EPIC
Institute of Technology
Powered by epam

Intro to the subject

Course: Systems Architecture

Lecturer: Gleb Lobanov

February 19, 2024

EPIC
Institute of Technology
Powered by epam

Contents

EPIC
Institute of Technology
Powered by epam

01

Introduction

In this section, there will be a brief introduction about myself and a description of this course.

EPIC
Institute of Technology
Powered by epam

About me

Gleb Lobanov

Current Workplace – B2B YANGO

Tutoring – MIPT, Tinkoff, olympiad schools and so on

Tg/discord - glebodin,
if you need something special – feel free to ask

EPIC Institute of Technology
Powered by epam

Grading

$$M_{\text{final}} = \min(5, round(0.7 \cdot M_{\text{practice}} + 0.3 \cdot M_{\text{individual\_work}} + M_{\text{bonuses}}))$$

$$M_{\text{practice}} = amount\_solved / amount\_necessary * 5$$

$$M_{\text{indivividual\_work}} = 0.4 \cdot M_{\text{midterm}} + 0.6 \cdot M_{\text{exam}}$$

M_{bonuses} may be given on various unpredictable special occasions. Almost never is negative 🙃

EPIC Institute of Technology
Powered by <epam>

Codestyle

Automatic:

- TAB is strictly prohibited
- Tabulation is 4 spaces
- Tabulation style is K&R, Linux Kernel

Manual:

- Meaningful naming
- Meaningful commenting
- One Code style for one file

EPIC
Institute of Technology
Powered by epam

Motivation

- ➔ Job in low-level applications(security, high load and so on)
- ➔ Less problems in every program

EPIC Institute of Technology
Powered by <epam>

What pitfalls can you meet if you skip this course?

- ➔ Wrong understanding of asymptotics
- ➔ Bad memory usage in your programs (for example Cache Miss).
- ➔ Not knowing and not using SIMD, AVX and so on
- ➔ Leaky abstraction

EPIC
Institute of Technology
Powered by epam

02

Why O-notation is bad

In this section, we will explain why O-notation doesn't always apply in reality.

EPIC Institute of Technology
Powered by epam

Problems of O-notation

- ➔ Different programming languages
- ➔ Different processor architecture
- ➔ Different cost of operations

EPIC Institute of Technology
Powered by <epam>

Python

```
1.  n = 128
2.  d = [[0] * n] * n
3.
4.  for i in range(n):
5.    for j in range(n):
6.      d[i][j] = i + j
7.
8.  for k in range(n):
9.    for i in range(n):
10.     for j in range(n):
11.       d[i][j] = min(d[i][j], d[i][k] + d[k][j])
12.
```

Success #stdin #stdout 1.2s 9732KB

https://ideone.com/3cU8kk

EPIC Institute of Technology
Powered by <epam>

C++

```cpp
#include <iostream>

using namespace std;
const int n = 1024;
int d[n][n];

int main() {

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      d[i][j] = i + j;
    }
  }

  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
      }
    }
  }

}
```

Success #stdin #stdout 1.28s 7672KB

https://ideone.com/2yAssk

EPIC
Institute of Technology
Powered by epam

Rust

```rust
use std::cmp;

fn main() {
    const n: usize = 1024;
    let mut d = [[0; n]; n];
    for i in 0 .. n - 1 {
        for j in 0 .. n - 1 {
            d[i][j] = i + j
        }
    }
    for k in 0 .. n - 1 {
        for i in 0 .. n - 1 {
            for j in 0 .. n - 1 {
                d[i][j] = cmp::min(d[i][j], d[i][k] + d[k][j])
            }
        }
    }
}

```

Success #stdin #stdout 1.13s 10064KB

stdin

```
Standard input is empty
```

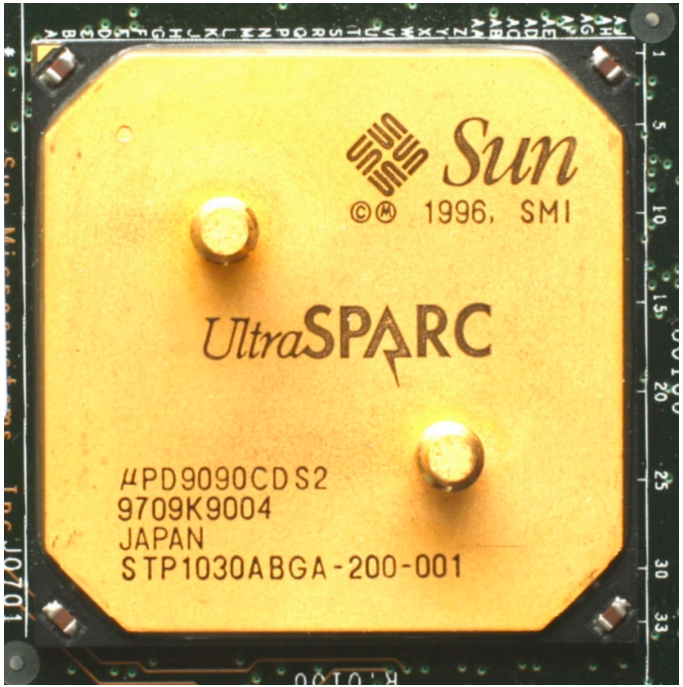
stdout

```
Standard output is empty
```

https://ideone.com/nc5mqp

EPIC
Institute of Technology
Powered by <epam>

Computer architecture



→ RISC

→ CISC

→ MISC

EPIC Institute of Technology
Powered by epam

Tricky example

- Intel

  compare_exchange

  https://en.wikipedia.org/wiki/Load-link/store-conditional

- ARM

  LL/SC

  https://en.wikipedia.org/wiki/Load-link/store-conditional

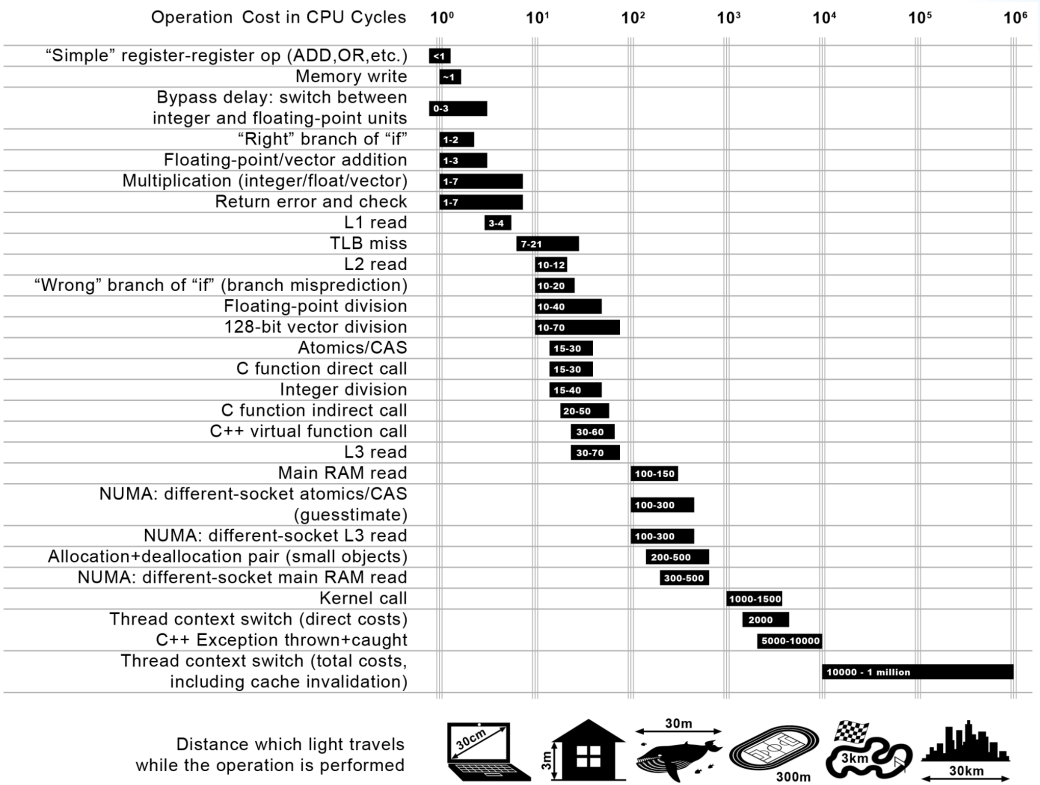
EPIC Institute of Technology
Powered by epam

Elementary operations in O-notation

- Bit operations (bitwise and, bitwise or, …);
- arithmetic operations (add, multiply, divide, ...);
- some comparison operations (compare, greater, less, ..);
- some logical operations (and, or, …);
- some memory operations (take from memory, put to memory, ...);
- some IO operations (read, write, ...);
- If, exception and more other.

EPIC Institute of Technology
Powered by <epam>

Not all CPU operations are created equal



source

EPIC Institute of Technology
Powered by epam

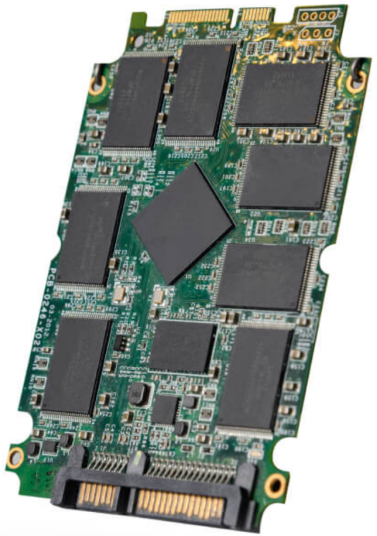
Alternatives

- PRAM
- External Memory Model

EPIC
Institute of Technology
Powered by <epam>

HDD vs SSD

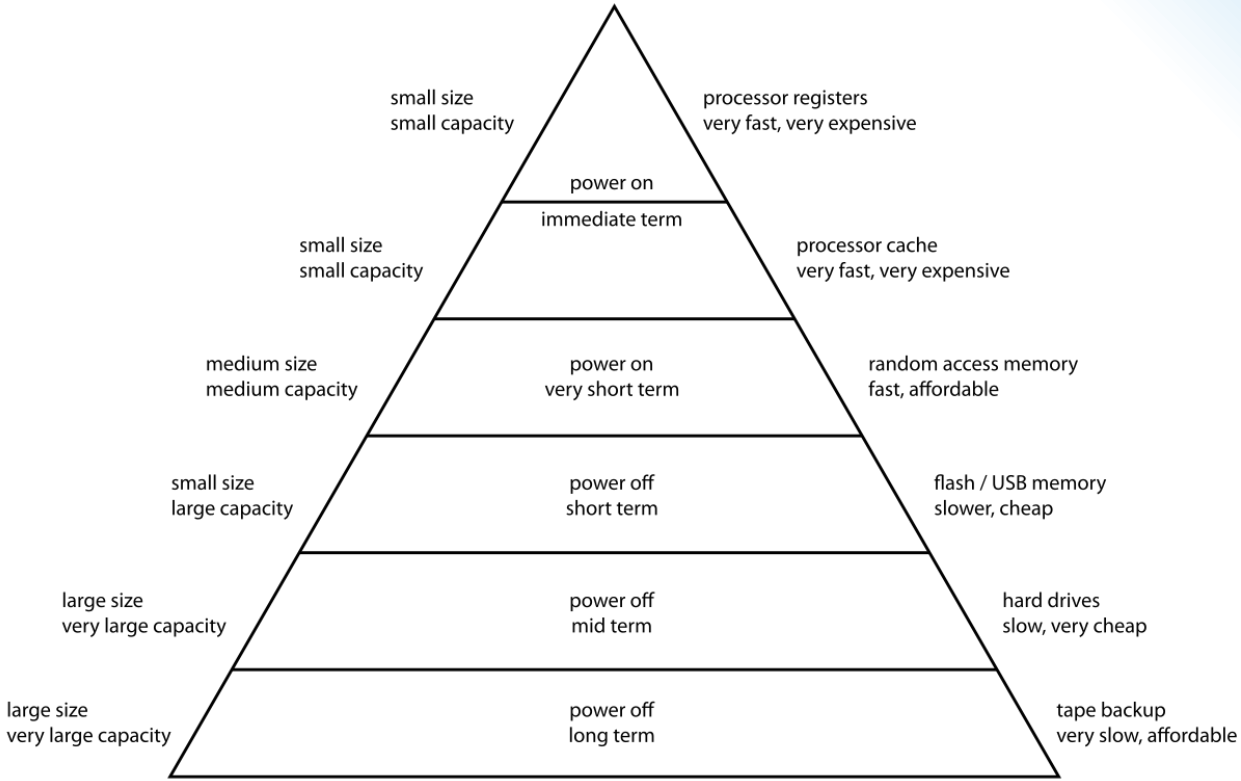
HDD



SSD

EPIC
Institute of Technology
Powered by <epam>

Computer Memory Hierarchy



EPIC
Institute of Technology
Powered by epam

Real Life

Here is an approximate comparison table for commodity hardware in 2021

Type	M	B	Latency	Bandwidth	\$/GB/mo[1]
L1	10K	64B	2ns	80G/s	-
L2	100K	64B	5ns	40G/s	-
L3	1M/core	64B	20ns	20G/s	-
RAM	GBs	64B	100ns	10G/s	1.5
SSD	TBs	4K	0.1ms	5G/s	0.17
HDD	TBs	-	10ms	1G/s	0.04
S3	∞	-	150ms	∞	0.02[2]

EPIC
Institute of Technology
Powered by <epam>

Caches as beer

Holly Cummins (holly_cummins@hachyderm.io)
@holly_cummins · **Follow**

L1 cache is a beer in hand, L3 is fridge, main memory is walking to the store, disk access is flying to another country for beer. @net0pyr

EPIC
Institute of Technology
Powered by epam

03

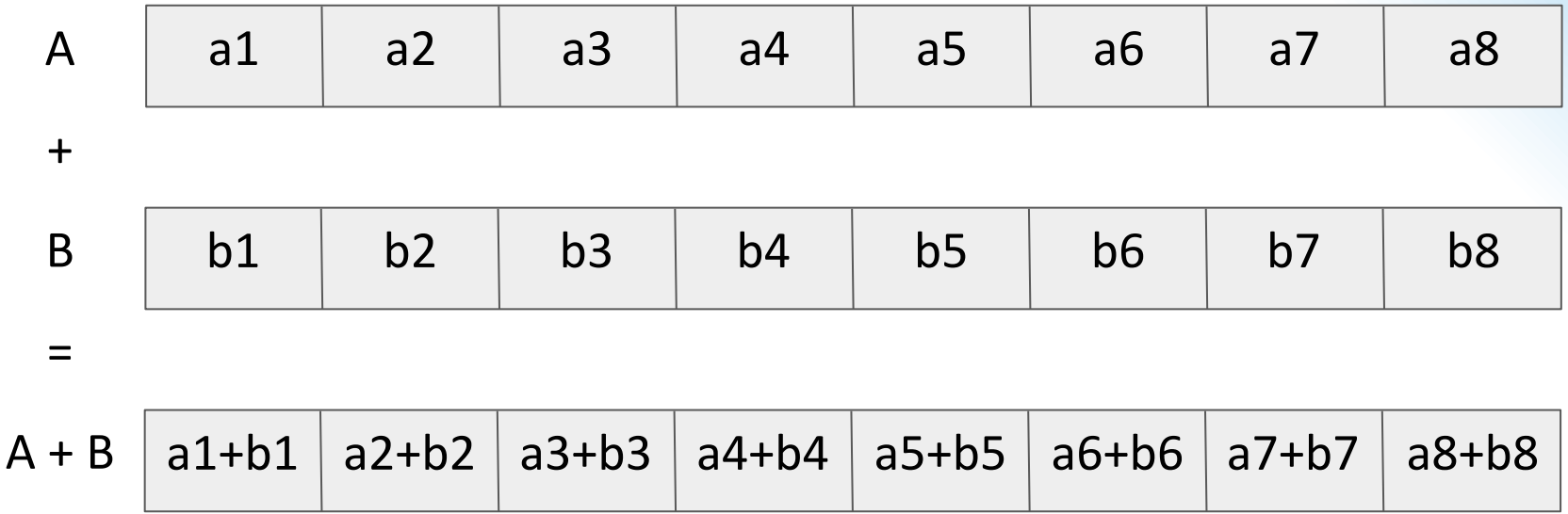
AVX

In this section, we will discuss vectorized computations.

EPIC
Institute of Technology
Powered by epam

type(ai) = type(bi) = double

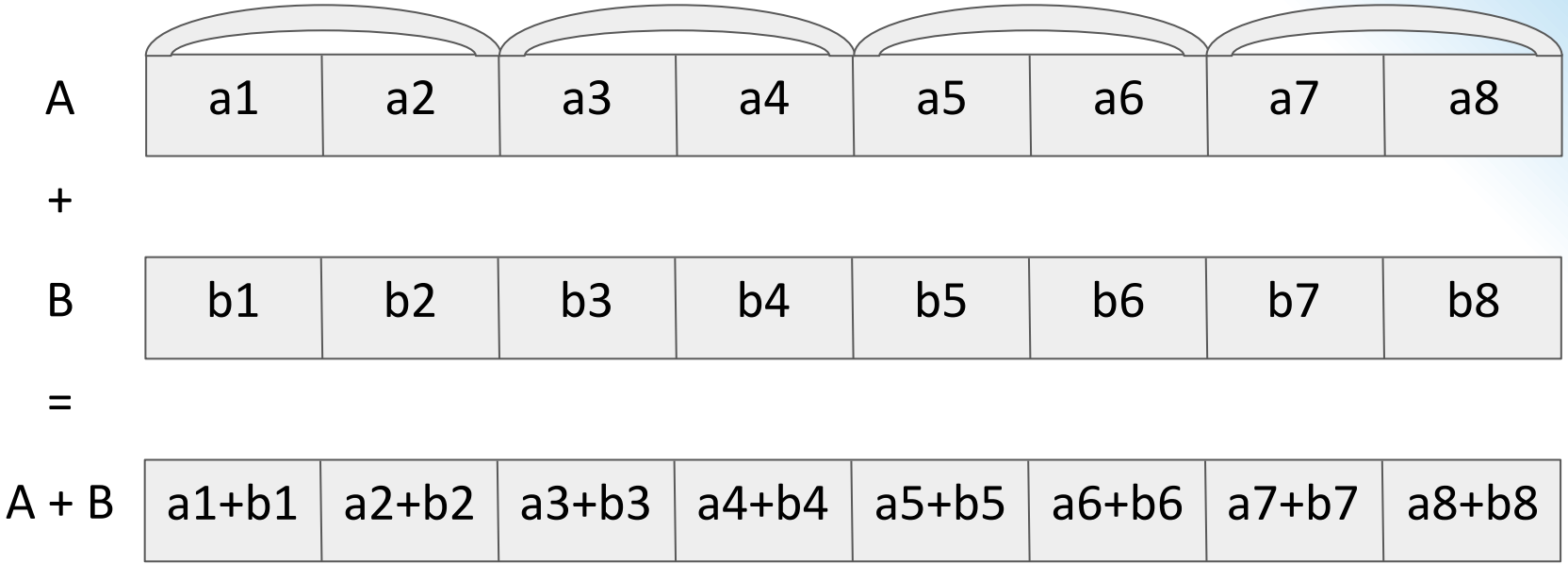
Regular



EPIC
Institute of Technology
Powered by epam

type(ai) = type(bi) = double

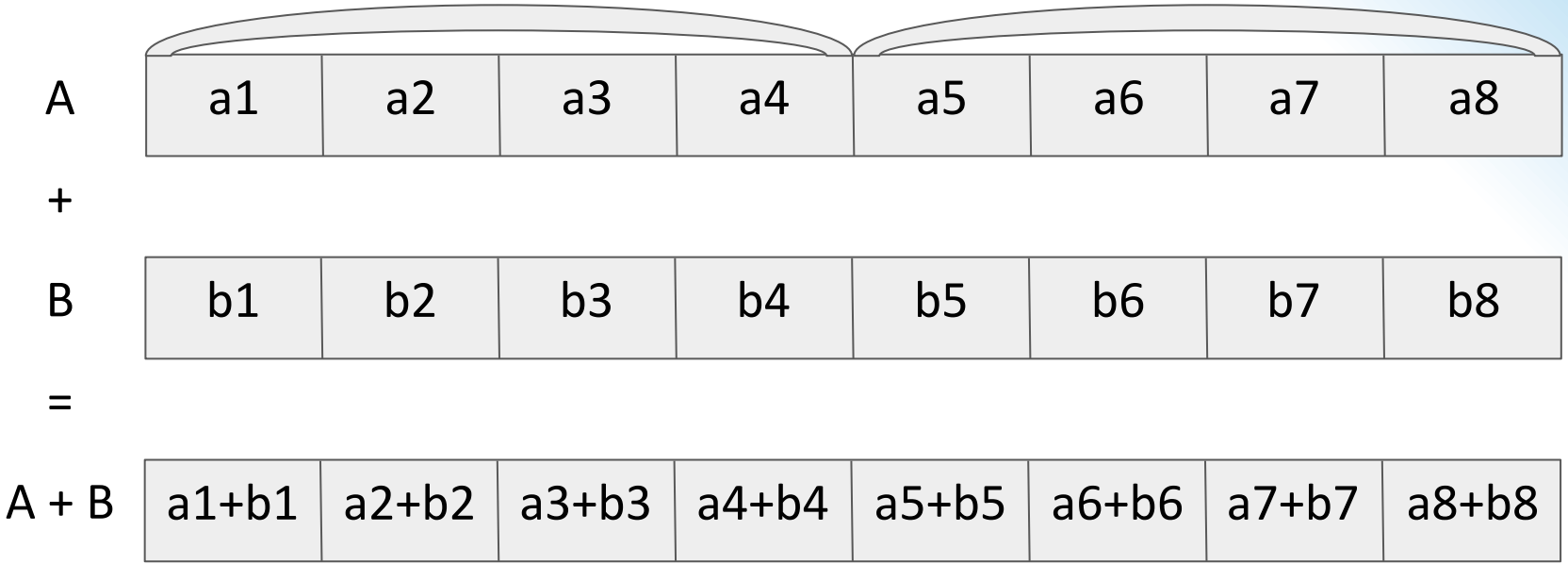
SSE



EPIC
Institute of Technology
Powered by epam

type(ai) = type(bi) = double

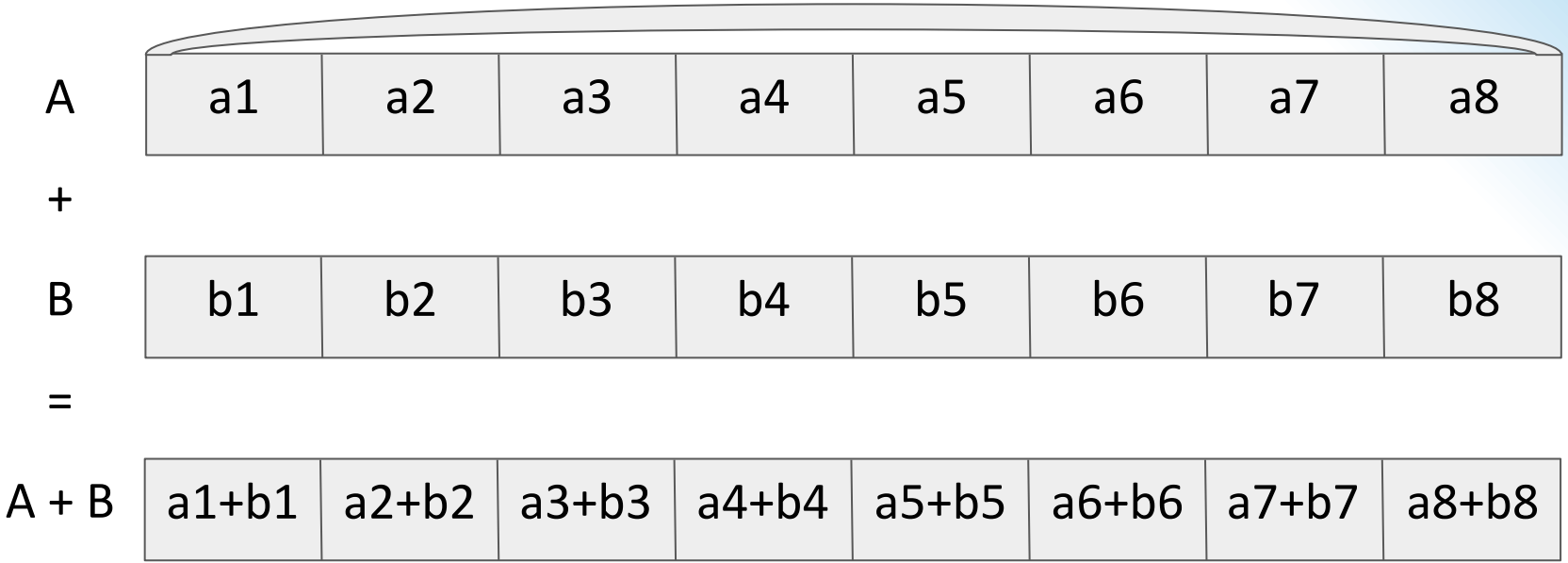
AVX



EPIC
Institute of Technology
Powered by epam

type(ai) = type(bi) = double

AVX512



Instruction Set

- [ ] MMX
- [ ] SSE family
- [ ] AVX family
- [ ] AVX-512 family
- [ ] AMX family
- [ ] SVML
- [ ] Other

Categories

- [ ] Application-Targeted
- [ ] Arithmetic
- [ ] Bit Manipulation
- [ ] Cast
- [ ] Compare
- [ ] Convert
- [ ] Cryptography
- [ ] Elementary Math Functions
- [ ] General Support
- [ ] Load
- [ ] Logical
- [ ] Mask
- [ ] Miscellaneous
- [ ] Move
- [ ] OS-Targeted
- [ ] Probability/Statistics
- [ ] Random

Search Intel Intrinsics

source link

```
void _mm_2intersect_epi32 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2)                vp2intersectd
void _mm256_2intersect_epi32 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2)             vp2intersectd
void _mm512_2intersect_epi32 (__m512i a, __m512i b, __mmask16* k1, __mmask16*               vp2intersectd
k2)
void _mm_2intersect_epi64 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2)                vp2intersectq
void _mm256_2intersect_epi64 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2)             vp2intersectq
void _mm512_2intersect_epi64 (__m512i a, __m512i b, __mmask8* k1, __mmask8* k2)             vp2intersectq
void _aadd_i32 (int* __A, int __B)                                                          aadd
void _aadd_i64 (__int64* __A, __int64 __B)                                                  aadd
void _aand_i32 (int* __A, int __B)                                                          aand
void _aand_i64 (__int64* __A, __int64 __B)                                                  aand
__m128i _mm_abs_epi16 (__m128i a)                                                           pabsw
__m128i _mm_mask_abs_epi16 (__m128i src, __mmask8 k, __m128i a)                             vpabsw
__m128i _mm_maskz_abs_epi16 (__mmask8 k, __m128i a)                                         vpabsw
__m256i _mm256_abs_epi16 (__m256i a)                                                        vpabsw
__m256i _mm256_mask_abs_epi16 (__m256i src, __mmask16 k, __m256i a)                         vpabsw
__m256i _mm256_maskz_abs_epi16 (__mmask16 k, __m256i a)                                     vpabsw
__m512i _mm512_abs_epi16 (__m512i a)                                                        vpabsw
__m512i _mm512_mask_abs_epi16 (__m512i src, __mmask32 k, __m512i a)                         vpabsw
__m512i _mm512_maskz_abs_epi16 (__mmask32 k, __m512i a)                                     vpabsw
__m128i _mm_abs_epi32 (__m128i a)                                                           pabsd
```

EPIC Institute of Technology
Powered by <epam>

Without AVX

edit fork download

```
1.  #include <bits/stdc++.h>
2.
3.  using namespace std;
4.  const int SIZE = 50000000;
5.
6.  double a[SIZE], b[SIZE], c[SIZE];
7.
8.  int main() {
9.      for (int i = 0; i < SIZE; i++) {
10.         a[i] = b[i] = i;
11.     }
12.     for (int i = 0; i < SIZE; i++) {
13.         c[i] = a[i] + b[i];
14.     }
15.
16.     return 0;
17. }
```

Success #stdin #stdout 0.27s 1175544KB

stdin

```
Standard input is empty
```

stdout

```
Standard output is empty
```

https://ideone.com/5FBzG0

EPIC Institute of Technology
Powered by epam

With AVX

```
1.  #pragma GCC target("avx2")
2.  #include <bits/stdc++.h>
3.  #include <immintrin.h>
4.
5.  using namespace std;
6.  const int SIZE = 50000000;
7.
8.  double a[SIZE], b[SIZE], c[SIZE];
9.
10. int main() {
11.     for (int i = 0; i < SIZE; i += 4) {
12.         a[i] = b[i] = i;
13.     }
14.     for (int i = 0; i < SIZE; i += 4) {
15.         __m256d x = _mm256_loadu_pd(&a[i]);
16.         __m256d y = _mm256_loadu_pd(&b[i]);
17.         __m256d z = _mm256_add_pd(x, y);
18.         _mm256_storeu_pd(&c[i], z);
19.     }
20.
21.     return 0;
22. }
```

Success #stdin #stdout 0.25s 1175288KB

https://ideone.com/4F3CSR

EPIC Institute of Technology
Powered by epam

Real Task

Let's say we have a task:

1) Find the number of numbers < x on the segment
2) Change the number.

In our algorithms course, we've already learned how to do this quite efficiently (using segment tree, Cartesian tree, or a sqrt decomposition).

But what if there's a much simpler way to do it?

EPIC
Institute of Technology
Powered by epam

Real Task

```
if (t == 1) {
            a--;
            int l = (a / d) * d;
            int r = min(n, l + d);
            int pb = lower_bound(g + l, g + r, s[a]) - g;
            if (s[a] < b) {
                for (; pb < r - 1 && g[pb + 1] < b; pb++)
{                   g[pb] = g[pb + 1];
                }
                g[pb] = b;
            } else {
                for (; pb > l && g[pb - 1] > b; pb--) {
                    g[pb] = g[pb - 1];
                }
                g[pb] = b;
            }
            s[a] = b;
        } else {
            cin >> c;
            b--;
            int l = (b / d) * d;
            int r = min(n, l + d);
            int ans = 0;
            if (b > ((l + r) >> 1) || r >= c) {
                int t = min(r, c);
                for (int i = b; i < t; i++) {
                    if (s[i] < a) {f
                        ans--;
                    }
                }
            }
```

EPIC
Institute of Technology
Powered by epam

Real Task

```
cin >> tp;
if (tp == 1) {
    int i, x;
    cin >> i >> x;
    a[i - 1] = x;
} else {
    int m, l, r;
    cin >> m >> l >> r;
    int cl = 0;
    for (int i = l - 1; i < r; ++i)
        cl += a[i] < m;
```

EPIC Institute of Technology
Powered by <epam>

Comparison

	nsqrt_opt.cpp	stupid_solution.cpp
	OK 0 / 1	OK 0 / 0
	OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000
	OK 15 / 1 / 1.00000	OK 0 / 0 / 1.00000
	OK 15 / 1 / 1.00000	OK 15 / 0 / 1.00000
	OK 467 / 1 / 1.00000	OK 3946 / 0 / 1.00000
	OK 1122 / 1 / 1.00000	OK 7675 / 0 / 1.00000
	OK 1107 / 1 / 1.00000	TL 15000 / 0 / 1.00000
	OK 763 / 1 / 1.00000	OK 4055 / 0 / 1.00000
	OK 15 / 1 / 1.00000	OK 0 / 0 / 1.00000
	OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000
	OK 0 / 1 / 1.00000	OK 15 / 0 / 1.00000
	OK 483 / 1 / 1.00000	OK 2901 / 0 / 1.00000
	OK 1076 / 1 / 1.00000	TL 15000 / 0 / 1.00000
	OK 1075 / 1 / 1.00000	TS 0 / 0 / 1.00000
	OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000
	OK 1091 / 1 / 1.00000	TS 0 / 0 / 1.00000
	OK 46 / 1 / 1.00000	OK 187 / 0 / 1.00000
	OK 15 / 1 / 1.00000	OK 78 / 0 / 1.00000
	OK 1060 / 1 / 1.00000	TS 0 / 0 / 1.00000
	OK 1076 / 1 / 1.00000	TL 15000 / 0 / 1.00000
	OK 171 / 1 / 1.00000	OK 919 / 0 / 1.00000

EPIC
Institute of Technology
Powered by epam

Eazy AVX

```
1  #pragma GCC target("avx2")
2  #pragma GCC optimize("O3")
```

EPIC
Institute of Technology
Powered by <epam>

nsqrt_opt.cpp	pragma_avx.cpp	stupid_solution.cpp
OK 0 / 1	OK 0 / 0	OK 0 / 0
OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000	OK 0 / 0 / 1.00000
OK 15 / 1 / 1.00000	OK 0 / 0 / 1.00000	OK 0 / 0 / 1.00000
OK 15 / 1 / 1.00000	OK 15 / 0 / 1.00000	OK 15 / 0 / 1.00000
OK 467 / 1 / 1.00000	OK 1029 / 0 / 1.00000	OK 3946 / 0 / 1.00000
OK 1122 / 1 / 1.00000	OK 1949 / 0 / 1.00000	OK 7675 / 0 / 1.00000
OK 1107 / 1 / 1.00000	OK 5444 / 0 / 1.00000	TL 15000 / 0 / 1.00000
OK 763 / 1 / 1.00000	OK 1091 / 0 / 1.00000	OK 4055 / 0 / 1.00000
OK 15 / 1 / 1.00000	OK 0 / 0 / 1.00000	OK 0 / 0 / 1.00000
OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000	OK 0 / 0 / 1.00000
OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000	OK 15 / 0 / 1.00000
OK 483 / 1 / 1.00000	OK 732 / 0 / 1.00000	OK 2901 / 0 / 1.00000
OK 1076 / 1 / 1.00000	OK 5287 / 0 / 1.00000	TL 15000 / 0 / 1.00000
OK 1075 / 1 / 1.00000	OK 5444 / 0 / 1.00000	TL 15000 / 0 / 1.00000
OK 0 / 1 / 1.00000	OK 0 / 0 / 1.00000	OK 0 / 0 / 1.00000
OK 1091 / 1 / 1.00000	OK 5101 / 0 / 1.00000	TL 15000 / 0 / 1.00000
OK 46 / 1 / 1.00000	OK 61 / 0 / 1.00000	OK 187 / 0 / 1.00000
OK 15 / 1 / 1.00000	OK 30 / 0 / 1.00000	OK 78 / 0 / 1.00000
OK 1060 / 1 / 1.00000	OK 5101 / 0 / 1.00000	TL 15000 / 0 / 1.00000
OK 1076 / 1 / 1.00000	OK 4976 / 0 / 1.00000	TL 15000 / 0 / 1.00000
OK 171 / 1 / 1.00000	OK 249 / 0 / 1.00000	OK 919 / 0 / 1.00000

AVX

```
}
__m256i mm = _mm256_set1_epi32(m);
__m256i ansm = _mm256_set1_epi32(0);
for (int i = l; i + B <= r; i += B) {
    __m256i b0 = _mm256_lddqu_si256(reinterpret_cast<__m256i *>(a + i));
    __m256i b1 = _mm256_lddqu_si256(reinterpret_cast<__m256i *>(a + i + 8));
    b0 = _mm256_cmpgt_epi32(mm, b0);
    b1 = _mm256_cmpgt_epi32(mm, b1);
    ansm = _mm256_sub_epi32(ansm, b0);
    ansm = _mm256_sub_epi32(ansm, b1);
}
ans += _mm256_extract_epi32(ansm, 0);
ans += _mm256_extract_epi32(ansm, 1);
ans += _mm256_extract_epi32(ansm, 2);
ans += _mm256_extract_epi32(ansm, 3);
ans += _mm256_extract_epi32(ansm, 4);
ans += _mm256_extract_epi32(ansm, 5);
ans += _mm256_extract_epi32(ansm, 6);
ans += _mm256_extract_epi32(ansm, 7);
```

EPIC Institute of Technology
Powered by epam

Extract

```
__int32 _mm256_extract_epi32 (__m256i a, const int index)
```

Synopsis

```
__int32 _mm256_extract_epi32 (__m256i a, const int index)
#include <immintrin.h>
Instruction: Sequence
CPUID Flags: AVX
```

Description

Extract a 32-bit integer from `a`, selected with `index`, and store the result in `dst`.

Operation

```
dst[31:0] := (a[255:0] >> (index[2:0] * 32))[31:0]
```

EPIC Institute of Technology
Powered by epam

Set

```
__m256i _mm256_set1_epi32 (int a)
```

Synopsis

```
__m256i _mm256_set1_epi32 (int a)
#include <immintrin.h>
Instruction: Sequence
CPUID Flags: AVX
```

Description

Broadcast 32-bit integer `a` to all elements of `dst`. This intrinsic may generate the `vpbroadcastd`.

Operation

```
FOR j := 0 to 7
        i := j*32
        dst[i+31:i] := a[31:0]
ENDFOR
dst[MAX:256] := 0
```

EPIC Institute of Technology
Powered by epam

Load

```
__m256i _mm256_lddqu_si256 (__m256i const * mem_addr)
```

vlddqu

Synopsis

```
__m256i _mm256_lddqu_si256 (__m256i const * mem_addr)
#include <immintrin.h>
Instruction: vlddqu ymm, m256
CPUID Flags: AVX
```

Description

Load 256-bits of integer data from unaligned memory into `dst`. This intrinsic may perform better than `_mm256_loadu_si256` when the data crosses a cache line boundary.

Operation

```
dst[255:0] := MEM[mem_addr+255:mem_addr]
dst[MAX:256] := 0
```

EPIC Institute of Technology
Powered by epam

Compare

```
__m256i _mm256_cmpgt_epi32 (__m256i a, __m256i b)
```

Synopsis

```
__m256i _mm256_cmpgt_epi32 (__m256i a, __m256i b)
#include <immintrin.h>
Instruction: vpcmpgtd ymm, ymm, ymm
CPUID Flags: AVX2
```

Description

Compare packed signed 32-bit integers in `a` and `b` for greater-than, and store the results in `dst`.

Operation

```
FOR j := 0 to 7
        i := j*32
        dst[i+31:i] := ( a[i+31:i] > b[i+31:i] ) ? 0xFFFFFFFF : 0
ENDFOR
dst[MAX:256] := 0
```

AVX

honest_avx_opt.cpp	nsqrt_opt.cpp
OK 15 / 1	OK 0 / 1
OK 0 / 1 / 1.00000	OK 0 / 1 / 1.00000
OK 0 / 1 / 1.00000	OK 15 / 1 / 1.00000
OK 15 / 1 / 1.00000	OK 15 / 1 / 1.00000
OK 654 / 5 / 1.00000	OK 467 / 1 / 1.00000
OK 904 / 5 / 1.00000	OK 1122 / 1 / 1.00000
OK 1622 / 5 / 1.00000	OK 1107 / 1 / 1.00000
OK 717 / 5 / 1.00000	OK 763 / 1 / 1.00000
OK 0 / 1 / 1.00000	OK 15 / 1 / 1.00000
OK 0 / 1 / 1.00000	OK 0 / 1 / 1.00000
OK 0 / 1 / 1.00000	OK 0 / 1 / 1.00000
OK 452 / 5 / 1.00000	OK 483 / 1 / 1.00000
OK 1621 / 5 / 1.00000	OK 1076 / 1 / 1.00000
OK 1590 / 5 / 1.00000	OK 1075 / 1 / 1.00000
OK 0 / 1 / 1.00000	OK 0 / 1 / 1.00000
OK 1590 / 5 / 1.00000	OK 1091 / 1 / 1.00000
OK 62 / 1 / 1.00000	OK 46 / 1 / 1.00000
OK 15 / 1 / 1.00000	OK 15 / 1 / 1.00000
OK 1575 / 5 / 1.00000	OK 1060 / 1 / 1.00000
OK 1544 / 5 / 1.00000	OK 1076 / 1 / 1.00000
OK 171 / 2 / 1.00000	OK 171 / 1 / 1.00000

EPIC Institute of Technology
Powered by epam

Usage

- Numpy

  https://numpy.org/devdocs/reference/simd/index.html

- Gaming

  https://www.techpowerup.com/forums/threads/avx-vs-non-avx.265592/

- Many life situations

EPIC
Institute of Technology
Powered by epam

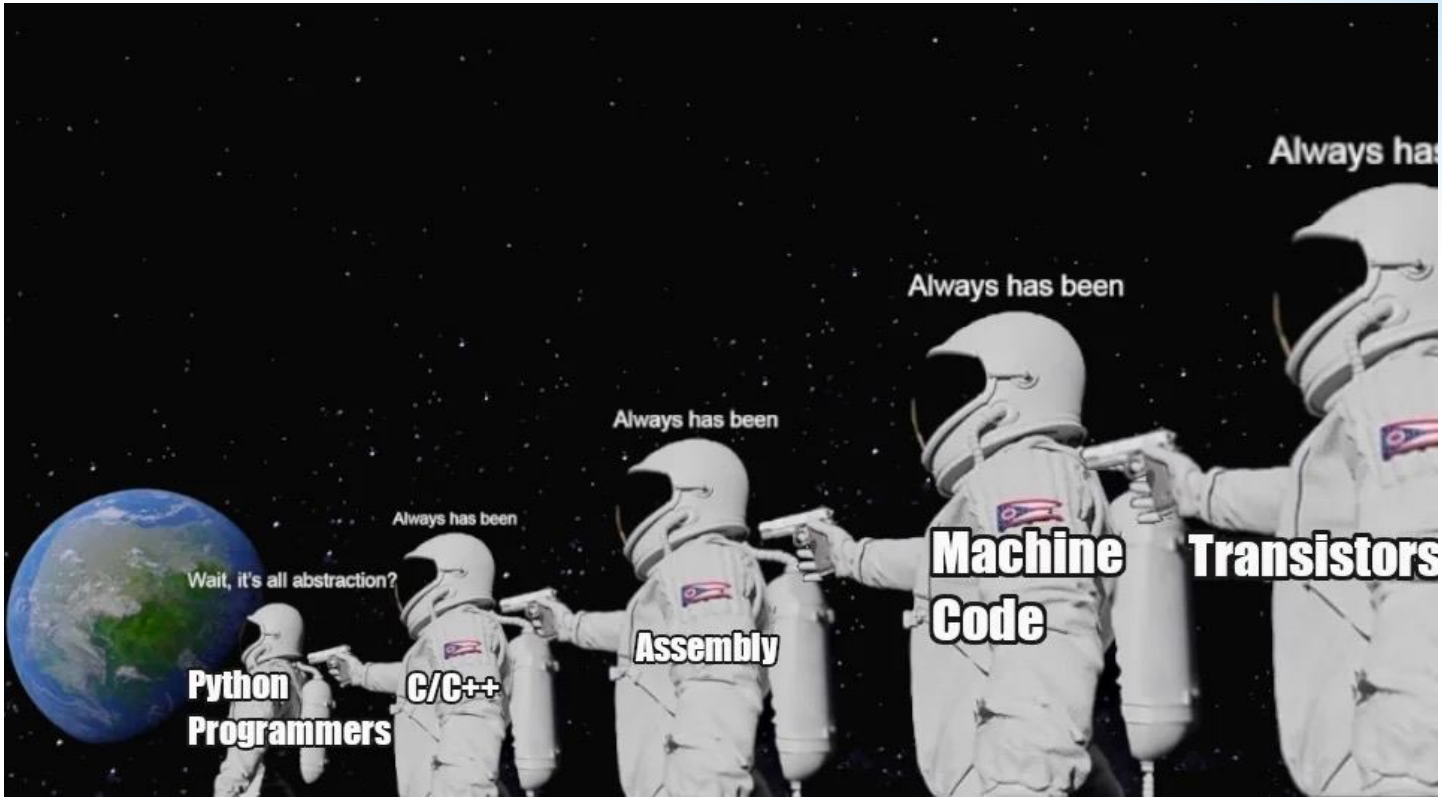
04

Leaky abstraction

In this section, we will discuss vectorized computations.

EPIC
Institute of Technology
Powered by <epam>

Leaky abstraction



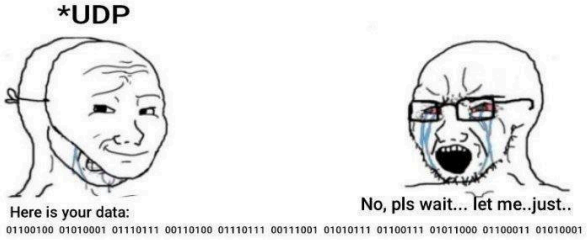
EPIC
Institute of Technology
Powered by epam

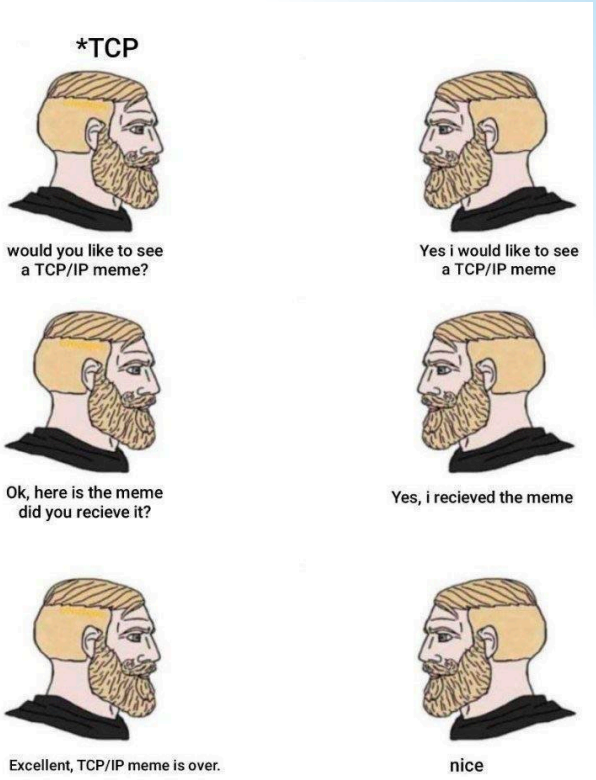
Vector

The storage of the vector is handled automatically, being expanded as needed. Vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. This way a vector does not need to reallocate each time an element is inserted, but only when the additional memory is exhausted. The total amount of allocated memory can be queried using `capacity()` function. Extra memory can be returned to the system via a call to `shrink_to_fit()`. (since C++11)

EPIC
Institute of Technology
Powered by <epam>

TCP vs UDP





EPIC
Institute of Technology
Powered by epam

Premature optimization

“Premature optimization is the root of all evil”

- Knuth

EPIC Institute of Technology
Powered by epam

PyPy

Source:

```
1  n = 128
2  d = [[0] * n] * n
3
4  for i in range(n):
5    for j in range(n):
6      d[i][j] = i + j
7
8  for k in range(n):
9    for i in range(n):
10     for j in range(n):
11       d[i][j] = min(d[i][j], d[i][k] + d[k][j])
12
```

Switch off editor Tab size: 4

Run

Language: PyPy 3.9.10 (7.3.9, 64bit)

Input:

Choose file No file chosen

No more than 256 KB

Output:

```
=====
Used: 46 ms, 2104 KB
```

First 255 bytes only

EPIC Institute of Technology
Powered by epam

Without AVX

```
1.  #pragma GCC target("avx2")
2.  #pragma GCC optimize("O2,unroll-loops")
3.
4.  #include <bits/stdc++.h>
5.  #include <immintrin.h>
6.
7.  using namespace std;
8.  const int SIZE = 50000000;
9.
10. double a[SIZE], b[SIZE], c[SIZE];
11.
12. int main() {
13.     for (int i = 0; i < SIZE; i++) {
14.         a[i] = b[i] = i;
15.     }
16.     for (int i = 0; i < SIZE; i++) {
17.         c[i] = a[i] + b[i];
18.     }
19.
20.     return 0;
21. }
```

https://ideone.com/ceA9iU

Success #stdin #stdout 0.26s 1175352KB

EPIC
Institute of Technology
Powered by epam

That's All Folks!