We talked about memory management but several problems remain:

- External Fragmentation
- Processes (or sum of processes) larger than physical memory
- Sharing

Motivation

Sharing Between Processes
- Schemes so far have considered only a single address space per process
  - A single name space per process
  - No sharing

Multiple Name Spaces

How can one share code and data between programs?
Segmentation

- New concept: A segment — a memory “object”
  - A virtual address space
- A process now addresses objects — a pair \((s, \text{addr})\)
  - \(s\) — segment number
  - \(\text{addr}\) — an offset within an object

Two separate registers.
Segment register + offset register

Single register.
Logically divided

Memory Management Basics

Segmentation allows sharing
- ... but leads to poor memory utilization
  - We might not use much of a large segment, but we must keep the whole thing in memory (bad memory utilization).
  - Suffers from external fragmentation
  - Allocation/deallocation of arbitrary size segments is complex
  - Writing problems to adapt to the amount of System Memory is very difficult.
    - Write it to a minimum, but when more is available, the application can’t use it.
    - Write it to the maximum, but then it won’t run on machines with less.
- What if my program must have more memory than the minimum?
  - Use a technique called overlays.

Paging

- Physical memory partitioned into equal sized frames

A memory address is a pair \((f, o)\)
  - \(f\) — frame number \((f_{\text{max}}\) frames)
  - \(o\) — frame offset \((o_{\text{max}}\) bytes/frames)

Physical address = \(o_{\text{max}} \times f + o\)

Physical Address Specification

- Example: A 16-bit physical address space with 512 byte frames
  - \(\log_2 512 = 9\) ⇒ 9 bits for offset
  - 7 bits for frame number
  - \((f,o) = (3,6)\)

\[
\begin{align*}
\text{PA:} & \quad 00000011000000110 \\
& \quad 0x606
\end{align*}
\]
Paging

- A process' virtual address space is partitioned into equal sized \( \text{pages} \).

\[ [\text{page}] = [\text{frame}] \]

A virtual address is a pair \((p, o)\).

- \( p \) — page number \((p_{\text{max}} \text{ pages})\)
- \( o \) — page offset \((o_{\text{max}} \text{ bytes/pages})\)

Virtual address = \( o_{\text{max}} \cdot p + o \)

Virtual Address Space

\( 2^n - 1 = (p_{\text{max}} - 1, o_{\text{max}} - 1) \)

Virtual Address Translation

- A page table maps (virtual) pages to (physical) frames.

Page Table Implementation

- 1 table per process
- Part of process' state

Contents:
- Flags — dirty bit, resident bit, clock/reference bit
- Frame number

Pages are mapped to frames

Pages are contiguous in a VAS...
- But pages are arbitrarily located in physical memory, and
- Not all pages mapped at all times
Virtual Address Translation

- Problem — VM reference requires 2 memory references!
  - One access to get the page table entry
  - One access to get the data

- Page table can be very large; a part of the page table can be on disk.
  - For a machine with 64-bit addresses and 4 k pages, what is the size of a page table?

- What to do?
  - Most computing problems are solved by some form of...
    - Indirection
    - here: Caching

Translation Lookaside Buffer

- Cache recently accessed page-to-frame translations in a TLB
  - For TLB hit, physical page number obtained in 1 cycle
  - For TLB miss, translation is updated in TLB
  - Has typically better than 99% hit ratio!!

Multi-Level Paging

- Add additional levels of indirection to the page table by sub-dividing page number into k parts
  - Create a "tree" of page tables
  - TLB still used, just not shown

Virtual Memory

- We have achieved:
  - Eliminated external fragmentation
  - Enabled sharing

- TBD:
  - Process with more virtual than physical memory

- Problem: Size of the page table
  - 32 bit, assuming 4k pages
    - $2^{32}$ addresses
    - $2^{20} \approx 1$ million pages
Multi-Level Paging

- Example: Two-level paging

![Multi-Level Paging Diagram]

The Problem of Large Address Spaces

- So far, we have looked at Hierarchical Paging (aka. Forward mapped page table) implementations.
- With large address spaces (64-bits) forward mapped page tables become cumbersome.
  - E.g. 7 levels of tables, requires 7 memory references to resolve an address.
  - With several address spaces active simultaneously, they consume large amounts of system memory (must be resident).
- Total (virtual) address space size is growing faster than system (real) memory size.

Mappings are limited by total system memory

- Other schemes make use of the fact that the number of pages mapped to physical frames is proportional to the number of frames, not the sum of all the address spaces:
  - HTabs – Hashed Page Tables
  - Inverted Page Tables

HTabs – Hashed Page Tables

- Hashing function (used by both HW and OS) maps a Virtual Address (page number, pid,) to an index.
- A fixed-location table is indexed by index, which points to the first of a linked list of mapping VA to frame mappings that looks like:
  `<pid, page#, fram#, next entry>`
Inverted Page Tables

- Hash page numbers to find corresponding frame number
- Page frame number is not explicitly stored (1 frame per entry)
- Protection, dirty, used, resident bits also in entry

Virtual Memory and Paging

- A process’ VAS is its context
  - Contains its code, data, and stack
- Code pages are stored in a user’s file on disk
  - Some are currently residing in memory; most are not
- Data and stack pages are also stored in a file
  - Although this file is typically not visible to users
  - File only exists while a program is executing

OS determines which portions of a process’ VAS are mapped in memory at any one time

Page Fault

- References to non-mapped pages generate a page fault

Page fault handling steps:
- Processor runs the exception handler
- OS blocks the running thread
- OS starts read of the unmapped page
- OS resumes/initiates some other thread
- Read of page completes
- OS maps the missing page into memory
- OS restart the faulting thread

What about the OS Address Space structures?

- Using Hierarchical Paging, the OS and MMU structures can be the same.
  - When a thread in an AS is “executing”, its Hierarchical Paging structures must (all) be resident in memory.
  - When not executing, it could be swapped out.

- For HTab and Inverted Page Tables, the OS still needs Hierarchical Paging – like structures for managing AS mappings.
  - When a thread in an AS is “executing”, its Hierarchical Paging structures do not need to be resident.
  - Of course, this means the OS’ page-fault code could take a page fault! Recursion can be managed, but is hard.
Virtual Memory Performance

- To understand the overhead of paging, compute the effective memory access time (EAT)
  \[ EAT = \text{memory access time} \times \text{probability of a page hit} + \text{page fault service time} \times \text{probability of a page fault} \]

Example:
- Memory access time: 20 ns
- Disk access time: 25 ms
- Let \( p \) = the probability of a page fault
- \( EAT = 20(1-p) + 25,000,000 \cdot p \)

- To realize an \( EAT \) within 5% of minimum, what is the largest value of \( p \) we can tolerate?

Summary

- Physical and virtual memory partitioned into equal-size units
- Size of VAS unrelated to size of physical memory
- Virtual pages are mapped to physical frames
- Simple placement strategy
- There is no external fragmentation
- Key to good performance is minimizing page faults

Paging

- Which pages are in memory?
  - Pre-Paging: OS loads some pages in advance
  - Demand-Paging: Start blank, load when needed

- What happens on a page fault?
  - We load the page from disk into memory

- What happens if there is no more physical memory available?
  - We have to evict a loaded page to disk (page out).

- Which page should we pick?
  - Page replacement policy
Page Replacement Algorithms
- Typically Sum VAS >> Physical Memory
- With demand paging, physical memory fills quickly
- When a process faults & memory is full, some page must be swapped out
  - Handling a page fault now requires 2 disk accesses not 1!
- Which page should be replaced?
  - Local replacement — Replace a page of the faulting process
  - Global replacement — Possibly replace the page of another process

Evaluating Page Replacement Algorithms
- Record a trace of the pages accessed by a process
  - Example: (Virtual) address trace...
    - (3,0), (1,9), (4,1), (2,1), (5,3), (2,0), (1,9), (2,4), (3,1), (4,8)
    - generates page trace
      - 3, 1, 4, 2, 5, 2, 1, 2, 3, 4 (represented as c, a, d, b, e, b, a, b, c, d)
  - Simulate the behavior of a page replacement algorithm on the trace and record the number of page faults generated
    - fewer faults → better performance

Optimal Page Replacement
- Clairvoyant Replacement: Replace page that won’t be needed for the longest time in the future

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page Frames</td>
<td>0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Faults

Time page needed next

<table>
<thead>
<tr>
<th>Time page needed next</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 7</td>
</tr>
<tr>
<td>b = 6</td>
</tr>
<tr>
<td>c = 9</td>
</tr>
<tr>
<td>d = 10</td>
</tr>
</tbody>
</table>
Optimal Page Replacement

- Clairvoyant Replacement: Replace page that won't be needed for the longest time in the future

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page Frames</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>Faults</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Time page needed next</td>
<td>a = 7</td>
<td>a = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = 6</td>
<td>b = 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c = 9</td>
<td>c = 13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d = 10</td>
<td>e = 14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIFO Replacement

- Simple to implement
- A single pointer suffices

Performance with 4 page frames:

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page Frames</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>Faults</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

Locality

- The principle of locality
  - A program’s page-access patterns can be broken into time-phases, where each phase consists of the time during which the same number of pages are being accessed.
  - For almost all applications, these phases are long enough for the OS to identify the set of pages being accessed during the phase.
Locality

- Does the locality principle let us divine what page will next be faulted on?
  - No.

- Can we divine which of the pages currently in memory is least likely to be re-referenced again?
  - Yes.
  - It’s the one last referenced the longest time ago.

- Operating systems use some form of LRU for page replacement (frame selection) implementation.

Least Recently Used Replacement

- Replace the page that hasn’t been referenced for the longest time

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Faults

- Time page last used
  - a = 2
  - b = 4
  - c = 3

Least Recently Used Replacement

- Replace the page that hasn’t been referenced for the longest time

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Faults

- Time page last used
  - a = 2
  - b = 4
  - c = 3
  - d = 3
  - e = 5
Clock Algorithm

- Maintain a circular list of pages resident in memory
- Use a clock (or used/referenced) bit to track how often a page is accessed
- The bit is set whenever a page is referenced
- Clock hand sweeps over pages looking for one with used bit = 0
- Replace pages that haven't been referenced for one complete revolution of the clock

Clock Replacement

```plaintext
func Clock_Replacement
begin
while (victim page not found) do
  if(used bit for curr.page == 0) then
    replace current page
  else
    reset used bit
    advance clock pointer
  end
end
end
```

Second Chance Clock Algorithm

- There is a significant cost to replacing "dirty" pages
- Modify the Clock algorithm to allow dirty pages to always survive one sweep of the clock hand
- Use both the dirty bit and the used bit to drive replacement
### Second Chance Replacement

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>a⁺⁺</td>
<td>d</td>
<td>b⁺⁺</td>
<td>e</td>
<td>b</td>
<td>a⁺⁺</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Frames</td>
<td>0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>Page table entries for resident pages:</td>
<td>10</td>
<td>a</td>
<td>10</td>
<td>b</td>
<td>10</td>
<td>c</td>
<td>10</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Local Page Replacement

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Frames</td>
<td>0</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>Page table entries for resident pages:</td>
<td>10</td>
<td>a</td>
<td>10</td>
<td>b</td>
<td>10</td>
<td>c</td>
<td>10</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

CS 439 (Helmeryer)  9/28/2015
Page Replacement Strategies

- **Victim Frame Selection**
  - Local: only considering current process
  - Global: looking at all processes

- **Local page replacement**
  - Steal a frame from the faulting process
  - At any time, the number of frames per-process is fixed (but adjusted periodically).

- **Global page replacement**
  - Steal the system-wide LRU, no matter what process it belongs to
  - The number of frames per-process is dynamic

The Working Set Model of Page Replacement

- Assume recently referenced pages are likely to be referenced again soon
- only keep those pages recently referenced in memory (called the working set)
  - The number of frames allocated to a process will vary over time, as the process moves between phases.
  - Thus pages may be removed even when no page fault occurs

- A process is allowed to execute only if its working set fits into memory
  - Why?
  - The working set model performs implicit load control

Working Set Replacement

- **Keep track of the last \( \tau \) references**
  - The pages referenced during the last \( \tau \) memory accesses are the working set
  - \( \tau \) is called the window size

- **Example: Working set computation, \( \tau = 4 \) references:**

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>Pages in Memory</td>
<td>Page a</td>
<td>( F )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page b</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page c</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page d</td>
<td>( F )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page e</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Working Set Replacement

- **Keep track of the last \( \tau \) references**
  - The pages referenced during the last \( \tau \) memory accesses are the working set
  - \( \tau \) is called the window size

- **Example: Working set computation, \( \tau = 4 \) references:**

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>Pages in Memory</td>
<td>Page a</td>
<td>( F )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page b</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page c</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page d</td>
<td>( F )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Page e</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working Set</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Page-Fault-Frequency Replacement

- Explicitly attempt to minimize page faults
  - When page fault frequency is high — increase working set size
  - When page fault frequency is low — decrease working set size

**Algorithm:**
- Keep track of the rate at which faults occur
  - When a fault occurs, compute the time since the last page fault
  - Record the time, \( t_{last} \), of the last page fault
- If the time between page faults is "large" then reduce the working set size (WSS)
  - If \( t_{current} - t_{last} > \tau \), then remove from memory all pages not referenced in \([t_{last}, t_{current}]\)
- If the time between page faults is "small" then increase working set
  - If \( t_{current} - t_{last} \leq \tau \), then add faulting page to the working set

---

**Example with window size = 2**

- If \( t_{current} - t_{last} > 2 \), remove pages not referenced in \([t_{last}, t_{current}]\) from the working set
- If \( t_{current} - t_{last} \leq 2 \), just add faulting page to the working set

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
</tr>
<tr>
<td>Pages in Memory</td>
<td>( \text{Page a} )</td>
<td>( \text{Page b} )</td>
<td>( \text{Page c} )</td>
<td>( \text{Page d} )</td>
<td>( \text{Page e} )</td>
<td>( \text{Page f} )</td>
<td>( \text{Page g} )</td>
<td>( \text{Page h} )</td>
<td>( \text{Page i} )</td>
<td>( \text{Page j} )</td>
<td></td>
</tr>
<tr>
<td>Faults</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td>( \bullet )</td>
<td></td>
</tr>
<tr>
<td>( t_{cur} - t_{last} )</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Thrashing

- How to prevent it?

Load Control

- When the multiprogramming level should be decreased, which process should be swapped out?
  - Lowest priority process?
  - Smallest process?
  - Largest process?
  - Oldest process?
  - Faulting process?