Operating Systems

Memory Swapping

Lecture Overview

- Translation Look-aside Buffers (TLB)
- Page Tables
- Swapping
- Advanced Page Tables
  - Inverted Tables
  - Segmented Tables
  - Multi-level Page Tables
- Page Faults

Last Week

Memory Virtualisation

- Segmentation
- Paging

Revision: Paging

Advantages:

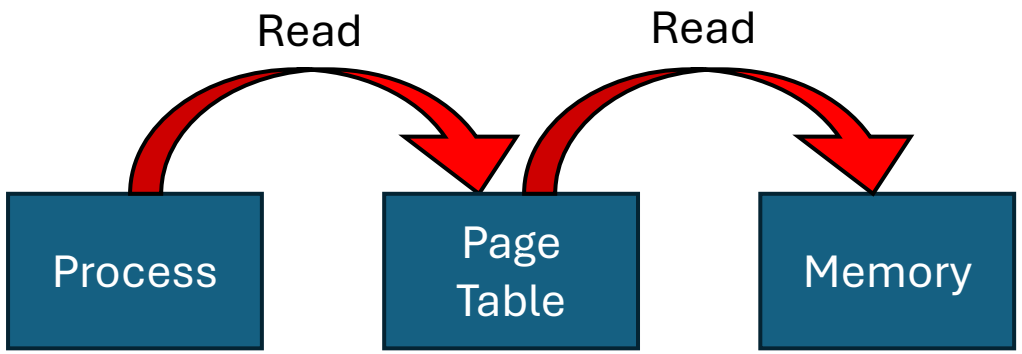
- No external fragmentation
- Easy to manage
- Swapping is easy

Disadvantages:

- Multi-level page tables…
  - Slow…

Revision: Paging

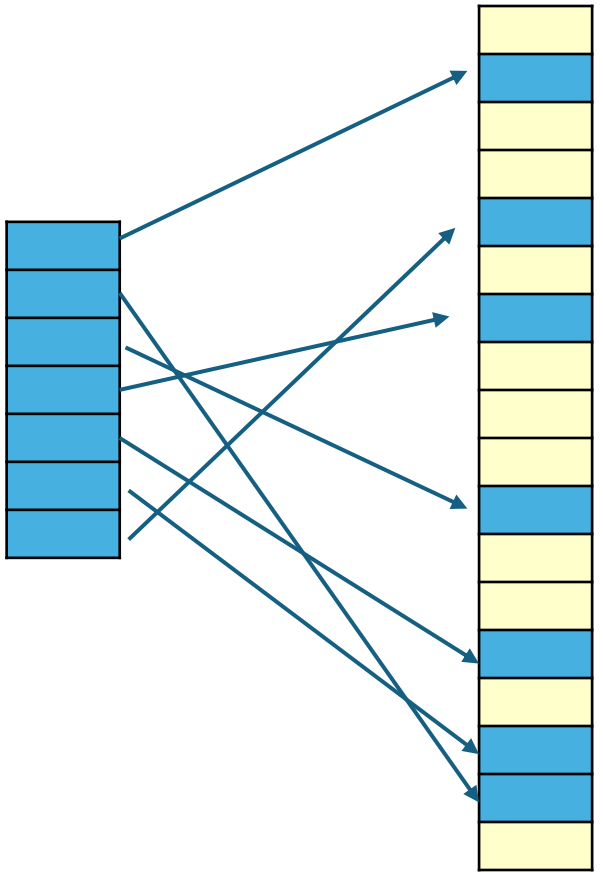
1: Get VPN (from a register)	Fast
2: Calculate address of Page Table Entry	Fast
3: Read Page Table Entry from memory	**Slow**
4: Extract Page Frame Number	Fast
5: Build the Physical Address	Fast
6: Read contents of Physical Address	**Slow**



Locality


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

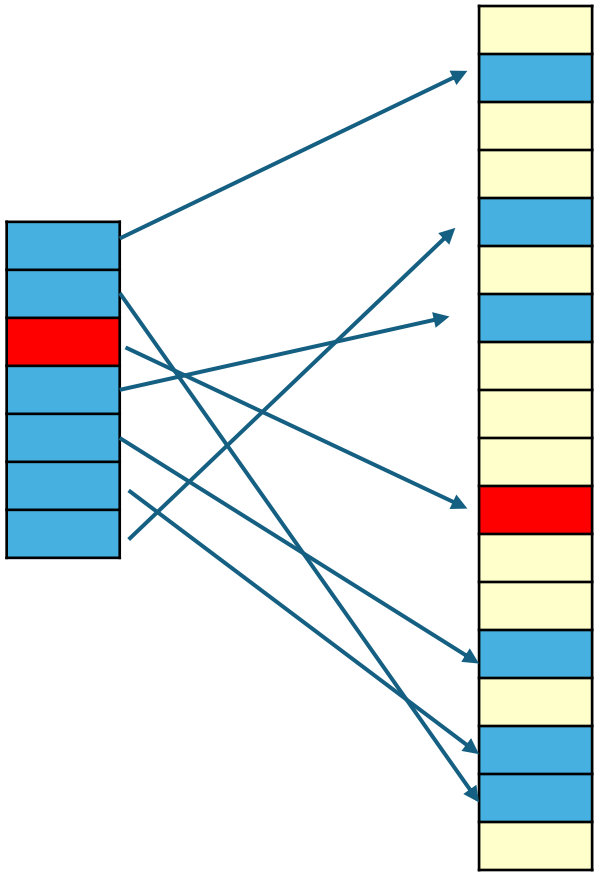
Some standard code



Locality


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

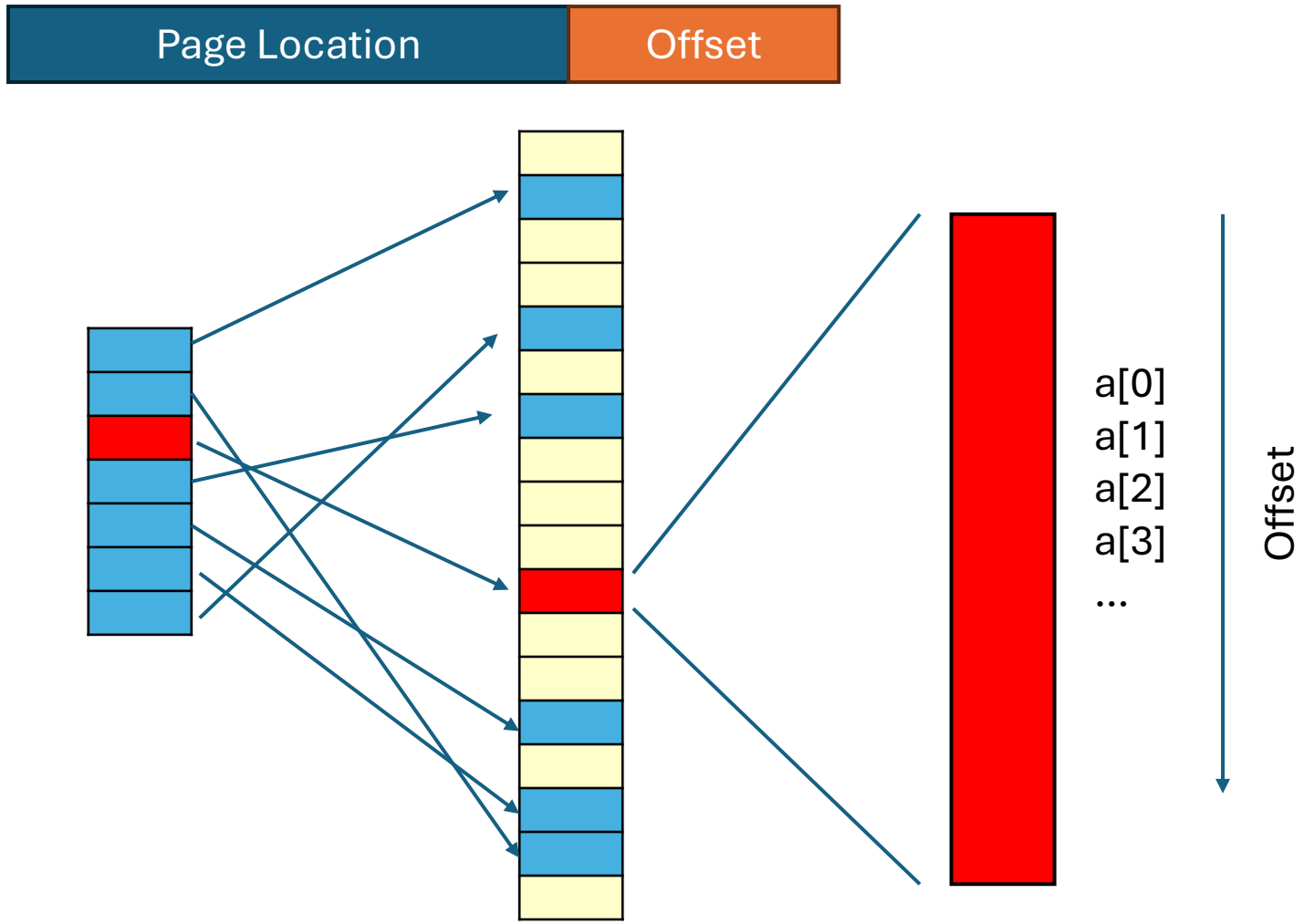
Some standard code



Locality

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

Some standard code



Why would we keep loading this same page of memory over and over again?

Locality

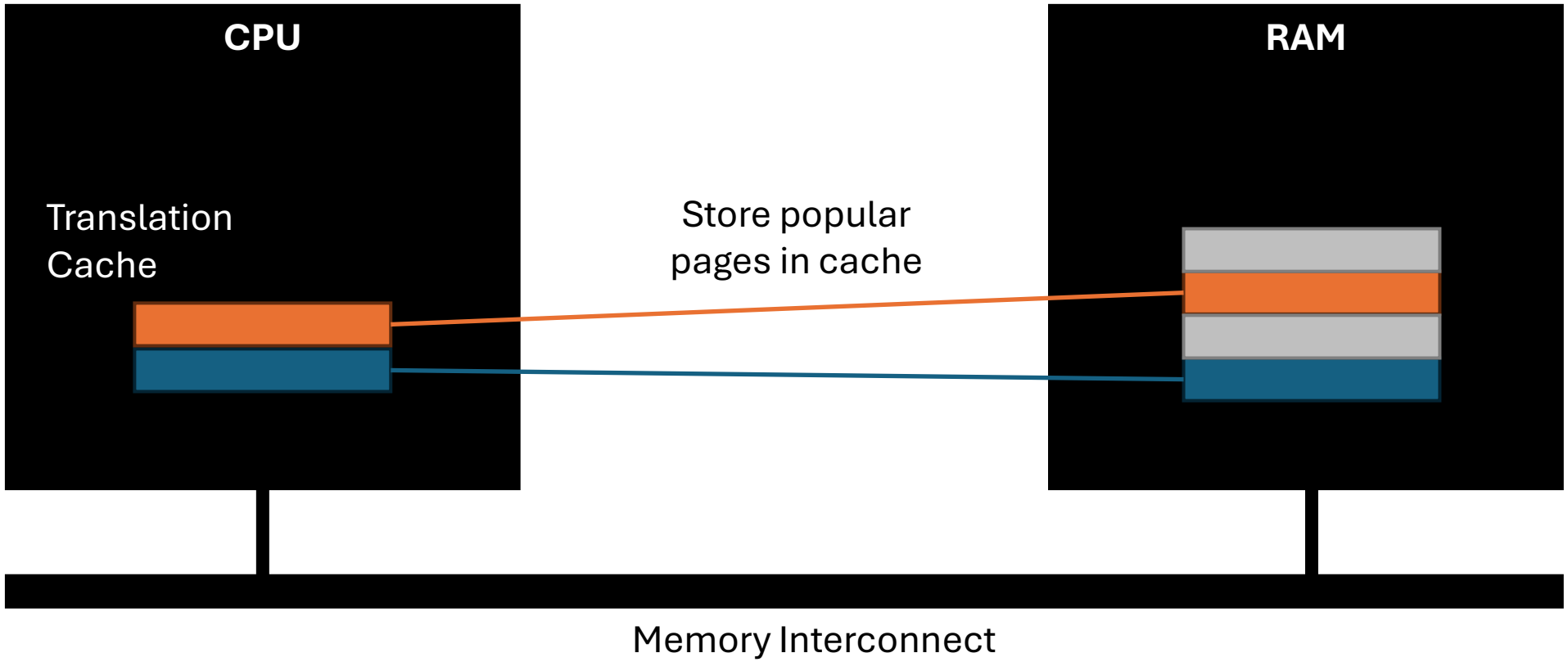
Spatial Locality

- Memory accesses in logically written programs do not access arbitrary parts of memory. Memory access location's are locally related in space.

Temporal Locality

- Memory accesses in logically written programs do not access arbitrary parts of memory. Memory accesses which occur at similar times tend to occur in similar spaces.

Strategy: Caching



Translation Lookaside Buffers (TLB)

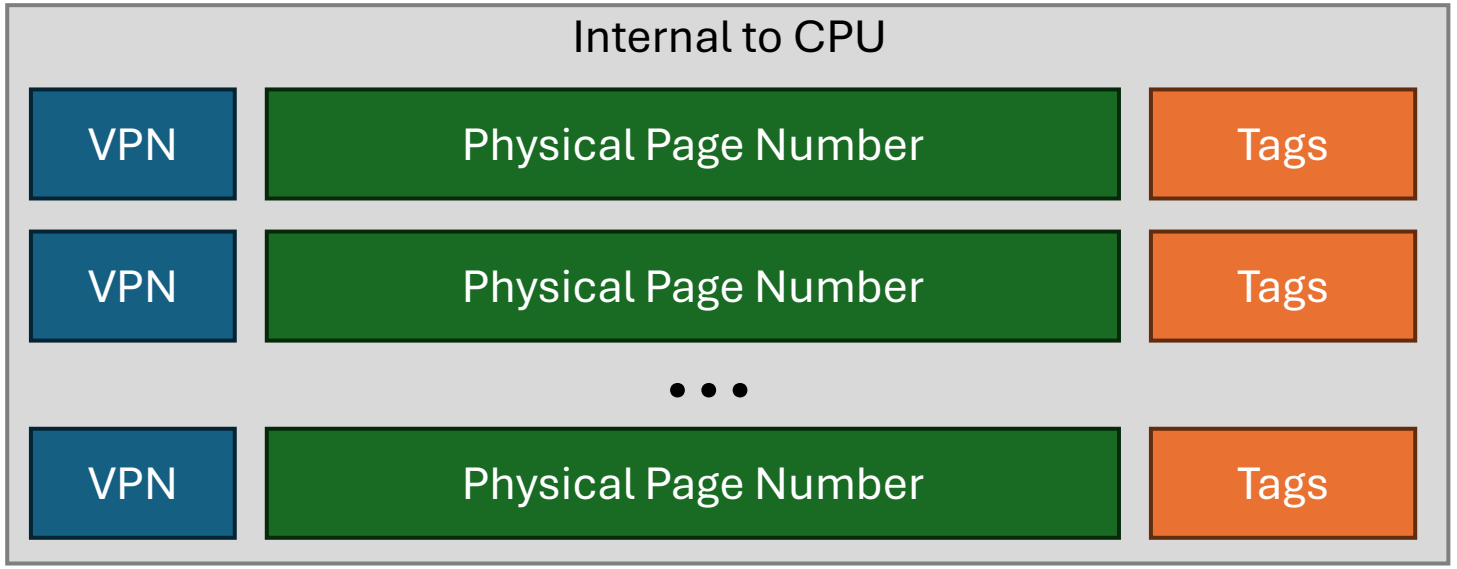
Translation Lookaside Buffer (TLB)

Goals

- Access Data Fast
  - Small table
  - Direct route to CPU internals
  - Optimised hardware for lookup

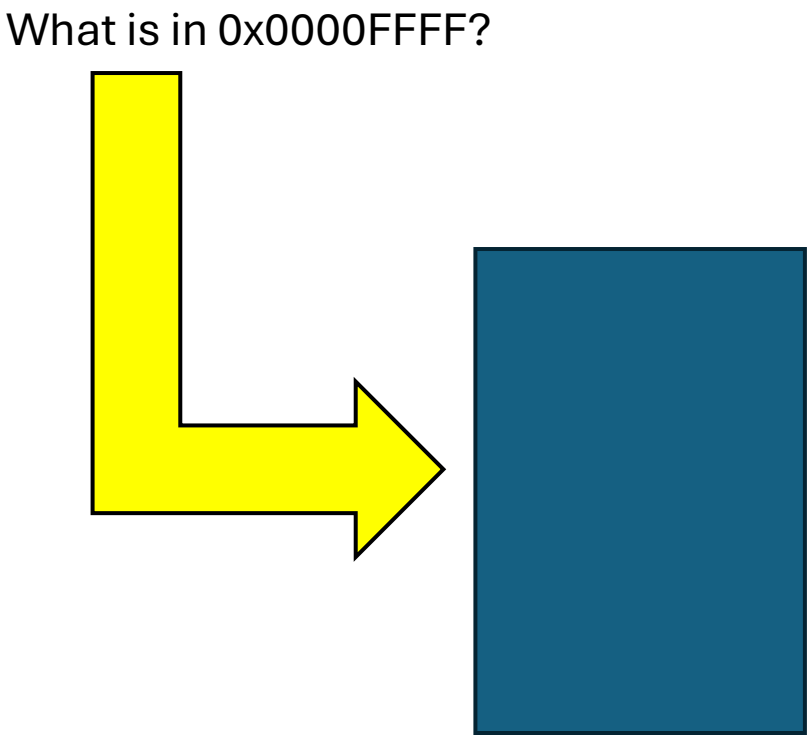
Implementation

- Content Addressable Memory

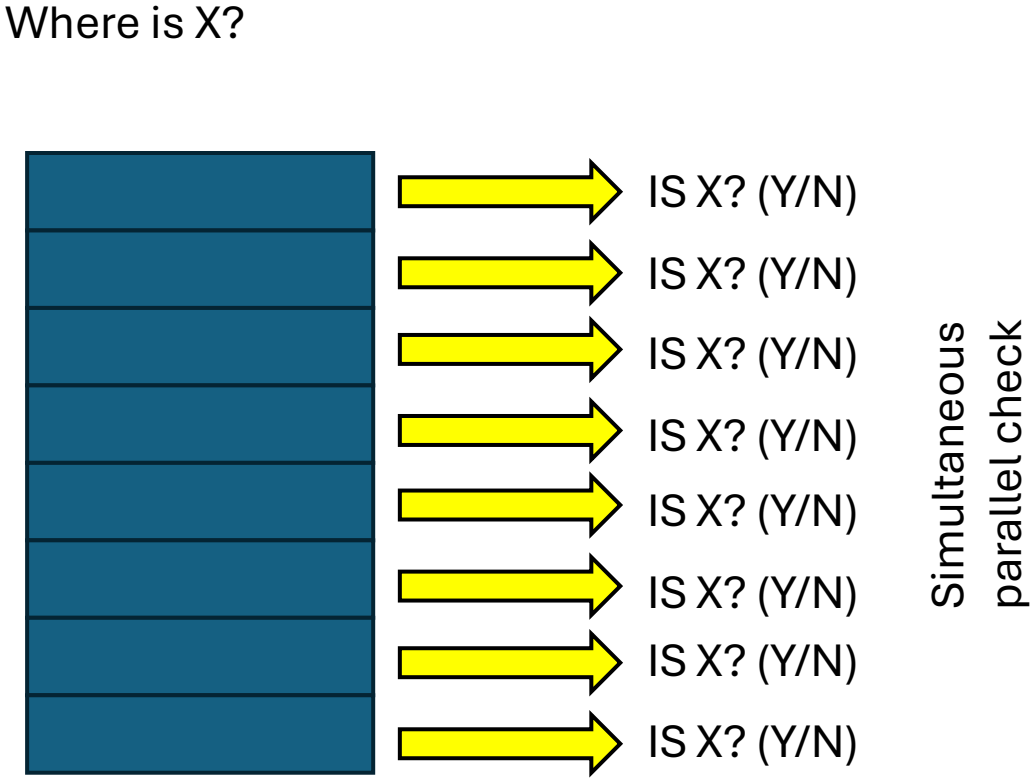


Content Addressable Memory

Normal Memory



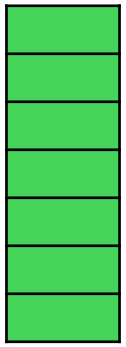
CAM



TLB in action

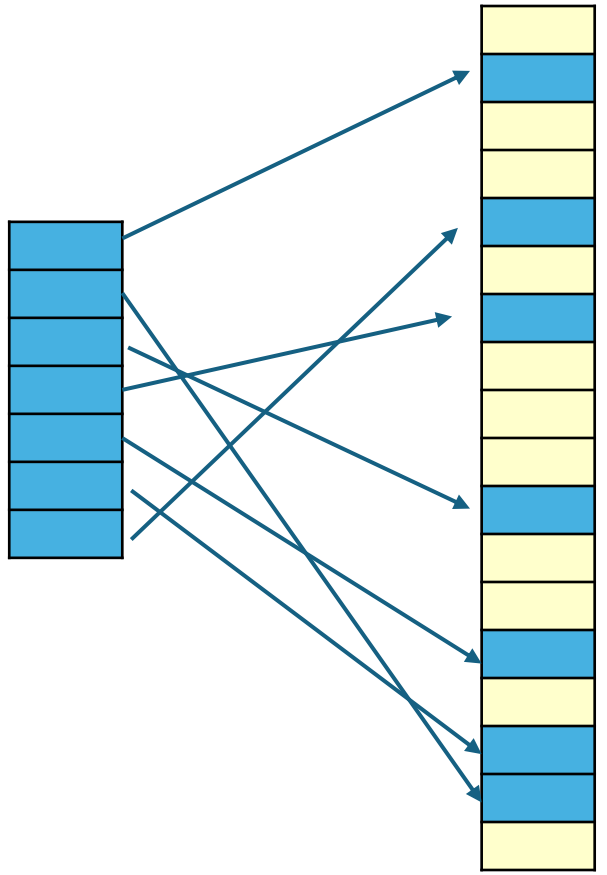
```
int a[];
int sum = 0;
for (int i=0; i < N; i++)
{
        sum += a[i];
}
```

Some standard code



TLB

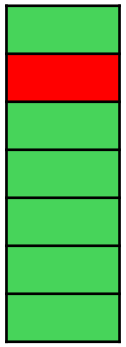
load 0x0000



TLB in action


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

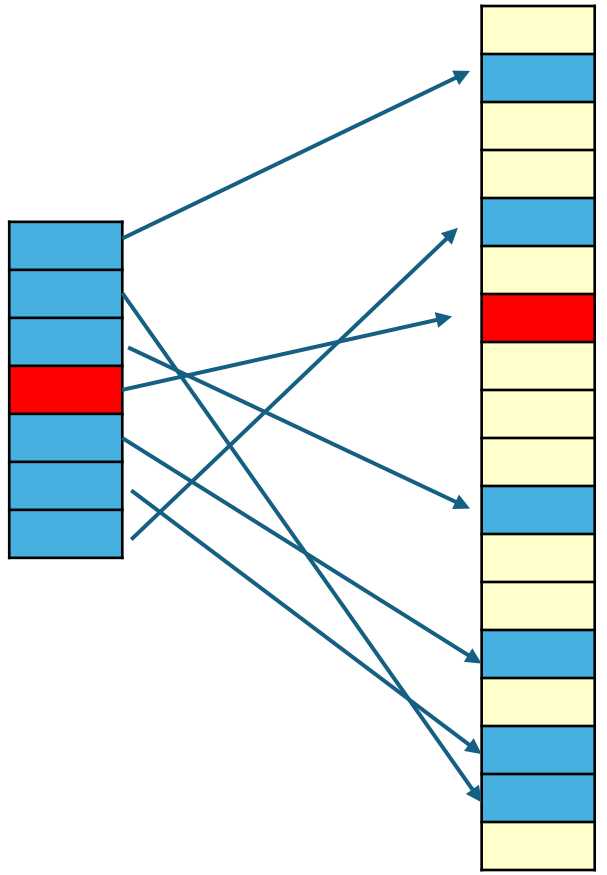
Some standard code



TLB

Store the address

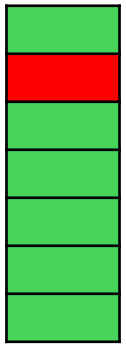
load 0x0000



TLB in action


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

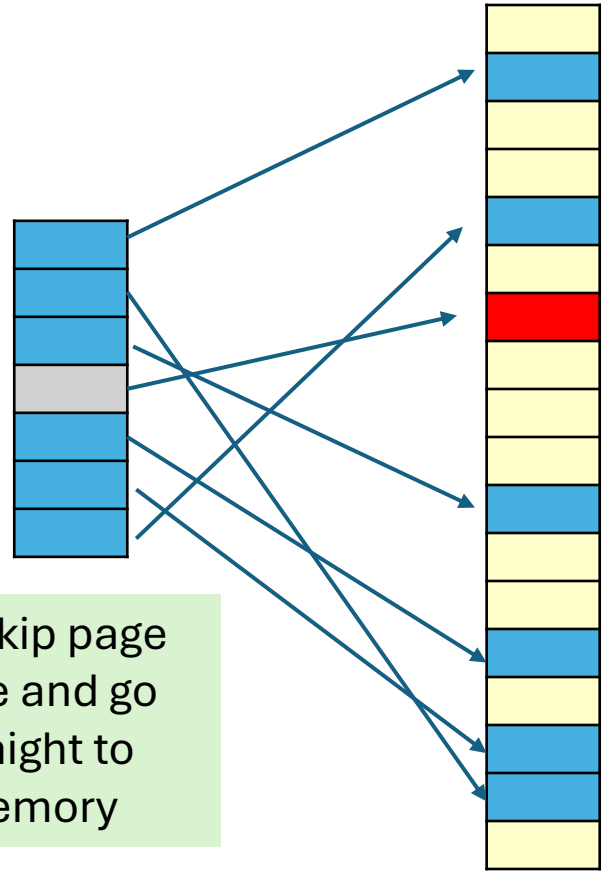
Some standard code



TLB

Store the address

load 0x0000
load 0x0004

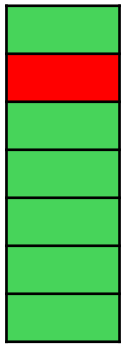


We skip page table and go straight to memory

TLB in action


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

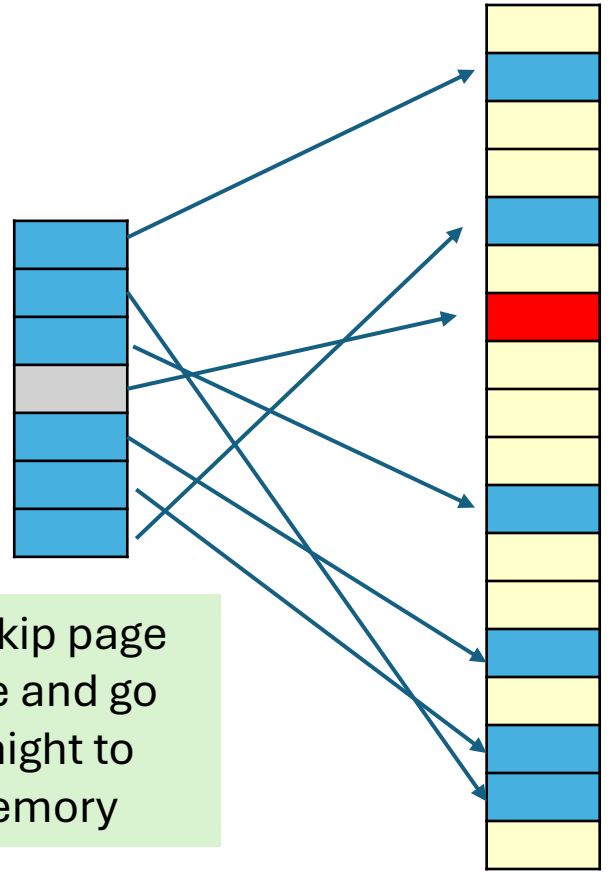
Some standard code



TLB

Store the address

load 0x0000
load 0x0004
load 0x0008



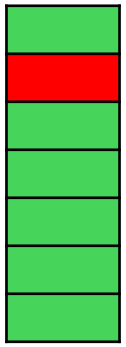
We skip page table and go straight to memory

TLB in action


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

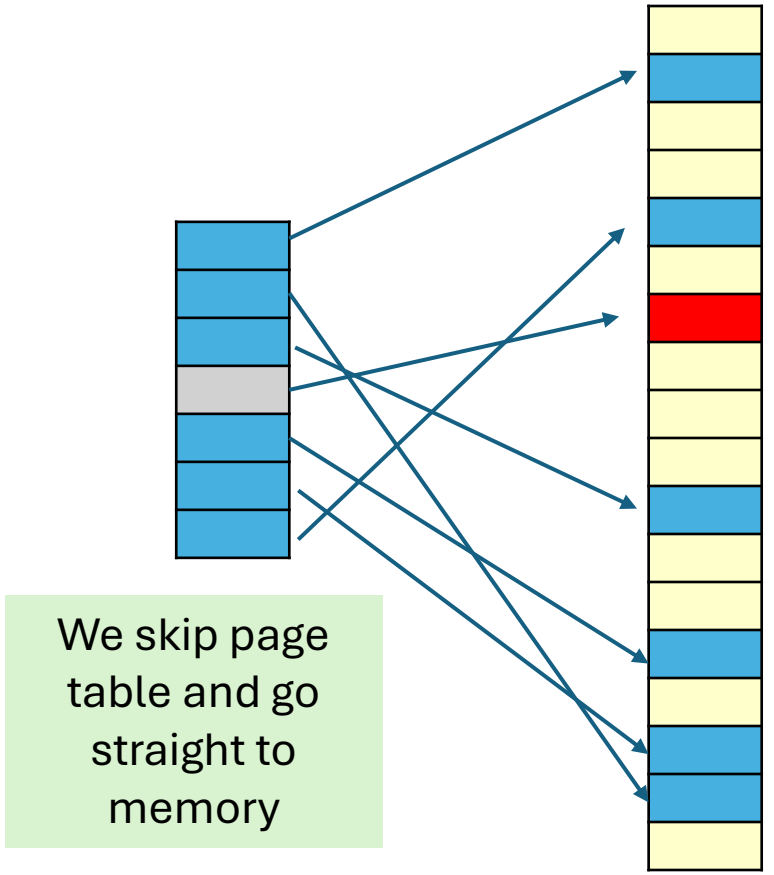
Some standard code

load 0x0000
load 0x0004
load 0x0008
…



TLB

Store the address



We skip page table and go straight to memory

TLB Summary

Problem #1

- A page table is huge

Solution #1

- Use a multi-level tree-like page table

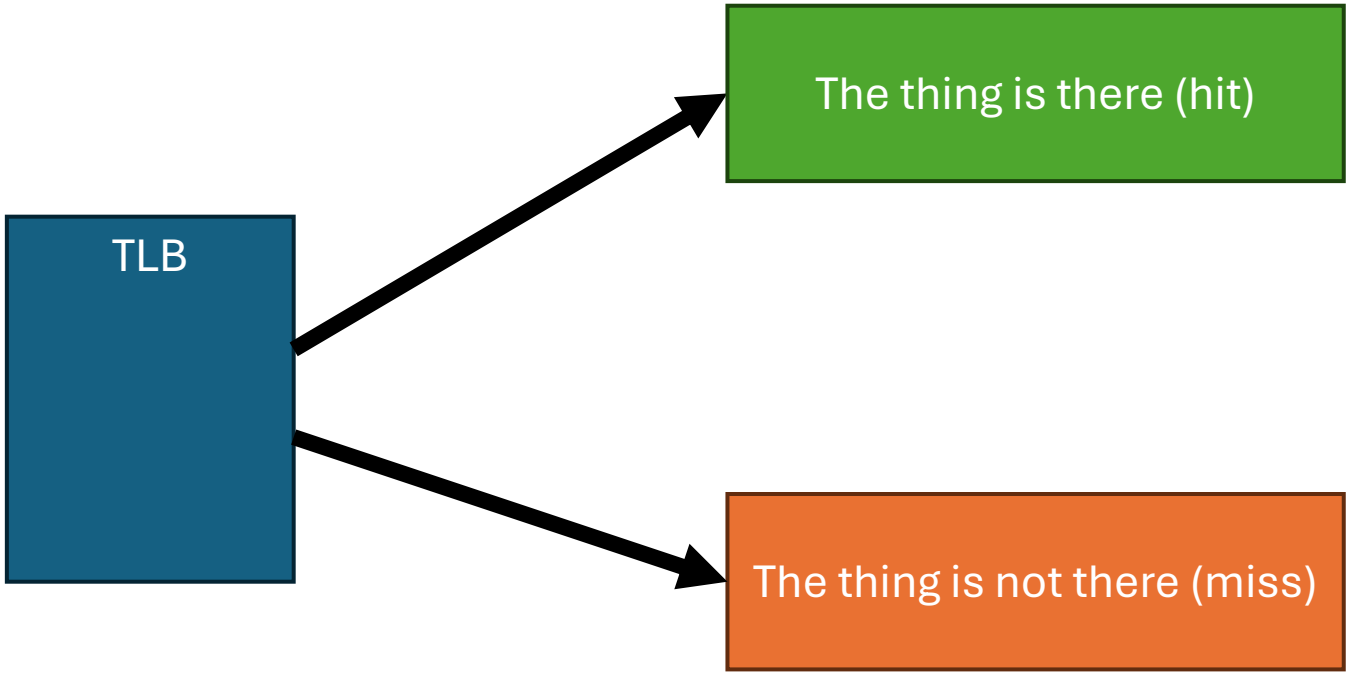
Problem #2

- Each level of the tree makes memory access slower

Solution #2

- Cache commonly used address lookups

TLB Performance?



Miss rate

$$\frac{TLB\ misses}{TLB\ lookups}$$

TLB Performance

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

Some standard code

load 0x0000

Some thinking:

Page = 4KB = 4096 bytes

32 bit int = 4 bytes

$\Rightarrow$1024 ints/page

If N = 2000?

We need 2 pages, hence 2 misses.

Miss rate = $\frac{2}{2000} = 0.001$

TLB Performance

Activity:

How could we improve TLB performance?

TLB Performance

Some Ideas:

- Page size: Big pages => less pages => less misses
- TLB size: More pages => less misses
- TLB… psychic? Correct pages => less misses

TLB Performance: Locality

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

We hit **sum** repetitively making it *temporally similar*

Page longevity?

The values of **a** are *spatially similar*

Page size?

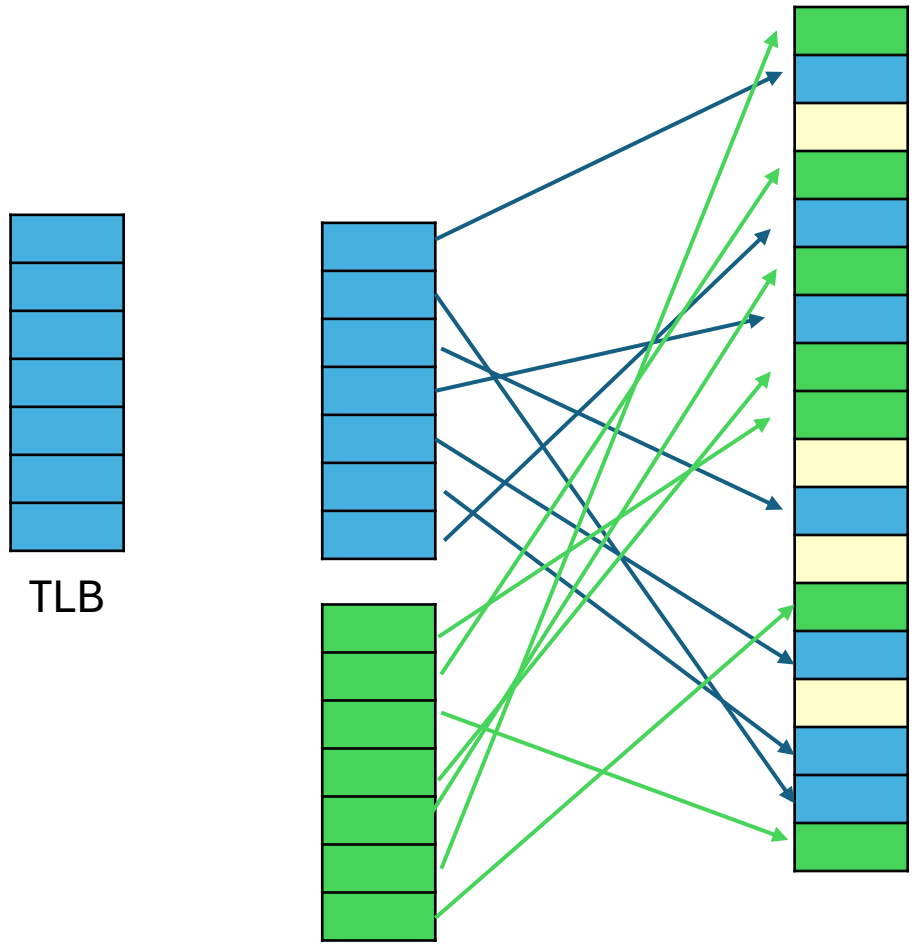
TLBs & Processes

Oh dear...

Page Replacement & Processes?

What happens on a context switch?

- Good chance everything is wrong…



Page Replacement & Processes?

What happens on a context switch?

- Good chance everything is wrong…

Option #1

- Flush (empty) the TLB?

Pros:

- Simple

Cons:

- New misses (cold start)

Page Replacement & Processes?

What happens on a context switch?

- Good chance everything is wrong…

Option #2

- Flag each TLB entry with the address space (or PID)?

Pros:

- Simple

Cons:

- Extra comparison required

Page Replacement & Processes?

What happens on a context switch?

- Good chance everything is wrong…

Multiple TLBs are sometimes split for data, instructions, regular pages, super pages

Option #3

- Big/multiple TLBs

Pros:

- Obvious

Cons:

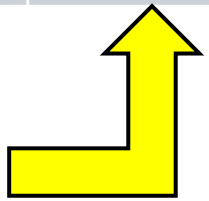
- Physical constraints limit this
- Expensive

Page Replacement & Processes

Process A

Process A

VPN	PFN	Valid	prot	ASID
10	100	1	rwx	1
-	-	-	-	-
10	170	1	rwx	2
-	-	-	-	-



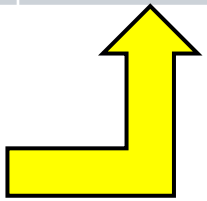
ASID can identify which pages are for you

Page Replacement & Processes

Process A

Process A

VPN	PFN	Valid	prot	ASID
10	100	1	rwx	1
-	-	-	-	-
10	170	1	rwx	2
-	-	-	-	-



Tells you what you can do with this page

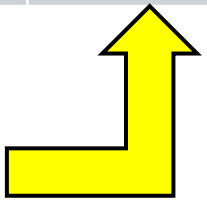
Page Replacement & Processes

Process A

Process A

VPN	PFN	Valid	prot	ASID
10	110	1	rwx	1
-	-	-	-	-
10	110	1	rwx	2
-	-	-	-	-

Two processes use the same bit of memory



This can be done but it is a manual process for intentional memory sharing

A few options for sharing

Read Only

- Two processes can share read-only data with no risk of concurrency issues

Fully Shared

- Concurrency issues
- Quick data sharing

Copy on Write

- Share as if Read-Only until told otherwise.
- Any writes trigger a copy of the offending page.

TLB Implementation

What happens on a miss?

- Hardware Managed
  - Hardware traverses the tables, finds the entry and updates the TLB
  - Hardware then triggers another TLB call
  - OS sets up the page tables and handles page faults

  x86, x86-64, ARMv8

- Software Managed
  - Miss => OS trap, do the table look-up
  - Write the mapping into the TLB using privileged instructions
  - Resume original instruction

  MIPS, SPARC, RISC-V

TLB Implementation

What happens on a miss?

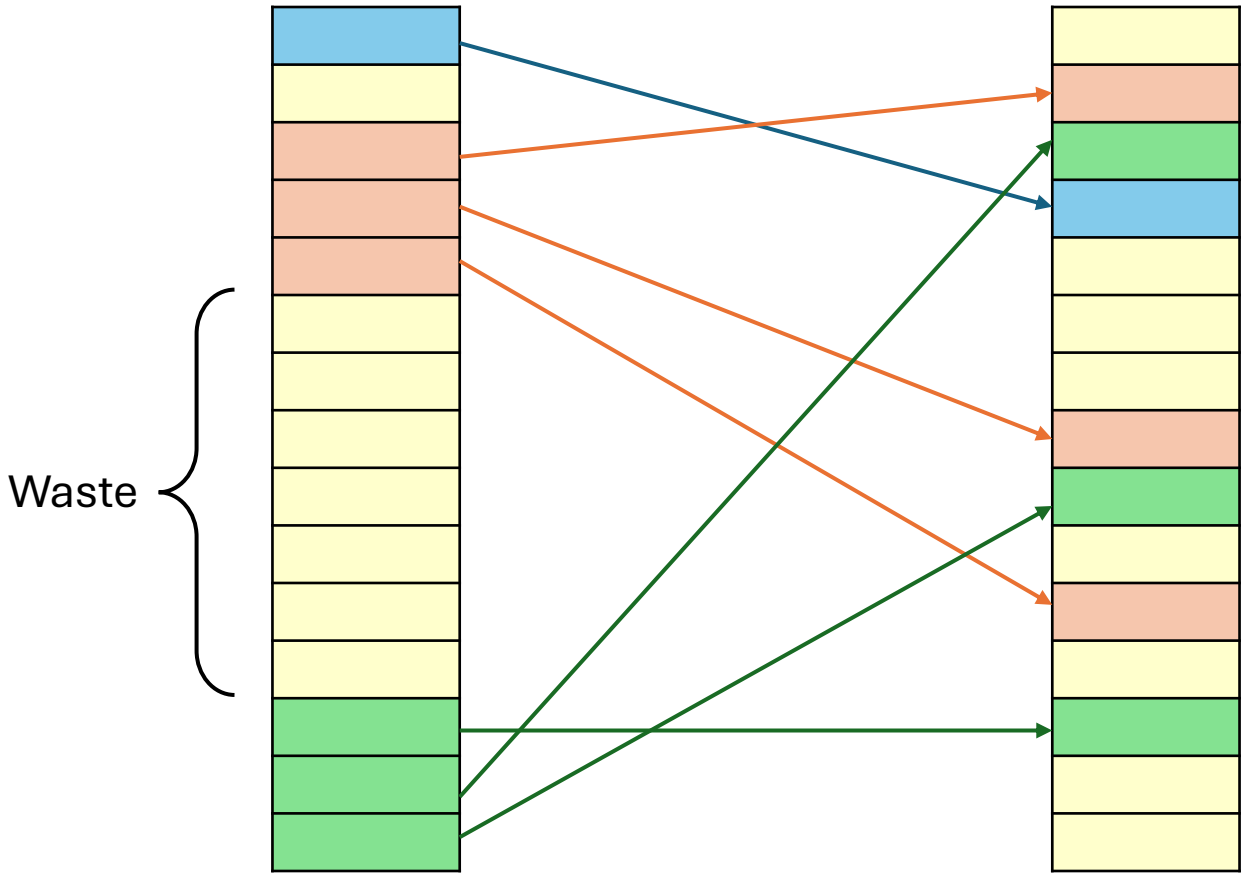
- Hardware Managed
  - Faster
  - Rigid page table structure
- Software Managed
  - More flexible
  - Slower

Paging Summary

Summary:

- Pages simplify memory
- Less external fragmentation
- Slow (can be sped up with TLB)
- Good for sequential workloads (spatial locality)
- Complicated by context switches
- Page Tables are Big!

Page Tables & Waste

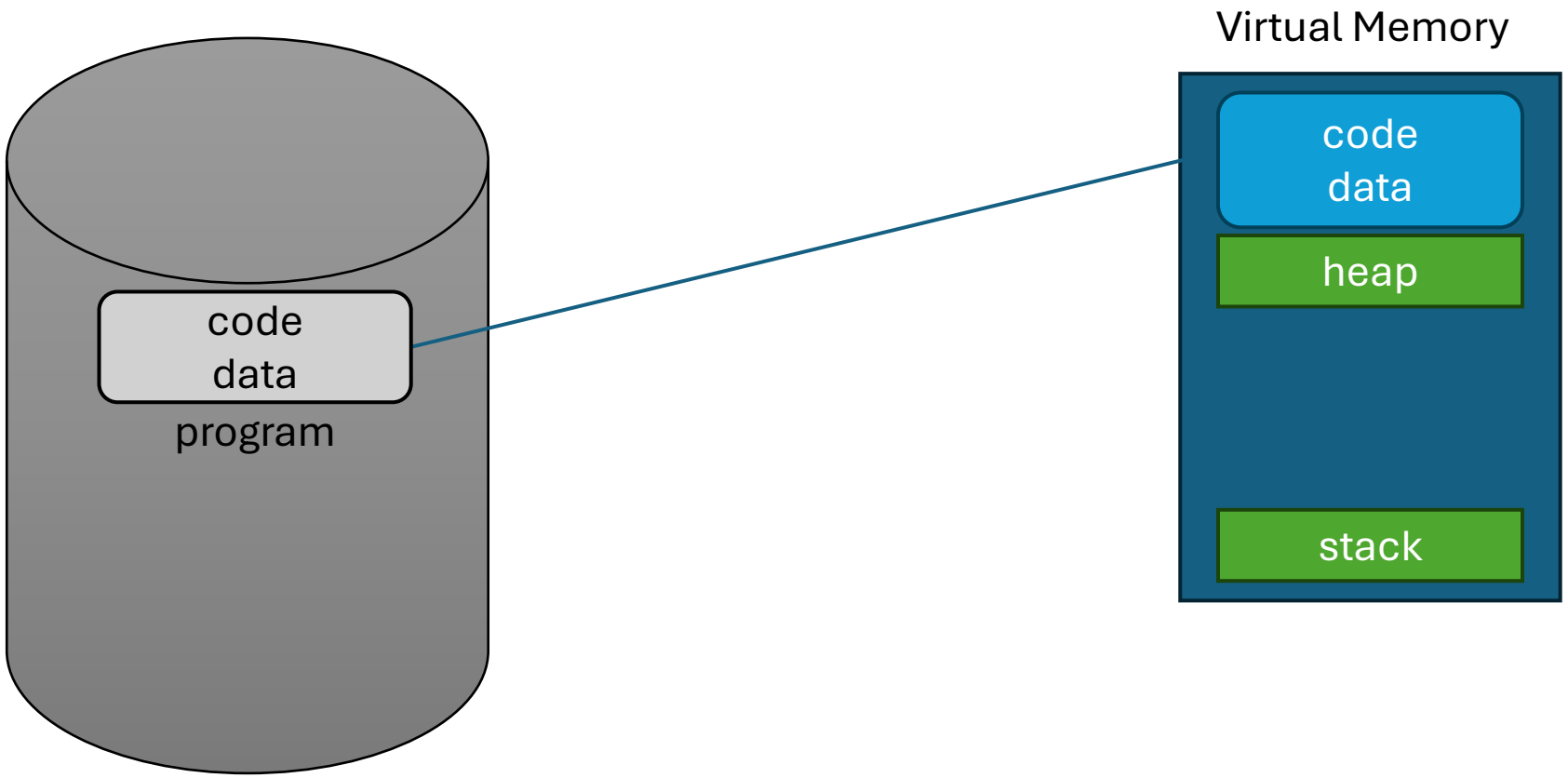


Where is the memory we aren’t using?

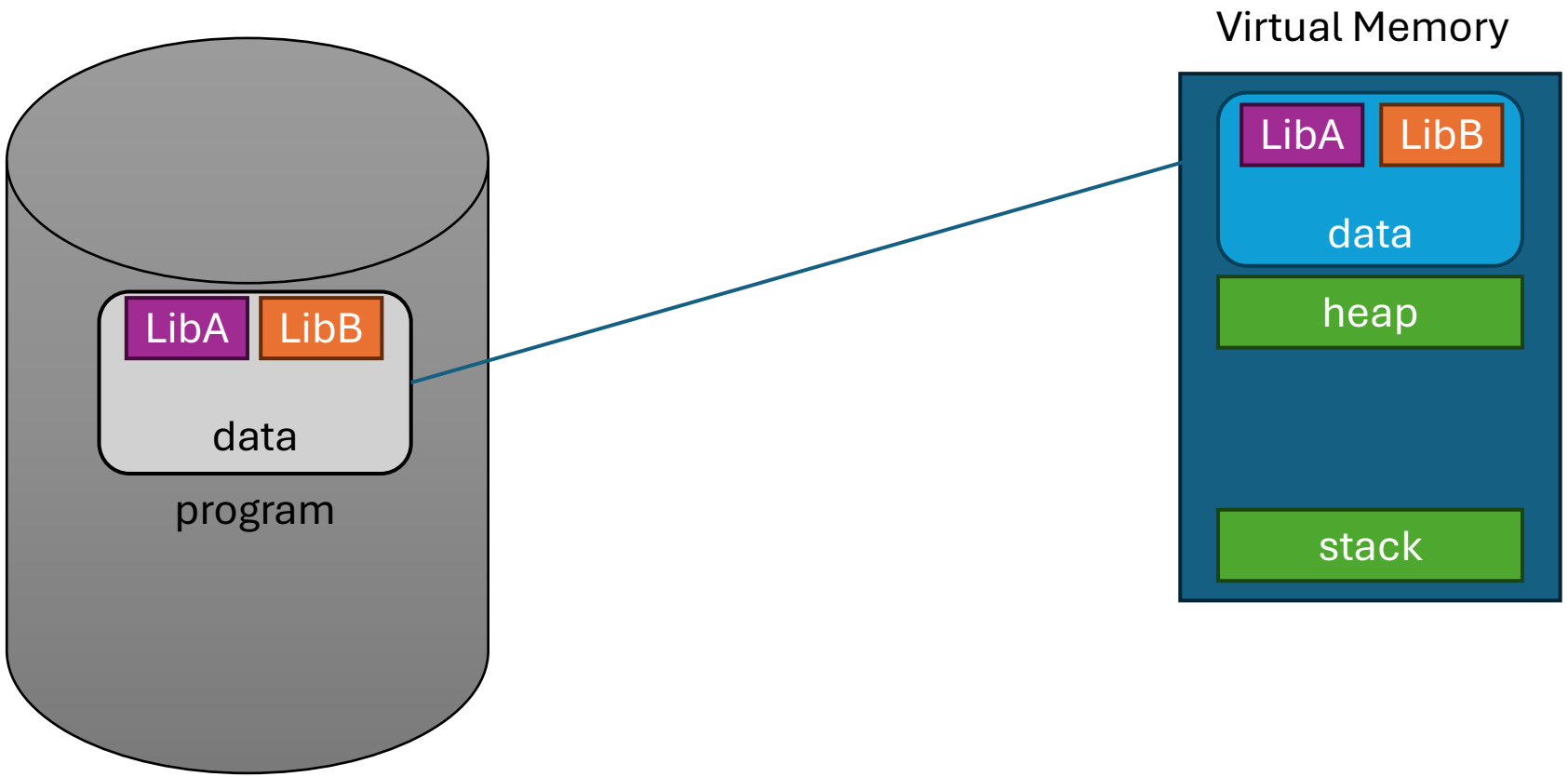
Libraries & Memory

Eliminating redundancy

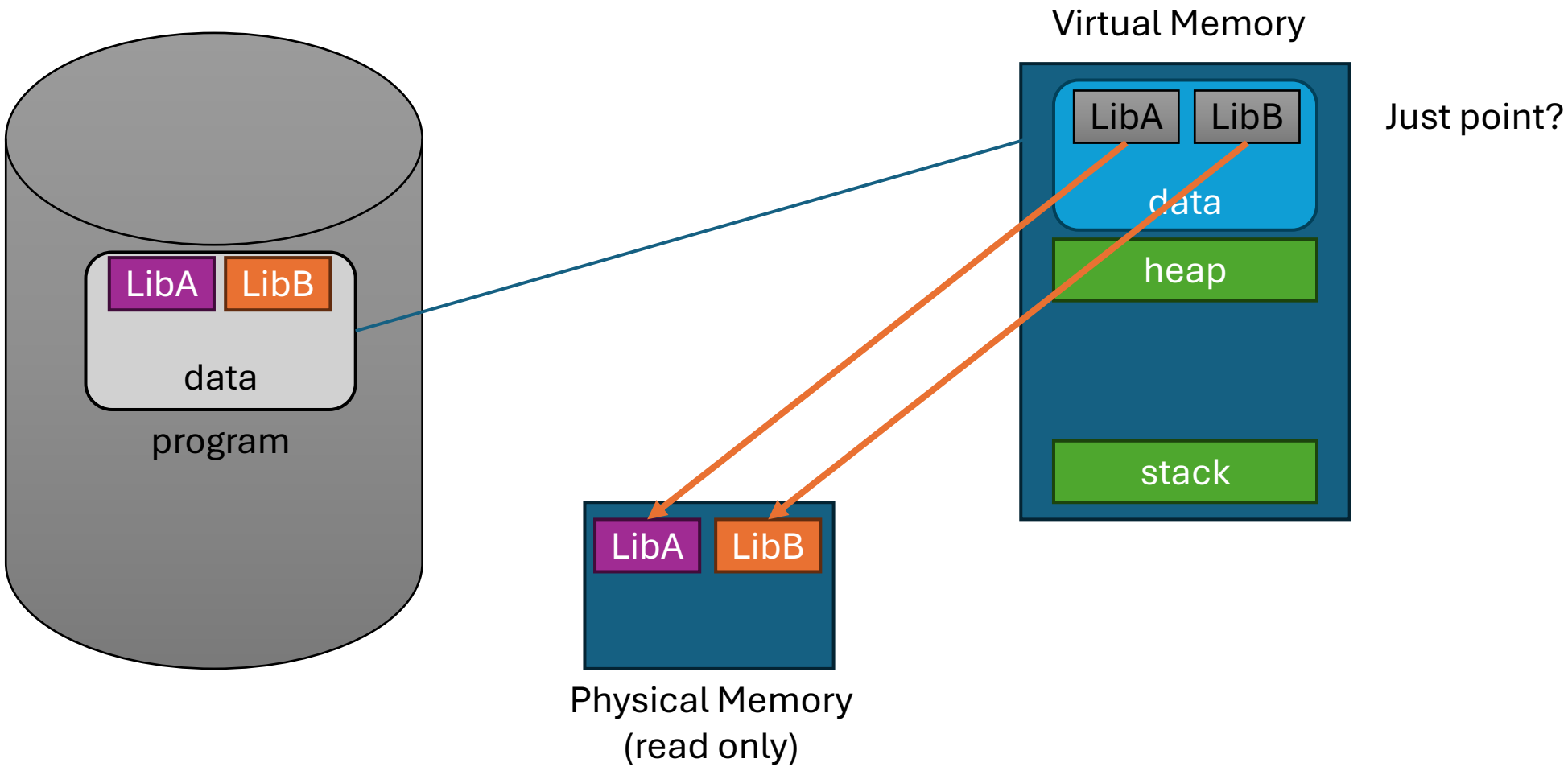
Basic Program



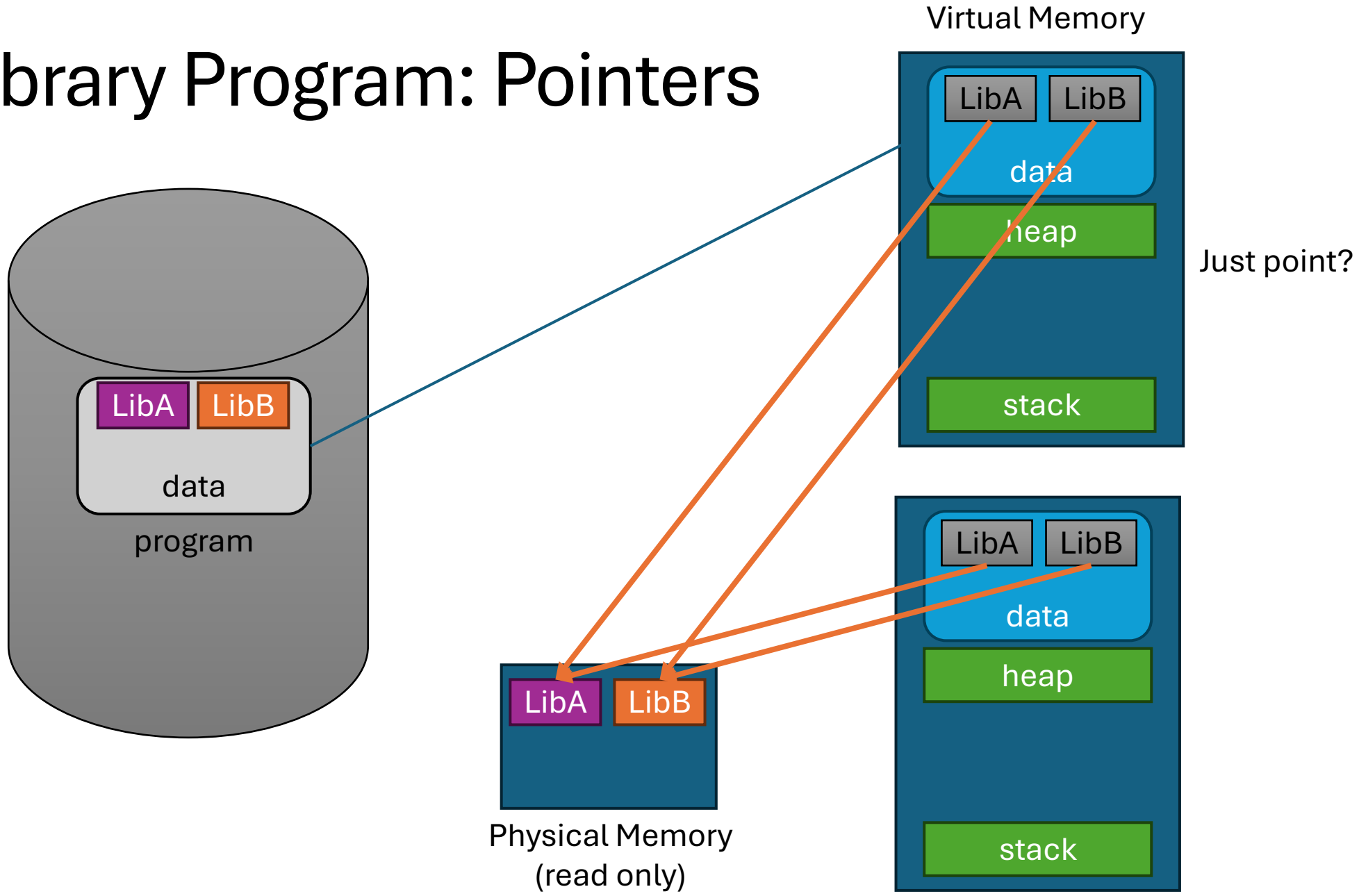
Library Program



Library Program: Pointers



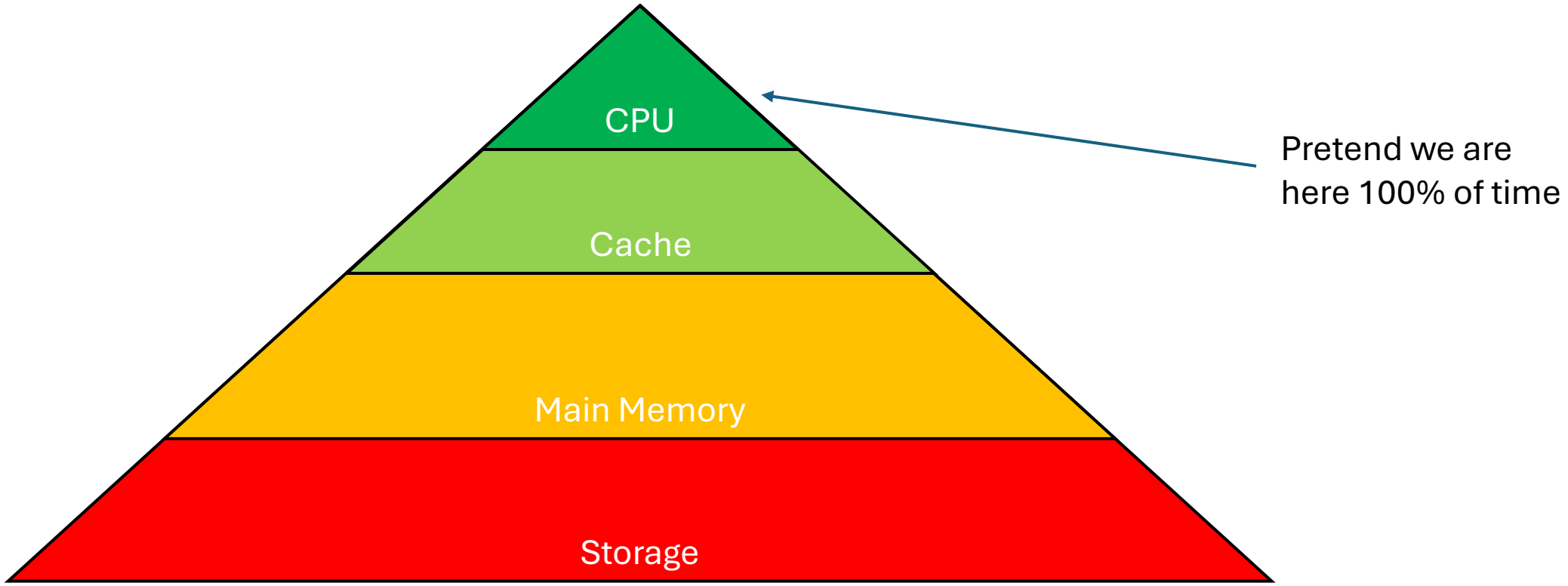
Library Program: Pointers



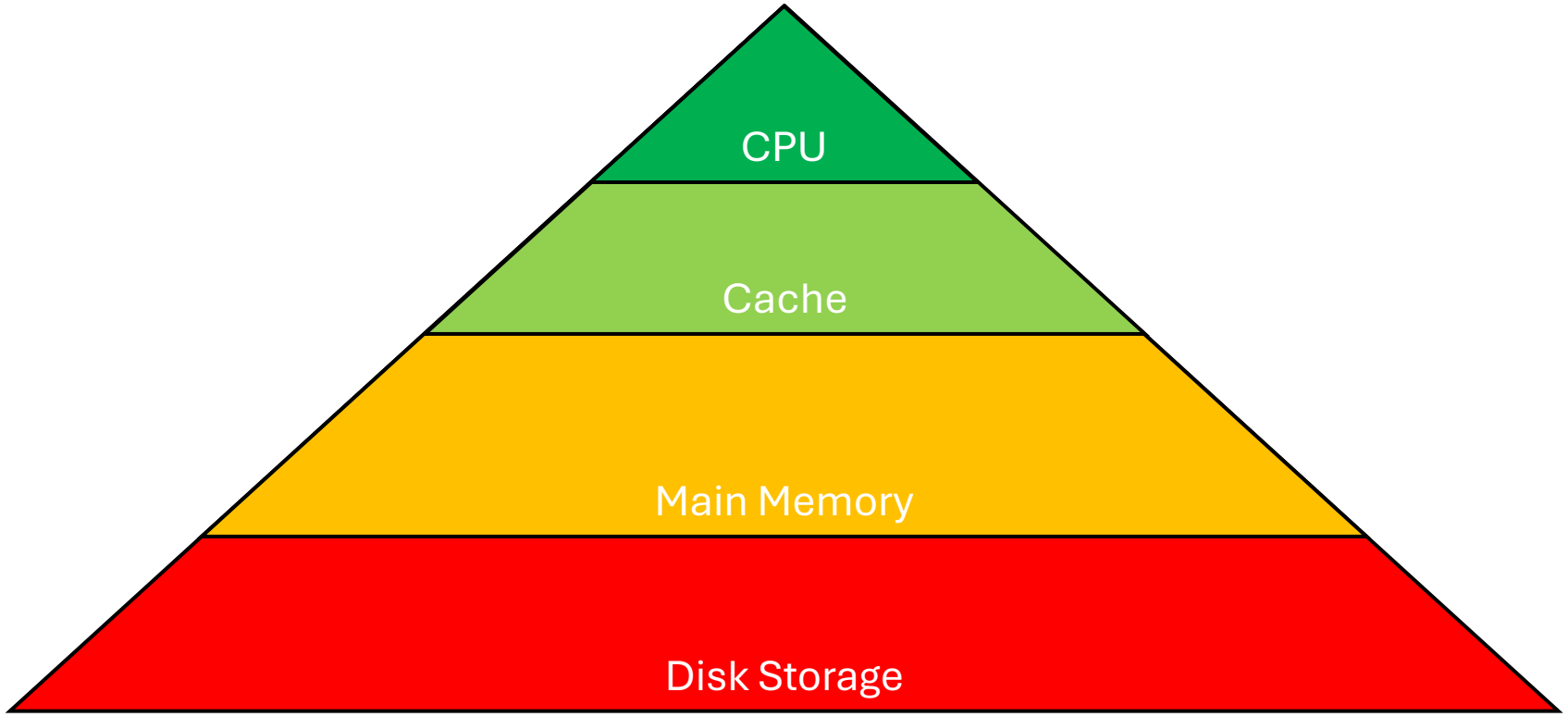
Extra Flags

Some thoughts about memory and bits

How to think about of memory…



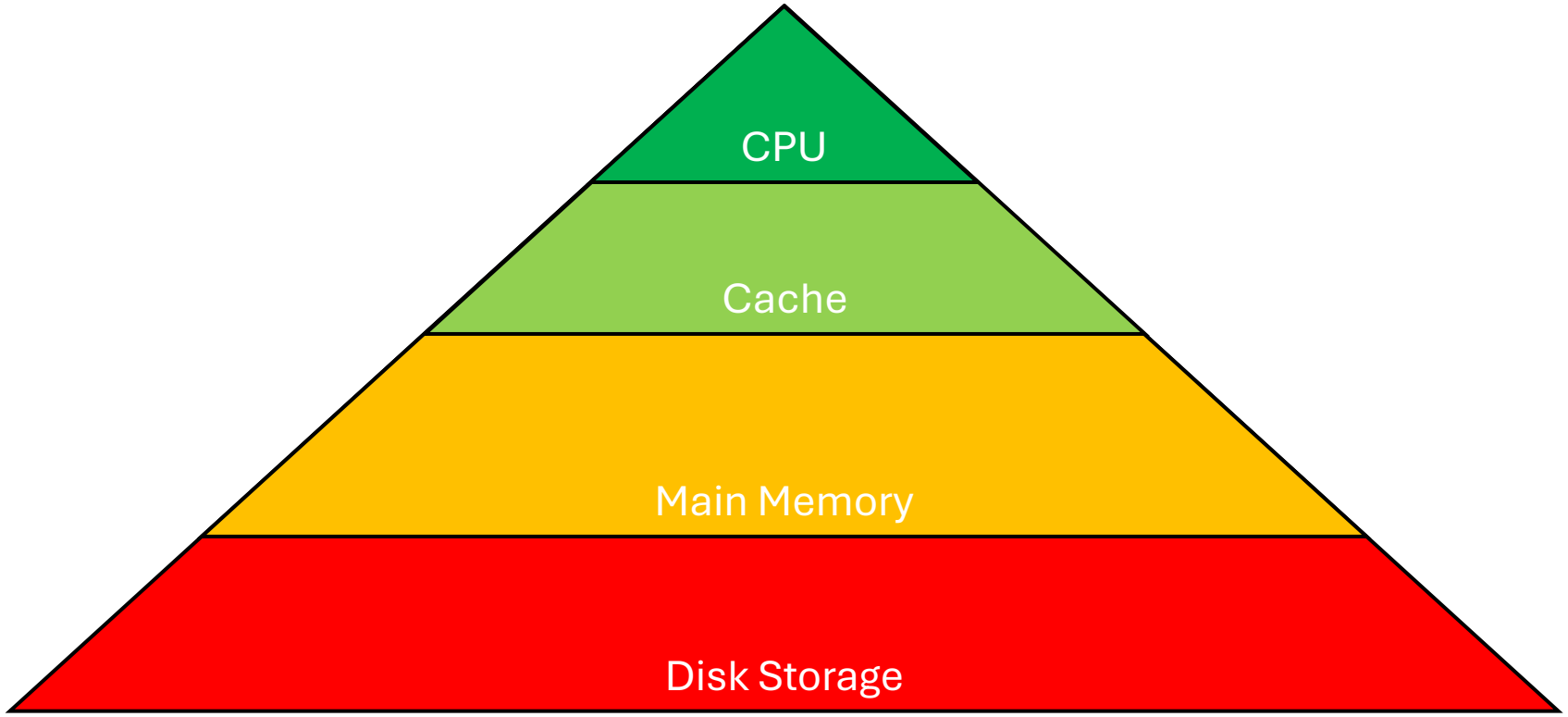
How to think about of memory…



Present = 1

Present = 0

How to think about of memory…

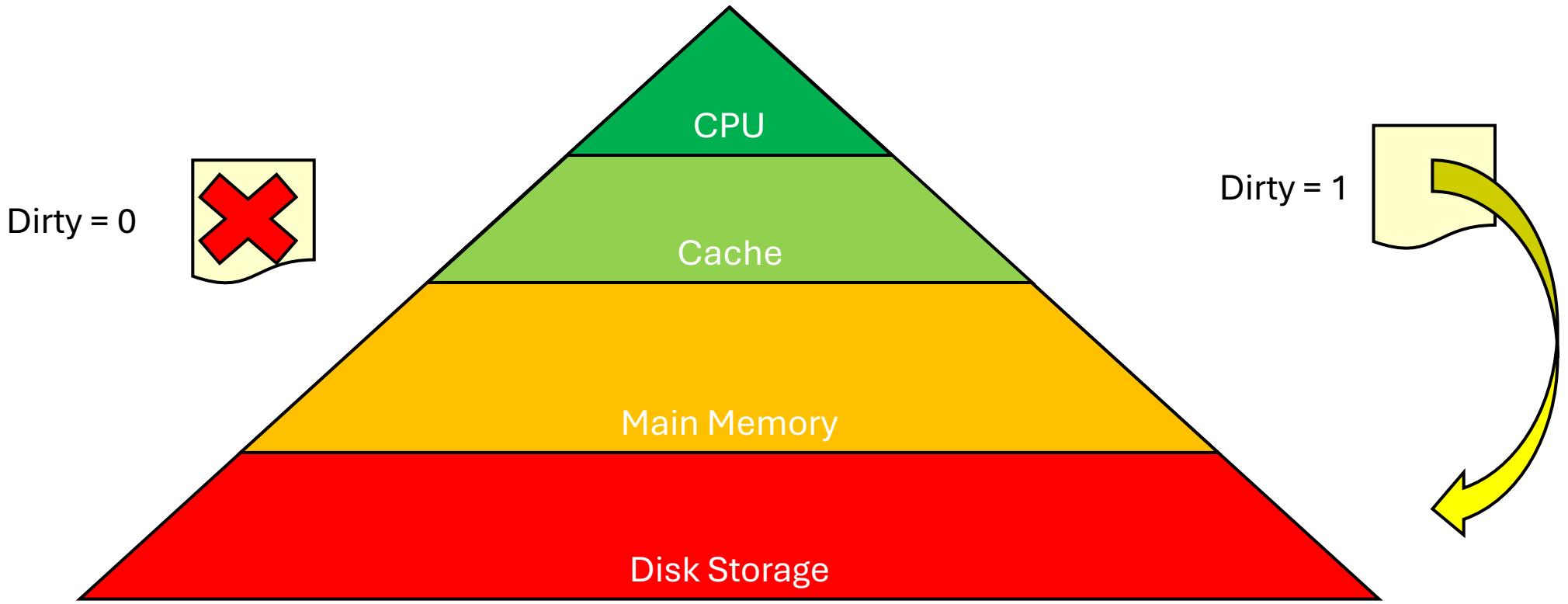


Valid = 1

Valid = 1

Valid = 0

How to think about of memory...



Page Faults

What actually happens

Demand Paging

What is it?

- Only load a page into memory if you need it?

Pros:

- Faster start-up (no need to load whole program on startup)
- Saves memory (only have ‘used’ pages)

Cons:

- Each new page access is slow…

Page Faults

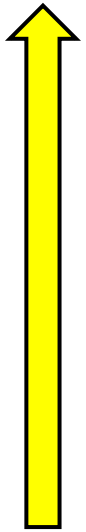
What is it?

- The CPU detects a fault when:
  - TLB lookup misses (i.e., the page isn’t cached)
  - The page table entry is:
    - Present = 0 (i.e., not in physical memory currently)
    - Protection violation (writing to a read-only section, using a kernel page etc)
    - Other fault flags

Page Faults

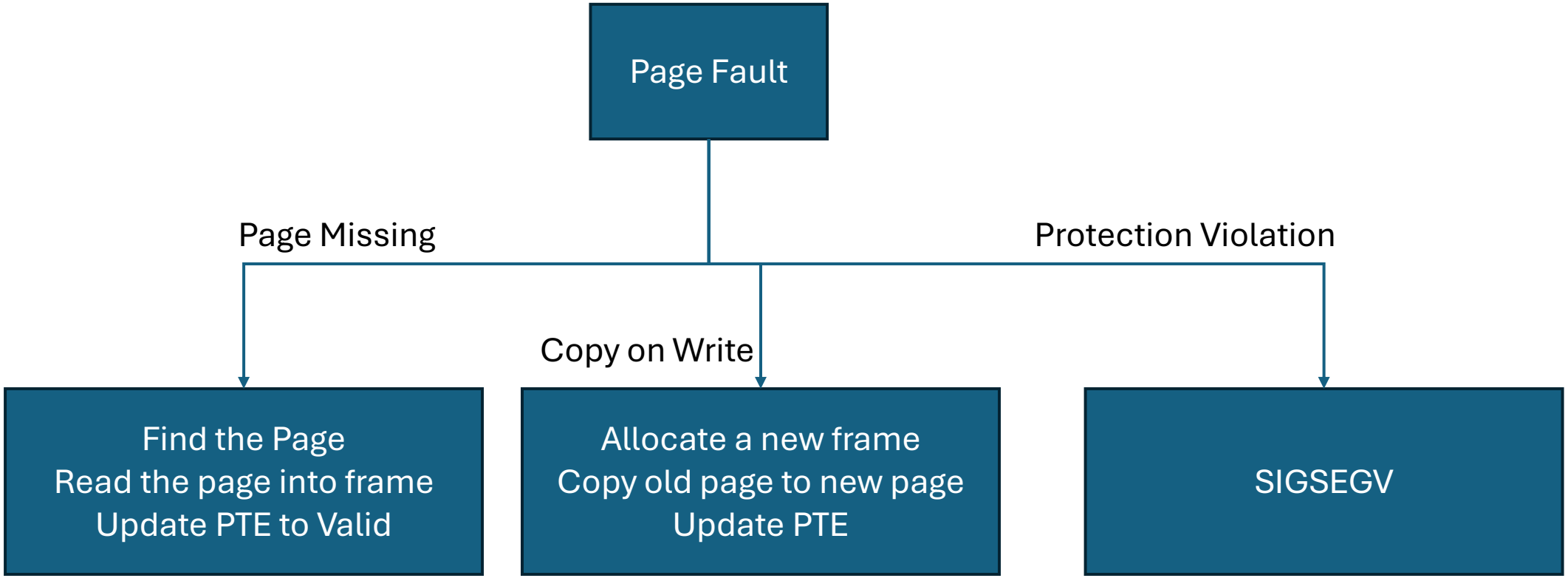
What happens?

- Need to save the faulting instruction's state.
- Run a page-fault handler (OS)

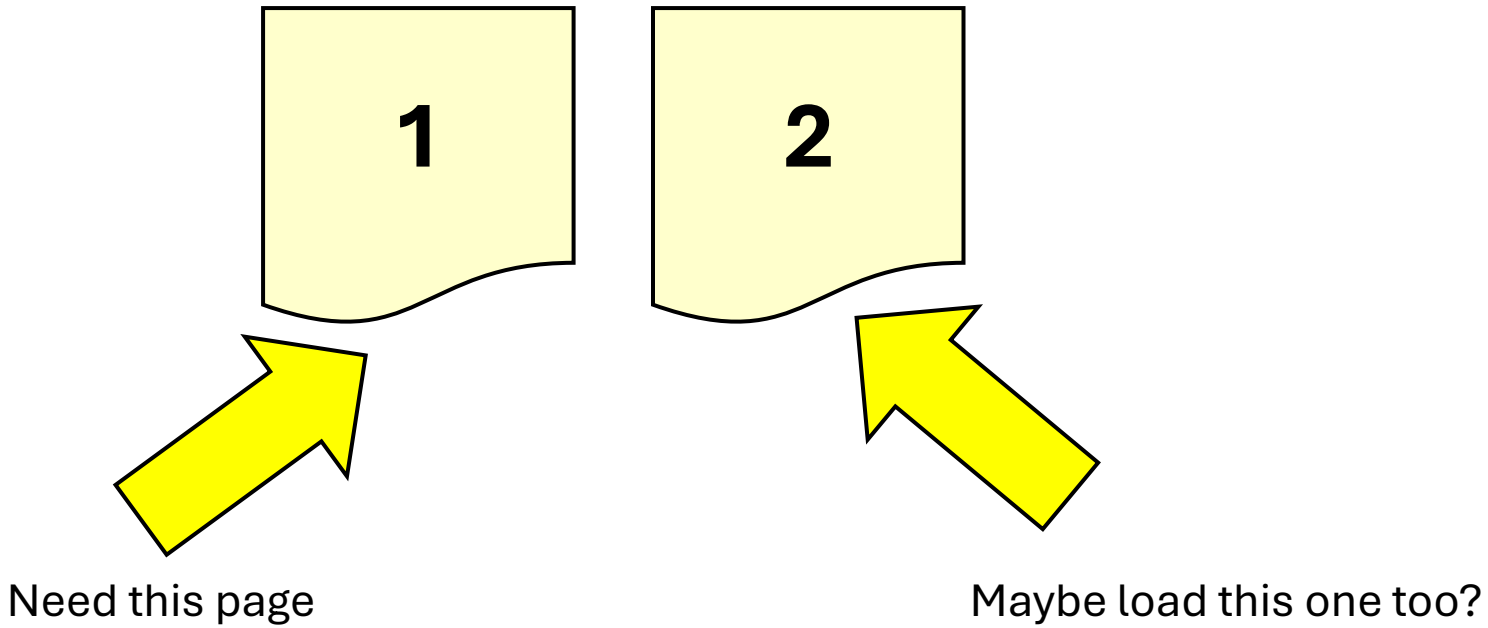


Restoring this is actually non-trivial

Page Faults


Page Fault
Page Missing
Copy on Write
Protection Violation
Find the Page
Read the page into frame
Update PTE to Valid
Allocate a new frame
Copy old page to new page
Update PTE
SIGSEGV

Pre-paging



Replacement

Who to keep? Who to lose?

TLB Swapping Strategies

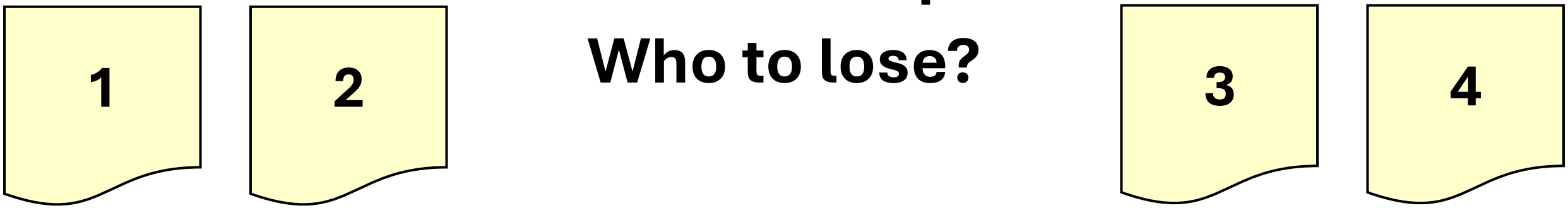
Activity:

You are managing the TLB cache with only 4 spots. You get regular requests from different parts of memory.

What is the strategy?

Who to keep?

Who to lose?



TLB Replacement Policies: Optimal

Option #1:

- Be psychic, eject the one you won’t need for the longest

Pros:

- Perfect

Cons:

- Being psychic is difficult

TLB Replacement Policies: Optimal

Option #1:

- Be psychic, eject the one you won't need for the longest

Why?

- We can benchmark other page replacement algorithms…

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

-															
-															
-															
-															

TLB

TLB Replacement Policies: Optimal

Example

The start is easy

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

1															
-															
-															
-															

TLB

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

1	1														
-	5														
-	-														
-	-														

TLB

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

1	1	1													
-	5	5													
-	-	4													
-	-	-													

TLB

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

1	1	1	1												
-	5	5	5												
-	-	4	4												
-	-	-	3												

TLB

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

Now we have our first problem

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

Not 1

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

Not 2

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

Not 4

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

Not 3

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

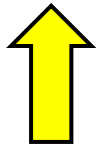
2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	



Kill 5

1	1	1	1	1											
-	5	5	5	5											
-	-	4	4	4											
-	-	-	3	3											

TLB

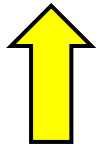
2

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	



Kill 5

1	1	1	1	1											
-	5	5	5	**2**											
-	-	4	4	4											
-	-	-	3	3											

TLB

TLB Replacement Policies: Optimal

Example

Which to victim to kill?

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
-	5	5	5	2	2	2	2	2	2	2	2	2	2	2	
-	-	4	4	4	4	4	4	4	4	4	4	4	4	4	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	3	

TLB 5

At this point, it doesn't matter any more

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

misses					hits									miss	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	5	
-	5	5	5	2	2	2	2	2	2	2	2	2	2	2	
-	-	4	4	4	4	4	4	4	4	4	4	4	4	4	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	3	

TLB 5

TLB Replacement Policies: Optimal

Example

1	5	4	3	2	1	2	4	3	1	3	4	2	1	5	

misses					hits									miss	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	5	
-	5	5	5	2	2	2	2	2	2	2	2	2	2	2	
-	-	4	4	4	4	4	4	4	4	4	4	4	4	4	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	3	
TLB														5	

$$\frac{6\ misses}{15\ accesses} = 0.4$$

TLB Replacement Policies: LRU

Option #2:

- Eject the least recently used page!

Pros:

- Simple idea?

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

-															
-															
-															
-															

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1											
-	5	5	5	5											
-	-	-	3	3											
-	-	-	-	2											

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1										
-	5	5	5	5	5										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB 4

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

Not 2

1	1	1	1	1	1										
-	5	5	5	5	5										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB 4

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

Not 3

1	1	1	1	1	1										
-	5	5	5	5	5										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB 4

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

Not 1

1	1	1	1	1	1										
-	5	5	5	5	5										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB 4

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1										
-	5	5	5	5	4										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB

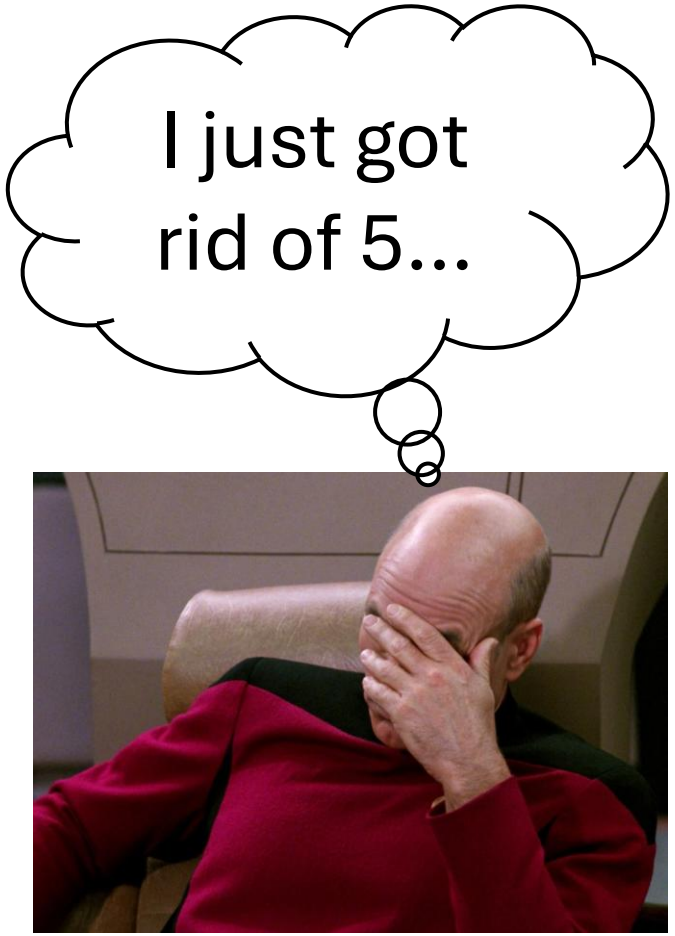
TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1										
-	5	5	5	5	4										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB



TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5									
-	5	5	5	5	4	4									
-	-	-	3	3	3	3									
-	-	-	-	2	2	2									

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5								
-	5	5	5	5	4	4	4								
-	-	-	3	3	3	3	3								
-	-	-	-	2	2	2	2								

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5							
-	5	5	5	5	4	4	4	4							
-	-	-	3	3	3	3	3	3							
-	-	-	-	2	2	2	2	2							

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5							
-	5	5	5	5	4	4	4	4							
-	-	-	3	3	3	3	3	3							
-	-	-	-	2	2	2	2	2							

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5						
-	5	5	5	5	4	4	4	4	4						
-	-	-	3	3	3	3	3	3	3						
-	-	-	-	2	2	2	2	2	1						

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5	5					
-	5	5	5	5	4	4	4	4	4	4					
-	-	-	3	3	3	3	3	3	3	3					
-	-	-	-	2	2	2	2	2	1	1					

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5	5	5				
-	5	5	5	5	4	4	4	4	4	4	4				
-	-	-	3	3	3	3	3	3	3	3	3				
-	-	-	-	2	2	2	2	2	1	1	1				

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5	5	5				
-	5	5	5	5	4	4	4	4	4	4	4				
-	-	-	3	3	3	3	3	3	3	3	3				
-	-	-	-	2	2	2	2	2	1	1	1				

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5	5	5	2			
-	5	5	5	5	4	4	4	4	4	4	4	4			
-	-	-	3	3	3	3	3	3	3	3	3	3			
-	-	-	-	2	2	2	2	2	1	1	1	1			

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5	5	5	2	2		
-	5	5	5	5	4	4	4	4	4	4	4	4	4		
-	-	-	3	3	3	3	3	3	3	3	3	3	3		
-	-	-	-	2	2	2	2	2	1	1	1	1	1		

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	1	5	5	5	5	5	5	2	2	2	
-	5	5	5	5	4	4	4	4	4	4	4	4	4	4	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	5	
-	-	-	-	2	2	2	2	2	1	1	1	1	1	1	

TLB

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

hits

misses

1	1	1	1	1	1	5	5	5	5	5	5	2	2	2	
-	5	5	5	5	4	4	4	4	4	4	4	4	4	4	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	5	
-	-	-	-	2	2	2	2	2	1	1	1	1	1	1	

TLB

$$\frac{9\ misses}{15\ accesses} = 0.6$$

TLB Replacement Policies: LRU

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

Optimal

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
-	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
-	-	-	3	3	3	3	3	3	3	3	3	2	2	2	
-	-	-	-	2	4	4	4	4	4	4	4	4	4	4	

$$\frac{5\ misses}{15\ accesses} = 0.33$$

LRU

1	1	1	1	1	1	5	5	5	5	5	5	2	2	2	
-	5	5	5	5	4	4	4	4	4	4	4	4	4	4	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	5	
-	-	-	-	2	2	2	2	2	1	1	1	1	1	1	

$$\frac{9\ misses}{15\ accesses} = 0.6$$

TLB

TLB Replacement Policies: LRU

Option #2:

- Eject the least recently used page!

Complications:

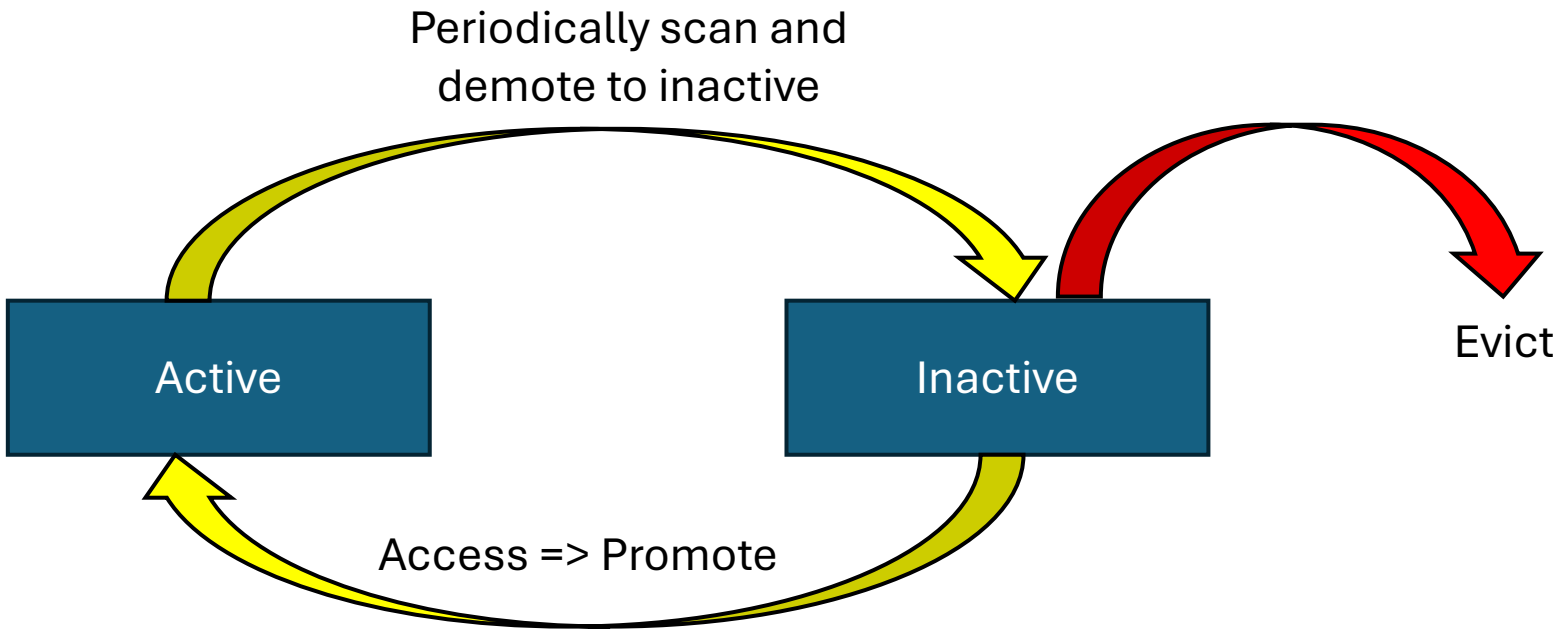
- Calculating the Least Recently Used
  - Use a timestamp? Very slow
  - Use an ‘access bit’, occasionally reset it Approximate?

TLB Replacement Policies: LRU

Option #2:

- Eject the least recently used page!

Two-Hand Clock



TLB Replacement Policies: FIFO

Option #3:

- Eject on FIFO order

Pros:

- Simple idea?

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1															
-															
-															
-															

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1														
-	5														
-	-														
-	-														

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1													
-	5	5													
-	-	-													
-	-	-													

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1												
-	5	5	5												
-	-	-	3												
-	-	-	-												

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1											
-	5	5	5	5											
-	-	-	3	3											
-	-	-	-	2											

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4										
-	5	5	5	5	5										
-	-	-	3	3	3										
-	-	-	-	2	2										

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4									
-	5	5	5	5	5	5									
-	-	-	3	3	3	3									
-	-	-	-	2	2	2									

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4								
-	5	5	5	5	5	5	5								
-	-	-	3	3	3	3	3								
-	-	-	-	2	2	2	2								

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4							
-	5	5	5	5	5	5	5	5							
-	-	-	3	3	3	3	3	3							
-	-	-	-	2	2	2	2	2							

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4							
-	5	5	5	5	5	5	5	5							
-	-	-	3	3	3	3	3	3							
-	-	-	-	2	2	2	2	2							

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4						
-	5	5	5	5	5	5	5	5	1						
-	-	-	3	3	3	3	3	3	3						
-	-	-	-	2	2	2	2	2	2						

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4	4					
-	5	5	5	5	5	5	5	5	1	1					
-	-	-	3	3	3	3	3	3	3	3					
-	-	-	-	2	2	2	2	2	2	2					

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4	4	4				
-	5	5	5	5	5	5	5	5	1	1	1				
-	-	-	3	3	3	3	3	3	3	3	3				
-	-	-	-	2	2	2	2	2	2	2	2				

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4	4	4	4			
-	5	5	5	5	5	5	5	5	1	1	1	1			
-	-	-	3	3	3	3	3	3	3	3	3	3			
-	-	-	-	2	2	2	2	2	2	2	2	2			

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4	4	4	4	4		
-	5	5	5	5	5	5	5	5	1	1	1	1	1		
-	-	-	3	3	3	3	3	3	3	3	3	3	3		
-	-	-	-	2	2	2	2	2	2	2	2	2	2		

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4	4	4	4	4		
-	5	5	5	5	5	5	5	5	1	1	1	1	1		
-	-	-	3	3	3	3	3	3	3	3	3	3	3		
-	-	-	-	2	2	2	2	2	2	2	2	2	2		

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	
-	5	5	5	5	5	5	5	5	1	1	1	1	1	1	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	5	
-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	

TLB

TLB Replacement Policies: FIFO

Example

1	5	1	3	2	4	5	4	3	1	3	4	2	1	5	

hits

misses

1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	
-	5	5	5	5	5	5	5	5	1	1	1	1	1	1	
-	-	-	3	3	3	3	3	3	3	3	3	3	3	5	
-	-	-	-	2	2	2	2	2	2	2	2	2	2	2	

TLB

$$\frac{6\ misses}{15\ accesses} = 0.4$$

TLB Replacement Policies: FIFO

Option #3:

- Eject on FIFO order

Complications:

- Needs more memory
- Can cause… more page faults with more TLB entries?
- Poor match for data usage

TLB Replacement Policies: FIFO

Belady's Anomaly

3	2	1	5	3	2	4	3	2	1	5	4				

3	3	3	5	5	5	4	4	4	4	4	4				
	2	2	2	3	3	3	3	3	1	1	1				
		1	1	1	2	2	2	2	2	5	5				

$$\frac{9\ misses}{12\ accesses} = 0.75$$

3	3	3	3	3	3	4	4	4	4	5	5				
	2	2	2	2	2	2	3	3	3	3	4				
		1	1	1	1	1	1	2	2	2	2				
			5	5	5	5	5	5	1	1	1				

$$\frac{10\ misses}{12\ accesses} = 0.83$$

TLB Replacement Policies: Random

Option #4:

- Random!!

Pros:

- Simple
- Hard to ‘game’

TLB Replacement Policies

Optimal

- Optimal & Impossible

LRU

- Approximatable
- Pretty good

Random

- Cheap and OK

FIFO

- Easy
- Can be really bad

TLB Replacement Policies

Further ‘Improvements’

- Frequency of use?
  - Need to remember everyone
- LRU-2
  - The second use… (i.e. twice)
- Track multiple victims?
- Evict non-**dirty** pages first (for free)

- Writeback List
  - Keep a list, if you are writing to disk, take from the list instead
- Demand Zero List
  - Keep a short list of empty pages to avoid security problems

Page Replacement Design

- Fixed Space Algorithms
  - Processes can only evict their own pages
  - Works for some proceses
- Variable Space Algorithms
  - Processes can grown and shrink their number of pages
  - One process can ruin it for everyone
  - Clock is one such algorithm

Extras

Some extra things

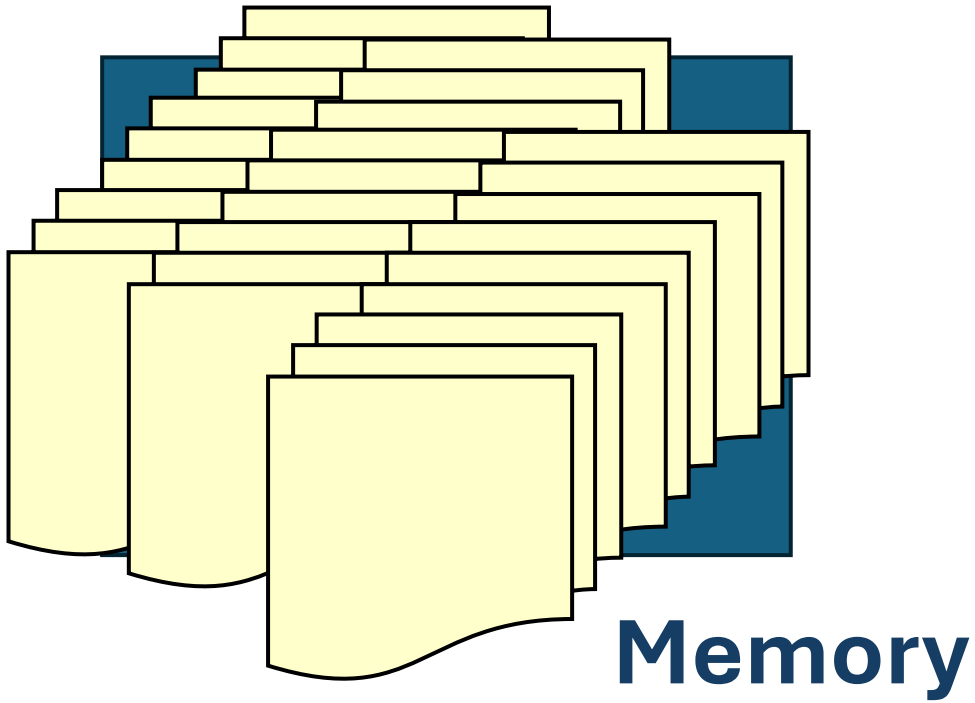
Thrashing

The Problem

- Too many processes demanding too much memory

The Symptoms

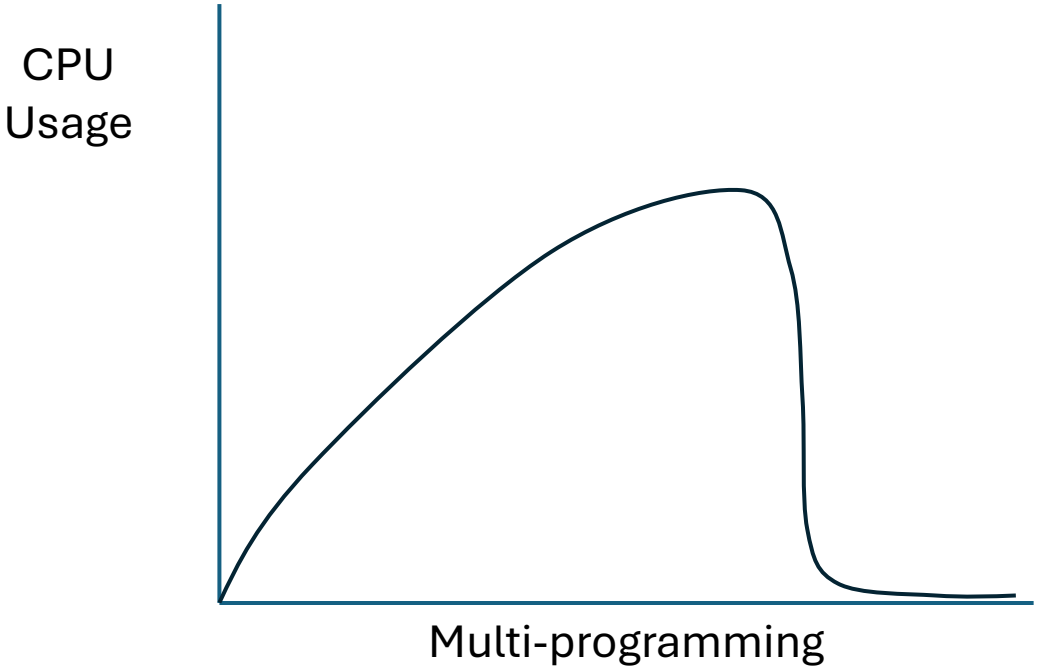
- Low user CPU usage (<10%) constant faulting
- High CPU usage (>70%) system servicing faults

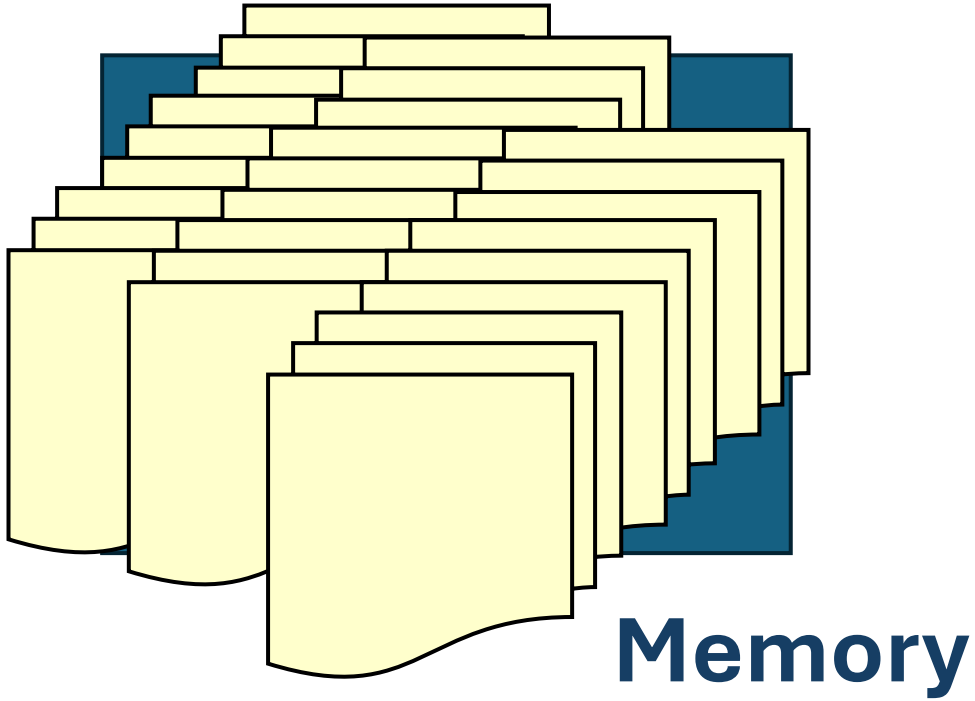


Thrashing

The Solution

- Stop running some processes?





Page Fault Exceptions

Page Fault Exceptions

- Not Mapped
- Not Resident
- Access Violation (r/w/e)

Exception handled by OS

- Performed in Kernel Mode

User Exception Handling

- User can register to perform handling of some exceptions
  - Not all exceptions are allowed
  - OS dependent

Guard Pages

```
int main()
{
    int a[];
    int b = a[0];
}
```

NULL…

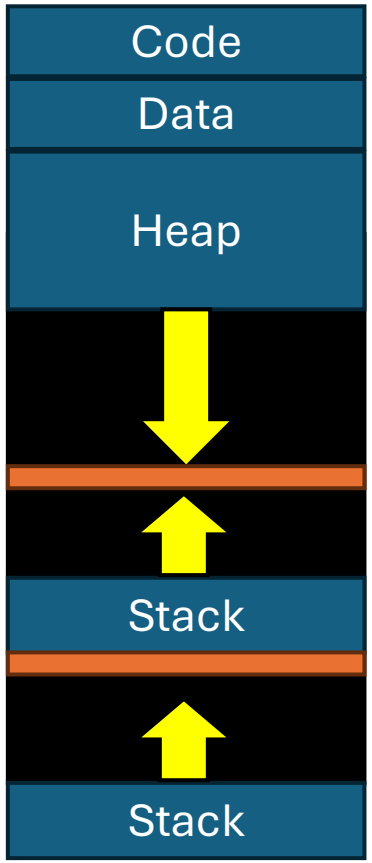
Zeroth page is often protected to simplify null references.

- Can be subverted with a big enough offset… i.e. a[4096]

Guard Pages: Threads

Can use **guard** pages to catch wayward stack growth.

- Doesn't require actual memory, just a guard page table entry



Watchpoints

Problem:

- Debugging a memory problem

Debugger:

- Set protection on the page the variable is on… triggers an exception (can find the address)

File Mapping

Point Paging for a region of memory at a regular file:

- mmap call

The Idea:

- You map a file into RAM as a page.
- Your program accesses the file like it is an array.

Why?

- Fast (only pages touched get loaded)
- Shared mapping can happen
- Memory efficient

Snapshots and Checkpoints

Idea

- Use memory mapping to a file with the contents of a process's memory creating a 'snapshot' of a process
- Requires additional support, but can potentially be used to 'restart' a process mid-way.

Why?

- Long lived computations?
- Sleeping processes?
- Persistent Programming?

Orthogonal Persistence

Idea:

- What if you want to just… keep stuff around (advanced programming languages).
  - I.e., heap objects don’t disappear at process completion

Why?

- Lazy Loading (huge datasets)
- Crash-resilience

Distributed Shared Memory

Idea:

- What if you had multiple machines which pretend to be one big single machine?

Why?

- More optimisations

Thoughts:

- Access (from someone else)
- Copy-on-Write
- Broadcasting ‘dirty’ status

Summary

Memory Swapping:

- Translation Lookaside Buffer
  - Page sizes? TLB reach?
  - Hardware/OS implementation?
- Paging
  - Inverted Tables
  - Segmented Page Tables
  - Multi-level Page Tables
- Page Replacement Algorithms

Questions?