



*Paris,
France*



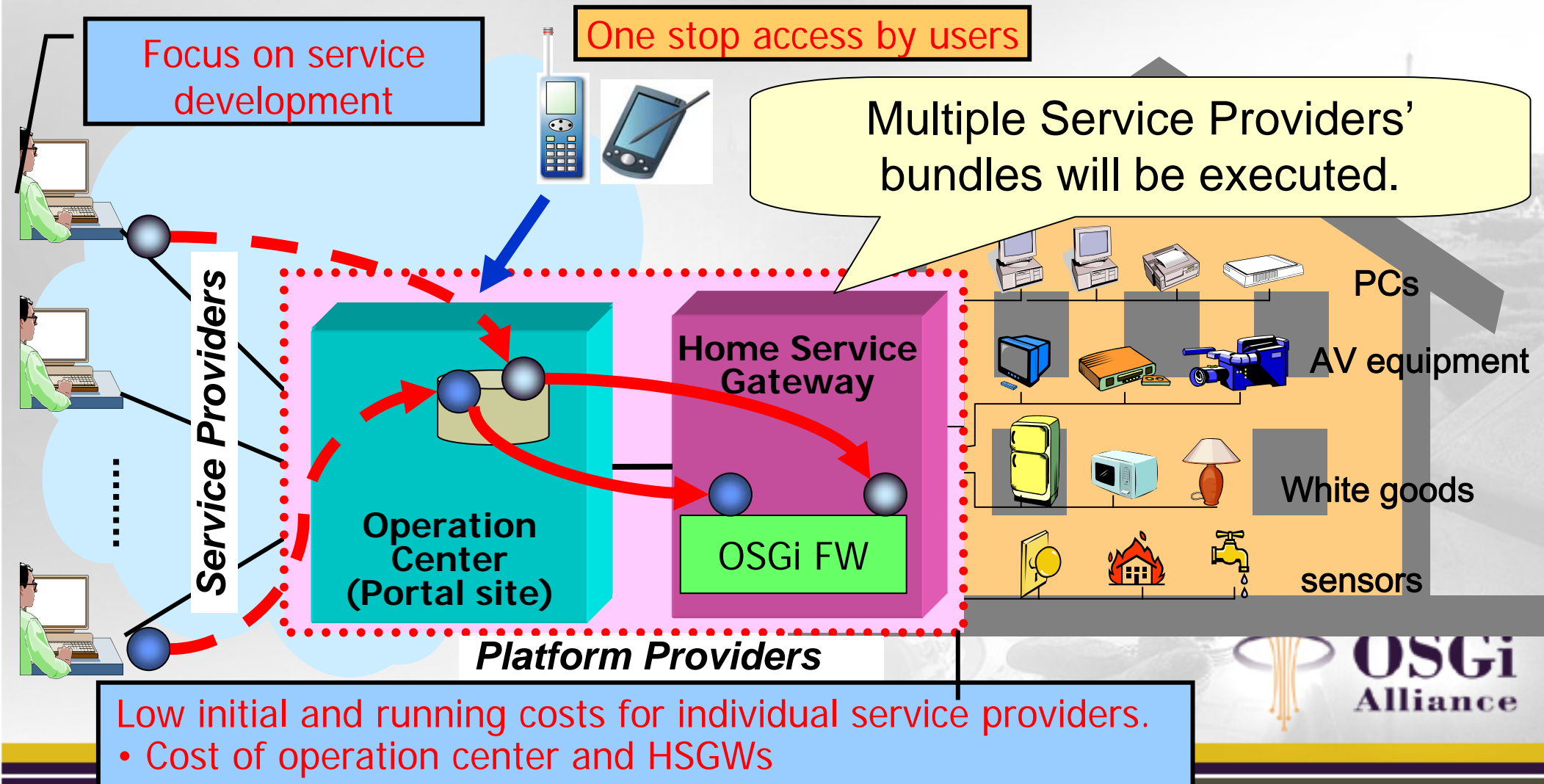
Monitoring and Managing Computer Resource Usage on OSGi Frameworks



Ikuo YAMASAKI
Research Engineer
NTT Cyber Solution Laboratories

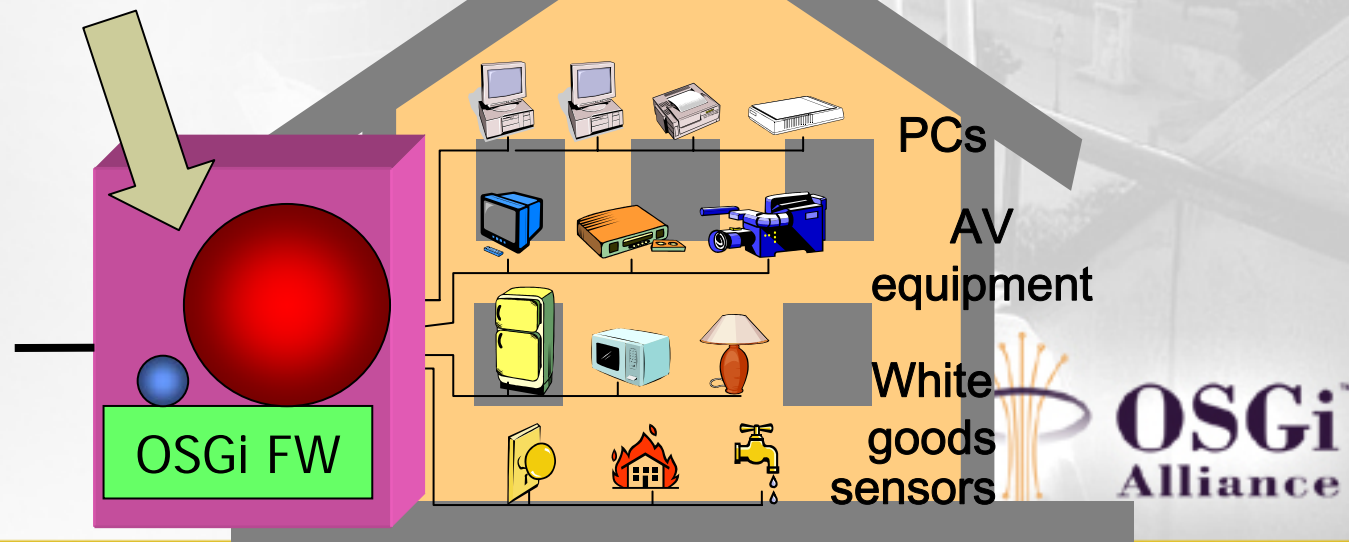
Background: Service Aggregation Platform

- Operation Center and Home Service Gateway (HSGW) are shared by multiple **Service Providers**.



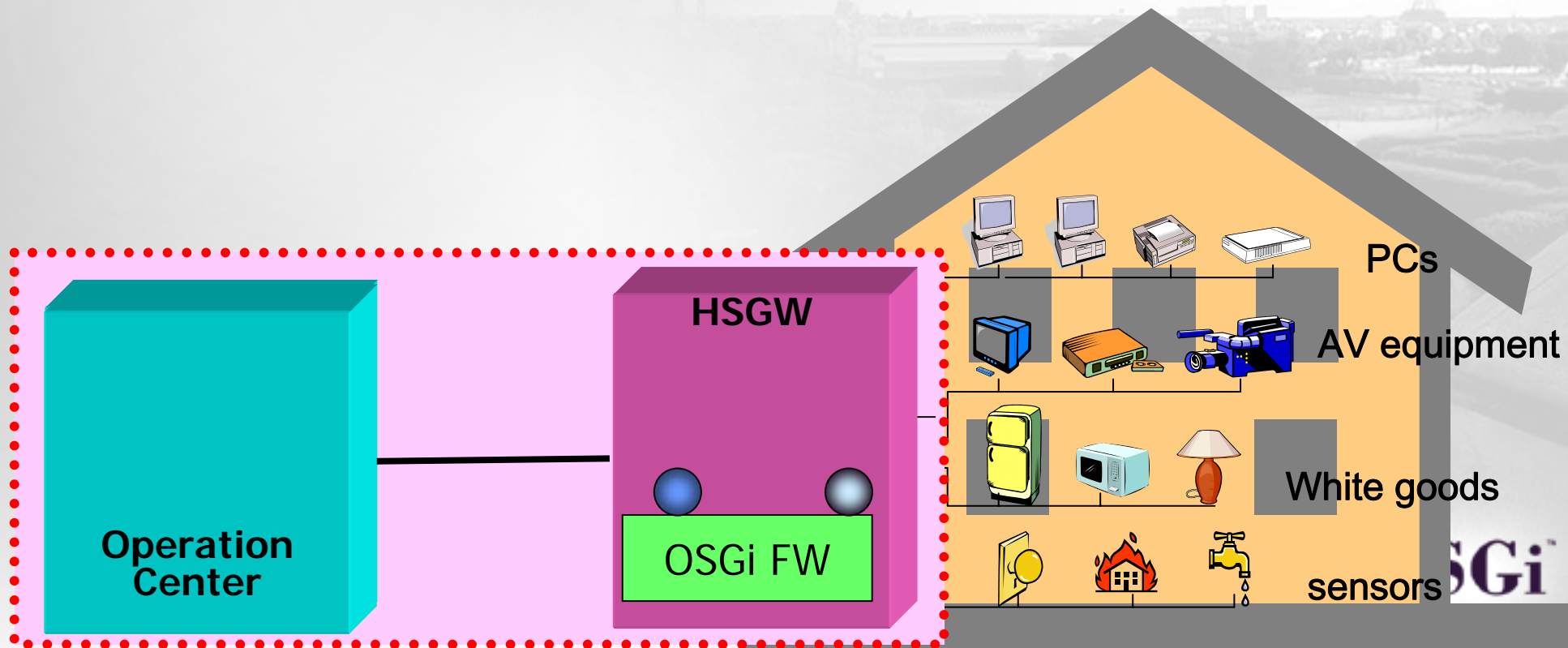
Background : resource starvation

- In OSGi, bundles create **threads**
 - After BundleActivator.start(bc) returns, threads do their jobs.
- **Threads** can deplete computer resources (CPU usage, memory usage)
 - Malicious code
 - Not intentionally, wrong implementation.
- Especially, resource starvation **by different Service Providers' bundles** is a big problem.



Our General Goal

- Develop (Remote) Monitoring and Management system of HSGW
 - Protect 'good' bundles by preventing malicious or misbehaving bundles on the same FW from depleting resources.



Design Concept

- **Lightweight**

- The monitoring & managing system adds minimal overhead.
- use **in service phase, not test phase.**

- **Backwards Compatibility**

- Any bundle that has been written must be able to work on the system.
- **Without rewriting source codes.**

- **Independent of specific vendors**

- Use only standardized measures.
- **No specific vendor's OSGi-FW or JVM.**



Resource Usage to Monitor

- Threads created by a bundle consume
 - **CPU resources** ← Target of this work
 - Our Definition: Sum of CPU usage by all **threads created by the bundle.**
 - **Memory resources** ← Out of scope
 - Definition of “how much memory a thread and a bundle is using” is controversial.
 - There is no way to monitor it, even in J2SE5.0.

To achieve the goal

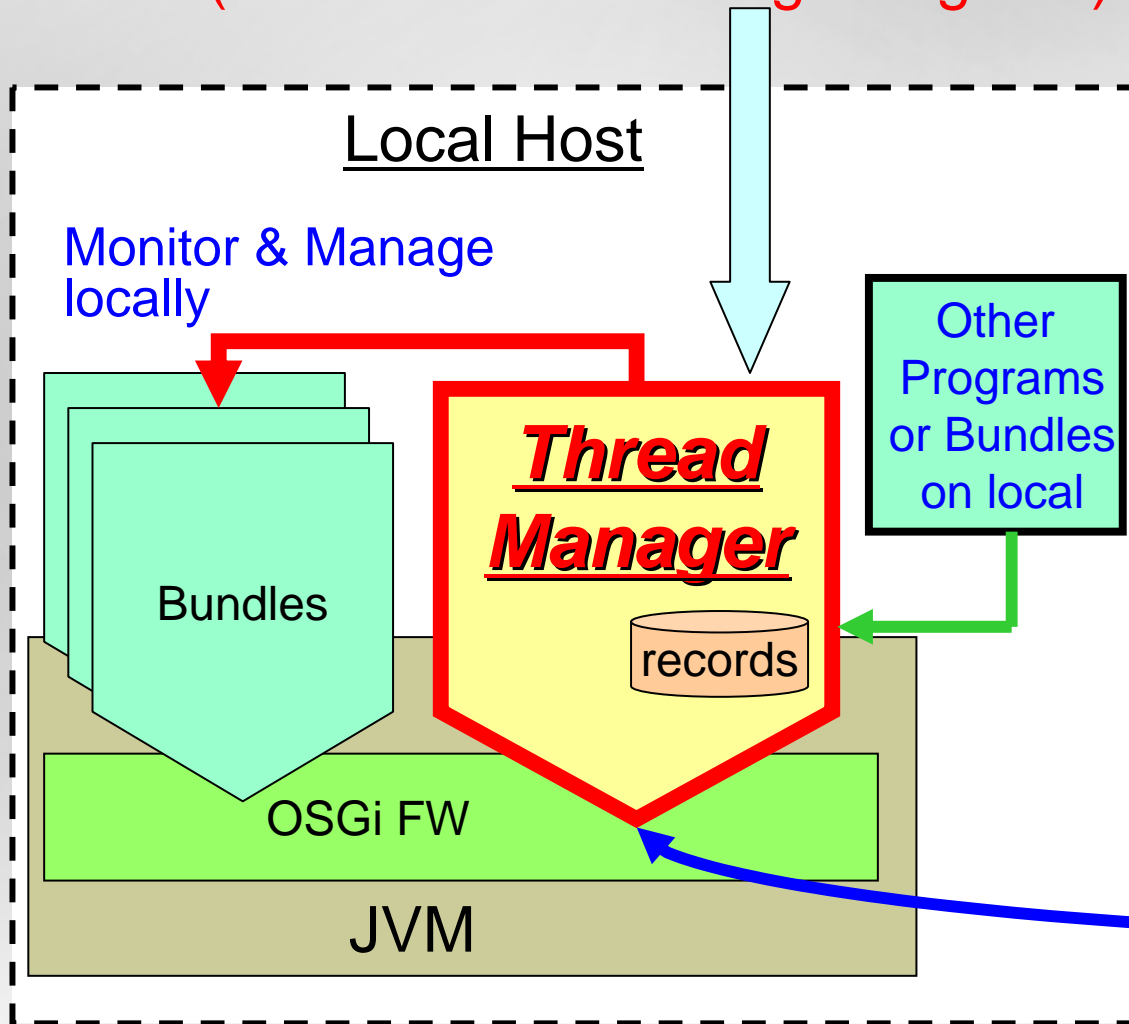
- **Management Entity** needs to
 1. **Monitor CPU usage consumed by each bundle.**
 - a. Monitor CPU usage of each thread.
 - b. Manage the relations between bundles and threads, and Calculate each bundle's CPU usage.
 2. **According to the management policy adopted, some reaction needed.**
 - If any bundle consumes too many resources,
 - Take action against excessive bundle, to protect other bundles' activities.

Management Policies (Mgt. Policies)

- Adaptability to Multiple **mgt. policies** is required.
 - **Triggers** : When and Which bundle should be suppressed because of its excessive consumption?
 - e.g.
 - If system CPU usage exceeds 97% for the latest 10 min, the bundle consuming the most should be suppressed.
 - A bundle whose CPU usage exceeds 90% for the last 10 min should be suppressed.
 - **Reactions** : How to suppress the bundle?
 - a. Change the thread's priority
 - b. Kill the thread
 - c. Stop the bundle (and kill all the threads created)
 - **Activators** : Which entity **triggers** the **reaction**?
 1. Entity working on the same host.
 2. Remote monitoring and management entity.

Our System Architecture

Platform Provider provides
(as one of the Managed Agents)



1. Local entity

- Automatic reaction
 - Owns mgt. policy.
 - Even if the network connection is dead, it works.

2. Remote entity

- Automatic reaction
 - Owns mgt. policy.
- *Platform Provider's operator driven reaction*

How to monitor **thread** CPU usage ?

- **Design Concept**

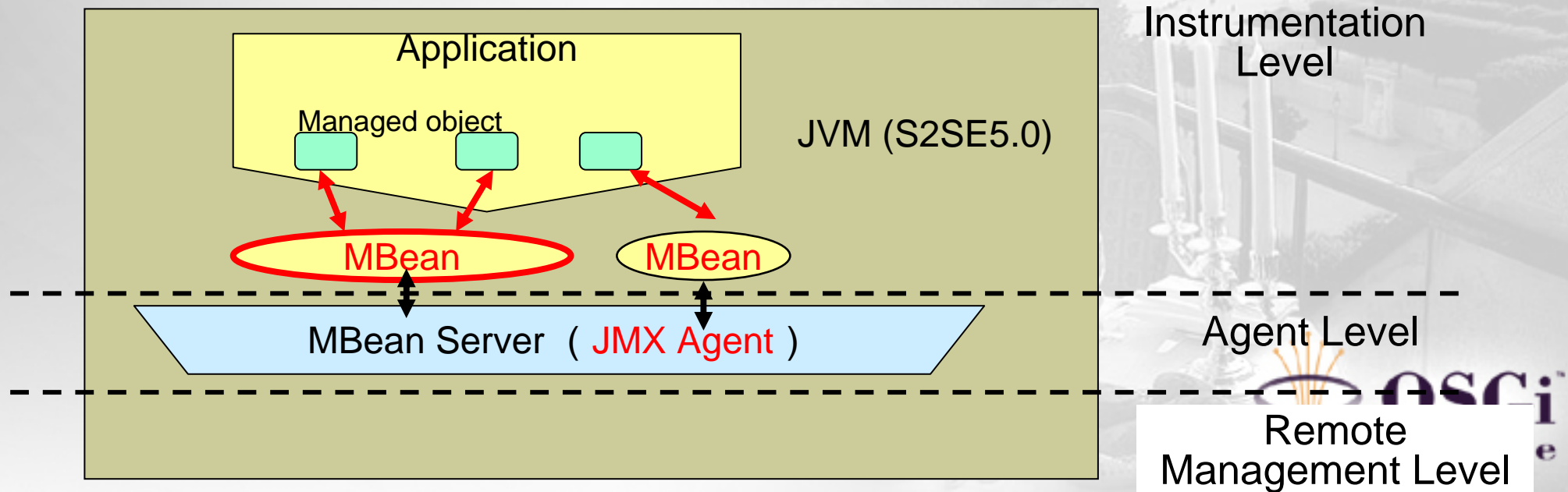
- Lightweight
- Backward compatible
- **Independent of specific vendors**

- **JMX** of **J2SE5.0** (used to be called “Tiger”) can monitor every thread.

- Demerit: J2SE5.0 requires high spec machines...
 - Difficult to deploy in embedded systems.

JMX (Java Management eXtention)

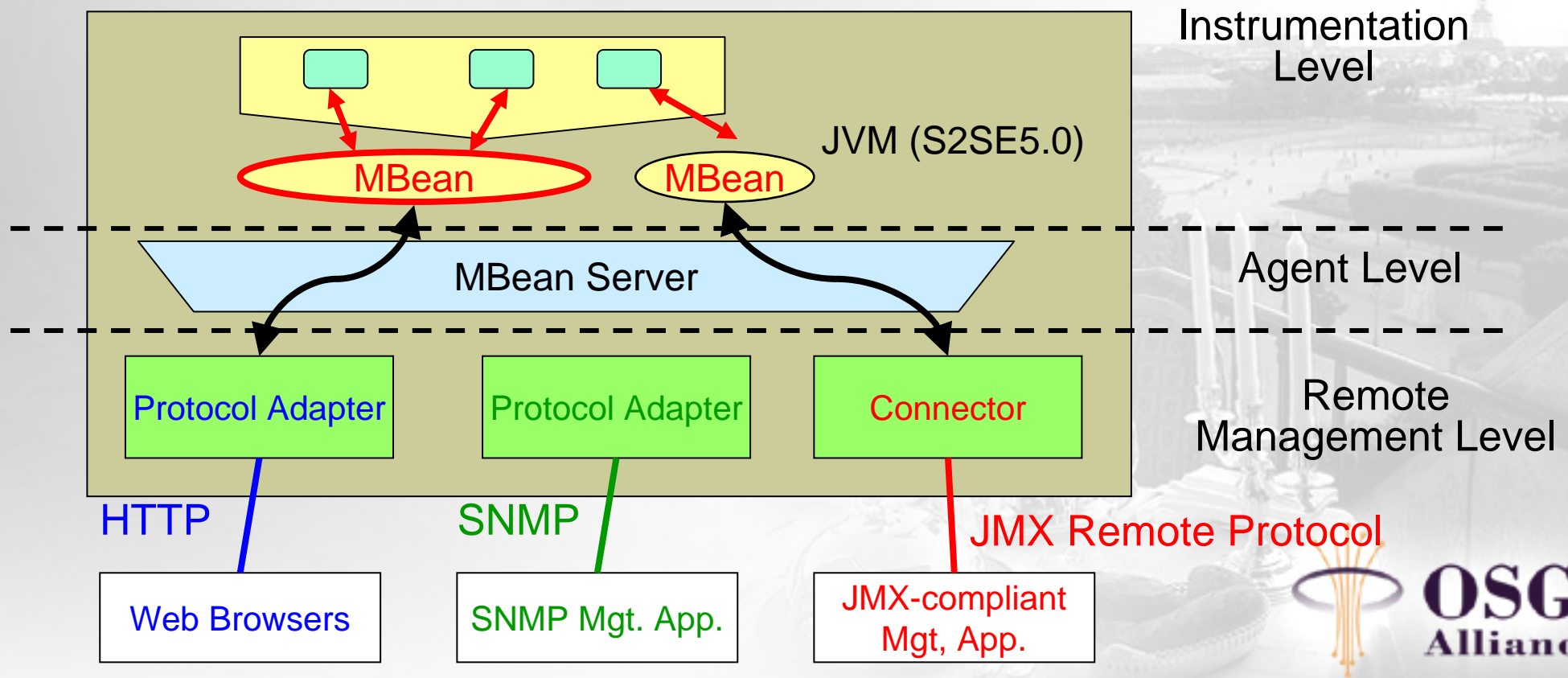
1. Provides **standardized measures** to **monitor** and **manage** Java applications.
 - Developer can define own **MBean** (Managed Bean)
 - Getter / setter style
 - JMX Agent (**MBean Server**) enables management of target object through registered **MBean**



JMX (Java Management eXtention)

2. Resources can be monitored and managed, either **remotely** or **locally**.

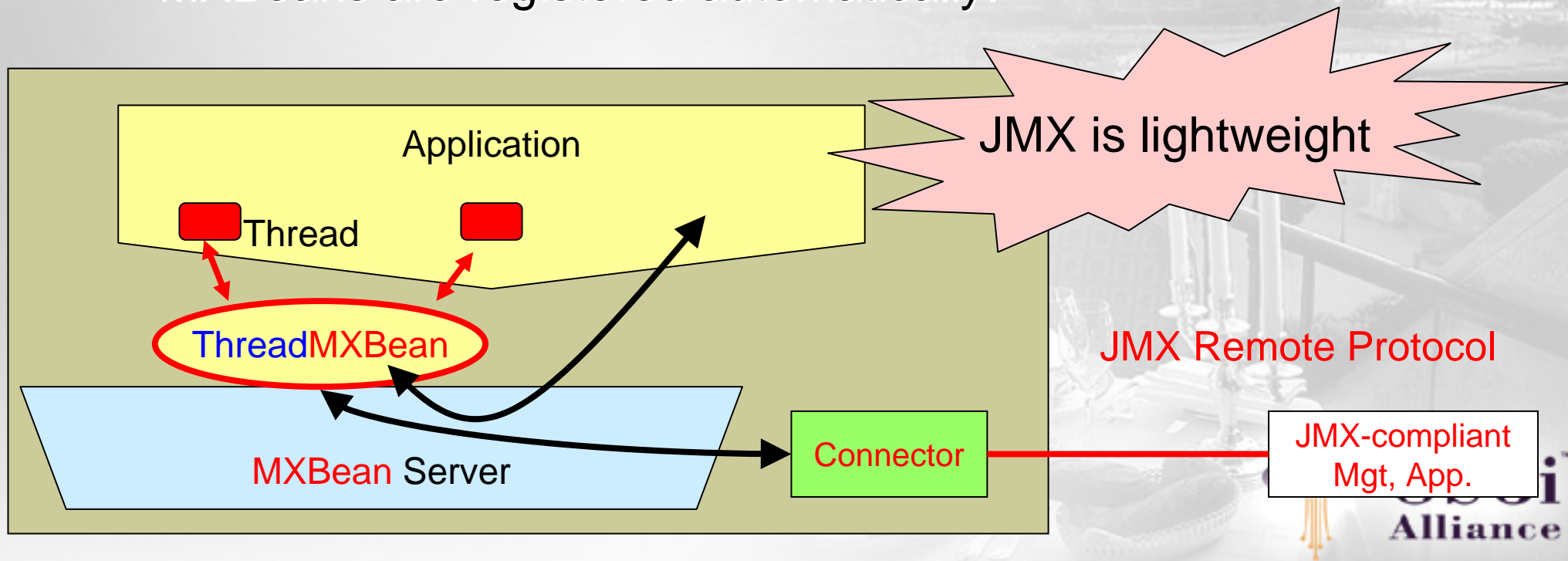
- Protocol is replaceable.
 - **RMI connector** uses Java RMI.



JMX (Java Management eXtention)

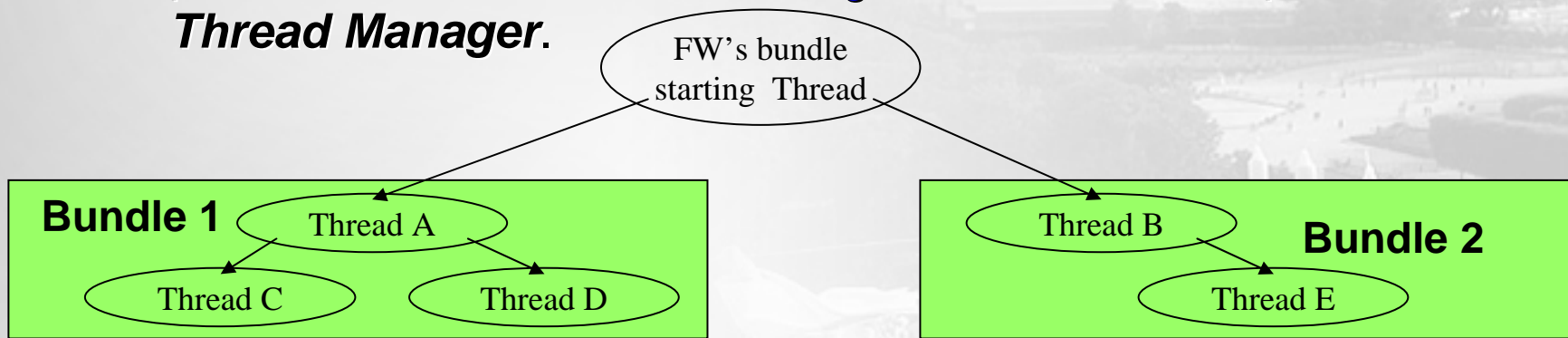
3. Platform MBean Server (MXBean Server) is built in JVM

- Several types of **MXBeans** are defined in the spec.
 - Every thread can be monitored and managed through **ThreadMXBean**
 - including **CPU usage**
- MXBeans are registered automatically.



How to monitor **bundle** CPU usage ?

- We define **Bundle-Thread-Tree (BTT)**
 - in order for *Thread Manager* to maintain the **relations** between **every thread** and **its creating bundle**
 - At the creation of a thread,
 - pair of the **thread** and **its creating bundle** must be reported to the ***Thread Manager***.



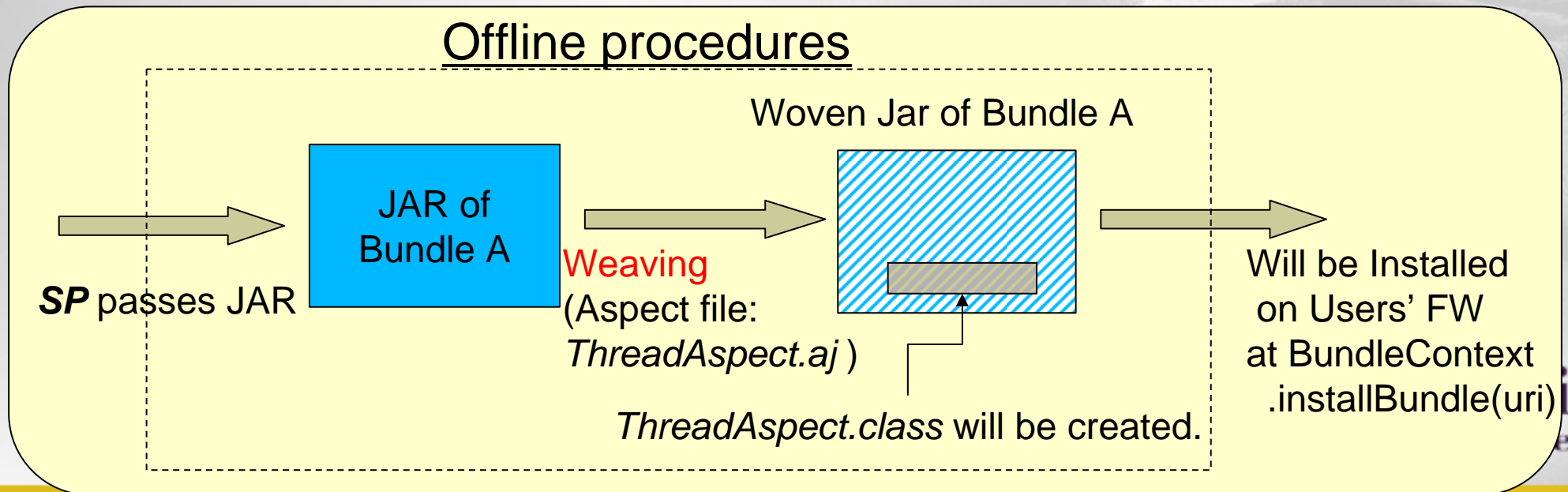
- If rewriting codes is acceptable, no problem.
- But our **Design Concept: Backward compatibility**
 - No codes rewriting permitted.

Byte Code Weaving: AOP (AspectJ)

- We adopt **Byte Code Weaving**.
 - **AOP (Aspect Oriented Programming)**: AspectJ
- “**Weave**” : AOP allows us to insert **advice** before/after/around **pointcut** in the byte code.
 - **Advice** : byte code to be woven
 - **Pointcut** : where to weave the byte code.
 - E.g. **Insert the specified code at the beginning / end of the specified method of the specified class.**
- **Byte code** can be altered even **without source code** of the program.

Offline Bundle Weaving

1. **Service Provider (SP)** passes its bundle JAR to **Platform Provider** (without source code).
2. **Platform Provider** weaves the aspects into the bundle and posts it on the web server.
3. OSGi-FWs of HSGW will install the woven bundle from the web server.



Aspect to Weave into bundles 1

- **Pointcut 1:** Before a **bundle** is started
- **Advice Inserted:** set its **bundle object** into the field variable “*bundle*” in *ThreadAspect* class.

```
public class Activator implements BundleActivator {  
    public void start(BundleContext bc) throws Ex {  
        Bundle bundle = bc.getBundle();  
        ThreadAspect.getInstance().setBundle(bundle);  
        System.out.println("OSGi: Bundle started");  
        ...  
    }  
    ...  
}
```

Pseudo woven code

Aspect to Weave into bundles 2

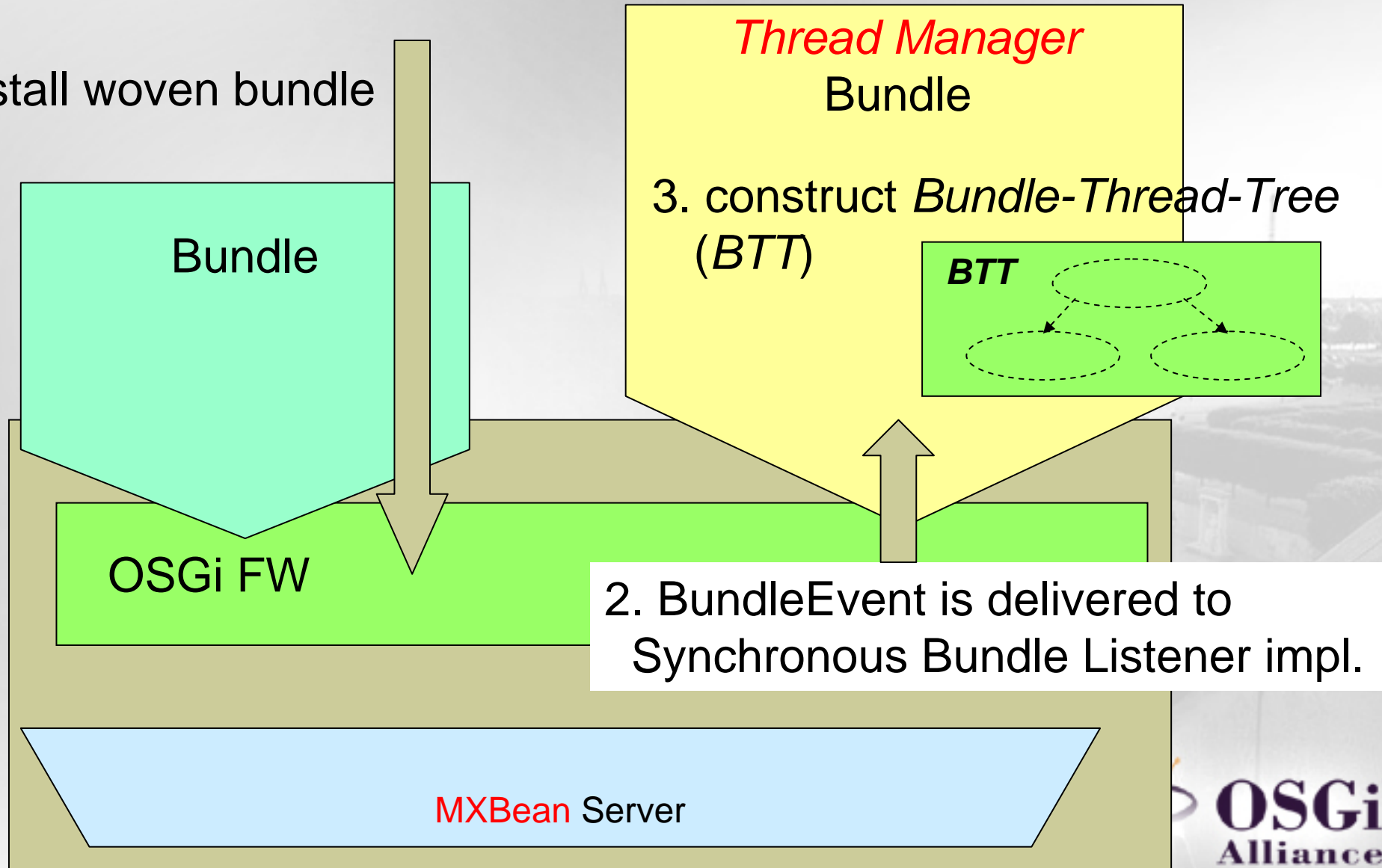
- **Pointcut 2:** Before a **thread** is started
- **Advice Inserted:**
 - A) retrieve its **bundle object** from the field variable “*bundle*” in *ThreadAspect* class, and
 - B) notify the relation between **the thread** and **bundle creating it** to *ThreadManager*.

```
public class ThreadCreate {  
    public void doTask() {  
        Thread t = new Thread(new Runnable() {  
            public void run() {...;}  
        });  
        Bundle bundle = ThreadAspect.getInstance().getBundle();  
        ThreadManager tm = ThreadManager.getInstance();  
        tm.addThread(t, bundle);  
        t.start();  
    }  
}
```

Pseudo woven code

Operation of the System

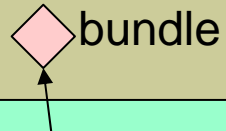
1. Install woven bundle



Operation of the System

4. Bundle is started

ThreadAspect



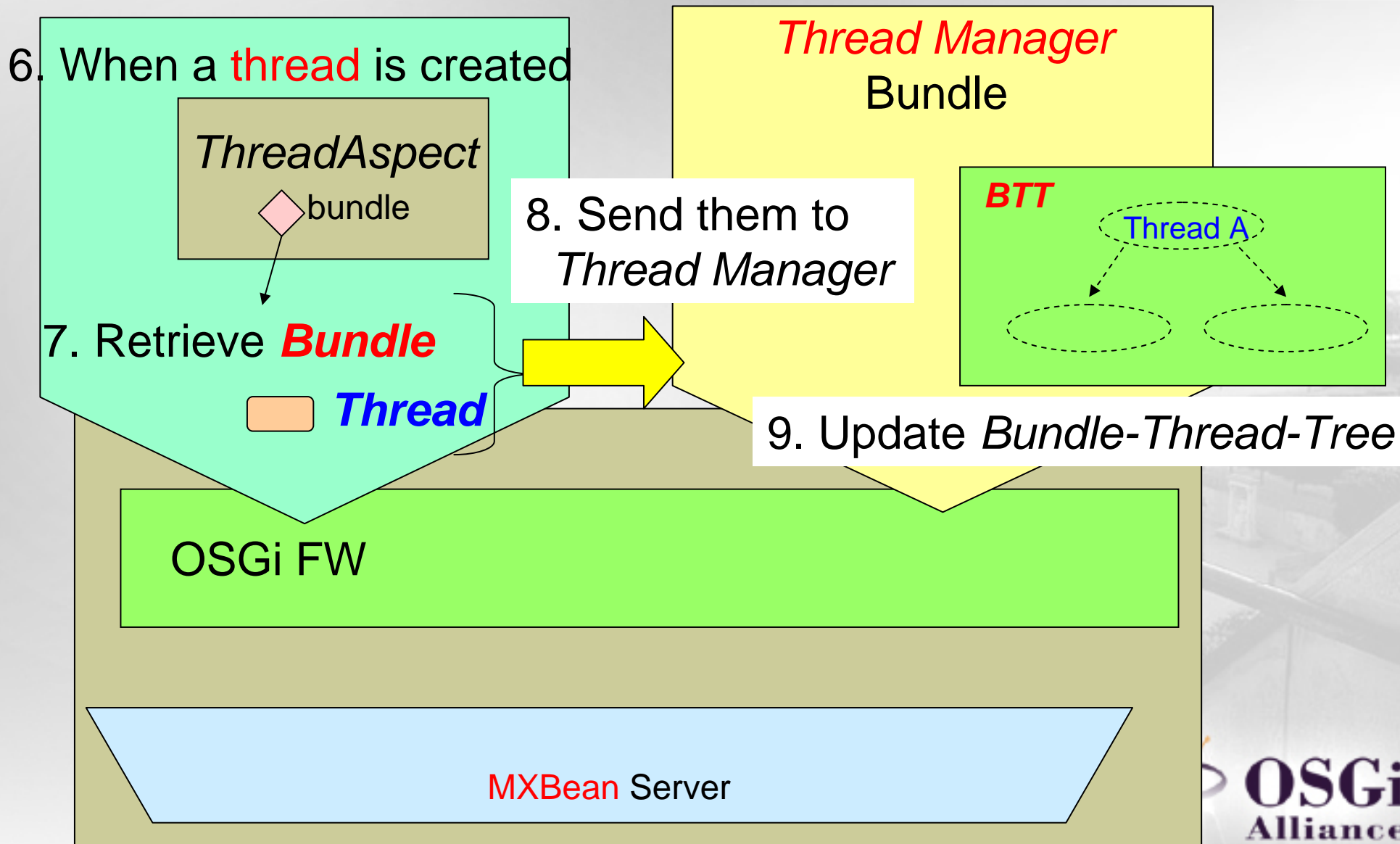
5. Store *Bundle* object

Thread Manager
Bundle

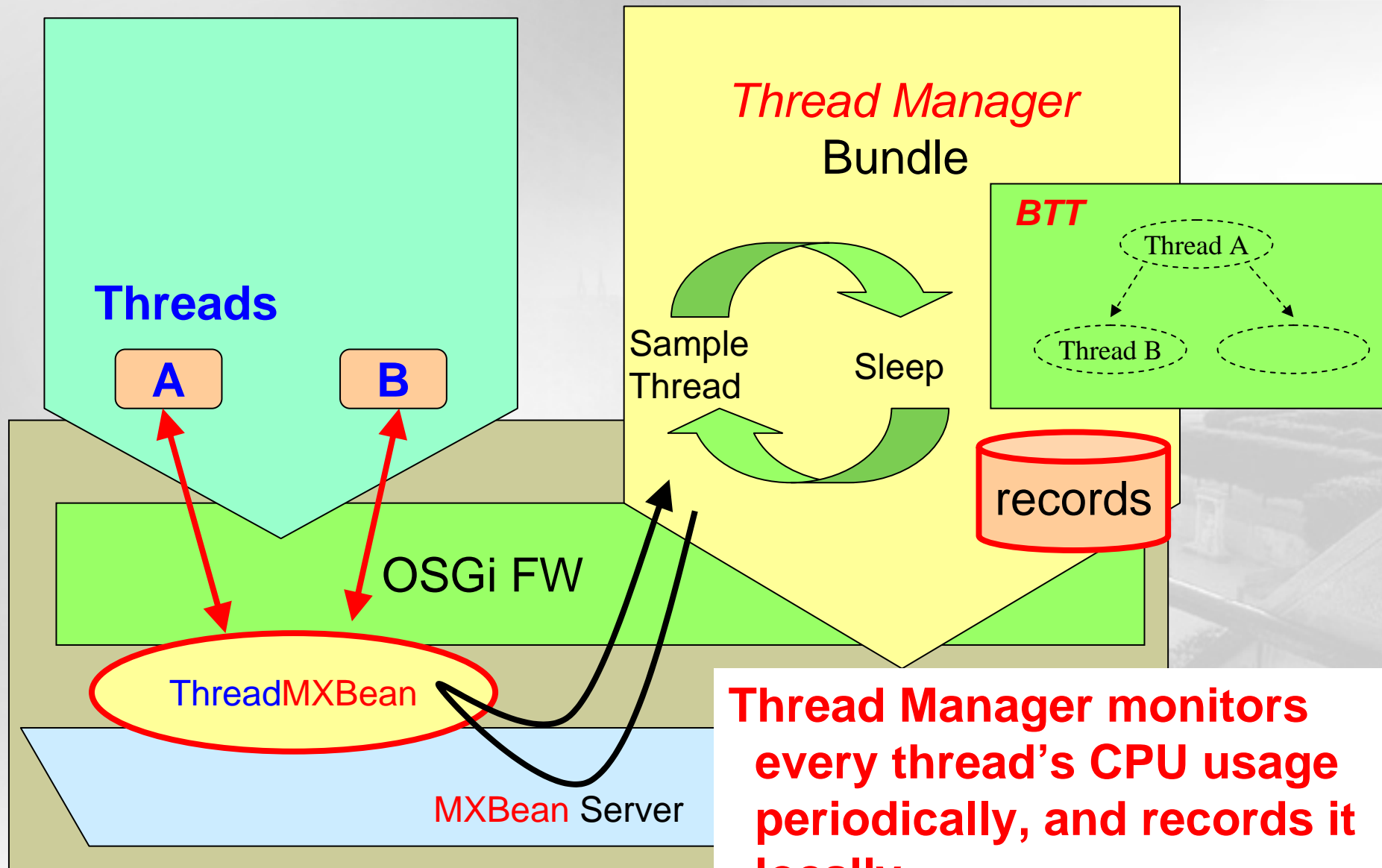
OSGi FW

MXBean Server

Operation of the System

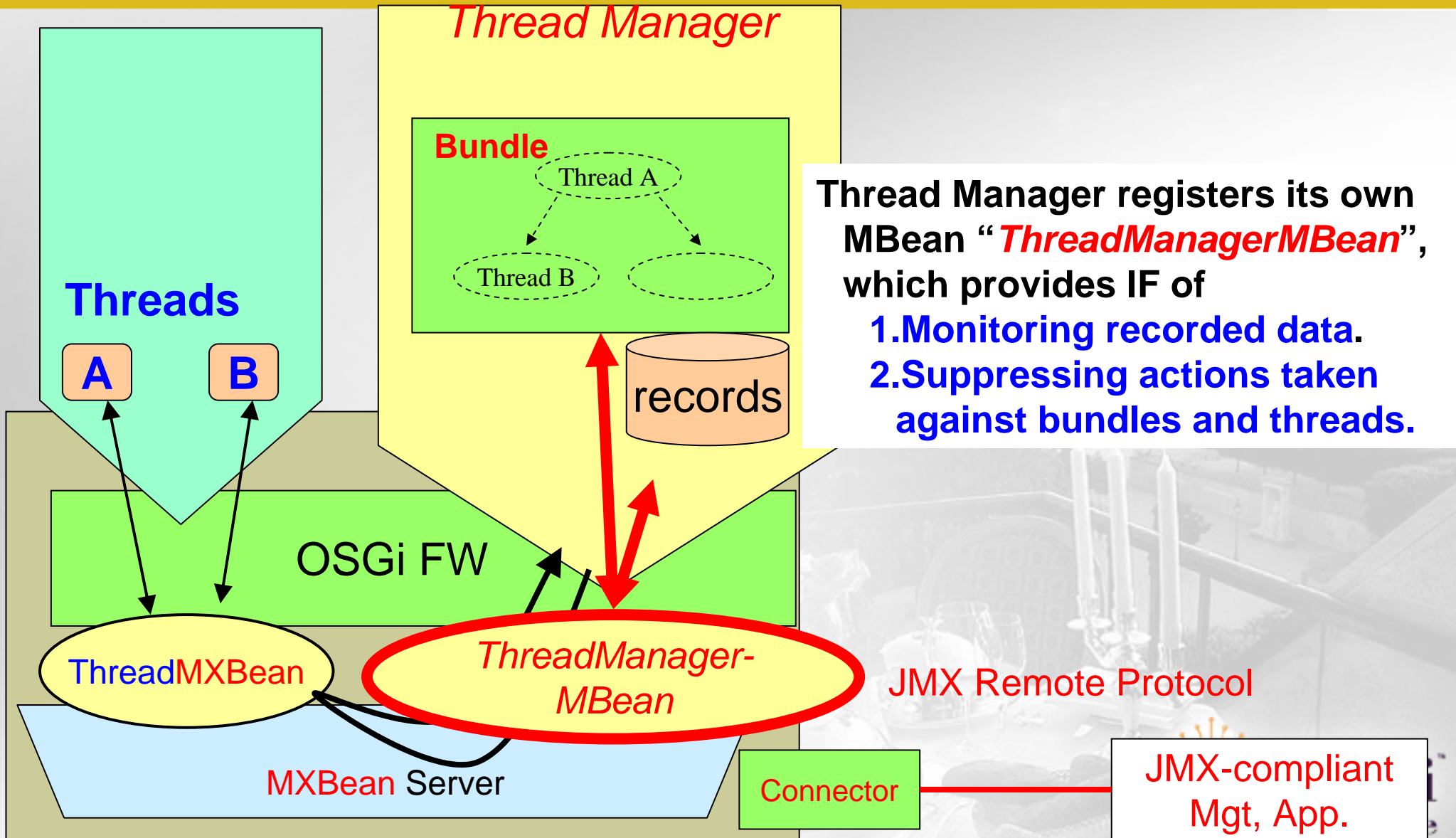


Operation of the System



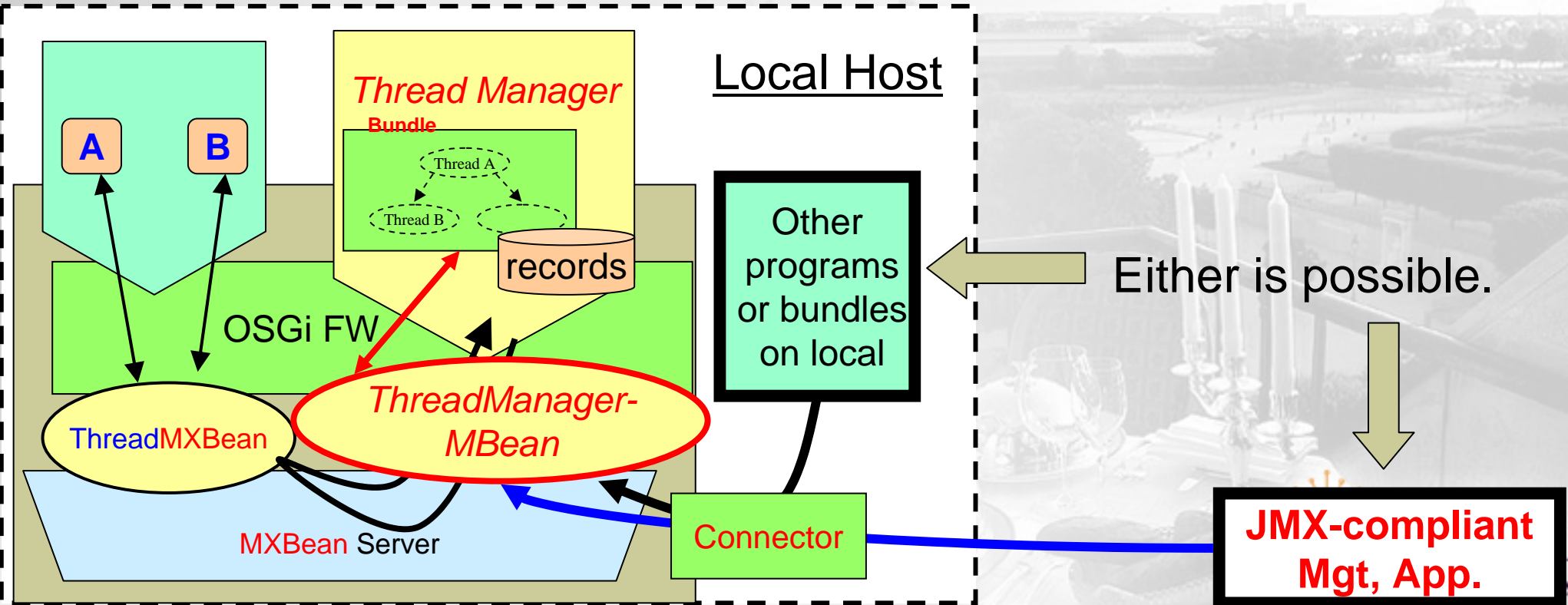
Thread Manager monitors every thread's CPU usage periodically, and records it locally.

ThreadManagerMBean



No restrictions on Mgt. Policies

- Diverse *mgt. policies* can be supported
 - Triggers
 - Reactions
 - Activators



Summary

- Background:
 - Threads created by bundle can deplete computer resources.
 - Monitoring and management system is required, especially in Service Aggregation Platform.
- Proposal:
 - CPU usage management system on OSGi FW adopting
 - **JMX in J2SE5.0**
 - Lightweight
 - Independent of specific vendors
 - **AOP (AspectJ)**
 - No rewrites of existing code needed
 - Proposed architecture can support diverse management policies
 - Trigger decision
 - Reactions to suppress the bundle and threads.
 - Activators (remotely and locally)