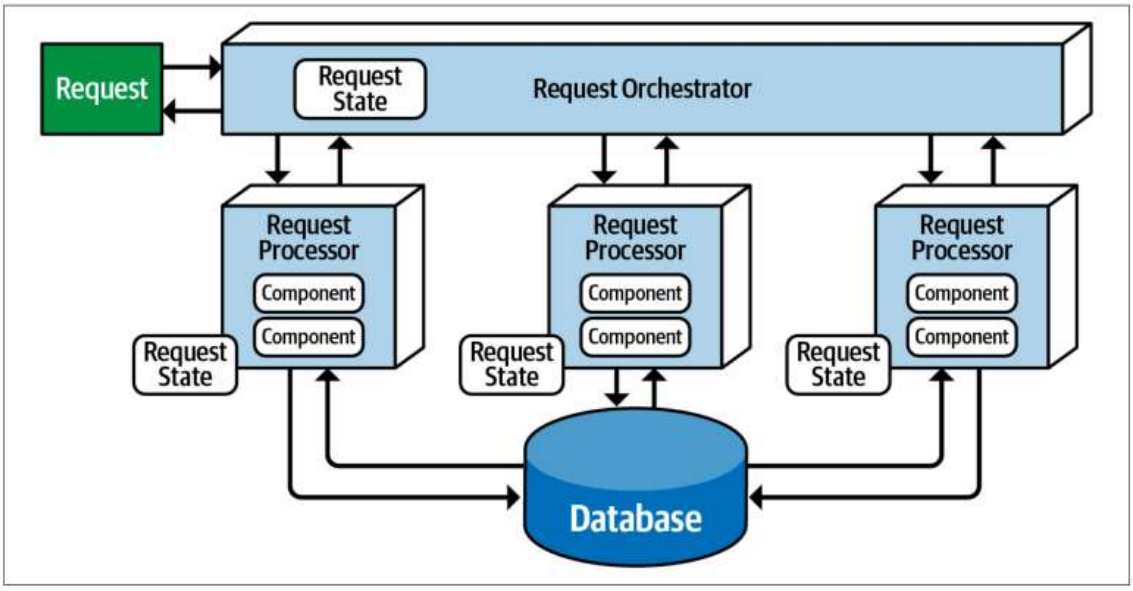
14 - Arquitetura Orientada a Eventos

2025-12-21 11:14 am

- Popular para escalabilidade e alto desempenho
- Processamento assíncrono de eventos, desacoplado.
- Geralmente modelo *request-based:*

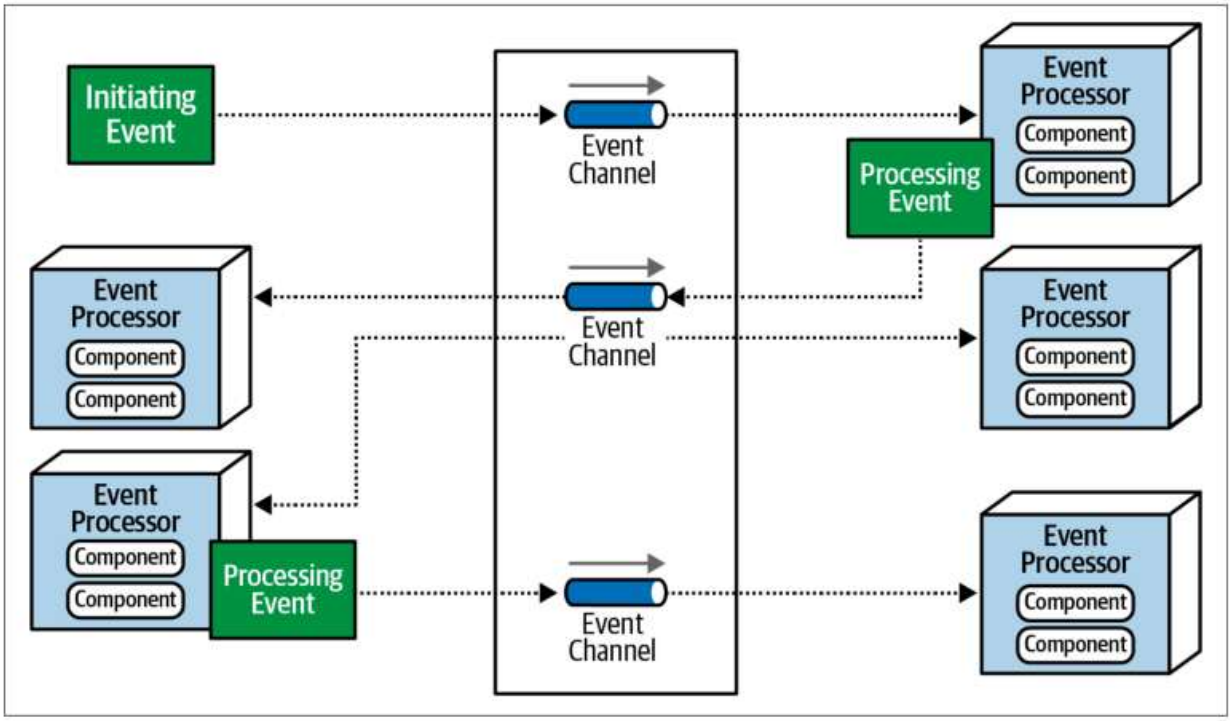


  - *Figure 14-1. Request-based model*
  - Usa um orquestrador de requisições.
    - Direciona as requisições cada uma a um processador de requisições.
    - Processadores lidam com a requisição em si.
- Modelo *event-based:*
  - Reage baseado em algum evento.
  - Não é um requisição em si, mas algo que ocorreu.

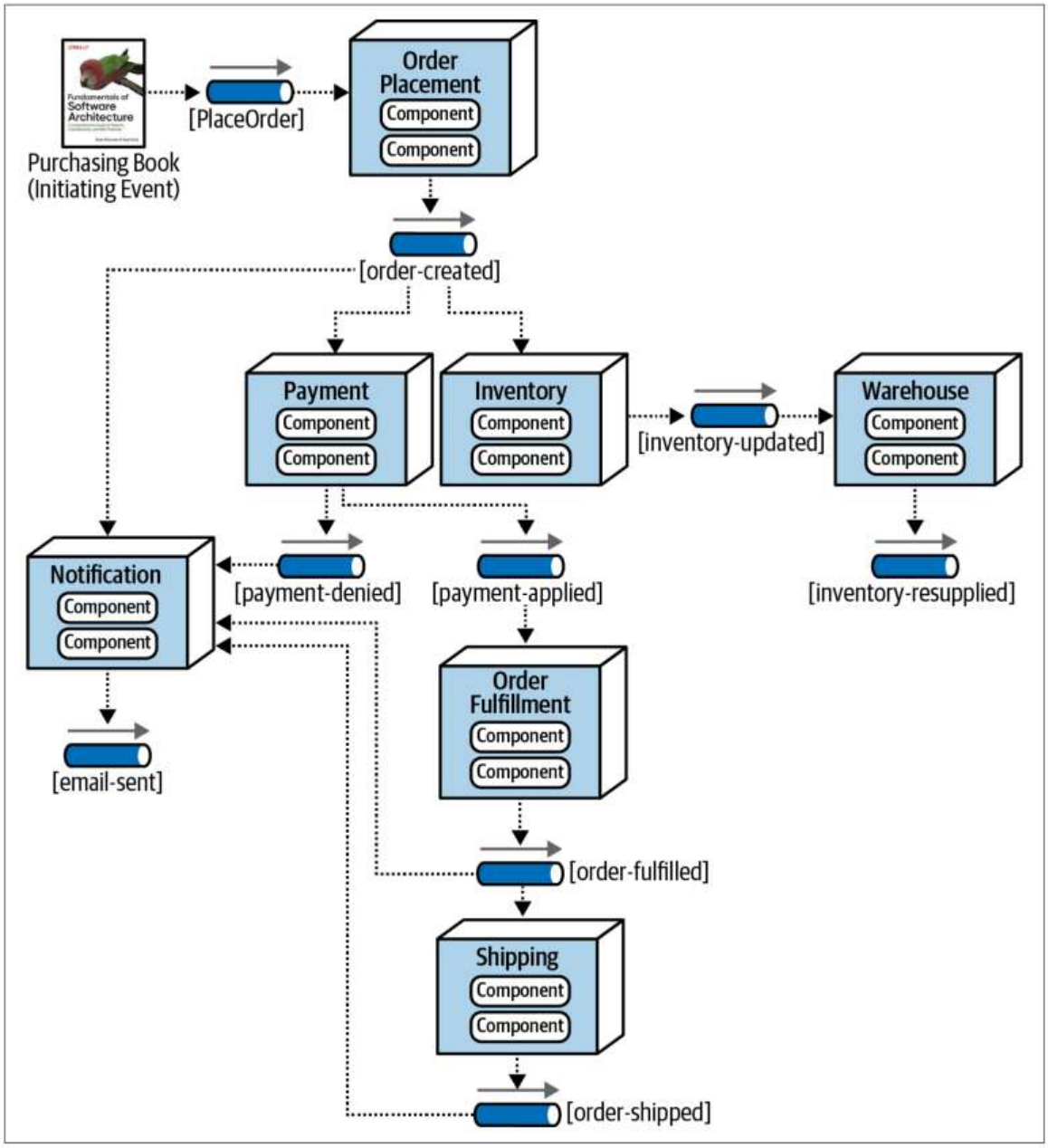
Topologia

- Dois tipos:
  - **Mediator Topology**
    - Quando precisa de controle sobre o workflow do processamento
  - **Broker Topology**
    - Quando precisa de responsividade e controle dinâmico sobre o processamento de eventos.

Broker Topology



- *Figure 14-2. Broker topology*
- Não existe um mediador central.
- Eventos fluem como um rio entre componentes processadores.
- Bom quando processamento é relativamente simples.
- Há quatro componentes principais:
    - **Evento iniciador:**
        - Dá partida ao fluxo, um evento qualquer.
    - **Broker de Eventos:**
        - Tem o canal de eventos
    - **Processador de Evento:**
        - Tem os componentes que processam o evento
    - **Evento de Processamento:**
        - Enviado ao broker para mais processamento se necessário.
- Continua até o evento final não propagar mais nada que desencadeie um evento.
- Broker de eventos costuma ser federado.
- Tópicos usado no modelo *publish-subscribe*
- É boa prática que cada processador publique para todo o sistema o que foi feito, não só para as filas imediatamente relevantes.
    - Facilita extensibilidade caso sejam criados novos processadores.
- Exemplo:



- *Figure 14-4. Example of the broker topology*

- Eventos disparam em cadeia.
- Processadores podem escalar independentemente uns dos outros.
- Problemas:
  - Não há controle geral do fluxo.
  - Lidar com erros é difícil.
  - Um erro pode travar tudo, ou pode dar erro apenas num parte e a outra parte continuar como se não houvesse erro.

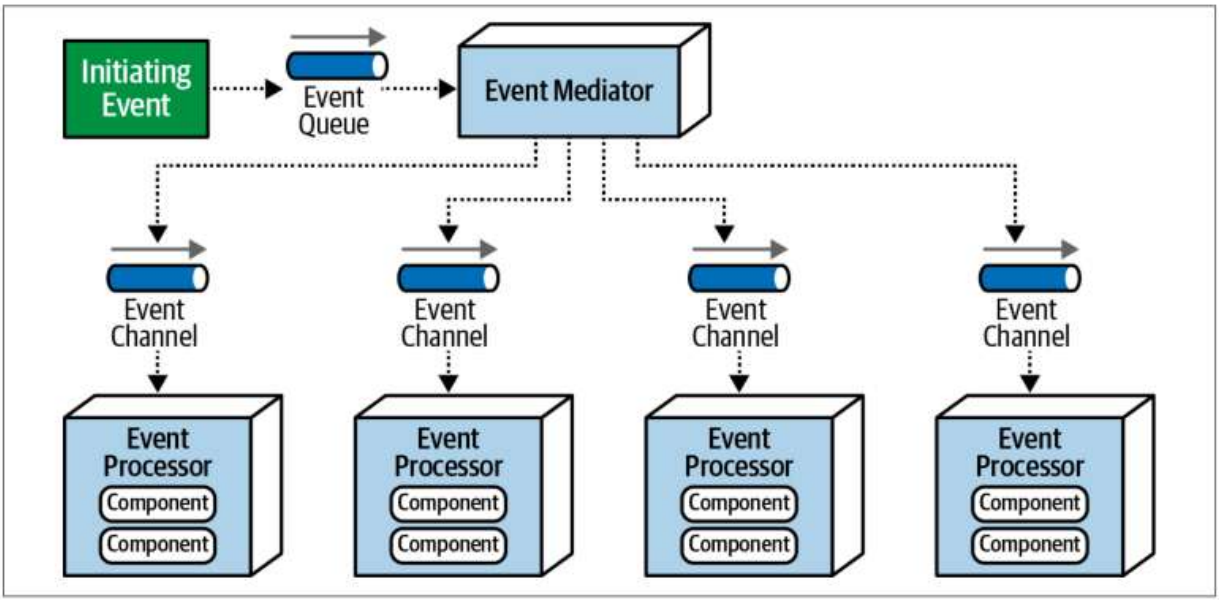
Table 14-1. Trade-offs of the broker topology

Advantages	Disadvantages
Highly decoupled event processors	Workflow control
High scalability	Error handling
High responsiveness	Recoverability
High performance	Restart capabilities
High fault tolerance	Data inconsistency

-

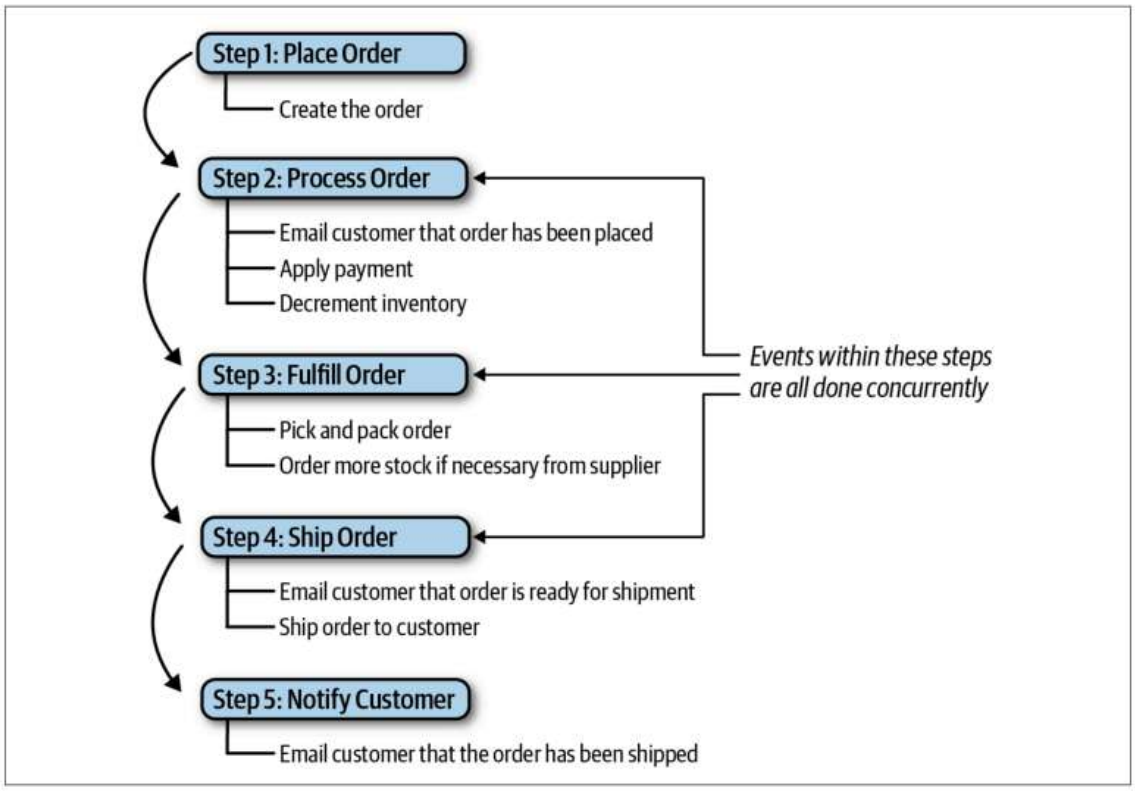
Mediator Topology

- Tenta corrigir problemas da Broker topology.
- Tem um mediador central

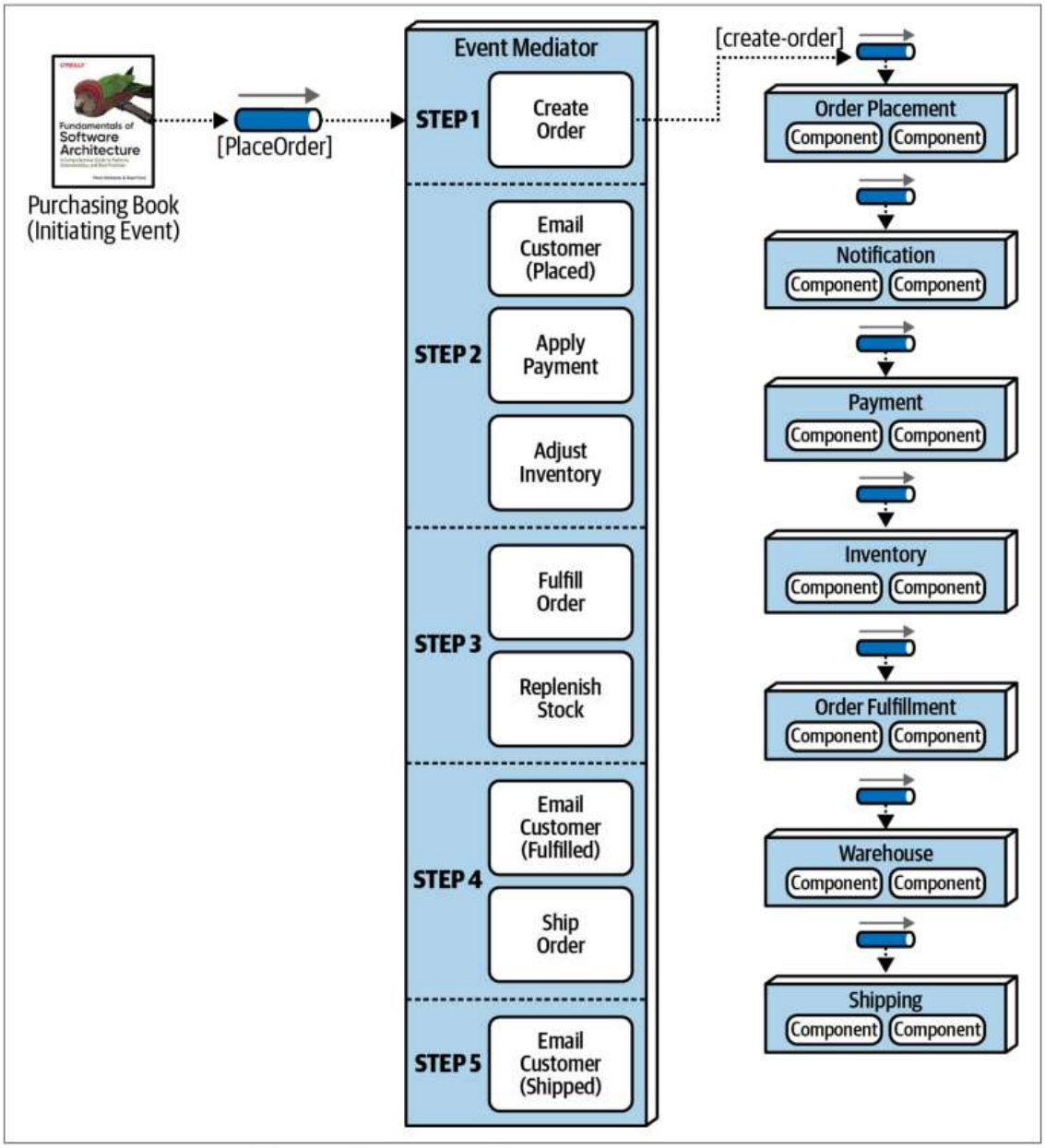


- *Figure 14-5. Mediator topology*
- Inicia com um evento iniciador
    - Mas difere da broker topology, o evento é mandado a uma fila de iniciação de eventos, aceita por um mediador de eventos.
- Mediador direciona cada evento aos canais (geralmente filas) correspondentes.
- Processador escuta a fila, processa e responde ao mediador.
    - Processador não publica pra todo o sistema o que foi feito.
- Geralmente há vários mediadores, agrupando eventos ou por domínio.
    - e.g. um mediador para eventos de clientes.
- Apache Camel, Mule ESB ou Spring Integration são mediadores comuns e simples.

- Se for muito complexo e cheio de condicionais, Apache ODE ou Oracle BPEL podem ser melhores.
    - Baseados em BPEL (Business Process Execution Language), XML-like que descreve os passos.
        - Completo mas complexo, geralmente tem um GUI para ajudar.
- Eventos podem ter human-in-the-loop, nem todo sistema tem suporte a isso
- Pode haver vários mediadores diferentes, que classificam complexidade de evento e redirecionam para outro mediador mais adequado.
- Na topologia de mediador, mediador sabe os passos necessários até o fim:



    - *Figure 14-7. Mediator steps for placing an order*



- *Figure 14-8. Step 1 of the mediator example*
- Mediador espera o sucesso de todos os processadores de um dado passo pra partir para o próximo.
- Mantém estado do evento.
- Pode parar fluxo até que alguma condição ocorra.
- Lida com *comandos* (futuro) mais do que eventos (*passado*)
- É mais complexo e tem pior desempenho que broker topology
  - Mediador em si pode ser um gargalo.
- Acoplamento é maior.

Table 14-2. Trade-offs of the mediator topology

Advantages	Disadvantages
Workflow control	More coupling of event processors
Error handling	Lower scalability
Recoverability	Lower performance
Restart capabilities	Lower fault tolerance
Better data consistency	Modeling complex workflows

Capacidade Assíncrona

- Toda comunicação é assíncrona.
- Responsividade vs Desempenho
  - Responsividade é sobre UX e dar mensagem ao usuário
  - Desempenho é a velocidade do processo em si
  - Usuário precisa esperar o processamento inteiro ser confirmado ou só receber uma mensagem de acknowledgement imediata?
- Maior problema é *error handling

Error Handling

- *Workflow event pattern*
- Delega o erro para um processador de workflows e passa pra próxima mensagem imediatamente

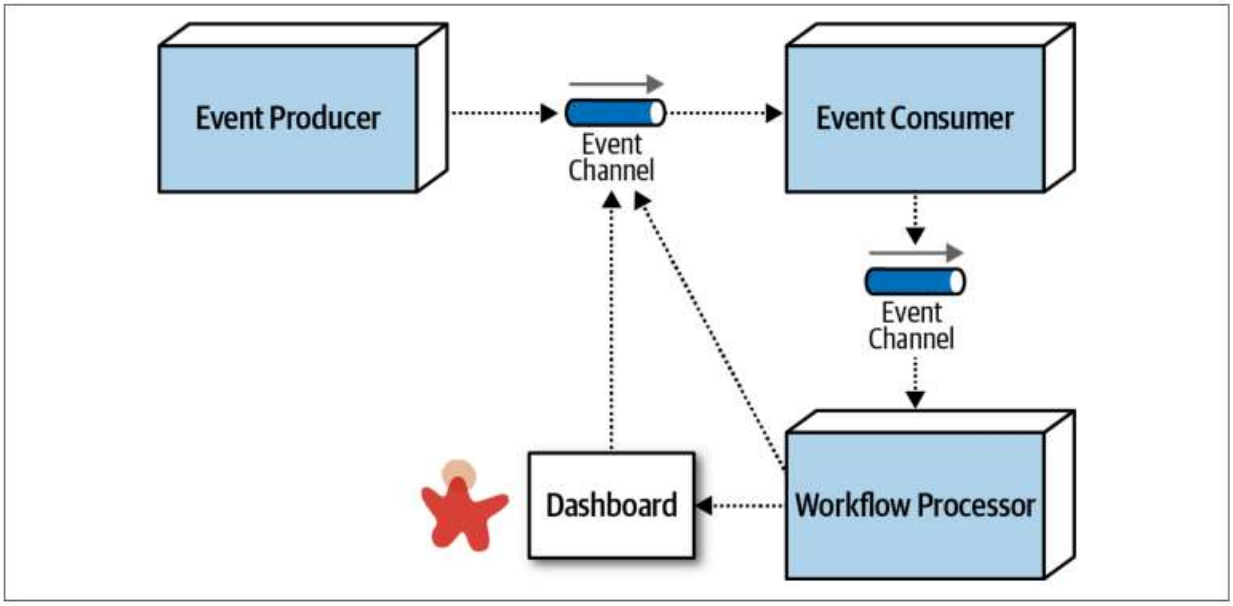


Figure 14-14. Workflow event pattern of reactive architecture

- Workflow processor tenta descobrir e tratar o erro.
  - Manda o resultado de volta pra fila para ser reprocessado, agora sem erro.
  - Se ainda não der certo, pode mandar o erro para um componente (*dashboard*) que aceita intervenção manual, para então ser tratado manualmente e redirecionado à fila.
- Exemplo:

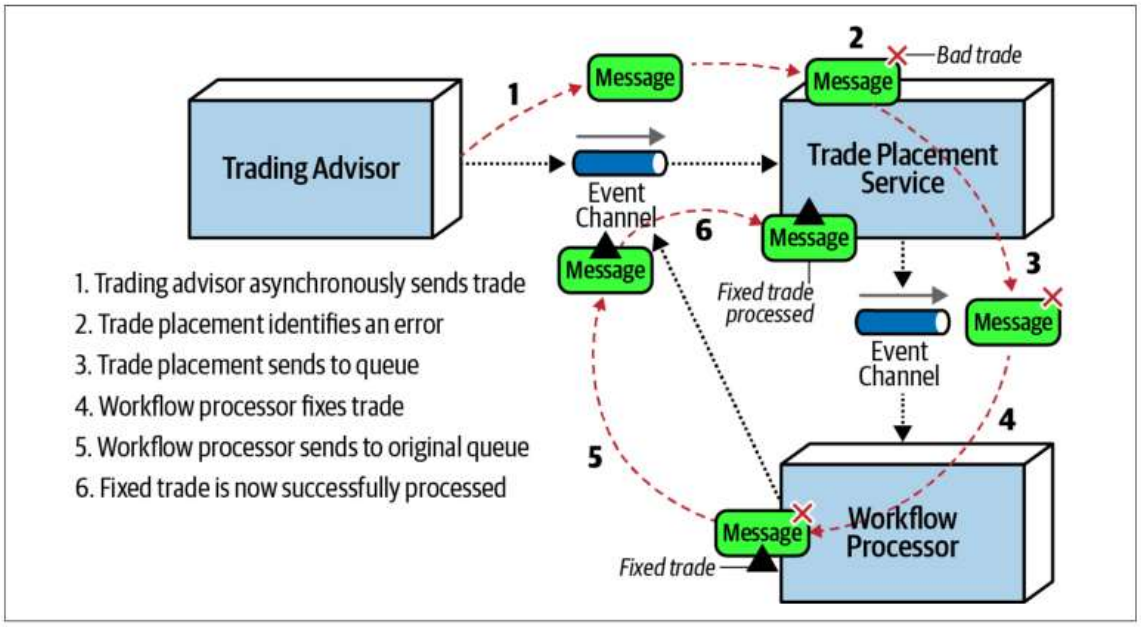


Figure 14-15. Error handling with the workflow event pattern

- Não é impossível manter a ordem original da fila, mas requer uma complexidade a mais de guardar as mensagens que devem aguardar em uma outra fila para manter a ordem até que o erro seja arrumado.

Prevenindo Perda de Dados

- Mensagens assíncronas podem se perder no caminho
- Erros podem ter origem em:
  1. Mensagem nunca chega à fila ou o broker falha
  2. Processador retira uma mensagem da fila mas falha durante processamento da mensagem.
  3. Persistência da mensagem falha.

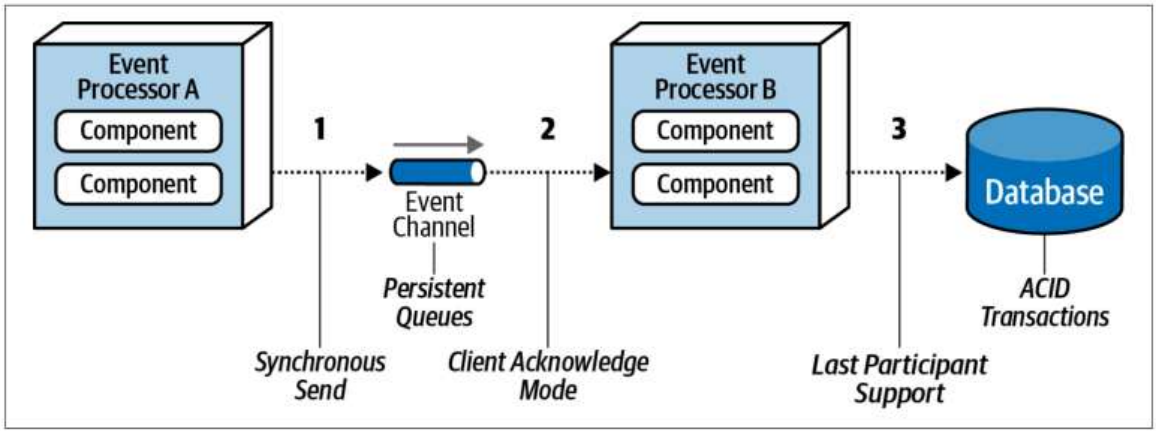


Figure 14-17. Preventing data loss within an event-driven architecture

- **Problema 1:**
  - Evitável com "*synchronous send*"
    - Mensagem é armazenada pelo broker não só em memória, mas em um datastore, que persiste caso o broker falhe.
    - Mensagem só é produzida quando broker garante que já foi persistida.
- **Problema 2:**
  - Evitável com *client ackowledge mode*

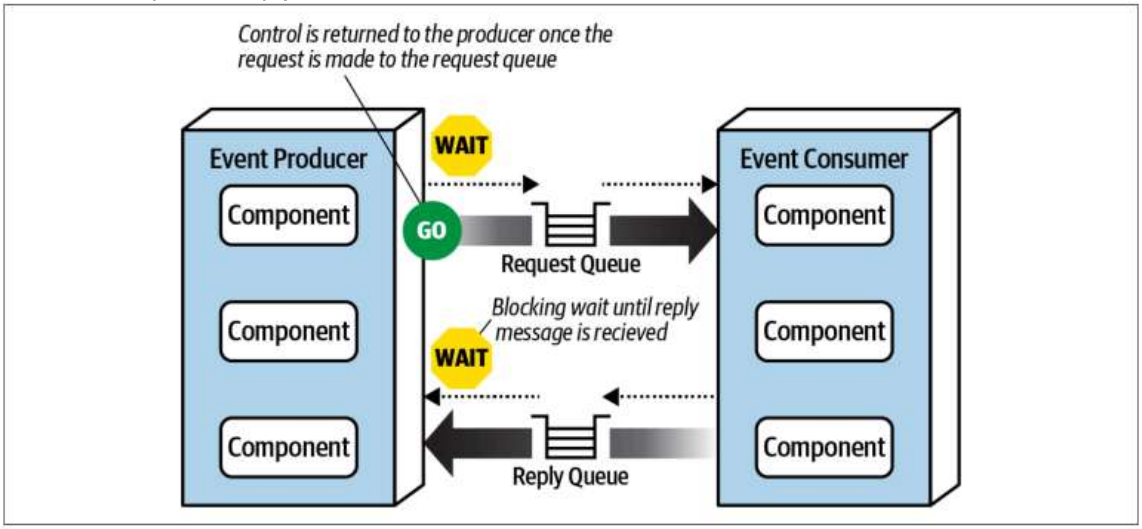
        - Mensagem é preservada na fila e intocável até que o processamento mande um sinal de sucesso.
- **Problema 3:**
    - Evitável com *last participant support (LPS)*
    - Uso de princípio ACID
    - Mensagem só é removida da fila quando há garantia da persistência no banco.

Capacidade de Broadcast

- Publicar eventos sem saber quem os receberá.
    - Pode ter vários subscribers da mesma mensagem.
- Altamente desacoplado.
- Requisito geralmente necessário.

Request-Reply

- E se uma ação depender de outra informação para ser executada?
    - e.g. comprar um livro requer ter um orderId
- Comunicação síncrona necessária.
- Mensageria *request-reply*
    - Duas filas, *request* e *reply*



    - *Figure 14-19. Request-reply message processing*
    - Produtor manda a request messag e espera
        - Consumidor processa e requisição e devolve mensagem para a fila de *reply*.
    - Comum usar um Correlation ID (CID) para isso
        - Produtor do evento guarda um CID de evento e espera até que receba de volta esse mesmo ID do consumidor.
    - Também pode ser usado uma fila de reply temporária:
        - Fila criada para uma requisição específica e deletada em seguida após receber a resposta.

Escolher entre Request-Based e Event-Based

- Request-based melhor para fluxo bem definido e que requer controle de fluxo.
- Event-based melhor para mais responsividade e escala.

Table 14-3. Trade-offs of the event-driven model

Advantages over request-based	Trade-offs
Better response to dynamic user content	Only supports eventual consistency
Better scalability and elasticity	Less control over processing flow
Better agility and change management	Less certainty over outcome of event flow
Better adaptability and extensibility	Difficult to test and debug
Better responsiveness and performance	
Better real-time decision making	
Better reaction to situational awareness	

-

Arquitetura Event-Driven Híbridas

- Arquitetura de eventos pode estar inserida em outra arquitetura para ter as vantagens da arquitetura de eventos.
    - E.g. microsserviços.

Características

Architecture characteristic	Star rating
Partitioning type	Technical
Number of quanta	1 to many
Deployability	★★★
Elasticity	★★★
Evolutionary	★★★★★
Fault tolerance	★★★★★
Modularity	★★★★
Overall cost	★★★
Performance	★★★★★
Reliability	★★★
Scalability	★★★★★
Simplicity	★
Testability	★★

- *Figure 14-22. Event-driven architecture characteristics ratings*
- Altamente paralelizável e escalável.
- Não é simples e é difícil de testar.
    - Request-based é mais fácil de testar que event-based.
    - Árvore de possibilidades pode ser muito complexa.
- Fácil de estender.