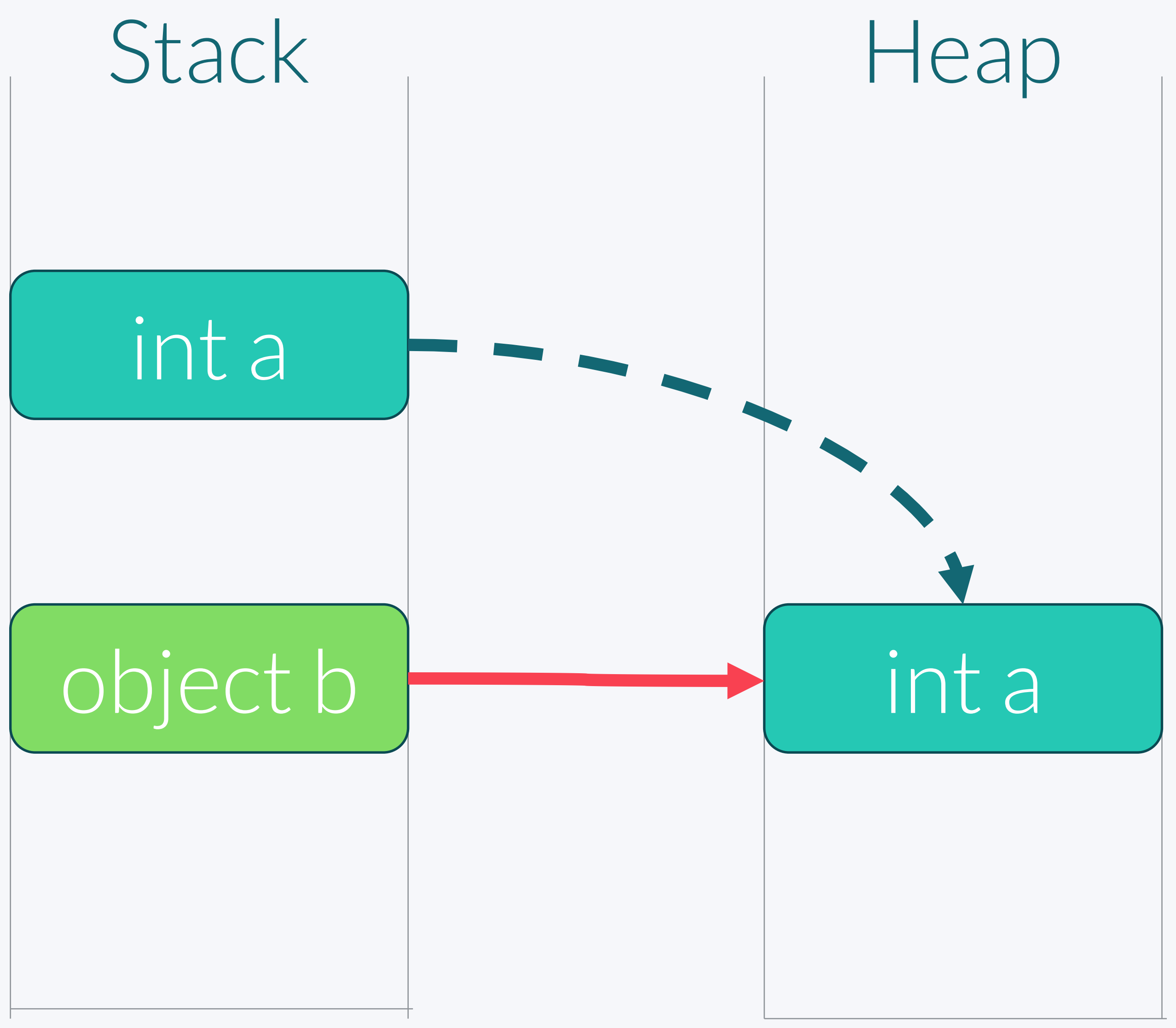
FOUR SIMPLE TIPS TO IMPROVE C#

PERFORMANCE

Learn actionable performance tricks to optimize your C# code and make it run much faster

TIP 1

THE OVERHEAD OF BOXING



```
int a = 1234;
object b = null;
b = a;  // <--- boxing
```

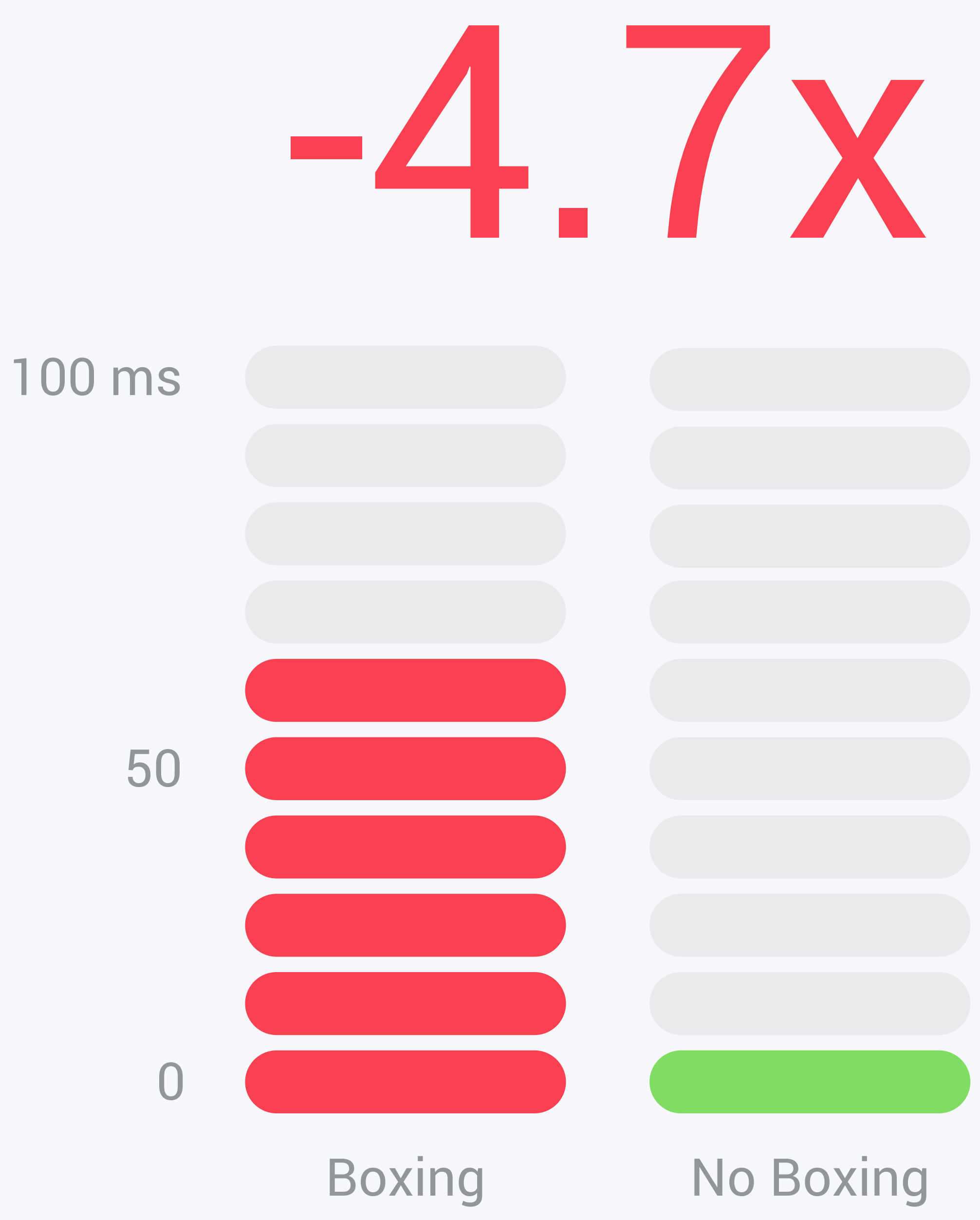
In the code on the left, I am assigning an **int** variable a to an **object** variable b.

This is not trivial, because the integer is stored on the stack and the object variable can only refer to objects on the heap.

The dotNET runtime executes the assignment by copying the integer to a new object on the heap. The object reference b can now refer to this new object.

This process is called boxing, and it slows down your code. Boxing is **4.7 times** slower than the same code without boxing.

Try to avoid boxing as much as possible in your own code. Don't use the **object** type to store value type data.

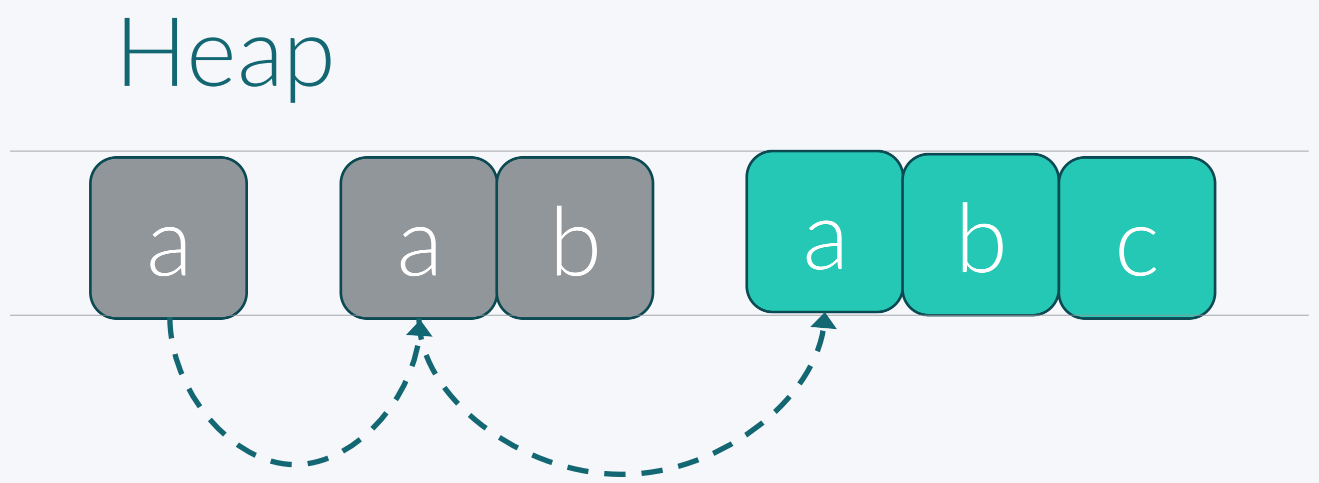


BENCHMARK

mdfarragher.com

TIP 2

TWO WAYS TO ADD STRINGS TOGETHER

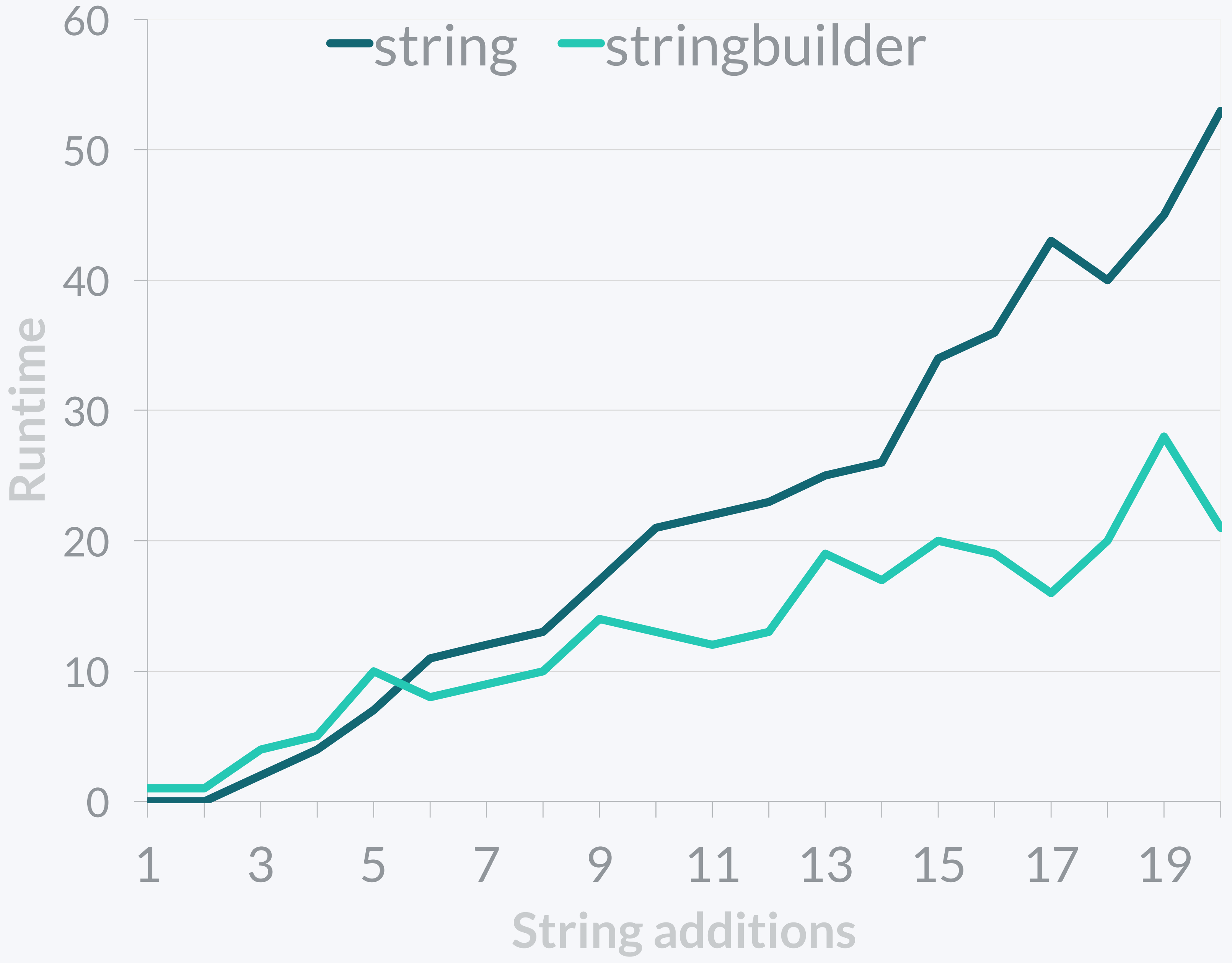


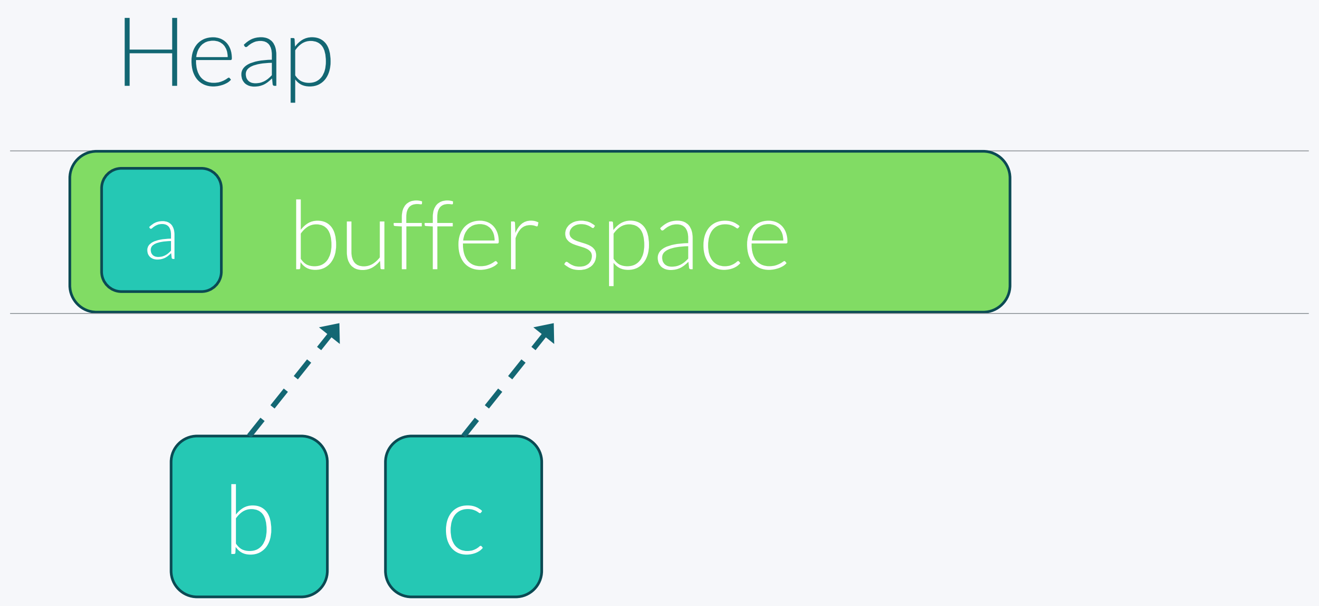
```
string s = "a";
s += "b";
s += "c";
```

When adding strings together, each operation creates a new string on the heap, leaving a dereferenced unmodified string behind.

This makes strings immutable.

When adding **more than 4 strings** together, the StringBuilder consistently outperforms the string.





```
var s = new StringBuilder("a”);
s.Append("b");
s.Append("c");
```

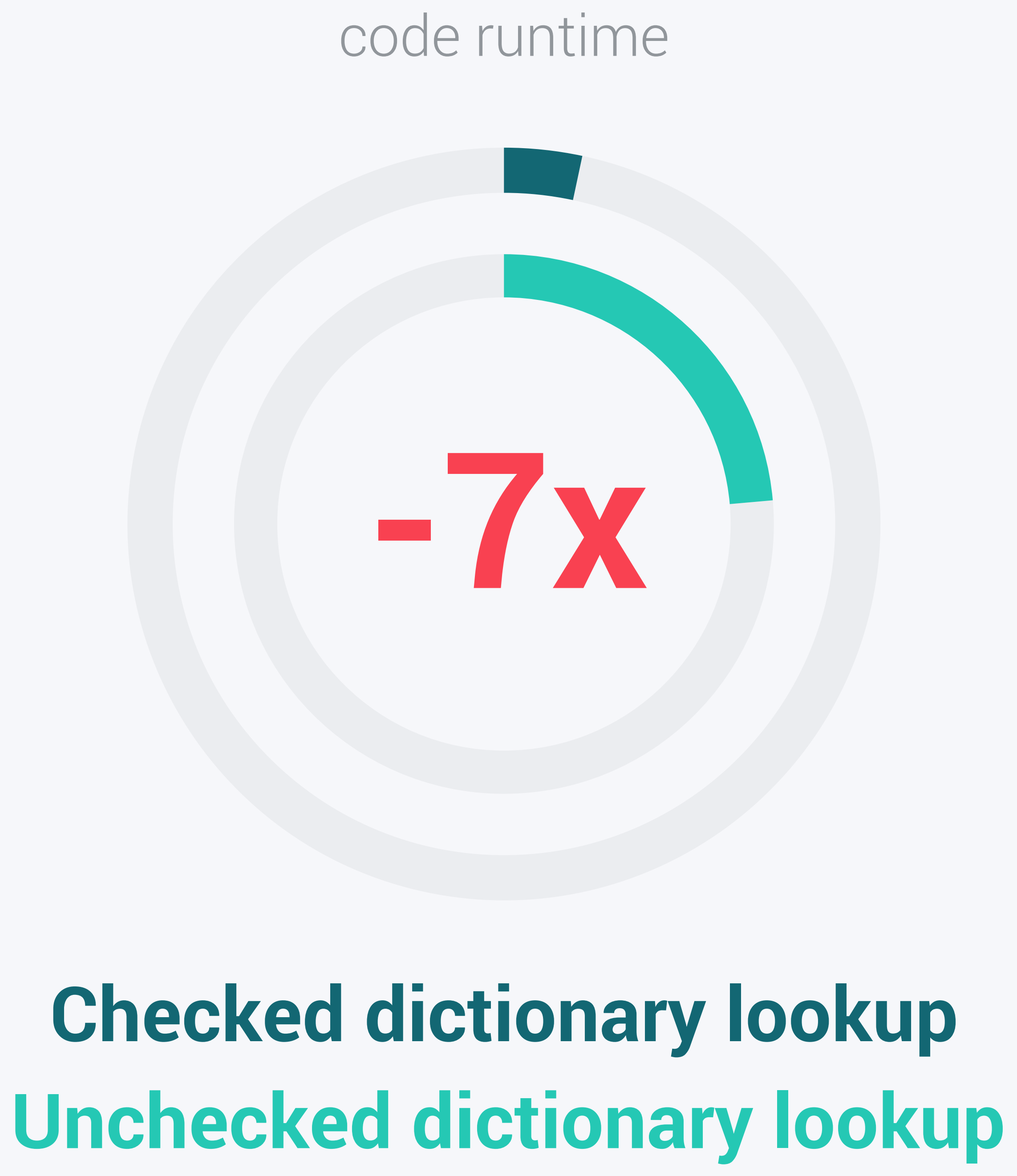
When adding strings to a StringBuilder, each operation writes into available buffer space in memory.

This makes StringBuilders mutable.

mdfarragher.com

TIP 3

THE EXCEPTION PENALTY



Looking up values in a dictionary and catching KeyNotFoundException is 7x slower than using ContainsKey

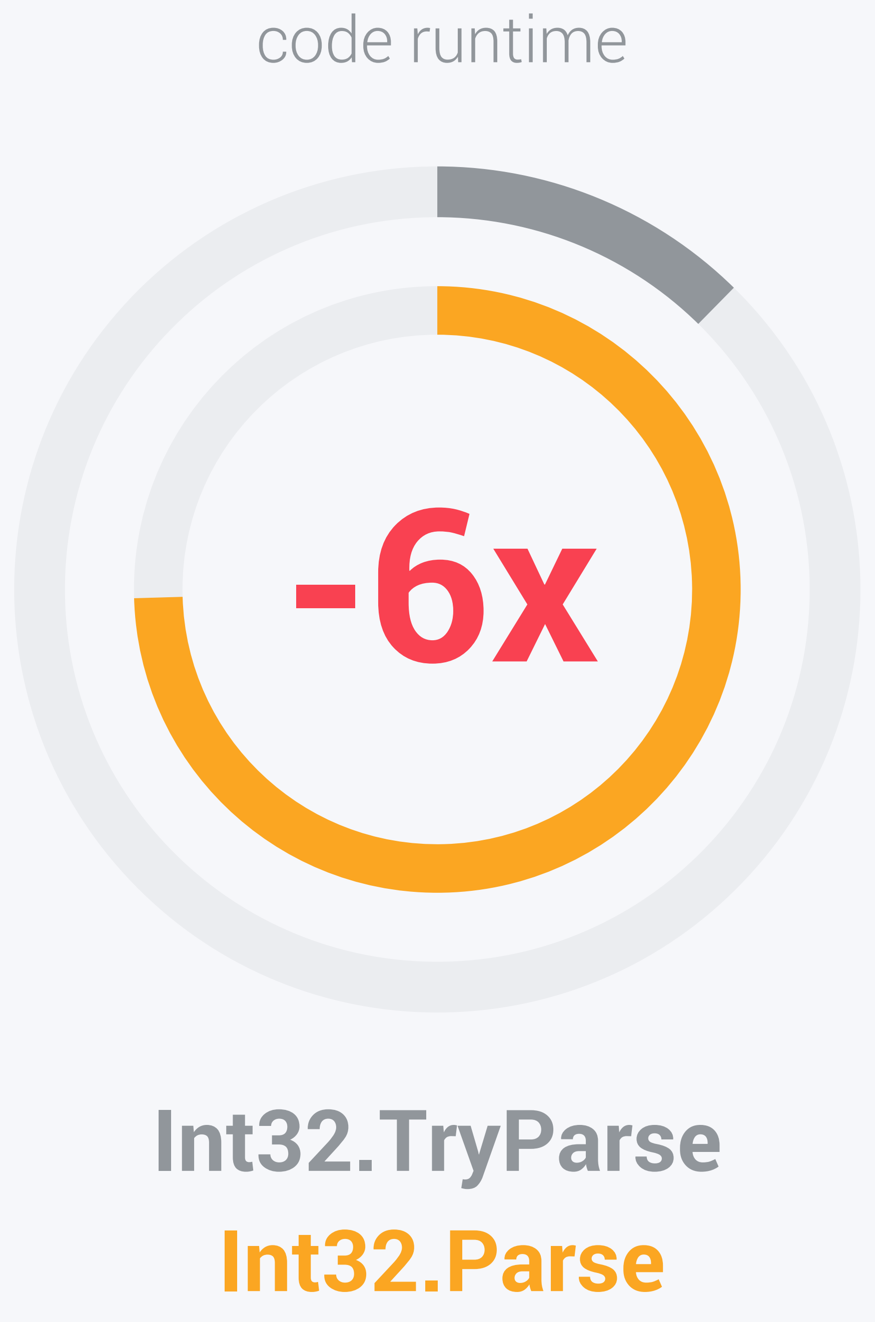
When you throw an exception, the dotNET runtime captures the current state of the running process and then unwinds the stack to find the nearest catch block.

This operation is very slow, which makes exceptions only suitable for diagnostic purposes.

When looking up values in a dictionary, catching KeyNotFoundException is **7x slower** than calling ContainsKey.

When parsing random input data, using Int32.Parse and catching FormatException is **6x slower** than using TryParse instead.

Don’t use exceptions for control flow.

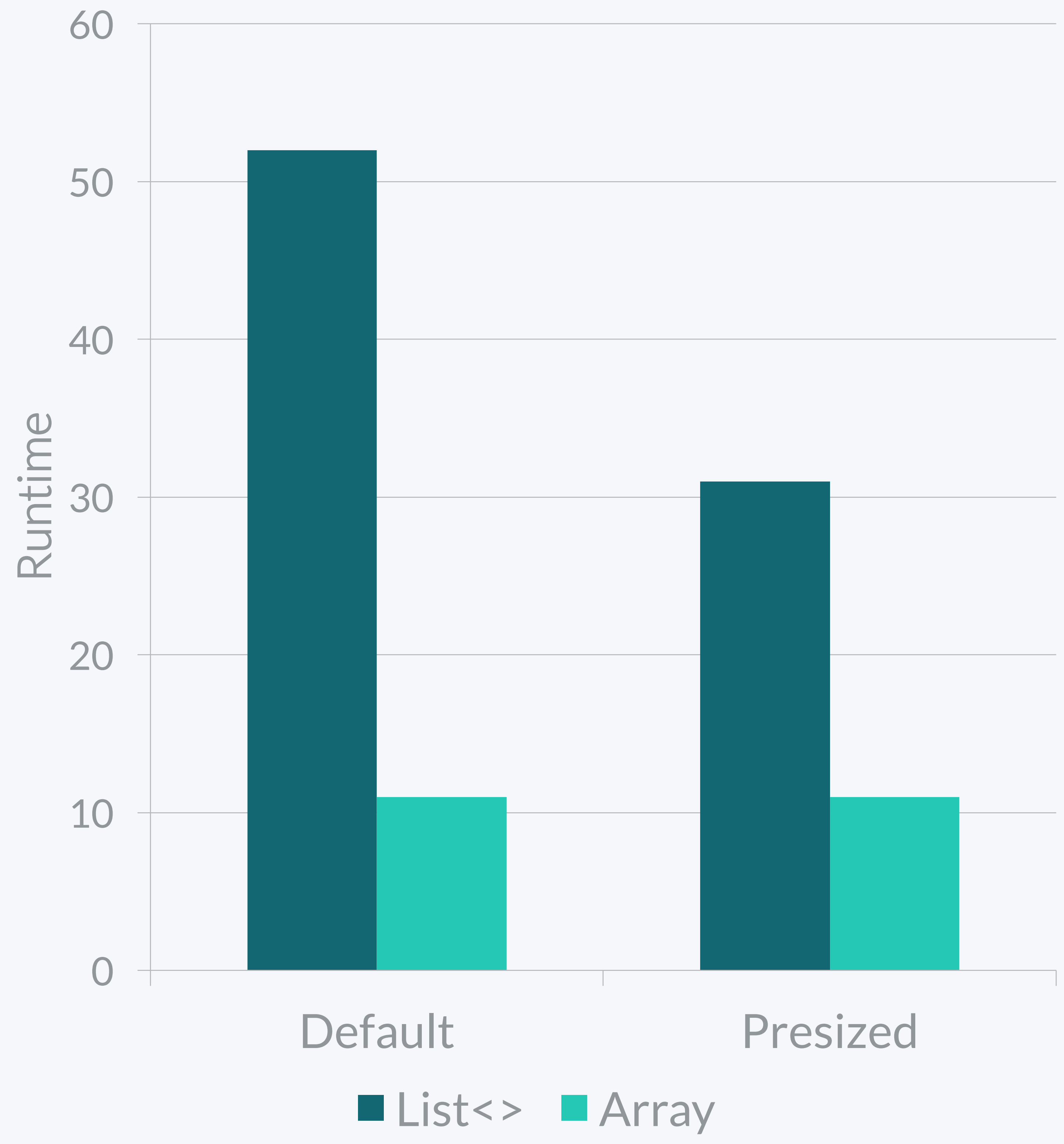


Using Int32.Parse on random input data and catching FormatException is 6x slower than using Int32.TryParse

mdfarragher.com

TIP 4

ARRAYS OUTPERFORM GENERIC LISTS



Generic lists are ideal for storing data when you don't know in advance how many elements there will be. Lists automatically resize to make room for new data.

This makes lists **4.7x slower** slower than arrays.

But even when we pre-size the list to the correct number of elements, it is still **2.8x slower** than an array.

This is because the dotNET runtime has dedicated IL instructions for handling 1-dimensional arrays. Arrays compile to high-performance machine code.

If you want maximum performance, use 1-dimensional arrays if you can.

Compiled IL code:

```
// list[1] = 1
ldloc.0
ldl.i4.1
ldlc.i4.1
callvirt list<int>.get_Item(…)
```

Compiled IL code:

```
// array[1] = 1
ldloc.0
ldc.i4.1
ldc.i4.1
stelem.i4 // <-- dedicated IL
```

mdfarragher.com

Would you like a course in

C# Code Performance?

My performance tune-up covers these four optimizations in detail, and also teaches you how to optimize your code for the Garbage Collector.

only $9

https:// training.mdfarragher.com/p/csharp-performance-tune-up http://mdfarragher.com

MARK FARRAGHER

I will teach you hard and soft skills to take your tech leadership to the next level

/mdfarragher

@mdfarragher

/in/mdfarragher

mark@mdfarragher.com