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

...

@davekarow

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

William Gibson

Coming up:

- What a Long Strange Trip It's Been
- Definitions
- Stories From Role Models
- Summary Checklist

What a long, strange trip it's been...

- Punched my first computer card at age 5
- Happy accident: Unix geek in the 80's
- Wrapped apps at Sun in the 90's to modify execution on the fly
- PM for developer tools
- PM for synthetic monitoring
- PM for load testing
- Dev Advocate for "shift left" performance testing
- Evangelist for progressive delivery & "built in" feedback loops

Definitions

Continuous Delivery

From Jez Humble
https://continuousdelivery.com/

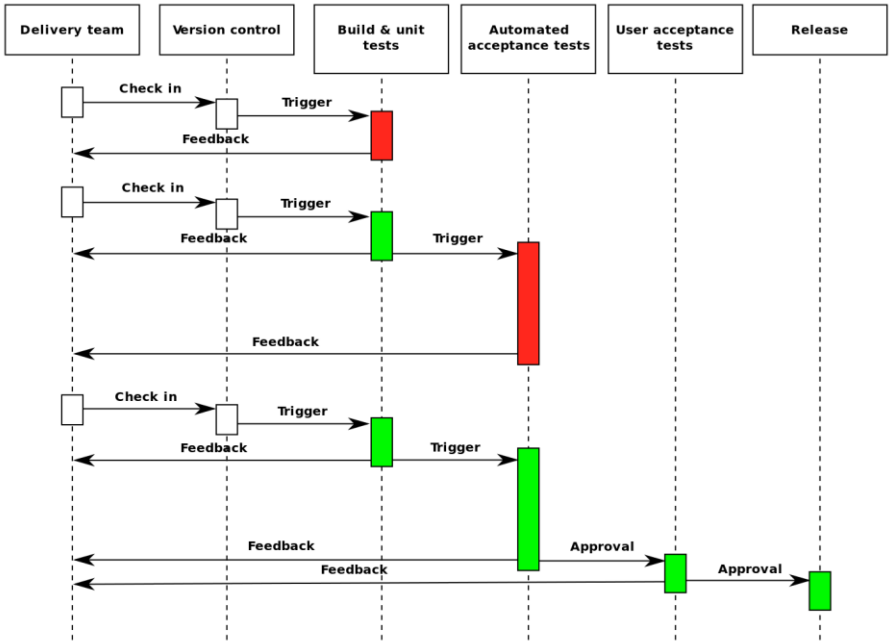
...the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

___

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

Control of the CD Pipeline?

Nope.



Observability of the CD Pipeline?

Nope.

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

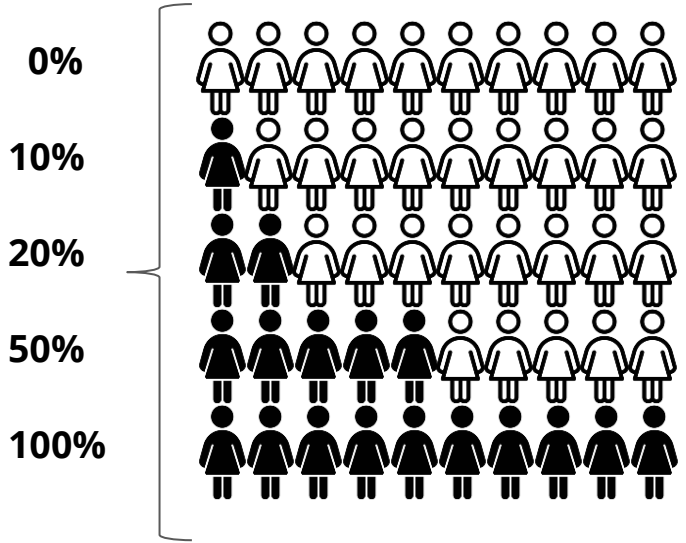
How Do We Make Deploy != Release

and

Revert != Rollback

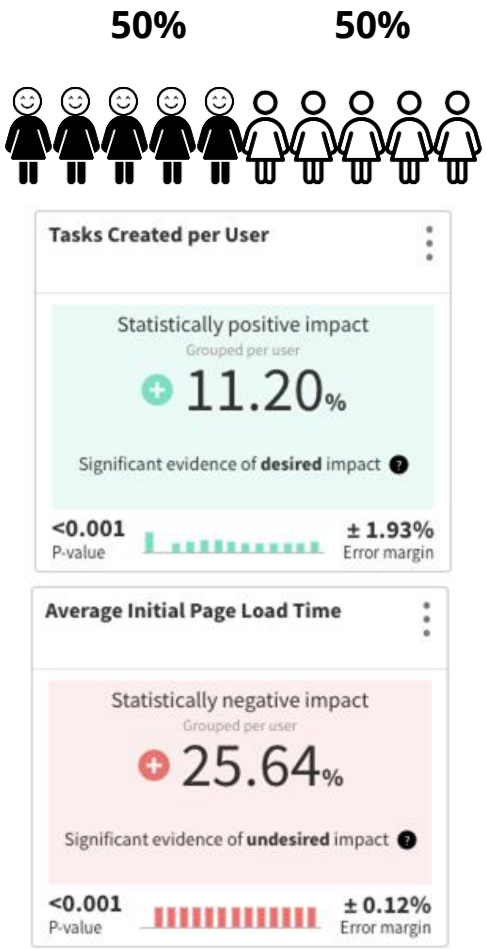
Feature Flag

Progressive Delivery Example



Feature Flag

Experimentation Example



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 :-(

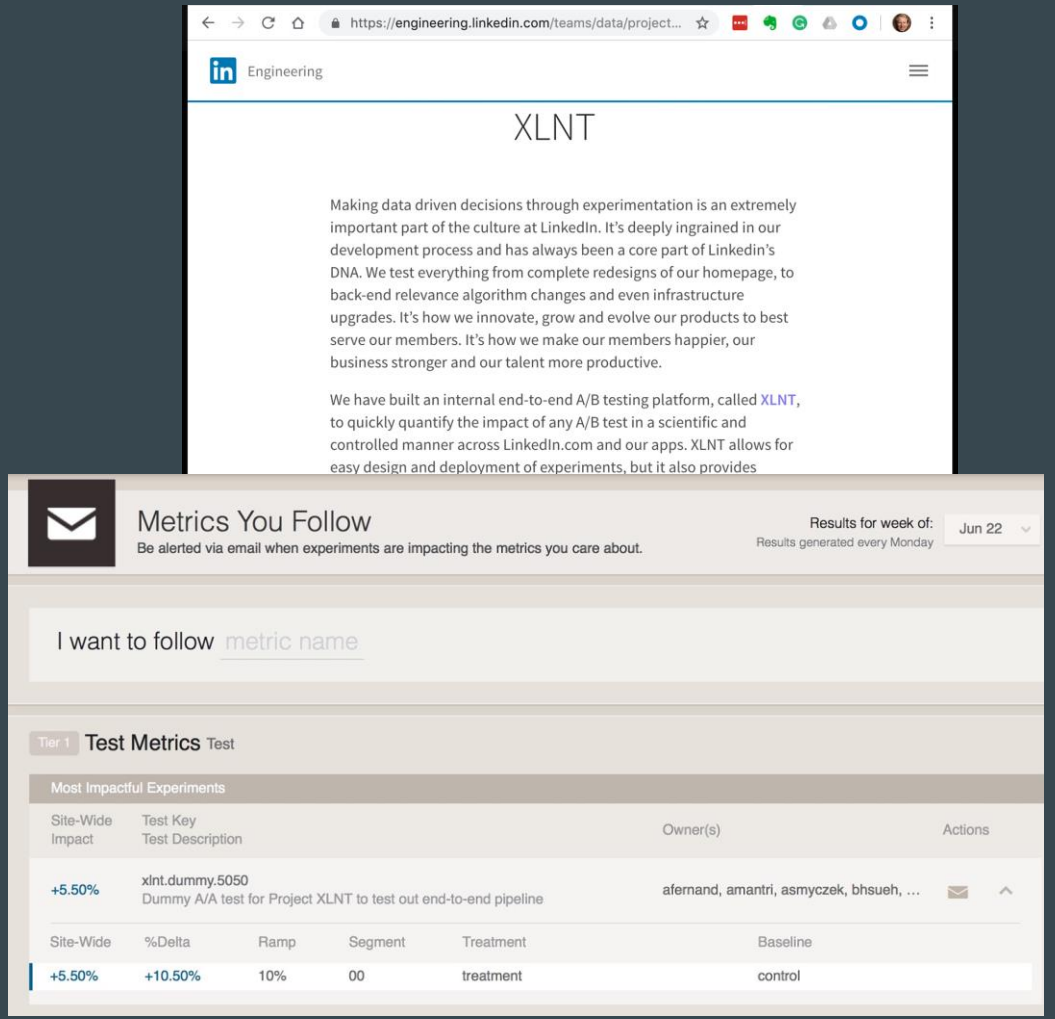
Essentially just feature flags without automated feedback

LinkedIn XLNT Today

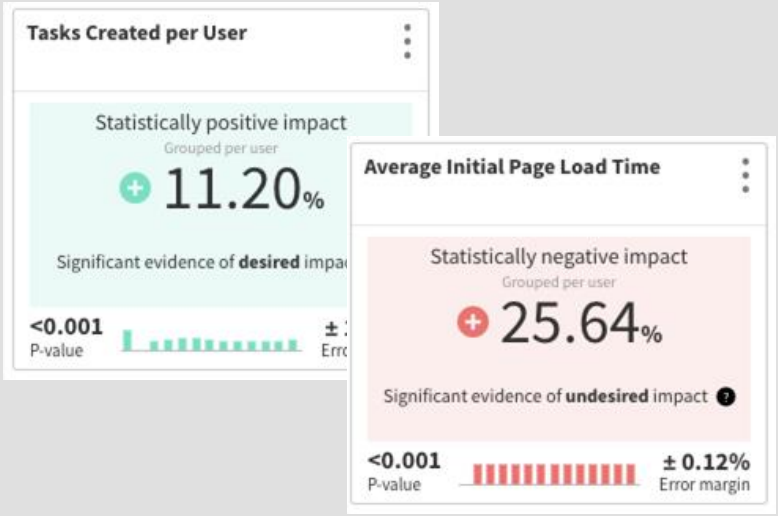
A controlled release (with built-in observability) every 5 minutes

100 releases per day

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



Guardrail metrics



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**

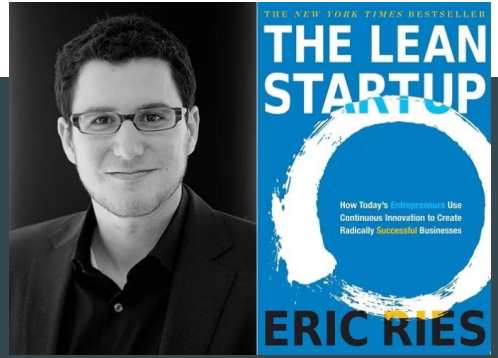
- Active for the first three minutes of feature release
- Severe degradation → automatic abort of **that feature**
- Acceptable divergence from core value of local ownership and responsibility where it's a "no brainer" that users are being negatively impacted

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**

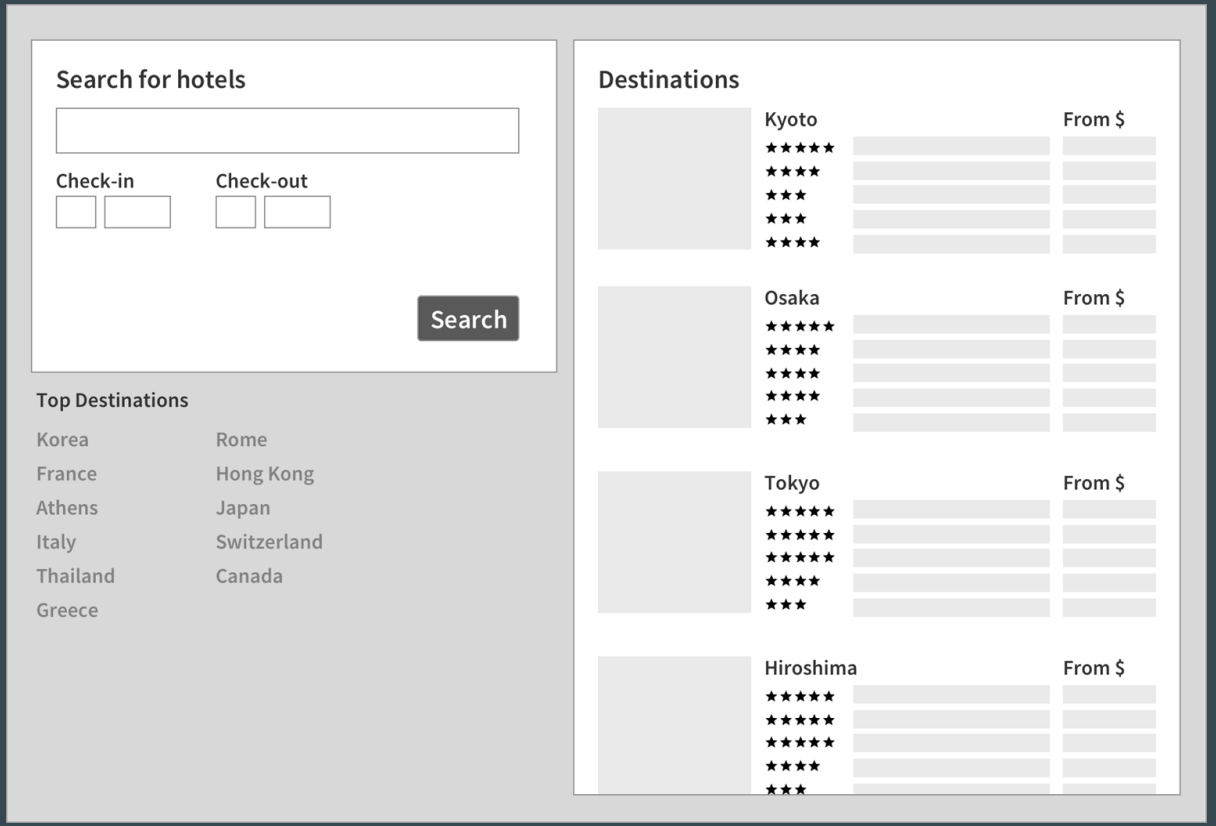
Instead of making complex plans that are based on a lot of assumptions, you can make constant adjustments with a steering wheel called the Build-Measure-Learn feedback loop.



The **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**



Lukas Vermeer's tale of **humility**

Search for hotels

Check-in

Check-out

Search

Top Destinations

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

Destinations

Kyoto

Osaka

Tokyo

Hiroshima

Facebook Gatekeeper

Taming Complexity

States

Interdependencies

Uncertainty

Irreversibility

https://www.facebook.com/notes/10003304133331 56/

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/

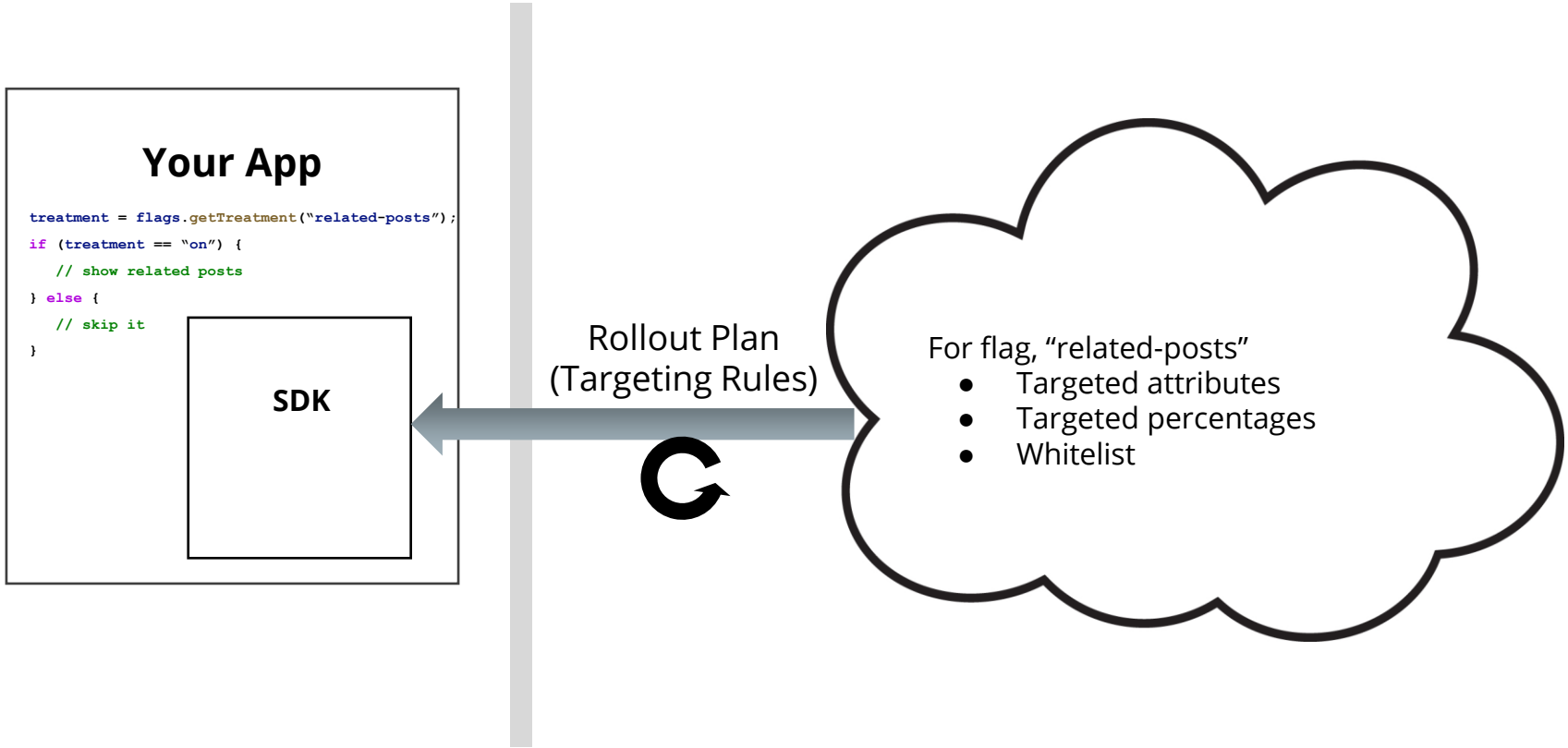
Summary Checklist: Three Foundational Pillars & Two Key Use Cases

Foundational Pillar #1

Decouple deploy (moving code into production) from release (exposing code to users)

- ❏ Allow changes of exposure w/o new deploy or rollback
- ❏ Support targeting by UserID, attribute (population), random hash

Pillar #1: Sample Architecture and Data Flow

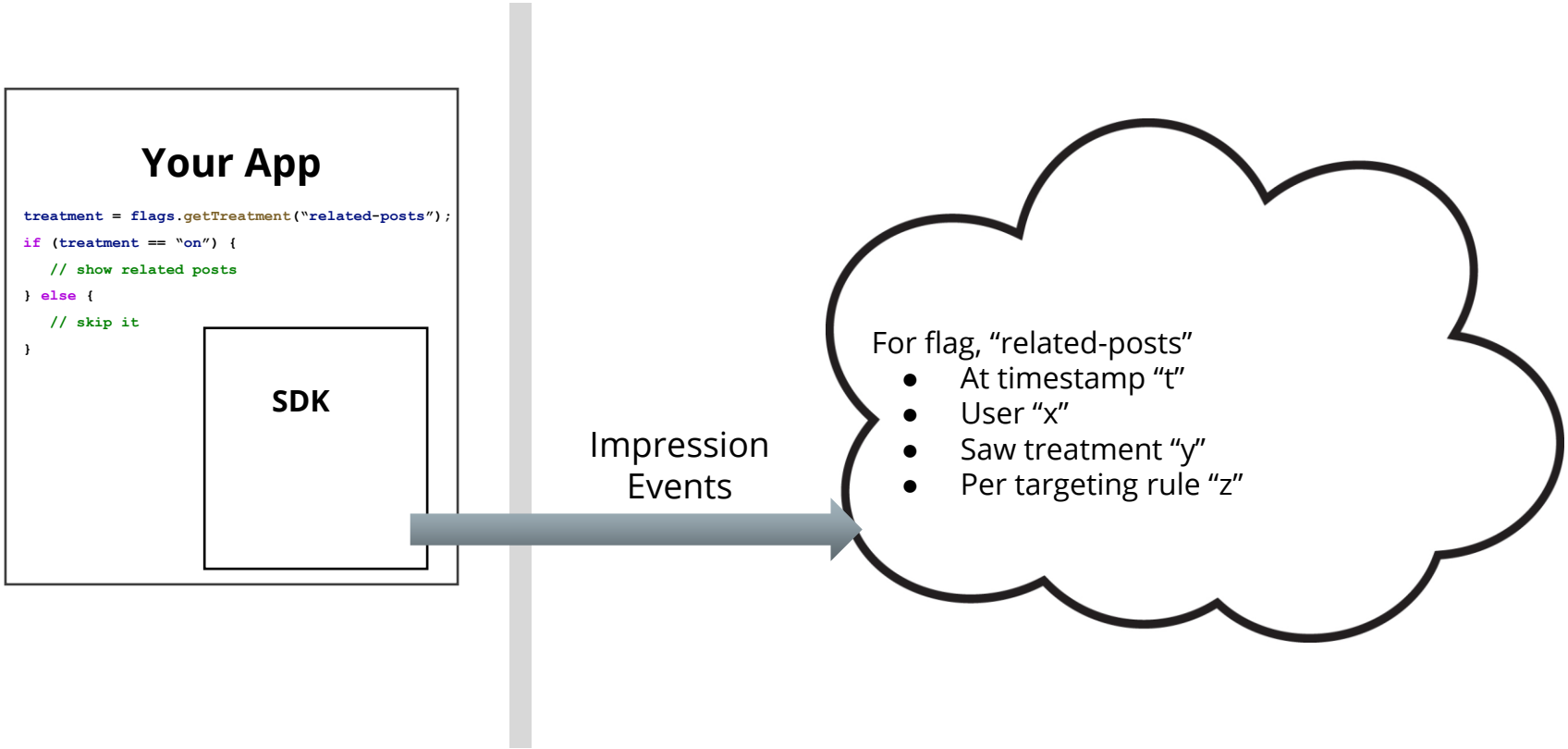


Foundational Pillar #2

Automate a reliable and consistent way to answer, “Who have we exposed this code to so far?”

- ❑ Record who hit a flag, which way they were sent, and why.
- ❑ Confirm that targeting is working as intended
- ❑ Confirm that expected traffic levels are reached

Pillar #2: Sample Architecture and Data Flow

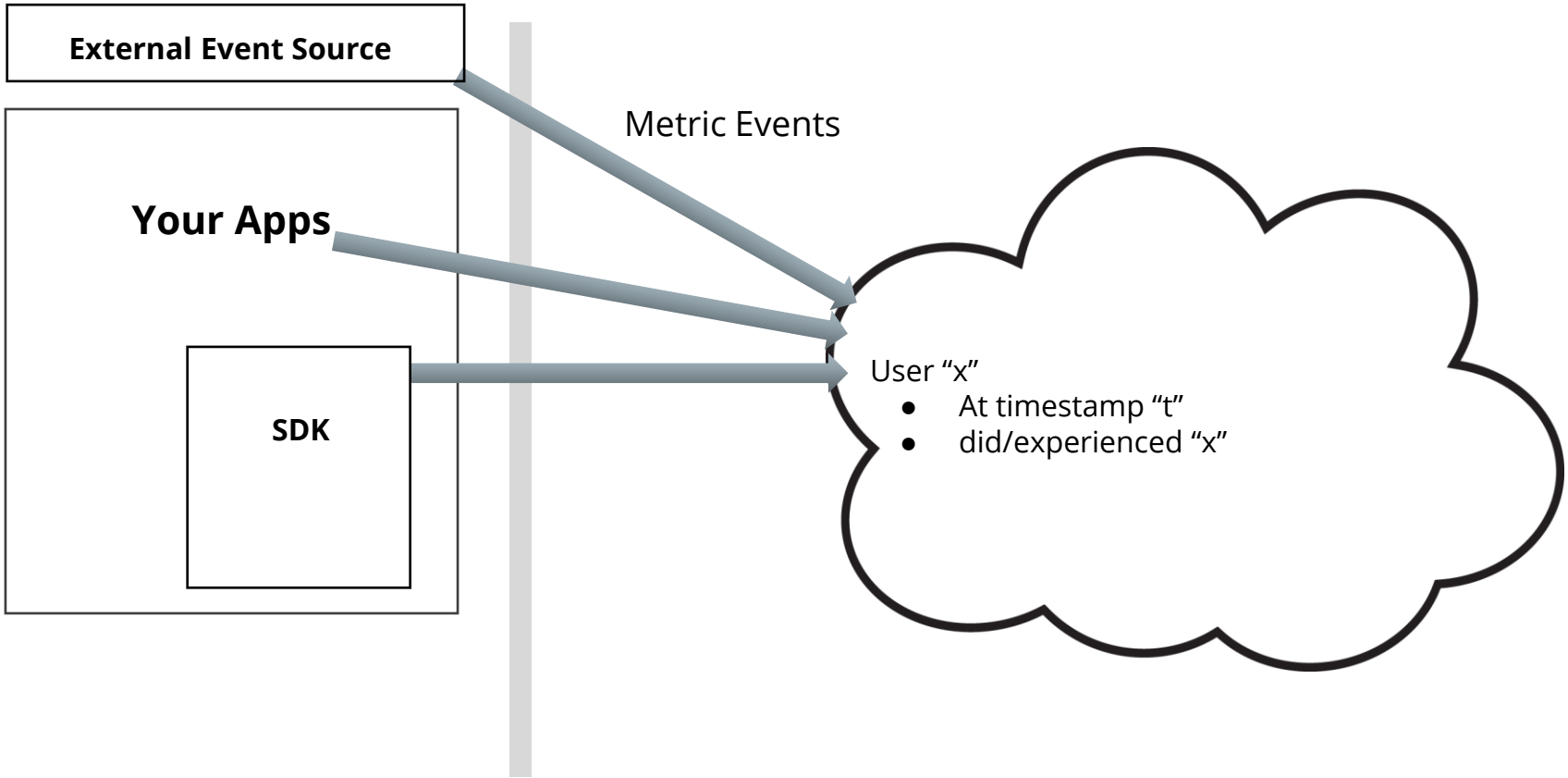


Foundational Pillar #3

Automate a reliable and consistent way to answer, "How is it going for them (and us)?"

- ❑ Automate comparison of system health (errors, latency, etc…)
- ❑ Automate comparison of user behavior (business outcomes)
- ❑ Make it easy to include "Guardrail Metrics" in comparisons to avoid the local optimization trap

Pillar #3: Sample Architecture and Data Flow



Use Case #1: Release Faster With Less Risk

Limit the blast radius of unexpected consequences so you can replace the “big bang” release night with more frequent, less stressful rollouts.

Build on the three pillars to:

- ❏ Ramp in stages, starting with dev team, then dogfooding, then % of public
- ❏ Monitor at feature rollout level, not just globally (vivid facts vs faint signals)
- ❏ Alert at the team level (build it/own it)
- ❏ Kill if severe degradation detected (stop the pain now, triage later)
- ❏ Continue to ramp up healthy features while “sick” are ramped down or killed

Use Case #2: Engineer for Impact (Not Output)

Focus precious engineering cycles on “what works” with experimentation, making statistically rigorous observations about what moves KPIs (and what doesn’t).

Build on the three pillars to:

- ❏ Target an experiment to a specific segment of users
- ❏ Ensure random, deterministic, persistent allocation to A/B/n variants
- ❏ Ingest metrics chosen before the experiment starts (not cherry-picked after)
- ❏ Compute statistical significance before proclaiming winners
- ❏ Design for diverse audiences, not just data scientists (buy-in needed to stick)

Whatever you are,
try to be a good one.

William Makepeace Thackeray