

Using the Java™ Management Extensions (JMX™) API for Monitoring and Management

Sang Shin
sang.shin@sun.com

Agenda

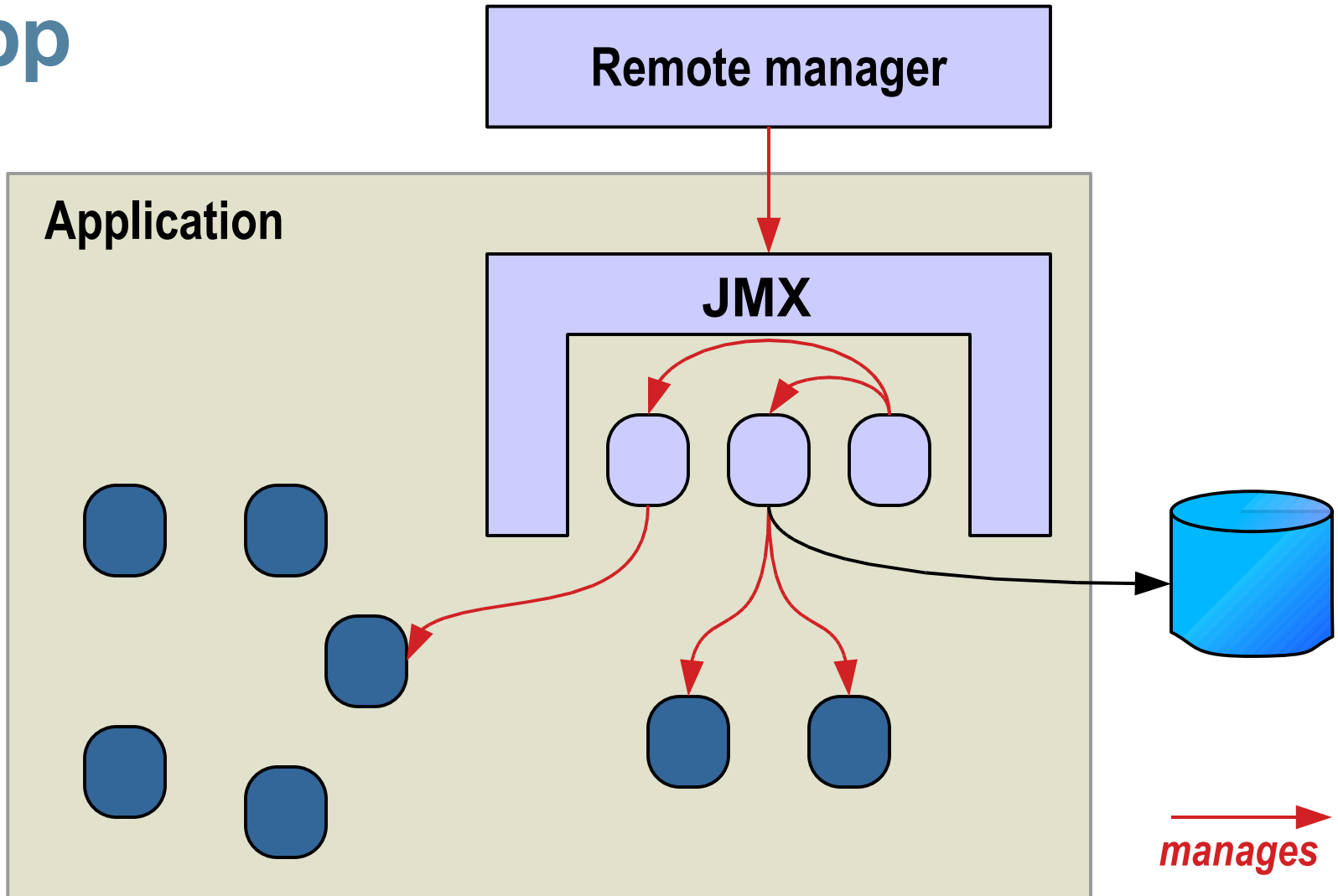
Introduction to the JMX™ API

Defining your own instrumentation

Accessing your instrumentation remotely

New features in Java SE 6

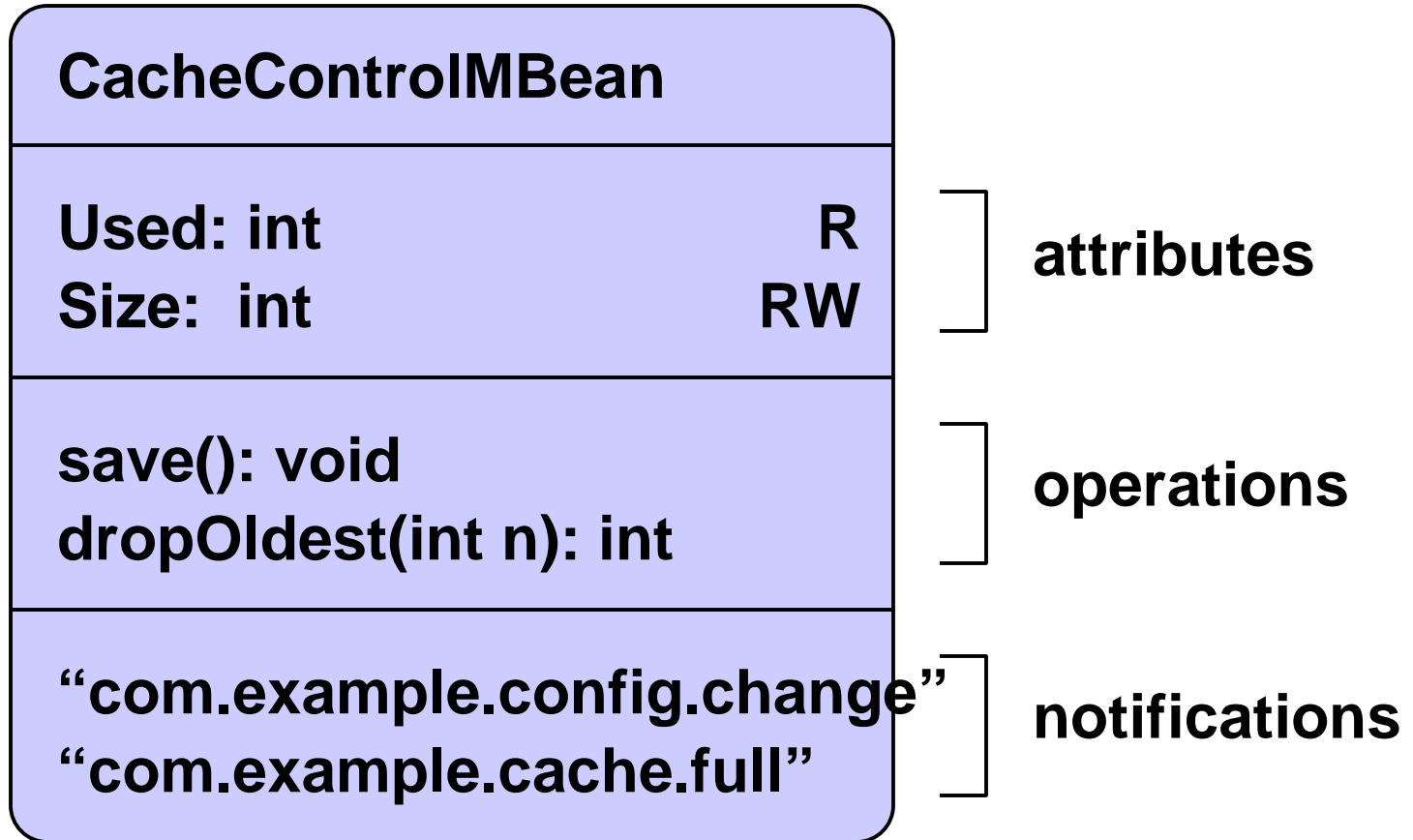
Adding JMX instrumentation to your app



MBeans

- An MBean is a named *managed object* representing a *resource*
 - > application configuration setting
 - > program module
 - > user identity
 - > device
 - > etc
- An MBean can have:
 - > *attributes* that can be read and/or written
 - > *operations* that can be invoked
 - > *notifications* that the MBean can send

MBean example



Naming MBeans

- Every MBean has a name
- A name is an instance of the ObjectName class (javax.management.ObjectName)
- A name has a domain and one or more key properties

com.example:type=CacheControl

com.example:type=CacheControl,name=whatsitCacheControl

java.lang:type=Threading

java.lang:type=MemoryPool,name=PS Perm Gen

domain

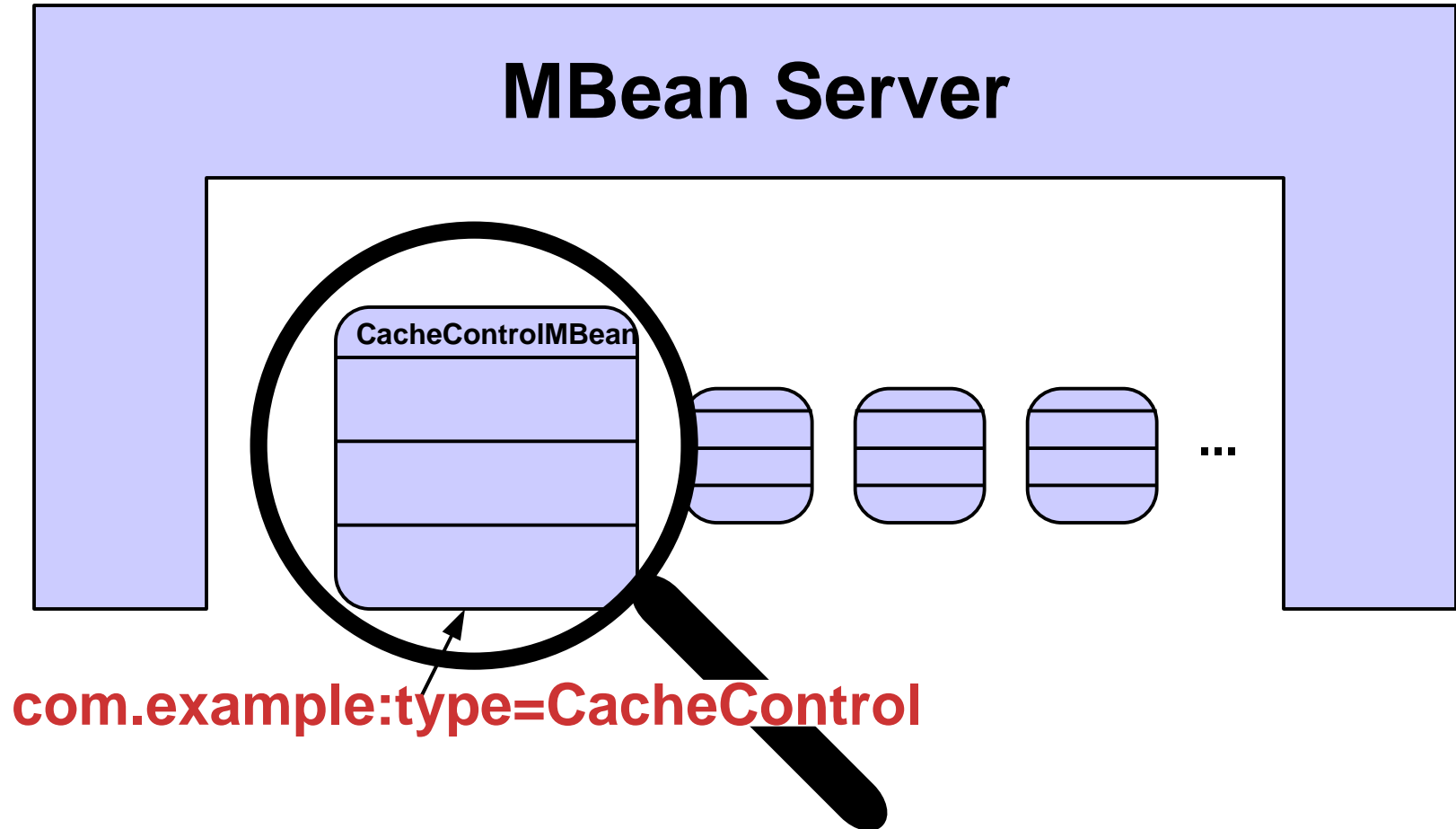
key properties



MBean Server

- To be useful, an MBean must be registered in an *MBean Server*
- Each MBean is registered with its `ObjectName`
- Usually, the only access to MBeans is through the MBean Server
- You can have more than one MBean Server per Java™ Virtual Machine (JVM™ machine)
- But usually, as of Java SE 5, everyone uses the *Platform MBean Server*
 - > `java.lang.management.ManagementFactory.
getPlatformMBeanServer()`

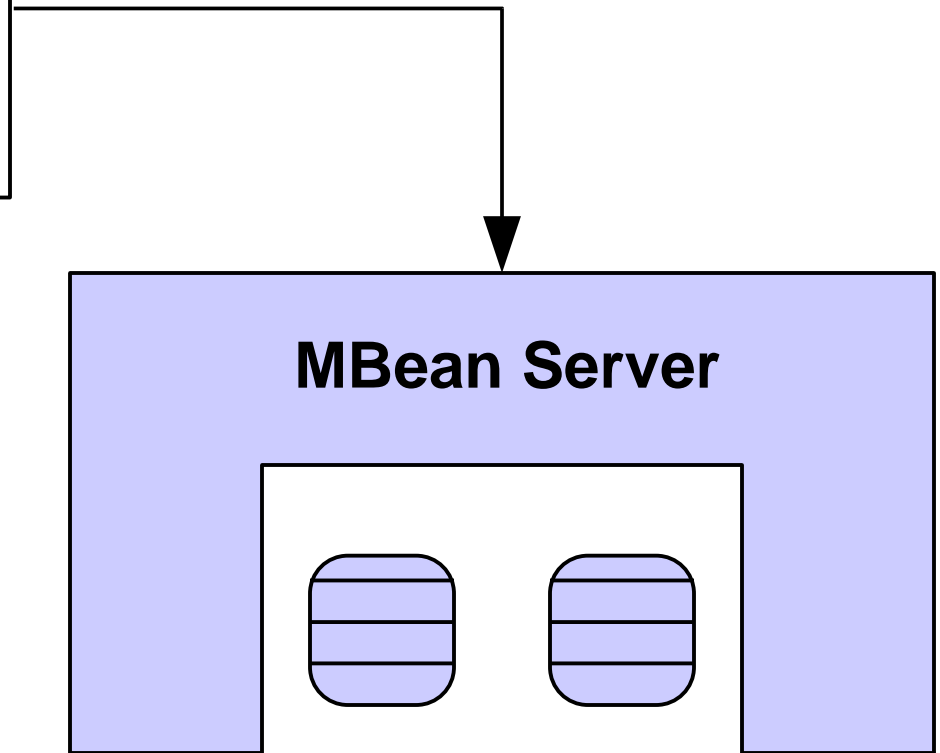
MBean Server



MBean Server

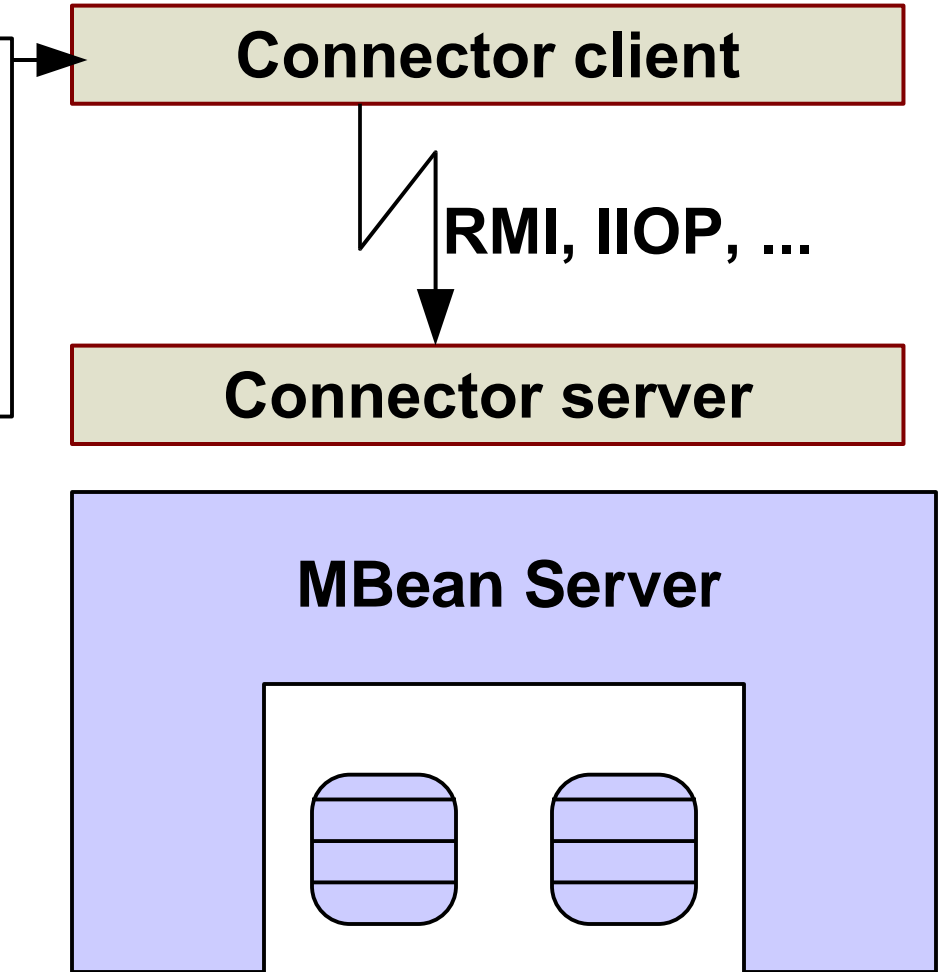
Local clients

```
MBeanServer mbs;  
  
mbs.createMBean(...);  
mbs.invoke(...);  
mbs.queryMBeans(...);
```



MBean Server Connector clients

```
MBeanServerConnection
  mbs ;
mbs.createMBean (...);
mbs.invoke (...);
mbs.queryMBeans (...);
```

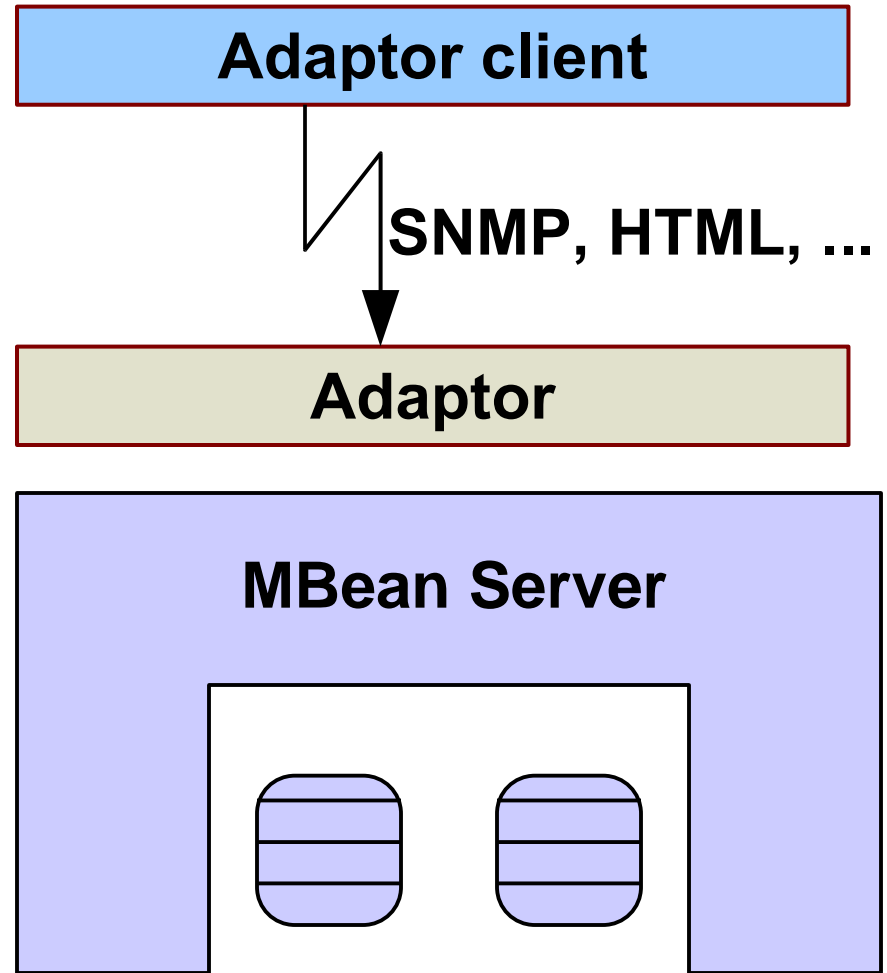


MBean Server

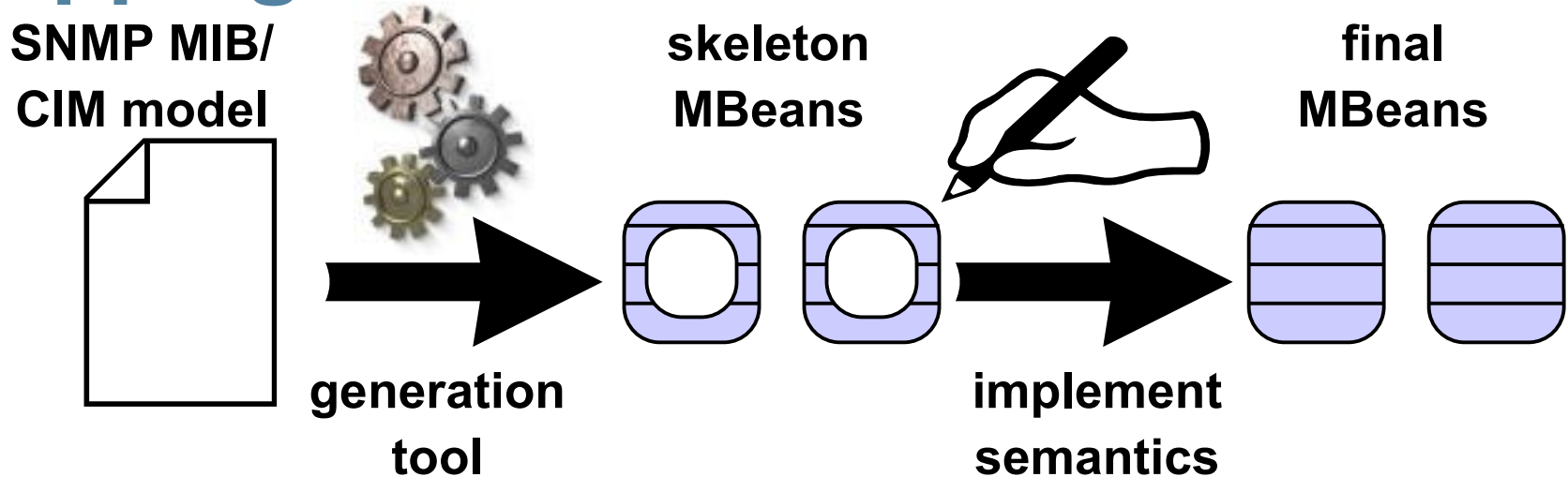
Connector notes

- Connectors defined by the JMX Remote API (JSR 160)
- Unrelated to the J2EE™ Connector Architecture
- Java SE architecture includes RMI and RMI/IIOP connectors
- JSR 160 also defines a purpose-built protocol, JMXMP
 - > Fits into existing security infrastructures via SASL
- Future work: a SOAP-based connector for the Web Services world (JSR 262)

MBean Server Adaptor clients



Mapping SNMP or CIM to JMX API

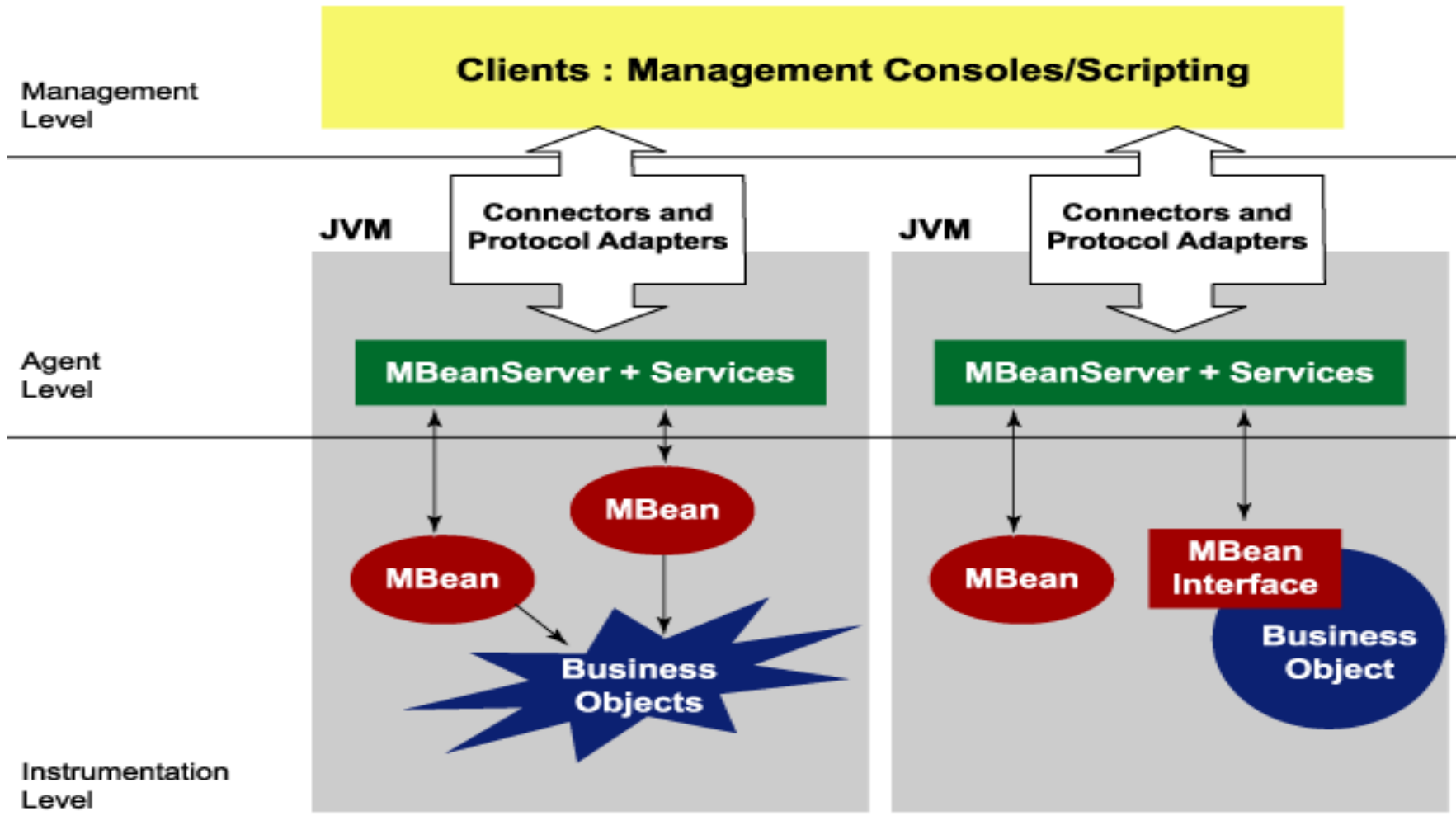


- Generation not currently standard
 - > proprietary solutions exist (Sun's is JDMK)
- Implementing semantics may mean mapping to another, “native” JMX API model
- Automated reverse mapping from JMX API to SNMP or CIM gives poor results

JMX API Services

- JMX API includes a number of pre-defined services
 - > Services are themselves MBeans
- Monitoring service (thresholding)
 - > javax.management.monitor
- Relation service (relations between MBeans)
 - > javax.management.relation
- Timer service
 - > javax.management.timer
- M-let service
 - > javax.management.loading

JMX Architecture



Agenda

Introduction to the JMX™ API

Defining your own instrumentation

Accessing your instrumentation remotely

New features in Java SE 6

Standard MBeans

- By far the simplest way to define an MBean
- Create a Java™ interface called *SomethingMBean* that defines the MBean's attributes and operations
- Create a Java™ class called *Something* that implements that interface

Standard MBeans: example (1/3)

CacheControlMBean.java

```
public interface CacheControlMBean {
    // a read-write attribute called Size
    // of type int
    public int getSize();
    public void setSize(int size);

    // a read-only attribute called Used
    // of type int
    public int getUsed();

    // an operation called save with no parameters
    // and no return value
    public void save() throws IOException;
}
```

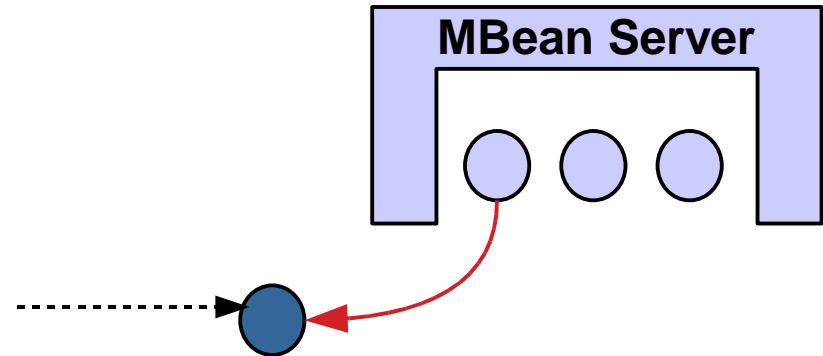
Standard MBeans: example (2/3)

CacheControl.java – attributes

```
public class CacheControl
    implements CacheControlMBean {
    public CacheControl(Cache<?> cache) {
        this.cache = cache;
    }

    public int getSize() {
        return cache.getSize();
    }
    public void setSize(int size) {
        cache.setSize(size);
    }

    public int getUsed() {
        return cache.getUsed();
    }
    ...
    private Cache<?> cache;
}
```



Standard MBeans: example (3/3)

CacheControl.java – operations

```
public class CacheControl
    implements CacheControlMBean {

    public CacheControl(Cache<?> cache) {
        this.cache = cache;
        cache.setSize(prefs.getInt("CacheSize",
                                   cache.getSize()));
    }

    public void save() {
        prefs.putInt("CacheSize", cache.getSize());
    }
    ...
    private static final Preferences prefs =
        Preferences.userNodeForPackage(
            CacheControl.class);
    // one way to do persistence
}
```

Registering your MBean

```
MBeanServer mbs =  
    ManagementFactory.getPlatformMBeanServer();  
  
ObjectName name =  
    new ObjectName("com.example:type=CacheControl");  
CacheControl cc = new CacheControl(cache);  
mbs.registerMBean(cc, name);
```

- There is also a createMBean operation
 - > mostly interesting for remote clients

```
mbs.createMBean(GaugeMonitor.class.getName(), name);
```

Agenda

Introduction to the JMX™ API

Defining your own instrumentation

Accessing your instrumentation remotely

New features in Java SE 6

Running a Java app with remote access to JMX instrumentation

- With Sun's JDK 5.0 on supported platforms:

```
> java -Dcom.sun.management.jmxremote MainClass
```

- Then, in another window on the same machine:

```
> jconsole
```

- With NetBeans JMX plugin this is one button
- Accessing from a remote machine is also possible with a bit more work
- In JDK 6 you won't need the property any more

Agenda

Introduction to the JMX™ API

Defining your own instrumentation

Accessing your instrumentation remotely

New features in Java SE 6

MXBeans: Problem statement (1)

- An MBean interface can include arbitrary Java™ programming language types

```
public interface ThreadMBean {  
    public ThreadInfo getThreadInfo();  
}  
  
public class ThreadInfo {  
    public String getName();  
    public long getBlockedCount();  
    public long getBlockedTime();  
    ...  
}
```

- When values must be grouped atomically

MXBeans: Problem statement (2)

- An MBean interface can include arbitrary Java™ programming language types

```
public interface ThreadMBean {  
    public ThreadInfo getThreadInfo();  
}
```

- Client must have these classes
- What about generic clients like jconsole?
- What about versioning?

Open MBeans

- JMX™ API defines **Open MBeans**
 - > javax.management.openmbean
- Predefined set of basic types
 - > Integer, String, Date, ObjectName, ...
- Complex types made using arrays and/or two predefined compositional types
 - > CompositeData
 - > TabularData

MXBeans (1)

- MXBeans were designed for the instrumentation of the VM itself (JSR 174)
 - > Already exist in `java.lang.management`
 - > User-defined MXBeans are new in Mustang
- Management interface still a bean interface
- Can reference arbitrary types, with some restrictions
- JMX™ API **wraps** an instance of this interface in an Open MBean

MXBeans (2)

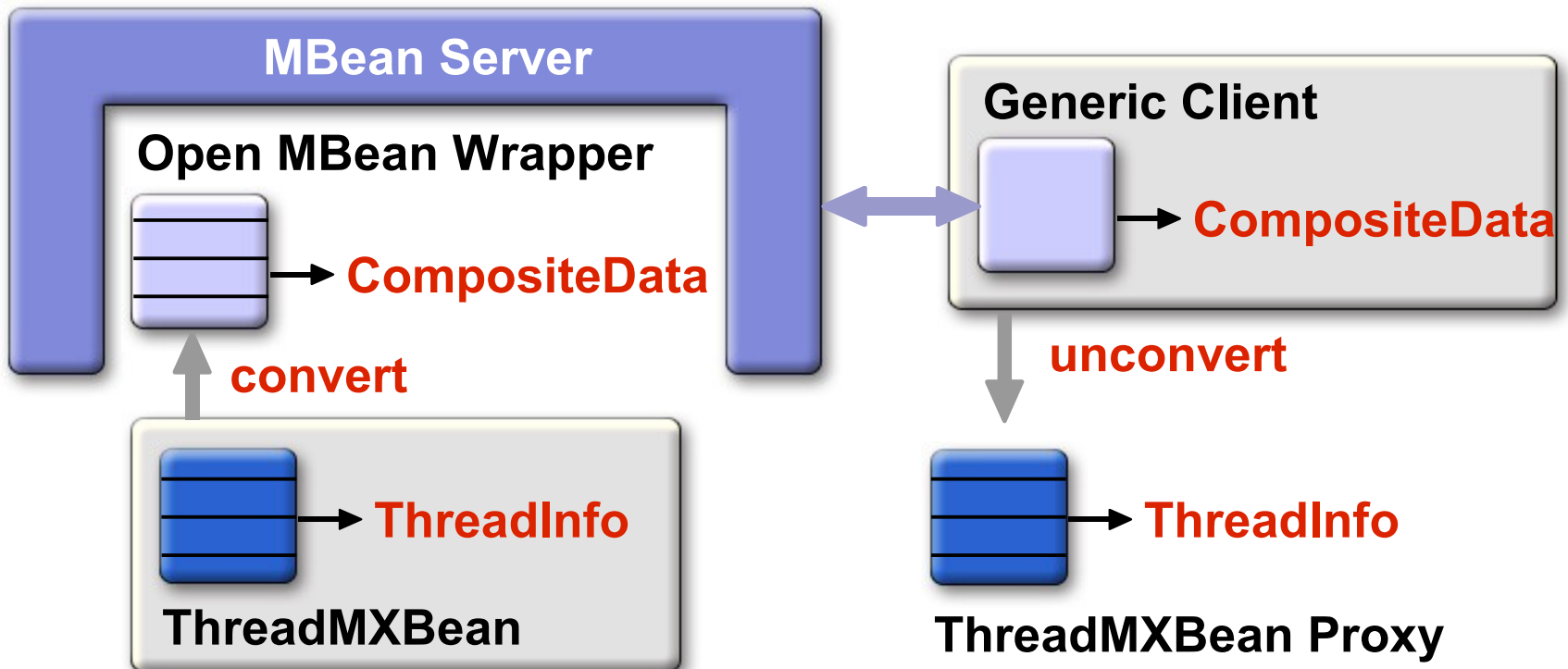
```
public interface ThreadMXBean {  
    public ThreadInfo getThreadInfo();  
}
```

```
public class ThreadMXBeanImpl implements ThreadMXBean {  
    // Do not need Something/SomethingMXBean naming  
    public ThreadInfo getThreadInfo() {  
        return new ThreadInfo(...);  
    }  
}
```

```
ThreadMXBean mxbean = new ThreadMXBeanImpl();  
ObjectName name =  
    new ObjectName("java.lang:type=Threading");  
  
mbs.registerMBean(mxbean, name);
```

MXBeans (3)

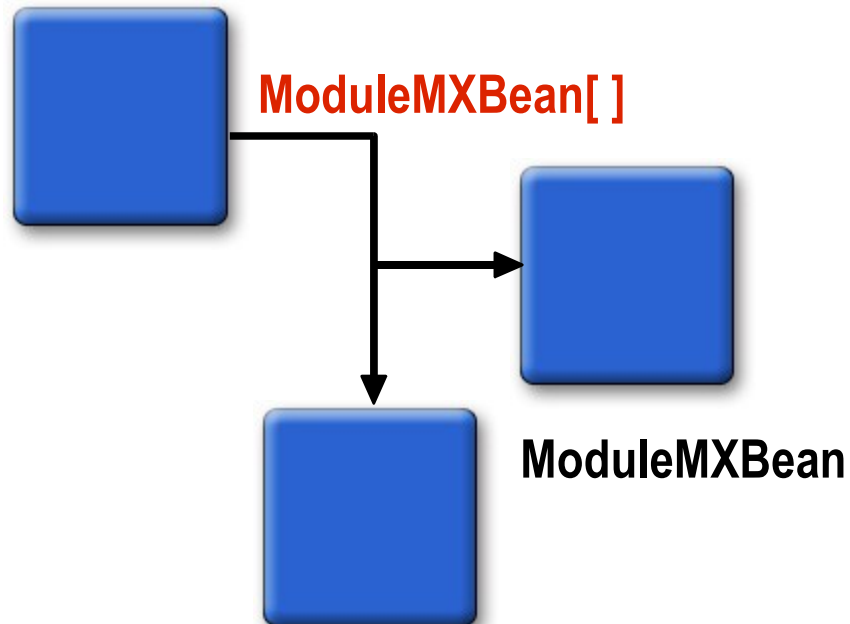
- Generic client can access as Open MBean
- Model-aware client can make ThreadMXBean proxy



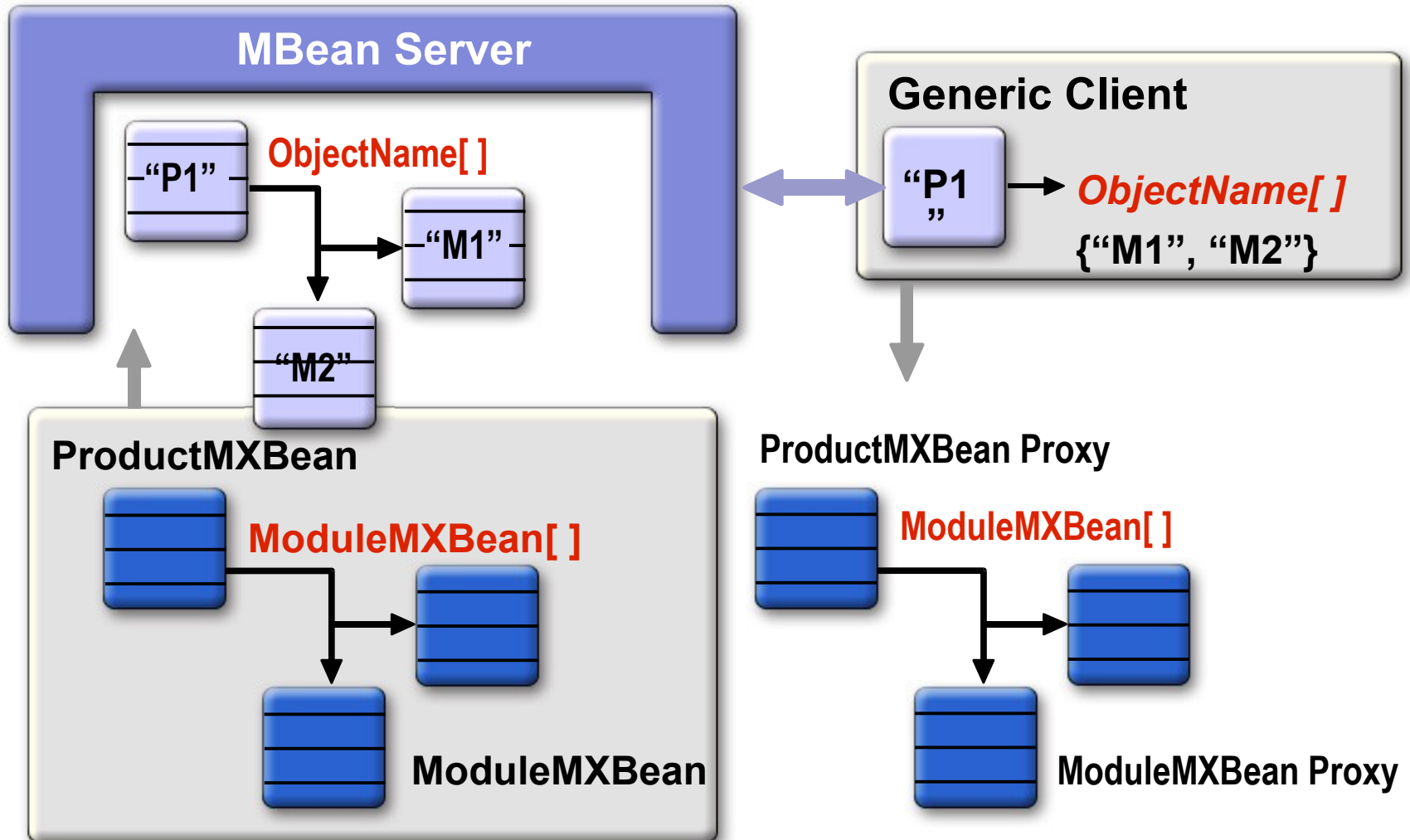
MXBean References (1)

```
public interface ProductMXBean {  
    ModuleMXBean[] getModules();  
}
```

ProductMXBean



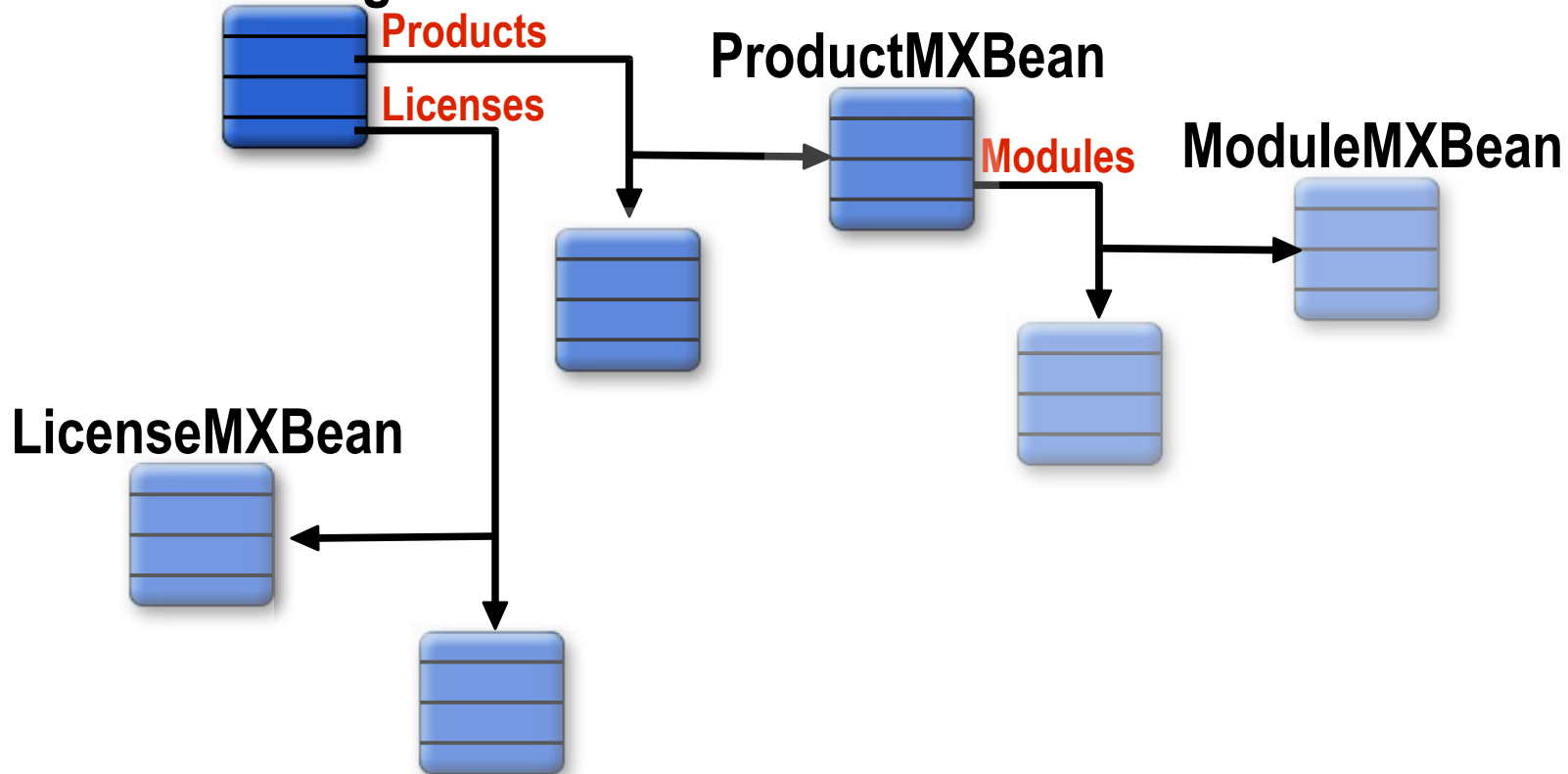
MXBean References (2)



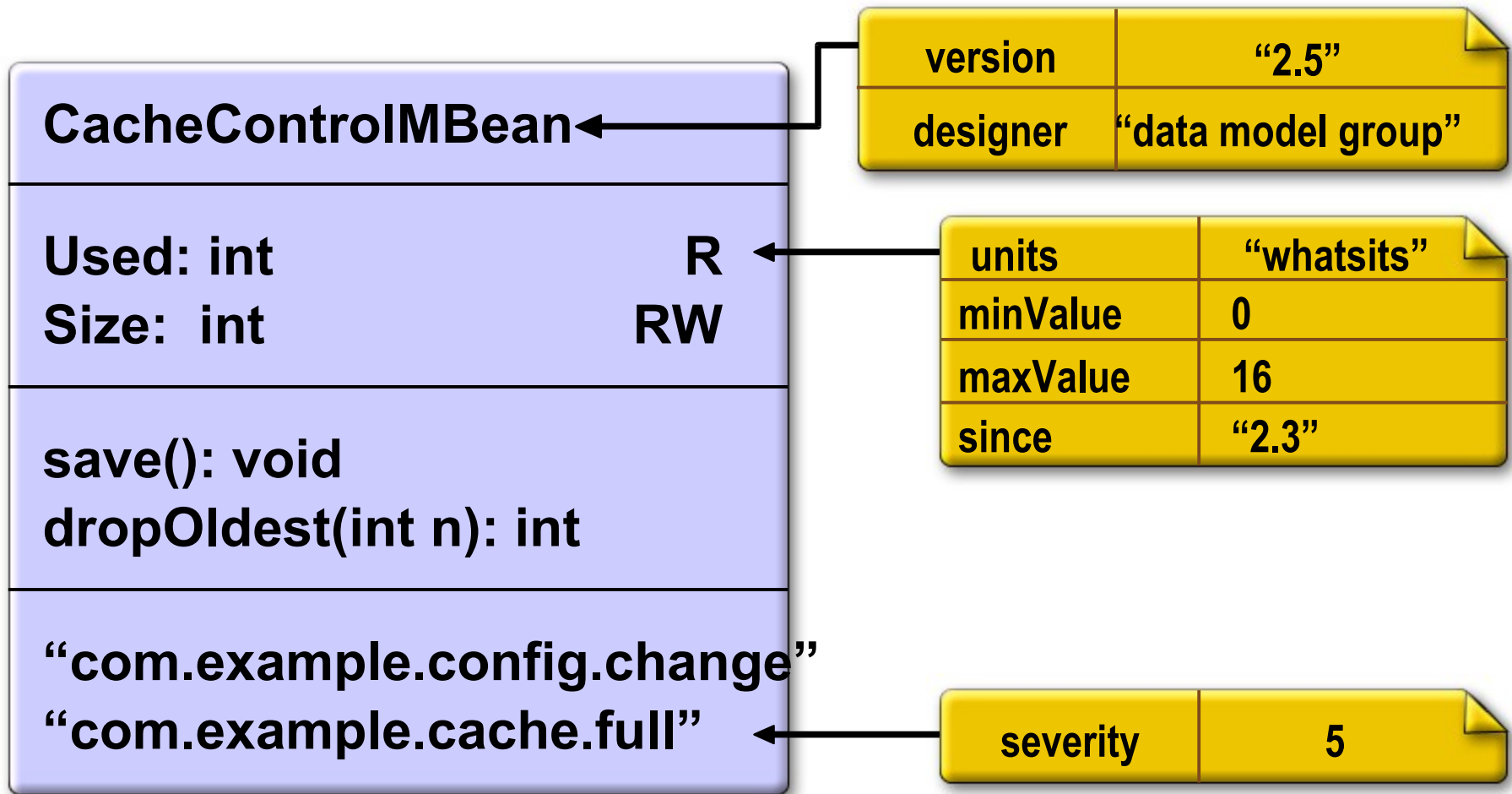
MXBean References (3)

Navigating from a starting point

InstallationManagerMXBean



Descriptors



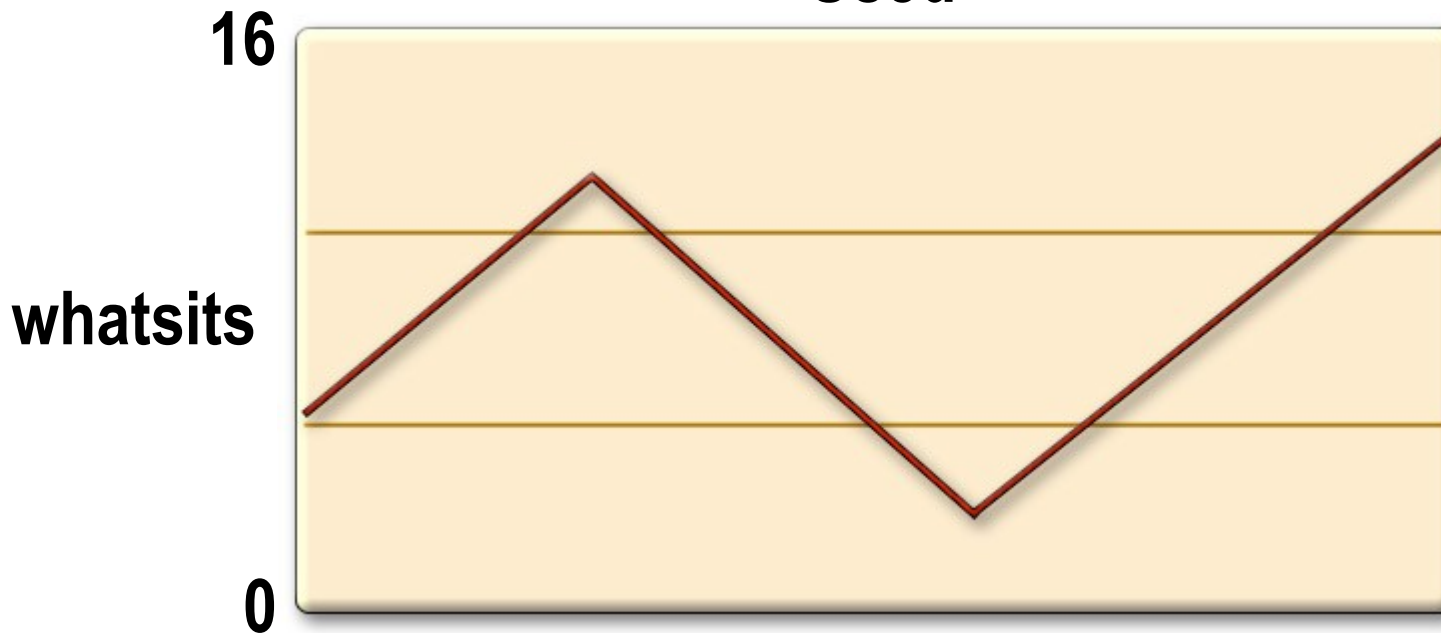
Descriptors and Generic Clients

(like jconsole)

Used: int **R**

units	"whatsits"
minValue	0
maxValue	16

Used



Descriptor details

- Classes MBeanInfo, MBeanAttributeInfo, etc., now have an optional Descriptor
- Every attribute, operation, notification can have its own Descriptor
- Descriptor is set of (key,value) pairs
- Some keys have conventional meanings
- Users can add their own keys
- Descriptors have always existed in Model MBeans

Descriptor Annotations

```
public interface CacheControlMBean {
    @Units("whatsits") @Range(minValue=0, maxValue=16)
    public int getUsed();
}
```

Used: int

R

units	"whatsits"
minValue	0
maxValue	16

- With definitions like:

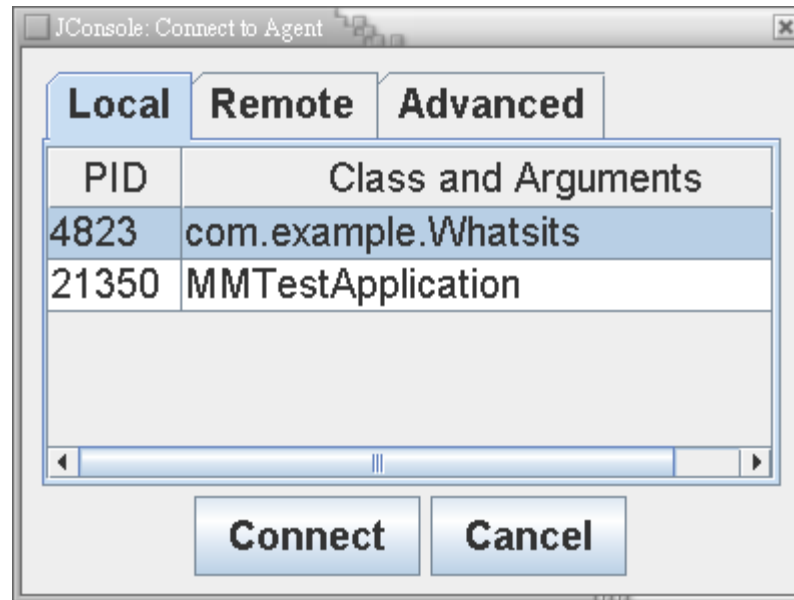
```
public @interface Range {
    @DescriptorKey("minValue")
    public int minValue();
    @DescriptorKey("maxValue")
    public int maxValue();
}
```

Some Other changes in Java SE 6

- Generified at last!
 - > Set<ObjectName> queryNames(...)
- More powerful ObjectName wildcards
 - > domain:type=Dir,path="/root/*"
- Simpler Notification use
 - > NotificationBroadcasterSupport(MBeanNotificationInfo[])
 - > class StandardEmitterMBean extends StandardMBean
- Monitor attributes of complex type
 - > MonitorMBean.setObservedAttribute("ThreadInfo.size")

<http://jdk6.dev.java.net>

In case the demo doesn't work (1/3)



In case the demo doesn't work (2/3)

J2SE 5.0 Monitoring & Management Console: 4823@localhost

Connection

Summary Memory Threads Classes **MBeans** VM

MBeans

Tree

- JMImplementation
 - com.example
 - CacheControl**
 - java.lang
 - java.util.logging

Attributes Operations Notifications[170] Info

Name	Value
Size	10

Used

Discard chart

Refresh

In case the demo doesn't work (3/3)

J2SE 5.0 Monitoring & Management Console: 4823@localhost

Connection

Summary | Memory | Threads | Classes | **MBeans** | VM

MBeans

Tree

- JMImplementation
- com.example
 - CacheControl**
- java.lang
- java.util.logging

Attributes		Operations		Notifications[209]		Info
TimeStamp	Se...	Se...	Message	Source		
19:28:15:4...	...	409	cache overflow ..	com.exam...		
19:28:13:4...	...	408	cache overflow ..	com.exam...		
19:28:09:3...	...	407	cache overflow ..	com.exam...		
19:28:07:3...	...	406	cache overflow ..	com.exam...		
19:28:02:3...	...	405	cache overflow ..	com.exam...		
19:28:01:2...	...	404	cache overflow ..	com.exam...		
19:28:00:2...	...	403	cache overflow ..	com.exam...		
19:27:59:2...	...	402	cache overflow ..	com.exam...		
19:27:57:2...	...	401	cache overflow ..	com.exam...		
19:27:56:2...	...	400	cache overflow ..	com.exam...		
19:27:54:2...	...	399	cache overflow ..	com.exam...		

Subscribe | Unsubscribe | Clear

For More Information

- <http://java.sun.com/jmx>
- jmx-forum@java.sun.com
- JMX Expert (JSR spec lead)
 - > Eamonn.McManus@Sun.Com
- <http://weblogs.java.net/blog/emcmanus>
- Netbeans JMX plugin
 - > <http://www.netbeans.org/kb/articles/jmx-getstart.html>
 - > jean-francoise.denise@sun.com

Using the Java™ Management Extensions (JMX™) API for Monitoring and Management

Sang Shin

Sun Microsystems

Standard MBeans: example (4)

CacheControl.java – notification outline

```
public class CacheControl
    extends NotificationBroadcasterSupport
    implements CacheControlMBean {
    public CacheControl(final Cache<?> cache) {
        this.cache = cache;
        Thread t = new Thread() {
            public void run() {
                while (true) {
                    cache.waitForOverflow();
                    Notification n = new Notification(
                        "com.example.cache.overflow",
                        CacheControl.this, seqNo++,
                        "cache overflow");
                    sendNotification(n);
                }
            }
        };
        t.start(); // or cache could support callbacks
    }
}
```