AI Native Software Development

From AI Experiments to AI Native Development

Introductions

Christian

Husband

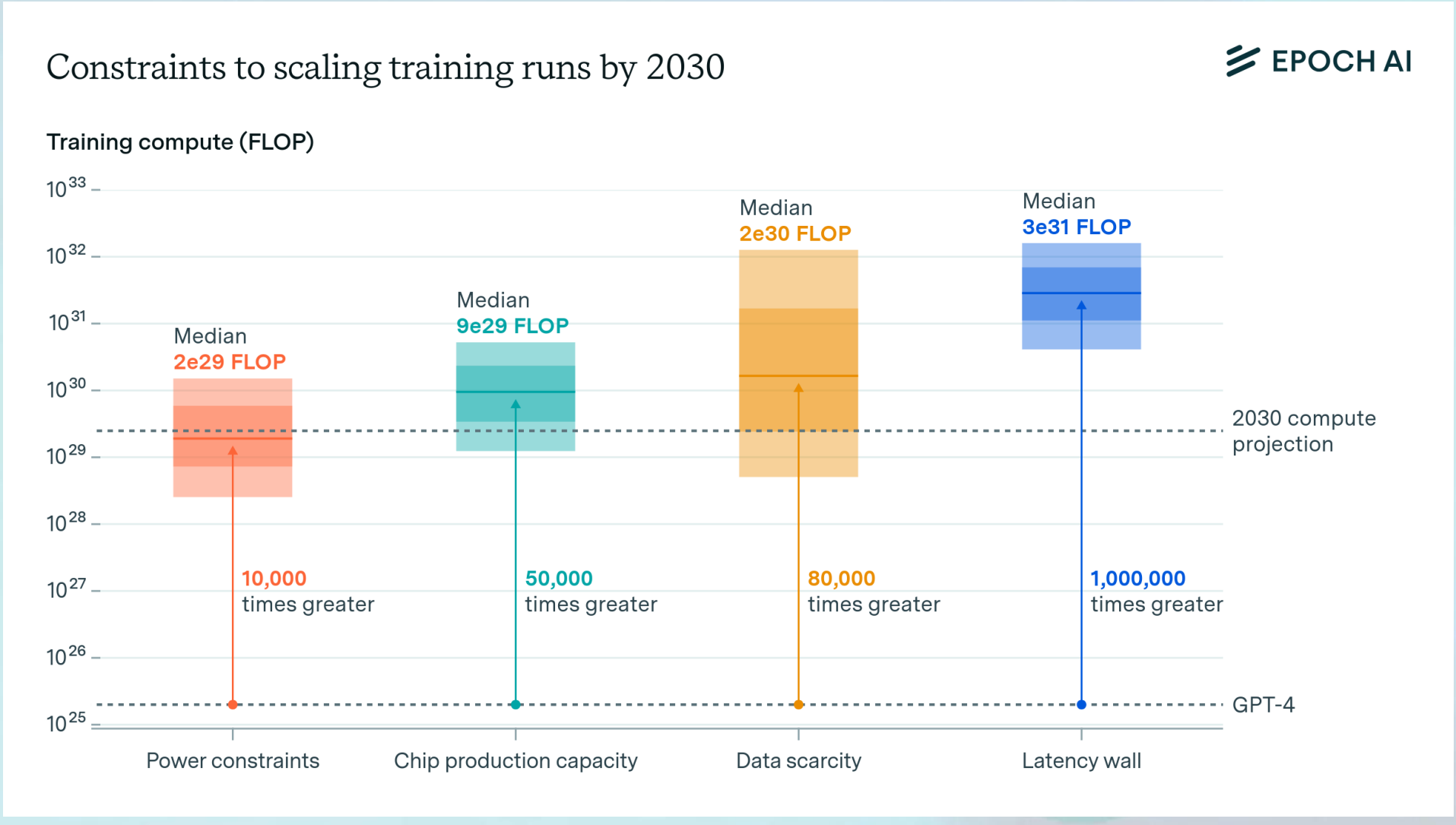
Father

Passionate Geek

Life-long learner

State of the Industry

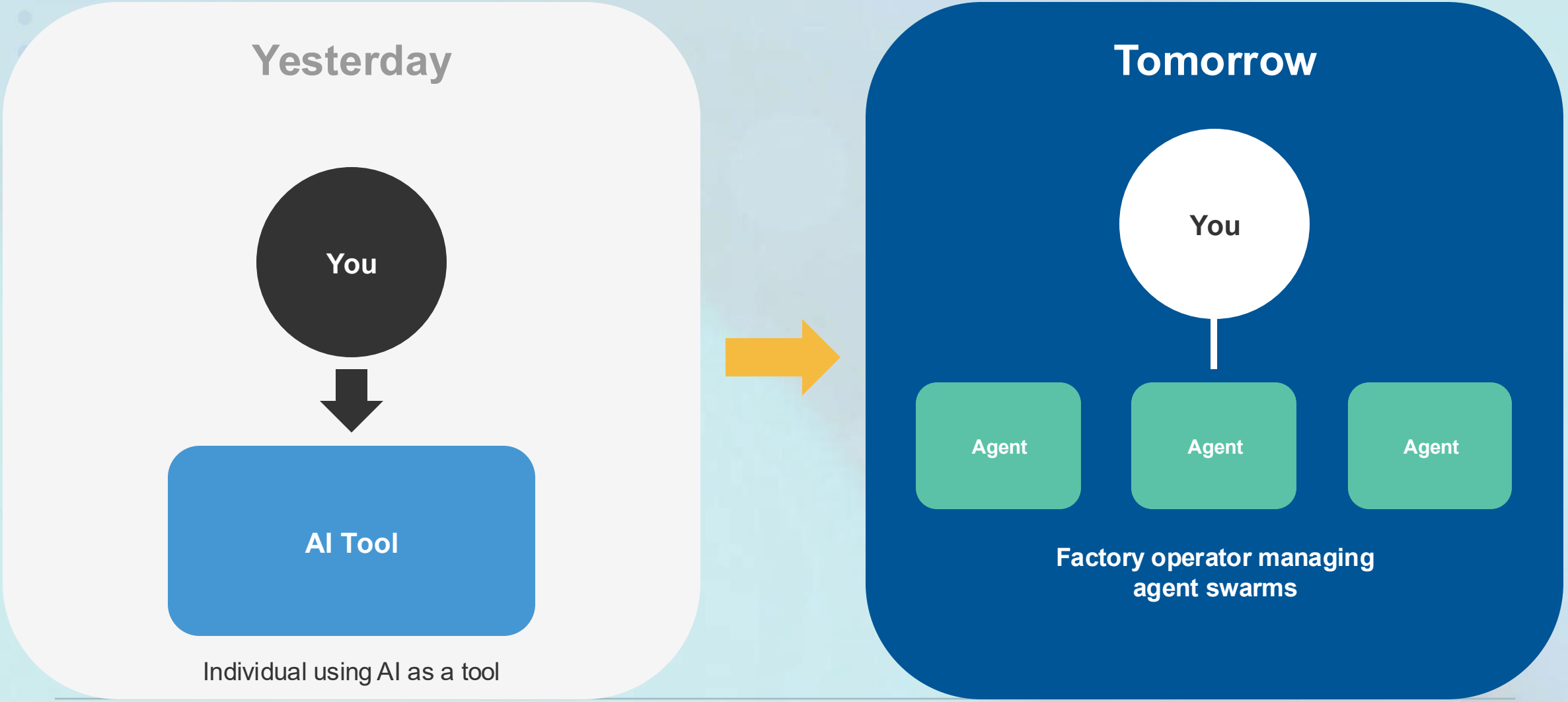
Understanding Demand



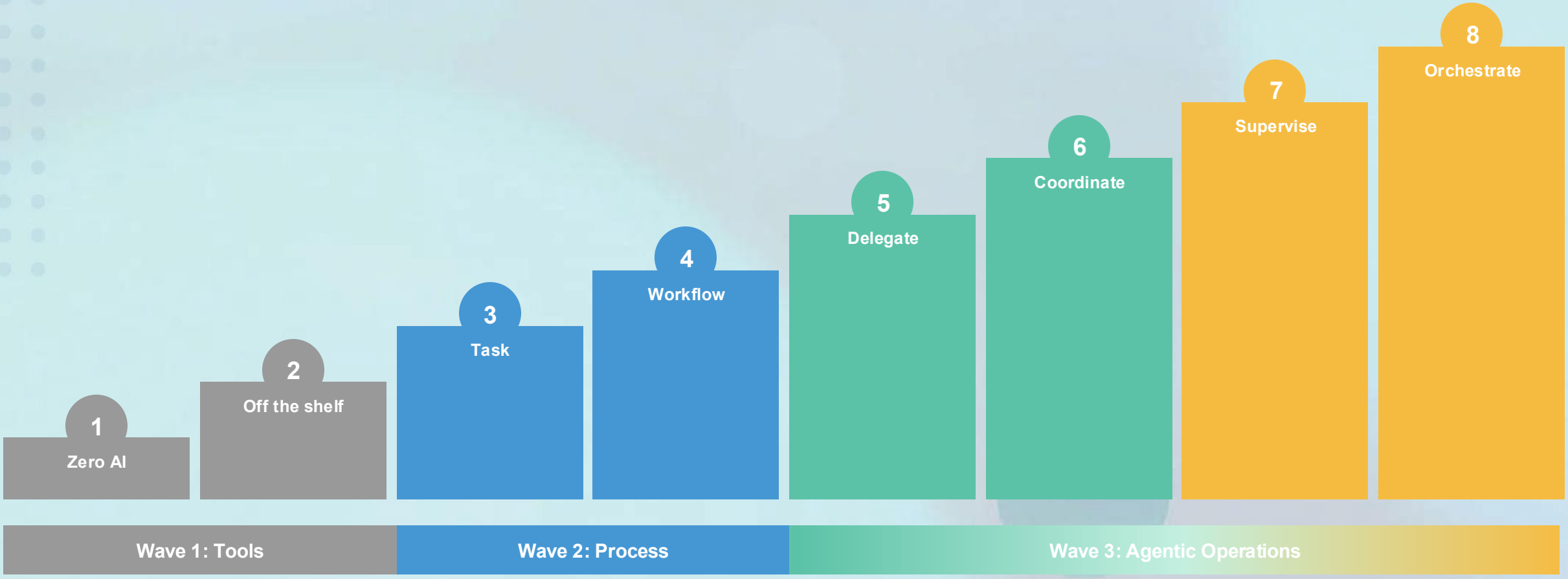
improving

What does it look like going forward?

The Fundamental Shift in How We Work



The Trust Evolution: 8 Stages of AI Maturity



As trust increases, humans shift from doing to directing

Q1 : AI Usage Frequency

Metric	Improving	Gallup (All Workers)	Gallup (Tech Workers)
Daily Use	66.9%	12%	31%
Frequent (weekly+)	93.9%	26%	57%
Any use (yearly)	99.3%	46%	77%
Never	0.7%	49%	23%

Source: Gallup Workforce Survey Q3/Q4 2025
URL: https://www.gallup.com/workplace/699689/ai-use-at-work-rises.aspx

Improving is dramatically ahead — 99.3% of respondents use AI at least yearly vs. only 46% nationally (77% for tech workers). Daily usage at Improving (66.9%) is **more than 5.5x the national average** (12%) and **more than 2x** the tech industry benchmark (31%).

Q4 : What Types of AI Tools Do You Use?

Response	Improving	Gallup	Delta
Chatbots/virtual assistants	**91.1%**	61%	**+30.1**
AI coding assistants	**62.2%**	14%	**+48.2**
Image/video/audio generators	20.7%	8%	+12.7
AI writing/editing tools	17.3%	36%	-18.7
Data science/analytics tools	7.5%	10%	-2.5

Source: Gallup Workforce Survey Q3/Q4 2025
URL: https://www.gallup.com/workplace/699689/ai-use-at-work-rises.aspx

Coding assistant usage is over 4x the national average (62.2% vs 14%). This reflects Improving's technical workforce.
Writing tools are below national average (17.3% vs 36%) — possibly because technical staff favor coding assistants over general writing tools.

THE CRITICAL MOMENT

The Autonomy Inflection Point

Before: Permission Mode

- Human approves every action
- AI waits for permission
- Slow but controlled
- Trust is low
- Human does verification

After: Autonomous Mode

- AI acts autonomously
- Permissions turned off
- Fast but risky
- Trust is high
- Governance becomes critical

When you turn off permissions, governance infrastructure becomes essential

The Intelligence Operating System

Infrastructure for the autonomous era:
Governance when permissions are off

The Three Layers of Intelligence OS

LAYER 1

Hybrid Intelligence

Human expertise + AI swarms working together
From tool user to factory operator

LAYER 2

Governance Engine

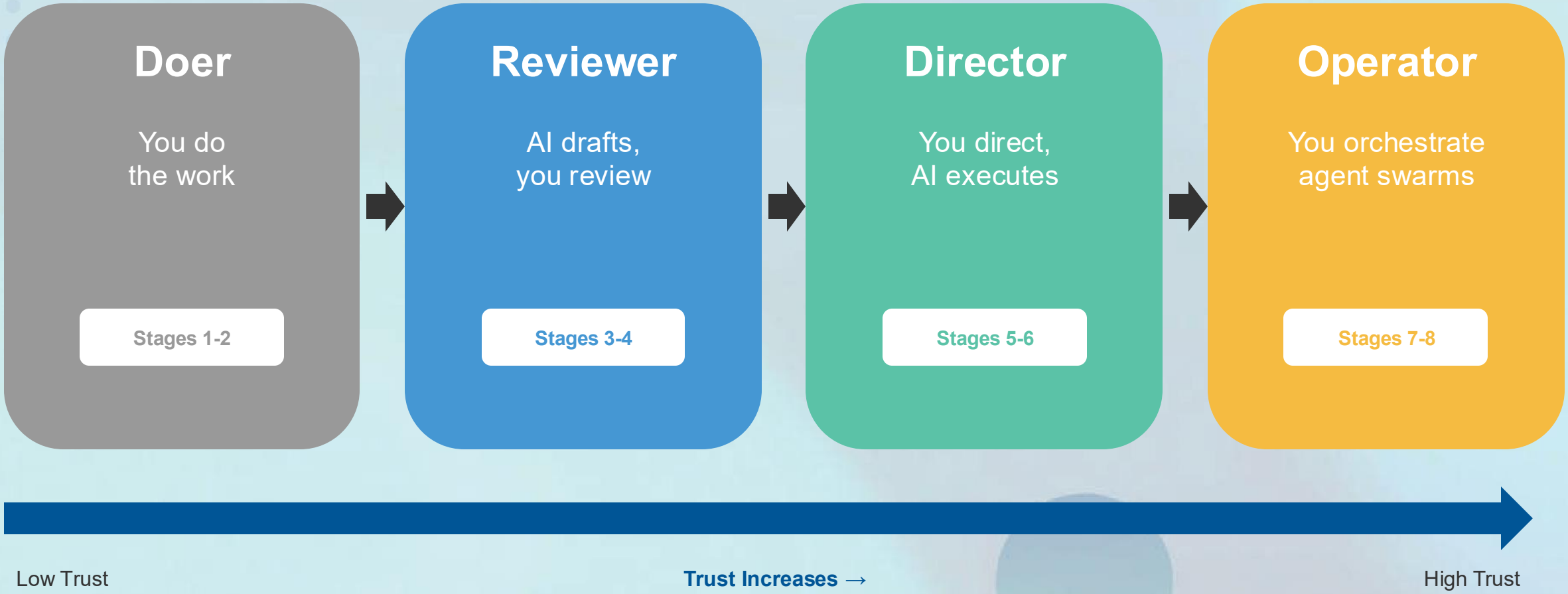
The safety net for autonomous mode
Boundaries, trust, accountability

LAYER 3

Operating Model

Orchestration infrastructure
How you manage 10+ agents at scale

The Role Evolution: Contributor → Operator



Your value shifts from doing work to directing outcomes

RESEARCH

The Skill Leveling Effect (before stage 4)

Junior Skills Users (Learning)

+35% Productivity Gain

Expert Users

+3%

Strategic Insight

AI democratizes capability across stages.

Those at early career stages gain the most.

Stage 7-8 operators add value through orchestration, not individual output.

Source: Fortune 500 Enterprise AI Analysis (2024)

Stage 2 Permissions — AI-Assisted Development

Stage 2 → Stage 3

The Bridge: Identifying Tasks in a Workflow

Stage 2 → Stage 3: The Key Shift

Stop thinking about individual prompts. Start mapping your entire workflow — every step, every input, every output. Then identify which steps AI can own autonomously.

1. Document the Workflow

Map every step in your development process: requirements → design → code → test → review → deploy

2. Define Constraints

What must AI never do? Where must humans stay in control? What errors are predictable?

3. Build Review Checklists

Create systematic review patterns that catch known failure modes before output is used

Exercise: Workflow Task Identification

Map a real development workflow and identify AI-ready tasks

Workflow Step	Who Does It Today?	AI Ready?	Constraints / Checkpoints	What Could Go Wrong?
Parse requirements into user stories	Developer	Yes	Schema validation	Missing edge cases
Generate boilerplate / scaffolding	Developer	Yes	Project conventions	Wrong framework patterns
Write unit tests from spec	Developer	Yes	Coverage thresholds	Tests that pass trivially
Implement business logic	Developer	Partial	Architecture review gate	Logic errors, wrong assumptions
Code review	Peer	Partial	Human approval required	Missing security concerns
Write deployment scripts	DevOps	Yes	Environment constraints	Wrong env configs

Autonomy Inflection Point: You need documented workflows and defined constraints before turning off step-by-step approval.

Demo: From Ad-Hoc Prompting to Task Agents

Stage 2 → Stage 3 in practice

1. **Start in Permission Mode**

   Show a typical Cursor/Copilot workflow — prompt, review, accept, repeat. Highlight the approval bottleneck.

2. **Map the Workflow**

   Take a real feature request and decompose it into discrete steps on a whiteboard/doc. Identify inputs and outputs per step.

3. **Create a Task Agent**

   Write a system prompt + constraints for one workflow step (e.g., 'generate unit tests from this spec'). Include what the agent must NOT do.

4. **Run Autonomously**

   Let the task agent run without step-by-step approval. Review the output, not the process. Show how constraints prevent common errors.

Learning: The shift from Stage 2 → 3 is about trusting documented, constrained agents to run without your line-by-line approval

Stage 3 → Stage 4

Building the Workflow Agent + Governance Infrastructure

RESEARCH

Governance: The Autonomy Safety Net

Autonomy Without Governance

Success Rate

Autonomy With Governance

Success Rate

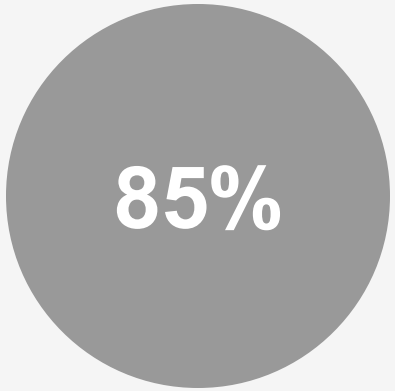
As trust increases and permissions decrease, governance becomes critical infrastructure

Source: Gartner AI Governance Research (2024)

RESEARCH

Human-in-the-Loop: Even in Autonomous Mode

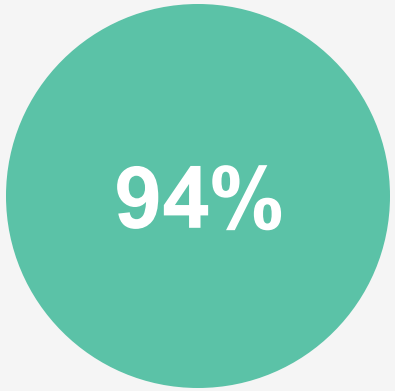
Full Manual



Stage 1-2

You do
everything

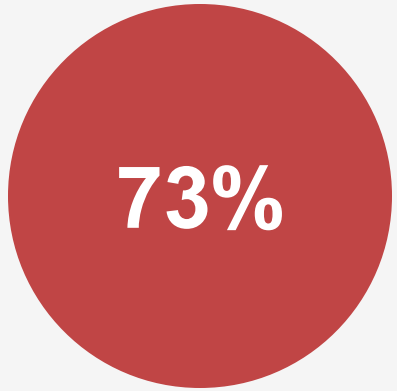
Human + AI



Stage 5-6

You direct,
AI executes

Full Autonomy



Stage 8 risk

AI alone,
no oversight

Even with 10+ agents, the operator's judgment remains the quality gate

Source: MIT Sloan Management Review (2024)

Stage 3 → Stage 4: The Key Shift

Once enough task agents are running reliably at Stage 3, you can compose them into a workflow-level 'Workflow agent' that orchestrates the specialist agents and includes governance, auditing, and protection systems.

Workflow agent Design

- Which agent manages the overall workflow?
- What specialist agents does it coordinate?
- What is the handoff protocol between agents?

Governance Infrastructure

- Capturing artifacts for every agent action
- Boundary enforcement — what agents cannot access
- Error handling and rollback procedures

Human Collaboration Map

- Where does the team approve?
- Where does the team clarify?
- Where does the team redirect?

Readiness Tip: You're ready for Stage 4 when you can draw the workflow of stage 3 agents on a whiteboard.

Demo: Composing Task Agents into a Workflow Agent

Stage 3 → Stage 4 in practice

1. **Show Individual Task Agents**

   Run 3-4 task agents from Stage 3 independently. Show how they each produce good output but don't coordinate.

2. **Build the Workflow Agent**

   Create an orchestration layer that calls task agents in sequence, passes outputs as inputs, and manages state across steps.

3. **Add Governance Hooks**

   Insert audit logging, boundary checks, and error handlers. Show what happens when an agent exceeds its boundaries.

4. **Map Human Touchpoints**

   Configure the Workflow agent to pause at approval gates, request clarification at ambiguity points, and report status to humans.

Learning: Governance isn't overhead — it's the infrastructure that makes autonomous agents safe to scale.

Stage 4 Workflow agent — Governance & Workflow Orchestration

Your Role: Director | AI manages the workflow, humans collaborate at defined points

Governance Infrastructure

Audit Trails

Every agent action creates artifacts as needed timestamps, inputs, outputs, and decisions made

Error Handling

Automatic rollback, human escalation, and graceful degradation when agents fail

Human Approval Points

Pre-defined moments where the workflow pauses for human review, clarification, or redirection

Boundary Enforcement

Agents cannot access data, systems, or make decisions outside their defined scope

Quality Gates

Automated checks between agent handoffs — output validation before the next agent proceeds

Access Control

Fine-grained permissions for what each agent can read, write, execute, and deploy

improving

Stage 4 → Stage 5

Beyond the IDE — Status & Output Over Watching the Stream

Stage 4 → Stage 5: The Key Shift

Stop watching agents work. Start reviewing what they produce. The move to non-IDE agents means agents run as background processes — you interact through status dashboards and output artifacts, not by watching code stream across your screen.

Before: IDE-Bound (Stage 4)

- Watch agent code in real-time in Cursor/VS Code
- One agent visible at a time in your editor
- Attention locked to the active stream

After: Non-IDE Agents (Stage 5)

- Agents run as CLI tools / background processes
- You review status updates and output artifacts
- Attention freed to direct, not observe

Demo: Non-IDE Agents — Headless Agents

Stage 4 → Stage 5 in practice

1. **Show IDE-Bound Agent**

   Run a Cursor agent completing a feature. Notice: your attention is locked to watching code stream. You can't do anything else.

2. **Launch Command Line Tools**

   Same task, but as a terminal command. Agent runs in background. You see status, not stream.

3. **Review Output, Not Process**

   Agent finishes. You review the PR/diff — the deliverable. You never watched the code being written. Status: complete, tests: passing.

4. **The Mental Shift**

   You are now a manager reviewing deliverables, not a pair programmer watching keystrokes. Your attention is freed.

Learning: When you stop watching agents work, you gain the capacity to direct multiple agents simultaneously.

Stage 5 Non-IDE Agents — Status-Driven Development

Your Role: Status Monitor | Agents report what they've done, not how they're doing it

CLI Tools

Terminal-based agent that runs headless, produces PRs and status reports

Background task execution

GitHub Actions Agents

CI/CD triggered agents that run on push, PR, or schedule events

Event-driven automation

Codex / Devin

Sandboxed development agents that work in isolated environments

Isolated agent execution

Systems Advice: Build status interfaces first. If you can't monitor agent health, you can't scale agent count.

Stage 5 → Stage 6

Parallel Streams — Multiple Agents, Isolated Workstreams

Stage 5 → Stage 6: The Key Shift

Now that agents work in the background and you monitor by status, the next step is running multiple agents simultaneously on isolated workstreams. Each agent gets its own branch, its own context, and its own deliverable.

Isolation

Each agent works in its own branch/sandbox. No cross-contamination of context.

Status Aggregation

A dashboard or feed that shows all agent progress in one view.

Merge Strategy

How do parallel outputs get combined? Conflict resolution protocol.

Demo: Running Parallel Agent Workstreams

Stage 5 → Stage 6 in practice

1. **Launch Multiple Agents**

   Start 3 agents simultaneously: Agent A (feature implementation), Agent B (test suite), Agent C (documentation). Each on its own branch.

2. **Monitor the Dashboard**

   Show a status view: Agent A — in progress (55%), Agent B — completing tests (80%), Agent C — docs generated (100%). No need to watch code.

3. **Handle Convergence**

   Agents finish at different times. Review each PR independently. Show the merge strategy when outputs need to combine.

4. **Capture Learnings**

   What worked: isolation prevented conflicts. What didn't: Agent A needed context Agent B discovered. Solution: shared context protocol.

Learning: Parallel agents multiply output but require isolation and a merge strategy. Context sharing between agents is the hardest problem.

Stage 6 Parallel Agents — Isolated Multi-Stream Development

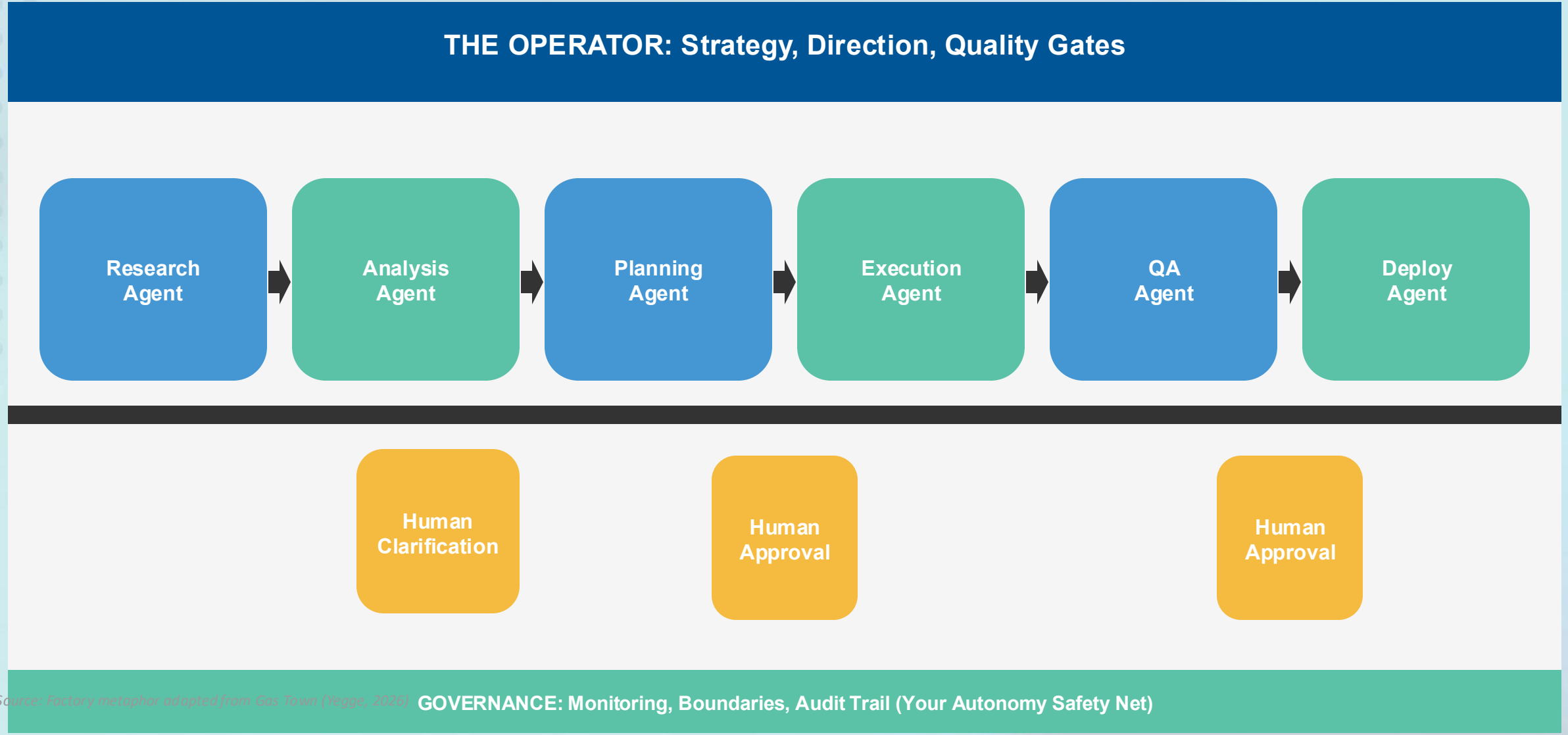
Your Role: Coordinator | Multiple agents work simultaneously, you manage the portfolio

Parallel Agent Architecture

Stream A	**Feature Agent** Implement new API endpoint	In Progress	65%
Stream B	**Test Agent** Write integration test suite	Completing	88%
Stream C	**Docs Agent** Generate API documentation	Complete	100%
Stream D	**Refactor Agent** Modernize legacy module	Queued	0%

Systems Advice: Start with 2-3 parallel agents. Scale only when your monitoring and merge strategy are proven.

The Agentic Factory: You as Operator



Source: Factory metaphor adopted from Gas Town (Yegge, 2026)

Talent: Smaller but higher number of teams enabling higher capacity and higher feature throughput in large international airline

From

Two pizza team of 8-10 people

Example shared functional roles

- SRE/ Devops
- TPM
- Business analyst
- Data analysts/ engineers

To

One pizza team of 3-5 people

x2 number of pods for more features

Example shared functional roles

- ML / dev ops
- Security Engineer
- Enterprise Architects
- QA engineer

Stages 7 + 8 Agent Orchestration — Agents Coordinating Agents

Your Role: Operator / Strategist | Meta-agents manage agent teams, you set direction

What Changes at This Level

• A meta-agent (orchestrator) assigns work to other agents

• The orchestrator decides which agents to spin up, what context they need, and how outputs combine
• Agents can request help from other agents autonomously

• The human defines the goal and constraints — the agent system figures out the execution plan
• Self-improving: the system learns from completed tasks and optimizes agent assignments

• This is the 'factory operator' mental model

Orchestration Architecture

Human: Sets goals, reviews milestones, adjusts strategy

Demo: Agent Orchestration in Action

Stages 7 + 8 — Agents coordinating agents

1. **Define the Goal**

   Give the orchestrator a high-level objective: 'Build a REST API for user management with auth, tests, docs, and deployment config.'

2. **Watch Delegation**

   The orchestrator breaks the goal into tasks, assigns them to specialist agents, and provides each with the context they need.

3. **Agent-to-Agent Communication**

   Show how the Code Agent requests schema info from the Planning Agent, and how the Test Agent auto-generates from the Code Agent's output.

4. **Human Checkpoint**

   The orchestrator pauses at the architecture review gate. Human approves or redirects. Agents resume with the updated direction.

Learning: At this stage, your job is defining 'what' and 'why' — the agent system handles 'how', 'when', and 'who'.

Key Learnings at Each Progression

What changes in your mindset, systems, and output as you progress

2→3

Trust requires documentation. You can't grant autonomy without defined constraints.

Workflow mapping + constraint definition

3→4

Governance is infrastructure, not overhead. Without it, agent systems are unsafe at scale.

Workflow agent + audit trails + approval gates

4→5

Stop watching, start reviewing. Attention is the bottleneck — free it by moving agents out of the IDE.

Non-IDE agents + status dashboards

5→6

Isolation enables parallelism. Without isolation, parallel agents create chaos, not velocity.

Branch isolation + merge strategy + monitoring

7+8

Define 'what' and 'why', let agents handle 'how'. The orchestrator replaces your task management.

Meta-agents + autonomous delegation

Your Next Steps

1. **Identify Your Current Stage**
   Be honest — most teams are at Stage 2. That's OK. The progression is intentional.

2. **Document One Workflow**
   Pick your most repeated development workflow. Map every step. Identify AI-ready tasks.

3. **Build Your First Task Agent**
   Write a constrained agent for one workflow step. Define what it must NOT do. Run it.

4. **Add Governance Before Scaling**
   Audit logging, boundary enforcement, and error handling — before you add more agents.

5. **Practice the Mental Shift**
   Move from watching to reviewing. From doing to directing. From one agent to many.

If you need help, Improving has a program for bringing teams to Stage 4 effectively in 12 weeks

Anything we want to dive further into?

Questions