Distributed Systems
ECEG-6504

Processes

Surafel Lemma Abebe (Ph. D.)

Topics

- Introduction
- Threads
- Code migration
- Agents in distributed systems

Introduction

- Definition of a **process**
  - A program in execution
  - An asynchronous activity
  - The 'animated sprit' of a procedure in execution
  - The entity to which processors are assigned
  - The 'dispatchable' unit

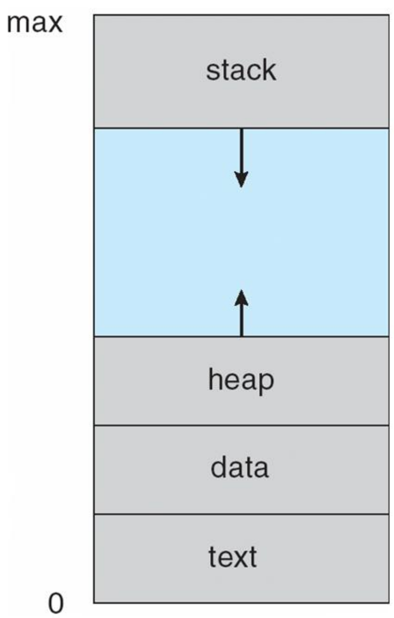
⇒ No universally agreed upon definition

  ⇒ "A program in execution" is mostly used

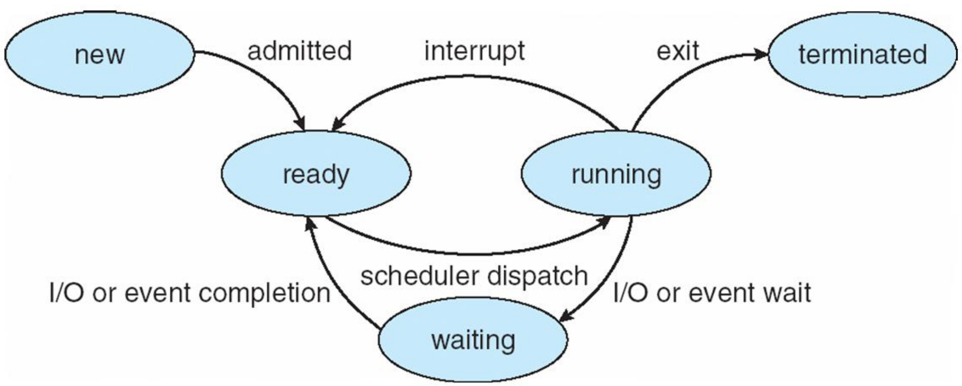
- Are processes and programs the same?
  - No
- What is the **difference** between process and program?
  - Process is an "active" entity, while a program is a "passive" entity
  - Program is only part of a process

Introduction…

- Difference between process and program…
  - Process includes
    - Program code, also called **text section**
    - Current value of **program counter**
    - Contents of **processor registers**
    - **Stack** containing temporary data
      - Function parameters, return addresses, local variables
    - **Data section** containing global variables
    - **Heap** containing memory dynamically allocated during run time
  - One program can be several processes
    - Consider multiple users executing the same program

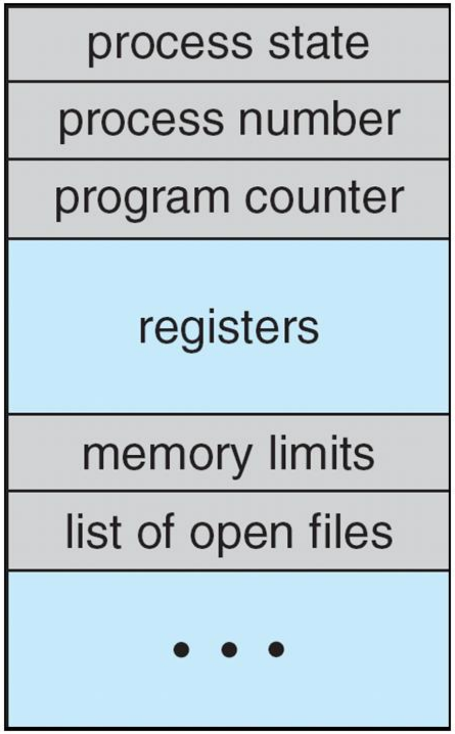


- Process has different states



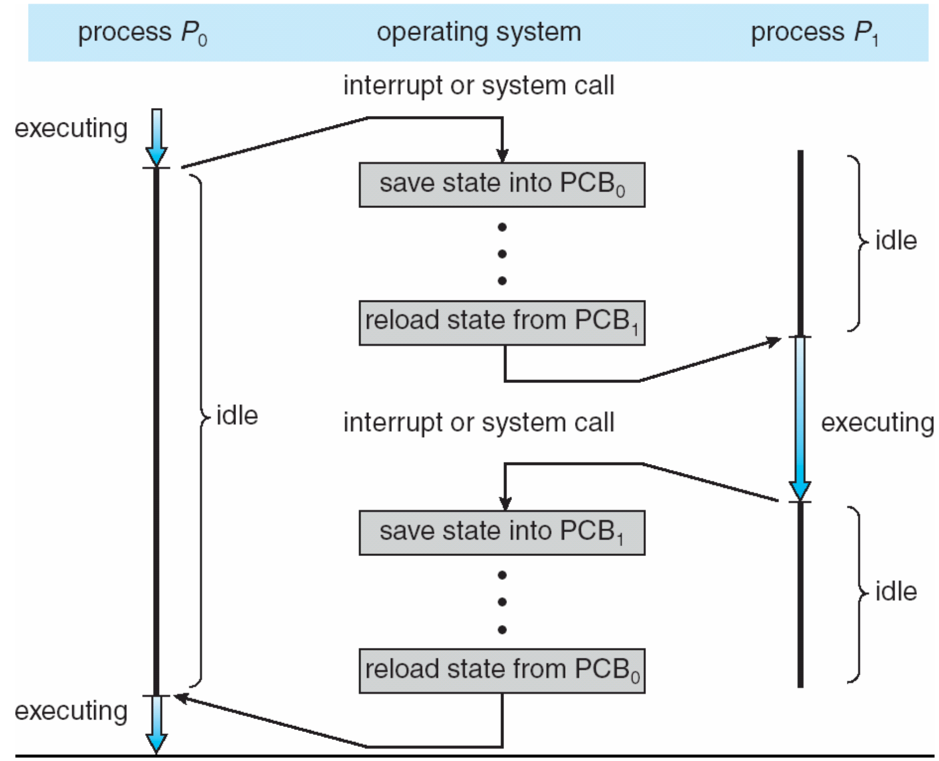
Introduction…

- Process Control Block (PCB)
  - Information associated with each process
  - Also called task control block
    - Process state
      - Running, waiting, etc
    - Program counter
      - Location of instruction to next execute
    - CPU registers
      - Contents of all process-centric registers
    - CPU scheduling information
      - Priorities, scheduling queue pointers
    - Memory-management information
      - Memory allocated to the process
    - Accounting information
      - CPU used, clock time elapsed since start, time limits
    - I/O status information
      - I/O devices allocated to process, list of open files



Introduction…

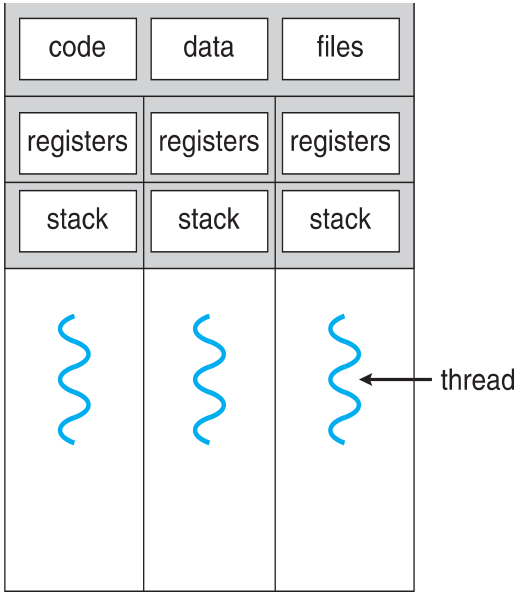
- Context Switch
  - Happens when CPU switches **from one process to another process**
  - **Context** of a process is represented in the PCB
  - Initiated by a **scheduler**
  - Scheduler determines
    - When a running process is to be interrupted
    - Which process from the ready queue will run next



Threads

- What are threads?
  - Thread is a single sequence of stream within a process
  - Basic unit of CPU utilization

  - Has some properties of processes
  - Allow multiple executions of streams in a process
  - Comprises a threadID, a program counter, a register set, and a stack
  - Are threads independent of one another like processes?
    - No. Threads share their code section, data section, OS resources (e.g., opened files)



multithreaded process

Threads…

- Threads vs Processes
  - Similarities
    - Like processes threads share CPU and only one thread is active (running) at a time (for one processor)
    - Like processes, threads within a process execute sequentially
    - Like processes, thread can create children
    - And like process, if one thread is blocked, another thread can run

  - Differences
    - Unlike processes, threads are not independent of one another
    - Unlike processes, all threads can access every address in the task
    - Unlike processes, thread are design to assist one another
      - Note that processes might or might not assist one another because processes may originate from different users
    - Unlike processes, threads do not try to achieve higher degree of concurrency transparency

Threads...

- Context switching
  - Threads share the same address space
    - Thread context switching can be done entirely independent of the operating system
  - Process switching is generally more expensive as it involves getting the OS in the loop, i.e., trapping to the kernel
  - Creating and destroying threads is much cheaper than doing so for processes

- Processes are building blocks of DS
- DS requires to have more fine grained control at the level of threads
  - Helps to achieve higher performance

Threads…

- Main issue in thread implementation - OS
  - Should an OS kernel provide threads, or should they be implemented as user-level packages?
- User-level solution
  - All operations can be completely handled within a single process

    ⇒ implementations can be extremely efficient
  - All services provided by the kernel are done on behalf of the process in which a thread resides

    ⇒ if the kernel decides to block a thread, the entire process will be blocked

    ⇒ if the kernel can't distinguish threads, how can it support signaling events to them?

Threads...

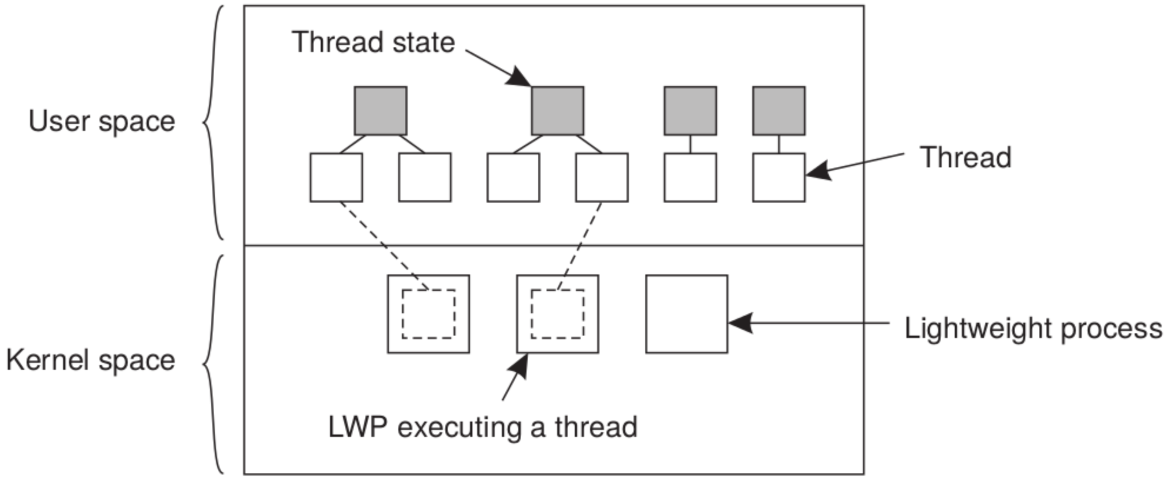
- Kernel solution
  - The whole idea is to have the kernel contain the implementation of a thread package

    => This means that all operations return as system calls

    - Operations that block a thread are no longer a problem: the kernel schedules another available thread within the same process
    - Handling external events is simple: the kernel (which catches all events) schedules the thread associated with the event
    - Problem
      - Loss of efficiency due to the fact that each thread operation requires a trap to the kernel
- Solution
  - Try to mix user-level and kernel-level threads into a single concept: Light weight processes

Threads...

- Lightweight processes (LWP)
  - Introduce a two-level threading approach: lightweight processes that can execute user-level threads
    - LWP runs in the context of a single (heavy-weight) process
    - Thread package is implemented in user space
    - Thread package can be shared by multiple LWP



Threads…

- Lightweight processes (LWP)
  - Principal operation
    - User-level thread does system call

      ⇒ The LWP that is executing that thread, blocks

      The thread remains bound to the LWP
      - The kernel can schedule another LWP having a runnable thread bound to it
    - A thread calls a blocking user-level operation

      ⇒ Do context switch to a runnable thread, (then bound to the same LWP)
      - When there are no threads to schedule, an LWP may remain idle, and may even be removed (destroyed) by the kernel
  - Note
    - This concept has been virtually abandoned – it's just either user-level or kernel-level threads.

Threads...

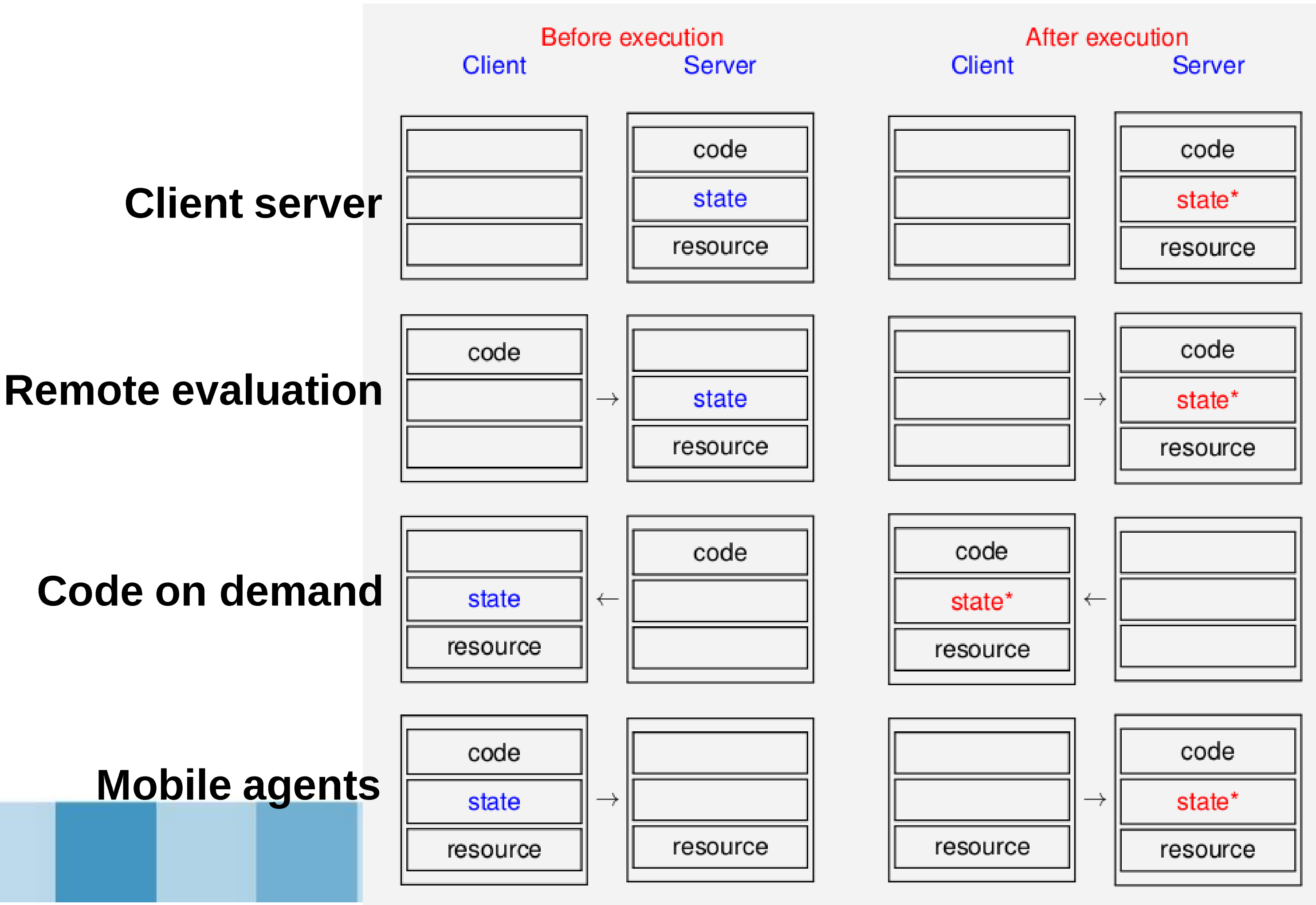
- Threads in DS
  - Property of threads (not blocking the entire process) makes them attractive for DS
  - Multithreaded Web client
    - Hiding network latencies
      - Web browser scans an incoming HTML page, and finds that more files need to be fetched
      - Each file is fetched by a separate thread, each doing a (blocking) HTTP request
      - As files come in, the browser displays them
  - Multiple request-response calls to other machines (RPC)
    - A client does several calls at the same time, each one by a different thread
    - It then waits until all results have been returned
    - Note
      - If calls are to different servers, we may have a linear speed-up

Threads...

- Threads in DS...
  - Improve performance
    - Starting a thread is much cheaper than starting a new process
    - Having a single-threaded server prohibits simple scale-up to a multiprocessor system
    - As with clients
      - Hide network latency by reacting to next request while previous one is being replied
    - At servers
      - Help attain high performance by exploiting parallelism
  - Better structure
    - Most servers have high I/O demands
      - Using simple, well-understood blocking calls simplifies the overall structure
    - Multithreaded programs tend to be smaller and easier to understand due to simplified flow of control

Code migration

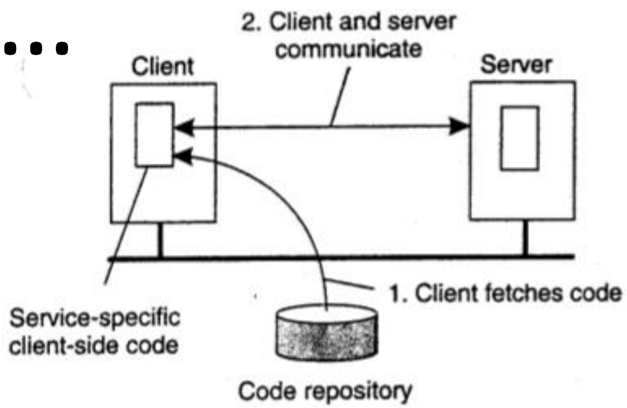
- Communication is not limited to only passing data
  - In some situations, code could also be migrated
  - Example: Implementation of a service in the context of code migration



Code migration…

- Reasons for code migration
  - Performance
    - Move processes from a heavily-loaded to lightly-loaded machines
      - Load is expressed in terms of
        - » CPU queue length
        - » CPU utilization
    - Based on qualitative reasoning
      - Assumption
        - » It generally makes sense to process data close to where the data resides
      - Supports parallelism

Code migration…



- Reasons for code migration…
  - Flexibility
    - Traditional approach
      - Partition the application into different parts and, decide in advance where each part should be executed
    - Provide implementation no sooner than is strictly necessary
      - E.g., when the client binds to the server
      - Advantage
        - » Client-server protocol could be changed as often as one wants
          - Uses a standard interface
        - » Clients need not have all the software preinstalled
      - Disadvantage
        - » Security

Code migration…

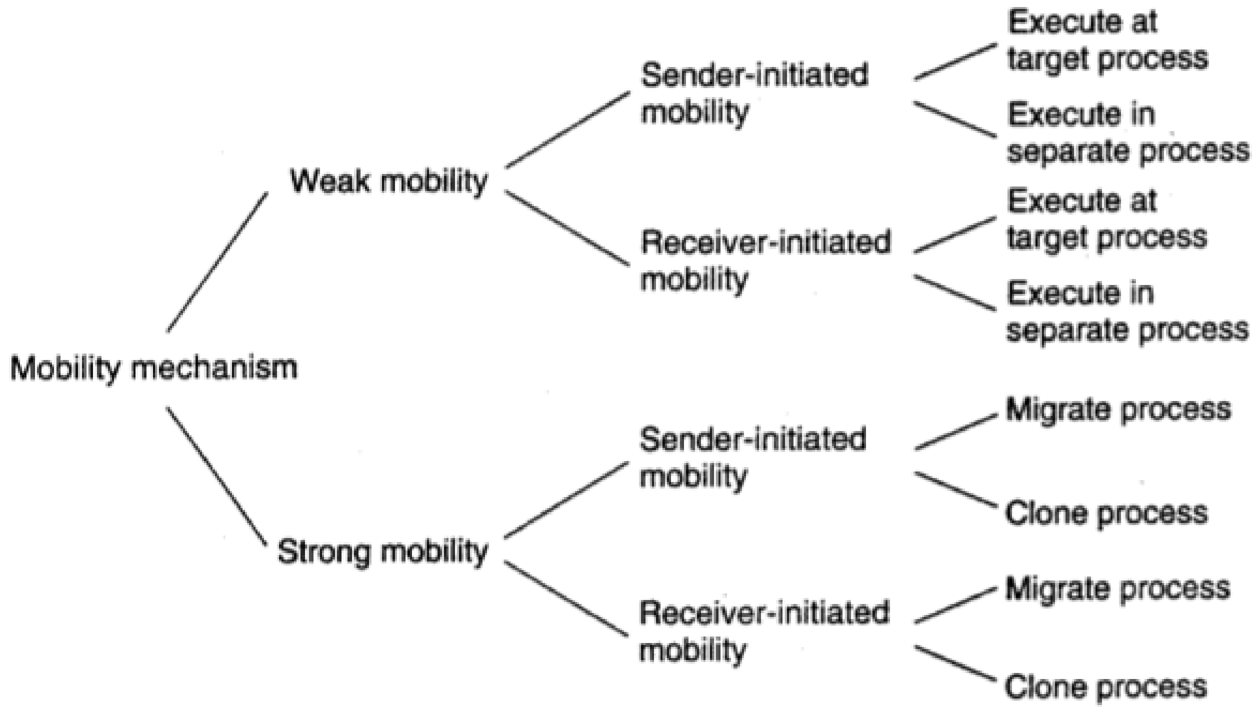
- Models for code migration
  - Process consists of three segments
    - Code segment
      - Consists set of instructions that make up the program being executed
    - Resource segment
      - Contains references to external resources needed by the process
      - E.g., files, printers, devices,…
    - Execution segment
      - Stores the current execution state of a process
      - Contains private data, stack and program counter

Code migration…

- Models for code migration…
  - Weak mobility
    - Move only code and data segment (containing initialization data)s
      - Starts execution form one of several predefined starting positions
        - » E.g., Java applets
      - Relatively simple, especially if code is portable
    - Could be executed by the target process or a separate process
  - Strong mobility
    - Move component, including execution segment
      - Migration: move entire object from one machine to the other
      - Cloning: start a clone, and set it in the same execution state
  - Could further be classified as
    - Code shipping (push/sender-initiated)
    - Code fetching (pull/receiver-initiated)

Code migration…

- Models for code migration…



Code migration…

- Migration and local resources
  - Resource segment cannot always be simply transferred along with other segments
    - E.g., reference to a specific TCP port vs reference to a file using an absolute URL
  - Types of process-to-resource bindings
    - By identifier
      - Process requires a specific instance of a resource
      - E.g., local communication end points
    - By value
      - Process requires the value of a resource
      - E.g., the set of cache entries, libraries
    - By type
      - Process requires that only a type of resource is available
      - E.g., a color monitor

Code migration…

- Migration and local resources…
  - We could change the reference to resources, but could not affect the kind of process-to-resource binding
  - Resource-to-machine binding
    - Fixed
      - Resource cannot be migrated
      - E.g., local hardware, local communication end point
    - Fastened
      - Resource can, in principle, be migrated but only at high cost
      - E.g., local databases, websites
    - Unattached
      - Resource can easily be moved along with the object
      - E.g., a cache, data files associated with only the program

Code migration…

- Migration and local resources…

Resource-to-machine binding

	Unattached	Fastened	Fixed
ID	MV (or GR)	GR (or MV)	GR
Value	CP (or MV, GR)	GR (or CP)	GR
Type	RB (or MV, GR)	RB (or GR, CP)	RB (or GR)

GR = Establish global systemwide reference

MV = Move the resource

CP = Copy the value of the resource

RB = Re-bind to a locally available resource

Code migration…

- Migration in heterogeneous system
  - Problem
    - Target machine may not be suitable to execute the migrated code
    - Definition of process/thread/processor context is highly dependent on local hardware, operating system and runtime system
  - Solution
    - Make use of an abstract machine that is implemented on different platforms
      - Interpreted languages, effectively having their own VM
      - Virtual VM

Agents

- Some definitions
  - *"An agent is anything that can be viewed as perceiving its environment through sensors and acting on that environment through effectors*" (Russell and Norvig 1995)
  - *"[An agent is] a piece of software that performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully. The software should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result*" (Hermans 1996)
  - *"A software entity that performs tasks on behalf of another entity, be it a software, a hardware, or a human entity" (Shehory 2014)*

Agents…

- Dimensions of agenthood
  - Core set that we find central to the definition and development of software agents
    - Autonomy
      - Refers to the ability of an agent to perform unsupervised computation and action, and to pursue its goals without being explicitly instructed for doing so
    - Intelligence
      - Originates from the agent having to act on behalf of another
      - Agents that reason about serving others and act accordingly
      - May required capabilities
        - » Learning, reasoning, planning, and decision making
          - Allow agent to make educated decisions and to behave rationally
          - Allow agent to be goal-oriented

Agents...

- Dimensions of agenthood...
  - Sociality
    - Agent might need to interact with other agents and coordinate, collaborate, or compete to meet its goals
  - Mobility
    - Agents may be able to change their logical or physical location
    - When do we say an agent moved?
      1. When agent moves from its current execution environment to another
      2. When agents reside on mobile devices and the device moves
  - ...