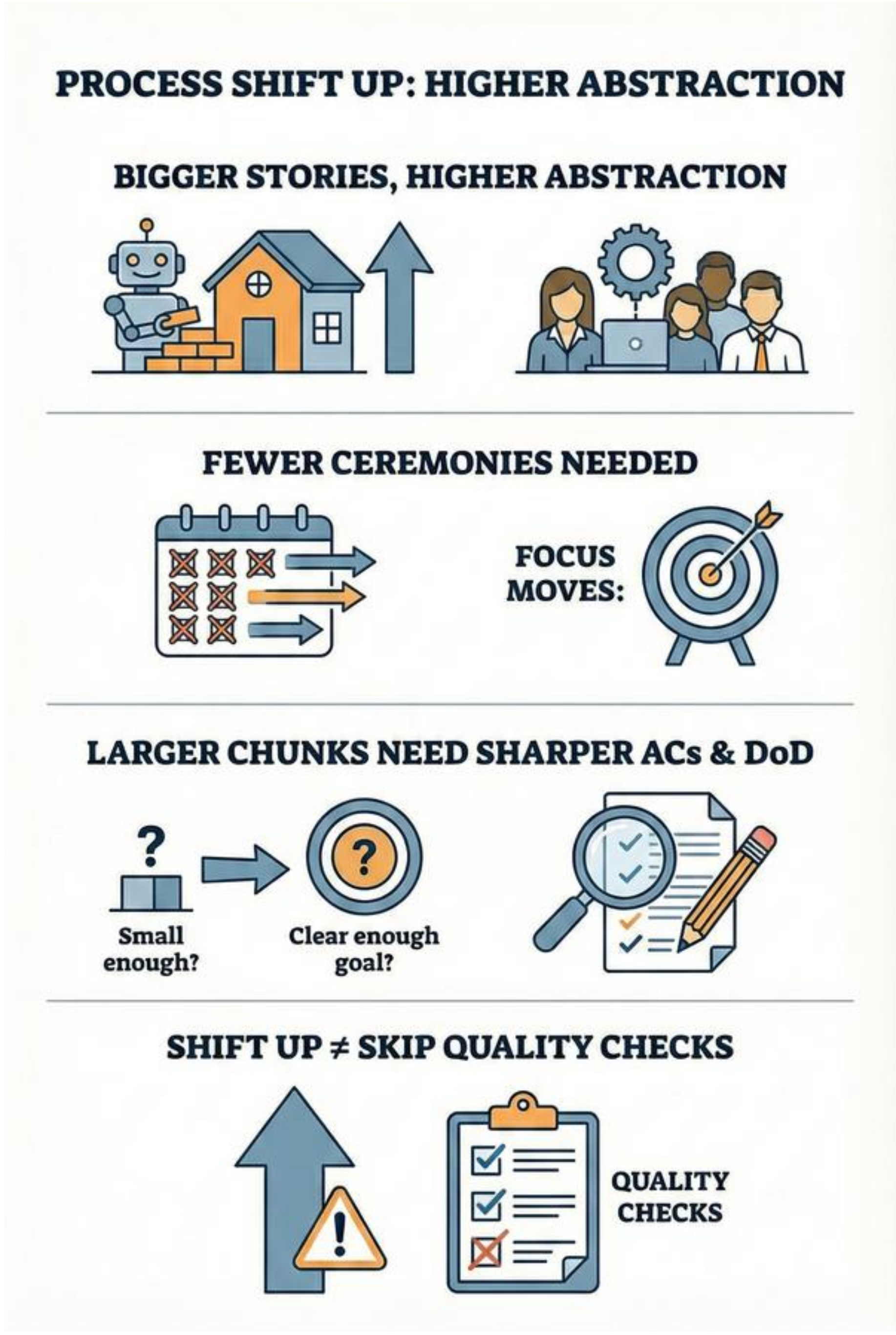
BALANCE
Let go enough for speed, control enough for quality

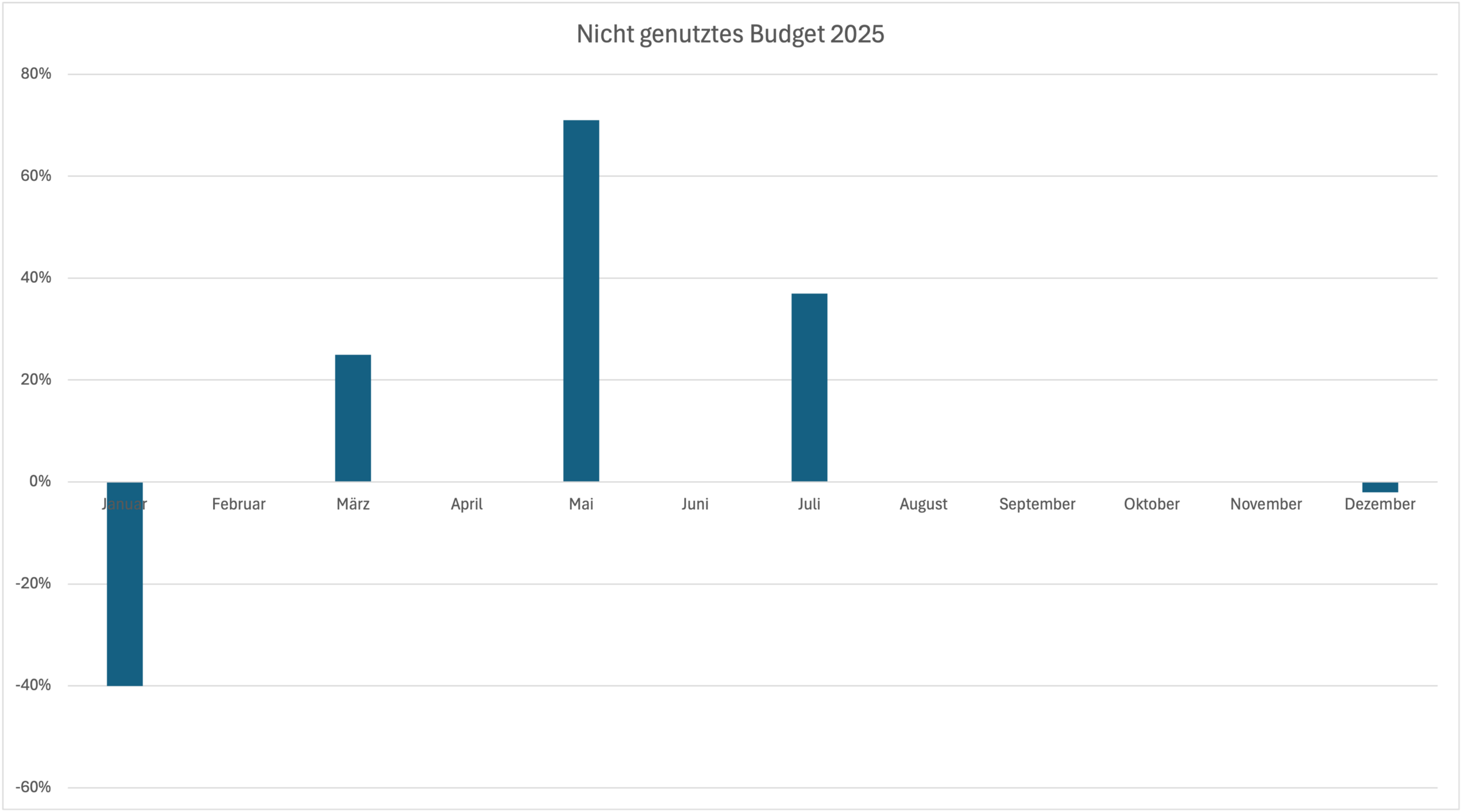
SENIORITY

GOOD DOMAIN KNOWLEDGE

Process shift up

- Bigger stories; agents work at higher abstraction
- Fewer ceremonies needed
- Focus moves: “small enough?” → “clear enough goal?”
- Larger chunks need sharper ACs & DoD
- Shift up ≠ skip up, quality checks needed!


PROCESS SHIFT UP: HIGHER ABSTRACTION
BIGGER STORIES, HIGHER ABSTRACTION
FEWER CEREMONIES NEEDED
FOCUS MOVES:
LARGER CHUNKS NEED SHARPER ACs & DoD
Small enough?
Clear enough goal?
SHIFT UP ≠ SKIP QUALITY CHECKS
QUALITY CHECKS


Nicht genutztes Budget 2025
80%
60%
40%
20%
0%
-20%
-40%
-60%
Januar
Februar
März
April
Mai
Juni
Juli
August
September
Oktober
November
Dezember

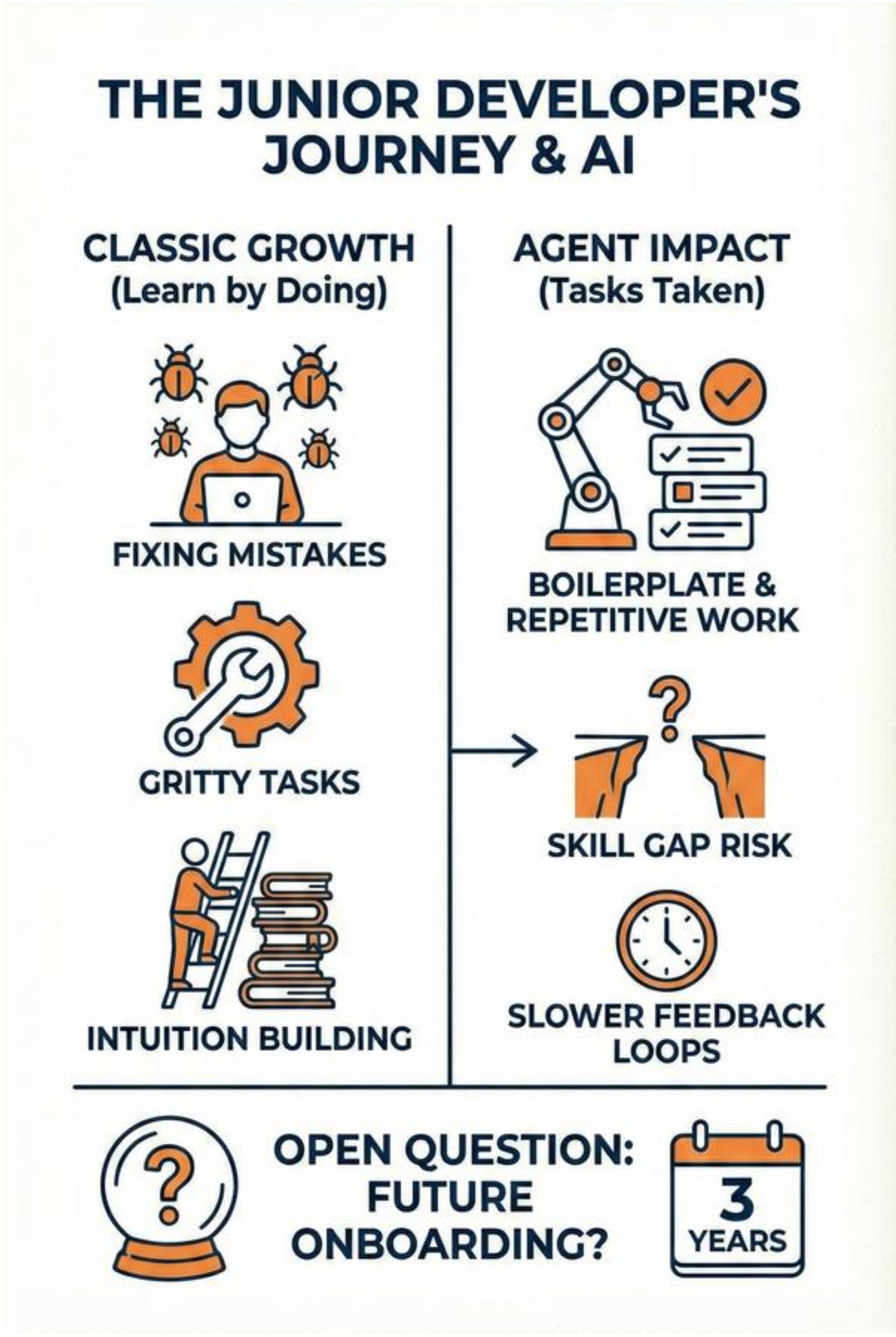
Challenges

Senior Challenges

- Pairing with machines instead of people
- Fewer knowledge exchanges
- Risk: teams drift into solo operators
- Countermove: keep human pairing intentional

Junior Challenges

- Classic growth: learn by doing, fixing mistakes, gritty tasks
- Agents take over many “junior” tasks
- Risk: skill gap and missing intuition
- Slower real-world feedback loops


THE JUNIOR DEVELOPER'S JOURNEY & AI
CLASSIC GROWTH (Learn by Doing)
AGENT IMPACT (Tasks Taken)
FIXING MISTAKES
BOILERPLATE & REPETITIVE WORK
GRITTY TASKS
SKILL GAP RISK
INTUITION BUILDING
SLOWER FEEDBACK LOOPS
OPEN QUESTION: FUTURE ONBOARDING?
3 YEARS

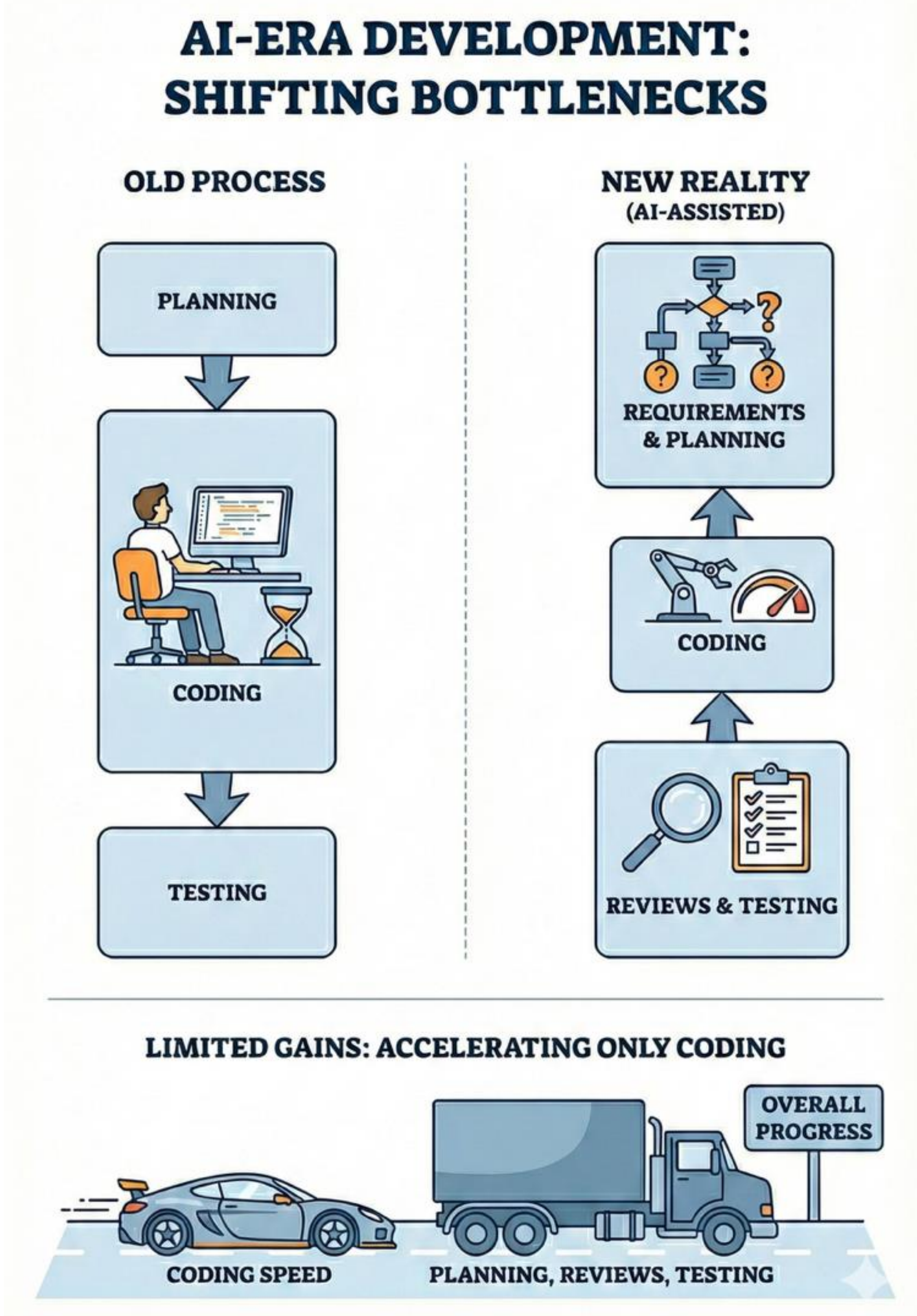
Team Challenges

- **Varying Performance**
- **Lone Developer**
- **Bus Factor**
- **Rejection of AI**: Sceptics vs Early Adopters

AI ADOPTION:
TEAM CHALLENGES
PERFORMANCE GAP
WORKFLOW SILOS
KNOWLEDGE RISK
ADOPTION TENSION
SOLUTION: OPEN DIALOG

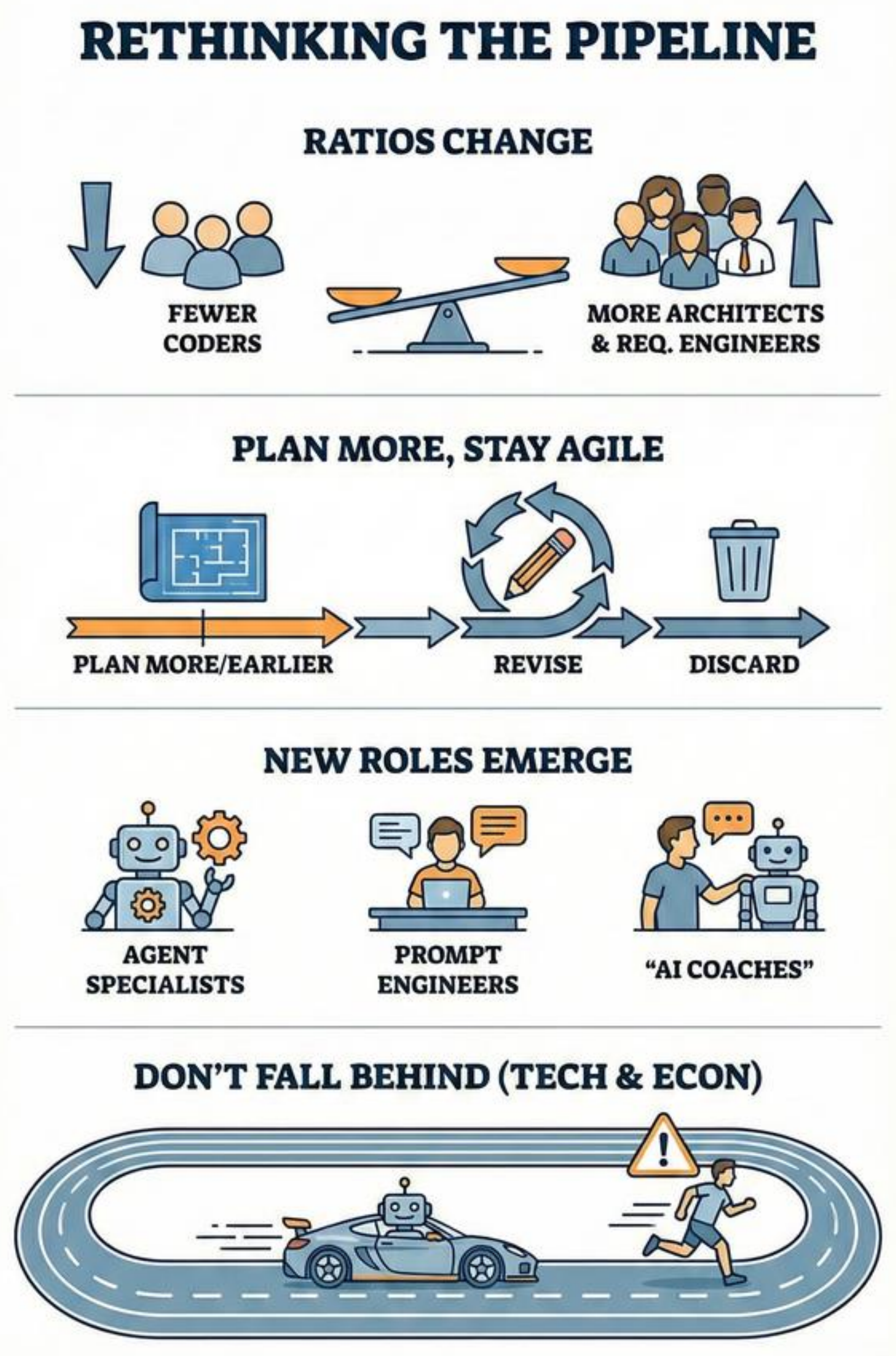
Orga: New bottlenecks

- Coding speeds up; reqs/review/tests lag
- Bottleneck moves to reqs & planning
- APIs/dependencies matter more
- Faster coding alone = small gains


AI-ERA DEVELOPMENT: SHIFTING BOTTLENECKS
OLD PROCESS
PLANNING
CODING
TESTING
NEW REALITY (AI-ASSISTED)
REQUIREMENTS & PLANNING
CODING
REVIEWS & TESTING
LIMITED GAINS: ACCELERATING ONLY CODING
OVERALL PROGRESS
CODING SPEED
PLANNING, REVIEWS, TESTING

Orga: new balance

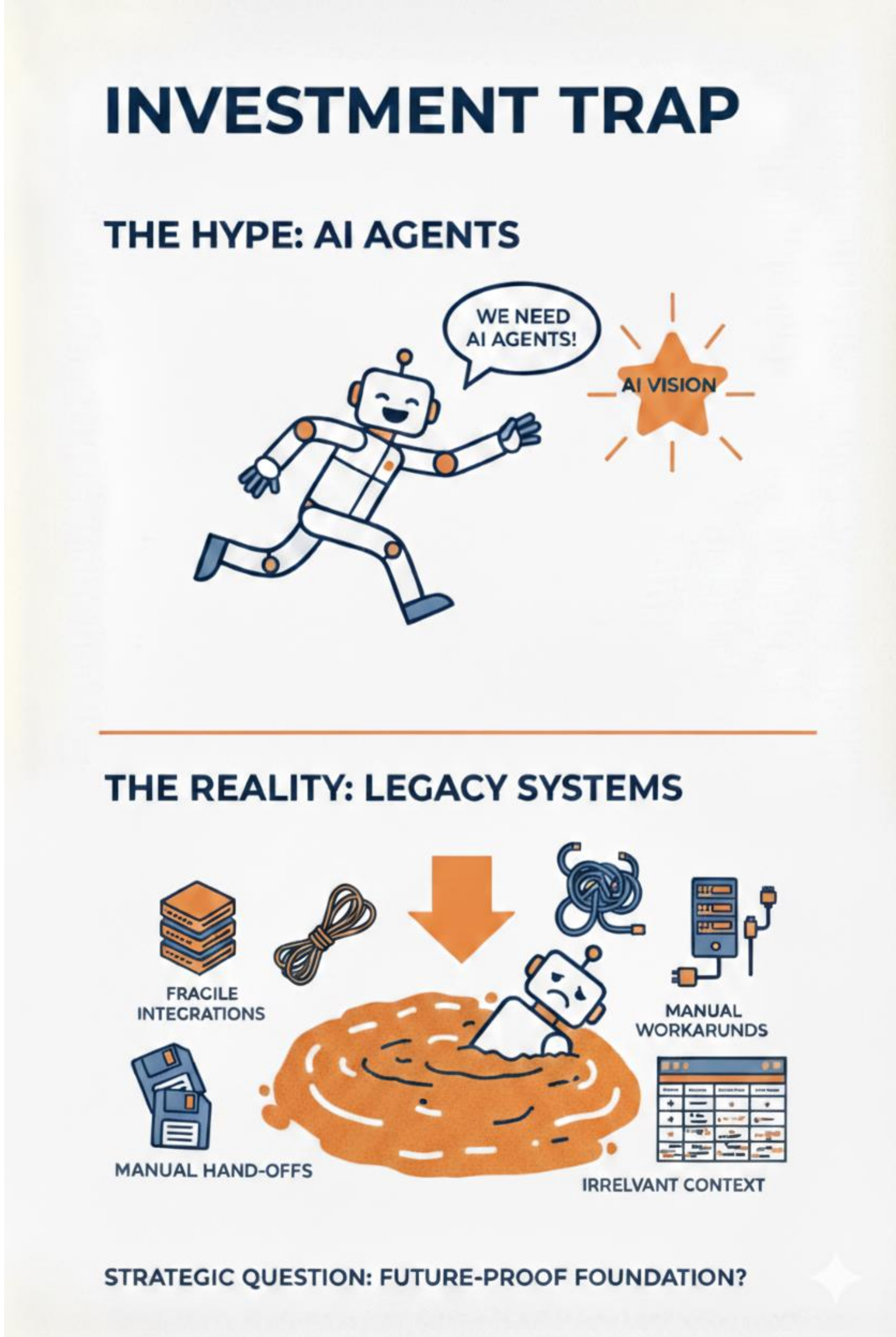
- Pipeline needs a rethink
- Fewer coders, more architects/REs
- Plan earlier, longer, better, stay agile
- New roles: agent/prompt/AI coaches
- Stay tech & business-competitive



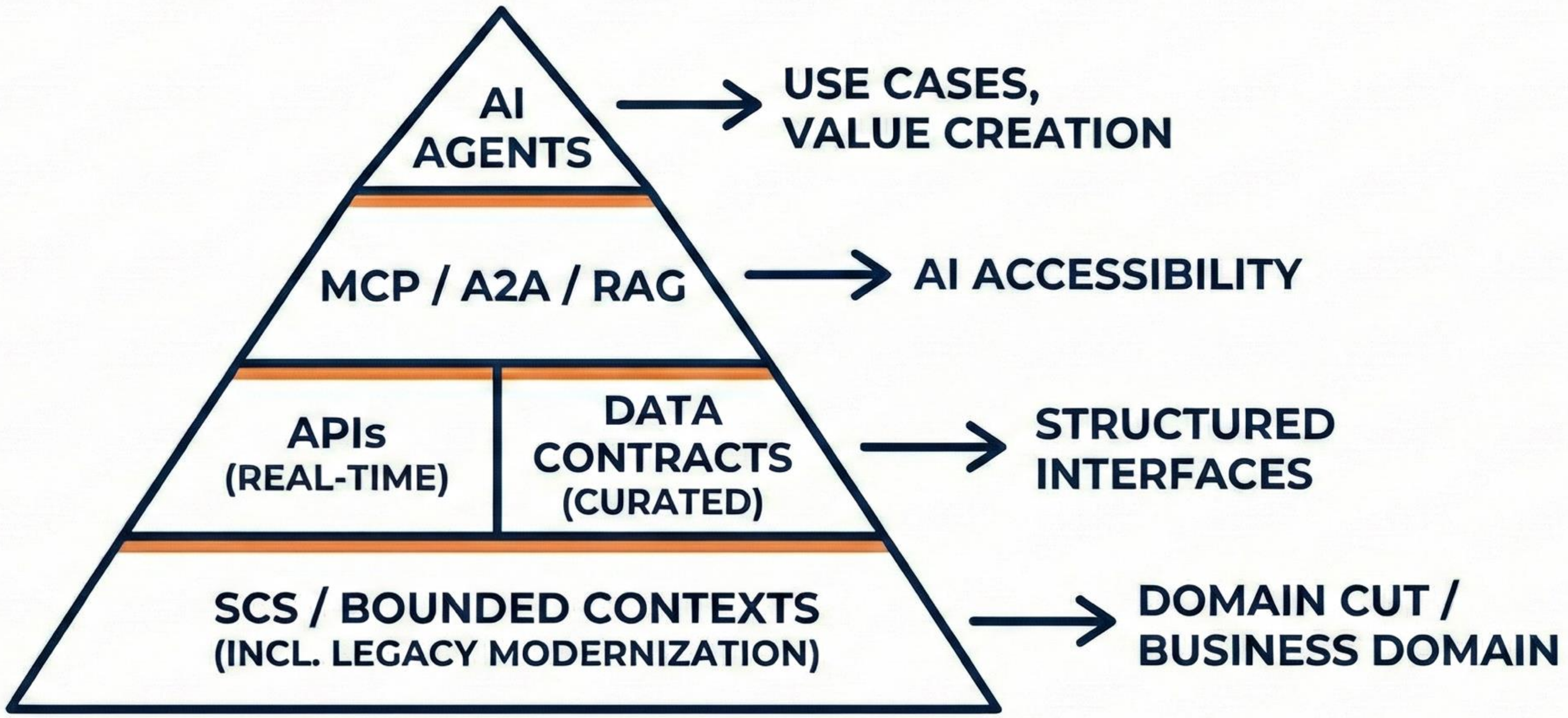
AI Strategy

Where to invest

- “We need agents or we’ll fall behind”
- Agents need **data + system**
- Bad interfaces → workarounds + fragile integrations
- Bad Data → **too much context, too little signal**
- **More Capa, Orga needs time, not ready for agents**
- Build what **still pays off later**



AI PYRAMID


AI AGENTS
USE CASES, VALUE CREATION
MCP / A2A / RAG
AI ACCESSIBILITY
APIs (REAL-TIME)
DATA CONTRACTS (CURATED)
STRUCTURED INTERFACES
SCS / BOUNDED CONTEXTS (INCL. LEGACY MODERNIZATION)
DOMAIN CUT / BUSINESS DOMAIN

- EACH LAYER IS A PREREQUISITE FOR THE NEXT
- THE DEEPER THE INVESTMENT, THE BROADER THE BENEFIT

Wrap-up

A *10x* faster engine

demands chassis, brakes, etc.

to scale accordingly

Have a huge backlog *ready* for development

Agents are becoming
***better*, more *autonomous*, and more *specialized*.**

Multi-agent systems will take on longer, end-to-end tasks.

The developer’s role shifts toward *architecture* and *product* direction.

Thank you.
Questions?

INNOQ
www.innoq.com

Ole Wendland
ole.wendland@innoq.com

innoQ Schweiz GmbH

Schutzengelstr. 57
6340 Baar
+41 41 743 0116

Hardturmstr. 253
8005 Zürich