3. manages (moves, removes, reallotes) pages in memory.

2. then selectively allocates pages to frames in memory, and

1. divides and assigns processes to fixed sized pages.

Today: Paging

Processes typically do not use their entire space in memory all the time.

- Dynamic Relocation
- Static Relocation
- Uniprogramming

Last Class: Memory Management
Example: Paging

- Paging does not eliminate internal fragmentation (1/2 page per process)
- By dividing memory into fixed size pages, we can eliminate external
  allocated contiguously in memory.
- The logical memory of the process is contiguos, but pages need not be
  pages greatly simplify the hole fitting problem
- Keep only those parts of a process in memory that are actually being used

90/10 rule: Processes spend 90% of their time accessing 10% of their space

Motivation & Features

Department of Computer Science, UMass Amherst
A page table keeps track of the page frame in memory in which the page is located.

- In paging, the virtual address identifies the page and the page offset translates virtual address to actual physical address in memory.
- The OS lays the process down on pages and the paging hardware translates virtual addresses to actual addresses.
- Process segments contain page, virtual address from 0 to size of the process.
- Processes use a virtual (logical) address to name memory locations.

**Virtual Address:**

**Problem:** How do we find addresses when pages are not allocated?
### Paging Hardware: Practical Details

<table>
<thead>
<tr>
<th>d: Page offset</th>
<th>p: Page number</th>
</tr>
</thead>
</table>

1. The low order \( n \) bits select the offset in the page.
2. The high order \( m \) — \( n \) bits of a virtual address select the page.
3. Virtual address space of size \( 2^m \) pages and a page of size \( 2^n \), then addresses easier. For example, given powers of 2 make the translation of virtual addresses into physical addresses easier. For example, given:

- \( 8192 \) bytes per page.
- \( 512 \) bytes and page size (frame size) are typically a power of 2 between \( 512 \) bytes and

---

### Protection

Protection is provided with the same mechanisms as used in dynamic relocation.

The hardware does the translation.

Mapping is invisible to the process; the OS maintains the mapping and

Think of the page table as a set of relocation registers, one for each frame.

Paging is a form of dynamic relocation, where each virtual address is bound by the paging hardware to a physical address.
Given virtual address 24, do the virtual to physical translation.

- What part is p, and d?

- How many bits for an address. Assume we can address 1 byte increments?

- How big is the page table?

### Address Translation Example

<table>
<thead>
<tr>
<th>Memory size</th>
<th>Page size</th>
<th>Frames in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 bytes</td>
<td>32 bytes</td>
<td>7</td>
</tr>
<tr>
<td>224 bytes</td>
<td>30 bytes</td>
<td>7</td>
</tr>
<tr>
<td>192 bytes</td>
<td>28 bytes</td>
<td>7</td>
</tr>
<tr>
<td>160 bytes</td>
<td>26 bytes</td>
<td>7</td>
</tr>
<tr>
<td>128 bytes</td>
<td>24 bytes</td>
<td>7</td>
</tr>
<tr>
<td>96 bytes</td>
<td>22 bytes</td>
<td>7</td>
</tr>
<tr>
<td>64 bytes</td>
<td>20 bytes</td>
<td>7</td>
</tr>
<tr>
<td>32 bytes</td>
<td>18 bytes</td>
<td>7</td>
</tr>
</tbody>
</table>

Page size = 16 bytes
Memory size = 256 bytes
Typical TLB sizes range from 8 to 2048 entries.
- If memory accesses have locality, address translation has locality too.

The frame (value) in which they are stored.

TLB: a fast fully associative memory that stores page numbers (key) and

Memory: Advantages? Disadvantages?

Registers: Advantages? Disadvantages?

How should we store the page table?

Making Paging Efficient

What needs to happen on a context switch?

Given virtual address 13, do the virtual to physical translation.

What part is p, and d?

Byte increment?

How many bits for an address? Assume we can address only 4 bits (4)

Address Translation Example
The Translation Look-aside Buffer (TLB)

- v: valid bit that says the entry is up-to-date
- TLB Hit: hit in TLB
- TLB Miss: miss in TLB
- Page Table Hit: hit in page table
- Page Table Miss: miss in page table
- Load: load operation

Costs of Using The TLB

1. What is the effective memory access cost if the page table is in memory?
2. What is the effective memory access cost with a TLB?

A large TLB improves hit ratio, decreases average memory cost.
too big, and many systems use a multilevel paging scheme (refer to Section 7.5.3). 

**Multilevel Paging:**

1. Copy the page table base register value to the PCB.
2. Copy the TLB to the PCB (optionally).
3. Flush the TLB.
4. Restore the page table base register.
5. Restore the TLB if it was saved.

**On a context switch:**

- Possibly a copy of the TLB.
- The page table.

The Process Control Block (PCB) must be extended to contain:

---

**Saving/Restoring Memory on a Context Switch**

---

Initializing Memory when Starting a Process

---
Paging requires more complex OS to maintain the page table.
- Paging requires hardware support in the form of a TLB to be efficient enough.
- Translating from a virtual address to a physical address is more time-consuming.

However, paging has its costs:
- They enable processes to run when they are only partially loaded in main memory.
- They allow sharing of code pages among processes, reducing overall memory requirements.
- They eliminate the problem of external fragmentation and therefore the need for
  - Paging is a big improvement over segmentation.

Summary

Can greatly reduce overall memory requirements for commonly used applications.
- Process requests the same program
- The OS keeps track of available resident code in memory and reuses them if a new
  system call
- The user program (e.g., emacs) marks text segment of a program as resident with a
  but the virtual address map to the same physical address.
- A shared page may exist in different parts of the virtual address space of each process.
- Sharing of pages is similar to the way threads share text and memory with each other.
- Shared code must be resident, that means the process that are using it cannot
  change it (e.g., no idea in resident code).

Paging allows sharing of memory across processes, since memory used by a