Spring and Coherence Recipes

David Whitmarsh, Phil Wheeler

December 4, 2013

Outline

Outline

Benefits of IoC Frameworks

- “Pluggability”
- Testability
- Configurability

SpringAwareCacheFactory, Coherence Spring Integration

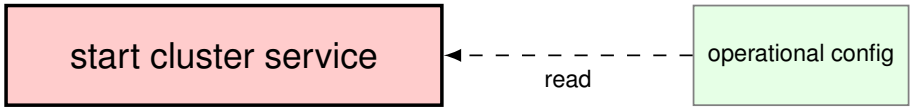
	SpringAware-CacheFactory	Spring Integration project
Version	to 3.7	12.1.2
Uses incubator	? Yes/No/Maybe	× No
Inject beans in cache-config	✓ Yes	✓ Yes
Set placeholders as bean properties	✓ Yes	✓ Yes
Pre-instantiate ApplicationContext	✓ Yes	× No

Outline

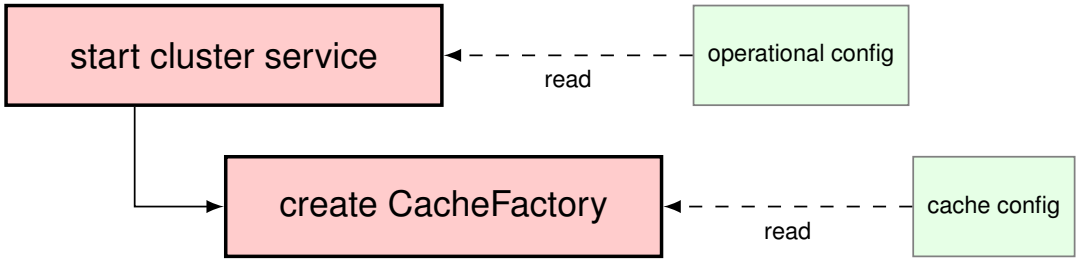
Static Initialisation

- Coherence cluster startup conditions ill-defined
- Unit testing is harder
- No way to access <class-scheme> objects

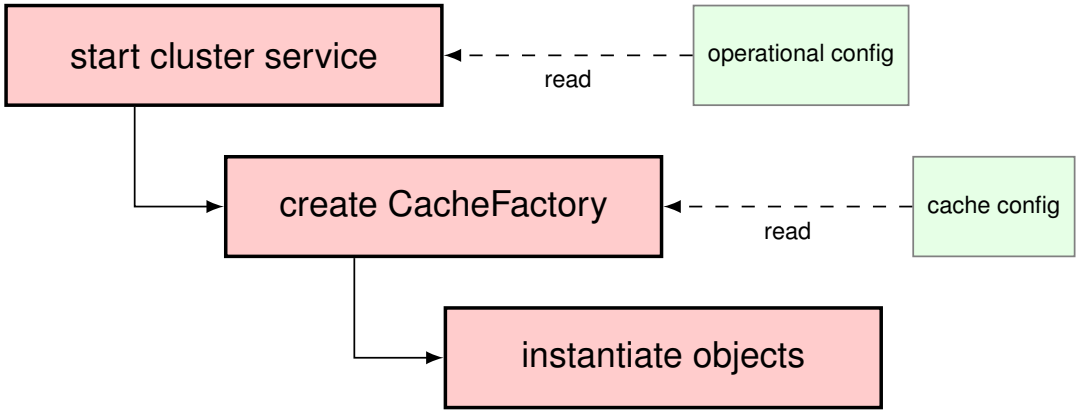
Coherence Node Startup



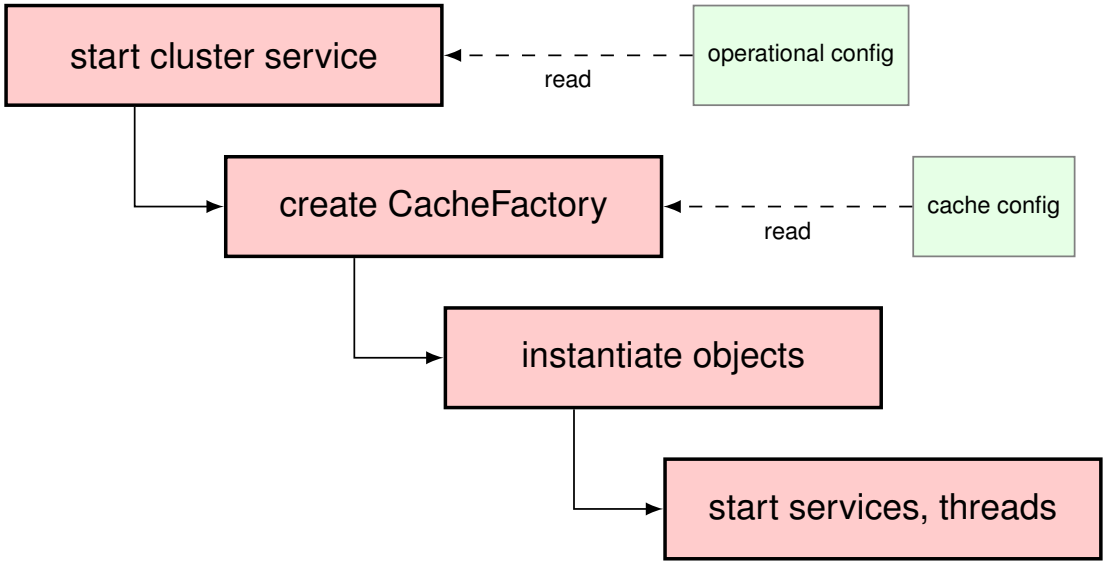
Coherence Node Startup



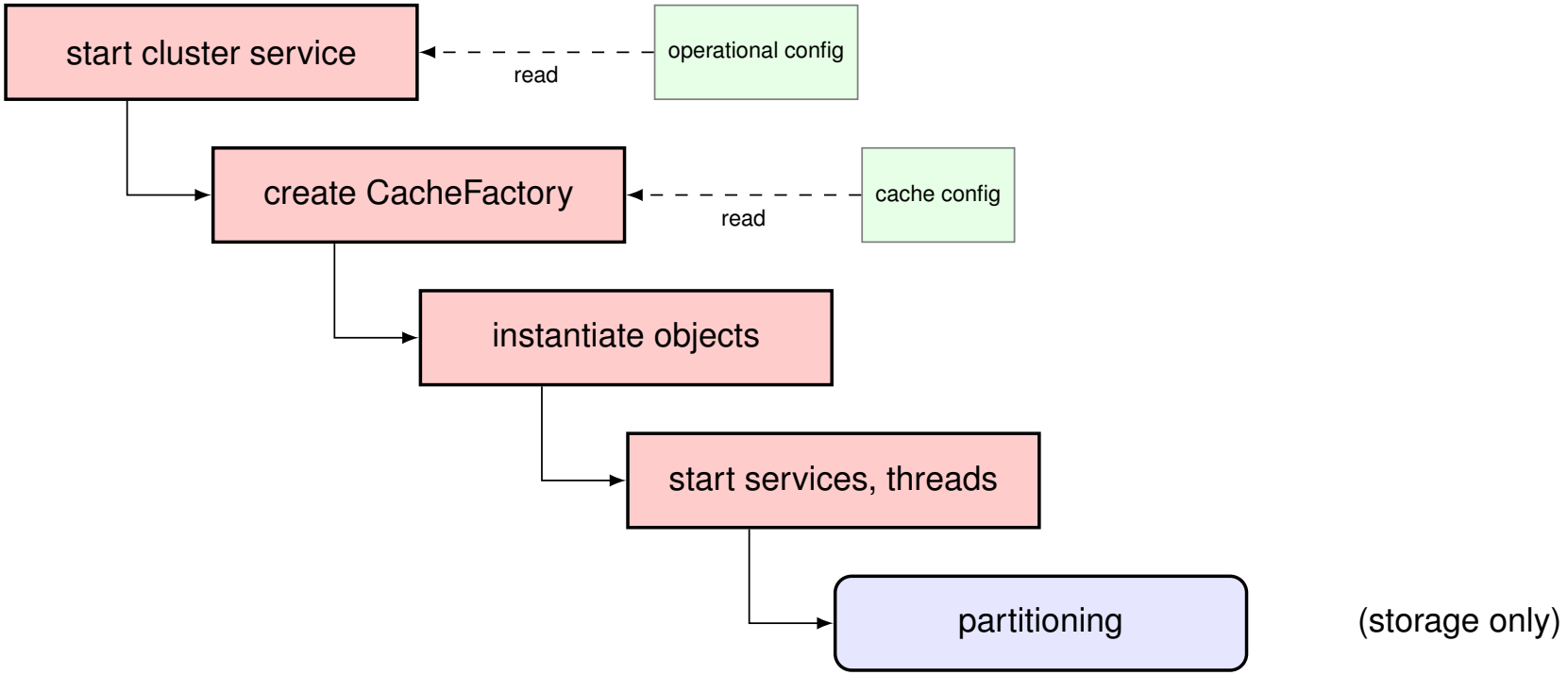
Coherence Node Startup



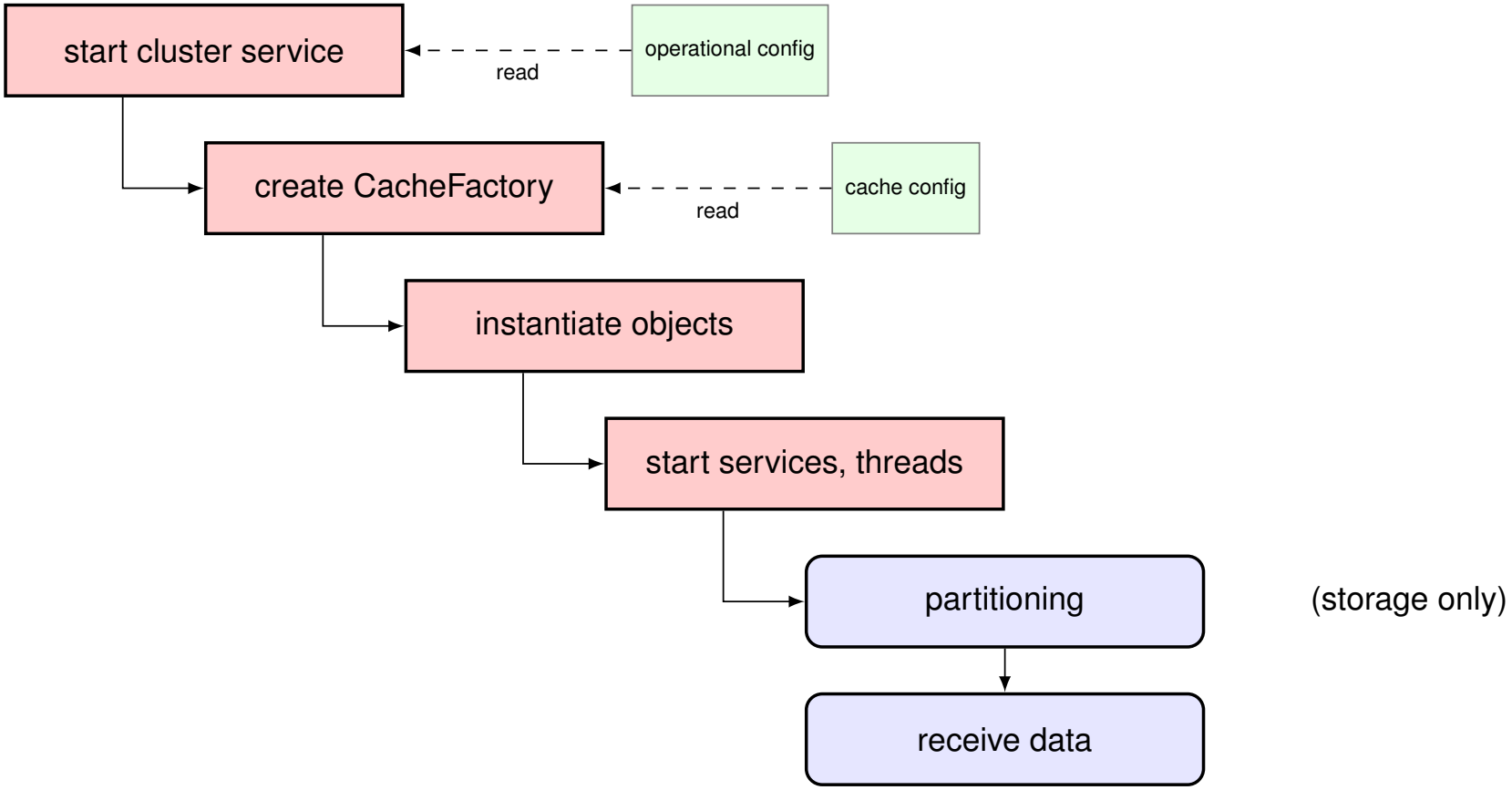
Coherence Node Startup



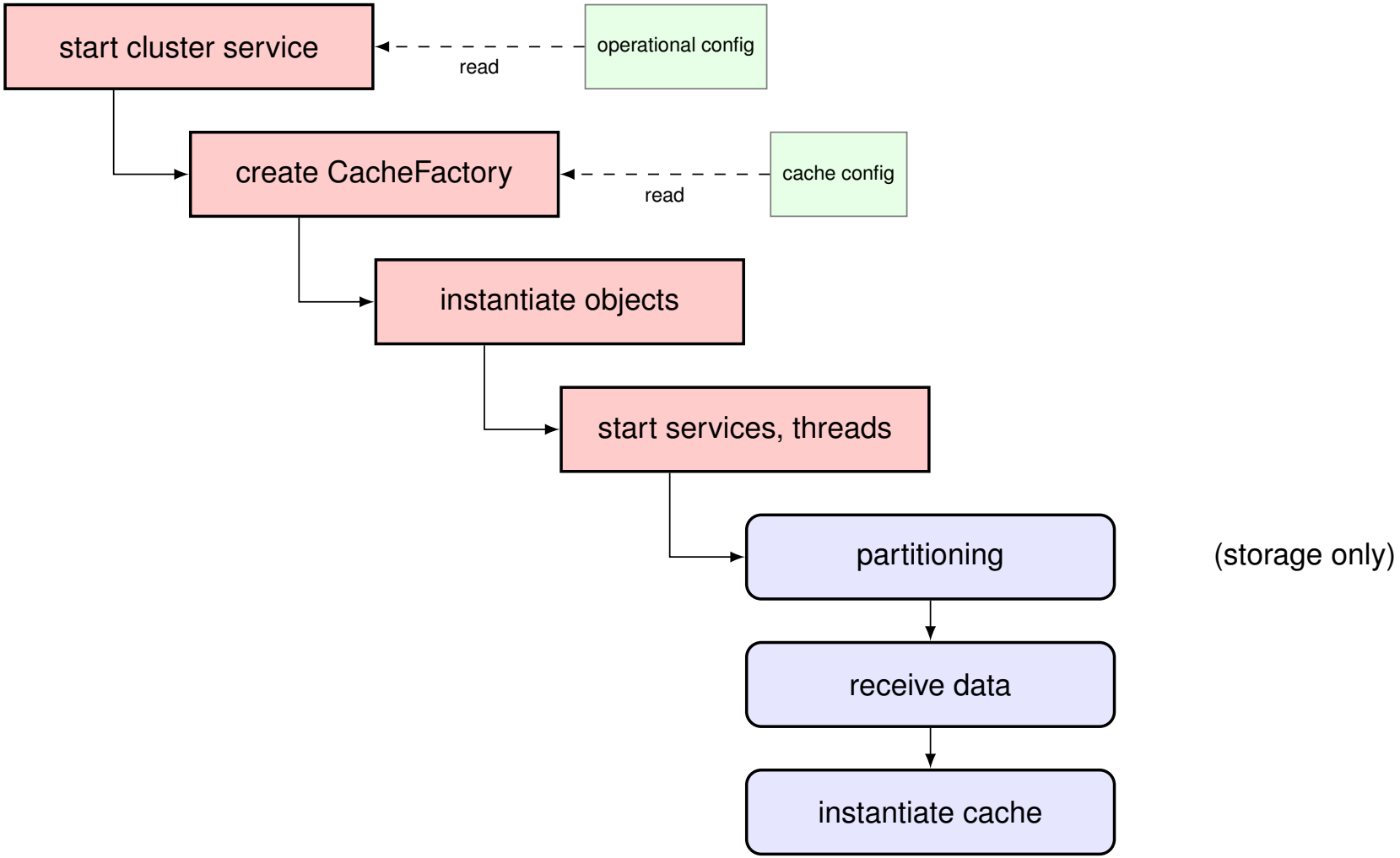
Coherence Node Startup



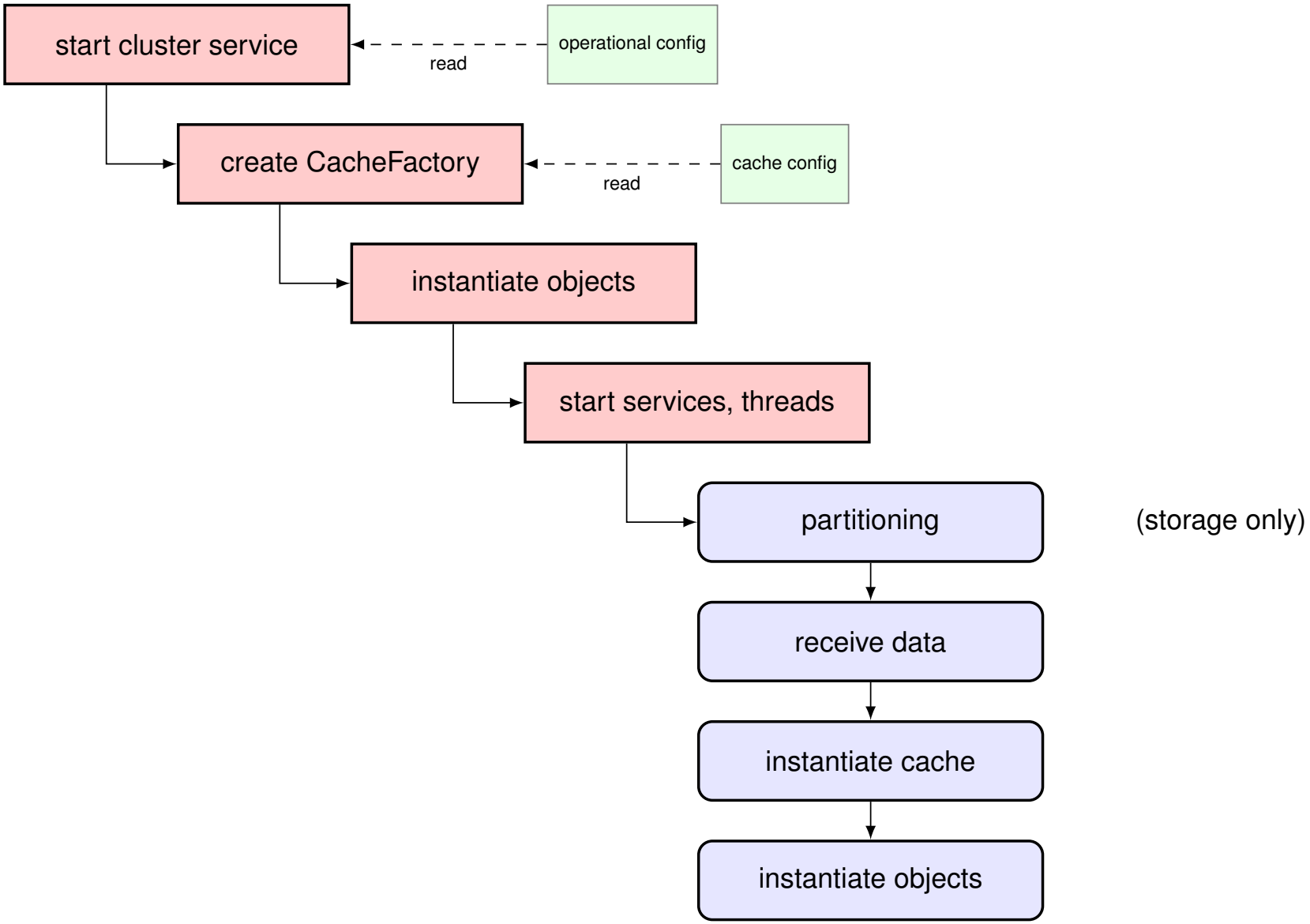
Coherence Node Startup



Coherence Node Startup



Coherence Node Startup

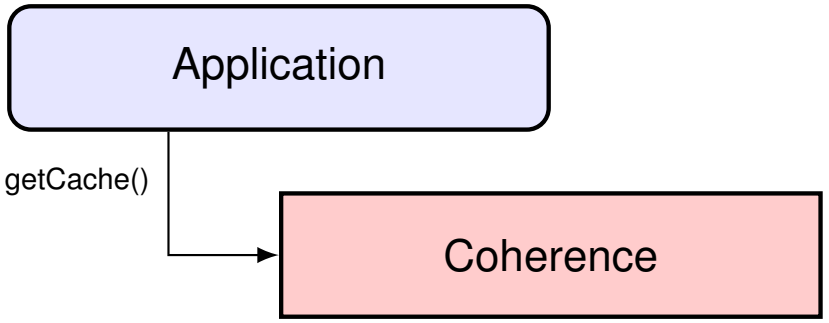


Circular Dependencies

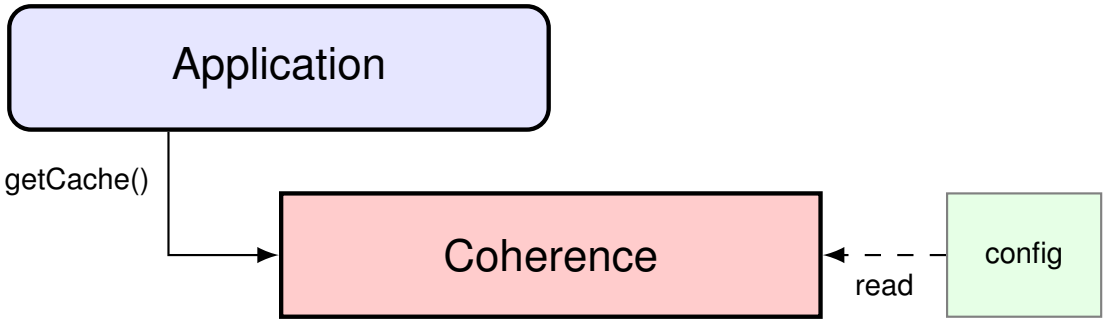
Explosion risk

Circular Dependencies

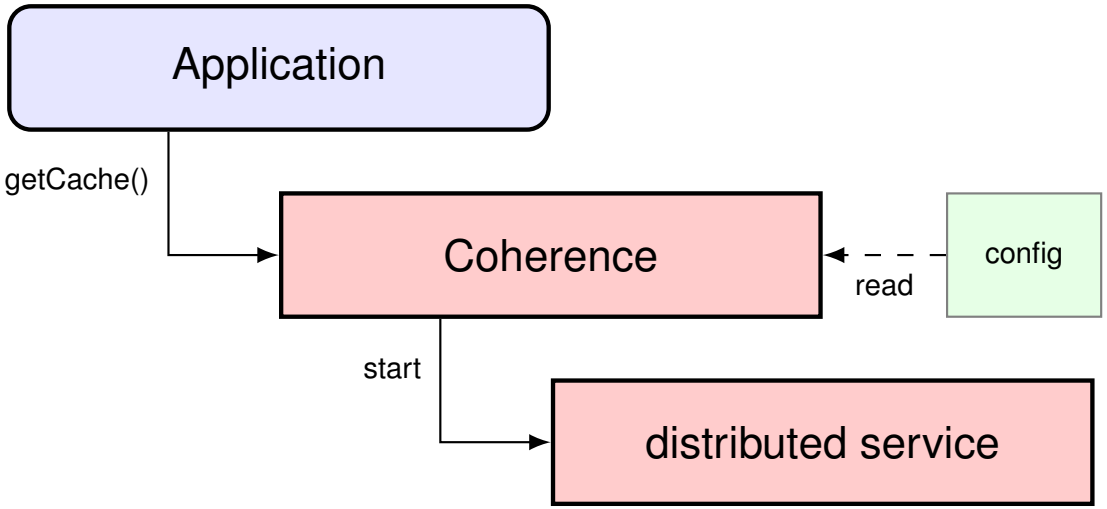
Circular Dependencies



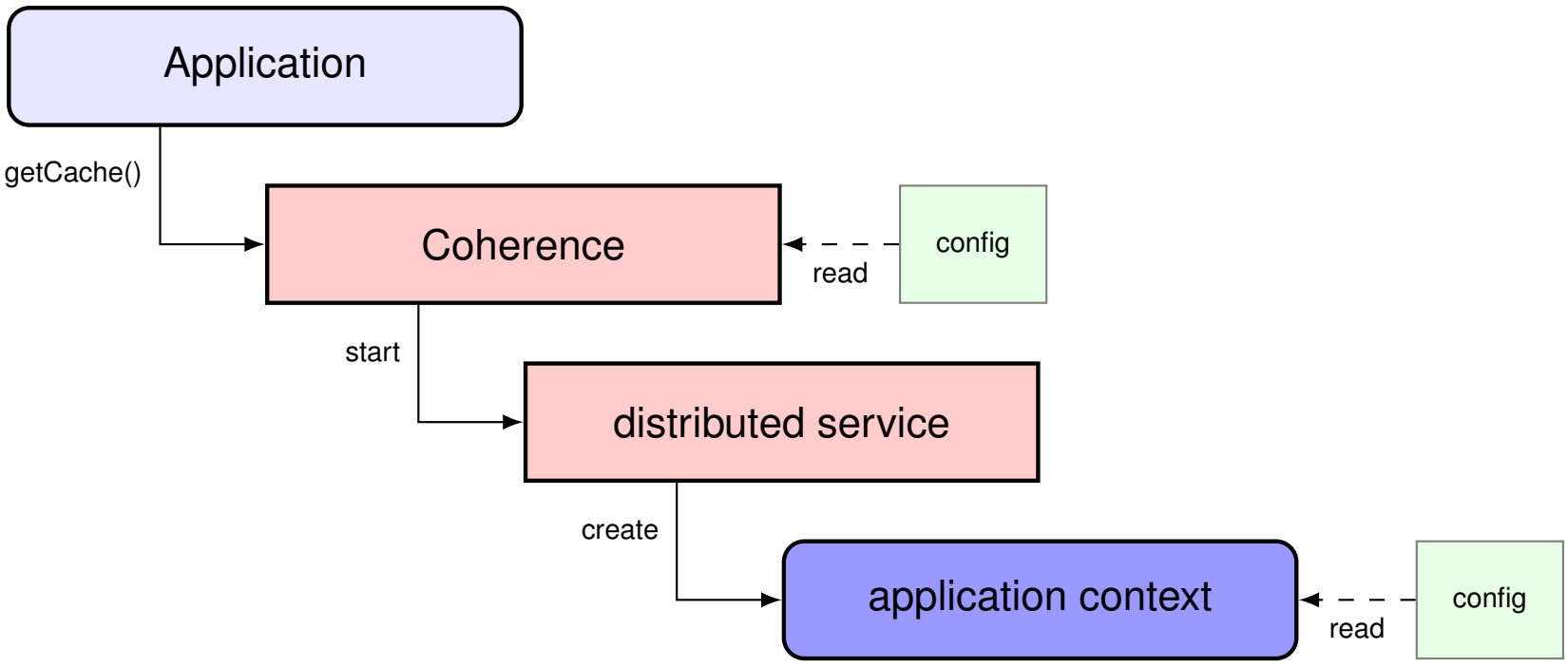
Circular Dependencies



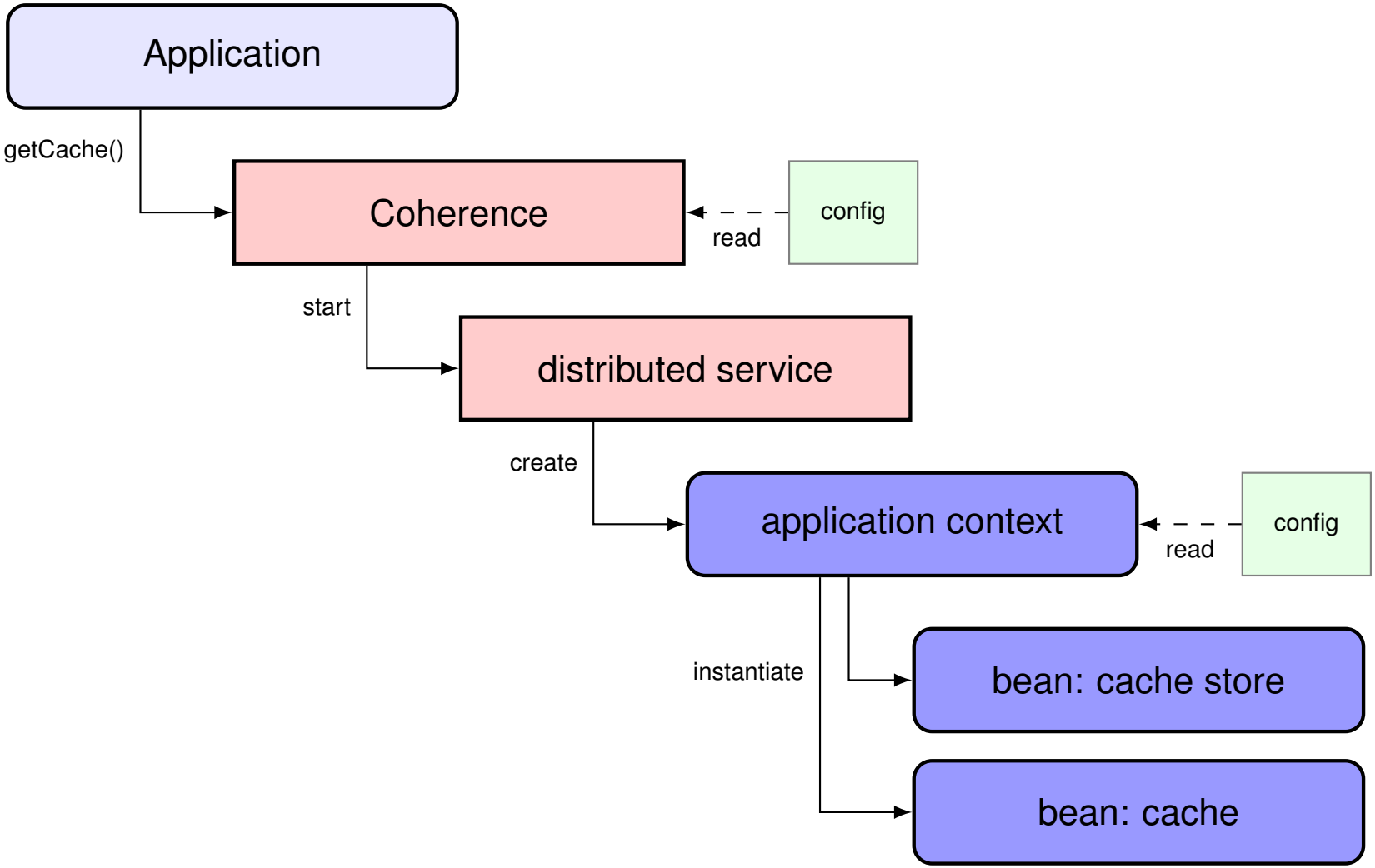
Circular Dependencies



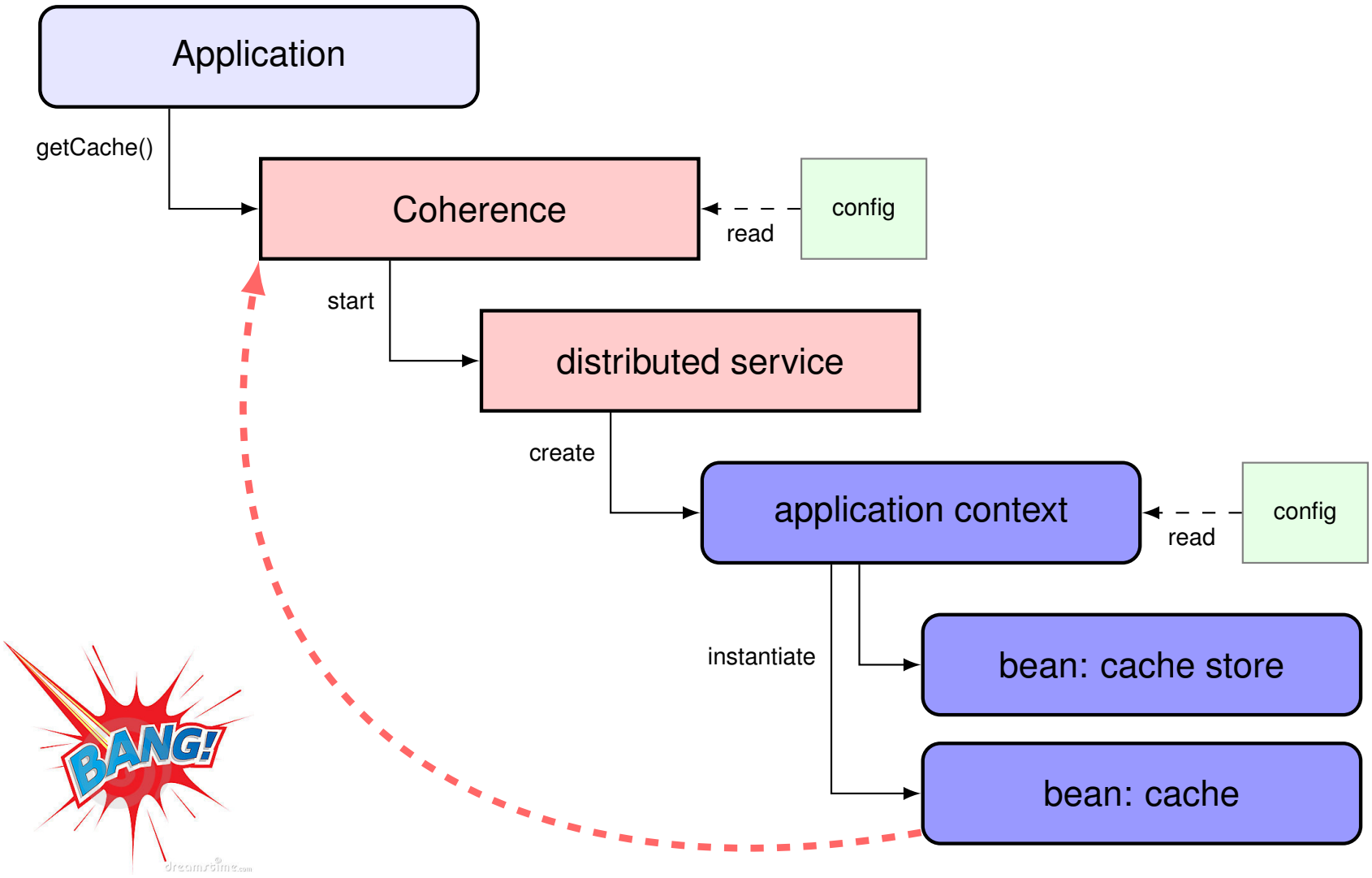
Circular Dependencies

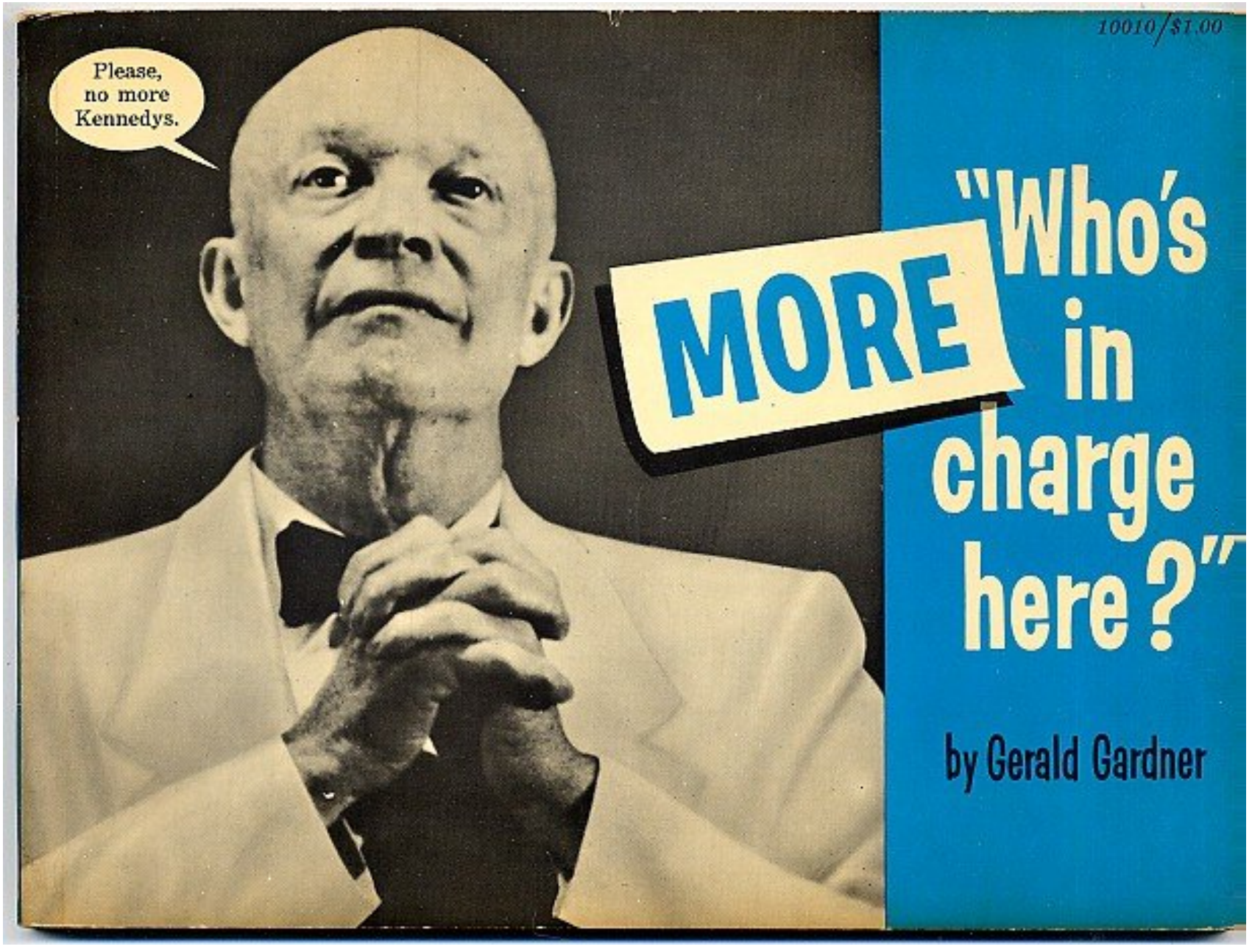


Circular Dependencies



Circular Dependencies




10010/$1.00
Please, no more Kennedys.
MORE "Who's in charge here?"
by Gerald Gardner

Who's in charge?

- Spring and Coherence both try to control instantiation

Who's in charge?

- Spring and Coherence both try to control instantiation
- Which one is designed as an IoC framework?

Who's in charge?

- Spring and Coherence both try to control instantiation
- Which one is designed as an IoC framework?
- Keep the framework in control ✓

Solutions?

- lazy instantiation?
- lazy initialisation?
- explicit dependencies?
- lifecycle?
- separate contexts

Outline

Objects used by Coherence

- Beans needed by Coherence during startup
- Beans needed as caches are repartitioned
- May implement Coherence interfaces
- May not call Coherence APIs during instantiation or initialisation

CoherenceBeanContext - these can and should be created before starting Coherence

Objects that use Coherence

- Beans that reference Coherence
- Application logic
  - JMS readers, message processing, etc
  - Embedded web server
- May call Coherence APIs during instantiation or initialisation
- May be empty or absent for storage, proxy, etc nodes

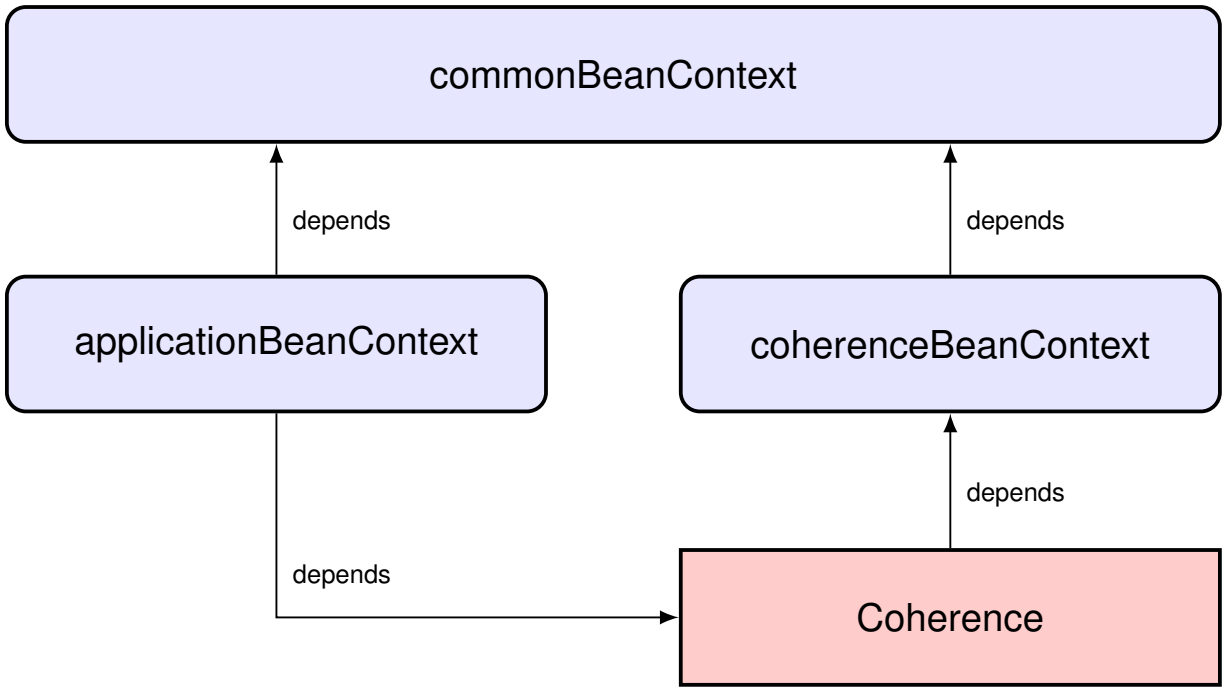
ApplicationBeanContext - these cannot be created until Coherence is available

CommonObjects

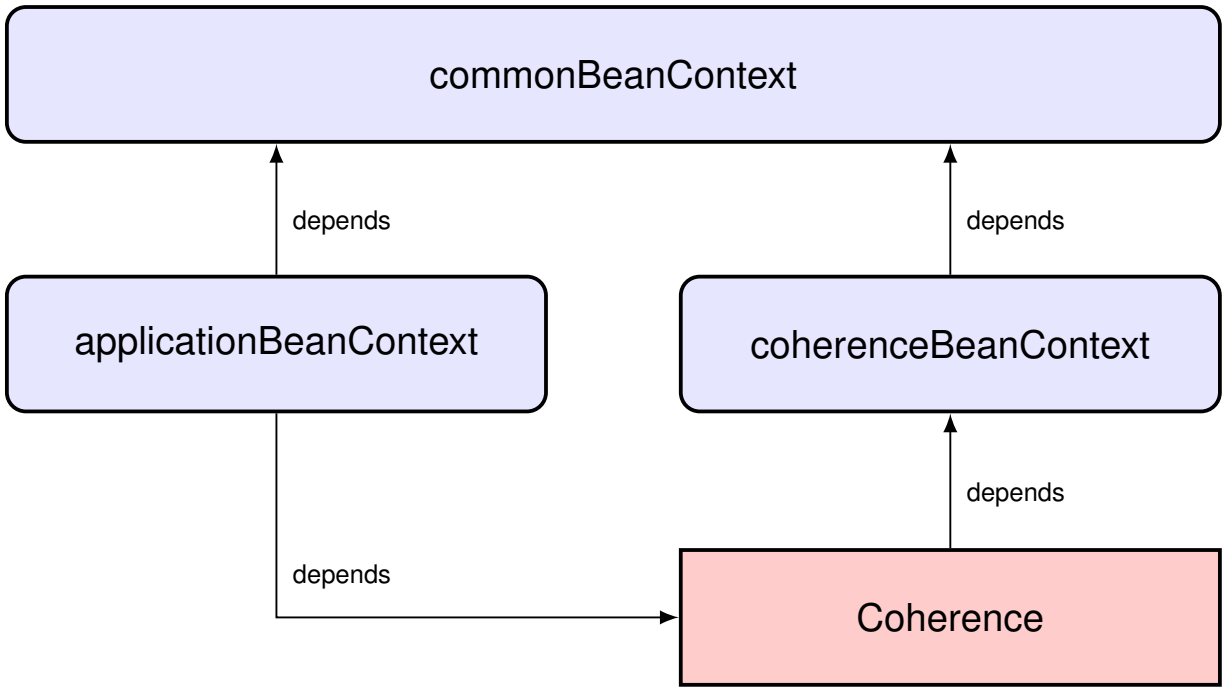
- Beans reference by Coherence and Application
- Common resources
  - DataSources
  - JMS Connections
  - etc
- May not call Coherence APIs during instantiation or initialisation
- Parent context of CoherenceBeanContext and ApplicationBeanContext

CommonBeanContext

Contexts & initialisation order

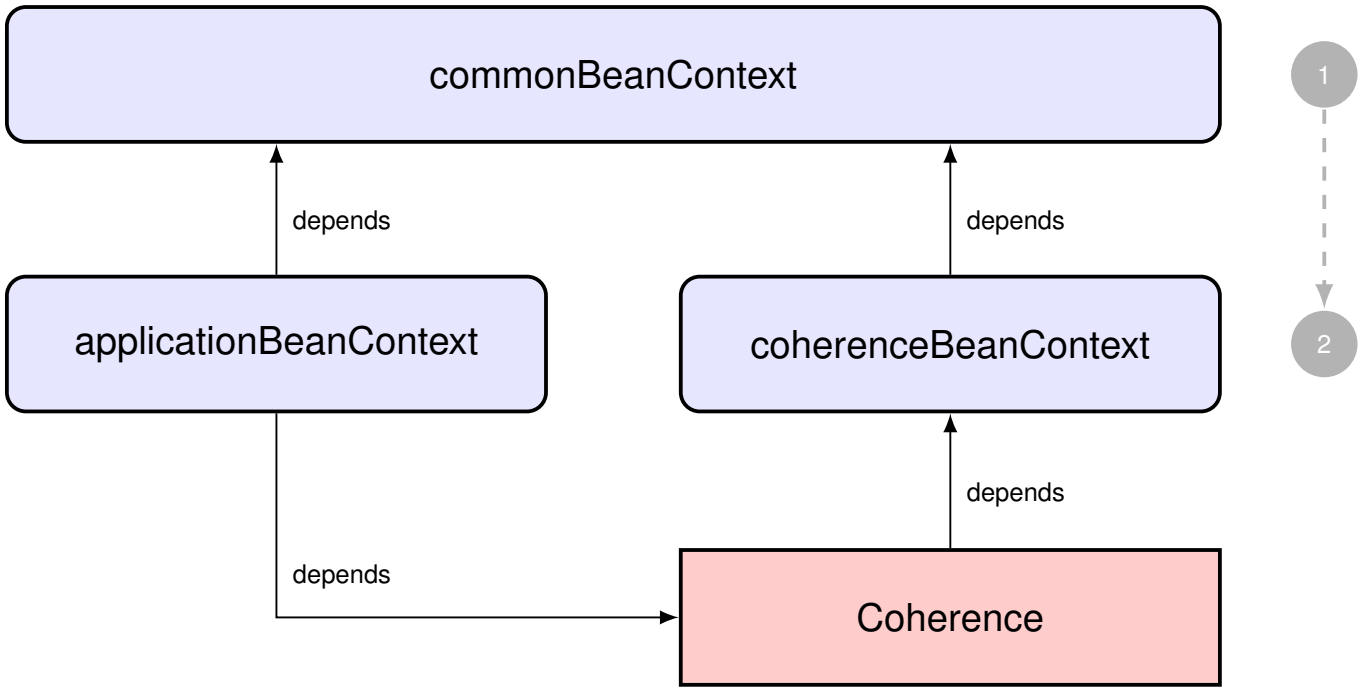


Contexts & initialisation order



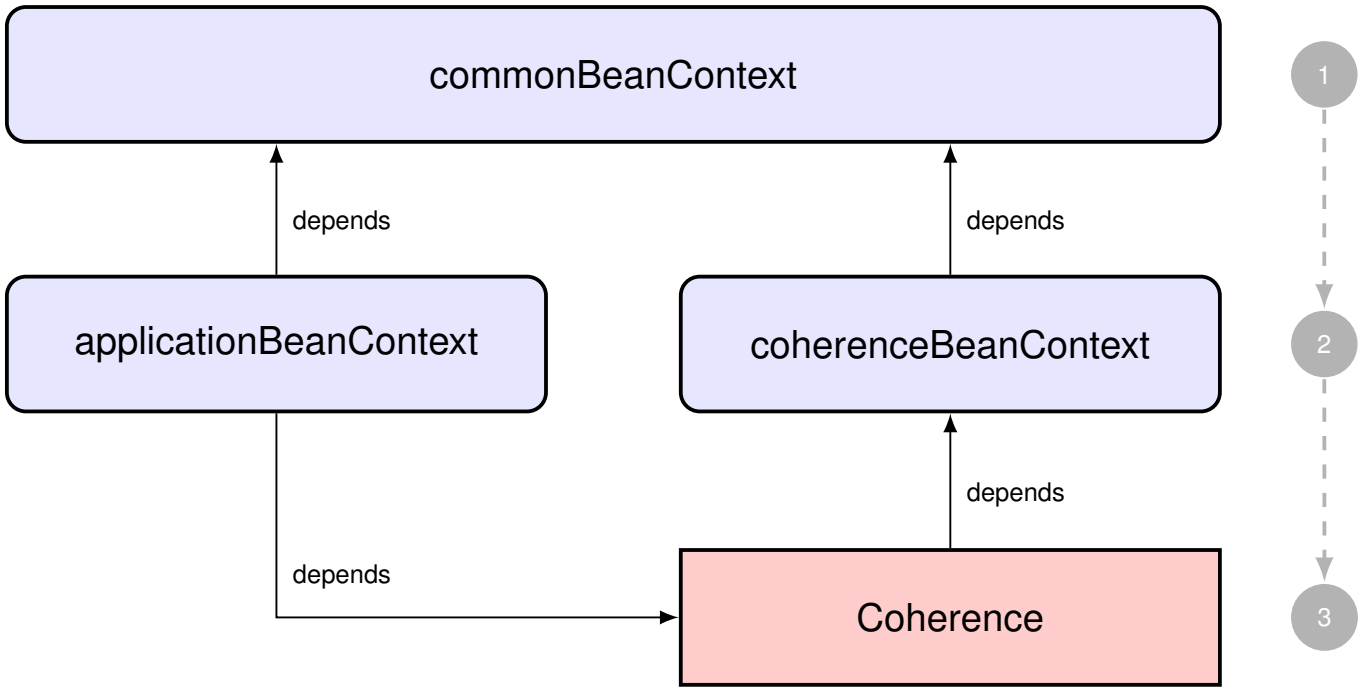
x order of initialisation

Contexts & initialisation order



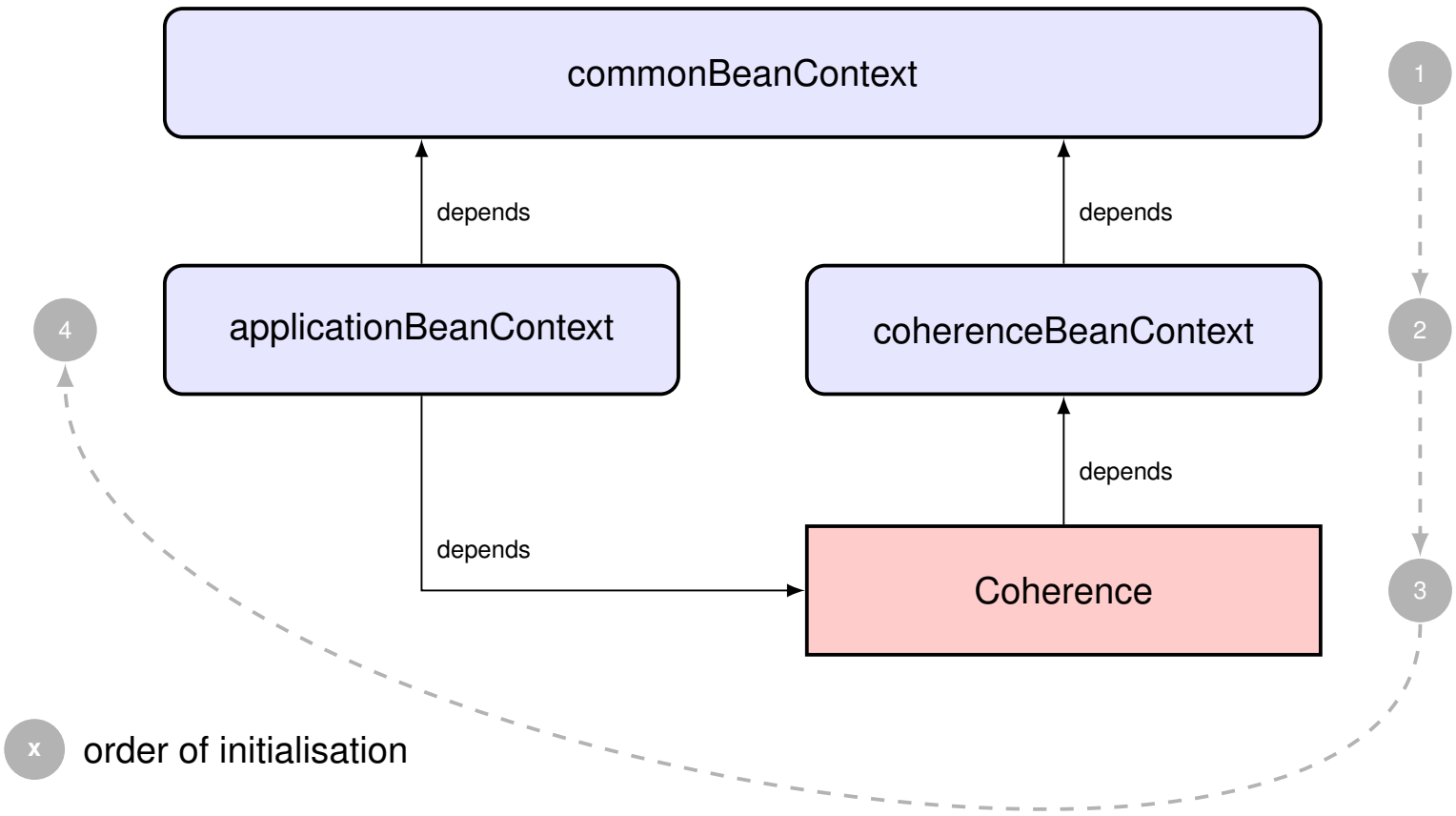
x order of initialisation

Contexts & initialisation order



x order of initialisation

Contexts & initialisation order



Prevent Premature Instantiation

```
public class ValidatingCacheFactoryBuilder extends
        DefaultCacheFactoryBuilder {

    private static boolean buildOk = false;

    @Override
    public ConfigurableCacheFactory getConfigurableCacheFactory(
            ClassLoader loader) {
        if (!buildOk) {
            throw new IllegalStateException("Attempt to build a
                cache factory too early");
        }
        return super.getConfigurableCacheFactory(loader);
    }

    static void enableBuild() {
        buildOk = true;
    }
}
```

Define the contexts

Use a master context: masterContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

   <bean id="commonBeanContext"
         class="org.springframework.context.support.ClassPathXmlApplicationContext">
     <constructor-arg><value>UtilBeanContext.xml</value></constructor-arg>
   </bean>
```

Define the contexts

Use a master context: masterContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

   <bean id="commonBeanContext"
         class="org.springframework.context.support.ClassPathXmlApplicationContext">
     <constructor-arg><value>UtilBeanContext.xml</value></constructor-arg>
   </bean>

   <bean id="coherenceBeanContext"
         class="org.springframework.context.support.ClassPathXmlApplicationContext">
     <constructor-arg>
       <list><value>CoherenceBeanContext.xml</value></list>
     </constructor-arg>
     <constructor-arg>
       <ref bean="commonBeanContext"/>
     </constructor-arg>
   </bean>
```

Define the contexts

Use a master context: masterContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

   <bean id="commonBeanContext"
         class="org.springframework.context.support.ClassPathXmlApplicationContext">
     <constructor-arg><value>UtilBeanContext.xml</value></constructor-arg>
   </bean>

   <bean id="coherenceBeanContext"
         class="org.springframework.context.support.ClassPathXmlApplicationContext">
     <constructor-arg>
       <list><value>CoherenceBeanContext.xml</value></list>
     </constructor-arg>
     <constructor-arg>
       <ref bean="commonBeanContext"/>
     </constructor-arg>
   </bean>

   <bean id="applicationBeanContext" lazy-init="true"
         class="org.springframework.context.support.ClassPathXmlApplicationContext">
     <constructor-arg>
       <list><value>ApplicationBeanContext.xml</value></list>
     </constructor-arg>
     <constructor-arg>
       <ref bean="commonBeanContext"/>
     </constructor-arg>
   </bean>
</beans>
```

Access a Bean

```
public class BeanLocator {

    private static ApplicationContext applicationContexts =
        new ClassPathXmlApplicationContext(
            "classpath:masterContext.xml");

    public static BeanFactory getContext(String contextName) {
        return (BeanFactory)
            applicationContexts.getBean(contextName);
    }

    public static Object getBean(String contextName,
        String beanName) {
        return getContext(contextName).getBean(beanName);
    }
}
```

Reference a Bean

```
<cachestore-scheme>
    <class-scheme>
        <class-factory-name>
            org.cohbook.configuration.spring.BeanLocator
        </class-factory-name>
        <method-name>getBean</method-name>
        <init-params>
            <init-param>
                <param-value>coherenceBeanContext</param-value>
            </init-param>
            <init-param>
                <param-value>exampleCacheLoader</param-value>
            </init-param>
        </init-params>
    </class-scheme>
</cachestore-scheme>
```

Starting a Node

```
public static void main(String[] args) {

    loadCoherenceProperties();

    BeanLocator.getContext("coherenceBeanContext");

    CacheFactory.ensureCluster();

    BeanFactory bf =
        BeanLocator.getContext("applicationBeanContext");

    // Application logic

}
```

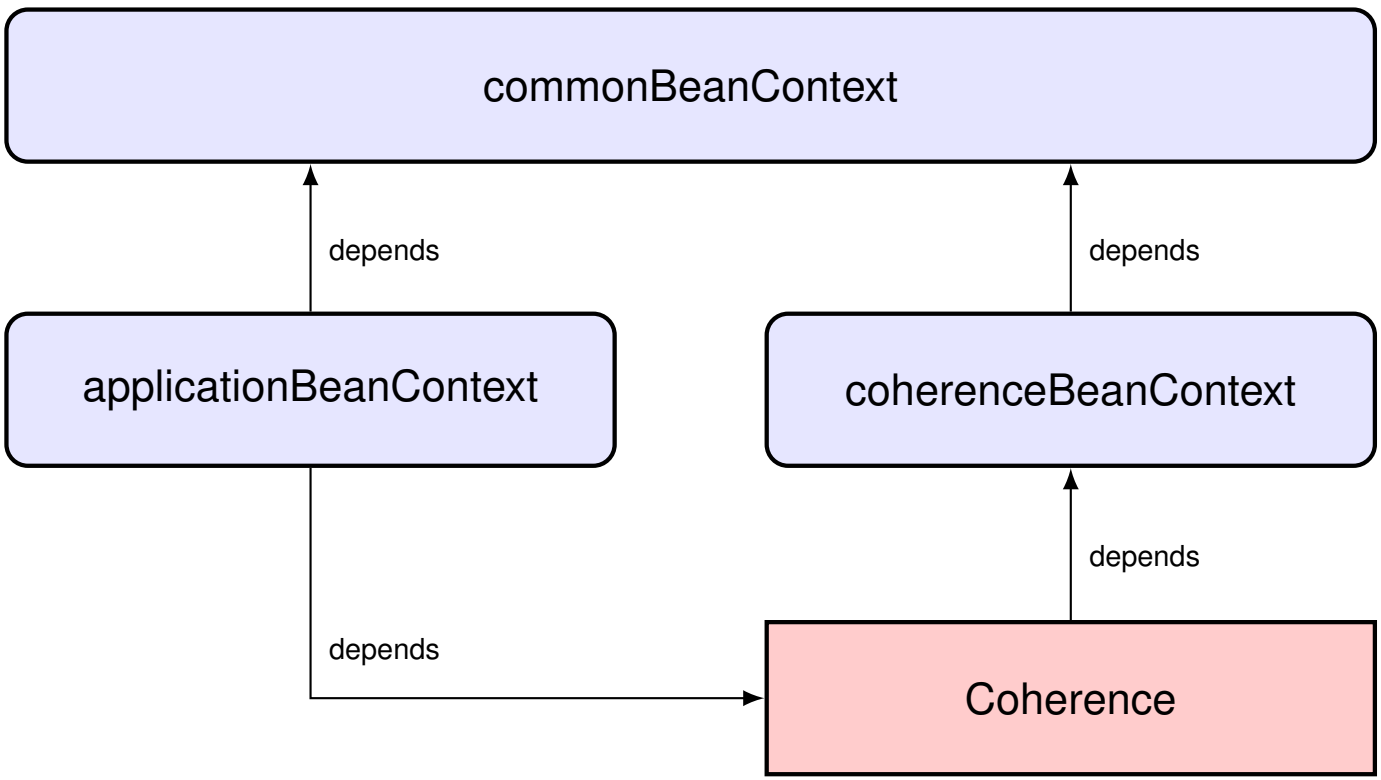
Outline

Circular dependency

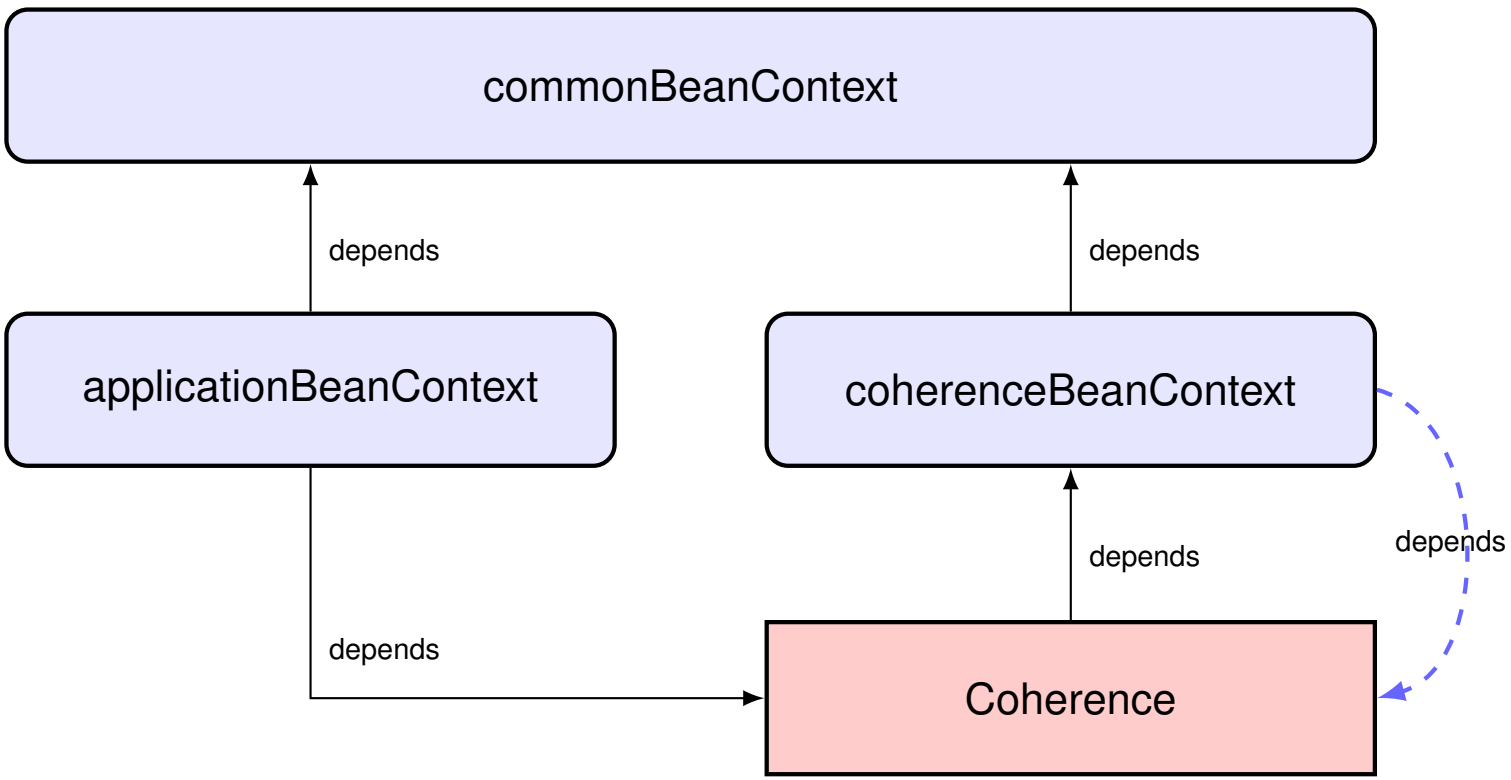
Problem: What if a bean that Coherence depends on, also depends on Coherence?

e.g. A Controllable CacheStore that references a replicated control cache to determine whether to persist.

Circular Dependency 2



Circular Dependency 2



Using Spring SmartLifeCycle

- After Coherence beans instantiated and initialised
- Permit Coherence to start
- Start Coherence
- Implement start method in beans with circular dependencies

Permit Coherence to Start

```
public static class BuilderLifeCycle implements SmartLifecycle {

    public static final int BEFORE_CLUSTER_PHASE = 100;
    public static final int CLUSTER_PHASE = 200;
    public static final int AFTER_CLUSTER_PHASE = 300;

    public void start() {
        ValidatingCacheFactoryBuilder.enableBuild();
        CacheFactory.ensureCluster();
    }

    public int getPhase() {
        return CLUSTER_PHASE;
    }

    public boolean isAutoStartup() {
        return true;
    }
    .
    .
    .
}
```

Starting a Node with SmartLifecycle

. . . In the Coherence beans context . . .

```
<bean class="...BuilderLifeCycle"/>
```

Starting a Node with SmartLifecycle

. . . In the Coherence beans context . . .

```
<bean class="...BuilderLifeCycle"/>
```

. . . The main method . . .

```
public static void main(String[] args) {

    loadCoherenceProperties();

    BeanLocator.getContext("coherenceBeanContext").start();

    BeanFactory bf =
        BeanLocator.getContext("applicationBeanContext");

    // Application logic

}
```

Outline

Recipe: Avoid static Coherence Methods

```
<bean id="cacheFactory"
    class="com.tangosol.net.CacheFactory"
    factory-method="getConfigurableCacheFactory"/>
```

Recipe: Avoid static Coherence Methods

```
<bean id="cacheFactory"
    class="com.tangosol.net.CacheFactory"
    factory-method="getConfigurableCacheFactory"/>

<bean id="exampleCache" factory-bean="cacheFactory"
    factory-method="ensureCache">
    <constructor-arg value="test"/>
    <constructor-arg><null/></constructor-arg>
</bean>
```

Recipe: Avoid static Coherence Methods

```
<bean id="cacheFactory"
    class="com.tangosol.net.CacheFactory"
    factory-method="getConfigurableCacheFactory"/>

<bean id="exampleCache" factory-bean="cacheFactory"
    factory-method="ensureCache">
    <constructor-arg value="test"/>
    <constructor-arg><null/></constructor-arg>
</bean>

<bean id="distributedService" factory-bean="cacheFactory"
    factory-method="ensureService">
    <constructor-arg value="exampleDistributedService"/>
</bean>
```

Recipe: Avoid static Coherence Methods

```
<bean id="cacheFactory"
    class="com.tangosol.net.CacheFactory"
    factory-method="getConfigurableCacheFactory"/>

<bean id="exampleCache" factory-bean="cacheFactory"
    factory-method="ensureCache">
    <constructor-arg value="test"/>
    <constructor-arg><null/></constructor-arg>
</bean>

<bean id="distributedService" factory-bean="cacheFactory"
    factory-method="ensureService">
    <constructor-arg value="exampleDistributedService"/>
</bean>

<bean id="cluster" class="com.tangosol.net.CacheFactory"
    factory-method="getCluster"/>
```

Configuration Macro 1

```
<cachestore-scheme>
    <class-scheme>
        <class-factory-name>
            org.cohbook.configuration.spring.BeanLocator
        </class-factory-name>
        <method-name>getBean</method-name>
        <init-params>
            <init-param><param-value>coherenceBeanContext</param-value></init-param>
            <init-param><param-value>dynamicCacheLoader</param-value></init-param>
            <init-param><param-value>cacheName</param-value></init-param>
            <init-param><param-value>{cache-name}</param-value></init-param>
        </init-params>
    </class-scheme>
</cachestore-scheme>
```

Configuration Macro 2

```
public class BeanLocator {

    ...

    public static Object getBean(
            String contextName, String beanName,
            String propertyName, Object
            propertyValue) {

     if (propertyValue instanceof Value) {
            propertyValue = ((Value)propertyValue).get();
        }
        Object bean = getContext(contextName).getBean(beanName);
        PropertyAccessor accessor =
            PropertyAccessorFactory.forBeanPropertyAccess(bean);
        accessor.setPropertyValue(propertyName, propertyValue);
        return bean;
    }
}
```

Configuration Macro 3

```
<bean id="dynamicCacheLoader"
        class="org.cohbook.configuration.spring.DynamicCacheLoader"
    scope="prototype">
    <constructor-arg>
        <ref bean="exampleDataSource"/>
    </constructor-arg>
</bean>
```

Injection In Grid Objects 1

- Transported objects may need references to beans
  - Invocable
  - EntryProcessor
  - Filter
  - Aggregator
  - Cache keys/values (if you really must)
- How and when do we inject these values?

Injection In Grid Objects 1

- Transported objects may need references to beans
  - Invocable
  - EntryProcessor
  - Filter
  - Aggregator
  - Cache keys/values (if you really must)
- How and when do we inject these values?
  - During deserialisation!

Injection In Grid Objects 2

```
public class SpringSerializer implements Serializer {

    private final Serializer delegate;
    private final String applicationContextName;
```

Injection In Grid Objects 2

```
public class SpringSerializer implements Serializer {

    private final Serializer delegate;
    private final String applicationContextName;

    public SpringSerializer(String applicationContextName,
        Serializer delegate) {
        this.applicationContextName = applicationContextName;
        this.delegate = delegate;
    }
```

Injection In Grid Objects 2

```
public class SpringSerializer implements Serializer {

    private final Serializer delegate;
    private final String applicationContextName;

    public SpringSerializer(String applicationContextName,
        Serializer delegate) {
        this.applicationContextName = applicationContextName;
        this.delegate = delegate;
    }

    private AutowireCapableBeanFactory getBeanFactory() {
        return BeanLocator.getContext(applicationContextName)
                .getAutowireCapableBeanFactory();
    }
```

Injection In Grid Objects 2

```
public class SpringSerializer implements Serializer {

    private final Serializer delegate;
    private final String applicationContextName;

    public SpringSerializer(String applicationContextName,
        Serializer delegate) {
        this.applicationContextName = applicationContextName;
        this.delegate = delegate;
    }

    private AutowireCapableBeanFactory getBeanFactory() {
        return BeanLocator.getContext(applicationContextName)
                .getAutowireCapableBeanFactory();
    }

    @Override
    public Object deserialize(BufferInput in) throws
        IOException {

        Object result = delegate.deserialize(in);

        getBeanFactory().autowireBeanProperties(result,
           AutowireCapableBeanFactory.AUTOWIRE_BY_TYPE, false);

        return result;
    }
```

Injection In Grid Objects 2

```
public class SpringSerializer implements Serializer {

    private final Serializer delegate;
    private final String applicationContextName;

    public SpringSerializer(String applicationContextName,
        Serializer delegate) {
        this.applicationContextName = applicationContextName;
        this.delegate = delegate;
    }

    private AutowireCapableBeanFactory getBeanFactory() {
        return BeanLocator.getContext(applicationContextName)
                .getAutowireCapableBeanFactory();
    }

    @Override
    public Object deserialize(BufferInput in) throws
        IOException {

        Object result = delegate.deserialize(in);

        getBeanFactory().autowireBeanProperties(result,
           AutowireCapableBeanFactory.AUTOWIRE_BY_TYPE, false);

        return result;
    }

    @Override
    public void serialize(BufferOutput bufferoutput, Object obj)
            throws IOException {
        delegate.serialize(bufferoutput, obj);
    }
}
```

Recipe: Spring JMX Exporter

```
public class SpringCoherenceJMXExporter extends MBeanExporter {
```

Recipe: Spring JMX Exporter

```
public class SpringCoherenceJMXExporter extends MBeanExporter {

    @Override
    protected void doRegister(Object mbean, ObjectName
        objectName)
            throws JMXException {
```

Recipe: Spring JMX Exporter

```
public class SpringCoherenceJMXExporter extends MBeanExporter {

    @Override
    protected void doRegister(Object mbean, ObjectName
        objectName)
            throws JMXException {

        Registry registry =
            CacheFactory.ensureCluster().getManagement();
```

Recipe: Spring JMX Exporter

```
public class SpringCoherenceJMXExporter extends MBeanExporter {

    @Override
    protected void doRegister(Object mbean, ObjectName
        objectName)
            throws JMXException {

        Registry registry =
            CacheFactory.ensureCluster().getManagement();

        String sname =
            registry.ensureGlobalName(objectName.getCanonicalName());
```

Recipe: Spring JMX Exporter

```
public class SpringCoherenceJMXExporter extends MBeanExporter {

    @Override
    protected void doRegister(Object mbean, ObjectName
        objectName)
            throws JMXException {

        Registry registry =
            CacheFactory.ensureCluster().getManagement();

        String sname =
            registry.ensureGlobalName(objectName.getCanonicalName());

        registry.register(sname, mbean);
    }
}
```

Finally...

david.whitmarsh@sixwhits.com

prw.wheeler@gmail.com

`http://www.coherencecookbook.org/downloads`

Luncheon!