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

2. then selectively allocates pages to frames in memory and

1. divides and assigns processes to fixed sized pages.

Page

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

**Today: Paging**

---

*Dynamic Relocation*

*Static Relocation*

*Unpaging*

---

Last Class: Memory Management
Example

- Paging does not eliminate internal fragmentation (1/2 page per process)
- By dividing memory into fixed size pages, we can eliminate external allocation continuously in memory.
- The logical memory of the process is contiguous, but pages need not be contiguous.
- 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

Pagin Motivation & Features
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.
- Page frames are contained in the page table, which maps virtual addresses to actual physical addresses in memory.
- The OS lays the process down on pages and the paging hardware translates virtual addresses into physical addresses.
- Processes generally contain pages that are not allocated.

**Virtual Address:**

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

- Paging is a form of dynamic relocation, where each virtual address is bound by the paging hardware to a physical address.
- Think of the page table as a set of relocation registers, one for each frame.
- Mapping is invisible to the process; the OS maintains the mapping and the hardware does the translation.
- Protection is provided with the same mechanisms as used in dynamic relocation.

Paging Hardware: Practical Details

- Page size (frame sizes) are typically a power of 2 between 512 bytes and 8192 bytes per page.
- Powers of 2 make the translation of virtual addresses into physical addresses easier. For example, given:
  1. virtual address space of size $2^m$ bytes and a page of size $2^n$ bytes
  2. the high order $m-n$ bits of a virtual address select the page frame
  3. the low order $n$ bits select the offset in the page frame

- $p$: page number
- $d$: page offset
- $m-n$: page size
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>
</tr>
</thead>
<tbody>
<tr>
<td>256 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>224 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>224 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>192 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>192 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>160 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>160 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>96 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>96 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>64 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>64 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>32 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>32 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>32 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>32 bytes</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>

Memory size = 256 bytes
Page size = 16 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 1, do the virtual to physical translation.

- What part is p, and q?

- Byte increments?

- How many bits for an address? Assume we can address only 1 word (4)

---

### Address Translation Example
The Translation Lookaside Buffer (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.
Details

Multilevel Paging: If the virtual address space is huge, page tables get too big, and many systems use a multilevel paging scheme (refer for OSC). (Note: The multilevel paging scheme allows for efficient memory management by dividing the address space into smaller, manageable pages and using levels of page tables to map between virtual and physical addresses.)

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

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

Replacing an existing entry if the TLB is full.
6. As process executes, OS loads TLB entries as each page is accessed.

5. OS starts process.
4. OS marks all TLB entries as invalid (flushes the TLB).
3. The OS puts each page in a frame and then puts the frame number in
   the corresponding entry in the page table.
2. If no free frames are found, then allocate these frames to pages. Else free
   frames that are no longer needed.

Initializing Memory when Starting a Process
Paging requires more complex OS to maintain the page table.

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 compaction.
- They reduce the overhead associated with page table maintenance.

**Summary**

- Paging allows sharing of memory across processes, since memory used by a process no longer needs to be contiguous.
- 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 short page fault.
- 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.
- Change in text is not visible to the process that is using it, but cannot.

- The OS keeps track of available resident code.