Fantastic Frameworks

...and How to Avoid Them

Grzegorz Piwowarek

@pivovarit

{ 4comprehension.com }

Senior Software Engineer @ Hazelcast

Trainer @ Bottega IT Minds

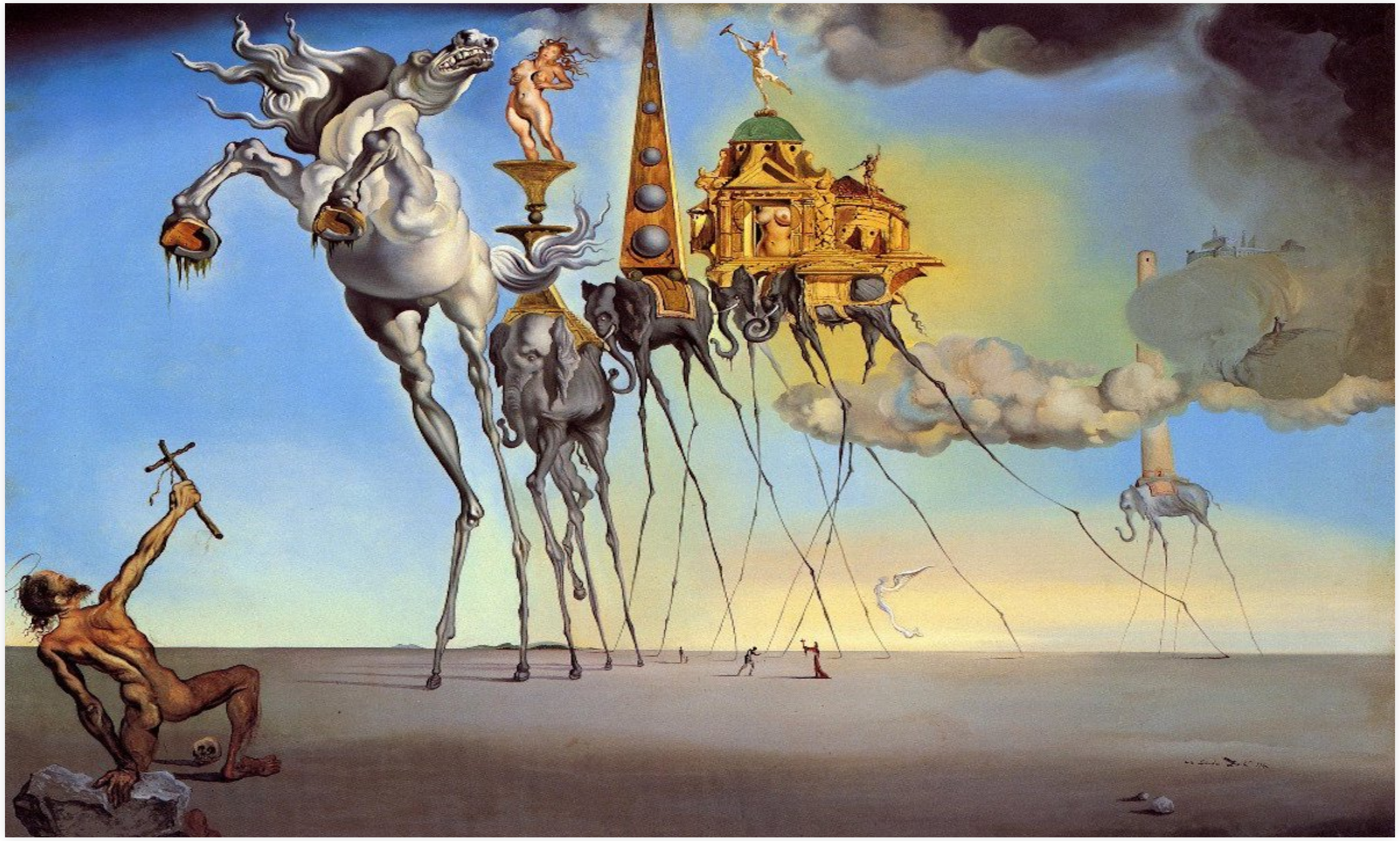
@pivovarit

Developers back in 1996
I just fit Pokemon Red in 512kB
Developers now
Help, my hello world uses 1GB of RAM

@pivovarit

"Regarding opiniated / unopiniated frameworks, I think they appeal to different types of developers. I now have 20 years of experience. In my first 5-8 years I loved frameworks like Spring. However, as I gained more experience, I felt I needed a framework's "opinion" less and less. Now I avoid them."

https://www.infoq.com/news/2018/10/the-road-to-micronaut-1.0



"Scala programmer confronts Java project that uses Maven, Spring, and Hibernate" - Salvador Dali, oil painting, 1946

https://twitter.com/progpaintings/status/723276501081190400

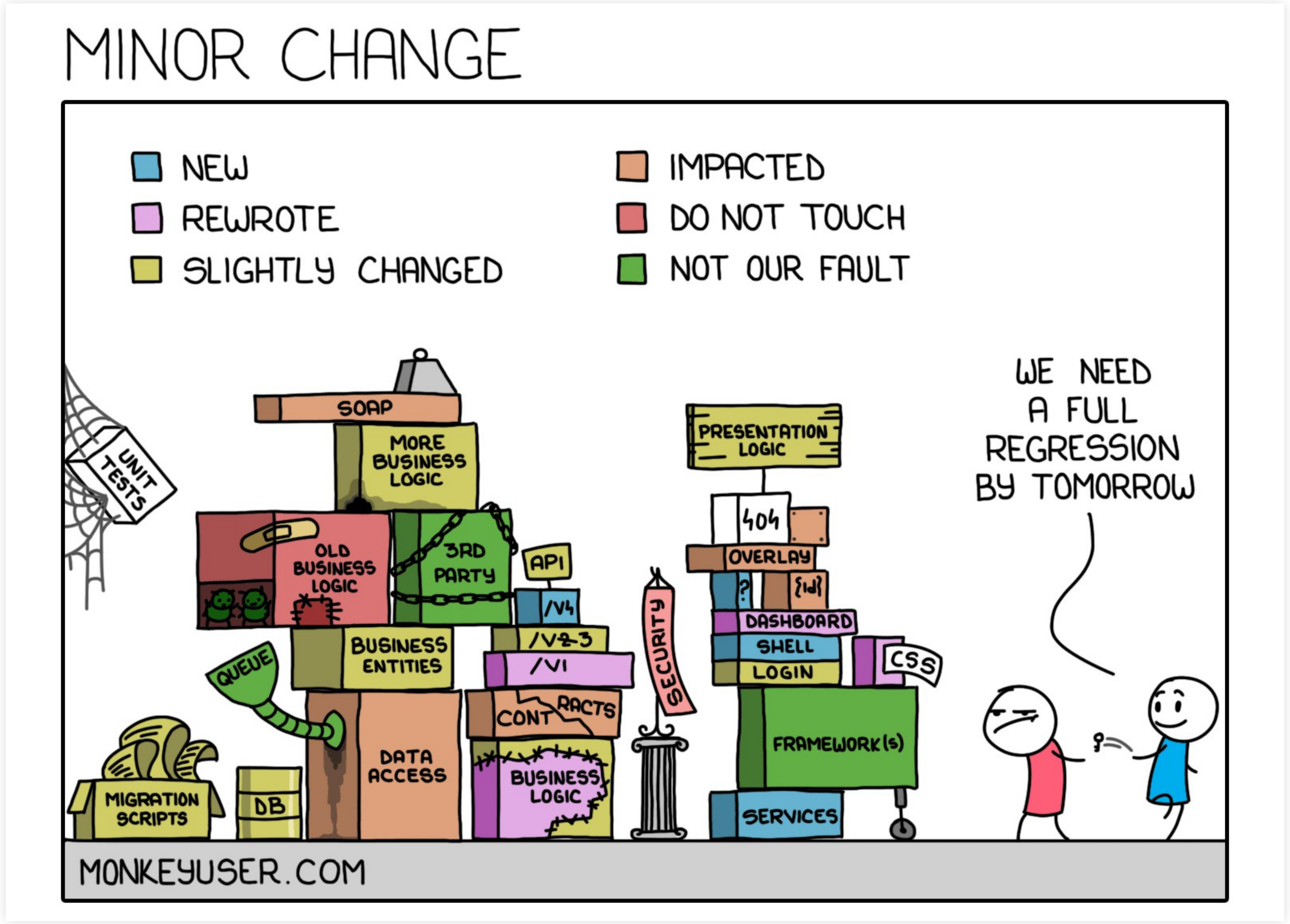
I. Invasive Frameworks

II. Magic

What's magic?

- Runtime reflection
- Classpath scanning
- Thread-locals
- Runtime annotation-processing
- Proxies/AOP

@pivovarit

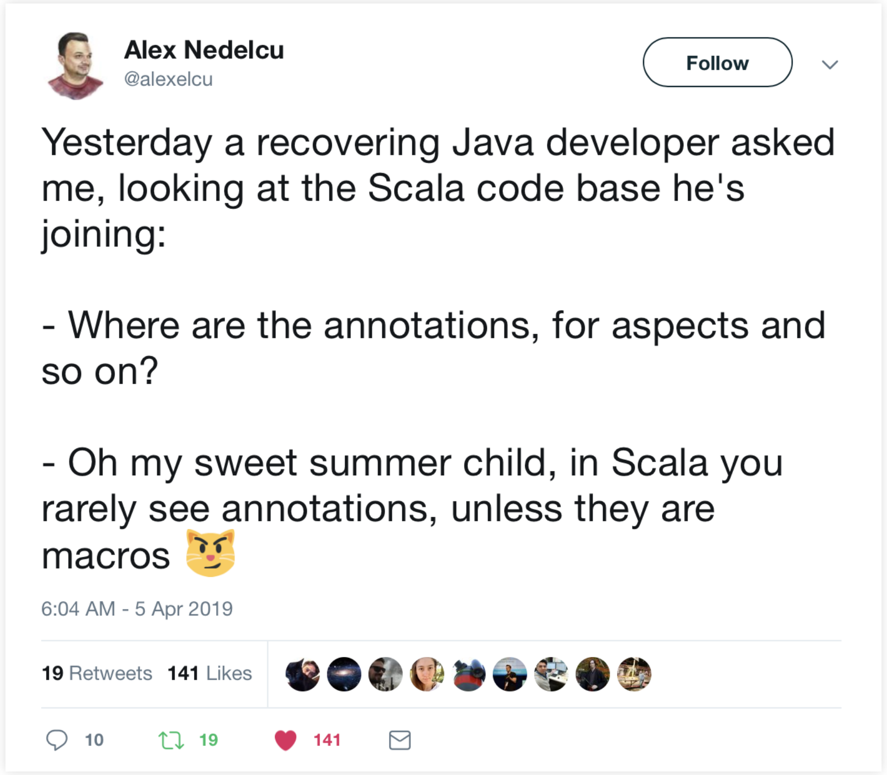

MINOR CHANGE
NEW
REWROTE
SLIGHTLY CHANGED
IMPACTED
DO NOT TOUCH
NOT OUR FAULT
UNIT TESTS
SOAP
MORE BUSINESS LOGIC
OLD BUSINESS LOGIC
3RD PARTY
API
/V4
/V2-3
/V1
BUSINESS ENTITIES
QUEUE
DATA ACCESS
CONTRACTS
BUSINESS LOGIC
MIGRATION SCRIPTS
DB
SECURITY
PRESENTATION LOGIC
404
OVERLAY
?
{id}
DASHBOARD
SHELL
LOGIN
CSS
FRAMEWORK(S)
SERVICES
WE NEED A FULL REGRESSION BY TOMORROW
MONKEYUSER.COM

@pivovarit

Meanwhile in Scala ecosystem...

(...) It's not a web-framework but rather a more general toolkit for providing and consuming HTTP-based services.

https://doc.akka.io/docs/akka-http/current/introduction.html#philosophy


Alex Nedelcu
@alexelcu

Yesterday a recovering Java developer asked me, looking at the Scala code base he's joining:

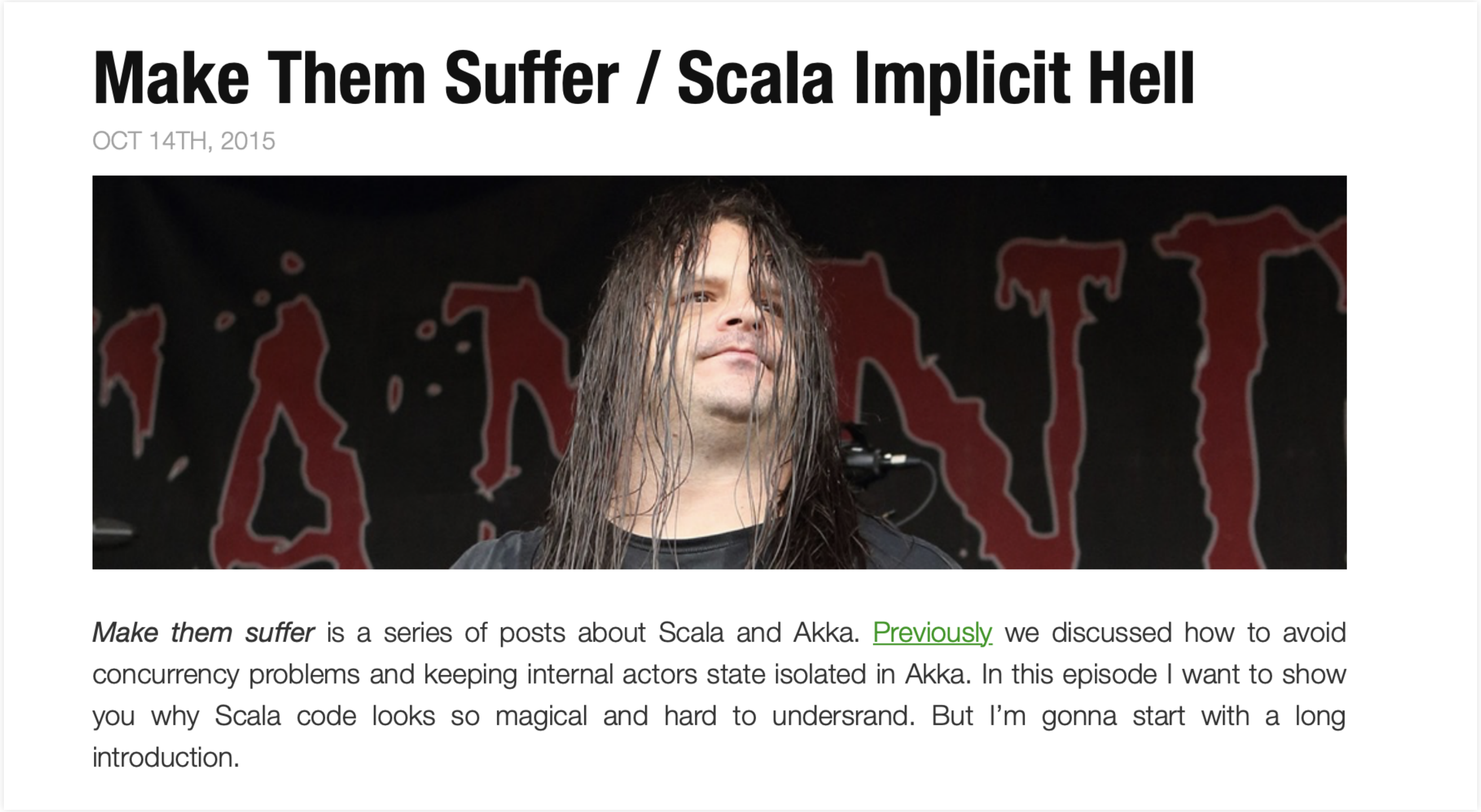
- Where are the annotations, for aspects and so on?

- Oh my sweet summer child, in Scala you rarely see annotations, unless they are macros 😼

6:04 AM - 5 Apr 2019

19 Retweets 141 Likes

https://twitter.com/alexelcu/status/1114016039841538048



Make Them Suffer / Scala Implicit Hell

OCT 14TH, 2015

Make them suffer is a series of posts about Scala and Akka. Previously we discussed how to avoid concurrency problems and keeping internal actors state isolated in Akka. In this episode I want to show you why Scala code looks so magical and hard to undersrand. But I'm gonna start with a long introduction.

http://spiridonov.pro/2015/10/14/scala-implicit-hell/

@pivovarit

Dependency Hell

Dependency hell is a colloquial term for the frustration of some software users who have installed software packages which have dependencies on specific versions of other software packages.

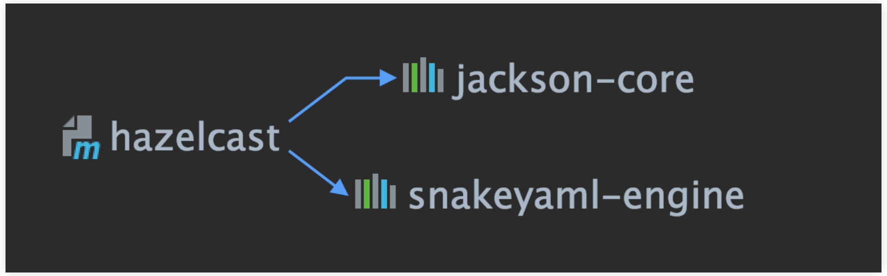
https://en.wikipedia.org/wiki/Dependency_hell

Zero-Dependencies

```
<dependencies>
    ...

    <dependency>
        <groupid>org.assertj</groupid>
        <artifactid>assertj-core</artifactid>
        <version>${assertj.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Not that easy to apply in practice

… but internal dependencies can be shaded*



This and other architectural decisions can be enforced

arch-unit

```java
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.classes;

 @Test
 void shouldHaveZeroDependencies() {
     classes()
       .that().resideInAPackage("com.pivovarit.collectors")
       .should()
       .onlyDependOnClassesThat()
       .resideInAnyPackage("com.pivovarit.collectors", "java..")
       .as("the library should depend only on core Java classes")
       .because("so that users don't experience dependency hell")
       .check(classes);
 }

@Test
void shouldHaveSinglePackage() {
    classes()
      .should().resideInAPackage("com.pivovarit.collectors")
      .check(classes);
}
```

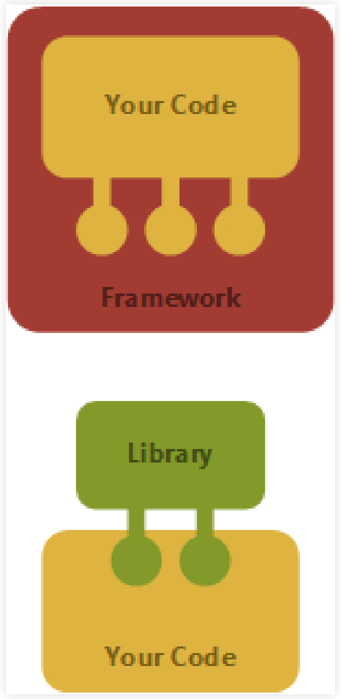
source

@pivovarit

Frameworks vs Libraries

Frameworks

Puts work into a frame



http://tomasp.net/blog/2015/library-frameworks/

@pivovarit

Frameworks exist beyond software

Franchizes

Project Management Methodologies (for example, Scrum)

Frameworks

Take time to learn and outdate quickly

Own you

Increase complexity/weight

Go out of fashion/get abandoned

Force you to keep up

Require expert help

Probably provide more than you need

Hazelcast IMDG/Jet

Is not a framework but a toolkit, it doesn't own your application.

(...) Hazelcast is also used in academia and research as a framework for distributed execution and storage.

(...) Hazelcast as its distributed execution framework.

https://en.wikipedia.org/wiki/Hazelcast

- Wouldn’t you like to focus on solving real problems and

let framework authors do the boring parts?

- Of course! but…

"All non-trivial abstractions, to some degree, are leaky."

https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/

So you spend time learning someone’s abstraction and not the underlying technologies

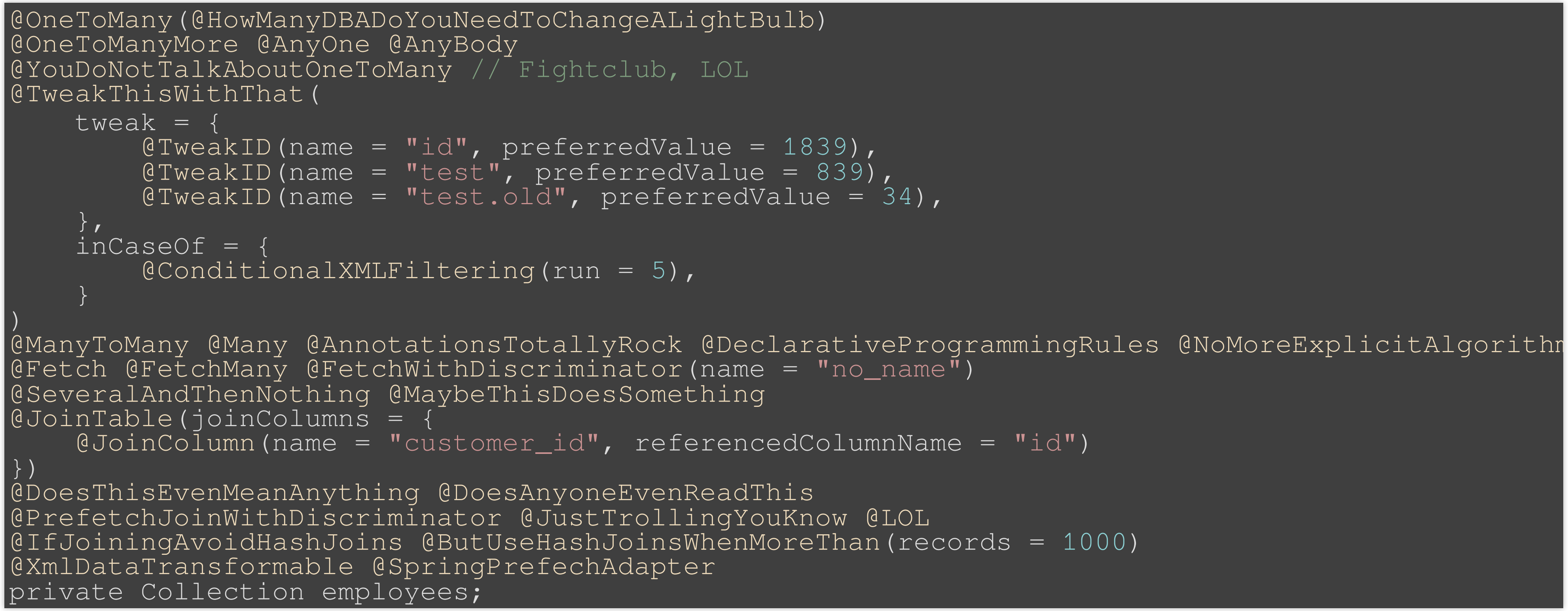
for example, ORM

JPA/Hibernate

Often seen: domain full of framework-specific annotations

```
package com.pivovarit.domain;

import javax.persistence.Column;
import javax.persistence.Entity;

@Entity
public class PersistedUser {

    private Long id;

    @Column
    private String name;
}
```

Not really a problem as long as the model is simple and graspable



```
@OneToMany(@HowManyDBADoYouNeedToChangeALightBulb)
@OneToManyMore @AnyOne @AnyBody
@YouDoNotTalkAboutOneToMany // Fightclub, LOL
@TweakThisWithThat(
    tweak = {
        @TweakID(name = "id", preferredValue = 1839),
        @TweakID(name = "test", preferredValue = 839),
        @TweakID(name = "test.old", preferredValue = 34),
    },
    inCaseOf = {
        @ConditionalXMLFiltering(run = 5),
    }
)
@ManyToMany @Many @AnnotationsTotallyRock @DeclarativeProgrammingRules @NoMoreExplicitAlgorith
@Fetch @FetchMany @FetchWithDiscriminator(name = "no_name")
@SeveralAndThenNothing @MaybeThisDoesSomething
@JoinTable(joinColumns = {
    @JoinColumn(name = "customer_id", referencedColumnName = "id")
})
@DoesThisEvenMeanAnything @DoesAnyoneEvenReadThis
@PrefetchJoinWithDiscriminator @JustTrollingYouKnow @LOL
@IfJoiningAvoidHashJoins @ButUseHashJoinsWhenMoreThan(records = 1000)
@XmlDataTransformable @SpringPrefechAdapter
private Collection employees;
```

http://www.annotatiomania.com

@pivovarit

SQL is complete and self-sufficient, too bad that not type-safe…

```
String sql = create
   .select(BOOK.TITLE, AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME)
   .from(BOOK)
   .join(AUTHOR).on(BOOK.AUTHOR_ID.eq(AUTHOR.ID))
   .where(BOOK.PUBLISHED_IN.eq(1948))
   .getSQL();
```

https://www.jooq.org

Kotlin Exposed

```
fun findLastRentalId(customerId: CustomerId): RentalId? =
  RentalTable
    .select { RentalTable.customerId eq customerId.value }
    .orderBy(RentalTable.startDate, true)
    .map { RentalId(it[RentalTable.id]) }
    .firstOrNull()
```

https://github.com/JetBrains/Exposed

JdbcTemplate

```
@Override
public Collection<Movie> findAll() {
    return jdbcTemplate.query("SELECT * FROM movie",
      JdbcTemplateMovieRepository::mapper);
}
```

```
private static Movie mapper(ResultSet resultSet, int i)
  throws SQLException {
    return new Movie(new MovieId(
      resultSet.getLong("id")),
      resultSet.getString("title"),
      MovieType.valueOf(resultSet.getString("type")));
}
```

But frameworks don't need to be that invasive as well, if you don't let them be.

Let's look again

```
package com.pivovarit.domain;

import javax.persistence.Column;
import javax.persistence.Entity;

@Entity
public class PersistedUser {

    private Long id;

    @Column
    private String name;
}
```

@pivovarit

ORM as plugin

```java
package com.pivovarit.domain;

public class User {
    private final Long id;
    private final String name;
}
```

```java
package com.pivovarit.persistence;

import javax.persistence.Column;
import javax.persistence.Entity;

@Entity
public class PersistedUser {

    private Long id;

    @Column
    private String name;
}
```

Spring

```java
import org.springframework.*;

@Component
public class FooFacade {

    @Autowired
    private FooService fooService;

    @PostConstruct
    public void foo() {
        fooService.foo();
    }
}
```

```java
public static void main(String[] args) {
    // ???
}
```

- - Tight coupling with the DI framework
- - Forces weak encapsulation
- - Forces mutability

```
import org.springframework.*;

@Component
public class FooFacade {

    private final FooService fooService;

    @Autowired
    public FooFacade(FooService fooService) {
        this.fooService = fooService;
    }

    @PostConstruct
    public void foo() {
        fooService.foo();
    }
}
```

```
public static void main(String[] args) {
    FooFacade fooFacade = new FooFacade(new FooService());
    fooFacade.foo();
}
```

- - Tight coupling with the DI framework
- + Can be instantiated independently
- + Internals can be encapsulated

@pivovarit

```java
import org.springframework.*;

@Component
public class FooFacade {

    private final FooService fooService;

    // @Autowired
    public FooFacade(FooService fooService) {
        this.fooService = fooService;
    }

    @PostConstruct
    public void foo() {
        fooService.foo();
    }
}
```

```java
public static void main(String[] args) {
    FooFacade fooFacade = new FooFacade(new FooService());
    fooFacade.foo();
}
```

- - Tight coupling with the DI framework
- + Can be instantiated independently
- + Internals can be encapsulated

```java
public class FooFacade {

    private final FooService fooService;

    public FooFacade(FooService fooService) {
        this.fooService = fooService;
        foo();
    }

    void foo() {
        fooService.foo();
    }
}
```

```java
public static void main(String[] args) {
    FooFacade fooFacade = new FooFacade(new FooService());
}
```

- ...but where's the config?

```java
import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration
public class FooConfiguration {

    @Bean
    FooService fooService() {
        return new FooService();
    }

    @Bean
    FooFacade fooFacade(FooService fooService) {
        return new FooFacade(fooService);
    }
}
```

- Domain code free of framework configuration

@pivovarit

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FooConfiguration {

    @Bean
    FooFacade fooFacade() {
        return new FooFacade(new FooService());
    }
}
```

- Domain code free of framework configuration

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class FooFacade {

    @Autowired
    private FooService fooService;

    public void foo() {
        fooService.foo();
    }
}
```

Same framework - both invasive and non-invasive

```java
public class FooFacade {

    private final FooService fooService;

    public FooFacade(FooService fooService) {
        this.fooService = fooService;
    }
}
```

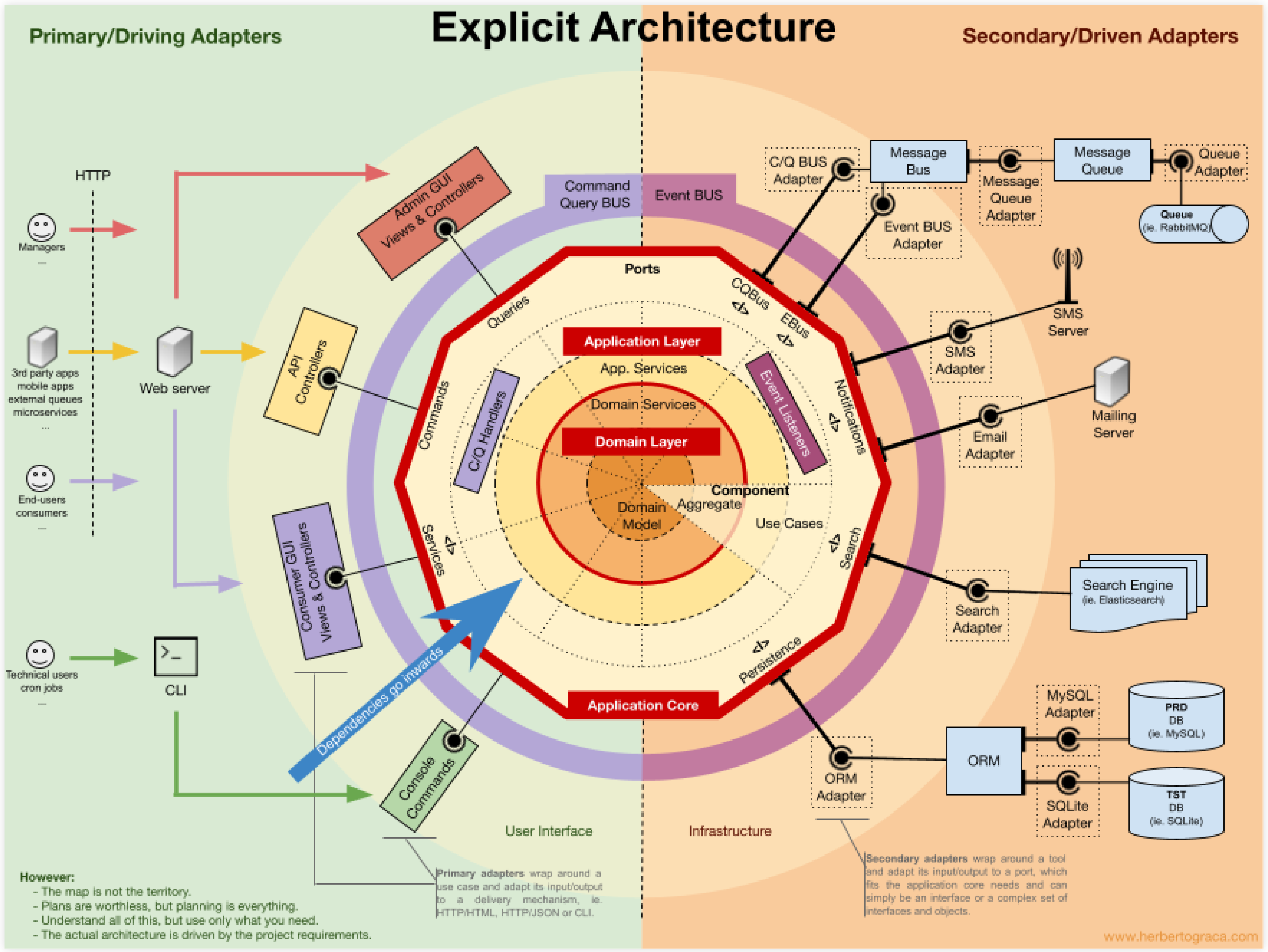
```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class FooConfiguration {

    @Bean
    FooFacade fooFacade() {
        return new FooFacade(new FooService());
    }
}
```

Composition/Delegation is your friend

```
public interface UserSessionRepository {
    // ...
}
```

```
public class CachingUserSessionRepository
  implements UserSessionRepository {

    private final UserSessionRepository;
    // ...
}
```

Pragmatic Hexagonal/Clean Architecture



Primary adapters wrap around a use case and adapt its input/output to a delivery mechanism, ie. HTTP/HTML, HTTP/JSON or CLI.

Secondary adapters wrap around a tool and adapt its input/output to a port, which fits the application core needs and can simply be an interface or a complex set of interfaces and objects.

However:
- The map is not the territory.
- Plans are worthless, but planning is everything.
- Understand all of this, but use only what you need.
- The actual architecture is driven by the project requirements.

@pivovarit

Axon RabbitMQ/Kafka Story

```java
import org.axonframework.eventhandling.annotation.EventHandler;

public class SessionEventsHandler {

    //...

    @EventHandler
    public void handle(SessionStartedEvent event) {
        // ...
    }

    @EventHandler
    public void handle(SessionEndedEvent event) {
        // ...
    }
}
```

```java
@FunctionalInterface
public interface EventHandler {
    Map<String, EventRoute> getRoutingConfig();
}
```

```java
public class SessionEventsHandler implements EventHandler {

    // ...

    private void playerSessionStarted(SessionStartedEvent event) {
        // ...
    }

    private void playerSessionEnded(SessionEndedEvent event) {
        // ...
    }

    @Override
    public Map<String, EventRoute> getRoutingConfig() {
        return Map.ofEntries(
          route(
            SessionEndedEvent.ROUTING_KEY,
            SessionEndedEvent.class,
            this::playerSessionEnded),
          route(
            SessionStartedEvent.ROUTING_KEY,
            SessionStartedEvent.class,
            this::playerSessionStarted)
        );
    }
}
```

```java
public class RabbitEventStreamListener implements MessageListener {

    // ...

    private final Map<String, EventRoute> eventRoutes;

    @Override
    public void onMessage(Message message) {
        var route = eventRoutes.get(message.getMessageProperties().getReceivedRoutingKey());
        if (null != route) {
            route.getEventHandler().accept(new DomainMessage(...);
        } else {
            log.warn("couldn't find a matching routing for routing key");
        }
    }
}
```

```java
@Configuration
@Profile("kafka-eventstream")
public class KafkaEventStreamConfiguration {

    @Bean
    public KafkaEventStreamListener kafkaEventStreamListener(
      List<EventHandler> handlers, ObjectMapper mapper) {

        var eventRoutes = handlers.stream()
          .flatMap(h -> h.getRoutingConfig().entrySet().stream())
          .collect(
            groupingBy(Map.Entry::getKey,
              mapping(Map.Entry::getValue, toUnmodifiableSet())));

        return new KafkaEventStreamListener(mapper, eventRoutes);
    }
}
```

@pivovarit

```java
@Configuration
@Profile("rabbit-eventstream")
public class RabbitEventStreamConfiguration {

    @Bean
    public Collection<MessageListenerContainer> rabbitEventStreamListener(
      @Value("${rabbitmq.axon.exchange}") String axonExchangeName,
      @Value("${rabbitmq.axon.queuePrefix}") String axonQueuePrefix,
      AmqpAdmin amqpAdmin,
      RabbitListenerContainerFactory eventstreamRabbitListenerContainerFactory,
      List<EventHandler> handlers,
      ConfigurableBeanFactory beanFactory,
      ObjectMapper mapper) {

        TopicExchange axonExchange = new TopicExchange(axonExchangeName);
        amqpAdmin.declareExchange(axonExchange);

        return handlers.stream()
          .map(handler -> buildRabbitListener(...))
          .collect(toUnmodifiableList());
    }

    private static MessageListenerContainer buildRabbitListener(
      EventHandler handler,
      ConfigurableBeanFactory beanFactory,
      ObjectMapper mapper,
      String queuePrefix,
      AmqpAdmin amqpAdmin,
      TopicExchange axonExchange,
      RabbitListenerContainerFactory eventstreamRabbitListenerContainerFactory) { ... }
}
```

@pivovarit

Things to try

Unopinionated: Ratpack

Ratpack is a set of Java libraries for building scalable HTTP applications.

It is a lean and powerful foundation, not an all-encompassing framework.

https://ratpack.io https://www.youtube.com/watch?v=rqCHb9M3uiI

@pivovarit

```java
public class MyApp {

    public static void main(String[] args) throws Exception {
      RatpackServer.start(s -> s
          .serverConfig(c -> c.baseDir(BaseDir.find()))
          .registry(Guice.registry(b -> b.module(MyModule.class)))
          .handlers(chain -> chain
              .path("foo", ctx -> ctx.render("from the foo handler"))
              .path("bar", ctx -> ctx.render("from the bar handler"))
              .prefix("nested", nested -> {
                nested.path(":var1/:var2?", ctx -> {
                  Map<String, String> pathTokens = ctx.getPathTokens();
                  ctx.render(
                    "from the nested handler, var1: " + pathTokens.get("var1") +
                      ", var2: " + pathTokens.get("var2")
                  );
                });
              })
              .path("injected", MyHandler.class)
              .prefix("static", nested -> nested.fileSystem("assets/images", Chain::files))
              .all(ctx -> ctx.render("root handler!"))
          )
      );
    }
  }
```

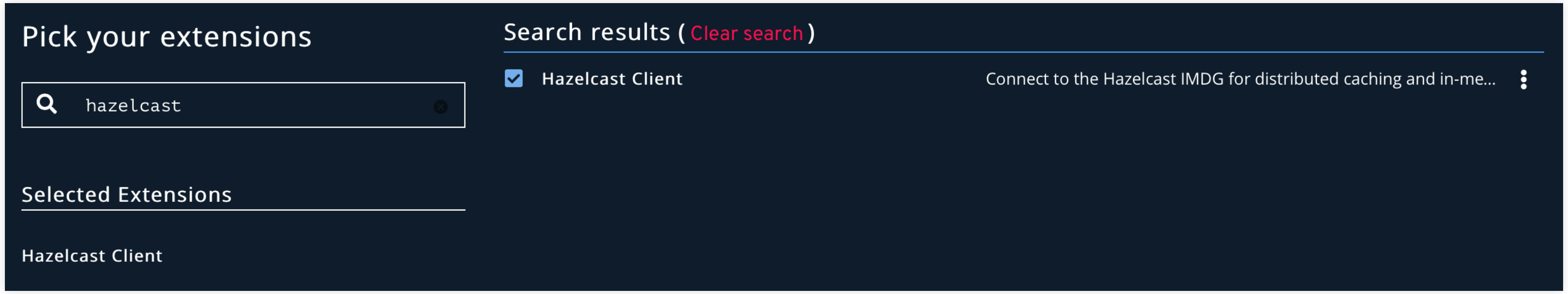
@pivovarit

Opinionated: Quarkus

```
$ ./my-native-java-rest-app
Quarkus started in 0.008s
```

Memory (RSS) in Megabytes*
*Tested on a single-core machine
REST
REST
+ CRUD

@pivovarit



Pick your extensions

hazelcast

Search results (Clear search)

☑ Hazelcast Client Connect to the Hazelcast IMDG for distributed caching and in-me...

Selected Extensions

Hazelcast Client

@pivovarit

Using common sense is the ultimate Best Practice™.

http://tonsky.me/blog/disenchantment/

Thank You!

https://pivovarit.github.io/talks/fantastic-frameworks

@pivovarit

hazelcast®

Q&A

@pivovarit