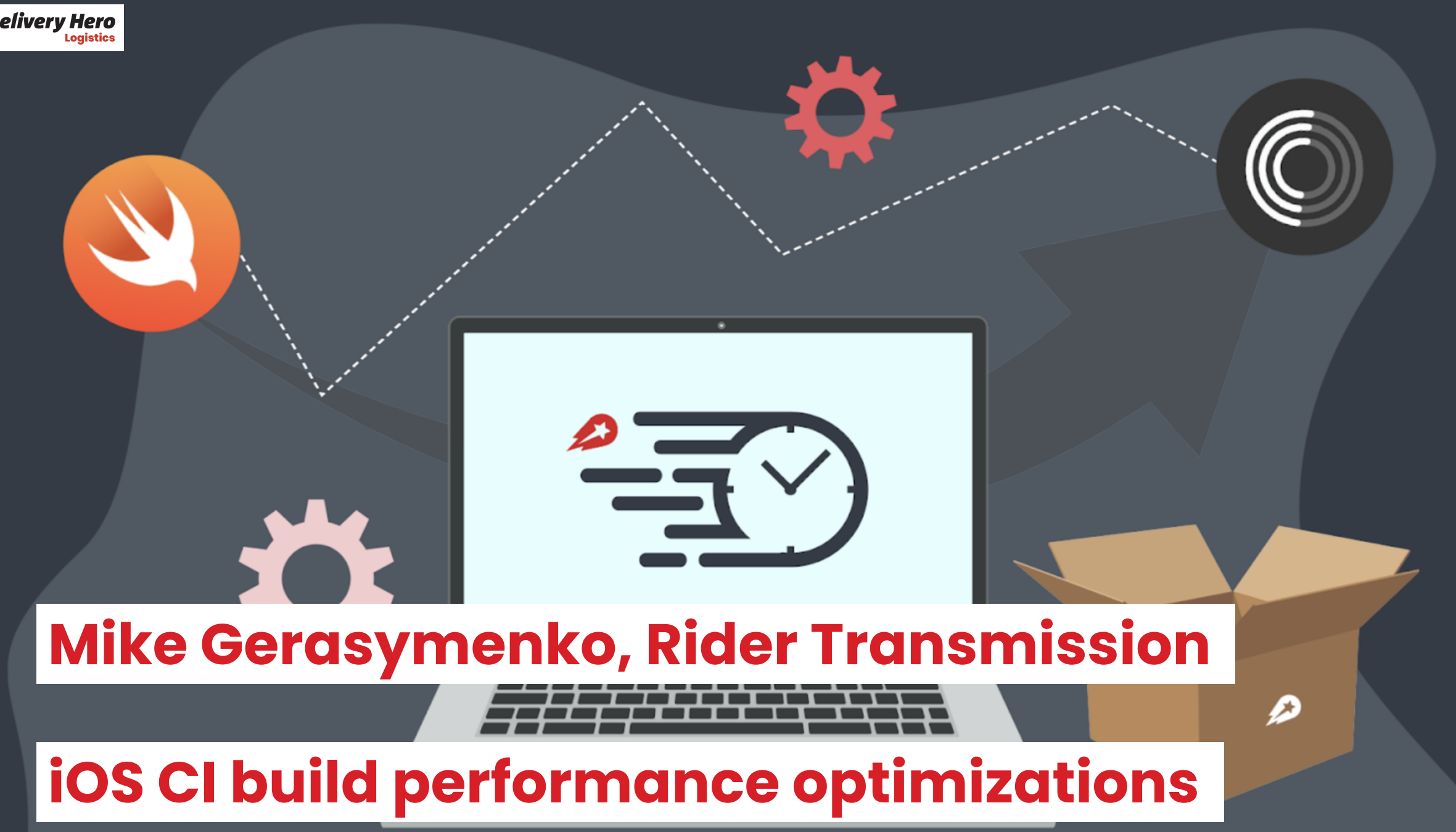
Delivery Hero
Logistics



Mike Gerasymenko, Rider Transmission

iOS CI build performance optimizations

Delivery Hero
Logistics

Summary

A series of build optimizations were implemented for the iOS Rider application.

Those optimizations improved CI build time and build time for the engineers.

CI build time is especially significant, going from ca. 30 minutes per build to 5 on average.

Delivery Hero
Logistics

What usually contributes to the CI build time?

- Step 1: Test execution
- Step 2: CI Machine type
- Step 3: Compilation

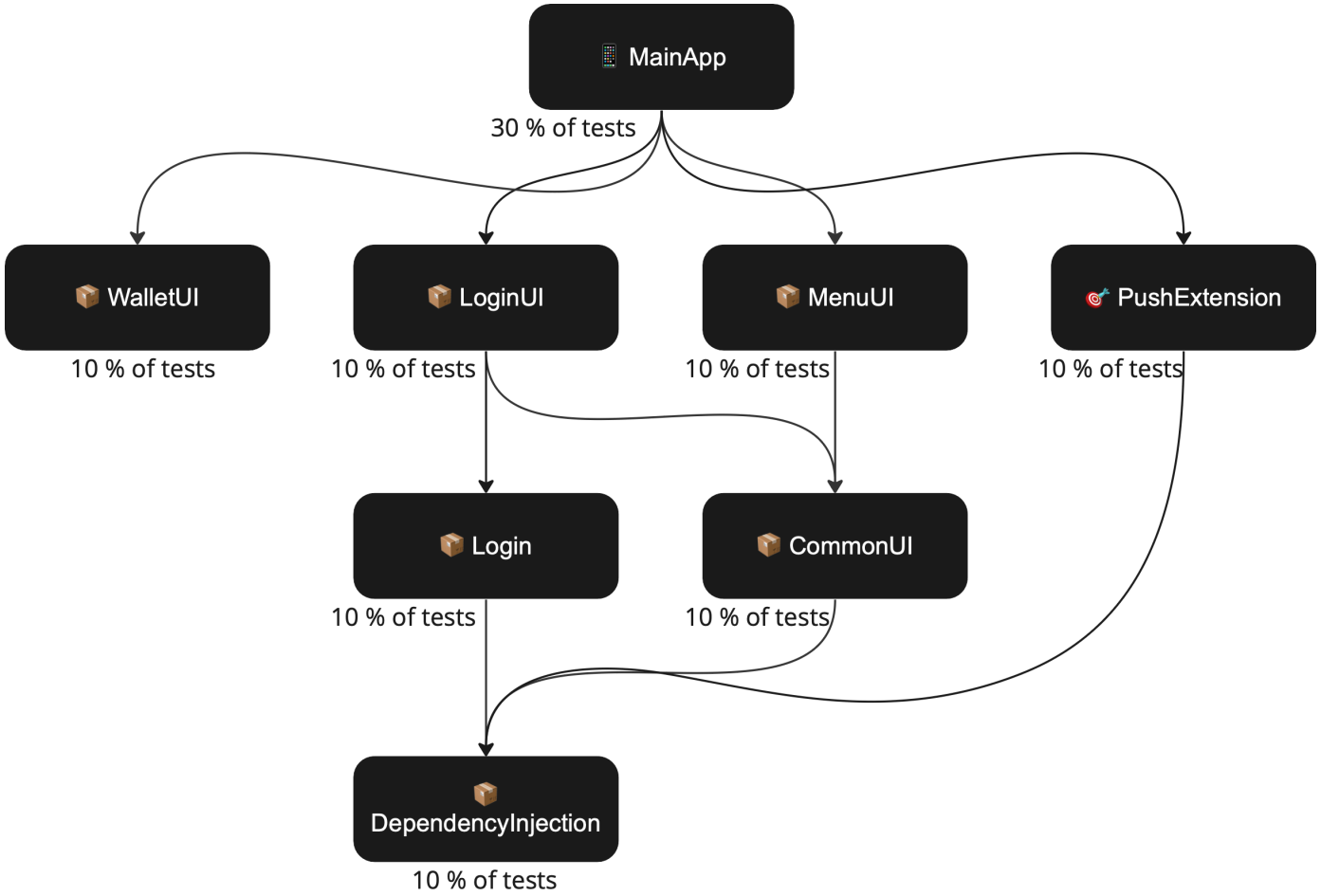
Delivery Hero
Logistics

Step 1: Test execution

Delivery Hero
Logistics

Modules

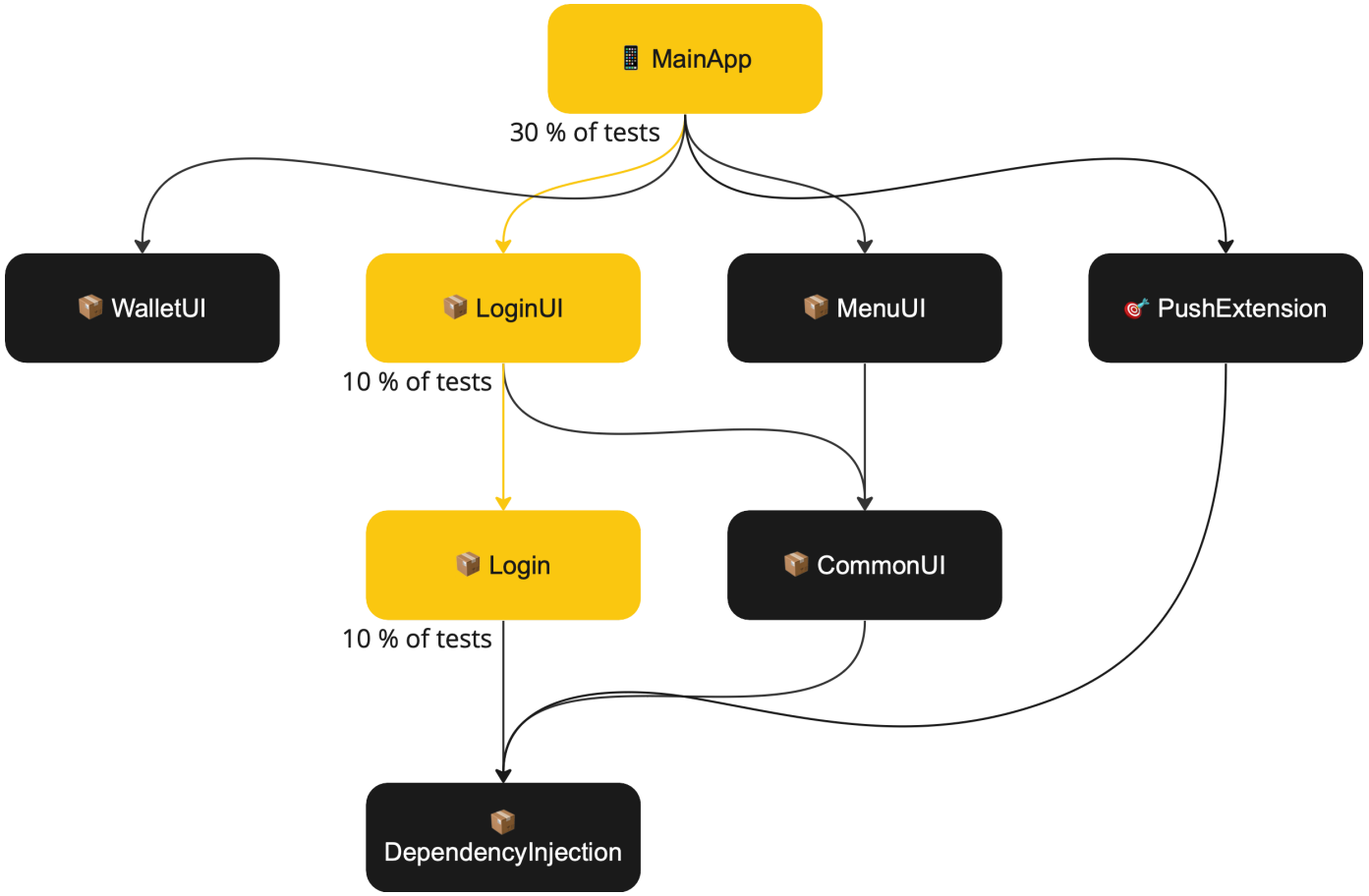
The Rider application is modularized using Swift Package Manager (around 40 modules in total)



Delivery Hero
Logistics

Change

If the 📦`Login` module is changed, it would only affect the 📦 `LoginUI` and the 📱 `MainApp`.

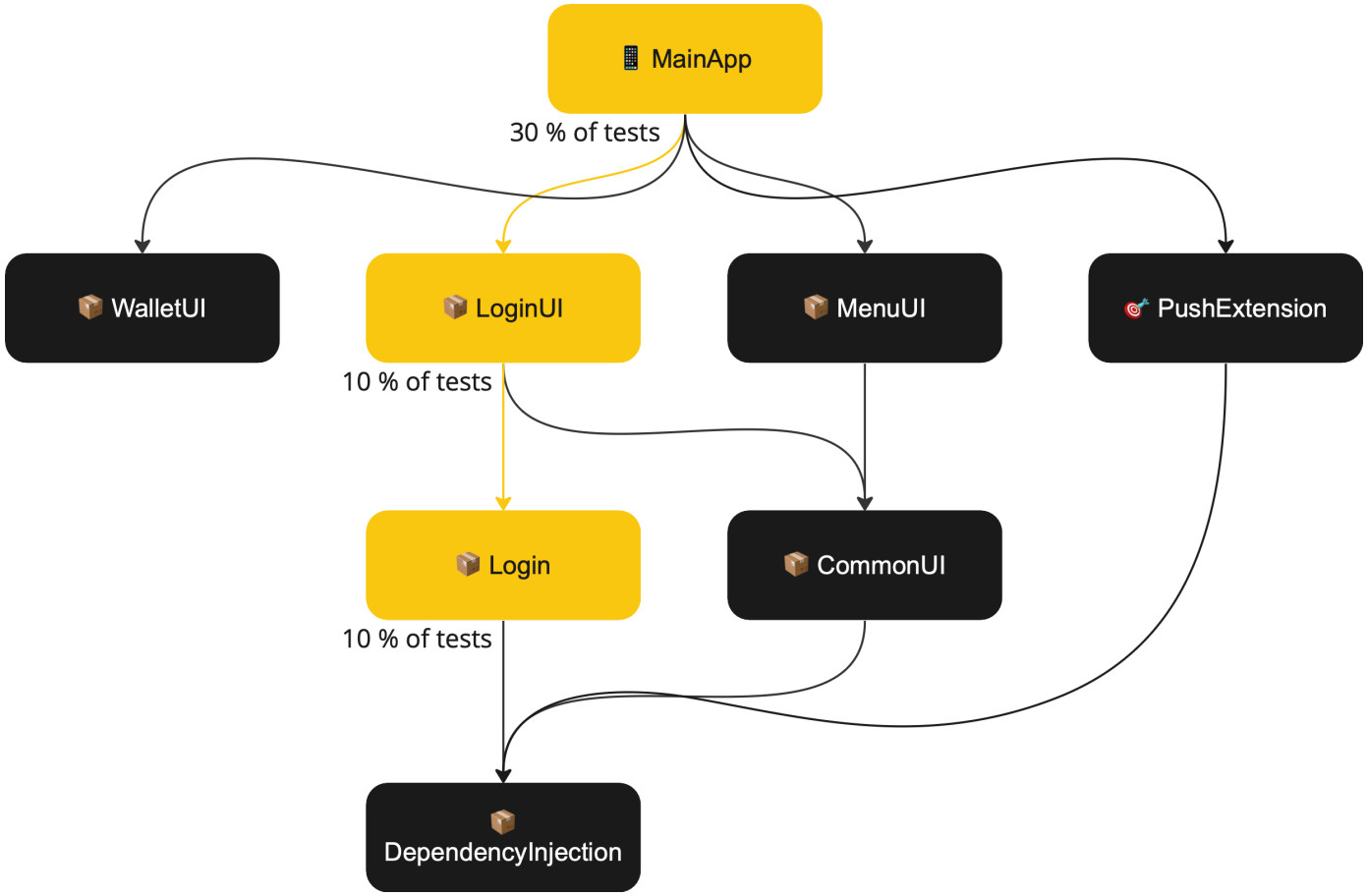


Does it make sense to test all the modules if we know only the 📦`Login` module has been changed?

Absolutely no. If the module was not changed, and not using the changed modules we can skip testing it.

Delivery Hero
Logistics

We can only run 50% of the tests and get the same results.



Delivery Hero
Logistics

But how can we know?

1. Detecting what is changed

Well, Git allows us to find what files were touched in the changeset.

```
Root
├── Dependencies
│   └── Login
│       ├── ❗LoginAssembly.swift
│       └── ...
├── MyProject.xcodeproj
└── Sources
```

2. Build the dependency graph

Going from the project to its dependencies, to its dependencies, to dependencies of the dependencies, ...

Can be achieved with *xcodeproj* gem or a similar library.

Dependencies between packages can be parsed with `swift package dump-package`.

BTW, This is the moment the Leetcode graph exercises would pay off

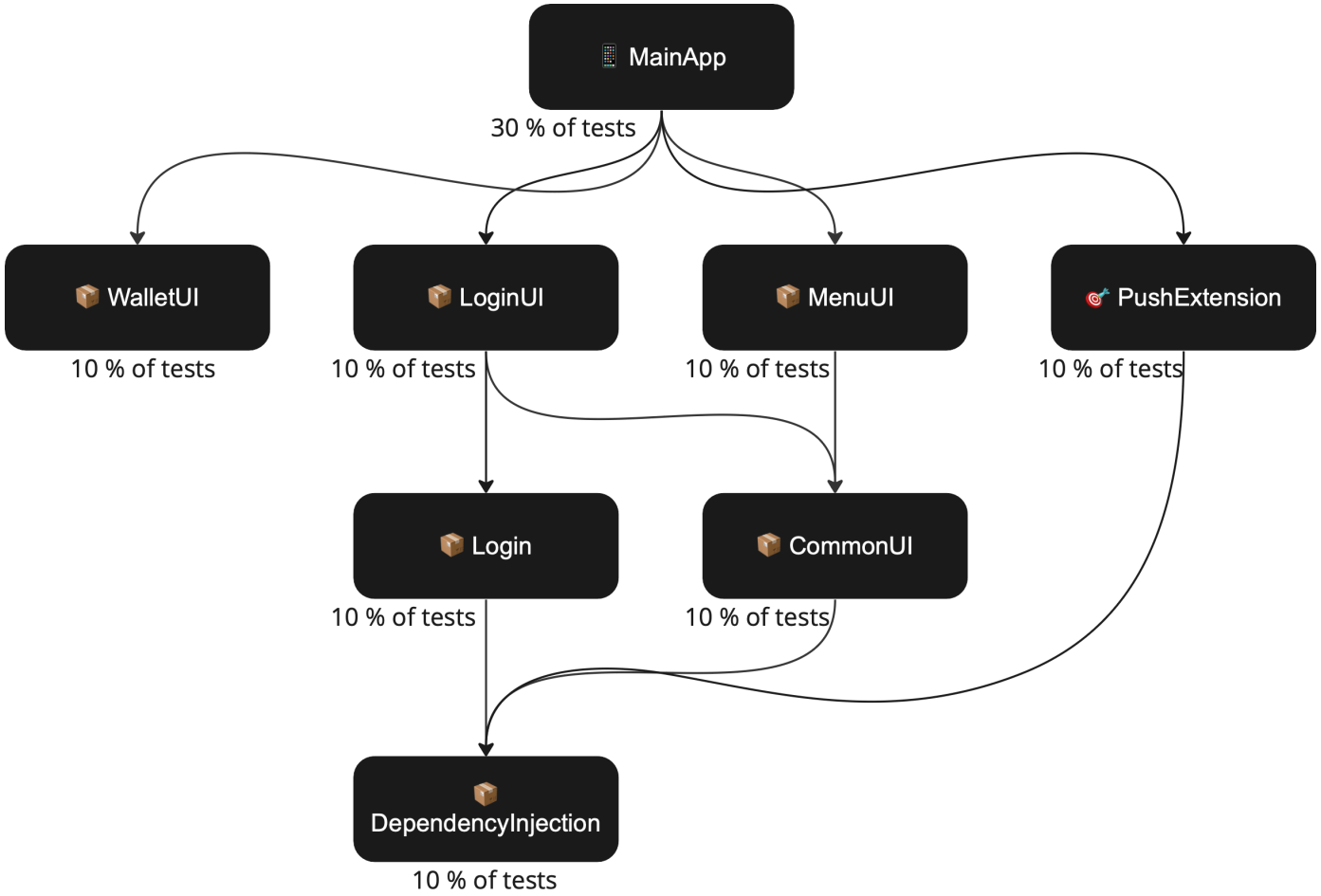
2.5. Save the list of files for each dependency

This is important, so we'll know which files affect which targets.

Delivery Hero
Logistics

3. Traverse the graph

Go from every changed dependency all the way up, and save a set of dependencies you've touched.

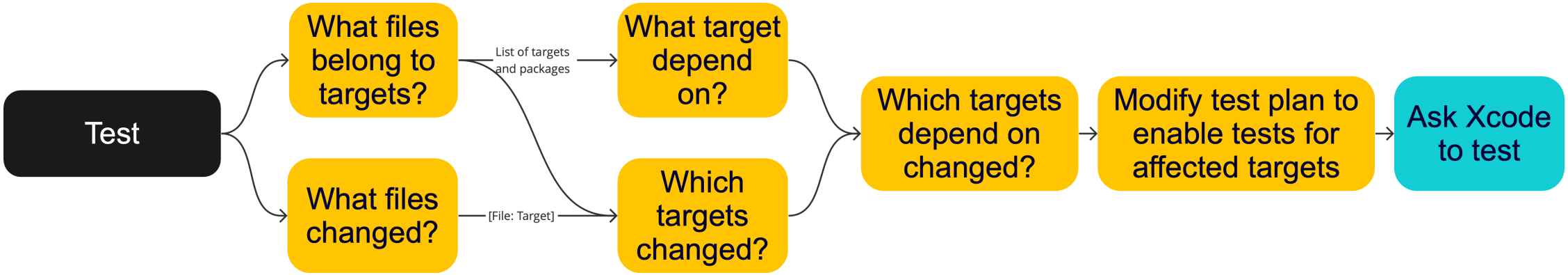


4. Disable tests that can be skipped in the scheme / test plan

This is actually the hardest part. Dealing with obscure Xcode formats. But if we get that far, we will not be scared by 10-year-old XMLs.

Delivery Hero
Logistics

Overview



- Pull Request Link 🔗

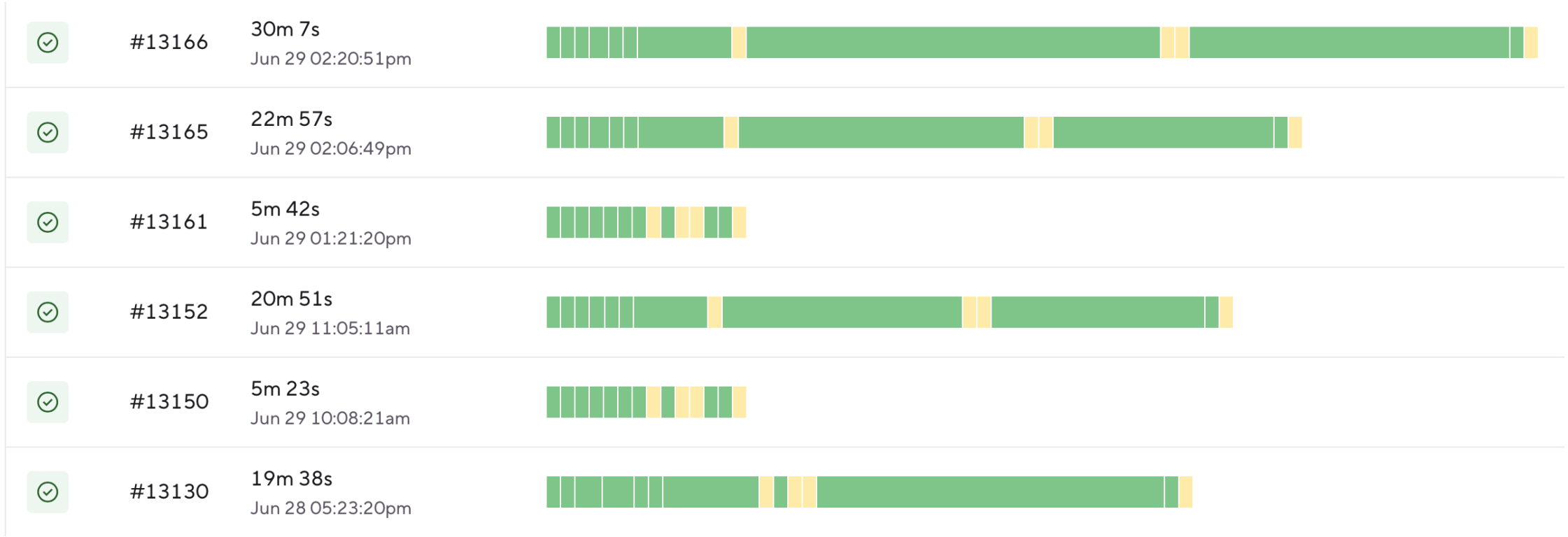
Delivery Hero Logistics

Step 1: Results

Execution type / minutes	Unit tests duration	Snapshot tests duration	Total duration	Improvement per run
Old	15 min	10 min	ca. 32-35 min	n/a
New (only Roadrunner changed)	10 min	10 min	ca. 26 min	5 min
New (no files in targets	20 sec	20 sec	ca. 7 min	25 min

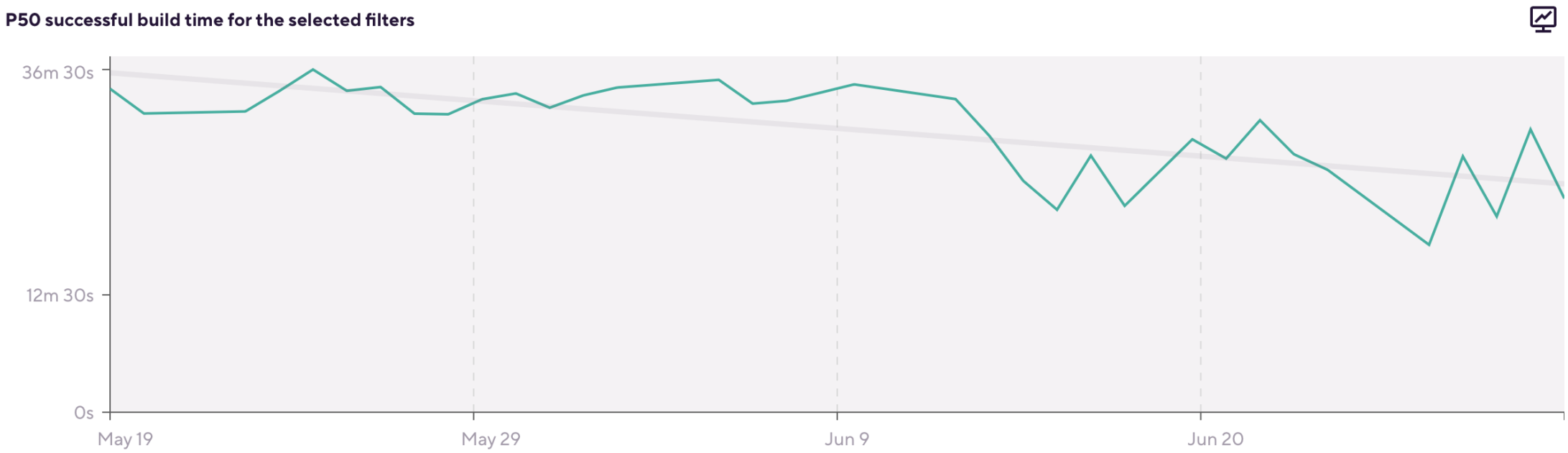
Delivery Hero
Logistics

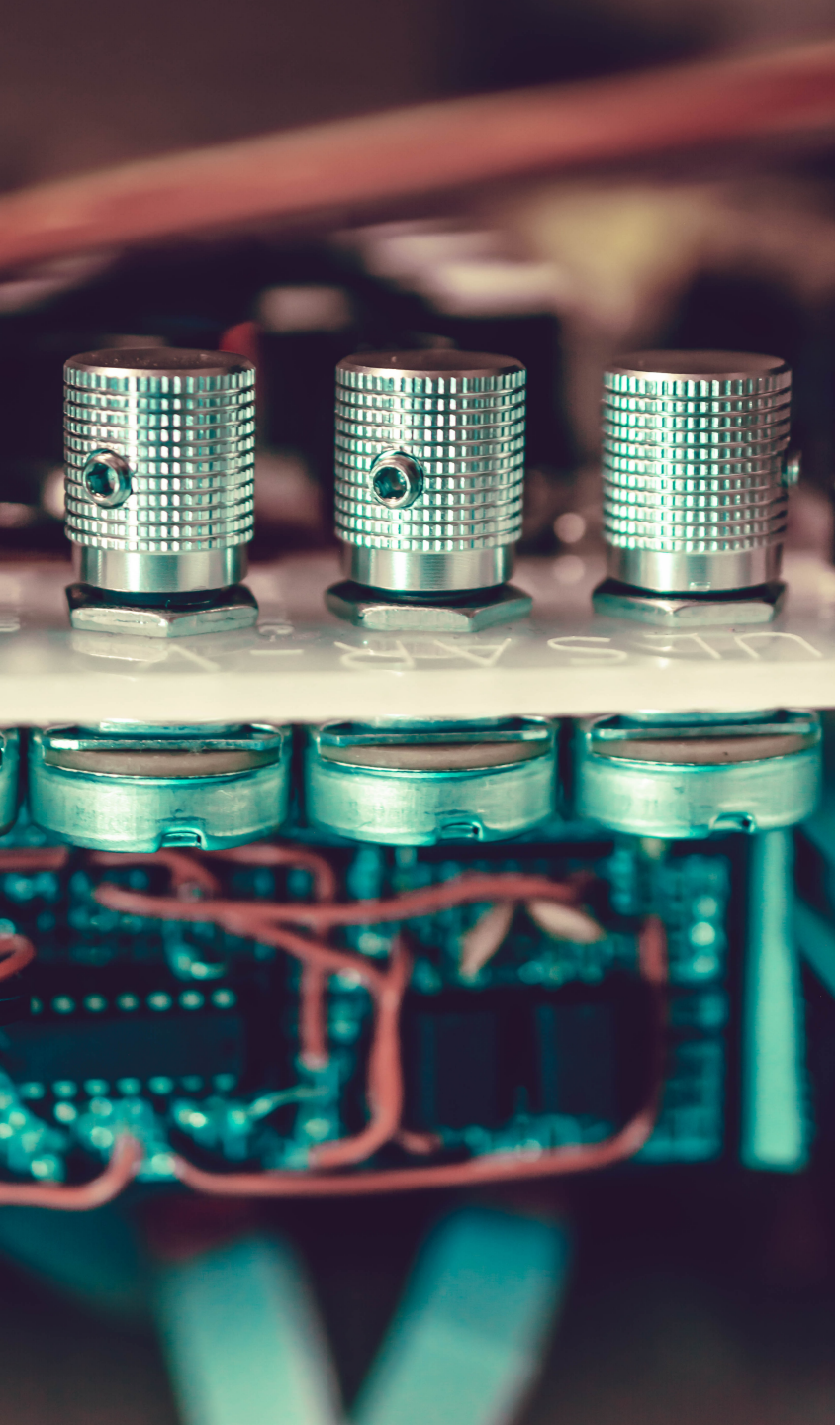
Step 1: Build examples



Delivery Hero
Logistics

Step 1: Big picture





Delivery Hero
Logistics

Step 1: Summary

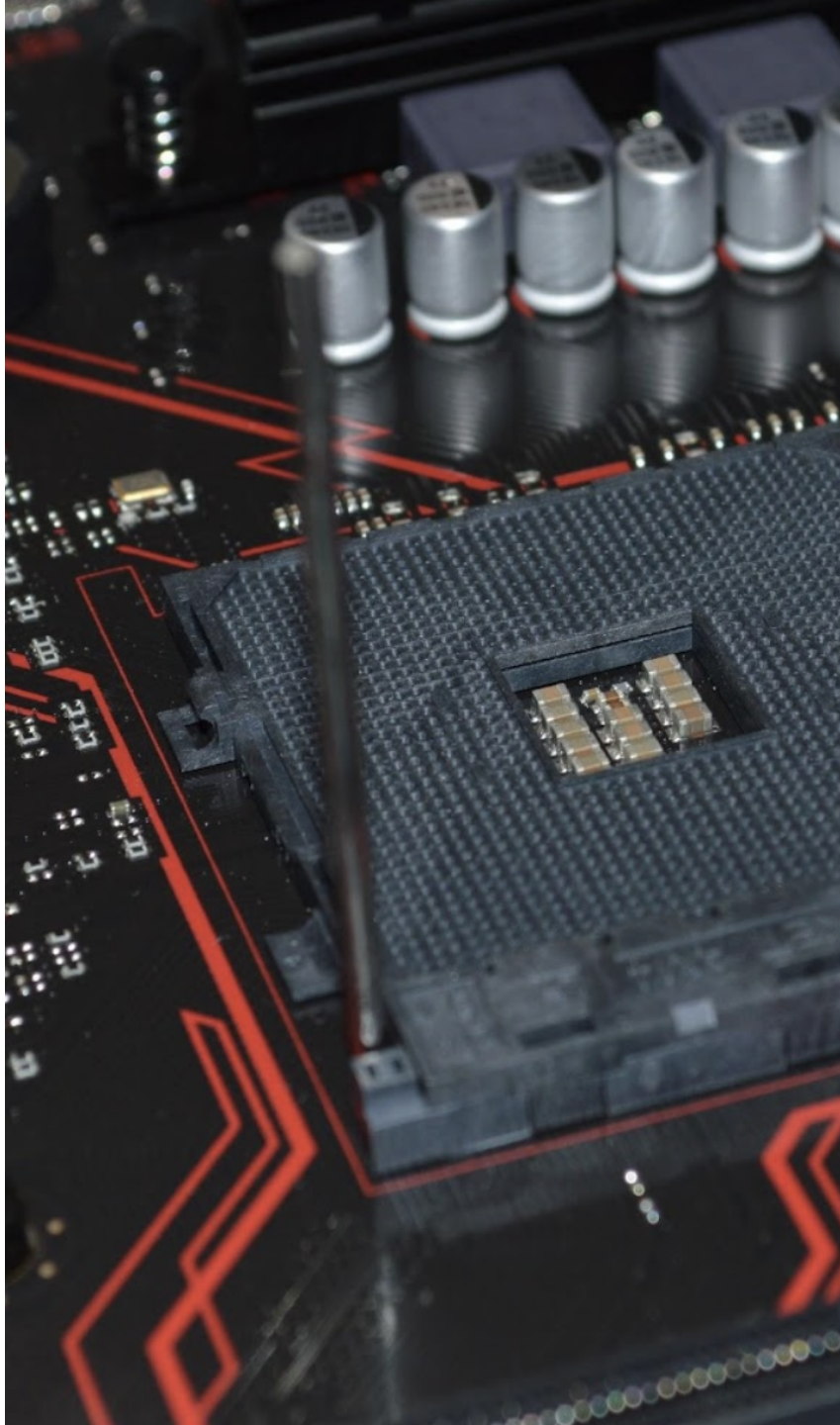
We went from consuming 30 minutes per build on average to ca. 20 minutes per build (conservative estimation).

This optimization allows the company to save ca. 30% of CI build credits for Pull Request builds for future years.

Delivery Hero
Logistics

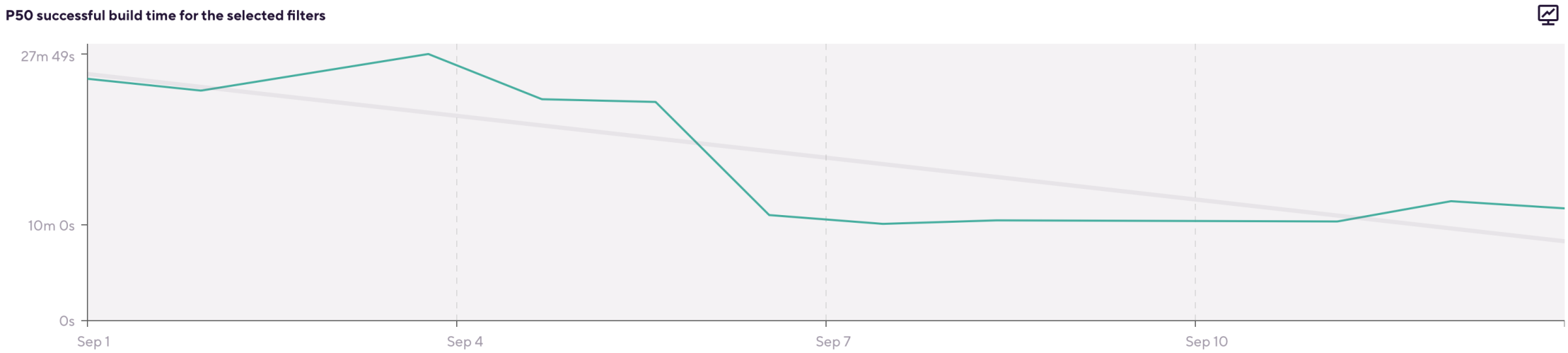
Step 2: CI Machine Type

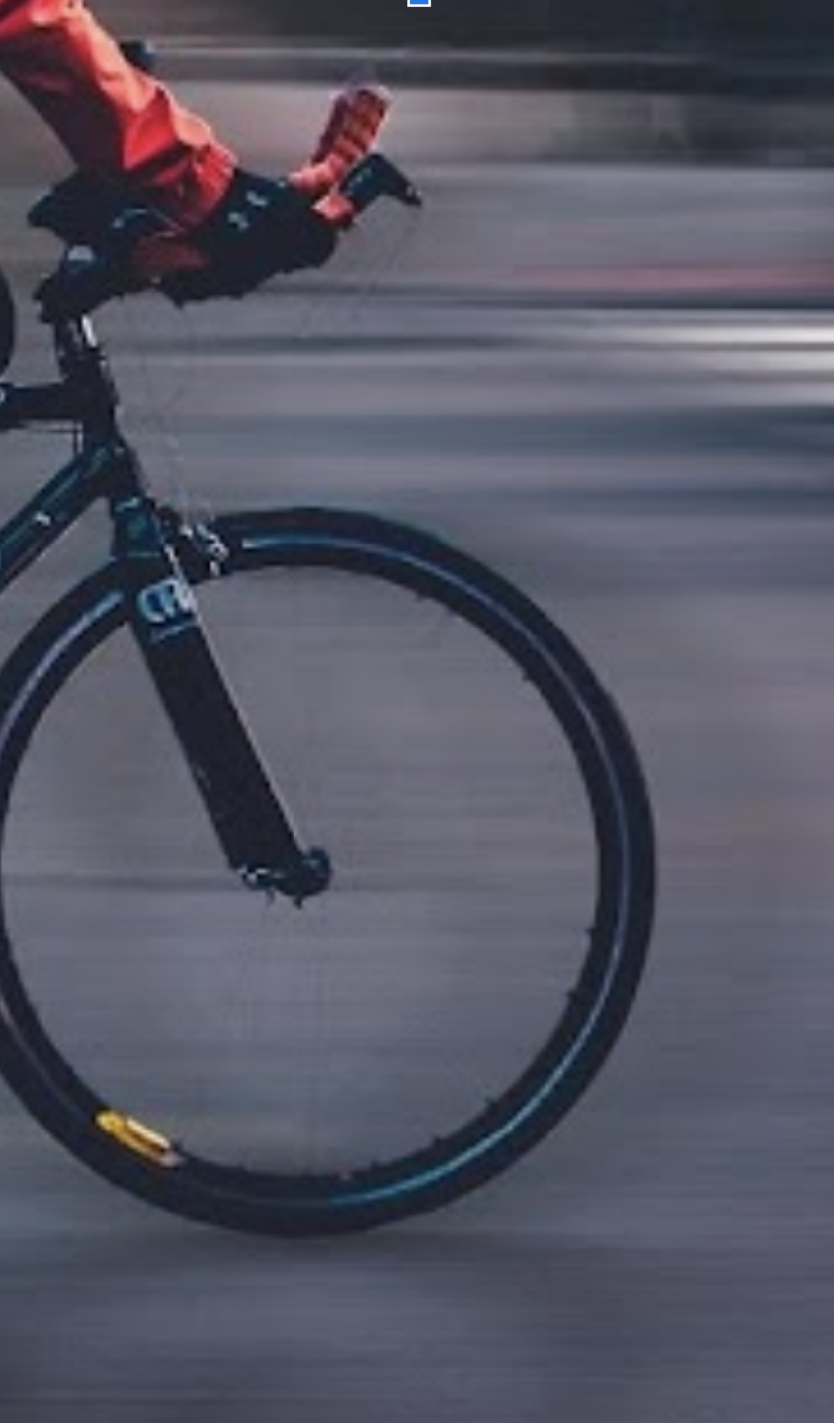
In September 2022, we received an opportunity to test Bitrise Apple Silicon workers.



Delivery Hero
Logistics

They were, in average, 50% faster in compiling and test our project:





Delivery Hero
Logistics

Effectiveness

However, they were also more expensive. We decided to find out if they are a good bang for the buck.

Methodology

We were able to switch the CI machine type. For the second and third weeks of October 2022, we switched CI machines to Intel cheaper workers and collected daily statistics on credit consumption per build.

We have different build types and bi-weekly release cadence, which should give us a representative result.

Analysis Sheet Link 🔗

Pull Request Link 🔗

Delivery Hero
Logistics

	Credits	Mins	Builds	Credit Per Build	Machine type
Week 4 in September	**16,546.00**	**2,757.67**	**366.00**	45.21	M1 Elite XL 6 credits/min 8 CPU @3.2GHz 12 GB RAM Running on Apple Silicon, M1 Mac Minis
Sep 26, 2022	4,732.00	788.67	89.00	53.17	
Sep 27, 2022	3,438.00	573.00	81.00	42.44	
Sep 28, 2022	2,434.00	405.67	57.00	42.70	
Sep 29, 2022	3,592.00	598.67	83.00	43.28	
Sep 30, 2022	1,972.00	328.67	49.00	40.24	
Week 1 in October	**19,405.33**	**3,234.22**	**422.00**	45.98	
Oct 3, 2022	3,800.67	633.44	83.00	45.79	
Oct 4, 2022	5,278.00	879.67	108.00	48.87	
Oct 5, 2022	3,286.00	547.67	73.00	45.01	
Oct 6, 2022	2,838.00	473.00	68.00	41.74	
Oct 7, 2022	3,800.67	633.44	83.00	45.79	
Week 2 in October 22	**29,060.00**	**4,843.33**	**575.00**	50.54	
Oct 10, 2022	5826	1,456.50	98	59.45	Elite 8 vCPU @3.2 GH 35 GB 4/min g2.8core
Oct 11, 2022	4780	1,195.00	94	50.85	
Oct 12, 2022	5888	1,472.00	94	62.64	
Oct 13, 2022	5874	1,468.50	135	43.51	
Oct 14, 2022	6204	1,551.00	145	42.79	
		% credit/build Intel consumes more		14.10%	

Delivery Hero
Logistics

Step 2: Summary

CI credits use: With Bitrise Apple Silicon "Elite" machines, we are making builds 2x faster and saving ca. 14% Bitrise credits on all build types.

Engineer focus: PR check pipeline time went from 20 minutes to ca. 11 minutes.

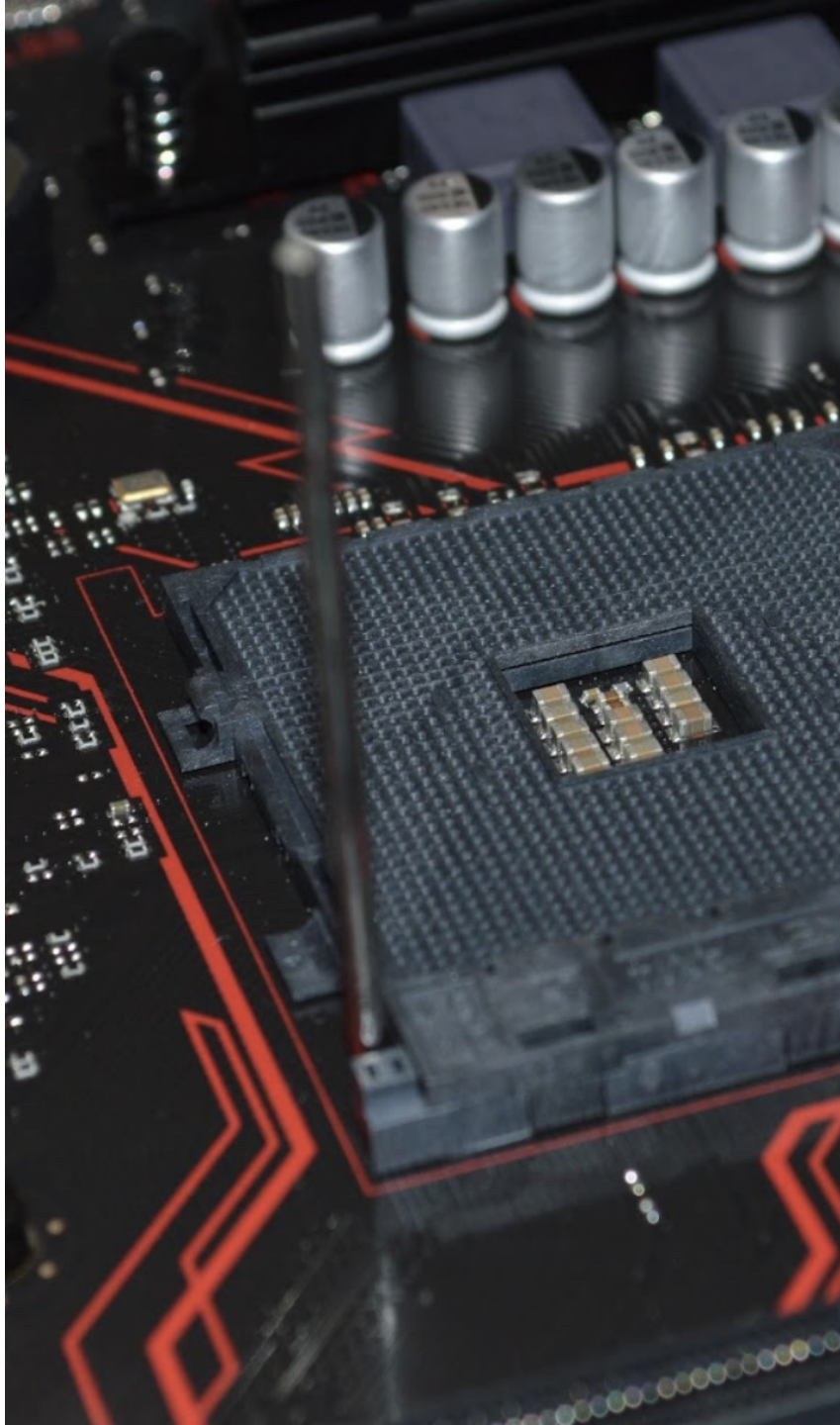
Step 3: Compilation

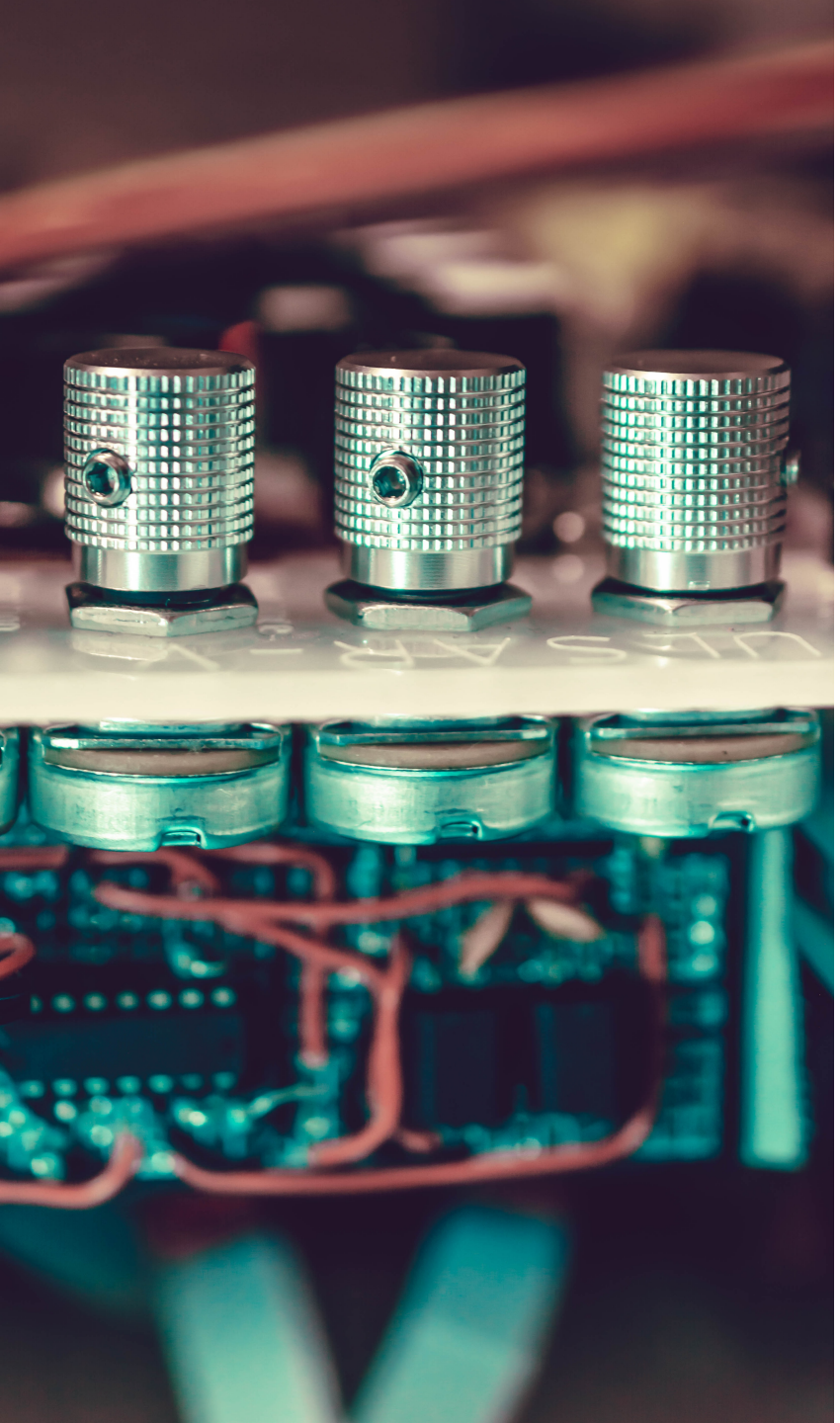
Delivery Hero
Logistics

Why build the same things over and over again?

> “Insanity is doing the same thing over and over and expecting different results.”

Albert Einstein





Delivery Hero
Logistics

Same Input + Same Process = Same Output

This optimization idea is used in many places. So when building the same source files with the same compiler, we should get the same binaries.

Essentially, we'll cache the build results.

Delivery Hero
Logistics

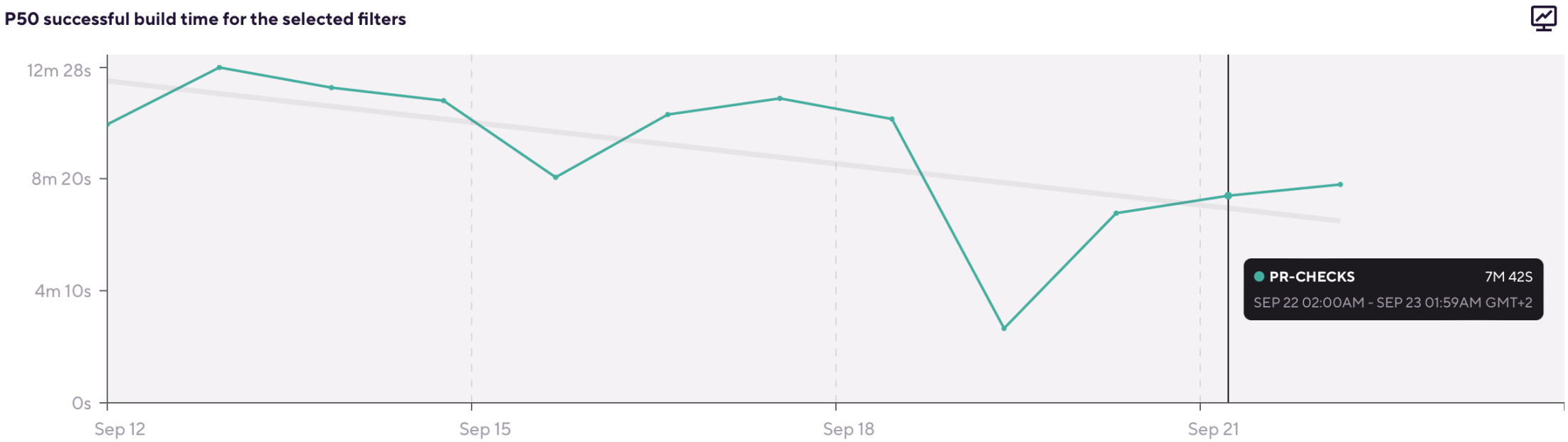
What we'll do

We'll move from using external dependencies compiled by SPM at build time, to using pre-compiled versions of the same dependencies.

Pull Request Link 🔗

Delivery Hero
Logistics

Step 3: Stats



Delivery Hero
Logistics

Step 3: Results

We made every build compile ca. 4 minutes faster (on CI and cold builds on engineer's machines).

CI credits use: We are using 36% fewer CI credits on a typical PR check pipeline.

Engineer focus: going from 11 to ca. 7 minutes per build (almost no time to make tea while CI compiles ☕).

Delivery Hero
Logistics

Summary

We started from spending ca. 32-35 minutes per PR checks pipeline on the CI.

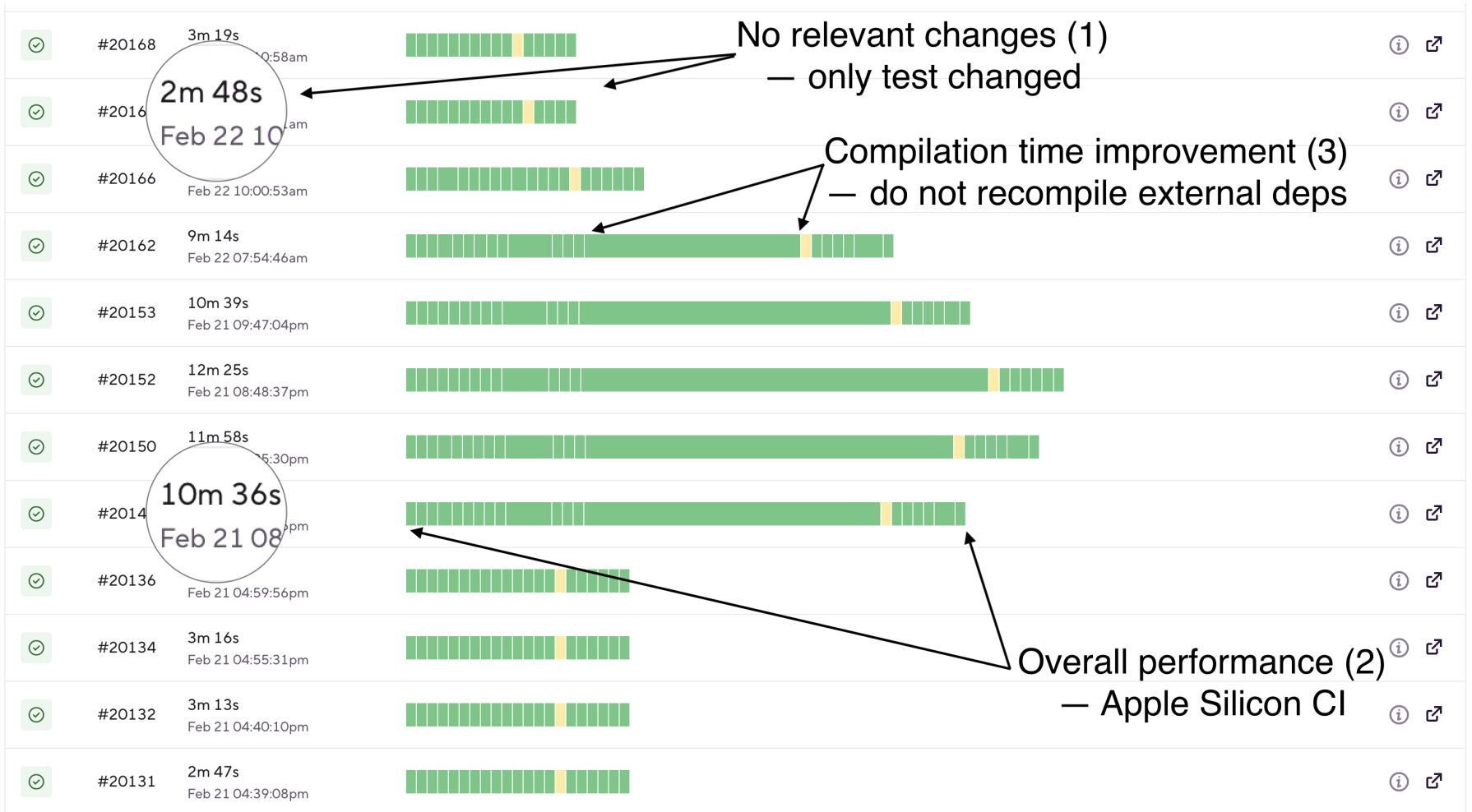
Now Rider Application is compiled and tested on the CI in average 2-7 minutes.

In 7 minutes we execute ca. 2300 unit tests and two critical UI tests.

In case the change is not relevant to the code, build takes only 2 minutes.

Delivery Hero
Logistics

State Feb 2023



Going forward (Vision)

Build performance

Thanks to the modular structure, we are able to go beyond what we have now.

We can ship pre-compiled modules, only building from the source the module engineer is currently working on.

Going forward (Vision)

Testing what changed

Analyze the scope of the new release. Regression testing would rely on the information about which modules were changed in the current release.

We can execute less tests on the modules that were not directly changed.

We can skip testing modules that were not affected by any change.

Delivery Hero
Logistics

Thanks for listening!

Questions