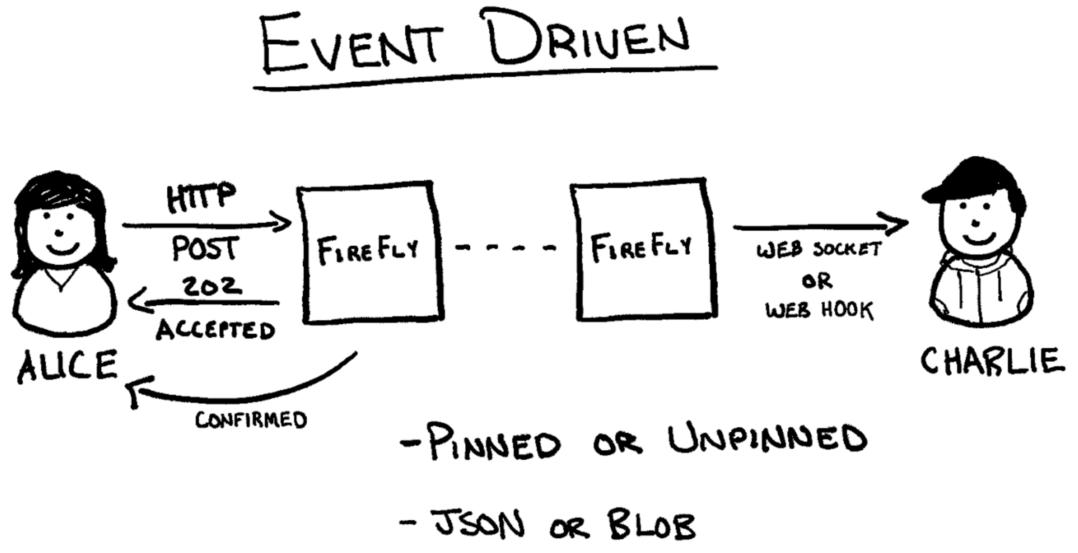
Architecture Series: Episode 7

Event-driven programming

Peter Broadhurst

Community Call 4th August 2021

FIREFLY



FIREFLY

Request/reply vs. event-driven

Request/reply (sync)

Send one request, <u>*wait*</u> for one response. Process that response when it arrives, then move onto the next thing.

App feedback: Spin until it's done
Patterns: One-to-one
Outcomes: Success, failure, timeout (undefined)
Failure handling strategies:

- Idempotent APIs – safely retry

Events (async)

Send events when something happens. Keep track of state. Process responses/confirmations/follow-ons as they happen

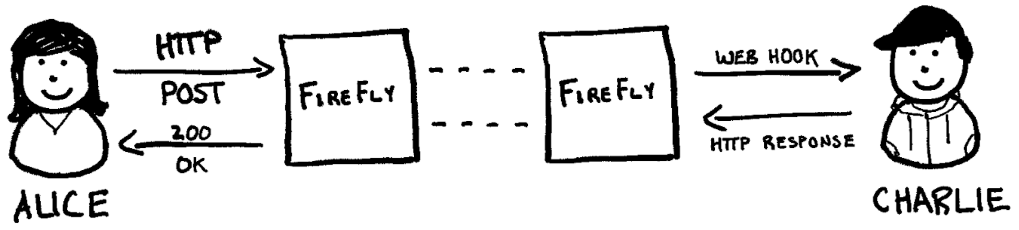
App feedback: Live update every time it changes (inc. UX)
Patterns: One-to-one, one-to-many, many-to-many, many-to-one
Outcomes: A set of state changes in a <u>*deterministic sequence*</u>
Failure handling strategies:

- Idempotent processing – re-process duplicates
- Compensation logic (sagas)
- Rejection with or without feedback

FireFly provides both

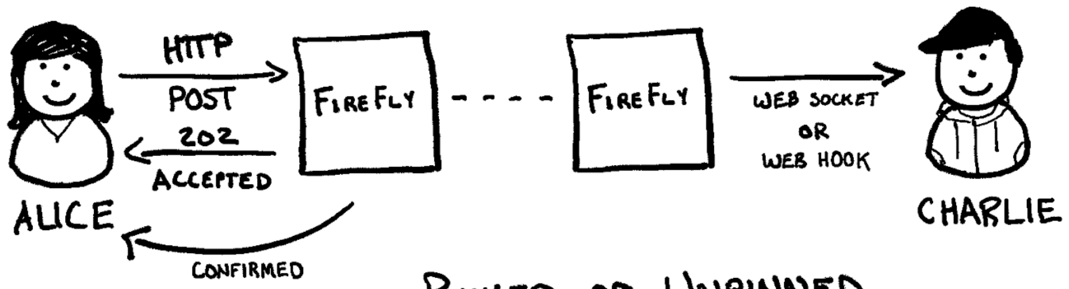
(we'll come back to this)

Request/Response



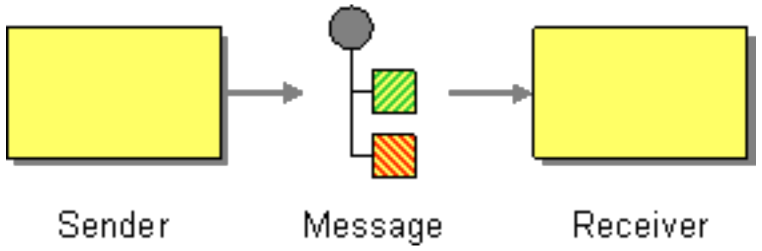
- Pinned or Unpinned
- JSON or Blob

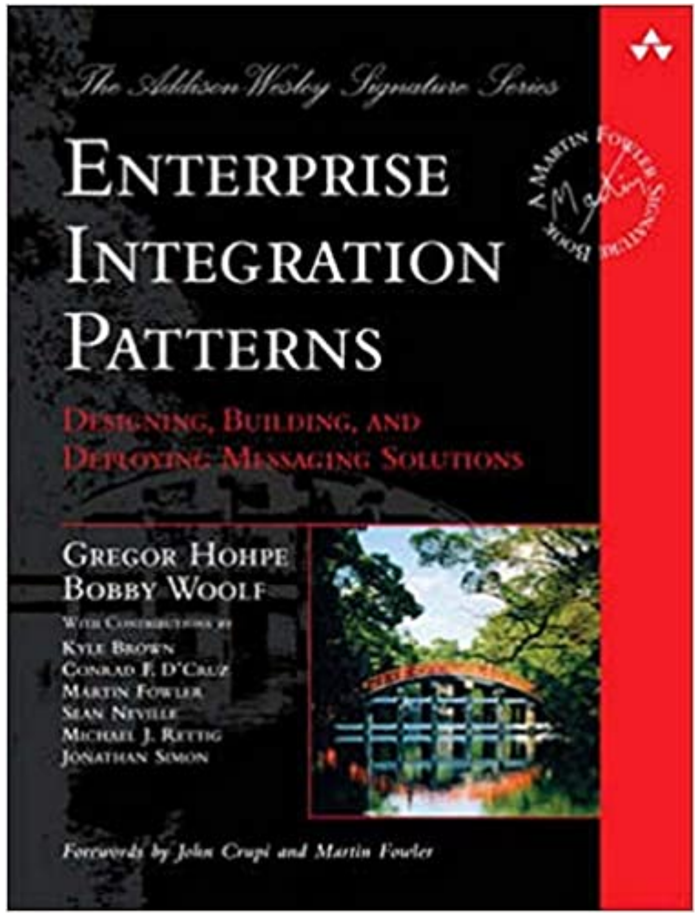
Event Driven



- Pinned or Unpinned
- JSON or Blob

Event-driven enterprise architecture isn't new…




The Addison Wesley Signature Series

ENTERPRISE INTEGRATION PATTERNS

DESIGNING, BUILDING, AND DEPLOYING MESSAGING SOLUTIONS

A MARTIN FOWLER SIGNATURE BOOK

GREGOR HOHPE
BOBBY WOOLF

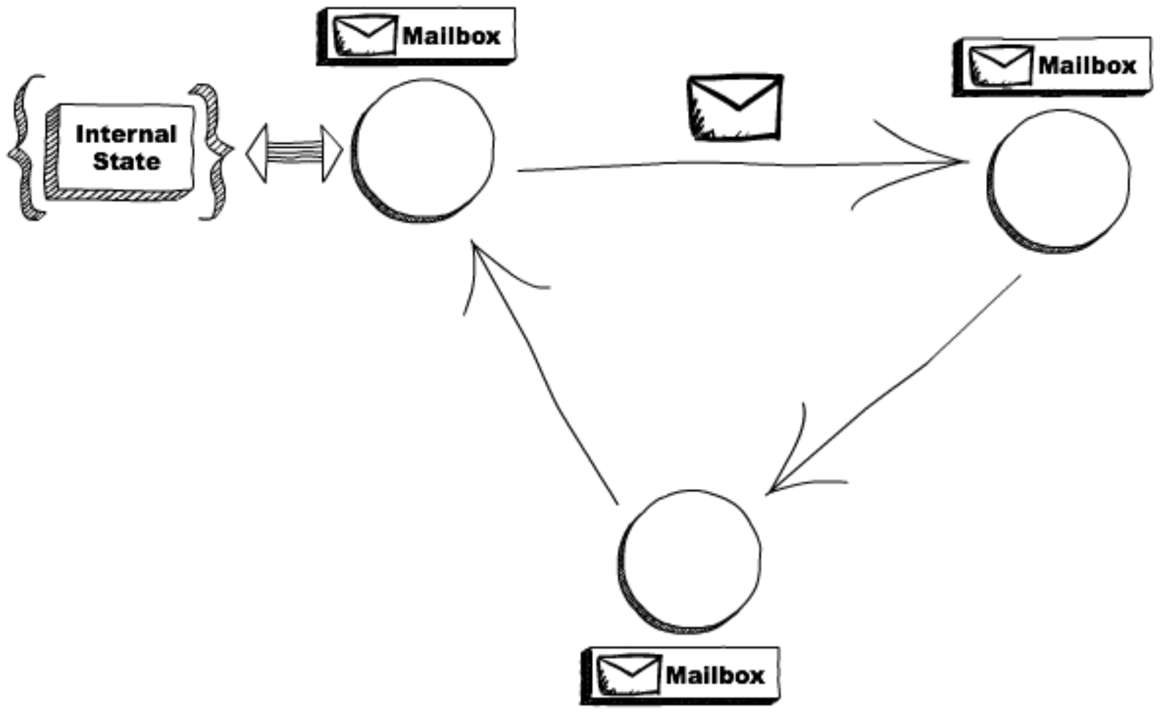
WITH CONTRIBUTIONS BY
KYLE BROWN
CONRAD F. D'CRUZ
MARTIN FOWLER
SEAN NEVILLE
MICHAEL J. RETTIG
JONATHAN SIMON

Forewords by John Crupi and Martin Fowler

Published 2003 … the year I started in this game
https://www.enterpriseintegrationpatterns.com/

FIREFLY

A decade later event-driven programming inspired a wave of new programming languages/patterns



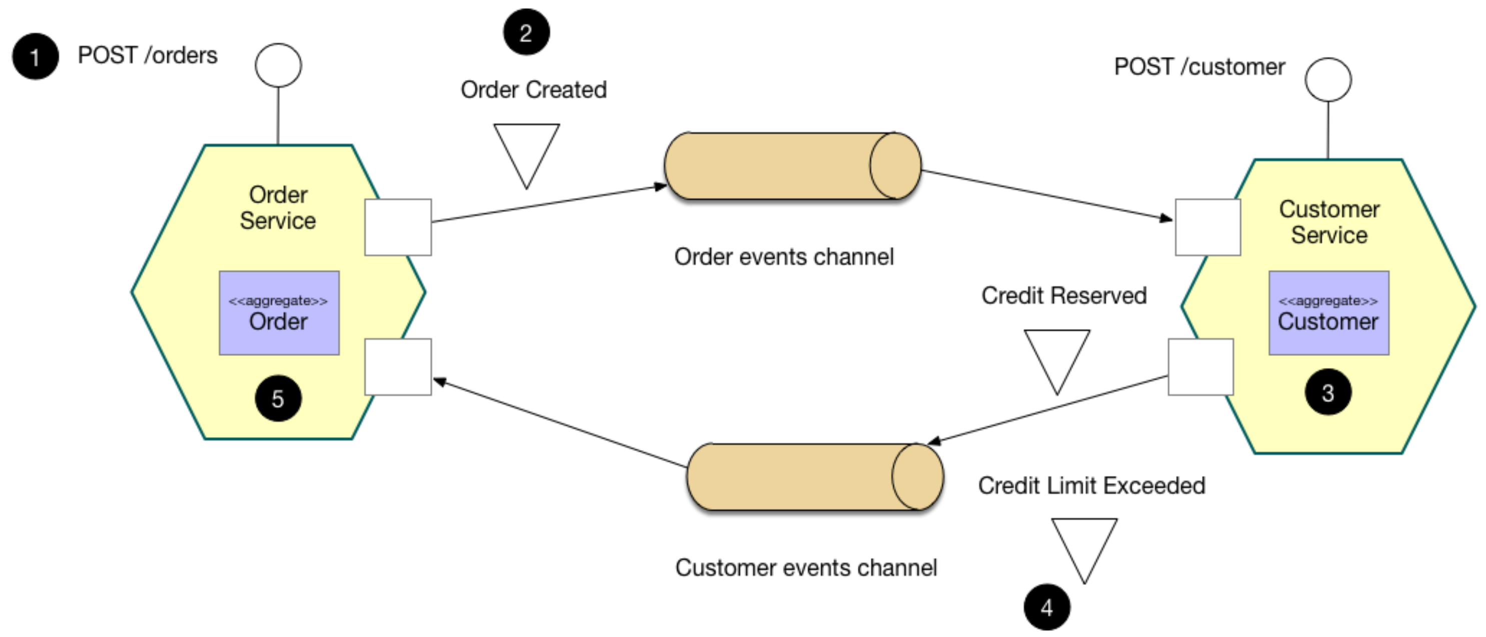
Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback.

2014: https://www.reactivemanifesto.org/

Image credit… and a good read:
https://www.brianstorti.com/the-actor-model/

FIREFLY

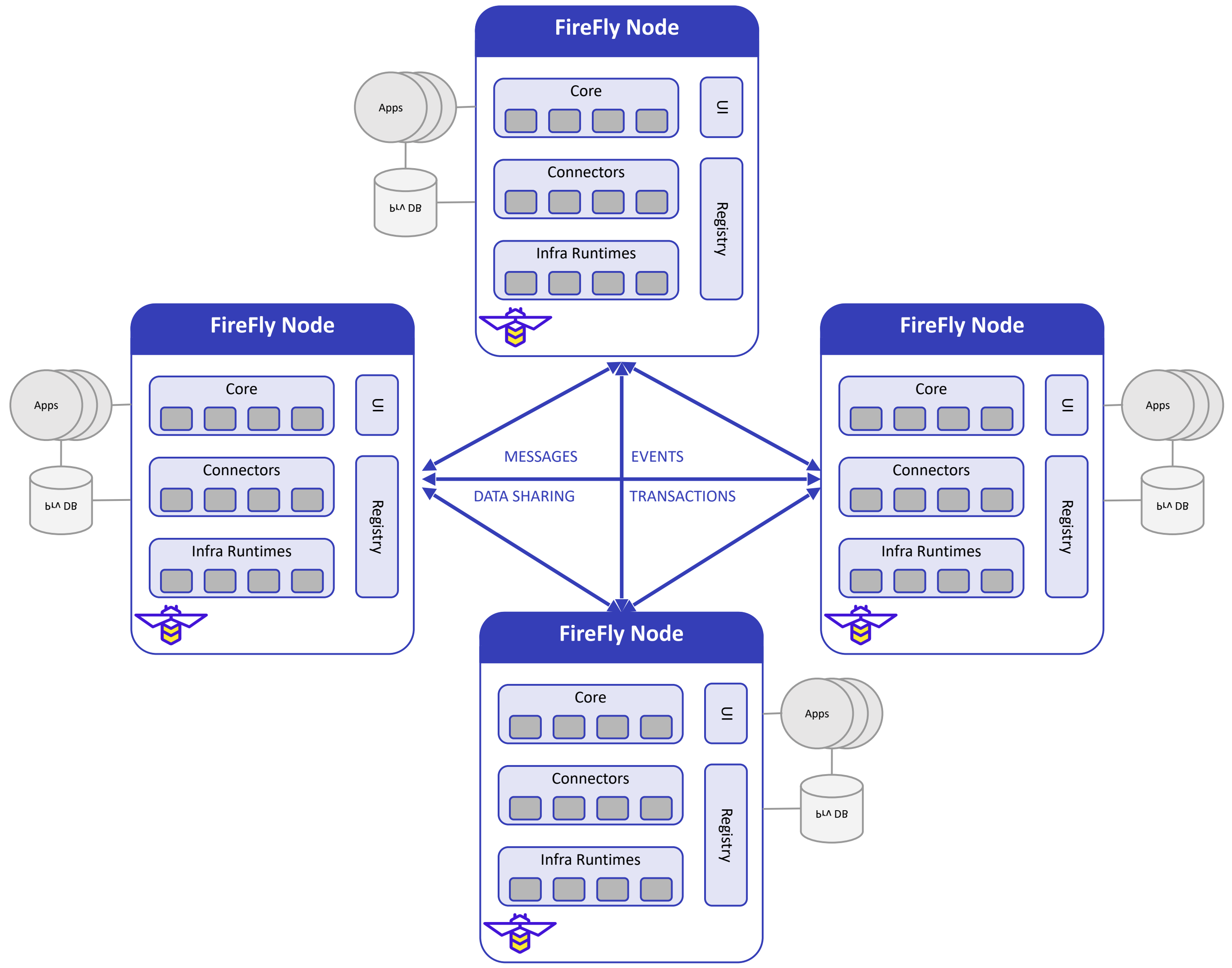
Microservice events-driven patterns evolved in a post-ACID age of REST and at-least once delivery



… in different databases owned by different services the application cannot simply use a local ACID transaction. (Chris Richardson)

2017: https://microservices.io/patterns/data/saga.html

The next phase in event-driven apps: Decentralized applications in a multi-party system



What **hasn't** changed:

- *Most* processing is member specific
  - Automated proprietary core systems
  - Human decision making
  - Agreed state transitions
- *Most* data is privately replicated
  - Governed by business needs and security
  - Stored private in each member
  - Synchronized to multiple core systems

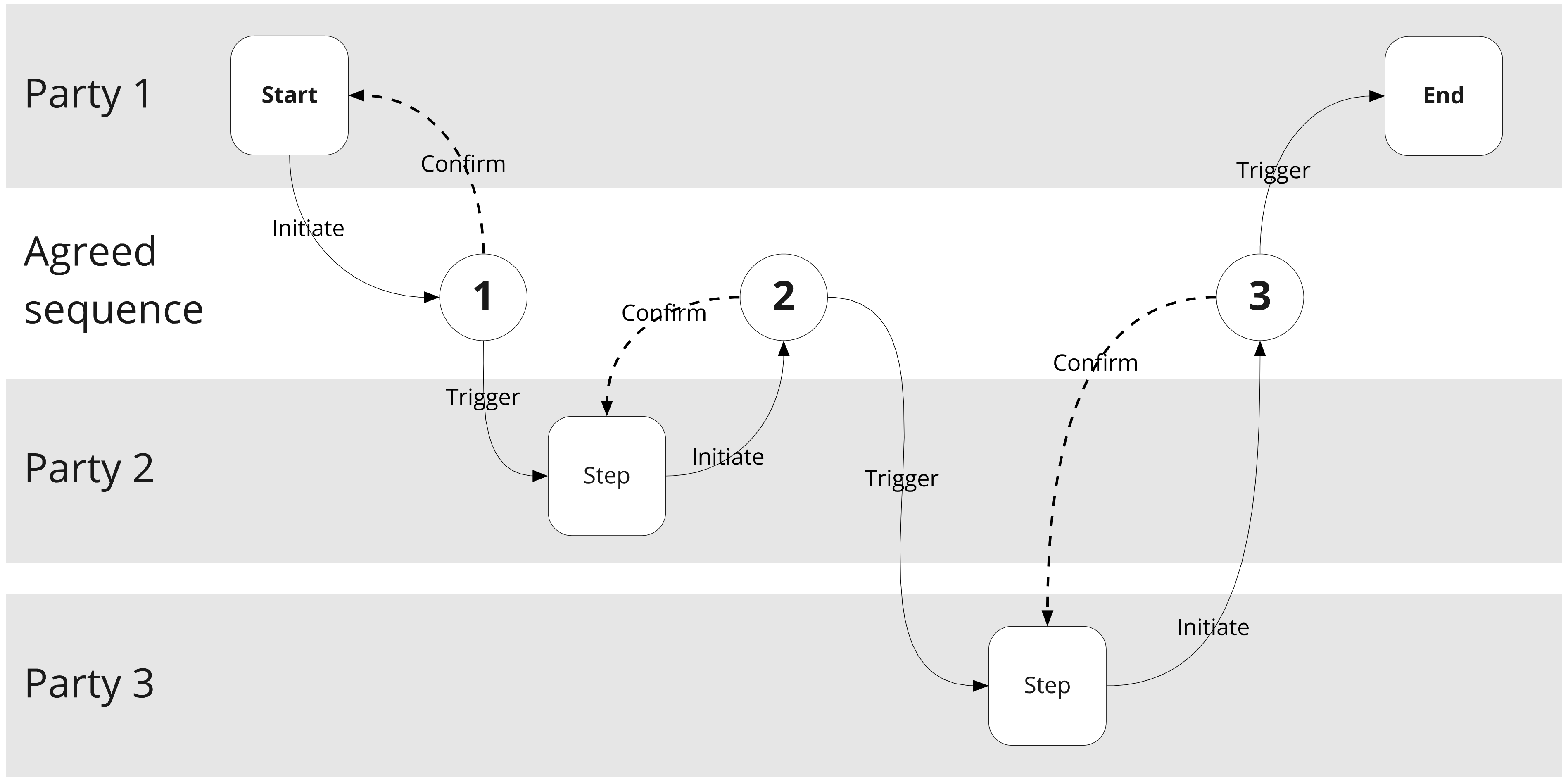
What **has** changed:

- *Some* logic can be executed deterministically
  - Blockchain
  - Trusted compute / zero-knowledge
- *Some* data/proofs can be stored centrally
  - Blockchain / IPFS
- *Multiple* parties share a single event sequence
  - This is **revolutionary** for event-driven apps

FIREFLY

Microservices to multi-party - key difference 1:
You must process your <u>own</u> events in an order shared with other members

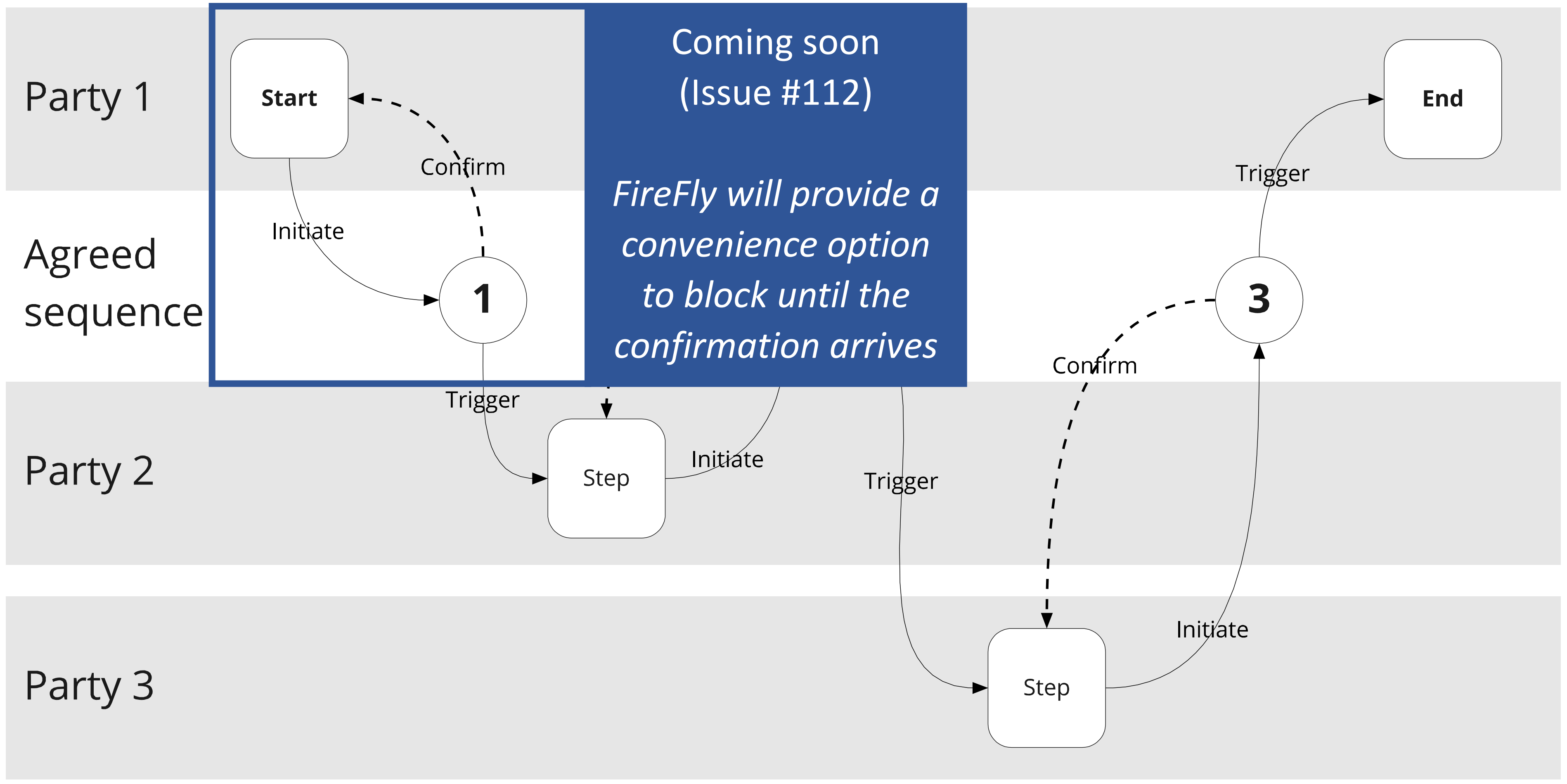
Multi-party business process flow



FIREFLY

Microservices to multi-party - key difference 1:
You must process your <u>own</u> events in an order shared with other members

Multi-party business process flow



Coming soon
(Issue #112)

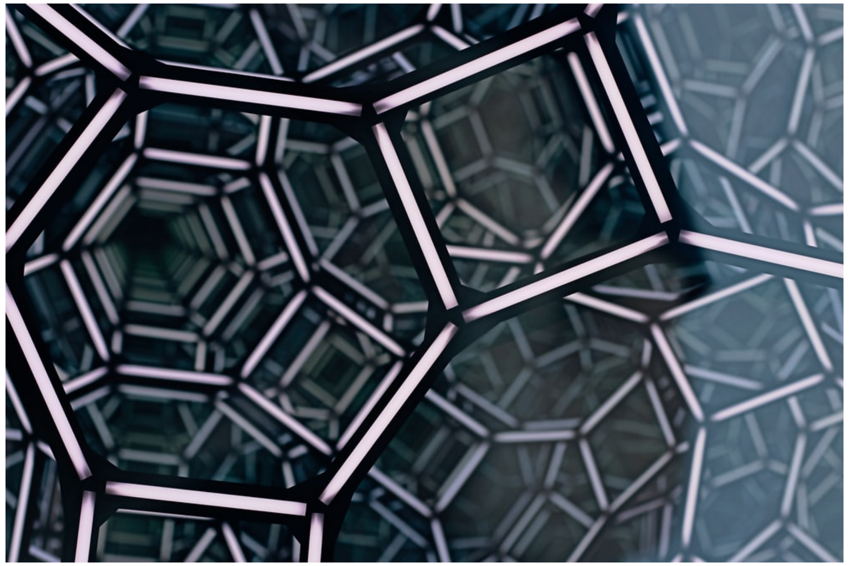
FireFly will provide a convenience option to block until the confirmation arrives

Microservices to multi-party - key difference 2:
Event history can go back to time=0 and be immutable – supporting late join/replay

Traditional Message-Queues and Streams

Store + forward data reliably

- Optimized for short-term storage (seconds/minutes)
- Optimized for low latency delivery (milliseconds)
- Capable of coping with periods of downtime (hours/days)
- Designed to decouple system availability
- Ordering is guaranteed only within a single runtime (broker)



Blockchain Ledgers

Complete history of time

- Optimized for building immutable transaction history (years/decades)
- Optimized for establishing multi-party consensus (seconds/minutes)
- Designed for wide fault tolerance – including byzantine (indefinite)
- Designed to decouple sovereign IT infrastructures
- Ordering is guaranteed globally within a ledger (blockchain/channel)

FIREFLY

Microservices to multi-party - key difference 2:
Event history can go back to time=0 and be immutable – supporting late join/replay

You need both!

Traditional Message-Queues and Streams

Store + forward data reliably

- Optimized for short-term storage (seconds/minutes)
- Optimized for low latency delivery (milliseconds)
- Capable of coping with periods of downtime (hours/days)
- Designed to decouple system availability
- Ordering is guaranteed only within a single runtime (broker)

Blockchain Ledgers

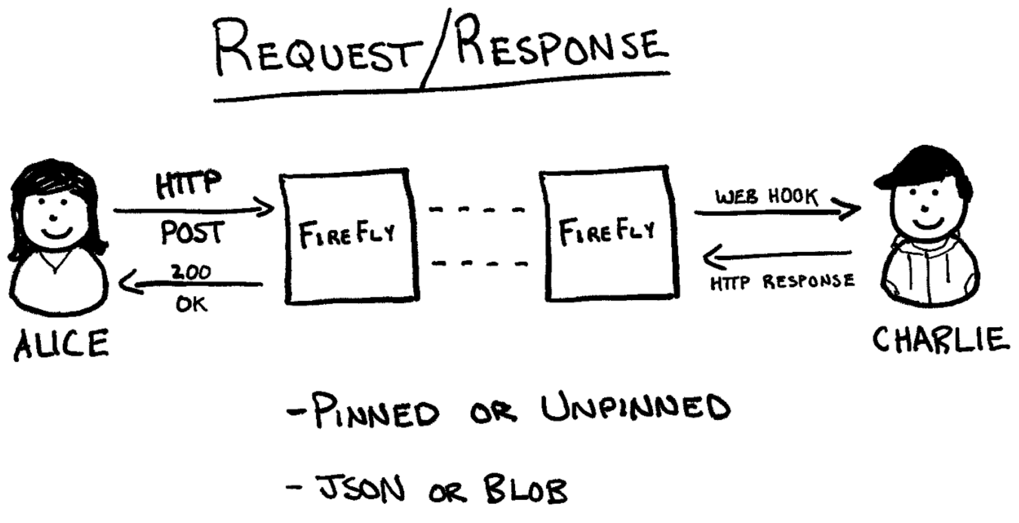
Complete history of time

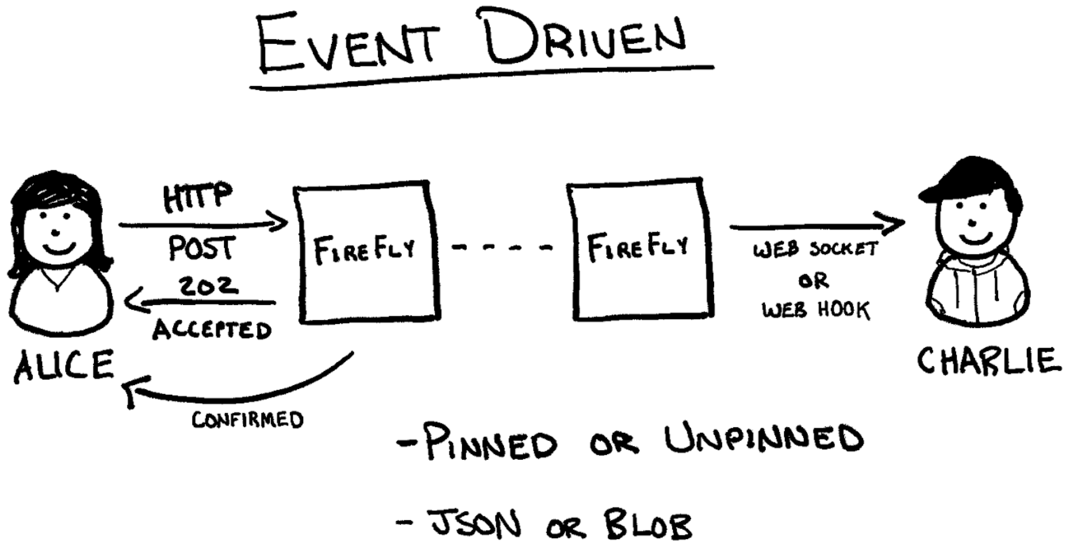
- Optimized for building immutable transaction history (years/decades)
- Optimized for establishing multi-party consensus (seconds/minutes)
- Designed for wide fault tolerance – including byzantine (indefinite)
- Designed to decouple sovereign IT infrastructures
- Ordering is guaranteed globally within a ledger (blockchain/channel)

FIREFLY

FireFly provides both

(back to the practical dev info)

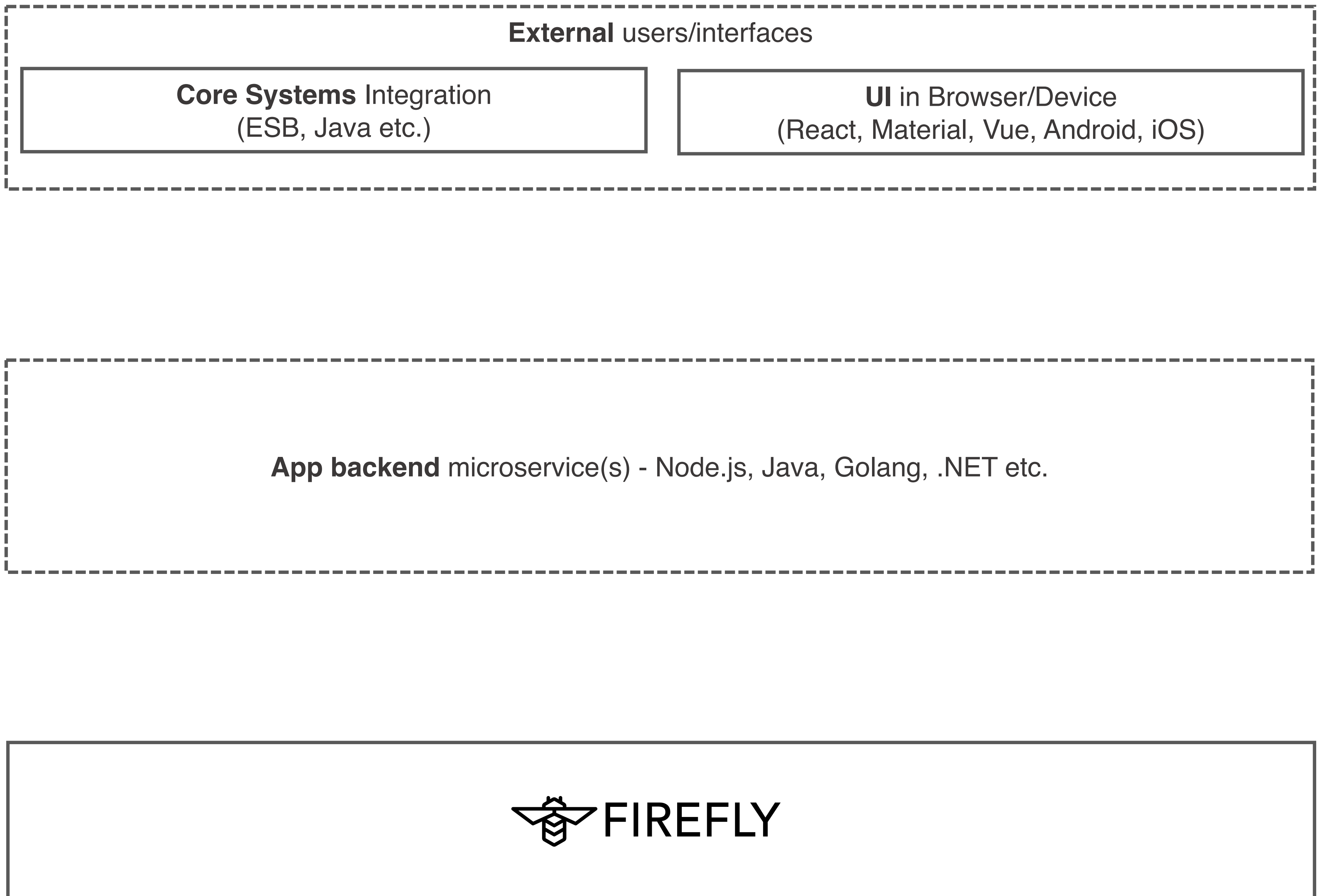




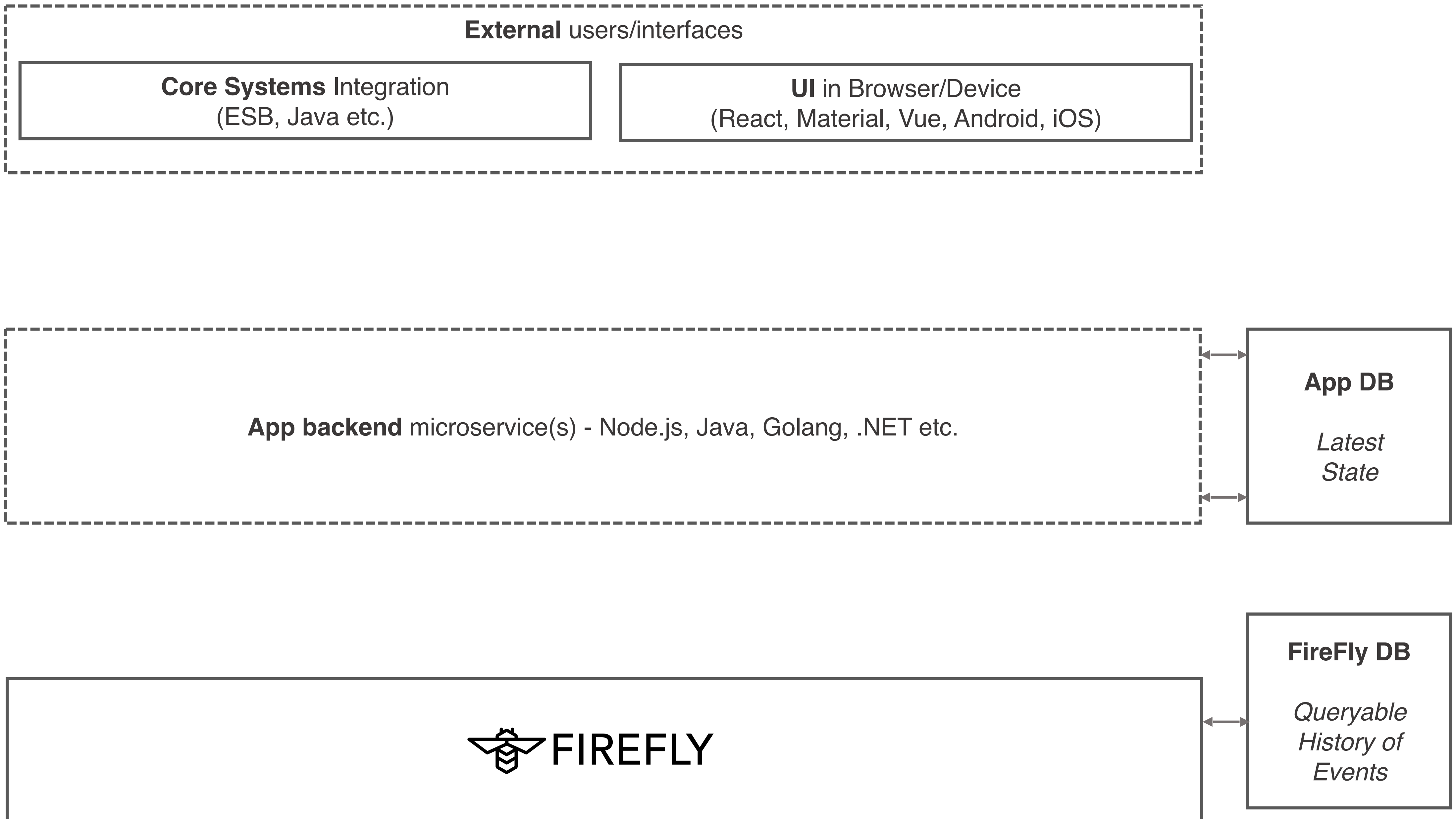
Focusing here today

FIREFLY

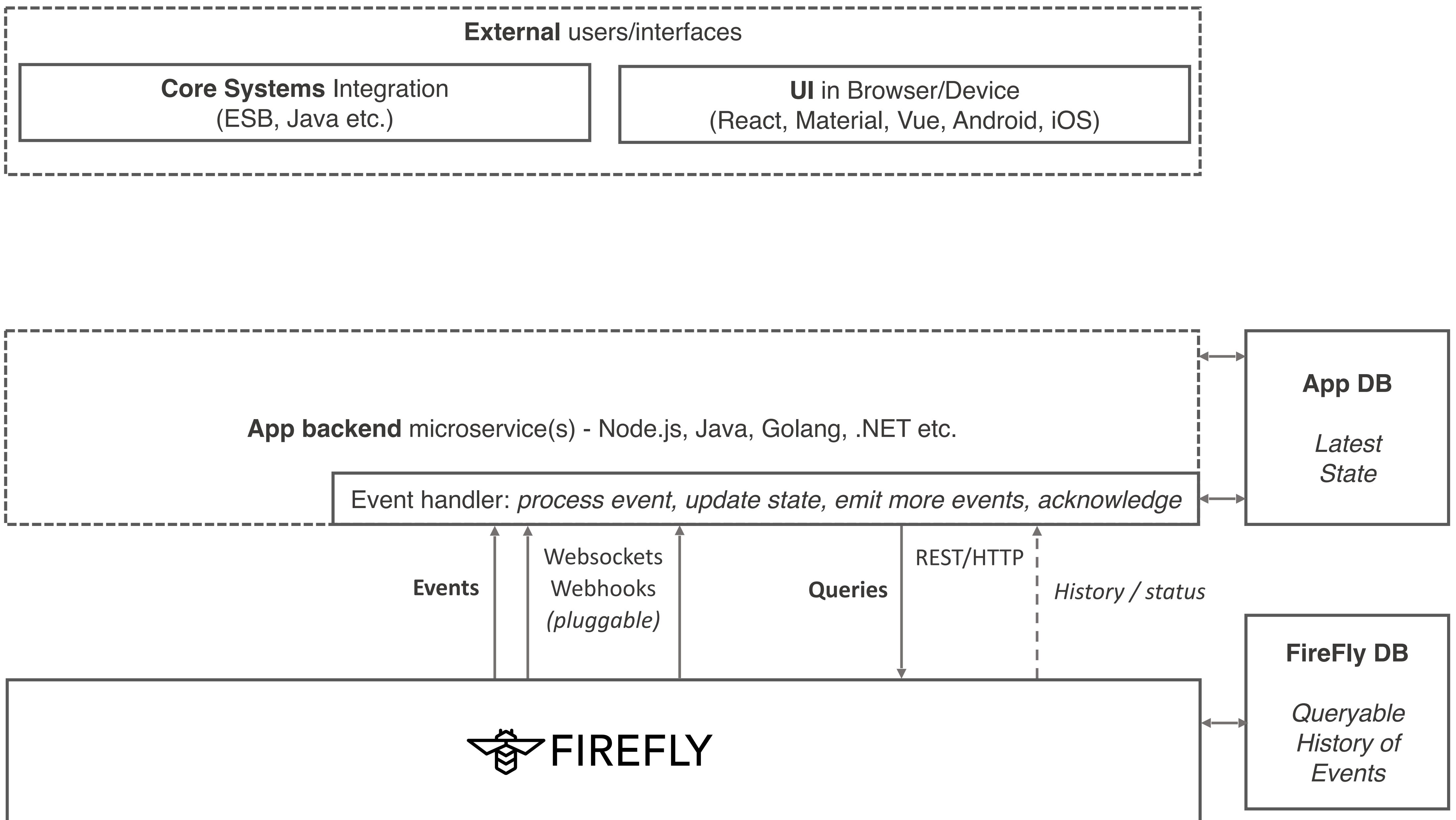
What an event-driven FireFly app looks like


External users/interfaces
Core Systems Integration (ESB, Java etc.)
UI in Browser/Device (React, Material, Vue, Android, iOS)
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
FIREFLY

What an event-driven FireFly app looks like


External users/interfaces
Core Systems Integration (ESB, Java etc.)
UI in Browser/Device (React, Material, Vue, Android, iOS)
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
App DB
Latest State
FireFly DB
Queryable History of Events
FIREFLY

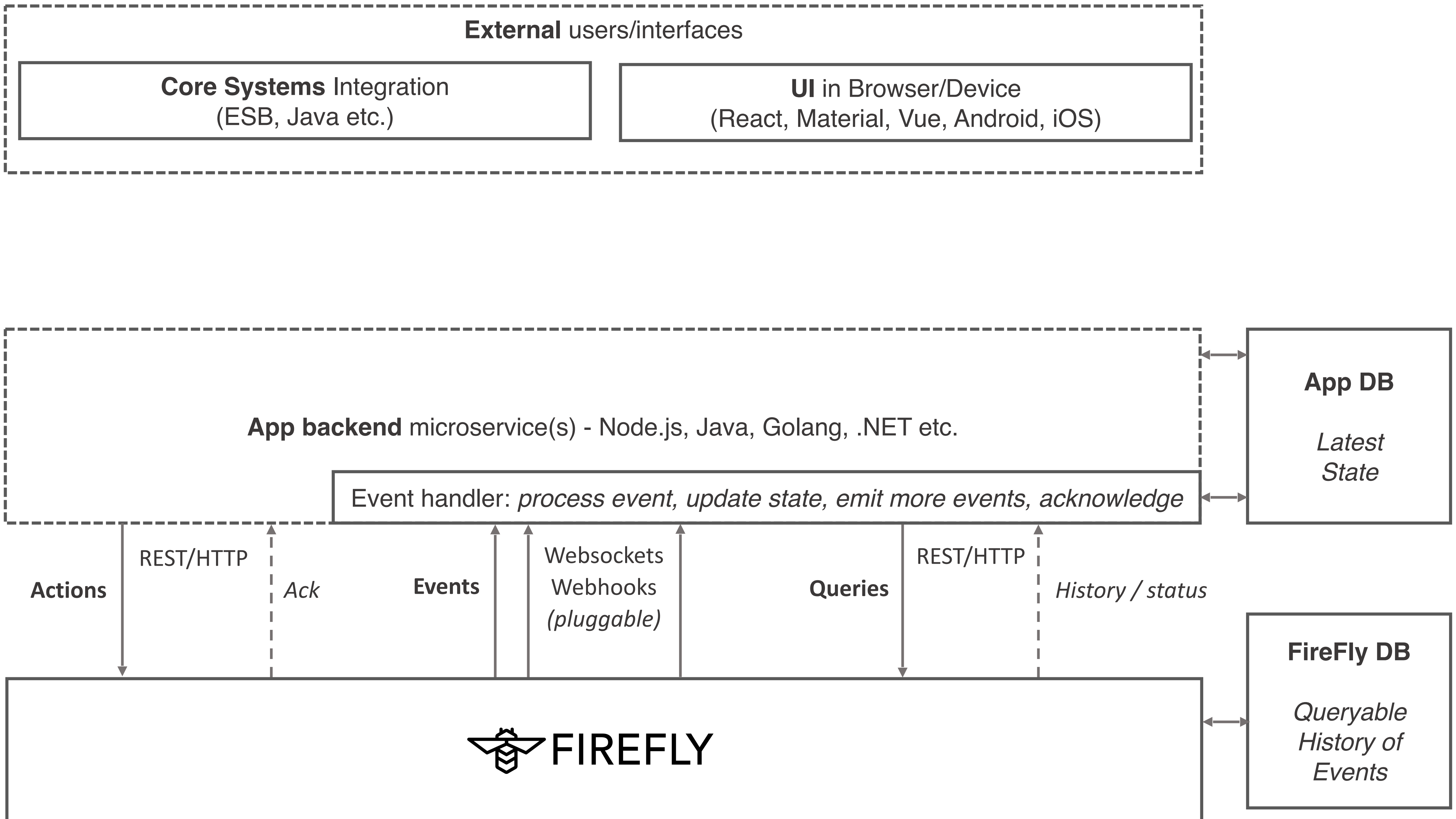
What an event-driven FireFly app looks like



External users/interfaces
Core Systems Integration (ESB, Java etc.)
UI in Browser/Device (React, Material, Vue, Android, iOS)
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
Event handler: *process event, update state, emit more events, acknowledge*
App DB
Latest State
Events
Websockets
Webhooks
(pluggable)
Queries
REST/HTTP
History / status
FireFly DB
Queryable History of Events
FIREFLY

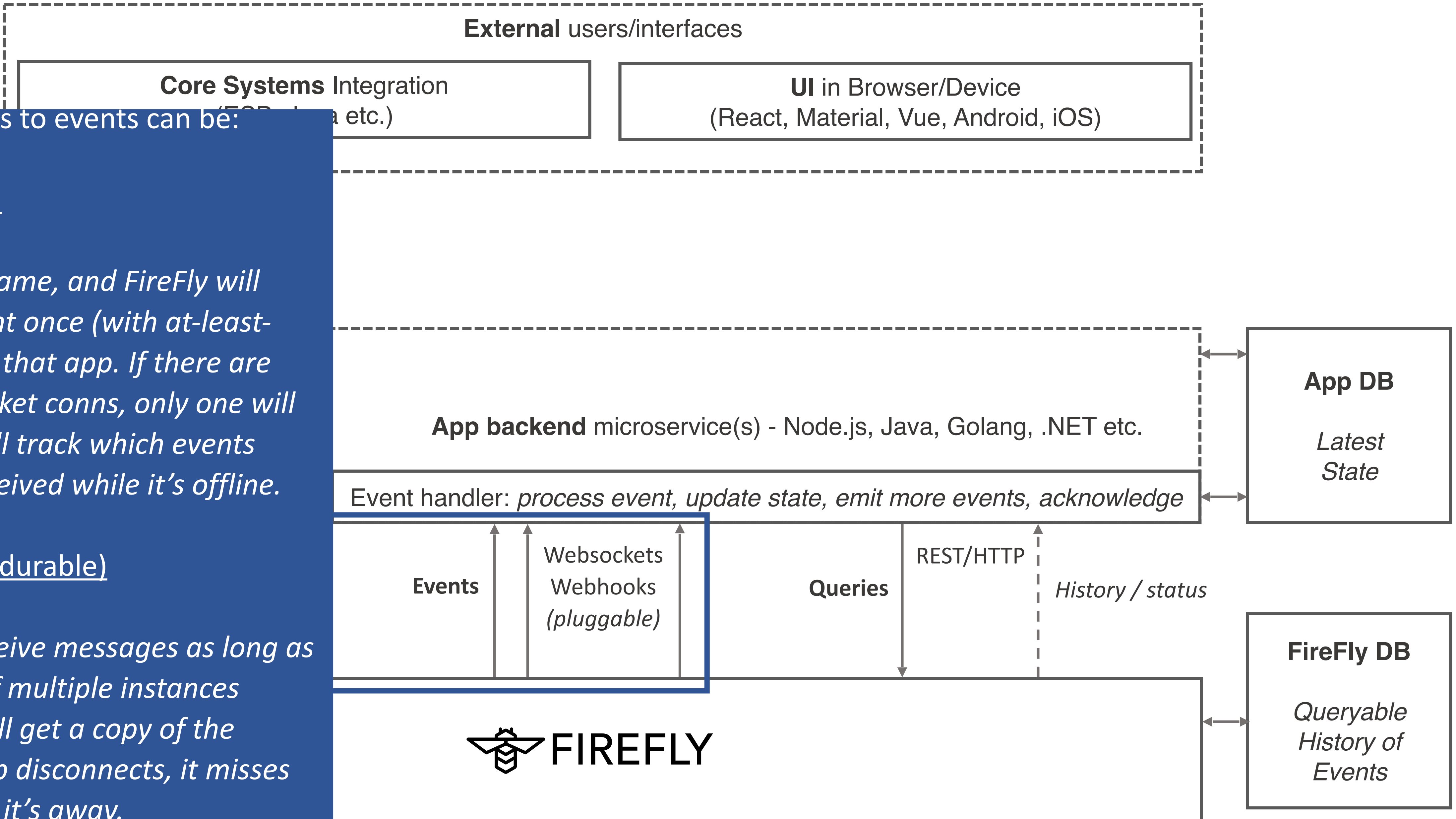
FIREFLY

What an event-driven FireFly app looks like



External users/interfaces
Core Systems Integration (ESB, Java etc.)
UI in Browser/Device (React, Material, Vue, Android, iOS)
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
Event handler: *process event, update state, emit more events, acknowledge*
App DB
Latest State
Actions
REST/HTTP
Ack
Events
Websockets
Webhooks
(pluggable)
Queries
REST/HTTP
History / status
FireFly DB
Queryable History of Events
FIREFLY

What an event-driven FireFly app looks like


External users/interfaces
Core Systems Integration
(ESB, Java etc.)
UI in Browser/Device
(React, Material, Vue, Android, iOS)
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
Event handler: *process event, update state, emit more events, acknowledge*
Events
Websockets
Webhooks
(pluggable)
Queries
REST/HTTP
History / status
App DB
Latest State
FireFly DB
Queryable History of Events
FIREFLY

App subscriptions to events can be:

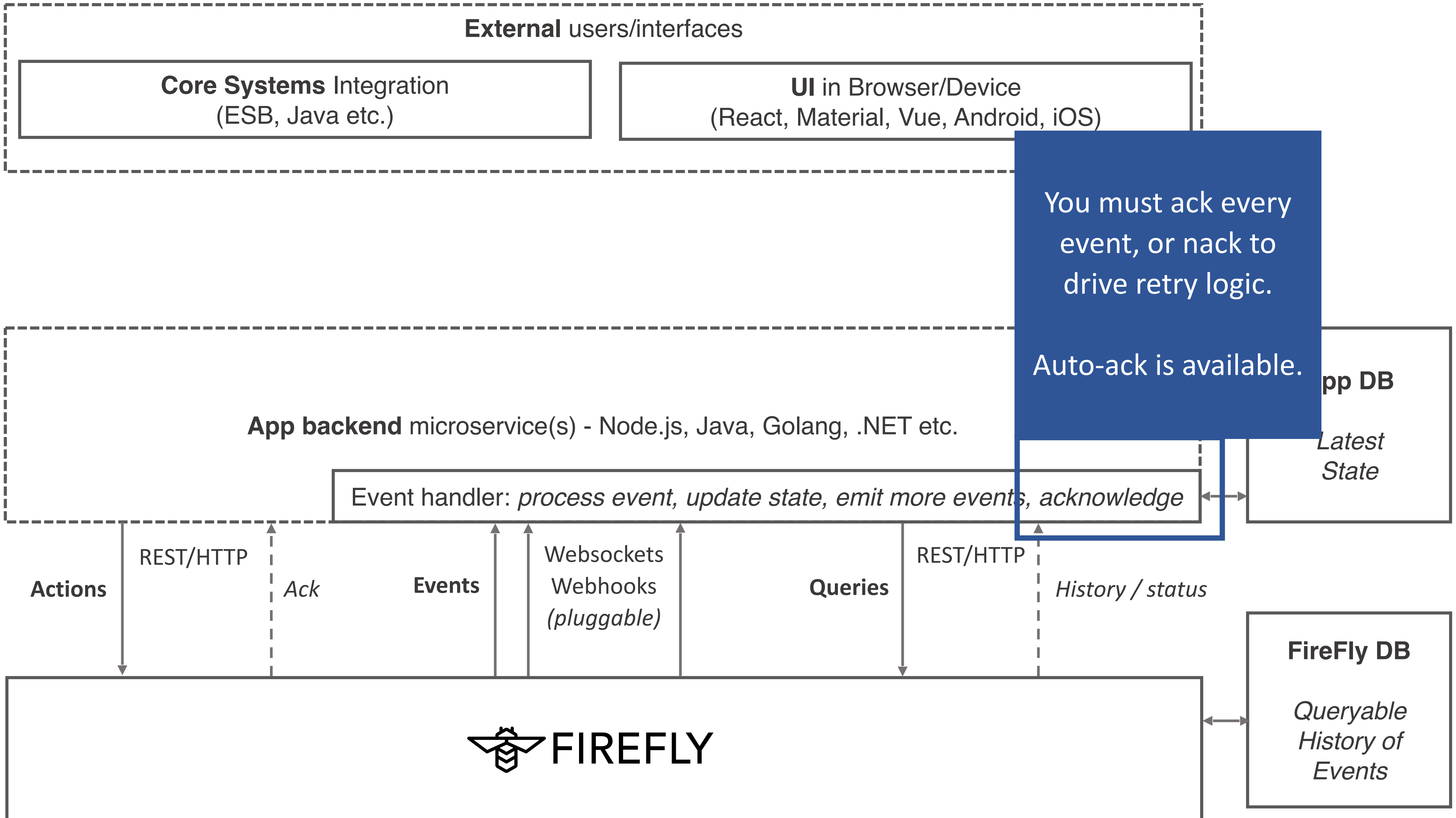
<u>Named (durable)</u>

Your app has a name, and FireFly will deliver each event once (with at-least-once delivery) to that app. If there are multiple WebSocket conns, only one will get it. FireFlly will track which events your app has received while it's offline.

<u>Ephemeral (non-durable)</u>

Your app will receive messages as long as it is connected. If multiple instances connect, each will get a copy of the event. If your app disconnects, it misses any events while it's away.

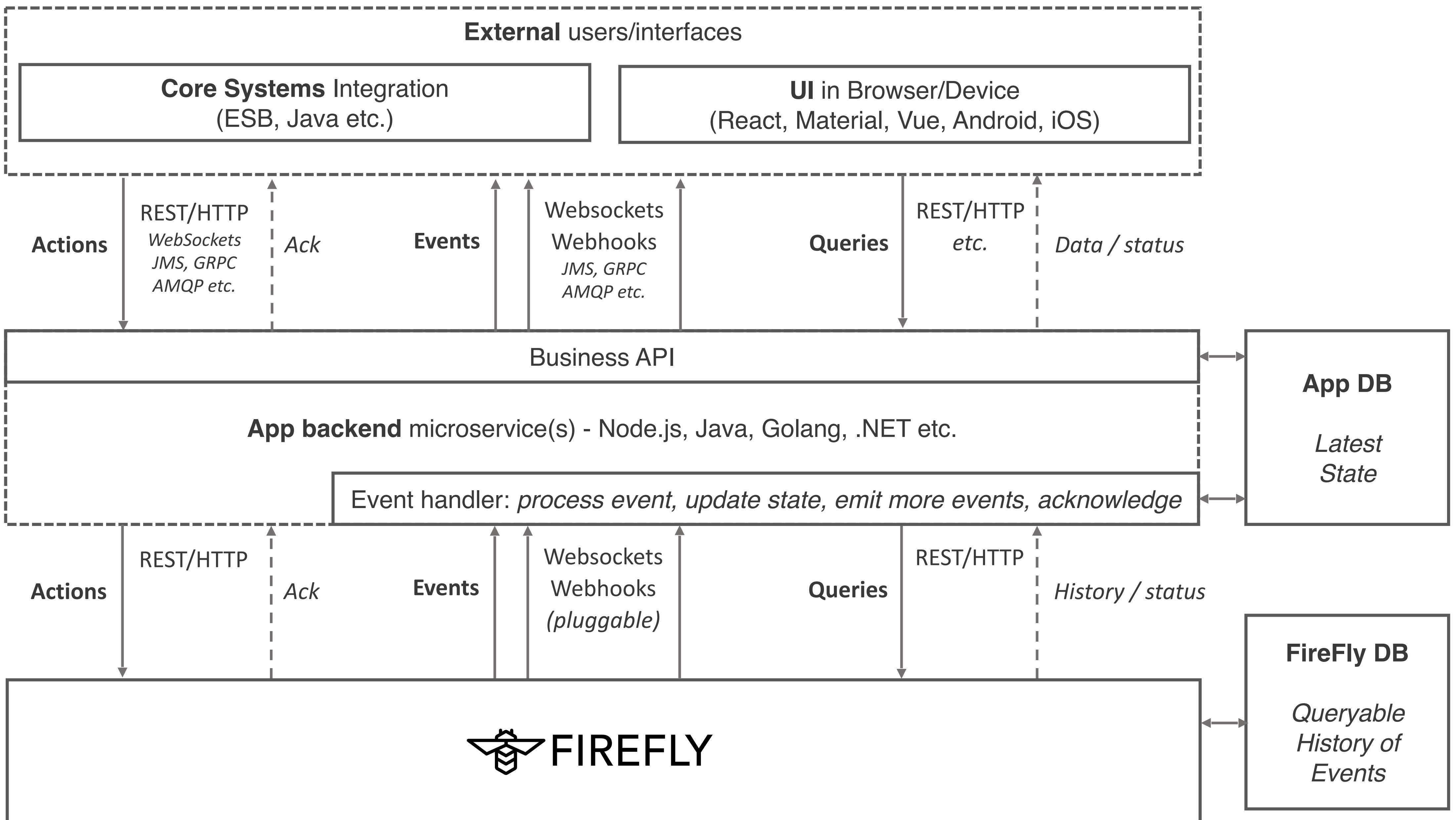
What an event-driven FireFly app looks like


External users/interfaces
Core Systems Integration (ESB, Java etc.)
UI in Browser/Device (React, Material, Vue, Android, iOS)
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
Event handler: process event, update state, emit more events, acknowledge
pp DB
Latest State
Actions
REST/HTTP
Ack
Events
Websockets
Webhooks
(pluggable)
Queries
REST/HTTP
History / status
FIREFLY
FireFly DB
Queryable History of Events

You must ack every event, or nack to drive retry logic.

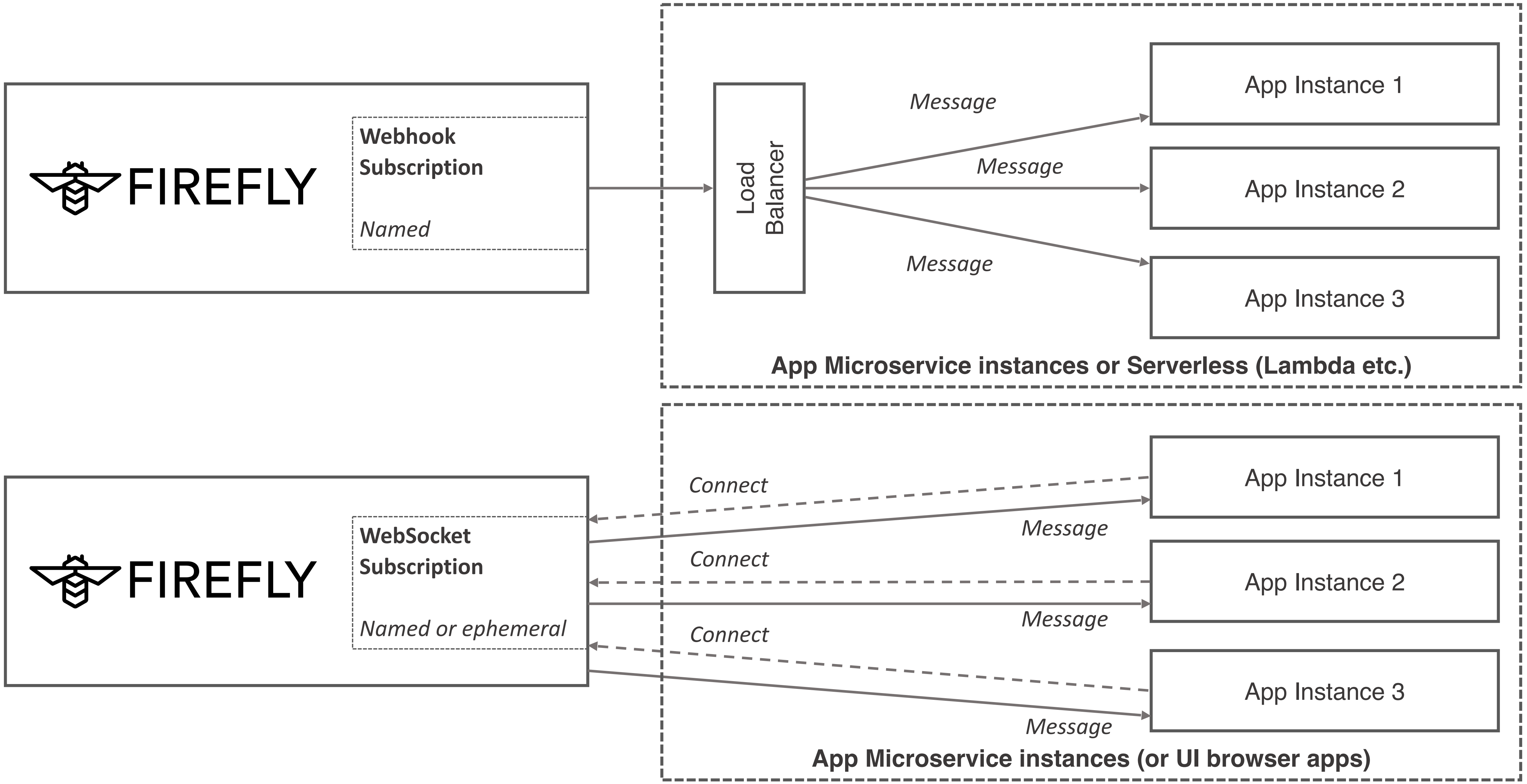
Auto-ack is available.

What an event-driven FireFly app looks like



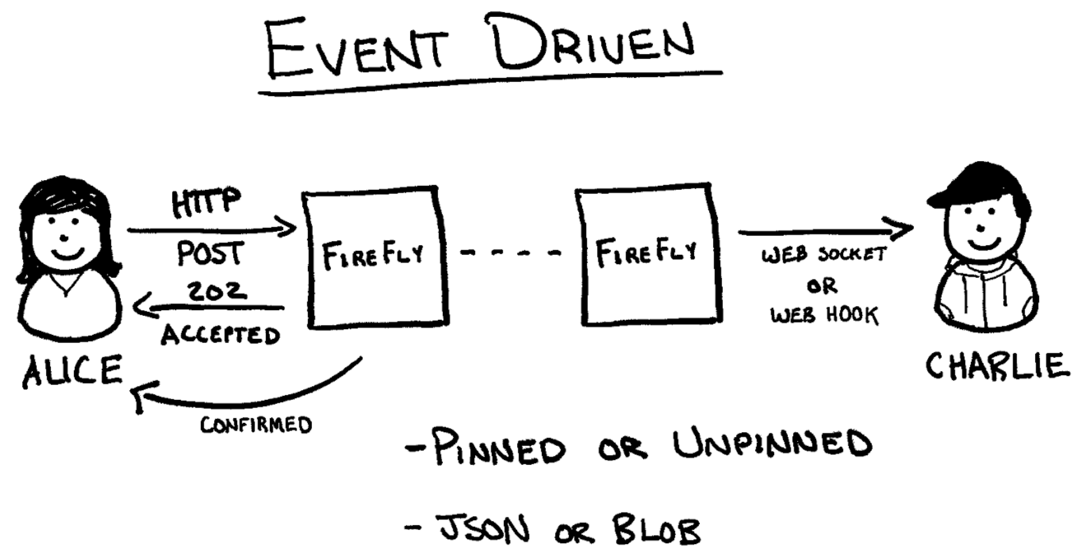
External users/interfaces
Core Systems Integration (ESB, Java etc.)
UI in Browser/Device (React, Material, Vue, Android, iOS)
Actions
REST/HTTP *WebSockets JMS, GRPC AMQP etc.*
Ack
Events
Websockets Webhooks *JMS, GRPC AMQP etc.*
Queries
REST/HTTP *etc.*
Data / status
Business API
App backend microservice(s) - Node.js, Java, Golang, .NET etc.
Event handler: *process event, update state, emit more events, acknowledge*
App DB *Latest State*
Actions
REST/HTTP
Ack
Events
Websockets Webhooks *(pluggable)*
Queries
REST/HTTP
History / status
FIREFLY
FireFly DB *Queryable History of Events*

Webhooks vs. WebSockets



Open Discussion

Community Call 4th August 2021


EVENT DRIVEN
HTTP
POST
202
ACCEPTED
FIREFLY
FIREFLY
WEB SOCKET
OR
WEB HOOK
ALICE
CHARLIE
CONFIRMED
- PINNED OR UNPINNED
- JSON OR BLOB