Getting the Best Performance Gains from Parallel Streams

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

Agenda

How get the best performances

What can affect them?

Auto-boxing

Pointer chasing

Avoiding Boxing and Auto-Boxing

Boxing makes an Integer from an int

And unboxing does the opposite

Java can box and unbox automatically

```java
int i = 3;
Integer j = 1;      // auto-boxing

int k = i + j;      // auto-unboxing
Integer l = i + j; // auto-unboxing then auto-boxing

Integer m = j + l; // auto-unboxing then auto-boxing
```

Auto-boxing and unboxing is a useful feature

That has a cost!

How much?

```java
int[]     ints     = {1, 2, 3}; // array of int
Integer[] integers = {1, 2, 3}; // array of Integer

List<Integer> list = List.of(1, 2, 3);
```

Auto-boxing and unboxing is a useful feature

That has a cost!

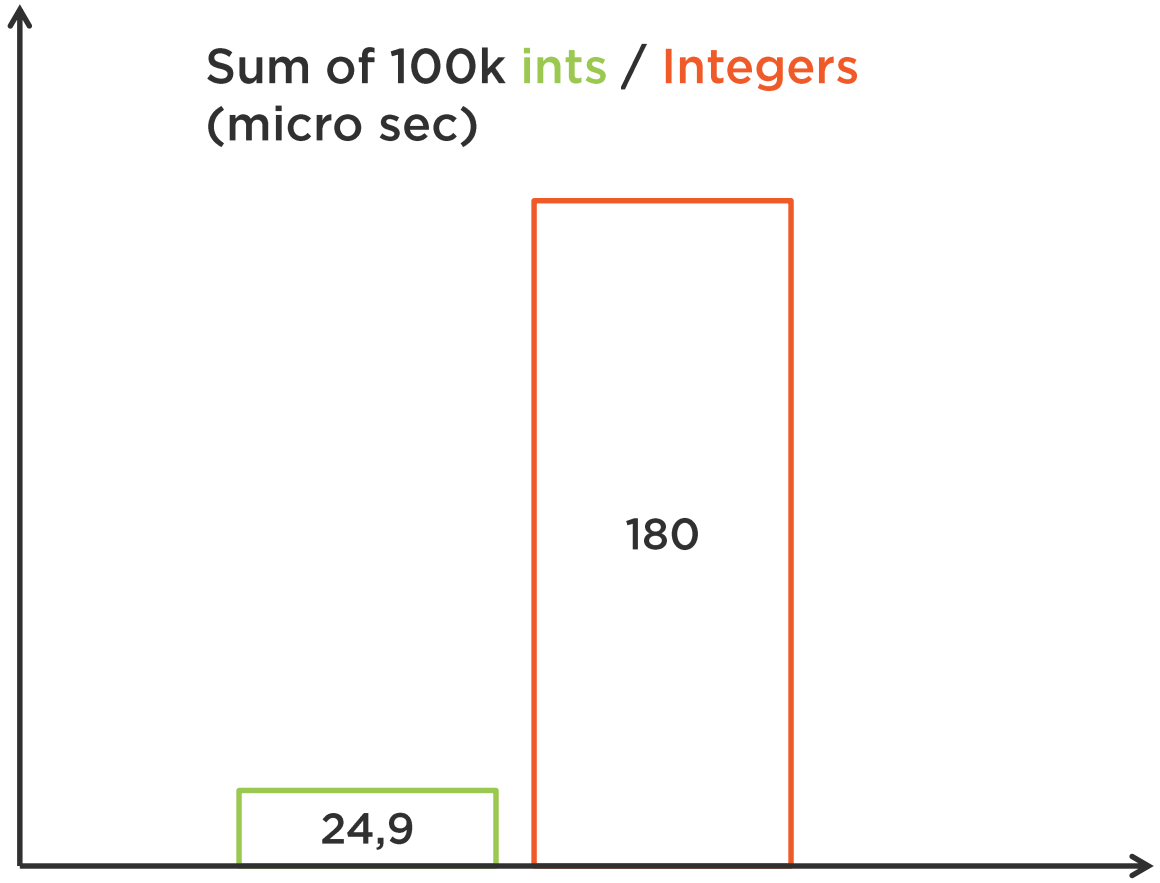
How much?

Demo

Let us write some code!

See the difference between computations on int and Integer

And result is...



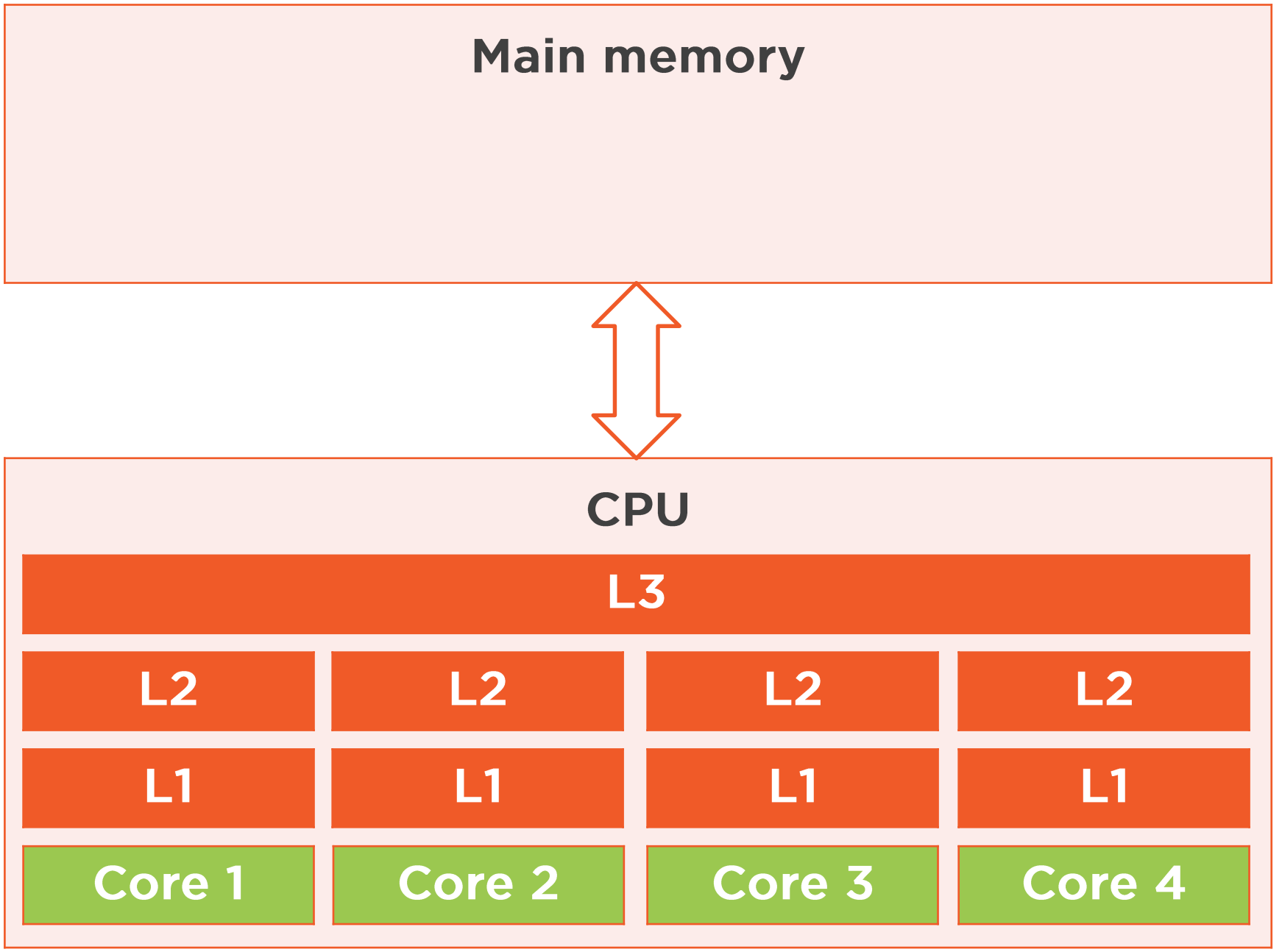
And result is...

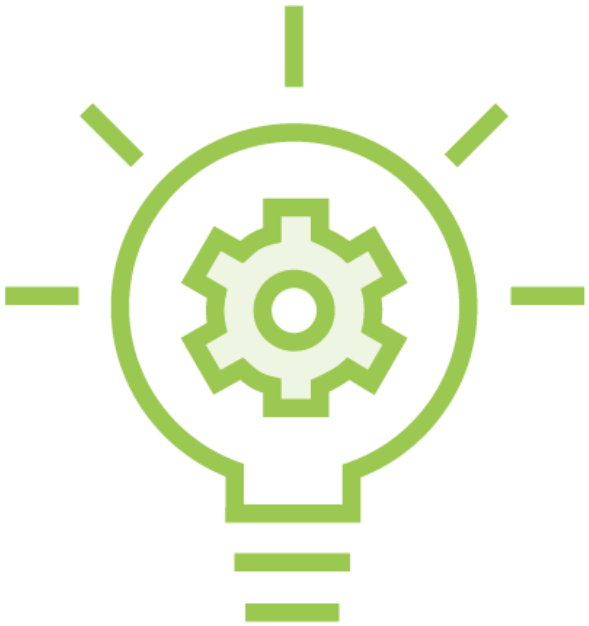
```
Benchmark                                              (N)  Mode  Cnt    Score    Error  Units
M03_Boxing_PointerChasing.calculate_sum_of_ints     100000  avgt   15   24,877 ±  0,132  us/op
M03_Boxing_PointerChasing.calculate_sum_of_integers 100000  avgt   15  179,788 ±  1,567  us/op

M03_Boxing_PointerChasing.calculate_sum_of_range    100000  avgt   15  279,027 ±  2,064  us/op
M03_Boxing_PointerChasing.calculate_sum_of_range_boxed 100000  avgt   15  935,172 ± 32,170  us/op
```

Avoiding Pointer Chasing

What is pointer chasing?

Let us see how CPU are working


Main memory
CPU
L3
L2
L2
L2
L2
L1
L1
L1
L1
Core 1
Core 2
Core 3
Core 4

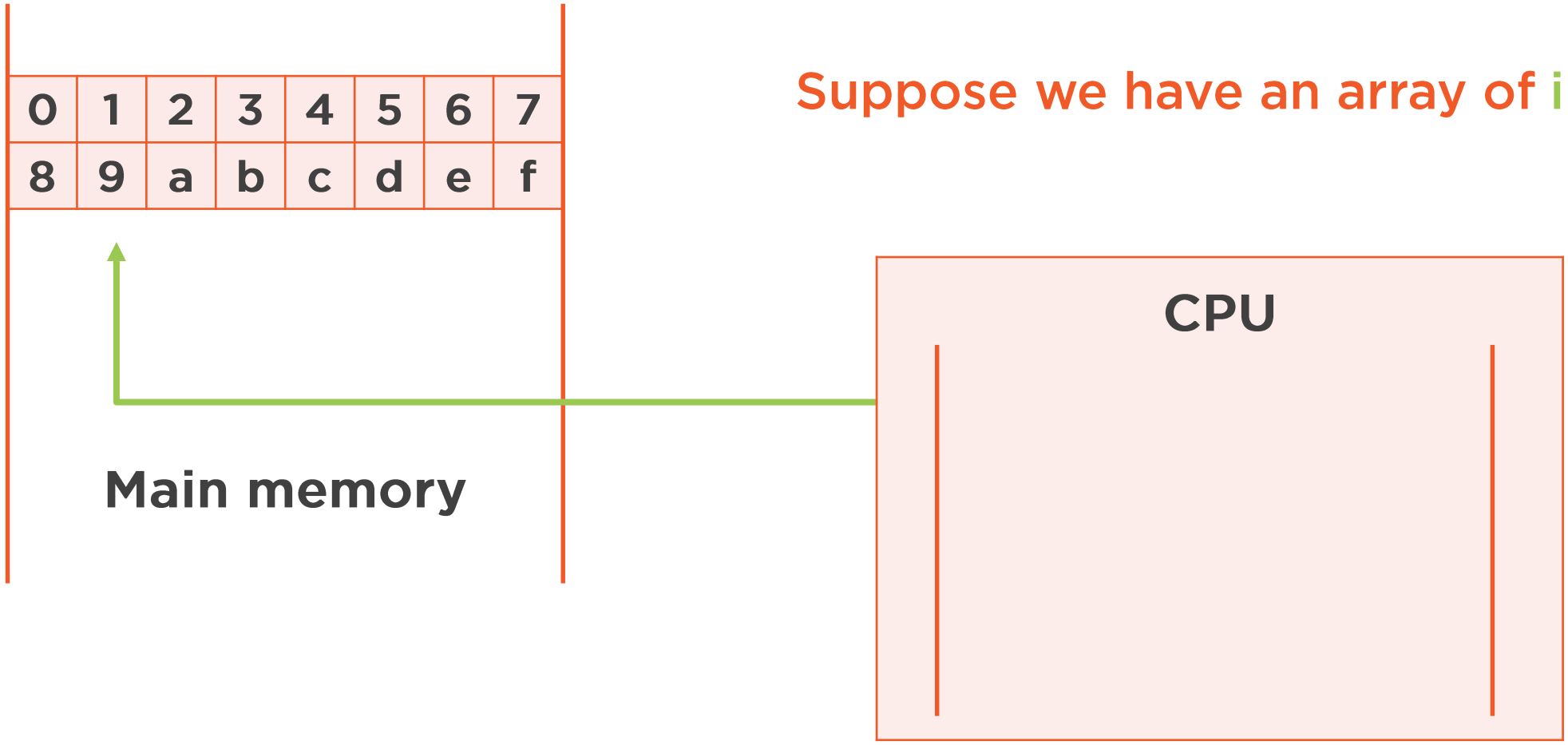


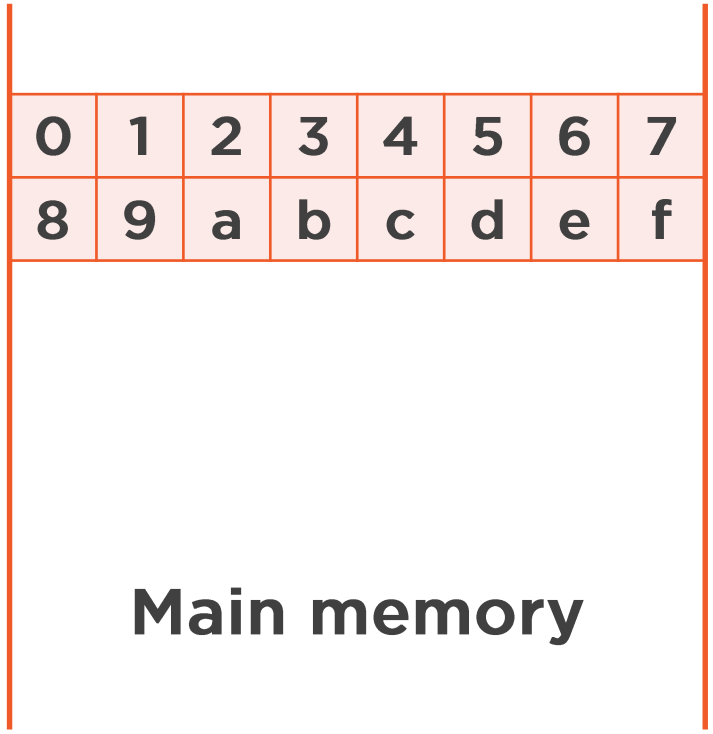
The memory is organized in lines

Data is transferred line by line between the main memory and the CPU caches

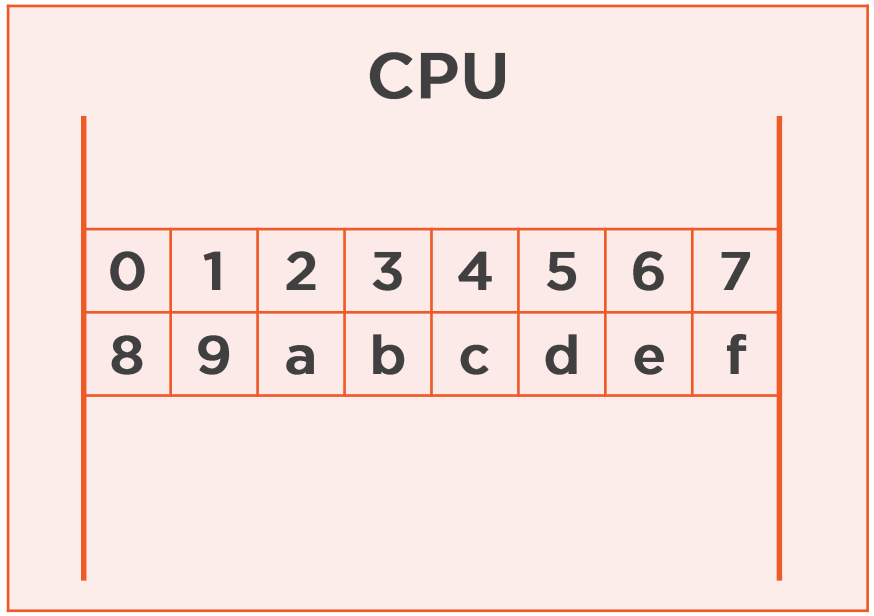
An array is stored in a contiguous zone of the memory

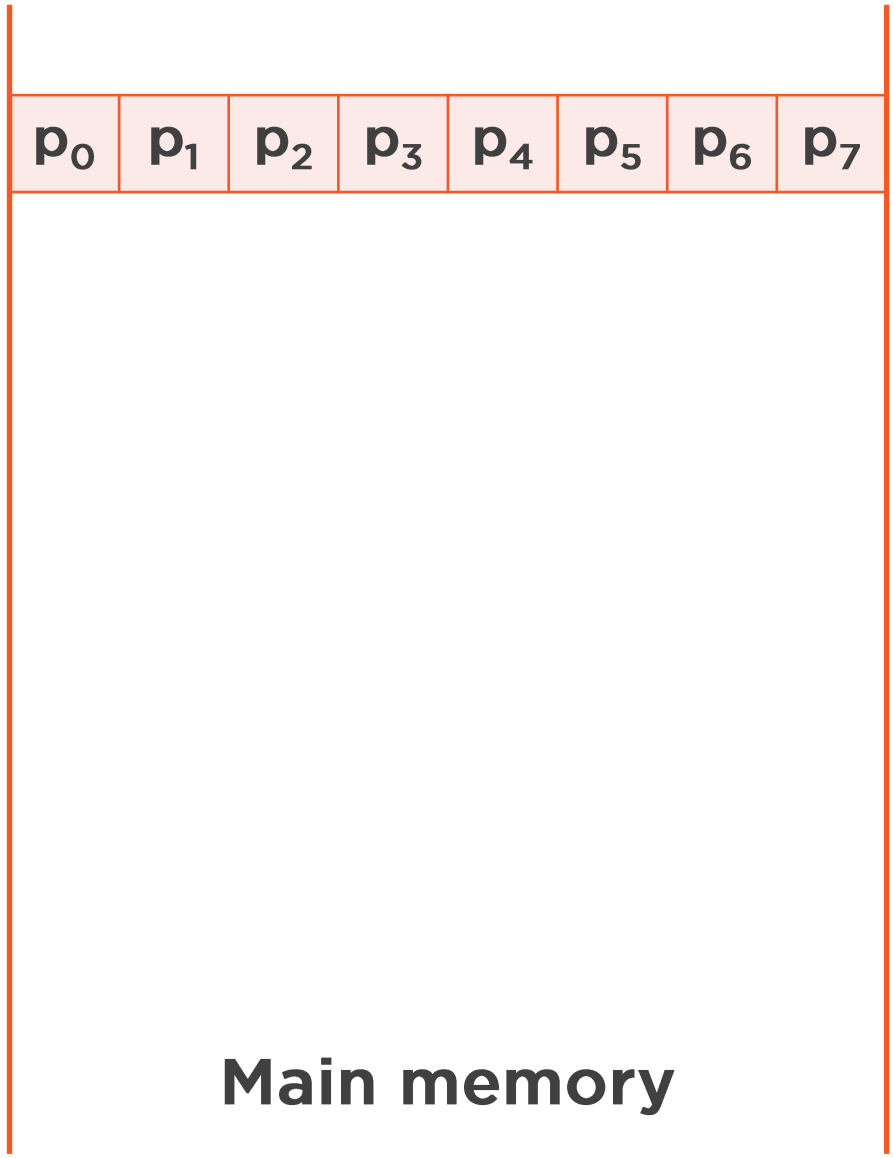
Suppose we have an array of int



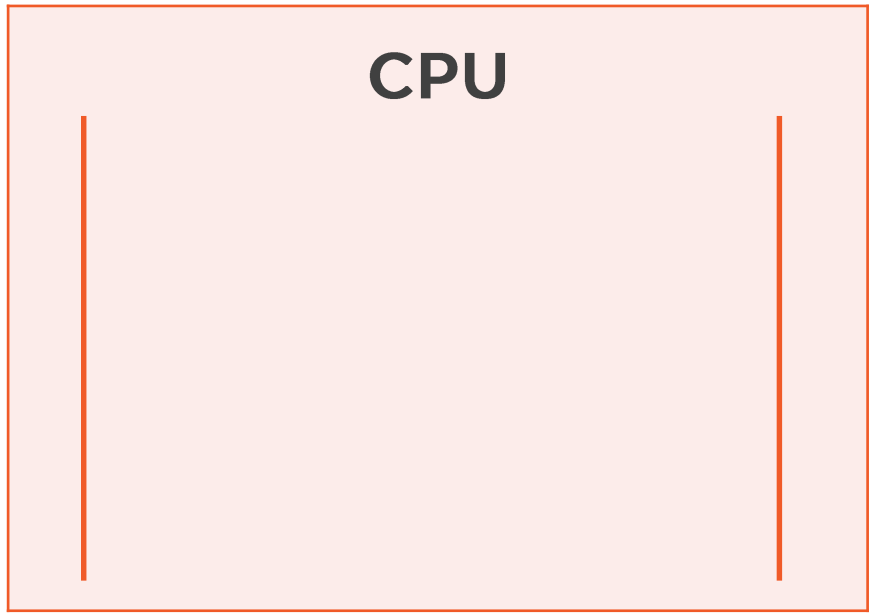


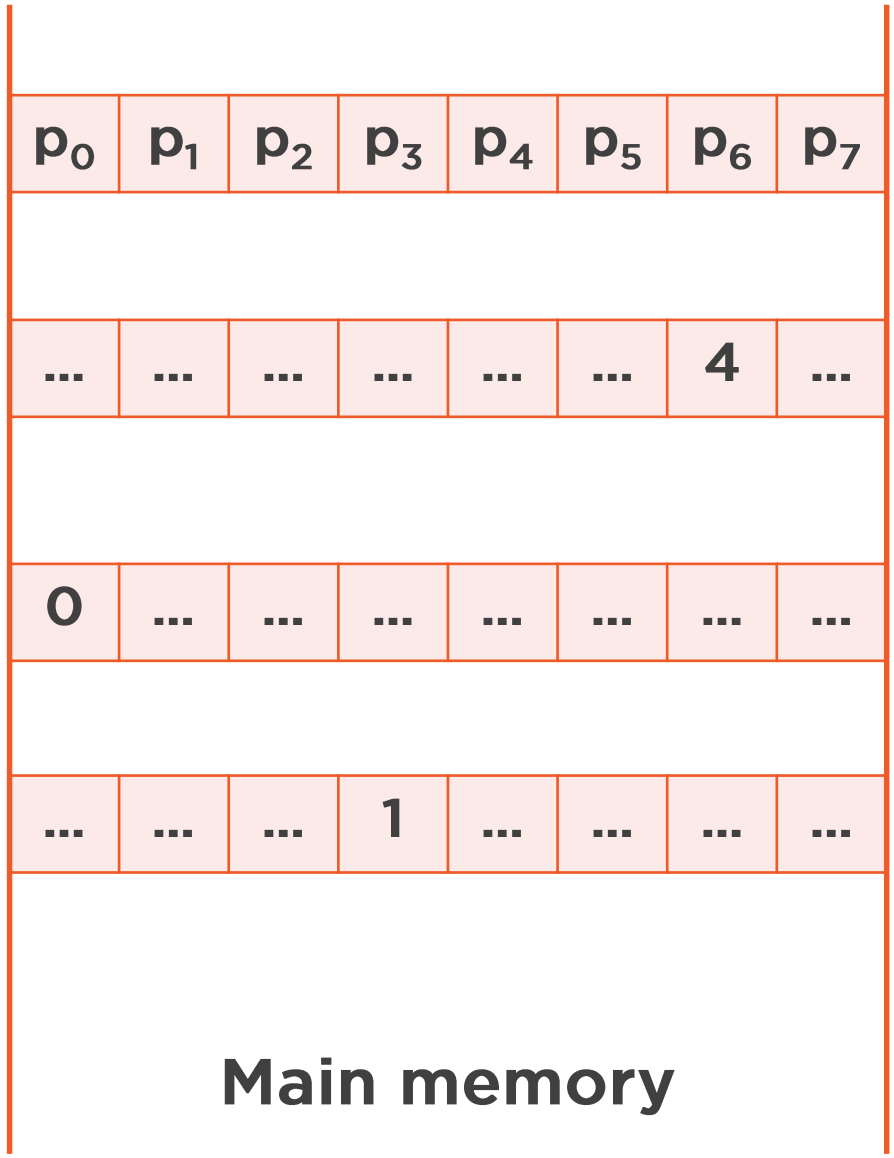
Suppose we have an array of int
Data transfer is very efficient!



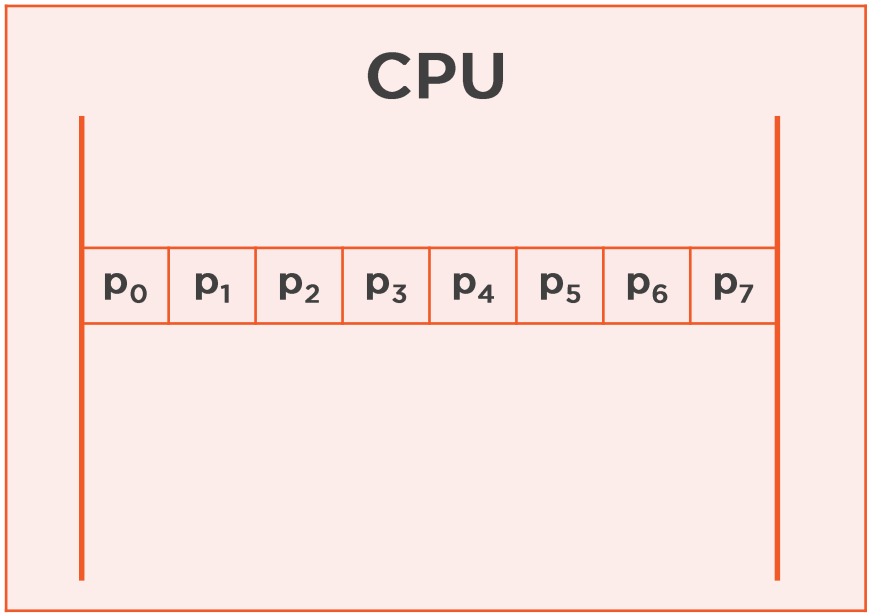


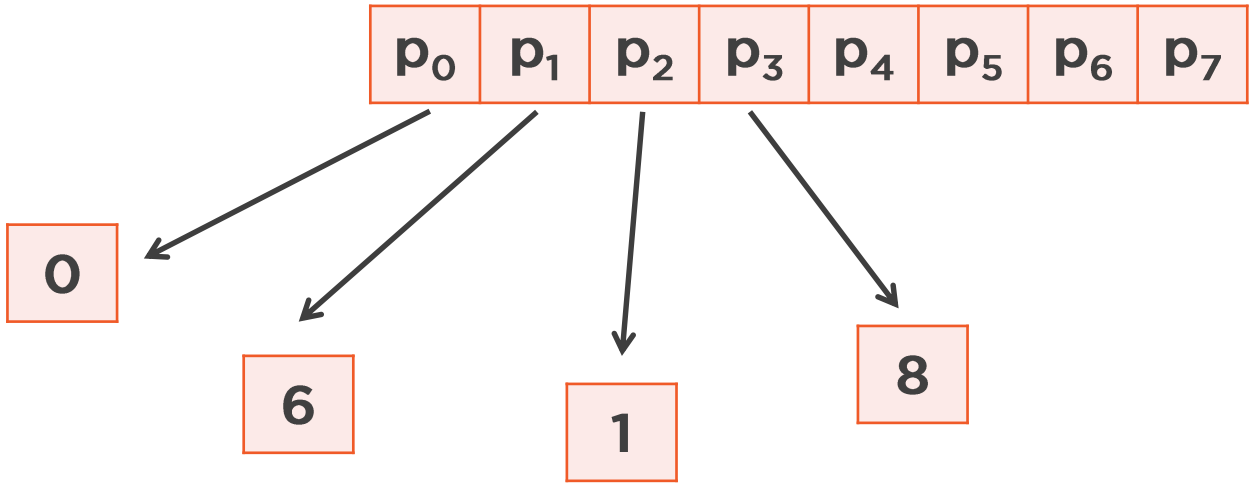
Suppose we have an array of Integer



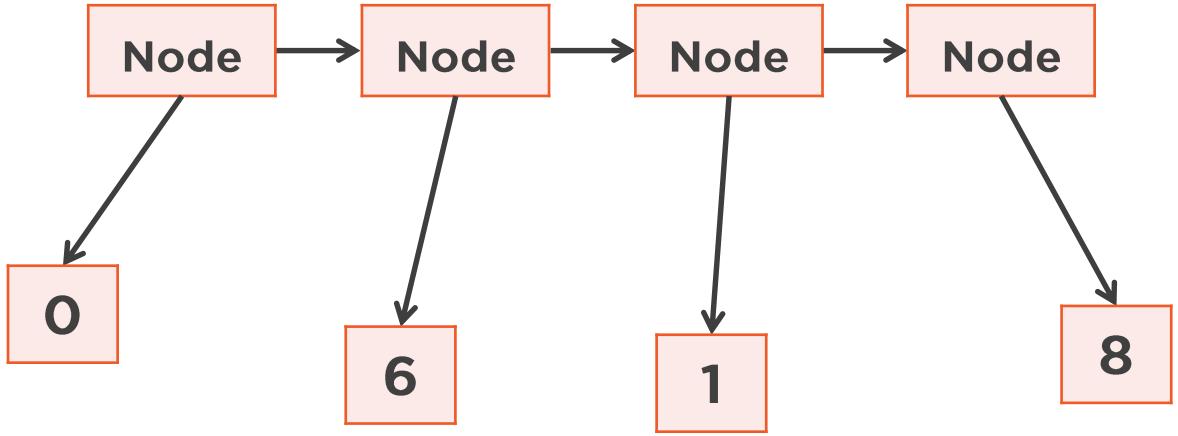


Suppose we have an array of Integer
Memory layout is not “cache friendly”





ArrayList<Integer>



LinkedList<Integer>

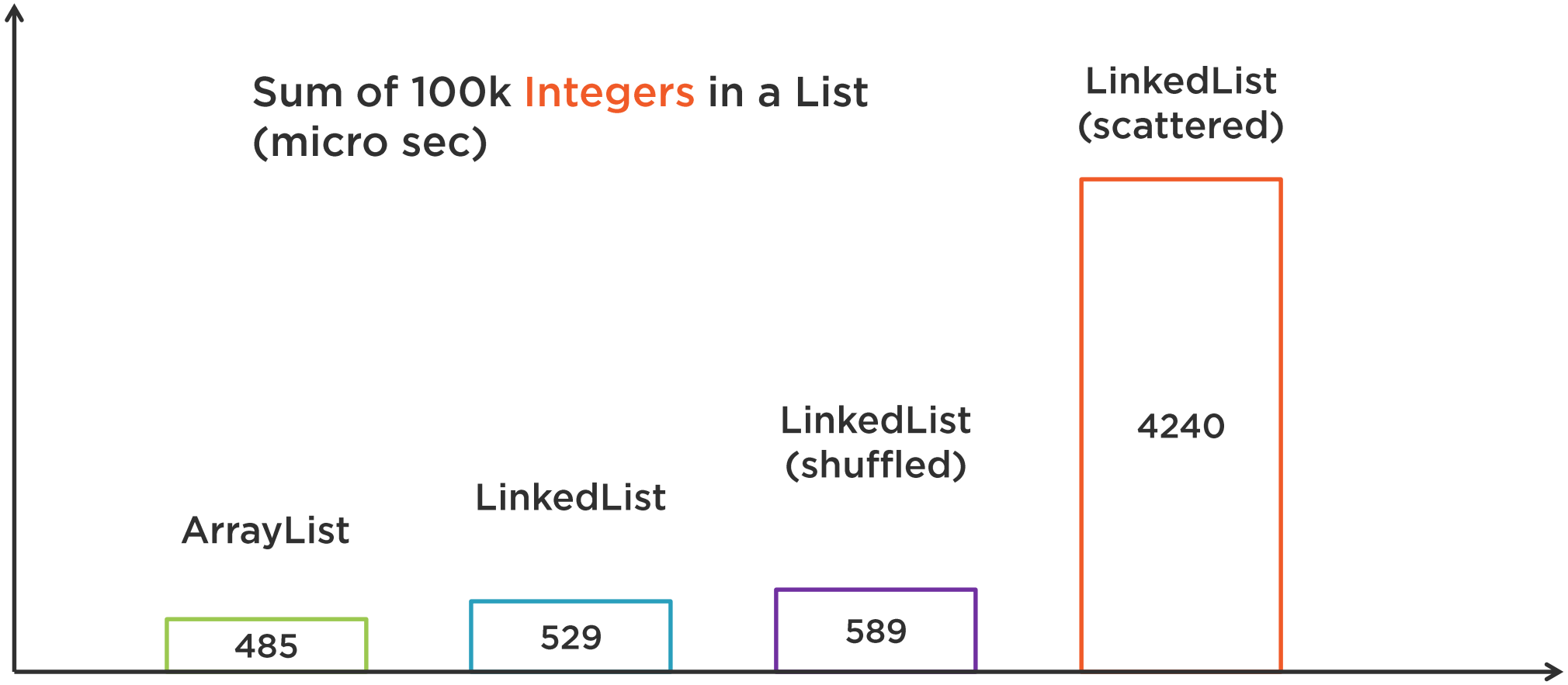
What is the cost of “cache unfriendly” data structures?

Demo

Let us write some code!

And see the cost of pointer chasing

And result is...



And result is...

```
Benchmark                                                    (N)  Mode  Cnt      Score      Error  Units
M03_Boxing_PointerChasing.calculate_sum_of_array_list     100000  avgt    9    485,598 ±    1,857  us/op
M03_Boxing_PointerChasing. calculate_sum_of_linked_list   100000  avgt    9    529,755 ±    7,862  us/op

M03_Boxing_PointerChasing. calculate_sum_of_linked_list_shuffled   100000  avgt    9    589,530 ±    6,249  us/op

M03_Boxing_PointerChasing. calculate_sum_of_linked_list_scattered  100000  avgt    9  4236,442 ±   89,772  us/op
```

Prefer ArrayList over LinkedList

Module Wrap Up

What did you learn?

What is costly when dealing with data processing?

1) Boxing and unboxing

2) Pointer chasing