Virtual Memory Management: Recap

- Key concept: Demand paging
  - Load pages into memory only when a page fault occurs
- Issues:
  - Placement strategies
    - Place pages anywhere – no placement policy required
  - Replacement strategies
    - What to do when there exist more jobs than can fit in memory
  - Load control strategies
    - Determining how many jobs can be in memory at one time

User Program 1
User Program 2
User Program 3
User Program 4

Page Replacement Algorithms

- Typically $\sum \text{VAS} \gg \text{Physical Memory}$
- 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

Optimal Strategy: Clairvoyant Replacement

- Replace the page that won’t be needed for the longest time in the future

Initial allocation

<table>
<thead>
<tr>
<th>Time</th>
<th>Request</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Page needed next</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

Faults

- Simulation of behavior of a page replacement algorithm on the trace and record the number of page faults generated
  - Fewer faults → better performance
### Least Recently Used (LRU) Replacement

- Use the recent past as a predictor of the near future
- 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>
<tr>
<td>Pages</td>
<td>b</td>
<td>c</td>
<td>a</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>Time page</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>last used</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>

**Frame List**

```
Frame List
0 1 2 3 4 5 6 7 8 9 10
```

### Performance with 4 page frames:

- Simple to implement
  - A single pointer suffices

<table>
<thead>
<tr>
<th>Faults</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>d</td>
<td>c</td>
<td>b</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>
</tbody>
</table>

**Frame List**

```
Frame List
0 1 2 3 4 5 6 7 8 9 10
```

### How to Implement LRU?

- Maintain a "stack" of recently used pages

<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>Pages</td>
<td>b</td>
<td>c</td>
<td>a</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>LRU page stack</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>Page to replace</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>

**Frame List**

```
Frame List
0 1 2 3 4 5 6 7 8 9 10
```

### Local Replacement: FIFO

- Simple to implement
  - A single pointer suffices

**Frame List**

```
Frame List
0 1 2 3 4 5 6 7 8 9 10
```

### How to Implement FIFO

```
FIFO
Page
Replacement
```

### Least Recently Used (LRU) Replacement

```
LRU
Replacement
```

### Local Replacement: FIFO

```
FIFO
Page
Replacement
```

### How to Implement FIFO

```
FIFO
Page
Replacement
```

### Least Recently Used (LRU) Replacement

```
LRU
Replacement
```

### Local Replacement: FIFO

```
FIFO
Page
Replacement
```
What is the goal of a page replacement algorithm?
- A. Make life easier for OS implementer
- B. Reduce the number of page faults
- C. Reduce the penalty for page faults when they occur
- D. Minimize CPU time of algorithm

Optimization: Second Chance Algorithm
- There is a significant cost to replacing “dirty” pages
  - Why?
    - Must write back contents to disk before freeing!
  - 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

Approximate LRU: The 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

Second Chance Example
- Before clock sweep
  - Used bit
- After clock sweep
  - Used bit

Rediscount bit

Page table entries
for resident pages:

Faults:

Page

Frames

Time

0
1
2
3
4
5
6
7
8
9
10
Requests

0
1
2
3

Faults:

Page table entries
for resident pages:

Faults:

Page table entries
for resident pages:

Faults:
### Local Replacement and Memory Sensitivity

**Second Chance Example**

<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>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>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>a</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>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</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 Faults</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

**Requests**

- a
- b
- c
- d
- e

**Faults**

None

### Temporal vs. Physical Locality

- 90% of the execution of a program is sequential.
- Most iterative constructs consist of a relatively small number of instructions.
- When processing large data structures, the dominant cost is sequential processing on individual structure elements.
- Temporal vs. physical locality

### Optimal Replacement with a Variable Number of Frames

**Example:** \(r = 4\)

<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>c</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>a</td>
</tr>
<tr>
<td>Frames in Memory</td>
<td>Page a</td>
<td>*</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>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>2</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>3</td>
<td>Page d</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</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>
</tbody>
</table>

**Example:**

- VMIN — Replace a page that is not referenced in the next \(r\) accesses.
- Example: \(r = 4\)

### Second Chance Example

**Local Replacement and Memory Sensitivity**

<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>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>
</tr>
<tr>
<td>Frames</td>
<td>0</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>
</tr>
<tr>
<td>1</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>
</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>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>

**Requests**

- a
- b
- c
- d

**Faults**

None

### Optimal Replacement with a Variable Number of Frames

**Example:**

- VMIN — Replace a page that is not referenced in the next \(r\) accesses.
- Example: \(r = 4\)
The Working Set Model

- Assume recently referenced pages are likely to be referenced again soon.
- ... and only keep those pages recently referenced in memory (called the working set).
- Thus pages may be removed even when no page fault occurs.
- The number of frames allocated to a process will vary over time.
- A process is allowed to execute only if its working set fits in memory.
- The working set model performs implicit load control.

Working Set Page Replacement

- Keep track of the last \( r \) references (excluding faulting reference).
  - The pages referenced during the last \( r \) memory accesses are in the working set.
  - \( r \) is called the window size.
- Example: Working set computation, \( r = 4 \) references:

```
Time   0  1  2  3  4  5  6  7  8  9  10
Requests c  c  d  b  c  e  c  e  a  d
Pages in Memory
  Page a  •  •  •  •  •  •  •  •  •  •
  Page b  •  •  •  •  •  •  •  •  •  •
  Page c  •  •  •  •  •  •  •  •  •  •
  Page d  •  •  •  •  •  •  •  •  •  •
  Page e  •  •  •  •  •  •  •  •  •  •
```

Page Fault Frequency Replacement

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

```
Time   0  1  2  3  4  5  6  7  8  9  10
Requests c  c  d  b  c  e  c  e  a  d
Pages in Memory
  Page a  •  •  •  •  •  •  •  •  •  •
  Page b  •  •  •  •  •  •  •  •  •  •
  Page c  •  •  •  •  •  •  •  •  •  •
  Page d  •  •  •  •  •  •  •  •  •  •
  Page e  •  •  •  •  •  •  •  •  •  •
```
Load Control: Fundamental Trade-off

- High multiprogramming level
  \[ MPL_{\text{low}} = \frac{\text{number of page frames}}{\text{minimum number of frames required for a process to execute}} \]
- Low paging overhead
  \[ MPL_{\text{low}} = 1 \text{ process} \]

- Issues
  - What criterion should be used to determine when to increase or decrease the MPL?
  - Which task should be swapped out if the MPL must be reduced?

Load Control Done Wrong

<table>
<thead>
<tr>
<th>CPU Device</th>
<th>I/O Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Paging Device</td>
</tr>
</tbody>
</table>

- Assume memory is nearly full
- A chain of page faults occur
  - A queue of processes forms at the paging device
- CPU utilization falls
  - Operating system increases MPL
  - New processes fault, taking memory away from existing processes
  - CPU utilization goes to 0, the OS increases the MPL further...

System is thrashing — spending all of its time paging

Load Control and Thrashing

- Thrashing can be ameliorated by local page replacement
- Better criteria for load control: Adjust MPL so that:
  - mean time between page faults (MTBF) = page fault service time (PFST)
  - \( \sum W_S \) = size of memory

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