Virtual Memory Overview

- **Real Memory**: Main memory
- **Virtual Memory**: The memory perceived by the user or programmer
  - Implemented through paging and segmentation with page swapping
- **Properties of Paging and Segmentation**
  - **Dynamic Address Translation**: Memory references are logical addresses that are dynamically translated into physical addresses at run time.
  - **Non-contiguous Main Memory**: A process may be broken up into pieces that need not be contiguously located in main memory.
- **Potential Benefits**
  - **Effective Multiprogramming**: More processes in main memory
  - **Less Memory Constrained**: Loosens memory constraints
  - **Protection**: Memory references only to own physical memory

Demand Paging

- Bring a page into main memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed only when it is referenced
  - Abort invalid references
  - Swap in pages when referenced but not in main memory
- **Effective only when there is a large degree of spatial locality over short time periods.**
- **Thrashing**: A condition in which a processor spends most of its time swapping pages rather than executing instructions.

Matrix Multiply (1)

```
int A[N][N], B[N][N], C[N][N];
for (int i=0; i<N; i++)
  for (int j=0; j<N; j++)
    for (int k=0; k<N; k++)
      C[i][j] += A[i][k] * B[k][j];
```

- **Highest locality**: Instructions and A matrix
- **What if 1 page will hold 1 row?**

![Matrix Multiply Diagram](image)

- Instruction Reference String: I, I, ..., I
- A[][] Reference String: A0, A0, ..., A0, A1, A1, ...
- B[][] Reference String: B0, B1, B2, ..., B0, B1, B2, ...
- C[][] Reference String: C0, C0, ..., C0, C1, C1
Matrix Multiply (2)

- Composite memory reference string (n = N-1):
  \[ I, I, A0, B0, I, I, I, I, A0, B1, I, I, \ldots I, I, A0, Bn, I, I ; CO \rightarrow C[0][n] \]
  \[ I, I, A0, B0, I, I, I, I, A0, B1, I, I, \ldots I, I, A0, Bn, I, I ; CO \rightarrow C[0][1] \]
  \[ I, I, A0, B0, I, I, I, I, A0, B1, I, I, \ldots I, I, A0, Bn, I, I ; CO \rightarrow C[0][0] \]
  \[ \ldots \]
  \[ I, A1, B0, I, I, I, I, A1, B1, I, I, \ldots I, I, A1, Bn, I, I ; CI \rightarrow C[1][0] \]
  \[ \ldots \]
  \[ I, An, B0, I, I, I, I, An, B1, I, I, \ldots I, I, An, Bn, I, I ; Cn \rightarrow C[n][n] \]

- Physical memory allocation of 4 frames
  » Initial Placement:
    \[ I \rightarrow \text{Page 0, A}[0][*] \rightarrow 1, B[0][*] \rightarrow 2, C[0][*] \rightarrow 3 \]
  » Page reference string (F = Page Fault):
    \[ 0, 0, 1, 2, 0, 0, 0, 0, 1, F, 0, 0, 0, 1, F, 0, 0, \ldots \]

- Instructions and A[][] show high temporal locality

Typical Virtual Memory Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Size</td>
<td>512 to 8KB</td>
</tr>
<tr>
<td>Hit Time</td>
<td>5 to 100 nsec</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>5 to 30 msec</td>
</tr>
<tr>
<td>Main Memory Size</td>
<td>64 MB to 2 GB</td>
</tr>
<tr>
<td>Desired Hit Rate</td>
<td>&gt;= 99.9%</td>
</tr>
</tbody>
</table>

Paged Virtual Memory Issues

- How is a page frame found if it is in main memory?
  » Page Table: Physical address = (Frame #, Offset)

- How large should a page frame be?
  » Big page: 1) Small page table; 2) more efficient read/write; 3) greater internal fragmentation; 4) higher process load time

- Which page frame should be replaced on a virtual memory miss?
  » The one that is the least likely to be referenced in the future

- Where can a page be placed in main memory?
  » Almost anywhere in main memory \( \rightarrow \) Need associative hardware for address translation

- When should a page frame be written back to the swap device (disk)?
  » Only if it has been modified (it is dirty) and as late as possible

Basic Page Table Structure

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>page#</td>
<td>frame#</td>
</tr>
<tr>
<td>offset</td>
<td>offset</td>
</tr>
</tbody>
</table>

Valid Bit (In memory?)
Modified Bit (Dirty?)

Program Paging Hardware Memory
Huge Virtual Address Spaces

- If the page size is 1 KB in a virtual memory that can be as large as \(2^{32}\) bytes (4 gigabytes), how many entries will the page table have?
  \(2^{32}/2^{10} = 2^{22}\), about 4 million entries
- How many pages will be occupied by the page table if each row is 32 bits (4 bytes)?
  \(2^{22}2^{2}/2^{10} = 2^{14} = 16\) K pages!
  - Page tables can be huge and are subject to paging

Two-Level Page Table

- Example: Pentium

Inverted Page Table

- Example: Power PC
  - One row for each page frame in physical (real) memory

Inverted Page Table With Hashing
**Translation Lookaside Buffer (TLB)**

- Mechanism to reduce number of memory accesses from 2 to 1 for each address reference

![TLB Diagram]

**Page Fault Handling (1)**

1. Use TLB
   - If not in TLB, use page table (part of PCB) to update TLB (page must be in memory (valid bit ON))
2. If invalid, trap to OS and terminate process; if valid but not in memory, page in from swap area
3. Find location on disk
4. Find a free page frame for incoming page
5. Read page into free frame
   - If memory is full, perform *page replacement*
6. Update page table
7. Restart interrupted instruction

**Page Fault Handling (2)**

**Typical TLB Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>4 to 8 bytes (1 page entry)</td>
</tr>
<tr>
<td>Hit Time</td>
<td>2.5 to 5 nsec (1 clock cycle)</td>
</tr>
<tr>
<td>Miss Penalty</td>
<td>50 to 150 nsec</td>
</tr>
<tr>
<td>TLB Size</td>
<td>32 bytes to 8 KB</td>
</tr>
<tr>
<td>Desired Hit Rate</td>
<td>98% to 99.9%</td>
</tr>
</tbody>
</table>
Page Fault Rate

- How many memory operations occur in the inner loop of the matrix multiply example?

```java
R1 ← 0;
for (int k=0; k<N; k++) {
  R2 ← A[i][k];   // Load R2 with A[i][k]
  R3 ← B[k][j];   // Load R3 with B[k][j]
  R4 ← R2 * R3;   // A[i][k] + B[k][j]
  R1 ← R1 + R4;   // C[i][j] += ...
}
R1 → C[i][j];   // Store value of C[i][j]
```

- Inner Loop: About 2N data memory loads, 1 store
- **Page Fault Rate:** The number of page faults per memory reference

Matrix Multiply Page Fault Rate

- **Composite memory reference string (n = N-1):**
  ```
  I, I, A0, B0, I, I, I, I;  I, I, A0, B1, I, I;  ... I, I, A0, Bn, I, I;  C0 // C[0][0]
  I, I, A0, B0, I, I, I, I;  I, I, A0, B1, I, I;  ... I, I, A0, Bn, I, I;  C0 // C[0][1]
  ...
  I, I, An, B0, I, I, An, B1, I, I;  ... I, I, An, Bn, I, I;  Cn // C[n][n]
  ```

- **Page reference string (F = Page Fault)**
  - Same assumptions as earlier (4 frames allocated, etc)
  - 0, 0, 1, 2, 0, 0; 0, 0, 1, F; 0, 0, 1, F; 0, 0; ... 
- **Page Fault Rate = 1/6 = 0.667**