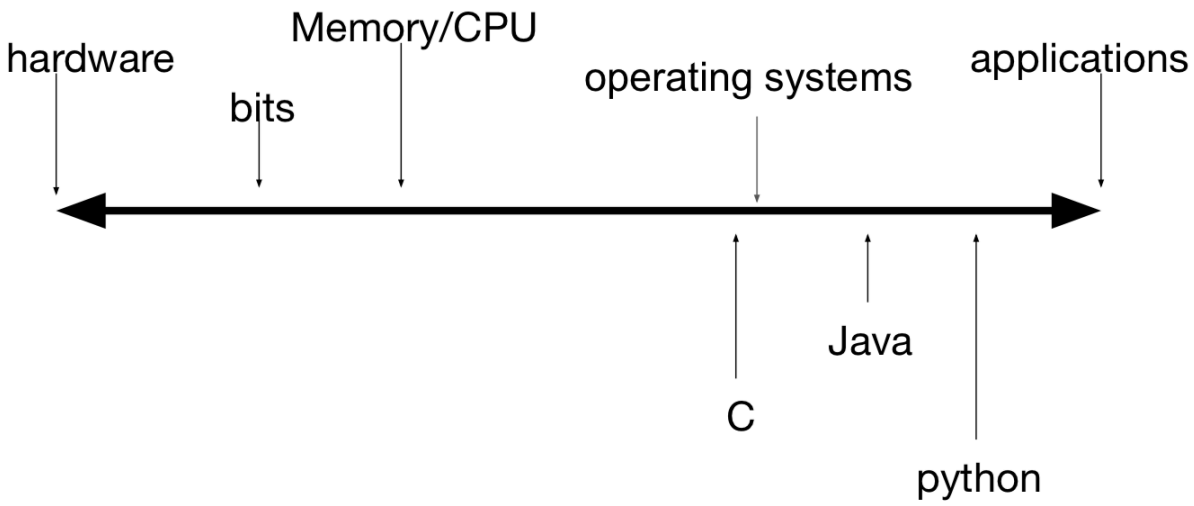
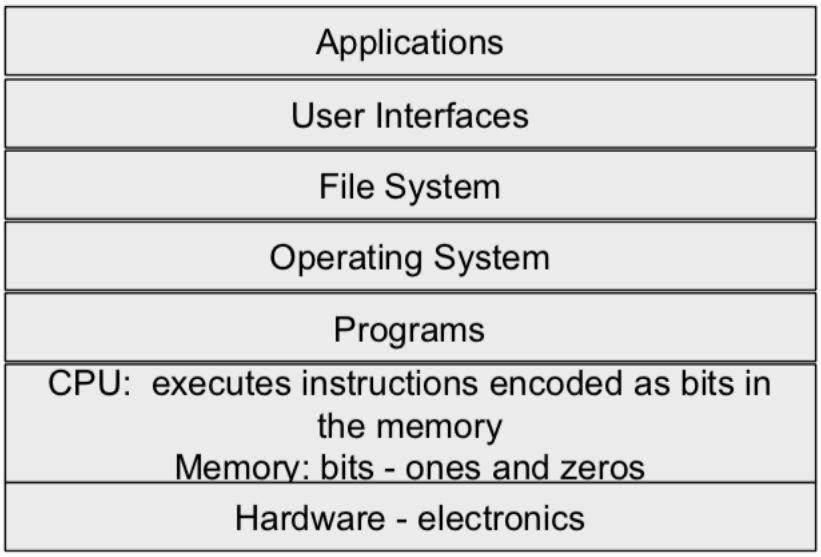
INFO1112 Notes



INFO1112 will cover content from 'bits' to 'applications

INFO1112 Week 1 Lecture – Introduction

- The computer system that you interact with is built in a series of layers where each layer builds on the one below.



Bits and Bytes

Why 0/1?

- Easy to build a circuit that is on or off to represent physically the mathematical entities 0/1.
- A simple switch is off (0) or on (1).
- A transistor is a simple switch: in modern computers, the binary digits (0/1 or bits) are represented by the state of transistor switches → there are many ways to represent 0/1 physically.
- We can represent (almost) any piece of information as a sequence of 0's and 1's.

Memory

- The memory of a computer is made up of bits → but not just a simple stream of bits.
- Memory is divided into 8-bit pieces called bytes → These pieces are each given a number from 0 to the number of bytes in the memory.
- Each of these numbers is a memory address.

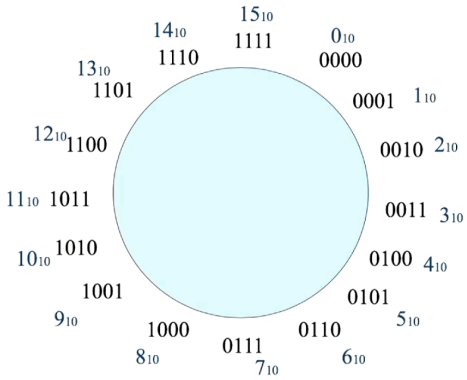
Numbers

- Computer systems use the binary number system, in which these patterns can be stored as bytes → each byte is only 8 bits long – this puts a limit on the range of values that can be represented.
- Numbers in the computer are not the same as integers in mathematics.

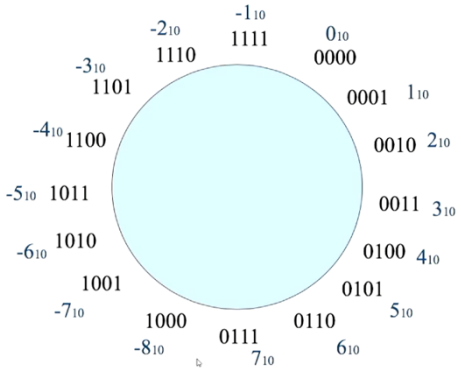
About Finiteness

- Computers are inherently finite, number representations are finite, memory is always finite → memory 'leaks' matter (you forget to deallocate memory).

Number Circle for 4 bit numbers - Unsigned Interpretation



Number Circle for 4 bit numbers - 2s Complement Interpretation



- Not necessarily true with 2's complement numbers → overflow.

Using our 4 bit 2's complement numbers:

If a > 0 and b > 0, is a + b > 0?
If a < 0 and b < 0, is a + b < 0?

Is x squared >= 0 for all int x?

Real Numbers

- Potentially infinite precision as well as infinite size.
- Need to impose limits in a real machine.
- Many schemes for representing real numbers.
- In fixed point representation, how do we represent 1/3 in binary? → you can't, although you can reach an increasing level of accuracy.

Representations

- Characters are often represented in a single byte → but this restrict the range of characters, so multi-byte representations such as Unicode as available.
- Strings are sequence of bytes.
- Integers are often 4 bytes (for single precision).
- Data structures are combinations of primitive types.

Characters and Character Codes

ASCII

- A = 1000001 = 65 (decimal)
- B = 1000010 = 66 etc.
- Only 0-127 characters can be represented in ASCII, which is fine for English but could be a waste of space with other languages since there are so many character codes in use.
- Unicode is the newest universal standard.
    - → 32 bits/character
    - → Big waste of space for English
    - → Solution: encode the encoding → most common characters end up as 8 bits.

Instructions or Code

- As well as data, we also store program instructions or program code in the memory.
- A program written in a language like C is translated into machine instructions that are loaded into the memory and executed by the central processing unit (CPU).
- Programs can create machine instructions, store them in the memory and then execute them.

CPU

- The CPU reads instructions from memory and executes them.
- The instructions are represented in bits and stored in a sequence of bytes e.g. 01100100 10001100 → this might mean in some hypothetical machine "add the integer in address 4 to the integer in address 8 and store the address to address 12".
- The idea to store instructions in the same memory as values was the major breakthrough in the development of computers; "stored program computers".

Instructions

- This might be broken down as:
    - 0110 code for 'add' instruction
    - 0100 address 4

1000 address 8

1100 address 12

Registers

- Real machines usually have special storage in the CPU called registers.
- These are usually the size of an integer and are used for intermediate results e.g.

  Load R1, 4 //load contents of address 4 into R1

  Add R1, 8 //add coentents of address 8 to R1

  Store R1, 12 //store R1 into address 12

Assembly Language

- Instructions like: Load R1, 4 //load contents of address 4 into R1, are a human readable way of writing the machine instruction called assembly language.
- Assembly language is translated (by a program called an assembler) into the binary presentation.

Index Registers

- As well as general purpose registers for intermediate results, the CPU might have special registers that contain addresses or offsets used when fetching a value from memory.
- This makes moving through strings and lists of things easier.

Intel 386 assembler

```
.text
.global _start
_start:

mov $4, %eax /* write system call */
mov $1, %ebx /* stdout */
mov $msg, %ecx
mov $msgend-msg, %edx
int $0x80

mov $1, %eax /* _exit system call */
mov $0, %ebx /* EXIT_SUCCESS */
int $0x80

.data
msg: .ascii "Hello, world\n"
msgend:
```

Operating Systems

- A “simple” system consists of memory and CPU and input/output devices.
- Program instructions are stored in memory and executed by the CPU and may read/write data from external devices such as the keyboard and the screen.
- Systems such as Windows or Apple Macintosh provide far more:

  → sophisticated graphical user interface

  → other I/O devices such as a mouse

→ a file system

→ network connections

→ concurrency (running more than one program at a time)

→ security

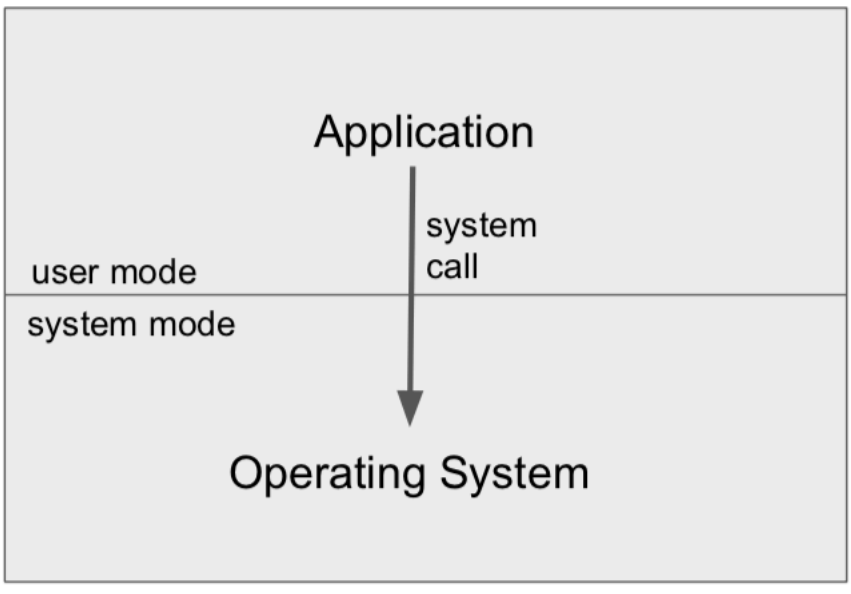
- These services are generally provided by an Operating System.

Unix Operating System

- Invented long ago (1969) in a place called Bell Laboratories by Dennis Ritchie and Ken Thompson. Many others contributed.
- Released to Universities in source code form in 1975.
- Many versions of the original and many clones (e.g. Linus) have been produced since then. All are recognisable Unix based systems.
- Linux is a Unix clone developed originally by Linus Torvalds, a Finnish programmer → now most likely the most widely used system.

More on Operating Systems

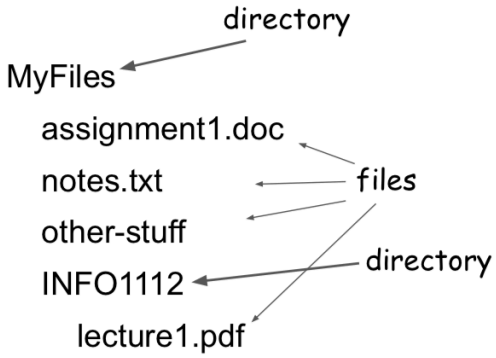
- An operating system is a special program that is privileged (can access and control all of the hardware) and provides all of the services. For some services such as the graphical user interface, the operating system is supplemented by application programs.
- To enforce the distinction between the operating system and user programs, the CPU usually has (at least) two modes of operation: system and user.
- To switch from user mode to system mode (e.g. when an application program requests a service such as reading a file), a special CPU instruction is executed – system call. This will cause execution to continue in the operating system program in the system mode.



The File System

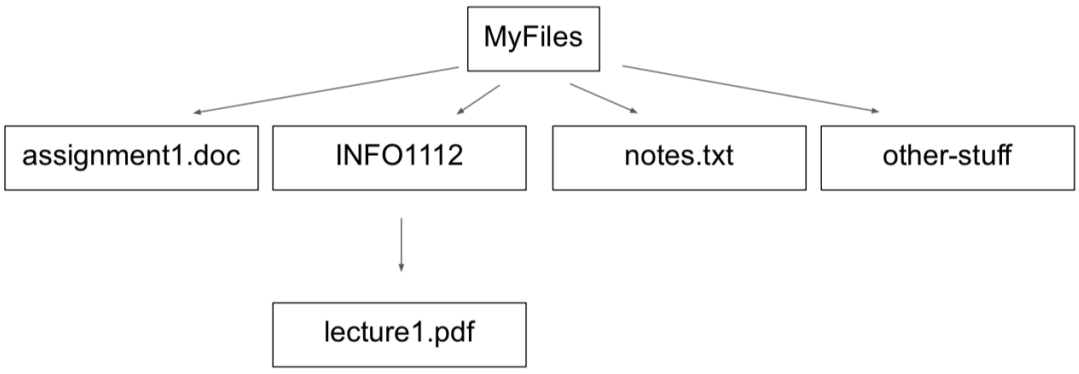
- To store "persistent" data that doesn't disappear when we turn off the computer, we need storage other than main memory. This can be on a separate device such as a disk drive connected to your computer or on another computer connected to the network.
- "Disk" drives are implemented with various technologies, but all provide persistent storage.
- The raw storage provided by these mass storage devices is not so convenient to use, so in nearly all cases we structure the storage into *files* and *directories*, called the file system.

The file system (in Unix) consists of a set of directories (or folders) that can contain files or sub-directories.



48

Another way to look at this is:



This is called a ***tree*** data structure

49

- The words and structure of the file system are called the name space.
- To specify a given file or directory in the Unix file system name space we write the path name of the file or directory.
- E.g. to specify the assignment1.doc file we might write: /MyFiles/assignment1.doc → tell us that MyFiles is a directory at the root of the file system and that assignment1.doc is contained within.

User Interface – The Command Line

- Originally, systems only had simple character input and display – no windows, no mouse. To use the system you typed characters that were interpreted by a special program called a command interpreter or in Unix, the shell.
- The shell analysed the character string and ran the program you specified.
- The important information is that the shell is a program that anyone could write.

The Unix Shell

- Since it is relatively easy to write a shell command line interpreter, many have been developed over the years.
- The original is simply called “sh” but in this course we will use one called “bash”.
- The name comes from “Bourne Again Shell” since bash is a clone of a shell written at Bell Labs by Steve Bourne.
- There are many others e.g. csh, ksh etc.

Here are some the commands for working with files that you saw in first semester:

`cd`	Change directory
`mkdir`	Make directories
`ls`	List directory contents
`rmdir`	Remove directory
`rm`	Remove
`mv`	Move (also used to rename files)
`cp`	Copy
`pwd`	Print Working Directory

And here are some special names for elements of the file system:

`..`	Parent Directory	`cd ..`
`.`	Current Directory	`./program`
`-`	Previous directory	`cd -`
`~`	Home directory	`cd`
`/`	Root directory	`cd /`

INFO1112 Week 2 Lecture - More Shell, Processes and Namespace

More Advanced Shell

- The shell commands you learnt so far allow you to move around the file system, look at files, rename and delete files. But the shell, in combination with simple application programs, is far more.
- The shell is a sophisticated programming language with control structures and variables just like Python.
- When combined with simple text processing applications/command, it can do very powerful tasks.
- You should always consider using a shell program rather than write a python program for any simple text processing task.

Shell Variables

- The shell has variables like conventional programming languages. To give a variable a value, we use an assignment statement.

```
$ NAME=bob
$
```

Note: no spaces around the = sign!

- We retrieve the value of a variable by putting $ at the beginning:

```
$ echo $NAME
bob
$
```

echo is a command that prints its arguments

- Shell variables can only hold strings of characters but this still makes them very useful. For example, they could contain a file name:

```
$ FILE=fred
$ cat $FILE
some text in file called fred
$
```

- By using an application program, we can do arithmetic:

```
$ A=1
$ expr $A + 1
2
$
```

- In order to get the result back into A if we want A=A+1, we can get the standard output of a command and use it as an argument.
- Enclosing a command in backquotes (`) means "replace the command with its output".

```
$ A=`expr $A + 1`
$ echo $A
2
$
```

- The below will evaluate and then print the arguments of 'echo' which is 'echo hello' → printing 'hello'.

```
$ `echo echo hello`
```

Shell Control Structures

- When combined with shell variables and pipes, the shell becomes a powerful programming language.
- We can store shell commands in a file and run them as if they were another command → "shell scripts".

Here is a very simple shell script:

```
#!/bin/bash
echo $1 is your name
```

If we store those lines in a file called "myname", we can run it using:

```
$ bash myname Bob
Bob is your name
```

specifies what command to use to interpret the script

```
#!/bin/bash
echo $1 is your name
```

```
#!/bin/bash
echo $1 is your name
```

special variable that specifies the first argument for the script

Pipes

- You can redirect the output of a command so that it goes into a file instead of the screen:

```
$ echo hello >myfile

$ cat myfile

hello
```

- You can also take the output of one command and send it to the input of another using a pipe:

```
$ cat myfile | wc

   1      1      6
```

- The | will send the output of one command as input of another, with 'wc' meaning 'word count'.
- You can use pipes to combine commands in many useful ways.
- Unix has a lot of simple commands that process text:

wc	line/word/character count
cut	remove a portion of each line (eg cut out columns)
paste	paste several files together vertically
sed	stream editor
tr	transliterate one class of character to another
sort	sort a file on one or more fields
grep	search for matching lines
uniq	remove duplicate lines
fmt	format the text

Processes and the Unix Kernel

- In normal operation a Unix system has many programs stored in memory at the same time. If it is a single CPU system only one program can run at a time. The Unix kernel is responsible for switching between programs.
- This depends on a feature of CPUs called an interrupt. This is where a currently running program is interrupted, all of its "state" (memory, cpu registers etc) is saved, and the kernel is entered. Later the kernel might restore the state and restart the program. The program will continue as if nothing had happened.
- Programs in memory have a standard layout which we will discuss later.
- The combination of the memory occupied by the program and all of its state (which also includes details of what files it currently has open), is called a PROCESS.
- A Unix system will usually have a large number of processes in the memory at any one time. On a single CPU machine, only one will be executing (or possibly none if the kernel is running).
- A typical small system will have more than 150 processes!

Processes

- We can find out what processes are running with the ps command.
- ps allows you to get information on all processes or a subset. Below, we only see processes belonging to the logged in user (User Ids will be discussed soon).

```
$ ps
  PID TTY          TIME CMD
 6438 pts/0    00:00:00 bash
17380 pts/0    00:00:00 ps
```

- The ps command can display more information about running processes.

```
$ ps al|more
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY      TIME COMMAND
0  1003 19208 19207  20   0  21208  4964 wait   Ss   pts/0      0:00 -bash
0  1003 24486 19208  20   0  27640  1452 -      R+   pts/0    0:00 ps al
0  1003 24487 19208  20   0   8236   748 pipe_w S+   pts/0      0:00 more
```

owner user id
process id
parent process id
priority
command

Documentation – the Man Command

- All Unix systems have documentation about commands that can be accessed with the “man” command.

```
$ man ps
PS(1)            User Commands                          PS(1)
NAME
    ps - report a snapshot of the current processes.
SYNOPSIS
    ps [options]
DESCRIPTION
    ps displays information about a selection of the active processes.  …...
```

User IDs

- When a process is running it has an associated User Identification number or UID.
- This is used in the file system to indicate the owner of a file or directory.
- We map the UID onto a name using the password file stored in /etc/passwd.
- Each entry is a line of text. In this password file below, 1003 is the UID, /home/info1112 is the home directory and /bin/bash is the shell program for this user.

```
info1112:x:1003:1003:Bob Kummerfeld,,,:/home/info1112:/bin/bash
```

File Ownership

- The owner of a file is specified by a UID. The ls command looks up the password file and shows the name associated with the UID.

```
$ ls -l
total 4
-rw-rw-r-- 1 info1112 info1112 6 Jul 26 12:46 myfile
```

owner

- The info1112 field is the 'group owner' of the file.

Groups

- As well as User IDs, unix has 'Group Ids' or gid.
- The /etc/group file contains the mapping of gid onto group name.
- When an account is created, it also gets a group of the same name.

```
info1112:x:1003:
```

- Files have an owner but also a group. Permission (read, write, execute/search) are given for the owner, for the group and for anyone else.
- The /etc/group entry can contain multiple user ids for a group.

```
adm:x:4:syslog,info1112
```

that entry shows that "syslog" and "info1112" are members of group "adm".

Superuser ID

- The uid zero is very special. A process running with owner uid 0 is privileged and can access any file and run any program. This means it can access any device (since devices are files in the namespace). The user with uid 0 is called the superuser.
- System parameters are stored in files owned by uid 0 so the superuser can reconfigure the system.
- The superuser is given the user name "root" since the first process that is executed by the system is the root of a tree of processes.
- The superuser can also kill any running process.
- The superuser is all powerful and therefore very dangerous - beware!

More Shell Commands

- To find out the UID of the running shell we can use the command:

```
$ id
uid=1003(info1112) gid=1003(info1112) groups=1003(info1112)
```

- This shows the user id, group id and what groups the user is a member of.
- When a program is started by the shell (e.g. the id command), it is run with the uid of the shell. This means that the commands you run can access files owned by you or you have group permission for.

File Access

- Files have a several associated attributes: name, owner, group, access permissions, position in the name space, location in mass storage. These attributes are also called the metadata for the file.
- A program (or process) can access a file using the open system call. Your program will be suspended briefly while the operating system kernel takes the path name you specify, finds it in the file system, and sets up a data structure containing the metadata. The data structure also shows where your program is up to when reading the file sequentially.
- In a running process, this data structure is referred to by a file descriptor which is a small integer.

Open Files

- When a program is started, the process is automatically given three open files and associated file descriptors:

0	standard input
1	standard output
2	standard error output

- If you are running a program from the command line, these correspond to the keyboard and the screen.

Open Files at the Shell Level

- When you run a command from the shell, the standard input/output/error files can be redirected using the following notation, and more generally, N> will redirect output for file descriptor N.

<	for standard input file
>	for standard output file
2>	for standard error file

Example: `echo Hello >myfile`
this will write the string "Hello" to standard output, but standard output is redirected to the file 'myfile'

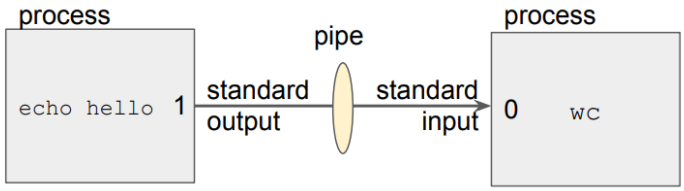
Pipes

- One of the great inventions in Unix was the ability to create a special "file" called a pipe at run time that had the property that whatever you wrote to it was available to read (for a modest amount of data) without it being stored on mass storage.
- If this "pipe" was shared by two processes, one process could write data into it and the other process would receive the data when reading.

- The shell has a special syntax that uses the character '|' to create a pipe between two programs. For example:

Pipes and the Shell

- When a shell runs the command line "echo hello | wc" it creates a pipe, then starts "echo hello" with standard output replaced by the pipe file descriptor, then starts "wc" with standard input replaced by the pipe file descriptor.
- In this way two processes can be made to send data from output of one to input of the other.



Devices in the Namespace

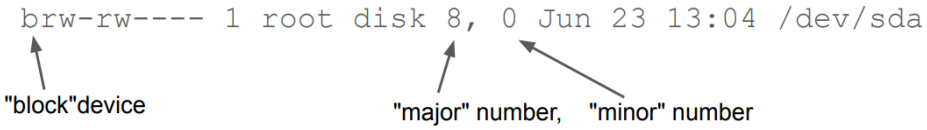
- Another invention of Unix was to make access to devices the same as any other file: it has a name in the namespace and the standard open/close/read/write/delete system calls do roughly the same thing for devices as they do for normal files.
- For example, on a typical Linux system /dev/sda is a pathname that refers to a disk drive. By opening and reading from that file you can access the whole device starting at the beginning.

```
$ ls -l /dev/sda

brw-rw---- 1 root disk 8, 0 Jun 23 13:04 /dev/sda
```

Device

- In Unix all devices are accessed this way. Other systems (eg Windows) have special names (eg C:) for devices.
- When a device file is opened, the operating system looks at the file metadata and sees that this is not an ordinary file and passes it to the appropriate device driver. This is a part of the operating system that manages the particular type of device.
- For example, a disk drive or a network interface. The device type can be seen in an "ls -l":



Other Special Files

- As well as device files, the namespace can specify pseudo files that, when opened and read, return information about the system.

For example:

```
$ cat /proc/uptime
3713661.60 29657491.61
```

This shows the total up time and idle time for the system in seconds.

Files in /proc are not device files but are served by a special file system driver.

INFO1112 Week 3 Lecture – Processes and the File System

From Programs to Processes

- A program is stored on the persistent storage (e.g. disk) in a file.
- When a program is started, the operating system reads the file and generates a memory image that contains code and data → while doing that it will also read and incorporate any functions you use from the library.
- The image is read into some free memory and the associated metadata structures are created (eg standard input/out/error file descriptors etc).
- Lastly the operating system starts the process executing.

System Calls for Processes

- Unix has a number of system calls that manage processes.
- Remember that a system call is like a function call to the operating system, except the privilege level switches from the user to the superuser and the execution continues in the operating system kernel code.
- The main system calls for managing processes are:

`fork`

`exec` (has many variations)

`wait`

The 'os' System Call

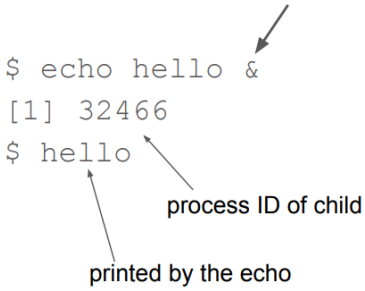
- fork creates a new process (the child) where the memory image and the metadata are an exact copy of the process that called fork (the parent), except for:
    - → the child process has a new process ID
    - → the child process has the parent process ID of the process that called fork
    - → various values of resource metadata (e.g. CPU time used) are reset
- Note that this means the child shares the same files initially.
- After the fork call, both the child and the parent continue execution as if the fork function had returned, except that in the child fork returns 0 and, in the parent, fork returns the process ID of the child.

The 'wait' System Call

- wait suspends execution of the process that calls the wait function until one of its child processes exits.
- wait returns the process ID of the child that has exited and the exit status.
- If a child process terminates and the parent is not 'waiting' for it, then it enters the 'zombie' state until the parent process waits for it.

Starting a Parallel Process with the Shell

- Normally, when you type a command to the shell, it is executed, and the shell waits for it to finish.
- The shell has a syntax for running a process in parallel.



Starting a Python Process

- When a Python process is started from the shell, the arguments are passed in a list available in the 'sys' module → $0 is the command and so on.

```
#!/usr/bin/python

import sys

print ("first argument is ", sys.argv[1])
```

Starting a Process from Python

- Python has an interface to the system calls fork/exec/wait so we can create new processes within python, start another program, and wait for it to terminate.
- The 'os' module contains methods (functions) for many of the Unix system calls.

```
os.fork()
os.execle(path, arg0, arg1, …env)
```

shell environment variables passed to process

arguments passed to process

path to file containing the program

parent process

```
pid = os.fork()
if pid == 0:
     print ("\tHi! I'm the child process.")
     print ("\tsleeping for 5 seconds....")
     time.sleep(5)
     print ("\t... exit with status 99.")
     sys.exit(99)
elif pid == -1:
     print ("yikes! fork failed!")
     sys.exit(1)
else:
     print ("I'm the parent process")
     print ("waiting for child with PID", pid)
     wval = os.wait()
     print ("wait over! process ID was ")
     print (wval[0], "exit status",wval[1]>>8)
```

child process

```
pid = os.fork()
if pid == 0:
     print ("\tHi! I'm the child process.")
     print ("\tsleeping for 5 seconds....")
     time.sleep(5)
     print ("\t... exit with status 99.")
     sys.exit(99)
elif pid == -1:
     print ("yikes! fork failed!")
     sys.exit(1)
else:
     print ("I'm the parent process")
     print ("waiting for child with PID", pid)
     wval = os.wait()
     print ("wait over! process ID was ")
     print (wval[0], "exit status",wval[1]>>8)
```

The ‘exec’ System Call

- After the fork system call, we have a copy of the parent process running as the child, which is ok, but it is more useful to be able to run another program as the child → we can do this with the exec system call.
- Exec comes in several versions that all specify what program to execute (pathname etc) while some specify program arguments, and some specify the shell environment variables.
- Exec is a simple transfer of control. The calling process is completely replaced by the new program.

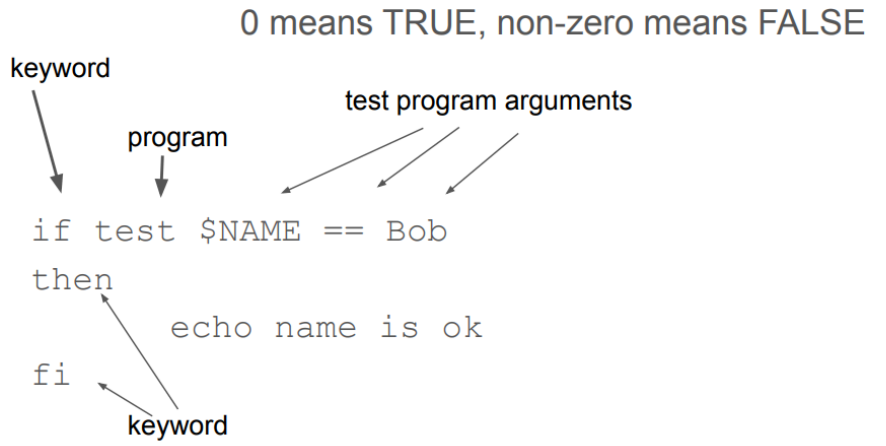
```
pid = os.fork()
if pid == 0:
    print ("\tHi! I'm the child process.")
    print ("\tI'm going to sleep for 5 seconds....")
    time.sleep(5)
    print ("""\t... and now start the command "echo 'This is the child'""")
    os.execl("/bin/echo", "echo", "This is the child")
    print ("oops! It looks like exec failed")
    sys.exit(1)
elif pid == -1:
    print ("yikes! fork failed!")
    sys.exit(1)
else:
    print ("I'm the parent process and I'm waiting for the child with process
ID ", pid)
    wval = os.wait()
    print ("wait over! process ID: ", wval[0], " exit status:", wval[1]>>8)
print ("finished.")
```

Stopping a Process

- Processes can exit voluntarily, or they can be terminated by another process using the kill system call.
- Kill operates by sending a signal to the process (more about signals in the next lecture) with the default signal being TERM to terminate.
- Signals can only be sent to another process if both the sender process and the receiver process have the same UID or if the sender is the superuser. There is a kill command that can be used from the shell.

Shell – Shell If Statement

- The shell has an 'if' statement like conventional programming languages → the statement is testing the return value of a program.



Note: The character "`[`" is a synonym for "`test`". The character "`]`" is ignored by "`test`"

General form is: `if` program+args `then` program+args [`else` program+args] `fi`

File System

- Unix has a tree structured namespace that allow us to access files stored on mass storage, devices and system status details.
- The file system is a way of implementing access to objects in this namespace. The system status is provided by a file system module in the kernel.
- Most conventional file systems manage mass storage to store data files.

File Systems on Mass Storage

- There are many ways to lay out the file data on mass storage → Both the file data and the file metadata (name, permissions etc) must be stored so the file is "persistent".
- This means there are many implementations of file systems, optimised for different features such as speed, reliability/redundancy etc.
- For example: FAT16, FAT32, exFAT in the Windows operating system, ext2, ext4, BtrFS, ZFS etc for the Linux operating system.
- Linux distributions have a default file system.
- The important point is that all the Unix file systems provide the same interface.

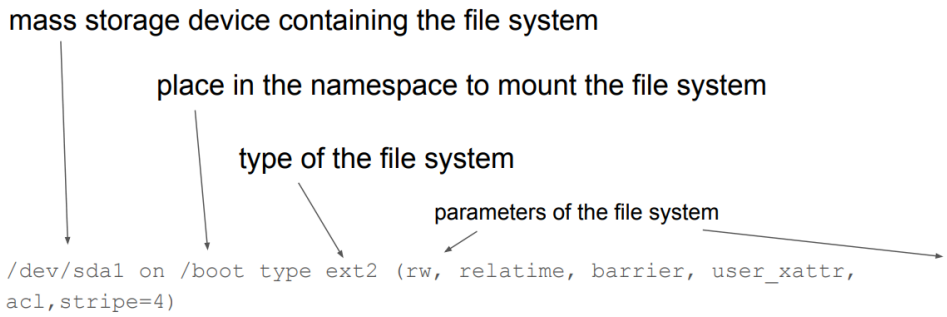
File Systems Interface

- A file system module for the operating system provides a standard interface to the namespace (creat/delete/open/close/read/write….) and as long as they implement that they can provide any type of data in response.
- If you are accessing a normal file on mass storage, this usually means the data from the file.
- However, a file system module can return other information and that's how the system information is provided via a file system "mounted" on /proc.

File System Mounting

- Unix has a system call that grafts a file system into the namespace tree at some point.
- There is also a mount command that runs the mount system call so you can mount new file systems from the command line.
- Typing mount by itself will give you a list of mounted file systems → this can be long since many services are implemented this way.

```
$ mount
/dev/sda1 on /boot type ext2 (rw, relatime, barrier, user_xattr,
acl,stripe=4)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```



When you open a file with a pathname in the namespace (eg /home/info1112/SomeFiles) the operating system traverses the path, works out which file system is responsible and hands the path to that file system to handle the open call.

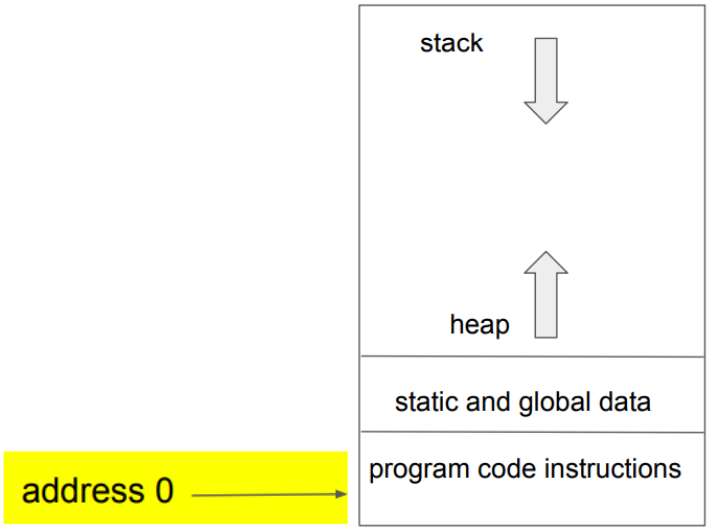
User File Systems

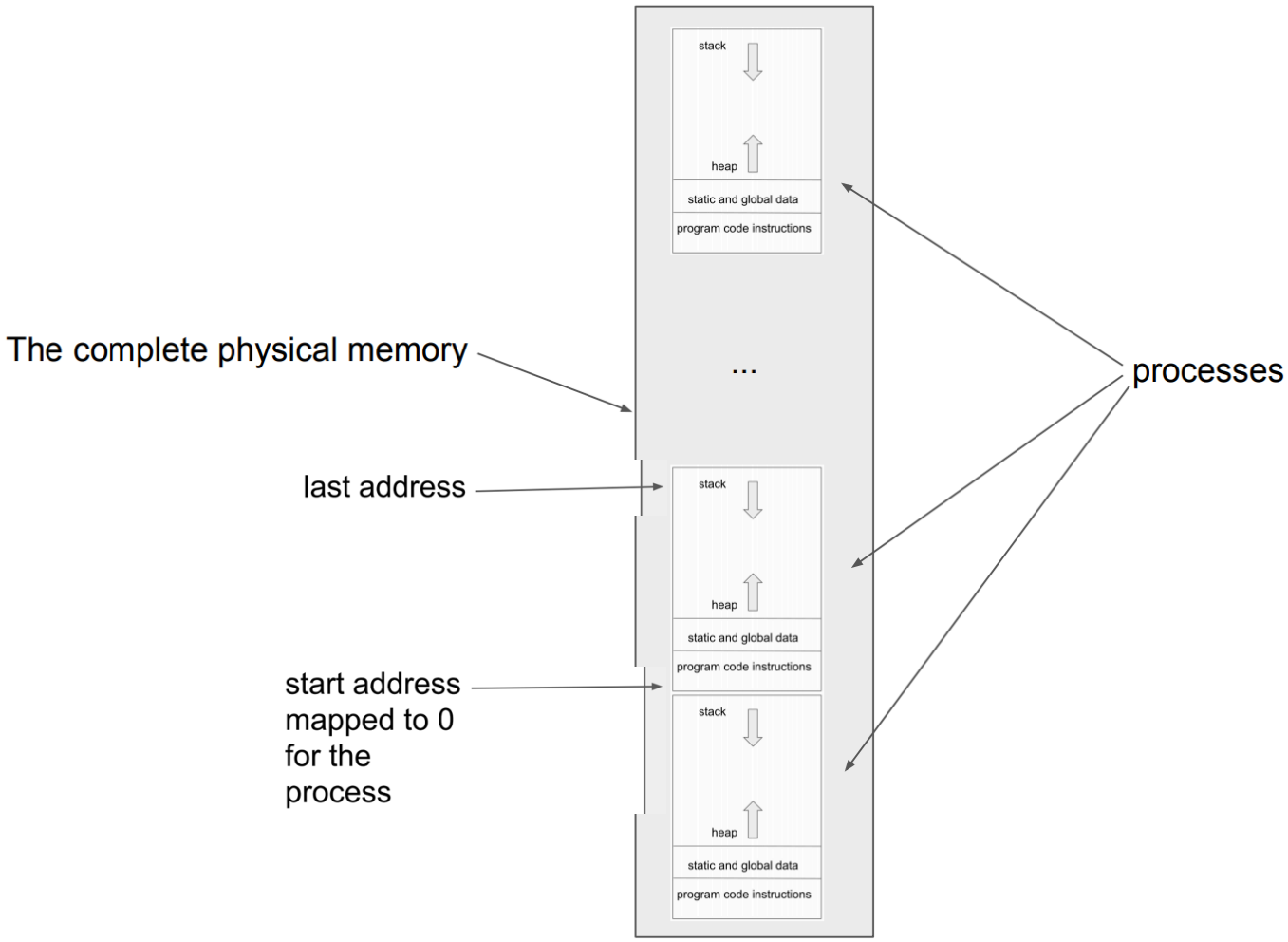
- The approach of implementing file systems with standard interfaces and then grafting them into the name space gives great flexibility.
- There is even a file system module that allows the file system implementation to run in user space called "fuse".
- This means normal users/developers can easily implement a new service and put it into the namespace.

INFO1112 Week 4 Lecture – Memory Management, Scheduling, Boot Sequence

Programs in Memory

- When a program is ready to run as a process, it is stored in memory in a particular layout → the stack is for temporary variables whilst the heap is for data structures.





Memory Mapping

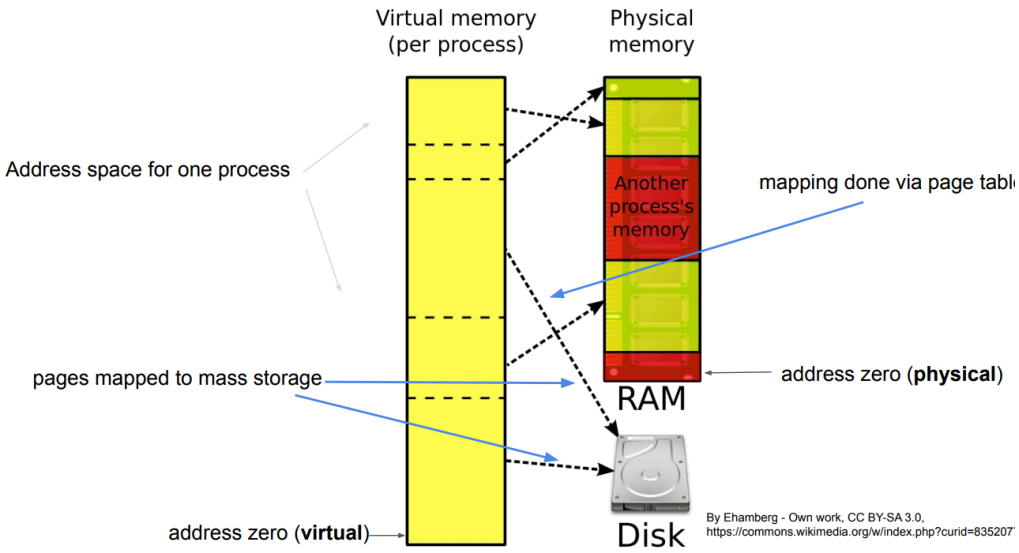
- The problem with this is that there are many processes in the memory at the same time. How can they all be located at address zero? → this is achieved by hardware in the CPU that remaps the address space for each process when it is running.
- For example, if a process memory image is stored starting at the real address 1000, then the kernel of the operating system can use privileged instructions to set a special register that says “all references to addresses are offset by 1000”, that is, a reference to address 0 will actually refer to address 1000.
- In addition, there is a register that puts an upper limit on the addresses the process can use. This prevents the process code from accessing beyond its allocated memory area.

Virtual Memory

- This simple memory mapping scheme was the early approach to memory management and protection.
- Now we use a more sophisticated (and complex) scheme that involves remapping smaller blocks of memory called "pages".
- There is a table in the system that contains an entry for every page of the process that maps the virtual address of the page used by the process to the physical address in the main memory or an indication that the page is currently on mass storage.
- This is known as the page table. The kernel manages this table and moves pages to and from mass storage.

Page Table Example

Process ID	Process (Virtual) Address	Physical Memory Address	Mass Storage Address
1456	100000	12000000	0
1456	120000	0	32100
...	...	...	...
...	...	...	...



- What happens if a program attempts to use memory that the page table has mapped to mass storage?
- In that case, the access will cause an interrupt that will transfer control to the kernel → An interrupt is like a forced function call to the kernel that occurs between two instructions in such a way that the CPU state is saved and can be restored when the kernel restarts the process.
- In this way the process has no indication that it was interrupted.
- When the kernel gains control, it will find the page on disk (stored in the swap area) and read it into a free part of main memory → then setup up the page table to point to the new physical location and return to the process.
- If the kernel needs to read in a page from disk and discovers there is no free physical memory available, it has to first write an existing block out to the swap area to free up the memory.
- If this happens too frequently, the system will become very slow and inefficient → this is because the mass storage is much slower than RAM and it takes a long time to position a disk and read the page into memory.

Virtual Memory Optimisations

- "shared memory" means that some processes have pages that are mapped to the same physical memory. In this way they can either share read-only data, or read-only code such as library functions. Or share writable memory for sharing data (careful!).
- "copy on write" means that when a process forks, it shares physical memory pages between the processes until one or other wants to change (write) a page. At that point a writable copy is made.

Scheduling

- As we saw earlier, one a single CPU system, only one process can execute at a time.
- Other processes in the memory are sleeping - waiting for some resource to become available or waiting for their turn to use the CPU.
- Usually the resource they are waiting for is usually some form of I/O: waiting for the data to be delivered from a peripheral such as a disk drive or a keyboard etc.
- Sometimes the process can be waiting for the virtual memory system to bring in a page.
- Eventually each process will need to execute and so need a turn using the CPU.
- Deciding which process gets to run next is called scheduling.

Scheduler

- The basic idea is that the scheduler maintains an ordered queue of processes waiting to run → the ordering is decided with some measure of priority.
- The simplest scheduling algorithm is called "round robin" scheduling because it give each process an equal slice of the CPU time and just circulates around processes. More sophisticated schedulers take into account a priority value and other factors.
- A lot of research has been done on schedulers and there is a very large number of algorithms. Some of the earliest and important work was carried out here at USYD by Prof Judy Kay and Piers Lauder who developed an entire class of schedulers called "fair share" schedulers.

Schedulers and Linux

Processes in Linux have a priority value that can be seen on a "ps al" listing:

```
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY      TIME COMMAND

4     0     1     0    20    0 185248  5792 -      Ss   ?       1:07 /sbin/init
```

priority

- The priorities range from negative numbers (the highest priority) to positive numbers (the lowest priority). A normal user process starts with priority 20.
- You can set the priority of one of the processes you create using the “nice” command.

```
nice -N command
```

- This will add N to the priority of the command when it is run.

```
$ ps l
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY      TIME COMMAND
0  1001  3577  3574  20   0  21288  5104 wait   Ss   pts/1       0:00 -bash
0  1001 12728  3577  20   0  27640  1532 -    R+   pts/1     0:00 ps l
0  1001 29861 29860  20   0  21288  5092 wait_w Ss+  pts/0       0:00 -bash
```

```
$ nice ps l
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY      TIME COMMAND
0  1001  3577  3574  20   0  21288  5104 wait   Ss   pts/1       0:00 -bash
0  1001 12729  3577  30  10  27640  1436 -    RN+  pts/1     0:00 ps l
0  1001 29861 29860  20   0  21288  5092 wait_w Ss+  pts/0       0:00 -bash
$
```

- You can change the priority of a running process using the “renice” command.

```
$ renice -n N process-id
```

Starting the System: Bootstrapping

- Relates to the process from turning on power until you have a running system.

First Stage – Hardware

- The computer has several sorts of main memory: RAM (Random Access Memory), ROM (Read Only Memory), and flash memory.
- The RAM is volatile - the contents are lost when the power is turned off.
- The ROM is persistent - the contents are maintained even with the power off but ROM cannot be changed after initial manufacture.
- Flash memory is also persistent but can be changed by a program.
- The ROM contains the initial instructions that are executed when the power is first turned on for the first time. When you first turn on the computer the hardware does some test operations. This is before any program is executed.

Second Stage

- When you turn on a microcomputer such as Arduino based systems (as used in ELEC1601), the start-up is very simple.
- The first instruction to be executed is at a particular memory location in the address space → this initial address is in persistent memory (flash or ROM) and is usually address zero.
- At this initial address there is a jump instruction that transfers control to program code for the next stage of the boot sequence. The program code is also stored in persistent memory.
- This stage does more hardware checks, works out what devices are connected (disk drives etc).

Third Stage

- The program stored in the persistent memory (also called the firmware) then works out where to find the next stage of the boot program, loads it into memory and jumps to it.
- Typically this is done by looking at a fixed list of places that is stored in the flash memory.
- You can look at this list in older PCs using the "BIOS" (Basic I/O System) program → the locations are devices like hard drive, USB mem, CD ROM drive etc.
- The program it loads is stored on disk in the Master Boot Record (MBR), normally the first block on the device.

Master Boot Record

- The MBR is very restricted in size → it is stored on the first block of a disk (or other mass storage) and is less than 512 bytes long.
- It contains a program that reads in the next stage of boot program (in Linux it's called GRUB - GRand Unified Boot loader) as well as a table of device configuration for the boot device.

Fourth Stage

- The program (GRUB) that is read from mass storage can be much larger and it will typically have a driver for a file system on mass storage → this is because the kernel of the operating system is stored on a file in a file system.
- This part of the boot program will navigate the namespace of the file system on mass storage and find the file containing the kernel → it will then load it into memory and start it running.
- There may be several versions of the kernel in the file system and GRUB works out which one to load.

Fifth Stage

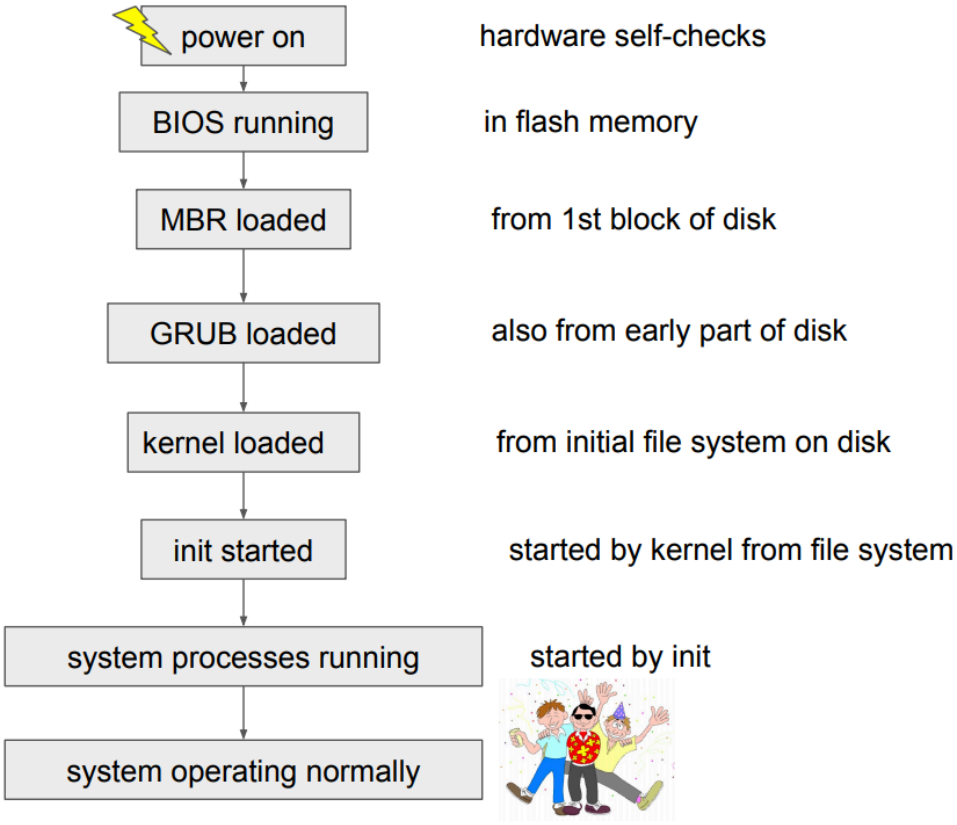
- The kernel is now running.
- It performs final, high-level hardware checks and then loads device drivers and kernel modules as required → this is decided by configuration files stored in the initial file system.
- The kernel will then mount file systems.

Sixth Stage

- The kernel is now ready to start real processes.
- The first process in a Unix system is called "init" → this initial process is started by the kernel and this is the only process that the kernel starts.
- From then on processes are only started by other processes and "init" is the first → all processes can trace their ancestry back to init.
- It starts system processes (daemons) and initializes all active subsystems.
- In a server only system (no graphical display) it will start a "login" program.
- For a windows graphical interface system, init will start a window manager, which in turn runs the login process.

Seventh Stage

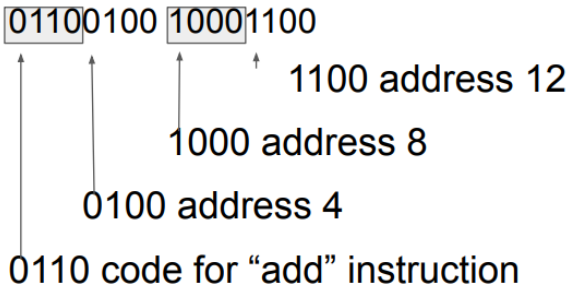
- At this point, everything is loaded, logins are accepted, background tasks are running.



INFO1112 Week 5 Lecture – Virtual Machines and Containers

CPU, Instructions, and Registers

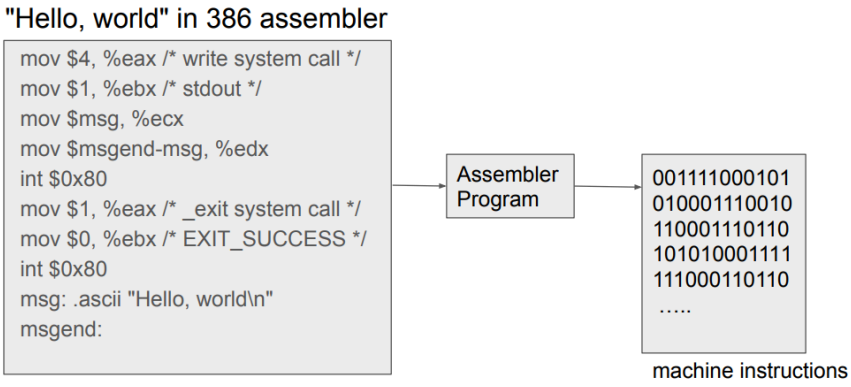
- The CPU reads instructions from memory and interprets or executes them.
- The idea to store instructions in the same memory as values was the major breakthrough in the development of computers → "stored program computers".



- As well as general purpose registers for intermediate results, the cpu might have special registers that contain addresses or offsets used when fetching a value from memory → making moving through strings and lists of things easier.

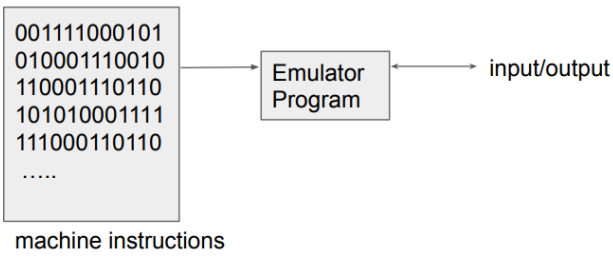
Assembly Language

- When writing programs, we use a human readable shorthand (for 01100100…) called 'assembly language'.
- Our hypothetical instruction might be written as: ADD 4, 8, 12.
- We use a program called an Assembler to read instruction in assembly language and translate them into the binary numbers that represent machine instructions.

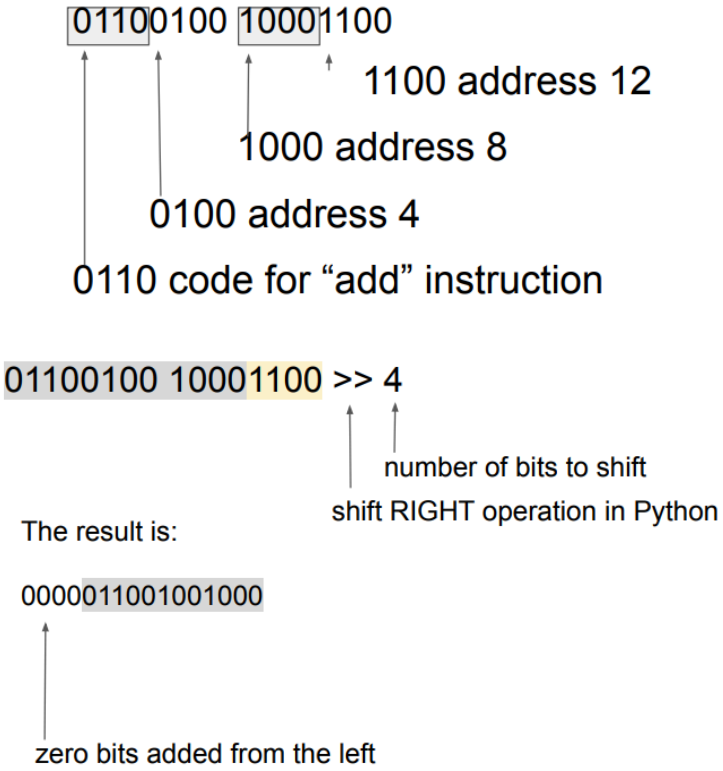


Emulator Programs

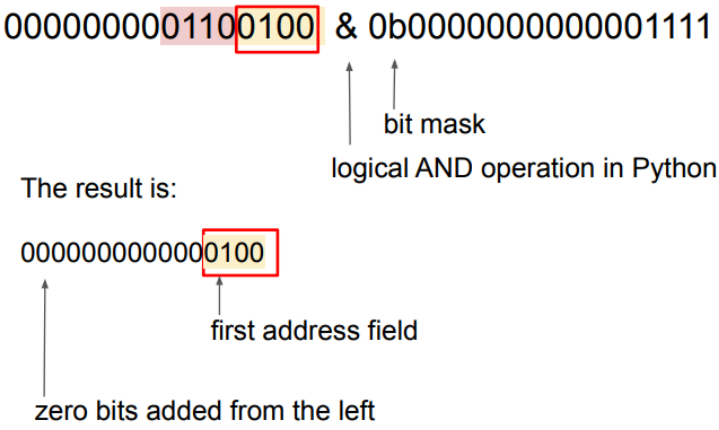
- As well as using a program to translate Assembly language into binary machine code, we can also write a program that reads the binary instructions, works out what they mean, and then carries out the instruction.
- This type of program is called an Emulator or Virtual Machine, because it emulates the instructions, pretending to be the real CPU.



- An emulator reads each instruction as a binary number and breaks it down into the key parts, using logical operations - shift and AND and OR.
- A shift operation simply shifts the bits left or right the specified amount → usually a shift to the right will lose any bits that go off the end, similar for the shift left.



- This result is the instruction operation code.
- If we wanted to find the first address operand in the instruction, a simple shift right by 8 would still leave the operation code. Hence, we use:



- This is called MASKING → where the second number can be thought of as a "mask" that allows bits through wherever the mask has a one.

Emulators and Virtual Machines

- Theoretically, we can write an emulator program in any language to emulate ANY existing instruction set and so run programs written for any other computer on our computer.
- The downside is that our emulated programs will run very slow → this can be sped up by writing it in a low-level language and optimising how we do things.

Virtual Machines

- Emulators generally emulate all the instructions and registers of a CPU but if they also emulate all the system aspects such as memory management and I/O, we usually call them Virtual Machines.
- VMs are very complete emulations capable of running complete operating systems in the emulator → eg. you can run Linux on your Windows machine by running it in a virtual machine.
- By far the most popular are VMs for the Intel 8086 architecture found in Windows PCs.

Why Use a Virtual Machine?

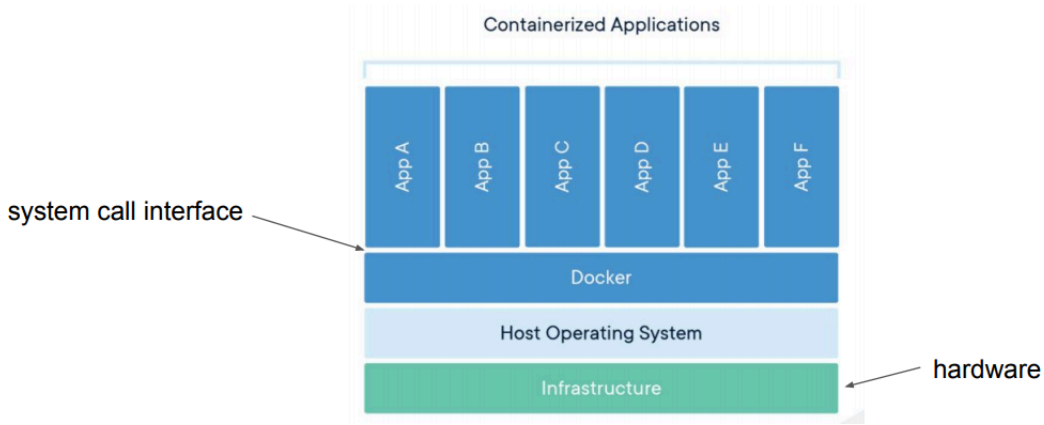
- It allows you to run a different or different version of an operating system.
- Allows you to run older software.
- Provides isolation (mostly) from bugs or malicious code.
- Allows you to test software in a different environment.
- Allows you to develop and test software on many different target environments from the same machine.
- Allows many small systems to simultaneously run on a single large machine → cloud provides extensively use VMs.

Another Approach: Containers

- The application view of its environment is the system call interface and the namespace.
- Since the namespace (files, devices etc.) are all accessed via the system call interface (open, close, read, write, fork, exec etc.), if all the system calls from an application could be captured and emulated, the application could be tricked into thinking it was running in any required environment or system.

Containers: Docker

- A very popular container systems is produced by Docker Inc.

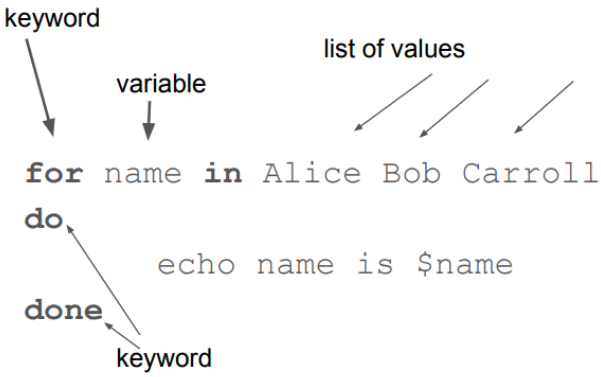


- Containers are much lighter weight than full VMs and are widely supported on cloud systems (remotely accessible computer servers sold as a service).
- They have many of the advantages of VMs e.g. isolation.

The Shell

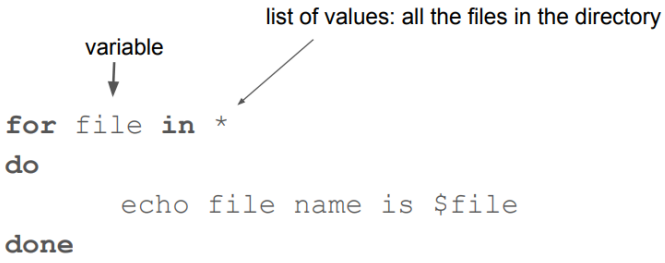
Shell for Statement

- The shell has a “for” statement like conventional programming languages. The statement repeats a loop for each value in a list of values.
- The list of values could be generated by a shell wildcard or the output of a program (backquotes).



General form is: **for** variable **in** list **;** **do** statement list **done**

Don't forget that the list of values could be generated by a shell wildcard or the output of a program (backquotes):



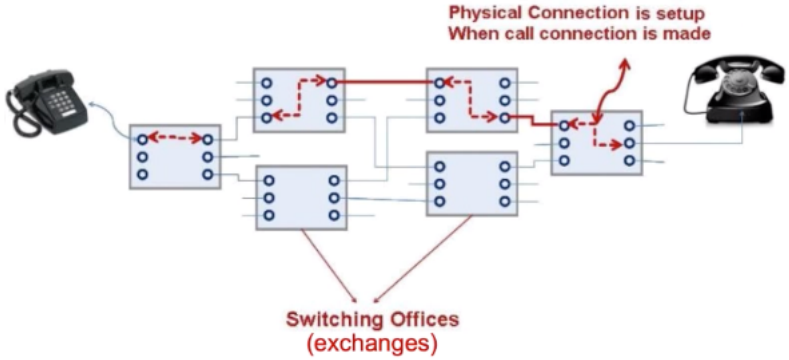
INFO1112 Week 6 Lecture – Introduction to Networks

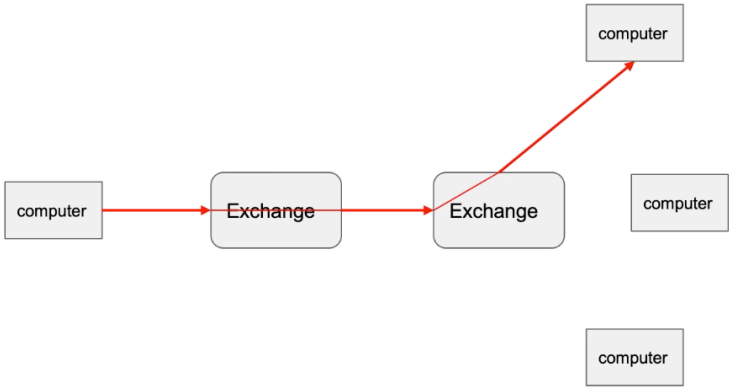
History

- The internet of today has its origins in the 1960's with the development of "packet switched networks". Up until that time, computers were connected using telephone network technology ("circuit switching").
- Circuit switching involves creating a complete circuit form the source to the destination when a call is requested and shutting the circuit down when the data transfer is complete.

Circuit Switching

- With circuit switching, the data is sent in a continuous stream over the same circuit for the duration of the call.
- The intermediate systems are usually called exchanges and there were relatively few of them.
- Circuit switching for process to process communication is now rare.





- Advantages:

  → Each data connection has a reserved bandwidth and so there's a constant delay for data flowing over the connection.

- Problems:

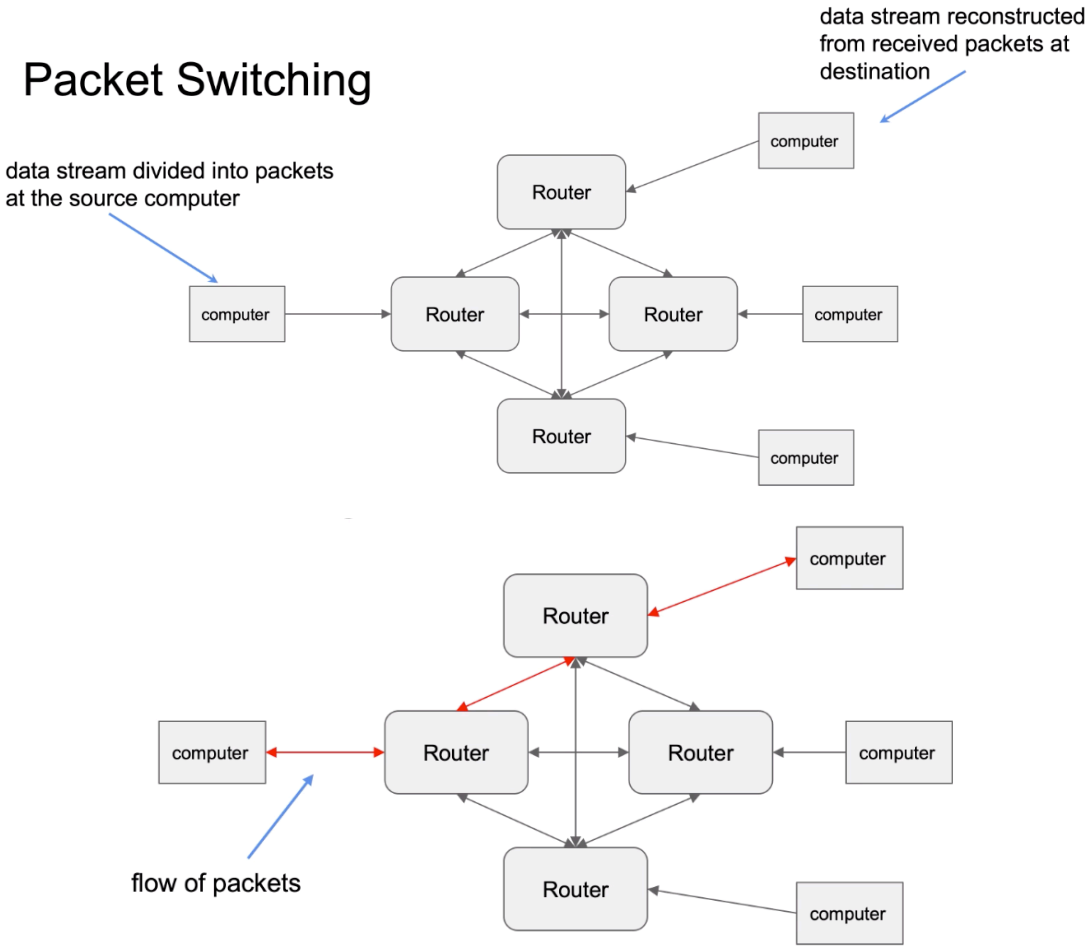
  → Reserved bandwidth is wasted if no data is being transmitted.

  → An interruption to the circuit cannot be recovered from without making a new call.

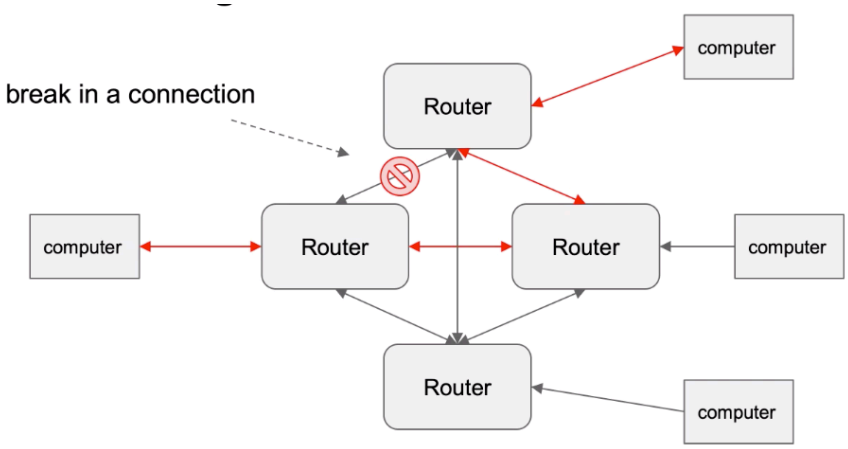
- The technique of packet switching was invented to fix these problems, but at the cost of giving up guaranteed bandwidth.

Packet Switching

- Instead of sending data as a continuous stream over a fixed circuit, packet switching involves breaking the stream up into small chunks or packets, sending them independently through the network and reassembling them in the correct order at the destination.
- The packets don't necessarily all follow the same path or route through the network.



- Packets can follow different routes through the networks, so they can be rerouted around breaks or congestion in the network → it gives the network resilience in the face of faults and overloading.



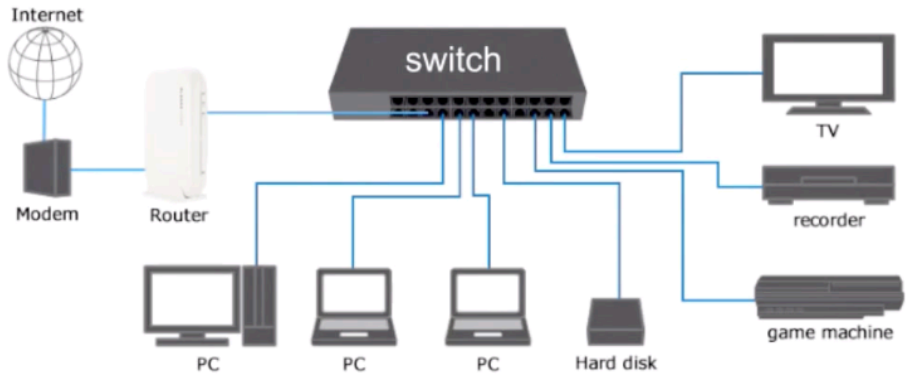
- Another major advantage is that the data from different connections can share a channel → the most common network technology today involves packet switching and by far the most widely used standards are the Internet standards.
- Even phone calls are now transmitted using internet technology and not old-style circuit switching.

LANs and WANs

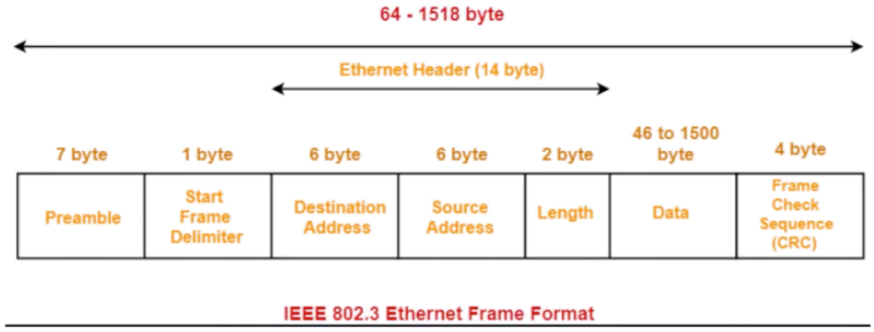
- Connections between computers fall into two broad categories → local area networks (LAN) and wide area networks (WAN).
- A LAN can be as small as your home and as large as a campus. A WAN is global and consists of interconnected LANs.
- The term “interconnected networks” is where we get the term internet.
- Both LANs and WANs use the packet switching technique to transfer data.

Local Area Networks

- The most common form of LAN uses ethernet technology for the interconnection.
- In a wired ethernet network, the computers are connected to each other using twisted pair wiring via a router or switch.



- A WiFi network in your home carries the packets using wireless connections. This consists of a WiFi access point or switch that each device connects to using wireless transmission.
- In wired and wireless ethernet networks, the data packet is carried in an ethernet frame (how the data is formatted) which is another sort of packet. So we essentially have a packet (data) inside another packet (ethernet).



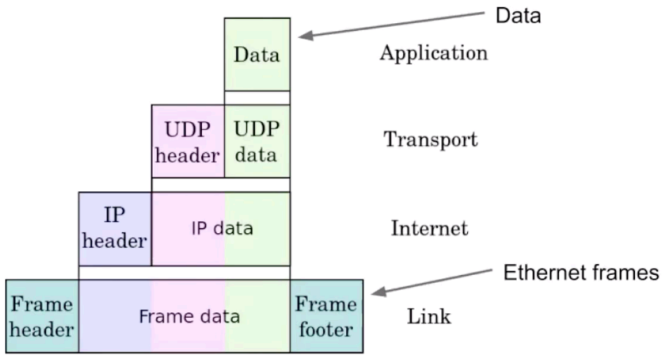
Source and Destination addresses are 6 bytes (48 bits) long and are unique.

Ethernet

- Ethernet packets or frames are carried point to point only from a device to the access point or switch → the data packets are forwarded onto other switches and eventually reach their destination.
- The data packets may make many hops along the route to the destination. For each hop, they will be carried by a different lower level network message → this is called layering.

Layering

- Network services often have many layers where a message is carried inside another message which is carried inside another and so on.

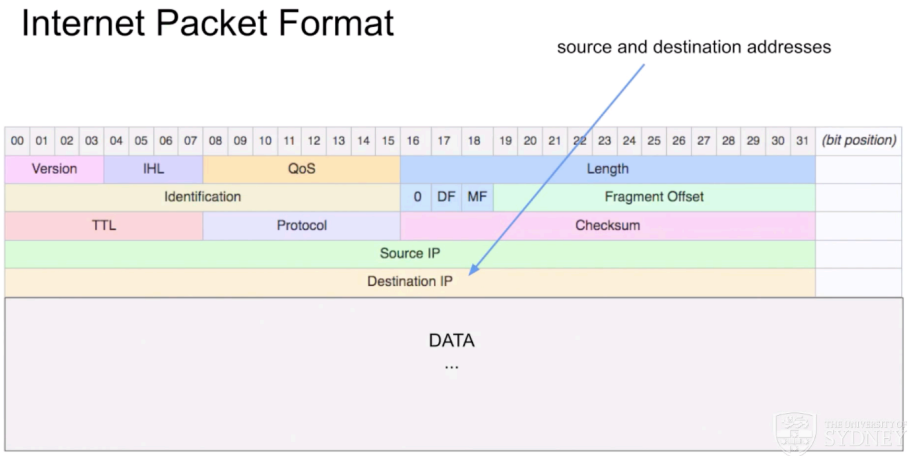


Internet History Continued

- The early internet packet switching technique was developed by the British and Americans at about the same time. The first production network was in Britain but the system that developed into today's internet came from the USA.
- The Advanced Research Project Agency (ARPA) in the USA funded a major research project on packet switched networks in the 1960's at the height of the cold war → gave the ability to withstand major nuclear war disruption.
- The first network was called ARPANET and started in 1969, linking universities and research centres across the USA.
- In the late 1970's, after 10 years of development, the basic protocols were revised and at the beginning of 1983, the network switched over to the system we use today.
- The first internet link to Australia was established in 1989 by a team from USYD, UMelb and NASA.

Internet Packet – IP

- The data stream is divided into chunks called internet packets or IP.
- Internet packets can be up to 64K bytes long, but are typically much smaller e.g. 1500 bytes.
- The packet consists of a header containing the length and other information about the packet including the destination address and source address. This is followed by the actual data.
- The header is 20 bytes long , with an important part of the packet being the address fields.



Internet Packet Addressing

- The source and destination address are 32 bits or 4 bytes each.
- The convention is that an internet address is written as four decimal numbers in the range 0-255 separated by dots. This is a lot easier to write than a 32-bit binary or hexadecimal number.
- An IP address is used to identify each computer and also to determine the route to the computer.

Internet Protocol

- A network protocol is the set of rules for controlling the sequence of packets that are exchanged between computers.
- The network protocols have to be implemented by all the computers on the network. The Internet has a set of these protocols that govern all the layers of the network.

History: Internet Protocol Documentation

- As the internet protocols and procedures were developed, they were documented in a set of documents called “Request for Comments” or RFC. The name comes from the fact that they were used originally for proposals that eventually became internet standards.
- The internet tradition is that protocols are developed by committees that anyone can join through an organisation called the Internet Engineering Task Force or IETF.

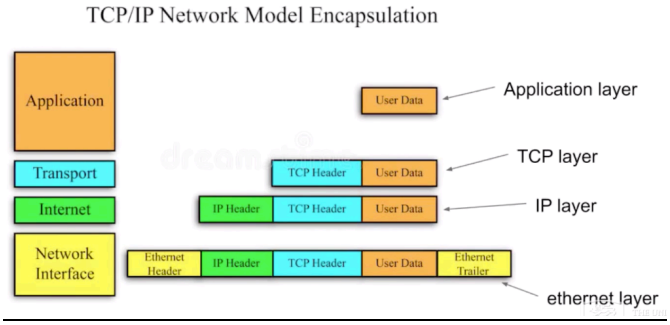
Internet Protocol Errors

- A common requirement in computer to computer communication is to send a large message or stream of data. We can use packet switching to break up the data stream and send the packets independently, but this alone can have problems.
- If a package is lost or corrupted due to a hardware failure (electrical interference) or congestion is detected and the packet travels via a longer route, this can cause packets to fail to arrive or to arrive out of order. We can detect most data errors at the Internet Packet level using a checksum.
- How can we recover when an error is detected?

Transmission Control Protocol – TCP

- The solution is to create a protocol layered on top of the Internet Packets, that carries the data packets – layering!

- This protocol maintains a sequence number in each packet so out-of-order packets can be reassembled in the correct order, as well as a way of requesting the retransmission of packets with errors or those that don't arrive in a reasonable time, and a way of acknowledging that packets have been received correctly.
- This protocol is known as TCP.

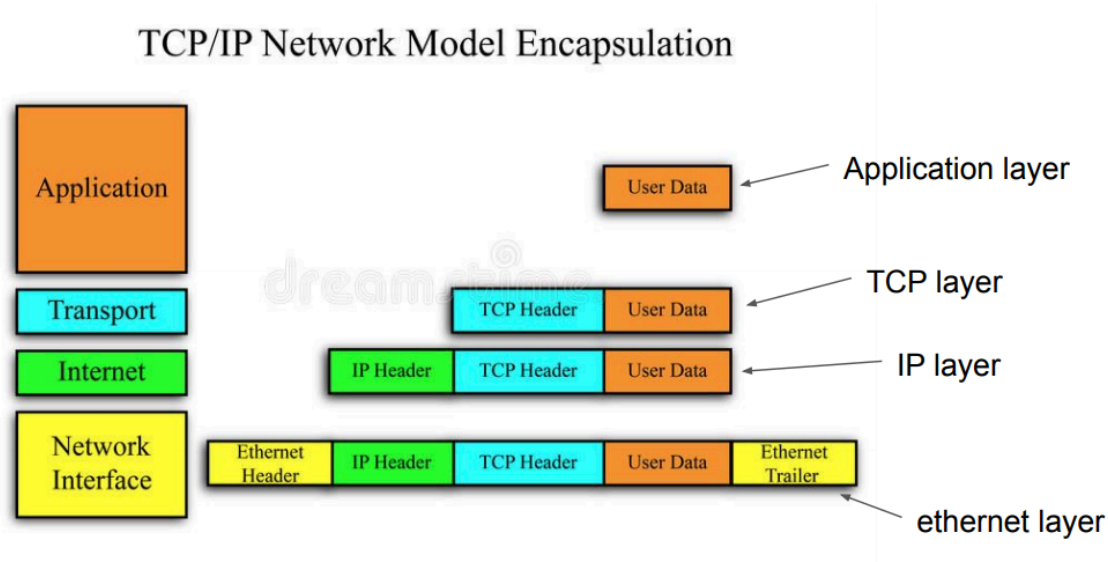


Application Layer

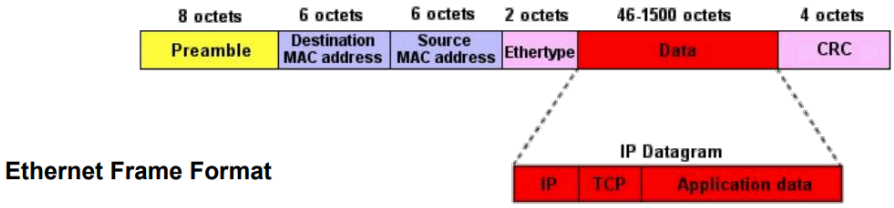
- What has been called "user data" is actually more layers since it is usually data for an application. For example, a web page or video stream etc.
- These applications will usually have their own protocols with packet formats and rules for exchanging messages.

INFO1112 Week 7 Lecture – LAN and WAN Routing

TCP/IP Network Layering



- Internet frames are used to carry higher level protocol messages.
- The physical layer (hardware) could be wireless e.g. WiFi, or wired e.g. ADSL, fibre, but the ethernet frame format is the same.
- The ethernet layer uses globally unique 48-bit addresses and every device has an address assigned during manufacture. Ethernet frames are usually sent over a single hop while higher level packets may travel through many switches and routers.



Ethernet Frame Format

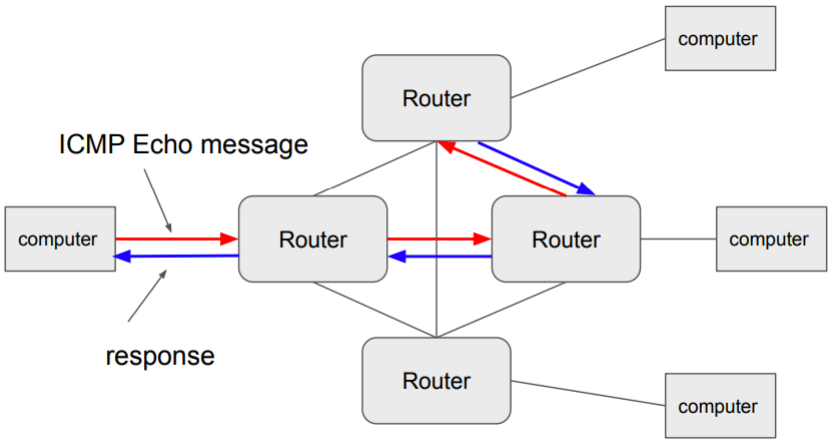
IP Packet Level Protocols

- The internet packet carries higher level protocols. The most important are the Transmission Control Protocol or TCP, and the User Datagram Protocol or UDP.
- The TCP provides a reliable stream using sequence numbers on the packet and requesting retransmission for lost packets.
- UDP is a simple short message type protocol that offers no guarantees of delivery unless you embed a higher-level protocol.
- In addition to TCP and UDP, there is also a protocol for exchanging control messages about the Internet itself: Internet Control Message Protocol or ICMP.

ICMP

- ICMP provides messages that indicate the status of a connection or error.
- ICMP has a simple echo request where the destination simply sends the packet back to the source. This is used in the 'ping' command to test if a host is responding.

- An ICMP can have a time-to-live (actually a number of hops) set on an echo message. When this number of hops is reached, the intermediate node sends the message back with a 'time-exceeded' indication → used to trace the route through networks.



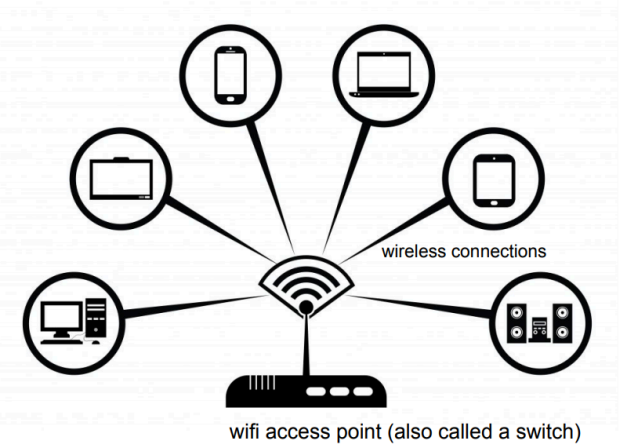
- The above traceroute has a maximum hop count of three which is then exceeded.

Routing: Deciding the Path

- The IP packets are transported through the network, hop by hop, to the destination. Each packet has an IP address, a 32 bit or 4-byte number written as 4 decimal numbers in the range 0 – 255 separated by dots e.g. 129.78.8.1.
- This IP address is used to decide how to get the packet to the destination.

LAN Routing

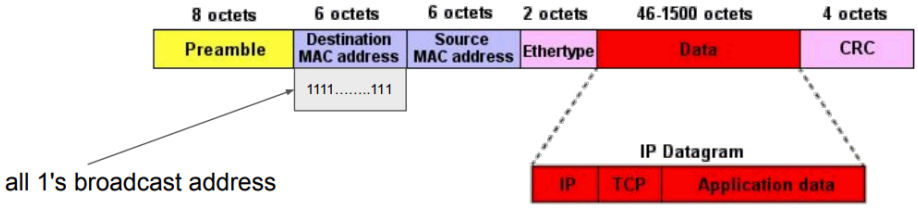
- A Local Area Network is usually a "broadcast" network on a shared medium. A good example is WiFi: every node on the network can hear packets sent by other nodes.
- It would be very inefficient if all the nodes had to decode every packet sent. Nodes usually only decode packets that have their (ethernet) address as the destination in the packet (ethernet frame).
- How does a node discover the destination ethernet address of another node?



WiFi Network

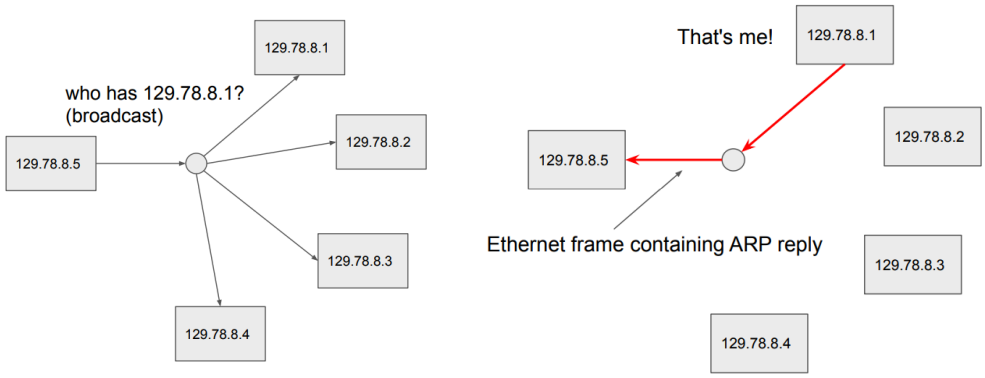
Lan Broadcast Addresses

- The answer is that in addition to decoding packets sent to its own address, a node will also decode packets sent to a special “broadcast” address. An ethernet broadcast address consists of all 1s.



Address Resolution Protocol

- The Address Resolution Protocol or ARP is used by a node to discover the ethernet address of other nodes. A node will send an ARP message to the broadcast address that says, “who has xxx.xxx.xxx.xxx?”
- The target node will hear that on the broadcast address and reply with a message directly to the source node and so the source will discover the Ethernet address of the target.

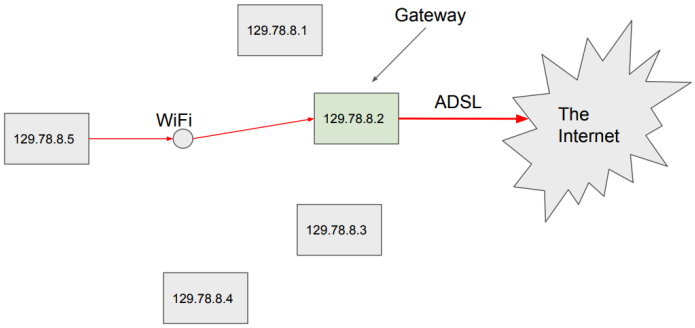


LAN Addressing

- A typical block would be 256 addresses long e.g. 129.78.7.0 to 129.78.8.255.
- All the addresses in that block would be found inside a given LAN → to test if a node address is inside this LAN, simply use a mask that is 8 1s or 255. If what remains is 129.78.8.0, the node is inside.

Beyond the LAN

- If the required destination is outside the range, how do you detect it? → All nodes inside a LAN have an IP address chosen from a contiguous range. This range is always a block starting with the right-hand bits set to zero. The first node has address 1 in the block.
- If a destination address is outside the LAN block of addresses, then the node sends the packet to a preconfigured “gateway” address that is inside the LAN → basically “I don’t know who this is – you handle it!”.



Routing Table

- We can generalise this with a “routing table”, with the interface to be sent to the packet displayed at the end.
- It is displayed with the Linux “route” command, with more entries able to be added to the table.

```
Kernel IP routing table
Destination  Gateway      Genmask      Flags Metric Ref  Use Iface
default      gw-11.cs.usyd.e 0.0.0.0       UG   0    0        0 em1
129.78.8.0   *            255.255.252.0   U     0    0        0 em1
```

WAN Routing

- Routing outside the LAN on the general internet is done with similar routing tables, but in this case, the tables are maintained by “edge systems” talking to each other using a routing protocol such as “Border Gateway Protocol”.
- The IP address range is divided into blocks called “Autonomous Systems (AS)”, and the information about each AS is exchanged between routers running BGP.
- In 2018, there were over 60,000 ASs, and the field for an AS number has recently been increased from 16 bits (64k) to 32 bits.

Who Manages the Numbers?

- Addresses and related numbers in the internet are managed by an organisation called the Internet Assigned Numbers Authority. For many purposes, IANA is responsible for top level allocation, but delegates lower level allocation to regional organisations.
- E.g. IP addresses in the Asia Pacific region are allocated by the Australia Pacific Network Information Centre (APNIC) in Brisbane.

Running out of Numbers?

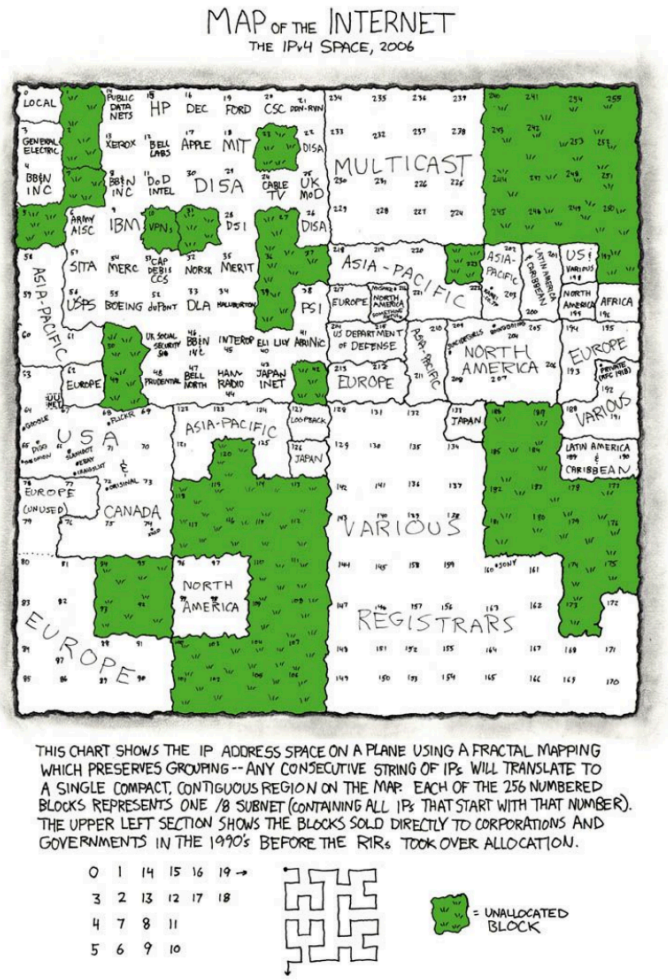
- In version 4 of the Internet Protocol (IPv4), the addresses are 32 bits (roughly 4.3 billion). This seems like a lot, but we have practically run out of addresses. Some areas have small unassigned blocks and are recycling addresses.
- The most common solution is to reuse addresses from address ranges that have been designated as “private” by IANA:

10.0.0.0 to 10.255.255.255
172.16.0.0 to 172.31.255.255
192.168.0.0 to 192.168.255.255

- A configured home router or access point will almost certainly have an address in this range.
- If an address Is in one of the private ranges, it has to be “translated” to a single non-private address at your router. The router remembers which private address is used and manages this translation → Network Address Translation or NAT.

XKCD address map from 2006

Most of the unallocated areas are now gone (2019).



IPv6

- The complete solution to the address exhaustion problem is a new version of the Internet Protocol: version 6 (version 5 was experimental and never went anywhere).
- IPv6 has 128-bit addresses and a better approach to routing.
- Unfortunately, because IPv4 works so well, techniques like NAT are still used in such wide deployment that it is taking a very long time to switch over to IPv6 → all systems have it available and all routers implement it.

Different Shells

- The shell is just an ordinary program that reads shell commands, either interactively from the user typing or from file (shell script).
- We can designate any program as a "shell" and write scripts that are interpreted by the program.
- To tell the system what program to use to interpret your script, we put a line at the beginning of the script:

```
#!/home/bob/myshell
```

- When the system wants to run a program after an "exec" system call, it looks at the first two bytes (16 bits) of the file. If it is "#!", it runs the program specified on the rest of the line with input redirected from the file.
- Languages that are not compiled run this way, such as Python e.g. at the start of a python program, you put:

```
#!/usr/bin/python
```

INFO1112 Week 8 Lecture – Domain Name Service and Application Layer Protocols

Domain Name System

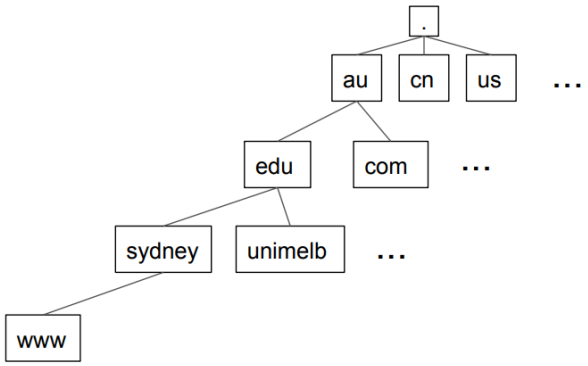
- In the very early days of the internet, computers on the net had addresses like 129.78.5.11, but they also had simple names that were human friendly → the mapping of the human friendly name to an IP address was stored in a single file (hosts.txt) and distributed at regular intervals to all computers on the net (at that time there were only a few hundred computers on the network).
- The network was rapidly growing, and a better solution was needed. In 1982, a more efficient system was developed called the Domain Name System or DNS → it since been updated with better security, but essentially remains the same.

DNS Efficiency

- The old system' inefficiency was on account of it being centralised – a single file containing all the name and number pairs → the key to improving the efficiency of domain name lookup was to make the service distributed.
- Instead of looking in a single file in your system that was only updated occasionally, in the DNS, you perform lookup requests via the network to remote machines that have the necessary information.

How is it Distributed?

- The domain names make up a hierarchy. A name like www.sydney.edu.au is a little like a file pathname except we use a dot instead of a slash and it is in reverse order. The domain names form a tree.



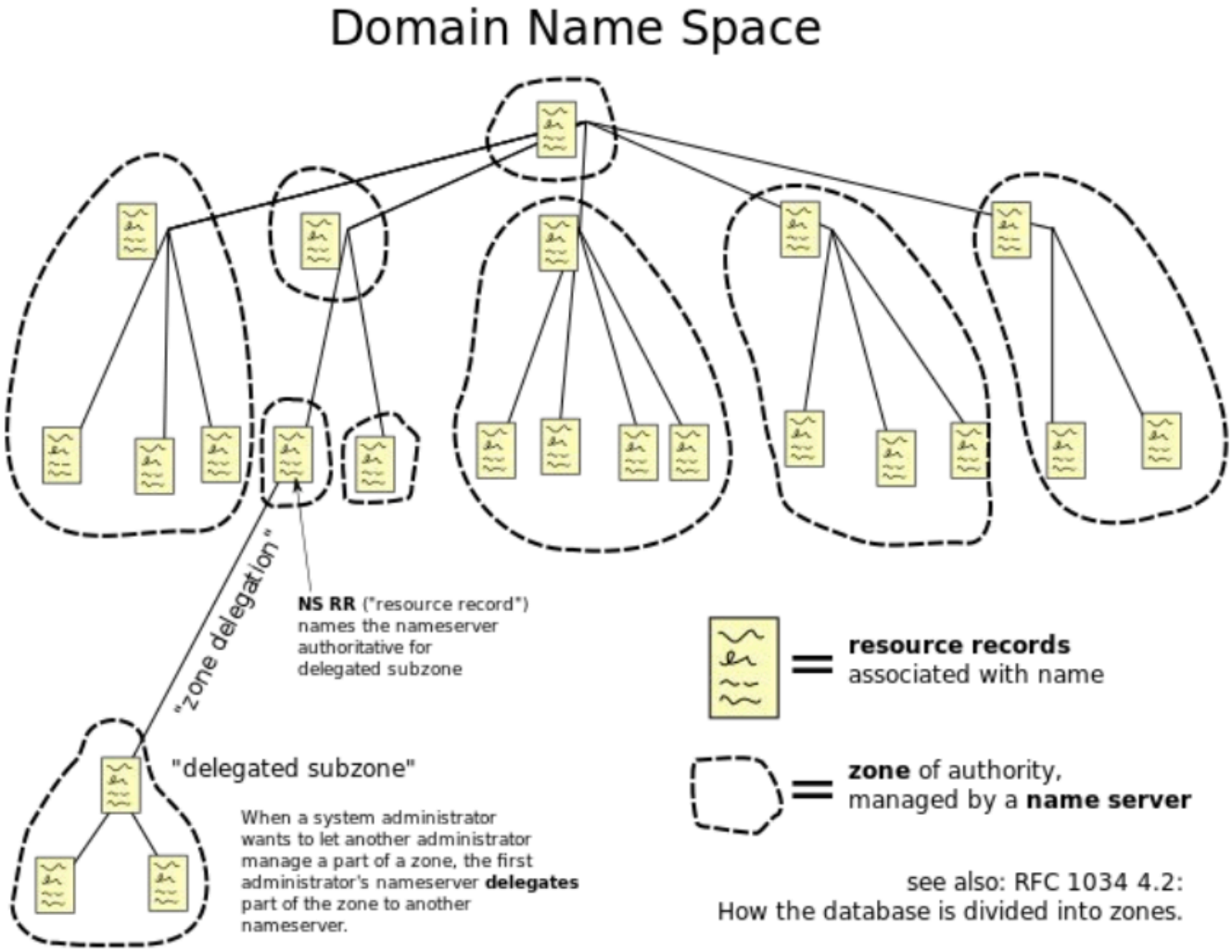
- The root (top dot level) is usually not written, but contains the names of other country top-level names.
- Critical to the operation of the Internet, the advanced of having the database distributed is that each zone is responsible for managing names in that zone → distributed management e.g. USYD updates .sydney.edu.au DNS records when machines are added or removed.

Nameservers

- At each level in the hierarchy, there are computers that run "nameservers" for that level and maintain information about the level below.
- When you start your computer, the network configuration is loaded that contains its IP address and the IP address of a nameserver to use for DNS lookups.

Name Lookup

- When your system wants to lookup a domain name, it asks the preconfigured nameserver using the DNS protocol. The DNS protocol sends messages using UDP.
- The domain namespace is divided into “zones” that each have one authority managing the name space and one or more nameservers that handle queries for domain name records.



How Does a System Know Who to Ask?

- When your system is started it needs to find its IP address and the name server to use. The most common method is to use the DHCP protocol: Dynamic Host Configuration Protocol.
- Your system sends out a broadcast packet that requests configuration information from any available DHCP server. The server responds with a message that contains the IP address and the IP address of the DNS server to use.

DNS Records and Record Types

- The actual domain name service database contains DNS resource records that contain information related to the name.
- Each record contains:

Name	full domain name of the node in the tree
Type	Type of the record, ie what information it carries
Class	Class, really only IN meaning Internet
TTL	Time to live in seconds before this record expires
Data Length	Length of the data field
Data	the value of the record

- There are many types of record stored in the DNS:

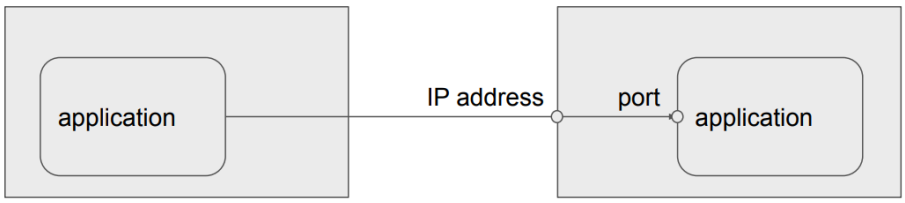
Type	Description	Example
A	IPv4 address	129.78.8.1
AAAA	IPv6 address	fe80::3617:ebff:fea1:e43f
CNAME	alias	e14794.dscj.akamaiedge.net.
MX	Mail Exchanger	au-smtp-inbound-1.mimecast.com.

Application Protocols

- The upper layers of protocol in the Internet are where application programs communicate → two major examples of application layer protocols are HTTP (HyperText Transfer Protocol) used to fetch web pages, and SMTP (Simple Mail Transfer Protocol) used to send email.
- These protocols are based on simple lines of text that are sent to and from the relevant server.

TCP Ports

- The address of the source and destination is an IP address such as 129.78.8.1 → TCP connections also have an extra level of addressing called a port.
- A port is a 16-bit number that identifies an application running within a system and listening for connections to that port number.

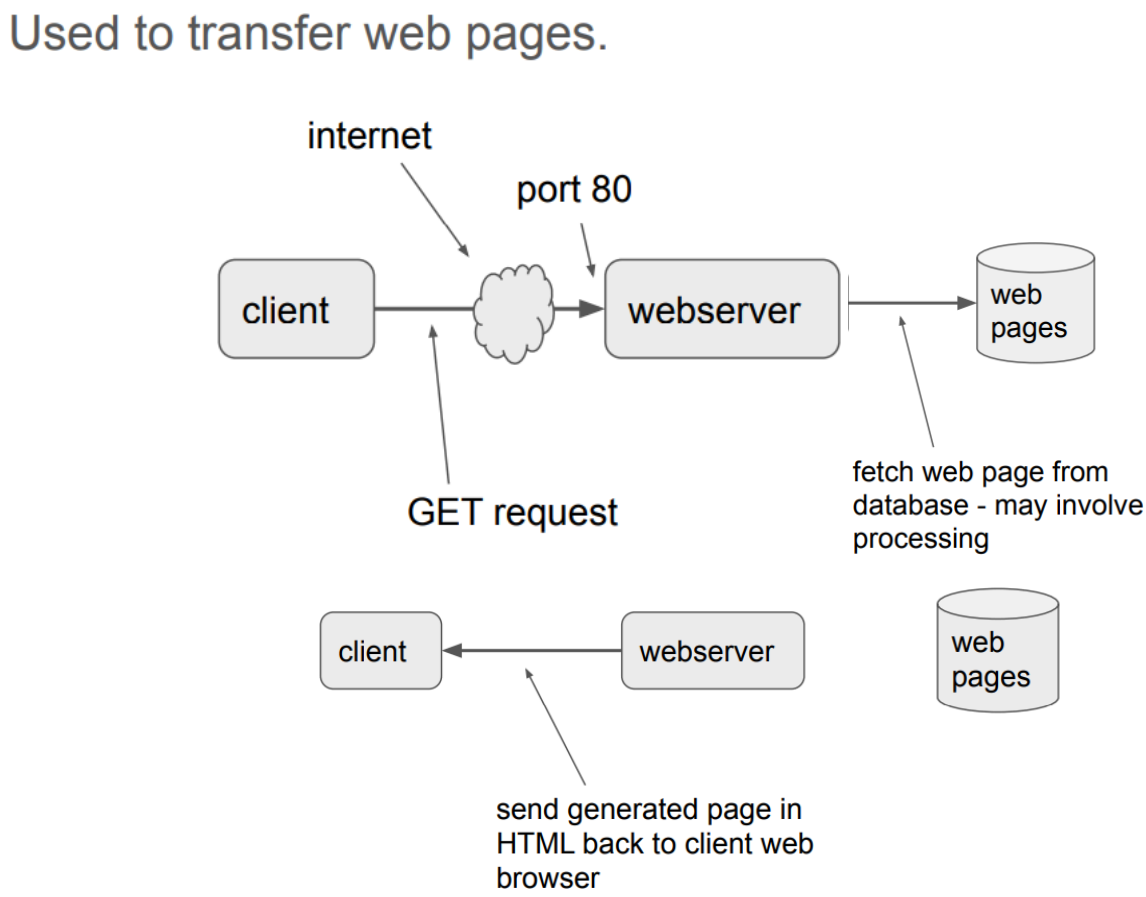


Port Numbers

- Port numbers are 16 bits so there are 64,000 possible numbers. Many of these are reserved for particular protocols but a large number remain for general use.

- In particular, the web protocol HTTP has port 80 reserved, and the email protocol SMTP has port 25 reserved.

HyperText Transfer Protocol – HTTP



Client:

```
GET /index.html HTTP/1.1
```

Server:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html>
<head>
<title>An Example Page</title>
</head>
<body>
    Hello World, this is a very
    simple HTML document.
</body>
</html>
```

Request

Header

blank line

Body

Simple Mail Transfer Protocol – SMTP

- STMP is also text line based with a small set of simple commands.
- An email client is fairly easy to develop. The client first determines the mail server for the destination domain using the DNS MX record. It then makes a connection to the mail server and the server will send an

initial status message, the client replies, the server responds and so on with the mail item included in the exchange (S: server, C: client).

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.com
S: 250 smtp.example.com, I am glad to meet you
C: MAIL FROM:<bob@example.com>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.com>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 January 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice.
C: This is a test message with 5 header fields and 4 lines in the message body.
C: Your friend,
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```

Email Format

- The format of email is another standard. It consists of a header and body section, much like the HTTP response message. The header has a series of lines with “keyword:” at the beginning followed by a blank line. The body is any text.

INFO1112 Lecture 10 Lecture – Network Security

- For network connections, we wish to provide:

  → Confidentiality: protecting information from interception, and ensuring that information remains confidential if intercepted.

  → Integrity: detecting whether information has been tampered with.

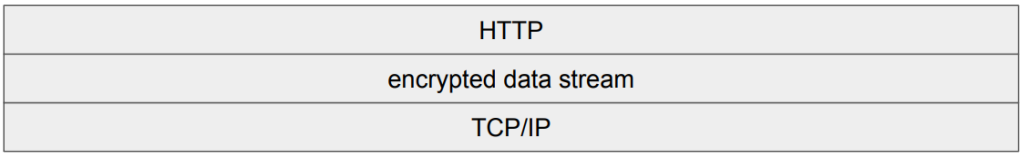
  → Authenticity: ensuring we know who the sender is, non-repudiation (some way of checking who sent something)

Security When Browsing a Web

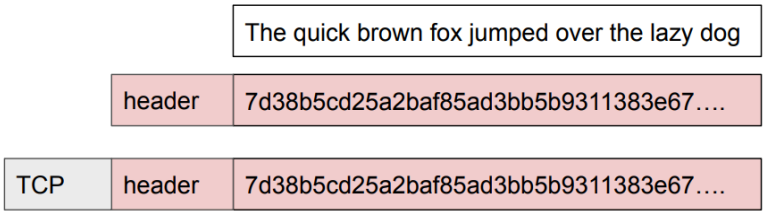
- We need confidentiality when browsing the web, otherwise an attacker may be able to monitor our traffic and get access to usernames and passwords for services such as banking.
- The HTTP web protocol has a secure version called HTTPS. In fact, it is the same protocol but the data on the transport layer connection (TCP/IP) is encrypted.

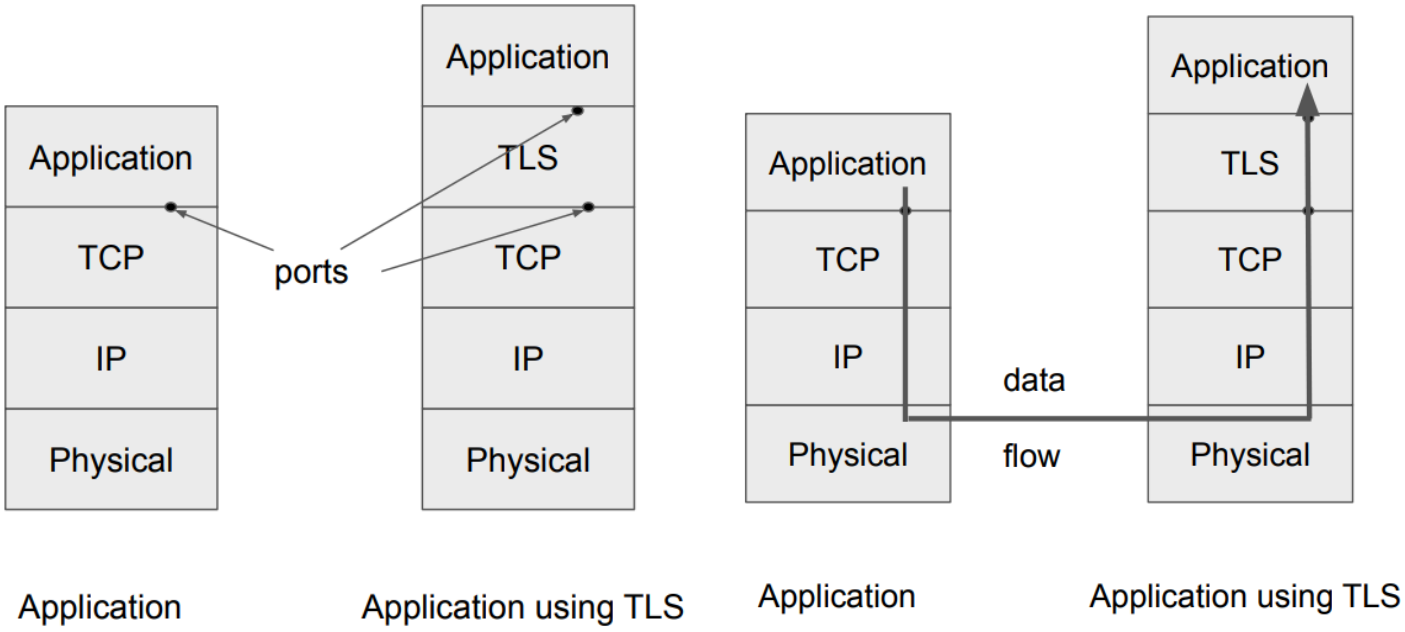
Layering

- Using the principle of layering, we can provide security for existing protocols such as HTTP.



- In the encryption layer, we scramble data in such a way that it can't be understood by an attacker, but the intended recipient can un-scramble it to get the original data.
- We do the "scrambling" using cryptology → TLS is the transport layer security port.





Cryptography

- Cryptography is used to guard against interception of communication.
- A message m is to be sent from Alice to Bob → Carroll may be able to observe the communication channel.
- Alice converts m to an encrypted version using an encryption function E(m); Alice sends E(m) on the channel; Bob receives it and applies a decryption function to calculate D(E(m)); the resulting message is the original one, m.
- m is called the plaintext, E(m) is called the ciphertext → the intention is that Carroll may see E(m), but they cannot find m.

Codes and Keys

- Usually, encoding is done by an algorithm with a parameter k, called a key (or password, passphrase).
    → It is difficult to keep algorithms secret, but easier to keep keys secret.
    → The key k is a shared secret, known by Alice and Bob but not known by Carroll – can be easily changed.
    → We have to assume that Carroll knows the algorithm.
- Example algorithm: shift all letters to the left by n, the key is the amount of shift, n.
- Cipher text is E(m, k) while the decoder is D(c, k) where c is ciphertext.
- The main attack against this is try to find out the key.

Using Encryption

- To protect a web transaction e.g. sending your banking ID and password, we can encrypt the data stream with a key that the user knows and the server knows.
- The problem with this is that it is difficult to set up a secret key for a casual transaction that both the user and the server know.
- We can't send the key over the connection without first encrypting it and therefore we need a secret key.

Message Integrity

- To check if a message has changed, we can calculate a type of checksum called "hash" or "message digest". This is a short value (e.g. 128 bits) calculated from a long message.
- Different messages should have different digests.
    → If the hash function is H, it is hard to find a message N for which H(N) = H(m).
    → A good H(m) should have a "one-way function property" – the inverse function is very hard to compute.
- Example algorithms include the USA standard, SHA-256.
- Message digests are used to check the integrity of a message → python provides a library module for this, hashlib.

Public-Key Cryptography

- Traditional ciphers depend on the key being known to Alice and Bob but not Carroll.
    → They are called symmetric or shared-secret codes – both E and D functions make use of the same key k.

→ It is hard to arrange safe sharing, especially between parties who are not in close physical contact e.g. between customer and ecommerce site.

- A different approach uses secrets which are only know to one party → public key cryptography, asymmetric encryption.

Alice wants to send Bob a message. Bob knows a secret or private key $k_{PRIV,B}$, and a related (but different) public key $k_{PUB,B}$

- Bob announces $k_{PUB,B}$ (eg on a web page) to everyone, including Alice
- Alice produces ciphertext using an encryption algorithm E with Bob's public key $k_{PUB,B}$ - ciphertext: $c = E(m, k_{PUB,B})$
- When Bob receives ciphertext c, he decrypts using an algorithm D with parameter $k_{PRIV,B}$
- A public key crypto system has the property that $D(E(m, k_{PUB,B}), k_{PRIV,B}) = m$
- Also, it must be difficult to work out $k_{PRIV,B}$ from knowing $k_{PUB,B}$

- Note that Alice doesn't need any secret information to send a confidential message to Bob!

Example Public-Key Algorithms

- The RSA system is the most famous.

→ invented by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT in 1977, although it had done before that by Clifford Cocks (British Intelligence Services) in 1973, but it was not disclosed to public.

→ Based on prime numbers and computer arithmetic (exponentiation) → it is easy to find prime numbers, and easy to multiply numbers, but hard to factor numbers that are not prime.

- Other systems based on more complicated number theory and mathematics:

→ Discrete logarithm system

→ Knapsack algorithms

→ Elliptic curve algorithms

- Implementations are slow.

→ Many thousand times slower than symmetric encryption algorithms although is it is not practical for most communications.

Authenticity – Digital Signature

- If D and E can operate in either order (RSA has this property), $D(E(m)) = m = E(D(m))$, then they can be used to "sign" a message.
- If Alice wants to prevent forgery of a message, combine messages m with $h = D(m, K_{PRIV,A})$.
- Bob can check that m corresponds to this signature, by seeing whether $m = E(h, K_{PUB,A})$.
- i.e. Bob can prove that Alice sent the message, since only Alice has the private key necessary.
- Combine signature and encryption to protect against both integration and forgery.

Key Establishment

- Public key crypto is much slower than conventional shared secret crypto. For any arbitrary pair of users to communicate securely and with trust using conventional crypto, we can do the following:

- Alice can choose a key value (can be random), and send it to Bob encrypted with Bob's public key and signed with Alice's secret key.

    → the key value is a secret they share: they both know the value, but the interceptor Eve doesn't know that.
- The shared secret can then be used in conventional symmetric cryptography.

Public Key Infrastructure

- How can we protect the information about Bob's public key from attack, and get it securely to Alice (establish trust)?
- Modification attack: if an attacker can change the website where Bob's public key is stored, and put on a different value there.

    → Attacker can then read messages intended for Bob.

    → Attacker could prevent Bob reading messages intended for him (denial of service).
- Bob can pass his public key to Alice through trusted intermediaries or bob can prove his identity to a mutually trusted third party, who issues a certificate that link the public key with their statement that they have been convinced of Bob's identity.

Digital Certificates

- A digital certificate is a data structure that certifies the ownership of a public key by a named subject.

    → this allows others to rely upon signatures or on assertions made by the private key that corresponds to the certified public key.
- Certificates are managed by 'certificate' authorities' that are trusted organisations. Certificates are signed by the secret key of the certificate authority and can be checked using the public key.
- To protect the public key of the certificate authority, a certificate is issued by a higher-level authority.

HTTPS Example

- When you request a web page using a URL starting with HTTPS, the web server sends a public key to the browser in the form of a signed certificate.
- This key is checked using the public key of the certificate authority (browsers usually store these locally).

SSH

- SSH or Secure Shell is a program available on all Unix systems for secure connection.
- SSH has its own communications layer above TCP/IP that provides encrypted login sessions and other services.
- The simplest use of SSH is to provide a confidential session between Unix systems. The authentication is handled in the usual way with the login program asking for a password.
- SSH can also be configured to use public key cryptography if a user stores their public key on the remote system and then uses their private key to encrypt a message for authentication.

SCP

- SCP or Secure CoPy is a feature of SSH that allows a user to securely copy a file from one machine to another.

- The user can use password authentication or public key authentication and SSH will send the file to the SSH server on the remote machine. A user can also copy a file from a remote machine securely.

SSH Services

- SSH is also used to provide a number of services where it is acting as a secure carrier for other protocols.
- For e.g. the X11 display protocol can be used over SSH and provide remote window sessions to Unix/X11 based systems.
- SSH can also be used for tunnelling lower level protocols. This is where SSH effectively acts as a pipe for any protocol and so can carry TCP/IP (so this is TCP/IP over SSH over TCP/IP) → useful for VPNs.

INFO1112 Week 11 Lecture – Android Architecture and Cloud Computing

History

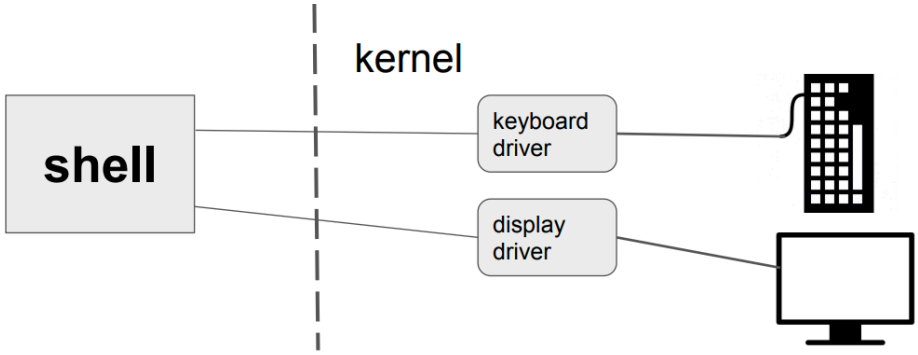
- Initially, Unix was purely command lined based in 1969. It didn't acquire a windows and mouse system until 1984.
- The original idea of a window-based system was created at Xerox Palo Alto Research Centre (PARC) in the early 1970s, and before that, the mouse was invented in 1964 at the Standard Research Institute. Researchers at PARC created a workstation called the Alto using windows and a mouse.
- In 1981, Apple produced a system called the Lisa.

Unix Windows

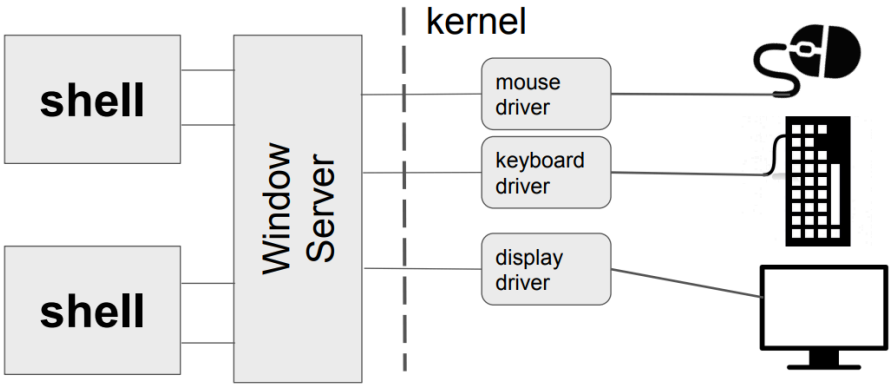
- The first and still widely used window system for Unix was 'X Windows", created in 1984 at MIT.

How Do We Do Windows?

- When you have a simple command line interface, the shell reads from the keyboard and writes characters to the display via the operating system kernel.

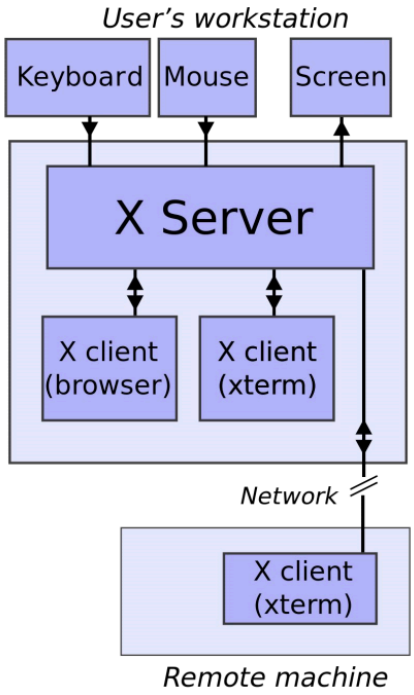


- For a multi-window system, the shell (or any other application) communicates with a "window server" that manages the display and keyboard and mouse.



- The window server is responsible for maintaining information about the whole display. It has data structures that represent all the display elements such as windows, scroll bars etc.
- The window server is a large complex system that can be further broken down into parts, one of which is a window manager. In Unix, there are many window managers to choose from, all with a different visual style.
- The same overall architecture is used for other systems such as MacOS, except another version of Unix is used rather than Linux. However, you can run an X window server under MacOS and then run X applications on a Max.

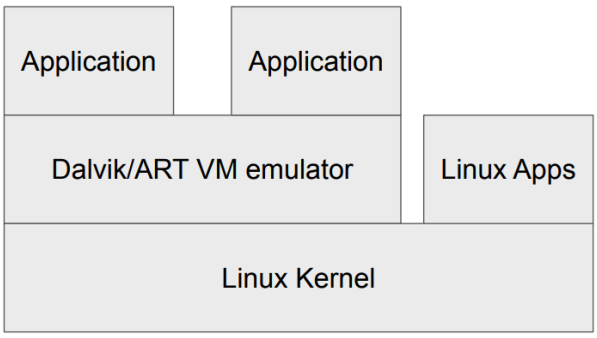
X Windows Architecture



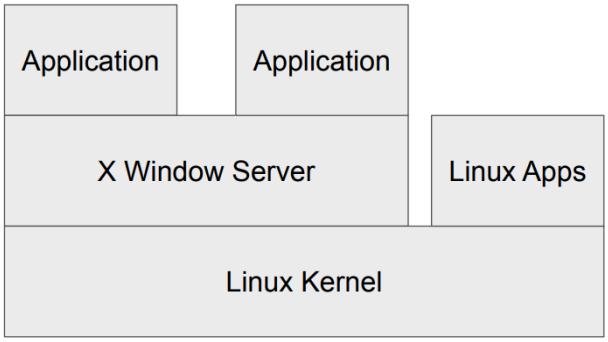
Android

- Android was originally designed to run on mobile phone devices. It is now used on various devices such as TVs, set-up boxes for TV, tablets etc → in 2019, there are 2.5 billion Android systems.
- The base of Android is Linux. It is estimated that the Linux kernel used in Android is 95% the same as a standard Linux kernel. The differences are mainly in device drivers for particular phones.
- Android also uses an extra software layer like the window server for applications to access the screen and touch input. This is similar to other window systems such as X.
- Android is written in the Java language. A powerful, higher level language that is probably the most widely used language (although Python is almost as popular).
- Java is translated form the high level language to a virtual machine code, which is then emulated by a program.
- In normal Java (non-Android), the virtual machine is called the "Java Virtual Machine" or JVM. While the JVM is open source , the software is owned by the Oracle corporation, (but originally developed by Sun Microsystems). If the JVM is used in commercial products, it has to be licenced from Oracle.
- Android initially used a different architecture and virtual machine to avoid this problem. The virtual machine code for Android was called Dalvik.

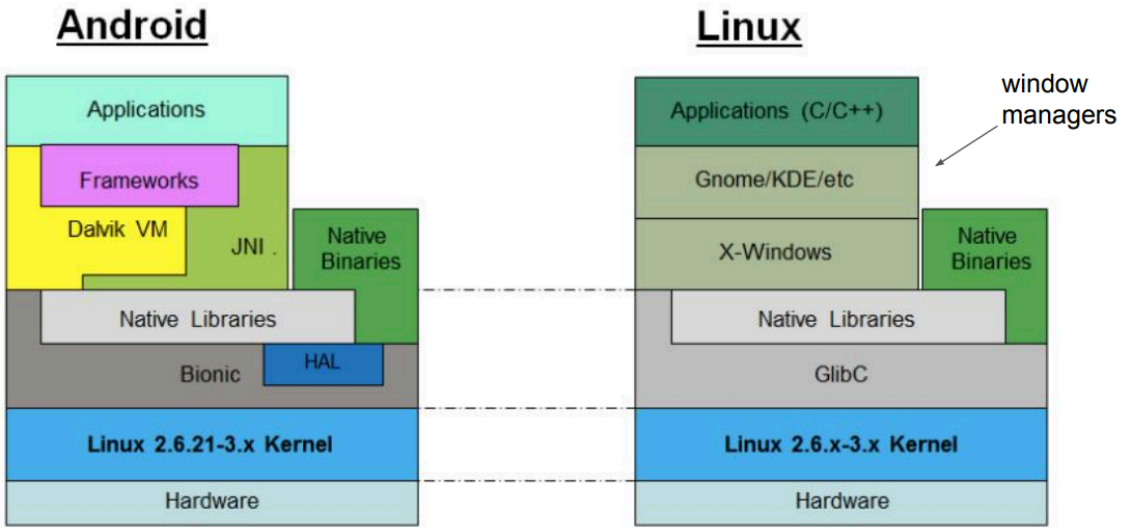
- Since all normal applications in Android are implemented in Java, the original architecture for Android looked like this:



- The equivalent architecture for X Windows looks like this:



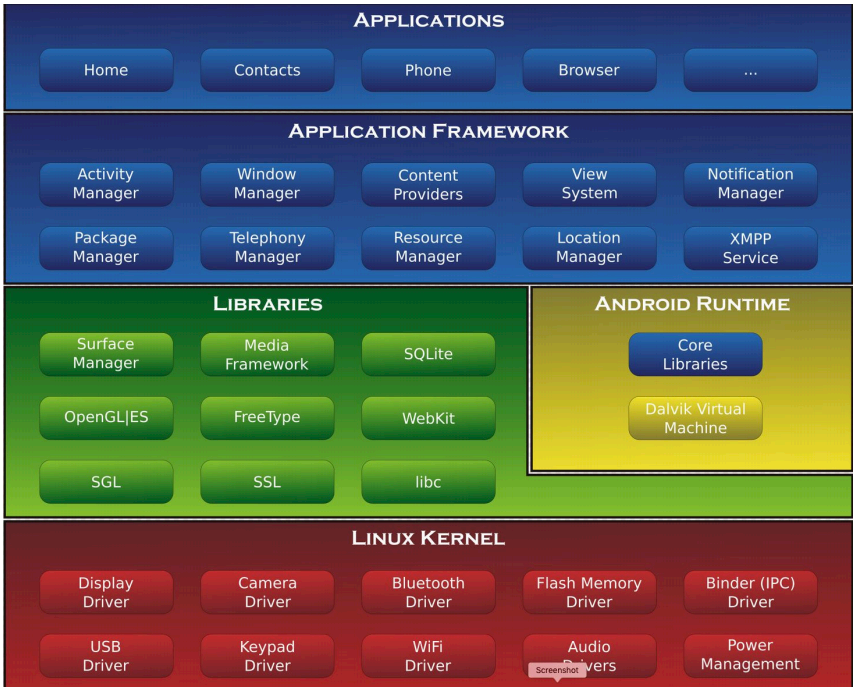
- This means that the display handling and touch screen input for Android is carried out in the Dalvik or ART (Android Run Time) virtual machine emulator.
- Just as X windows usually has a Window Manager that manages windows, Android also has “Frameworks” that manage the look and feel of the screen interface.
- In reality, things are a little bit more complicated, but this was the general structure.
- Today (2019), Android has moved away from the Dalvik VM and towards native binaries. This is to improve performance.
- The Dalvik bytecode is now translated into machine code and not interpreted.



Android Internals

- JNI in this diagram is the Java Native Interface. It enables Java code running in a Dalvik to call and are called by programs specific to hardware and Linux platforms and Linux platform and libraries written in other languages such as C, C++ and assembly.

- BIONIC is the diagram in an equivalent, but rewritten, copy of the standard C library glibc. This was also done due to licencing restrictions.
- HAL standards for Hardware Abstraction Layer and give a standard interface to screens and touch input. This is necessary because different phones may have different touch hardware.



Cloud Computing

- The cloud is a network of remote servers hosted on the Internet to store manage and process data, rather than a local server or a personal computer.
- Other features of the cloud are that applications can be rapidly deployed and scaled, and the cloud allows easy sharing of resources in an efficient manner.

Cloud Providers

- There are many companies providing cloud services. These include: Amazon AWS, Microsoft Azure, IBM BlueMix, Google.
- IN most cases, their cloud product grew out of in-house use of a cloud architecture to provide computing services for their own enterprise.
- Amazon was the first to do this and is the leader in providing cloud services.

How Does it Work? (Basic Cloud Service – Remote VM Running an Application)

- The cloud provider maintains a large pool of physical servers of various sizes (CPU, memory etc) and runs virtual machines on them.
- A user configures the type of machine they want via a web page, choosing options such as CPU power, memory, storage etc → the provider then "provisions" an appropriate VM running one of the physical servers.
- The VM may be the only one on that server or it might be one of many. For example, a very small VM might co-exist with many other small VMs on a large physical server.

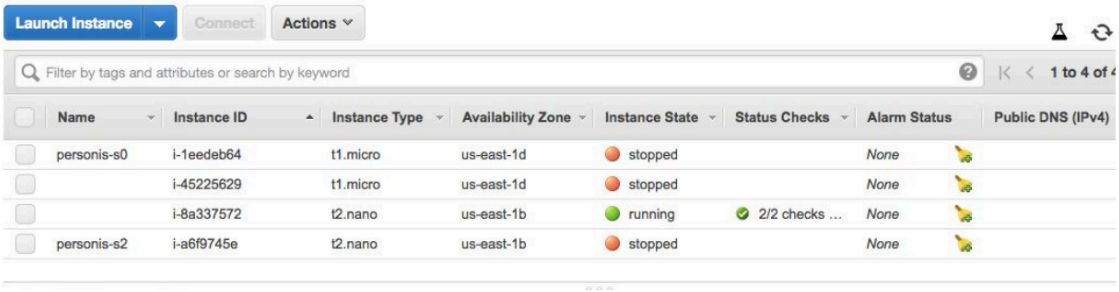
- Once the VM has been configured (including the operating system) and started, the user can remotely connect (using ssh or remote desktop etc) and finish configuration. Finally the user can start the application, eg a web server.
- A key service provided is the ability to manage your VM remotely via an API A cloud API, or Application Program Interface lets a programmer configure, start, stop etc. the cloud VMs from a program. This means that VMs can be initiated in response to heavier load, for example.

Amazon AWS

- AWS was one of the earliest cloud providers and has evolved dozens of services to make it easy to deploy the user's applications and services.
    - → Elastic Bean Stalk – configure and deploy applications
    - → S3 – simple storage service
    - → Databases → relational, NoSQL, etc.
    - → Domain name and IP address management etc.

Example EC2 Console

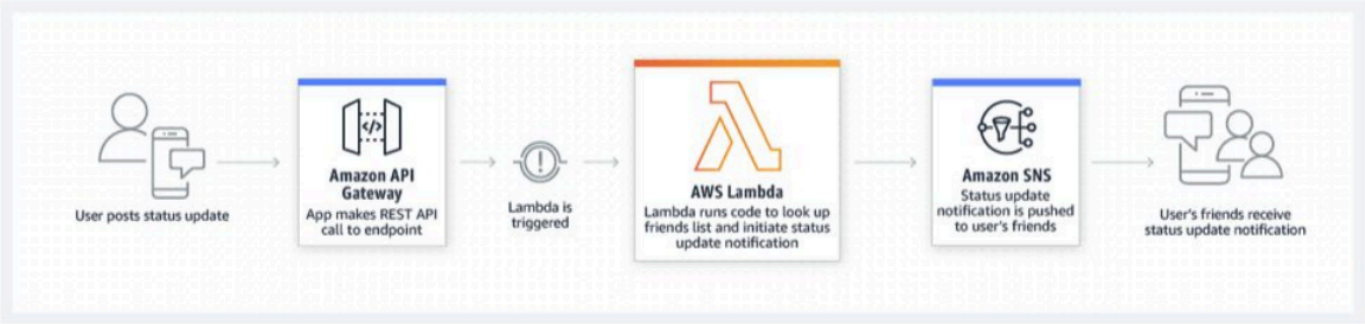
- The image below shows four VM instances, 3 stopped, 2 "micro" size and 2 "nano" size.



- The running instance has been running for about 5 years, but not continuously. It costs about $AU7/month to run a fairly light load.
- Twice in that time Amazon have notified that the underlying hardware was being upgraded and the instance needed to be stopped and restarted. Since it is a Virtual Machine, it could be stopped and restarted with no loss of state.
- The restarted VM was running on different, newer, hardware. The instance has also been restarted regularly to do security updates to the operating system and software.
- When the instance was first started there were less data centres available and this one was started in the data centre on the east coast of the USA. There are now many data centres around the world.

"Serverless" Computing

- As well as deploying a full VM to run applications, Amazon and other cloud providers have a so-called "serverless" option. This allows you to call a single program function (that may call many other things) and have a VM automatically deployed, run your function, and shut down (this obviously still involves a server).
- The Amazon service is called Lambda and can run code in many languages (Python, Java, C#, JavaScript).
- With this type of service you only pay for the CPU time you need (i.e. no idle time), instances are started and scaled automatically.



Storage and Database Services

- Cloud providers also provide ways of saving and retrieving your data. This can be a simple storage like Amazon Simple Storage Service, or a complex database.
- They also provide various grades of service. For example, Amazon provides a service for archival storage of very large datasets (Amazon Glacier) → It is very cheap but can take hours to retrieve a dataset.

Security Services

- A key requirement of cloud services is security.
- All the services are available via the internet and access must be controlled. Cloud services usually provide sophisticated security services to manage the access.
- In addition, they provide protections from malware and denial of service attacks. These services would be very expensive for a small user but are available at low cost from the cloud provider.

Internet of Things (IoT)

- A fairly new service in the cloud is the set of services to manage billions of devices in the so-called "Internet of Things" → refers to embedded sensors and processors in everyday devices such household appliances or cars.
- Many appliances now contain devices that monitor the device and send notifications. For example, a washing machine that sends a notification when the cycle is complete.
- It is estimated that there will be billions of such devices in the next few years. Each device has very limited compute power, so handing off the processing to the cloud is a logical service Amazon has a complete IoT handling service.

The Cloud

- Cloud computing is central to today's Internet. Without cloud computing it would be very difficult to build and scale modern web services.
- Centralised, private approaches would be very much more expensive and difficult to scale quickly. That approach also makes it hard for small companies with a new application to service growth.
- Cloud computing provides a complete range of services and computing power, all at a reasonable price.

INFO1112 Week 12 Lecture – Data Management

- Data are facts that can be recorded, are important for users, and need to persist past the point where it is used during a computation.
- Databases are a collection of data that is usually quite large and contains all the data needed to operate an organisation.
- Database Management Systems (DBMS) are software packages designed to store and manage one or more databases, allowing shared access from many programs at the same time or over a long span of time.

Categories of Organisational Data

- Personnel
- Inventory
- Orders
- Partners
- Transactions
- Tasks

Importance of Data

Enterprise

- The day to day operations and long-term planning of an organisation depend on accurate data.
- Data is a major asset for the organisation → consider the losses of missing or wrong data, and the cost to replace it.
- E.g. a supermarket chain needs data to pay workers, suppliers, decide when to offer discounts and what on etc.

Personal

- Data is important for many purposes (social, entertainment, communication etc.) e.g. calendar, profile picture, music, photos, messages, password storage etc.
- Some will be stored by the service providers in their own systems (and used to deliver services), some will be kept on your device for your applications to share.
    - → Service provider keeps data on many users.
    - → Locally, data is kept for many applications

Additional

- The state of a database mirrors the state of the real world.
- Data is shared between tasks and allows connections to be explored e.g. how employee characteristics influence productivity, or how your calendar can be used to choose appropriate music styles.

Managing Data

- The usual way to build an information system today is to write many application programs which all use a shared collection of data.

→ The data is stored in a DBMS.

→ For core activities in large organisations, the DBMS is usually a commercial one (Oracle, MS SQLServer, IBM, DB2 etc.)

→ For smaller organisations, or non-mission-critical parts of large organisations, often use Open Source systems (PostgreSQL, MySQL etc.) – really huge sets of data in large organisations often use these systems as well e.g. Facebook's data on likes.

→ Personal applications often use a small-scale DBMS (SQLite in every phone, many IoT devices).

- Users run application programs which access the data through the DBMS.

→ Programs can retrieve data and present it to the users.

→ Programs can update the data based on input from the users.

Meta-Data as Data

- Knowing how the data is structured (often called its schema) is essential to working with the data.

→ Data is useless if one can't interpret it, and over time, if a person leaves, who knows what a value of string means e.g. Alan Fekete,162,46,447.

- Schema and other meta-data should be stored with the data it describes.

→ There should be some facilities for asking about and updating the meta-data.

- A key idea in DBMS's is for the database itself to store descriptions of the format of the data, called the "System Catalogue" or "Data Dictionary". This sort of information is often called the meta-data.

"What" not "How"

- It is convenient to indicate declaratively what information is needed, and leave it to the system to work out how to process through the data to extract what you need.

→ Programming is hard, and choosing between different computations is hard.

- Users should be offered a way to express their requests declaratively.

→ A query language can be based on logic e.g. SELECT…WHERE.

Administrative Services

- The performance that users get, and the security of the whole system, can be greatly affected by the way the data is stored and the settings of system configurations → what helps one user may be bad for others, but some changes have wide benefits.
- The needs of the whole organisation can be considered by people who are responsible to the whole, rather than to individuals or subsets.
- A system should offer an interface for administrators to adjust parameters, structures etc.

→ One for users should offer an interface to provide hints that can help the administrators.

DBMS's

- Each DMBS supports some particular approach in which the users/applications can observe the data → the most common being the relational model, with others such as the key-value model, and the JSON (semi-structured) model.

Relational DBMS's

- Most DBMS's today store data that follows a relational structure.
- All data is seen by users as tables of related, uniformly structure information.

  → No duplicate rows where the order is not important.

  → Each entry is simple: integer, string, etc.

  → Matching values in different tables indicate connections.

Supplier:

SuppID	Name	Phone
8703	Heinz	0293514287
8731	Edgell	0378301294
8927	Kraft	0299412020
9031	CSR	0720977632

Product:

ProdID	Descr	SuppID
29012	Peas with Mint, 400g	8731
30086	Peas and Carrots, 450g	8703
31773	Salted Peanuts, 500g	8731

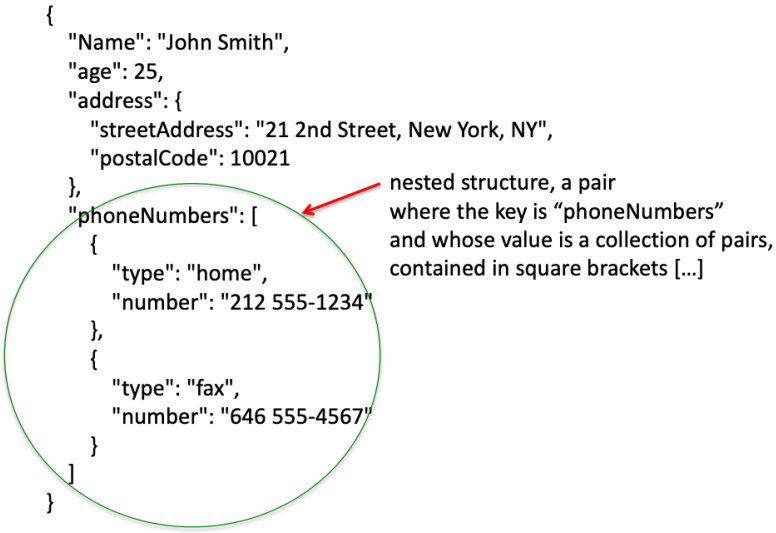
JSON DBMS's

- Some DBMS's keep data in a tree-like structure called JSON.

  → Very common format for data exchange between web sites.

  → Represents a fact as a pair with value and its meaning/key.

- These sets of pairs can be nested, and arranged in either ordered or unordered collections.



Instance

- An instance is the contents of the database at a single time.

  → Specific values, which describe a specific situation in the world.

  → Every update changes the instance.

Schema

- The schema in relational databases describe what tables exist, what columns are called, and the type of each.
- The instance at any time must fit the pattern of the schema → the schema changes rarely, if at all.
- The schema can also include integrity constraints, which restrict the possible instances, which the DMBA must enforce these at all times e.g. typically in relational DBMS, declare a column or combination of columns that will be the primary key (guaranteed to be unique between the rows).

Languages

- DDL: Data Definition Language is used to define the schema. For the relational model, it allows one to tell the DMBS what tables exist, and what structure they have.
- DML: Data Manipulation Language is used to access the data. It includes commands to update or change the contents of the database, and it also has commands that can retrieve information from the database ("queries").
- For a relational DBMS, both DML and DDL are in SQL.
    → SQL is quite declarative in style (the programmer says "what is needed" not "how to find it").
    → SQL is a standard, but each vendor has variations, so one doesn't have complete portability.
    → Applications are written to call the DBMS through SQL commands.
- For key-value DMBS, the schema are typically so simple that DDL isn't needed; DML is put (key, value) and get (key).

- The *powerful* command
  SELECT – FROM – WHERE
  retrieves data (rows) from one or more tables of a relational database that fulfill a search condition
- Example 1:

```
SELECT SuppID, Phone
  FROM Supplier
 WHERE Name='Heinz'
```

- Example 2:

```
SELECT COUNT(*)
  FROM Product
 WHERE SuppID = 8703
```


Transactions

- A transaction is a collection of one or more operations on one or more databases, which reflects a single real-world transition.
- In the real world, this happened completely, or it didn't happen at all (Atomicity). Once it has happened, it isn't forgotten (Durability).
- Commerce examples/student record system e.g. transference of money between accounts, purchase of a group of products, registration for a class

Atomicity

- Two possible outcomes for a transaction.
    → It commits: all the changes are made.

→ It aborts: no changes are made.

- Transaction's activities are all or nothing. Furthermore, once an outcome has been reached, it doesn't change.

Rollback

- If the application code gets to a place where it can't complete the transaction successfully, it can execute ROLLBACK.
- This causes the system to "abort" the transaction → the database returns to the state without any of the previous changes made by activity of the transaction.

Access Control

- Different users or applications should have appropriate rights to see and/or modify data.
  - → e.g. you may grant friends the right to see some posts, while others are for family.
  - → e.g. only managers are allowed to update salaries.
- Some aspects are controlled in the application code, while others are enforced by DBMS.
  - → In relational DBMS's, each table can have its own access rights defined.

```
GRANT privilege_list
   ON table
   TO user_list
   [WITH GRANT OPTION]
```

- privileges: **SELECT, INSERT, DELETE, UPDATE, REFERENCES**

```
REVOKE privilege_list
   ON table
 FROM user_list
```

A Database without a DBMS?

- Can one have a database without a DBMS to store it in? Yes, the information can be stored in files which are accessed directly by all the programs that need to use the data.
  - → Files are good at keeping data for a long time.
- This was common for business data in the 1950's, and is still common in science today.
  - → E.g. geologists collects samples, and does analysis of chemical composition, and produces a file with the information.
  - → This file can be sent to other scholars as email attachment, or made available for download.

Comparison

Drawbacks of Using File Systems to Store Data

- Data redundancy and inconsistency: multiple file formats, duplication of information in different files.

- Difficulty in accessing data: need to write a new program to carry out each new task.
- No central authority: different programmers might want different choices of files and formats, and there is no easy way to enforce organisational control over the valuable data.
- Integrity problems: integrity constraints (e.g. account balance > 0) become part of program code, and it is hard to add new constraints or change existing ones.
- Atomicity of updates: failures may leave database in an inconsistent state with partial updates carried out e.g. transfer of funds from one account to another should either complete or not happen at all.
- Concurrent access by multiple users: concurrent access needed for performance → uncontrolled concurrent accesses can lead to inconsistencies e.g. two people reading and updating a balance at the same time.
- Security problems.

Database Systems Offer Solutions to All the Above Problems

- This comes at a price in performance for simple processing and also in money, for commercial platforms.
- Finally, DBMS's are targeted at the most common sorts of information; they often work poorly on specialised data (e.g. gene structures, images etc).

DBMS Architecture

- Most often, DBMS's runs as a separate process.
    → Often in a separate machine "server" that is dedicated for that purpose and administered centrally.
    → Sometimes, a whole dedicated cluster of machines is used to run the DBMS software.
- Alternative: DBMS as a code library that is included in each application.
    → SQLite is an example.

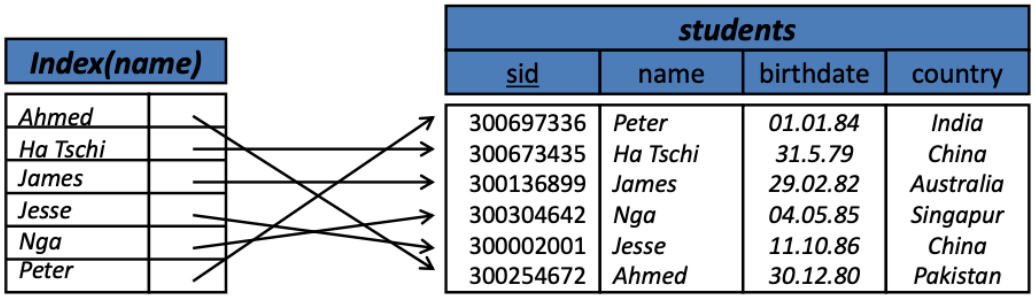
DMBS Design

- A DBMS is a very complicated piece of software → designs based on five decades of academic research and industry innovation.
- Typically a DBMS is structured as layers:
    → Data access layer.
    → Query processing layer.
    → Services on the side for security, transaction management etc.

Access Structure

- The data must be arranged to use space efficiently, and also to allow efficient retrieval of individual fields.
- Many different designs are used in different platforms, while the most common design is to store data so the access for the particular contents of a key is very fast (primary key in relational DBMS).
    → tree structures or hash table.

→ SQL allows users to also indicate that an index is needed for fast content-based access to any particular column (or combination of columns).

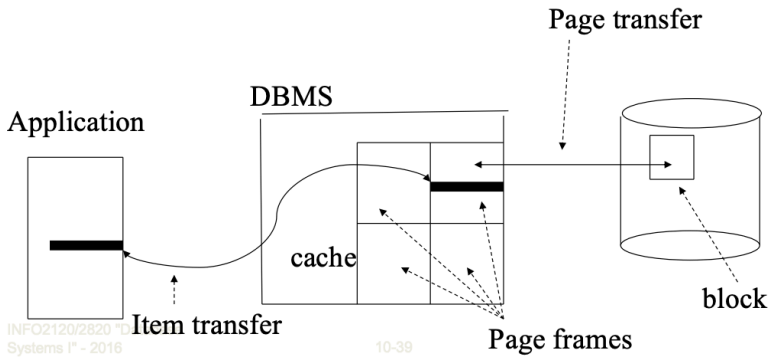


Storage Hierarchy

- DBMS's manage data in both volatile memory (DIMM etc.) and on persistent disk (HDD or SSD).
- Sometimes, the DBMS keeps some data on disk in one or more files, and works through the OS to access them.
- Often, the DBMS bypasses the OS and accesses the "raw disks" through disk drivers.
- Disks are accessed as block-at-a-time (e.g. 4K bytes).
- Most common: database physical structure is arranged in pages (often equal to a block, or several blocks), with the disk holding an authoritative copy.
    - → Memory holds a "buffer" cache of recently used pages.
    - → To speed up access if/when the data is needed again.
    - → DBMS controls the movement between the buffer and disk.

Storage Access

- A database file is partitioned into fixed-length storage units called data pages. Pages are units of both storage allocation and data transfer.
- Idea 1: Store related information close together on disk ('clustered').
- Idea 2: Database Buffer: Database system seeks to minimise the number of block transfers between the disk and memory.



Query Optimisation

- DBMS takes query from the user/application and uses knowledge of physical structures to find an efficient way to compute the result.
    → e.g. use an index if possible, to avoid scanning all the rows of a large table.
    → e.g. avoid comparing every pair of rows from two tables; instead, use sorting or hashing or index to find only the matching rows.
- There are many ways to compute the same result. The DBMS can estimate the cost of an approach (using statistic about the instance e.g. number of rows, size of each row).
    → Typically, the cost measure is taken as "number of blocks transferred between disk and memory".
- DBMS can identify many approaches for a given query, estimate the cost of each, then choose the lowest estimated cost as the way to perform the query.

Query Processing

- Once a query plan is chosen, it gets executed.
- Typically, the execution is pipelined so many steps are done on each record in turn (while it stays in registers).
    → sometimes, one needs to put several partial results into memory, before resuming calculation e.g. to find the average of several values.

Exploiting Parallelism

- On modern hardware, with multiple cores, one wants to keep them all busy but also to keep data locally on one core's cache memory (to avoid costs of synchronisation).
- Ways need to be found to have the DBMS use multiple threads → running different queries and steps of the same query.

Transaction Management

- Support for transaction capability depends on several interwoven system components.
- They also interact with design choices for buffer management and query processing.

Logging

- The log is an append-only collection of entries, showing all the changes to data that happened in the order as they happened e.g. when T1 changes quantity in row 3 from 15 to 75, this fact is recorded as a log entry.
- Log also shows when transactions start/commit/abort.
- Log must be frequently flushed to persistent store (e.g. disk) in particular, before commit.
- Log allows transaction rollback and also durability (recovery after system crash).

Locks

- DBMS's have a 'lock table' which remembers which transactions have set locks on which data, and in which modes.
- Locks are requested by the system itself when needed or requested explicitly by application code (SQL "LOCK TABLE" statement).
- Lock requests that conflict with already held locks are delayed until the lock is available, blocking any progress by the transaction that needs the lock.