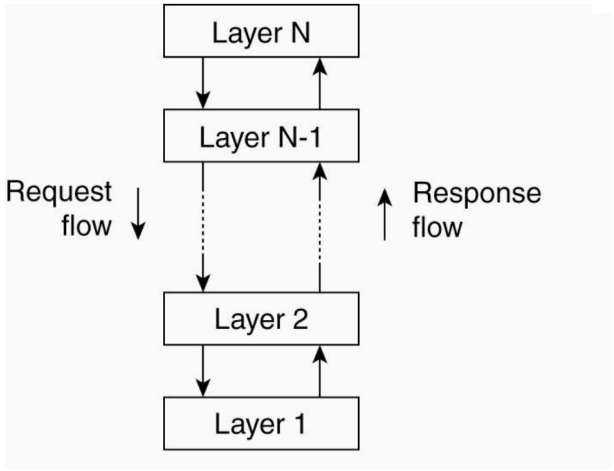
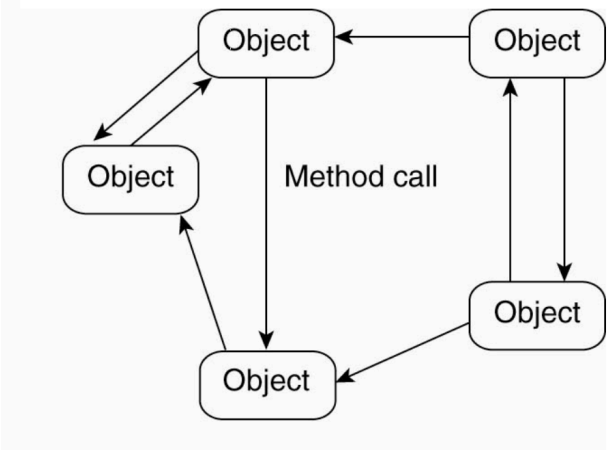
Lecture 1

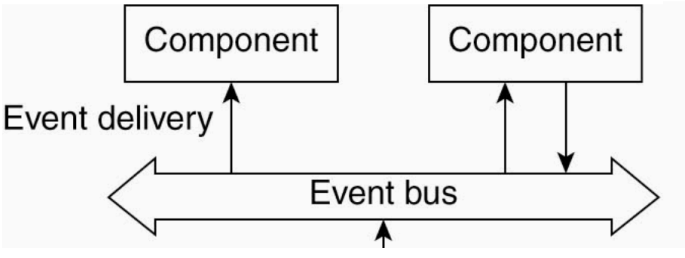
Monday, 7 August 2017 2:28 PM

Distributed system and network principle

- Software architecture:
  - Component organization
    - Layer-based architecture requests (resp. response) do downward (resp. u
    - Network-based architecture communicate through Remote Procedure Cal
  - Communication organization:
    - Event-based architecture: communication through events, that optionally



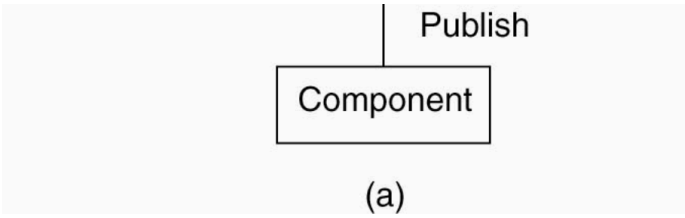




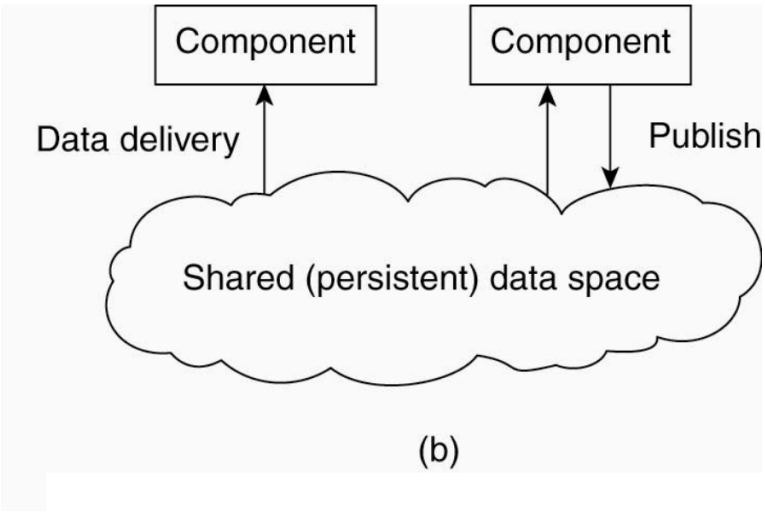
pward)

lls (RPCs)

carry data (subscribers get their desired events delivered)



- Data-centered architecture: through a shared repository, that contains da



- The client-server model:
  - The client server model:
    - The basic client -server model:
      - The client requests the service whereas the server provides the serv
      - The client and the server can be hosted on different machines
    - Stateless and stateful server:
      - Stateless server: does not record the state of its clients
      - Stateful server: maintains persistent information about its clients (c

	Stateless server	Stateful server
State	No info kept	Persistent info
Request	Self-contained	Can be split, generally faster
Upon failure	No recovery needed	State recovery needed (explicit
Example	Network file system (NFSv3)	Andrew file system (AFS)

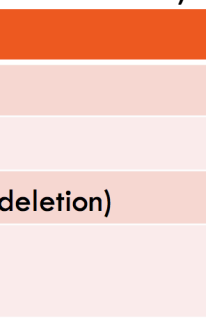
- The layered organization
  - Application layering:
    - Traditional three-layered views:
      - The user interface layer: contains the feature to control the applicat
      - The processing layer: contains the function of the application
      - The data layer: contains the data of the application

        Example: a search engine request spanning the traditional three layers

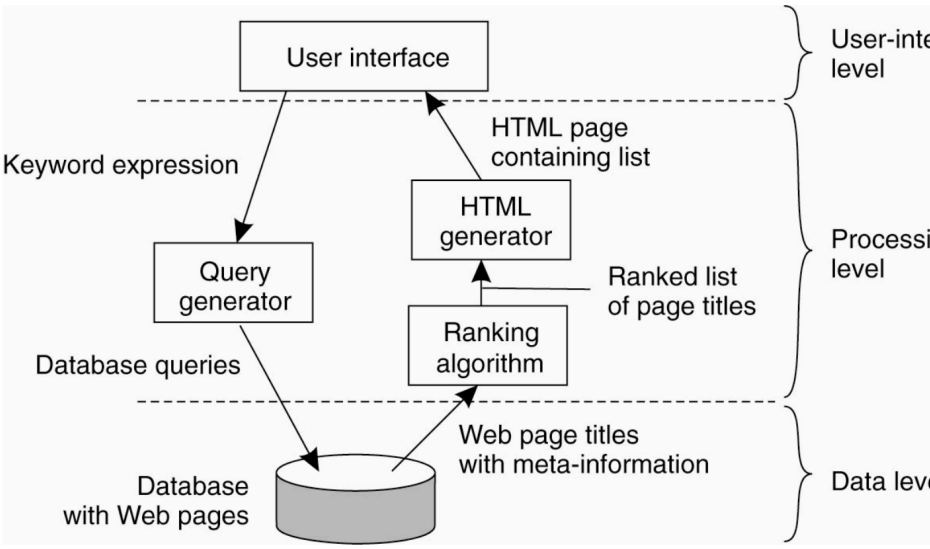
ta (e.g. files in a distributed file system)

vice

lient -->file)

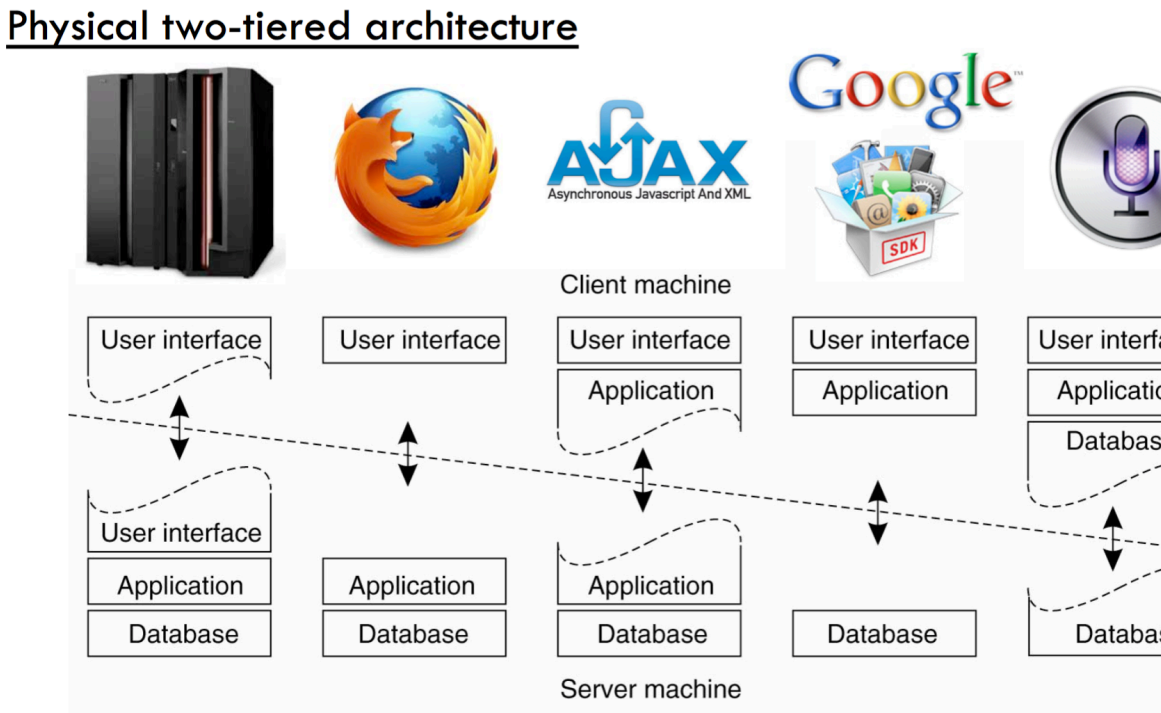


tion

- 

- Hosting different layers on different machines:
  - Three-tiered architecture:
    - each layer on separate machine
  - Two-tiered architecture:
    - Client
    - Single server configuration
  - Single-tiered architecture:
    - Dumb terminal
    - Mainframe configuration

- Multi-tiered architectures:



A single machine can act both as a client and a server

erface

ing

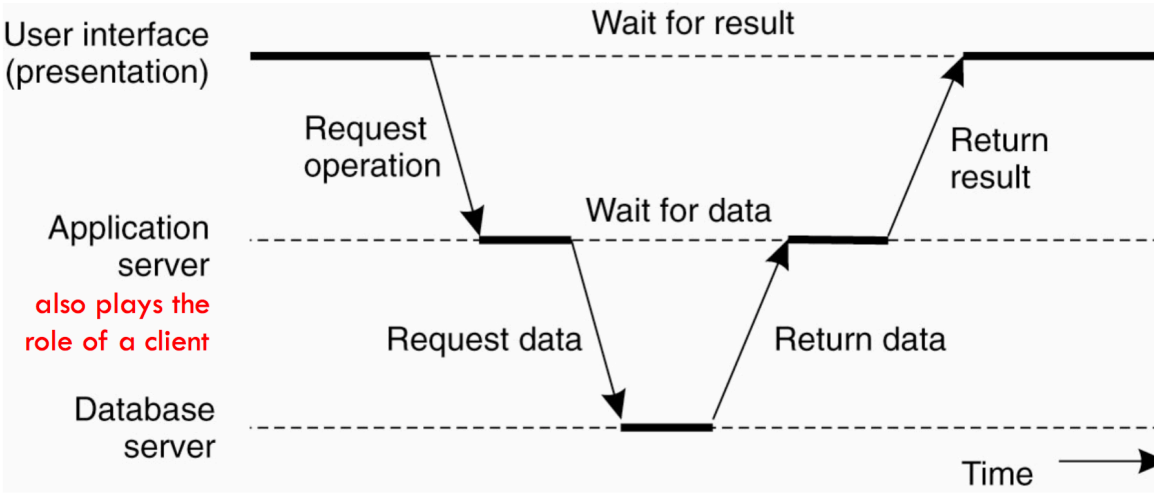
el

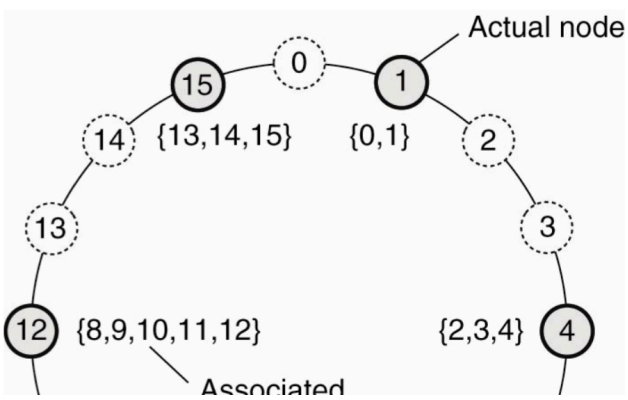
ace

on

se

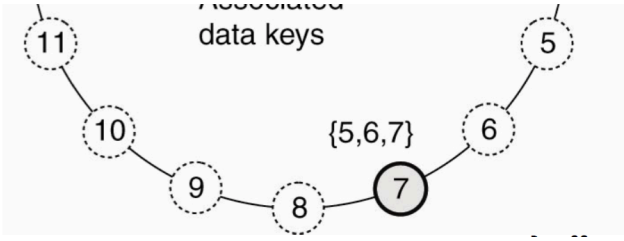
se

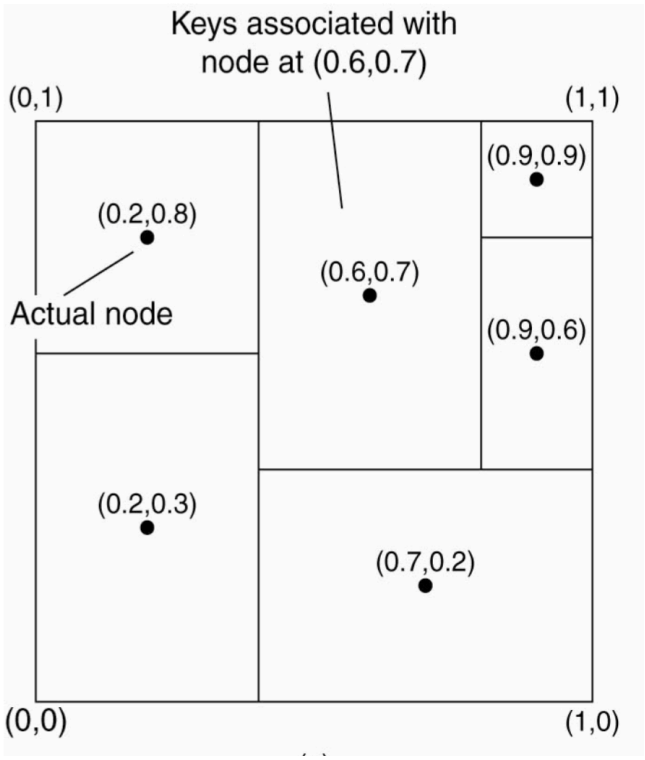
- 

    - Example:
        - Cloud computing:
            - Cloud computing: the delivery of computation or storage as a
            - The client hosts the user interface to launch the computation
            - The servers handle most of the computation upon request an
                - One server asks data to another server
                - Another does the computation
- Peer-to-peer organization:
    - P2P model:
        - Every machine acts similarly
            - Every machine is both a client and a server
            - Not centralized control: the responsibility is distributed evenly
            - Even the program executing on each machine is similar
        - Examples:
            - Chord, a Distributed Hash Table (DHT)
                - As a node:
                    - Have a successor peer
                    - Have a predecessor peer
                    - Have some shortcuts to other nodes to speedup deliver
                    - Be responsible of a subset of the system data items (bas
                - 

service to end-users
and prints the results
d sends back the results to the client:

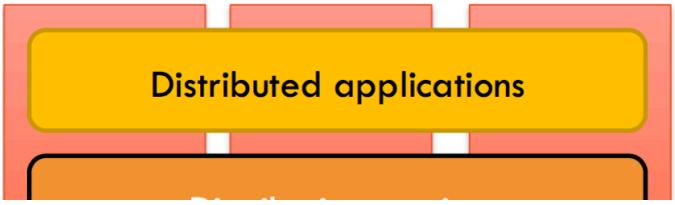
y of requests
sed on my unique identifier)



- Content Addressable Network (CAN), another DHT
  - As a node:
    - Be responsible of a region of the system (based on my u
    - Have few neighbors, the nodes with adjacent regions, w
  - 

- BitTorrent, a file sharing application
  - Used for LINUX distribution, software patches, distributing
  - Goal: quickly replicate large files to large number f clients
  - Web server hosts a .torrent file
  - A tracker (server or a DHT) tracks downloaders/ owners of a fi
  - Files are divided into chunks
  - Downloaders download chunks from themselves (and owner)
  - Tit-for-tat: the more one shares (server), the faster the it can

- Distributed operating system:
  - Distributed operating system:
    - A single system image, the system maintains a single copy of the resource

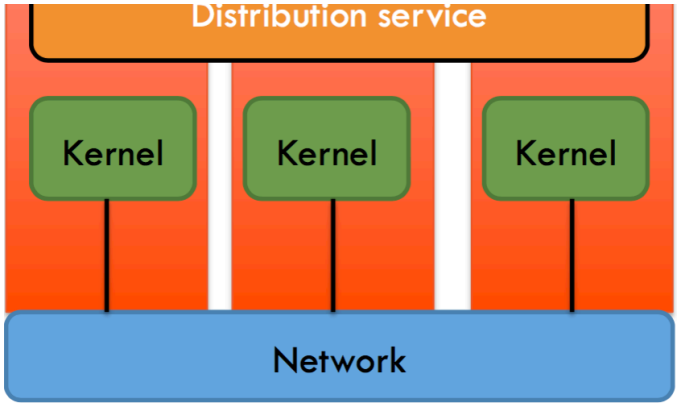


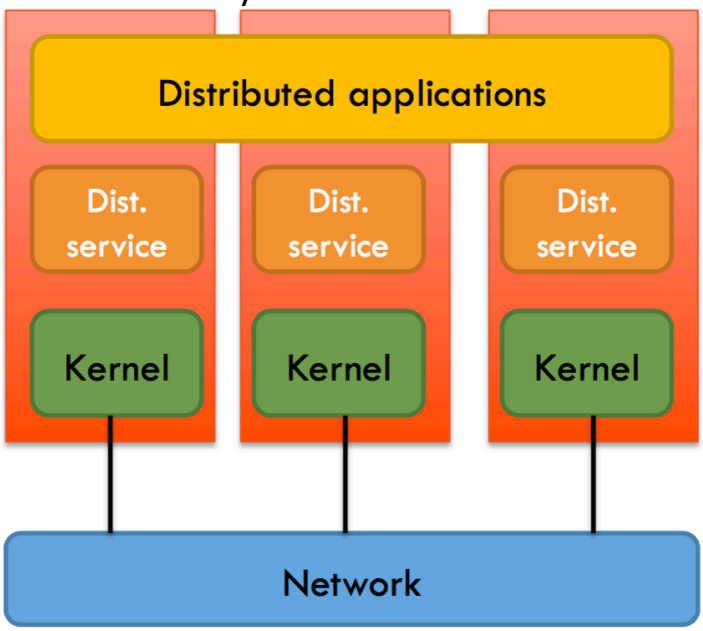
unique identifier)
with which I can communicate

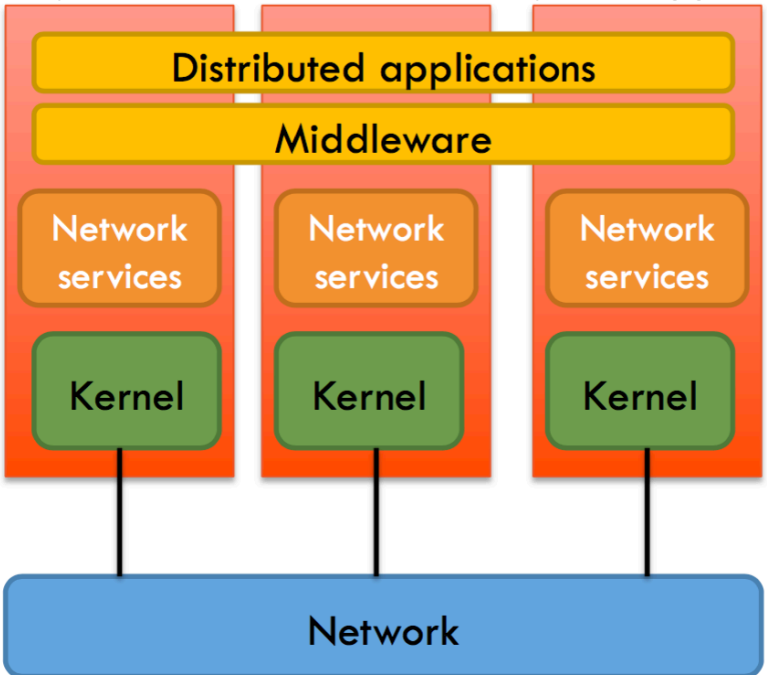
ile

download (client)

s

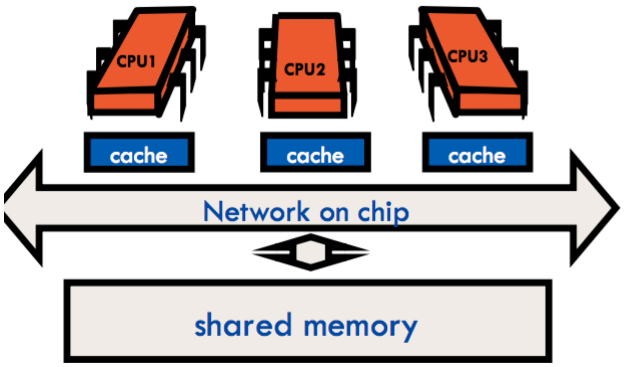
- 

- Network operating system:
  - Machines provide resources to other machines (e.g., UNIX rlogin)
  - The OS can vary from one machine to another, essentially file sharing
  - 

- Middleware:
  - A layer over the network services providing general services to applicatio
  - 

ns in a vary transparent manner (system can differ)

Concurrency

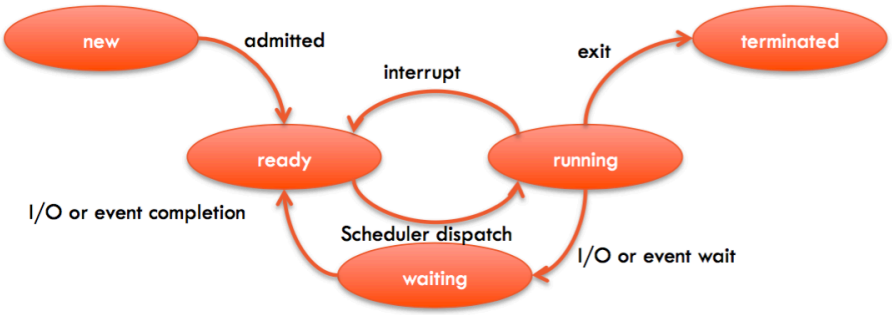
- Uniprogramming and multiprogramming
  - Uniprogramming: one program execution at a time
    - MS/DOS
    - Early macintosh
    - Batch processing
  - Many program execution on personal computers:
    - Browsing the web
    - Sending emails
    - Typing a letter
    - Burning a DVD
  - Multiprogramming: more than one program execution at a time
    - Multics
    - UNIX/Linux
    - Windows NT
    - Mac OS X
  - Multi/many-cross
    - 

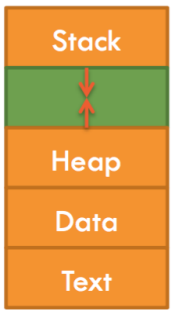
- Concurrency:
  - Assume a single central processing unit (CPU)
  - The way to give the illusion to the users of multiple CPUs:
    - Answer: Scheduling program execution, one after the other during sm the time
      - 

Processes and threads:

- Process:
  - An abstraction representing a program execution

all fractions of

        - When to switch from one process to another?
            - Timer interrupt goes off, explicit yield, I/O, etc.
        - Each virtual CPU needs to store the process context:
            - Program counter, stack pointer, other registers
        - A process switches from a state to another:
            -


- Address space:
    - A unit of management of a process's virtual memory
    - Process address space:
        - Stack: temporary data extensible towards lower virtual addresses
        - Heap: memory allocated dynamically extensible to higher virtual addr
        - Data section: global variable
        - Text region: program code
        -


- Process management:
    - Process creation:
        - Process allocation: the act of choosing the host of the process
            - The system V ("5") provide command to execute a process at an workstation
            - The amoeba system choose a host from a pool of processors to
        - Execution environment: an address space w/ initialized content and o
            - Static: program text, heap and stack regions created from a list
            - Dynamic: UNIX fork shares program text and copies stack and he
    - Interprocess cpmmunication (IPC):
        - Pipes/message queue/shared memory segments
        - Requires costly context switches
- Context-switch:
    - An interrupt requires kernel intervention
        - S1. process a switches from user to kernel mode
            - Changing the memory map in the MMU (memory management

esses

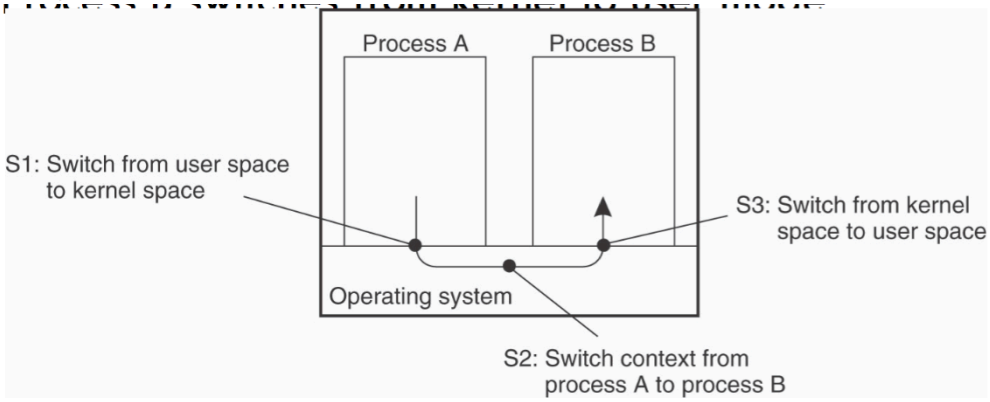
idle

execute it
pen files:

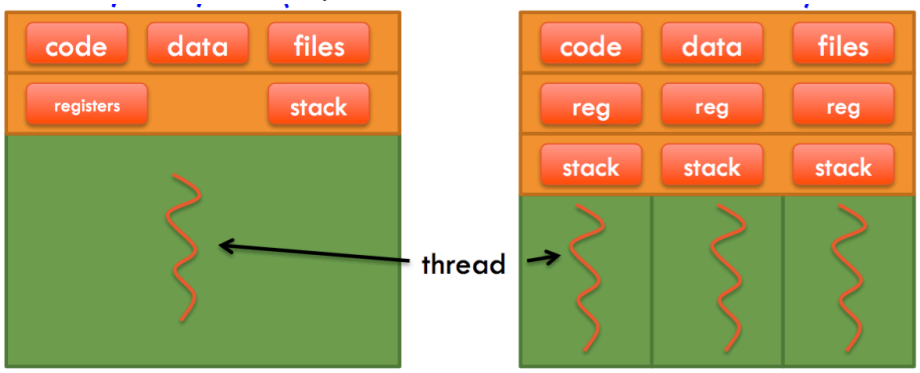
eap regions

unit)

        - Changing the memory map in the MMU (memory management
        - Flushing the TLB (translation lookaside buffer)
    - S2. context-switch (swapping processes from disk to main memory)
    - S3. process B switches from kernel to user mode
    -



- Thread:
    - Smallest unit of CPU utilization
        - Portable OS interface (POSIX) threads
            - Current activity: program counter
            - Stack: temporary data
            - No data, code, files (but it shares them with other threads)
            -



        - Communication between threads always through memory
            - Does not need IPC
            - No context switches required (when purely at user-level)
        - User-level threads library and kernel-scheduled thread
            - User-level thread library:
                - Cheap to create and destroy
                - Blocking system call freezes the entire process of the threa threads)
                - Cheap context-switch:
                    - Few instruction to store and reload CPU register valu
                    - No need to change memory map in MMU
                    - No need to flush the TLB

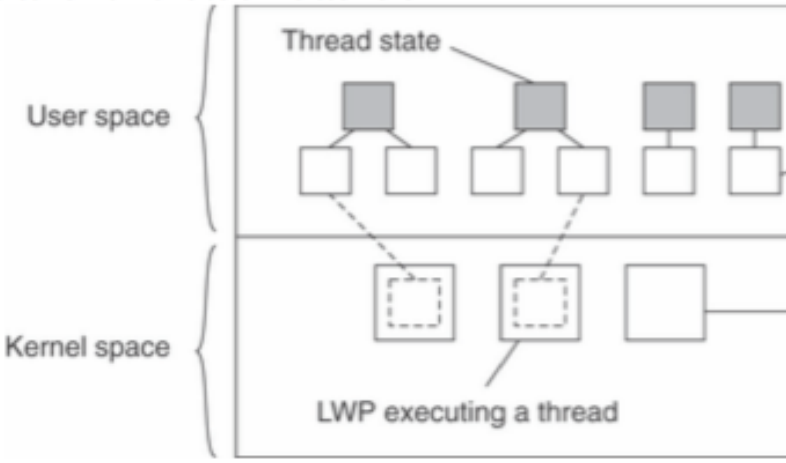
            - Kernel-scheduled threads:

dmit)

ad (and other

ues

        - Costly to create and destroy
        - Do not block the current process (and other threads) upon system call (I/O)
        - Costly context-switch (similar to process context switch)
            - Needs to change memory map in MMU
            - Needs to flush the TLB
- User-level threads library and kernel-scheduled thread
    - Lightweight processes (LWP):
        - Runs in the context of a process (potentially multiple LWP
        - Current threads are in a table
        - Once an LWP finds a runnable thread:
            - It locks the table in user-mode to update it
            - It switches context to that thread in user-mode
        - Does not block upon system calls:
            - Passing from user to kernel mode in the LWP contex
            - If the LWP is blocked, the kernel context-switches to
            - Implying a context-switch also at the user level
            -

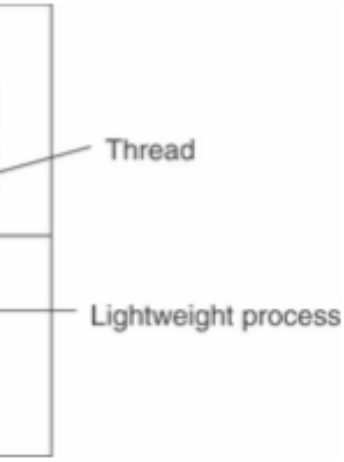


- Process and thread:
    - Process:
        - Isolated: prevents one process from interfering with anoth
        - Inefficient: starting/ terminating a process and context sw
    - Threads:
        - Non-isolated: avoiding incorrect interferences makes prog
        - Efficient: a thread is a lightweight version of a process
- Client multi-threading
    - Ex: Google Chrome web browser
        - Multithreading: each tab runs its own thread
        - A JavaScript error does not crash the main Chrome proces
            - Only one tab freezes, the user can close it independ
        - Takes benefit of multicore architectures, each thread runs core

n blocking

s per process)

xt
o another LWP



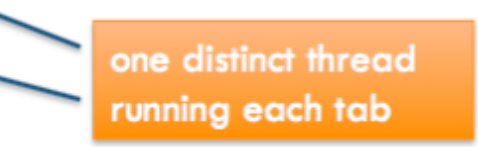
ner
witches are costly

gramming harder

s:
ently of others
s on a separate

- ◆ 

- Java threads:


one distinct thread
running each tab