JNDI and JMS

Cloud Computing are Pervasive!

- Major players include

the rackspace cloud

Windows Azure

Google Cloud Platform

amazon web services™

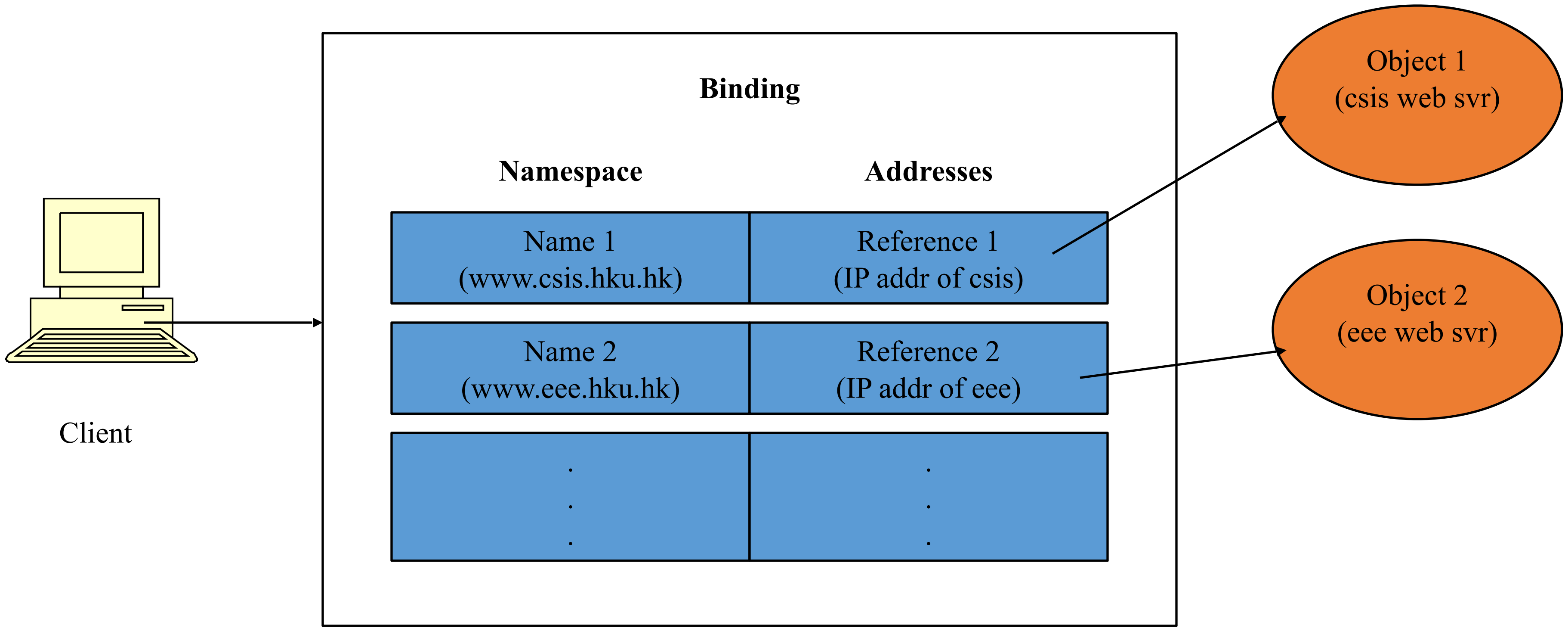
"Messaging" in Clouds are Essential Communications within/among Apps

- Major users include products / companies such as

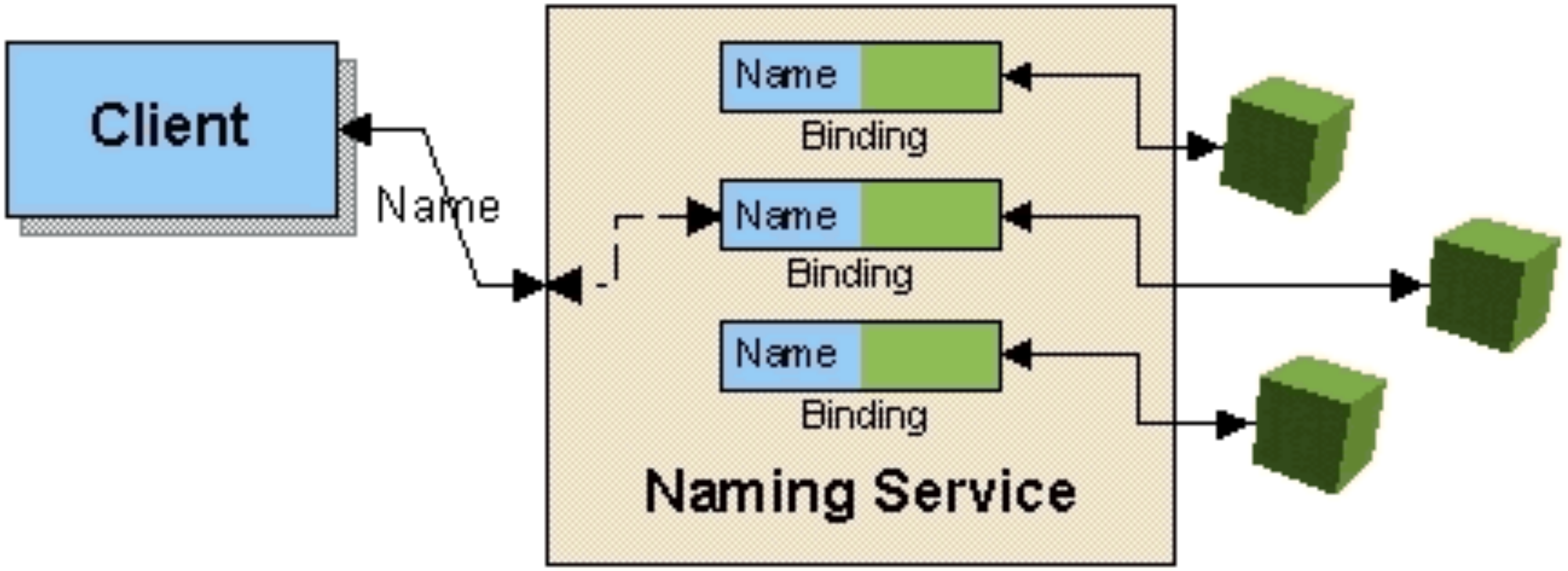
Java Naming and Directory Interface (JNDI)

Functions of Naming Service

- *bind*: bind a name to an object or object reference (information about how to access an object)
- *lookup*: looks up an object or object reference by its name

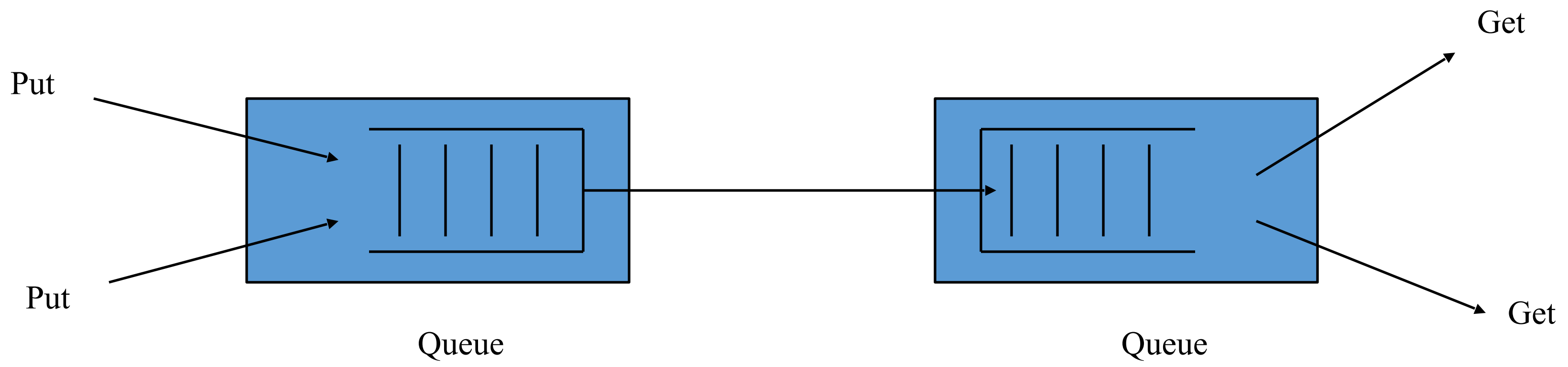


Naming Service: Binding names to objects



Message Queuing (A Preview)

- Program-to-queue communication
- Actions:
  - Put: puts a message into the queue
  - Get: takes a message out of the queue
- Message queue software:
  - Transfer of messages from queue to queue
  - Ensures message arrives eventually and only one copy of the message is placed in the destination queue



JMS And JNDI (Preview of the Two's Relation)

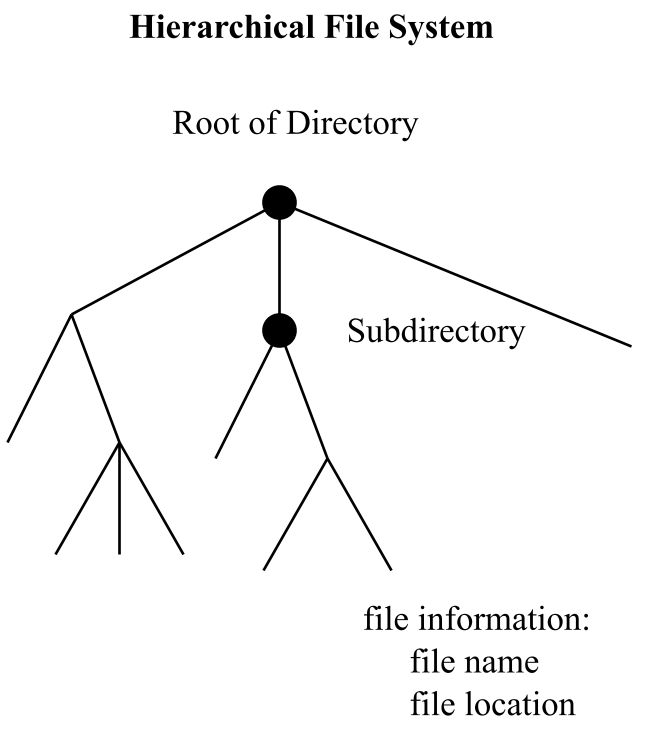
- JNDI provides a mechanism by which clients can perform a "lookup" to find the correct information to connect to each other.
- Using JNDI provides the following advantages:
    - It hides provider-specific details from JMS clients.
    - It abstracts JMS administrative information into Java objects.
    - Since there will be JNDI providers for all popular naming services, this means JMS providers can deliver one implementation of administered objects that will run everywhere. Thereby eliminating deployment and configuration issues.

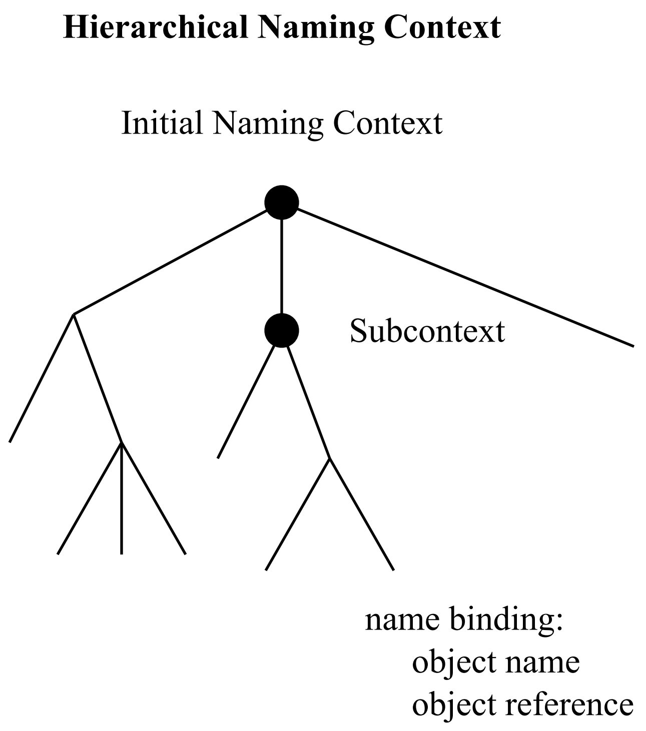
Naming Service

- Associates names with objects, which can then be located by names, e.g. catalog in a library to locate books
- In a large enterprise, an application may need to access services provided by other applications $\Rightarrow$ need a naming service to locate these services
- Naming service can be centralized or distributed
- Namespace: set of names in a naming service
- Naming context: a set of name-to-object bindings sharing the same naming convention
  - Context object: an object with methods to bind names, unbind names, rename objects and list existing bindings
  - Subcontext: a name in a context is also bound to another context with the same naming conventions

Hierarchical Naming Context

- Name contexts that organized in a hierarchy
- Root is called the initial naming context, serves as a starting point to resolve names, e.g. root of Unix file system is /



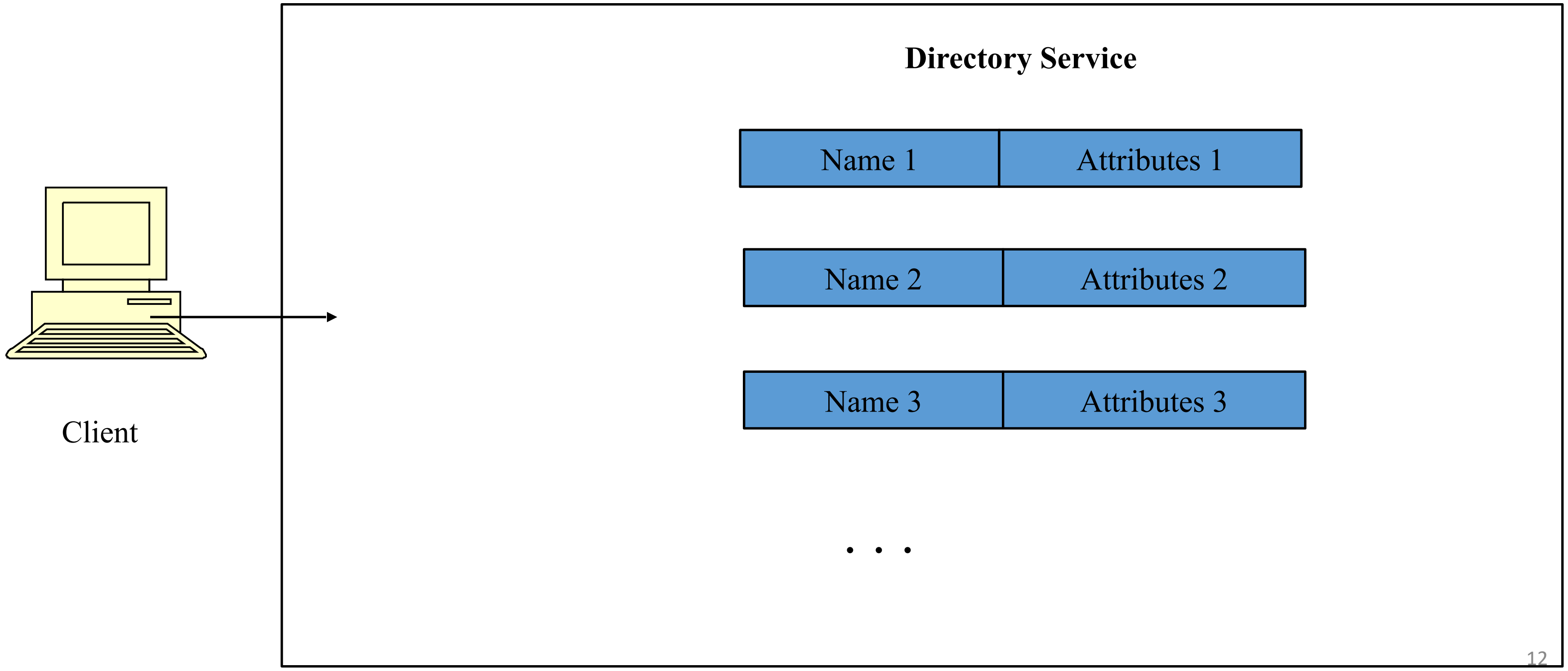


Directory Service

- Directory service: stores, distributes, searches and retrieves objects
    - Directory object: object in the directory, also called directory entry
    - An extension of a naming service: not only bind objects with names, but also contains attributes for each directory object
    - Attribute: provides the description about the object
    - Object can be search based on its attributes
- E.g. a book in a library can be modeled as a directory object that has the following attributes:
    - Name of the book, names of authors, category, abstract, publisher, date of publication, ISBN, call numbers of copies, locations of copies, …
- An attribute has an attribute identifier and a set of attribute values, e.g. Book “Java Enterprise in a Nutshell”
    - Attribute identifier: *authors*
    - Values: *David Flanagan*, *Jim Fareley*, *William Crawford* and *Kris Magnusson*
- Directory service can be centralized or distributed

Directory Search

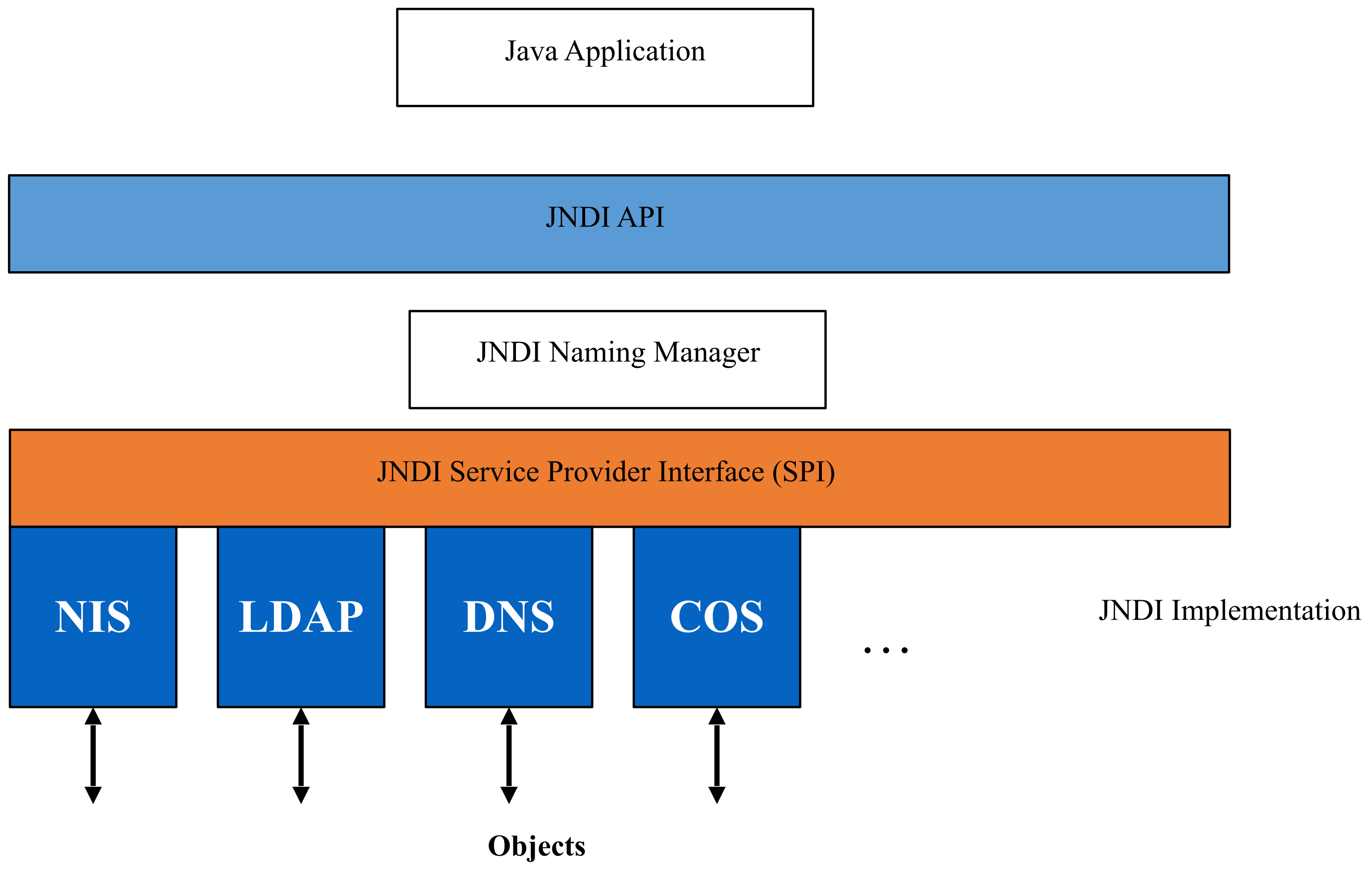
- A directory service supports search of directory objects by names and by queries based on a logical expression of the object's attributes (search filter), e.g.
  - Search a book by name "Java How to Program"
  - Search for books written by "David Flanagan" and "Jim Fareley"



Java Naming and Directory Interface (JNDI)

- A naming and directory service in the Java environment
- Does not replace existing naming/directory services, only provides a common interface (JNDI API):
    - Common Object Services (COS) naming: is used to store and access references to CORBA objects
    - Domain Name System (DNS): Internet naming service to map host names (e.g. www.csis.hku.hk) into IP addresses (147.8.179.15)
    - Lightweight Directory Access Protocol (LDAP): simpler version of X.500 network directory protocol
    - Novell Directory Service (NDS): directory service for Novell Netware networks
    - Network Information System (NIS): a naming system from Sun Microsystems used to access files from a number of hosts with a single ID and password

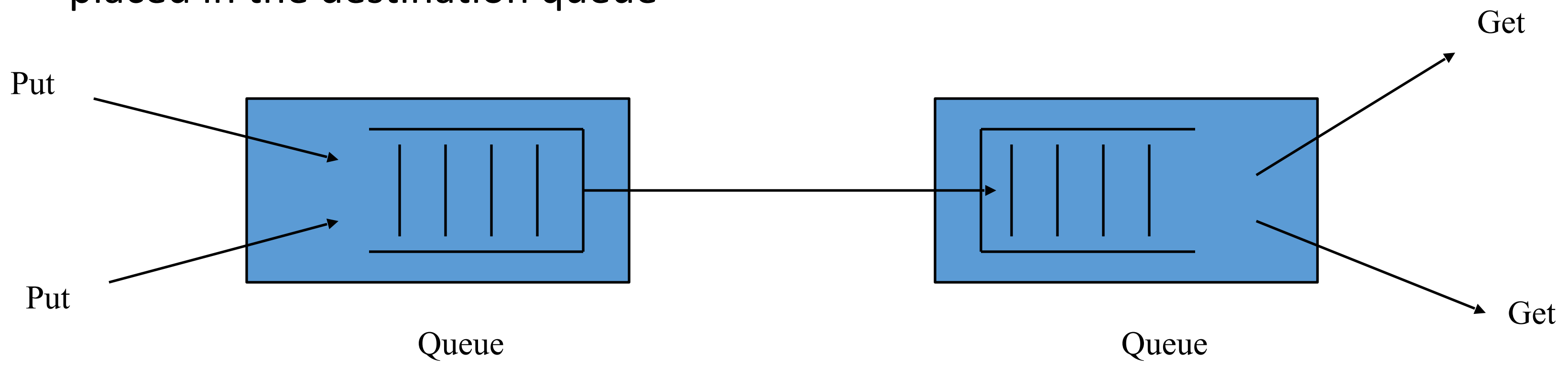
JNDI Architecture



Java Messaging Service (JMS)

Message Queuing

- Program-to-queue communication
- Message queue: like a very fast mail box, i.e. put a message in a box without the recipient being active
- Actions:
  - Put: puts a message into the queue
  - Get: takes a message out of the queue
- Message queue software:
  - Transfer of messages from queue to queue
  - Ensures message arrives eventually and only one copy of the message is placed in the destination queue



Message Queue Software

- Queue have names
- The queues are independent of program, i.e. many programs can perform Puts and Gets on the same queue, and a program can access many queues
- If the network goes down, messages will wait in the queue until the network comes up
- The queues can be put onto a disk so that the queue is not lost even the system goes down
- The queue can be a resource manager and co-operate with a transaction manager, i.e. if the message is put in a queue during a transaction and the transaction later aborted, then the DB will be rolled back and the message is removed from the queue
- Efficient: used in applications require sub-second response time

Message Oriented Middleware (MOM)

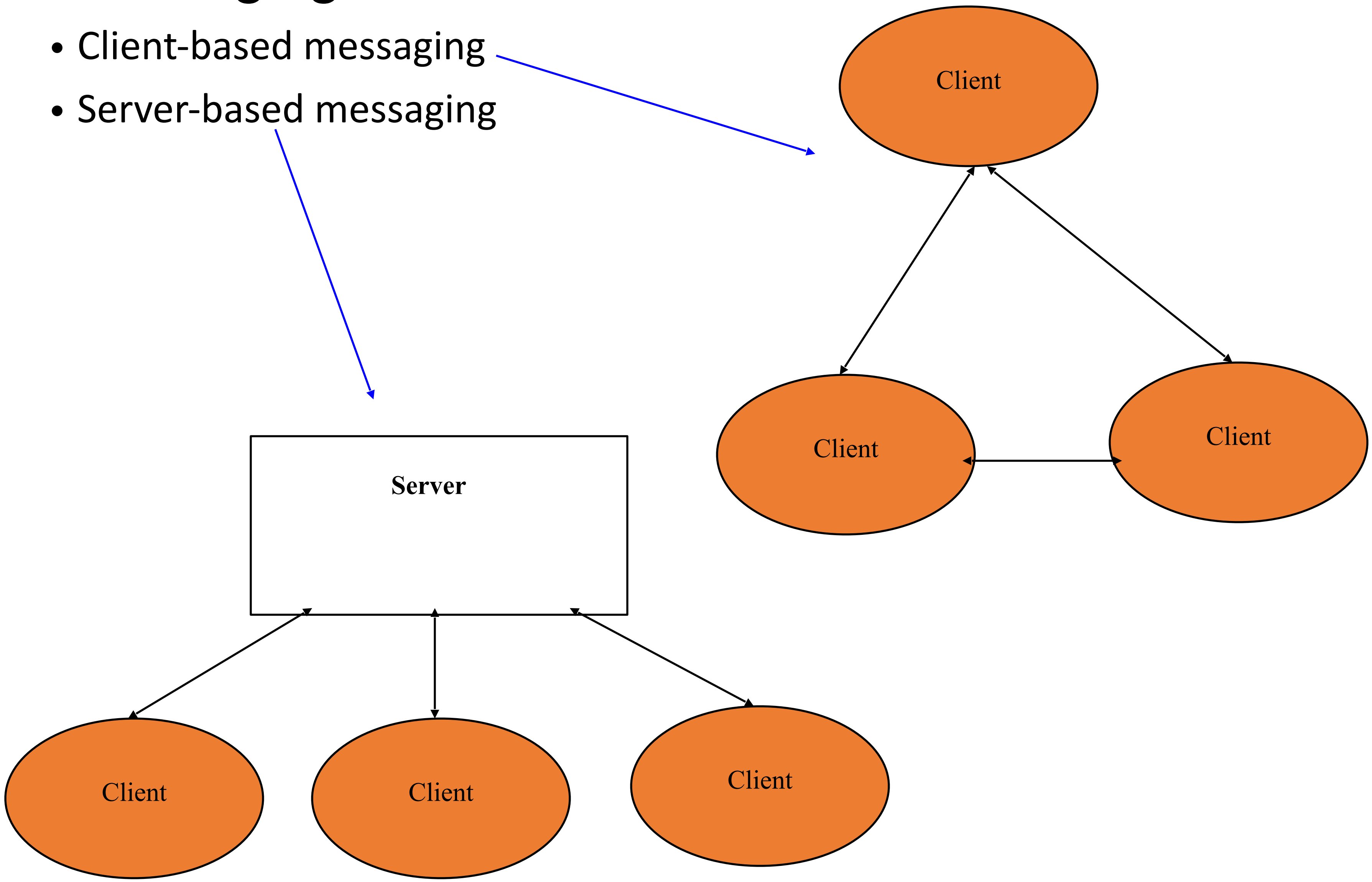
- A mechanism for programs to communicate with each other using messages
- A message is the content of a single communication between 2 or more programs
- Messaging support is provided by message-oriented-middleware (MOM)
- MOM:
  - Defines what a message looks like to a program
  - How a program sends a message
  - How a program receives a message
- Messaging service decouples the sender and receiver
- Java Messaging Service (JMS) is a standard Java-based interface to the messaging services of a MOM

Message Queue Software

- Products: IBM MQSeries, Microsoft MSMQ, BEA Systems Tuxedo/Q, JMS in J2EE

Messaging Architecture

- Client-based messaging
- Server-based messaging



Behaviors

- What happens to a message if the intended receiver is not available?
  - Store for later delivery vs. ignore the message
- What happens when the server goes down? How does the client find out?
- Do all clients receive all messages?
  - Clients want to receive messages by specifying the corresponding *message destination*
- Can any client send messages to any other client?
  - Message provider limits the message destinations
- What can a message include?
  - Binary messages, text messages, messages that have key-value pairs, objects

Messaging Domains

- The models for the programs that are communicating
- Point-to-point messaging: one client to send messages to another client and vice versa, 2 models:
  - Direct messaging: a client directly send a message to another client, similar to RMI
  - Message queue: senders put messages into a queue (supported by JMS)
- Publish-subscribe messaging: multiple programs to receive the same message
  - JMS defines publish-subscribe around the concept of a topic
  - Publishers send messages to a topic, subscribers receive all messages sent to that topic
  - Used if a group of programs want to notify each other of a particular event
- Request-reply messaging: a program sends a message and expects to receive message in return
  - JMS supports the use of a “reply to” field in the above 2 domains

JMS Overview

- Designed by a group of MOM providers and Sun, available in J2EE
- The Java Message Service is a Java API that allows applications to create, send, receive, and read messages
- The JMS API minimizes the set of concepts a programmer must learn to use messaging products but provides enough features to support sophisticated messaging applications

JMS Overview (cont)

- The JMS API enables communication that is not only loosely coupled but also:
  - **Asynchronous.** A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
  - **Reliable.** The JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or to receive duplicate messages.

JMS Overview (cont)

- Messages can be consumed in either of two ways:
  - **Synchronously.** A subscriber or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
  - **Asynchronously.** A client can register a *message listener* with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's onMessage() method, which acts on the contents of the message.