GO PROFILING & OPTIMISATION

Kenneth Miles

tommymcguiver@gmail.com

ABOUT ME

Technical BA @ Jumbo last 2 years. Creating professional software for 13 +years. Embedded software developer -> Backend Web Dev -> Tech BA

Programming Languages:

- Golang
- Java
- C
- PHP
- Perl

AGENDA

- Start with buggy slow program
- Analyse it with go tooling
- Make it fast & correct & understand how its running

WHY?

- Optimise resource usage
- Save Money

I ASSUME YOU KNOW...

- Fundamental Go language skills
- Go has awesome tools
- Go is hell fast, let me demonstrate

LETS PLAY

- Race detector
- CPU Profiling
- Memory & Garbage Profiling
- Go datastructures under the hood
- Live profiling

PPROF

Profiling is useful for identifying expensive or frequently called sections of code. The Go runtime provides profiling data in the format expected by the pprof visualization tool. Data is sampled at 100/second
https://github.com/google/pprof

RACE DETECTOR

https://golang.org/doc/articles/race_detector.html

Data races are among the most common and hardest to debug types of bugs in concurrent systems. A data race occurs when two goroutines access the same variable concurrently and at least one of the accesses is a write. Won't detect races without test coverage. It's important to have good coverage.

RACE DETECTOR

- go test -v -race
- go run -race mysrc.go // to run the source file
- go build -race mycmd // to build the command
- go install -race mypkg // to install the package

CPU PROFILING

- go test -bench Benchmark_ServeHttp -benchmem -cpuprofile prof.cpu > bench.0
- go tool pprof randomnumber.test prof.cpu
- (pprof) top10 -cum
- (pprof) list HandleRandom
- (pprof) weblist HandleRandom

MEMORY & GARBAGE PROFILING

- go test -bench Benchmark_ServeHttp -benchmem -memprofile prof.mem > bench.0
- go tool pprof randomnumber.test prof.mem
- (pprof) top10 -cum
- (pprof) list HandleRandom
- (pprof) disasm HandleRandom
- (pprof) weblist HandleRandom

MUTEX BLOCKING

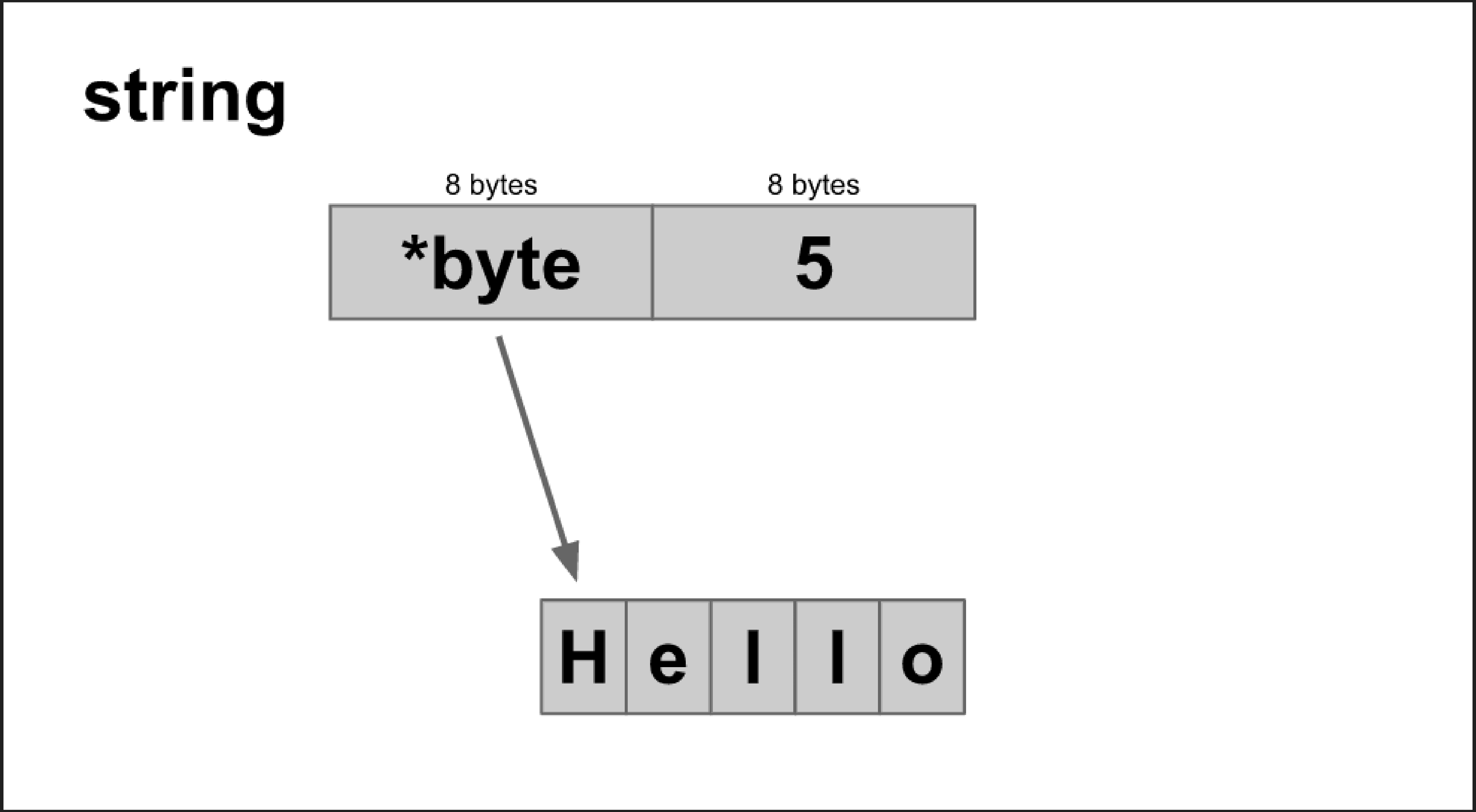
Block profile shows where goroutines block waiting on synchronization primitives

- go test -bench Health -blockprofile=prof.block
- go tool pprof randomnumber.test prof.mem
- (pprof) top10 -cum
- (pprof) list HandleRandom
- (pprof) disasm HandleRandom
- (pprof) weblist HandleRandom

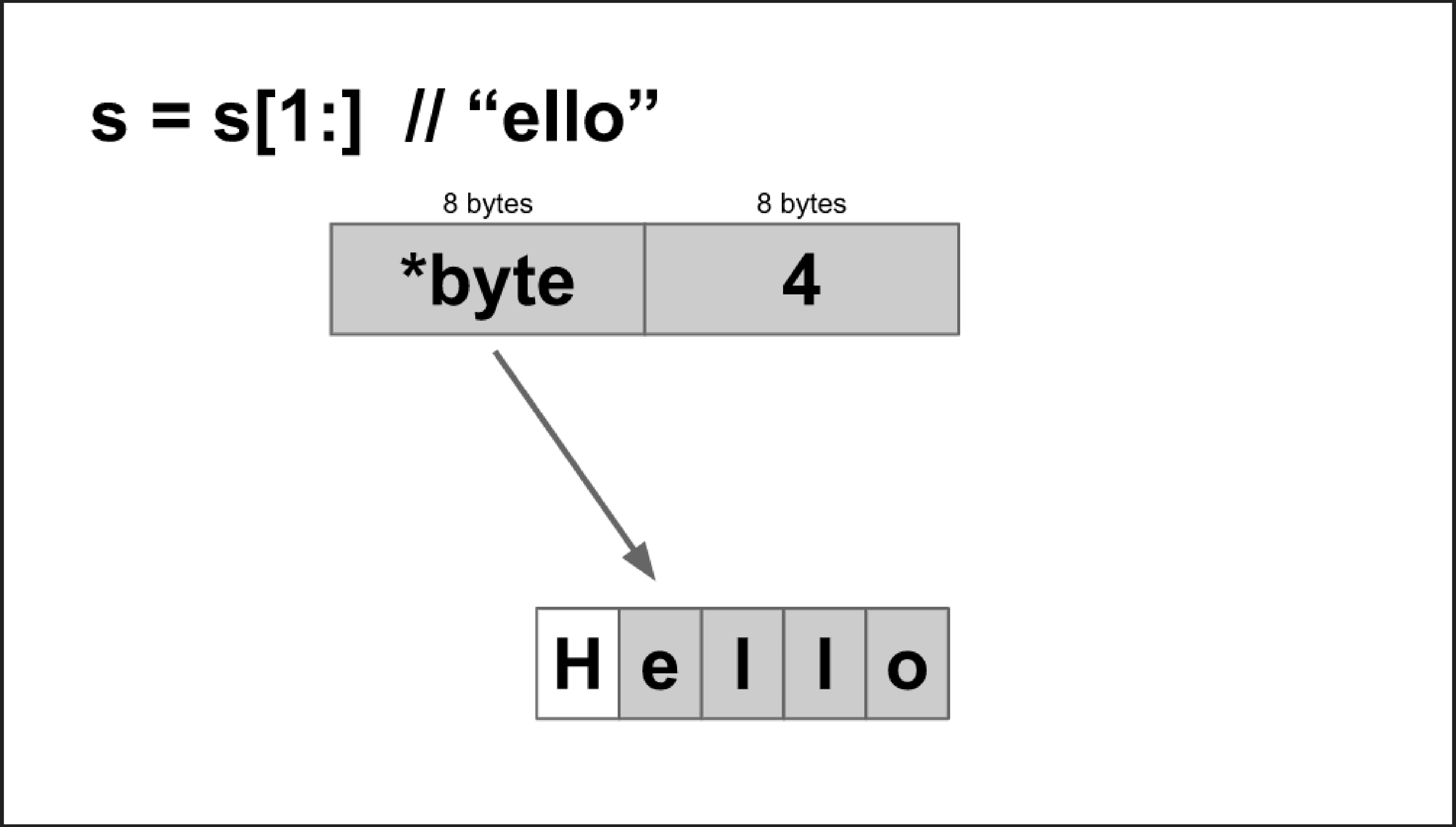
GO BUILT-IN DATA STRUCTURES

- string
- slice
- interface
- map
- chan
- map

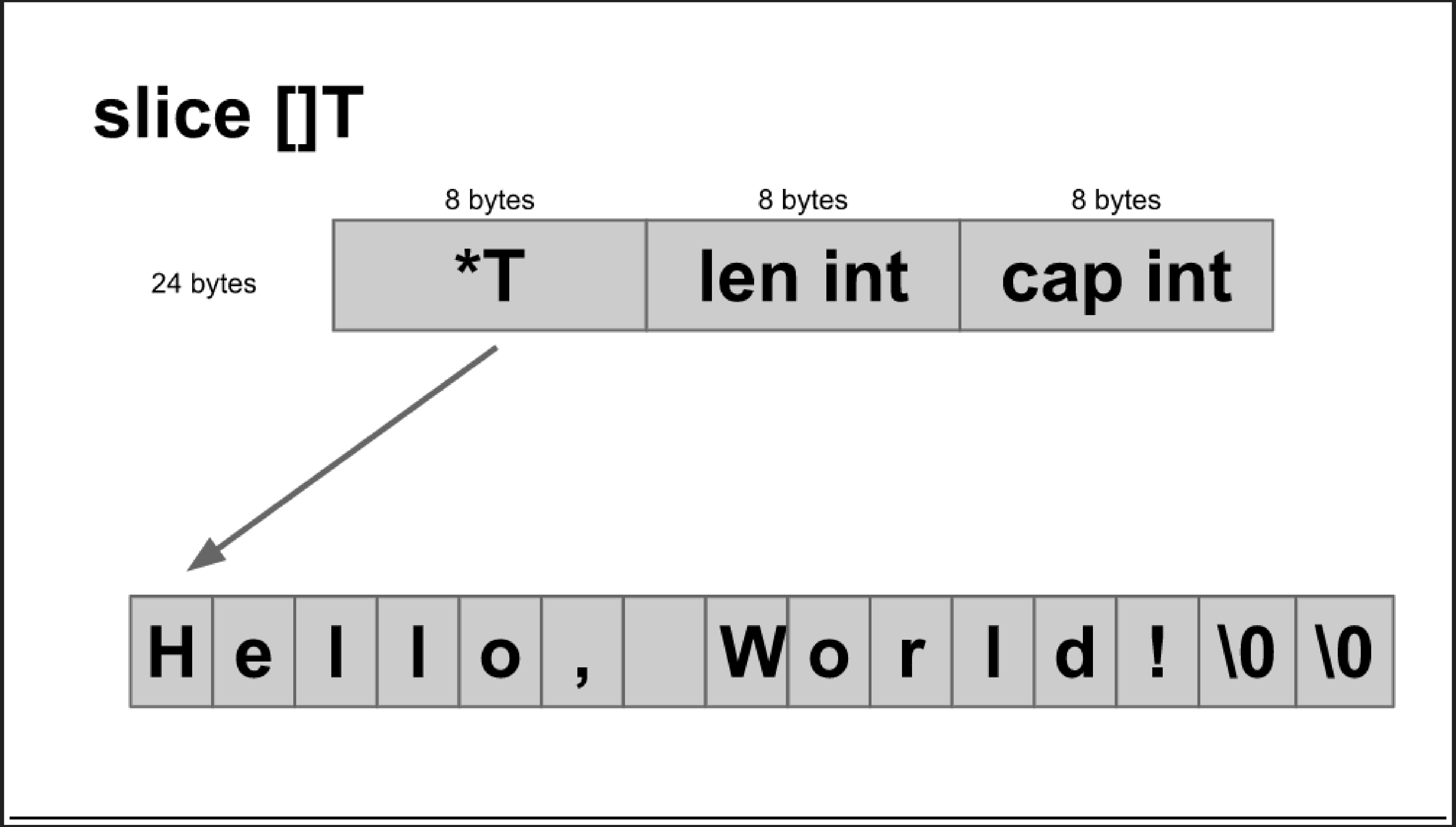
GO BUILT-IN DATA STRUCTURES



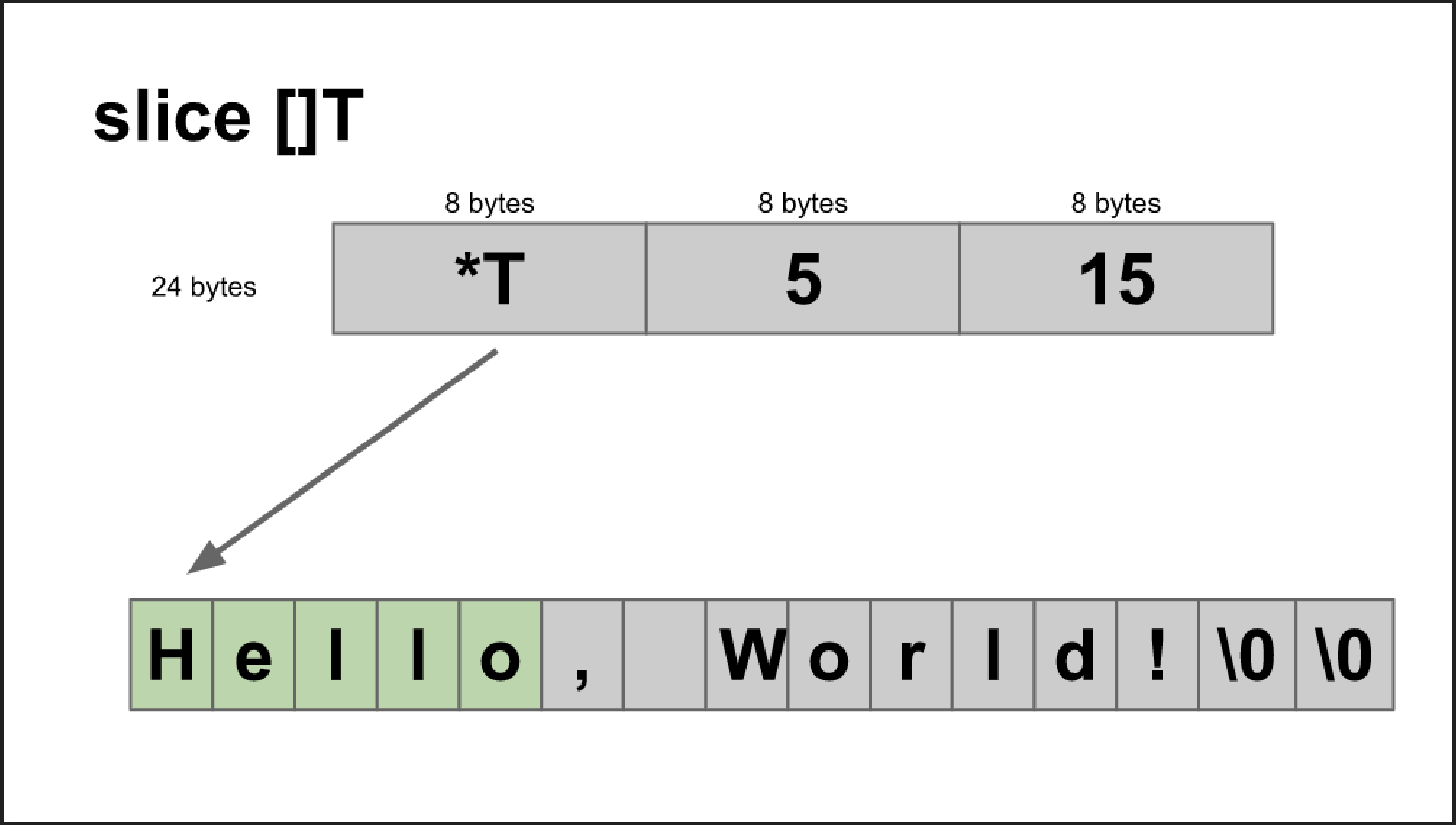
GO BUILT-IN DATA STRUCTURES



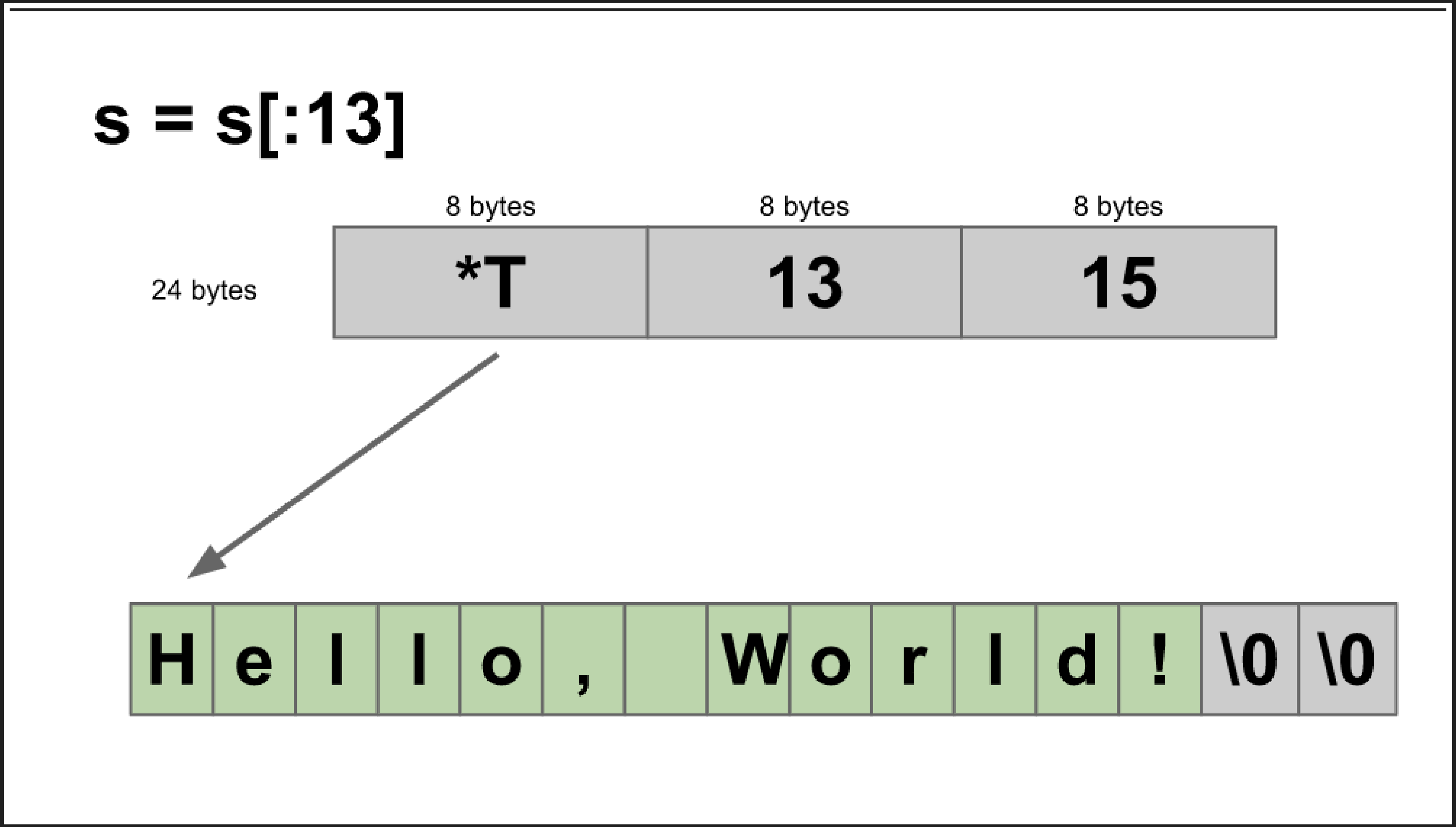
GO BUILT-IN DATA STRUCTURES



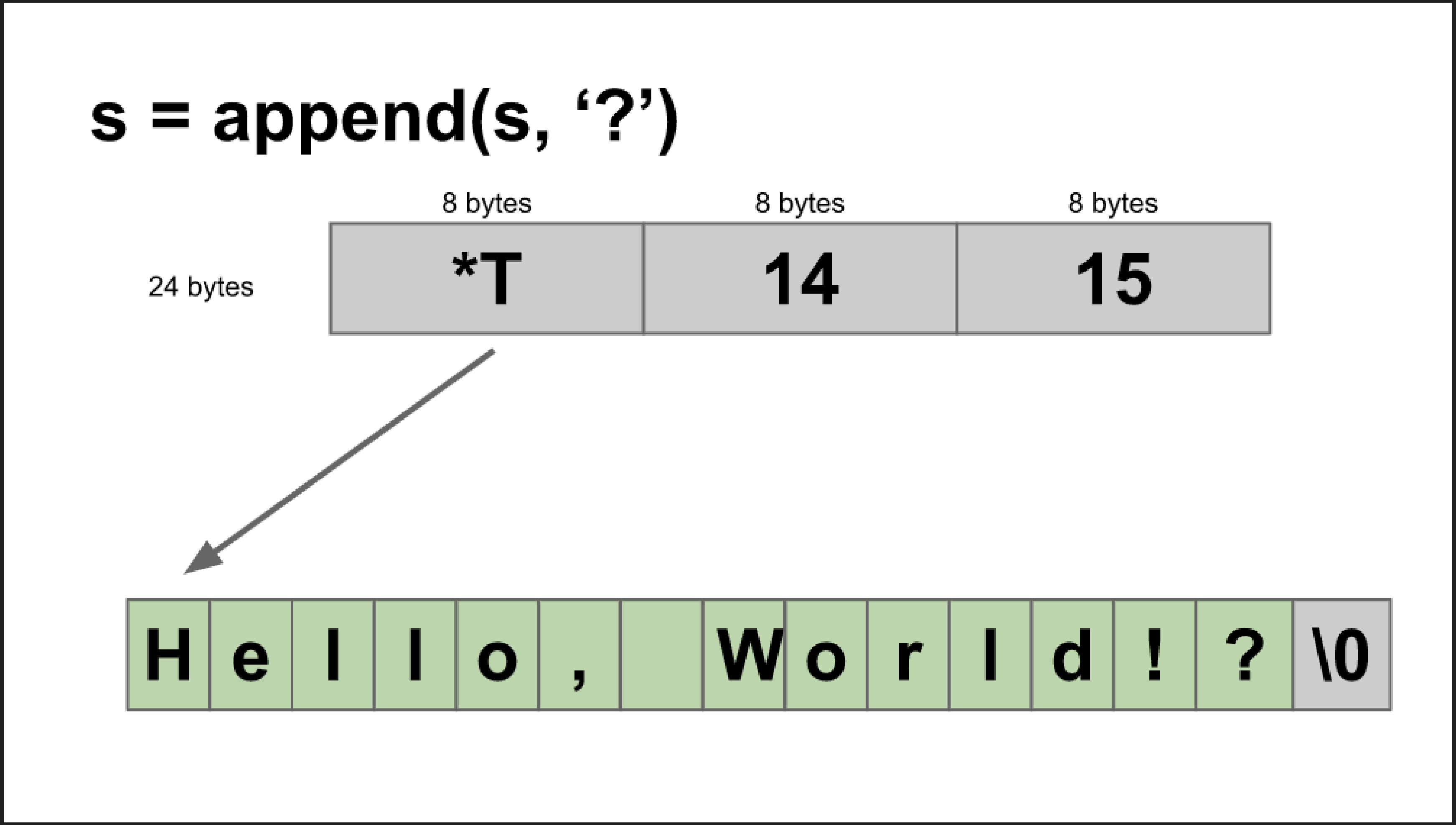
GO BUILT-IN DATA STRUCTURES



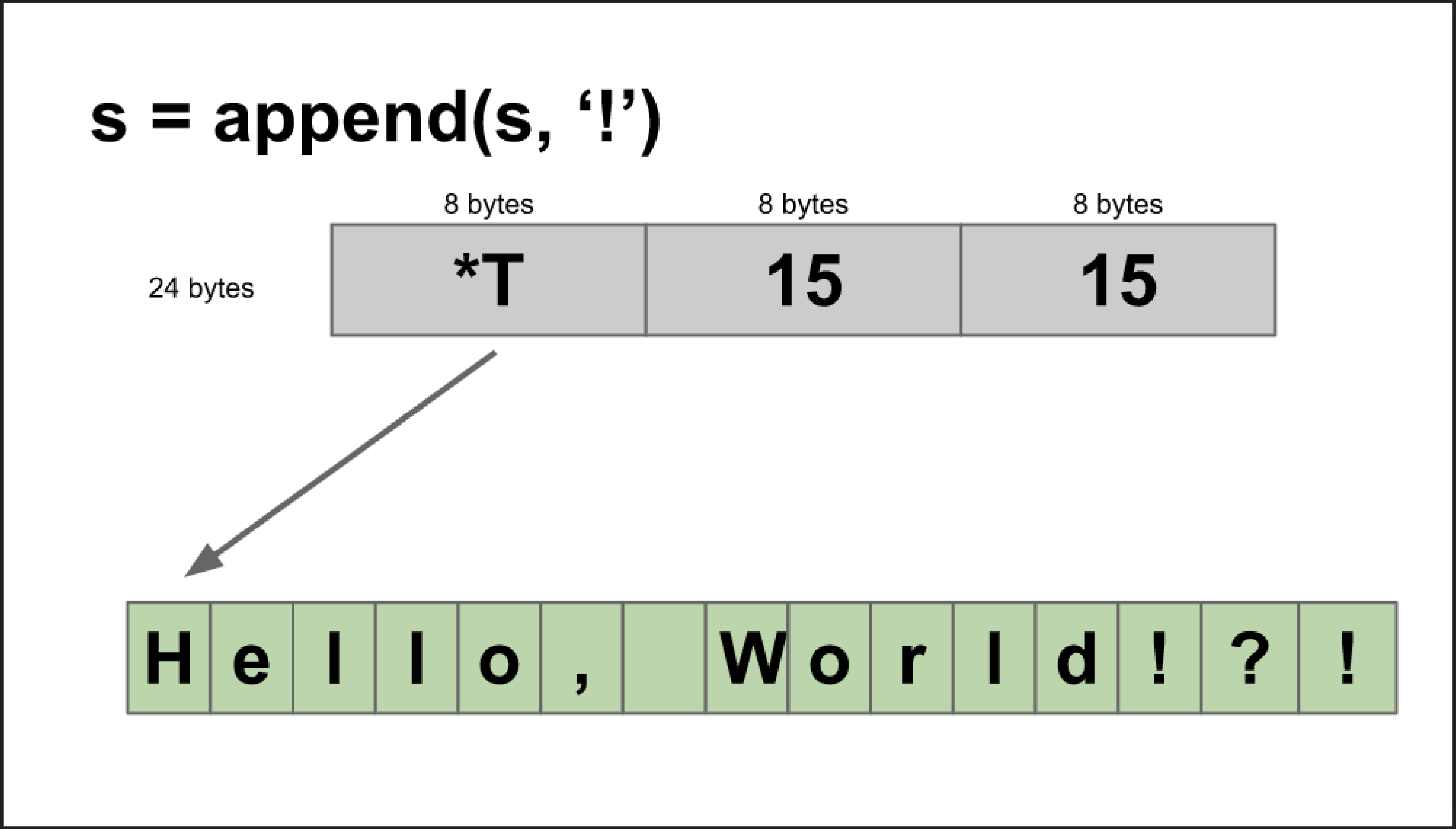
GO BUILT-IN DATA STRUCTURES



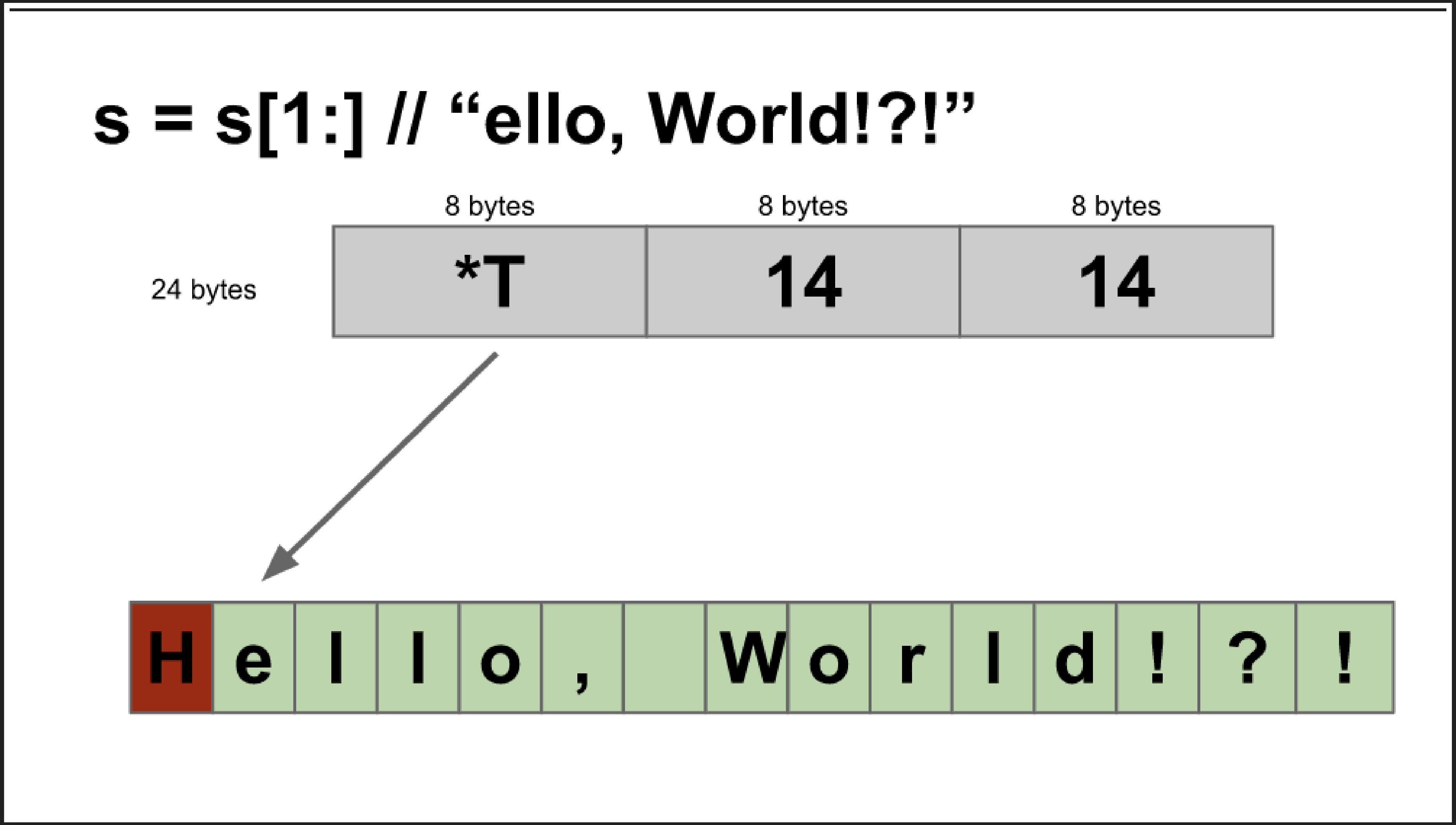
GO BUILT-IN DATA STRUCTURES



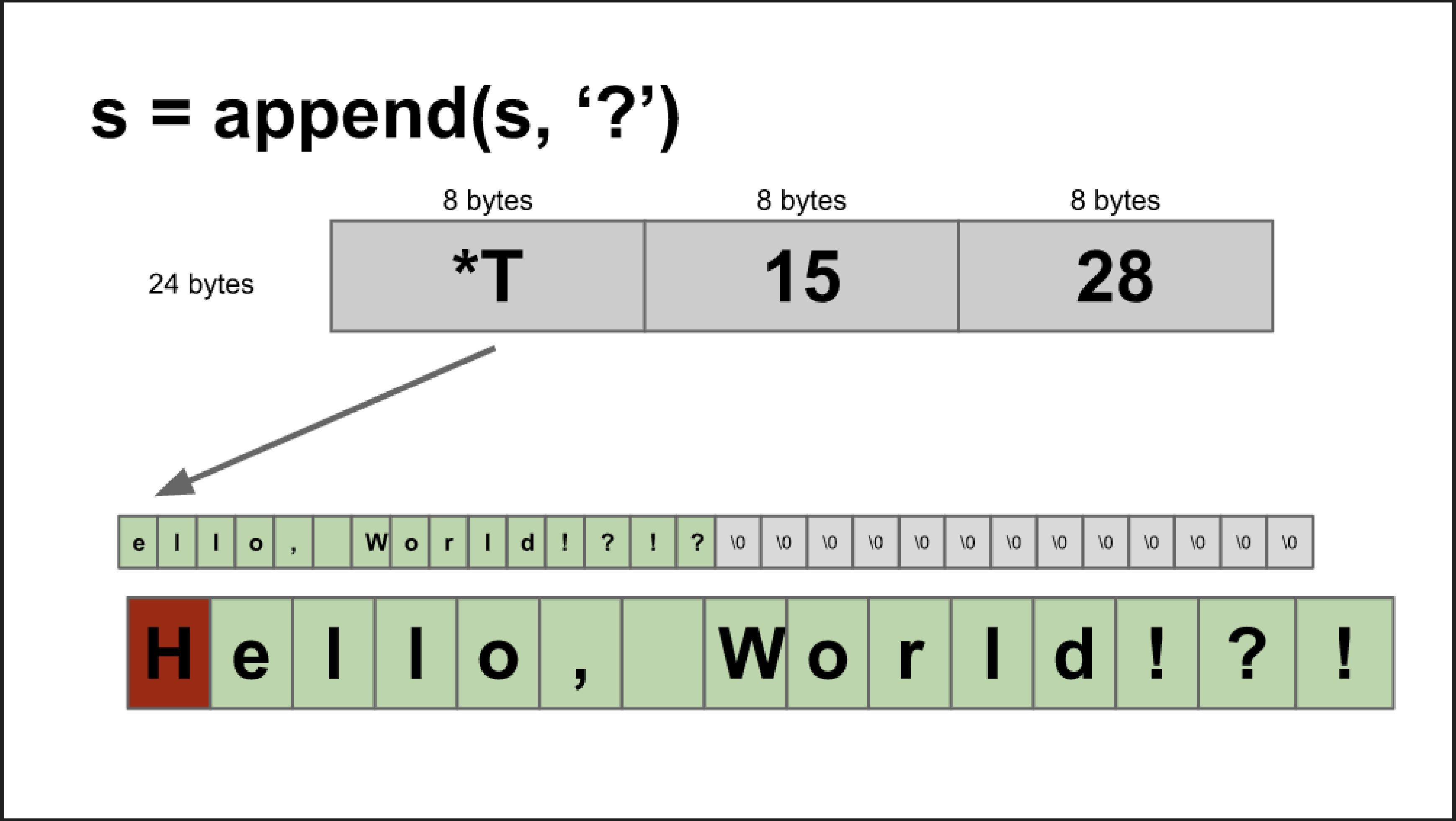
GO BUILT-IN DATA STRUCTURES



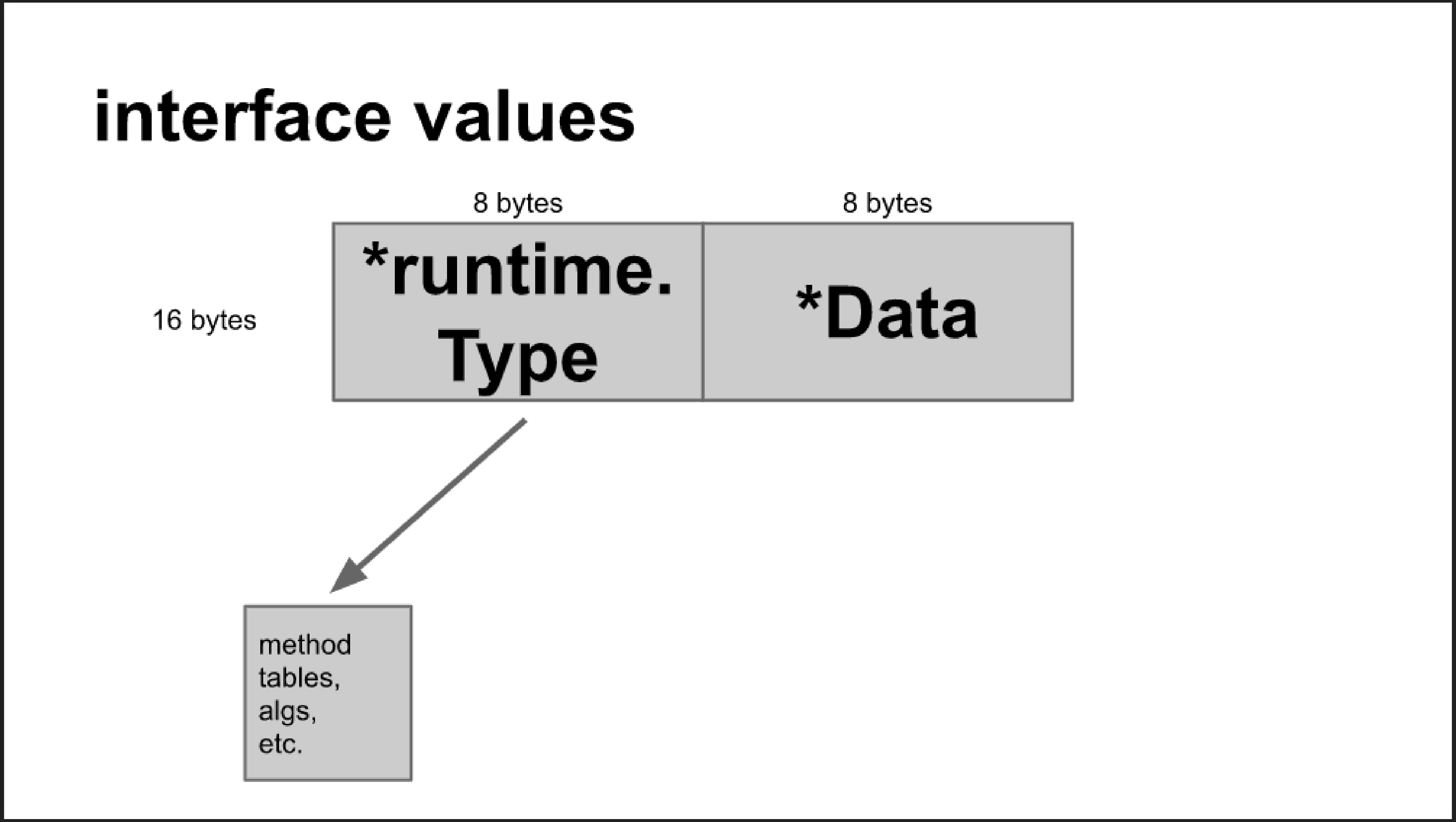
GO BUILT-IN DATA STRUCTURES



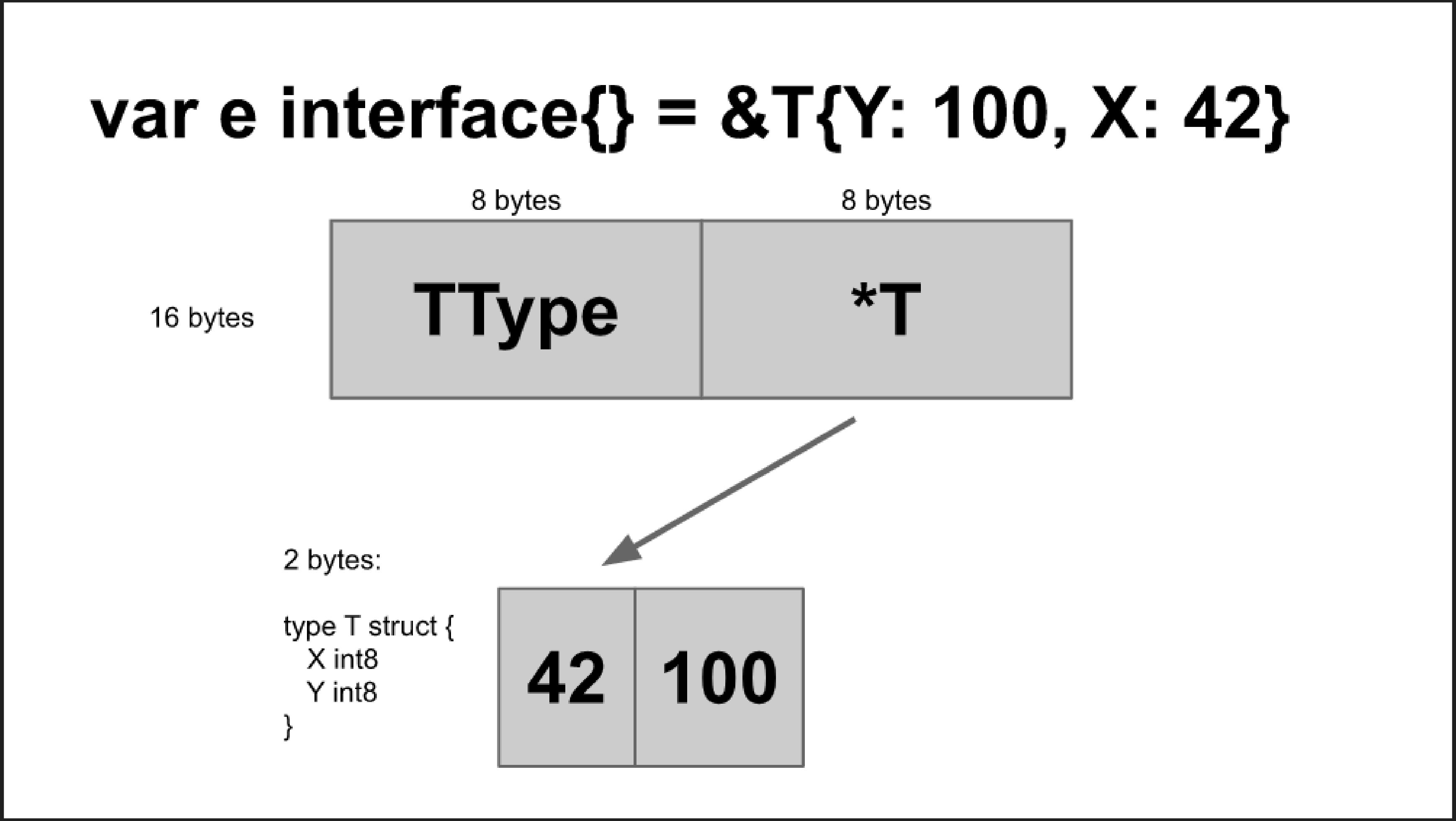
GO BUILT-IN DATA STRUCTURES


s = append(s, ‘?’)
8 bytes
8 bytes
8 bytes
24 bytes
*T
15
28
e l l o , W o r l d ! ? ! ? \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
H e l l o , W o r l d ! ? !

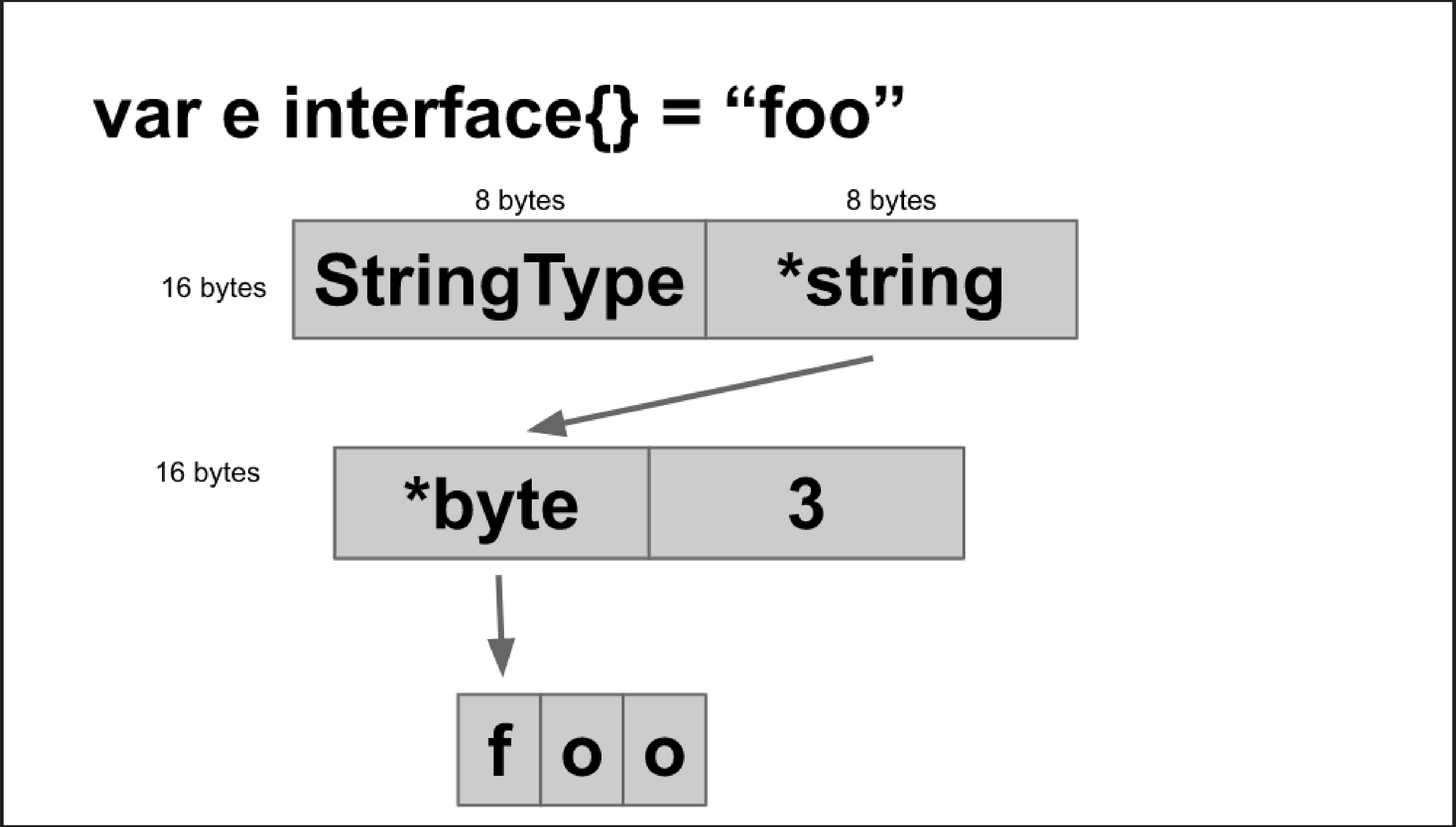
GO BUILT-IN DATA STRUCTURES



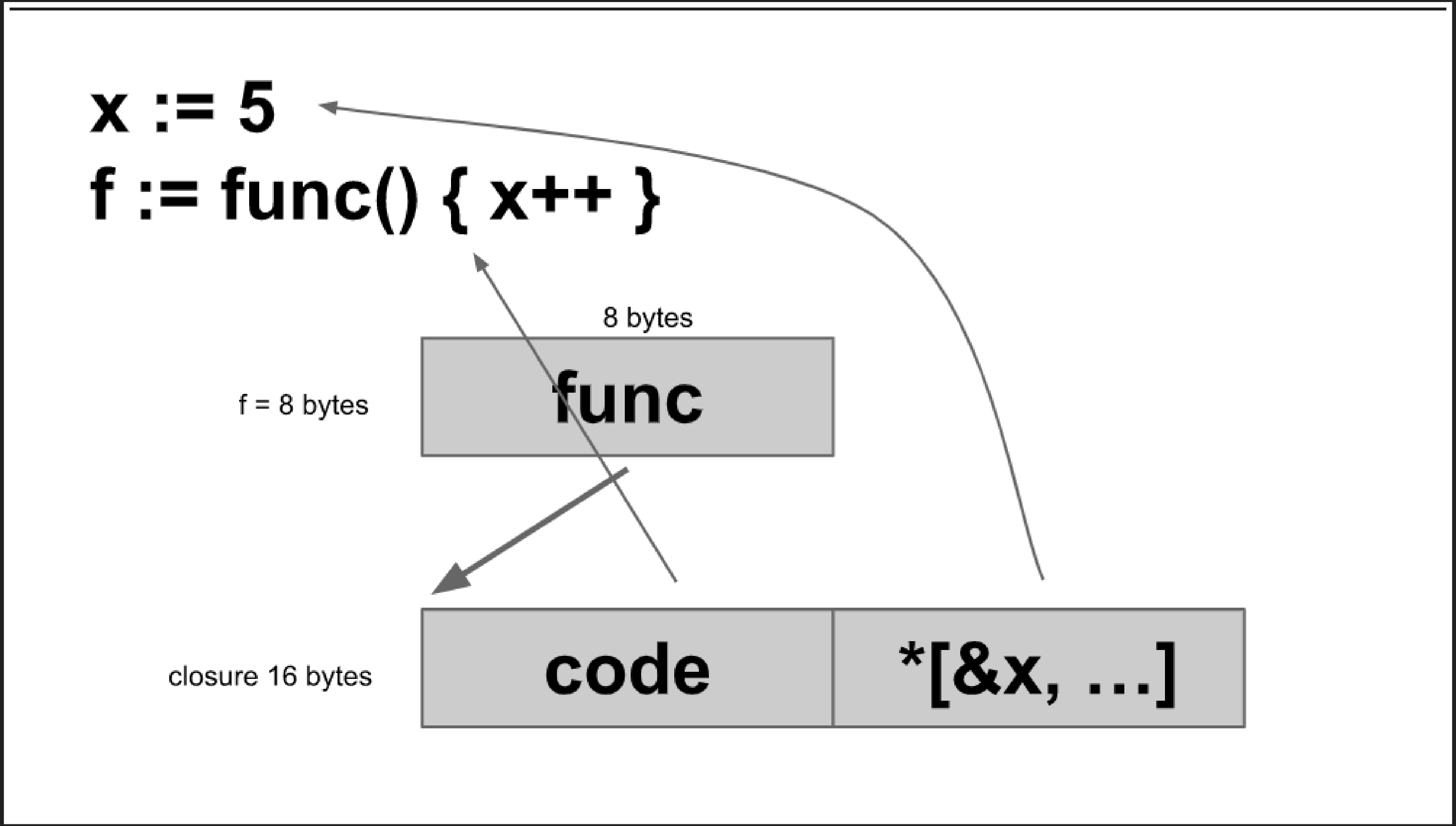
GO BUILT-IN DATA STRUCTURES



GO BUILT-IN DATA STRUCTURES



GO BUILT-IN DATA STRUCTURES


x := 5
f := func() { x++ }
8 bytes
f = 8 bytes
func
closure 16 bytes
code
*[&x, …]