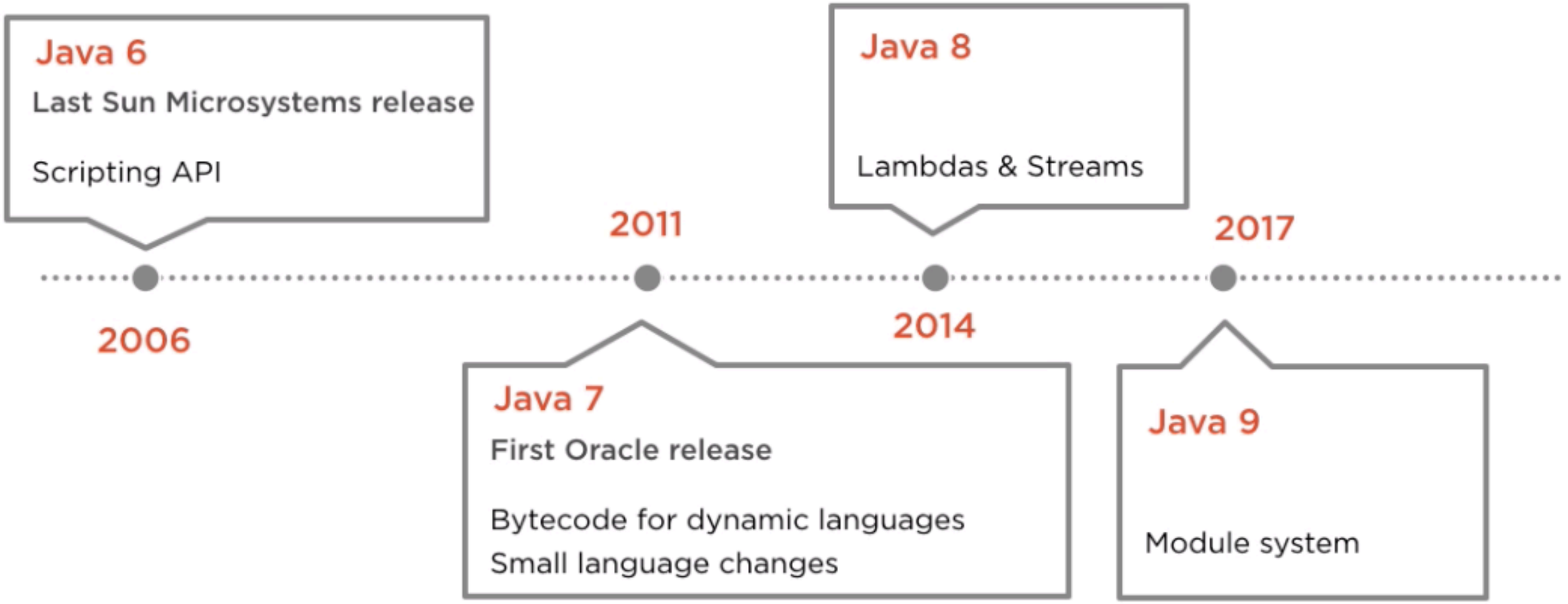
*Introduction*

# JAVA 9

Nanda

# Java 9 in Perspective



- Modules
- JShell
- Library & Language improvements
- New APIs
- Desktop Java Enhancements
- Performance & Security

# The Java Platform Module System

**One of the biggest changes in Java, ever**

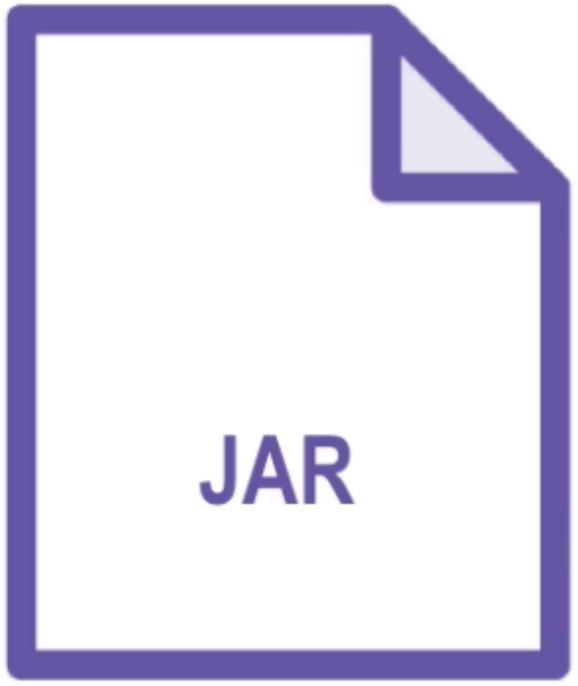
Language

Compiler

Virtual Machine

**Tooling**

# Before the Modular JDK




rt.jar

One huge library

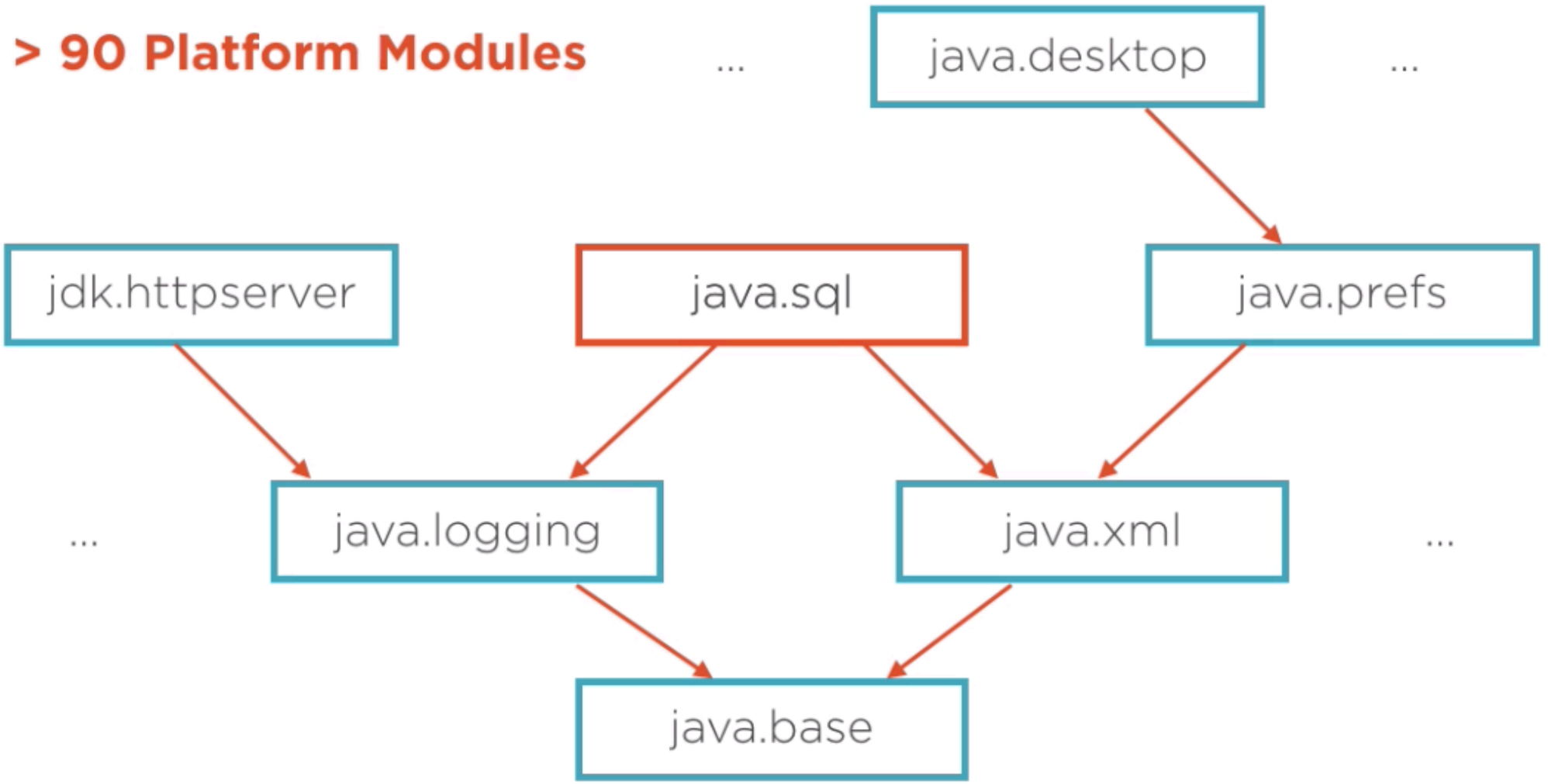
Many entangled classes

Hard to evolve

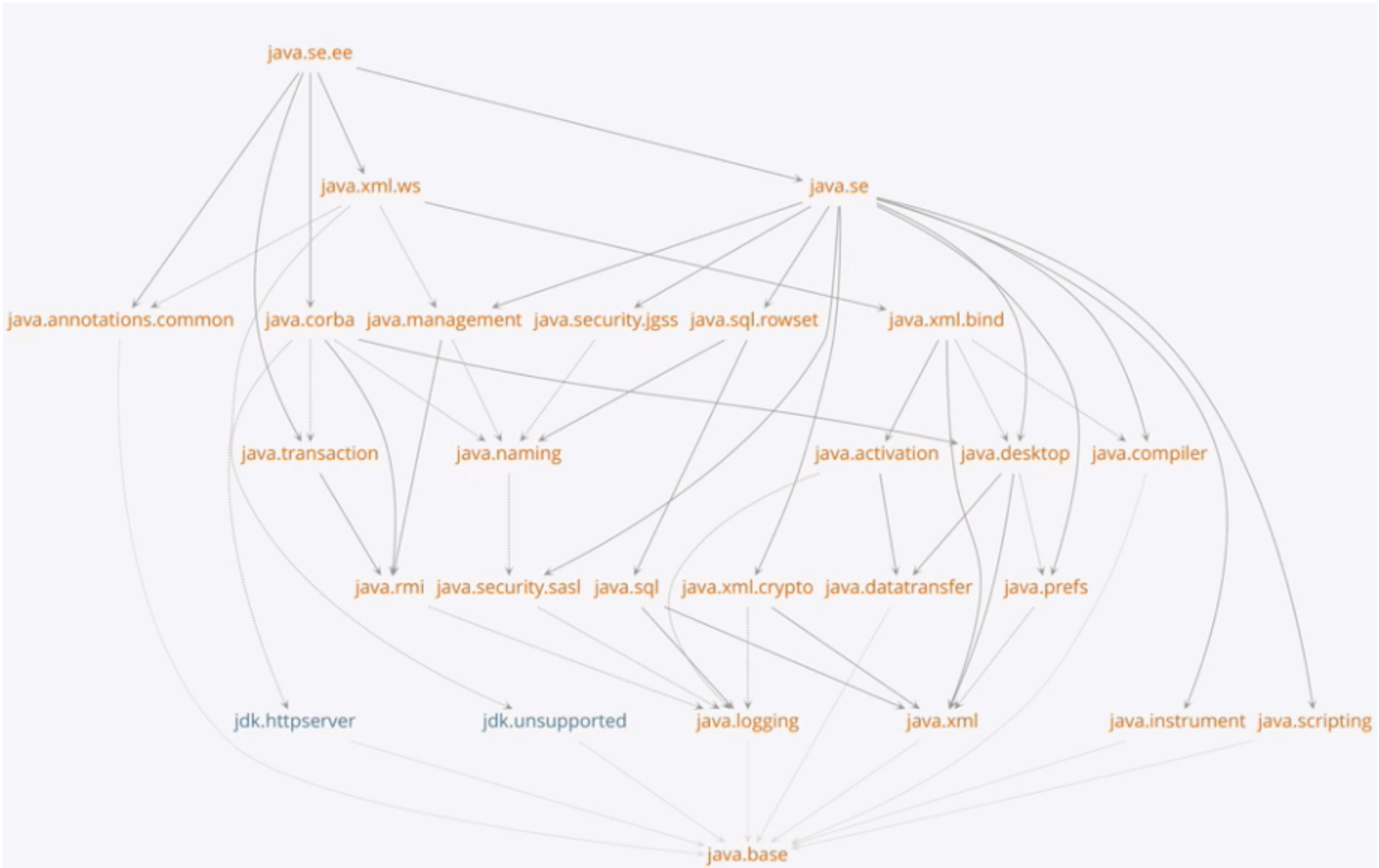
**Restricted by backward-compatibility**

# The Modular JDK: Explicit Dependencies

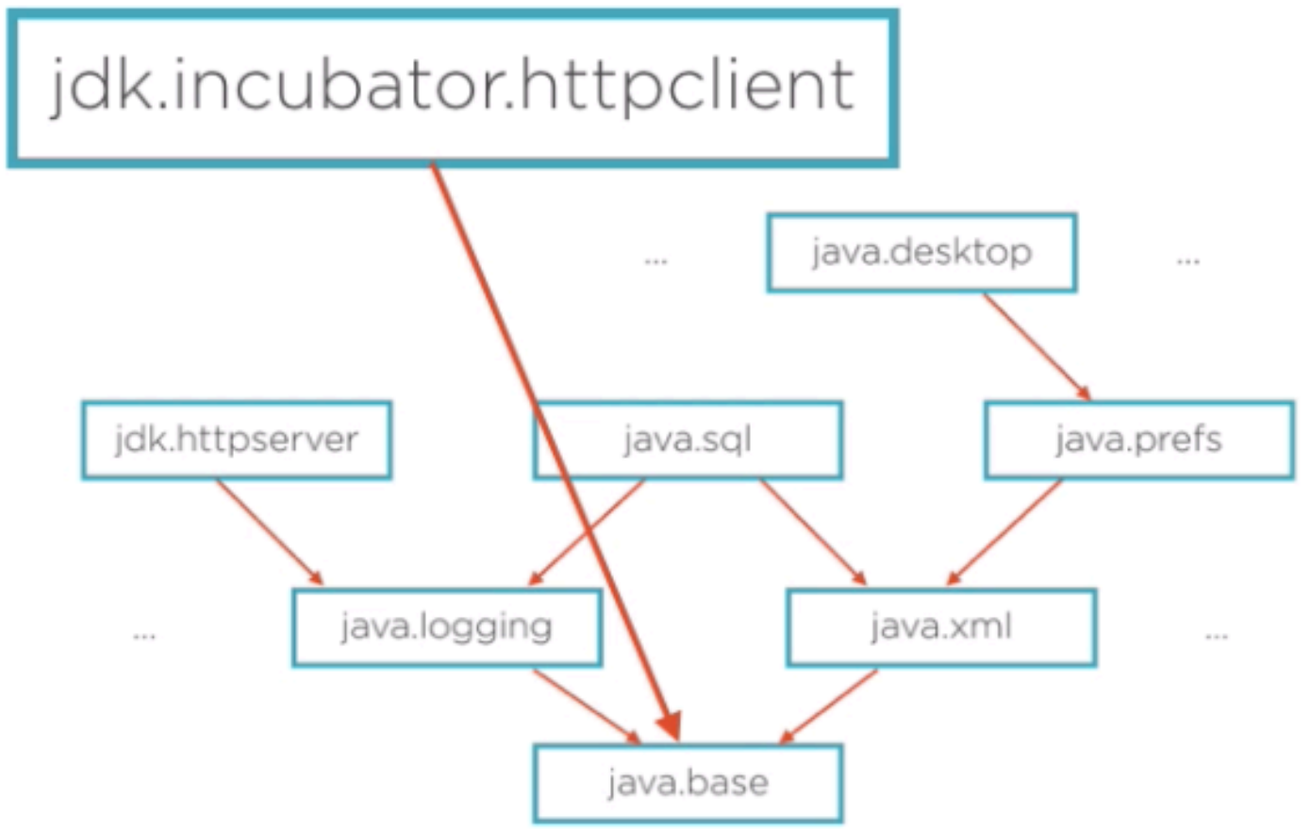
**> 90 Platform Modules**



# The Modular JDK: Explicit Dependencies



# The Modular JDK: Advantages




**Future-proof**

# What is a Module?

- A module has a **name**, it **groups** related code & is **self-contained**

## module-info.java

```
module java.base {

  exports java.lang;
  exports java.util;
  exports java.io;
  // and more

}
```

## Module Descriptors

## module-info.java

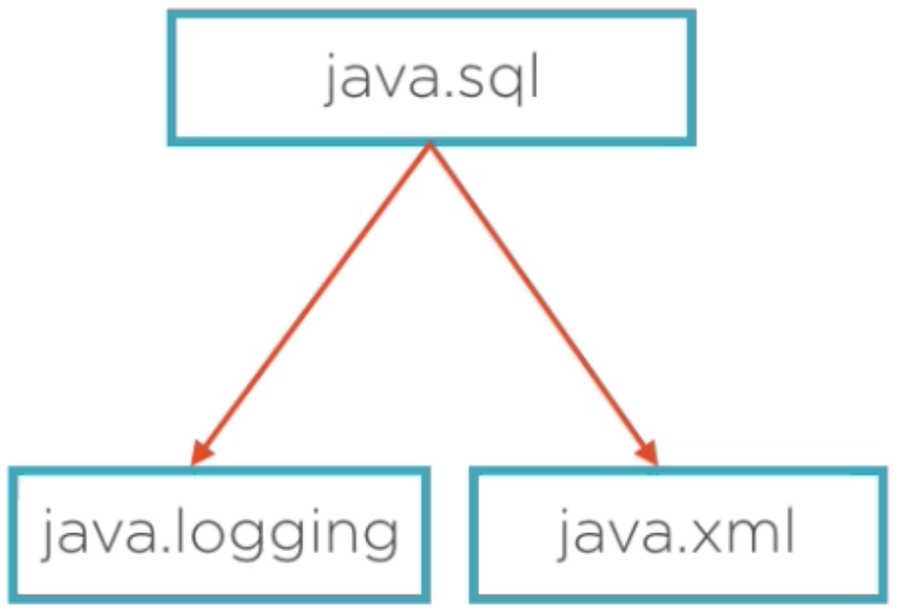
```
module java.sql {

  exports java.sql;
  exports javax.sql;
  exports
    javax.transaction.xa;

  requires java.logging;
  requires java.xml;

}
```

## Module Descriptors



- # java —list-modules
- # java —list-modules | grep “java\.”
- # java —describe-module java.sql

# JShell

## What Is JShell?

**R**ead → type code

**E**val → execute code

**P**rint → see results

**L**oop → interactively refine

Like Lisp, Groovy, Scala

# Why JShell?

## Quickly test ideas

No `public static void main(String[] args)` ceremony

Direct feedback without compilation

## Exploring APIs

Code completion

Built-in documentation

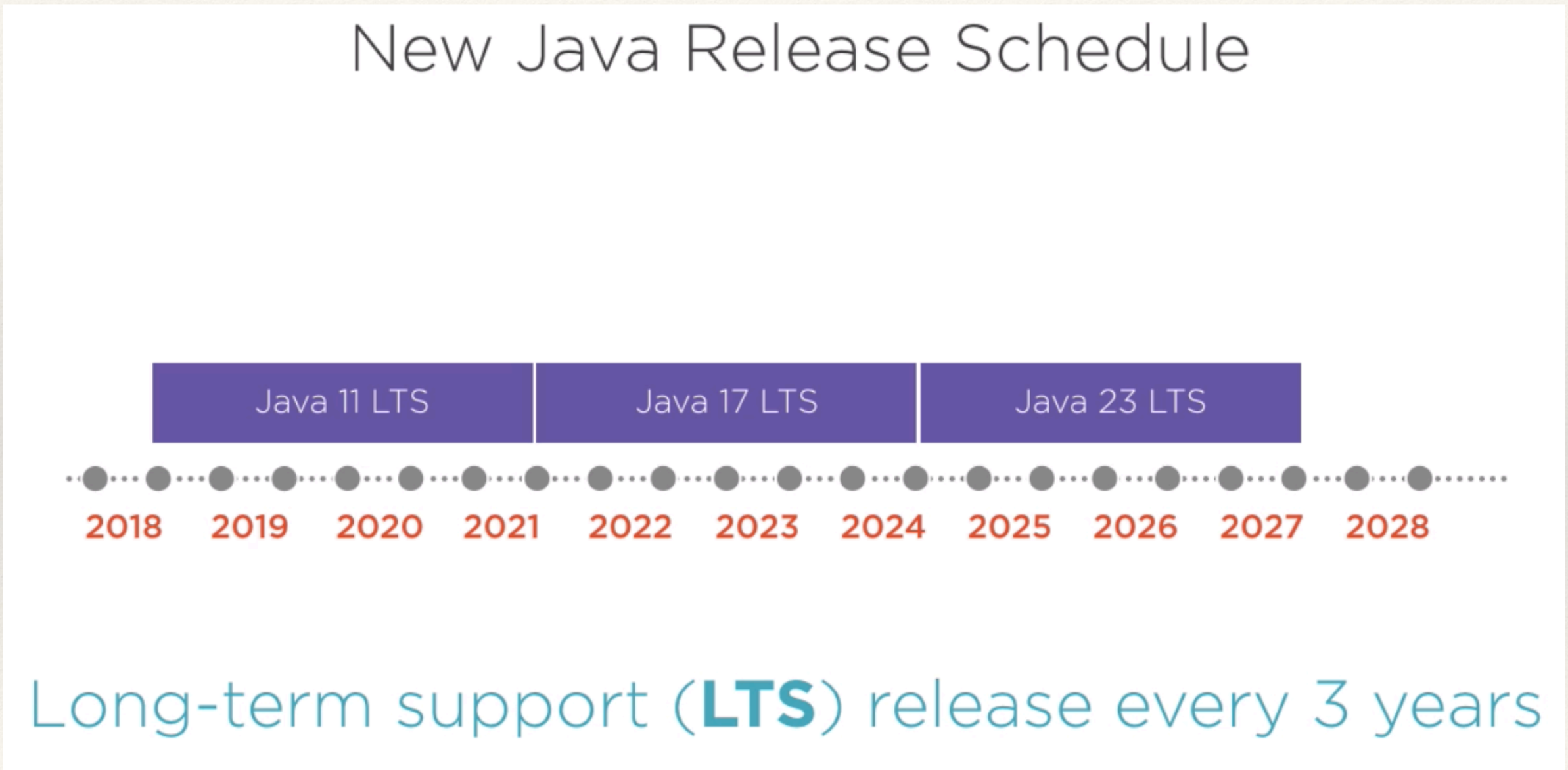
## Teaching tool

No IDE, just Java

# Java 10

- Introduction
- Local-variable Type Inference
- Performance Improvements

# Java 10


New Java Release Schedule
Java 11 LTS
Java 17 LTS
Java 23 LTS
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
Long-term support (LTS) release every 3 years

# Java 10

## New Java Release Schedule: Impact

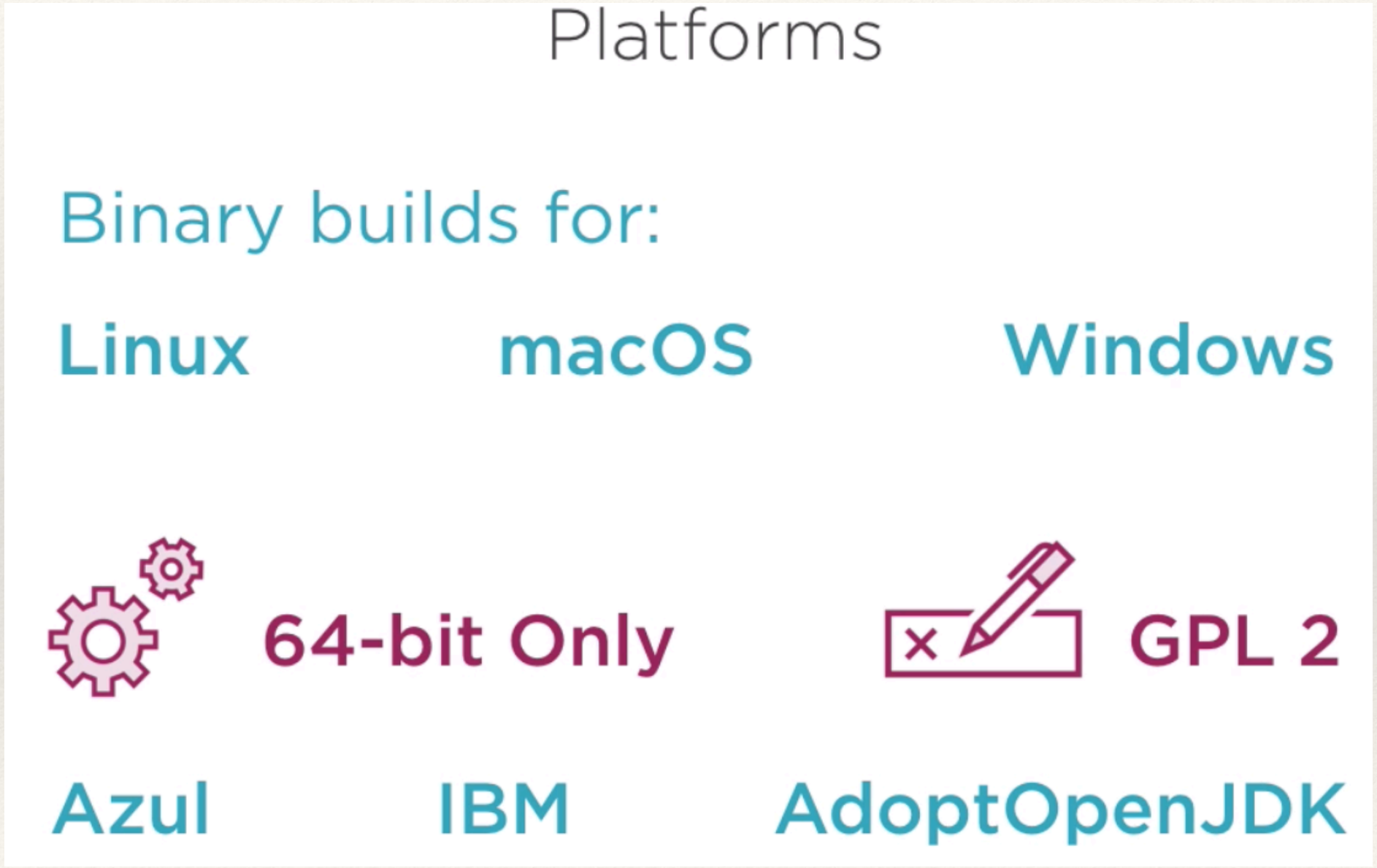
Increased frequency

Smaller & incremental

Non-LTS releases are **not** experimental

Deprecations/removals between LTS

# Java 10


Platforms
Binary builds for:
Linux
macOS
Windows
64-bit Only
GPL 2
Azul
IBM
AdoptOpenJDK

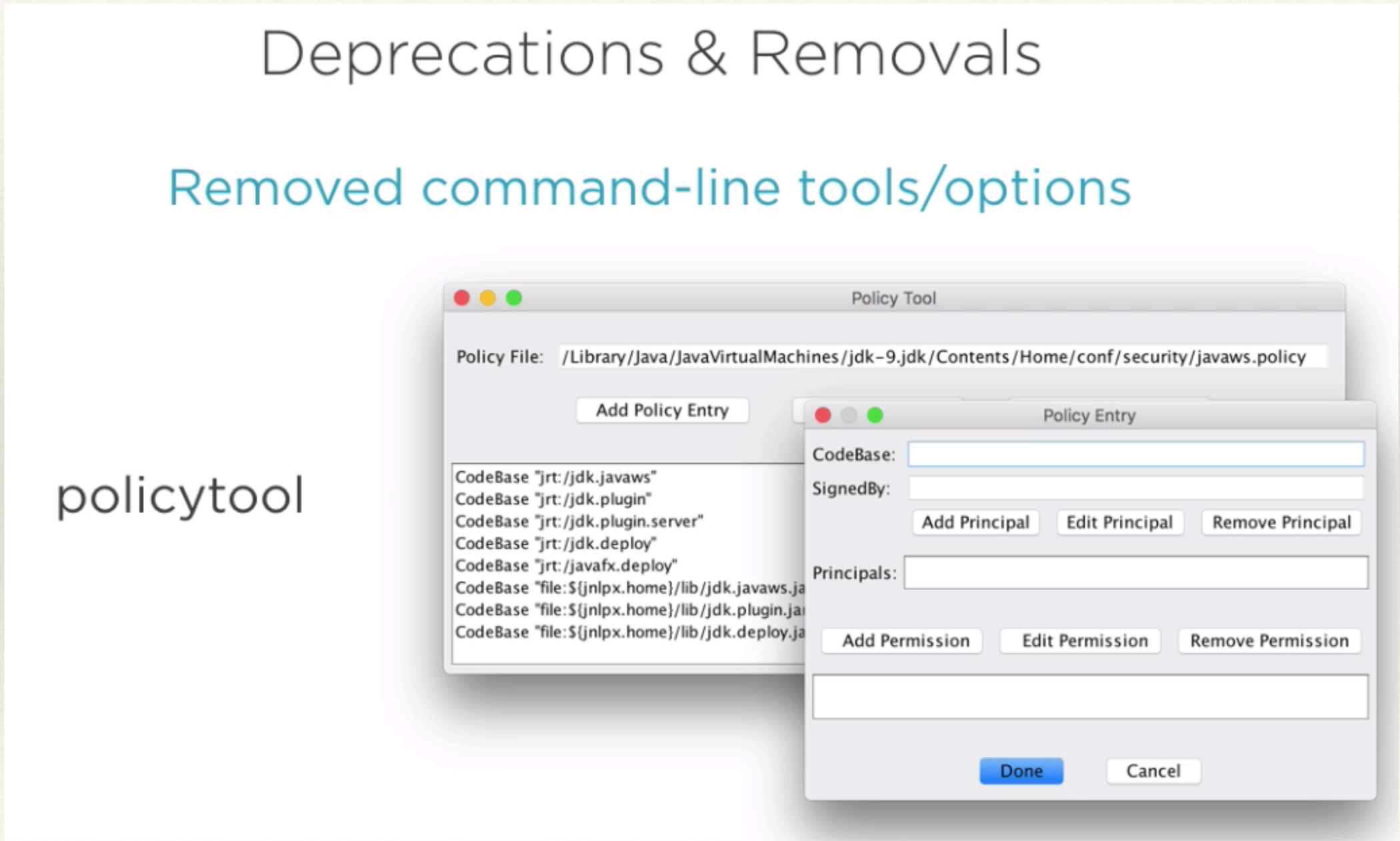
# Deprecations & Removals

## Deprecations & Removals

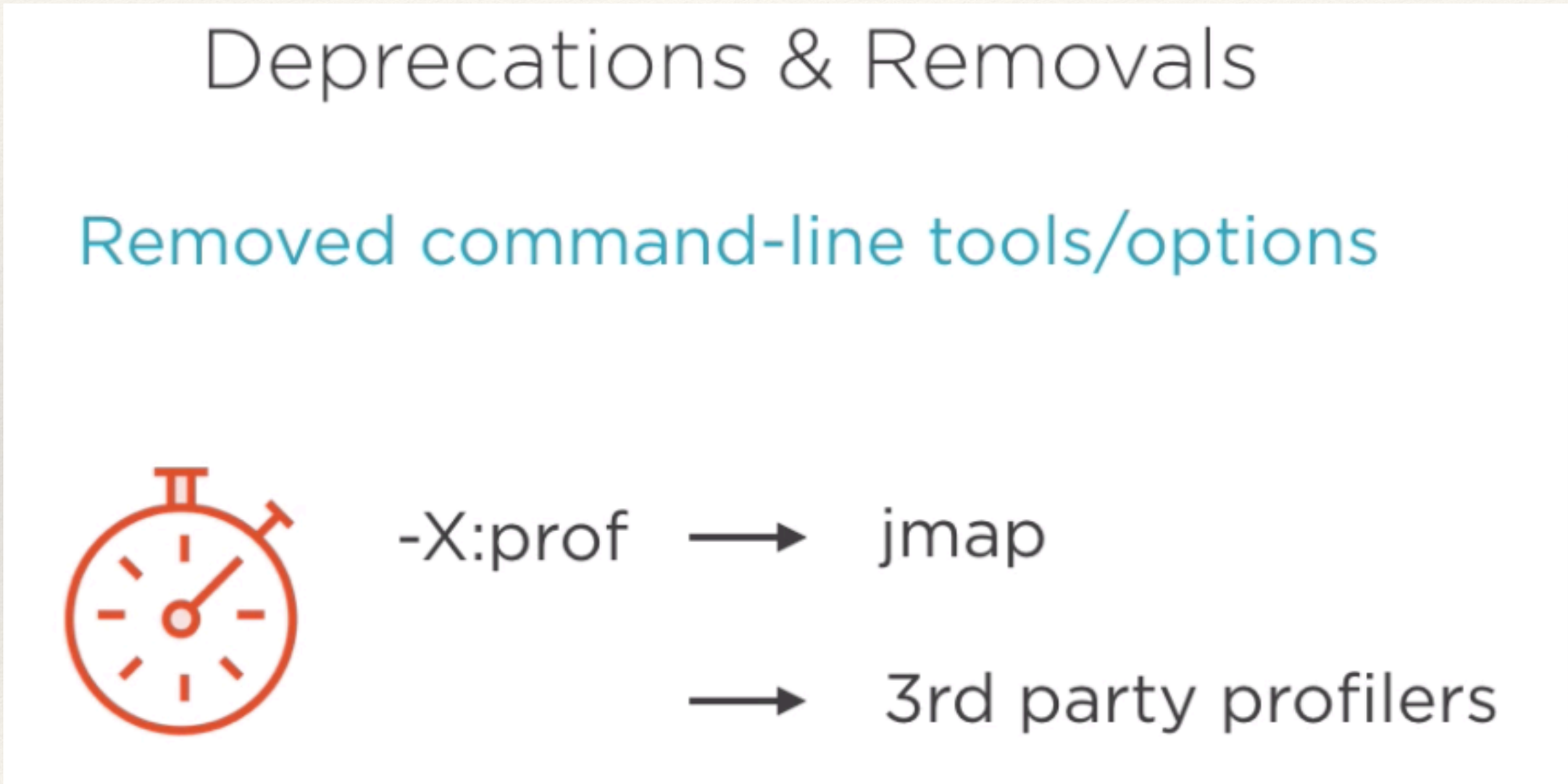
### Removed command-line tools/options

javah ⟶ javac -h <dir>

# Deprecations & Removals



# Deprecations & Removals



# Deprecations & Removals

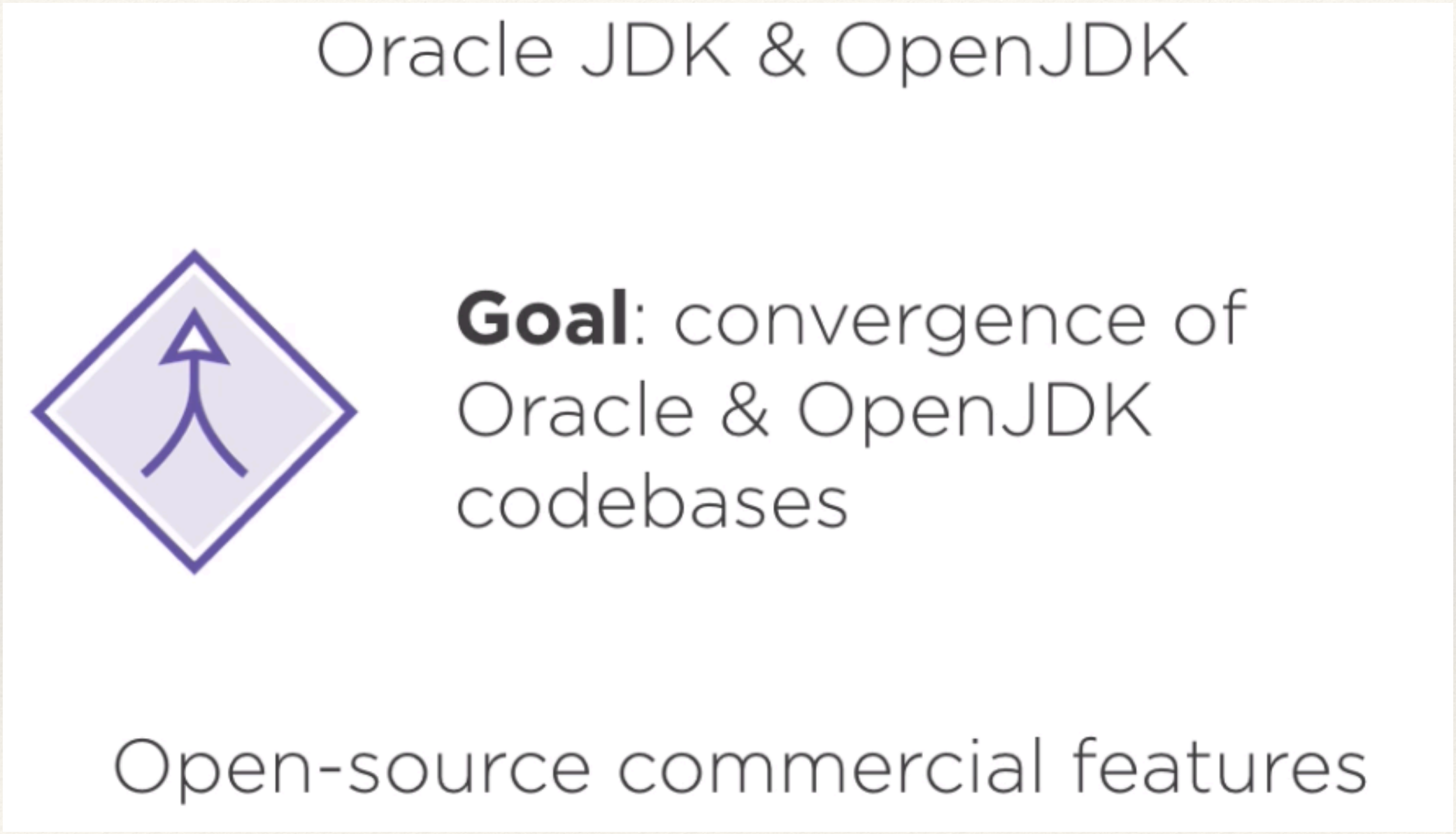
## Deprecations: APIs

```
java.security.acl   ⟶  java.security

java.security.{Certificate,Identity,IdentityScope,Signer}

javax.security.auth.Policy ⟶  java.security.Policy
```

# Oracle JDK & OpenJDK



# Oracle JDK & OpenJDK

# Oracle JDK & OpenJDK

Root Certificates

cacerts

Oracle's Root CA program to be ported to OpenJDK

# Local Variable Type Inference

- **“var”** for data type
- As the name suggests, it is for local variables

```
void  display() {

  var str = “Hello!”;

}
```

# Local Variable Type Inference

Why?

```
URL url = new URL("http://javamodularity.com");
URLConnection connection = url.openConnection();
BufferedInputStream inputStream =
    new BufferedInputStream(connection.getInputStream());
```

```
var bookurl = new URL("http://javamodularity.com");
var connection = bookurl.openConnection();
var bookStream = new BufferedInputStream(connection.getInputStream());
```

Focus on variable names

# Local Variable Type Inference

## Why Not?

Code is typically written **once**, but read **many** times by many people

```
var result = aService.findTheThing();
```

# Local Variable Type Inference

Reserved Type Name

Is var a new keyword? No!

```
var var = "var";
```

```
package com.myproject.var;
```

Backward compatibility for existing identifiers

# Local Variable Type Inference

Type Inference

```
int counter = 0;
counter += 1;
```

```
0: iconst_0
1: istore_1
2: iinc          1, 1
```

```
var counter = 0;
counter += 1;
```

```
0: iconst_0
1: istore_1
2: iinc          1, 1
```

# Local Variable Type Inference

## Type Inference

### Not new in Java:

```
List<String> myList = new ArrayList<>()
```

```
List<String> myList = Collections.<String>emptyList()
```

```
Predicate<String> p = s -> s.length() > 3
```

# Local Variable Type Inference

## Limitations of Type Inference

```
var myList = new ArrayList<>();
```

ArrayList<Object>

```
var myList = new ArrayList<String>();
```

# Local Variable Type Inference

Local Type Inference Only

Method parameters

Return types

Fields

Catch blocks

```
public void aMethod(var input) {
   ...
}
```

# Local Variable Type Inference

## Application Class Data Sharing

Improve VM start-up time

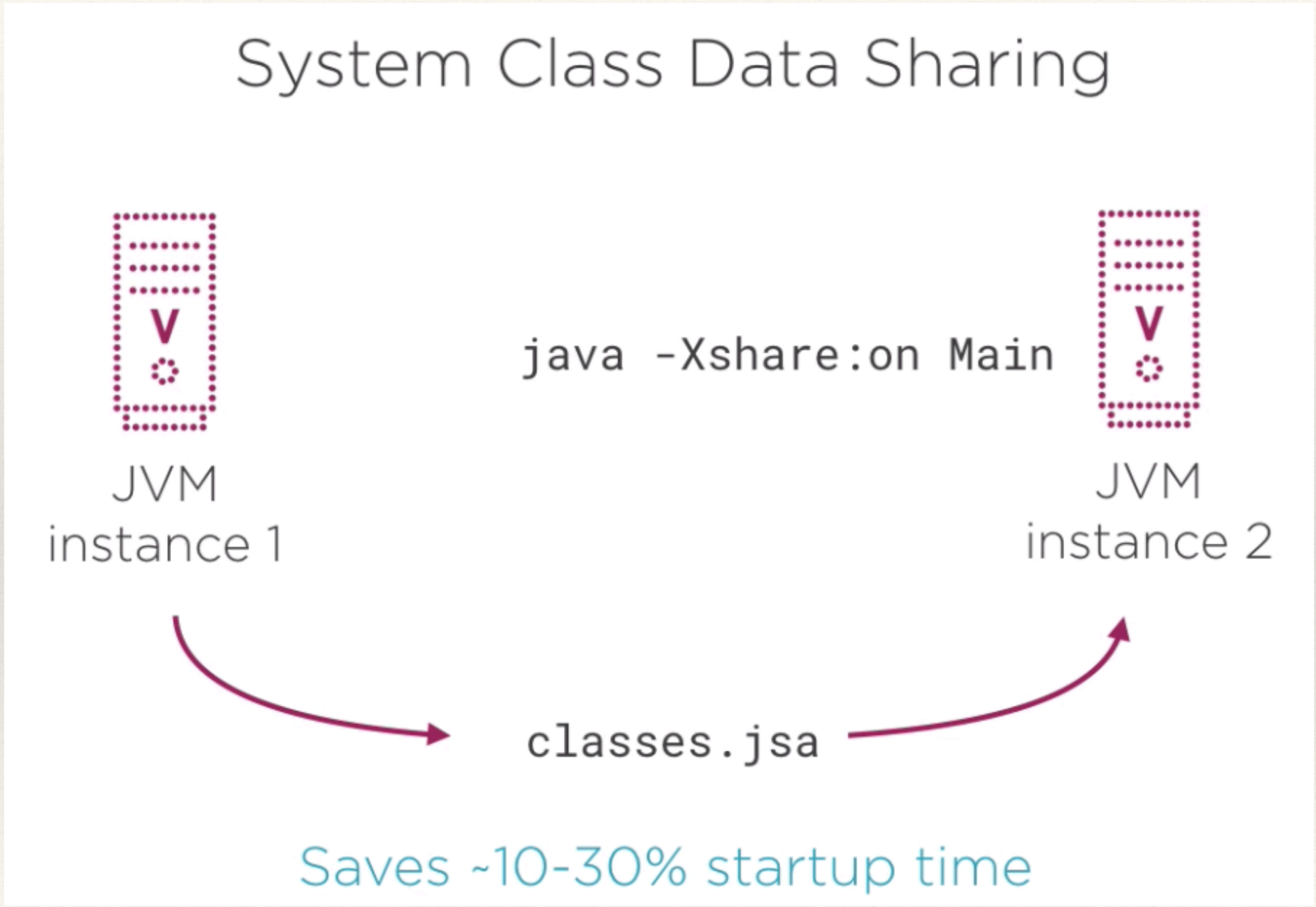
Reduce memory footprint

Shared lib of class metadata

Originally system classes only

Useful with multiple VMs running same code, or with repeated executions

# Local Variable Type Inference



# Local Variable Type Inference

## Improved Container Awareness

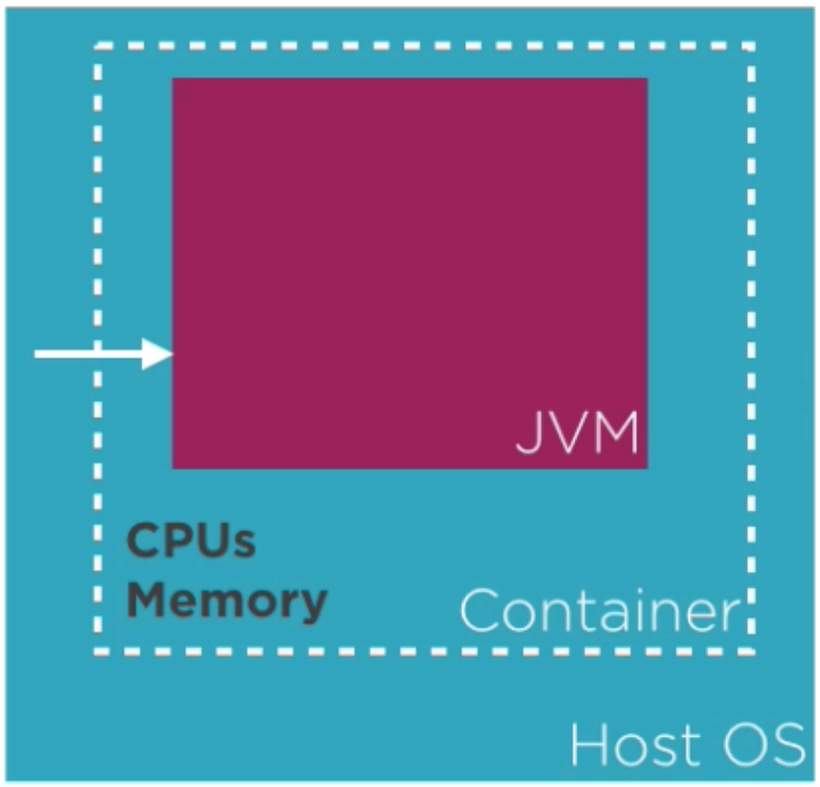
Java now detects container

Query container instead of host OS

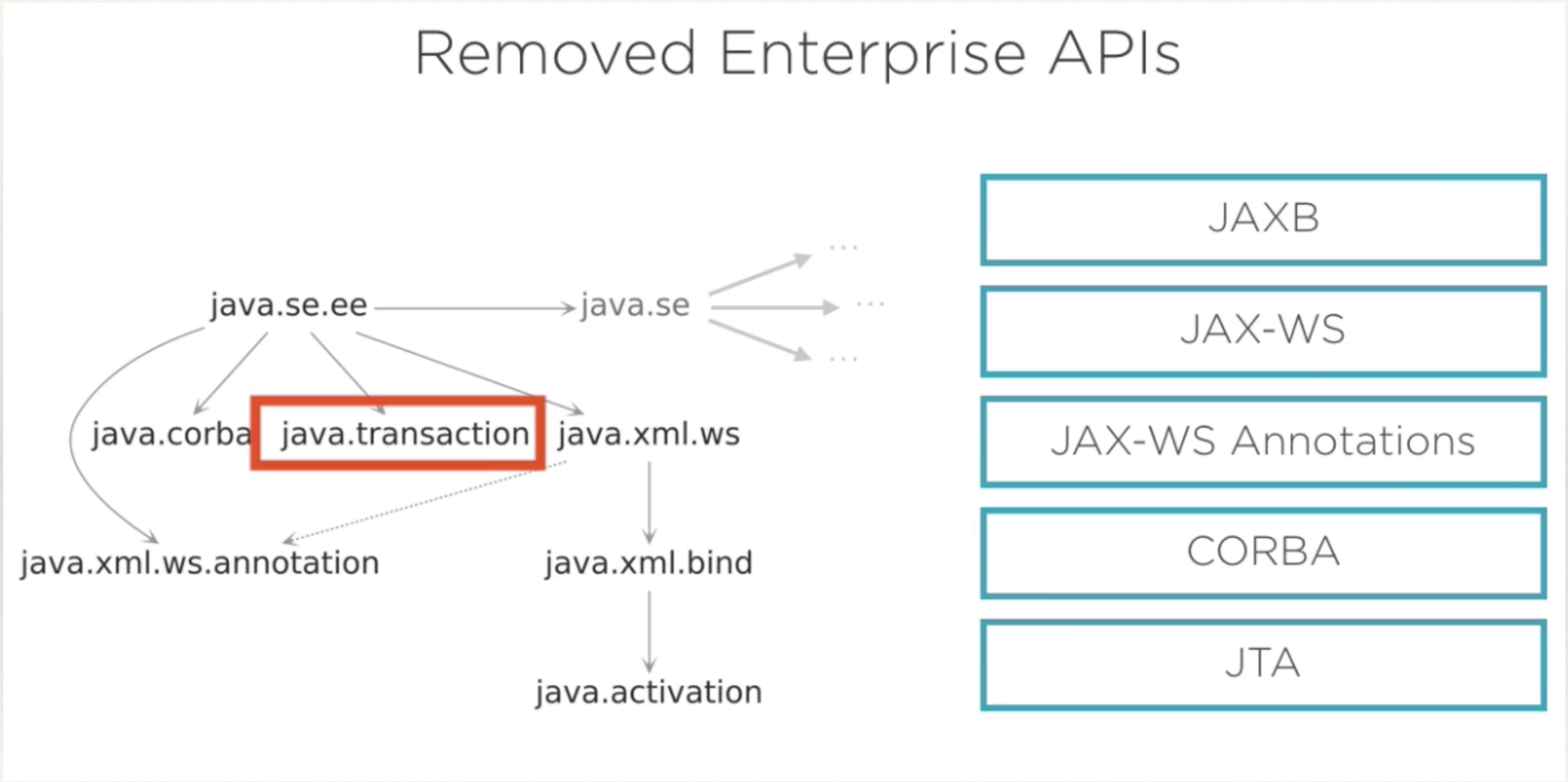
Can attach to Java process

**Linux & Docker only**

```
-XX:ActiveProcessorCount=<n>
-XX:InitialRAMPercentage
-XX:MaxRAMPercentage
-XX:MinRAMPercentage
```



# Java 11


Removed Enterprise APIs
java.se.ee
java.se
...
java.corba
java.transaction
java.xml.ws
java.xml.ws.annotation
java.xml.bind
java.activation
JAXB
JAX-WS
JAX-WS Annotations
CORBA
JTA

# Java 11

## Removed & Deprecated Technologies

~~Applets~~

~~Java Web Start~~


Nashorn

# Java 11

New HttpClient API

Ability to run source code directly

Two new garbage collectors

Other improvements

# Java 11

## Removed Enterprise APIs: Alternatives

java.xml.bind

JAXB

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
</dependency>
```

```
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
</dependency>
```

# Java 11

## Removed Methods

**Thread**

```
countStackFrames()
~~destroy()~~
resume()
stop()
~~stop(Throwable obj)~~
suspend()
```

```
java.lang.System::runFinalizersOnExit

java.lang.Runtime::runFinalizersOnExit
```

# Java 11

## Removed Methods

SecurityManager

```
checkAwtEventQueueAccess
checkSystemClipboardAccess
checkTopLevelWindow
checkMemberAccess
```

```
checkPermission(java.security.Permission)
```

# Java 11

## Launching Single-file Source-code

Hello.java

```
$ java Hello.java
Hello Pluralsight!
```

`--source <version>`

# Java 11

## Scripting

./listfiles

#!
shebang

```
#!/usr/bin/java --source 11
import java.nio.file.*;

public class ListFiles {
  public static void main(String[] args)
                    throws Exception {
    Files.walk(Paths.get(args[0]))
         .forEach(System.out::println);
  }
}
```