Computational Physics

What is Computational Physics?

Basic Computer Hardware

Operating Systems

Programming Languages

Problem solving environment

What is Computational Physics?

"Computational Physics is a synthesis of theoretical analysis, numerical algorithms and computer programming."

P. L. DeVries, Am. J. Phys. vol 64, 364 (1996)

Computational Physics is a tool for solving complex numerical problems in Physics.

Why do we need Computational Physics?

- Physics tries to describe how nature works
- Often we need mathematical equation

  (unless you are a poet or philosopher)
- Using equations we create models to describe nature
- Exact (analytic) solutions are very rare unless a model is a simple one

Why do we need Computational Physics?

- Therefore we need computational physics when :
- ✓ we cannot solve the problem analytically
- ✓ we have too much of data to process

Many, if not most, problems in contemporary physics could never be solved without computers.

Computational physics in contemporary physics

- **Numerical calculations:** solutions of well defined mathematical problems to produce numerical solutions. Ex. Differential equations, integrations,
- **Visual animations:** the human eye and the visual processing power of the brain is a very sophisticated tool. Ex. 2D & 3D plots, animations, colour schemes & textures
- **Computer simulations:** testing models of nature. Ex. Weather forecast
- **Data collection and analysis:** in experimental research
- **Symbolic manipulation:** Ex. Mathematica, Maple

Classification of Computer Models

- **Deterministic or Stochastic Models**
  - Deterministic Models: Outcome of deterministic models depend on initial conditions
  - Stochastic Models: an element of randomness exists
- **Dynamic or Static Models**
  - Dynamic Models: changes in time
  - Static Models: does not change in time

Computer simulations (few examples)

- Molecular Dynamics simulation
- Weather forecast
- Design of complex systems (aircraft,..)
- Financial markets
- Traffic
- Games

More...

- Many natural phenomena are non-linear, and a small change in a variable might produce a large effect.

  But just few non-linear problems can be solved analytically.
- Systems with many variables or many degrees of freedom are interesting.

Millennium Simulation – Largest N-body simulation carried out thus far (more than 10^{10} particles)

Millennium Run

- The Millennium Run used more than 10 billion particles to trace the evolution of the matter distribution of the University of size 2 billion light-years.
- It took the principal supercomputer at the Max Plank Society's Supercomputing Centre in Garching, Germany more than a month.
- By applying sophisticated modelling techniques to 25Tb of stored output, scientists were able to recreate evolutionary histories for 20 million or so galaxies and for the supermassive black holes which occasionally power quasars at their hearts.

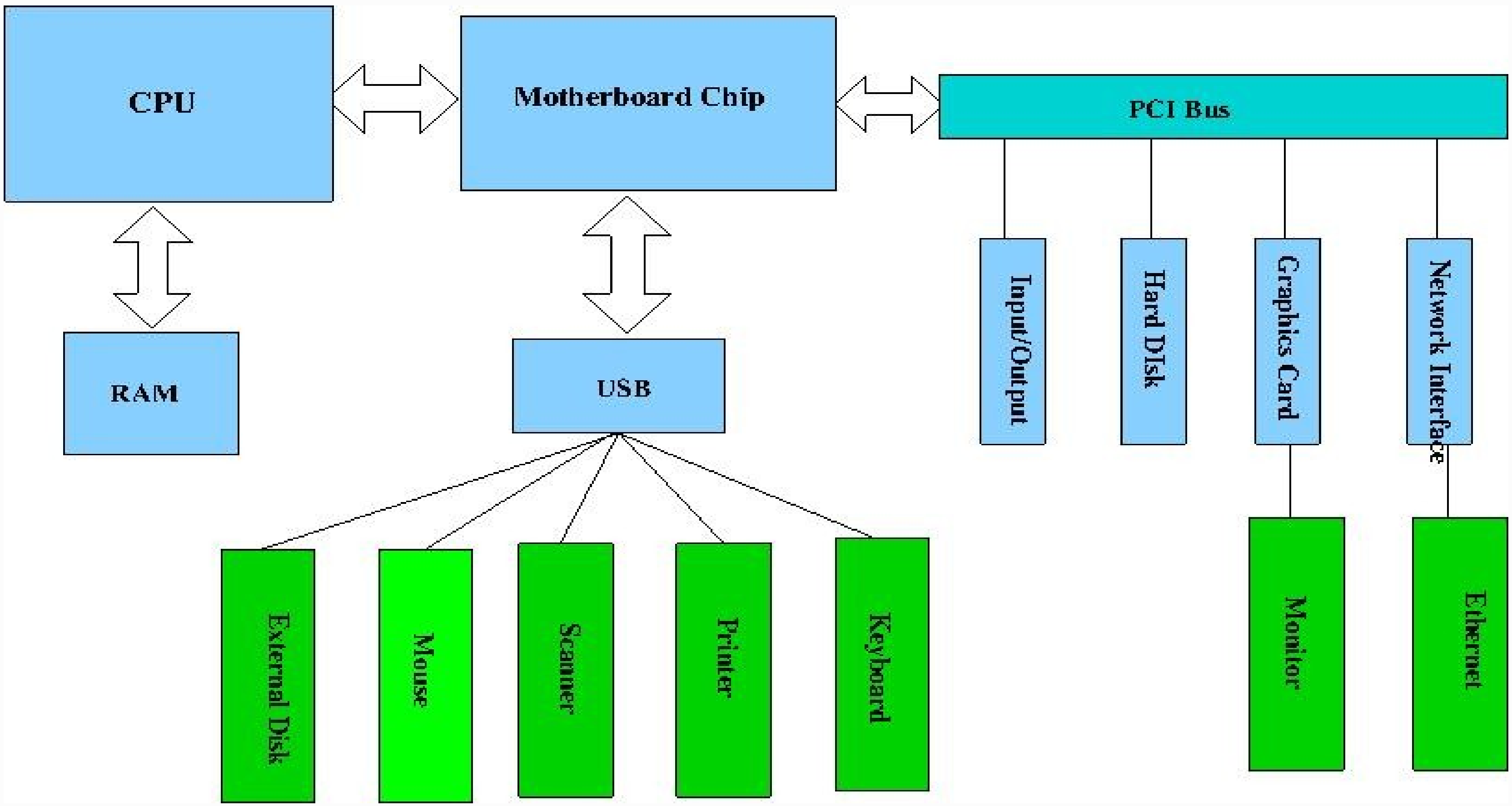
Computer Basics

- Hardware – Amazing progress. Twice processing power in 18 months. (Moore's Law: density at min. cost of transistors on IC's doubles every 2 years)
- *Do we have twice more results in Physics every 18 months?*

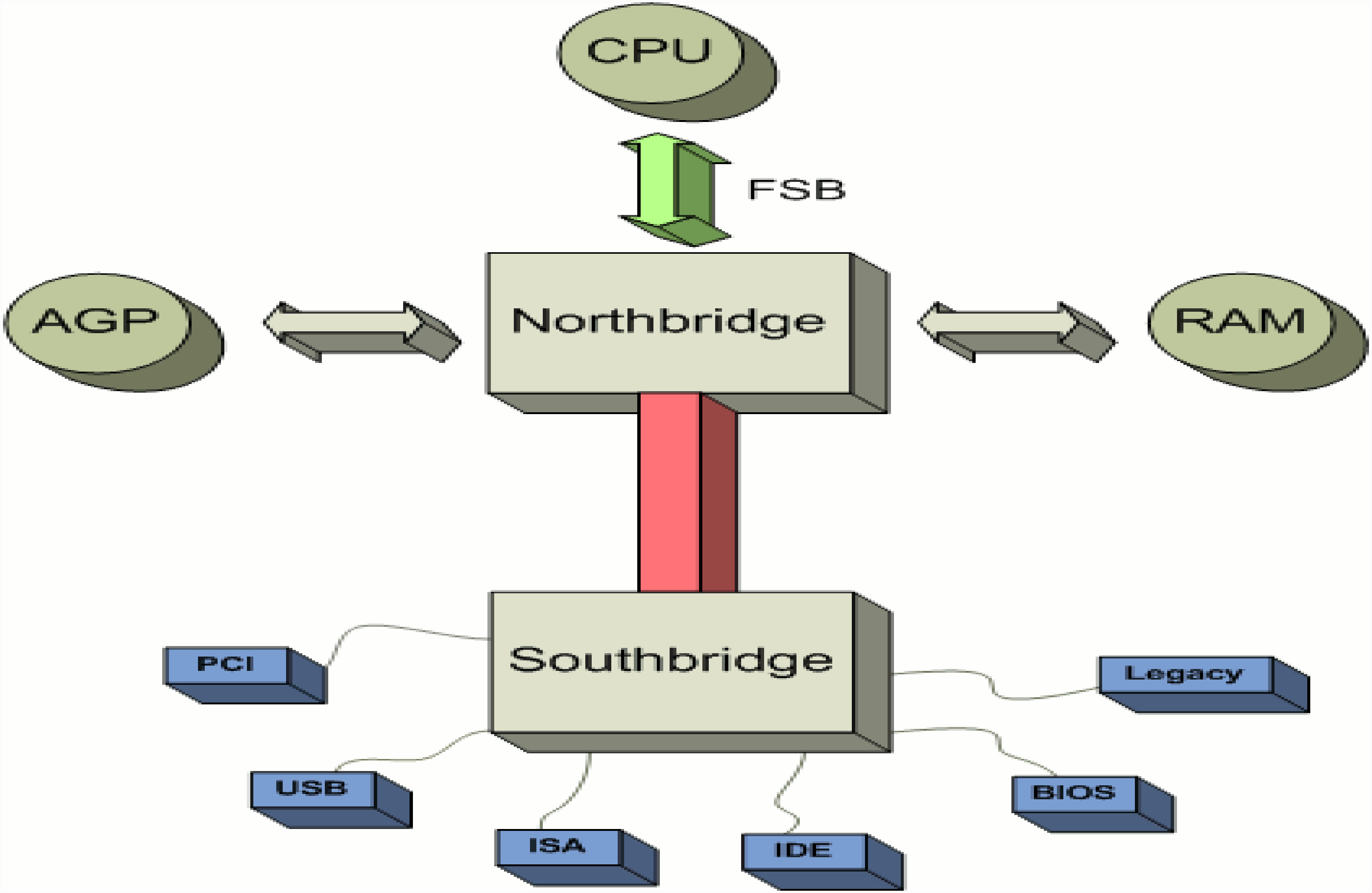
Computers in computational physics

- **Desktop Computer** *(OS: Linux/Unix, BSD,..)*
- **Clusters** *(OS: Linux)* – set of connected computers that work as a single system
- **Supercomputers** *(OS: Linux/Unix)*

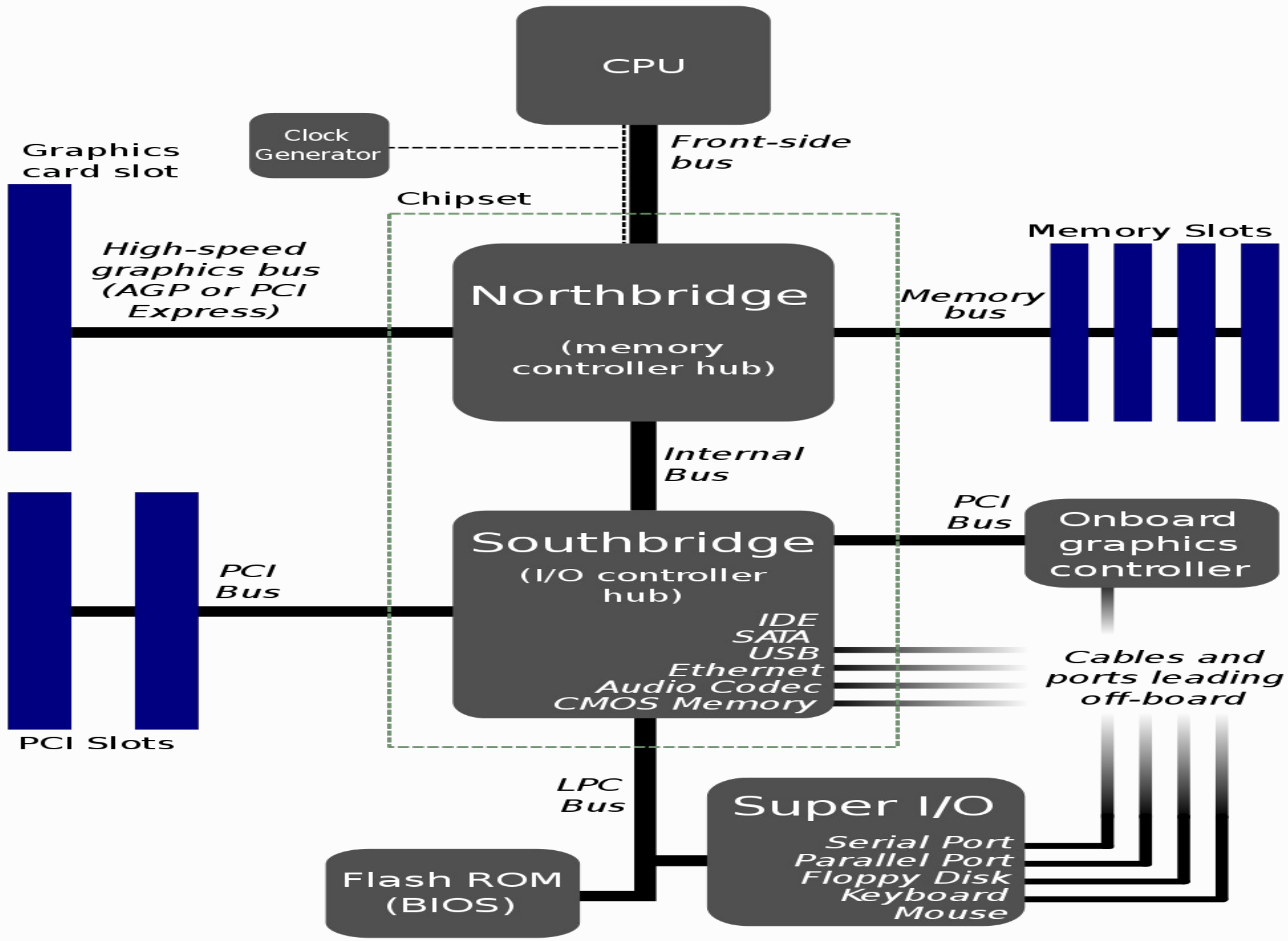
Basic Computer Hardware



Northbridge/Southbridge Layout



Motherboard



Hardware (internal)

- CPU – Central Processing Unit (in GHz), cache memory – cache 1, cache 2
- RAM – Random Access Memory (in GB or MB) communication with CPU by bus (MHz)
- PCI – Peripheral Component Interconnect
- USB – Universal Serial Bus
- HDD – Hard Disk Drive
- Graphics Card
- Network Interface (GB/s or MB/s)

Hardware (peripheral)

- Keyboard (I/O)
- Mouse (I/O)
- Printer (I/O)
- Monitor (Graphics Card)
- Ethernet (Network)
- Scanner, external storage, ..

Critical Hardware components for computations

- Desktops

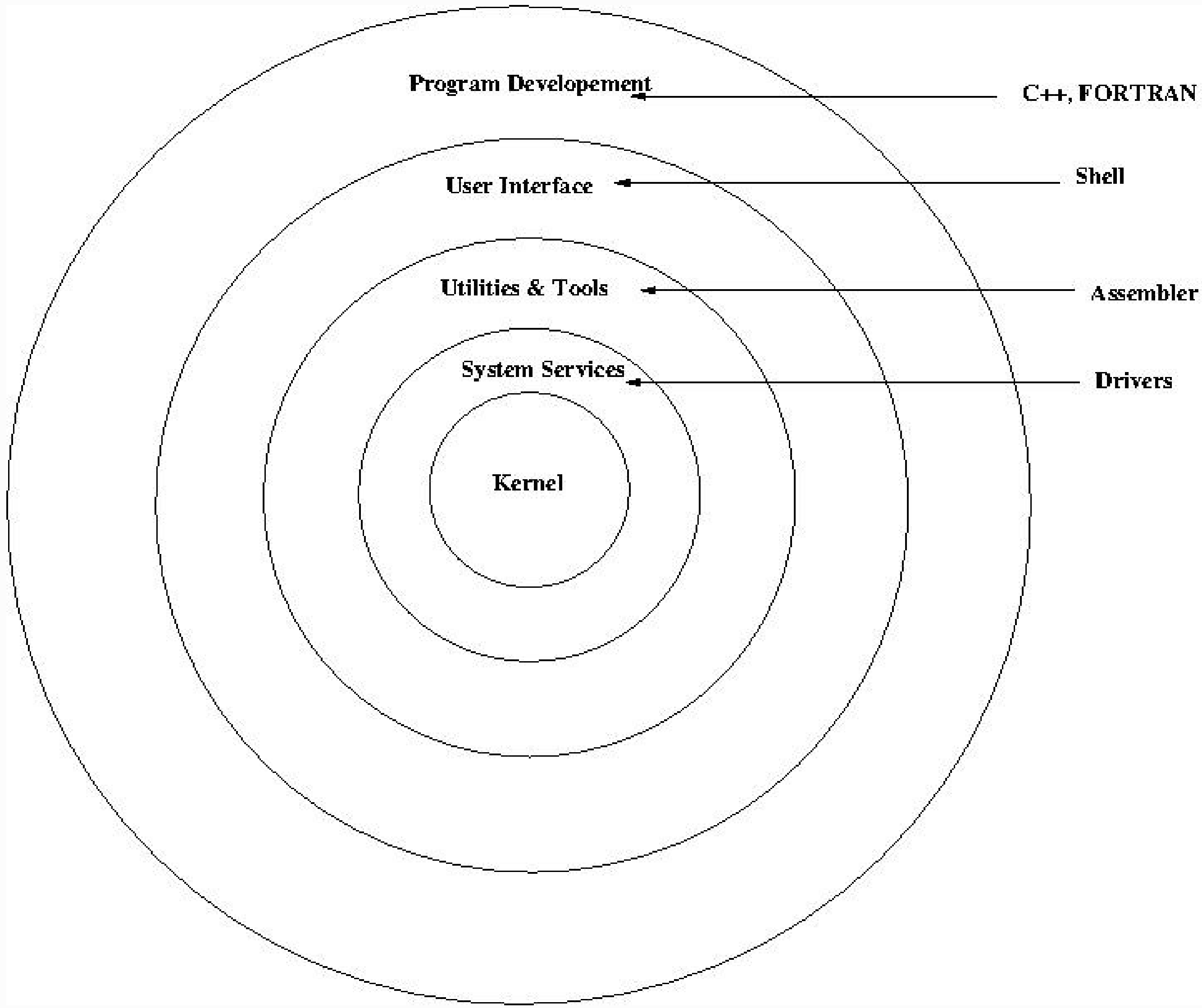
  CPU, RAM, FSB (Front-side bus) speed
- Clusters

  CPU & RAM

  No. of CPUs

  Fast communication between nodes

Software



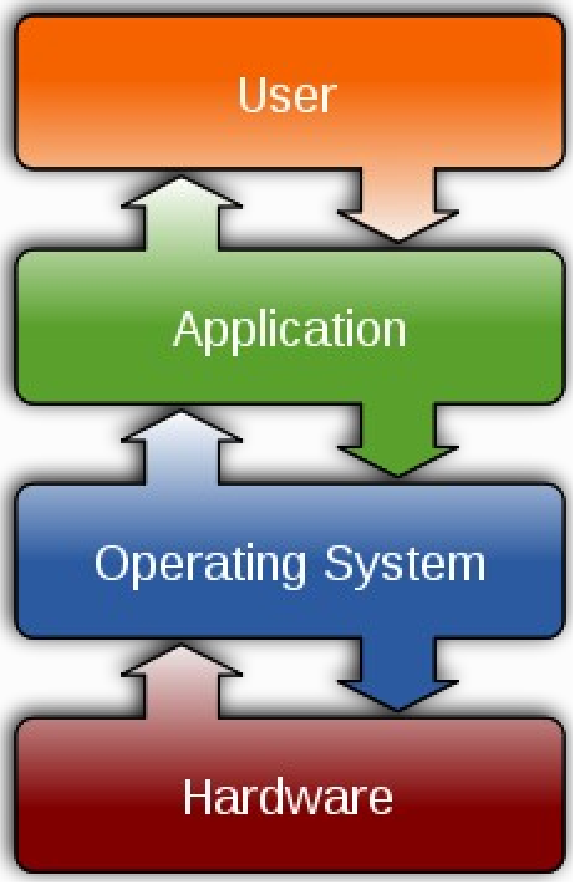
Program Developement
C++, FORTRAN
User Interface
Shell
Utilities & Tools
Assembler
System Services
Drivers
Kernel

Software: Operating Systems

Operating system – common features:

- Process management
- Memory management
- Interrupts
- File system
- Device Drivers
- Networking (TCP/IP, UDP)
- Security (Process/Memory protection)
- I/O

Operating System



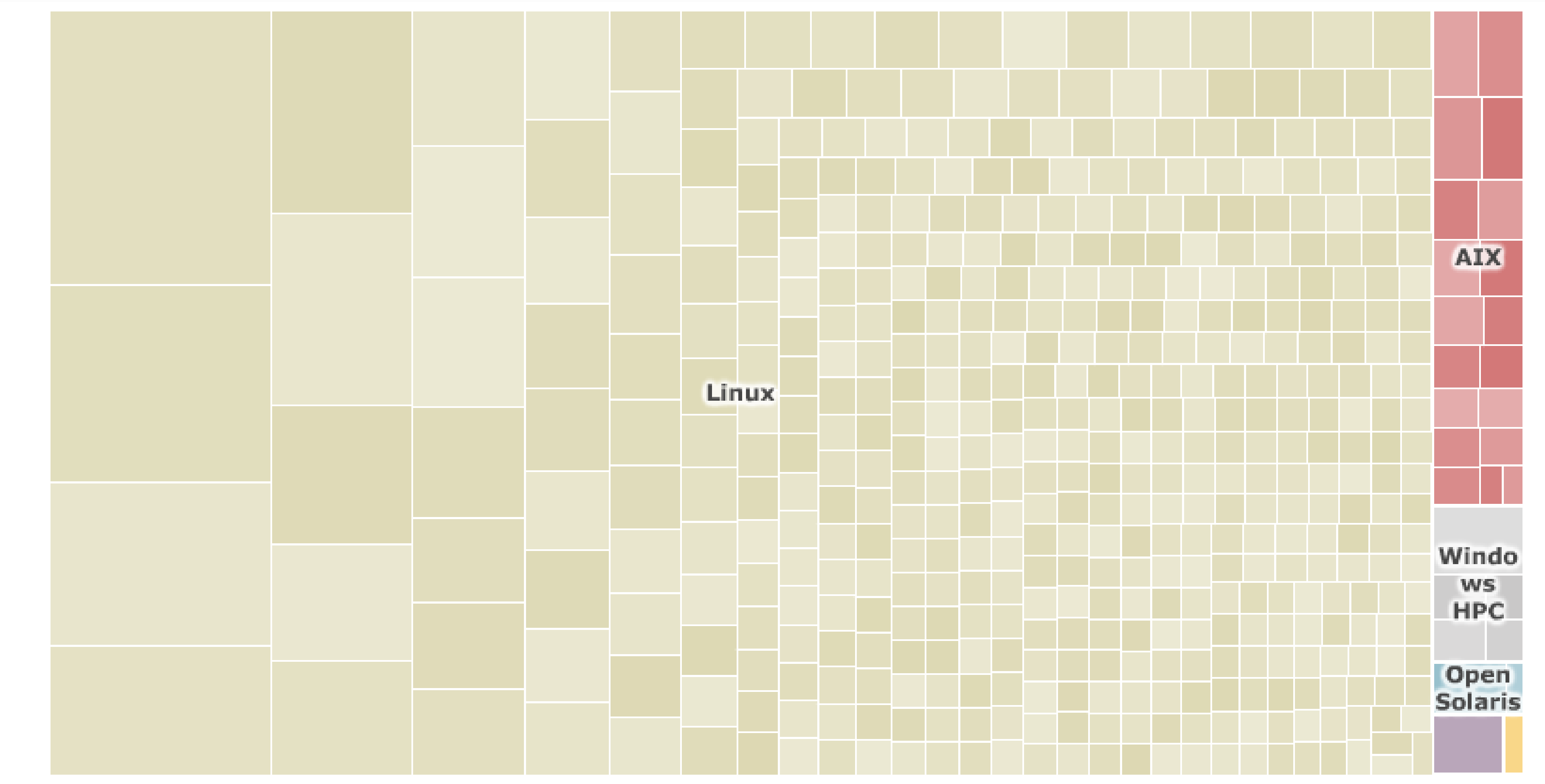
Types of Operating System

- **Multi-User:** Allows multiple users to access computer system concurrently
- **Multi-tasking:** Allows multiple programs to run concurrently
- **Multi-processing:** Supports multiple programs on more than one CPU
- **Multi-threading:** Allows different parts of a single program to run concurrently
- **Real Time:** Aims at executing real-time applications

Comparison of some popular OS

	Multi-user	Multi-tasking	Multi-processing	Multi-threading	Real Time	License
Linux/Unix	Yes	Yes	Yes	Yes	Yes (some distros)	GNU Public License (GPL)
Micro$oft Windows	No	Yes	Limited	No	No	proprietary
Mac OSX	No	Yes	Limited	No	No	proprietary

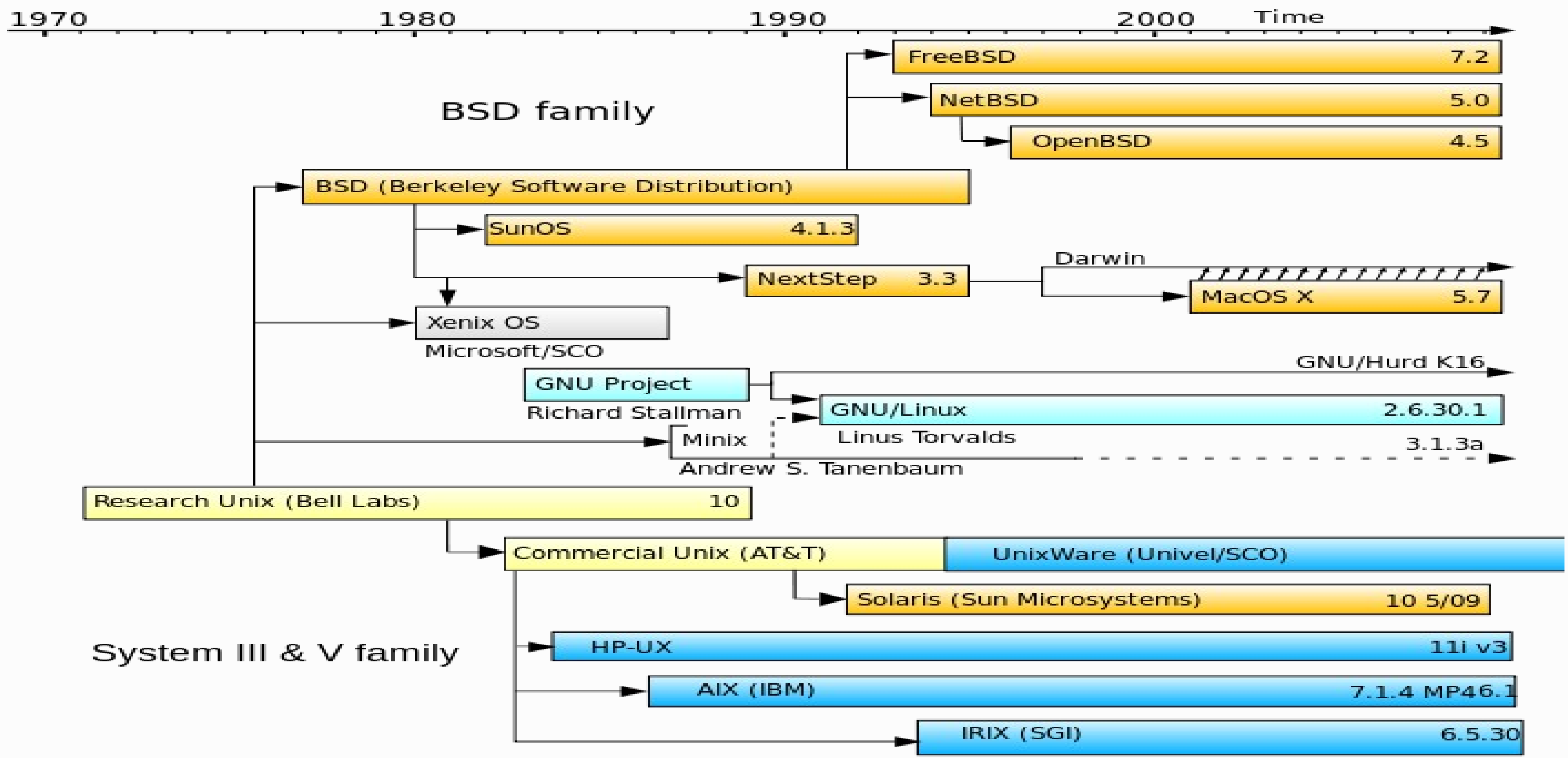
Supercomputer OS



Top 500 Supercomputers

	Linux	Unix	BSD	Mixed	HPC Window
#	462	24	1	11	2

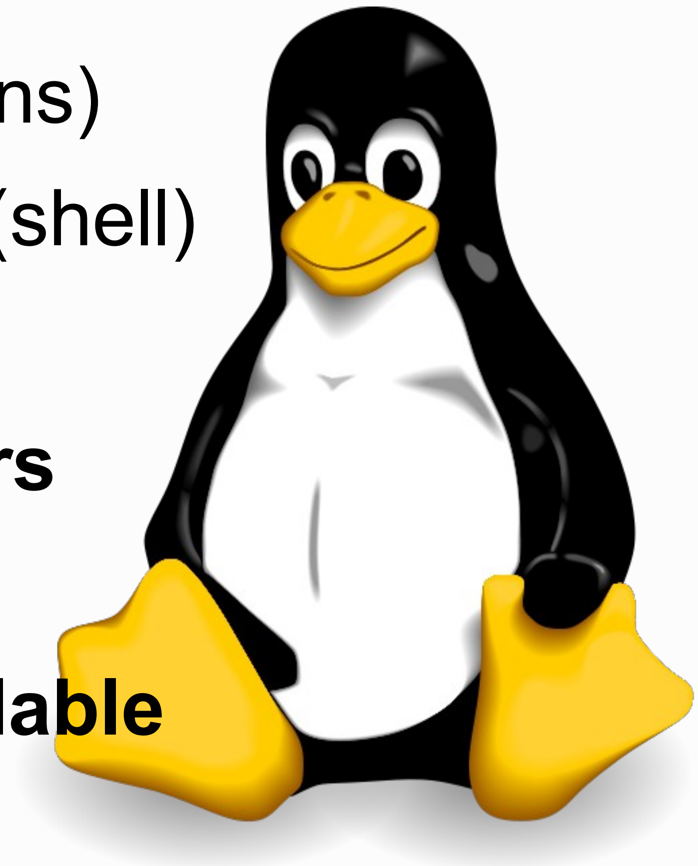
OS: timeline



1970
1980
1990
2000
Time
FreeBSD 7.2
NetBSD 5.0
OpenBSD 4.5
BSD family
BSD (Berkeley Software Distribution)
SunOS 4.1.3
Darwin
NextStep 3.3
MacOS X 5.7
Xenix OS
Microsoft/SCO
GNU/Hurd K16
GNU Project
Richard Stallman
GNU/Linux 2.6.30.1
Linus Torvalds
Minix
3.1.3a
Andrew S. Tanenbaum
Research Unix (Bell Labs) 10
Commercial Unix (AT&T)
UnixWare (Univel/SCO)
Solaris (Sun Microsystems) 10 5/09
System III & V family
HP-UX 11i v3
AIX (IBM) 7.1.4 MP46.1
IRIX (SGI) 6.5.30

GNU/LINUX: common features

- **Multi-user** (user accounts, multiple users logged in simultaneously)
- **Multi-tasking** (servers, daemons)
- **GUI** (X window system) & **CLI** (shell)
- **Hardware support**
- **Networking & Network servers**
- **Application support**
- **Robust, stable, secure & scalable**



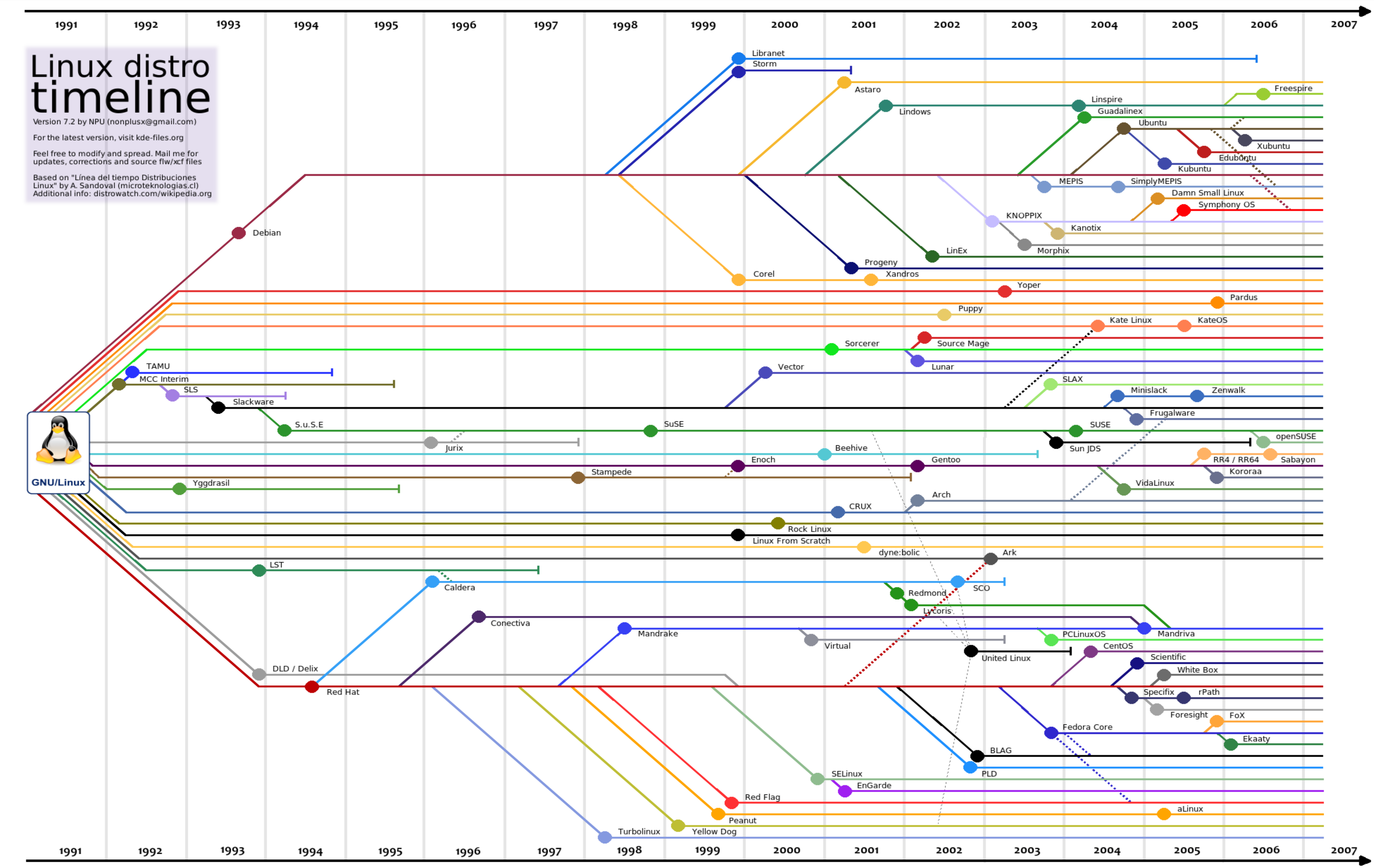
GNU/LINUX: brief history

- 1983 – **Richard Stallman** started the **GNU Project**. Goal to create completely “free” Unix-compatible software system.
- 1985 – Stallman started **Free Software Foundation** and wrote the **GNU Public License** (**GPL**) by 1989
- By 1990 most programs required in an OS was completed except the kernel
- 1991 – **Linux Trovalds** then graduate student at University of Helsinki, initiated work on **Linux kernel**
- Developers worked to integrate GNU components with Linux kernel to form a fully functional and “free” **GNU/LINUX** operating system

GNU/LINUX Distros

redhat	MEPIS	turbolinux	LUNAR	EvilEntity	debian	Vine Linux	cAos/CentOS	MiniKazit	UTUTO
archlinux	m0n0wall	jamd linux	Knoppix STD	gentoo linux	DeLi Linux	Hiweed	amlug	slackware linux	yellow dog linux
Fedora	LPG	PLD	SLAX	COREL LINUX	Progeny	GeexBox	BIGLINUX	FREEDUC	Lycoris
EnGarde	Mandrakelinux	BeatrIX	Linspire	SuSE	中文延伸套件	YOPER	BearOps	ASPLINUX	kalango
Slackintosh	Frugalware	Foresight	Mint	PCLinuxOS	Haydar Linux	sabayon	ubuntu	JULEX	blag

GNU/LINUX Distro timeline



GNU/LINUX and Computation

- Supercomputers
- Clusters
- Desktops
- Servers
- Compilers
- Applications

GNU/LINUX: basic use

- **Graphical User Interface** (X window system)

  Desktop Environments: Gnome, XFCE, KDE, LXDE,...

  Crtl + Alt + F7

- **Command Line Interface** (shell)

  Crtl + Alt + F1 to F6

Shell commands

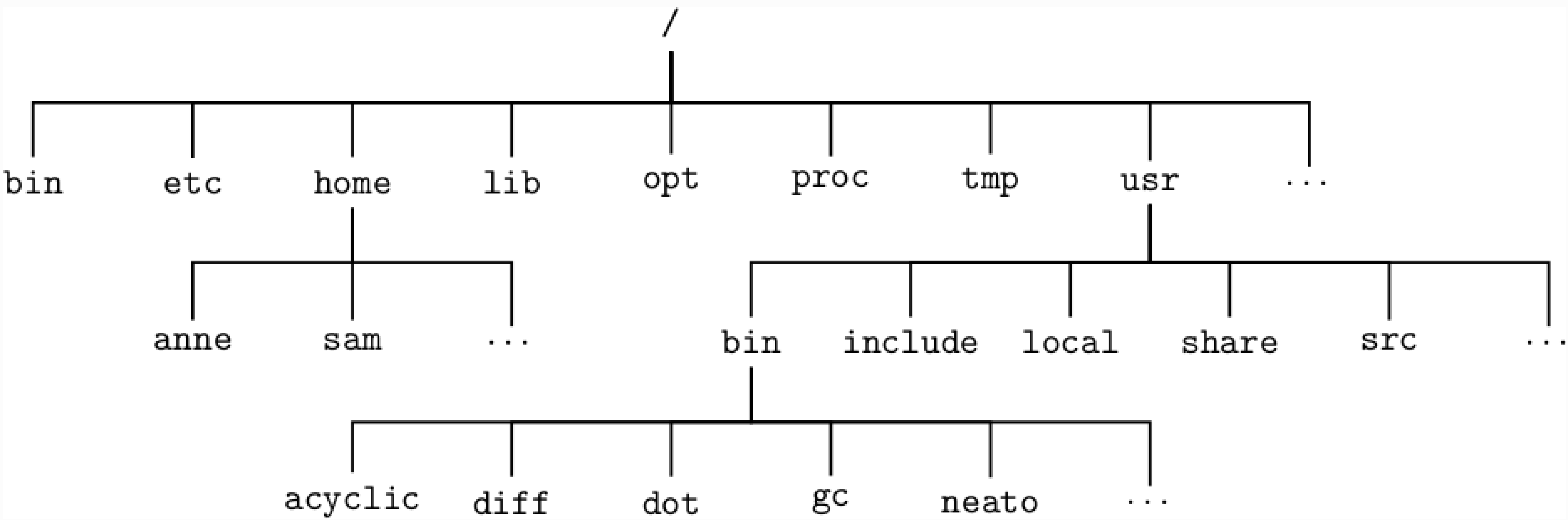
- Shell commands are case sensitive
- Getting help on some command:

  man ***command***
- Directory listing: **ls -a -l -h**
- Copying file: **cp -r -i** ***source destination***
- Moving file: **mv -i** ***source destination***
- Creating Directory: **mkdir** ***directory-name***
- Deleting file: **rm -i** ***file-name***
- Changing Directory: **cd directory-name**
- Changing file permission: **chmod ugoa +/- rwx filename**
- Changing password: **passwd**
- Exiting shell or logout: **exit**

File system hierarchy

- **/root** – root user's home directory
- **/dev** – essential devices
- **/boot** – boot loader files, eg. kernel
- **/etc** – system-wide configuration files
- **/proc** – virtual filesystem documenting kernel & process status as text files
- **/bin** – common Linux command binaries
- **/sbin** – essential system binaries
- **/lib** – libraries essential for binaries in /bin and /sbin
- **/var** – variable files whose content continually changes during operation, eg. logs
- **/usr** – user applications
- **/home** – users home directories
- **/media** – mount point for removable media, eg. cdrom, usb drive,

File system hierarchy



Read only file viewers

- **less file-name**
- **more file-name**
- **cat file-name** – concatenates file and prints to standard output
- **tail -f file-name** – outputs (& follows) last portion of a file
- **diff file-name1 file-name2** – compare files line by line

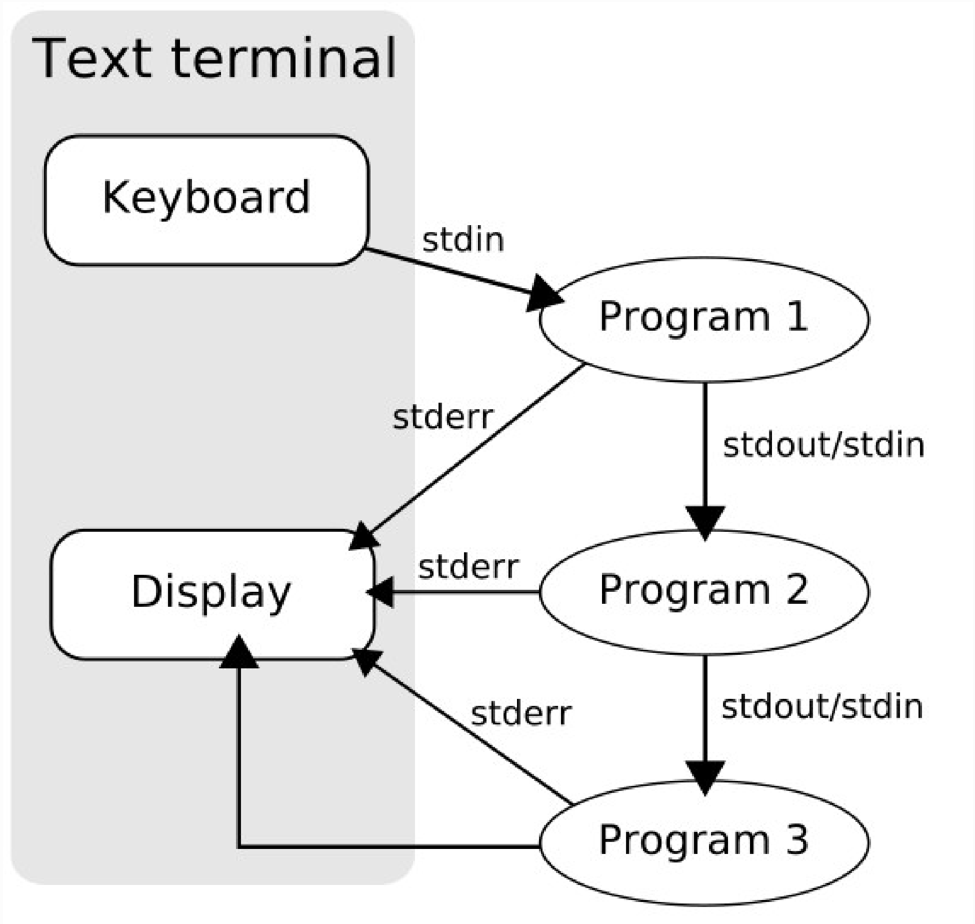
File editors

- **pico** – text based editor for beginners
- **nano** – text based editor for beginners
- **vi** – text based editor for advanced users
- **gvim** – GUI for vi editor
- **emacs** – graphical editor
- **gedit** – another graphical editor from Gnome

Anonymous Pipe

- Set of process chained by their standard streams
- Output of each process (stdout) feeds directly as input (stdin) to next process
- Each connection implemented by an anonymous pipe **|**
- By default standard errror streams (stderr) of the processes are merged and directed to the console, and not passed through the pipe
- Ex. **ls -al | grep file-name**

Anonymous Pipe



Named Pipe (FIFO)

- Uses filesystem, unlike conventional anonymous pipe
- Two separate processes can access the same pipe by name
- Explicitly created using mkfifo or mknod
- **mkfifo my_pipe**
- Ex. **ls -al > my_pipe**

  cat < my_pipe

I/O Redirection

- **command > filename**

  Writes the output of command to filename

- **command >> filename**

  Writes output of command to end of filename

- **command < filename**

  command takes input from filename

Shell

- Shell accepts commands and passes on to the kernel
- Shell is a command language interpreter
- Tip: to find all available shells in your system, type **cat /etc/shells**
- Tip: to find your current shell, type

  echo $SHELL

Shell script

- Sequential series of shell commands written on a text file

Why shell script?

- Useful to create your own commands
- Can take input from user, file and output them on screen
- Saves time
- Automates useful tasks
- System administration can also be automated

Shell script example

```
#
#
# Script to print user information who currently login , current date & time
#
clear
echo "Hello $USER"
echo "Today is \c ";date
echo "Number of user login : \c" ; who | wc -l
echo "Calendar"
cal
exit 0
```

Variables in Shell

- System variables – created and maintained by the operating system. Defined in CAPITAL LETTERS
- User defined variables (UDV) – created and maintained by the user. Defined in lower case.

System Variables

BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/xxxx	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

User Defined Variables

Variable name = value

Rules for variables

- Variable name must begin with alphanumeric or _ followed by alphanumeric
- No spaces on either side of =
- Case sensitive
- Special characters like ?,*, etc cannot be used
- NULL variable is defined as:

  VAR=

  VAR=""

Rules for variables

- Variable name must begin with alphanumeric or _ followed by alphanumeric
- No spaces on either side of =
- Case sensitive
- Special characters like ?,*, etc cannot be used
- NULL variable is defined as:

  VAR=

  VAR=””

Print variables

echo $variable-name

```
# Example
# Script to test MY knowledge about variables!
#
myname=Vivek
myos = Debian
myno=5
echo "My name is $myname"
echo "My os is $myos"
echo "My number is myno, can you see this number"
```

Shell arithmetics

expr var1 math-operator var2

Ex. echo `expr 1 + 3`

About Quotes

“	Double quotes	Removes meaning of anything enclosed (except $ and \ )
'	Single quotes	Anything enclosed remains unchanged
`	Back quotes	To execute a command

Examples of quotes

- **echo “Today is date”**
- **echo “Today is `date`”**

Read statement

read variable1, variable2, ….., variableN

Ex.

```
echo "Your first name please:"
read fname
echo "Hello $fname, Lets be friend!"
```

Conditional statement

if condition

then

execute if condition is true or exit status is 0

else

execute if condition not true

fi

Condition testing

test expression *or* **[expression]**

Works with integer, file types, character strings

Mathematical Comparators in [expr]

-eq	Equal to
-ne	Not equal to
-lt	Less than
-le	Less than equal to
-gt	Greater than
-ge	Greater than equal to

String Comparisons in [expr]

string1 = string2	Equal to
string1 != string2	Not equal to
string	*string* not null or not defined
-n *string*	*string* not null and does exist
-z string	*string* null and does exist

File testing in [expr]

-s *file*	Non empty file
-f *file*	File exists and not a directory
-d *file*	Directory exists and not a file
-w *file*	Writeable file
-r *file*	Read only file
-x *file*	Executable file

Logical Operators in Shell Scripts

! *expression*	NOT
expresssion1 -a *expression2*	AND
expression1 -o *expression2*	OR

“For” Loop in Shell Script

for ((expr1; expr2; expr3))

do

….. *execute until expr2 is true*

done *#evaluate expr3*

Ex.

```
for (( i=0; i <= 5; i++ ))
do
    echo $i
done
```

"While" Loop in Shell Script

while [condition]

do

Execute when condition is true

done

Wild Cards

*****	Matches any string or group of characters
?	Matches any single character
[...]	Matches any one of the enclosed characters *Note: A pair of characters separated with – denotes a rangem ex. [a-c]* *If first character is ^ or !, then characters not enclosed is matched, ex. [!a-r]*

Exit status

- Once a command is executed, it returns two types of values:

  1. return value zero (0): command successful

  2. return value non-zero: command unsuccessful or error executing command

- This value is known as **Exit Status**
- To determine Exit Status, use **$?** variable of shell

Example of Exit Status

- **rm unknown-file**

  echo $?

- **ls**

  echo $?

Shell script example

```
#!/bin/bash
# This script clears the terminal, displays a greeting and gives information
# about currently connected users.  The two example variables are set and displayed.

clear   # clear terminal window

echo "The script starts now."

echo "Hi, $USER!"  # dollar sign is used to get content of variable
echo

echo "I will now fetch you a list of connected users:"
echo
w  # show who is logged on and
echo   # what they are doing

echo "I'm setting two variables now."
COLOUR="black"# set a local shell variable
VALUE="9"  # set a local shell variable
echo "This is a string: $COLOUR" # display content of variable
echo "And this is a number: $VALUE" # display content of variable
echo

echo "I'm giving you back your prompt now."
echo
```