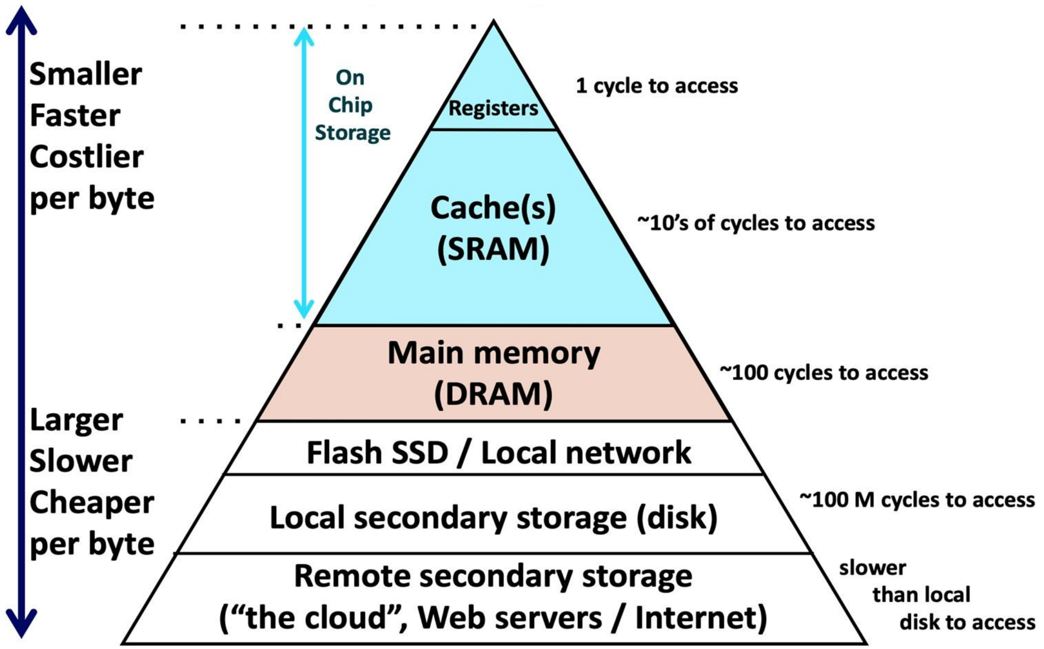
Memory Organization

COMP201 Lab Session
Spring 2023

KOÇ UNIVERSITY

Recall: Memory Hierarchy



KOÇ UNIVERSITY

Why do we need Memory Hierarchies?

Some fundamental properties of computer systems

- Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
- The gap between CPU and main memory speed is widening.
- Locality comes to the rescue!

These fundamental properties of hardware and software suggest an approach for organizing memory and storage systems known as a memory hierarchy.

Fundamental idea of a memory hierarchy

- For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.

(Ideal): The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

KOÇ UNIVERSITY

Caching in Memory Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware MMU
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Virtual Memory	4-KB pages	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

KOÇ UNIVERSITY

Cache Example #1: TIO Breakdown

Assume a system with the following properties:

- Cache Size: 1 MB
- Block Size: 64 Bytes
- 4-way Set-Associative
- 36-bit byte-addressable address space.

Complete the TIO address breakdown:

TAG	SET	OFFSET
36 - 12 - 6	**12**	**6**

KOÇ UNIVERSITY

Cache Example #2: TIO Breakdown

Assume a system with the following properties:

- Cache Size: 16 KB
- Line Size: 32 Bytes

What would be the values of each of the three fields for the following addresses?

Address	Tag	Index	Offset
`0x00B248AC`			
`0x5002AEF3`			
`0x10203000`			
`0x0023AF7C`			

Cache Example #2: TIO Breakdown

Assume a system with the following properties:

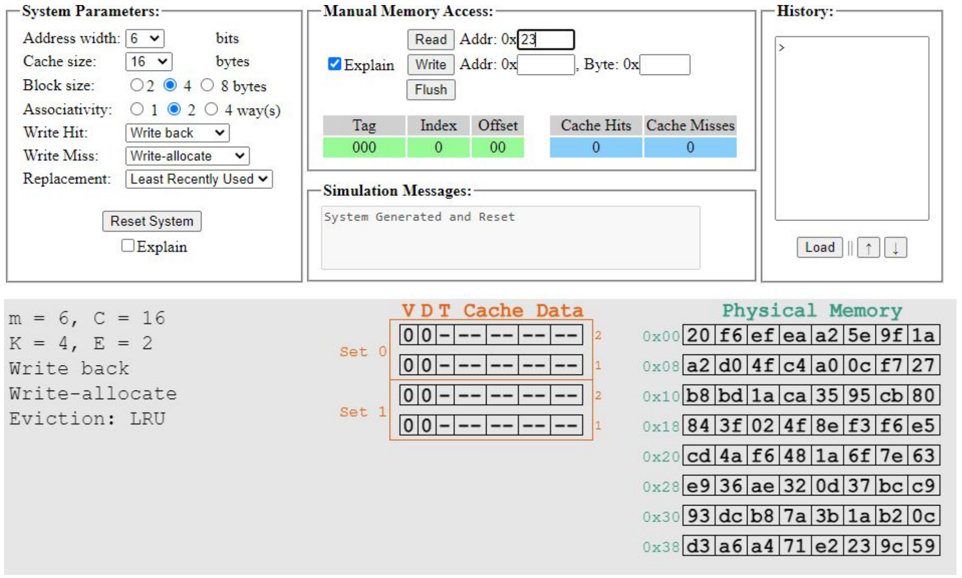
- Cache Size: 16 KB
- Line Size: 32 Bytes

What would be the values of each of the three fields for the following addresses?

Address	Tag	Index	Offset
0x00B248AC	0x2C9	0x45	0xC
0x5002AEF3	0x1400A	0x177	0x13
0x10203000	0x4080	0x180	0x0
0x0023AF7C	0x8E	0x17B	0x1C

Cache Simulator

- Simulates usage of Cache
- Step-by-step explanation
- Adjustable system parameters
- Cache hits, misses, counts and history
- Physical Memory and Cache Memory can be visualized



System Parameters:	
Address width:	6 bits
Cache size:	16 bytes
Block size:	2 [✓ 4] 8 bytes
Associativity:	1 [✓ 2] 4 way(s)
Write Hit:	Write back
Write Miss:	Write-allocate
Replacement:	Least Recently Used

[✓ Explain] Read Addr: 0x23

Tag	Index	Offset	Cache Hits	Cache Misses
000	0	00	0	0

```
System Generated and Reset
```

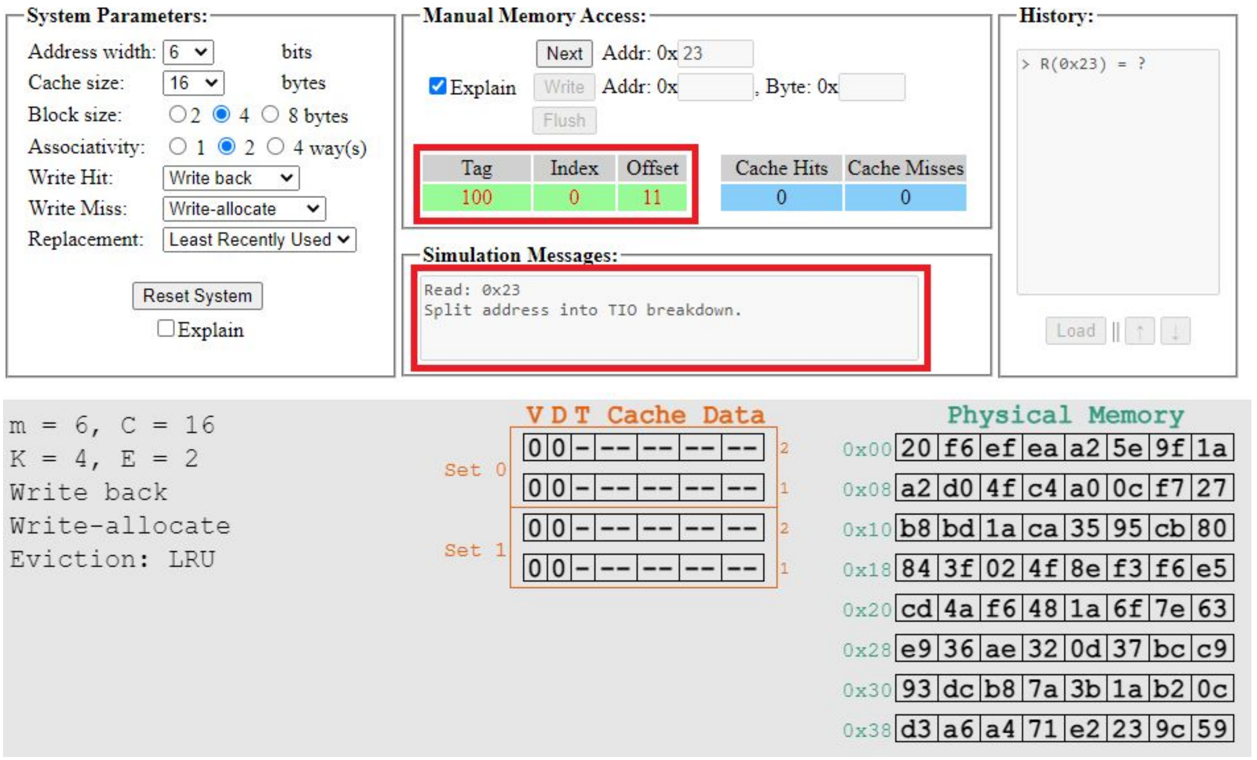
```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

	V	D	T	Cache Data				
Set 0	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory								
0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59

KOÇ UNIVERSITY

https://courses.cs.washington.edu/courses/cse351/cachesim/



System Parameters:

Address width: 6 bits
Cache size: 16 bytes
Block size: ○ 2 [✓ 4] ○ 8 bytes
Associativity: ○ 1 [✓ 2] ○ 4 way(s)
Write Hit: Write back
Write Miss: Write-allocate
Replacement: Least Recently Used

Reset System
☐ Explain

Manual Memory Access:

Next Addr: 0x 23
☑ Explain Write Addr: 0x , Byte: 0x
Flush

Tag	Index	Offset
100	0	11

Cache Hits	Cache Misses
0	0

Simulation Messages:

```
Read: 0x23
Split address into TIO breakdown.
```

History:

```
> R(0x23) = ?
```

Load || ↑ ↓

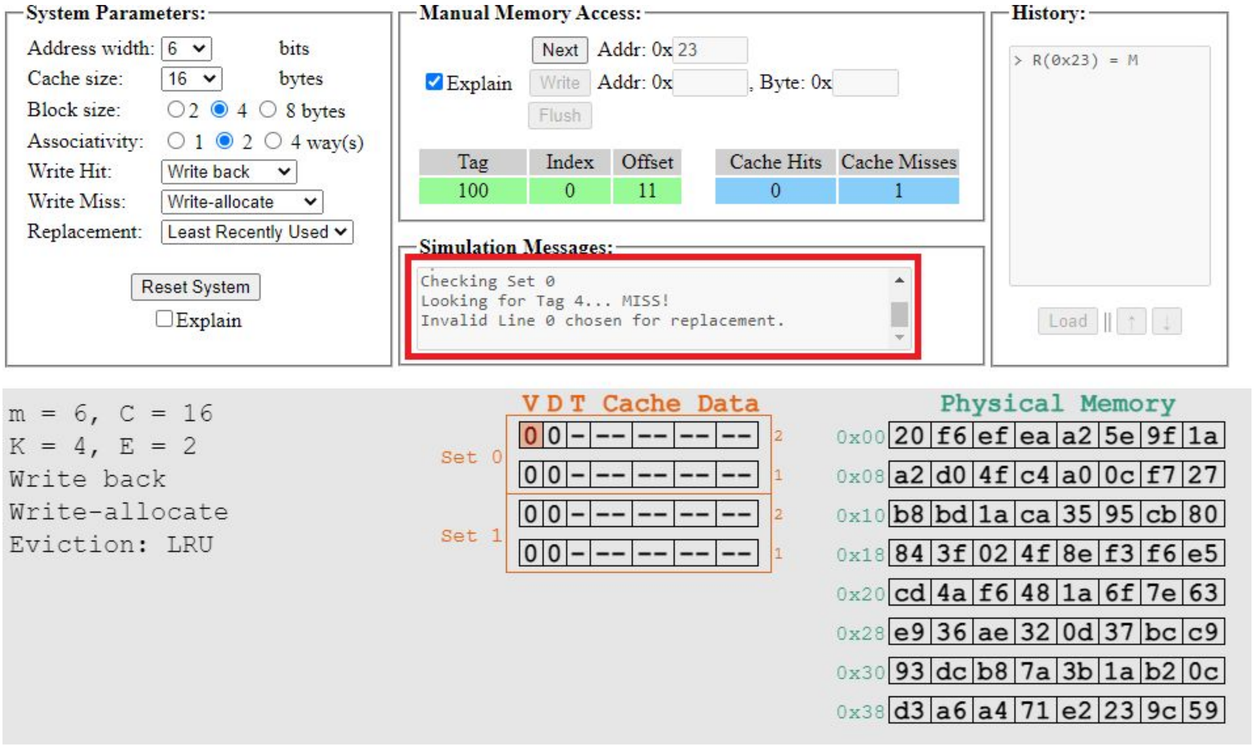
```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

V D T Cache Data

	V	D	T	Cache Data				
Set 0	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59



System Parameters:

Address width:	6	bits
Cache size:	16	bytes
Block size:	○ 2 ● 4 ○ 8	bytes
Associativity:	○ 1 ● 2 ○ 4	way(s)
Write Hit:	Write back	
Write Miss:	Write-allocate	
Replacement:	Least Recently Used	

Reset System

☐ Explain

Manual Memory Access:

☑ Explain

Next Addr: 0x 23

Write Addr: 0x , Byte: 0x

Flush

Tag	Index	Offset
100	0	11

Cache Hits	Cache Misses
0	1

Simulation Messages:

```
Checking Set 0
Looking for Tag 4... MISS!
Invalid Line 0 chosen for replacement.
```

History:

```
> R(0x23) = M
```

Load || ↑ ↓

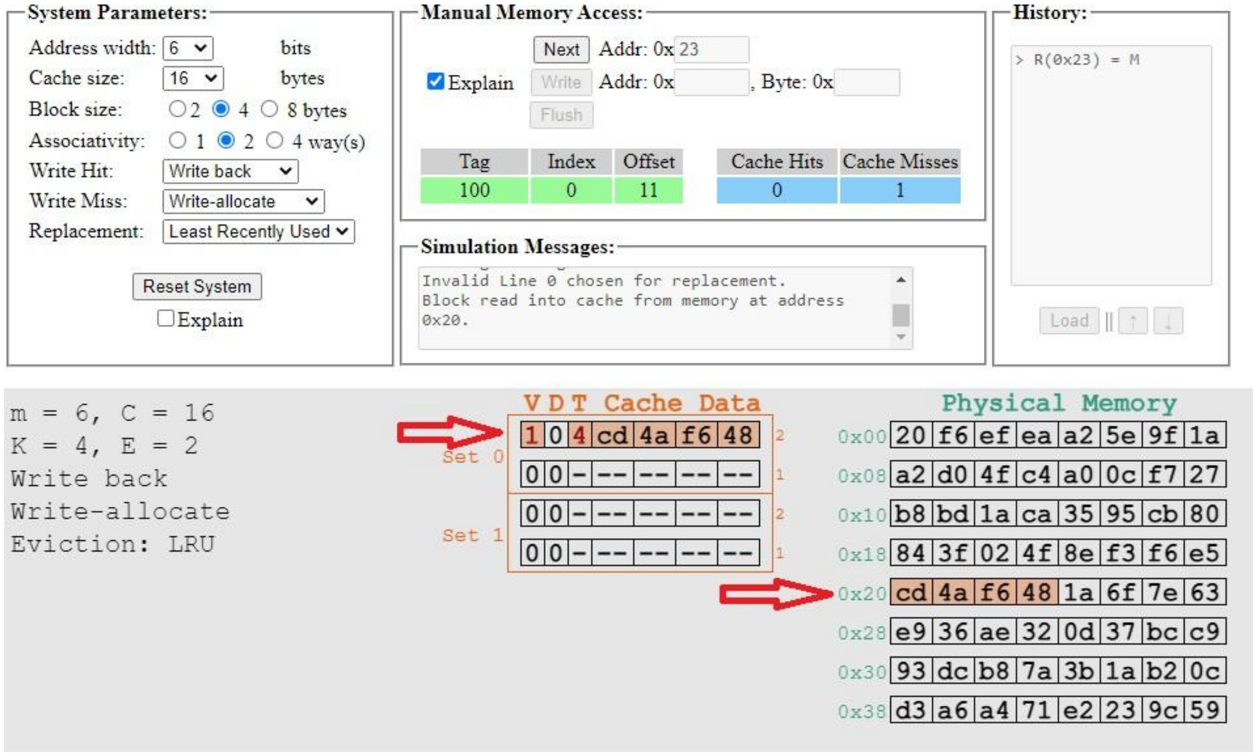
```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

V D T Cache Data

	V	D	T	Cache Data				
Set 0	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59



System Parameters:

Address width:	6	bits
Cache size:	16	bytes
Block size:	○ 2 ● 4 ○ 8	bytes
Associativity:	○ 1 ● 2 ○ 4	way(s)
Write Hit:	Write back	
Write Miss:	Write-allocate	
Replacement:	Least Recently Used	

Manual Memory Access:

Addr: 0x 23

Addr: 0x , Byte: 0x

Tag	Index	Offset
100	0	11

Cache Hits	Cache Misses
0	1

History:

```
> R(0x23) = M
```

Simulation Messages:

```
Invalid Line 0 chosen for replacement.
Block read into cache from memory at address
0x20.
```

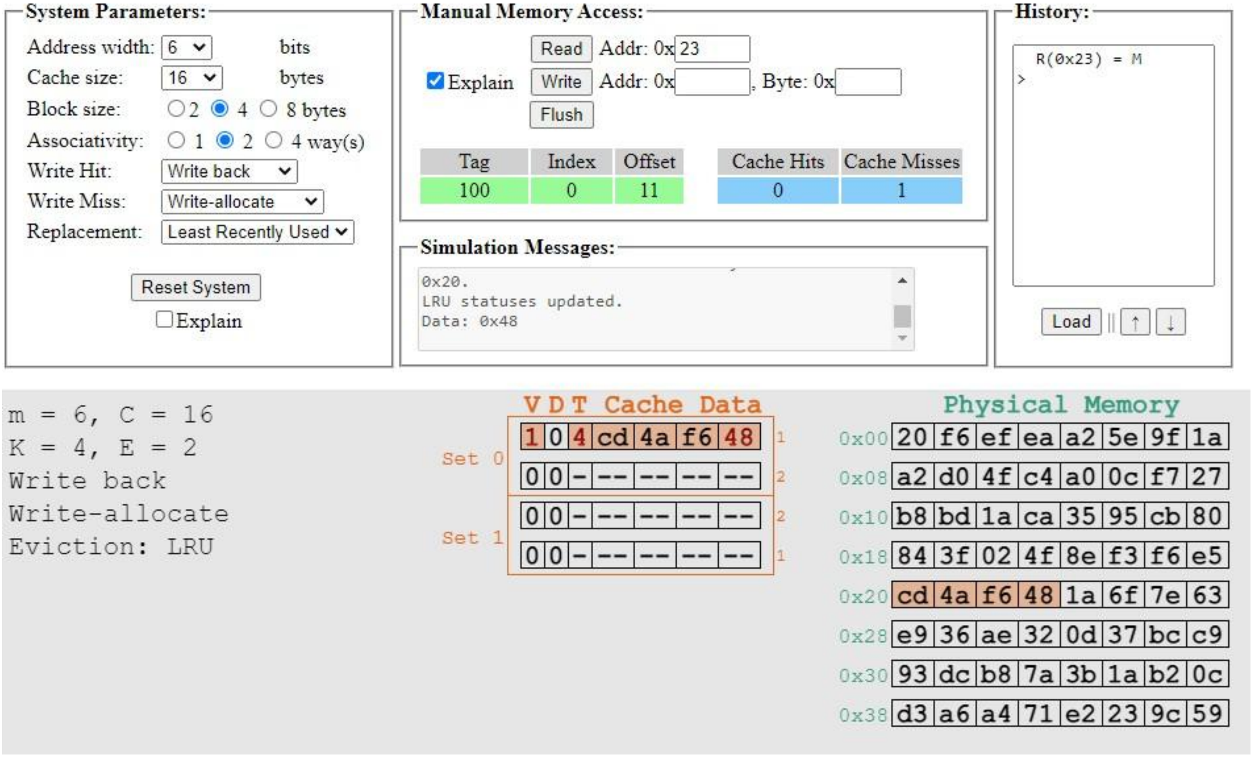
```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

V D T Cache Data

	V	D	T					
Set 0	1	0	4	cd	4a	f6	48	2
	0	0	-	--	--	--	--	1
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59



System Parameters:

Address width:	6	bits
Cache size:	16	bytes
Block size:	○ 2 ● 4 ○ 8	bytes
Associativity:	○ 1 ● 2 ○ 4	way(s)
Write Hit:	Write back	
Write Miss:	Write-allocate	
Replacement:	Least Recently Used	

Reset System

☐ Explain

Manual Memory Access:

☑ Explain

Read Addr: 0x 23

Write Addr: 0x , Byte: 0x

Flush

Tag	Index	Offset
100	0	11

Cache Hits	Cache Misses
0	1

History:

```
R(0x23) = M
>
```

Load || ↑ ↓

Simulation Messages:

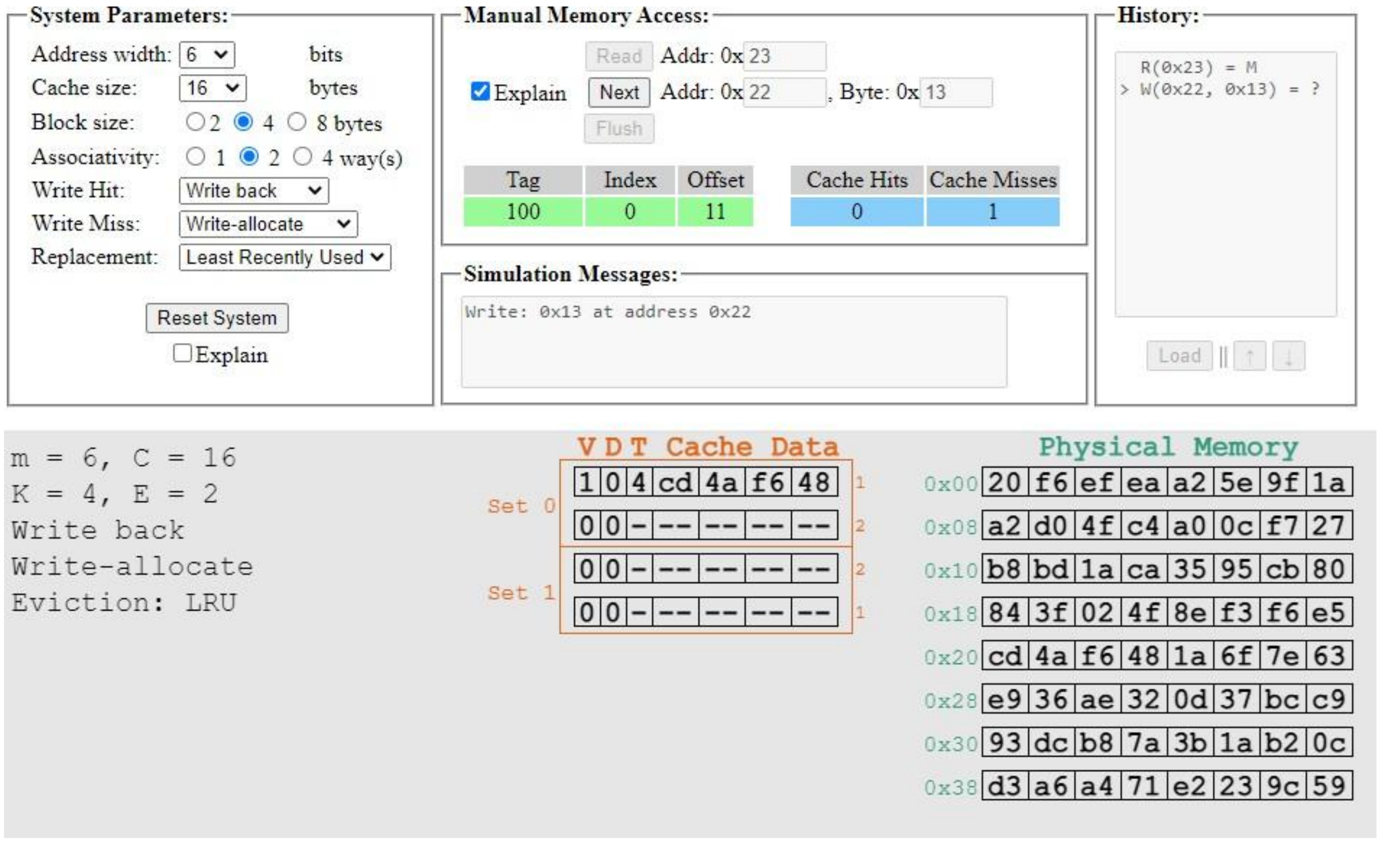
```
0x20.
LRU statuses updated.
Data: 0x48
```

```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

	V	D	T	Cache Data				
Set 0	1	0	4	cd	4a	f6	48	1
	0	0	-	--	--	--	--	2
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory								
0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59

Cache Simulator: Writing 0x13 at 0x22



System Parameters:

Address width:	6	bits
Cache size:	16	bytes
Block size:	○2 ●4 ○8	bytes
Associativity:	○1 ●2 ○4	way(s)
Write Hit:	Write back	
Write Miss:	Write-allocate	
Replacement:	Least Recently Used	

Reset System
☐Explain

Manual Memory Access:

Read Addr: 0x 23
☑Explain Next Addr: 0x 22 , Byte: 0x 13
Flush

Tag	Index	Offset
100	0	11

Cache Hits	Cache Misses
0	1

Simulation Messages:

```
Write: 0x13 at address 0x22
```

History:

```
  R(0x23) = M
> W(0x22, 0x13) = ?
```

Load || ↑ ↓

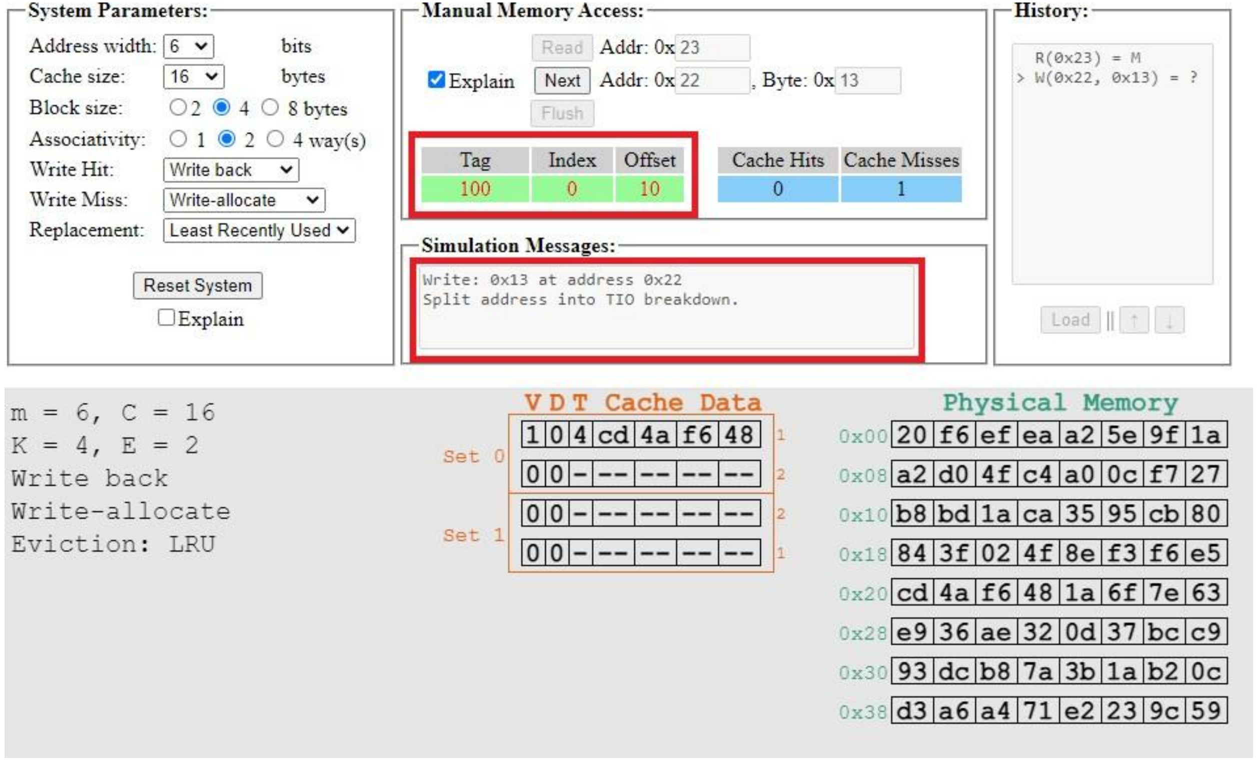
```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

V D T Cache Data

	V	D	T					
Set 0	1	0	4	cd	4a	f6	48	1
	0	0	-	--	--	--	--	2
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59



System Parameters:

Address width:	6	bits
Cache size:	16	bytes
Block size:	○ 2 ● 4 ○ 8 bytes	
Associativity:	○ 1 ● 2 ○ 4 way(s)	
Write Hit:	Write back	
Write Miss:	Write-allocate	
Replacement:	Least Recently Used	

Manual Memory Access:

☑ Explain

Addr: 0x 23

Addr: 0x 22, Byte: 0x 13

Tag	Index	Offset
100	0	10

Cache Hits	Cache Misses
0	1

Simulation Messages:

```
Write: 0x13 at address 0x22
Split address into TIO breakdown.
```

History:

```
  R(0x23) = M
> W(0x22, 0x13) = ?
```

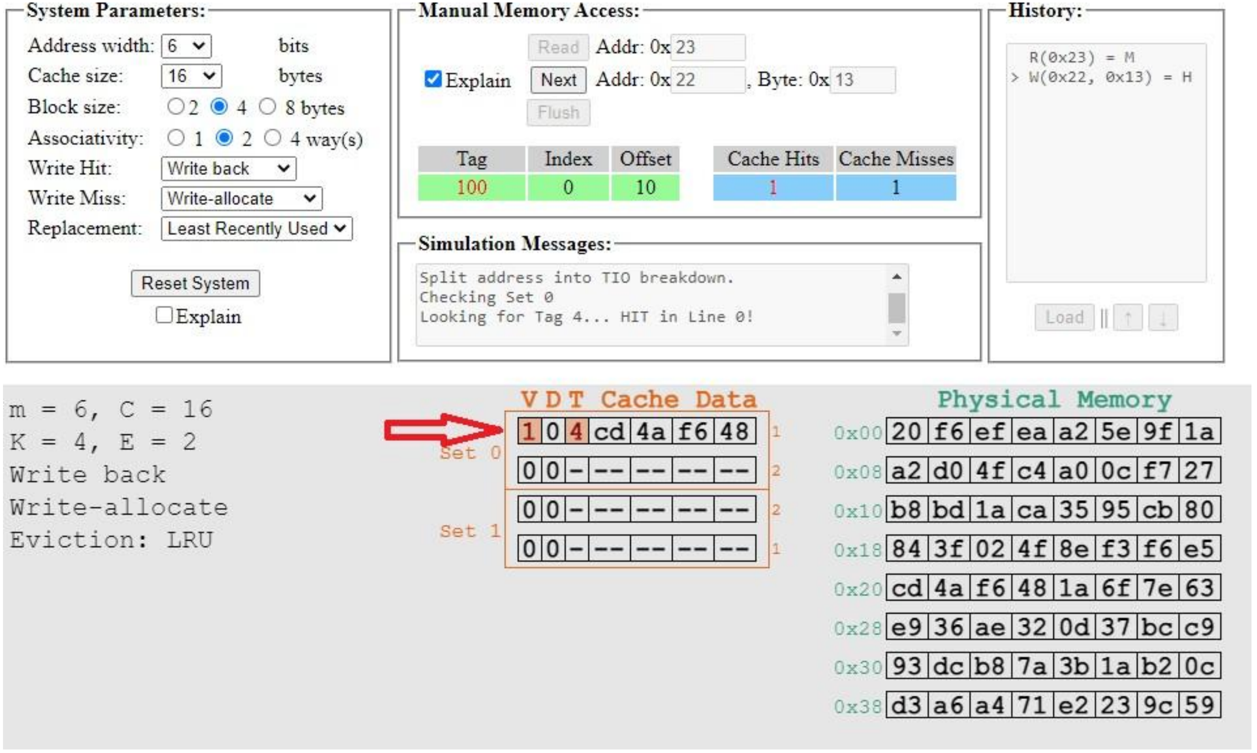
```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

V D T Cache Data

	V	D	T					
Set 0	1	0	4	cd	4a	f6	48	1
	0	0	-	--	--	--	--	2
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59



System Parameters:

Address width: 6 bits
Cache size: 16 bytes
Block size: ○ 2 [✓ ● 4] ○ 8 bytes
Associativity: ○ 1 [✓ ● 2] ○ 4 way(s)
Write Hit: Write back
Write Miss: Write-allocate
Replacement: Least Recently Used

Reset System
☐ Explain

Manual Memory Access:

Read Addr: 0x 23
☑ Explain Next Addr: 0x 22 , Byte: 0x 13
Flush

Tag	Index	Offset		Cache Hits	Cache Misses
100	0	10		1	1

Simulation Messages:

```
Split address into TIO breakdown.
Checking Set 0
Looking for Tag 4... HIT in Line 0!
```

History:

```
  R(0x23) = M
> W(0x22, 0x13) = H
```

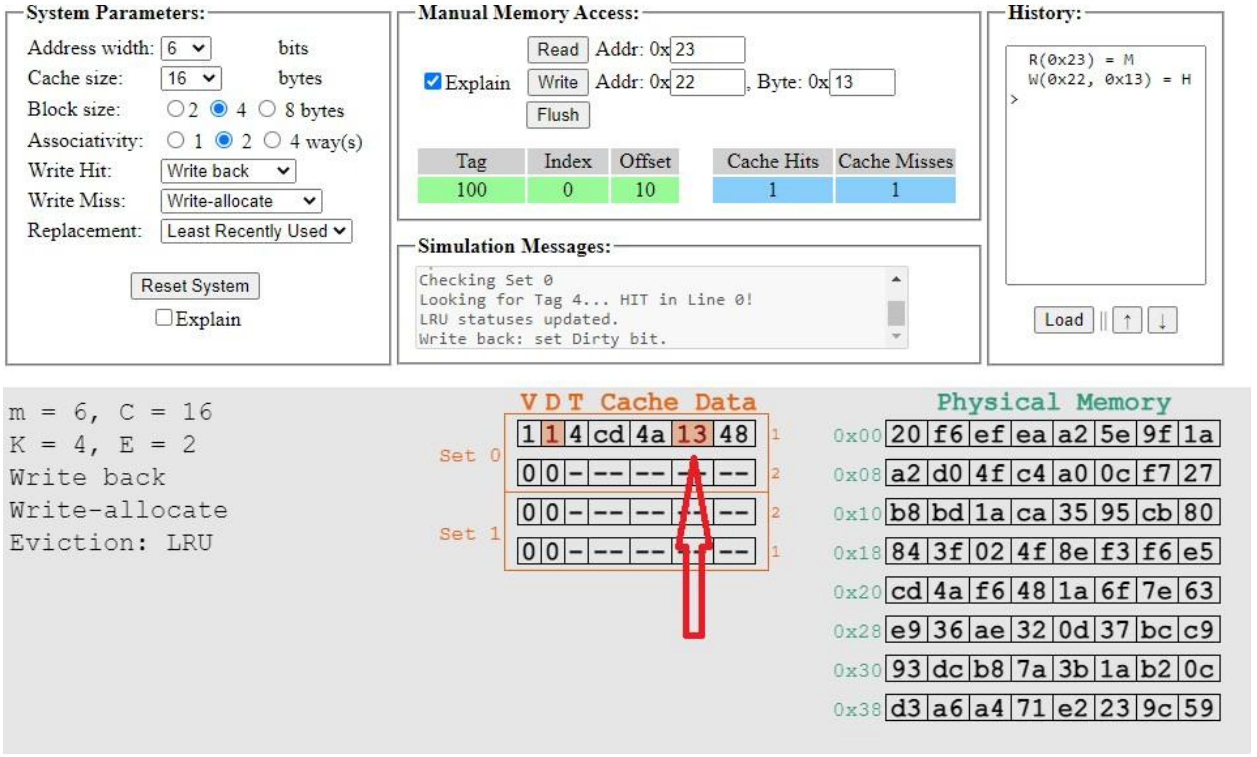
Load || ↑ ↓

```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

	V	D	T	Cache Data				
Set 0	1	0	4	cd	4a	f6	48	1
	0	0	-	--	--	--	--	2
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59



System Parameters:

Address width:	6	bits
Cache size:	16	bytes
Block size:	○ 2 ● 4 ○ 8 bytes	
Associativity:	○ 1 ● 2 ○ 4 way(s)	
Write Hit:	Write back	
Write Miss:	Write-allocate	
Replacement:	Least Recently Used	

Reset System

☐ Explain

Manual Memory Access:

☑ Explain

Read Addr: 0x 23

Write Addr: 0x 22 , Byte: 0x 13

Flush

Tag	Index	Offset
100	0	10

Cache Hits	Cache Misses
1	1

Simulation Messages:

```
Checking Set 0
Looking for Tag 4... HIT in Line 0!
LRU statuses updated.
Write back: set Dirty bit.
```

History:

```
R(0x23) = M
W(0x22, 0x13) = H
>
```

Load || ↑ ↓

```
m = 6, C = 16
K = 4, E = 2
Write back
Write-allocate
Eviction: LRU
```

	V	D	T	Cache	Data			
Set 0	1	1	4	cd	4a	13	48	1
	0	0	-	--	--	--	--	2
Set 1	0	0	-	--	--	--	--	2
	0	0	-	--	--	--	--	1

Physical Memory

0x00	20	f6	ef	ea	a2	5e	9f	1a
0x08	a2	d0	4f	c4	a0	0c	f7	27
0x10	b8	bd	1a	ca	35	95	cb	80
0x18	84	3f	02	4f	8e	f3	f6	e5
0x20	cd	4a	f6	48	1a	6f	7e	63
0x28	e9	36	ae	32	0d	37	bc	c9
0x30	93	dc	b8	7a	3b	1a	b2	0c
0x38	d3	a6	a4	71	e2	23	9c	59

Recall: General Caching Concepts: 3 Types of Cache Misses

- Cold (compulsory) miss
    - Cold misses occur because the cache starts empty and this is the first reference to the block.
- Capacity miss
    - Occurs when the set of active cache blocks (working set) is larger than the cache.
- Conflict miss
    - Most catches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
        - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
    - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
        - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

KOÇ UNIVERSITY

Cache Example #3: Effective Access Time

Find the EAT for a system with the following properties:

- Cache access time: 10 ns
- Cache miss rate: 1%
- Main Memory access time: 200 ns

$$EAT = T_{cache} + (1\text{-Hit Rate}) * T_{Memory}$$

$$= 10 + 0.01 * 200$$

$$= 10 + 2$$

$$= 12 \text{ ns}$$

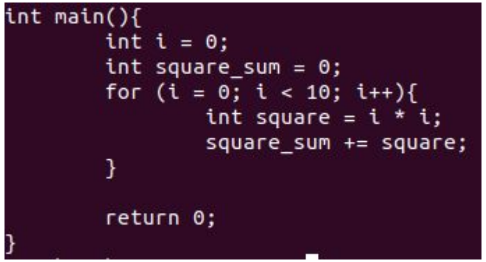
KOÇ UNIVERSITY

Locality in Programs

Principle of Locality:

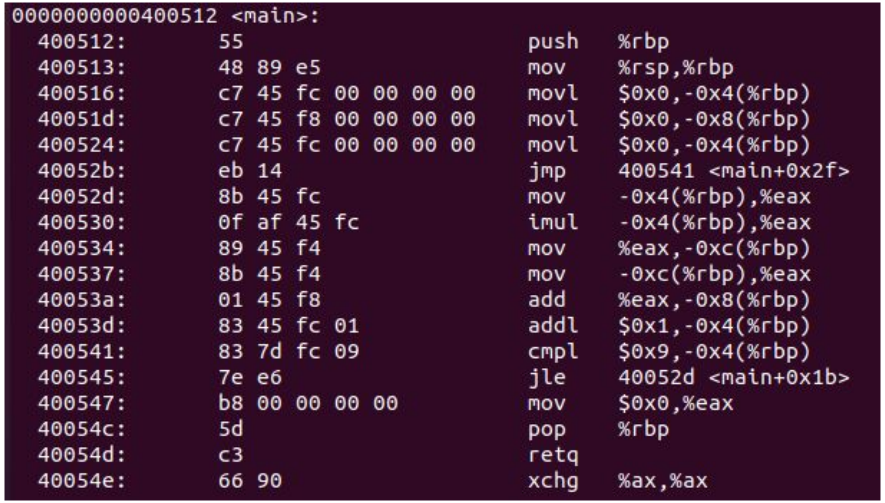
- Programs tend to use data and instructions with addresses near or equal to those they have used recently.

- **Temporal locality:**
  - Recently referenced items are likely be referenced in the near future.
- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time.



```
int main(){
        int i = 0;
        int square_sum = 0;
        for (i = 0; i < 10; i++){
                int square = i * i;
                square_sum += square;
        }

        return 0;
}
```



```
0000000000400512 <main>:
  400512:       55                      push   %rbp
  400513:       48 89 e5                mov    %rsp,%rbp
  400516:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  40051d:       c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
  400524:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  40052b:       eb 14                   jmp    400541 <main+0x2f>
  40052d:       8b 45 fc                mov    -0x4(%rbp),%eax
  400530:       0f af 45 fc             imul   -0x4(%rbp),%eax
  400534:       89 45 f4                mov    %eax,-0xc(%rbp)
  400537:       8b 45 f4                mov    -0xc(%rbp),%eax
  40053a:       01 45 f8                add    %eax,-0x8(%rbp)
  40053d:       83 45 fc 01             addl   $0x1,-0x4(%rbp)
  400541:       83 7d fc 09             cmpl   $0x9,-0x4(%rbp)
  400545:       7e e6                   jle    40052d <main+0x1b>
  400547:       b8 00 00 00 00          mov    $0x0,%eax
  40054c:       5d                      pop    %rbp
  40054d:       c3                      retq
  40054e:       66 90                   xchg   %ax,%ax
```

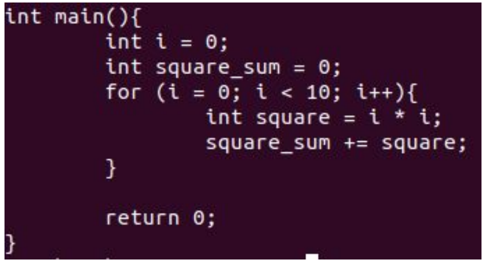
Temporal or Spatial Locality?

Locality in Programs

Principle of Locality:

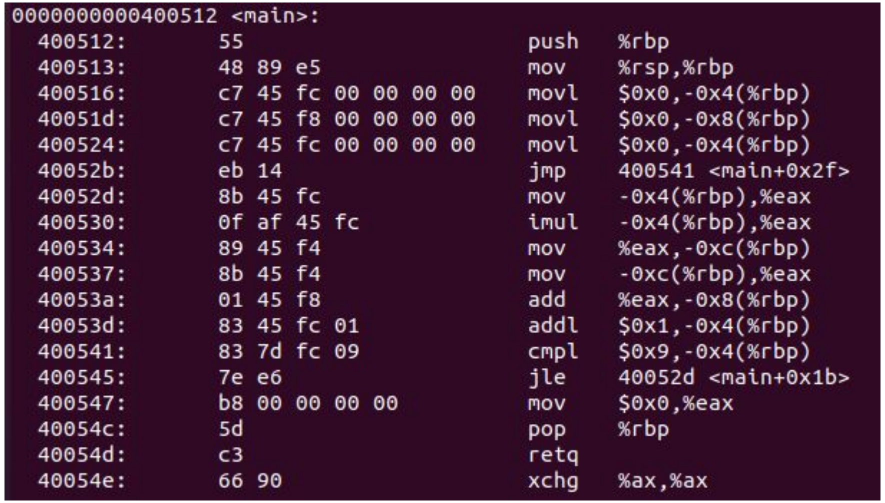
- Programs tend to use data and instructions with addresses near or equal to those they have used recently.

- **Temporal locality:**
  - Recently referenced items are likely be referenced in the near future.
- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time.



```
int main(){
        int i = 0;
        int square_sum = 0;
        for (i = 0; i < 10; i++){
                int square = i * i;
                square_sum += square;
        }

        return 0;
}
```



```
0000000000400512 <main>:
  400512:       55                      push   %rbp
  400513:       48 89 e5                mov    %rsp,%rbp
  400516:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  40051d:       c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
  400524:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  40052b:       eb 14                   jmp    400541 <main+0x2f>
  40052d:       8b 45 fc                mov    -0x4(%rbp),%eax
  400530:       0f af 45 fc             imul   -0x4(%rbp),%eax
  400534:       89 45 f4                mov    %eax,-0xc(%rbp)
  400537:       8b 45 f4                mov    -0xc(%rbp),%eax
  40053a:       01 45 f8                add    %eax,-0x8(%rbp)
  40053d:       83 45 fc 01             addl   $0x1,-0x4(%rbp)
  400541:       83 7d fc 09             cmpl   $0x9,-0x4(%rbp)
  400545:       7e e6                   jle    40052d <main+0x1b>
  400547:       b8 00 00 00 00          mov    $0x0,%eax
  40054c:       5d                      pop    %rbp
  40054d:       c3                      retq
  40054e:       66 90                   xchg   %ax,%ax
```

Temporal or Spatial Locality?

Both!

Recall: Spatial Locality in Arrays

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

(a)

Address	0	4	8	12	16	20
Contents	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Access order	1	3	5	2	4	6

(b)

Good Locality?

No! (Stride-N pattern)

Recall: Spatial Locality in Arrays

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

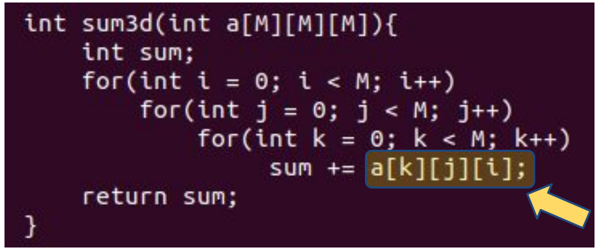
(a)

Address	0	4	8	12	16	20
Contents	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Access order	1	2	3	4	5	6

(b)

Good Locality?

Recall: Spatial Locality in Arrays

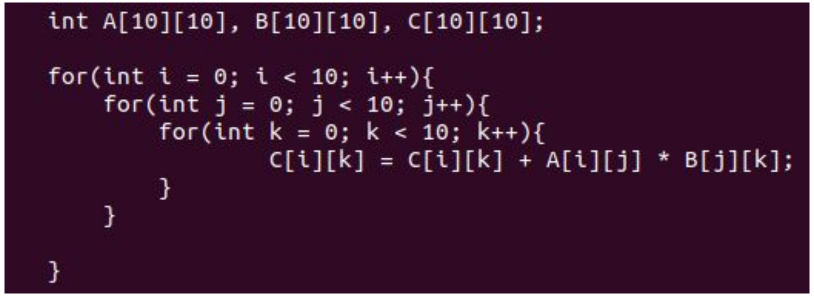


```
int sum3d(int a[M][M][M]){
    int sum;
    for(int i = 0; i < M; i++)
        for(int j = 0; j < M; j++)
            for(int k = 0; k < M; k++)
                sum += a[k][j][i];
    return sum;
}
```

Good Locality?

No!

Locality in Data



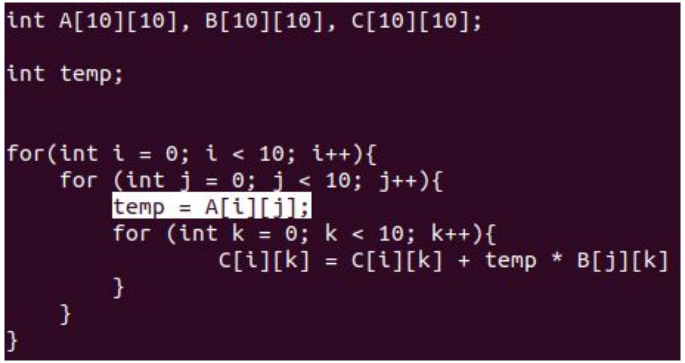
```
int A[10][10], B[10][10], C[10][10];

for(int i = 0; i < 10; i++){
    for(int j = 0; j < 10; j++){
        for(int k = 0; k < 10; k++){
                C[i][k] = C[i][k] + A[i][j] * B[j][k];
        }
    }

}
```

Good Locality?

Locality in Data



```
int A[10][10], B[10][10], C[10][10];

int temp;


for(int i = 0; i < 10; i++){
    for (int j = 0; j < 10; j++){
        temp = A[i][j];
        for (int k = 0; k < 10; k++){
                C[i][k] = C[i][k] + temp * B[j][k]
        }
    }
}
```

How about this one?

Concluding Observations

Programmer can optimize for cache performance

- How data structures are organized
- How data are accessed
  - Nested loop structure
  - Blocking is a general technique

All systems favor "cache friendly code"

- Getting absolute optimum performance is very platform specific
  - Cache sizes, line sizes, associatives, etc.
- Can get most of the advantage with generic code
  - Keep working set reasonably small (**temporal locality**)
  - Use small strides (**spatial locality**)

KOÇ UNIVERSITY

Callgrind

https://valgrind.org/

Code Profiling

- A **code profiler** is a tool to analyze a program and report on its resource usage
    - "resource" could be memory, CPU cycles, network bandwidth, and so on
- The program is run under control of a profiling tool
- During application development, a common step is to improve runtime performance using profiling tools.
- To not waste time on optimizing functions which are rarely used, one needs to know in which parts of the program most of the time is spent.
- Some example:
    - Callgrind, GProf, JConsol, CLR

KOÇ UNIVERSITY

Valgrind

the Valgrind framework supports a variety of runtime analysis tools

- memcheck
  - detects memory errors/leaks
- massif
  - reports on heap usage
- helgrind
  - detects multithreaded race conditions
- callgrind/cachegrind
  - profiles CPU/cache performance

KOÇ UNIVERSITY

Callgrind/cachegrind

- The Valgrind profiling tools are cachegrind and callgrind
- The cachegrind tool simulates the L1/L2 caches and counts cache misses/hits.
- The callgrind tool counts function calls and the CPU instructions executed within each call and builds a function callgraph
- The callgrind tool includes a cache simulation feature adopted from cachegrind, so you can actually use **callgrind for both CPU and cache profiling**.

KOÇ UNIVERSITY

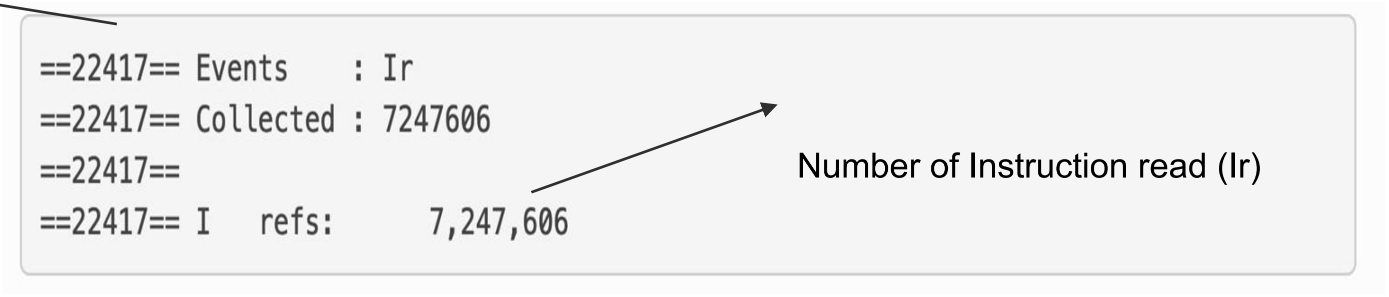
Basic Usage of Callgrind

- First, we need to compile our program with debugging enabled
  - `gcc -g -ggdb name.c -o name.out`
- You first need to run your program under Valgrind and explicitly request the callgrind tool (if unspecified, the tool defaults to memcheck)

```
  valgrind --tool=callgrind [possible options] name.out
program-arguments
```

- The result will be stored on the files callgrind.out.PID, where PID will be the process identifier.

Process identifier



```
==22417== Events    : Ir
==22417== Collected : 7247606
==22417==
==22417== I   refs:      7,247,606
```

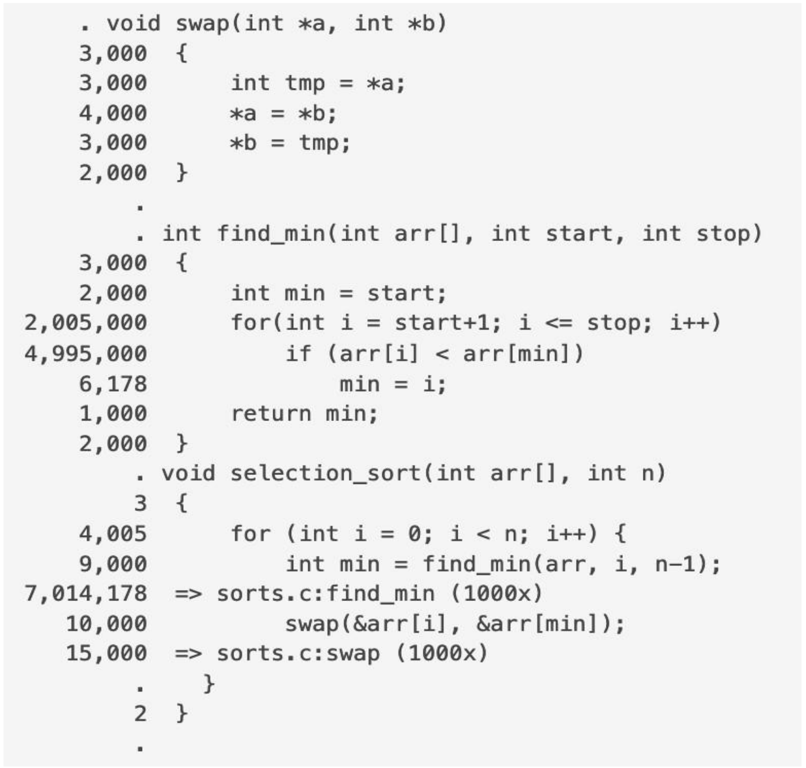
Number of Instruction read (Ir)

Basic Usage of Callgrind

Counting instructions with callgrind

- The callgrind output file is a text file, but its contents are not intended for you to read yourself.
- You can properly read the output using **`callgrind_annotate`**
  - **`callgrind_annotate --auto=yes callgrind.out.PID`**
- The **`--auto=yes`** option report counts for each C statement
- Do not forget to replace `PID` by the actual number.

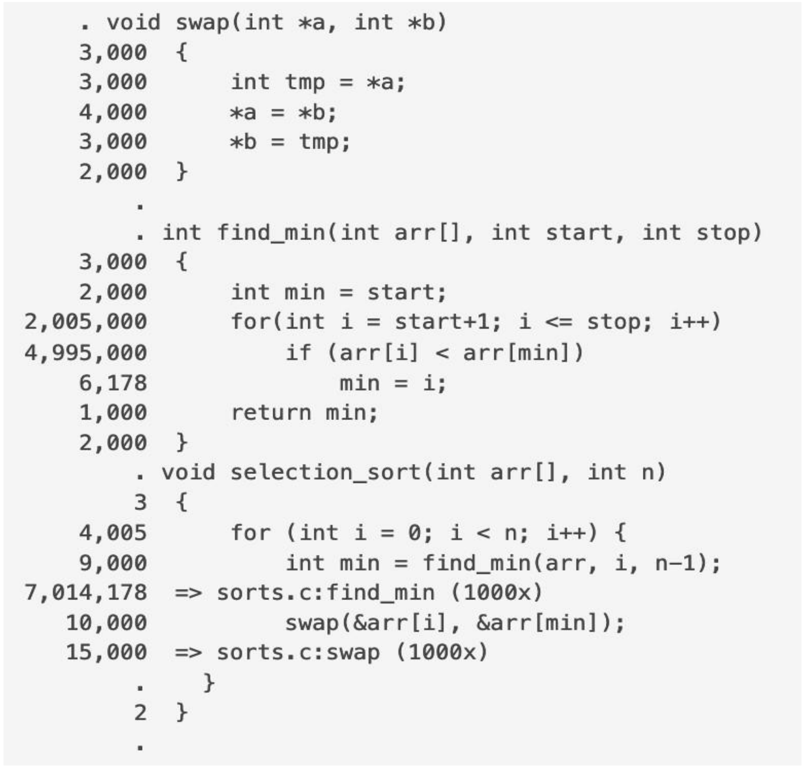
Sorts a 1000-member array using selection sort



```
        . void swap(int *a, int *b)
    3,000  {
    3,000      int tmp = *a;
    4,000      *a = *b;
    3,000      *b = tmp;
    2,000  }
        .
        . int find_min(int arr[], int start, int stop)
    3,000  {
    2,000      int min = start;
2,005,000      for(int i = start+1; i <= stop; i++)
4,995,000          if (arr[i] < arr[min])
    6,178              min = i;
    1,000      return min;
    2,000  }
        . void selection_sort(int arr[], int n)
        3  {
    4,005      for (int i = 0; i < n; i++) {
    9,000          int min = find_min(arr, i, n-1);
7,014,178  => sorts.c:find_min (1000x)
   10,000          swap(&arr[i], &arr[min]);
   15,000  => sorts.c:swap (1000x)
        .      }
        2  }
        .
```

Interpreting the results

- The Ir counts are basically the count of assembly instructions executed.
- By default, the counts are *exclusive*
    - The counts for a function include only the time spent in that function and not in the functions that it calls.
- By using exclusive counts you can detect the bottlenecks.
- Here, the work is concentrated in the loop to find the min value



```
          . void swap(int *a, int *b)
      3,000  {
      3,000      int tmp = *a;
      4,000      *a = *b;
      3,000      *b = tmp;
      2,000  }
          .
          . int find_min(int arr[], int start, int stop)
      3,000  {
      2,000      int min = start;
  2,005,000      for(int i = start+1; i <= stop; i++)
  4,995,000          if (arr[i] < arr[min])
      6,178              min = i;
      1,000      return min;
      2,000  }
          . void selection_sort(int arr[], int n)
          3  {
      4,005      for (int i = 0; i < n; i++) {
      9,000          int min = find_min(arr, i, n-1);
  7,014,178  => sorts.c:find_min (1000x)
     10,000          swap(&arr[i], &arr[min]);
     15,000  => sorts.c:swap (1000x)
          .      }
          2  }
          .
```

Basic Usage of Callgrind

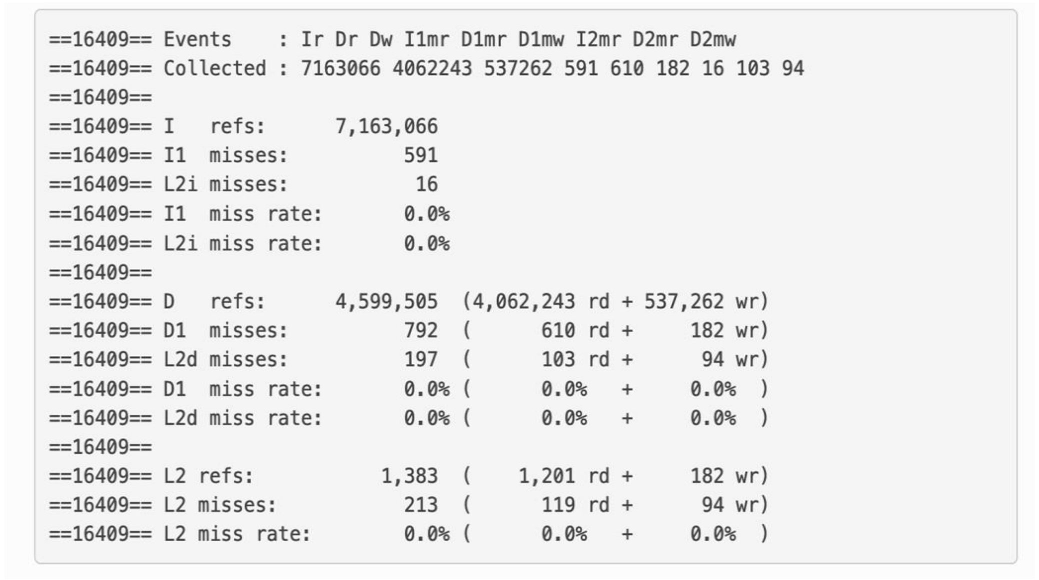
Adding in cache simulation

- Invoke valgrind by `--simulate-cache=yes`

```
valgrind --tool=callgrind --simulate-cache=yes name.out args
```

- The cache simulator models a machine with a split L1 cache (separate instruction I1 and data D1), backed by a unified second-level cache (L2).
- Similar to the previous example, callgrind_annotate should be used to interpret the output.

Callgrind Example



```
==16409== Events    : Ir Dr Dw I1mr D1mr D1mw I2mr D2mr D2mw
==16409== Collected : 7163066 4062243 537262 591 610 182 16 103 94
==16409==
==16409== I   refs:      7,163,066
==16409== I1  misses:          591
==16409== L2i misses:           16
==16409== I1  miss rate:       0.0%
==16409== L2i miss rate:       0.0%
==16409==
==16409== D   refs:      4,599,505  (4,062,243 rd + 537,262 wr)
==16409== D1  misses:          792  (      610 rd +     182 wr)
==16409== L2d misses:          197  (      103 rd +      94 wr)
==16409== D1  miss rate:       0.0% (      0.0%   +     0.0%  )
==16409== L2d miss rate:       0.0% (      0.0%   +     0.0%  )
==16409==
==16409== L2 refs:           1,383  (    1,201 rd +     182 wr)
==16409== L2 misses:           213  (      119 rd +      94 wr)
==16409== L2 miss rate:        0.0% (      0.0%   +     0.0%  )
```

Ir: I cache reads (instructions executed)

I1mr: I1 cache read misses (instruction wasn't in I1 cache but was in L2)

I2mr: L2 cache instruction read misses (instruction wasn't in I1 or L2 cache, had to be fetched

Dr: D cache reads (memory reads)

D1mr: D1 cache read misses (data location not in D1 cache, but in L2)
D2mr: L2 cache data read misses (location not in D1 or

L2) Dw: D cache writes (memory writes)

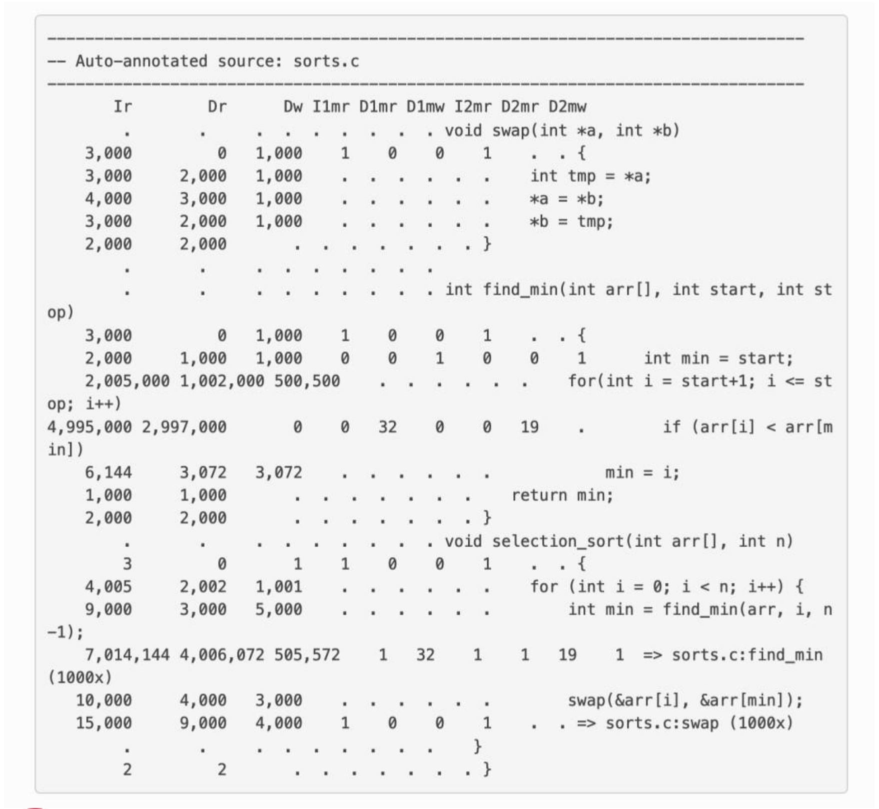
D1mw: D1 cache write misses (location not in D1 cache, but in L2)

D2mw: L2 cache data write misses (location not in D1 or L2)

It sounds like we have a cache friendly code.

KOÇ UNIVERSITY

Callgrind Example



```
--------------------------------------------------------------------------------
-- Auto-annotated source: sorts.c
--------------------------------------------------------------------------------
       Ir        Dr      Dw I1mr D1mr D1mw I2mr D2mr D2mw
        .         .       .    .    .    .    .    .    . void swap(int *a, int *b)
    3,000         0   1,000    1    0    0    1    .    . {
    3,000     2,000   1,000    .    .    .    .    .    .     int tmp = *a;
    4,000     3,000   1,000    .    .    .    .    .    .     *a = *b;
    3,000     2,000   1,000    .    .    .    .    .    .     *b = tmp;
    2,000     2,000       .    .    .    .    .    .    . }
        .         .       .    .    .    .    .    .    .
        .         .       .    .    .    .    .    .    . int find_min(int arr[], int start, int st
op)
    3,000         0   1,000    1    0    0    1    .    . {
    2,000     1,000   1,000    0    0    1    0    0    1     int min = start;
    2,005,000 1,002,000 500,500    .    .    .    .    .    .     for(int i = start+1; i <= st
op; i++)
4,995,000 2,997,000       0    0   32    0    0   19    .         if (arr[i] < arr[m
in])
    6,144     3,072   3,072    .    .    .    .    .    .             min = i;
    1,000     1,000       .    .    .    .    .    .    .     return min;
    2,000     2,000       .    .    .    .    .    .    . }
        .         .       .    .    .    .    .    .    . void selection_sort(int arr[], int n)
        3         0       1    1    0    0    1    .    . {
    4,005     2,002   1,001    .    .    .    .    .    .     for (int i = 0; i < n; i++) {
    9,000     3,000   5,000    .    .    .    .    .    .         int min = find_min(arr, i, n
-1);
    7,014,144 4,006,072 505,572    1   32    1    1   19    1  => sorts.c:find_min
(1000x)
   10,000     4,000   3,000    .    .    .    .    .    .         swap(&arr[i], &arr[min]);
   15,000     9,000   4,000    1    0    0    1    .    .  => sorts.c:swap (1000x)
        .         .       .    .    .    .    .    .    .     }
        2         2       .    .    .    .    .    .    . }
```

Ir: I cache reads (instructions executed)

I1mr: I1 cache read misses (instruction wasn't in I1 cache but was in L2)

I2mr: L2 cache instruction read misses (instruction wasn't in I1 or L2 cache, had to be fetched

Dr: D cache reads (memory reads)

D1mr: D1 cache read misses (data location not in D1 cache, but in L2)
D2mr: L2 cache data read misses (location not in D1 or

L2) Dw: D cache writes (memory writes)

D1mw: D1 cache write misses (location not in D1 cache, but in
L2)

D2mw: L2 cache data write misses (location not in D1 or L2)

Additional Points

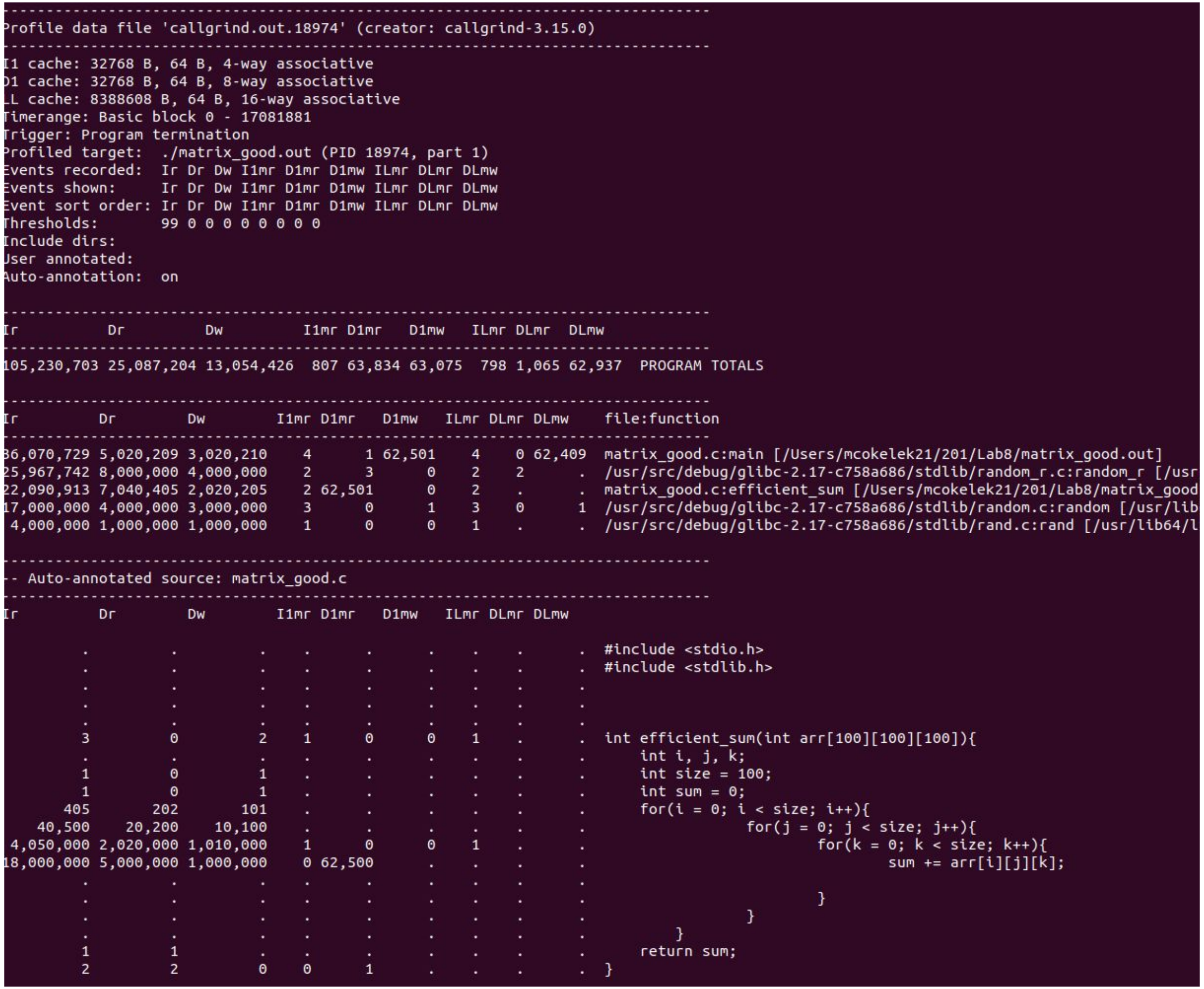
- L2 misses are much more expensive than L1 misses, so pay attention to passages with high **D2mr** or **D2mw** counts.

- Even a small number of misses can be quite important, as a L1 miss will typically cost around 5-10 cycles, an L2 miss can cost as much as 100-200 cycles

- Callgrind cannot detect the bottleneck of your program if it is related to file I/O

- Try to examine different paths of your program

KOÇ UNIVERSITY



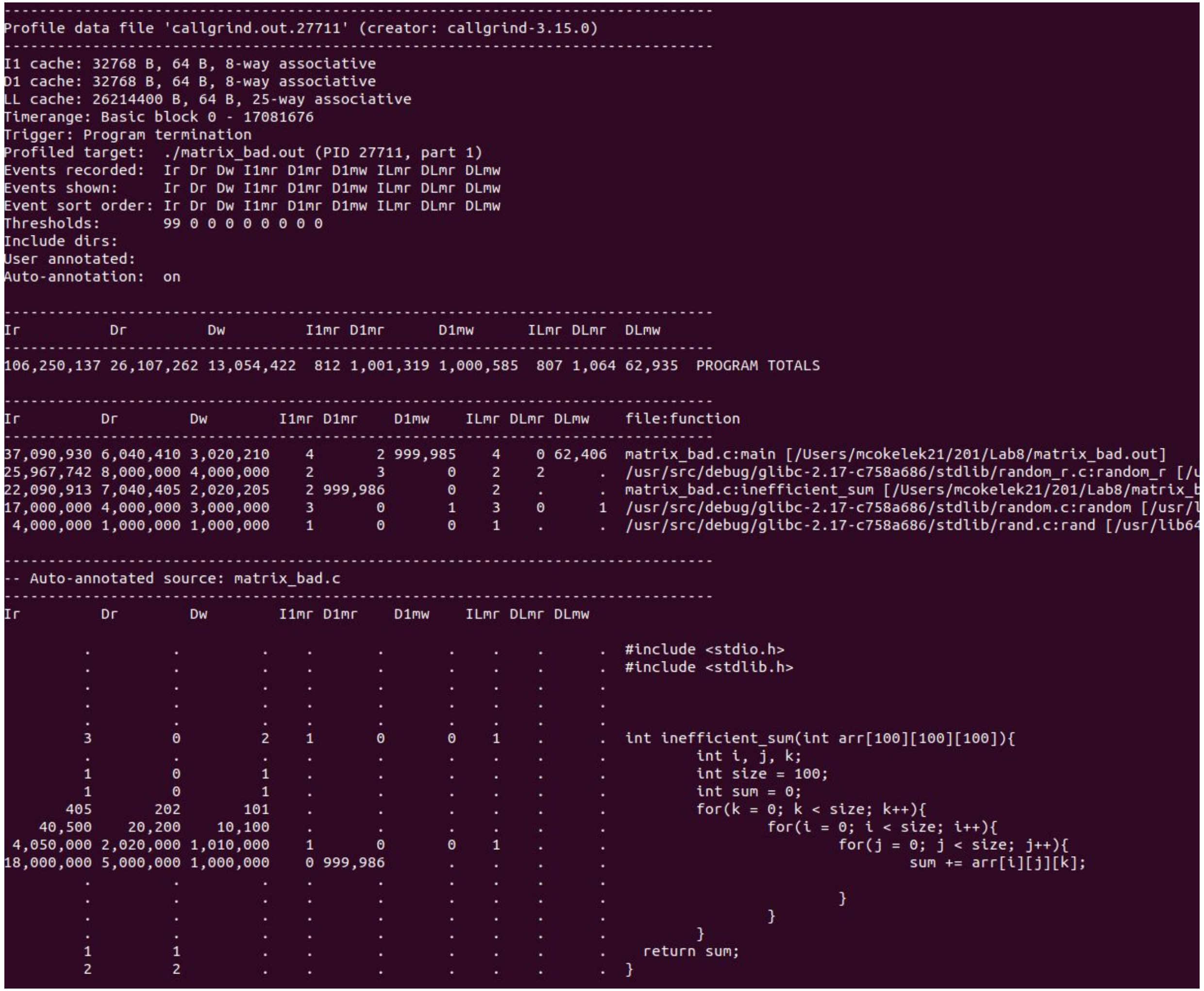
```
--------------------------------------------------------------------------------
Profile data file 'callgrind.out.18974' (creator: callgrind-3.15.0)
--------------------------------------------------------------------------------
I1 cache: 32768 B, 64 B, 4-way associative
D1 cache: 32768 B, 64 B, 8-way associative
LL cache: 8388608 B, 64 B, 16-way associative
Timerange: Basic block 0 - 17081881
Trigger: Program termination
Profiled target:  ./matrix_good.out (PID 18974, part 1)
Events recorded:  Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLmw
Events shown:     Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLmw
Event sort order: Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLmw
Thresholds:       99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:
Auto-annotation:  on

--------------------------------------------------------------------------------
Ir          Dr          Dw          I1mr D1mr   D1mw   ILmr DLmr  DLmw
--------------------------------------------------------------------------------
105,230,703 25,087,204 13,054,426  807 63,834 63,075  798 1,065 62,937  PROGRAM TOTALS

--------------------------------------------------------------------------------
Ir         Dr        Dw        I1mr D1mr   D1mw   ILmr DLmr DLmw    file:function
--------------------------------------------------------------------------------
36,070,729 5,020,209 3,020,210    4      1 62,501    4    0 62,409  matrix_good.c:main [/Users/mcokelek21/201/Lab8/matrix_good.out]
25,967,742 8,000,000 4,000,000    2      3      0    2    2      .  /usr/src/debug/glibc-2.17-c758a686/stdlib/random_r.c:random_r [/usr
22,090,913 7,040,405 2,020,205    2 62,501      0    2    .      .  matrix_good.c:efficient_sum [/Users/mcokelek21/201/Lab8/matrix_good
17,000,000 4,000,000 3,000,000    3      0      1    3    0      1  /usr/src/debug/glibc-2.17-c758a686/stdlib/random.c:random [/usr/lib
 4,000,000 1,000,000 1,000,000    1      0      0    1    .      .  /usr/src/debug/glibc-2.17-c758a686/stdlib/rand.c:rand [/usr/lib64/l

--------------------------------------------------------------------------------
-- Auto-annotated source: matrix_good.c
--------------------------------------------------------------------------------
Ir         Dr        Dw        I1mr D1mr   D1mw   ILmr DLmr DLmw

         .         .         .    .      .      .    .    .      .  #include <stdio.h>
         .         .         .    .      .      .    .    .      .  #include <stdlib.h>
         .         .         .    .      .      .    .    .      .
         .         .         .    .      .      .    .    .      .
         .         .         .    .      .      .    .    .      .
         3         0         2    1      0      0    1    .      .  int efficient_sum(int arr[100][100][100]){
         .         .         .    .      .      .    .    .      .      int i, j, k;
         1         0         1    .      .      .    .    .      .      int size = 100;
         1         0         1    .      .      .    .    .      .      int sum = 0;
       405       202       101    .      .      .    .    .      .      for(i = 0; i < size; i++){
    40,500    20,200    10,100    .      .      .    .    .      .                  for(j = 0; j < size; j++){
 4,050,000 2,020,000 1,010,000    1      0      0    1    .      .                          for(k = 0; k < size; k++){
18,000,000 5,000,000 1,000,000    0 62,500      .    .    .      .                                  sum += arr[i][j][k];
         .         .         .    .      .      .    .    .      .
         .         .         .    .      .      .    .    .      .                          }
         .         .         .    .      .      .    .    .      .                  }
         .         .         .    .      .      .    .    .      .          }
         1         1         .    .      .      .    .    .      .      return sum;
         2         2         0    0      1      .    .    .      .  }
```



```
--------------------------------------------------------------------------------
Profile data file 'callgrind.out.27711' (creator: callgrind-3.15.0)
--------------------------------------------------------------------------------
I1 cache: 32768 B, 64 B, 8-way associative
D1 cache: 32768 B, 64 B, 8-way associative
LL cache: 26214400 B, 64 B, 25-way associative
Timerange: Basic block 0 - 17081676
Trigger: Program termination
Profiled target:  ./matrix_bad.out (PID 27711, part 1)
Events recorded:  Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLmw
Events shown:     Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLmw
Event sort order: Ir Dr Dw I1mr D1mr D1mw ILmr DLmr DLmw
Thresholds:       99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:
Auto-annotation:  on

--------------------------------------------------------------------------------
Ir          Dr         Dw         I1mr D1mr      D1mw      ILmr DLmr  DLmw
--------------------------------------------------------------------------------
106,250,137 26,107,262 13,054,422  812 1,001,319 1,000,585  807 1,064 62,935  PROGRAM TOTALS

--------------------------------------------------------------------------------
Ir         Dr         Dw         I1mr D1mr    D1mw    ILmr DLmr DLmw    file:function
--------------------------------------------------------------------------------
37,090,930 6,040,410 3,020,210    4       2 999,985    4    0 62,406  matrix_bad.c:main [/Users/mcokelek21/201/Lab8/matrix_bad.out]
25,967,742 8,000,000 4,000,000    2       3       0    2    2      .  /usr/src/debug/glibc-2.17-c758a686/stdlib/random_r.c:random_r [/u
22,090,913 7,040,405 2,020,205    2 999,986       0    2    .      .  matrix_bad.c:inefficient_sum [/Users/mcokelek21/201/Lab8/matrix_b
17,000,000 4,000,000 3,000,000    3       0       1    3    0      1  /usr/src/debug/glibc-2.17-c758a686/stdlib/random.c:random [/usr/l
 4,000,000 1,000,000 1,000,000    1       0       0    1    .      .  /usr/src/debug/glibc-2.17-c758a686/stdlib/rand.c:rand [/usr/lib64

--------------------------------------------------------------------------------
-- Auto-annotated source: matrix_bad.c
--------------------------------------------------------------------------------
Ir         Dr         Dw         I1mr D1mr    D1mw    ILmr DLmr DLmw

        .         .         .    .       .       .    .    .      .  #include <stdio.h>
        .         .         .    .       .       .    .    .      .  #include <stdlib.h>
        .         .         .    .       .       .    .    .      .
        .         .         .    .       .       .    .    .      .
        .         .         .    .       .       .    .    .      .
        3         0         2    1       0       0    1    .      .  int inefficient_sum(int arr[100][100][100]){
        .         .         .    .       .       .    .    .      .          int i, j, k;
        1         0         1    .       .       .    .    .      .          int size = 100;
        1         0         1    .       .       .    .    .      .          int sum = 0;
      405       202       101    .       .       .    .    .      .          for(k = 0; k < size; k++){
   40,500    20,200    10,100    .       .       .    .    .      .                  for(i = 0; i < size; i++){
 4,050,000 2,020,000 1,010,000    1       0       0    1    .      .                          for(j = 0; j < size; j++){
18,000,000 5,000,000 1,000,000    0 999,986       .    .    .      .                                  sum += arr[i][j][k];
        .         .         .    .       .       .    .    .      .
        .         .         .    .       .       .    .    .      .                          }
        .         .         .    .       .       .    .    .      .                  }
        .         .         .    .       .       .    .    .      .          }
        1         1         .    .       .       .    .    .      .    return sum;
        2         2         .    .       .       .    .    .      .  }
```

References

1. Some of the slides are borrowed from materials in Stanford CS107, CMU15-213 and CS201, Portland State University
2. https://stackoverflow.com/questions/16699247/what-is-a-cache-friendly-code
3. https://www.valgrind.org/docs/manual/manual.html
4. The Cache Simulator and its demos are borrowed from materials in University of Washington,
   CSE 351

KOÇ UNIVERSITY