

# AGENDA

- The Golden Rule
- Hexagonal Architecture
- Request/Response vs. Fire and Forget
- Queues/Load Balancing vs. Publish-Subscribe
- Introducing Persistence
- Event Sourcing / CQRS
- From Modules to Microservices

# THE GOLDEN RULE

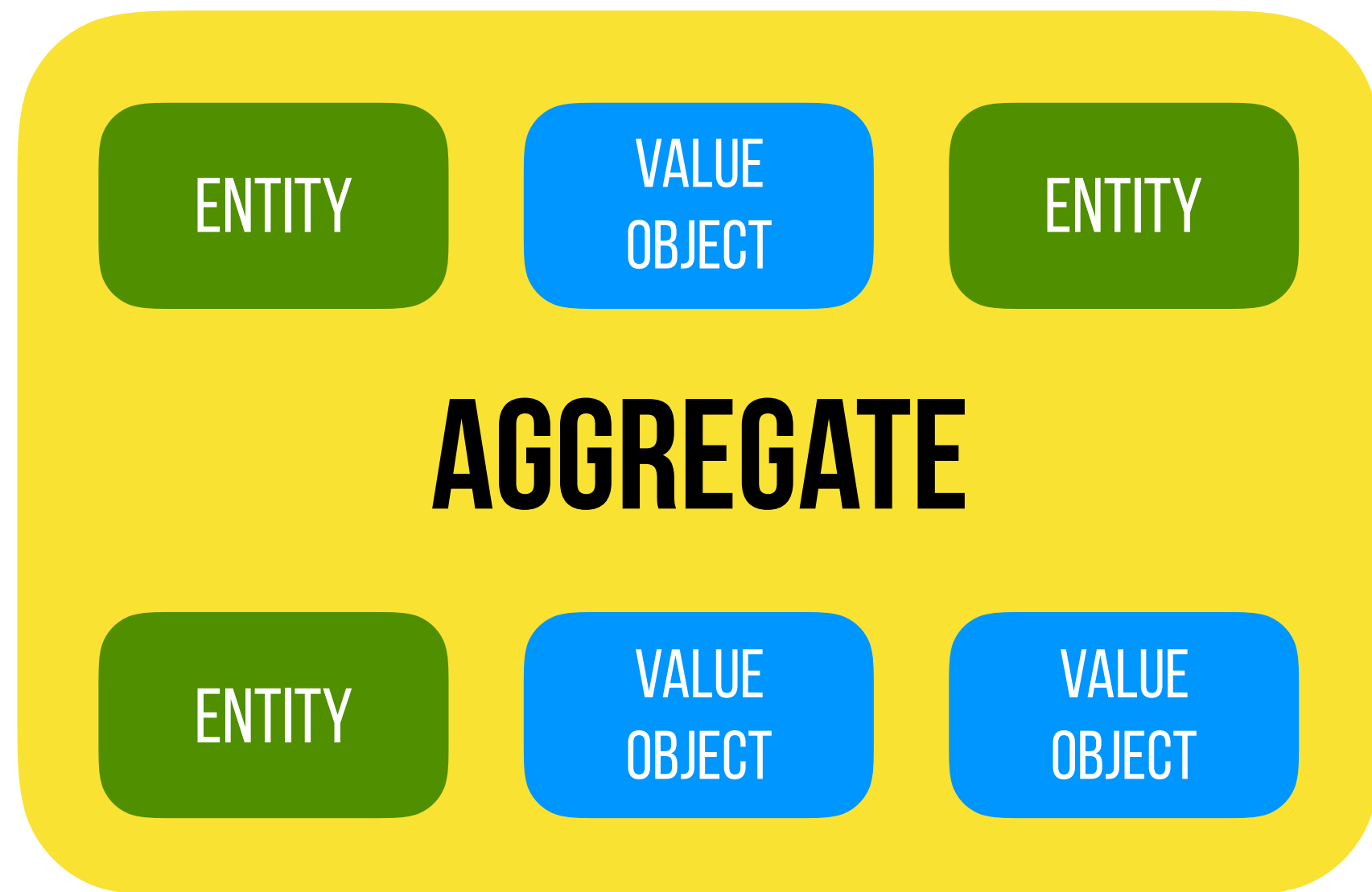
**$\text{Aggregate} \leq \text{SizeOf}(\text{Module}) \leq \text{Bounded Context}$**

# THE GOLDEN RULE



**$\text{Aggregate} \leq \text{SizeOf}(\text{Module}) \leq \text{Bounded Context}$**

# THE GOLDEN RULE

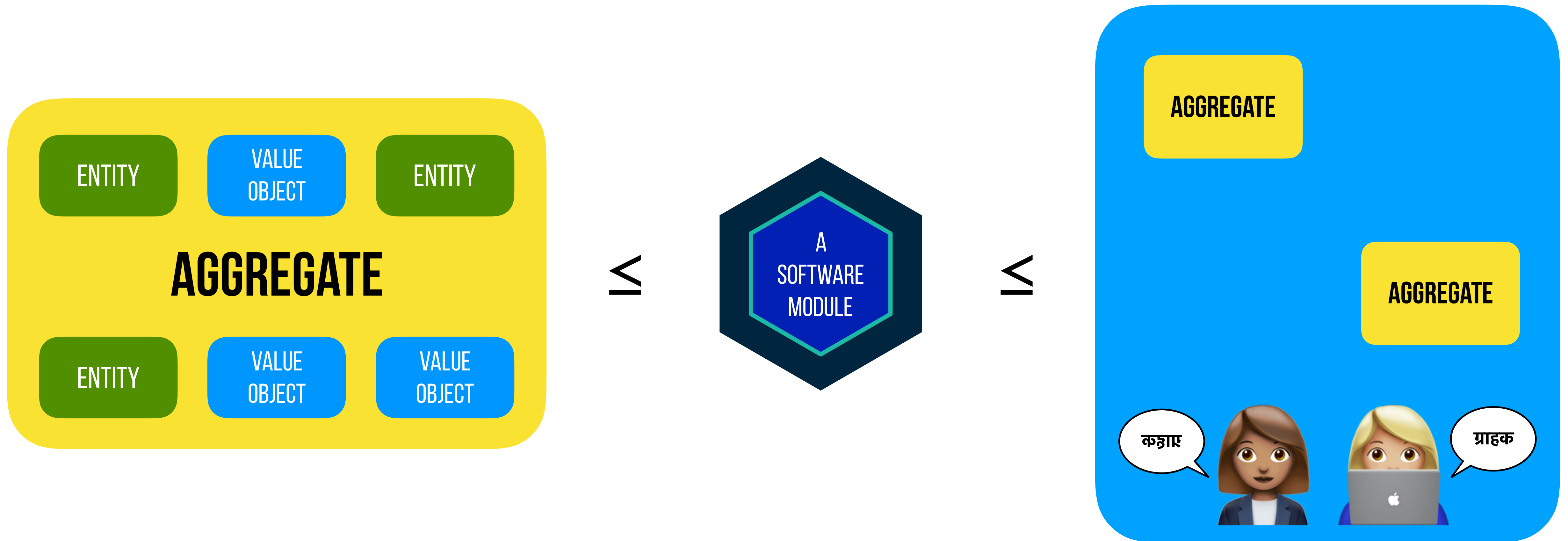


$\leq$



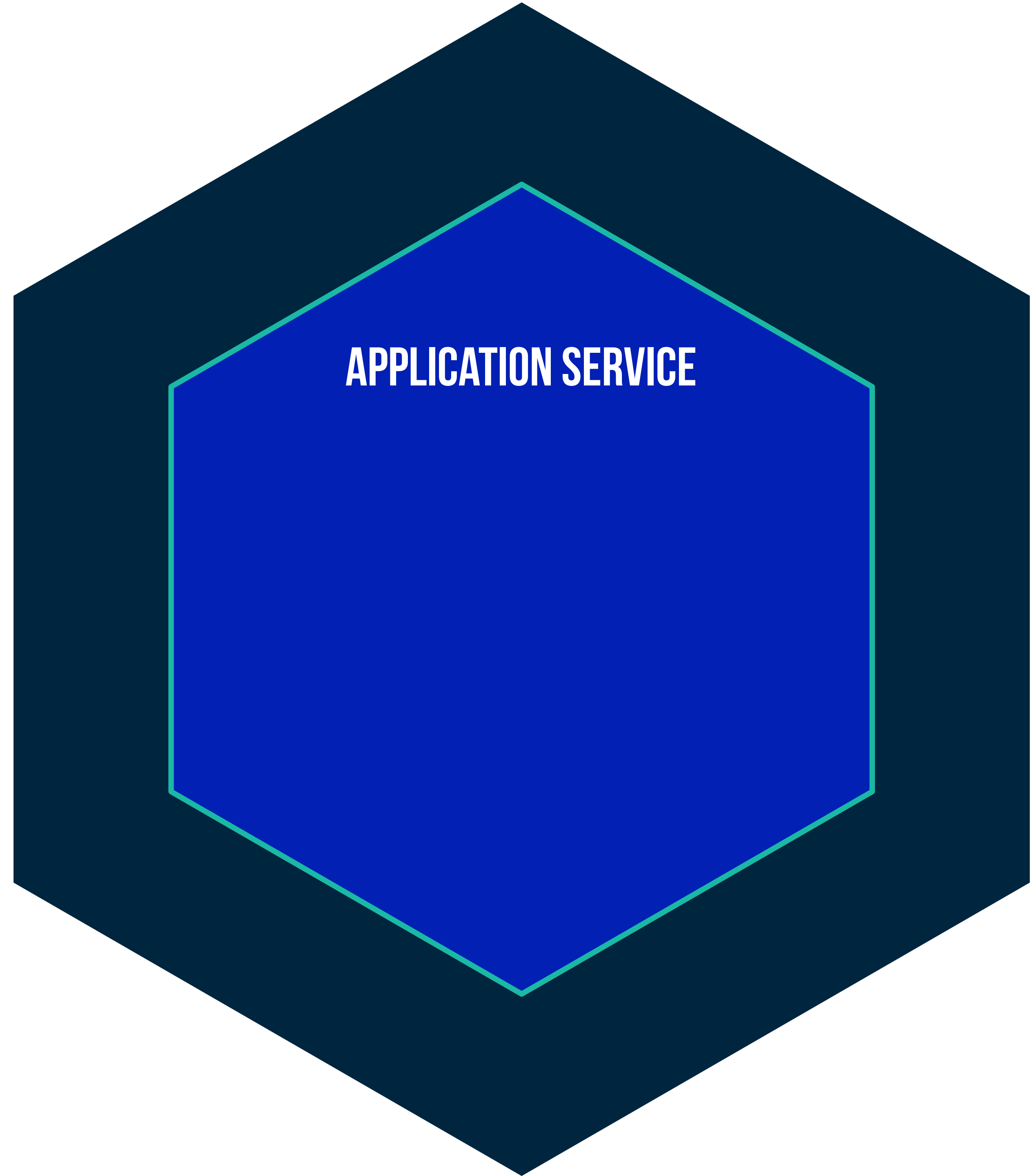
**Aggregate  $\leq$  SizeOf(Module)  $\leq$  Bounded Context**

# THE GOLDEN RULE



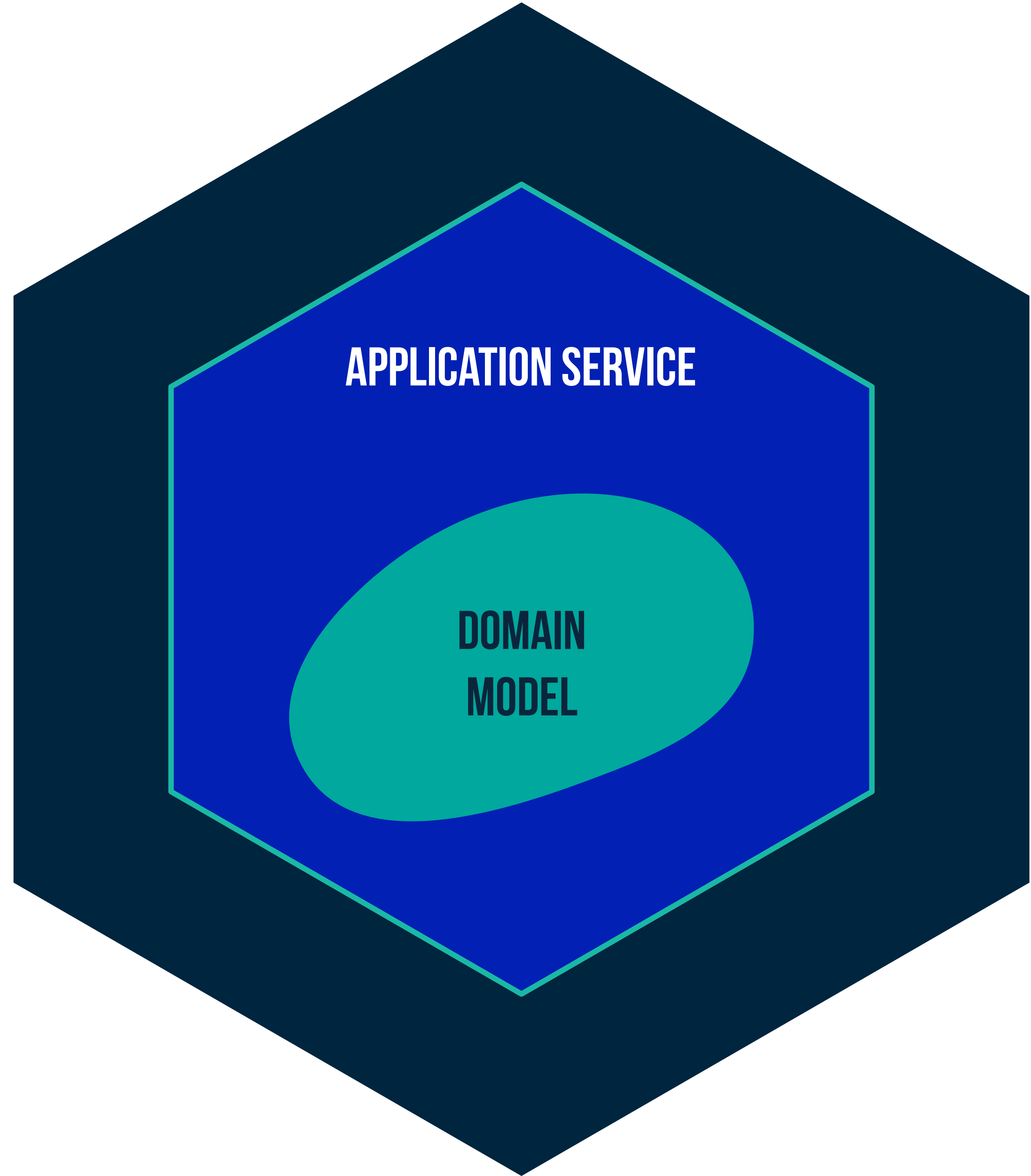
**Aggregate  $\leq$  SizeOf(Module)  $\leq$  Bounded Context**

# HEXAGONAL ARCHITECTURE



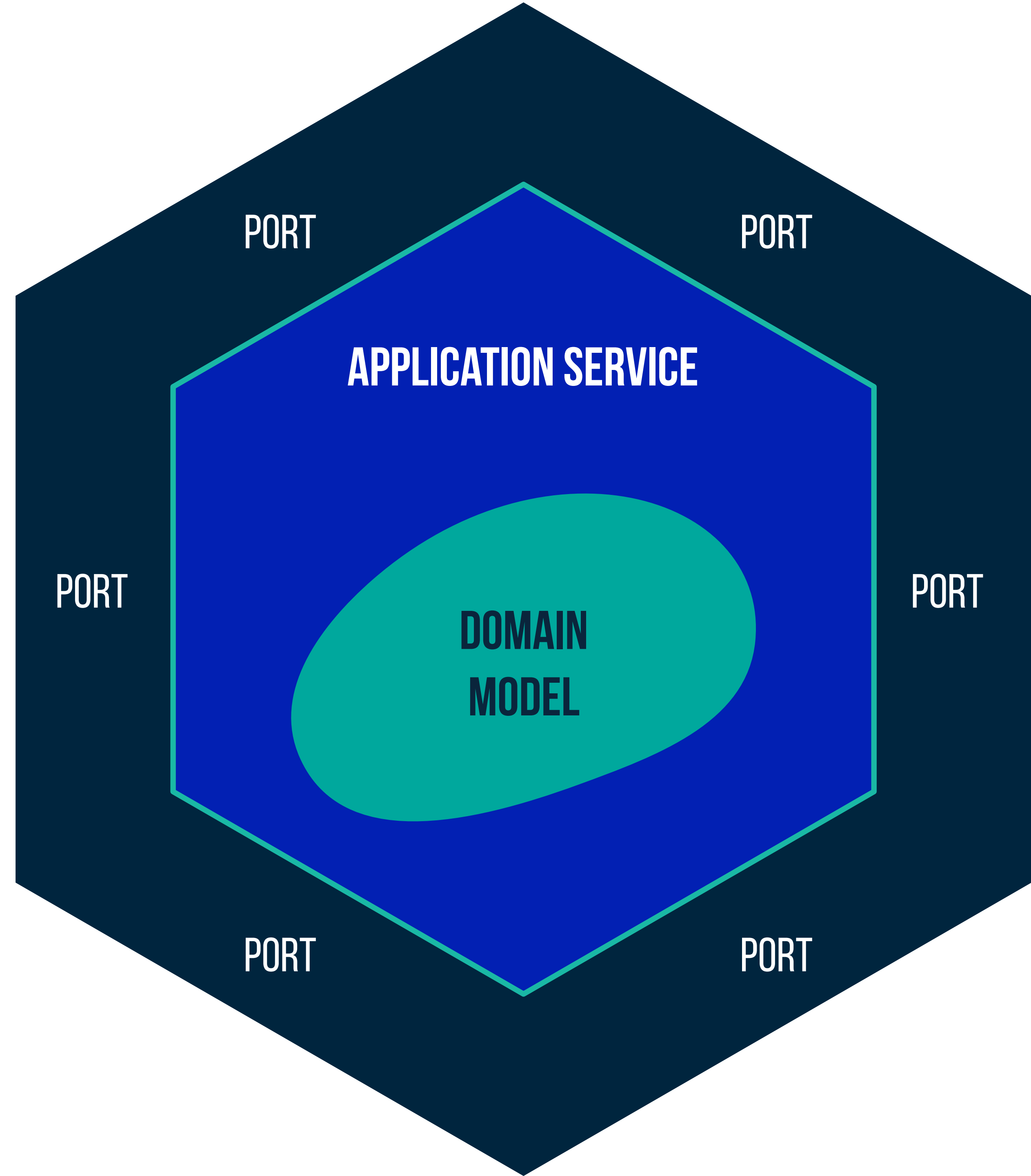
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.)  
wrapped by an Application Service



# HEXAGONAL ARCHITECTURE

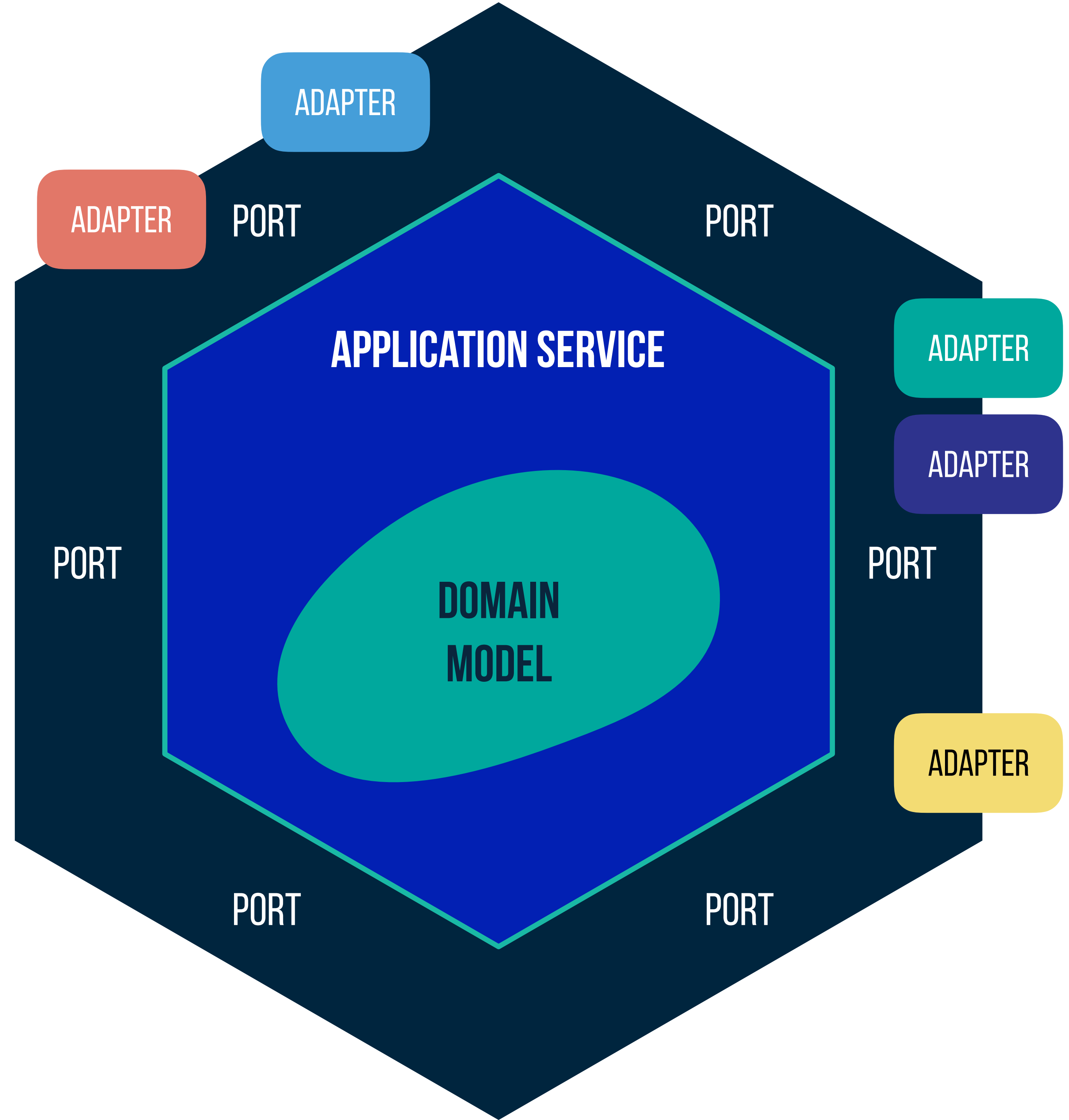
- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points ([interface](#))





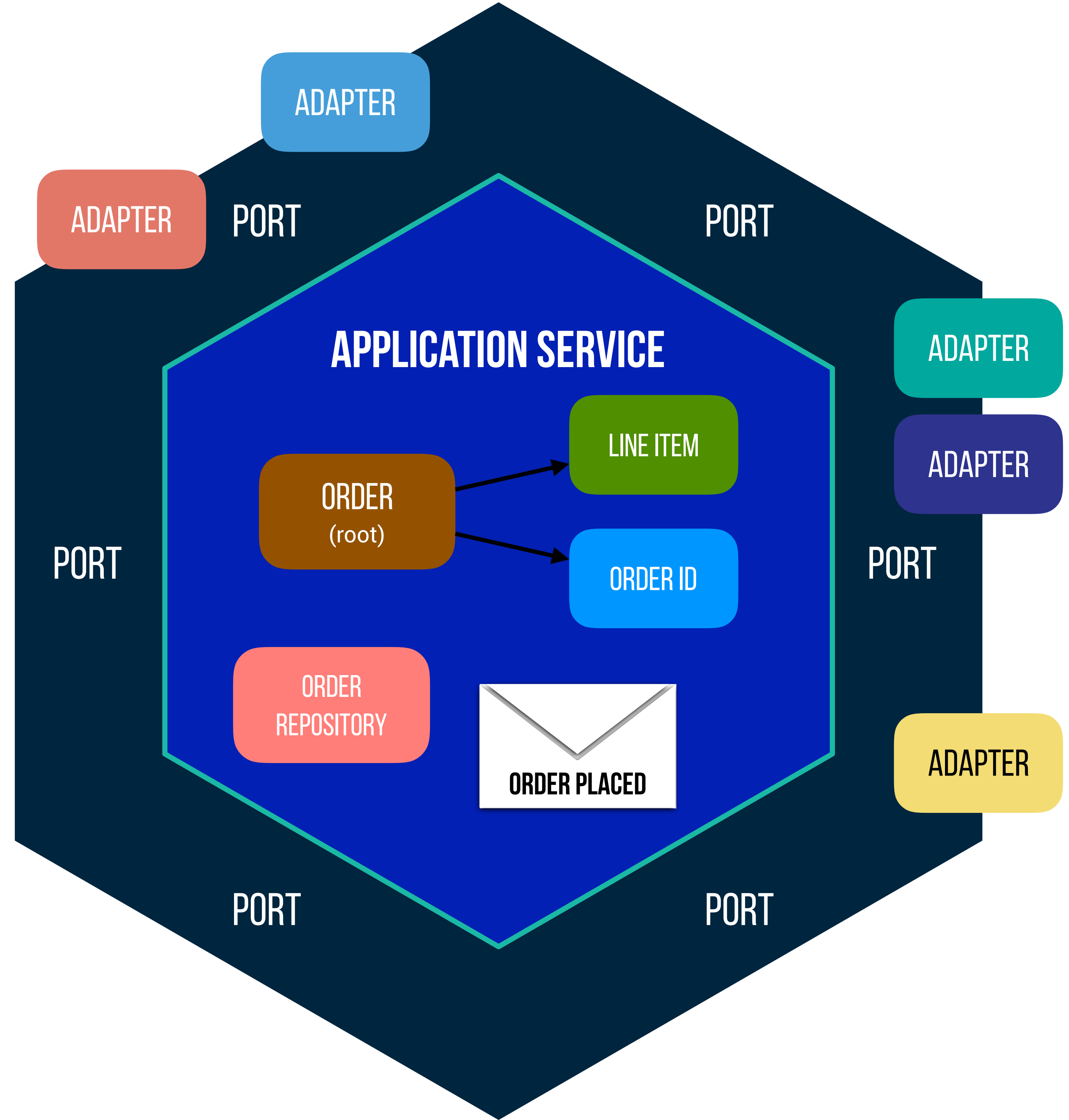
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points (*interface*)
- Adapters represent specific entry/exit implementations (*class*)



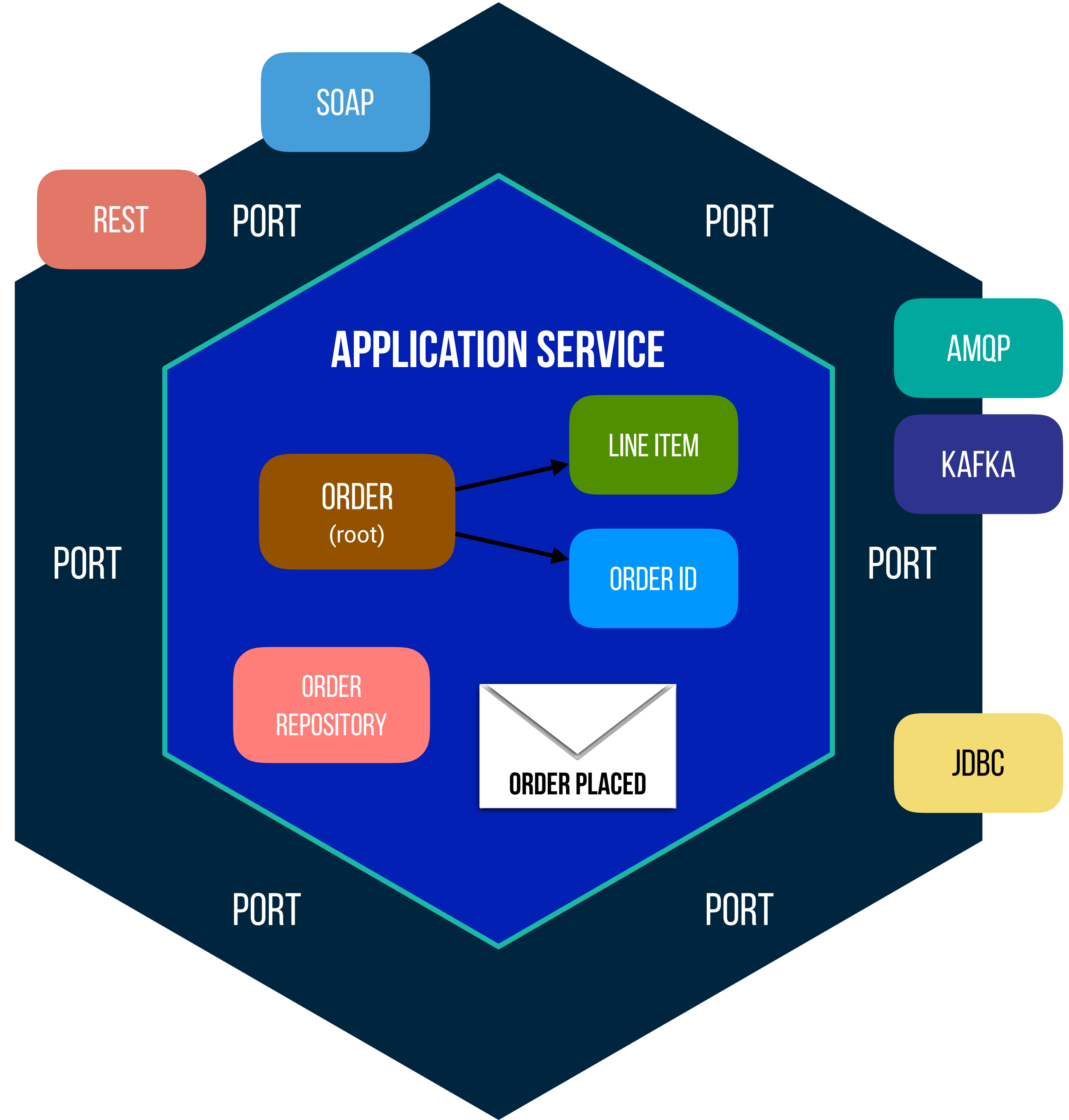
# HEXAGONAL ARCHITECTURE

- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points (*interface*)
- Adapters represent specific entry/exit implementations (*class*)

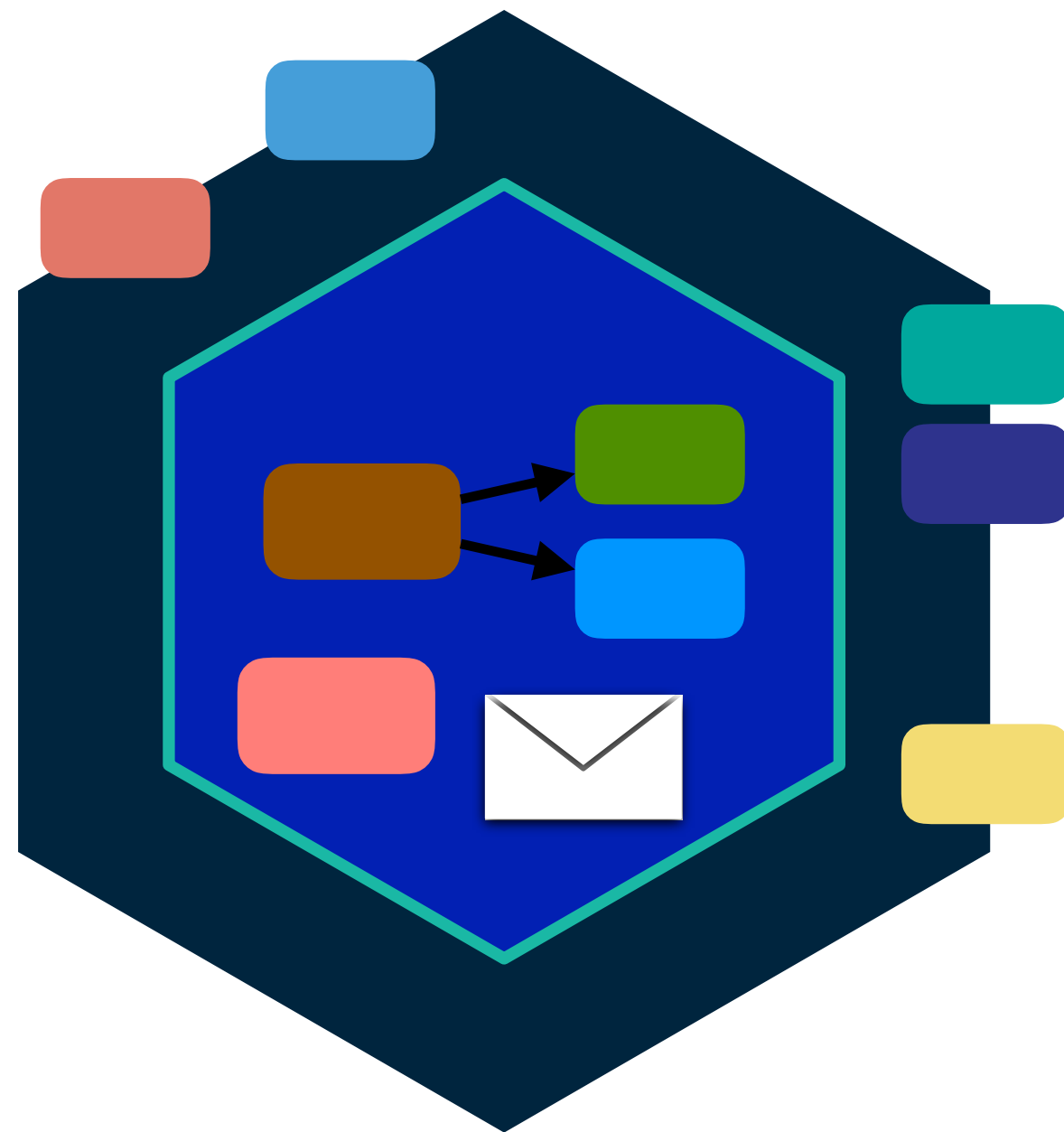


# HEXAGONAL ARCHITECTURE

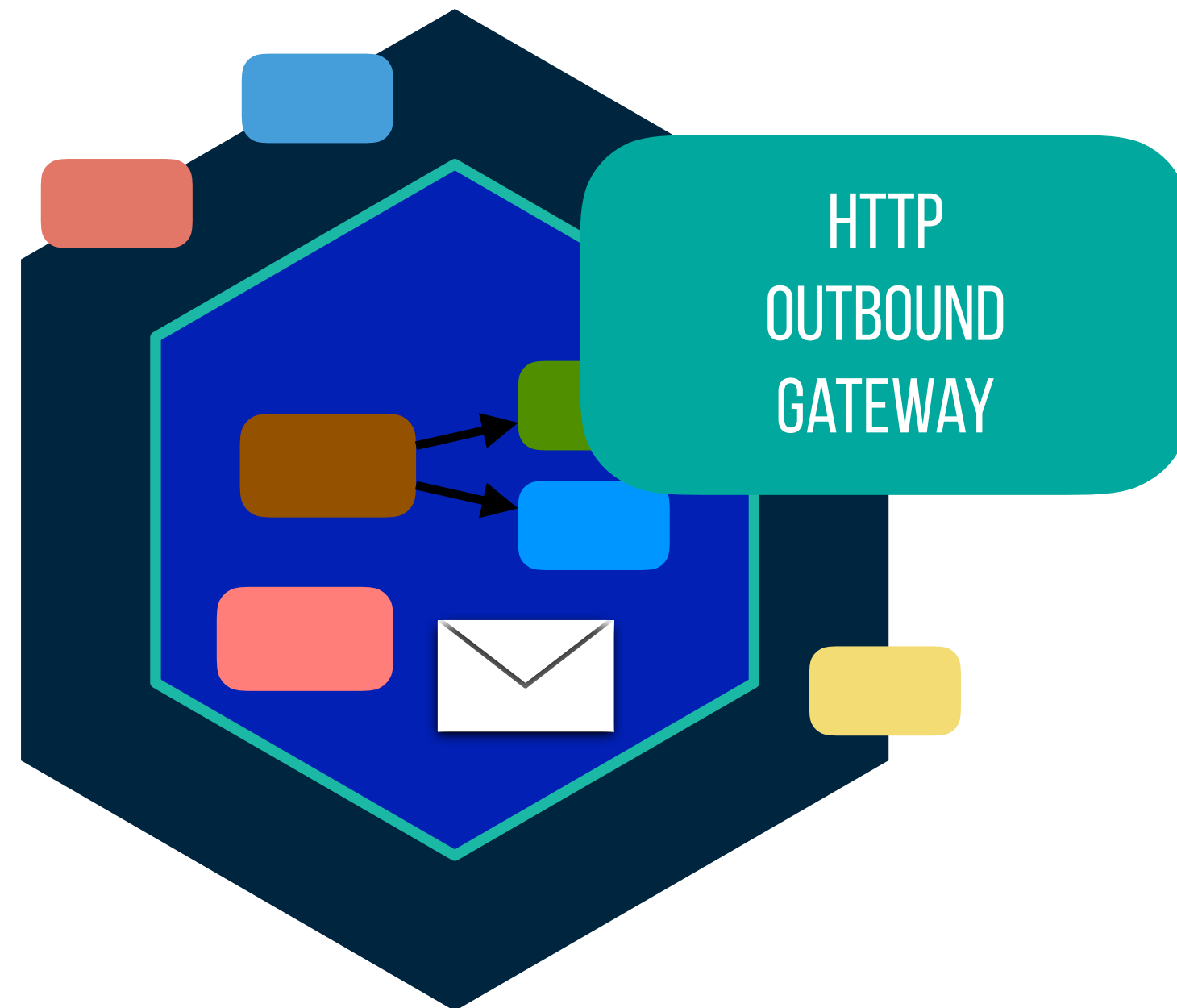
- Domain Model (Aggregates, etc.) wrapped by an Application Service
- Ports represent generic entry/exit points (*interface*)
- Adapters represent specific entry/exit implementations (*class*)



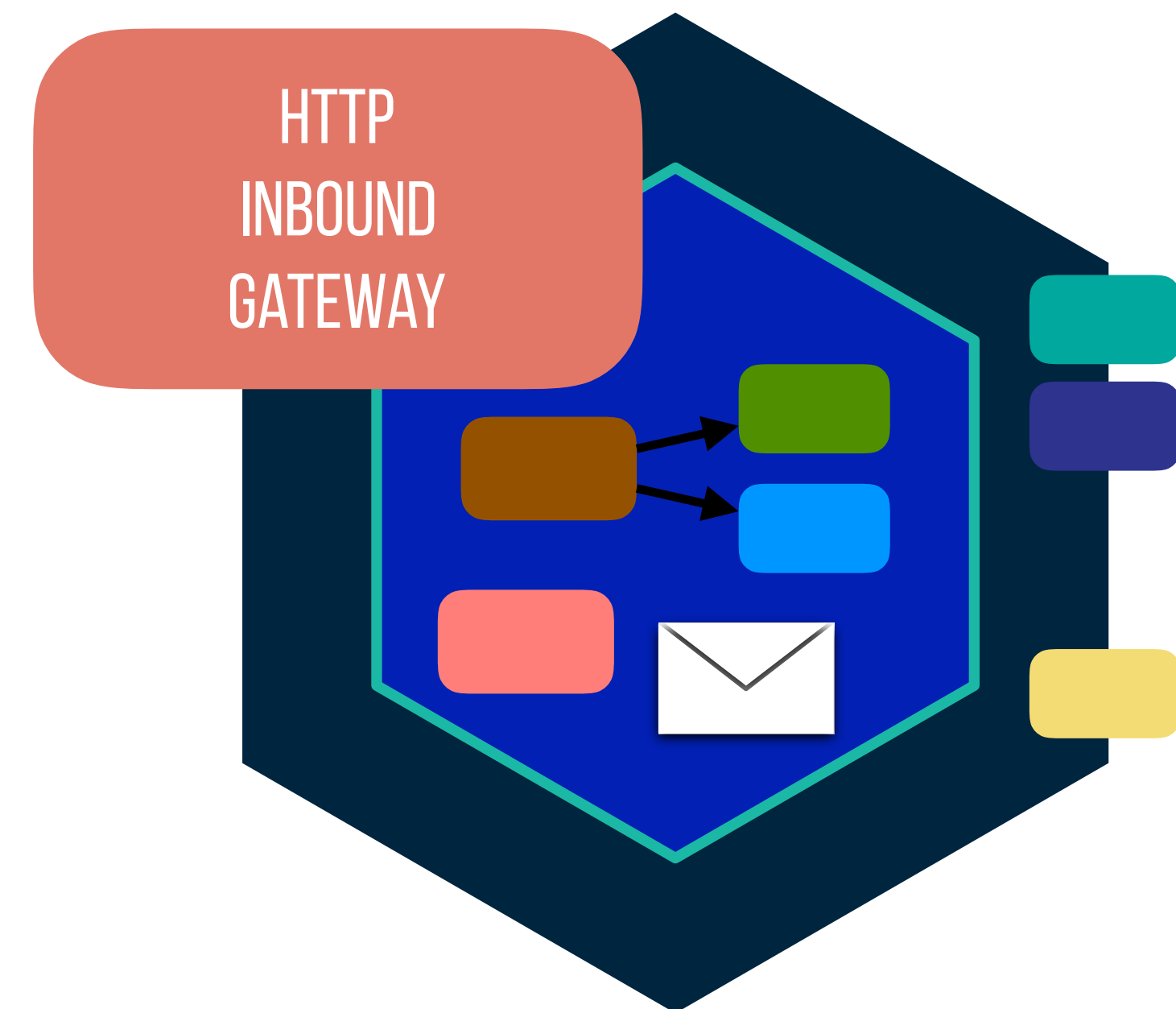
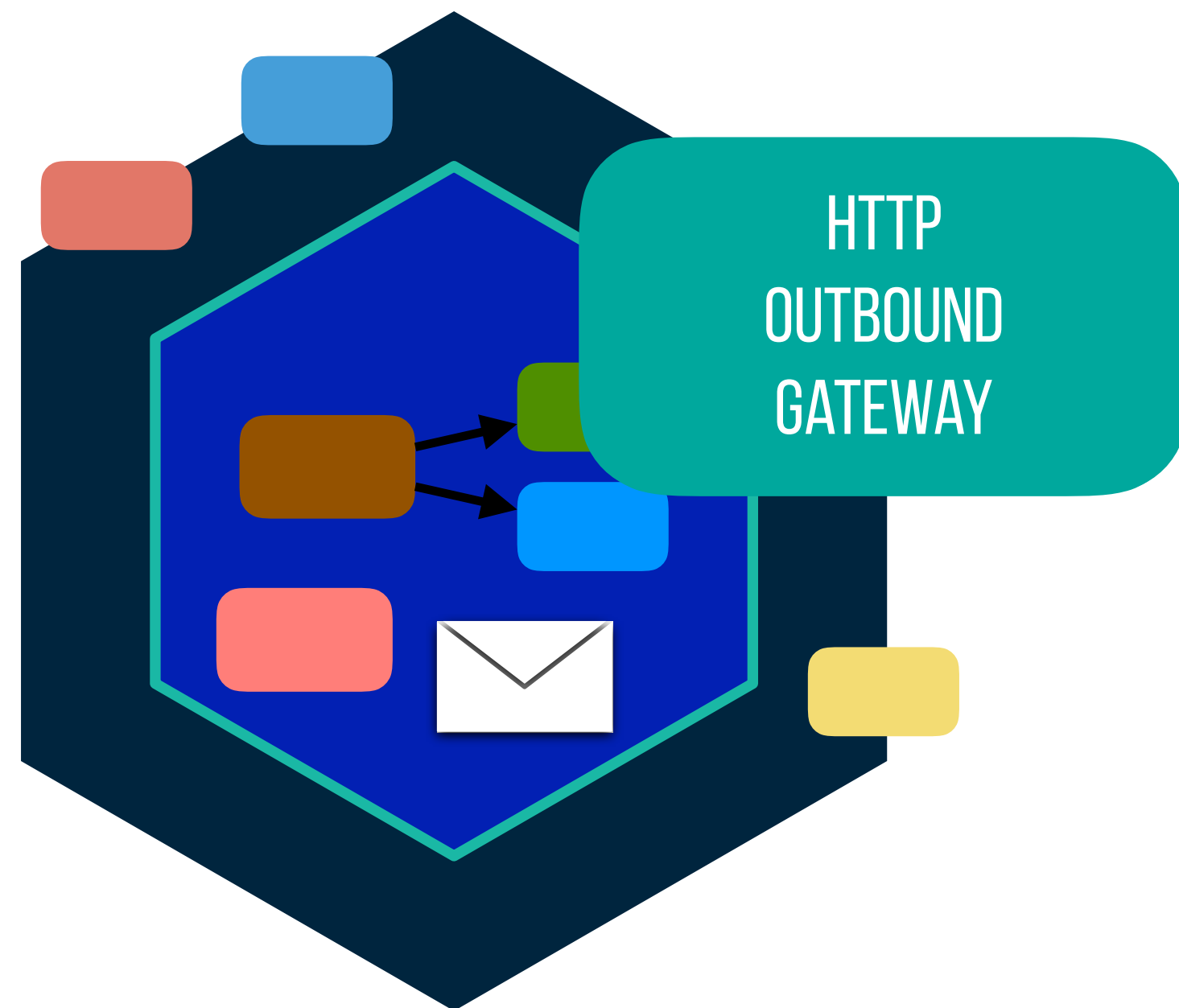
# REQUEST / RESPONSE



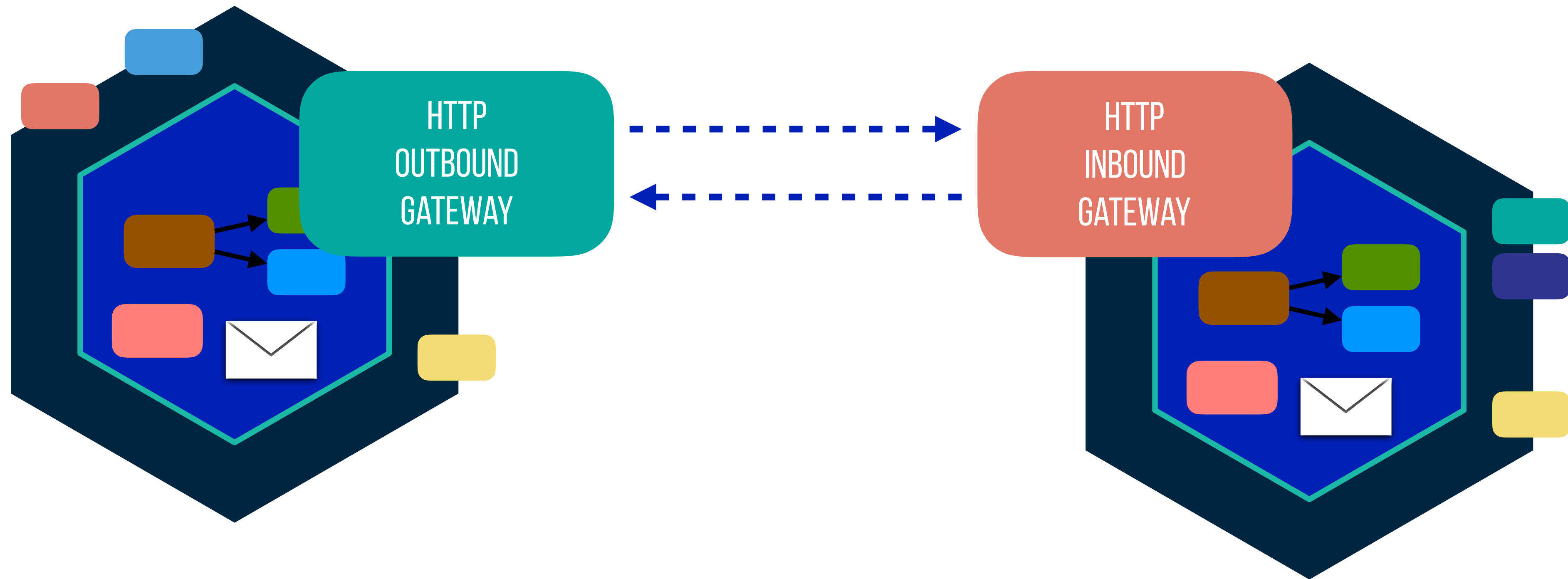
# REQUEST / RESPONSE



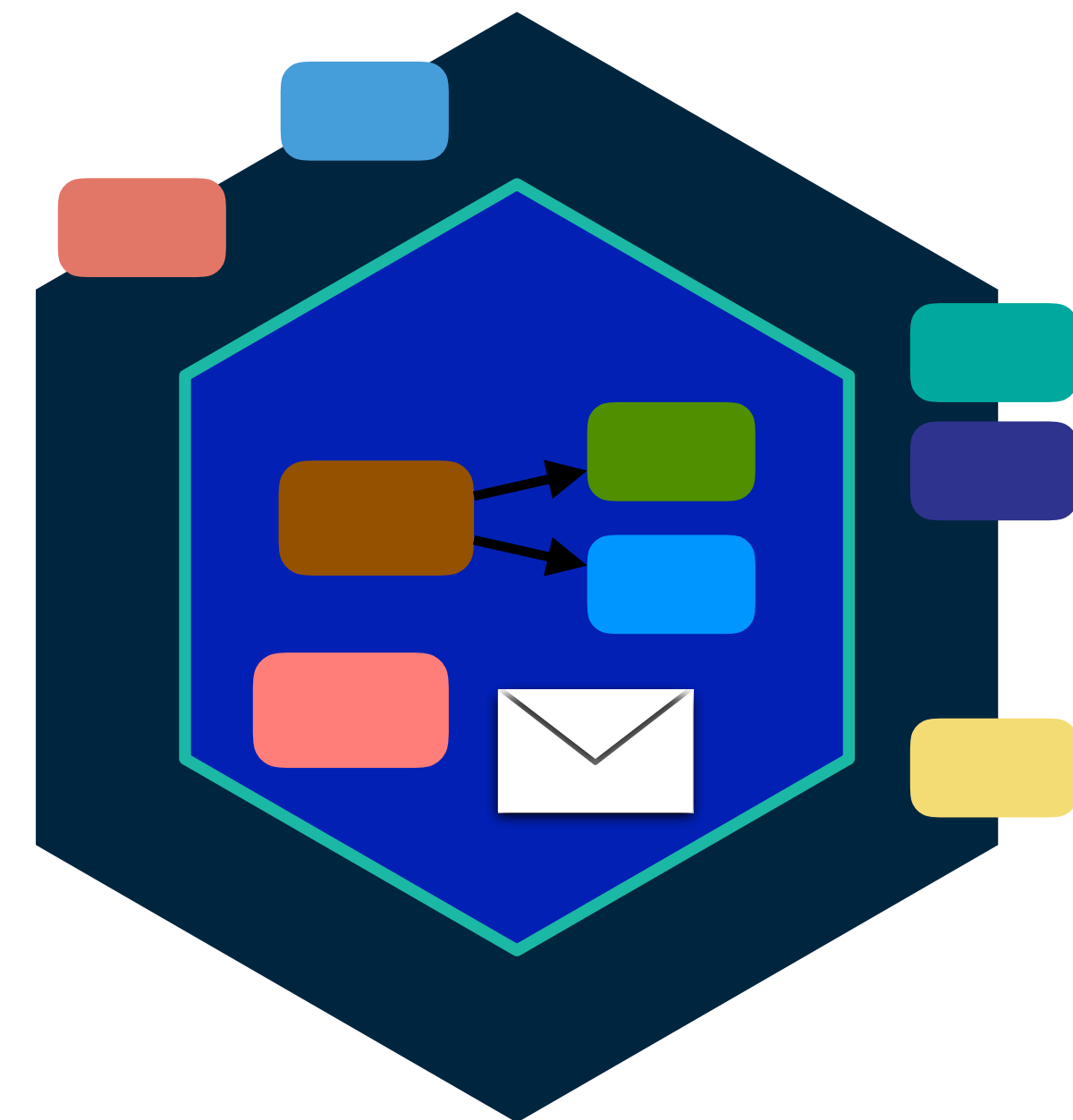
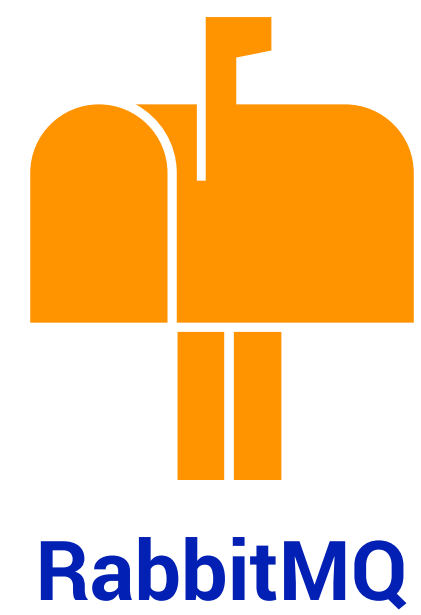
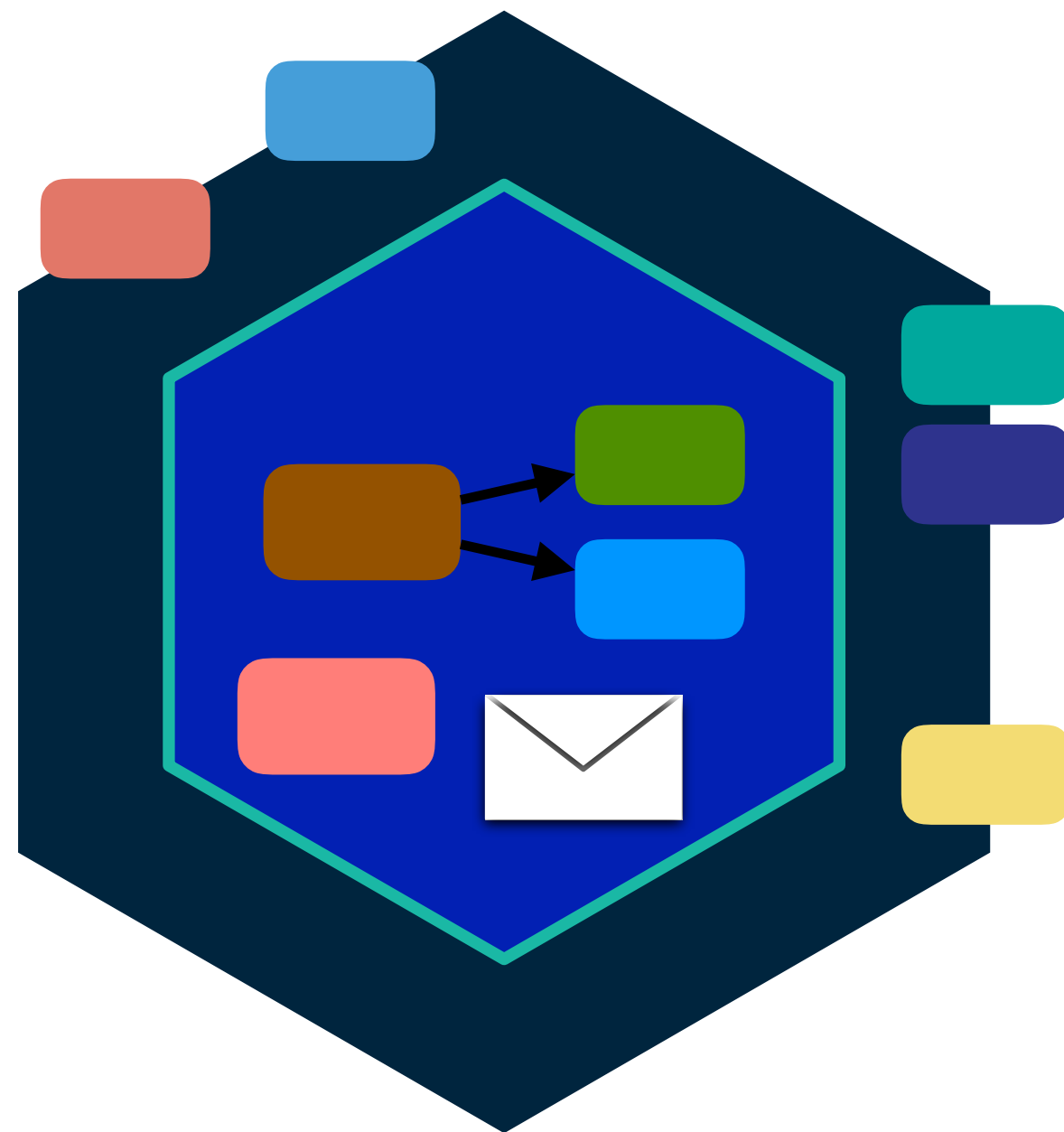
# REQUEST / RESPONSE



# REQUEST / RESPONSE

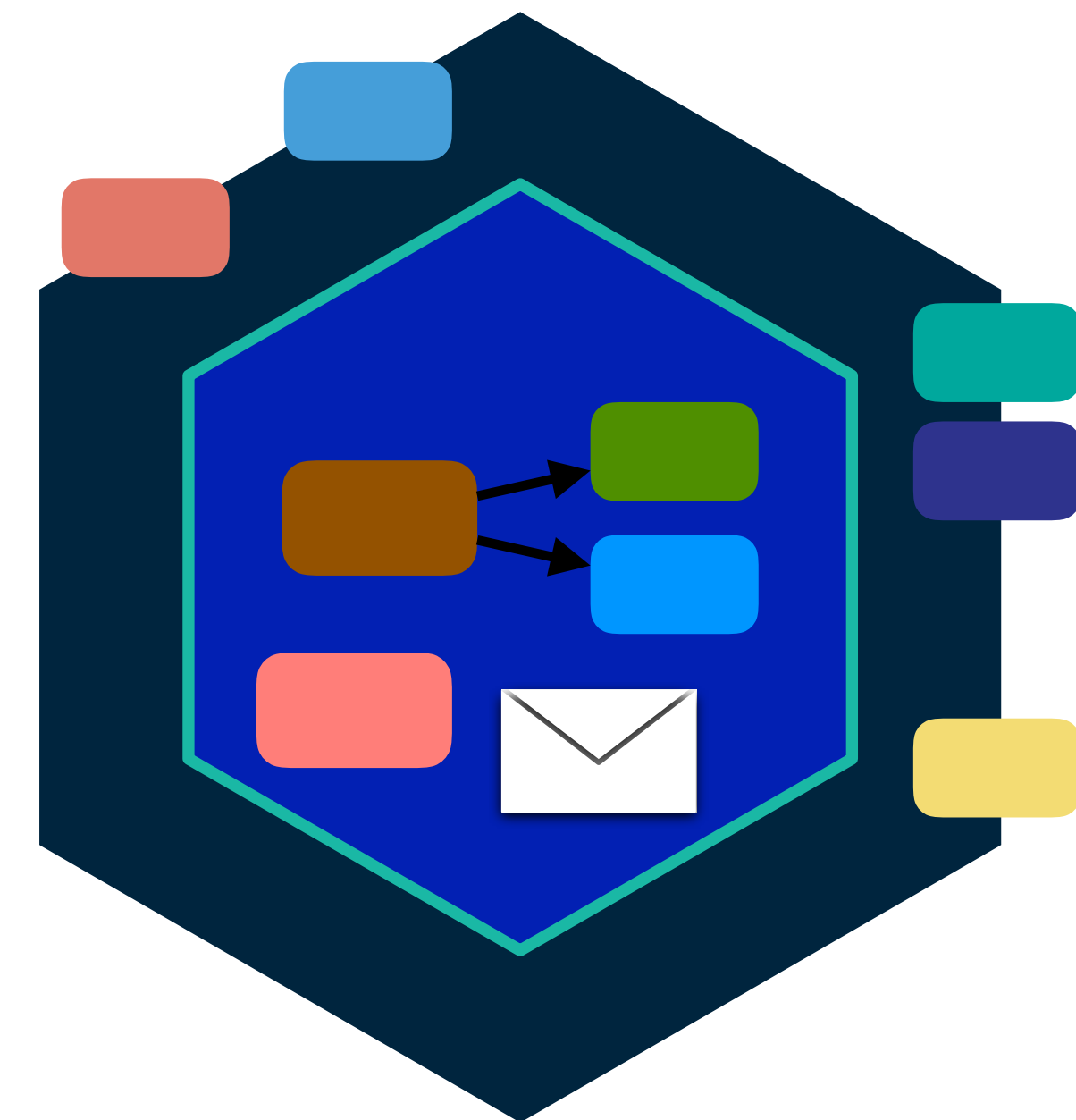
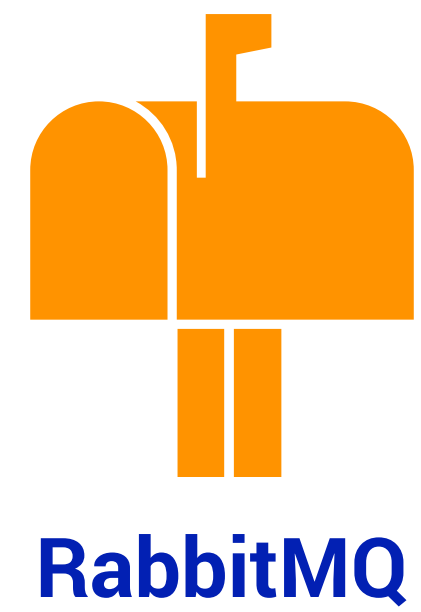
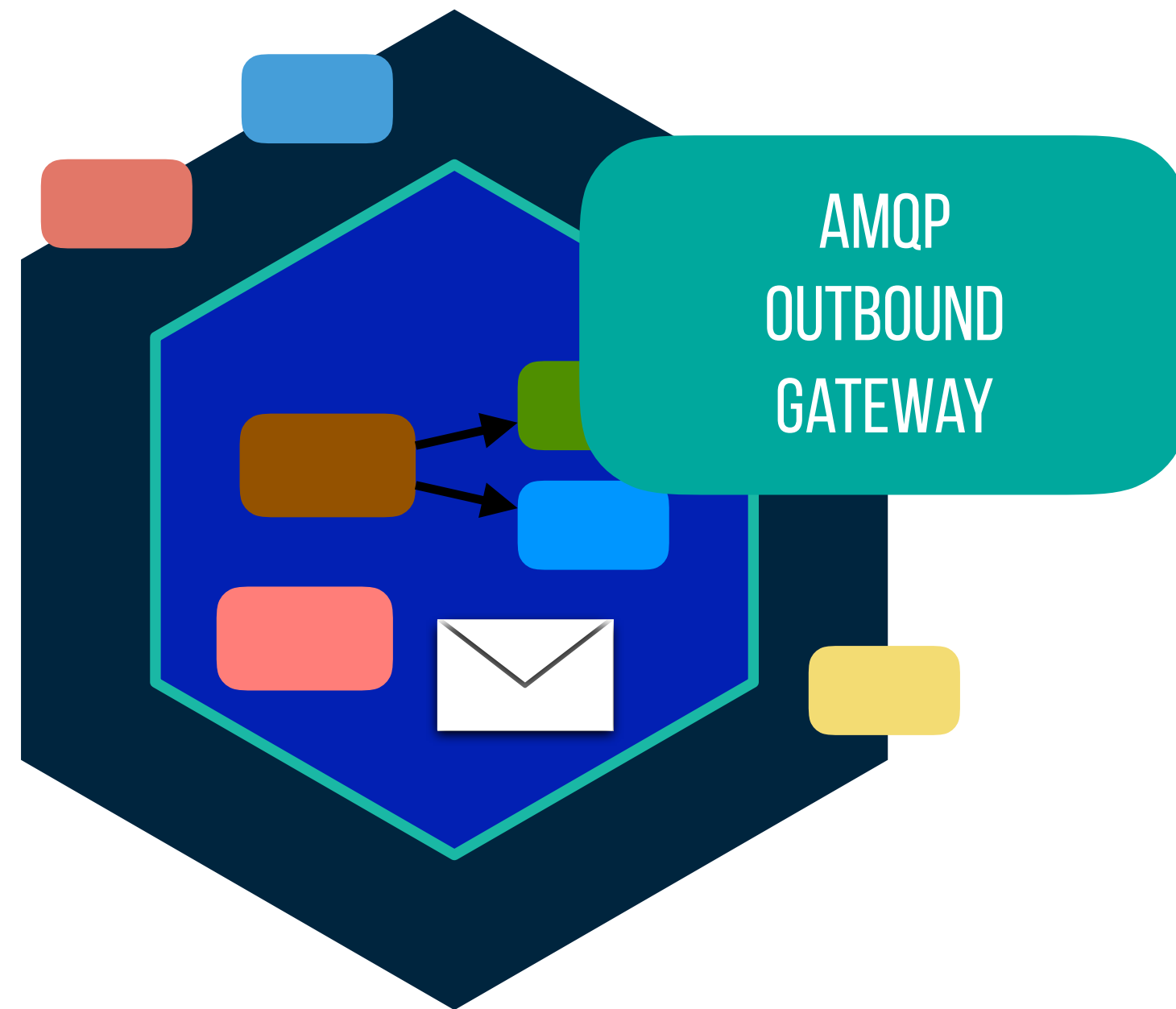


# FIRE AND FORGET

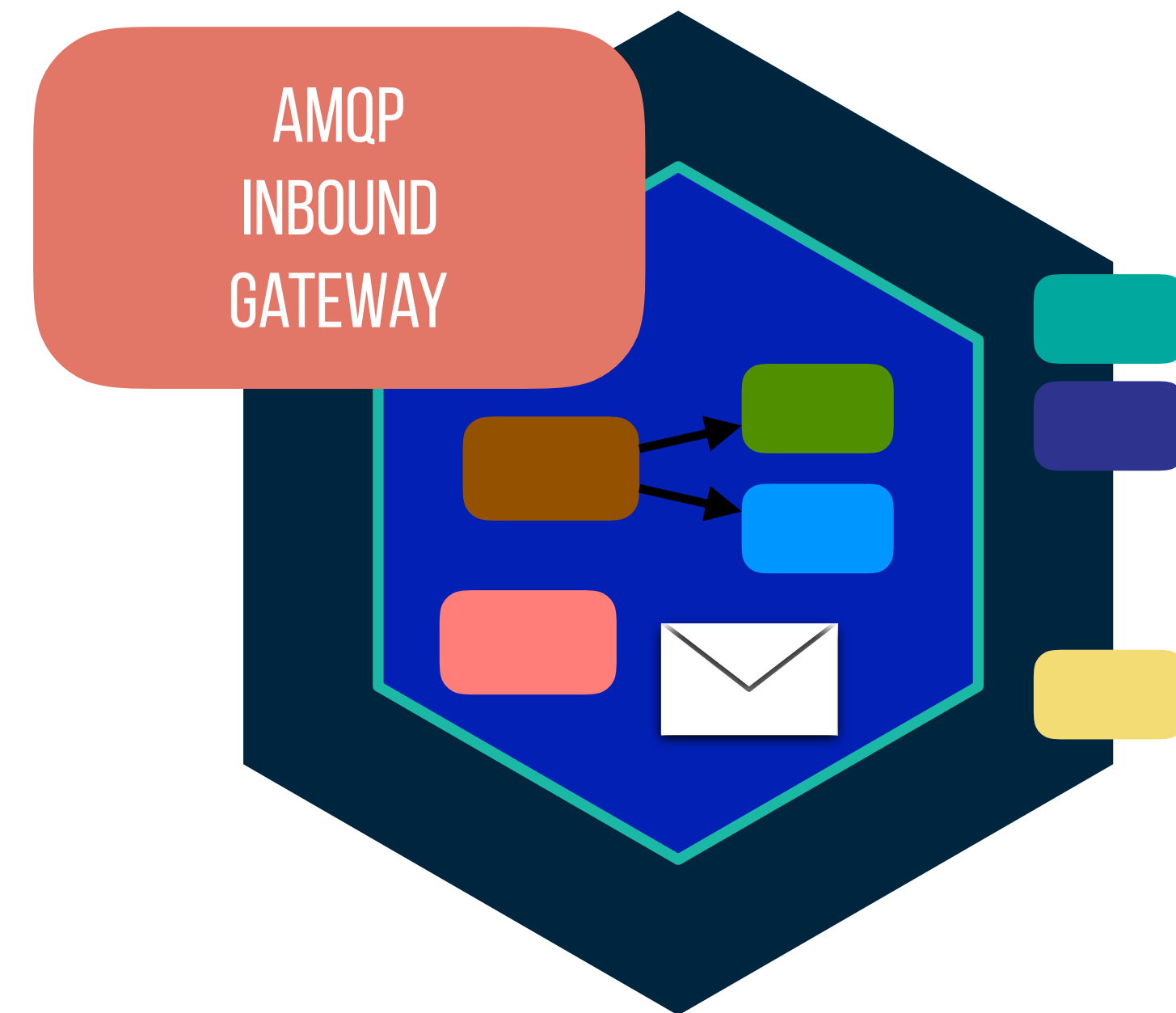
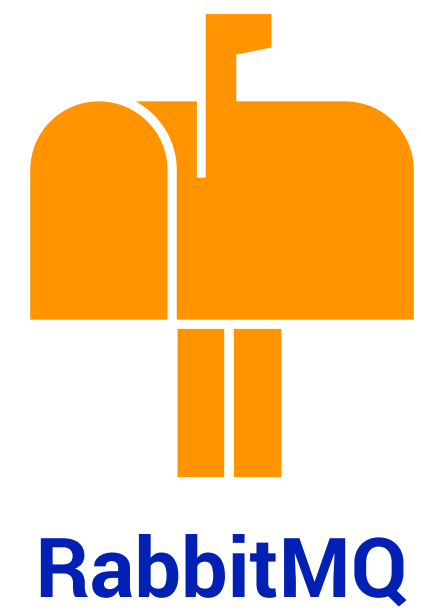
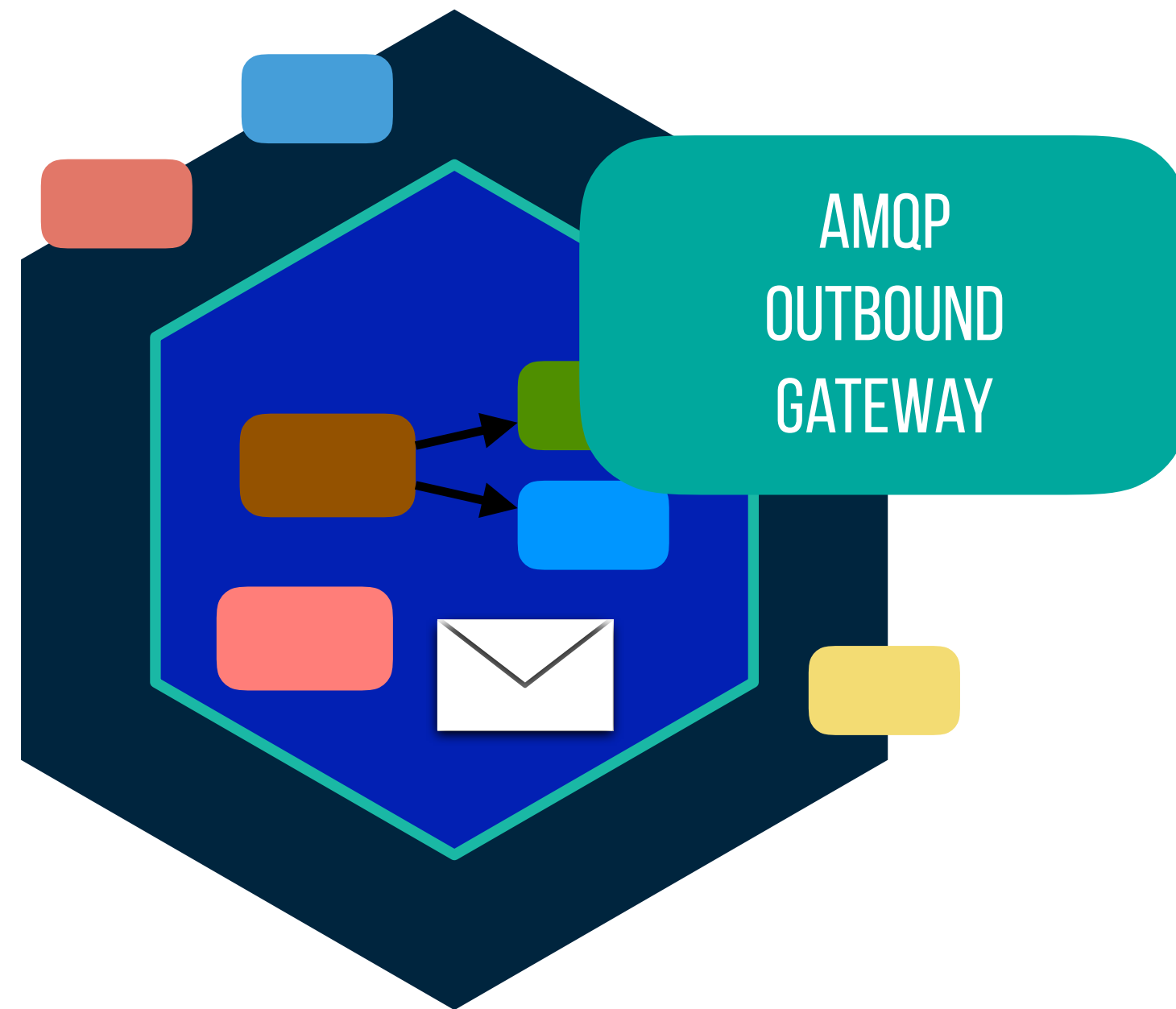




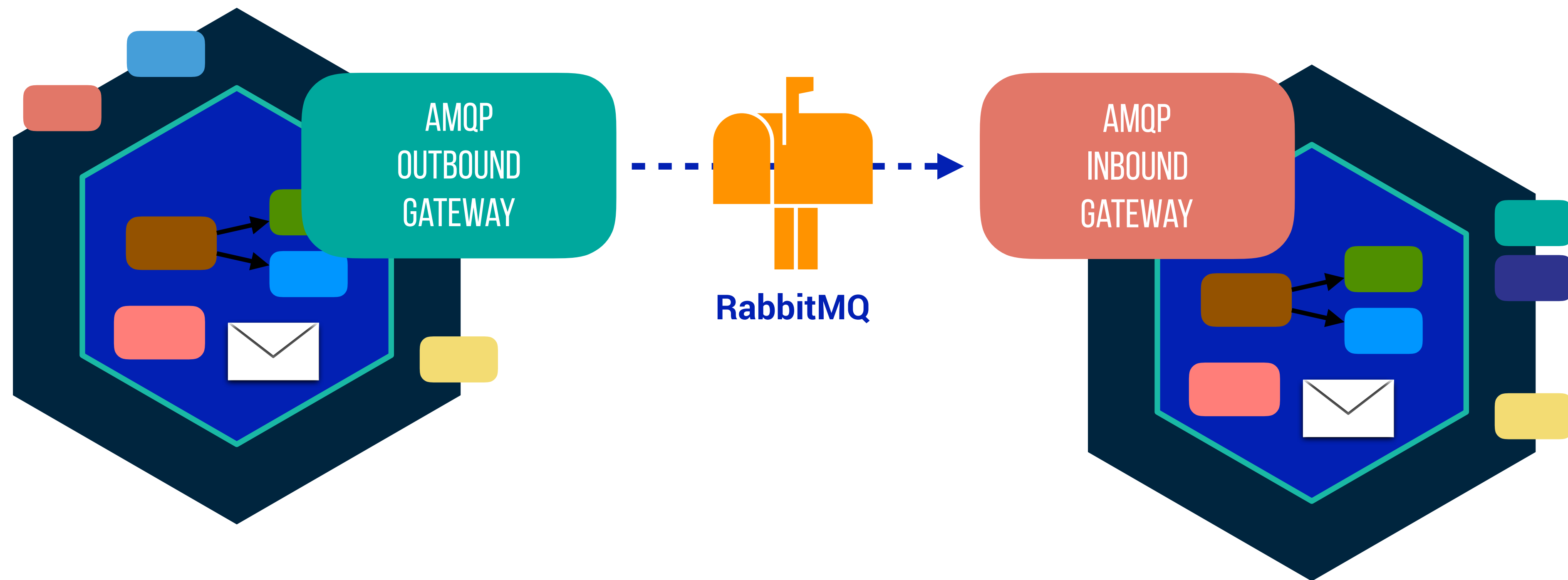
# FIRE AND FORGET



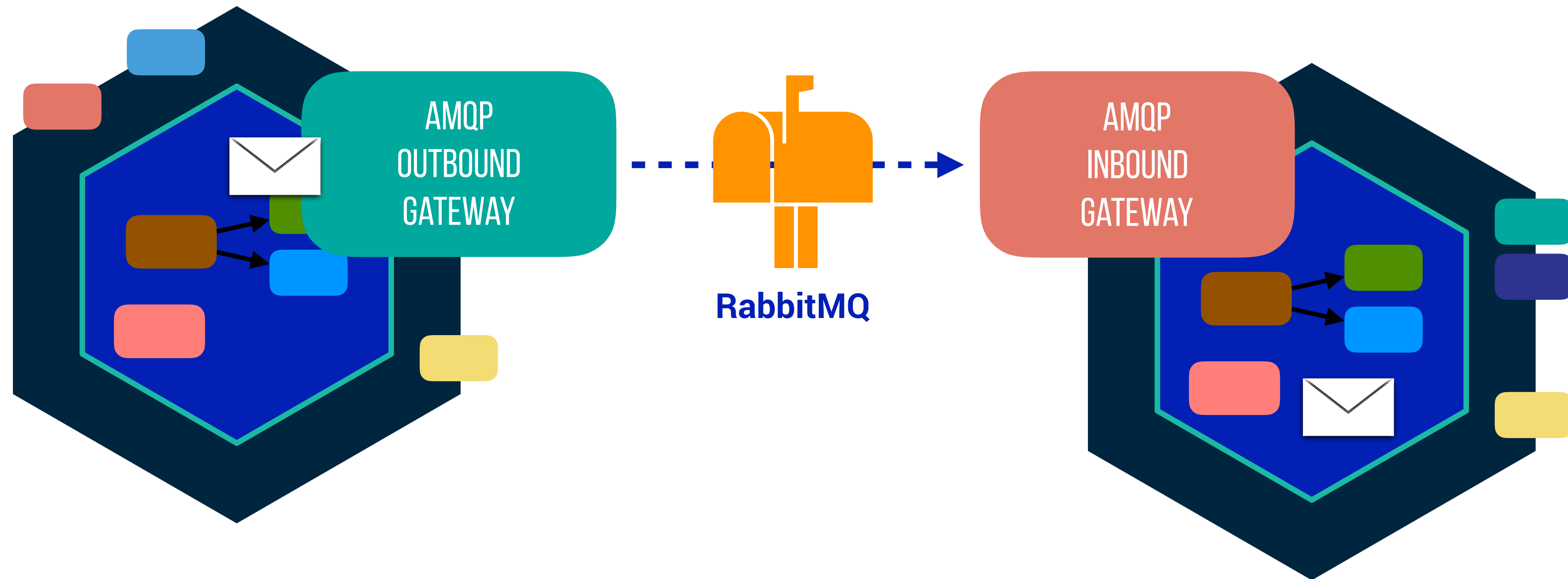
# FIRE AND FORGET



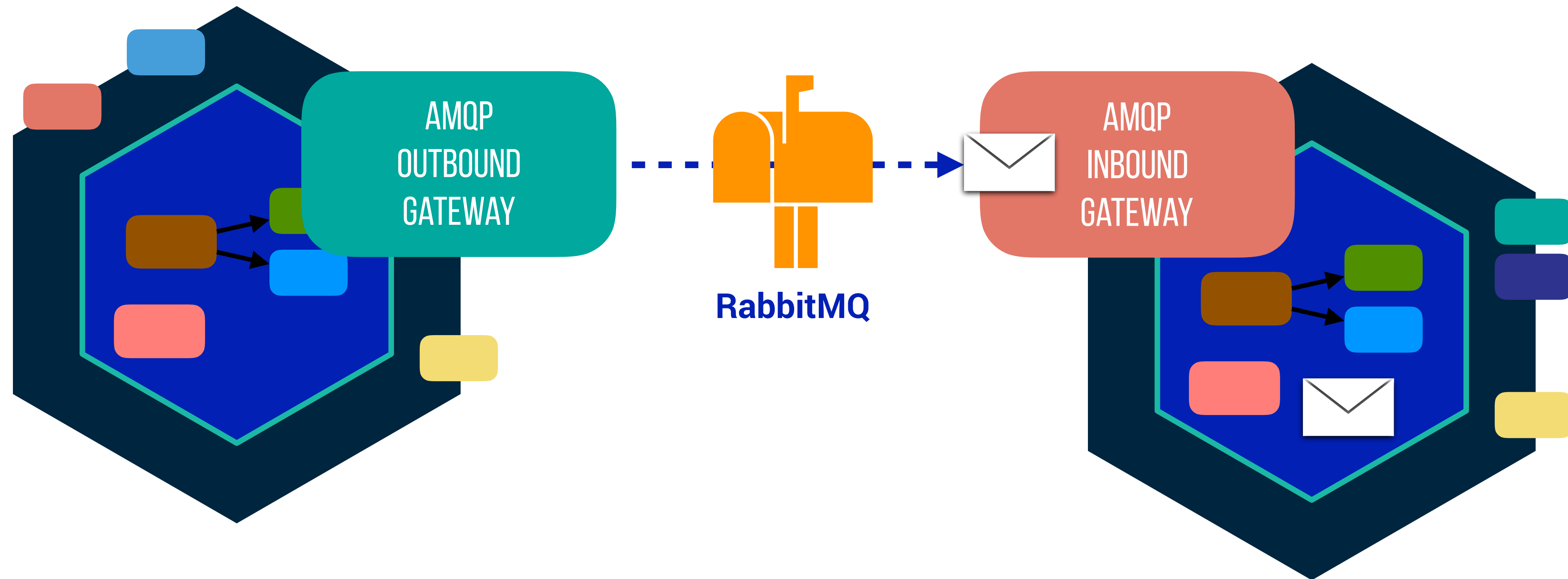
# FIRE AND FORGET



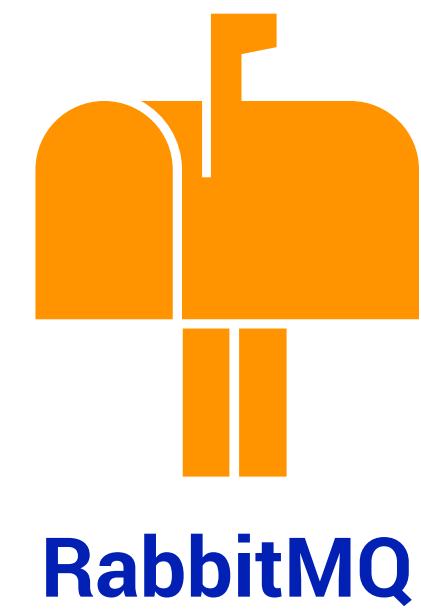
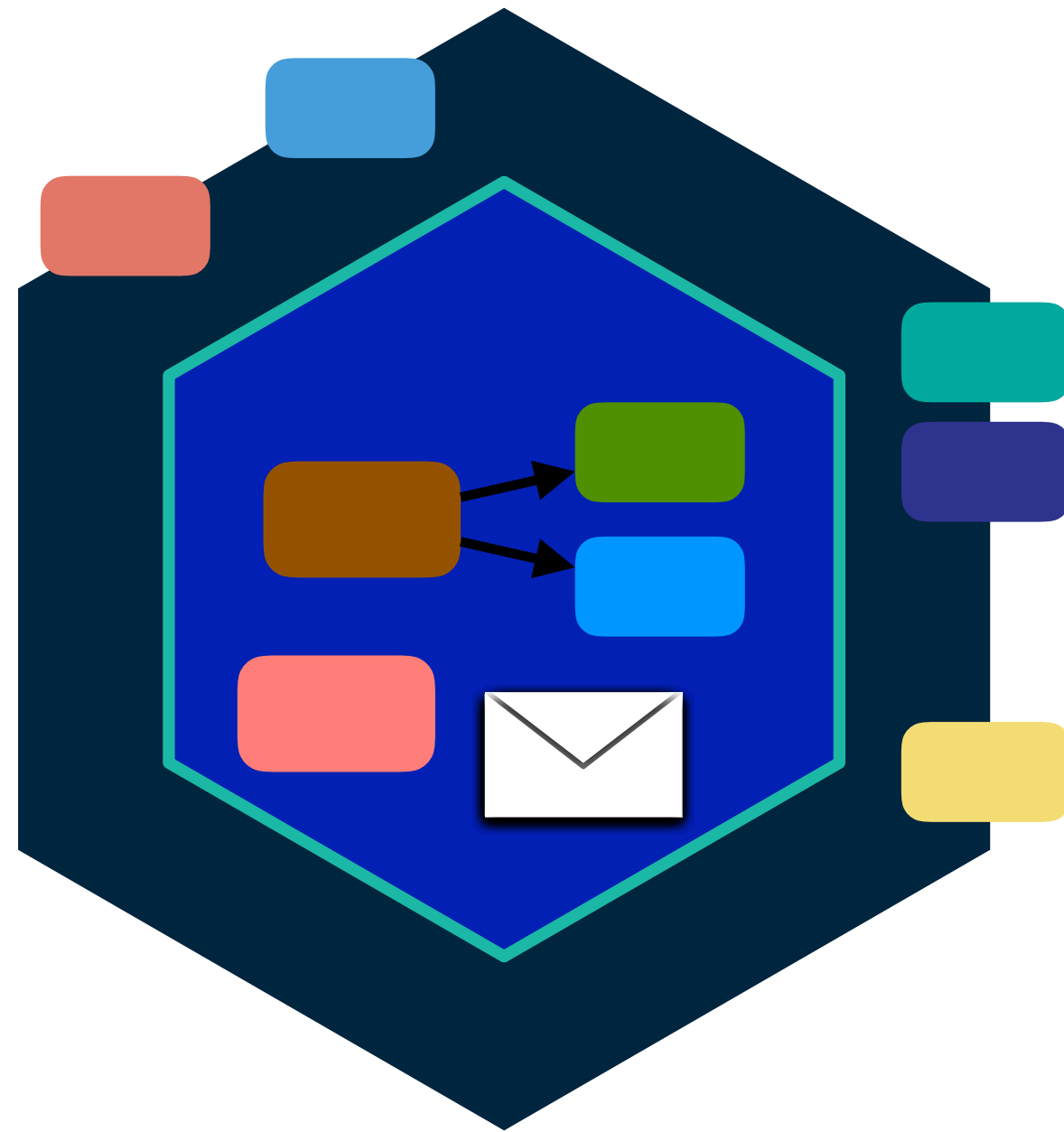
# FIRE AND FORGET



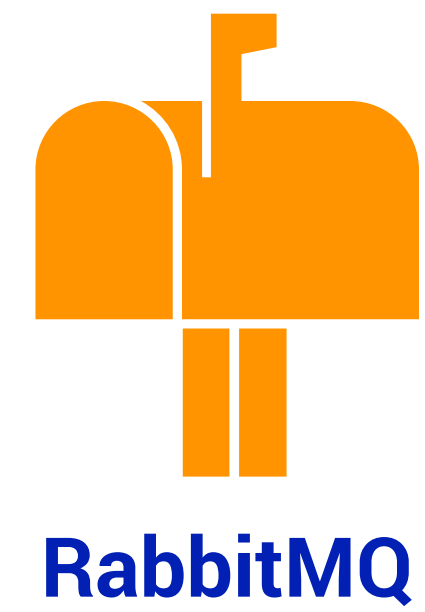
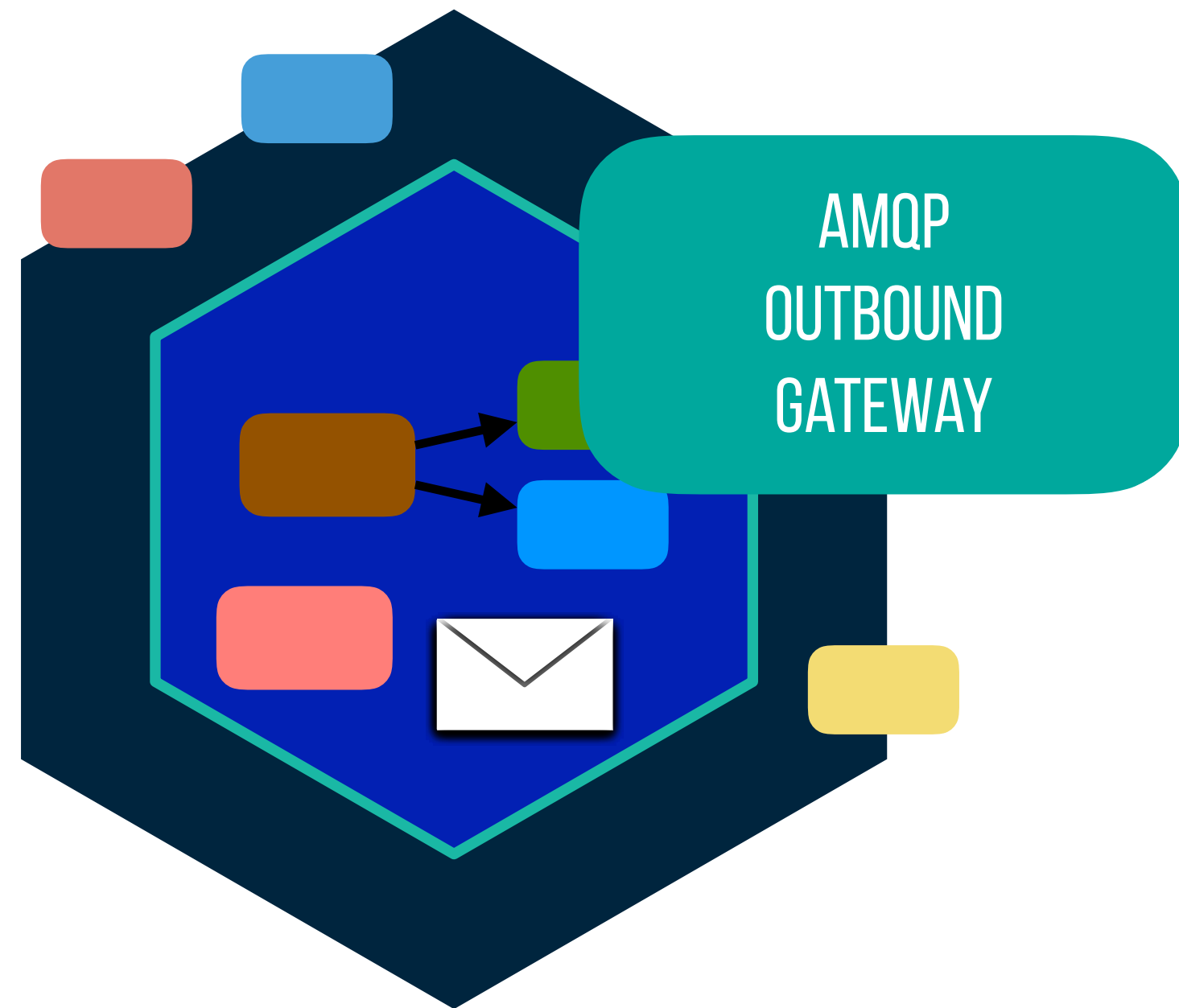
# FIRE AND FORGET



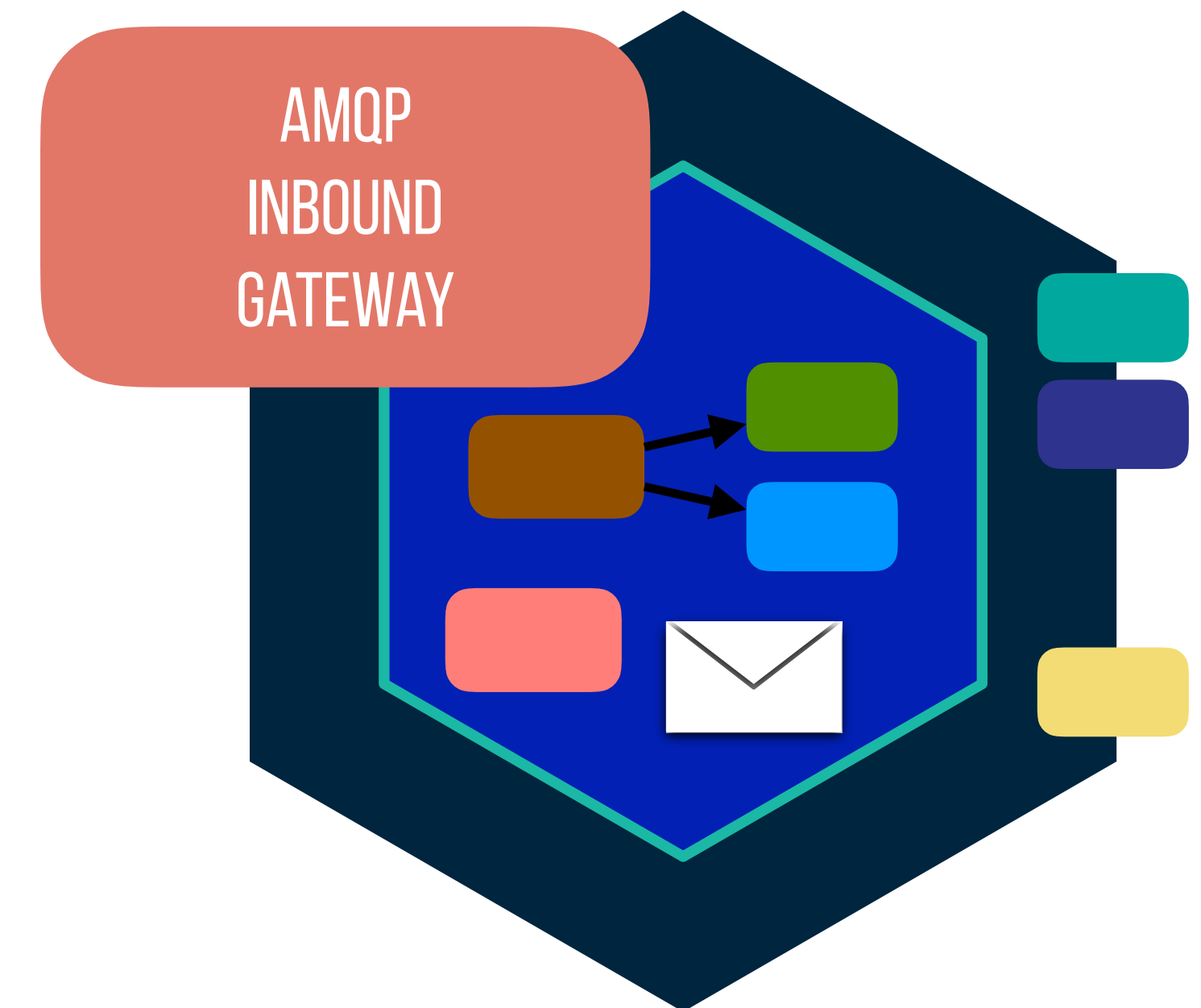
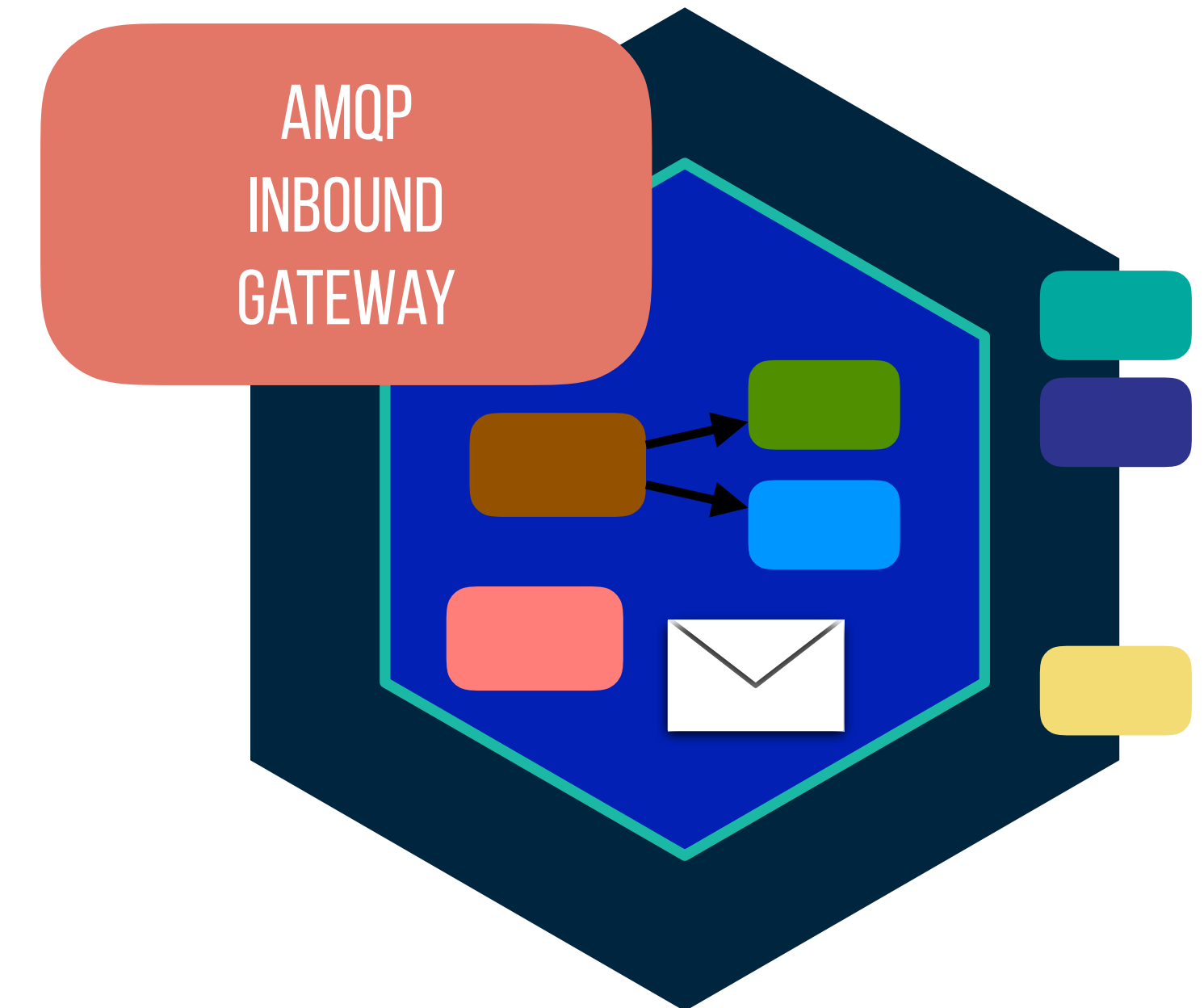
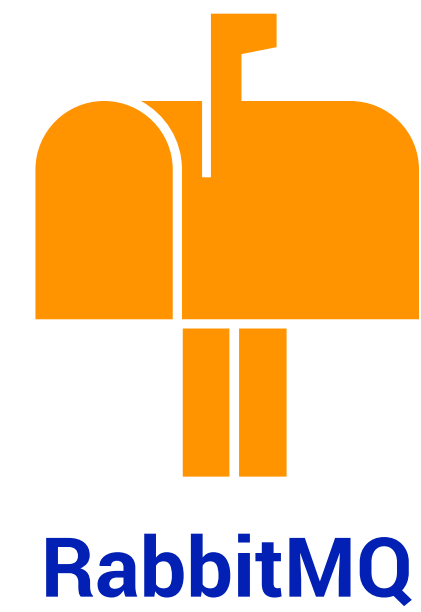
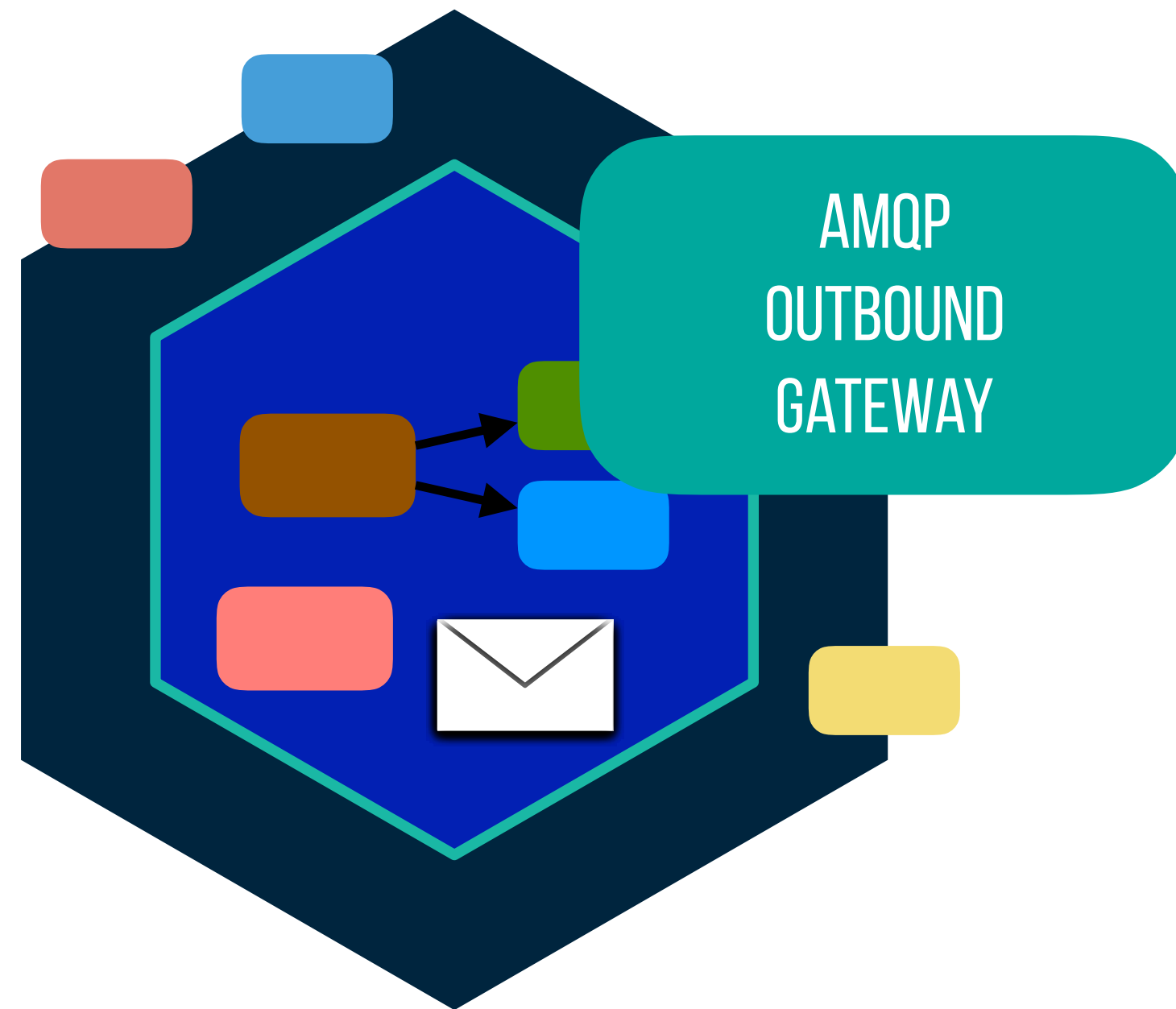
# QUEUES / LOAD BALANCING



# QUEUES / LOAD BALANCING

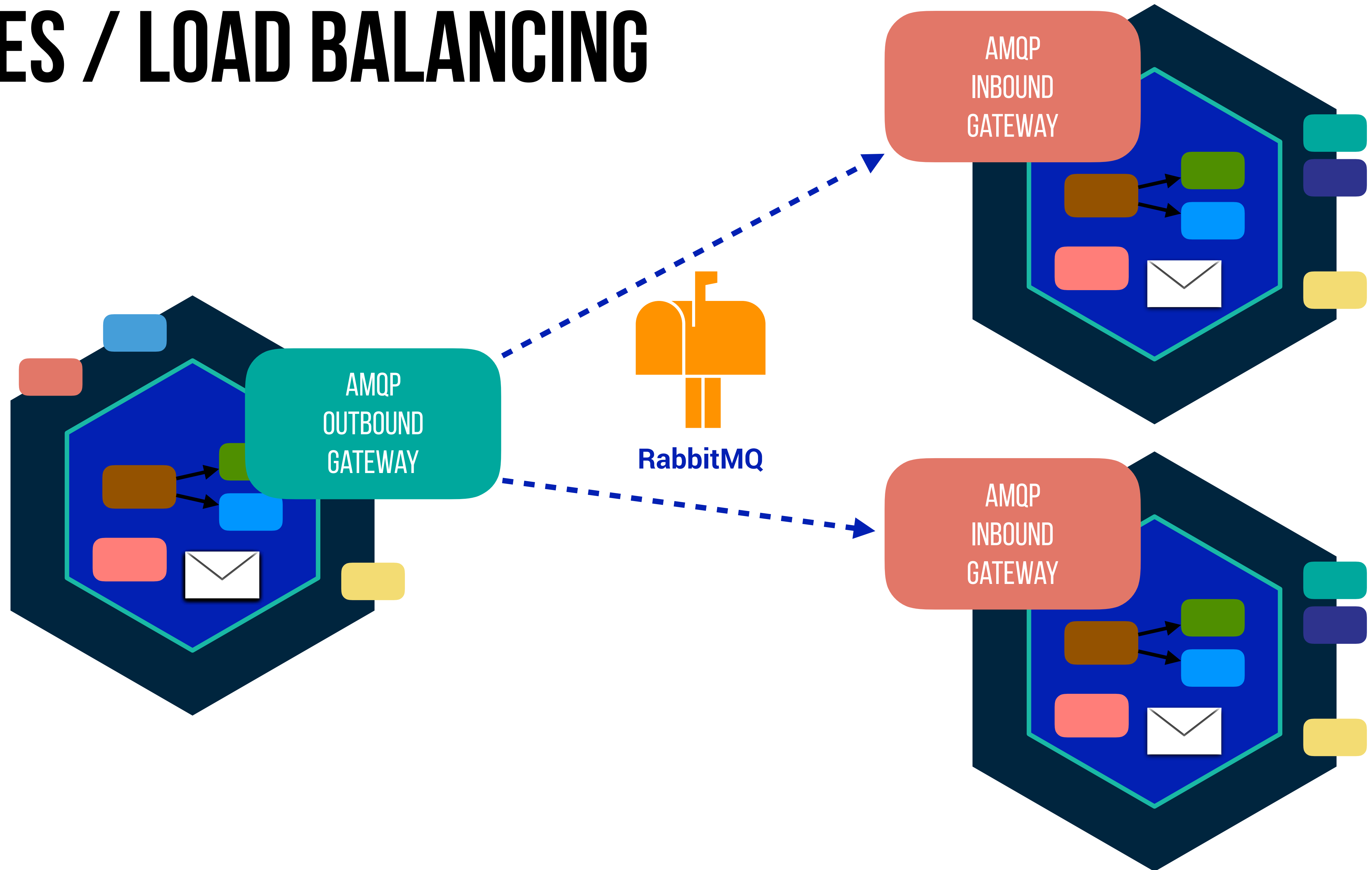


# QUEUES / LOAD BALANCING

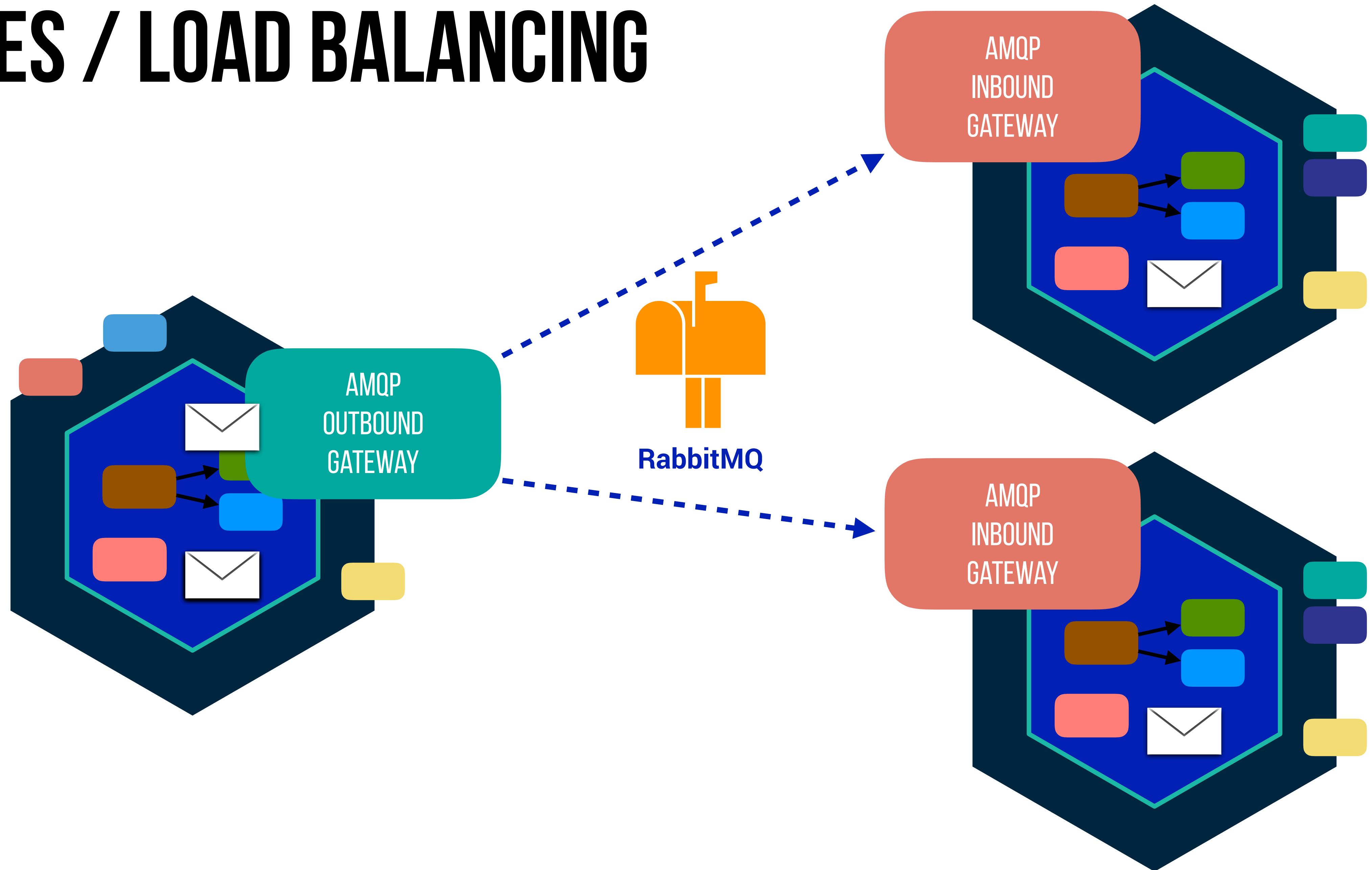




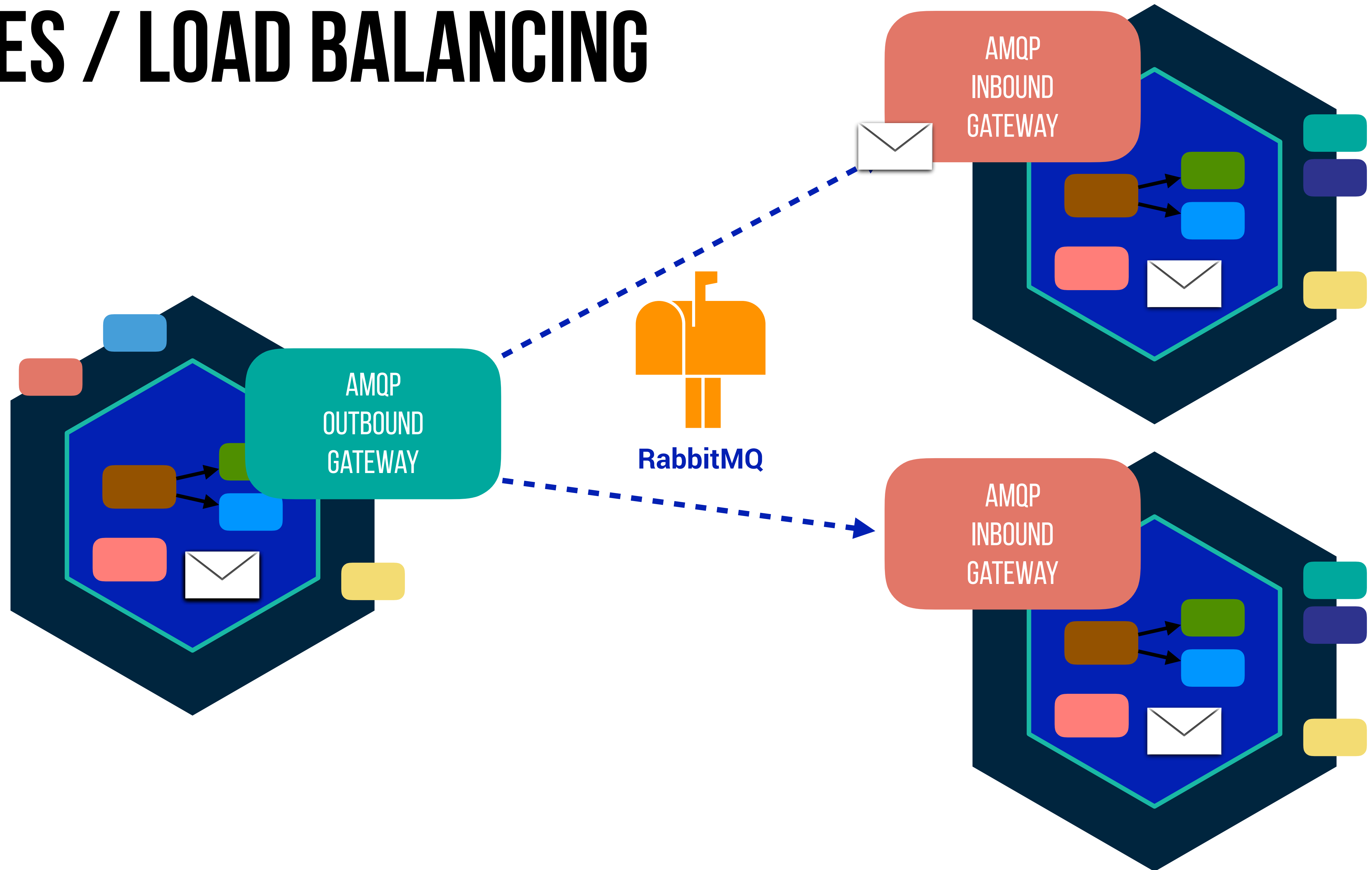
# QUEUES / LOAD BALANCING



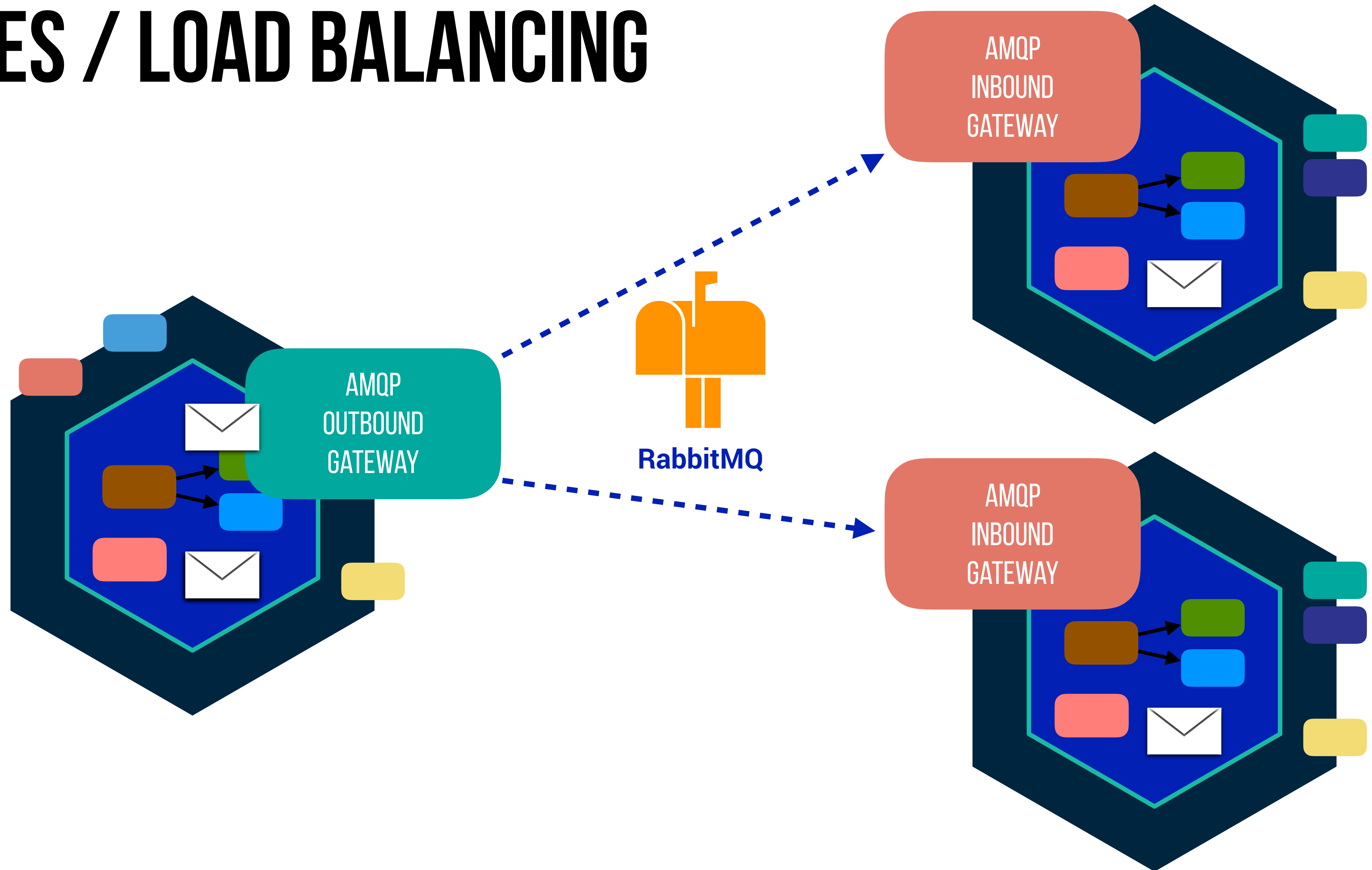
# QUEUES / LOAD BALANCING



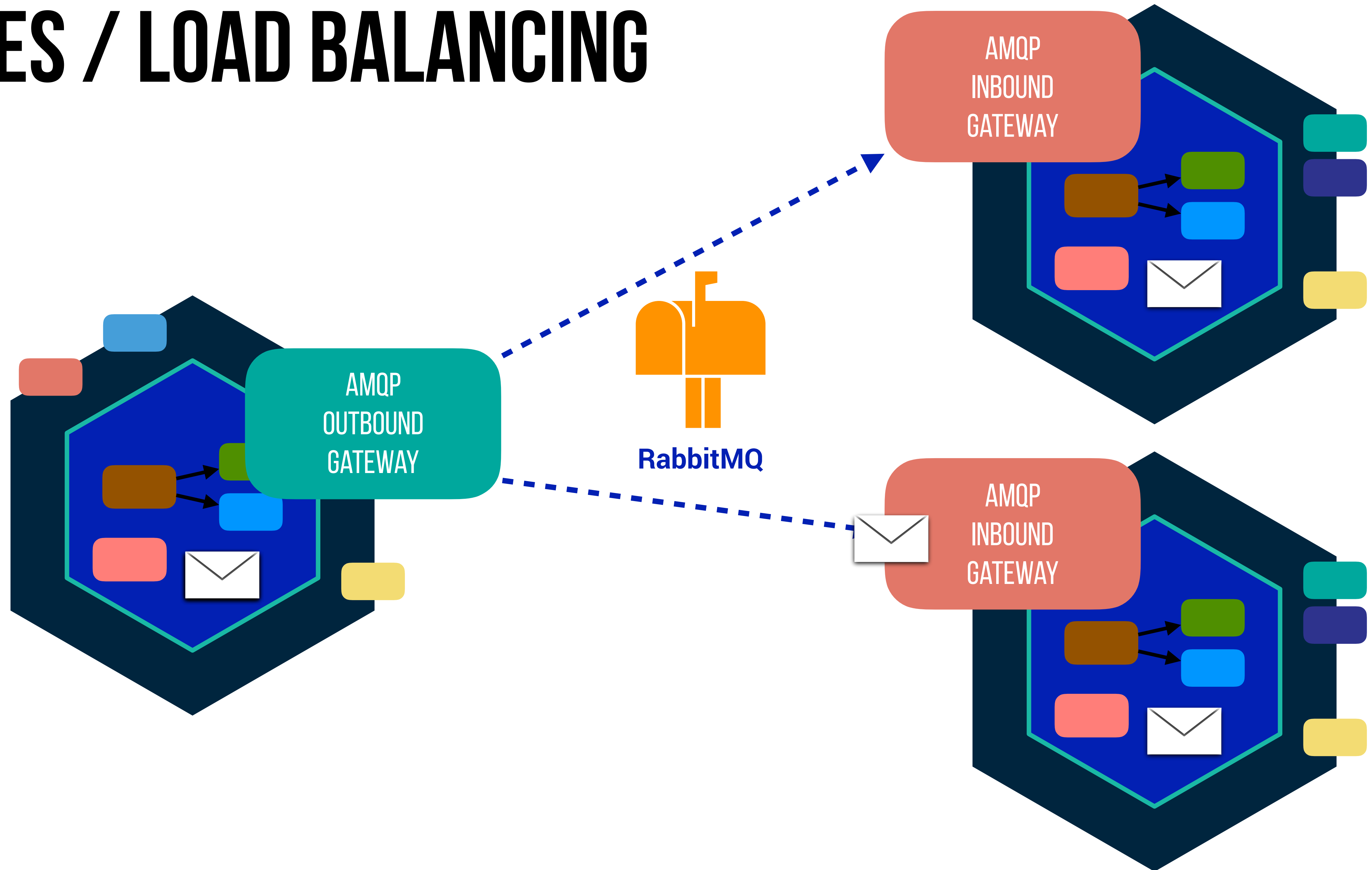
# QUEUES / LOAD BALANCING



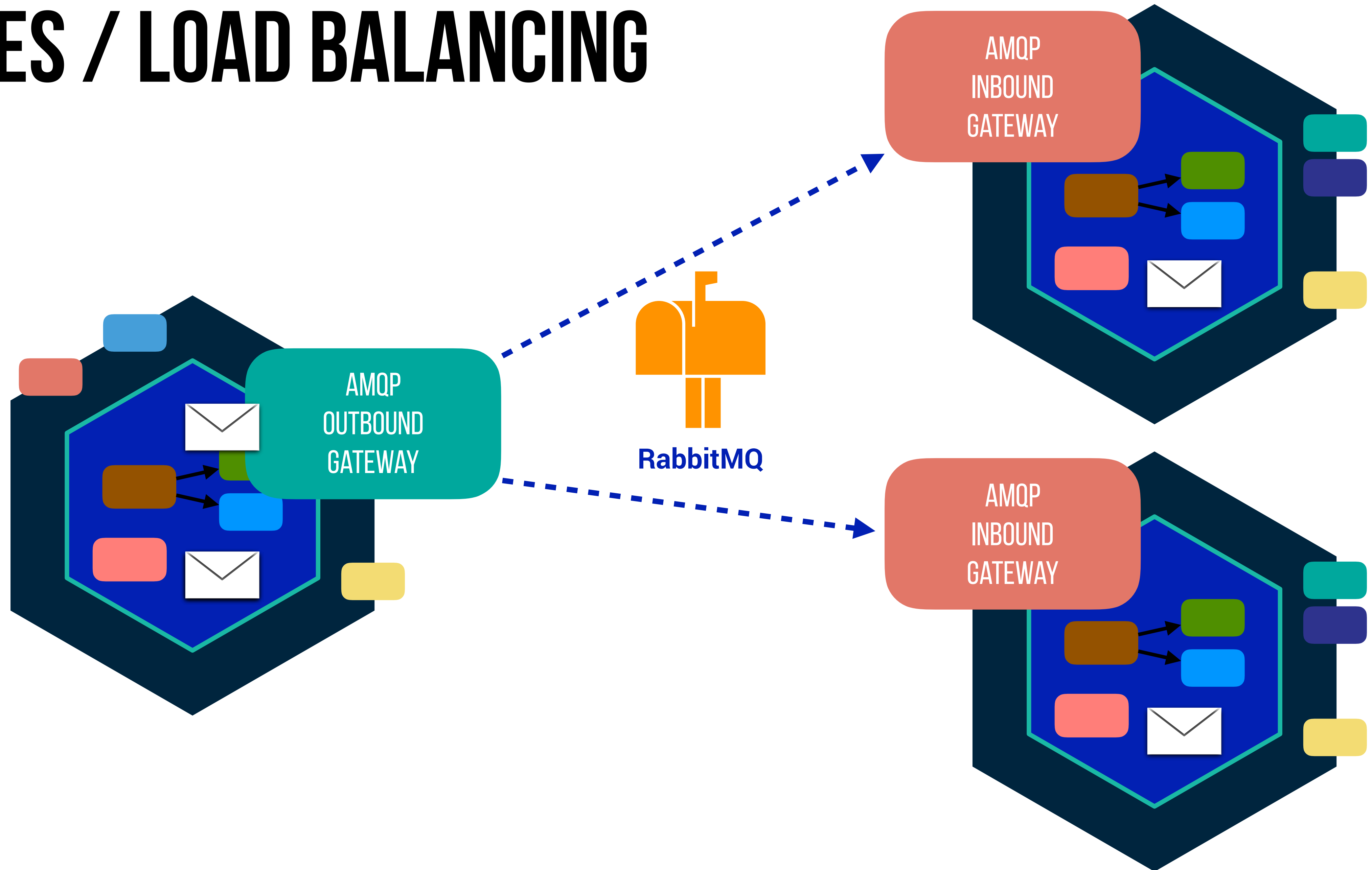
# QUEUES / LOAD BALANCING



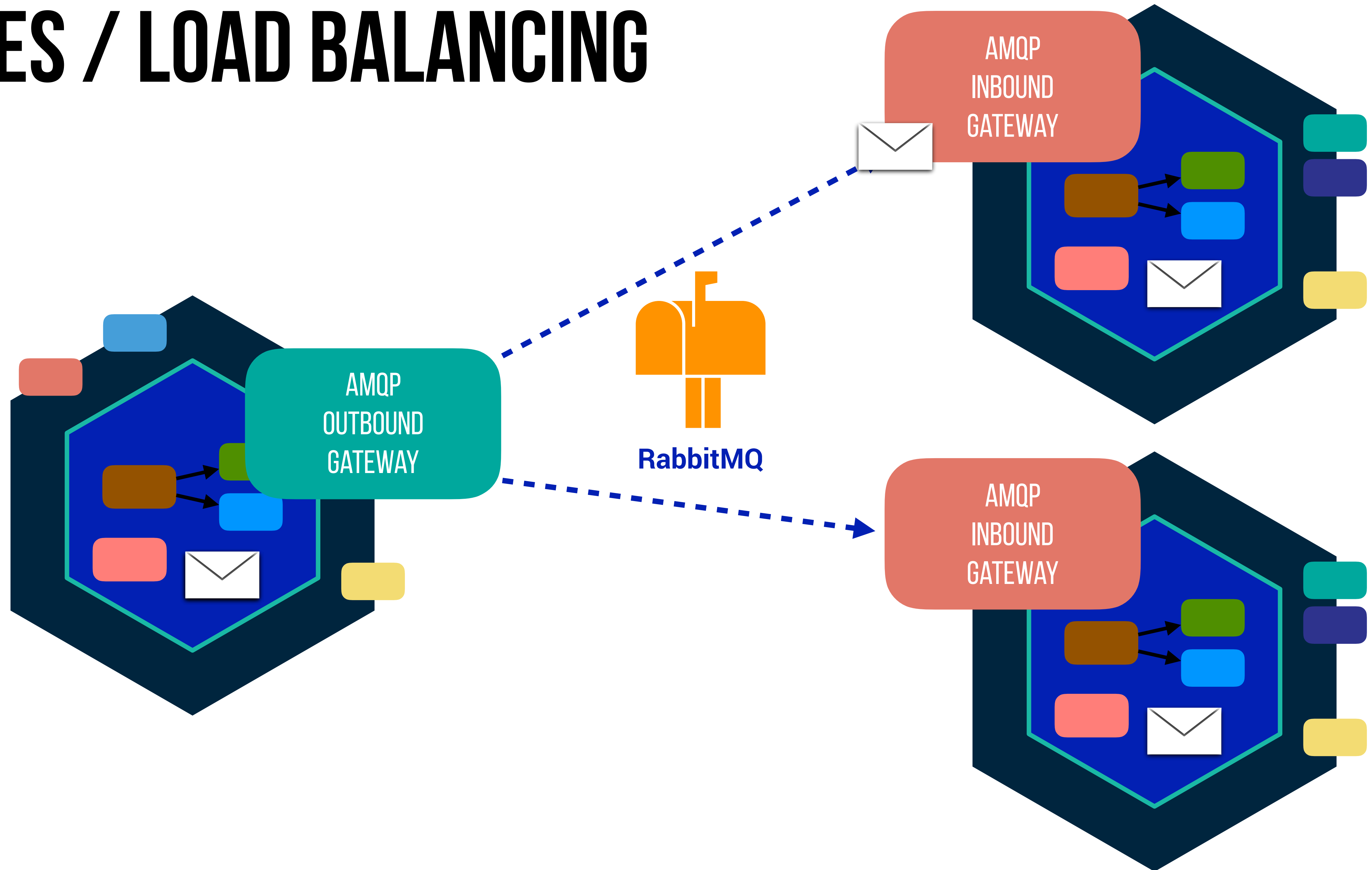
# QUEUES / LOAD BALANCING



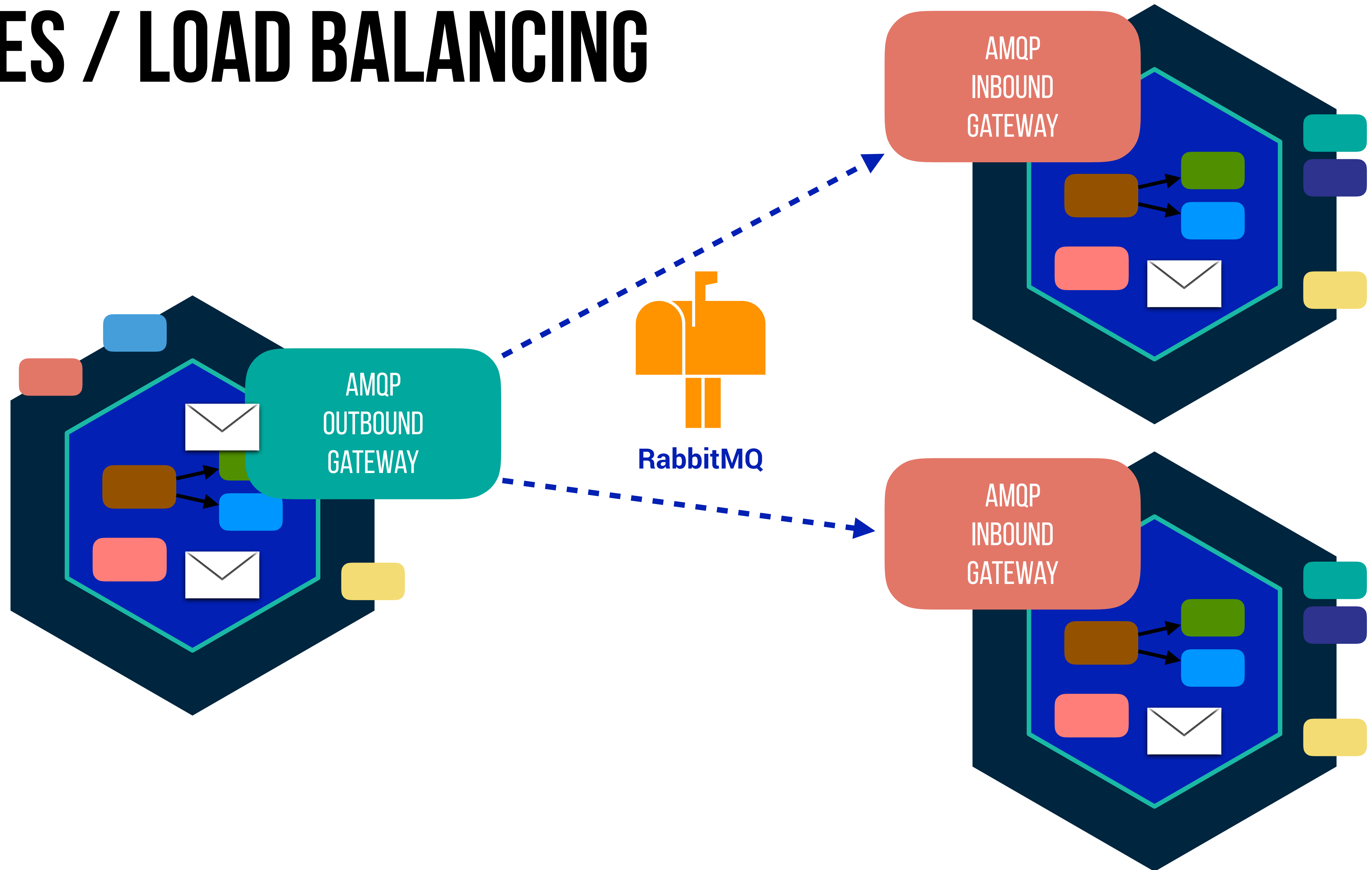
# QUEUES / LOAD BALANCING



# QUEUES / LOAD BALANCING

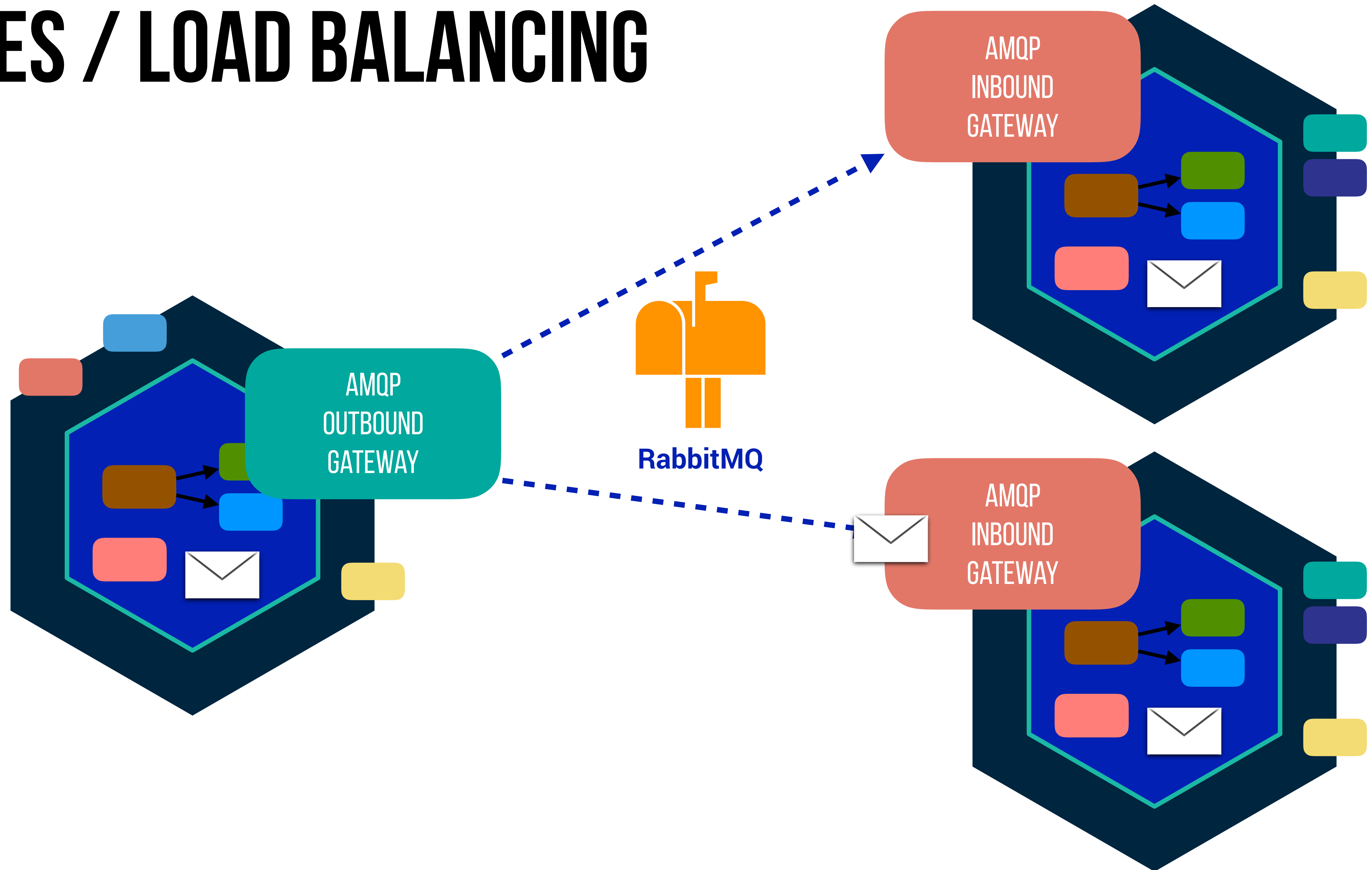


# QUEUES / LOAD BALANCING

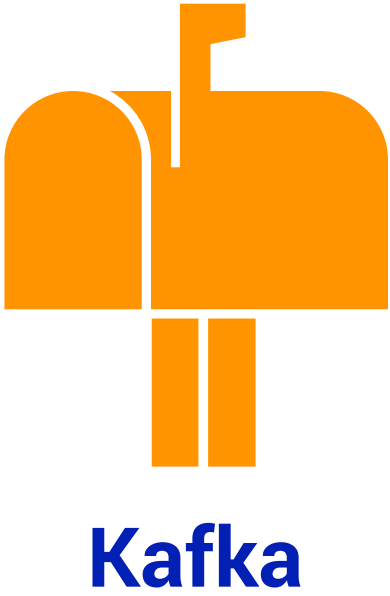




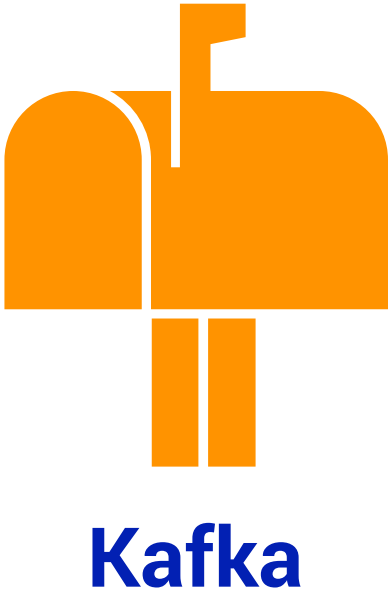
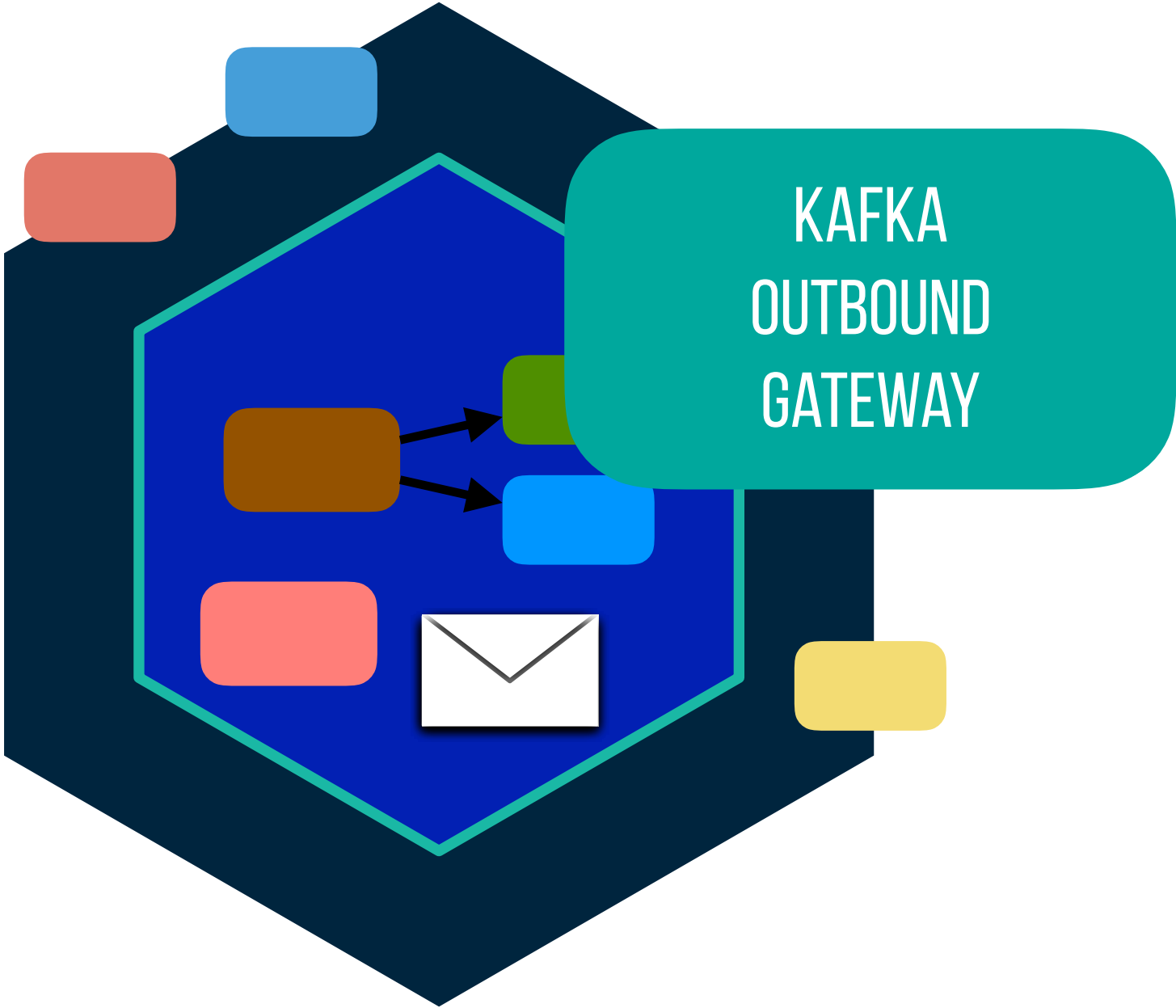
# QUEUES / LOAD BALANCING



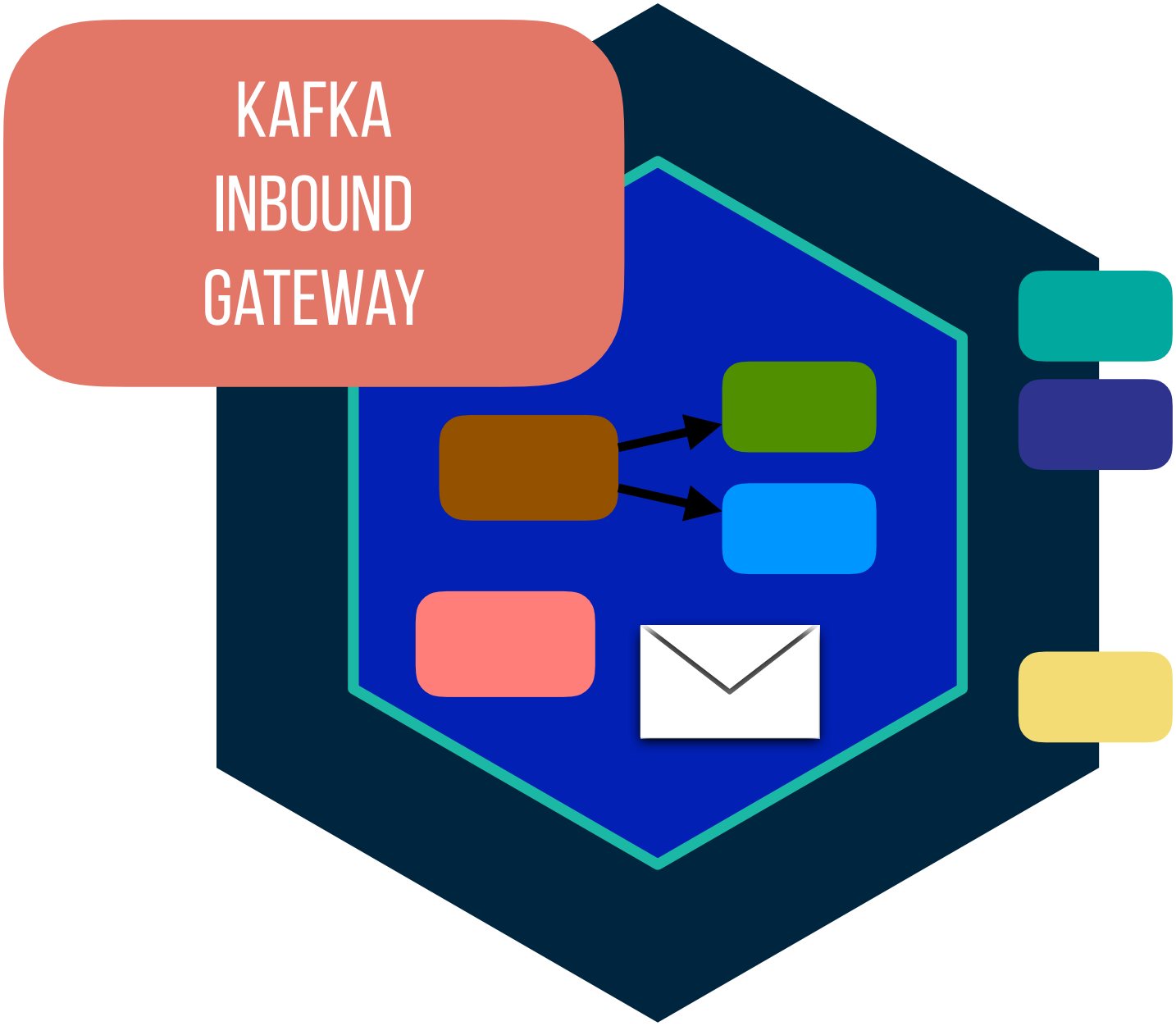
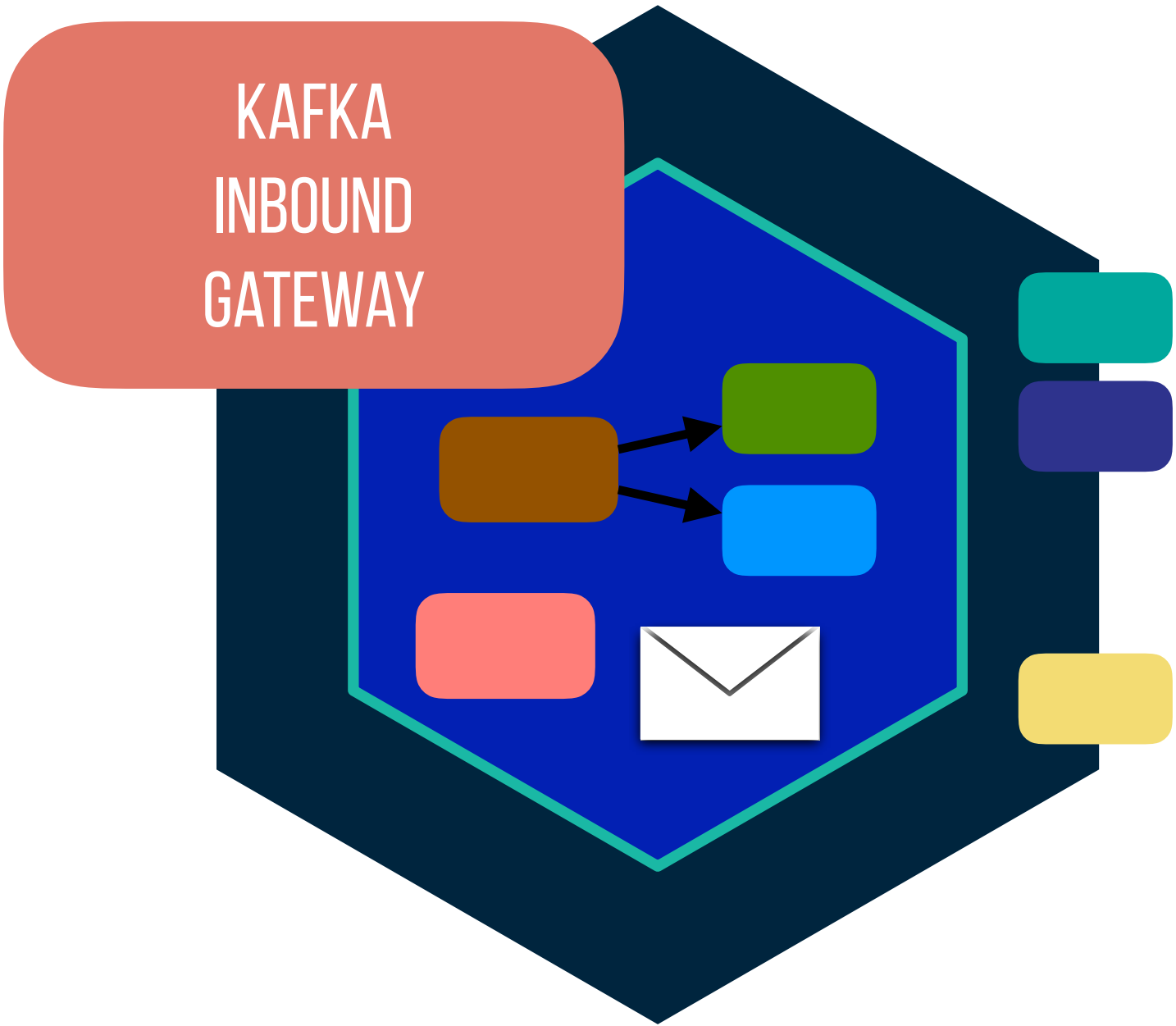
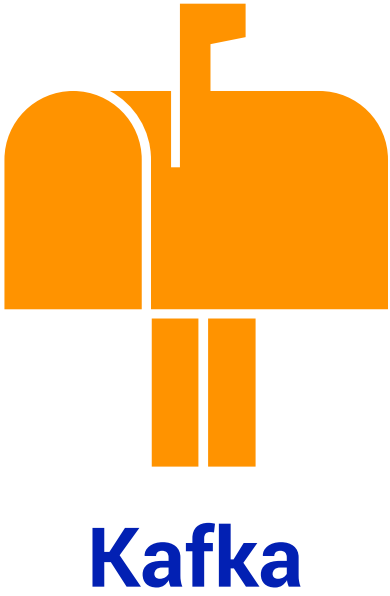
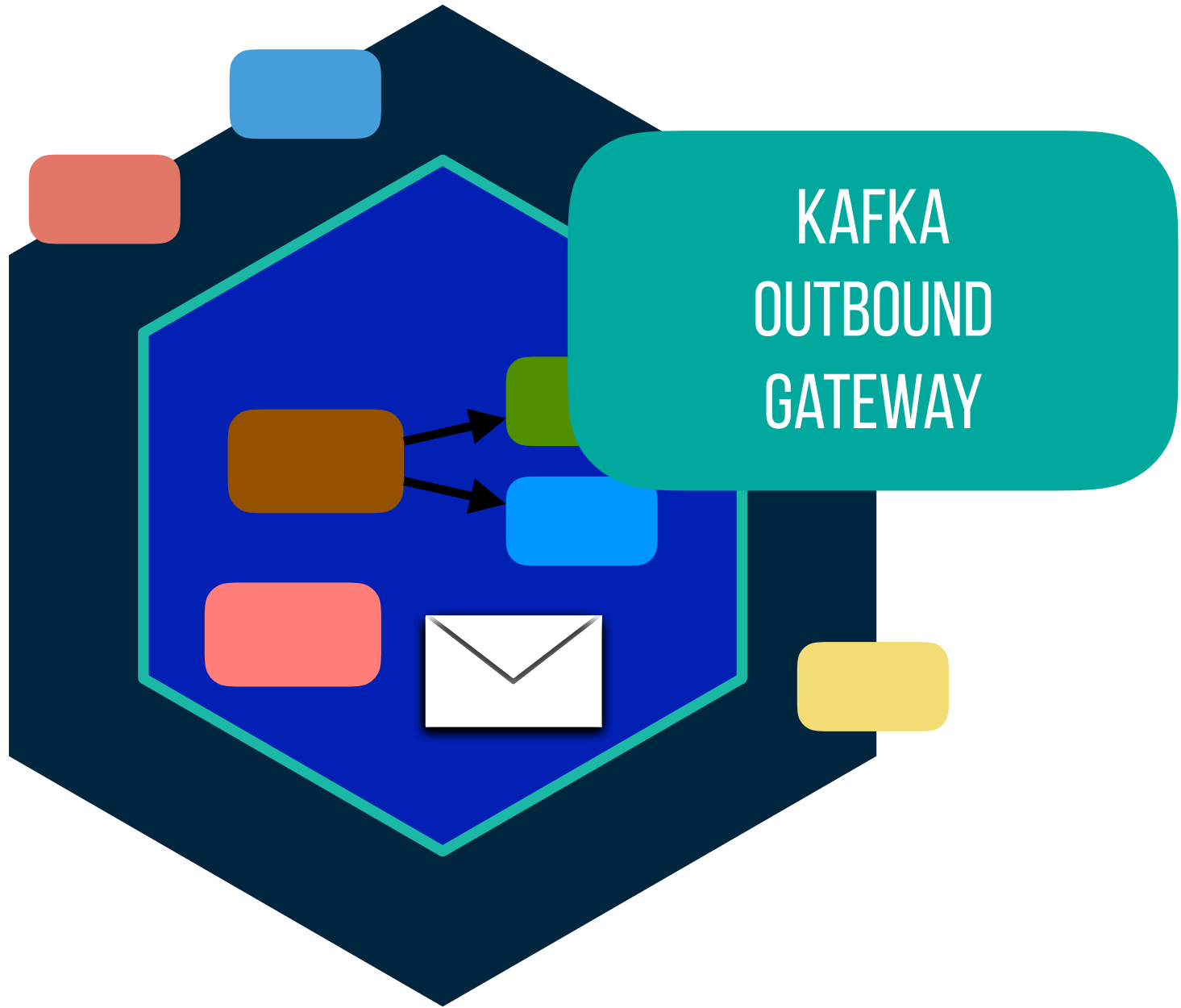
# PUBLISH - SUBSCRIBE



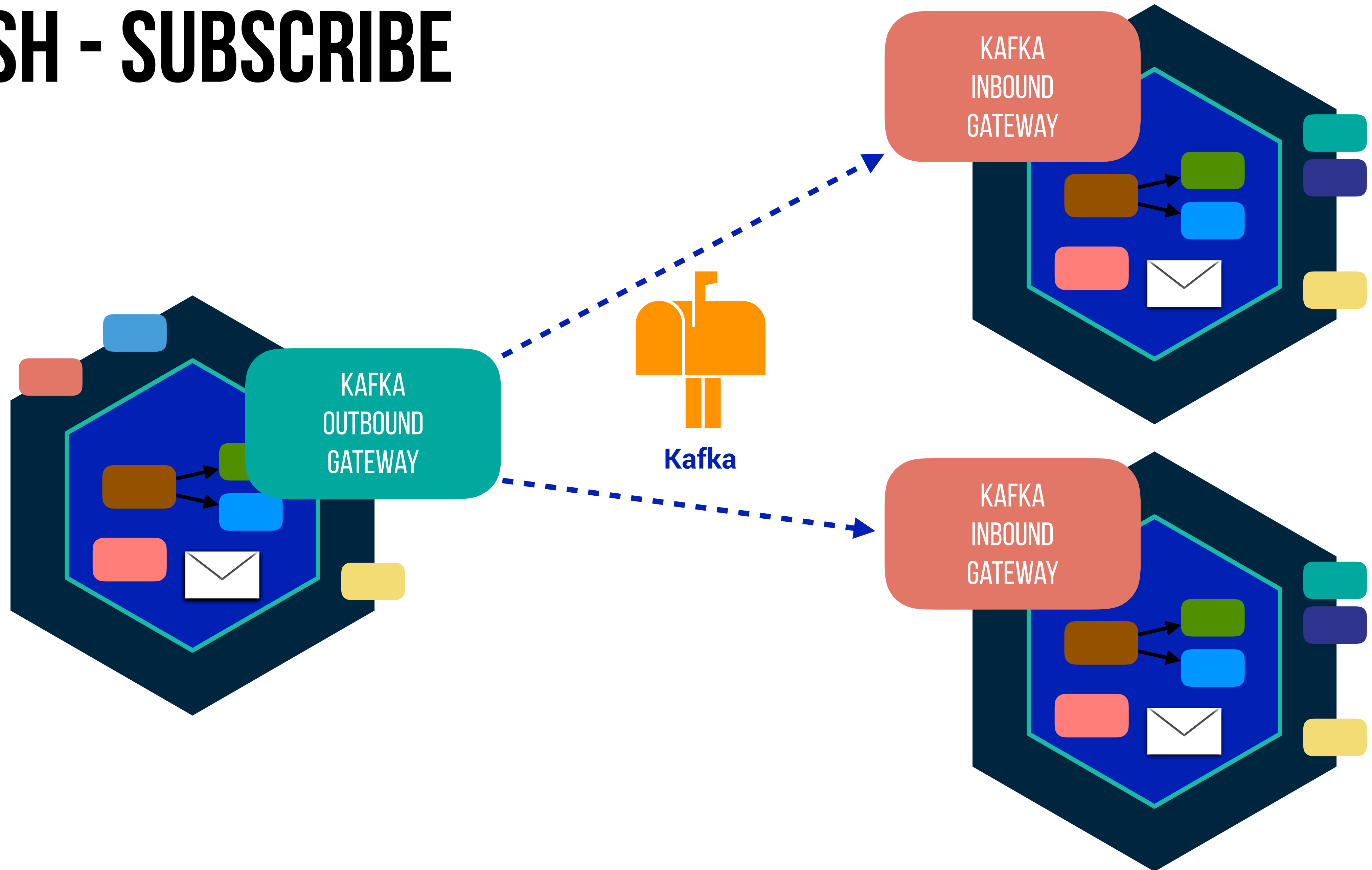
# PUBLISH - SUBSCRIBE



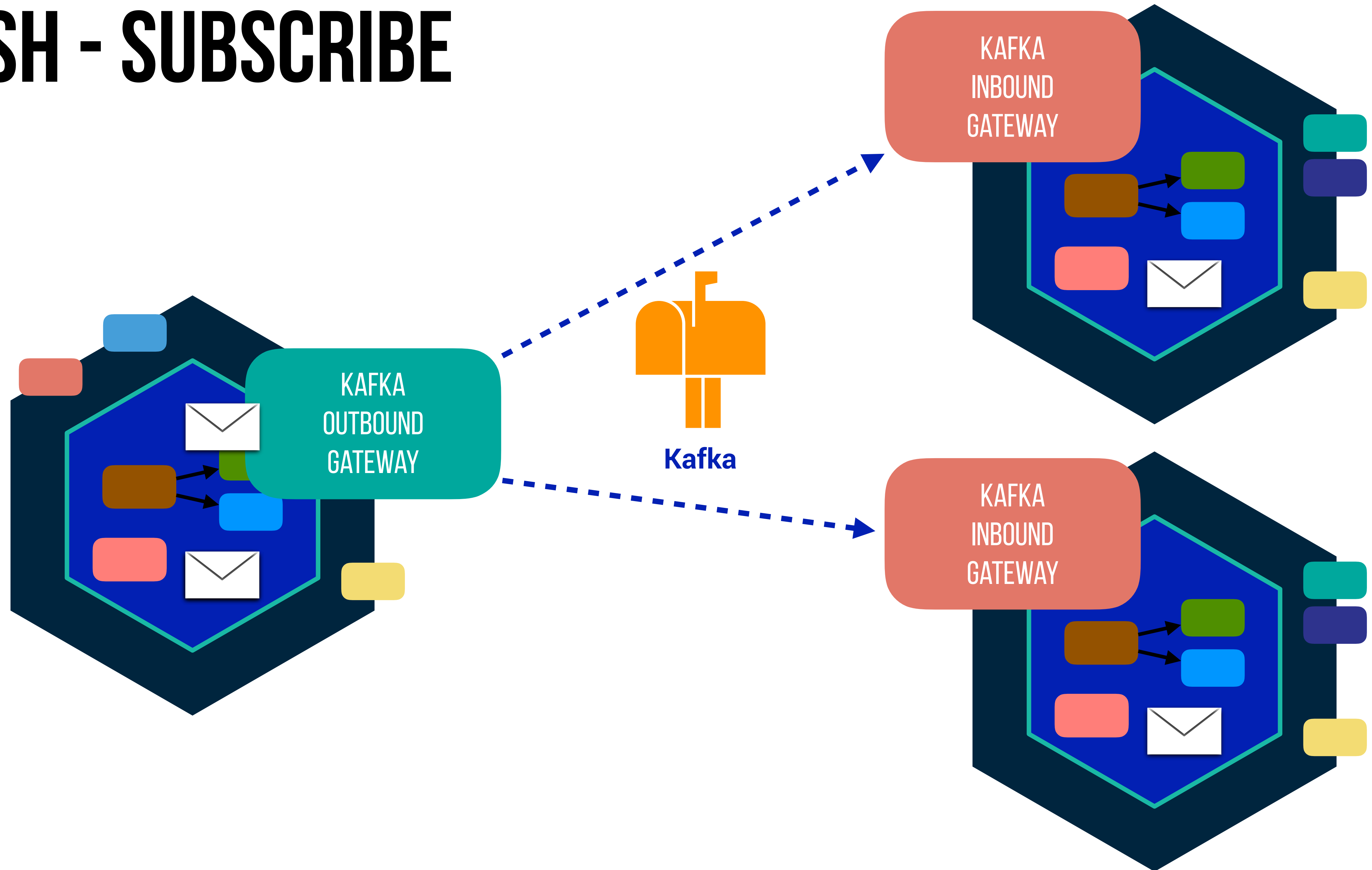
# PUBLISH - SUBSCRIBE



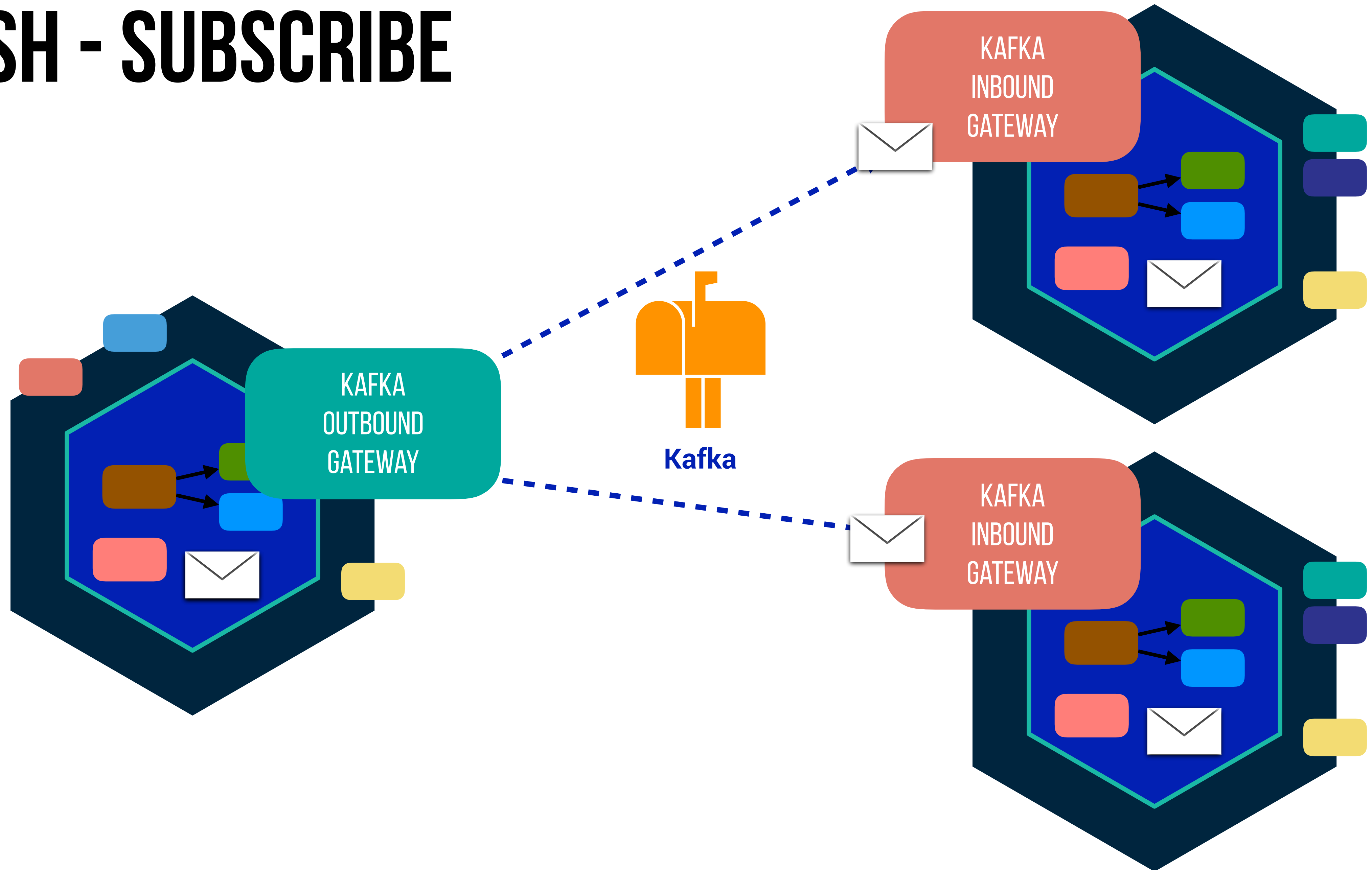
# PUBLISH - SUBSCRIBE



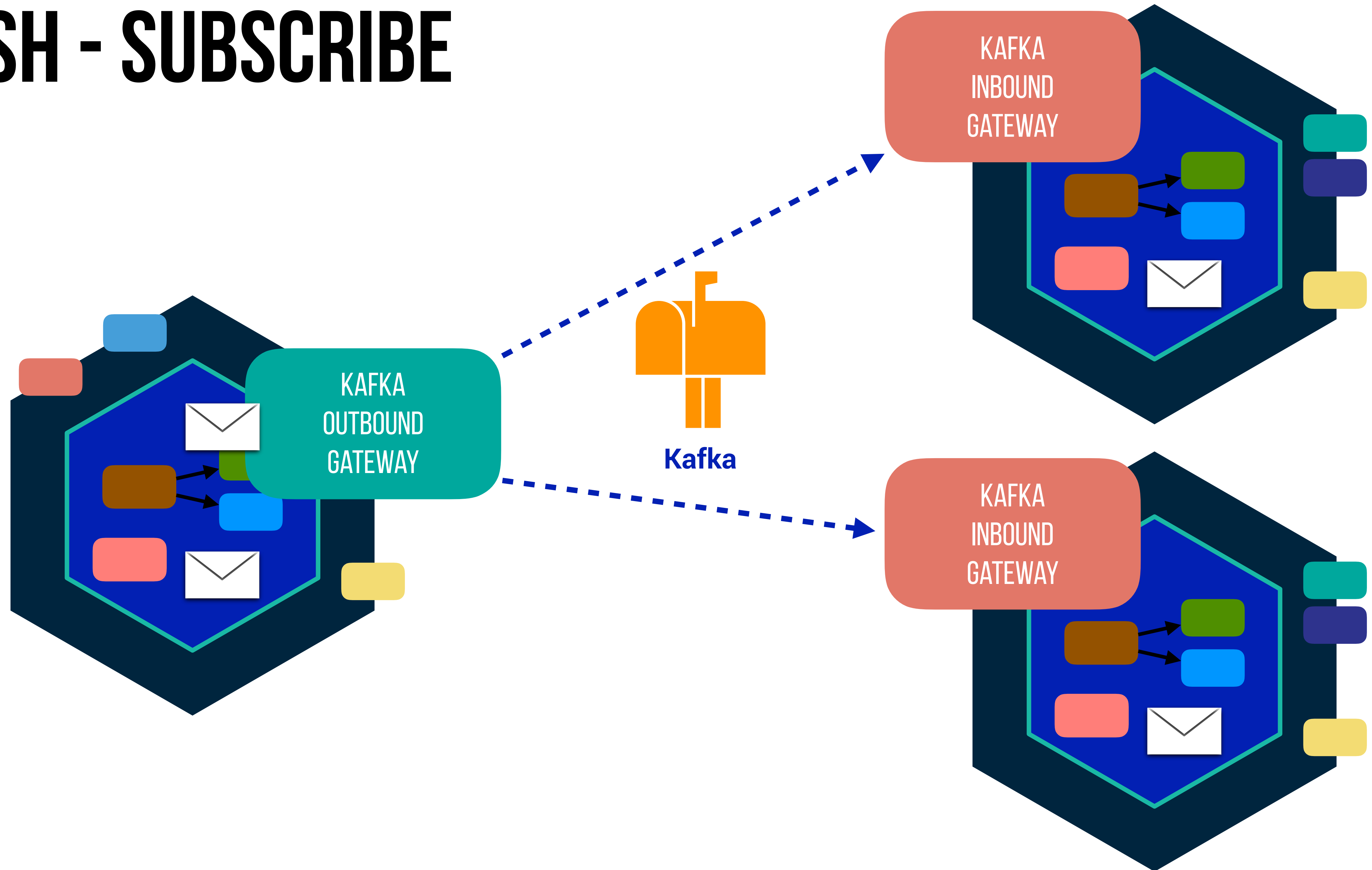
# PUBLISH - SUBSCRIBE



# PUBLISH - SUBSCRIBE

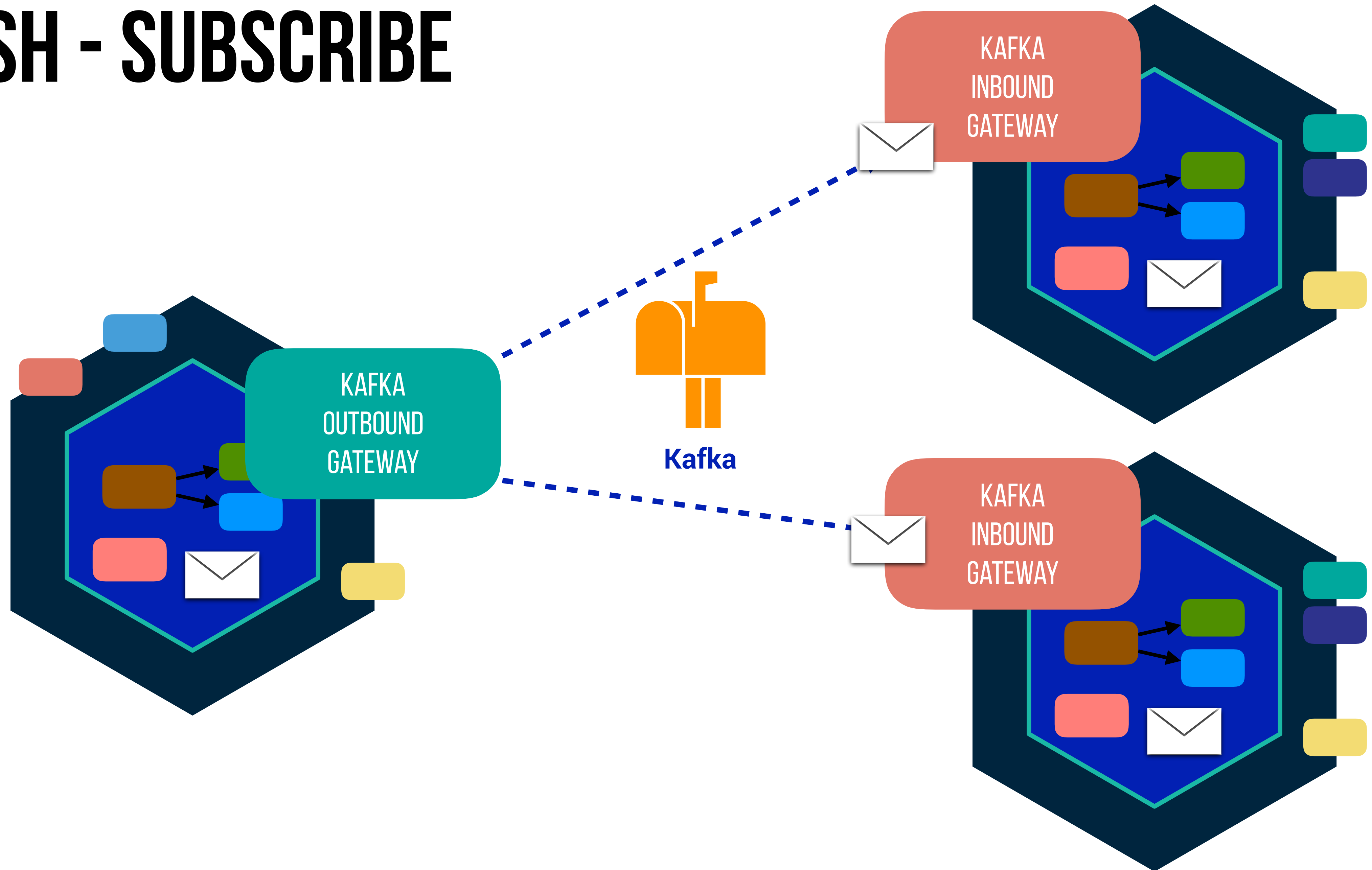


# PUBLISH - SUBSCRIBE

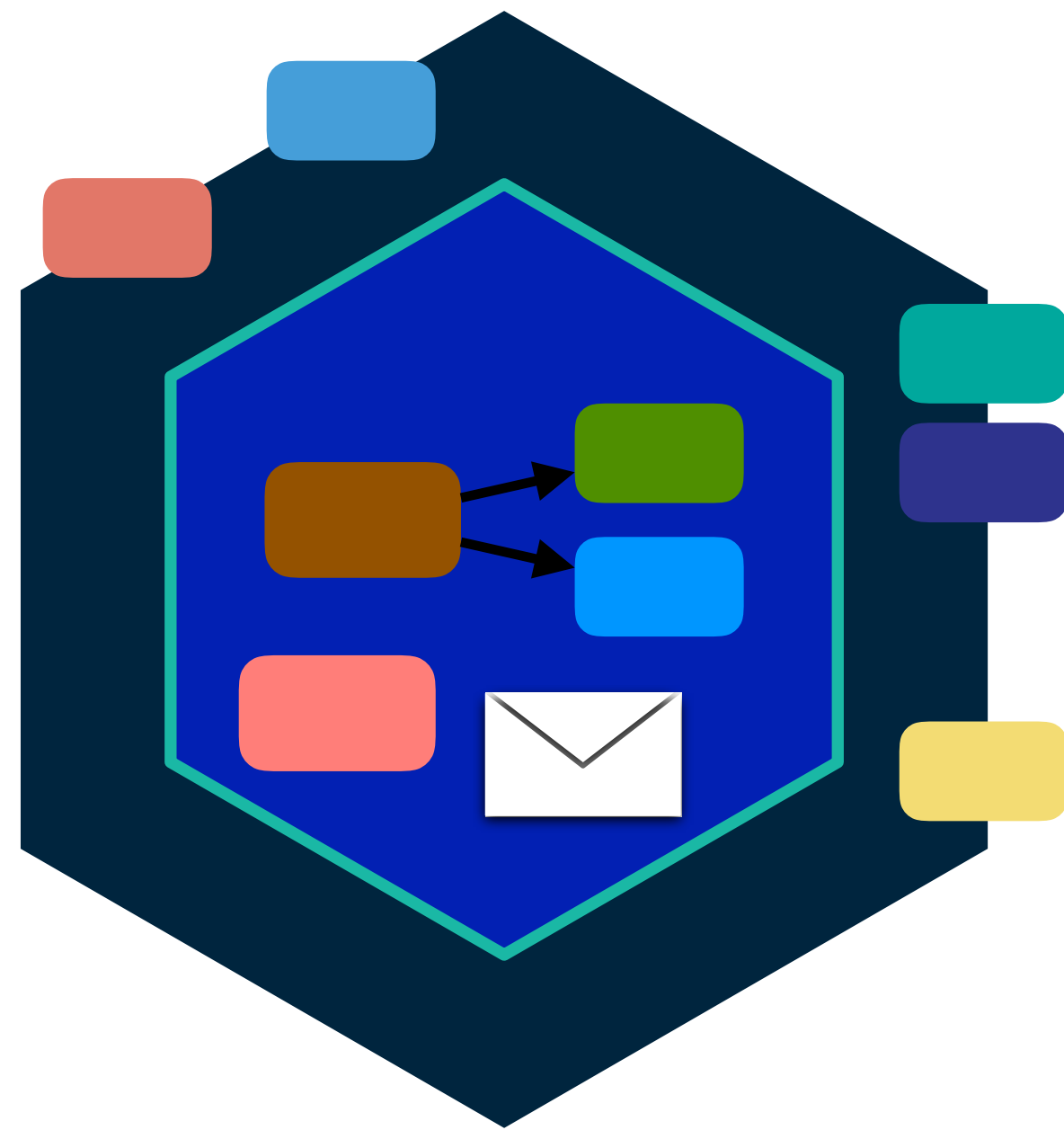




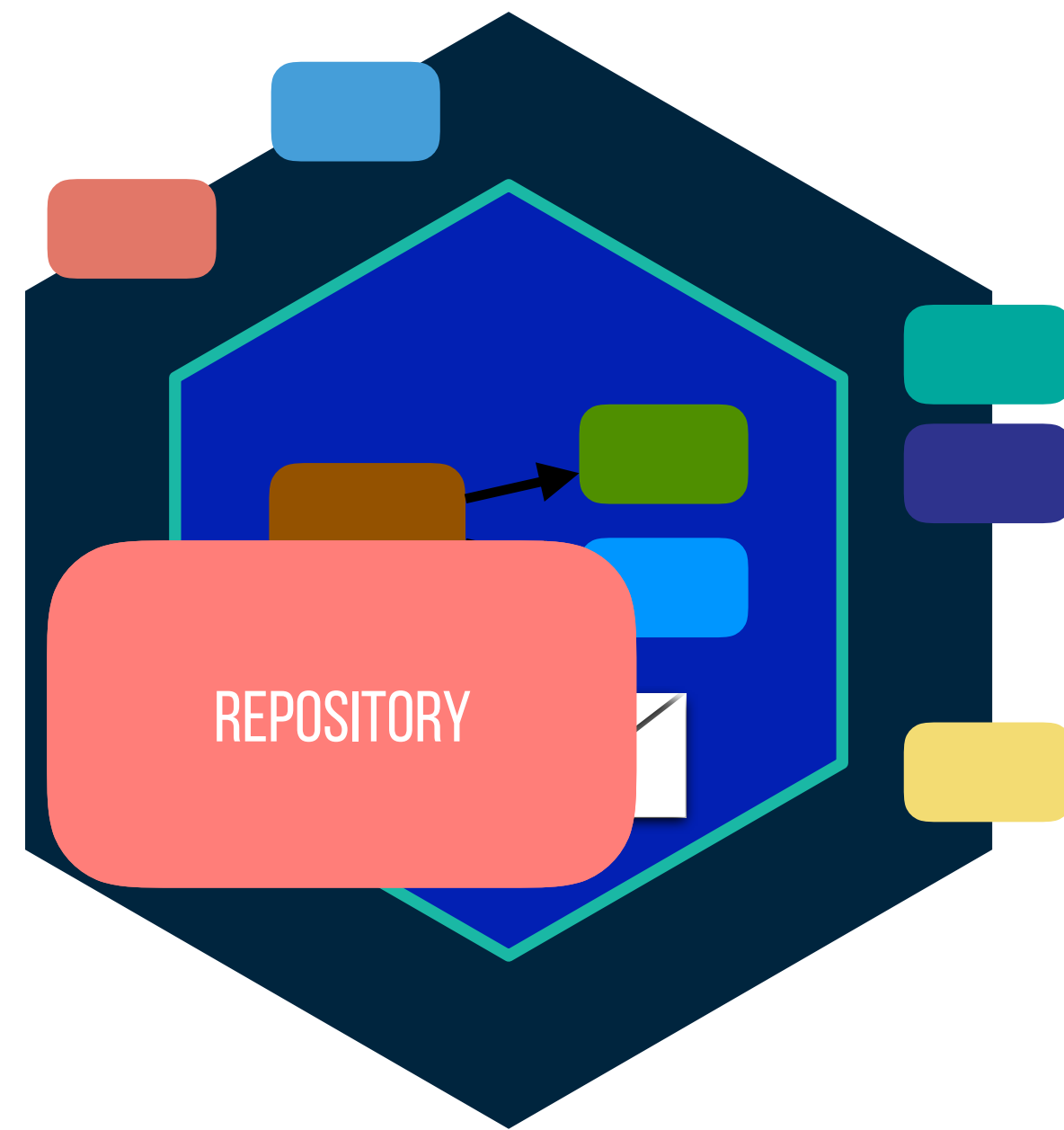
# PUBLISH - SUBSCRIBE



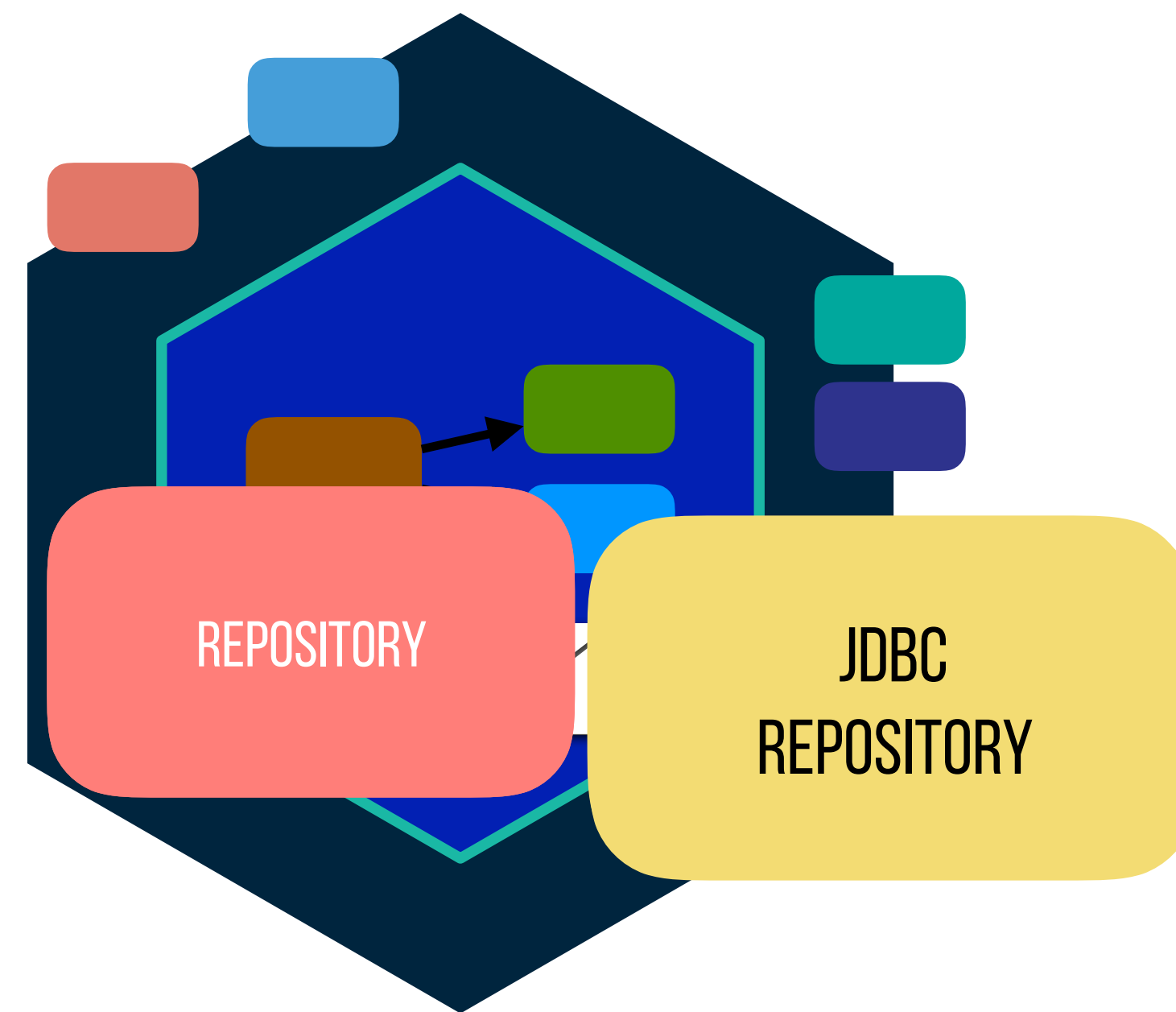
# INTRODUCING PERSISTENCE



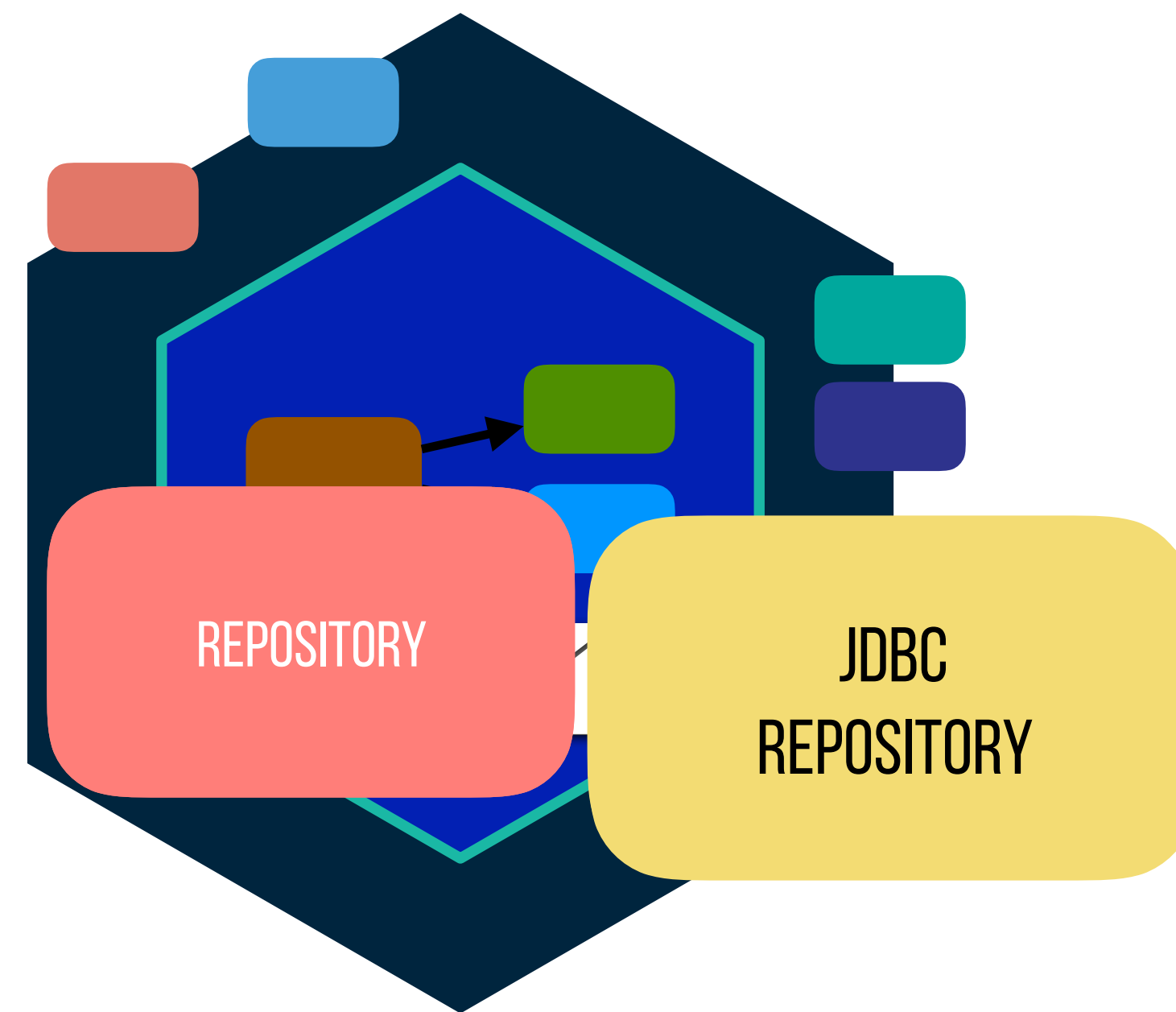
# INTRODUCING PERSISTENCE



# INTRODUCING PERSISTENCE

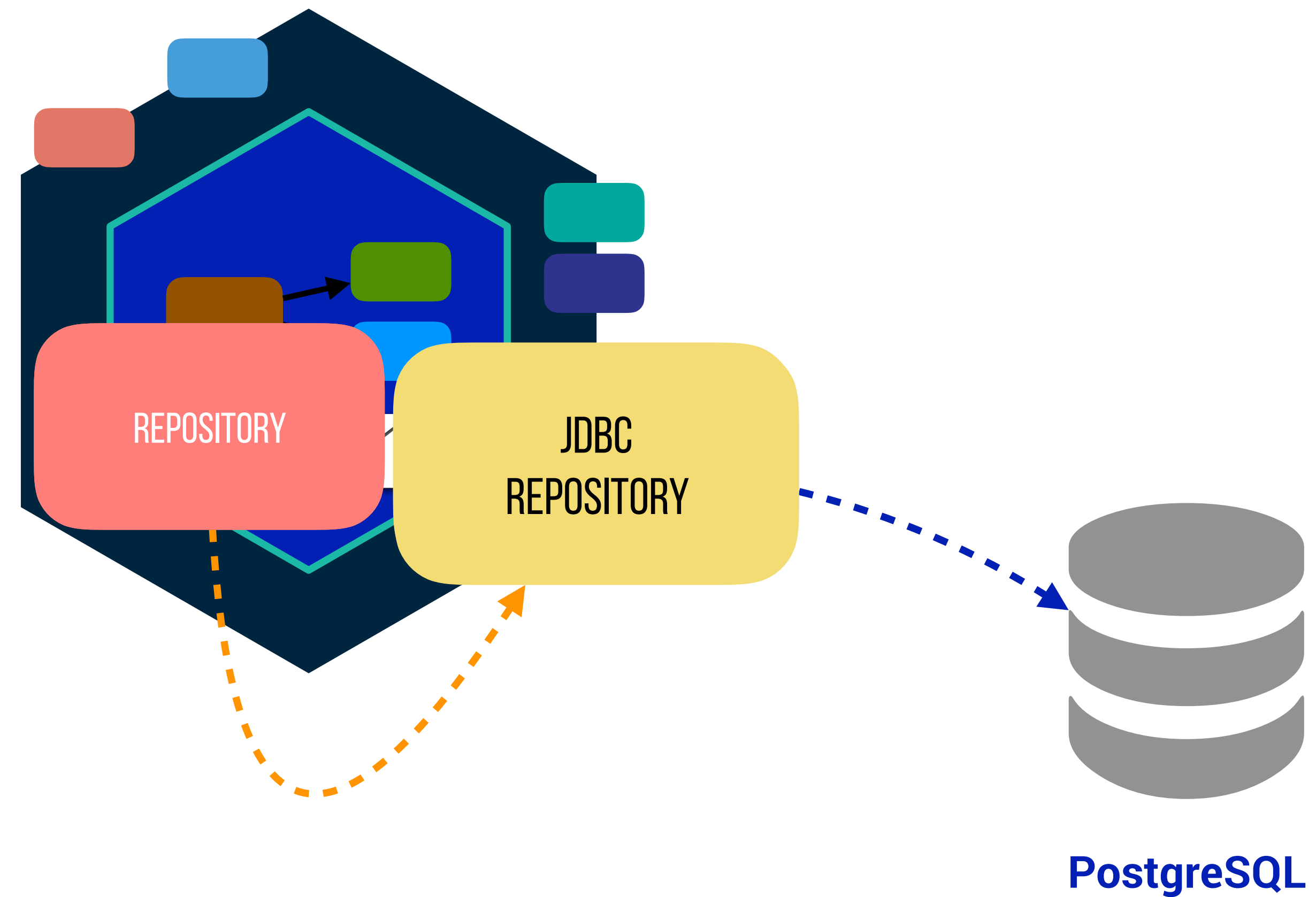


# INTRODUCING PERSISTENCE

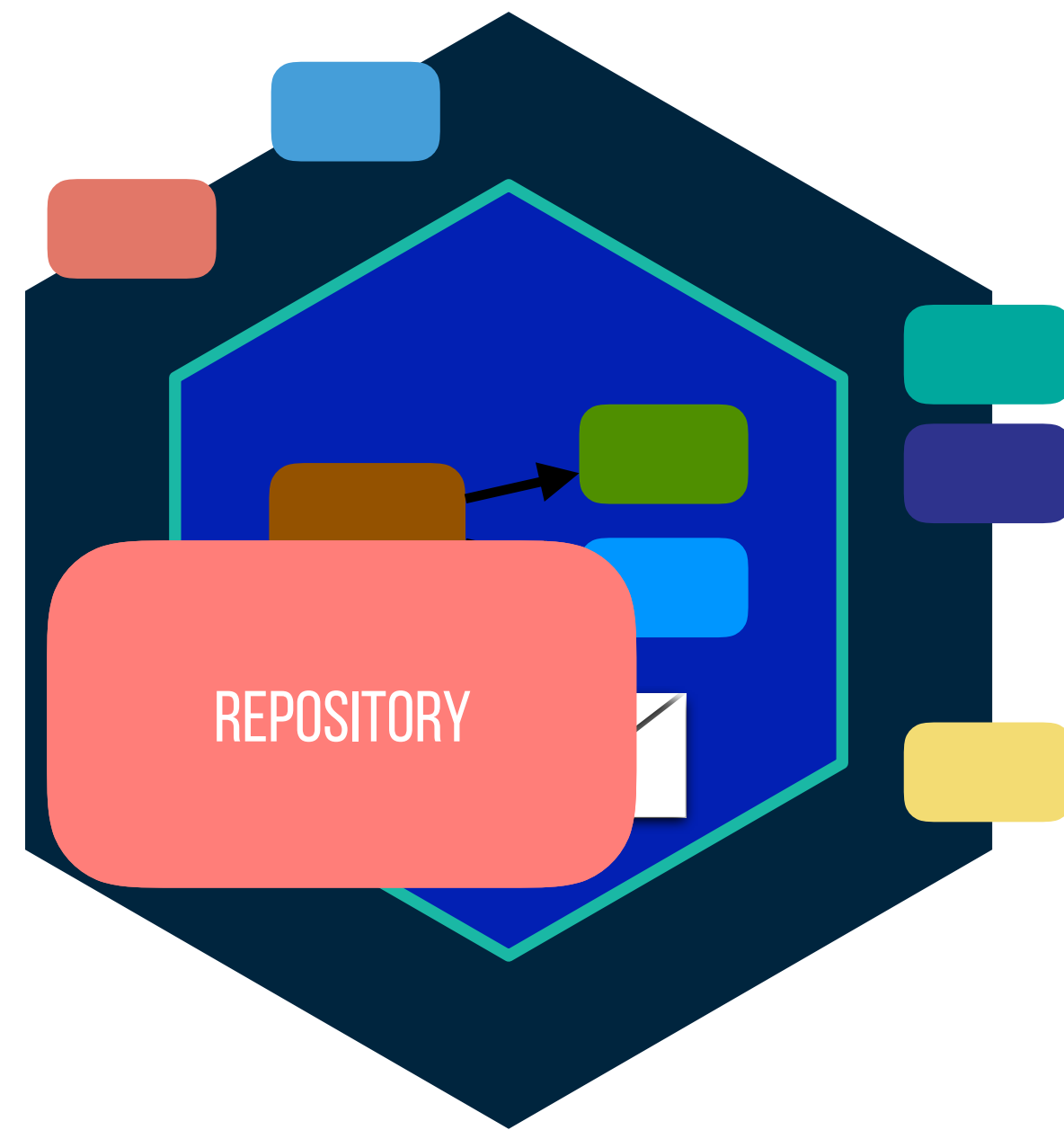


PostgreSQL

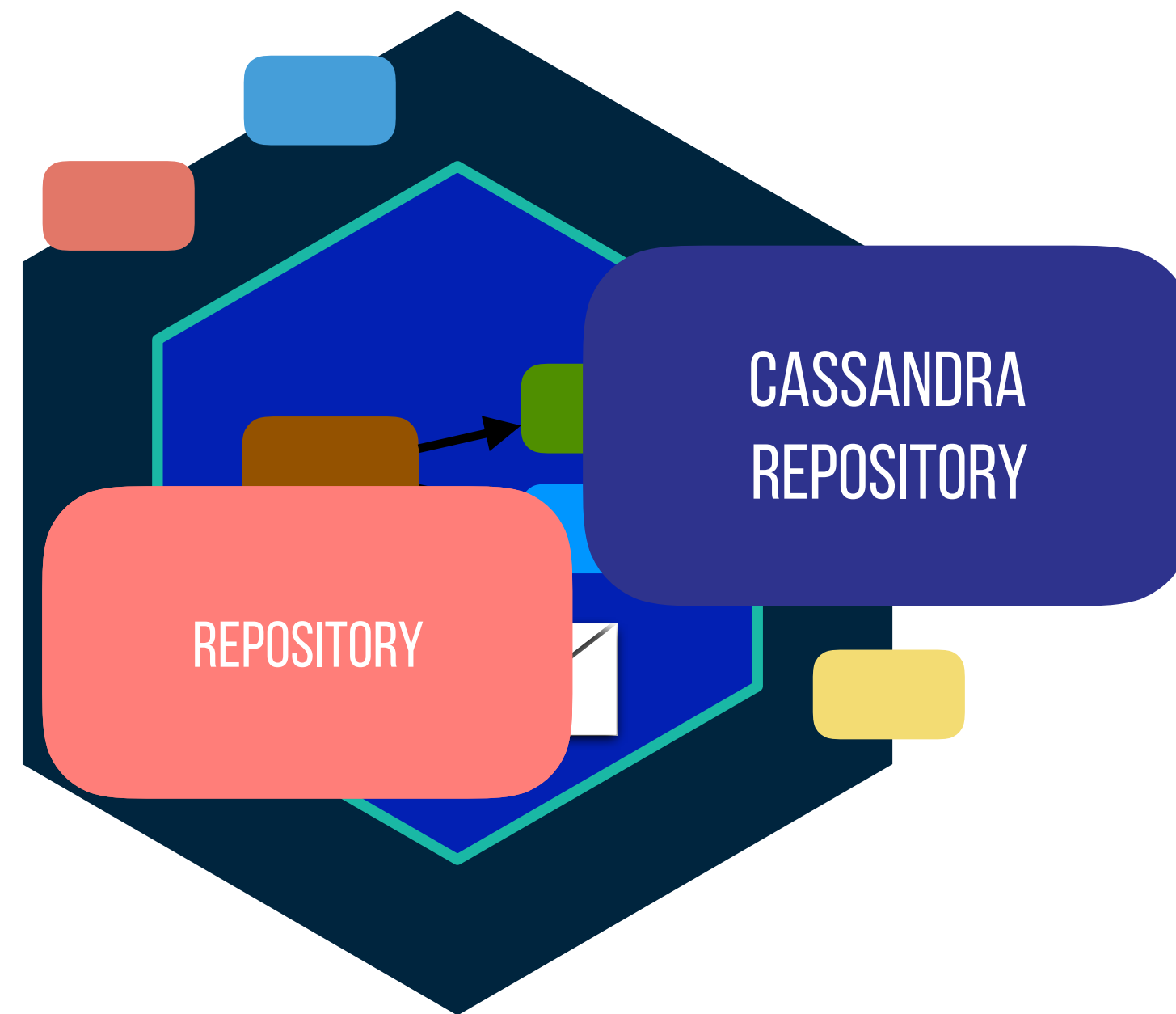
# INTRODUCING PERSISTENCE



# INTRODUCING PERSISTENCE

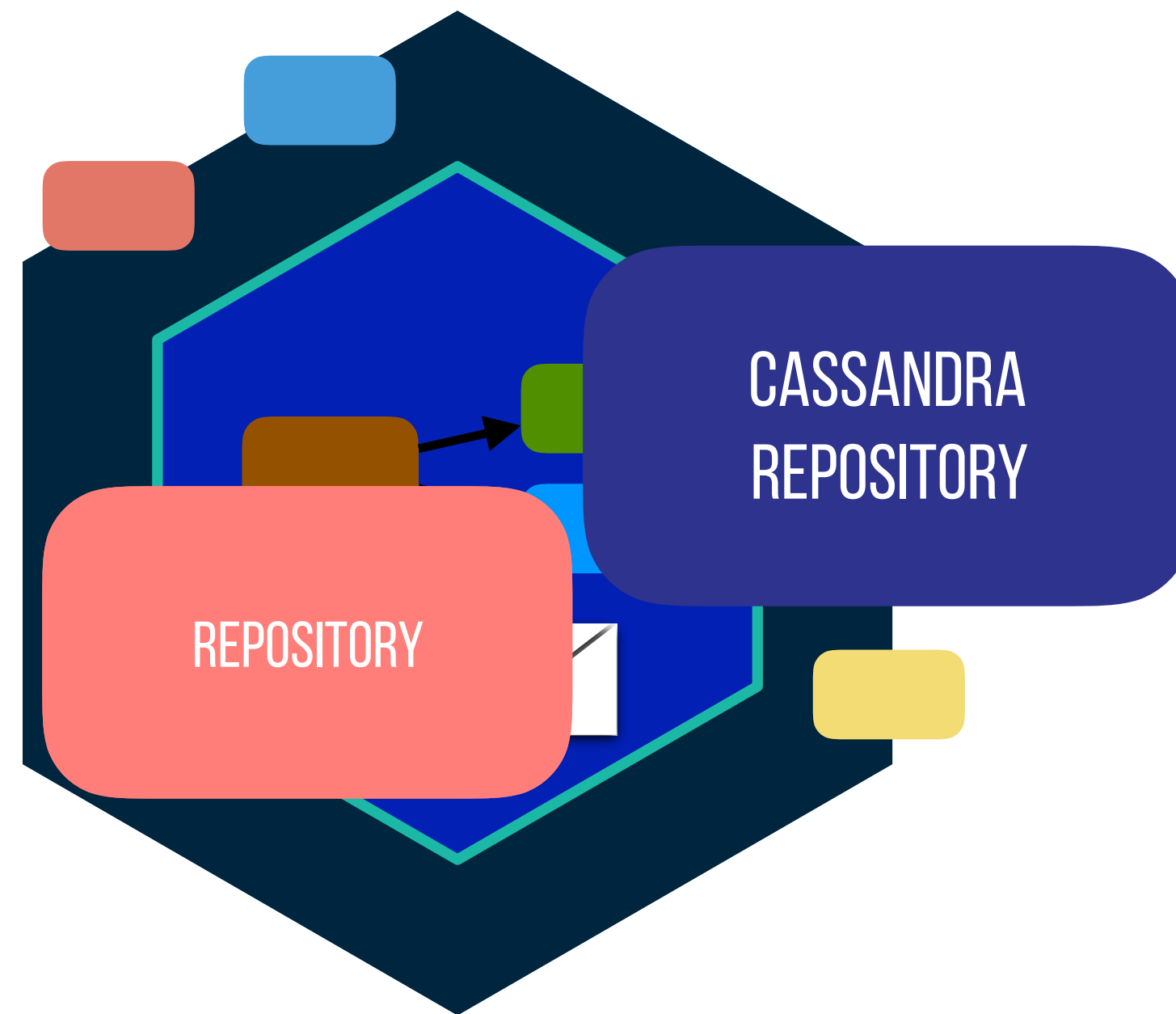


# INTRODUCING PERSISTENCE



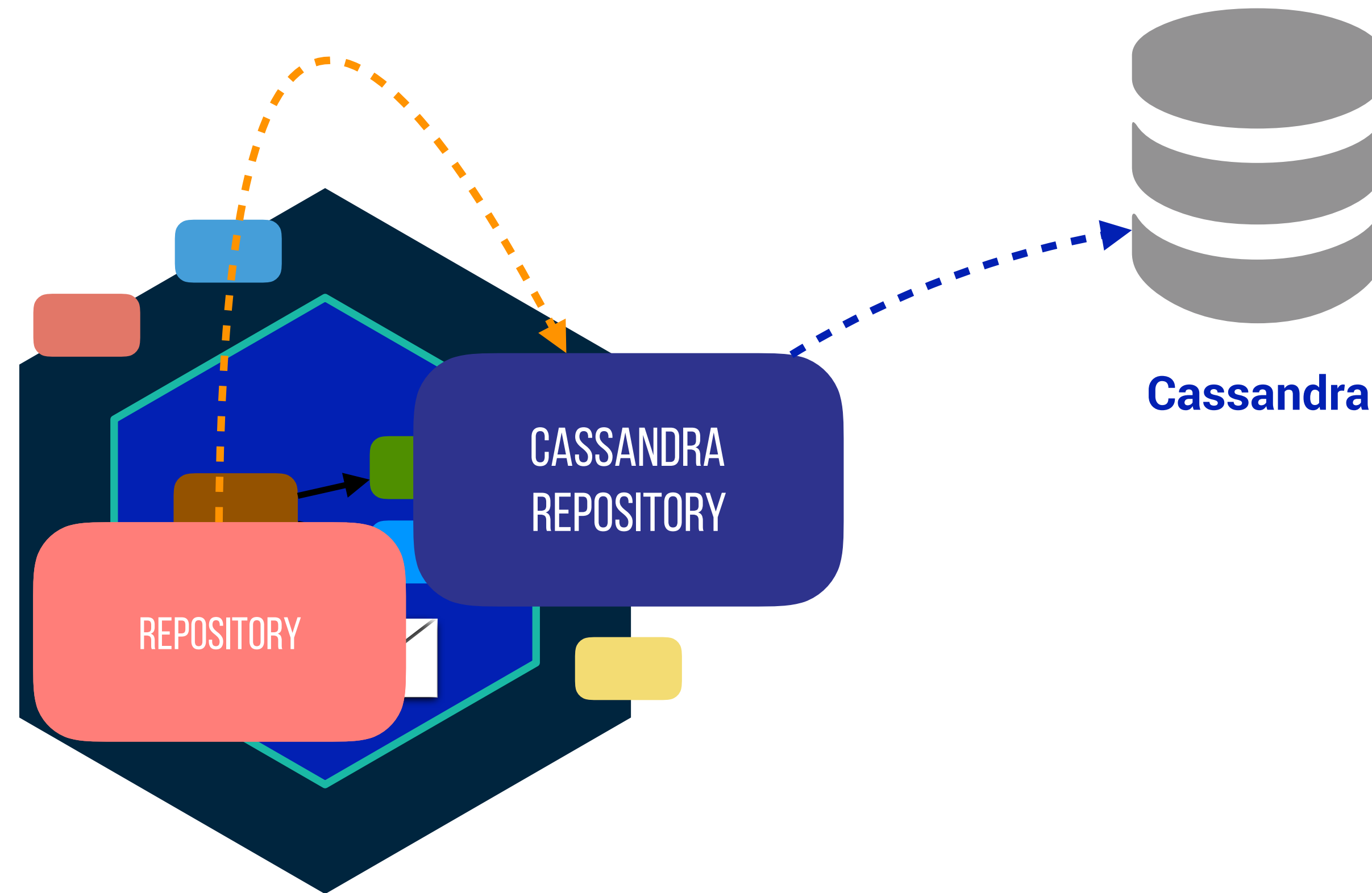


# INTRODUCING PERSISTENCE

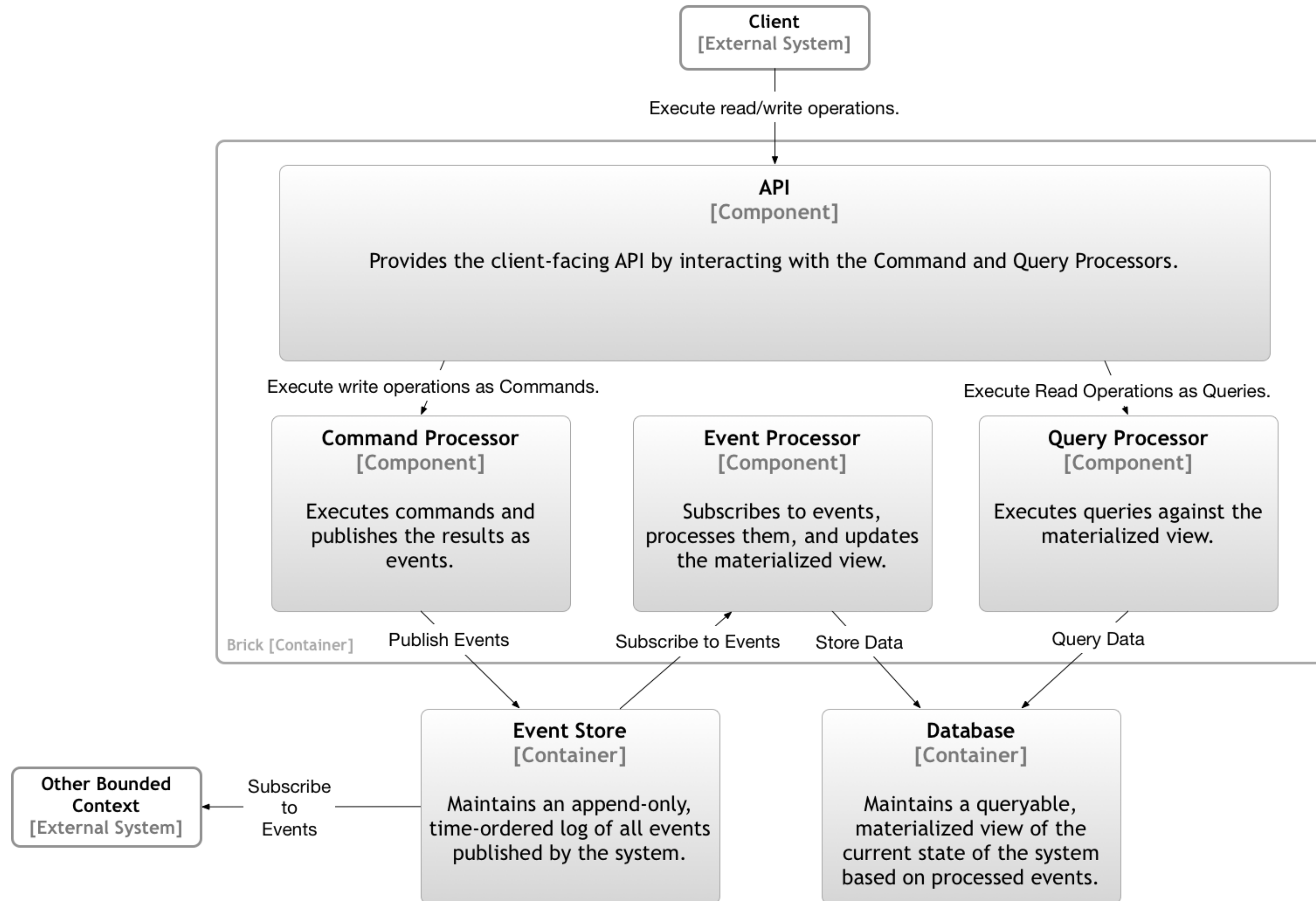


**Cassandra**

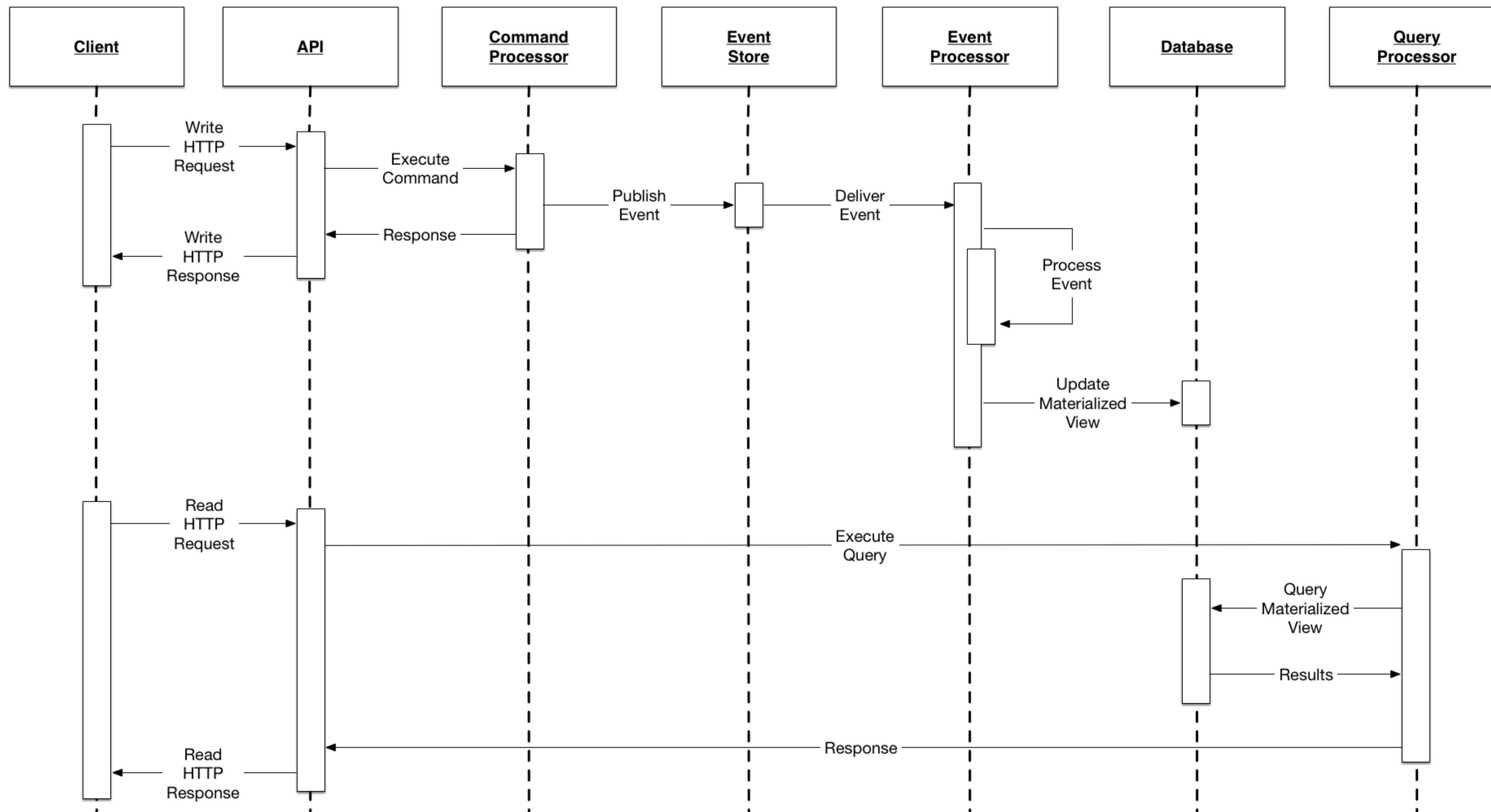
# INTRODUCING PERSISTENCE



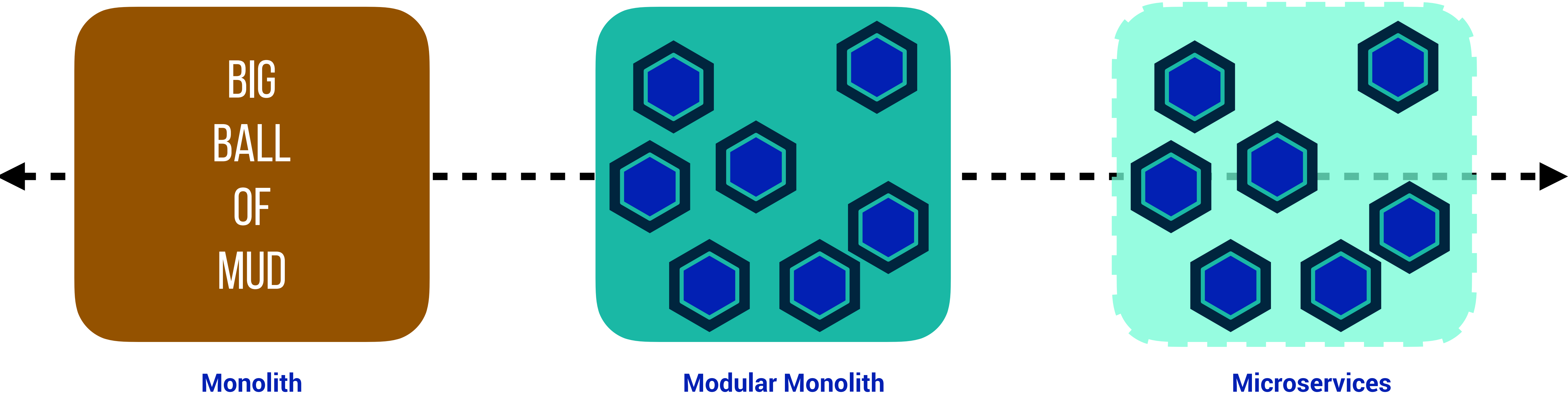
# EVENT SOURCING / CQRS



# EVENT SOURCING / CQRS



# DEPLOYMENT TOPOLOGY SPECTRUM



# MODULES ARE MODULES

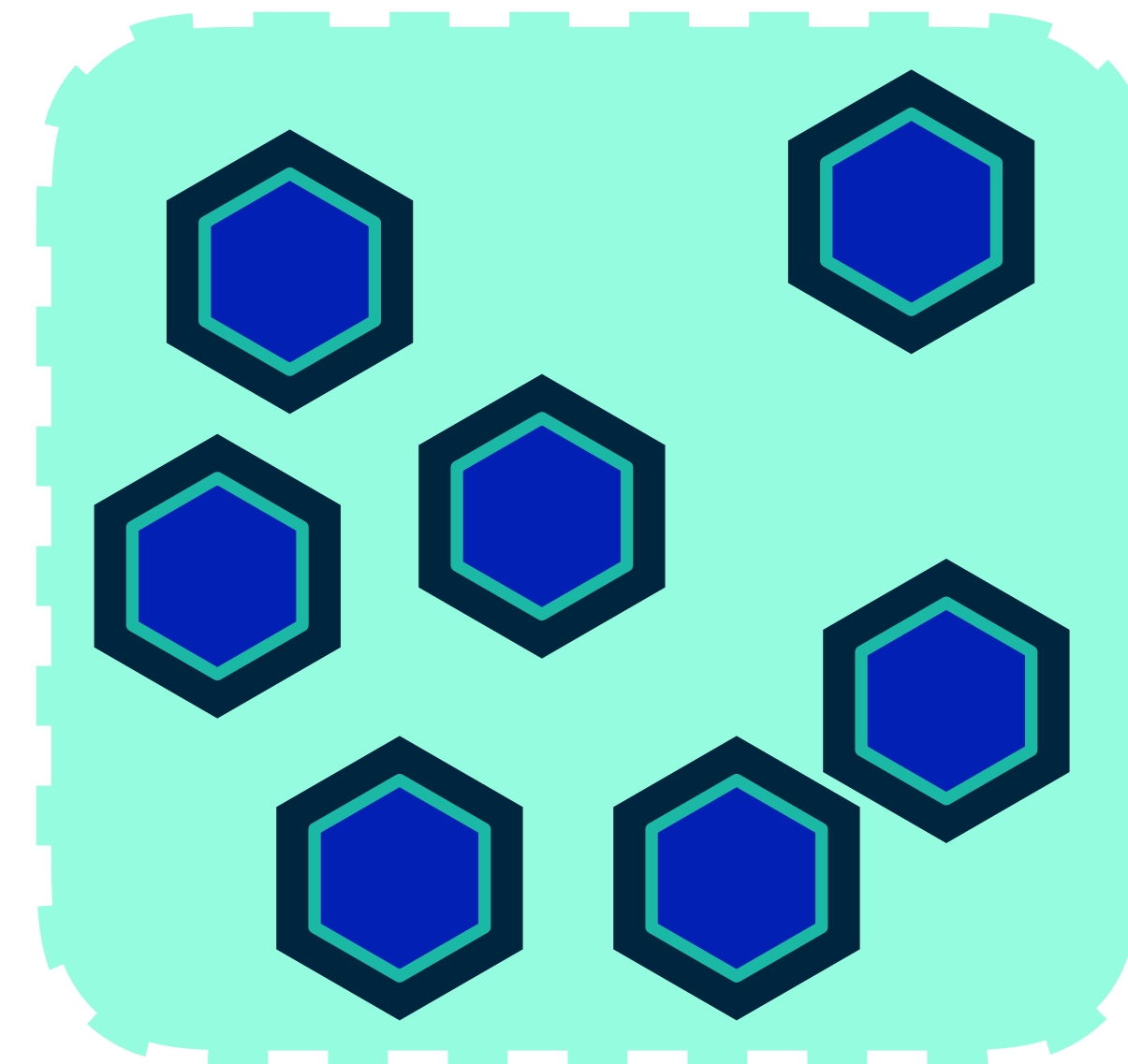
Modular Monolith



- High Cohesion
- Low Coupling
- Business Capability Focus
- Bounded Contexts / Aggregates
- Data Encapsulation
- Substitutable
- Composable



Microservices



- Individually Deployable
- Individually Upgradeable
- Individually Replaceable
- Individually Scalable
- Heterogeneous Tech Stacks

---

The **criteria** for effectively creating modules  
inside of a **monolithic** codebase **translates**  
into effective criteria for microservice  
**modularity.**

---

---

We want to maintain a modular structure  
that **can exist in either** a monolithic or  
microservices architecture.

---