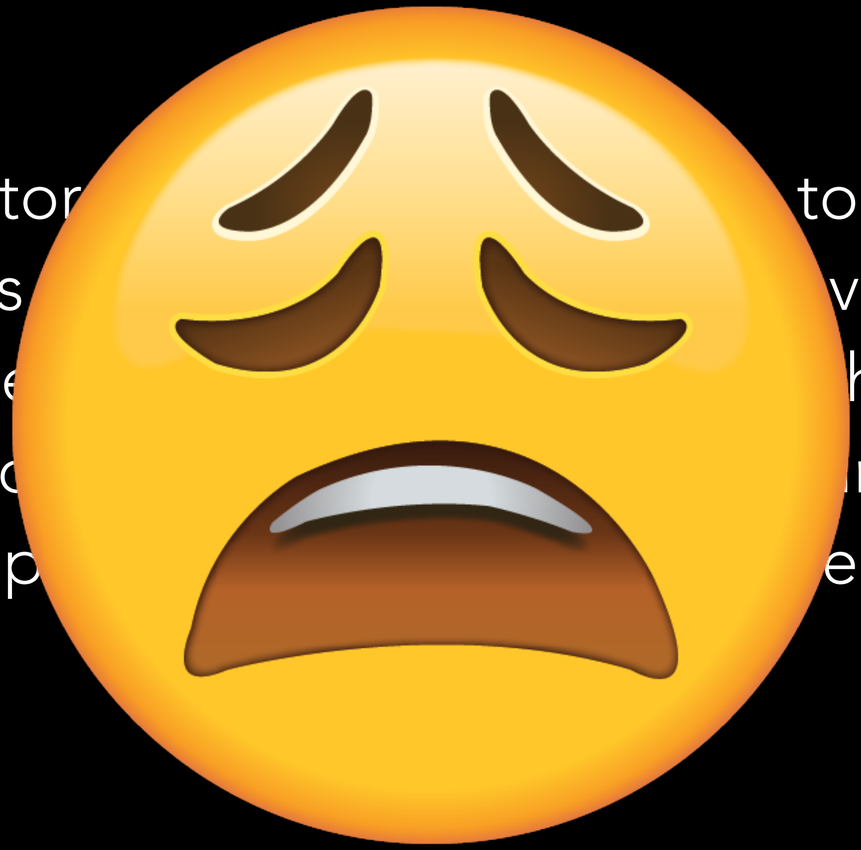
TALES OF OPERABILITY ANTI-PATTERNS

A DARK AND STORMY NIGHT

@kiranb

Kiran Bhattaram

It was a dark and stor torrents — except at occasional intervals violent gust of wind which swept up the hat our scene lies), rattling along the ho ng the scanty flame of the lamp e darkness.



What is operability?

- The ability to keep a system in a safe and reliable functioning condition, according to pre-defined operational requirements.

Characteristics of operability

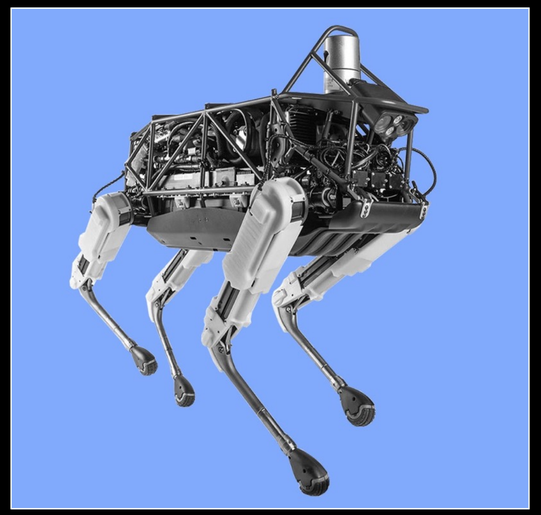
- safety & reliability
- scalability
- grace under pressure
- ease of upgrades
- observability
- usability
- cultural practices around incidents
- AND MORE

Characteristics of an operable system

- Converge towards a stable state.
- Give operators visibility and tools.
- Designed to be usable and unsurprising.

Agenda

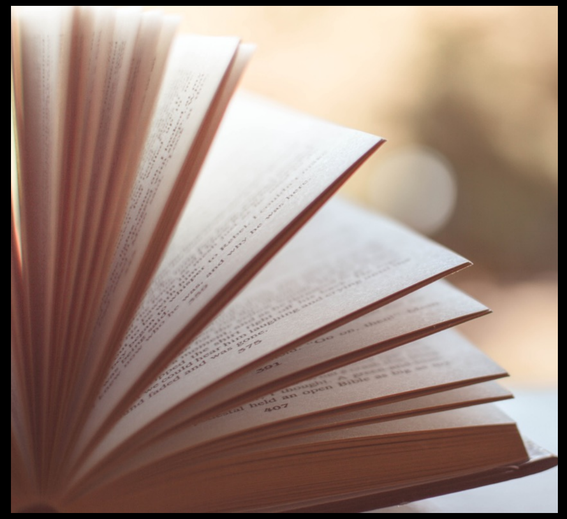
Robustness

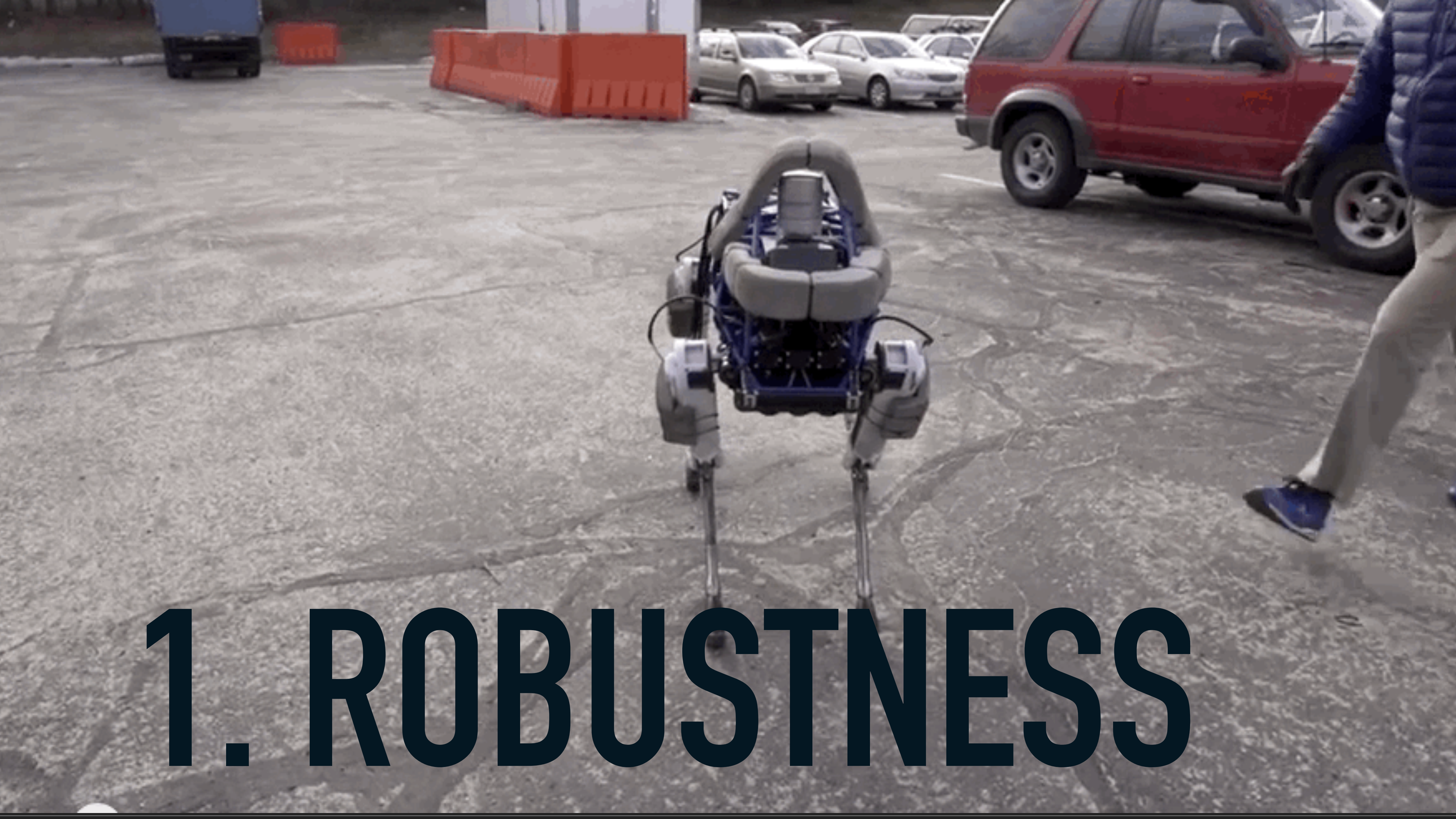


Observability

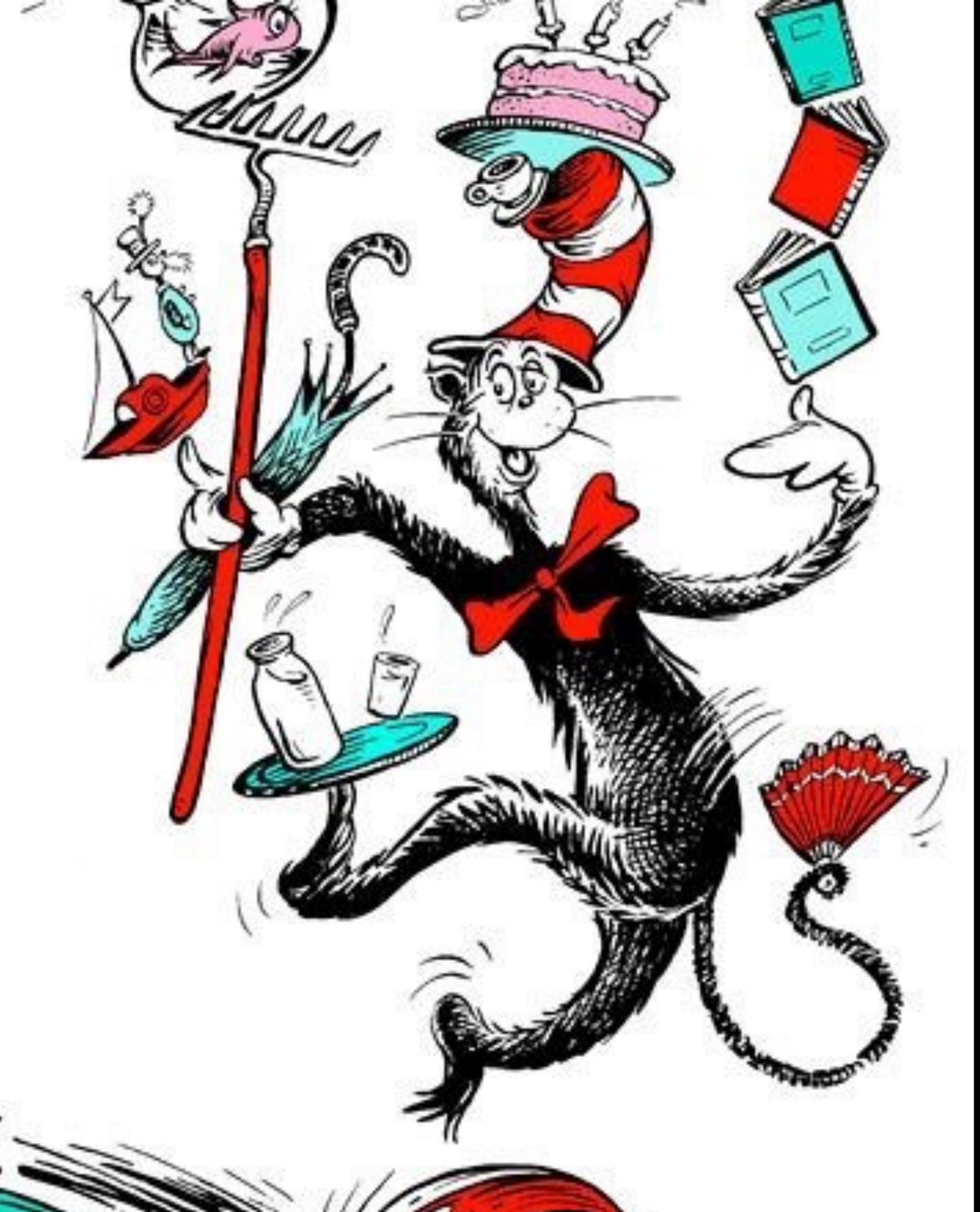
Usability

Review!





1. ROBUSTNESS



STORY 1

THE TALE OF THE SYSTEM THAT COULDN'T GIVE ANYTHING UP

Define your critical path.

Harvest, Yield and Scalable Tolerant Systems

$$\text{Yield} = \frac{\text{successful requests}}{\text{total requests}} \quad != \text{uptime}$$

* dropping requests

$$\text{Harvest} = \frac{\text{data available}}{\text{total data}}$$

* degrading response

Controlling yield: load shedding upstream requests

- categories of load shedders:
  - # of requests
  - # of concurrent requests (protect against the long tail)
  - overall fleet utilization (keep x% of workers for core traffic)

Controlling harvest: circuit breakers

- stop calling a dependency if it seems down!
- what do you return?
  - cached data
  - nil
  - or propagate the error upstream

Controlling harvest: circuit breakers & compartmentalization

http://idighardware.com/2013/10/fire-doors-everything-you-always-wanted-to-know-but-were-afraid-to-ask/

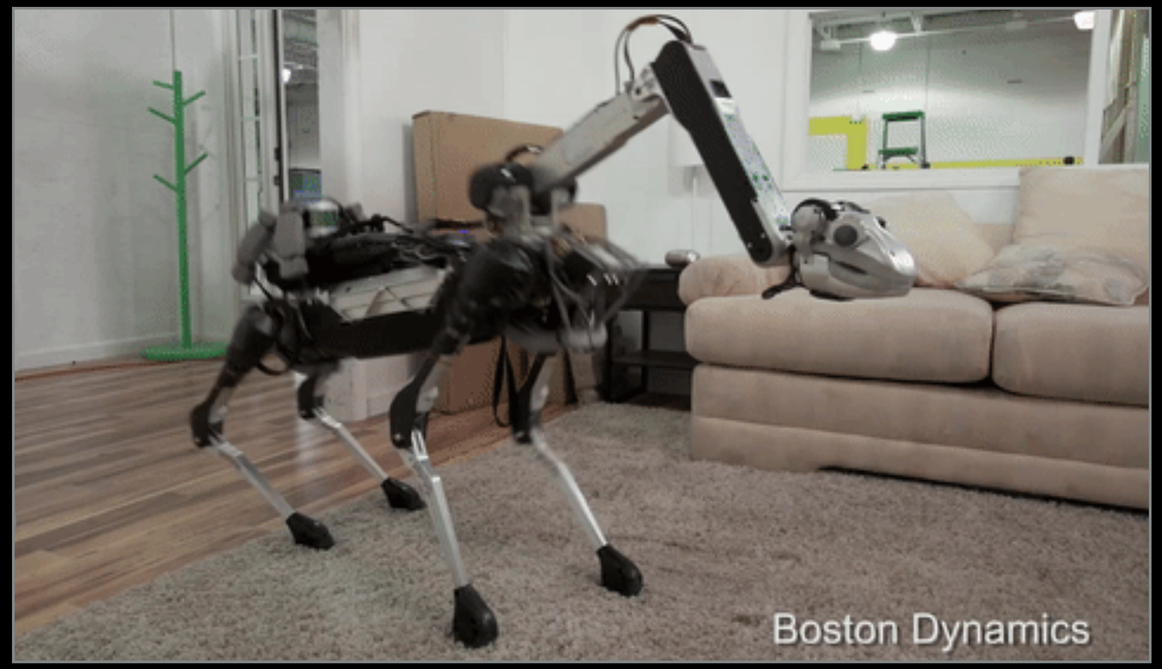
Putting it all together: giving things up

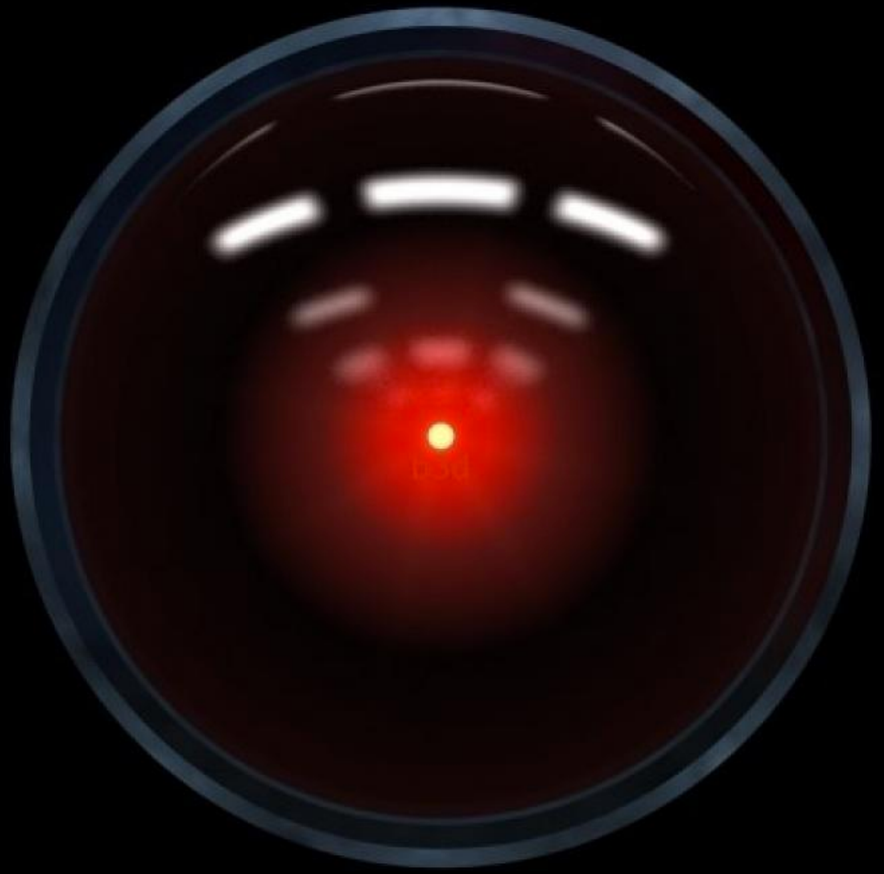
- Combine harvest/yield degradation in different ways to protect the critical path
- Monitor any degradation!
- Dark launch your rate limiters to check what they'd block.

Robustness, in review

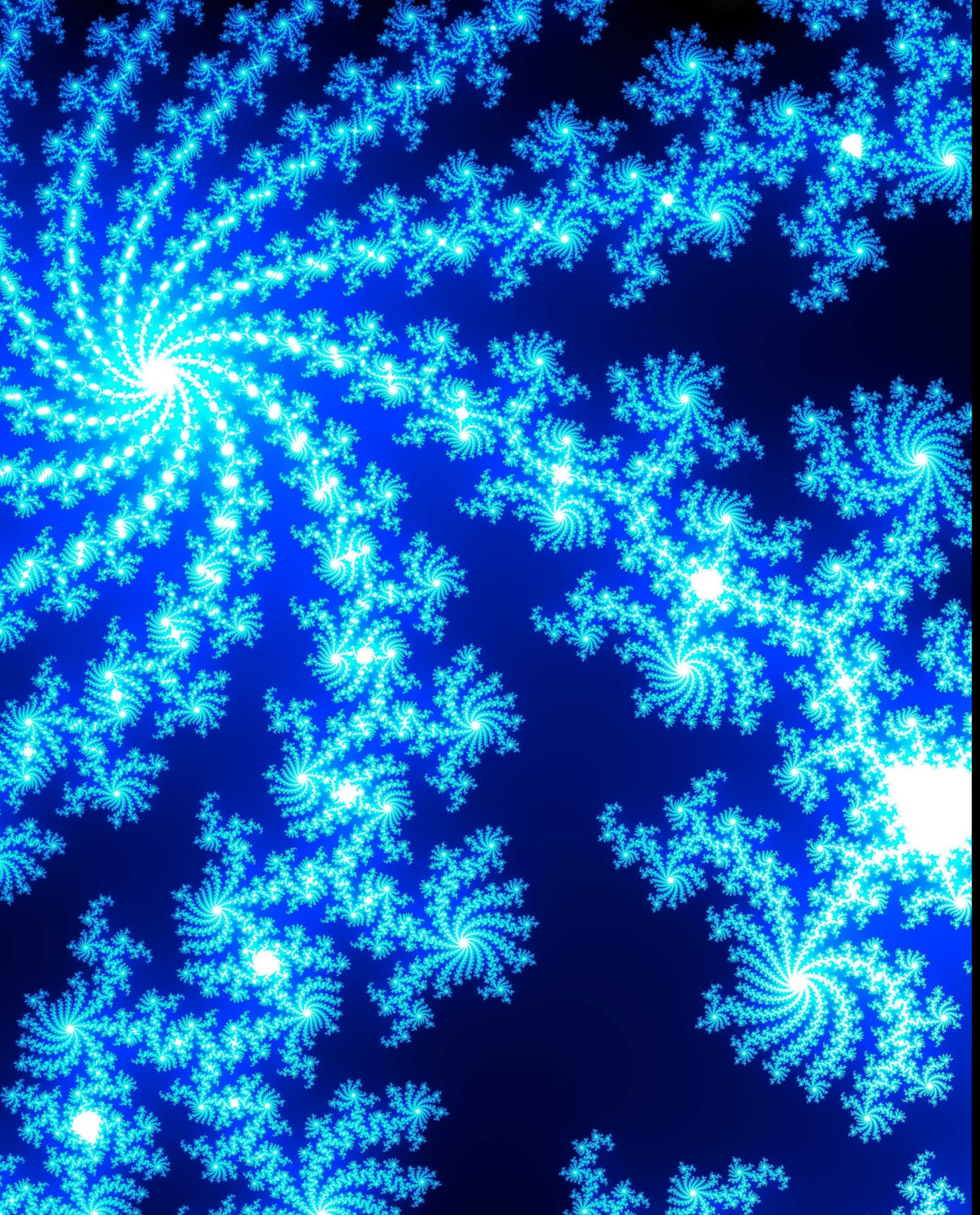
Converge to a stable state.

- know how the system sheds load
- know how it reacts to downstream failures





2. OBSERVABILITY



STORY 2

THE TALE OF THE FRACTAL QUEUE

Instrument EVERYTHING

- especially with queues
- percentiles, not averages
- don’t intermingle logs (keep a searchable trace ID on requests)

Over-collect data, but build dashboards carefully

- work metrics
  - is the system doing the thing it's supposed to?
- resource metrics
  - how are the components of the system behaving?
- build your dashboard with work metrics first.

STORY 4

THE TALE OF THE 64 ALERT WEEK

Don’t normalize deviance

Knowing what to alert on

- Monitor the alert volume of your system!
- Pages should be actionable and represent user pain.

Observability: what we learned

- Kiran has a special vendetta against unmonitored queues.
- Building good dashboards: work metrics & resource metrics.
- Monitor alert volume, too!

3. USABILITY

A quick side note: Nielsen Heuristics

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition vs. recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

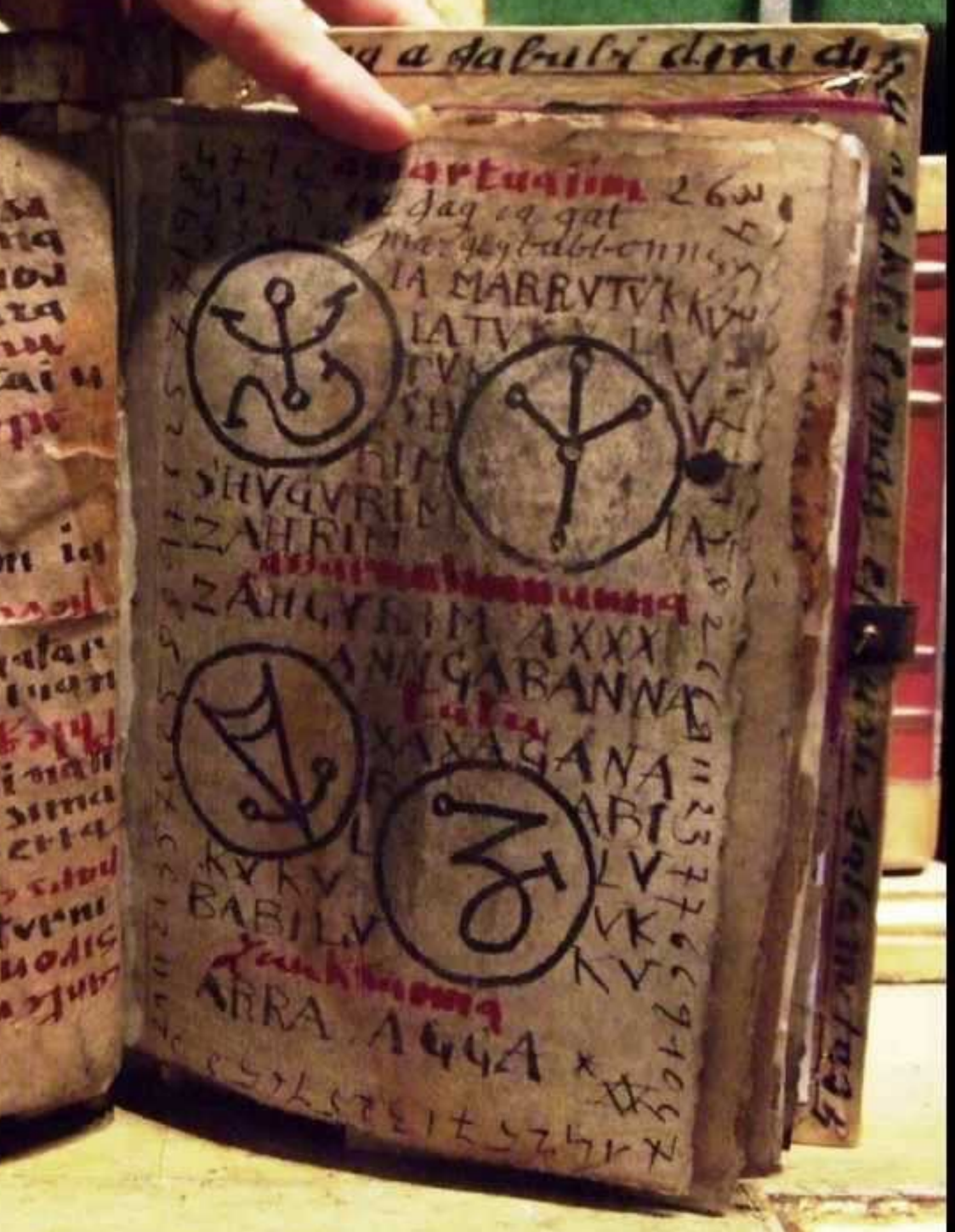
Story 5: the tale of the special snowflake service

Heuristic 4. Consistency and Standards

- pattern-matching across similar systems is really valuable!
- Choose boring technology: spend your innovation tokens wisely!

Heuristic 3. User control and freedom

- Tooling is a part of the service!
  - relatedly, deploy mechanisms are related to availability!
- Give operators the ability to change operational parameters.



STORY 6

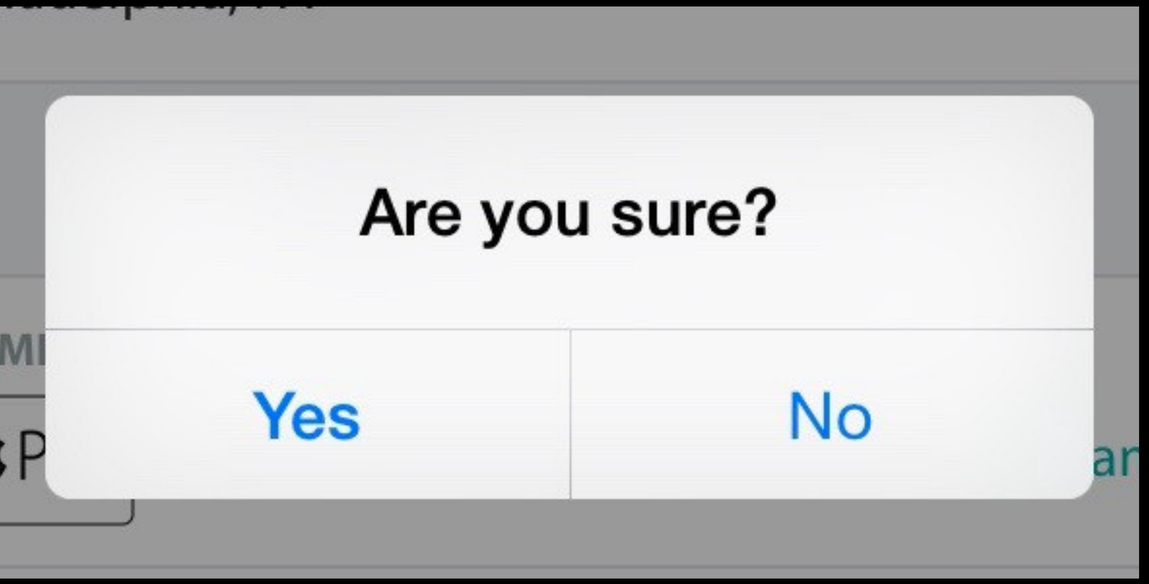
THE TALE OF THE OPS SPELL BOOK

Heuristic 6. Recognition v. recall

- Keep checklists minimal and heavily automated.
  - long flowcharts in a runbook are :(
- relatedly: scripting user communications is helpful.

Heuristic 1. Visibility of system status

- which of these are changes to production?
  - config changes
  - deploys
  - utility script runs
  - failovers
  - adding/decreasing capacity



STORY 7

THE TALE OF THE AMBIGUOUS ERROR MESSAGE

Heuristic 9. Help users recognize, diagnose, and recover from errors

- error messages are a crucial part of your interface
- Writing a good alert message:
  - expressed in plain language, precisely indicate the problem, and constructively suggest a solution (runbooks!)
  - (ex.) CRITICAL: Served 5% 5xx results in the last 5 minutes! <link to runbook>

Usability, in review

- Operational experience matters! Consider:
  - whether the system follows general conventions.
  - how it alerts operators to errors clearly and unambiguously.
  - how minimal and usable the tooling is.

Review

- Robustness
  - Does your system converge to a stable state?
- Observability
  - Can you infer what the internal state of the system looks like?
- Usability
  - Do your operators have control over the state of the system?
    Do you adhere to general standards?

STORY THE LAST

THE TALE OF THE SAD QUEUE

STORY THE LAST

A DARK AND STORMY NIGHT

Resources

- Harvest, Yield, and Scalable Tolerant Systems (Brewer & Fox)
- How Complex Systems Fail (Cook)
- "Going solid": a model of system dynamics and consequences for patient safety (Cook)
- Nielsen’s Usability Heuristics
- Choose Boring Technology (Dan McKinley)
- Site Reliability Engineering: How Google Runs Production Systems
- Stripe’s (upcoming) rate limiting blog post
- Collection of postmortems (Dan Luu)

On Designing and Deploying Internet-Scale Services, James Hamilton

- list of best practices, from design, to upgrades, to incident response

THANKS!

Thanks to Ines Sombra, Charity Majors, Alyssa Frazee, Rachel Sanders, and Andy Bonventre for review!

STUFF I COULDN'T GET TO

APPENDIX

decouple deploys from releases

- get a minimal version in dark-reads into production asap
  - corollary: have good kill switches!
- Know what rollbacks look like

collect operational metrics in this shadow phase

- Gain historical knowledge of what the system's healthy state looks like.
- Tweak your alerts and SLAs.
- Gameday the system! Write runbooks!