

Split memory into smaller chunks

Rather than moving memory around to make big enough spaces for processes we could have more than one section of physical memory accessed by the same process.

We need either a number of base-limit registers or a table of the information.

Chunk	base	limit
0	1024	1024
1	8192	512
2	4096	2048



The process sees
3.5K of contiguous
memory.

Two approaches

This technique evolved in two directions.

2. Same sized chunks – pages
3. Variable sized chunks – segments

Both have advantages and disadvantages.

Both use Memory Management Units (MMUs) in hardware to do the translation between the logical and the physical address.

Rather than doing a tedious calculation (where is logical address 2000 on the previous slide?) to find what chunk an address is in, we just split the address into two parts.

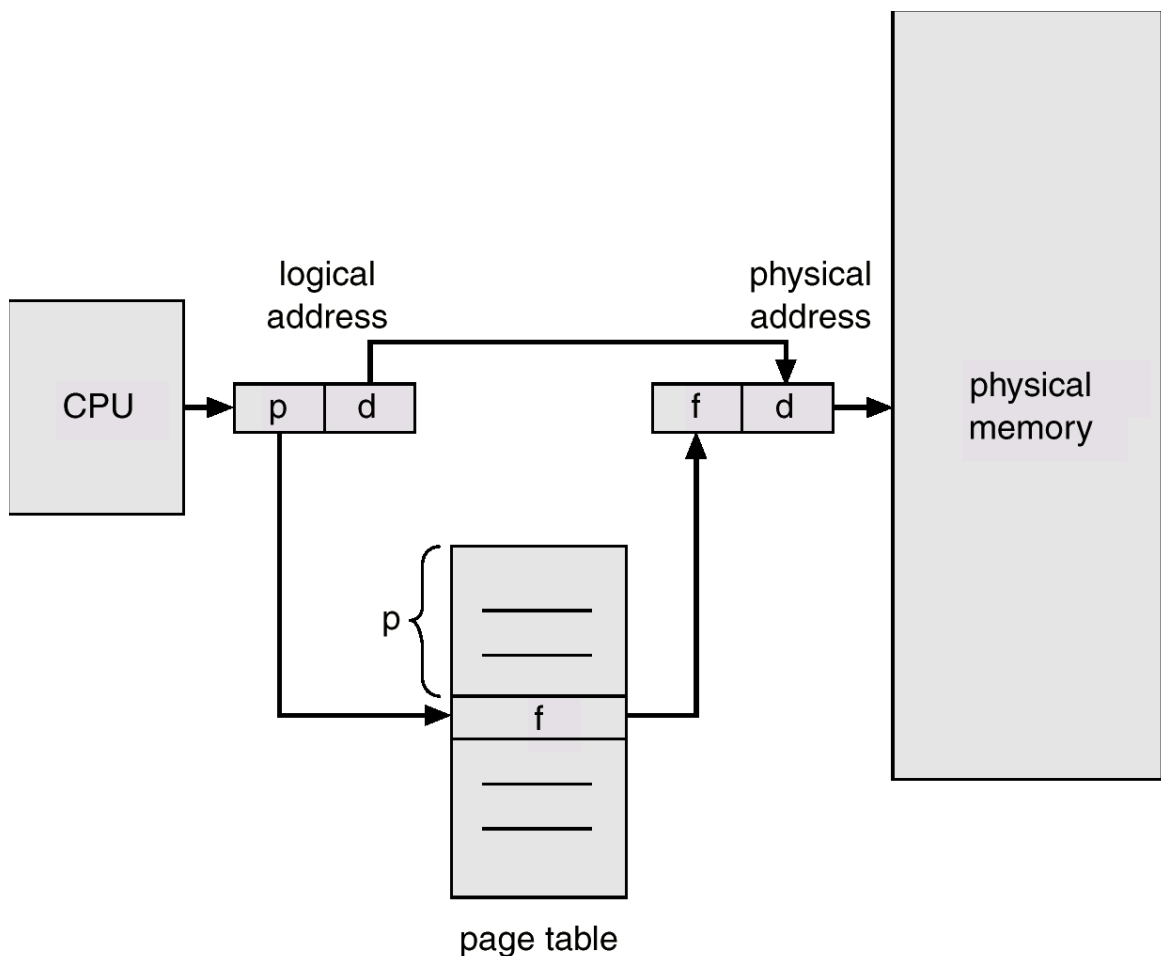
Then the translation is much simpler and looks very similar in both paged and segmented systems.

Paged system address translation

Logical address is divided into:

Page number – index into a *page table* which contains base address of each page in physical memory

Page offset – added to base address get the physical address.



In this case there is a constant number of bits for the offset. This means that pages are always powers of 2 in size – usually from 2048(11 bits) to 8192(13 bits).

Some systems allow variable sizes of pages.

Pentium variable sized pages

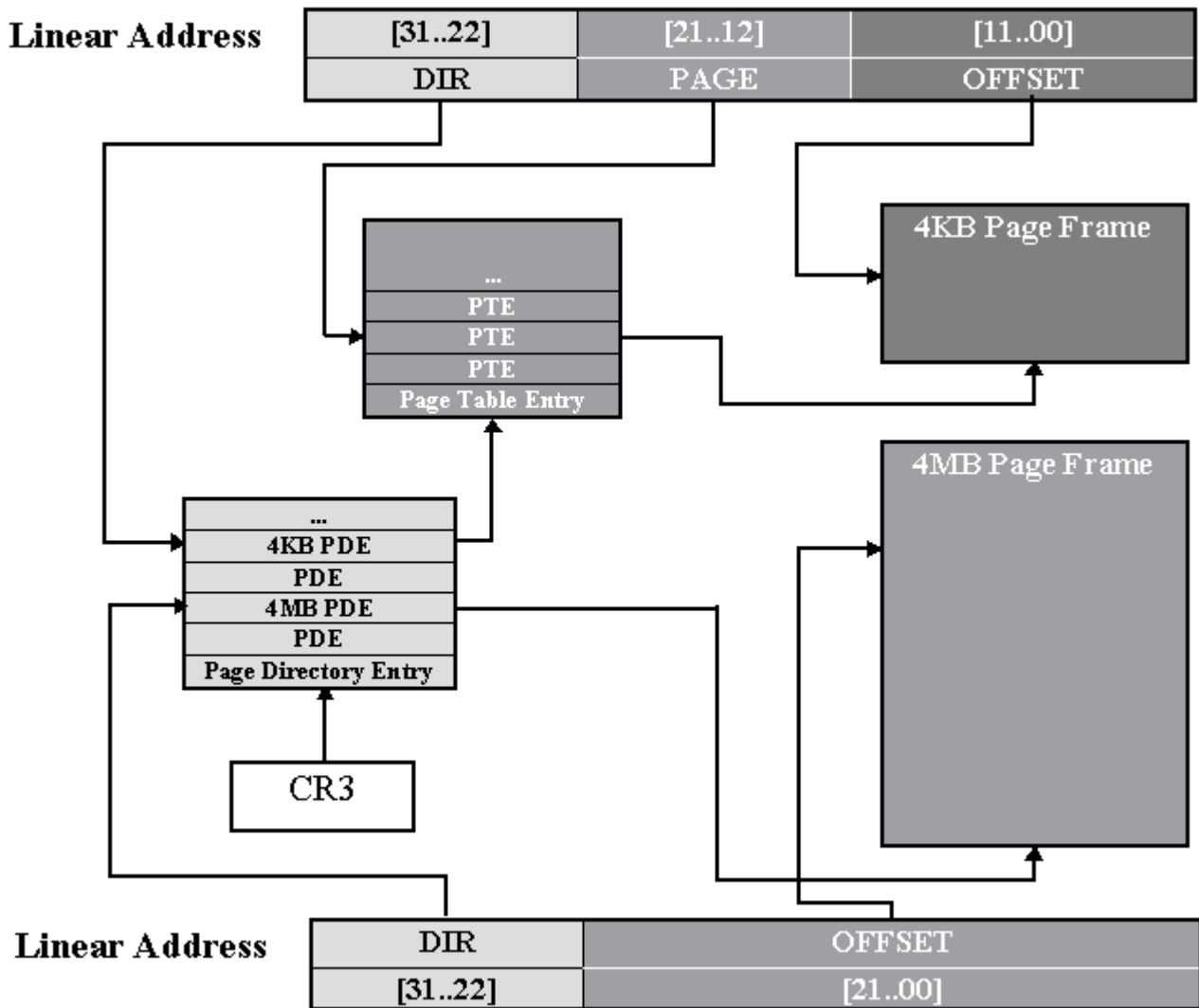


Diagram from <http://www.x86.org/articles/4mpages/4moverview.htm>

Frames and pages

The textbook makes a distinction between pages and frames.

A frame is a page sized chunk of physical memory that can be allocated to a page from a process.

A page is a frame sized chunk of logical memory that fits in a frame.

It is common to refer to both simply as pages (physical and logical).

Fragmentation

- No external fragmentation between pages.

- Internal fragmentation in any pages that are not completely used.

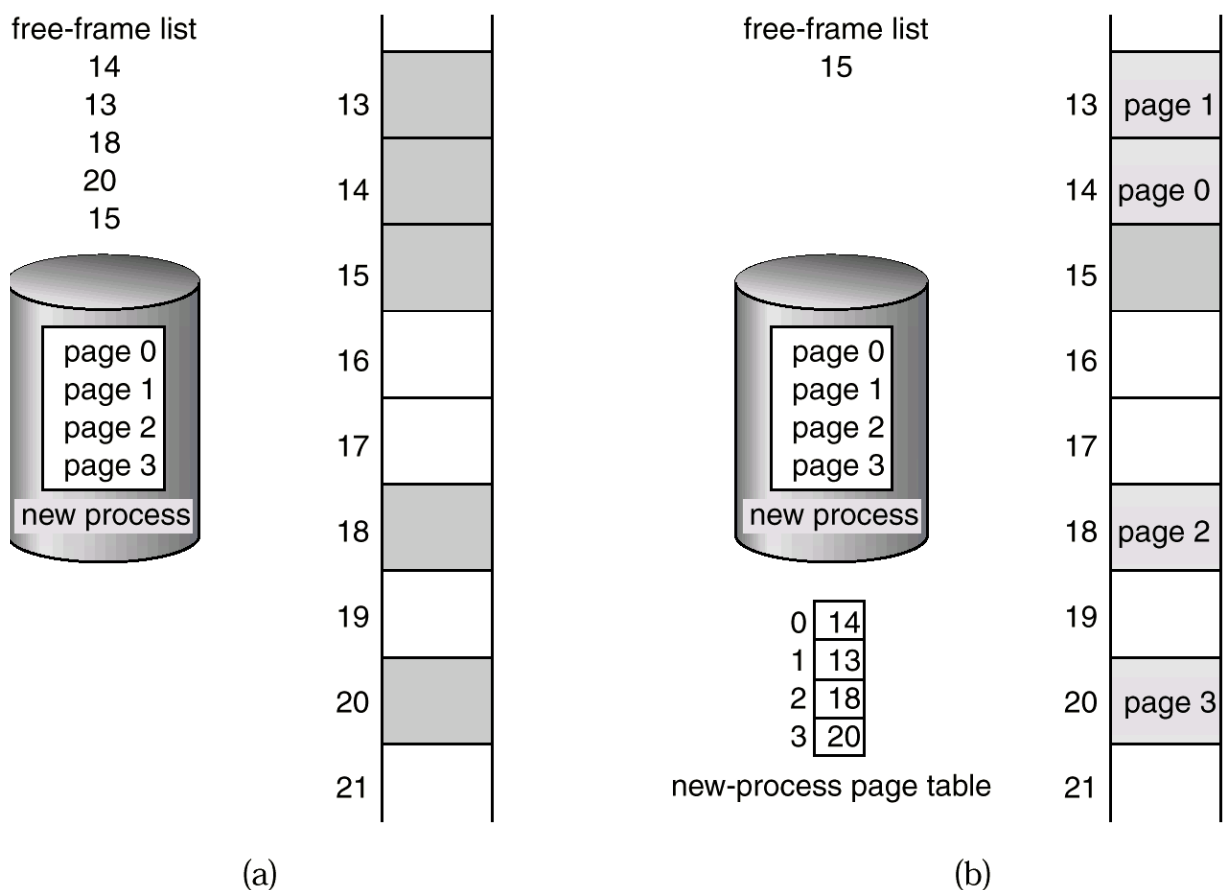
- On average $2 \times \frac{1}{2}$ a page per process (or $\frac{1}{2}$ a page per thread plus $\frac{1}{2}$ a page for heap space).

- So small pages save space but require larger page tables

Tables

Each process has its own page table.

And commonly the OS has a frame table with one entry for each frame showing whether it is free or not and which process is currently using it.



Different sized chunks

Rather than constant sized pages we could design our hardware to work with variable sized chunks – these are known as segments. (Not to be confused with variable sized pages.)

Memory model

How memory appears to be organised to the program (and programmer) is sometimes referred to as the memory model.

A segmented memory model is when we look at memory as a group of logical units all with addresses starting at zero.

Processes can have lots of segments

- code modules
- functions
- global variables
- activation records
- large data structures (arrays)
- objects

Segments

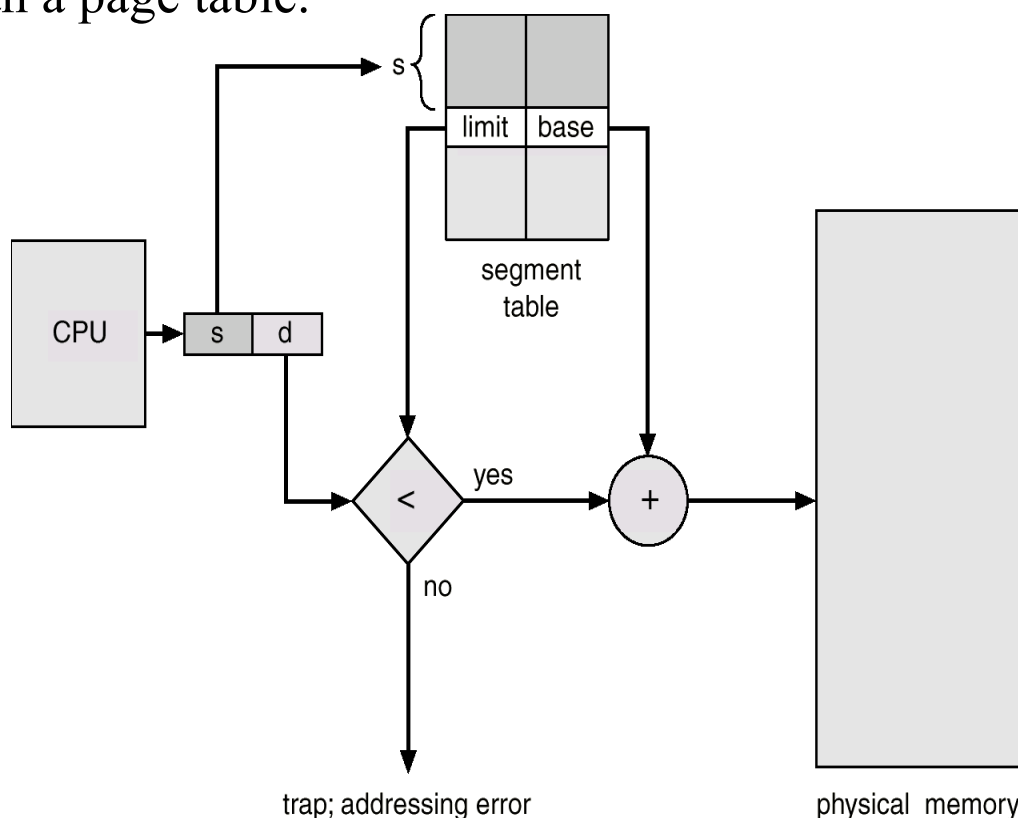
These logical units fit nicely into hardware specified segments where different amounts of memory can be allocated in one chunk.

Segments are contiguous blocks of memory.

We will get problems of external fragmentation.

Our memory addresses become
<segment name, displacement>.

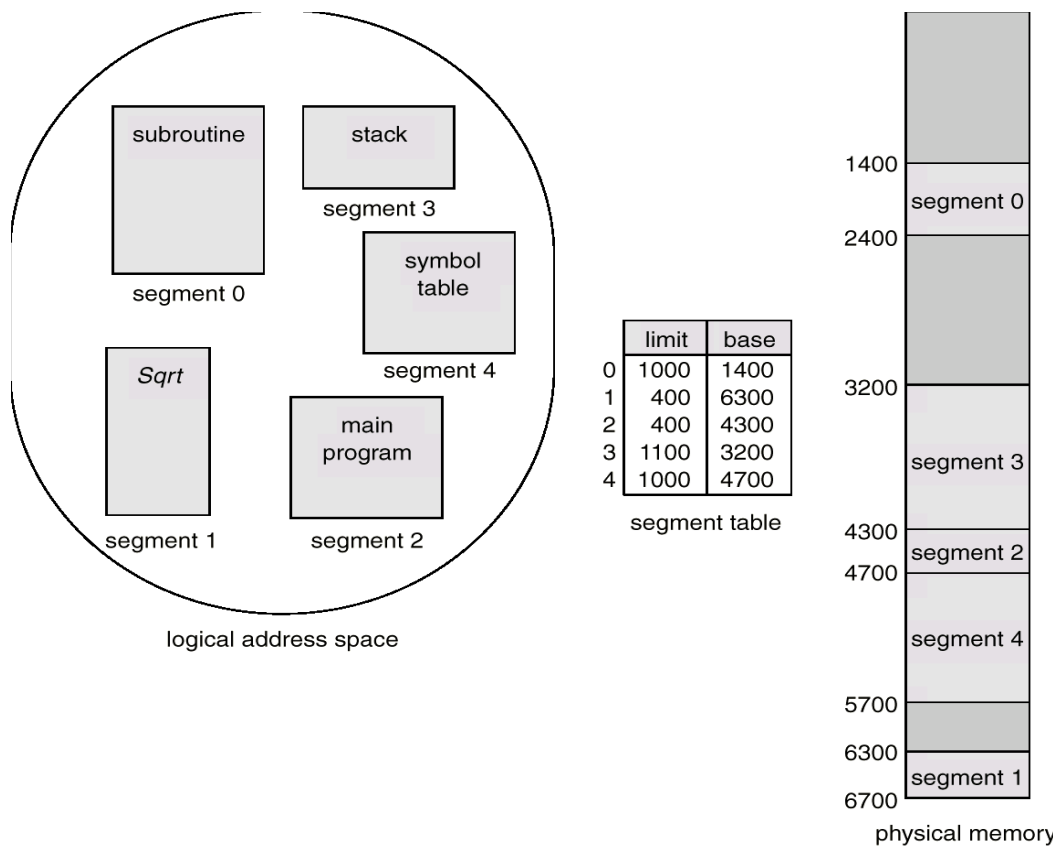
And the translation process looks very much like paging, except there is a length associated with each segment. We have a segment descriptor table rather than a page table.



Example

Ideally segments should be able to cover all of the addressable memory – this could mean that logical addresses might more bits than physical address.

Some segmented memory systems restrict segment sizes to prevent this.



Allocation strategies

Segments have no internal fragmentation – we only allocate the amount of space we want.

What is one obvious problem with this?

But we get external fragmentation.

We have seen the allocation strategies before:

- first fit
- next fit (first fit but starting from where we were up to)
- best fit
- worst fit

We can defragment memory if we want to find large enough chunks. Faster than defragmenting disks.

How much space in the holes

Knuth's 50% rule

If there are n segments there are $n/2$ holes.

Each segment is either below another segment or below a hole.
We always combine adjacent holes.



Each segment is released independently – so in a steady state system the space above each segment will alternate between being a hole and being free. 50% of the time there will be a hole above each segment.

If the average size of a hole is the same as the average size of a segment we need about $1/3$ of the memory free to keep this system running.

Half speed memory

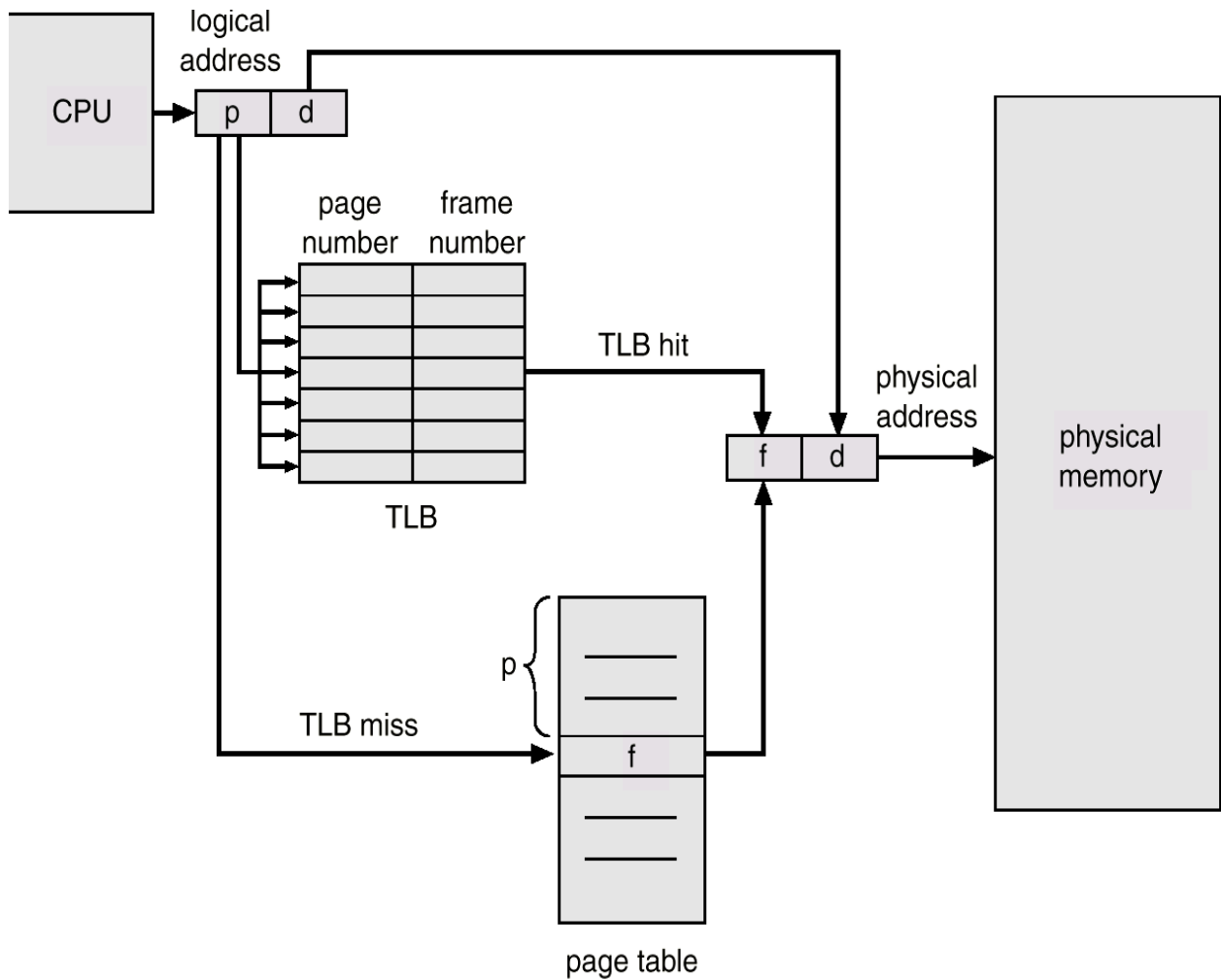
In both paged and segmented memories every logical memory access requires (at least) two memory accesses. One for the page/segment table and one for the actual data.

Actually the number of segments may be quite small and there may be registers for them.

So the MMUs cache recent page table information in a special fast-lookup hardware cache called *associative registers* or *translation look-aside buffers (TLBs)*

Page #	Frame #

TLB use



Average access times

TLB Lookup = ϵ time unit

Assume memory cycle time is β

Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.

Hit ratio = α

Effective Access Time (EAT)

$$\begin{aligned} \text{EAT} &= (\beta + \epsilon) \alpha + (2\beta + \epsilon)(1 - \alpha) \\ &= 2\beta - \alpha\beta + \epsilon \end{aligned}$$

e.g. $\alpha = 0.98$, $\beta = 1$, $\epsilon = 0.1$

EAT = 1.12 (compared to 2 for no TLB)

TLB coverage

TLB coverage is the amount of the address space that included in the TLB entries.

Typical TLB caches hold about 128 entries.

With 8K pages this is only a megabyte of memory.

As working sets (more on those later) increase this means lots of processes have a real performance hit, memory wise.

The solution is larger page sizes. This means more internal fragmentation. More IO (in virtual memory systems).

Variable page sizes can be used but they need clever allocation algorithms to be worthwhile.

Before next time

Read from the textbook

- 9.1 – Background
- 9.3 – Contiguous Memory Allocation
- 9.4 – Paging
- 9.5 – Segmentation