

# ACCELERATE **CHICAGO**

Craft software with confidence in 2026

# Agents gonna agent

Deterministic orchestration for systems we can't predict



Eric Johnson  
Developer Advocate  
@edjgeek

# Who am I?

- Principal Developer Advocate
- Serverless nerd
- 35+ years software architect
- Father of 5



# QUESTIONS?

ASK ME IN THE APP



# The challenge

What are we working back from?

# A buddy runs a disaster relief organization

He can:

- Evaluate the scope
- Estimate affected populations
- Calculate resources
- Coordinate a response.

He is so good that:

- FEMA calls him
- The Red Cross calls him
- The Salvation Army calls him
- He is genuinely one of a kind

**He does it all ... on a legal pad**

# The bus factor

What  
happens if my  
friend gets hit  
by a bus?



# So, I built a system

Multiple agents working together  
the way he works

- Triage
- Assess
- Plan
- Assign

The AI was easy!

Building a reliable system we  
could trust was less easy

# Pop quiz

You have an Ai agent running a 10-step workflow. Each step is 85% accurate.

What's the overall success rate?

# 20%

$$0.85^{10} = 0.197$$

Four out of five runs fail somewhere in the chain

*- Lussar's law*

# 40%

Of multi-agent pilots  
**fail within six months  
of production**

*- Gartner*



# It's not the agents

## ✓ Agent A

Does its job correctly. Produces the right output for its context. Checks out.

## ✓ Agent B

Also does its job correctly. Handles its task as designed. Also checks out.

## ✗ The system

Agent A finishes before B expects. Output format mismatches. Both write the same resource simultaneously. Classic distributed system failure.



No single agent is wrong. **The system is wrong.** Race conditions, state drift, missing contracts at component boundaries. We've been here before.

# The implicit contracts

Every handoff between agents is what I call an **implicit contract**. The timeout the calling agent assumes. The output format the receiving agent expects. What "success" means. What happens on failure.



## Timeout?

How long does the sender wait before assuming failure?



## Format?

What schema does the receiver expect to parse?



## Success?

What does a successful response even look like?



## Retry?

On failure, who retries — and from where?

None of it is written down. It lives in the prompts — if it lives anywhere at all. This is exactly how distributed systems fail quietly. Not with a bang. **With a mismatch nobody anticipated.**

# The wrong fixes

## ✗ Better Prompts

More instructions, more examples, more constraints in the system message. Taken to truly remarkable extremes. **But better prompts don't fix timing bugs.**

## ✗ Bigger Models

If the agent made the wrong call, clearly it wasn't smart enough. **But bigger models don't fix state drift.**

## ✗ Retry Loops

Just run it again until it works. **But retry loops without checkpoints re-run completed work and compound errors.**

These fixes address the agents. **The problem is between the agents.**

# The right fix = orchestration

Deterministic orchestration for systems we can't predict

# Durable orchestration landscape

## Azure Durable Functions

Replays orchestrator against event history in Azure Storage

## Temporal

Open-source, replays workflow code against a persistent event history

## Restate

Replays from a durable execution journal of completed operations

## Ingest

Re-executes functions with step-based memoization, returning cached results for completed steps

## AWS Lambda durable functions

Re-executes functions with step-based memoization, returning cached results for completed steps

# Benefits of durable functions

## The step

A unit of work that runs exactly once and creates a checkpoint when it completes

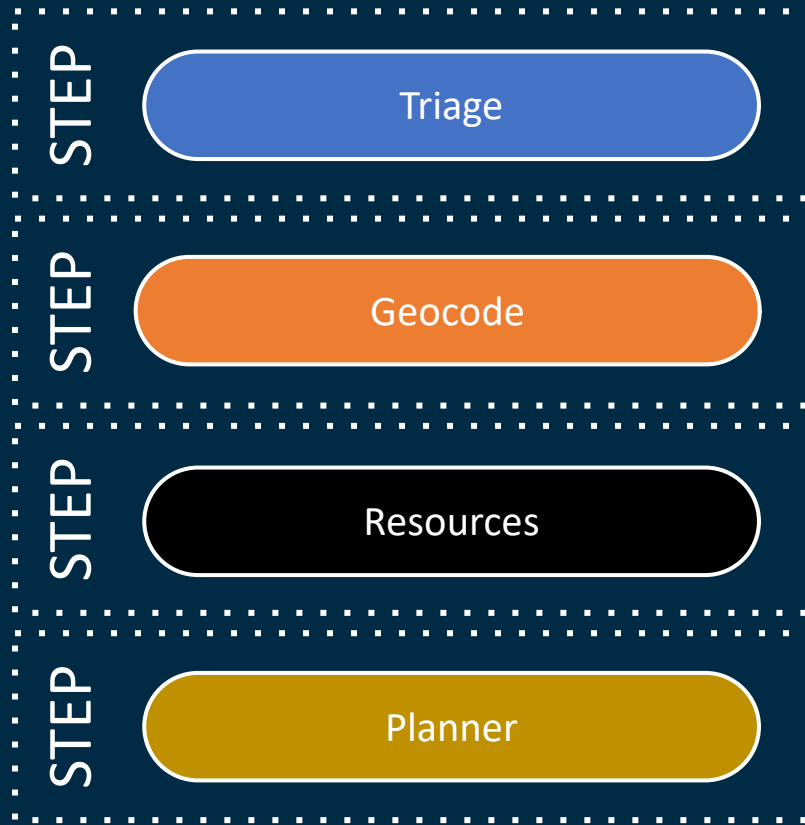
## The checkpoint

The results of a completed step, stored for replay

## The replay

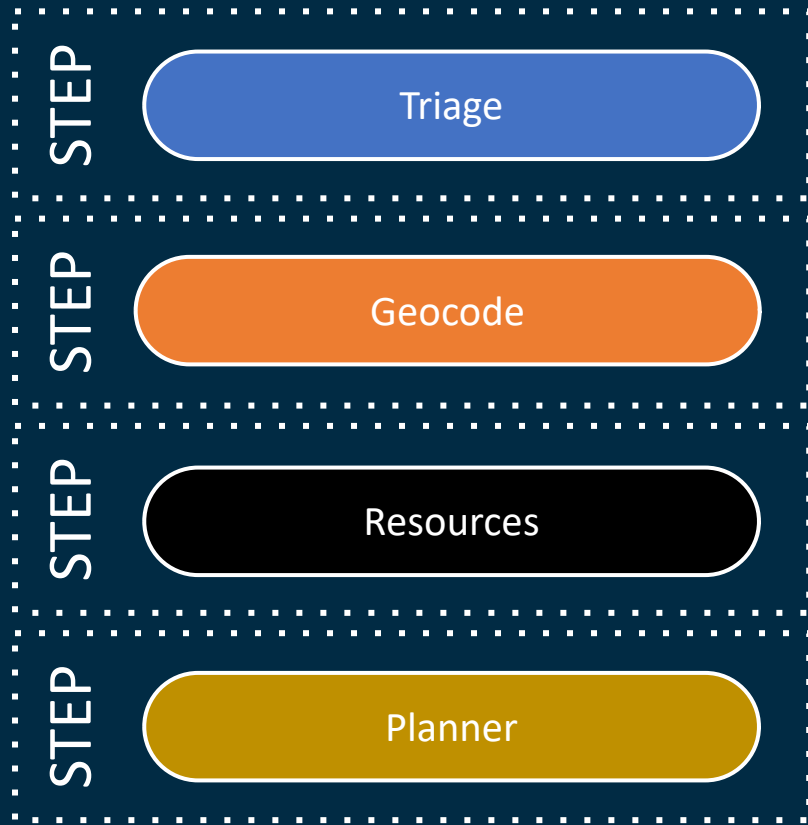
Resume from where you left off, skip completed work

# The replay



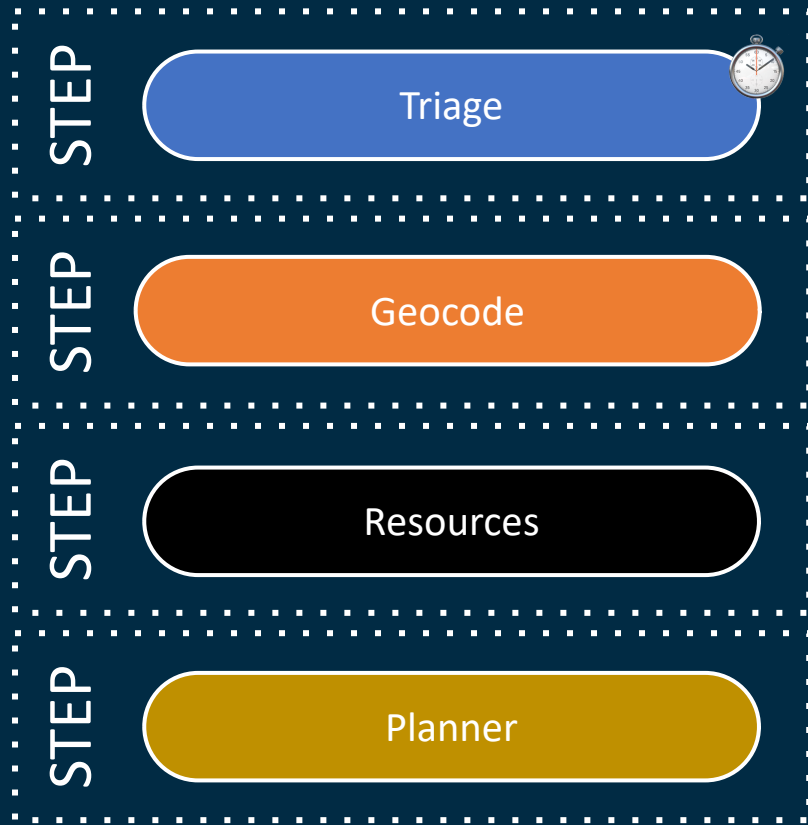
## Checkpoints

# The replay



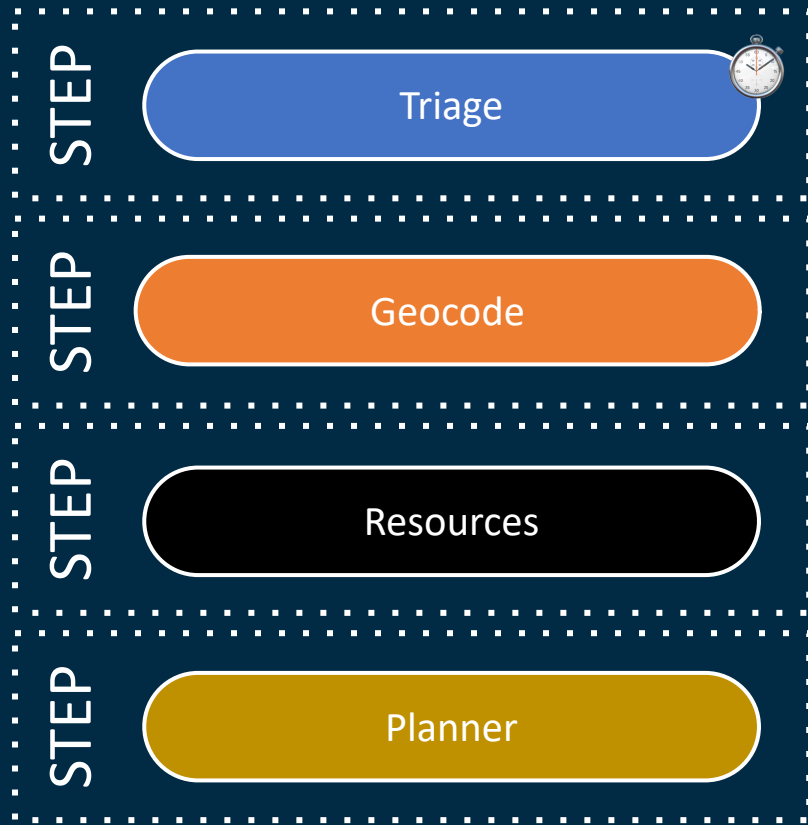
Checkpoints

# The replay



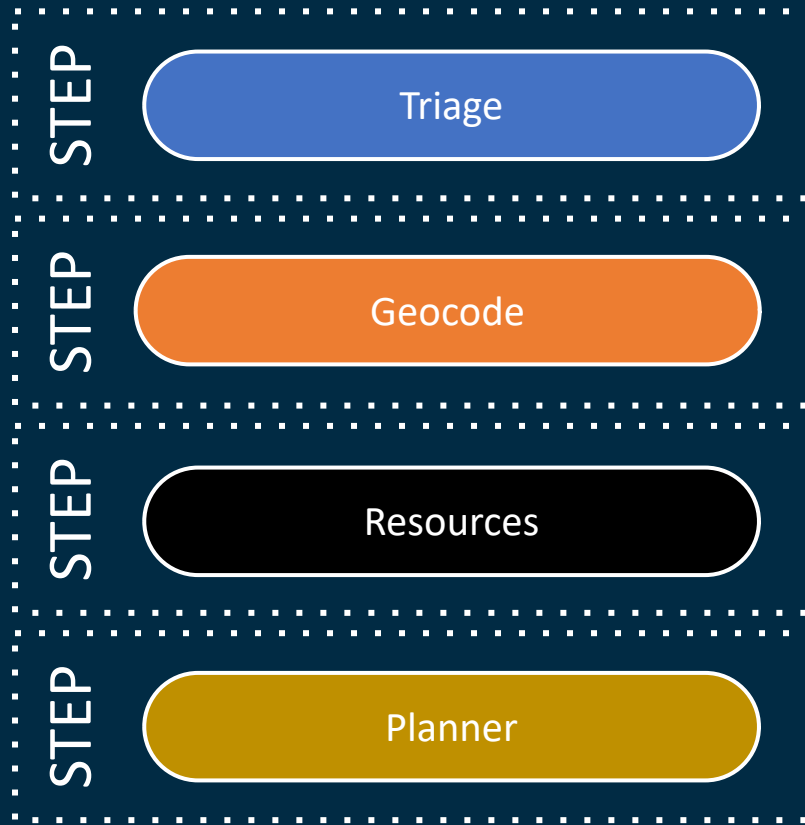
Checkpoints

# The replay



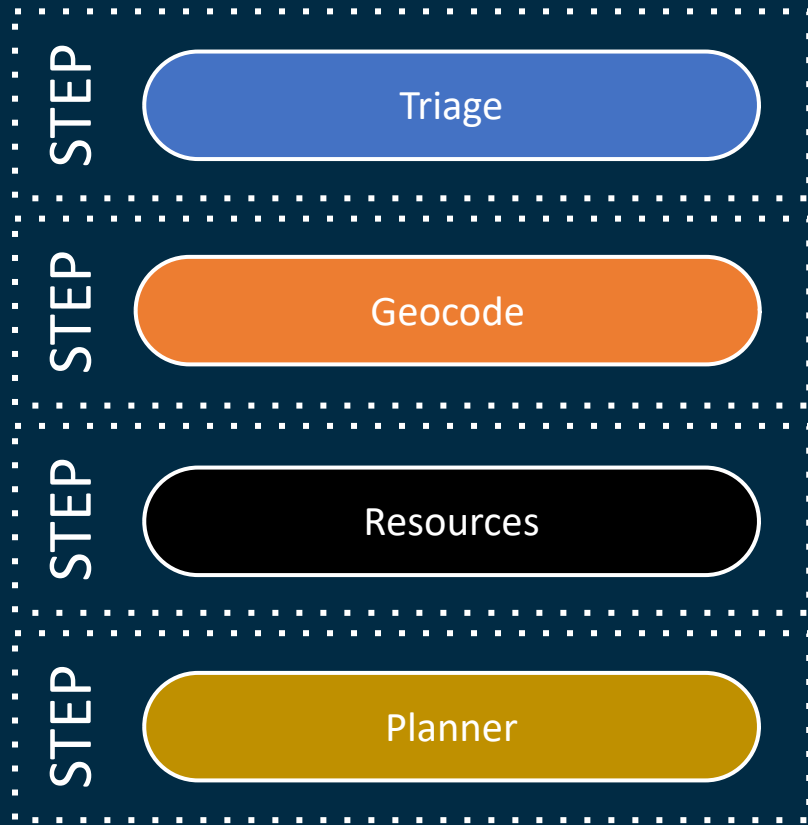
## Checkpoints

# The replay



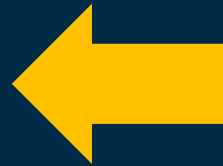
Checkpoints

# The replay



Checkpoints

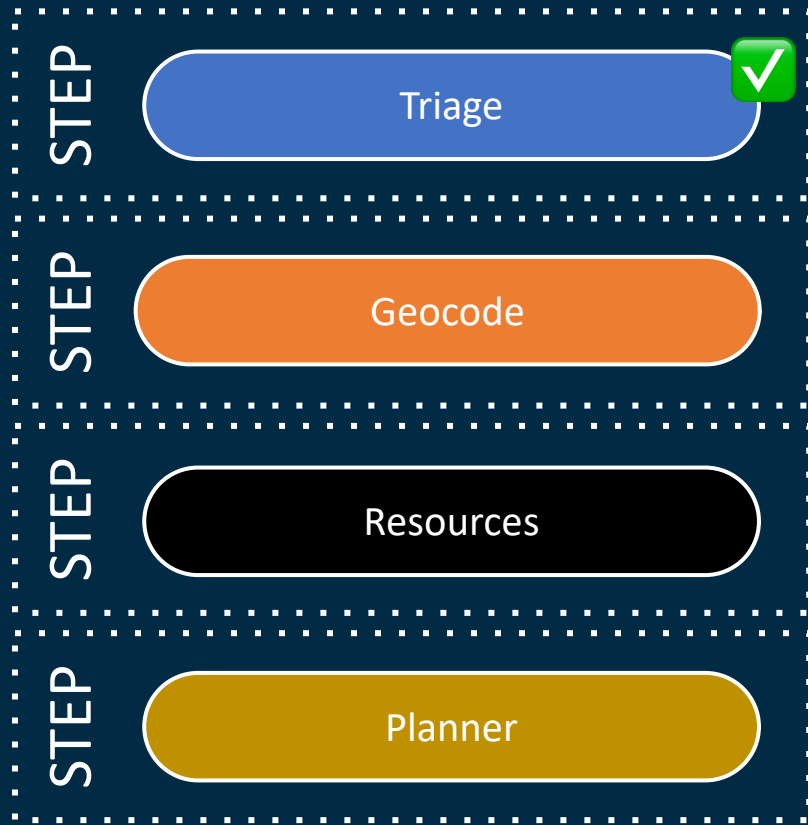
# The replay



## Checkpoints

✓ Triage results

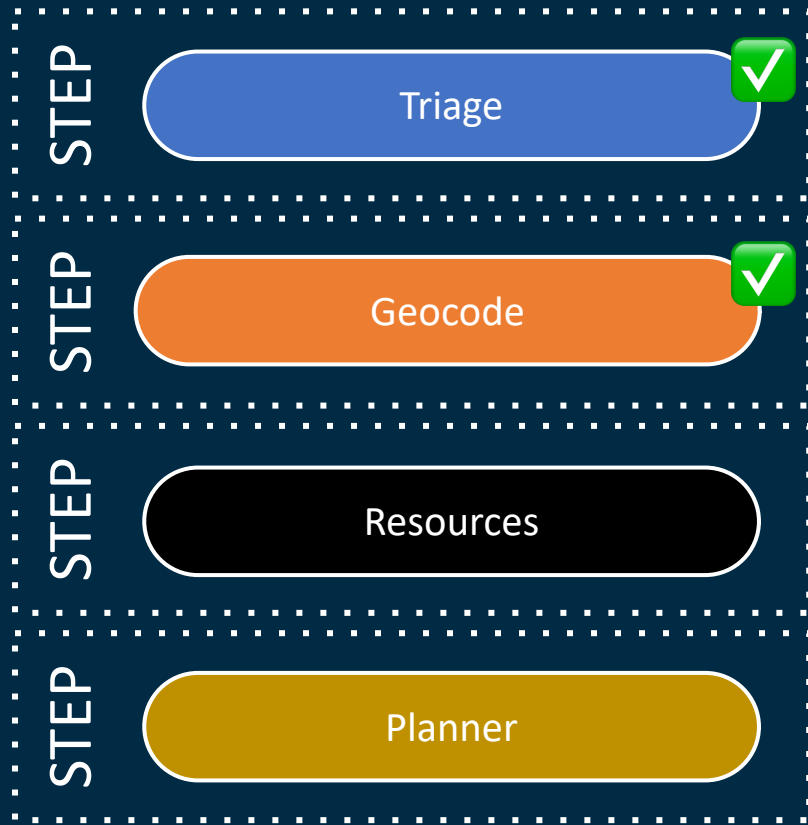
# The replay



## Checkpoints

✓ Triage results

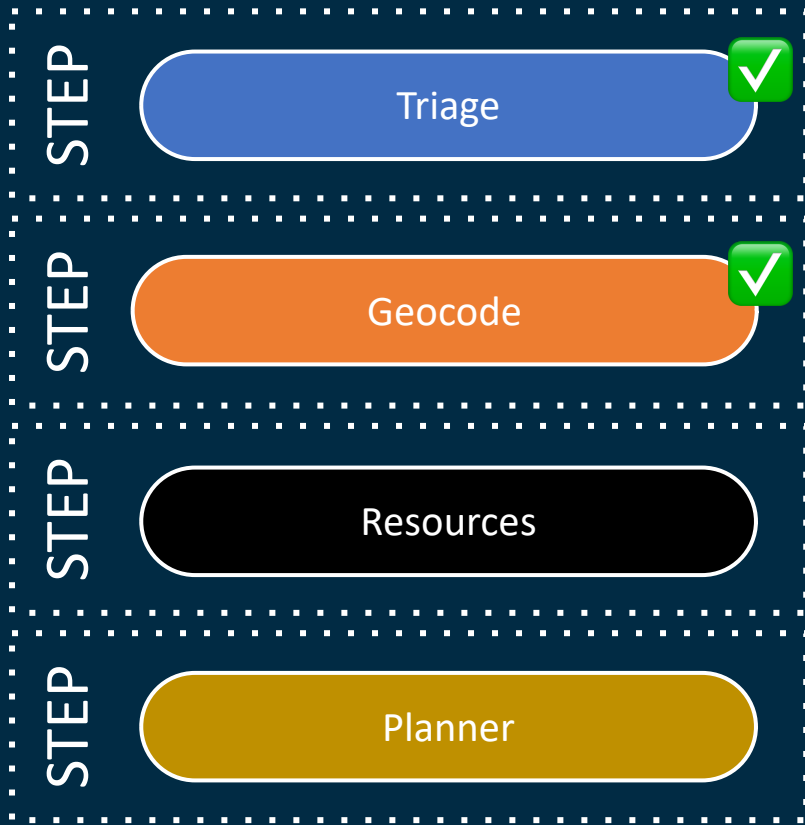
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results

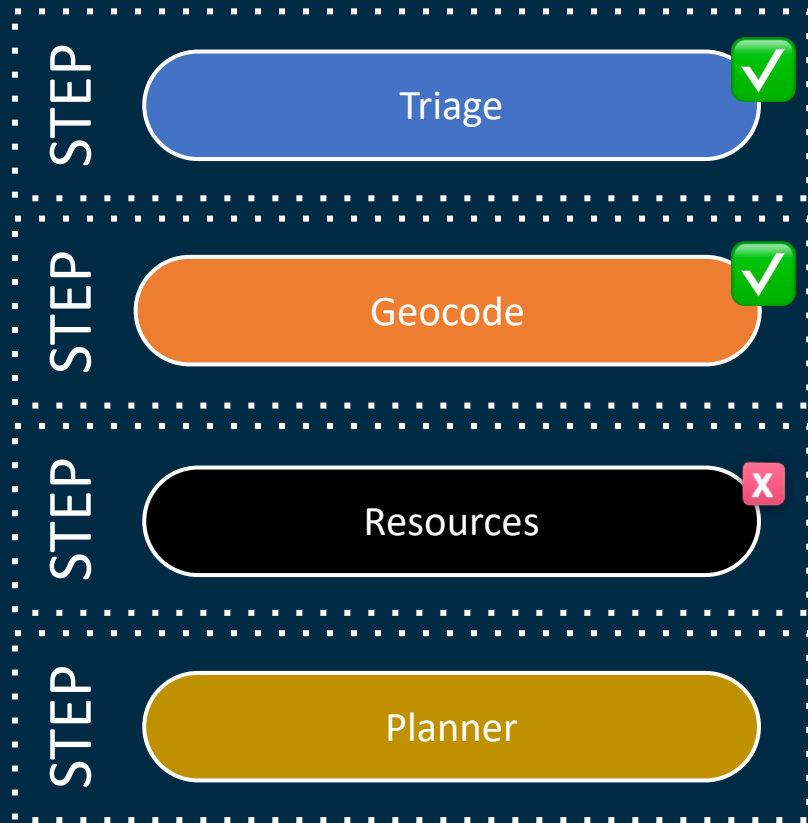
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results

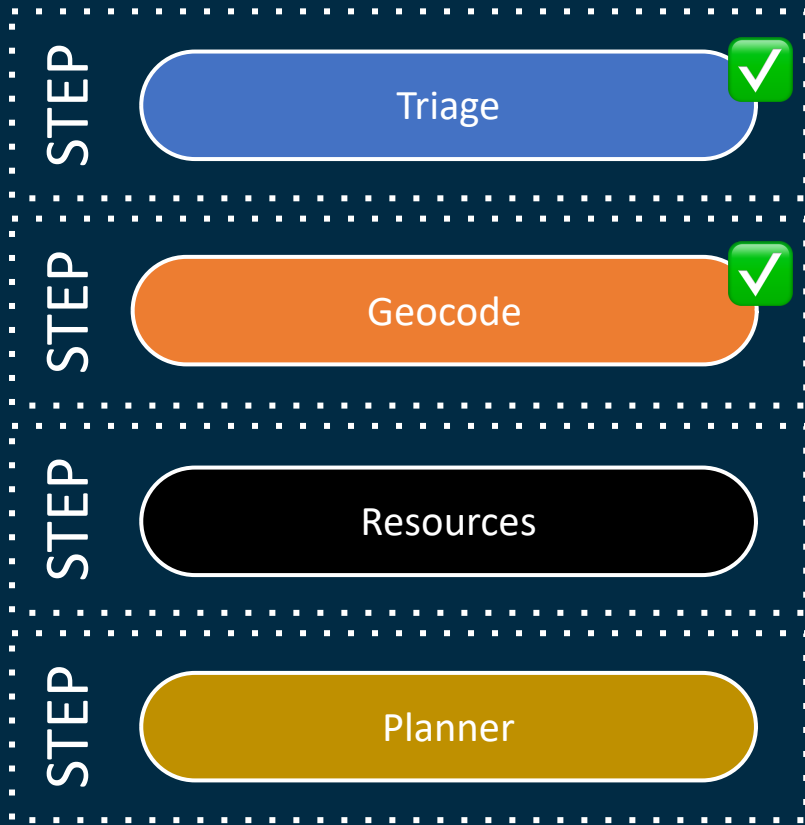
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results

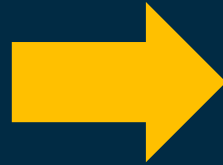
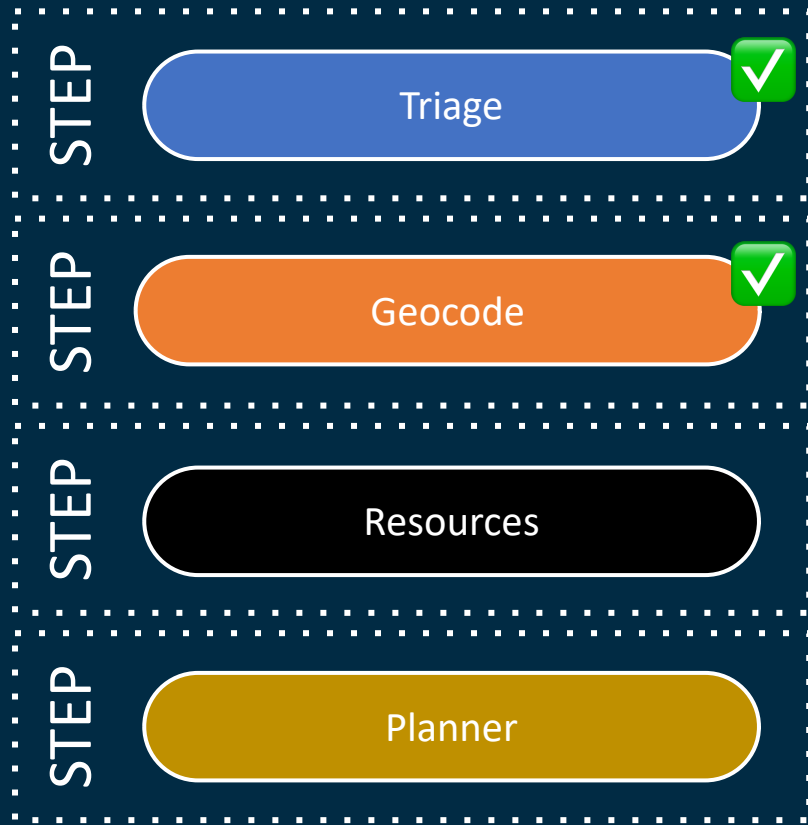
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results

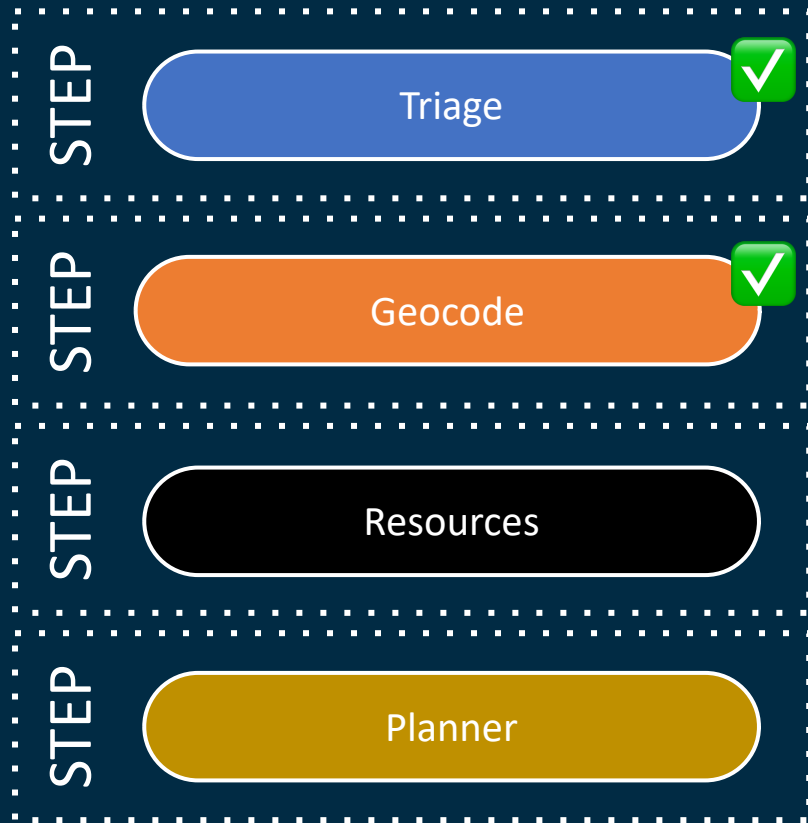
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results

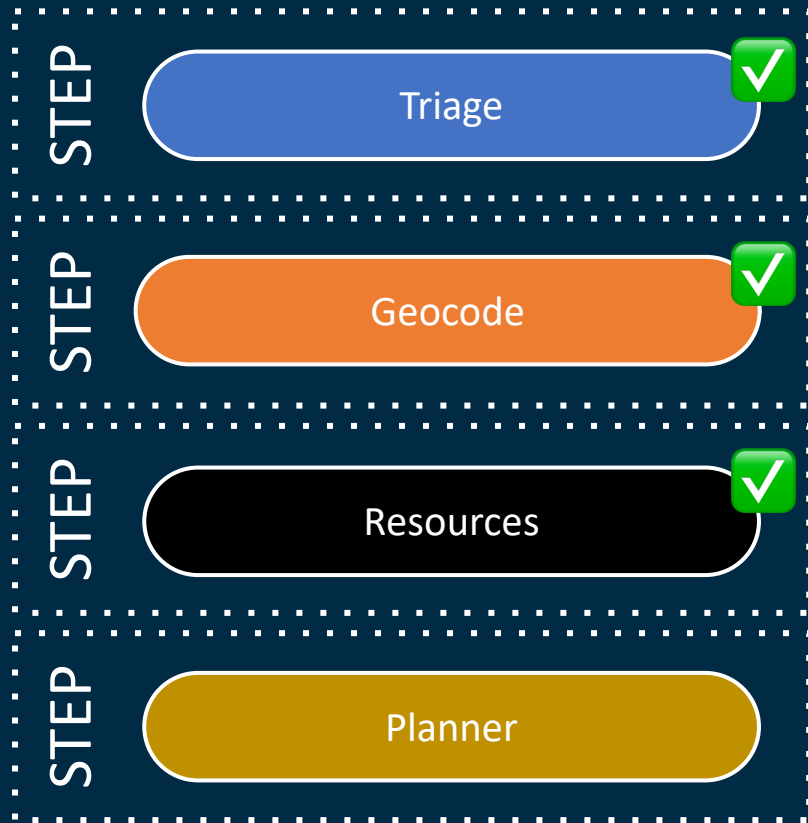
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results

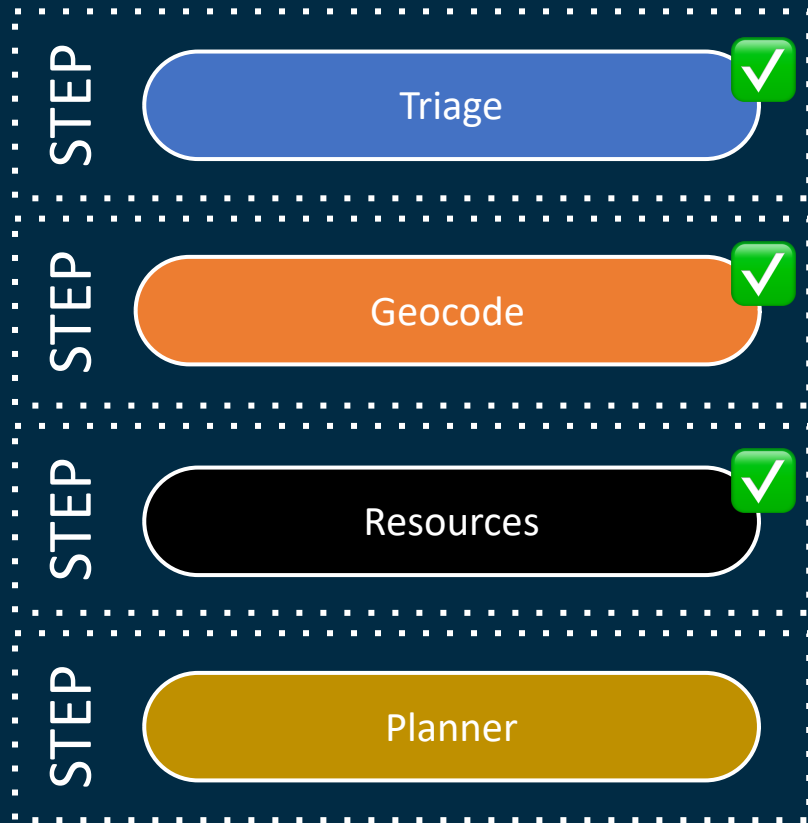
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results
- ✓ Resources results

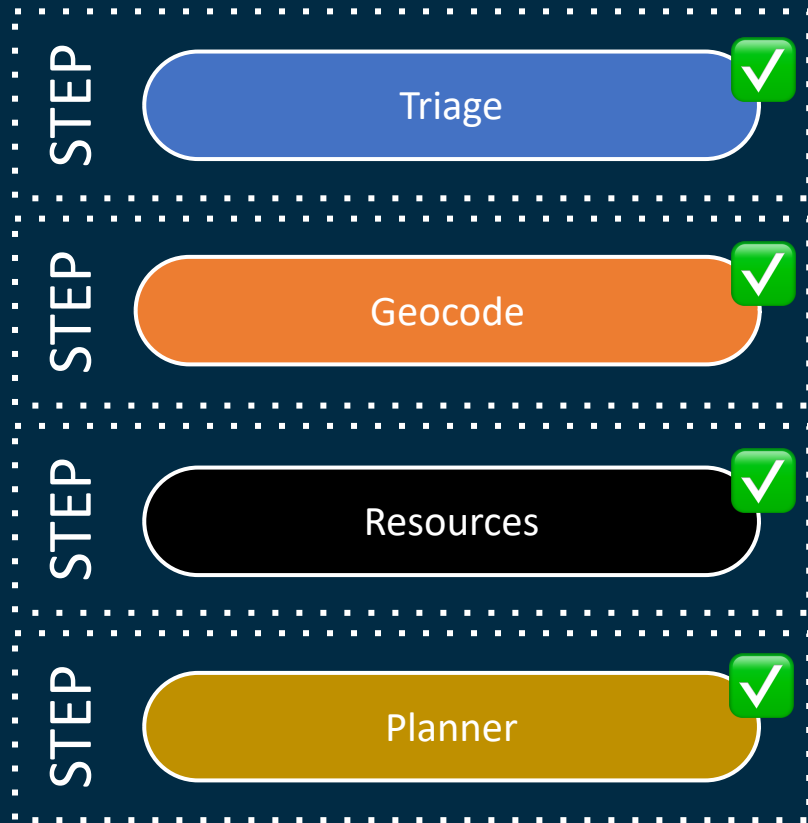
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results
- ✓ Resources results

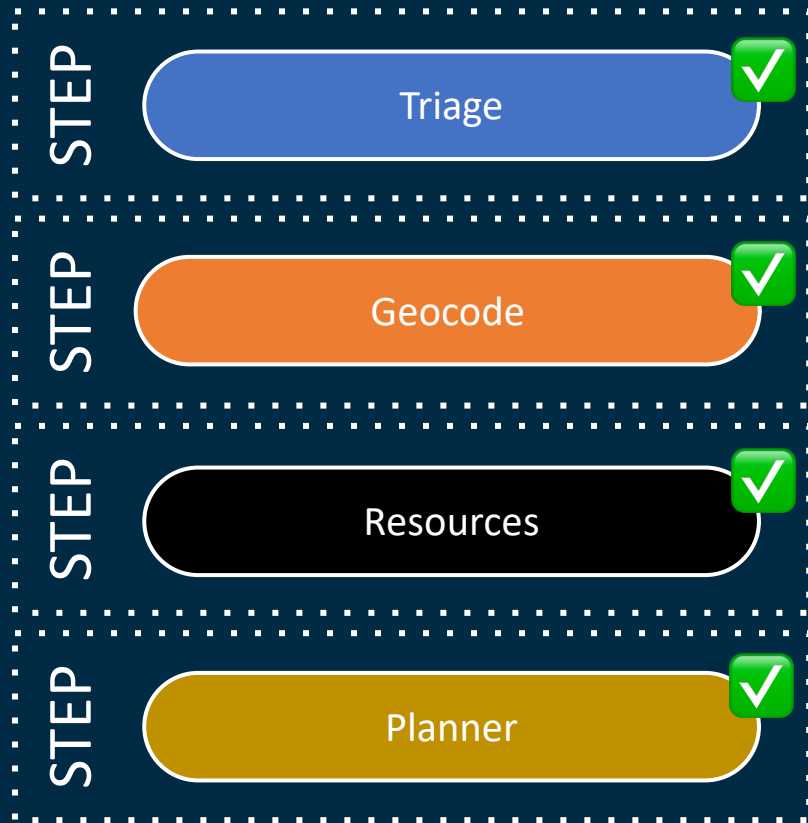
# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results
- ✓ Resources results
- ✓ Planner results

# The replay



## Checkpoints

- ✓ Triage results
- ✓ Geocode results
- ✓ Resources results
- ✓ Planner results

# The handler()

```
@durable_execution
def handler(event: dict, context: DurableContext) -> dict:
    triage = context.invoke(arns["triage"], name="triage")

    results = context.parallel([weather, population])
    w, p = results.get_results()

    context.step(step_push(incident_id, "done"))

    return {"status": "COMPLETE"}
```

# The step()

```
def step_push(step_ctx: StepContext, incident_id: str, phase: str) -> None:  
    _push(incident_id, phase)  
  
context.step(step_push(incident_id, "triage"))
```

# The parallel()

```
def weather(ctx):  
    return ctx.invoke(arns["weather"], name="weather")  
  
def population(ctx):  
    return ctx.invoke(arns["population"], name="population")  
  
results = context.parallel([weather, population])  
w, p = results.get_results()
```

# The waitForCallback()

```
def notify(callback_id: str, ctx):  
    _push(incident_id, "clarify", callbackId=callback_id)  
  
result = context.wait_for_callback(  
    submitter=notify,  
    name="clarify",  
)
```

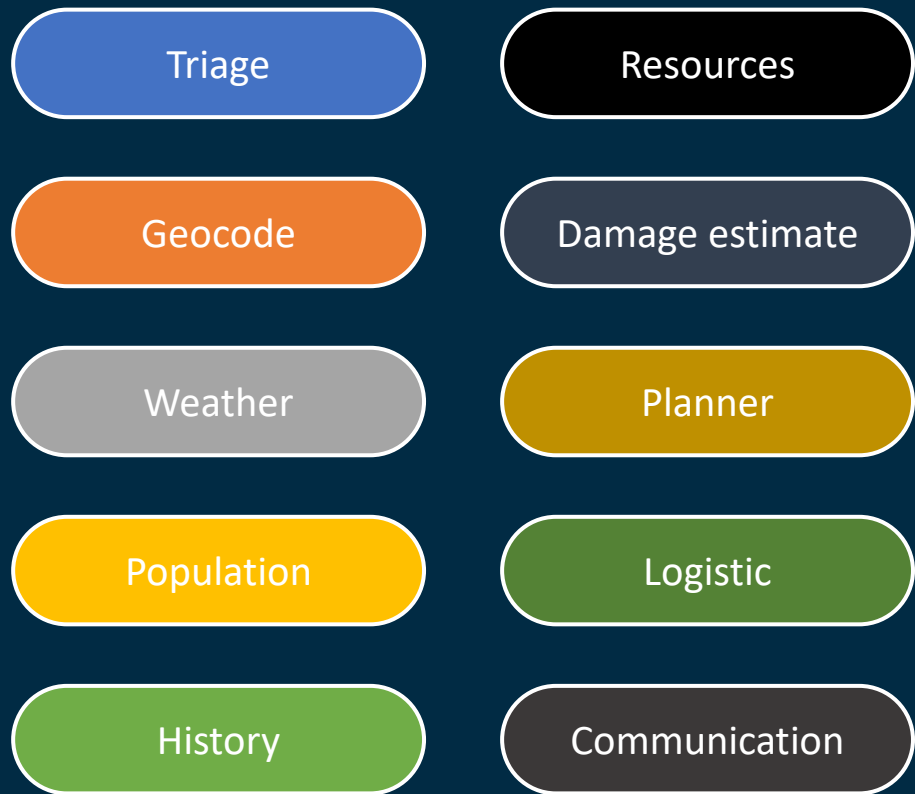
# Ald

AI driven disaster response planning

# The process



# Agent vs Tool



| Agents | Tools |
|--------|-------|
|        |       |

# Agent vs Tool

| Agents          | Tools      |
|-----------------|------------|
| Triage          | Geocode    |
| History         | Weather    |
| Damage estimate | Population |
| Planner         | Resources  |
| Logistic        |            |
| Communication   |            |

# Agent vs Tool

***Pro tip:*** Tools over agents whenever possible



| Agents          | Tools      |
|-----------------|------------|
| Triage          | Geocode    |
| History         | Weather    |
| Damage estimate | Population |
| Planner         | Resources  |
| Logistic        |            |
| Communication   |            |

# Easy or hard?

Triage

History

Damage estimate

Planner

Logistic

Communication

Easy

Hard

# Easy or hard?

| Easy            | Hard          |
|-----------------|---------------|
| Triage          | Planner       |
| History         | Logistic      |
| Damage estimate | Communication |

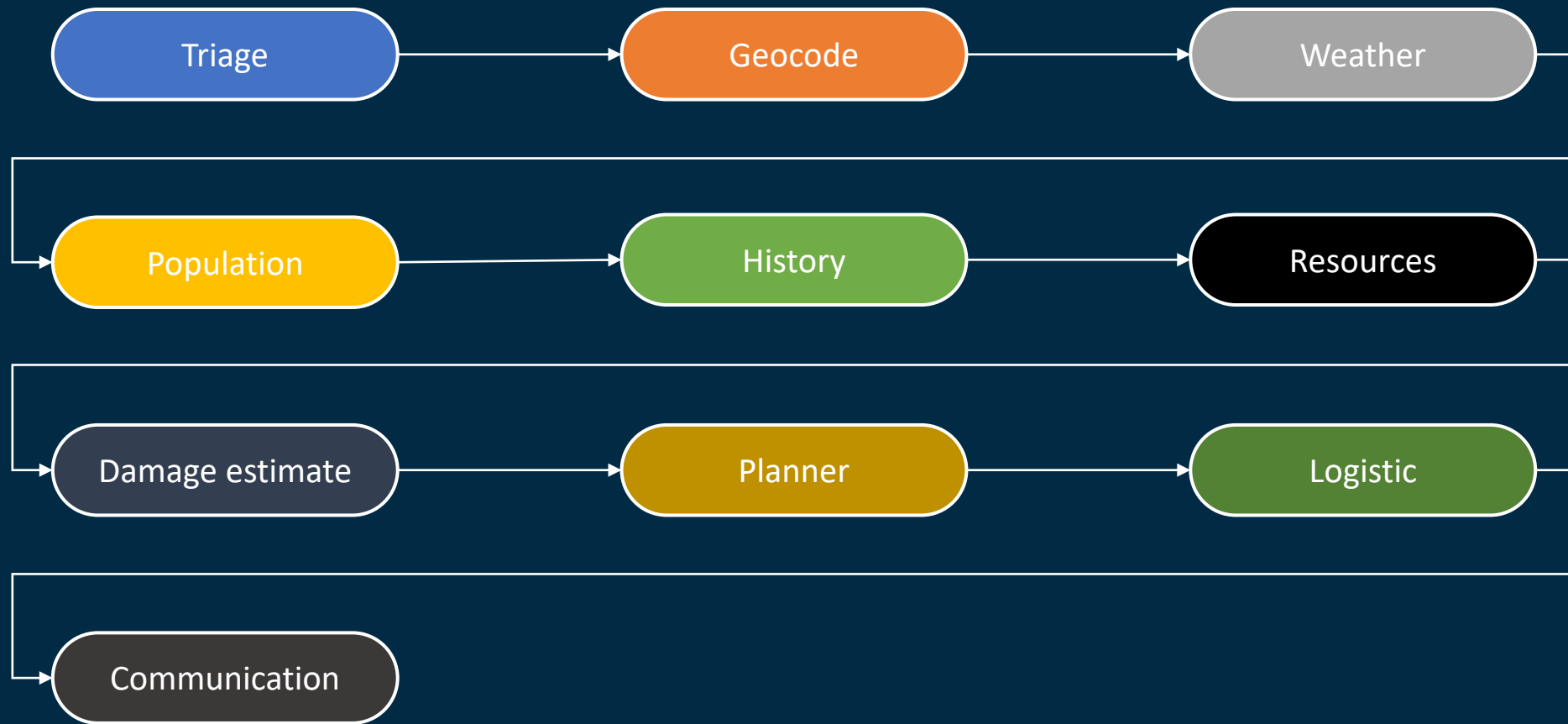
# Easy or hard?

***Pro tip:*** Don't  
overpower or  
overpay on your  
models

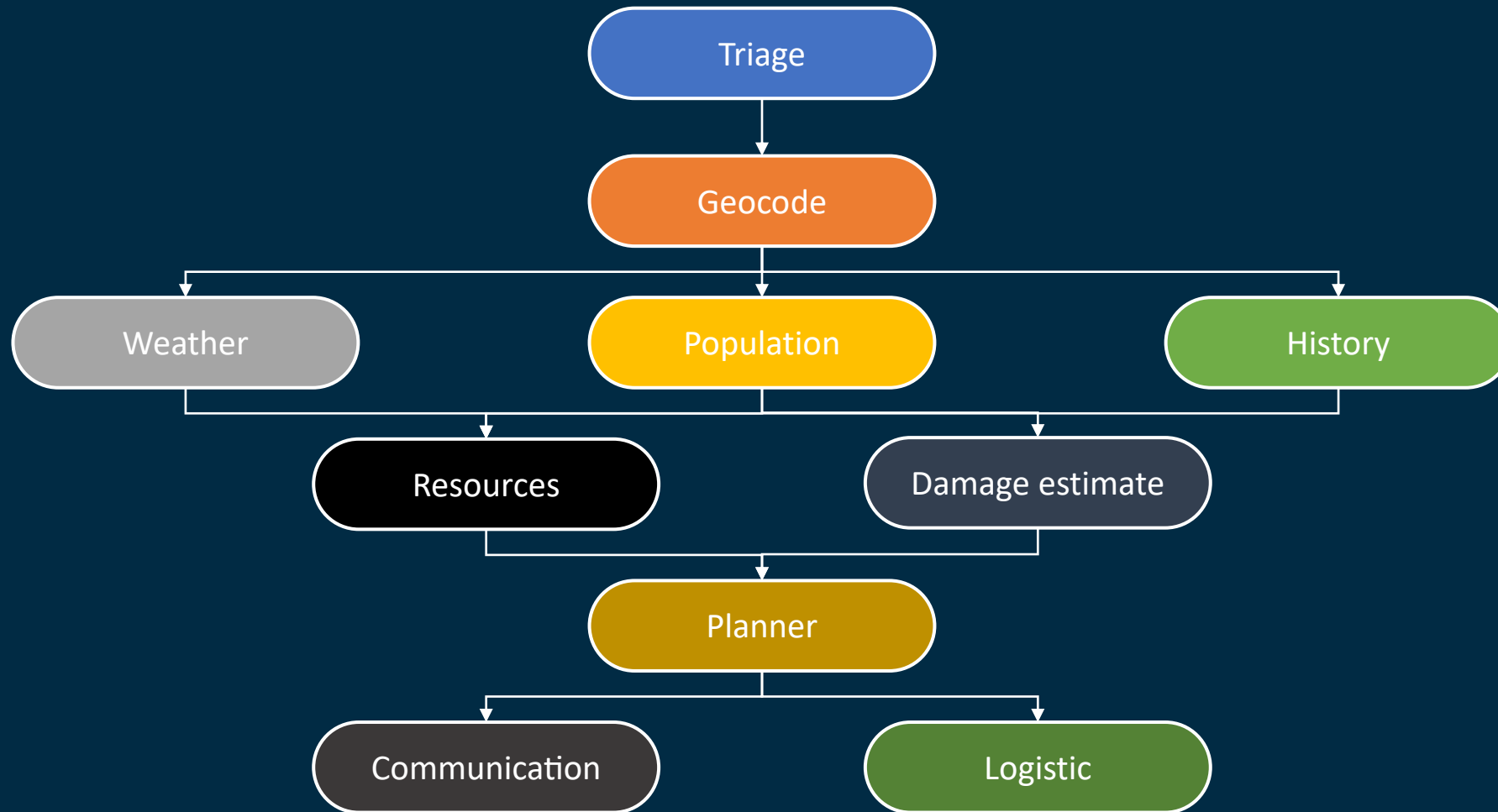


| Easy            | Hard          |
|-----------------|---------------|
| Triage          | Planner       |
| History         | Logistic      |
| Damage estimate | Communication |
| Nova Pro        | Sonnet 4.6    |

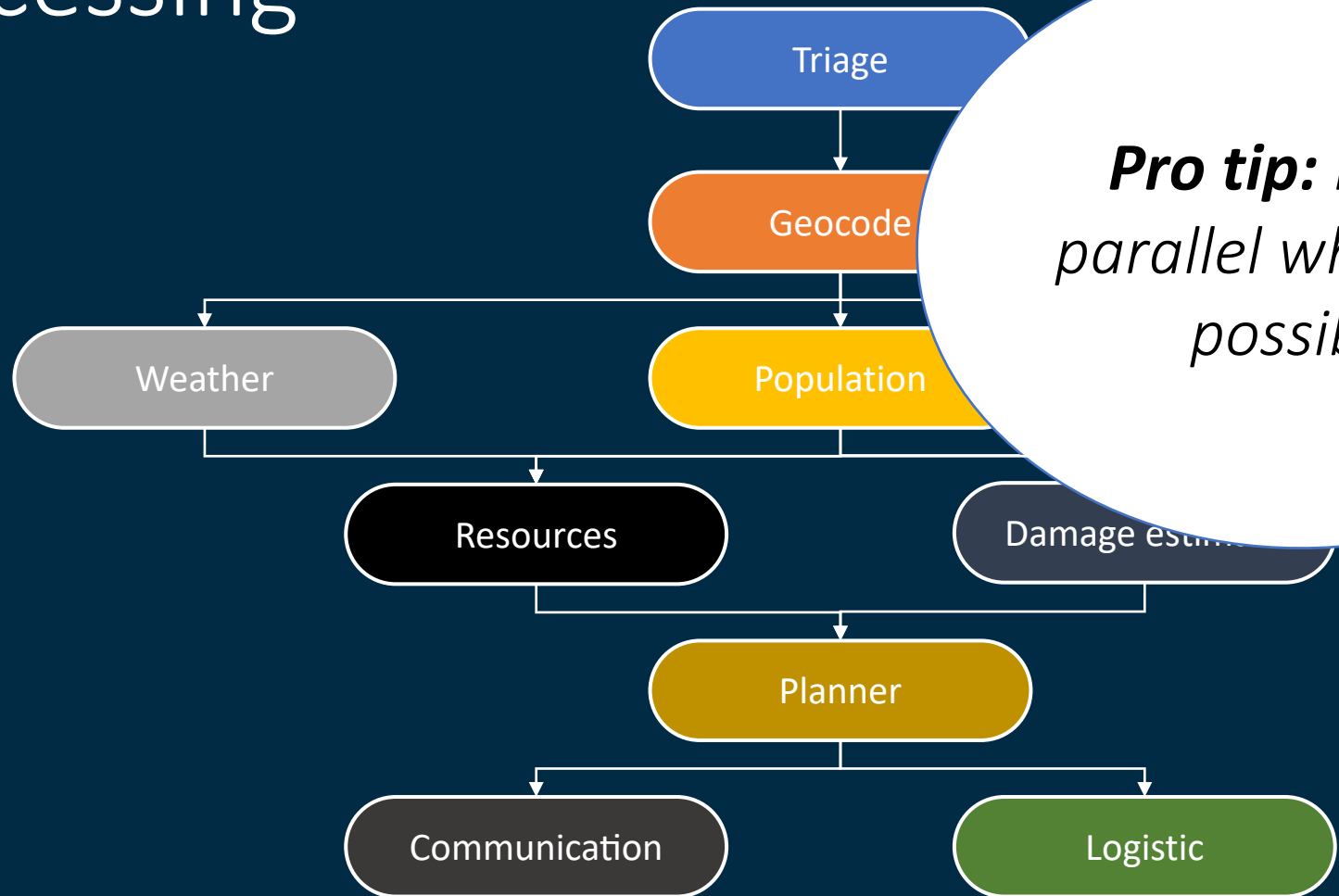
# The process in order ... kinda



# The process in order ... actually



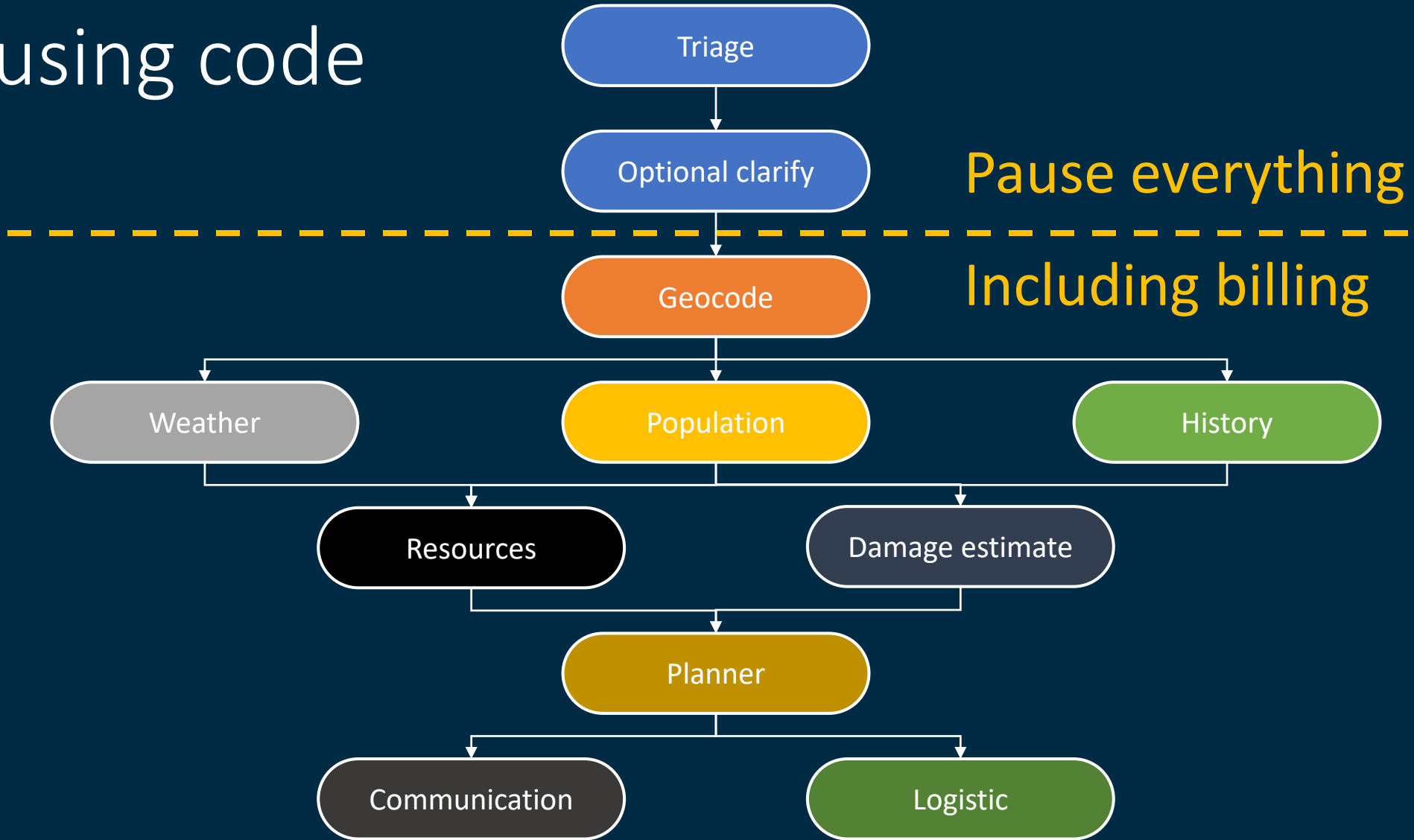
# Parallel processing



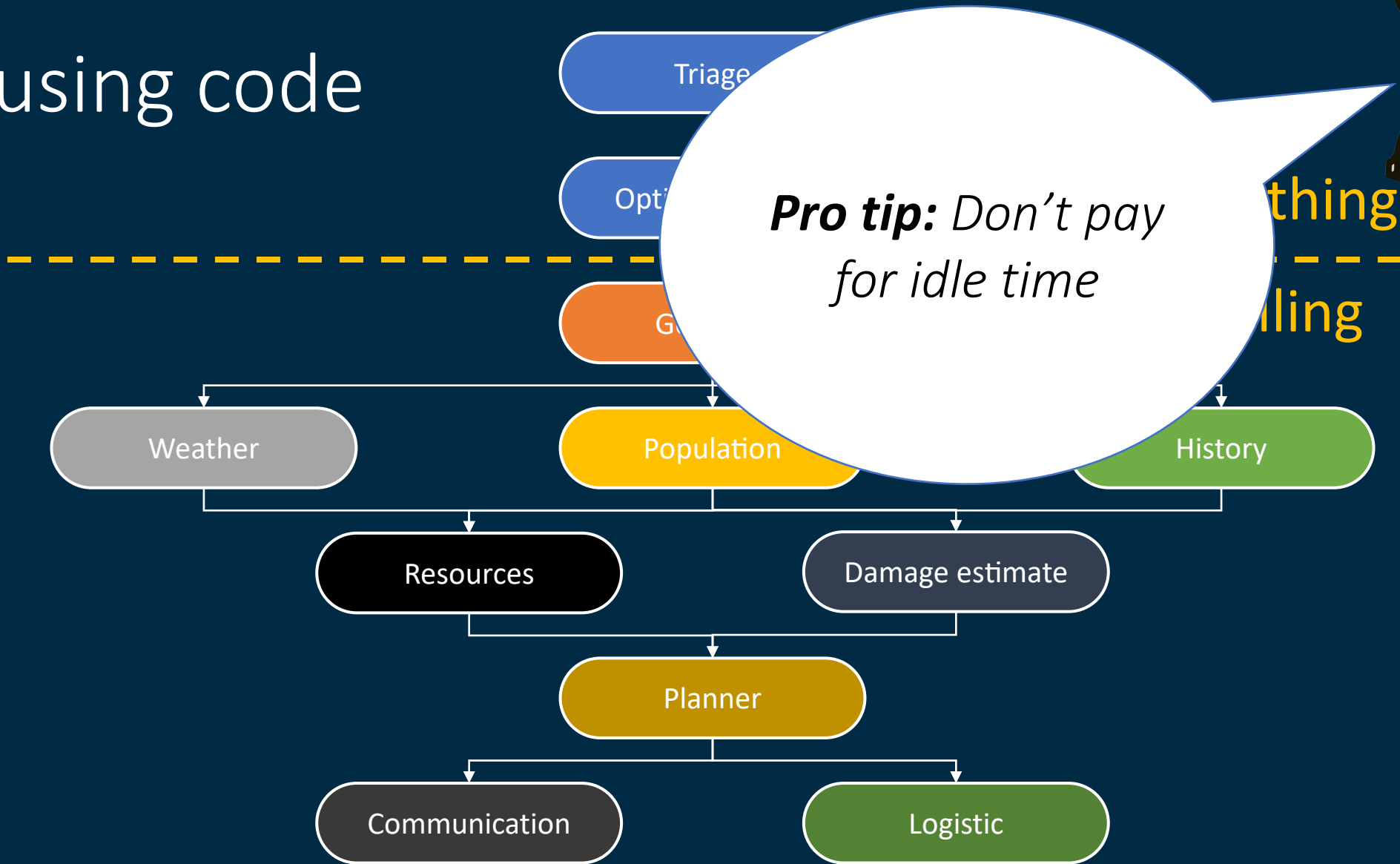
**Pro tip:** run in parallel whenever possible



# Pausing code

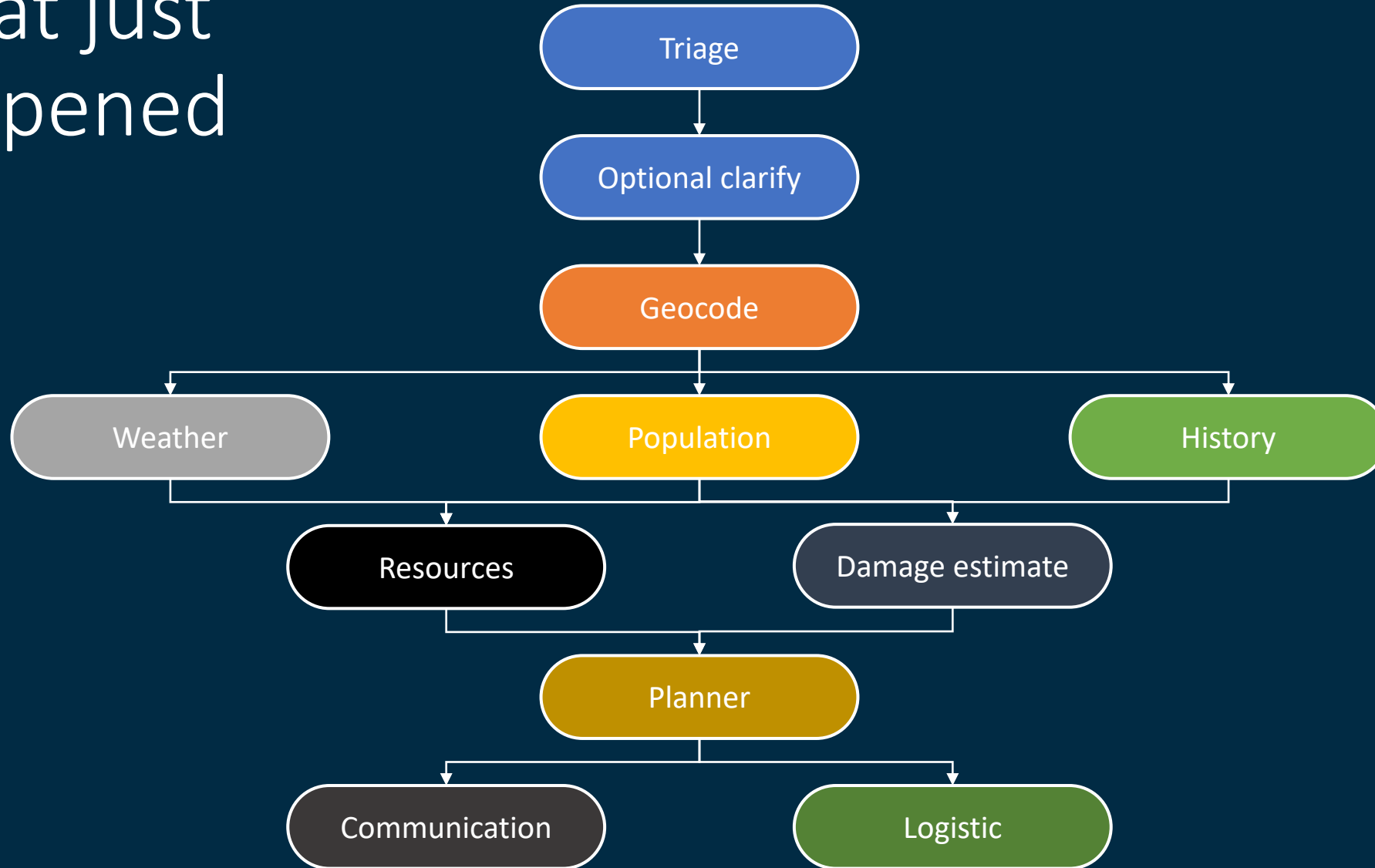


# Pausing code

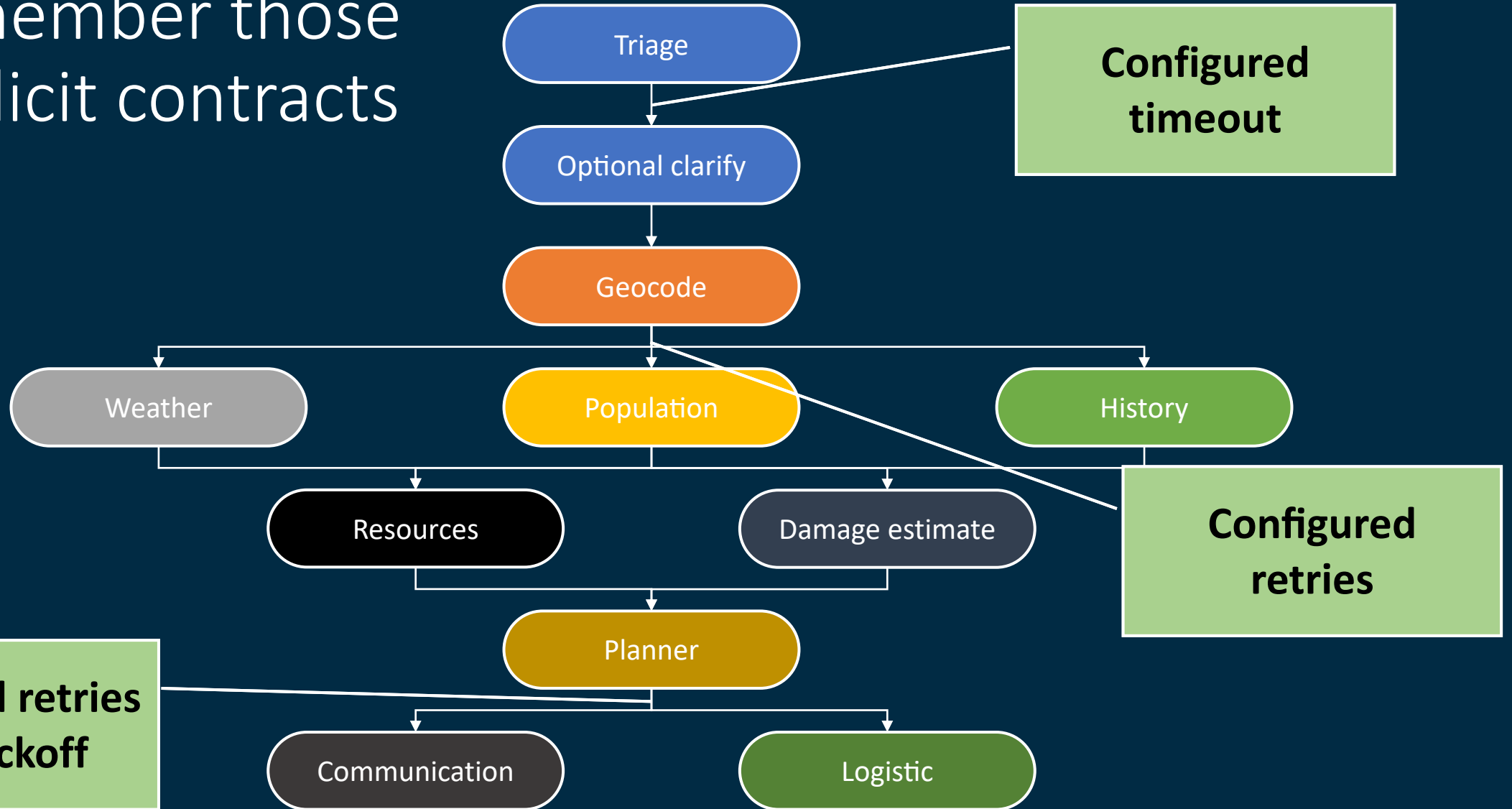


Let's see it in action

# What just happened



Remember those  
implicit contracts



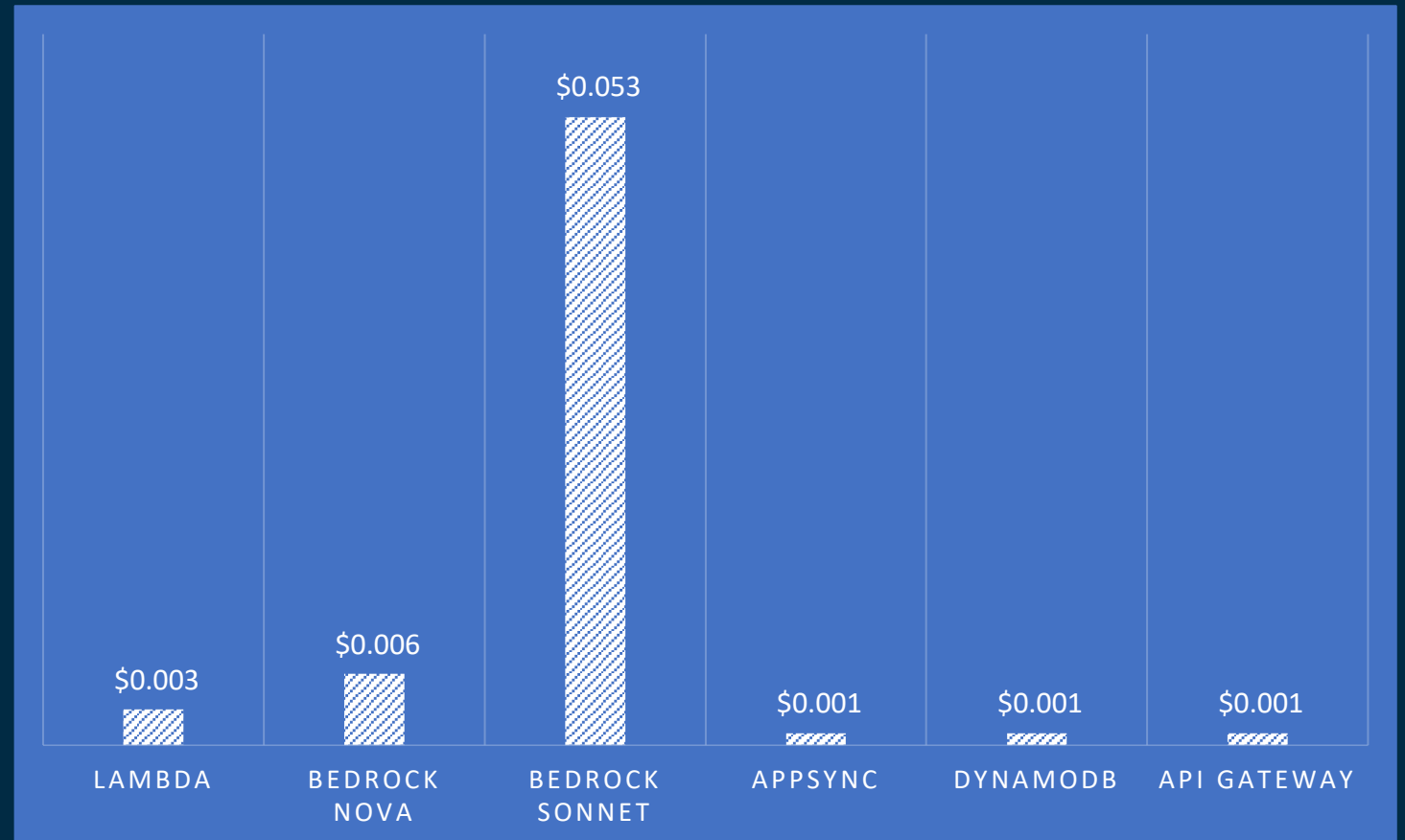
# The cost

## \$0.063 per incident

- 100 incidents/month = \$6.30
- 1,000 inc/month = \$63
- 10,000 inc/month = \$630

Swapping Planner/Logistics/Comms to Nova Pro would cut per-incident cost from \$0.063 to roughly \$0.015

At the expense of output quality.



# The wrap

# What this architecture gives you

- **Parallelism:** Independent agents ran concurrently, cutting wall-clock time
- **Recovery:** Any failure replays from the last checkpoint. No work is lost or duplicated. The system self-heals from exactly where it stopped.
- **Timeout Contracts:** Every wait is explicit, bounded, and handled by the infrastructure. No surprise hangs. No mystery timeouts buried in prompts.
- **Exactly-Once Execution:** Each step runs once and checkpoints its result. Retries and replays never duplicate side effects. Idempotency, by construction.

# Three things to walk away with

## Tool or agent?

Before every step, ask this question. If the answer is deterministic, write a function. **Spend non-determinism deliberately** — only where reasoning actually earns its keep.

## Design for twice

Design every step as if it will run twice. Idempotency isn't optional in a durable system. If your step has a side effect and you haven't thought about replay, **you haven't finished the design.**

## Name the contract

At every agent boundary consider timeout, format, and success definition. If it lives only in a prompt, it's implicit. **Move it to the orchestration layer.** These three questions catch the majority of failure modes before you ship.

# Resources

 **EDJGEEK**

HomeBlogAbout

***AGENTS GONNA AGENT: DETERMINISTIC ORCHESTRATION FOR SYSTEMS YOU CAN'T PREDICT***

**Demo Repo**

- **Ald — AI-Driven Disaster Response** — Full source for the live demo: Lambda Durable Function orchestrating 6 LLM agents and 4 deterministic tools into a real-time disaster response plan

**Cost to Run the Demo**

All serverless — you pay only per incident, nothing at rest.

| Service               | Per Incident |
|-----------------------|--------------|
| Lambda (11 functions) | \$0.003      |



[s12d.com/goto26](https://s12d.com/goto26)

**PLEASE RATE  
MY SESSION**

# *Thank you!*

@edjgeek

