Mensageria: da Teoria à Prática em Sistemas Reais

Escalando Sistemas Java em Cenários Críticos

———

Me. Gustavo Pereira Pazzini

Desenvolvedor Java · Fundunesp

São 11h. O Restaurante abriu.

- 600 alunos esperados para o almoço
- Filas começando a se formar
- Os alunos passam suas carteirinhas pelos leitores
- Sistema da catraca funcionando normalmente

- E, de repente...

O que pode ter acontecido?

1. Link de Comunicação

O link com a infraestrutura central ficou indisponível

2. Servidor Central

O servidor responsável pelo processamento está fora do ar

3. Energia Elétrica

Uma queda de energia deixou parte da infraestrutura indisponível

E agora, os alunos passam na catraca?

Sem Mensageria

- Sistema central indisponível
- Catraca não responde
- Alunos não conseguem passar

Com Mensageria

- Evento é armazenado
- Catraca continua funcionando
- Alunos continuam passando

Nosso Roteiro

– **O Problema:** As limitações da arquitetura síncrona

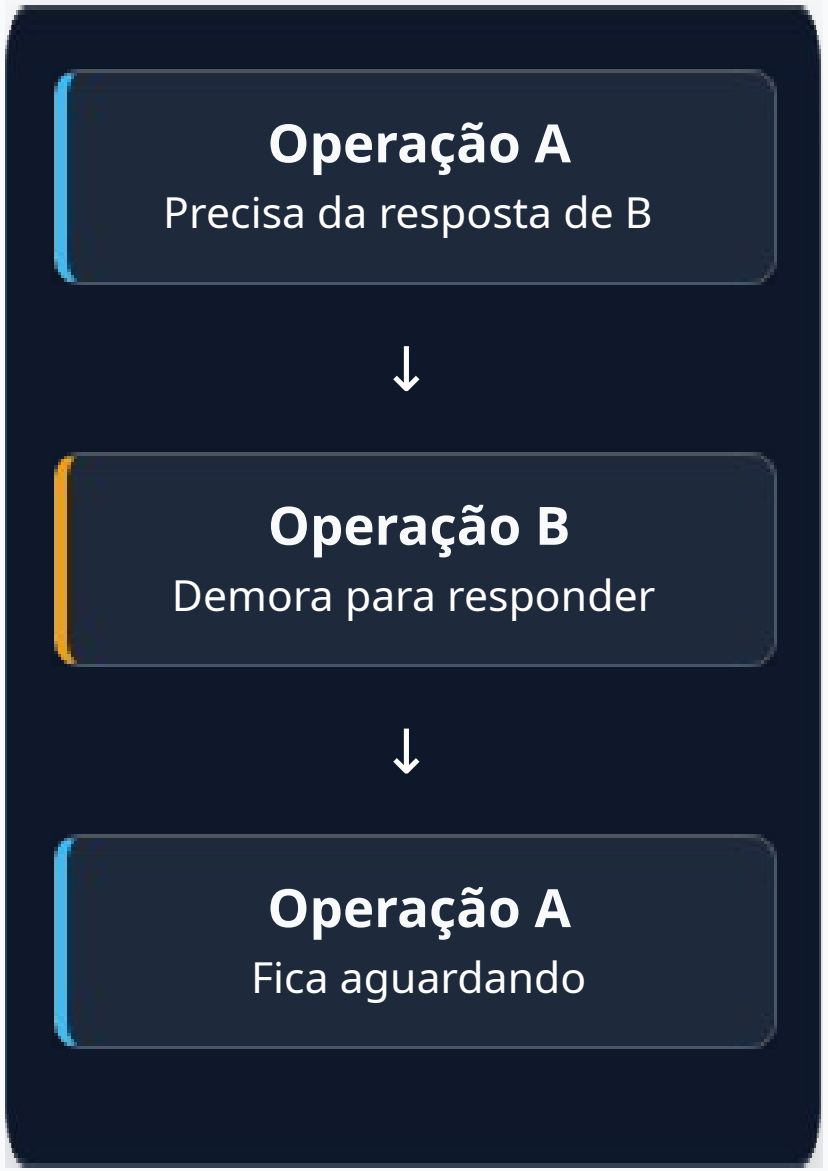
– **A Mentalidade:** Separando o "Agora" do "Depois"

– **A Prática:** Producers, Consumers e Brokers

– **Os Desafios:** Retries, Dead Letters e Mensagens Duplicadas

– **A Decisão:** Quando usar mensageria (e quando não usar)

O Problema



Espera Síncrona

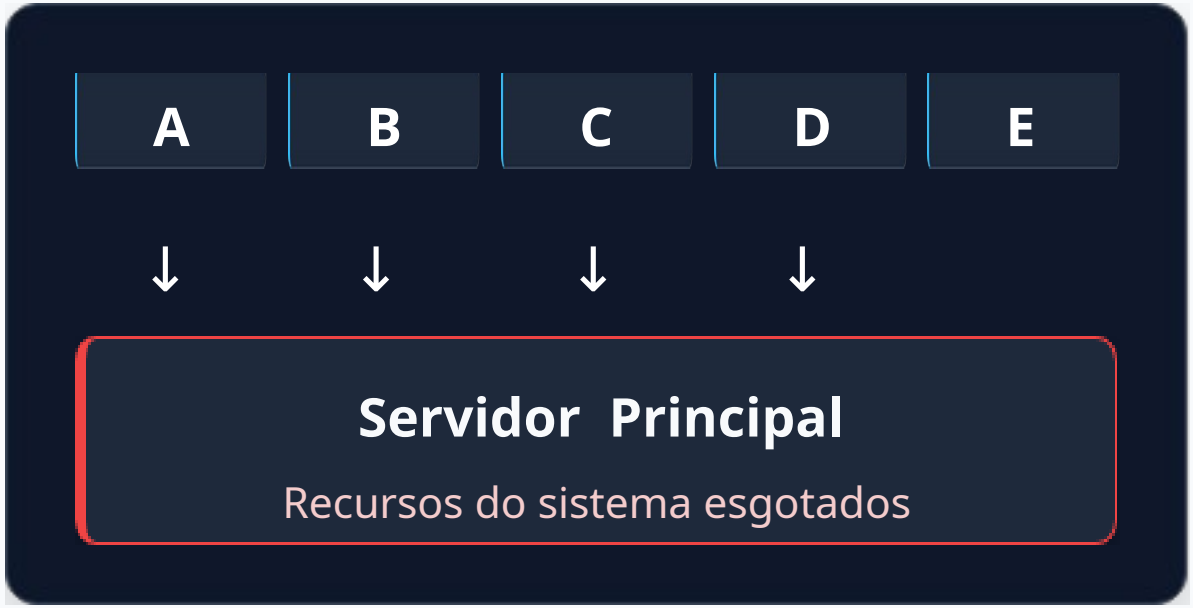
- Dependência direta entre serviços
- Operação bloqueada durante a espera
- O tempo de resposta de B interfere diretamente em A

> “A” não pode avançar enquanto “B” não responder

Efeito Cascata

- A falha pode se propagar pela cadeia
- Serviços dependentes também podem ser afetados
- A falha pode interromper operações subsequentes

> A falha em C afeta a operação do Serviço Principal



Esgotamento de Recursos

- Requisições permanecem ativas enquanto aguardam
- Conexões ficam ocupadas
- Com alta carga, os recursos podem se esgotar

> Esperas simultâneas aumentam o consumo de recursos

Falta de Autonomia

- Mudanças em um serviço podem afetar outros
- A disponibilidade de um serviço depende das dependências
- Resultado: “Monolito Distribuído”

> Os serviços passam a depender uns dos outros

A Mentalidade

"Ao fazer um pedido, o que mais importa é que ele seja realizado. Todo o resto pode acontecer depois, desde que aconteça."

Harry Percival & Bob Gregory (Architecture Patterns with Python)

Filtro de Processos: O "Agora" vs. O "Depois"

O AGORA

Alunos com fome na fila

- Validar se o aluno tem a refeição do dia
- Verificar se a refeição está disponível
- Girar a catraca

O DEPOIS

O sistema resolve no próprio ritmo

- Consolidar a refeição no Sistema Central
- Enviar notificações por e-mail
- Atualizar o dashboard da Nutricionista

A Armadilha do Síncrono

- **Catraca:** Lê o cartão e aguarda...
- **API Central & Banco:** *Incomunicáveis*
- **Catraca:** Aguarda...

> Após alguns minutos sem comunicação, é ativado o plano B

Plano B...

A Libertação: Cada coisa no seu tempo

O Fato

A refeição é validada localmente e a catraca gira

A Memória

O fato é registrado e guardado até ser enviado ao consumidor

O Trabalho

O consumidor lê a informação e processa no ritmo que conseguir

A Prática

Mas antes, do que é feita a Mensageria?

- Producer (Produtor): Sistema que gera a informação
- Message (Mensagem): Um pacote de dados que transporta uma informação
- Broker (Intermediário): Responsável por receber e encaminhar mensagens
- Consumer (Consumidor): Sistema que recebe e processa a mensagem
- ACK (Acknowledge): Confirma que a mensagem foi processada com sucesso

Principais Brokers:

Como escolher um Broker?

- **Modelo:** Queue, Topic / Pub-Sub
- **Escalabilidade:** Volume e taxa de mensagens
- **Integração:** Protocolos e suporte ao Java

Comparativo entre Brokers

Kafka

- Tópicos e partições
- Milhões msg/s*
- Eventos e grandes volumes de dados

RabbitMQ

- Filas e exchanges
- Centenas de milhares msg/s*
- Tarefas e integração entre sistemas

Artemis

- Tópicos e filas
- Milhões msg/s*
- Aplicações Java/JMS e Jakarta EE

O Fluxo da Mensagem

O fluxo assíncrono isola a catraca de qualquer instabilidade

```
rabbitTemplate.convertAndSend(
    "ru.exchange", // Exchange
    "acessos", // Routing Key
    mensagem // Message
);
```

Producer

- Gera a informação
- Publica uma mensagem
- Define a Routing Key
- Não conhece o Consumer

```
@Bean public DirectExchange
exchange() {
    // Cria a Exchange que recebe e
    roteia mensagens

    return new
    DirectExchange("ru.exchange");
}
```

Exchange

- Recebe mensagens
- Analisa a Routing Key
- Roteia as mensagens
- Encaminha para as Queues

```
@Bean public Queue filaDeAcessos() {
    // Cria a fila onde as mensagens
    aguardam o Consumer
    // true = fila durável

    return new
    Queue("ru.acessos.queue", true);
}
```

Queues

- Armazena mensagens
- Aguarda o Consumer
- Retêm mensagens até o processamento
- Pode ter vários Consumers

```
@Bean public Binding binding( Queue
filaDeAcessos, DirectExchange
exchange) {

    return BindingBuilder

    // Conecta a Queue à
    Exchange .bind(filaDeAcessos)
    .to(exchange)

    // Mensagens com essa Routing Key
    vão para a Queue
    .with("acessos");
}
```

Binding

- Conecta a Exchange na Queue
- Define uma regra de roteamento
- Utiliza uma Binding Key
- Determina mensagens chegam à Queue

```
// Dados que serão transportados
Acesso mensagem = new
Acesso( usuarioId, refeicaoId );
```

Message

- Transporta a informação
- Possui Body e metadados
- É enviada pelo Producer
- É processada pelo Consumer

```
@RabbitListener(queues =
"ru.acessos.queue")
public void consumir(Acesso mensagem)
{
    // Processa a mensagem recebida
    processar(mensagem);
}
```

Consumer

- Recebe mensagens
- Processa a informação
- Trabalha no seu próprio ritmo
- Envia o ACK

```
// Confirma que a mensagem foi processada

channel.basicAck(tag, false);
```

ACK

- Confirma o processamento
- Informa o RabbitMQ do sucesso
- Permite remover a mensagem
- Sem ACK, pode ocorrer reentrega

Os Desafios

Os principais desafios da Mensageria

Monitoramento

Como saber se o sistema está saudável?

Duplicação

E se a mesma mensagem for processada duas vezes?

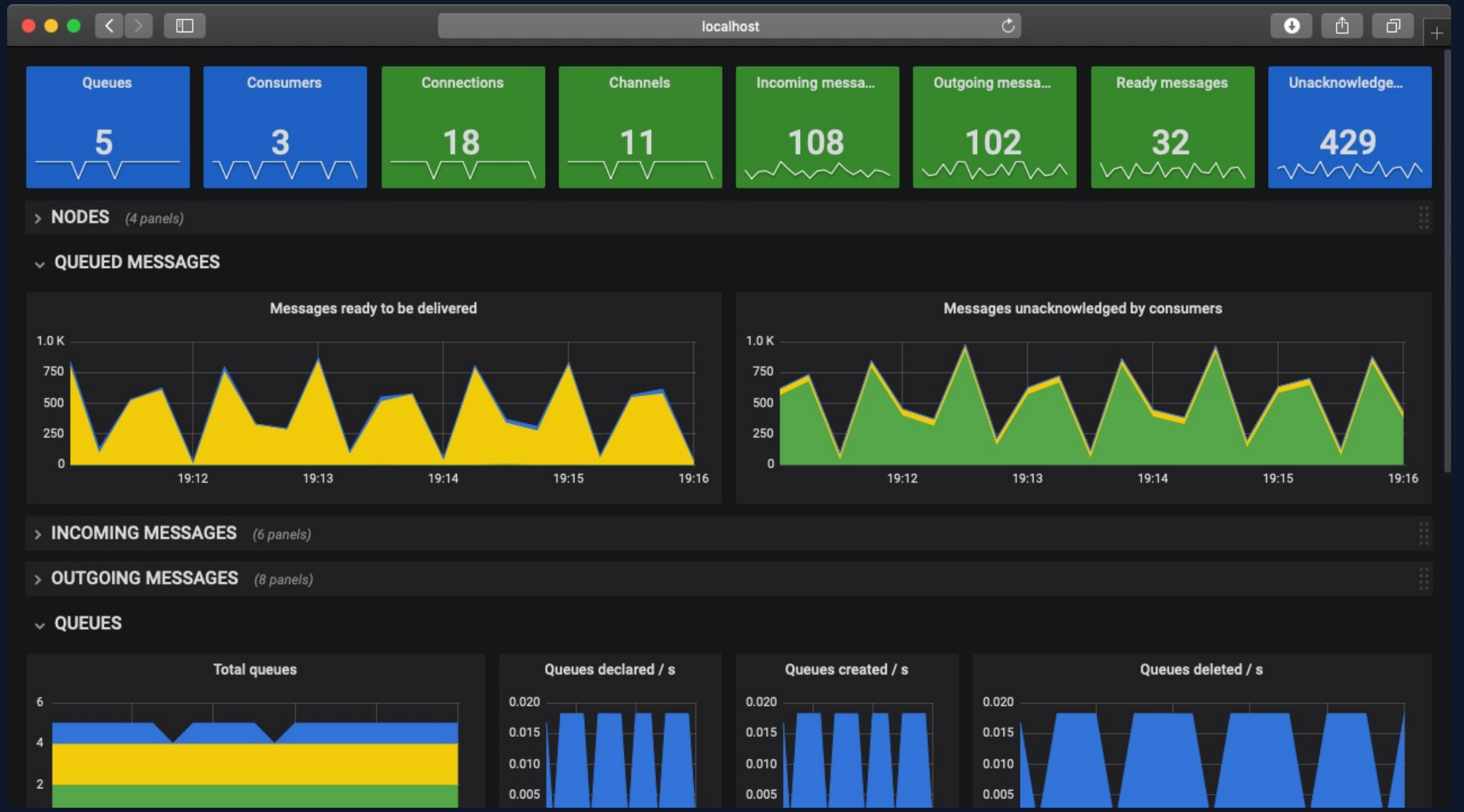
DQL

O que fazer quando uma mensagem não pode ser processada?

Monitoramento

- **A fila é finita**
- **Problema de vazão**
- **Produtor mais rápido que consumidor**
- **Acúmulo de mensagens**
- **Limitação física: memória, disco, CPU e rede**

Monitoramento: Prometheus e Grafana



Duplicação

- **Falha antes do servidor enviar o ACK**
- **A mensagem pode ser reenviada**
- **O consumidor pode processar a mesma mensagem mais de uma vez**

- **Cobrança duas vezes, por exemplo Mensageria não significa necessariamente "processar uma única vez"**

Duplicação: Idempotência

- **Processar a mesma mensagem várias vezes sem causar efeitos duplicados**

- **Identificar mensagens já processadas**

- **Usar um ID único da mensagem**

- **Registrar o processamento antes de aceitar novamente**

Dead Letter Queue (DLQ)

- **Fila de mensagens não entregues (DLQ)**
- **Mensagens que não conseguiram ser processadas**
- **Evita bloquear a fila principal**
- **Permite analisar o problema posteriormente**
- **Possibilita reprocessamento controlado**

Dead Letter Queue (DLQ): Antídoto

- **Identificar a causa do erro**
- **Corrigir o problema antes do reenvio**
- **Reprocessamento controlado**
- **Evitar duplicação**
- **Monitorar a DLQ**

A Decisão

Eu preciso desacoplar essa comunicação?

Comunicação Síncrona

- **Preciso da resposta agora**
- **Resultado imediato**
- **Fluxo simples**

(-) Menor complexidade

X

Mensageria

- **Posso processar depois**
- **Processamento assíncrono**
- **Maior desacoplamento**

(+) Maior complexidade

3 Sinais de que você precisa de filas

- **Processamento assíncrono:** A resposta não precisa acontecer imediatamente
- **Alto volume ou picos:** Absorver variações de carga sem sobrecarregar o consumidor
- **Desacoplamento entre serviços:** Produtor e consumidor operam de forma independente

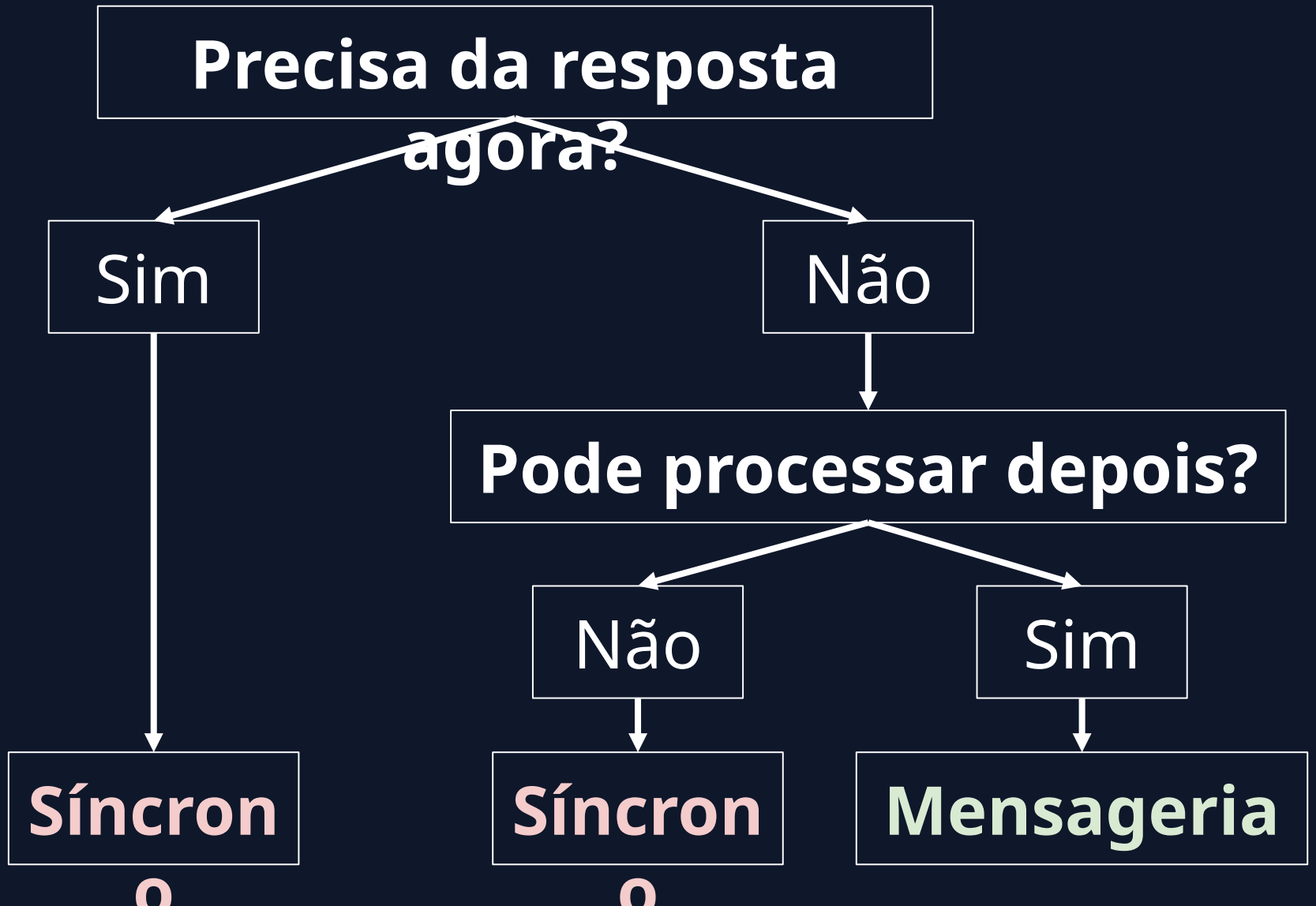
Quando esperar pela resposta deixou de ser necessário

O custo da complexidade

- ✗ A operação exige resposta imediata
- ✗ O fluxo é simples e direto
- ✗ A consistência imediata é necessária (Ex: Saldo em aplicativo)
- ✗ O volume não justifica a infraestrutura extra

Se a comunicação precisa ser imediata, o síncrono pode ser a melhor escolha

Diagrama de decisão



Para lembrar Amanhã

Mensageria não é sobre filas. É sobre decidir:

O que precisa acontecer agora?

O que pode acontecer depois?

E o que fazemos quando algo dá errado?

Obrigado!

Dúvidas?

Gustavo Pazzini

Mais de 10 anos de experiência na área de TI.

Programador na Fundunesp, desenvolvendo projetos para a Unesp.

Mestre e Bacharel em Ciência da Computação pela Unesp.

Sólida experiência no ecossistema Java e em tecnologias web.

Técnico em Informática formado pela Etec.