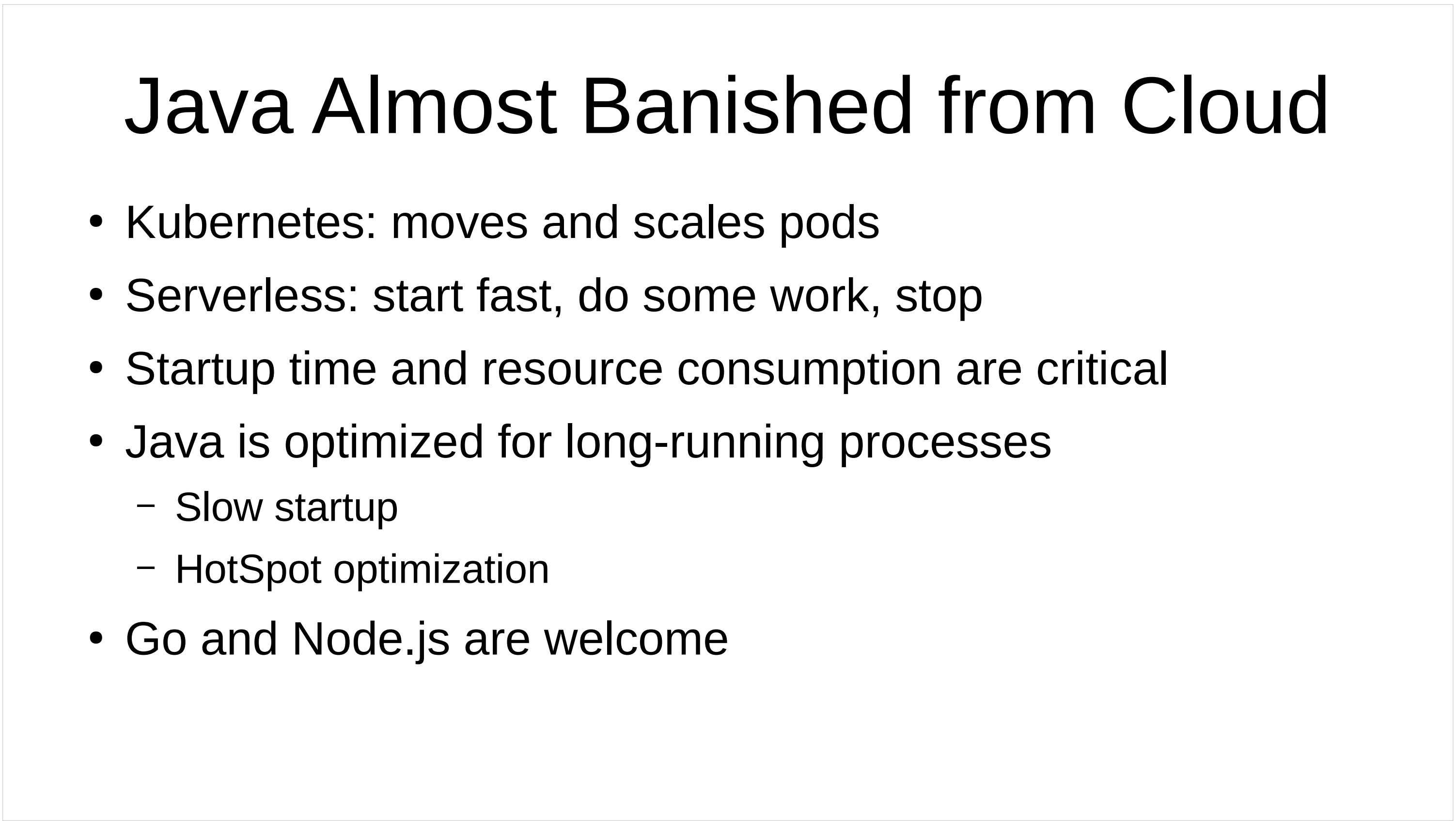
From Thorn...
....tail to Quarkus

Željko Trogrlić



Java Almost Banished from Cloud

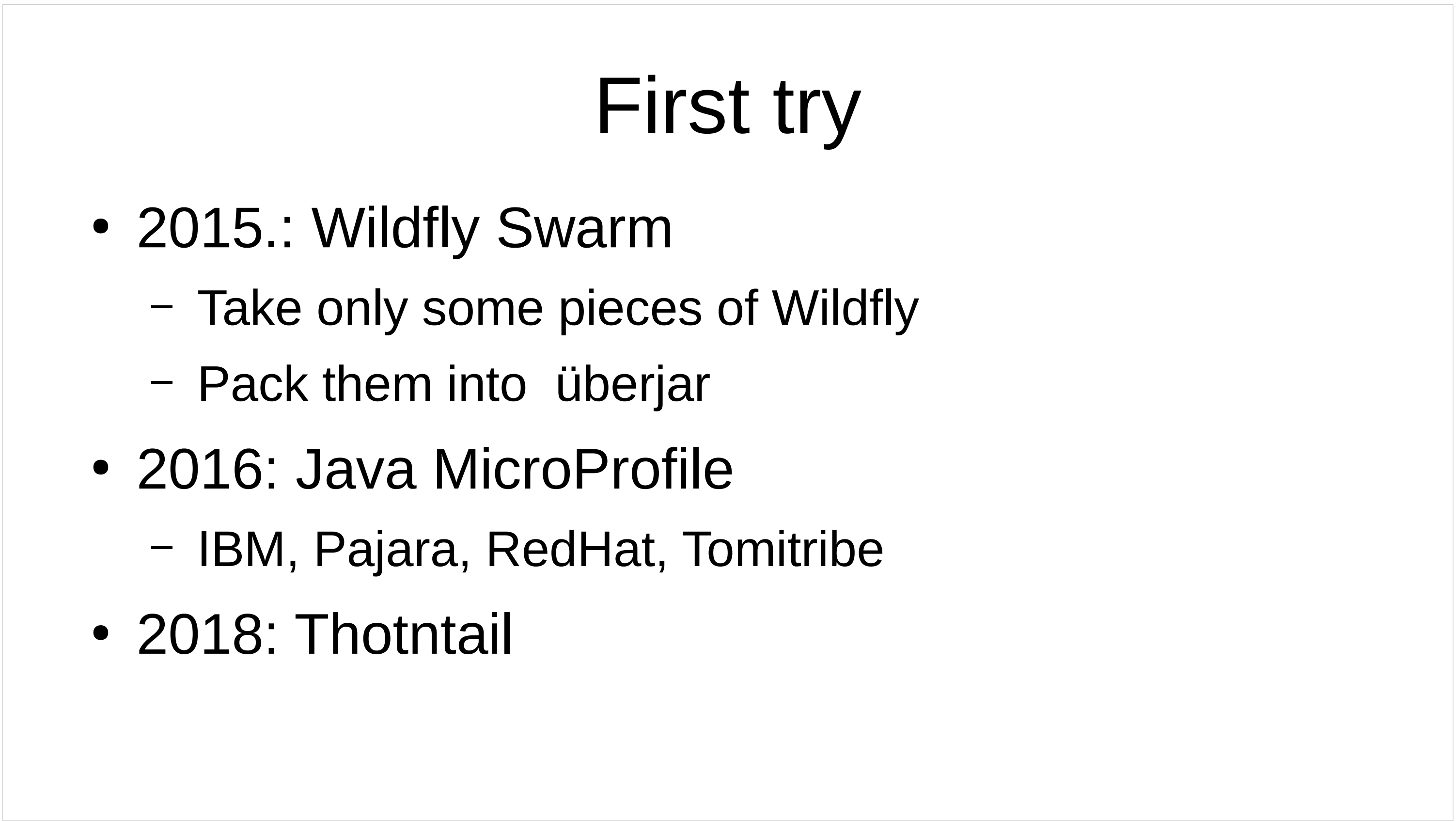
- Kubernetes: moves and scales pods
- Serverless: start fast, do some work, stop
- Startup time and resource consumption are critical
- Java is optimized for long-running processes
  - Slow startup
  - HotSpot optimization
- Go and Node.js are welcome

Serverless = FaaS

There is an overlap between application server and cloud framework (application management, high availability, load balancing); it makes Java app servers obsolete.

Cloud is heterogeneous and simpler to manage.

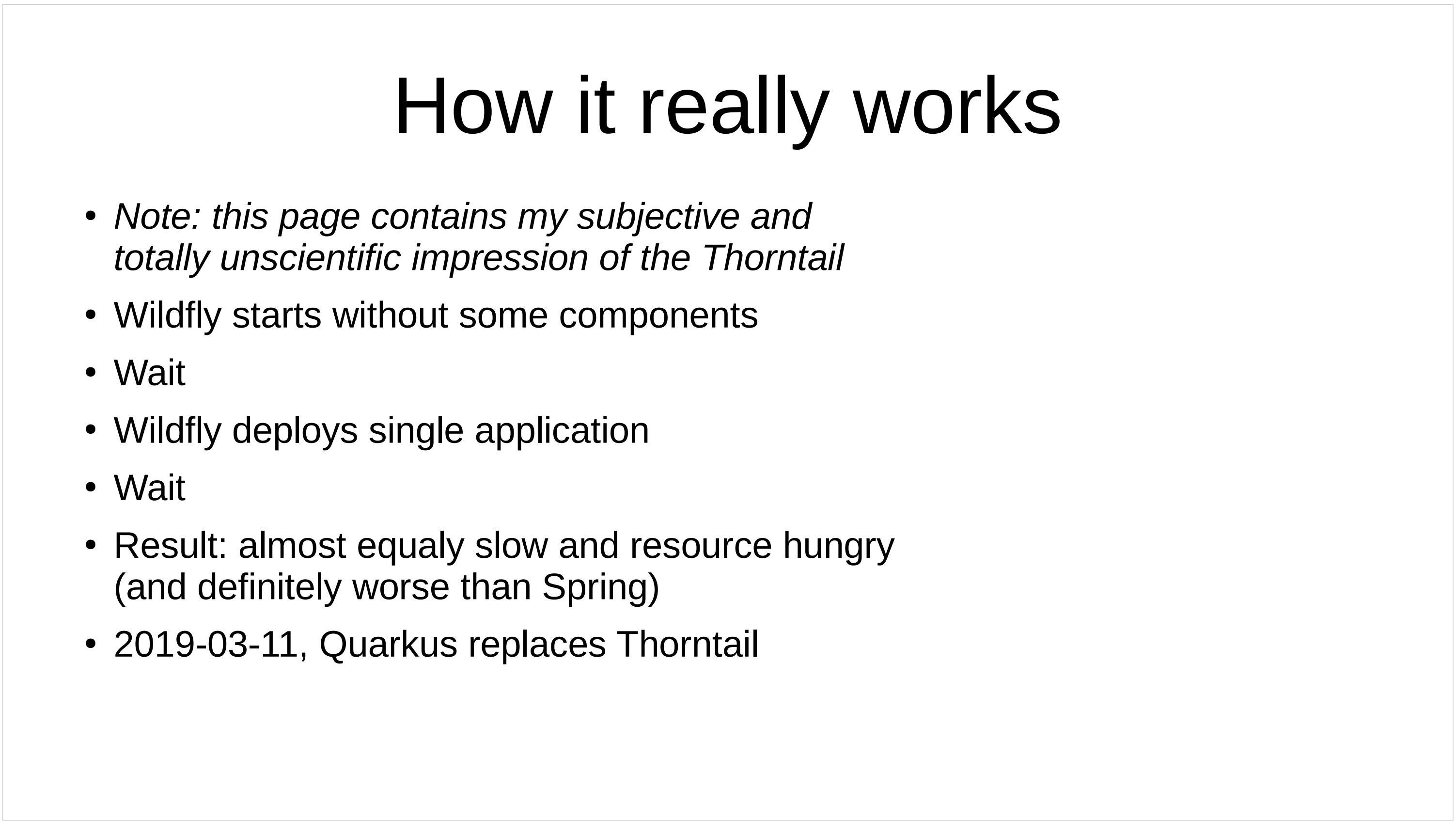
Java greatest advantages became weaknesses.



First try

- 2015.: Wildfly Swarm
  - Take only some pieces of Wildfly
  - Pack them into überjar
- 2016: Java MicroProfile
  - IBM, Pajara, RedHat, Tomitribe
- 2018: Thotntail

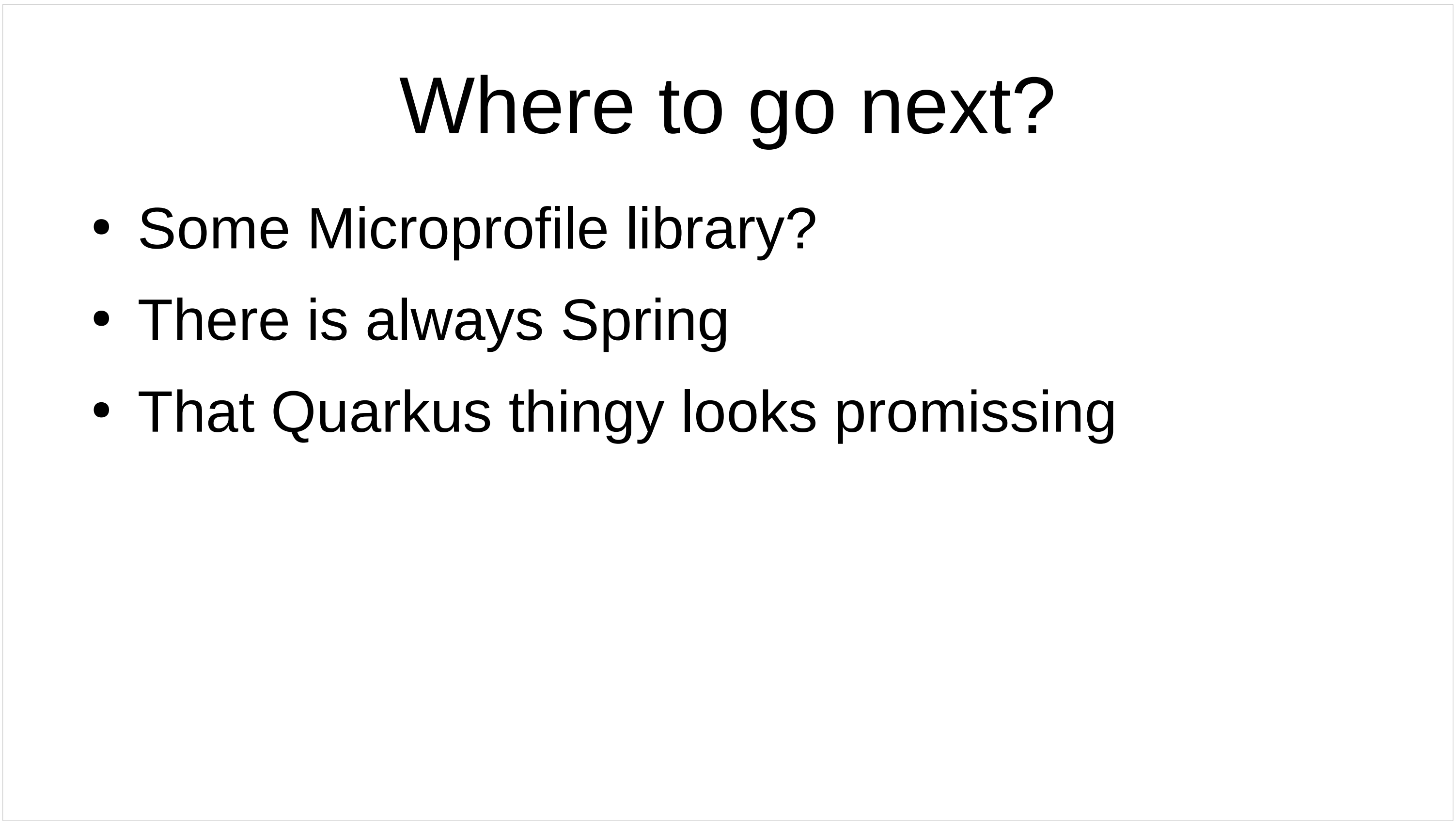
Attempts to make Java cloud-friendly started half decade ago.



How it really works

- *Note: this page contains my subjective and totally unscientific impression of the Thorntail*
- Wildfly starts without some components
- Wait
- Wildfly deploys single application
- Wait
- Result: almost equaly slow and resource hungry (and definitely worse than Spring)
- 2019-03-11, Quarkus replaces Thorntail

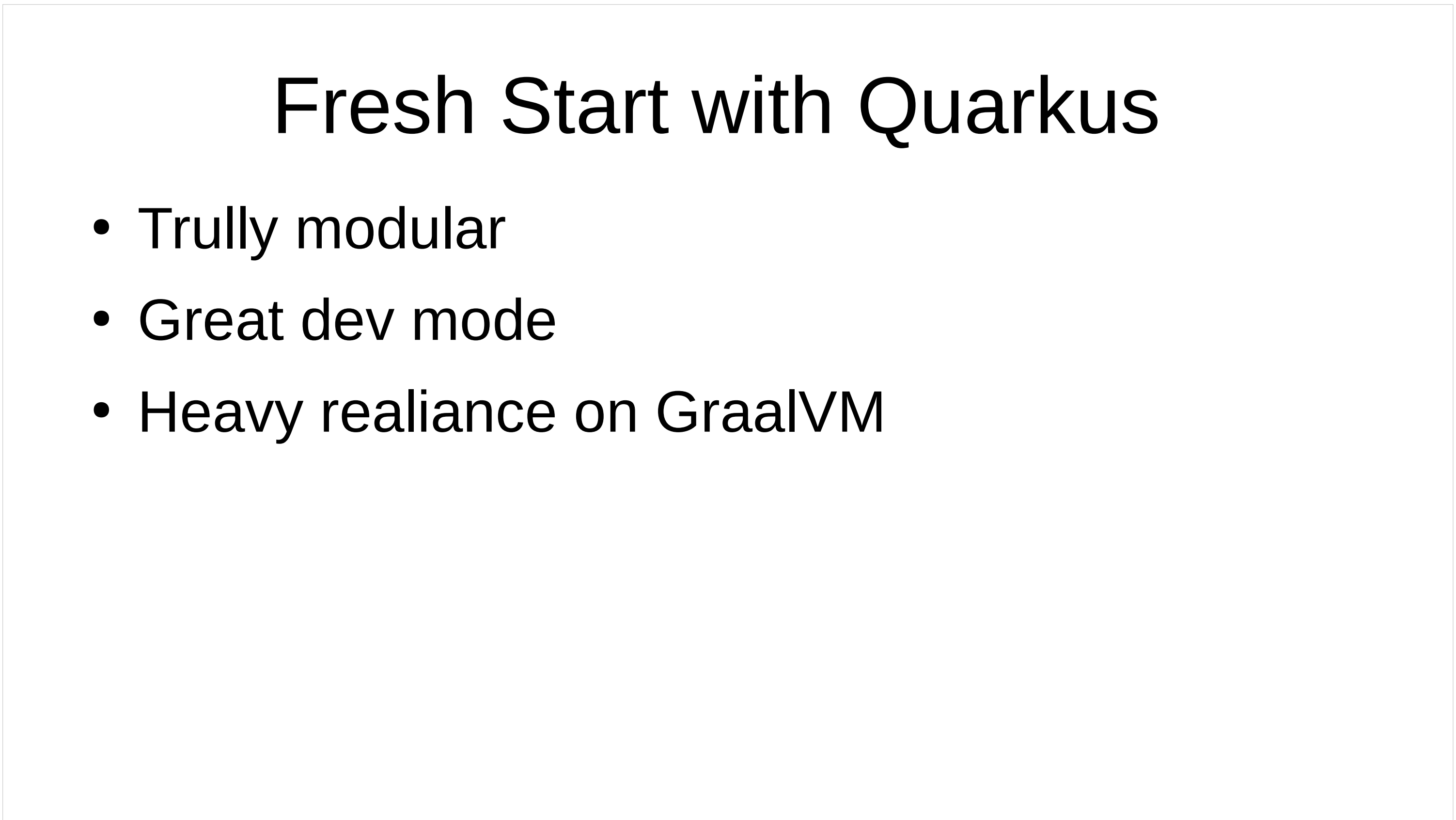
First Wildfly-based attempts did not bring enough to the table regarding speed and reduced resource usage. It was repackaging of existing product to buy some time.



Where to go next?

- Some Microprofile library?
- There is always Spring
- That Quarkus thingy looks promissing

So the question was what else to use, especially if projects already used Thorntail.

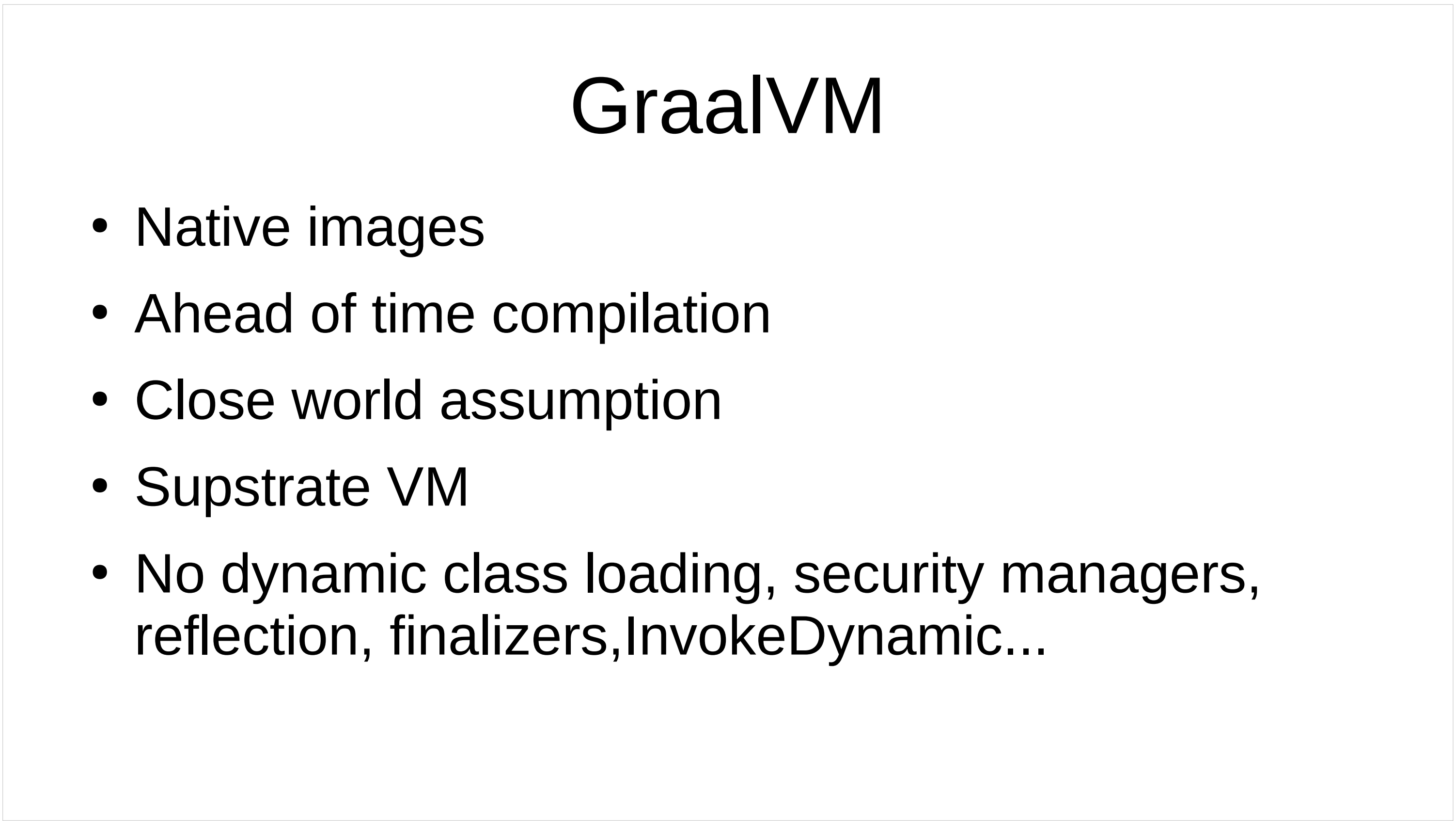


Fresh Start with Quarkus

- Trully modular
- Great dev mode
- Heavy realiance on GraalVM

Quarkus was a completely different take on cloud Java.

It is based on SmallRye Microprofile components with have reliance on GraalVM as a solution to cloud problem.



GraalVM

- Native images
- Ahead of time compilation
- Close world assumption
- Supstrate VM
- No dynamic class loading, security managers, reflection, finalizers,InvokeDynamic...

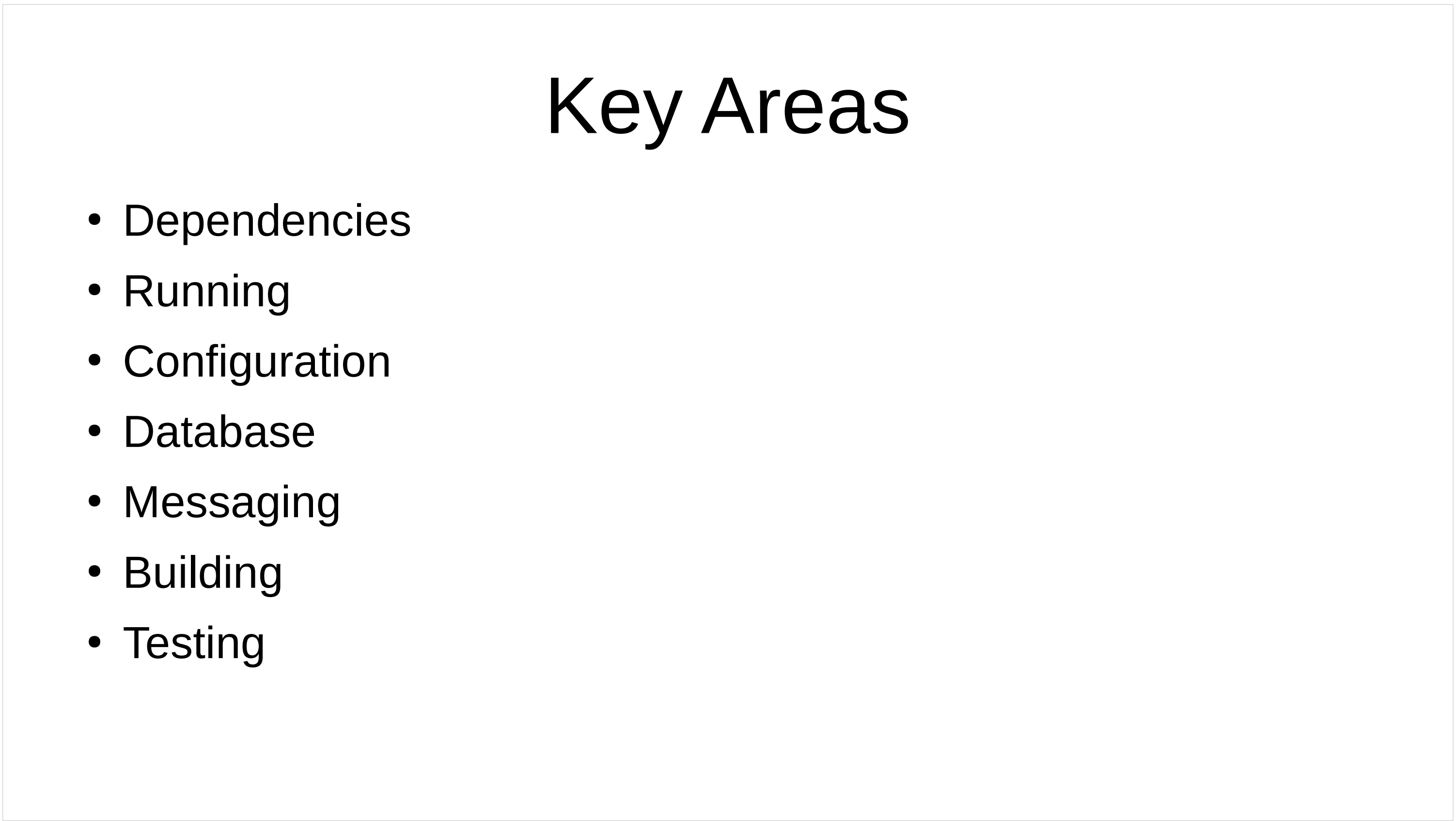
I emphasize GraalVM here as it heavy impact on Quarkus architecture. Quarkus is MicroProfile implementation for the GraalVM.

GraalVM is Java tool-set from Oracle. Main components are GraalVM compiler and GraalVM Native Image.

Compiler works with Hotspot while Native Image can create native images that run with the help of SupstrateVM. Generated code is heavily optimized and stripped of all unused classes. Static code results are already baked into image.

What is lost is dinamicity.ynamicity.

Graal giveth and Graal taketh away.

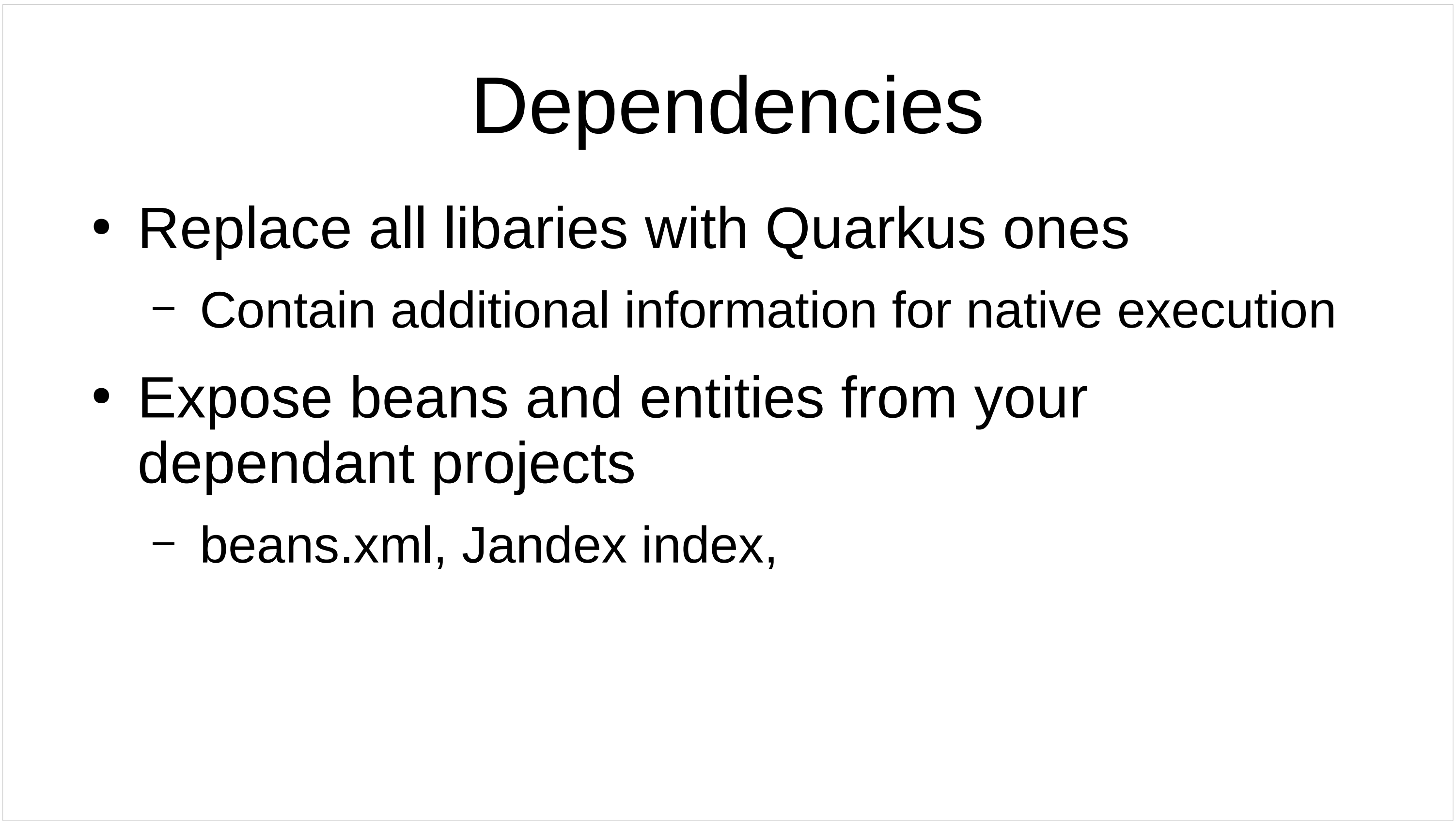


Key Areas

- Dependencies
- Running
- Configuration
- Database
- Messaging
- Building
- Testing

I worked on conversion of a real project for a customer from Thorntail to Quarkus.

In the next pages I will describe typical problems that I encountered.



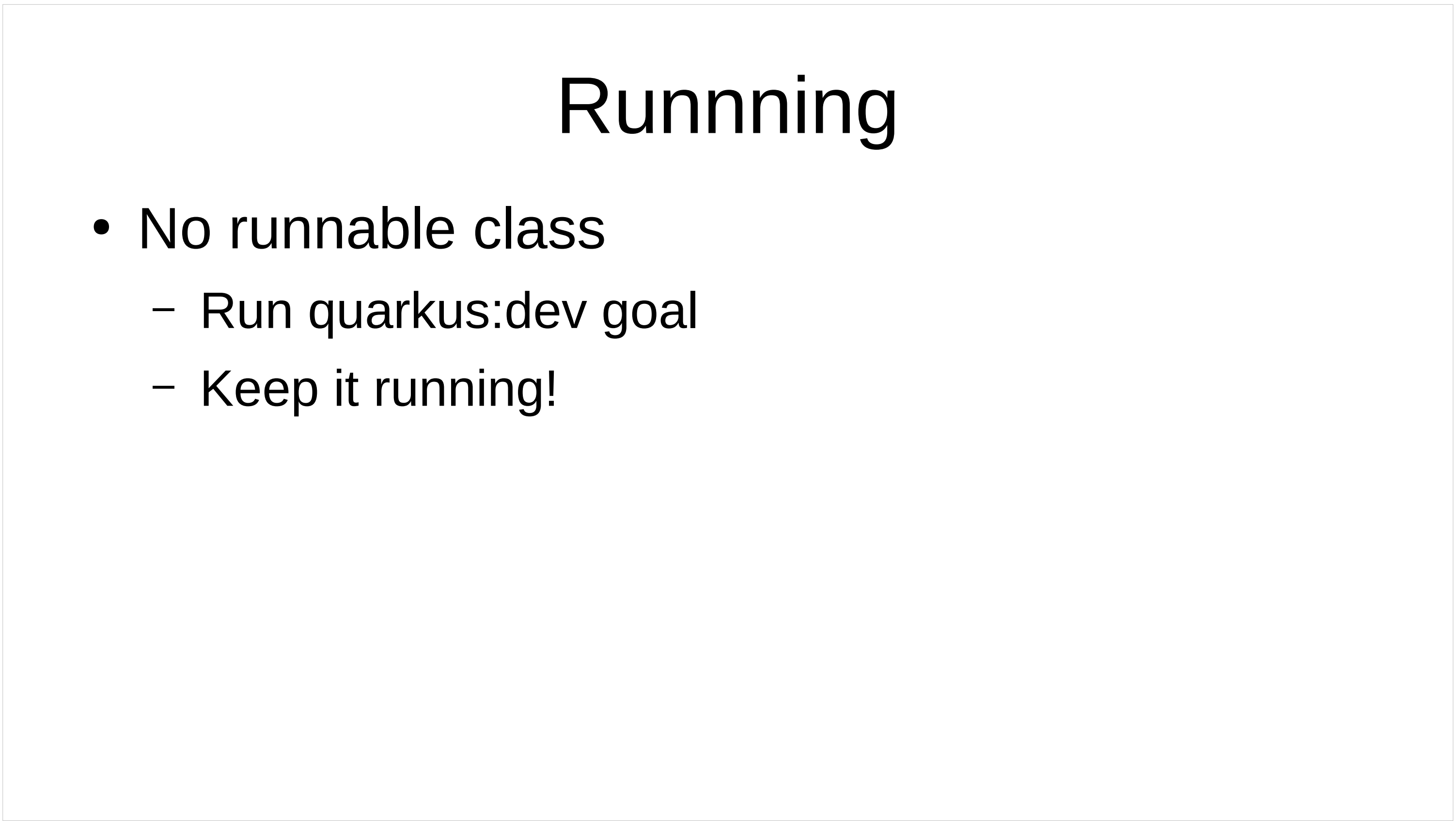
Dependencies

- Replace all libaries with Quarkus ones
  - Contain additional information for native execution
- Expose beans and entities from your dependant projects
  - beans.xml, Jandex index,

Quarkus defines many popular libraries as its extensions. Extensions contain „runtime” and „deployment” part, which makes libraries compatible with GraalVM.

Quarkus team worked directly with many library authors to make them compatible. Interesting example is removal of GUI login from MariaDB driver.

Our libraries had to be changed in some cases; for example, beans and JPA entities must be declared in order to be found during compilation.



Runnning

- No runnable class
  - Run quarkus:dev goal
  - Keep it running!

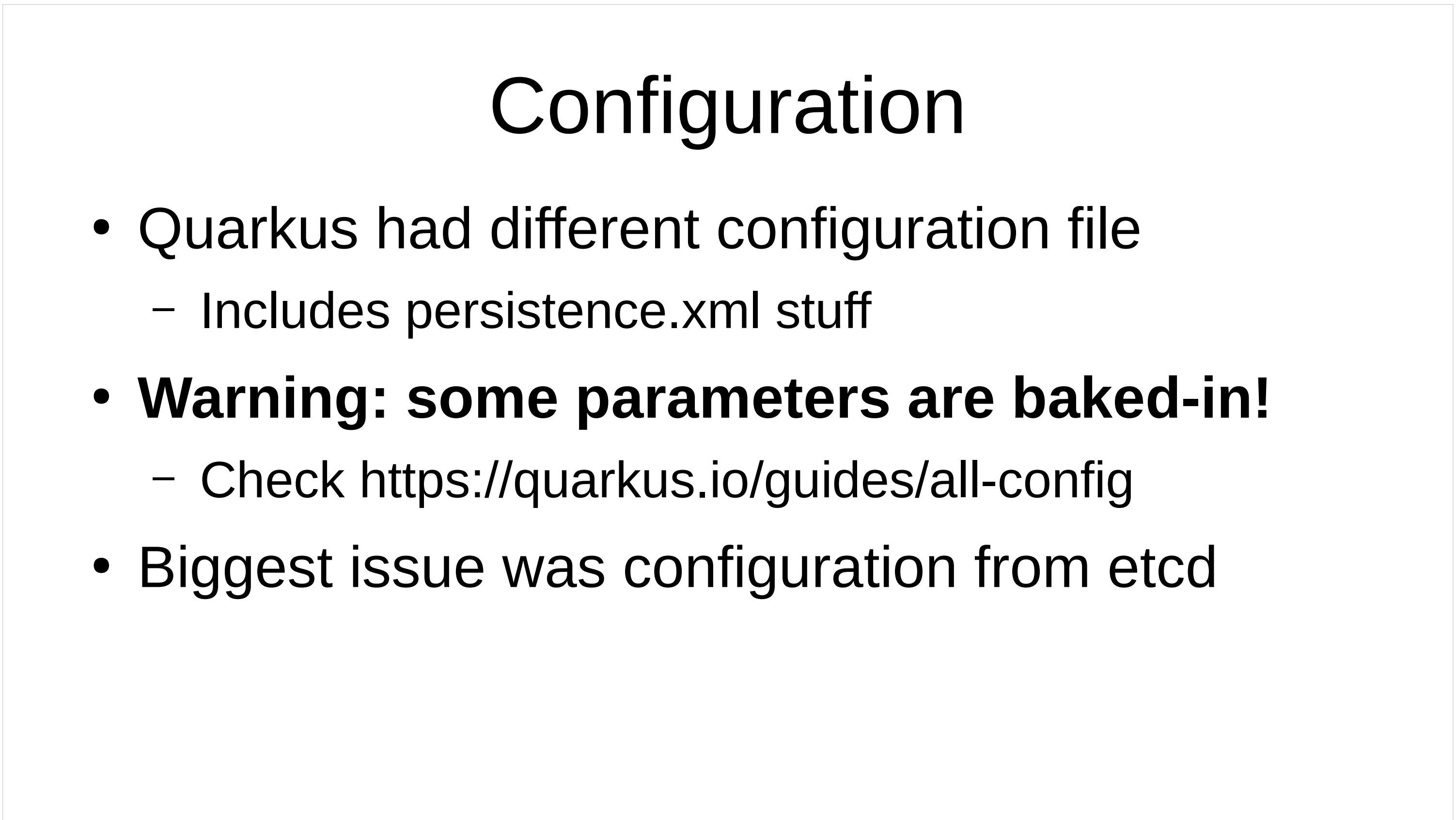
My first impulse was to define IntelliJ IDEA run profiles for running and debugging, but that is not how it works.

There is no main class.

Application must be started using quarkus:dev build tool goal.

Once you start the goal, keep it running; Quarkus will reload changes whenever you do HTTP request. TIP: if application does not have HTTP endpoint, just use heal.

Started application exposes debug port, to which IDE can attach using second run profile. Latest versions of IDEA simplifies that by providing the debug button in the „Run” window.



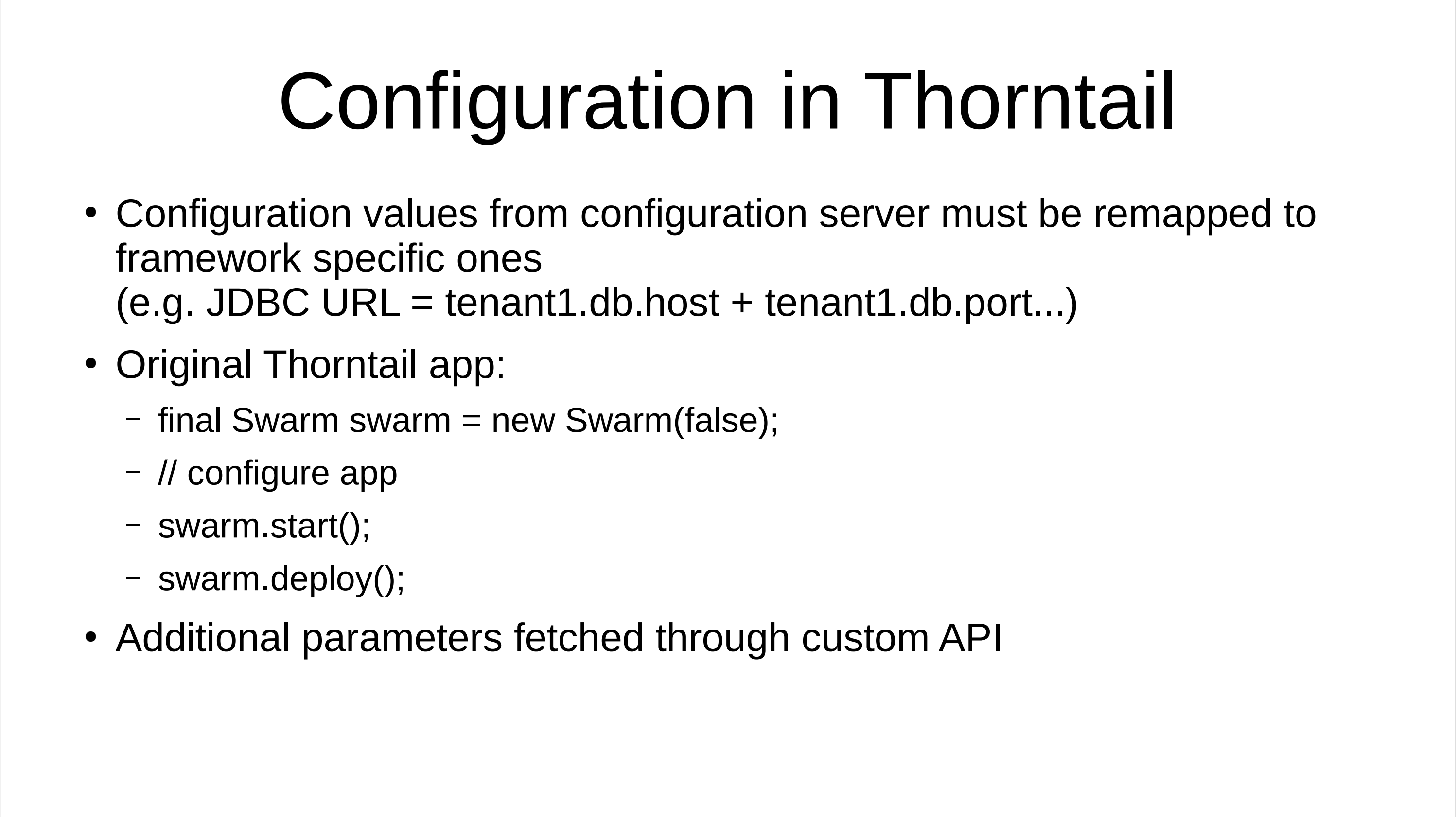
Configuration

- Quarkus had different configuration file
  - Includes persistence.xml stuff
- **Warning: some parameters are baked-in!**
  - Check https://quarkus.io/guides/all-config
- Biggest issue was configuration from etcd

Quarkus configures everything through single configuration file, either in properties or YAML format.

As building application creates optimized version where some stuff is resolved during the build process, some properties are immutable, e.g. HTTP port or using native SSL. Database drivers are also static so be careful.

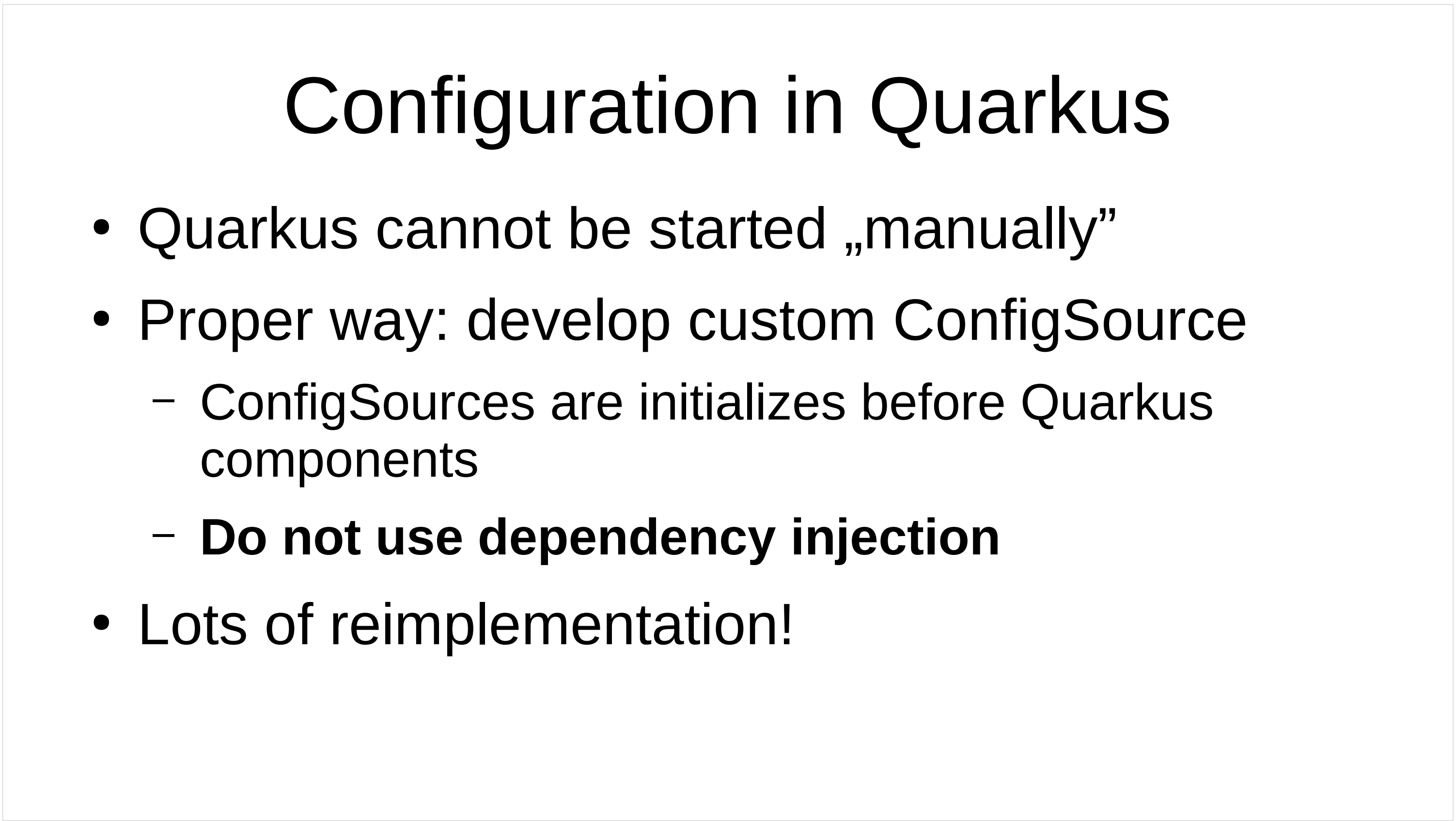
Biggest issue was using the configuration server, which is described on the next page.



Configuration in Thorntail

- Configuration values from configuration server must be remapped to framework specific ones
  (e.g. JDBC URL = tenant1.db.host + tenant1.db.port...)
- Original Thorntail app:
  - final Swarm swarm = new Swarm(false);
  - // configure app
  - swarm.start();
  - swarm.deploy();
- Additional parameters fetched through custom API

Existing application remapped some configuration values before starting the Thorntail. This is not possible with Quarkus.



Configuration in Quarkus

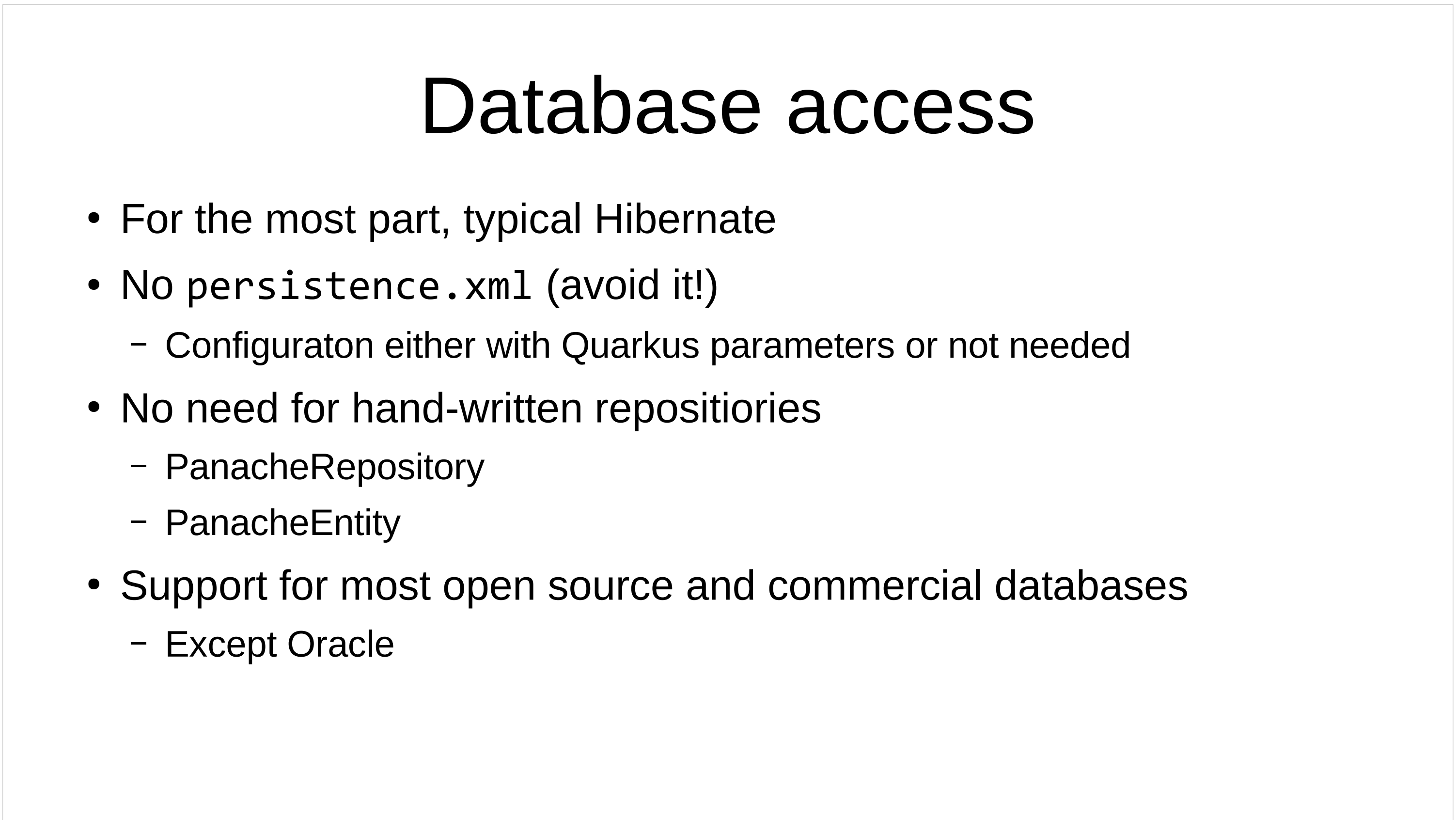
- Quarkus cannot be started „manually”
- Proper way: develop custom ConfigSource
  - ConfigSources are initializes before Quarkus components
  - **Do not use dependency injection**
- Lots of reimplementation!

Custom solution is replaced with standard ConfigSource. Implementing your own with heavy remapping could be tricky; be careful not to create recursive calls.

Configuring Application

- Additional benefit of ConfigSource: parameters fetched through standard API

  -
    ```
    @ConfigProperty(
       name=”myParamName”,
       defaultValue=0
    )
    int myParam;
    ```



Database access

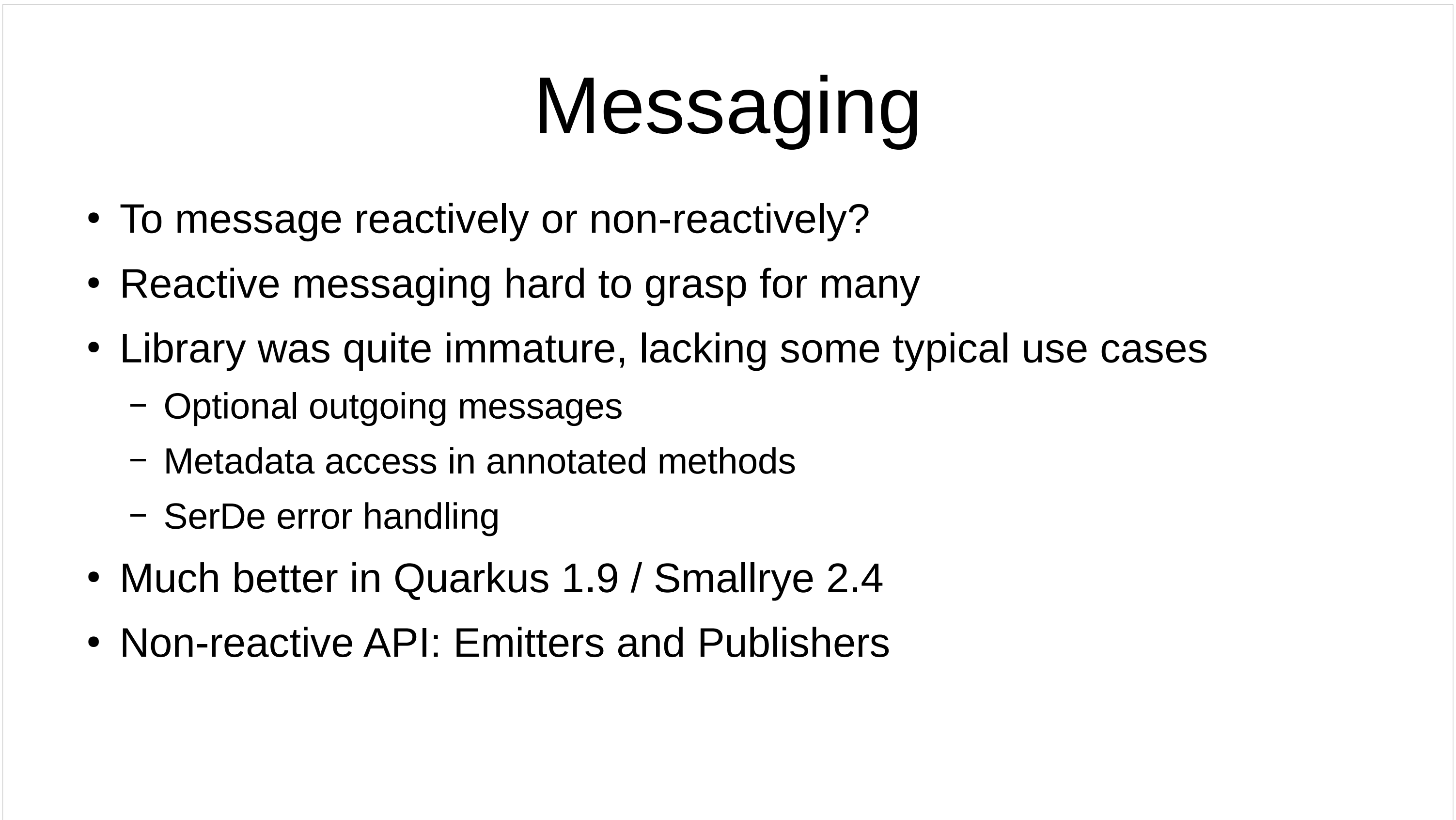
- For the most part, typical Hibernate
- No `persistence.xml` (avoid it!)
  - Configuraton either with Quarkus parameters or not needed
- No need for hand-written repositiories
  - PanacheRepository
  - PanacheEntity
- Support for most open source and commercial databases
  - Except Oracle

Although persistence.xml is supported, it can cause problems in different situations so avoid it. Most of the stuff is not needed anyway (entity declarations) and the rest can be configured using Quarkus properties.

Repositories are similar to Spring's, although class rather than interface based and with simplified query language rather than special method names.

Repositories are not needed if your entities implement PanacheEntity, but I do not like static methods on entities.

Driver extensions are created for most popular databases, except Oracle due to legal reasons. It is paradoxical because Oracle created GraalVM, but not the compatible drivers for their own database. With some tweaking, Oracle can be accessed in VM (non-native) mode.



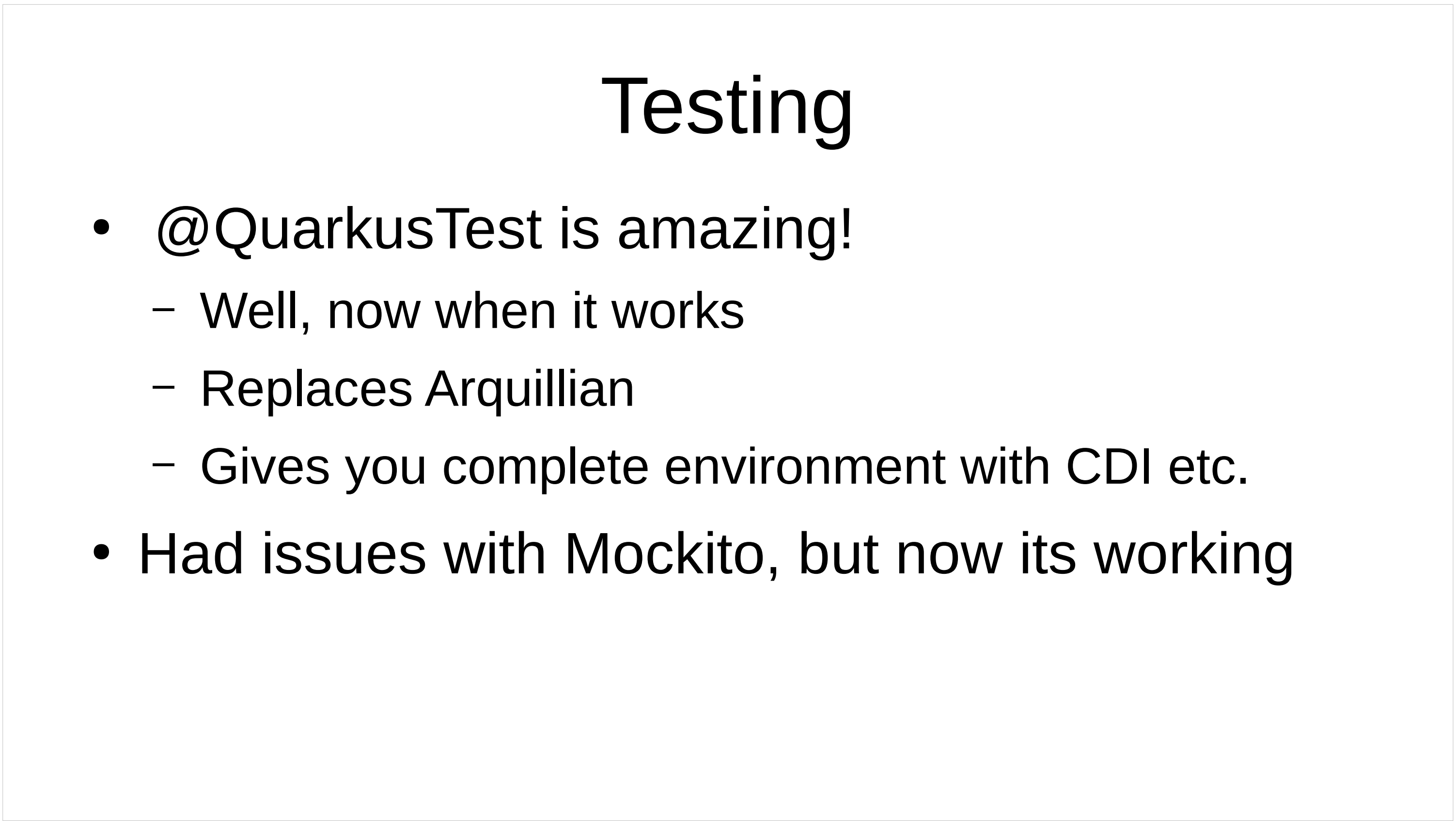
Messaging

- To message reactively or non-reactively?
- Reactive messaging hard to grasp for many
- Library was quite immature, lacking some typical use cases
  - Optional outgoing messages
  - Metadata access in annotated methods
  - SerDe error handling
- Much better in Quarkus 1.9 / Smallrye 2.4
- Non-reactive API: Emitters and Publishers

Official messaging API is SmallRye implementation of MicroProfile reactive messaging.

It is new library which is still heavily developed. There was interface changes, bugs and missing features, but now it provides simple, mostly stable and powerful API.

It can be also used in a non-reactive way.



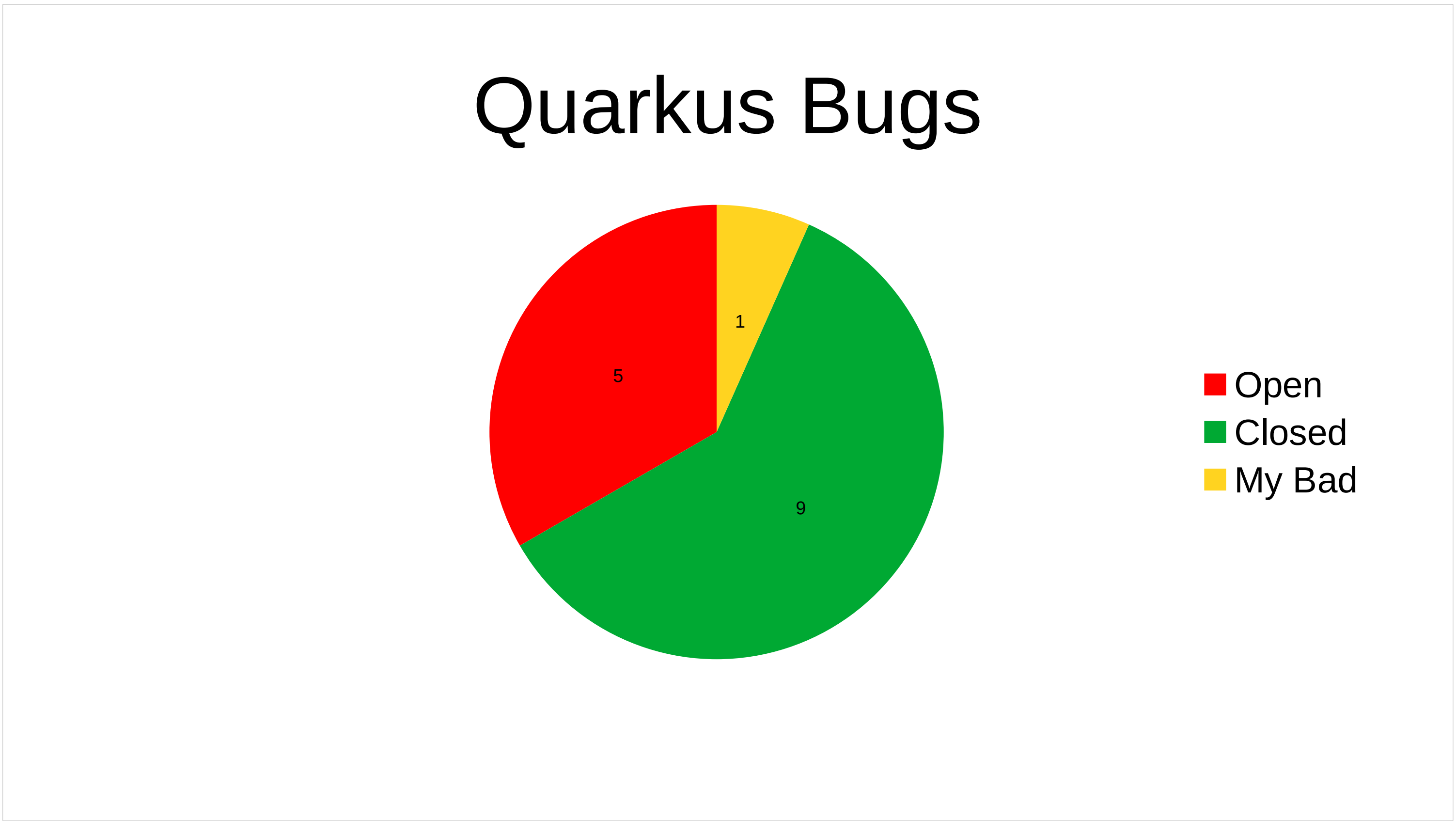
Testing

- @QuarkusTest is amazing!
  - Well, now when it works
  - Replaces Arquillian
  - Gives you complete environment with CDI etc.
- Had issues with Mockito, but now its working

If your definition of unit is bigger than a single class, @QuarkusTest will make your life easier.

With classic testing frameworks there is no support for @Inject or entity managers. You could either simulate everything with Mockito or try to configure Arquillian, which is not an easy task.

With @QuarkusTest you can write tests with deep structures of injected objects and access to in-memory database in no time.



More careful among the readers might ask how reliable is such new framework? Until spring of 2020., it was nightmare. Since then, situation improved a lot. Of 14 bugs that I have reported, 8 were resolved and one was my mistake. Quarkus team is very open for communication and suggestions.

To Quarkus or not

- Most serious issues were fixed around 1.4
- Recent versions cover everyday needs well
- Production ready
- Joy to work with

Thank you!