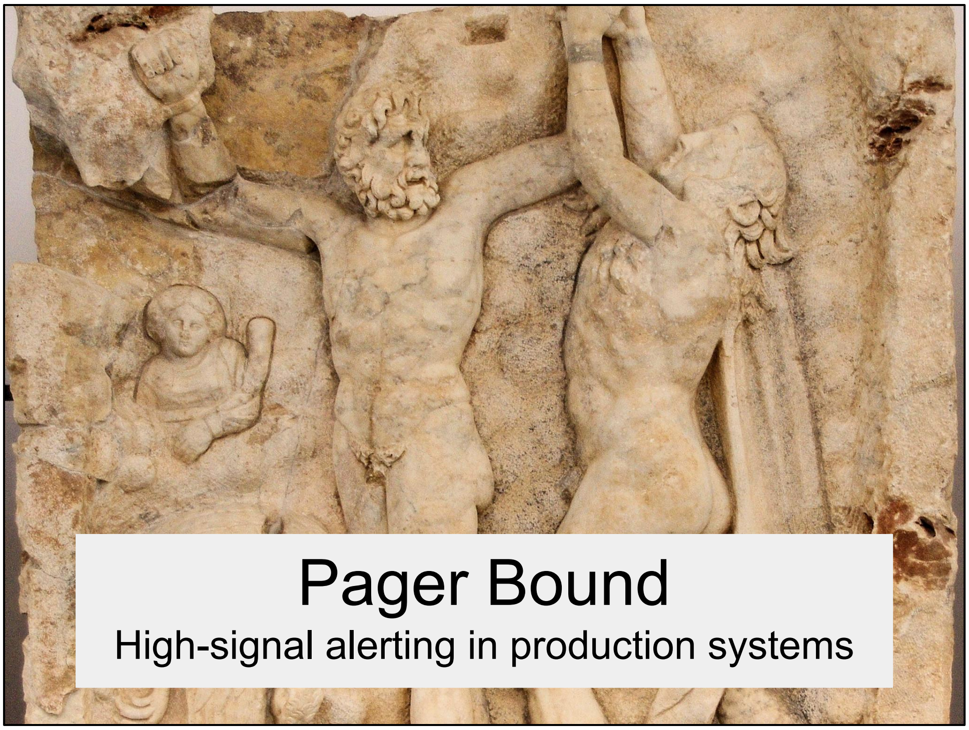


Pager Bound

High-signal alerting in production systems

This is a 25-minute talk intended for software and operations engineers. For intercom.io in Dublin on July 1st 2014.

Hello, thank you for having me.

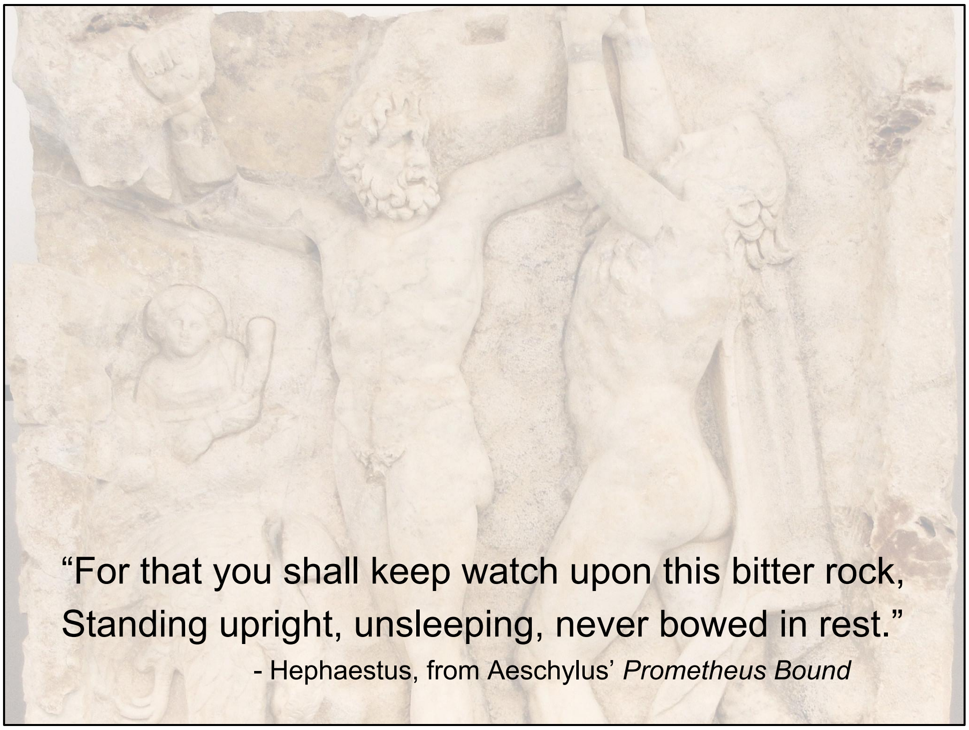
Rich asked me to talk about metrics, monitoring and alerting (really, paging).

This talk will be about 25 minutes or so, and I hope we'll have some time for more discussion at the end.

The content is somewhat aspirational; I've chosen to restrain my hedging and be fairly definite in the interests of clarity.

This poor soul on the slide is Prometheus, who was bound to a rock to have his liver pecked out eternally for giving fire to humankind.

Image:
Reliefs From The Sebasteion of Aphrodisas: Herakles Frees Prometheus
by Ken and Nyetta
https://flic.kr/p/bELTwa



“For that you shall keep watch upon this bitter rock,
Standing upright, unsleeping, never bowed in rest.”

- Hephaestus, from Aeschylus’ *Prometheus Bound*

I stumbled across this quote some years ago.

Let’s just say that it reminds me of certain pager rotations I have been on.

We can do better: there’s no need to be quite so tragic. Heroism in production systems is something that we should avoid.

So this talk will focus on design considerations for high-signal alerting. It’s based on a similar talk I gave to the Facebook Dublin ops crew in August of last year.

Quote:
Prometheus Bound by Aeschylus from Prometheus Bound and Other Plays in Penguin Classics
Transl. Philip Vellacott

Image:
Reliefs From The Sebasteion of Aphrodisas: Herakles Frees Prometheus
by Ken and Nyetta
https://flic.kr/p/bELTwa

A little about me. This is a photograph from the alternate world where I took that banking job in 2003.

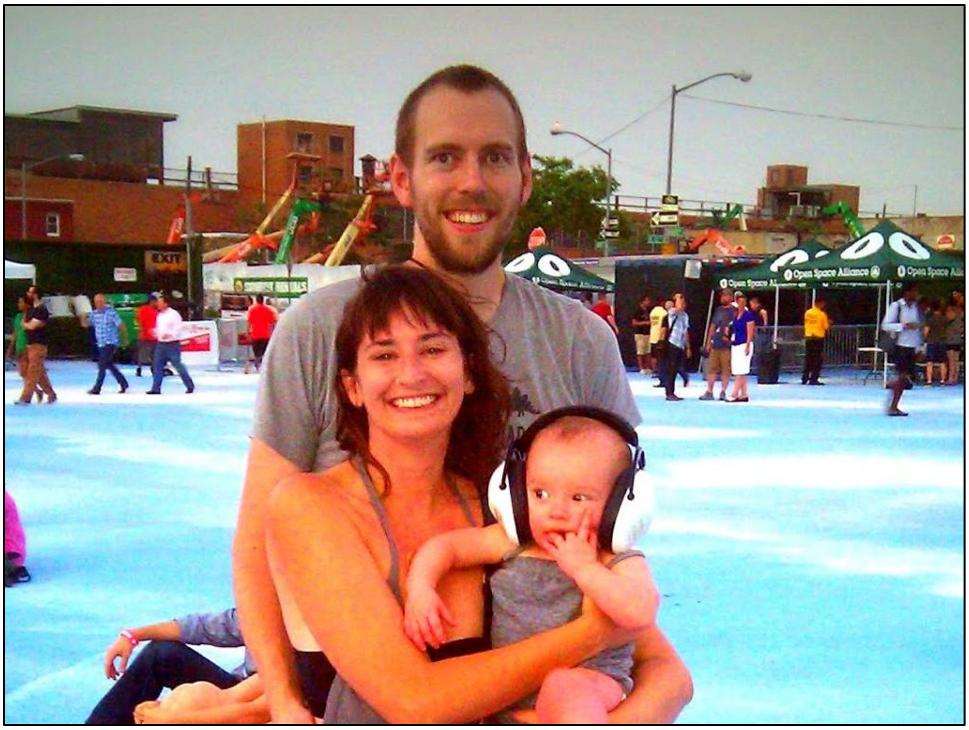
In our world, I ended up working as a sysadmin for a couple of Irish ISPs, and then for Google here in Dublin until recently.

There, I was a corporate sysadmin; a "product" Site Reliability Engineer in Ads; and an "infrastructure" SRE in Storage.

So it's worth noting up front that the environments I've worked in have been quite different to yours: large-scale service oriented architecture, many peer production teams.

However, many of the lessons learned apply generally.

Image:
Ads SRE Dublin's "formal friday", July 2009
by Alexey Klimov. Used with permission.



First, credit where credit is due:

This is Rob Ewaschuk (and his lovely family).

Rob and I were colleagues at Google. He is one of the better human beings on the planet.

He wrote a really good document called "A Philosophy on Alerting": this talk owes a lot to it.

Image:
Rob's G+ profile photo
https://plus.google.com/+RobEwaschuk
Snarfed on the assumption Rob won't mind. :o)

Second, a brief detour: what *is* an alert?

For our purposes, we're talking about notifications of interesting events sent by automated systems.

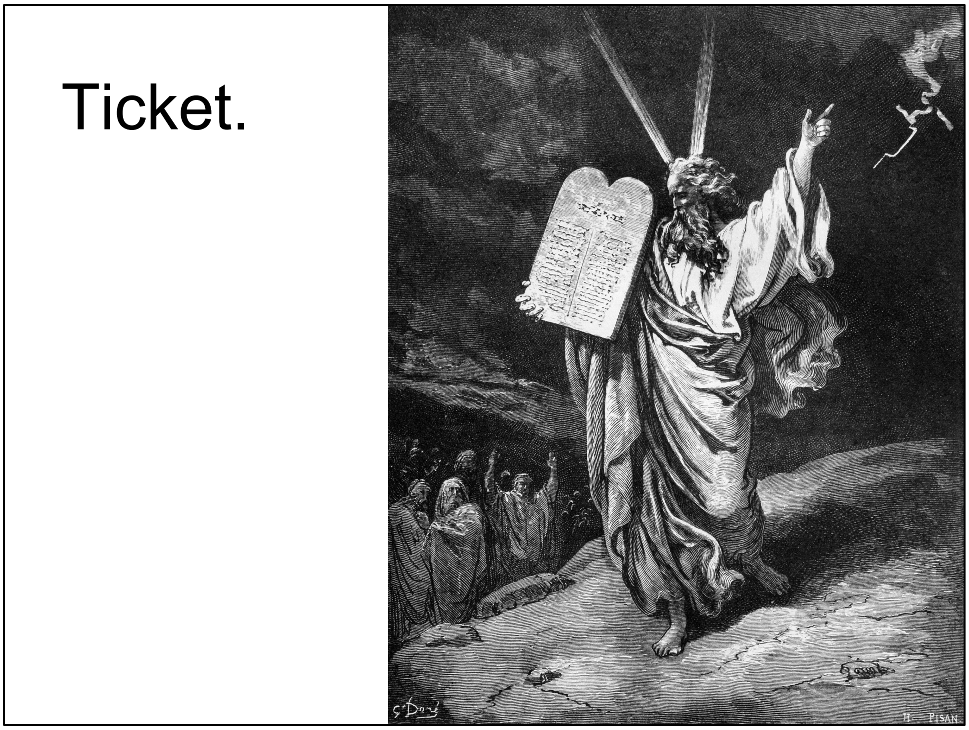
Let's talk about three alerting channels, and when each applies.

Image:
Reliefs From The Sebasteion of Aphrodisas: Herakles Frees Prometheus
by Ken and Nyetta
https://flic.kr/p/bELTwa

Page.

An urgent notification to a human: immediate human attention is required.

Image:
God Appears to Moses in Burning Bush. Painting from Saint Isaac's Cathedral, Saint Petersburg
By Eugène Pluchart
http://commons.wikimedia.org/wiki/File:Moses_Pluchart.jpg



Ticket.

An issue in some kind of tracking system; maybe a rolled-up report. Human attention is required but it can wait.

Image:
Gustav Doré: Moses Giving the Law on Mt. Sinai
By Ed Suominen
https://flic.kr/p/nEwAjT

Log.

A log message (or a timeseries datapoint): something we might want to investigate or correlate against later.

Image:
Illustration showing the story of Jacob.
Folio 12v from the Vienna Genesis.
http://commons.wikimedia.org/wiki/File:ViennGenFol12vJacob.jpg

~~Email.~~

Note the conspicuous absence of email as an alerting channel. This is intentional.

To paraphrase Rich in a previous job:

1. If you find something that's broken, fix it.
2. If it's emailing you, it's broken. See point 1.

Email alerting is a production antipattern in large systems.

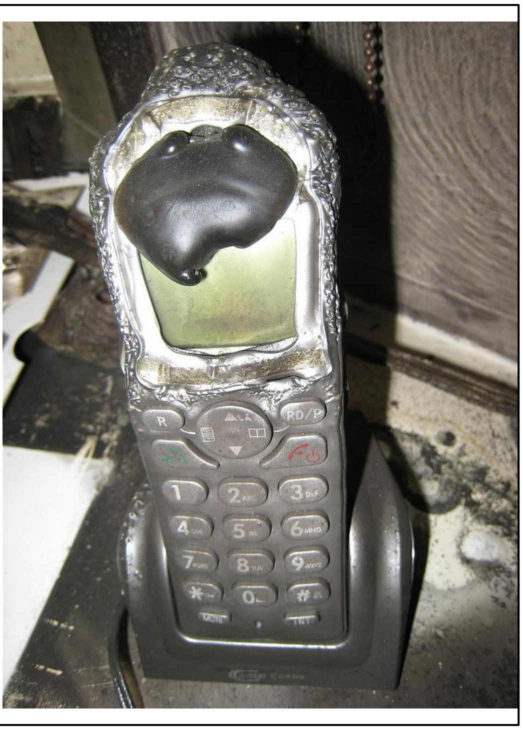
It's insidious, because it's essentially default operation for something like cron, and quite a few other systems that are used for raising simple notifications.

It's hard (locally) to see what harm one more email will do (globally).

However, as your systems scale, at best you have an expensive and slow logging mechanism; at worst, your team's personal mail filters become an integral part of your monitoring, or you miss a critical condition because mail delivery is wedged somewhere in your fleet.

Image:
God Appears to Moses in Burning Bush. Painting from Saint Isaac's Cathedral, Saint Petersburg
By Eugène Pluchart
http://commons.wikimedia.org/wiki/File:Moses_Pluchart.jpg

Oncall.



How do we decide what to page ourselves about?

On the one hand, we want to know when there are problems in our services.

On the other, no-one wants to be buried under a heap of pages of varying quality, holding a melted phone.

How do we avoid that?

How do we always have - as Rich puts it - the right alert, at the right time, to the right person, who should do something about it?

Image:
fire 05 - melted phone
By mjtmail (tiggy)
https://flic.kr/p/4YdvN4

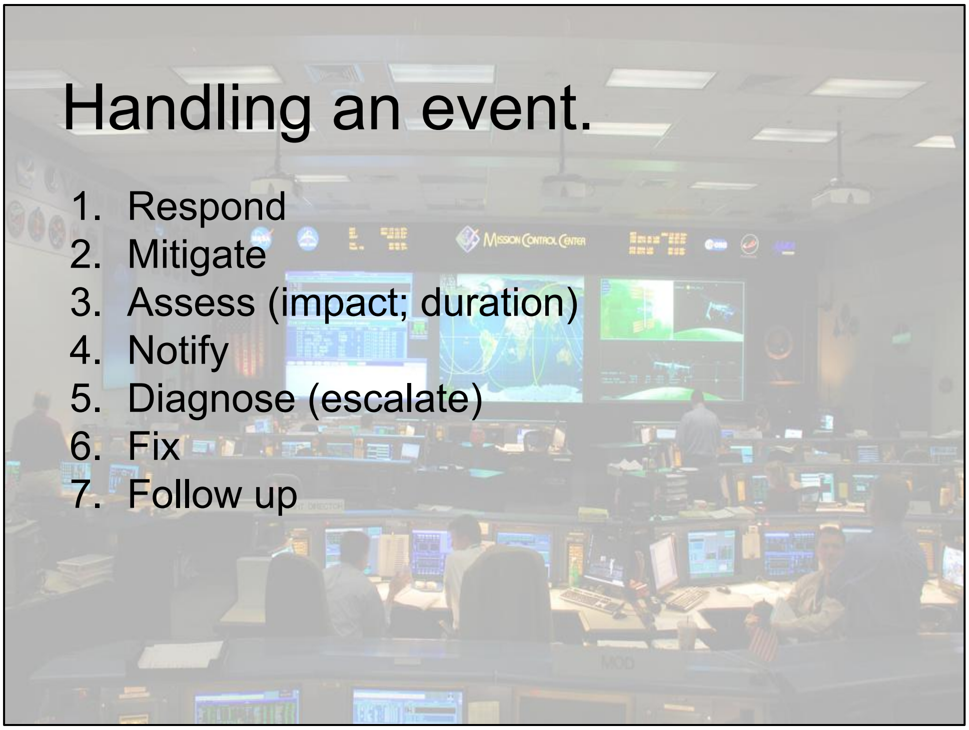
Those of us who watched a few too many ‘90s movies started out with an image of oncall being the network operations centre, or mission control.

A dimly-lit hive of activity with big screens and lots of heads-up information.

For me now, aged 36 and with two kids, I like to make sure oncall is either a (mostly) normal day at work, or sleeping soundly near my pager.

So what is it worth getting paged out of bed for?

Image:
Mission Control
By Mike Renlund
https://flic.kr/p/4yXQvG



Handling an event.

1. Respond
2. Mitigate
3. Assess (impact; duration)
4. Notify
5. Diagnose (escalate)
6. Fix
7. Follow up

On a slightly different tack, how many events can we handle per shift - say, 12 hours - and still do a good job?

10? 20? Not the way I define "good":

1. Respond to the page.
2. Apply available early mitigations (drain).
3. Assess impact; assess duration.
4. Notify affected customers.
5. Diagnose the problem. Escalate as necessary.
6. Fix it or put the fix in train.
7. Follow up: "5 whys" (but note Bill de hÓra's talk about the single-cause fallacy).
   - "What needs to happen so that this page never occurs again?"
   - Link to similar events; is there a class of problems emerging over time that we can eliminate with good production engineering?
   - Write blameless postmortems. This isn't about finding who to fire: this is about calmly and rationally figuring out how the decisions we made in the past got us into the situation we're in now, and what we're going to do about it.

If we want to do this kind of job for every event, then maybe 2 in 12 hours. Anything else is just going to wear away at our ability to make good decisions.

Image:

Mission Control
By Mike Renlund
https://flic.kr/p/4yXQvG

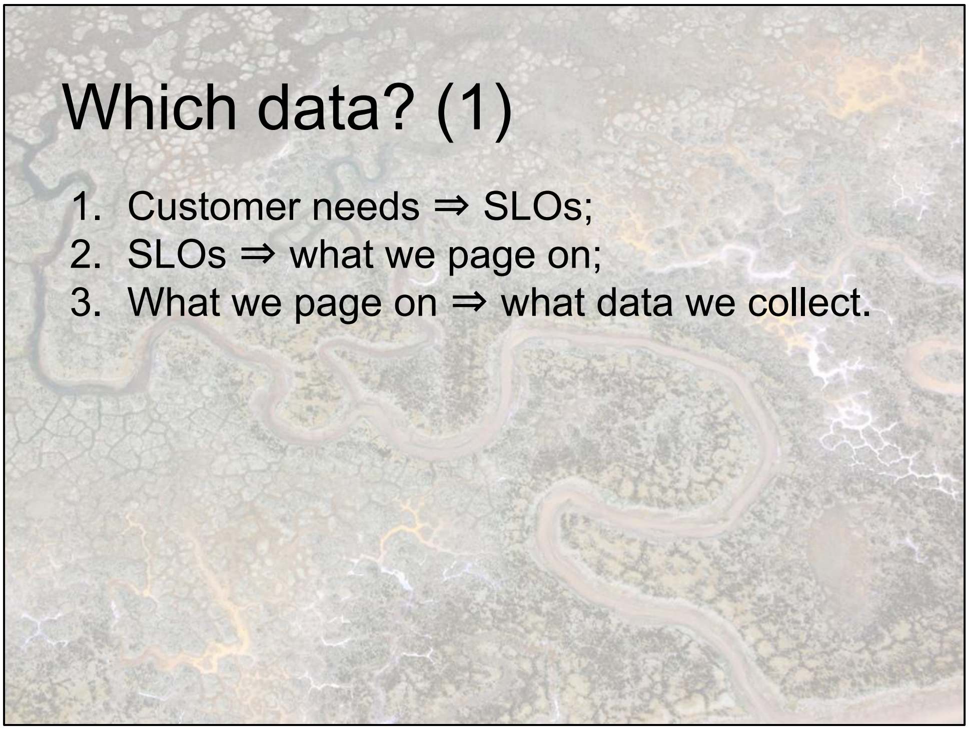
Too often, monitoring *grows* as follows:

- we are building a system;
- we think about stuff that might make it misbehave, and we page ourselves for that
    - or at least the stuff it's easy to get metrics for;
- a customer calls to tell us our system's broken;
- we figure it out & add new monitoring that'll catch it and page us next time.

This works, sort of, long-term. We get **a** monitoring system, it's catching most stuff, and growing organically.

But it's a reactive thing, it can get very noisy, and it is not the best job we can do as engineers.

Image:
Organic Forms
By Steve Jurvetson
https://flic.kr/p/6uPzYV



Which data? (1)

1. Customer needs ⇒ SLOs;
2. SLOs ⇒ what we page on;
3. What we page on ⇒ what data we collect.

I’ve avoided, so far, the fact that monitoring isn’t one thing. We’ve got (roughly speaking):

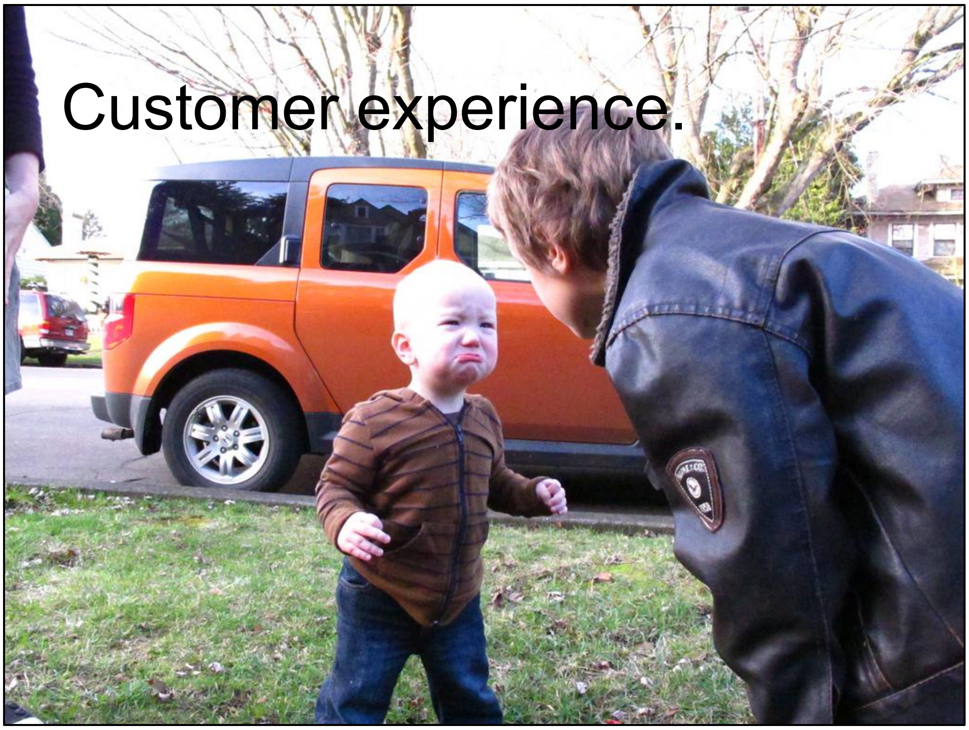
1. collection of historical data - trending, etc.
2. alerting, including paging.

These should be driven by the same data. Which data?

Here’s my take on it:

1. The needs of the customer (that is: the needs of the business) should drive service-level objectives
   - SLOs! Bear with me if this sounds like waffly business-contract bullshit - it’s genuinely useful.
2. Service-level objectives should drive what we page on.
3. What we page on should drive the monitoring data we collect.

Image:
Organic Forms
By Steve Jurvetson
https://flic.kr/p/6uPzYV



Customer experience.

Our customers, the users of our services - other engineers, end users, whatever - care about surprisingly few things.

- Availability
- Latency
- Durability
- Features
- ...

More or less in that order.

Granted that the **way** they care depends on who they are:

- an engineer might care more about having bounded tail latency;
- an end-user about whether their cat picture eventually uploads,
- or maybe the number of fail-whales they see in one day.

They **don't** care if 5% of application servers in Atlanta are flapping. They **don't** care that we just lost 50% of our capacity between Oregon and Taiwan.

When we frame things in terms of customer experience, we get happy results:

- We can talk right across the business, in easily-understandable language, about what our systems do and how they relate to the bottom line.
- We can make explicit tradeoffs in the design and launch of new features, re-architectures, capacity planning, etc.
- We can build an alerting infrastructure that is not guaranteed to make us feel like the kid in the slide.

Image:
sad face
By eyeliam
https://flic.kr/p/7y2pjr

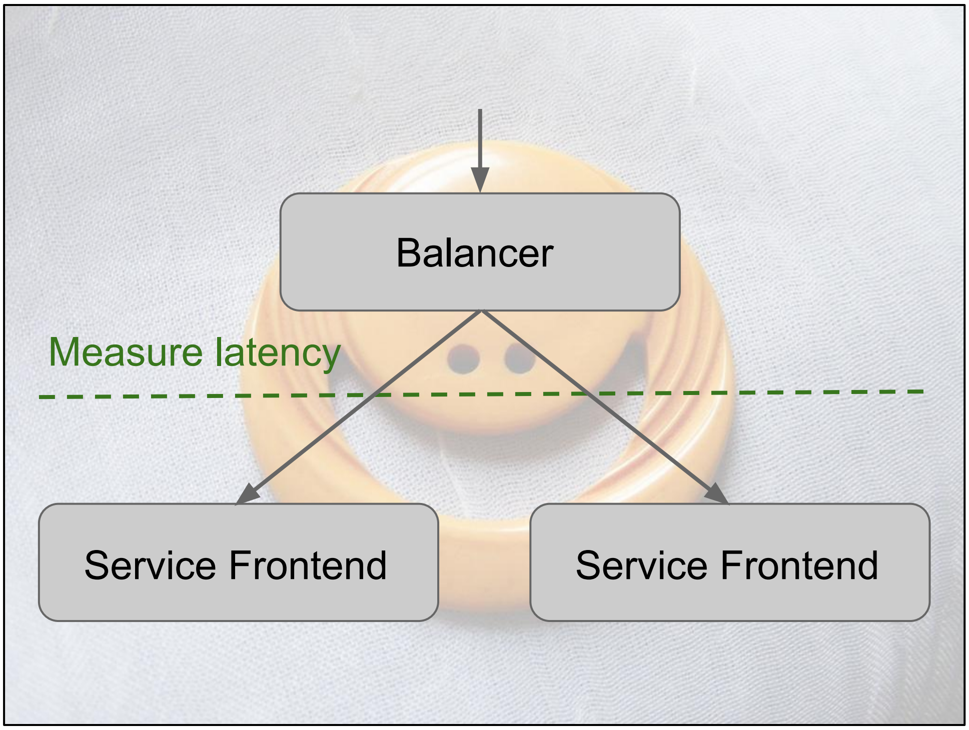
Service levels.

So we figure out how we’re going to track the customer experience - as closely as possible.

This may not be easy! As with any metric, we’ve got to be careful what we measure.

But if we can get some close proxies for customer experience, in the dimensions the customer cares about, these can be our **service level indicators**.

Image:
I'm a Happy Button, Why Aren't You?
By LearningLark
https://flic.kr/p/azjW1N



Balancer

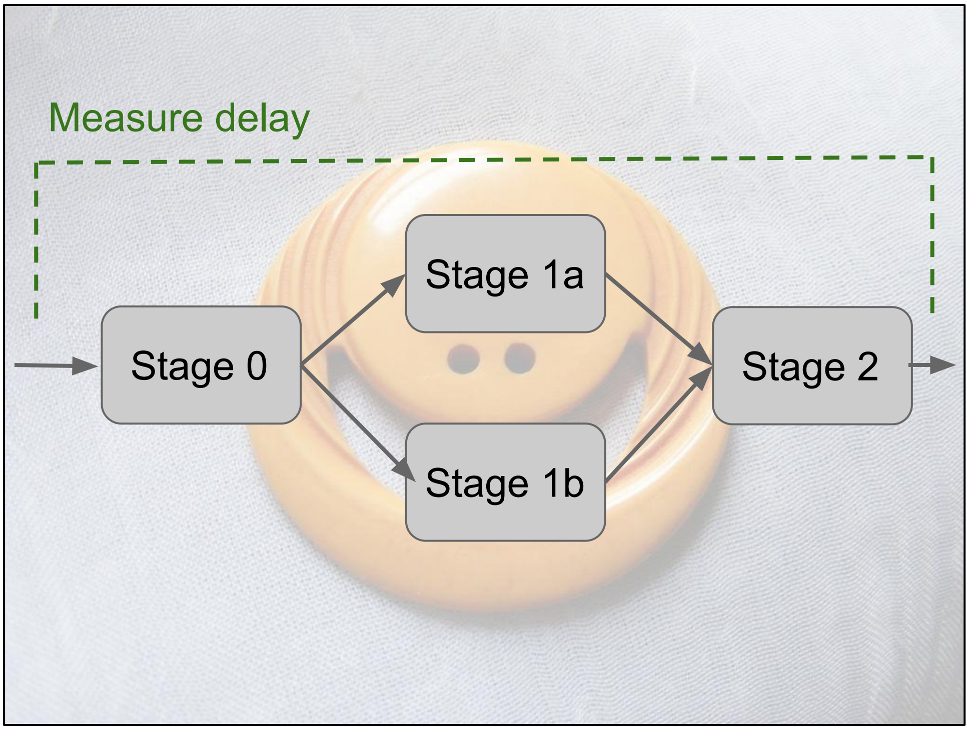
Measure latency

Service Frontend

Service Frontend

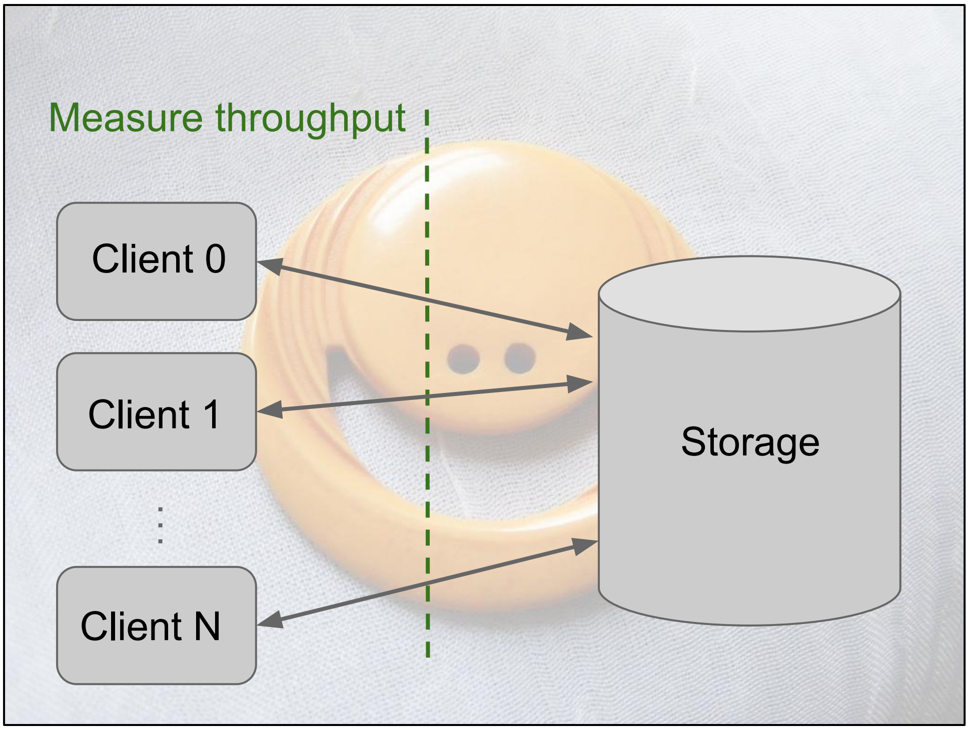
For example, we might measure latency on requests to our service from loadbalancers;

Image:
I'm a Happy Button, Why Aren't You?
By LearningLark
https://flic.kr/p/azjW1N



Or the delay between a work item arriving at and exiting a pipeline;

Image:
I'm a Happy Button, Why Aren't You?
By LearningLark
https://flic.kr/p/azjW1N



Or read & write throughput measured at the client in a storage subsystem.

Now, we can talk rationally about how the design & implementation of the system - and new features - impacts these indicators.

We can set objectives for them, which we are promising to defend - **service level objectives**.

These could get complex! This can be very hard! For instance, while it might be straightforward to bound some proportion of end-user-visible latency - for example 90th %ile < 50ms - providing good SLOs for writes to a shared storage system may require differentiating classes of write, maybe some concept of "conforming" writes, etc.

Figuring SLOs out and stating them explicitly helps everyone. Our customers, because

- technical customers have something to engineer against!
  - If we won't tell them, then we in fact have an implicit SLO, i.e. whatever the system can manage.
  - If we won't tell them, they'll loadtest our service and tell us instead, and suddenly we're defending an SLO that we never intended to.
- we create a useful feedback loop between customer experience and the service team - if we're not meeting our SLOs (or meeting them too easily!), this

- can drive useful prioritization discussions.

It helps us, because

- operations is no longer the “breathable air” of production - work has specific, measurable impact on things the business really cares about;
- we can push back on unreasonable requests or customers - that is, we can defend our production service;
- we know what to page ourselves about now.

Image:
I'm a Happy Button, Why Aren't You?
By LearningLark
https://flic.kr/p/azjW1N

Which data? (2)

You know what I’m going to say, right? We page on thresholds related to our SLOs: symptoms of problems affecting the customer experience.

That’s it; any other paging should be tactical (for example to get focused attention on some new, specific and ongoing bug).

What about stuff that might become critical and page soon? Ticket.

What about the stuff that we **know** will cause us to miss SLOs? Log, probably in our trending system so we can graph it. Automate it so a computer can fix or mitigate the problem instead of a human.

Attach a list of all known “current” **cause** conditions to each page, and a pointer to a dashboard.

Make it tremendously easy for the woken-up engineer to sift through proximate and intermediate causes to find the root(s).

But we don’t page on causes.

Image:
God Appears to Moses in Burning Bush. Painting from Saint Isaac's Cathedral, Saint

Petersburg
By Eugène Pluchart
http://commons.wikimedia.org/wiki/File:Moses_Pluchart.jpg

Page on symptoms.

Page on symptoms.

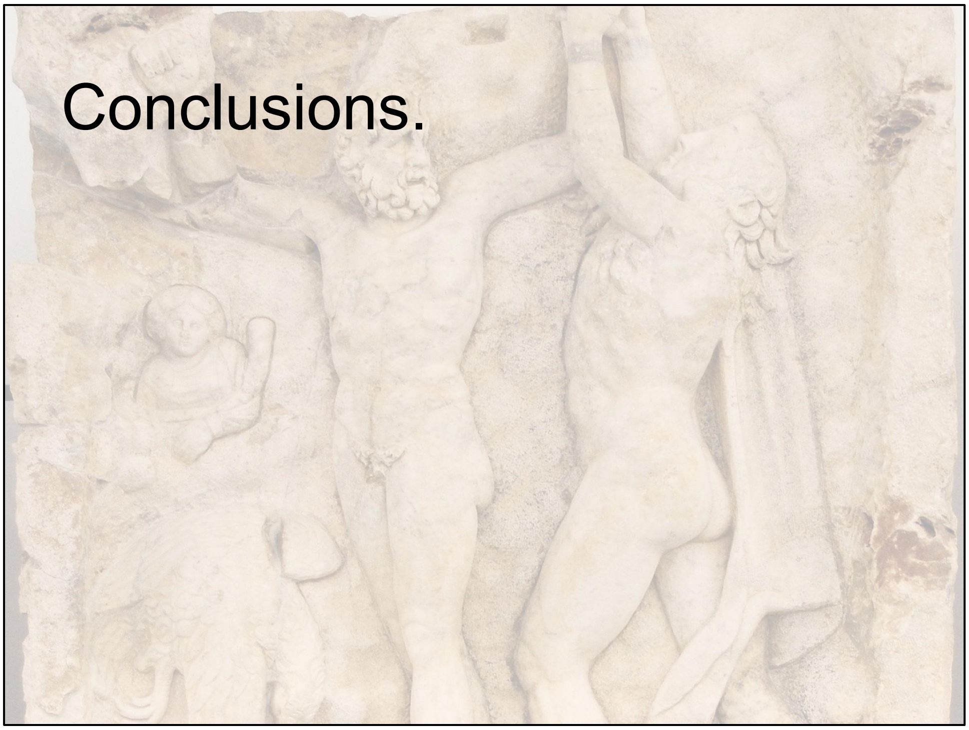
We’re going to have to catch the customer-facing symptoms **anyway**.

Any number of things could cause latency, so if we alert on **those**, we’re going to have a bad time.

Particularly with flaky or ill-defined metrics.

For example, at what threshold will some random storage backend’s unavailability cause us to miss our customer-facing latency SLO?

Image:
Day 59, Project 365 - 12.18.09
By William Brawley
https://flic.kr/p/7oMbJX



Conclusions.

So to recap,

- Good monitoring is something that needs to be designed: organic growth tends to not go so well;
- oncall is the pointy end of production - each shift should be driving change and making things better, not running to stay still;
- customer experience => SLOs & SLIs;
- page on the symptom, not the cause.

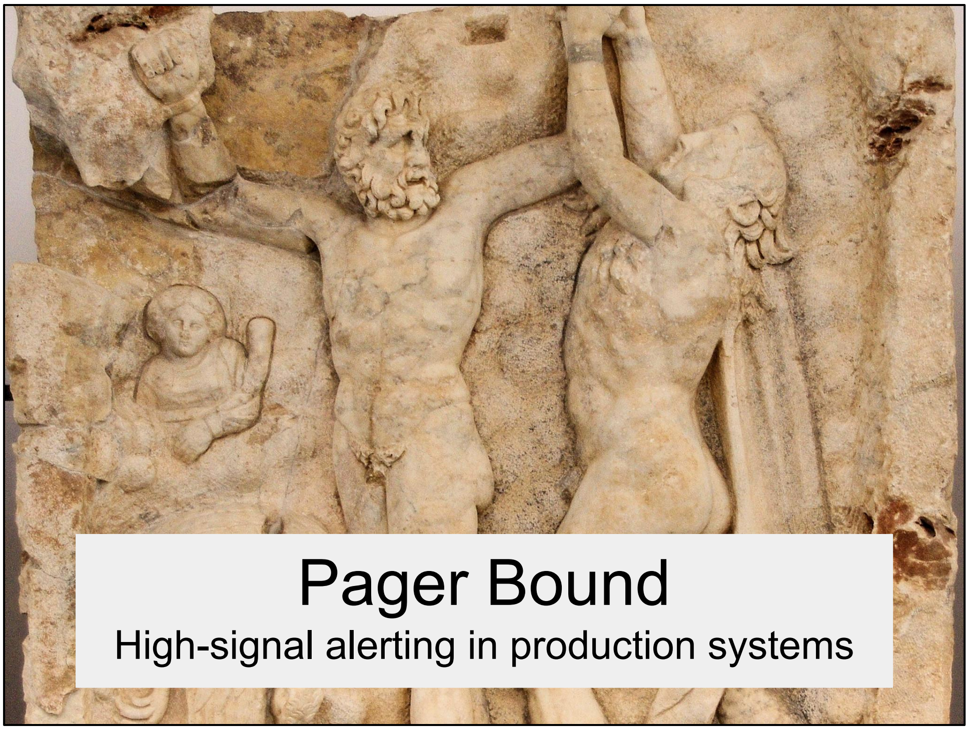
Doing a good job oncall is hard. Designing and building a monitoring system that will support you in that is hard too.

We need to be restless and ambitious when we're oncall. It's the position in which we can most clearly see the full state of our services, what is hurting our customers now, and what we most need to fix in the production environment.

A good shift holding the pager should generate pressure for change: given the rapid increase of the scale at which we work, complacency leads inevitably to a mountain of operational debt.

Image:
Reliefs From The Sebasteion of Aphrodisas: Herakles Frees Prometheus
by Ken and Nyetta

https://flic.kr/p/bELTwa



Pager Bound

High-signal alerting in production systems

Thank you for your time!

Questions?

Image:
Reliefs From The Sebasteion of Aphrodisas: Herakles Frees Prometheus
by Ken and Nyetta
https://flic.kr/p/bELTwa