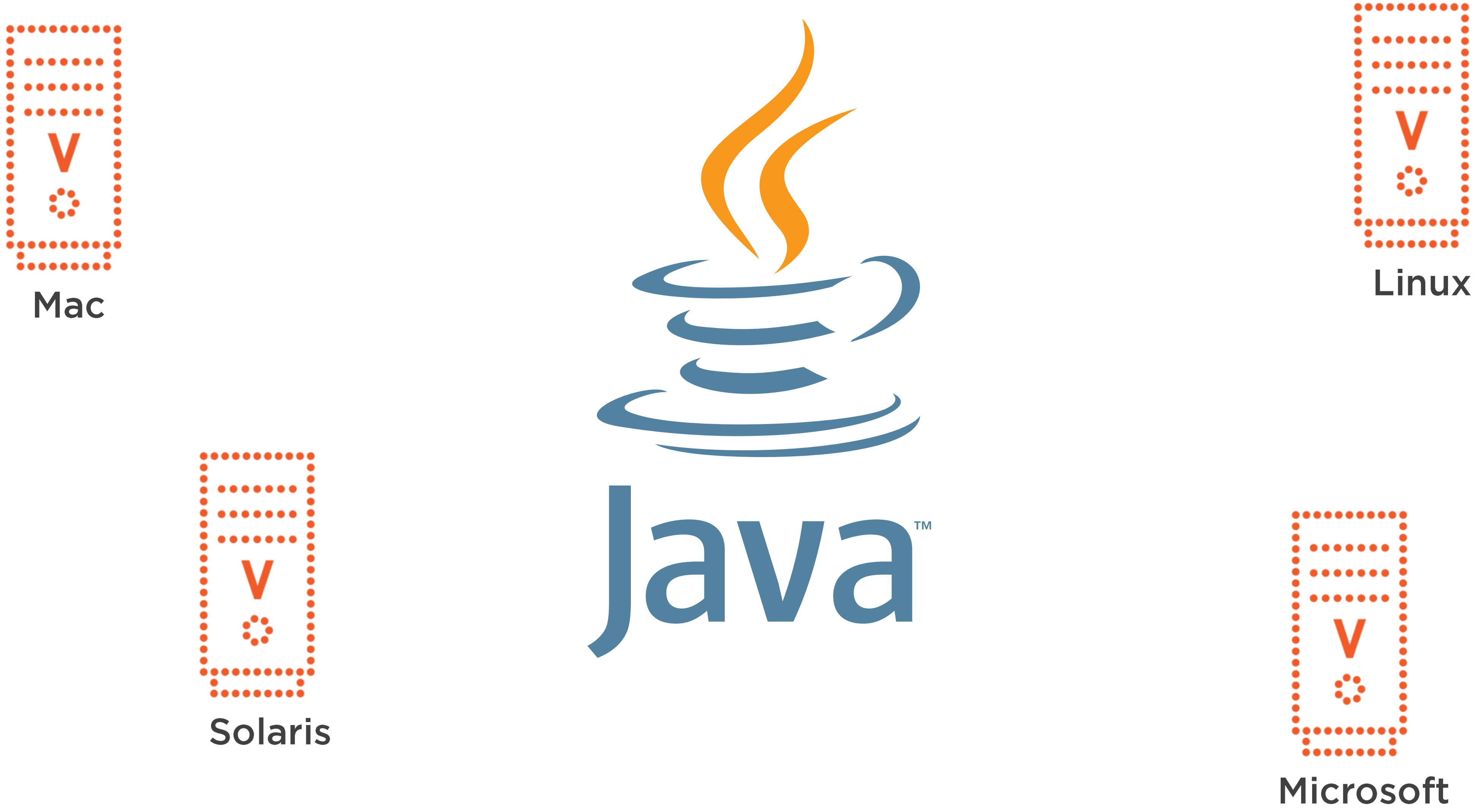
Working with Jenkins and Groovy Together

Chris B. Behrens
SOFTWARE ARCHITECT

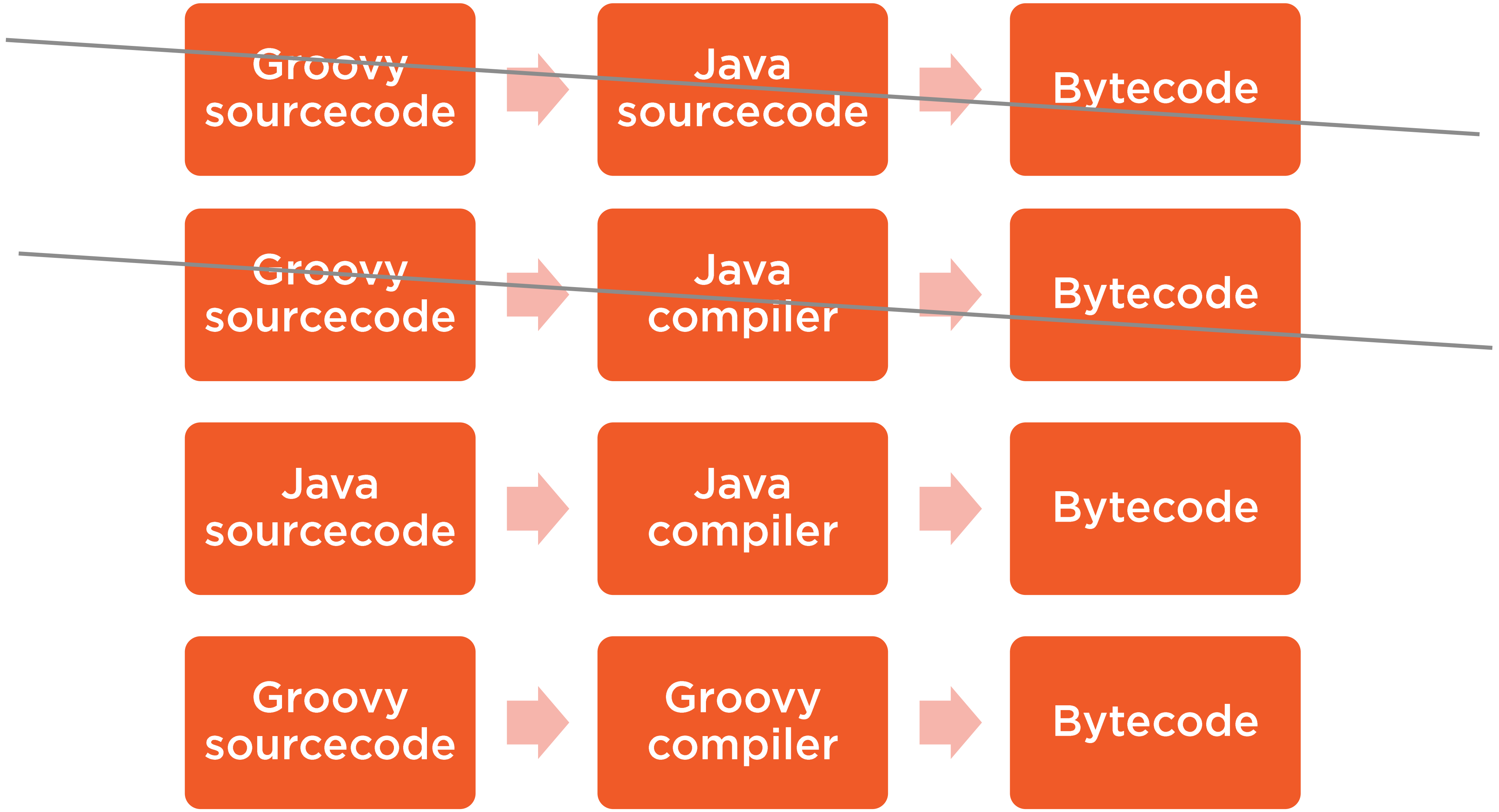
@chrisbbehrens


Mac
Linux
Java™
Solaris
Microsoft

Groovy runtime

~~Groovy runtime~~

Groovy Compiles to Bytecode



The Jenkins Plug-in Model

Applications that conform to a plug-in standard

Groovy support implemented as a plug-in

Demo

Install the Groovy plug-in using the UI

Configure the plug-in

Execute a simple Groovy script inside of Jenkins

System Groovy Steps

System steps have elevated privileges

System steps execute INSIDE the Jenkins Java VM

The Groovy console executes scripts as system scripts

More Default Imports in Jenkins

```
import jenkins.*

import jenkins.model.Jenkins

import hudson.*

import hudson.model.*


println(Jenkins.instance.pluginManager.plugins)
```

Demo

Execute a couple of scripts in the script console

Create a system script to execute another build

Work towards generalizing from hard-coded parameters

With parameters in script

With a parameter in the build definition

Script Types Wrap-Up

What's the point of a non-system script?

When system scripts can do everything that non-system scripts can...

"Everything outside of Jenkins internals"

Non-system scripts have less security freight

Executing Groovy Scripts on Startup

Configuration as Code

init.groovy.d

Demo

Create a short init script

- Set a system message identifying our server
- Disable the remember me checkbox on the login screen
- Create an init.groovy.d directory in bash

Copy our script there

Restart Jenkins

Verify our changes

Working with Exceptions in Groovy

Our startup script was the happy path

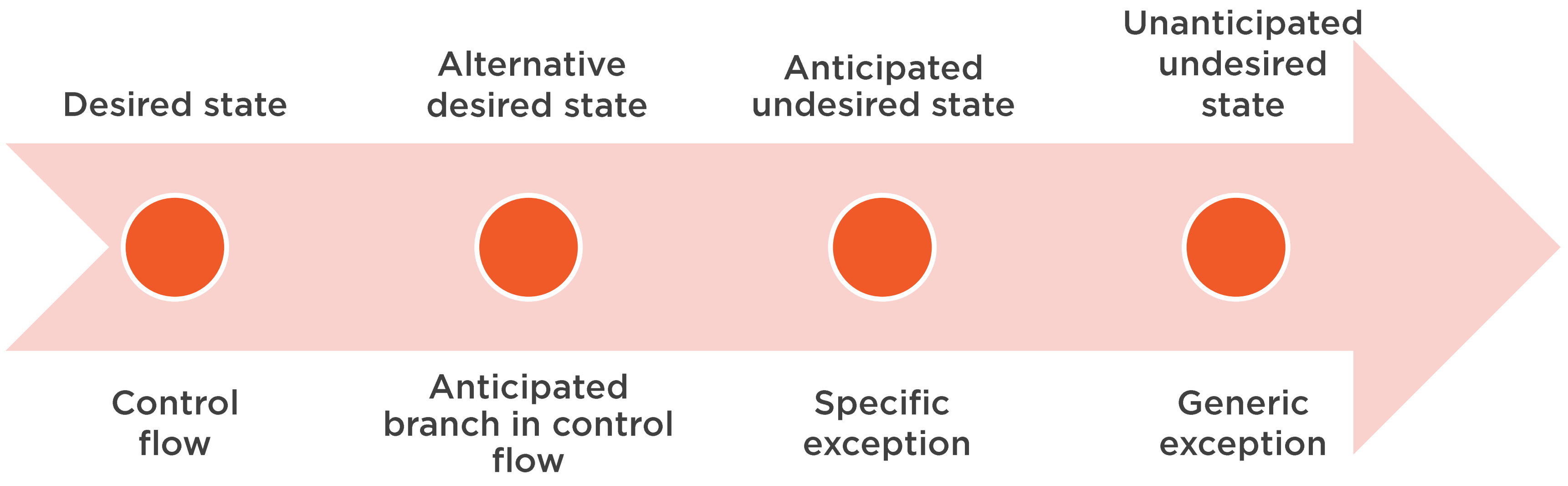
Infrastructure as code – configuration in script

An exception would keep Jenkins from starting

Make your script resilient against problems

Exceptions should be exceptional

Exceptions


Desired state
Alternative desired state
Anticipated undesired state
Unanticipated undesired state
Control flow
Anticipated branch in control flow
Specific exception
Generic exception

Exceptions Are Costly

Executing in the Java VM

The time to assemble the stack can add up

Handling Exceptions

```
try{
    // do stuff
}catch(ex){
    // do stuff when stuff breaks
}finally{
    // always do this stuff
}
```

Handling Exceptions

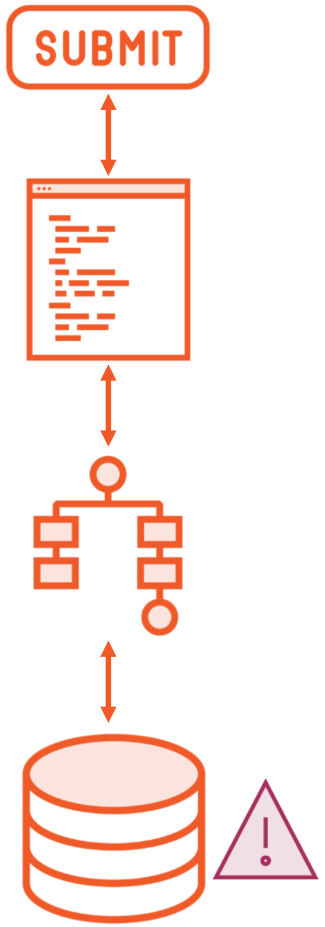
```
try{
    // do stuff
}finally{
    // always do this stuff
}
```

An Exceptions Interview Question

You're reviewing another programmer's code, and they have code inside a try block, and code inside a finally block, but with no catch block. What can we assume about the programmers intentions in implementing this model?

The programmer had cleanup code they wanted to execute, but wanted any possible exception to be handled higher up in the stack.

An Exception Example



Top-level Error Handlers

Let the exception reach the top of the stack

Good news: handling it everywhere is NOT the answer

Demo

Try-Catch-Finally blocks

- At multiple levels in the stack
- How to implement the model I described

Throwing our own exceptions

*Grabb*ing External Libraries

Grape is the Groovy package manager

Analogous to Nuget or npm

A Dependency Scenario

Connect to db for config

JdbcTemplate

Jdbc = Java Database Connectivity

Does not exist in default install

Grabbing Spring ORM

```
@Grab(group='org.springframework', module='spring-orm', version='5.2.4.RELEASE’)

// from mvnrepository.com

import org.springframework.jdbc.core.JdbcTemplate
```

Summary

Knowledge of Groovy essentials

Automate the triggering of a Jenkins build

Getting parameters into our script

Security considerations involved

Executing startup scripts

- The init.groovy.d directory
- A sample script useful for setting a few useful properties

Exception Handling

Importing external libraries with Grape

- Using Grab