The Worst Issue You Ever Dealt With

Laura Nolan
laura@requisitevariety.net

Or: How to Troubleshoot the Worst Issue of your Career

About Laura Nolan

- I keep bees (which are complex systems!) 🐝
- Contributor to *Site Reliability Engineering: How Google Runs Production Systems* (‘the SRE book’), *Seeking SRE*, *97 Things Every SRE Should Know,* InfoQ, and (most importantly) USENIX ;login: 📚
- Currently working independently and almost finished my MSc in Human Factors and Systems Safety at Lund University
- laura@requisitevariety.net, https://www.linkedin.com/in/lauralifts/

The New York Times

Slack Restores Service After Starting 2021 With Outage

The company apologized for the disruptions, which lasted for a few hours as users returned to work after the holidays.

Share full article 19

Blog post: bit.ly/ny-outage

Slack’s Outage on January 4th 2021

Slack’s Outage on January 4th 2021

Laura Nolan

Incidents and Issues

Incidents:

- Pressure to fix ASAP
- Usually lots of people collaborating
- No distractions
- Mitigations can often get us back on the air, no need to understand causal mechanism

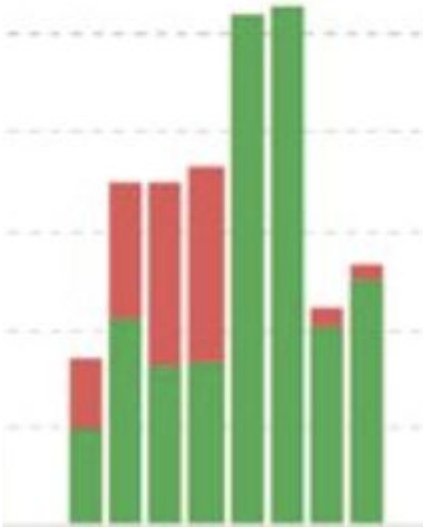
Issues:

- Time pressure is less
- Usually a smaller team or individual responsible for investigating
- Lots of distractions
- Understanding causal mechanism is critical to fixing or preventing recurrence

Troubleshooting versus Debugging

Troubleshooting Scenarios

- Something is broken or misbehaving
- Something is slow
- Something is *intermittently* broken
- Something is *intermittently* slow
- Some *subset* of requests (or other work) are slow or broken
- Unexplained high resource usage
- Some kind of other unexplained behaviour

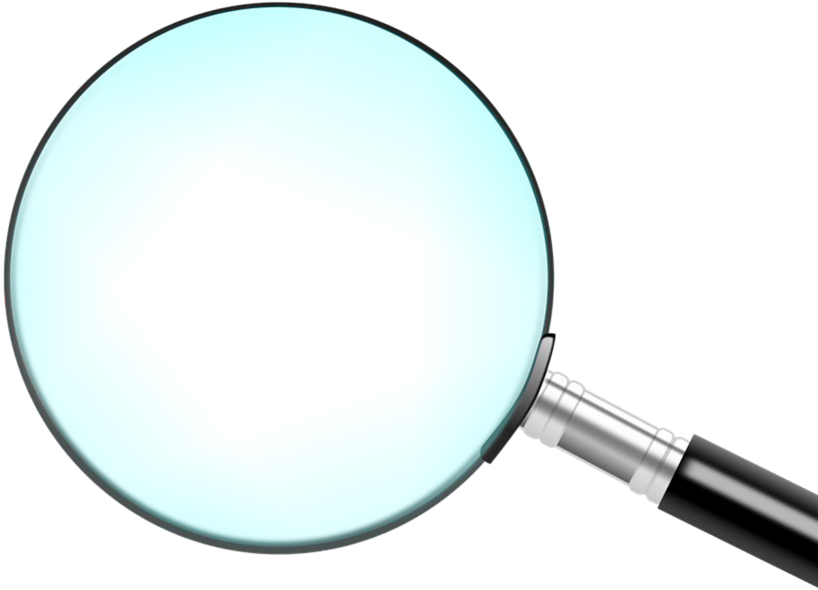


A troubleshooting story

- Small organisation, with a two-person DevOps team and several devs
- Software in Java and JS, running on a cloud provider
- Customers start to notice performance problems and report that the API seems to be browning out regularly

A troubleshooting story: DevOps investigate

- At this point the DevOps team are assigned to deal with the issue
- They check some consoles and find that the API service has been going unhealthy and being restarted after failing healthchecks, most days around peak time
- This has started a couple of months previously and was ongoing

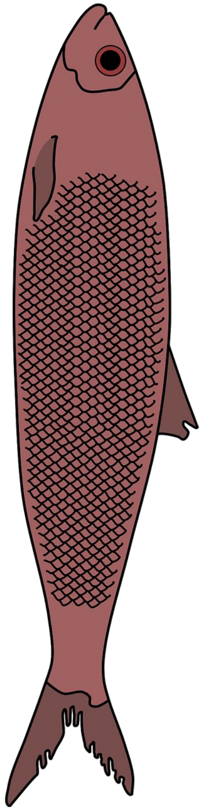


A troubleshooting story: it must be the DB!

- Service doesn’t have much in the way of metrics or tracing, none of these fancy newfangled observability technologies
- It has logs
- The logs don’t show anything interesting though
- But the only dependency this service has is the DB, so it must be slow DB, surely?

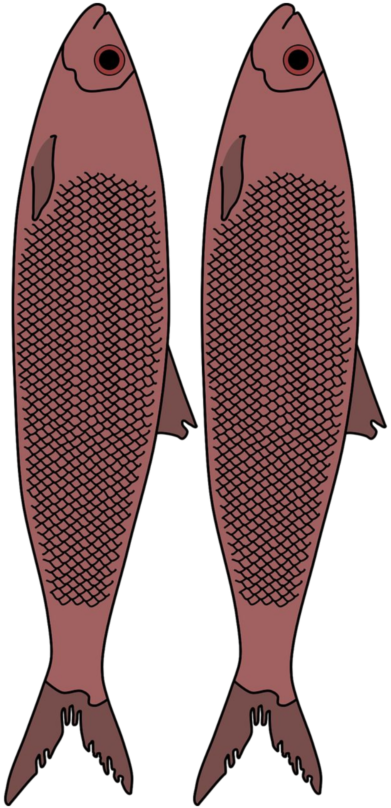
A troubleshooting story: DB red herring

- There was already a lot of concern around the DB in the org, so it was top of mind to improve DB performance
- DB consultant hired to optimise queries and indexes
- DB instances upscaled
- … and no improvement at all
- DB metrics all look very quiet, no evidence of DB performance problems there



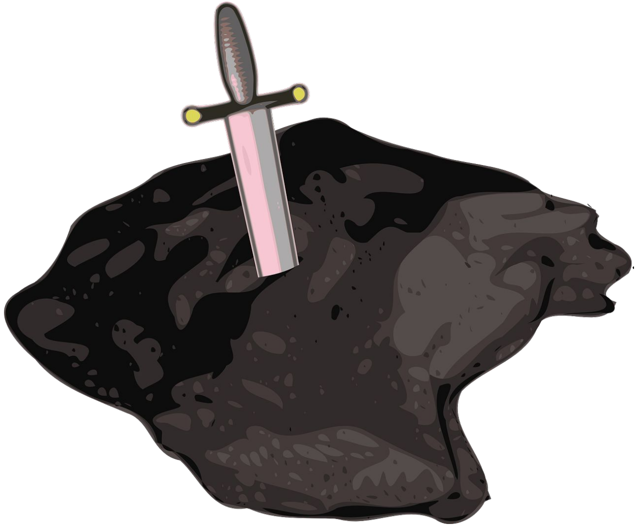
A troubleshooting story: another red herring

- Since the DB story has not proven fruitful, the next thing was to look at any changes that happened around time problem manifested
- Turns out around this time the app was moved from running directly on VM to a container service
- Maybe this is the problem! (although nobody sees how)
- But after 3 months, it is hard to revert back
- Should we invest the work in reverting, or work more on the DB query optimisation, or work on adding more observability, or something else?



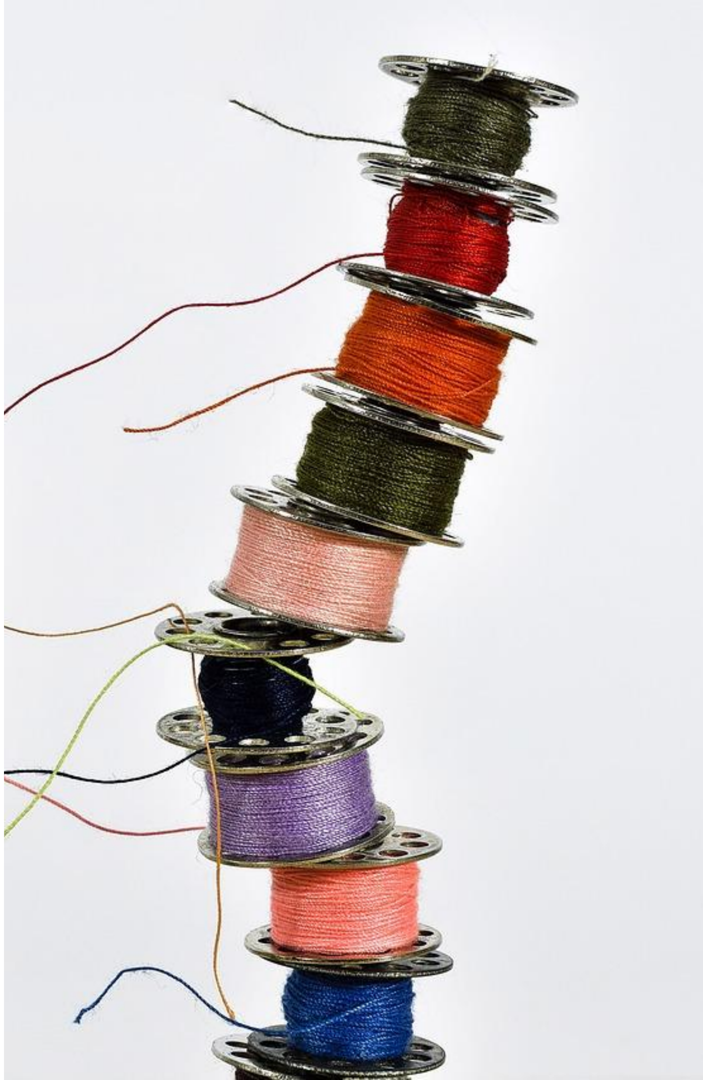
This team was stuck

- They do not know how to move forward effectively
- They have some theories about possible causes, but not really any good evidence for any of them
- All they really know at this point is that their API service is freezing periodically and being restarted, and that this is causing customer pain
- Issue is owned by DevOps as owners of last resort, and they are not application engineers



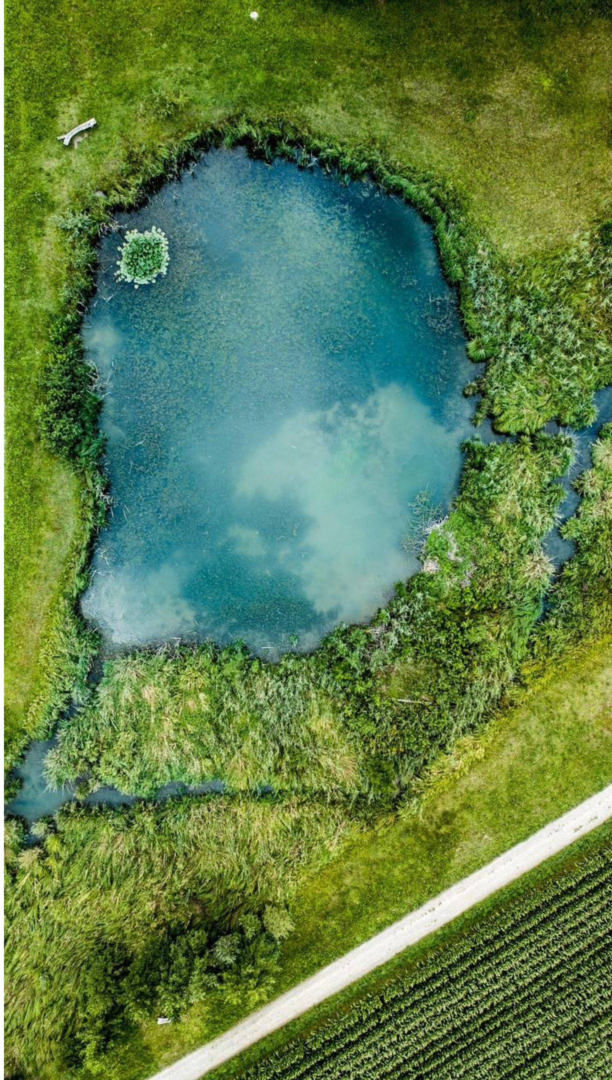
Getting unstuck

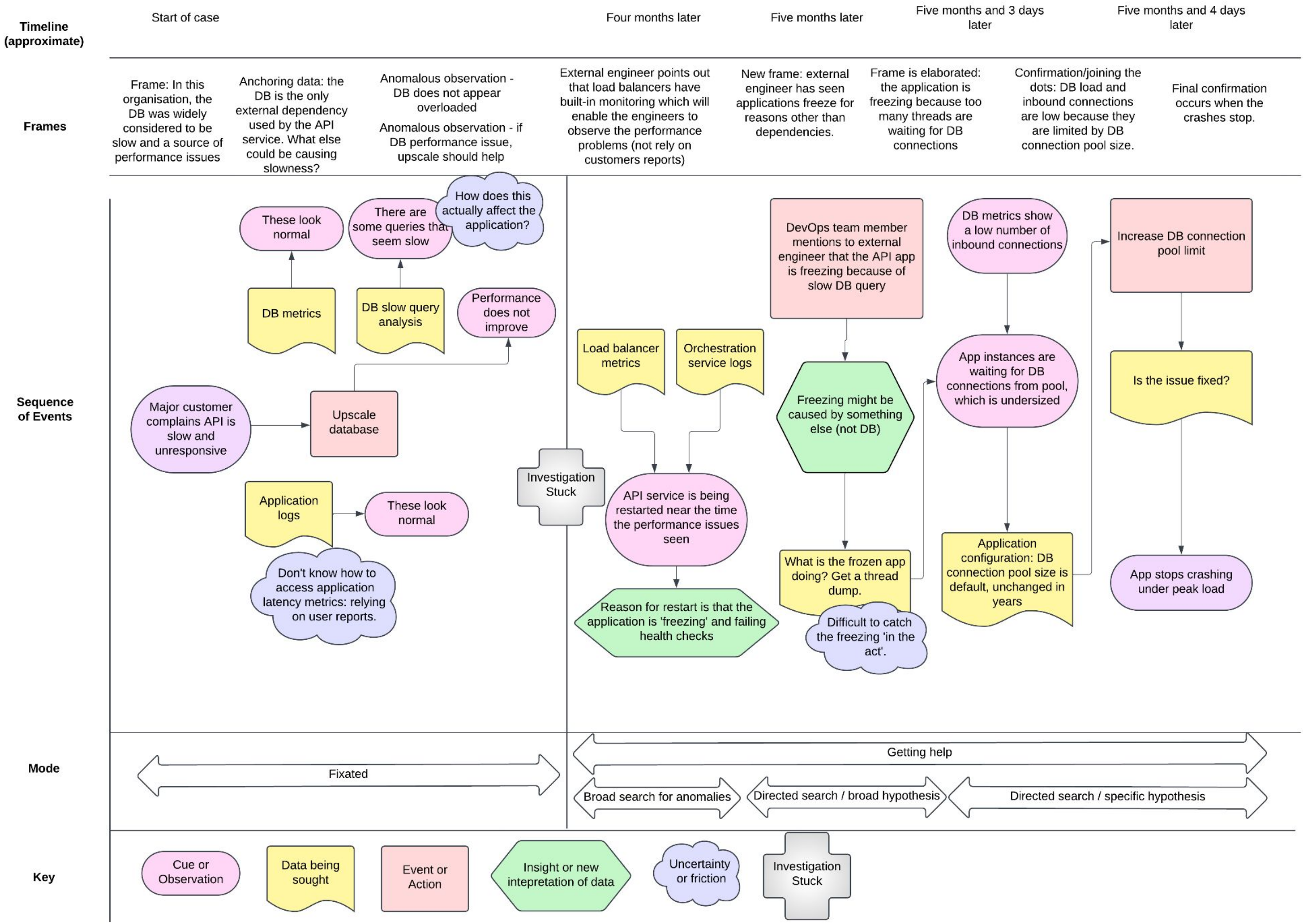
- Another external engineer joins the effort
- Doesn’t know anything about the specific service but has done Java performance work
- Thinks that DB slowness is unlikely to cause an application to totally lock up this way
- Advises to get Java thread dumps next time the application freezes



Getting unstuck

- DevOps engineer reads thread dump and is confused - thinks it shows the application has run out of threads, due to lack of awareness of connection pooling
- External engineer reads it and interprets it as application running out of DB connections due to connection pool configuration
- Increasing the connection pool count fixes the problem immediately





Timeline (approximate)

Start of case

Four months later

Five months later

Five months and 3 days later

Five months and 4 days later

Frames

Frame: In this organisation, the DB was widely considered to be slow and a source of performance issues

Anchoring data: the DB is the only external dependency used by the API service. What else could be causing slowness?

Anomalous observation - DB does not appear overloaded

Anomalous observation - if DB performance issue, upscale should help

External engineer points out that load balancers have built-in monitoring which will enable the engineers to observe the performance problems (not rely on customers reports)

New frame: external engineer has seen applications freeze for reasons other than dependencies.

Frame is elaborated: the application is freezing because too many threads are waiting for DB connections

Confirmation/joining the dots: DB load and inbound connections are low because they are limited by DB connection pool size.

Final confirmation occurs when the crashes stop.

Sequence of Events

Major customer complains API is slow and unresponsive

Upscale database

Performance does not improve

DB metrics

These look normal

DB slow query analysis

There are some queries that seem slow

How does this actually affect the application?

Application logs

These look normal

Don't know how to access application latency metrics: relying on user reports.

Investigation Stuck

Load balancer metrics

Orchestration service logs

API service is being restarted near the time the performance issues seen

Reason for restart is that the application is 'freezing' and failing health checks

DevOps team member mentions to external engineer that the API app is freezing because of slow DB query

Freezing might be caused by something else (not DB)

What is the frozen app doing? Get a thread dump.

Difficult to catch the freezing 'in the act'.

DB metrics show a low number of inbound connections

App instances are waiting for DB connections from pool, which is undersized

Application configuration: DB connection pool size is default, unchanged in years

Increase DB connection pool limit

Is the issue fixed?

App stops crashing under peak load

Mode

Fixated

Getting help

Broad search for anomalies

Directed search / broad hypothesis

Directed search / specific hypothesis

Key

Cue or Observation

Data being sought

Event or Action

Insight or new inpretation of data

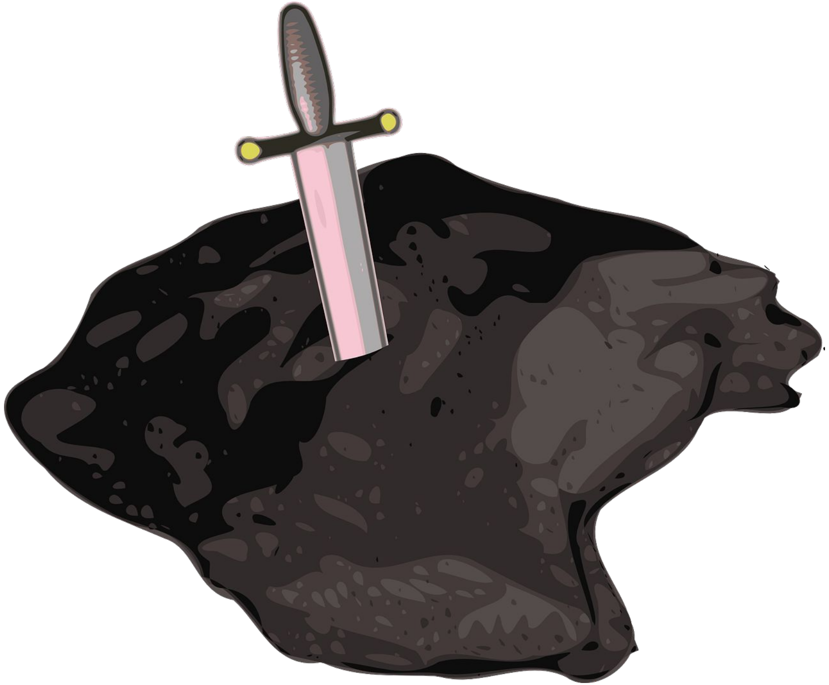
Uncertainty or friction

Investigation Stuck

What can we learn from this case?

- Expertise seems to matter: sometimes an expert can unravel a case rapidly that has stumped others for a long period of time
- It is easy to spend a lot of time on red herrings
- Troubleshooting cases can involve a lot of uncertainty about where the problems are
- Experience can help to highlight where problems are most likely to be
- You often have to *actively* seek out data you need, it may not be available in your logs and metrics
- To interpret the data effectively you need to have a mental model of the system
- You only know for sure you are right when the problem goes away!

Why is troubleshooting in digital systems challenging?



Intransparency

- We can’t observe internals of digital systems directly
- We rely on what logging, metrics, tracing are there
    - If we control the build then we can add more, but that takes time
- We can do profiling and packet tracing and other forms of introspection
    - But sometimes access can be tricky
    - May need to enable debug ports etc
    - Sometimes healthchecks can cause the broken instance you are trying to introspect to be killed
    - Lots of friction generally
- Understanding what we are seeing can often require system knowledge that we may not have

Logs are a double-edged sword

- The industry still relies very heavily on logs
- Most accessible form of observability: anyone can read lines of text
- Works great for many cases, too
- But logs are a poor fit for understanding performance problems
- Log levels can cause issues as well - information hiding
- Log volume can be overwhelming
- Logs are messy and noisy and lots of things in there may also be red herrings
- Not everything we care about may be in the logs

Management pointing in the wrong direction

- In some incidents I see management directing engineers to investigate particular avenues
- This can waste a lot of time if those are the wrong directions
- But engineers feel obliged to do that work, even when they don’t think the direction is the most likely cause of the issue

Uncertainty

- By its nature, troubleshooting involves a lot of uncertainty
    - What is the problem
    - How much impact is there
    - When will it be resolved
    - How much work will it take to resolve it
- Nobody likes uncertainty
- But effective troubleshooting requires tolerating uncertainty, and keeping an open mind towards alternative possibilities
    - Potential causes of observed behaviours
    - Potential ways of confirming or disconfirming different hypotheses
    - Potential effects of any actions that we take

You don't know what you don't know

- Many long-running incidents or problems involve elements of the system that we don't even know are there
  - In this case, the DB connection pool
- This is related to intransparency - we don't know about these things because we can't see them… until they break

Configuration is a pain point

- Configuration issues appear quite often in troubleshooting
- Major factor in 6 of my 14 difficult research cases

Configuration is intransparent

- Configuration often affects things in ways that are difficult to observe, as they aren’t in the direct path of execution - again, intransparency
  - Think of things like concurrency and preemption related config, networking config
  - Configuration issues are rarely solvable with logs

Configuration ownership and visibility

- Config is often ‘off to the side’, especially OS or application config
    - Rarely modified or considered
    - Often not really ‘owned’ by either dev or ops

Configuration defaults

- When the defaults ‘just work’ for a long time, we get a surprise when they don’t
- Default configs make discovery harder
- Config documentation often isn’t great
- Good practice: add a /config endpoint to your binaries to show running configs (not including secrets)

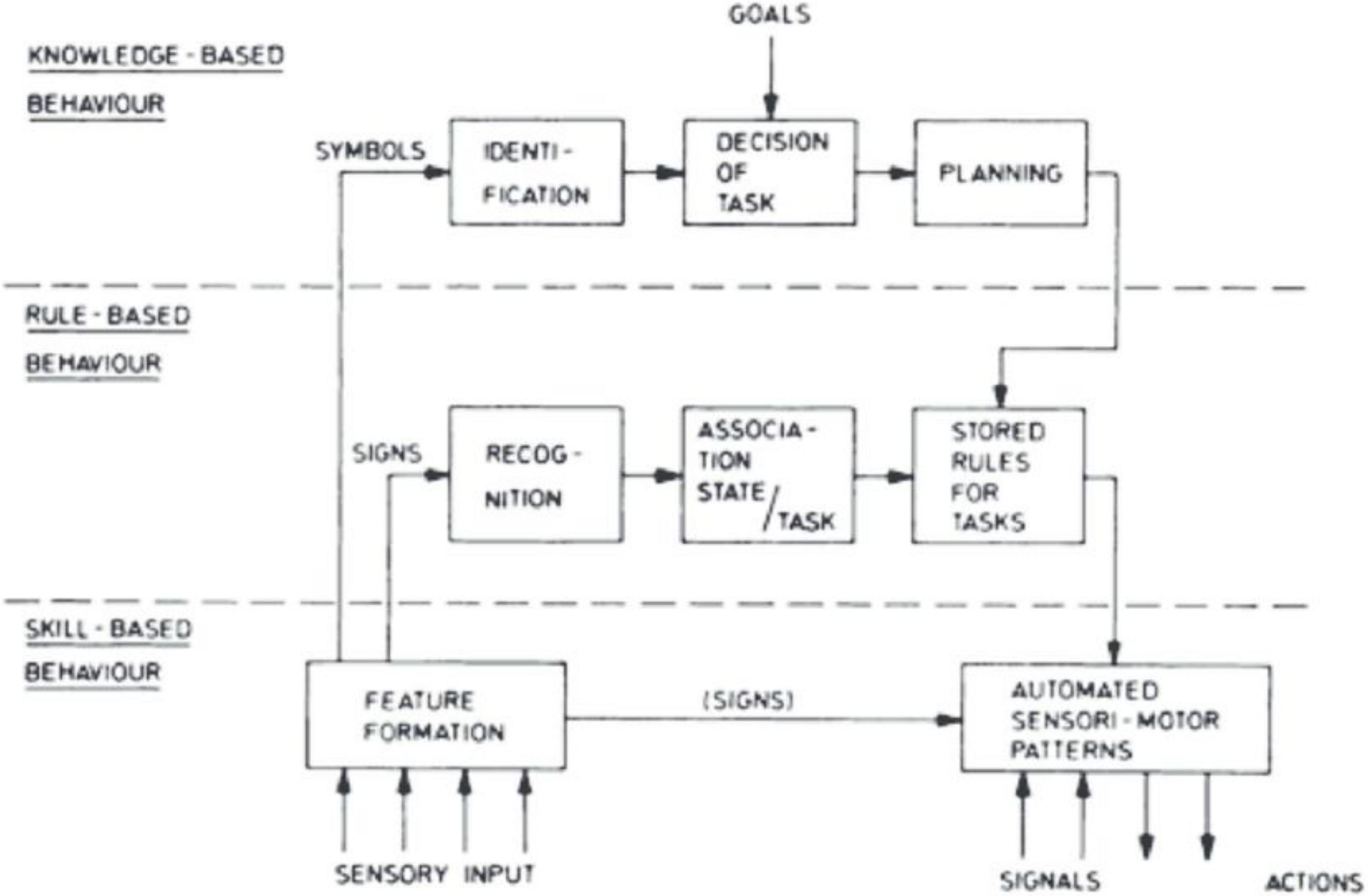
DevOps team are not always best placed

- Teams with titles like DevOps and SRE are often tasked with troubleshooting as the default owner of whatever is broken
- The implicit assumption is that people with these job titles are always good at troubleshooting
    - This was more true ten years ago when these job titles were usually held by people who had been sysadmins for their university internet societies in the early noughties
    - Now, many people come to the profession as cloud admins or CI/CD specialist, and it is a different skillset
    - Software engineers aren’t necessarily good at troubleshooting either, particularly if the problem isn’t in their code
- Many organisations seem to have a few go-to troubleshooters; senior people who have seen a lot and are comfortable with chasing issues across system boundaries

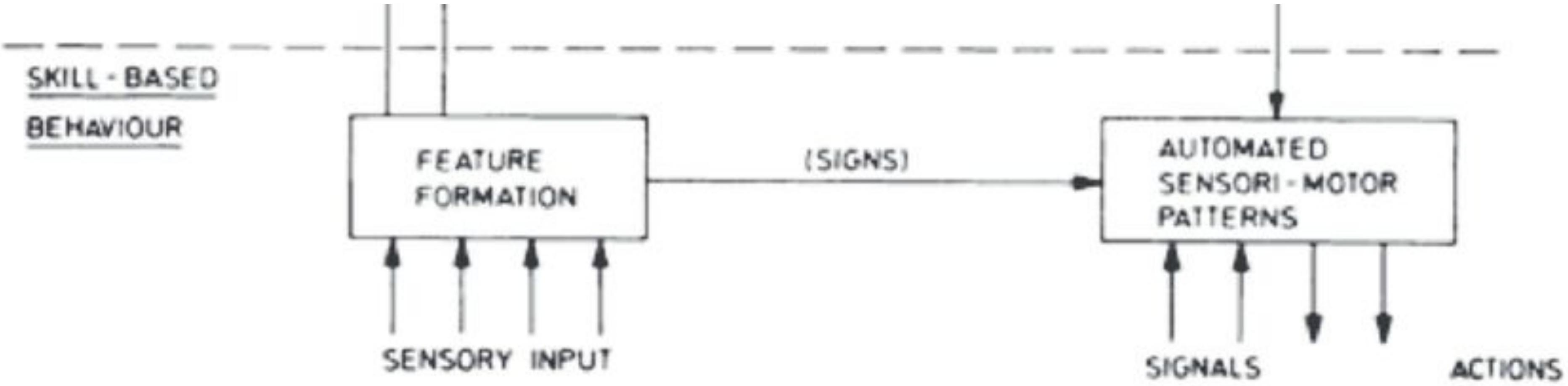
Determining causality

- The observability tools that we have tend to be geared towards showing us the state of the system
- Figuring out how the system got into that state may require more detective work
- Going from ‘what’ to how often requires engineers to synthesize observations with what they know about how systems work - which is hard!

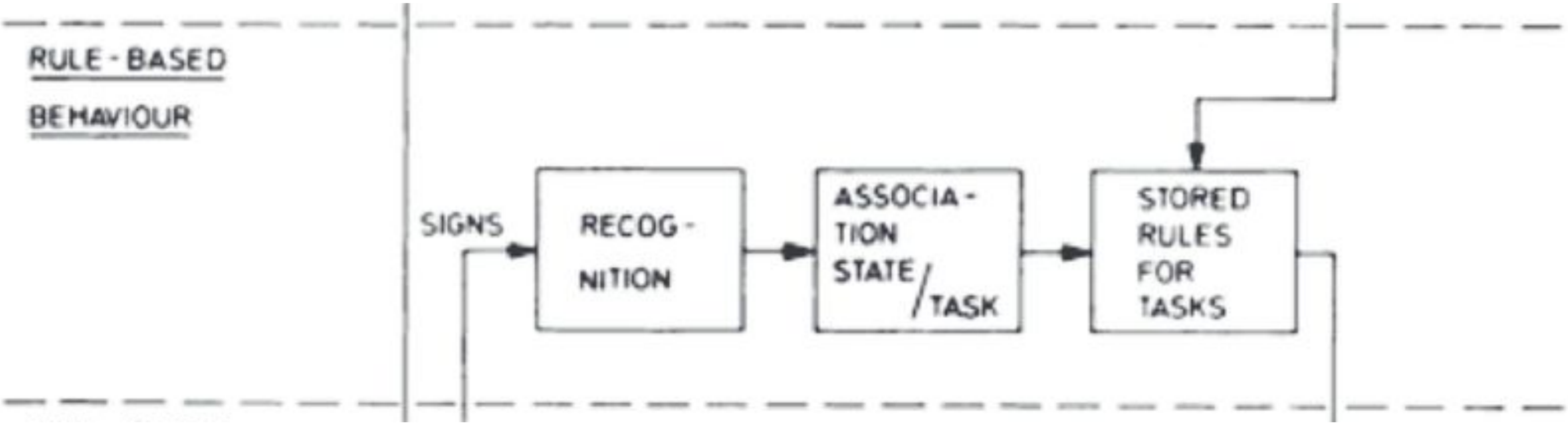
What does the research say about troubleshooting?


GOALS
KNOWLEDGE - BASED BEHAVIOUR
SYMBOLS
IDENTIFICATION
DECISION OF TASK
PLANNING
RULE - BASED BEHAVIOUR
SIGNS
RECOGNITION
ASSOCIATION STATE / TASK
STORED RULES FOR TASKS
SKILL - BASED BEHAVIOUR
FEATURE FORMATION
(SIGNS)
AUTOMATED SENSORI - MOTOR PATTERNS
SENSORY INPUT
SIGNALS
ACTIONS

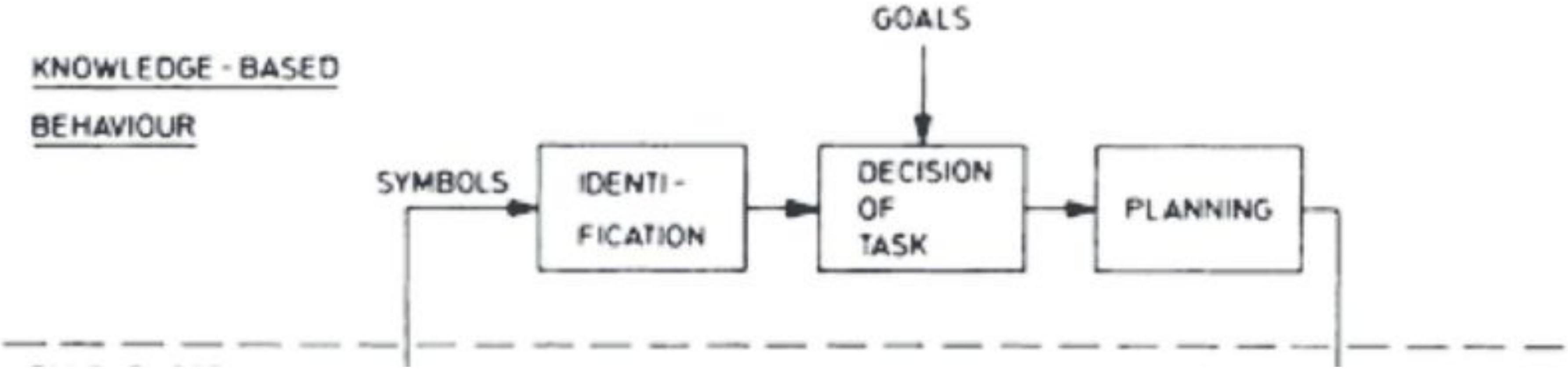
Skill


SKILL - BASED
BEHAVIOUR
FEATURE
FORMATION
(SIGNS)
AUTOMATED
SENSORI - MOTOR
PATTERNS
SENSORY INPUT
SIGNALS
ACTIONS

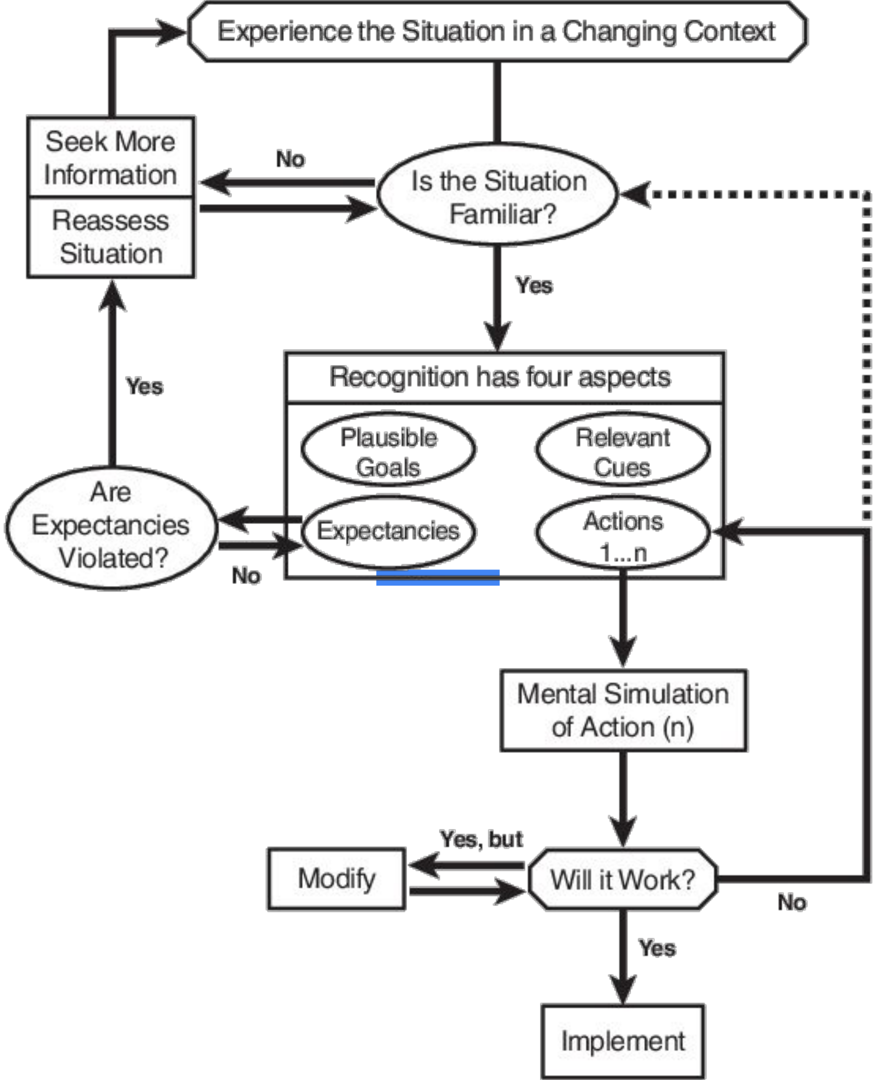
Rules


RULE-BASED
BEHAVIOUR
SIGNS
RECOG-NITION
ASSOCIATION STATE/TASK
STORED RULES FOR TASKS

Knowledge


KNOWLEDGE - BASED
BEHAVIOUR
GOALS
SYMBOLS
IDENTI - FICATION
DECISION OF TASK
PLANNING

DEPUTY CHIEF
F.D.N.Y.



Experience the Situation in a Changing Context

Is the Situation Familiar?

No

Seek More Information

Reassess Situation

Yes

Recognition has four aspects

Plausible Goals

Relevant Cues

Expectancies

Actions 1...n

Are Expectancies Violated?

Yes

No

Mental Simulation of Action (n)

Will it Work?

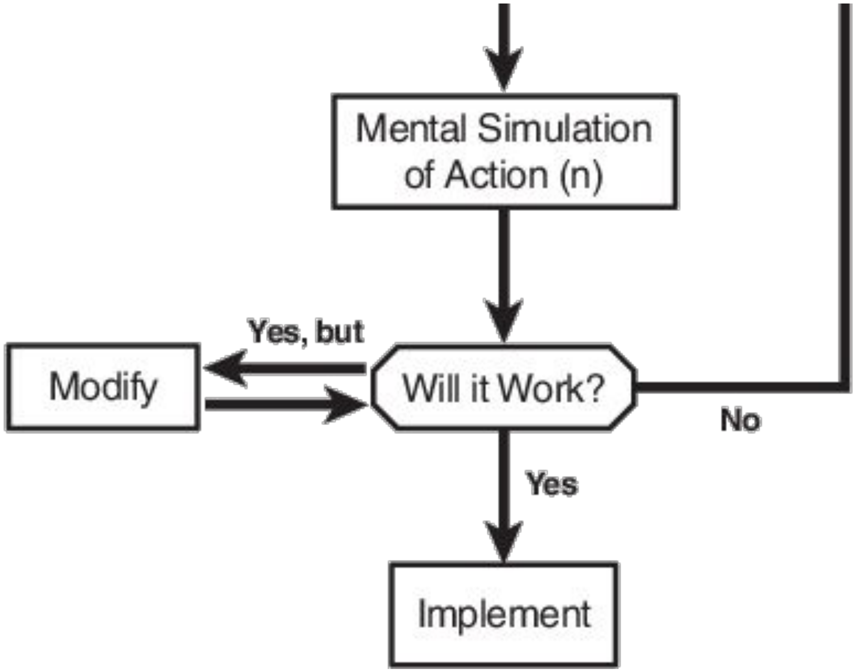
Yes, but

Modify

No

Yes

Implement



Mental Simulation

Limits on mental simulation

- Around three ‘moving parts’
- Maximum of six steps
- But experts can cheat using ‘chunking’

Experts process information differently

Klein's Data Frame Theory of Sensemaking

Sensemaking is an active process of understanding a situation. It is:

- The account we generate to explain events
- How we elaborate that account of events
- Questioning our current account of events when we find inconsistent data
- Fixation on an account
- Comparing alternative accounts of events

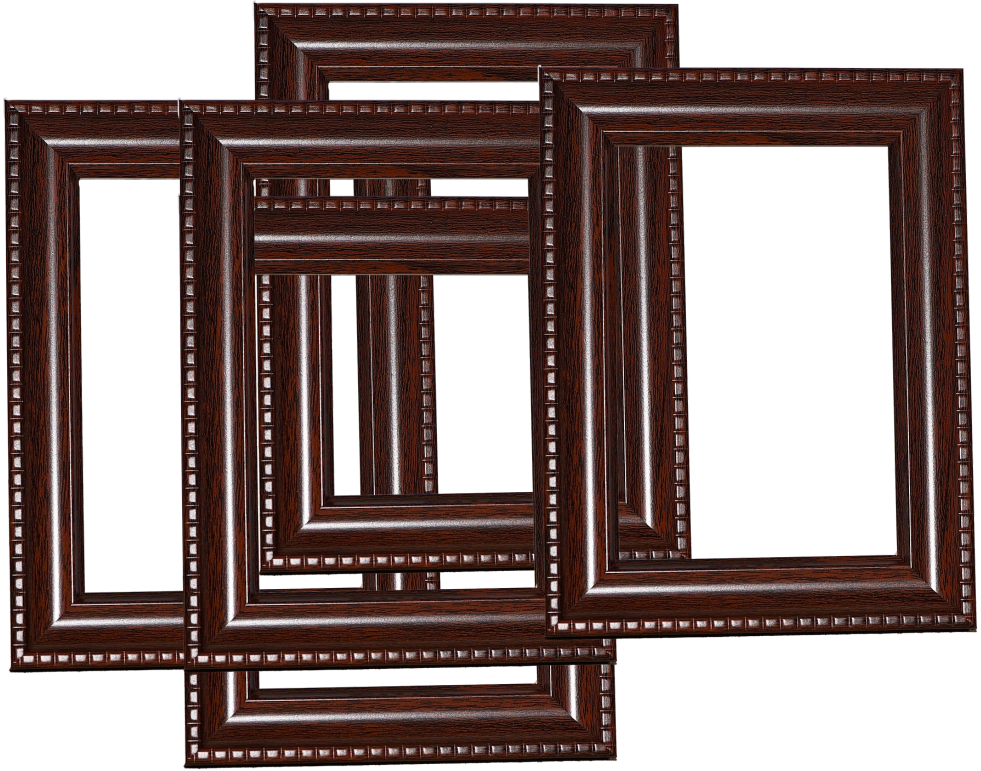
What is a Data Frame?

Think of them as stories, maps, plans.

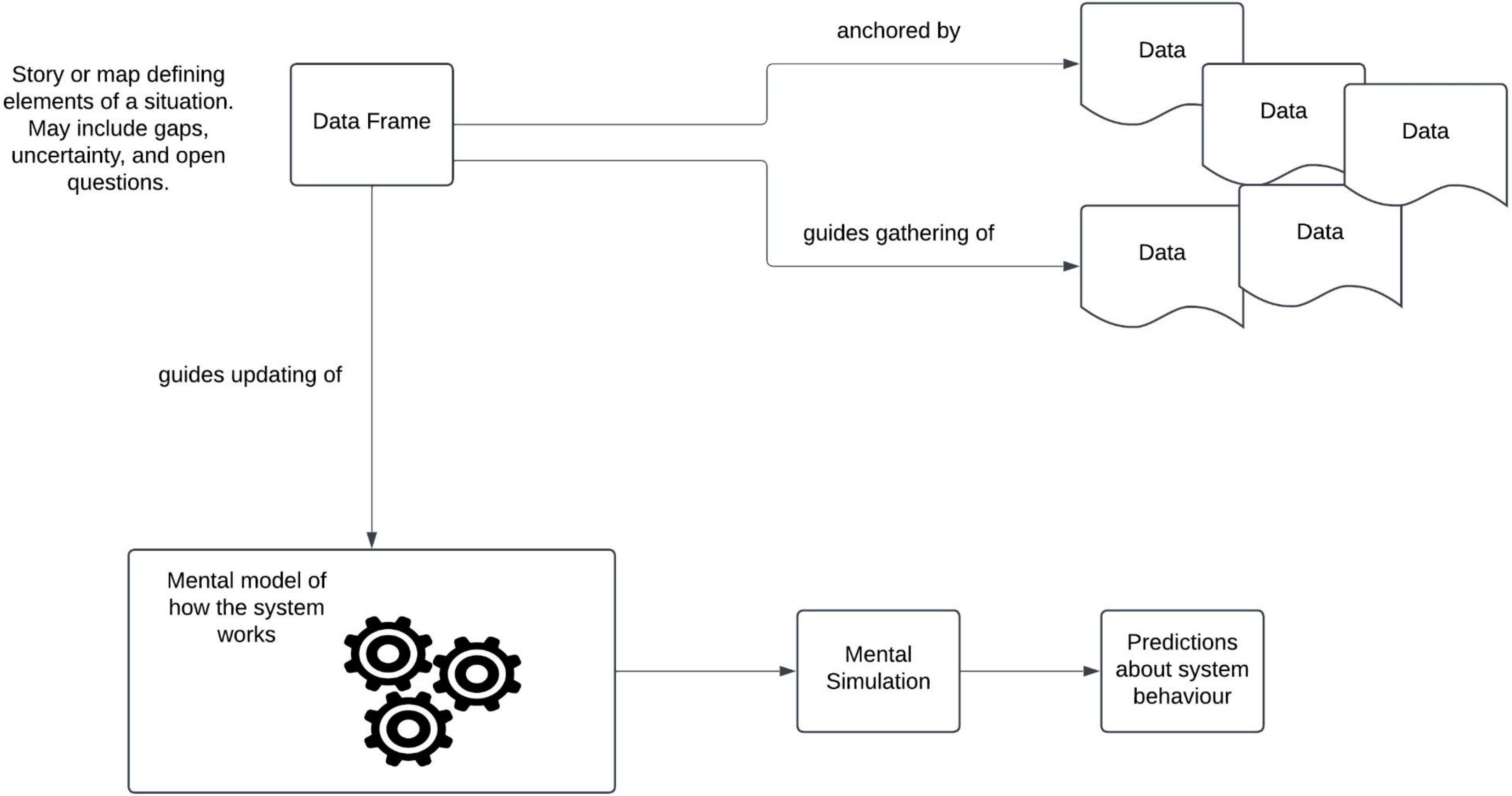
Frames relate elements to other elements. They structure the data we have about a situation, and they guide the search for more information.

They are normally ‘anchored’ on a small set of important data.

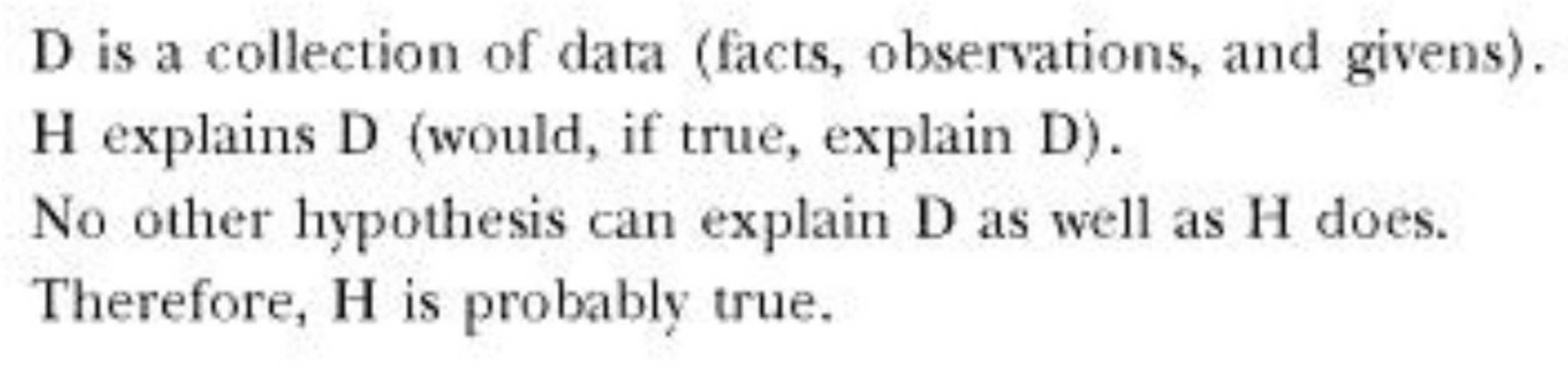
Experts have a richer set of frames



Sensemaking often involves using multiple frames at once


Story or map defining elements of a situation. May include gaps, uncertainty, and open questions.
Data Frame
anchored by
Data
Data
Data
guides gathering of
Data
Data
guides updating of
Mental model of how the system works
Mental Simulation
Predictions about system behaviour

Sensemaking uses abductive reasoning



D is a collection of data (facts, observations, and givens).
H explains D (would, if true, explain D).
No other hypothesis can explain D as well as H does.
Therefore, H is probably true.

Just-in-time mental models

- We need mental models to make sense of things
- May be comprehensive, or ‘just-in-time’
- Difficult troubleshooting cases usually involve some element of building just-in-time models on the fly
- The better the mental model you have to start with, the easier this is
    - This can involve understanding transferred from other, similar systems

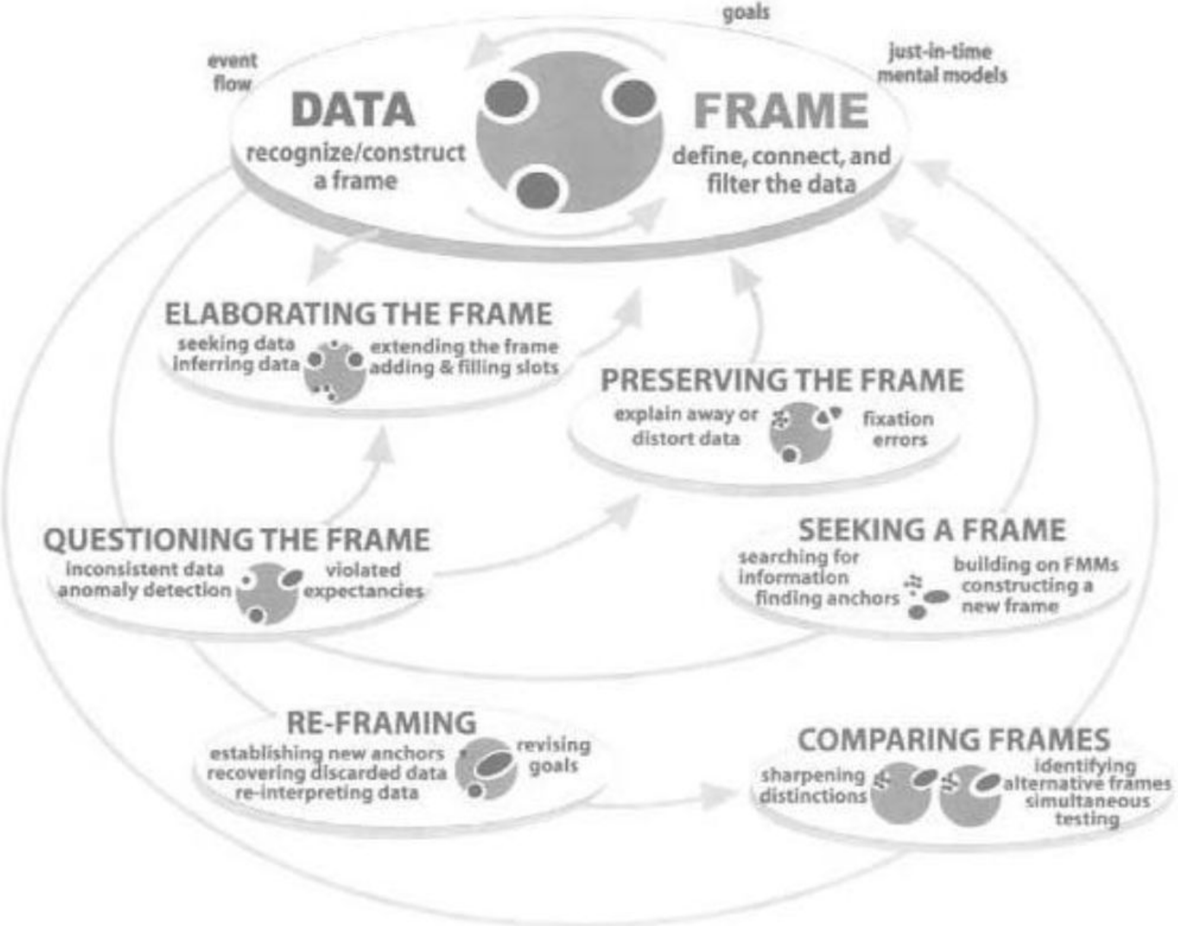
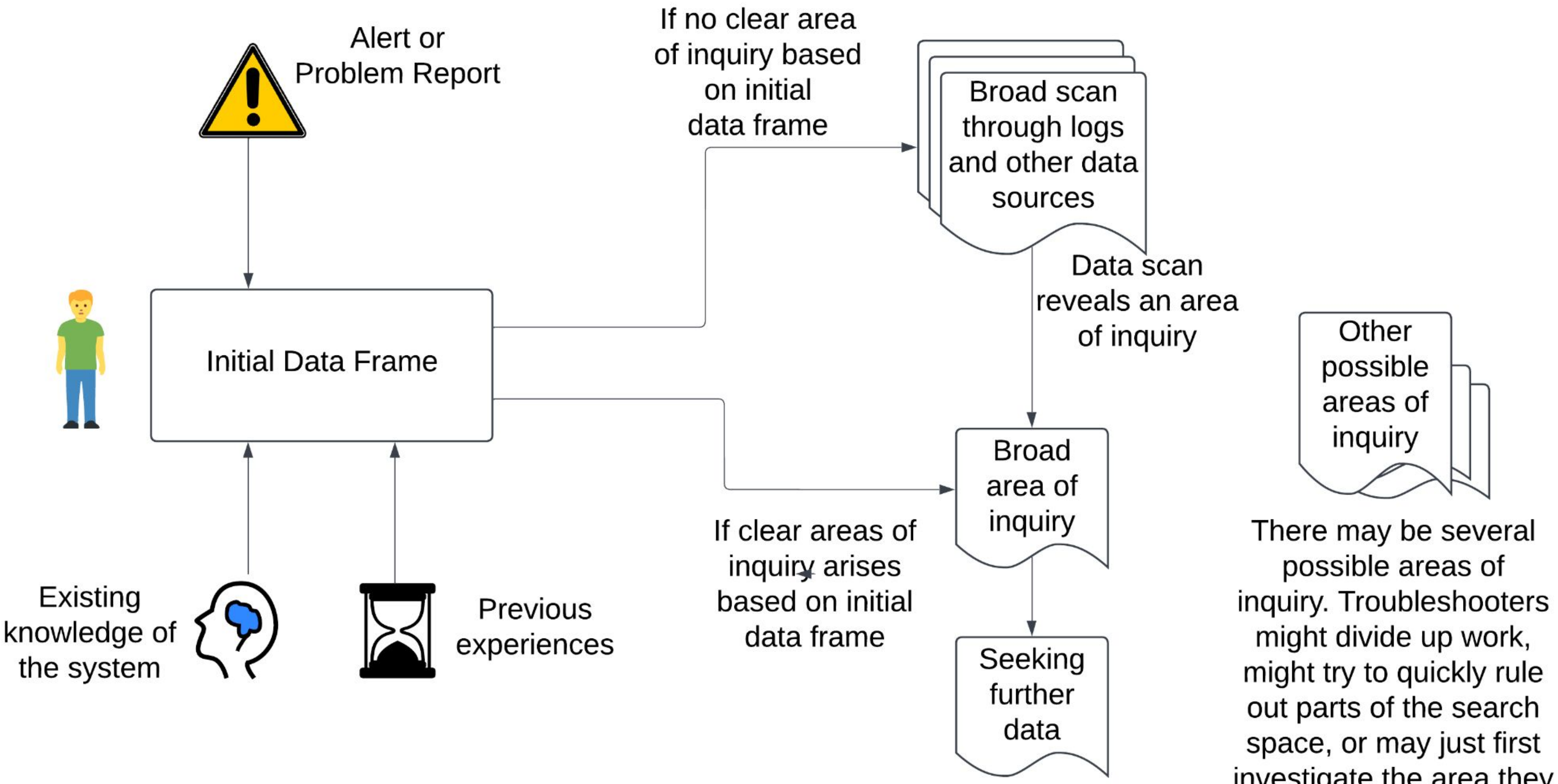


FIG. 6.1. Sensemaking activities.

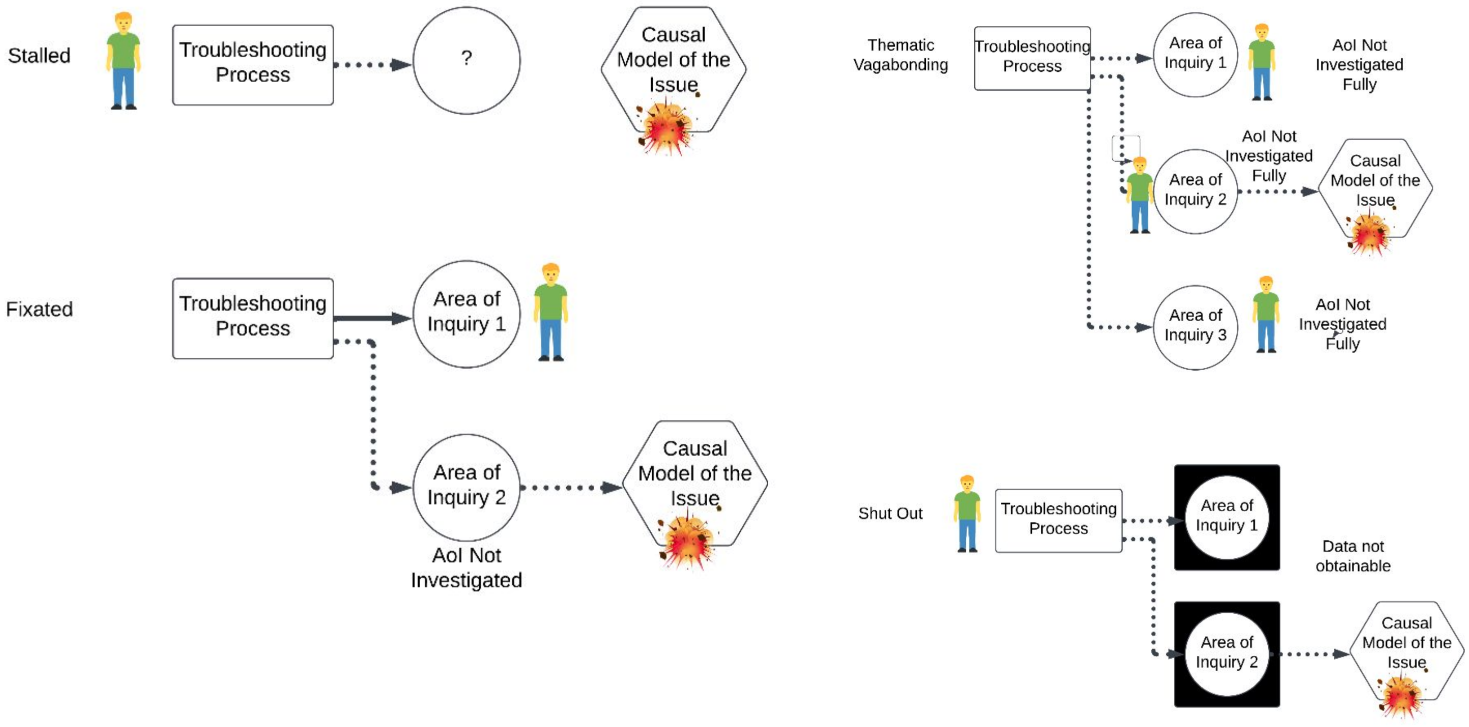
Frame Theory: Implications

- There is no generic sensemaking skill - it is always situated in a given domain of expertise
- Noticing anomalies is really important - unexpected data is what lets us improve or replace frames that aren’t working
- Having a rich set of frames is critically important
- Having comprehensive mental models of systems is probably less important than having a partial model plus the ability to expand that model via introspection of the system


Alert or Problem Report
If no clear area of inquiry based on initial data frame
Broad scan through logs and other data sources
Data scan reveals an area of inquiry
Initial Data Frame
Other possible areas of inquiry
Broad area of inquiry
If clear areas of inquiry arises based on initial data frame
Existing knowledge of the system
Previous experiences
Seeking further data

There may be several possible areas of inquiry. Troubleshooters might divide up work, might try to quickly rule out parts of the search space, or may just first investigate the area they believe is most likely to reveal the causal mechanism.

Maladaptive diagnostic modes



Stalled

Troubleshooting Process

?

Causal Model of the Issue

Fixated

Troubleshooting Process

Area of Inquiry 1

Area of Inquiry 2

AoI Not Investigated

Causal Model of the Issue

Thematic Vagabonding

Troubleshooting Process

Area of Inquiry 1

AoI Not Investigated Fully

Area of Inquiry 2

AoI Not Investigated Fully

Causal Model of the Issue

Area of Inquiry 3

AoI Not Investigated Fully

Shut Out

Troubleshooting Process

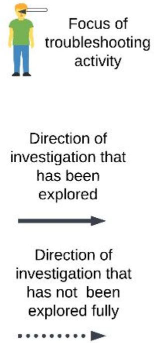
Area of Inquiry 1

Data not obtainable

Area of Inquiry 2

Causal Model of the Issue

Key

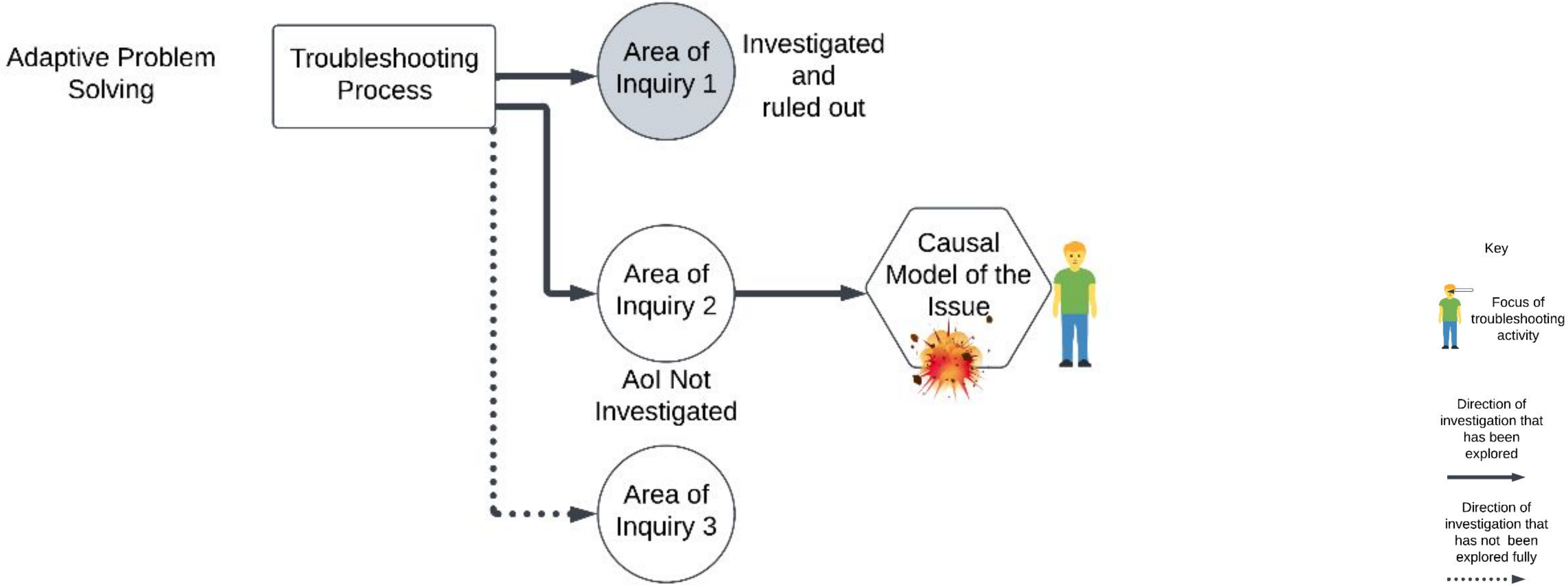


Focus of troubleshooting activity

Direction of investigation that has been explored

Direction of investigation that has not been explored fully

Adaptive problem solving mode



Adaptive Problem Solving

Troubleshooting Process

Area of Inquiry 1

Investigated and ruled out

Area of Inquiry 2

AoI Not Investigated

Area of Inquiry 3

Causal Model of the Issue

Key

Focus of troubleshooting activity

Direction of investigation that has been explored

Direction of investigation that has not been explored fully

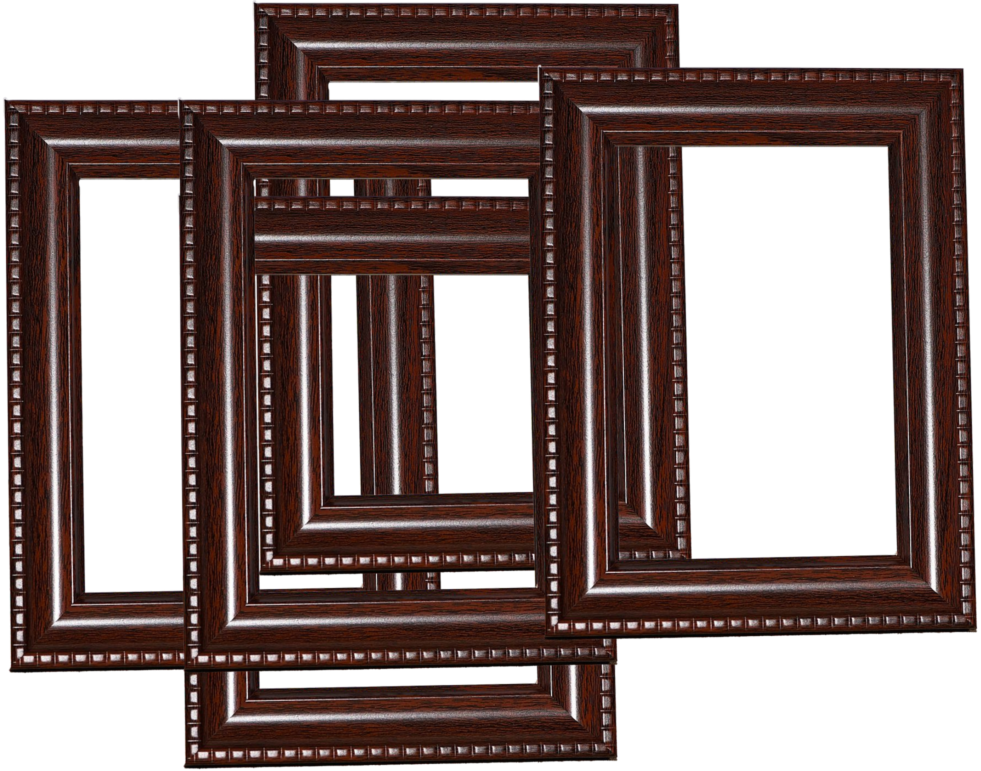
How to troubleshoot the worst issue of your career

- You may need to build some new just-in-time mental models
- You may need to look beyond logs and metrics
- You may be dealing with parts of the system you don’t even know exist, or configuration options you’ve never heard of
- Treat it as a search for answers, rather than needing to come up with hypotheses from limited data

How to troubleshoot the worst issue of your career

- All issues are troubleshootable
- It may take time and persistence
- It may take asking for help, reading docs, or learning new introspection tools

When you get to the other side you will have a richer set of frames.



Sources used in this talk

Jens Rasmussen. Skills, rules, knowledge; signals, signs, and symbols, and other distinctions in human performance models, *IEEE Transactions on Systems, Man, and Cybernetics* (1983).

Gary Klein. *Sources of Power: How People Make Decisions* (1999). MIT Press.

Gary Klein. *Streetlights and Shadows: Searching for the Keys to Adaptive Decision Making*. MIT Press.

John Flach et al. Decisionmaking in Practice: The dynamics of muddling through, *Applied Ergonomics* (2017).

Dorner, Dietrich. *The Logic of Failure,* Basic Books, 1997.

Gary Klein, J.K. Phillips, E.L. Rall, Deborah A. Peluso. A data-frame theory of sensemaking. In R. R. Hoffman (Ed.), *Expertise out of context: Proceedings of the Sixth International Conference on Naturalistic Decision Making* (pp. 113–155). Lawrence Erlbaum Associates Publishers.

Rudolph, J., Morrison, J., & Carroll, J. (2009). The Dynamics of Action-Oriented Problem Solving: Linking Interpretation and Choice, Academy of Management Review 34(4), 733-756.

Contact me: laura@requisitevariety.net | https://requisitevariety.net (slides available here)