CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Controlling Page Faults

When not enough physical memory is allocated to process, thrashing can occur

Thrashing can slow down entire system, not just single low-memory process

How do we know what is "enough" memory?

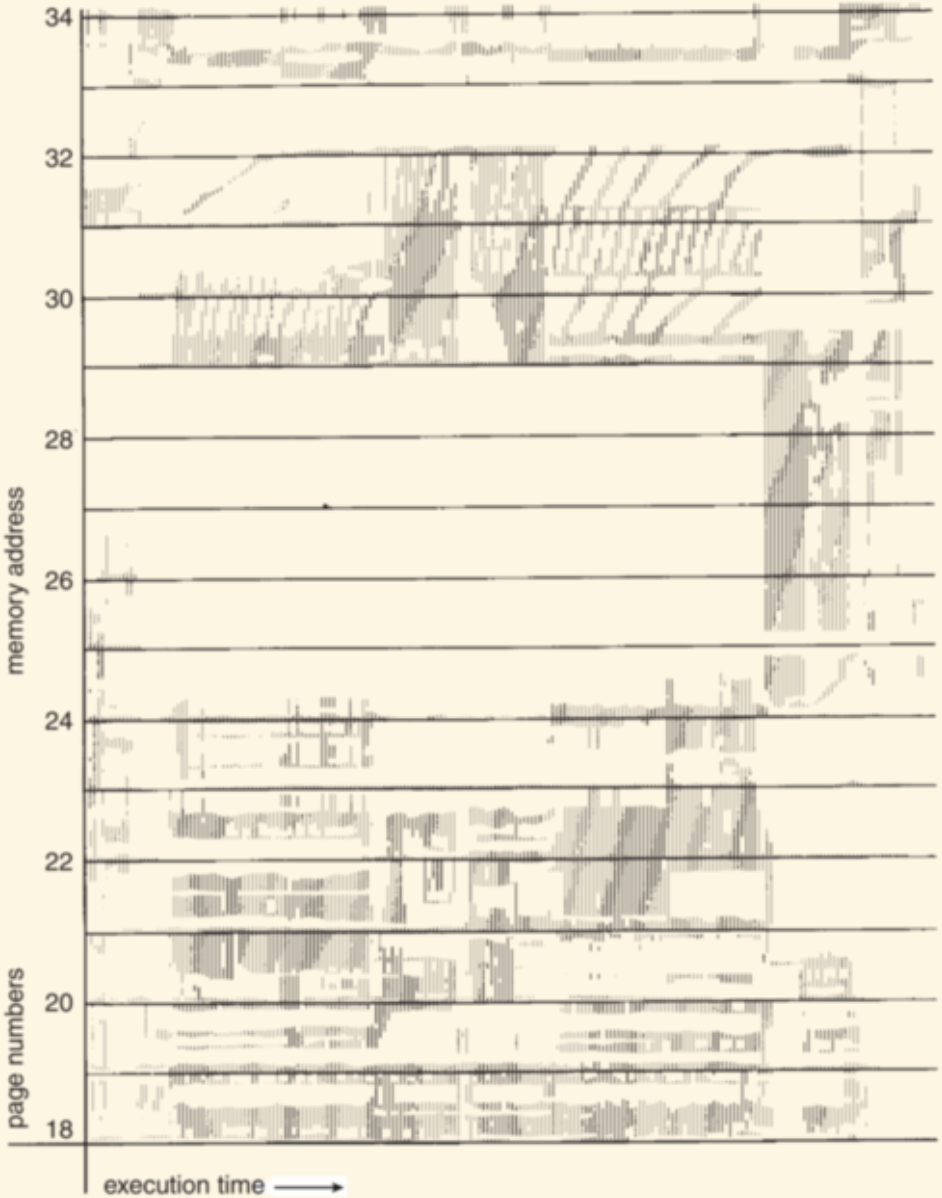
Models can help

Locality model assumes process moves through different phases of execution where it accesses related memory

For example, calling a function might access one part of stack, and when function returns different part is accessed

Another example -- performing set of operations on some array

Each **locality** is set of related memory addresses that are accessed near one another in time


34
32
30
28
26
24
22
20
18
memory address
page numbers
execution time

This is an *observation* about typical behavior of systems

Just as with caching, if programs did not follow this pattern of locality, virtual memory would not be efficient enough to be feasible

Assuming processes follow this model, goal should be to allocate enough frames to process to hold current locality

More simply, want to ensure each process has enough physical memory to hold pages that are actively being used

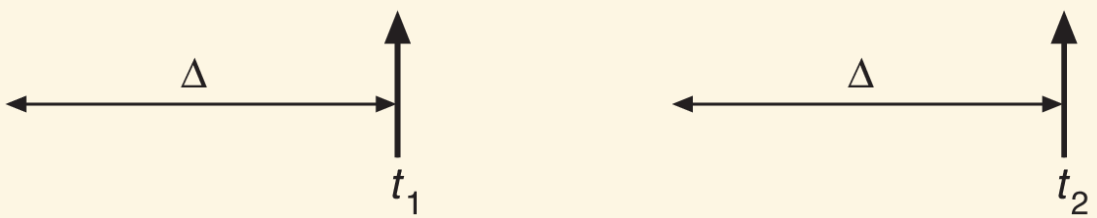
Attempt to capture current locality with working set

For some choice of size D, define **working set window** as most recent D memory accesses

Any pages accessed within **working set window** constitute **working set**

page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .



$WS(t_1) = \{1,2,5,6,7\}$

$WS(t_2) = \{3,4\}$

Recently accessed pages end up in working set

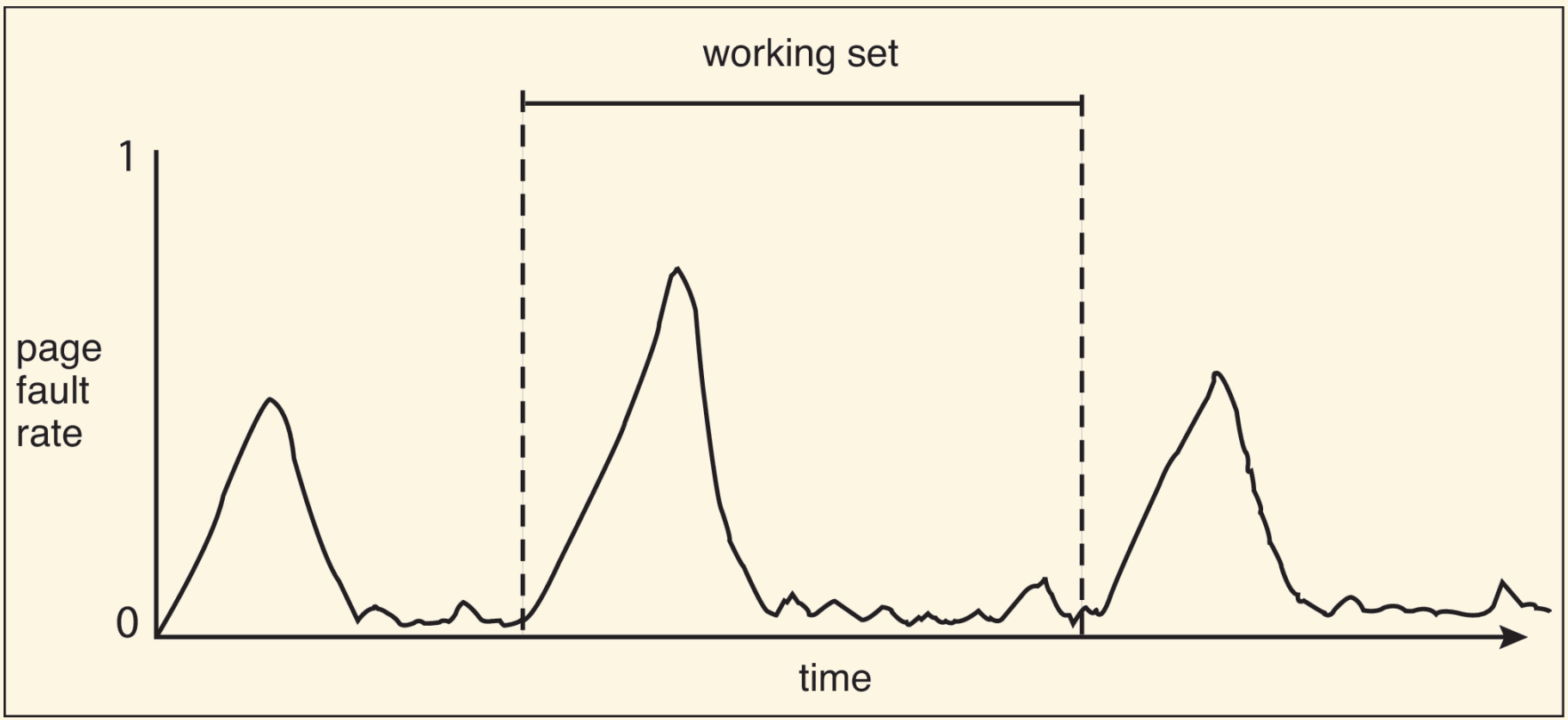
If page is not frequently accessed, it will evetually "fall out" of working set

Idea is that working set will capture pages currently in use and ignore pages that are no longer needed

Working set is approximation of current locality

Book calls this the **working-set model**

Note that working-set model is *based on* assumptions in locality model. Two models are related -- one does not supplant the other


working set
1
page
fault
rate
0
time

Need to choose window size D to capture locality

Too small, and system will evict useful pages

Too large, and memory will be wasted on useless pages

Total demand for frames in system is sum of working set sizes for every process

If demand goes above number of available frames, thrashing will occur

No hard and fast rule for choosing D

Once we understand the idea, using working set to allocate frames is very simple

Track working set of each process and allocate number of frames indicated by size of working set

If total demand on system exceeds total number of frames, swap a process out to disk and distribute its frames

Managing frames this way allows for good balance in system

- Processes have enough memory to avoid thrashing
- Processes are not assigned excess frames beyond what they need, so degree of multiprogramming can stay high

However, glossed over crucial implementation detail

To track working set, OS must keep track of *each* memory access

This is infeasible

Last time we saw a similar problem (LRU), we solved it with timer and reference bits

Similar idea works here

Timer goes off every so often, and OS

- records which pages have reference bit set
- clears all reference bits

Recoreded history of reference bits gives more granular view of when pages were accessed

Relies on working set extending back in time more than back in number of memory accesses

Increasing frequency of timer allows greater accuracy at cost of more overhead

Goal of working-set model was to keep number of page faults to a minimum while allowing for high degree of multiprogramming

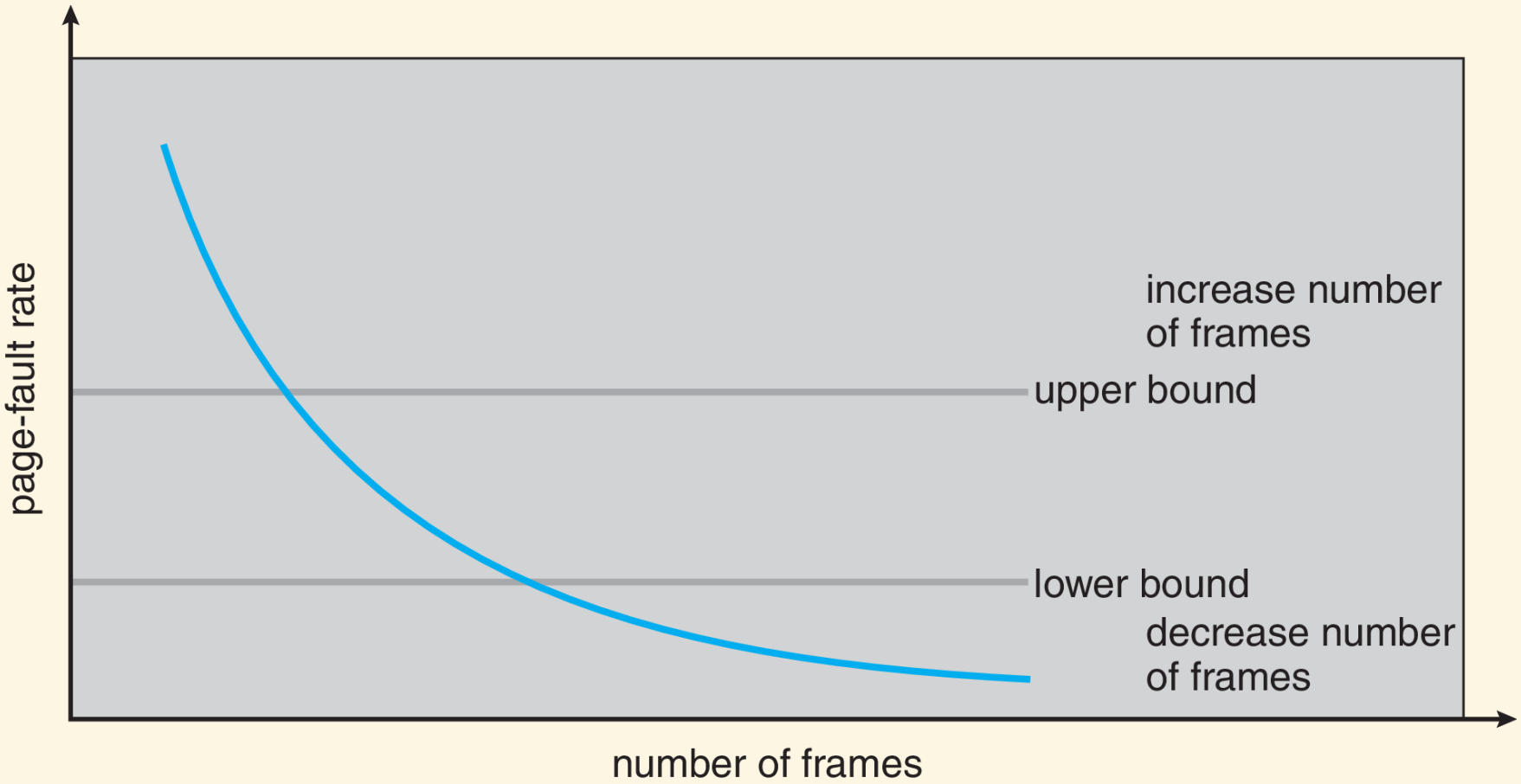
More direct method exists for keeping page fault low

Monitor page fault rate directly

Decide on maximum and minimum allowable rates

If process exceeds maximum, allocate additional frames

If process goes below minimum, deallocate frames


page-fault rate
increase number
of frames
upper bound
lower bound
decrease number
of frames
number of frames

Solves issue in very direct way

Tracking page faults does not have much overhead because OS must be trapped to every page fault regardless