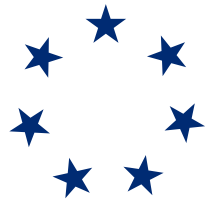
Memory: Paging System

Instructor: Hengming Zou, Ph.D.

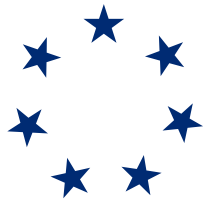
In Pursuit of Absolute Simplicity 求于至简，归于永恒

Content



- Space Management
- Design issues for paging systems
- Policy and mechanism

Space Management

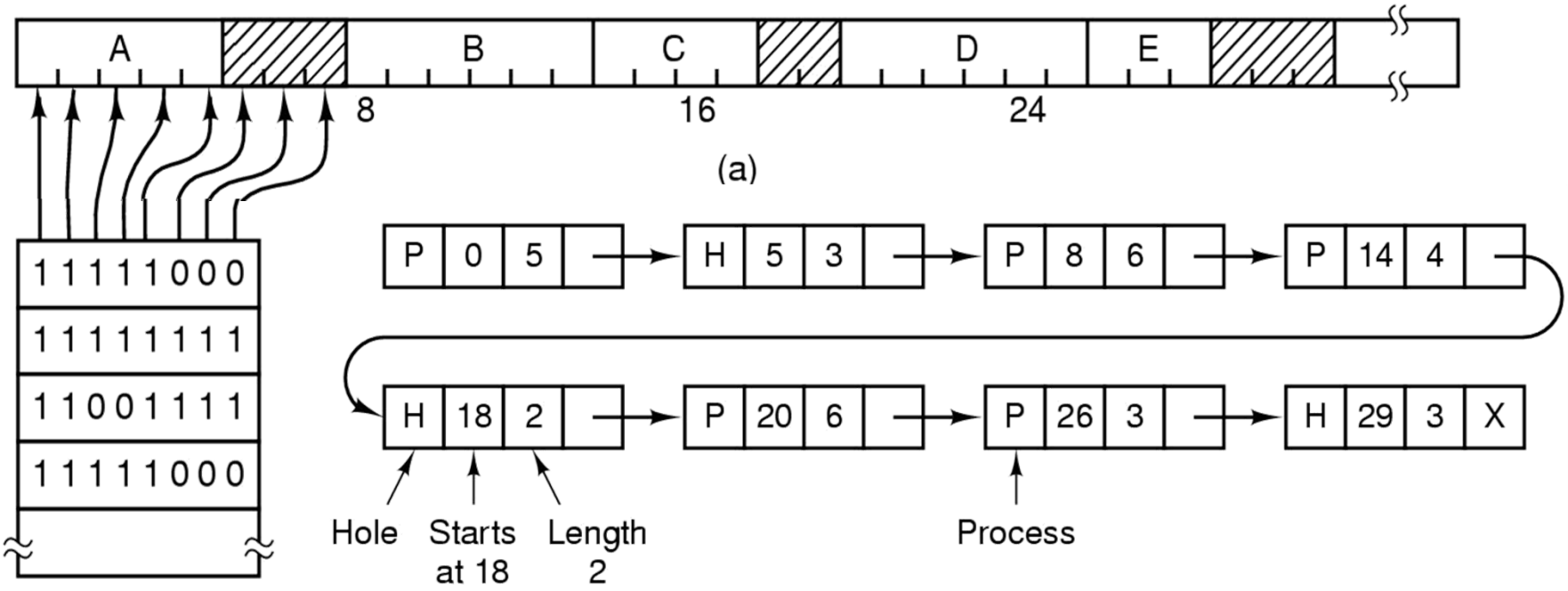


- Need keep track of memory used and available
- Two options to keep track of memory space
- Bit map approach
  - Divide memory into allocation units
  - Use one bit to mark if the unit is allocated or not

- Linked list approach
  - Divide memory into allocation units
  - Use a linked list to indicate allocated and available units in chunks

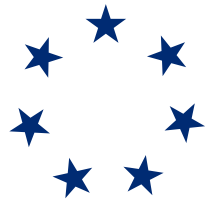
Space Management

Memory allocation

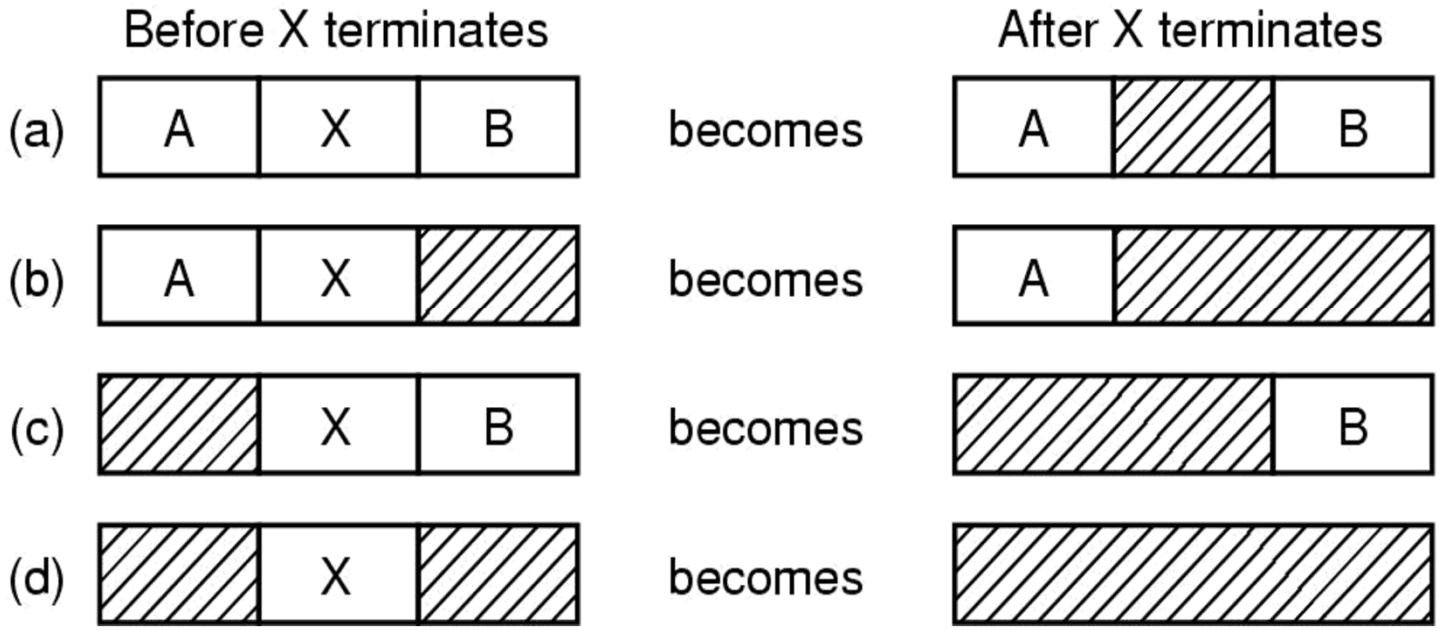


c) Linked list representation

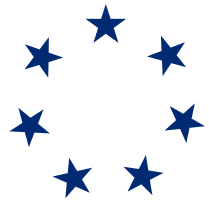
Space Management with Linked Lists



- What happens when units are released?
- May need to collapse linked list appropriately

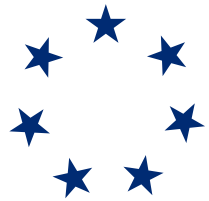


Design Issues for Paging Systems



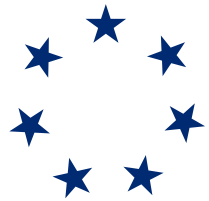
- Thrashing
- Local versus Global Allocation Policies
- Page size
- OS Involvement with Paging
- Page fault handling

Thrashing



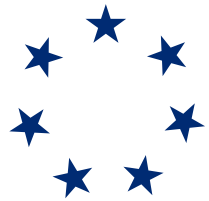
- What would happen with lots of big processes,
  - all actively using lots of virtual memory?
  - **Frequent page faults**
- What happens when a work set is big than the available memory frames?
  - Constant page fault to bring pages in and out
  - This is called “thrashing”

Thrashing



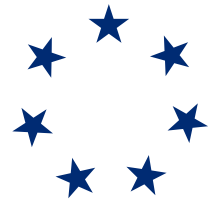
- Average access time =
  - hit rate * hit time + miss rate * miss time
  - e.g. hit time = .0001 ms, miss time = 10 ms
- 100% hit rate:
  - average access time is .0001 ms
- 99% hit rate:
- 90% hit rate:

Solutions to Thrashing



- If a single process is actively using more pages than can fit
  - there's no solution
  - that process (at least) will thrash
- If cause is the combination of several processes
  - Can alleviate by swapping all pages of a process out to disk
  - That process won't run at all
    - →but other processes will run much faster
  - Overall performance improves
- But be careful of Belady's Anomaly

Belady's Anomaly



All pages frames initially empty

	0	1	2	3	0	1	4	0	1	2	3	4
Youngest page	0	1	2	3	0	1	4	4	4	2	3	3
		0	1	2	3	0	1	1	1	4	2	2
Oldest page			0	1	2	3	0	0	0	1	4	4
	P	P	P	P	P	P	P			P	P	

9 Page faults

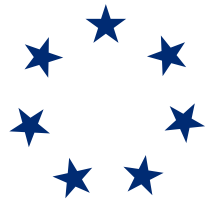
(a)

	0	1	2	3	0	1	4	0	1	2	3	4
Youngest page	0	1	2	3	3	3	4	0	1	2	3	4
		0	1	2	2	2	3	4	0	1	2	3
Oldest page			0	1	1	1	2	3	4	0	1	2
				0	0	0	1	2	3	4	0	1
	P	P	P	P			P	P	P	P	P	P

10 Page faults

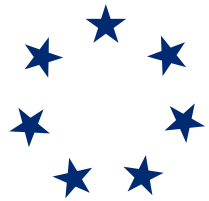
(b)

Local versus Global Allocation



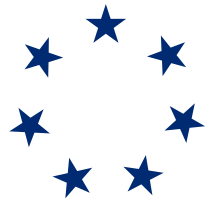
- When evict a page, do we only look at
  - pages of the same process for possible eviction
    - →Local allocation policy
  - Or do we look at the whole memory for victim?
    - →Global allocation policy

Local versus Global Allocation



- Global replacement:
  - Consider all pages equally for eviction need
- Local replacement:
  - Only consider pages belonging to the process needing a new page when looking for a page to evict
  - But how to set the # of pages assigned to a process?
- Generally, global has lower overall miss rate
  - but local is more “fair”

Local versus Global Allocation



	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

Local policy

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

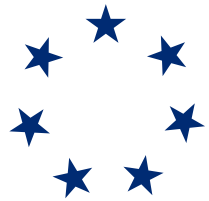
(b)

Global policy

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

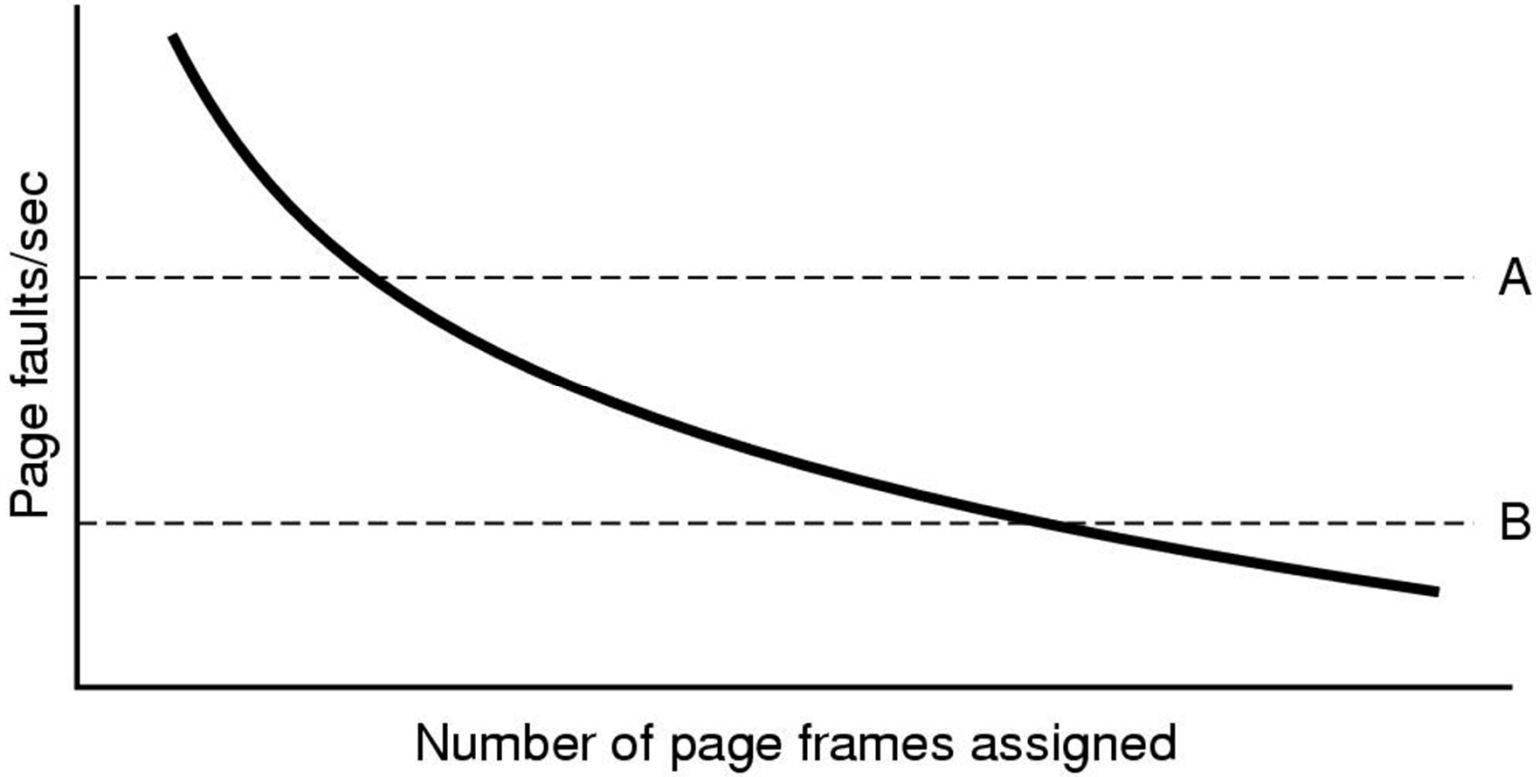
(c)

Local versus Global Allocation



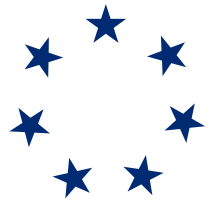
- In global allocation policy, use PFF to manage the allocation
  - PFF → page fault frequency
- If PFF is large, allocate more memory frames
- Otherwise, decrease the number of frames
- Goal is to maintain an acceptable PFF

Local versus Global Allocation



Page fault rate as a function of # of page frames assigned

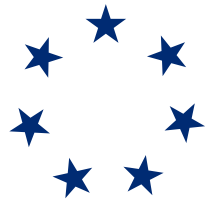
Page Size



- What happens if page size is small?
- What happens if page size is really big?
- Could we use a large page size but let other processes use the leftover space in the page?
- Page size is typically a compromise
  - e.g. 4 KB or 8 KB

- What happens to paging if virtual address space is sparse?
  - most of the address space is invalid,
  - with scattered valid regions

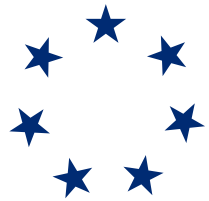
Small Page Size



- Advantages
  - less internal fragmentation
  - better fit for various data structures, code sections
  - less unused program in memory
- Disadvantages
  - programs need many pages, larger page tables

- Therefore, to decide a good page size, one needs to balance page table size and internal fragmentation

Page Size

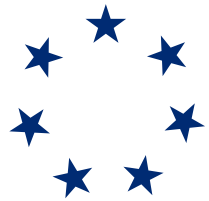


- Overhead due to PT and internal fragmentation can be calculate as:
  - $(s \times e)/p + p/2$ ← Internal fragmentation
- s = average process size in bytes
- p = page size in bytes
- e = page entry

Page table space

- Overhead is minimized when:
  - P=sqrt(2×s×e)

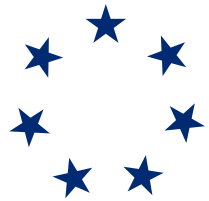
Fixed vs. Variable Size Partitions



- Fixed size (pages) must be compromise
  - too small a size leads to a large translation table
  - too large a size leads to internal fragmentation

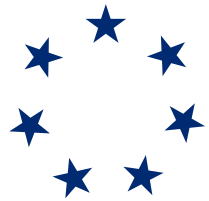
- Variable size (segments) can adapt to the need
  - but it's hard to pack these variable size partitions into physical memory
  - leading to external fragmentation

Load Control



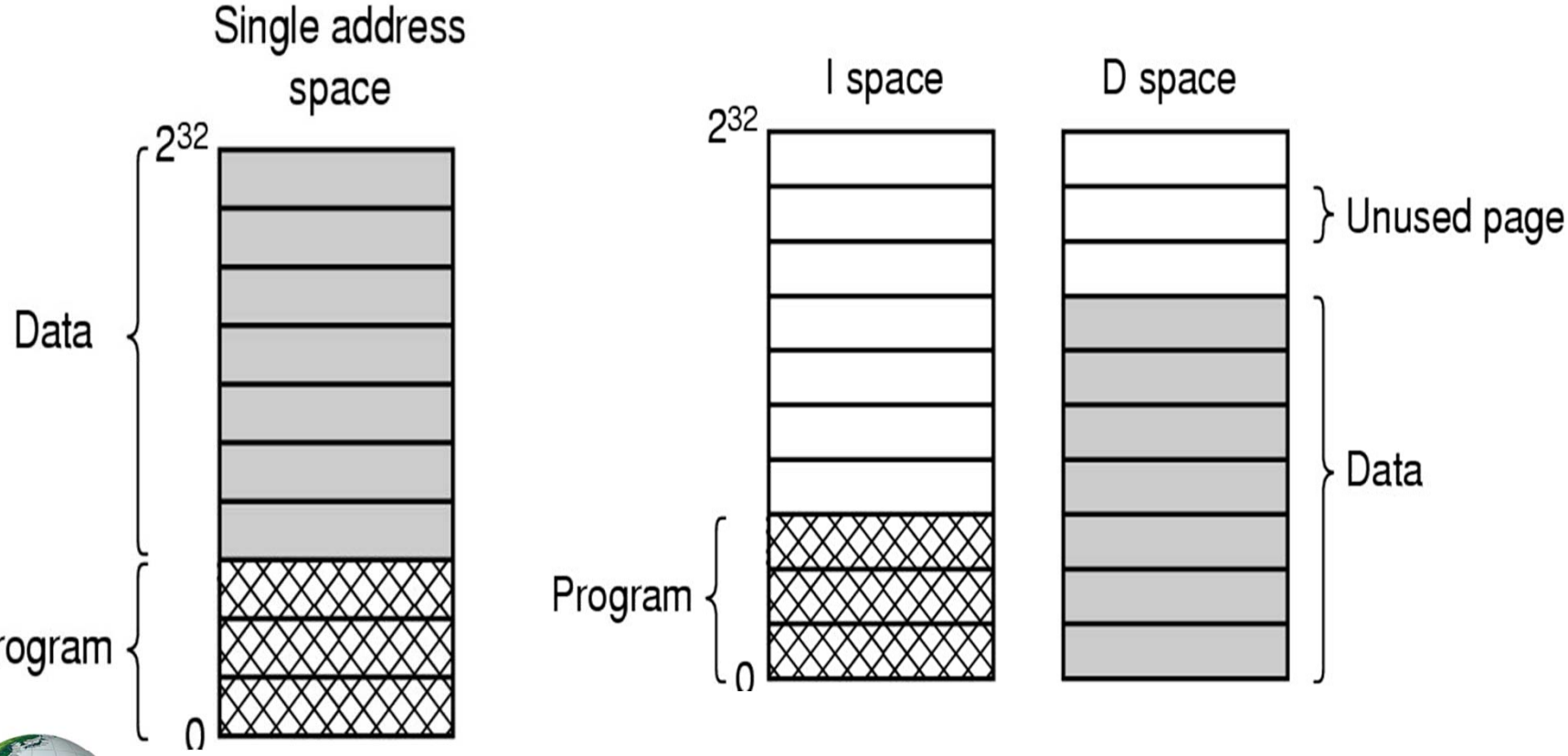
- Despite good designs, system may still thrash
- When PFF algorithm indicates:
  - some processes need more memory
  - but no processes need less
- Solution :
  - Reduce # of processes competing for memory
  - swap 1 or more to disk, divide up pages they held
  - reconsider degree of multiprogramming

Separate Instruction and Data Spaces

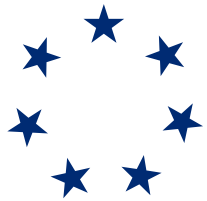


- With combined instruction and data space,
  - programmers have to fit everything into 1 space
- By separating instruction and data space, we:
  - Allows programmers more freedom
  - Facility sharing of program text (code)

Separate Instruction and Data Spaces

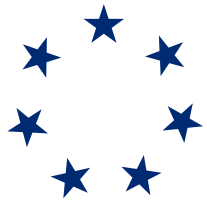


OS Involvement with Paging



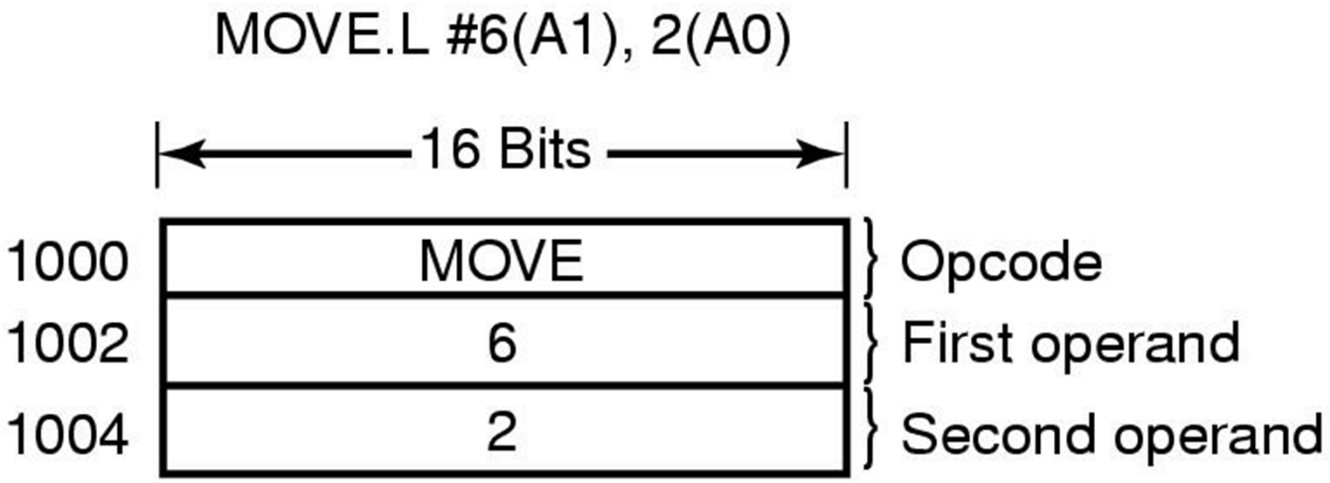
- Four times when OS involved with paging
- Process creation
  - determine program size, create page table
- Process execution
  - MMU reset for new process, TLB flushed
- Page fault time
  - determine the virtual address that causes the fault
  - swap target page out, needed page in
- Process termination time
  - release page table, pages

Page Fault Handling



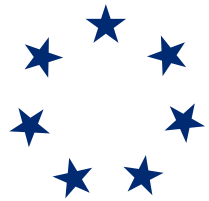
- Hardware traps to kernel, general registers saved
- OS determines which virtual page needed
- OS checks validity of address, seeks page frame
  - If selected frame is dirty, write it to disk
- OS brings scheduled new page in from disk
- Page tables updated
- Faulting instruction backed up to when it began
- Faulting process scheduled
- Registers restored and program continues

Instruction Backup

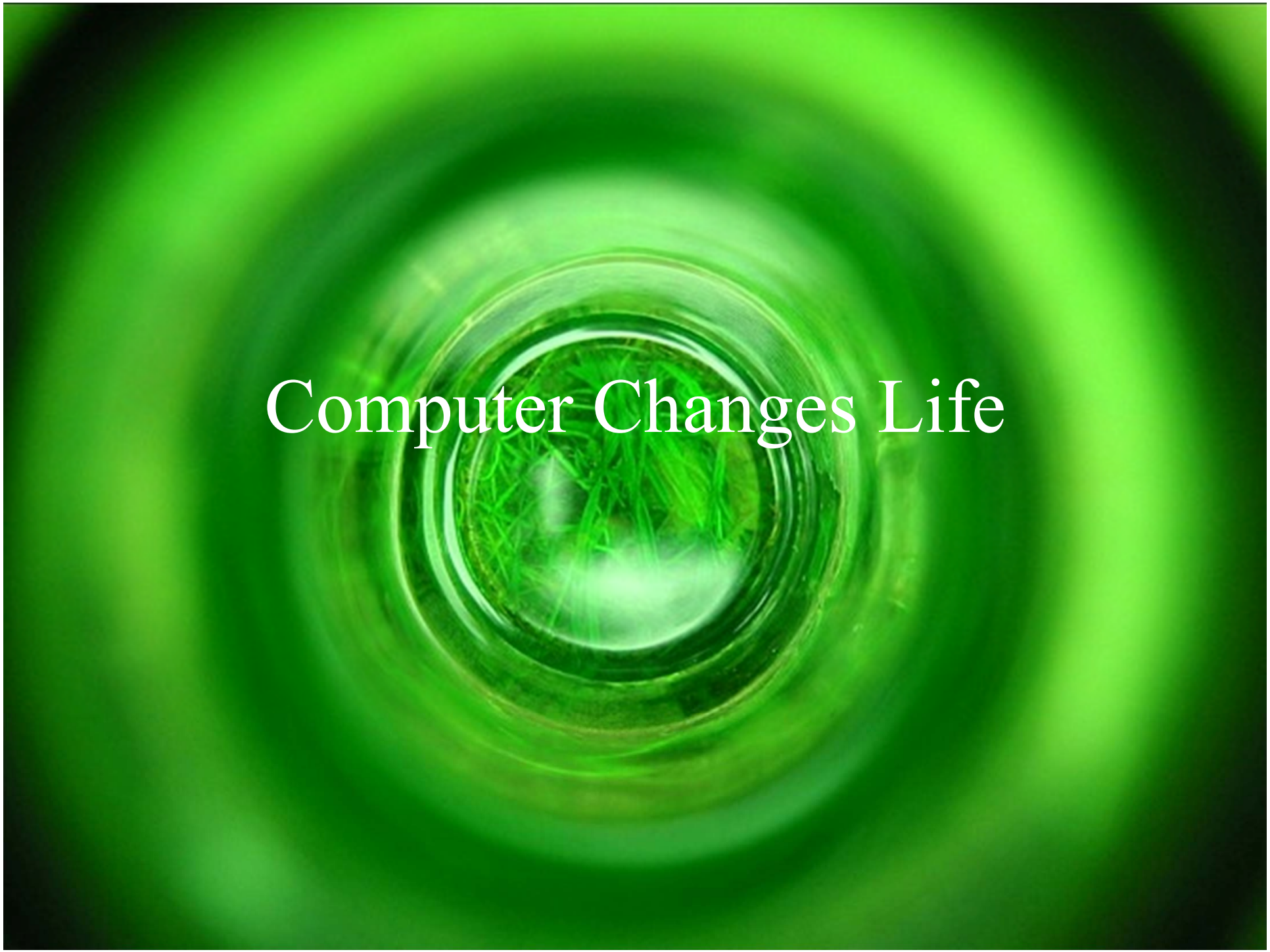


An instruction causing a page fault

Locking Pages in Memory



- Sometimes may need to lock a page in memory
  - i.e. prohibit its eviction from memory
- Proc issues call for read from device into buffer
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first proc may be chosen to be paged out
- Need to specify some pages locked
  - exempted from being target pages


Computer Changes Life