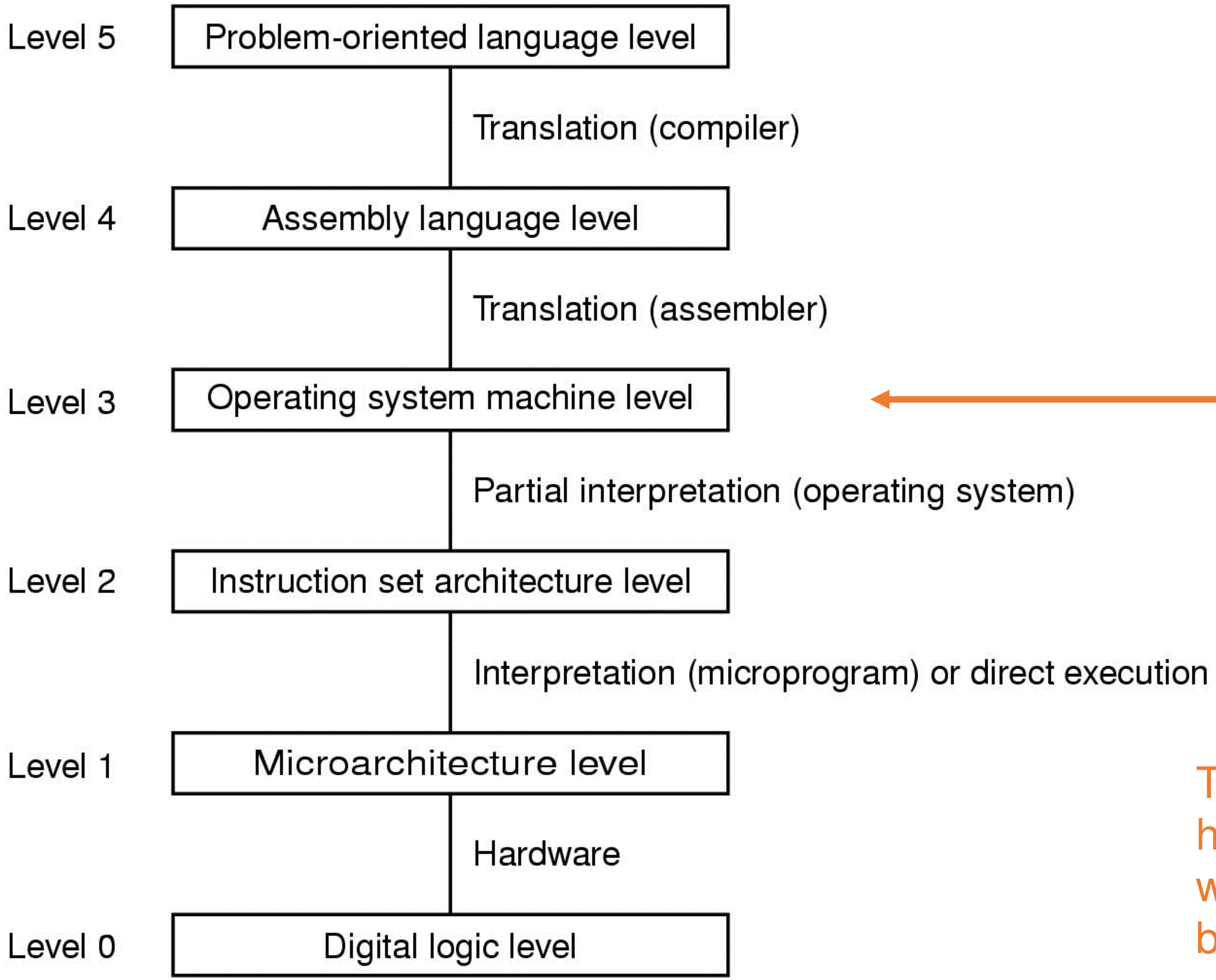
Graphical User Interfaces	Higher-level Programming
Operating Systems	
Low-level Programming	
Basic Machine Architecture	
Silicon	

CSCU9V4 Systems

Systems lecture 14

Operating Systems 1
Overview &
Virtual memory

OSM level (Tanenbaum)



Section 6, Tanenbaum

Tanenbaum has also written other books on OSs.

What is an Operating System?

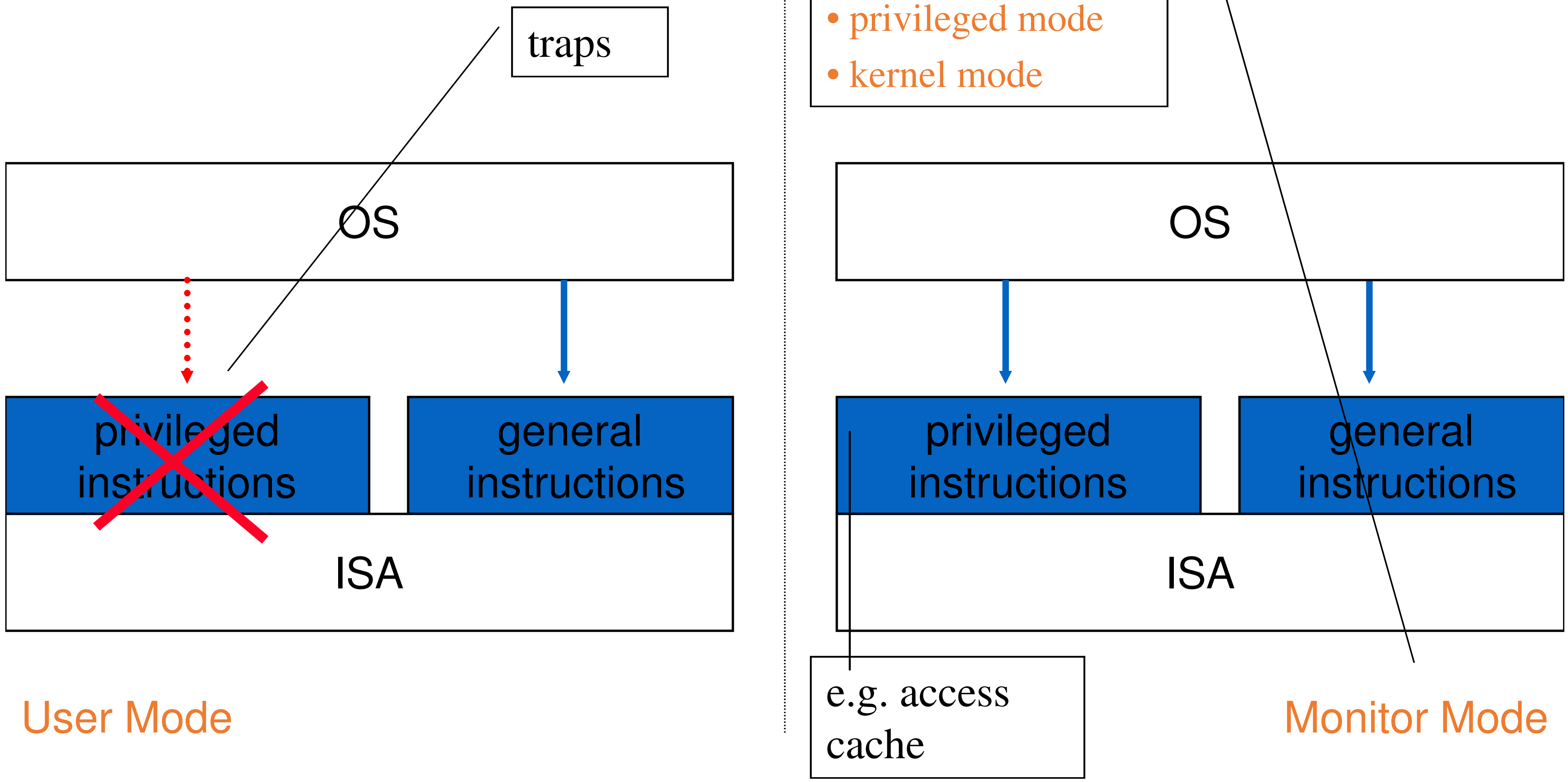
- Software that
  - Provides facilities
    - Independent of the precise underlying hardware
    - Portability
    - "**Virtual machine**"
  - Handles resources
    - Securely
    - Logically
    - Safely
    - Fairly
  - Also *may* (all desktop OS's supply these: but there are other non-desktop OS's as well)
    - Provide a GUI (graphical user interface)
    - Provide a File System
    - Handle external devices in a logical way

Computer Modes (1)

- **Supervisor mode protects CPU from errant users**
- **.. and users from each other**

Also called:

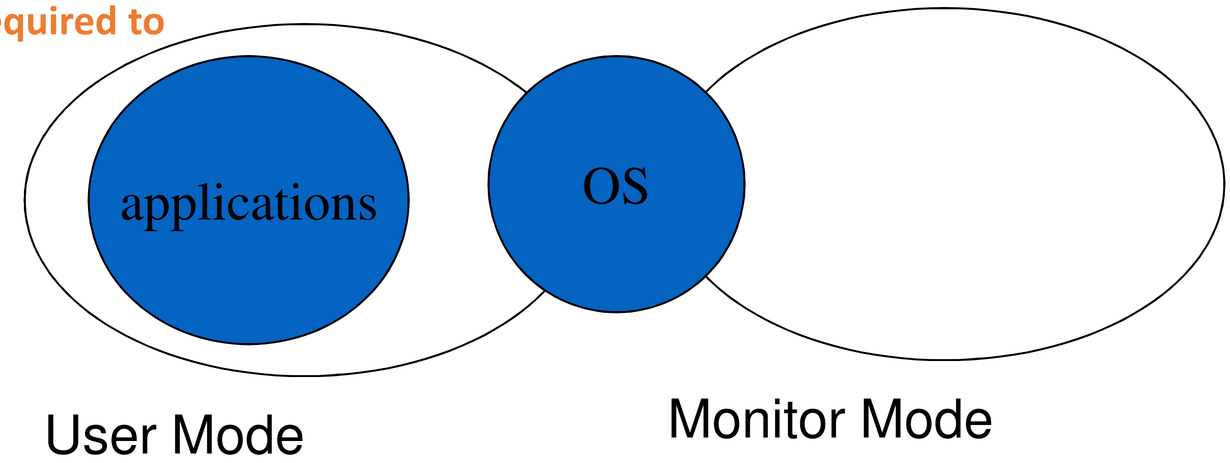
- monitor mode
- system mode
- privileged mode
- kernel mode



Computer Modes (2)

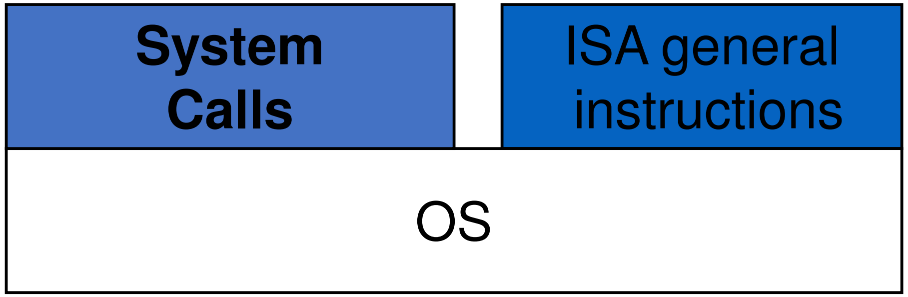
- **Applications run in user mode**
  - **your programs**
  - **editors**
  - **spreadsheet programs**
  - **compilers**

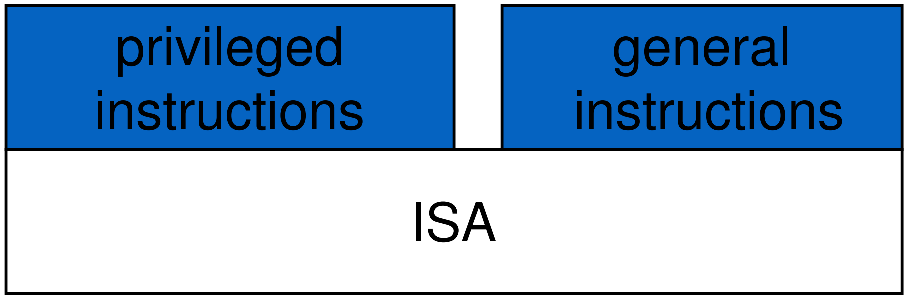
- **OS uses *both* modes**
  - **passwords: user mode**
  - **Schedulers and interrupt handlers: monitor mode: monitor mode only used when necessary**
  - **file system: either but OS required to enable user-mode**



Computer Modes (3)

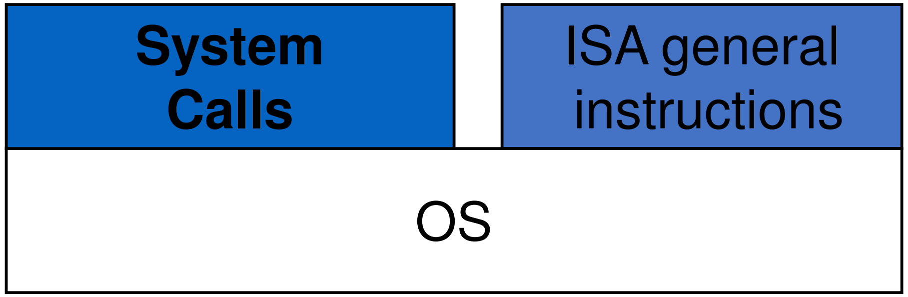
- **But users can (indirectly) run privileged instructions**
- **system calls (*some parts* of which will use monitor mode)**
  - **I/O**
    - **files**
    - **printers & peripheral devices**
  - **file system & directory management**
  - **process management**
  - **communications management**
    - **messaging**
    - ***remote* devices**
  - **miscellaneous**
    - **timing**
    - **security**
- **application does not leave user mode**
  - **Uses O.S. capabilities when required**
- **And memory?**

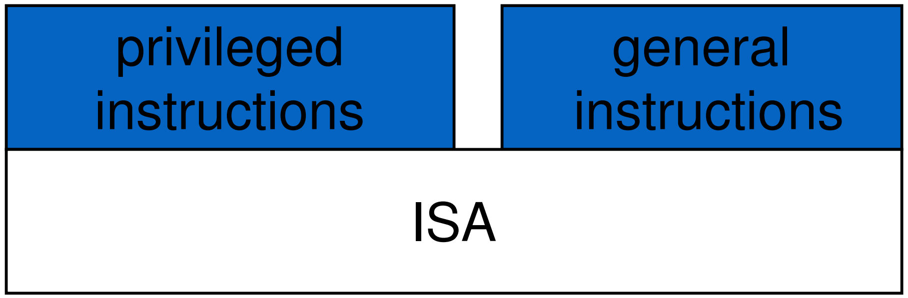




Computer Modes (4)

- **Instruction addresses in user mode will be adjusted by the OS**
- **memory management**
  - **paging**
  - **virtual memory**
  - **no concern to programmer**
  - **…. or even compiler writer**
  - **Done invisibly by OS**





System Calls

System Calls

- File management
  - **open, read, write, close, and lock files**
- Directory management
  - **create and delete directories; move files around**
- Process management
  - **spawn, terminate, trace, and signal processes**
- Memory management
  - **share memory among processes; protect pages**
- Getting/setting parameters
  - **get user, group, process ID; set priority**
- Dates and times
  - **set file access times; use interval timer; profile execution**
- Networking
  - **establish/accept connection; send/receive message**
- Miscellaneous
  - **enable accounting; manipulate disk quotas; reboot the system**

OS : *2 views*

- virtual files
  - **logical view rather than physical**
- virtual I/O
  - **again a logical view offered**
- virtual CPU
  - **in reality shared**
- virtual MM
  - **“physical” addresses of linker not used**

Virtual machine

- files
  - **shared – but 2 users cannot write**
- I/O
  - **shared**
- CPU
  - **time is allocated**
- virtual MM
  - **MM is “partitioned”**

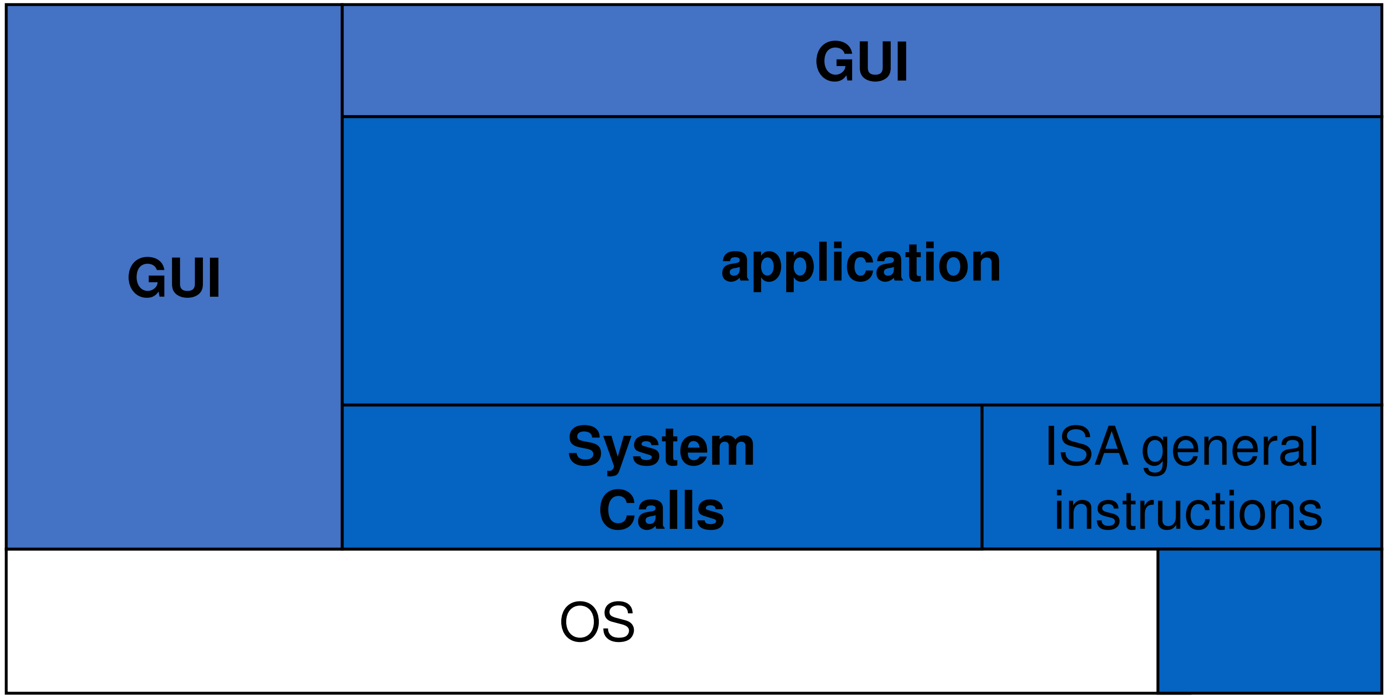
resource manager

so an OS ......

- Can allow multiple users
  - **Common access, but not interfering with each other**
- Can allow multiple processes
  - **on a single CPU (or a number of CPUs/cores)**
  - **running a number of programs “in parallel”**
- protects and allocates shared resources
  - **files**
  - **printers**
  - **MM**
  - **CPU (core) time**
- .. and ....
  - **Implements security**
  - **Ensures safe program execution**
  - **Provides access to communications**
  - **Provides access to remote facilities**

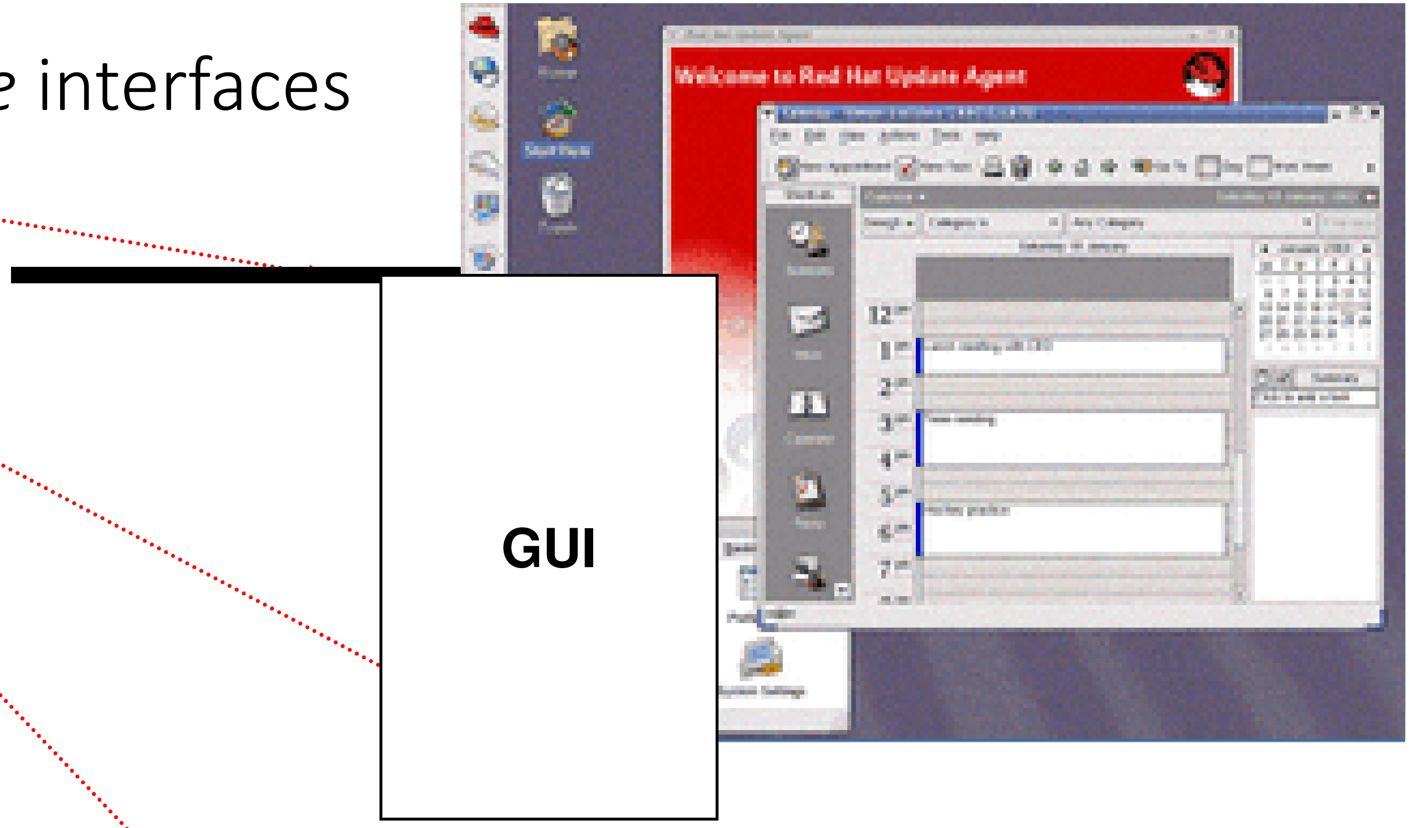
……but also a user perspective

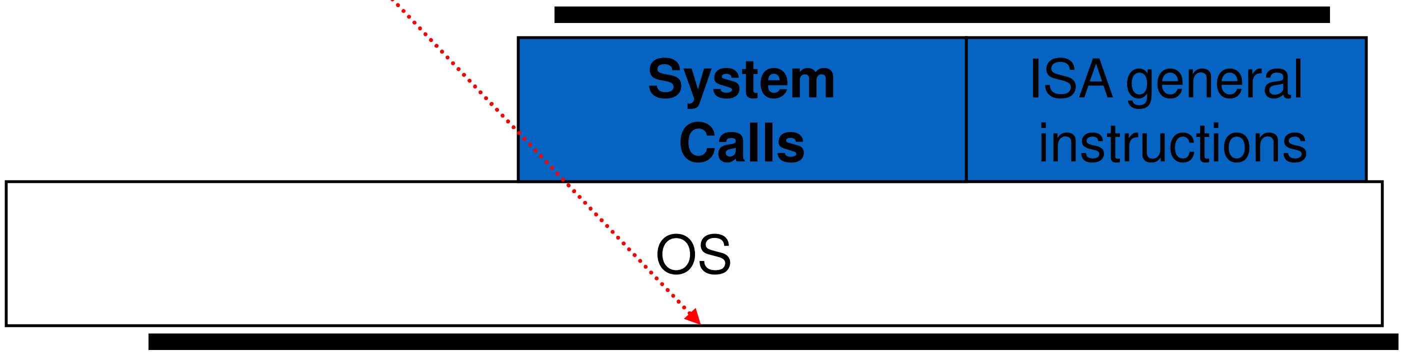
- **File access**
- **resource control**
- **Security and “levels” of access**



In effect *three* interfaces

- User GUI & CI
- application/API
- ISA





What if the sum of all program memory is greater than available/physical memory?

(Virtual memory)

Virtual Memory: why?

- The programs being run may not all fit into memory.

Windows Task Manager

File Options View Help

Applications | Processes | Performance | Networking

Image Name	User Name	CPU	Mem Usage	Peak Mem Usage	Page Fa...	VM Size	Threads
POWERPNT.EXE	lss	00	37,280 K	55,728 K	26,479	26,644 K	6
explorer.exe	lss	00	24,604 K	24,912 K	25,770	15,684 K	15
jusched.exe	lss	00	2,032 K	2,032 K	517	484 K	1
ACIntUsr.EXE	lss	00	2,912 K	3,044 K	870	908 K	2
editplus.exe	lss	00	5,820 K	5,824 K	1,494	1,576 K	2
acrotray.exe	lss	00	2,216 K	2,216 K	569	596 K	1
VPTray.exe	lss	00	9,436 K	9,460 K	6,321	5,200 K	8
taskmgr.exe	lss	02	1,748 K	4,616 K	1,631	1,308 K	3
PRONoMgr.exe	lss	00	3,628 K	3,636 K	946	1,272 K	3
pdesk.exe	lss	00	3,128 K	3,164 K	801	1,148 K	18
Rtvscan.exe	SYSTEM	00	50,468 K	66,956 K	710,095,...	42,920 K	59
jucheck.exe	lss	00	6,788 K	6,836 K	1,794	3,144 K	6
msmsgs.exe	lss	00	3,676 K	4,828 K	2,132	2,984 K	3
ctfmon.exe	lss	00	3,512 K	3,576 K	1,287	892 K	1
spoolsv.exe	SYSTEM	00	1,908 K	6,800 K	6,328	3,848 K	13
ccApp.exe	lss	00	8,904 K	9,004 K	3,382	5,924 K	9
lexbces.exE	SYSTEM	00	1,172 K	3,808 K	1,197	1,604 K	11
ccEvtMgr.exe	SYSTEM	00	2,388 K	6,132 K	3,547	3,996 K	15
ccSetMgr.exe	SYSTEM	00	3,220 K	4,676 K	3,078	3,720 K	6
alg.exe	LOCAL SERVICE	00	668 K	3,420 K	1,019	1,116 K	6
SMAgent.exe	SYSTEM	00	280 K	1,648 K	475	496 K	2
SMTray.exe	lss	00	2,476 K	2,476 K	633	620 K	1
svchost.exe	LOCAL SERVICE	00	1,984 K	6,744 K	31,245	3,828 K	19
retroclient.exe	SYSTEM	00	1,844 K	3,820 K	8,678	1,184 K	7
svchost.exe	NETWORK SERVICE	00	1,276 K	3,376 K	5,177	1,344 K	6
SavRoam.exe	SYSTEM	00	1,204 K	4,452 K	1,232	1,332 K	4
svchost.exe	SYSTEM	00	138,220 K	228,772 K	55,482,753	155,384 K	59
svchost.exe	NETWORK SERVICE	00	1,600 K	4,388 K	1,457	1,860 K	10
svchost.exe	SYSTEM	00	2,012 K	4,992 K	6,509	3,292 K	24
lsass.exe	SYSTEM	00	3,000 K	7,024 K	5,813,239	4,140 K	21
services.exe	SYSTEM	00	4,376 K	22,100 K	121,004	5,660 K	16
winlogon.exe	SYSTEM	00	1,524 K	17,960 K	1,637,187	9,520 K	19
csrss.exe	SYSTEM	00	2,896 K	5,072 K	22,229	1,740 K	14
RemotSvc.exe	SYSTEM	00	356 K	1,968 K	564	588 K	2
smss.exe	SYSTEM	00	108 K	688 K	305	172 K	3
mgabg.exe	SYSTEM	00	376 K	1,216 K	362	380 K	1
MDM.EXE	SYSTEM	00	868 K	2,908 K	835	900 K	4
wmpnetwk.exe	NETWORK SERVICE	00	3,000 K	8,508 K	2,609	5,608 K	14
svchost.exe	SYSTEM	00	660 K	3,312 K	935	1,484 K	8
DefWatch.exe	SYSTEM	00	2,196 K	5,000 K	11,868	1,548 K	3
ACLIENT.EXE	SYSTEM	00	1,352 K	8,256 K	26,293	4,308 K	5
firefox.exe	lss	00	24,384 K	29,904 K	29,904	14,424 K	10
System	SYSTEM	00	52 K	2,556 K	34,947	28 K	60
System Idle Process	SYSTEM	98	16 K	0 K	0	0 K	1

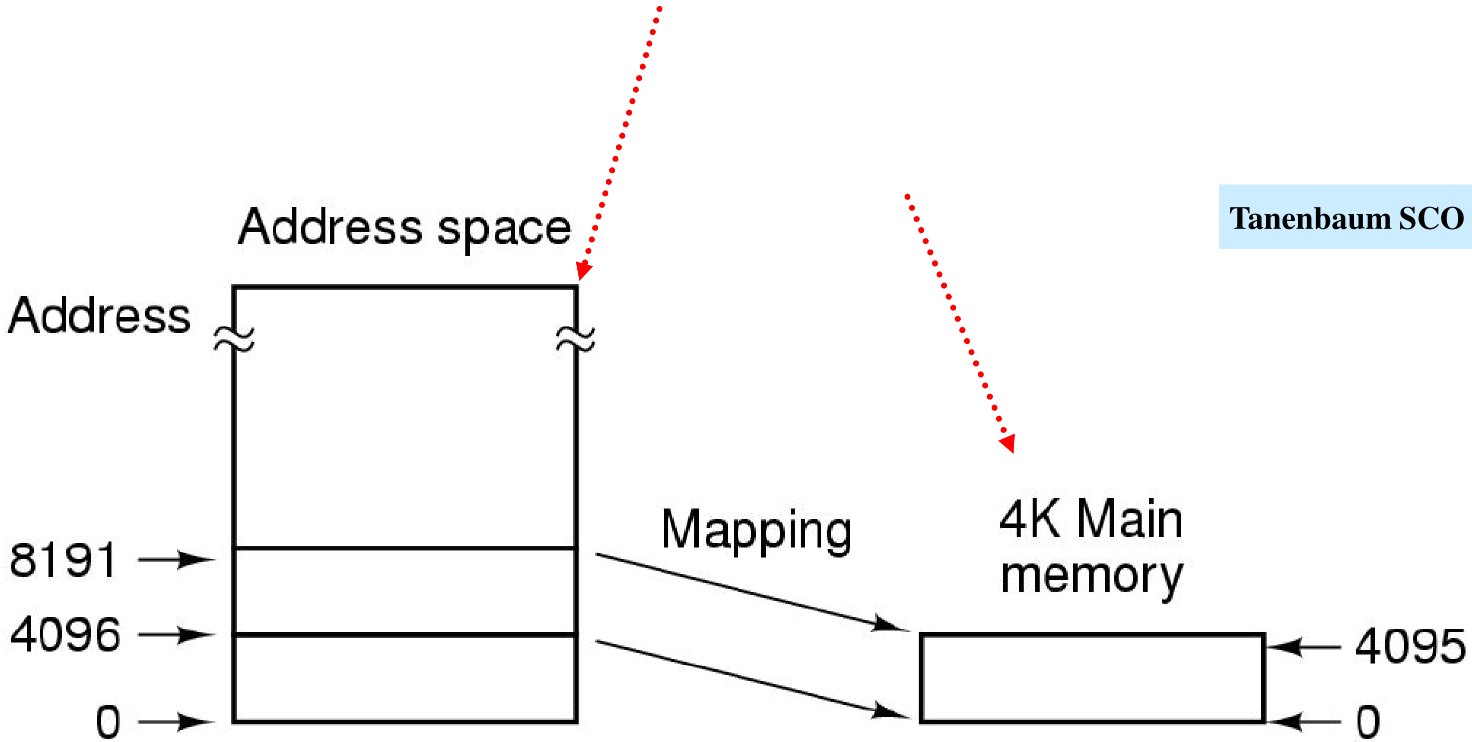
Show processes from all users

End Process

Processes: 44 | CPU Usage: 2% | Commit Charge: 477M / 2461M

Virtual memory

- Use the facts that
  - The machine can (probably!) address more memory than is on the machine
  - Applications don't use all their program at once
- Divide up the possible (virtual) memory into small areas (pages)
  - (4Kbytes here)
- Place the small areas into real (physical) memory



Tanenbaum SCO

Physical memory and virtual memory

- we have 2 address spaces
- … that of the physical memory
    - **physical address space**
    - **Might be 512Mbytes, or 1Gbyte, or 16GBytes…**
- … and that which the address bits allow us to address,
    - **virtual address space**
    - e.g. 32 bits would allow 4Gbytes, or 36 bits would allow 64GBytes

- Programs’ process space has corresponding virtual space (addresses)
  - And these are mapped into physical addresses.
- Programs are not aware of the limits imposed by the actual memory size
  - Only of the limits imposed by the virtual address size
    - Which is usually much larger
    - And large enough to hold all the programs being used.

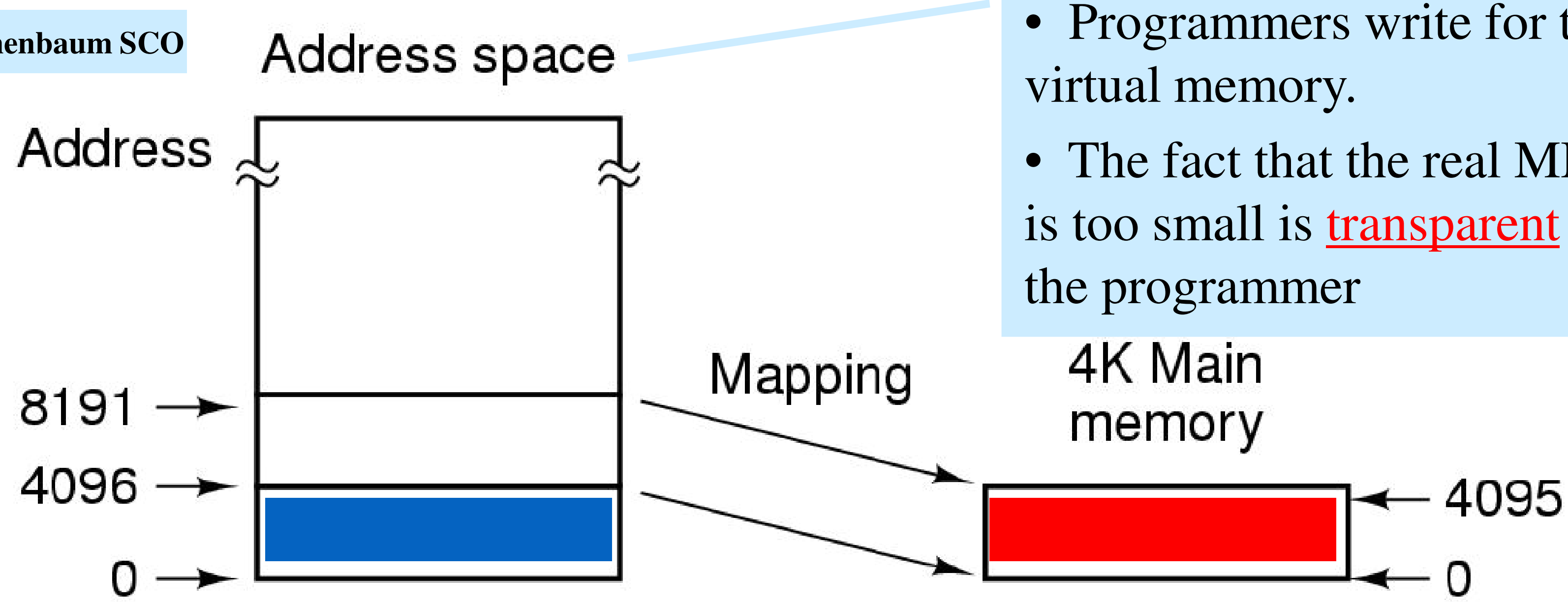
How virtual memory is used: demand paging

- Pages are loaded as they are required (i.e. on demand)
  - this is known as **demand paging**
- Program starts up: a single page is loaded), and
- This in turn loads a number of pages
- As time goes by some more get loaded
  - But many parts of the program are only required when specific functions are used
  - (think of indexing in Word, or error handling in a compiler)
  - As the program goes through different phases
    - Drawing in PowerPoint, for example
  - Different parts of the program are used, and parts that were in use cease to be in use.
- In fact most programs only ever use a small part of their whole program size

The Mechanism of paging

- assume a 4Kbyte (very tiny!) Main Memory.
- If the program is larger, and we want to execute an instruction at a (virtual) address above 4K ...
- we need to move that relevant 4Kbyte section into MM
- in practice the *contents* of the virtual address space is on (hard) disk
- we must also handle the "address change"

Tanenbaum SCO



- Programmers write for this virtual memory.
- The fact that the real MM is too small is transparent to the programmer

Implementing paging

- in practice physical memory has a number of pages (not just one)
- can be in the 000's
- here each page is 4Kbyte
- can be 4K to 256Mbytes
- again, it's not just a matter of swapping pages in and out …
- the computer only understands physical addresses, not virtual ones!
- The memory Management Unit (MMU) does the mapping of virtual to physical addresses.

Tanenbaum SCO

Page	Virtual addresses
≈	≈
15	61440 – 65535
14	57344 – 61439
13	53248 – 57343
12	49152 – 53247
11	45056 – 49151
10	40960 – 45055
9	36864 – 40959
8	32768 – 36863
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

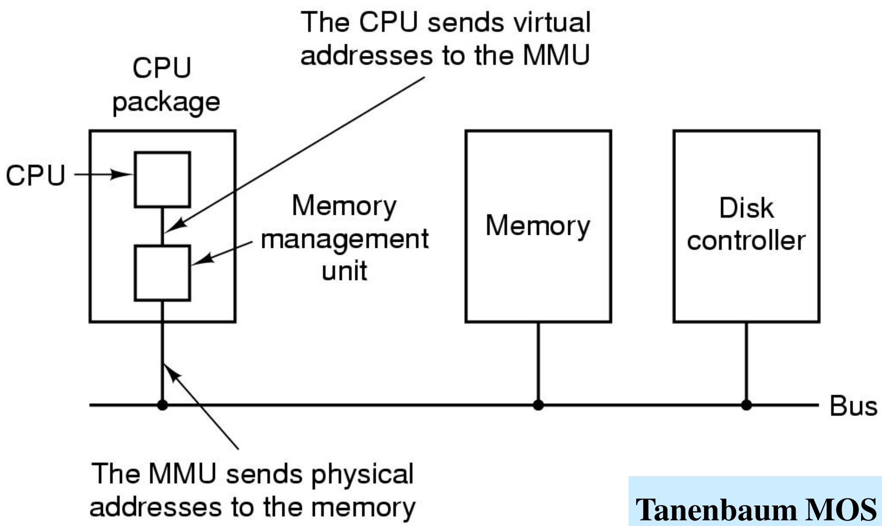
(a)

Bottom 32K of main memory

Page frame	Physical addresses
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(b)

Implementing paging: role of MMU



Tanenbaum MOS

- Memory Management Unit
- hardware close to the CPU

Tanenbaum SCO

3 bits

Page	Virtual addresses
≈	≈
15	61440 – 65535
14	57344 – 61439
13	53248 – 57343
12	49152 – 53247
11	45056 – 49151
10	40960 – 45055
9	36864 – 40959
8	32768 – 36863
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(a)

Bottom 32K of main memory

Page frame	Physical addresses
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(b)

32 bit address

MMU

15 bit address

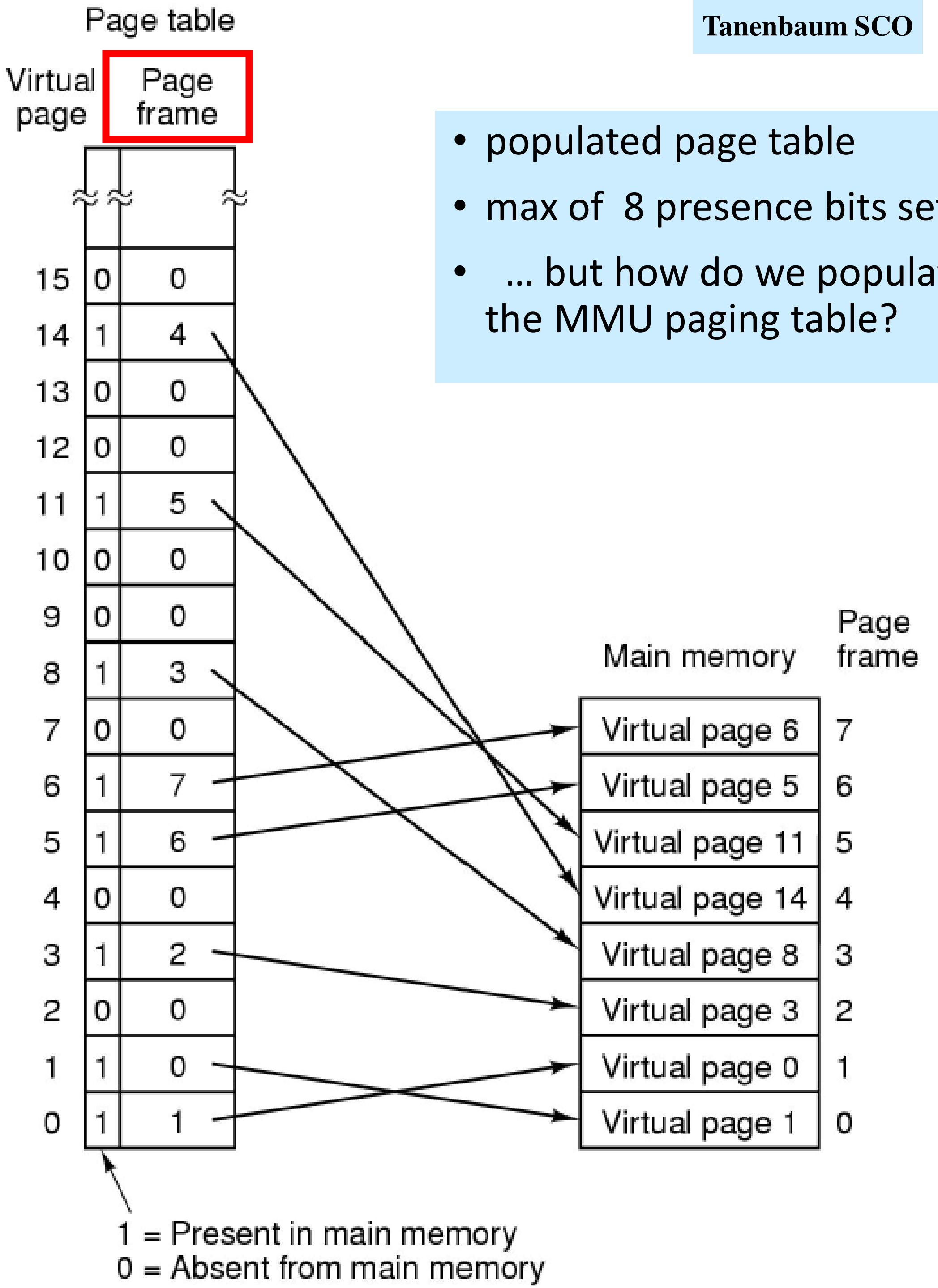
12 bits

Maps from the virtual address space to the physical address space

Structure of MMU (2)

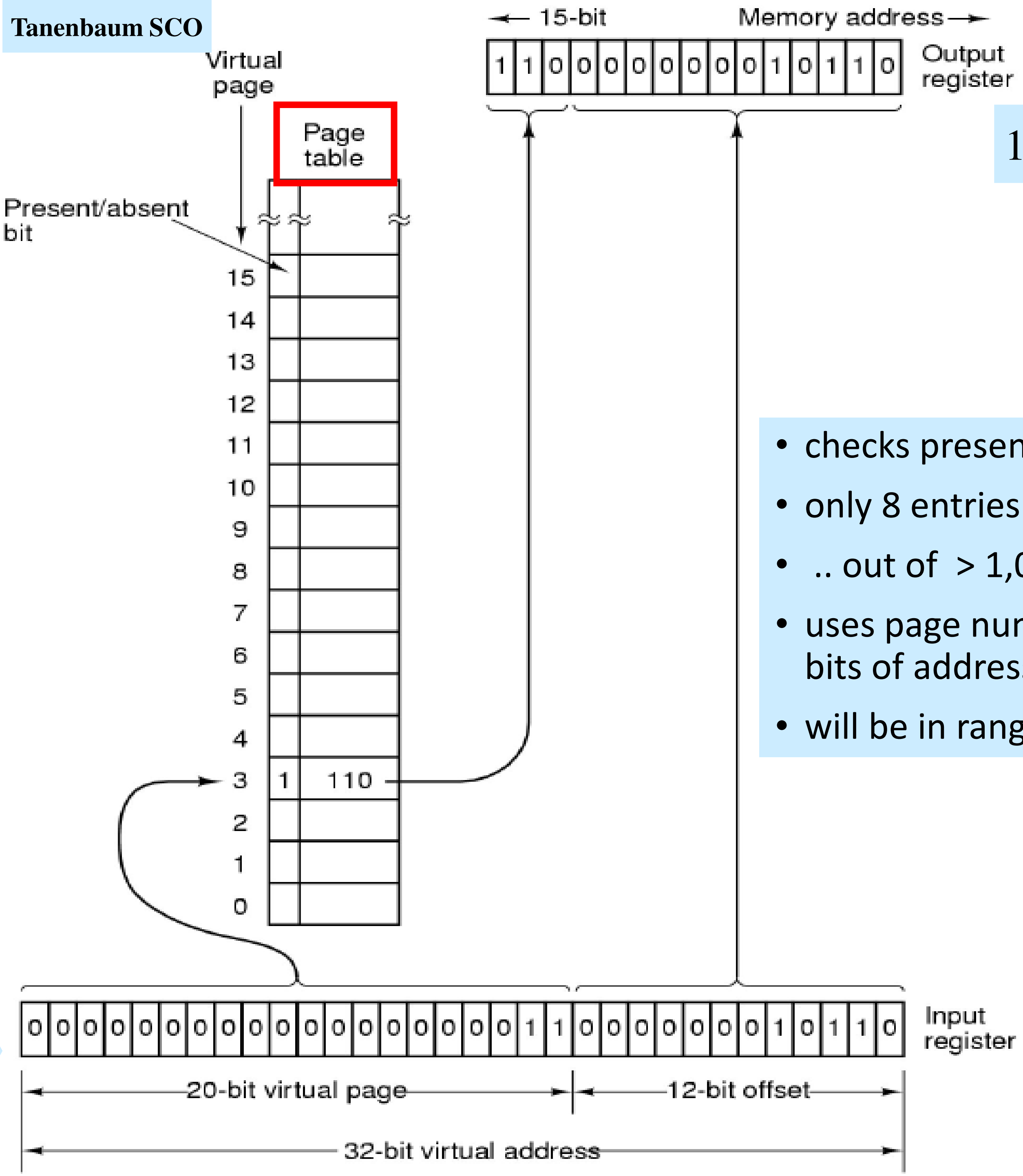
Tanenbaum SCO

- populated page table
- max of 8 presence bits set
- … but how do we populate and maintain the MMU paging table?



Structure of MMU (1)

Tanenbaum SCO

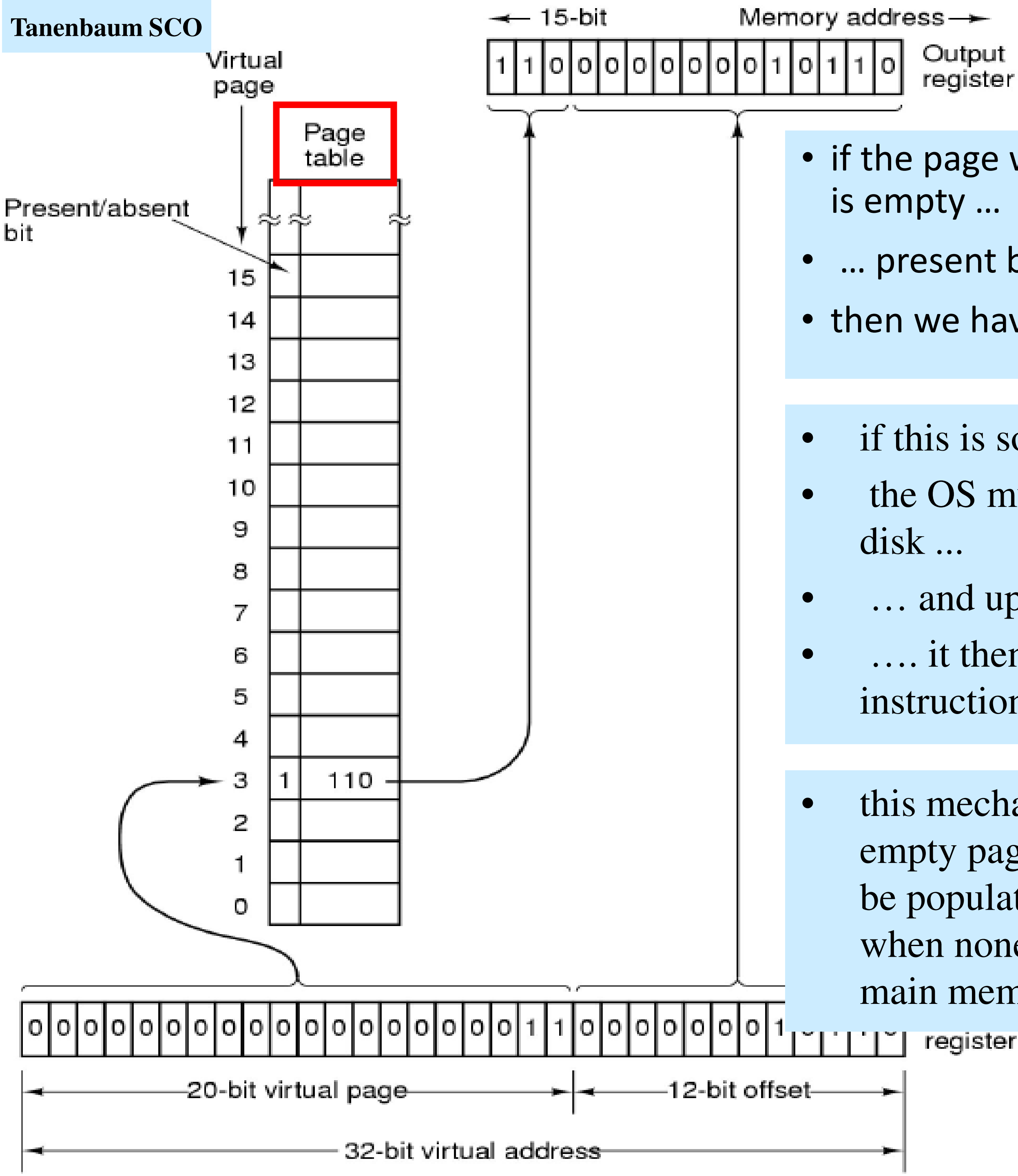


15 bit address

32 bit address

- checks presence bit
- only 8 entries set
- .. out of > 1,000,000
- uses page number for top bits of address
- will be in range 0 to 7

Page fault



- if the page we attempt to access is empty …
- … present bit clear/zero
- then we have a page fault

- if this is so …
- the OS must read in page from disk ...
- … and update page table
- …. it then can re-try the instruction

- this mechanism will ensure an empty page table will gradually be populated at the beginning when none of the program is in main memory

Issues

- A single instruction may cause more than 1 page fault
  - a new instruction may cause 1 page fault
  - the instruction may access another page on memory that is not in main memory - hence a second page fault
  - … and perhaps the instruction accesses a second memory in location in yet another page that is not present - and so a third page fault

- remember this mapping is for both instructions and data!
  - Instructions are read-only
  - But data may be read/write
    - Altered pages need rewritten to virtual (disk) memory

- Other issues….

Process Swapping

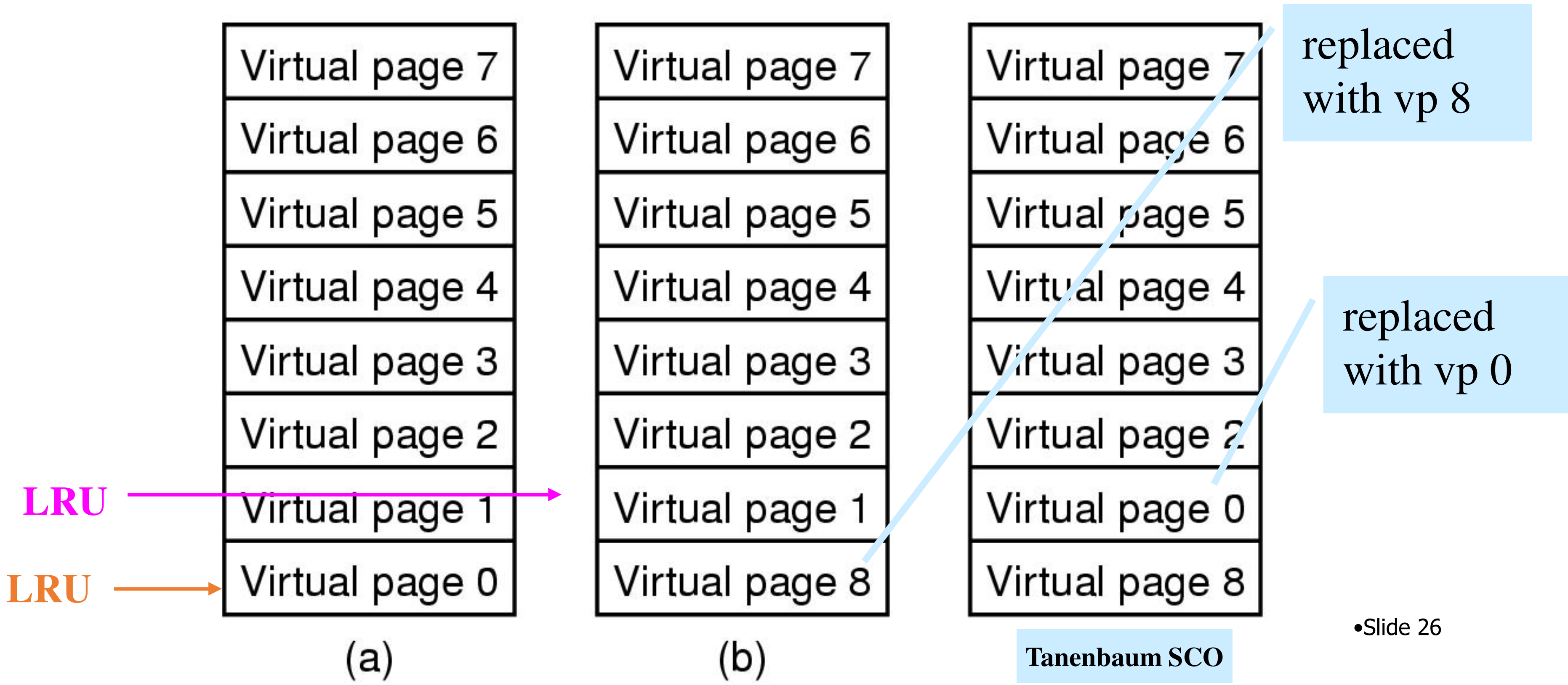
- there is a page table *for each process*
- if a CPU is being shared in time between a number of processes
- … then the table needs to be re-populated each time a process (re)starts
- better to have a **working set**
  - the set of recently used pages (explain why this works)
  - Much more efficient than just loading one page, and then using demand paging all over again
- based on "experience" some pages are more likely to be needed than others - so load the working set *before* the process re-starts.
- … that is the MM and the table.
- this is *locality of reference* already seen in caching

- whether we use demand paging or the working set approach we need a way of deciding which page should go, when we need a new page in a full MM ….

Page replacement policy (1)

- we are only touching on this - those seeking a larger array of algorithms are referred to Tanenbaum - *Modern Operating Systems*.

- LRU - Least Recently Used
- works well, but if the working-set is larger than the MM ……



Page replacement policy (2)

- FIFO - First In First Out
- or perhaps … LRL - Least Recently Loaded
- again works well, might throw out important pages ...
- .. also ... if the working-set is larger than the MM ……
- … you get a lot of **page faults**
- this is true of all algorithms for page replacement

- any program with a high percentage of page faults is said to be **thrashing**
  - reading a page from disk: 10mS
  - executing an instruction: a few nS
    - Effect: little useful processing gets done
- Solution: add more main memory
  - Indeed, adding more memory often has a much larger impact on performance than using a faster processor for this reason.

End of lecture