Lecture 12: SIMD and Vectors

COSC 273: Parallel and Distributed Computing

Spring 2023

Announcements

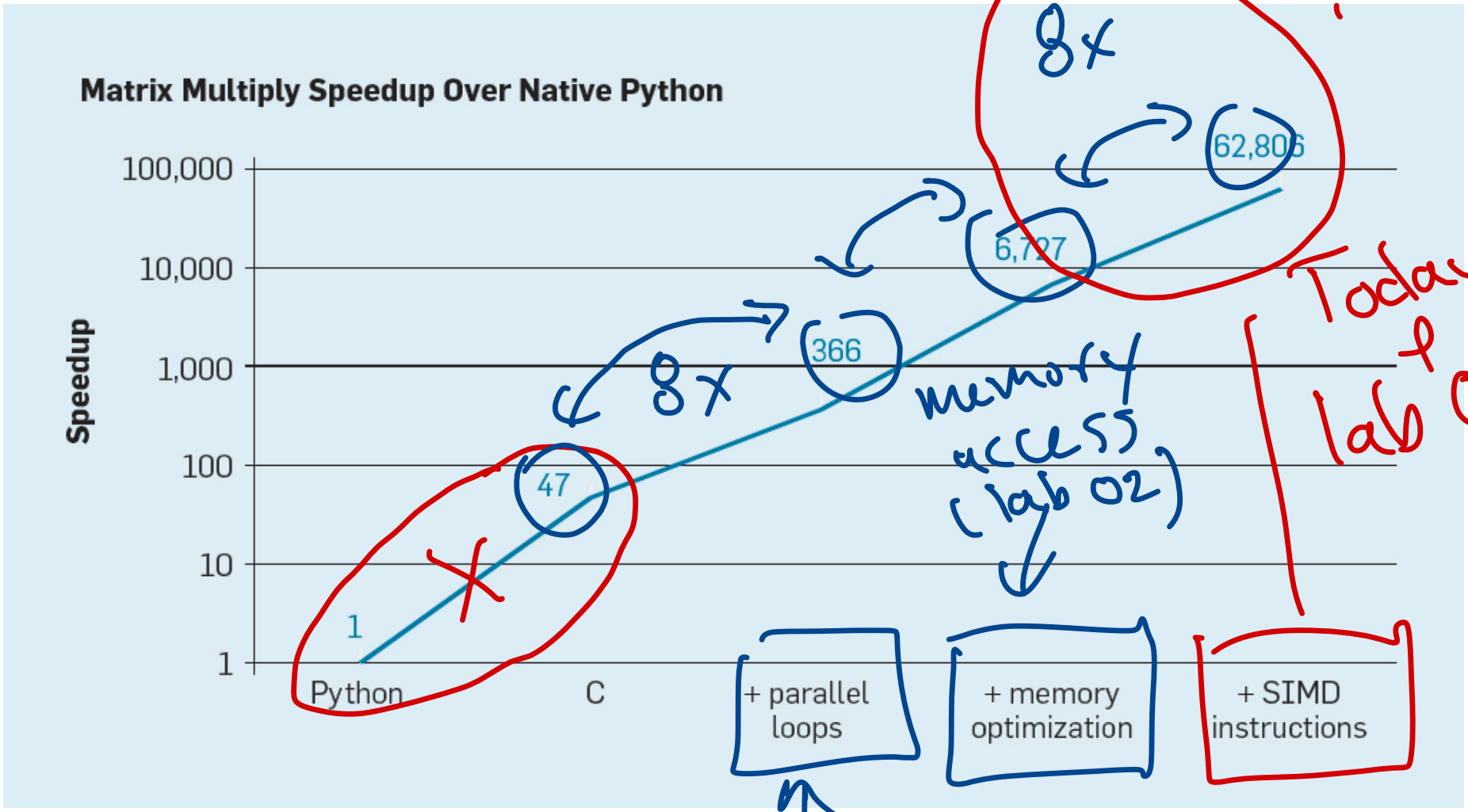
1. Homework 02: Now Due **Next** Friday (03/10)
2. Lab 03 will be due after spring break

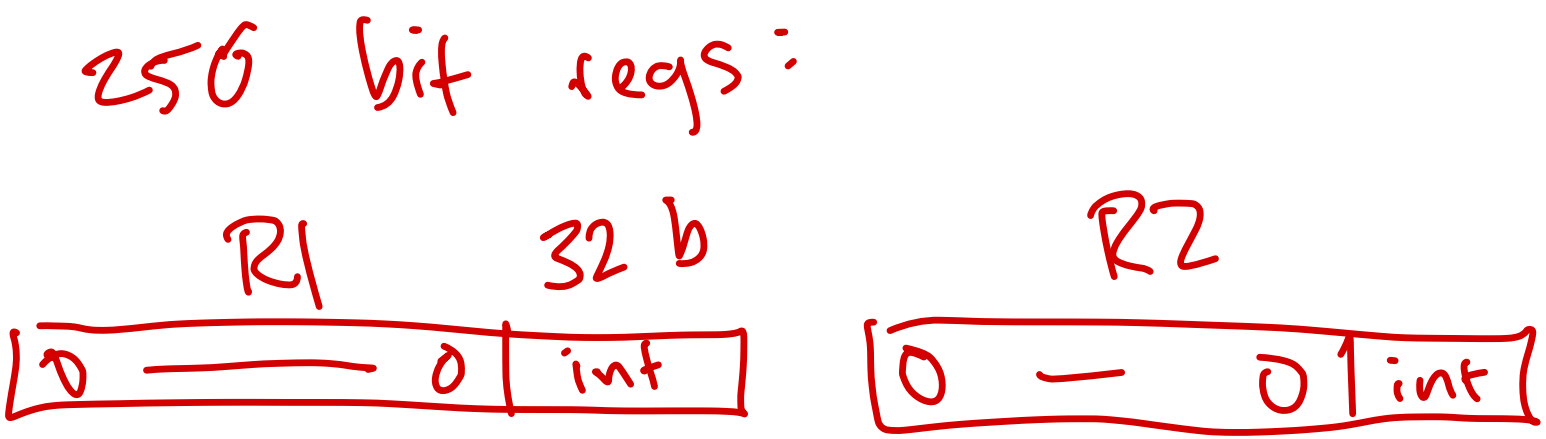
Outline

Single Instruction, Multiple Data

1. Hardware and SIMD instructions
2. Java `Vector` API
3. Benchmarking Notes

Performance, Again



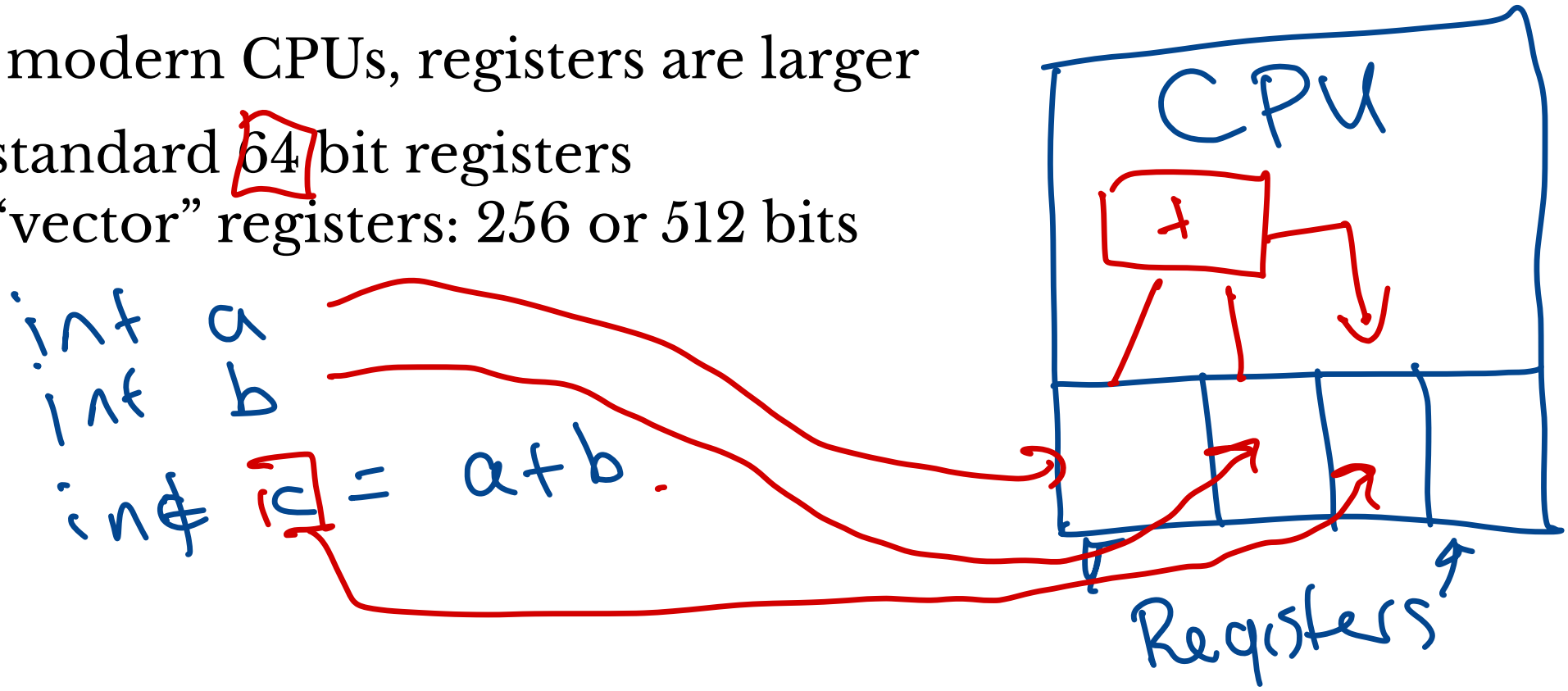


More Powerful Hardware

In `Java`, `int` and `float` values are 32 bits long

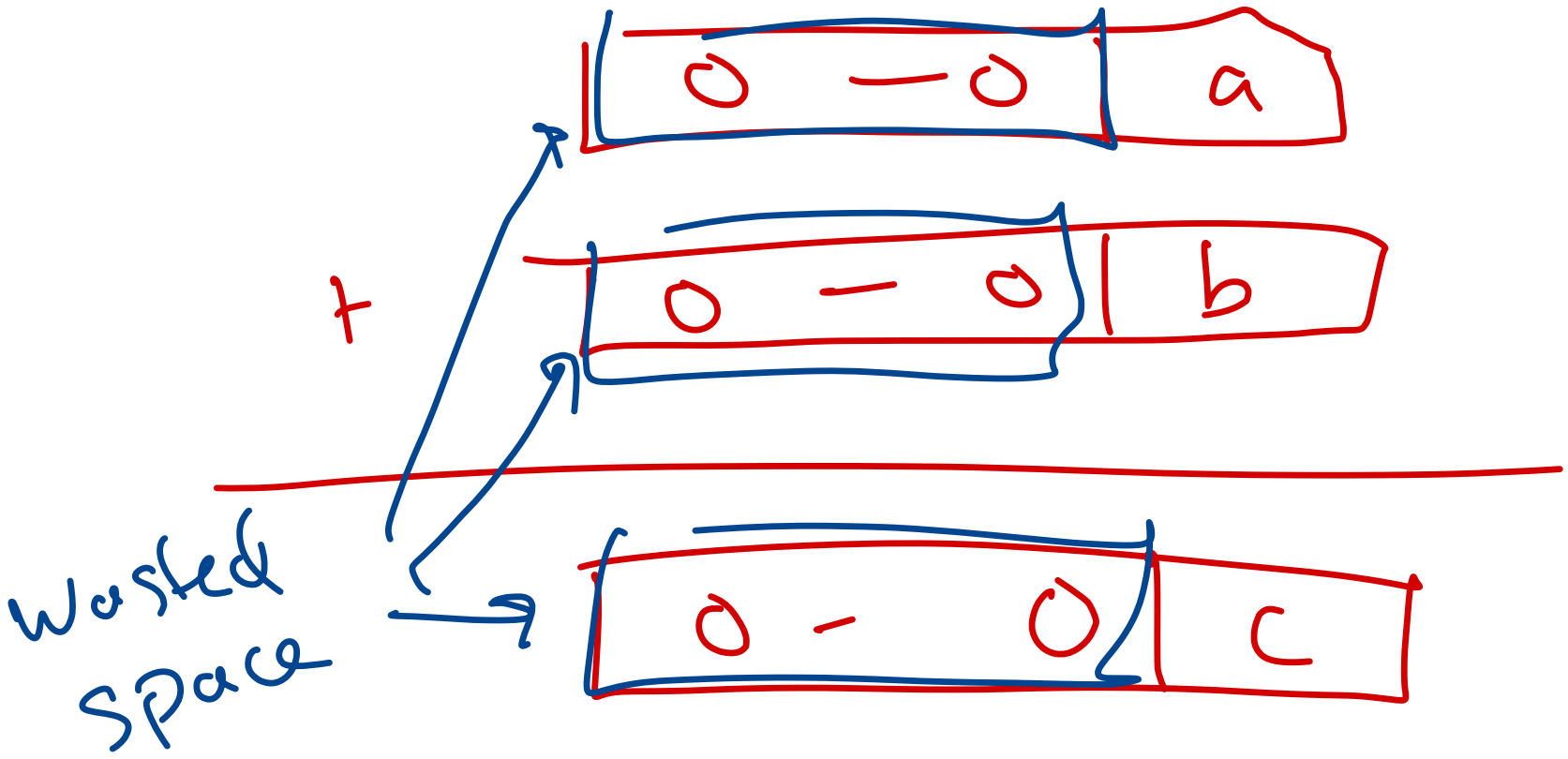
In modern CPUs, registers are larger

- standard 64 bit registers
- "vector" registers: 256 or 512 bits



Naive Operations

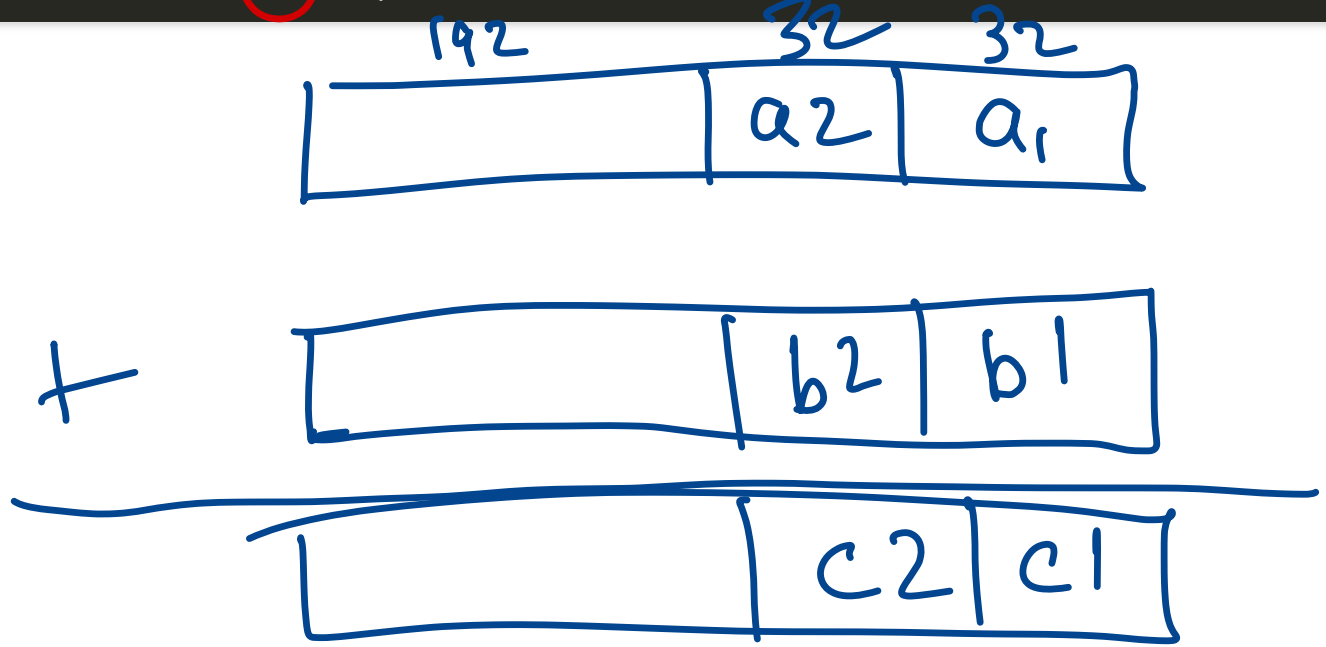
```
int a = 573842;
int b = 3847253;
int c = a + b;
```



SIMD Operations

```
int a1 = 573842;
int b1 = 3847253;
int c1 = a1 + b1;

int a2 = 38657548;
int b2 = 438573;
int c2 = a2 + b2;
```

Single instruction multiple data



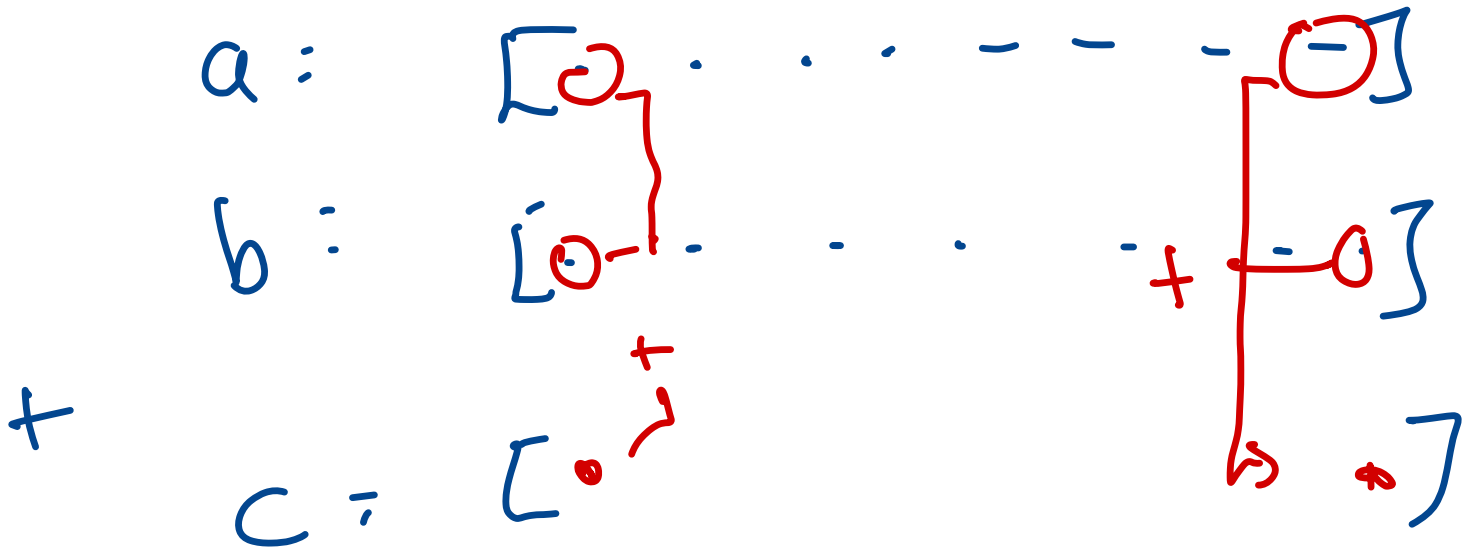
SIMD

Picture of SIMD Registers

Naive Loops

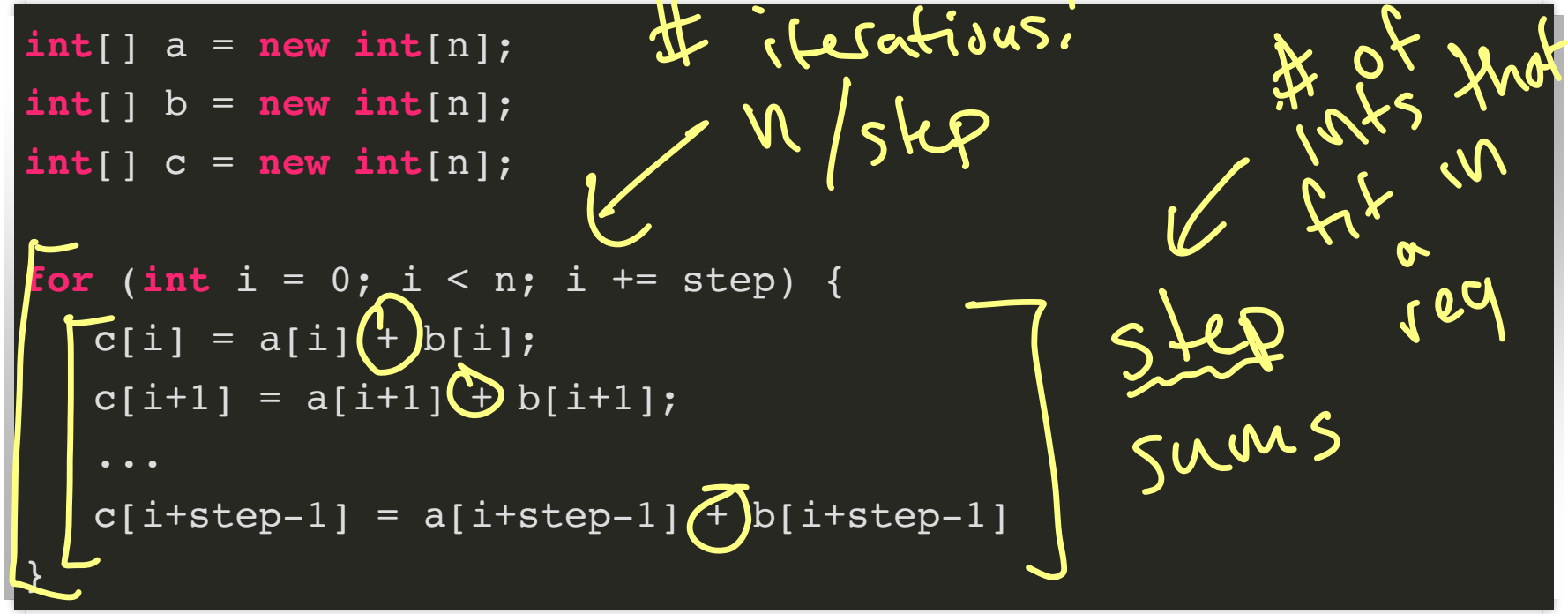
```
int[] a = new int[n];
int[] b = new int[n];
int[] c = new int[n];

for (int i = 0; i < n; i++) {
   c[i] = a[i] + b[i];
}
```

n iterations



Using Full Power

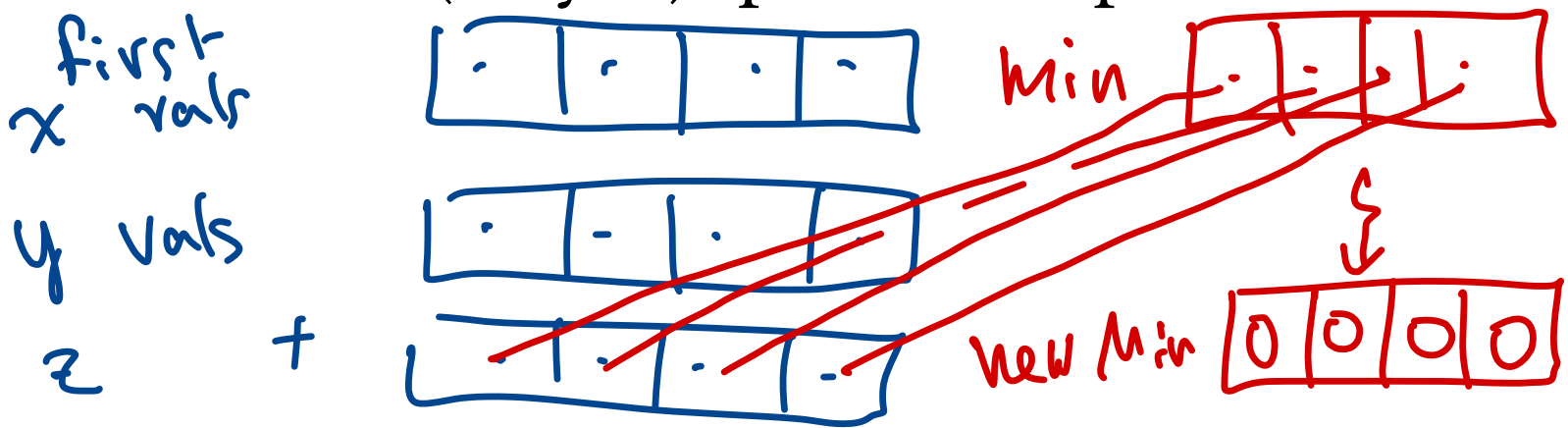
Suppose we can load `step` values into each register



Example from Lab 02

```
float min = Float.MAX_VALUE;
for (int k = 0; k < size; ++k) {
    float x = matrix[i][k];
    float y = matrix[k][j];
    float z = x + y;
    if (z < min) {
        min = z;
    }
}
shortcuts[i][j] = min;
```

Question. How could we (maybe) speed this up with SIMD parallelism?



SIMD Speed-up?

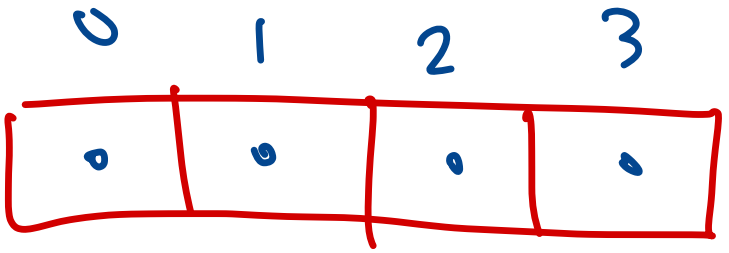
Hopefully ...

Java Vector API

Allows us to specify `Vector` objects

- `Vector` is like fixed-size array
  - elements are **lanes**
- tune `Vector` (bit) size to same as hardware registers
- perform elementary operations on entire vectors

FloatVector :



lanes

4 floats

Java Vector API

Allows us to specify `Vector` objects

- `Vector` is like fixed-size array
  - elements are **lanes**
- tune `Vector` (bit) size to same as hardware registers
- perform elementary operations on entire vectors

Notes:

- `Vector` API in Java 19, available as "incubator"
- Many optimizations already done (without `Vector`)

Example

Find entry-wise maximum of arrays:

```
VectorSpecies<Float> SPECIES = FloatVector.SPECIES_PREFERRED;
    ...
public static float[] vectorMax(float[] a, float[] b) {
    float[] c = new float[a.length];
    int step = SPECIES.length();
    int bound = SPECIES.loopBound(a.length);
    ...
}
```

obj. stores e.g. # lanes

lanes ← largest mult. of # lanes ≤ a.length

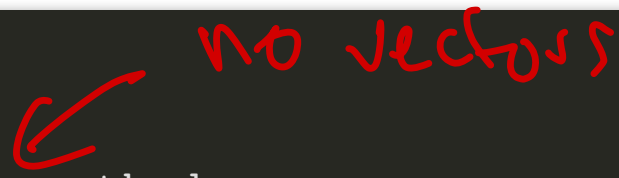
Example Continued

Find entry-wise minimum of arrays:

```
...
    int i = 0;
    for (; i < bound; i += step) {
        var va = FloatVector.fromArray(SPECIES, a, i);
        var vb = FloatVector.fromArray(SPECIES, b, i);
        var vc = va.max(vb);
        vc.intoArray(c, i);
    }
    for (; i < a.length; i++) {
        c[i] = Math.max(a[i], b[i]);
    }
    return c;
```

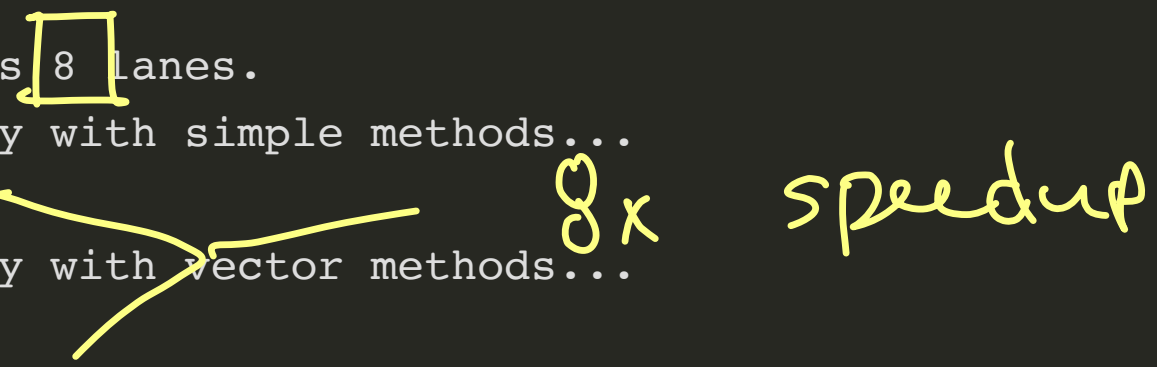
Speedup, Personal Computer

```
Hello, vectors!
The FloatVector has 8 lanes.
Computing max array with simple methods...
That took 625 ms.
Computing max array with vector methods...
That took 174 ms.
The arrays are equal!
```



Speedup, HPC Cluster

```
Hello, vectors!
The FloatVector has 8 lanes.
Computing max array with simple methods...
That took 518 ms.
Computing max array with vector methods...
That took 66 ms.
The arrays are equal!
```



Complications

Java `Vector` API is still an “incubator” feature

- not part of the “standard” language yet
- only available in Java 17+
  - my code works for Java 19

Using `Vector` API

To use `Vectors` your computer you must:

1. have newest Java installed
   - run `javac --version` from command line to see compiler version
   - run `java --version` to see JRE version
2. inlclude `Vector` package in program:

```
import jdk.incubator.vector.*;
```

3. compile and run telling Java you're using incubator features:

```
> javac --add-modules jdk.incubator.vector [files to compile]
> java --add-modules jdk.incubator.vector [program to run]
```

Using `Vector` API on HPC

Must load a module with correct version of Java:

```
>  module load amh-java/19.0.1
> javac --add-modules jdk.incubator.vector [files to compile]
> java --add-modules jdk.incubator.vector [program to run]
```

Better still:

- use `sbatch` as in homework assignments with all of these commands in the test script!

Benchmarking Notes

To give “accurate” measure of efficiency:

- test running time of method for **many** invocations
- run several invocations before starting timing
  - “warm up” primes hardware with correct instructions

Min-Plus Example

Input

- `float[] a`, size `n`
- `float[] b`, size `n`

Output

- minimum of `a[i] + b[i]` from `i = 0` to `n - 1`

Min-Plus Vanilla Implementation

```
float min = Float.MAX_VALUE;
for (int i = 0; i < a.length; i++) {
    float x = a[i]; float y = b[i];
    float z = x + y;
    if (z < min) {
        min = z;
    }
}
return min;
```

Min-Plus Vector Implementation

```
        int step = SPECIES.length();
        int bound = SPECIES.loopBound(a.length);
        var mv = FloatVector.broadcast(SPECIES, Float.MAX_VALUE);
        int i = 0;
        for (; i < bound; i += step) {
            var va = FloatVector.fromArray(SPECIES, a, i);
            var vb = FloatVector.fromArray(SPECIES, b, i);
            mv = mv.min(va.add(vb));
        }
        float min = mv.reduceLanes(VectorOperators.MIN);
    ...
```

Min-Plus Vector Implementation (2)

Cleanup:

```
        float min = mv.reduceLanes(VectorOperators.MIN);
        for (; i < a.length; i++) {
            float x = a[i];
            float y = b[i];
            float z = x + y;
            if (z < min) {
                min = z;
            }
        }
        return min;
```

Performance

Vanilla vs Vector on HPC

```
The FloatVector has 8 lanes.
Computing min-plus with simple methods...
That took 654 ms.
Computing min-plus with vector methods...
That took 254 ms.
c = 0.0054750443
d = 0.0054750443
The values are equal!
```

PC Performance, Demo

Lab 02b (Optional)

Add vector instructions to your shortcut program!

Next Time

1. More Vectors!
2. Lab 03