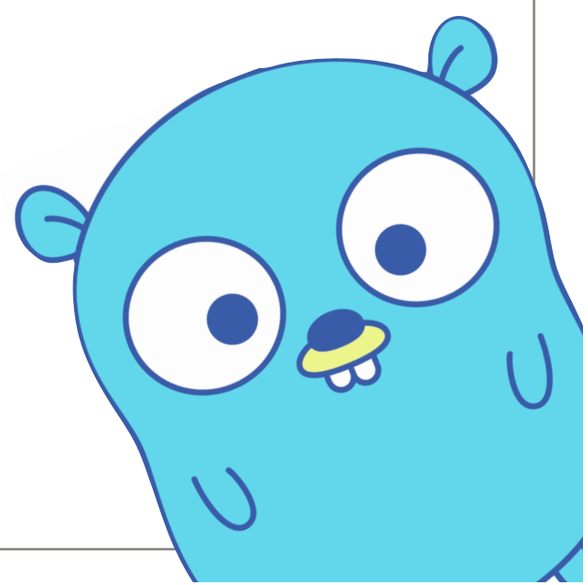
UP n Running with...

CONCURRENCY

...in GO

SUPERCHARGE your code
for maximum performance
on modern hardware



```
>>> UPnRunning.live()
```

Welcome!
Course Overview

Who is this course designed for?
Developers who ....

- are curious about concurrency in general - any language
- have some experience with Go, but little or no experience with how to implement concurrency
- have interest but no experience with Go, and want to see its concurrency model before committing to the language

Welcome!
Course Overview

Why learn concurrency in Go?

- SPEED and SCALABILITY / improved performance (10X improvement potential)
- Trends of increased latency and multi-core CPUs create both a NEED FOR and an OPPORTUNITY FOR concurrency
- It's what Go was designed for. Perhaps its greatest feature
- Or maybe it's all about the cash!

```
>>> UPnRunning.live()
```

Welcome!
Course Overview

Here's some of what we'll cover in this course:

- Concurrency theory, terminology, and parallels to everyday life
- wait groups, channels
- I/O-bound vs CPU-bound processes
- concurrency patterns - worker pools
- race conditions, deadlocks
- mutexes, condition variables, atomic variables

```
>>> UPnRunning.live()
```

Welcome!
Course Overview

How this course works (PLEASE don't skip!)

People learn by DOING and REINFORCEMENT

- Theory - "tell me"
- Demonstration - "show me"
- Hands-On - "let me try"
- Self-sufficiency - "I'm Up n Running"

```
>>> UPnRunning.live()
```

Welcome!
Course Overview

How this course works (PLEASE don't skip!)

Just a few recommendations for time management:

1. Plan your schedule - working (w/ breaks) and playing
2. Protect your study time, AND your "play" time - BOTH are essential
3. Focus! - consider headphones w N/C & light music
4. Celebrate your successes!

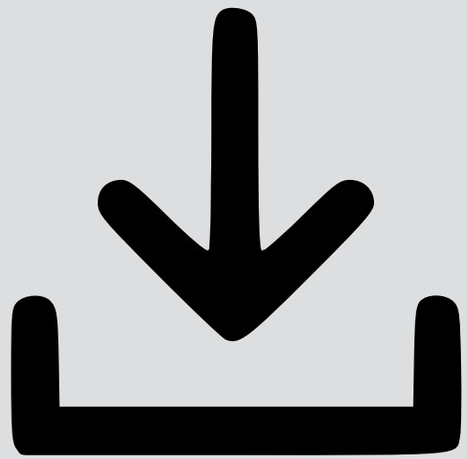
>>> UPnRunning.live()

Welcome!
Course Overview

How this course works (PLEASE don't skip!)

Downloading and Installing Course Resources:

- Master zip file downloaded at beginning of third section, organized by lesson sequence
- Minimal downloads within each section for greater efficiency
- The zip file contains all source code, concurrency sheet cheat, resource links, coding music playlist, all these slides as PDF, and more.



```
>>> UPnRunning.live()
```

Welcome!
Course Overview

A quick primer on the “classroom”

- If you’re not familiar with this platform, be sure to take a quick tour and become familiar.
- On the right-hand navigation, quickly jump to any lecture. Expand or collapse each section. Use checkboxes next to each lecture to track your progress.
- On the toolbar at the bottom of the video, you can play / pause, fast-forward / rewind, adjust volume, video speed, quality, add notes and switch in/out of full-screen mode.

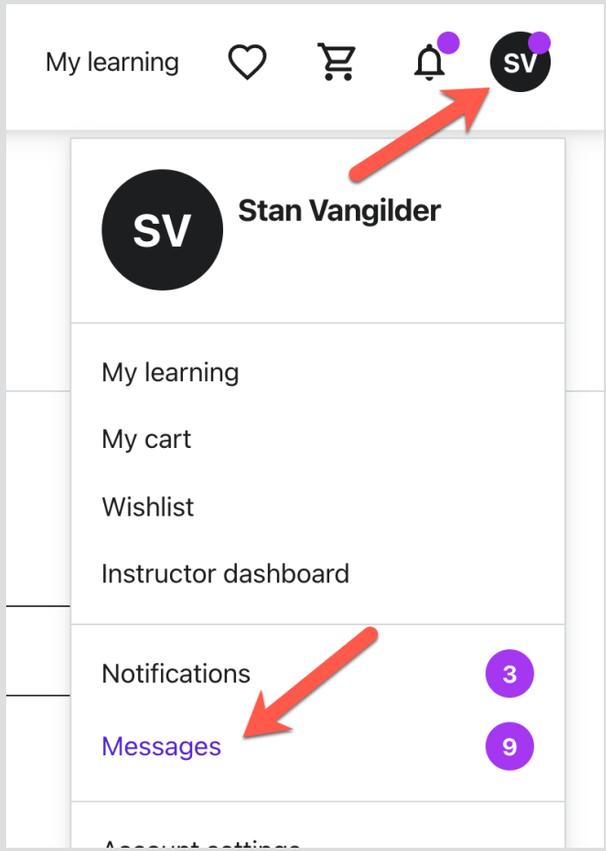
Welcome!
Course Overview

How to get HELP

- **Course content:** Message me through this platform - Click top right circular icon - then “Messages”. Or Use the Q&A forums below the video player.
- **Go language syntax, IDE or error messages:** Google or Stack Overflow is probably your fastest option.
- **Platform technical issues:** Contact Udemy @ support@udemy.com

```
>>> UPnRunning.live()
```

Welcome!
Course Overview



How to message me

>>> UPnRunning.live()

Welcome!
Course Overview

Ready? Let's

>>> UPnRunning.live()

Understanding Concurrency: Trends, Benefits and Challenges

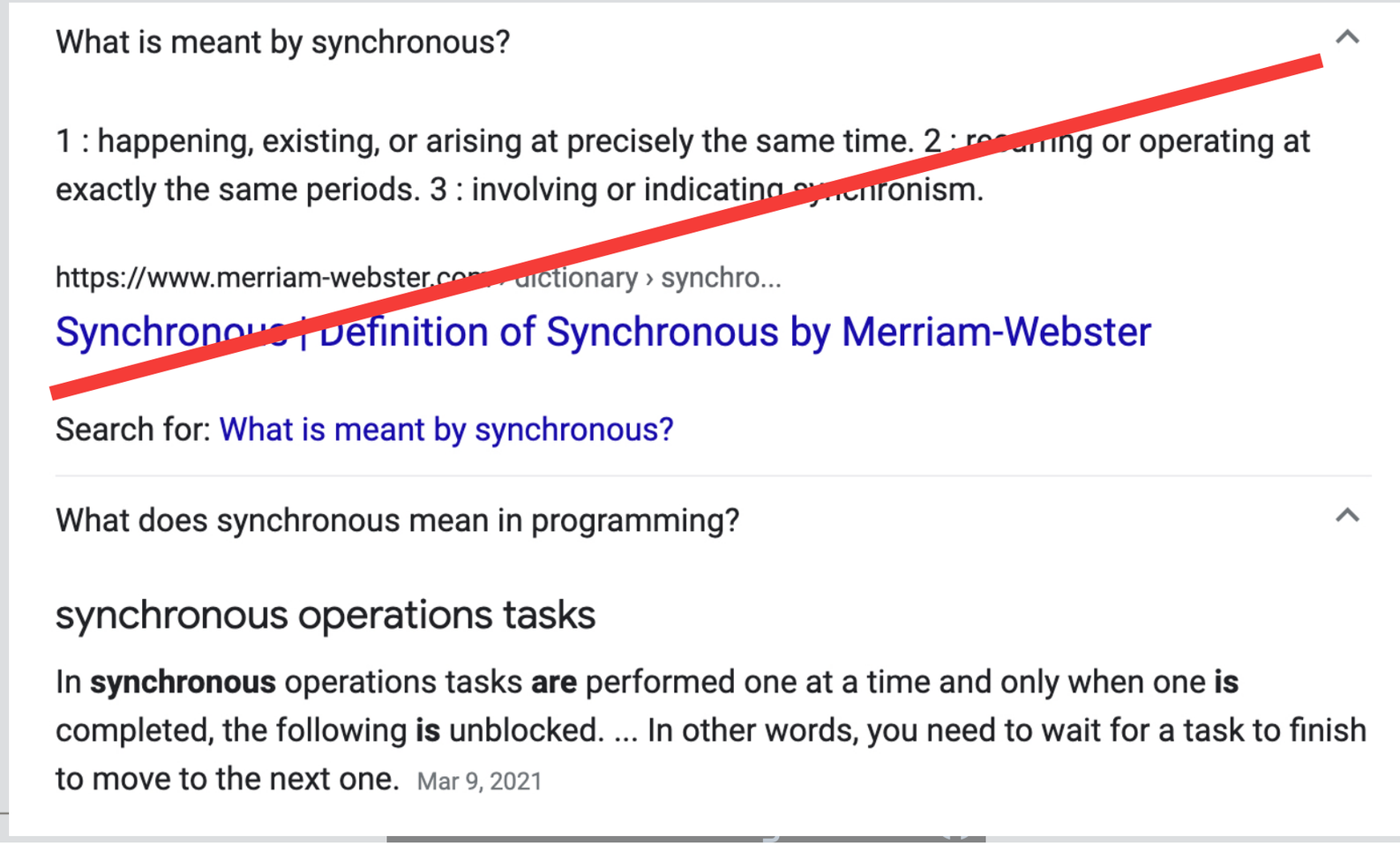
What is concurrency?

Definition: The potential for multiple processes to be IN PROGRESS at the same time

Definition: The composition of independently executing processes
- Rob Pike

Understanding Concurrency:

“synchronous” confusion.....

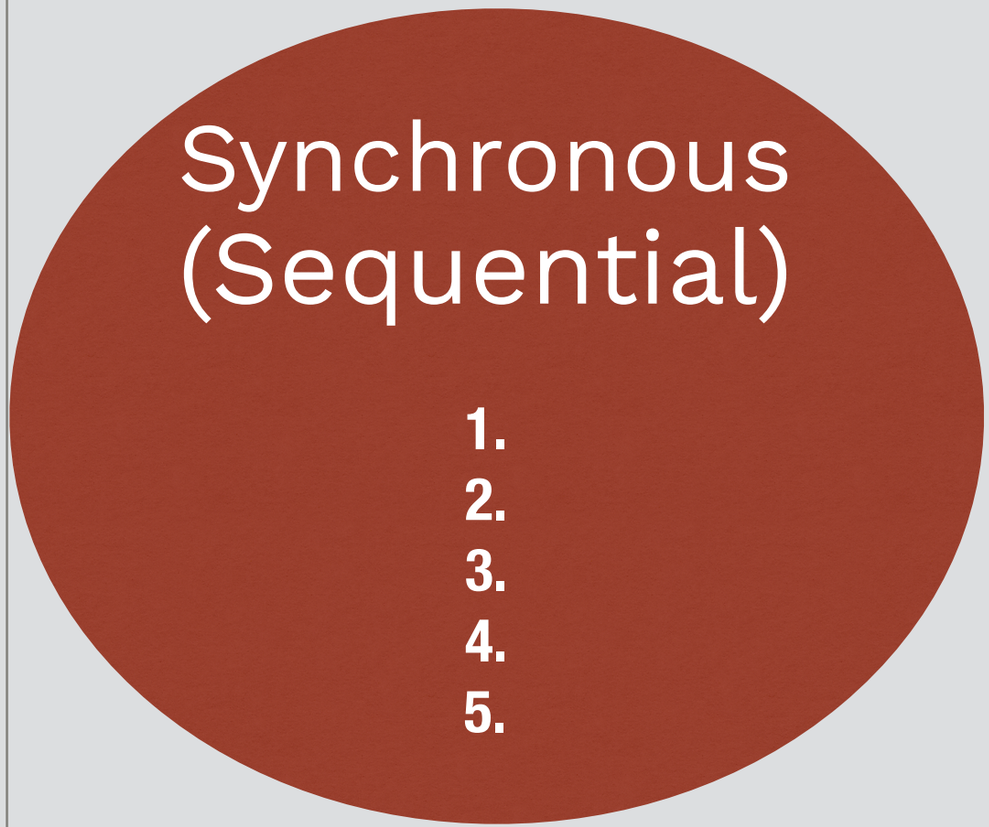


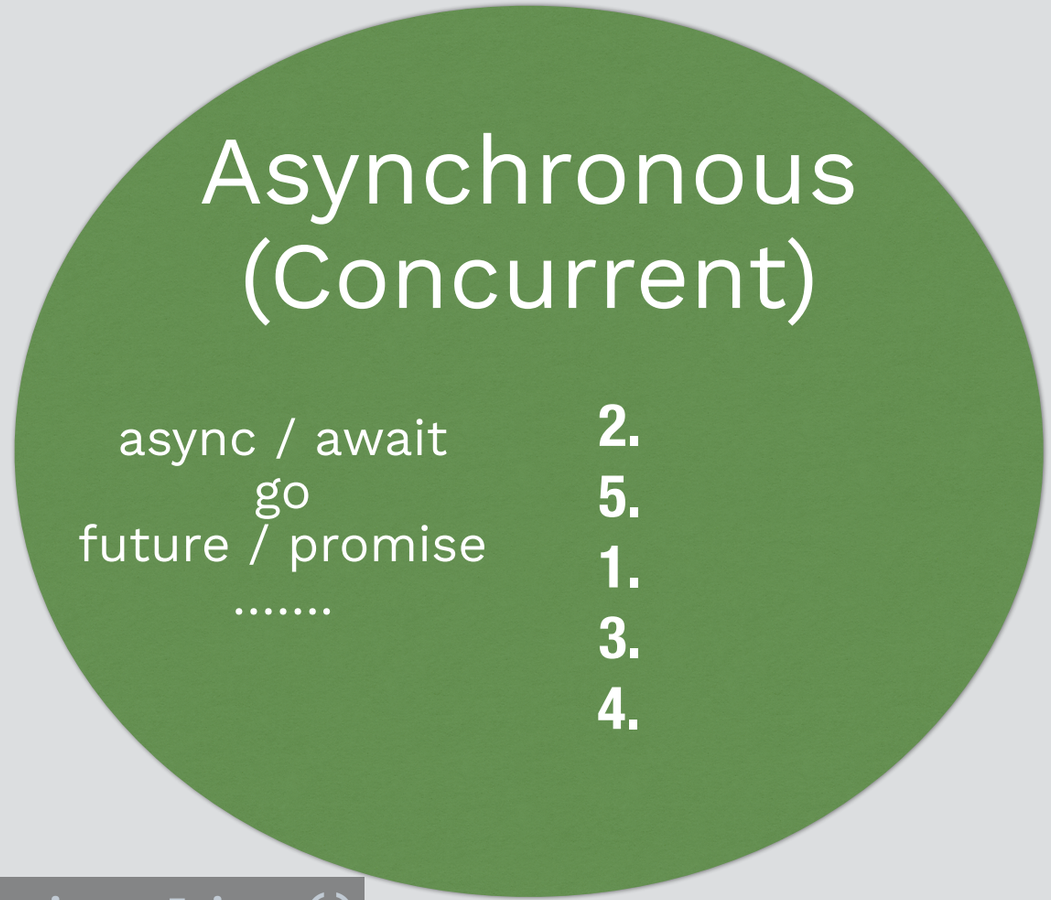
Understanding Concurrency:

From Wikipedia, the free encyclopedia

Asynchrony, in computer programming, refers to the occurrence of events independent of the main program flow and ways to deal with such events. These may be "outside" events such as the arrival of signals, or actions instigated by a program that take place concurrently with program execution, without the program *blocking* to wait for results.[1]

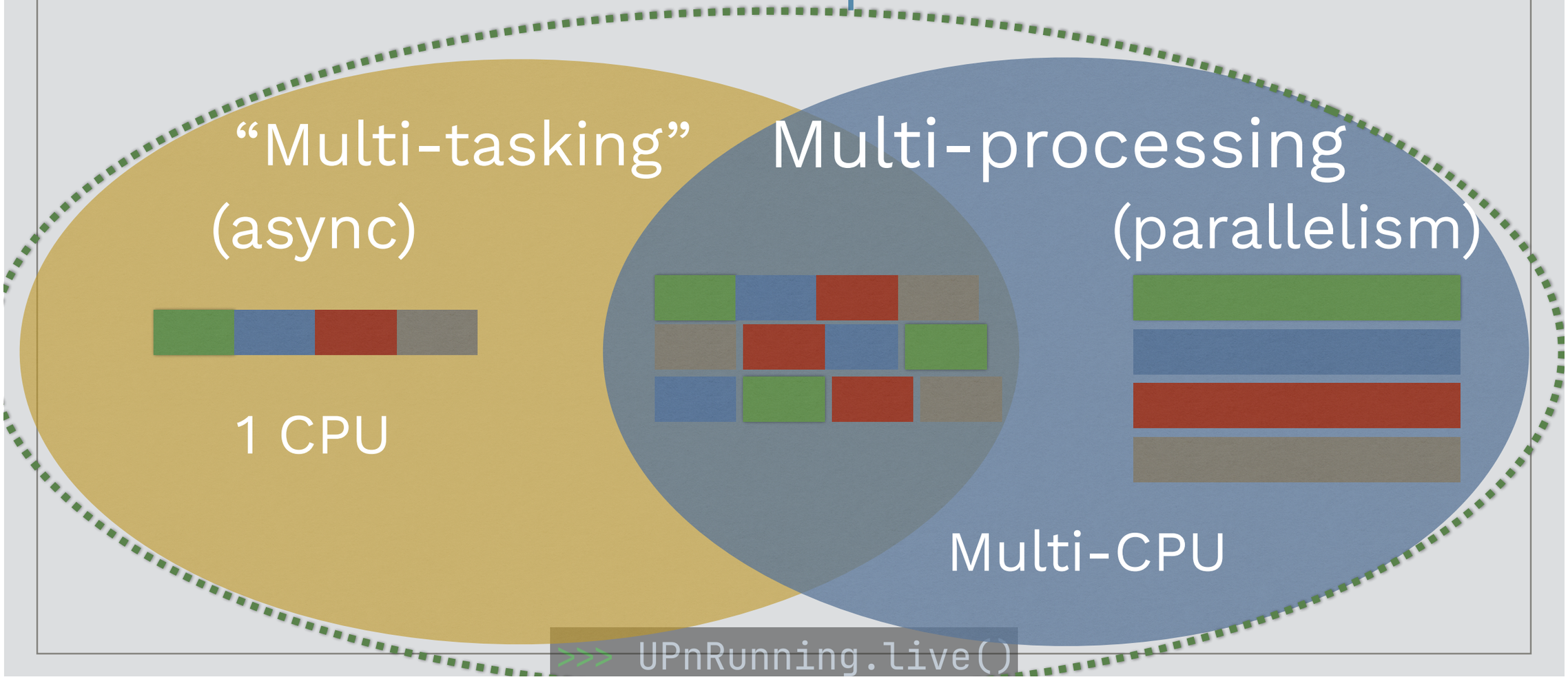
Understanding Concurrency:





>>> UPnRunning.live()

Understanding Concurrency:
It's more than parallelism




```
>>> UPnRunning.live()
```

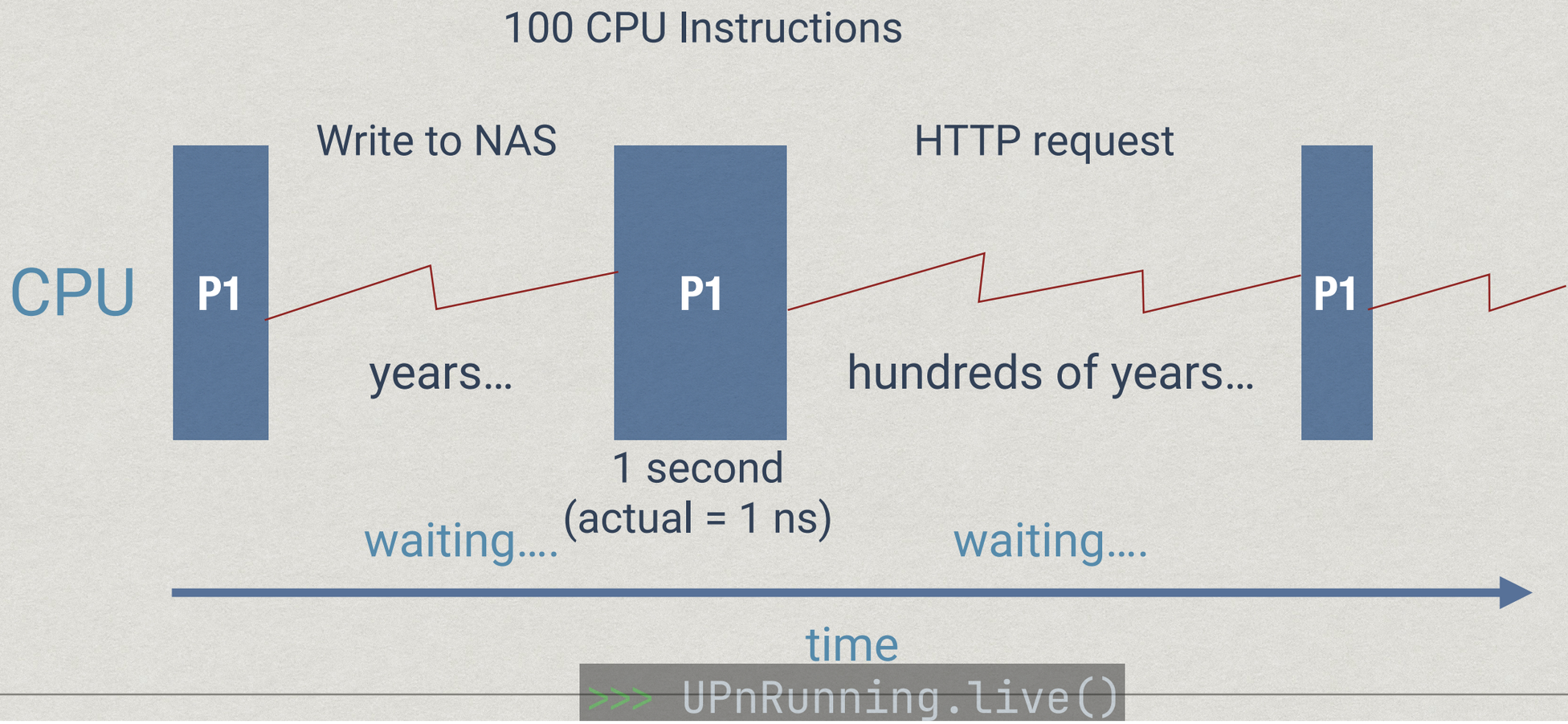
Understanding Concurrency: Trends, Benefits and Challenges

Trend #1

Increasing Latency w Increased Use of Cloud Apps and APIs

- OneDrive, iCloud Drive, Google Drive, Amazon Cloud Drive, Dropbox, ...
- Microsoft 365
- Adobe Creative Cloud
- Amazon Web Services
- Developer use of APIs

Latency



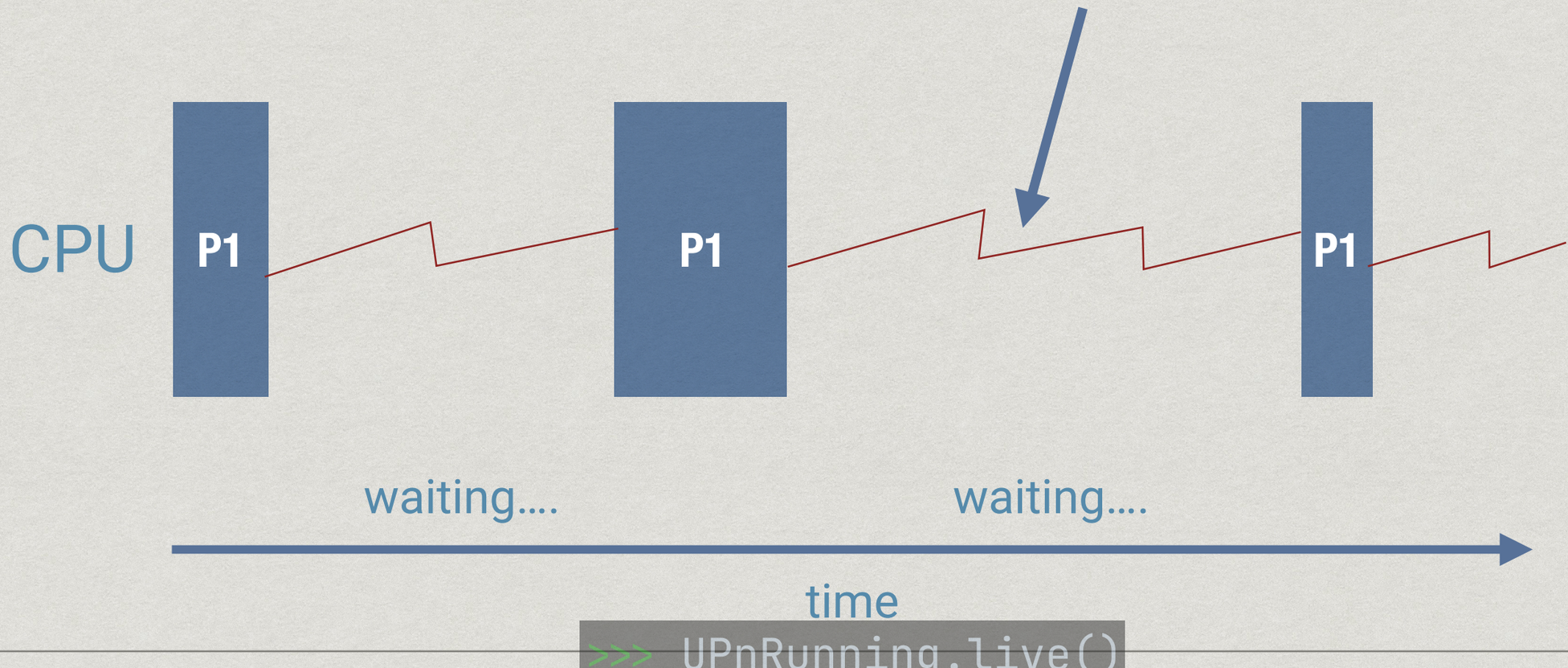

```
>>> UPnRunning.live()
```

Latency

Does this mean my computer is running slower?

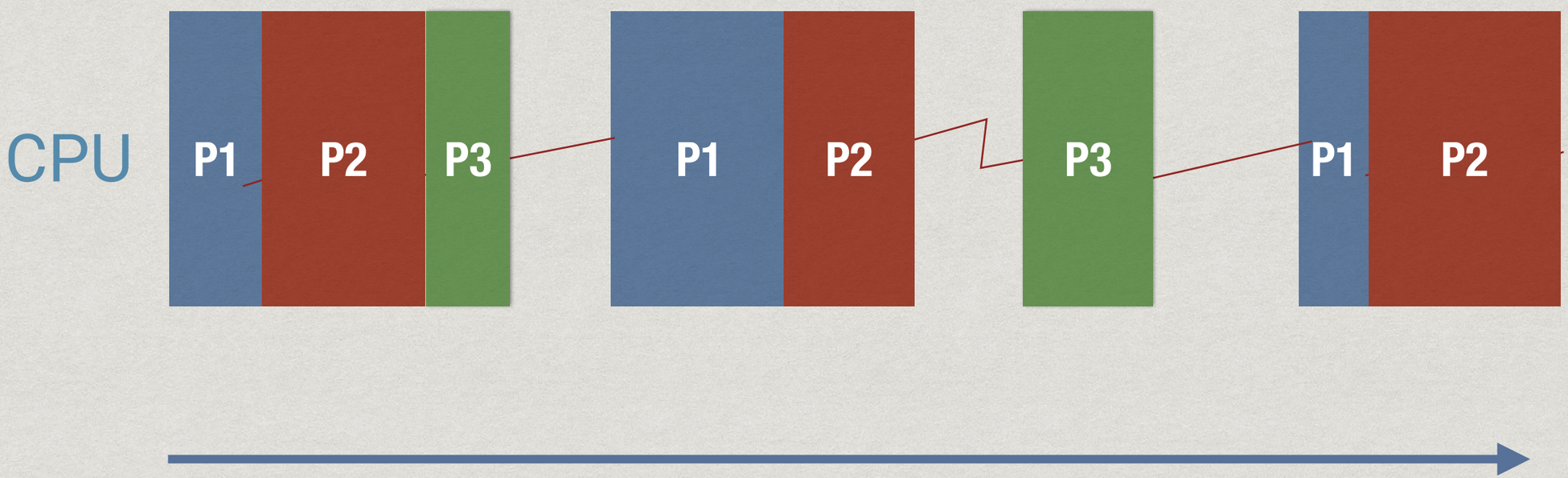
Maybe, but it could just mean you're not doing all the work.

Either way, it's an opportunity to get more done in the same amount of time, i.e., improve performance




```
>>> UPnRunning.live()
```

Latency ->
opportunity for concurrency




```
>>> UPnRunning.live()
```

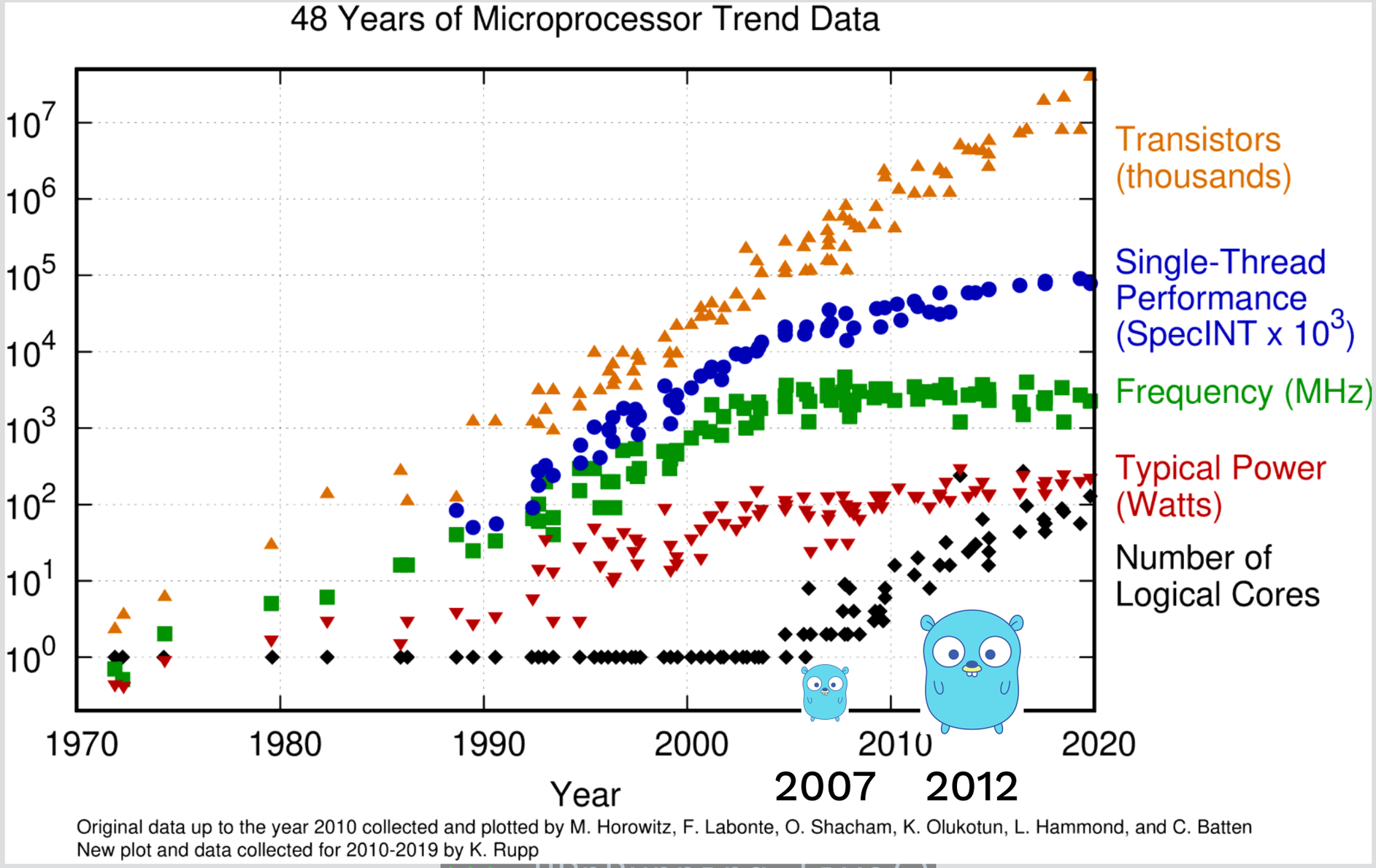
Understanding Concurrency: Trends, Benefits and Challenges

Trend #2

Multiple CPU Cores

In the last 10 years...

- Single-thread performance
- Clock frequency
- Number of cores




Overview
Displays
Storage
Support
Service
macOS Catalina
Version 10.15.7
MacBook Pro (Retina, Mid 2012)
Processor 2.6 GHz Quad-Core Intel Core i7
Memory 16 GB 1600 MHz DDR3
Graphics NVIDIA GeForc
Intel HD Graph
Serial Number C02KF0K
System Report...
So
™ and © 1983-2021 Apple Inc. All Rights Reserved. Lice
Overview
Displays
Storage
Support
Service
macOS Big Sur
Version 11.4
MacBook Pro (16-inch, 2019)
Processor 2.4 GHz 8-Core Intel Core i9
Memory 64 GB 2667 MHz DDR4
Startup Disk Macintosh HD
Graphics AMD Radeon Pro 5600M 8 GB
Serial Number C02F23R6MD6T
System Report...
Software Update...
™ and © 1983-2021 Apple Inc. All Rights Reserved. License and Warranty

Understanding Concurrency: Trends, Benefits and Challenges

Benefits vs Challenges

- Take advantage of latency in I/O-bound processes to improve performance on even a single processor
- Take advantage of multiple cores with CPU-bound processes for the fastest possible performance of your code on any hardware
    - BUT.....
- Not all code can be run concurrently
- Some additional complication to write concurrent code
- Beware of intermittent bugs due to race conditions and other unexpected behavior

```
>>> UPnRunning.live()
```

Setting Up Your Custom Development Environment

Installing Go - GOPATH, GOROOT

- Accept defaults if possible

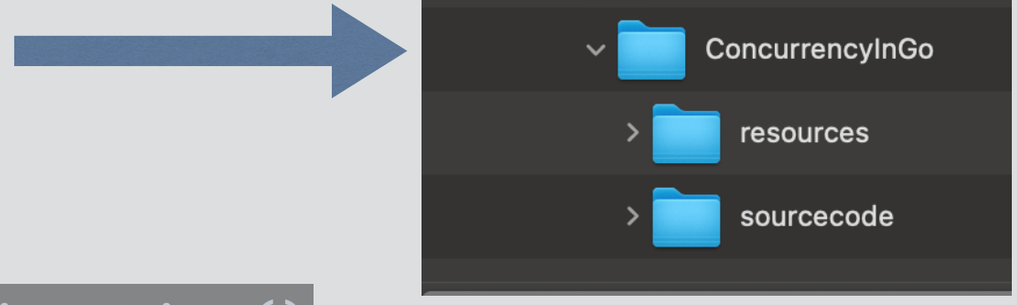
golang.org

Setting Up Your Custom Development Environment

Determine your folder / directory structure

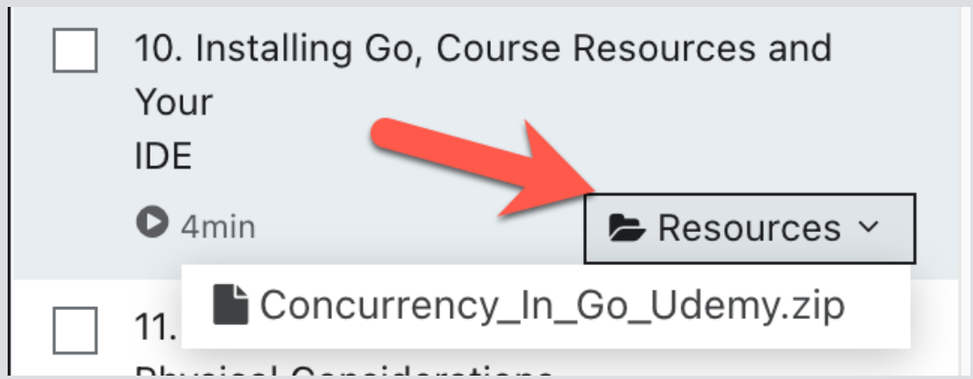
NOW!

- Course content can be installed in src directory as shown



Setting Up Your Custom Development Environment

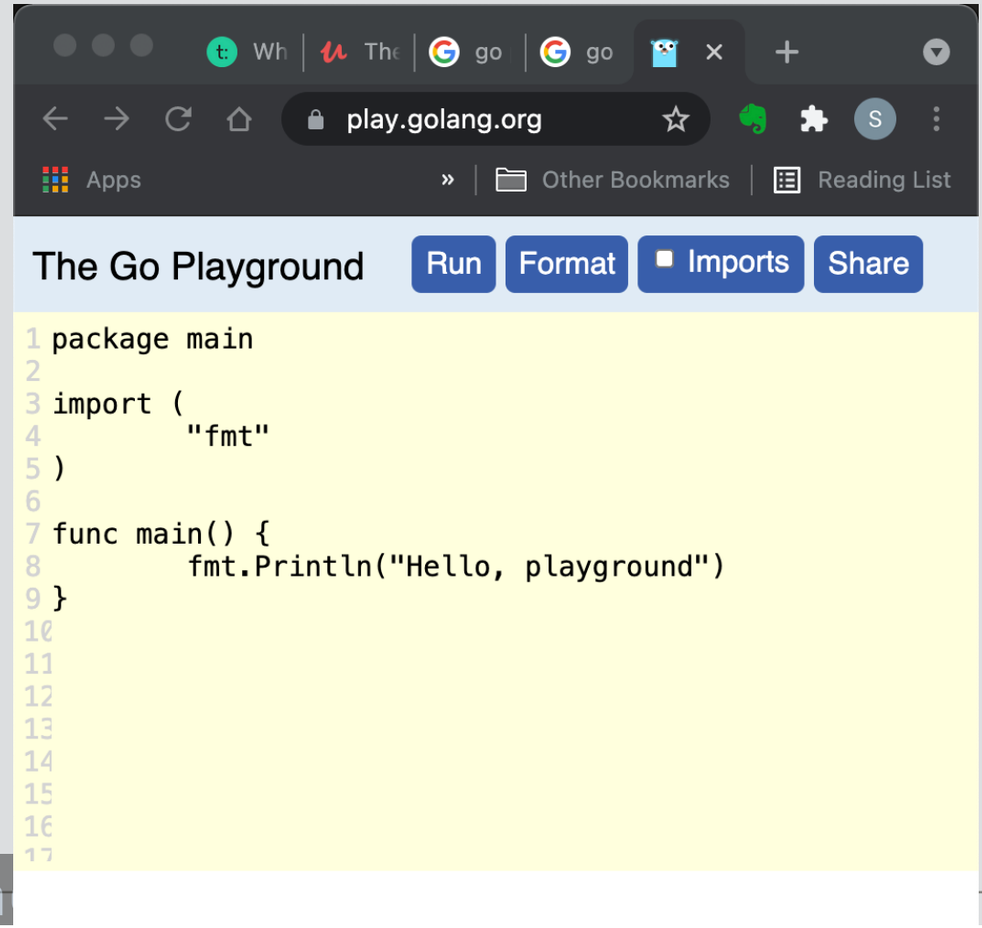
Downloading and Installing Course Resources:



Setting Up Your Custom Development Environment

The Go Playground

play.golang.org



>>> UPnRunnin

Setting Up Your Custom Development Environment

Installing your IDE:

(I will be using GoLand)

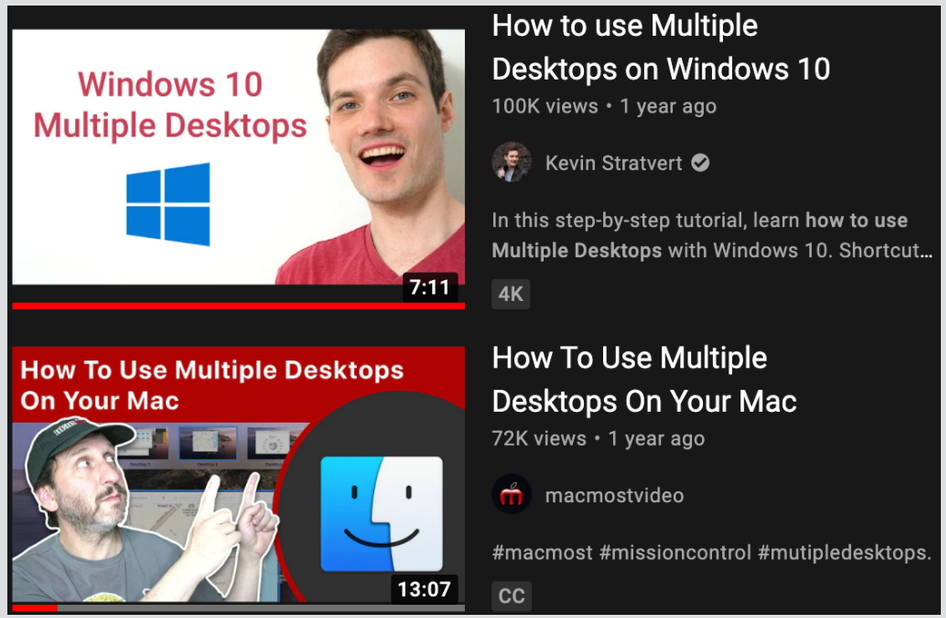
- GoLand 30-day FREE trial:
  - jetbrains.com/go/download
- Other popular IDEs / editors (FREE)
  - VS Code
  - Atom
  - Sublime (free trial)

Setting Up Your Custom Development Environment

Use 2 Screens

(even if you don't HAVE 2 screens)

- Split screen
- Multiple desktops
- Extra tablet
- Multiple monitors



Setting Up Your Custom Development Environment

My setup...

- Course
- YouTube
- Google
- Finder

- IDE
- IDE ref
- Lang ref
- Google
- Finder

Creating Our First Goroutine

- *“Inexpensive” concurrency thread built into the Go language*
- *Abstracts away memory management and other complexities*
- *Thousands of goroutines can run simultaneously with minimal overhead*

Creating Our First Goroutine

- *“Just add ‘go’ in front of any function call and you’re all set!”*
- `go doStuff()`
- Really?

Understanding Blocking Code

- “Blocking code does not allow its process to continue until blocking call finishes execution”

- `doSomething()`
- `doSomethingElse()`

Understanding Blocking Code

- “Blocking code does not allow its process to continue until blocking call finishes execution”
- “Blocking code allows execution to continue during waiting periods.”

```
>>> UPnRunning.live()
```

Understanding Blocking Code

- “Blocking" code is like “normal” code in that it must complete before the process can continue.
- What makes it unique in context of concurrency is that “completion” generally depends on status of other goroutines.
- Some goroutines can continue while others are blocked.

Understanding Blocking Code

In concurrency, blocking code is generally used to:

- Force func main() to wait for other goroutines to complete - waitgroups
- Synchronize goroutines at very specific points - for example, to exchange variables - channels
- Prevent two goroutines from accessing a shared variable at the same time – mutexes, condition variables, atomic variables

Using Waitgroups

Waitgroup syntax overview

- `var wg = sync.Waitgroup{}`
- `wg.Add(<int>)`
- `wg.Done()`
- `wg.Wait() (BLOCKING)`

Using Channels

- Channels are used to pass data between / synchronize goroutines, including func main
- Go philosophy:

“Don't communicate by sharing memory; share memory by communicating.”

Using Channels

Channel syntax overview

- `ch:= make(chan string)`
- `ch <- myData (BLOCKING)`
- `myVar <- ch  (BLOCKING)`
- `close(ch) (optional)`

Understanding IO-Bound vs CPU-Bound Processes

- I/O-bound processes have performance limitations due to I/O-related latency. Ex: http requests, network traffic
- CPU-bound processes have performance limitations due to CPU-related specs. Ex: intense encryption / decryption calculations

```
>>> UPnRunning.live()
```

Understanding IO-Bound vs CPU-Bound Processes

Observations:

- Multiple cores don't help with sequential tasks
- Just ONE core can see significant speed improvements with concurrency in IO-bound code due to latency
- CPU-bound code sees significant improvements up to the number of CPU cores. Concurrency is no help with ONE core.

```
>>> UPnRunning.live()
```

Understanding IO-Bound vs CPU-Bound Processes

Concurrency Benefit

	Sequential	IO-Bound	CPU-Bound
Single Core	N/A	✓	N/A
Multi Core	NO HELP!	✓	✓

>>> UPnRunning.live()

What could go wrong?
Race Conditions & Deadlocks

“A race condition occurs when two or more (goroutines) can access shared data and they try to change it at the same time.”

- Stack Overflow

This is very similar to a **merge conflict** on Github. Two collaborating developers read the same line of code, make different changes, then BOTH try to write their changes back to the master code-base.

What could go wrong?
Race Conditions

Purchases	Inventory	Sales
	1000 →	1000
900 ←	900 ←	-100
+100 →	1000	

What could go wrong?
Race Conditions

Purchases	Inventory	Sales
1000 ←	1000	→ 1000
	900 ←	-100
+100 →	1100	

What could go wrong?
Race Conditions

If you are only using wait groups, channels, and other tools specifically designed for communication with concurrency, race conditions cannot occur.

Problems can arise when goroutines share variables or other memory locations.

What could go wrong?
Race Conditions

Mutex syntax overview

- `mutex = sync.Mutex{}`
- `mutex.Lock()`
- `mutex.Unlock()`

What could go wrong?
Race Conditions

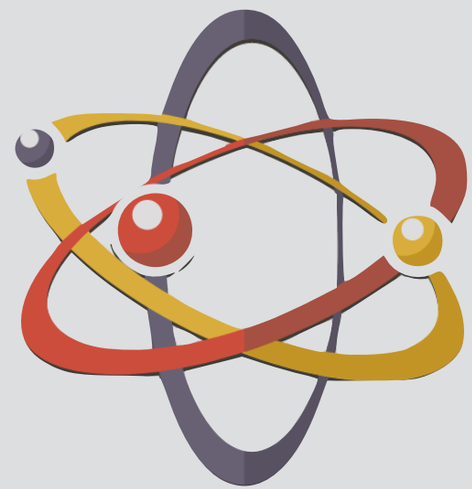
Condition variables syntax overview

- `var condition = sync.NewCond(&mutex)`
- `condition.Signal()`
- `condition.Broadcast()`
- `condition.Wait()  (BLOCKING)`

What could go wrong?
Race Conditions

Atomic variables syntax overview

- `Ex:  atomic.AddInt32(&inventory, 100)`
- Same as:
  - `mutex.Lock()`
  - `inventory += 100`
  - `mutex.Unlock()`



What processes are good candidates for concurrency?

Much of the code you write MUST be sequential since the output of one function becomes the input of the next.

Processes that are completely independent and share no data are ideal candidates for concurrency

Mastering concurrency “patterns” such as worker pools / pipelining will expand your opportunities for concurrency.

```
>>> UPnRunning.live()
```

Concurrency Patterns

Concurrency patterns such as worker pools provide an extra layer of control and reusability for concurrency in your code.

Ex: Worker pools can be used to limit the total CPU usage of a specific process - for example, a file backup utility that you'd like to run concurrently in the background.

Worker Pools

Jobs

To Call
Ambrosio Snaaijer
Talulla Byrd
Linus Baker
Erastos Thatcher
Feodosiya Bentsen
Lavinia Belcher
Zebulon Kiefer
Georgs Vemulakonda
Donncha Holgersen
Maret Parisi

Workers

Results

Completed Calls

```
>>> UPnRunning.live()
```

Worker Pools

Jobs

To Call
Ambrosio Snaaijer
Talulla Byrd
Linus Baker
Erastos Thatcher
Feodosiya Bentsen
Lavinia Belcher
Zebulon Kiefer
Georgs V
Donncha
Maret Parisi

Results

Completed Calls


```
>>> UPnRunning.live()
```

Worker Pools

Jobs

To Call
Feodosiya Bentsen
Lavinia Belcher
Zebulon Kiefer
Georgs V
Donncha
Maret Parisi

Results

Completed Calls
Ambrosio Snaaijer
Talulla Byrd
Linus Baker
Erastos Thatcher

```
>>> UPnRunning.live()
```

Worker Pools

Jobs (channel)

To Call

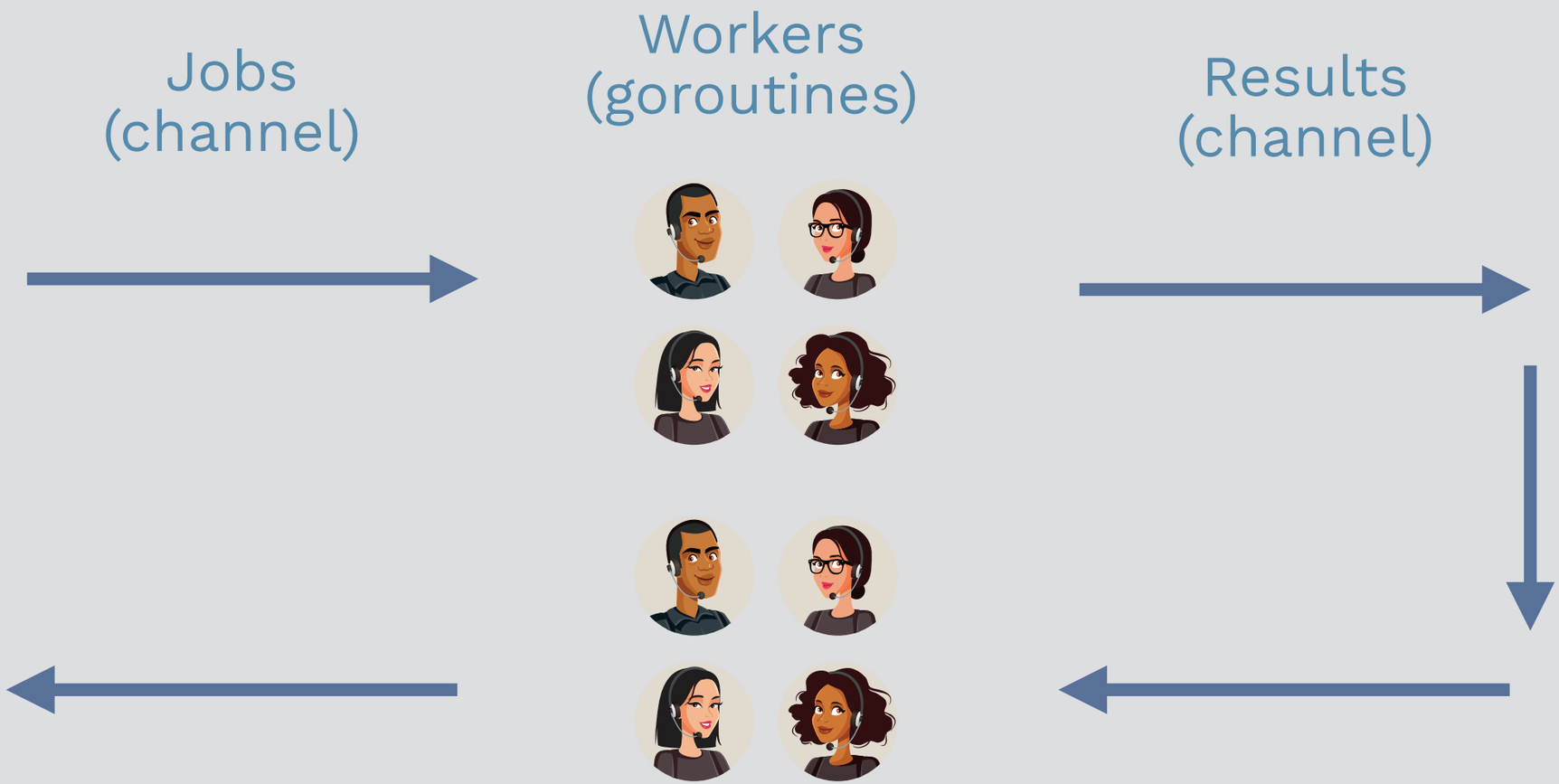
Workers (goroutines)

Results (channel)

Completed Calls
Talulla Byrd
Ambrosio Snaaijer
Linus Baker
Erastos Thatcher
Lavinia Belcher
Georgs Vemulakonda
Feodosiya Bentsen
Zebulon Kiefer
Donncha Holgersen
Maret Parisi

```
>>> UPnRunning.live()
```

Pipelines




```
>>> UPnRunning.live()
```

Major Takeaways From This Course

- Concurrency in Go is probably more straight-forward than in other languages you've explored. This is because concurrency was in the original design.

- Adding the "go" keyword in front of any function will create a (concurrent) goroutine, but it's up to you to ensure that func main() interacts with it as expected.

Major Takeaways From This Course

- Wait groups are a very simple way to ensure that func main() doesn't "forget about" goroutines, but it's a manual process and they have limited functionality.

- Channels are a very flexible and powerful way to manage, coordinate, and synchronize goroutines. They were designed for this purpose and do not create race conditions.

Major Takeaways From This Course

- Using channels is preferable to sharing variables, but if variables MUST be shared, race conditions can occur.

- Race conditions can be avoided using mutexes, atomic variables and/or other techniques.

Major Takeaways From This Course

- Recognizing IO-bound vs CPU-bound code isn't an absolute requirement for implementing concurrency in Go, but can help to understand the expected performance improvements.

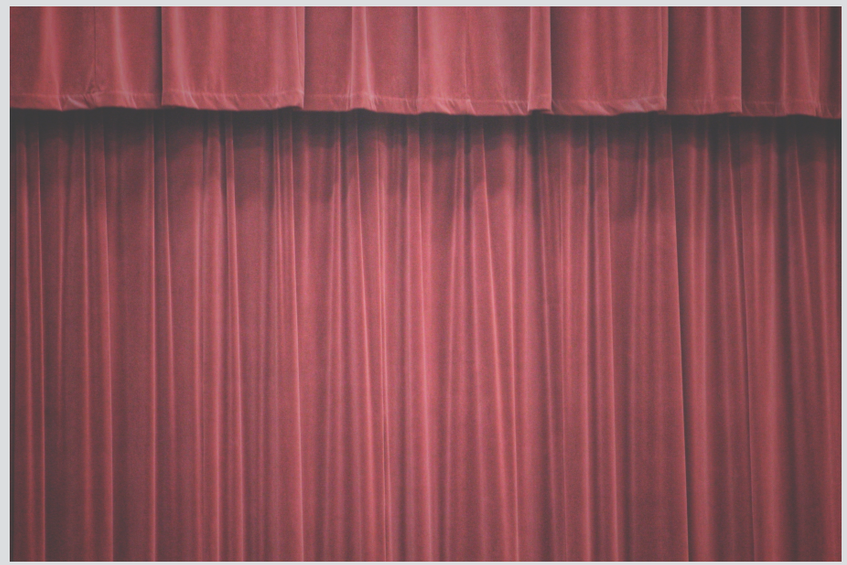
- Concurrency patterns such as worker pools can provide structure and more control over concurrency impacts.

Closing Credits

Thanks for joining me in this overview of concurrency in Go.

This course was the result of extensive experimentation, research, and incorporating learnings from dozens of excellent courses, blogs, YouTube videos, and many other resources.

I've attempted to list as many of them as I can here, and would encourage you to follow up with any or all of them as your time and interest allows.



```
>>> UPnRunning.live()
```

Research for Course

Name	Channel	Author	Link
Concurrency in Go	YouTube	Jake Wright	https://www.youtube.com/watch?v=LvgVSSpwND8
Go Concurrency Made Easy	YouTube	CodingTech	https://www.youtube.com/watch?v=DqHb5KBe7qI
Tour of Go	Go	golang.org	https://tour.golang.org/concurrency/1
Effective Go	Go	golang.org	https://golang.org/doc/effective_go#concurrency
Go Complete Developers Guide	Udemy	Stephen Grider	https://www.udemy.com/course/go-the-complete-develope
Learn to Code in Go	Udemy	Todd McLeod	https://www.udemy.com/course/learn-how-to-code/learn/le
Go By Example	Go	Mark McGranaghan	https://gobyexample.com/
Multithreading in Go	Udemy	James Cutajar	https://www.udemy.com/course/multithreading-in-go-lang/l
A Journey with Go	Medium	Vincent Blanchon	https://medium.com/a-journey-with-go/go-goroutine-os-thr
Microprocessor trend data	GitHub	Karl Rupp	https://github.com/karlrupp/microprocessor-trend-data
Demystifying async and await	YouTube	PyCharm	https://www.youtube.com/watch?v=F19R_M4Nay4&t=420
Concurrency in Python - latency	RealPython	Chris	https://realpython.com/python-concurrency/
How to Use Multiple Desktops on Mac	YouTube	MacMost	https://www.youtube.com/watch?v=H3hUk0zsxc4
Multiple desktops on Windows	YouTube	Microsoft	https://www.google.com/search?q=using+multiple+desktop
Pain-free concurrency in go	Medium	Lorenzo Peppoloni	https://medium.com/@l.peppoloni/pain-free-concurrency-in
Concurrency made easy	YouTube	Dave Cheney	https://www.youtube.com/watch?v=yKQOunhhf4A
Conurrency is not parallelism	YouTube	Rob Pike	https://www.youtube.com/watch?v=oV9rvDllKEg&t=1636s
Rethinking classical concurrency patterns	Youtube	Bryan C. Mills	https://www.youtube.com/watch?v=5zXAHh5tJqQ
Advanced Go concurrency patterns	Go Blog	Andrew Gerrand	https://blog.golang.org/io2013-talk-concurrency
Intro to Communicating sequential processes	Youtube	Kuz Le	https://www.youtube.com/watch?v=G9ePu0Nh2BQ
Restful API basics in Go	Youtube	AppliedGo	https://www.youtube.com/watch?v=iVXaPD_Jbu0&t=358s

Image Credits

TODOS

Photo by True Agency on <a href="https://unsplash.com/s/photos/developer?utm_source=unsplash&utm_m

Photo by <a href="https://unsplash.com/@glenncarstenspeters?utm_source=unsplash&utm_medium=referral&utm_content=creditCopy

Photo by Tingey Injury Law Firm on Unsplash

Photo by <a href="https://unsplash.com/@bill_oxford?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Bill

Photo by <a href="https://unsplash.com/@a_kehmeier?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Au

Photo by <a href="https://unsplash.com/@kaip?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Kai Pilger

Photo by Marcel Strauß on Unsplash

Photo by <a href="https://unsplash.com/@zanardi?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Everyc

Photo by <a href="https://unsplash.com/@malcoo?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Tomáš

Image by rawpixel.com

Photo by <a href="https://unsplash.com/@pietromattia?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Pi

Photo by <a href="https://unsplash.com/@rocinante_11?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">M

Photo by imgix

Photo by <a href="https://unsplash.com/@mrthetrain?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Josh

Photo by <a href="https://unsplash.com/@miracleday?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Ele

Photo by <a href="https://unsplash.com/@zoncoll?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Valeria

Image by rawpixel.com

Design vector created by macrovector - www.freepik.com

Photo by <a href="https://unsplash.com/@cdr6934?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Chris

Photo by <a href="https://unsplash.com/@nguyendhn?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Ng

Photo by <a href="https://unsplash.com/@kadh?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText">Kira auf d