Spring Under the Hood

Frank P Moley III
Kansas City Spring User Group

Who am I

- Internet Architect with Garmin International with 13+ years of software engineering experience
- Spring user, committer, and aficionado - Spring Certified Professional
- Public speaker focusing on Spring, Architecture, and team dynamics
- @fpmoles on Twitter, fpmoles on GitHub as well

Agenda

- Configuration styles of the Application Context
- Introduction to the Bean Lifecycle and explanations of the phases
- Points of interest for customizing the bean lifecycle
- Spring 4.0 changes to the runtime

Why should you care

- One does not need to understand a framework to use it, however to become an expert in its application, fundamental understanding is critical
- The more you understand how Spring works, the better suited you are to solve real world problems using Spring
- Spring is a pretty amazing framework, and you can learn a lot personally on craftsmanship buy studying the guts of Spring code

Configure the Application Context

- Everyone using Spring understands that configuration of some sort is necessary.
- Configuration Options
  - XML
  - Annotation Based
  - Java Config

XML Configuration

- Original form on configuration of the application context, precedes JDK 1.5 (5.0) when annotations where introduced
- Supports inheritance, namespaces, and “verbose” bean definition
- Still supported fully in Spring 4.0, however spring.io examples are limited in favor of Java Configuration

Annotation Configuration

- Component Scanning requires a hook (XML or Java Config)
- Classes in package are scanned for @Component or stereotype of it
- Beans can be injected at Constructor (preferred), field, or method
- Ambiguous definitions require @Qualifier
- JSR 330 Annotations supported

Java Configuration

- Java class based configuration, class annotated with @Configuration
- @Bean defines bean definitions
- Roughly similar to XML, few minor features removed, several gained

Which One?

- XML Config
  - Pros: Detailed blueprint, well documented behavior, clear picture with one stop shop, full control, limit Spring compile time dependencies, works with all code (yours and 3rd party)
  - Cons: “Verbose”, XML based

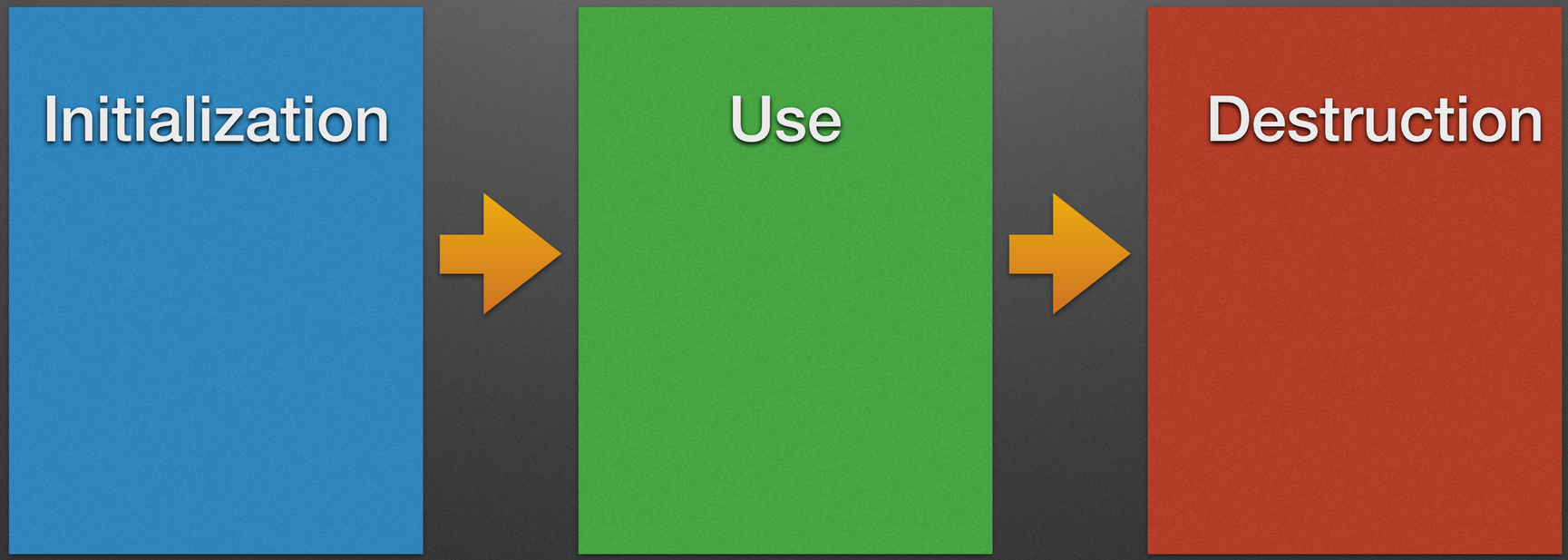
Which One?

- Annotation Based Config
  - Pros: Simple, Required in MVC, little to no configuration
  - Cons: Distributed configuration, less control, litter code with annotations that are for DI only, only works with your code, configuration and code together (Separation of Concerns)

Which One?

- Java Config
  - Pros: Mixes best of both worlds, it is Java, all new examples follow this pattern from spring.io
  - Cons: Lack of abstract configurations, not full DSL support yet for every package

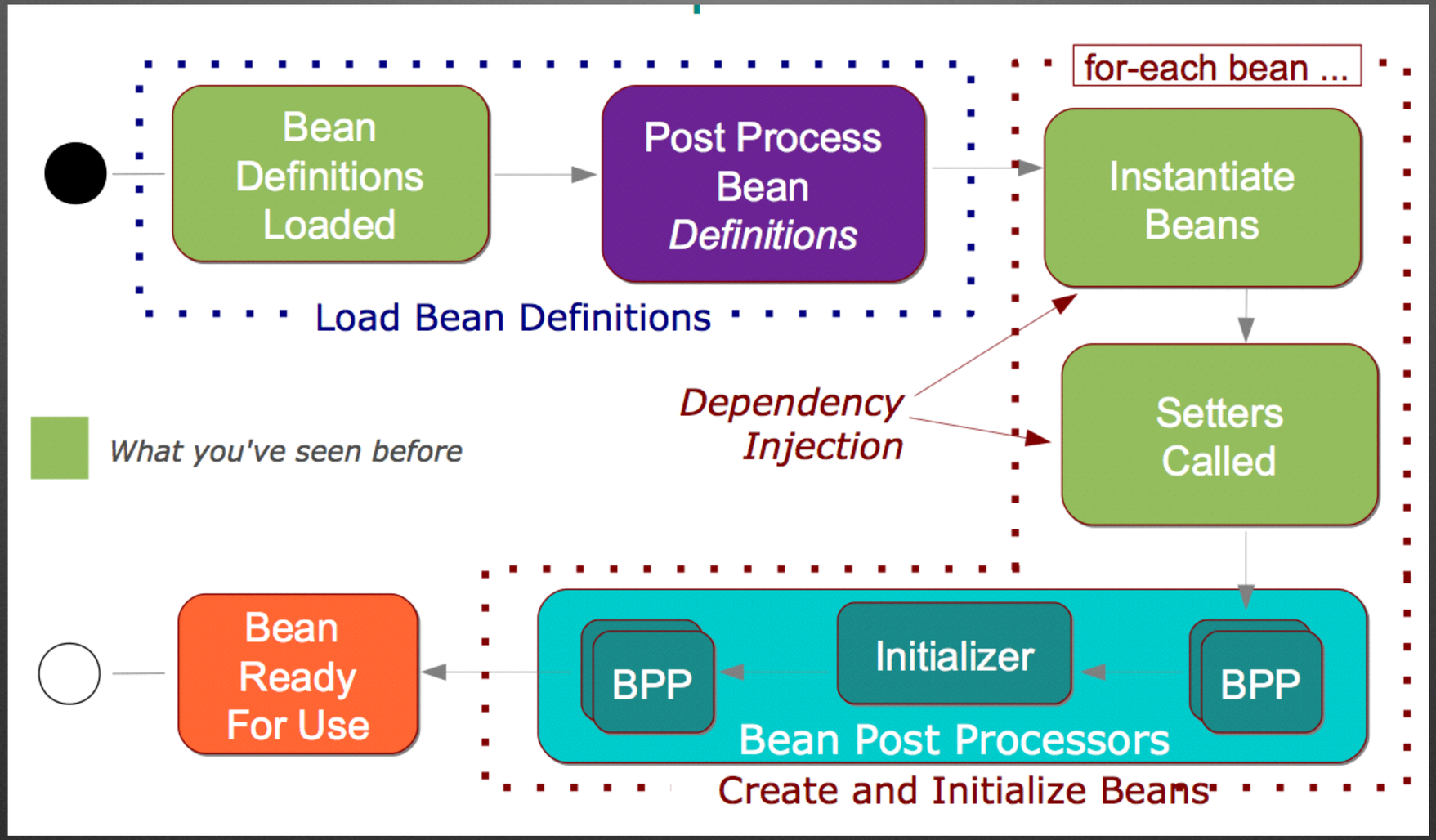
Introduction to the Lifecycle



Initialization Introduction

- Begins with the creation of the Application Context
- Bean Factory Initialization Steps
- Bean Initialization and Instantiation Steps

Overall Picture



Load Bean Definitions

- Starts with creation of ApplicationContext
- Context configures the bean factory and the beans contained in it
- Bean definitions identified via parsing or discovery
- Operations at this phase apply to all beans

Bean Factory Initialization Steps

- Parse Bean Definitions, as appropriate
- Load Bean Definitions into Bean Factory

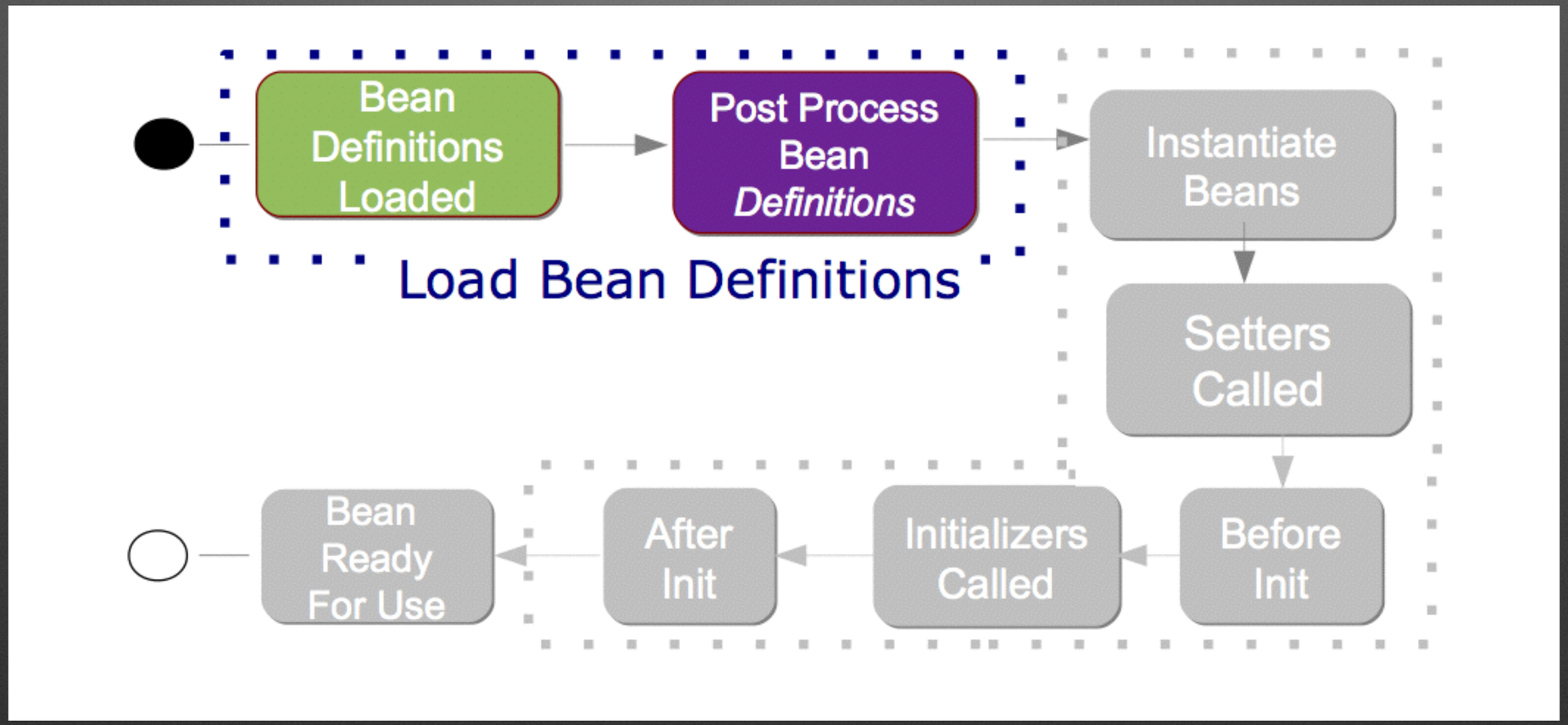
Bean Definitions Loaded

Bean Factory Initialization Steps (Cont)

- Post Processing of Bean Definitions occurs here
- First major extension point -> BeanFactoryPostProcessor Interface
  - Most Common one used = PropertyPlaceholderConfigurer

Post Process
Bean Definitions

Bean Factory Initialized



What has happened so far?

- Beans definitions have been loaded into factory, indexed by ID not name
  - Component Scanning
  - XML Parsing
  - Java Config class loading and parsing
- Beans have been Initialized

Bean Creation Operations

- Occurs after bean factory is configured
- Occurs on each bean in the factory, beans definitions already in heap
- Bean instances are created and maintained by the container for use

Instantiate Beans

- Beans are Instantiated -> singleton instances
- Dependency graph was identified during bean factory phase, used to determine order of instantiation
- Constructor injections occurs here, since the constructor is used :)

Instantiate Beans

Soap Box Time

- Notice that at this phase, dependencies that are part of the constructor are injected, not properties or autowired dependencies*
- Spring promotes good Object Oriented Practices
- If your dependency is required for your bean to operate, it should be injected in the constructor
- Properties (setters or autowired) should be used for optional or conditional dependencies only

Prototypes

- Take note that prototype beans are not instantiated at this point by default
- Only instantiated when needed, as such they live as initialized instances on the heap, but not instantiated
- Lazy loaded beans are not prototypes, they are not instantiated until needed but once they are defined they are singletons

Setters Called

- After all the beans have been Instantiated, setter methods are called
- Defined by Autowiring setters, autowiring with aspecting, or property definitions in configuration

Setters Called

Bean Post Processors

- There are three “phases” of bean post processing
- These are processing on individual beans, not the factory
- 2 basic types in the three phases, Initializers and all the rest (pre and post)

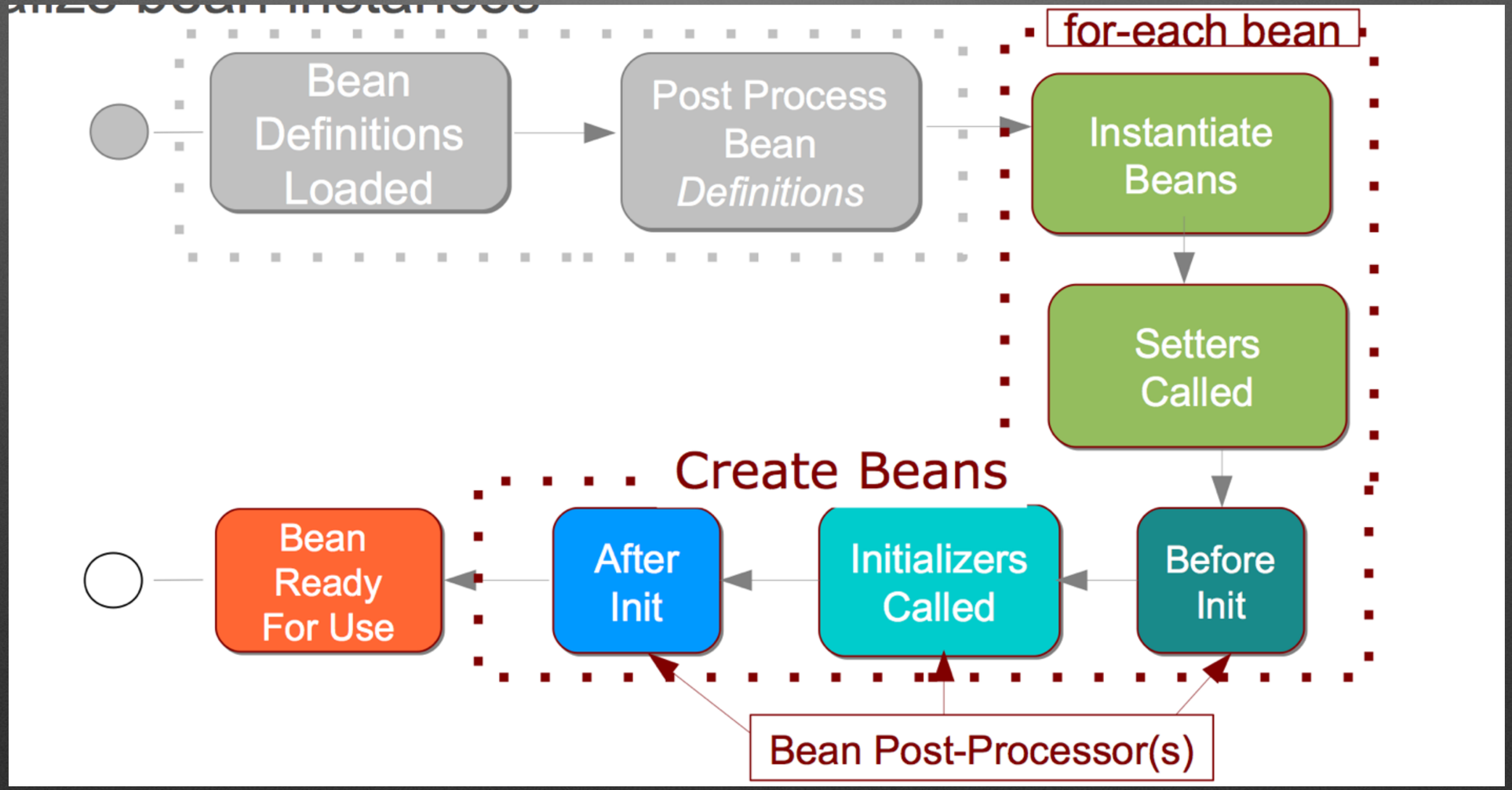
Initializers First
but really Second

- Methods marked with @PostConstruct (1 per class) for annotations
- Methods defined in XML as init-method
- Point to note -> Beans are not ready for use so be very carefully using injected dependencies in the Initializer, proxies may not exist

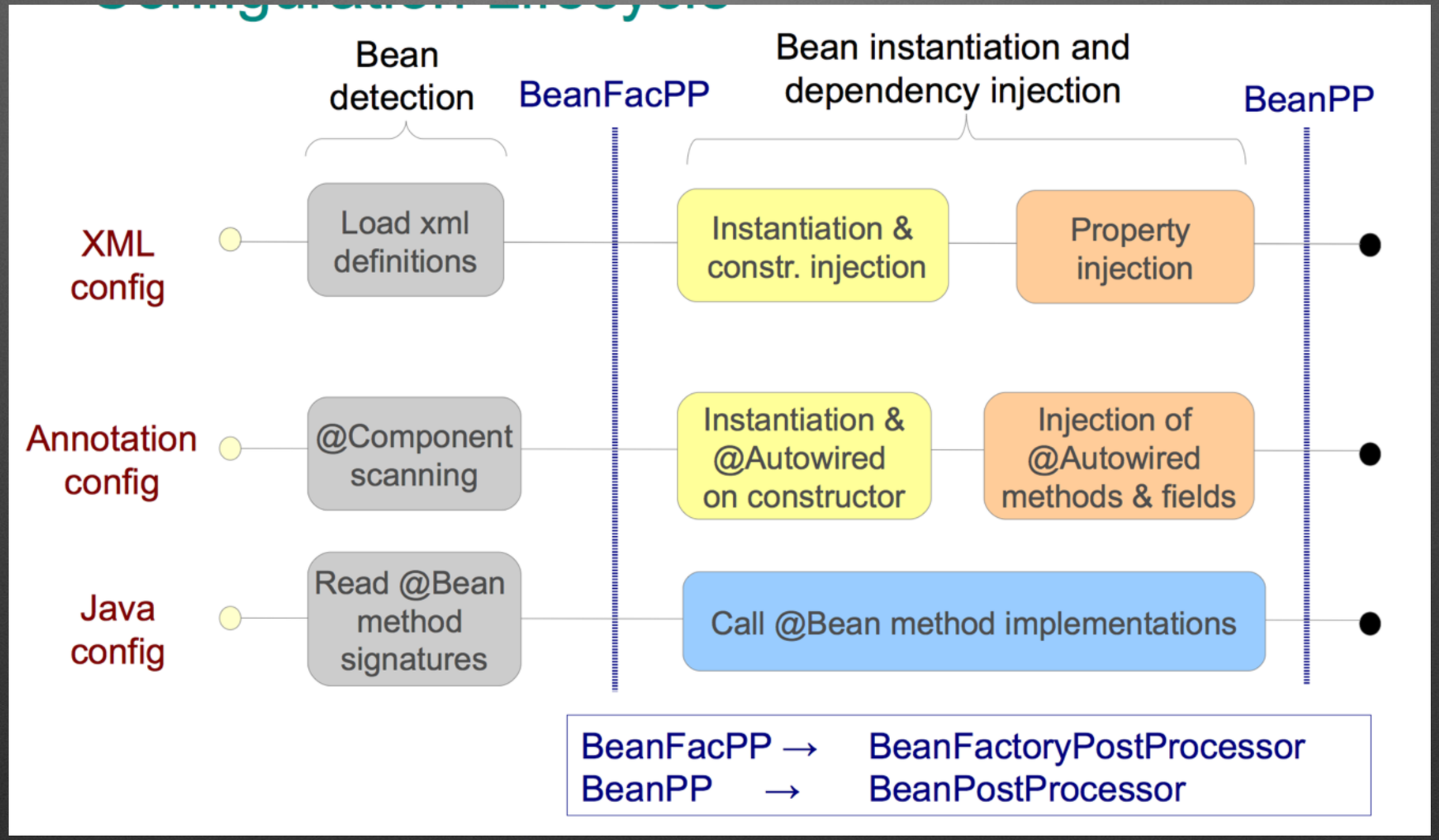
Bean Post Processor

- BeanPostProcessor Interface
  - Supports postProcessAfterInitialization and postProcessBeforeInitialization
- Not a common customer extension point, but used

Beans Ready For Use



Note on Config Differences



Use Phase

- 99% of the time is spent in this phase
- Beans can be accessed directly from the ApplicationContext or via Dependency Inject
  - ApplicationContextAware interface allows a class controlled by Spring to have a handle on the container it is in

Proxies and more Proxies

- Proxies are created in the init phase by dedicated Spring Framework BeanPostProcessors
- With 4.0, every bean gets a proxy, no longer based solely on implicit aspects or annotations
- Need to be aware of the implications of calling self with proxies

JDK Proxy vs CGLib Proxy

- JDK or dynamic proxies are built into JDK
  - Requires a Java Interface
- CGLib Proxies require CGLib to be on the class path
  - Used when no interface is available
  - Cannot be applied to final classes or methods!!
- <soapBox> Use Interfaces </soapBox>

Common Gotchas in Use Phase

- Objects created with the new keyword cause havoc
  - Not expecting object to stick around
  - Leave the application context without meaning to
- Understanding proxies, this is really important

Destruction

- When the context is closed or goes out of scope, there is a shutdown process
- Remember, only Garbage Collection can actually destroy the beans
- Interaction with the destruction phase can be important for closing connections safely or cleaning up other areas

Destroy Methods

- Methods annotated with @PreDestroy
- Methods listed as destroy-method (destroyMethod in Java Config)
  - method must be no-arg void method
- Destroy Method(s) is/are executed then the context itself is destroyed
- Garbage Collection will then clean it all up

Questions?