# CSM Conceptual Final Review

Raghav, Robin, Connor

Feedback Form

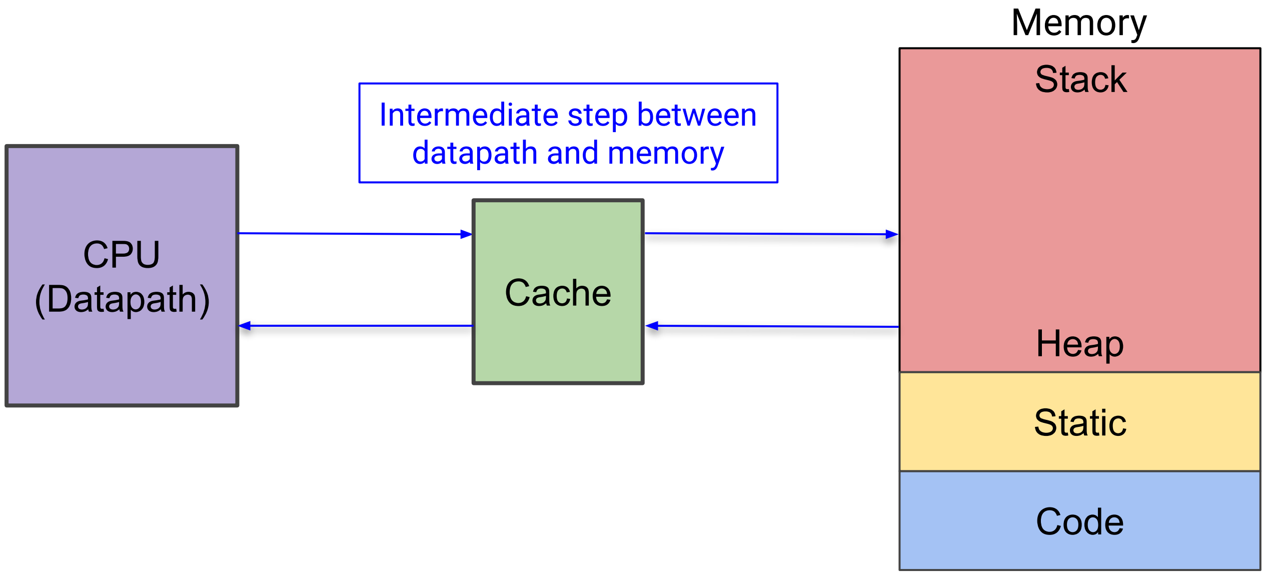
# Caches: Overview

**Temporal Locality**
If a memory location is accessed then it is likely to be accessed again

**Spatial Locality**
If a memory location is accessed then memory locations nearly are likely to be accessed too!

**Example:**

```
total = 0
for i in range(len(arr)):
    total = total + arr[i]
```



## Caches: Calculations (important formulae)

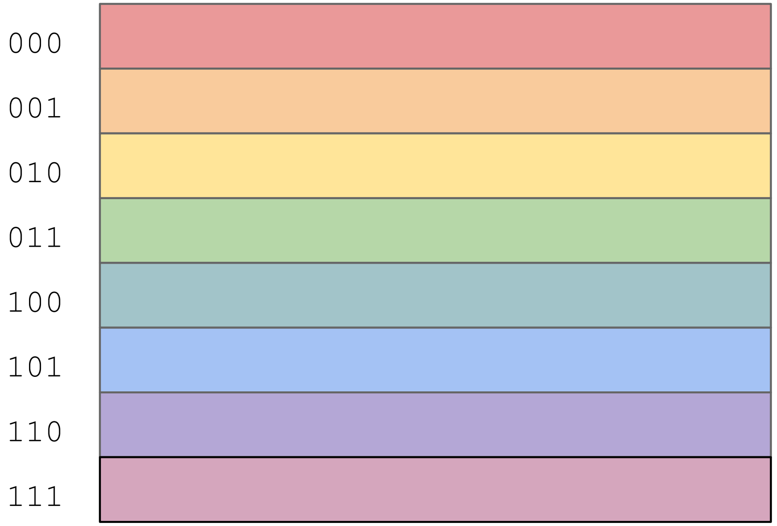
Usually this information is given

- Address Space (A) - in bits
- Cache Size (C) - in bytes
  - How large the cache is
- Block Size (B) - in bytes
  - Unit for caches, 1 Block = 1 entry in $$
- Associativity (N)
  - How many entries per index

Address Breakdown Calculations

- Offset (O) = $\log_2 B$
  - Lower bits to choose specific bytes in a block
- Index (I) = $\log_2 (C / B / N)$
  - Bits denotes the section the block resides in
- Tag (T) = A - I - O
  - Identifier for a block (unique id)

# Caches: Direct Mapped Caches



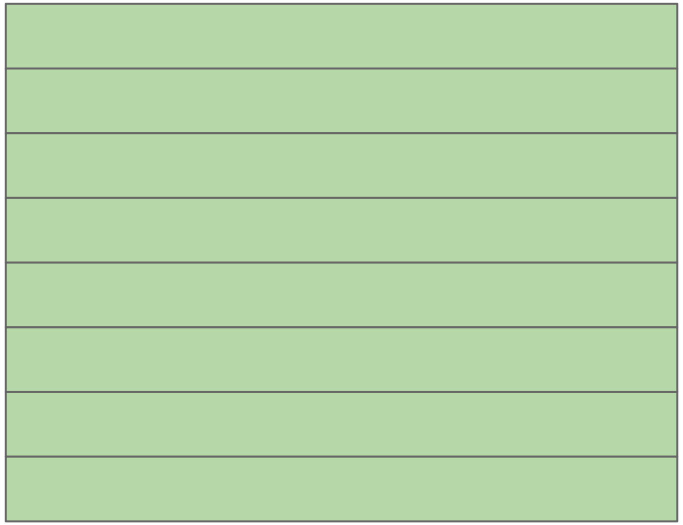
**Cache Properties**
Address Space (A) = 8 bit address space
Cache Size (C) = 32 B
Block Size (B) = 4 B
Associativity (N) = 1

**Cache Address Breakup**
Offset (O) = $\log_2 B$ = 2 bit
Index (I) = $\log_2 C / B / N$ = 3 bit
Tag (T) = 3 bit

0b 0000 1111

(Note: Walk through the example)

# Caches: Fully Associative Caches



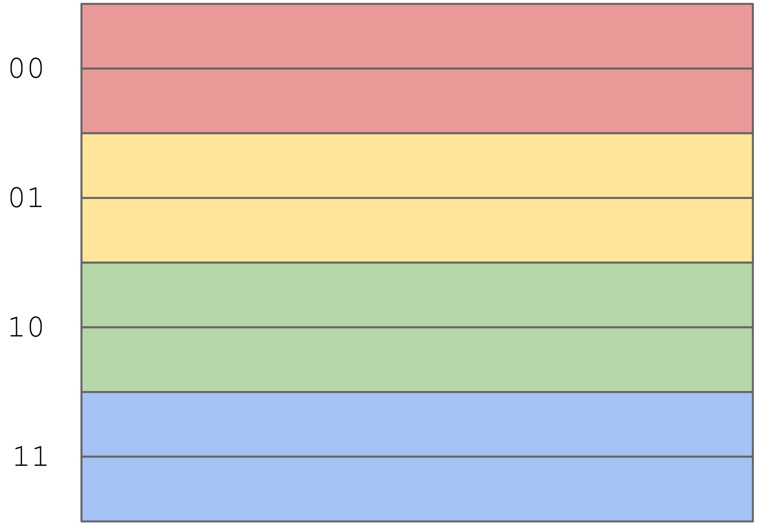
**Cache Properties**
Address Space (A) = 8 bit address space
Cache Size (C) = 32 B
Block Size (B) = 4 B
Associativity (N) = 8

**Cache Address Breakup**
Offset (O) = $\log_2 B$ = 2 bit
Index (I) = $\log_2 C / B / N$ = 0 bit
Tag (T) = 6 bit

0b 0000 1111

(Note: Explain why 0 index bits)

# Caches: N-Way Associative



**Cache Properties**
Address Space (A) = 8 bit address space
Cache Size (C) = 32 B
Block Size (B) = 4 B
Associativity (N) = 2

**Cache Address Breakup**
Offset (O) = $\log_2 B$ = 2 bit
Index (I) = $\log_2 C / B / N$ = 2 bit
Tag (T) = 4 bit

0b 0000 11 11

(Note: Explain why tag is specially important here)

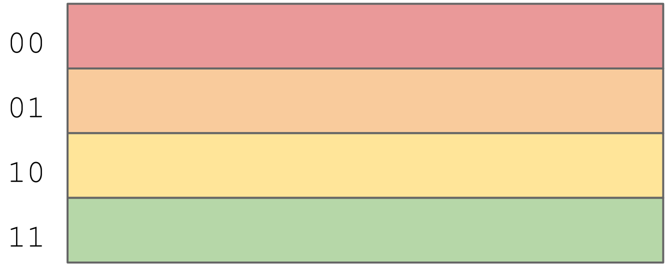
# Caches: Misses (3 C's)

**Compulsory Miss**
If data is accessed for the first time, since it is not in the cache
Can be avoided for subsequent accesses with spatial locality

**Conflict Miss**
If several memory locations map to the same cache block, then a conflict miss can occur

**Capacity Miss**
If the program is working with data much larger than the cache size then a capacity miss occurs

There is a tradeoff between increasing cache and block size with speed, miss rate and miss penalty



# Caches: Walkthrough

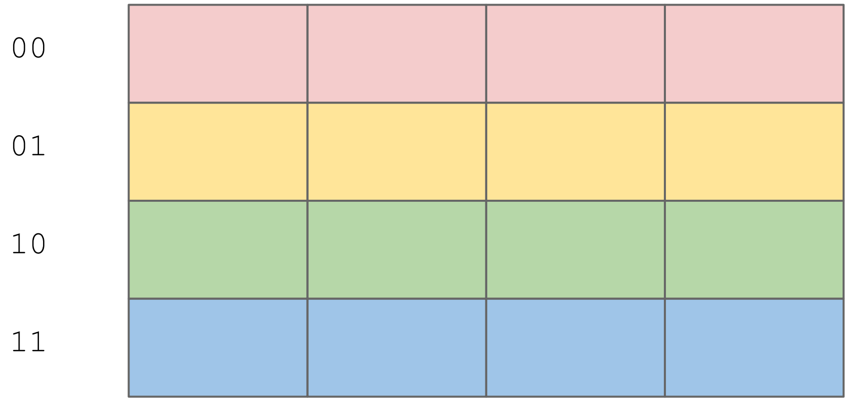
**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**What type of miss will occur?**



# Caches: Walkthrough

## Code

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

## Cache Properties

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

## 1st Iteration of Loop

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | | | | |
| 10 | | | | |
| 11 | | | | |

# Caches: Walkthrough

## Code

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

## Cache Properties

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

## 5th Iteration of Loop

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | arr[4] | arr[5] | arr[6] | arr[7] |
| 10 | | | | |
| 11 | | | | |

# Caches: Walkthrough

## Code

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

## Cache Properties

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

## 9th Iteration of Loop

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | arr[4] | arr[5] | arr[6] | arr[7] |
| 10 | arr[8] | arr[9] | arr[10] | arr[11] |
| 11 | | | | |

# Caches: Walkthrough

**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**13th Iteration of Loop**

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | arr[4] | arr[5] | arr[6] | arr[7] |
| 10 | arr[8] | arr[9] | arr[10] | arr[11] |
| 11 | arr[12] | arr[13] | arr[14] | arr[15] |

# Caches: Walkthrough

**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**17th Iteration of Loop**

| | | | | |
|---|---|---|---|---|
| 00 | arr[16] | arr[17] | arr[18] | arr[19] |
| 01 | arr[4] | arr[5] | arr[6] | arr[7] |
| 10 | arr[8] | arr[9] | arr[10] | arr[11] |
| 11 | arr[12] | arr[13] | arr[14] | arr[15] |

# Caches: Walkthrough

## Code

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

## Cache Properties

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

## 21st Iteration of Loop

| | | | | |
|---|---|---|---|---|
| 00 | arr[16] | arr[17] | arr[18] | arr[19] |
| 01 | arr[20] | arr[21] | arr[22] | arr[23] |
| 10 | arr[8] | arr[9] | arr[10] | arr[11] |
| 11 | arr[12] | arr[13] | arr[14] | arr[15] |

# Caches: Walkthrough

### Code

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

### Cache Properties

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

### 25th Iteration of Loop

| | | | | |
|---|---|---|---|---|
| 00 | arr[16] | arr[17] | arr[18] | arr[19] |
| 01 | arr[20] | arr[21] | arr[22] | arr[23] |
| 10 | arr[24] | arr[25] | arr[26] | arr[27] |
| 11 | arr[12] | arr[13] | arr[14] | arr[15] |

# Caches: Walkthrough

**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**29th Iteration of Loop**

| | | | | |
|---|---|---|---|---|
| 00 | arr[16] | arr[17] | arr[18] | arr[19] |
| 01 | arr[20] | arr[21] | arr[22] | arr[23] |
| 10 | arr[24] | arr[25] | arr[26] | arr[27] |
| 11 | arr[28] | arr[29] | arr[30] | arr[31] |

# Caches: Walkthrough

**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**33rd Iteration of Loop - Capacity Miss**

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | arr[20] | arr[21] | arr[22] | arr[23] |
| 10 | arr[24] | arr[25] | arr[26] | arr[27] |
| 11 | arr[28] | arr[29] | arr[30] | arr[31] |

# Caches: Walkthrough

## Code

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

## Cache Properties

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

### 37th Iteration of Loop - Capacity Miss

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | arr[4] | arr[5] | arr[6] | arr[7] |
| 10 | arr[24] | arr[25] | arr[26] | arr[27] |
| 11 | arr[28] | arr[29] | arr[30] | arr[31] |

# Caches: Walkthrough (2)

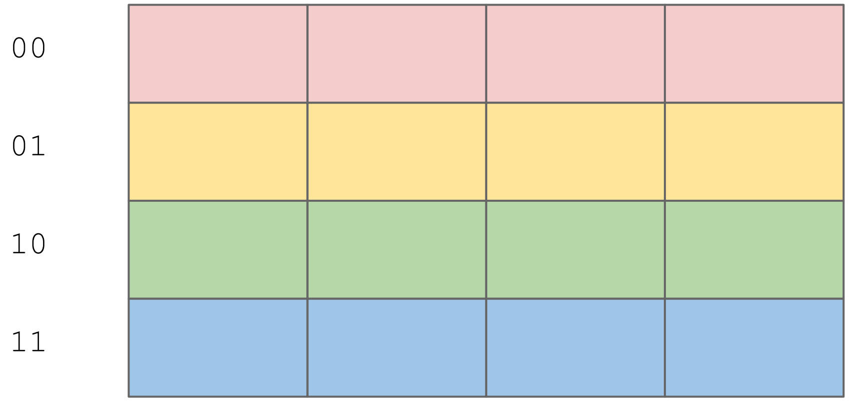
**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing 1st and 17th array elements by 1
arr[0] = arr[0] + 1;
arr[16] = arr[16] + 1;
arr[0] = arr[0] + 1;
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**What type of miss will occur?**



# Caches: Walkthrough (2)

**Code**

```
 3  int[] arr = new int[32];
 4  int i, j;
 5  for (i = 0; i < 32; i++) {
 6    arr[i] = i;
 7  }
 8
 9  // incrementing 1st and 17th array elements by 1
10  arr[0] = arr[0] + 1;
11  arr[16] = arr[16] + 1;
12  arr[0] = arr[0] + 1;
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**Access on Line 10**

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | | | | |
| 10 | | | | |
| 11 | | | | |

# Caches: Walkthrough (2)

**Code**

```
 3  int[] arr = new int[32];
 4  int i, j;
 5  for (i = 0; i < 32; i++) {
 6    arr[i] = i;
 7  }
 8
 9  // incrementing 1st and 17th array elements by 1
10  arr[0] = arr[0] + 1;
11  arr[16] = arr[16] + 1;
12  arr[0] = arr[0] + 1;
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**Access on Line 11**

| | | | | |
|---|---|---|---|---|
| 00 | arr[16] | arr[17] | arr[18] | arr[19] |
| 01 | | | | |
| 10 | | | | |
| 11 | | | | |

# Caches: Walkthrough (2)

**Code**

```
3   int[] arr = new int[32];
4   int i, j;
5   for (i = 0; i < 32; i++) {
6     arr[i] = i;
7   }
8
9   // incrementing 1st and 17th array elements by 1
10  arr[0] = arr[0] + 1;
11  arr[16] = arr[16] + 1;
12  arr[0] = arr[0] + 1;
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**Access on Line 12 - Conflict Miss**

| | | | | |
|---|---|---|---|---|
| 00 | arr[0] | arr[1] | arr[2] | arr[3] |
| 01 | | | | |
| 10 | | | | |
| 11 | | | | |

# Caches: AMAT

AMAT - Average Memory Access Time. Goal is to minimize AMAT by adjusting cache parameters

- Hit time - time to check if data is in the cache
- Miss rate - percentage of accesses of a program that are not in the cache
- Miss penalty - cost incurred by missing in cache (usually going to memory)

Formula

AMAT = hit time + miss rate * miss penalty

Can be recursive if L1, L2, memory (etc)

$\text{AMAT} = \text{HT}_{L1} + \text{MR}_{L1} * \mathbf{MP_{L1}}$

$\mathbf{MP_{L1}} = \text{HT}_{L2} * \text{MR}_{L2} * \text{MP}_{MEM}$

## Caches: AMAT Example

**Code**

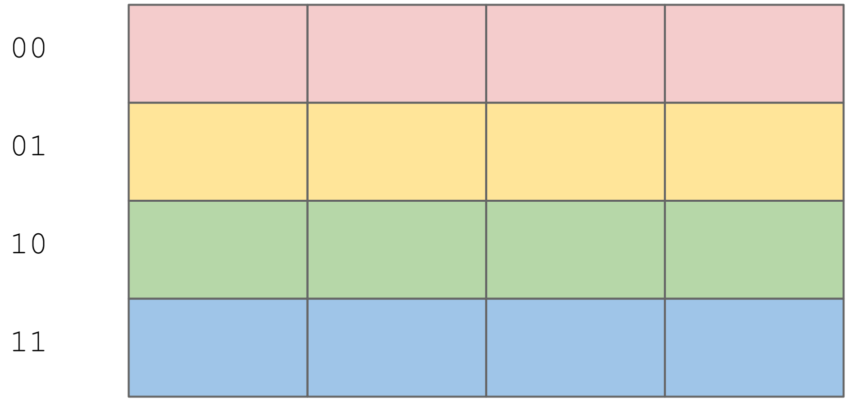
```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**Access Timing**

| | |
|---|---|
| L1 Access Time | = 8 cycles |
| Mem Access Time | = 100 cycles |



**Calculate the AMAT.**

# Caches: AMAT Example

**Code**

```
int[] arr = new int[32];
int i, j;
for (i = 0; i < 32; i++) {
  arr[i] = i;
}

// incrementing each array element by 1
for (j = 0; j < 64; j++) {
  int index = j%32
  arr[index] = arr[index] + 1;
}
```

AMAT = HT + MR * MP

AMAT = 8 + 0.75*100

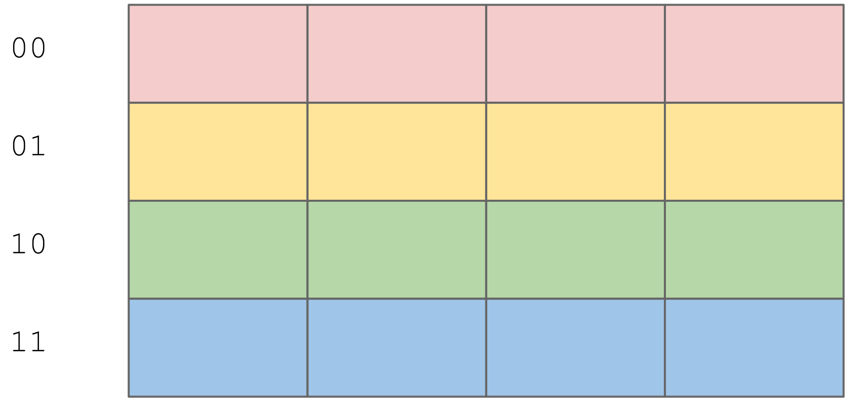
AMAT = 83 cycles

**Cache Properties**

| | |
|---|---|
| Cache Size (C) | = 16B |
| Block Size (B) | = 4B |
| Associativity (N) | = 1 |

**Access Timing**

| | |
|---|---|
| L1 Access Time | = 8 cycles |
| Mem Access Time | = 100 cycles |



# Caches: Block Replacement Policies

### LRU (Least Recently Used)

Cache out block which has been accessed (read or write) least recently

Pros:

- Takes advantage of Temporal Locality because recent past use implies likely future use:
- This is a very effective policy

Cons:

- Keeping track of the Least Recently Used is easy with 2 way set associative caches, but requires complicated hardware and much time to keep track of larger 4 or 6 way set associative caches

### FIFO (First In, First Out)

Track initial order of being added to the cache and remove accordingly.

### Random

If low temporal locality of workload, works ok

## Caches: Write Policies

**Write through policy**

Always write data to cache and to memory (through cache)

Pros:

- Forces memory and cache to be consistent

Cons:

- Slow (memory access is long)

**Cache coherence (Idea)**

What happens if two programs are referencing the same location in memory concurrently?

This is a similar to the concurrency problem faced by **pragma omp** with data races

**Write back policy**

Write data only to cache - update memory when block is removed (based on block replacement policies)

Pros

- Allows multiple write to cache block = 1 aggregate write to memory

Cons

- Requires dirty bit (1 - cache block needs to be written back)
- Inconsistency between caches and memory

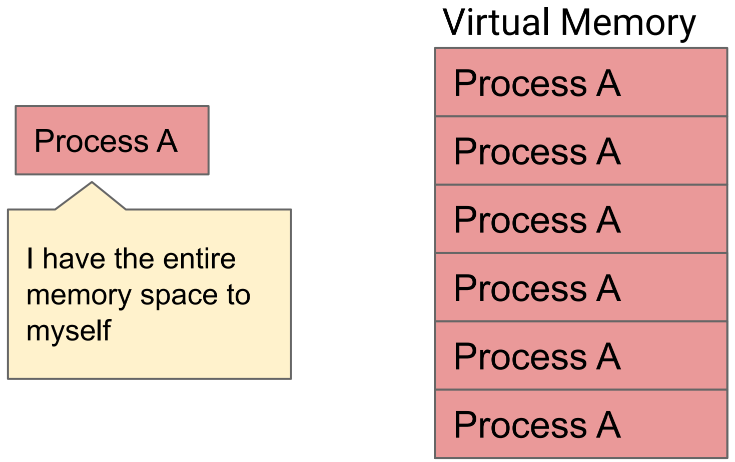
## Virtual Memory

- **Virtual Memory:** a large space that a process believes belongs entirely to itself
- **Physical Memory:** the limited RAM on your computer that all the processes/processors actually share

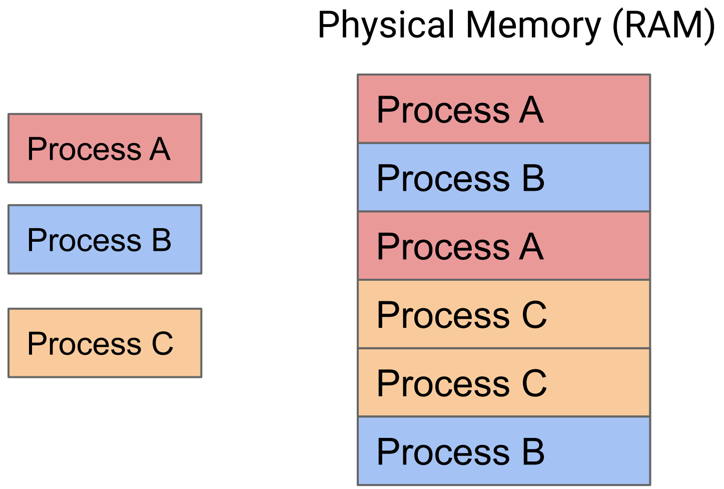
- Virtual Memory:
  - allows multiple processes to simultaneously occupy memory, and doesn’t allow other process to write to other’s memory
  - give each program the illusion of its own private address space

# Virtual Memory

The Illusion:



Reality: Other processes share the same physical memory space!



## Paged Memory

- **paged memory:** divide both physical and virtual memory into large chunks called **pages**
  - virtual page size = physical page size

- **Virtual address**: what your program uses
- **Physical address**: what actually determines where in memory to go

Virtual Address

| | |
|---|---|
| Virtual Page Number | Page offset |

Physical Address

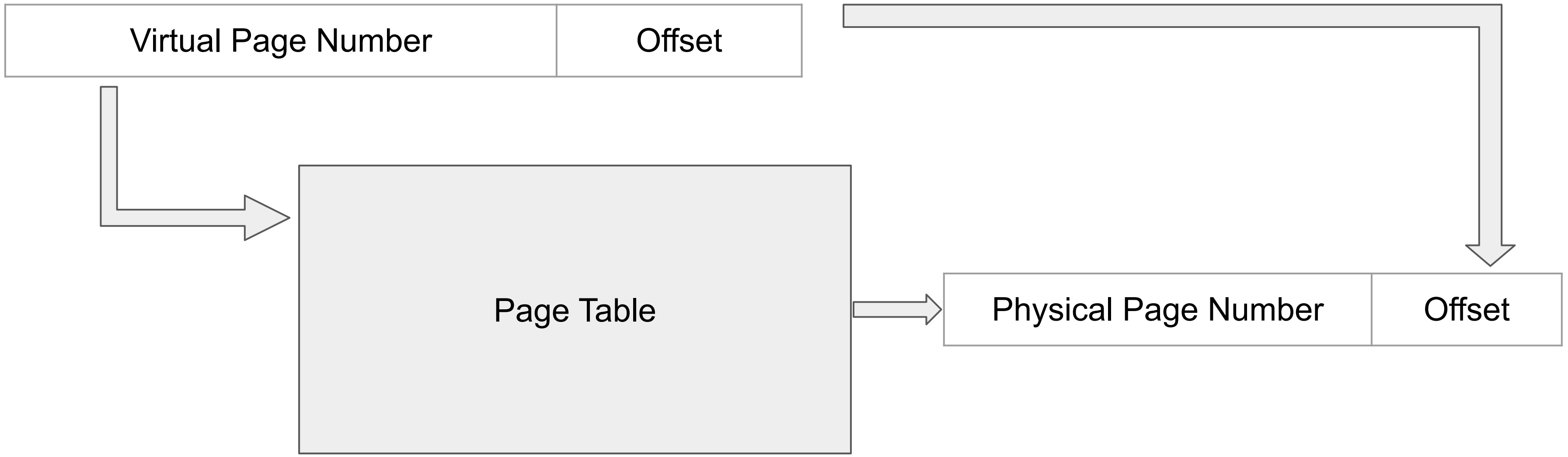
| | |
|---|---|
| Physical Page Number | Page offset |

# Page Table

- **page table:** keeps track of valid VPN → PPN mapping, **stored in memory**

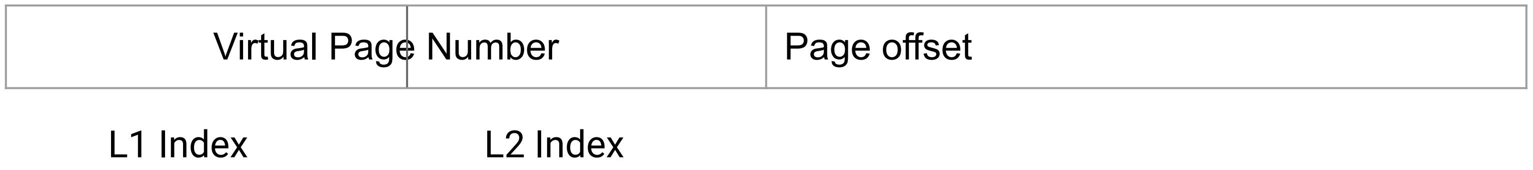
| Valid | Dirty | Permission | PPN |
|---|---|---|---|
| entry 0 | | | |
| entry 1 | | | |

- valid bit = 1: entry is valid, page is in physical memory
    - if 0: **page fault:** fetch page from disk, put into memory, and update the page table
- dirty bit = 1: page has been written to, will need to update disk upon eviction
- permission bits: tells us if we can read/write to this page
- **Page Table Base Register (PTBR):** holds address of the page table

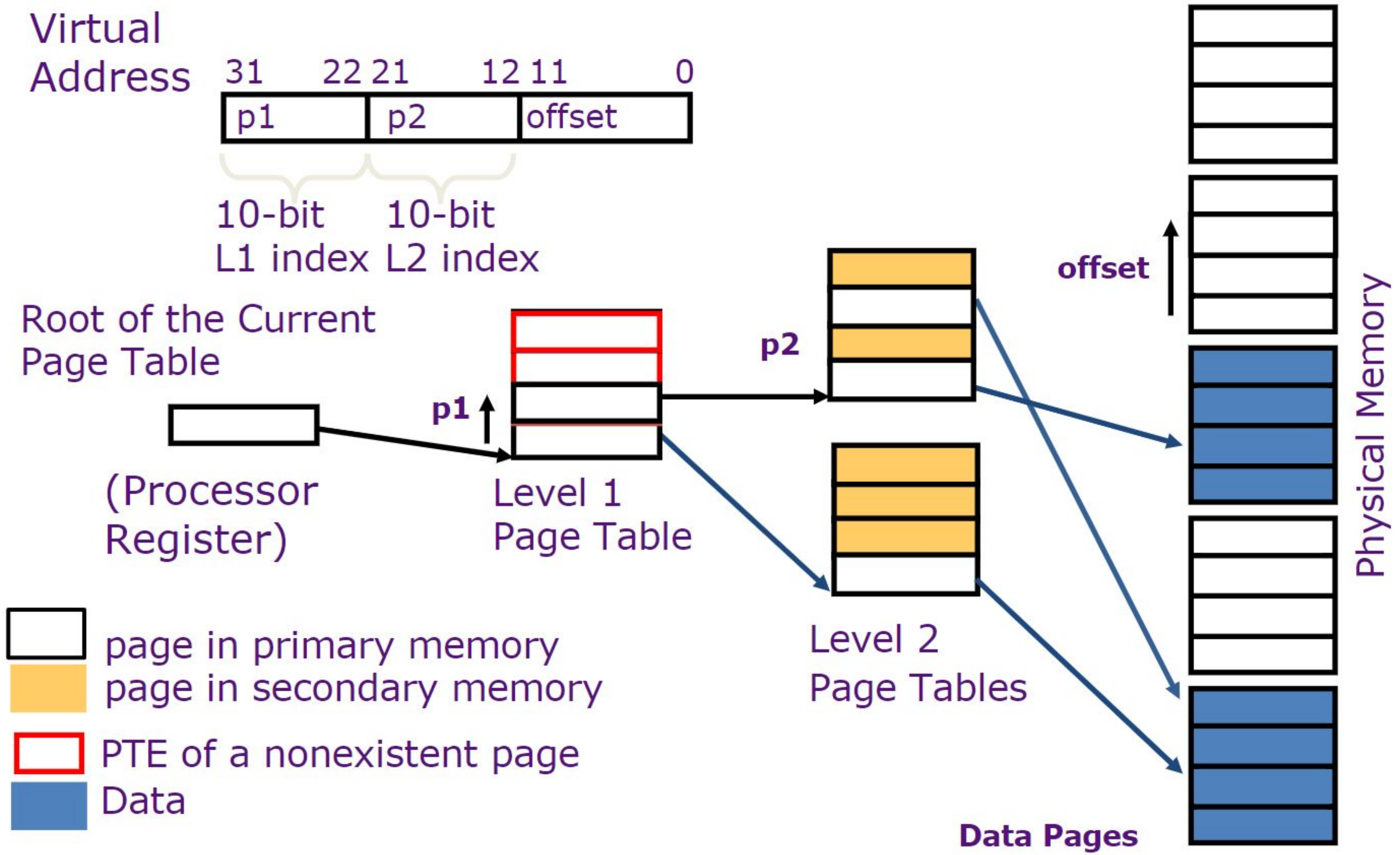

Virtual Page Number
Offset
Page Table
Physical Page Number
Offset

# Page Table

- issue: page table is sparsely populated, and very big, how can we avoid storing the whole thing in memory?
- solution: hierarchical page tables
  - split VPN into multiple sections



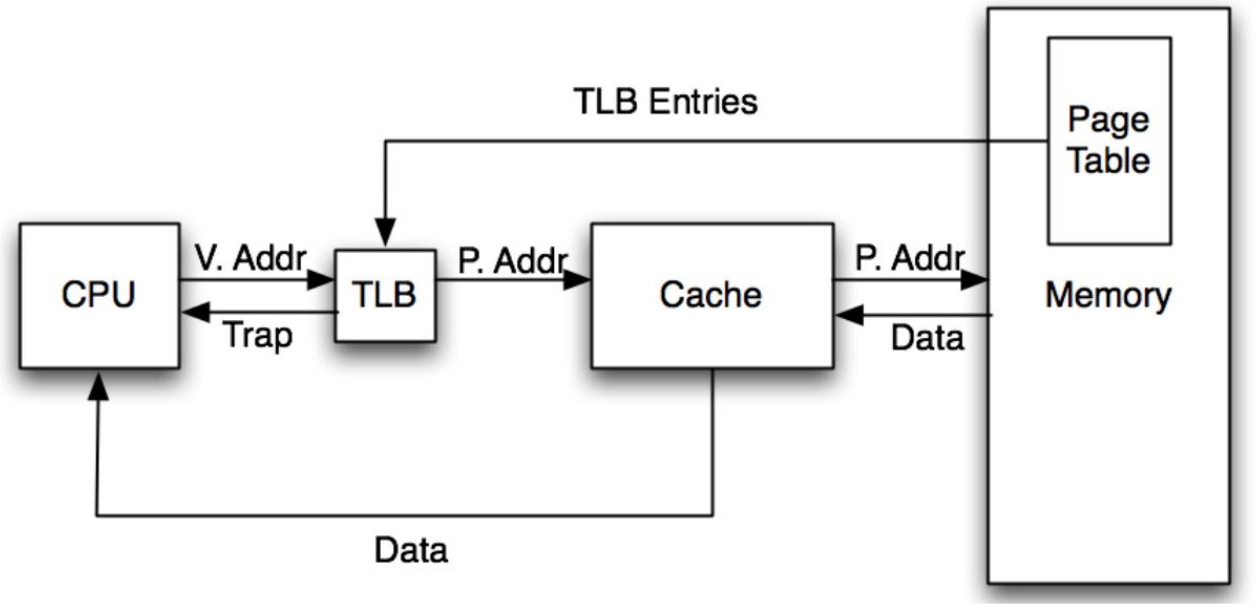
- search the L1 index in the level 1 page table to find the corresponding address of the level 2 page table
- search the L2 index in the level 2 page table to complete the mapping

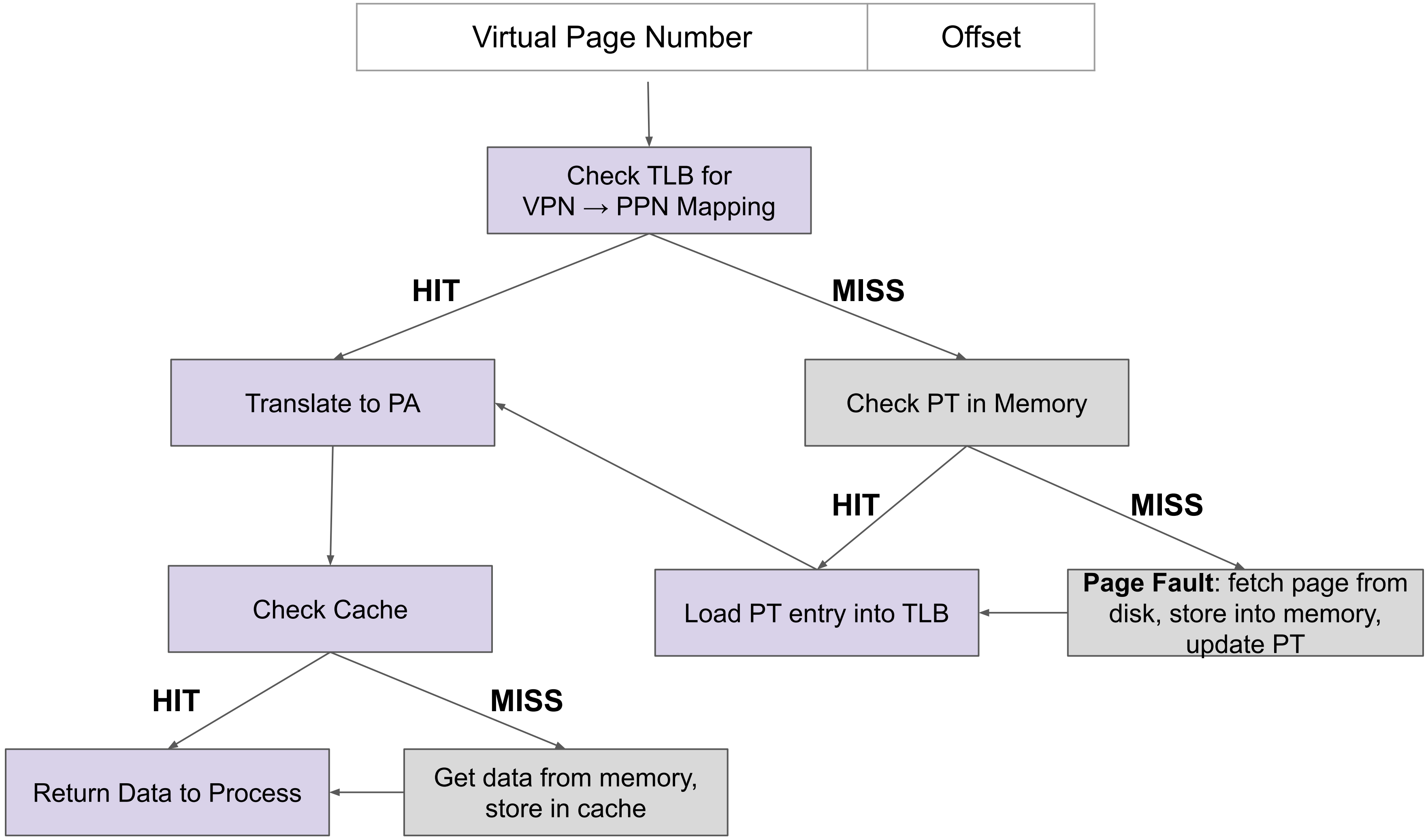

Virtual Address
31
22 21
12 11
0
p1
p2
offset
10-bit L1 index
10-bit L2 index
Root of the Current Page Table
(Processor Register)
p1
Level 1 Page Table
p2
Level 2 Page Tables
offset
Physical Memory
Data Pages
page in primary memory
page in secondary memory
PTE of a nonexistent page
Data

## Translation Lookaside Buffer

- Page table in memory: time expensive!
- solution: **translation lookaside buffer (TLB):** a cache for page table
  - hold a subset of the page table entries
  - pretty small, usually fully associative
- upon TLB miss:
  - go to memory, load PT entry into TLB, return translated address

# Flow


TLB Entries
Page Table
CPU
V. Addr
Trap
TLB
P. Addr
Cache
P. Addr
Data
Memory
Data


Virtual Page Number
Offset
Check TLB for VPN → PPN Mapping
HIT
MISS
Translate to PA
Check PT in Memory
HIT
MISS
Check Cache
Load PT entry into TLB
**Page Fault**: fetch page from disk, store into memory, update PT
HIT
MISS
Return Data to Process
Get data from memory, store in cache

# VM Access Walkthrough

3.1 Suppose we have a system with the following specs:

- Fully associative TLB with 2 entries with LRU replacement policy
- TLB is initially cold
- $2^{11}$ B virtual memory
- 10-bit physical address space
- 256 B page size

(a) How many bits are the VPN and PPN?

(b) How many entries should a single-level page table have?

(c) How many pages are in physical memory?

# VM Access Walkthrough

3.1 Suppose we have a system with the following specs:

- Fully associative TLB with 2 entries with LRU replacement policy
- TLB is initially cold
- $2^{11}$ B virtual memory
- 10-bit physical address space
- 256 B page size

(a) How many bits are the VPN and PPN?

offset 256 = 2^8 B → 8 bit offset   VPN = 11 bit - 8 bit = 3 bit   PPN = 10 bit - 8 bit = 2 bit

(b) How many entries should a single-level page table have?

# of entries = number of VM pages   3 bit VPN → $2^3$ = 8 entries

(c) How many pages are in physical memory?

# of pages = number of PM pages   2 bit PPN → $2^2$ = 4 entries

# VM Access Walkthrough

3.2 Walk through the memory accesses provided in the table below, and fill in the table with whether each access hits in the TLB, page table, or neither.
Assume the following:

- Physical Page 1 is initially occupied by another process (other physical pages are invalid)
- Physical memory has LRU replacement policy
- Break ties by choose the lowest available number

- Fully associative TLB with 2 entries with LRU replacement policy
- TLB is initially cold
- $2^{11}$ B virtual memory
- 10-bit physical address space
- 256 B page size

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | | | | | | | | | | |
| Pg Table | | | | | | | | | | |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | | | | | | | | | | |
| Pg Table | | | | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| | | |
| | | |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | | |
| 1 | 0 | (other) |
| 2 | | |
| 3 | | |

1) Separate Virtual Address
VPN first 3 bits
Offset 8 bits (ignored for this problem)

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | | | | | | | | | | |
| Pg Table | | | | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| | | |
| | | |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | | |
| 1 | 0 | (other) |
| 2 | | |
| 3 | | |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | | | | | | | | | |
| Pg Table | M | | | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| | | |
| | | |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 0 | Pink |
| 1 | 1 | (other) |
| 2 | | |
| 3 | | |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
Update LRU on every access (bc data access reads from memory)
```

Memory Access 0x123

| | |
|---|---|
| TLB: | Empty (Miss) |
| Page Table: | Empty (Fault) |
| Memory: | Fill 0 |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | | | | | | | | | |
| Pg Table | M | | | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 1 | 0 |
| | | |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 0 | Pink |
| 1 | 1 | (other) |
| 2 | | |
| 3 | | |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc data access reads from memory)
```

Memory Access 0x123

| | |
|---|---|
| TLB: | Empty (Miss) |
| Page Table: | Empty (Fault) |
| Memory: | Fill 0 |

Populate
- Table
- TLB

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | | | | | | | | |
| Pg Table | M | M | | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 1 | 0 |
| | | |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 1 | Pink |
| 1 | 2 | (other) |
| 2 | 0 | Purple |
| 3 | | |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x456

| | |
|---|---|
| TLB: | Empty (Miss) |
| Page Table: | Empty (Fault) |
| Memory: | Fill 2 |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | | | | | | | | |
| Pg Table | M | M | | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 1 | Pink |
| 1 | 2 | (other) |
| 2 | 0 | Purple |
| 3 | | |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x456

| | |
|---|---|
| TLB: | Empty (Miss) |
| Page Table: | Empty (Fault) |
| Memory: | Fill 2 |

Populate
- Table
- TLB

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | | | | | | | |
| Pg Table | M | M | M | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | | |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 2 | Pink |
| 1 | 3 | (other) |
| 2 | 1 | Purple |
| 3 | 0 | Blue |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
           Update PT (valid bit)
           Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x712

| | |
|---|---|
| TLB: | Full (Miss) |
| Page Table: | Empty (Fault) |
| Memory: | Fill 3 |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | | | | | | | |
| Pg Table | M | M | M | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 7 | 3 |
| 1 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 2 | Pink |
| 1 | 3 | (other) |
| 2 | 1 | Purple |
| 3 | 0 | Blue |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
Update LRU on every access (bc data access reads from memory)
```

Memory Access 0x712

| | |
|---|---|
| TLB: | Full (Miss) |
| Page Table: | Empty (Fault) |
| Memory: | Fill 3 |

Populate
- Table
- TLB (kick out pink by LRU)

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | | | | | | |
| Pg Table | M | M | M | | | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 7 | 3 |
| 0 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 2 | Pink |
| 1 | 3 | (other) |
| 2 | 0 | Purple |
| 3 | 1 | Blue |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x400

| | |
|---|---|
| TLB: | Hit (VPN 4) |
| Page Table: | N/A |
| Memory: | N/A |

Populate
- Update LRU of TLB
- Update LRU of Physical memory

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | | | | | |
| Pg Table | M | M | M | | M | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 7 | 3 |
| 0 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | | |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 3 | Pink |
| 1 | 0 | Orange |
| 2 | 1 | Purple |
| 3 | 2 | Blue |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x321

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Miss |
| Memory: | Kick index 1 |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | | | | | |
| Pg Table | M | M | M | | M | | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 3 | 1 |
| 1 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 3 | Pink |
| 1 | 0 | Orange |
| 2 | 1 | Purple |
| 3 | 2 | Blue |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc data access reads from memory)
```

Memory Access 0x321

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Miss |
| Memory: | Kick index 1 |

Populate
- Page Table
- TLB (Kick VPN 7)

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | | | | |
| Pg Table | M | M | M | | M | H | | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 3 | 1 |
| 1 | 4 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 3 | Pink |
| 1 | 0 | Orange |
| 2 | 1 | Purple |
| 3 | 2 | Blue |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x1AF

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Hit |
| Memory: | N/A |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | | | | |
| Pg Table | M | M | M | | M | H | | | | |

Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
      Update TLB
    - No → kick page in Memory
      Update PT (valid bit)
      Update TLB (LRU)

Update LRU on every access (bc data access reads from memory)

Memory Access 0x1AF

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Hit |
| Memory: | N/A |

Populate
- TLB (kick VPN 4) / Update LRU
- Update LRU in Physical Memory b/c data memory access

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 3 | 1 |
| 0 | 1 | 0 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 0 | Pink |
| 1 | 1 | Orange |
| 2 | 2 | Purple |
| 3 | 3 | Blue |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | M | | | |
| Pg Table | M | M | M | | M | H | M | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 3 | 1 |
| 0 | 1 | 0 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | | |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 1 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 1 | Pink |
| 1 | 2 | Orange |
| 2 | 3 | Purple |
| 3 | 0 | Gray |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
     - Yes → return PPN
             Update TLB
     - No → kick page in Memory
             Update PT (valid bit)
             Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x0FE

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Miss |
| Memory: | Kick Index 3 (Blue) |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | M | | | |
| Pg Table | M | M | M | | M | H | M | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 0 | 3 |
| 1 | 1 | 0 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 0 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 1 | Pink |
| 1 | 2 | Orange |
| 2 | 3 | Purple |
| 3 | 0 | Gray |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
```

Update LRU on every access (bc data access reads from memory)

Memory Access 0x0FE

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Miss |
| Memory: | Kick Index 3 (Blue) |

Populate

- Page Table (Invalidate, Blue)
- TLB (Kick Orange)

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | M | H | | |
| Pg Table | M | M | M | | M | H | M | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 0 | 3 |
| 0 | 1 | 0 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 0 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 0 | Pink |
| 1 | 2 | Orange |
| 2 | 3 | Purple |
| 3 | 1 | Gray |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x164

| | |
|---|---|
| TLB: | Hit (Update LRU) |
| Page Table: | N/A |
| Memory: | Update LRU |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | M | H | H | |
| Pg Table | M | M | M | | M | H | M | | | |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 0 | 3 |
| 1 | 1 | 0 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 0 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 1 | Pink |
| 1 | 2 | Orange |
| 2 | 3 | Purple |
| 3 | 0 | Gray |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
Update LRU on every access (bc
data access reads from memory)
```

Memory Access 0x001

| | |
|---|---|
| TLB: | Hit (Update LRU) |
| Page Table: | N/A |
| Memory: | Update LRU |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | M | H | H | M |
| Pg Table | M | M | M | | M | H | M | | | M |

TLB

| LRU | VPN | PPN |
|---|---|---|
| 0 | 0 | 3 |
| 1 | 1 | 0 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 1 | 0 |
| 2 | | |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | | |
| 6 | | |
| 7 | 0 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 2 | Pink |
| 1 | 3 | Orange |
| 2 | 0 | Red |
| 3 | 1 | Gray |

```
Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
            Update TLB
    - No → kick page in Memory
            Update PT (valid bit)
            Update TLB (LRU)
Update LRU on every access (bc data access reads from memory)
```

Memory Access 0x632

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Miss |
| Memory: | Kick Purple |

# VM Access Walkthrough

| Memory Access | 0x123 | 0x456 | 0x712 | 0x400 | 0x321 | 0x1AF | 0x0FE | 0x164 | 0x001 | 0x632 |
|---|---|---|---|---|---|---|---|---|---|---|
| TLB | M | M | M | H | M | M | M | H | H | M |
| Pg Table | M | M | M |  | M | H | M |  |  | M |

Algorithm
- Is VPN in TLB?
  - Yes → get PPN (done)
  - No → check PT if valid
    - Yes → return PPN
      Update TLB
    - No → kick page in Memory
      Update PT (valid bit)
      Update TLB (LRU)

Update LRU on every access (bc data access reads from memory)

TLB

| LRU | VPN | PPN |
|---|---|---|
| 1 | 0 | 3 |
| 0 | 6 | 2 |

Page Table

| VPN | Valid | PPN |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 1 | 0 |
| 2 |  |  |
| 3 | 1 | 1 |
| 4 | 0 | 2 |
| 5 |  |  |
| 6 | 1 | 2 |
| 7 | 0 | 3 |

Physical Memory

| Index | LRU | Page |
|---|---|---|
| 0 | 2 | Pink |
| 1 | 3 | Orange |
| 2 | 0 | Red |
| 3 | 1 | Gray |

Memory Access 0x632

| | |
|---|---|
| TLB: | Miss |
| Page Table: | Miss |
| Memory: | Kick Purple |

Populate
- Page Table (Invalidate Purple)
- Update TLB (Kick, Pink)

# SIMD (Single Instruction, Multiple Data)

Data-Level Parallelism - Executing one operation on multiple data streams

Intel’s SIMD instruction set - fetch 1 instruction → do work of multiple instructions

```
_mm_add_epi32(first_values, second_values)
```

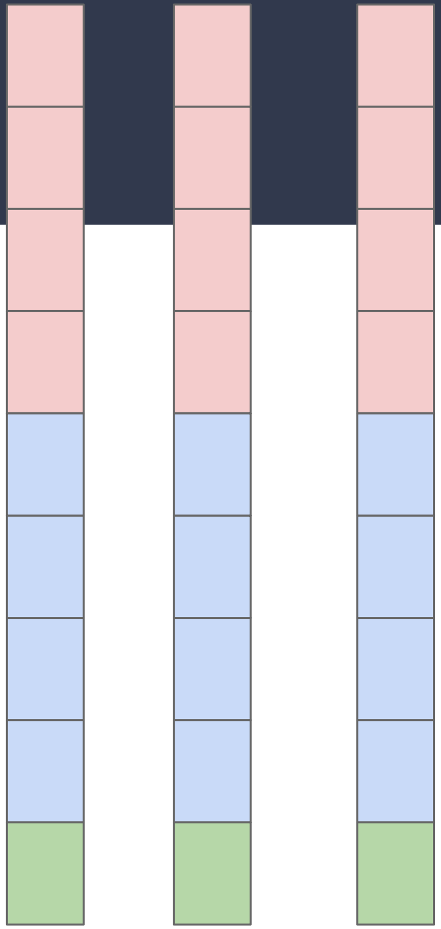
**M**ulti**M**edia extension
(They all start with this)

Arguments are
**E**xtended **P**acked **I**ntegers,
each **32**-bits in size
(signed)

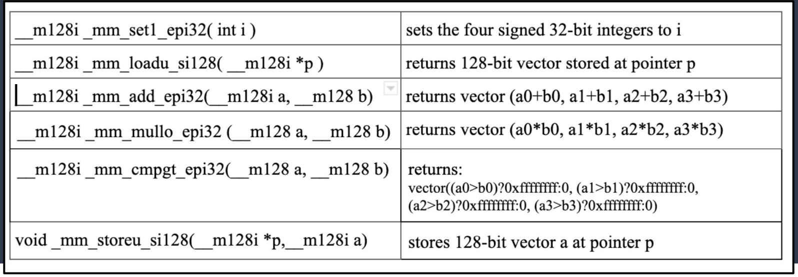
## SIMD

Standard Workflow of SIMD Code

1. Define a result **vector**
2. for loop through N / 4 * 4 while incrementing += 4
   a. **Load** operation into vectorized format
   b. Conduct operation maintain temporary result in result vector
3. **store** vector result into **array** format
4. handle tail case from N / 4 * 4  to N
   a. Conduct operation on 1 data each time
5. Result the aggregate of all results in array



# SIMD: Example



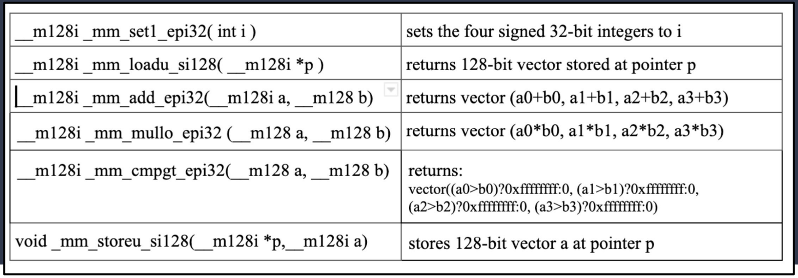
| | |
|---|---|
| __m128i _mm_set1_epi32( int i ) | sets the four signed 32-bit integers to i |
| __m128i _mm_loadu_si128( __m128i *p ) | returns 128-bit vector stored at pointer p |
| __m128i _mm_add_epi32(__m128i a, __m128 b) | returns vector (a0+b0, a1+b1, a2+b2, a3+b3) |
| __m128i _mm_mullo_epi32 (__m128 a, __m128 b) | returns vector (a0*b0, a1*b1, a2*b2, a3*b3) |
| __m128i _mm_cmpgt_epi32(__m128 a, __m128 b) | returns: vector((a0>b0)?0xffffffff:0, (a1>b1)?0xffffffff:0, (a2>b2)?0xffffffff:0, (a3>b3)?0xffffffff:0) |
| void _mm_storeu_si128(__m128i *p,__m128i a) | stores 128-bit vector a at pointer p |

1.2 Implement the following function using SIMD:

```
// SIMD code
static int selective_sum_vectorized (int n, int *a, int c) {
    int result[4];
    _m128i sum_v = _______________________________;
    _m128i cond_v = _______________________________;

    for (int i = 0; i < ___; i += ___) { //Vectorized loop
        _m128i curr_v = m128i_mm_loadu_si128(______________);
        __m128i tmp = _mm_cmpgt_epi32(____________);
        sum_v = _______________________________;
    }
   _mm_storeu_si128(________________________________________);

    for (int i = ___; i < ___; i += 1) { //Tail case
        result[0] += ___________________________________;
    }
  return ____________________________________________________;
```

```
// Sequential code
static int selective_sum_total (int n, int *a, int c) {
   int sum = 0;
   for (int i = 0; i < n; i += 1) {
      if (a[i] > c) {
         sum += a[i];
      }
   return sum;
}
```

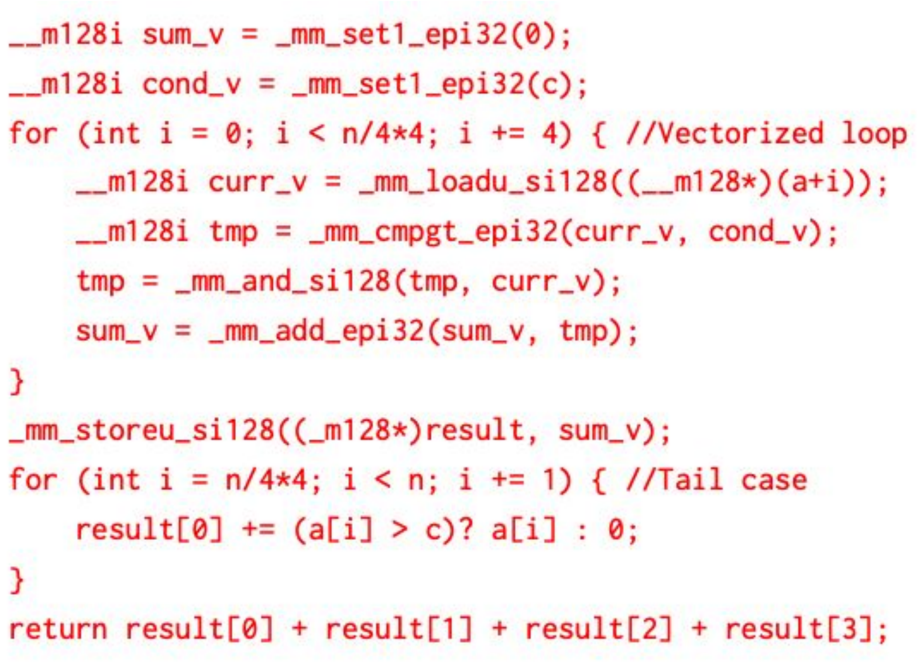
# SIMD: Example



| | |
|---|---|
| __m128i _mm_set1_epi32( int i ) | sets the four signed 32-bit integers to i |
| __m128i _mm_loadu_si128( __m128i *p ) | returns 128-bit vector stored at pointer p |
| __m128i _mm_add_epi32(__m128i a, __m128 b) | returns vector (a0+b0, a1+b1, a2+b2, a3+b3) |
| __m128i _mm_mullo_epi32 (__m128 a, __m128 b) | returns vector (a0*b0, a1*b1, a2*b2, a3*b3) |
| __m128i _mm_cmpgt_epi32(__m128 a, __m128 b) | returns: vector((a0>b0)?0xffffffff:0, (a1>b1)?0xffffffff:0, (a2>b2)?0xffffffff:0, (a3>b3)?0xffffffff:0) |
| void _mm_storeu_si128(__m128i *p,__m128i a) | stores 128-bit vector a at pointer p |

1.2 Implement the following function using SIMD:

```
// Sequential code
static int selective_sum_total (int n, int *a, int c) {
    int sum = 0;
    for (int i = 0; i < n; i += 1) {
        if (a[i] > c) {
            sum += a[i];
        }
    return sum;
}
```



```
__m128i sum_v = _mm_set1_epi32(0);
__m128i cond_v = _mm_set1_epi32(c);
for (int i = 0; i < n/4*4; i += 4) { //Vectorized loop
    __m128i curr_v = _mm_loadu_si128((__m128*)(a+i));
    __m128i tmp = _mm_cmpgt_epi32(curr_v, cond_v);
    tmp = _mm_and_si128(tmp, curr_v);
    sum_v = _mm_add_epi32(sum_v, tmp);
}
_mm_storeu_si128((_m128*)result, sum_v);
for (int i = n/4*4; i < n; i += 1) { //Tail case
    result[0] += (a[i] > c)? a[i] : 0;
}
return result[0] + result[1] + result[2] + result[3];
```

## Thread-Level Parallelism

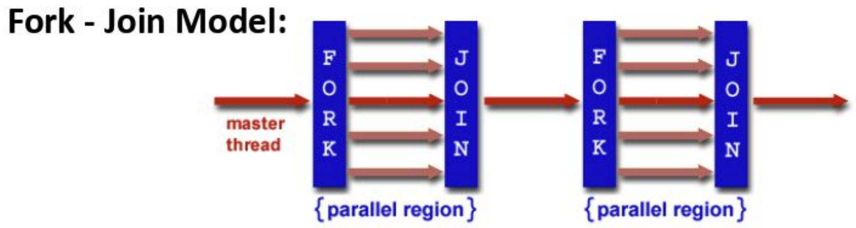
- Big question: how do we make computers faster?
- We can only go so fast performing one instruction at a time
    - If the clock rate gets any faster, there's too much heat produced!

- So why not perform multiple at a time?
- Hardware can be split up for extra speeed
    - Example of MIMD: each core can process a different instruction!

When you can perform more than 1 instruction at a time

# What is OpenMP?

- An API used for abstracting multi-core processing
- Uses a **Fork-Join** model:
  - Fork: Master thread creates team of parallel threads
  - Join: when team threads complete the statement in parallel region, they synchronize and “join” back together into 1 master thread



# How to use OpenMP

- Enclosed parallelized code in a **pragma**

```
int x; int y; // shared
#pragma omp parallel private (y)
{
    // code goes here
}
```

**Things to note:**

`x` is shared amongst threads, `y` is not!

Curly brace MUST go on the next line

- Each thread runs a copy of code within block
- Order is **non-deterministic**
- Variable defined outside pragma are shared
    - Unless the **private** keyword is used

## Useful OpenMP Features

```
#pragma omp parallel
{
     #pragma omp for
     for (int i = 0; i < n; i += 1) {
          // do stuff
     }
}
```

Is the same as ...

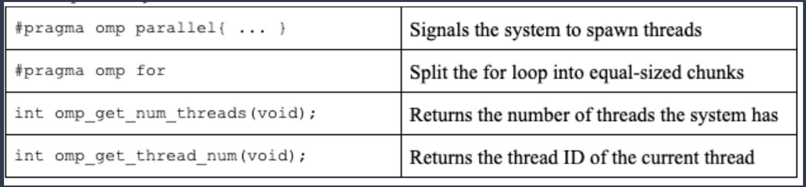
```
#pragma omp parallel for
for (int i = 0; i < n; i += 1) {
     // do stuff
}
```

`OMP_NUM_THREADS`

Thread sharing methods:

- OpenMP intrinsic to set number of threads:
  `omp_set_num_threads(x);`
- OpenMP intrinsic to get number of threads:
  `num_th = omp_get_num_threads();`
- OpenMP intrinsic to get Thread ID number:
  `th_ID = omp_get_thread_num();`

# Practice Question!



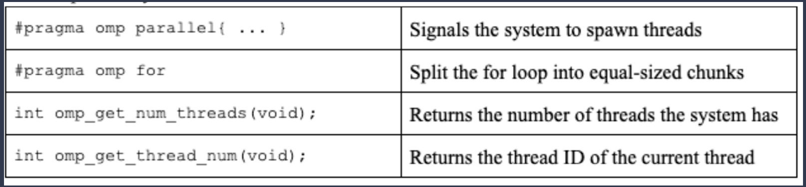
| | |
|---|---|
| `#pragma omp parallel{ ... }` | Signals the system to spawn threads |
| `#pragma omp for` | Split the for loop into equal-sized chunks |
| `int omp_get_num_threads(void);` | Returns the number of threads the system has |
| `int omp_get_thread_num(void);` | Returns the thread ID of the current thread |

2.2 Implement the following function using openMP:

```
// openMP for code
static int selective_square_parallelized (int n, int *a, int c) {
    #pragma omp parallel {
    #pragma omp for
    for (int i = ___; i < ___; i += ___) {
        if (_____ > _____) {
            a[i] *= _____;
        }
    }
    return a;
}
```

```
// Sequential code
static int selective_square_total (int n, int *a, int c) {
    for (int i = 0; i < n; i += 1) {
        if (a[i] > c) {
            a[i] *= a[i];
        }
    }
    return a;
}
```

# Practice Question!



| | |
|---|---|
| #pragma omp parallel{ ... } | Signals the system to spawn threads |
| #pragma omp for | Split the for loop into equal-sized chunks |
| int omp_get_num_threads(void); | Returns the number of threads the system has |
| int omp_get_thread_num(void); | Returns the thread ID of the current thread |

2.2 Implement the following function using openMP:

```
// openMP for code
static int selective_square_parallelized (int n, int *a, int c) {
    #pragma omp parallel {
    #pragma omp for
    for (int i = _0_; i < _n_; i += _1_) {
        if (_a[i]_ > _c_) {
            a[i] *= _a[i]_;
        }
    }
    return a;
}
```

```
// Sequential code
static int selective_square_total (int n, int *a, int c) {
    for (int i = 0; i < n; i += 1) {
        if (a[i] > c) {
            a[i] *= a[i];
        }
    }
    return a;
}
```

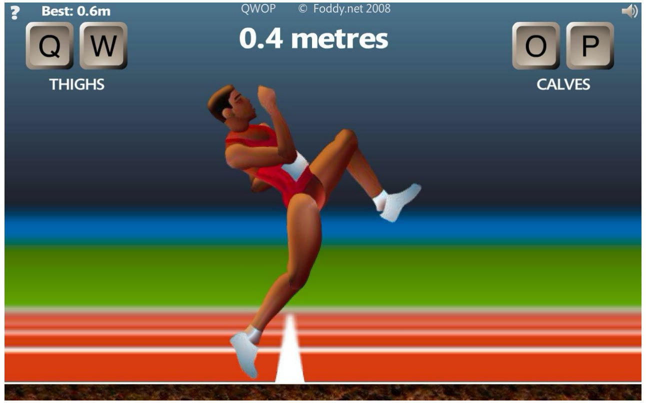
Answer: nothing changes!
#pragma omp for already takes care of the parallelism for us

# Data Races

- When shared variables are being modified in different ways by different threads, the final value is unknown
  - This is called **nondeterminism**

Solutions:

- Use a critical section
  - `#pragma omp critical`
  - Only allows one thread in at a time
- Create locks using atomic operations
  - OpenMP has `omp_lock_t, omp_init_lock, omp_set_lock`
  - RISC-V has `amoswap`



If your variables are uncoordinated, there will be a data race… and it won't be pretty…

# Data Race Practice

3.1 Suppose we have `int *A` that points to the head of an array of length `len`. Determine which statement (A) - (E) correctly describes the code execution and provide a one or two sentence justification.

(a) Consider the following code:

```
#pragma omp parallel for
for (int x = 0; x < len; x++){
    *A = x;
    A++;
}
```

Is the code
A) Always Incorrect
B) Sometimes Incorrect
C) Always Correct, Slower than Serial
D) Always Correct, Speed relative to Serial depends on Caching Scheme
E) Always Correct, Faster than Serial

# Data Race Practice

3.1 Suppose we have `int *A` that points to the head of an array of length `len`. Determine which statement (A) - (E) correctly describes the code execution and provide a one or two sentence justification.

(a) Consider the following code:

```
#pragma omp parallel for
for (int x = 0; x < len; x++){
    *A = x;
    A++;
}
```

Is the code
A) Always Incorrect
[✓ B) Sometimes Incorrect]
C) Always Correct, Slower than Serial
D) Always Correct, Speed relative to Serial depends on Caching Scheme
E) Always Correct, Faster than Serial

Answer:

For loop is split across threads, data race incrementing A. But if each thread runs in disjoint, answer will still be as expected.

# Data Race Practice

3.2 Suppose we have `int *A` that points to the head of an array of length `len`. Determine which statement (A) - (E) correctly describes the code execution and provide a one or two sentence justification.

(b) Consider the following code:

```
#pragma omp parallel
{
    for (int x = 0; x < len; x++){
        *(A+x) = x;
    }
}
```

Is the code
A) Always Incorrect
B) Sometimes Incorrect
C) Always Correct, Slower than Serial
D) Always Correct, Speed relative to Serial depends on Caching Scheme
E) Always Correct, Faster than Serial

# Data Race Practice

3.2 Suppose we have `int *A` that points to the head of an array of length `len`. Determine which statement (A) - (E) correctly describes the code execution and provide a one or two sentence justification.

(b) Consider the following code:

```
#pragma omp parallel
{
    for (int x = 0; x < len; x++){
        *(A+x) = x;
    }
}
```

Is the code
A) Always Incorrect
B) Sometimes Incorrect
[✓ C) Always Correct, Slower than Serial]
D) Always Correct, Speed relative to Serial depends on Caching Scheme
E) Always Correct, Faster than Serial

Answer:

Each thread will run this for loop to its entirety, so each *(A + x) will be set len times, but nothing changes.