In the cloud

message broker is an implementation detail

exactly-once.github.io

twitter.com/SzymonPobiega

linkedin.com/in/SzymonPobiega

Domain-Driven Design

Domain-Driven Design
Event Sourcing

Domain-Driven Design

Event Sourcing

Message delivery & processing

Broker?

But why?

What defines architecture?

What defines architecture?

Availability of capital

What defines architecture?

Availability of capital

Availability of workers

What defines architecture?

Availability of capital

Availability of workers

Management methodologies

Microservices is a product of

Microservices is a product of

Abundance of capital

Microservices is a product of

Abundance of capital

Shortage of software engineers

Microservices is a product of

Abundance of capital

Shortage of software engineers

Decentralized management

Designs the minimal thing that works

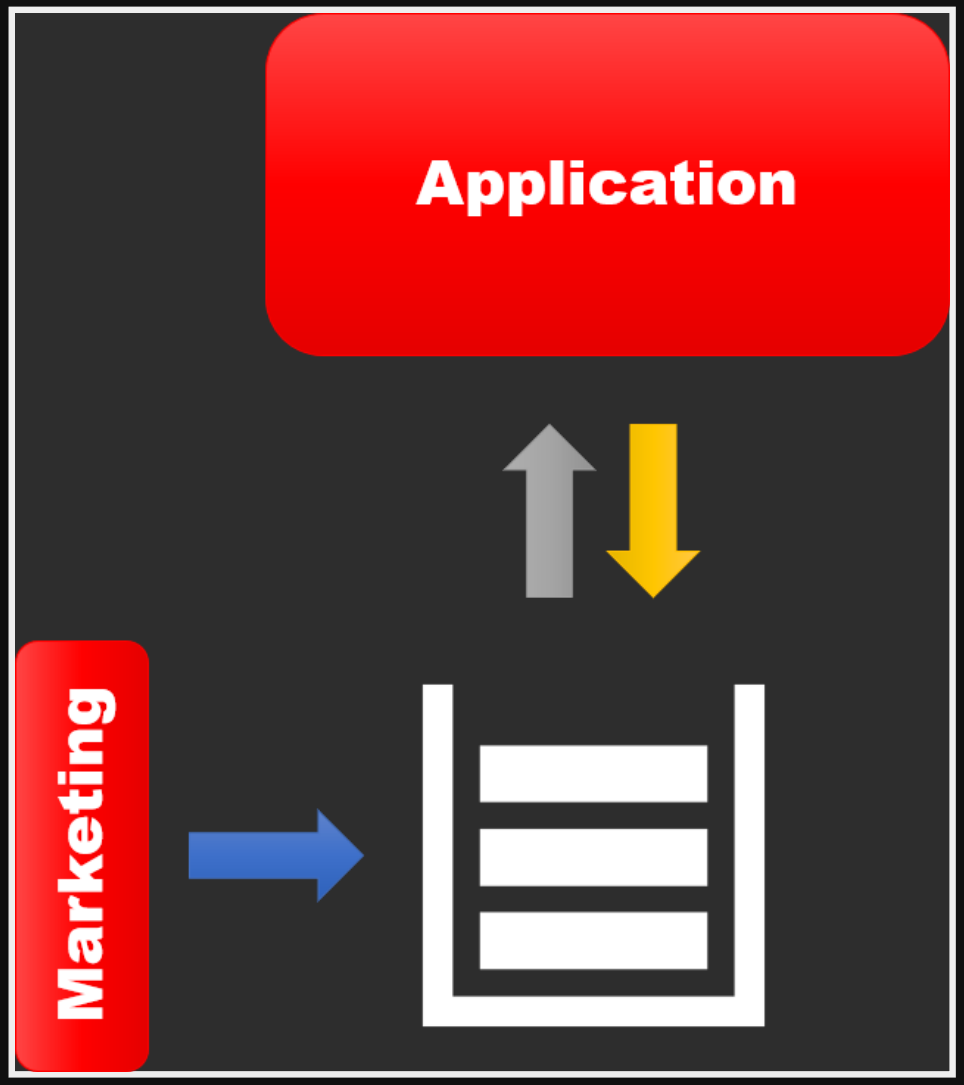
Designs the minimal thing that works

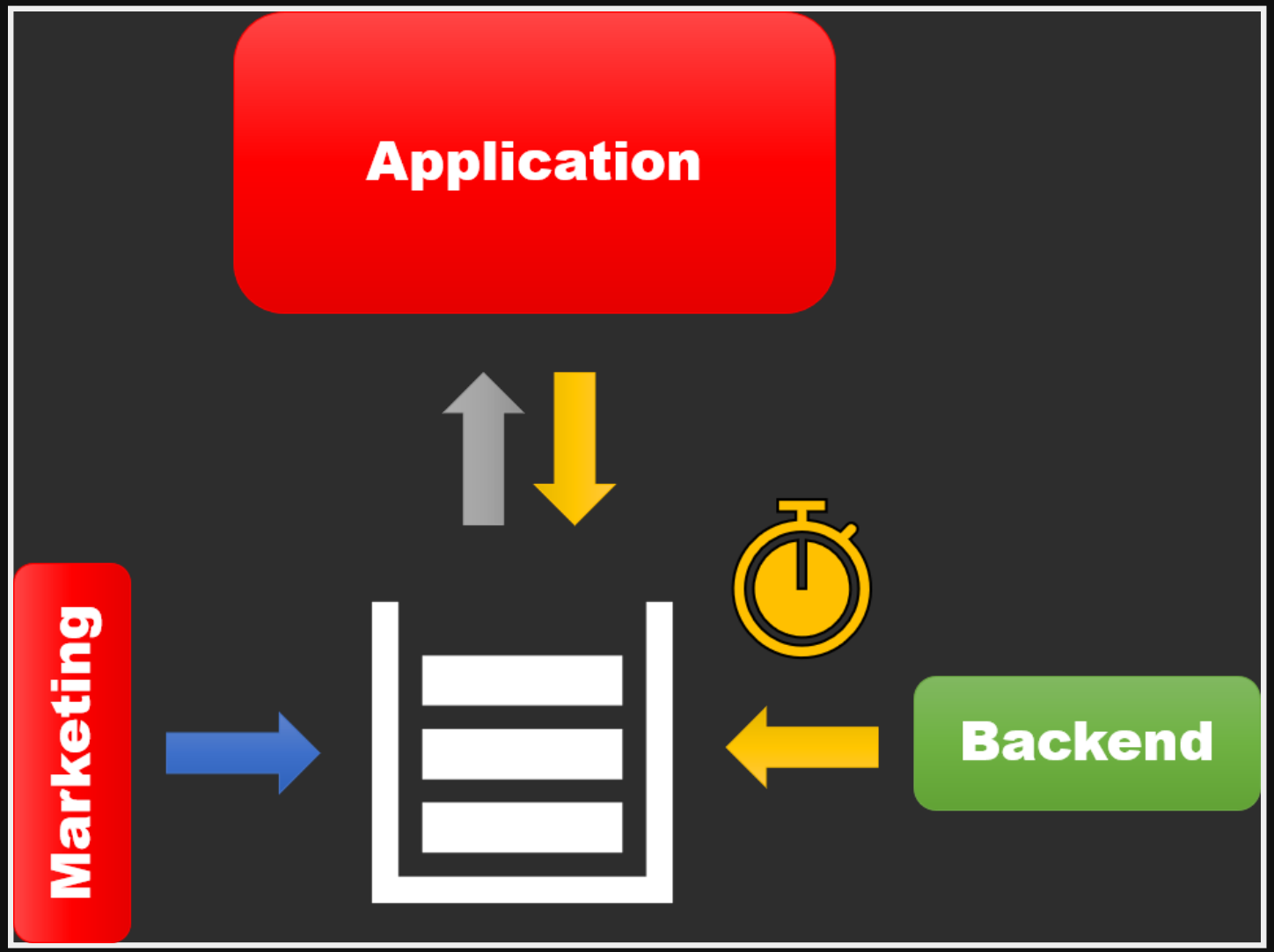
Optimizes for the lowest number of moving parts

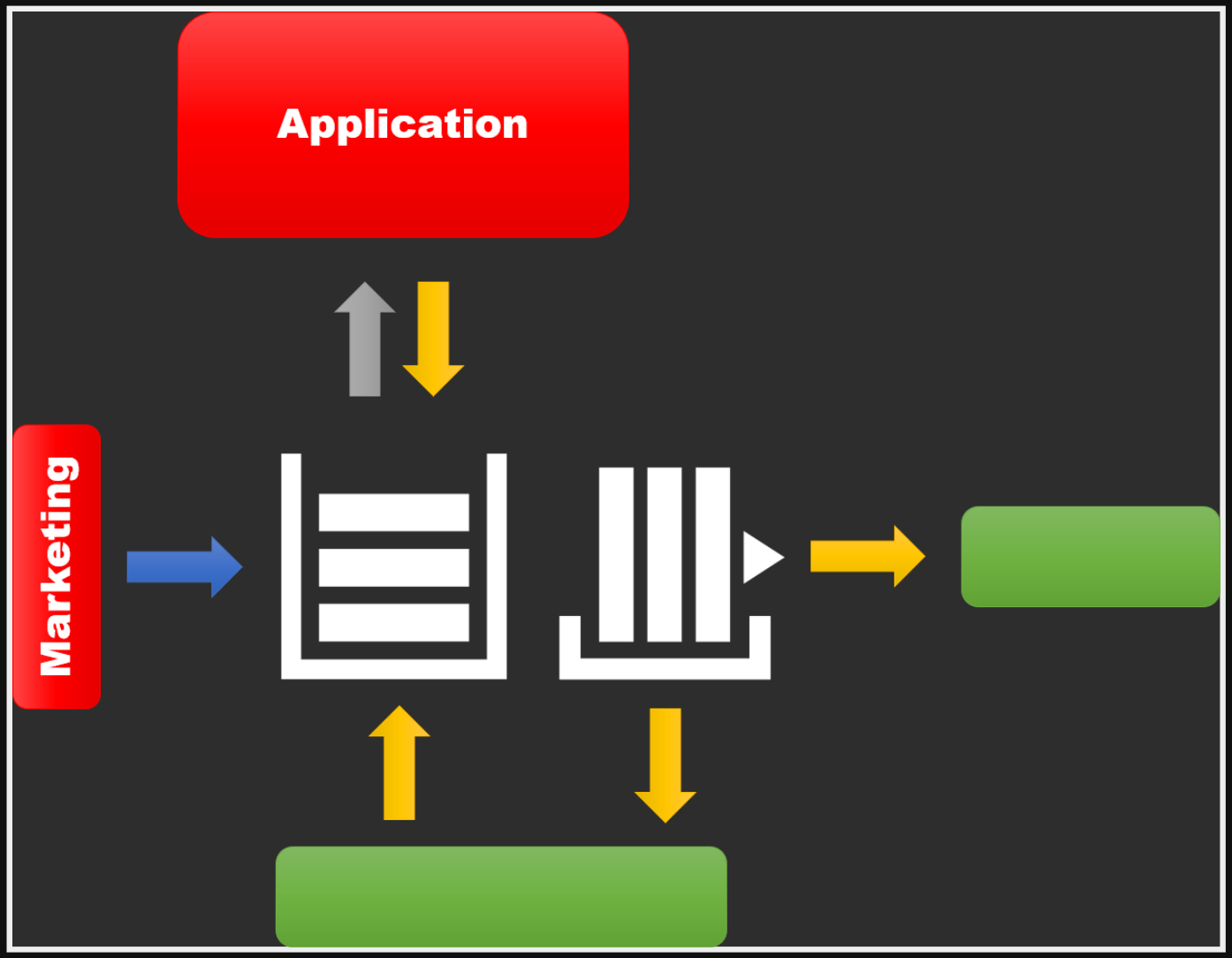
Designs the minimal thing that works

Optimizes for the lowest number of moving parts

Adds complexity as needed


Application
Marketing


Application
Marketing
Backend


Application
Marketing

Uses small scale building blocks

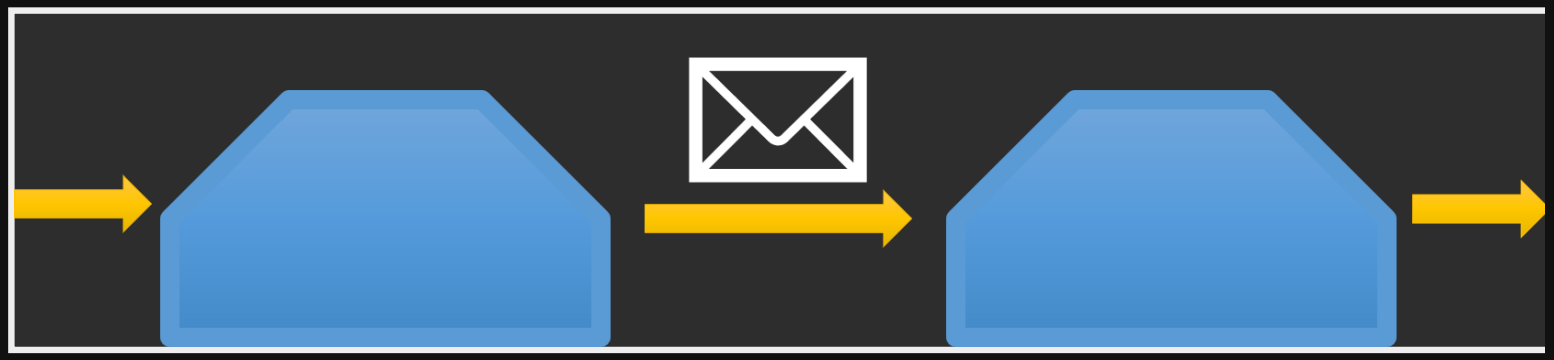
Uses small scale building blocks

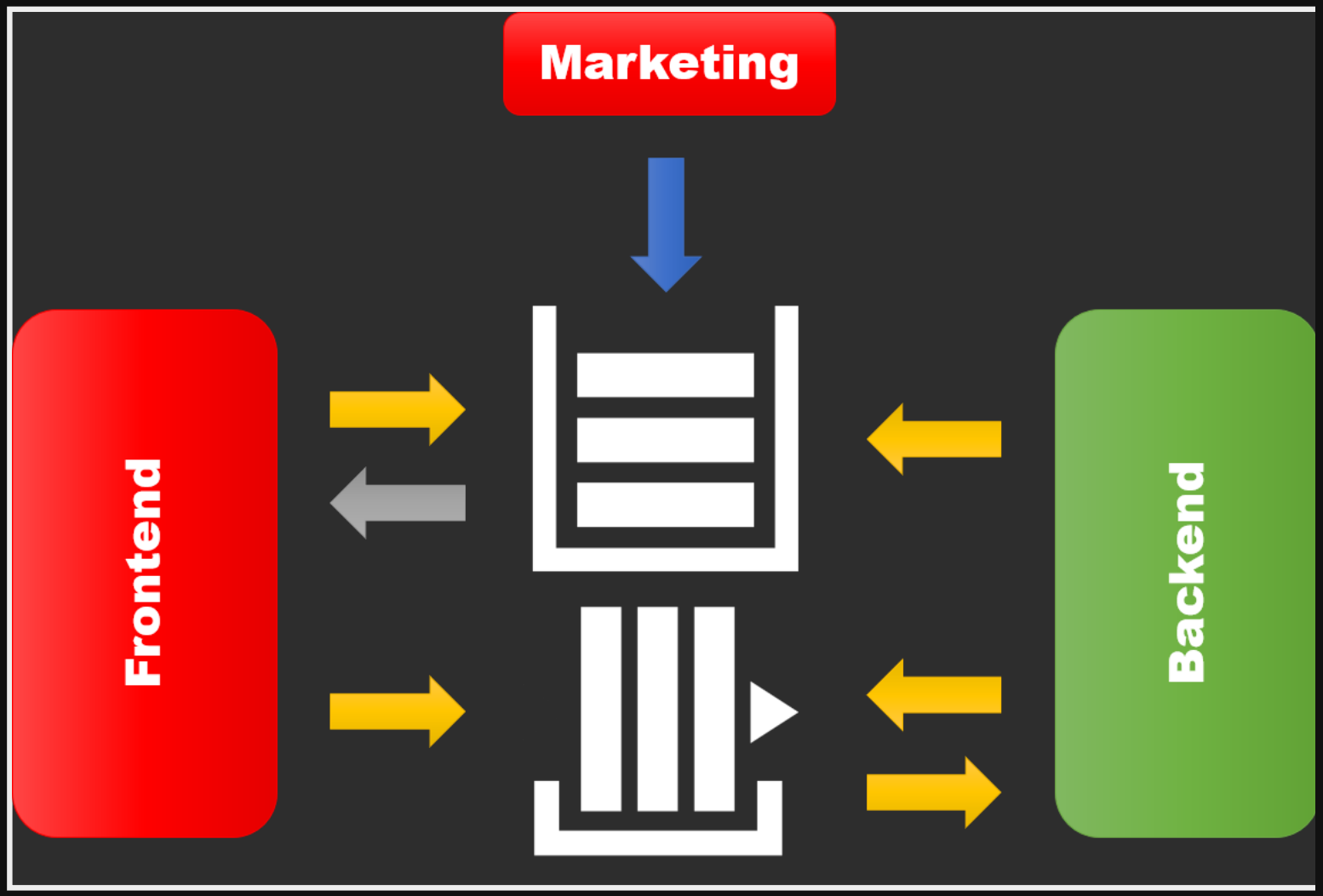
Ignores high-level components

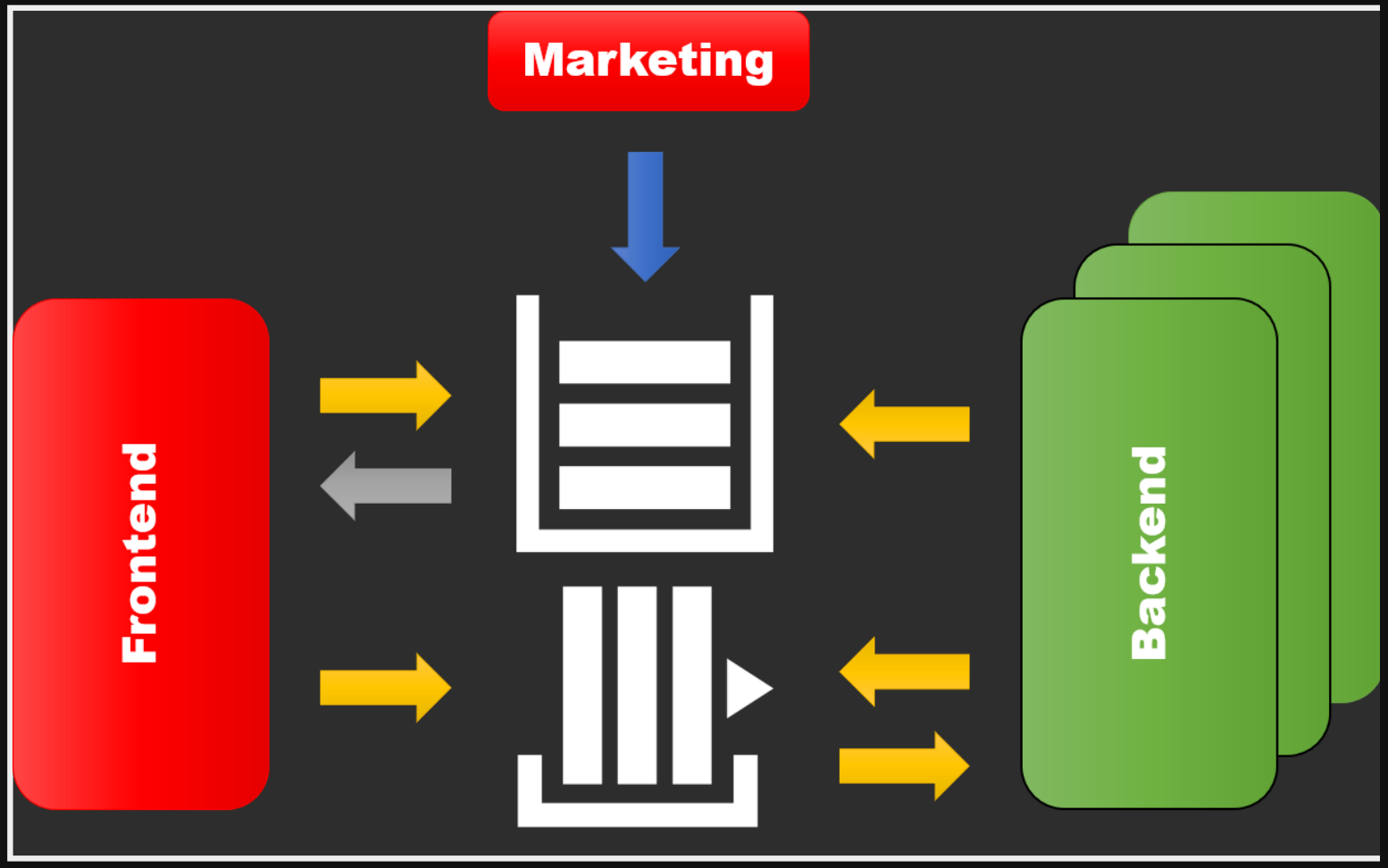
Uses small scale building blocks

Ignores high-level components

Optimizes for uniformity



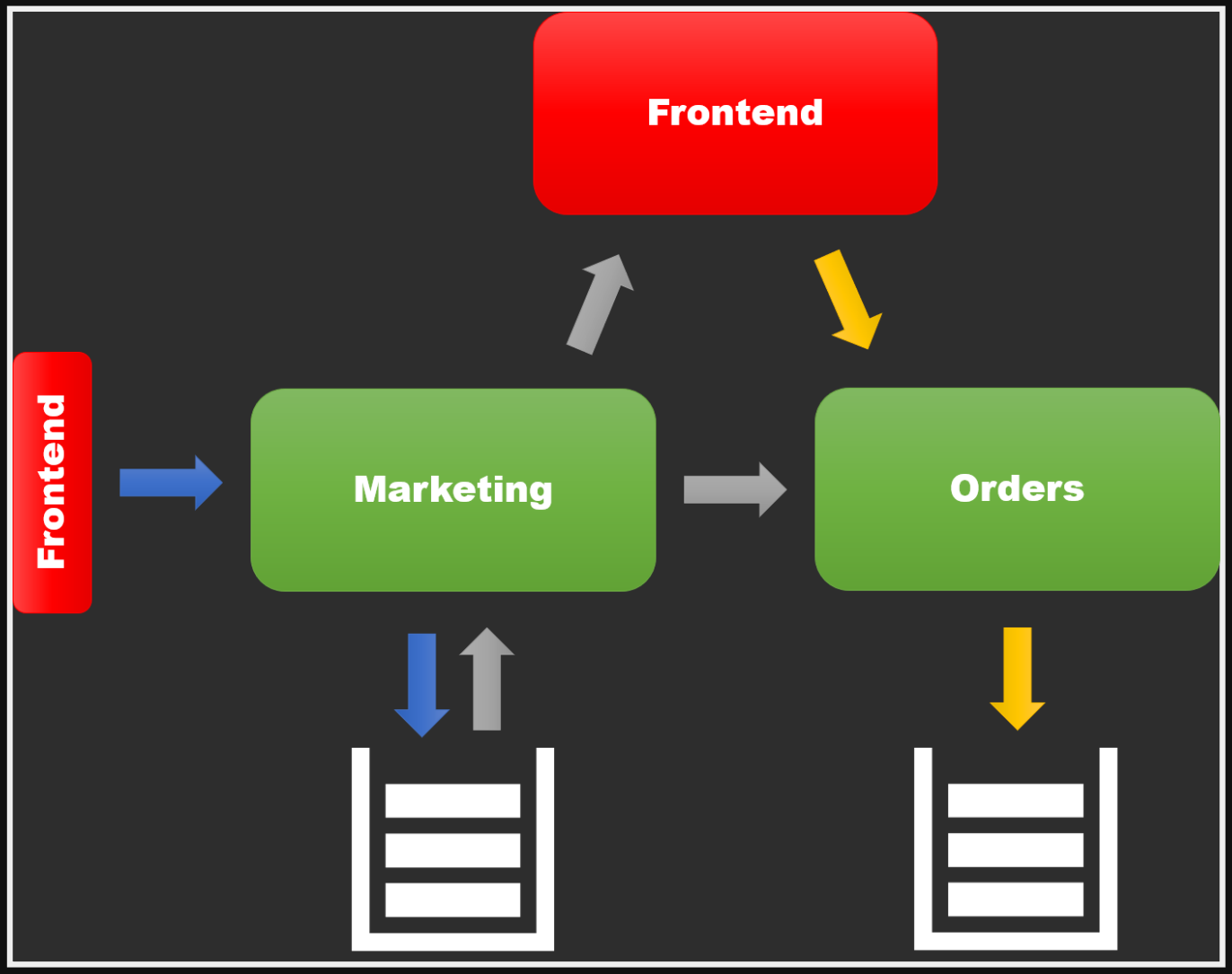

Marketing
Frontend
Backend

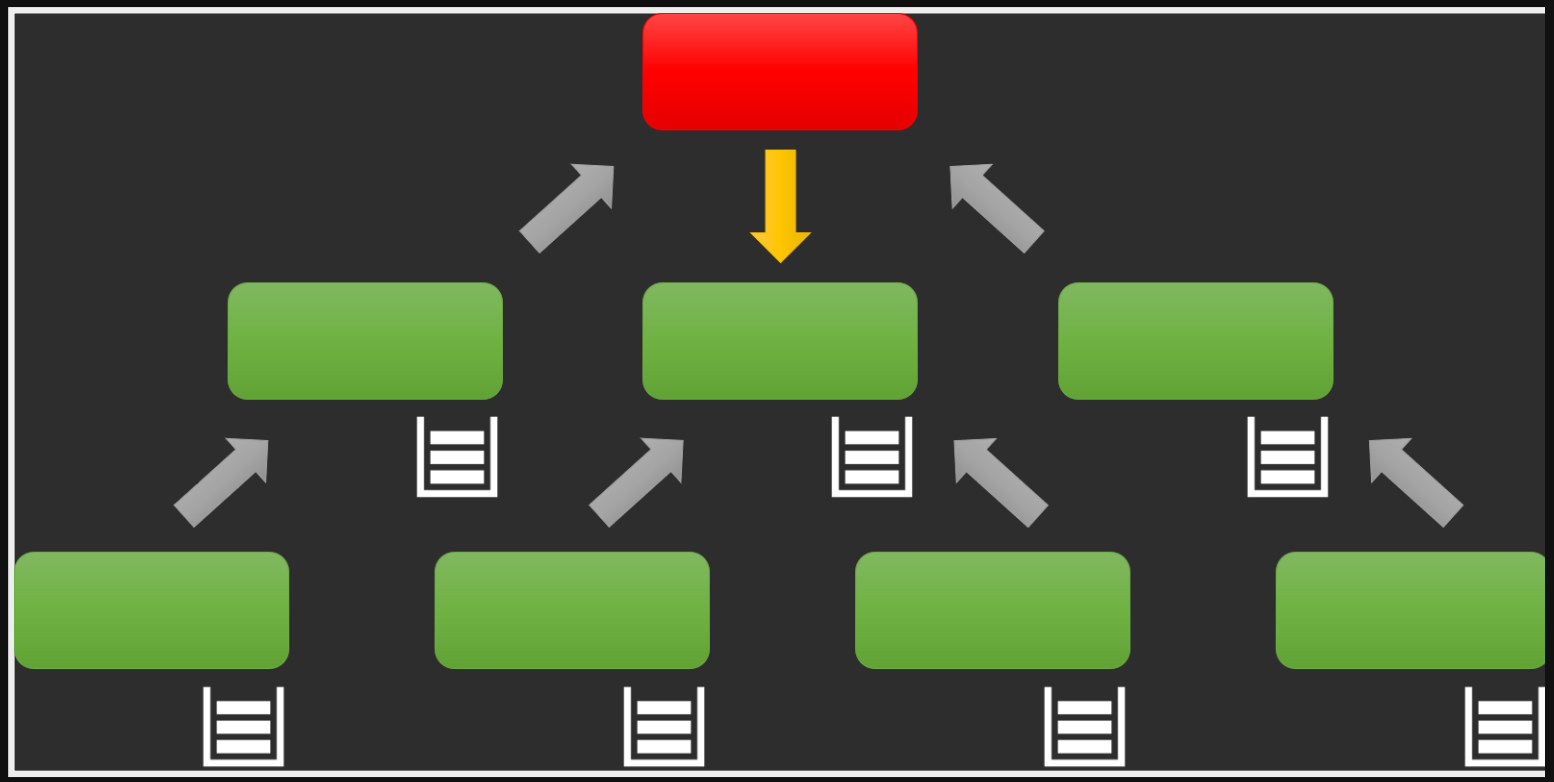

Marketing
Frontend
Backend

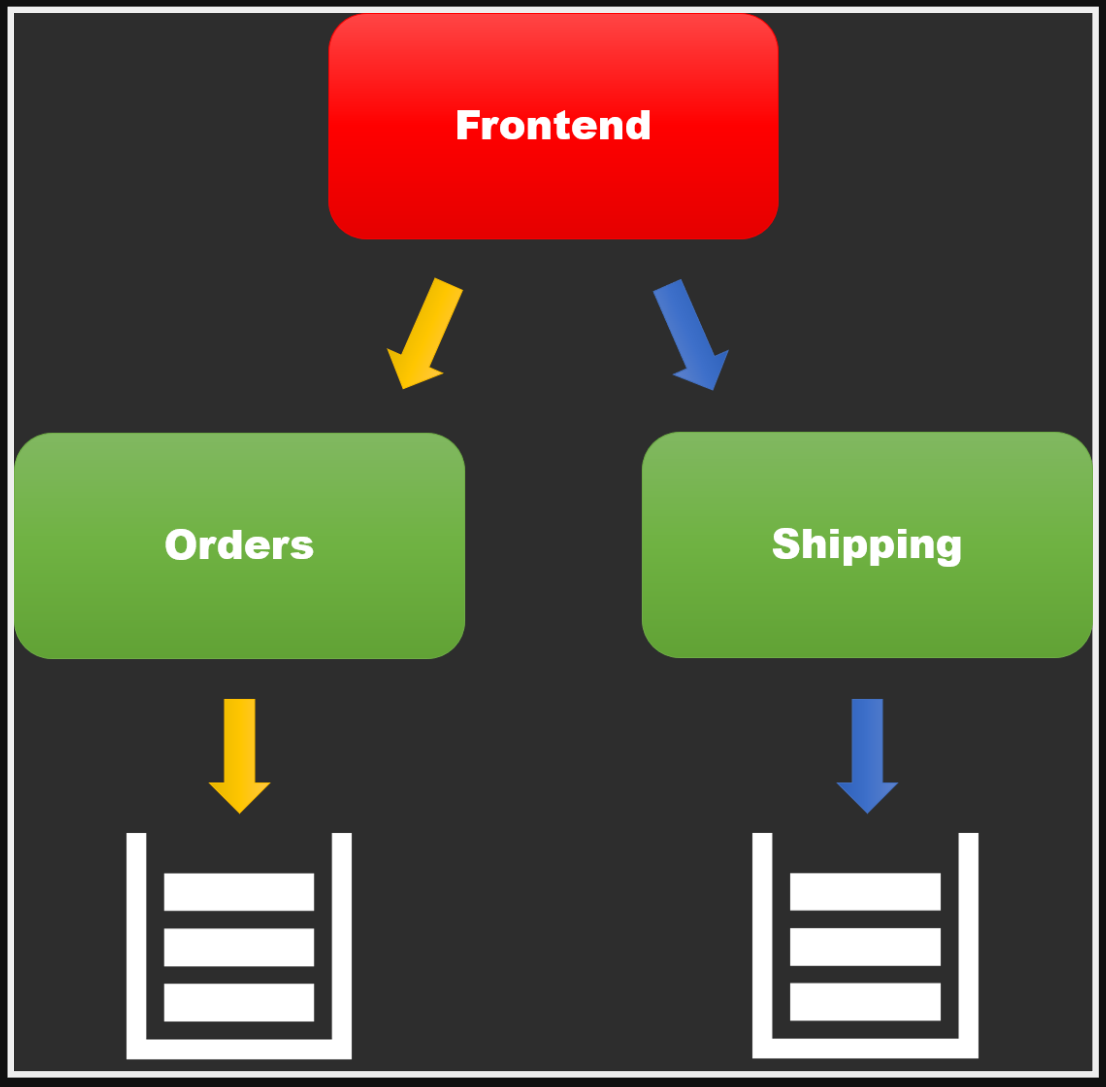
Divides the system into many components based on their functionality

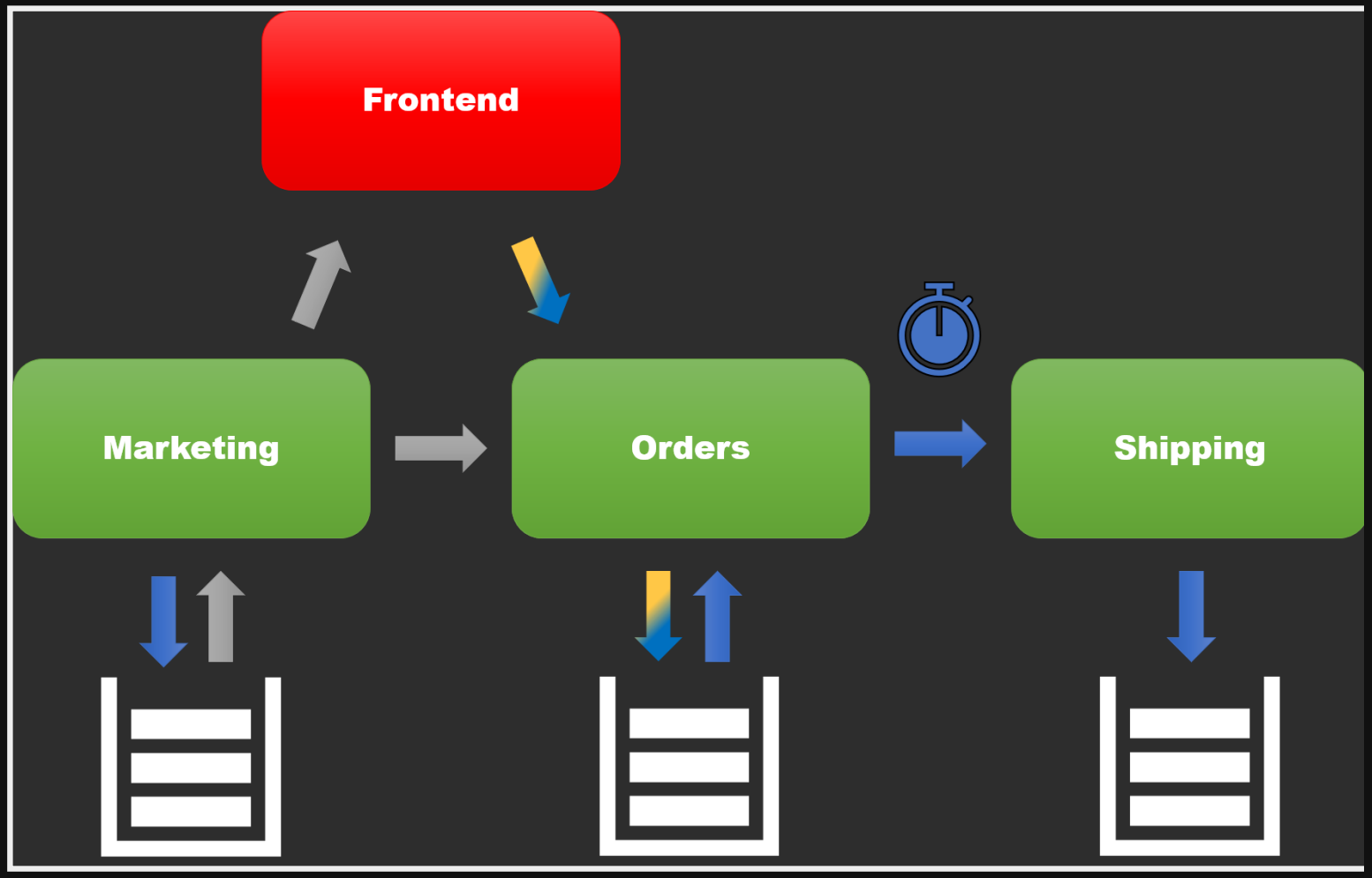
Divides the system into many components based on their functionality

Optimizes for low coupling at design-time


Frontend
Frontend
Marketing
Orders



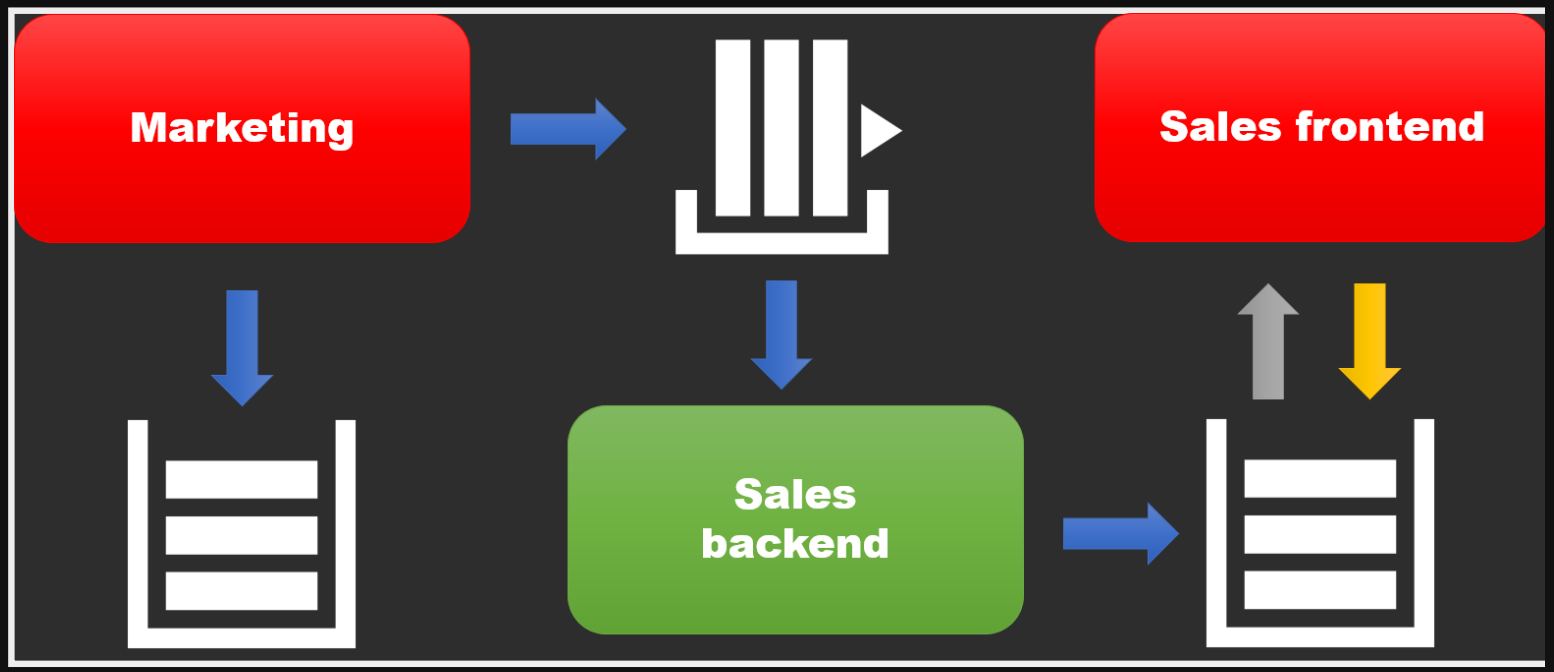

Frontend
Orders
Shipping

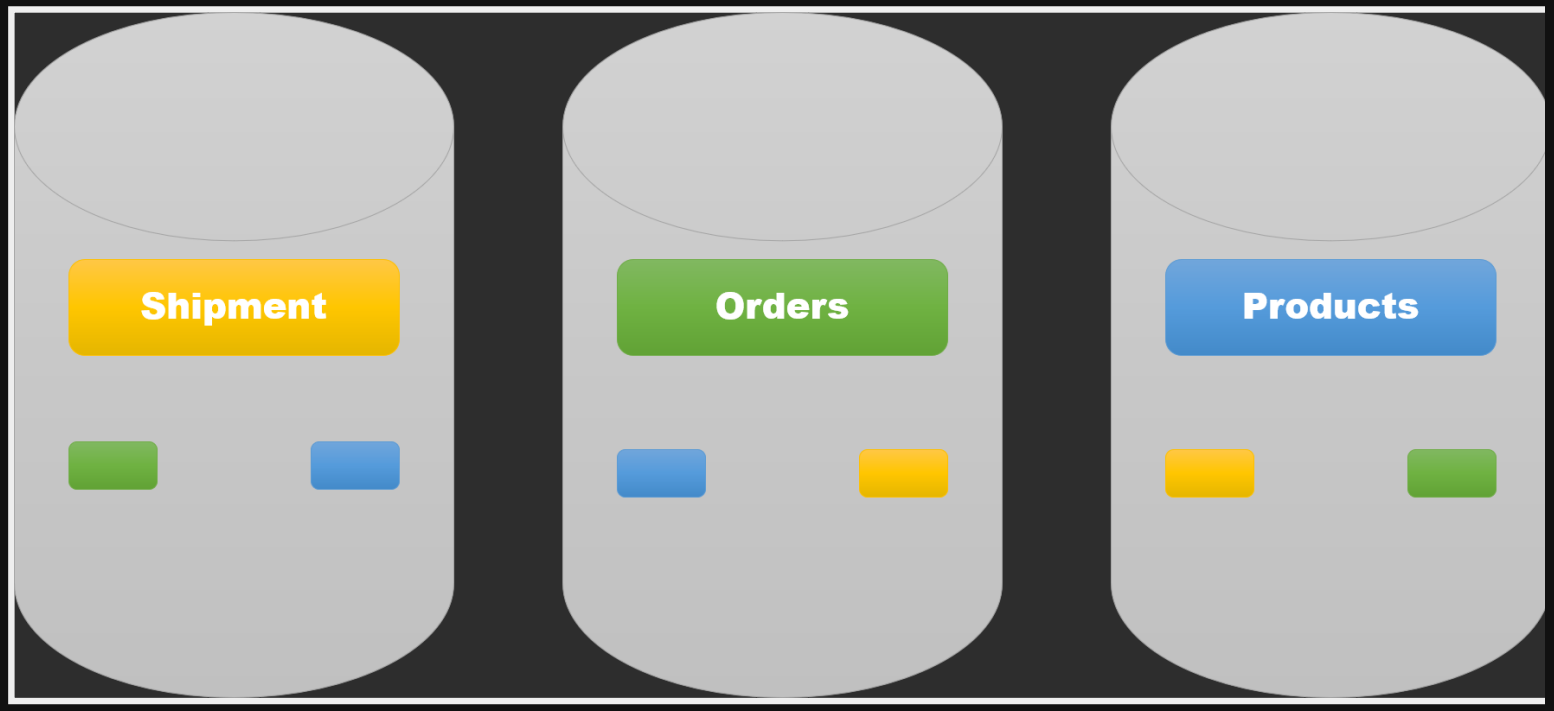

Frontend
Marketing
Orders
Shipping

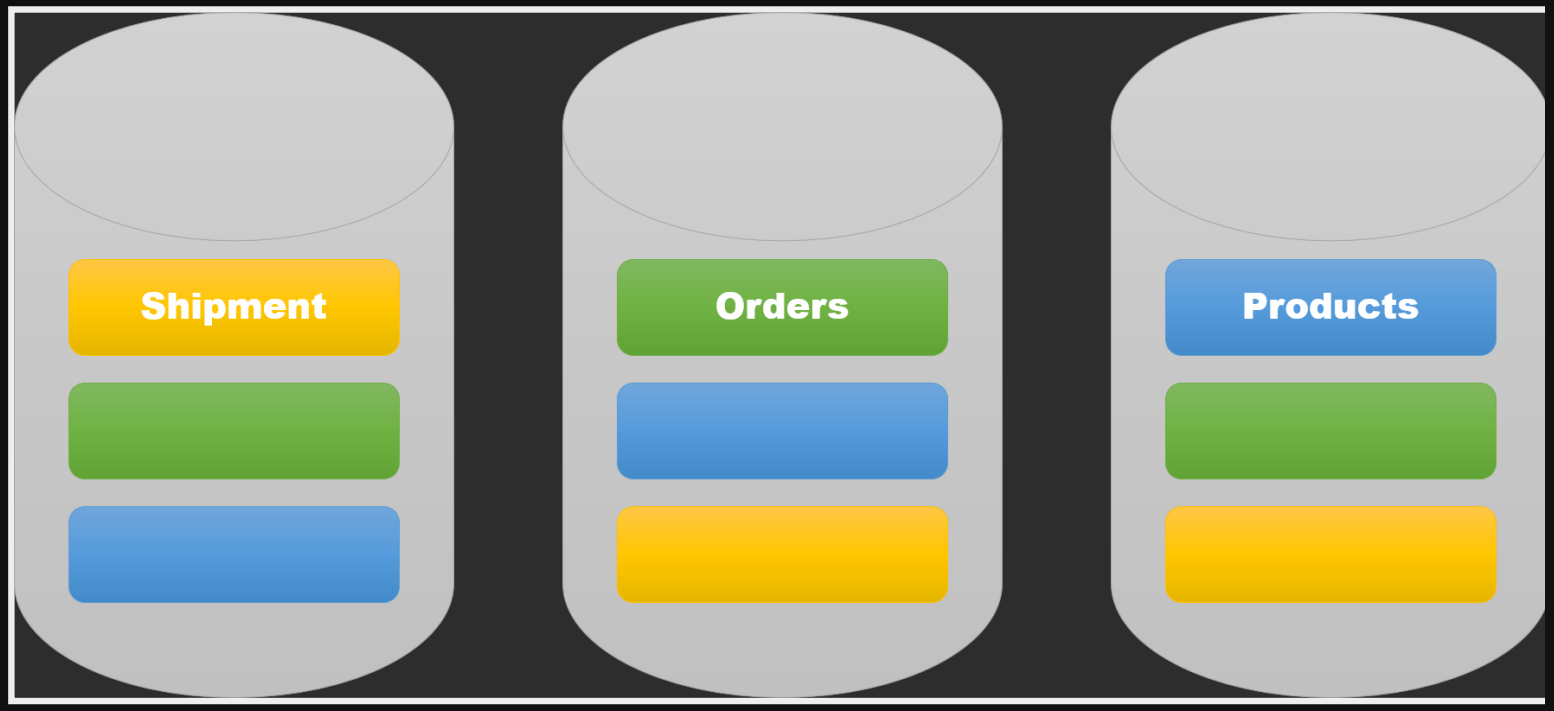
Divides the system into many components based on their functionality

Divides the system into many components based on their functionality

Optimizes for low coupling at run-time


Marketing
Sales frontend
Sales
backend

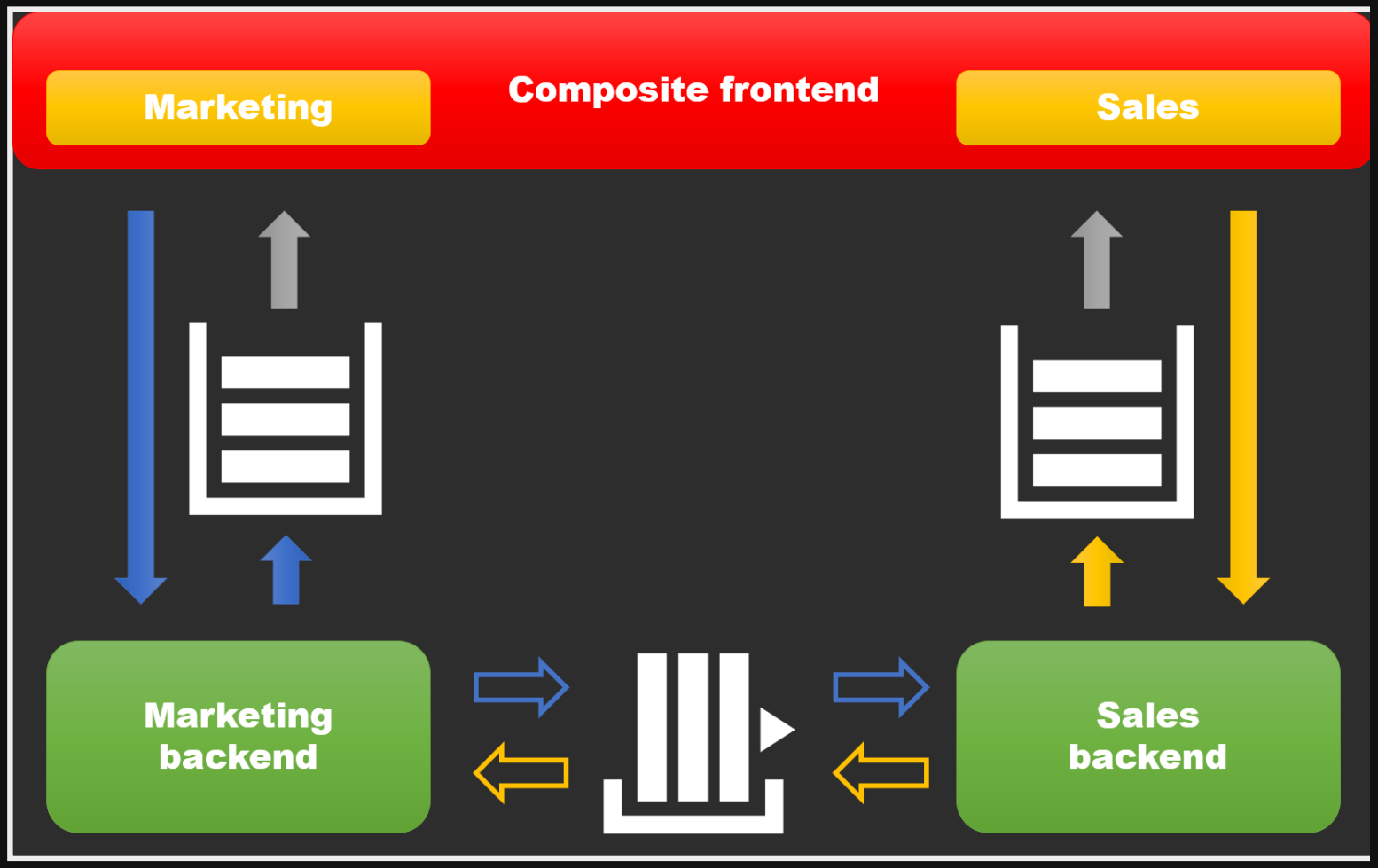

Shipment
Orders
Products


Shipment
Orders
Products

Divides the system into few components based on data they own

Divides the system into few components based on data they own

Optimizes for low data coupling

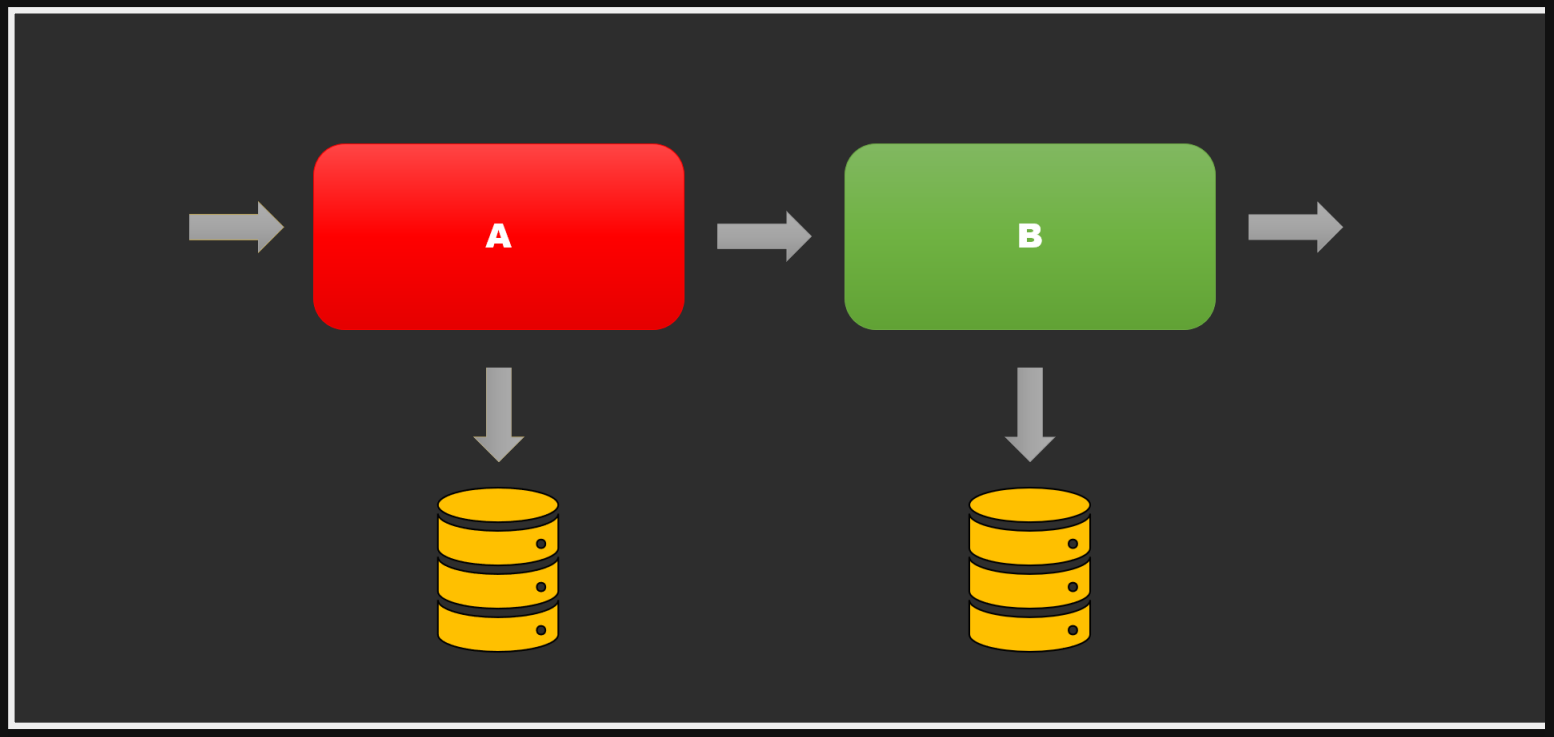

Marketing
Composite frontend
Sales
Marketing backend
Sales backend

Distributed systems

~~Distributed systems~~

~~Distributed systems~~

Distributed processes


A
B


WHAT DO WE WANT?!

- **Don't lose messages** 🗑

- Don't lose messages 🗑
- Apply state change only once 👯

- Don't lose messages 🗑
- Apply state change only once 👯
- Don't leak non-durable state 👻

Architecture is not an exact science

Architecture is not an exact science

Process distribution is not specific to any architectural style

Architecture is not an exact science

Process distribution is not specific to any architectural style

Process distribution requires consistent messaging

Architecture is not an exact science

Process distribution is not specific to any architectural style

Process distribution requires consistent messaging

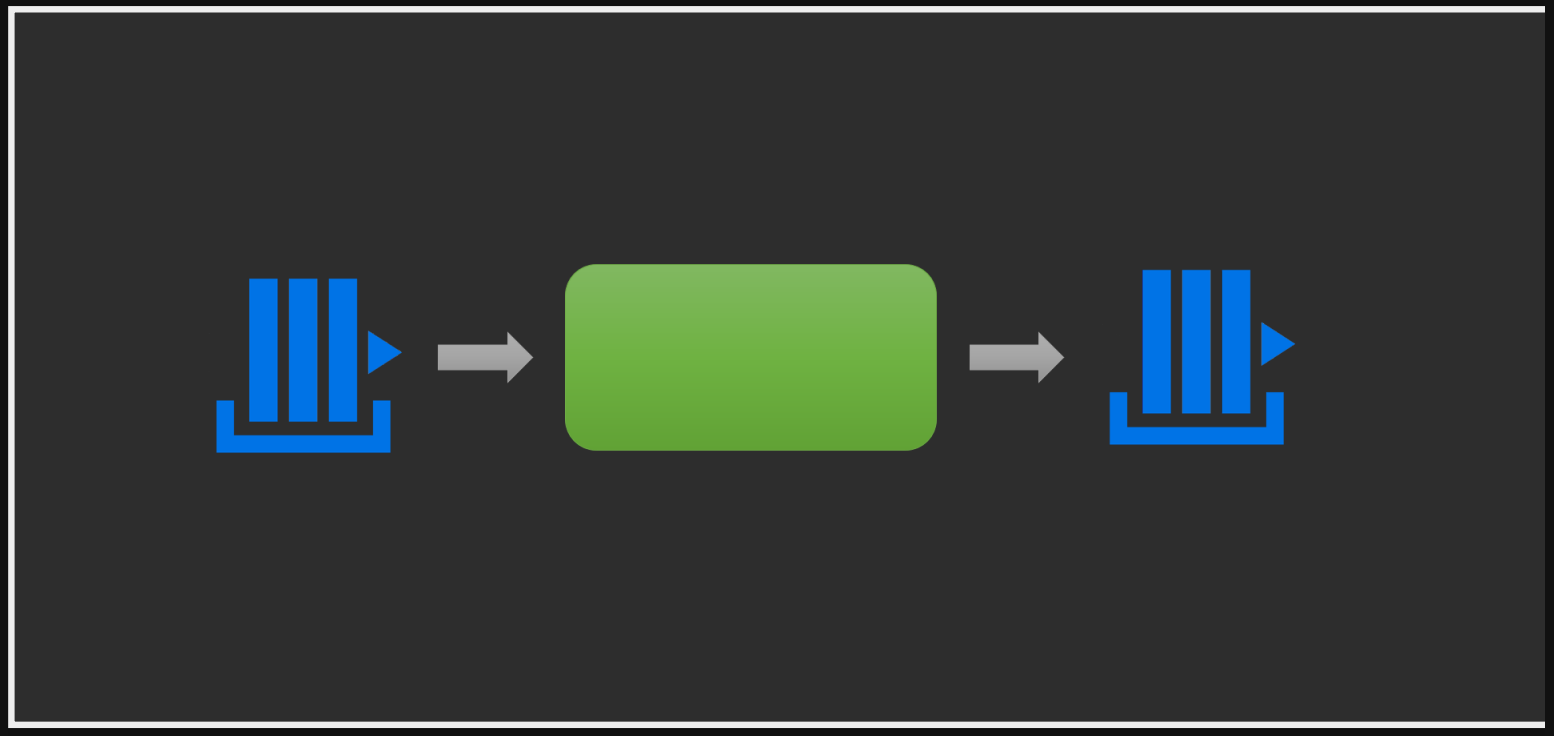
Consistent messaging is an exact science

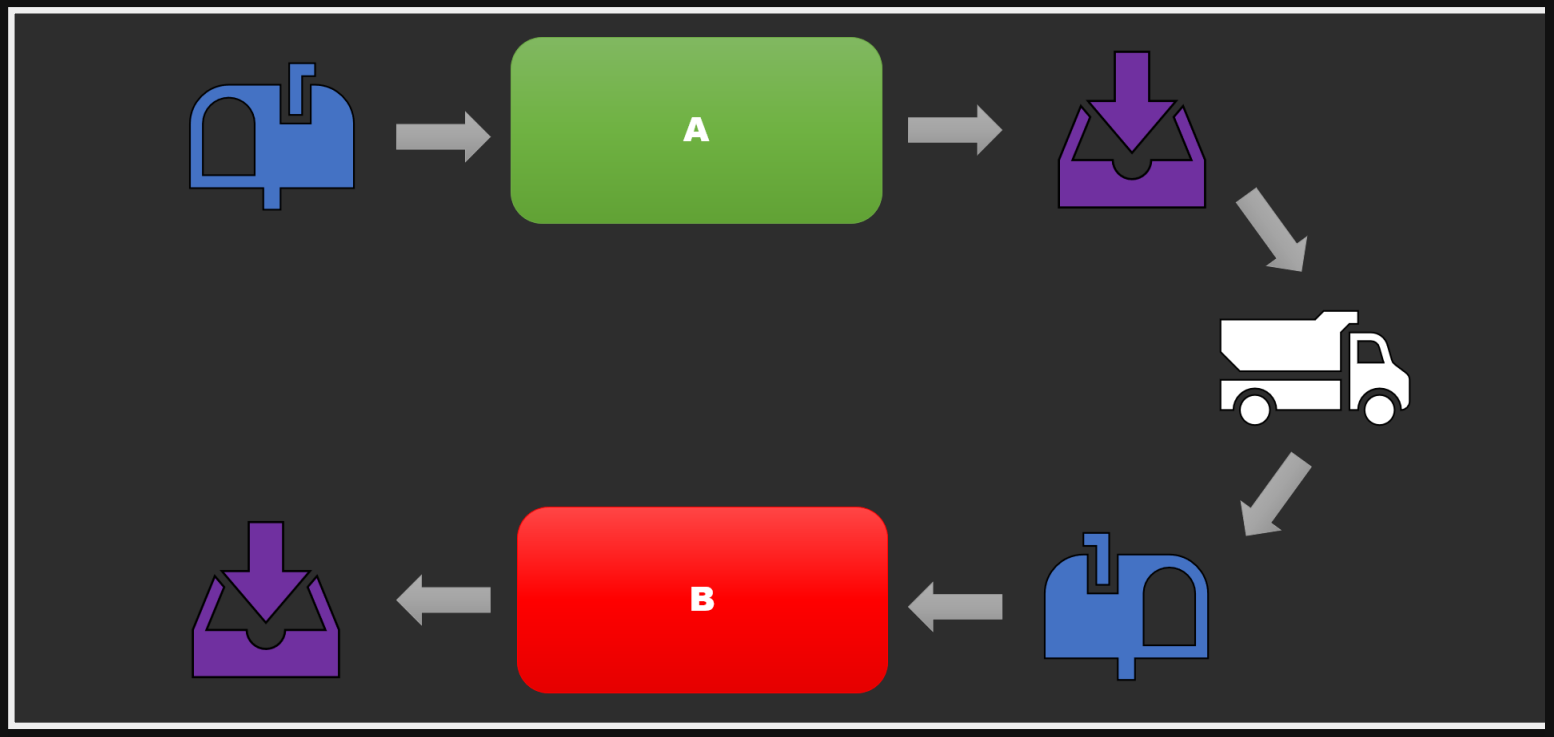
"... we want an endpoint to produce observable side-effects equivalent to some execution in which each logical message gets processed exactly-once"

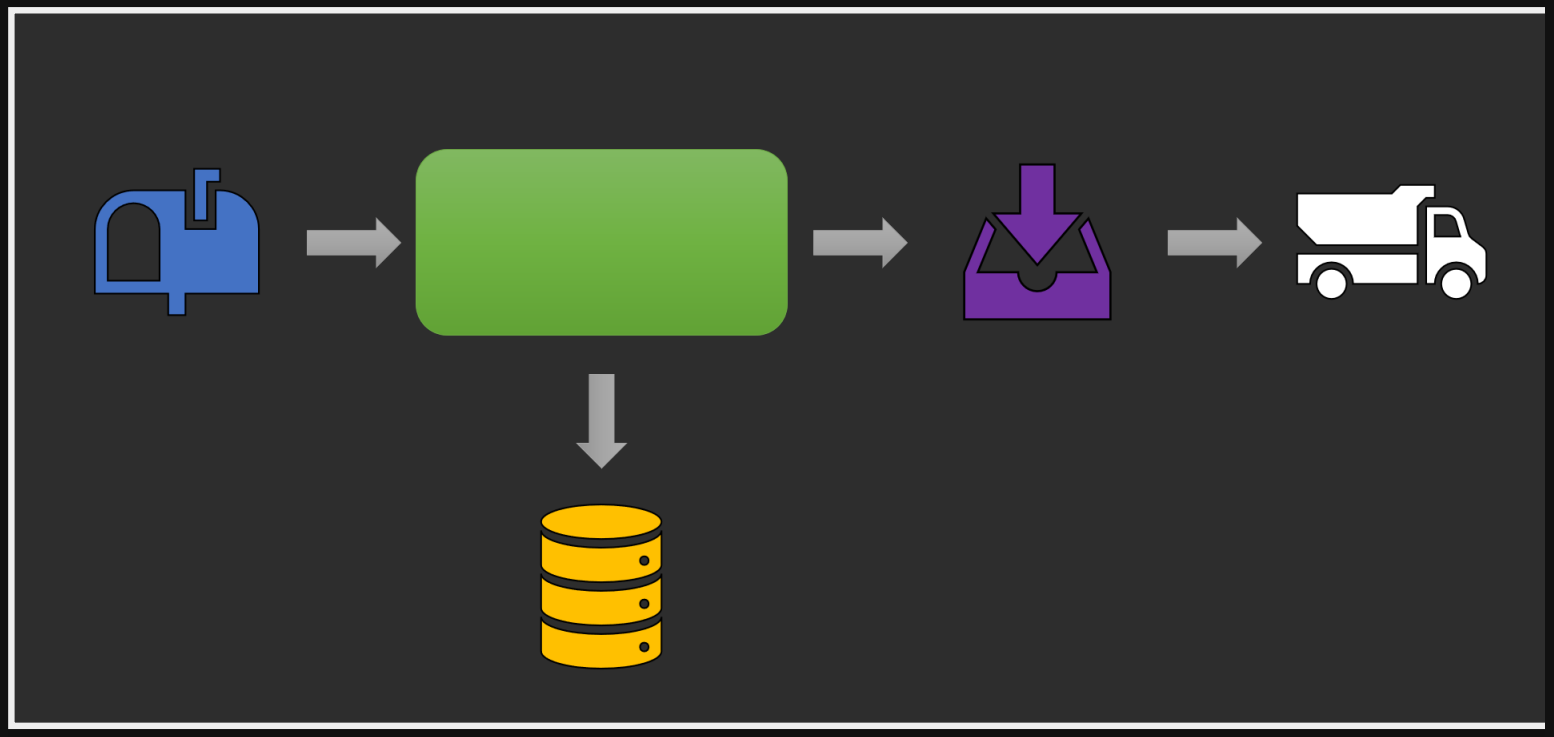
-- Tomasz Masternak

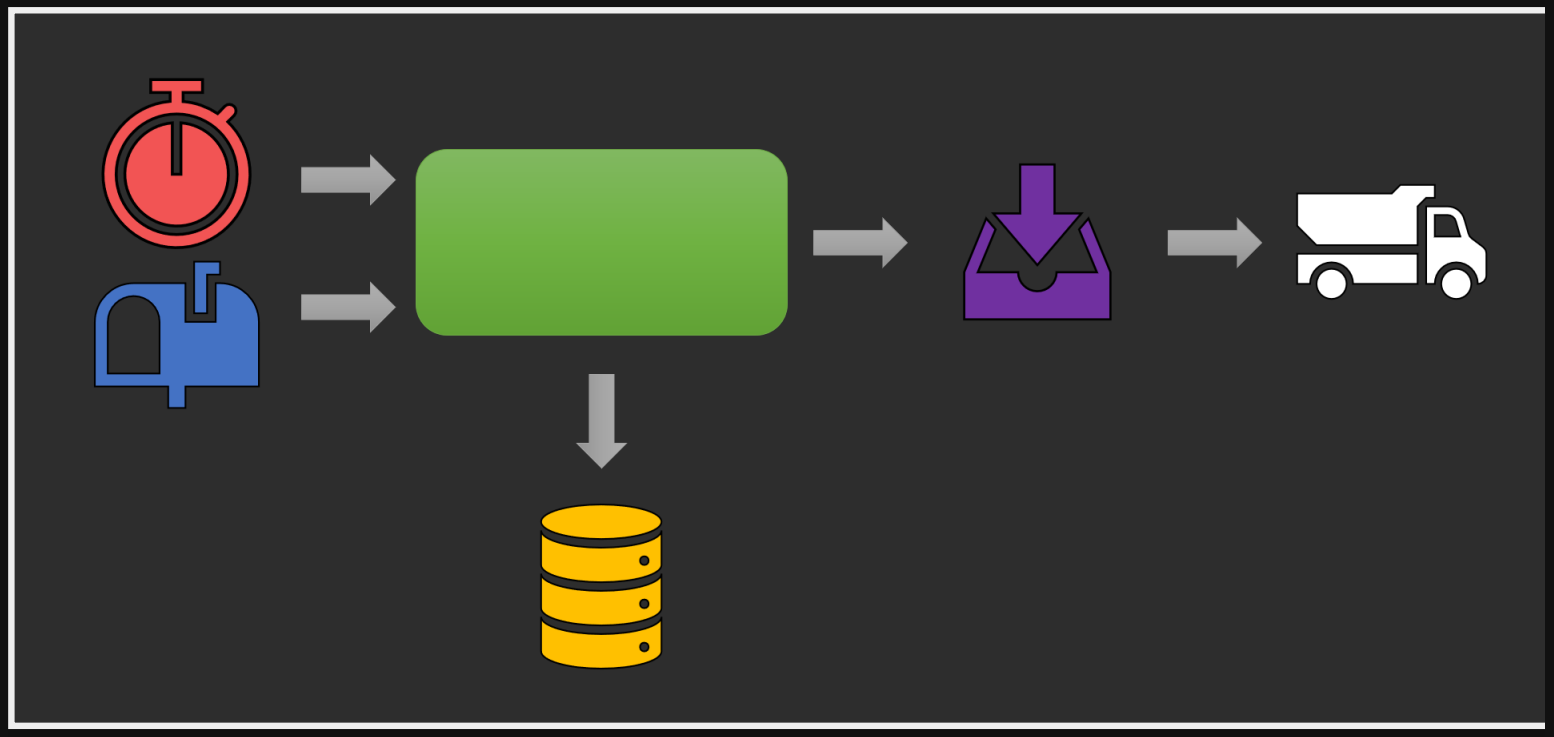
Consistent messaging

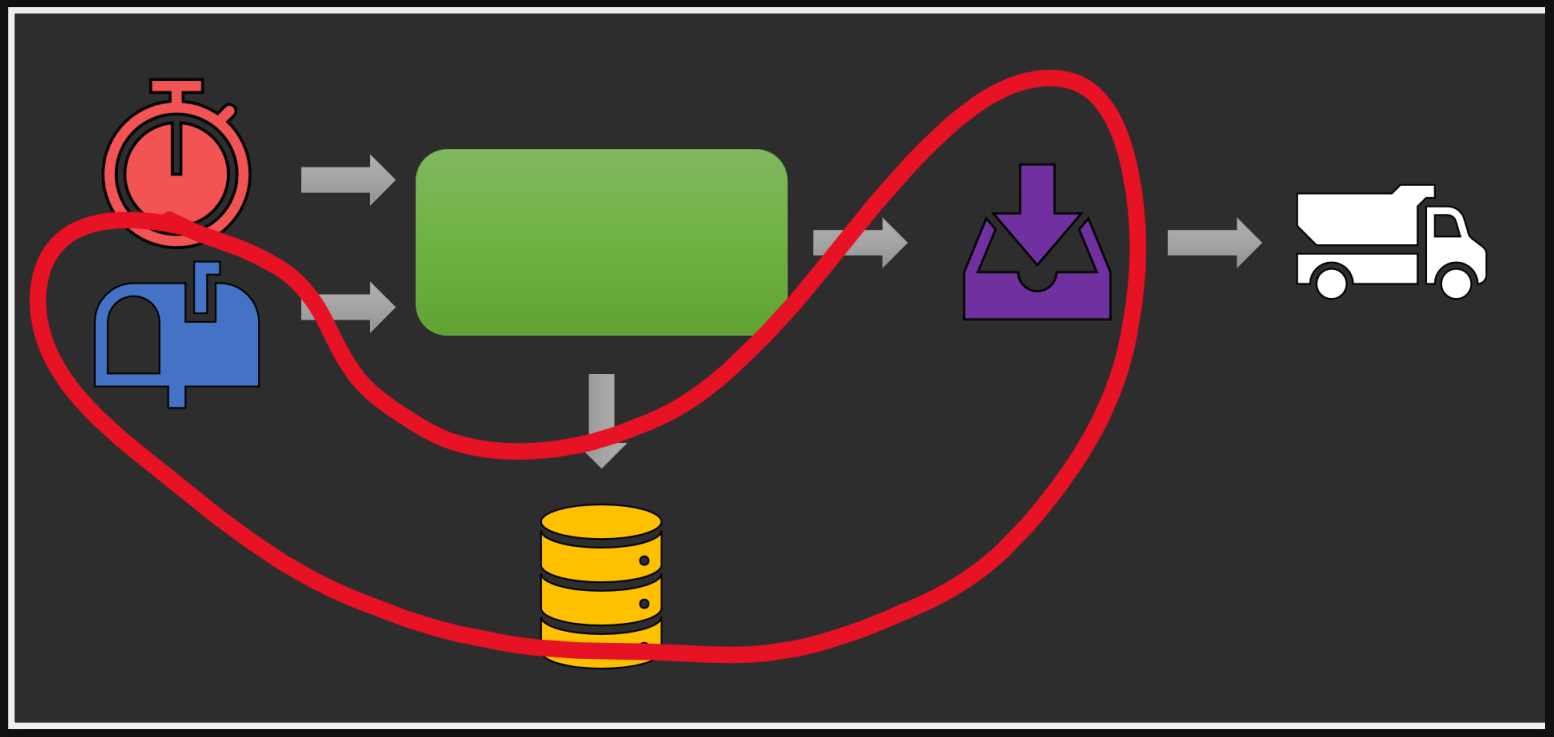
Model



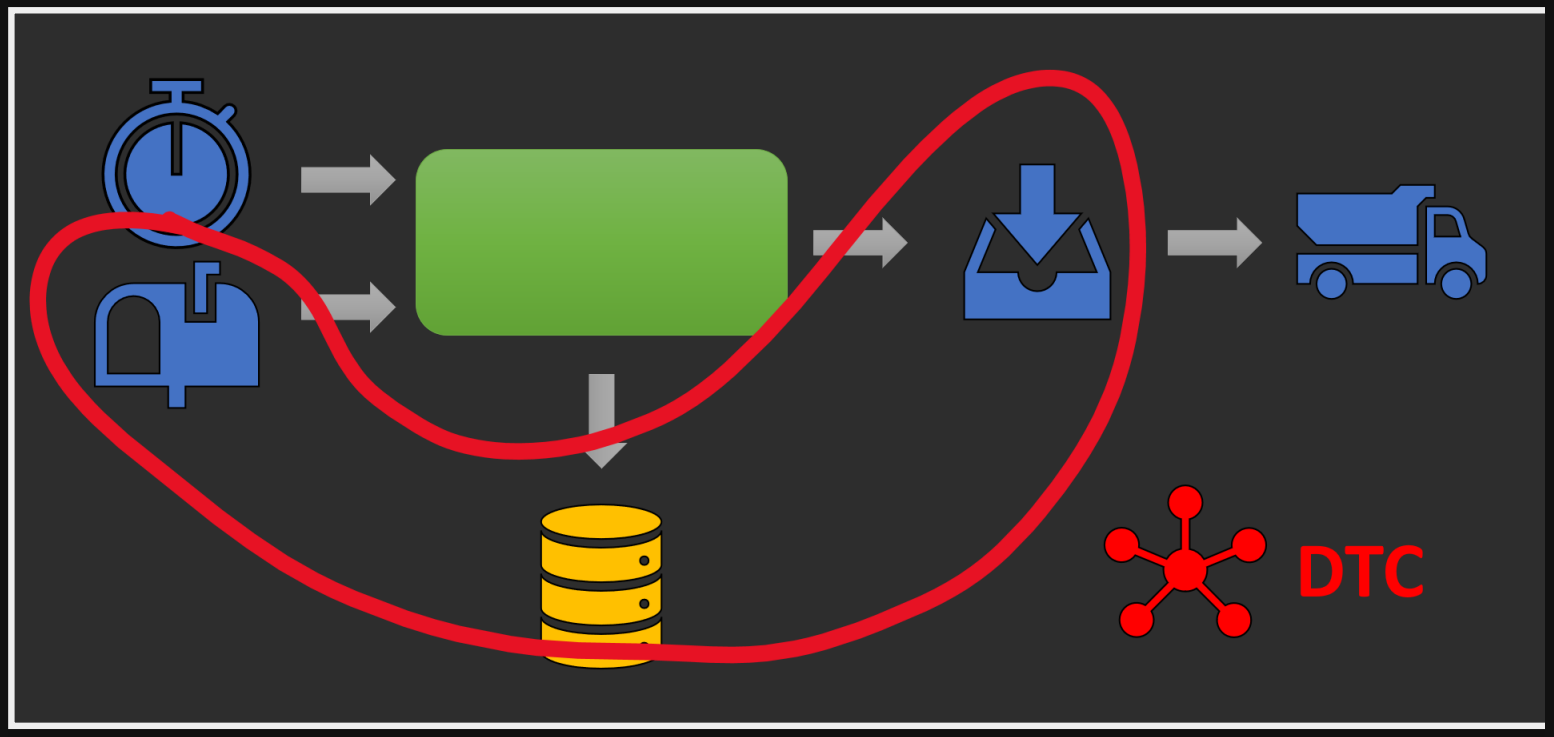

A
B

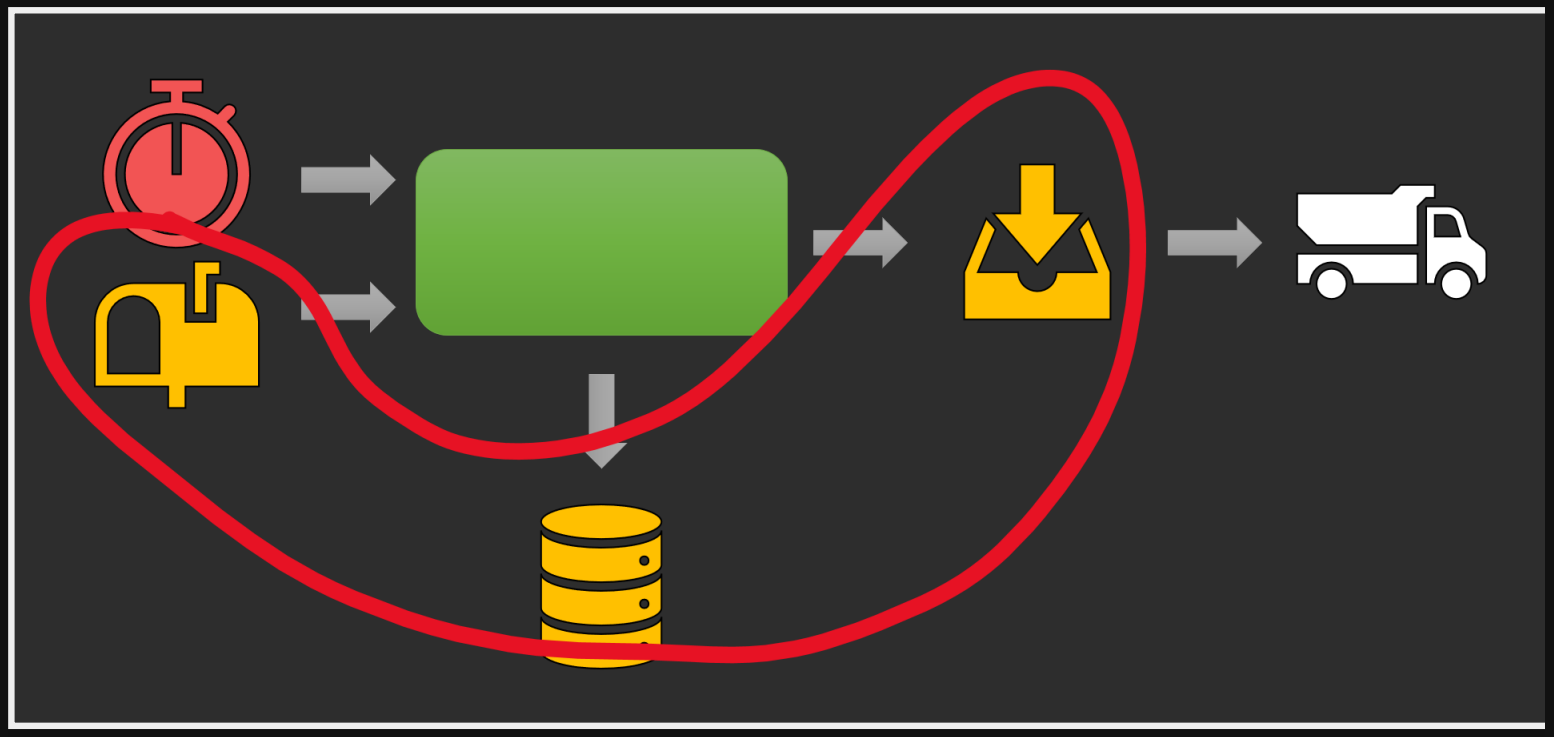


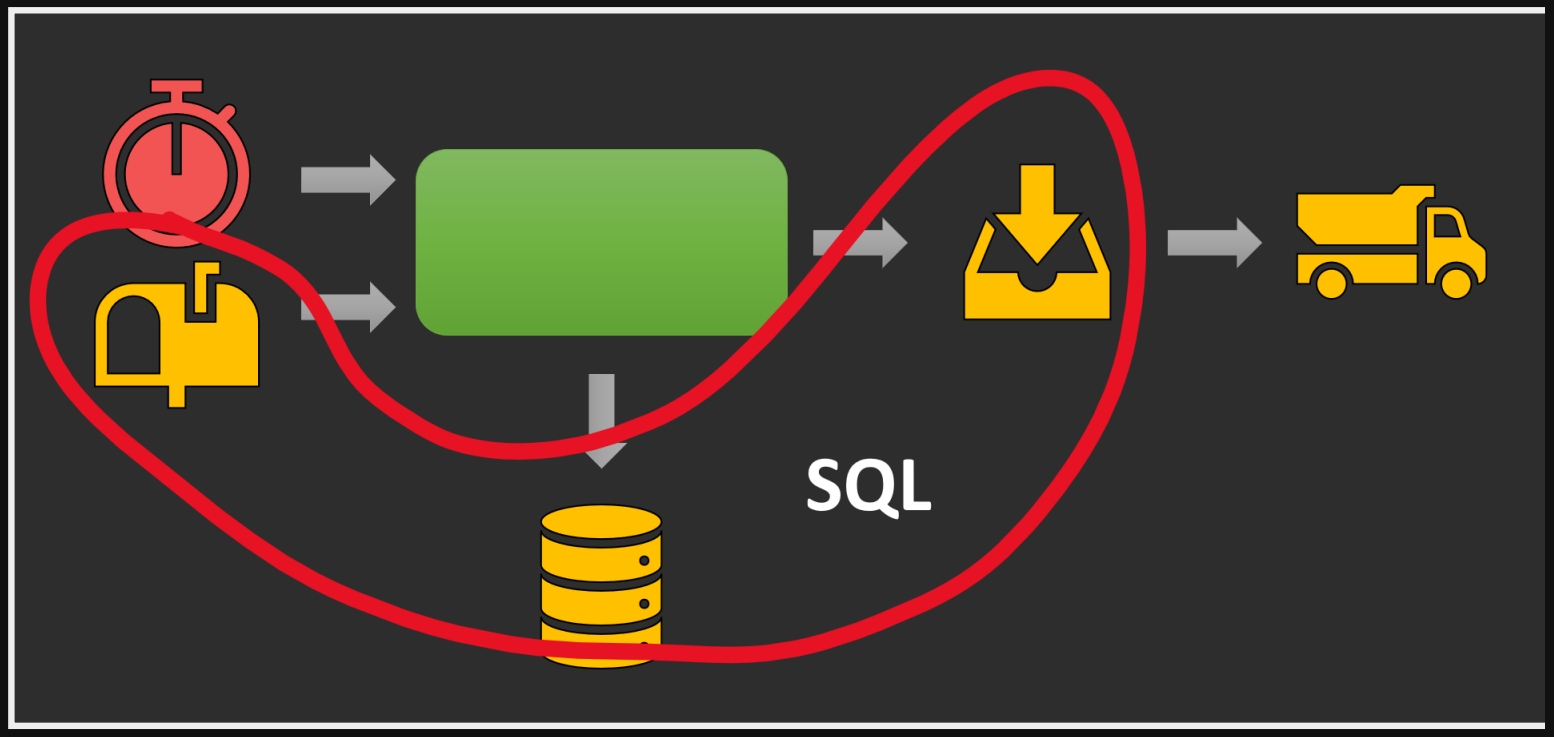


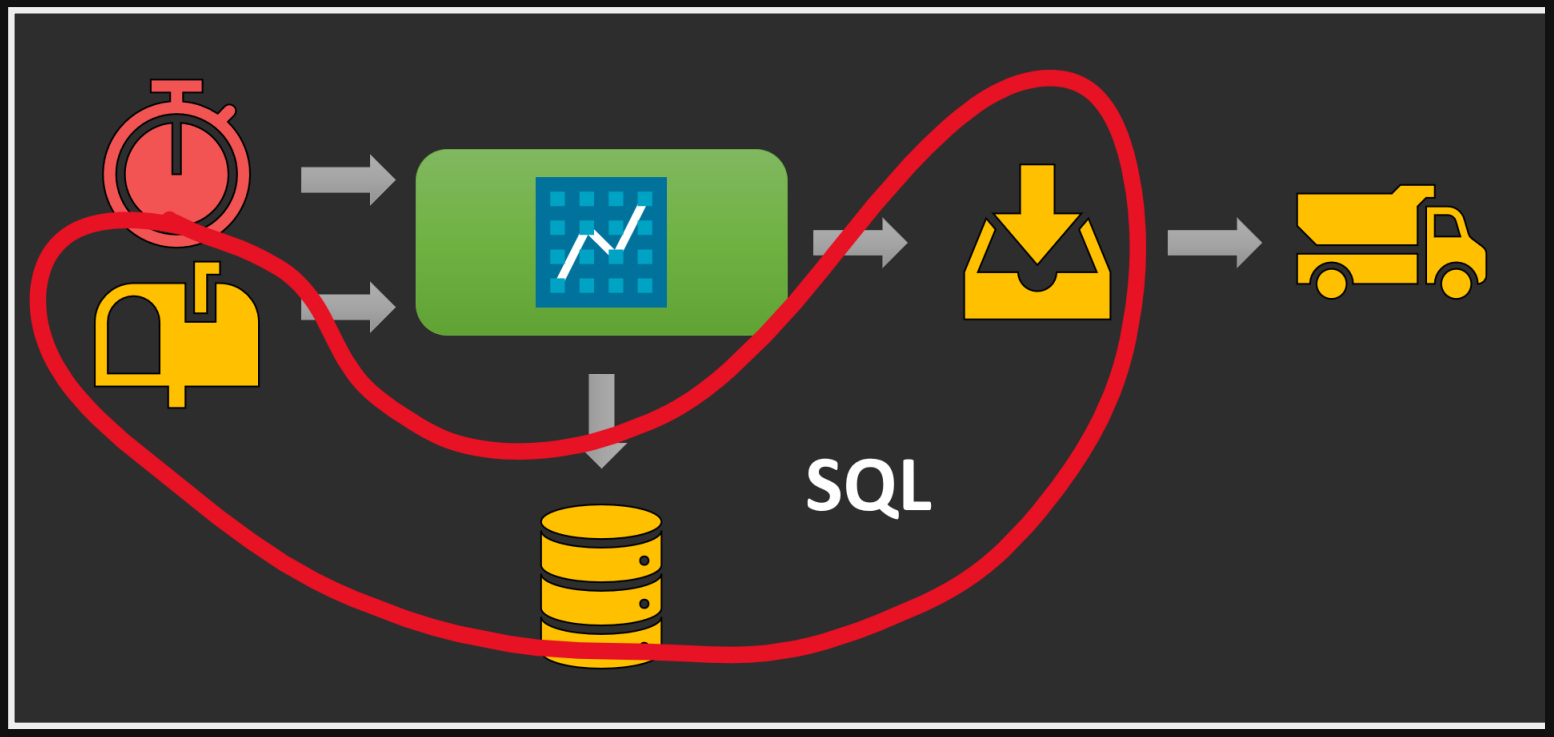



MSMQ


DTC




SQL


SQL

15 x 50 msg/s - 8 vCPU @ 15%

15 x 50 msg/s - 8 vCPU @ 15%

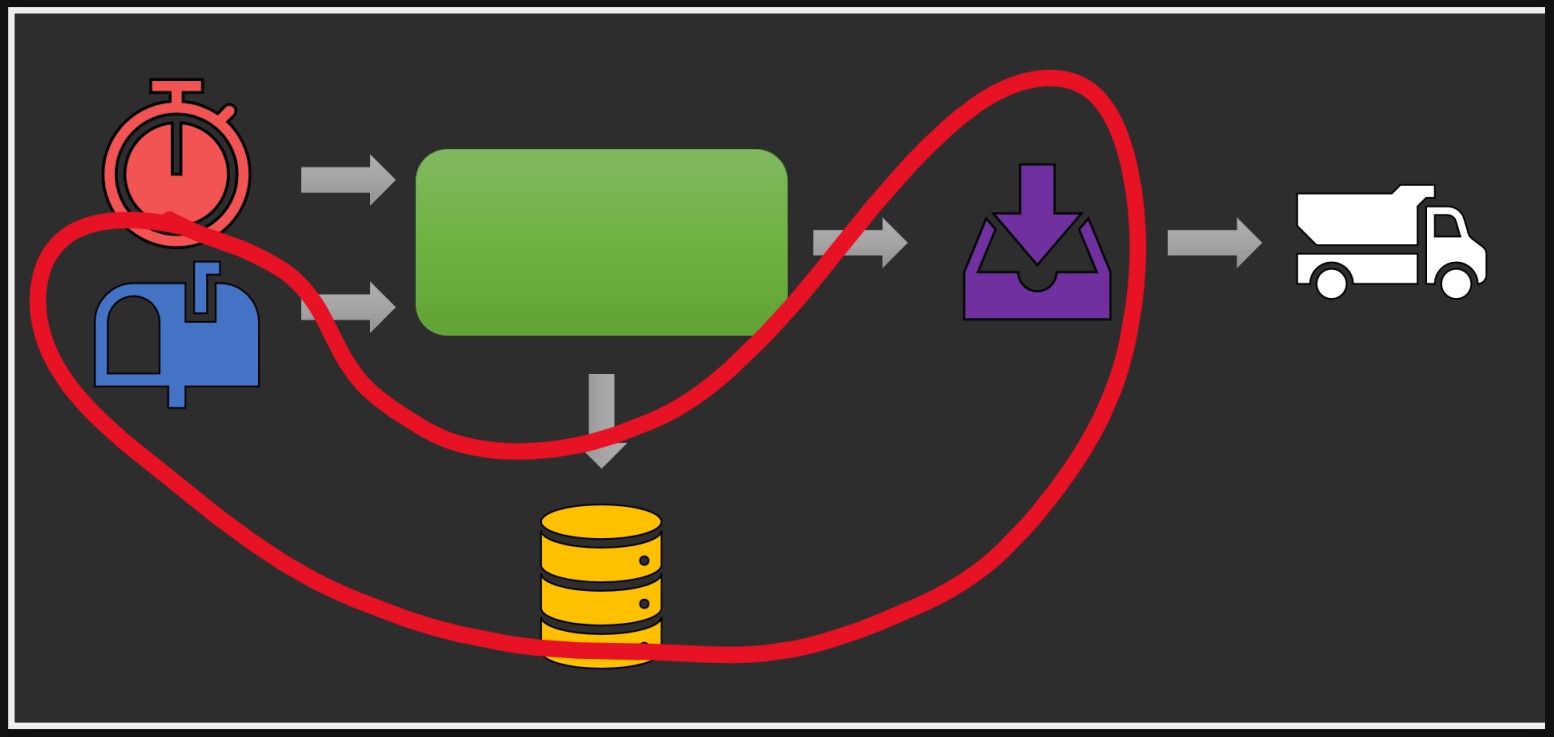
4 320 000

15 x 50 msg/s - 8 vCPU @ 15%

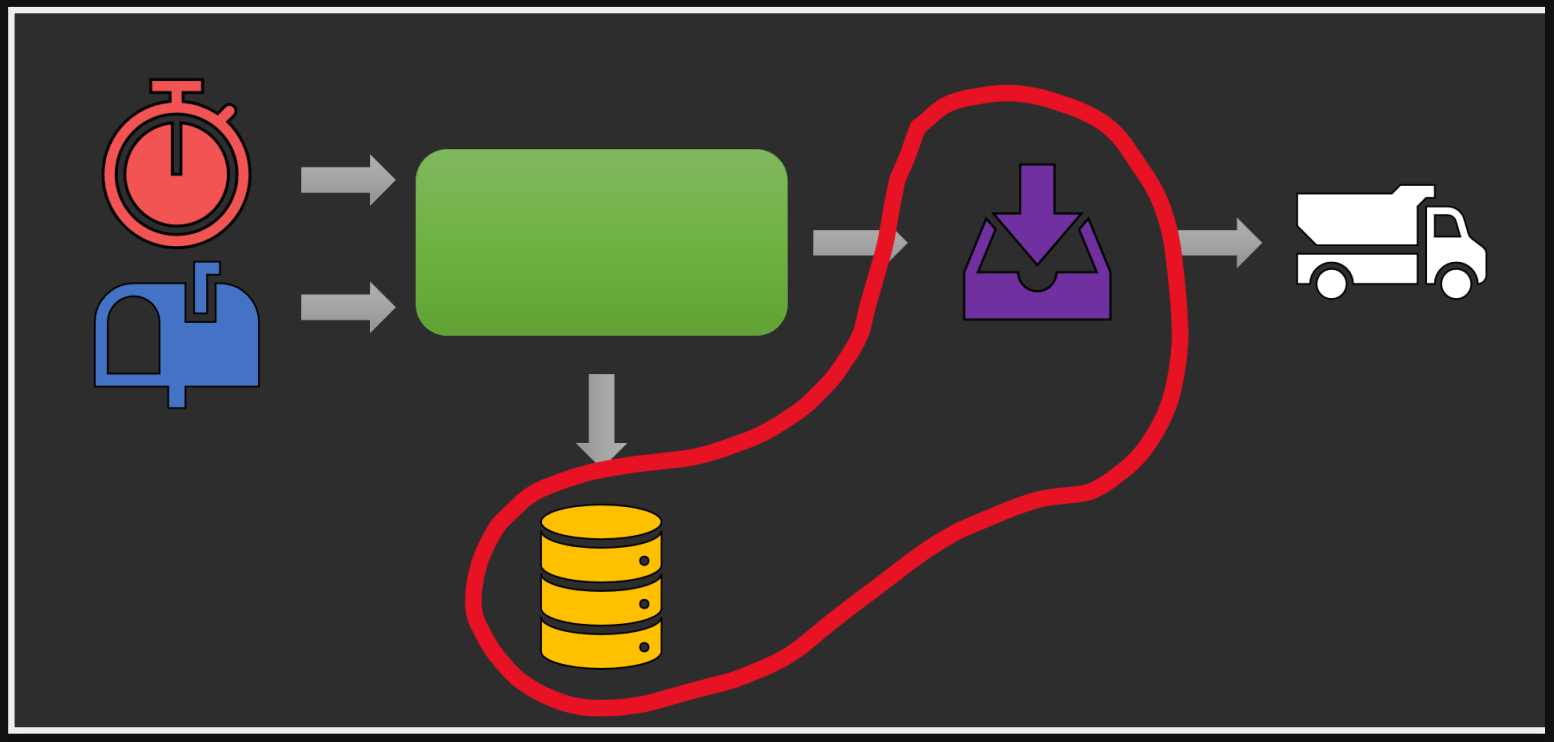
4 320 000

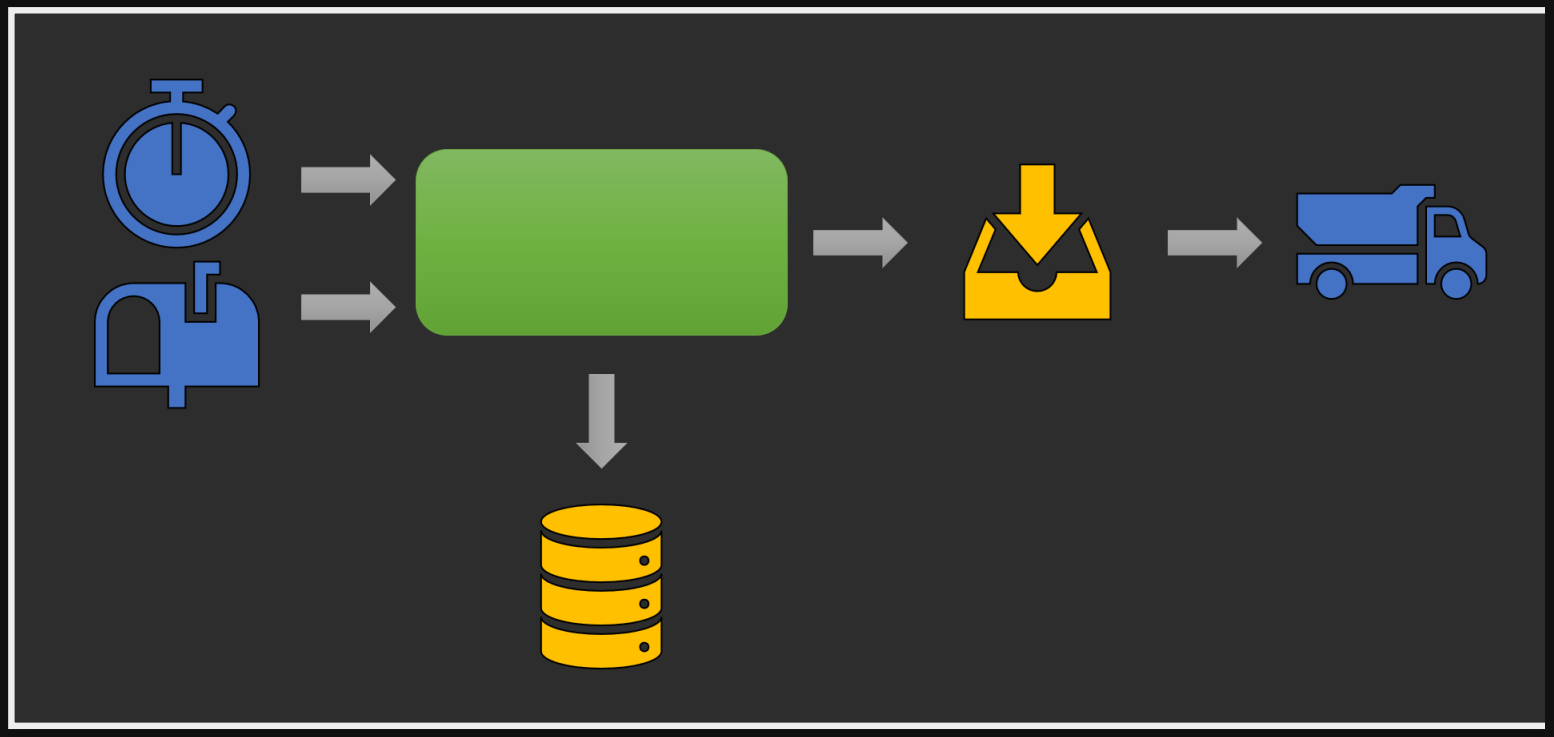
$ 48.42

Web Scale



Deduplication





```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

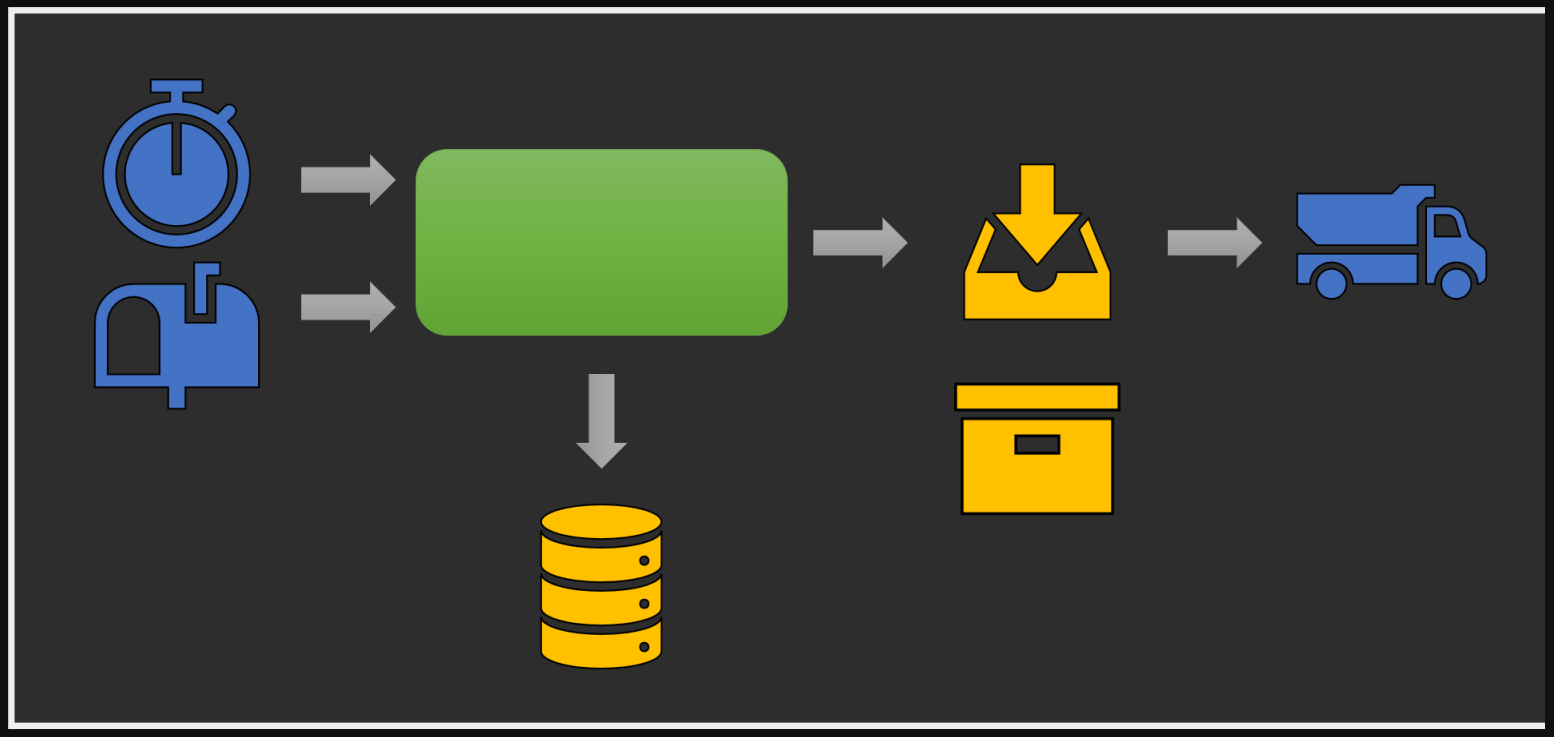
```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

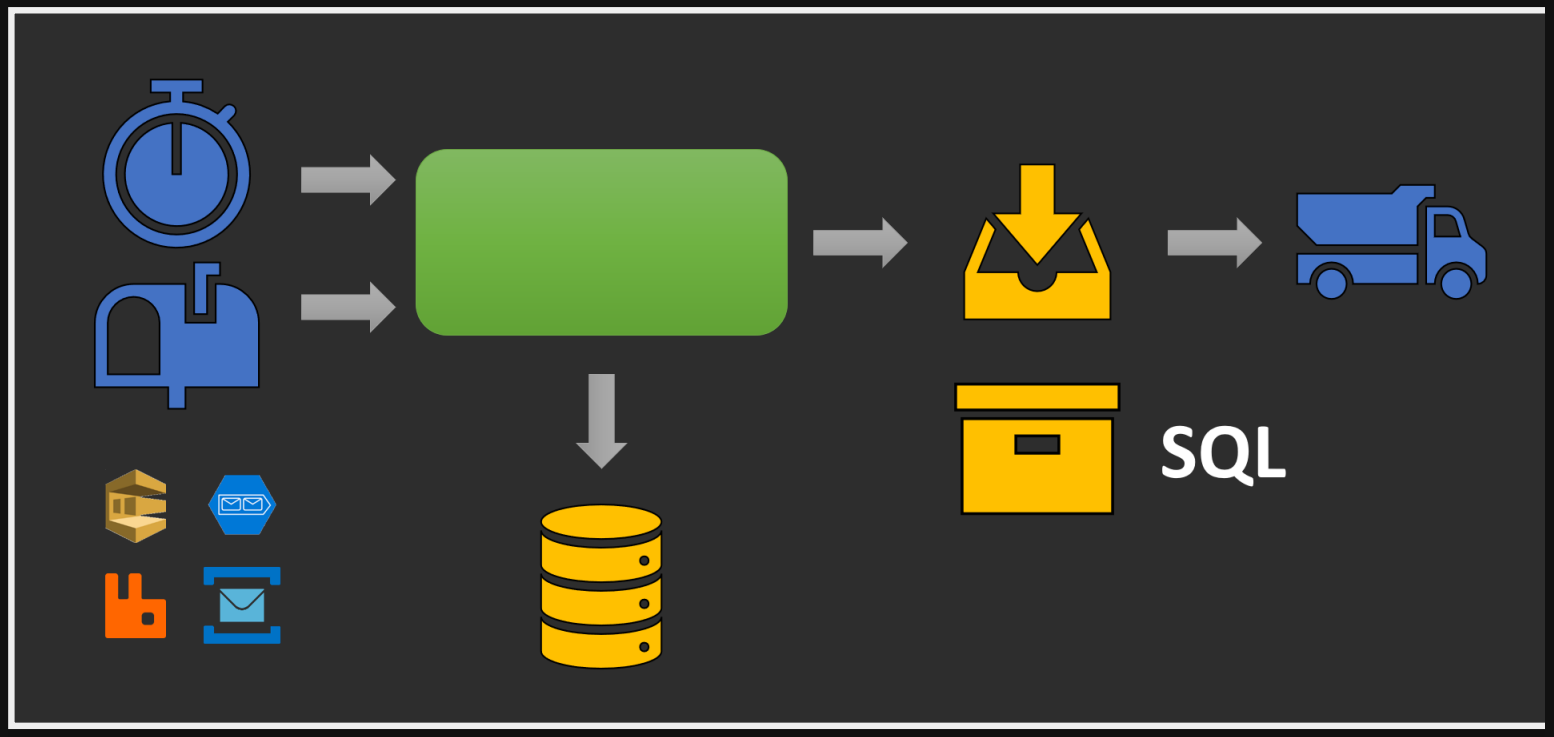
```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

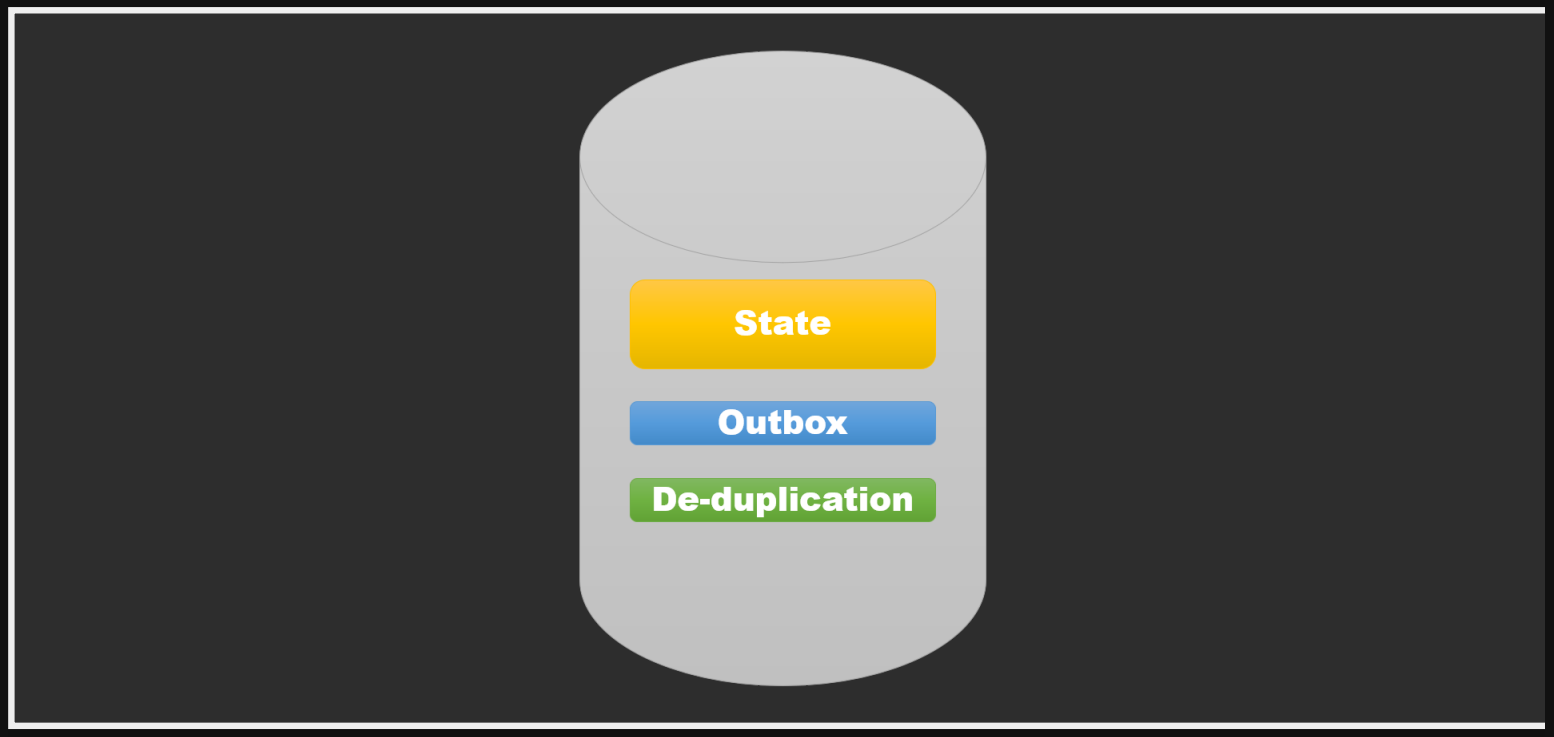
```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);🔥🔥🔥
```

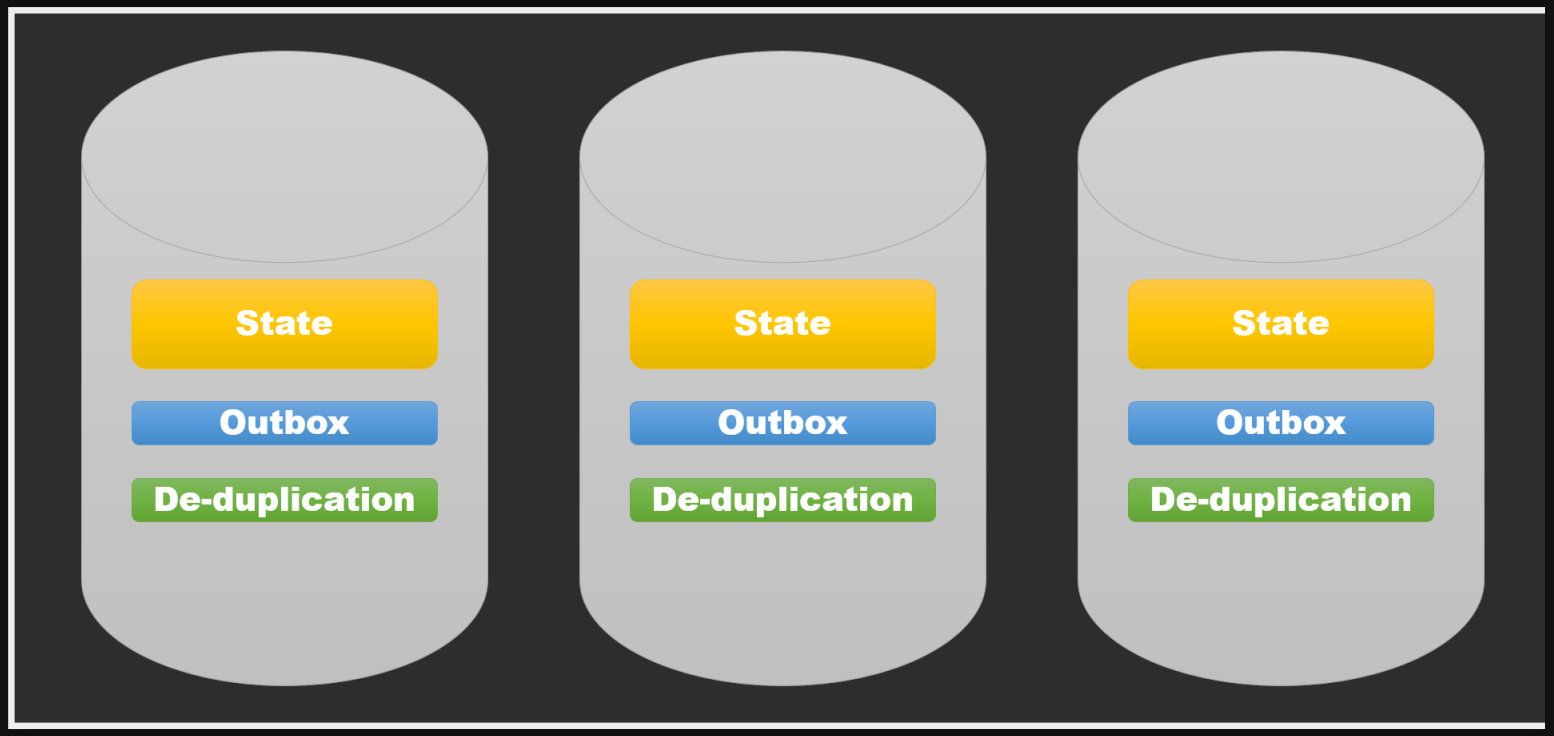


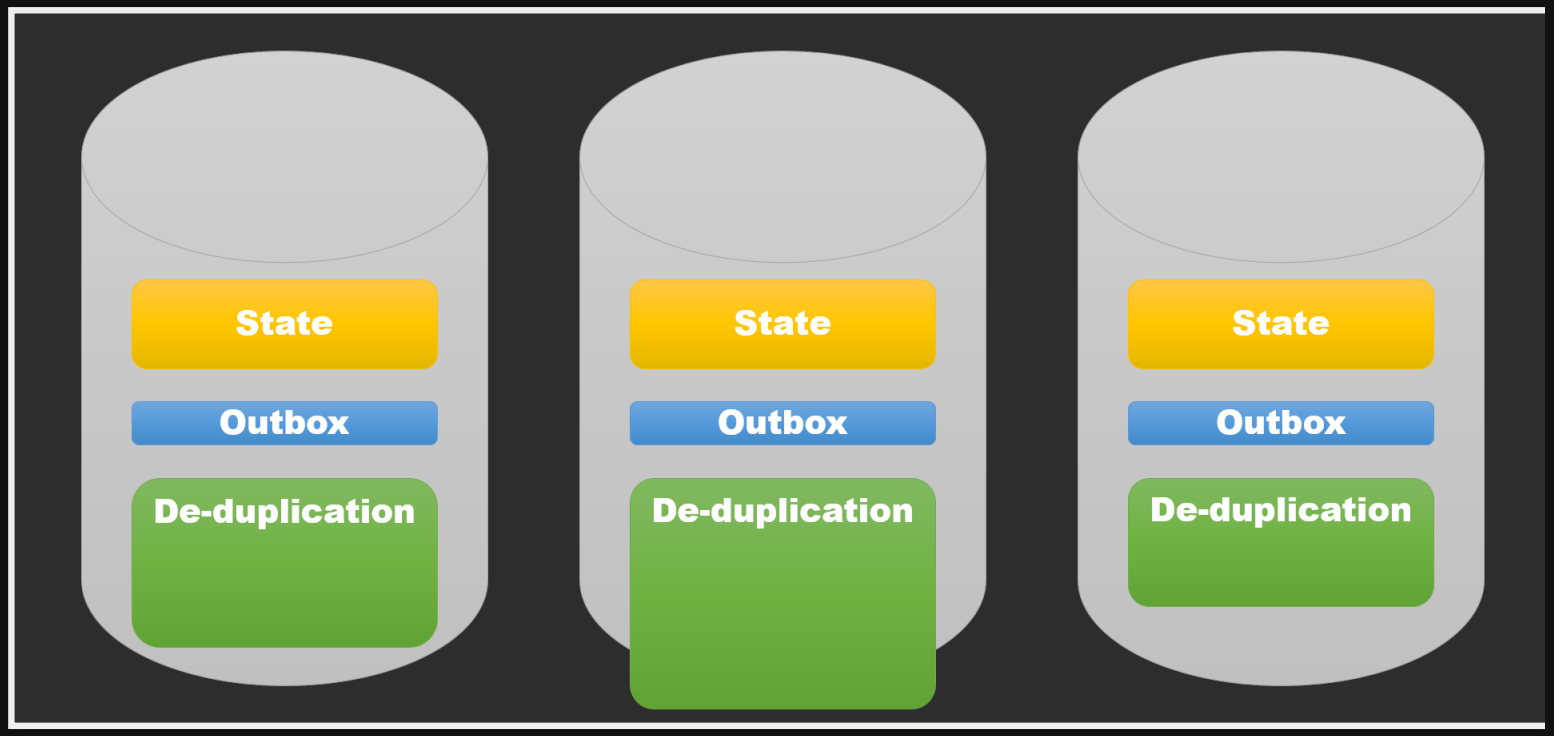

SQL


SQL

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (outbox.Dispatched) {
        return;
}
await Dispatch(incoming.Id);
await MarkAsDispatched(incoming.Id);
```


State
Outbox
De-duplication


State
Outbox
De-duplication
State
Outbox
De-duplication
State
Outbox
De-duplication


State
Outbox
De-duplication
State
Outbox
De-duplication
State
Outbox
De-duplication

Deduplication record is created when message is processed

Deduplication record is created when message is processed

Deduplication record is present in at least one store

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
1  if (HasDedupeEntry(incoming.Id)) {
2          return;
3  }
4  var outbox = await Load(incoming.Id);
5  if (outbox == null) {
6          var (state, outgoing) = Process(incoming);
7          await Save(incoming.Id, state, outgoing);
8  }
9  await Dispatch(incoming.Id);
10 await AddDedupeEntry(incoming.Id);🔥🔥🔥
11 await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

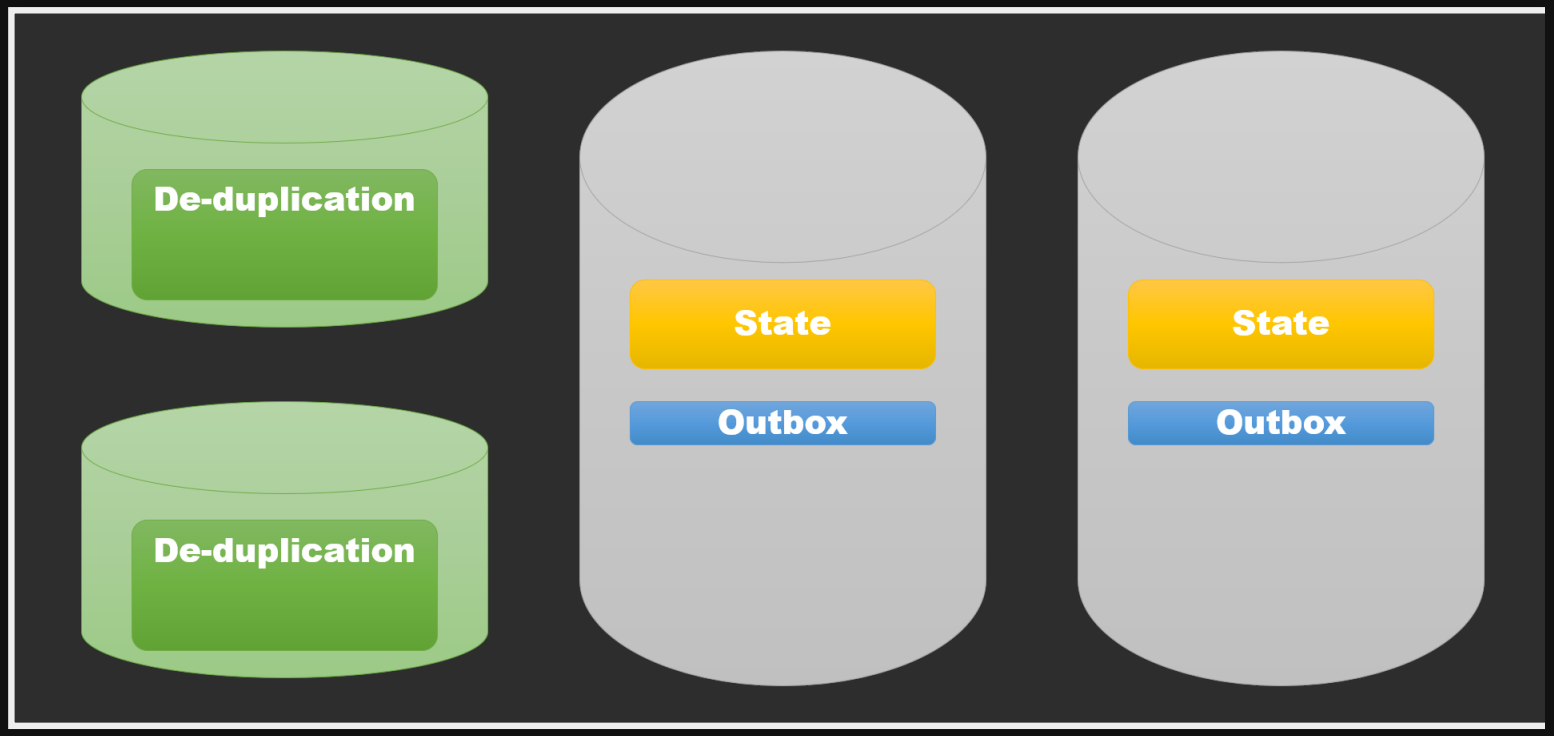
```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

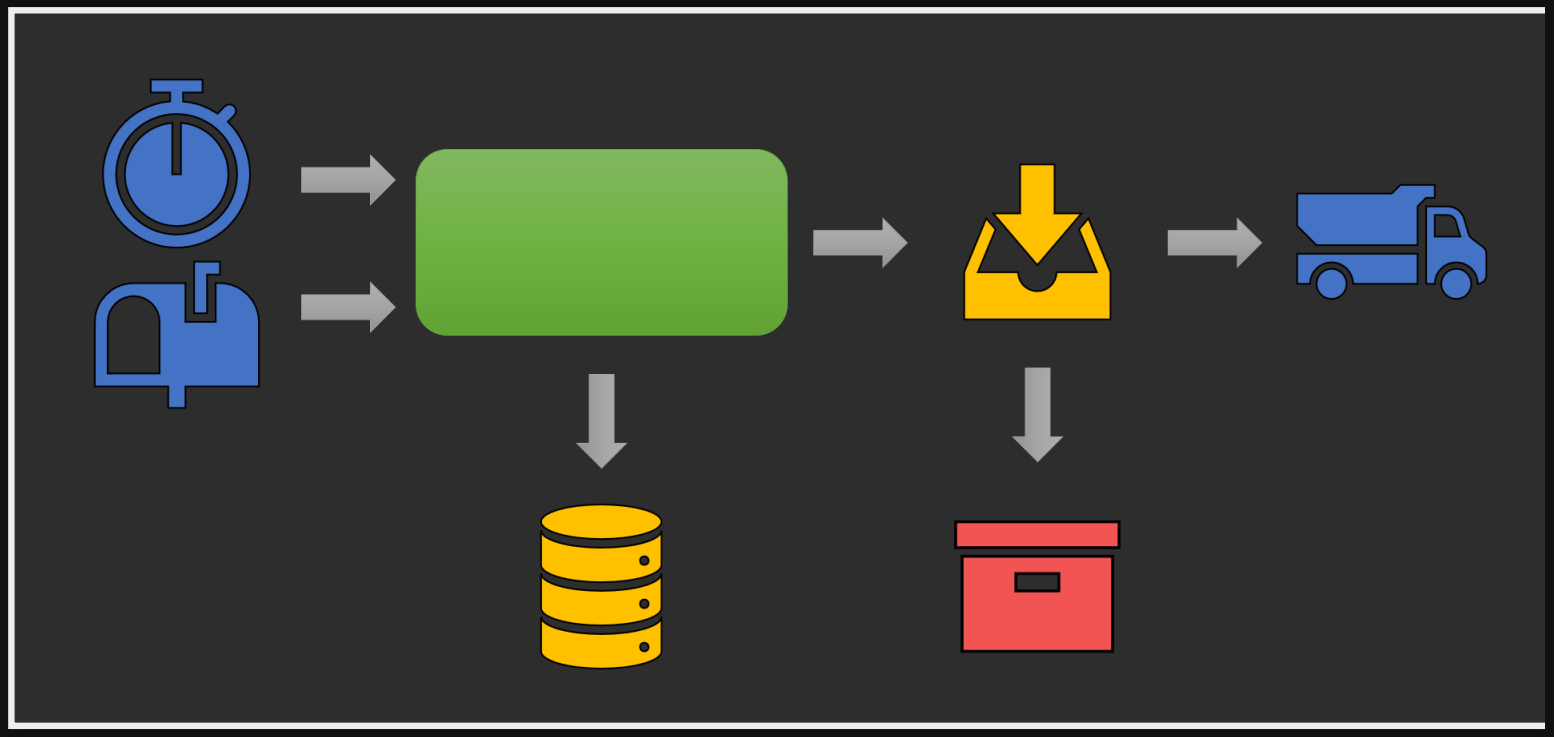
```
1  if (HasDedupeEntry(incoming.Id)) {
2          return;
3  }
4  var outbox = await Load(incoming.Id);
5  if (outbox == null) {
6          var (state, outgoing) = Process(incoming);
7          await Save(incoming.Id, state, outgoing);
8  }
9  await Dispatch(incoming.Id);
10 await AddDedupeEntry(incoming.Id);🔥🔥🔥
11 await RemoveOutbox(incoming.Id);🔥🔥🔥
```

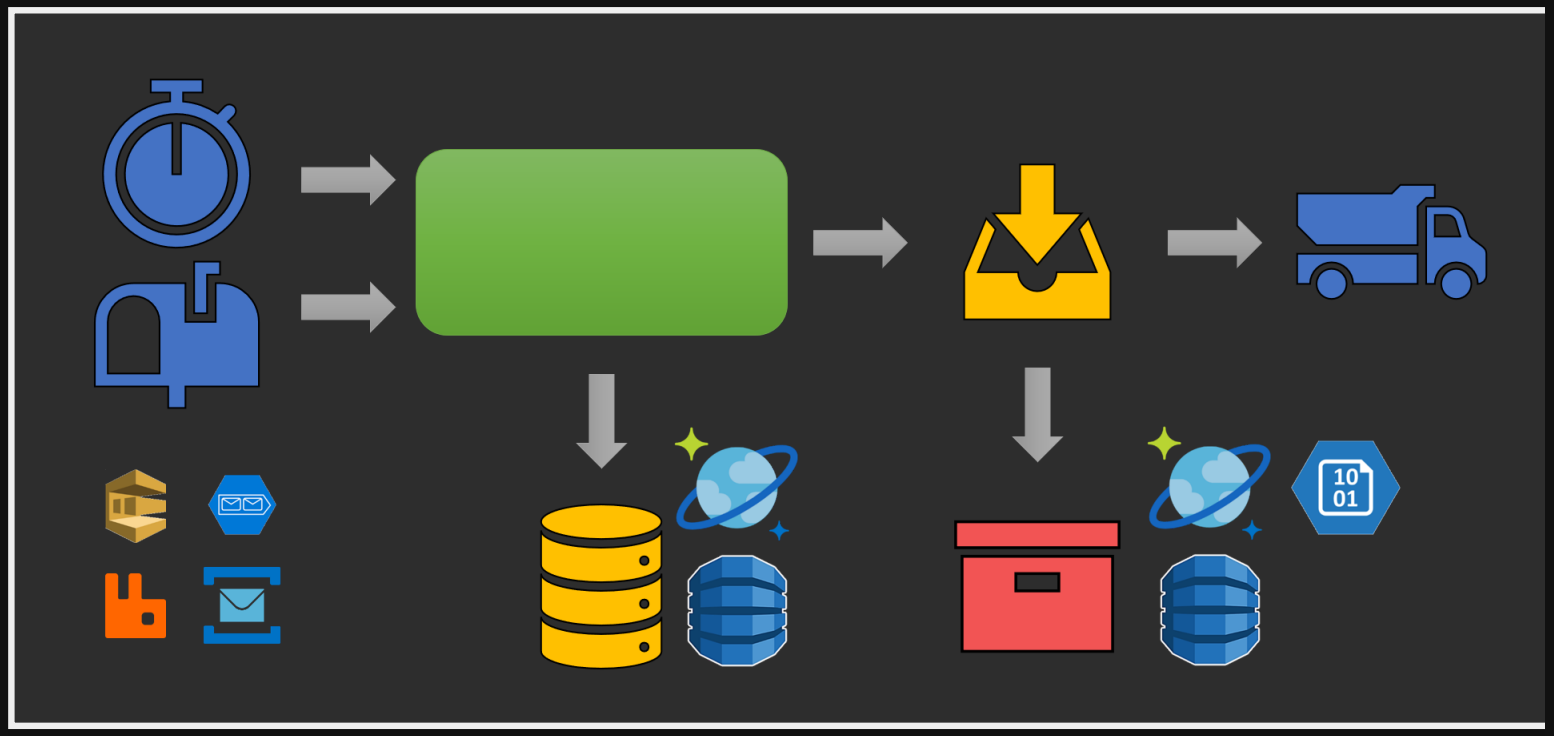
```
1  if (HasDedupeEntry(incoming.Id)) {
2          return;
3  }
4  var outbox = await Load(incoming.Id);
5  if (outbox == null) {
6          var (state, outgoing) = Process(incoming);
7          await Save(incoming.Id, state, outgoing);
8  }
9  await Dispatch(incoming.Id);
10 await AddDedupeEntry(incoming.Id);🔥🔥🔥
11 await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
if (HasDedupeEntry(incoming.Id)) {
        return;
}
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
await Dispatch(incoming.Id);
await AddDedupeEntry(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```


De-duplication
De-duplication
State
Outbox
State
Outbox




10
01

Consistency

```
1 PUT /dedupe/ABC 200
2 GET /dedupe/ABC 200
3
4 GET /dedupe/CDE 404 #HasDedupeEntry
5 PUT /dedupe/CDE 200 #AddDedupeEntry
6
7 #HasDedupeEntry
8 GET /dedupe/CDE 200
```

```
1 PUT /dedupe/ABC 200
2 GET /dedupe/ABC 200
3
4 GET /dedupe/CDE 404 #HasDedupeEntry
5 PUT /dedupe/CDE 200 #AddDedupeEntry
6
7 #HasDedupeEntry
8 GET /dedupe/CDE 200
```

```
1 PUT /dedupe/ABC 200
2 GET /dedupe/ABC 200
3
4 GET /dedupe/CDE 404 #HasDedupeEntry
5 PUT /dedupe/CDE 200 #AddDedupeEntry
6
7 #HasDedupeEntry
8 GET /dedupe/CDE 200
```

```
1 PUT /dedupe/ABC 200
2 GET /dedupe/ABC 200
3
4 GET /dedupe/CDE 404 #HasDedupeEntry
5 PUT /dedupe/CDE 200 #AddDedupeEntry
6
7 #HasDedupeEntry
8 GET /dedupe/CDE 200
```

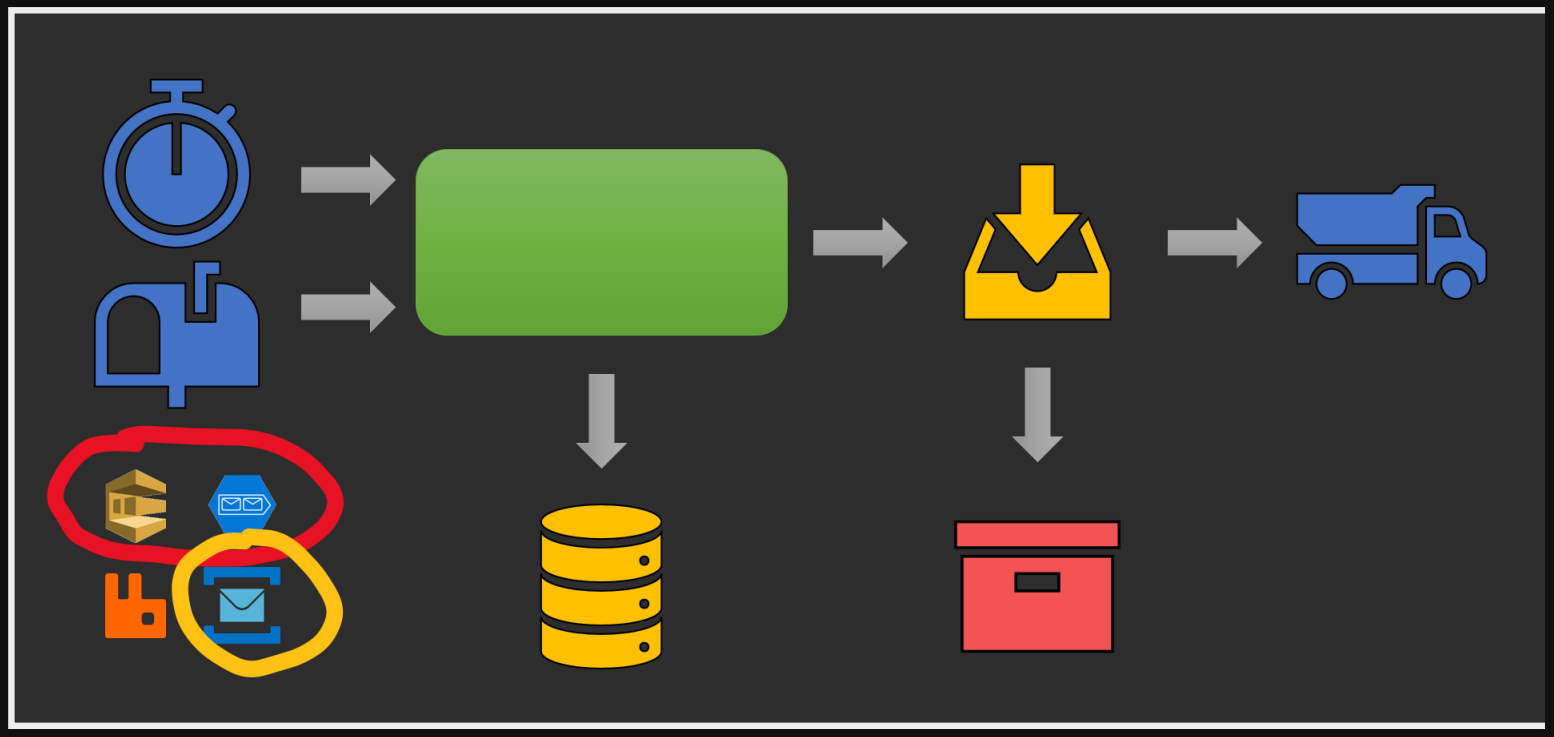
```
1 PUT /dedupe/ABC 200
2 GET /dedupe/ABC 200
3
4 GET /dedupe/CDE 404 #HasDedupeEntry
5 PUT /dedupe/CDE 200 #AddDedupeEntry
6
7 #HasDedupeEntry
8 GET /dedupe/CDE 200
```

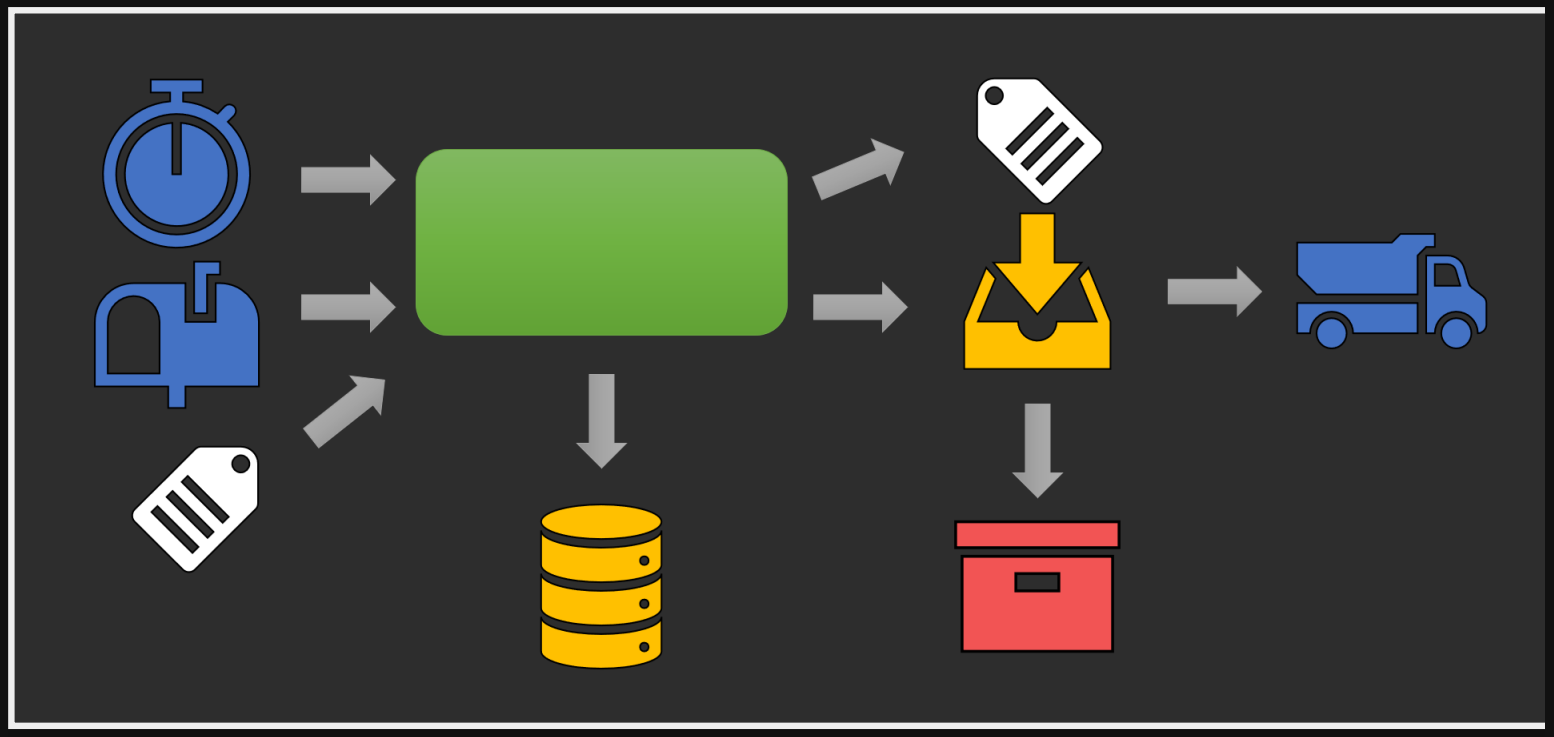
```
1 PUT /dedupe/ABC 200
2 GET /dedupe/ABC 200
3
4 GET /dedupe/CDE 404 #HasDedupeEntry
5 PUT /dedupe/CDE 200 #AddDedupeEntry
6
7 #HasDedupeEntry
8 GET /dedupe/CDE 200
```

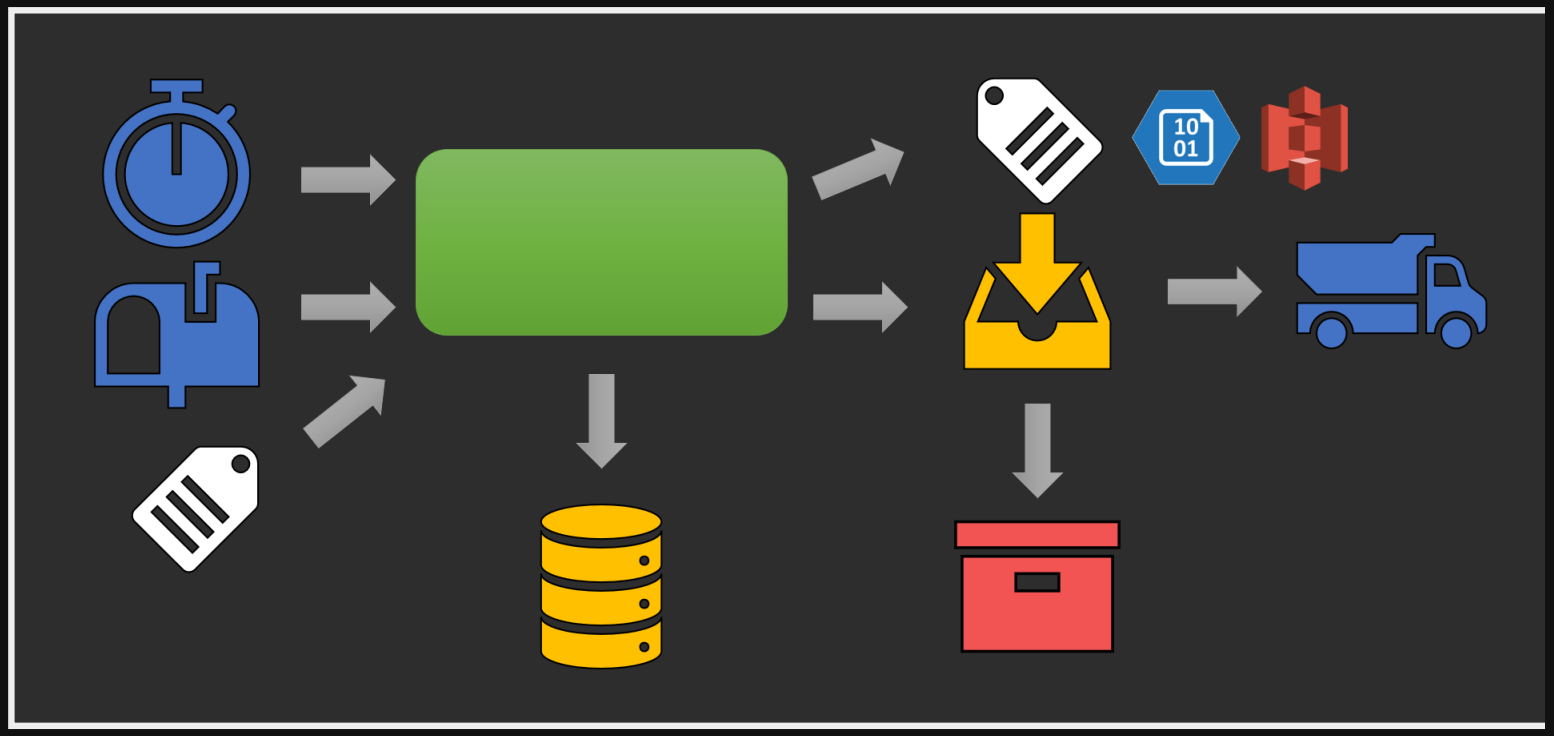
RTFM

Amazon S3 provides read-after-write consistency for PUTS of new objects in your S3 bucket in all regions with one caveat. The caveat is that if you make a HEAD or GET request to the key name (to find if the object exists) before creating the object, Amazon S3 provides eventual consistency for read-after-write.

Enterprise?






10
01

How many times can you delete a thing?

```
var payload = await GetPayload(incoming.Id);
if (payload == null) {
        return;
}
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

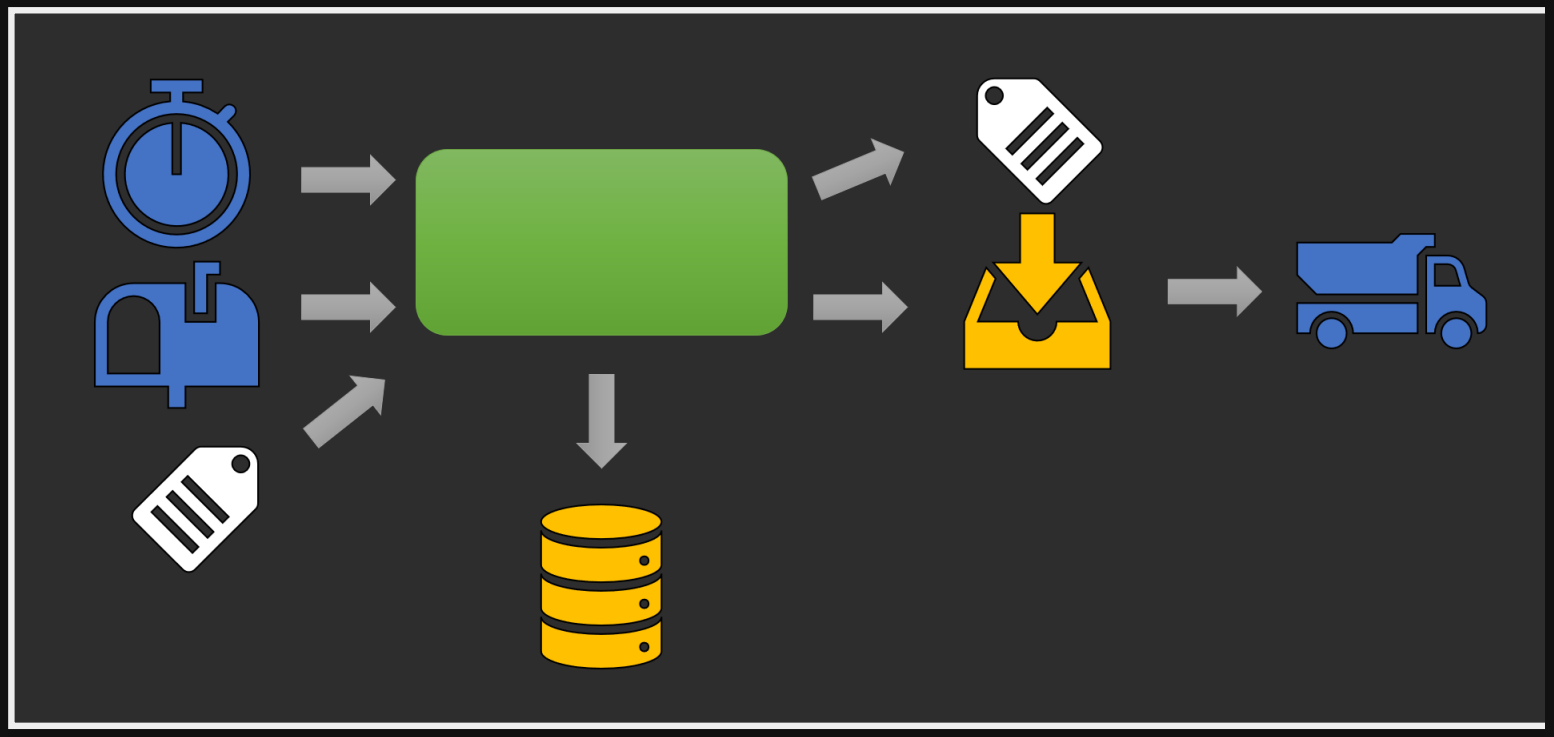
```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

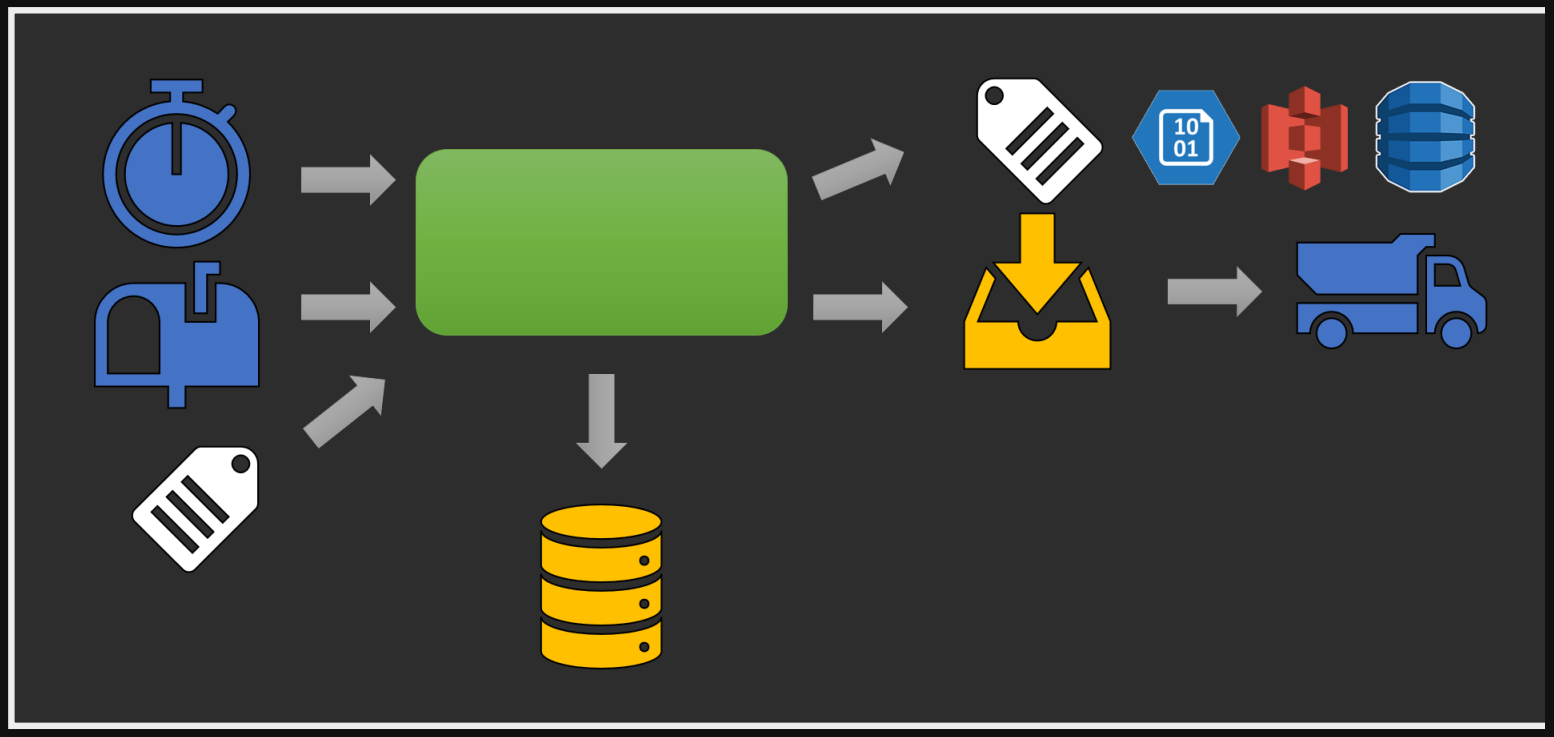
```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```

```
var outbox = await Load(incoming.Id);
if (outbox == null) {
        var (state, outgoing) = Process(incoming);
        await Save(incoming.Id, state, outgoing);
}
if (!outbox.Checked) {
        await CheckPayloads(outgoing);
        await MarkAsChecked(incoming.Id);
}
await Dispatch(incoming.Id);
await RemovePayload(incoming.Id);🔥🔥🔥
await RemoveOutbox(incoming.Id);🔥🔥🔥
```




10
01

```
1 PUT /dedupe/ABC 200 #CheckPayloads
2
3 GET /dedupe/ABC 200 #GetPayload
4 DELETE /dedupe/ABC 200 #RemovePayload
5
6 #GetPayload
7 GET /dedupe/ABC 200 
```

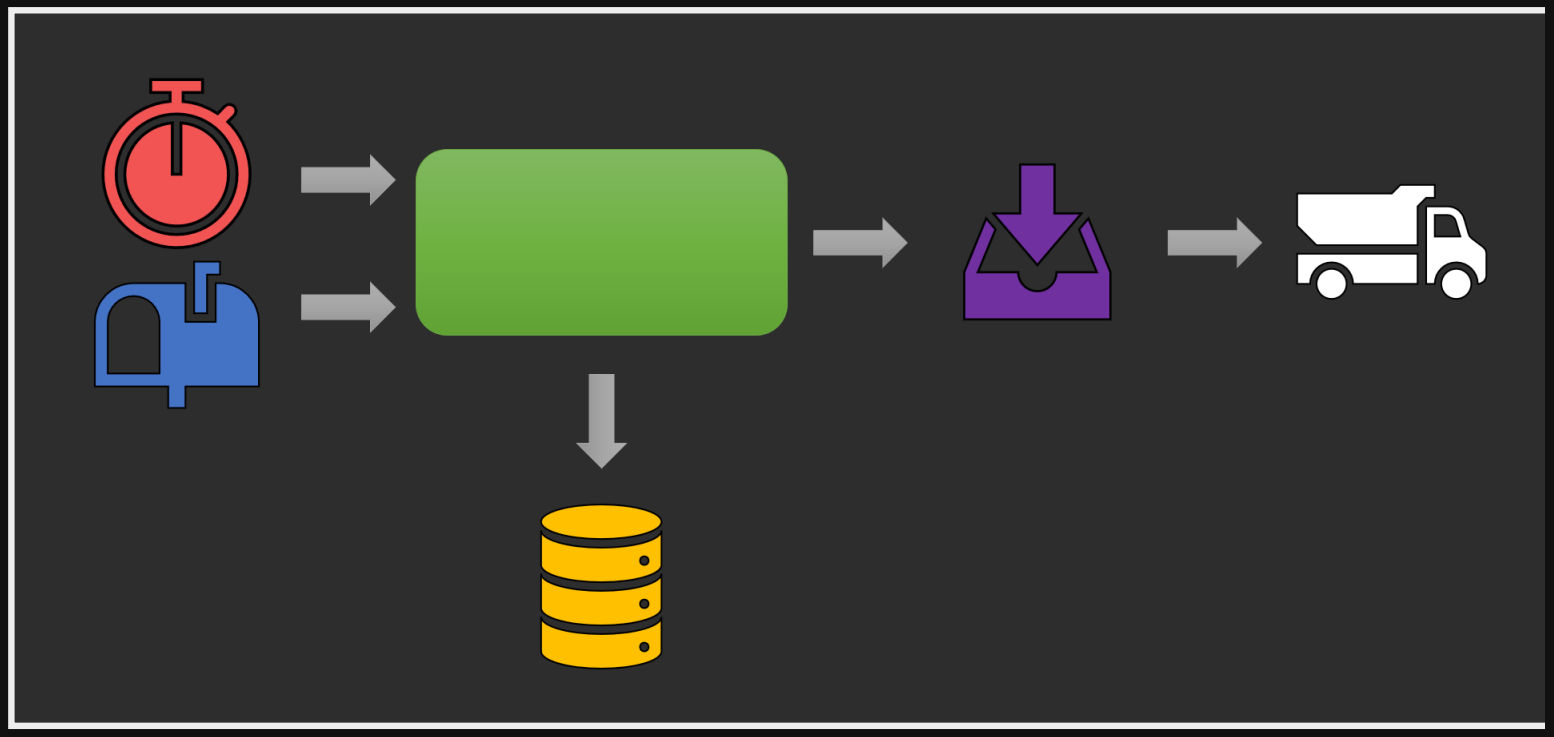
```
1 PUT /dedupe/ABC 200 #CheckPayloads
2
3 GET /dedupe/ABC 200 #GetPayload
4 DELETE /dedupe/ABC 200 #RemovePayload
5
6 #GetPayload
7 GET /dedupe/ABC 200
```

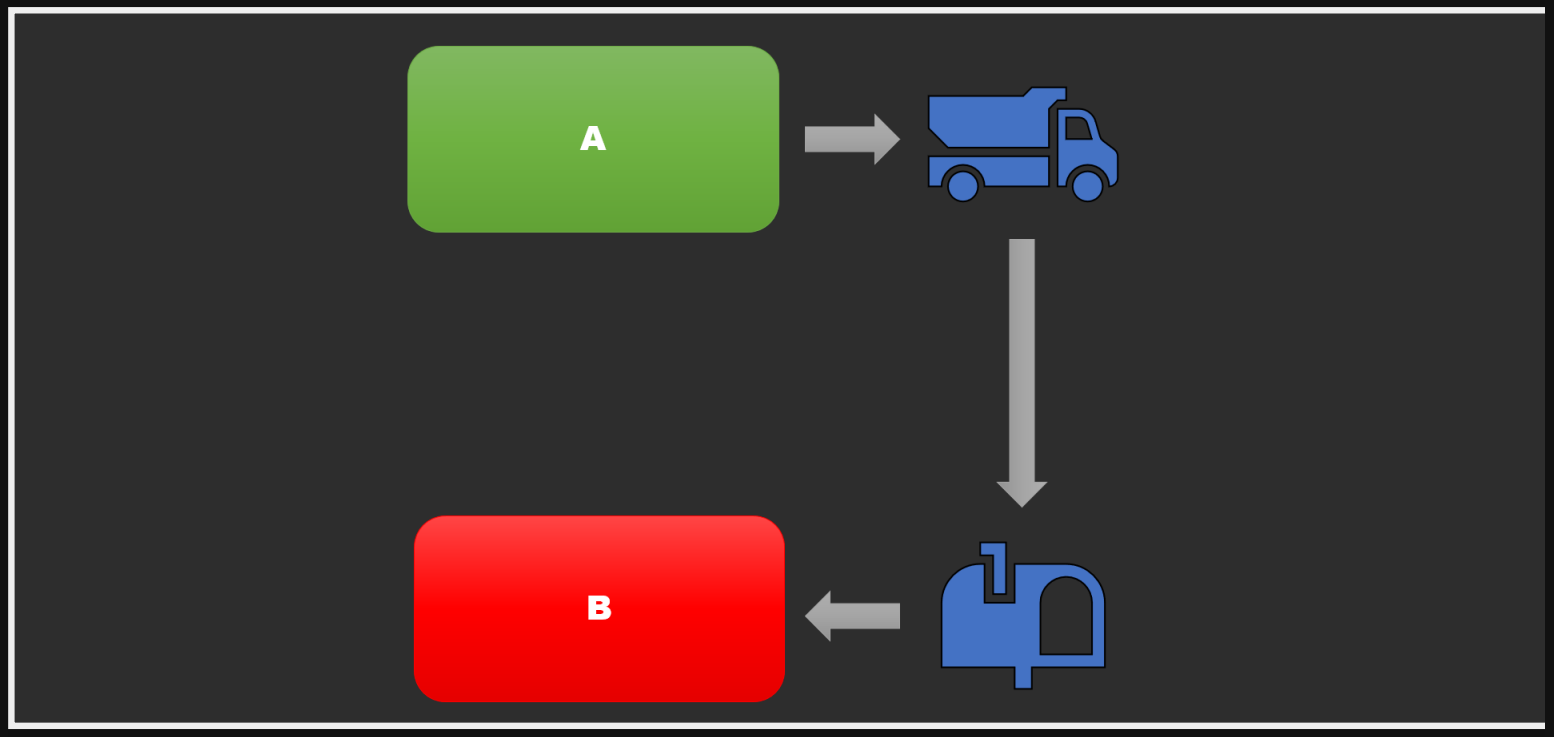
```
1 PUT /dedupe/ABC 200 #CheckPayloads
2
3 GET /dedupe/ABC 200 #GetPayload
4 DELETE /dedupe/ABC 200 #RemovePayload
5
6 #GetPayload
7 GET /dedupe/ABC 200 
```

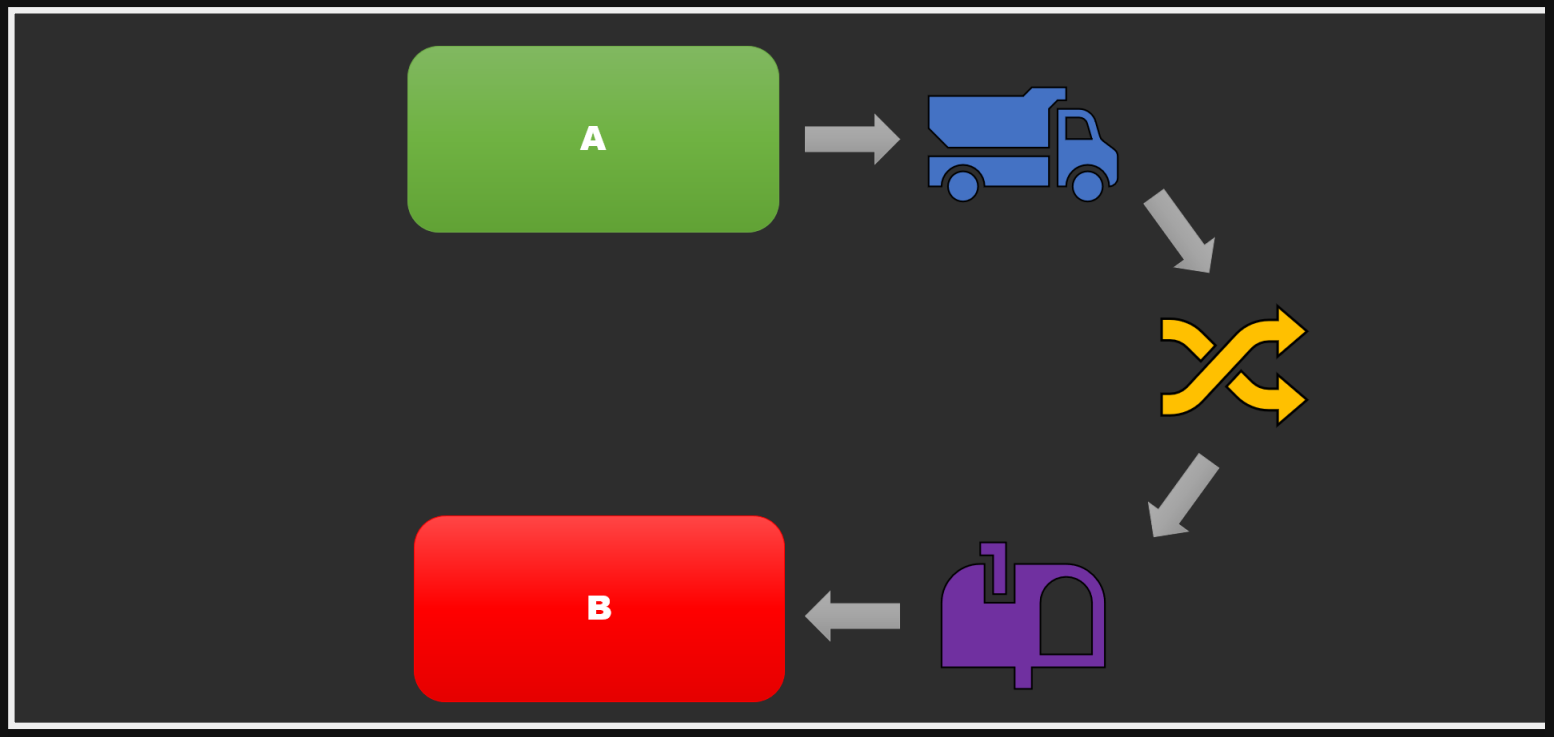
```
1 PUT /dedupe/ABC 200 #CheckPayloads
2
3 GET /dedupe/ABC 200 #GetPayload
4 DELETE /dedupe/ABC 200 #RemovePayload
5
6 #GetPayload
7 GET /dedupe/ABC 200 
```

```
1 PUT /dedupe/ABC 200 #CheckPayloads
2
3 GET /dedupe/ABC 200 #GetPayload
4 DELETE /dedupe/ABC 200 #RemovePayload
5
6 #GetPayload
7 GET /dedupe/ABC 200 
```

When can two services communicate?




A
B


A
B

Message broker is an implementation detail

Message broker is an implementation detail

Deduplication protocol is part of the contract

Cloud requires different mental models for technology

Cloud requires different mental models for technology

Using SQL database as a message queue can get you quite far

Cloud requires different mental models for technology

Using SQL database as a message queue can get you quite far

Exactly-once message processing is possible for infinite scale databases

Thank you!

github.com/SzymonPobiega/NServiceBus.Router

exactly-once.github.io

twitter.com/SzymonPobiega

linkedin.com/in/SzymonPobiega

Exactly Once

On distributed systems by @SzymonPobiega and @Masternak

Home

© 2019. All rights reserved.

Messaging infrastructure

Tue, Nov 19, 2019

The messaging infrastructure is all the components required to exchange messages between parts of the business logic code. There are two parts to the messaging infrastructure. One is the message broker that manages the message queues (and possibly topics). The other equally important part consists of all the libraries running in the same process as the application logic, that expose the API for processing messages. Whenever an application wants to send a message, it calls the in-process part of the messaging infrastructure which, in turn, communicates with the out-of-process part.

Read More...

Notes on 2PC

Thu, Oct 24, 2019

If there's a distributed protocol every software engineer knows it's Two-Phase Commit also know as 2PC. Although, in use for several decades1, it's been in steady decline mainly due to lack of support in cloud environments. For quite some time it was a de-facto standard for building enterprise distributed systems. That said, with the cloud becoming the default deployment model, designers need to learn how to build reliable systems without it.

Read More...