goto; chicago

Deliver Results, Not Just Releases: Control & Observability in CD

Dave Karow

@davekarow

goto; chicago

Click ‘Rate Session’ to rate session and ask questions.

follow us @gotochgo

Coming up:

- Who is this guy?
- Definitions
- Role Models
- Takeaways
- Q & A

Want a copy of this deck?

split.io/results

Who is this guy?

- Unix geek in the 80’s
- Wrapped apps at Sun in the 90’s
- Ran a developer “forum” back when CompuServe was a thing :-)
- PM for developer tools
- PM for synthetic monitoring
- PM for load testing
- Dev Advocate for “shift left” performance testing
- Evangelist for progressive delivery with automated feedback loops

The future is already here — it's just not very evenly distributed.

William Gibson

Definitions

Continuous Delivery

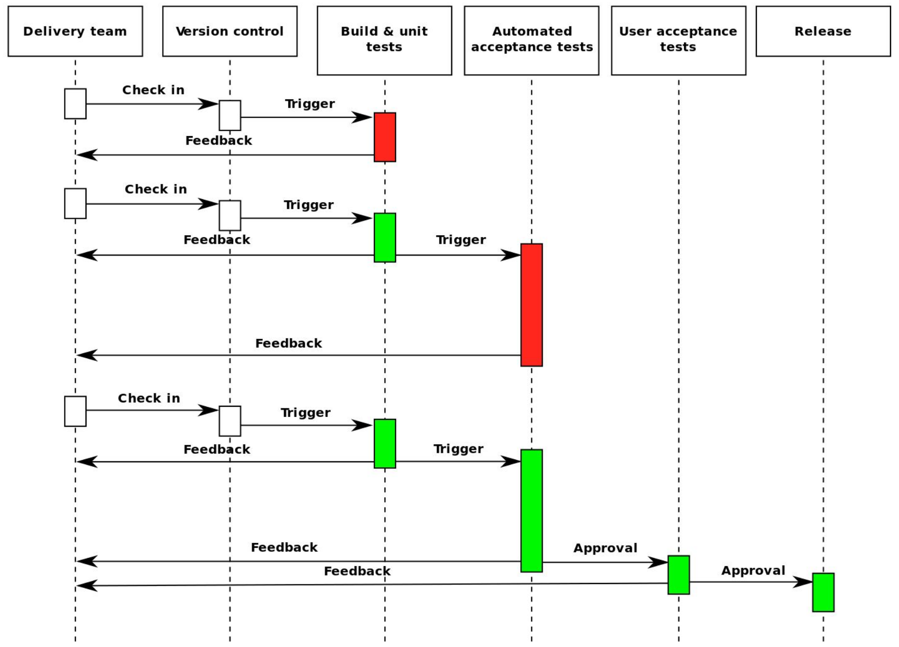
From Jez Humble
https://continuousdelivery.com/

…the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

So what sort of **control** and **observability** are we talking about here?

Control of the CD Pipeline?

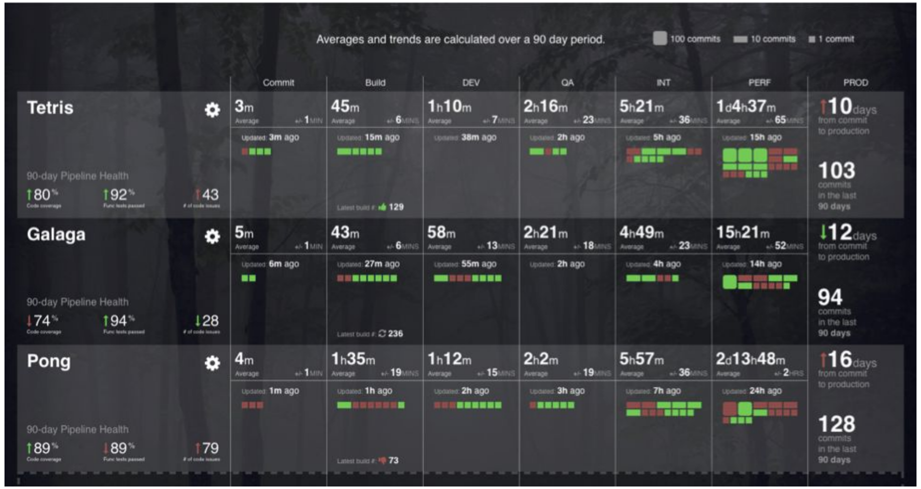
Nope.



Grégoire Détrez, original by Jez Humble [CC BY-SA 4.0]

Observability of the CD Pipeline?

Nope.



Averages and trends are calculated over a 90 day period. 100 commits ▬ 10 commits ■ 1 commit

	Commit	Build	DEV	QA	INT	PERF	PROD
Tetris 90-day Pipeline Health ↑80% Code coverage ↑92% Func tests passed ↑43 # of code issues	3m Average +/- 1 MIN Updated: 3m ago	45m Average +/- 6 MINS Updated: 15m ago Latest build #: 129	1h10m Average +/- 7 MINS Updated: 38m ago	2h16m Average +/- 23 MINS Updated: 2h ago	5h21m Average +/- 36 MINS Updated: 5h ago	1d4h37m Average +/- 65 MINS Updated: 15h ago	↑10 days from commit to production; 103 commits in the last 90 days
Galaga 90-day Pipeline Health ↓74% Code coverage ↑94% Func tests passed ↓28 # of code issues	5m Average +/- 1 MIN Updated: 6m ago	43m Average +/- 6 MINS Updated: 27m ago Latest build #: 236	58m Average +/- 13 MINS Updated: 55m ago	2h21m Average +/- 18 MINS Updated: 2h ago	4h49m Average +/- 23 MINS Updated: 4h ago	15h21m Average +/- 52 MINS Updated: 14h ago	↓12 days from commit to production; 94 commits in the last 90 days
Pong 90-day Pipeline Health ↑89% Code coverage ↓89% Func tests passed ↑79 # of code issues	4m Average +/- 1 MIN Updated: 1m ago	1h35m Average +/- 19 MINS Updated: 1h ago Latest build #: 73	1h12m Average +/- 15 MINS Updated: 2h ago	2h2m Average +/- 19 MINS Updated: 3h ago	5h57m Average +/- 36 MINS Updated: 7h ago	2d13h48m Average +/- 2 HRS Updated: 24h ago	↑16 days from commit to production; 128 commits in the last 90 days

https://hygieia.github.io/Hygieia/product_dashboard_intro.html

If not the **pipeline,** what then?

The **payload**

Whether you call it code, configuration, or change, it’s in the **delivery**, that we “show up” to others.

Control of Exposure

...blast radius
...propagation of goodness
...surface area for learning

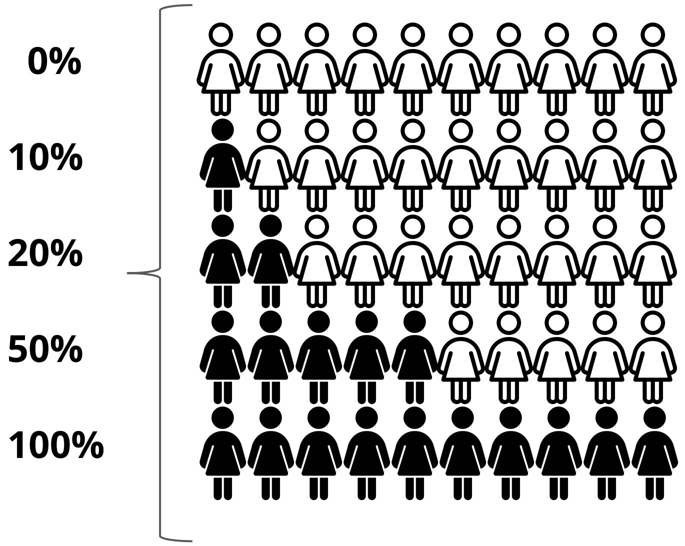
How Do We Make Deploy != Release

and

Revert != Rollback

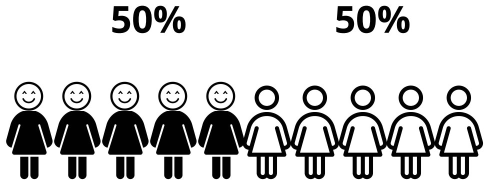
Feature Flag

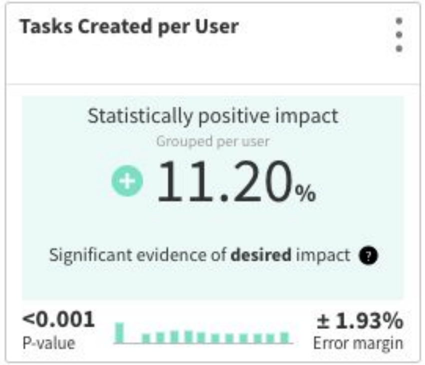
Progressive Delivery Example



Feature Flag

Experimentation Example




Tasks Created per User

Statistically positive impact

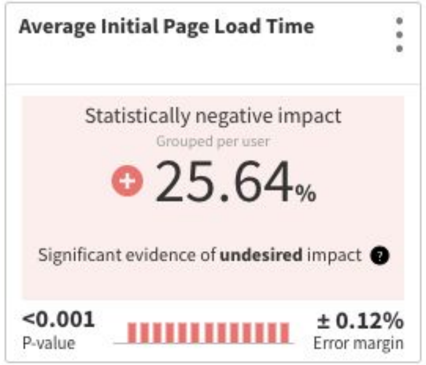
Grouped per user

+ 11.20%

Significant evidence of **desired** impact

<0.001 P-value

± 1.93% Error margin


Average Initial Page Load Time

Statistically negative impact

Grouped per user

+ 25.64%

Significant evidence of **undesired** impact

<0.001 P-value

± 0.12% Error margin

What a Feature Flag Looks Like In Code

Simple “on/off” example:

```
treatment = flags.getTreatment(“related-posts”);
if (treatment == “on”) {
   // show related posts
} else {
   // skip it
}
```

Multivariate example:

```
treatment = flags.getTreatment(“search-algorithm”);
if (treatment == “v1”) {
   // use v1 of new search algorithm
} else if (feature == “v2”) {
   // use v2 of new search algorithm
} else {
   // use existing search algorithm
}
```

Observability of Exposure

Who have we released to so far?

How is it going for them (and us)?

Who Already Does This Well?
(and is generous enough to share how)

LinkedIn
XLNT

LinkedIn early days: a modest start for XLNT

- Built a targeting engine that could “split” traffic between existing and new code
- Impact analysis was by hand only (and took ~2 weeks), so nobody did it :-(

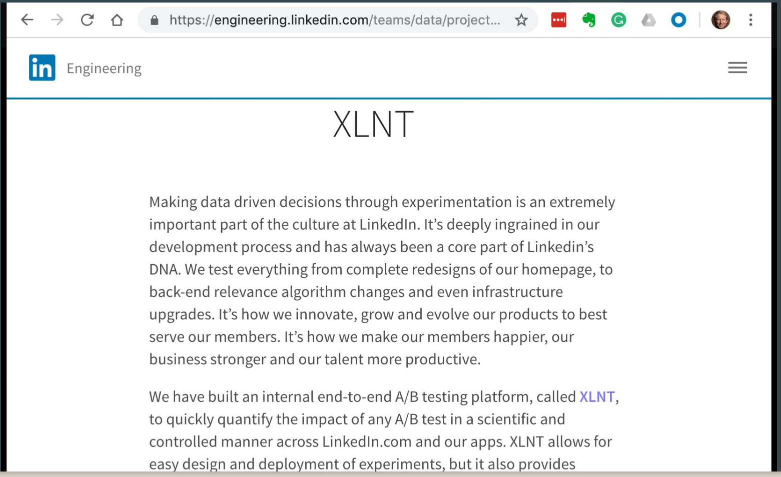
Essentially just feature flags without automated feedback

LinkedIn XLNT Today

A controlled release (with built-in observability) every 5 minutes

100 releases per day

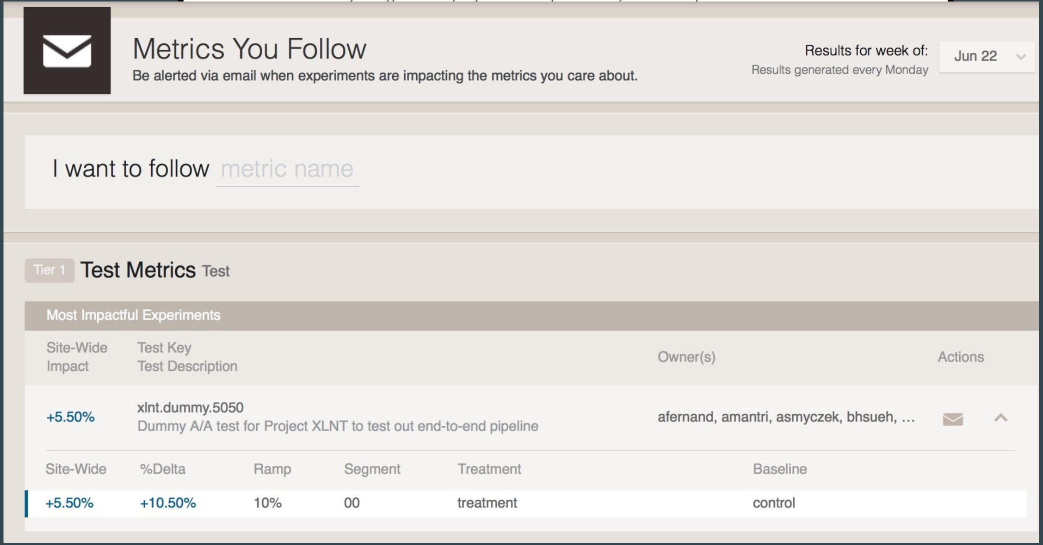
6000 metrics that can be "followed" by any stakeholder: "What releases are moving the numbers I care about?"



XLNT

Making data driven decisions through experimentation is an extremely important part of the culture at LinkedIn. It's deeply ingrained in our development process and has always been a core part of Linkedin's DNA. We test everything from complete redesigns of our homepage, to back-end relevance algorithm changes and even infrastructure upgrades. It's how we innovate, grow and evolve our products to best serve our members. It's how we make our members happier, our business stronger and our talent more productive.

We have built an internal end-to-end A/B testing platform, called XLNT, to quickly quantify the impact of any A/B test in a scientific and controlled manner across LinkedIn.com and our apps. XLNT allows for easy design and deployment of experiments, but it also provides



Metrics You Follow

Be alerted via email when experiments are impacting the metrics you care about.

Results for week of: Jun 22

Results generated every Monday

I want to follow metric name

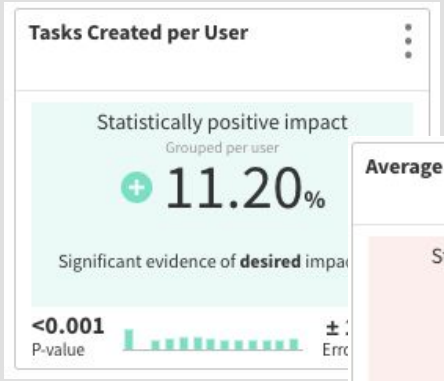
Tier 1 Test Metrics Test

Most Impactful Experiments

Site-Wide Impact	Test Key / Test Description	Owner(s)	Actions
+5.50%	xlnt.dummy.5050 / Dummy A/A test for Project XLNT to test out end-to-end pipeline	afernand, amantri, asmyczek, bhsueh, ...	

Site-Wide	%Delta	Ramp	Segment	Treatment	Baseline
+5.50%	+10.50%	10%	00	treatment	control

Guardrail metrics


Tasks Created per User

Statistically positive impact

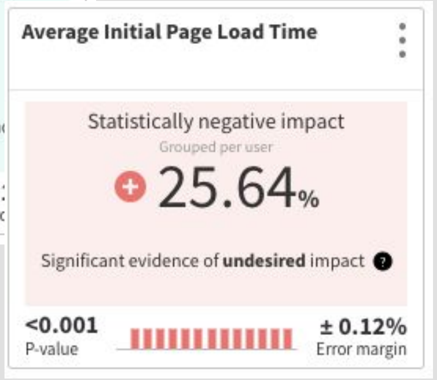
Grouped per user

11.20%

Significant evidence of **desired** impa

<0.001
P-value

±
Erro


Average Initial Page Load Time

Statistically negative impact

Grouped per user

25.64%

Significant evidence of **undesired** impact

<0.001
P-value

± 0.12%
Error margin

Lessons learned at LinkedIn

- Build for scale: no more coordinating over email
- Make it trustworthy: targeting and analysis must be rock solid
- Design for diverse teams, not just data scientists

Ya Xu
Head of Data Science, LinkedIn
Decisions Conference 10/2/2018

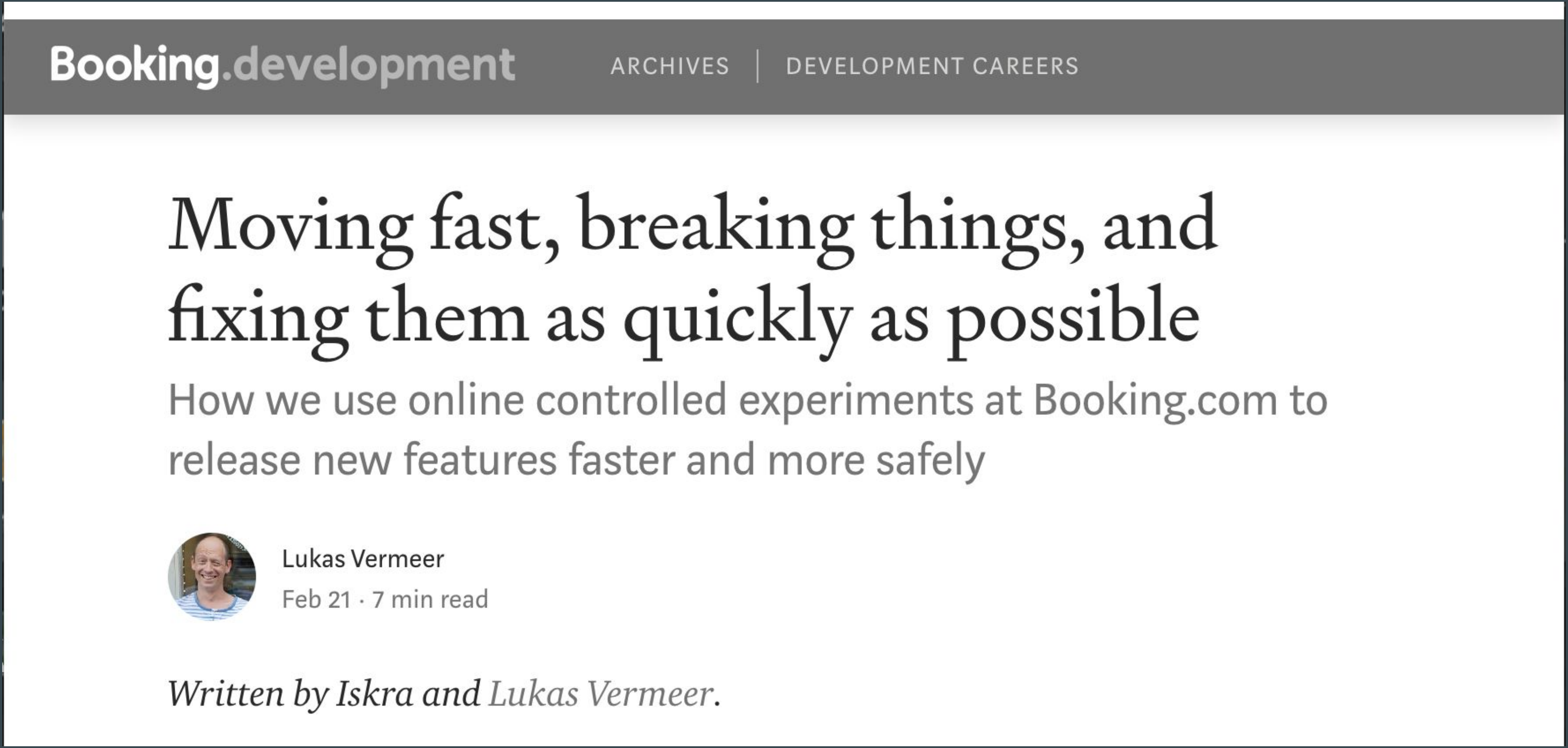
Why does balancing centralization (consistency) and local team control (autonomy) matter?

It increases the odds of achieving results you can trust and observations your teams will act upon.

Booking.com

Booking.com

- EVERY change is treated as an experiment
- 1000 “experiments” running every day
- Observability through two sets of lenses:
  - As a safety net: Circuit Breaker
  - To validate ideas: Controlled Experiments


Booking.development ARCHIVES | DEVELOPMENT CAREERS

Moving fast, breaking things, and fixing them as quickly as possible

How we use online controlled experiments at Booking.com to release new features faster and more safely

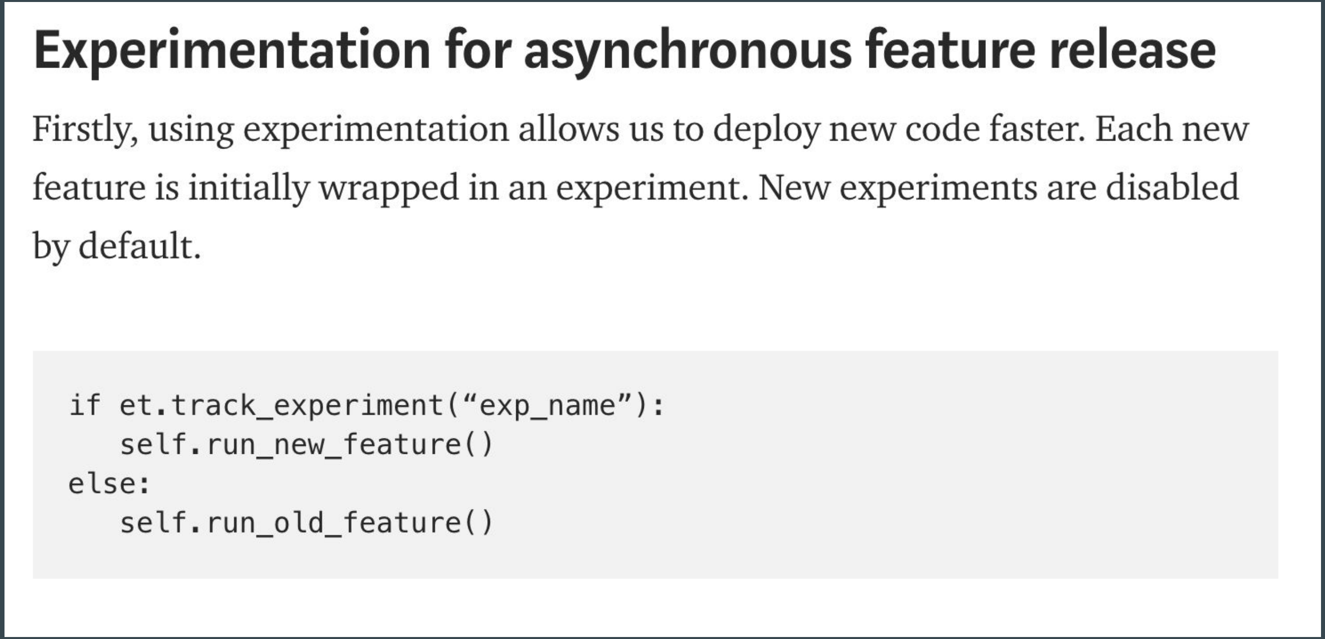
Lukas Vermeer

Feb 21 · 7 min read

Written by Iskra and Lukas Vermeer.

https://medium.com/booking-com-development/moving-fast-breaking-things-and-fixing-them-as-quickly-as-possible-a6c16c5a1185

Booking.com



Experimentation for asynchronous feature release

Firstly, using experimentation allows us to deploy new code faster. Each new feature is initially wrapped in an experiment. New experiments are disabled by default.

```
if et.track_experiment("exp_name"):
   self.run_new_feature()
else:
   self.run_old_feature()
```

Booking.com:
Experimentation for **asynchronous feature release**

- Deploying has no impact on **user experience**
- Deploy **more frequently** with **less risk** to business and users
- The big win is **Agility**

Booking.com:
Experimentation as a **safety net**

- Each new feature is wrapped in its own experiment
- Allows: monitoring and stopping of individual changes
- The **developer or team responsible for the feature** can enable and disable it…
- …**regardless of who deployed the new code that contained it.**

Booking.com: The **circuit breaker**

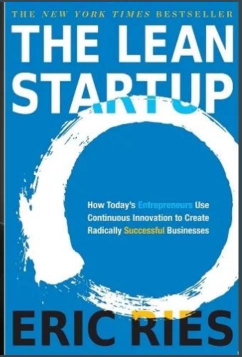
- Active for the first three minutes of feature release
- Severe degradation → automatic abort of **that feature**
- Acceptable divergence from core value of local ownership and responsibility where it's a "no brainer" that users are being negatively impacted

Booking.com: Experimentation as a way to **validate ideas**

- Measure (in a controlled manner) the impact changes have on user behaviour
- Every change has a clear objective (explicitly stated hypothesis on how it will improve user experience)
- Measuring allows validation that desired outcome is achieved

Booking.com: Experimentation to **learn faster**

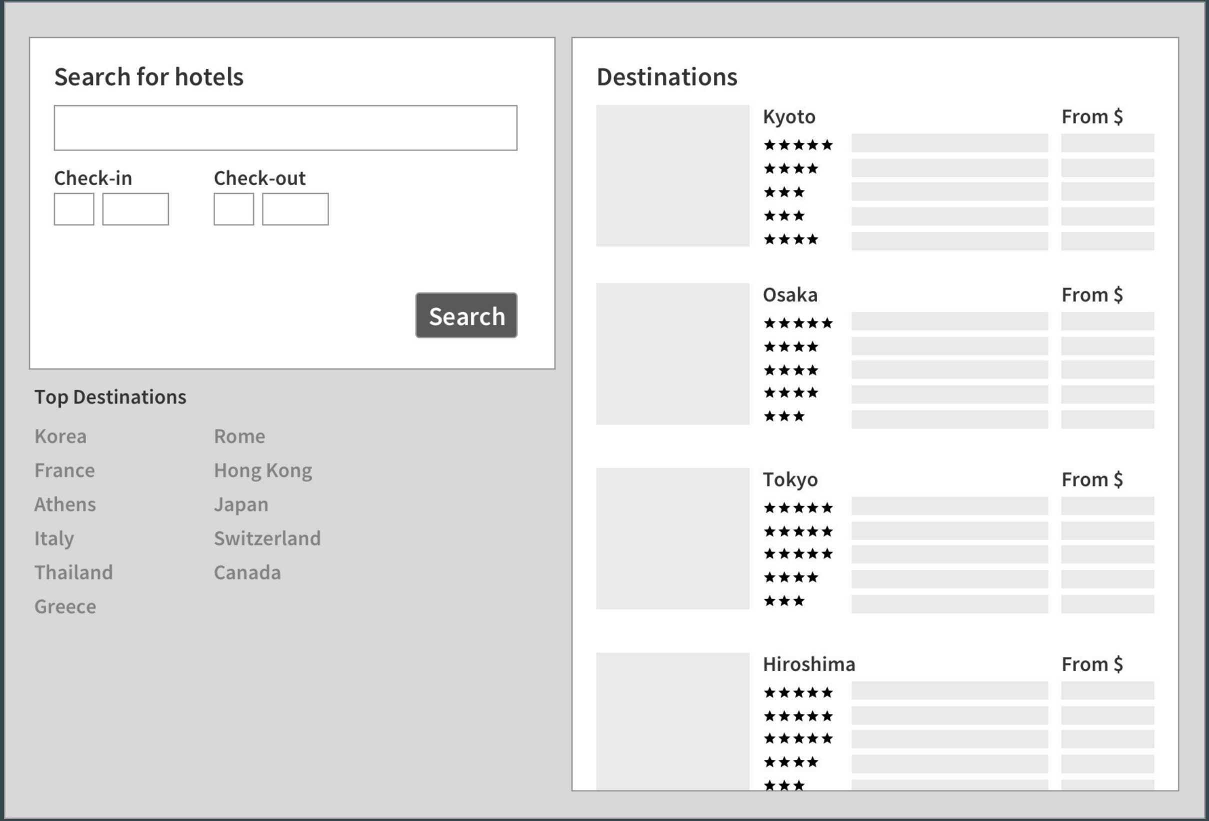
Instead of making complex plans that are based on a lot of assumptions, you can make constant adjustments with a steering wheel called the Build-Measure-Learn feedback loop.


THE NEW YORK TIMES BESTSELLER
THE LEAN STARTUP
How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses
ERIC RIES

The **quicker** we manage to validate new ideas
the **less time is wasted** on things that don't work
and the **more time is left to work on things that make a difference.**

In this way, **experiments also help us decide what we should ask, test and build next**.

Lukas Vermeer's tale of **humility**



Search for hotels

Check-in

Check-out

Search

Top Destinations

Korea	Rome
France	Hong Kong
Athens	Japan
Italy	Switzerland
Thailand	Canada
Greece	

Destinations

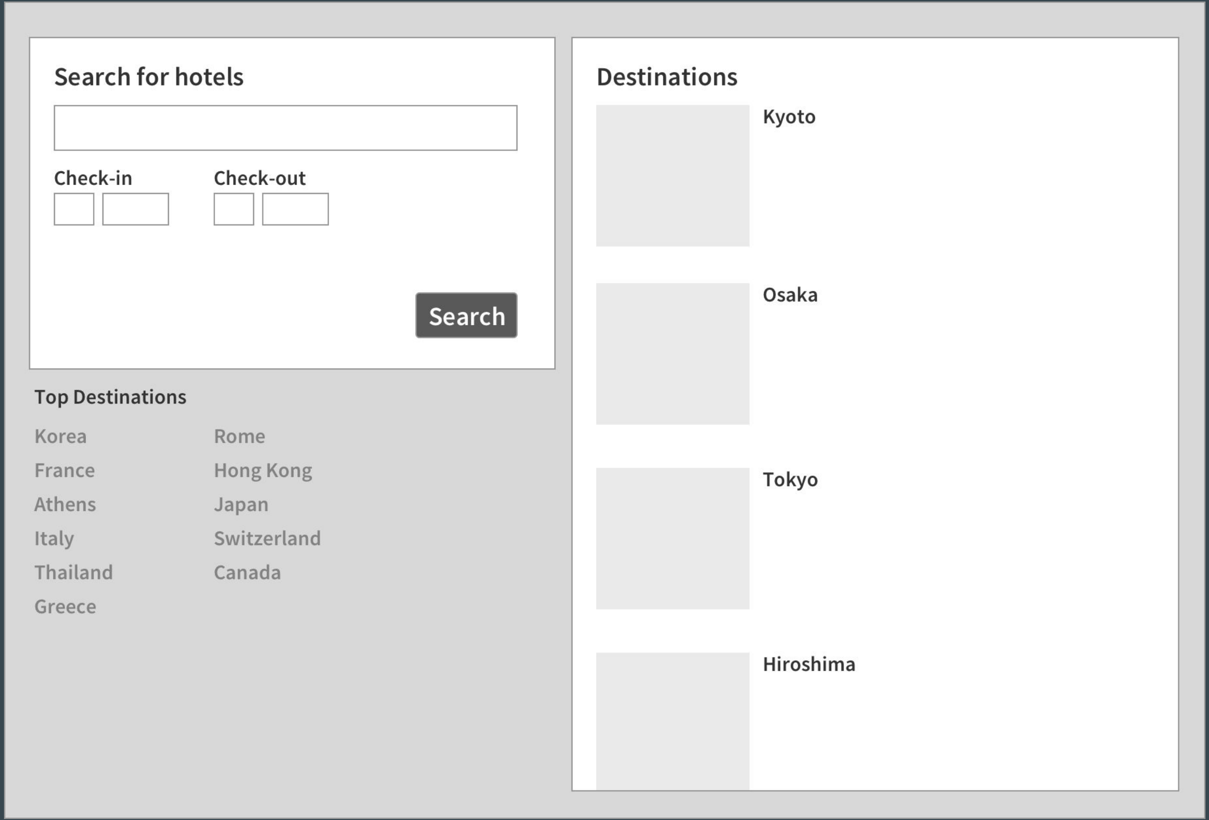
Kyoto — From $
★★★★★
★★★★
★★★
★★★
★★★★

Osaka — From $
★★★★★
★★★★
★★★★
★★★★
★★★

Tokyo — From $
★★★★★
★★★★★
★★★★★
★★★★
★★★

Hiroshima — From $
★★★★★
★★★★★
★★★★★
★★★★
★★★

Lukas Vermeer's tale of **humility**



Search for hotels

Check-in

Check-out

Search

Top Destinations

Korea	Rome
France	Hong Kong
Athens	Japan
Italy	Switzerland
Thailand	Canada
Greece	

Destinations

Kyoto

Osaka

Tokyo

Hiroshima

Facebook Gatekeeper

Taming Complexity

States

Interdependencies

Uncertainty

Irreversibility

Taming Complexity with Reversibility

KENT BECK · MONDAY, JULY 27, 2015

https://www.facebook.com/notes/1000330413333156/

Taming Complexity

States

Interdependencies

Uncertainty

Irreversibility

- **Internal usage.** Engineers can make a change, get feedback from thousands of employees using the change, and **roll it back in an hour.**
- **Staged rollout.** We can begin deploying a change to a billion people and, **if the metrics tank, take it back before problems affect most people using Facebook.**
- **Dynamic configuration.** If an engineer has planned for it in the code, we can turn off an offending feature in production in seconds. Alternatively, **we can dial features up and down in tiny increments (i.e. only 0.1% of people see the feature) to discover and avoid non-linear effects.**
- **Correlation.** Our correlation tools let us easily see the unexpected consequences of features so we know to turn them off even when those consequences aren't obvious.

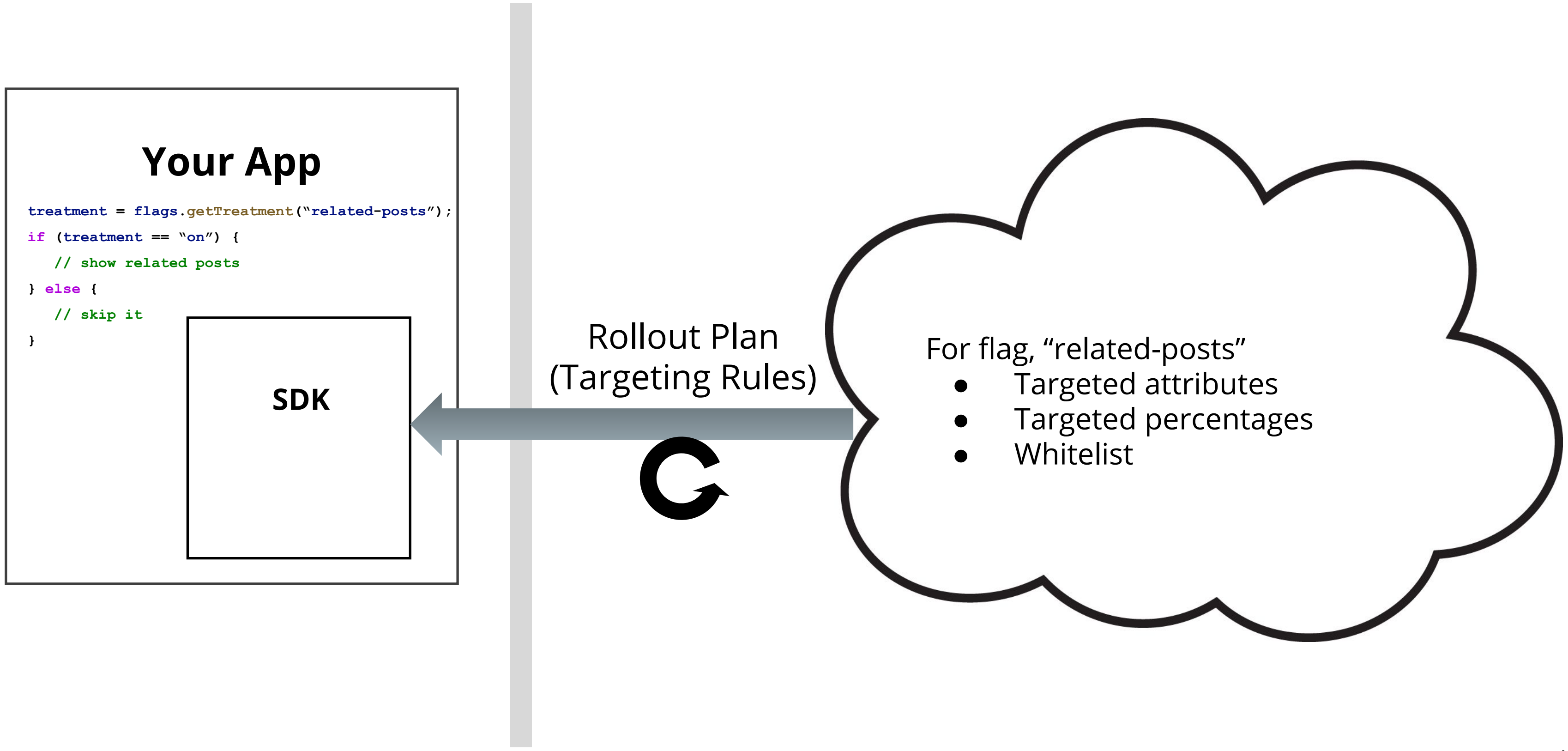
Taming Complexity with Reversibility
KENT BECK· JULY 27, 2015
https://www.facebook.com/notes/1000330413333156/

Takeaways

#1 Decouple Deployment from Release

Deploy is infra
Release is exposing bits to users

Sample Architecture and Data Flow



```
treatment = flags.getTreatment("related-posts");
if (treatment == "on") {
   // show related posts
} else {
   // skip it
}
```

For flag, "related-posts"
- Targeted attributes
- Targeted percentages
- Whitelist

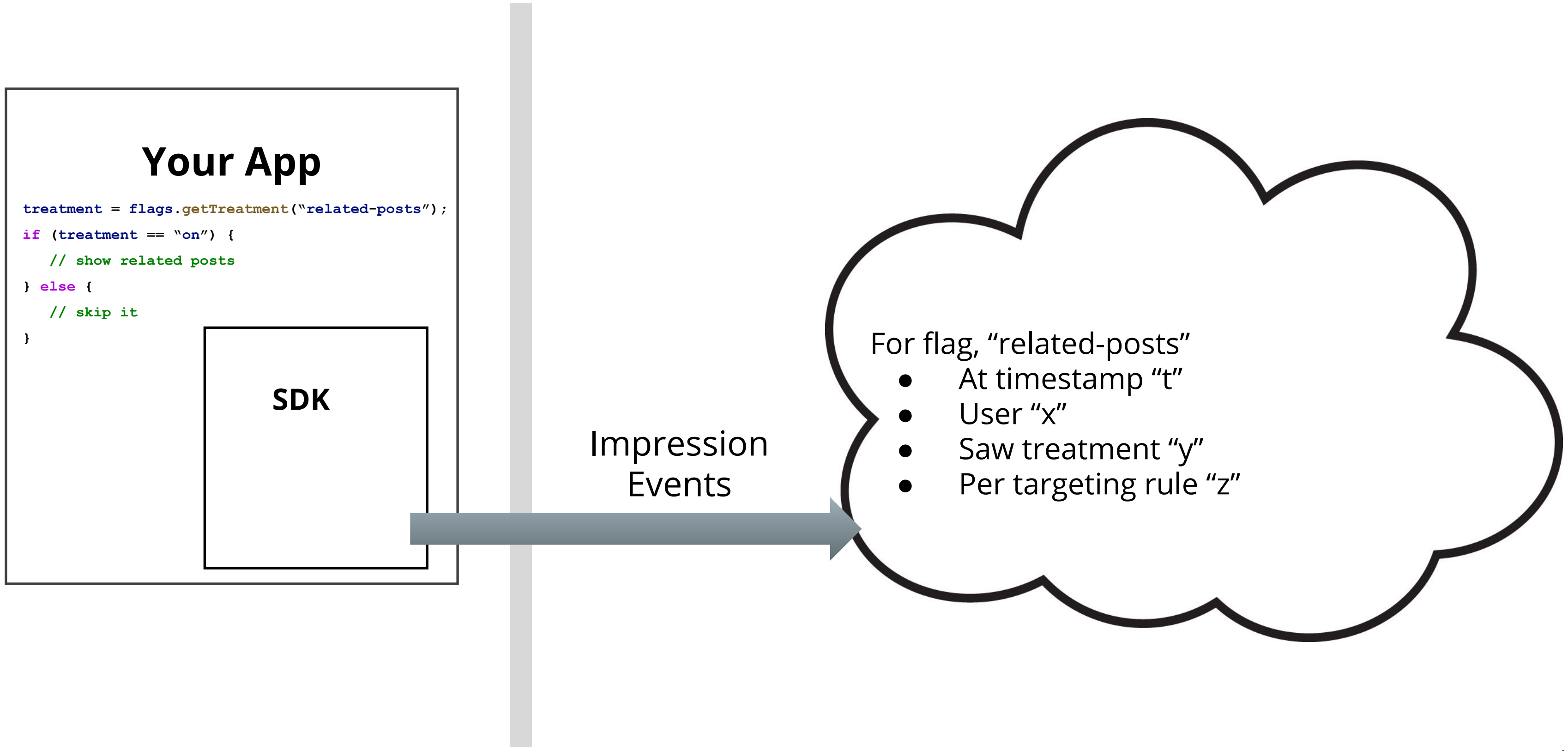
Where should you implement progressive delivery controls: front end or back end?

Favor the back-end, but put them as close to the location of “facts” you’ll use for decisions as possible.

#2 Build-In Observability

Know what’s rolling out, who is getting what, and why
Align metrics to control plane to learn faster

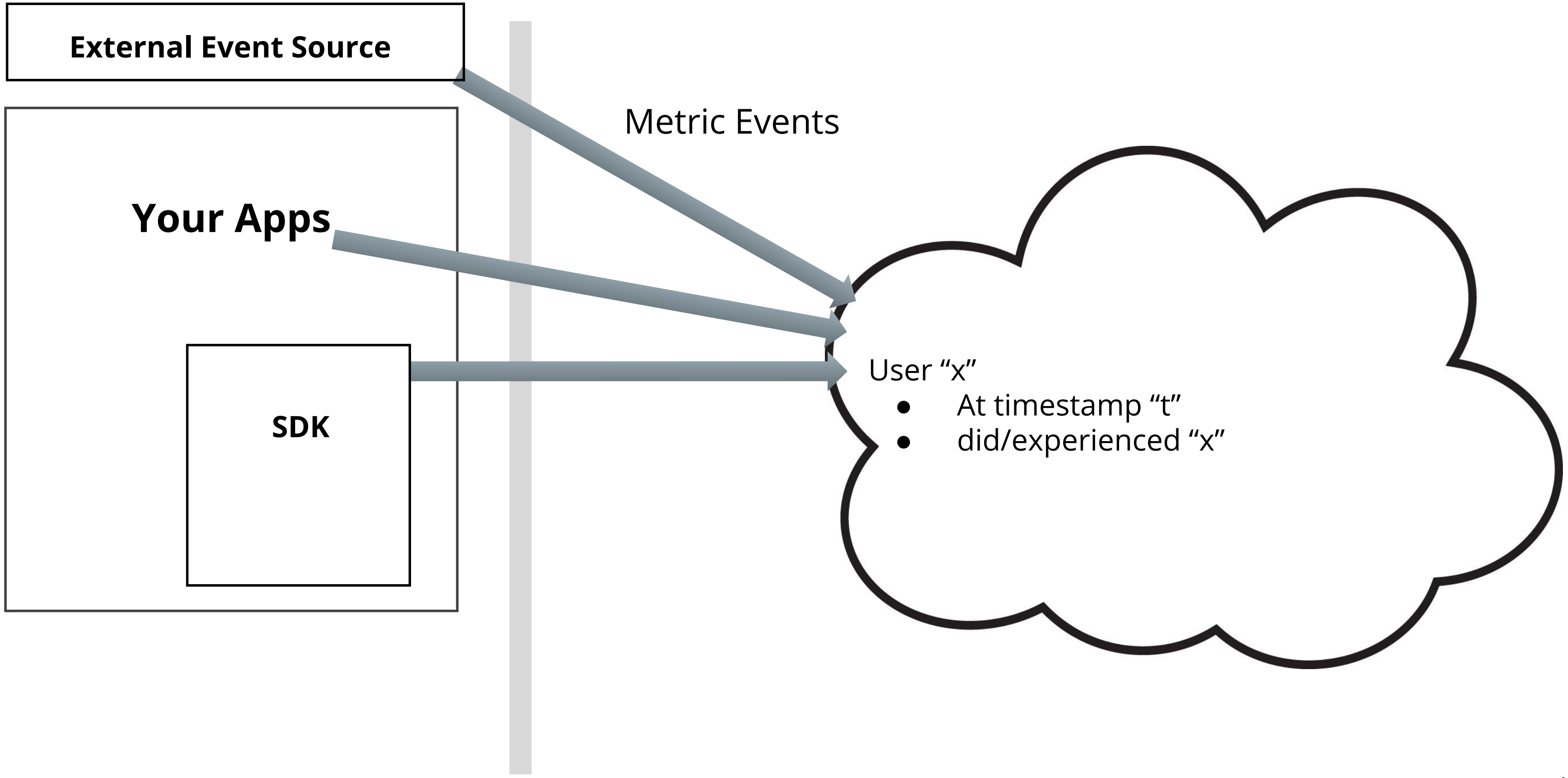
Sample Architecture and Data Flow



```
treatment = flags.getTreatment("related-posts");
if (treatment == "on") {
   // show related posts
} else {
   // skip it
}
```

For flag, "related-posts"
- At timestamp "t"
- User "x"
- Saw treatment "y"
- Per targeting rule "z"

Sample Architecture and Data Flow



What two pieces of data make it possible to attribute system and user behavior changes to any deployment?

1. unique_id (same user/account id evaluated by the feature flag decision engine.)
2. timestamp of the observation.

#3 Going beyond MVP yields significant benefits

Build for scale: solve for chaos
Make it trustworthy: make it stick
Design for diverse audiences: one source of truth

Q&A

Common Questions

How does this compare to “A/B testing”?

How long should feature flags stick around? Don’t they increase tech debt/dead code?

What’s the connection between feature flags and trunk based development?

Doesn’t testing become harder when there are many flags in the code?

“Whatever you are, be a good one.”

Want a copy of this deck?

split.io/results

goto; chicago

Please

Remember to rate this session

Thank you!

follow us @gotochgo