Principles of Computer Science II

Cloud Computing

Marco Zecchini

Sapienza University of Rome

Lecture 9

Why do we need cloud computing?

We need to run a task on a very large dataset. We have seen several algorithm techniques that we can run on it (dynamic programming, map-reduce, clustering) depending on our goal.
What about the hardware?
My laptop has:

- A quad-core processor at 1.8 GHz
- 16GB RAM
- 512 GB SSD
- No GPU

Why do we need cloud computing?

We need to run a task on a very large dataset. We have seen several algorithm techniques that we can run on it (dynamic programming, map-reduce, clustering) depending on our goal.
What about the hardware?
My laptop has:

- A quad-core processor at 1.8 GHz
- 16GB RAM
- 512 GB SSD
- No GPU

Example

Another laptop with two quad-core processors at 2.7 GHz executes the same tasks in 2x less time.

Why do we need cloud computing?

We need to run a task on a very large dataset. We have seen several algorithm techniques that we can run on it (dynamic programming, map-reduce, clustering) depending on our goal.
What about the hardware?
My laptop has:

- A quad-core processor at 1.8 GHz
- 16GB RAM
- 512 GB SSD
- No GPU

Example

Another laptop with 32 GB RAM has double the amount of memory to store the dataset in memory (if memory ends, we cannot run the task).

Why do we need cloud computing?

We need to run a task on a very large dataset. We have seen several algorithm techniques that we can run on it (dynamic programming, map-reduce, clustering) depending on our goal.
What about the hardware?
My laptop has:

- A quad-core processor at 1.8 GHz
- 16GB RAM
- 512 GB SSD
- No GPU

Example

Another laptop with a GPU can run the task 10x faster (or more).

Why do we need cloud computing?

We need to run a task on a very large dataset. We have seen several algorithm techniques that we can run on it (dynamic programming, map-reduce, clustering) depending on our goal.
What about the hardware?
My laptop has:

- A quad-core processor at 1.8 GHz
- 16GB RAM
- 512 GB SSD
- No GPU

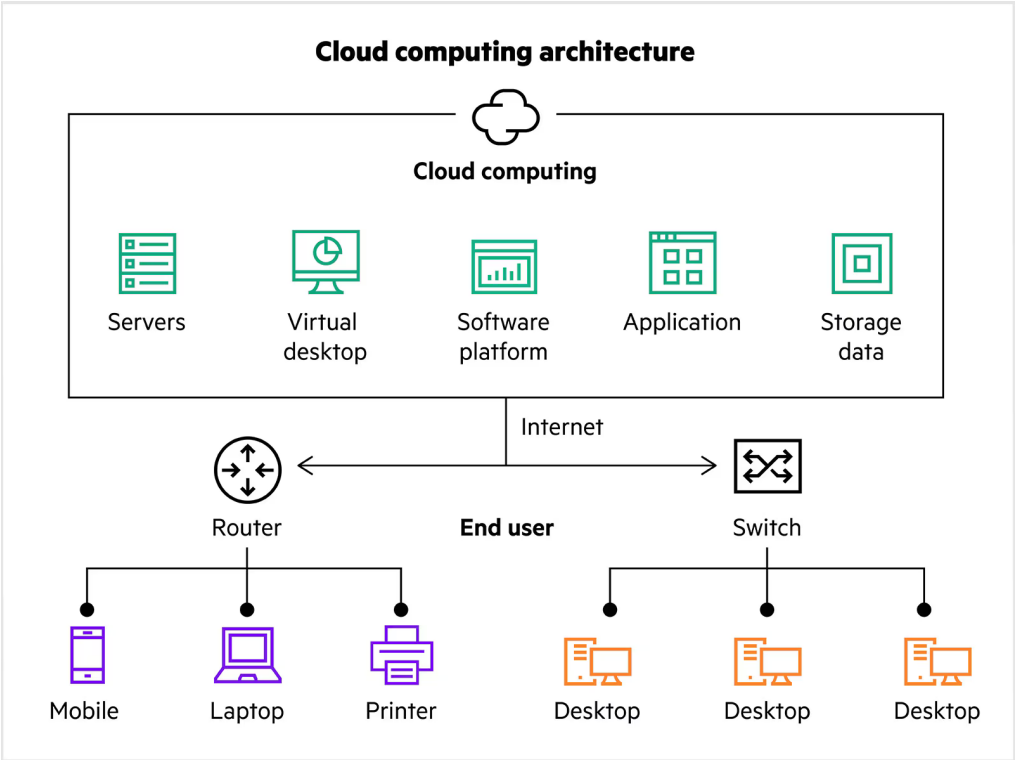
Example

The battery of my laptop might drain losing all the progress of my program/analysis could be lost.

What is cloud computing?

Cloud Computing

Cloud computing is the on-demand availability of computing resources (such as storage and infrastructure), as services over the internet. It eliminates the need for individuals and businesses to self-manage physical resources themselves, and only pay for what they use. (Google Cloud)



How can we use cloud computing?

There are different providers of such services. The most well-known are:

- Amazon Web Services
- Google Cloud
- Microsoft Azure
- Digital Ocean
- ...

AWS: Elastic Compute Cloud (EC2)

- AWS EC2 = Elastic Compute Cloud
- Resizable compute resources in the cloud.
- Minimizes the time to provision a server.
  - Introduce a new server within minimum delay.
  - Scale capacity up very fast.
- Quickly modify the capabilities of the compute instance.
  - Introduce additional computational, memory and storage capabilities.
  - Reduce computational, memory and storage capabilities.
- Shutdown - or completely remove resources.
  - Scale down very fast.
- Pay only for the resources you need.

Typical Use Cases

- Development and Testing Environments
- Hosting of Databases
- Hosting of web services
- Data analytics
- Code repository
- GPU-assisted machine learning
- High performance computing
- Video processing
- Backup and disaster recovery
- ...

EC2 Provisioning Options

- On Demand – Pay for the compute capacity by the hour.
  - No up-front payment or long-term commitment.
  - Short-term, spiky, or unpredictable workloads.
  - Applications development or testing.
- Spot Instances – Acquire spare capacity up to 90% off the on-demand price.
  - When start/end times are flexible.
  - Applications that are only feasible at very low compute prices.
  - Urgent computing needs for large amounts of additional capacity.
- Reserved Instances – Significant discount (up to 75%) compared to On-Demand instance pricing.
  - For applications that have steady state or predictable usage.
  - Long term ($\geq$ 1 year) to reduce their total computing costs.
- Dedicated Hosts – Physical servers dedicated for use use.

EC2 Instance Types

- General Purpose – balance of compute, memory and networking resources.
- Compute Optimized – ideal for compute bound applications that benefit from high performance processors.
- Memory Optimized – deliver fast performance for workloads that process large data sets in memory.
- Accelerated computing – use hardware accelerators, or co-processors, to perform functions, such as floating point number calculations, graphics processing, or data pattern matching, more efficiently than is possible in software running on generic CPUs.
- Storage optimized – for workloads that require high, sequential read and write access to very large data sets on local storage.

EC2 Instance Types & Resources

- CPU – 64-bit Arm, AMD EPYC 7000, Intel Xeon Platinum 8175M, Intel Xeon E5-2676.
  - 1 . . . 192 virtual CPUs – 1 thread = 1 vCPU.
- Memory – 1 . . . 512 GB.
- Network – up to 100 Gbps.
- Storage
  - Amazon Elastic Block Store (EBS) – easy to use, high performance block storage service.
  - 0 . . . 60 TB NVMe SSD – ensure best IOPS (Input/Output operations per second).
- Hardware Accelerators
  - NVIDIA Tesla V100 GPUs, NVIDIA K80 GPUs, NVIDIA T4 Tensor Core GPUs.
  - AWS Inferentia Chips.
  - Xilinx Virtex UltraScale+ VU9P FPGAs

Available OS & Software

- Operating Systems
  - Linux/Unix – Amazon Linux, Debian, Ubuntu, Red Hat, CentOS, SUSE, FreeBSD, Gentoo, Mint, ...
  - Windows – Server 2019, Server 2016, Server 2012.
- Databases – PostgreSQL, MySQL, MongoDB, Neo4J, Oracle Enterprise, Microsoft SQL, ...
- AWS Marketplace – a wide selection of commercial and free software from well-known vendors.

Pricing Examples

- General Purpose
  - t2.micro Linux or Windows – 2 vCPUs + 4 GB – 750 hours free per month, $0.05/h
  - a1.xlarge Linux – 4 64-bit ARM vCPUs + 8 GB – $0.1152/h
  - a1.xlarge Linux – 4 64-bit ARM vCPUs + 8 GB – $0.1152/h
  - m5.24xlarge Linux – 96 Xeon vCPUs + 337 GB – $5.136/h
  - m5.24xlarge Windows – 96 Xeon vCPUs + 337 GB – $9.552/h
- Compute Optimized
  - c5.xlarge Linux – 4 Xeon vCPUs + 8 GB – $0.192/Hour
  - c5.24xlarge Linux – 96 Xeon vCPUs + 192 GB – $4.608/Hour
- Hardware Accelerators
  - p3.2xlarge Linux – 1 NVIDIA Tesla V100 GPUs + 8 Xeon vCPUs + 61 GB – $3.305 per Hour
  - p3dn.24xlarge Linux – 8 NVIDIA Tesla V100 GPUs + 96 Xeon vCPUs + 768 GB – $33.711 per Hour

Amazon Elastic Block Store (EBS)

- Easy to use, high performance block storage service.
- Targeting both throughput and transaction intensive workloads.
  - Can be used for relational and non-relational databases.
  - Enterprise applications.
  - Big data analytics engines.
  - General purpose file systems.
  - Media workflows.
- Highly availability and durability – 99.999%
- Virtually unlimited scale – as little as a single GB of storage, or scale up to petabytes of data.
- Secure – encryption of data at-rest, data in-transit, and all volume backups.

EBS Volume Types – HDD based

- Throughput Optimized HDD (ST1) – ideal for frequently accessed, throughput-intensive workloads.
  - Large datasets and large I/O sizes, such as MapReduce, Kafka, log processing, data warehouse, and ETL workloads.
  - Low cost HDD volume.
  - Volume Size: 500 GB – 16 TB.
  - Max IOPS/Volume: 500
  - Max Throughput/Volume: 500 MB/s
  - Price: $0.045/GB-month
- Low-cost HDD (SC1) – ideal for less frequently accessed workloads with large, cold datasets.
  - Colder data requiring fewer scans per day.
  - Volume Size: 500 GB – 16 TB.
  - Max IOPS/Volume: 250
  - Max Throughput/Volume: 250 MB/s
  - Price: $0.025/GB-month

EBS Volume Types – SSD based

- Provisioned IOPS SSD (IO1) – high performance SSD volume designed for latency-sensitive transactional workloads.
  - I/O-intensive NoSQL & relational databases.
  - Volume Size: 4 GB – 16 TB.
  - Max IOPS/Volume: 64,000
  - Max Throughput/Volume: 1,000 MB/s
  - Price: \$0.125/GB-month + \$0.065/provisioned IOPS
- Default EBS volume type (GP2) – ideal for suitable for a broad range of transactional workloads.
  - Boot volumes, low-latency interactive apps, dev & test.
  - Volume Size: 1 TB – 16 TB.
  - Max IOPS/Volume: 16,000
  - Max Throughput/Volume: 250 MB/s
  - Price: \$0.10/GB-month

What is a Shell?

- One user interface to the operating system (different from graphical interface)
- Functionality:
  - Execute other programs
  - Manage files
  - Manage processes
- A program like any other
- Executed when you "open a Terminal"
- The shell
  - Allows the execution of command scripts
  - Enables alternative methods to carry out complex tasks
  - Provides variables

Shell Interactive Use

- The $ is called the "prompt"
- In the prompt we type the name of the command and press "Enter"
- The prompt allows
  - Command history
  - Command line editing
  - File expansion (tab completion)
  - Command expansion
  - Key bindings
  - Spelling correction
  - Job control

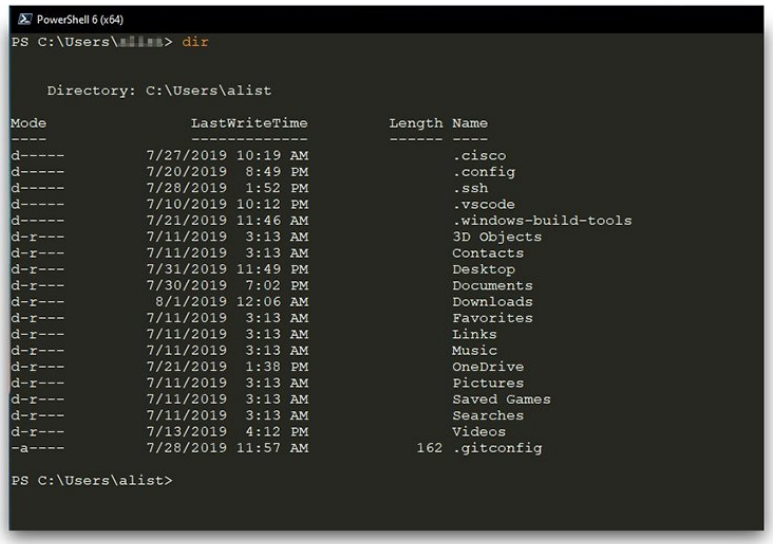
Prompt: The Command Line

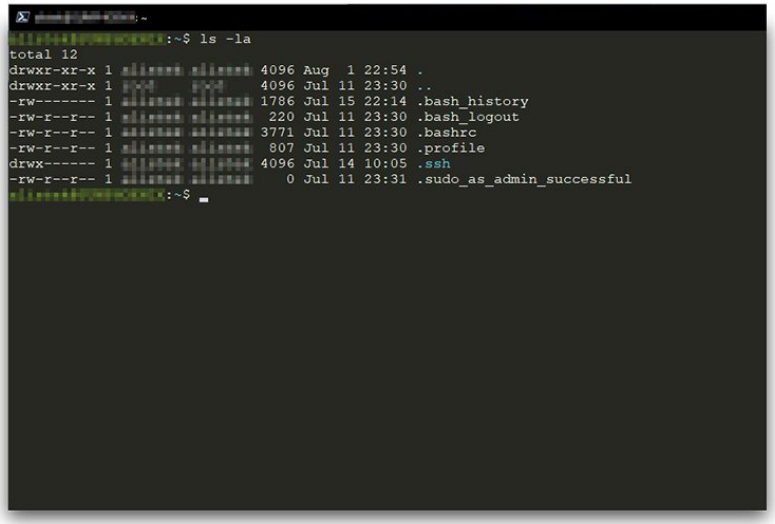
```
# date
Sat Apr 21 16:47:30 GMT 2007
```

Bash shell vs Windows PowerShell

PowerShell

Windows natively supports PowerShell that is different from Bash shell.





Bash shell vs Windows PowerShell

PowerShell

Windows natively supports PowerShell that is different from Bash shell.

You can access Bash shell in Windows with:

- Git Bash
- Tutorial to enable Ubuntu/Linux subsystem on Windows

Error Handling

- If we type a wrong command, an error message appears

Prompt: The Command Line

```
# datee
datee: no such file or directory
```

- The error message states that either the file or the folder (directory) was not found
  - In the prompt all commands are assumed to be connected to a file ...
- The arrow keys $\uparrow$ $\downarrow$ allow to look-up previous commands
- The arrow keys $\leftarrow$ $\rightarrow$ allow to move within the same command line

Terminating Command Execution

- We can interrupt the execution of a command by pressing *ctrl-c*
- We can "freeze" the output of the execution of a command by pressing *ctrl-s*
  - To "un-freeze" the output of a command we use *ctrl-q*
  - Note – only the output is frozen not the actual execution
- To close a terminal we use *ctrl-d*
  - We may need to press multiple times *ctrl-q*
  - All programs currently running will terminate

Manual Pages

- The command *man* allows to access the manual pages
- Manual pages are organized in categories
  1. Commands – *ls*, *cp*, *grep*
  2. System Calls – *fork*, *exit*
  3. Libraries
  4. I/O Files
  5. File Encoding Types
  6. Games
  7. Miscellaneous
  8. Administrator's Commands
  9. Documents
- We can request a page from a specific category
  man [category] [topic]

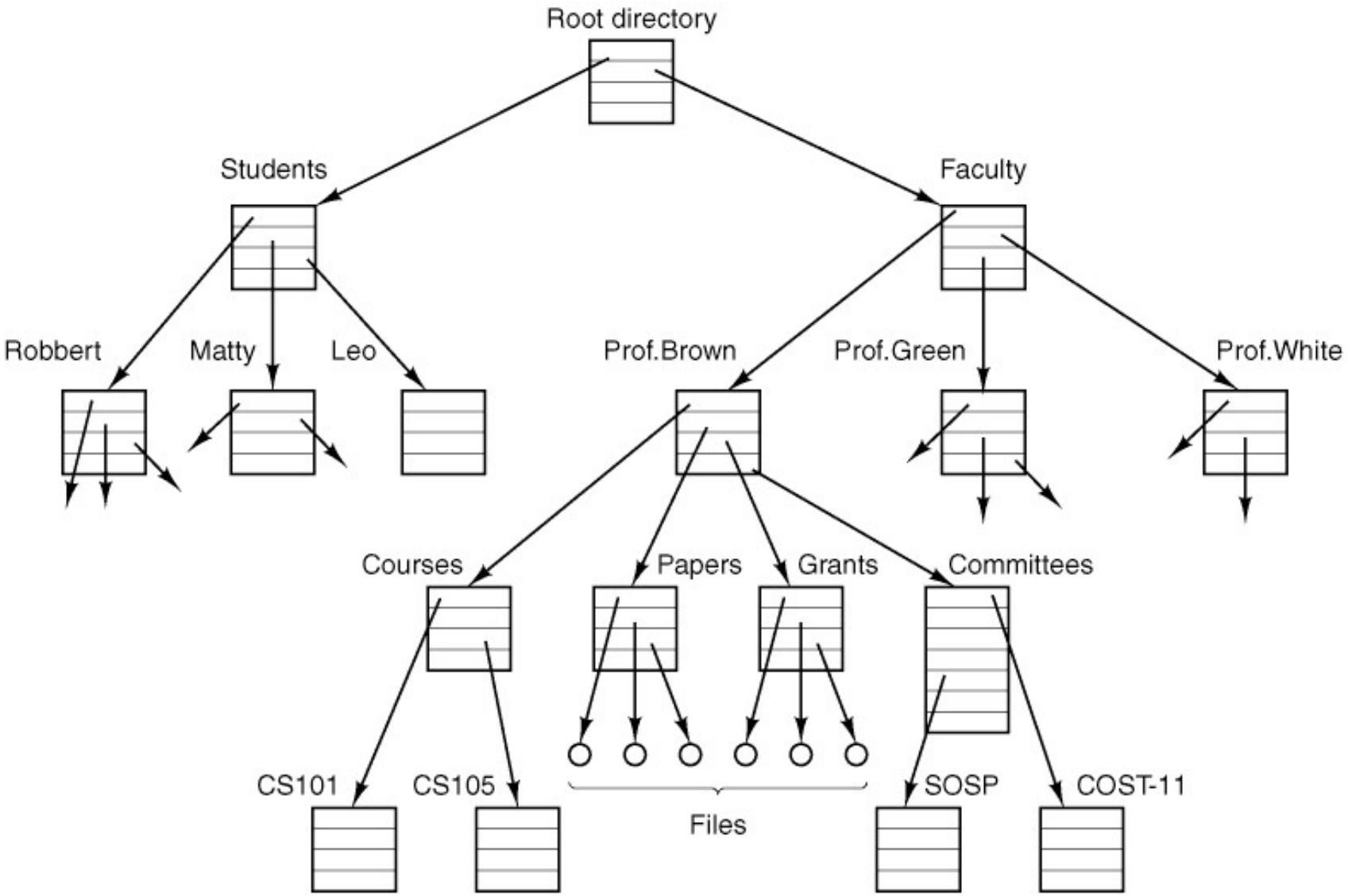
Manual Pages

```
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --archive
              same as -dR --preserve=all

       --attributes-only
              don't copy the file data, just the attributes
```

File System

- All system entities are abstracted as files
  - Folders and files
  - Commands and applications
  - I/O devices
  - Memory
  - Process communication
- The file system is hierarchical
  - Folders and files construct a tree structure
  - The root of the tree is represented using the /
- The actual structure of the tree depends on the distribution of Linux
  - Certain folders and files are standard across all Linux distributions

File System Example



Standard Folders

- `/bin` – Basic commands
- `/etc` – System settings
- `/usr` – Applications and Libraries
- `/usr/bin` – Application commands
- `/usr/local` – Applications installed by the local users
- `/sbin` – Administrator commands
- `/var` – Various system files
- `/tmp` – Temporary files
- `/dev` – Devices
- `/boot` – Files needed to start the system
- `/root` – Administrator's folder

Example of File Metadata

```
# ls -la
lrwxrwxrwx  1 bin    operator   2880 Jun  1  1993 bin
-r--r--r--  1 root   operator    448 Jun  1  1993 boot
drwxr-sr-x  2 root   operator  11264 May 11 17:00 dev
drwxr-sr-x 10 root   operator   2560 Jul  8 02:06 etc
drwxrwxrwx  1 bin    bin           7 Jun  1  1993 home
lrwxrwxrwx  1 root   operator      7 Jun  1  1993 lib
drwxr-sr-x  2 root   operator    512 Jul 23  1992 mnt
drwx------  2 root   operator    512 Sep 26  1993 root
drwxr-sr-x  2 bin    operator    512 Jun  1  1993 sbin
drwxrwxrwx  6 root   operator    732 Jul  8 19:23 tmp
drwxr-xr-x 27 bin    bin        1024 Jun 14  1993 usr
drwxr-sr-x 10 root   operator    512 Jul 23  1992 var
```

Navigating the File System

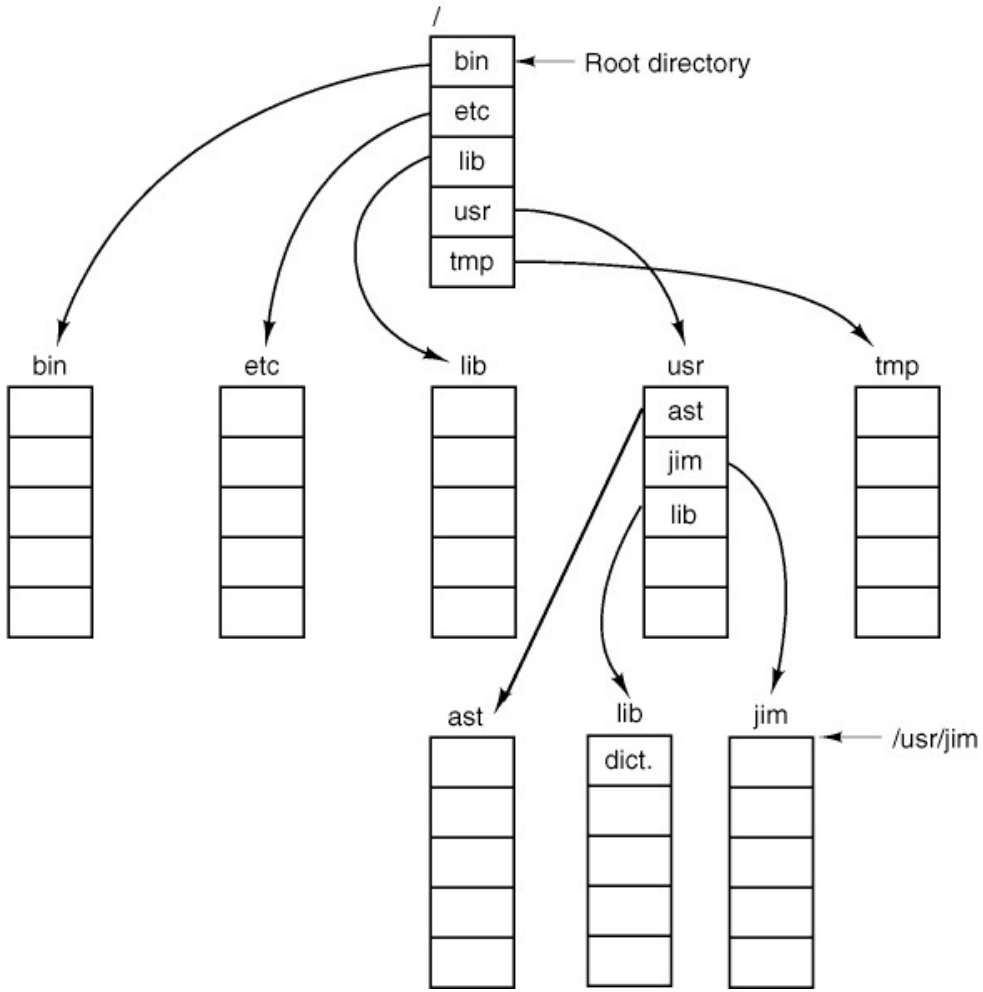
- Each folder contains two "virtual" folders

  ```
  ls -la
  .   ..
  ```

- The single dot represents the same folder

  `./myfile` ⇒ `myfile`

- The two dots represent the "parent" folder in the tree



File System Security

- For each file we have 16 bit to define authorization
  - 12 bit are used by the operator
  - They are split in 4 groups of 3 bit – 1 octal – each
- The first 4 bit cannot be changed
  - They characterize the type of the file (simple file, folder, symbolic link)
  - When we list the contents of a folder the first letter is used to signify:
    - – simple files
    d – folders
    l – symbolic links
- The next 3 bit are known as the s-bits and t-bit
- The last three groups are used to define the access writes for read 'r', write 'w' and execute 'x'
  - For the file owner, users of the same group, and all other users.

File System Permissions Examples

```
Type  Owner  Group  Anyone
  d   rwx    r-x    ---
```

- Folder
- The owner has full access
- All users that belong to the group defined by the file can read and execute the file – but not modify the contents
- All other users cannot access the file or execute it
- To access a folder we use the command *cd* given that we have permission to execute 'x'

Changing the File Permissions

Examples of File Permissions

```
Binary   Octal    Text
 001       1        x
 010       2        w
 100       4        r
 110       6       rw-
 101       5       r-x
  -       644    rw-r--r--
```

- The command *chmod* allows to modify the permissions
- There are 2 way to define the new permissions
  1. Defining the 3 Octal – e.g., *644*
  2. By using text – e.g., *a+r*

Some Examples of *chmod*

```
make read/write-able for everyone
# chmod a+w myfile

add the 'execute' flag for directory
# chmod u+x mydir/

open all files for everyone
# chmod 755 *

make file readonly for group
# chmod g-w myfile

descend recursively into directory opening all files
# chmod -R a+r mydir/
```

Changing the Owner and Group of a File

- The command *chown* allows to change the owner of a file
- The command *chgrp* allows to change the group of a file

```
give ownership to ichatz
# chown ichatz myfile

set group to students
# chgrp students mydir/

give ownership to pcs and group to students
# chgrp pcs:students myfile mydir/

descend recursively into directory opening all files
# chown -R ichatz mydir/
```

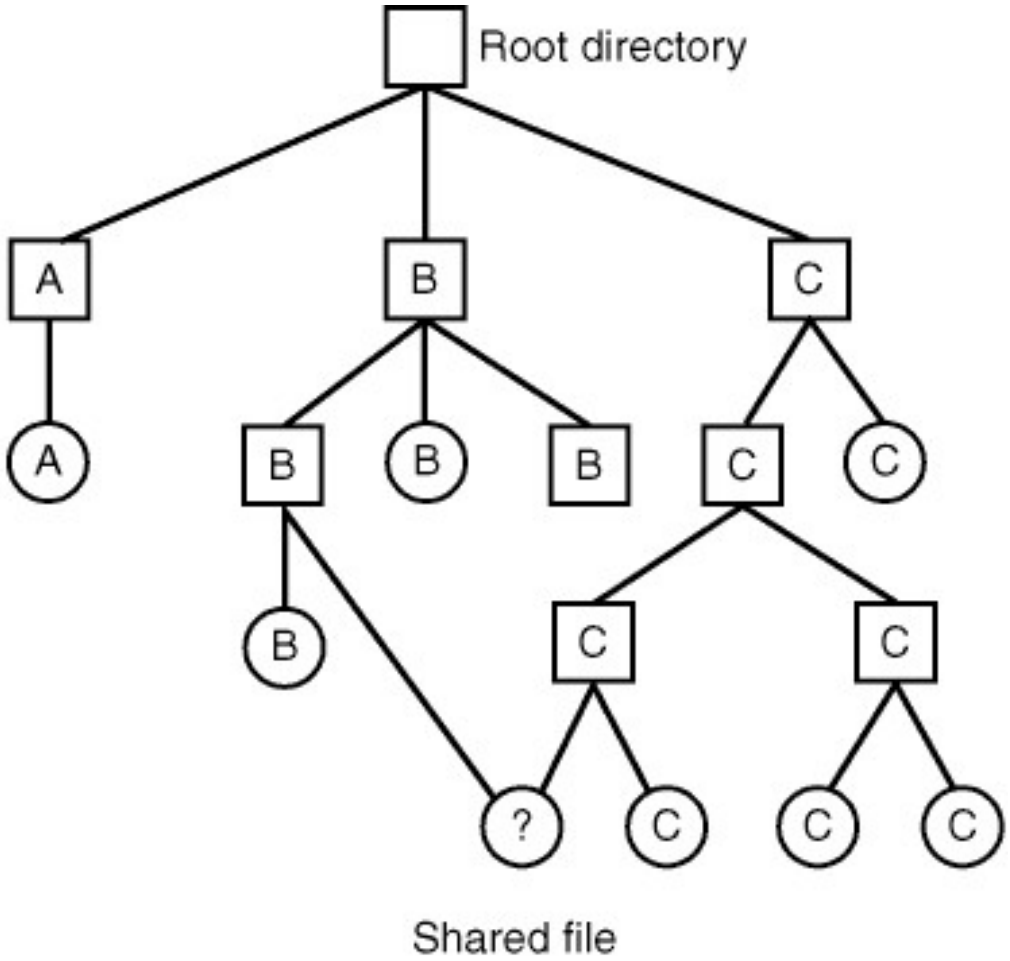
Symbolic Links

- The file system enables to create symbolic links
- Two types are provided
  - Symbolic link
  - Hard link
- The contents and metadata of the original file are used for all operations

```
create a symbolic link to a directory
# ln -s /var/log ./log
# ls -lg
lrwxrwxrwx 1 operator    8 Apr 25  log -> /var/log
```

- The contents and metadata of the original file are used for all operations
  - Except for deletion.

Examples of Symbolic Links



Access Dates

- For each file the system keeps track of
  - Date of last usage/access
  - Date of last change

```
check last usage time
# ls -lu
drwxrwxrwx  1 bin     bin             7 Apr 25   1993 home
lrwxrwxrwx  1 root    operator        7 Apr 25   1993 lib
drwx------  2 root    operator      512 Mar 30   1993 root

check last change time
# ls -lc
drwxrwxrwx  1 bin     bin             7 Apr 25   1993 home
lrwxrwxrwx  1 root    operator        7 Oct 27   1993 lib
drwx------  2 root    operator      512 Oct 27   1993 root
```