Function Composition in a Serverless World

Timirah James
Developer Advocate
Platform9 Systems
@timirahj

Erwin van Eyk
Software Engineer
Platform9 Systems
@erwinvaneyk

fission

PLATFORM9

@fissionio

First, what’s FaaS?

Function-as-a-Service *enable developers to deploy parts of an application on an “as needed” basis using short-lived functions.*

Benefits of FaaS:

- Complete abstraction of servers away from the developer
- Billing based on consumption and executions, not server instance sizes
- Scaling services is simplified

What is Function Composition?

The concept of (re)using smaller functions to create complex functions.

…Super function combinations

Example App

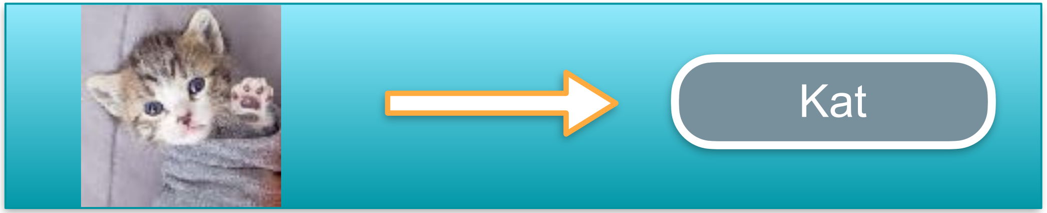
Function A

$$\begin{tikzcd}
 \arrow[r] & \boxed{\text{Recognize Image}} \arrow[r] & \text{Cat}
\end{tikzcd}$$

Function B

$$\begin{tikzcd}
\text{Cat} \arrow[r] & \boxed{\text{Translate Eng to Danish}} \arrow[r] & \text{Kat}
\end{tikzcd}$$

Can we combine both functions into one service?



Approaches

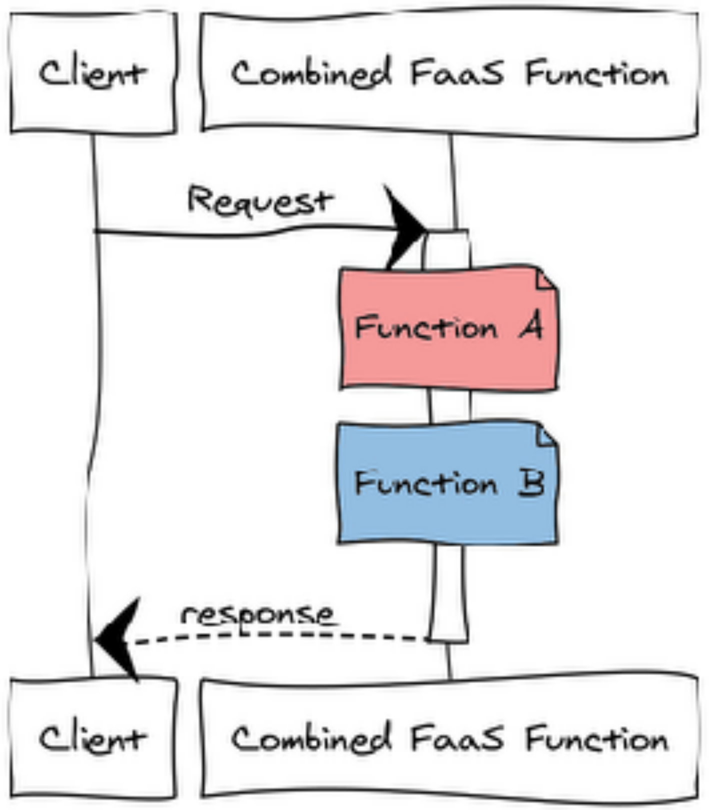
- Manual Compilation
- Direct/Chaining
- Coordinator
- Event-Driven
- Workflows

Manual Compilation

Merge functions on a source code level.

- One big function that calls all other individual functions.

- One big task from FaaS framework’s point-of-view.



```
func recognizeImage(image) {

 // A: Send the JPEG to 3rd Party AI
service for standard image tagging.

}

func translate() {

// B: Translate text from Eng to Danish

}

func combo() {
      recognize(image)
      translate()
}
```

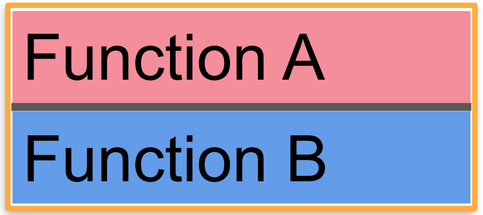
Pros:

- ✓ Very simple, no framework needed at all
- ✓ No serialization overhead

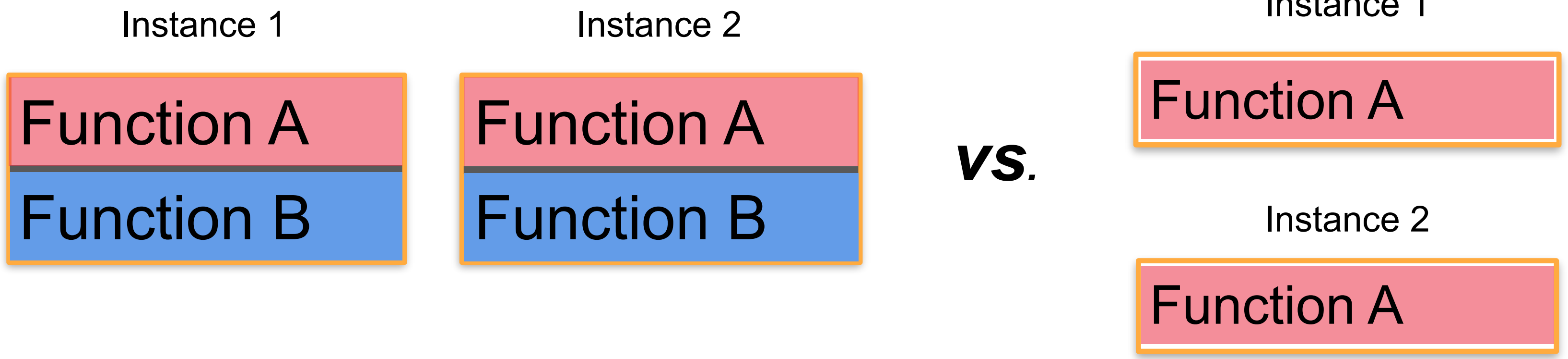
Cons:

- ✗ Function gets bigger and may load slowly
- ✗ **Cannot scale independently**

Merged Function



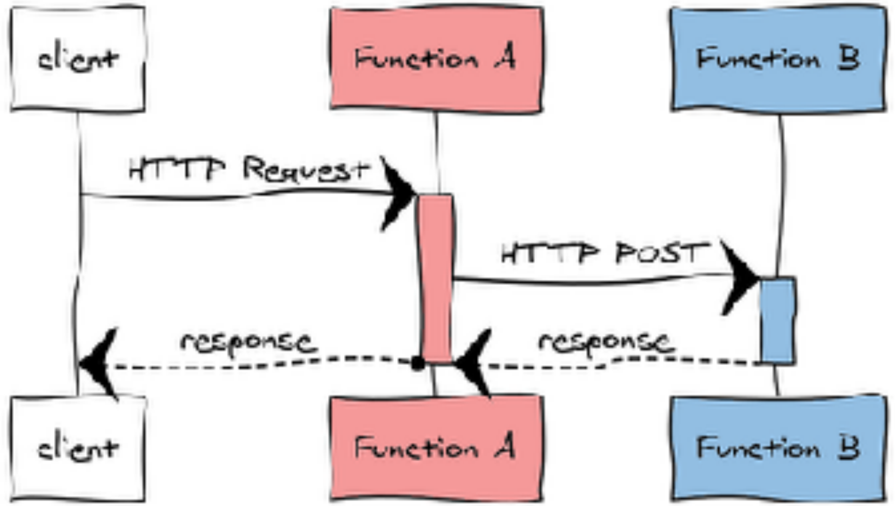
Scaling



Direct Functions (chaining)

Form a chain, calling each other.

- Each task is a separate FaaS function.
- Each function knows what comes after it and calls it.



```
func recognizeImage(image) {

 // A: Send the JPEG to 3rd Party AI
service for standard image tagging.

// HTTP call to translation function

}
```

```
func translate() {

// B: Translate text from Eng to
Danish

}
```

Pros:

✓ No external components needed

✓ No serialization overhead

Cons:

✗ Each function waits for the next function, wasting $

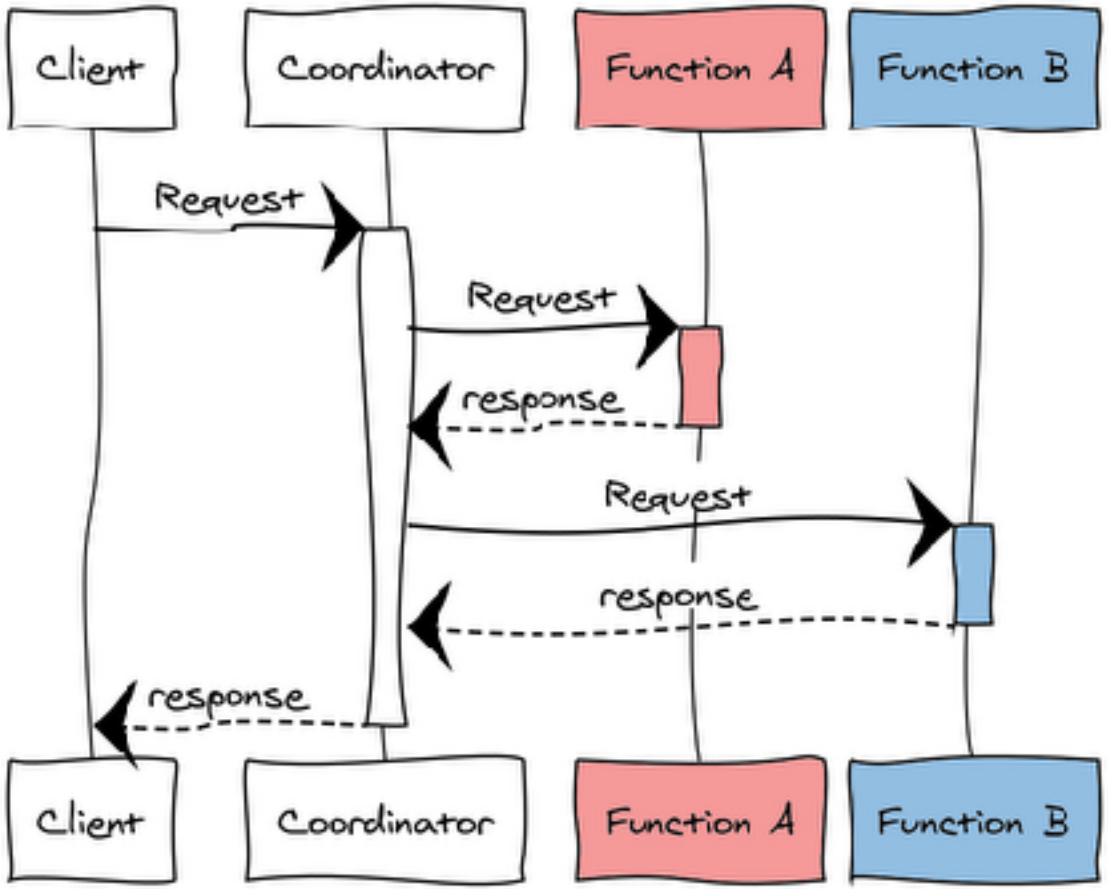
✗ Responsibility for things like handling failures, and thinking about fallbacks/retries.

✗ pains of updating a function

Coordinator Functions

Functions that manage the execution of other functions by calling them directly.

- One “omniscient” function calls each function (via remote HTTP); manages the execution flow.
- Similar to direct functions, except each function is unaware of the other functions.


Client
Coordinator
Function A
Function B
Request
Request
response
Request
response
response
Client
Coordinator
Function A
Function B

Pros:

- ✓ No need to modify the primitive functions
- ✓ Very flexible; user can manipulate the control flow how they like. (Separation of concerns)

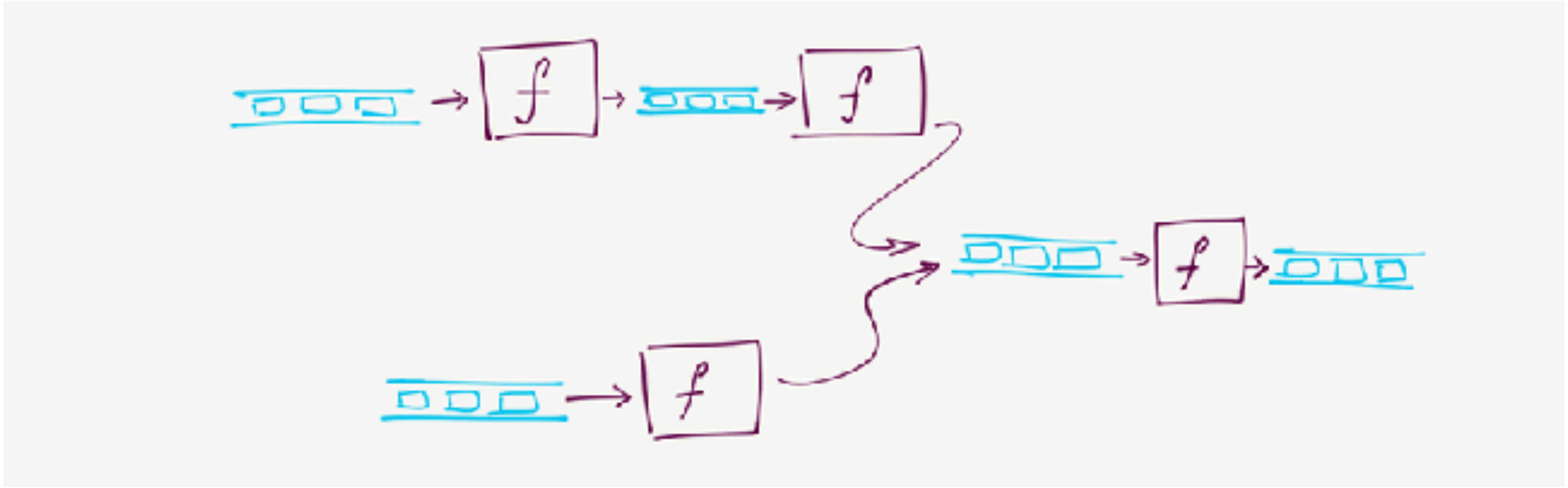
Cons:

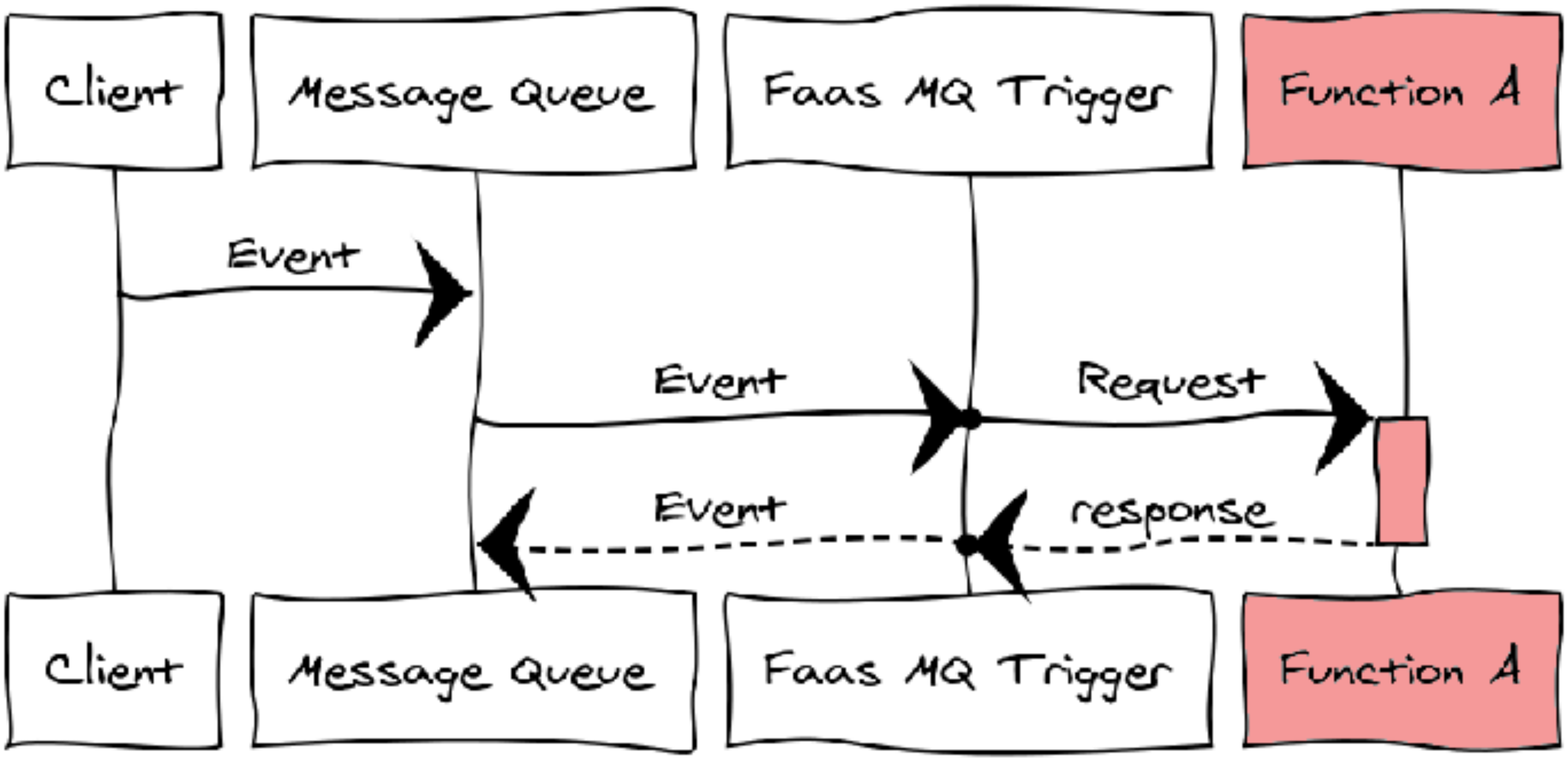
- ✗ Overhead of an extra function
- ✗ Coordinator is a long running function (it starts first, and ends last).

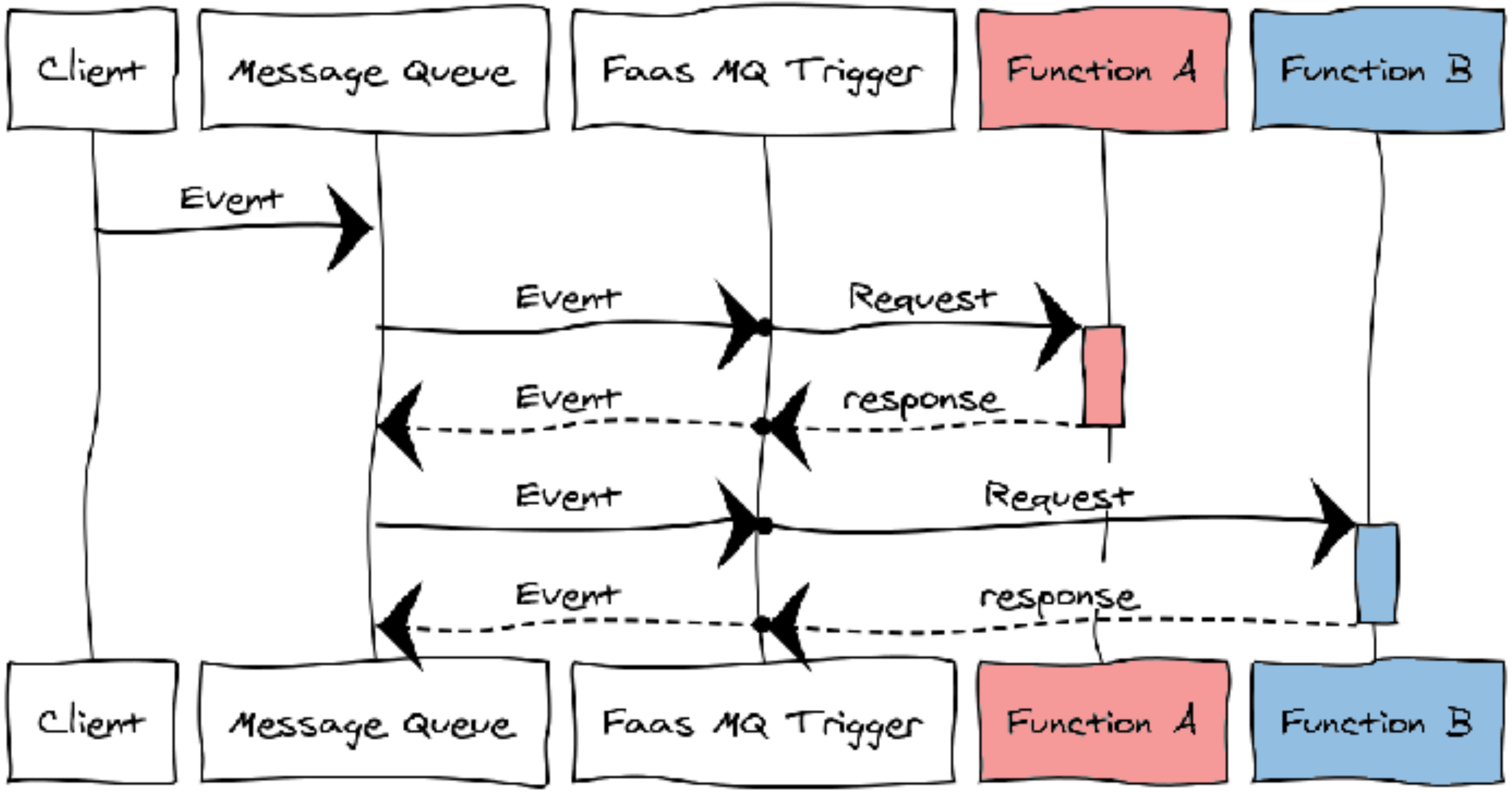
Event-Driven Function Composition

Functions emitting and reacting to events on message queues.

Idea: focus on the data flow instead of the control flow.




Client
Message Queue
Faas MQ Trigger
Function A
Event
Event
Request
Event
response
Client
Message Queue
Faas MQ Trigger
Function A


Client
Message Queue
Faas MQ Trigger
Function A
Function B
Event
Event
Request
Event
response
Event
Request
Event
response
Client
Message Queue
Faas MQ Trigger
Function A
Function B

Pros

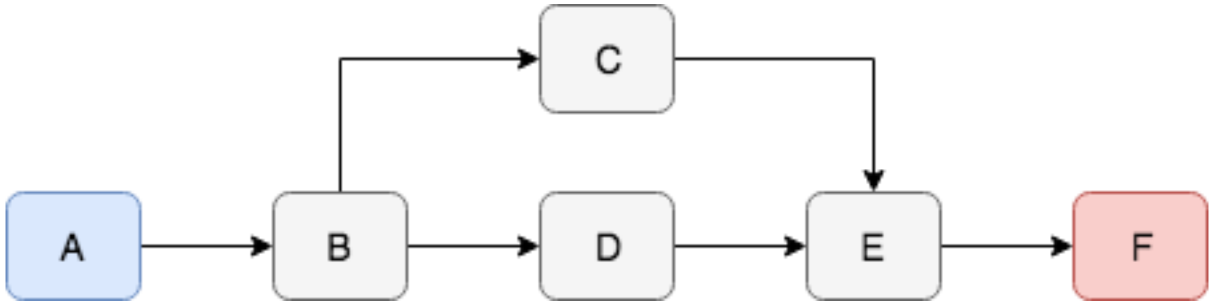
- Get all the luxury of message queues (e.g. messaging, error handling).
- Decoupled functions
- Commonly used and well understood architecture.

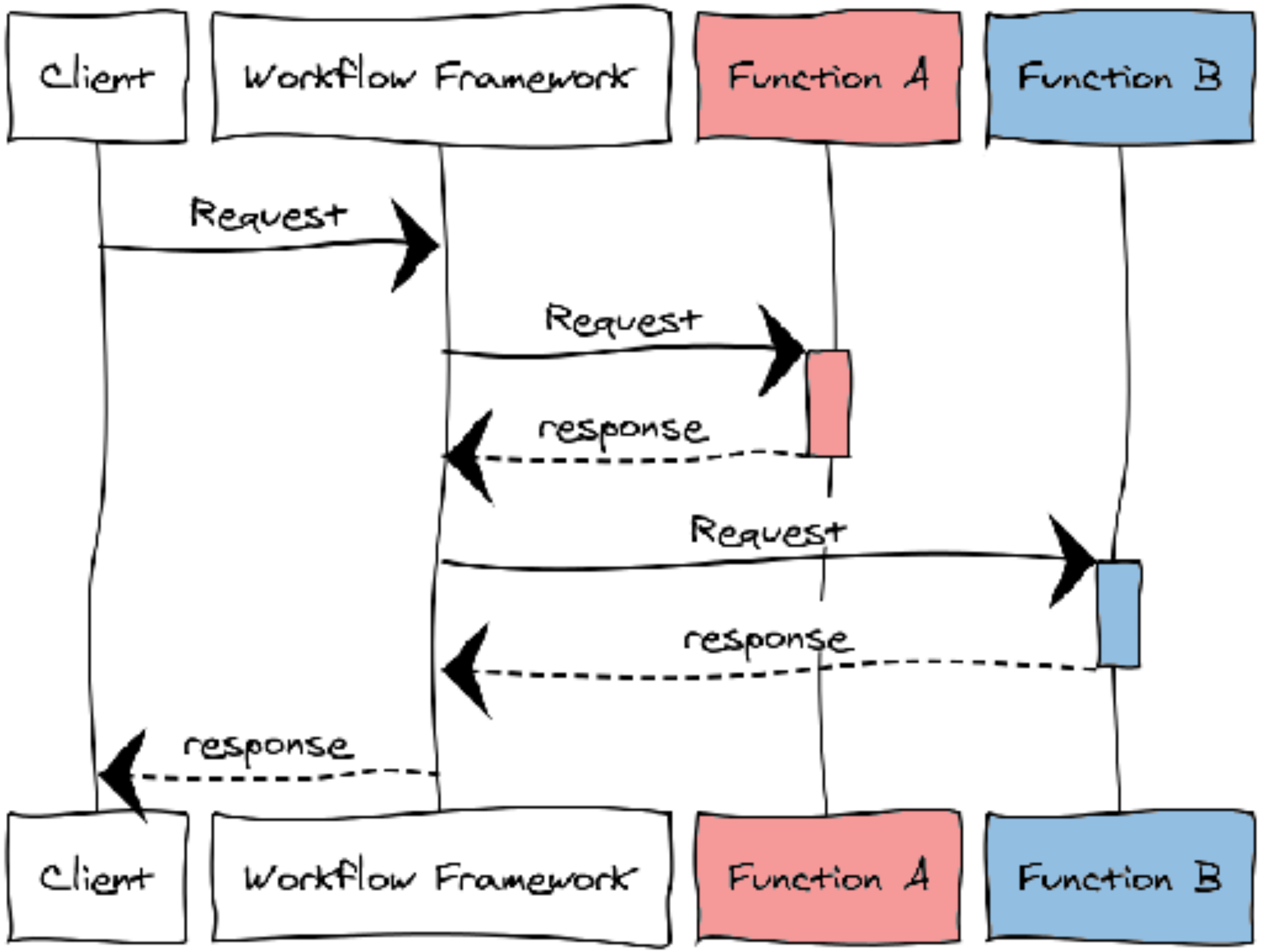
Cons

- Web of implicit dependencies.
- Difficult to version or upgrade functions.
- Supports limited control flow constructs. (e.g. conditional and on-error constructs)

Workflows

Create a “flowchart” of function interactions.




Client
Workflow Framework
Function A
Function B
Request
Request
response
Request
response
response
Client
Workflow Framework
Function A
Function B

Workflows are everywhere!

BPMN

Node-RED

camunda

luigi

Apache Airflow

argo

MISTRAL

an OpenStack Community Project

BRIGADE

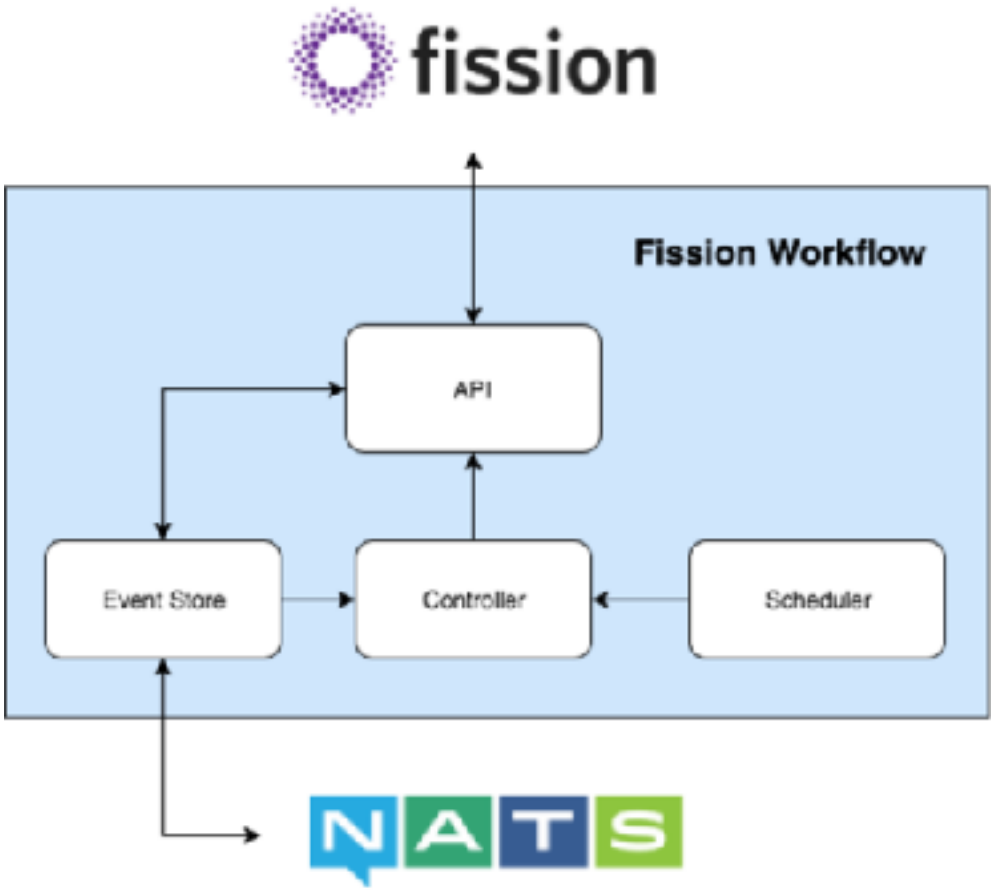
Business Processes | **Data Pipelining** | **DevOps**

FaaS-focused Workflows

AWS Step Functions

Azure Logic Apps

fission workflows



fission workflows

Deploys on Kubernetes and Fission

Stores **events** not state

Executes state machines

Demo

To follow along or to try it out:

https://github.com/fission/fission-workflows/examples/demo-kubecon2018

Pros

- Centralization of composition logic, logging, and visualization
- loosely coupled functions
- Handles communication complexity (latency, retries, failures, etc.)
- Improved performance (better/anticipating scheduling of functions)

Cons

- More infrastructure complexity
- Need to learn workflow-specific language or DSL

Approaches (recap)

- Manual Compilation
- Direct/Chaining
- Coordinator
- Event-Driven
- Workflows

Which approach should you use? 🤷‍♀️

Try them out here:
https://github.com/fission/faas-composition-patterns

Serverless is LIT!!!

THANK YOU.

Fission.io

blog.fission.io

slack.fission.io

github.com/fission

Twitter: @fissionio, @timirahj, @erwinvaneyk