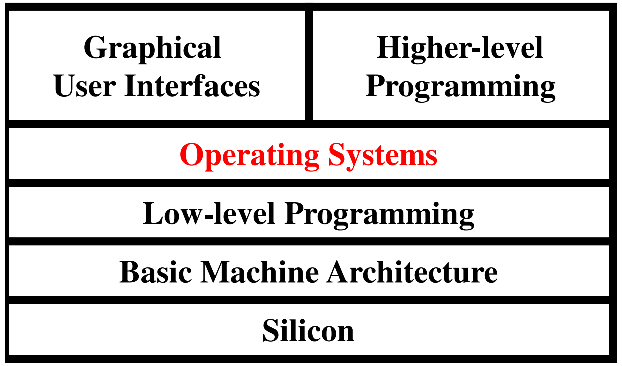


Graphical User Interfaces

Higher-level Programming

Operating Systems

Low-level Programming

Basic Machine Architecture

Silicon

CSCU9V4 Systems

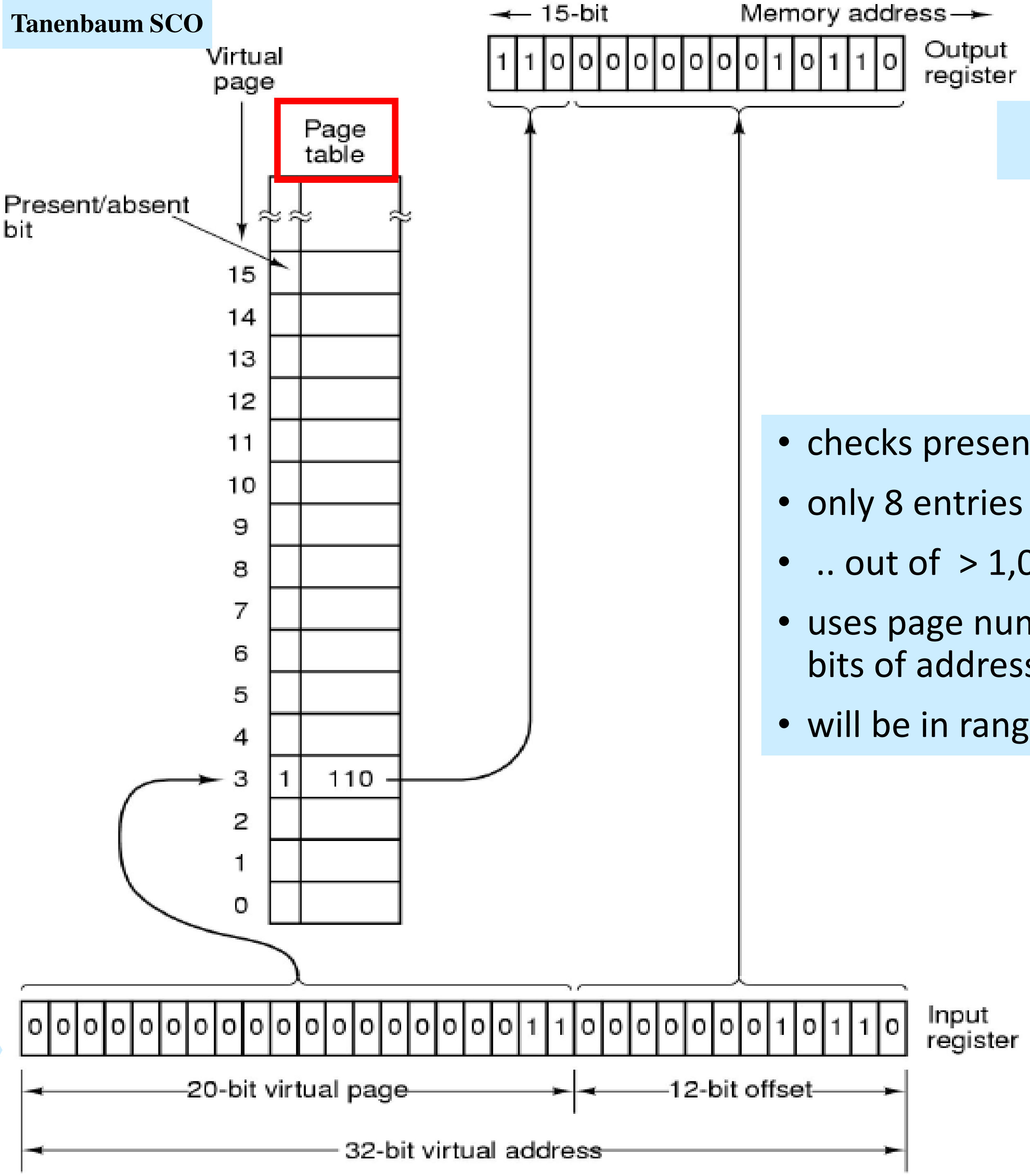
Systems lecture 15

Operating Systems 2

VM issues: Page sizes, Segmentation

implemented on a MMU

Tanenbaum SCO



15 bit P.A.

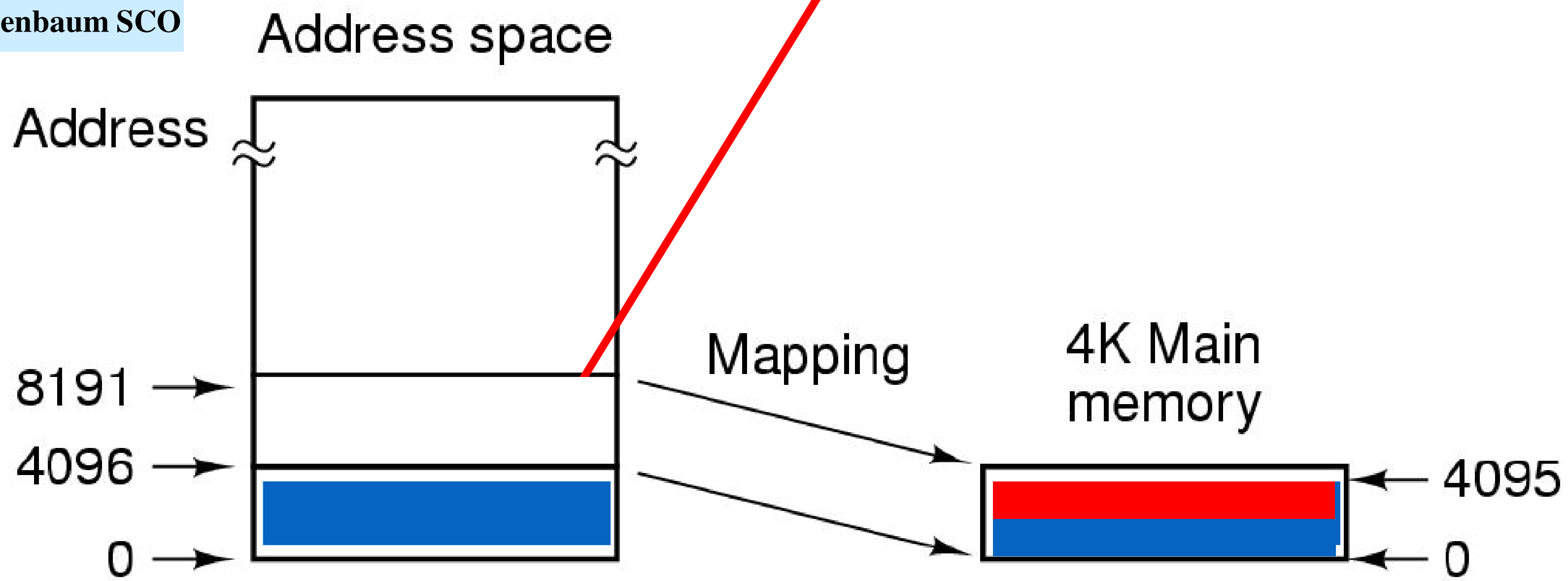
32 bit V. A.

- 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

What size should a page be?

- Fragmentation:
- assume a 4K pages
  - … so that is the finest resolution we can use
- … if we have a 5K program
  - … must ‘use’ 8K
- ***internal fragmentation***

Tanenbaum SCO



- **3 K ‘wasted’**

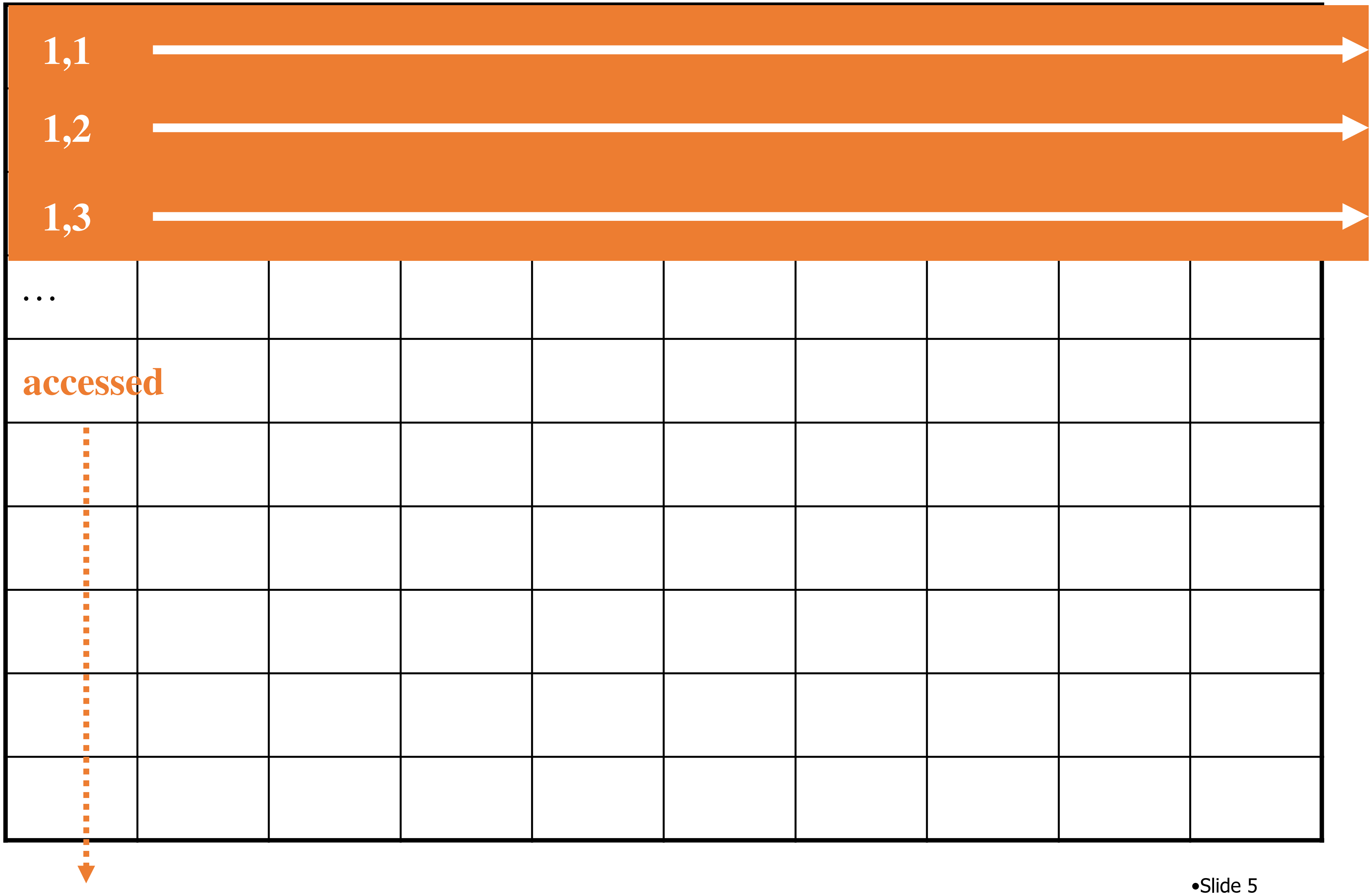
What size should a page be?

- smaller page size wastes less space.

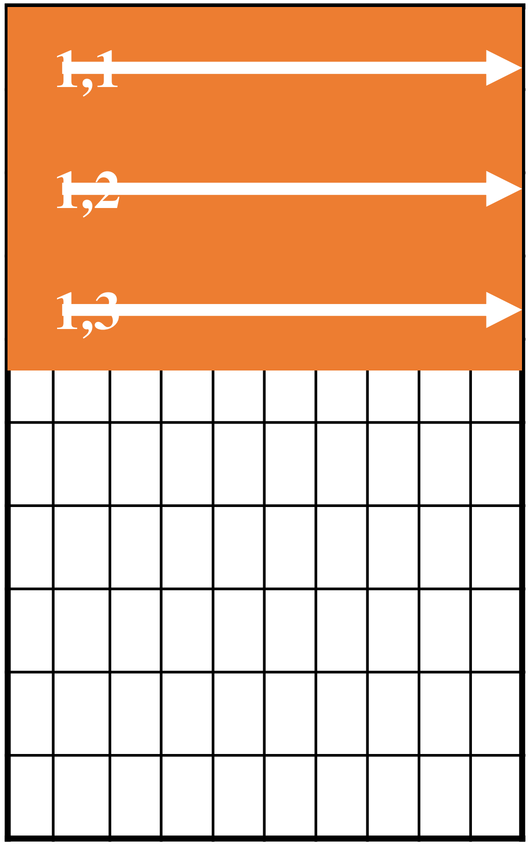
- if the page size is N bytes
  - the average wastage N/2 bytes
  - (Of course, we don’t actually waste nearly this much: mostly the blocks of virtual memory we bring into physical memory are much more than 1 page in size: we only waste this much of the last page)
- [probability of wasted space is the same for 1..(N-1)]
- … so best to have page sizes small ?

- Also…
- consider a 10,000 by 10,000 matrix of **8-byte floats** A[10000, 10000]
- say bytes of A stored in **column** order A[1,1], A[2,1], A[3,1] …

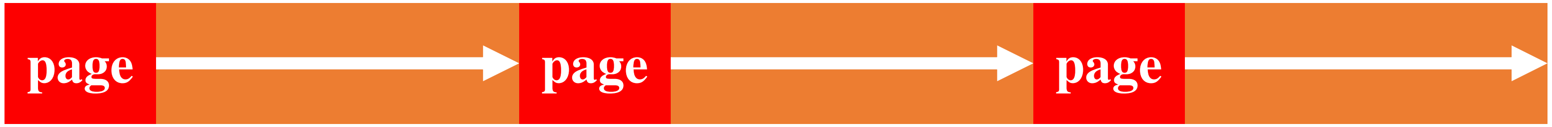
What size should a page be (3)



What size should a page be? (4)



- Let' s say that we carry out row-ordered calculations, (i.e. A[1,1], A[1,2], A[1,3] ….) inside a loop executed a few hundred times
  - the elements are 80,000 bytes apart
  - (i.e. row elements separated by 79,992 bytes!)



What size should a page be? (5)

- each element down a column requires a page to be brought in to physical memory from virtual memory
  - so with 8K pages we require 10,000 X 8K = 80Mbytes (approx)
  - i.e. the working set would be 80Mbytes

- if the page size was 1K, only 10Mbytes would be needed.

- so a 64M byte physical memory will thrash with 8K pages
- … but not with 1K pages!
  - … so again smaller pages seem better

- (How realistic is this? Huge array – 10,000,000,000 elements long)

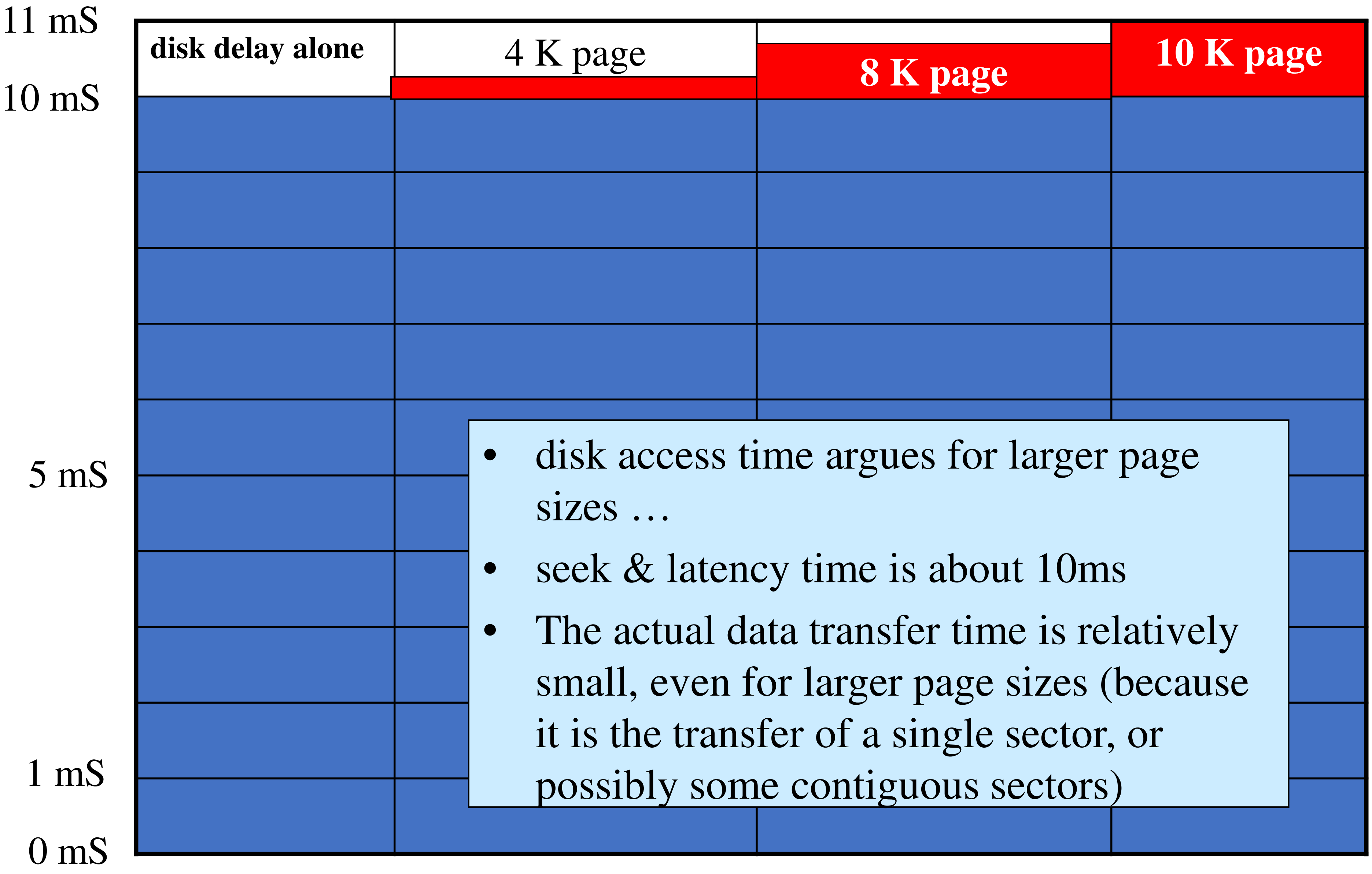
  - …. but ..

What size should a page be (6)

- There's a question of the efficiency of moving pages from virtual to physical memory.
- disk access time argues for larger page sizes
- seek & latency time is about 10ms
- data rates from disk controller are above 10Mbytes per second
- … or better than 1mS for each 10Kbytes

- Further, efficiency for disc reads/writes suggests that sectors on disk be quite large
  - Recall whole sectors are read or written at a time
- And efficient transfer between disc/main memory suggests page size be the same as sector size.

What size should a page be (7)



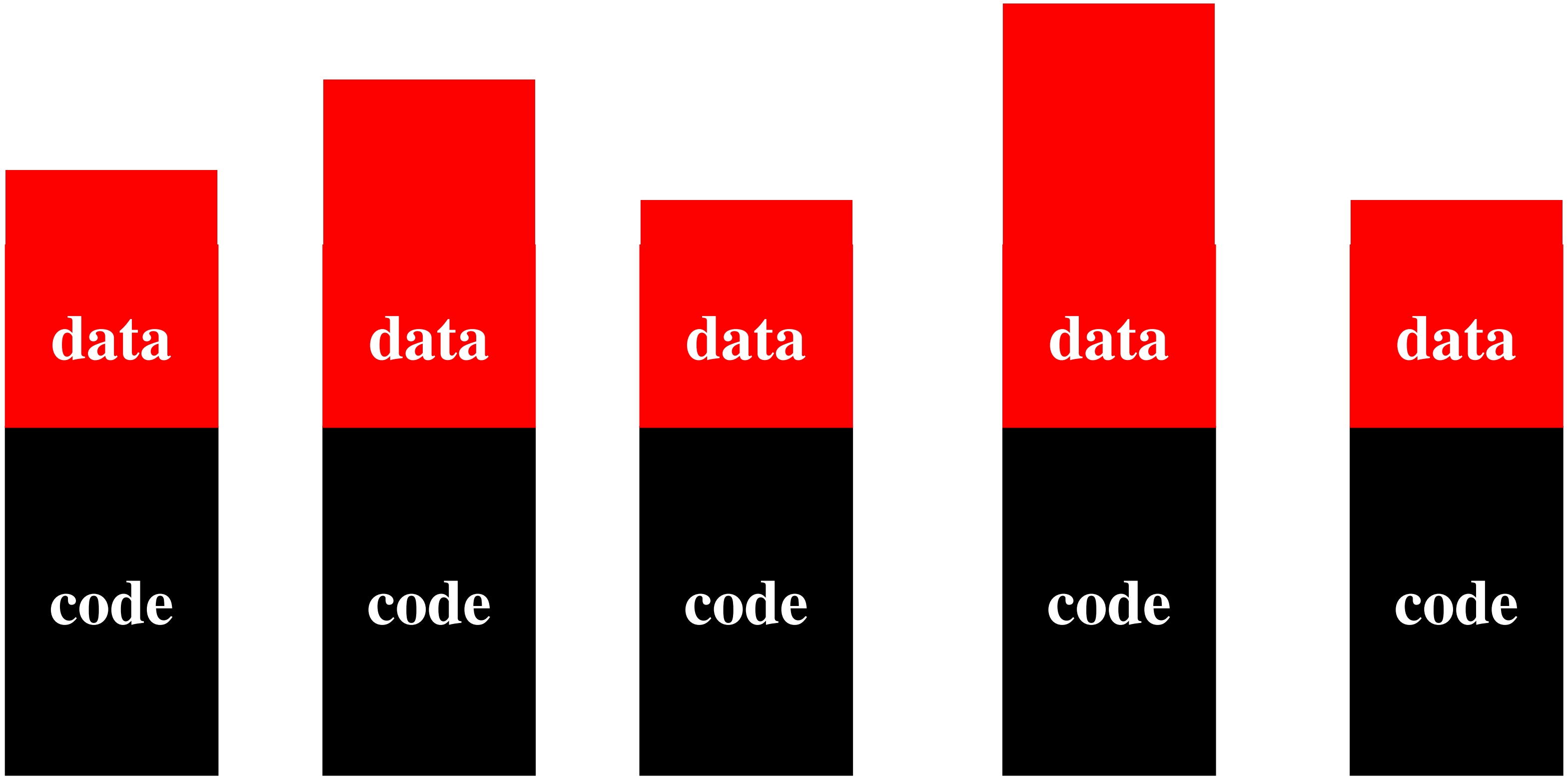
- disk access time argues for larger page sizes …
- seek & latency time is about 10ms
- The actual data transfer time is relatively small, even for larger page sizes (because it is the transfer of a single sector, or possibly some contiguous sectors)

Virtual Memory: Review.

- Virtual memory is now virtually universal for desk top machines
- Users do not need to worry about the total size of their programs
  - They can just keep starting up more, (so long as they fit in virtual memory).
- Efficient transfer between disk & memory argues for large page sizes
- Efficient memory use argues for smaller page sizes
  - IA64 architecture allowed for 4K or 4M pages!
- For a program to run efficiently, it is working set size that matters
- When the working set size for the programs in use exceeds the physical memory, the system runs very slowly indeed.
  - Thrashing occurs

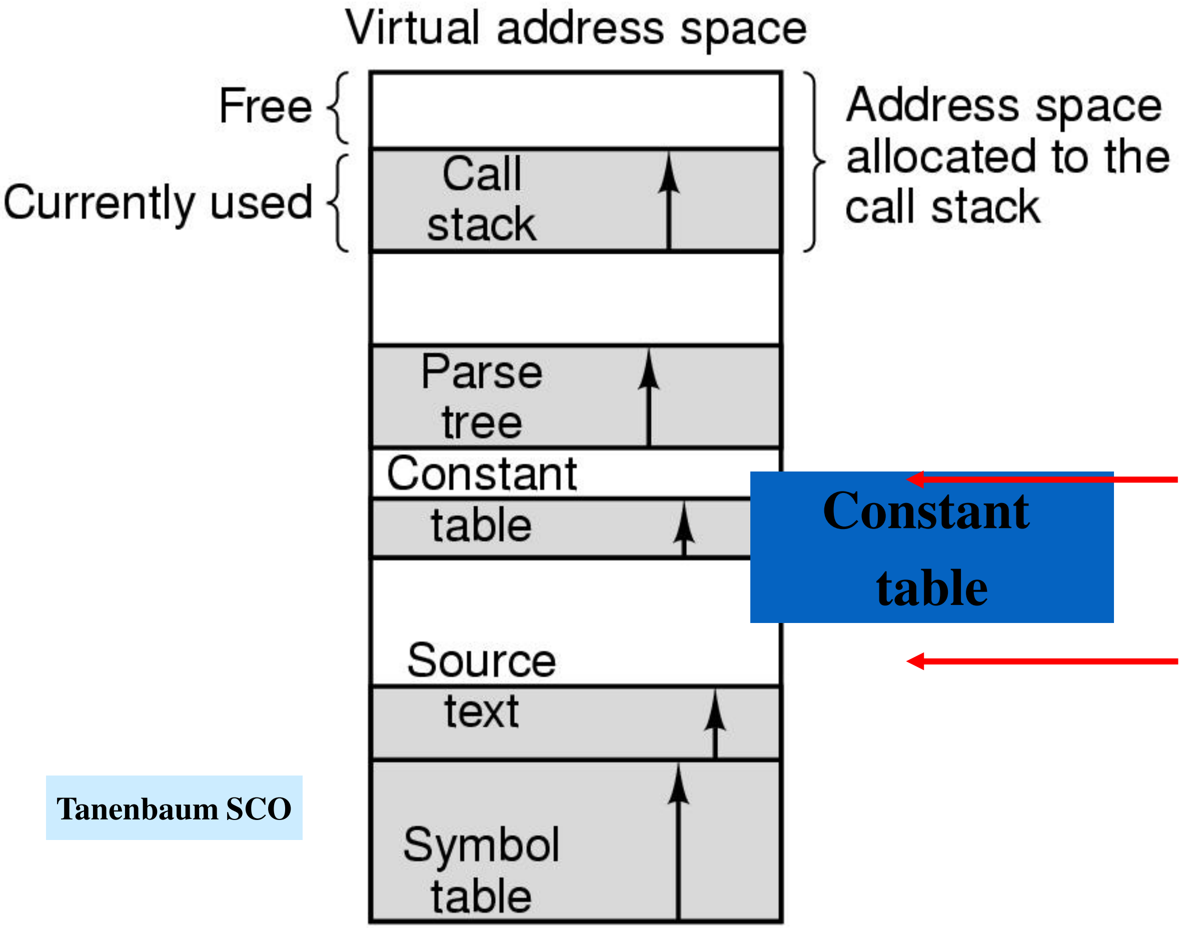
Segmentation (1)

- what if the a program has a dynamic volume of data as it runs?
- Program has code and data



Segmentation (2)

- if the data that is changing across a number of tables ….
- for example a compiler
- Or a program creating and destroying objects
- Or a program calling methods (or functions) which have a lot of local data

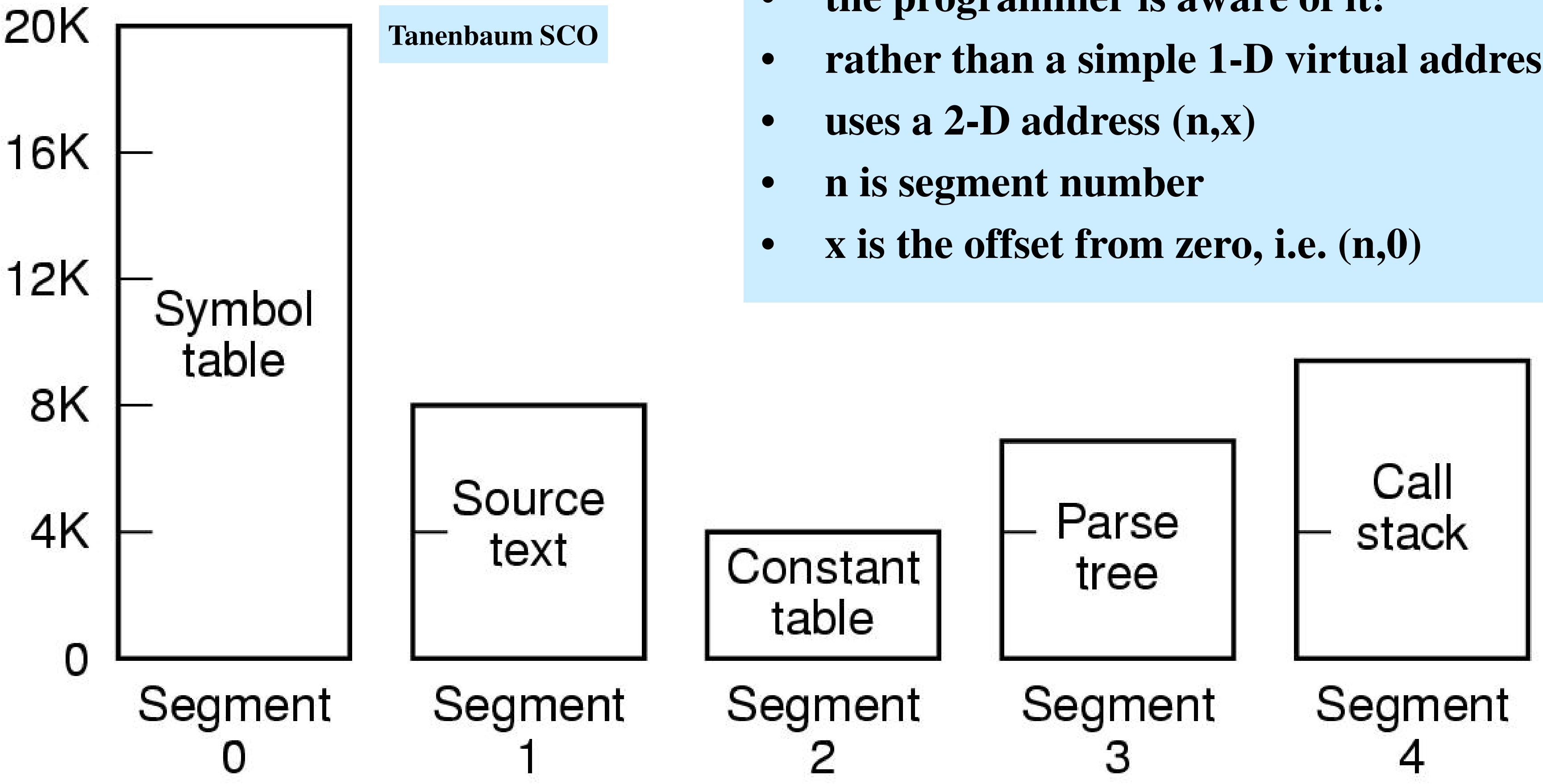


Tanenbaum SCO

Segmentation (3)

- better to split the data up ....
- ... into segments ...

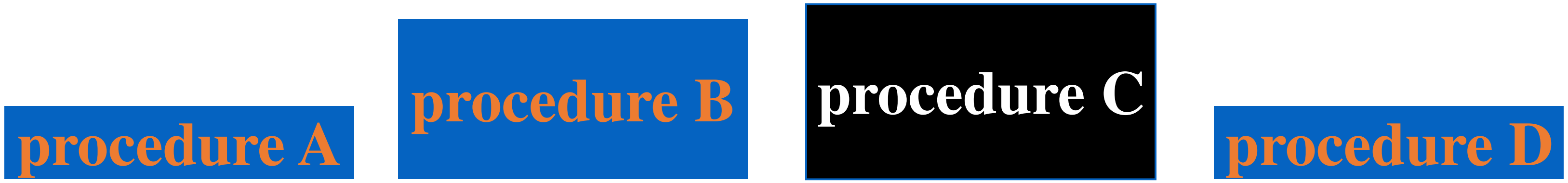
- each segment starts at zero
- to some max value
- shrink & grow independently
- the programmer is aware of it!
- rather than a simple 1-D virtual address
- uses a 2-D address (n,x)
- n is segment number
- x is the offset from zero, i.e. (n,0)



Tanenbaum SCO

Segmentation (4): *more advantages (1)*

- other advantages …
  - procedure linking is improved
  - can be compiled separately – all starting at address zero
  - do not need to compile the complete program



procedure A

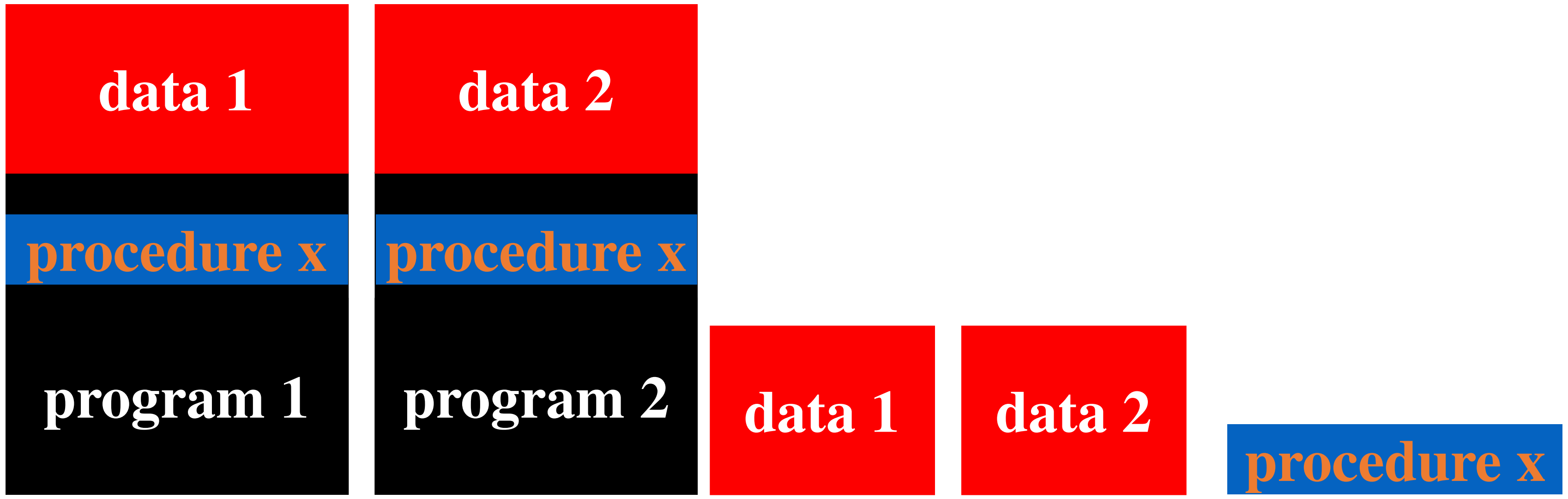
procedure B

procedure C

procedure D

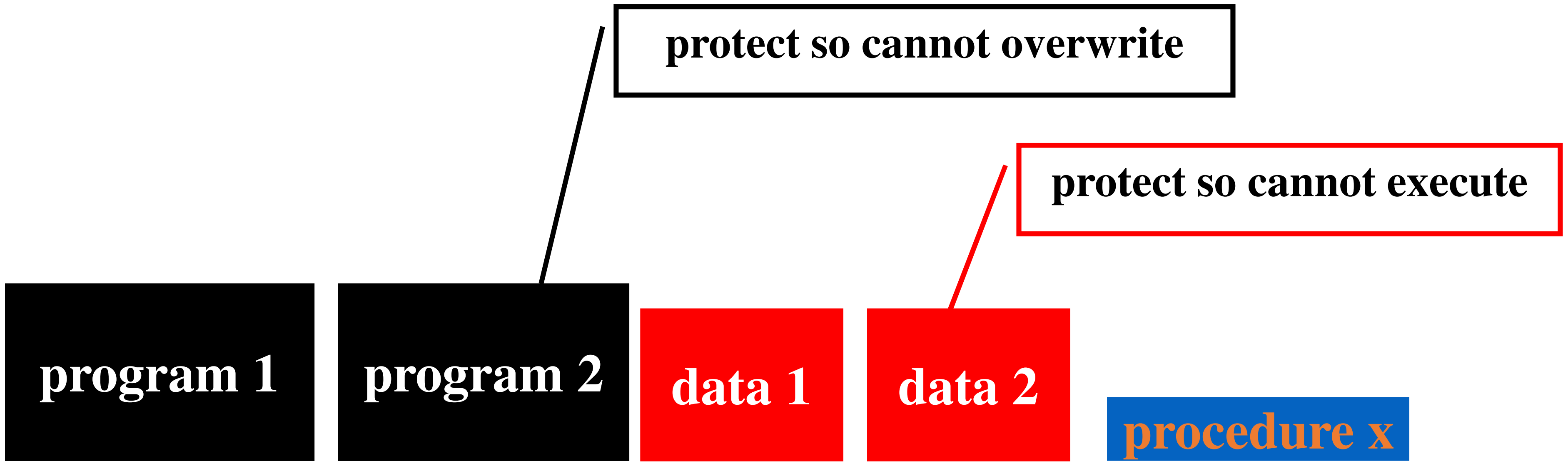
Segmentation (5): *more advantages (2)*

- other advantages …
  - can also share procedures between processes

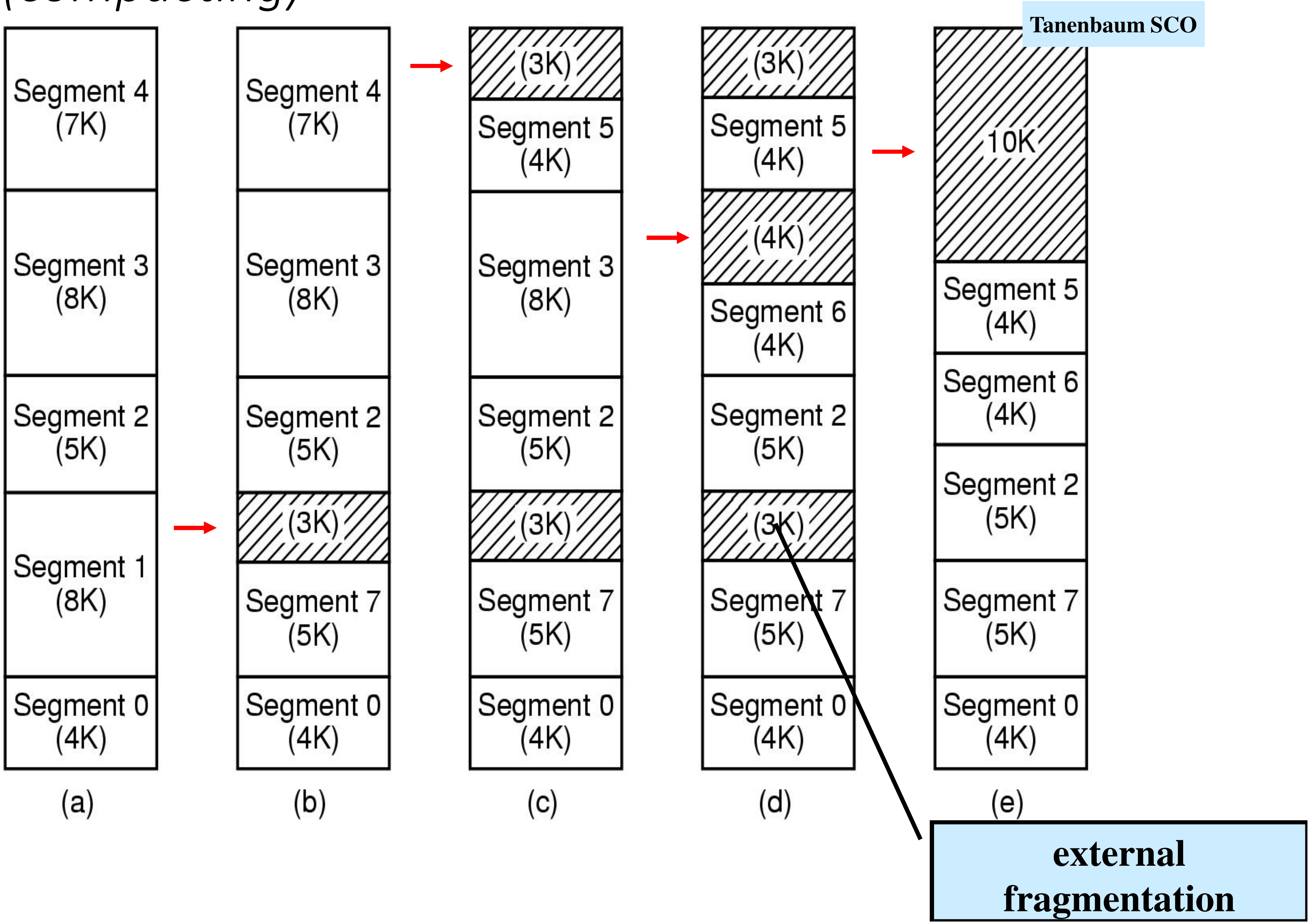


Segmentation (6): *protection*

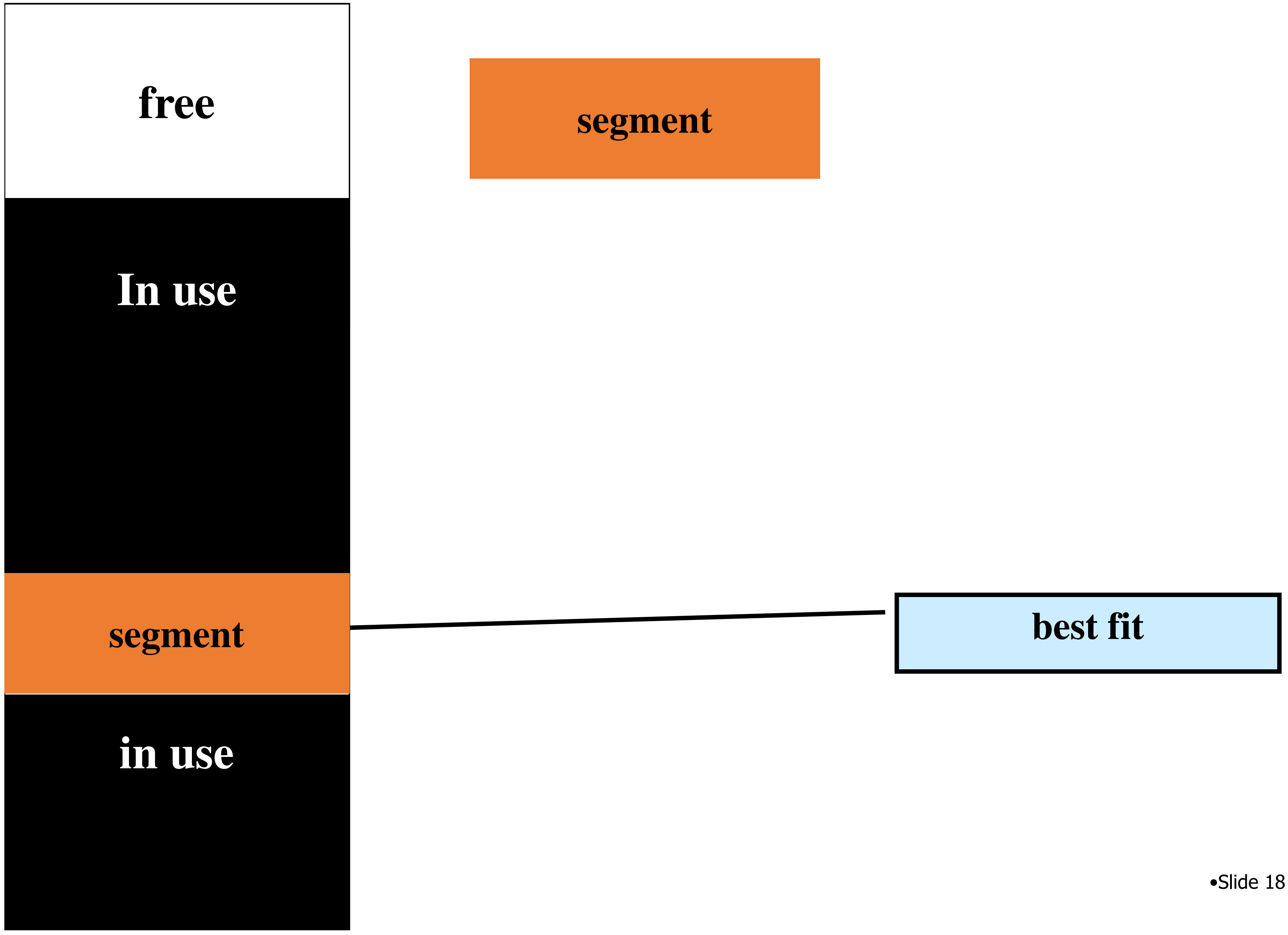
- yet another advantage …
  - the programmer is aware of each segment … so …
  - they can be of one type
  - so it can be *protected* as say read/write, and any attempt to execute it will fail … useful if program jumps to an incorrect address
  - And program can be protected as execute only, so any attempt to overwrite it will cause a fault.



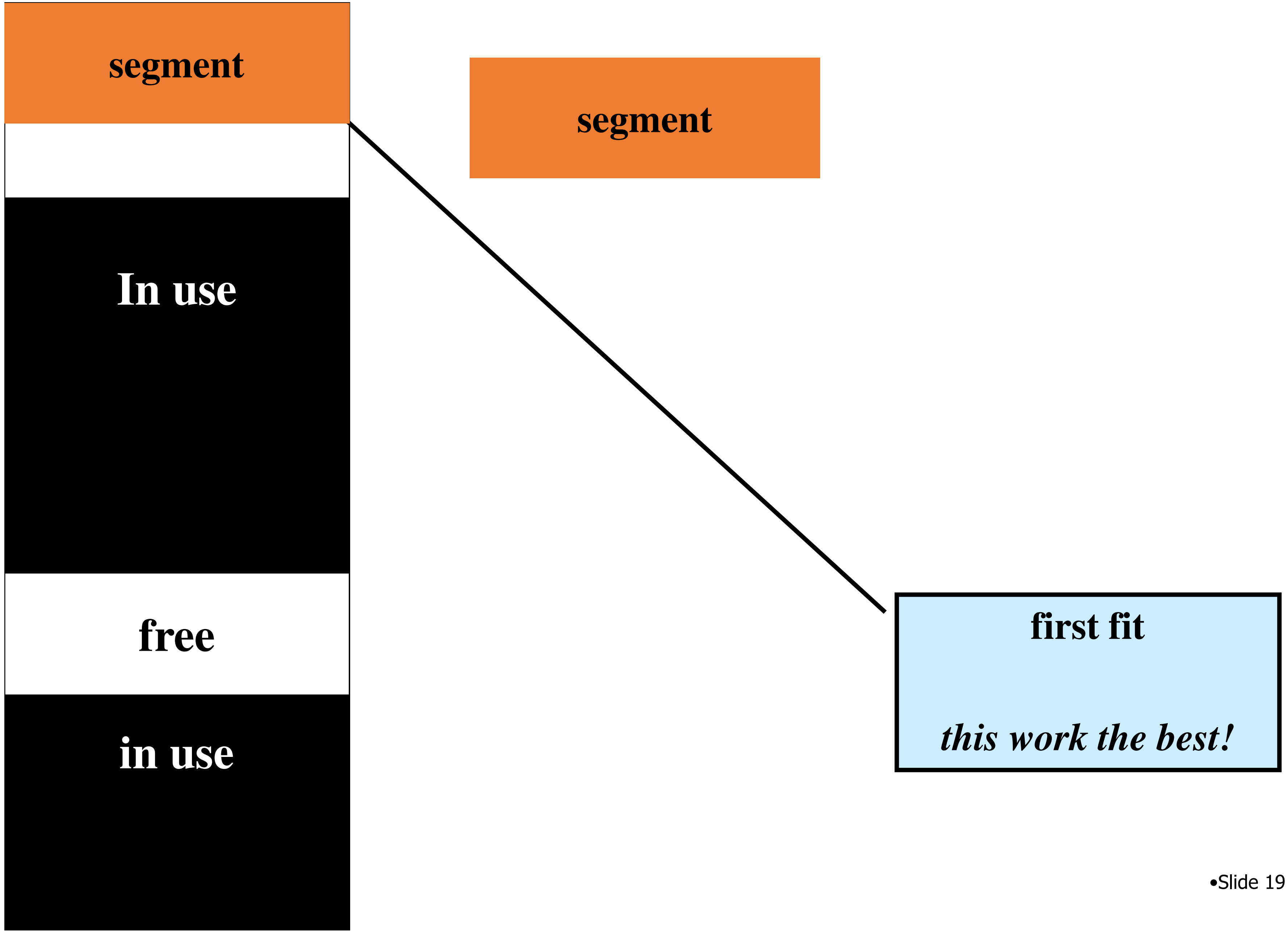
Segmentation (7): non-paged *implementation (compacting)*



Segmentation (8): *implementation (if no time to compact)*



Segmentation (8): *implementation (if no time to compact)*



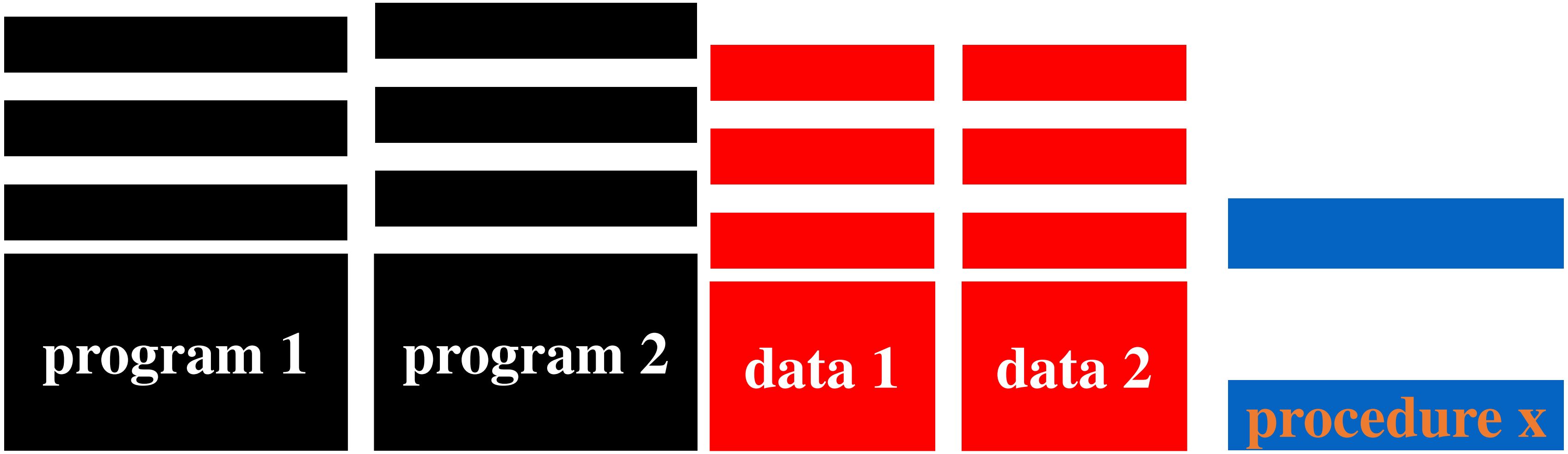
first fit

this work the best!

Segmentation (9): *implementation (paging)*

- Alternatively - each segment can be split into pages
  - each segment is a 1-D memory space
  - exactly as we saw before
  - so we can apply the same techniques as before

- common to have segmented paging arrangements
  - Intel
  - still a 2-D view to programmer – paging is transparent (as before)



End of lecture