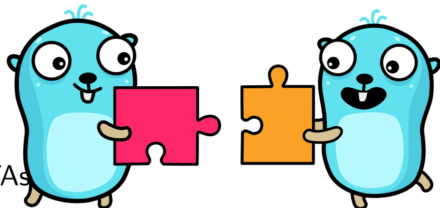
# CS3211 Tutorial 6

## Advanced Go concurrency patterns

(AY 25/26 Semester 2)

March 19, 2026



(Compiled by Benson, thanks to all past and current TAs)

# Contents

Fan-out, Fan-in

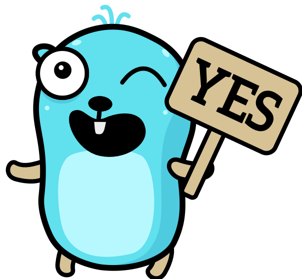
- Basic Implementation
- Serializer Goroutine
- Promises and Higher order Channels

Pipelining

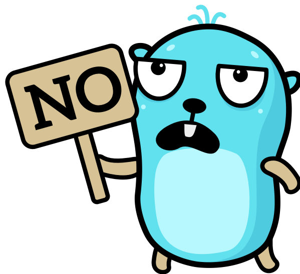
# Recap: Channels

- True or False: Reading from a closed channel causes a panic.

A.




B.

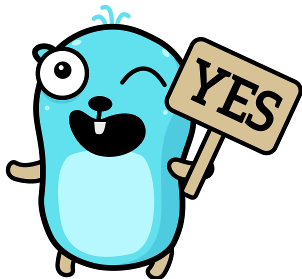



No, you get the default value.

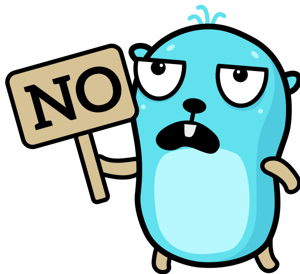
# Recap: Channels

True or False: It is possible for a `select` block to block the goroutine.

A.




B.




Yes, when there is no `default` case.

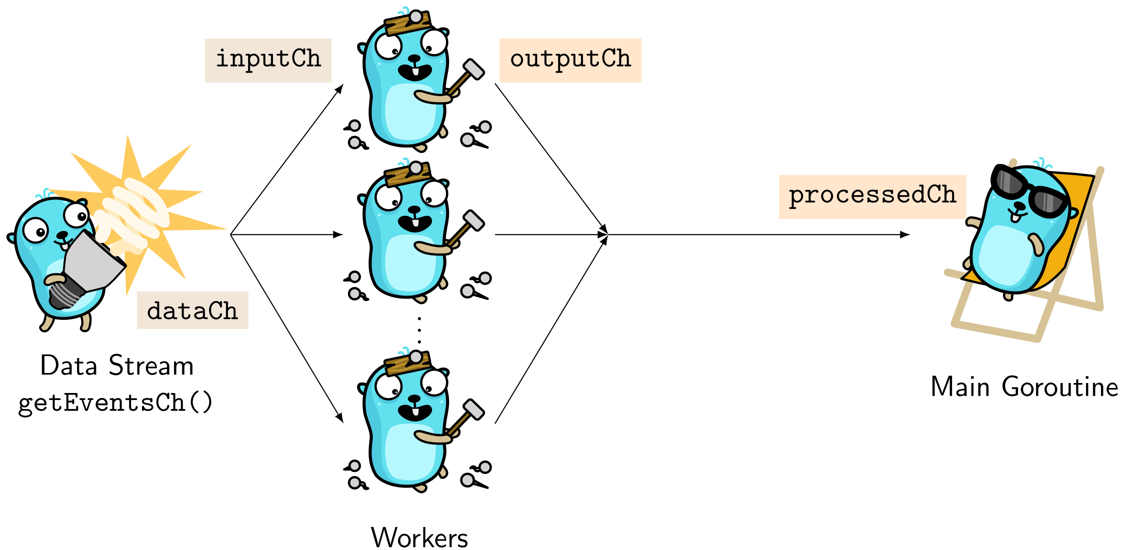
# Contents


Fan-out, Fan-in

- Basic Implementation
- Serializer Goroutine
- Promises and Higher order Channels

Pipelining

# Fan-out, Fan-in



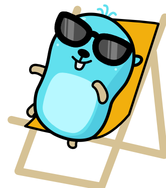
# Fan-out, Fan-in

```go
type Event struct {
  id       int64
  procTime time.Duration
}

func genEventsCh() chan Event {
  dataCh := make(chan Event)
  go func() {
    ... // generates the events
  }
  return dataCh
}

func (w *worker) start(
    ctx context.Context,
    fn EventFunc, wg *sync.WaitGroup,
) {
  go func() {
    ... // processes the events
  }
}
```

Link to godbolt

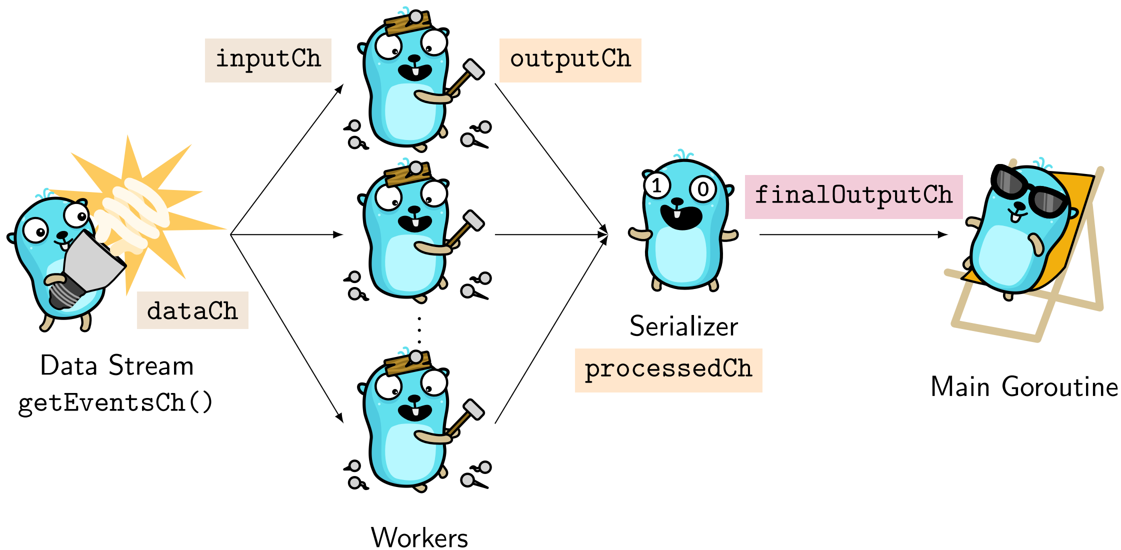
Data Stream

Workers



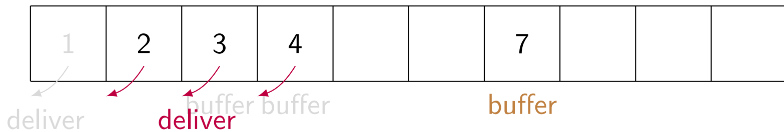
Main

# Fan-out, Fan-in with Serializer



# Fan-out, Fan-in with Serializer

Serializer

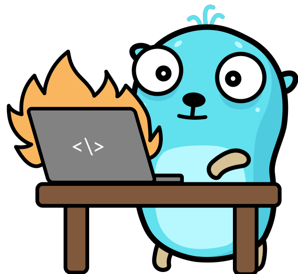


# Fan-out, Fan-in with Serializer

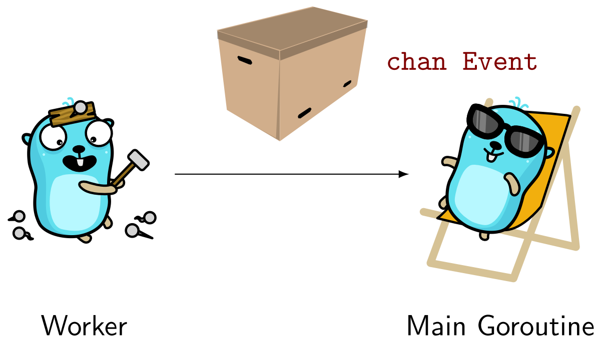
```
1  // Serializer goroutine
2  go func() {
3    eventMap := make(map[int64]Event) // our cache
4    var curEventId int64 = 1
5    for output := range processedCh {
6      ... // TODO: implement logic
7    }
8  }
```

Link to godbolt

Let's add the serializer together!

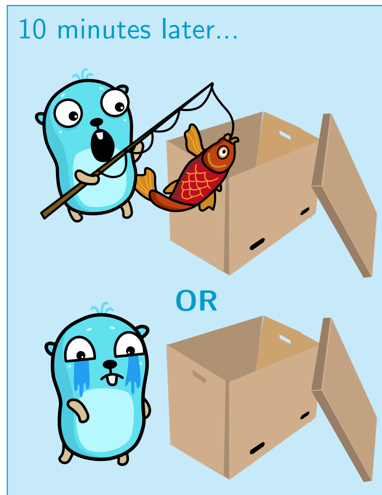


# Promises

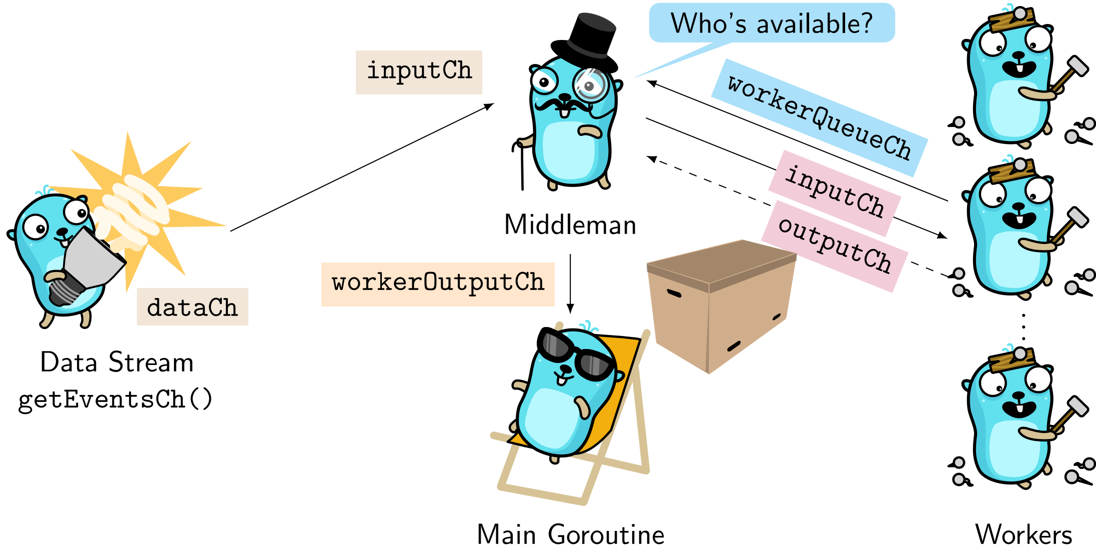



Worker

Main Goroutine



# Fan-out, Fan-in with Promises



# Fan-out, Fan-in with Promises

What is the correct type of `inputCh` and `outputCh`?

|  | `inputCh` | `outputCh` |
| --- | --- | --- |
| A. | `chan Event` | `chan Event` |
| B. | `chan Event` | `chan chan Event` |
| C. | `chan chan Event` | `chan Event` |
| D. | `chan chan Event` | `chan chan Event` |

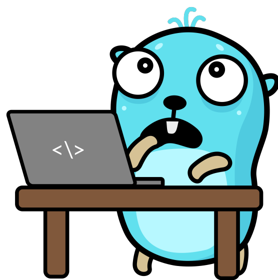
**A**. Only `workerOutputCh` is a `chan chan Event` – the worker doesn't know its output channel is used as a promise.

# Fan-out, Fan-in with Promises

```
1  type worker struct {
2    inputCh  chan Event
3    outputCh chan Event
4  }
5
6  func newWorker() *worker {
7    return &worker{
8      inputCh:  make(chan Event),
9      outputCh: make(chan Event),
10   }
11 }
12
13 // for the middleman
14 select {
15 case w := <-workerQueue:
16   workerOutputCh <- w.outputCh
17   w.inputCh <- e
18 case <-ctx.Done():
19   return
20 }
```

Link to godbolt

Let's add the middleman together!



# Contents

Fan-out, Fan-in

- Basic Implementation
- Serializer Goroutine
- Promises and Higher order Channels

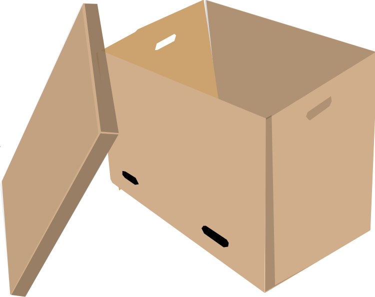
Pipelining

# A typical client-server

Server

Server

reqCh



Task Pool

# A typical client-server

Client

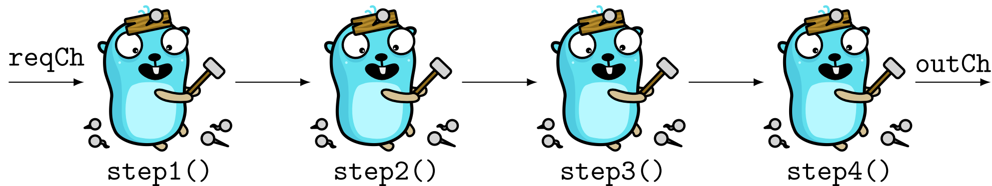
```go
for {
  select {
  case reqCh <- *NewReq():
  case <-ctx.Done():
    return
  }
}
```

Server

```go
for {
  select {
  case req := <-reqCh:
    processReqStep1(&req)
    processReqStep2(&req)
    processReqStep3(&req)
    processReqStep4(&req)
    req.Done()
  case <-ctx.Done():
    return
  }
}
```

Link to godbolt

# Pipelining

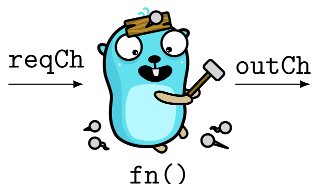



What if we know that the 4 steps take 2M, 5M, 10M, 2M iterations respectively?

# Pipelining

```
1  func toPipelineStage(
2    ctx context.Context,
3    reqCh <-chan Request, instances int,
4    fn func(*Request) *Request,
5  ) chan Request {
6    outCh := make(chan Request)
7    for i := 0; i < instances; i++ {
8      go func() {
9        for req := range reqCh {
10         select {
11         case outCh <- *fn(&req):
12         case <-ctx.Done():
13           return
14         }
15       }
16     }()
17   }
18   return outCh
19 }
```

Link to godbolt

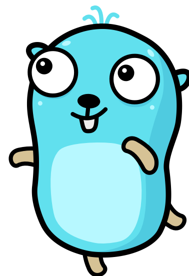



```
c := make(chan Request, 100)
// Make pipeline of 4 stages
c = toPipelineStage(ctx, c, 2, step1)
c = toPipelineStage(ctx, c, 5, step2)
c = toPipelineStage(ctx, c, 10, step3)
c = toPipelineStage(ctx, c, 5, step4)
```

# Summary

- Complex constructs using Goroutines and Channels:
  - **Fan-out**: Distributing intense work to many goroutines.
  - **Fan-in**: Synchronizing goroutines and serializing results.
  - **Pipelining**: Resource-constrained parallelism, allows separation of concerns between stages.

# Attendance



Session ID: 1940156
CS3211 Tutorial 6
19 March 2026 14:00-16:00
BIZ2-02-02 - SEMINAR ROOM 2-2

# That's it!

Anonymous Feedback (throughout the semester):

https://forms.gle/6a9T4t88wNwYxG4G8

The link to the tutorial slides will be posted in telegram.