WELCOME!

Develop Intelligence

Michael Van Sickle

Join Us in Making Learning Technology Easier

Develop Intelligence

Our mission…

Over 16 years ago, we embarked on a journey to improve the world by making learning technology easy and accessible to everyone.

…impacts everyone daily.

And it’s working. Today, we’re known for delivering customized tech learning programs that drive innovation and transform organizations.

In fact, when you talk on the phone, watch a movie, connect with friends on social media, drive a car, fly on a plane, shop online, and order a latte with your mobile app, you are experiencing the impact of our solutions.

Over The Past Few Decades, We've Provided

Over **62,300,000** expert-led learning hours

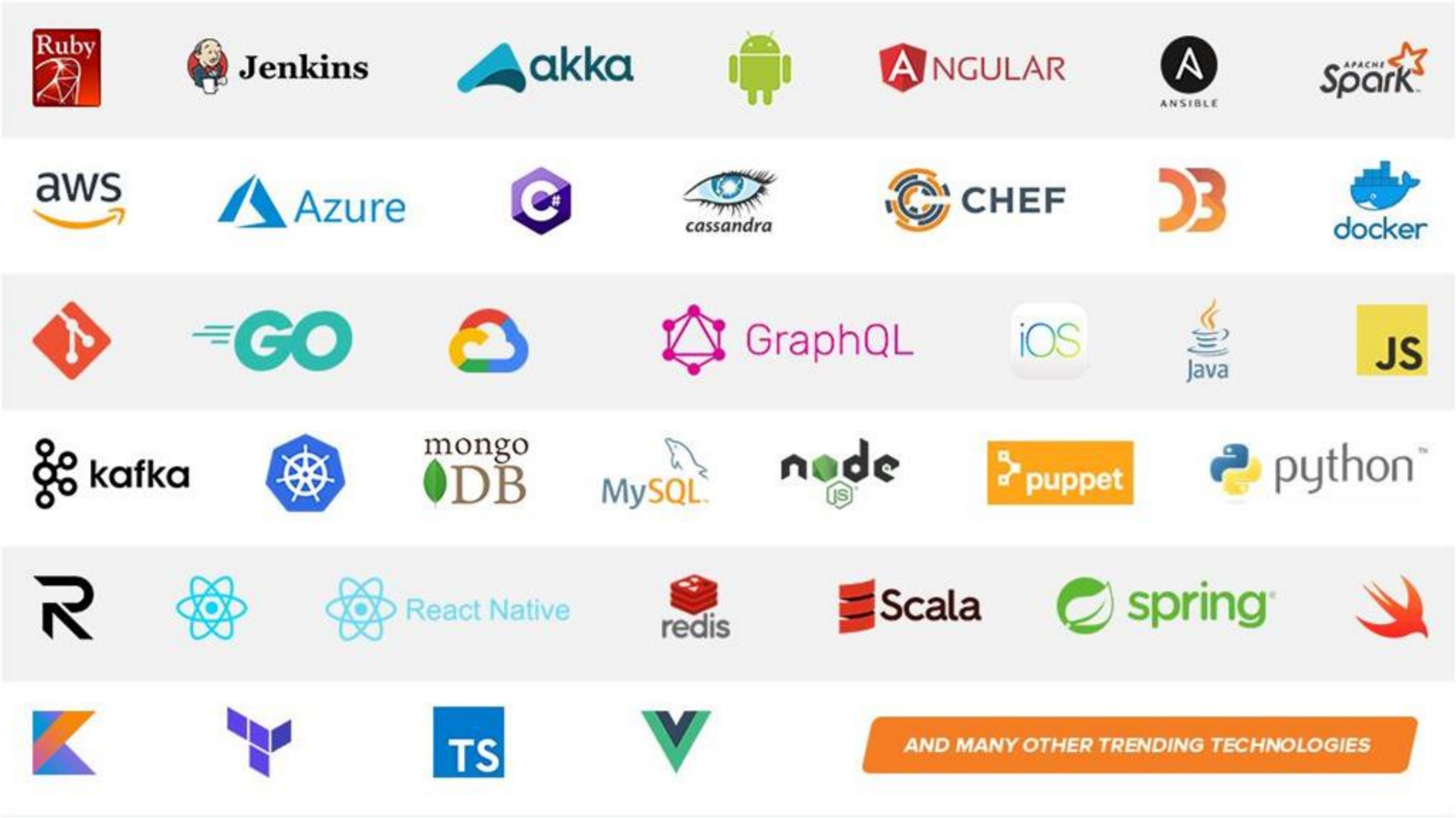
In 2019 Alone, We Provided

Training to over **13,500** engineers

Over **120** active trainers, with an average of over two decades of experience each.

Programs in **30** countries

Technologies we cover



Note About Virtual Trainings

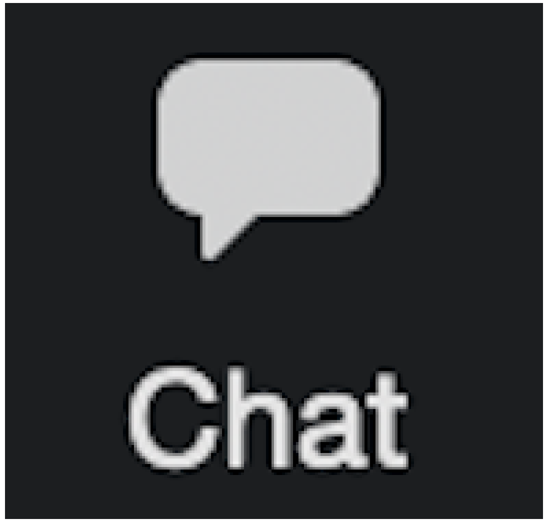
What we want

...what we've got

Virtual Training Expectations for You

Arrive on time / return on time

Mute unless speaking



Use chat or ask questions verbally

Virtual Training Expectations for Me

I pledge to:

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so
- Use an on-screen timer for breaks so you know when to be back

Recording Policy

Recordings are provided to participants who have attended the training, in its entirety. Recordings are provided as a value-add to your training, and should not be utilized as a replacement to the classroom experience.

Participants can expect the following:

- Recordings will be provided the Monday after class is completed
- Recordings will be provided for 14 days
- Recordings will be accessible as “View Only” status and cannot be copied
- Sharing recordings is strictly prohibited

To request recordings, please fill out the form linked in Learn++.

Thank you for your understanding and adhering to the policy.

Prerequisites

- Comfortable with basic concepts of Go
    - Variables, loops, logic, and concurrency mechanisms
- Familiarity with the most common parts of standard library
    - fmt, strings, errors, etc.
- Comfortable navigating standard library and adopting new parts of it

Objectives

- Reinforce your understanding of Go
- Explore best practices for solving many common problems
- Expand your understanding of Go’s testing and profiling tools
- Use live exercises and reviews to confirm your understanding

Administrative Issues

- Class time
- Breaks
- Lunch
- If you need to step away
  - Phone calls
  - Work emergencies
  - Family

Agenda

Develop Intelligence

- Language review
- Packages and Modules
- Concurrency
- HTTP Services
- Debugging
- Testing, Profiling, and Observability
- Code Generation
- Containerization
- Go Runtime

Course structure

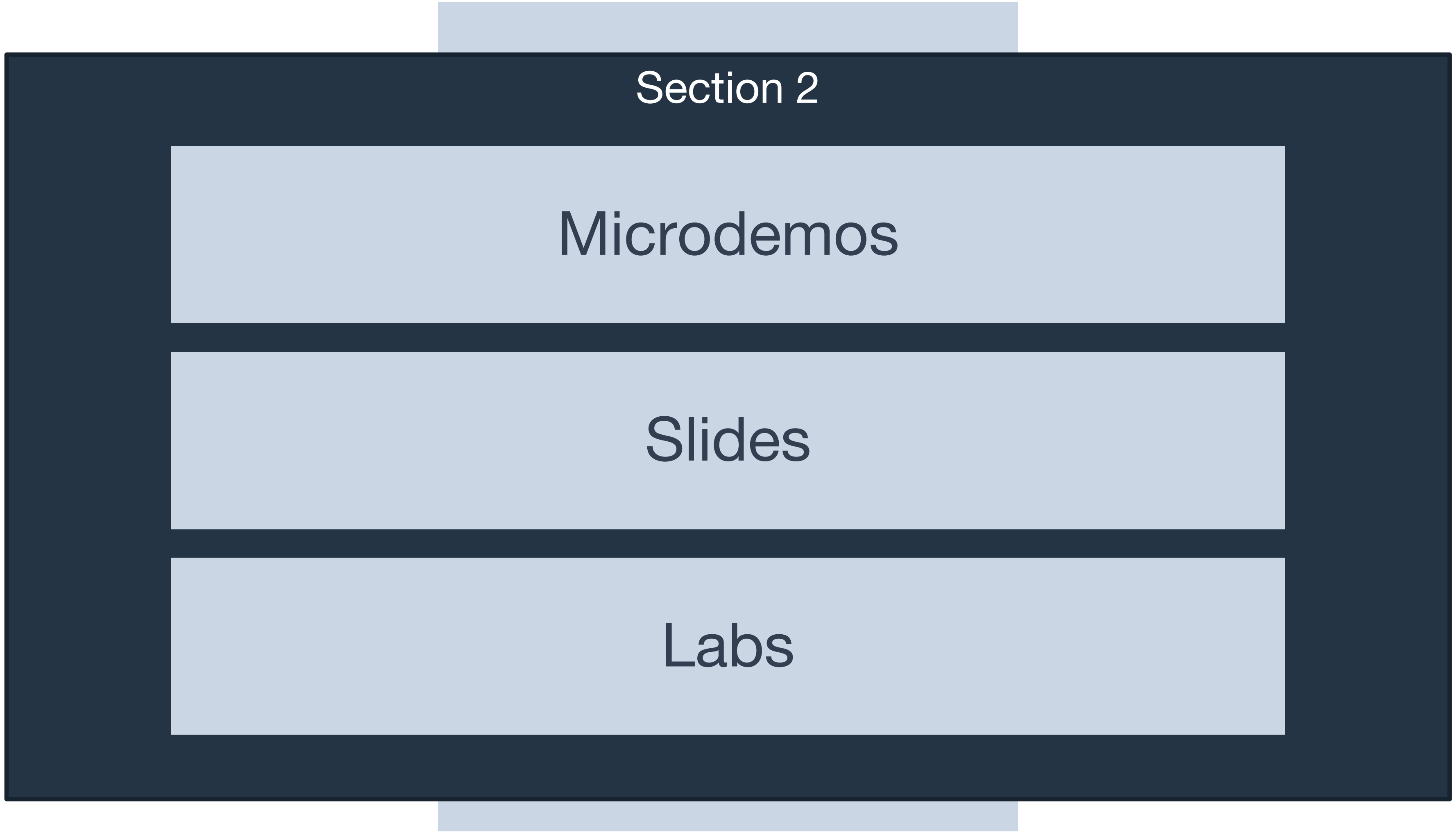
Section 1

Section 2

Section 3

Section …

Course structure



Section 2

Microdemos

Slides

Labs

DISCUSSION Time

❑Your name

❑Current role and time as a developer

❑Experience with Go

❑Hobbies

About me

- Michael VanSickle
- Instructor for DevelopIntelligence
- 20+ years of development experience
- 15+ years of training experience
- Live in Ohio with my wife, 2 daughters, and 2 mouthy cats
- Hobbies
  - Exercise
  - Volunteering
  - Exploring the local parks

Language Review

Develop Intelligence

The Basics

Language Review

log Package

flag Package

Concurrency

Contexts

Language Review

Topical Demos

In this lab, you will be finishing the implementation of an asynchronous logging service. This service is intended to receive HTTP messages and write the request body's content out a log file. You have been provided with the API and partial implementation of many functions and methods. Your goal is to finish the logging service and discuss design decisions that you made.

LAB

Packages

Develop Intelligence

Overview

Package Mechanics

Package-Oriented Design

Develop Intelligence

Packages

Package mechanics

Package Design Guidelines

- Use multiple files
- Keep types close
- Organize by responsibility

https://rakyll.org/style-packages/

Naming Packages

- Short and clear
- Lowercase
- No underscores
- Prefer Nouns
- Abbreviate judiciously

`package utilities`	◀ vague
`package data_layer`	◀ too long, contains underscores
`package dl`	◀ unclear
`package time`	◀ clear and concise
`package json`	◀ clear and concise

Develop Intelligence

Package Mechanics

Topical Demos

Scoping rules

```go
package user

type User struct {
      ID                 int
      Username           string
      password           string
}
func NewUser() *User {}
const maxUsers = 100
```

◀ public struct

◀ public field

◀ public field

◀ package-level field

◀ public function

◀ package-level constant

Package-level entities

```
// Package user manages users in our app.
package user

import “strings”

var currentUsers []*User
const MaxUsers = 100

// GetByID retrieves a copy of the
// user with the provided ID, if present.
// The second return value indicates
// if a user was found or not
func GetByID(id int) (User, bool) {}
```

◀ package comment

◀ package declaration

◀ import statement

◀ variable

◀ constant

◀ API comment

◀ function

Develop Intelligence

Packages

Package-Oriented Design

Designing a Package

Provide a clear solution

- Single responsibility
- Cohesive API

Focus on the consumer

- Simple to use
- Minimize API
- Encapsulate changes

Maximize reusability

- Reduce dependencies
- Minimize scope

Interface Strategies

concrete types

configuration

interfaces

behavior

Encapsulate changes

Maximize flexibility

concrete types

configuration and behavior

errors

avoid panics

Avoid abstracting too early

Interface Strategies

concrete types

net/http.Request

interfaces

net/http.Handler

Interface Strategies

concrete types

net/http.Response

errors

net/http.Get

Package Design Guidelines

Purpose

Usability

Portability

https://www.ardanlabs.com/blog/2017/02/design-philosophy-on-packaging.html

Package Design Guidelines - Purpose

- Packages must be named with the intent to describe what it provides.
- Packages must not become a dumping ground of disparate concerns.

Package Design Guidelines - Usability

- Packages must be intuitive and simple to use.
- Packages must respect their impact on resources and performance.
- Packages must protect the user’s application from cascading changes.
- Packages must prevent the need for type assertions to the concrete.
- Packages must reduce, minimize, and simplify their code bases

Package Design Guidelines - Portability

- Packages must aspire for the highest level of portability.
- Packages must reduce setting policies when it’s reasonable and practical.
- Packages must not become a single point of dependency.

The `main.go` file contains a simple RESTful API for a blogging service. Improve the organization of the program using packages. Make sure that packages are created following package-oriented design principles. Please note that the source code will need to be updated to ensure that the application continues to function.

LAB

Modules

Develop
Intelligence

Outline

Goals and Overview

Standard workflows

Versioning

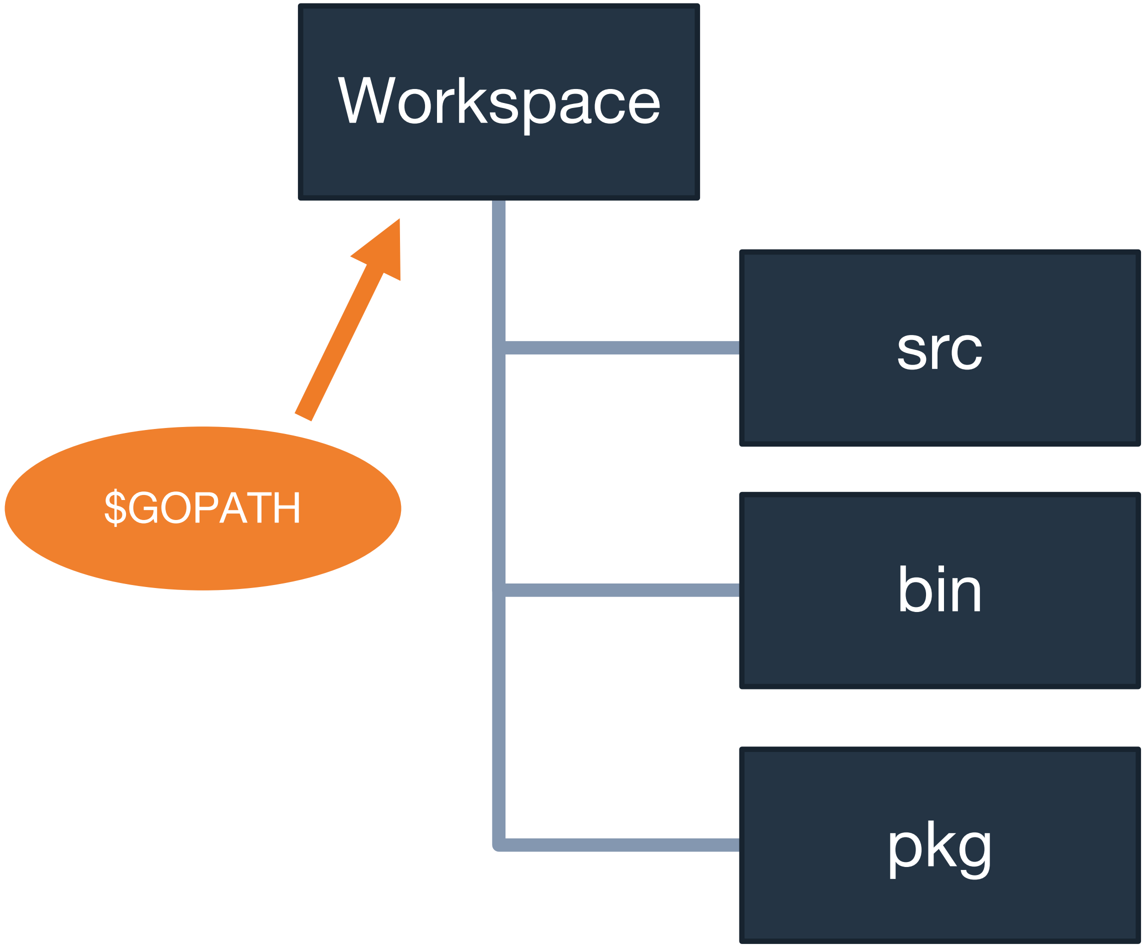
Identifying Conflicts

Develop Intelligence

Modules

Goals and Overview

Workspaces: Pre-cursor to Modules

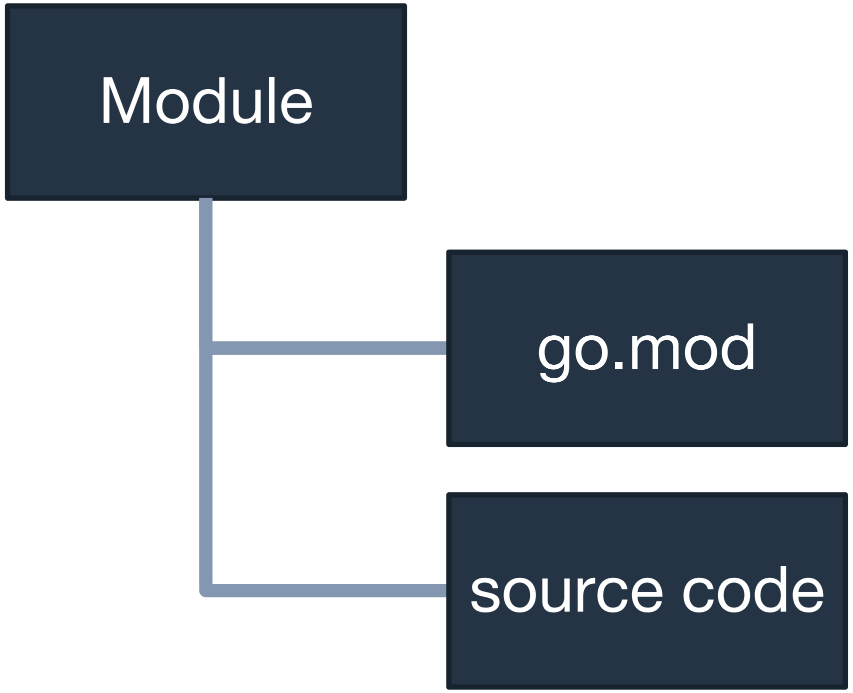


Application source code

Compiled programs

Compilation artifacts (e.g., package binaries)

Modules



Module

go.mod

source code

Module configuration

Compiled programs

Goals of Module System

Keep what works well with workspaces

Address weaknesses

Goal: Keep What Works Well

Retrieve dependencies

Simplify build process

Share projects*

Goal: Address Weaknesses

Versioning and API stability

Vendoring and reproducible builds

Overview of Modules

One or more related packages

Configured via go.mod file

Version controlled

Strict semantic versioning

Dependent libraries kept in cache

Integrity checks via checksums

Develop Intelligence

Modules

Standard Workflows

Develop Intelligence

Standard Workflows

Topical Demos

Common commands

`go mod init`	◀ Initialize a new module
`go get`	◀ Retrieve a module as a dependency
`go list -m`	◀ List module dependencies
`go mod verify`	◀ Verify module integrity
`go mod tidy`	◀ Remove unused dependencies
`go mod vendor`	◀ Download dependencies into vendor directory

Develop Intelligence

Modules

Versioning

Versioning

Semantic versioning

Changing major versions

Module queries

Develop Intelligence

Modules - Versioning

Semantic Versioning

Semantic Versioning

v1.5.3-pre1

Semantic Versioning

v1.5.3-pre1

v	Version prefix (required)
1	Major revision (likely to break backward compatibility)
5	Minor revision (new features, doesn’t break BC)
3	Patch (bug fixes, no new features, and doesn’t break BC)
pre1	Pre-release of new version, if applicable (text is arbitrary)

https://semver.org

Develop Intelligence

Modules - Versioning

Changing Major Versions

Develop Intelligence

Versioning

Topical Demos

Develop Intelligence

Modules - Versioning

Module Queries

Module Queries

go get github.com/gorilla/mux	◀ Retrieve latest published version of major version 0 or 1
go get github.com/gorilla/mux@latest	◀ Retrieve latest version of major version 0 or 1
go get github.com/gorilla/mux@v1.6.2	◀ Retrieve specific version
go get github.com/gorilla/mux/v2@v2.0.1	◀ Retrieve specific version beyond v1
go get github.com/gorilla/mux@main	◀ Retrieve main branch (might be master!)
go get github.com/gorilla/mux@<v1.6.2	◀ Retrieve most recent version before 1.6.2
go get github.com/gorilla/mux@>v1.6.2	◀ Retrieve first version after 1.6.2

Closest match wins!

Develop Intelligence

Modules

Identifying Conflicts

Develop Intelligence

Identifying Conflicts

Topical Demos

Identifying Conflicts

`go mod why -m`	◀	Why a module is included
`go mod graph`	◀	Print module tree in module / requirement pairs
`go mod edit`	◀	Edit the go.mod file programmatically
`go mod edit -replace=old=new`	◀	Replace module with another one
`go mod edit -exclude=path@version`	◀	Exclude module from build
`go mod edit -require=path@version`	◀	Include module in build (similar to go get ...)

Develop
Intelligence

Concurrency

Outline

Goroutines

Synchronization

Channels

Mutexes

Develop Intelligence

Goroutines

Guidelines

Guidelines

Goroutines are cheap – use them!

Know how a goroutine will stop when you start it

Use channels to communicate between goroutines

Use sync.WaitGroup to synchronize completion of tasks*

Develop
Intelligence

Channels

Outline

Buffered channels

Balancing producers and consumers

When to avoid channels

Develop Intelligence

Channels

Buffered Channels

Buffered Channels

```
ch := make(chan int)               // an unbuffered channel

ch = make(chan int, 10)            // a buffered channel
```

What is the difference between a buffered and unbuffered channel?

What are some advantages and disadvantages of buffered channels?

What are the alternatives to buffered channels?

Develop Intelligence

Buffered Channels

Topical Demos

Buffered channels

- Improve perf of production bursts
- Free producers from synchronization overhead
- Protect producers when messages might not have an active receiver

- Cannot easily resize buffer
- Extra memory required for buffer, whether used to not

Develop Intelligence

Channels

Balancing Producers and Consumers

Balancing Producers and Consumers

Topical Demos

Balancing Producers and Consumers

- The ideal number of producers and consumers is one
- Increase number of consumers to handle slow consumption
- Increase number of producers to avoid blocking signal sources
  - e.g., Web services that are sending data
- Avoid buffered channels to balance work loads

Beware of synchronization issues!

Develop Intelligence

Channels

When to Avoid Channels

When to Avoid Channels

- Channels are the best concurrency technique to use in Go
- Until they’re not!

When to Avoid Channels

Topical Demos

Channels

...are good at

- Synchronizing tasks
- Decoupling producers and consumers
- Transferring data ownership
- Distributing workloads
- Communicating async results

..are not so good at

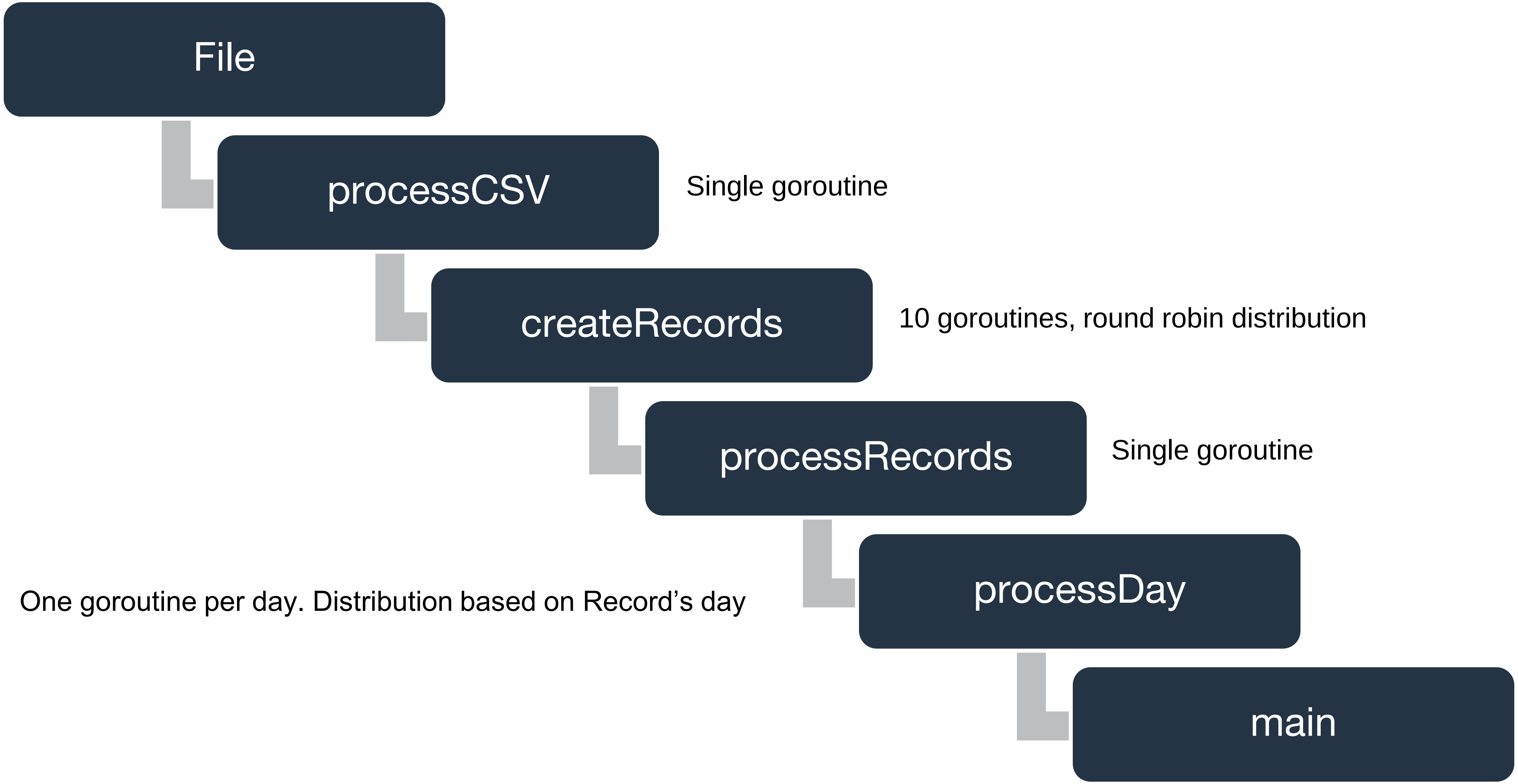
- Synchronizing memory
- Controlling access to shared resource
- Code with extremely high performance requirements

https://tinyurl.com/y25u4les

In this lab, you will complete an application that is performing an ETL (extract, transform, and load) operation. The application's goal is to process a data extract from the [NOAA Local Climatological Database (LCD)](https://www.ncdc.noaa.gov/cdo-web/datatools/lcd) and generate a report indicating the minimum, maximum, and average temperature for each day in the dataset. In order to maximize processing speed, this application will perform its calculations using multiple goroutines.

LAB

Goroutine Lab – ETL Processing



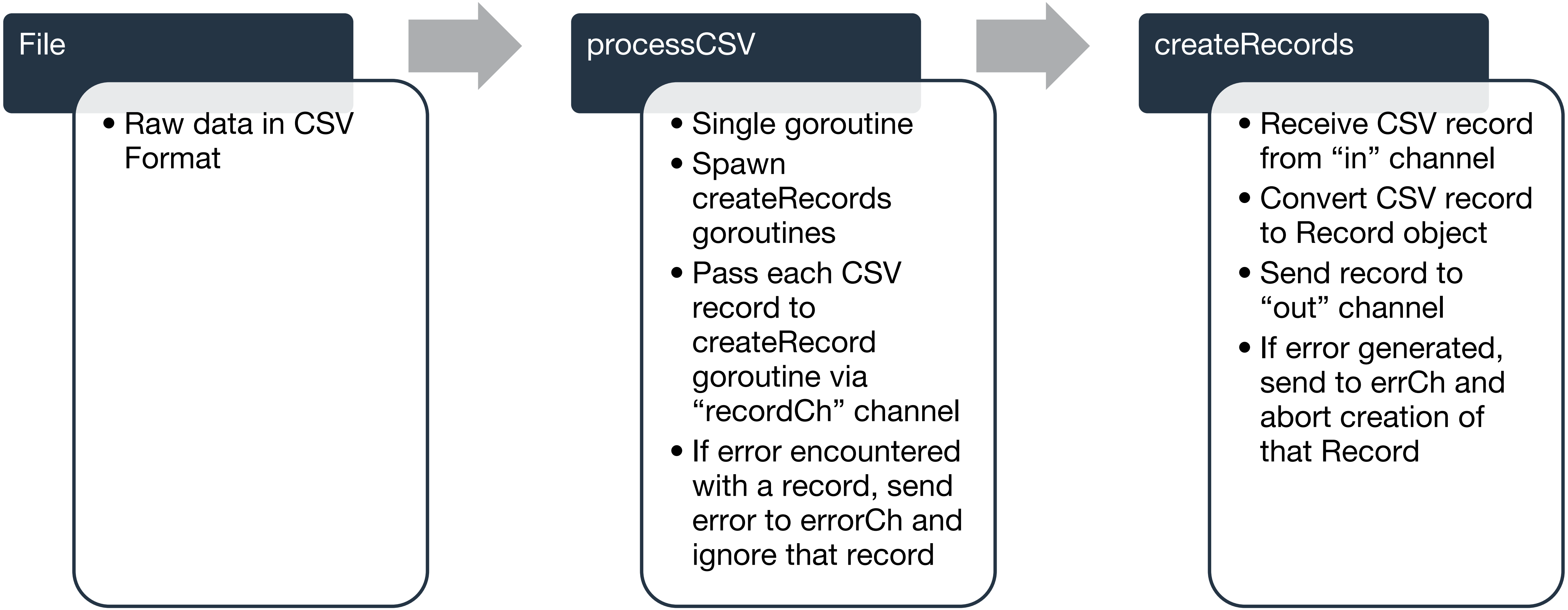
Single goroutine

10 goroutines, round robin distribution

Single goroutine

One goroutine per day. Distribution based on Record’s day

Goroutine Lab – ETL Processing



File

- Raw data in CSV Format

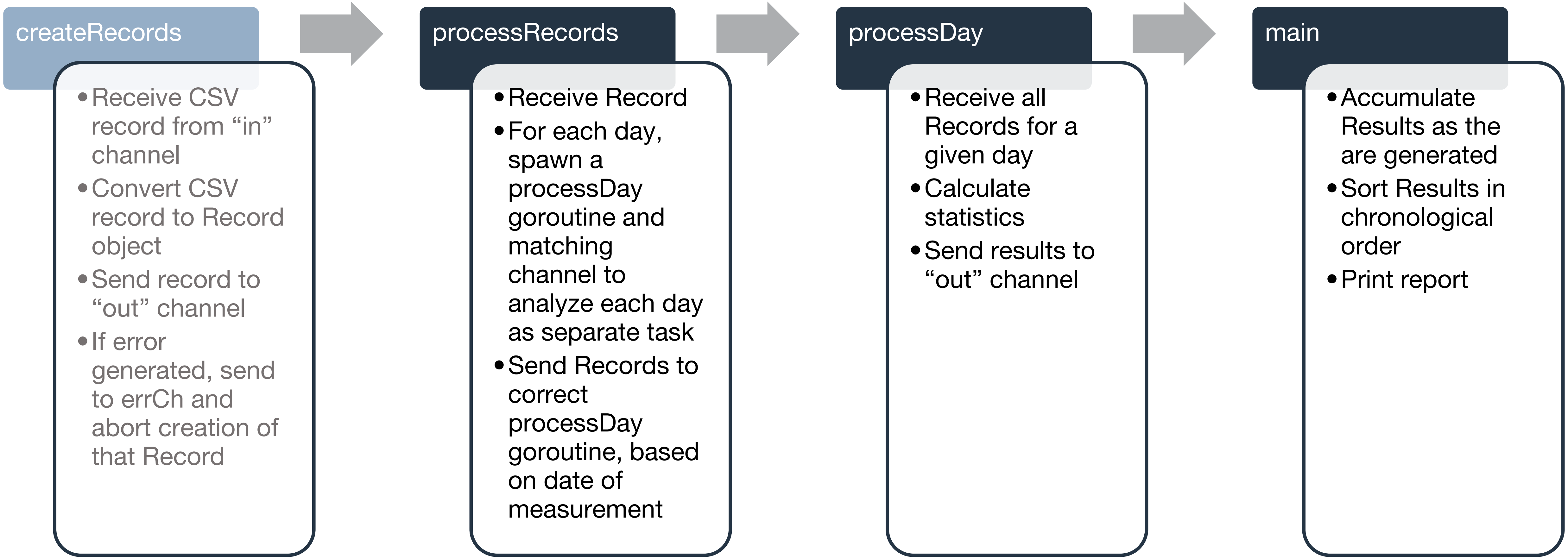
processCSV

- Single goroutine
- Spawn createRecords goroutines
- Pass each CSV record to createRecord goroutine via “recordCh” channel
- If error encountered with a record, send error to errorCh and ignore that record

createRecords

- Receive CSV record from “in” channel
- Convert CSV record to Record object
- Send record to “out” channel
- If error generated, send to errCh and abort creation of that Record

Goroutine Lab – ETL Processing



createRecords

- Receive CSV record from “in” channel
- Convert CSV record to Record object
- Send record to “out” channel
- If error generated, send to errCh and abort creation of that Record

processRecords

- Receive Record
- For each day, spawn a processDay goroutine and matching channel to analyze each day as separate task
- Send Records to correct processDay goroutine, based on date of measurement

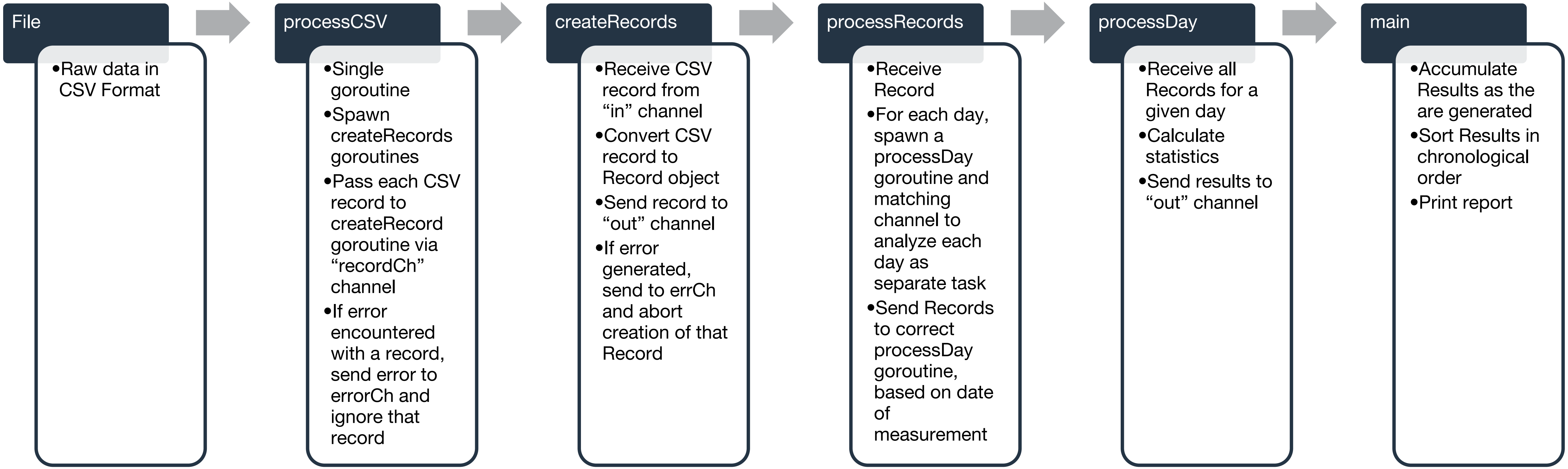
processDay

- Receive all Records for a given day
- Calculate statistics
- Send results to “out” channel

main

- Accumulate Results as the are generated
- Sort Results in chronological order
- Print report

Goroutine Lab – ETL Processing



File

- Raw data in CSV Format

processCSV

- Single goroutine
- Spawn createRecords goroutines
- Pass each CSV record to createRecord goroutine via “recordCh” channel
- If error encountered with a record, send error to errorCh and ignore that record

createRecords

- Receive CSV record from “in” channel
- Convert CSV record to Record object
- Send record to “out” channel
- If error generated, send to errCh and abort creation of that Record

processRecords

- Receive Record
- For each day, spawn a processDay goroutine and matching channel to analyze each day as separate task
- Send Records to correct processDay goroutine, based on date of measurement

processDay

- Receive all Records for a given day
- Calculate statistics
- Send results to “out” channel

main

- Accumulate Results as the are generated
- Sort Results in chronological order
- Print report

Develop Intelligence

HTTP Services

Outline

Basics

Routing Requests

Middleware

Develop Intelligence

HTTP Services

The Basics

Develop Intelligence

The Basics

Topical Demos

The Basics

```
import “net/http”
```

◀ Basic package for creating HTTP clients / servers

```
http.HandleFunc(“{route}”,
      http.HandlerFunc)
```

◀ Register a function to handle requests

```
func(http.ResponseWriter, *http.Request){}
```

◀ Signature of http.HandlerFuncs

```
http.Handle(“{route}”,
      http.Handler)
```

◀ Register an object to handle requests

```
func (...) ServeHTTP(http.ResponseWriter,
      *http.Request)
```

◀ Method required to meet http.Handler interface

The Basics

```
err := http.ListenAndServe(addr string,
      handler http.Handler)

err := http.ListenAndServeTLS(
      addr string,
      certFile string,
      keyFile string,
      handler http.Handler)
```

Pass ‘nil’ for http.Handler to use http.DefaultServeMux!

◀ Start listening for requests with default http.Server

◀ Start listening for secure request (e.g. https) with default http.Server

Develop
Intelligence

HTTP Services

Routing Requests

Develop Intelligence

Routing Requests

Topical Demos

Route matching

Best match wins!

```
http.Handle(“/foo”, ...)
```

◀ Match foo resource (e.g., a file).

- Trailing slash will be added automatically, if no more specific route available.

```
http.Handle(“/foo/”, ...)
```

◀ Match resources within foo (e.g, a directory)

```
http.Handle(“/foo/42”, ...)
```

◀ Match /foo/42 resource

Route matching – path splitting

```
// pattern /split/users/{:username}/roles/{:roleID}
func parametricRoutesSplit(w http.ResponseWriter, r *http.Request) {
    segments := strings.Split(r.URL.Path, "/")
    if len(segments) != 6 {
        w.WriteHeader(http.StatusNotFound)
        return
    }
    username := segments[2]
    roleIDRaw := segments[5]
    ...
```

Route matching – regular expressions

```
// pattern /regexp/users/{:username}/roles/{:roleID}
var pattern = regexp.MustCompile(`^\/regexp\/users\/(\S+?)\/roles\/(\d+?)$`)

func parametricRoutesRegexp(w http.ResponseWriter, r *http.Request) {
      matches := pattern.FindStringSubmatch(r.URL.Path)
      if len(matches) == 0 {
            w.WriteHeader(http.StatusNotFound)
            return
      }
      username := matches[1]
      roleIDRaw := matches[2]
      ...
```

Develop Intelligence

HTTP Services

Middleware

Develop Intelligence

Middleware

Topical Demos

Middleware structure

```
type myMiddleware struct{
      next   http.Handler
}

func (mw myMiddleware) ServeHTTP(w http.ResponseWriter, r *http.Request) {
      // manipulate request and/or response before primary handling
      // Note: add required headers here!

      mw.next.ServeHTTP(w, r) // pass to next handler (middleware or route handler)

      // manipulate response before returning to requester
}
```

Middleware - Global

```
http.ListenAndServe(“...”, nil)

http.ListenAndServe(“...”, &myMiddleware{
      next: nil
})
```

◀ Listen for requests with DefaultServeMux

◀ Wrap DefaultServeMux with global middleware

Middleware – Route Specific

```
http.Handle(“...”, &myHandler{})

http.Handle(“...”, &myMiddleware{
      next: &myHandler{},
})
```

- Register a handler on a route
- Wrap handler with route-specific middleware

- Create a program that contains a route-specific middleware.
    - The matched route should be of the form /users/{username}/roles/{roleID}
        - {username} and {roleID} are parametric and don’t have a fixed value
        - the handler should return the username and roleID to the requester
    - Create middleware for the above route that only allows the username "grogu" into the main handler, all other routes should receive a 401 - UNAUTHORIZED response
    - Add a handler for the root route that returns a 404 - NOT FOUND response
- BONUS:
    - create a global middleware that logs the path of each request to the default logger

LAB

Develop Intelligence

Debugging

Outline

Overview

Debugging with VS Code

Debugging

Overview

Develop Intelligence

Overview

Topical Demos

Debugging

Debugging with VS Code

Develop Intelligence

Debugging with VS Code

Topical Demos

Testing Go Programs

Outline

Review types

Table-driven tests

Testing web services

Testing

Review Test Types

Develop Intelligence

Test Types

Topical Demos

Testing

`foo_test.go`	◀ Test file name
`package foo`	◀ Package declaration for white-box testing
`package foo_test`	◀ Package declaration for black-box testing
`func TestFoo(t *testing.T) {}`	◀ Test function name
`go test`	◀ Run tests in current package
`go test {pkg1} {pkg2}`	◀ Run tests for specified packages
`go test ./...`	◀ Run tests for current package and descendants

Reporting test failures

Non-immediate failures

- `t.Fail()`
- `t.Error(...interface{})`
- `t.Errorf(string, ...interface{})`

Immediate failures

- `t.FailNow()`
- `t.Fatal(...interface{})`
- `t.Fatalf(string, ...interface{})`

Benchmark tests

```
func BenchmarkFoo(b *testing.B) {
      // setup code
      b.ResetTimer()
      for i := 0; i < b.N; i++ {
            ...
      }
      b.StopTimer()
      // tear down code
}

go test -bench .

go test -bench . -benchtime 1m
```

◀ Benchmark test signature

◀ Start benchmark timer

◀ Run benchmarked iterations

◀ Stop benchmark timer

◀ Include benchmark tests in test run

◀ Tune b.N to run tests for approx. 1 minute

Example tests

```
func ExampleFoo() {
        fmt.Println(“Hello,“)
        fmt.Println(“World”)
        // Output:
        // Hello,
        // World
}
```

◀ Example test signature

◀ Start describing expected output to stdout

```
func Example{FunctionName}
func Example{TypeName}
func Example{Type}_{Method}
func Example{*}_suffix
```

◀ Example for function

◀ Example for type

◀ Example for type’s method

◀ Description of example test

```
godoc
```

◀ Tool to view documentation, including examples
`golang.org/x/tools/cmd/godoc`

Develop Intelligence

Testing

Table-driven Tests

Develop Intelligence

Table-driven Tests

Topical Demos

Testing

Testing Web Services

Develop Intelligence

Testing Web Services

Topical Demos

- Create two benchmark tests that compare the performance of the `encoding/json` package's Marshal function with Encoders.
    - The first benchmark test should marshal the `toMarshal` object into a JSON representation
        - the benchmark should capture the Marshal() call and the writing of that result to a `bytes.Buffer`
        - Note: the buffer can be shared between tests, but should be reset after each iteration.
        - Note: errors should be ignored
    - The second benchmark test should create a json.Encoder and use it's Encode method to create the JSON representation
        - Only the Encode() method call should be included in the benchmark
        - Use a shared `bytes.Buffer` to capture the result
        - Note: the buffer should be reset after each iteration.
        - Note: errors should be ignored

LAB

Profiling Go Programs

Develop Intelligence

Outline

Code coverage reports

Profiling programs

Profiling web services

Develop Intelligence

Profiling

Code Coverage Reports

Develop Intelligence

Code Coverage

Topical Demos

Code Coverage Reports

```
go test -cover
```

◀ Run tests with basic coverage stats

```
go test -coverprofile cover.out
```

◀ Generate coverage report to cover.out

```
go tool cover
```

◀ Analyze coverage report

```
go test -coverprofile cover.out
      -covermode count
```

◀ Set cover mode

- set – is statement executed
- count – execution count
- atomic – execution count (threadsafe)

Profiling

Profiling Programs

Develop Intelligence

Profiling Programs

Topical Demos

Profiling

`go test -{profiletype} {dest}`	◀ Run test with profile type
`go test ... -{profiletype}Rate {num}`	◀ Set profiling rate
`go tool pprof myprofile.out`	◀ Analyze profile with pprof
`go tool pprof -http localhost:3000 prf.out`	◀ Explore profile with local web server

http://graphviz.org/

Profiling Options

- blockprofile
- cpuprofile
- memprofile
- mutexprofile
- trace

Profiling

Profiling Web Services

Profiling Programs

Topical Demos

Profiling

```
import _ “net/http/pprof”


go tool pprof http://localhost:8000/debug/pprof/heap                  // memory
go tool pprof http://localhost:8000/debug/pprof/profile               // CPU
go tool pprof http://localhost:8000/debug/pprof/block                 // goroutines
go tool pprof http://localhost:8000/debug/pprof/trace?seconds=5       // trace


http://localhost:8000/debug/pprof                                      // website
```

The program created in the goroutines lab contains a large amount of concurrent tasks. In this lab, you'll analyze a slightly modified version of that lab's solution to what aspects are using the most resources.

- write a benchmarking test that only tests the time required to run the `execute` function
  - use the `bald-mountain_co.csv` file for raw data, but load it's data only one time (e.g. outside of the benchmark loop)
  - Note: the application closes channels when it's done with them. You will need to recreate the channels for each iteration
    - make sure that this time is excluded from the benchmark time
- Run the benchmarking test and determine how much time the analysis takes as well as how much memory is required
- Determine which part of the program consumes the largest amount of memory
- Determine which part of the program requires the greatest amount of CPU time
- Determine what part of the program is the source of the greatest delays
    - Hint: use a block profile

Observability

Outline

Measuring Resource Usage

Execution Tracing

Observing Web Services

Observability

Measuring Resource Usage

Develop Intelligence

Measuring Resource Usage

Topical Demos

Measuring Resource Usage

```
runtime.NumCPU()

runtime.Version()

runtime.NumGoroutine()

var ms runtime.MemStats
runtime.ReadMemStats(&ms)

https://golang.org/pkg/runtime/#MemStats
```

- Report number of logical CPUs on machine
  - Not same as runtime.GOMAXPROCS!
- Version of Go used to compile program
- Number of goroutines currently allocated
- Structure to store memory snapshot
- Take memory snapshot
- All the stats!

Develop Intelligence

Observability

Execution Tracing

Develop Intelligence

Execution Tracing

Topical Demos

Generating execution traces with tests

`go test .`	◀ Run all tests
`go test . -trace=trace.out`	◀ Run tests and generate trace (trace.out)
`go tool trace trace.out`	◀ Analyze an execution trace

Generating execution traces in code

```
import “runtime/trace”

trace.Start(io.Writer)
trace.Stop()

reg := trace.StartRegion(
      context.Context,
      regionType string)
reg.End()

ctx, task := race.NewTask(context.Context,
      taskType string)
task.End()
```

◀ Package containing trace API

◀ Start a manual trace, writing results to io.Writer

◀ Stop trace

◀ Create a region to trace execution time

◀ End region in trace

◀ Create a trace for a user-defined operation

◀ End task in trace

Develop Intelligence

Observability

Observing Web Services

Develop Intelligence

Observing Web Services

Topical Demos

Observing Web Services

```
import _ “net/http/pprof”
```

- Package to adding tracing code to service
  - Don’t use in production!

```
wget -O trace.out
       {host}/debug/pprof/trace?seconds=5
```

- Generate execution trace for 5 seconds and record results to trace.out
  - Can use cURL, Postman, or any other tool that can retrieve binary result from call

```
go tool trace trace.out
```

- Analyze execution trace of web service call

Code Generation

Develop Intelligence

Develop Intelligence

Code Generation

Topical Demos

Code Generation

```
go generate


//go:generate {command} {argument...}


$GOARCH       // execution architecture
$GOOS         // execution operating system
$GOFILE       // base name of file
$GOLINE       // line containing directive
$GOPACKAGE    // package containing $GOFILE
$DOLLAR       // literal dollar sign ($)
              // used for var expansion
```

◀ Command to initiate code generation

◀ Code generation directive

◀ Preset environment variables

Create a program that will generate a package with strongly typed `Add` functions of the form `f(A, B) -> A + B` (e.g. f(1, 2) -> 3).

- The function named should include the type: `AddInt`, `AddFloat32`, etc.
- The types to be generated should be passed into the program via a flag
- The destination package should also be passed in as a flag
- The generated code should include the standard comment:
  - `// Code generated DO NOT EDIT`
- Create a test program that contains the correct directive to generate Add functions for complex64s and strings.
  - Specify the destination package to be `main` using code generation environment variables
- Test the program to ensure it functions as expected

LAB

Develop
Intelligence

Containerization

Containerization

The Basics

End-End Testing

Debugging Containerized Apps

Develop
Intelligence

Go Runtime

Outline

Scheduler

Garbage Collector

GODEBUG

Develop Intelligence

Scheduler

Topical Demos

Useful Go Scheduler functions

`runtime.Gosched()`	◀ Current goroutine yields OS thread
`runtime.Goexit()`	◀ Exit the current goroutine (deferreds fire!)
`runtime.LockOSThread()`	◀ Bind current goroutine to current OS thread
`runtime.UnlockOSThread()`	◀ Unbind current goroutine from OS thread
`GOMAXPROCS(i int) int`	◀ Set/get max number of OS threads

Garbage Collector

Topical Demos

Useful Go Scheduler functions

```
runtime.GC()

runtime.SetFinalizer(obj, finalizer)

runtime.KeepAlive(obj)
```

◀ Manually trigger garbage collection

◀ Set function to run when obj is about to be GC’d

◀ Keep obj alive until KeepAlive call
  - mostly for interop

Scheduler

Topical Demos

THANK YOU

Develop
Intelligence