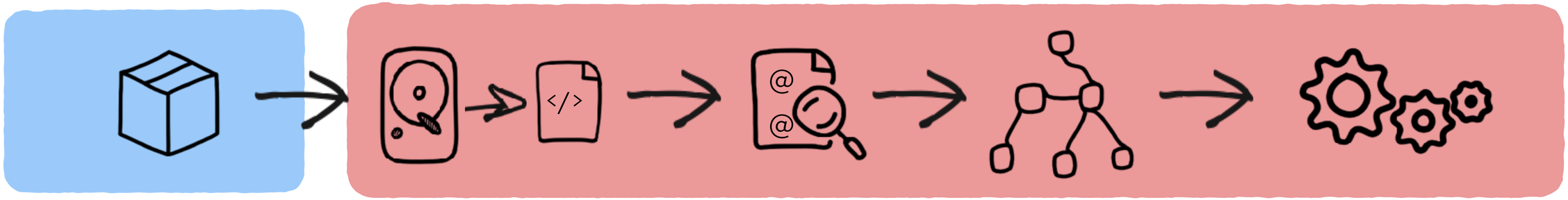
build time *runtime*



packaging
(maven, gradle...)

load and parse
- config files
- properties
- yaml
- xml
- etc.

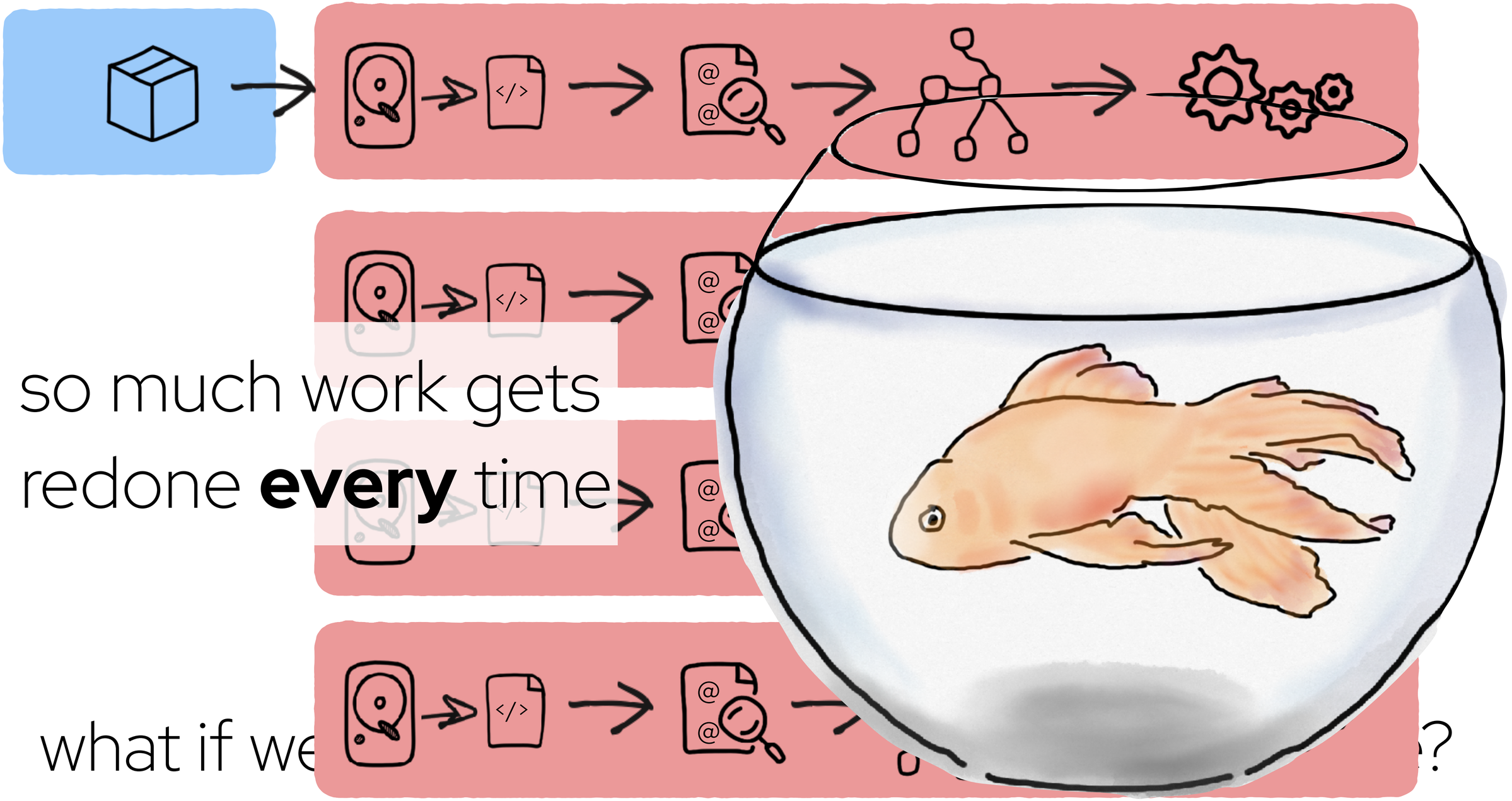
- classpath scanning
- annotation discovery
- attempt to load class to enable/disable features

build a metamodel of the world

start
- thread pools
- I/O
- etc.

ready to do work!

how does a java framework start?



so much work gets redone **every** time

what if we

?

speed example: JTA auto-wiring

```
Class.forName(“LikelyJTAImplementation”);
Class.forName(“APossibleJTAImplementation”);
Class.forName(“AnotherJTAImplementation”);
…
Class.forName(“NicheJTAImplementation”);
Class.forName(“VeryNicheJTAImplementation”);
```

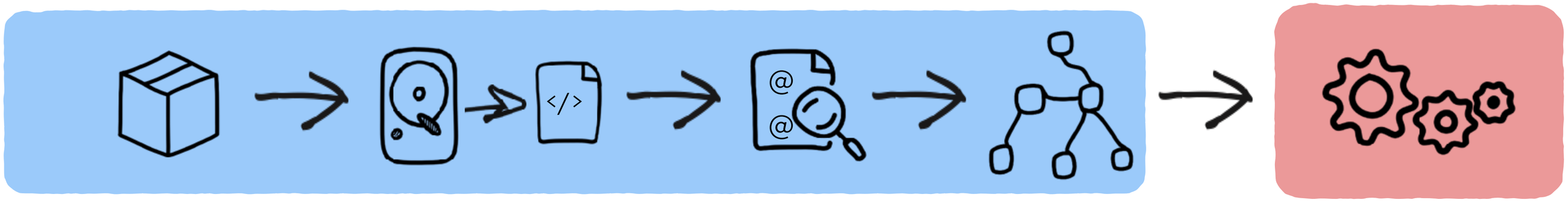
~129

auto-wiring attempts

every `ClassNotFoundException`

how do we fix all this?

build time *runtime*

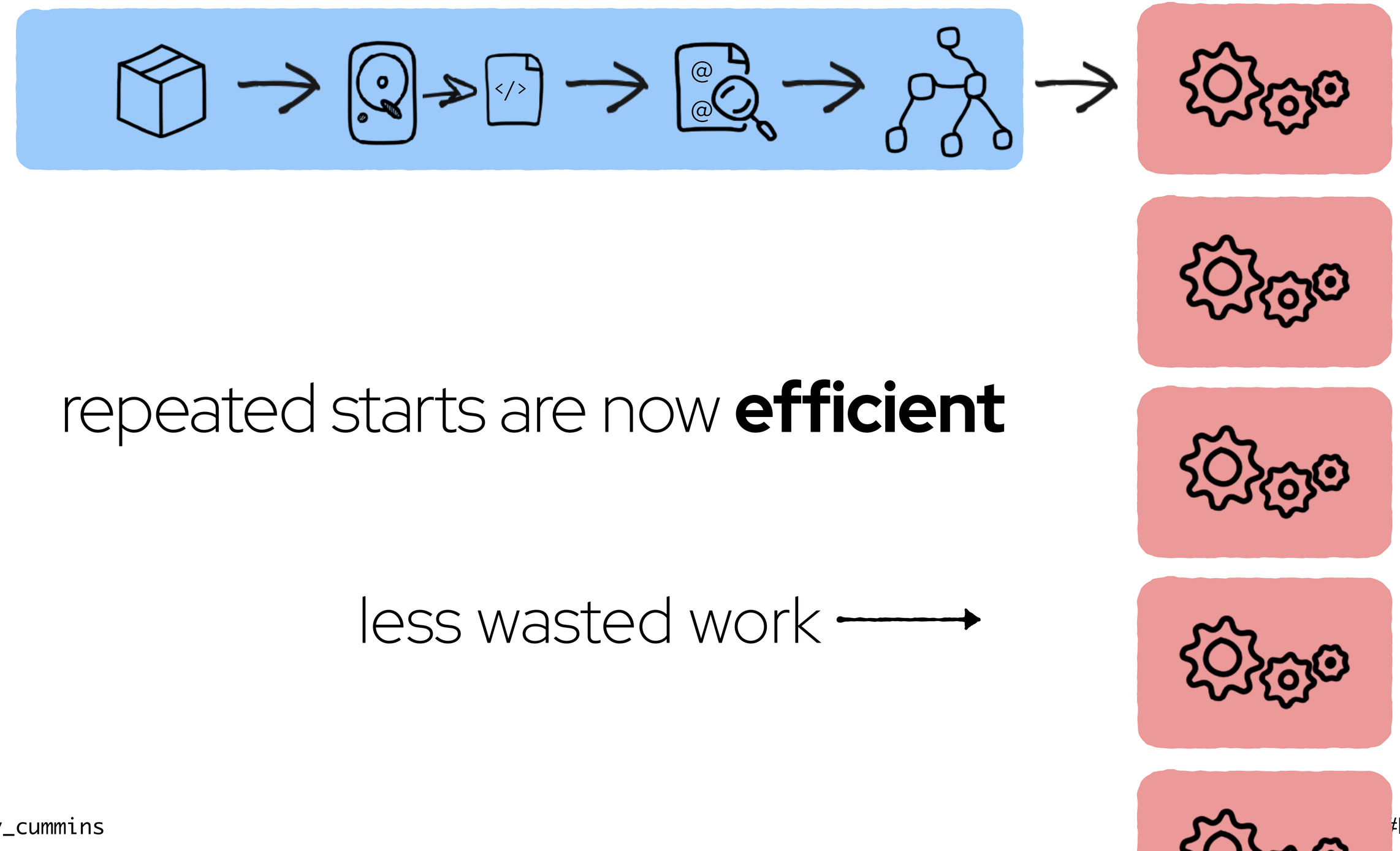


start

- thread pools
- I/O
- etc.

ready to do work!

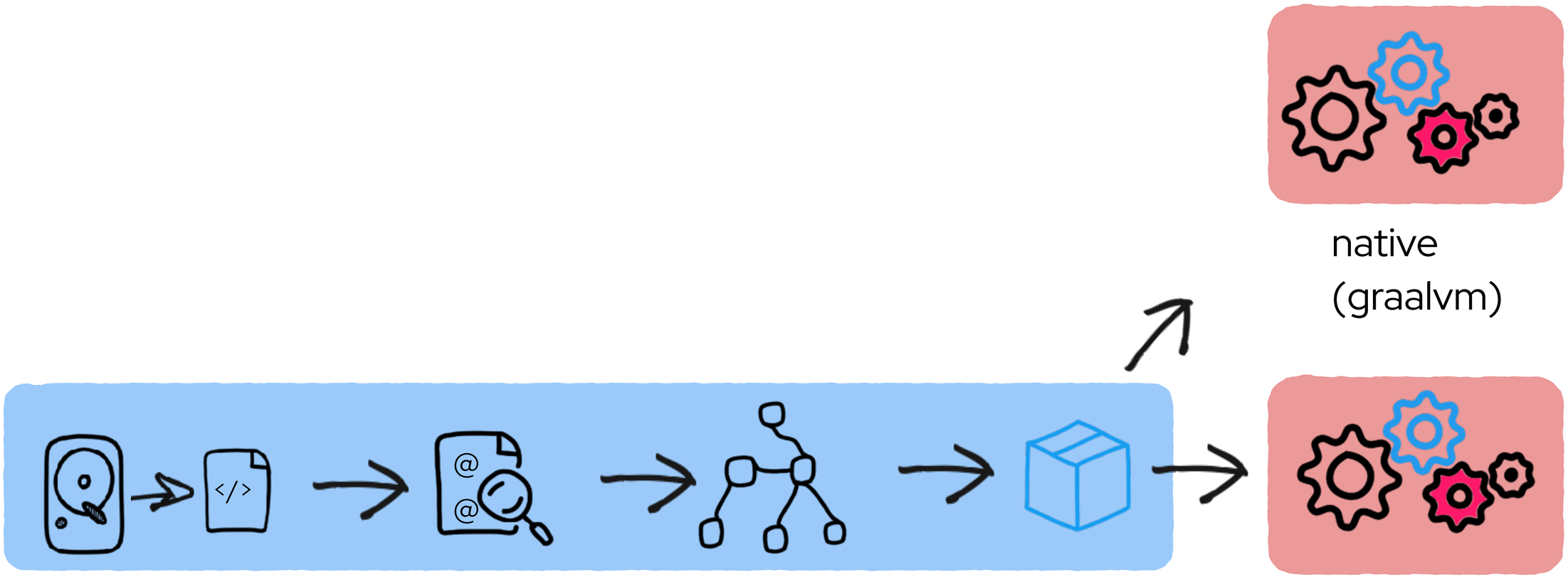
what if we initialize at build time?



repeated starts are now **efficient**

less wasted work ⟶

the Quarkus way enables native compilation



ok but we don't need native

quarkus is faster and smaller than legacy frameworks, even running on the jvm

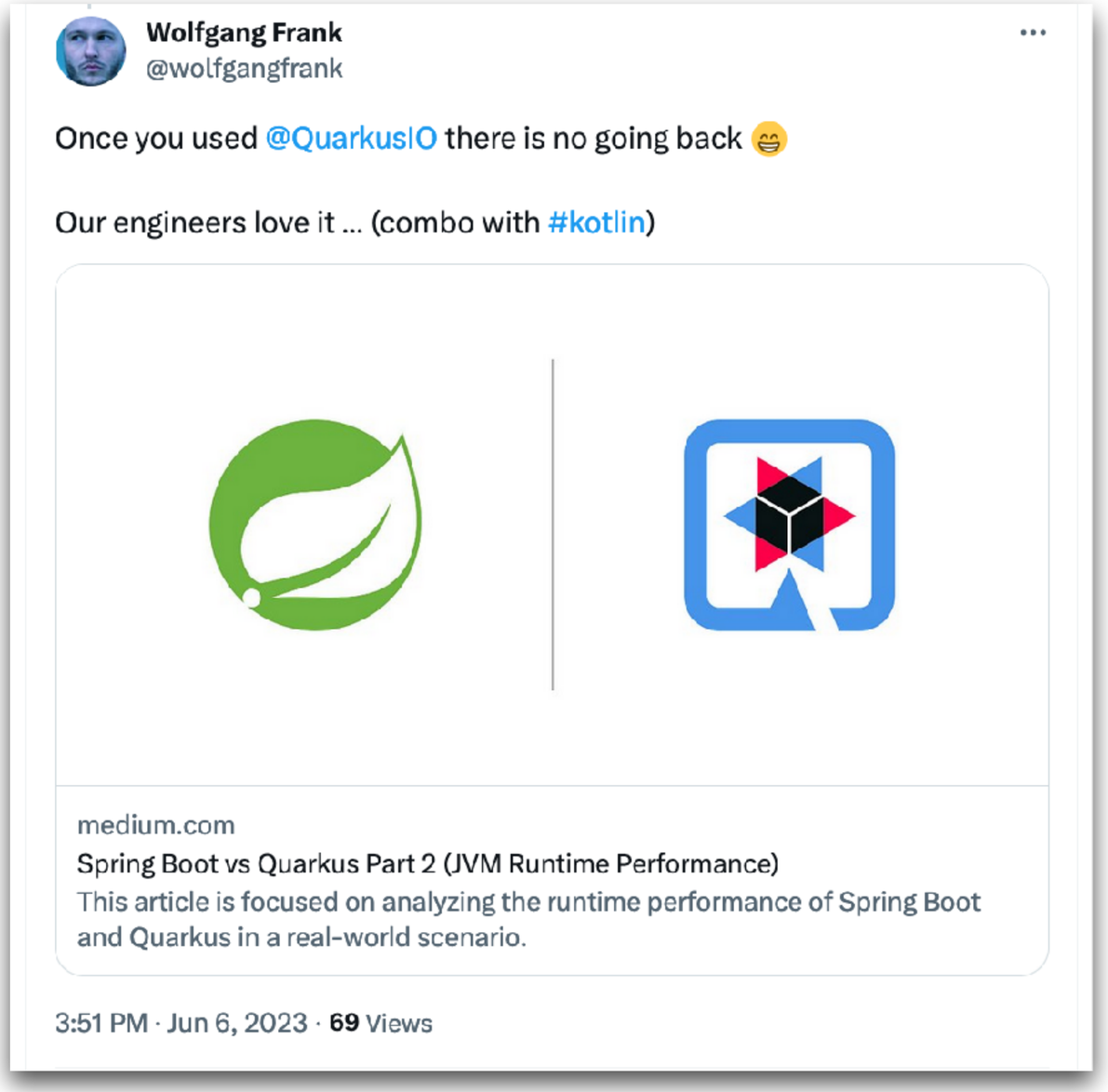
doing more up-front

- speeds up start
- shrinks memory footprint
- improves throughput (!)

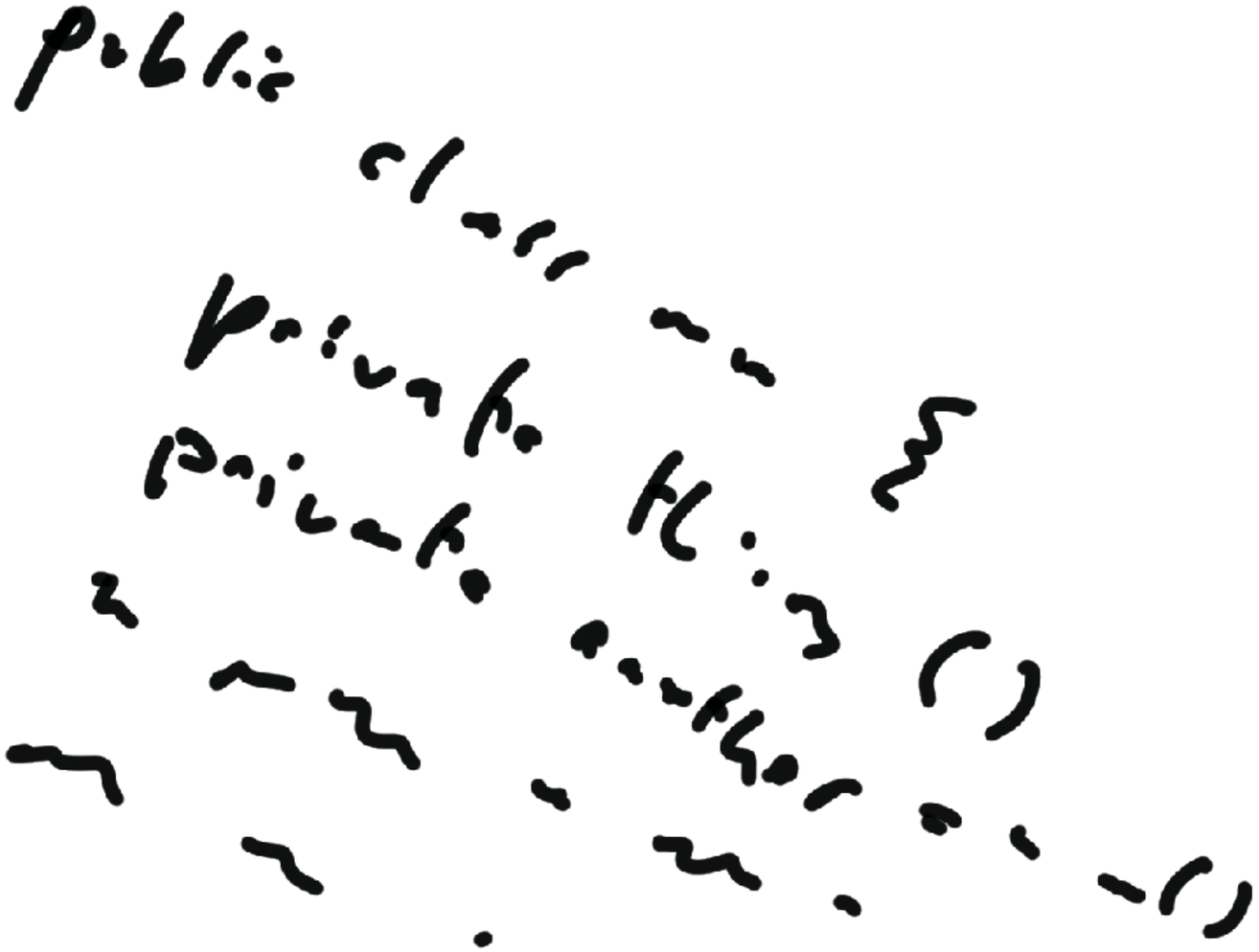
ok but is performance all there is?

ok but is performance all there is?

developer joy


Wolfgang Frank
@wolfgangfrank
Once you used @QuarkusIO there is no going back
Our engineers love it ... (combo with #kotlin)
medium.com
Spring Boot vs Quarkus Part 2 (JVM Runtime Performance)
This article is focused on analyzing the runtime performance of Spring Boot and Quarkus in a real-world scenario.
3:51 PM · Jun 6, 2023 · 69 Views

to the **code**!



```
mvn quarkus:dev
```

zero-config live coding

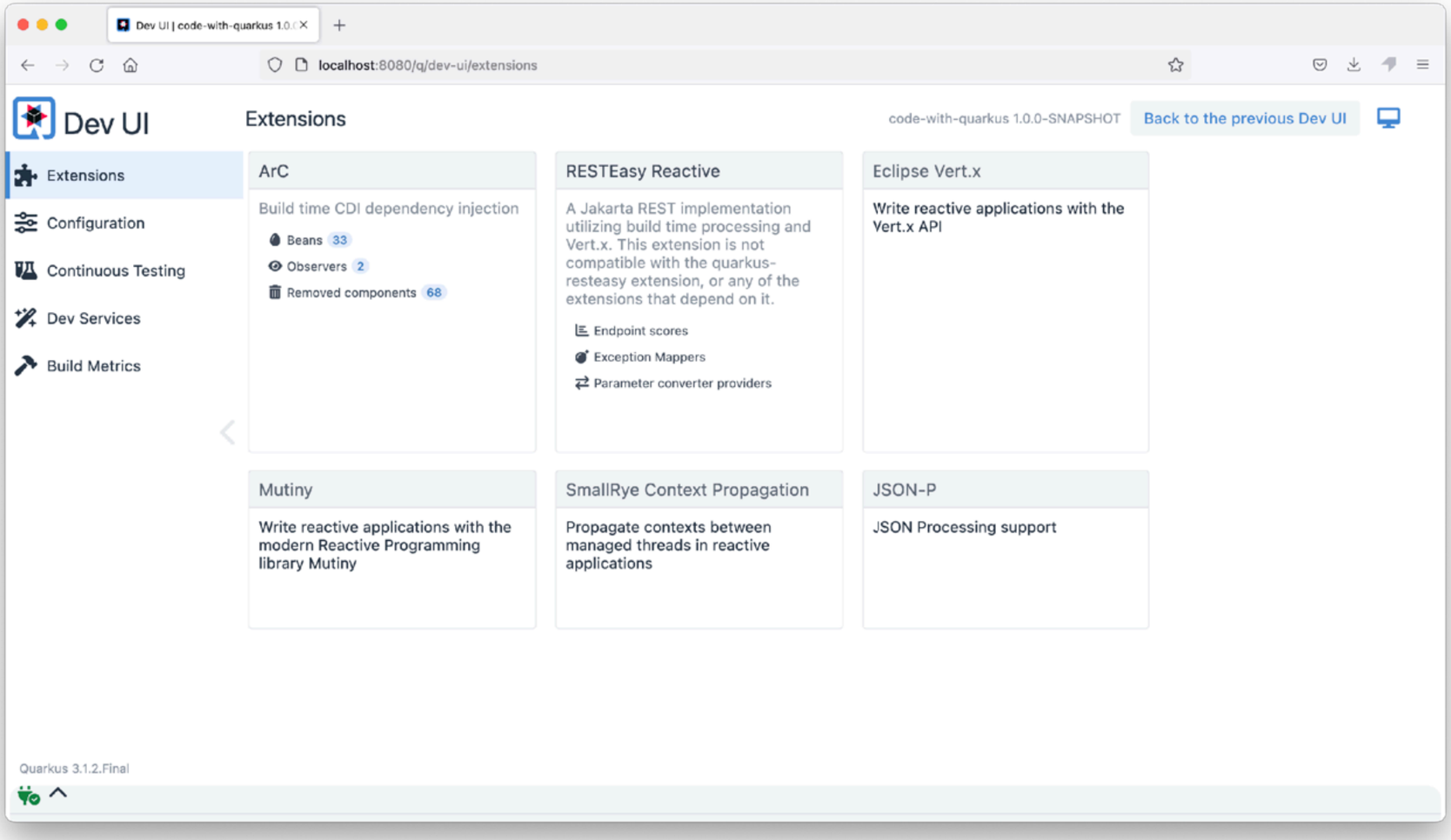
`mvn quarkus:dev`

continuous testing

tests are run on every code change

“reverse code coverage” means only relevant tests run

developer UI



zero-config testcontainers

testcontainers integration

```
@TestConfiguration(proxyBeanMethods = false)
public class ContainersConfig {
    @Bean
    @ServiceConnection
    public PostgreSQLContainer<?> postgres() {
        return new PostgreSQLContainer<>(DockerImageName.parse("postgres:14"));
    }
}

public class TestApplication {
    public static void main(String[] args) {
        SpringApplication
                .from(MySpringDataApplication::main)
                .with(ContainersConfig.class)
                .run(args);
    }
}
```

```
@Import(ContainersConfig.class)
```

zero-config testcontainers integration

```
# configure your datasource
quarkus.datasource.db-kind = postgresql
quarkus.datasource.username = sarah
quarkus.datasource.password = connor
quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/mydatabase

# drop and create the database at startup
quarkus.hibernate-orm.database.generation = drop-and-create
```

the only thing you need to do to make testcontainers work is not configure anything

quarkus also auto-invokes flyway and liquibase

auto-provision services

"dev services"
using testcontainers under the hood

- databases
- redis
- keycloak
- kafka
- elasticsearch
- kubernetes
- ... or add your own

more opinions, less boilerplate

example: logging

```
package com.example;
import io.quarkus.logging.Log;

public class MyService {
    private static final Logger log = Logger.getLogger(MyService.class);

    public void doSomething() {
        Log.info("It works!");
    }
}
```

example: declaring an application

```
package org.acme;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringDemo {

        public static void main(String[] args) {
                SpringApplication.run(SpringDemo.class, args);
        }

}
```

example: panache + hibernate

```
@ApplicationScoped
public class GreetingRepository implements PanacheRepository<Greeting> {

    public Entity findByName(int name) {
        return find("name", name).firstResult();
    }

}
```

DAO

```
    void delete(Entity entity) {}


    Entity findById(Id id) {}


    List<Entity> list(String query, Sort sort, Object... params) {
        return null;
    }

    Stream<Entity> stream(String query, Object... params) {
        return null;
    }

    long count() {
        return 0;
    }

    long count(String query, Object... params) {
        return 0;
    }

}
```

repository pattern

```
@Entity
public class Greeting extends PanacheEntity {

    public String name;

    public LocalDate issued;

    @Version
    public int version;

    public static List<Greeting> getTodaysGreetings() {
        return list("date", LocalDate.now());
    }
}
```

active record pattern

“but isn’t that dynamism expensive?”

no.

it’s done at **build** time

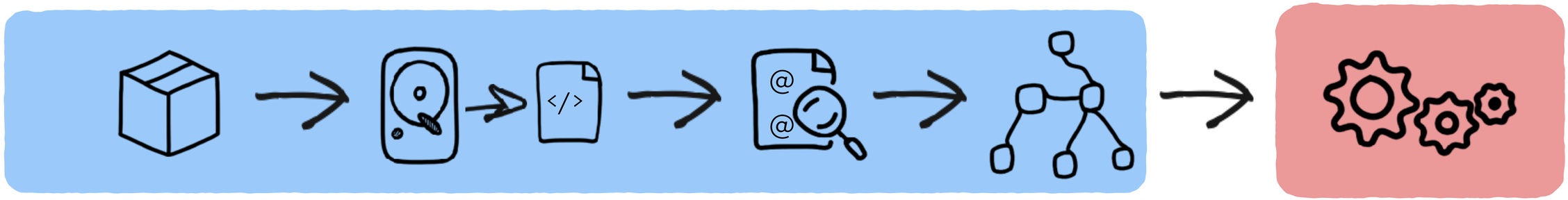
ok but i don't like magic

enrichment happens at build time

no performance drag

doing more up-front enables better devex

we can do cool code introspections here that would be too expensive and annoying to do at runtime



ok but i still don't like magic

the old ways all still work

but you don't **have** to type all
the stuff unless you **want** to

ok but we use spring

for existing applications, there **is** a trade-off there.

ok but we use spring

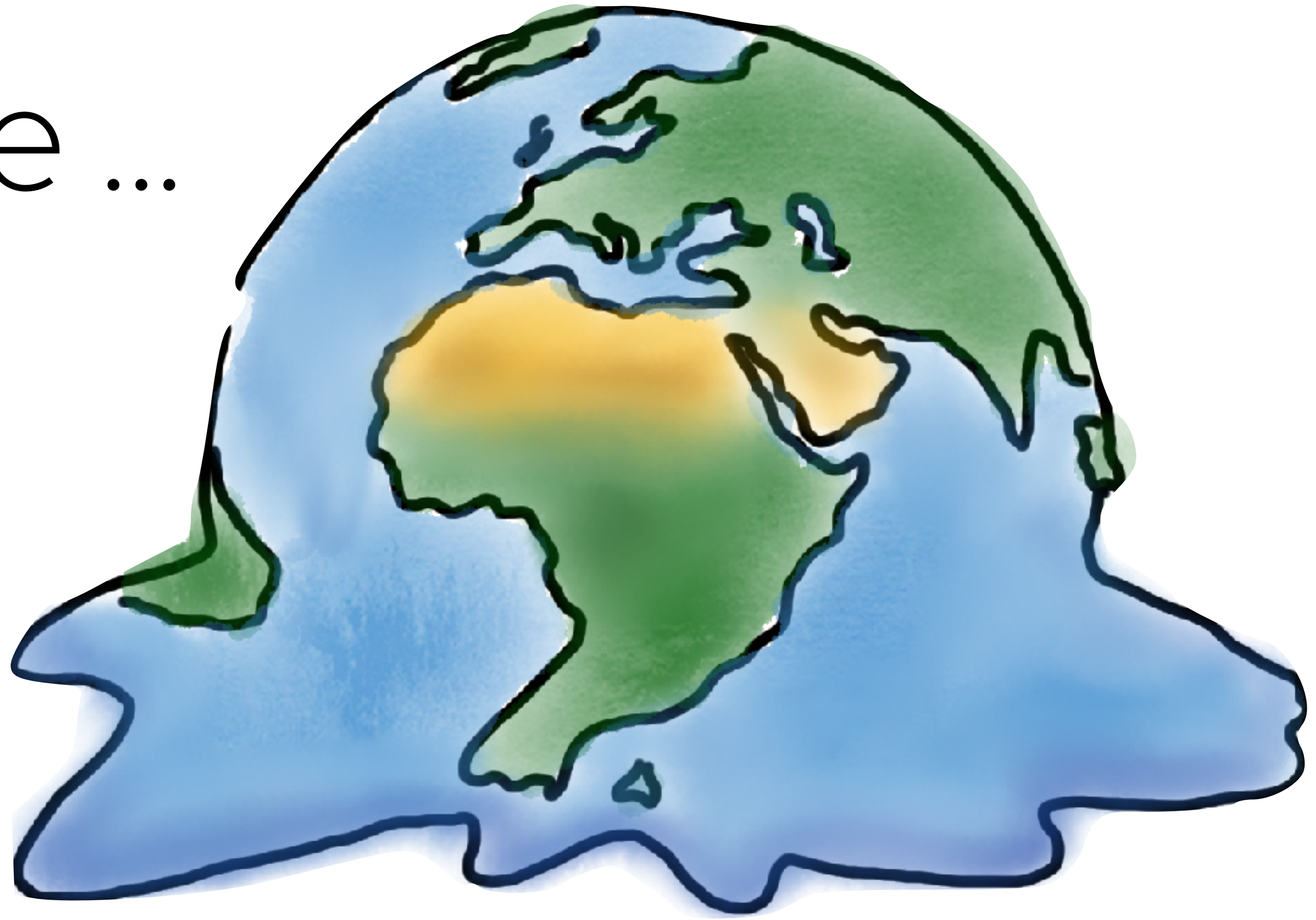
option 1: compatibility libraries

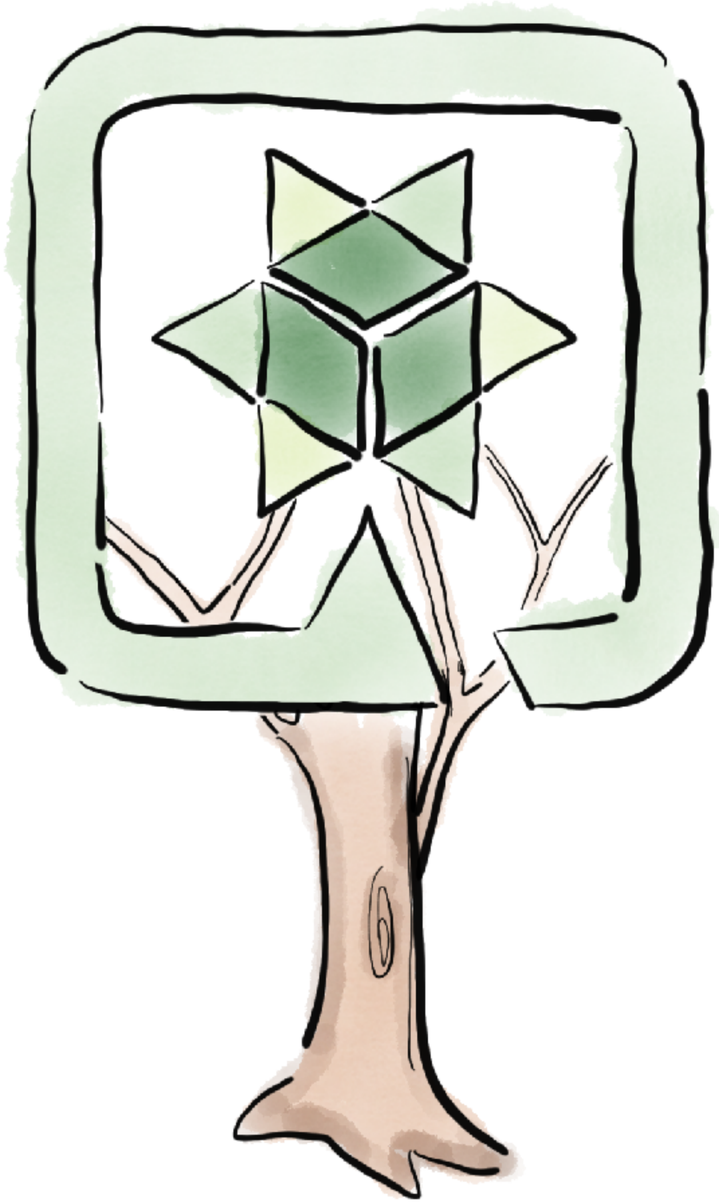
```
<dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-spring-web</artifactId>
</dependency>
<dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-spring-data-jpa</artifactId>
</dependency>
```

option 2: migration tooling

- migration toolkit for applications (mta)
- windup
- open rewrite

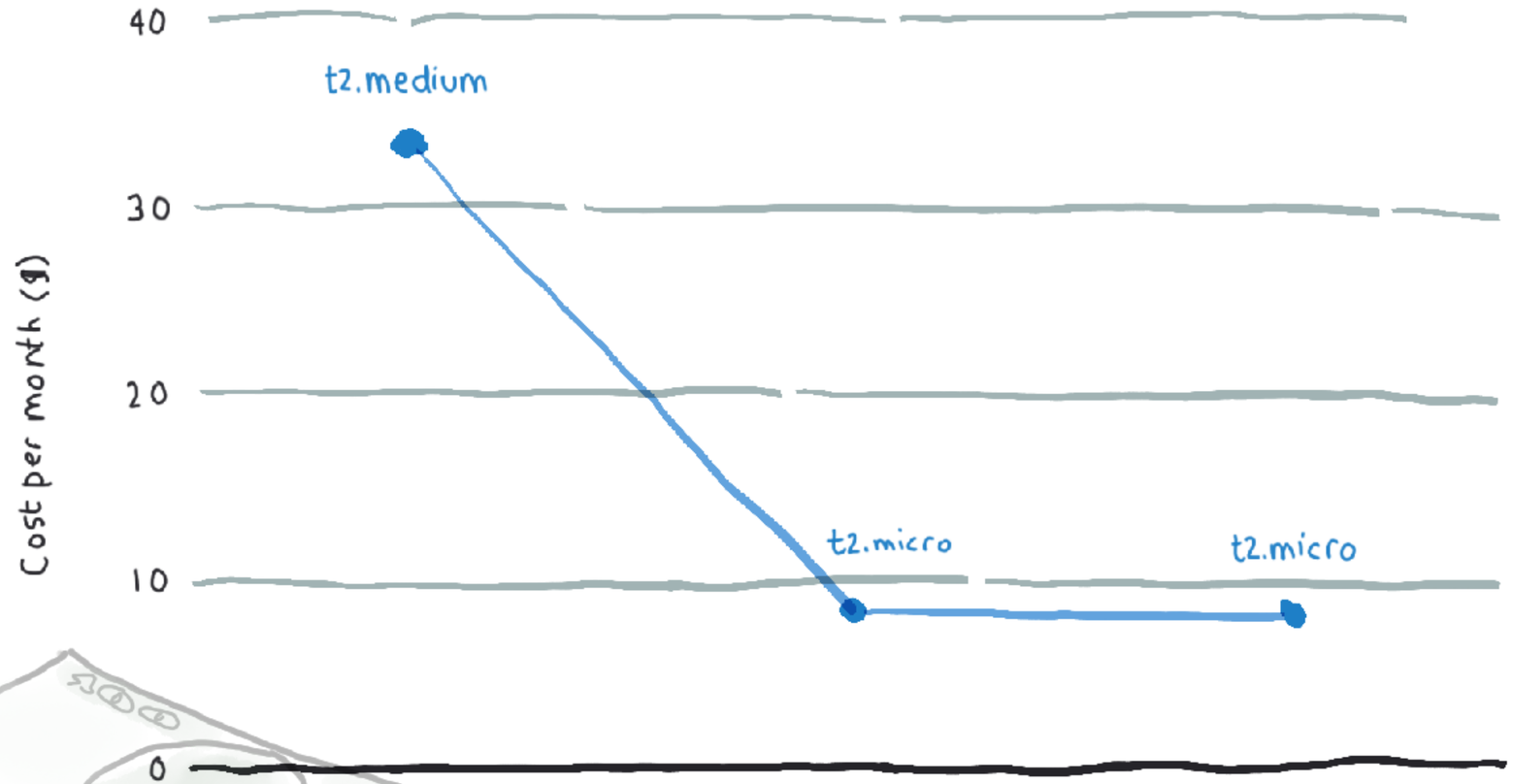
the big picture ...

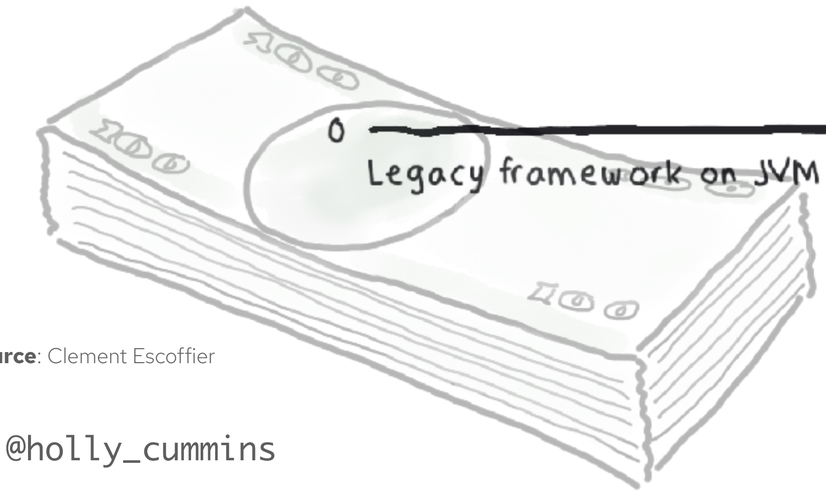




does being faster
and lighter mean
quarkus is **greener**?

experiment 1: cloud





Source: Clement Escoffier

Setup:

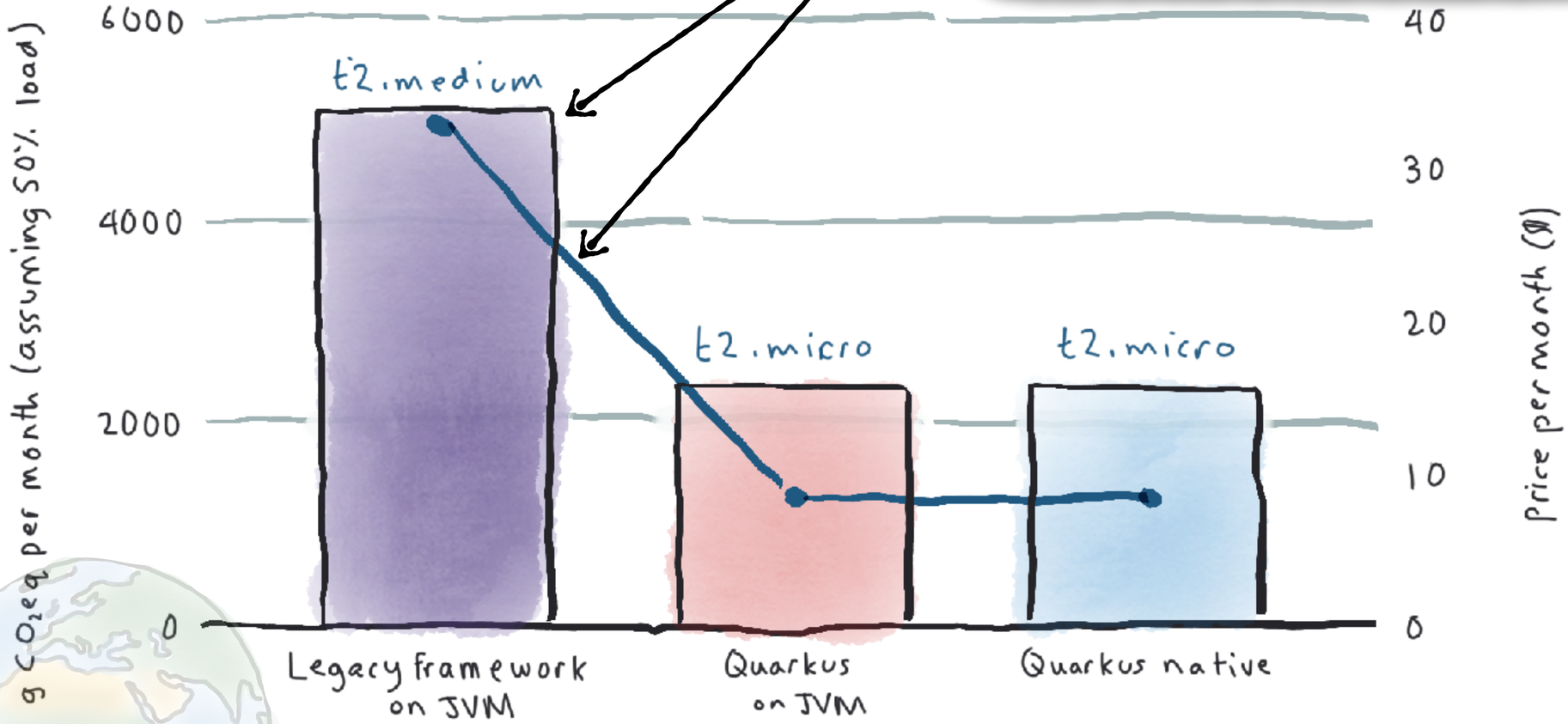
- 800 requests/second, over 20 days
- SLA > 99%
- AWS instances

Assumptions:

- Costs are for us-east-1 data centre

interpolated carbon metrics

the *carbon* is lower because the cost is lower



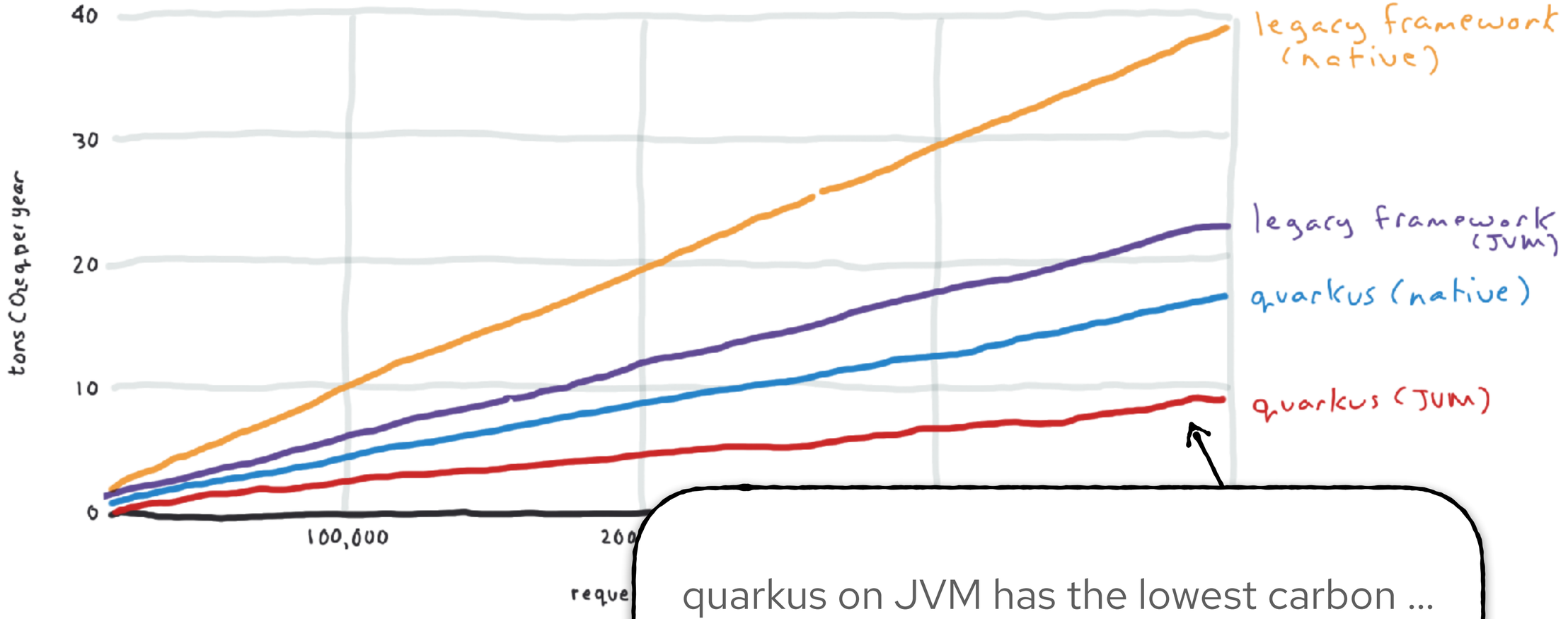
Source: Clement Escoffier x Teads

Setup:

- 800 requests/second, over 20 days
- SLA > 99%

Assumptions:

experiment 2: RAPL measurements



quarkus on JVM has the lowest carbon ...
because it has the highest *throughput*

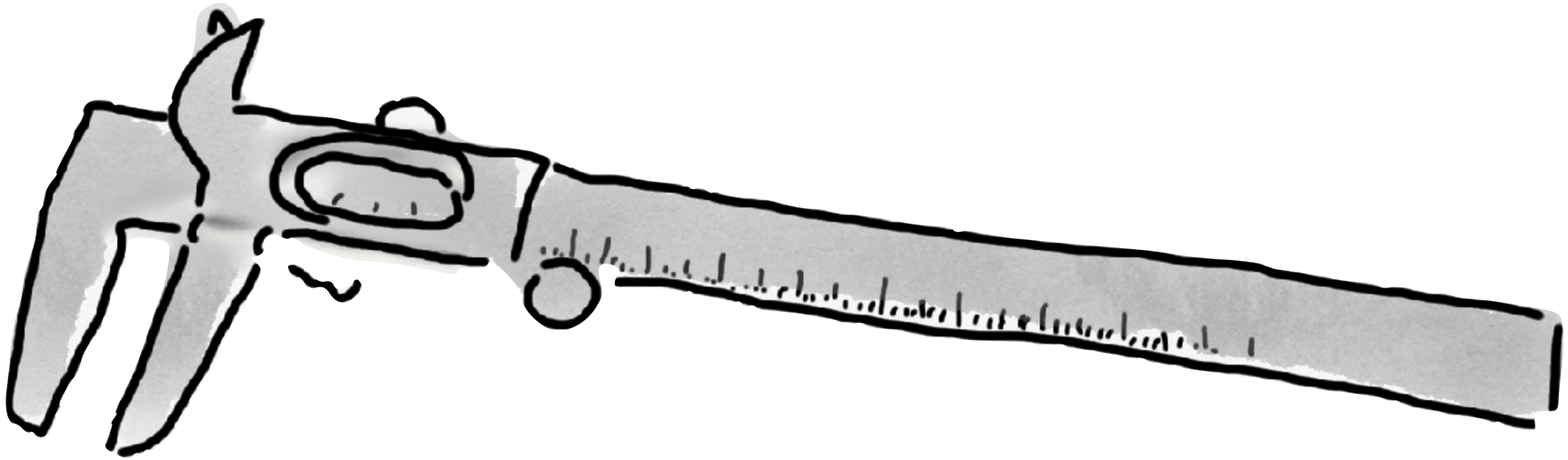
Setup:

- REST + CRUD
- large heap
- RAPL energy measurement
- US energy mix

Source: John O'Hara

carbon measurements: conclusions

- quarkus cuts carbon by ~2-3x*
- native consumes **more** carbon than JVM



tl;dpa

(too long didn't pay attention)

frictionless development experience

auto-provision services
zero-config live coding
continuous testing
developer UI

deployment density

lower cloud bill

Medium

Nano

greener

thank you!

Holly Cummins
Red Hat

slides

https://hollycummins.com/quarkus-cheaper-greener-happier-summit-connect/