Page Size

Small pages:

- less amount of internal fragmentation
- more pages required per process
- more pages per process means larger page tables
- larger page tables means large portion of page tables in virtual memory
- larger number of pages may be found in main memory, so reduced # of page faults
- process makes more page to page transitions, so more chances for a page fault

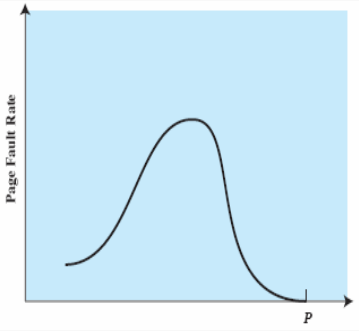
Large pages:

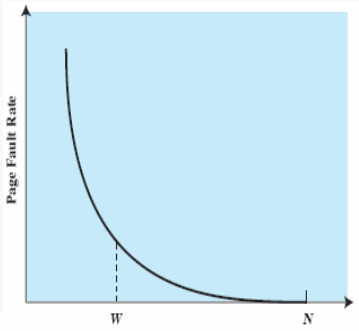
- secondary memory is designed to efficiently transfer large blocks of data so a large page size is better
- pages may be more likely to contain references to far-away locations, leading to increased # of page faults

Expectation: as time goes on during execution, the pages in memory will all contain portions of the process near recent references. Page faults low.

Empirical Paging Results

As page size increases, the page fault rate initially rises since the process has fewer pages resident in main memory, then falls to zero as page size approaches the size of the process image.





As the number of allocated frames approaches the number of pages N in the process image, page faults drop to zero (surprise!).

The decrease is most rapid before the frame allocation is sufficient to hold the *working set* of the process.

Example Page Sizes

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Fetch Policy

Fetch Policy
- determines when a page should be brought into memory

Demand paging only brings pages into main memory when a reference is made to a location on the page
- many page faults when process first started
- fetches only pages that are actually needed by the process
- easy to make decision
- universal choice for deployed paged VM systems

Prepaging brings in more pages than needed
- more efficient to bring in pages that reside contiguously on the disk
- need some criteria for picking pages to pre-fetch, no good ones seem to exist
- also called *anticipatory fetching*
- intuitively appealing, not established as effective

Replacement Policy

Replacement policy decides which resident page will be replaced.

- Belady's Optimal policy: page removed should be the page least likely to be referenced in the near future
- Belady's Optimal policy is clearly infeasible
- most policies predict the future behavior on the basis of past behavior

If selected page has been modified it must be written back to virtual memory before being overwritten.

Frame Locking

- if frame is locked, it may not be replaced
- kernel of the operating system
- control structures
- I/O buffers
- associate a lock bit with each frame

Basic Replacement Algorithms

Optimal policy

Selects for replacement that page for which the time to the next reference is the longest

Impossible to have perfect knowledge of future events

Least Recently Used (LRU)

Replaces the page that has not been referenced for the longest time

By the principle of locality, this should be the page least likely to be referenced in the near future

Each page could be tagged with the time of last reference. This would require a great deal of overhead.

First-in, first-out (FIFO)

Treats page frames allocated to a process as a circular buffer

Pages are removed in round-robin style

Simplest replacement policy to implement

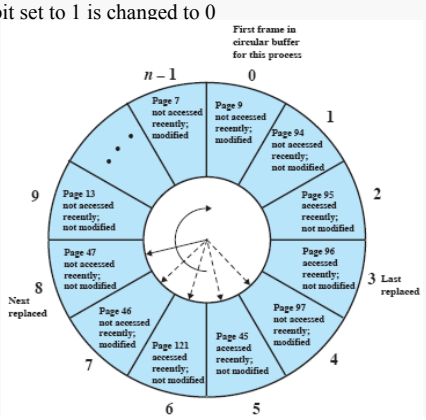
Page that has been in memory the longest is replaced

These pages may be needed again very soon

Clock Replacement Algorithm

Clock Policy

- Additional bit called a *use bit*
- When a page is first loaded in memory, the use bit is set to 1
- When the page is referenced, the use bit is set to 1
- When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
- During the search for replacement, each use bit set to 1 is changed to 0



Examples