CS550 "Advanced Operating Systems"

Instructor: **Professor Xian-He Sun**

- Email: sun@iit.edu
- Office: SB235C
- Class time: Monday, Wed., 3:15pm-4:30pm, **HH MEZZANINE**
- Office hour: Monday, Wednesday, 4:45-5:45pm
- http://www.cs.iit.edu/~sun/cs550.html

- TA: Mr. Hua Xu, Email: hxu40@hawk.iit.edu
- Office Hour: 11am - 12pm, Tuesday
- meet.google.com/kfp-pysg-cat
- Office Hour: 12pm - 1pm, Tuesday & Thursday

  meet.google.com/bnn-eqao-htg

- Blackboard:
  - http://blackboard.iit.edu
- Substitute lecturer:
  - Anthony Kougkas, assistant research professor
  - akougkas@hawk.iit.edu

Chapter 3: Processes

- Overview of processes and threads
- Virtualization
- Clients
- Servers
- Code/Process migration

Client & Server

Outline

- Networked user interfaces
- Thin-client network computing
- Client-side software for distribution transparency
- Concurrent versus iterative servers
- End-points
- Interrupting a server
- Stateless vs stateful servers
- Server clusters

Concurrent vs Iterative servers

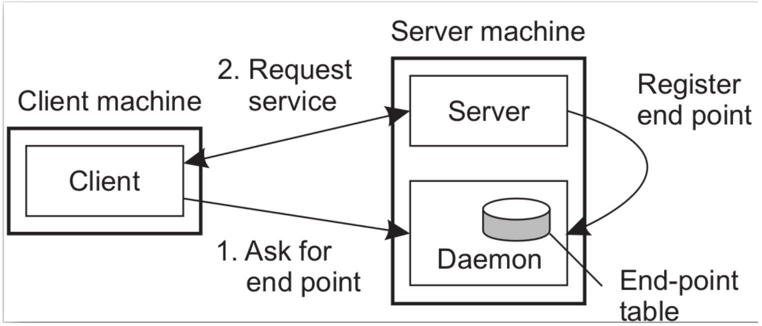
- **Iterative server:**
  - the server itself handles the request and, if necessary, returns a response to the requesting client
- **Concurrent server:**
  - does not handle the request itself but passes it to a separate thread or another process, after which it immediately waits for the next incoming request.
  - Multithreaded: fork a new process for each new incoming request

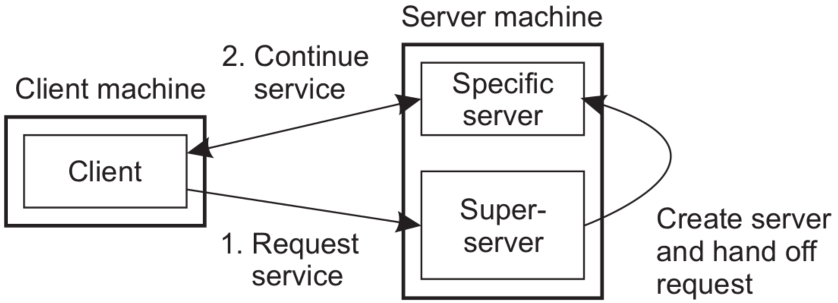
Contacting a server

- Clients send requests to an **end point**, also called a **port**, at the machine where the server is running.
- Each server listens to a specific end point.
- How do clients know the end point of a service?
  - Globally assign end points for well-known services
    - FTP port 21, HTTP port 80 etc
  - client needs to find only the network address of the machine where the server is running → Name services

Contacting a server

- Dynamically assigned to a service by its local OS
  - special daemon running on each machine that runs servers
  - daemon keeps track of the current end point of each service implemented by a co-located server
  - The daemon itself listens to a well-known end point.
  - Use **superservers** for better resource utilization





Interrupting a server

- Whether and how a server can be interrupted?
  - abruptly exit the client application (which will automatically break the connection to the server),
  - immediately restart it and pretend nothing happened.
  - The server will eventually tear down the old connection, thinking the client has probably crashed.
- Better solution:
  - **Out-of-band:** data that is to be processed by the server before any other data from that client.
    - server listens to a separate control end point to which the client sends out-of-band data, while at the same time listening (with a lower priority) to the end point through which the normal data passes.
    - send out-of-band data across the same connection through which the client is sending the original request → Urgent data

Stateless vs Stateful servers

- A **stateless server** does not keep information on the state of its clients, and can change its own state without having to inform any client.
  - e.g. Web server:
    - responds to incoming HTTP requests (store or fetch some data)
    - When the request has been processed, the Web server forgets the client completely.
    - The collection of files that a Web server manages, can be changed without clients having to be informed.

Stateless vs Stateful servers

- A **stateless server** might have information about clients but no disruption of service if this info is lost.
  - For example, a Web server generally logs all client requests.
  - This information is useful, for example, to decide whether certain documents should be replicated, and where they should be replicated to.
  - Clearly, there is no penalty other than perhaps in the form of suboptimal performance if the log is lost.

Stateless vs Stateful servers

- A particular form of a stateless design is known as **soft state**.
- In this case, the server promises to maintain state on behalf of the client, but only for a limited time.
- After that time has expired, the server falls back to default behavior, thereby discarding any information it kept on account of the associated client.
- An example of this type of state is:
  - a server promising to keep a client informed about updates, but only for a limited time.
  - After that, the client is required to poll the server for updates.

Stateless vs Stateful servers

- A **stateful server** generally maintains persistent information on its clients. This means that the information needs to be explicitly deleted by the server.
- e.g., file server:
  - allows a client to keep a local copy of a file
  - maintains a table containing (client, file) entries
  - keeps track of which client currently has the update permissions on which file
  - can improve the performance of read and write operations
  - Drawback: If the server crashes, it has to recover its table of (client, file) entries, or otherwise it cannot guarantee that it has processed the most recent updates on a file.

Stateless vs Stateful servers

- Session state (temporary):
  - It is associated with a series of operations by a single user and should be maintained for some time, but not indefinitely.
  - It is often maintained in three-tiered client-server architectures, where the application server needs to access a database server through a series of queries before being able to respond to the requesting client.
  - No real harm is done if session state is lost, provided that the client can simply re-issue the original request.
  - Allows for simpler and less reliable storage of state.

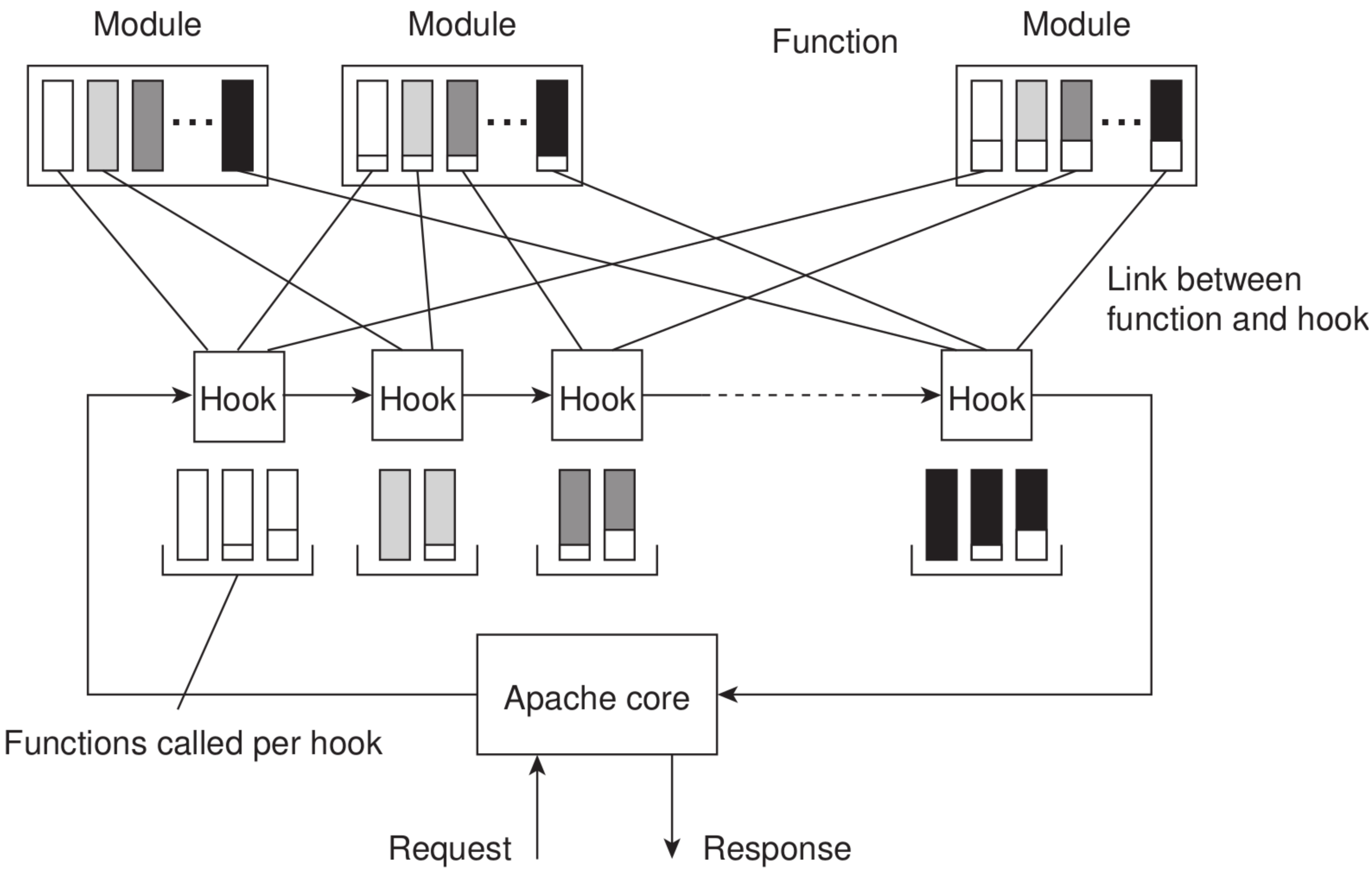
Stateless vs Stateful servers

- Permanent state:
  - Information maintained in databases e.g., customer information, keys associated with purchased software, etc.
  - Maintaining session state already implies a stateful design
  - Requires special measures when failures do happen
  - Requires explicit assumptions about the durability of state stored at the server.

Example: The Apache Web server

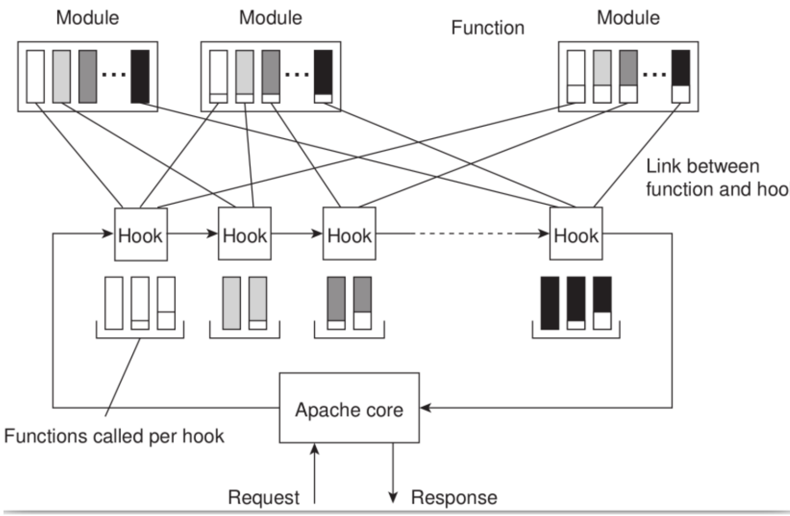
- Extremely popular server, estimated to be used to host approximately 50% of all Web sites.
- Highly configurable and extensible, and at the same time largely independent of specific platforms.
- Provides its own basic runtime environment, which is then subsequently implemented for different operating systems.
- Apache Portable Runtime (APR):
  - a library that provides a platform-independent interface for file handling, networking, locking, threads, and so on.
  - portability is largely guaranteed provided that only calls to the APR are made and that calls to platform-specific libraries are avoided.

Example: The Apache Web server



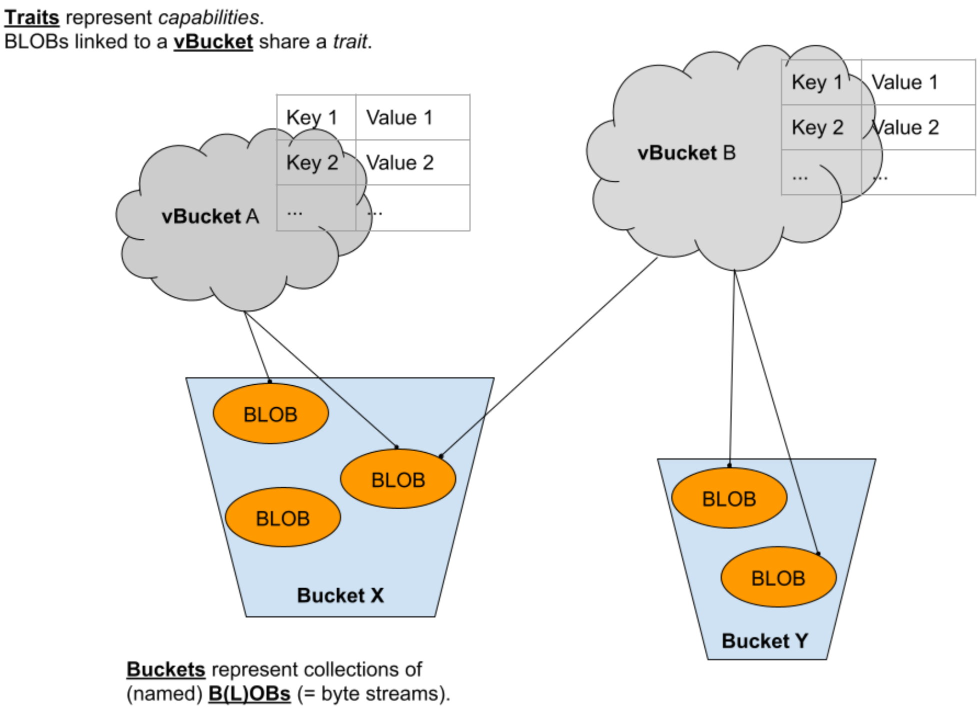
Example: The Apache Web server

- The functions associated with a hook are all provided by separate **modules**.
- Every hook can contain a set of functions that each should match a specific function prototype (i.e., list of parameters and return type).
- A module developer will write functions for specific hooks.
- When compiling, the developer specifies which function should be added to which hook.



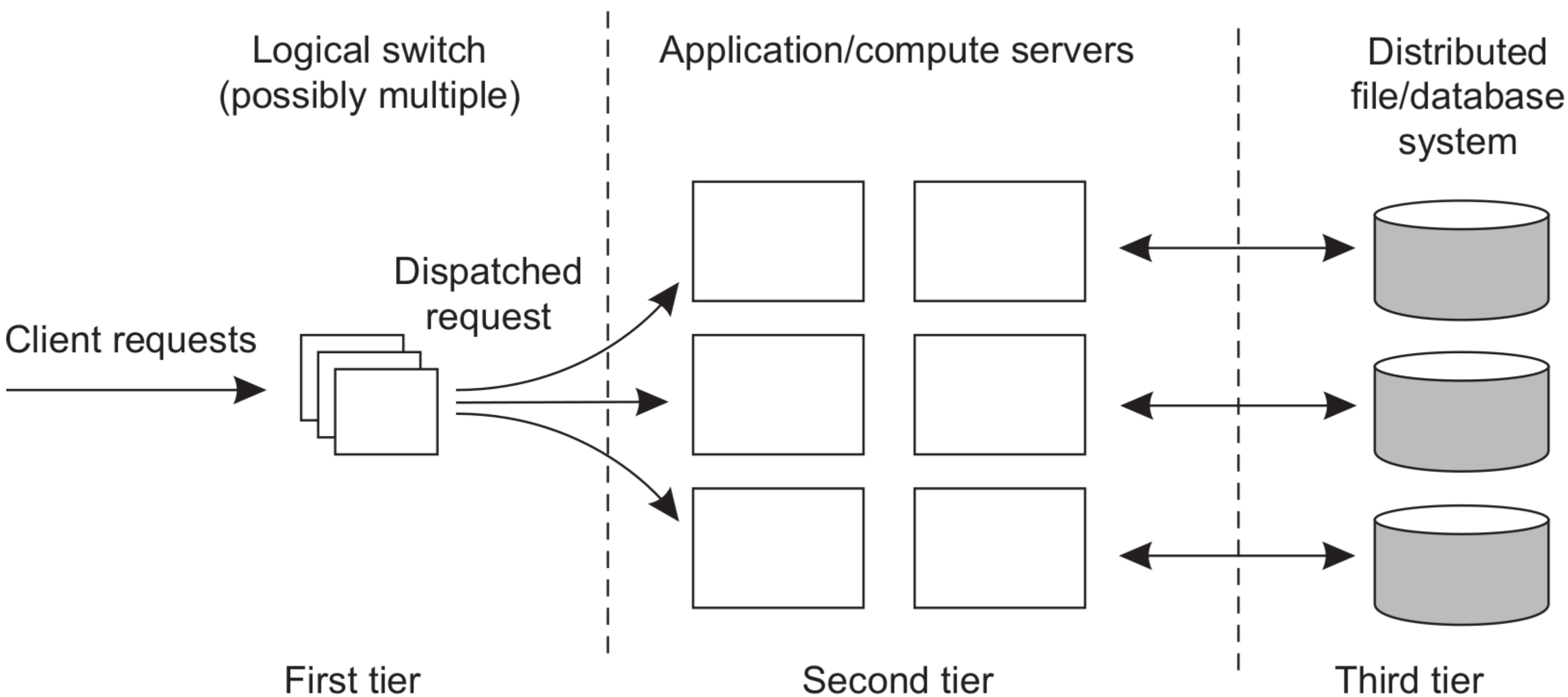
Hermes Data Model

- Blobs
  - Unit of data as key-value pairs
  - Value as uninterpreted byte arrays
  - Stored internally as a collection of buffers across multiple tiers
- Bucket
  - Collection of blobs
  - Flat blob organization
- Virtual Bucket
  - Linked blobs across buckets
  - Attached capabilities
- Traits
  - Ordering, grouping, filtering
  - Compression, deduplication, etc



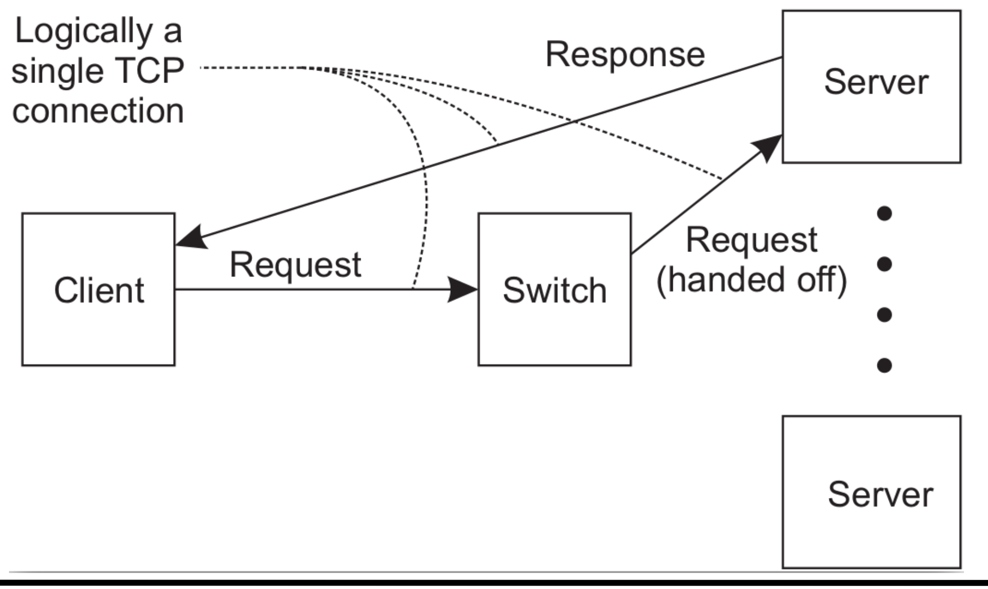
Server clusters

- A server cluster is nothing else but a collection of machines connected through a network, where each machine runs one or more servers.



Request dispatching

- Design goal for server clusters:
  - Hide the fact that there are multiple servers
  - A single access point, implemented through a hardware switch such as a dedicated machine.
  - For scalability and availability, a server cluster may have multiple access points

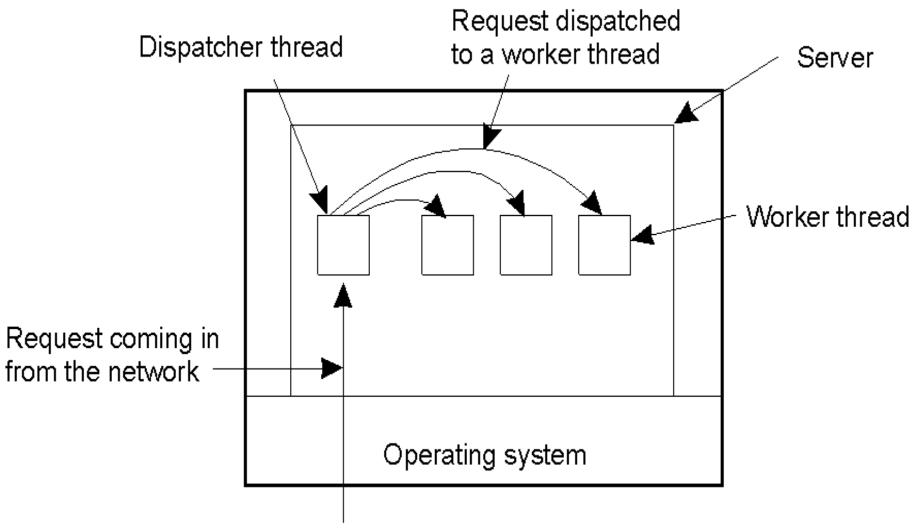


Multi-threaded Clients Example

- Web Browsers such as IE are multi-threaded
- Such browsers can display data before entire document is downloaded: performs multiple simultaneous tasks
  - Fetch main HTML page, activate separate threads for other parts
  - Each thread sets up a separate connection with the server
    - Uses blocking calls
  - Each part (gif image) fetched separately and in parallel
  - Advantage: connections can be setup to different sources
    - Ad server, image server, web server…

Multi-threaded Server Example

- Apache web server: pool of pre-spawned worker threads
  - Dispatcher thread waits for requests
  - For each request, choose an idle worker thread
  - Worker thread uses blocking system calls to service web request



Summary

- Overview of clients and servers (today)
  - Networked user interfaces
  - Thin-client network computing
  - Client-side software for distribution transparency
  - Concurrent versus iterative servers
  - End-points
  - Interrupting a server
  - Stateless vs stateful servers
  - Server clusters

- Readings:
  – Chpt 3.3&3.4 of textbook

Outline

- Code and process migration
  - Motivation
  - How does migration occur?
  - Resource migration
- Load balancing

Motivation

- Key reasons: performance and flexibility
- Process migration (aka *strong mobility*)
  - Improved system-wide performance – better utilization of system-wide resources
- Code migration (aka *weak mobility)*
  - Shipment of server code to client
  - Ship parts of client application to server
  - Improve parallelism

Motivation

- Performance
  - From heavily-loaded to lightly-loaded machines
  - Exploit parallelism, e.g., searching for information in the web through the development of mobile agent, that moves from site to site
  - fault tolerance, e.g., moving from failure-prone to failure-free machines

Code migration for performance

- Consider a client-server system in which:
  - the server manages a huge database
  - client application needs to perform many database operations involving large quantities of data
  - **better to ship** part of the client application to the server and send only the results across the network
  - Avoid network congestion leading to better performance

Code migration for performance

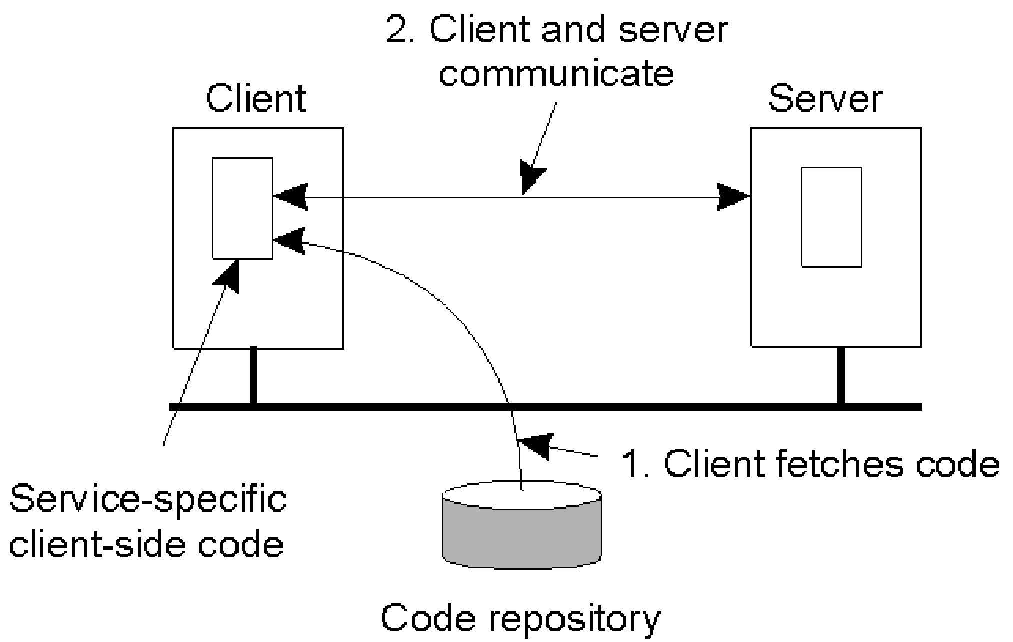
- Consider an interactive database application:
  - clients need to fill in forms that are subsequently translated into a series of database operations
  - Process the form at the client side, and send only the completed form to the server
  - avoid a relatively large number of small messages needing to cross the network
  - client perceives better performance, while at the same time the server spends less time on form processing and communication

Code migration for performance

- Consider searching the Web:
  - implement a search query in the form of a small mobile program, called **mobile agent**.
  - Agent moves from site to site
  - Linear speedup if we make several copies of such a program, and send each off to different sites
  - Concern: security!

Motivation

- Flexibility
  - Dynamic configuration of distributed system
  - Clients don’t need preinstalled software – download on demand



Code migration for flexibility

- Suppose a server implements a standardized interface to a file system:
  - server makes use of a proprietary protocol to allow remote clients to access the file system
  - the client-side implementation of the file system interface would need to be linked with the client application
  - Problem: this approach requires that the software be readily available to the client at the time the client application is being developed!

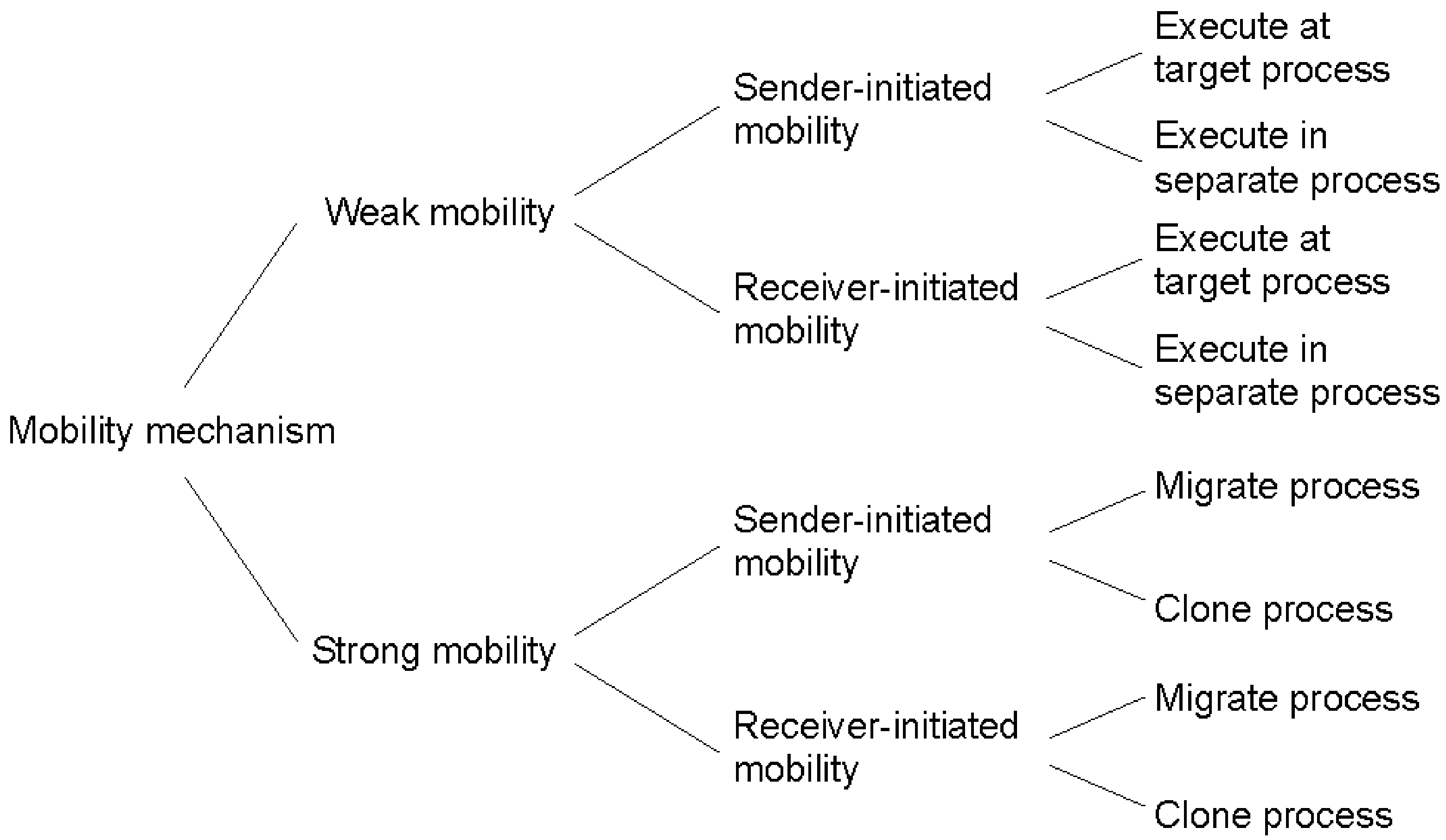
Code migration for flexibility

- Alternative:
  - let the server provide the client's implementation when the client binds to the server
  - client dynamically downloads the implementation, goes through the necessary initialization steps, and subsequently invokes the server
  - protocol for downloading and initializing code is standardized
  - necessary that the downloaded code can be executed on the client's machine
  - Benefit: clients need not have all the software preinstalled to talk to servers

Migration models

- Process = code segment + resource segment + execution segment
- Weak versus strong (runtime info, process) mobility
- Sender-initiated versus receiver-initiated
  - Sender-initiated (code is with sender)
    - Client sending a query to database server
    - Client should be pre-registered
  - Receiver-initiated
    - Java applets
    - Receiver can be anonymous

Models for Code Migration



Do Resources Migrate?

- Depends on process-to-resource bindings
  - By identifier: specific web site, ftp server
  - By value: Java libraries
  - By type: printers, local devices
- Depends on resource-to-machine bindings
  - Unattached to any node: data files
  - Fastened resources: database, web sites
  - Fixed resources: local devices, communication end points

Migration in Heterogeneous Systems

- Systems can be heterogeneous
  - Portability?
- Code migration in heterogeneous systems is being tackled by scripting languages and highly portable languages such as Java
- Rely on a (process) virtual machine that either
  - directly interprets source code (as in the case of scripting languages), or
  - interprets intermediate code generated by a compiler (as in Java).

Migration in Heterogeneous Systems

- Other solutions:
  - migrate not only processes, but entire computing environments
  - compartmentalize the overall environment and provide processes in the same part their own view on their computing environment
  - Typical example: virtual machines
    - Running an operating system and a suite of applications

Migration in Heterogeneous Systems

- Migrate VMs:
  - Major advantage is that processes can remain ignorant of the migration itself
  - They need not be interrupted in their execution, nor should they experience any problems with used resources.
  - Resources are either migrating along with a process, or the way that a process accesses a resource is left unaffected.

Migration in Heterogeneous Systems

- Three ways to handle VM migration:

1) Pushing memory pages to the new machine and resending the ones that are later modified during the migration process.
2) Stopping the current virtual machine; migrate memory, and start the new virtual machine.
3) Letting the new virtual machine pull in new pages as needed, that is, let processes start on the new virtual machine immediately and copy memory pages on demand.