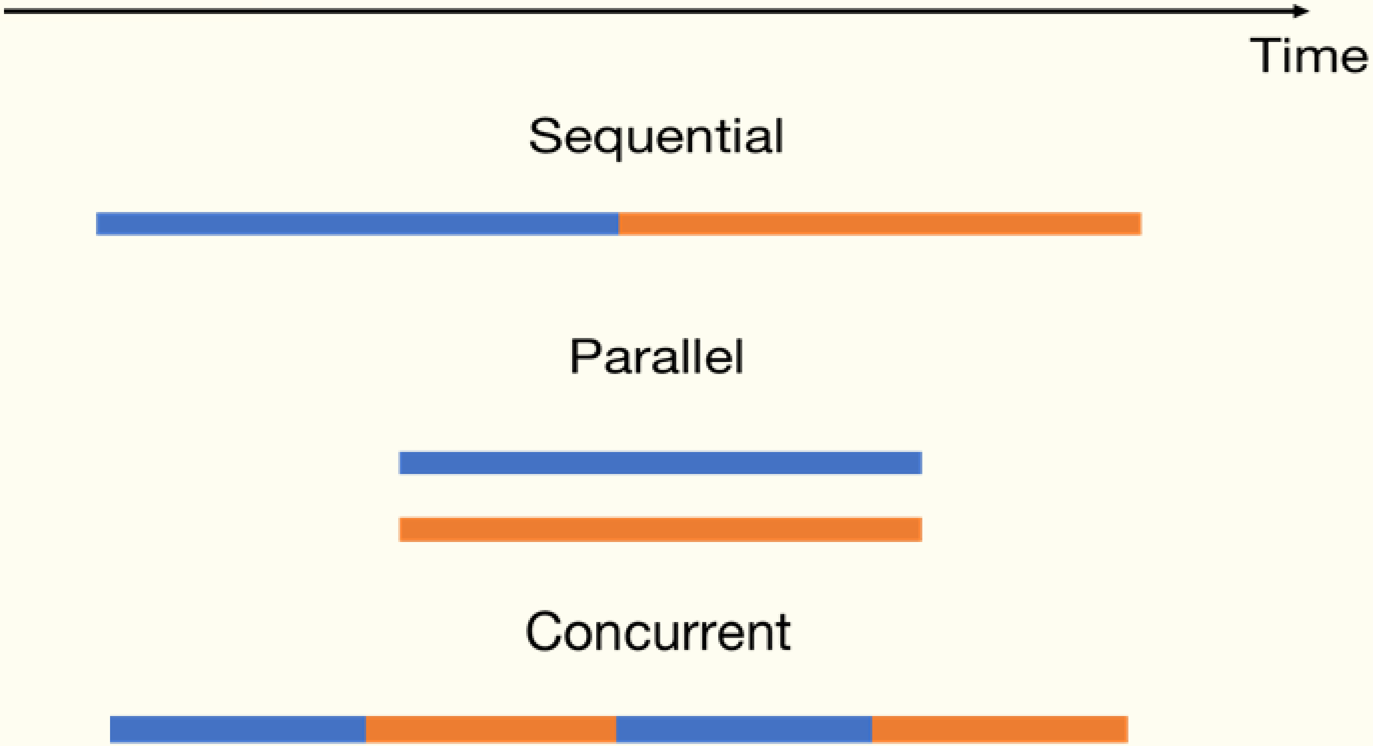
Concurrency and RPCs in Go

CS 240: Computing Systems and Concurrency
Lab 2

Jihao Xin
Jihao.xin@kaust.edu.sa

Concurrency

Sequential, Concurrent, Parallel



Time

Sequential

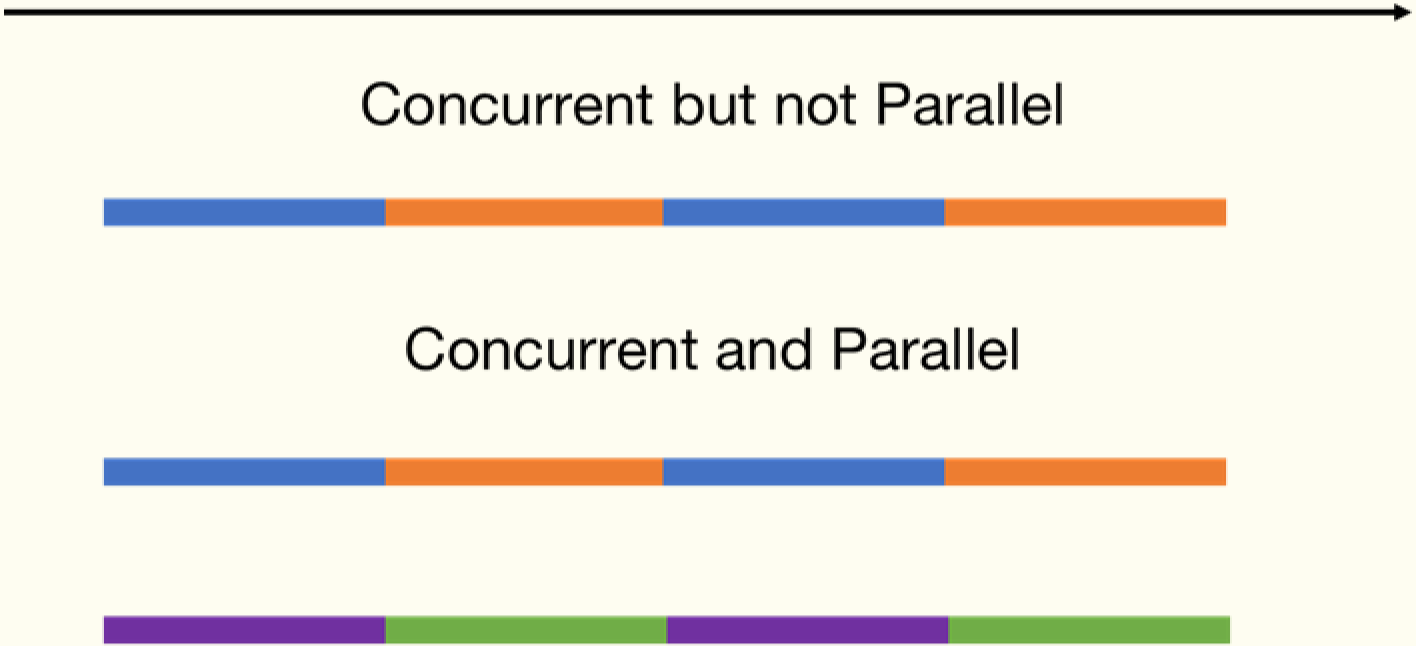
Parallel

Concurrent

“Concurrency is about dealing with lots of things at once.
Parallelism is about doing lots of things at once.”

- Rob Pike

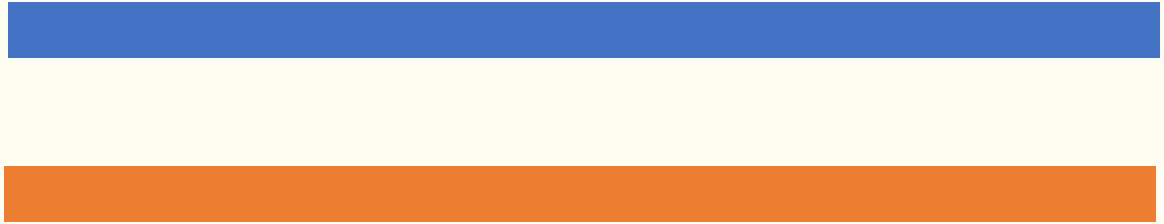
Concurrent ≠ Parallel



Concurrent but not Parallel

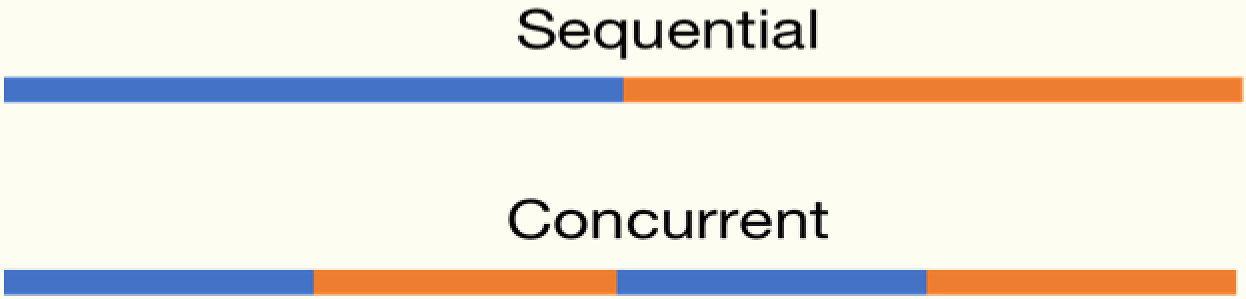
Concurrent and Parallel

Parallel → Concurrent



Parallel is more strict

Why Concurrent?



May end at same time

Why Concurrent?

- Running of multiple applications

  “Pretend” to be parallel to user

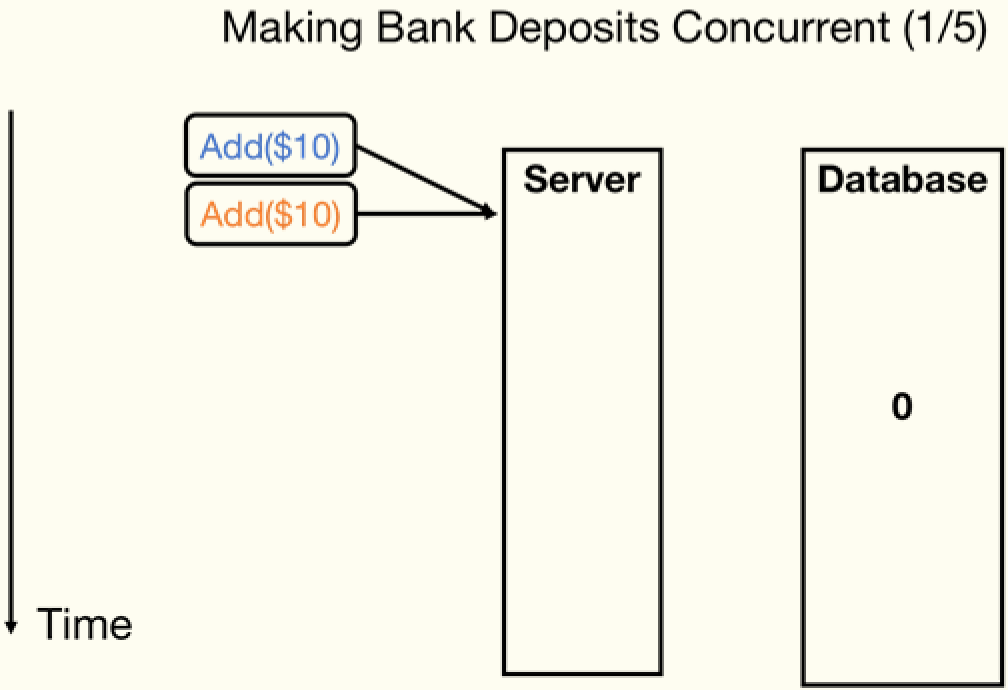
- Better utilization & performance

  With OS support, when A use CPU, B can use NIC

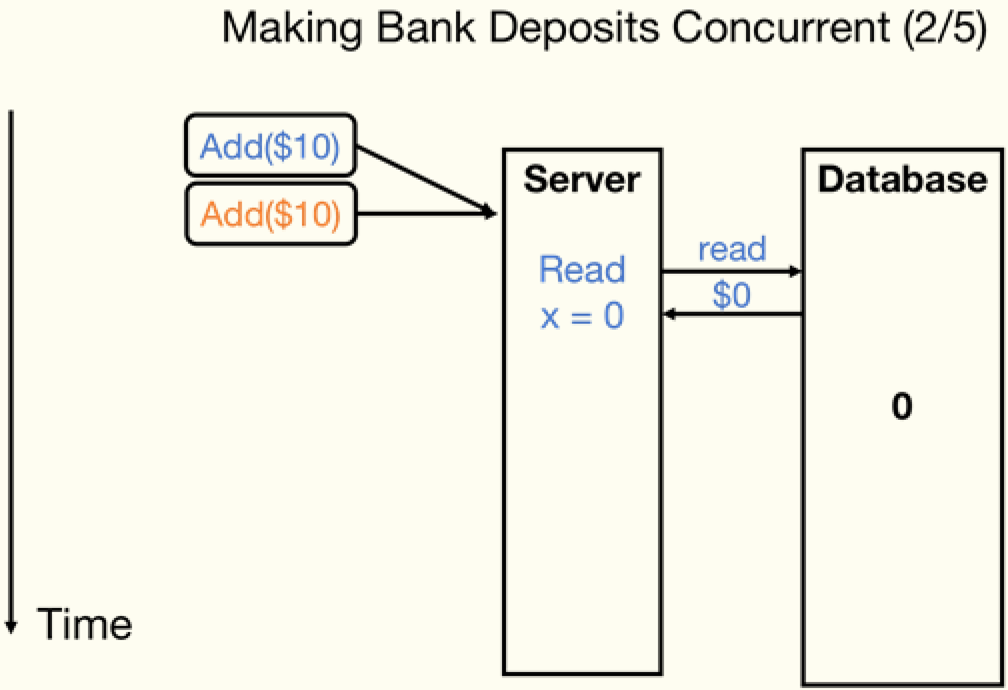
- Better average response time

  If A waiting a TCP package, B does not need to wait

Concurrency Issue

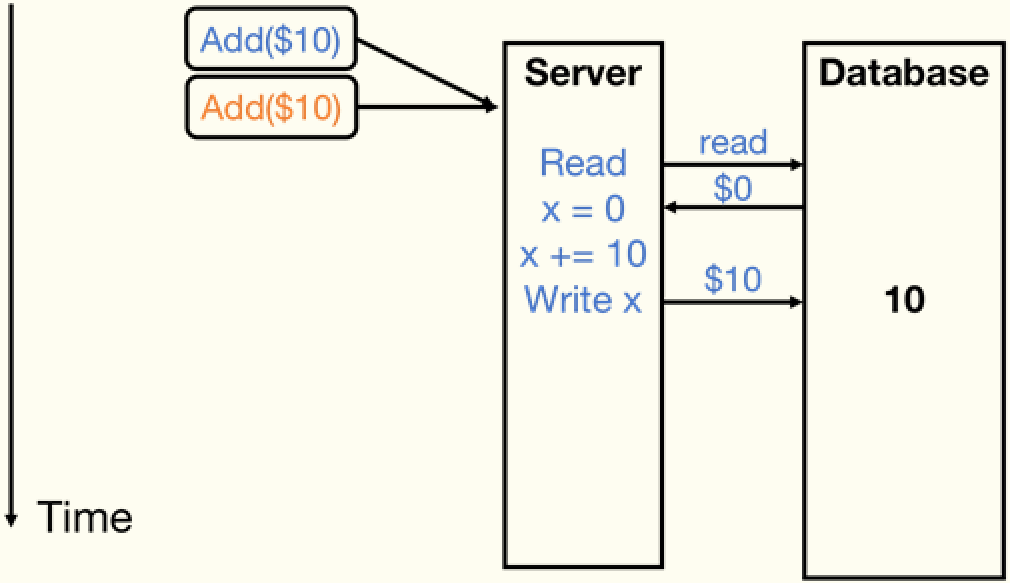


Concurrency Issue



Concurrency Issue

Making Bank Deposits Concurrent (3/5)



Add($10)

Add($10)

Server

Read
x = 0
x += 10
Write x

read

$0

$10

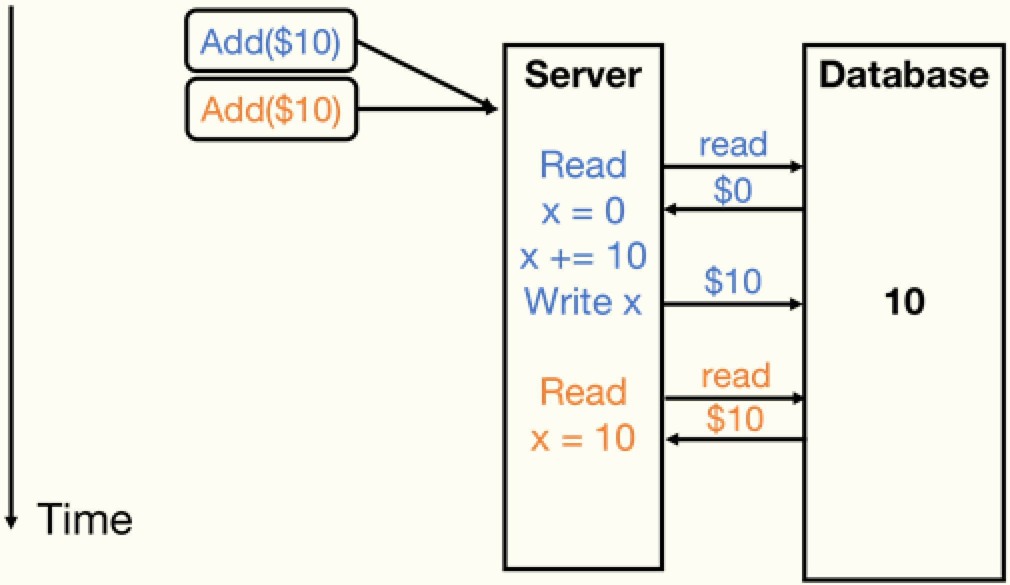
Database

10

Time

Concurrency Issue

Making Bank Deposits Concurrent (4/5)



Add($10)

Add($10)

Server

Read

x = 0

x += 10

Write x

Read

x = 10

Database

10

read

$0

$10

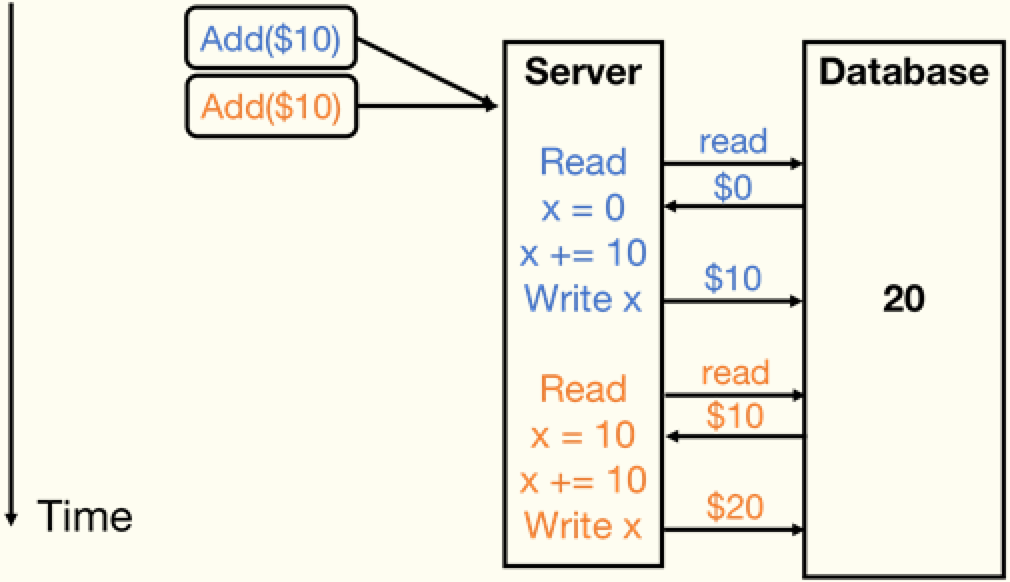
read

$10

Time

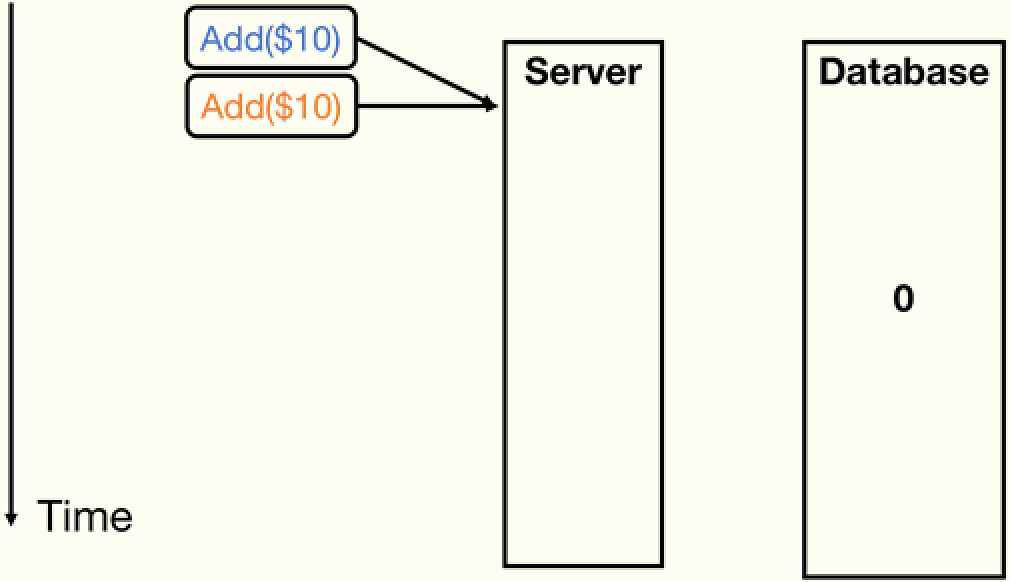
Concurrency Issue

Making Bank Deposits Concurrent (5/5)



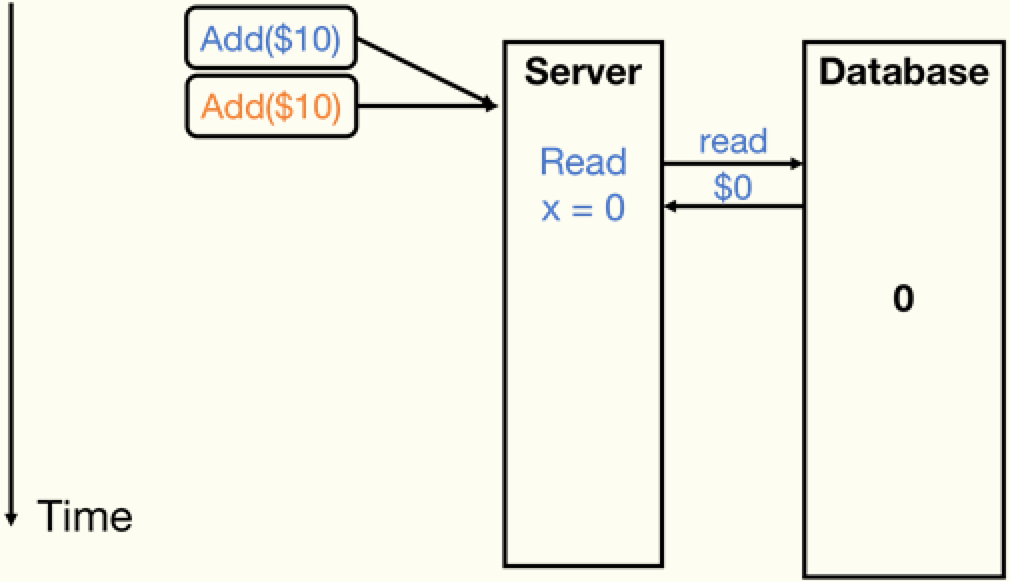
Concurrency Issue

Concurrent Bank Deposits! Yay? (1/5)



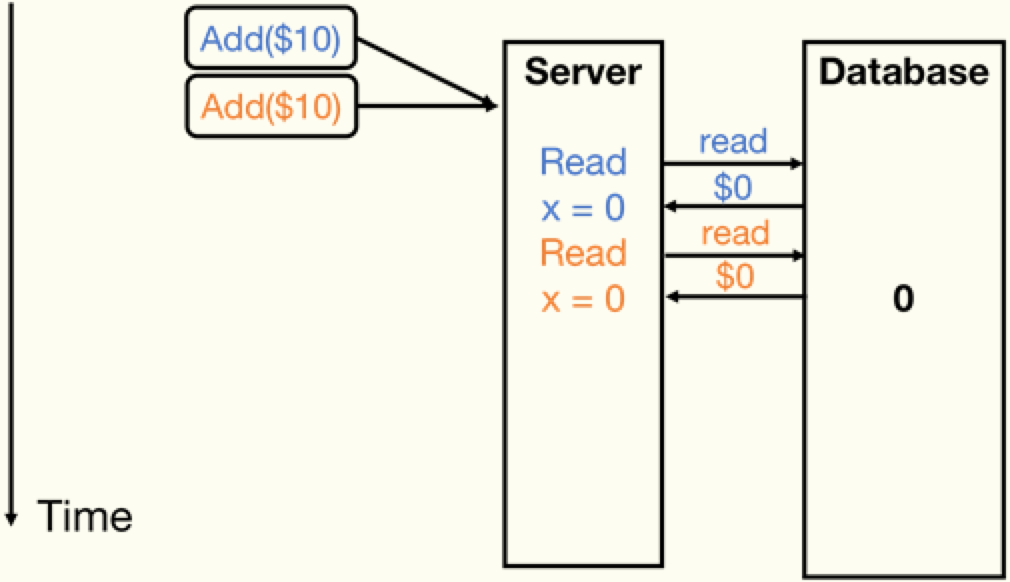
Concurrency Issue

Concurrent Bank Deposits! Yay? (2/5)



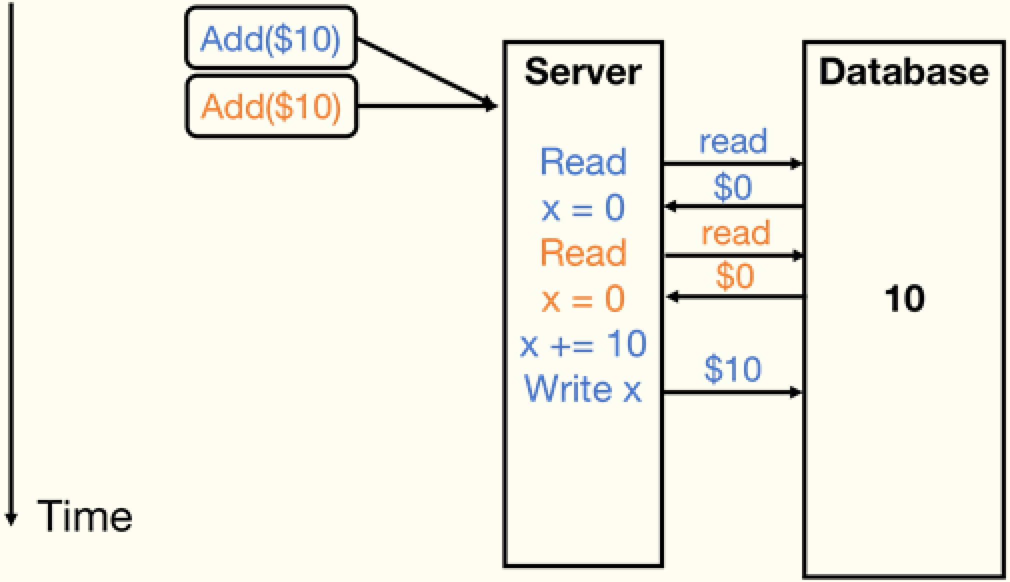
Concurrency Issue

Concurrent Bank Deposits! Yay? (3/5)



Concurrency Issue

Concurrent Bank Deposits! Yay? (4/5)



Add($10)

Add($10)

Server

Read
x = 0
Read
x = 0
x += 10
Write x

read

$0

read

$0

$10

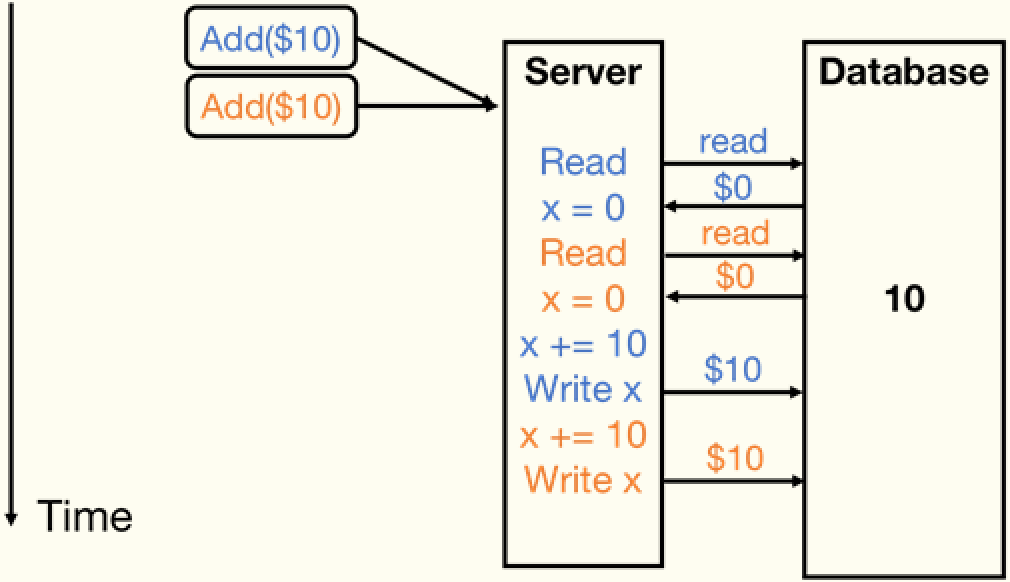
Database

10

Time

Concurrency Issue

Concurrent Bank Deposits! Yay? (5/5)



Concurrency Issue

Threads are:

Mutually dependent

Execute simultaneously

Access shared resource

→

- Deadlock
- Race condition
- Starvation

Synchronization

- Locks
  Limit access using shared memory

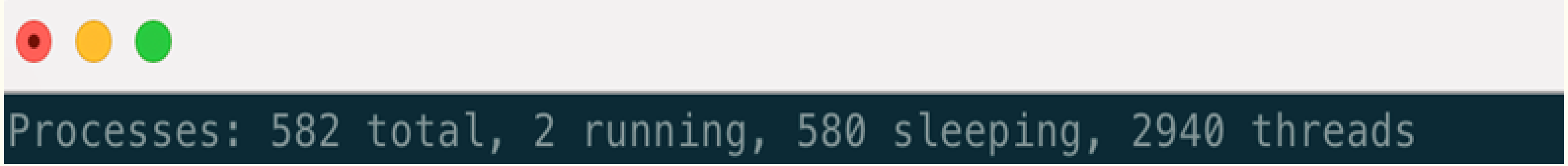
- Channels
  Pass information using a queue

A nice concurrency visualization:
https://divan.dev/posts/go_concurrency_visualize/

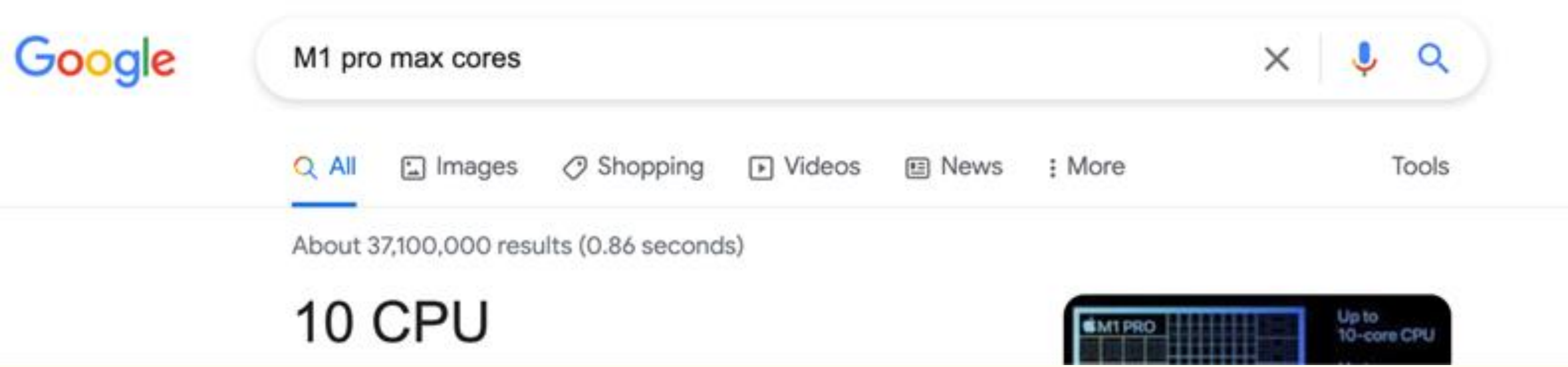
Threads

- What is a Thread?
- How many threads can we create?
- How many threads can run in parallel?

Multi-cores
Hyper-Threading
Pipeline Execution
Task-Level Parallelism
...



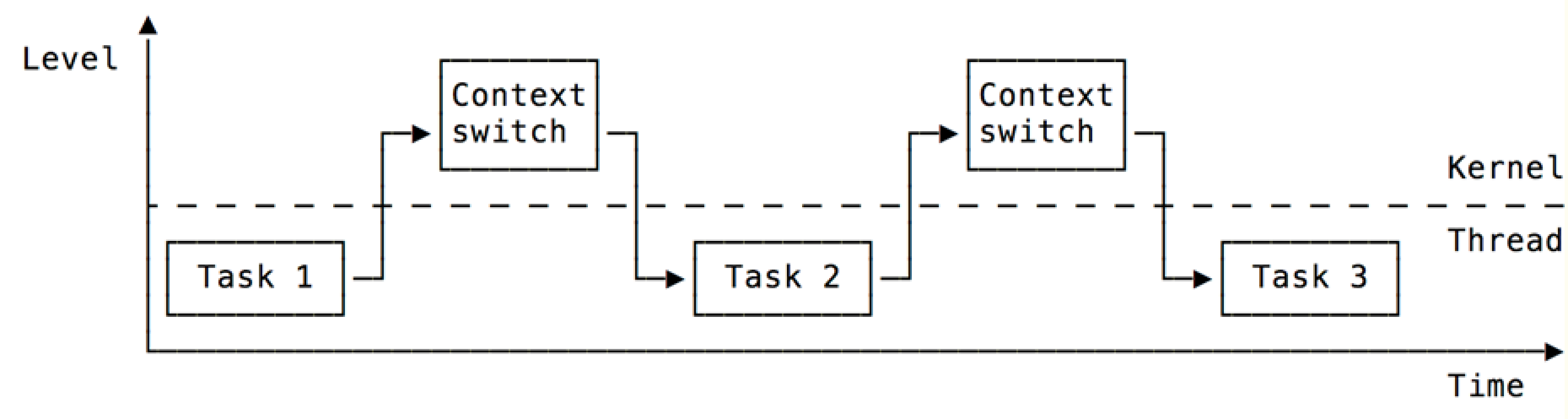
```
Processes: 582 total, 2 running, 580 sleeping, 2940 threads
```



About 37,100,000 results (0.86 seconds)

10 CPU

Thread Switching

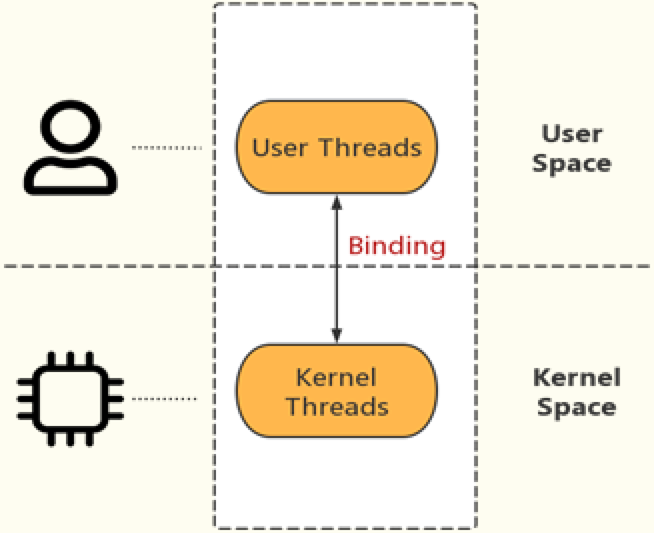


Large overhead!

How do we improve?

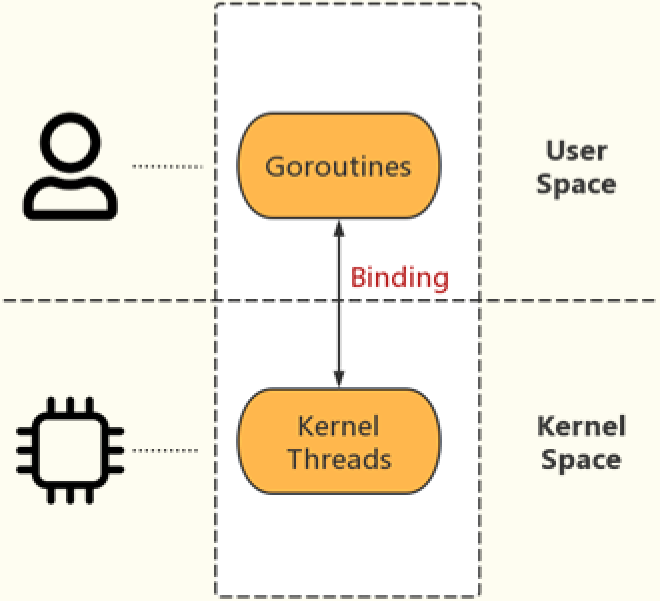
Thread Switching

- Can we switch “thread” in user space?



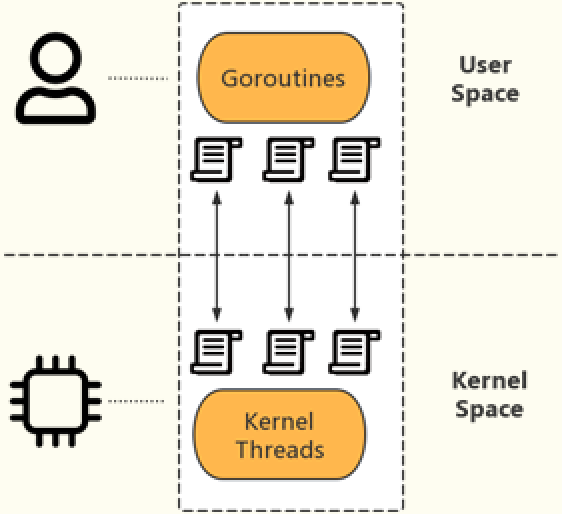
Goroutines

- In Go, let’s call it “routines”

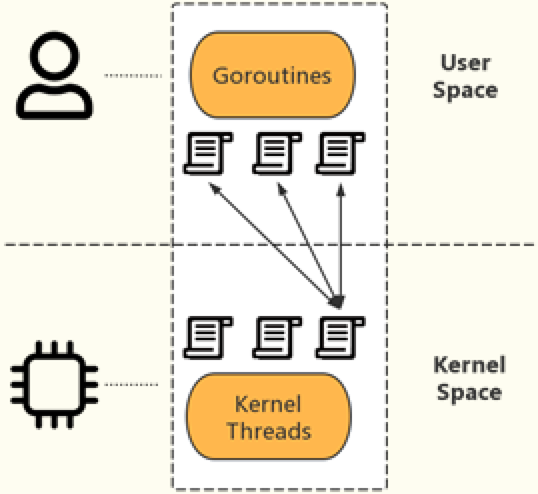


Goroutines

- How does the Binding work?



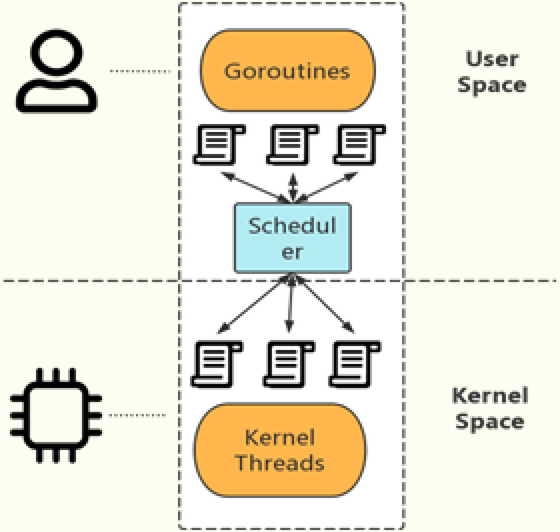
1:1



M:1

Goroutines

- Go does the **“Thread Switching”** by user-space scheduler.
- $GOMAXPROCS - By default your core numbers.



M:N

Goroutines

- How to launch a Go routine？
  Just Go!

```
func say(s string) {
        for i := 0; i < 5; i++ {
                time.Sleep(100 *
time.Millisecond)
                fmt.Println(s)
        }
}

func main() {
        go say("world")
        say("hello")
}
```

Go Channels

- The way routines communicate
- “A typed conduit through which can send and receive values”

```
func sum(s []int, c chan int) {
        sum := 0
        for _, v := range s {
                sum += v
        }
        c <- sum // send sum to c
}
```

```
func main() {
        s := []int{7, 2, 8, -9, 4,
0}

        c := make(chan int)
        go sum(s[:len(s)/2], c)
        go sum(s[len(s)/2:], c)
        x, y := <-c, <-c //
receive from c

        fmt.Println(x, y, x+y)
}
```

RPC

Practice

Recall

RPC (Remote Procedure Call)

A client will execute some function on a remote server

- Client makes local requests with parameters
- RPC library encodes the request,& parameters
- Send to server
- Server decodes the request & parameters
- Procedure is executed on the server
- Server sends reply back to the client

gRPC

- Go *net/rpc* by default uses *gob* to encode
- Client and server may use different encoding scheme
- Communication needs a "*common language*"
- **Protobuf -** data struct serialization (the common language translator)
- **gRPC: Protobuf + RPC**