CSE 4733/6733 - Operating System 1

S. Torri, T. Ritter

Mississippi State University

2025-02-18

CSE 4733/6733

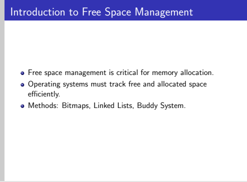
CSE 4733/6733 - Operating System 1

S. Torri, T. Ritter

Mississippi State University

Introduction to Free Space Management

- Free space management is critical for memory allocation.
- Operating systems must track free and allocated space efficiently.
- Methods: Bitmaps, Linked Lists, Buddy System.



Introduction to Free Space Management

- Free space management is critical for memory allocation.
- Operating systems must track free and allocated space efficiently.
- Methods: Bitmaps, Linked Lists, Buddy System.

1. Before diving into paging and address translation, let's understand how the OS manages free space in memory.
2. Think of free space like available parking spots; the OS must track which spots are empty and assign them efficiently.

Key Reasons Why Free Space Management Matters

- Prevents Wasted Memory
- Optimizes Performance
- Supports Dynamic Memory Needs
- Ensures Stability & Predictability

Key Reasons Why Free Space Management Matters

- Prevents Wasted Memory
- Optimizes Performance
- Supports Dynamic Memory Needs
- Ensures Stability & Predictability

1. **Prevents Wasted Memory**: Without effective free space tracking, allocated memory can become scattered, leading to fragmentation. Poor management results in unusable memory blocks, even when there is "free" space.
2. **Optimizes Performance**: Fast allocation and deallocation ensure low latency for memory requests. Reducing fragmentation allows the system to minimize paging overhead (this connects to your TLB discussion).
3. **Supports Dynamic Memory Needs**: OS memory demands change dynamically. Efficient tracking enables smooth scaling for multi-process workloads. Poor memory tracking leads to high page table overhead, increasing TLB misses.

Key Reasons Why Free Space Management Matters

- Prevents Wasted Memory
- Optimizes Performance
- Supports Dynamic Memory Needs
- Ensures Stability & Predictability

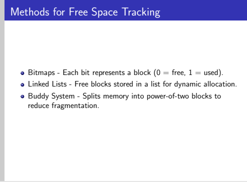
Key Reasons Why Free Space Management Matters

- Prevents Wasted Memory
- Optimizes Performance
- Supports Dynamic Memory Needs
- Ensures Stability & Predictability

1. **Ensures Stability and Predictability**: Without structured memory tracking, memory allocations may fail unpredictably. OS crashes and instability can occur when free memory isn't managed properly.

Methods for Free Space Tracking

- Bitmaps - Each bit represents a block (0 = free, 1 = used).
- Linked Lists - Free blocks stored in a list for dynamic allocation.
- Buddy System - Splits memory into power-of-two blocks to reduce fragmentation.



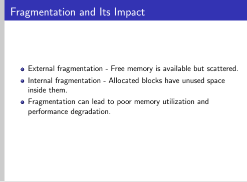
Methods for Free Space Tracking

- Bitmaps - Each bit represents a block (0 = free, 1 = used).
- Linked Lists - Free blocks stored in a list for dynamic allocation.
- Buddy System - Splits memory into power-of-two blocks to reduce fragmentation.

1. - "Bitmaps are simple but require scanning to find free blocks." - "Linked lists allow quick merges but add pointer overhead." - "The Buddy System helps balance efficiency and performance."

Fragmentation and Its Impact

- External fragmentation - Free memory is available but scattered.
- Internal fragmentation - Allocated blocks have unused space inside them.
- Fragmentation can lead to poor memory utilization and performance degradation.

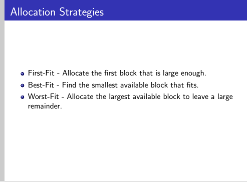

Fragmentation and Its Impact

- External fragmentation - Free memory is available but scattered.
- Internal fragmentation - Allocated blocks have unused space inside them.
- Fragmentation can lead to poor memory utilization and performance degradation.

1. - "Imagine a bookshelf with empty spaces between books - that's fragmentation." - "External fragmentation wastes large portions of memory."

Allocation Strategies

- First-Fit - Allocate the first block that is large enough.
- Best-Fit - Find the smallest available block that fits.
- Worst-Fit - Allocate the largest available block to leave a large remainder.

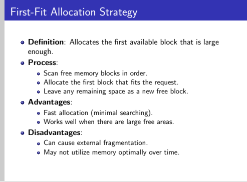

Allocation Strategies

- First-Fit - Allocate the first block that is large enough.
- Best-Fit - Find the smallest available block that fits.
- Worst-Fit - Allocate the largest available block to leave a large remainder.

1. - "First-Fit is fast but can cause fragmentation." - "Best-Fit minimizes leftover space but increases search time." - "Worst-Fit can lead to larger continuous free blocks."

First-Fit Allocation Strategy

- **Definition**: Allocates the first available block that is large enough.
- **Process**:
  - Scan free memory blocks in order.
  - Allocate the first block that fits the request.
  - Leave any remaining space as a new free block.
- **Advantages**:
  - Fast allocation (minimal searching).
  - Works well when there are large free areas.
- **Disadvantages**:
  - Can cause external fragmentation.
  - May not utilize memory optimally over time.

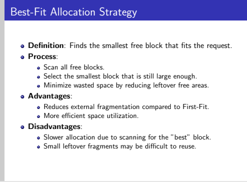

First-Fit Allocation Strategy

- **Definition**: Allocates the first available block that is large enough.
- **Process**:
  - Scan free memory blocks in order.
  - Allocate the first block that fits the request.
  - Leave any remaining space as a new free block.
- **Advantages**:
  - Fast allocation (minimal searching).
  - Works well when there are large free areas.
- **Disadvantages**:
  - Can cause external fragmentation.
  - May not utilize memory optimally over time.

1. First-Fit is efficient when memory is mostly free but can degrade performance due to fragmentation.

Best-Fit Allocation Strategy

- **Definition**: Finds the smallest free block that fits the request.
- **Process**:
  - Scan all free blocks.
  - Select the smallest block that is still large enough.
  - Minimize wasted space by reducing leftover free areas.
- **Advantages**:
  - Reduces external fragmentation compared to First-Fit.
  - More efficient space utilization.
- **Disadvantages**:
  - Slower allocation due to scanning for the "best" block.
  - Small leftover fragments may be difficult to reuse.



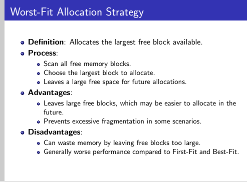
Best-Fit Allocation Strategy

- **Definition**: Finds the smallest free block that fits the request.
- **Process**:
  - Scan all free blocks.
  - Select the smallest block that is still large enough.
  - Minimize wasted space by reducing leftover free areas.
- **Advantages**:
  - Reduces external fragmentation compared to First-Fit.
  - More efficient space utilization.
- **Disadvantages**:
  - Slower allocation due to scanning for the "best" block.
  - Small leftover fragments may be difficult to reuse.

1. Best-Fit is good for maximizing memory usage but can lead to tiny, unusable fragments.

Worst-Fit Allocation Strategy

- **Definition**: Allocates the largest free block available.
- **Process**:
  - Scan all free memory blocks.
  - Choose the largest block to allocate.
  - Leaves a large free space for future allocations.
- **Advantages**:
  - Leaves large free blocks, which may be easier to allocate in the future.
  - Prevents excessive fragmentation in some scenarios.
- **Disadvantages**:
  - Can waste memory by leaving free blocks too large.
  - Generally worse performance compared to First-Fit and Best-Fit.



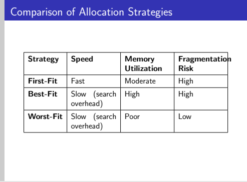
Worst-Fit Allocation Strategy

- **Definition**: Allocates the largest free block available.
- **Process**:
  - Scan all free memory blocks.
  - Choose the largest block to allocate.
  - Leaves a large free space for future allocations.
- **Advantages**:
  - Leaves large free blocks, which may be easier to allocate in the future.
  - Prevents excessive fragmentation in some scenarios.
- **Disadvantages**:
  - Can waste memory by leaving free blocks too large.
  - Generally worse performance compared to First-Fit and Best-Fit.

1. Worst-Fit aims to maintain large free areas but often results in poor memory utilization.

Comparison of Allocation Strategies

Strategy	**Speed**	**Memory Utilization**	**Fragmentation Risk**
First-Fit	Fast	Moderate	High
Best-Fit	Slow (search overhead)	High	High
Worst-Fit	Slow (search overhead)	Poor	Low

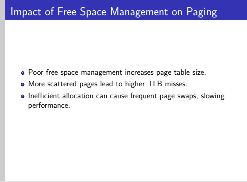

Comparison of Allocation Strategies

Strategy	Speed	Memory Utilization	Fragmentation Risk
First-Fit	Fast	Moderate	High
Best-Fit	Slow (search overhead)	High	High
Worst-Fit	Slow (search overhead)	Poor	Low

1. No single strategy is perfect—trade-offs depend on system workload and memory usage patterns.
2. Best-Fit is great for maximizing memory usage but may lead to fragmentation.
3. First-Fit is efficient when memory is mostly free, but fragmentation becomes a concern over time.
4. Worst-Fit is rarely used in practice due to inefficient memory utilization.

Impact of Free Space Management on Paging

- Poor free space management increases page table size.
- More scattered pages lead to higher TLB misses.
- Inefficient allocation can cause frequent page swaps, slowing performance.

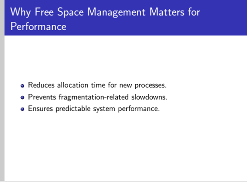

Impact of Free Space Management on Paging

- Poor free space management increases page table size.
- More scattered pages lead to higher TLB misses.
- Inefficient allocation can cause frequent page swaps, slowing performance.

1. - "Poor free space management means more fragmented pages, which increases lookup times." - "Efficient allocation reduces page table overhead."

Why Free Space Management Matters for Performance

- Reduces allocation time for new processes.
- Prevents fragmentation-related slowdowns.
- Ensures predictable system performance.

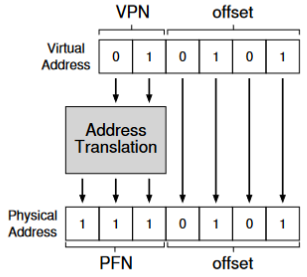


Why Free Space Management Matters for Performance

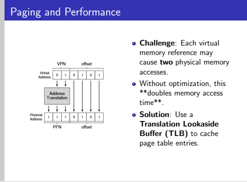
- Reduces allocation time for new processes.
- Prevents fragmentation-related slowdowns.
- Ensures predictable system performance.

1. - "Without efficient free space tracking, memory allocation can slow down over time." - "Well-managed memory keeps the system responsive and efficient."

Paging and Performance



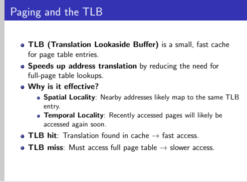
- **Challenge**: Each virtual memory reference may cause **two** physical memory accesses.
- Without optimization, this **doubles memory access time**.
- **Solution**: Use a **Translation Lookaside Buffer (TLB)** to cache page table entries.



1. **Why does this matter?** Every memory access requires looking up a page table entry, which adds overhead.
2. **Real-world analogy:** Imagine checking a physical map every time you drive to a location vs. having a GPS preloaded with frequent routes.
3. **Key takeaway:** Without caching, paging becomes expensive. This is where the TLB comes in.

Paging and the TLB

- **TLB (Translation Lookaside Buffer)** is a small, fast cache for page table entries.
- **Speeds up address translation** by reducing the need for full-page table lookups.
- **Why is it effective?**
  - **Spatial Locality**: Nearby addresses likely map to the same TLB entry.
  - **Temporal Locality**: Recently accessed pages will likely be accessed again soon.
- **TLB hit**: Translation found in cache → fast access.
- **TLB miss**: Must access full page table → slower access.



Paging and the TLB

- **TLB (Translation Lookaside Buffer)** is a small, fast cache for page table entries.
- **Speeds up address translation** by reducing the need for full-page table lookups.
- **Why is it effective?**
  - **Spatial Locality**: Nearby addresses likely map to the same TLB entry.
  - **Temporal Locality**: Recently accessed pages will likely be accessed again soon.
- **TLB hit**: Translation found in cache → fast access.
- **TLB miss**: Must access full page table → slower access.

1. **Why is this important?** Without a TLB, every virtual memory reference requires a full-page table lookup, which slows performance.
2. **TLB hit rate:** A high hit rate means the TLB is doing its job well. A low hit rate means more page table lookups, causing slowdowns.
3. **Analogy:** Think of a **TLB like a speed dial** on your phone. If a contact is in speed dial (cache), you call them instantly (fast lookup). Otherwise, you must go through the contact list (slow lookup).

Paging and the TLB

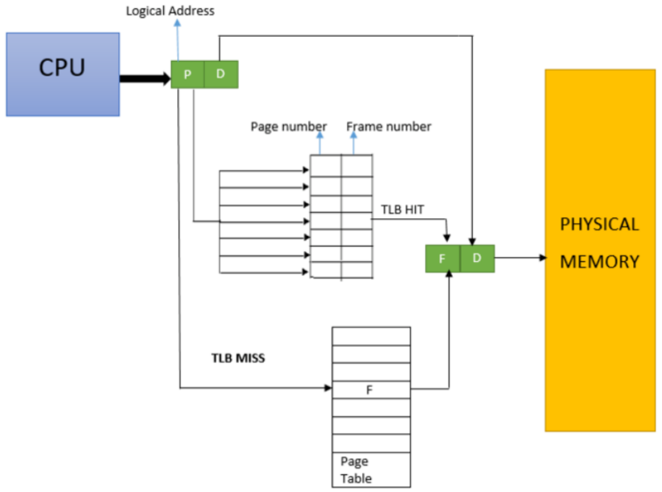
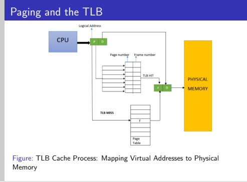


Figure: TLB Cache Process: Mapping Virtual Addresses to Physical Memory



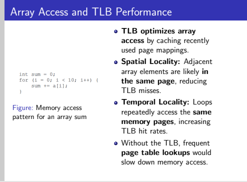
1. **What is happening in the diagram?** The CPU generates a logical address with a page number and offset. If the TLB contains the mapping (hit), it directly provides the frame number. If not, the system consults the full page table (miss), causing a delay.
2. **Real-world analogy:** Think of the TLB like a frequent contacts list in a phone. If a number is saved, you dial it instantly. If not, you must search manually, taking extra time.
3. **Key takeaway:** TLBs are critical for high-performance memory access and reducing paging overhead.

Array Access and TLB Performance

```
int sum = 0;
for (i = 0; i < 10; i++) {
    sum += a[i];
}
```

Figure: Memory access pattern for an array sum

- **TLB optimizes array access** by caching recently used page mappings.
- **Spatial Locality:** Adjacent array elements are likely **in the same page**, reducing TLB misses.
- **Temporal Locality:** Loops repeatedly access the **same memory pages**, increasing TLB hit rates.
- Without the TLB, frequent **page table lookups** would slow down memory access.



Array Access and TLB Performance

```
int sum = 0;
for (i = 0; i < 10; i++) {
    sum += a[i];
}
```

Figure: Memory access pattern for an array sum

- **TLB optimizes array access** by caching recently used page mappings.
- **Spatial Locality:** Adjacent array elements are likely **in the same page**, reducing TLB misses.
- **Temporal Locality:** Loops repeatedly access the **same memory pages**, increasing TLB hit rates.
- Without the TLB, frequent **page table lookups** would slow down memory access.

1. **How does TLB help?** Instead of looking up every array index in the page table, the TLB caches recent translations, making repeated accesses much faster.
2. **Analogy:** Think of scanning a bookshelf. If the books are in order (**spatial locality**), you find them quickly. If you keep reading the same book (**temporal locality**), there's no need to search again.
3. **Key takeaway:** The TLB greatly enhances performance by reducing the need for page table lookups, especially in **looped array operations**.

Array Example: Virtual Page Numbers (VPN) and TLB Performance

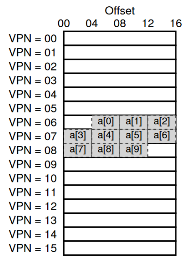


Figure: Array mapped across multiple Virtual Page Numbers (VPN)

- **Contiguous array elements** span multiple **Virtual Page Numbers (VPNs)**.
- If the entire array **fits within the TLB**, access is fast.
- If the array **exceeds TLB coverage**, frequent page table lookups slow down performance.
- **Locality of Reference:** Nearby elements are likely accessed together, benefiting from caching.

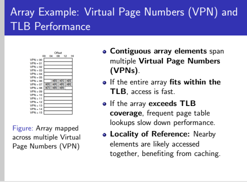

Array Example: Virtual Page Numbers (VPN) and TLB Performance

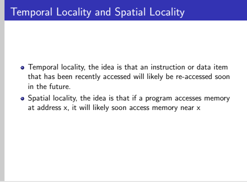
Figure: Array mapped across multiple Virtual Page Numbers (VPN)

- **Contiguous array elements** span multiple **Virtual Page Numbers (VPNs)**.
- If the entire array **fits within the TLB**, access is fast.
- If the array **exceeds TLB coverage**, frequent page table lookups slow down performance.
- **Locality of Reference:** Nearby elements are likely accessed together, benefiting from caching.

1. **What is happening here?** The array is spread across multiple virtual pages (VPNs). The TLB caches page table entries to speed up access.
2. **Why does this matter?** If the array is too large for the TLB, each new page access requires a full page table lookup, increasing memory access time.
3. **Analogy:** Imagine reading a large book where each chapter is on a different floor of a library. If you can store the whole book on your desk (TLB hit), reading is fast. If not, you have to retrieve pages frequently (TLB miss).

Temporal Locality and Spatial Locality

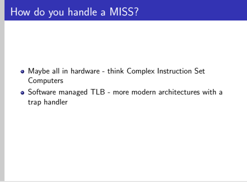
- Temporal locality, the idea is that an instruction or data item that has been recently accessed will likely be re-accessed soon in the future.
- Spatial locality, the idea is that if a program accesses memory at address x, it will likely soon access memory near x



1. TLB Reality Check:
   - Cache always has a MISS and HIT possibility - see some cache traces in the book
   - Scale of performance between CACHE and standard memory access is dramatic, a few CPU cycles vs. nanoseconds
   - Reality checks on performance are the costs and the physics involved

How do you handle a MISS?

- Maybe all in hardware - think Complex Instruction Set Computers
- Software managed TLB - more modern architectures with a trap handler

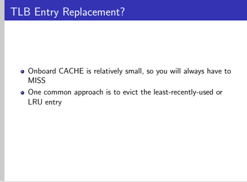

How do you handle a MISS?

- Maybe all in hardware - think Complex Instruction Set Computers
- Software managed TLB - more modern architectures with a trap handler

1. TLB Reality Check:
   - Startup, nothing loaded and have to get popular VPN to PFN loaded over time
   - Flush on a context switch?
   - How do we minimize the miss rate (or increase the hit rate)

TLB Entry Replacement?

- Onboard CACHE is relatively small, so you will always have to MISS
- One common approach is to evict the least-recently-used or LRU entry

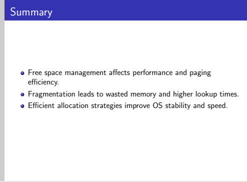

TLB Entry Replacement?

- Onboard CACHE is relatively small, so you will always have to MISS
- One common approach is to evict the least-recently-used or LRU entry

1. TLB Replacement:
   - Flush on every context switch, or have an address space identifier (ASID)
   - Support of ASID says we can have TLB that supports multiple processes at the same time
   - LRU means we have to keep track of the usage and timestamp entries
   - LRU helps with the locality of reference; old stuff is dropped for the recent access

Summary

- Free space management affects performance and paging efficiency.
- Fragmentation leads to wasted memory and higher lookup times.
- Efficient allocation strategies improve OS stability and speed.


Summary

- Free space management affects performance and paging efficiency.
- Fragmentation leads to wasted memory and higher lookup times.
- Efficient allocation strategies improve OS stability and speed.

1. - "Understanding free space management is key to optimizing memory use." - "Next, we'll look at advanced paging techniques and memory hierarchies."

References I

2025-02-18 CSE 4733/6733

└─References