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

Term Project

- See http://www.cs.iit.edu/~sun/html/report2.html
- A **two-page** project proposal due by **Jan. 29, 2024** (format and examples)
- Final project report is due on April 25, 2024

- Example topics
  - Study and practice of some middleware programming-environment, software packages, applications.
  - Study and analyze some distributed environment, architectures, and network structures.
  - Study the distributed solution of certain application package, algorithm, and system software.
  - Performance metric, measurement, and benchmark.
  - Study and practice of some visualization tools.
  - Survey of certain topics.
  - Any other topics that are relevant to this course.

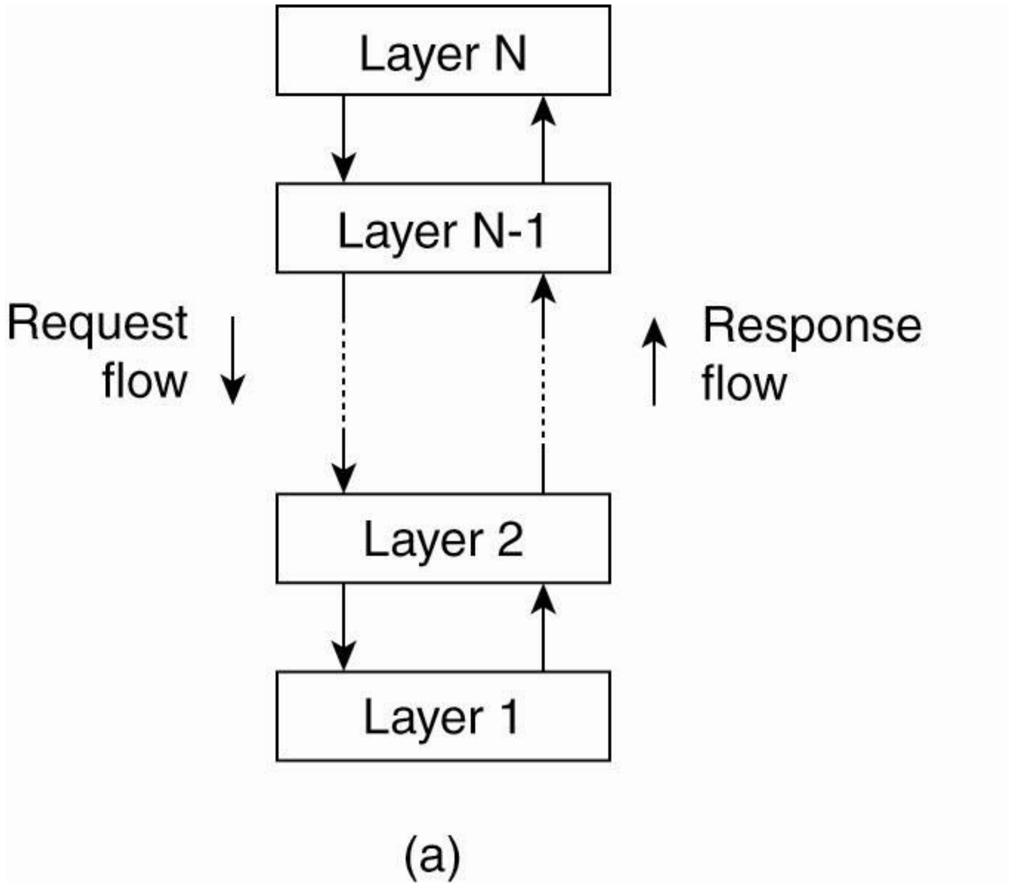
Questions?

•What is the difference between operating system and (software) system?
•What is the difference between Network OS and Distributed OS?
•What is the difference between Distributed OS and Distributed (software) system?
•What is middleware?
•What is the difference between middleware and distributed (software) system?

Architectural Styles

- Software architectures
  - **Logical organization** of distributed systems into **software components**
- Component
  - A modular unit with well-defined interfaces that is replaceable within its environment
- Four important (structure) styles
  - Layered architectures
  - Object-based architectures
  - Resource-based architectures
  - Publish-subscribe architectures
    - Event-based
    - Shared-data based

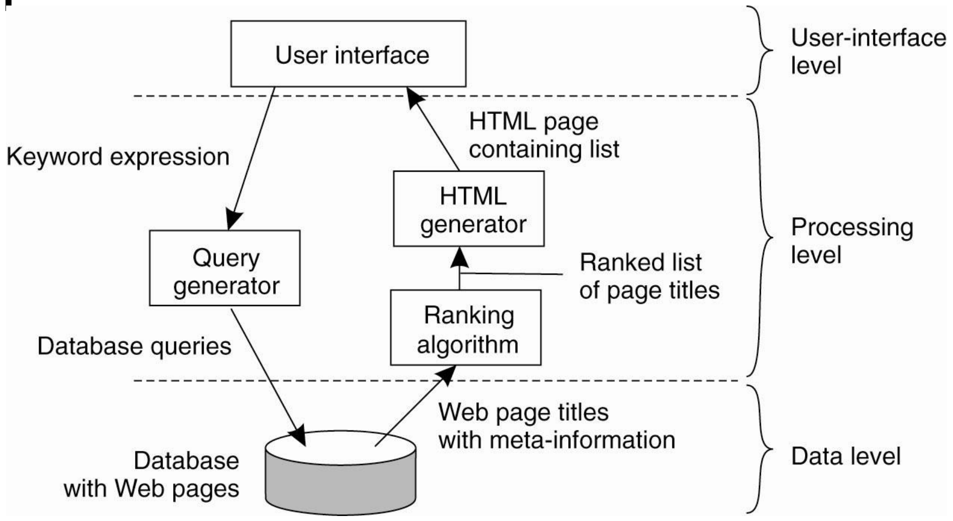
Layered Style



Components are organized in a layered fashion where a component at Layer L_i can call components at the underlying layer L_{i-1}

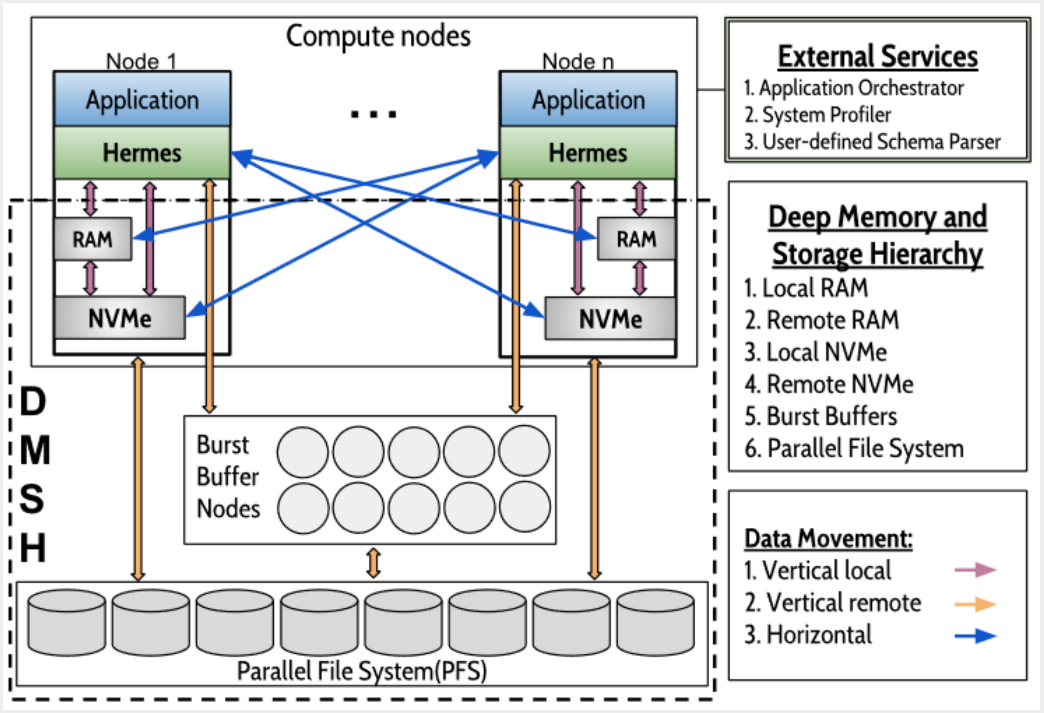
Layered Style Example

- Application Layering
  - There is no clear distinction between a client and a server
- Since many client-server applications are targeted toward supporting user access to DB
  - The user-interface level
  - The processing level
  - The data level



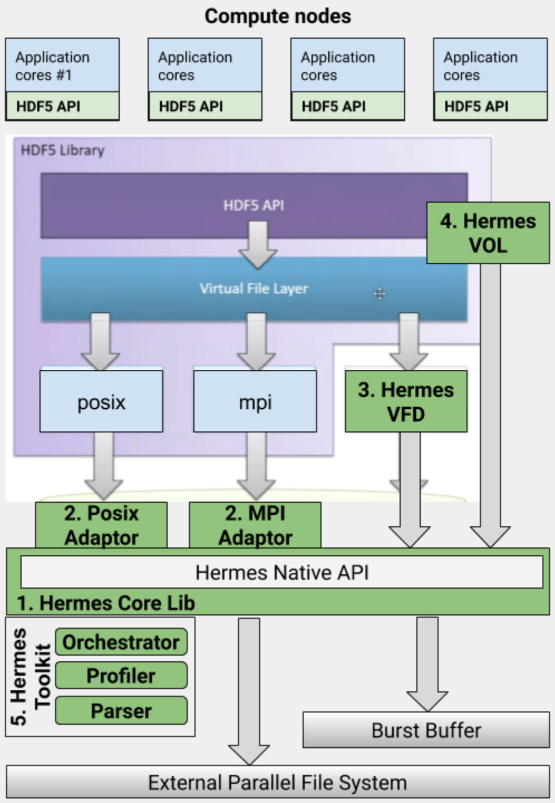
Hermes Architecture

- Hermes machine model:
  - Large amount of RAM
  - Local NVMe and/or SSD device
  - Shared Burst Buffers
  - Remote disk-based PFS
- Hierarchy based on:
  - Access Latency
  - Data Throughput
  - Capacity
- Two data paths:
  - Vertical (within node)
  - Horizontal (across nodes)

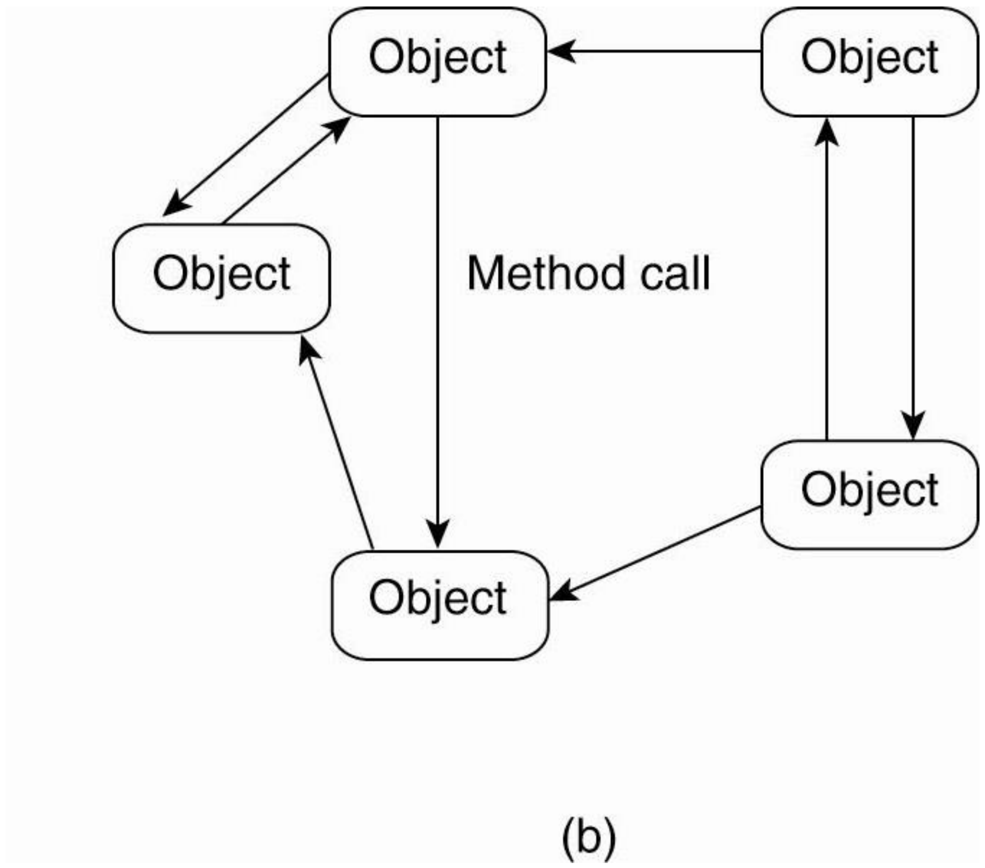


Hermes Ecosystem

1. Hermes core library
   a. Manages tiers transparently
   b. Facilitates data movement in the hierarchy
   c. Provides native buffering API
2. Hermes Adapters
   a. POSIX, MPI-IO, Pub-Sub, etc
      i. Intercept I/O calls to Hermes
      ii. Boosts legacy app support
3. Hermes VFD (virtual file drive)
   a. Directs HDF5 I/O to Hermes native API
4. Hermes VOL (virtual object layer)
   a. Captures application's behavior and provides hints to Hermes core lib
5. Hermes Toolkit

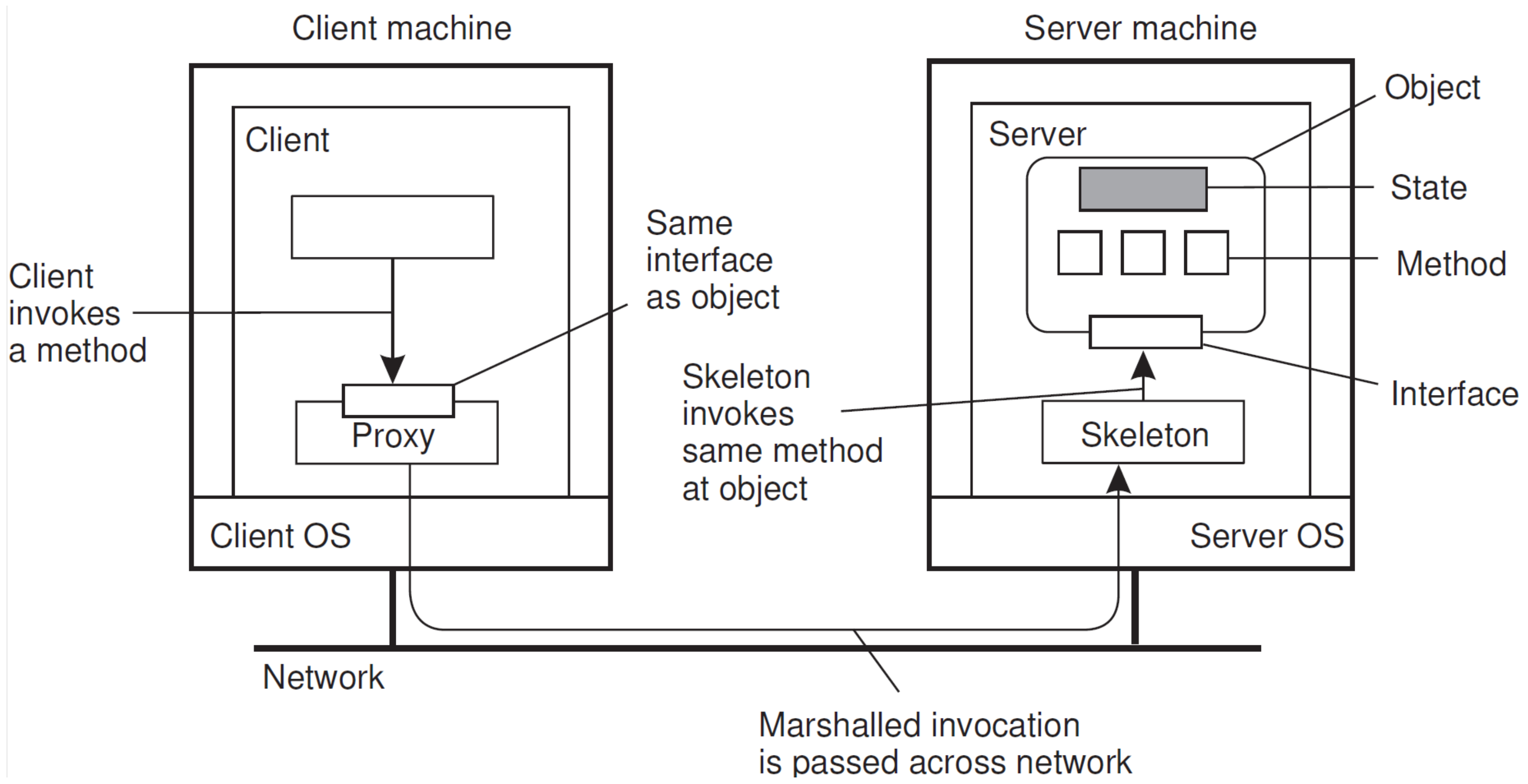


Object-Based Style



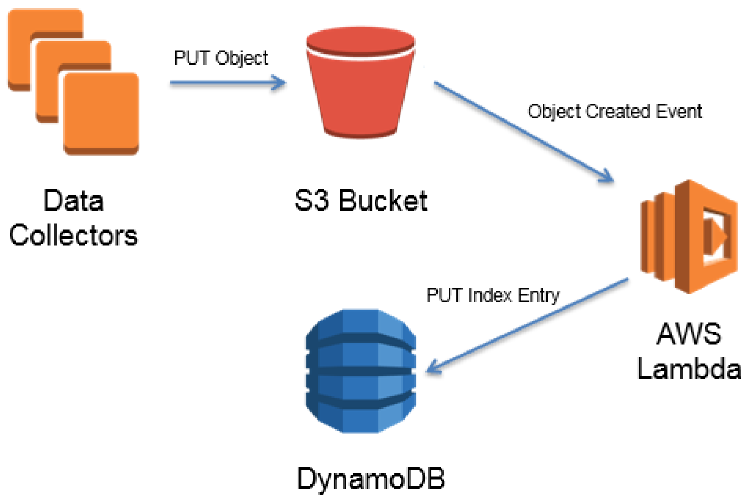
Each **object** corresponds to a **component**, and these components are connected through a **procedure call** mechanism

Object-Based Style Example



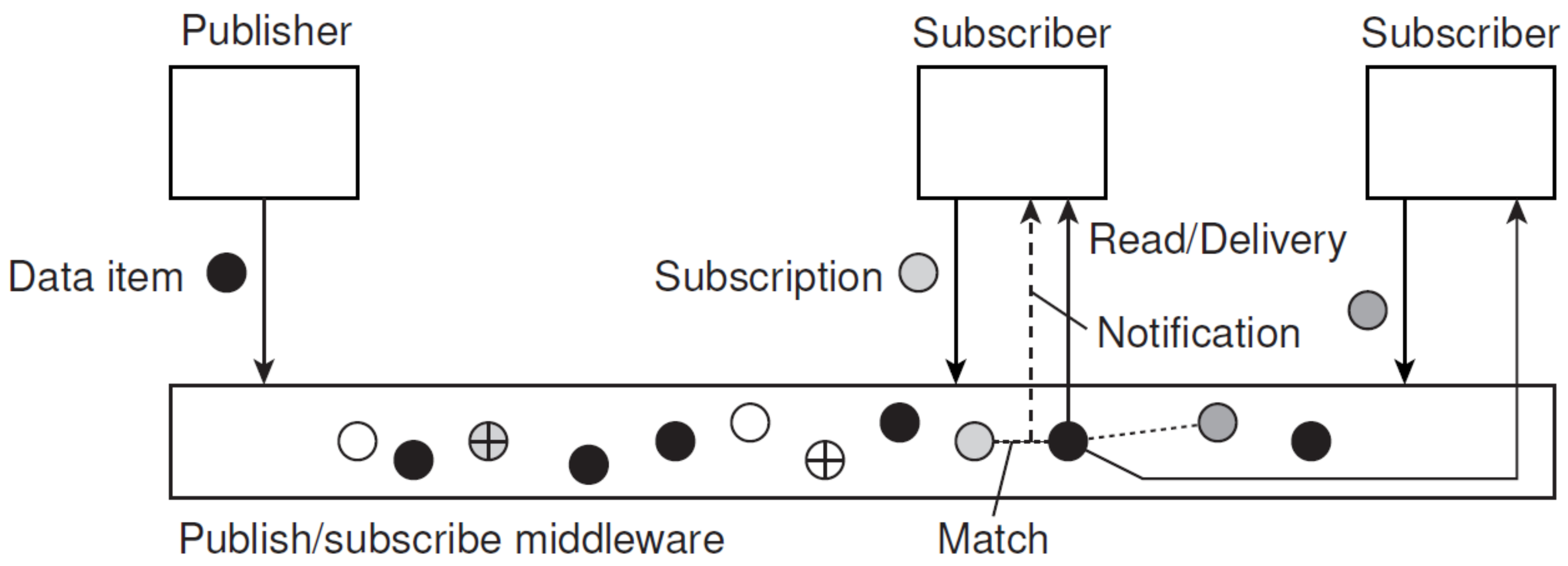
Resource-based Style

- Huge collection of resources that are individually managed by components.
- Resources may be added or removed by (remote) applications, and likewise can be retrieved or modified. This approach has now been widely adopted for the Web and is known as Representational State Transfer (REST).
- There are four key characteristics of what are known as RESTful architectures:
  - Resources are identified through a single naming scheme
  - All services offer the same interface, consisting of at most four operations
  - Messages sent to or from a service are fully self-described
  - After executing an operation at a service, that component forgets everything about the caller (referred to as a stateless execution)



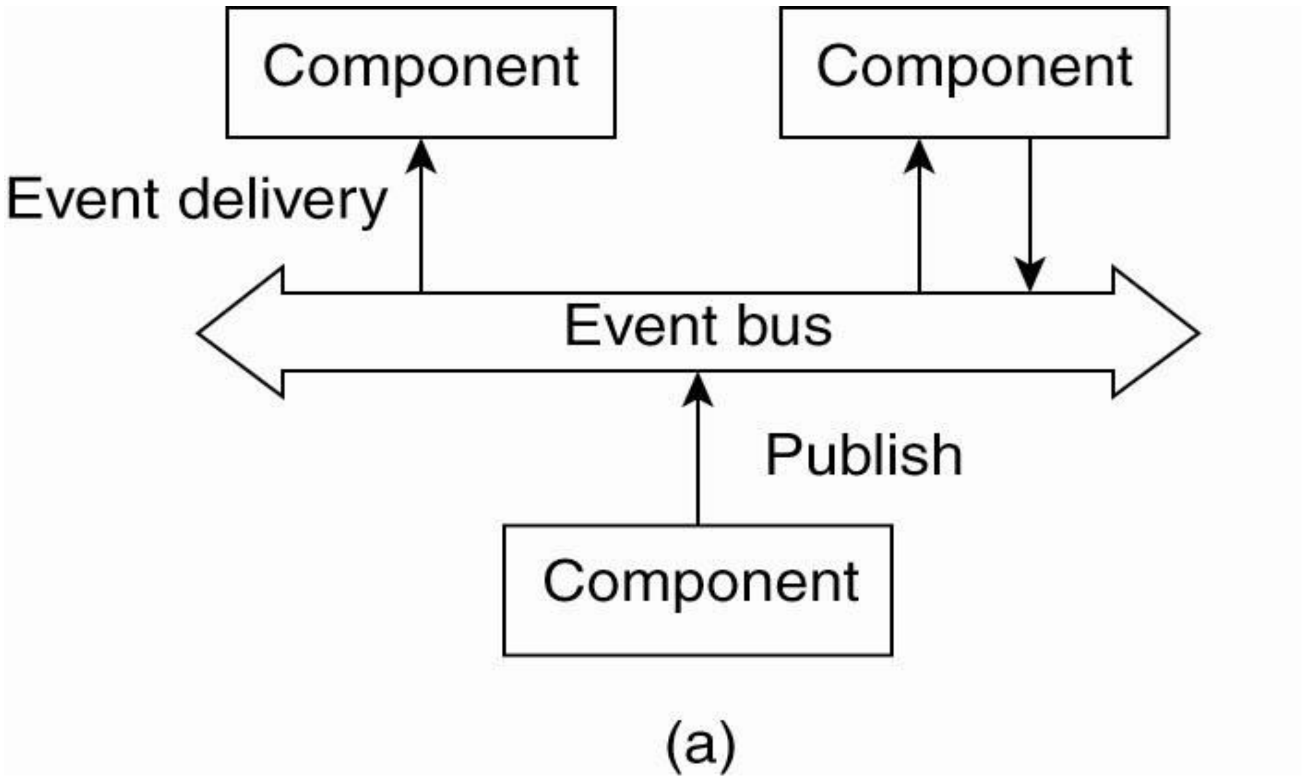
Operation	Description
PUT	Create a new resource
GET	Retrieve the state of a resource in some representation
DELETE	Delete a resource
POST	Modify a resource by transferring a new state

Publisher and Subscriber



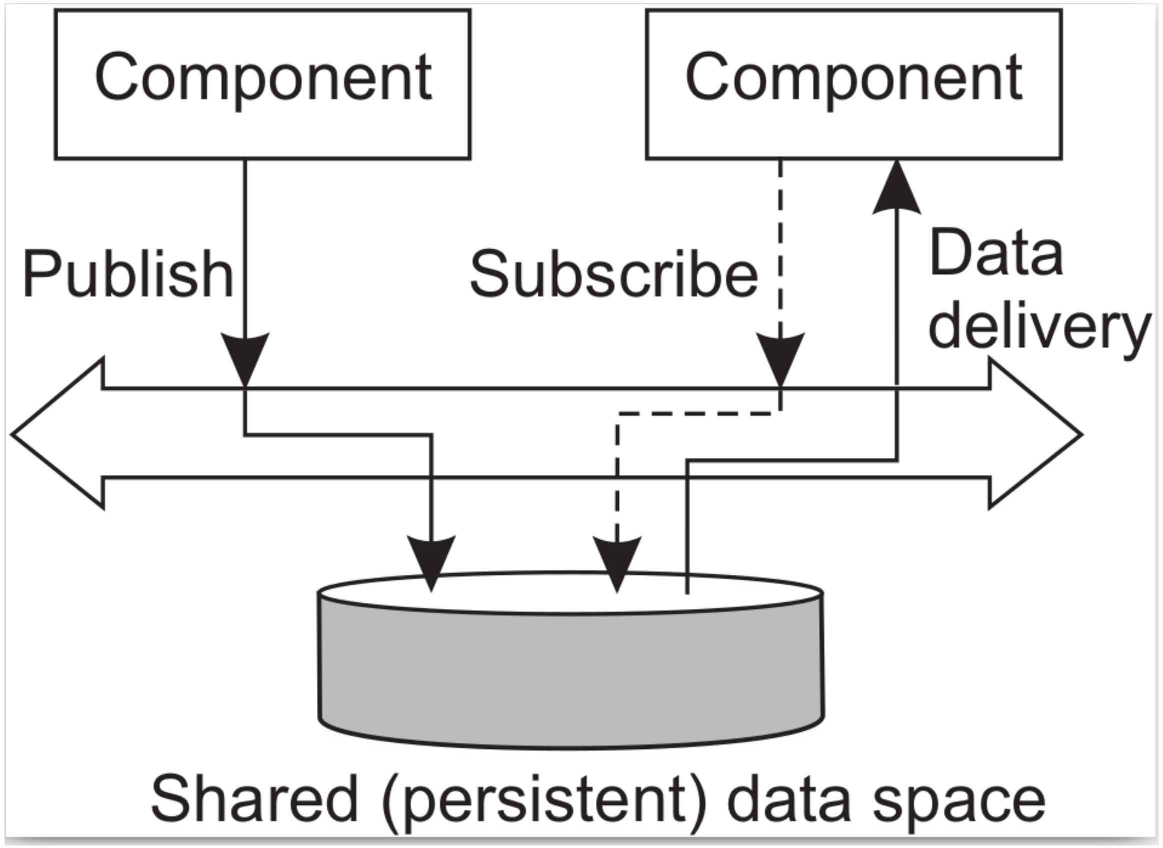
The Principle of the exchanging data items between publishers and subscribers

Event-Based Style



Processes communicate through the propagation of **events**

Shared-data Style

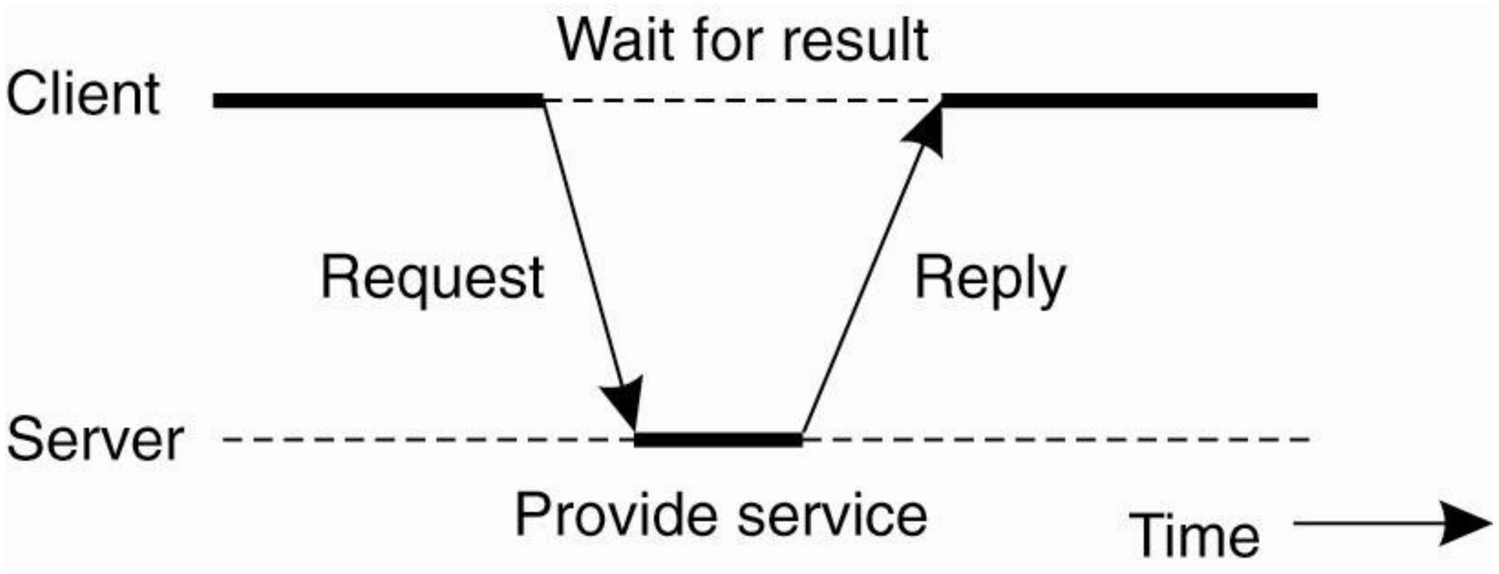


Processes communicate through **a common repository**;

System Architecture

- Instantiate and place software components on real machines
- Important (control) architectures
  - Centralized
  - Decentralized
  - Hybrid

Centralized Architectures

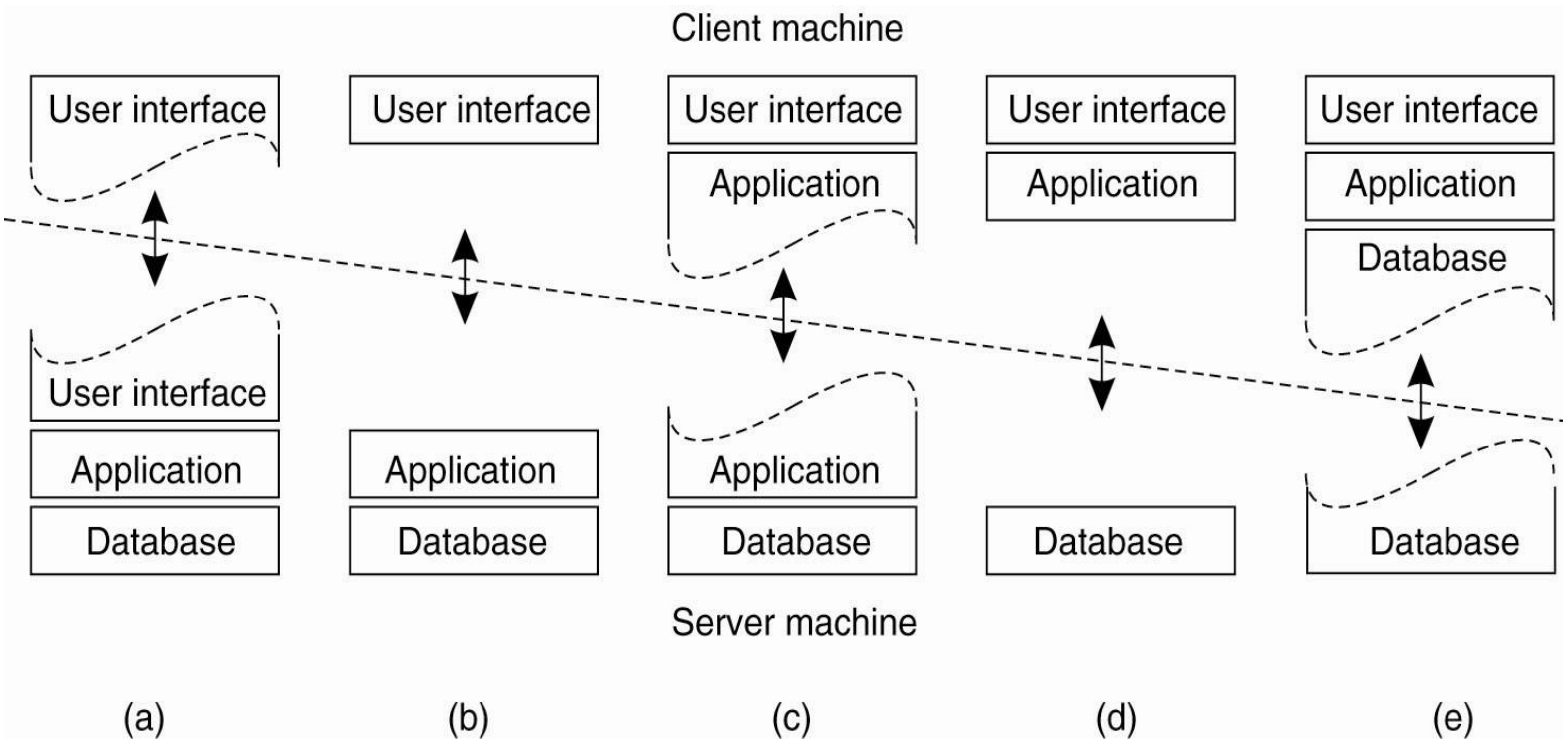


Client-server model; Two process groups:

- what is a server?
- what is a client?
- aka **request-reply behavior**

Multi-Tiered Architecture

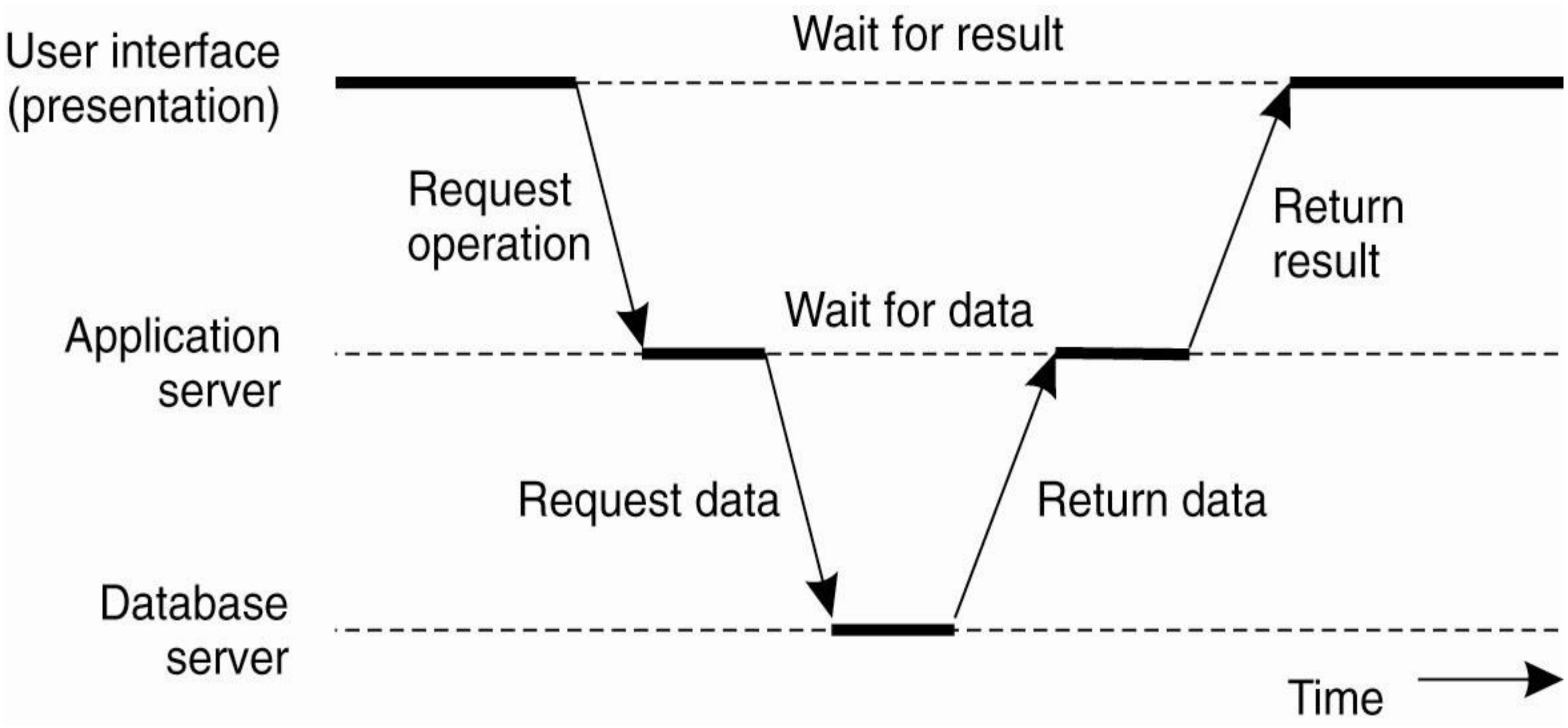
Physically distribute a client-server application across several machines => **Multi-tiered architectures**



Centralized Architectures

Examples of **multi-tiered architectures**:

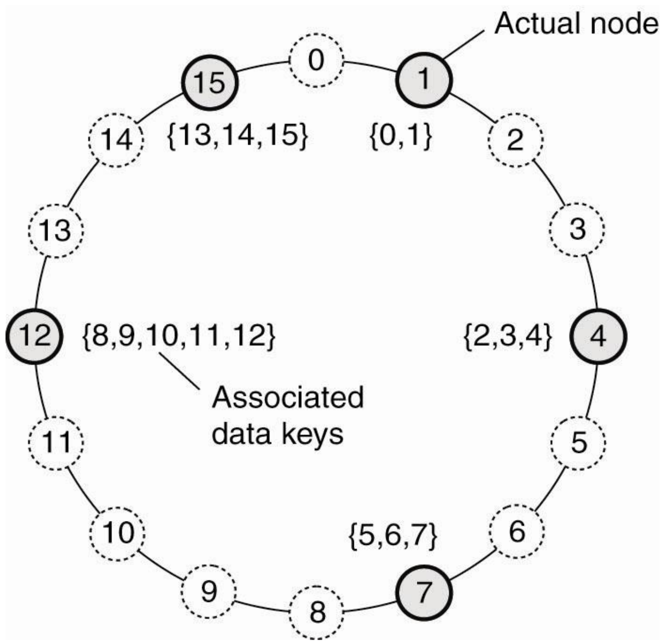
a single server is being replaced by multiple servers running on different machines



Decentralized Architectures

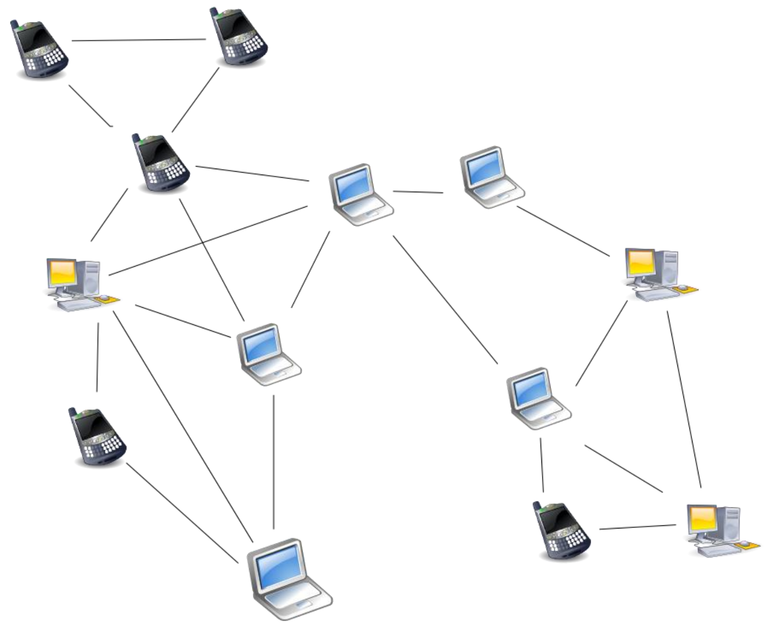
- **Peer-to-peer systems**
  - The processes that constitute a p2p system are all equal
  - Much of the interaction between processes is symmetric--- each proc will act as a client and a server
  - Focuses on how to organize the processes in **an overlay network**
    - Structured vs. unstructured

- **Structured P2P architectures**
  - Nodes (i.e., processes) are organized in an overlay that adheres to a specific, deterministic topology:
    - a ring,
    - a binary tree,
    - a grid, etc.



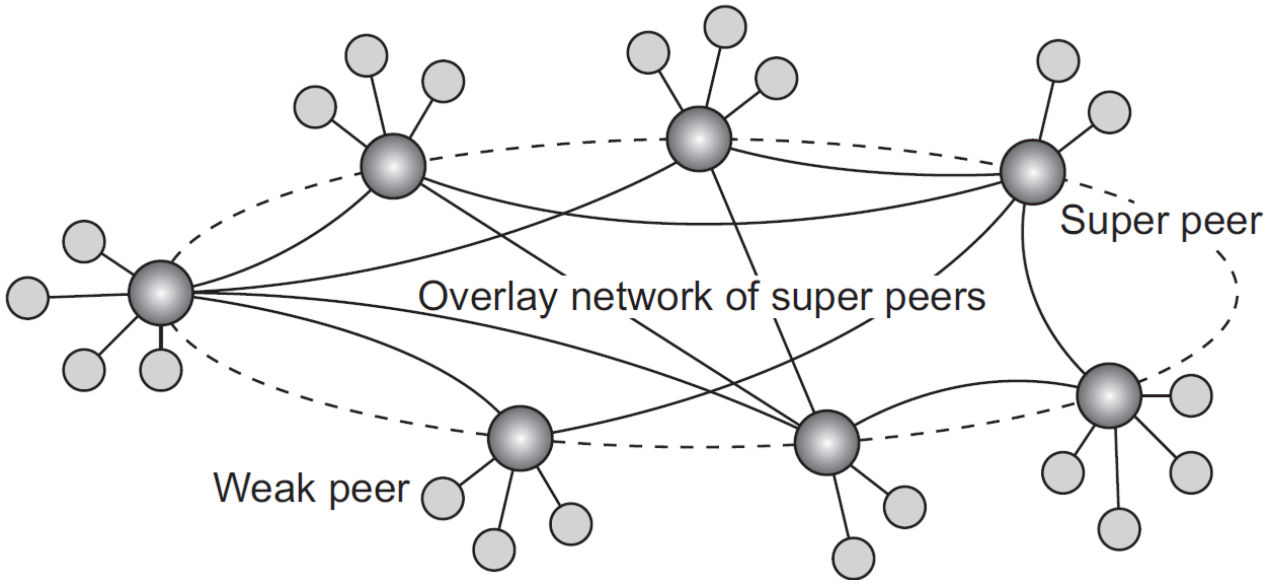
Decentralized Architectures

- **Unstructured P2P architectures**
  - Each node maintains an ad hoc list of neighbors.
  - When a node joins it often contacts a well-known node to obtain a starting list of other peers in the system.
  - This list can then be used to find more peers:
    - Flooding
    - Random walk
  - The resulting overlay resembles what is known as a random graph



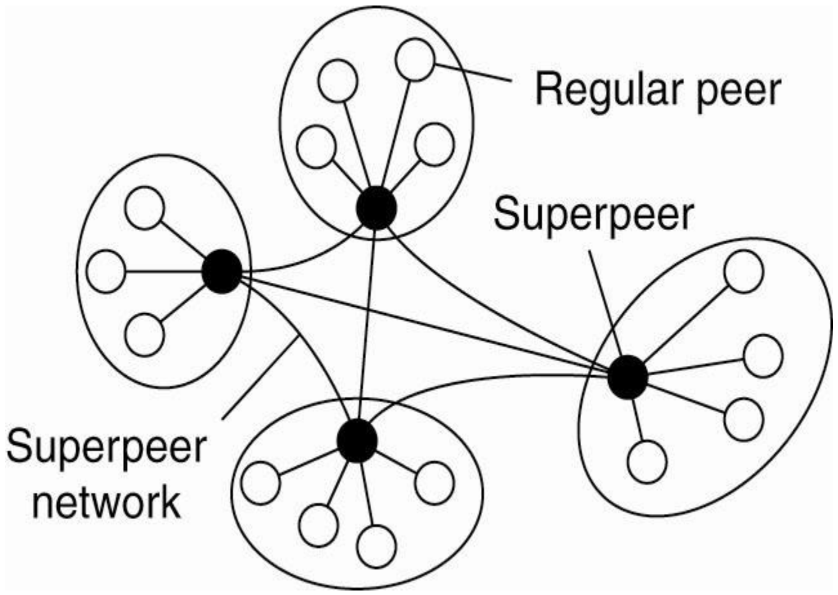
Hierarchical P2P Network

- Super peers, weak peers, overlay network
- Example: the Skype network



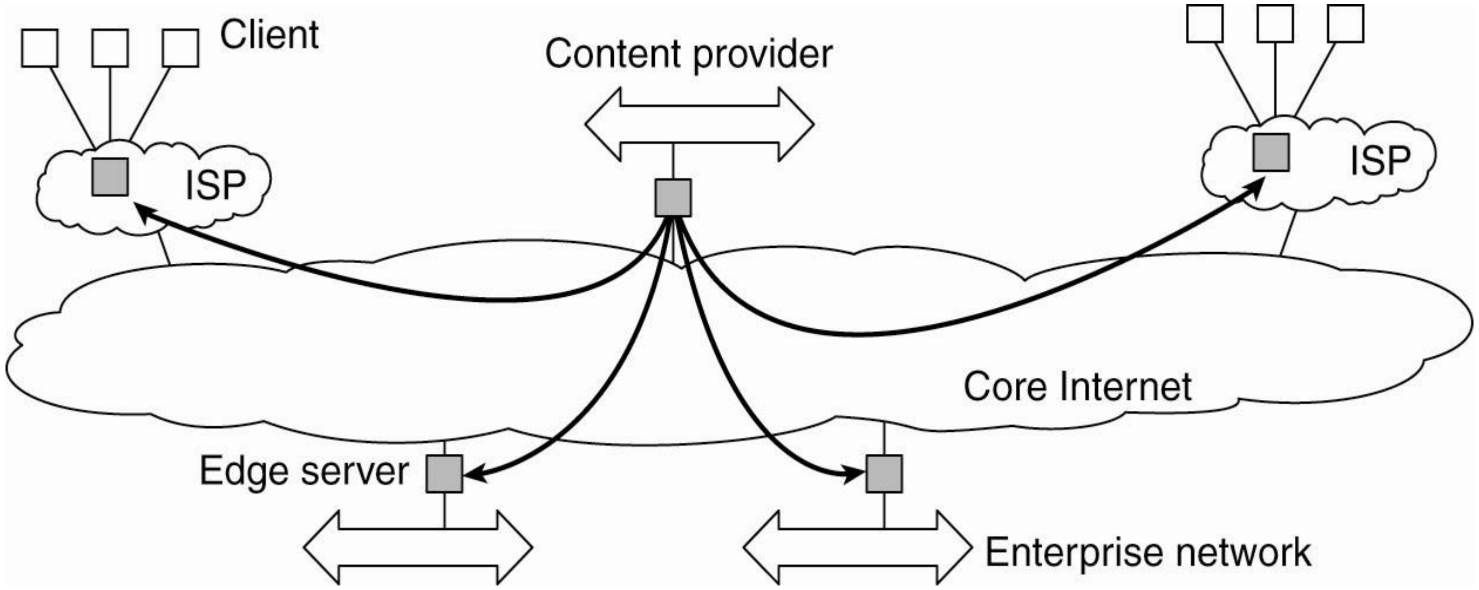
Decentralized Architectures

- How to locate relevant data in unstructured P2P systems?
- One solution: using superpeers



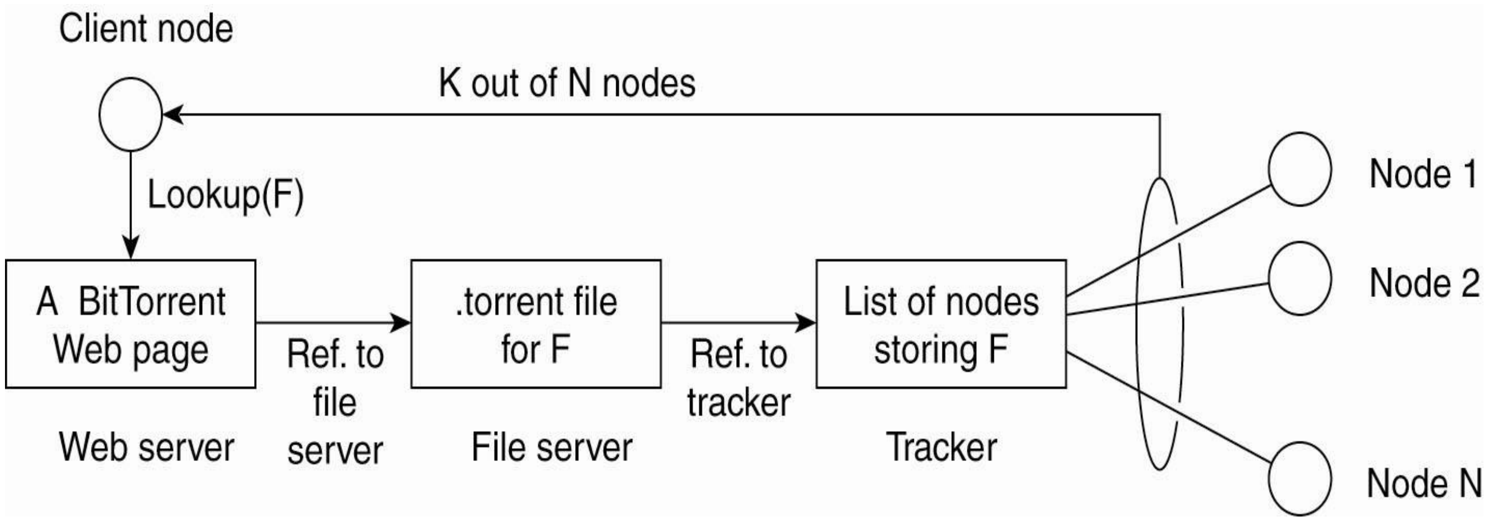
Hybrid Architectures

- Client-server solutions are combined with decentralized architectures
- **Edge-server systems**



Hybrid Architectures

- **Collaborative distributed systems**
  - BitTorrent example



Summary

- Architectural styles
- Example Architecture
- Readings
  - Chpt 2 of textbook

Chapter 3: Processes

- Overview of processes and threads
  - Processes
    - Process scheduling
  - Thread
    - Why use thread
    - Thread types
  - Virtualization
  - Client
  - Server
  - Code/Process Migration

Processes

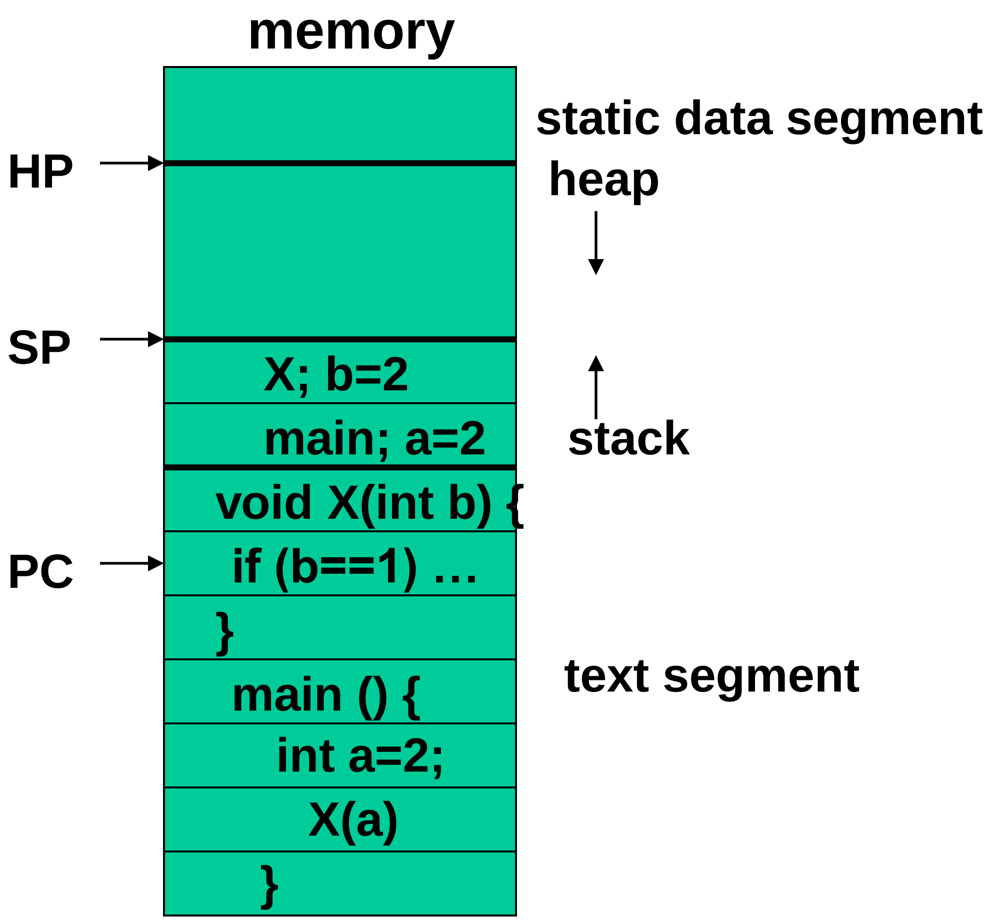
- A process is a piece of code in execution
- Processes is changed by performing a context switch
- Processes communicate either with message passing or shared memory
- A process is either New, Ready, Waiting, Running, or Terminated
- Processes are represented as PCBs in the OS
- Uniprocessor scheduling algorithms
  - Round-robin, shortest job first, FIFO, lottery scheduling, EDF

What’s in a Process?

- Dynamic execution context of an executing program
- Several processes may run the same program, but each is a distinct process with its own state
- Process state includes (without communication):
  - The code for the running program;
  - The static data;
  - Space for dynamic data (heap)& the heap pointer (HP);
  - The Program Counter (PC) indicating the next instruction;
  - An execution stack and the stack pointer (SP);
  - Values of CPU registers;
  - A set of OS resources;
  - Process execution state (ready, running, etc.)

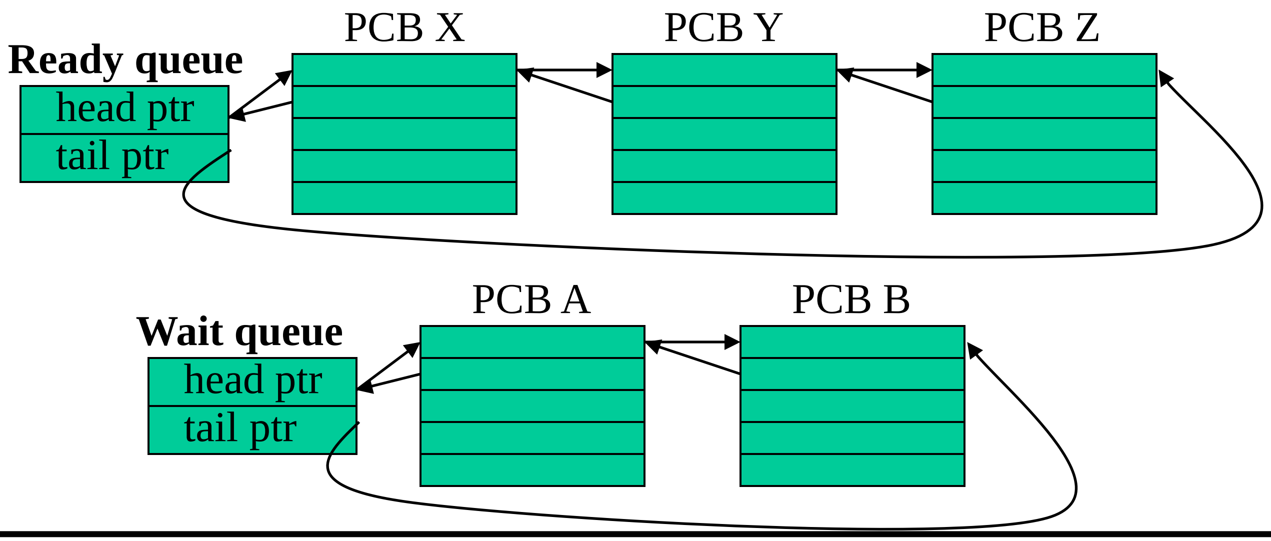
Example: Process State in Memory

```
        void X(int b) {
PC →        if (b==1) …
        }
        main() {
            int a=2;
            X(a);
        }
```



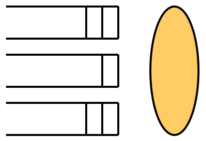
Process State Queues

- Process Control Blocks (PCBs)
  - PCBs contain process state, PC, SP, General Purpose Registers, memory management info, owner, list of open files, scheduling info, I/O status, …
- The OS maintains all the PCBs in state queues
- Each I/O device has its own wait queue



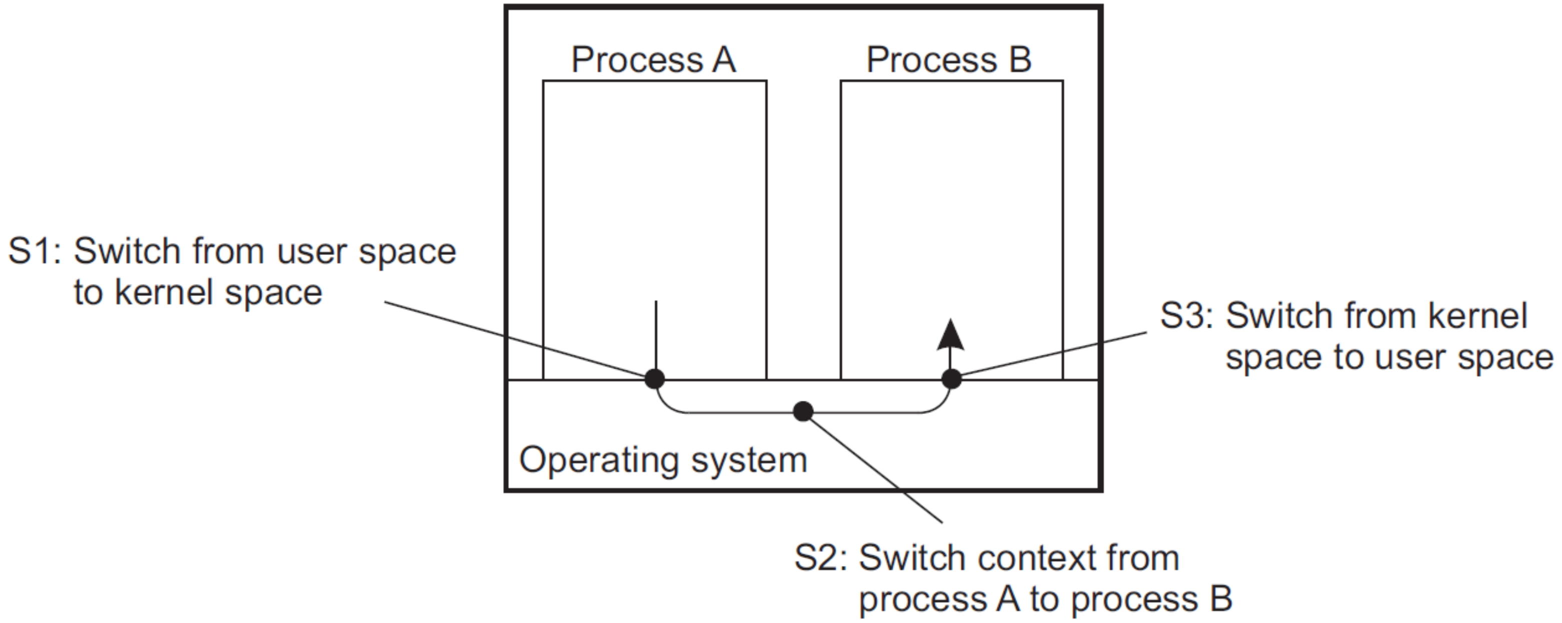
Process Scheduling

- Priority queues
  - Use strict priority scheduling
  - Example: page swapper, kernel tasks, real-time tasks, user tasks



- Multi-level feedback queue
  - Multiple queues with priority
  - Processes dynamically move from one queue to another
  - Gives higher priority to I/O bound or interactive tasks
  - Lower priority to CPU bound tasks
  - Round robin at each level

Context Switch


Process A
Process B
S1: Switch from user space to kernel space
S3: Switch from kernel space to user space
Operating system
S2: Switch context from process A to process B

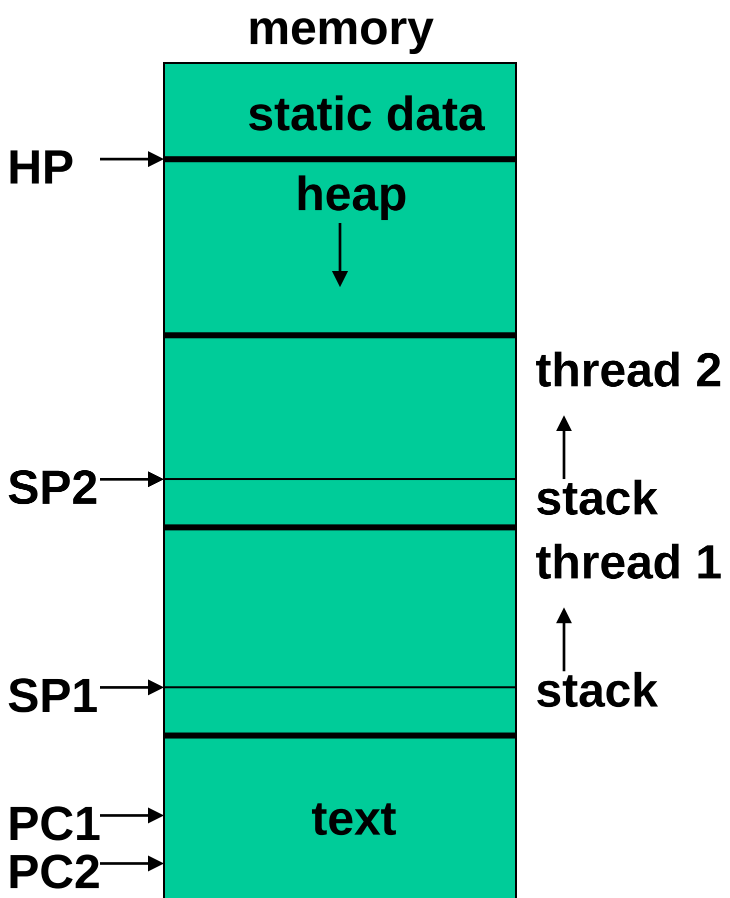
Interprocess Communication

Threads

- A thread defines a single sequential execution stream within a process
- Threads are bound to a single process
- Each thread has its own stack, PC, registers, …
- Each process may have multiple threads of control within it:
  - The address space of a process is shared among all its threads
    - share global variables
    - one thread can read/write other thread’s data (stack…)
    - no protection between threads
  - No system calls are required to cooperate among threads
  - Simpler than message passing and shared-memory

Example: Threaded Program

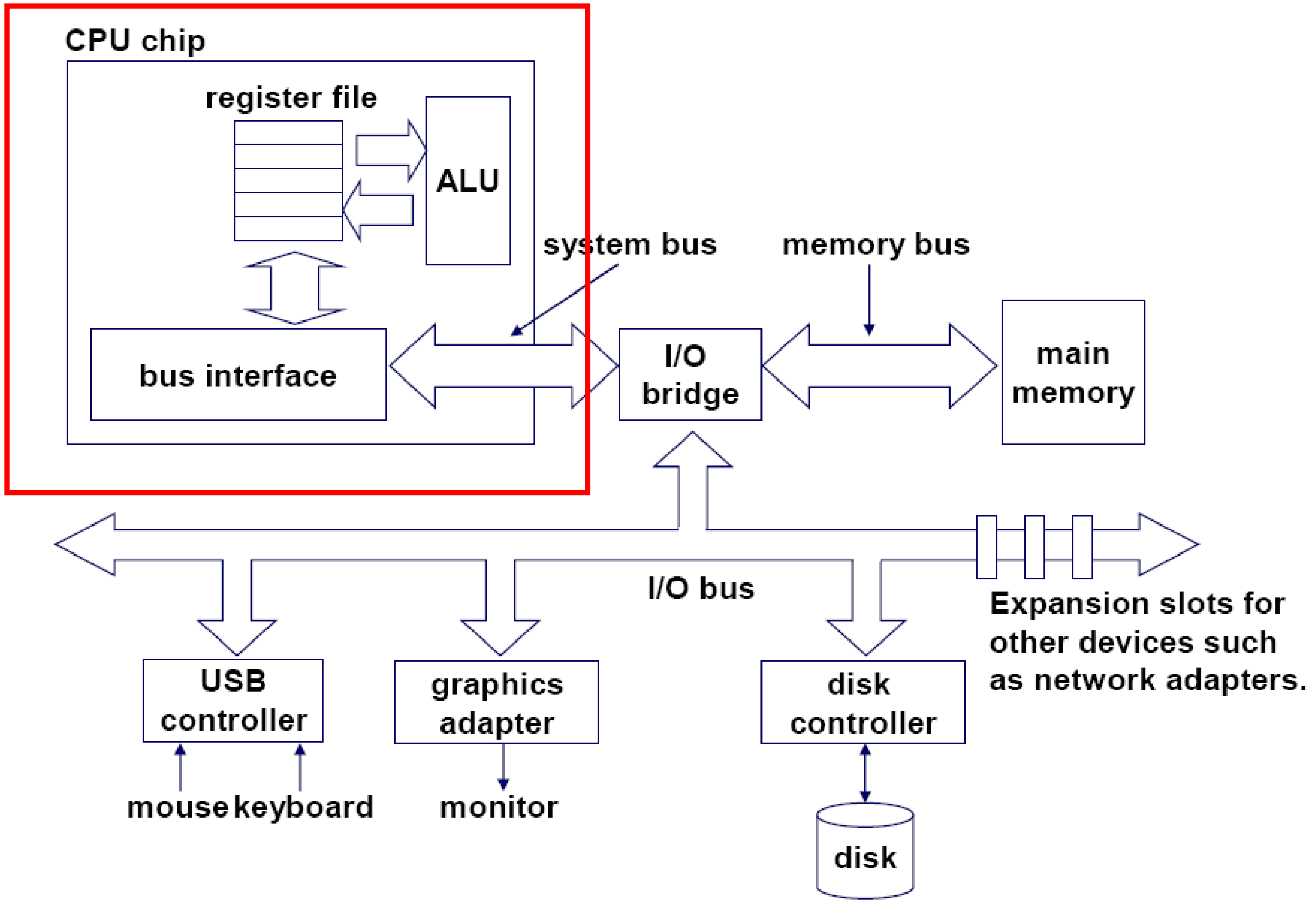
```
main
   global in,out,n,buffer[n];
   in=0;out=0;
   fork_thread(producer());
   fork_thread(consumer());
end

producer
  repeat
    nextp=produced item
    while in+1 mod n = out do no-op
    buffer[[in]=nextp; in=(in+1) mod n

consumer
  repeat
    while in=out do no-op
     nextc=buffer[out]; out=(out+1) mod n
     consume item nextc
```



Example: Threaded Program

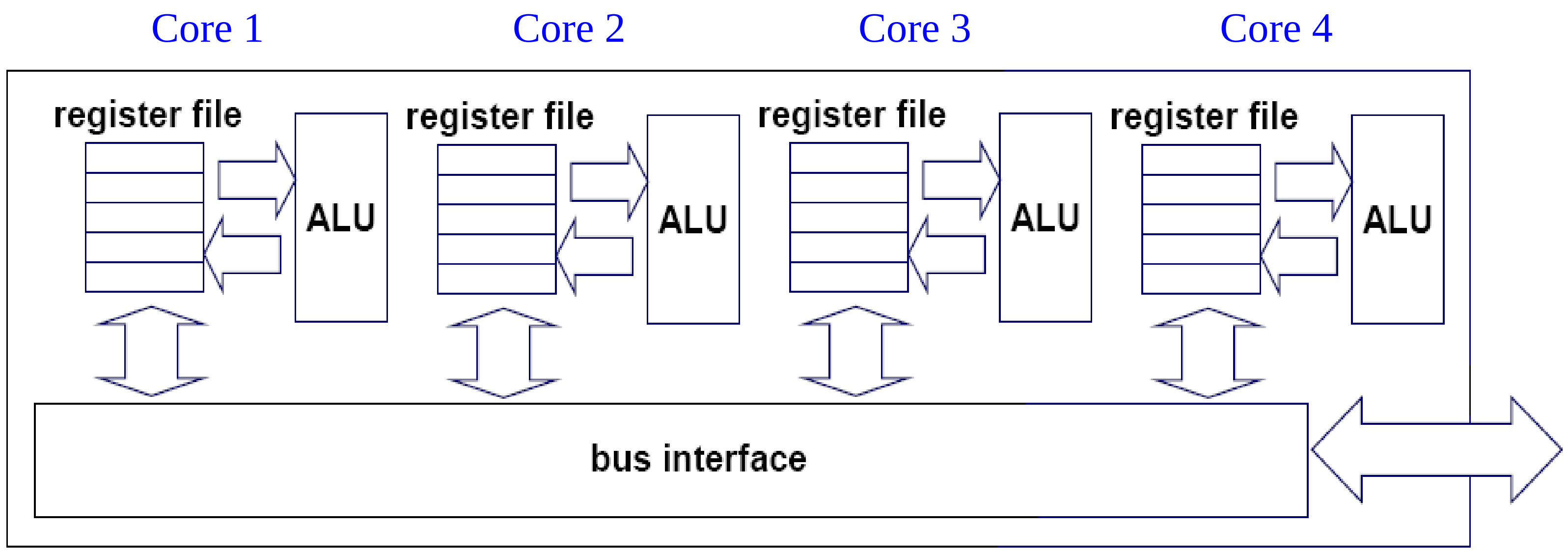
```
main
   global in,out,n,buffer[n];
   in=0;out=0;
   fork_thread(producer());
   fork_thread(consumer());
end

producer
  repeat
    nextp=produced item
    while in+1 mod n = out do no-op
    buffer[[in]=nextp; in=(in+1) mod n

consumer
  repeat
    while in=out do no-op
     nextc=buffer[out]; out=(out+1) mod n
     consume item nextc
```

Single Core Computer

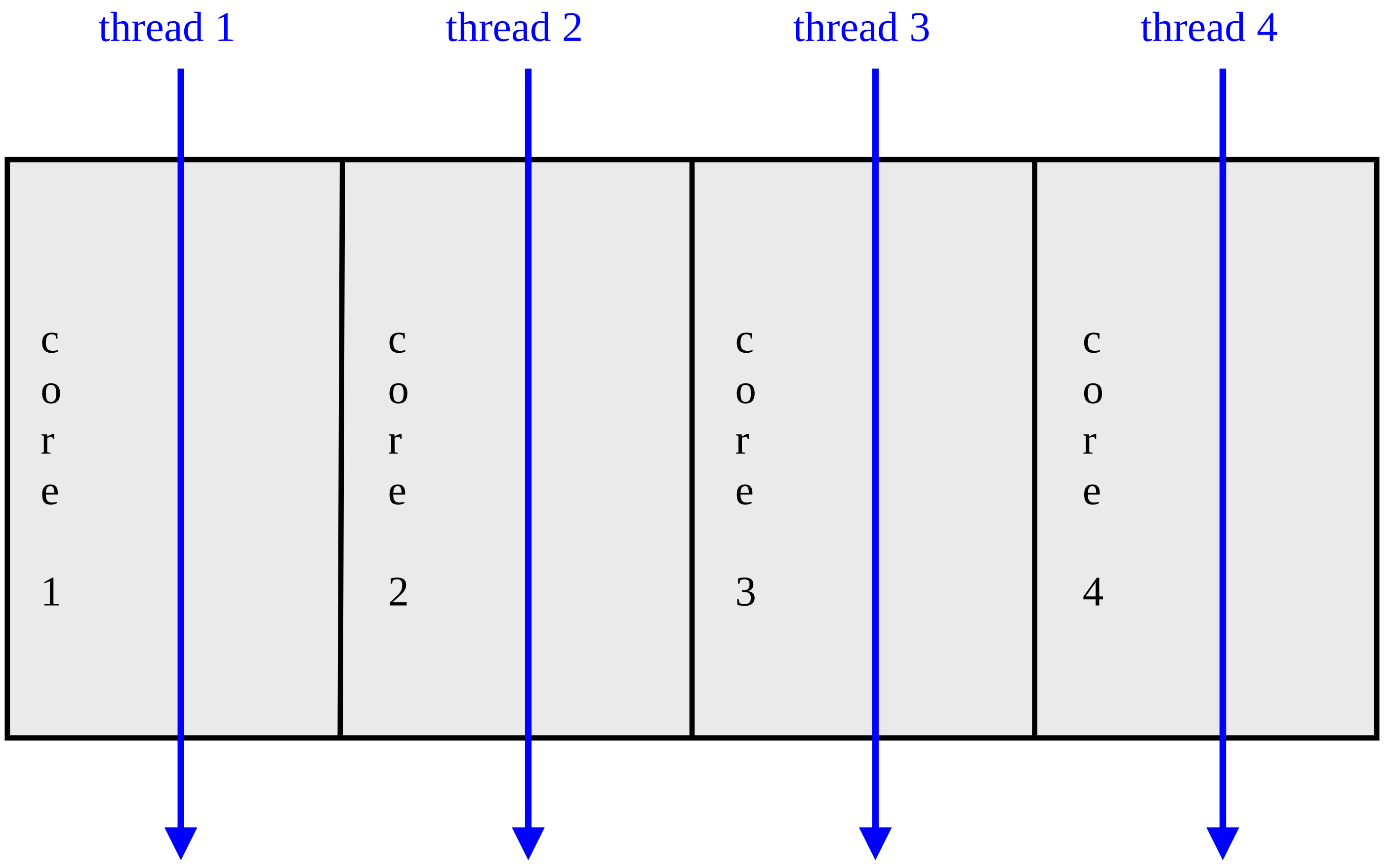


Multicore CPUs

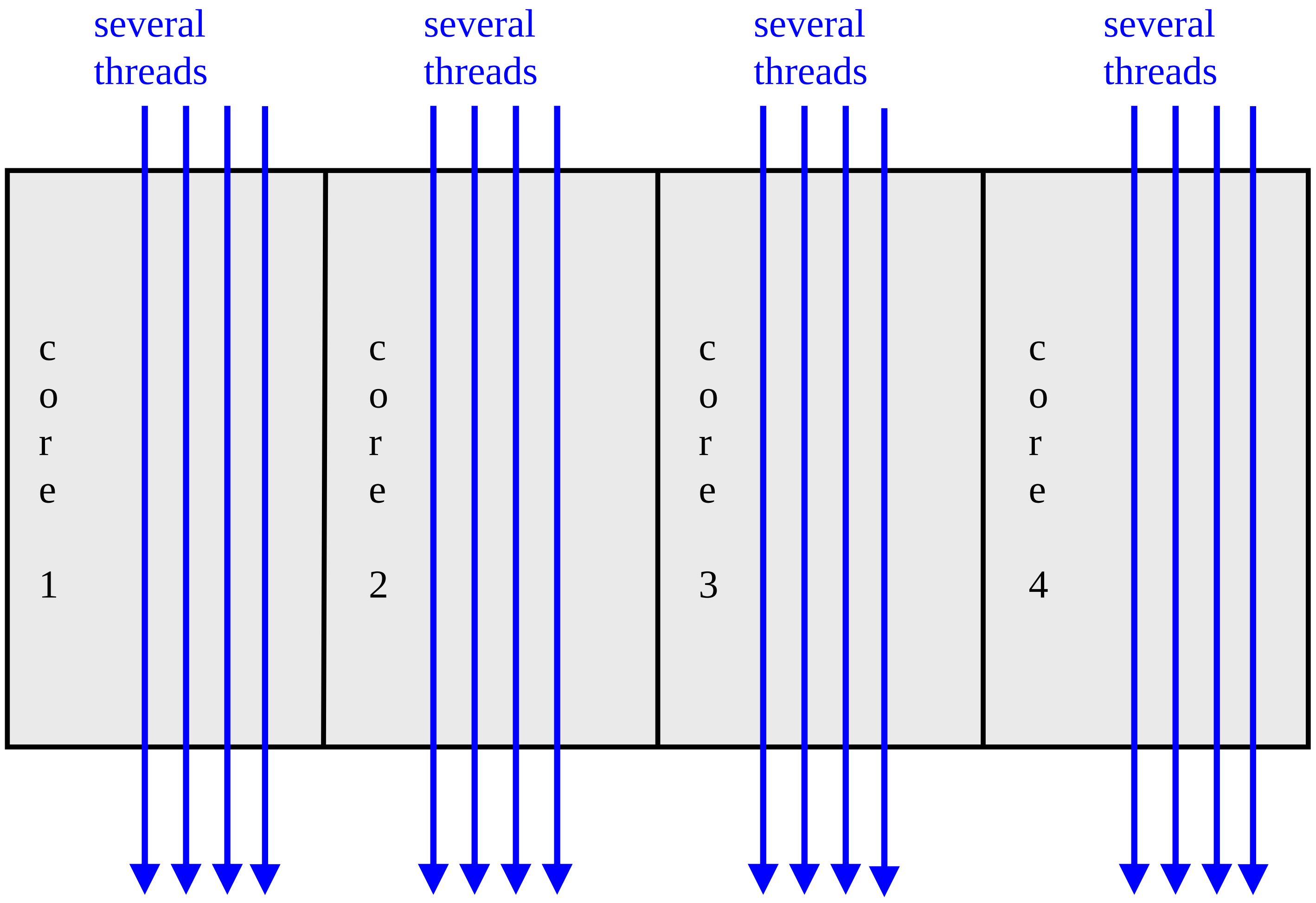
- Most of current Intel and AMD multicore CPUs just put multiple cores



Single-threaded Multicore CPUs



Multi-threaded Multicore CPUs



Thread Management

- Creation and deletion of threads
  - Static versus dynamic
- Critical sections
  - Peer threads
  - Synchronization primitives: blocking, spin-lock (busy-wait)
  - Condition variables
- Global thread variables
- Kernel versus user-level threads

Static vs. Dynamic Threads

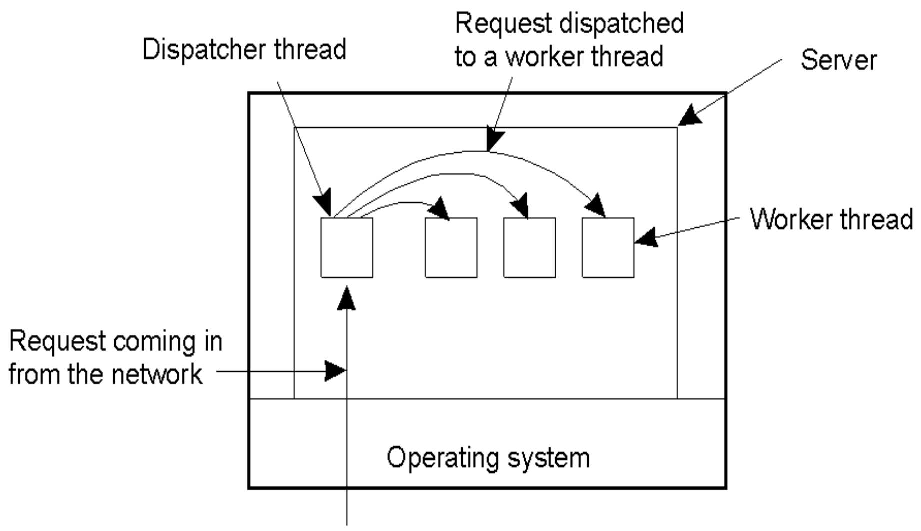
- Static threads
  - Number of threads decided when program written or compiled
  - threads allocated fixed stack
- Dynamic threads
  - Created and destroyed dynamically
  - Create call specifies
    - main program; stack size; scheduling priority …
  - Normally process starts with one thread
    - can create threads as necessary
  - Termination
    - voluntary (finish job)
    - involuntary (killed from outside)

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


Dispatcher thread
Request dispatched to a worker thread
Server
Worker thread
Request coming in from the network
Operating system