Let's Go Asynchronous

Tomáš Sedláček

CTO, dataddo.com

mail@kedlas.cz

https://www.linkedin.com/in/tomasedlacek/

Q&A

Monolithic app

~~Monolithic app~~

Synchronously communicating microservices

~~Monolithic app~~

~~Synchronously communicating microservices~~

Asynchronously communicating microservices
AWS Managed RabbitMQ

~~Monolithic app~~

~~Synchronously communicating microservices~~

Asynchronously communicating microservices

~~Managed RabbitMQ~~

PGQ

Meet Orafo

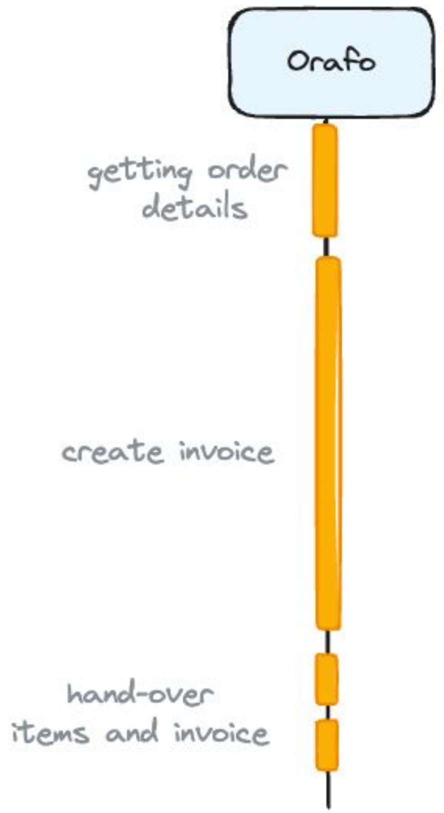
Order process

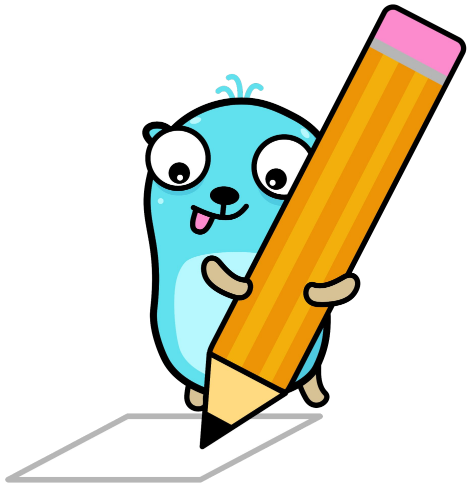
- get and validate items
- get and validate customer details
- apply coupons
- create the order db record
- generate invoice
- process payment
- order shipping
- synchronize with CRM/ERP systems
- ...

Simplified order process

1. get order details (validate items & customer)
2. generate invoice
3. hand-over

Synchronous processing

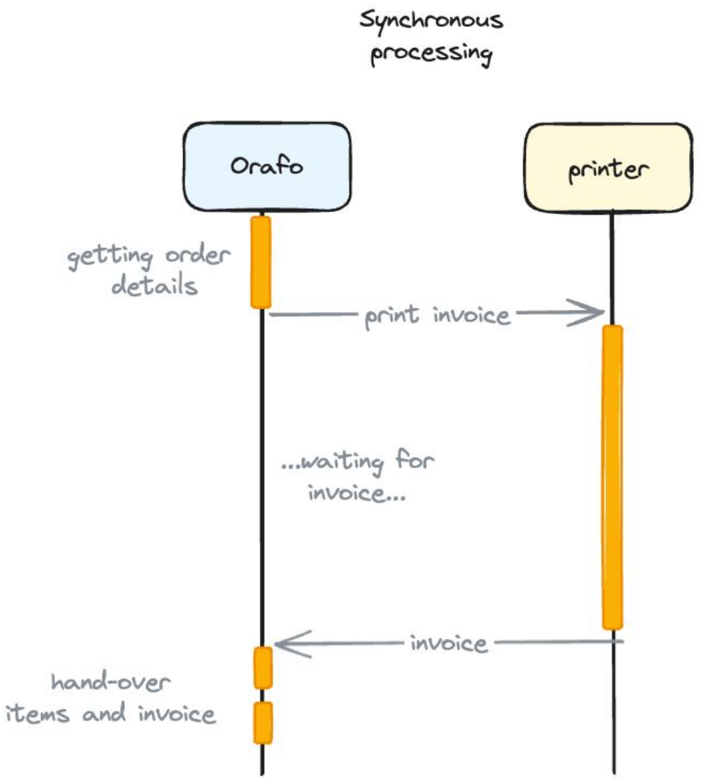


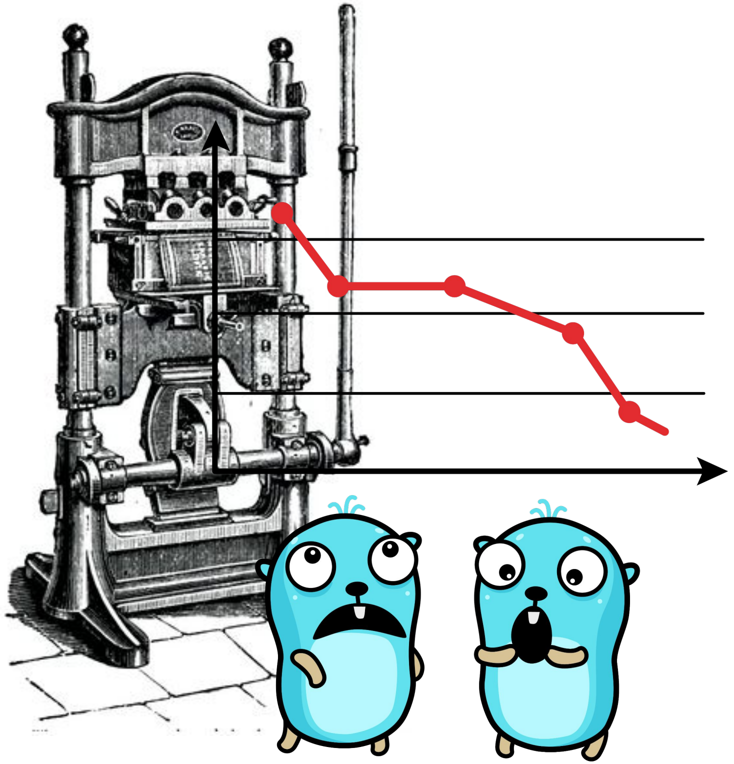


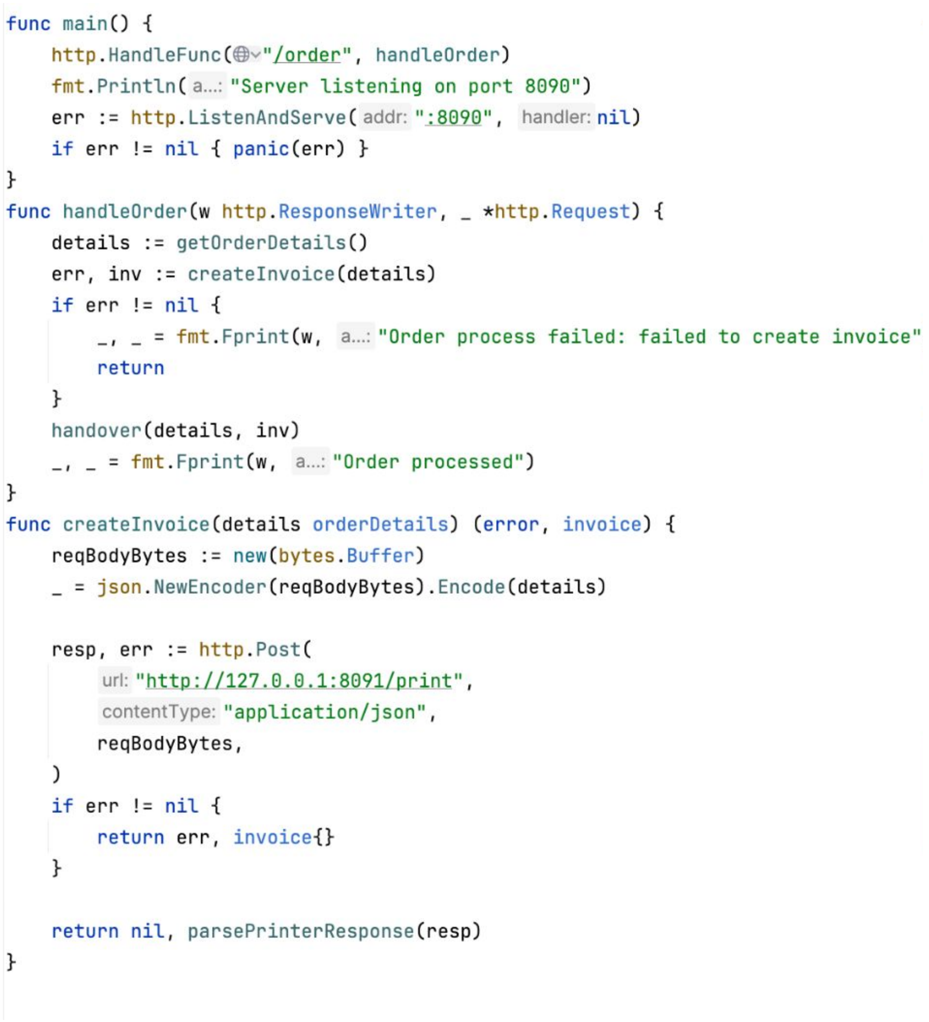
sync/http

```go
type orderDetails struct{}
type invoice struct{}

func main() {
	http.HandleFunc("/order", handleOrder)
	fmt.Println( a...: "Server listening on port 8090")
	err := http.ListenAndServe( addr: ":8090",  handler: nil)
	if err != nil {
		panic(err)
	}
}
func handleOrder(w http.ResponseWriter, _ *http.Request) {
	details := getOrderDetails()
	inv := createInvoice(details)
	handover(details, inv)
	_, _ = fmt.Fprint(w,  a...: "Order processed")
}
func getOrderDetails() orderDetails {
	time.Sleep(100 * time.Millisecond)
	return orderDetails{}
}
func createInvoice(_ orderDetails) invoice {
	time.Sleep(5 * time.Second)
	return invoice{}
}
func handover(_ orderDetails, _ invoice) {
	time.Sleep(200 * time.Millisecond)
}
```


Synchronous processing
Orafo
printer
getting order details
print invoice
...waiting for invoice...
invoice
hand-over items and invoice

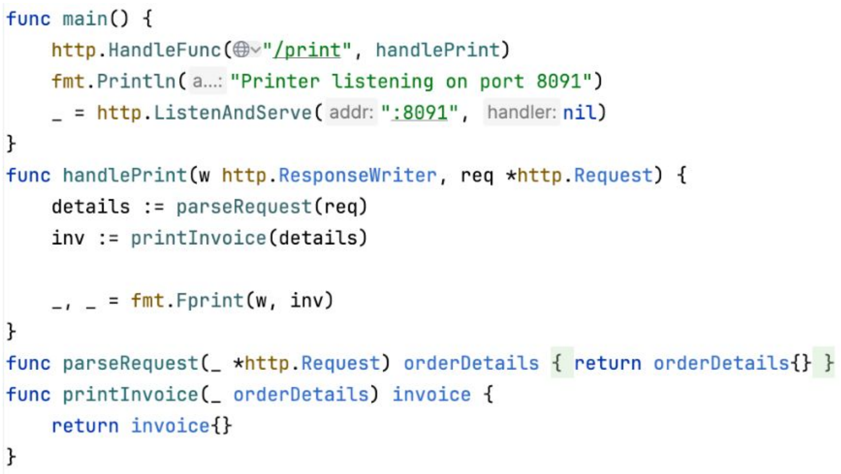




```
func main() {
    http.HandleFunc("/order", handleOrder)
    fmt.Println( a...: "Server listening on port 8090")
    err := http.ListenAndServe( addr: ":8090",  handler: nil)
    if err != nil { panic(err) }
}
func handleOrder(w http.ResponseWriter, _ *http.Request) {
    details := getOrderDetails()
    err, inv := createInvoice(details)
    if err != nil {
        _, _ = fmt.Fprint(w,  a...: "Order process failed: failed to create invoice"
        return
    }
    handover(details, inv)
    _, _ = fmt.Fprint(w,  a...: "Order processed")
}
func createInvoice(details orderDetails) (error, invoice) {
    reqBodyBytes := new(bytes.Buffer)
    _ = json.NewEncoder(reqBodyBytes).Encode(details)

    resp, err := http.Post(
        url: "http://127.0.0.1:8091/print",
        contentType: "application/json",
        reqBodyBytes,
    )
    if err != nil {
        return err, invoice{}
    }

    return nil, parsePrinterResponse(resp)
}
```



```
func main() {
    http.HandleFunc("/print", handlePrint)
    fmt.Println( a...: "Printer listening on port 8091")
    _ = http.ListenAndServe( addr: ":8091",  handler: nil)
}
func handlePrint(w http.ResponseWriter, req *http.Request) {
    details := parseRequest(req)
    inv := printInvoice(details)

    _, _ = fmt.Fprint(w, inv)
}
func parseRequest(_ *http.Request) orderDetails { return orderDetails{} }
func printInvoice(_ orderDetails) invoice {
    return invoice{}
}
```

sync/http-printer

synchronous code

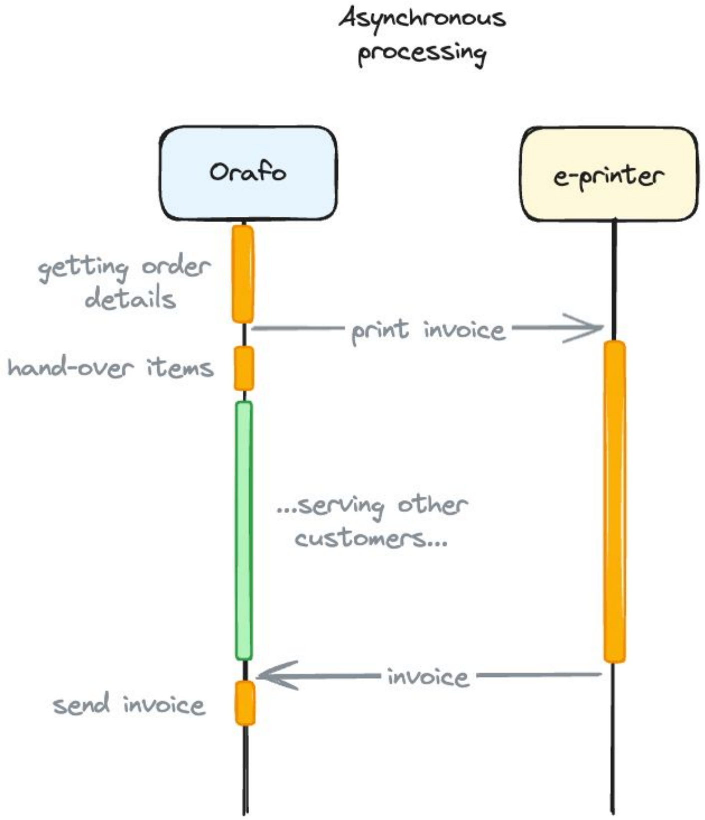
- simple to write, read and debug
- ordered, goes function by function
- total time is the sum of each function times
- usually highly I/O dependant
- no concurrency, unless you use goroutines, async/await or similar

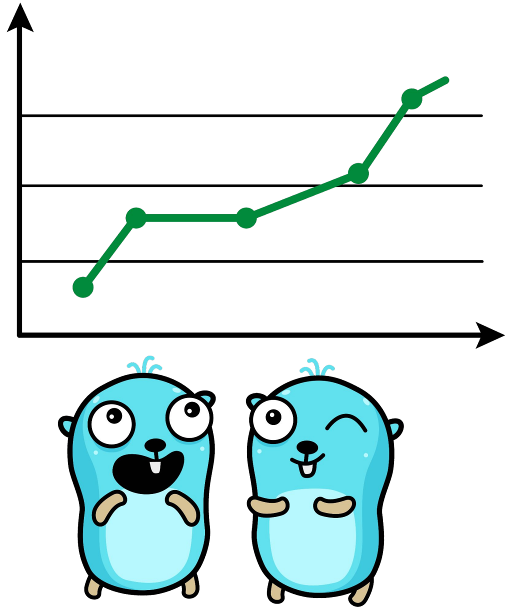
synchronous code

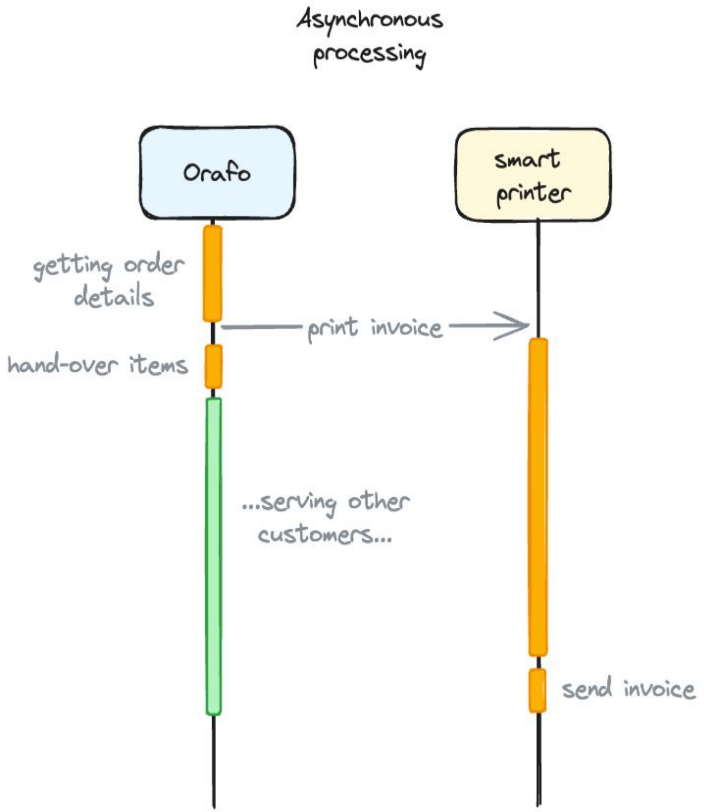
- simple to write, read and debug
- ordered, goes function by function
- total time is the sum of each function times
- highly I/O dependant
- no concurrency, unless you use goroutines, async/await or similar

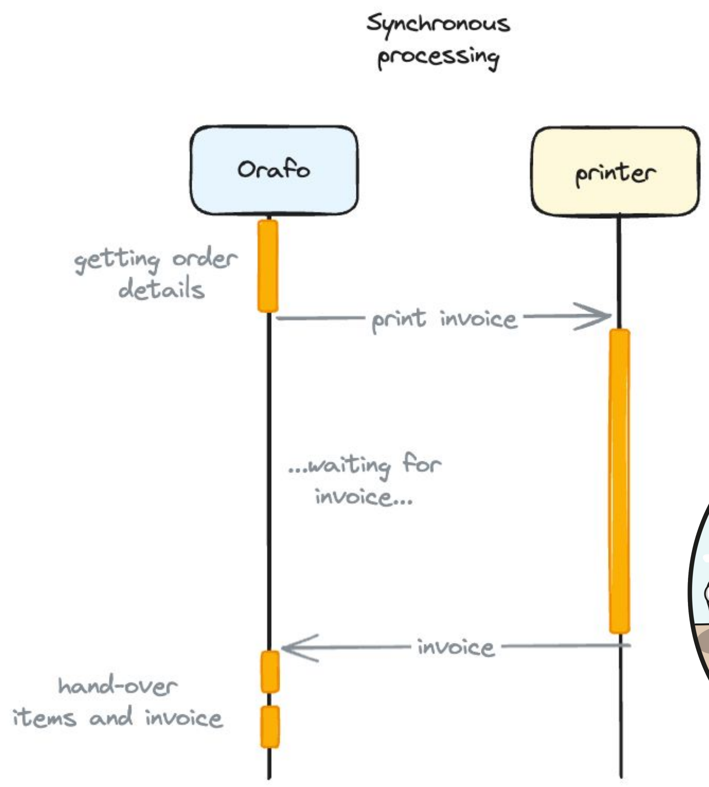
- on failures you must rollback, retry or ignore error
  - printer is offline
  - printer crash
  - printer out of ink, printer busy, printer timeout, ...

Let's Go Asynchronous

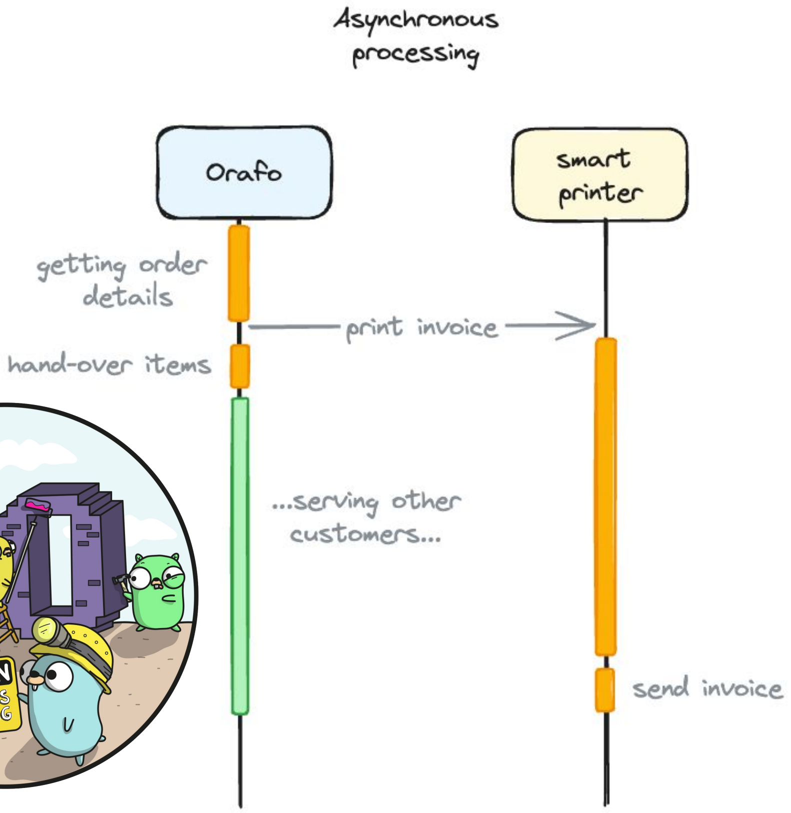

Asynchronous processing
Orafo
e-printer
getting order details
print invoice
hand-over items
...serving other customers...
invoice
send invoice



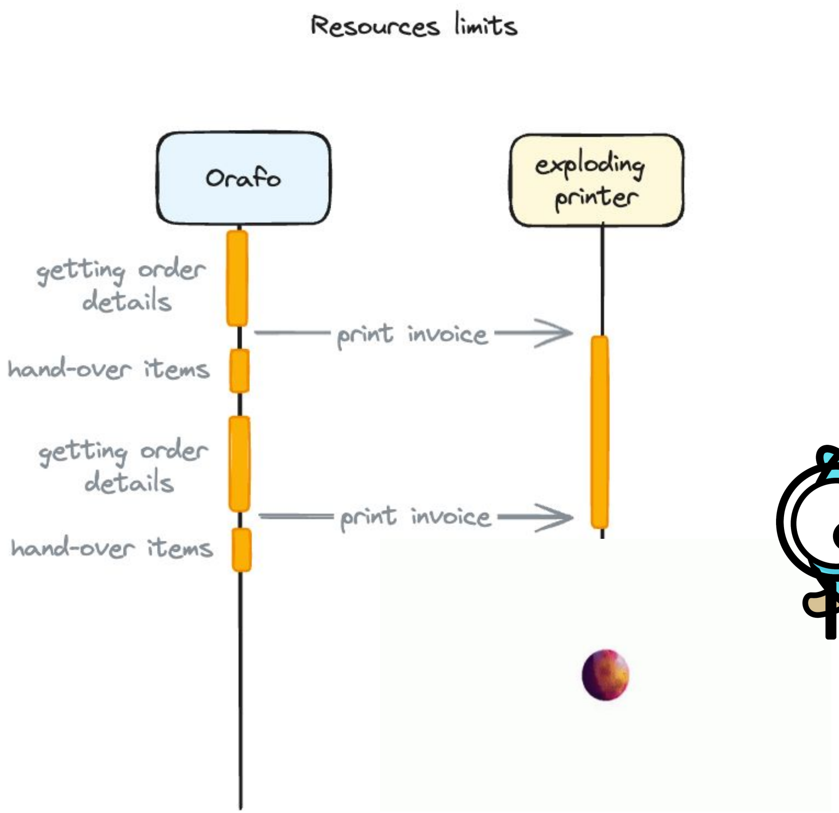

Asynchronous processing
Orafo
smart printer
getting order details
print invoice
hand-over items
...serving other customers...
send invoice

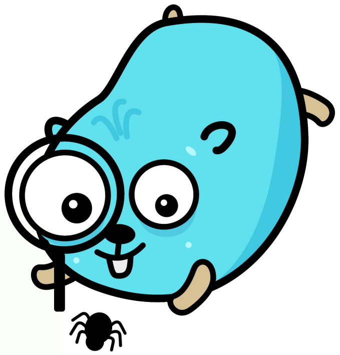

Synchronous processing
Orafo
printer
getting order details
print invoice
...waiting for invoice...
invoice
hand-over items and invoice

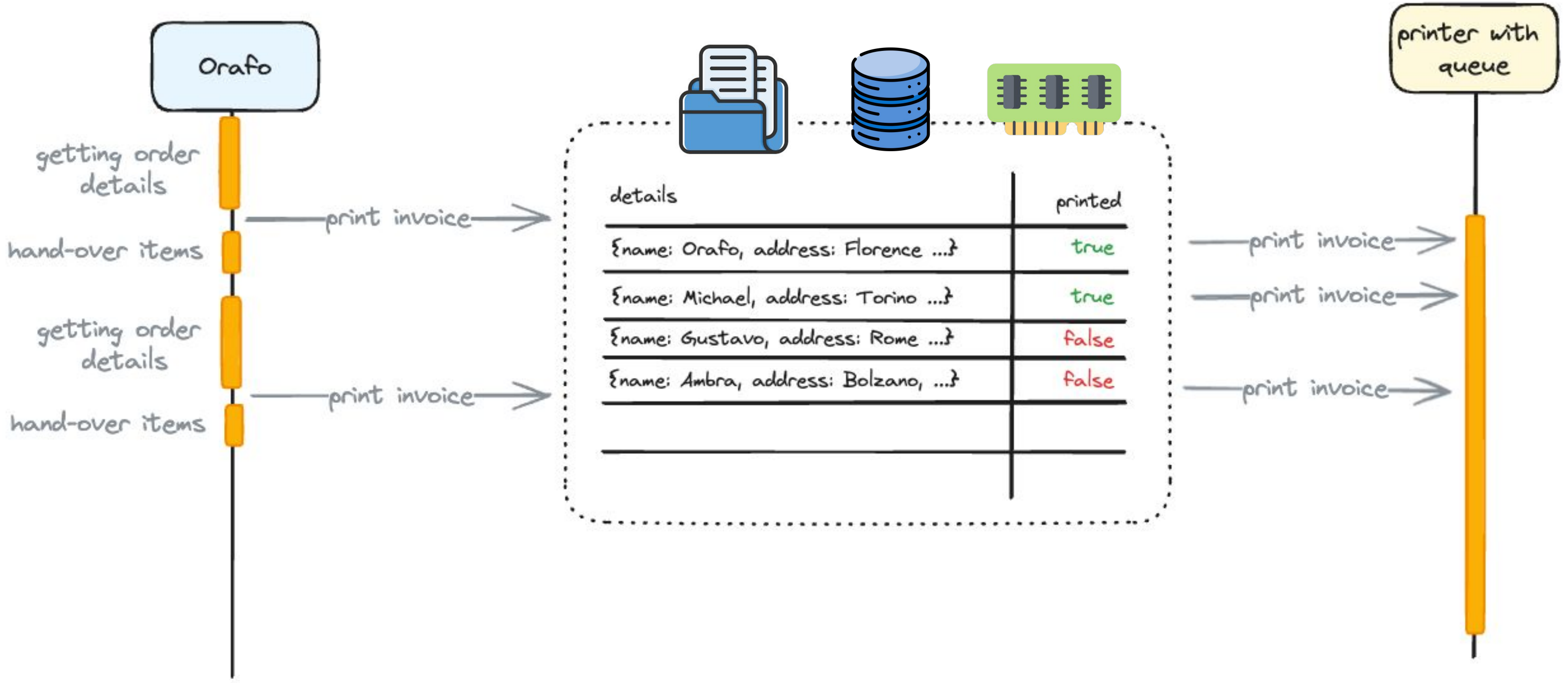
CAUTION
GOPHERS WORKING


Asynchronous processing
Orafo
smart printer
getting order details
print invoice
hand-over items
...serving other customers...
send invoice

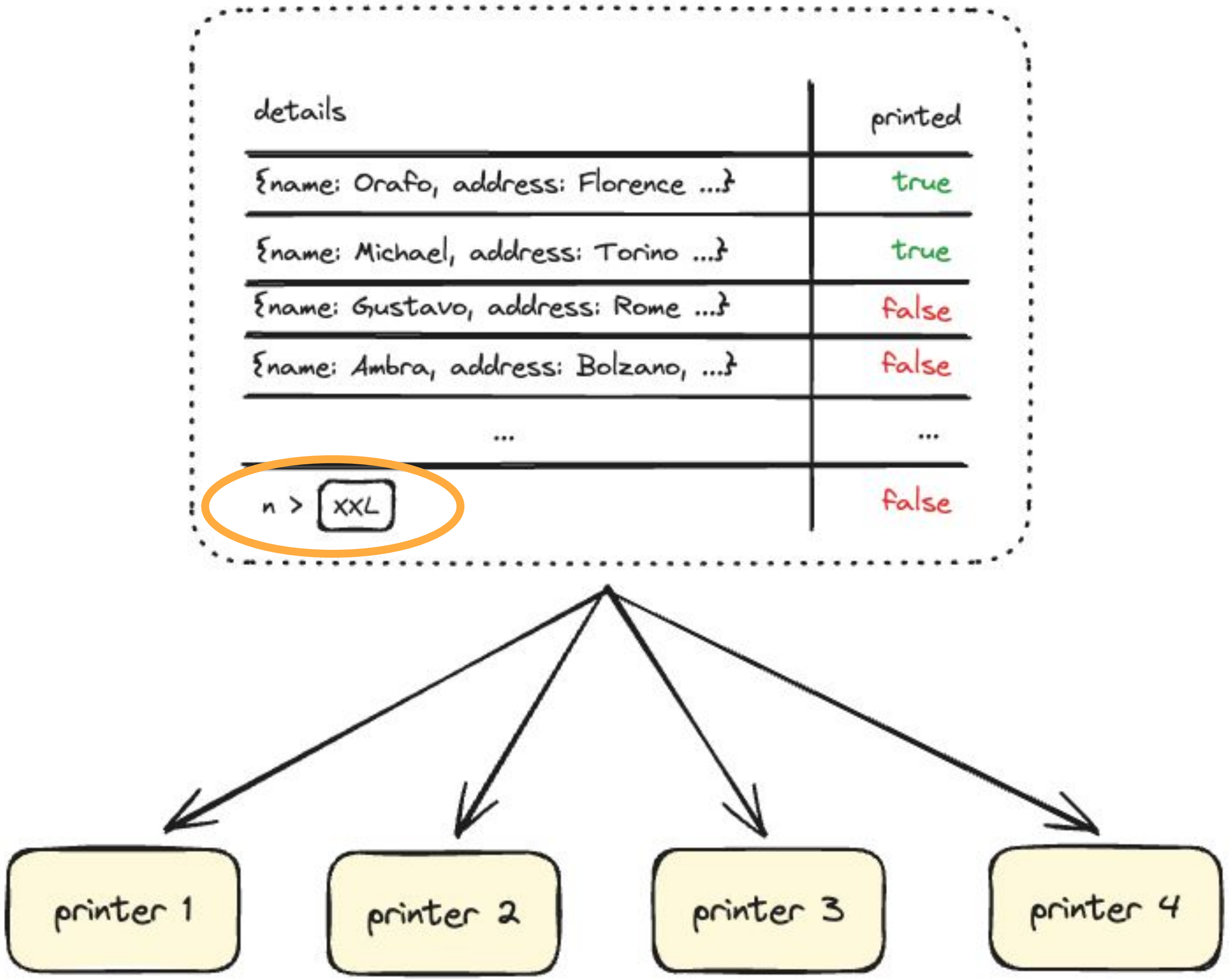
async/http/printer



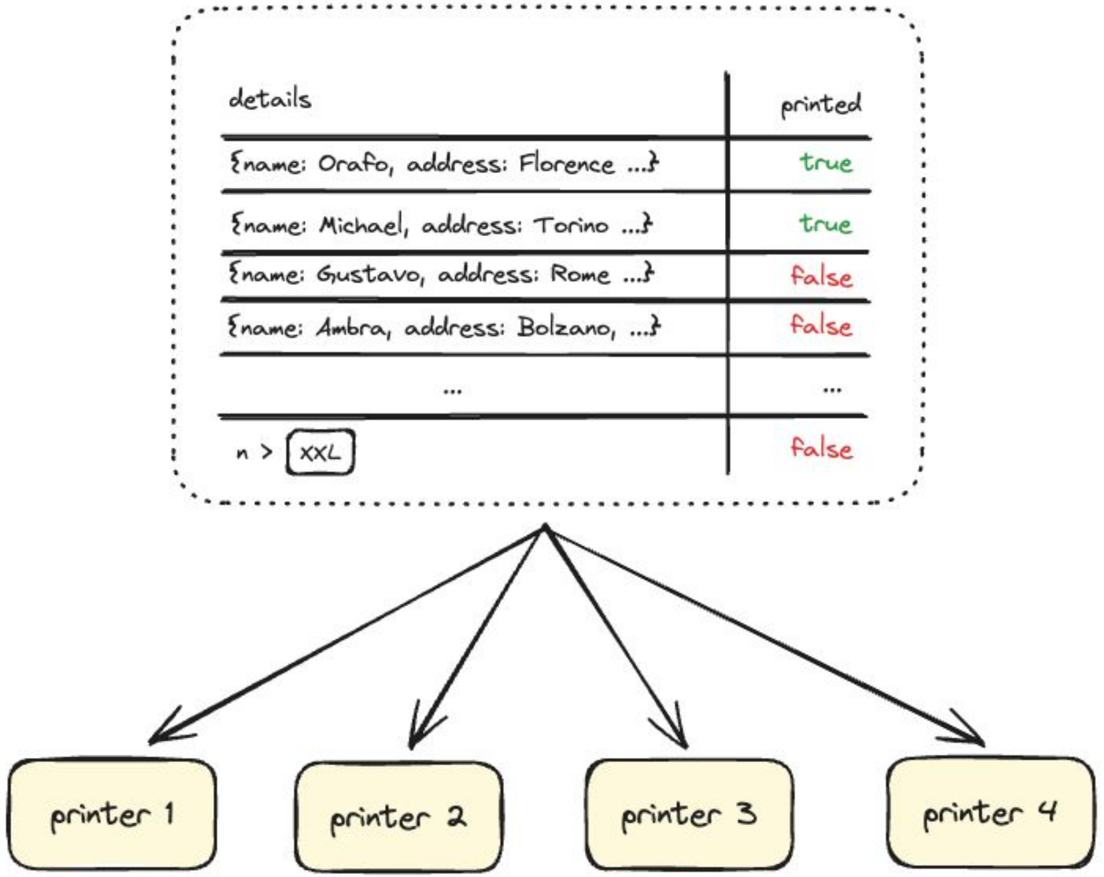




details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false

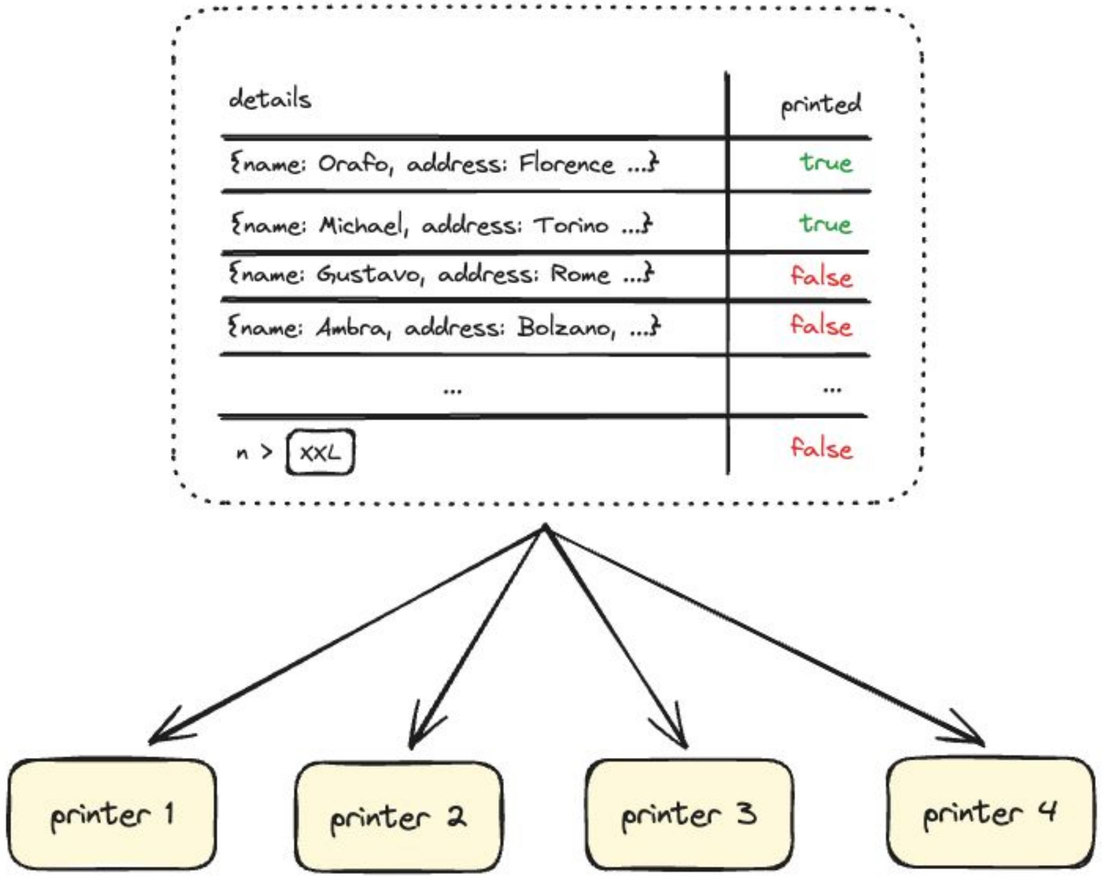


details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > $\boxed{\text{XXL}}$	false



details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > XXL	false

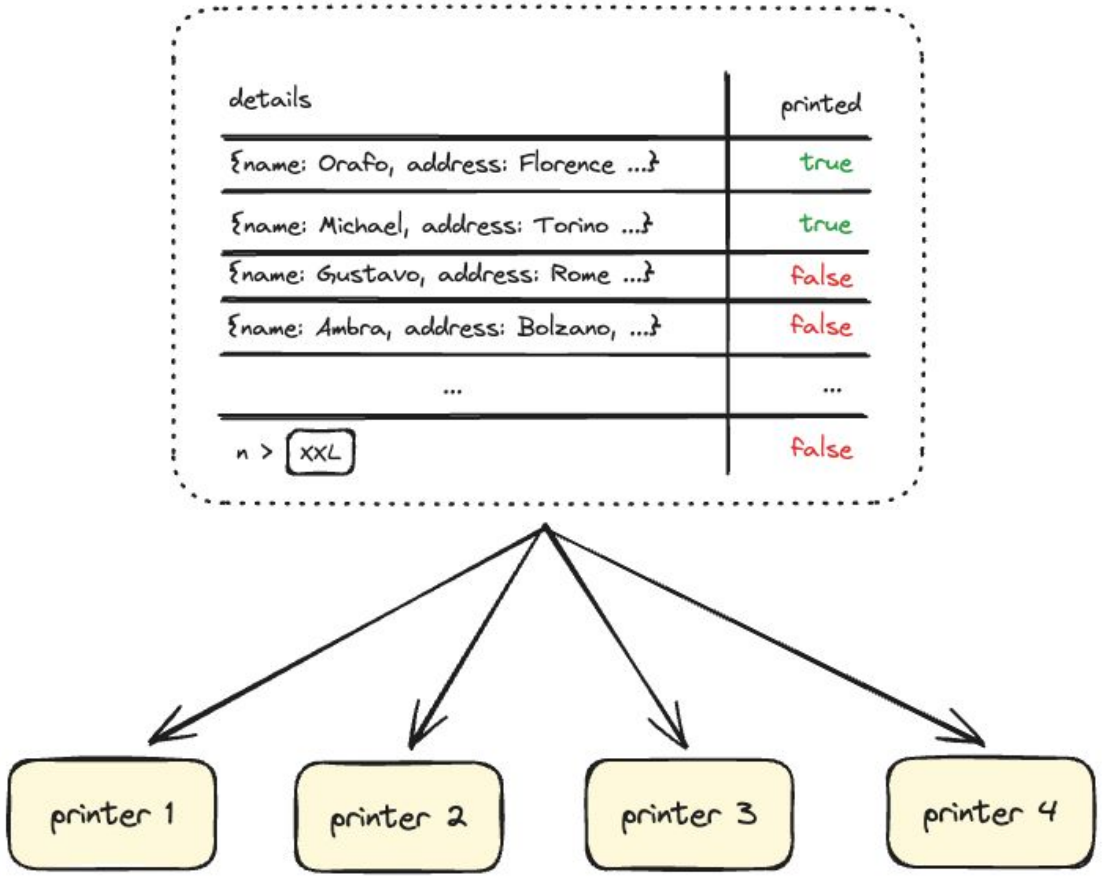
race conditions



details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > XXL	false

race conditions

routing

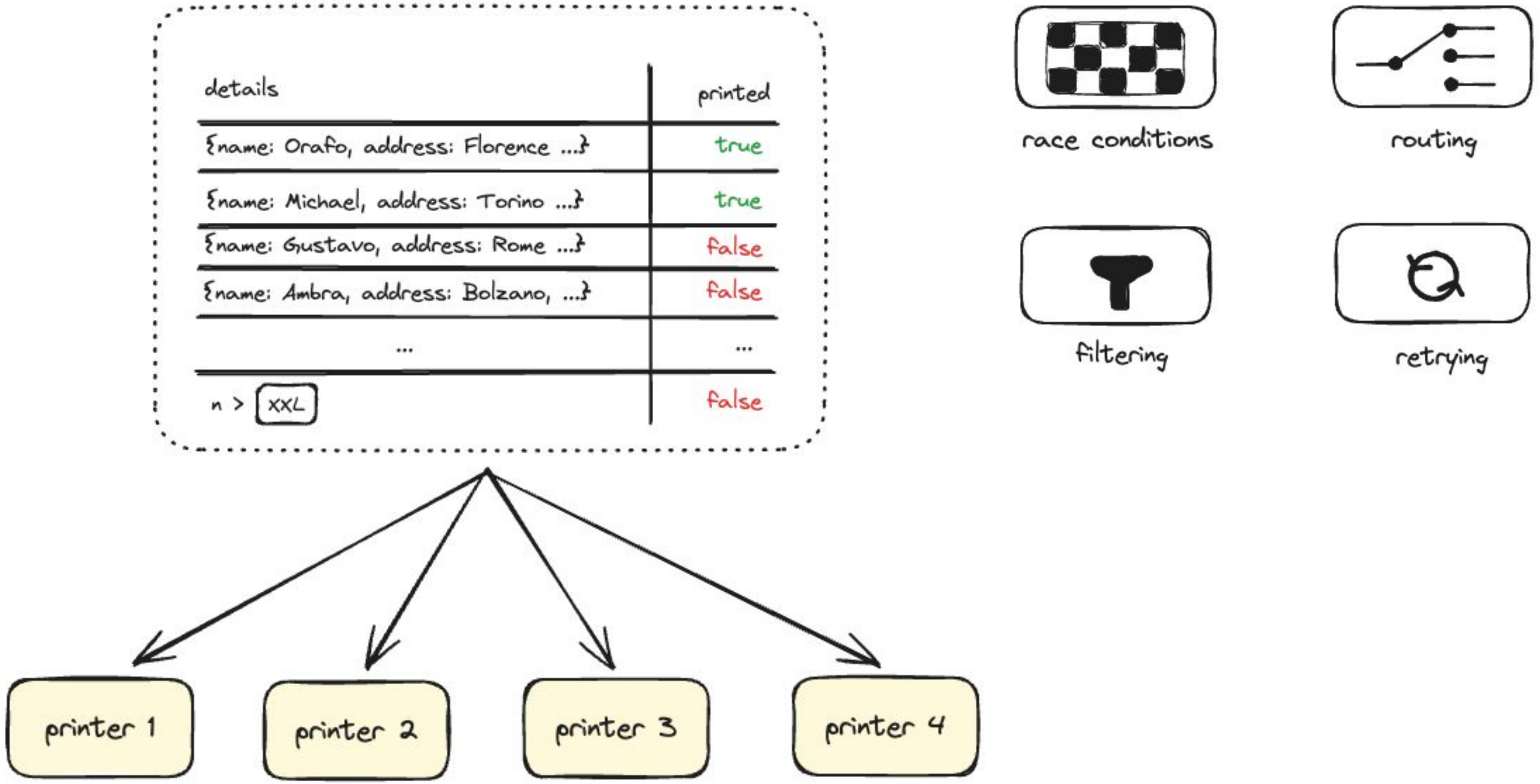


details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > $\boxed{\text{XXL}}$	false

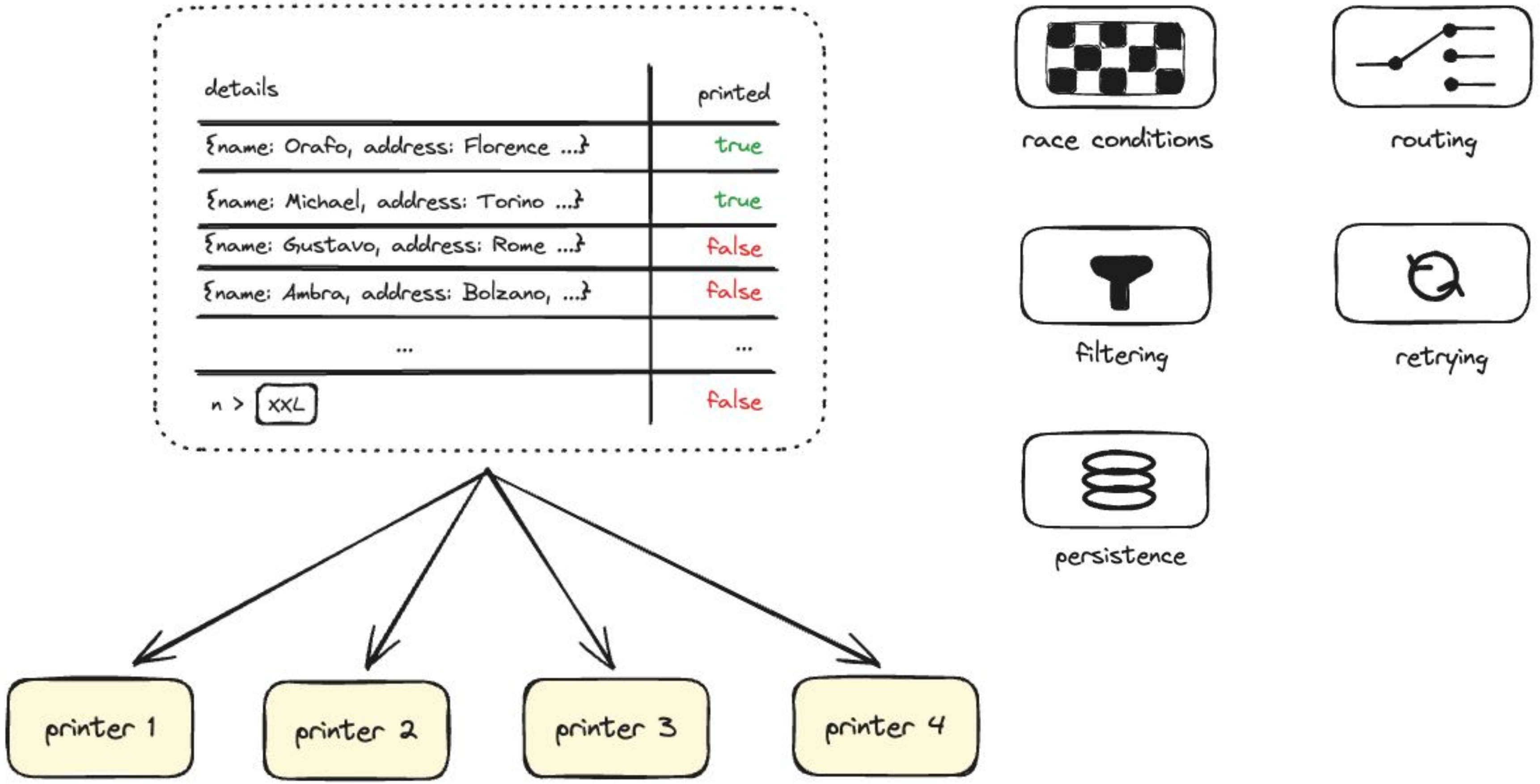
race conditions

routing

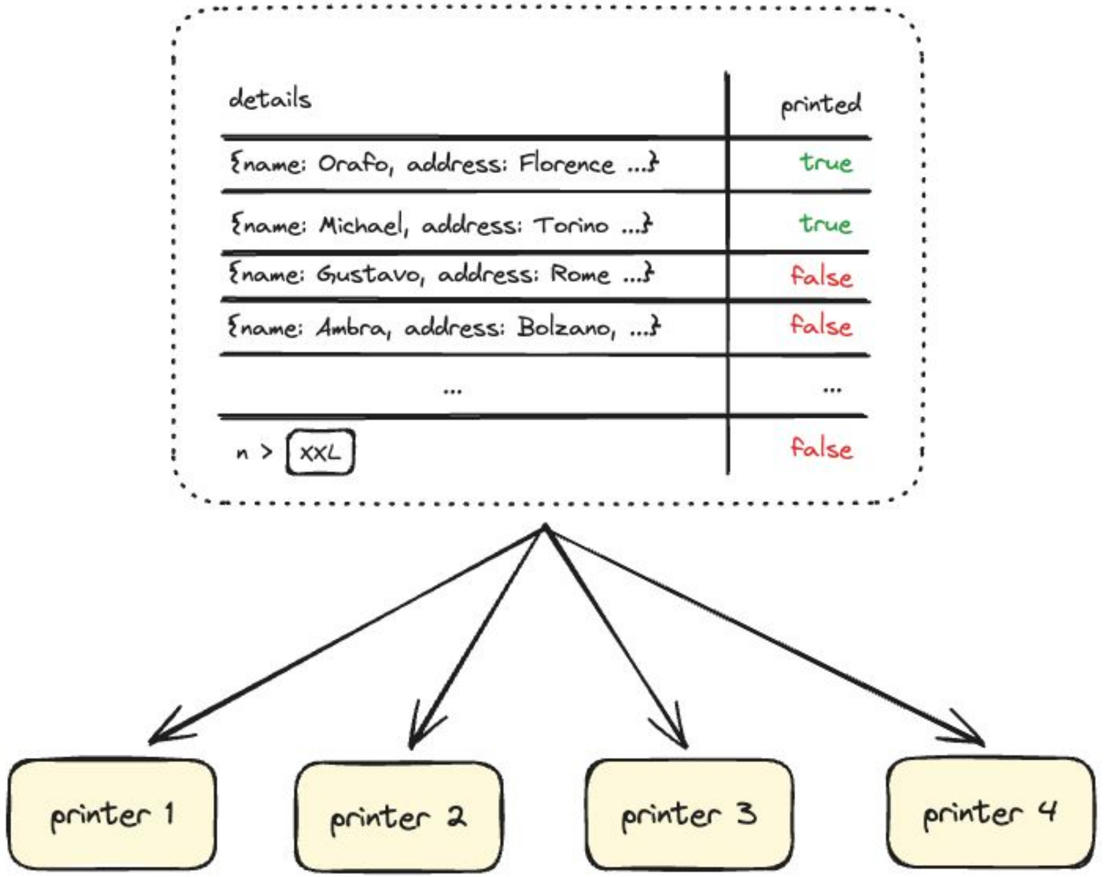
filtering



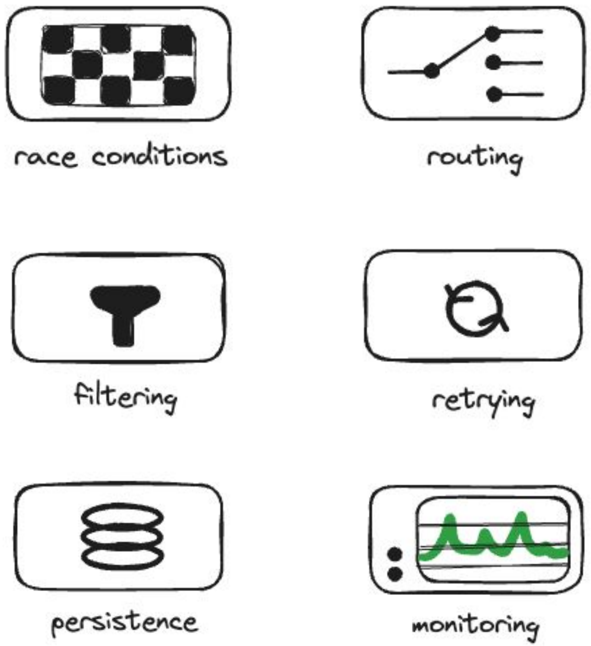
details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > XXL	false



details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > XXL	false



details	printed
{name: Orafo, address: Florence ...}	true
{name: Michael, address: Torino ...}	true
{name: Gustavo, address: Rome ...}	false
{name: Ambra, address: Bolzano, ...}	false
...	...
n > XXL	false



Message brokers

APACHE kafka®

RabbitMQ

amazon SQS

ACTIVEMQ

redis

NATS

Cloud Pub/Sub

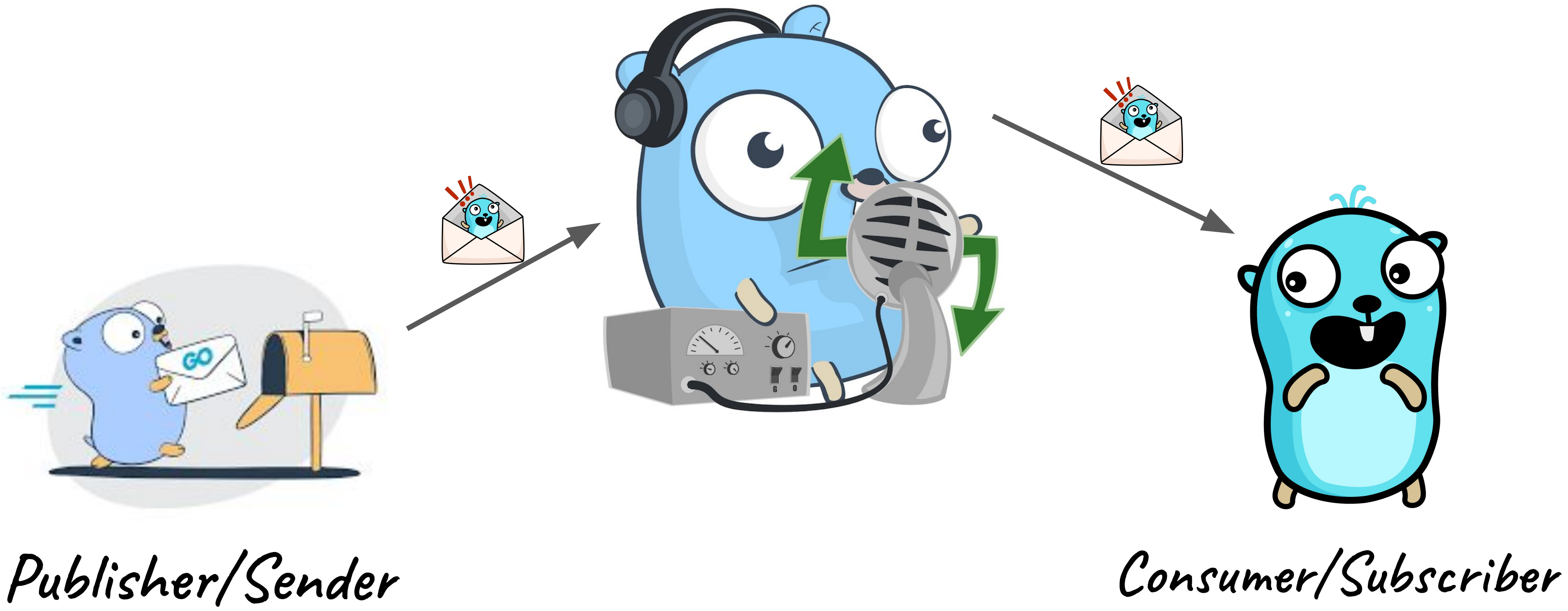
ØMQ

IBM MQ

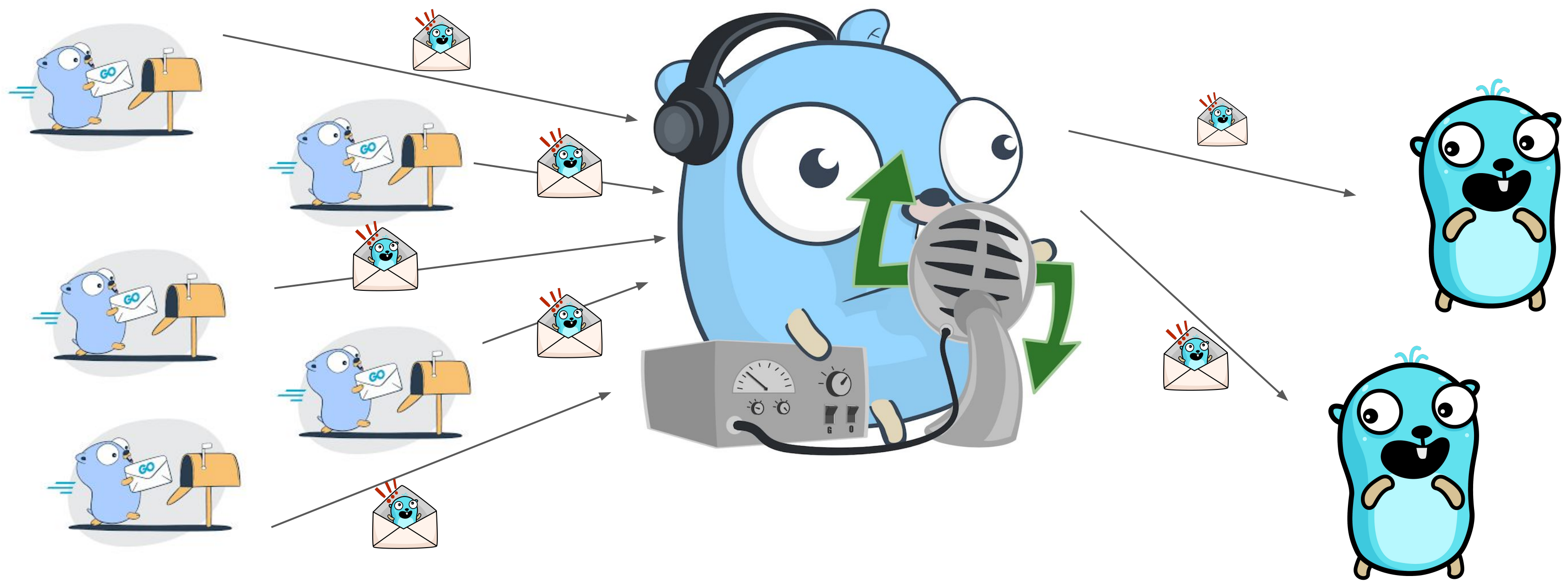
PULSAR

go.dataddo.com/pgq

Broker in the middle



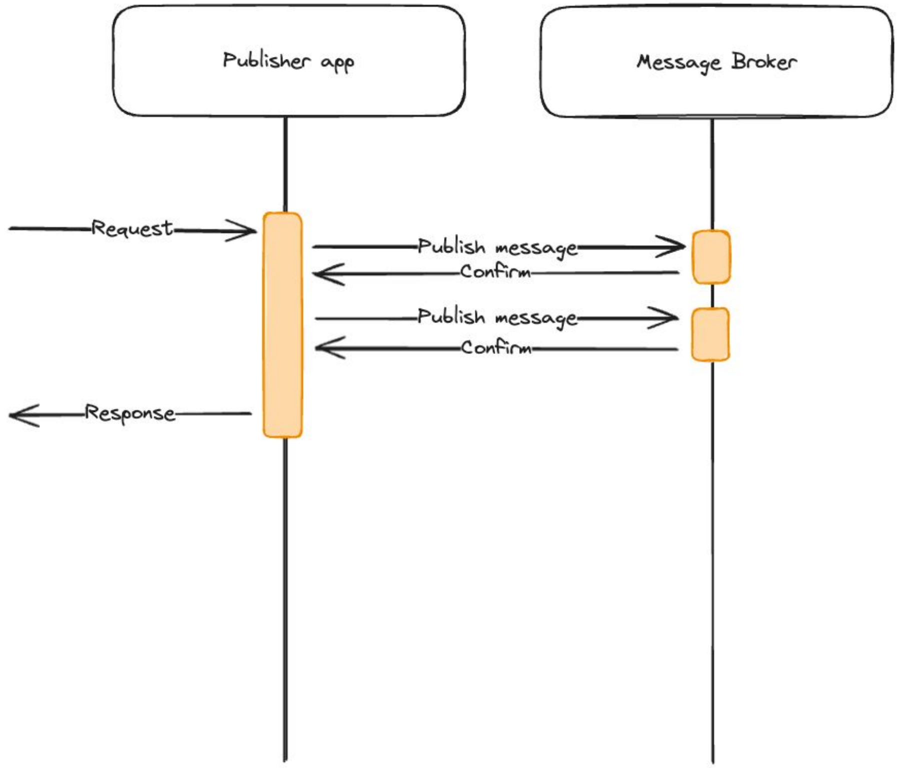
Broker in the middle



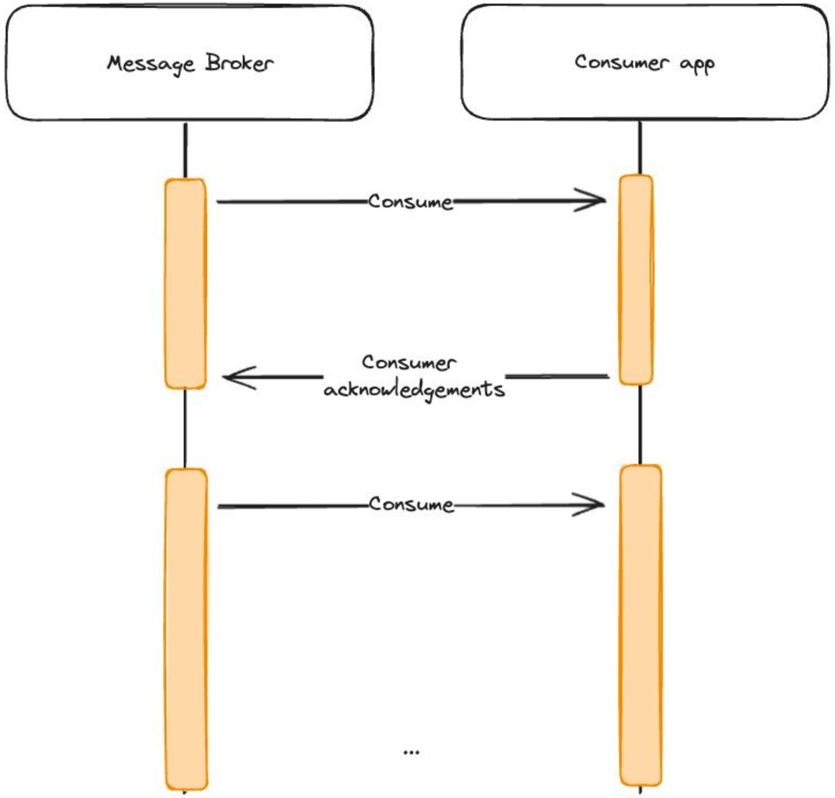
Publishers

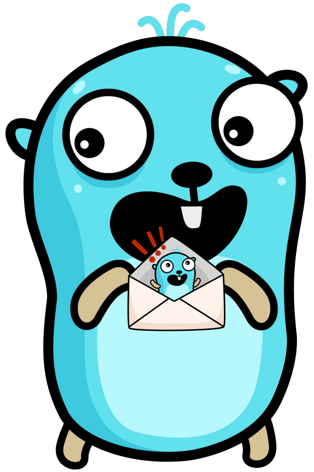
Consumers

Publish


Publisher app
Message Broker
Request
Publish message
Confirm
Publish message
Confirm
Response

Consume / Subscribe

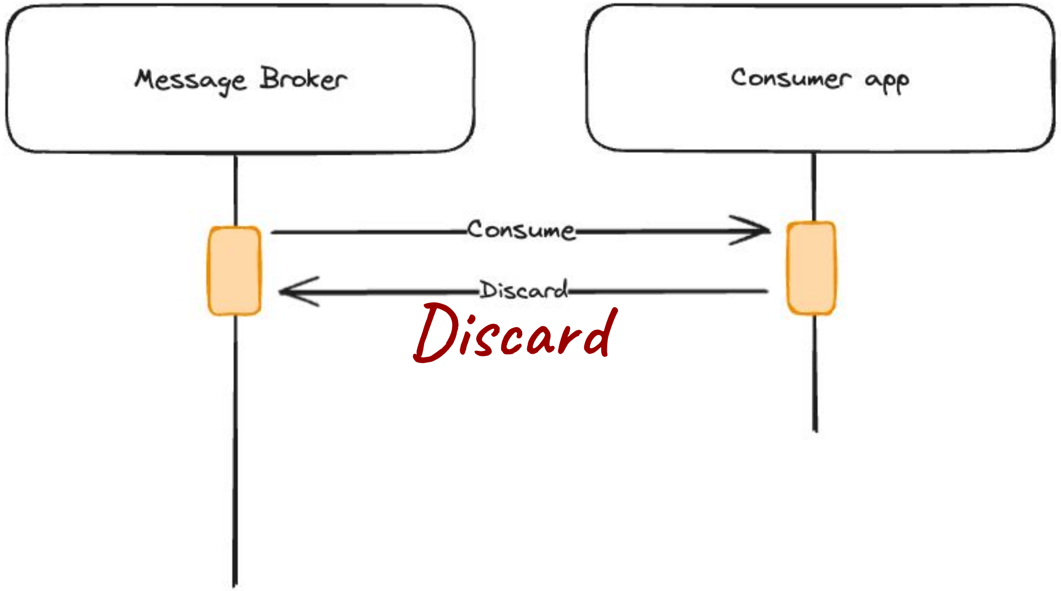




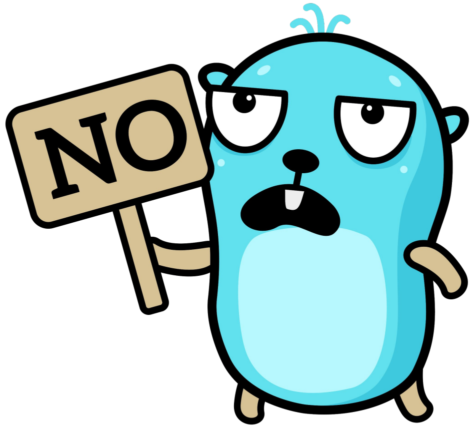
Consumer Acknowledgements

Discard = will not process the message

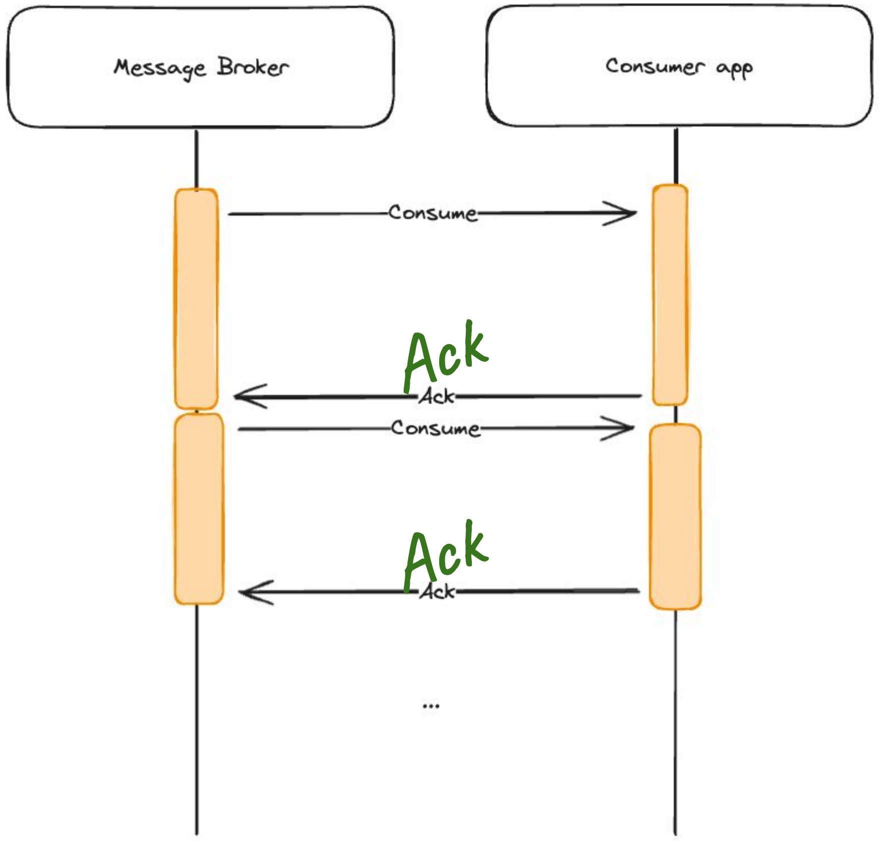
Reject

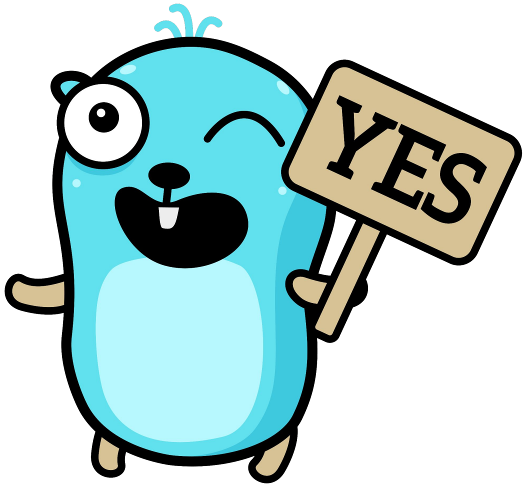


Discard

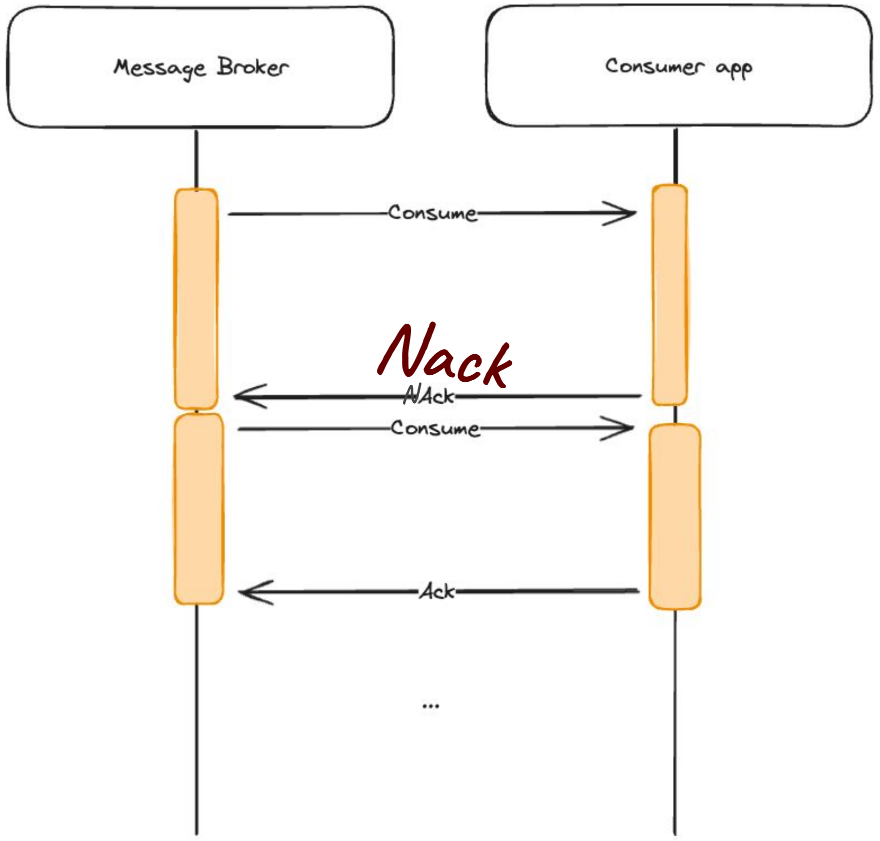


Acknowledge = I processed the message

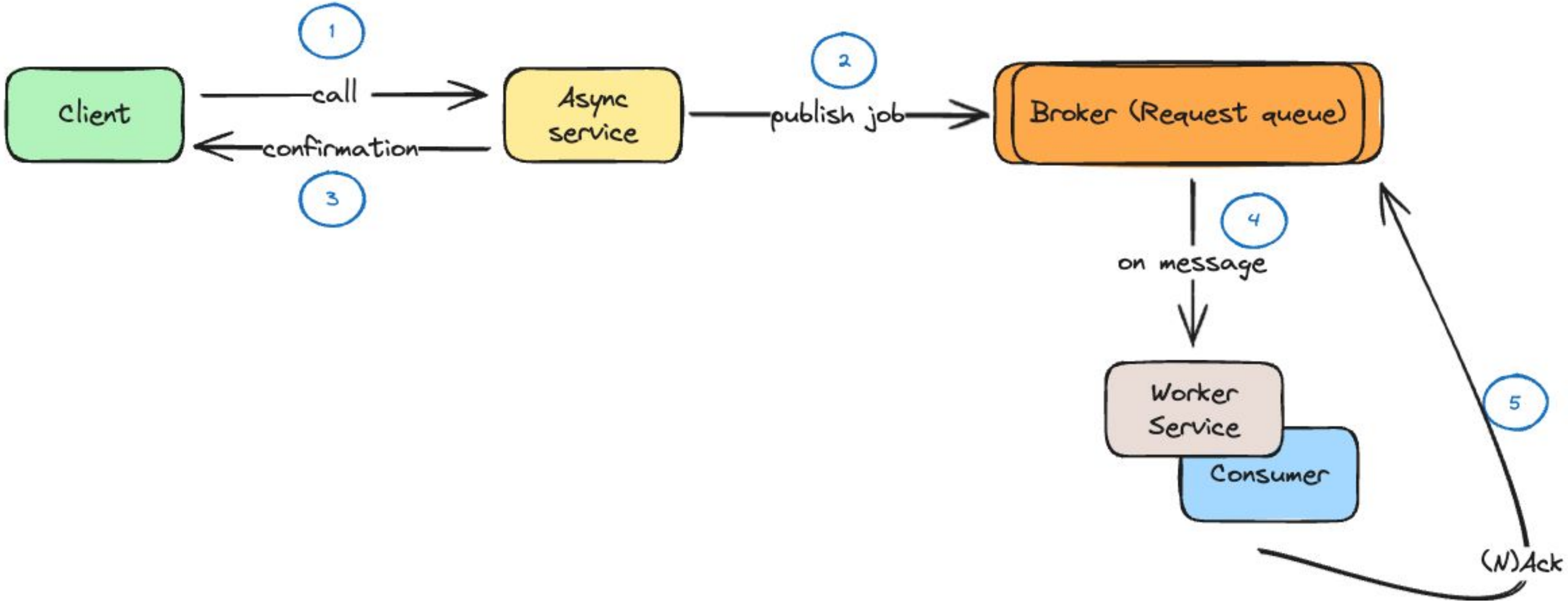




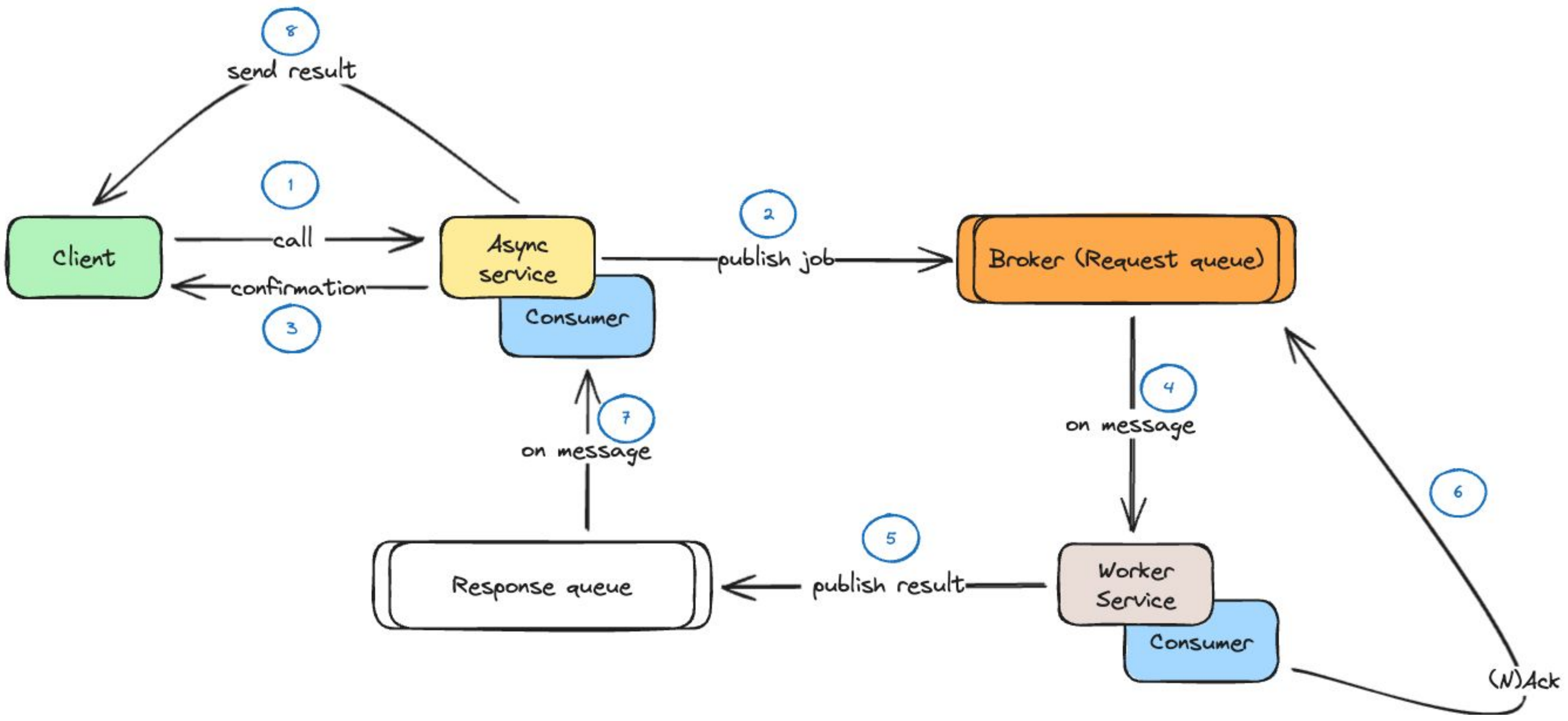
Negative Acknowledge = I couldn't process the message



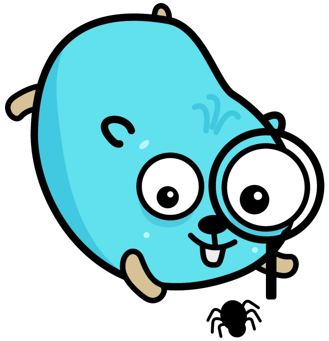
Request processing

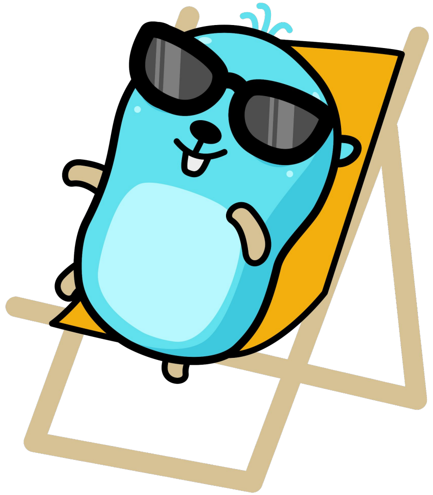


Request-response processing



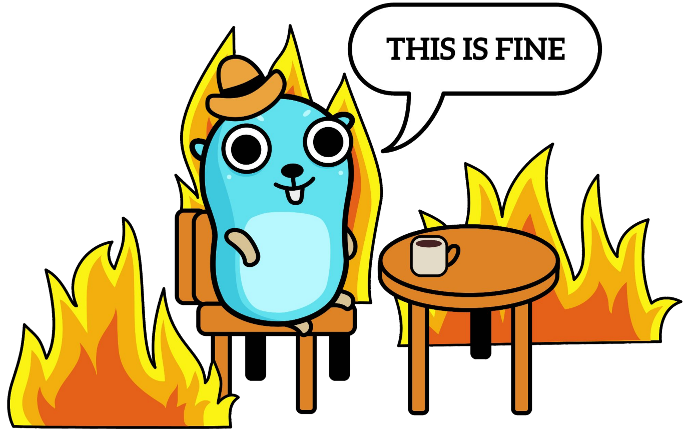
async/rabbit/printer

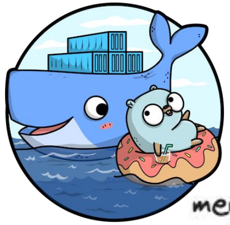




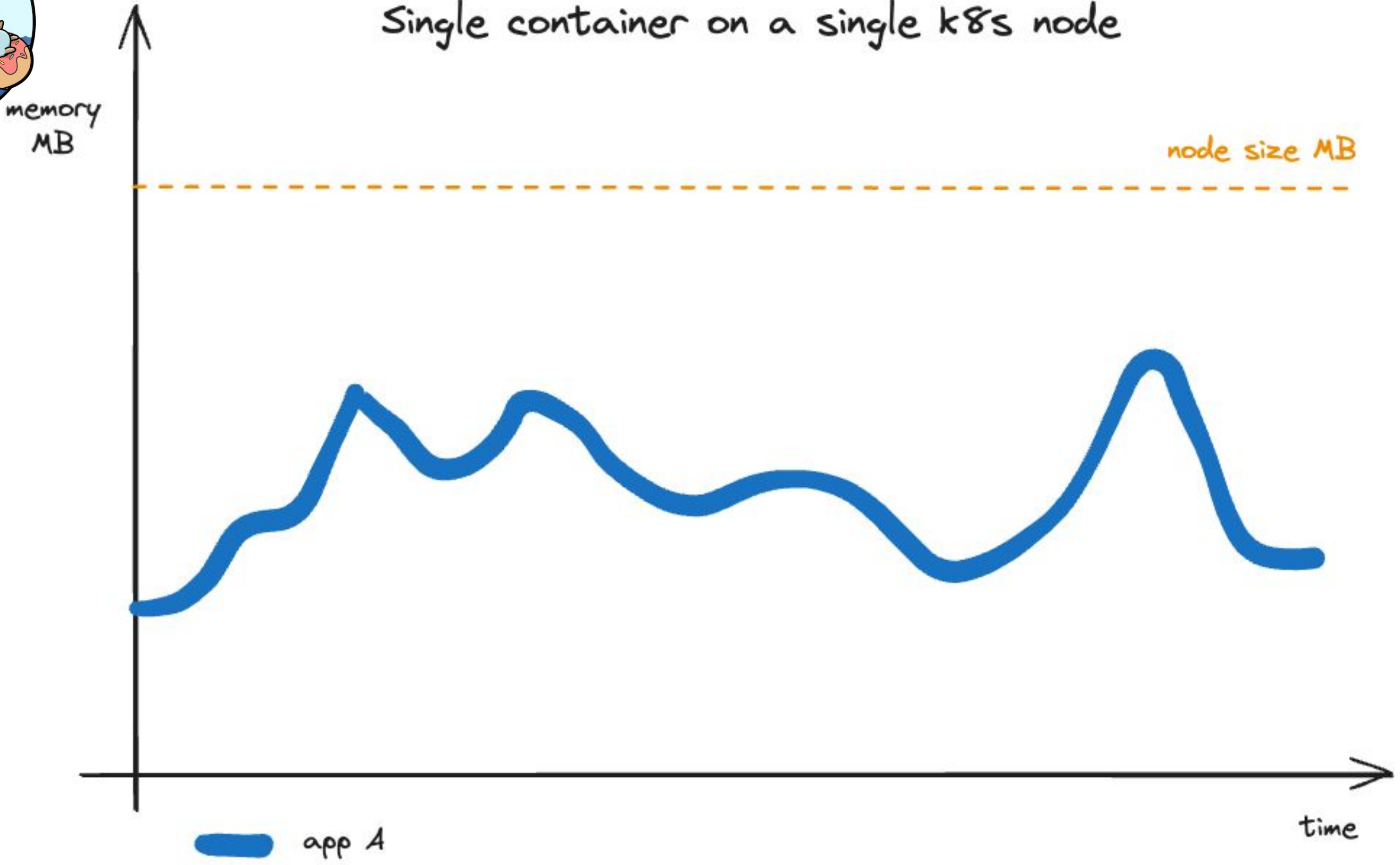
DEMO

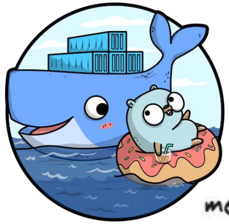
The theory is nice,
but the pitfalls reveal the practice



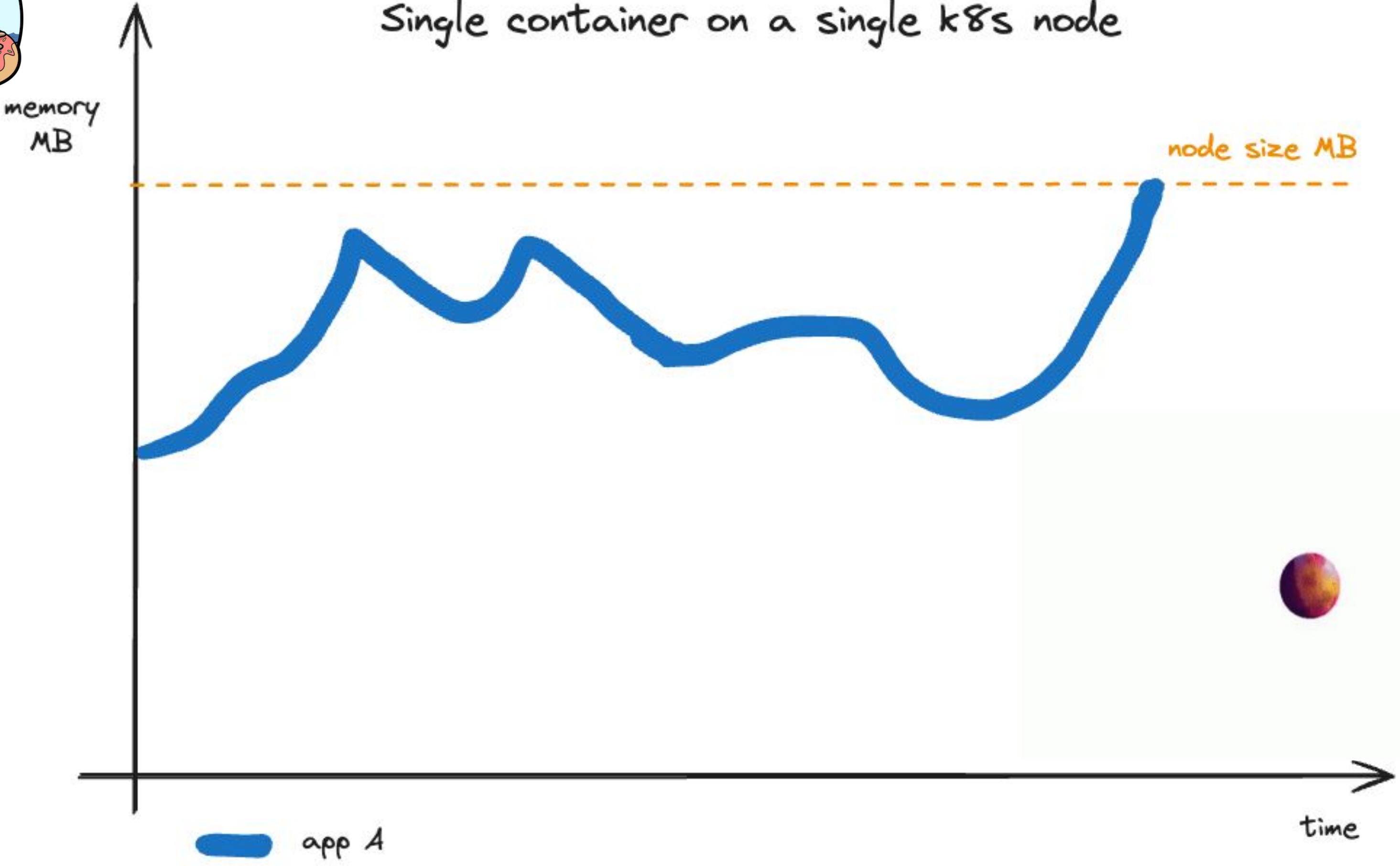


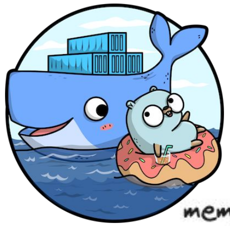
Single container on a single k8s node



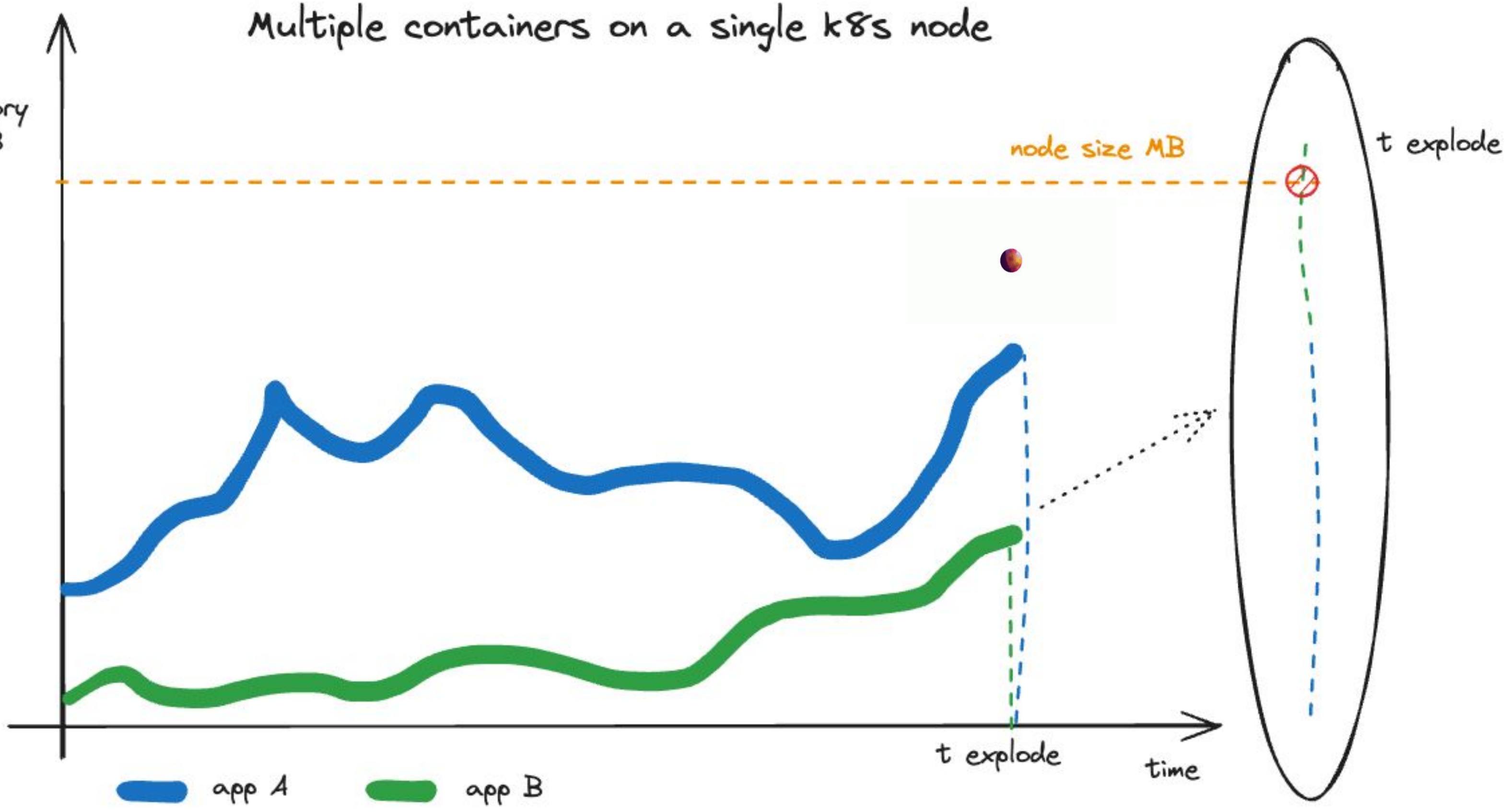


Single container on a single k8s node

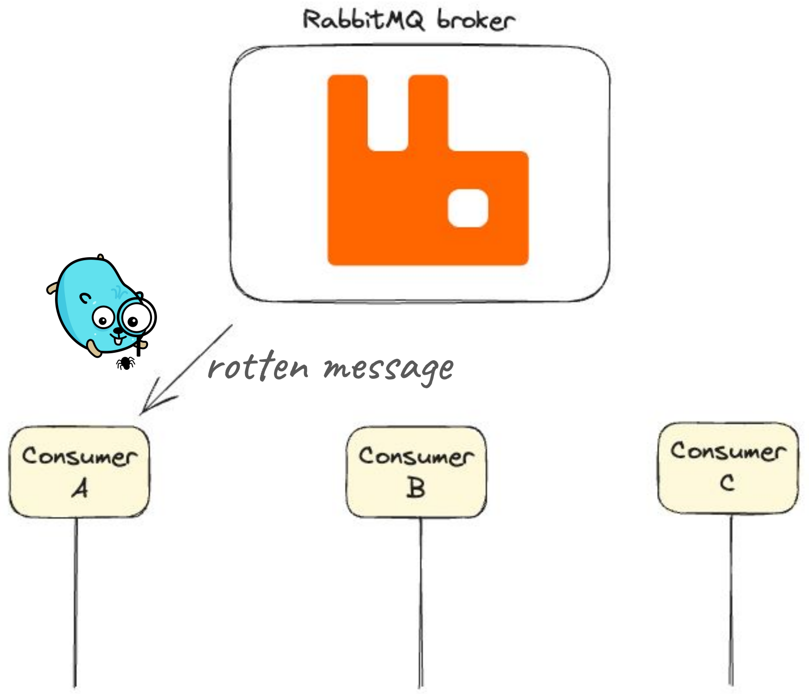




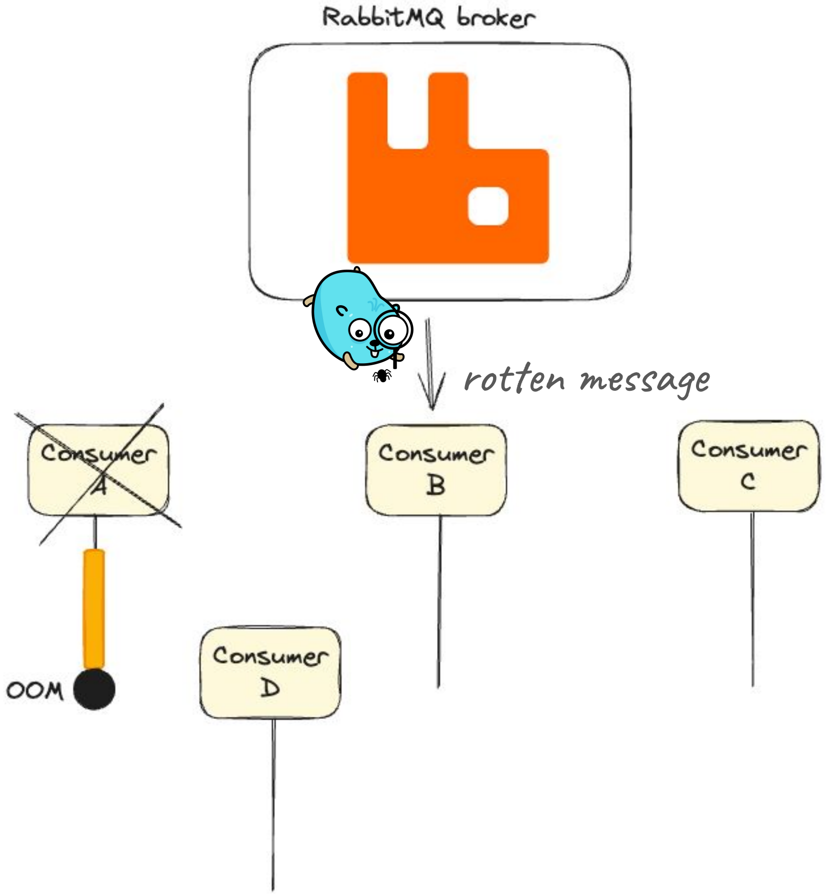
Multiple containers on a single k8s node



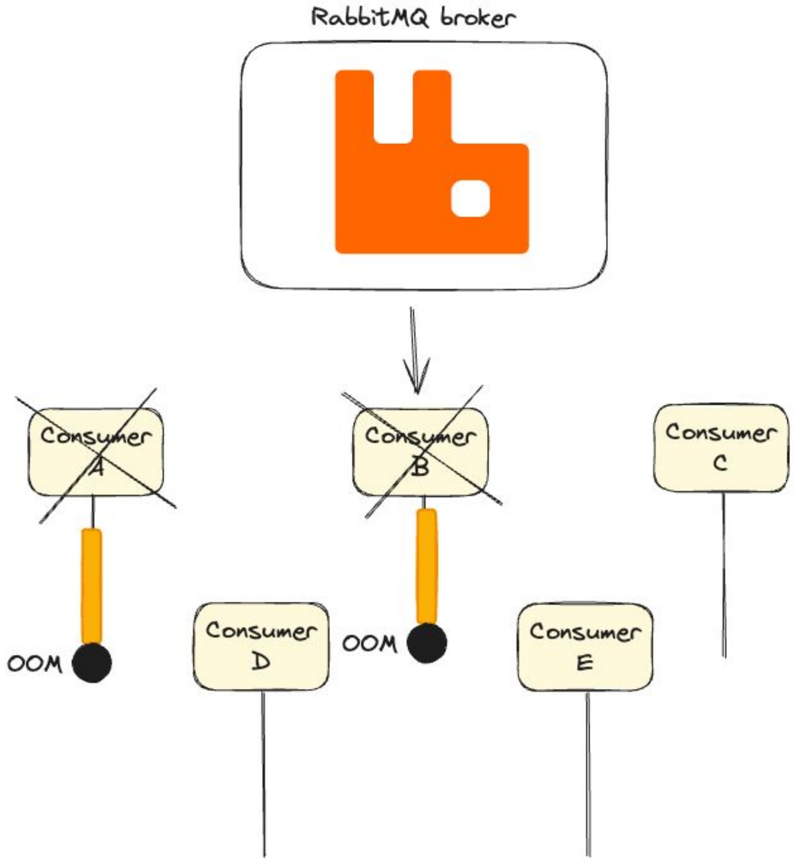
OOM


RabbitMQ broker
rotten message
Consumer A
Consumer B
Consumer C

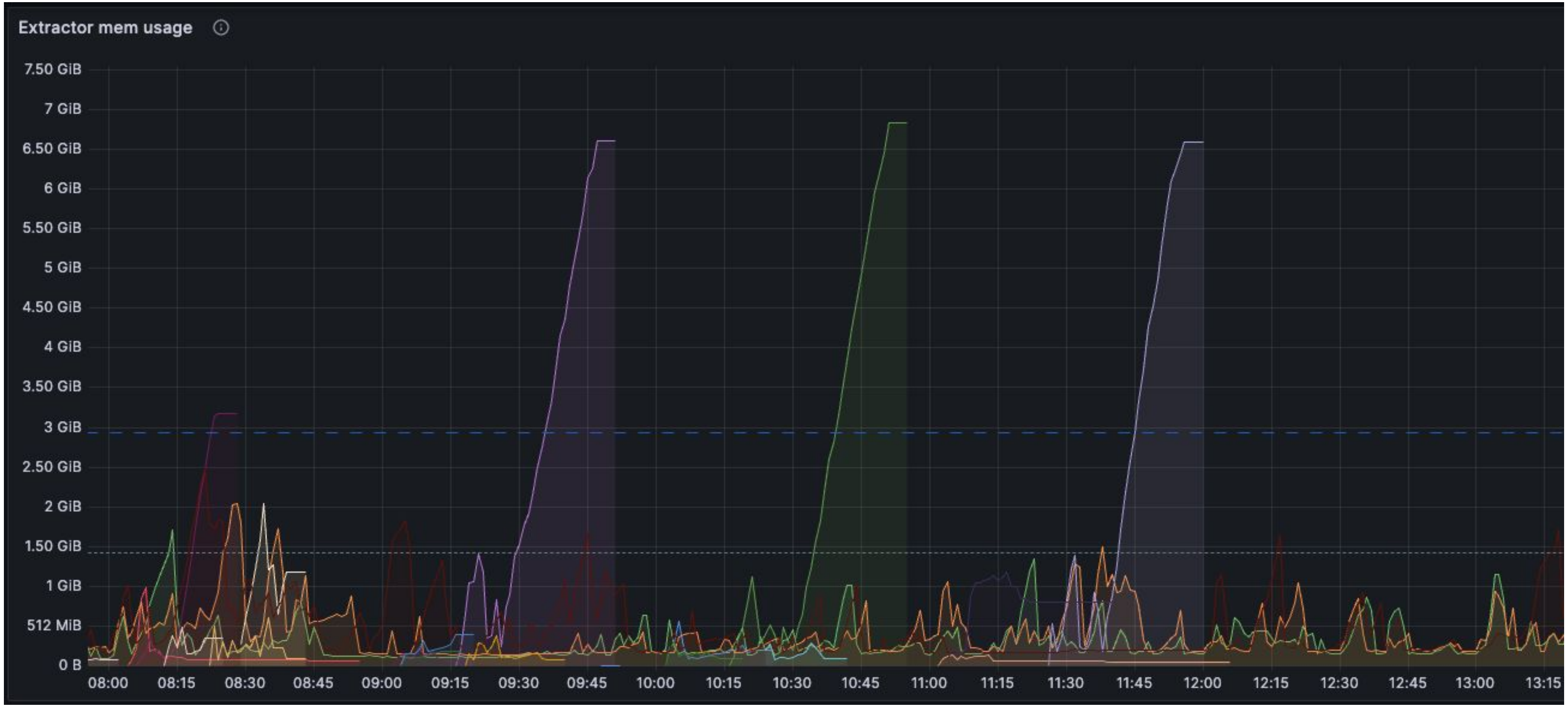
OOM


RabbitMQ broker
rotten message
~~Consumer A~~
Consumer B
Consumer C
Consumer D
OOM

OOM

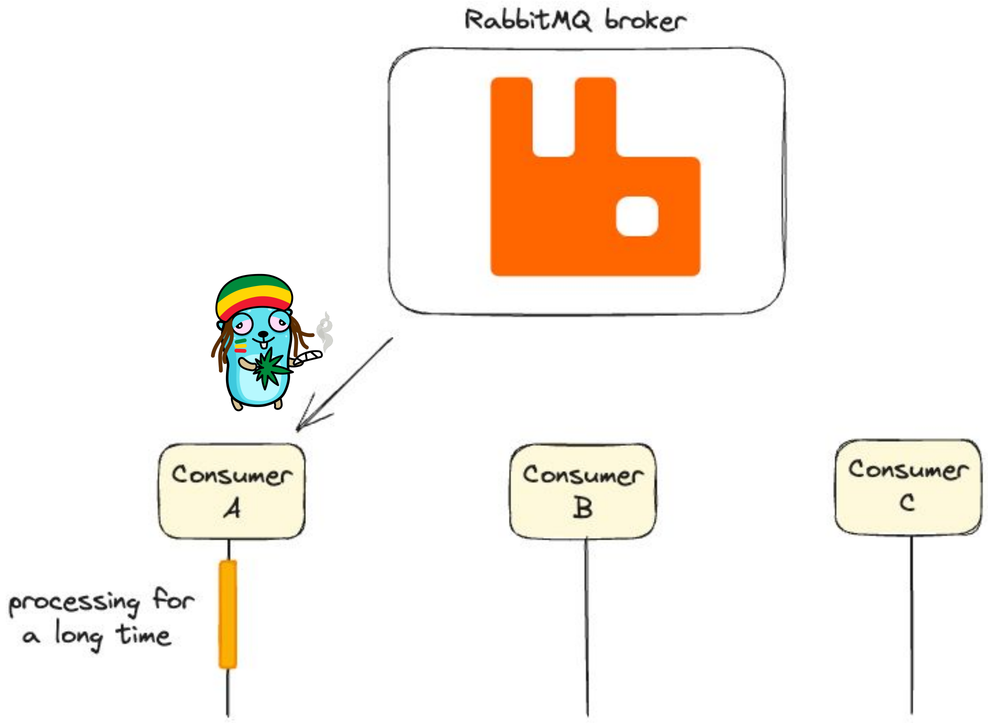


RabbitMQ broker
Consumer A
Consumer B
Consumer C
OOM
Consumer D
OOM
Consumer E

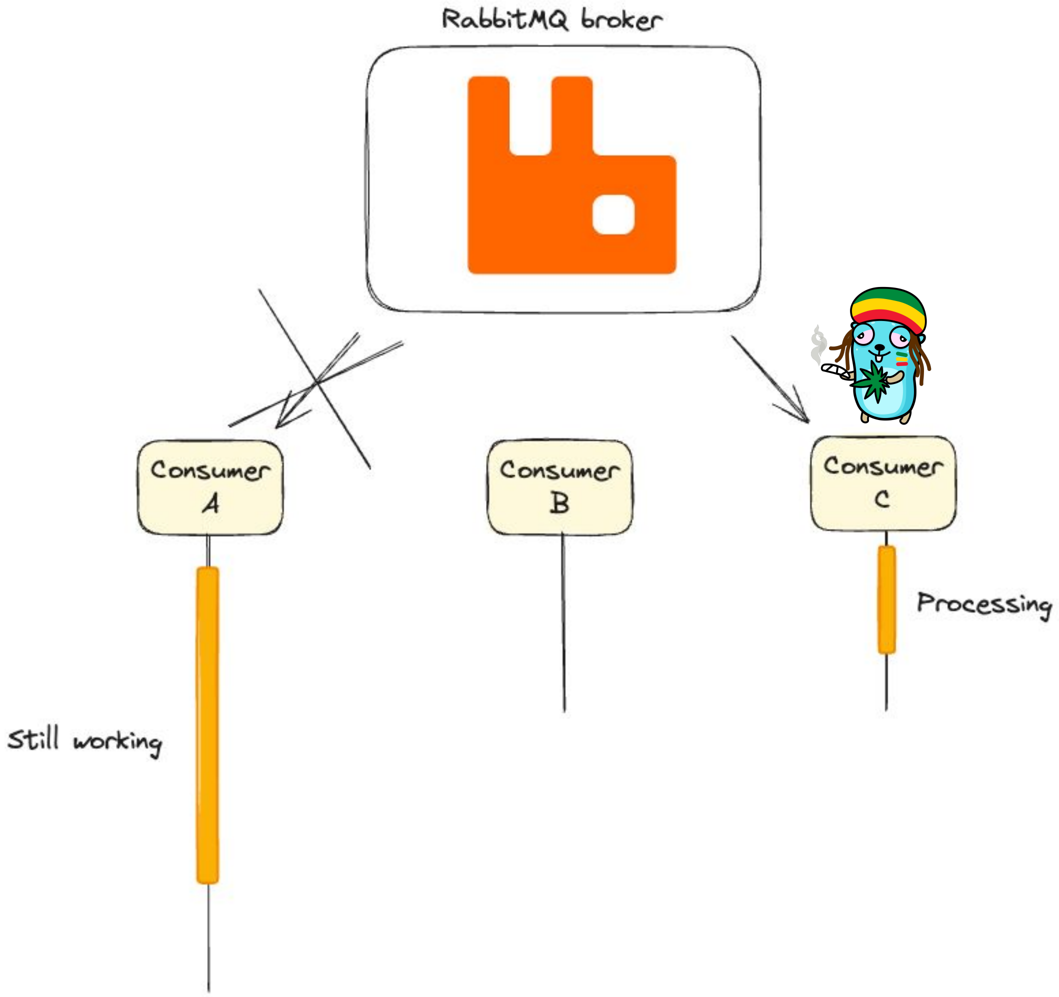

Extractor mem usage
7.50 GiB
7 GiB
6.50 GiB
6 GiB
5.50 GiB
5 GiB
4.50 GiB
4 GiB
3.50 GiB
3 GiB
2.50 GiB
2 GiB
1.50 GiB
1 GiB
512 MiB
0 B
08:00
08:15
08:30
08:45
09:00
09:15
09:30
09:45
10:00
10:15
10:30
10:45
11:00
11:15
11:30
11:45
12:00
12:15
12:30
12:45
13:00
13:15

Deadlock on Delivery Acknowledgement Timeout

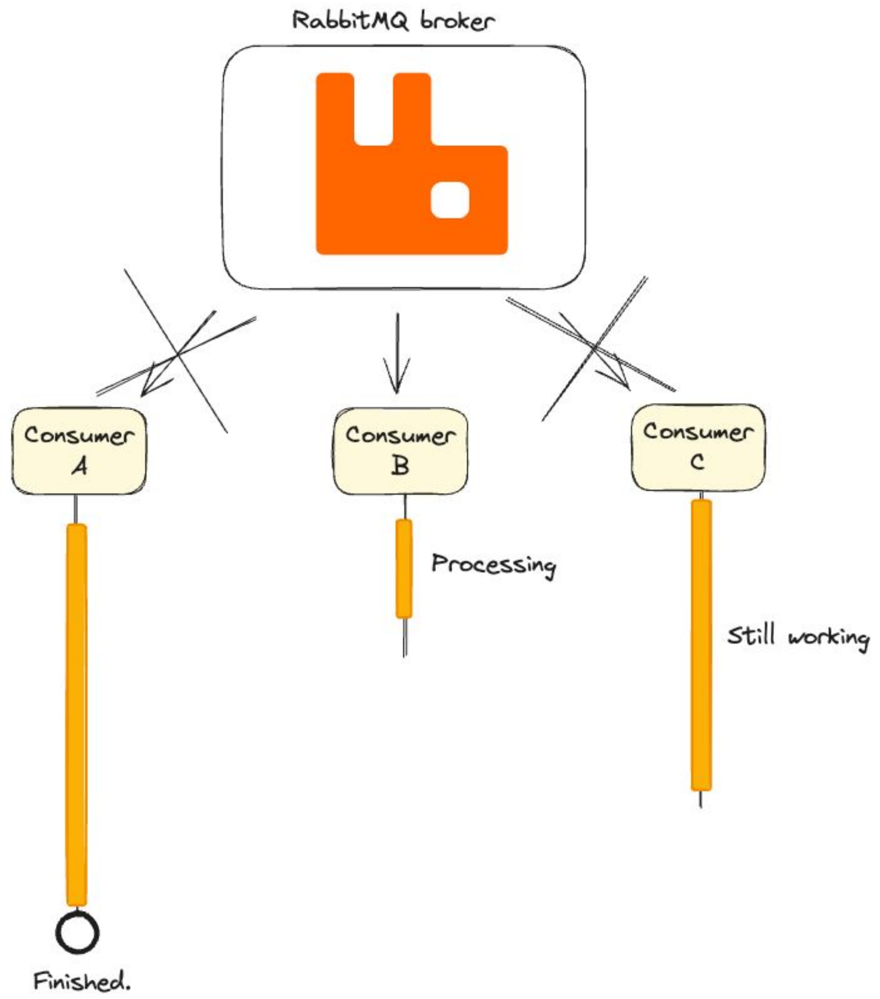
Ack timeout


RabbitMQ broker
Consumer A
Consumer B
Consumer C
processing for a long time

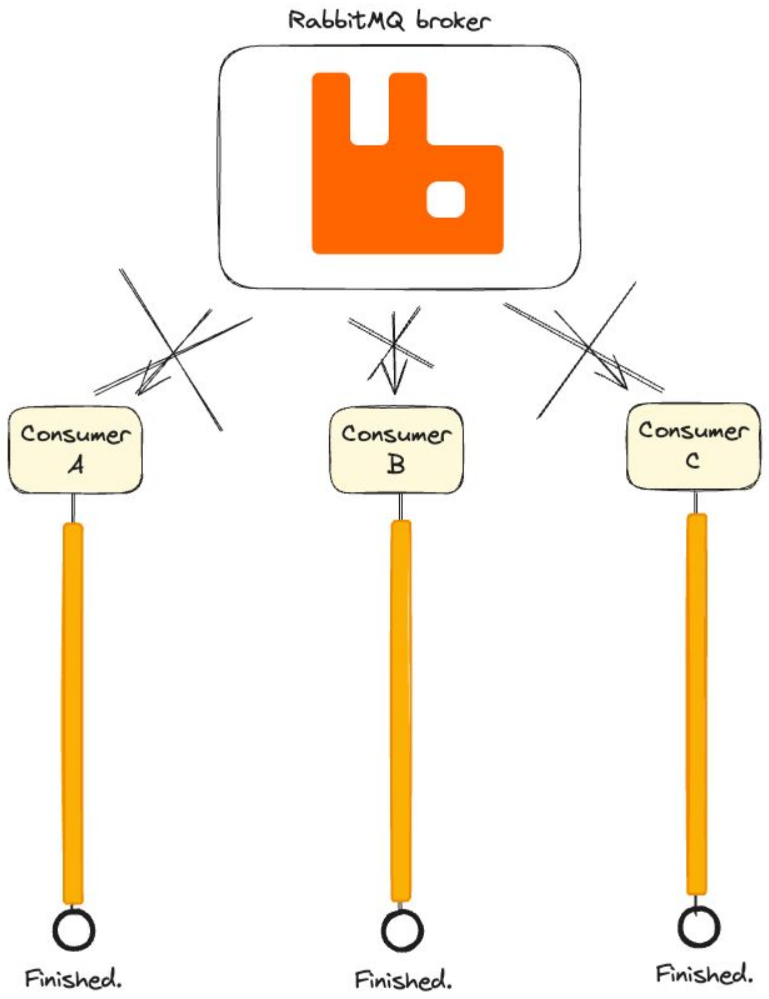
Ack timeout


RabbitMQ broker
Consumer A
Consumer B
Consumer C
Still working
Processing

Ack timeout


RabbitMQ broker
Consumer A
Consumer B
Consumer C
Processing
Still working
Finished.

Ack timeout


RabbitMQ broker
Consumer A
Consumer B
Consumer C
Finished.
Finished.
Finished.

go.dataddo.com/pgq

async/pgq

DEMO

Where is the asynchronicity helpful?

Where is the asynchronicity helpful?

Order Processing Systems

Handling e-commerce orders, where each order might involve multiple steps like payment processing, updating external systems, shipping and notifications, which can be done asynchronously.

Where is the asynchronicity helpful?

Order Processing Systems

Background Jobs and Task Queues

Tasks such as email sending, file processing, or generating reports, which can be offloaded to a queue and processed by worker services.

Where is the asynchronicity helpful?

Order Processing Systems

Background Jobs and Task Queues

Decoupling microservices

When microservices need to communicate without waiting for each other, allowing for better decoupling and improved resilience of the system.

Where is the asynchronicity helpful?

Order Processing Systems

Background Jobs and Task Queues

Decoupling microservices

Data Ingestion and ETL (Extract, Transform, Load) Pipelines

Ingesting large volumes of data from various sources and processing it can be done asynchronously to handle high throughput and avoid blocking.

Where is the asynchronicity helpful?

Order Processing Systems

Background Jobs and Task Queues

Decoupling microservices

Data Ingestion and ETL (Extract, Transform, Load) Pipelines

Load Balancing and Scaling

Distributing workloads across multiple servers or instances to handle varying load more effectively.

Where is the asynchronicity helpful?

Order Processing Systems

Background Jobs and Task Queues

Decoupling microservices

Data Ingestion and ETL (Extract, Transform, Load) Pipelines

Load Balancing and Scaling

> Distributing workloads across multiple servers or instances to handle varying load more effectively.

Real time notifications, logging, event-driven architectures, ...

Disadvantages of async code

Increased complexity in logic & code

It is usually harder to debug. Flow of the program may not be intuitive.

Disadvantages of async code

Increased complexity in logic & code

It is usually harder to debug. Flow of the program may not be intuitive.

Harder testing and debugging

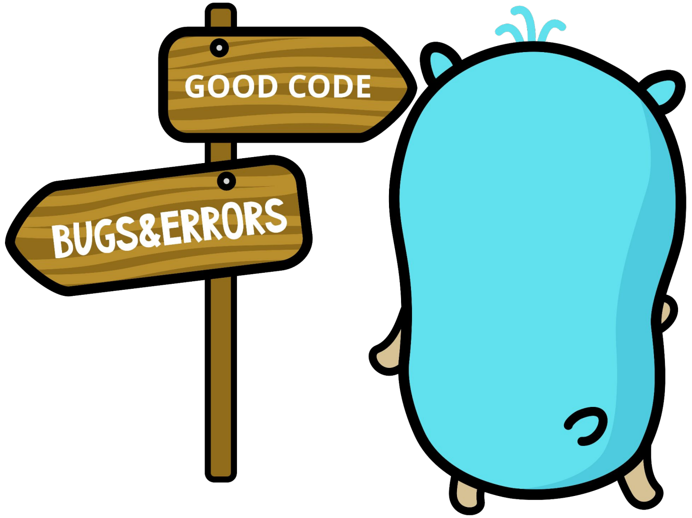
Reproducing errors is more difficult, error can happen only under some circccumns.

Disadvantages of async code

- Increased complexity in logic & code
  - It is usually harder to debug. Flow of the program may not be intuitive.
- Harder testing and debugging
  - Reproducing errors is more difficult, error can happen only under some circcumns.
- Race conditions, data consistency, deadlocks
  - Inconsistent When using shared resource => need for locks

Deciding sync or async

It depends.



More resources

- Messaging Patterns - Enterprise Integration Patterns

  www.enterpriseintegrationpatterns.com/patterns/messaging

- Watermill website   watermill.io
- Dataddo PGQ package, PGQ Youtube video   go.dataddo.com
- Gopher icons   github.com/MariaLetta/free-gophers-pack

I am nearly OOM,

but happy to answer your questions.

Q&A

Buffer slides

Following slides probably will not used at all.

Synchronous or Asynchronous?

Deciding sync or async

- Blocking vs. Non-blocking
  - If the result is needed immediately to proceed, a synchronous call makes sense

Deciding sync or async

Blocking vs. Non-blocking

If the result is needed immediately to proceed, a synchronous call makes sense

Responsiveness

In applications with user interfaces, synchronous jobs block the main thread.

Deciding sync or async

Blocking vs. Non-blocking

If the result is needed immediately to proceed, a synchronous call makes sense

Responsiveness

In applications with user interfaces, synchronous jobs block the main thread.

Control flow

Synchronous calls ensure the tasks are executed in the order.

Asynchr. calls can reduce the execution time, but their coordination may be tricky.

Deciding sync or async

Blocking vs. Non-blocking

If the result is needed immediately to proceed, a synchronous call makes sense

Responsiveness

In applications with user interfaces, synchronous jobs block the main thread.

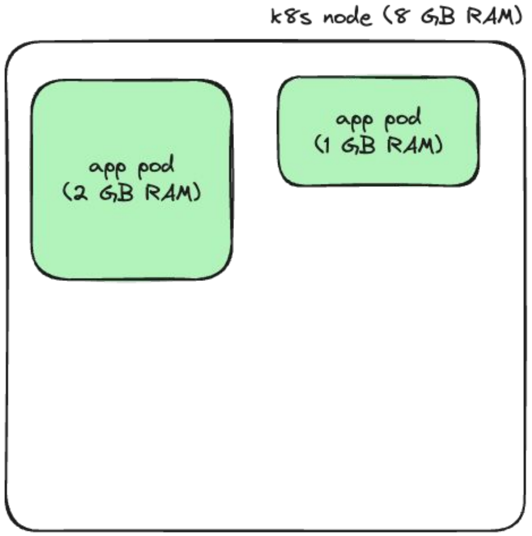
Control flow

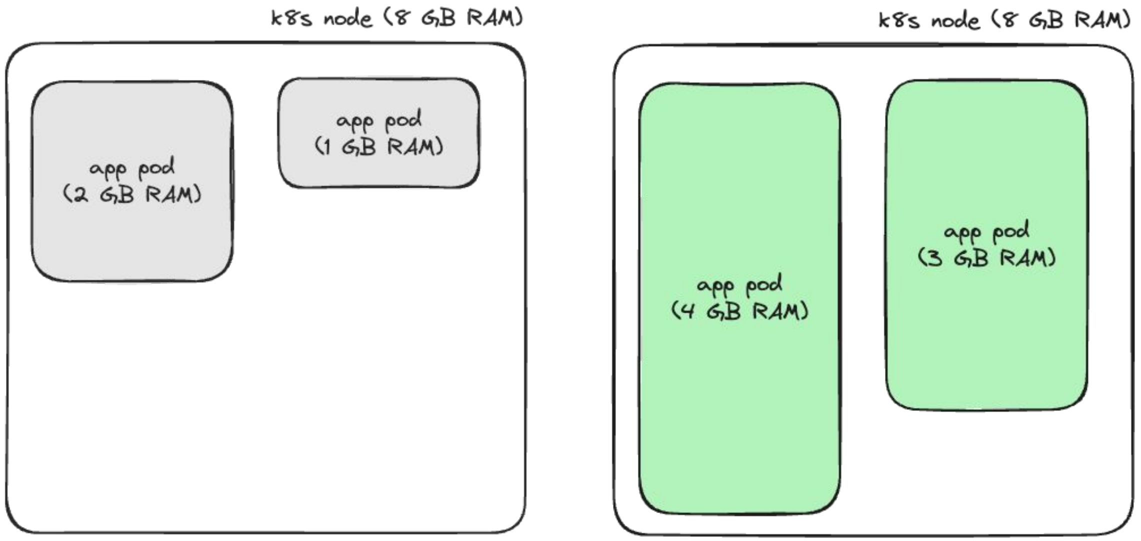
Synchronous calls ensure the tasks are executed in the order.

Asynchr. calls can reduce the execution time, but their coordination may be tricky.

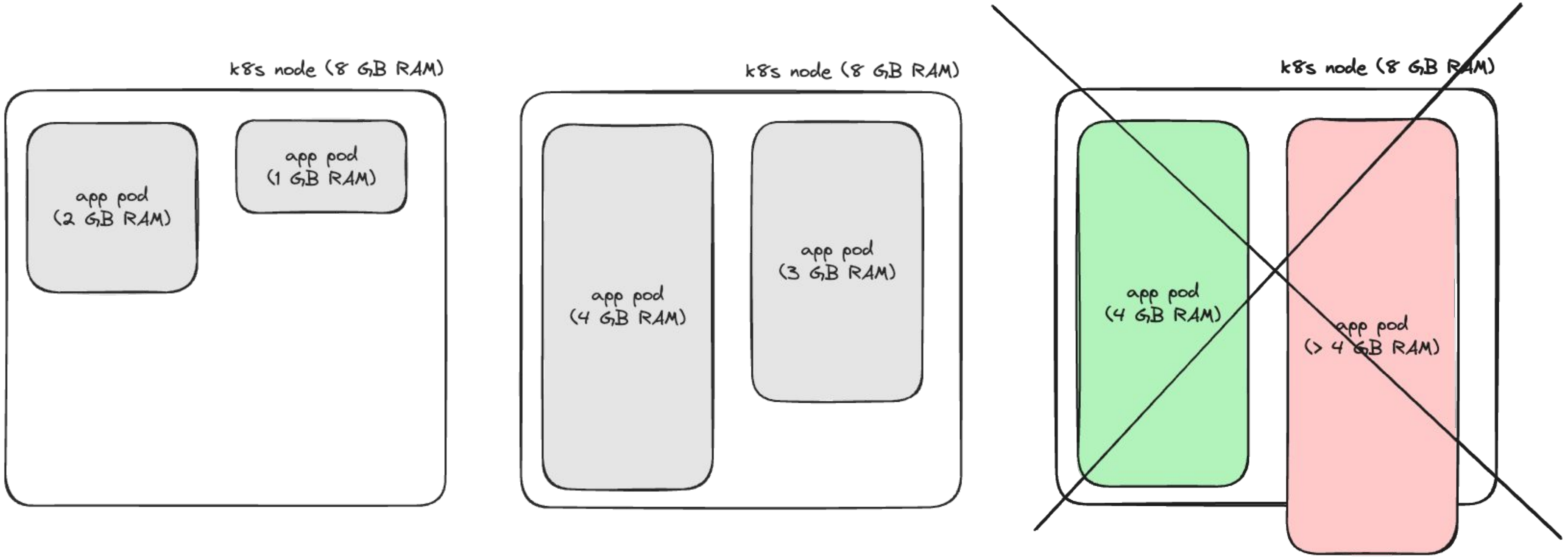
Scalability

Asynchronous processing can help scale more effectively (free resources)


k8s node (8 GB RAM)
app pod (2 GB RAM)
app pod (1 GB RAM)


k8s node (8 GB RAM)
app pod (2 GB RAM)
app pod (1 GB RAM)
k8s node (8 GB RAM)
app pod (4 GB RAM)
app pod (3 GB RAM)

OOM


k8s node (8 GB RAM)
app pod (2 GB RAM)
app pod (1 GB RAM)
k8s node (8 GB RAM)
app pod (4 GB RAM)
app pod (3 GB RAM)
k8s node (8 GB RAM)
app pod (4 GB RAM)
app pod (> 4 GB RAM)