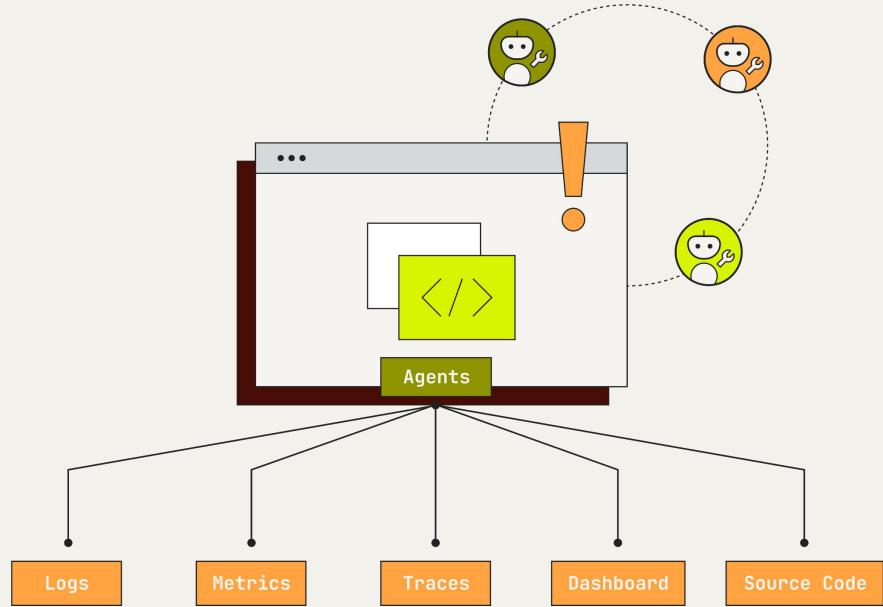


# Agentic AI for software in production





**Mayank Agarwal**  
Founder & CTO  
Resolve AI

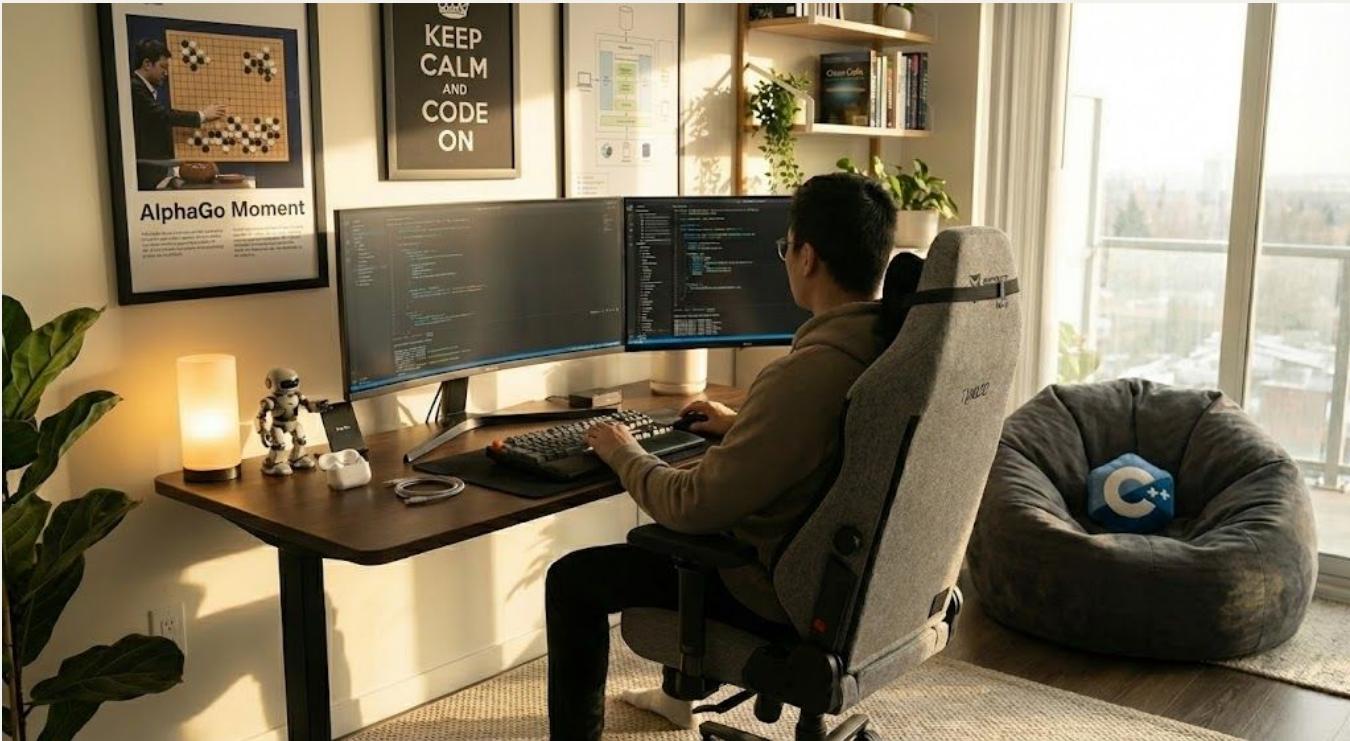


**Milind Ganjoo**  
Member of Technical Staff  
Resolve AI

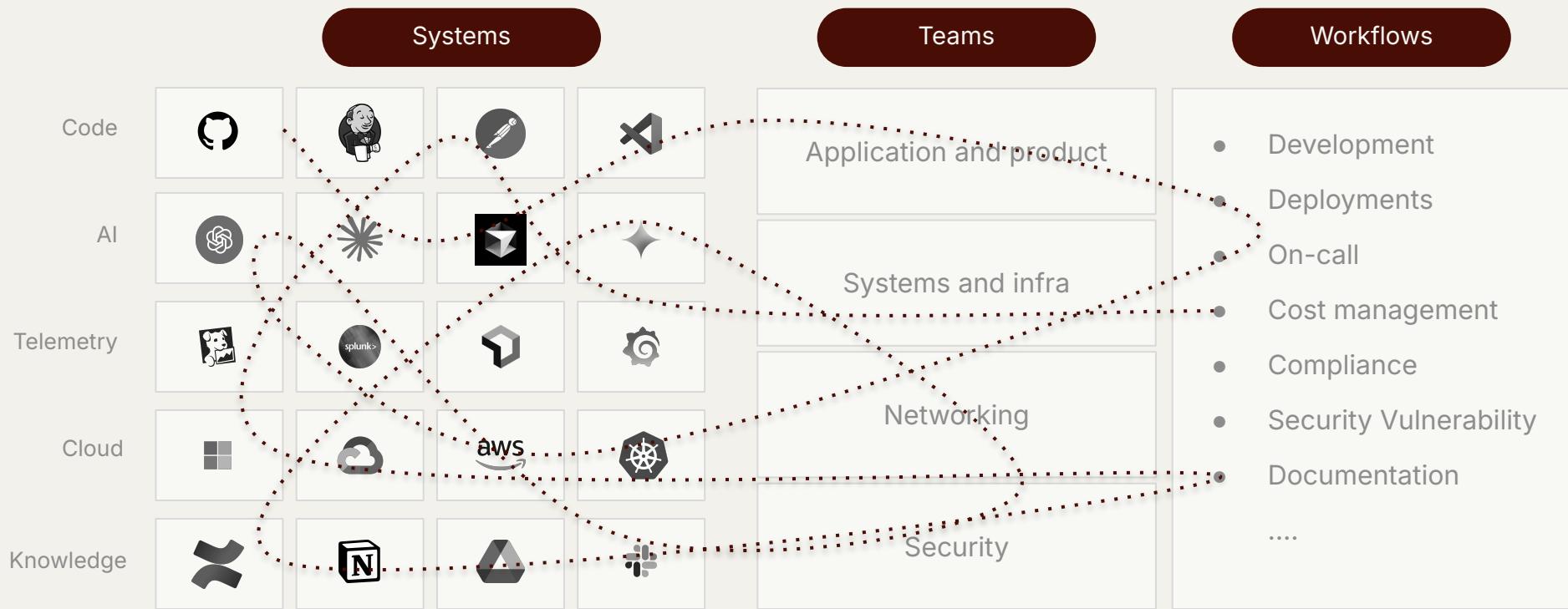




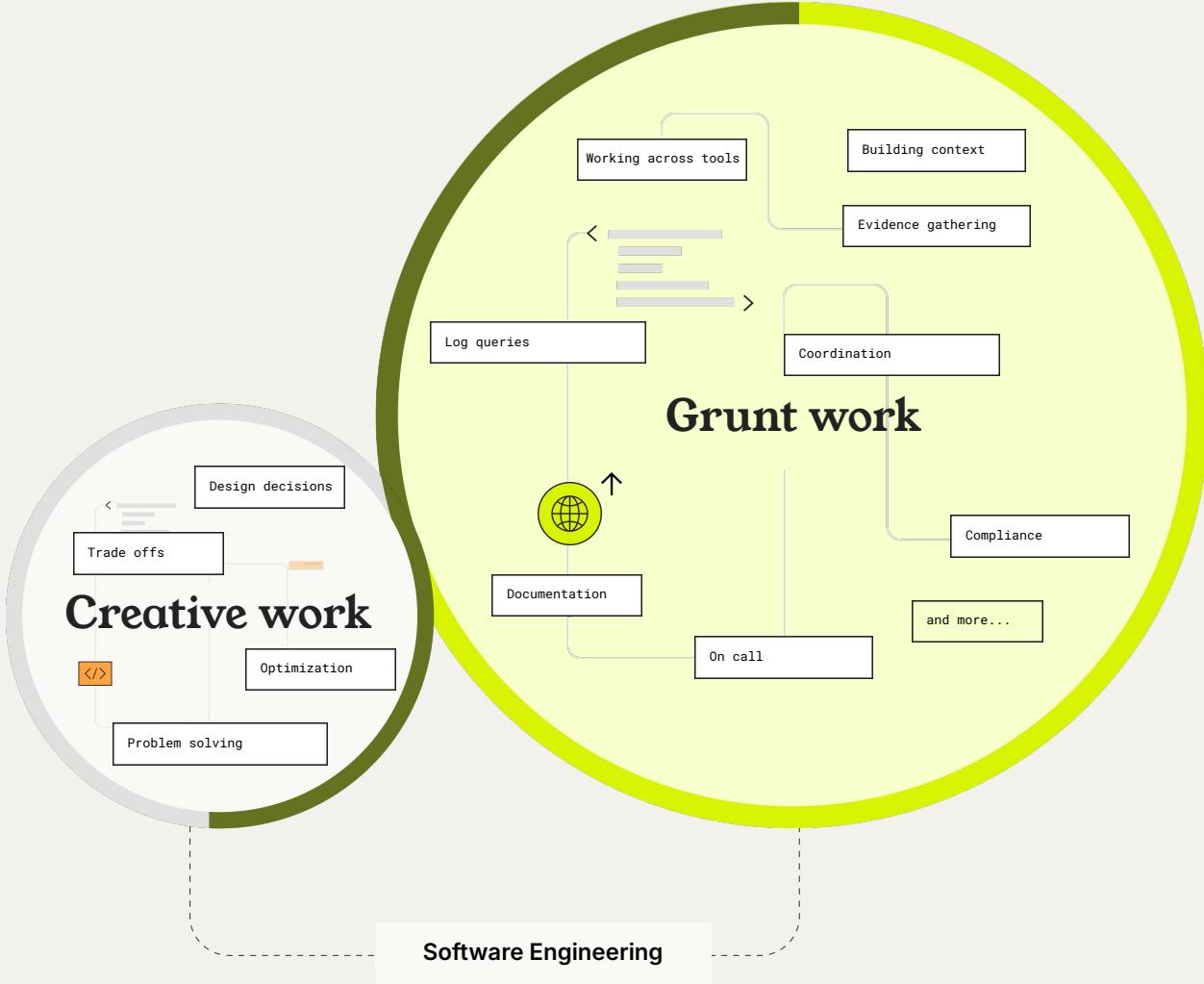
# How we imagine software engineering



# In reality, software engineering is complex and messy

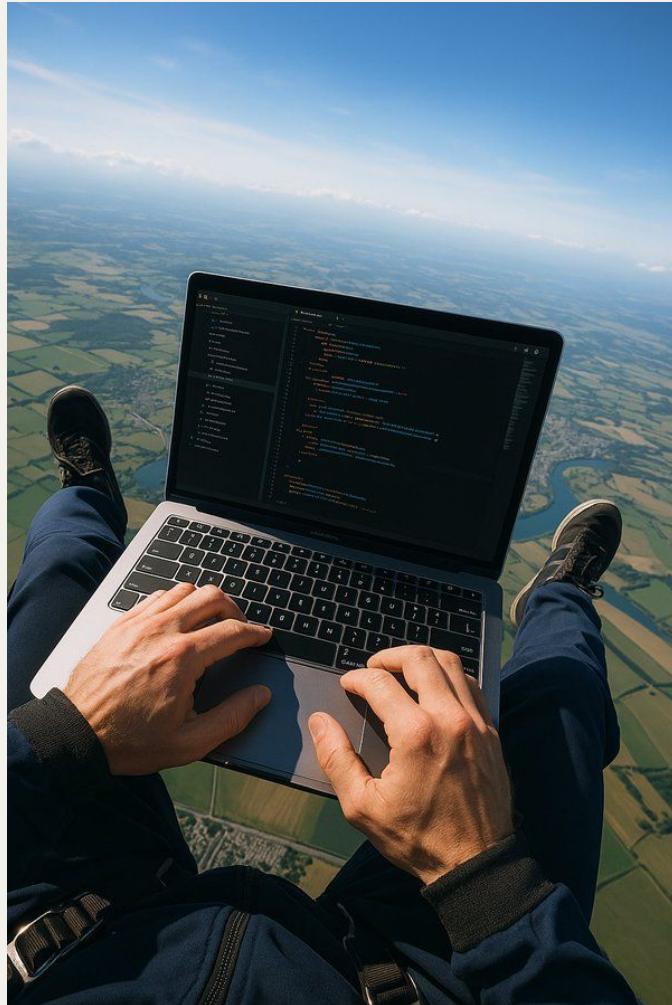


**Software  
Engineers  
spend 70+% of  
their time on  
Grunt work**



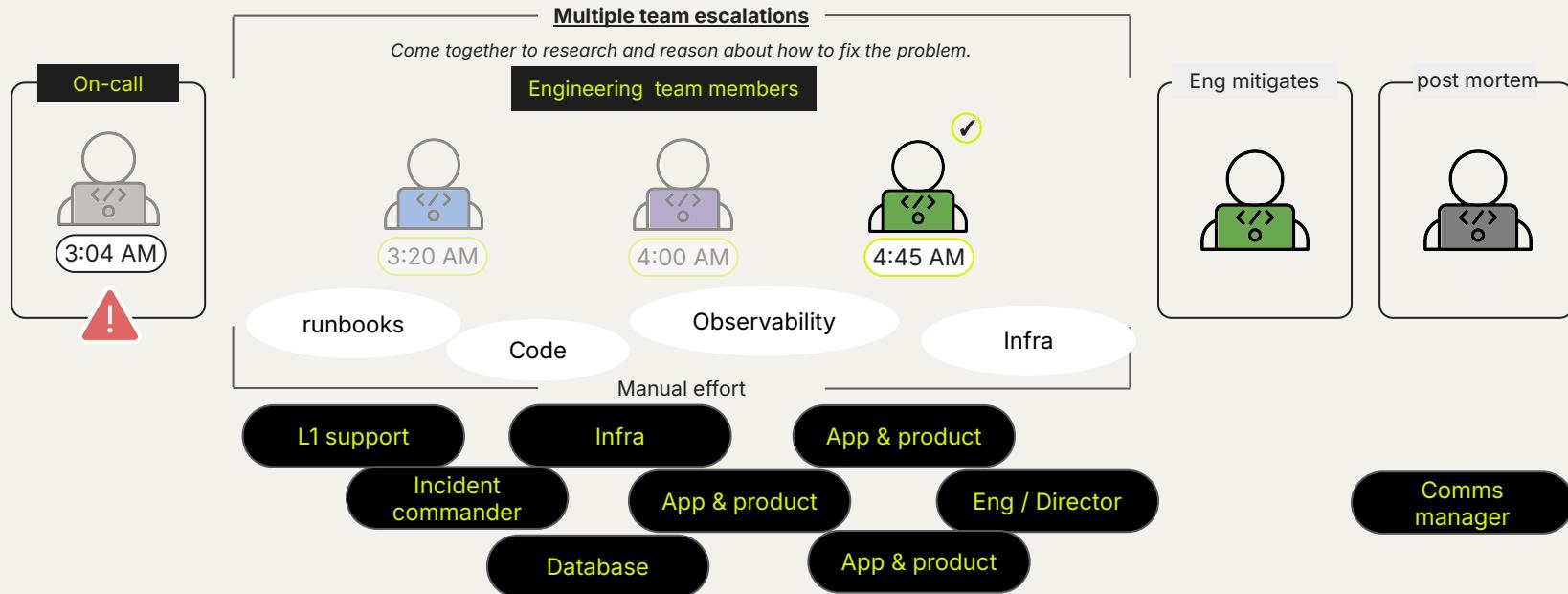


# Life of a software engineer who is on call





# What happens when someone is paged at 3:04 AM?

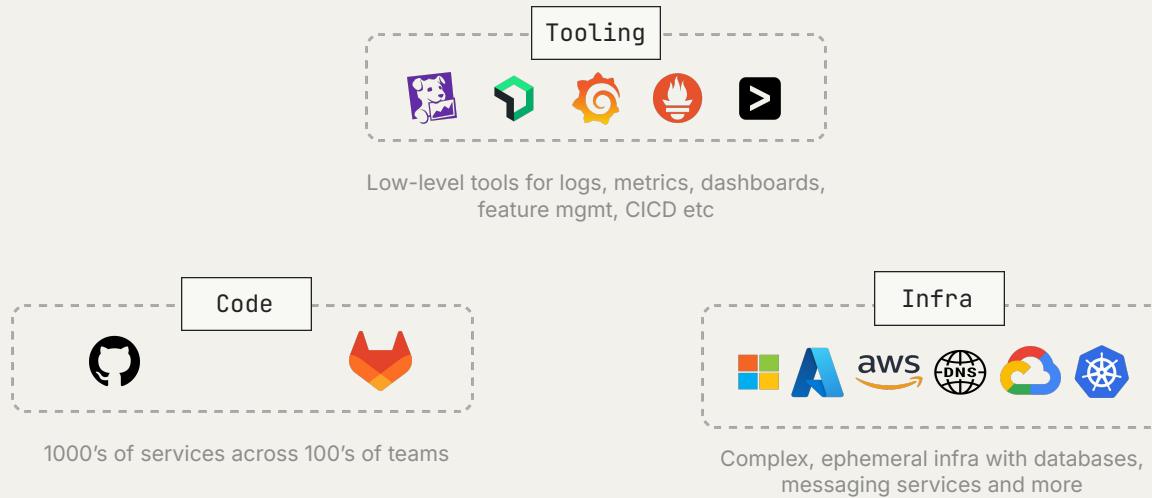




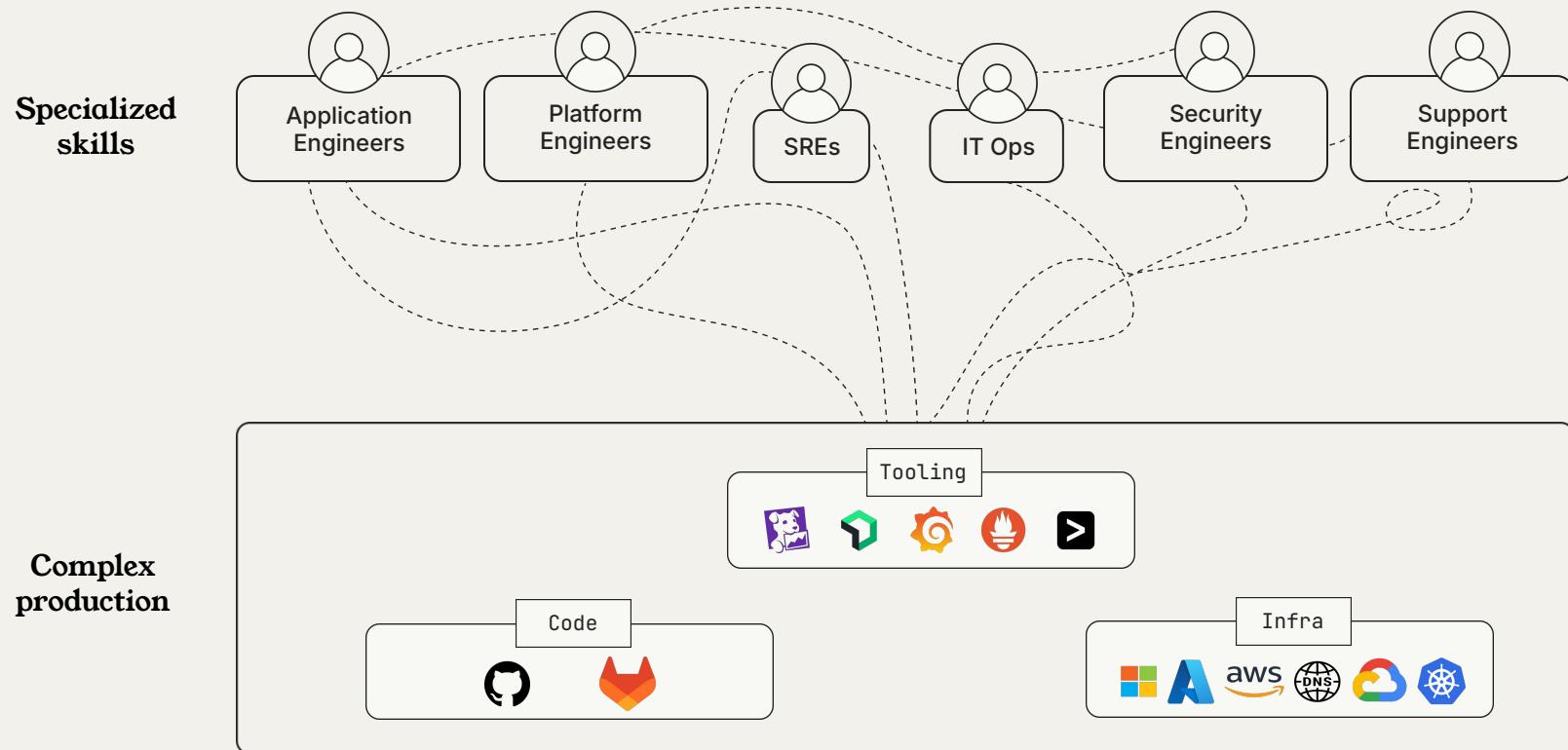
# What makes production hard for humans (and models)?



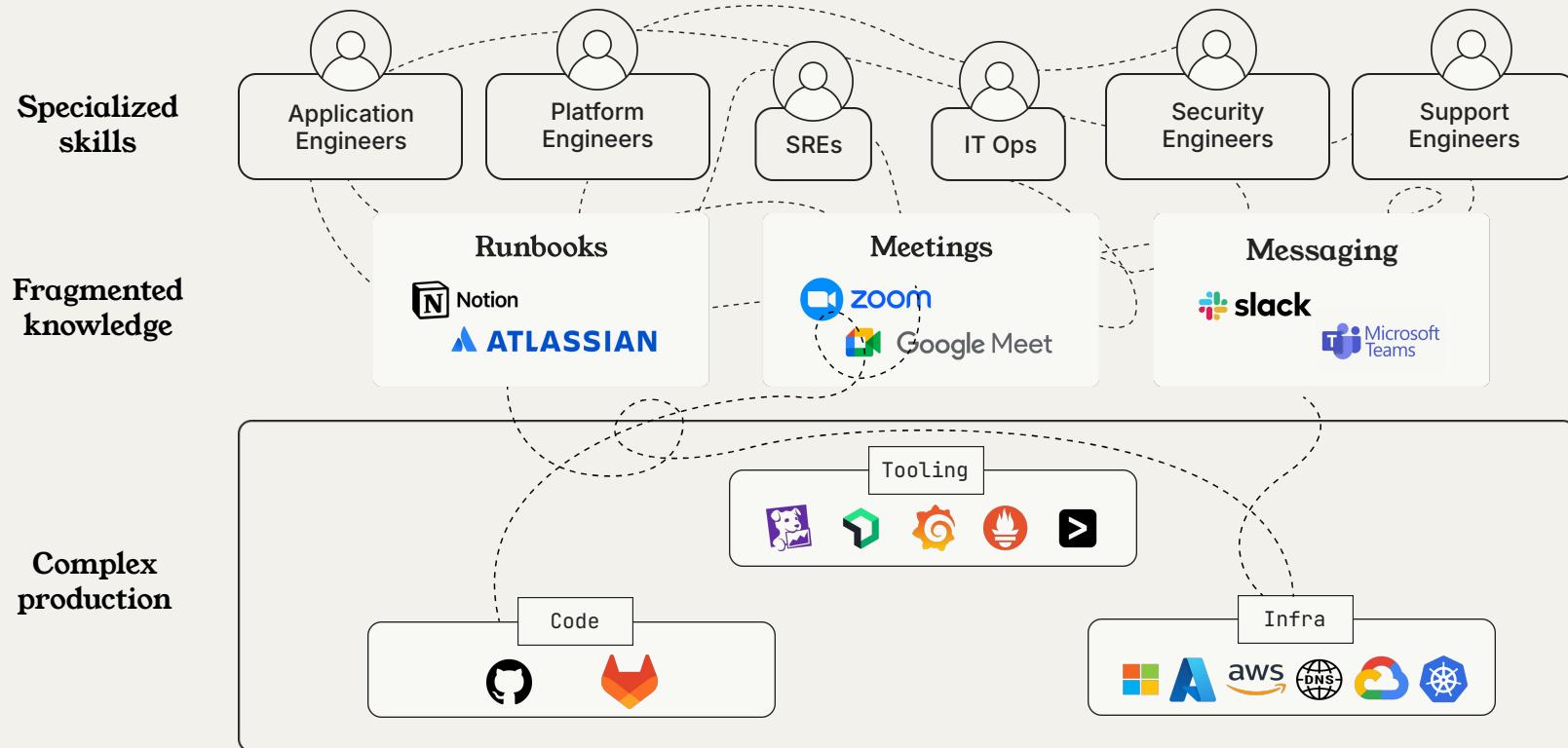
# Navigating siloed data across many systems and tools



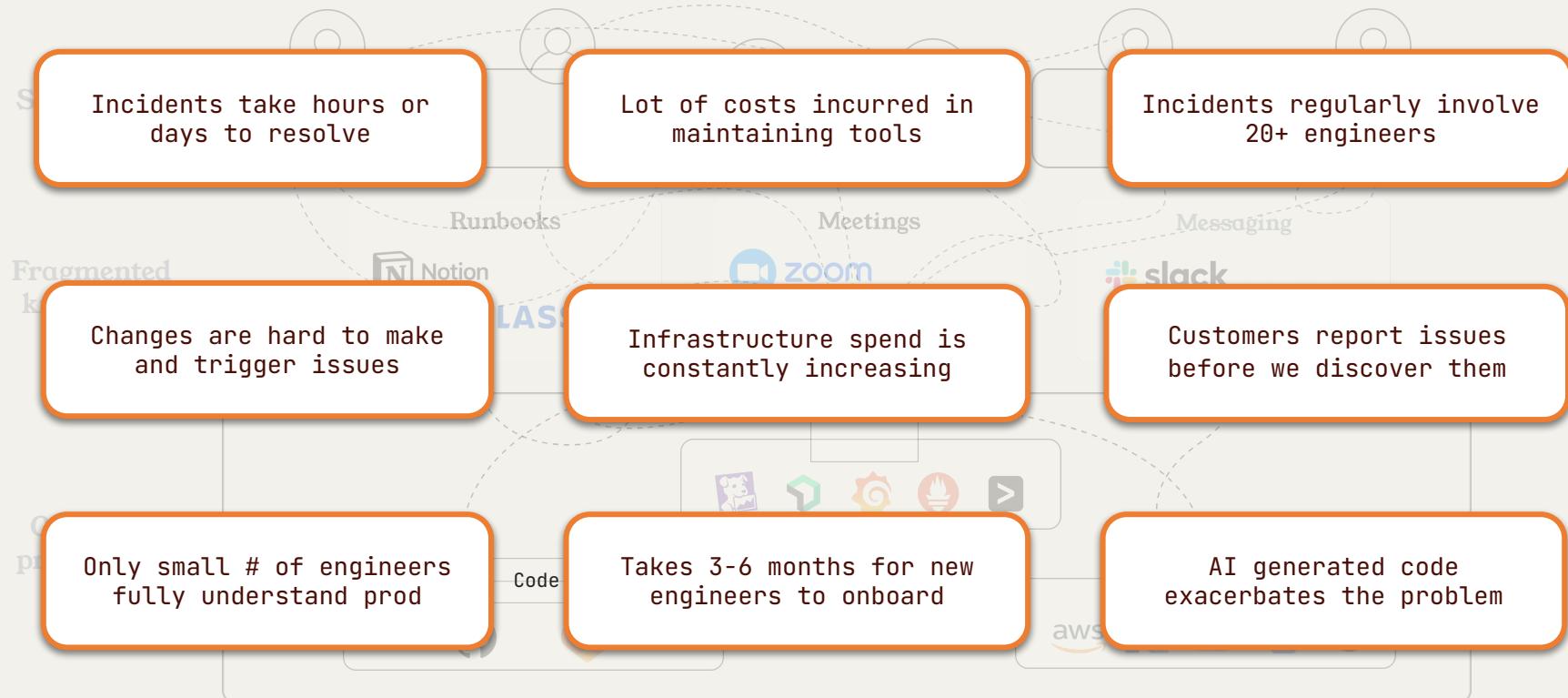
# Coordinating across multiple teams with varied expertise



# Fragmented and often undocumented context



... directly impacts revenue and costs every day

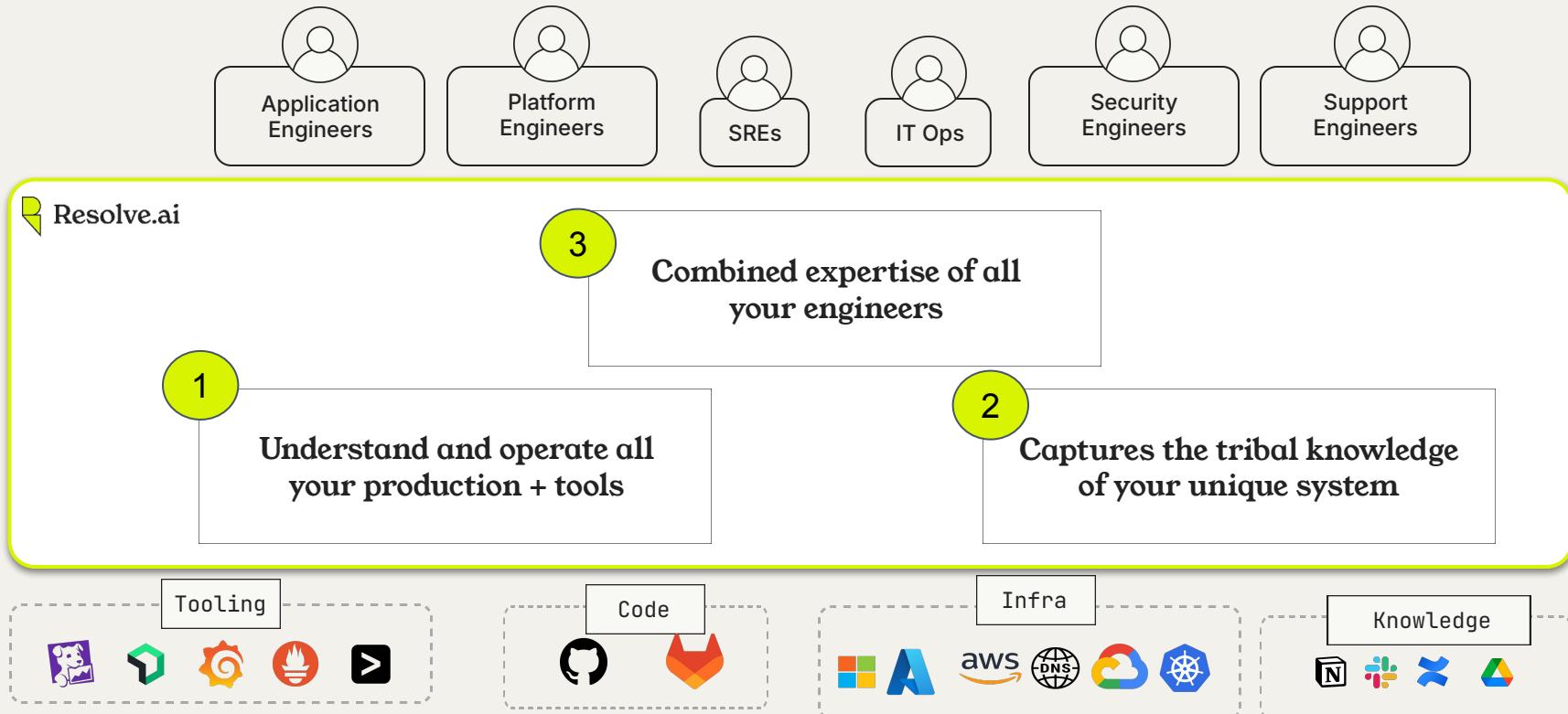




# What is needed for AI to help engineers with production systems?



# Agent-first approach to work on production systems





# Let us see it in action

# How AI for production systems works

01

Production systems are complex and always changing

Understands and operates all your production tools

02

Knowledge is fragmented or undocumented

Captures the tribal knowledge and gets smarter over time

03

Investigations need expertise from multiple teams

Combines expertise of all your engineers

# Production systems are complex and always changing

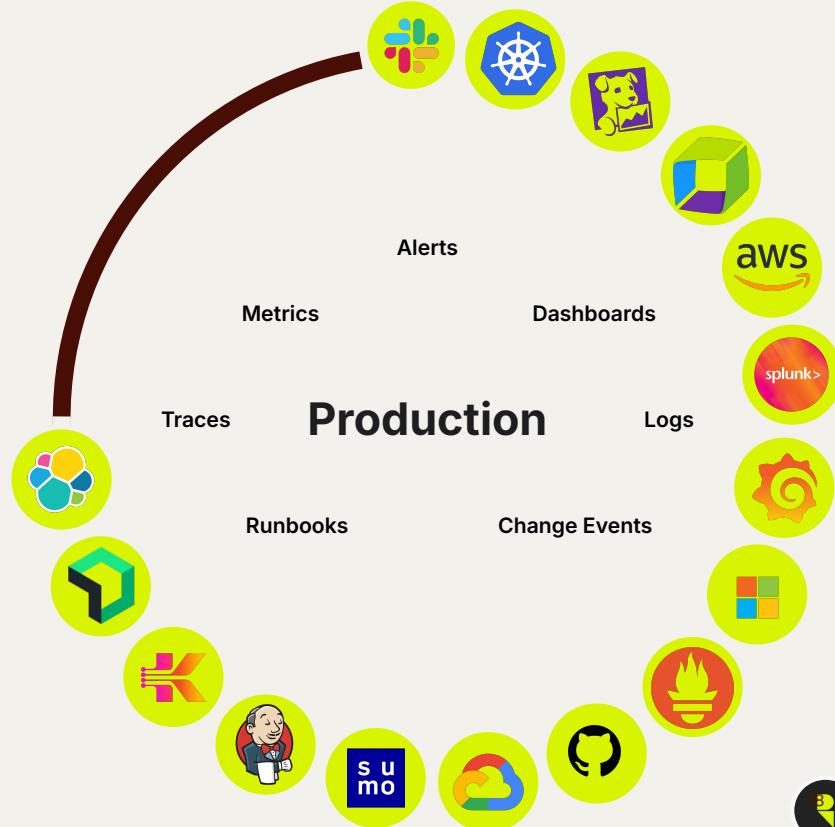
```
243 void showNote(int mm){  
244     FILE *fp;  
245     int i = 0, isFound = 0;  
246     system("cls");  
247     fp = fopen("note.dat", "rb");  
248     if(fp == NULL){  
249         printf("Error in opening file");  
250     }  
251     while(fread(&R, sizeof(R), 1, fp)){  
252         if(R.mm == mm){  
253             gotoxy(10, 5+i);  
254             printf("Note %d Data", R.mm);  
255             isFound = 1;  
256             i++;  
257         }  
258     }  
259     if(isFound == 0){  
260         gotoxy(10, 5);  
261         printf("This Month coming soon");  
262     }  
263     gotoxy(10, 7+i);  
264     printf("Press any key to begin");  
265     getch();  
266 }
```

- Hundreds of tools: different query languages, access mechanisms, and operational behaviors for each
- Should map across code <> infra <> telemetry and identify complex dependencies
- *Massive scale (millions of logs lines, thousands of metrics, dozens of platforms, etc.)*



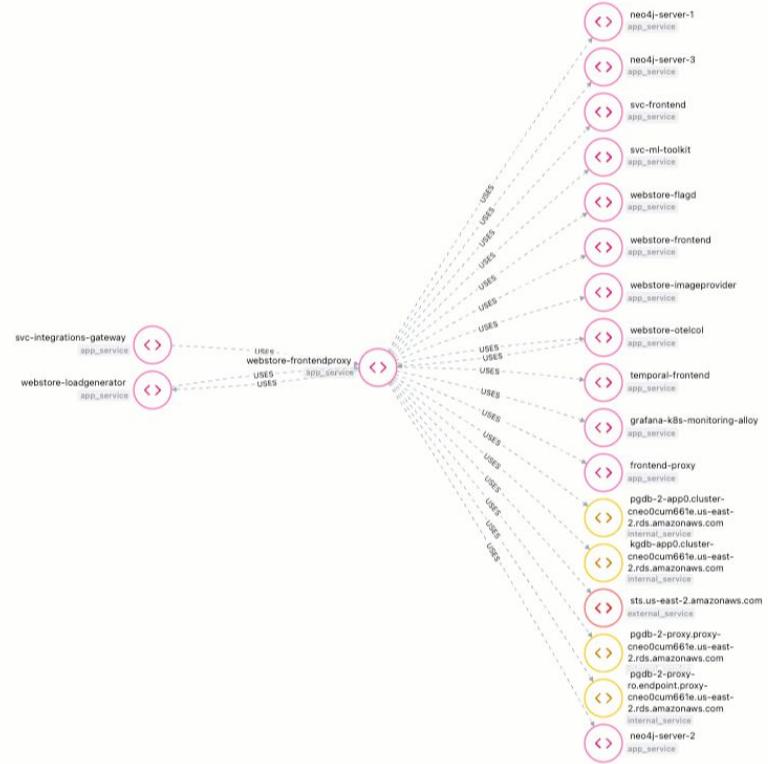
# ★ AI systems should deeply understand your production

- 1 Connects to code, infra, tools, and knowledge



# AI systems should deeply understand your production

- 1 Connects to code, infra, tools, and knowledge
- 2 Models how your systems works



# 💡 AI systems should deeply understand your production

1 Connects to code, infra, tools, and knowledge

2 Models how your systems works

3 Navigates to the right nodes in the graph to gather evidence

I'll help you analyze the ecommerce app code, map out the infrastructure, and provide right-sizing recommendations for the deployments. Let me break this down into steps.

#### ⌚ Exploring code files and changes

Found 9 relevant files

Oct 24, 11:45 AM - Oct 31, 11:45 AM • resource: default • Analyzed 9 files

#### ⌚ Navigating your infrastructure graph

Complete

Oct 30, 11:45 AM - Oct 31, 11:45 AM • resource: default

Now let me analyze the resource usage patterns for these deployments to understand their actual load and provide right-sizing recommendations:

#### ⌚ Exploring metrics

Oct 30, 11:45 AM - Oct 31, 11:45 AM • resource: checkout



# ❖ AI systems should deeply understand your production

- 1 Connects to code, infra, tools, and knowledge
- 2 Models how your systems works
- 3 Navigates to the right nodes in the graph to gather evidence
- 4 Operates every tool or system like experts

Checking your logs from AWS and Loki

Found 6 log queries and 1 metric query  
Yesterday 2:35 PM - 3:35 PM • resource: cart • Analyzed 1,991 logs

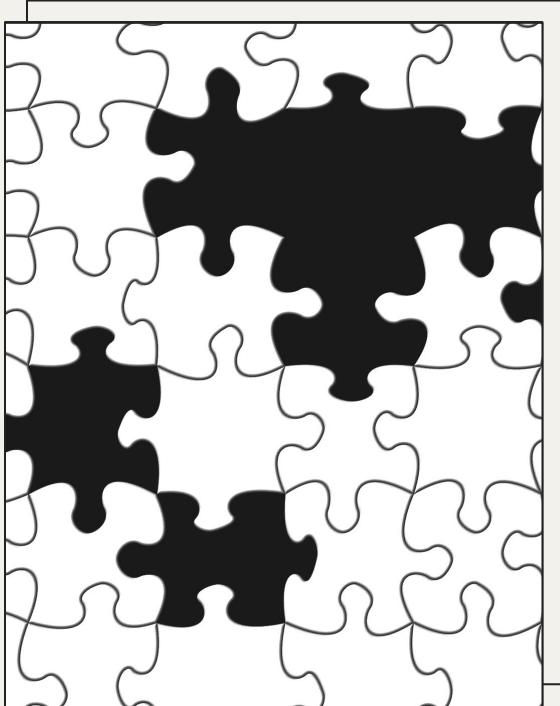
Queries

- log analysis (3 logs) Relevant  
`loki {instance_type="grafana"}`
- Redis connectivity errors (84 logs) Relevant  
`loki {service_name="cart"}`
- Redis connectivity error analysis (100 logs) Relevant  
`loki {instance_type!="abc"}`
- Application error analysis (50 logs) Relevant  
`loki {instance_type!="abc"}`
- log analysis for cart service (98 logs) Relevant  
`loki {instance_type=""}`
- Redis connectivity errors in cart service (87 logs) Relevant  
`loki {k8s_cluster_name="otel-demo"}`
- Cart service during spike periods (200 logs) Relevant  
`loki {k8s_cluster_name="otel-demo"}`





# Knowledge is fragmented or undocumented



- Knowledge is scattered across different runbooks, documentation, chats (if you're lucky! Might be completely undocumented otherwise)
- In-the-loop feedback or learnings are often undocumented or lost
- New investigations need to adapt continuously and require learnings from previous iterations

# ❖ AI systems should capture tribal knowledge and get smarter with every interaction

## 1 Captures company and team-wide knowledge

The screenshot shows a user interface for managing product knowledge. At the top, there's a search bar with the text "product-knowledge". Below it, a navigation bar includes tabs for "Knowledge" (which is selected), "Custom", and "Docs". On the left, a sidebar lists "Configuration", "Alerts", "Slack channels", "Resources", "Dashboards", "Team settings", and "Members" (with a notification count of 4). The main content area is titled "Custom knowledge" and describes it as "Team-specific reference materials, providing context in investigations". It contains a section for "Charles Babbage custom instructions" which was last updated by Charles Babbage 3 days ago. Below this, there are three main sections: "Log selection guideline", "Frontend service", and "Order Processing service". Each section contains a bulleted list of items.

- If the user is asking for something related to order-processing-service that needs log analysis, then DO NOT use the primary log aggregation system as it does not have the application logs or pod logs.
- In code repo and kubernetes (and associated telemetry), this is known as frontend-service
- SSO, SAML, Login, or authentication (case insensitive) is tied to frontend-service

- In code repo and kubernetes (and associated telemetry), this is known as order-processing



# AI systems should capture tribal knowledge and get smarter with every interaction

- 1 Captures company and team-wide knowledge
- 2 Remembers In-the-loop feedback/teachings

Can you show me spikes in 500 errors in the past 24hrs for @frontend

The frontend service experienced **two major spike periods** in 500 errors over the past 24 hours:

**Spike Period (Nov 4 ~19:52 - Nov 5 ~07:00 UTC)**

- **Peak errors:** ~52k/hour on `/api/products/OLJCESPC7Z`
- **Secondary:** ~36k/hour on `/api/recommendations`
- **Recovery:** Error rates dropped to near zero around 07:00 UTC

**Most Impacted Endpoints**

1. ● `/api/products/OLJCESPC7Z` (GET) - highest error volume in both periods
2. ● `/api/recommendations` (GET) - consistently second-highest errors
3. ● `/api/checkout` (POST) - significant errors during second spike

The pattern shows intermittent issues with an 8-hour recovery window between spikes.

46 minutes ago    



# 💡 AI systems should capture tribal knowledge and get smarter with every interaction

- 1 Captures company and team-wide knowledge
- 2 Remembers In-the-loop feedback/teachings
- 3 Retrieves context specific information

I'll investigate this alert about an anomalous decrease in retail\_ledger\_buy transfer workflow activity. Let me start by gathering context from runbooks and past investigations, then analyze the alert details.

#### Runbooks & Artifacts

Found 2 artifacts  
Searched 11 sources

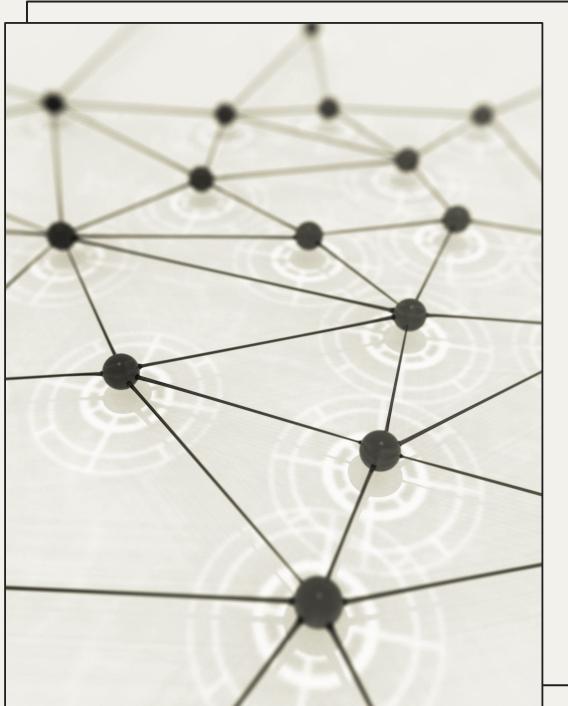
#### Past Investigation Learnings

Found 2 relevant learned insights  
Analyzed 2 investigations

Now let me create my initial investigation plan based on the alert details, runbook guidance, and past learnings:



# 💡 Investigating production needs expertise from multiple teams



- Triaging an incident is hard. Especially for novel incidents or new engineers
- Sequential investigations take a lot of time if not on the right path
- Coordination across teams adds time or loss of information in hand-offs
- Organizational and expertise boundaries make it hard to gather context



# 💡 AI systems should combine expertise of all engineers across teams

## 1 Creates an investigation plan

### Plan

- ✓ Analyzing Alerts
  - ✓ Understanding the alerts and current error log volume
  - ✓ Determining the problem timeframe and scope
- ✓ Gathering the relevant evidence
  - ✓ Search frontend service logs to identify error messages and patterns
  - ✓ Check for request rates, errors, and latency
  - ✓ Identify if errors are originating from downstream dependencies
- ✓ Getting you to root cause
  - ✓ Investigate services for configuration issues
  - ✓ Check for recent deployments or configuration changes



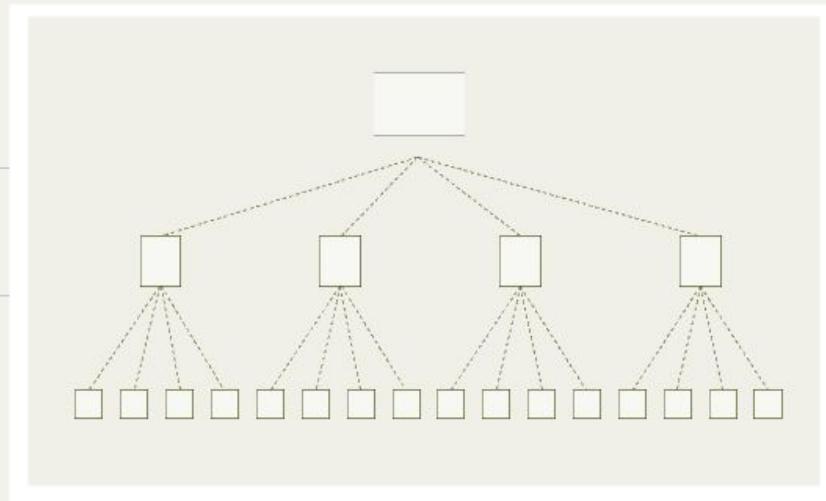
# ❖ AI systems should combine expertise of all engineers across teams

- 1 Creates an investigation plan

---

- 2 Pursues multiple hypotheses in parallel

---

# 💡 AI systems should combine expertise of all engineers across teams

- 1 Creates an investigation plan

---

- 2 Pursues multiple hypotheses in parallel

---

- 3 Refines plan continuously until you get to root cause

## 1 Working Theory

⌚ High Confidence

Intermittent Redis connectivity failure amplified by poor connection retry logic causing cascading latency

The frontend latency alert was triggered by a cascading failure originating from intermittent Redis (valkey-cart) connectivity issues in the cart service. These errors originated from the ValkeyCartStore.EnsureRedisConnected() method \*, which uses a synchronous blocking lock with 30 retry attempts and exponential backoff starting at 1000ms \*.

The root cause is the combination of: (1) an underlying intermittent Redis connectivity issue (likely network-level given the absence of resource constraints and infrastructure changes), and (2) poor resilience in the cart service's connection handling code \* \* that lacks circuit breaker patterns and uses blocking synchronous retries, amplifying the impact of Redis unavailability into 10+ second delays that cascaded to the frontend.

Suggested Next Steps >



# ❖ AI systems should combine expertise of all engineers across teams

- 1 Creates an investigation plan

---

- 2 Pursues multiple hypotheses in parallel

---

- 3 Refines plan continuously until you get to the root cause

---

- 4 Enables multi user collaboration across org boundaries to get to the right answer every time

## Timeline

- Thu October 23 4:24pm - Active database connections began sustained increase on pgdb-2-app0-instance-2 \*
- Thu October 23 4:41pm - Major pod scaling event: ~27 new svc-entity-graph-ingest pods created vs baseline of 1-2 \*
- Thu October 23 4:44pm - Transaction commit rate spiked to 1,100 commits/sec (nearly double baseline) \*
- Thu October 23 4:48pm - PostgreSQL deadlock count began increasing from baseline \*
- Thu October 23 4:49pm - Deadlock rate started sharp increase \*
- Thu October 23 4:49pm - Karpenter evicted underutilized pods, triggering replacement pods \*
- Thu October 23 4:50pm - First deadlock error logged in EventReconciler \*
- Thu October 23 4:54pm - New pods performing database schema initialization \*
- Thu October 23 4:55pm - Latest deadlock error logged \*
- Thu October 23 4:55pm - Alert fired



# How AI for production systems works

01

Production systems are complex and always changing

Understands and operates all your production tools

02

Knowledge is fragmented or undocumented

Captures the tribal knowledge and gets smarter over time

03

Investigations need expertise from multiple teams

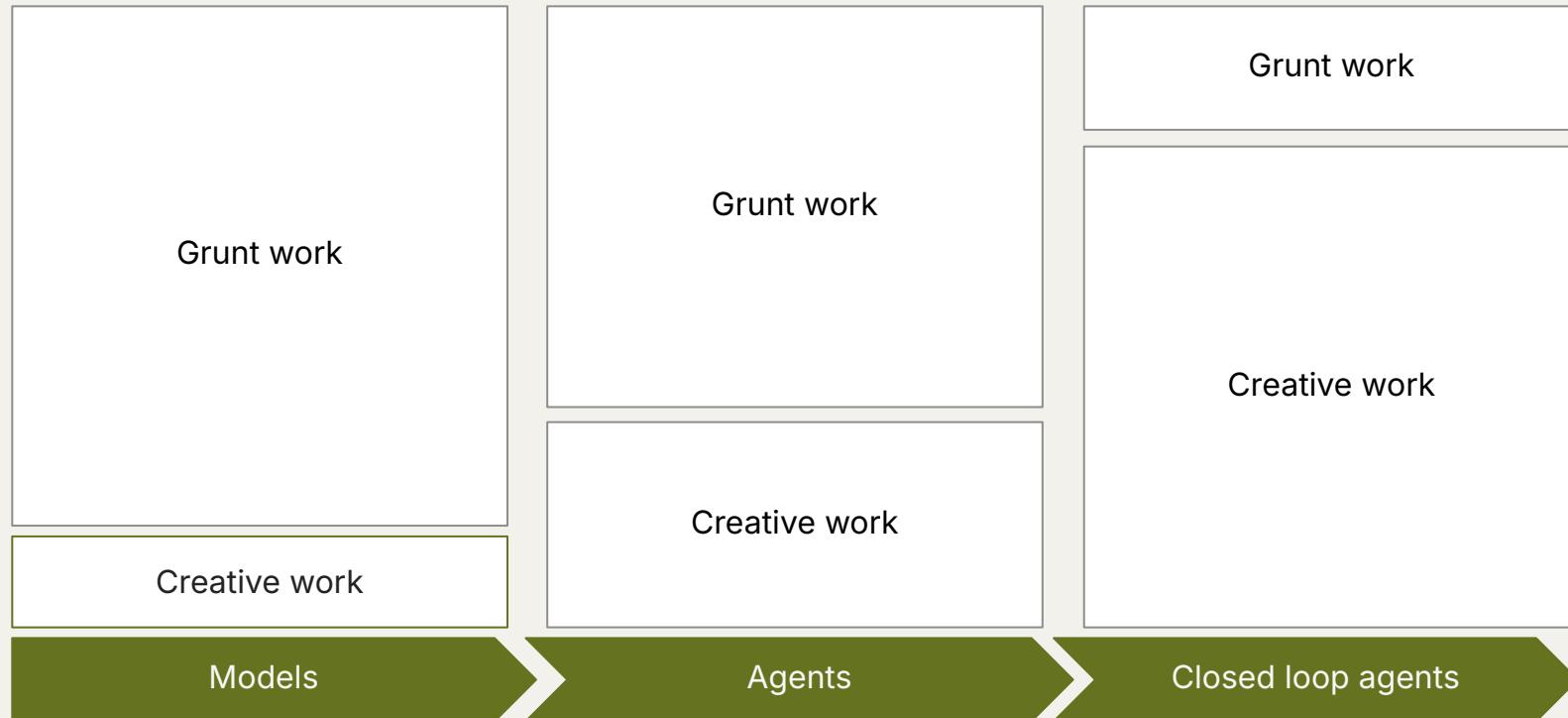
Combines expertise of all your engineers

# ❖ Lessons learned building AI for prod

- **This isn't just a model problem**
  - Navigating production requires a lot of domain expertise which is coded into the architecture. You can't just prompt-engineer models to build production AI
- **Context windows are limited and production context is infinite**
  - Can't fit 10M logs in any context window. Intelligence is knowing WHAT to query, WHEN, and HOW to filter based on production understanding.
- **Working with tools is a non-trivial problem**
  - Raw APIs are unusable: large responses, outputs are messy, and meant for humans. Must build AI systems that can filter noise, return structured summaries, handle errors gracefully, work in parallel
- **Evals are as hard as the product itself**
  - Building evals requires replicating production complexity - (services, dependencies, etc.,). Without evals, you can't trust the outputs.

# ❖ AI is changing software engineering

By next year software engineering will look fundamentally different





Resolve.ai

# Thank You



We're hiring!



[linkedin.com/in/resolveai](https://linkedin.com/in/resolveai)



@resolveai

# Q&A

◊ LET'S DISCUSS