

Memory

Course: Systems Architecture

Lecturer: Gleb Lobanov

April 1, 2024



EPIC

Institute of Technology
Powered by epam

Contents

05 Caches

06 RAM

07 SSD

08 HDD

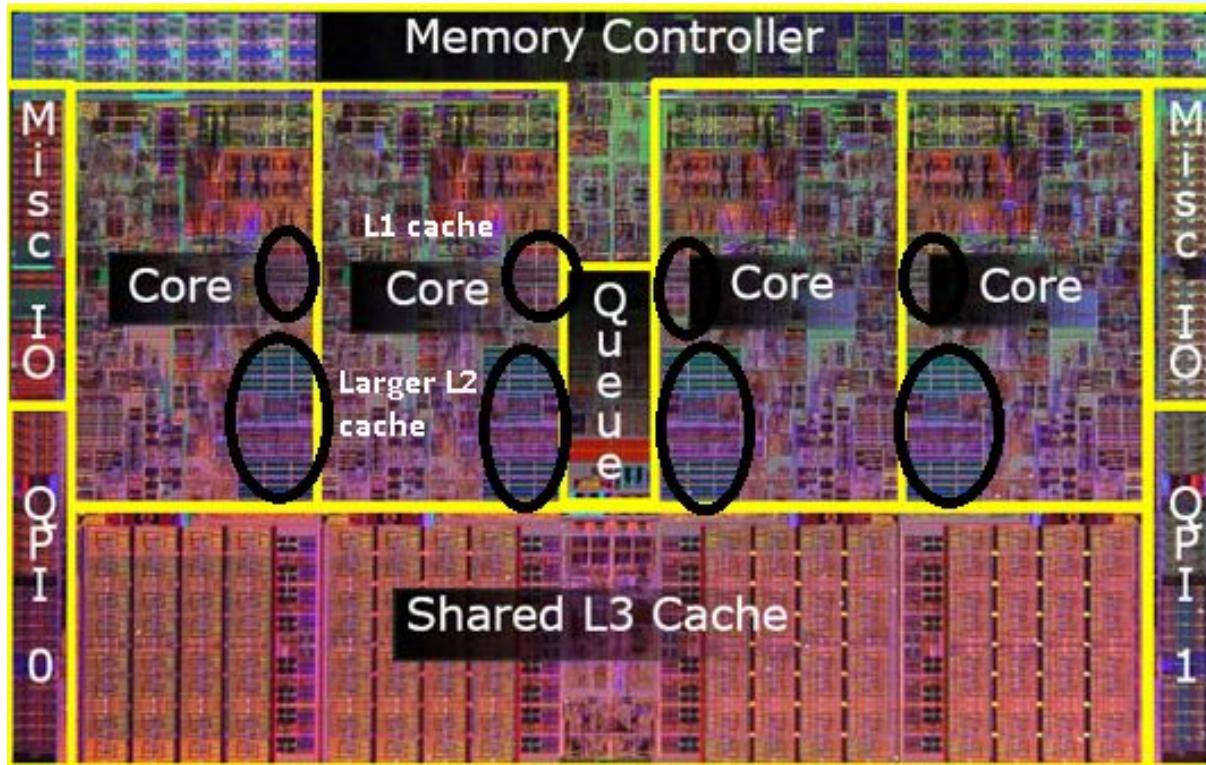
09 Cache-friendliness



EPIC

Institute of Technology
Powered by epam

Caches



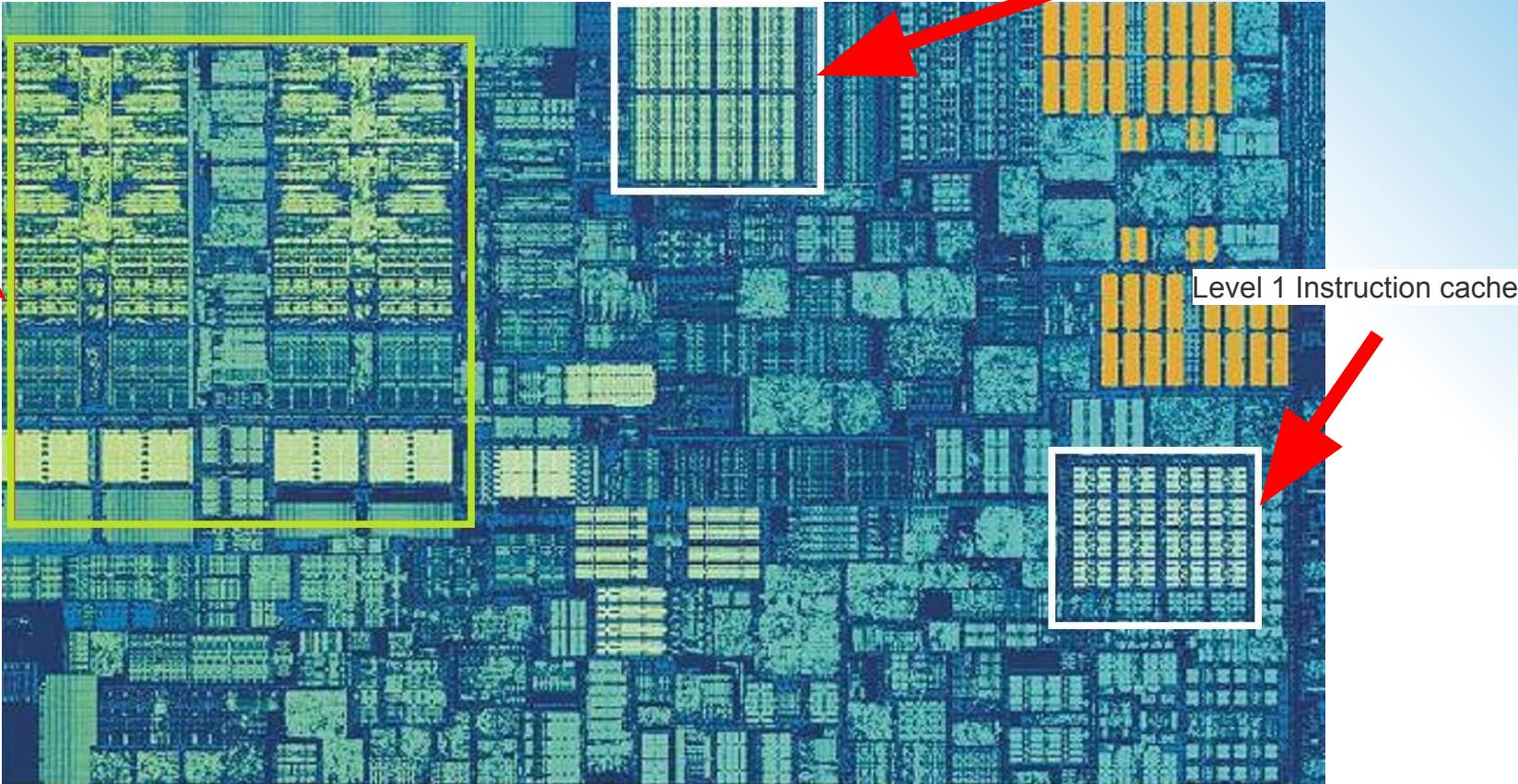


EPIC

Institute of Technology
Powered by epam

Caches

ALU



Level 1 Data cache

Level 1 Instruction cache



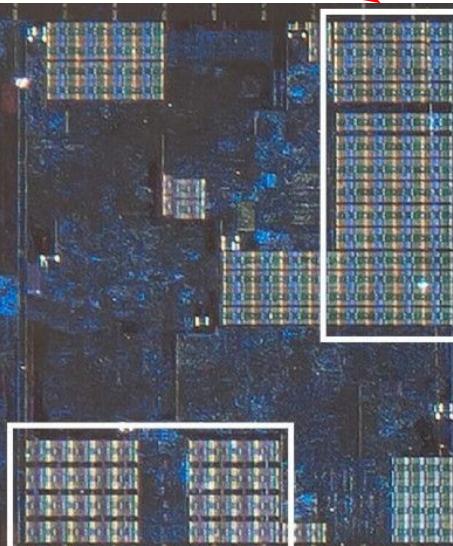
EPIC

Institute of Technology
Powered by epam

Caches

Level 1

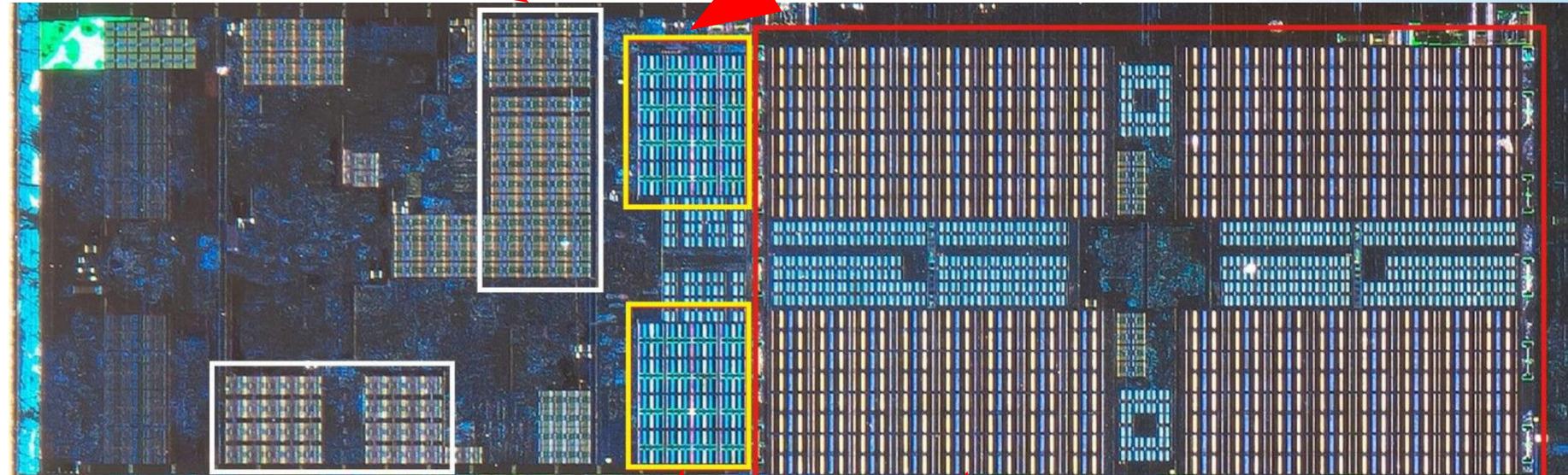
Level 2



Level 1

Level 2

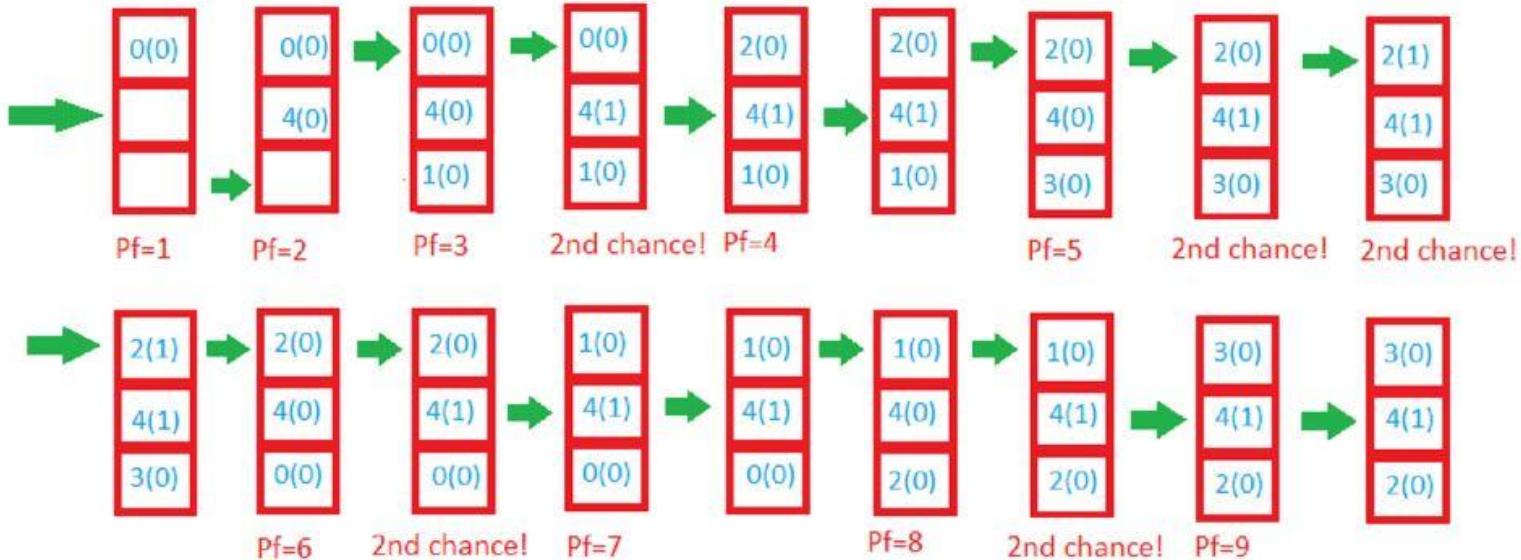
Level 3





Cache in DB

Page sequence: 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4



06

RAM

In this section, we will discuss RAM.



EPIC

Institute of Technology
Powered by epam

RAM





SDR





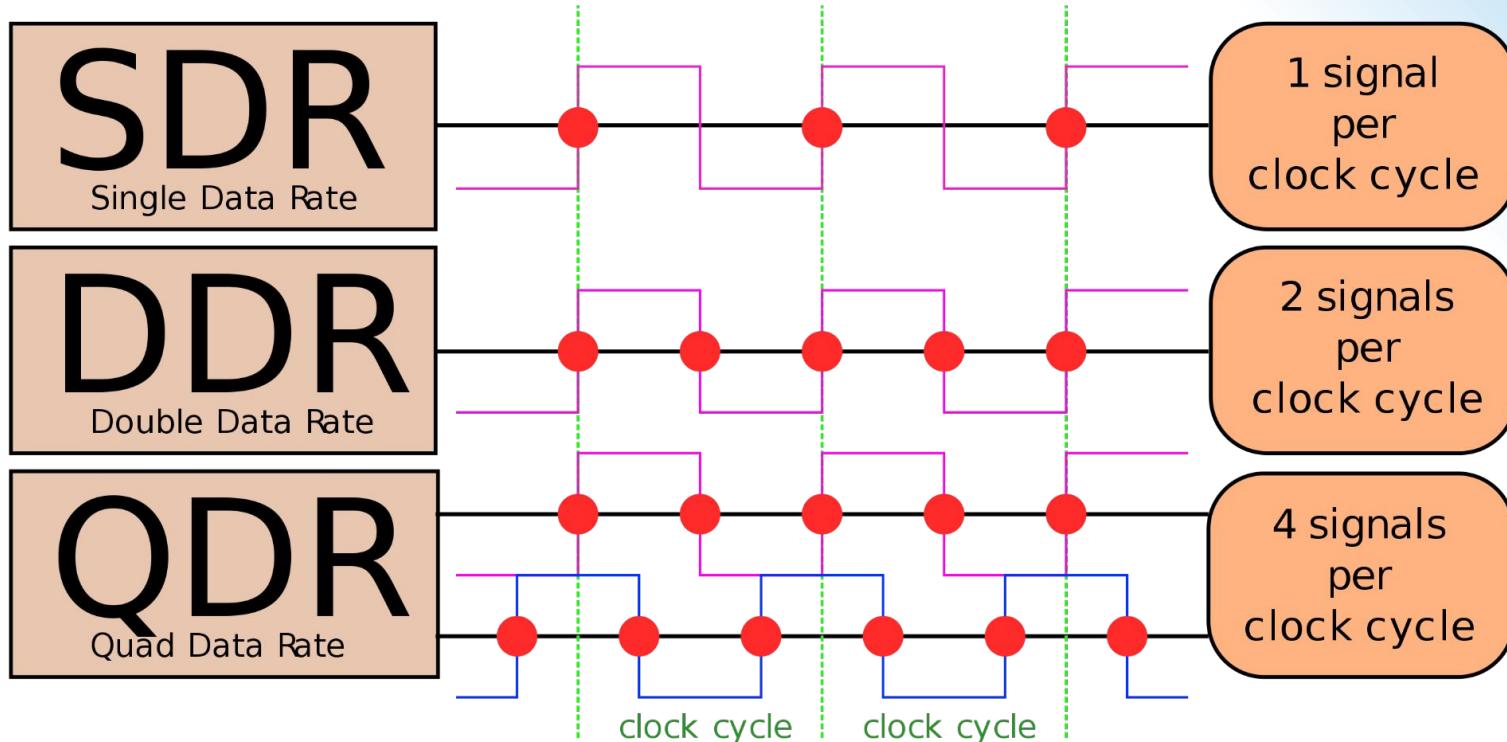
EPIC

Institute of Technology
Powered by epam

DDR

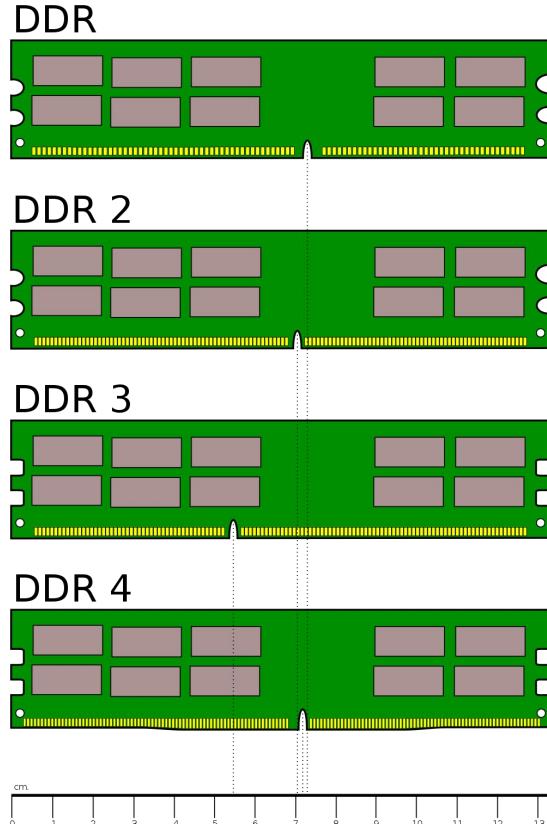


How is working





DDR version



Example

8th Generation Intel® Core™ i3 Desktop Processors | DDR4-**2400**

64 GB

Sponsored ⓘ

8GB DDR4-2666 PC4-21300 Non-ECC UDIMM Desktop PC RAM Memory Upgrade

 87

\$19⁹⁹

Delivery **Mon, Apr 15**

Ships to Serbia

Add to cart



Example

11th Generation Intel® Core™ i7 Desktop Processors

DDR4-3200

128 GB

Sponsored ⓘ



8GB DDR4-2666 PC4-21300 Non-ECC UDIMM Desktop PC RAM Memory Upgrade

★★★★★ v 87

\$19⁹⁹

Delivery **Mon, Apr 15**

Ships to Serbia

Add to cart

Example

11th Generation Intel® Core™ i7 Desktop Processors

DDR4-3200

128 GB





EPIC

Institute of Technology
Powered by 

Example

CPU	Mainboard	Memory	SPD	Graphics	Bench	About
General						
Type	DDR5			Channels #	2 x 32-bit	
Size	32 GBytes			DC Mode		
NB Frequency						
Timings						
DRAM Frequency		2997.6 MHz				
FSB:DRAM		1:30				
CAS# Latency (CL)		38.0 docks				
RAS# to CAS# Delay (tRCD)		38 docks				
RAS# Precharge (tRP)		38 docks				
Cycle Time (tRAS)		78 docks				
Bank Cycle Time (tRC)		116 docks				
Command Rate (CR)						
DRAM Idle Timer						
Total CAS# (tRDRAM)						
Row To Column (tRCD)						

The screenshot shows the MSI Click BIOS 5 interface. At the top, it displays the date and time (21:36 Sun 31 Mar, 2024) and the BIOS version (Advanced F7). The top right corner includes icons for a camera (F12), a monitor (En), and a gear.

CPU Speed: 4.70 GHz
DDR Speed: 6000 MHz

CREATOR GENIE

A-XMP: 1, 2
Profile: EXPO, 1, 2

Boot Priority: Icons for UEFI, CD/DVD, USB, and SD cards.

EZ Mode:

- CPU** (selected)
- Memory** (highlighted)
- Storage**
- Fan Info**
- Help**

Current DRAM Frequency: 6000 MHz

Current DRAM Size: 32768 MB

Model	Size
DIMM1: Empty	-
DIMM2: TEAM GROUP INC.	16384 MB
DIMM3: Empty	-
DIMM4: TEAM GROUP INC.	16384 MB

EXPO Profile 1: DDR5 6000MHz 38-38-38-78 1.250V
EXPO Profile 2: DDR5 5600MHz 40-40-40-84 1.200V

M-Flash

ITPM 2.0: ON

CSM /UEFI: ON

ErP Ready: OFF

AHCI /RAID: ON

HD Audio Controller: ON

EZ LED Control: ON

Favorites

Hardware Monitor



EPIC

Institute of Technology
Powered by epam

Overclock

The screenshot shows the ASRock UEFI OC Tweaker interface. The main menu bar includes Main, OC Tweaker (selected), Advanced, Tool, H/W Monitor, Security, Boot, and Exit. A sub-menu bar shows My Favorite (selected) and Easy Mode (F6). The OC Tweaker menu has sections for DRAM Configuration, DRAM Tweaker, DRAM Timing Configuration, XMP 2.0 Profile 1, Load XMP Setting, BCLK Frequency, DRAM Reference Clock, DRAM Frequency (selected), DRAM Clock, and Primary Timing.

A dropdown menu for DRAM Frequency lists the following values:

- DDR4-6533
- DDR4-6666
- DDR4-6800
- DDR4-6933
- DDR4-7066
- DDR4-7200
- DDR4-7333
- DDR4-7466
- DDR4-7600
- DDR4-7733
- DDR4-7866
- DDR4-8000
- DDR4-8133
- DDR4-8266
- DDR4-8400

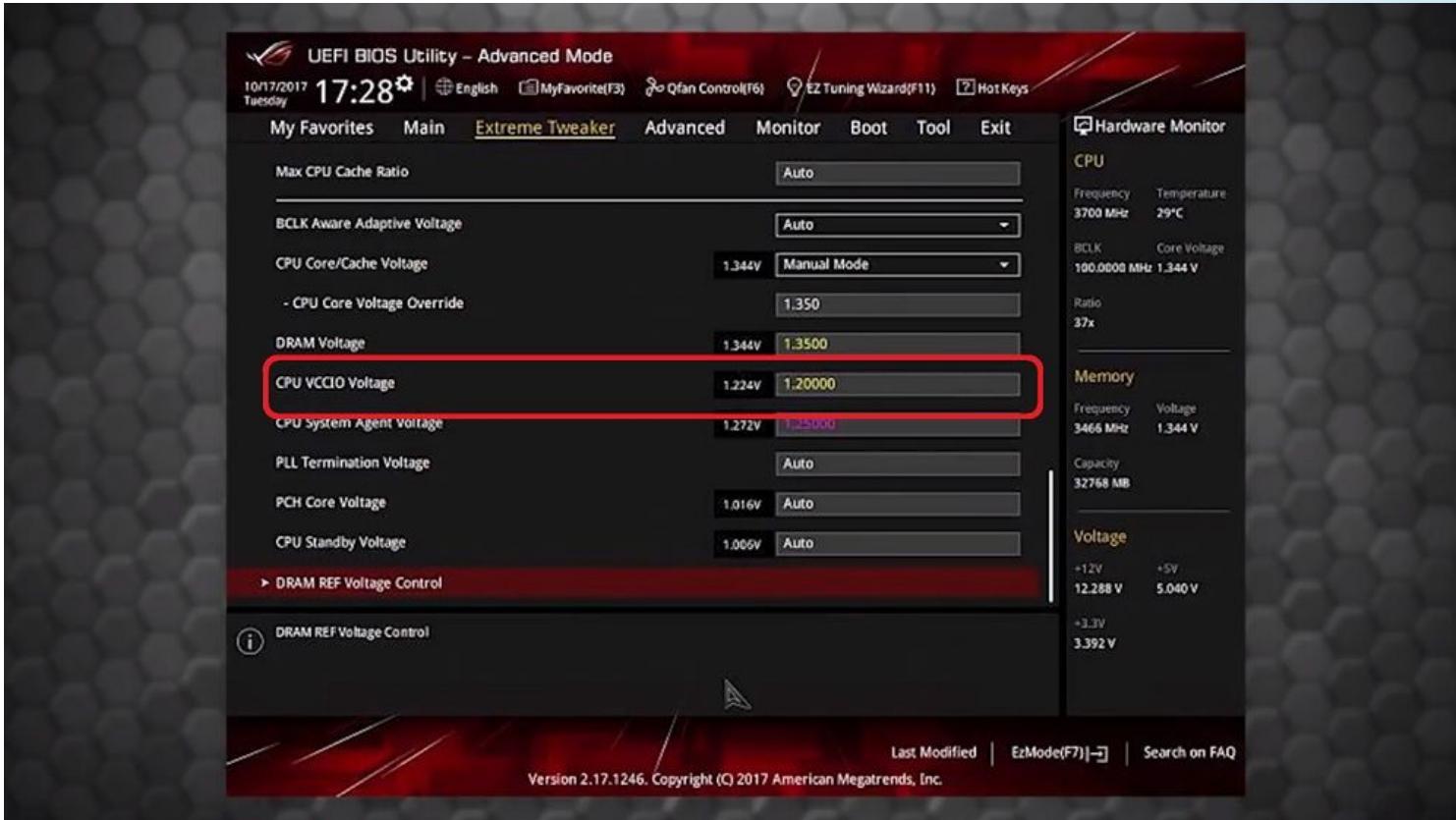
A description box states: "If [Auto] is selected, the motherboard will detect the memory module(s) inserted and assign the appropriate frequency automatically." It also includes a QR code and a link to "Get details via QR code".

At the bottom, there are language and date/time settings: English and Fri 10/12/2018, 12:08:50.

Result



Good solution



Result





EPIC
Institute of Technology
Powered by epam

Good solution

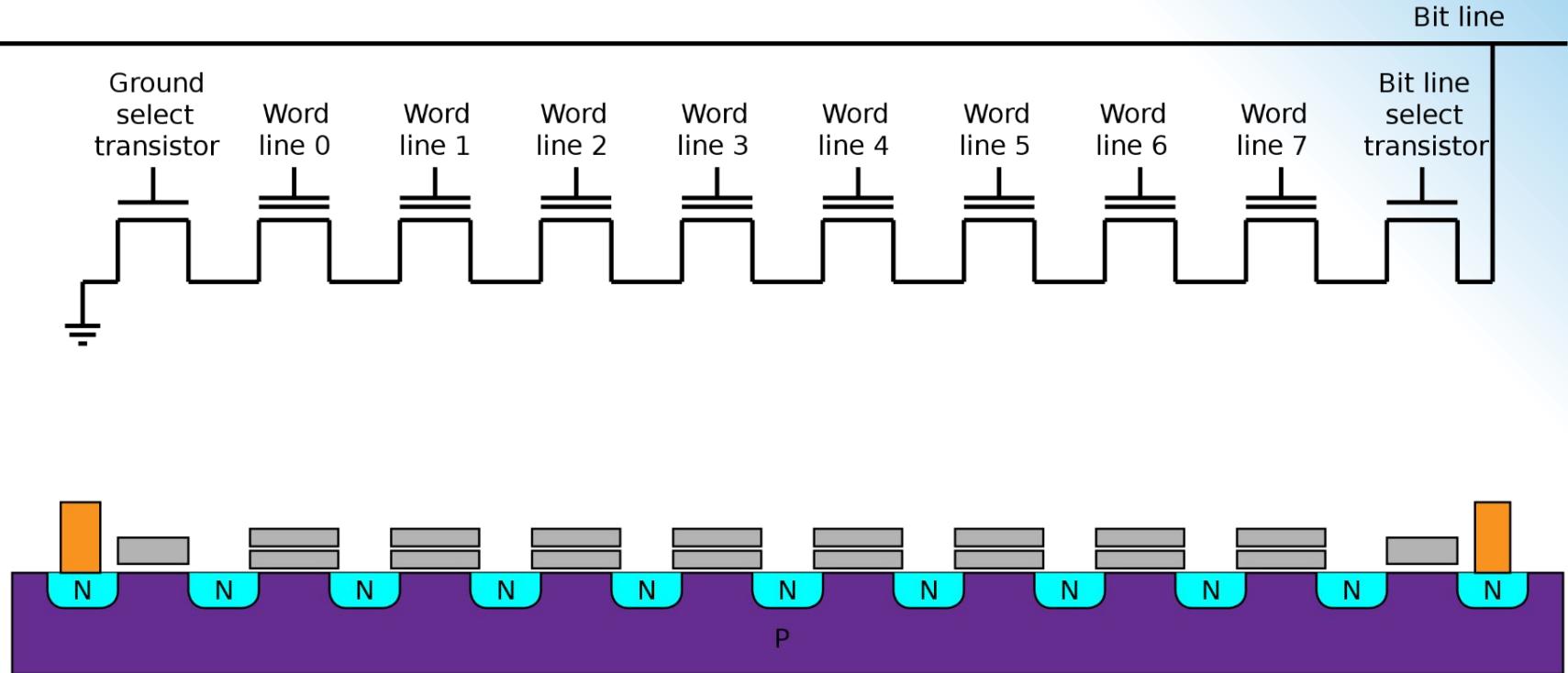


07

SSD

In this section, we will discuss SSD.

NAND





EPIC

Institute of Technology
Powered by epam

Cell types of storage

1 bit per cell

2 bits per cell

3 bits per cell

4 bits per cell

5 bits per cell

1

0

SLC

11

10

01

00

MLC/DLC

111

110

101

100

011

010

001

000

TLC

1111

1110

1101

1100

1011

1010

1001

1000

0111

0110

0101

0100

0011

0010

0001

0000

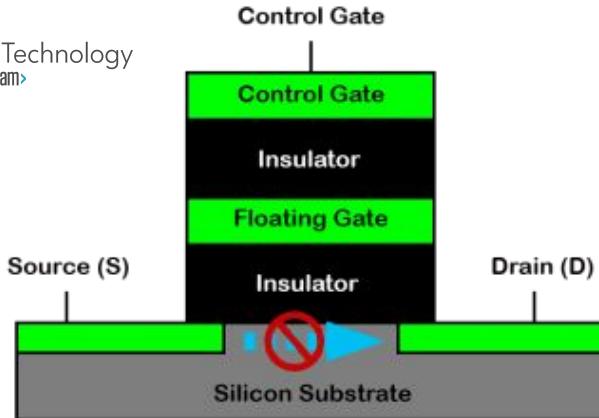
QLC

PLC

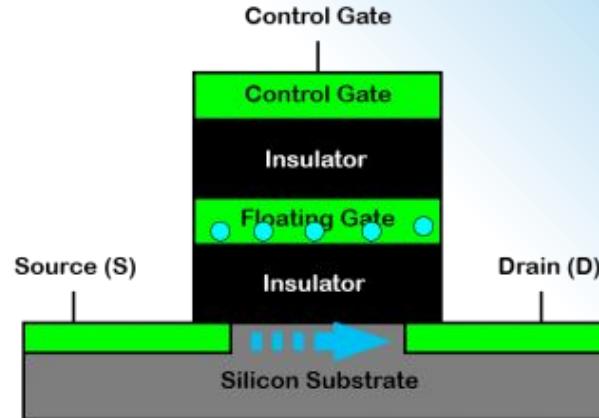


EPIC

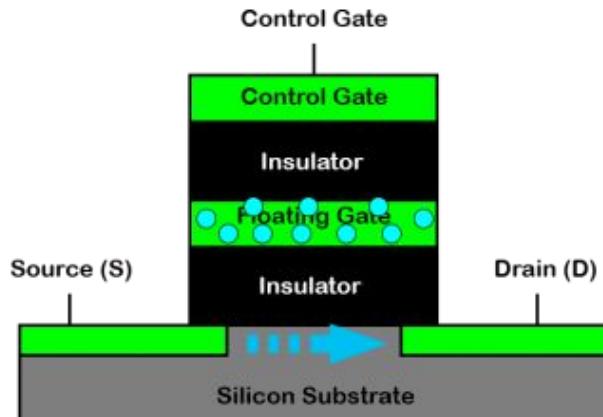
Institute of Technology
Powered by epam



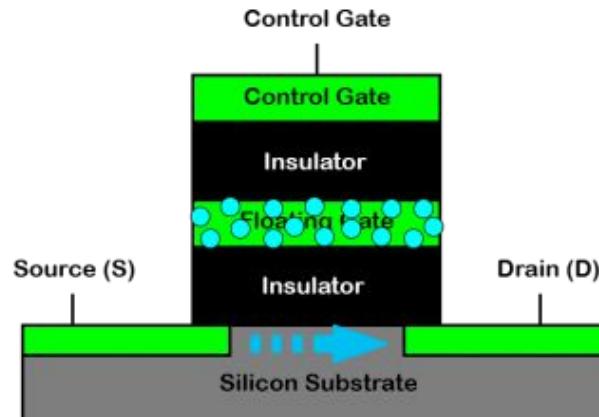
State 1 - No Charge



State 2 - Lightly Charged



State 3 - Medium Charge



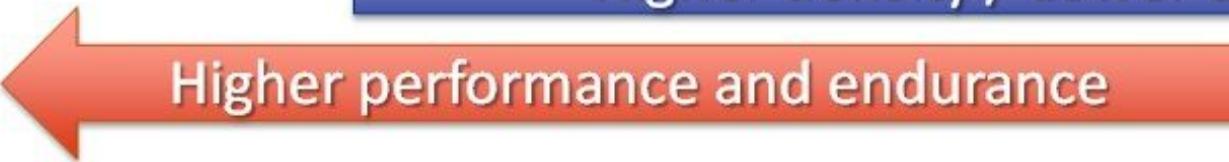
State 4 - Highly Charged

Benchmarks

	SLC	MLC	TLC
Bits per cell	1	2	3
P/E Cycles	100,000	3,000	1,000
Read Time	25 µs	50 µs	~75 µs
Program Time	200-300 µs	600-900 µs	~900-1350 µs
Erase Time	1.5-2 ms	3 ms	4.5 ms



Higher density / Lower cost



Higher performance and endurance

07

HDD

In this section, we will discuss HDD.

HDD

The picture below shows a internal and external view of 2.5-inch HDD:



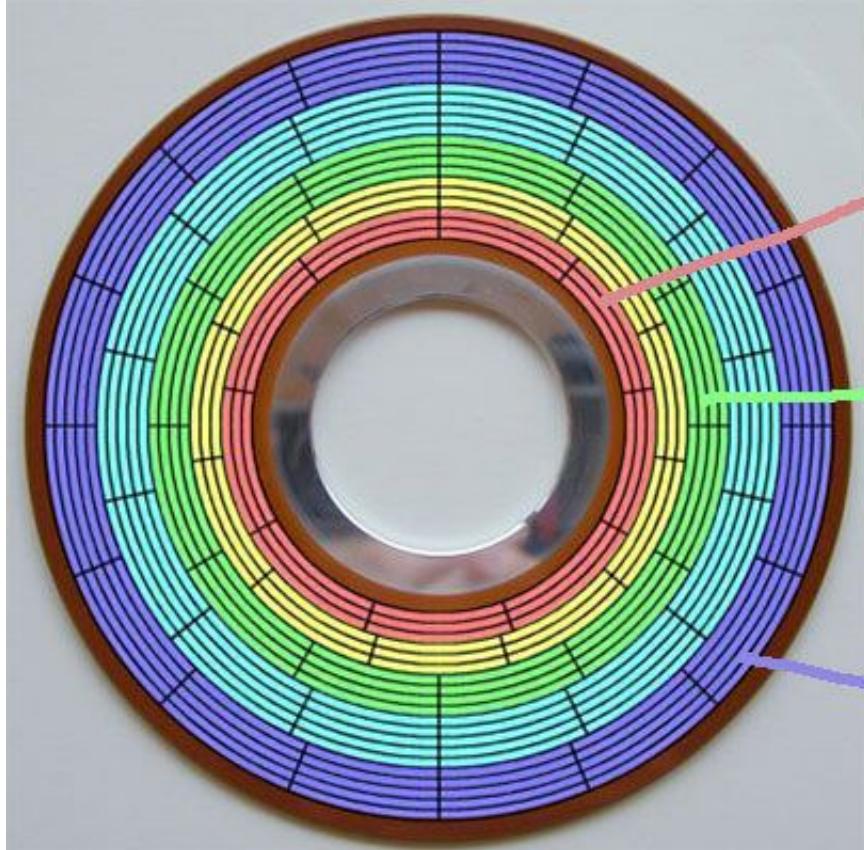
Internal View



External View



HDD division



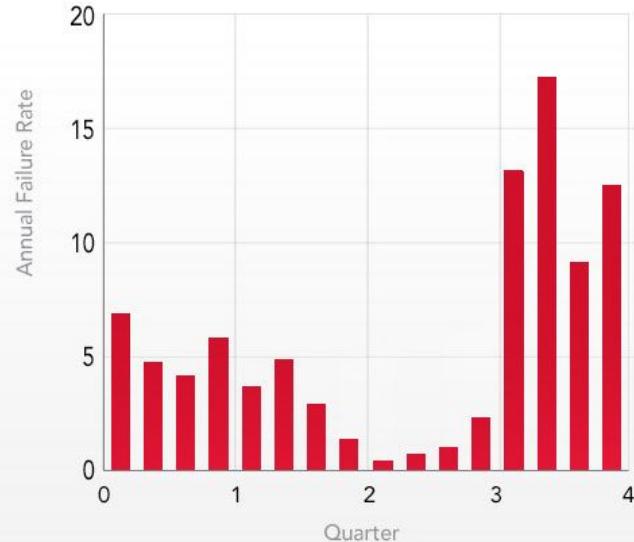
9 sectors
per track

12 sectors
per track

16 sectors
per track

Reliability

Annual Failure Rate Each Quarter





EPIC
Institute of Technology
Powered by epam

RAID

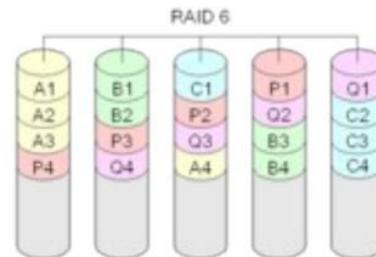
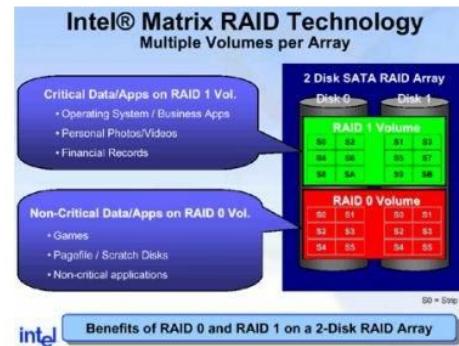
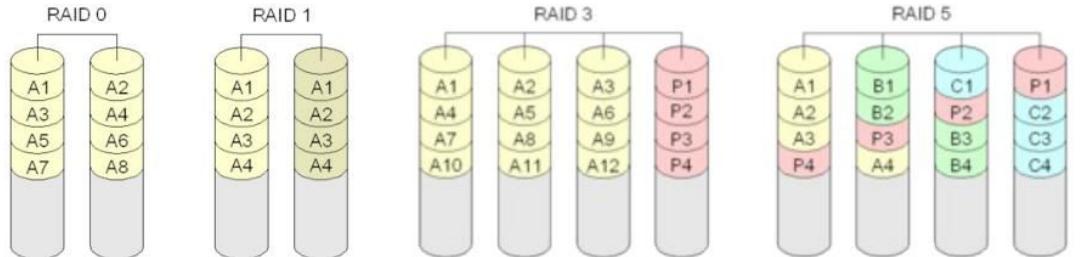




EPIC

Institute of Technology
Powered by epam

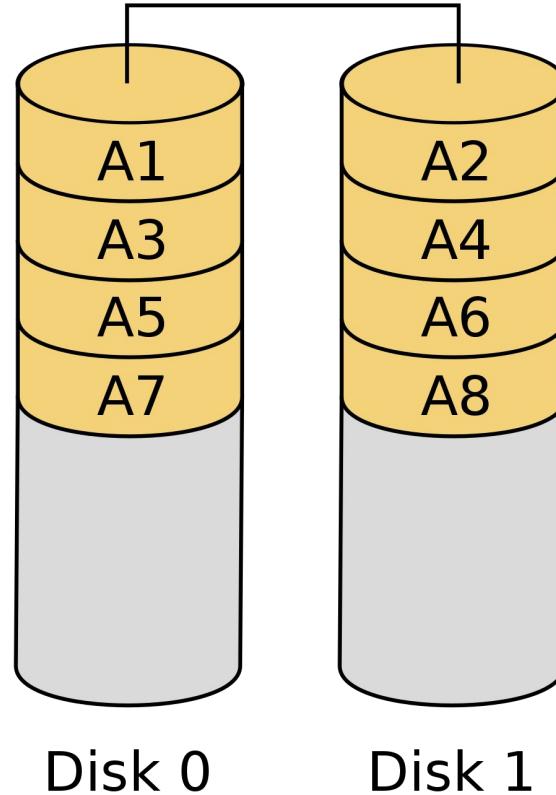
RAID



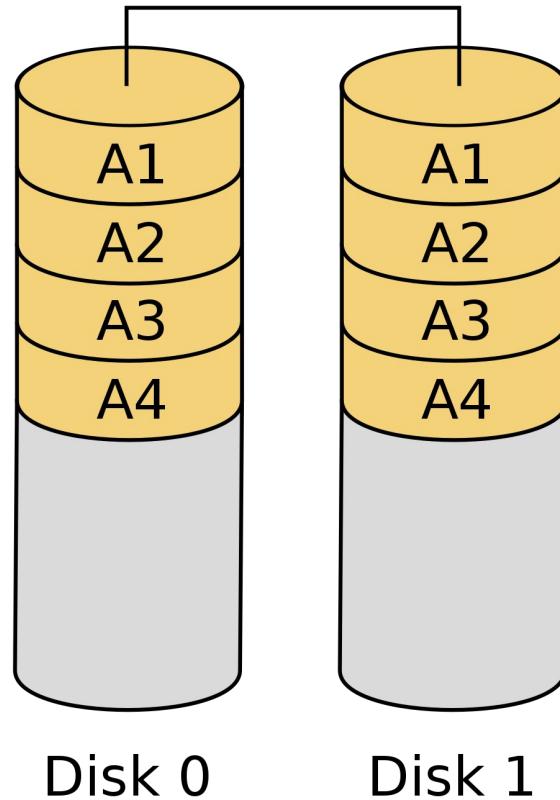
RAID



RAID 0

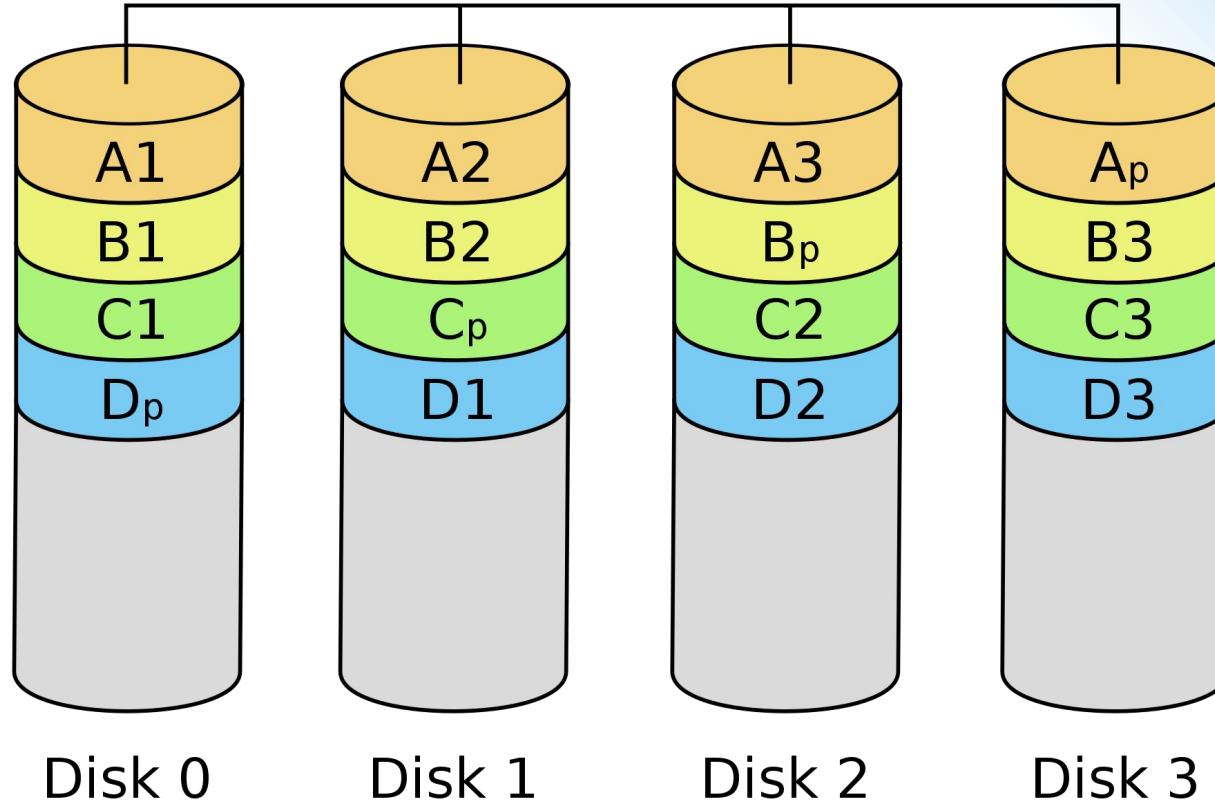


RAID 1





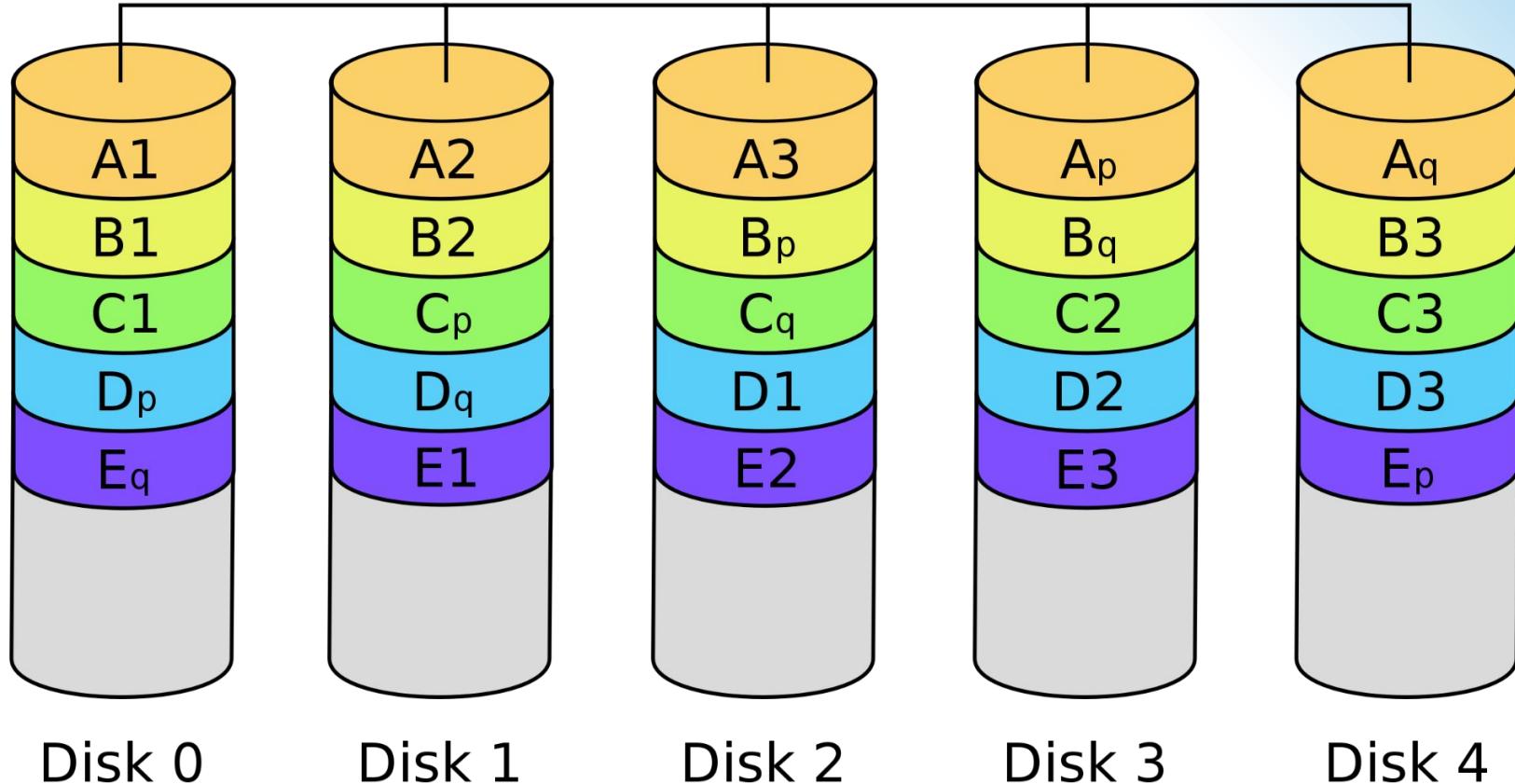
RAID 5



Paritet in RAID 5

- 1) $A_1 \text{ XOR } A_1 = 0$ (always!!!!)
- 2) $A_1 \text{ XOR } A_2 \text{ XOR } A_3 = \text{num}$
- 3) $\text{num} \text{ XOR } A_1 \text{ XOR } A_2 = A_3$

RAID 6



Paritet in RAID 6

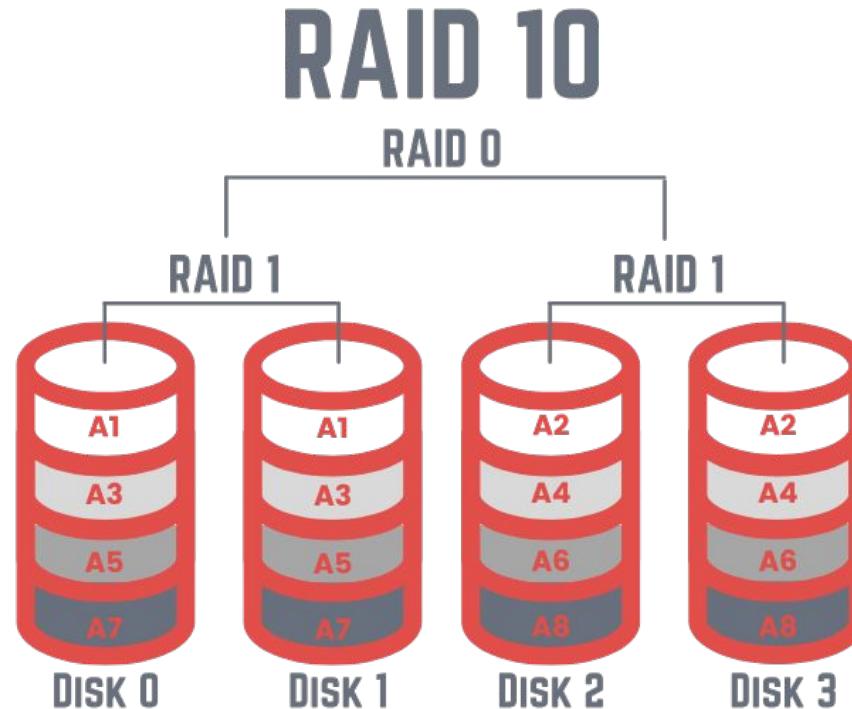
$$\mathbf{P} = \bigoplus_i \mathbf{D}_i = \mathbf{D}_0 \oplus \mathbf{D}_1 \oplus \mathbf{D}_2 \oplus \dots \oplus \mathbf{D}_{n-1}$$

$$\mathbf{Q} = \bigoplus_i g^i \mathbf{D}_i = g^0 \mathbf{D}_0 \oplus g^1 \mathbf{D}_1 \oplus g^2 \mathbf{D}_2 \oplus \dots \oplus g^{n-1} \mathbf{D}_{n-1}$$

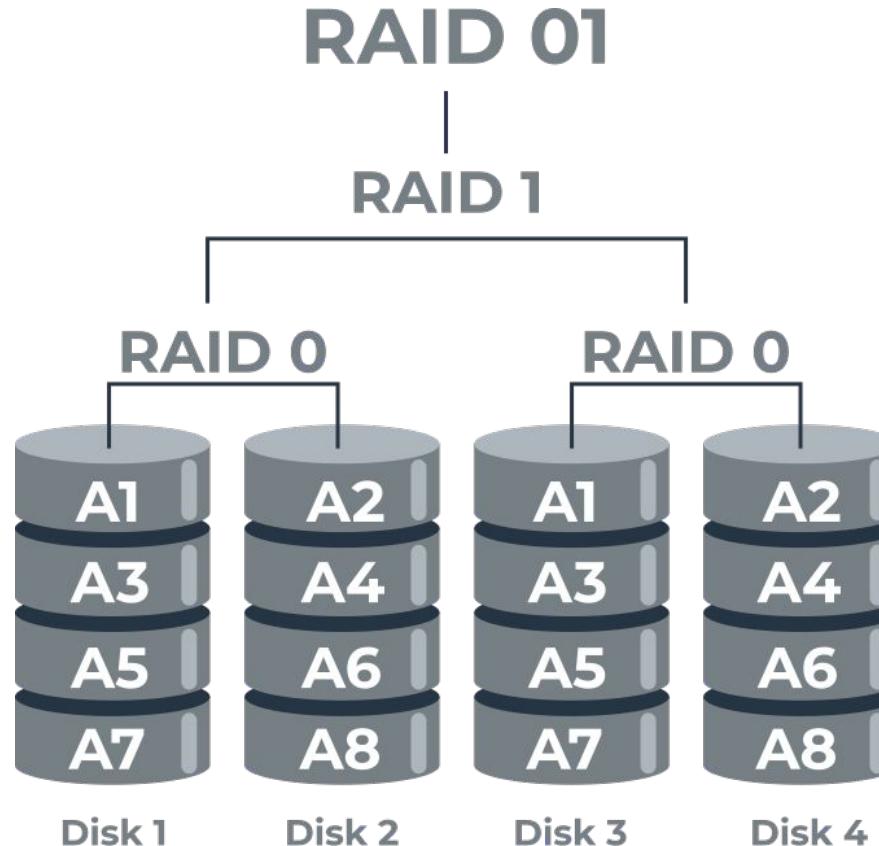
$$A = \mathbf{P} \oplus \left(\bigoplus_{\ell: \ell \neq i \text{ and } \ell \neq j} D_\ell \right) = D_i \oplus D_j$$

$$B = \mathbf{Q} \oplus \left(\bigoplus_{\ell: \ell \neq i \text{ and } \ell \neq j} g^\ell D_\ell \right) = g^i D_i \oplus g^j D_j$$

RAID 10 (RAID 1+0)



RAID 01 (RAID 0+1)



08

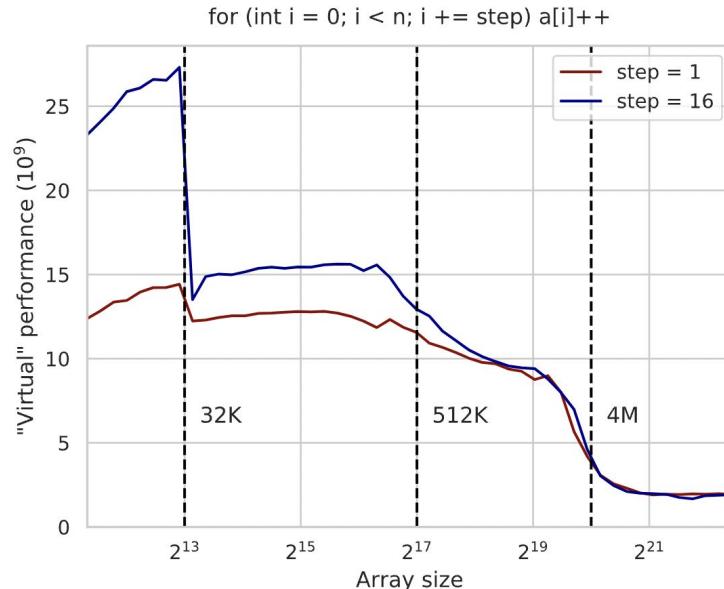
Cache-friendliness

In this section, we will discuss Cache-friendly ideas.

Funny thing.

```
for (int i = 0; i < N; i += D)
    a[i]++;
```

If we run it with $D = 1$ and $D = 16$, we can observe something interesting:



Performance is normalized by the total time to run benchmark, not the total number of elements incremented

Simple example

```
1 static void OneCycle() {
2     for (int i = 0; i < 10000; i++) {
3         z[i] = x[i] - y[i];
4         z[i] = z[i] * z[i];
5     }
6 }
7
8 static void TwoCycles() {
9     for (int i = 0; i < 10000; i++) {
10        z[i] = x[i] - y[i];
11    }
12    for (int i = 0; i < 10000; i++) {
13        z[i] = z[i] * z[i];
14    }
15 }
```



EPIC

Institute of Technology
Powered by epam

Comparison

Quick C++ Benchmark

Run Quick Bench locally Support Quick Bench Suite ▾ More ▾

```
6
7     }
8     for (auto _ : state) {
9         for (int i = 0; i < 10000; i++) {
10            z[i] = x[i] - y[i];
11            z[i] = z[i] * z[i];
12        }
13
14        int sum = 0;
15        for (int i = 0; i < 10000; i++) {
16            sum += z[i];
17        }
18        if (sum == 0) {
19            exit(1);
20        }
21    }
22 BENCHMARK(OneCycle);
23
24 static void TwoCycles(benchmark::State& state) {
25     int z[10000], x[10000], y[10000];
26     for (int i = 0; i < 10000; i++) {
27         x[i] = i % 100;
28         y[i] = i % 151;
29     }
30     for (auto _ : state) {
31         for (int i = 0; i < 10000; i++) {
32             z[i] = x[i] - y[i];
33         }
34         for (int i = 0; i < 10000; i++) {
35             z[i] = z[i] * z[i];
36         }
37     }
38     int sum = 0;
39     for (int i = 0; i < 10000; i++) {
40         sum += z[i];
41     }
42     if (sum == 0) {
43         exit(1);
44     }
45 }
46 BENCHMARK(TwoCycles);
47
```

compiler = Clang 15.0 ▾ std = c++20 ▾ optim = O2 ▾
STL = libstdc++(GNU) ▾

Run Benchmark Record disassembly Clear cached results

Charts Assembly

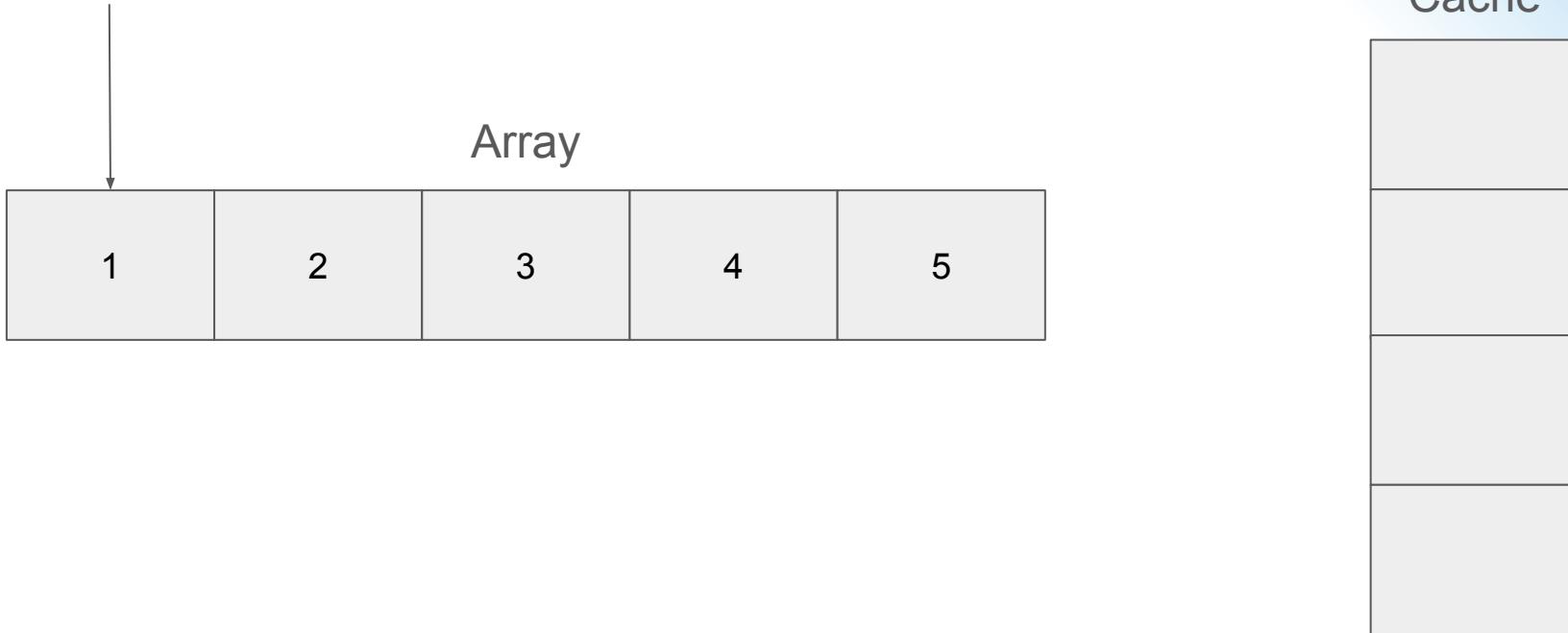
The chart shows the ratio of CPU time to Noop time for two benchmarks: OneCycle and TwoCycles. The Y-axis represents the ratio, ranging from 0 to 18,000. The X-axis labels are OneCycle and TwoCycles. The OneCycle bar is blue and reaches approximately 9,500. The TwoCycles bar is yellow and reaches approximately 17,000.

Benchmark	Ratio (CPU time / Noop time)
OneCycle	~9,500
TwoCycles	~17,000

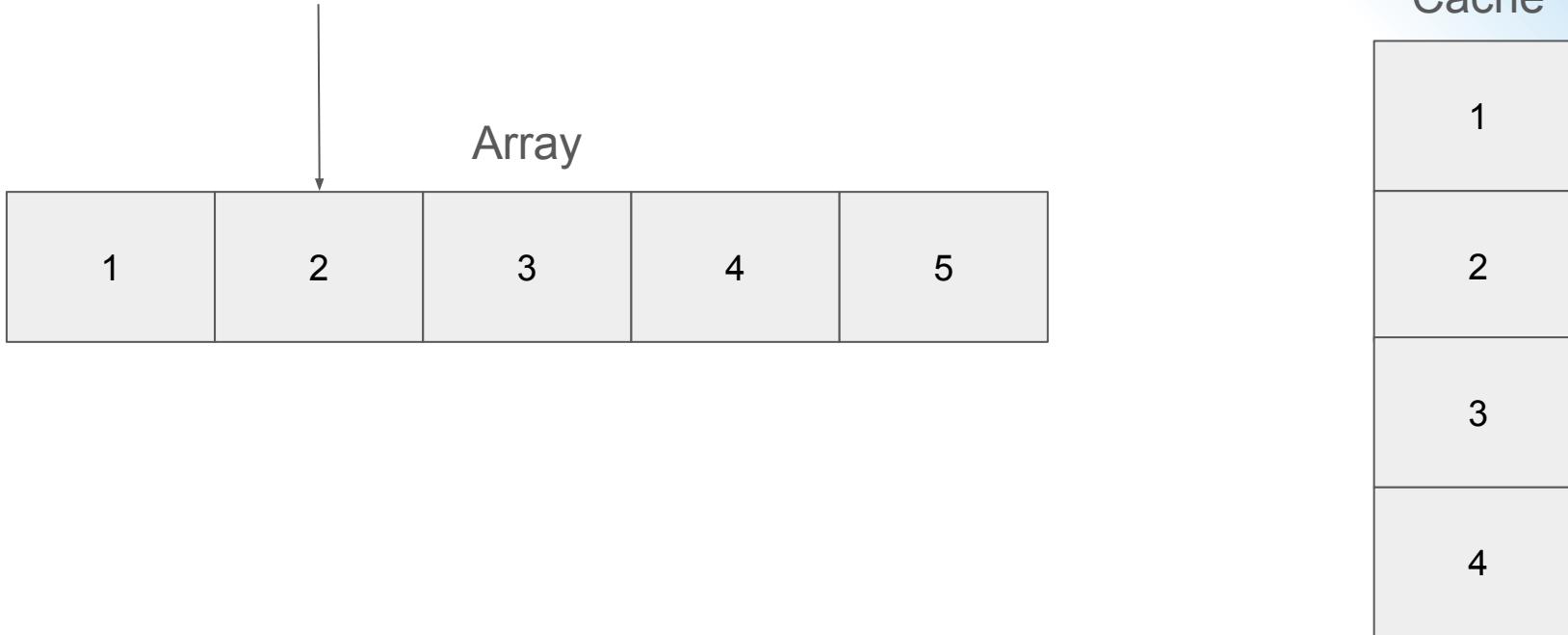
ratio (CPU time / Noop time)
Lower is faster

Show Noop bar

How is it working?

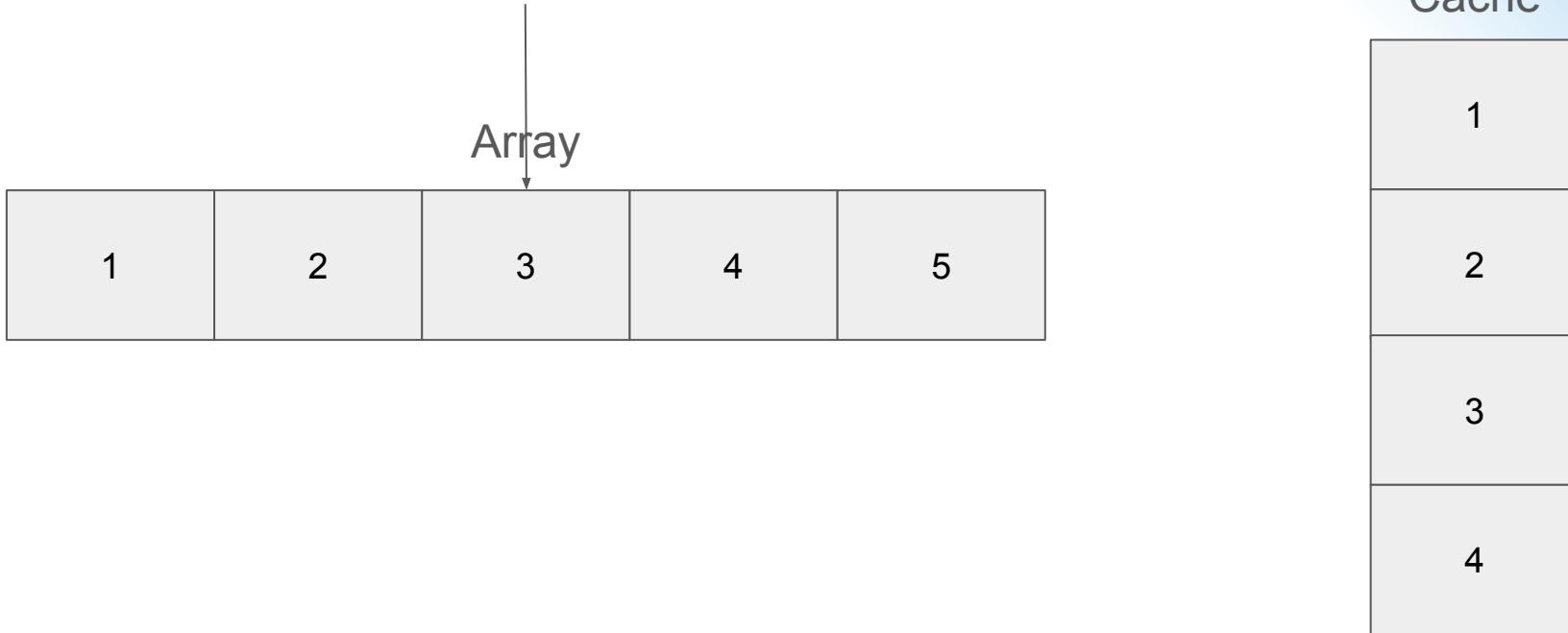


How is it working?

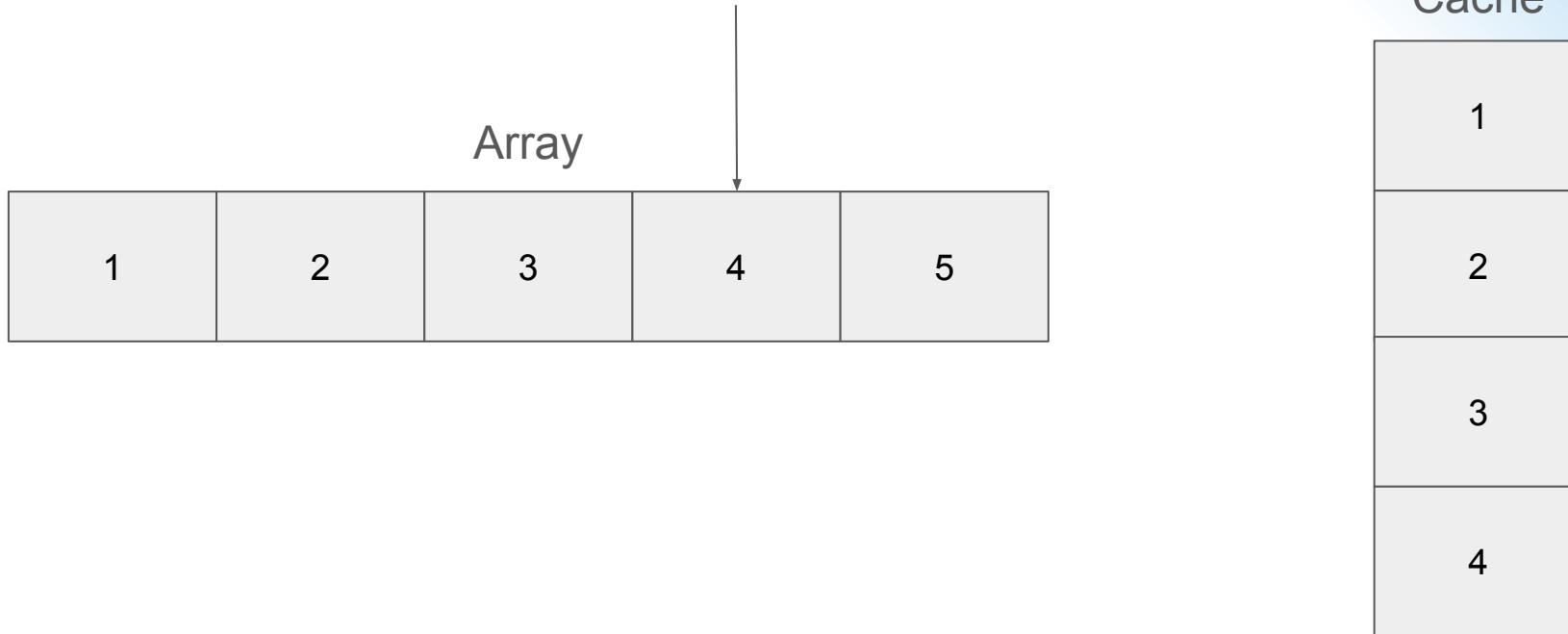




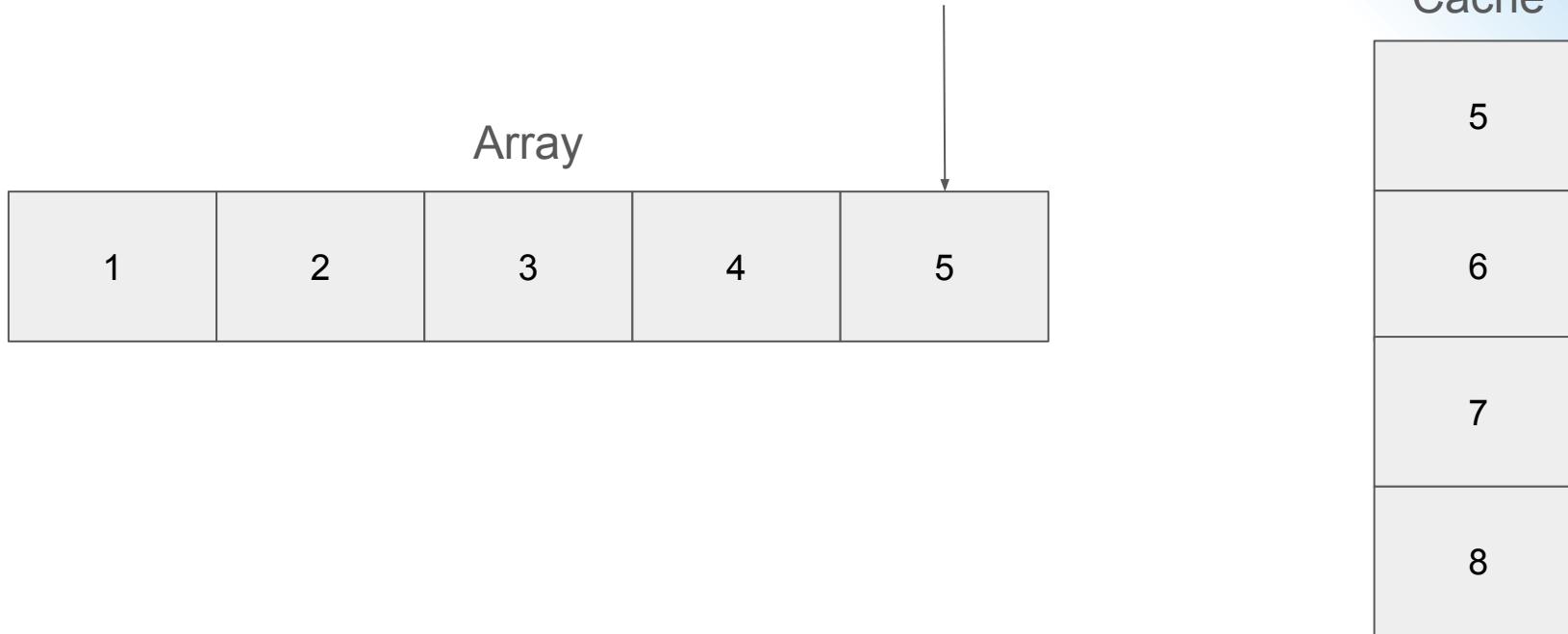
How is it working?



How is it working?



How is it working?





EPIC

Institute of Technology
Powered by epam

Matrix

Matrix

@math-only-math.com

@math-only-math.com

$$\begin{bmatrix} 2 & 5 & 3 & 4 \\ 4 & 7 & 1 & 5 \\ 3 & 0 & 5 & 8 \end{bmatrix}$$

or,

$$\begin{array}{cccc} 2 & 5 & 3 & 4 \\ 4 & 7 & 1 & 5 \\ 3 & 0 & 5 & 8 \end{array}$$

Transpose of a matrix

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3}$$

$$A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$



EPIC

Institute of Technology
Powered by epam

Transpose of a matrix

```
5. int main() {
6.     int n, m;
7.     cin >> n >> m;
8.     int matrix[n][m], matrix_t[m][n];
9.     for (int i = 0; i < n; i++) {
10.         for (int j = 0; j < m; j++) {
11.             cin >> matrix[i][j];
12.         }
13.     }
14.
15.     // transpose
16.     for (int i = 0; i < n; i++) {
17.         for (int j = 0; j < m; j++) {
18.             matrix_t[j][i] = matrix[i][j];
19.         }
20.     }
21.
22.     for (int i = 0; i < m; i++) {
23.         for (int j = 0; j < n; j++) {
24.             cout << matrix_t[i][j] << " ";
25.         }
26.         cout << "\n";
27.     }
28.
29.     return 0;
30. }
```

Success #stdin #stdout 0.01s 5300KB



stdin

```
3 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

stdout

```
1 6 11
2 7 12
3 8 13
4 9 14
5 10 15
```



EPIC

Institute of Technology
Powered by epam

Cache-friendly

```
15.     // transpose
16.     int block_size = 2;
17.
18.     for (int i = 0; i < n; i += block_size) {
19.         for (int j = 0; j < n; j += block_size) {
20.             for (int ii = i; ii < i + block_size && ii < n; ii++) {
21.                 for (int jj = j; jj < j + block_size && jj < n; jj++) {
22.                     matrix_t[jj][ii] = matrix[ii][jj];
23.                 }
24.             }
25.         }
26.     }
27.
28.     for (int i = 0; i < m; i++) {
29.         for (int j = 0; j < n; j++) {
30.             cout << matrix_t[i][j] << " ";
31.         }
32.         cout << "\n";
33.     }
34.
35.     return 0;
36. }
```

Success #stdin #stdout 0s 5296KB

(stdin

```
4 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

(stdout

```
1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16
```

Comparing

```

 edit  fork  download  copy
1. #include <bits/stdc++.h>
2.
3. using namespace std;
4.
5. int main() {
6.     int n = 10000, m = 10000;
7.     int matrix[n][m], matrix_t[m][n];
8.     for (int i = 0; i < n; i++) {
9.         for (int j = 0; j < m; j++) {
10.             matrix[i][j] = (i + j);
11.         }
12.     }
13.
14.     // transpose
15.     for (int i = 0; i < n; i++) {
16.         for (int j = 0; j < m; j++) {
17.             matrix_t[j][i] = matrix[i][j];
18.         }
19.     }
20.
21.     for (int i = 0; i < m; i += 100) {
22.         cout << matrix_t[i][i] << " ";
23.     }
24.
25.     return 0;
26. }

```

Success #stdin #stdout 0.47s 784800KB

comments ()

stdin copy

Standard input is empty

stdout copy

```

0 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600 2800 3000 3200 3400 3600 3800 4000
4200 4400 4600 4800 5000 5200 5400 5600 5800 6000 6200 6400 6600 6800 7000 7200 7400 7600 7800 80
00 8200 8400 8600 8800 9000 9200 9400 9600 9800 10000 10200 10400 10600 10800 11000 11200 11400 1
1600 11800 12000 12200 12400 12600 12800 13000 13200 13400 13600 13800 14000 14200 14400 14600 14
800 15000 15200 15400 15600 15800 16000 16200 16400 16600 16800 17000 17200 17400 17600 17800 180
00 18200 18400 18600 18800 19000 19200 19400 19600 19800

```

```

 copy
15.     int block_size = 4;
16.
17.     for (int i = 0; i < n; i += block_size) {
18.         for (int j = 0; j < n; j += block_size) {
19.             for (int ii = i; ii < i + block_size && ii < n; ii++) {
20.                 for (int jj = j; jj < j + block_size && jj < n; jj++) {
21.                     matrix_t[jj][ii] = matrix[ii][jj];
22.                 }
23.             }
24.         }
25.     }
26.
27.     for (int i = 0; i < m; i += 100) {
28.         cout << matrix_t[i][i] << " ";
29.     }
30.
31.     return 0;
32. }

```

Success #stdin #stdout 0.34s 784732KB

comments (?)

stdin copy

Standard input is empty

stdout copy

```

0 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600 2800 3000 3200 3400 3600 3800 4000
4200 4400 4600 4800 5000 5200 5400 5600 5800 6000 6200 6400 6600 6800 7000 7200 7400 7600 7800 80
00 8200 8400 8600 8800 9000 9200 9400 9600 9800 10000 10200 10400 10600 10800 11000 11200 11400 1
1600 11800 12000 12200 12400 12600 12800 13000 13200 13400 13600 13800 14000 14200 14400 14600 14
800 15000 15200 15400 15600 15800 16000 16200 16400 16600 16800 17000 17200 17400 17600 17800 180
00 18200 18400 18600 18800 19000 19200 19400 19600 19800

```

<https://ideone.com/t3jbT>



EPIC
Institute

Institute of Technology
Powered by 

Cache misses



EPIC

Institute of Technology
Powered by epam

```
7.     int M[size][size];
8.
9.     int matrixsum1() {
10.         int sum = 0;
11.         for (int i = 0; i < size; i++) {
12.             for (int j = 0; j < size; j++) {
13.                 sum += M[i][j];
14.             }
15.         }
16.
17.         return sum % 1503;
18.     }
19.
20.    int matrixsum2() {
21.        int sum = 0;
22.        for (int i = 0; i < size; i++) {
23.            for (int j = 0; j < size; j++) {
24.                sum += M[j][i];
25.            }
26.        }
27.        return sum % 1503;
28.    }
29.
30.    int main() {
31.        for (int i = 0; i < size; i++) {
32.            for (int j = 0; j < size; j++) {
33.                M[i][j] = (i + j) % 100;
34.            }
35.        }
36.        int ans = 0;
37.        for (int i = 0; i <= size; i++) {
38.            ans += matrixsum1();
39.        }
40.        cout << ans;
41.        return 0;
42.    }
```

<https://ideone.com/TJrl23>

**EPIC**Institute of Technology
Powered by epam

```
30. int main() {
31.     for (int i = 0; i < size; i++) {
32.         for (int j = 0; j < size; j++) {
33.             M[i][j] = (i + j) % 100;
34.         }
35.     }
36.     int ans = 0;
37.     for (int i = 0; i <= size; i++) {
38.         ans += matrixsum1();
39.     }
40.     cout << ans;
41.     return 0;
42. }
```

Success #stdin #stdout 0.5s 7504KB

stdin

Standard input is empty

stdout

198198

**EPIC**Institute of Technology
Powered by epam

```
30. int main() {
31.     for (int i = 0; i < size; i++) {
32.         for (int j = 0; j < size; j++) {
33.             M[i][j] = (i + j) % 100;
34.         }
35.     }
36.     int ans = 0;
37.     for (int i = 0; i <= size; i++) {
38.         ans += matrixsum2();
39.     }
40.     cout << ans;
41.     return 0;
42. }
```

Success #stdin #stdout 1.33s 7456KB

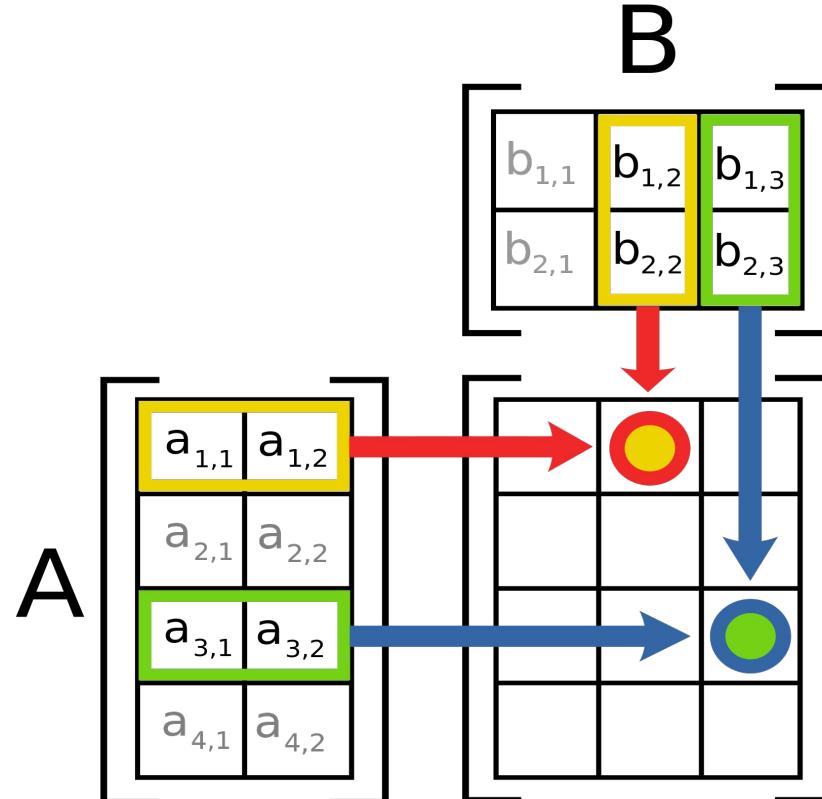
stdin

Standard input is empty

stdout

198198

Multiplication





EPIC

Institute of Technology
Powered by epam

Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A B C

A, B and C are square metrices of size $N \times N$

a, b, c and d are submatrices of A, of size $N/2 \times N/2$

e, f, g and h are submatrices of B, of size $N/2 \times N/2$



EPIC

Institute of Technology
Powered by epam

Multiplication

```
static void MatrixMultIJK() {  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < n; j++) {  
            int sum = 0;  
            for (int k = 0; k < n; k++) {  
                sum += A[i][k] * B[k][j];  
            }  
            C[i][j] = sum;  
        }  
    }  
}
```

```
78. int main() {
79.     for (int i = 0; i < n; i++) {
80.         for (int j = 0; j < n; j++) {
81.             A[i][j] = (i + j) % 5;
82.             B[i][j] = (i + j) % 5;
83.         }
84.     }
85.     MatrixMultIJK();
86.     cout << "A : \n";
87.     for (int i = 0; i < n; i++) {
88.         for (int j = 0; j < n; j++) {
89.             cout << A[i][j] << " ";
90.         }
91.         cout << "\n";
92.     }
93.     cout << "B : \n";
94.     for (int i = 0; i < n; i++) {
95.         for (int j = 0; j < n; j++) {
96.             cout << B[i][j] << " ";
97.         }
98.         cout << "\n";
99.     }
100.    cout << "C : \n";
101.    for (int i = 0; i < n; i++) {
102.        for (int j = 0; j < n; j++) {
103.            cout << C[i][j] << " ";
104.        }
105.        cout << "\n";
106.    }
107. }
```

stdout

```
A :
0 1 2 3 4
1 2 3 4 0
2 3 4 0 1
3 4 0 1 2
4 0 1 2 3
B :
0 1 2 3 4
1 2 3 4 0
2 3 4 0 1
3 4 0 1 2
4 0 1 2 3
C :
30 20 15 15 20
20 30 20 15 15
15 20 30 20 15
15 15 20 30 20
20 15 15 20 30
```

```
static void MatrixMultIJK() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int sum = 0;
            for (int k = 0; k < n; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
}
```

```
static void MatrixMultJIK() {
    for (int j = 0; j < n; j++) {
        for (int i = 0; i < n; i++) {
            int sum = 0;
            for (int k = 0; k < n; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
}
```

```
static void MatrixMultKIJ() {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            int r = A[i][k];
            for (int j = 0; j < n; j++) {
                C[i][j] += r * B[k][j];
            }
        }
    }
}
```

```
static void MatrixMultIKJ() {
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < n; k++) {
            int r = A[i][k];
            for (int j = 0; j < n; j++) {
                C[i][j] += r * B[k][j];
            }
        }
    }
}
```

```
static void MatrixMultJKI() {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            int r = B[k][j];
            for (int i = 0; i < n; i++) {
                C[i][j] += A[i][k] * r;
            }
        }
    }
}
```

```
static void MatrixMultKJI() {
    for (int k = 0; k < n; k++) {
        for (int j = 0; j < n; j++) {
            int r = B[k][j];
            for (int i = 0; i < n; i++) {
                C[i][j] += A[i][k] * r;
            }
        }
    }
}
```



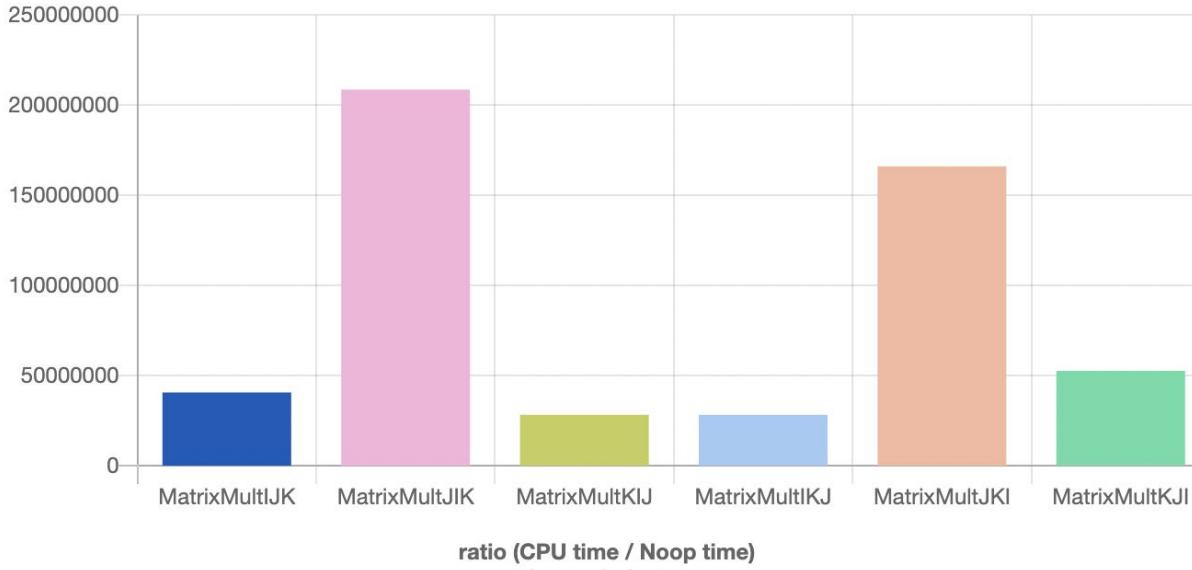
EPIC

Institute of Technology
Powered by epam

Comparison

Charts

Assembly



Show Noop bar





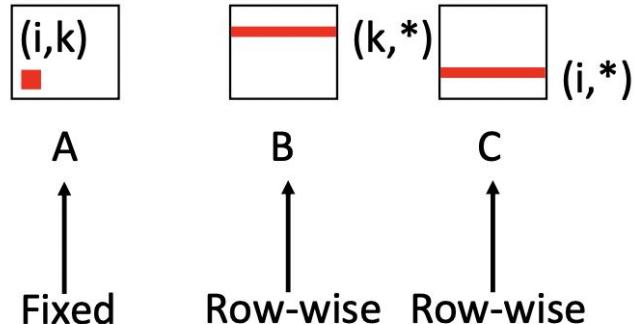
EPIC

Institute of Technology
Powered by epam

Why?

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

Inner loop:



Misses per inner loop iteration:

A
0.0

B
0.25

C
0.25



EPIC

Institute of Technology
Powered by 

That's All Folks!