

Using A/B Testing

to improve our apps

Good morning! I hope you've had your coffee! Let's talk about A/B Testing.

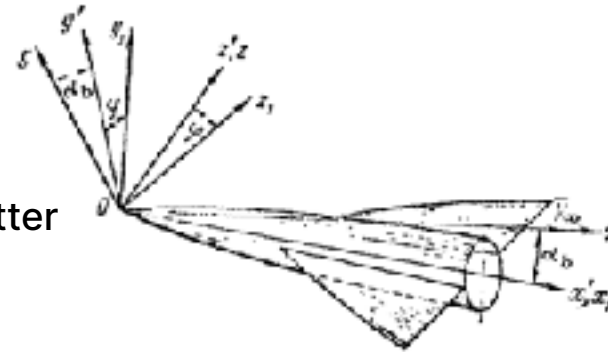
Hi!

Making our apps better

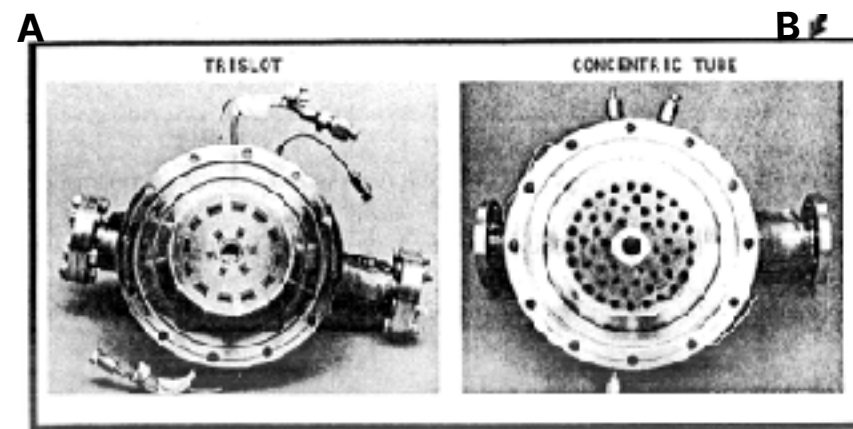
The Theory

Some Code

Common Pitfalls



- This talk is about making our apps better,
- about p-values and confidence scores and things to keep in mind
- We will look at some example code on how to run a/b testing experiments, and calculate the results
- and we'll look at common pitfalls that make a/b testing experiments less valuable if you don't consider them



Sources, slides and text

github.com/TelemetryDeck/ab-testing-talk


You don't need to take notes. I'll upload all slides to this URL. And at the end of the talk, I will show this URL again.

ORIGINAL PAGE IS
OF POOR QUALITY



Fig. 25. Air-breathing cosmic aircraft.



 TelemetryDECK

Quick intro: hi, I'm Daniel! I'm a mobile developer, a CTO, and a data wrangler for TelemetryDeck, a privacy-first analytics service for mobile apps, web apps and desktop apps.

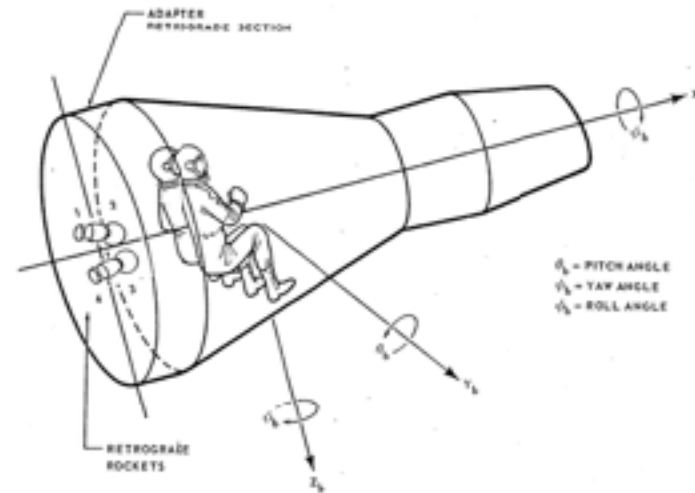
I love space, that's why you'll see lots of NASA technical diagrams on these slides. There's usually a connection to the slide topic, but don't try to read into them too much. They're just there to look really nice and cool and stuff.

Make App Better = Good

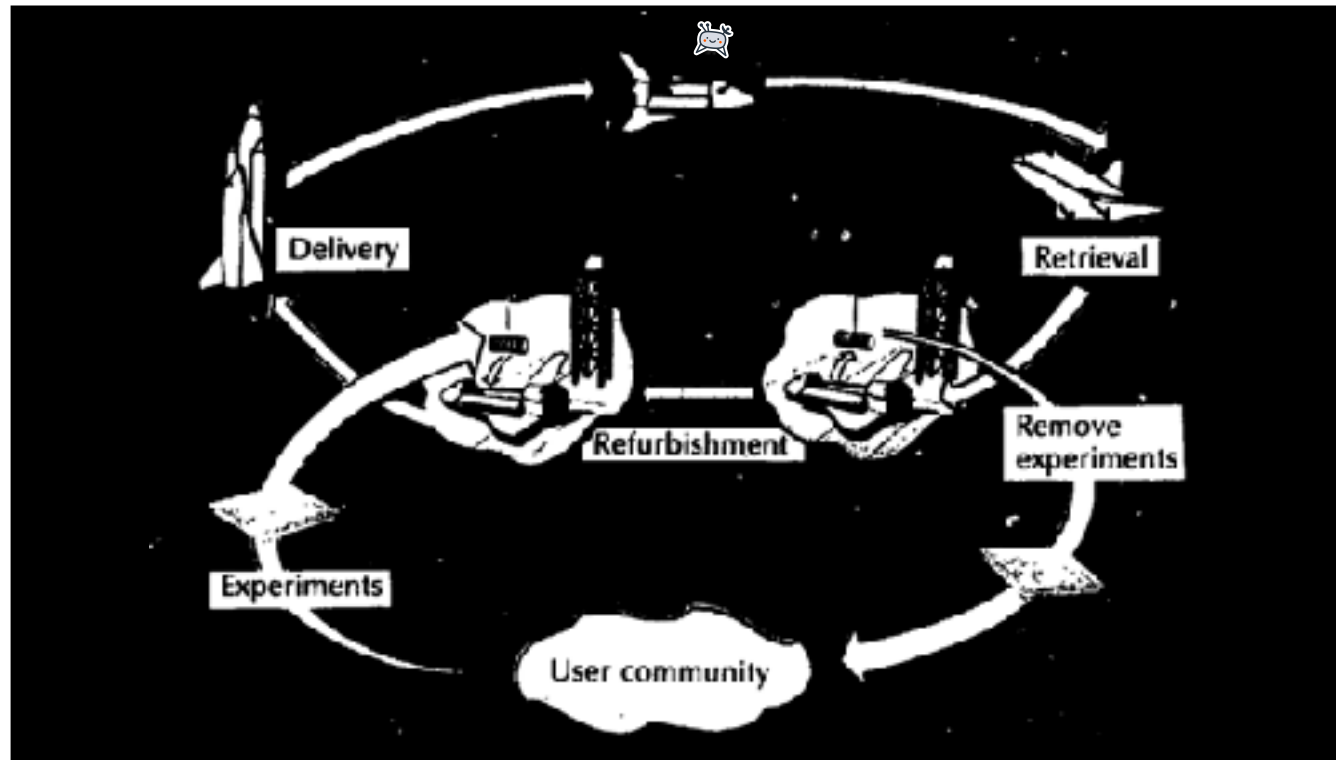
Revenue

Work load

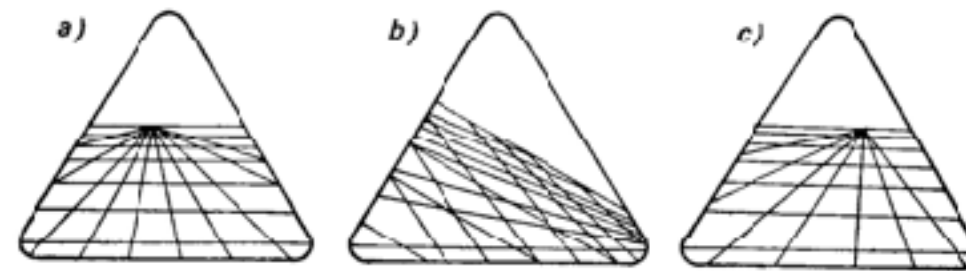
Happiness



- We need to continuously improve our app's user interface and user experience. Why?
- That leads to users flowing better towards in app purchase and improves conversion rates
- If users are less confused about our apps, that also means less support load
- And good apps just make people happy! Happy customers use ours apps more, give better App Store ratings, recommend us to their friends, and give us a warm glowy feeling in our bellies. Well, they do for me.



- There are many ways to improve our apps and find the places where people drop out or experience friction in our apps
 - We'll hear from your users directly
 - Usage data and telemetry data can tell us where people are experiencing problems
 - We can iterate version-by-version over problems until we get it right
- A/B Testing should also be in the toolbox for large and small developers as part of continuous optimization!
- Because it allows us to quickly test out ideas and improve our apps in a MEASURABLE, Data-Driven Way!



Concepts

Let's get us all on the same page on how A/B tests work, by refreshing the various concepts that we need.

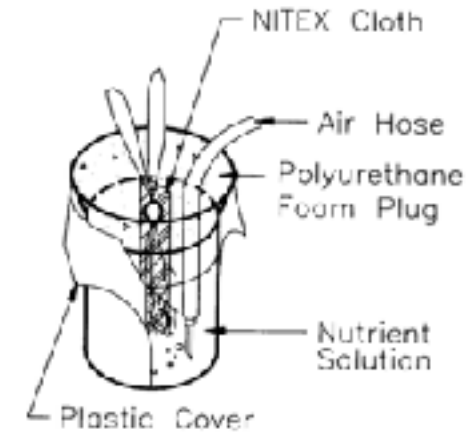


Fig. 2. Superdwarf wheat plants growing under an array of red LEDs.

All the examples in literature usually have a farmer — but wait, we're space themed, so imagine we're farmers IN SPACE — and we want to know which type of LED gives better results for growing wheat in weightlessness.

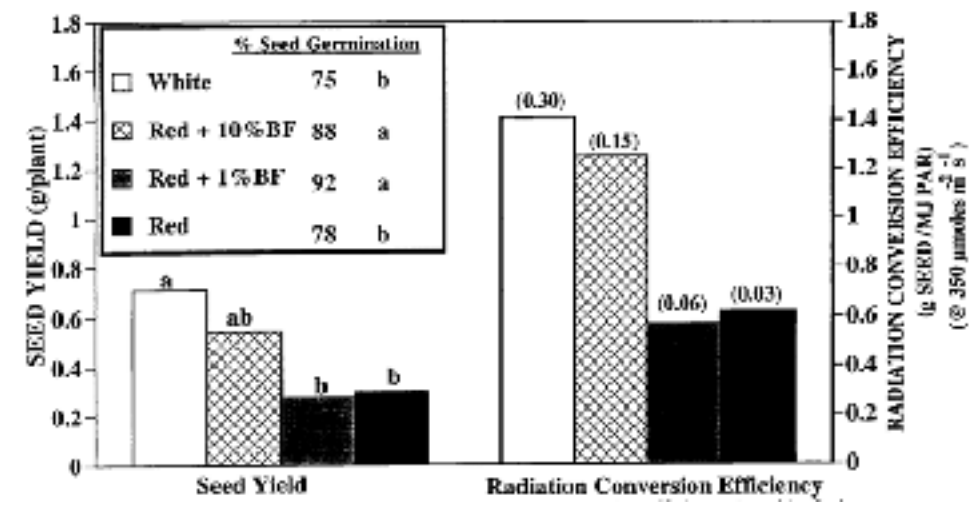
- So we split up our fields into smaller chunks and randomly assign them to either White LEDs or Red LEDs.

H_0 – “A and B are equal”



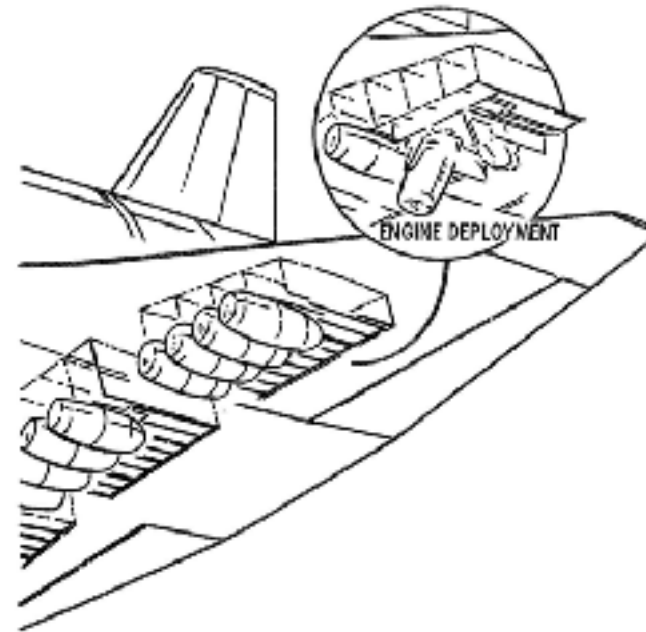
p – Likelihood that H_0 is true
(given the statistical distribution of test data)

- The Null hypothesis or H zero is the statement that we're trying to REJECT here, so we're defining that as „both LEDs have the same effect“ or “the experiment is the same as the control group”
- We then calculate the p-value of that null hypothesis from the statistical distribution of our test result data. That gives us a number between 0 and 1 that shows us how LIKELY it is that H0 is true.
- If the p-value is large, A and B are most likely equal
- If the p-value is small, that means that there is in fact a significant difference. We've DISPROVED the null hypothesis that there is no difference, and therefore we have proven there IS a difference.

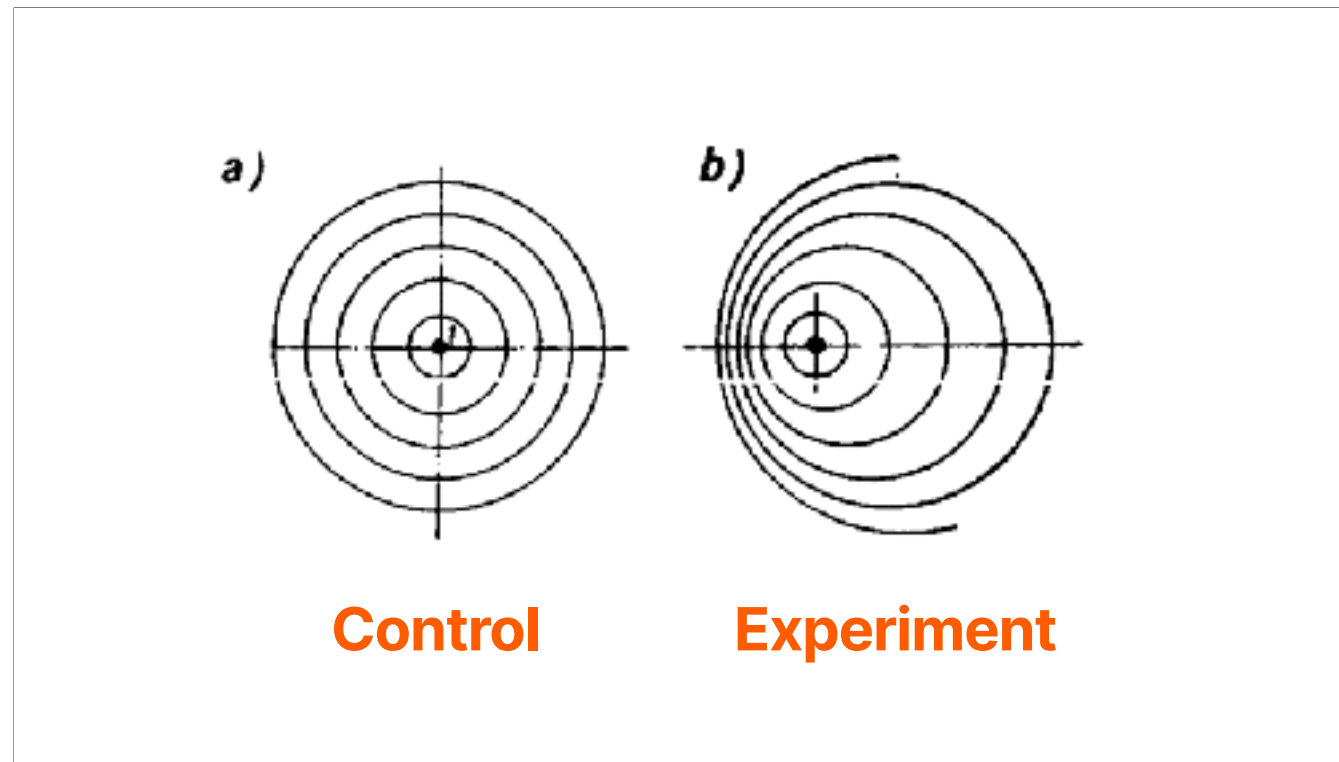


- We've just proven there is a difference and not just random fluctuation because we've rejected the null hypothesis.
- We then pick the better performing of the two LEDs, the one with the higher yields.
- We've proven that it is measurably better (and we can also see how much better it is)

A/B Testing Apps

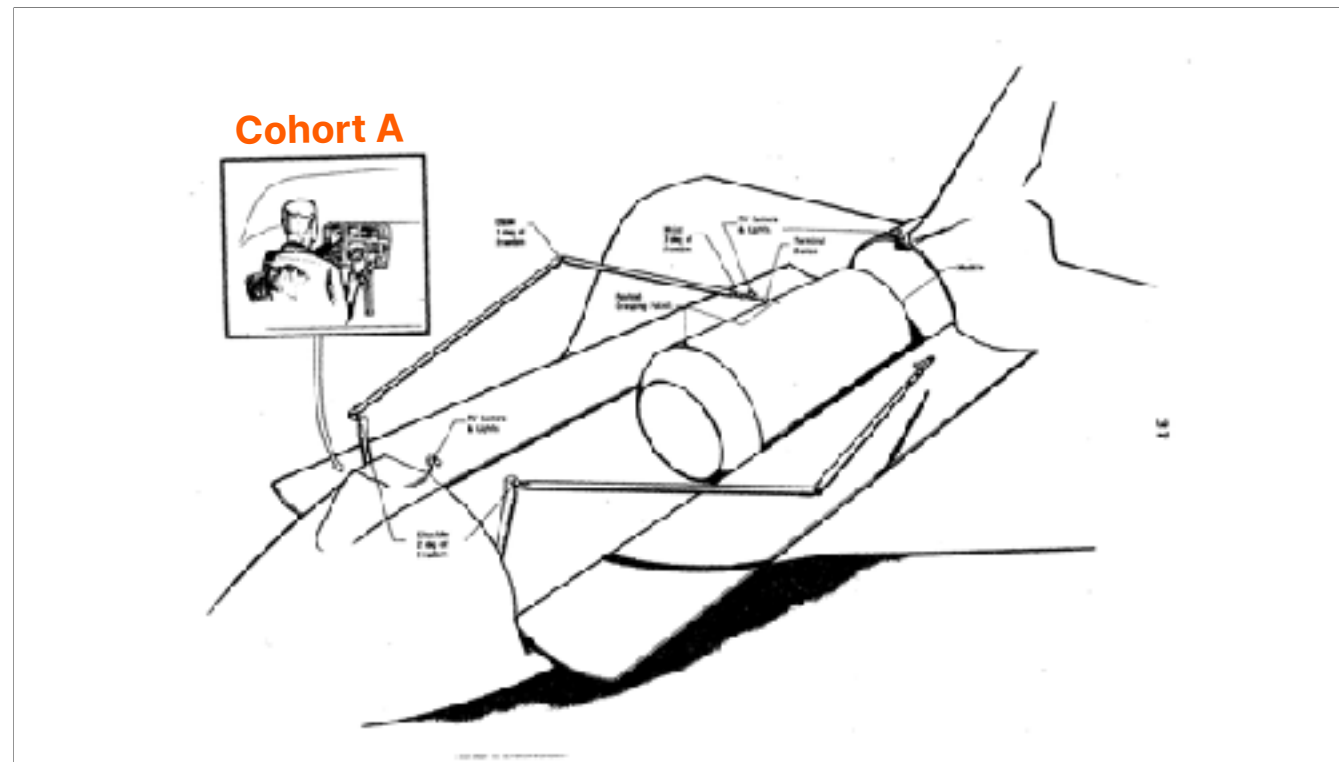


- With our APPS, we want to check whether there's a difference between two versions of the same screen, or two behaviors or two designs.
- We distribute our users into different buckets, show them the feature corresponding to the bucket, and measure the success somehow.

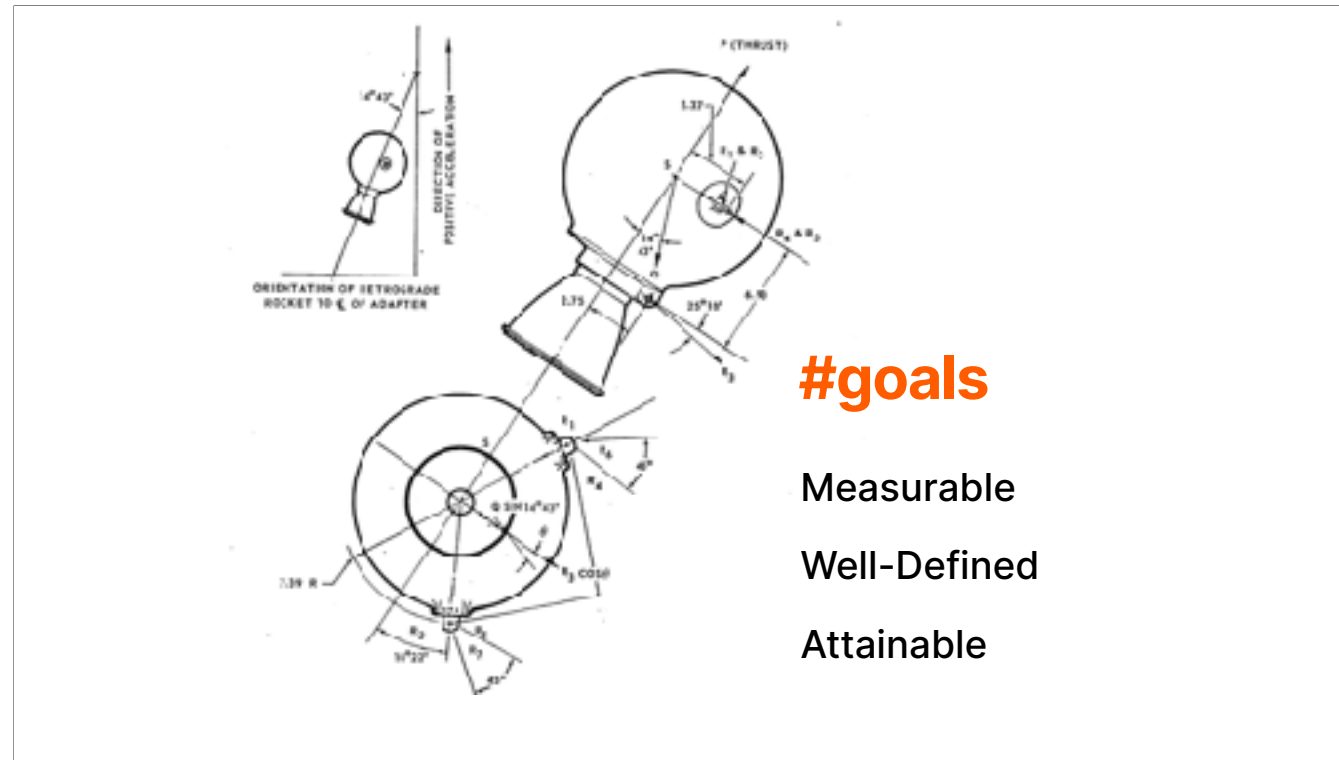


Buckets or Groups of users are called Cohorts

- A and B are our cohorts, the sets of users who are taking part in either alternative
- Other names are control group and experiment
- usually, the control group stays the same, and experiment is the change that we're trying out



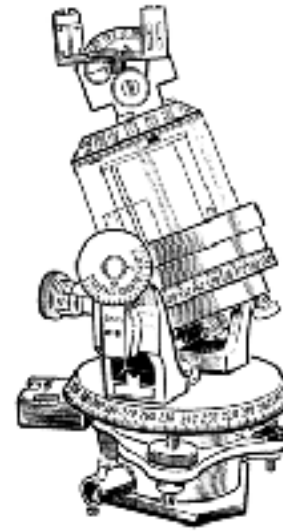
- How do we decide who's in group A and who's in group B
- The easiest way is to decide randomly for each user whether they are in group a or group b,
- then store that choice somewhere so it stays consistent.
- We could also sync it for users with multiple devices.
- We could also let an external feature flag service decide whether a user gets the new version or not
- We DONT need to put the same amount of users into each cohort. Maybe we just want to try out the feature with a few percent of our users. As long as a statistically significant number of users is taking part, that's fine.



- we need to Find our goals and hypotheses
 - good goals are measurable and well defined:
 - "Increased IAP"
 - "Better success of a feature"
 - "Improved onboarding"
 - "The app should be nicer" is not measurable.

- Hypotheses need a success criterion, a measurable telemetry signal that tells us a user has succeeded in the experiment.

Measure at the end



We then run tests, and we use a measuring tool to collect the test data. For us, this is usually a server or service that collects usage data. Ideally the usage data is anonymous telemetry so that we're not creeping on people, and don't have to ask for opt-in consent in most cases

Code

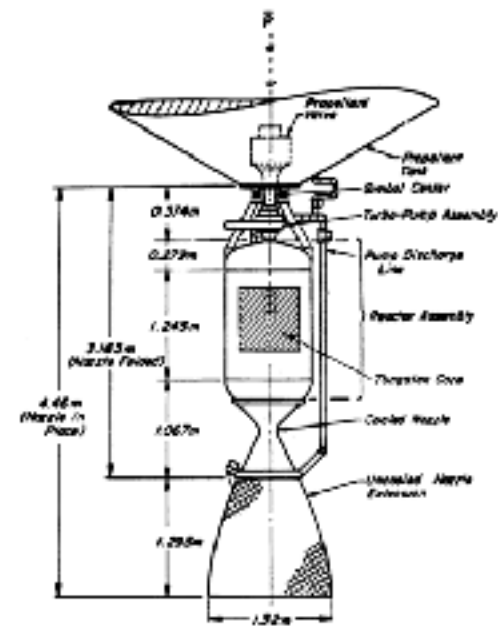


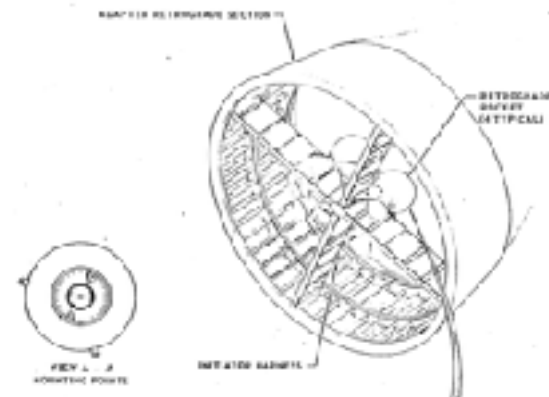
Fig. 2. Schematic diagram of the Small Nuclear Rocket Engine designed during the NRO program. The nuclear reactor core was constructed with a variable configuration of the metal-hydrogen used to convert the nuclear energy into heat.

Lets finally look at some real world examples. The following code is all in Swift and SwiftUI, but you can do this just as well on your platform of choice.

```

@AppStorage("welcomeScreenCohort") var welcomeScreenCohort: String = {
    if Bool.random() {
        return "B"
    } else {
        return "A"
    }
}

```



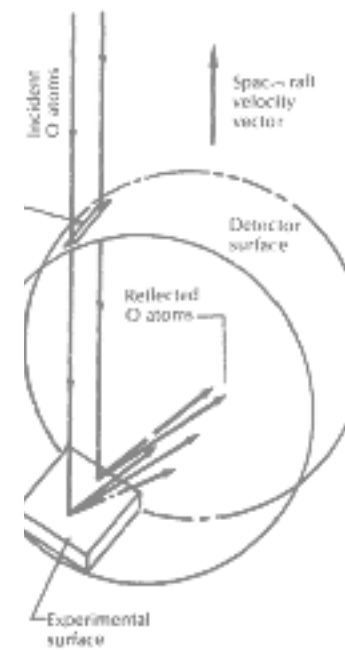
Here we decide randomly which cohort a user belongs to and store that so it'll be the same from then on.

```

// If we're in the experiment cohort,
// show an alternative image
var welcomeImagePath: String {
    if welcomeScreenCohort = "B" {
        return "newWelcomeImage"
    }

    return "appIcon"
}

```



Based on this cohort, we show different user interfaces. Here we show a different banner image on the welcome screen.

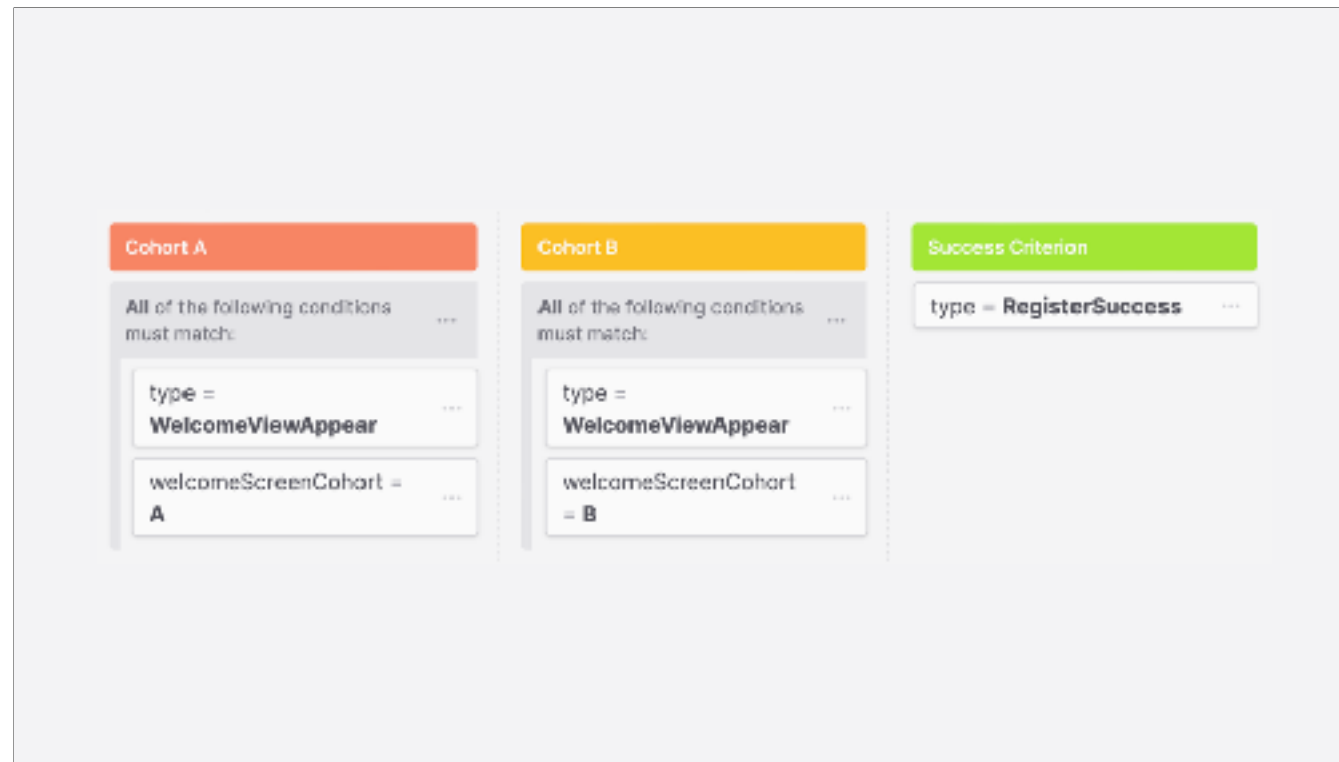
```
WelcomeView {  
    // ...  
}  
.onAppear {  
    TelemetryManager.send("WelcomeViewAppear", with:  
        ["welcomeScreenCohort": welcomeScreenCohort])  
}
```



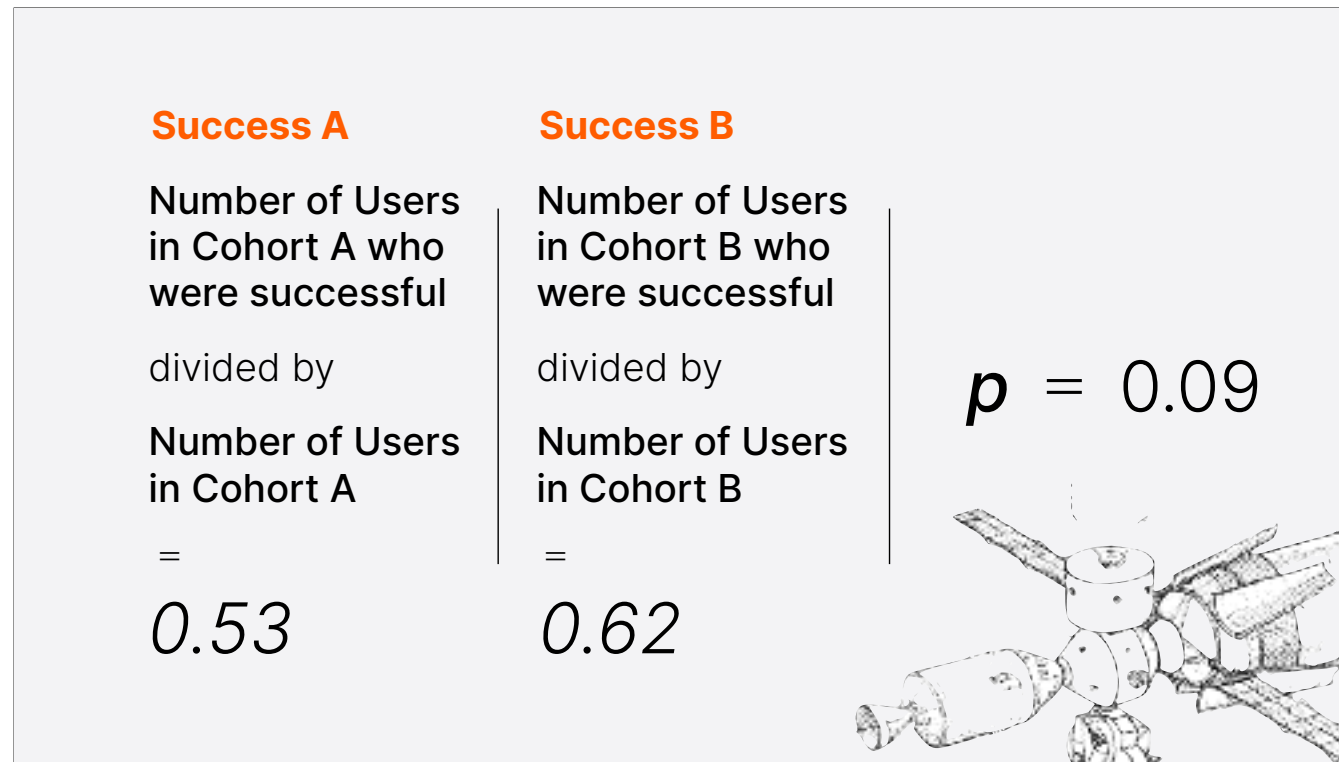
Whenever a user enters the experiment, we send a telemetry signal that includes the cohort



And when a user SUCCEEDS in the experiment – here our success criterion is someone clicking the REGISTER button – we send another telemetry signal



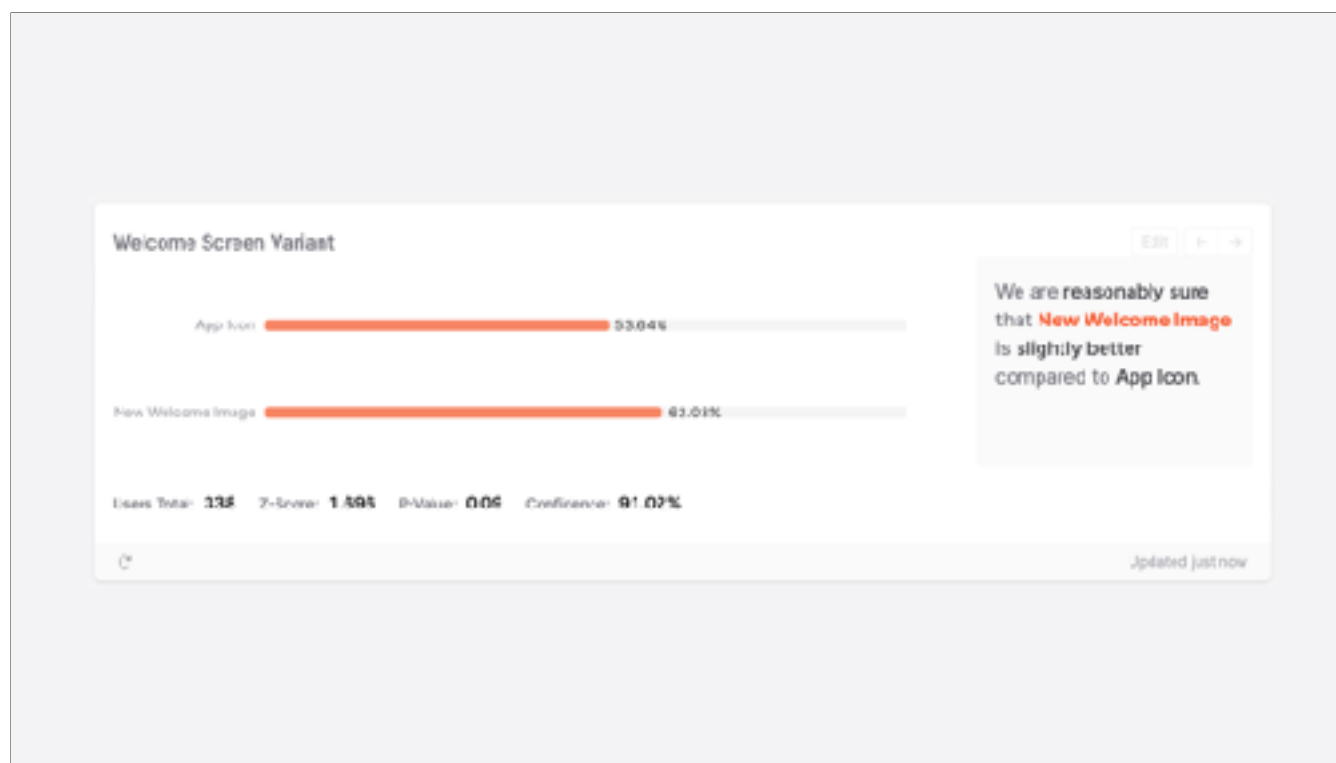
So here's how I'm defining my cohorts and success criteria in TelemetryDeck. I'm defining filters that describe the cohorts, and the success criteria.



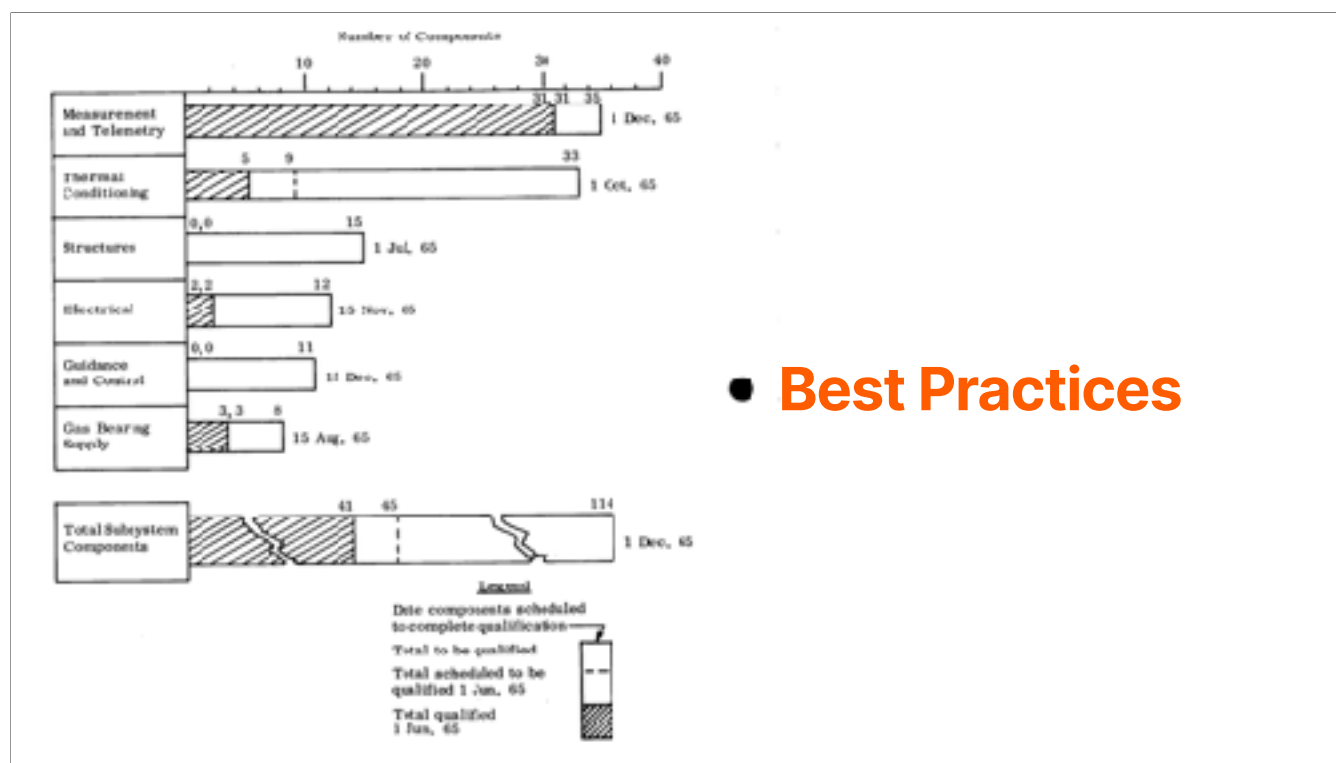
For success A, we're determining the number of users who were in cohort A and ALSO sent a success criteria signal. We divide that number by just the number of users in cohort A. We do the same with B, and see that B is performing a bit better than A.

But how can we be sure it's just random chance? The p-value! By calculating the p-value of our Null Hypothesis – remember, the null hypothesis is that both cohorts perform the same – we get a p-value of 0.09 in this example, which rejects our null hypothesis, proving that the two are different.

Now, 0.09 is not super strong, 0.05 would be better, 0.01 would be way better, but for us it's enough to decide to go with B for now. Scientists usually demand a p-value of no larger than 0.05


















Double checking with TelemetryDeck, and it comes to the same result, phew!



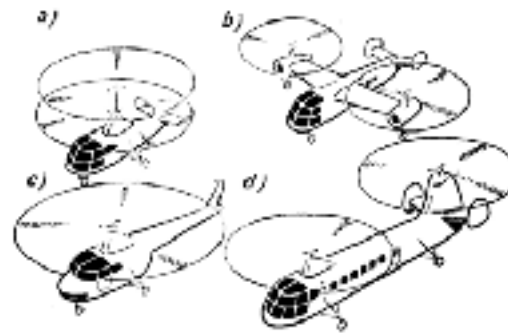
● Best Practices

Lets go over some best practices to remember when A/B testing

Test the
important
parts

LEGEND	
TCA	THRUST CHAMBER ASSEMBLY
	CONNECTED
	NOT CONNECTED
$T_1, T_2 \& T_3$	TEMPERATURE (TRANSMITTED TO INSTRUMENT PANEL INDICATOR)
$P_1 \& P_2$	PRESSURE (TRANSMITTED TO INSTRUMENT PANEL INDICATOR)
	MANUAL VALVE
	CARTRIDGE-ACTUATED VALVE (NORMALLY CLOSED)
	CARTRIDGE-ACTUATED VALVE (NORMALLY OPEN)
	PRESSURE TRANSDUCER
	RELIEF VALVE
	SOLENOID VALVE (NORMALLY CLOSED)
	PRESSURE SWITCH
	BURST DIAPHRAGM
	CHECK VALVE
	FILTER
	GROUND TEST CONNECTION
	MOTOR OPERATED VALVE
	FLOW LIMITING ORIFICE

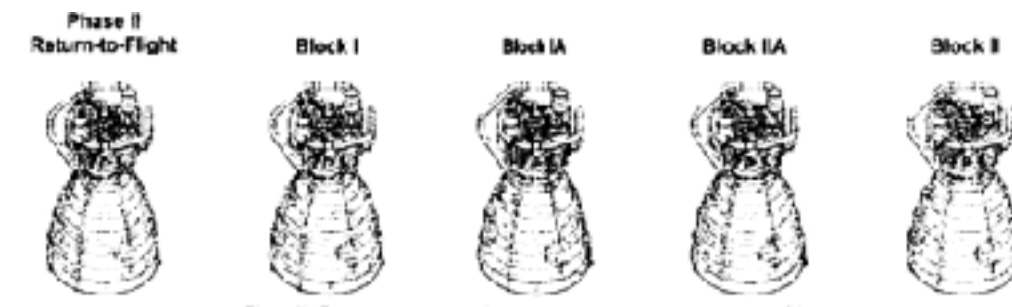
Test the important parts. Figure out which parts of your app are most critical on the golden path and test those first.



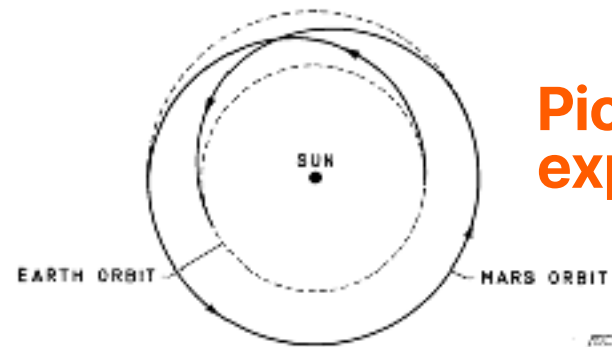
Pick a large sample size

- choose the right sample size: at least 100 users, more is better

Randomize Users & Avoid Biases



It's easy to pick of biases from your user base, from their country of origin, from the OS version, from the size of their device, etc. Try to identify and eliminate as many external variables as possible and be as random as possible. Test things in isolation if possible.



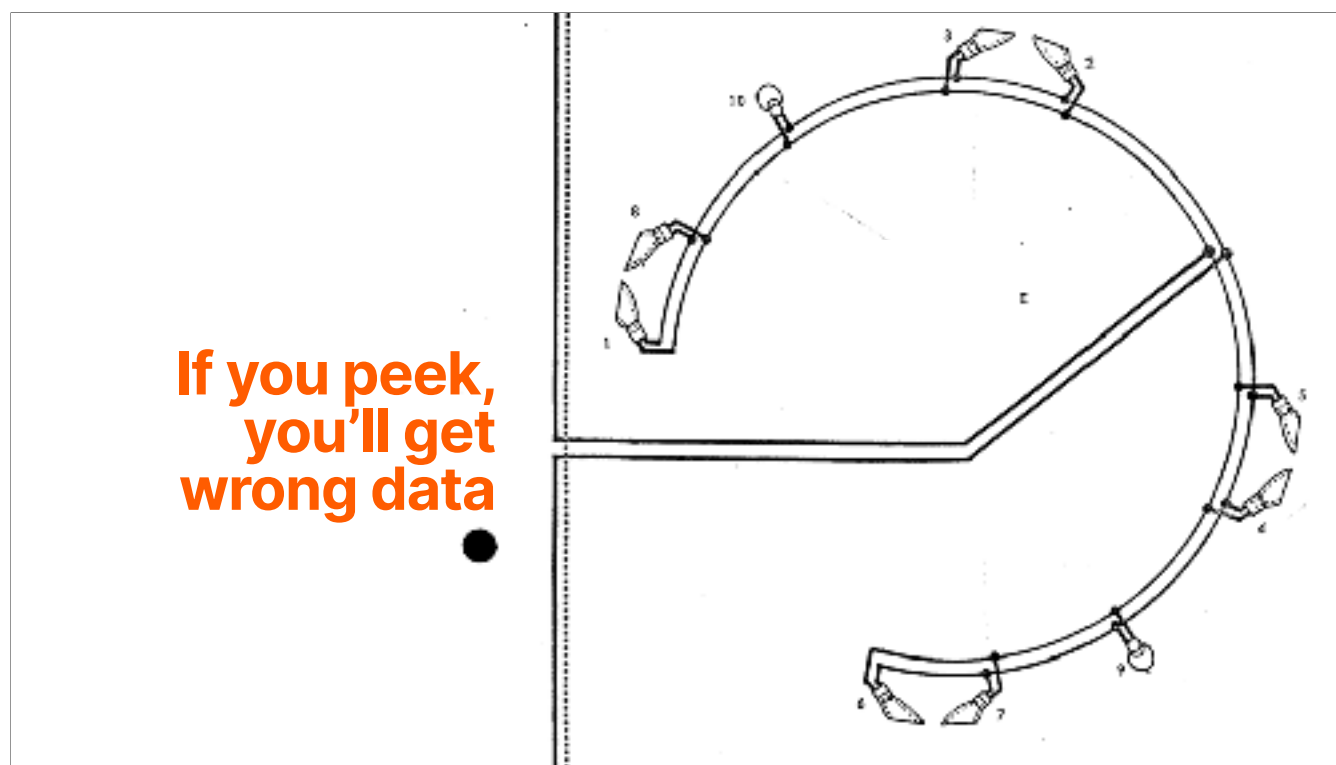
Pick a sufficient
experiment duration

TIME IN DAYS FOR MARS JOURNEY

SYSTEM	ESCAPE FROM EARTH ORBIT	COAST	DESCENT TO MARS ORBIT	WAIT	TOTAL
LOW THRUST IF $W_0 = 10^{-4}$	127	268	85	245	1205
IMPULSE ROCKET	—	268	—	415	951

Figure 2. - Flight path to Mars.

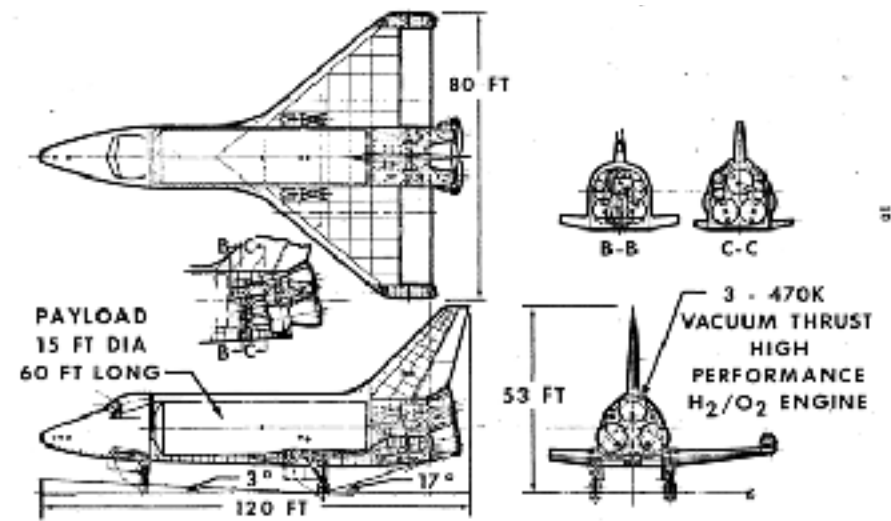
- Don't just run a test for a day or two. Run them for 10 days at least! 30 days is better. 60 days is way better.



Farmer example: only checked data at one pre-specified point in time, the harvest

With modern platforms, we can peek at the problem every day, and check if the confidence values are high enough.

However, that can fluctuate. If you run a test for 30 days, and peek every day, there's a over 60% chance that you get a false positive confidence value at least once.



Lets make our apps awesome

And lastly, let's remember why we are doing this: because we want our apps to be awesome! Thank you!

THIS PAGE INTENTIONALLY LEFT BLANK.

TelemetryDeck

telemetrydeck.com

Social

@daniel@social.telemetrydeck.com

[@breakthesystem.bsky.social](https://bsky.social)

Sources, slides and text

github.com/TelemetryDeck/ab-testing-talk

Questions, thank you's, and hit me up during the conference for TelemetryDeck pins