Lecture 4

New addition to RMI:
Remote Object Activation

Idea of activation

- Services are registered with rmiregistry but not instantiated; they are inactive until called upon by a client.

- Special daemon process, remote method invocation activation system daemon (rmid), listens for calls and instantiates RMI services on demand.

- Services are dormant until invoked and activated just in time for use by a RMI client.

RMI activation service

- Features of the service:
  - the first request for reference to the object automatically creates the remote object (also called *lazy activation)*;
  - activatable objects that run in the same JVM can be grouped in an “activation group”;
  - remote objects that were destroyed due to a system failure or exit, can be restarted.

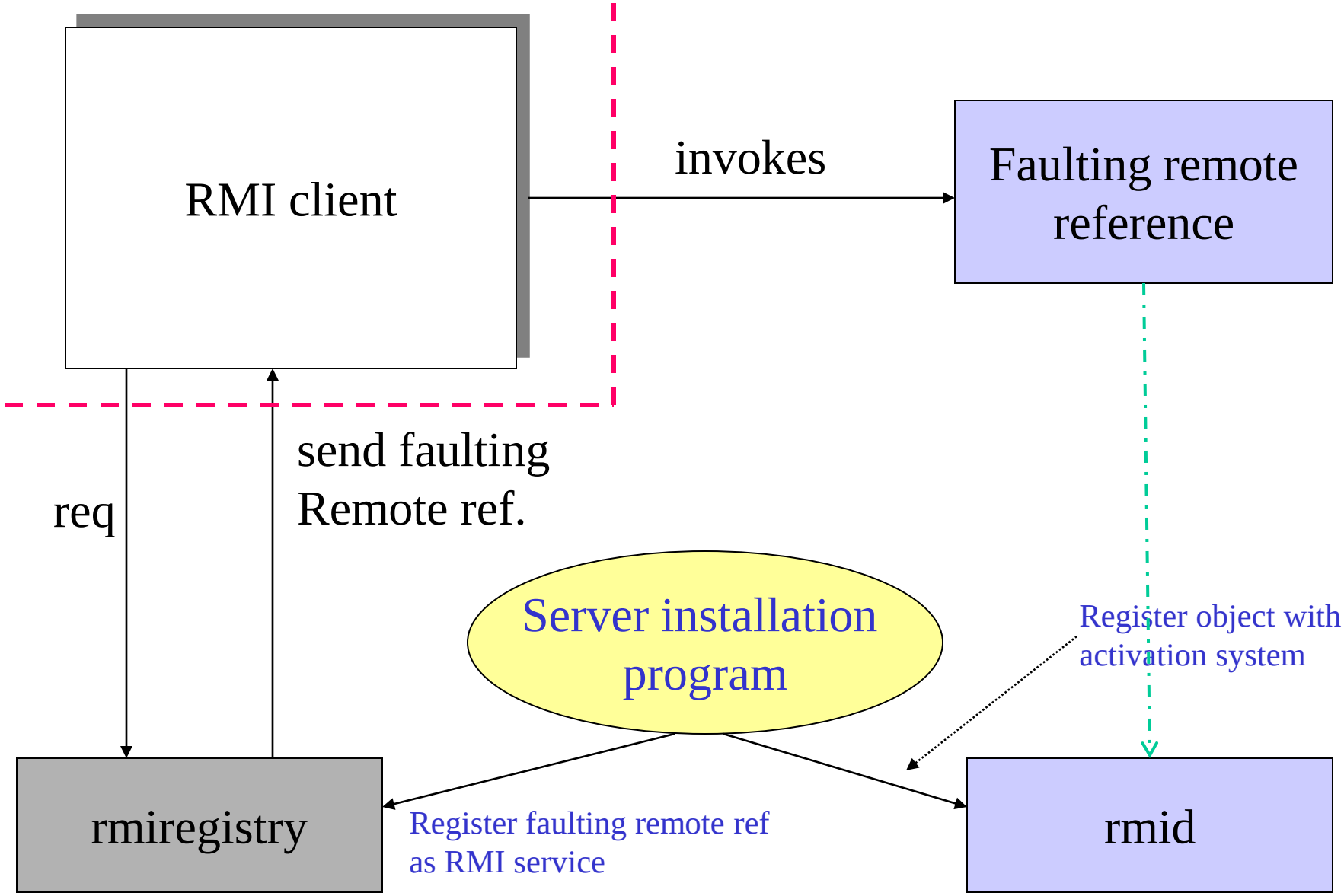
Faulting remote reference

- To be locatable, a service must register with rmiregistry – that normally requires the object to be instantiated.
- A *server installation program* notifies the activation system of an activatable remote object and registers the faulting remote reference with the rmiregistry.
- Object activation avoids instantiating RMI servers, activating them as required ➔ a faulting remote reference is registered.
- A faulting remote reference acts as a proxy between the remote client and the as-yet-unactivated server.

- The faulting reference maintains both
  - a *persistent handle* (an activation identifier), and
  - a *transient remote reference* to the target remote object.

- The remote object's *activation identifier* contains enough information to engage the activation server in activating the object.

- The *transient reference* is the actual "live" reference to the active remote object that can be used to invoke its methods.

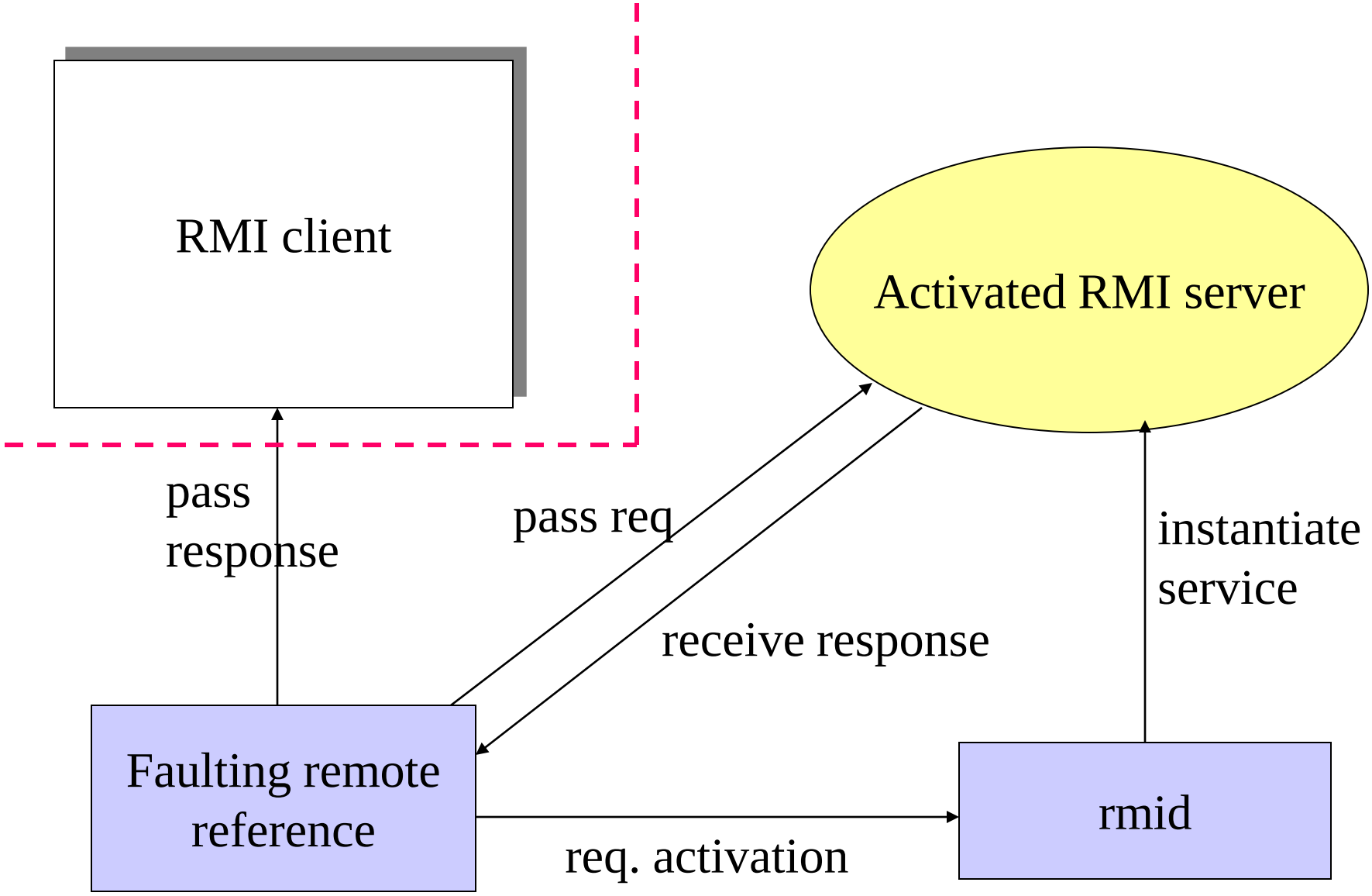

RMI client
invokes
Faulting remote reference
req
send faulting Remote ref.
Server installation program
Register object with activation system
rmiregistry
Register faulting remote ref as RMI service
rmid

How it works

- When the faulting reference is activated, it checks to see if it already has a reference to the server.

- The first time the faulting reference receives a call, the server must be activated before being used:
  - rmid creates a new instance of the server, and the call is passed to the newly activated object.
  - Methods' requests to the faulting reference are forwarded onward to the activated object and then results are returned back to the client.

- The client is at all times completely unaware of the details of the server implementation.


RMI client
Activated RMI server
pass
response
pass req
receive response
instantiate
service
Faulting remote
reference
rmid
req. activation

Creating an activatable remote object

- There is no difference between a normal remote interface and one implemented by an activatable object:

```
public interface MyRemoteInterface extends java.rmi.Remote
{
        public void doSomething () throws java.rmi.RemoteException;
}
```

- Then, the implementation that extends the java.rmi.activation.Activatable class must be created. It'll provide both a constructor and a doSomething() method.

Registration of activatable objects

1. Create an activation group descriptor that may include some properties.

   ActivationGroupDesc groupDescriptor = new ActivationGroupDesc (new Properties(), null);

2. Register the activation group descriptor with the RMI activation daemon process. This process is represented by the Activation System interface. Applications do not create or extend this interface. Instead a ref to the activation daemon process is obtained:

```
ActivationSystem system = activationGroup.getSystem();
```

Once a ref to the ActivationSystem class is obtained, the group descriptor is registered using the ActivationSystem.registerGroup method, which returns an ActivationGroupID instance.

```
ActivationGroupID groupID =
    system.registerGroup(groupDescriptor);
```

3. Actually, it creates the activation group. This creates and assigns an activation group for the JVM, and notifies the activation system that the group is active:

```
ActivationGroup.createGroup(groupID, groupDescriptor, 0);
```

4. Create an activation descriptor that describes an activatable remote object.

   It stores the class name of the remote object, the URL , (optionally) a MarshalledObject that contains a serialized version of the remote object and the group assignment for the object.

```
ActivationDesc desc = new ActivationDesc (groupID
        "ActivatableLightBulbServer", strLoc, null);
```

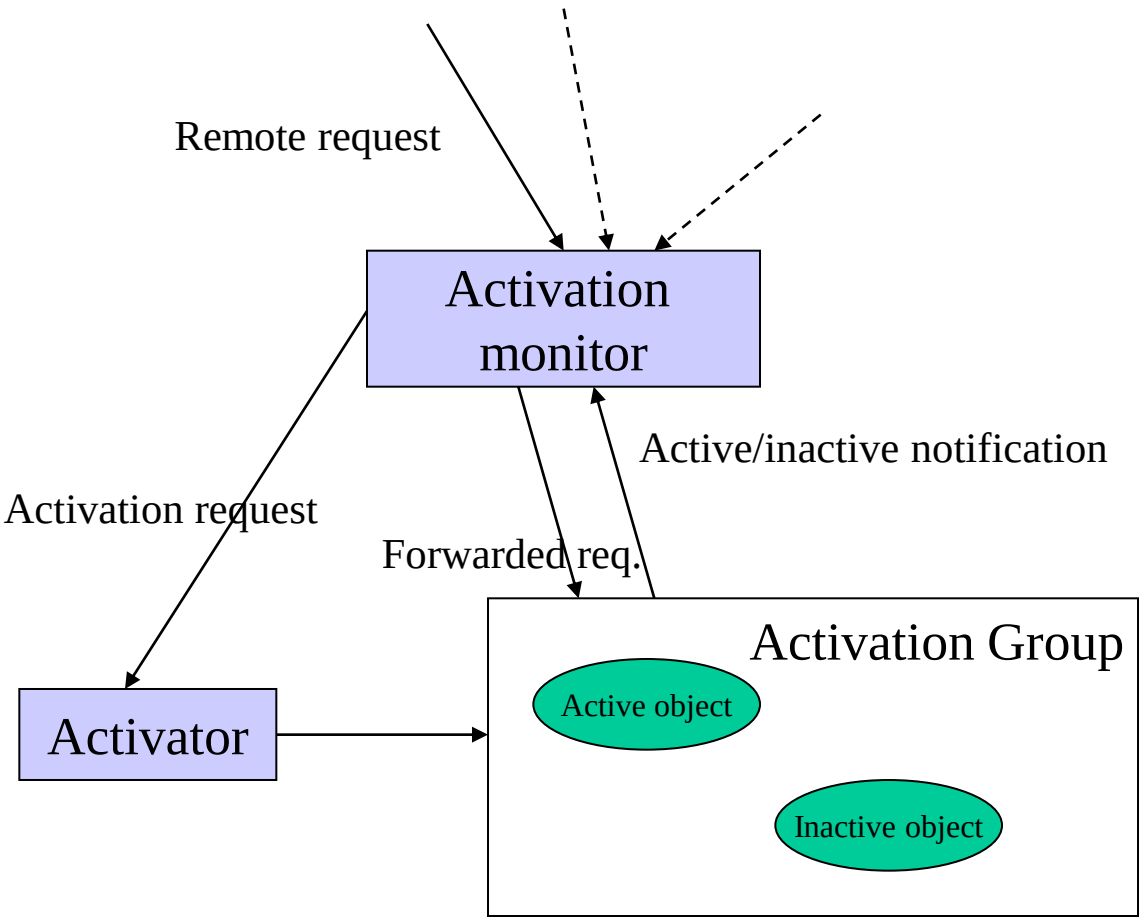
5. Register the activation descriptor with the activation system – the activation system learns the name and location of the class definition file. A remote stub will be returned, which can be registered as a RMI service:

```
Remote stub = Activatable.register(desc);
```

6. Add a registry entry for the RMI service, so that clients may locate the service – a remote stub is registered.

Naming.rebind(registration, stub)

Activation system at work



Activation

- When an object needs to be activated, the activation system first looks up the ActivationDesc for the object and then looks for the class referenced in the ActivationDesc, using the URL to load the class bytecodes.

- Once the class has been loaded, the activation system creates an instance of the class by calling the activation constructor, which takes an ActivationID and a MarshalledObject as arguments.

- The ActivationID is issued by the activation system, and the MarshalledObject contains the data previously registered with the ActivationDesc.

Applications

- Consider the following app: call home and activate the security cameras, then one receives video streams on the mobile phone, or

- The intrusion detection system detects suspect movements and calls mobile phone where the video player is activated.

Conclusions

- Activatable objects can be used in complex distributed systems where some services are used seldom. By keeping them dormant, host's resources are saved.

- In addition, after some time of inactivity, the server can be shut down.

- The process of activating an activatable service is transparent to the client, however its registration is a supplementary operation.