Working set model

Process' behavior

- Locality of reference: most of the time the last *k* references are within a finite set of pages < a large address space
- The set of pages a process is currently using is called the *working set* of the process
- Knowing the working set of processes we can do very sophisticate things (e.g. pre-paging)

Definition of working set

We define the working set of information $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t-\tau, t)$.

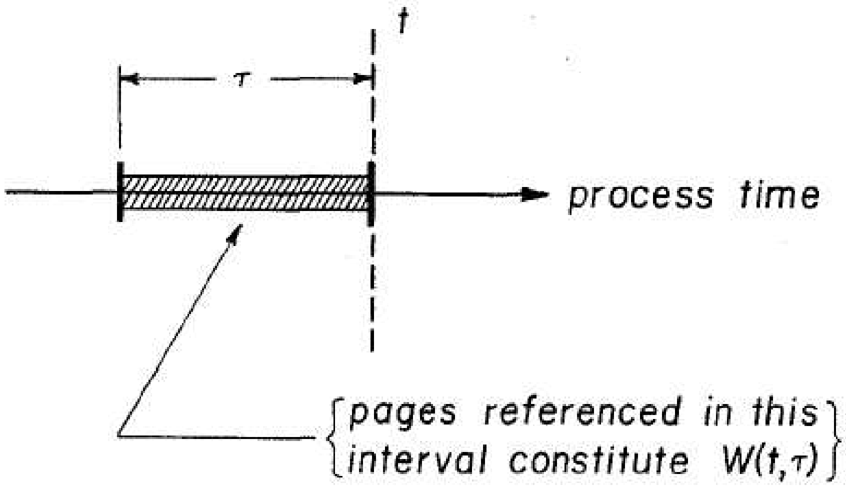


FIG. 2. Definition of $W(t, \tau)$

More definitions

- τ is the working set parameter
- The elements of $W(t, \tau)$ are pages but they can be anything else used by a process (any unit of information)

Working set size

- $w(t, \tau)$ = number of pages in $W(t, \tau)$.

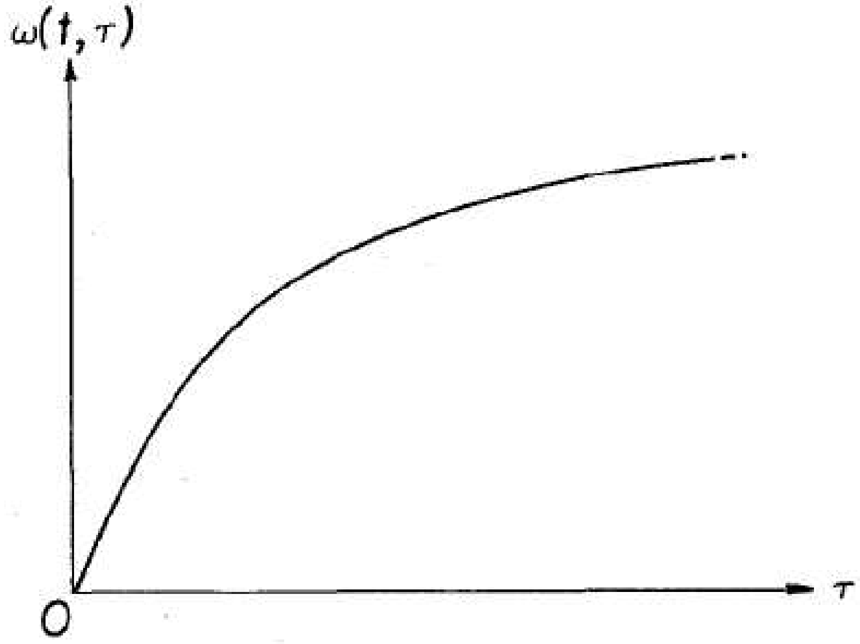


FIG. 3. Behavior of $\omega(t, \tau)$

Properties of the WS

- P1, size,
  - $w(t,0)=0$
  - Monotonically increasing & concave downward (see previous slide)

- P2, prediction
  - $W(t, \tau)$ is a good predictor of $W(t+\alpha, \tau)$ for $\alpha<\tau$

Benchmarking the WS

- P3, reentry rate

$$\lambda(\tau) = \frac{1 - F_x(\tau)}{\overline{x}} \quad \text{where} \quad F_x(\tau) = \Pr\{x \le \tau\} \quad \text{and} \quad \overline{x} = \int_0^{\infty} t \cdot f_x(t)\, dt$$

Average process time between references to same page

T is the time required to transfer a page from auxiliary memory (disk) to main memory, Φ is the traffic rate (total)

$$\Phi(\tau) = \frac{\beta M \lambda(\tau)}{1 + \lambda(\tau) T} \quad \text{where} \quad \beta \quad \text{fraction of pages used by all WS}$$

M is the total number of pages
This is an equilibrium situation which assumes some traffic which is bidirectional, thus the bus should have a bandwidth of at least $2\Phi(\tau)$

Benchmarking the WS

- P4, sensitivity to changes in τ

$$\sigma(\tau) = -\frac{d}{d\tau}\lambda(\tau) = \frac{f_x(\tau)}{\overline{x}}$$

That is, if τ is decreased by $d\tau$, $\lambda(\tau)$ increases by $\sigma(\tau)\, d\tau$
$\sigma(\tau) >= 0$; reducing τ can never result in a decrease in the reentry rate $\lambda(\tau)$.

Choice of τ

- To choose τ we need to consider:
  - M
  - T
  - When τ is small, check Φ (estimate starting from x) to make sure we're in a reasonable range
  - When τ is large, need to determine this starting from the desired number of WS in memory

Residency

- How long a page is in memory

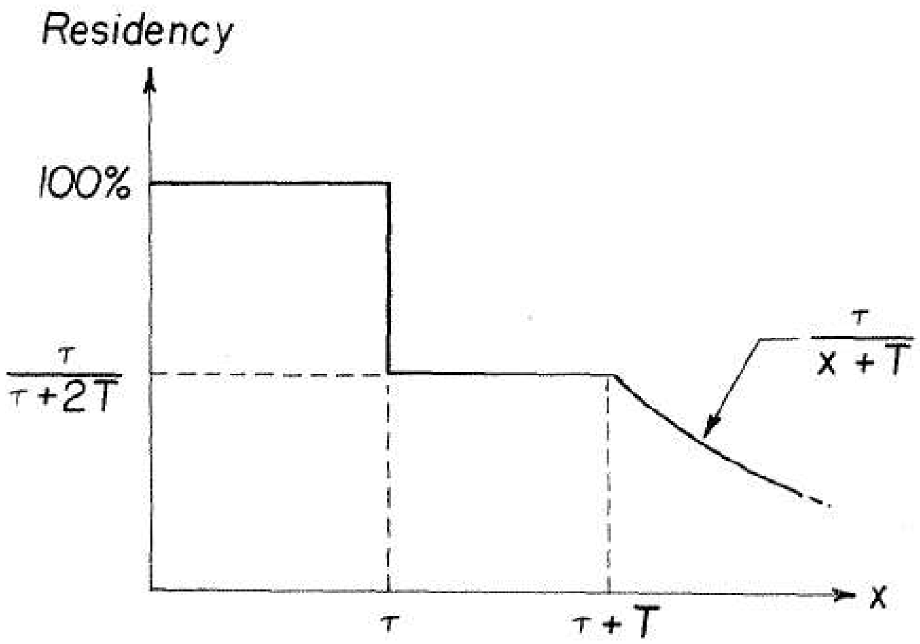


Fig. 4. Residency

Measuring the WS

- In theory it can be done in HW with timers
- In practice (software), sampling the page table

$$sample = \frac{\tau}{K}$$

K determines the grain of the sampling

Implementation

As indicated by Figure 5, each page table entry contains an "in-core" bit M, where $M = 1$ if and only if the page is present in main memory. It also contains a string of *use bits* $u_0, u_1, \cdots, u_K$. Each time a page reference occurs $1 \rightarrow u_0$. At the end of each sampling interval σ, the bit

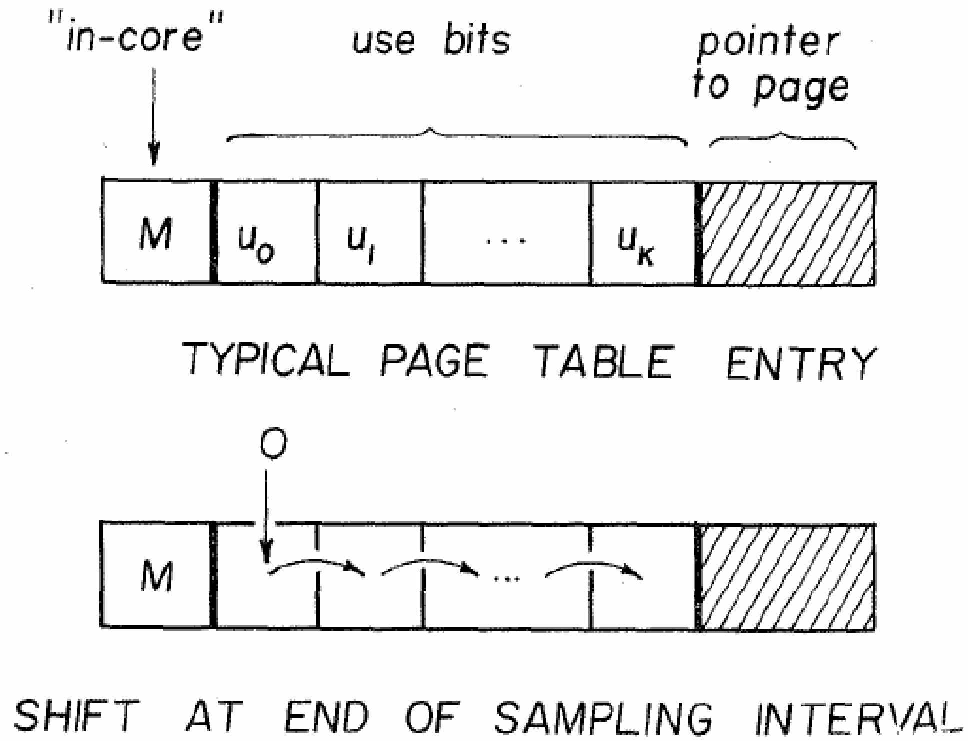


FIG. 5. Page table entries for detecting $W(t, K\sigma)$

Implementation

pattern contained in $u_0, u_1, \cdots, u_K$ is shifted one position, a 0 enters u_0, and u_K is discarded:

$$
\begin{aligned}
u_{K-1} &\longrightarrow u_K \\
&\vdots \\
u_0 &\longrightarrow u_1 \\
0 &\longrightarrow u_0 \; .
\end{aligned}
\tag{15}
$$

Then the logical sum U of the use bits is computed:

$$U = u_0 + u_1 + \cdots + u_K \, , \tag{16}$$

so that $U = 1$ if and only if the page has been referenced during the last K sampling intervals; of all the pages associated with a process, those with $U = 1$ constitute its working set $W(t, K\sigma)$. If $U = 0$ when $M = 1$, the page is no longer in a working set and may be removed from main memory.

WS based algorithm

- Store time information in the table entries
- At clock interrupt handle R bits as usual (clear them)
- At page fault, scan entries:
  - If R=1 just store current time in the entry
  - If R=0 compute "current-last time page was referenced" and if > *threshold* the page can be removed since it's no longer in the working set (not used for *threshold* time)
- **Note**: we're using time rather than actual memory references

WSClock algorithm

- Use the circular structure (as seen earlier)
- R=1, page in the WS – don’t remove it
- R=0, M=0 no problem (as before)
- M=1, schedule disk write appropriately to procrastinate as long as possible a process switch
  - No write is schedulable (R=1 always), just choose a clean page

Segmentation

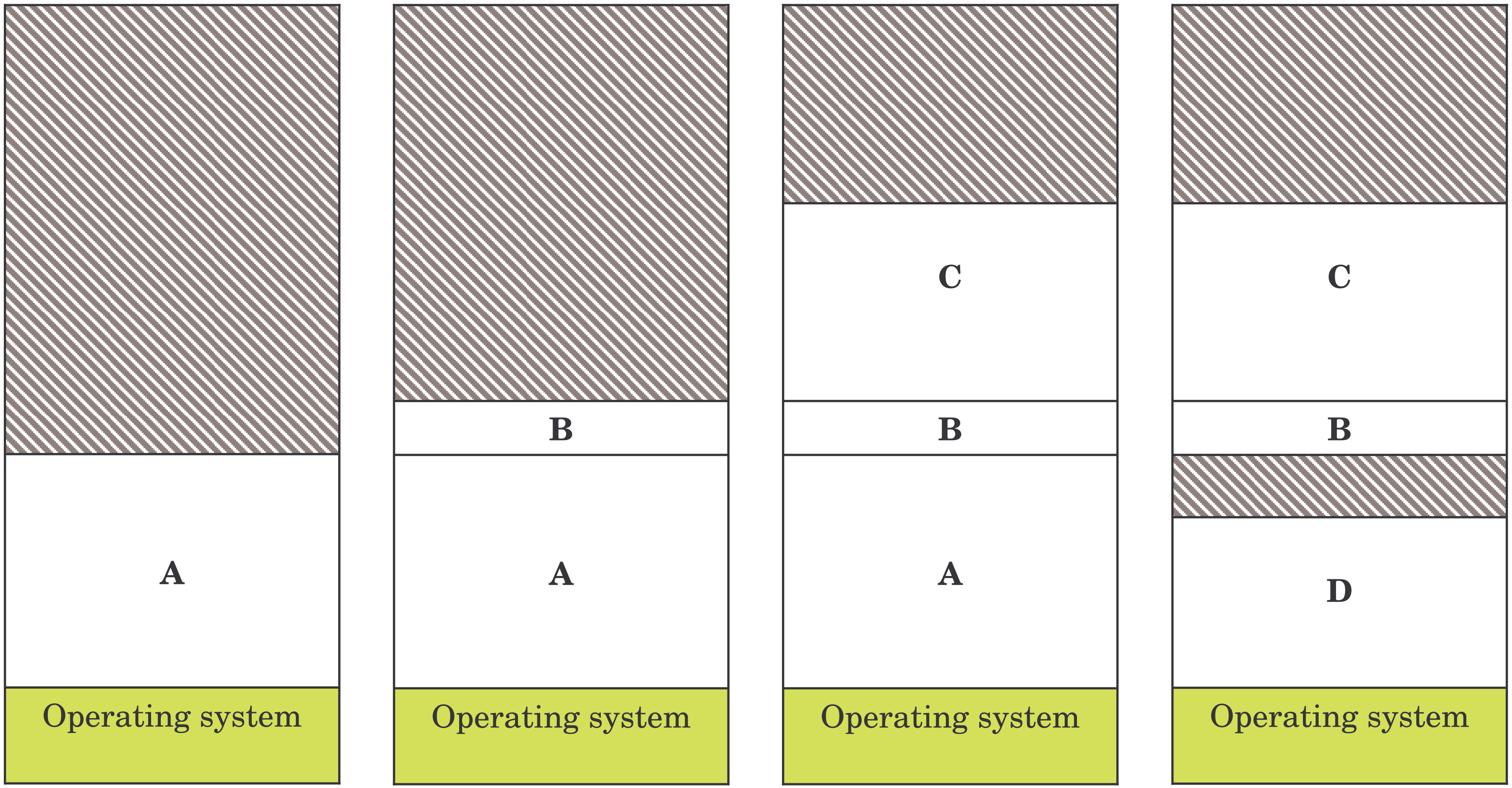
Why?

- Many separate address spaces (segments) (e.g. data, stack, code, and many others if needed)
- Each segment is separate (e.g. addresses from 0 to some MAX)
- Segments might have different lengths
- Segment number + address within segment
- Linking is simplified (libraries within different segments can assume addresses starting from 0) – e.g. if a part of the libraries is recompiled the remainder of the code is unaffected
- Shared library (DLL's) implementation is simpler (the sharing is simpler)

Comparing paging and segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuate be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Pure segmentations

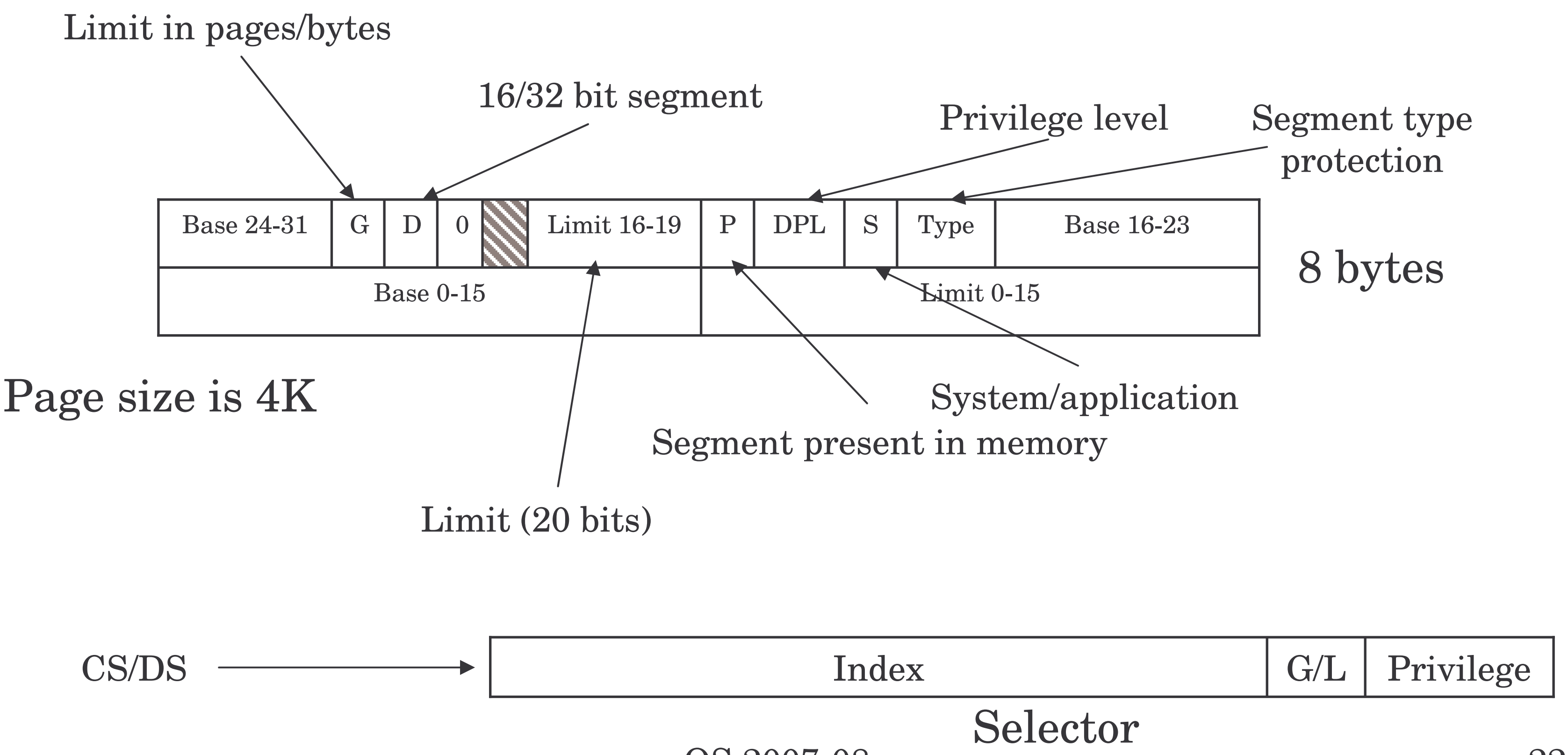


Segmentation + paging (Pentium)

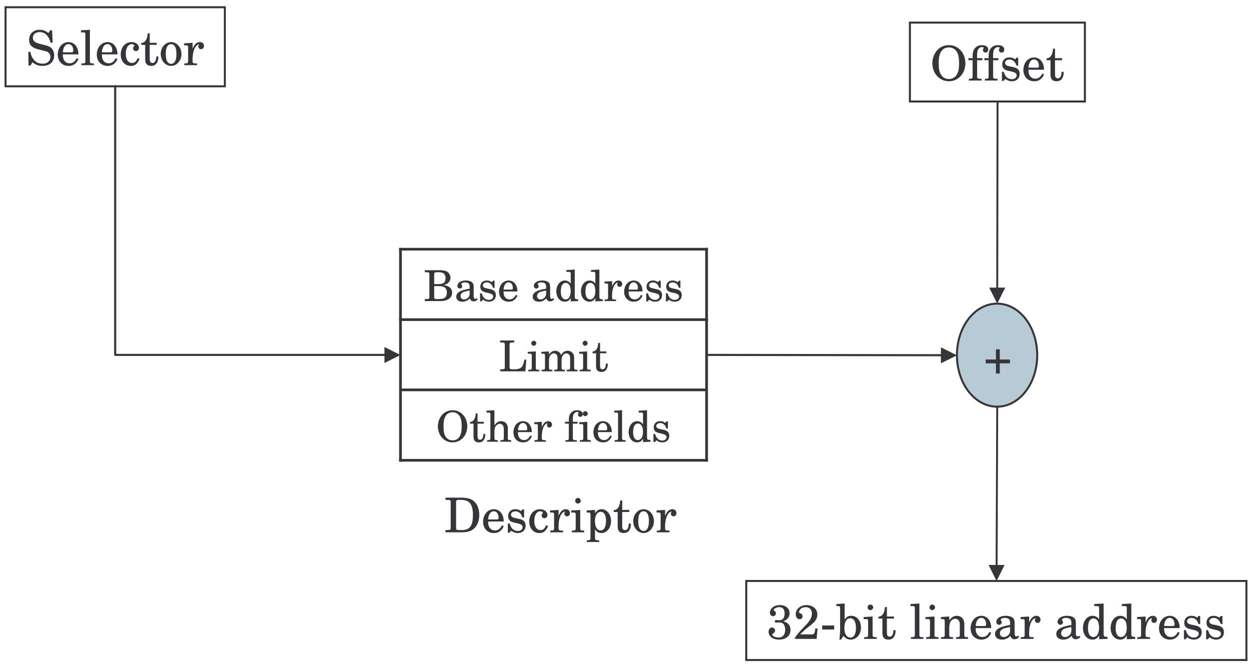
- 16K segments
- 1G 32bit words (DoubleWords)
- Two tables: LDT, GDT – Local (to the process) and global (to the processor) descriptor table
- To work with a segment the machine loads the segment number into a special register (CS, DS, etc.) – CS, DS are 16 bit registers
- The descriptor of the segment (see next slide)

The segment descriptor

- This is used by the microcode within the Pentium to work with segments

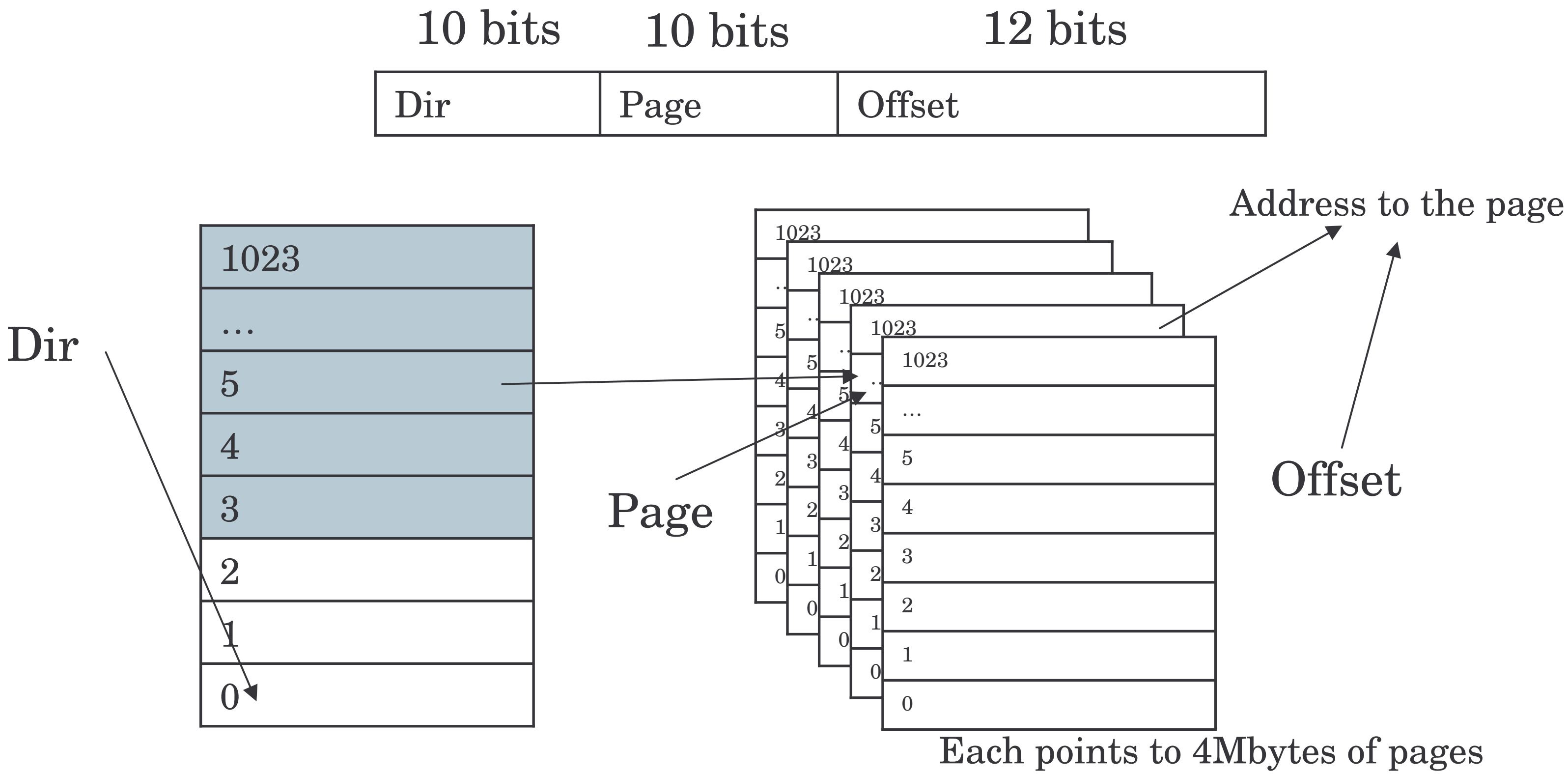


Getting the address



Paging on the Pentium

- 2-level page table in memory



More on the Pentiums

- TLB, to avoid repeated accesses to memory
- The whole thing can be used with just a single segment to obtain a linear 32bit address space
- Set base and limit appropriately
- Protection (a few bits)