Going concurrent

in programming contests

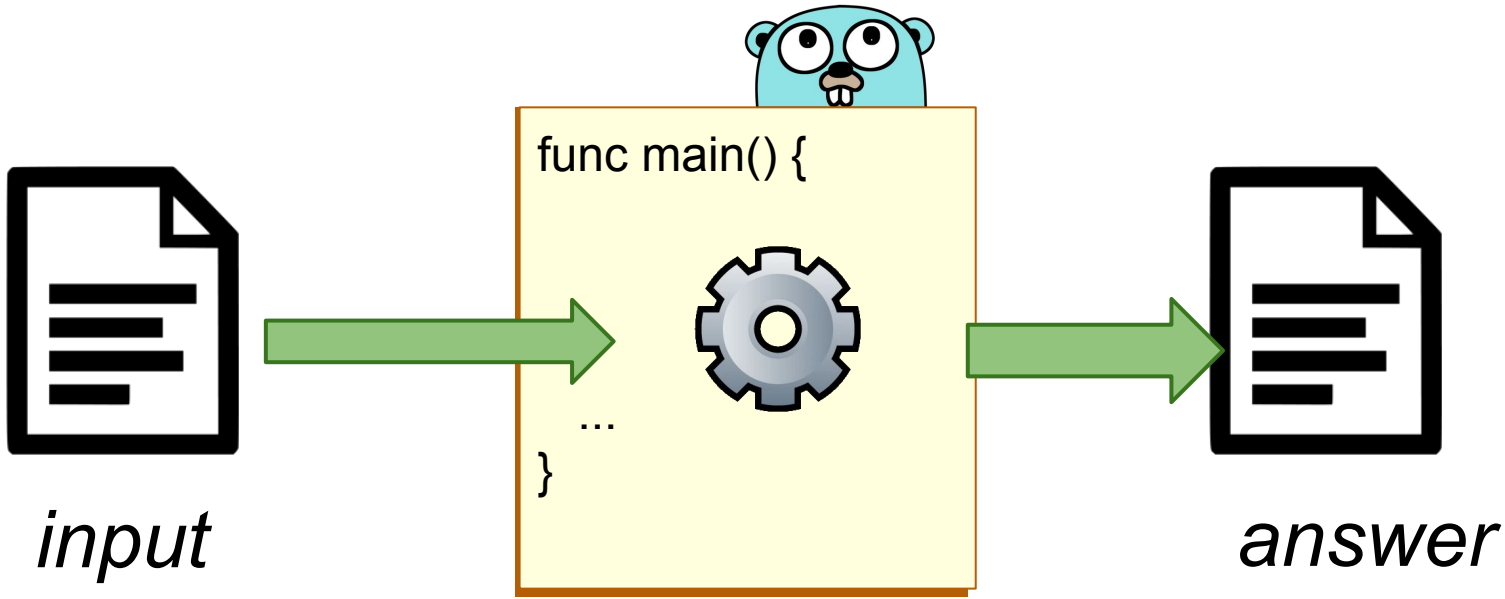
Problem

Find the sum of all these numbers

171 220 244 215 1
88 200 113 34 99
91 22 291 90 214

Fastest contestants win a t-shirt !!

Read input
Write answer
Upload answer



Local vars

Quick, short and readable

```
func solve() interface{} {
    N := readInt()
    a := make([]int, N)
    for i := range a {
        a[i] = readInt()
    }

    sum := 0
    for _, x := range a {
        sum += x
    }
    return sum
}
```

Global vars

Also quick, short and readable

```
var N int
var a []int

func read() {
    N = readInt()
    a = make([]int, N)
    for i := range a {
        a[i] = readInt()
    }
}
```

```
func solve() interface{} {
    sum := 0
    for _, x := range a {
        sum += x
    }
    return sum
}
```

Quick, short and readable (local vars, or global vars)

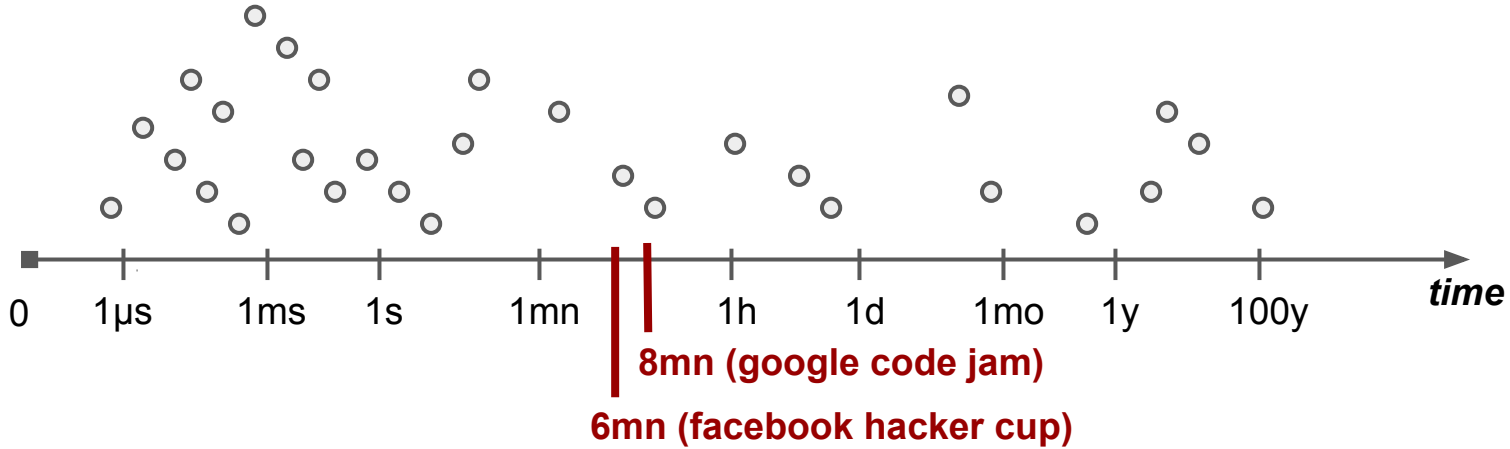
But it’s sequential !

Quick, short and readable (local vars, or global vars)

But it’s sequential !

But it doesn’t matter !

Common solving times (log scale)



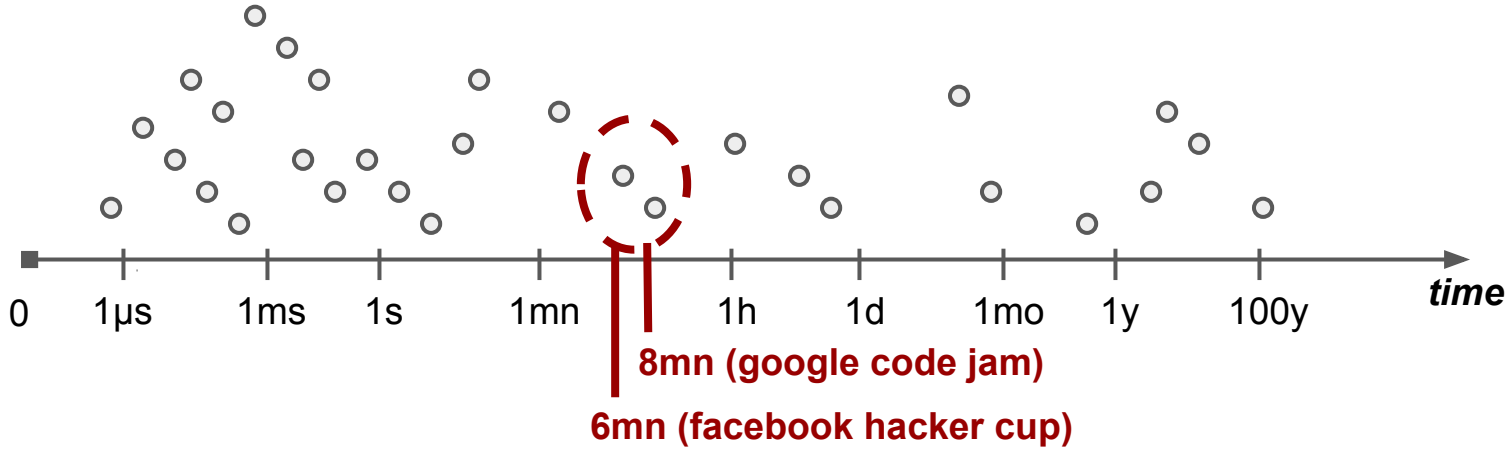
Quick, short and readable (local vars, or global vars)

But it’s sequential !

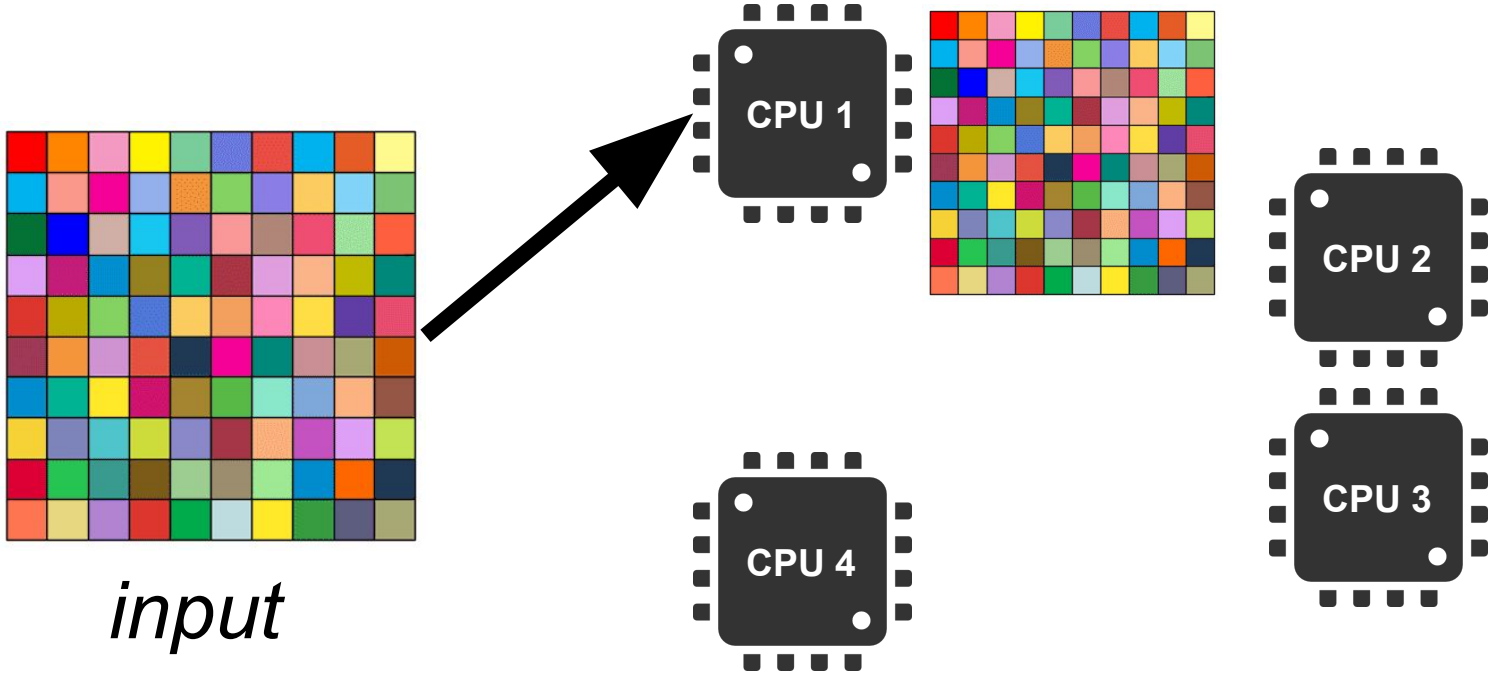
But it doesn’t matter !

… but sometimes (not often) it does matter !

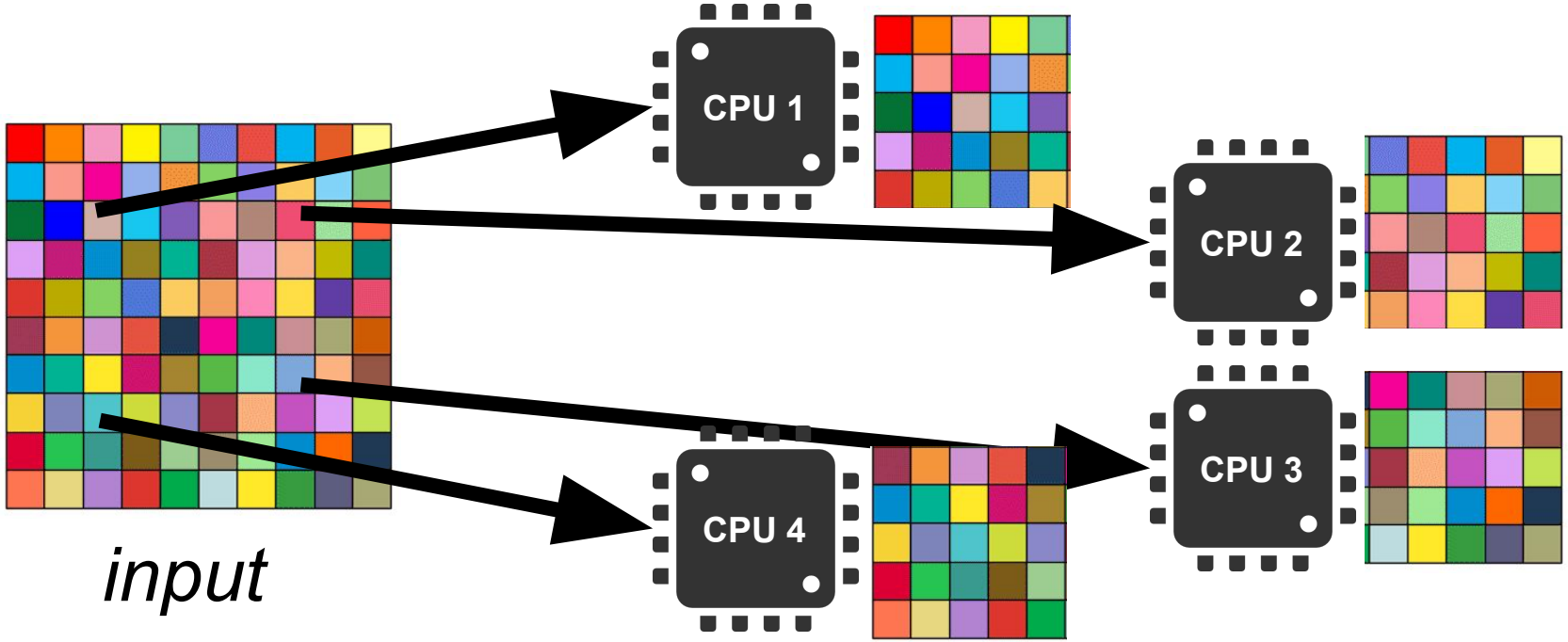
Common solving times (log scale)



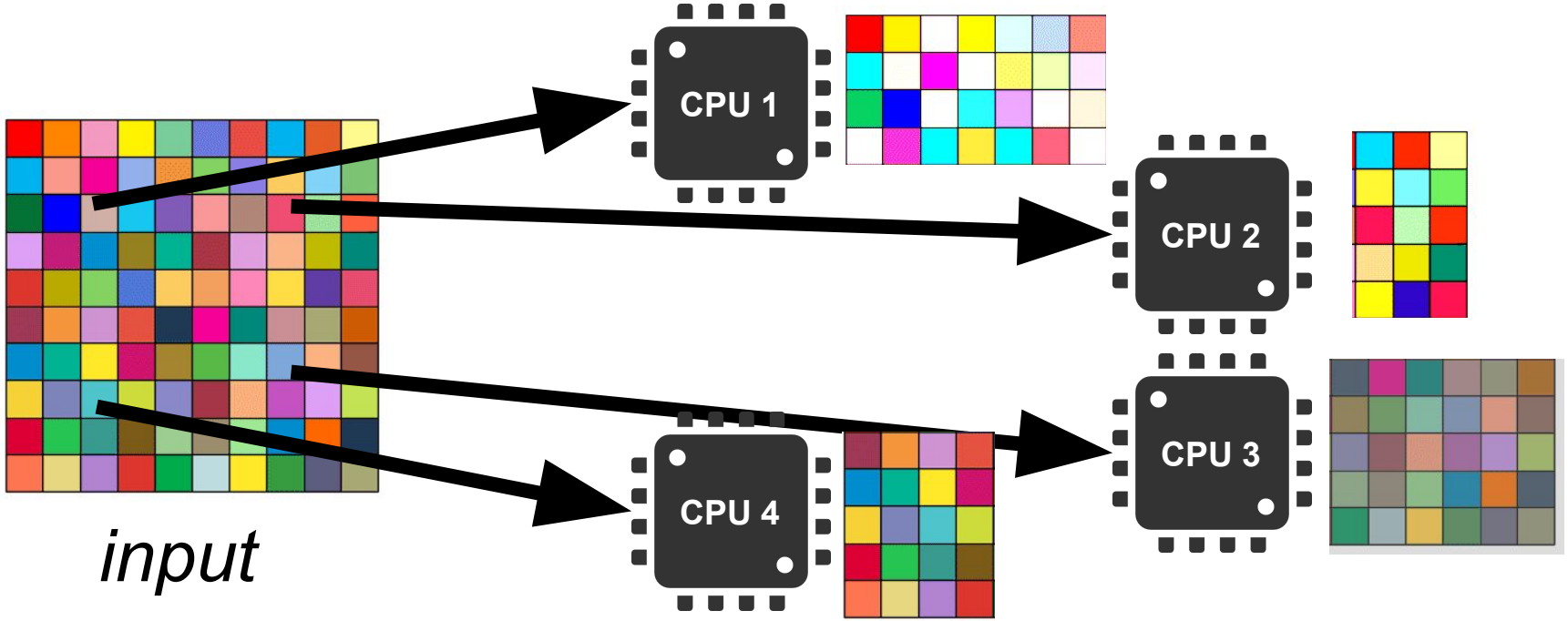
Work dispatch (no concurrency)



Work dispatch (homogeneous input)



Work dispatch (heterogeneous input)



Structured and concurrent

```
type Case struct {
    N int
    a []int
}

func (z *Case) read() {
    z.N = readInt()
    z.a = make([]int, z.N)
    for i := range z.a {
        z.a[i] = readInt()
    }
}
```

```
func (z *Case) solve() interface{} {
    sum := 0
    for _, x := range z.a {
        sum += x
    }
    return sum
}
```

Conciseness

```
d2 := z.x*z.x + z.y*z.y
```

struct fields

```
d2 := x*x + y*y
```

global or local vars

Readable, sequential

```
func solve() interface{} {
    N := readInt()
    a := make([]int, N)
    for i := range a {
        a[i] = readInt()
    }


    sum := 0
    for _, x := range a {
        sum += x
    }
    return sum
}
```

Readable and concurrent

```
var readWG sync.WaitGroup

func solve() interface{} {
    N := readInt()
    a := make([]int, N)
    for i := range a {
        a[i] = readInt()
    }
    readWG.Done()

    sum := 0
    for _, x := range a {
        sum += x
    }
    return sum
}
```

Aux funcs ?

```
var readWG sync.WaitGroup

func solve() interface{} {
    N := readInt()
    a := make([]int, N)
    for i := range a {
        a[i] = readInt()
    }
    readWG.Done()
```

```
    delta := func(x int) int {
        return a[0] - x
    }

    sum := 0
    for _, x := range a {
        sum += delta(x)
    }
    return sum
}
```

Also available in chan flavor

```
func solve(solving chan<- Ø) interface{} {
    N := readInt()
    a := make([]int, N)
    for i := range a {
        a[i] = readInt()
    }
    solving <- ø

    sum := 0
    for _, x := range a {
        sum += x
    }
    return sum
}
```