Jini

Jini

- Platform (“middleware”) for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications

Jini serves as an example for a number of similar platforms (UPnP, Bluetooth SDP, SLP, HAVi, Salutation, e-speak, HP Chai,…)

Jini

- Platform ("middleware") for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications

  - framework of APIs with useful functions / services
  - helper services (discovery, lookup,...)
  - suite of standard protocols and conventions

Jini

- Platform (“middleware”) for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications

  - services, devices, ... find each other automatically (“plug and play”)
  - dynamically added / removed components
  - changing communication relationships
  - mobility

Jini

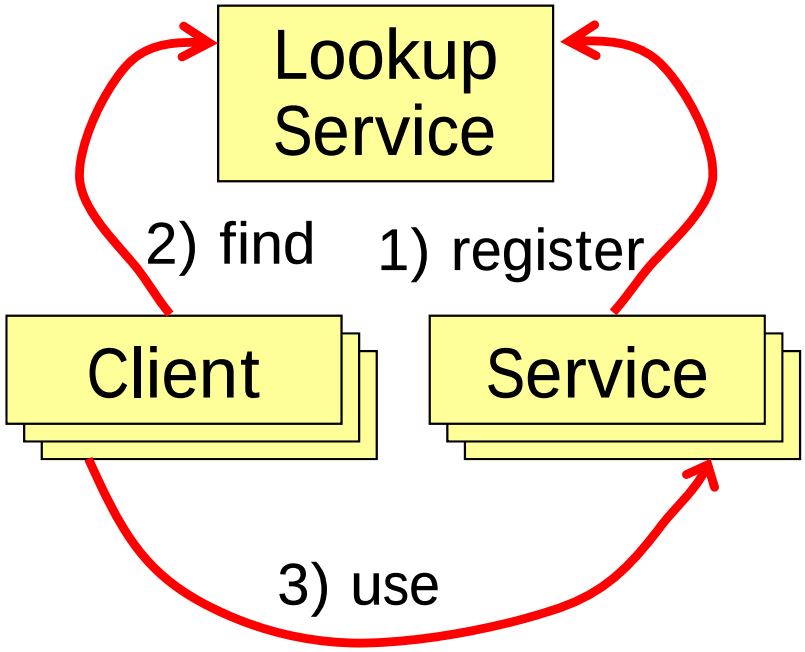
- Platform (“middleware”) for dynamic, cooperative, spontaneously networked systems
  - facilitates implementation of distributed applications
- Based on Java
  - uses RMI (Remote Method Invocation)
  - code shipping
  - requires JVM / bytecode everywhere
- Service-oriented
  - (almost) everything is considered a service

Service Paradigm

- Almost everything relevant is a service
- Jini's run-time infrastructure offers mechanisms for adding, removing, finding, and using services
- Services are defined by interfaces and provide their functionality via their interfaces
  - services are characterized by their type and their attributes (e.g., "600 dpi", "version 21.1")
- Services (and service users) may "spontaneously" form a so-called federation

Jini: Global Architecture

- Lookup Service (LUS)
  - main registry entity and brokerage service for services and clients
  - maintains information about available services
- Services
  - specified by Java interfaces
  - register together with proxy objects and attributes at the LUS
- Clients
  - know the Java interfaces of the services, but not their implementation
  - find services via the LUS
  - use services via proxy objects



Network Centric

- Jini is based on the network paradigm
  - network = hardware and software infrastructure
- View is “network to which devices are connected to”, not “devices that get networked”
  - network always exists, devices and services are transient
- Jini supports dynamic networks and adaptive systems
  - adding and removing components or communication relations should only minimally affect other components

Spontaneous Networking

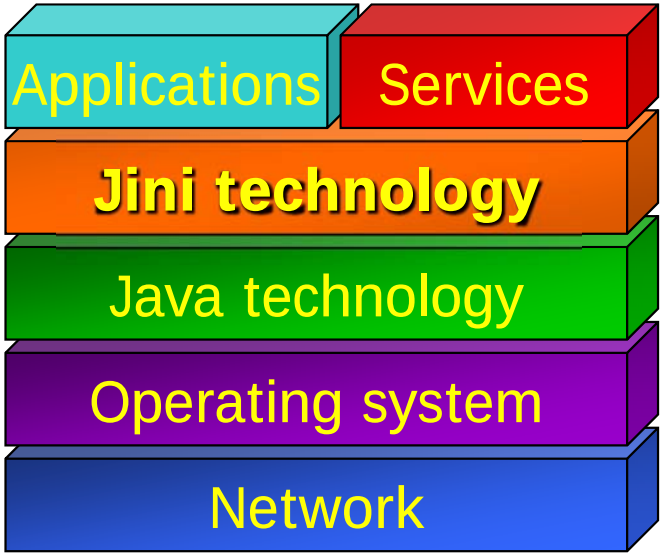
- Objects in an open, distributed, dynamic world find each other and form a transitory community
  - cooperation, service usage, ...
- Typical scenario: client wakes up (device is switched on, plugged in, ...) and asks for services in its vicinity
- Finding each other and establishing a connection should be fast, easy, and automatic

Some Fallacies of Common Distributed Computing Systems

- The “classical” idealistic view...
  - the network is reliable
  - latency is zero
  - bandwidth is infinite
  - the network is secure
  - the topology is stable
  - there is a single administrator
- ...isn’t true in practice
  - Jini acknowledges and addresses some of these issues

Bird's-Eye View on Jini as a Middleware Infrastructure

- Jini consists of a number of APIs
- Is an extension to the Java platform dealing with distributed computing
- Is an abstraction layer between the application and the underlying infra-structure (network, OS)



Jini's Use of Java

- Jini requires JVM (as bytecode interpreter)
  - homogeneity in a heterogeneous world

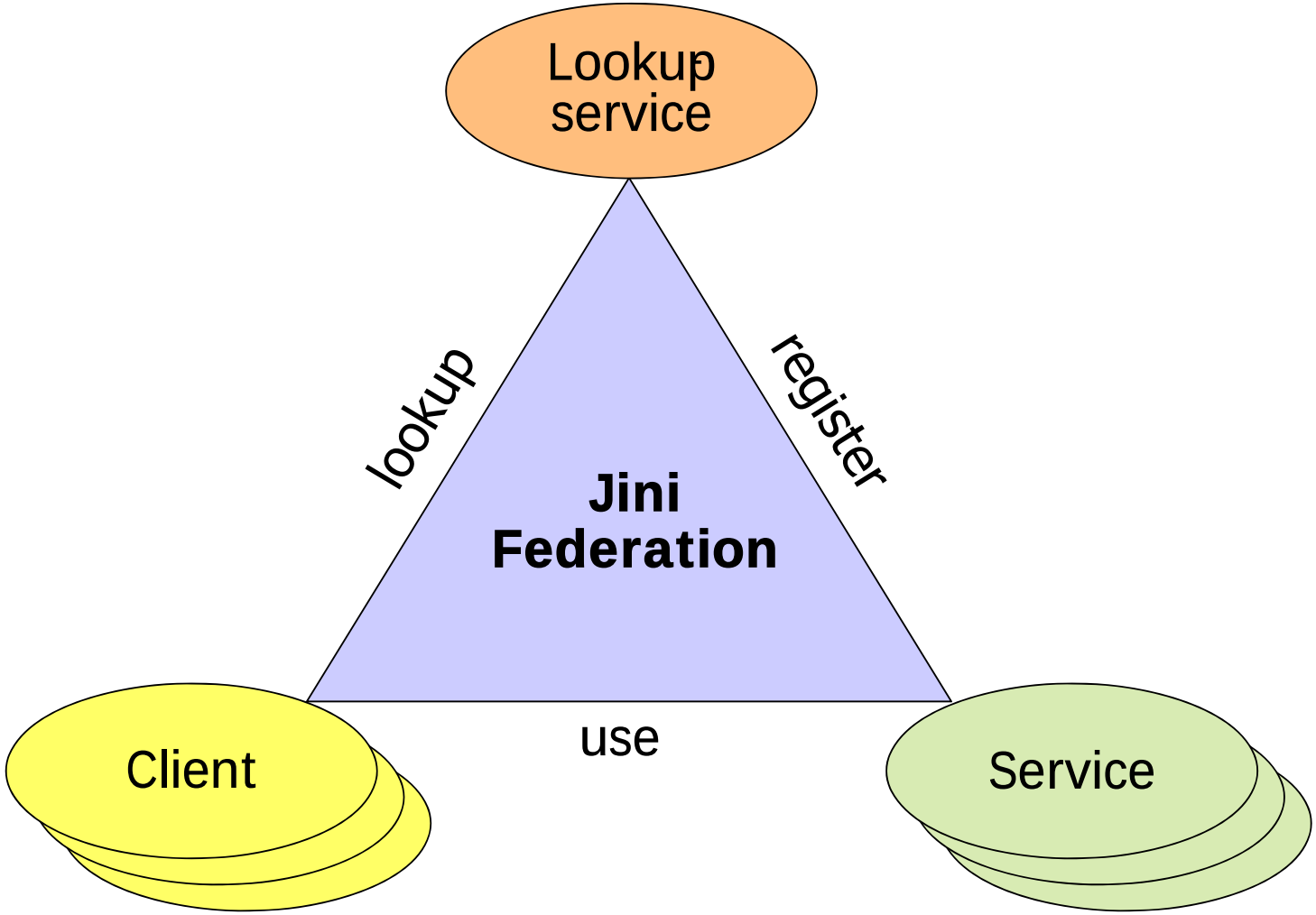
run protocols for discovery and join; have a JVM

- Devices that are not "Jini-enabled" have to be managed by a Jini-enabled software proxy (somewhere in the net)

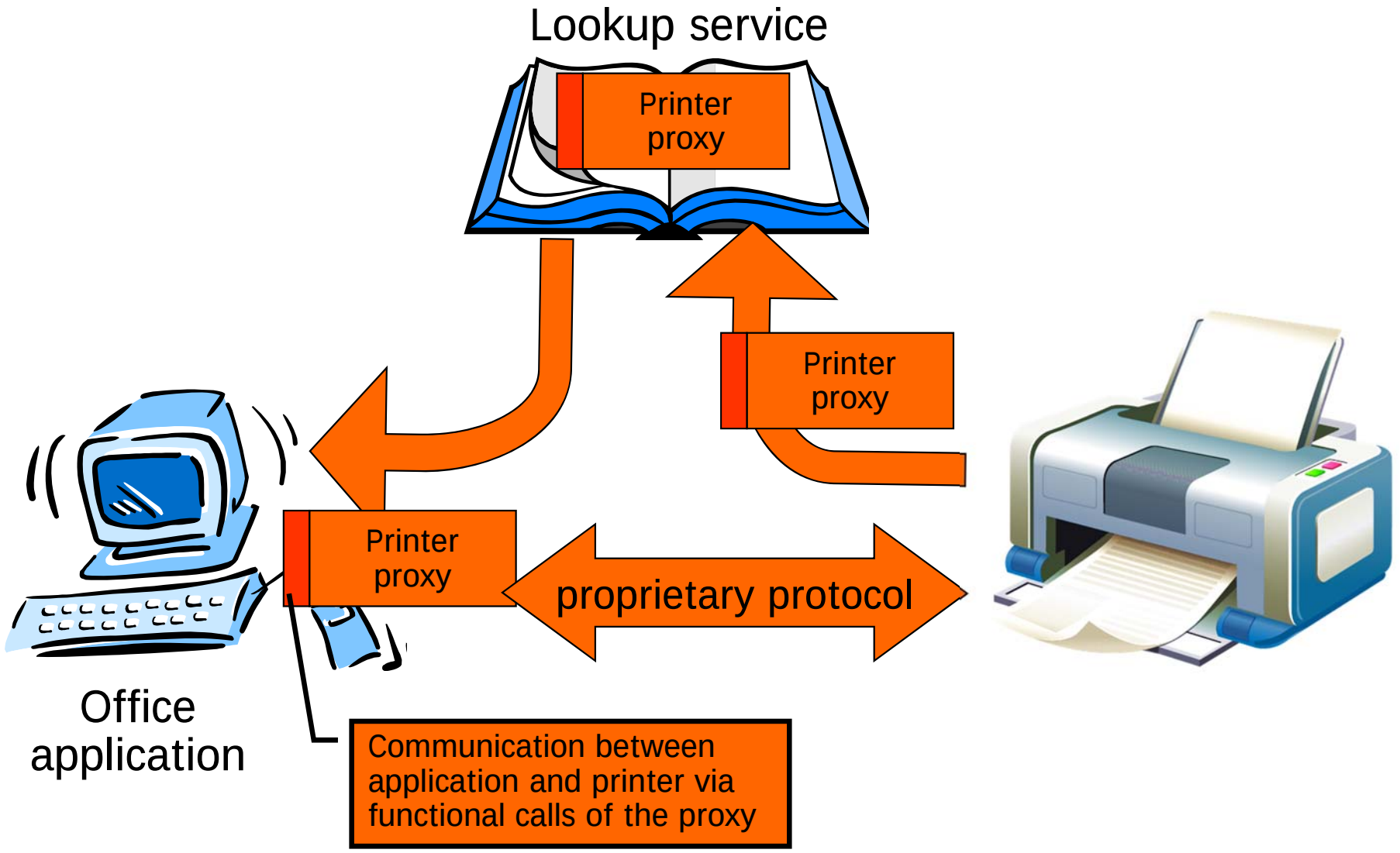
Main Components of the Jini Infrastructure

- Lookup service (LUS)
  - as repository / naming service / trader
- Protocols
  - discovery & join, lookup of services
  - based on TCP/UDP/IP
- Service proxy objects
  - transferred from service to clients (via LUS)
  - represent the service locally at the client

Lookup Service



Example


Lookup service
Printer proxy
Printer proxy
Printer proxy
proprietary protocol
Office application
Communication between application and printer via functional calls of the proxy

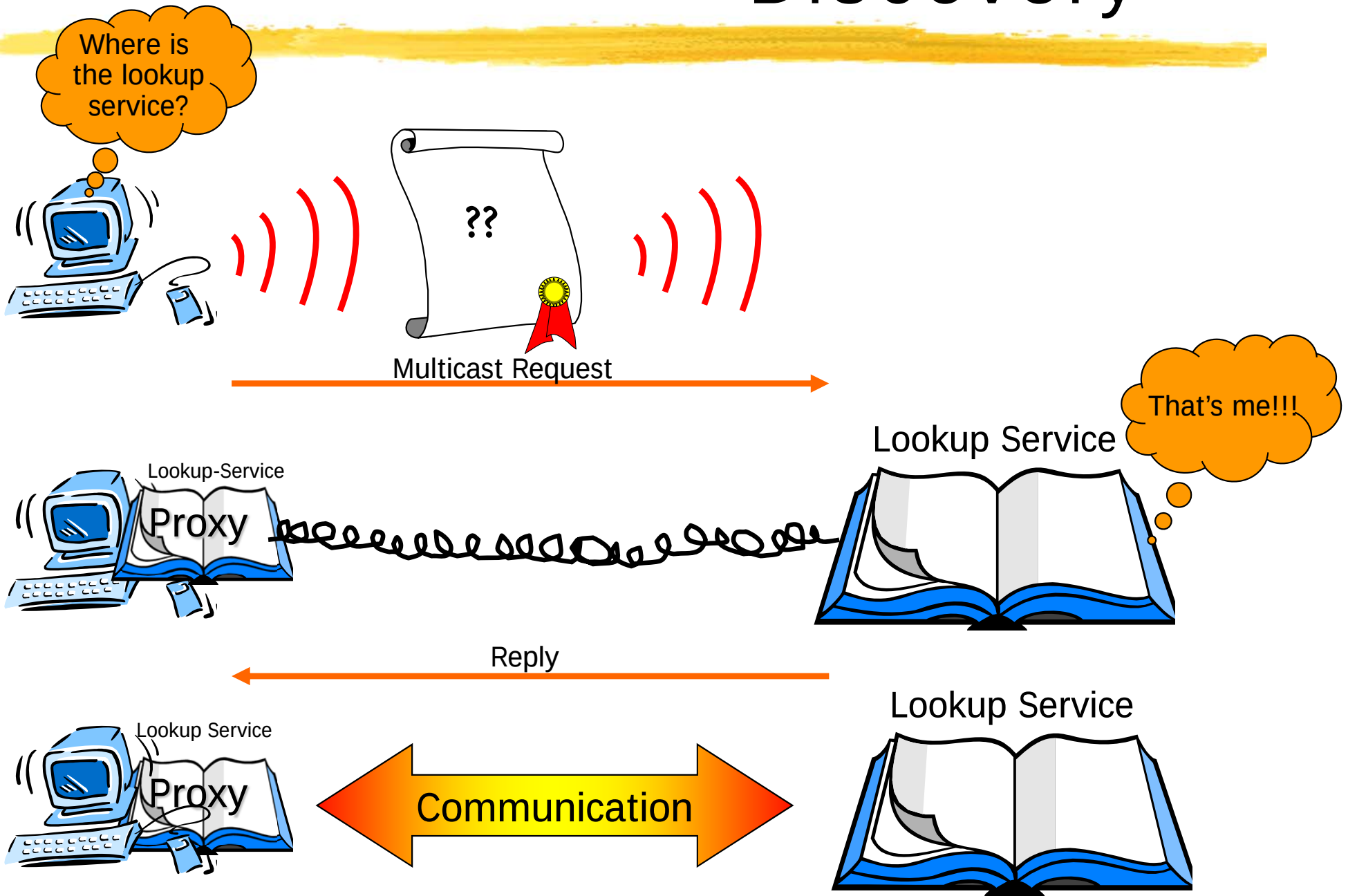
Lookup Service

- Uses Java RMI for communication
  - objects („proxies“) can migrate over the network
- Stores besides the name/address of a service:
  - set of attributes
    - e.g., printer(color: true, dpi: 600, ...)
  - proxies, which may be complex classes
    - e.g., user interfaces
- Further possibilities:
  - responsibility can be distributed to a number of (logically separated) lookup services
  - increase robustness by running redundant lookup services

Discovery: Finding a LUS

- Goal: Find a lookup service (without knowing anything about the network) to
  - advertise (register) an application service, or
  - find (look up) an existing application service
- Discovery protocol:
  - multicast to well-known address/port
  - lookup service replies with a serialized object (its proxy)
    - from then on communication with the LUS is via this proxy

Discovery


Where is the lookup service?
??
Multicast Request
That's me!!!
Lookup Service
Lookup-Service
Proxy
Reply
Lookup Service
Lookup Service
Proxy
Communication

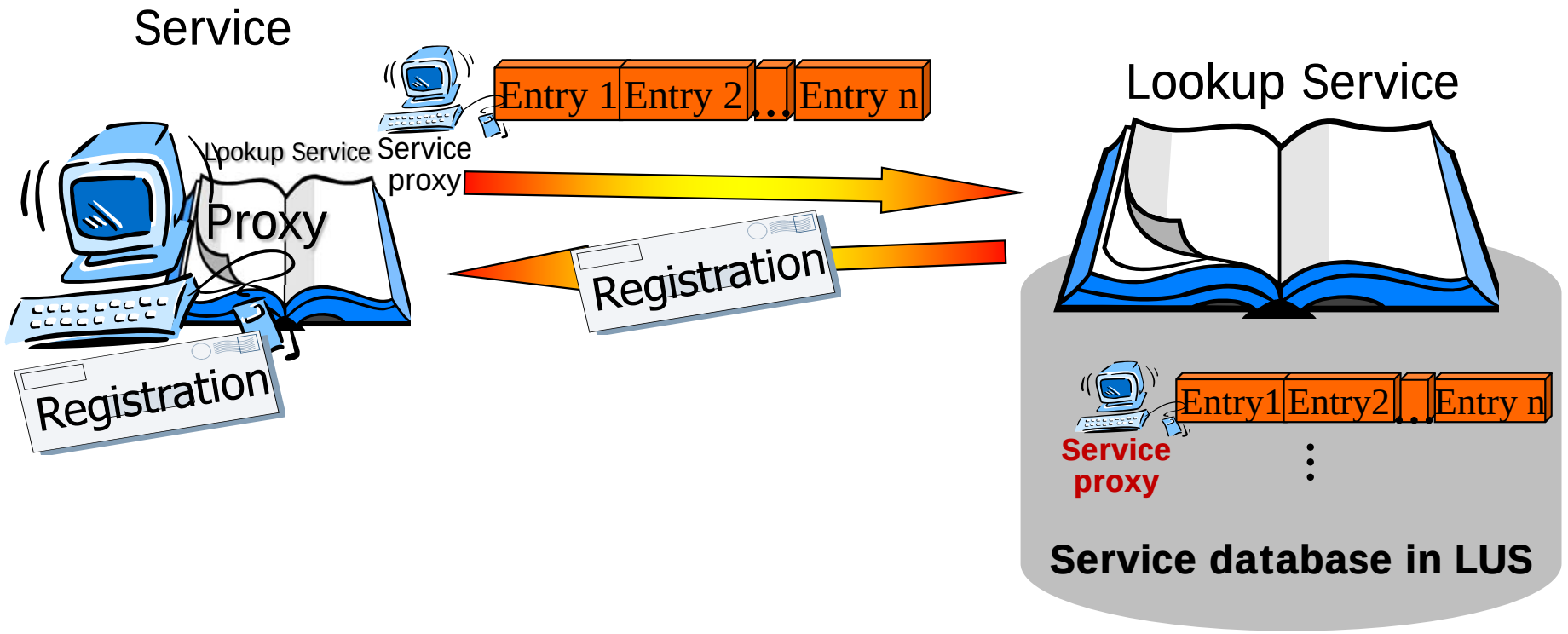
Multicast Discovery Protocol

- Search for lookup services
  - no information about the host network needed
- Discovery request uses multicast UDP packets
  - multicast address for discovery (224.0.1.85)
  - default port number of lookup services (4160)
  - recommended time-to-live is 15
  - usually does not cross subnet boundaries
- Discovery reply is establishment of a TCP connection
  - port for reply is included in multicast request packet

Join: Registering a Service

- Assumption: Service provider already has a proxy of the lookup service (→ discovery)
- It uses this proxy to register its service
- Gives to the lookup service
  - its service proxy
  - attributes that further describe the service
- Service provider can now be found and its service be used in this Jini federation

Join


Service
Entry 1
Entry 2
Entry n
Lookup Service
Service proxy
Proxy
Registration
Registration
Entry1
Entry2
Entry n
Service proxy
Service database in LUS

Join: More Features

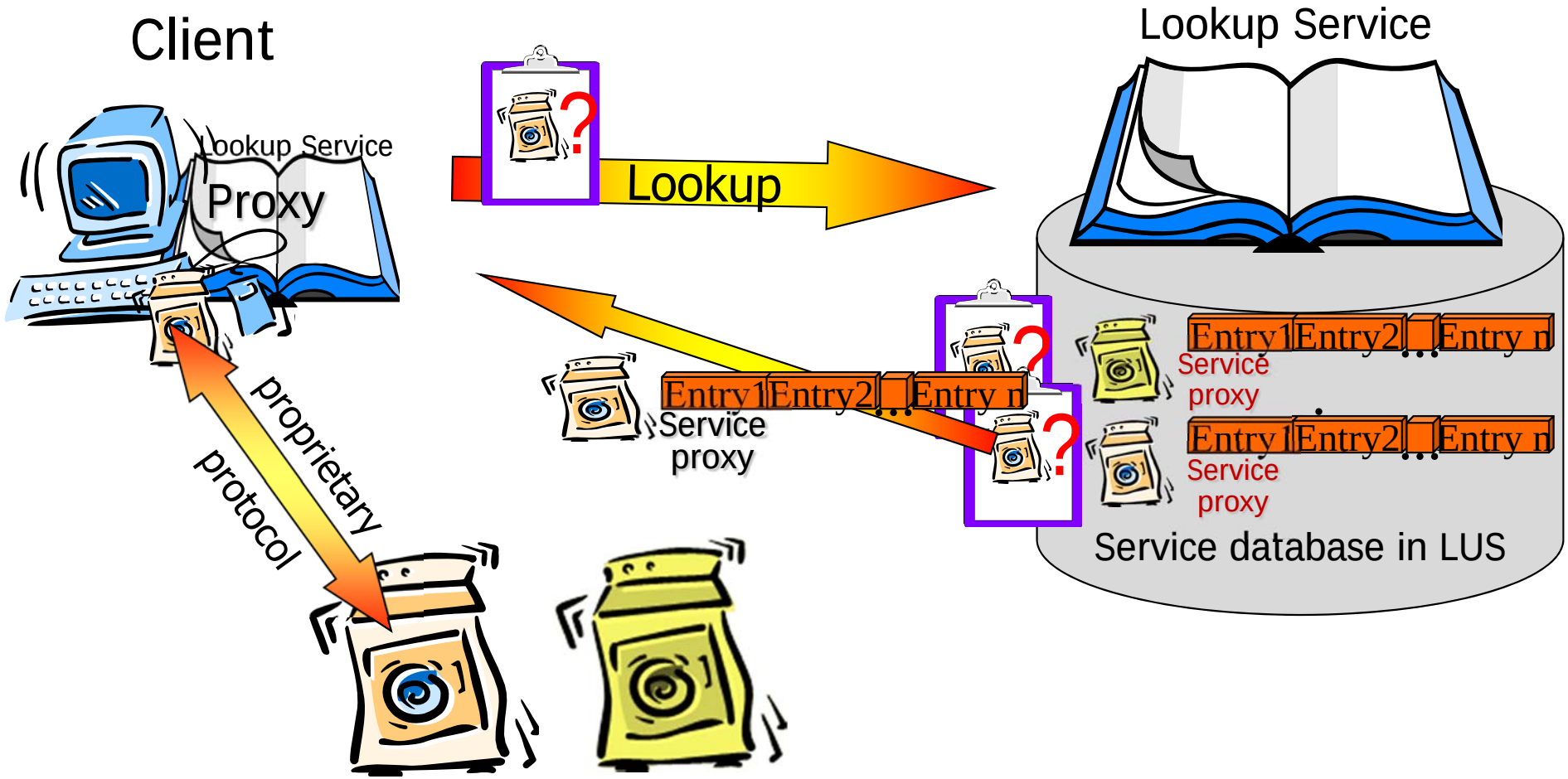
- To join, a service supplies:
  - its proxy
  - a ServiceID (a “universally unique identifier”)
  - a set of attributes
- Service waits a random amount of time after start-up
  - prevents packet storms after restarting a network segment
- Registration with a lookup service is bound to a lease
  - service has to renew its lease periodically

Lookup: Searching Services

- Client creates query for lookup service
  - matching by registration number of service (ServiceID) and/or service type and/or attributes
  - wildcards are possible („null“)
- Via its proxy at the client, the lookup service returns zero, one or more matches (i.e., server proxies)
- Selection among several matches is done by client

- Client uses the service by calling functions of the service proxy
- Any proprietary protocol between service proxy and service provider is possible

Lookup


Client
Lookup Service
Proxy
Lookup
Lookup Service
Entry1
Entry2
Entry n
Service proxy
Entry1
Entry2
Entry n
Service proxy
Entry1
Entry2
Entry n
Service proxy
Service database in LUS
proprietary protocol

Proxies

- Proxy object is stored in the LUS upon registration
  - serialized object
  - implements the service interfaces
- Upon request, service proxy is sent to the client
  - client communicates with service implementation via its proxy: client invokes local methods of the proxy object
  - proxy implementation hidden from client

Smart Proxies

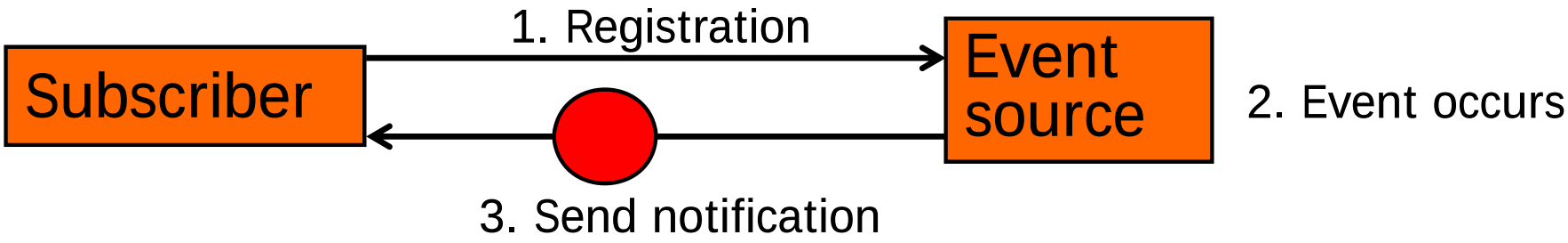
- Parts of (or the whole) service functionality may be executed by the proxy at the client
  - example: when dealing with large volumes of data, it usually makes sense to preprocess parts of the data (e.g., compressing video data before transfer)
- Partition of service functionality depends on service implementer's choice
  - client needs appropriate resources

Leases

- Leases are contracts between two parties
- Leases introduce a notion of time
  - resource usage is restricted to a certain time frame
- Repeatedly express interest in some resource:
  - I'm still interested in X
    - renew lease periodically
    - lease renewal can be denied
  - I don't need X anymore
    - cancel lease or let it expire

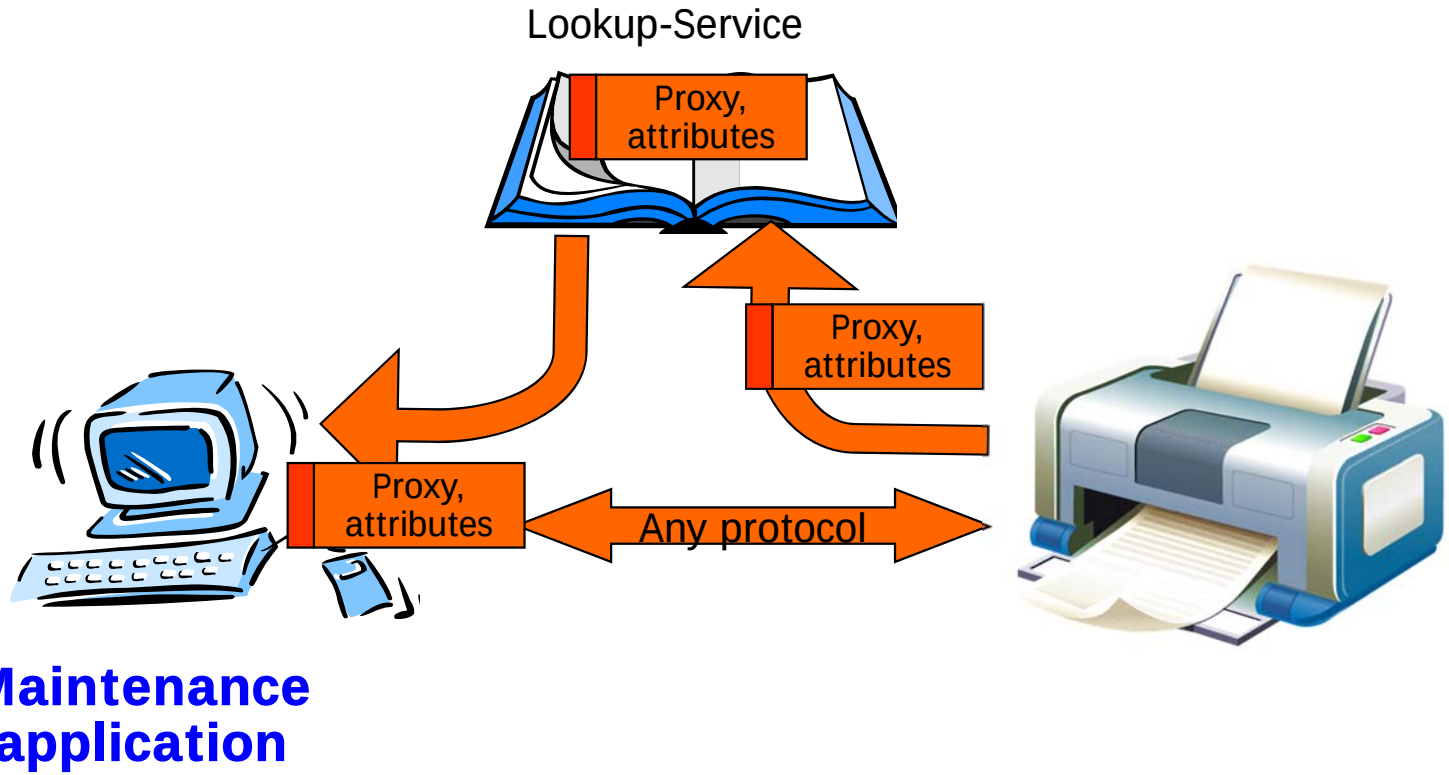
Distributed Events

- Objects on one JVM can register interest in certain events of another object on a different JVM
- "Publisher/subscriber" model



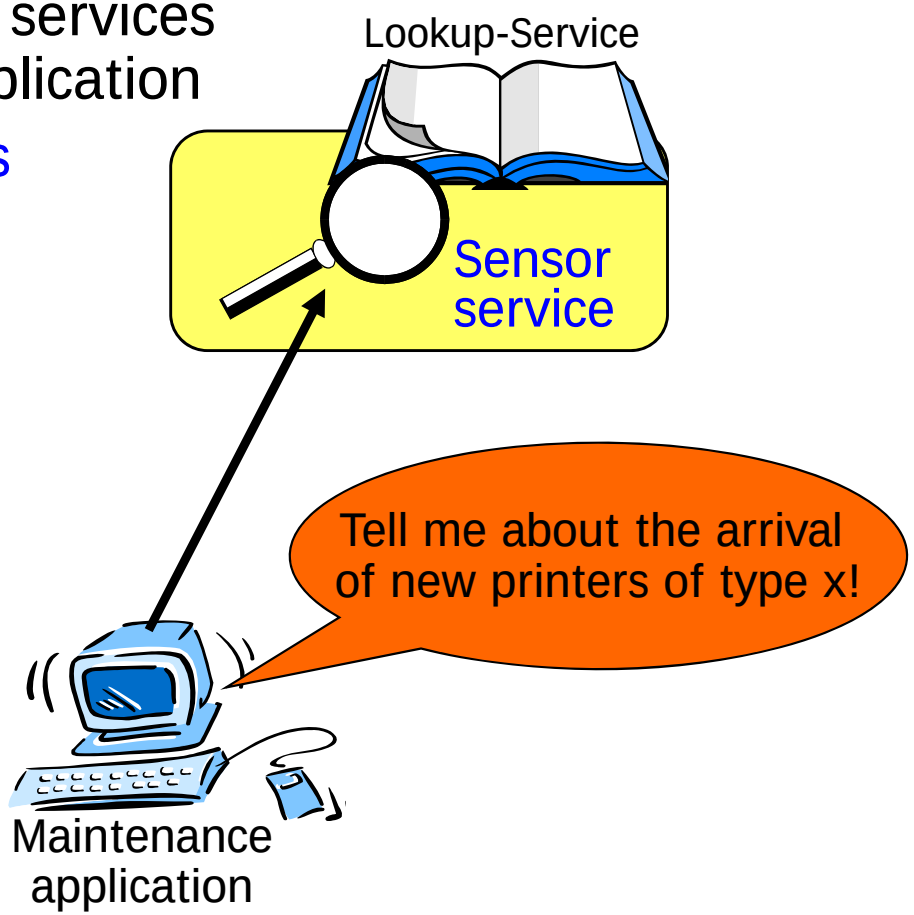
Distributed Events – Example

- Example: printer is plugged in
  - printer registers itself with local lookup service
- Maintenance application wants to update software



Distributed Events – Example

- Search for printers is “outsourced” to the lookup service
  - “sensor service” looks for certain services on behalf of the maintenance application
  - maintenance application registers for events showing the arrival of certain types of printers
  - sensor service observes the lookup service
  - notifies application as soon as matching printer arrives via distributed events



Distributed Events – Example

- Example: printer arrives, registers with lookup service
  - printer performs discovery and join
  - sensor finds new printer in lookup service
  - checks if there is an event registration for this type of printer
  - notifies all interested objects
  - maintenance application retrieves printer proxy and updates software