Whoops! Where did my architecture go?

Approaches to architecture management for Java and Spring applications

Oliver Gierke

Oliver Gierke

SpringSource Engineer

Spring Data

ogierke@vmware.com

olivergierke

www.olivergierke.de

Background

5 years of consulting

Lots of code reviews

Eoin Woods‘ talk on InfoQ

Roadmap

Architecture 101

A plain Java based approach

Hera

The 3 C′ of Architecture

Constraints

Complexity

Change

The 2 C's of Architecture

Complexity

Change

Architecture 101

Know your dependencies

Granularity

Modules

Layers

Vertical slices

Subsystems

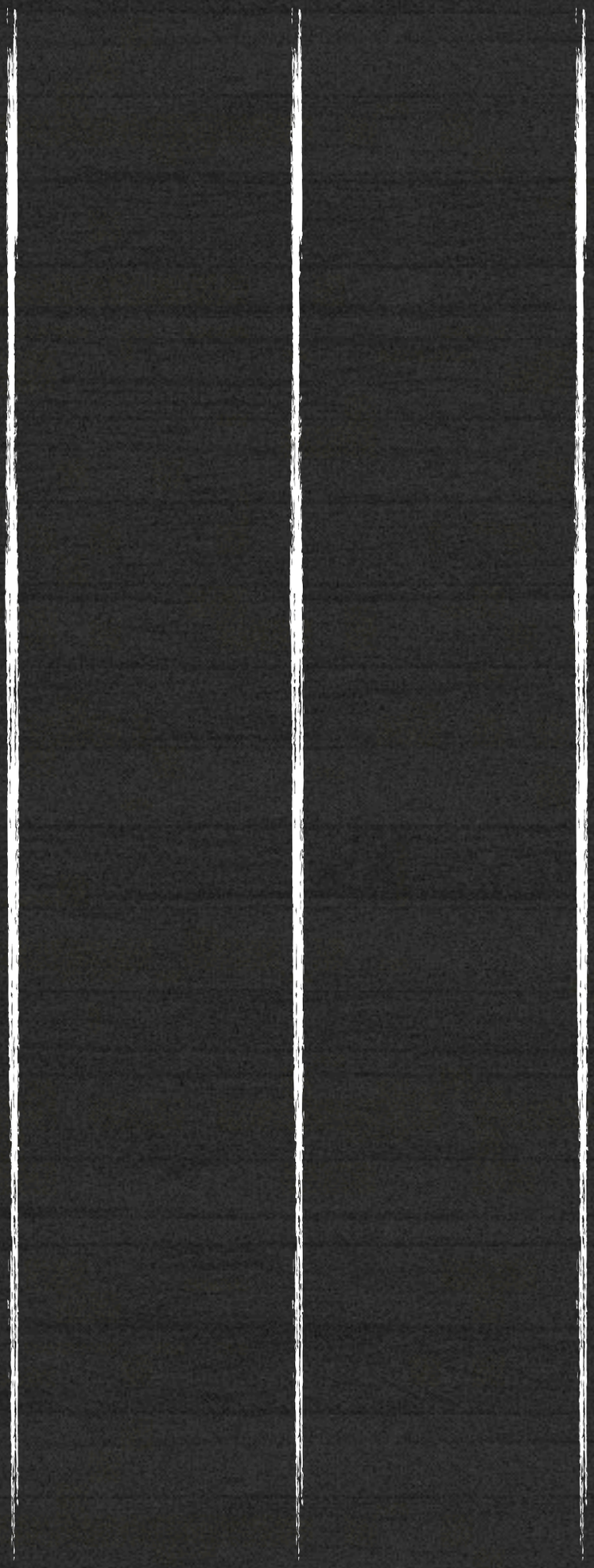
Granularity

Java ARchive

Package

Class

Of layers and slices...



Presentation

Service

Data Access

Presentation

Service

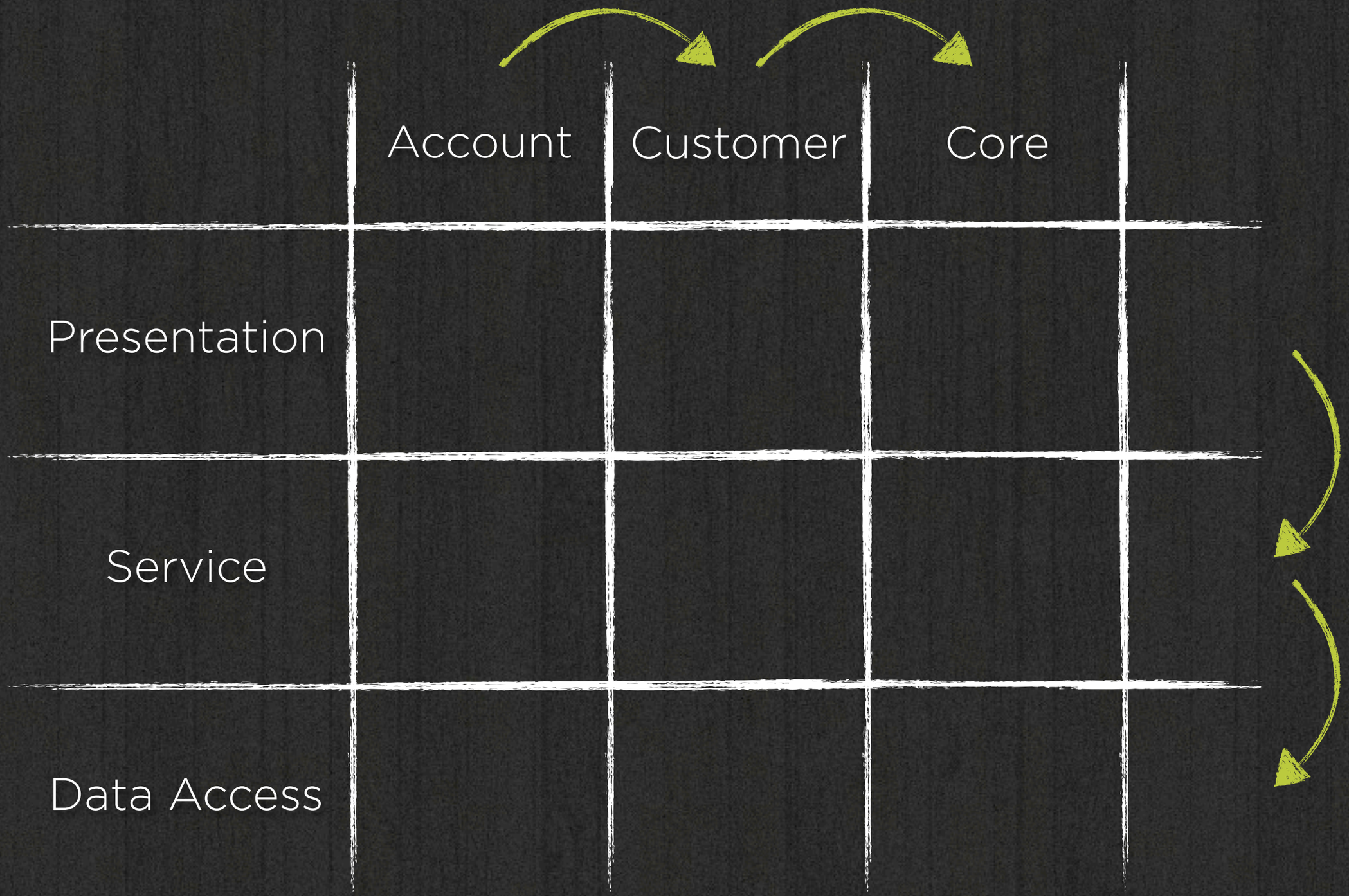
Data Access

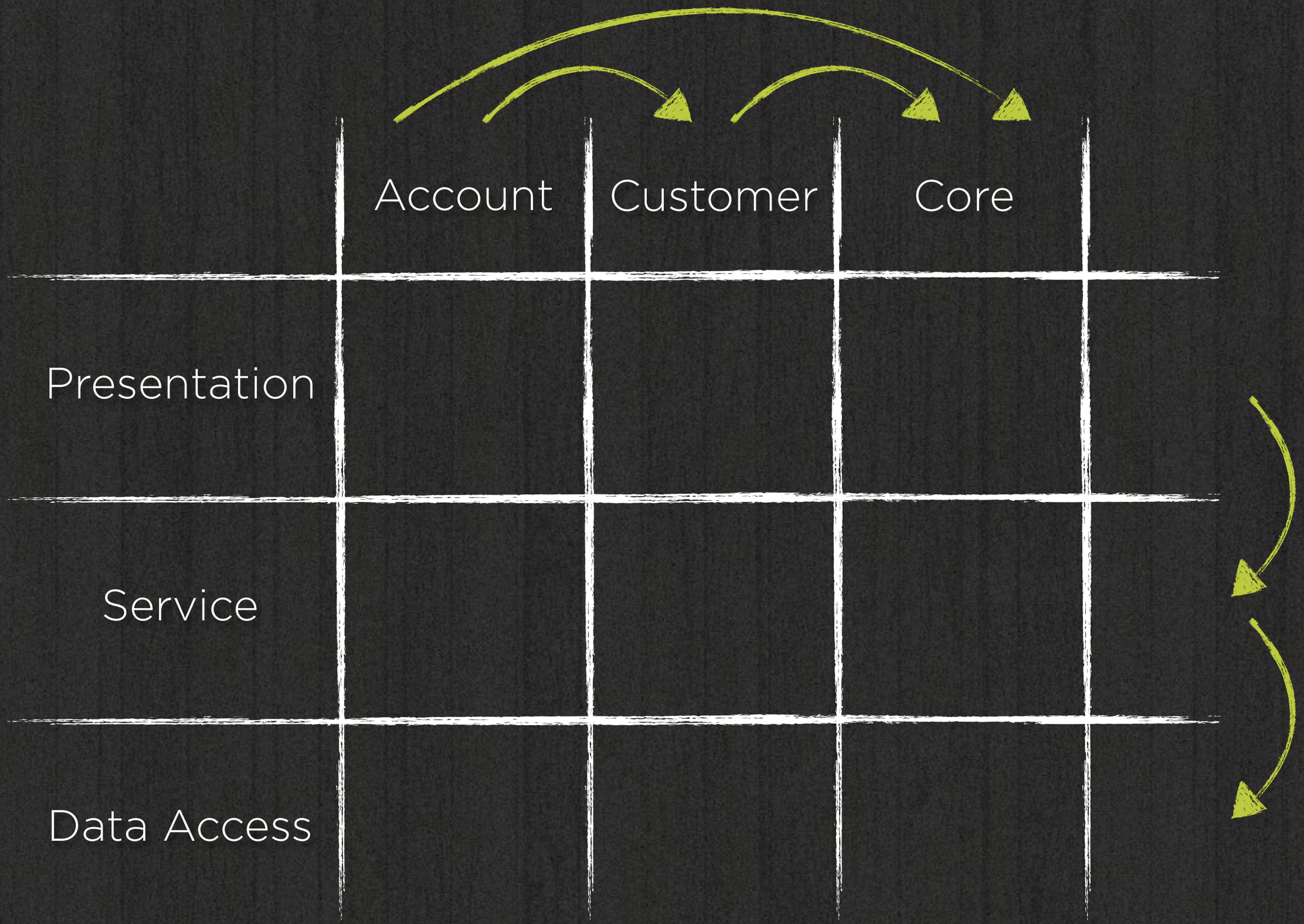
Presentation

Service

Data Access

	Account	Customer	Core
Presentation			
Service			
Data Access			


Account
Customer
Core
Presentation
Service
Data Access


Account
Customer
Core
Presentation
Service
Data Access

Layers

Well understood

Known to developers

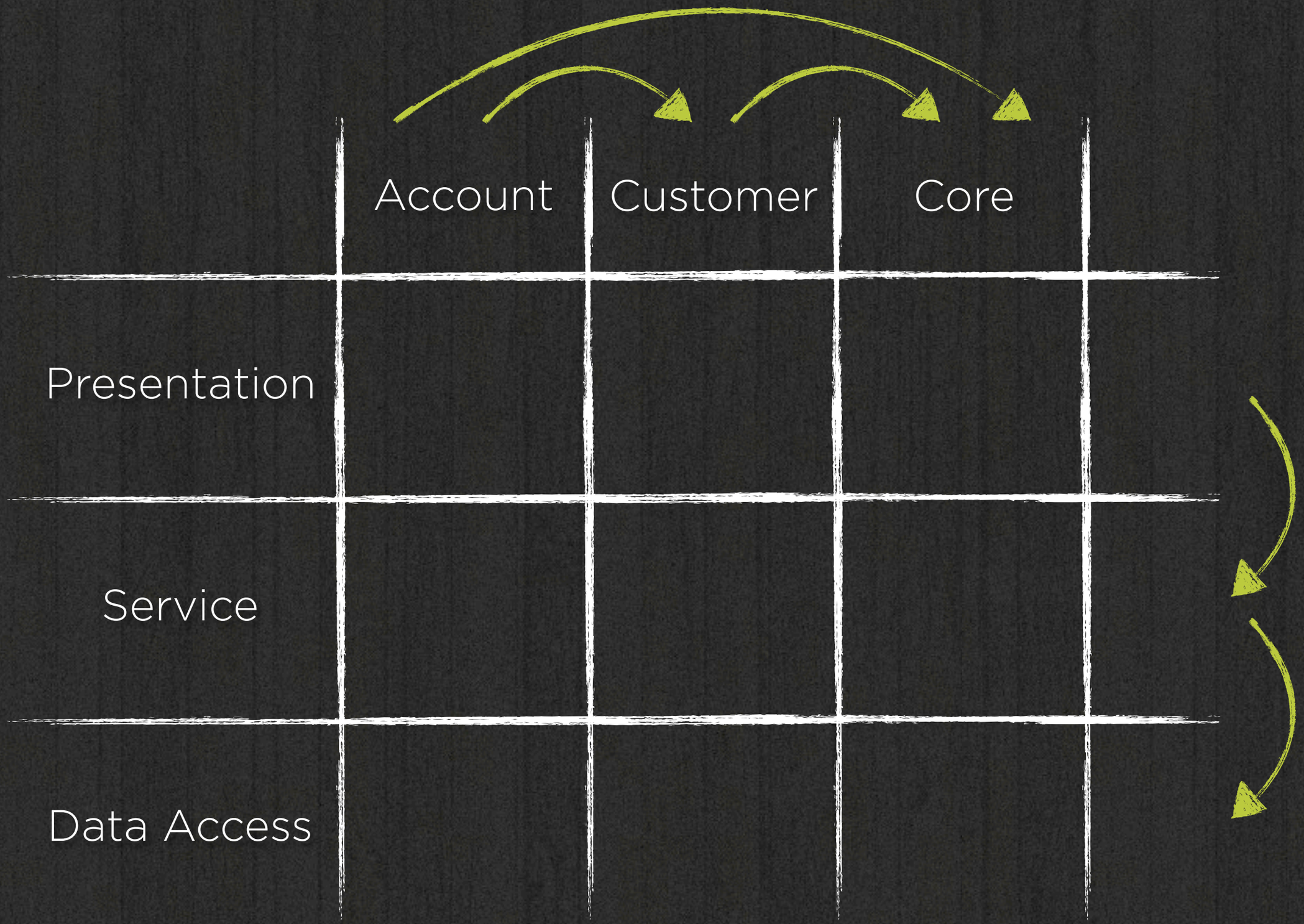
Less important to business

Slices

Hardly understood

New to developers

Key for business req


Account
Customer
Core
Presentation
Service
Data Access

"How to implement an architecture inside a codebase?

Architecture vs. Codebase

"How to implement an architecture inside a codebase?

"How to implement an architecture inside a codebase?

"How to enforce an architecture inside a codebase?

Code analysis

JDepend

Sonar

Demo

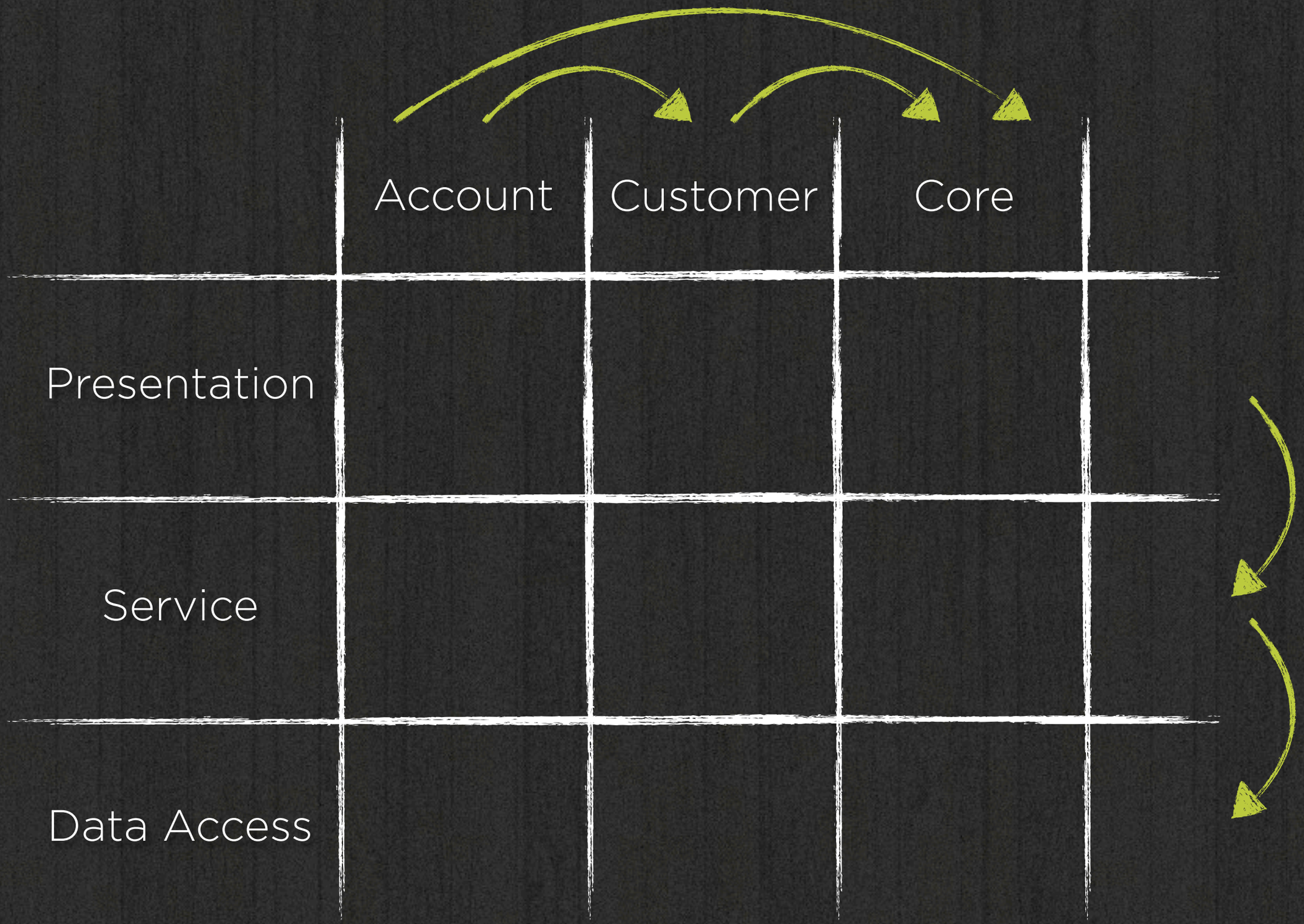
Sonargraph

Formerly known as SonarJ

Demo

A plain Java based approach

"How far can we get with **plain Java means** only?


Account
Customer
Core
Presentation
Service
Data Access

Packages

.... layer.slice ?

.... slice.layer ?

... slice ?

...domain.core

...service.core

...repository.core

...core.domain

...core.service

...core.repository

... core

... customer

... account

Layers first

Leaks slice internals

Lower layers visible to everyone

Slices first/only

Start with package per slice

Expose interfaces and domain types

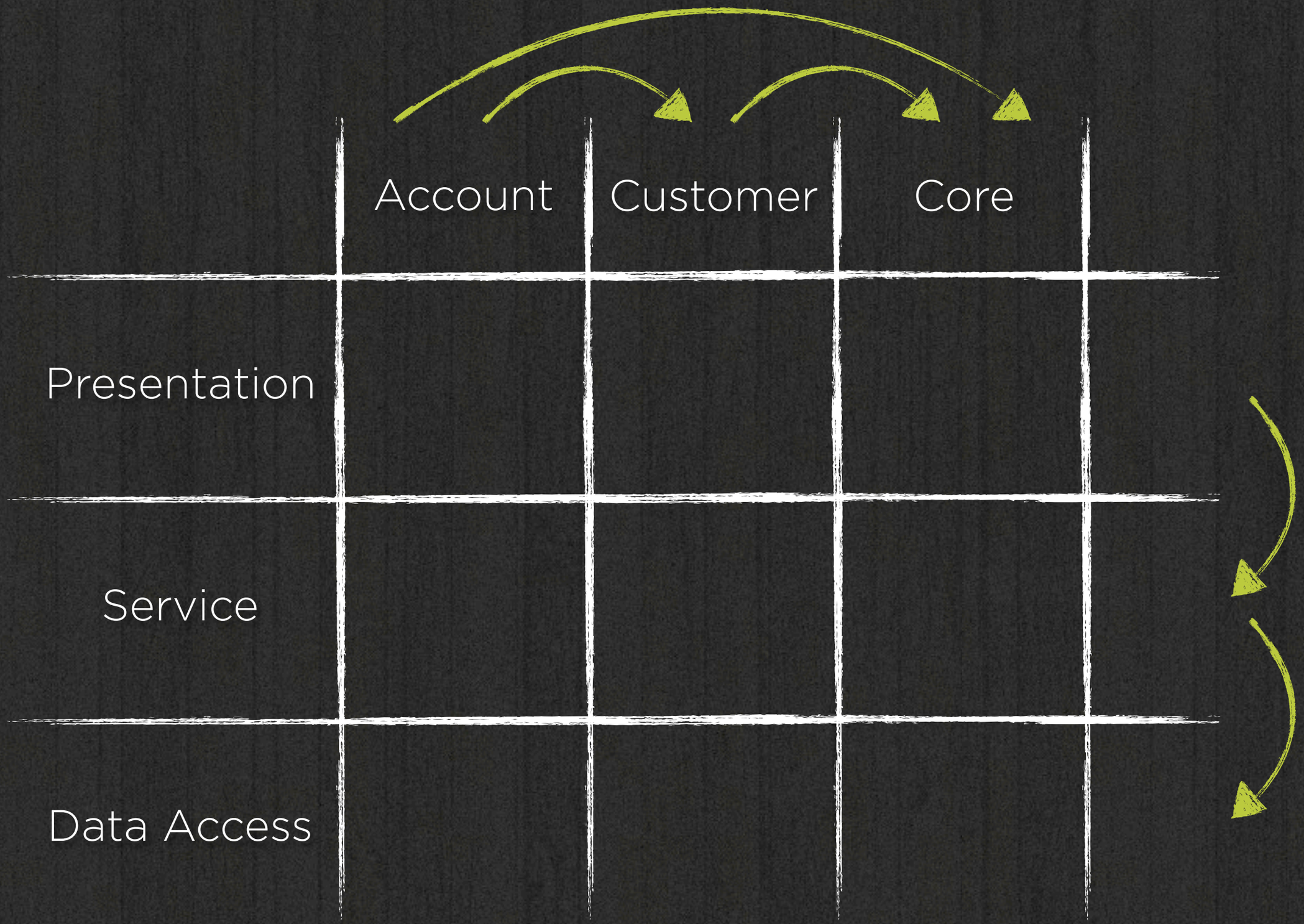
Keep implementations private

Slices first/only

Encapsulates business module

Internals understood anyway

"Start with **less** packages and the **least visibility** possible...


Account
Customer
Core
Presentation
Service
Data Access

Account

Customer

Core

Presentation

Service

Data Access

Subsystems

Background

Risk management at German public bank

Quite a few other SpringSource clients

Systems

grow big!

Remember the 2 C's?

Complexity & Change

" How to **localize change**?

"Separate frequent from infrequent change!

Granularity

Class

Package

Java ARchive

Host


Host
SPI
SPI
SPI

Host

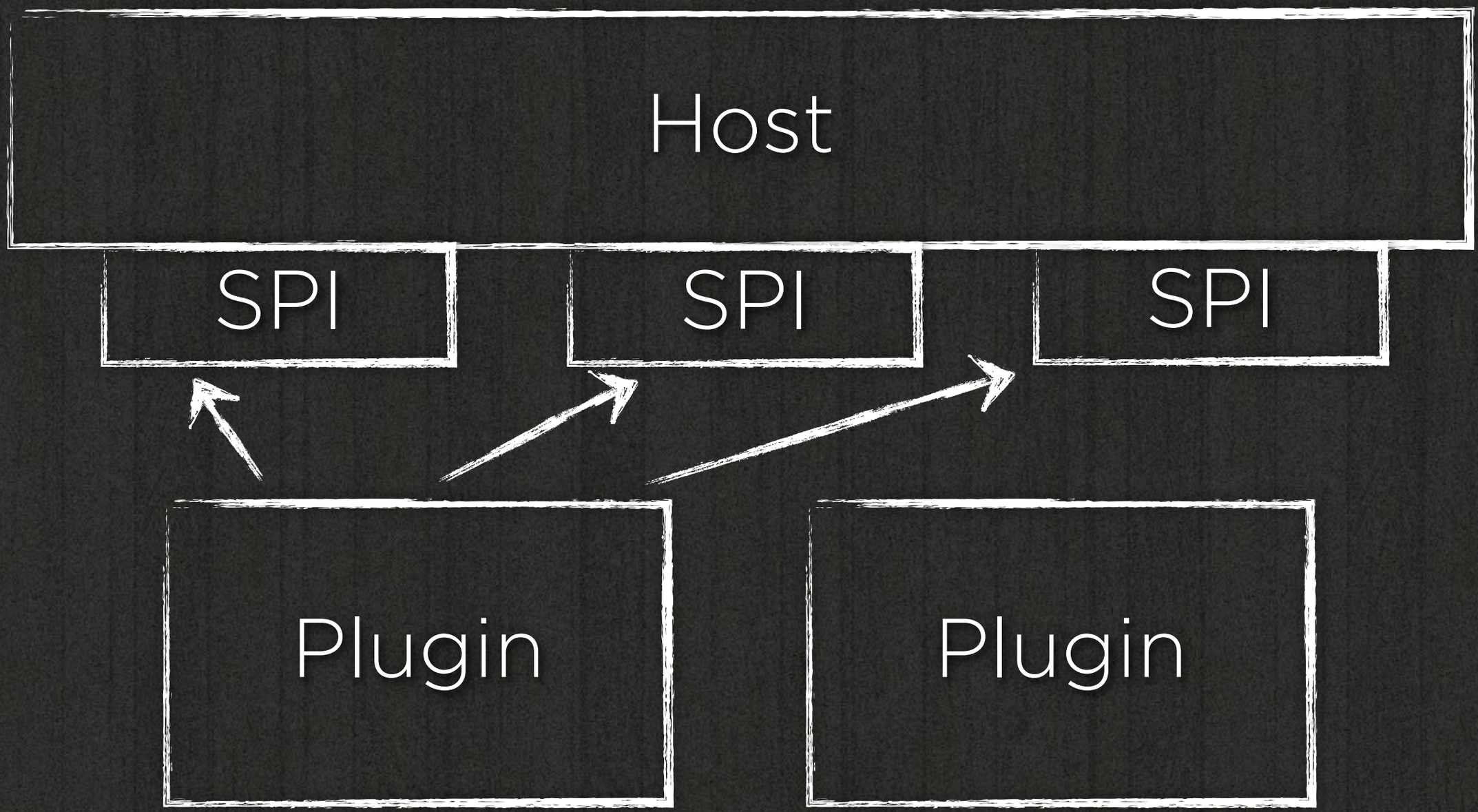
SPI

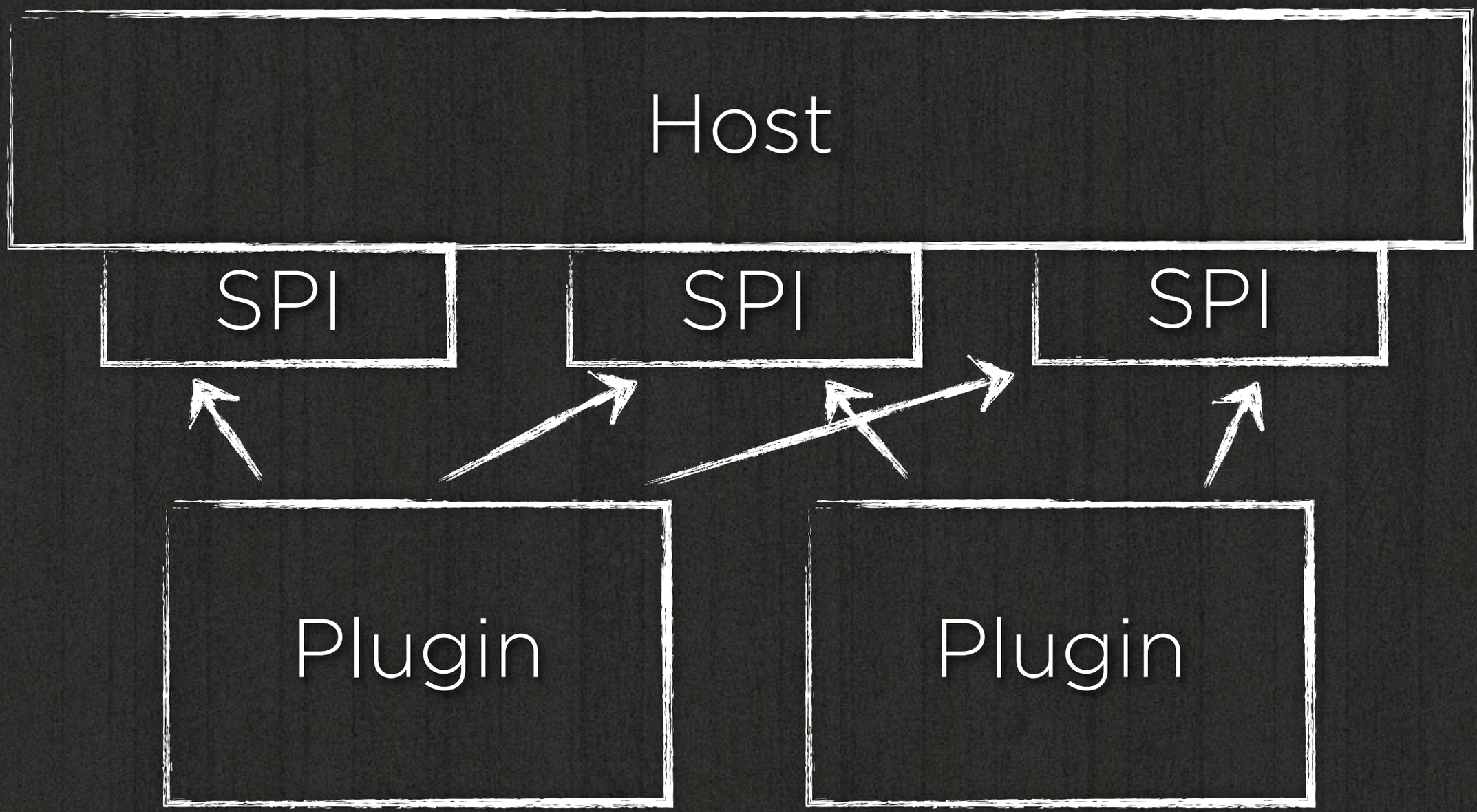
SPI

SPI

Plugin

Plugin


Host
SPI
SPI
SPI
Plugin
Plugin


Host
SPI
SPI
SPI
Plugin
Plugin

Context

No OSGi

Spring based

Build-time composition

Don‘t touch the host system

Host

Plugin

Plugin

Host

App

Plugin

Plugin

Host

"How to make the host aware of the plugins?

"How to dynamically collect Spring beans of a given type?

classpath*:META-INF/
spring/plugin-context.xml

Host


Host
SPI
SPI
SPI

Host

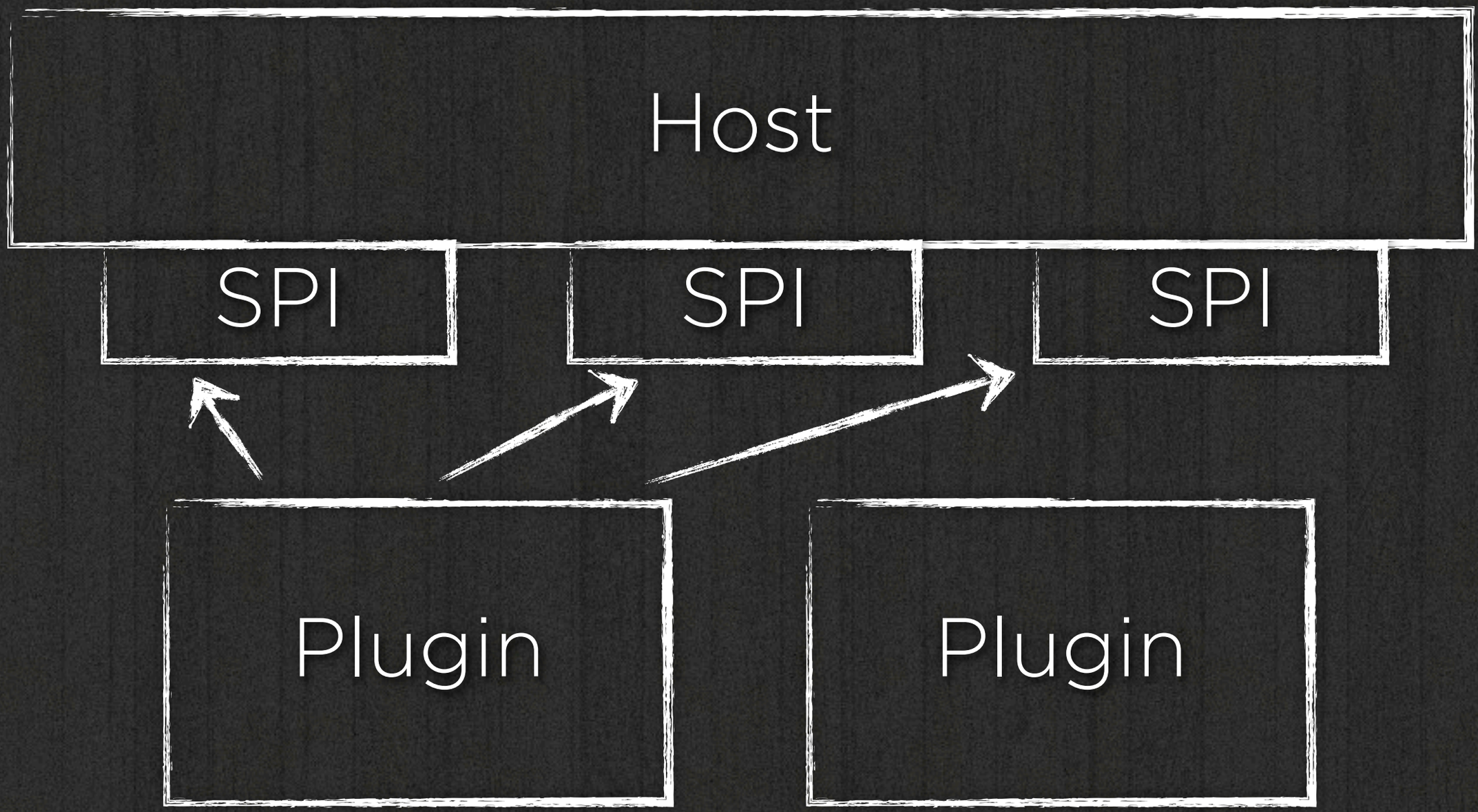
SPI

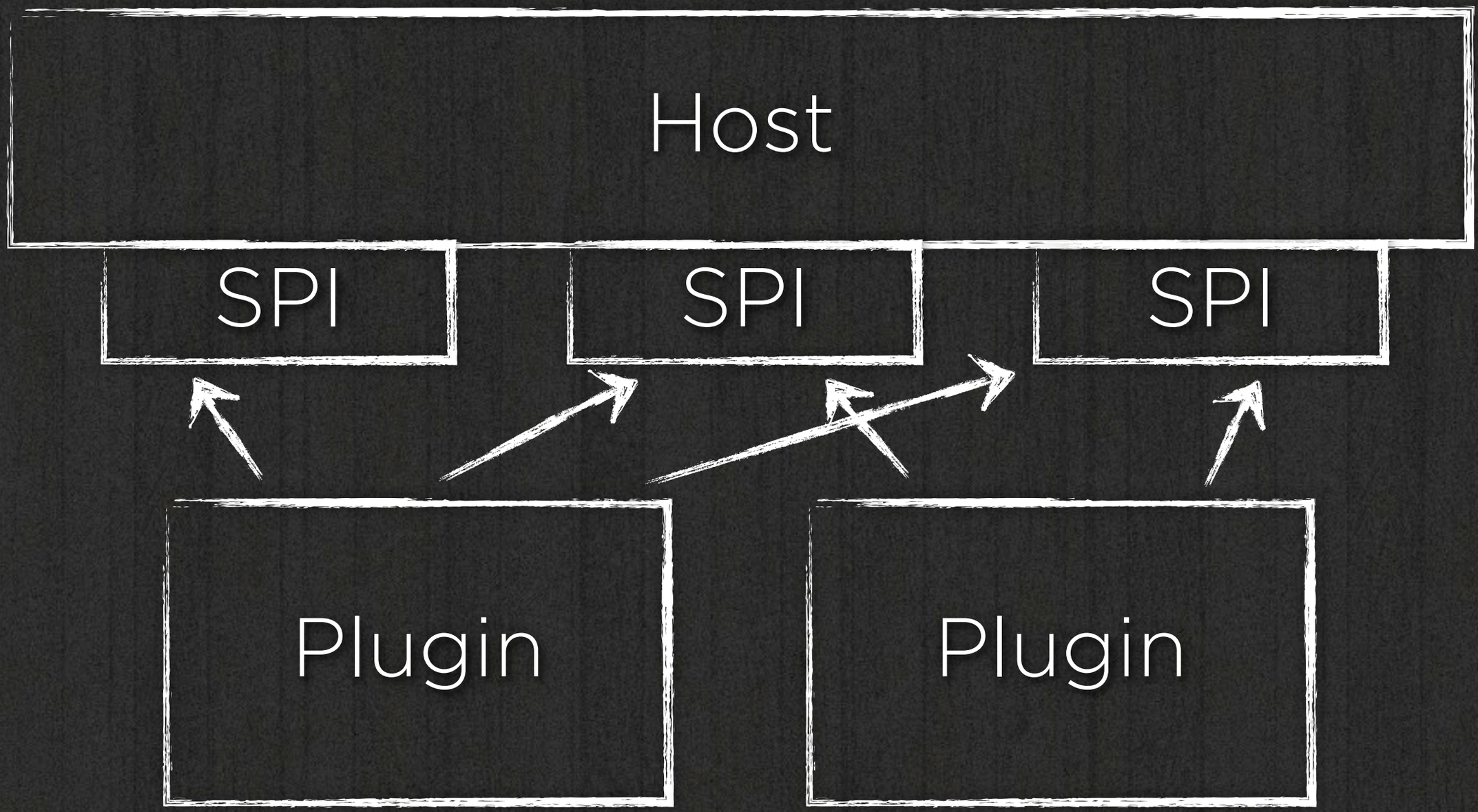
SPI

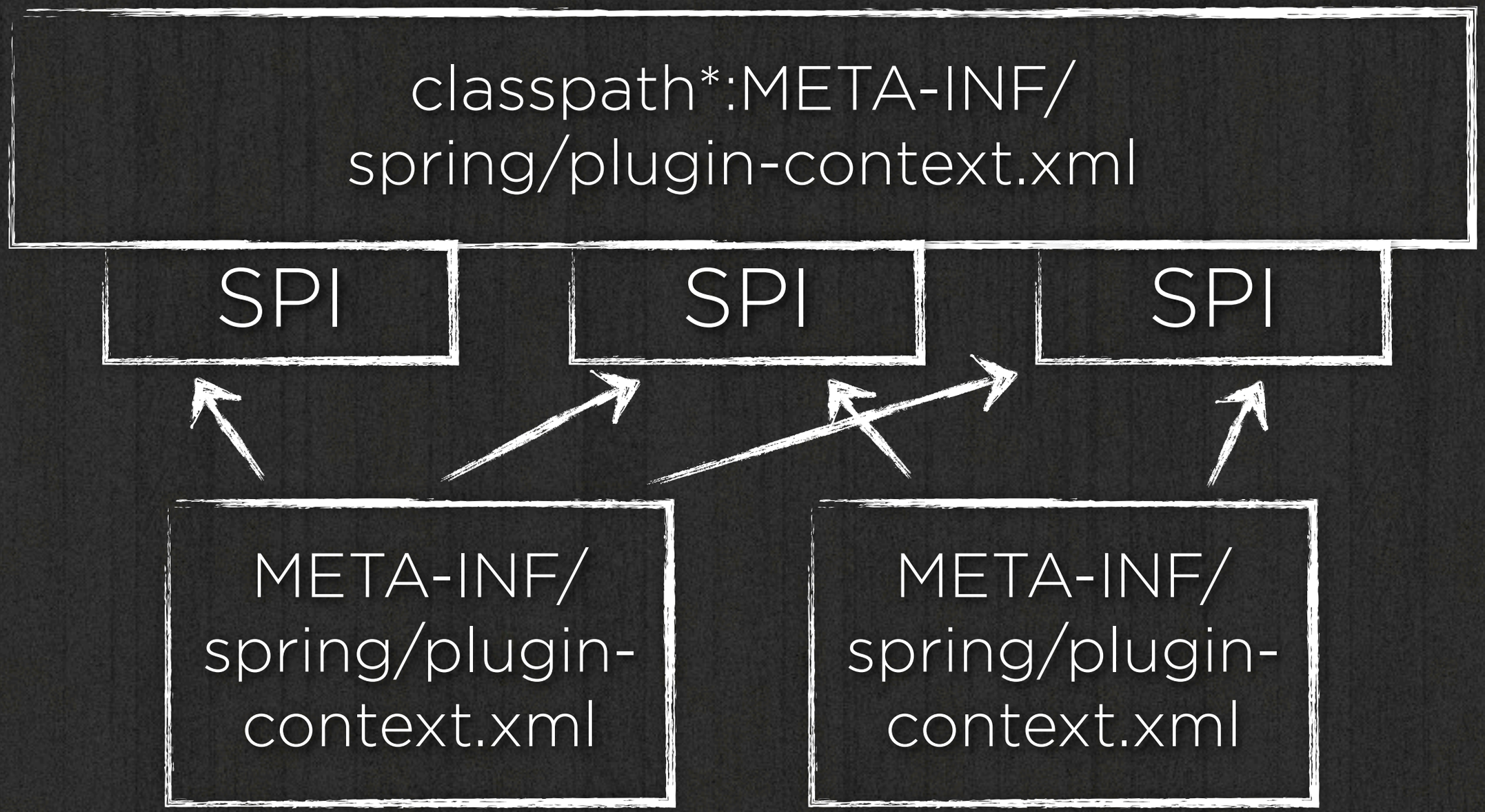
SPI

Plugin

Plugin


Host
SPI
SPI
SPI
Plugin
Plugin


Host
SPI
SPI
SPI
Plugin
Plugin


classpath*:META-INF/
spring/plugin-context.xml
SPI
SPI
SPI
META-INF/
spring/plugin-
context.xml
META-INF/
spring/plugin-
context.xml

```java
@Component
public class MyComponentImpl implements TransferService {

    private List<MyPlugin> plugins;

    @Autowired
    public MyComponentImpl(List<MyPlugin> plugins) {
        this.plugins = plugins;
    }
    ...
}

public interface MyPlugin {
    void doSomething();
}
```

Demo

XML?

Back in the days

(XML?)

Back in the days

Probably not a big deal anymore

Easy access?

```java
@Component
public class MyComponentImpl implements TransferService {

    private List<MyPlugin> plugins;

    @Autowired
    public MyComponentImpl(List<MyPlugin> plugins) {
        this.plugins = plugins;
    }
    ...
}

public interface MyPlugin {
    void doSomething();
}
```

```
@Component
public class MyComponentImpl implements TransferService {

    // Constructor omitted

    public Result myMethod(SomeParameter parameter) {

        // Select the first one to match to invoke?
        // Select multiple ones to invoke?
        // Select and fallback to one if none found?
        // Select and throw an exception if none found?
    }
}
```

Spring Plugin

"The smallest plugin system ever!

Plugins

Selection criterion

Callback method

Let the implementation decide

Registry

Equipped with plugins

Common access patterns

```
public interface Plugin<T> {

    public boolean supports(T delimiter );
}


public interface PluginRegistry<S extends Plugin<T>, T> {

    T getPluginFor(S delimiter);
    T getPluginFor(S delimiter, T default);
    <E extends Exception> T getPluginFor(S del, E e) throws E;

    List<T> getPluginsFor(S delimiter);
    ...
}
```

```java
@Component
public class MyComponentImpl implements TransferService {

    private PluginRegistry<MyPlugin, String> plugins;

    @Autowired
    public MyComponentImpl(
        PluginRegistry<MyPlugin, String> plugins) {
        this.plugins = plugins;
    }
}

public interface MyPlugin extends Plugin<String> {
    void doSomething();
}
```

```
@Component
public class MyComponentImpl implements TransferService {
    private final MyPlugin defaultPlugin = new
        MyDefaultPlugin();
    public Result myMethod(String parameter) {
        // Select the first one to match to invoke?
        ... = plugins.getPluginFor(parameter);
        // Select multiple ones to invoke?
        ... = plugins.getPluginsFor(parameter);
        // Select and fallback to one if none found?
        ... = plugins.getPluginFor(parameter, defaultPlugin);
        // Select and throw an exception if none found?
        ... = plugins.getPluginsFor(parameter, new
            RuntimeException());
    }
}
```

OrderAware PluginRegistry

Respects @Order/Ordered

OAPR.reverse()

Bells‘n‘whistles

FactoryBean

Spring namespace

Lazy-eval

Spring Plugin

github.com/SpringSource/spring-plugin

Apache 2.0

Spring Integration²

```
<bean class="....FirstSamplePlugin" />
<bean class="....SecondSamplePlugin" />

<int:channel id="input" />
<int:channel id="output" />

<int-plugin:dynamic-service-activator
    input-channel="input"
    outputChannel="output"
    plugin-type= "....MyPlugin"
    method= "myBusinessMethod"
    delimiter="payload"
    invocation-arguments= "payload" />
```

Demo

Take-aways

Know your dependencies

On every granularity

Start as strict as possible

Get lenient where necessary

Thanks & credits

Eoin Woods - Talk @ InfoQ

Uwe Friedrichsen - Slides @ Slideshare

Resources

Spring Data JPA @ GitHub

Sonargraph

Spring Plugin