

非同期処理実行基盤

**Delayed 脱出 → Solid Queue 完全移行への旅路。**

**Kaigi on Rails 2025**

**2025/9/27**



**小林翔平 @srockstyle**



# Sponsor RubyStackNews

Reach senior Ruby & Rails engineers worldwide

Put your brand in front of experienced developers, tech leads, and decision makers across the global Ruby ecosystem.

[Become a Sponsor](#)

## Reach hundreds of Ruby & backend candidates daily

Ruby Stack News connects  
your company with  
experienced developers  
looking for meaningful  
work.



→ Post a job on Ruby Stack News

Promote your job on Ruby Stack News

## Apply to curated Ruby & Ruby on Rails jobs

Discover curated opportunities  
from companies hiring  
experienced Ruby  
developers.



Register on our job board  
and unlock opportunities  
for experienced developers.

Apply on our job board

KONNICHWA, I'm Shohei Kobayashi@srockstyle

# Site Reliability Engineer

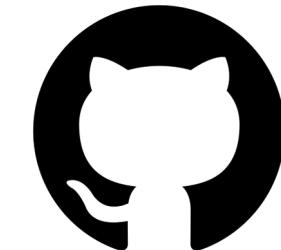
in Platform Engineering Unit at Studist



@srockstyle



@srockstyle.com



github.com/srockstyle



<https://www.srockstyle.com/>

苗字で呼ばれると返事できない体になつてゐるので、  
**しょーへーさん** or **すろっくさん**って呼んでください

KONNichiwa, I'm Shohei Kobayashi@srockstyle

# Site Reliability Engineer

in Platform Engineering Unit at Studist



KONNICHWA, I'm Shohei Kobayashi@srockstyle

# Site Reliability Engineer

in Platform Engineering Unit at Studist



**Engineer history 2005 ~**

KONNichiwa, I'm Shohei Kobayashi@srockstyle

# Site Reliability Engineer

in Platform Engineering Unit at Studist



**Engineer history 2005 ~**



**Rails3 ~**

KONNICHWA, I'm Shohei Kobayashi@srockstyle

# Site Reliability Engineer

in Platform Engineering Unit at Studist



**Engineer history 2005 ~**



**Rails3 ~**



**Vine Linux 2.0 ~**  
**Fedora Core 1 ~**

KONNICHWA, I'm Shohei Kobayashi@srockstyle

# Site Reliability Engineer

in Platform Engineering Unit at Studist



**Engineer history 2005 ~**



**Rails3 ~**



**Vine Linux 2.0 ~**  
**Fedora Core 1 ~**

**Web2.0 !**



# イラスト描いたりします

社内のプレゼン資料に使って同僚にドヤ顔するのが趣味

東方 Project 好きだぜ



# Delayed → Solid Queue 完全移行への旅路。

and all the things in between

Kaigi on Rails2025  
September 27th,2025



## 入社直後に大規模プロジェクトを推進！経験豊富なエンジニアが見つけた合意形成の新たなかたち

♡ 21

studist スタディスト公式note  
2025年8月7日 10:39

「合意形成」

言葉にすらのは簡単ですが、日々の業務においてアカペラでまでに難しく、主

スタディスト Tech Blog



スタディストがお届けする発展と成長の物語。技術、デザイン、組織、マネジメントなどを発信します。

[Follow publication](#)

## 非同期処理実行基盤、いざSolidQueueへ！ Teachme Bizの非同期処理新航路への旅路。

 Shohei Kobayashi [Follow](#) 16 min read · Jun 24, 2025

19



...  
↑  
↓  
...

SREユニットの小林（し）(@srockstyle) です。

<https://note.com/studist/n/nb16a0b57bbdf>

<https://studist.tech/delayed-job-to-solidqueue-migration-fd50ad239a>

# アジェンダ

- はじめに: Teachme Biz について
- 課題 Part その 1
- 選定
- 設計と実装そして移行
- 課題 Part その 2
- まとめ

- はじめに: Teachme Biz について
- 課題 Part その 1
- 選定
- 設計と実装そして移行
- 課題 Part その 2
- まとめ

# マニュアルで生産性向上

手順が見える、伝わる、拡がる  
マニュアル作成・共有システム

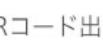
5分で  
わかる

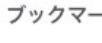
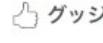
資料をダウンロードする



# リリース 10 年以上の Rails アプリケーション

99. スタディスト 社内共有

Language ▾  スライドショー  PDF出力  QRコード出力 

ブックマーク  ゲッジョブ  コメント(2)  編集  複製 4 人がゲッジョブと言っています。

最終更新 :   タイトルとURLをコピー

## スタディスト地方出張グルメ

出張先での楽しみはグルメ。スタディストのメンバーたちが全国各地で見つけた絶品店の紹介マニュアルです。

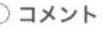


FOOD AND DRINKS  
vector background

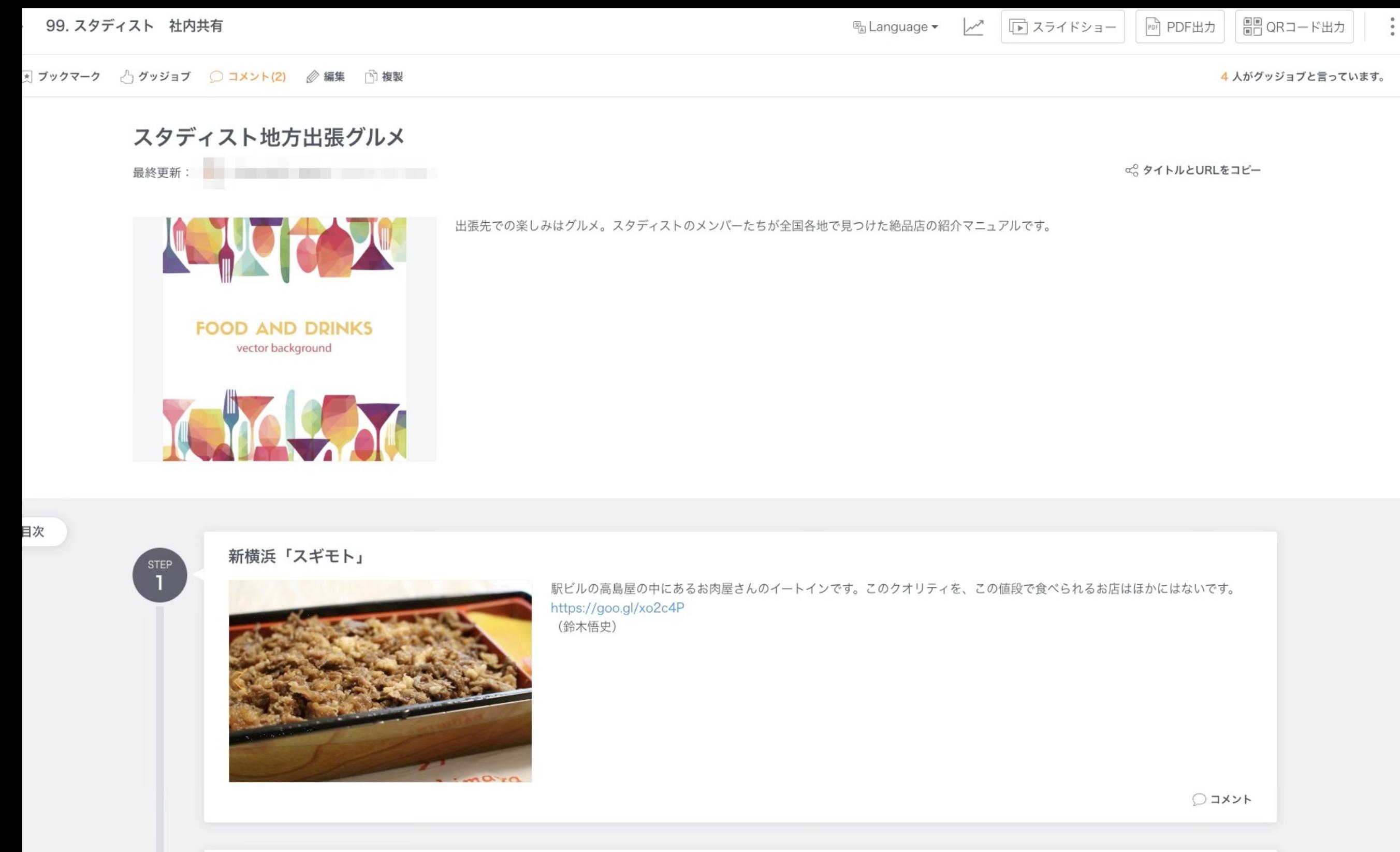
新横浜「スギモト」

駅ビルの高島屋の中にあるお肉屋さんのイートインです。このクオリティを、この値段で食べられるお店はほかにはないです。  
<https://goo.gl/xo2c4P>  
(鈴木悟史)



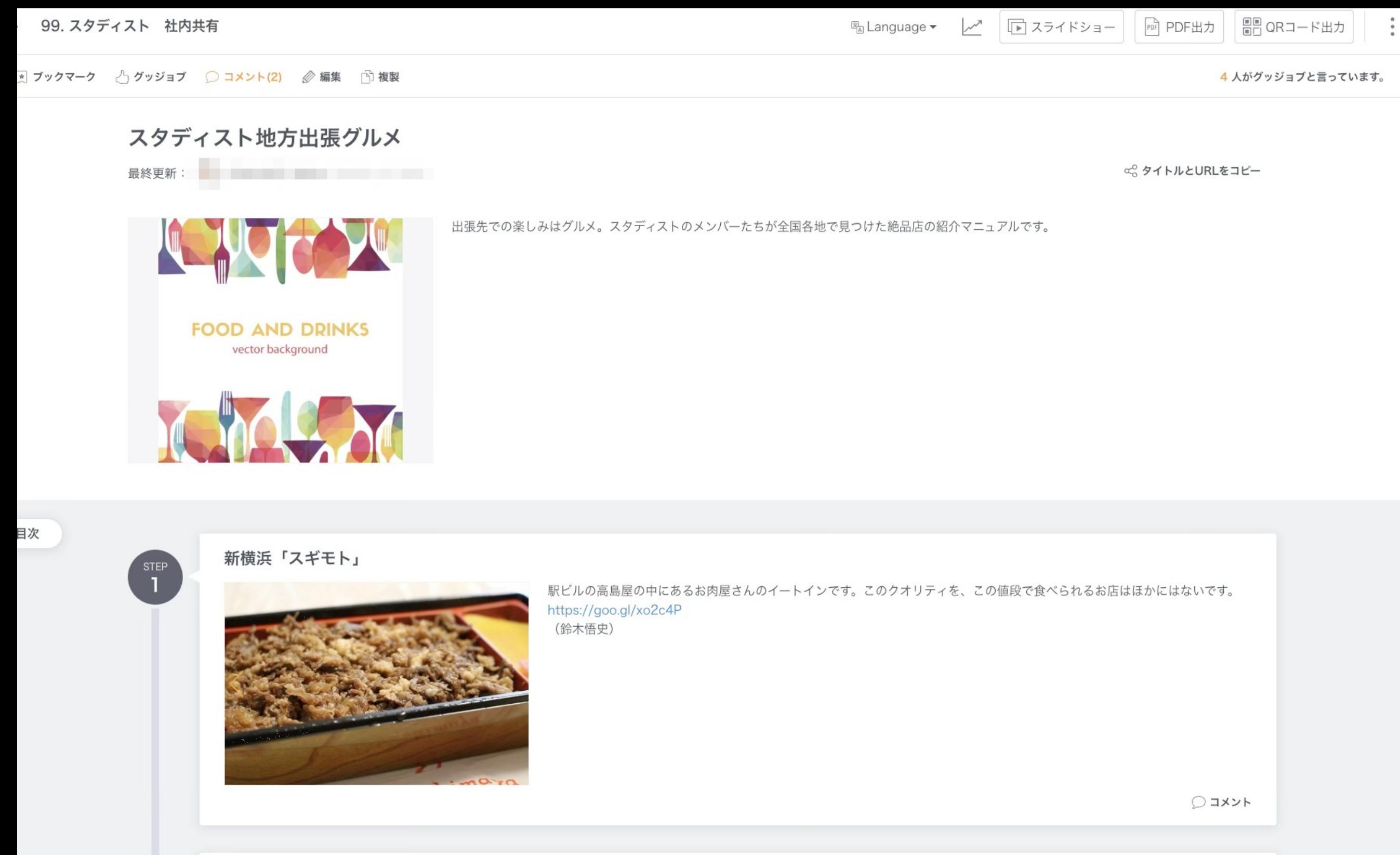
 コメント

# リリース 10 年以上の Rails アプリケーション



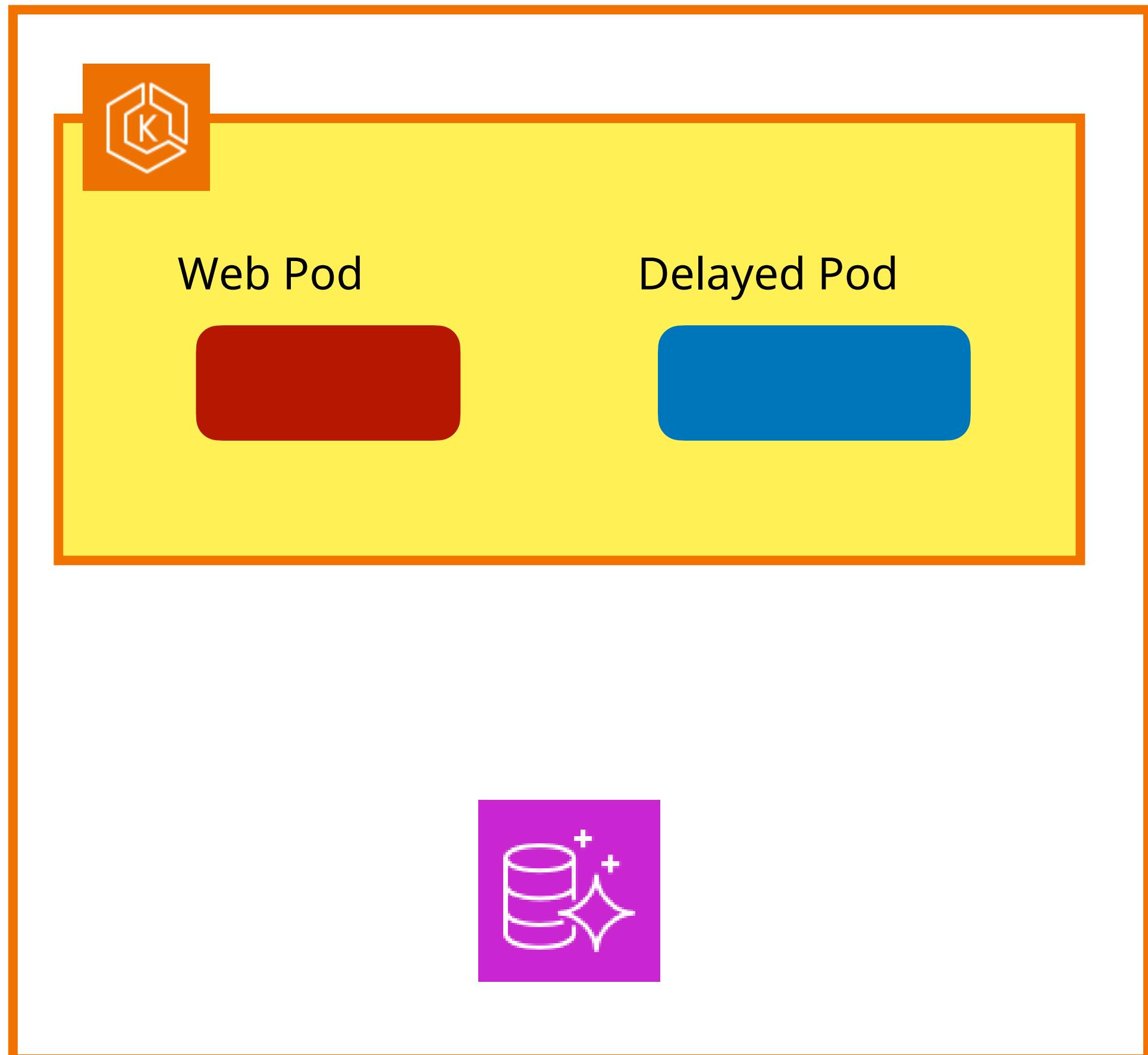
2012 年リリース Rails3.2.1

# リリース 10 年以上の Rails アプリケーション

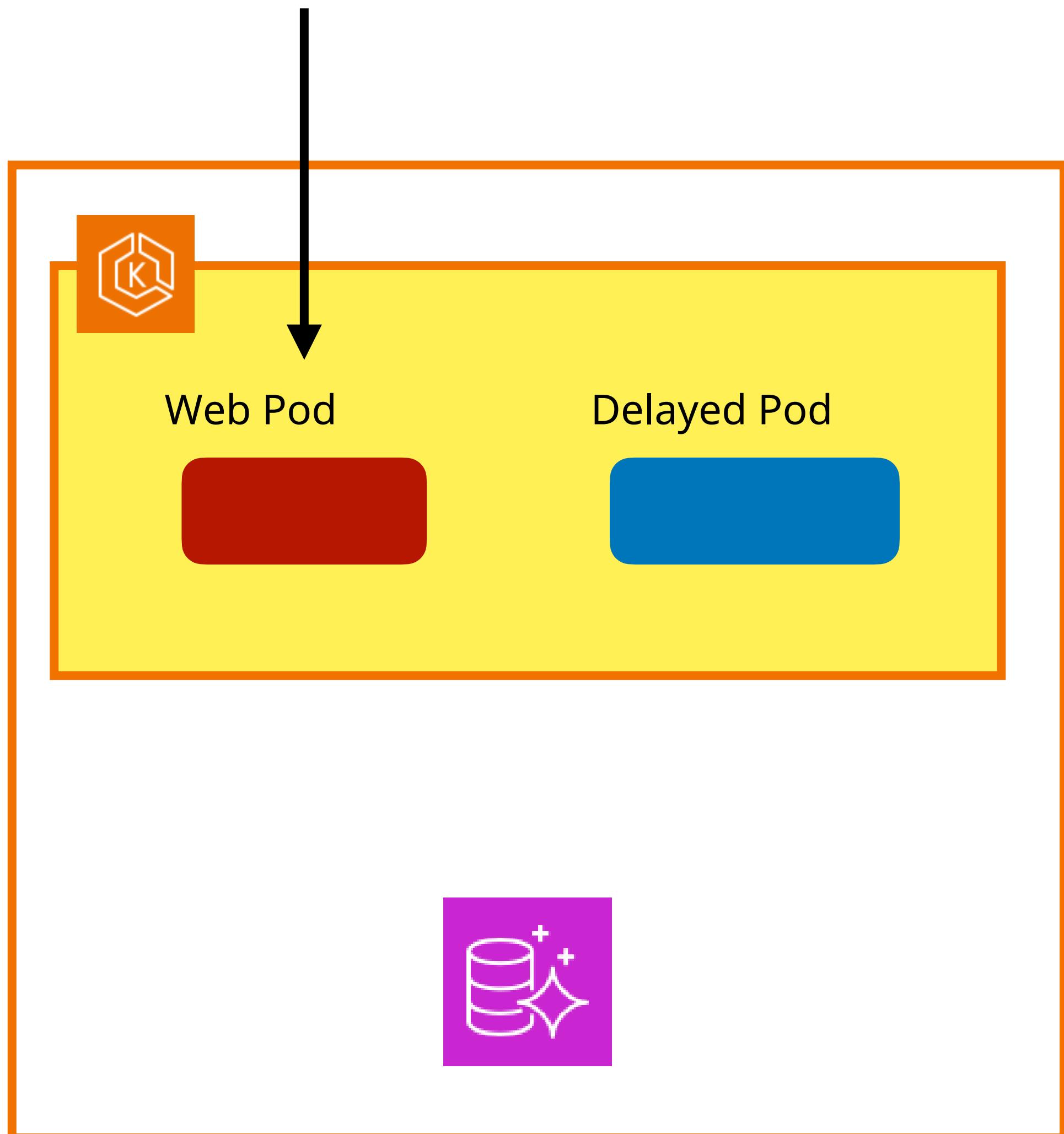


2012 年リリース Rails3.2.1 > 2024 年 10 月当時 Rails7.1

# 非同期処理の簡単なおさらい

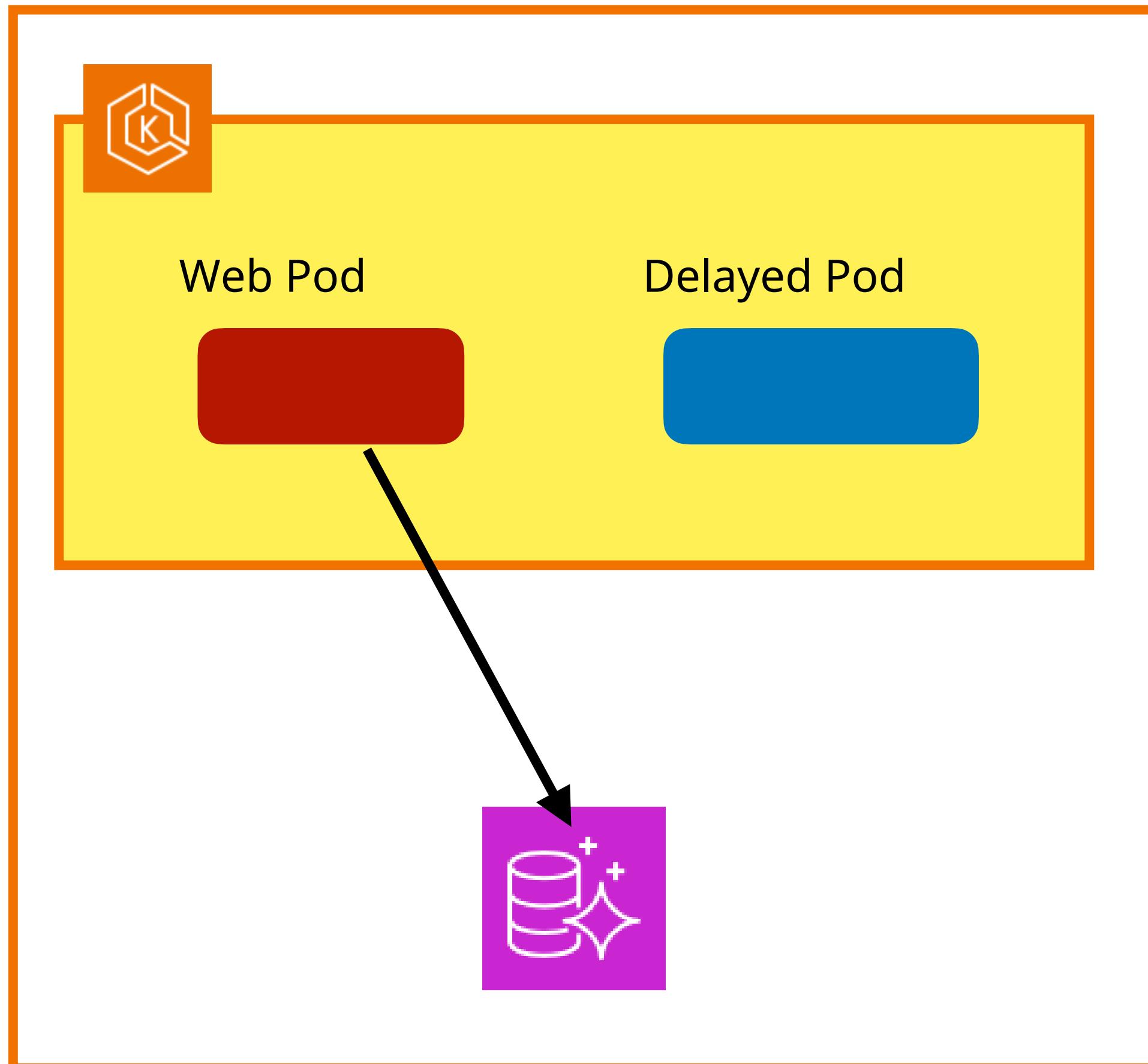


# 非同期処理の簡単なおさらい



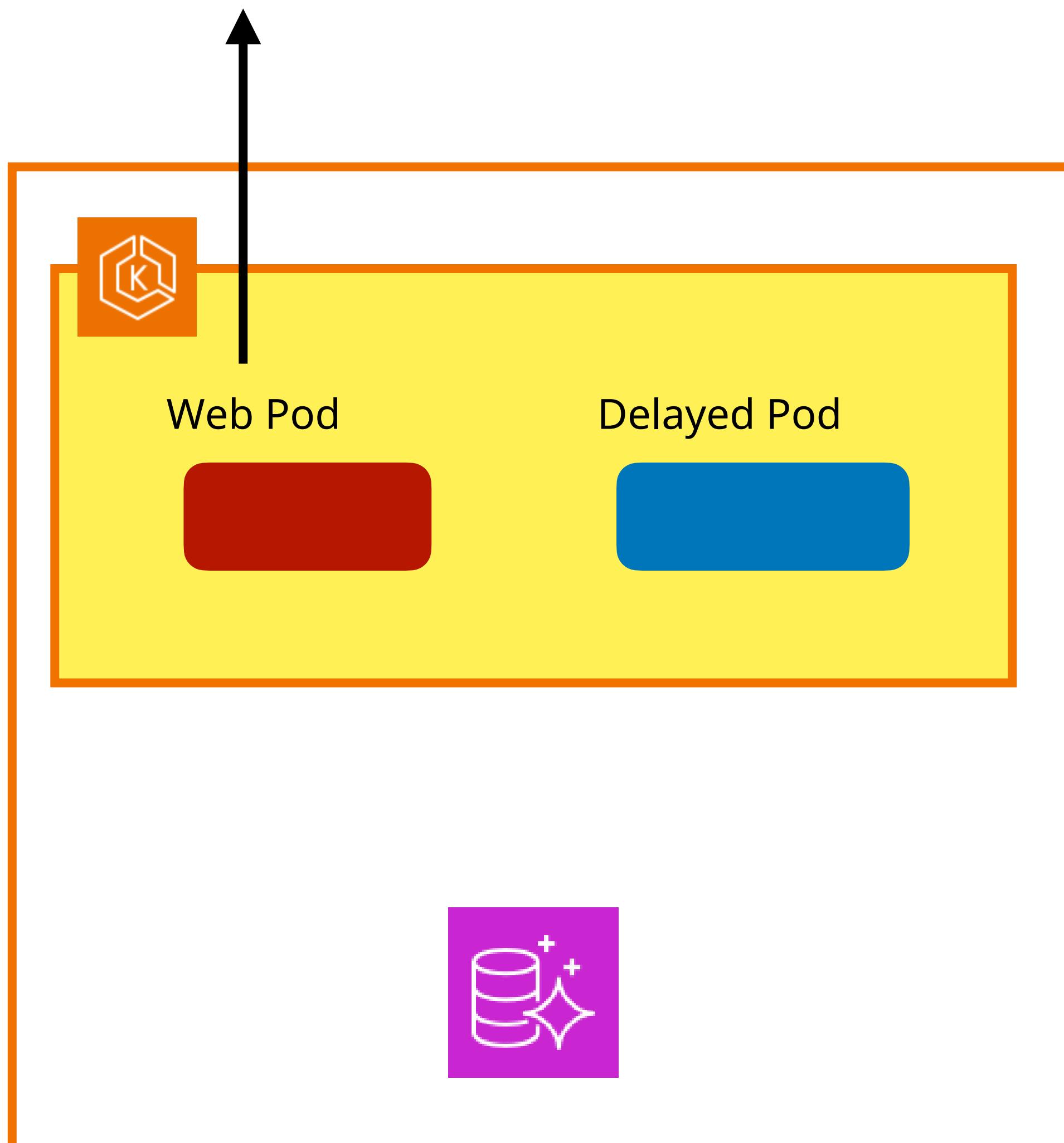
1. ユーザが操作する

# 非同期処理の簡単なおさらい



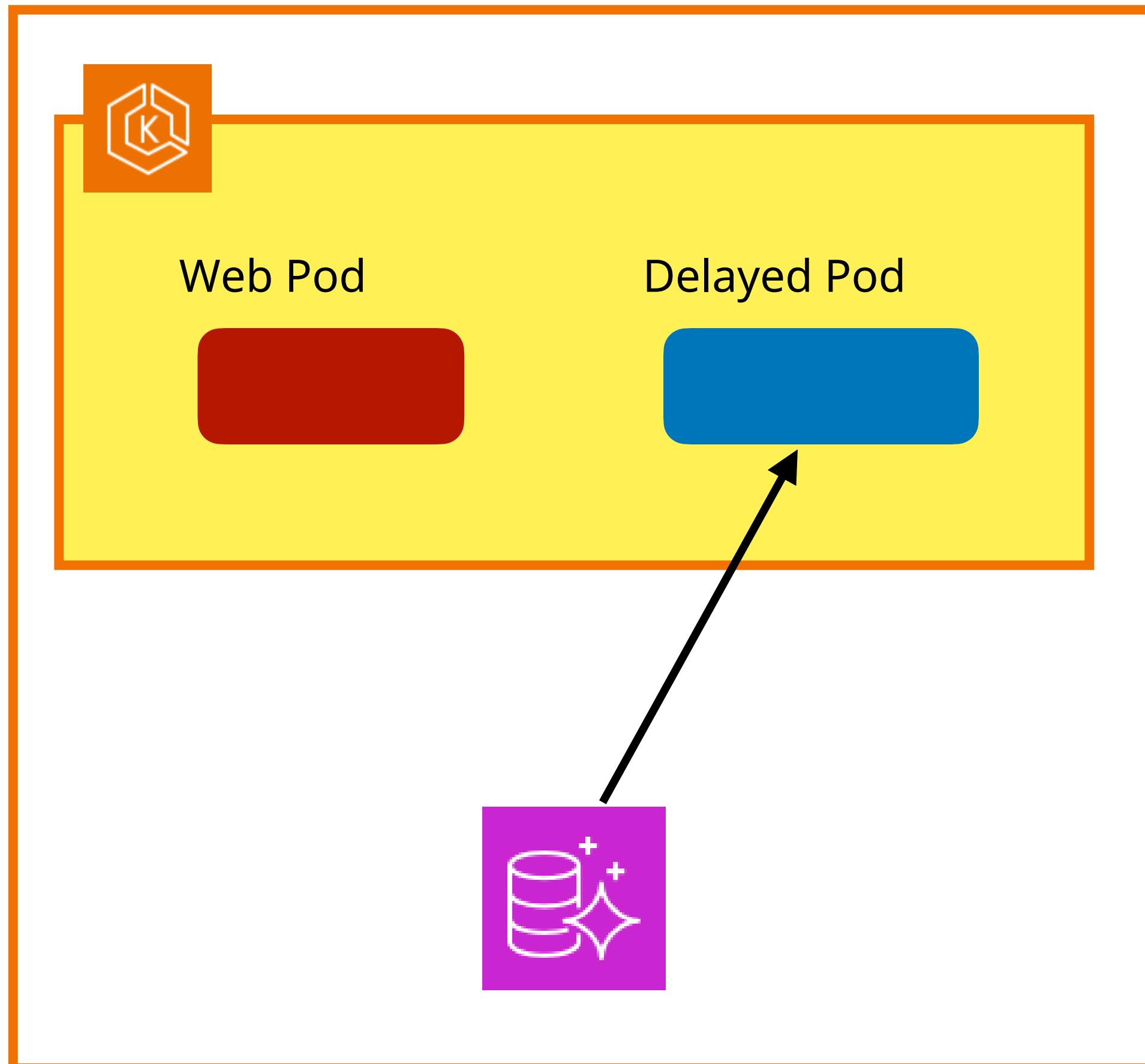
1. ユーザが操作する
2. DB に処理がエンキューされる

# 非同期処理の簡単なおさらい



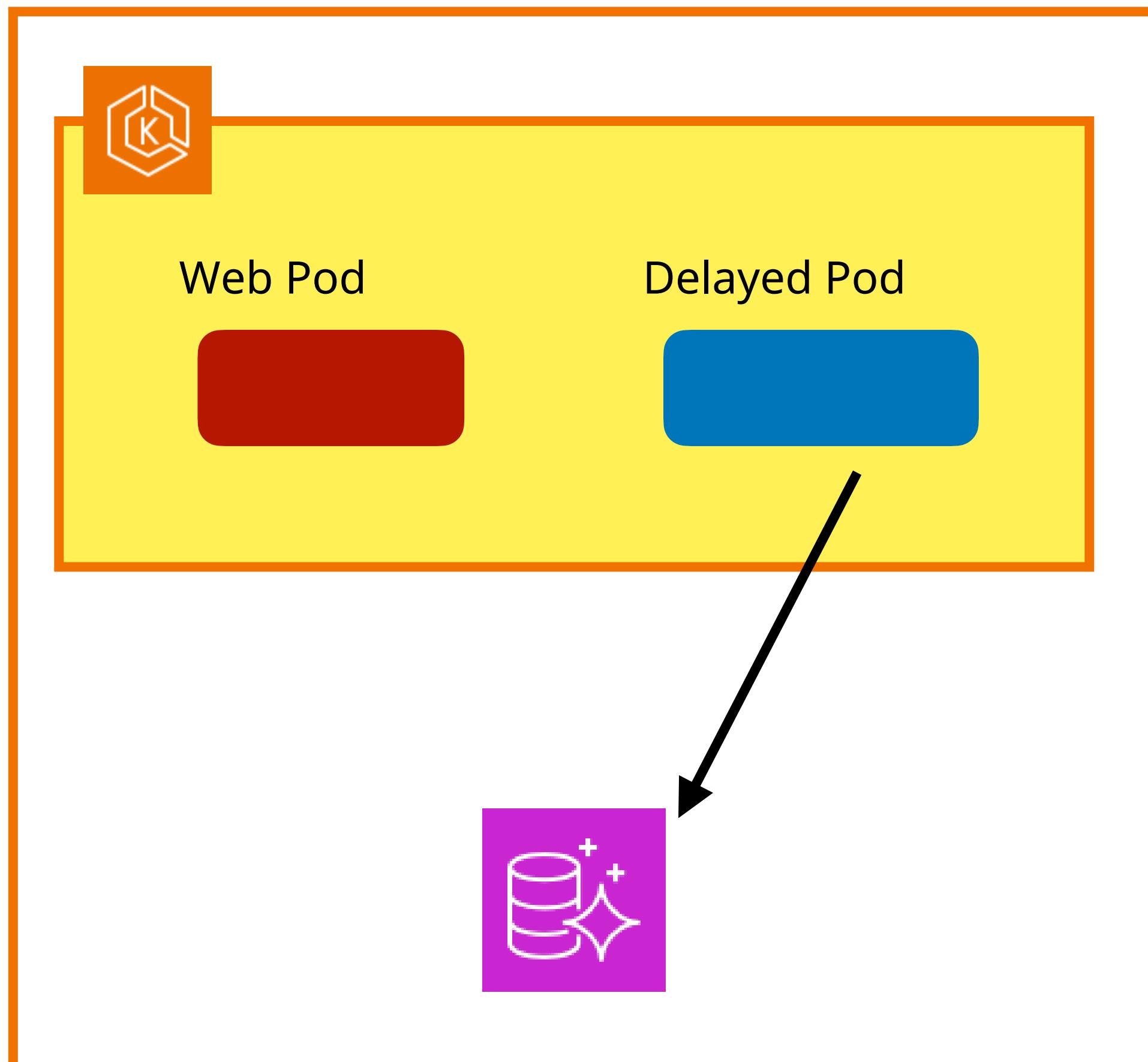
1. ユーザが操作する
2. DB に処理がエンキューされる
3. ユーザにレスポンスが返される

# 非同期処理の簡単なおさらい



1. ユーザが操作する
2. DB に処理がエンキューされる
3. ユーザにレスポンスが返される
4. 非同期バックエンドが処理

# 非同期処理の簡単なおさらい



1. ユーザが操作する
2. DB に処理がエンキューされる
3. ユーザにレスポンスが返される
4. 非同期バックエンドが処理
5. 処理結果を DB に保存して完了

# 重要な処理＝非同期処理が多い

Chatbot で Chat 開始



初期マニュアル作成



フォルダの人数計算



各種情報更新後メール送信



顧客情報削除処理



AI を使った自動生成



動画・ PDF からマニュアル作成



- はじめに: Teachme Biz について
- **課題 Part その 1**
- 選定
- 設計と実装そして移行
- 課題 Part その 2
- まとめ

# 7 years ago...

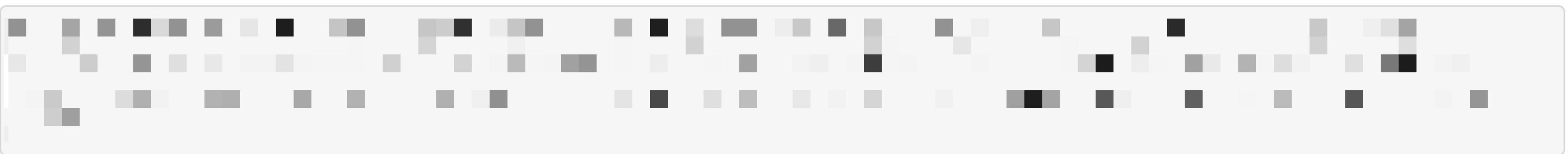
Messages Add canvas Files Bookmarks +



**katsuhisa\_ / Katsuhisa Kitano / 北野 勝久** 10:29 AM

August 7th, 2018 ▾

こんな感じでERROR が発生していることはわかってるんだけど、  
それがプロセスを殺したかどうかまでは出てない感じ。  
おそらく、このERROR が出たタイミングでプロセスが死んでる。(と推測している。)



10:30 AM

またデッドロックかあ

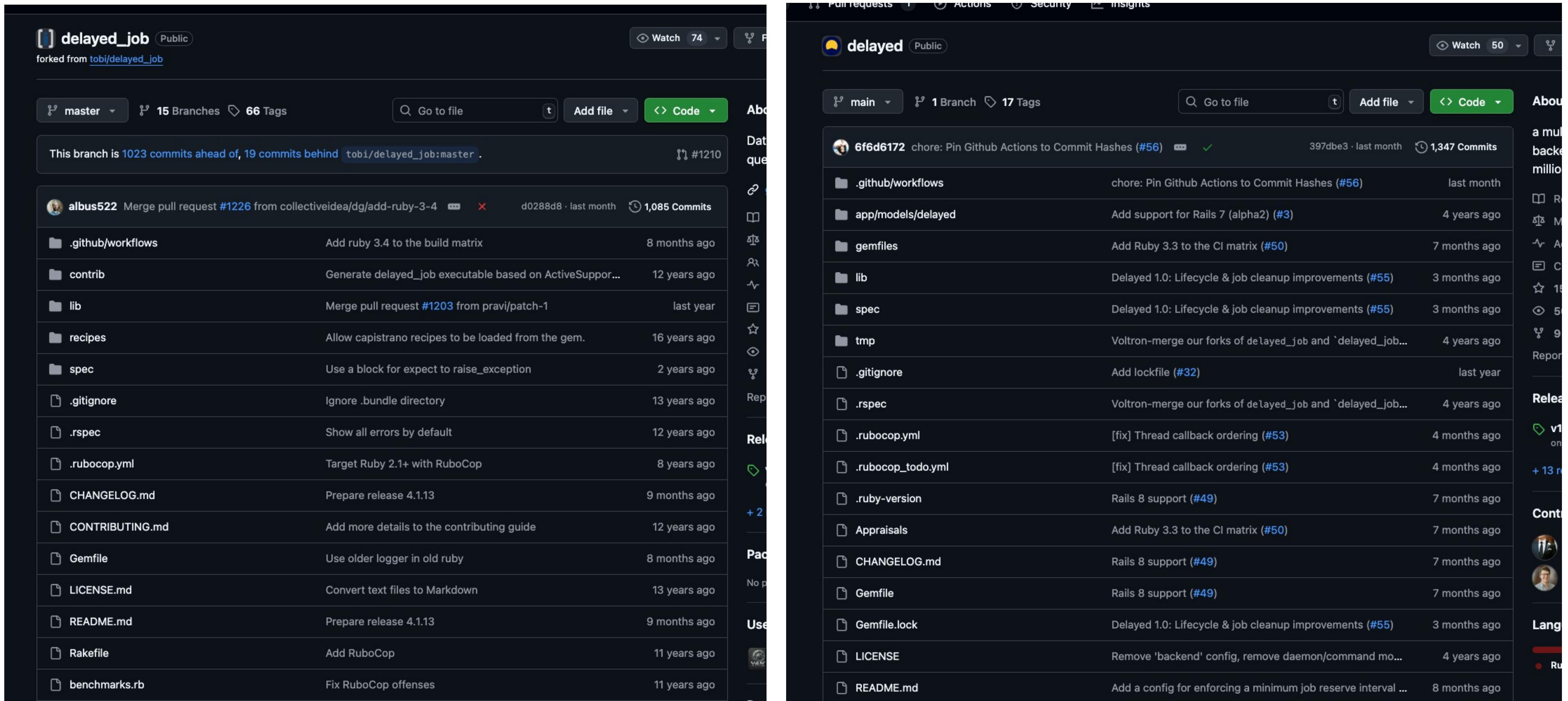


**katsuhisa\_ / Katsuhisa Kitano / 北野 勝久** 10:31 AM

そうなんだよねえ

# DelayedJob / Delayed

## DB をキューとする非同期バックエンド



**delayed\_job** (Public) forked from [tobi/delayed\\_job](#)

master 15 Branches 66 Tags

This branch is 1023 commits ahead of, 19 commits behind [tobi/delayed\\_job:master](#).

**DelayedJob** (Public) 6f6d6172 chore: Pin Github Actions to Commit Hashes (#56) 397dbe3 · last month 1,347 Commits

- .github/workflows chore: Pin Github Actions to Commit Hashes (#56) last month
- app/models/delayed Add support for Rails 7 (alpha2) (#3) 4 years ago
- gemfiles Add Ruby 3.3 to the CI matrix (#50) 7 months ago
- lib Delayed 1.0: Lifecycle & job cleanup improvements (#55) 3 months ago
- spec Delayed 1.0: Lifecycle & job cleanup improvements (#55) 3 months ago
- tmp Voltron-merge our forks of delayed\_job and `delayed\_job...` 4 years ago
- .gitignore Add lockfile (#32) last year
- .rspec Voltron-merge our forks of delayed\_job and `delayed\_job...` 4 years ago
- .rubocop.yml [fix] Thread callback ordering (#53) 4 months ago
- .rubocop\_todo.yml [fix] Thread callback ordering (#53) 4 months ago
- .ruby-version Rails 8 support (#49) 7 months ago
- Appraisals Add Ruby 3.3 to the CI matrix (#50) 7 months ago
- CHANGELOG.md Rails 8 support (#49) 7 months ago
- Gemfile Rails 8 support (#49) 7 months ago
- Gemfile.lock Delayed 1.0: Lifecycle & job cleanup improvements (#55) 3 months ago
- LICENSE Remove 'backend' config, remove daemon/command mo... 4 years ago
- README.md Add a config for enforcing a minimum job reserve interval ... 8 months ago

# 課題とやりたいこと

課題とやりたいこと

パフォーマンスとスケーラビリティの向上

柔軟なジョブ管理と優先制御

開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上



綺麗に線形にスケールアウトできるようにしたい

## 柔軟なジョブ管理と優先制御

## 開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい

## 柔軟なジョブ管理と優先制御

## 開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

## 開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

- 優先度を定義したら優先度を守ってほしい

## 開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

- 優先度を定義したら優先度を守ってほしい
- ジョブごとにキューと Worker を分けたい

## 開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

- 優先度を定義したら優先度を守ってほしい
- ジョブごとにキューと Worker を分けたい
- 定期実行処理をまとめたい

## 開発と運用の効率化

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

- 優先度を定義したら優先度を守ってほしい
- ジョブごとにキューと Worker を分けたい
- 定期実行処理をまとめたい

## 開発と運用の効率化

- ActiveJob に移行したい（Delayed 専用の書き方と ActiveJob の混在廃止）

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

- 優先度を定義したら優先度を守ってほしい
- ジョブごとにキューと Worker を分けたい
- 定期実行処理をまとめたい

## 開発と運用の効率化

- ActiveJob に移行したい（Delayed 専用の書き方と ActiveJob の混在廃止）
- できるだけ Rails Way に乗せたい

# 課題とやりたいこと

## パフォーマンスとスケーラビリティの向上



綺麗に線形にスケールアウトできるようにしたい



処理できるジョブの数を増やしたい



リソースを有効活用したい

## 柔軟なジョブ管理と優先制御



優先度を定義したら優先度を守ってほしい



ジョブごとにキューと Worker を分けたい



定期実行処理をまとめたい

## 開発と運用の効率化



ActiveJob に移行したい (Delayed 専用の書き方と ActiveJob の混在廃止)



できるだけ Rails Way に乗せたい

# 線形的なスケールアウト不可能問題：前提

非同期処理は二種類存在。

# 線形的なスケールアウト不可能問題：前提

非同期処理は二種類存在。

A: ジョブとして長時間の処理をしてもらうもの

例：初回グループ作成 / 解約グループ削除

# 線形的なスケールアウト不可能問題：前提

非同期処理は二種類存在。

A: ジョブとして長時間の処理をしてもらうもの

例：初回グループ作成 / 解約グループ削除

B: とりあえず非同期にするが、なるべく早く処理して欲しいもの

例：全従業員にメール通知 / Chatbot の返信 / AI で自動マニュアル作成

# 線形的なスケールアウト不可能問題：前提

非同期処理は二種類存在。

A: ジョブとして長時間の処理をしてもらうもの

例：初回グループ作成 / 解約グループ削除

B: とりあえず非同期にするが、なるべく早く処理して欲しいもの

例：全従業員にメール通知 / Chatbot の返信 / AI で自動マニュアル作成

# 線形的なスケールアウト不可能問題

本屋（Web アプリケーション）



# 線形的なスケールアウト不可能問題

本屋（Web アプリケーション）



本がたくさん売れるから

店員（処理プロセス）を 1 人から 2 人としたよ！



# 線形的なスケールアウト不可能問題

本屋（Web アプリケーション）



本がたくさん売れるから

店員（処理プロセス）を 1 人から 2 人としたよ！



2 人としたからお客様（処理）を捌くスピードも

2 倍だ！

# 線形的なスケールアウト不可能問題

本（処理）がたくさん売れる（入る）から

店員（処理プロセス）を1人から2人にしたのに…



# 線形的なスケールアウト不可能問題

本（処理）がたくさん売れる（入る）から  
店員（処理プロセス）を1人から2人としたのに…



電卓は一個しかねえぞ！！！



# 線形的なスケールアウト不可能問題

本（処理）がたくさん売れる（入る）から  
店員（処理プロセス）を1人から2人としたのに…

電卓は一個しかねえぞ！！！



2人としたのにお客さんを捌くスピードが…

# 線形的なスケールアウト不可能問題

本（処理）がたくさん売れる（入る）から  
店員（処理プロセス）を1人から2人としたのに…

電卓は一個しかねえぞ！！！



2人としたのにお客さんを捌くスピードが…

2倍じゃねえ！

# 線形的なスケールアウト不可能問題

ジョブがいっぱいあるのでいっぱい処理したい

Delayed

DB

# 線形的なスケールアウト不可能問題

ジョブがいっぱいあるのでいっぱい処理したい

処理 Worker を増やす = 処理がもっと捌けるはず

Delayed

Delayed

Delayed

DB

# 線形的なスケールアウト不可能問題

ジョブがいっぱいあるのでいっぱい処理したい

処理実行!

Delayed

Delayed

Delayed

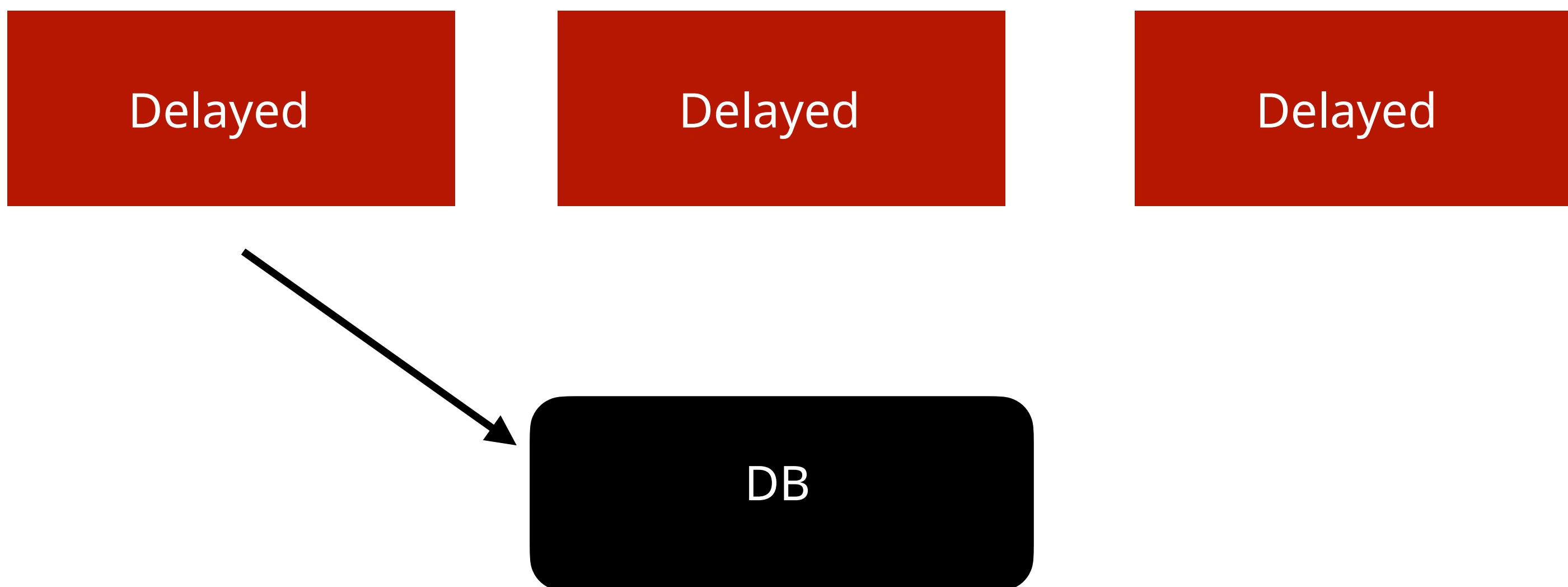
DB

# 線形的なスケールアウト不可能問題

ジョブがいっぱいあるのでいっぱい処理したい

処理実行!

1 台の Worker が取得中

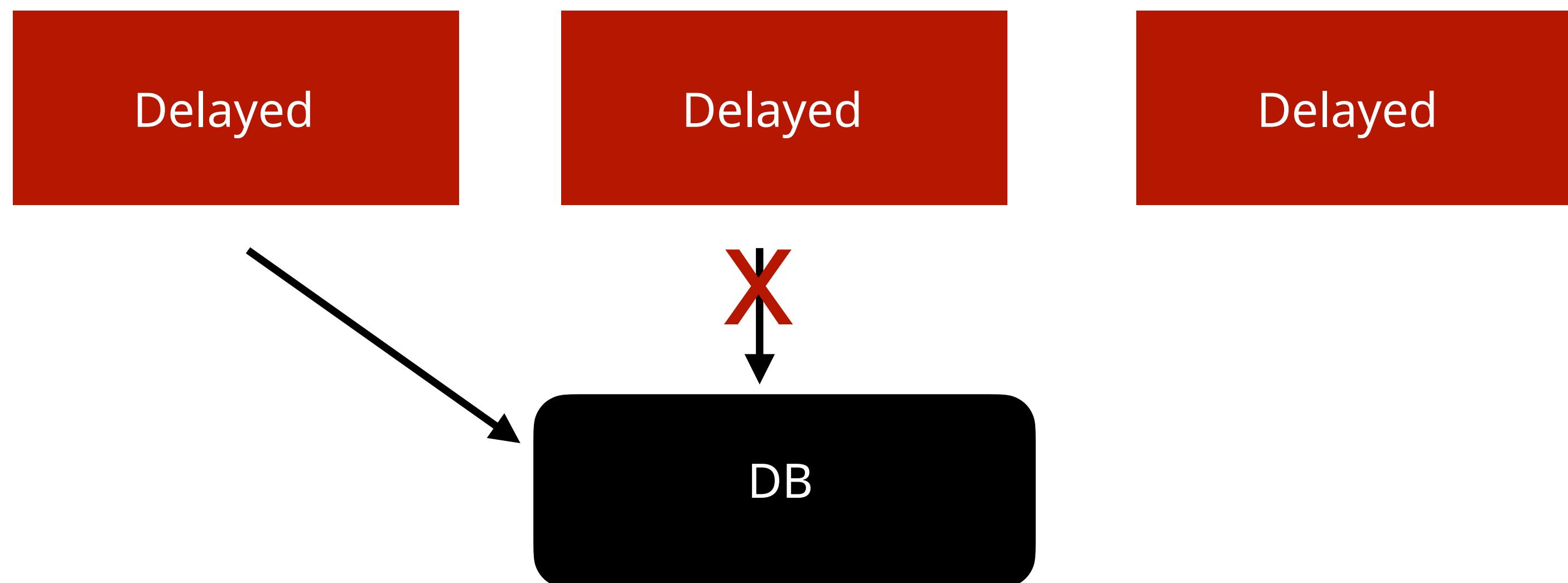


# 線形的なスケールアウト不可能問題

ジョブがいっぱいあるのでいっぱい処理したい

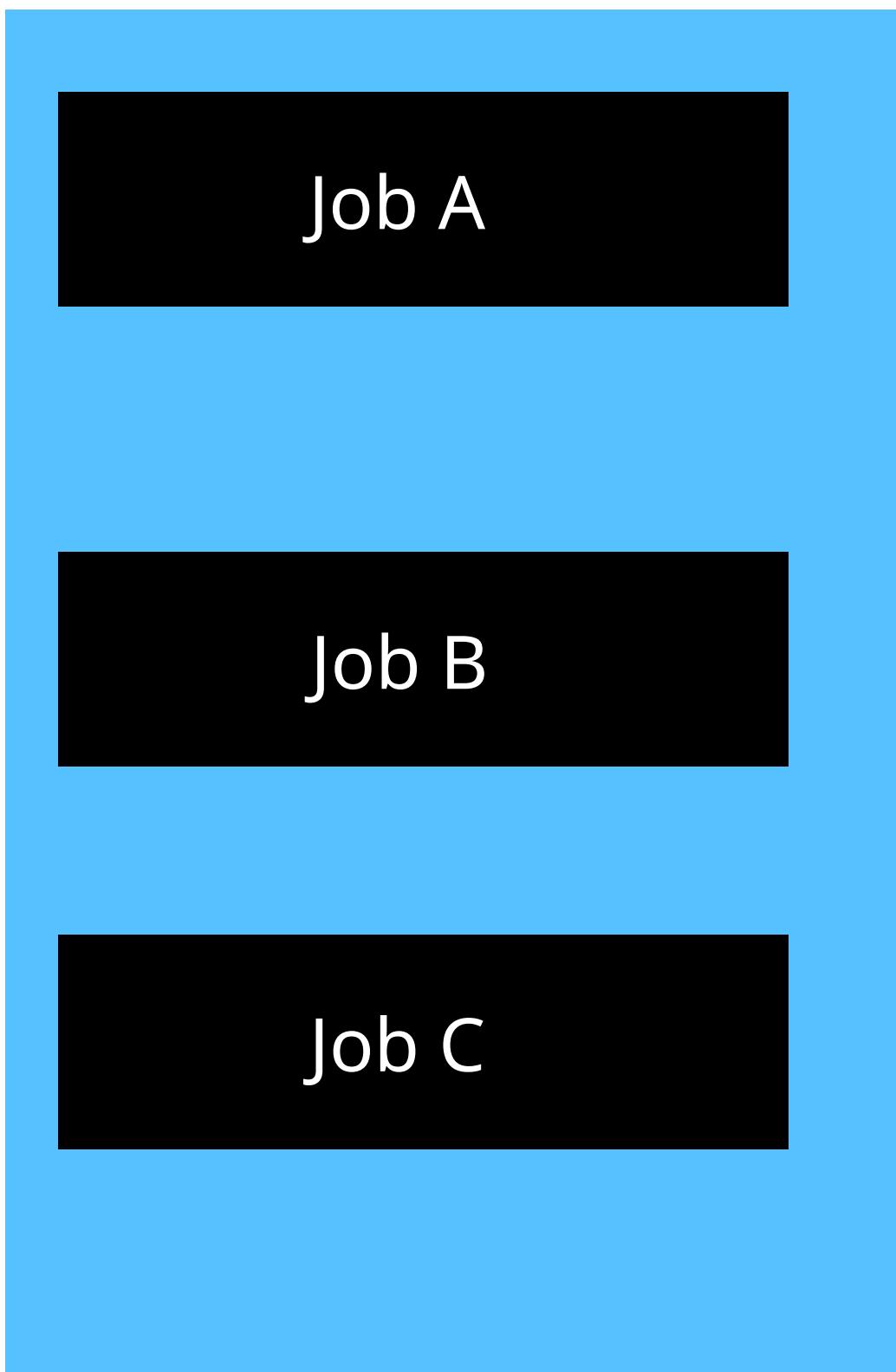
処理実行!

他の worker は待つ。



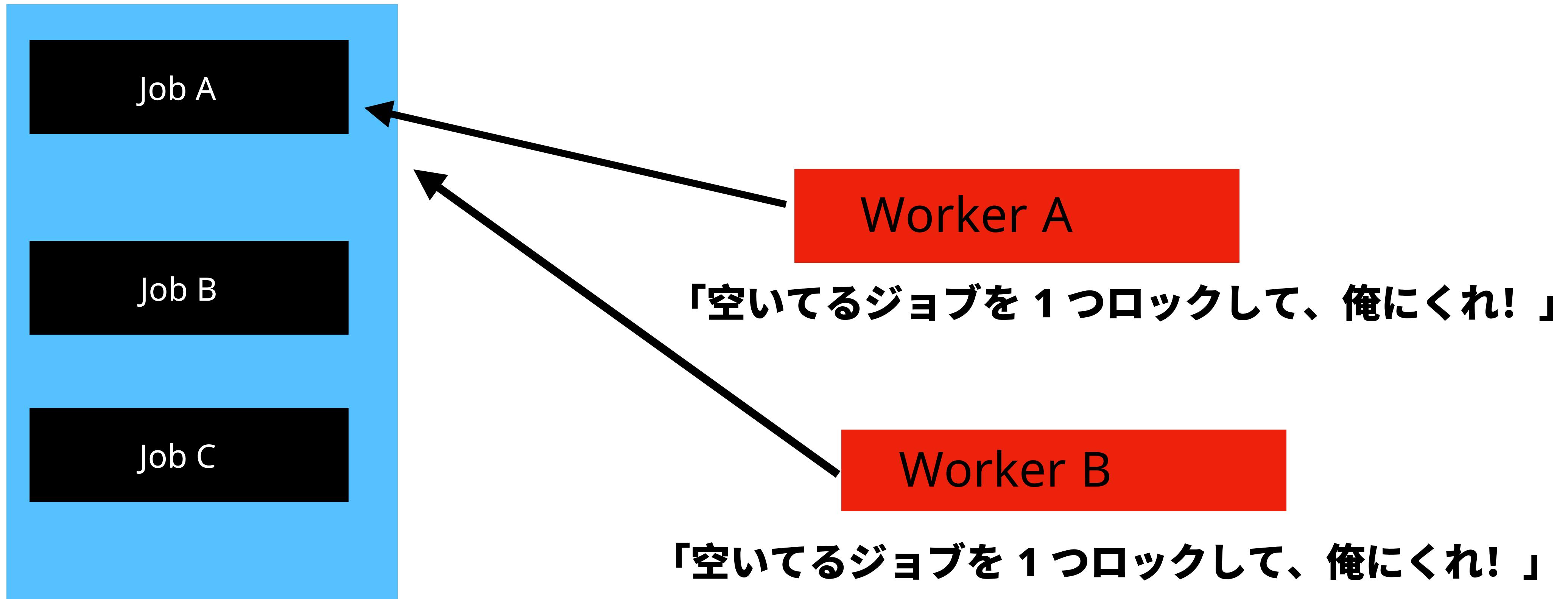
# 線形的なスケールアウト不可能問題

Job Table

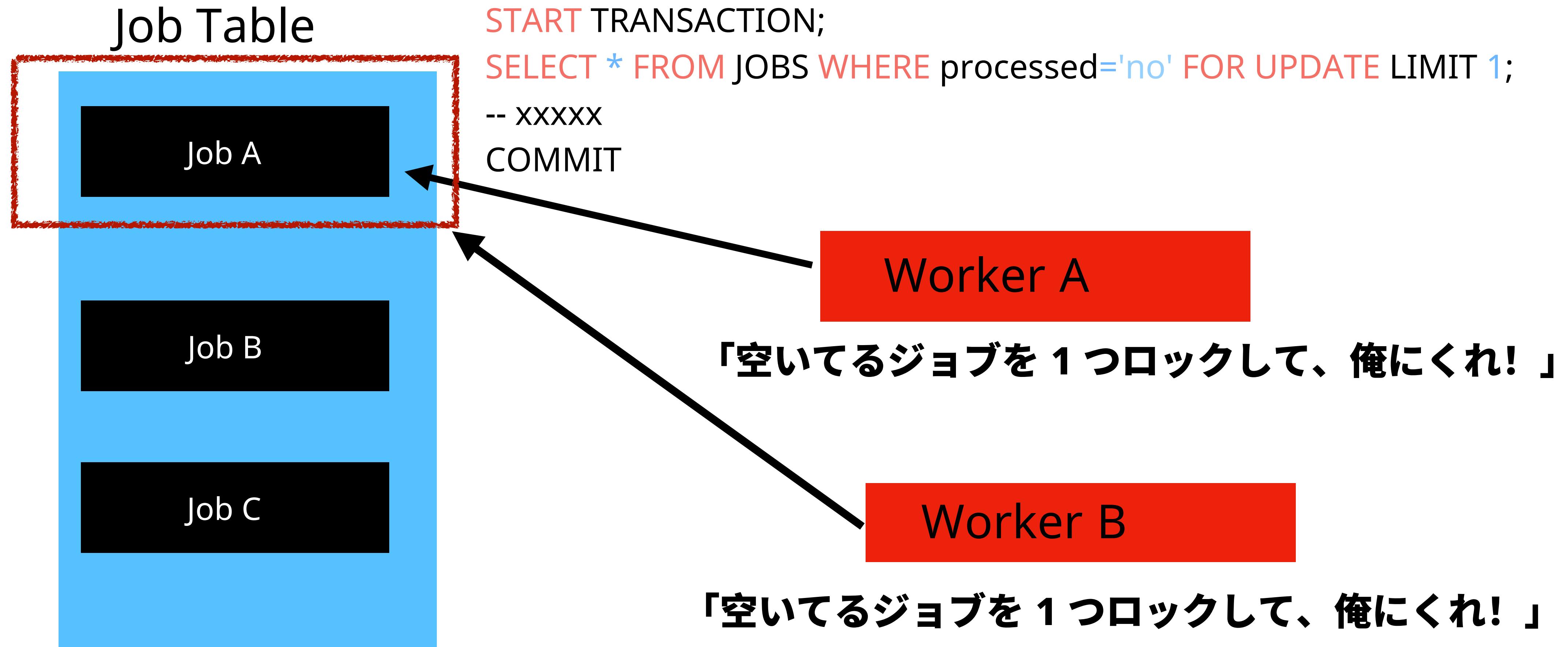


# 線形的なスケールアウト不可能問題

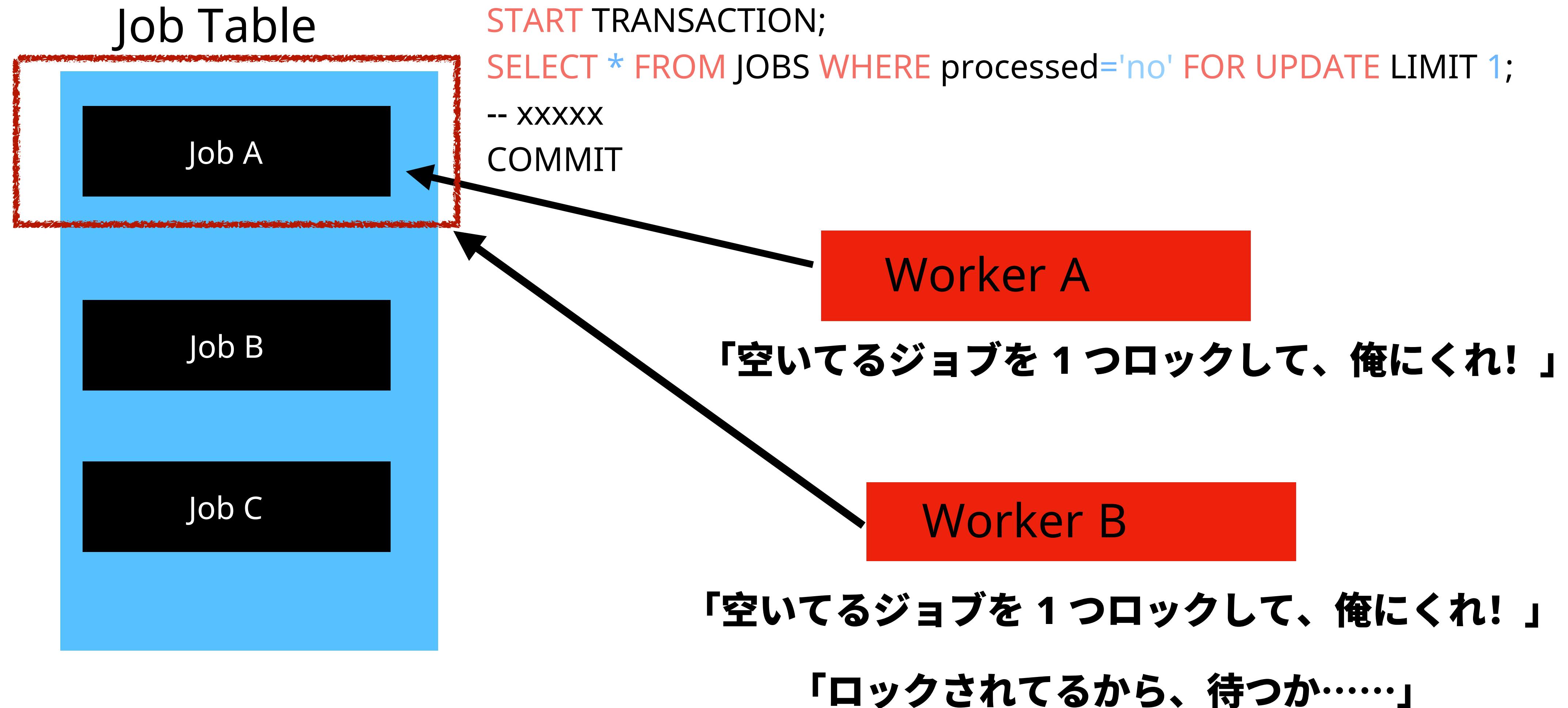
Job Table



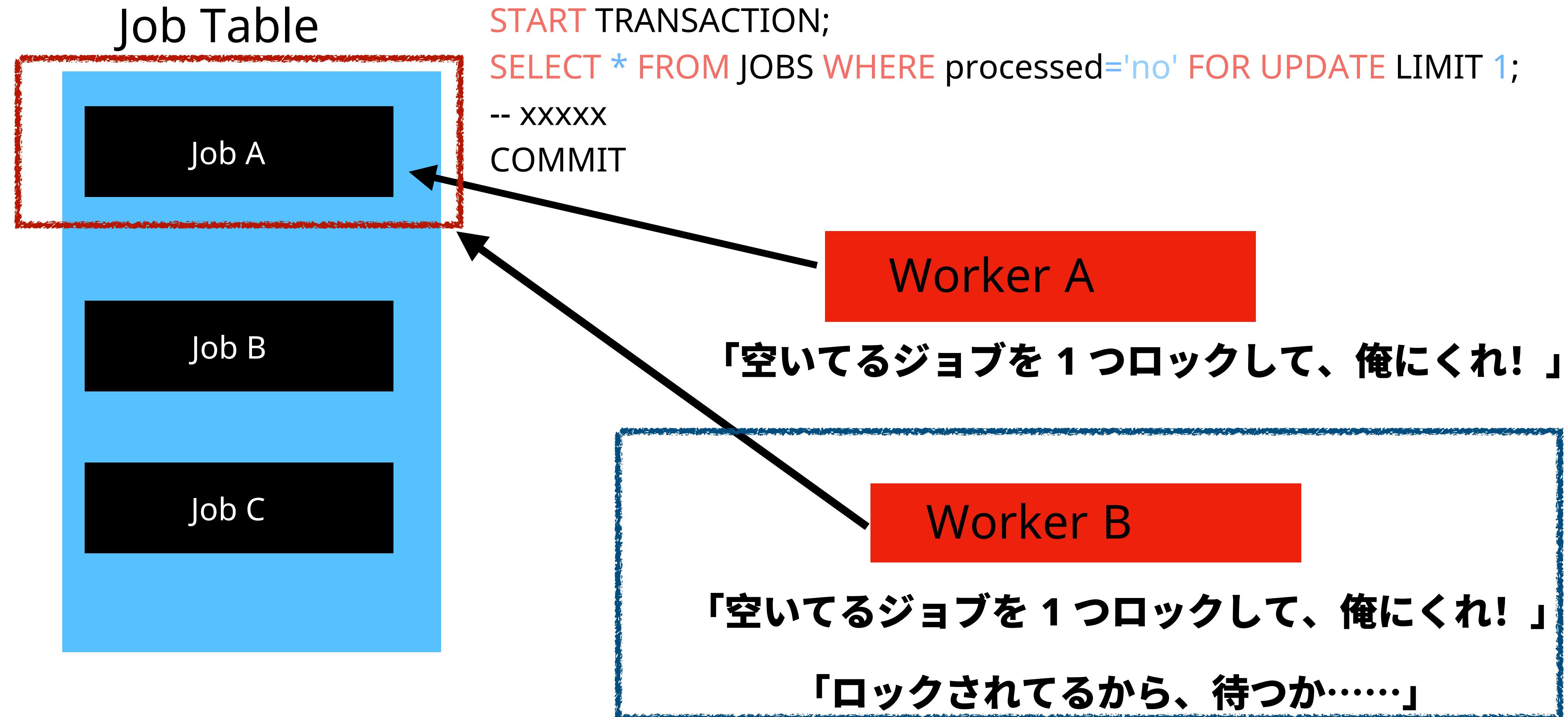
# 線形的なスケールアウト不可能問題



# 線形的なスケールアウト不可能問題

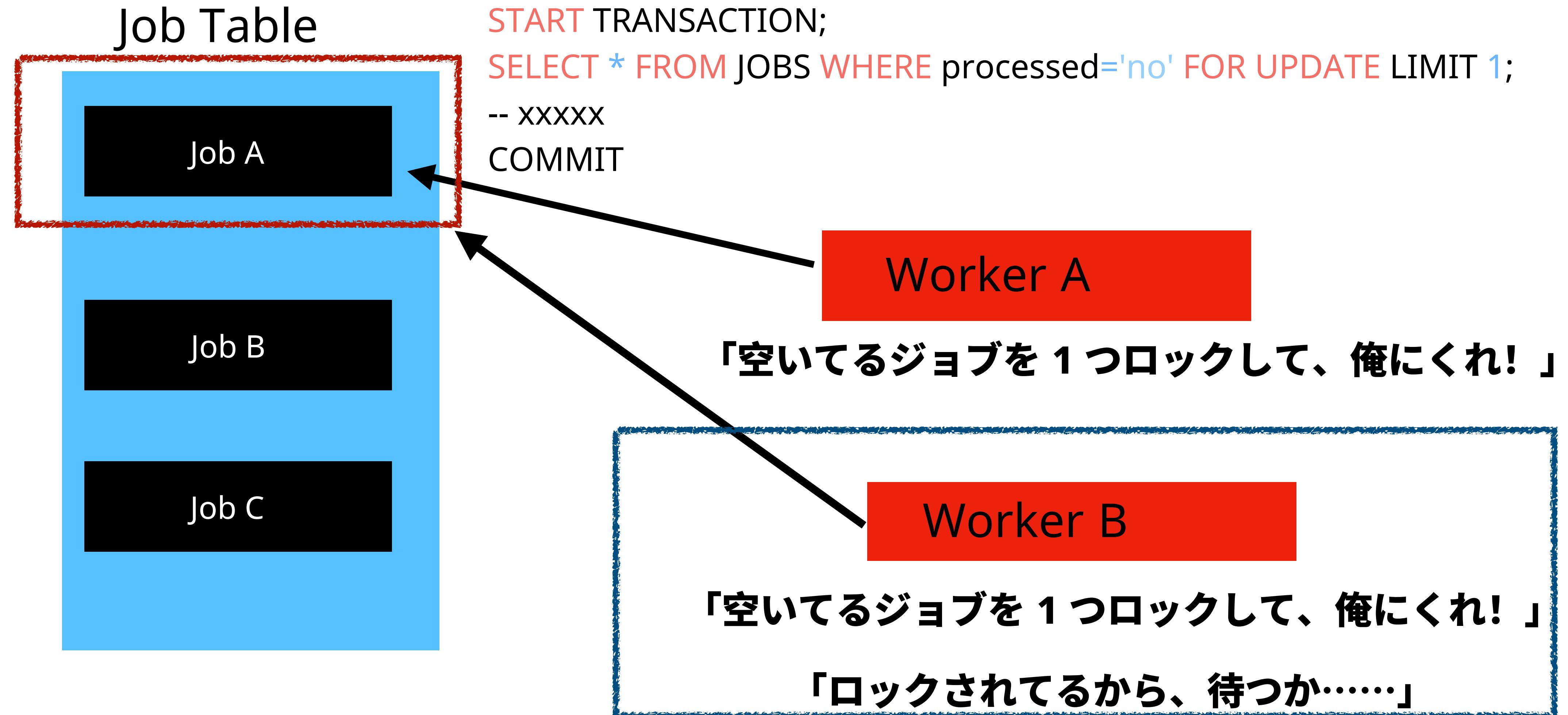


# 線形的なスケールアウト不可能問題



Transaction が commit されるまで JobB をとりにいかず、待ち worker になる

# 線形的なスケールアウト不可能問題



Transaction が commit されるまで JobB をとりにいかず、待ち worker になる  
= Worker を増やしても、ジョブの処理速度が綺麗に倍になってくれない

- はじめに: Teachme Biz について
- 課題 Part その 1
- **選定**
- 設計と実装
- 移行
- 課題 Part その 2
- まとめ

# 現状

現状

非同期処理

ActiveJob を使ってるもの

# 現状

## 非同期処理

ActiveJob を使ってるもの

ClassName.delay.exec といった .delay メソッドを使ってるものの

# 現状

## 非同期処理

ActiveJob を使ってるもの

ClassName.delay.exec といった .delay メソッドを使ってるもの

## 定期処理管理

Whenever -> デプロイ時に crontab 作成

# 現状

## 非同期処理

ActiveJob を使ってるもの

ClassName.delay.exec といった .delay メソッドを使ってるものの

## 定期処理管理

Whenever -> デプロイ時に crontab 作成

## ジョブのキュー

DB in MySQL8

# 現状

## 非同期処理

ActiveJob を使ってるもの

ClassName.delay.exec といった .delay メソッドを使ってるものの

## 定期処理管理

Whenever -> デプロイ時に crontab 作成

## ジョブのキュー

DB in MySQL8

## SRE チームで合意した方針

運用対象は増やさない & 運用対象は極力減らす

# Sidekiq vs Solid Queue

# 比較観点の一例

# 比較観点の一例

- 線形スケールアウト可能

# 比較観点の一例

- 線形スケールアウト可能
- 新しい AWS リソースが不要

## 比較観点の一例

- 線形スケールアウト可能
- 新しい AWS リソースが不要
- 用途に応じて複数のキュー管理ができること

## 比較観点の一例

- 線形スケールアウト可能**
- 新しい AWS リソースが不要**
- 用途に応じて複数のキュー管理ができる**
- DB ロックでパフォーマンス劣化しない**

## 比較観点の一例

- 線形スケールアウト可能
- 新しい AWS リソースが不要
- 用途に応じて複数のキュー管理ができる
- DB ロックでパフォーマンス劣化しない
- ジョブのトランザクション内呼び出し対応

# 比較観点の一例

- 線形スケールアウト可能
- 新しい AWS リソースが不要
- 用途に応じて複数のキュー管理ができる
- DB ロックでパフォーマンス劣化しない
- ジョブのトランザクション内呼び出し対応
- Rails との親和性

# 比較観点の一例

- 線形スケールアウト可能**
- 新しい AWS リソースが不要**
- 用途に応じて複数のキュー管理ができる**
- DB ロックでパフォーマンス劣化しない**
- ジョブのトランザクション内呼び出し対応**
- Rails との親和性**
- 定期実行処理基盤がある**

# 比較観点の一例

- 線形スケールアウト可能**
- 新しい AWS リソースが不要**
- 用途に応じて複数のキュー管理ができる**
- DB ロックでパフォーマンス劣化しない**
- ジョブのトランザクション内呼び出し対応**
- Rails との親和性**
- 定期実行処理基盤がある**
- 無課金で使いたい**

# Sidekiq Pro

- 線形スケールアウト可能**
- 新しい AWS リソースが不要**
- 用途に応じて複数のキュー管理が可能**
- DB ロックでパフォーマンス劣化しない**
- ジョブのトランザクション内呼び出し対応**
- Rails との親和性**
- 定期実行処理基盤がある**
- 無課金で使いたい**

# Sidekiq Pro のいいところ

# Sidekiq Pro のいいところ

- ✓ めっちゃ強力 Batches 機能

# Sidekiq Pro のいいところ



めっちゃ強力 Batches 機能

たくさんのジョブ 1まとめ実行で、大量データも安全・安心処理

# Sidekiq Pro のいいところ

-  めっちゃ強力 Batches 機能

たくさんのジョブ 1まとめ実行で、大量データも安全・安心処理

-  ジョブが失敗しても心配不要！

# Sidekiq Pro のいいところ

## めっちゃ強力 Batches 機能

たくさんのジョブ 1まとめ実行で、大量データも安全・安心処理

## ジョブが失敗しても心配不要！

複数のジョブを一つのトランザクションとして扱える**Job Failures**

ジョブが失敗した時に自動的にリトライする**Reliable Queues**

失敗しても安心設計だぜ！

# Sidekiq Pro のいいところ

## めっちゃ強力 Batches 機能

たくさんのジョブ 1まとめ実行で、大量データも安全・安心処理

## ジョブが失敗しても心配不要！

複数のジョブを一つのトランザクションとして扱える**Job Failures**

ジョブが失敗した時に自動的にリトライする**Reliable Queues**

失敗しても安心設計だぜ！

## 最強のスケジューリング機能

# Sidekiq Pro のいいところ

## めっちゃ強力 Batches 機能

たくさんのジョブ 1まとめ実行で、大量データも安全・安心処理

## ジョブが失敗しても心配不要！

複数のジョブを一つのトランザクションとして扱える**Job Failures**

ジョブが失敗した時に自動的にリトライする**Reliable Queues**

失敗しても安心設計だぜ！

## 最強のスケジューリング機能

ジョブの実行順序を細かく制御できる **Unique Jobs**

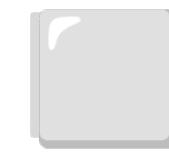
特定の時間までジョブの実行を遅らせる**Scheduled Jobs**

# 自社事情をふまえた Sidekiq Pro の評価

# 自社事情をふまえた Sidekiq Pro の評価

- 新しい AWS リソースが必要

# 自社事情をふまえた Sidekiq Pro の評価



新しい AWS リソースが必要

**Valkey or Redis が必要**

# 自社事情をふまえた Sidekiq Pro の評価

- 新しい AWS リソースが必要  
**Valkey or Redis が必要**
- 秘めたる力を完璧に使いこなすには大量にある ActiveJob バッチの書き換えがいる

# 自社事情をふまえた Sidekiq Pro の評価

- 新しい AWS リソースが必要  
**Valkey or Redis が必要**
- 秘めたる力を完璧に使いこなすには大量にある ActiveJob バッチの書き換えがいる  
**ActiveJob だと全ての機能を使いきれない**

# 自社事情をふまえた Sidekiq Pro の評価

- 新しい AWS リソースが必要  
**Valkey or Redis が必要**
- 秘めたる力を完璧に使いこなすには大量にある ActiveJob バッチの書き換えがいる  
**ActiveJob だと全ての機能を使いきれない**
- お金!

# 自社事情をふまえた Sidekiq Pro の評価

- 新しい AWS リソースが必要  
**Valkey or Redis が必要**
- 秘めたる力を完璧に使いこなすには大量にある ActiveJob バッチの書き換えがいる  
**ActiveJob だと全ての機能を使いきれない**
- お金!  
**有料!**

# SolidQueue

-  **用途に応じて複数のキュー管理が可能**
-  **新しい AWS リソースが不要**
-  **線形スケールアウト可能**
-  **DB ロックでパフォーマンス劣化しない**
-  **ジョブのトランザクション内呼び出し対応**
-  **Rails との親和性**
-  **定期実行処理基盤がある**
-  **無課金で使いたい**

# **Solid Queue のいいところ . 1**

# **Solid Queue のいいところ . 1**

線形スケールアウト大得意

# Solid Queue のいいところ . 1

線形スケールアウト大得意

ジョブを実行する Worker を増やせば増やすほどパフォーマンスが上がる。

ジョブ取得の仕組みがロックを最小限するような安心設計

# Solid Queue のいいところ . 1

線形スケールアウト大得意

ジョブを実行する Worker を増やせば増やすほどパフォーマンスが上がる。

ジョブ取得の仕組みがロックを最小限するような安心設計

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

# Solid Queue のいいところ . 1

線形スケールアウト大得意

ジョブを実行する Worker を増やせば増やすほどパフォーマンスが上がる。

ジョブ取得の仕組みがロックを最小限するような安心設計

## パフォーマンスとスケーラビリティの向上

-  綺麗に線形にスケールアウトできるようにしたい
-  処理できるジョブの数を増やしたい
-  リソースを有効活用したい

# Solid Queue のいいところ . 1

## ✓ 線形スケールアウト大得意

ジョブを実行する Worker を増やせば増やすほどパフォーマンスが上がる。

ジョブ取得の仕組みがロックを最小限するような安心設計

## パフォーマンスとスケーラビリティの向上

- ✓ 綺麗に線形にスケールアウトできるようにしたい
- ✓ 処理できるジョブの数を増やしたい
- ✓ リソースを有効活用したい

# **Solid Queue のいいところ . 2**

# Solid Queue のいいところ . 2

- ✓ 運用・インフラストラクチャの効率化

# Solid Queue のいいところ . 2

✓ 運用・インフラストラクチャの効率化

既存の Aurora インスタンスに新しく `create database` するだけ

新しいインフラリソースが不要

# Solid Queue のいいところ . 2

- ✓ 運用・インフラストラクチャの効率化

既存の Aurora インスタンスに新しく `create database` するだけ

新しいインフラリソースが不要

持ち物が増えない

# **Solid Queue のいいところ . 3**

# Solid Queue のいいところ . 3

ActiveJob バックエンドだぜ

# Solid Queue のいいところ . 3

ActiveJob バックエンドだぜ

**Rails チームが開発している非同期バックエンド!**

# Solid Queue のいいところ . 3

 ActiveJob バックエンドだぜ

**Rails チームが開発している非同期バックエンド!**

# Solid Queue の物足りないところ

# Solid Queue の物足りないところ



監視機能がないんだぜ

# Solid Queue の物足りないところ



監視機能がないんだぜ

自前でなんとかしろスタイル

# とりあえず見た感じやりたいこと全部解決してくれそう

## パフォーマンスとスケーラビリティの向上

-  綺麗に線形にスケールアウトできるようにしたい
-  処理できるジョブの数を増やしたい
-  リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

-  優先度を定義したら優先度を守ってほしい
-  ジョブごとにキューと Worker を分けたい
-  定期実行処理をまとめたい

## 開発と運用の効率化

-  ActiveJob に移行したい（Delayed 専用の書き方と ActiveJob の混在廃止）
-  できるだけ Rails 標準機能に寄せたい

# **Solid Queue の内部実装の理解**

# **Solid Queue の内部実装の理解**

どうして必要？

# Solid Queue の内部実装の理解

どうして必要？

監視

パフォーマンス・チューニング

障害対応とトラブルシューティング

# Solid Queue の内部実装の理解

どうして必要？

## 監視

DB 構成を把握し、クエリを投げてメトリクス取得する

## パフォーマンス・チューニング

## 障害対応とトラブルシューティング

# Solid Queue の内部実装の理解

どうして必要？

## 監視

DB 構成を把握し、クエリを投げてメトリクス取得する

## パフォーマンス・チューニング

ジョブの処理するフローを理解しワーカープロセス数などの調節

## 障害対応とトラブルシューティング

# Solid Queue の内部実装の理解

どうして必要？

## 監視

DB 構成を把握し、クエリを投げてメトリクス取得する

## パフォーマンス・チューニング

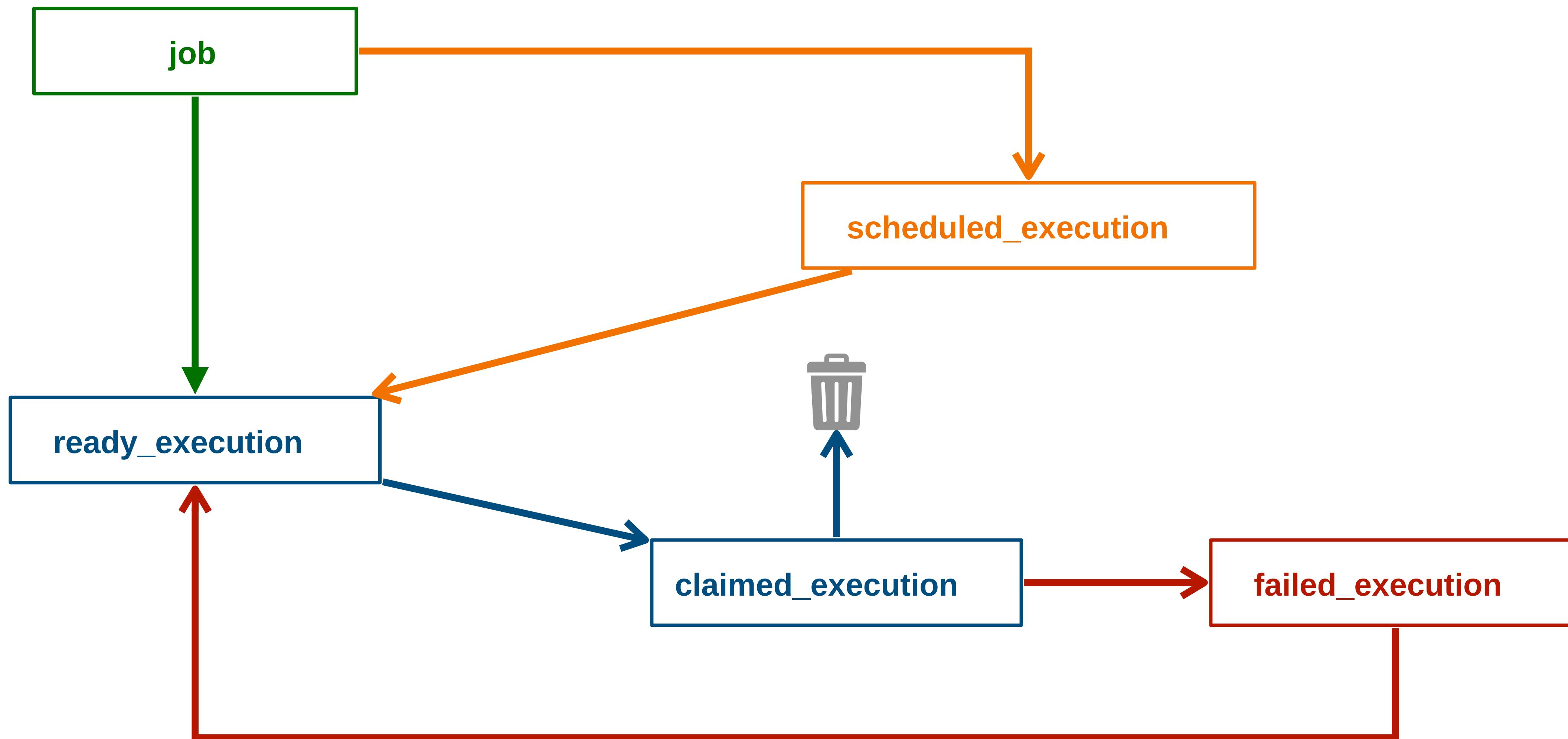
ジョブの処理するフローを理解しワーカープロセス数などの調節

## 障害対応とトラブルシューティング

DB 構成とジョブ処理フローを理解しトラブルを未然に防いだり  
発生時に素早く対応できるようにしておく

# Solid Queue の内部実装の理解

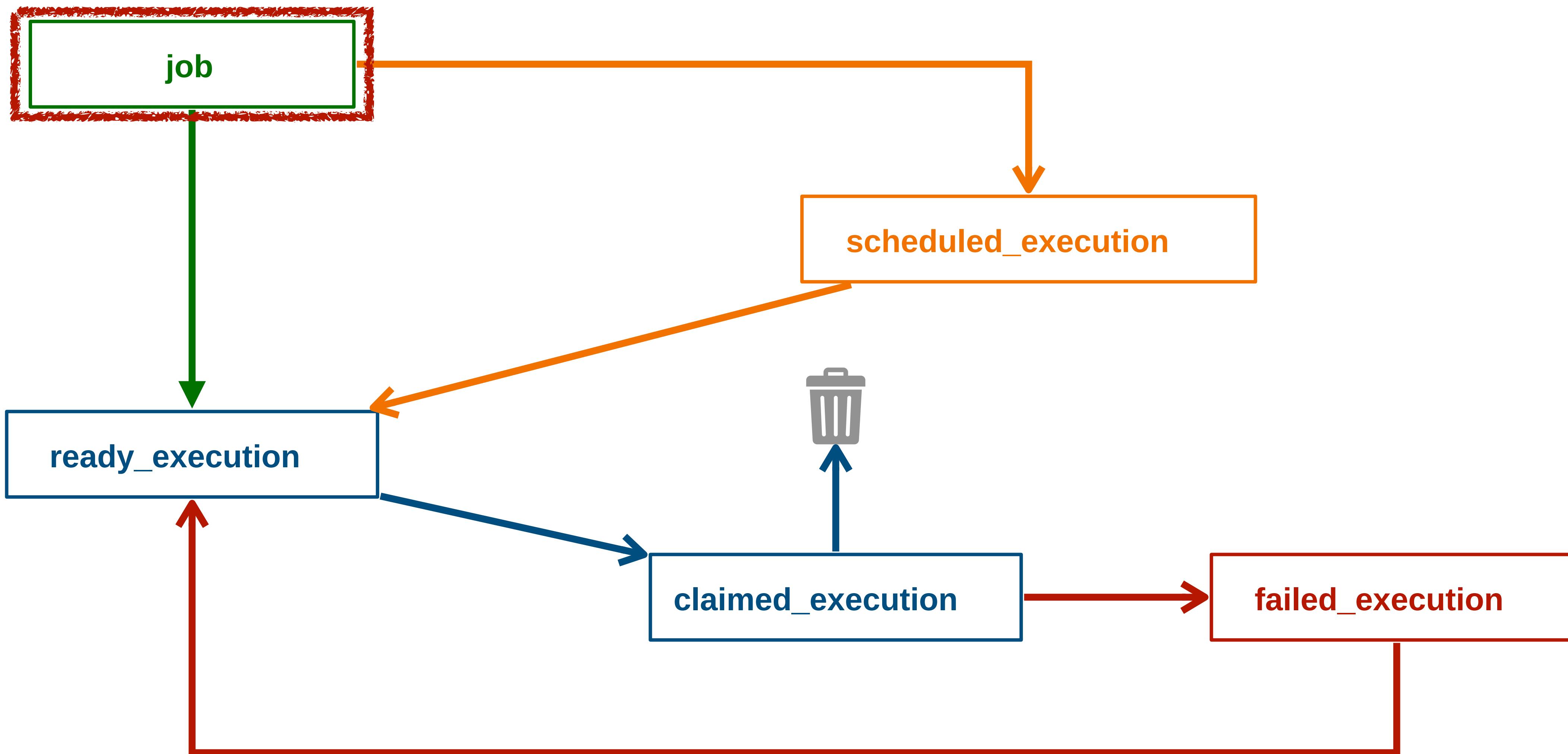
全体図（間に色々あるけど主要なものだけ）



# Solid Queue の内部実装の理解

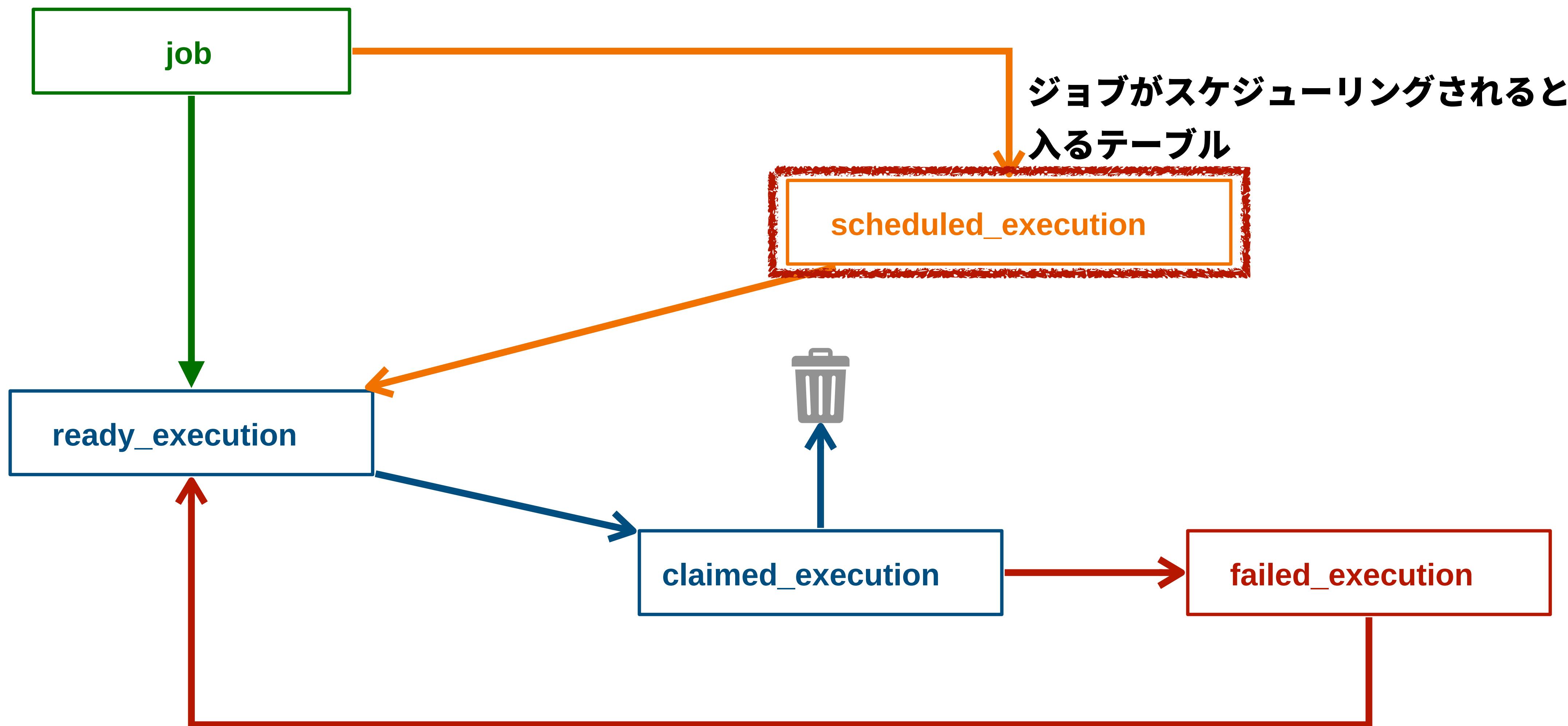
全体図（間に色々あるけど主要なものだけ）

ジョブがエンキューされると入るテーブル



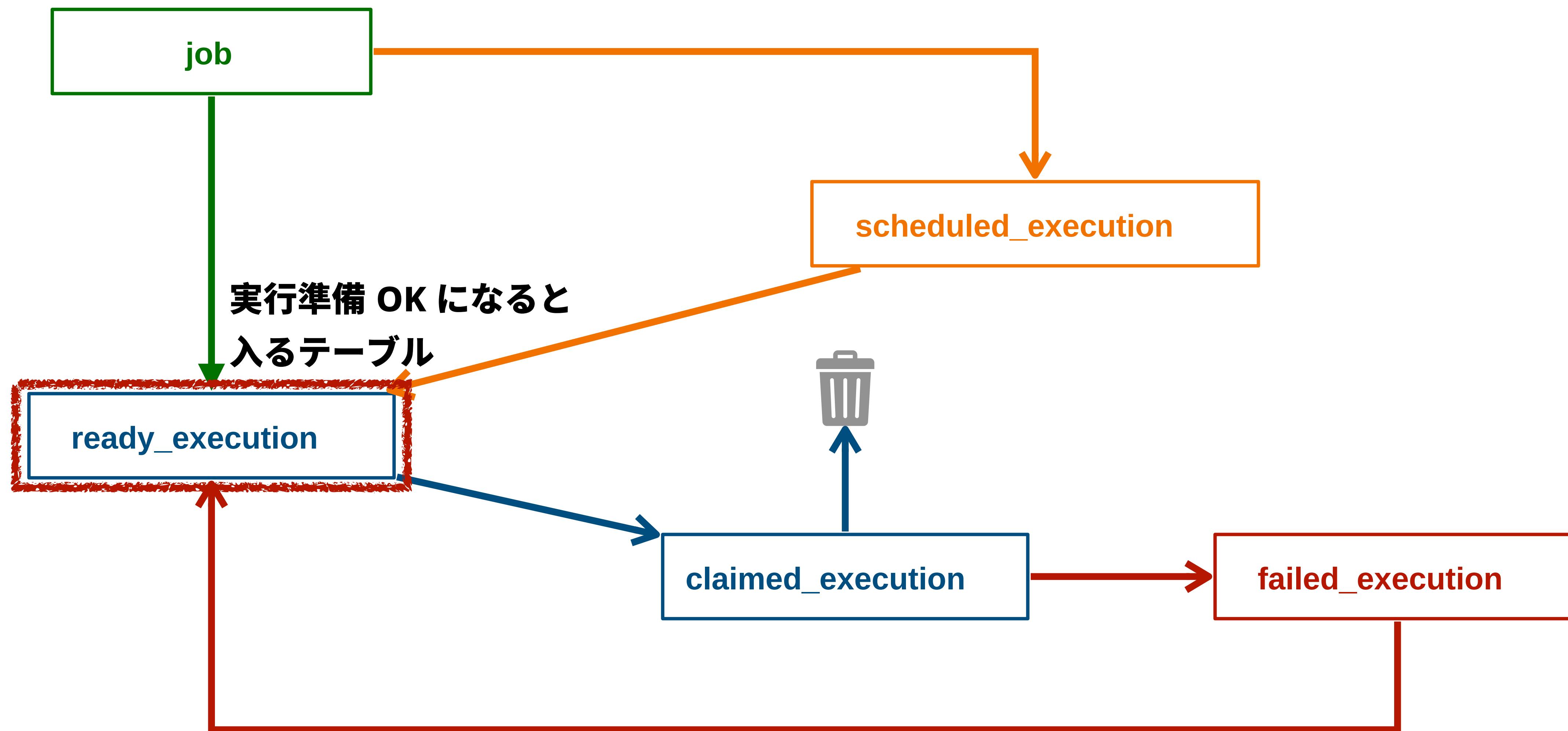
# Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なものだけ）



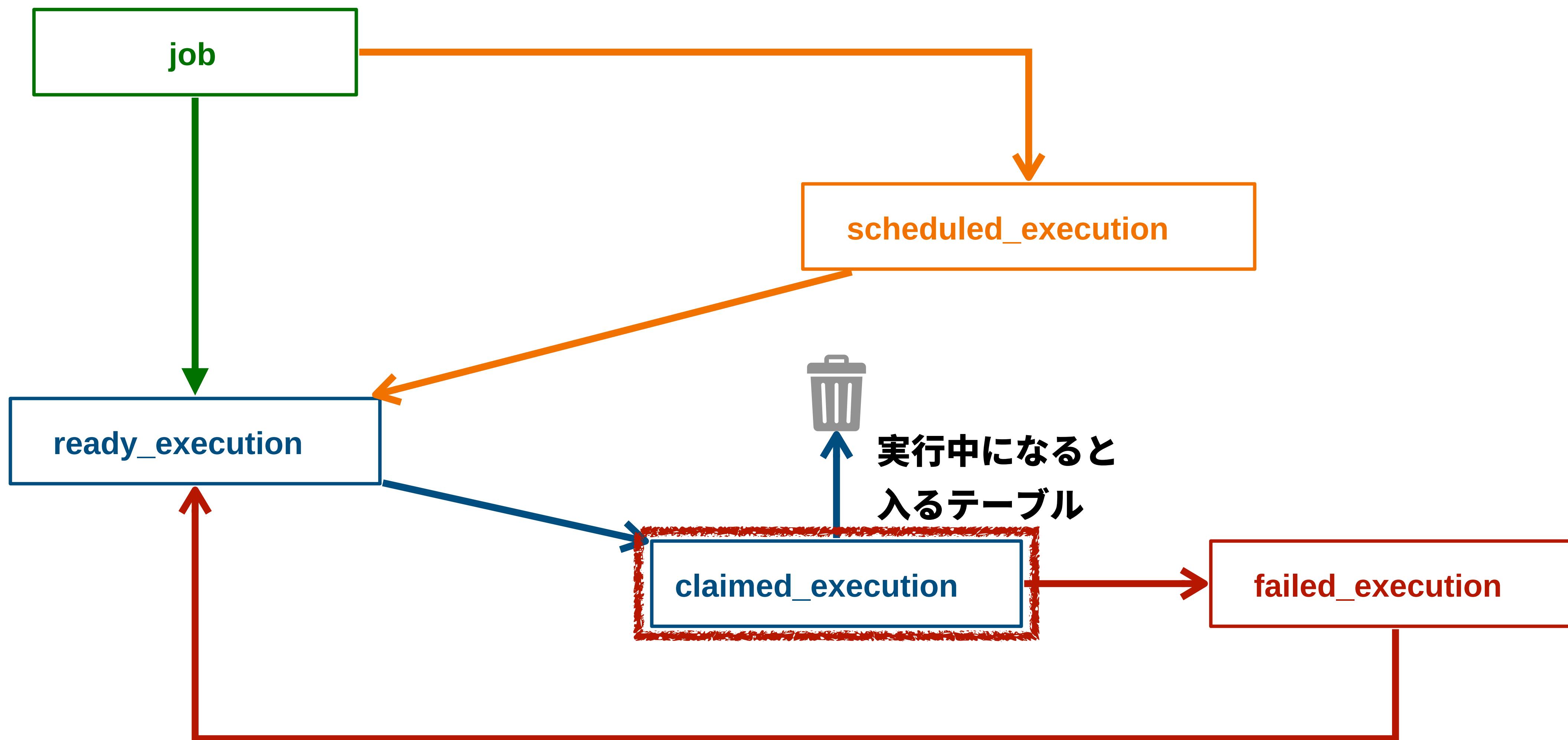
# Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なもののだけ）



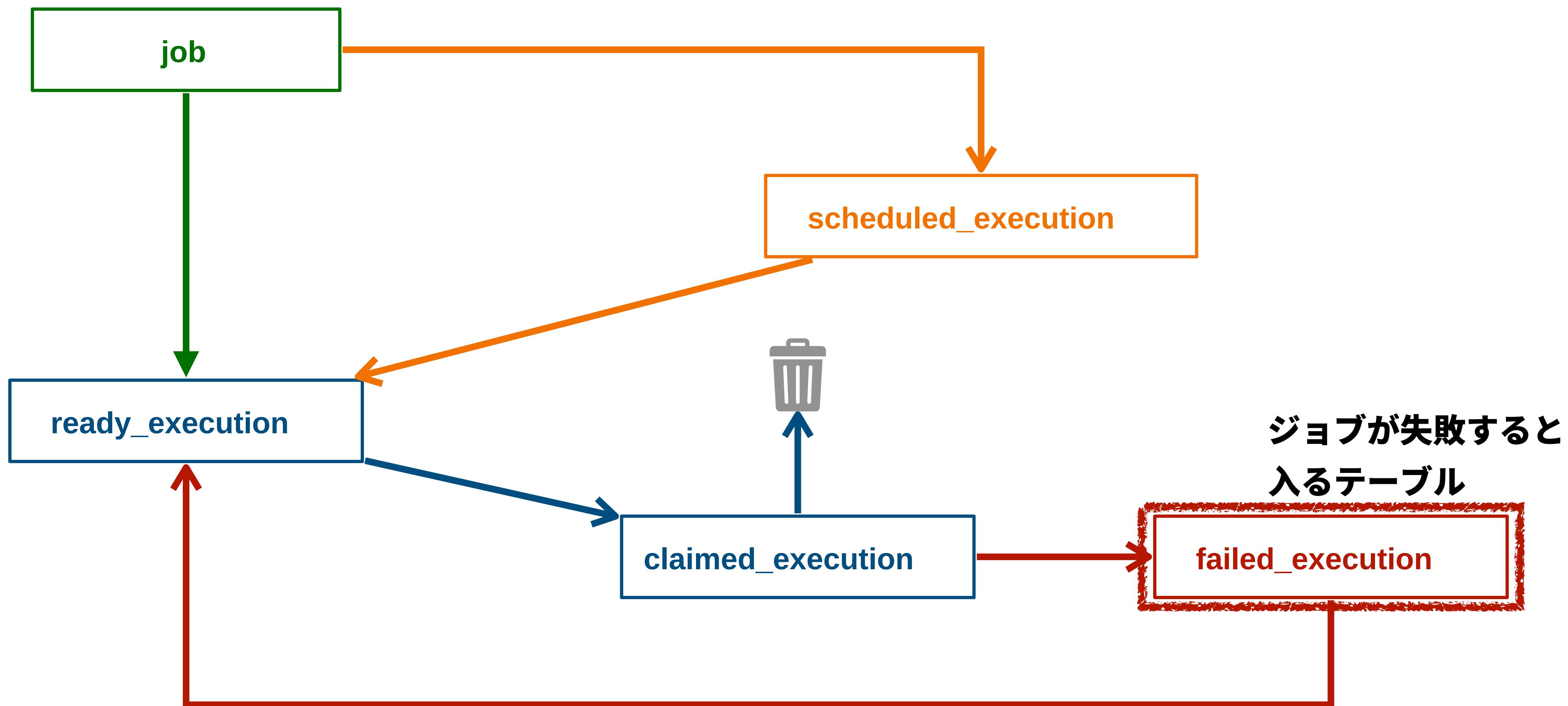
# Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なもののだけ）



# Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なものだけ）



# Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合

job

scheduled\_at

scheduled\_execution

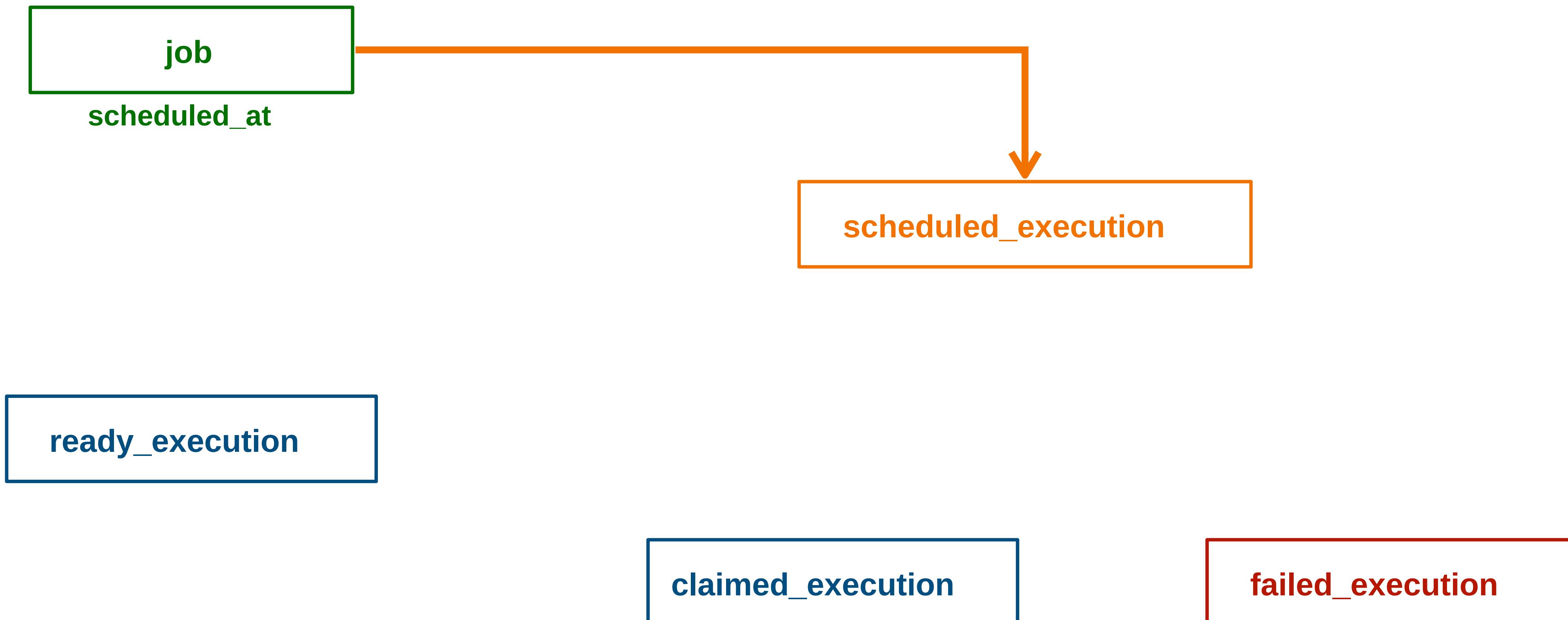
ready\_execution

claimed\_execution

failed\_execution

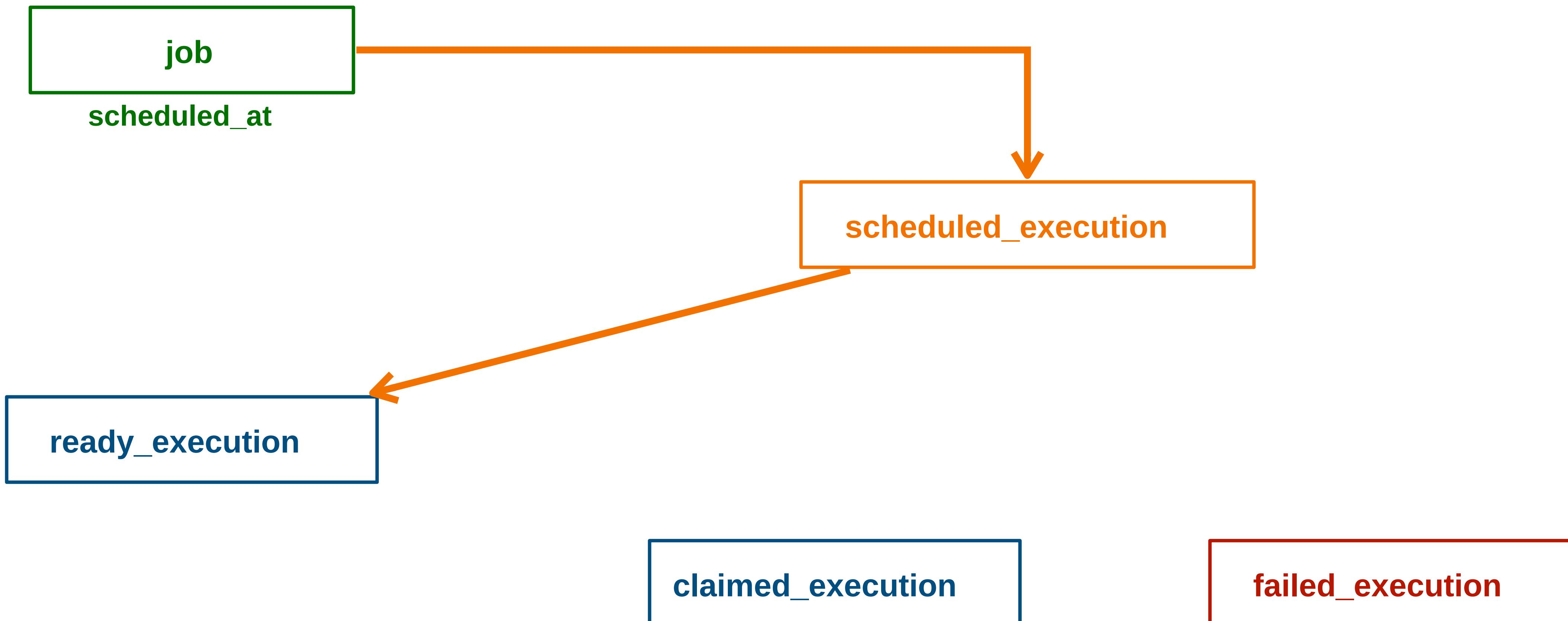
# Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



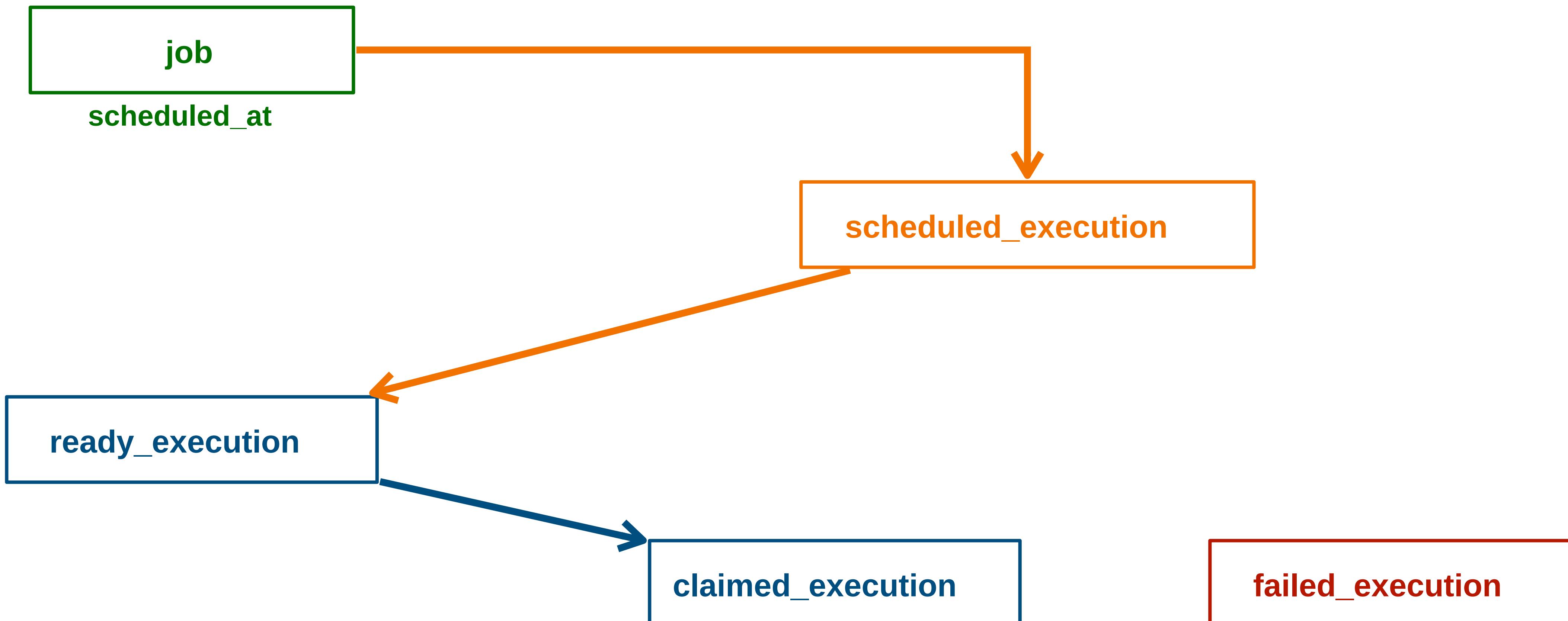
# Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



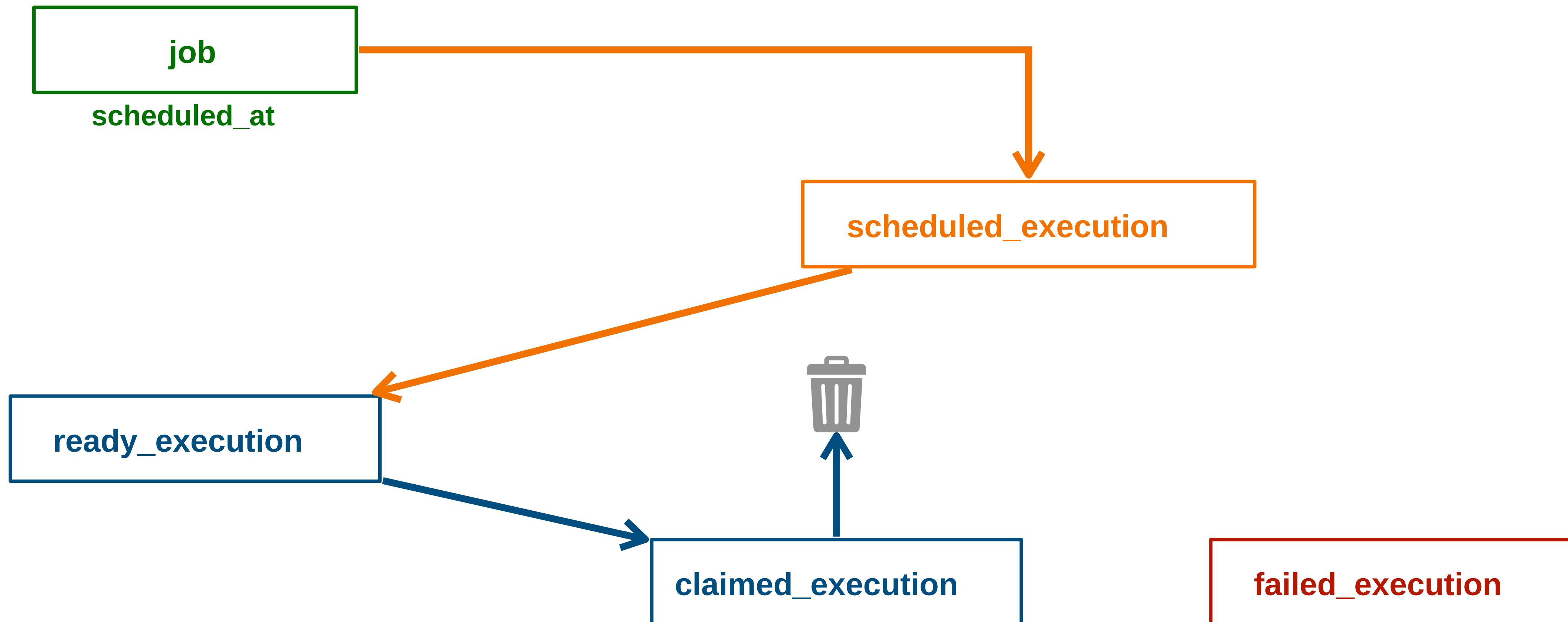
# Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



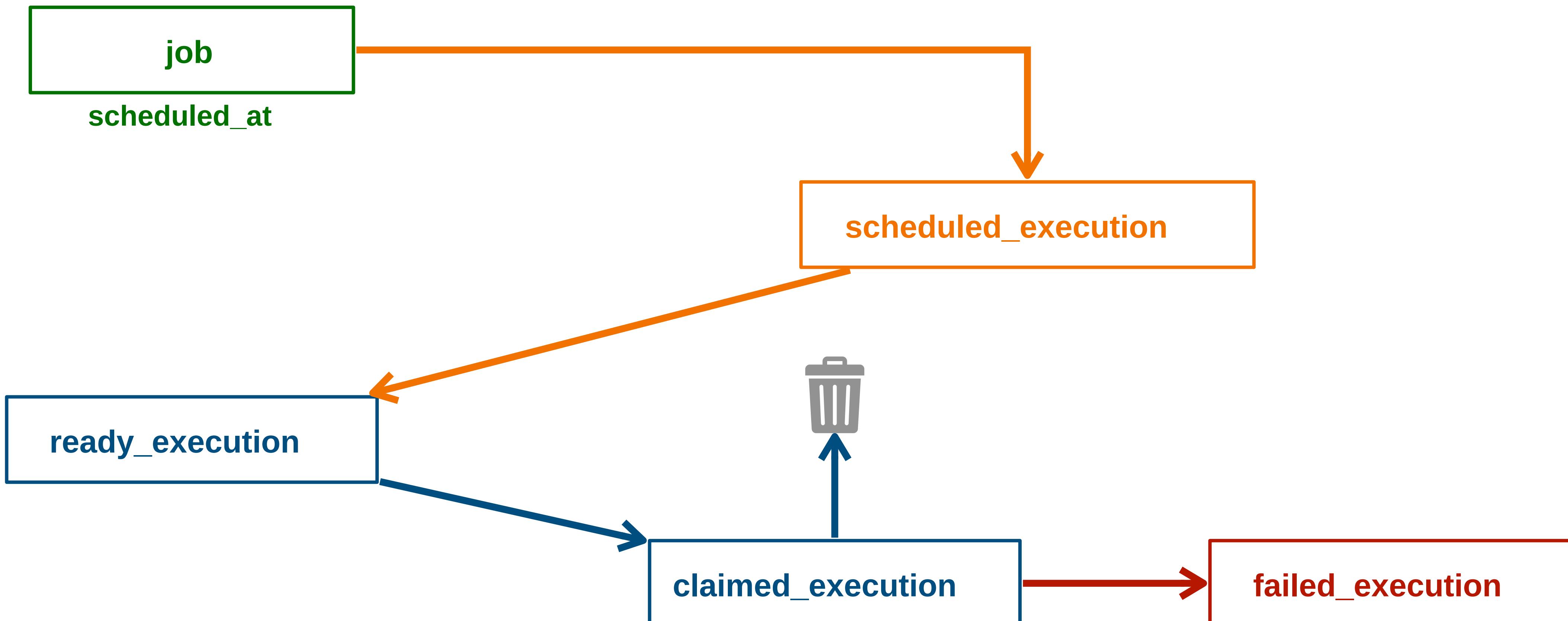
# Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



# Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



# Solid Queue の内部実装の理解

すぐ実行したいジョブの場合

job

scheduled\_execution

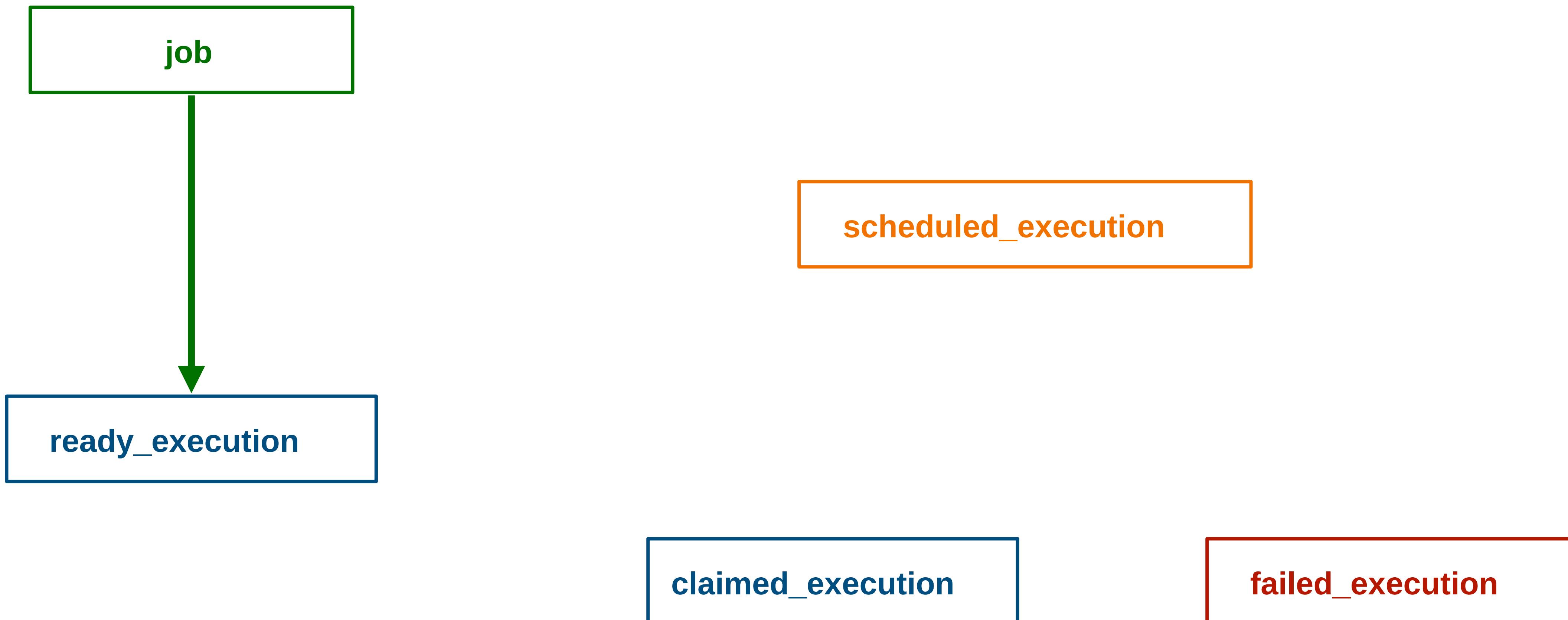
ready\_execution

claimed\_execution

failed\_execution

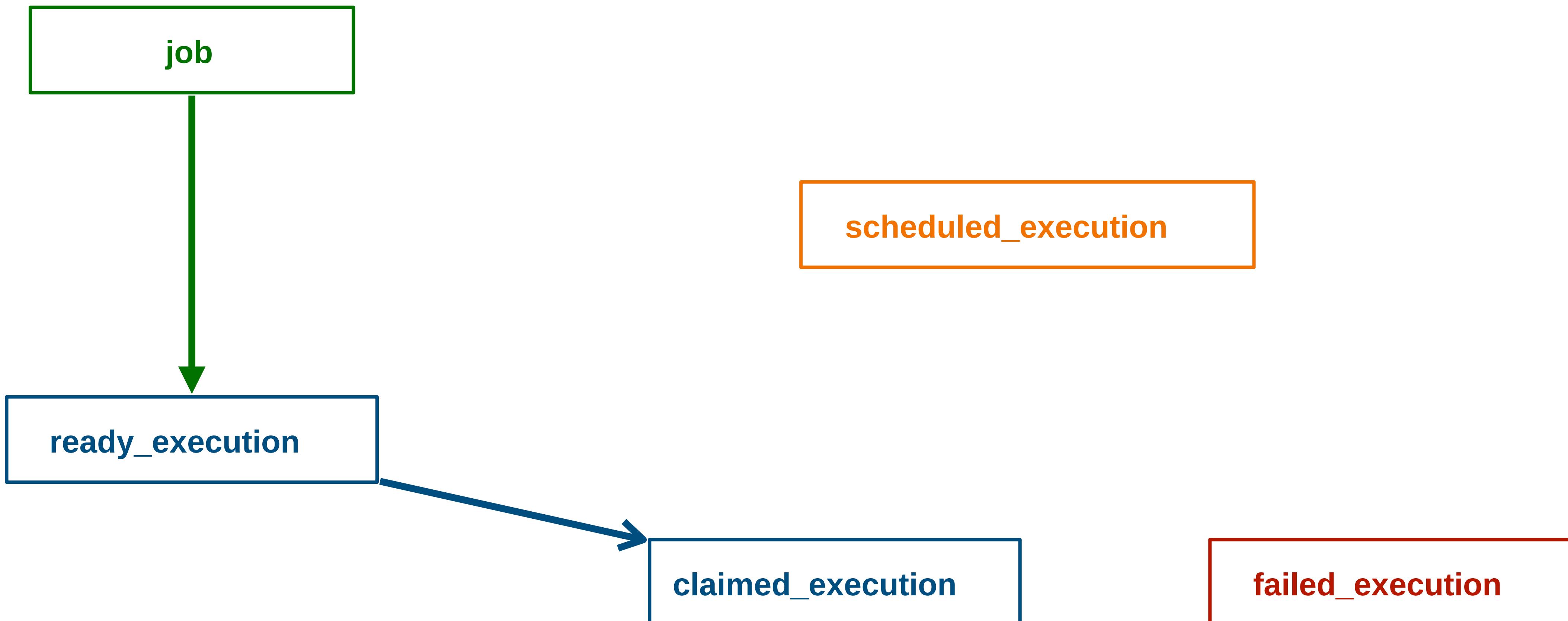
# Solid Queue の内部実装の理解

すぐ実行したいジョブの場合



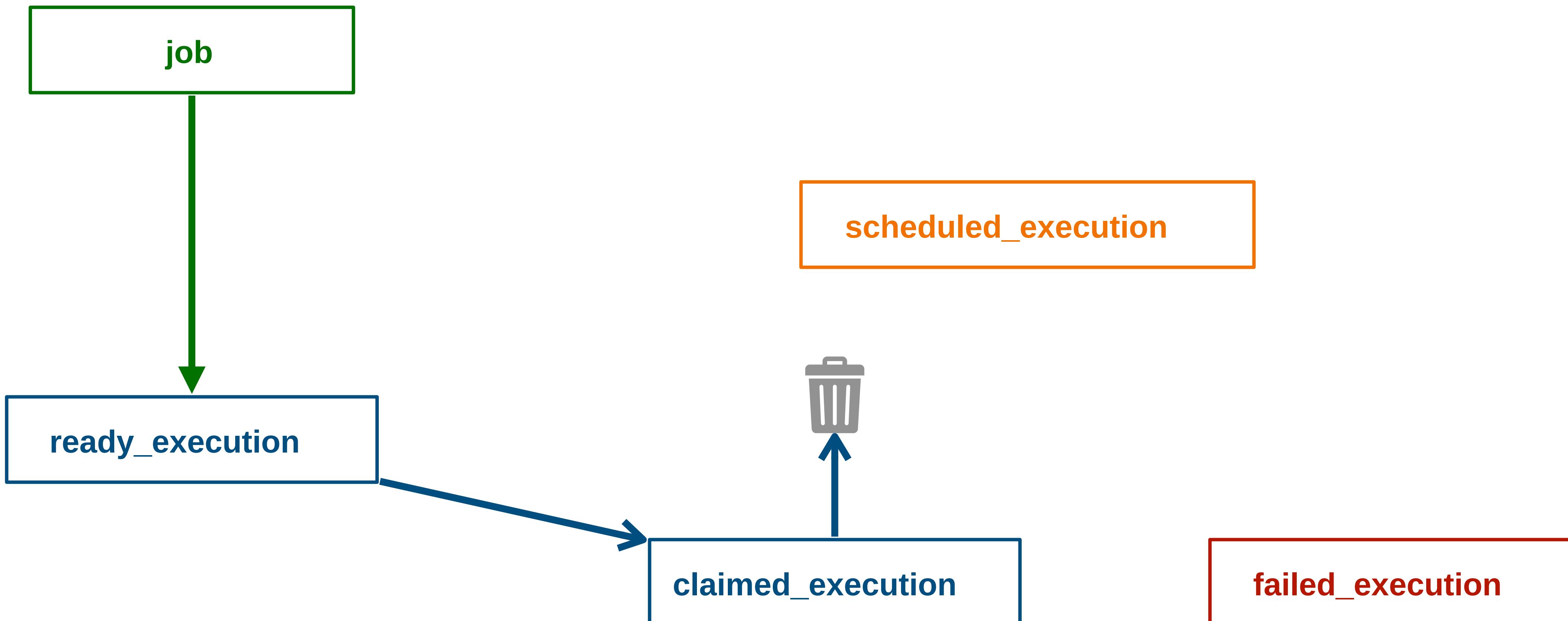
# Solid Queue の内部実装の理解

すぐ実行したいジョブの場合



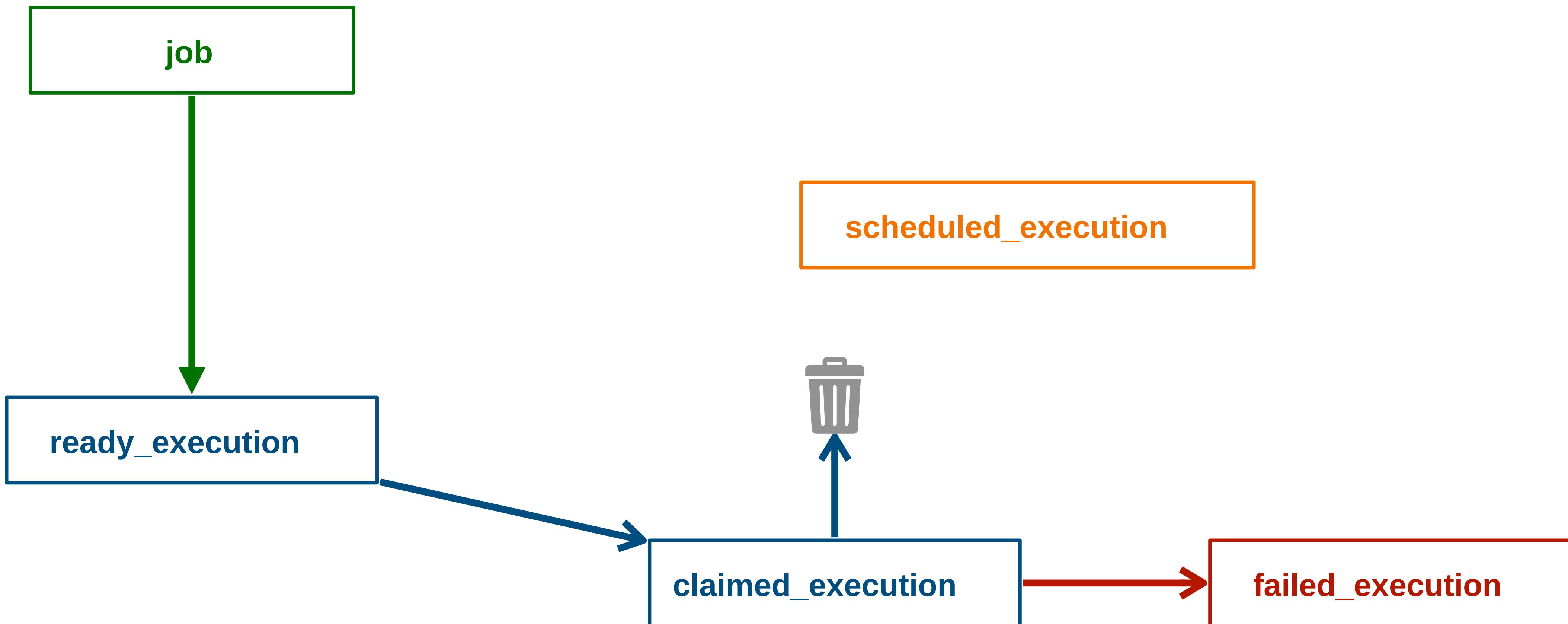
# Solid Queue の内部実装の理解

すぐ実行したいジョブの場合

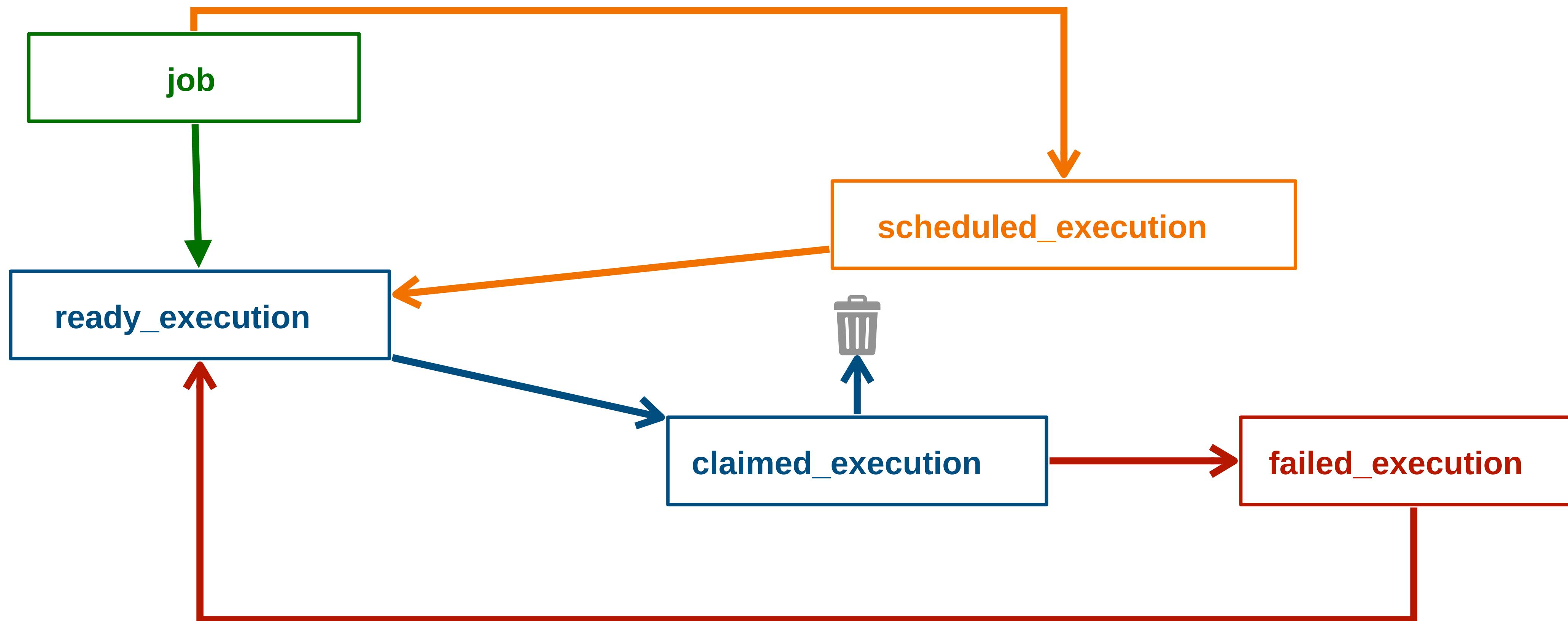


# Solid Queue の内部実装の理解

すぐ実行したいジョブの場合

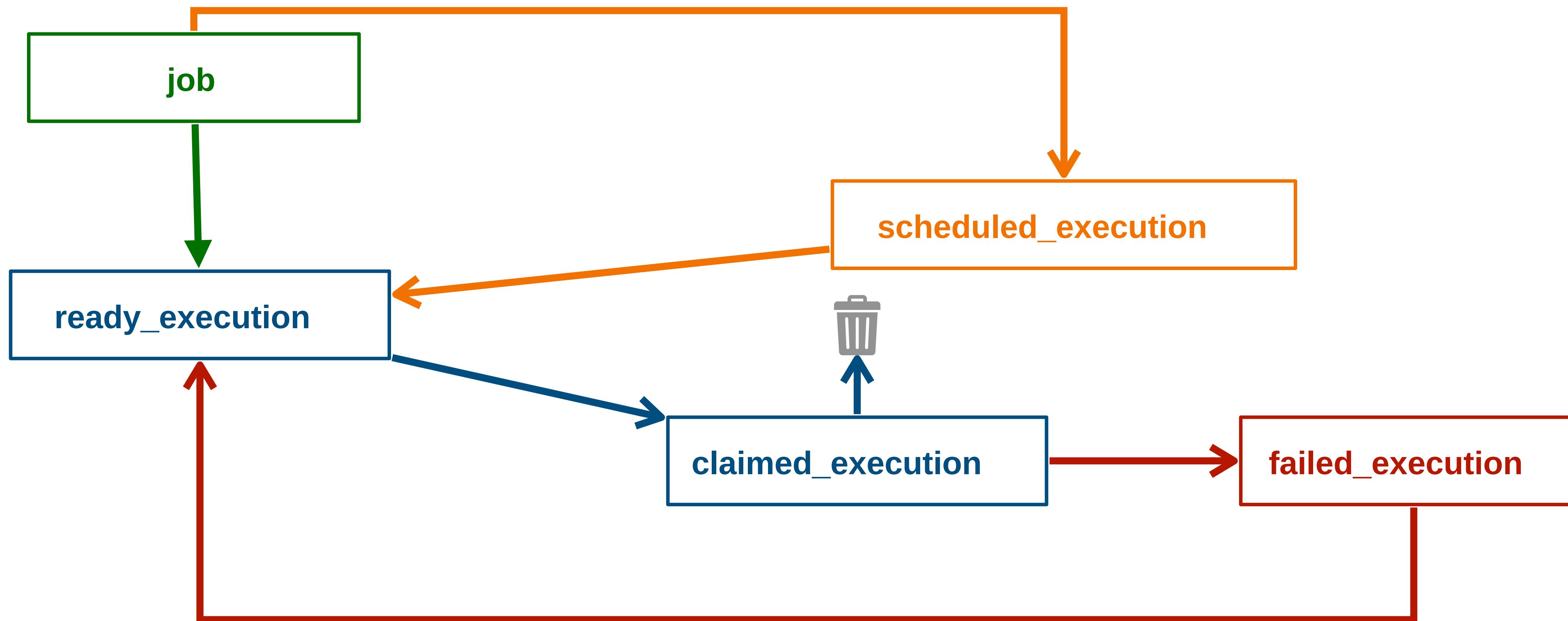


# Solid Queue の内部実装の理解



# Solid Queue の内部実装の理解

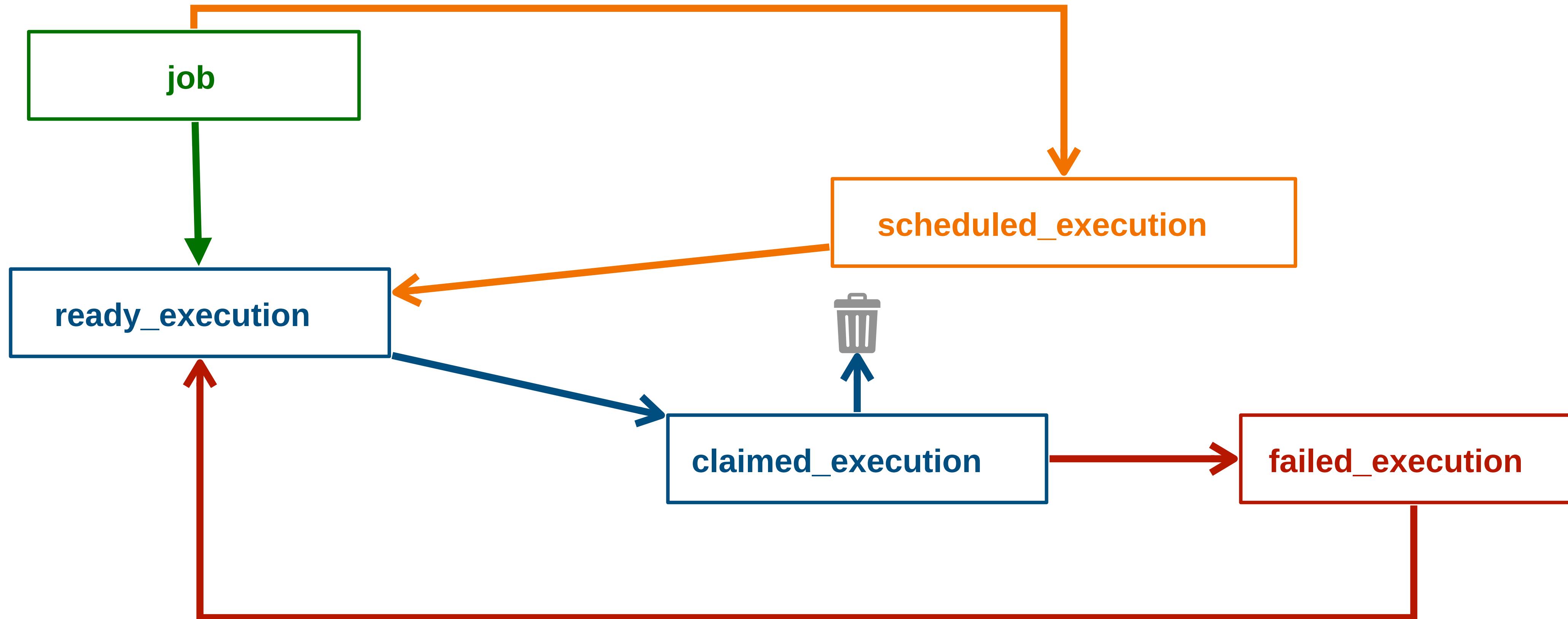
監視時はそれぞれのテーブルを頭に入れてクエリを書く



# Solid Queue の内部実装の理解

監視時はそれぞれのテーブルを頭に入れてクエリを書く

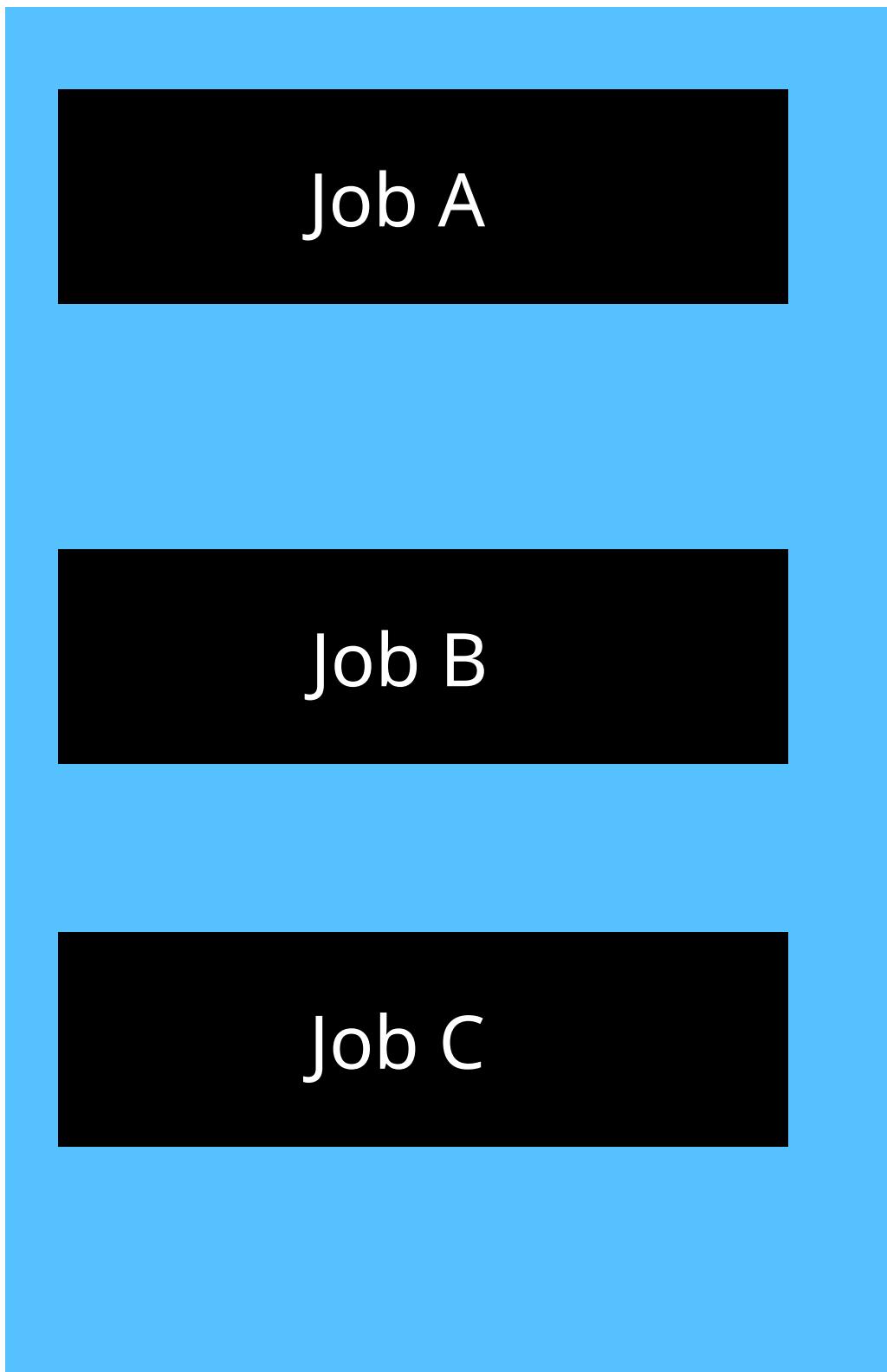
トラブル時はどこに実行したジョブのデータが入っているかで現状がわかる



# **Solid Queue で解決する線形スケールアウト**

# Solid Queue で解決する線形スケールアウト

Job Table

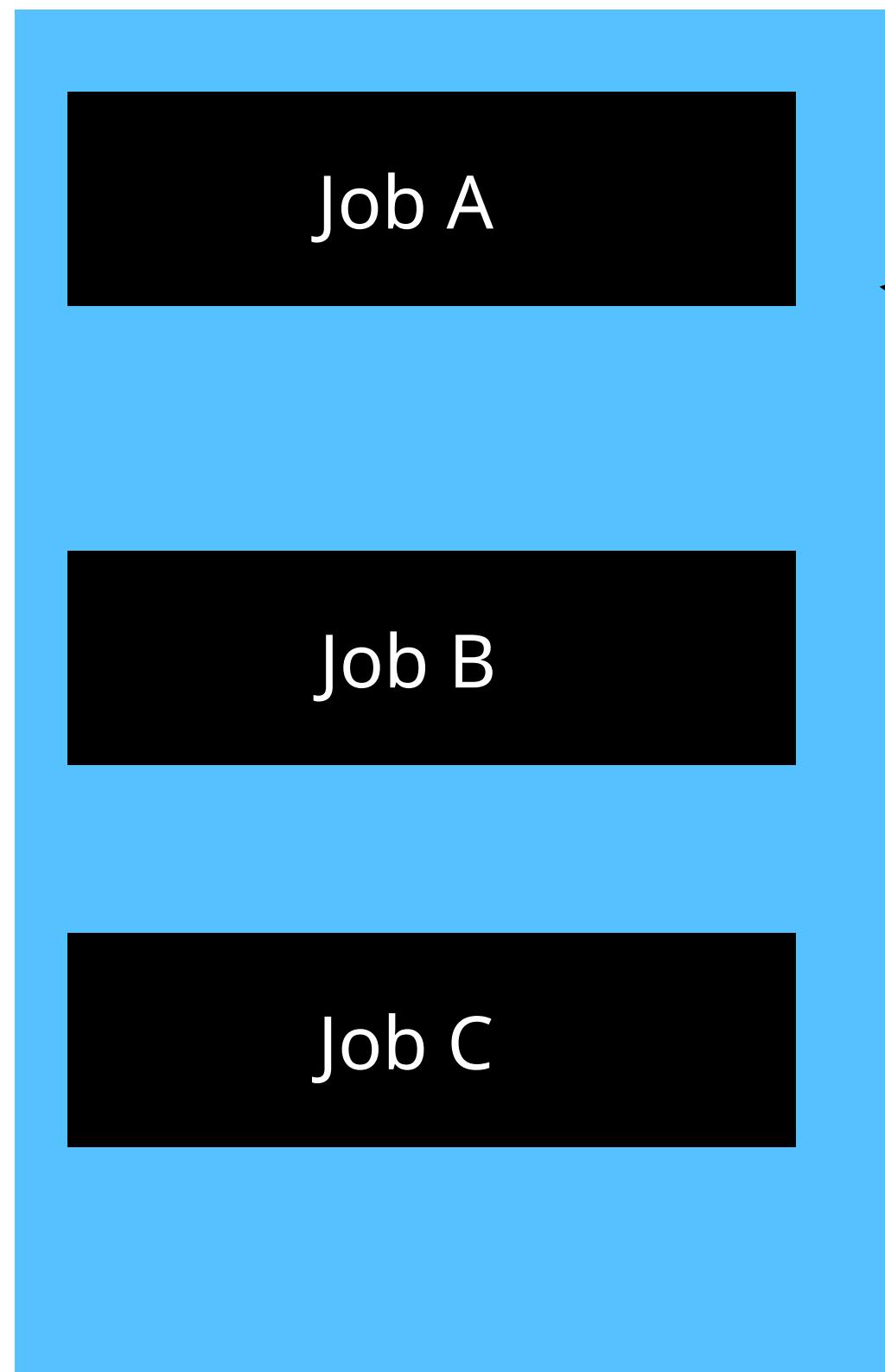


Worker A

Worker B

# Solid Queue で解決する線形スケールアウト

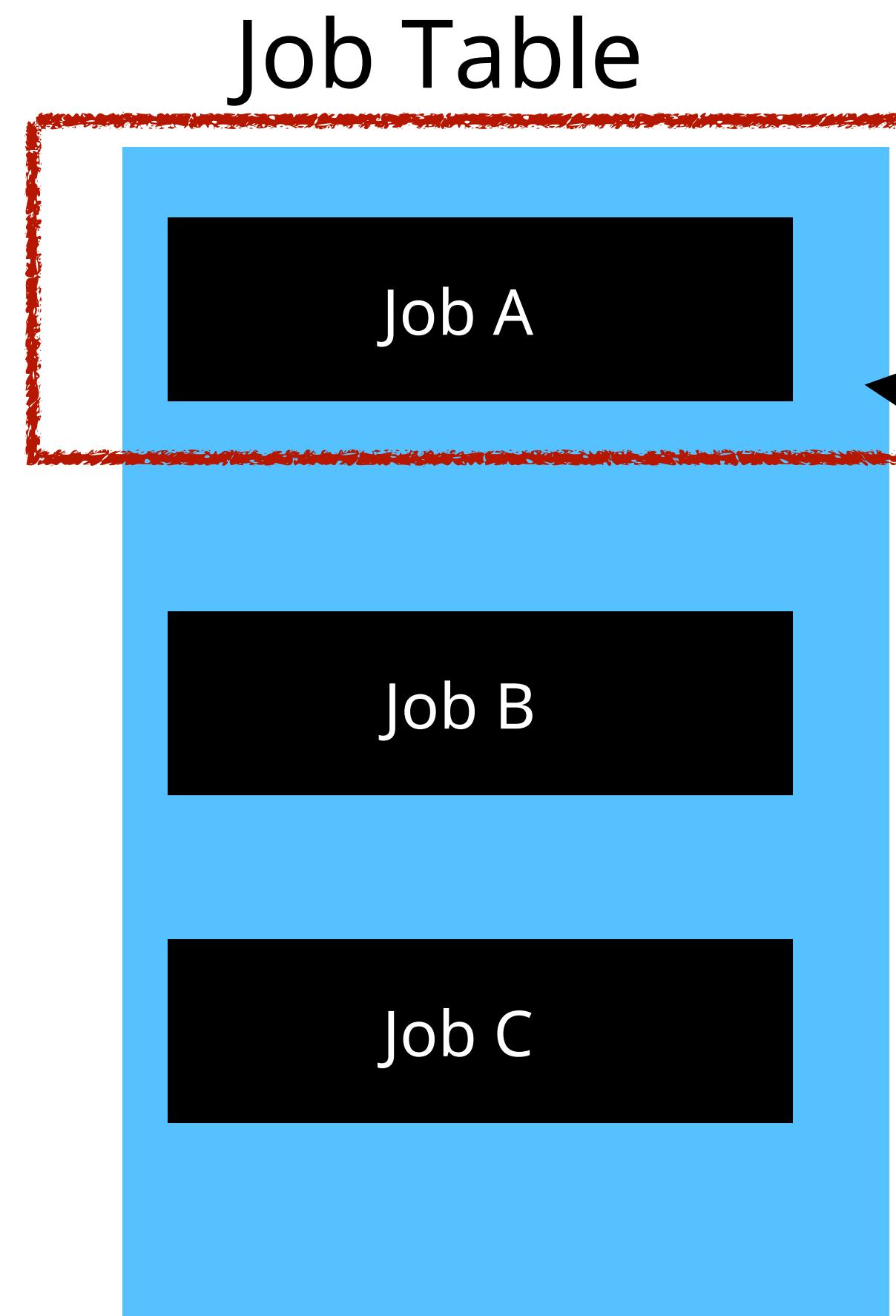
Job Table



「空いてるジョブを 1 つロックして、俺にくれ！」



# Solid Queue で解決する線形スケールアウト



```
START TRANSACTION;  
SELECT * FROM jobstable FOR UPDATE SKIP LOCKED LIMIT 1;
```

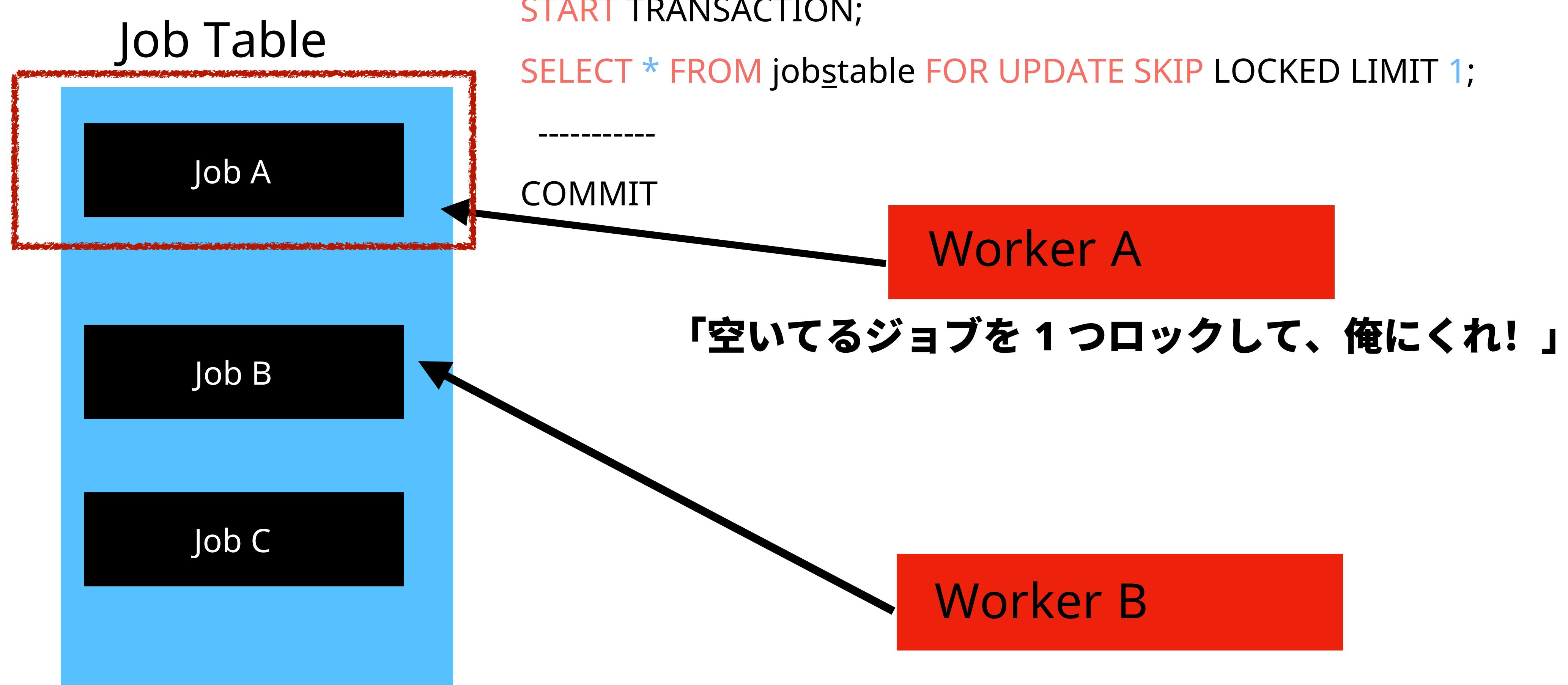
-----  
COMMIT

Worker A

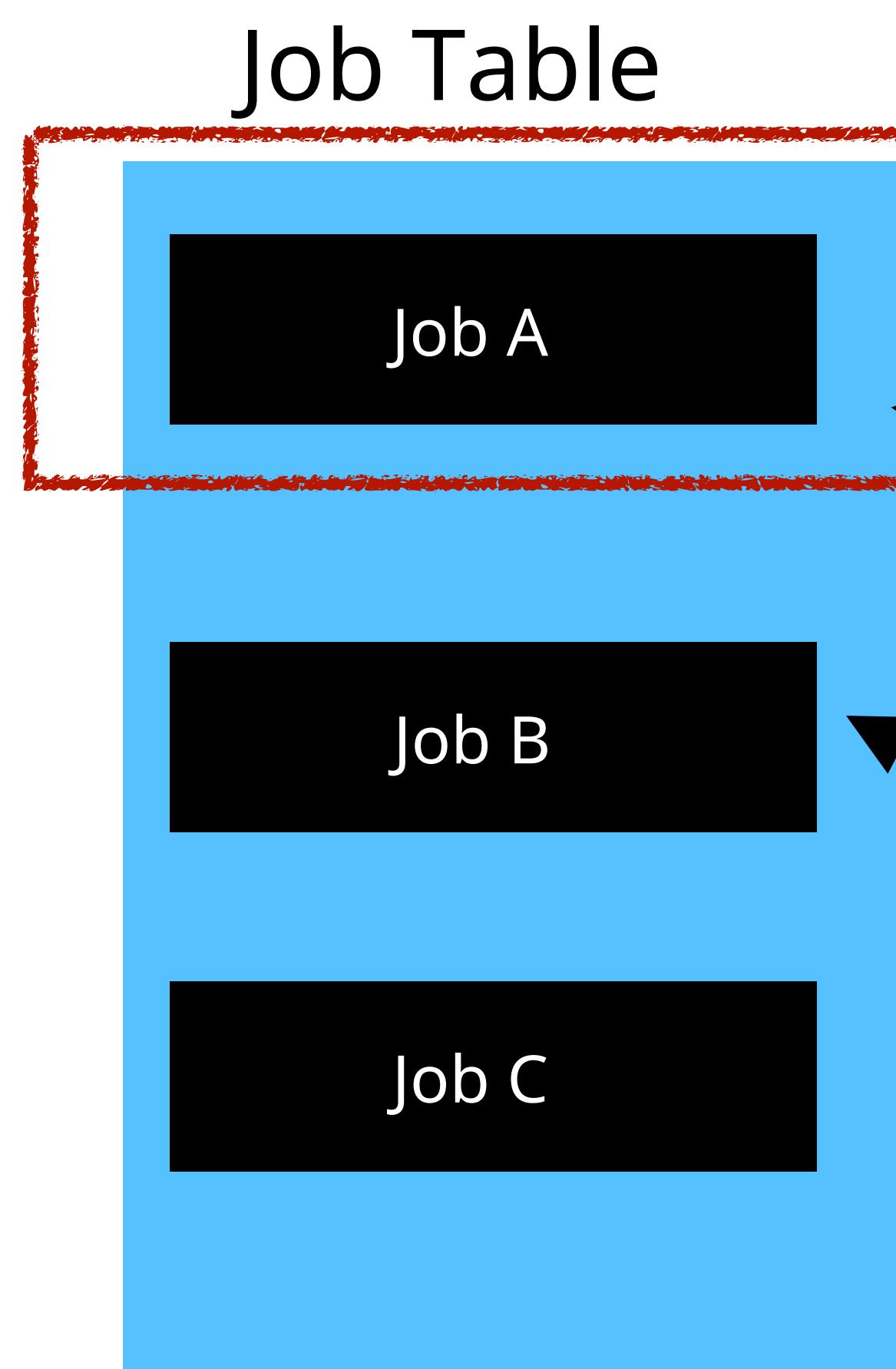
「空いてるジョブを 1 つロックして、俺にくれ！」

Worker B

# Solid Queue で解決する線形スケールアウト



# Solid Queue で解決する線形スケールアウト



```
START TRANSACTION;  
SELECT * FROM jobstable FOR UPDATE SKIP LOCKED LIMIT 1;
```

-----  
COMMIT

Worker A

「空いてるジョブを 1 つロックして、俺にくれ！」

Worker B

「空いてるジョブを 1 つロックして、俺にくれ！  
ただし、誰かがロックしているジョブなら無視していいぜ！」

# Solid Queue の線形的なスケールアウト

本（処理）がたくさん売れる（入る）から

店員（処理プロセス）を1人から2人とした

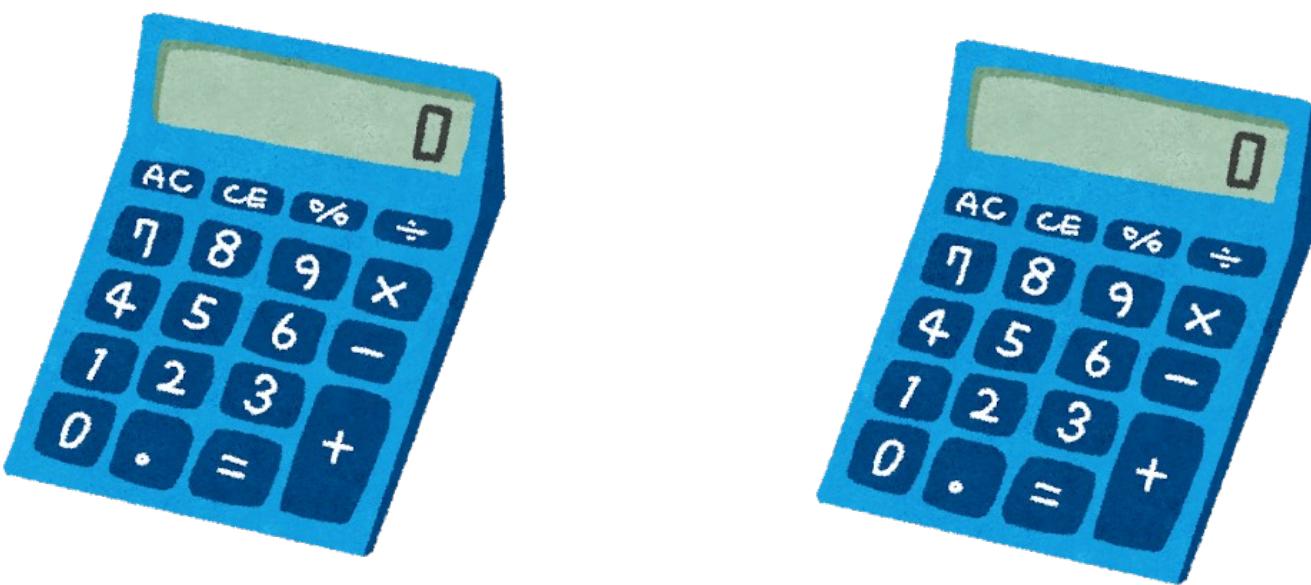


# Solid Queue の線形的なスケールアウト

本（処理）がたくさん売れる（入る）から  
店員（処理プロセス）を 1 人から 2 人にした



電卓を二つ用意して、お客様は  
空いている方に行ってもらうよ

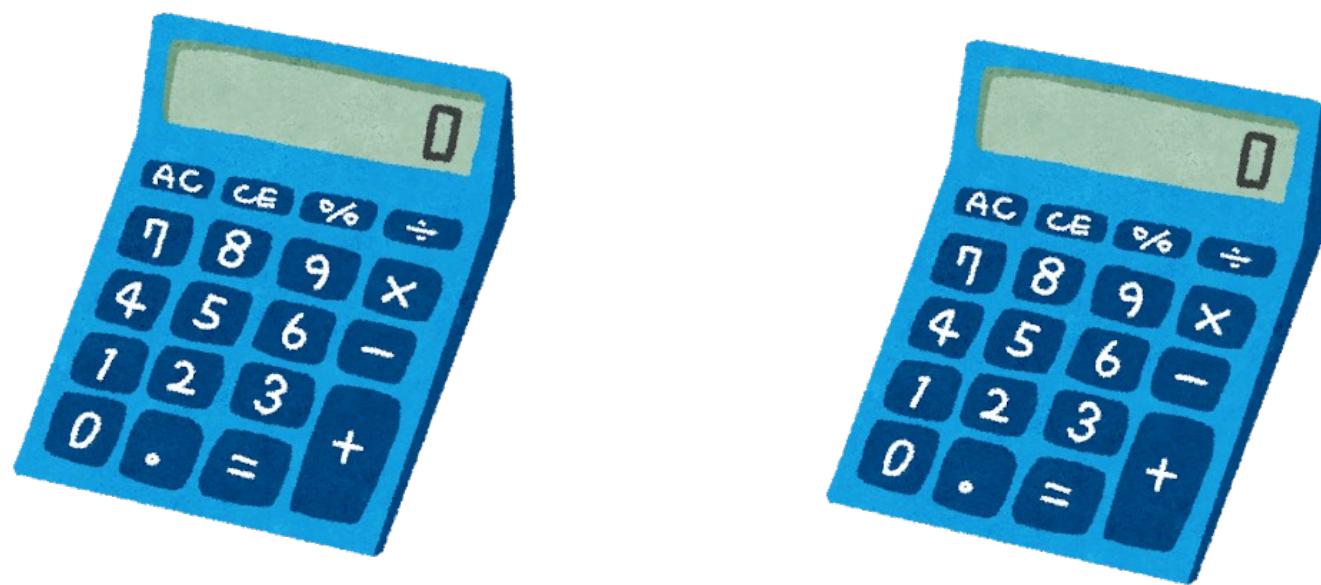


# Solid Queue の線形的なスケールアウト

本（処理）がたくさん売れる（入る）から  
店員（処理プロセス）を 1 人から 2 人にした



電卓を二つ用意して、お客様は  
空いている方に行ってもらうよ



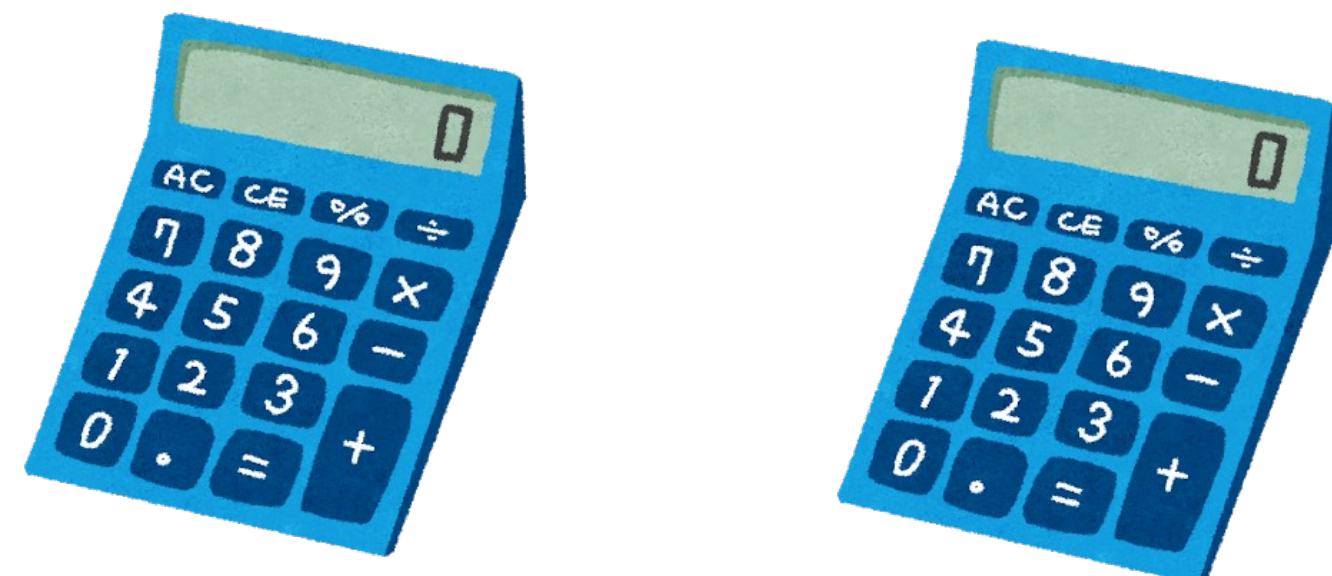
待ち時間ロックされないのでお客様を捌くスピードが…

# Solid Queue の線形的なスケールアウト

本（処理）がたくさん売れる（入る）から  
店員（処理プロセス）を 1 人から 2 人にした



電卓を二つ用意して、お客様は  
空いている方に行ってもらうよ



待ち時間ロックされないのでお客様を捌くスピードが…

倍だぜ！

- はじめに: Teachme Biz について
- 課題 Part その 1
- 選定
- **設計と実装そして移行**
- 課題 Part その 2
- まとめ

# 移行手順

- Delayed 固有メソッドの撲滅
- ActiveJob で明示的に Delayed を指定
- SolidQueue 設定
- 監視
- SolidQueue を利用するように全 Job を変更
- gem uninstall delayed

# 移行手順

-  **Delayed 固有メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

TestClass.delay.send\_mail

delayed\_jobs Table

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

TestClass.delay.send\_mail

delayed\_jobs Table

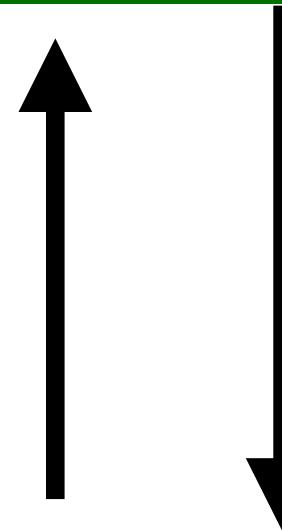


# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

TestClass.delay.send\_mail

delayed\_jobs Table



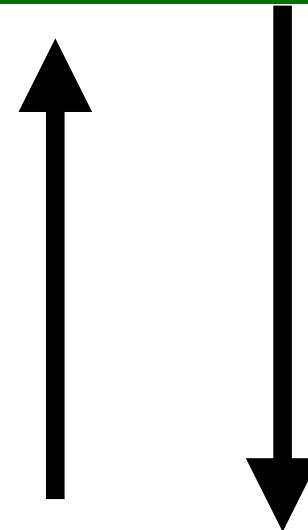
# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

處理開始! Go!

TestClass.delay.send\_mail

delayed\_jobs Table



# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

## Pros

- シンプルで分かりやすい、直感的
- ActiveRecordのモデルと相性が良い

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

## Pros

- シンプルで分かりやすい、直感的
- ActiveRecordのモデルと相性が良い

## Cons

- キュー指定が不可
- 優先度を細かく指定が難しい
- **Delayed** に依存した書き方になってしまう

# ".delay" Method

TestClass.delay.send\_mail( user\_id: user.id)

## Pros

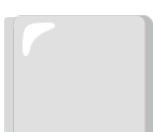
- シンプルで分かりやすい、直感的
- ActiveRecordのモデルと相性が良い

## Cons

- キュー指定が不可
- 優先度を細かく指定が難しい
- **Delayed** に依存した書き方になってしまう

Delayed からのがれられない……

# 移行手順

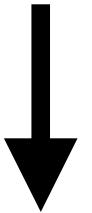
-  **Delayed 固有メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# ActiveJob で明示的に Delayed 指定

```
TestClass.delay.send_mail(  
  user_id: user.id  
)
```

# ActiveJob で明示的に Delayed 指定

```
TestClass.delay.send_mail(  
  user_id: user.id  
)
```



```
class TestClassJob < ApplicationJob  
  self.queue_adapter = :delayed  
  queue_as :mail
```

```
  def perform(user_id)  
    # ~~~~~  
  end  
end
```

# ActiveJob で明示的に Delayed 指定

```
TestClass.delay.send_mail(  
  user_id: user.id  
)
```



```
class TestClassJob < ApplicationJob  
  self.queue_adapter = :delayed  
  queue_as :mail
```

```
  def perform(user_id)  
    # ~~~~~  
  end  
end
```

# ActiveJob で明示的に Delayed 指定

```
TestClass.delay.send_mail(  
  user_id: user.id  
)
```

```
class TestClassJob < ApplicationJob  
  self.queue_adapter = :delayed  
  queue_as :mail
```

```
  def perform(user_id)  
    # ~~~~~  
  end  
end
```

SolidQueue を install しても Delayed で動作

# ActiveJob で明示的に Delayed 指定

```
TestClass.delay.send_mail(  
  user_id: user.id  
)
```

```
class TestClassJob < ApplicationJob  
  self.queue_adapter = :delayed  
  queue_as :mail
```

```
def perform(user_id)  
  # ~~~~~  
end  
end
```

SolidQueue を install しても Delayed で動作

移行中 SolidQueue 関連の PR をマージしても

稼働中システムには影響なし！

# 移行手順

-  **Delayed 固有メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# Solid Queue Configuration

config/solid\_queue/queue\_config.yml

production:

  dispatchers:

- polling\_interval: 1
- batch\_size: 500

Dispatcher

  workers:

- queues: "default"

solidqueuejobs

solidqueuereadyexecutions

  threads: 2

  processes: 5

  polling\_interval: 5

- queues: "mail"

  threads: 2

  processes: 5

  polling\_interval: 1

worker:default

worker: mail

# Solid Queue Configuration

config/solid\_queue/queue\_config.yml

production:

dispatchers:

- polling\_interval: 1
- batch\_size: 500

workers:

- queues: "default"

threads: 2

processes: 5

polling\_interval: 5

- queues: "mail"

threads: 2

processes: 5

polling\_interval: 1

Dispatcher

solidqueuejobs

solidqueuereadyexecutions

worker:default

worker: mail

# Solid Queue Configuration

config/solid\_queue/queue\_config.yml

production:

dispatchers:

- polling\_interval: 1
- batch\_size: 500

workers:

- queues: "default"

threads: 2

processes: 5

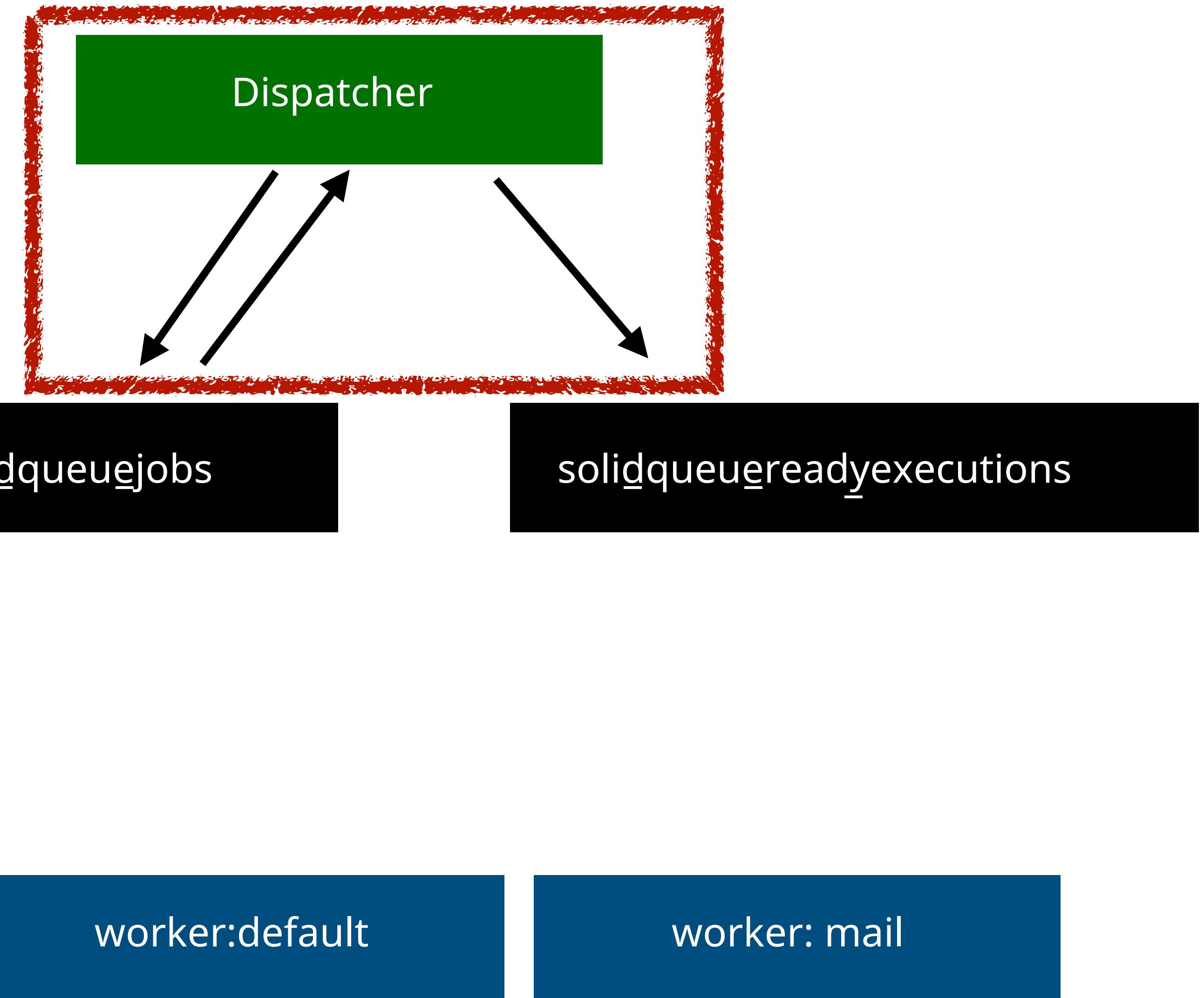
polling\_interval: 5

- queues: "mail"

threads: 2

processes: 5

polling\_interval: 1



# Solid Queue Configuration

config/solid\_queue/queue\_config.yml

production:

dispatchers:

- polling\_interval: 1
- batch\_size: 500

workers:

- queues: "default"

threads: 2

processes: 5

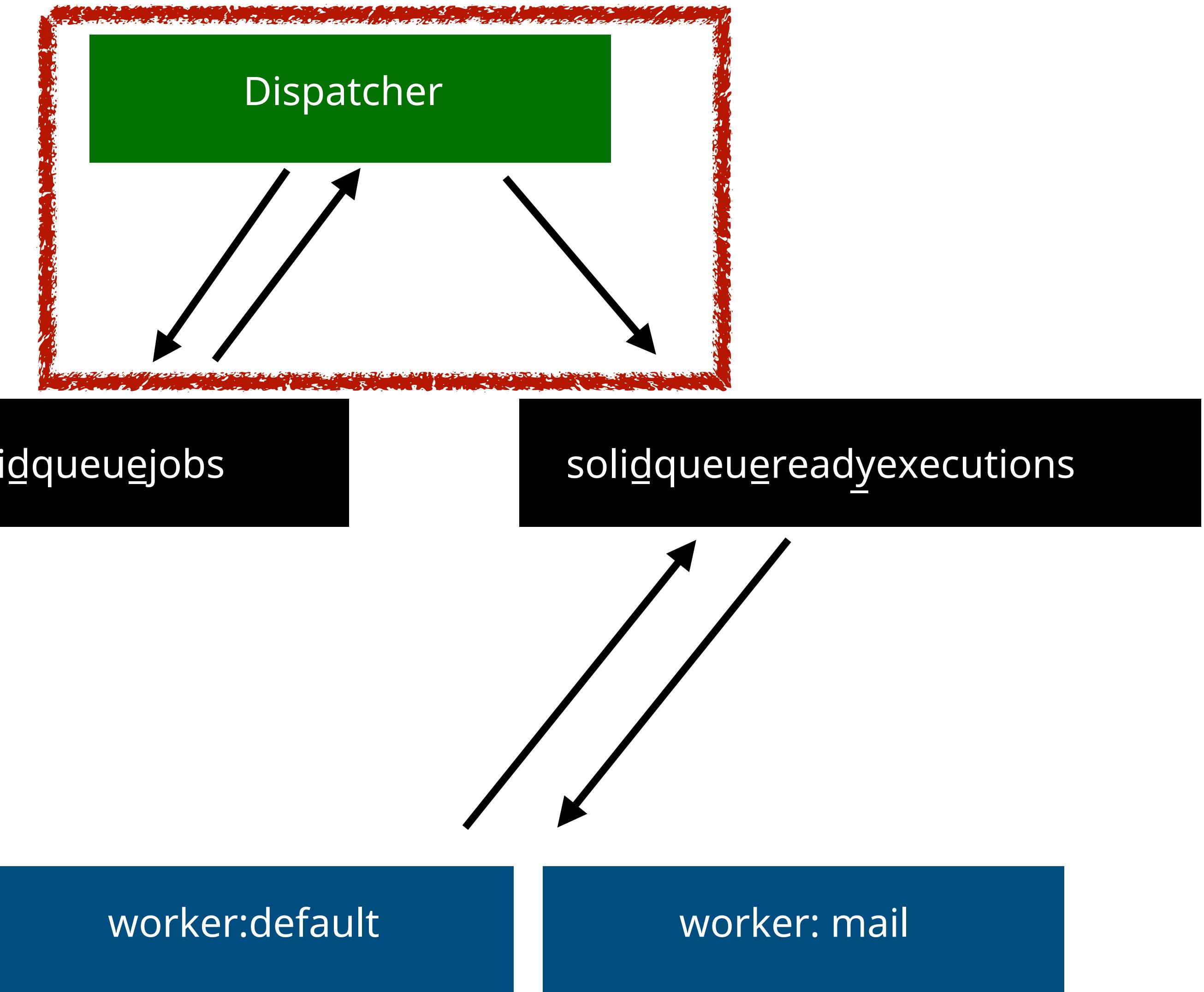
polling\_interval: 5

- queues: "mail"

threads: 2

processes: 5

polling\_interval: 1



# Solid Queue Configuration:2

config/solid\_queue/queue\_config.yml

production:

  dispatchers:

- polling\_interval: 1
- batch\_size: 500

  workers:

- queues: "default"
  - threads: 2
  - processes: 5
  - polling\_interval: 5
- queues: "mail"
  - threads: 2
  - processes: 5
  - polling\_interval: 1

# Solid Queue Configuration:2

config/solid\_queue/queue\_config.yml

```
production:
```

```
  dispatchers:
```

- polling\_interval: 1
- batch\_size: 500

```
  workers:
```

- queues: "default"
  - threads: 2
  - processes: 5
  - polling\_interval: 5

- queues: "mail"

```
  threads: 2
```

```
  processes: 5
```

```
  polling_interval: 1
```

通常のジョブ

# Solid Queue Configuration:2

config/solid\_queue/queue\_config.yml

production:

  dispatchers:

- polling\_interval: 1
- batch\_size: 500

  workers:

- queues: "default"
- threads: 2
- processes: 5
- polling\_interval: 5

- queues: "mail"
- threads: 2
- processes: 5
- polling\_interval: 1

通常のジョブ

メール送信用

# Solid Queue Configuration3

config/initializers/solid\_queue.rb

```
config.solid_queue.use_skip_locked = true
```

```
config.solid_queue.shutdown_timeout = 600.seconds
```

```
config.solid_queue.silence_polling = true
```

```
config.solid_queue.preserve_finished_jobs = false
```

# Solid Queue Configuration3

config/initializers/solid\_queue.rb

**UPDATE SKIP LOCKED を使う**

config.solid\_queue.use\_skip\_locked = true

config.solid\_queue.shutdown\_timeout = 600.seconds

config.solid\_queue.silence\_polling = true

config.solid\_queue.preserve\_finished\_jobs = false

# Solid Queue Configuration3

config/initializers/solid\_queue.rb

**UPDATE SKIP LOCKED を使う**

config.solid\_queue.use\_skip\_locked = true

**プロセスに TERM シグナル送信後プロセス終了までの時間 ( 10 分)**

config.solid\_queue.shutdown\_timeout = 600.seconds

config.solid\_queue.silence\_polling = true

config.solid\_queue.preserve\_finished\_jobs = false

# Solid Queue Configuration3

config/initializers/solid\_queue.rb

**UPDATE SKIP LOCKED を使う**

config.solid\_queue.use\_skip\_locked = true

**プロセスに TERM シグナル送信後プロセス終了までの時間 ( 10 分)**

config.solid\_queue.shutdown\_timeout = 600.seconds

**ワーカーとディスパッチャをポーリングするときの Active Record ログを抑制する**

config.solid\_queue.silence\_polling = true

config.solid\_queue.preserve\_finished\_jobs = false

# Solid Queue Configuration3

config/initializers/solid\_queue.rb

## **UPDATE SKIP LOCKED を使う**

config.solid\_queue.use\_skip\_locked = true

## **プロセスに TERM シグナル送信後プロセス終了までの時間 ( 10 分)**

config.solid\_queue.shutdown\_timeout = 600.seconds

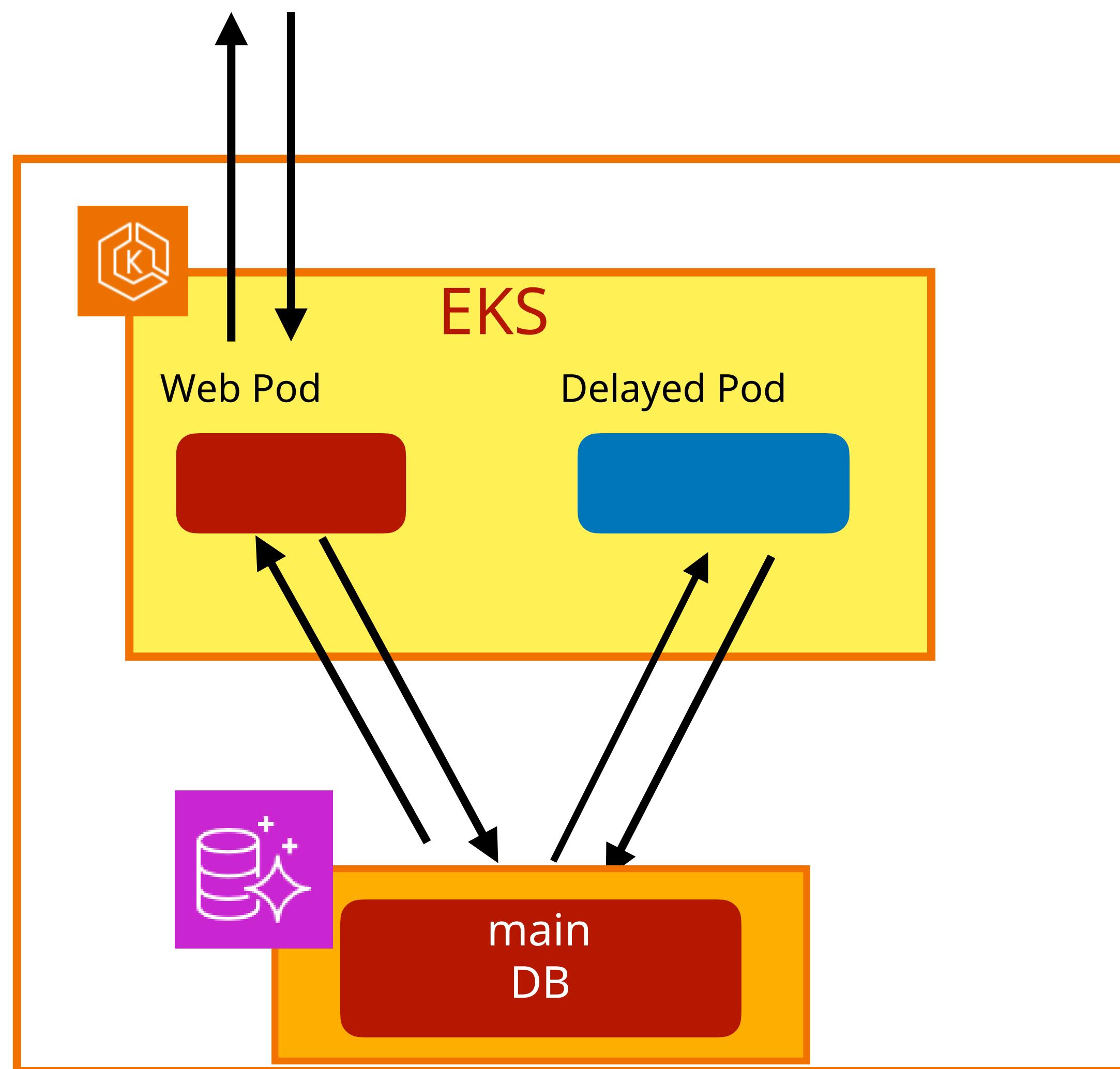
## **ワーカーとディスパッチャをポーリングするときの Active Record ログを抑制する**

config.solid\_queue.silence\_polling = true

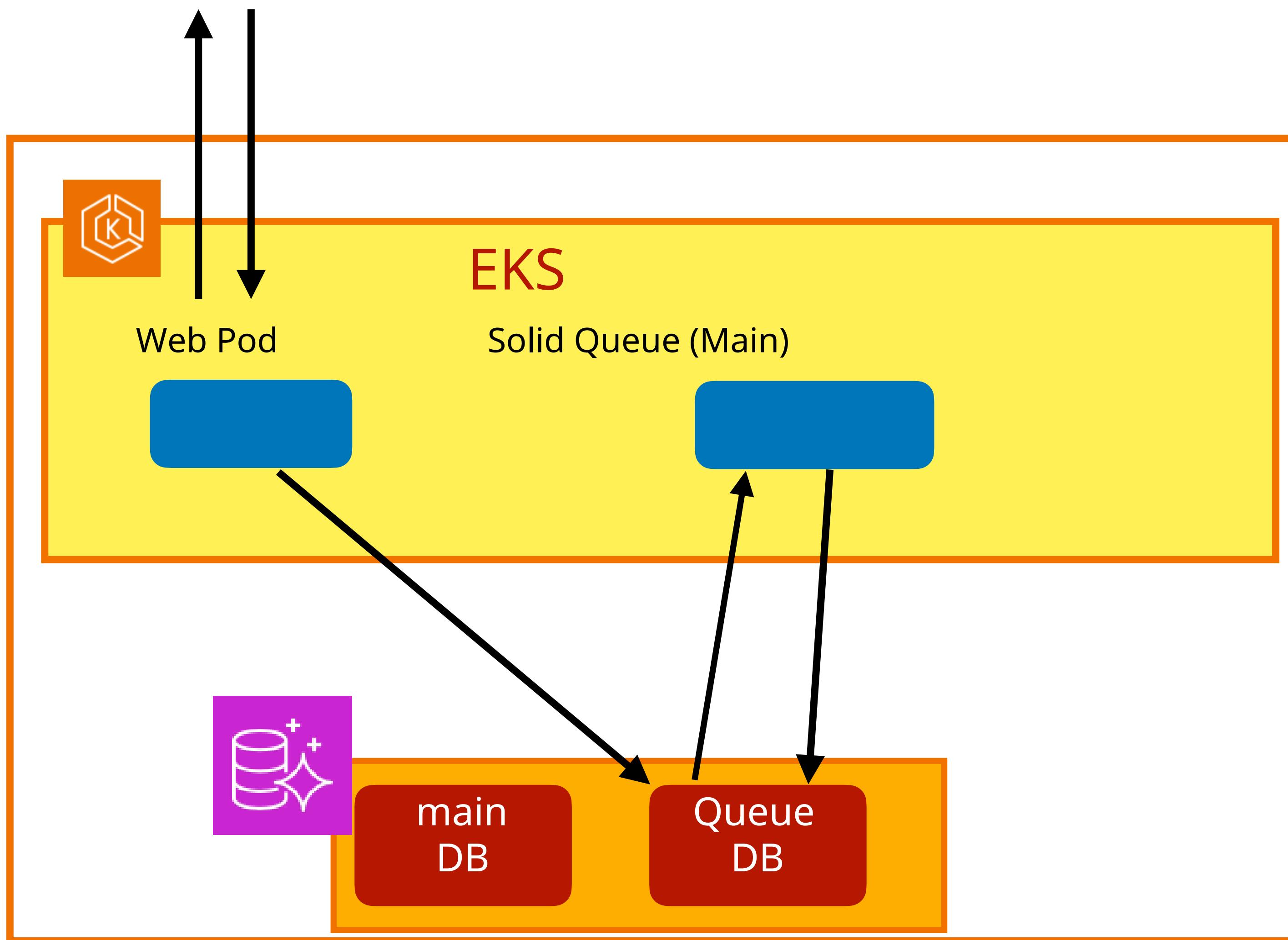
## **完了したジョブをテーブルに残さない**

config.solid\_queue.preserve\_finished\_jobs = false

# Infrastructure(Before)



# Solid Queue Infrastructure(After)



# 移行手順

-  **.delay メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# Migration Requirements

-  **.delay メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# 監視するぞ！ (delayed 編 )

- `delayed.job.count` - ジョブの総数
- `delayed.job.future_count` - 未来に実行されるジョブの数
- `delayed.job.working_count` - 現在処理中のジョブ数
- `delayed.job.workable_count` - 処理待ちのジョブ
- `delayed.job.erroring_count` - 実行に失敗したジョブ (失敗回数が 1 回以上)
- `delayed.job.failed_count` - 最終的に失敗したジョブ (もうリトライできない)
- `delayed.job.max_lock_age` - ずっとロックされ続けているジョブ
- `delayed.job.max_age` - 最も古いジョブの経過時間

**Delayed の機能として  
提供されてるんだぜ！**

# 監視するぞ！ (Solid Queue 編 )

- **job.count** - ジョブの総数
- **job.future\_count** - 未来に実行されるジョブの数
- **job.working\_count** - 現在処理中のジョブ数
- **job.workable\_count** - 処理待ちのジョブ
- **job.erroring\_count** - 実行に失敗したジョブ (失敗回数が 1 回以上)
- **job.failed\_count** - 最終的に失敗したジョブ (もうリトライできない)
- **job.max\_lock\_age** - ずっとロックされ続けているジョブ
- **job.max\_age** - 最も古いジョブの経過時間

# 監視するぞ！ (Solid Queue 編)

- `job.count` - ジョブの総数
- `job.future_count` - 未来に実行されるジョブの数
- `job.working_count` - 現在処理中のジョブ数
- `job.workable_count` - 処理待ちのジョブ
- `job.erroring_count` - 実行に失敗したジョブ (失敗回数が 1 回以上)
- `job.failed_count` - 最終的に失敗したジョブ (もうリトライできない)
- `job.max_lock_age` - ずっとロックされ続けているジョブ
- `job.max_age` - 最も古いジョブの経過時間

**Solid Queue の機能として  
提供されていないぜ！**

# 監視するぞ！（Solid Queue 編）

# 此前実装しました！

/ reporting\_solid\_queue\_job.rb

srockstyle 【非同期処理】 Datadogで最も古い処理待ちジョブの経過時間を取り扱うときに分単位になってしまふのを秒単位に (#16699) 4d419a9 · 4 months ago History

Code Blame 157 lines (139 loc) · 5.75 KB

```
1  # frozen_string_literal: true
2
3  # SolidQueueのジョブキューの状態を監視し、メトリクスを収集するジョブ
4  class Monitoring::ReportingSolidQueueJob < ApplicationJob
5    include Monitoring::SolidQueueMetricsHelper
6    include SolidQueueLogger
7    include SolidQueueRetryHandler
8    self.queue_adapter = :solid_queue unless Rails.env.test?
9    around_perform :solid_queue_logger
10   queue_as :teachme_monitoring
11
12  METRICS = %w[
13    future_count
14    erroring_count
15    failed_count
16    working_count
17    workable_count
18    max_age
19  ].freeze
20
21  # NOTE: 以下のメトリクスをSQLを見直して取得するようとする
22  # max_lock_age
23  # alert_age_percent
24
25  IGNORE_MONITORING_CLASS = %w[
26    Monitoring::ReportingSolidQueue
27  ].freeze
28
29  AGE_ALERT_THRESHOLD = 30.minutes
30
31  def perform
32    queues = fetch_queues
33    ActiveSupport::Notifications.instrument('solid_queue.monitor.run', default_tags) do
34      METRICS.each { |metric| emit_metric!(metric, queues) }
35    end
36  end
37
38  private
39
40  def default_tags
41    %w[ solid_queue monitoring ]
42  end
43
44  def emit_metric!(metric, queues)
45    metric = metric.to_s
46    metric = metric[0..1] + metric[2..-1].downcase
47    metric = metric[0..1] + metric[2..-1].downcase
48    metric = metric[0..1] + metric[2..-1].downcase
49    metric = metric[0..1] + metric[2..-1].downcase
50    metric = metric[0..1] + metric[2..-1].downcase
51    metric = metric[0..1] + metric[2..-1].downcase
52    metric = metric[0..1] + metric[2..-1].downcase
53    metric = metric[0..1] + metric[2..-1].downcase
54    metric = metric[0..1] + metric[2..-1].downcase
55    metric = metric[0..1] + metric[2..-1].downcase
56    metric = metric[0..1] + metric[2..-1].downcase
57    metric = metric[0..1] + metric[2..-1].downcase
58    metric = metric[0..1] + metric[2..-1].downcase
59    metric = metric[0..1] + metric[2..-1].downcase
60    metric = metric[0..1] + metric[2..-1].downcase
61    metric = metric[0..1] + metric[2..-1].downcase
62    metric = metric[0..1] + metric[2..-1].downcase
63    metric = metric[0..1] + metric[2..-1].downcase
64    metric = metric[0..1] + metric[2..-1].downcase
65    metric = metric[0..1] + metric[2..-1].downcase
66    metric = metric[0..1] + metric[2..-1].downcase
67    metric = metric[0..1] + metric[2..-1].downcase
68    metric = metric[0..1] + metric[2..-1].downcase
69    metric = metric[0..1] + metric[2..-1].downcase
70    metric = metric[0..1] + metric[2..-1].downcase
71    metric = metric[0..1] + metric[2..-1].downcase
72    metric = metric[0..1] + metric[2..-1].downcase
73    metric = metric[0..1] + metric[2..-1].downcase
74    metric = metric[0..1] + metric[2..-1].downcase
75    metric = metric[0..1] + metric[2..-1].downcase
76    metric = metric[0..1] + metric[2..-1].downcase
77    metric = metric[0..1] + metric[2..-1].downcase
78    metric = metric[0..1] + metric[2..-1].downcase
79    metric = metric[0..1] + metric[2..-1].downcase
80    metric = metric[0..1] + metric[2..-1].downcase
81    metric = metric[0..1] + metric[2..-1].downcase
82    metric = metric[0..1] + metric[2..-1].downcase
83    metric = metric[0..1] + metric[2..-1].downcase
84    metric = metric[0..1] + metric[2..-1].downcase
85    metric = metric[0..1] + metric[2..-1].downcase
86    metric = metric[0..1] + metric[2..-1].downcase
87    metric = metric[0..1] + metric[2..-1].downcase
88    metric = metric[0..1] + metric[2..-1].downcase
89    metric = metric[0..1] + metric[2..-1].downcase
90    metric = metric[0..1] + metric[2..-1].downcase
91    metric = metric[0..1] + metric[2..-1].downcase
92    metric = metric[0..1] + metric[2..-1].downcase
93    metric = metric[0..1] + metric[2..-1].downcase
94    metric = metric[0..1] + metric[2..-1].downcase
95    metric = metric[0..1] + metric[2..-1].downcase
96    metric = metric[0..1] + metric[2..-1].downcase
97    metric = metric[0..1] + metric[2..-1].downcase
98    metric = metric[0..1] + metric[2..-1].downcase
99    metric = metric[0..1] + metric[2..-1].downcase
100   metric = metric[0..1] + metric[2..-1].downcase
101   metric = metric[0..1] + metric[2..-1].downcase
102   metric = metric[0..1] + metric[2..-1].downcase
103   metric = metric[0..1] + metric[2..-1].downcase
104   metric = metric[0..1] + metric[2..-1].downcase
105   metric = metric[0..1] + metric[2..-1].downcase
106   metric = metric[0..1] + metric[2..-1].downcase
107   metric = metric[0..1] + metric[2..-1].downcase
108   metric = metric[0..1] + metric[2..-1].downcase
109   metric = metric[0..1] + metric[2..-1].downcase
110   metric = metric[0..1] + metric[2..-1].downcase
111   metric = metric[0..1] + metric[2..-1].downcase
112   metric = metric[0..1] + metric[2..-1].downcase
113   metric = metric[0..1] + metric[2..-1].downcase
114   metric = metric[0..1] + metric[2..-1].downcase
115   metric = metric[0..1] + metric[2..-1].downcase
116   metric = metric[0..1] + metric[2..-1].downcase
117   metric = metric[0..1] + metric[2..-1].downcase
118   metric = metric[0..1] + metric[2..-1].downcase
119   metric = metric[0..1] + metric[2..-1].downcase
120   metric = metric[0..1] + metric[2..-1].downcase
121   metric = metric[0..1] + metric[2..-1].downcase
122   metric = metric[0..1] + metric[2..-1].downcase
123   metric = metric[0..1] + metric[2..-1].downcase
124   metric = metric[0..1] + metric[2..-1].downcase
125   metric = metric[0..1] + metric[2..-1].downcase
126   metric = metric[0..1] + metric[2..-1].downcase
127   metric = metric[0..1] + metric[2..-1].downcase
128   metric = metric[0..1] + metric[2..-1].downcase
129   metric = metric[0..1] + metric[2..-1].downcase
130   metric = metric[0..1] + metric[2..-1].downcase
131   metric = metric[0..1] + metric[2..-1].downcase
132   metric = metric[0..1] + metric[2..-1].downcase
133   metric = metric[0..1] + metric[2..-1].downcase
134   metric = metric[0..1] + metric[2..-1].downcase
135   metric = metric[0..1] + metric[2..-1].downcase
136   metric = metric[0..1] + metric[2..-1].downcase
137   metric = metric[0..1] + metric[2..-1].downcase
138   metric = metric[0..1] + metric[2..-1].downcase
139   metric = metric[0..1] + metric[2..-1].downcase
140   metric = metric[0..1] + metric[2..-1].downcase
141   metric = metric[0..1] + metric[2..-1].downcase
142   metric = metric[0..1] + metric[2..-1].downcase
143   metric = metric[0..1] + metric[2..-1].downcase
144   metric = metric[0..1] + metric[2..-1].downcase
145   metric = metric[0..1] + metric[2..-1].downcase
146   metric = metric[0..1] + metric[2..-1].downcase
147   metric = metric[0..1] + metric[2..-1].downcase
148   metric = metric[0..1] + metric[2..-1].downcase
149   metric = metric[0..1] + metric[2..-1].downcase
150   metric = metric[0..1] + metric[2..-1].downcase
151   metric = metric[0..1] + metric[2..-1].downcase
152   metric = metric[0..1] + metric[2..-1].downcase
153   metric = metric[0..1] + metric[2..-1].downcase
154   metric = metric[0..1] + metric[2..-1].downcase
155   metric = metric[0..1] + metric[2..-1].downcase
156   metric = metric[0..1] + metric[2..-1].downcase
157   metric = metric[0..1] + metric[2..-1].downcase
```

# 監視するぞ！（Solid Queue 編）

# 此前実装しました！

/ reporting\_solid\_queue\_job.rb

srockstyle [非同期処理] Datadogで最も古い処理待ちジョブの経過時間を取り扱うときに分単位になってしまふのを秒単位に (#16699) 4d419a9 · 4 months ago History

Code Blame 157 lines (139 loc) · 5.75 KB

```
1  # frozen_string_literal: true
2
3  # SolidQueueのジョブキューの状態を監視し、メトリクスを収集するジョブ
4  class Monitoring::ReportingSolidQueueJob < ApplicationJob
5    include Monitoring::SolidQueueMetricsHelper
6    include SolidQueueLogger
7    include SolidQueueRetryHandler
8    self.queue_adapter = :solid_queue unless Rails.env.test?
9    around_perform :solid_queue_logger
10   queue_as :teachme_monitoring
11
12  METRICS = %w[
13    future_count
14    erroring_count
15    failed_count
16    working_count
17    workable_count
18    max_age
19  ].freeze
20
21  # NOTE: 以下のメトリクスをSQLを見直して取得するようにする
22  # max_lock_age
23  # alert_age_percent
24
25  IGNORE_MONITORING_CLASS = %w[
26    Monitoring::ReportingSolidQueue
27  ].freeze
28
29  AGE_ALERT_THRESHOLD = 30.minutes
30
31  def perform
32    queues = fetch_queues
33    ActiveSupport::Notifications.instrument('solid_queue.monitor.run', default_tags) do
34      METRICS.each { |metric| emit_metric!(metric, queues) }
35    end
36  end
37
38  private
39
40  def default_tags
41    %w[teachme monitoring]
42  end
43
44  def emit_metric!(metric, queues)
45    metric_name = metric.to_s.underscore
46    metric_name = metric_name[0..1].upcase + metric_name[2..-1] if metric_name =~ /\b[A-Z]/
47    metric_name = metric_name + '_count' if metric_name =~ /\b[A-Z]/
48    metric_name = metric_name + '_age' if metric_name =~ /\b[A-Z]/
49    metric_name = metric_name + '_percent' if metric_name =~ /\b[A-Z]/
50    metric_name = metric_name + '_lock' if metric_name =~ /\b[A-Z]/
51    metric_name = metric_name + '_error' if metric_name =~ /\b[A-Z]/
52    metric_name = metric_name + '_failed' if metric_name =~ /\b[A-Z]/
53    metric_name = metric_name + '_workable' if metric_name =~ /\b[A-Z]/
54    metric_name = metric_name + '_working' if metric_name =~ /\b[A-Z]/
55    metric_name = metric_name + '_future' if metric_name =~ /\b[A-Z]/
56    metric_name = metric_name + '_erroring' if metric_name =~ /\b[A-Z]/
57    metric_name = metric_name + '_locked' if metric_name =~ /\b[A-Z]/
58    metric_name = metric_name + '_locked_percent' if metric_name =~ /\b[A-Z]/
59    metric_name = metric_name + '_locked_error' if metric_name =~ /\b[A-Z]/
60    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
61    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
62    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
63    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
64    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
65    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
66    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
67    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
68    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
69    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
70    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
71    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
72    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
73    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
74    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
75    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
76    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
77    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
78    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
79    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
80    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
81    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
82    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
83    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
84    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
85    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
86    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
87    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
88    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
89    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
90    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
91    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
92    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
93    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
94    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
95    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
96    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
97    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
98    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
99    metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
100   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
101   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
102   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
103   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
104   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
105   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
106   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
107   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
108   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
109   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
110   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
111   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
112   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
113   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
114   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
115   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
116   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
117   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
118   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
119   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
120   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
121   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
122   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
123   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
124   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
125   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
126   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
127   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
128   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
129   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
130   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
131   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
132   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
133   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
134   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
135   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
136   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
137   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
138   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
139   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
140   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
141   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
142   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
143   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
144   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
145   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
146   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
147   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
148   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
149   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
150   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
151   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
152   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
153   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
154   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
155   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
156   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
157   metric_name = metric_name + '_locked_error_percent' if metric_name =~ /\b[A-Z]/
```

# 監視するぞ! (Solid Queue 編)

# 監視するぞ! (Solid Queue 編)

## キューの状態メトリクス

メトリクス名	解説
future_count	将来実行されるジョブの数。特定の時刻に実行がスケジュールされているジョブの合計
workable_count	処理待ちのジョブの数。ワーカーにいつでも実行してもらえる状態のジョブの合計
working_count	現在処理中のジョブの数。ワーカーが今まさに動かしているジョブの合計
retry_count	再実行を待っているジョブの数。一度失敗して、リトライ処理が設定されているジョブの合計

# 監視するぞ! (Solid Queue 編)

## キューの状態メトリクス

メトリクス名	解説
future_count	将来実行されるジョブの数。特定の時刻に実行がスケジュールされているジョブの合計
workable_count	処理待ちのジョブの数。ワーカーにいつでも実行してもらえる状態のジョブの合計
working_count	現在処理中のジョブの数。ワーカーが今まさに動かしているジョブの合計
retry_count	再実行を待っているジョブの数。一度失敗して、リトライ処理が設定されているジョブの合計

## パフォーマンスメトリクス

メトリクス名	解説
throughput_rate	直近 1 分間に処理が完了したジョブの数
max_age	最長待機時間。最も長く待っているジョブの経過時間
median_wait_time	待機時間の中央値
max_lock_age	最長実行。
average_lock_age	平均実行時間。実行中のジョブの平均的な処理時間

# 監視するぞ！(Solid Queue 編)

## キューの状態メトリクス

メトリクス名	解説
future_count	将来実行されるジョブの数。特定の時刻に実行がスケジュールされているジョブの合計
workable_count	処理待ちのジョブの数。ワーカーにいつでも実行してもらえる状態のジョブの合計
working_count	現在処理中のジョブの数。ワーカーが今まさに動かしているジョブの合計
retry_count	再実行を待っているジョブの数。一度失敗して、リトライ処理が設定されているジョブの合計

## パフォーマンスメトリクス

メトリクス名	解説
throughput_rate	直近 1 分間に処理が完了したジョブの数
max_age	最長待機時間。最も長く待っているジョブの経過時間
median_wait_time	待機時間の中央値
max_lock_age	最長実行。
average_lock_age	平均実行時間。実行中のジョブの平均的な処理時間

## エラーアラートメトリクス

メトリクス名	解説
erroring_count	エラー付きで失敗したジョブの数。エラーメッセージが付いている失敗ジョブの合計。
failed_count	直近1時間で失敗したジョブの総数。エラーの有無を問わず、失敗したジョブの合計。
stale_jobs_count	停滞しているジョブの数。長時間（1時間以上）実行中のジョブで、デッドロックや無限ループの可能性。
error_rate	エラー率。直近1時間の全ジョブのうち、エラーで失敗した割合。
worker_utilization	ワーカーの使用率。割り当てられたワーカーがどれだけ働いているかの割合。
alert_age_percent	アラート閾値比率だ。優先度を考慮した待機時間が、事前に設定した閾値にどれだけ近づいているかを示す割合。

# ★ SolidQueueダッシュボード

Saved Views ▼ Filter by: rails\_env production ▼ backend \* ▼ queue \* ▼

Share

Anomalies

Show Overlays

Configure

+ Add Widgets

1h Past 1 Hour

1h

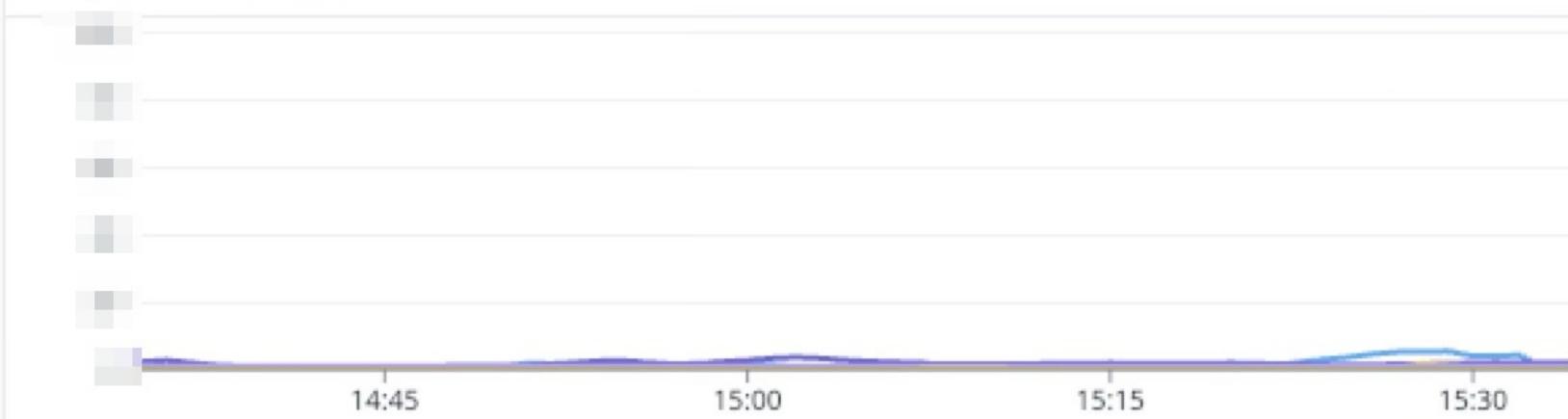
Past 1 Hour

◀

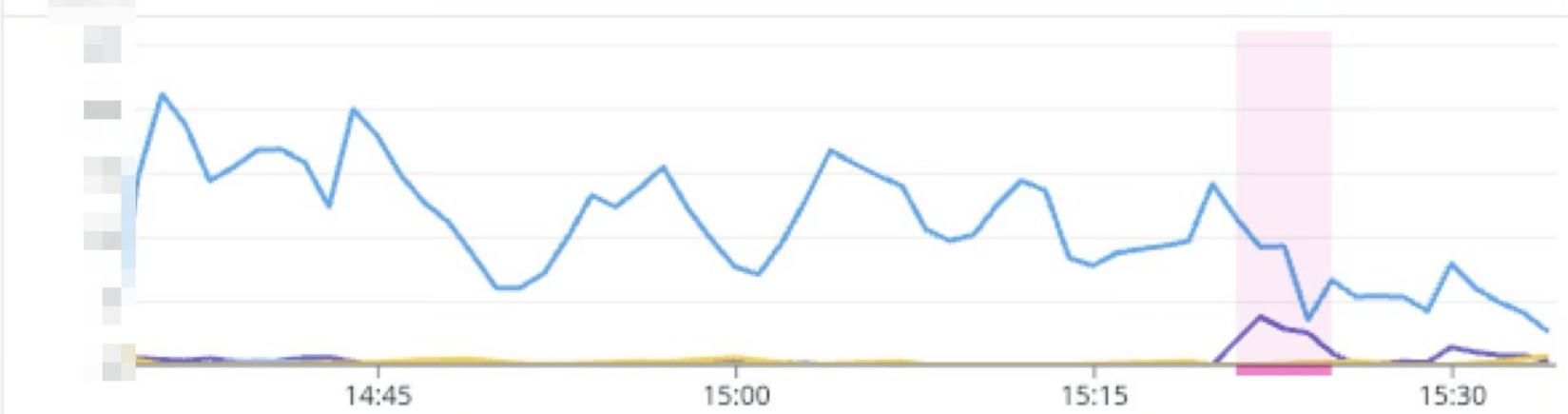
▶

🔍

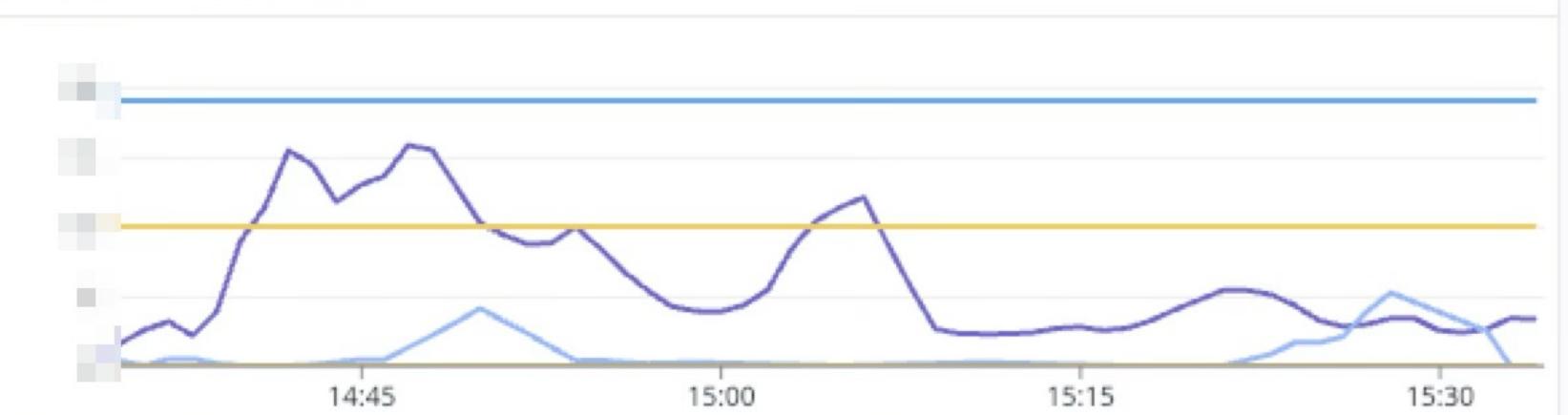
処理中のジョブ数



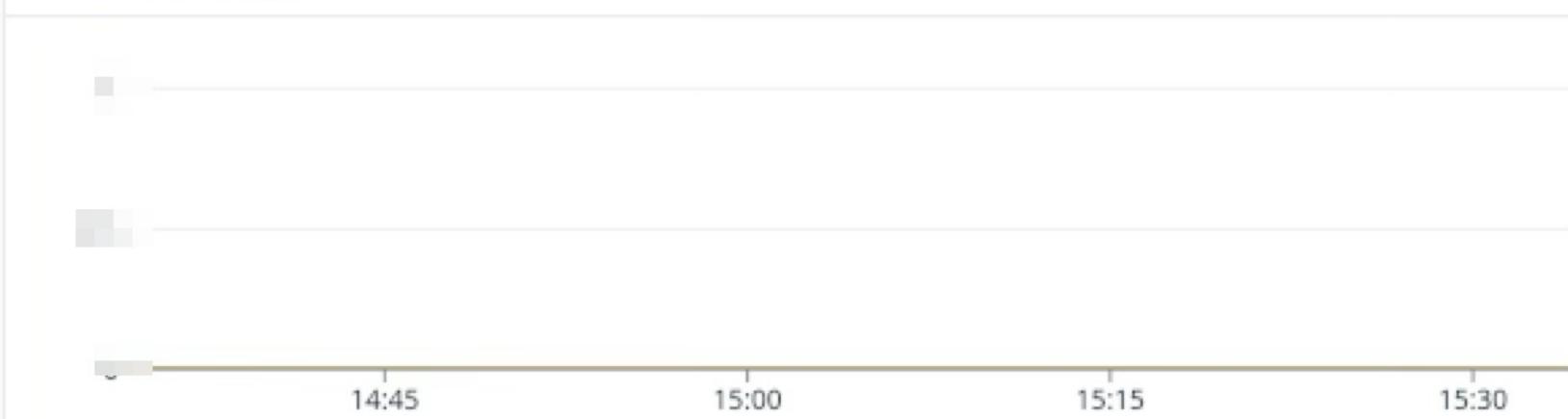
処理待ちのジョブ数



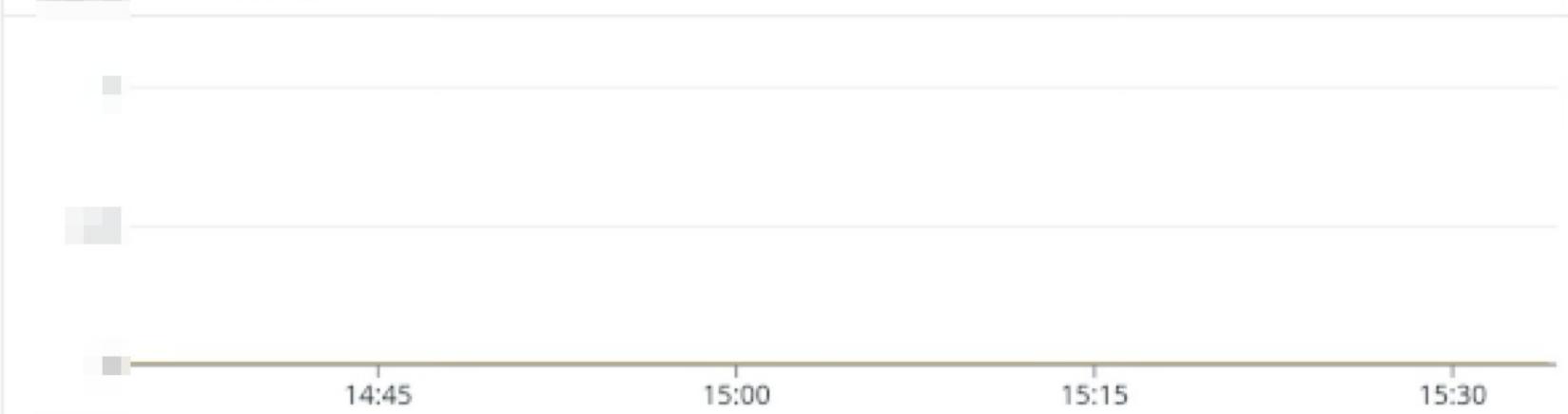
実行予定のジョブ数



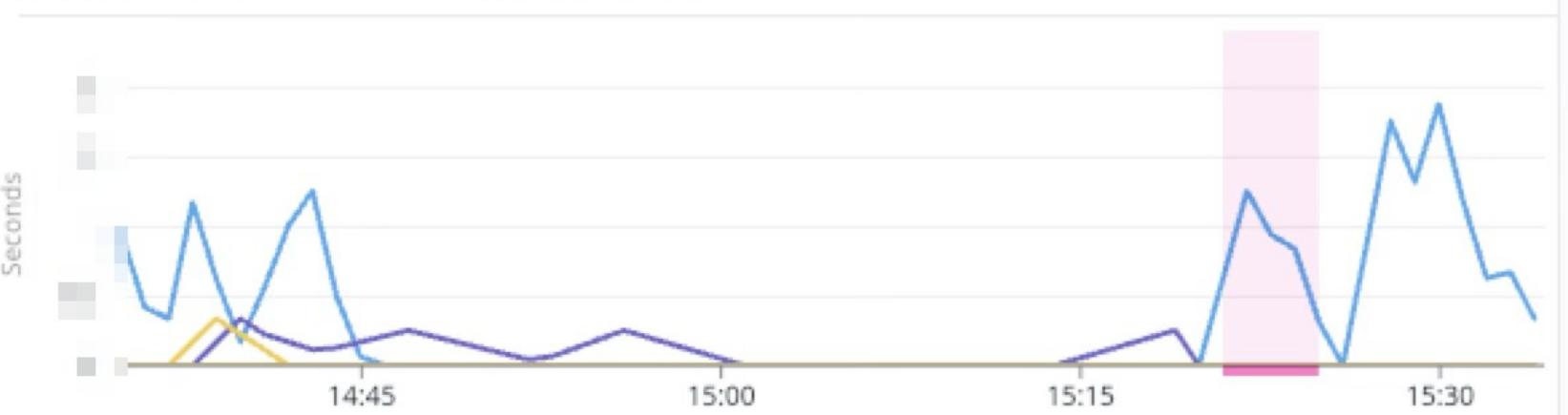
エラー発生回数



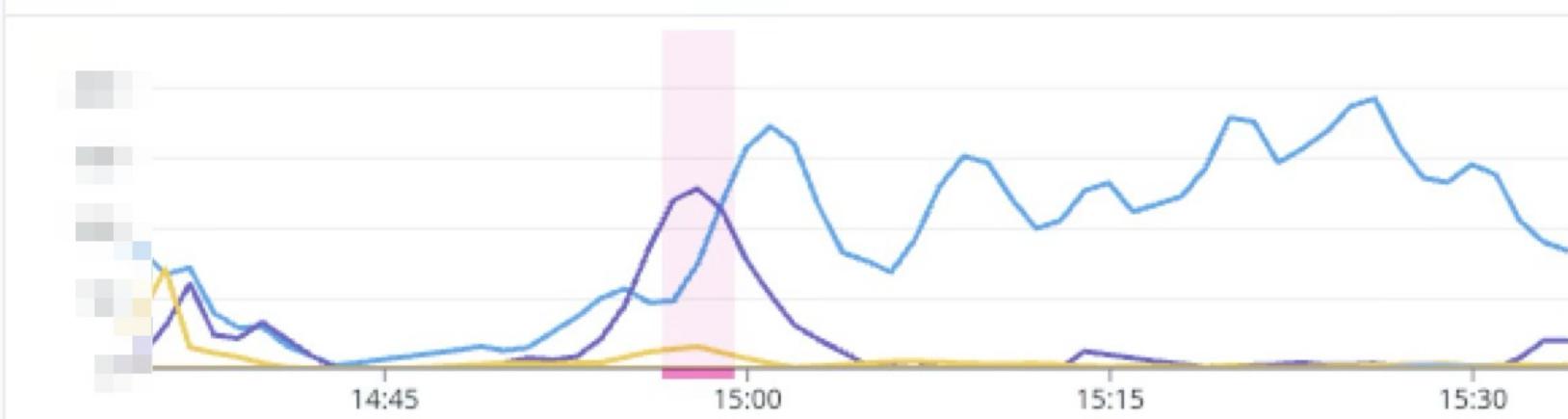
失敗したジョブ数



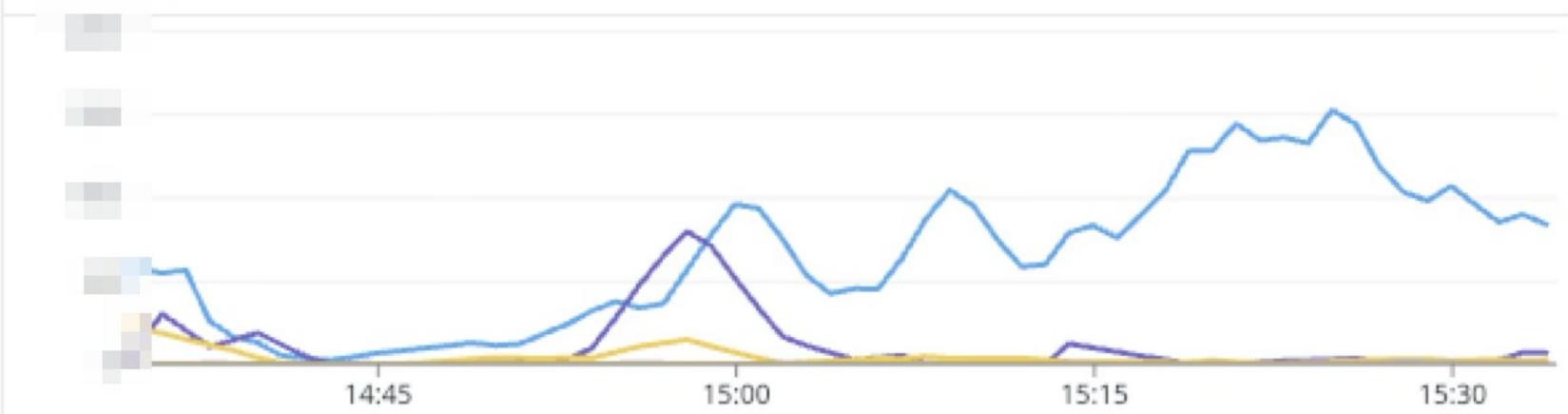
最も古い処理待ちジョブの経過時間 (秒)



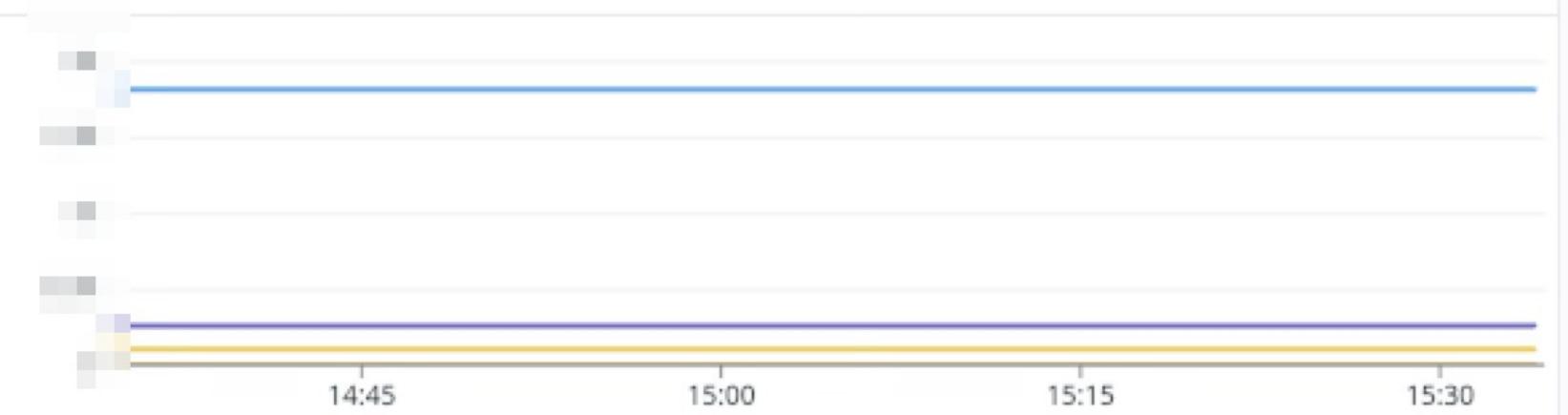
最も長く実行されているジョブの経過時間 (秒)



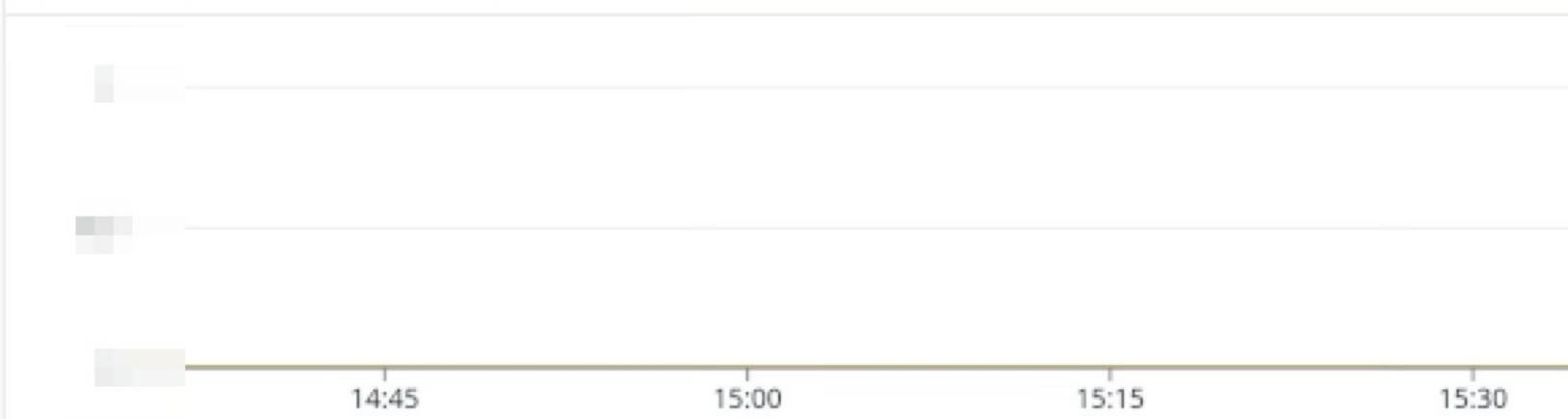
平均実行時間 (秒)



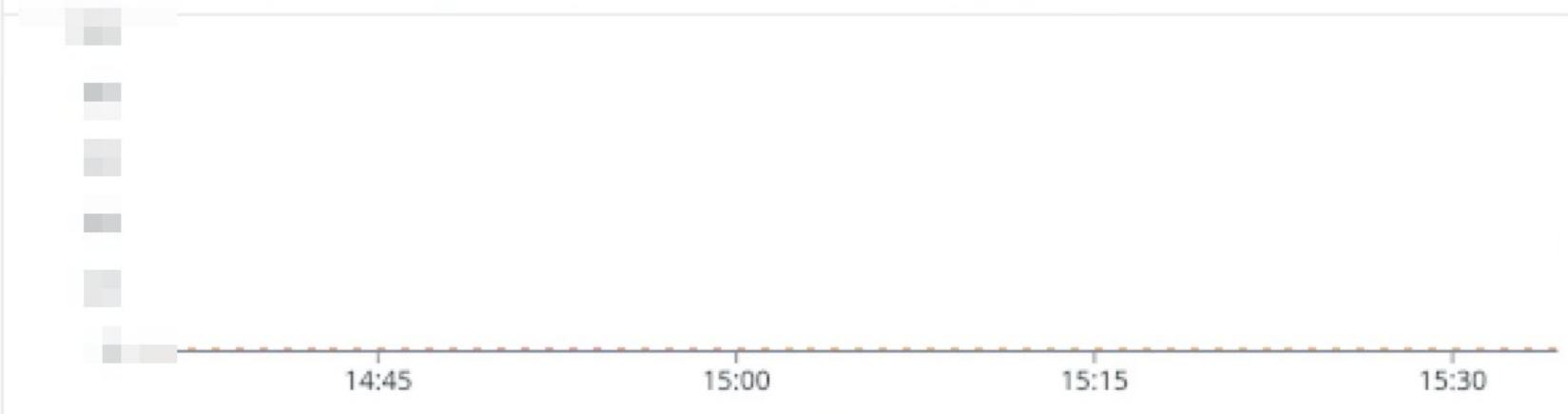
リトライ待機中ジョブ数



長時間実行ジョブ数 (1時間以上)



優先度に応じた待機時間閾値に対する到達率(%) / 低いほどOK



Add [Widgets](#) or [Powerpacks](#)

# Migration Requirements

-  **.delay メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# Delayed -> SolidQueue!

# Delayed -> SolidQueue!

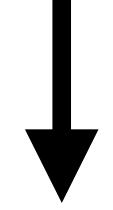
```
class TestClassJob < ApplicationJob
  self.queue_adapter = :delayed
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
end
```

# Delayed -> SolidQueue!

```
class TestClassJob < ApplicationJob
  self.queue_adapter = :delayed
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
```



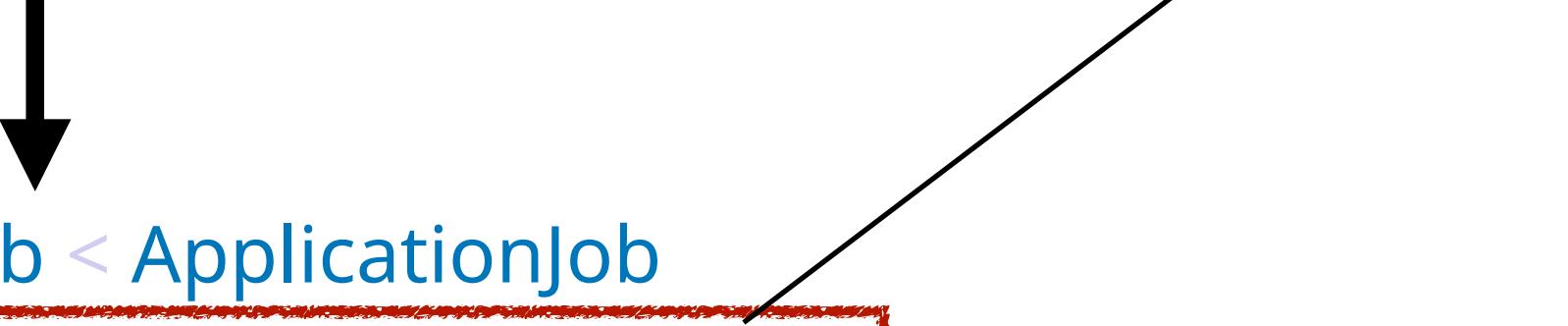
```
class TestClassJob < ApplicationJob
  self.queue_adapter = :solid_queue
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
```

# Delayed -> SolidQueue!

```
class TestClassJob < ApplicationJob
  self.queue_adapter = :delayed
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
end
```



delayed -> solidqueue に変更!

```
class TestClassJob < ApplicationJob
  self.queue_adapter = :solid_queue
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
end
```

# Delayed -> SolidQueue!

```
class TestClassJob < ApplicationJob
  self.queue_adapter = :delayed
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
end
```

```
class TestClassJob < ApplicationJob
  self.queue_adapter = :solid_queue
  queue_as :mail
```

```
def perform(user_id)
  # ~~~~~
end
end
```

delayed -> solidqueue に変更!

x Jobs Count

# SolidQueueリリース計画20250305

## Overview

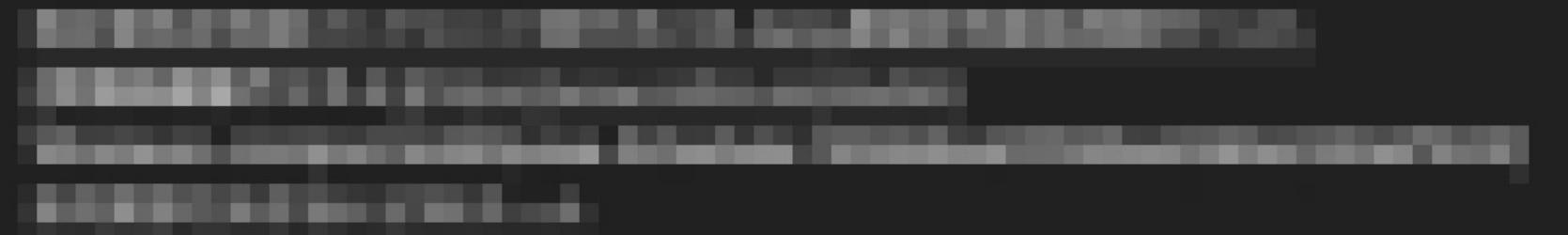
ジョブを一斉にリリースするのでその計画です。

作業する人

- @Shohei Kobayashi

## Input

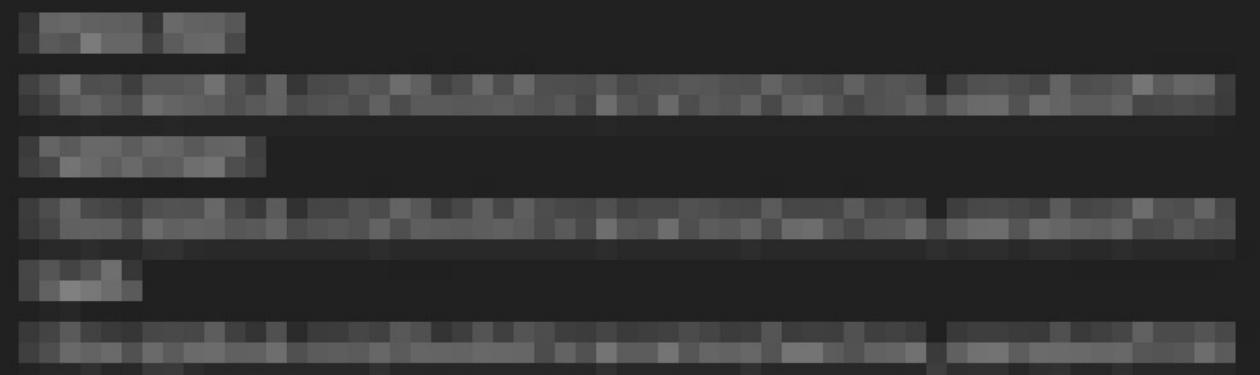
notion



duvel RR

基盤設計と導入

apps



## Goal

システムで使うデフォルトをSolidQueueにするのがマージされればOK

#16254 全ての環境のデフォルトをSolidQueueにする Open

# SolidQueueリリース計画20250305

## Overview

ジョブを一斉にリリースするのでその計画です。

作業する人

- @Shohei Kobayashi

## Input

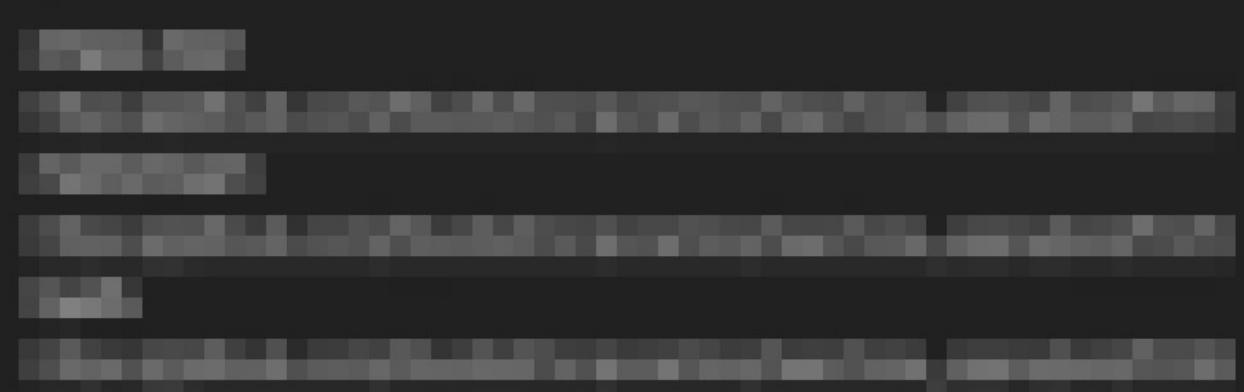
notion



duvel RR

基盤設計と導入

apps



## Goal

システムで使うデフォルトをSolidQueueにするのがマージされればOK

#16254 全ての環境のデフォルトをSolidQueueにする Open

**2025/3/5 完全移行作業! done!**

# SolidQueueリリース計画20250305

## Overview

ジョブを一斉にリリースするのでその計画です。

作業する人

- @Shohei Kobayashi

## Input

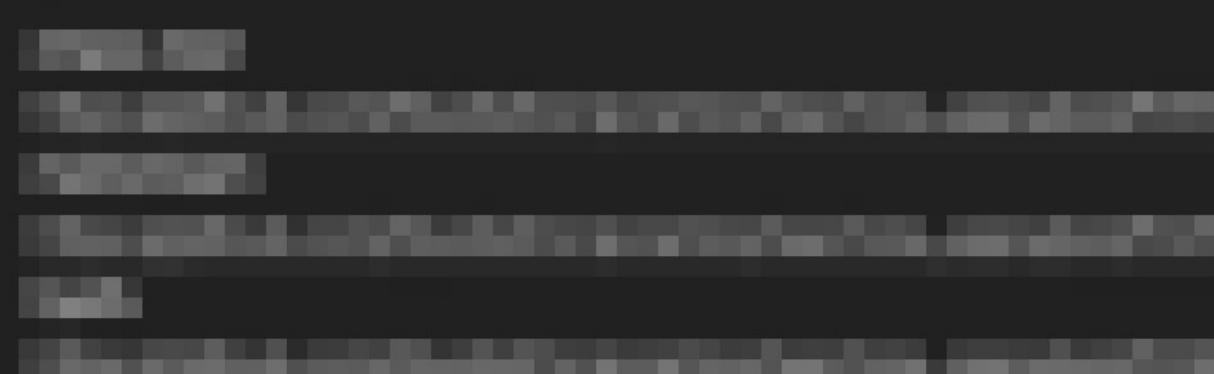
notion



duvel RR

基盤設計と導入

apps



## Goal

システムで使うデフォルトをSolidQueueにするのがマージされればOK

#16254 全ての環境のデフォルトをSolidQueueにする Open

2025/3/5 完全移行作業! done!

やったか!?

# Migration Requirements

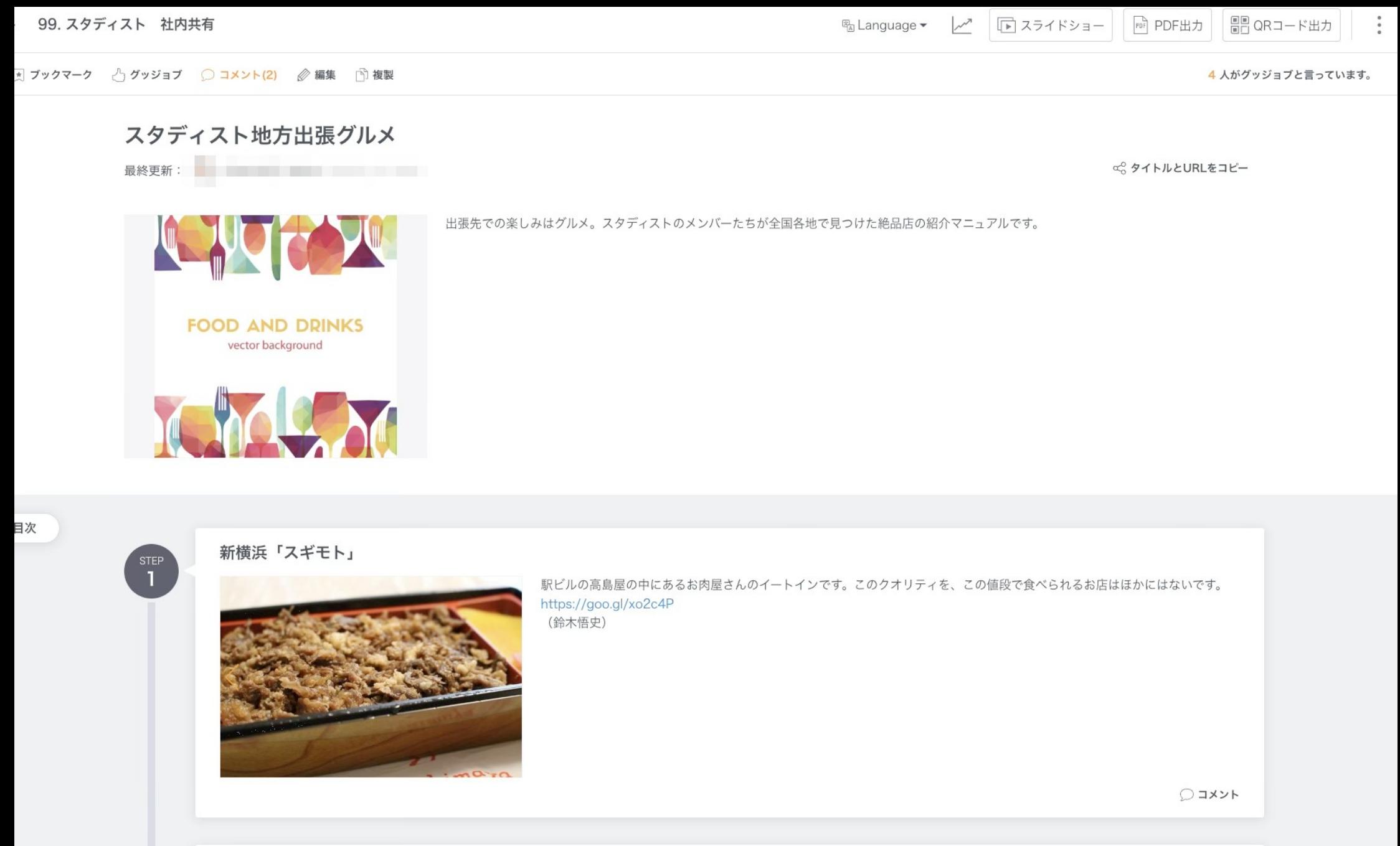
-  **.delay メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

**できてないぜ！**

- はじめに： Teachme Biz について
- 課題 Part1 (移行前)
- 選定
- 設計と実装
- 移行
- 課題 Part2( 移行後 )
- まとめ

**enqueueeaftertransactionncommit 効いてない問題**

# enqueue\_after\_transaction\_commit 設定されてない?



2012年リリース Rails3.2.1 -> 2025年3月当時 Rails7.1

**enqueue\_after\_transaction\_commit** とは

# **enqueueaftertransactioncommit とは**

**DB のトランザクション整合性を有効にできるオプション。**

# **enqueue\_after\_transaction\_commit** とは

DB のトランザクション整合性を有効にできるオプション。

```
class ApplicationJob < ActiveJob::Base
```

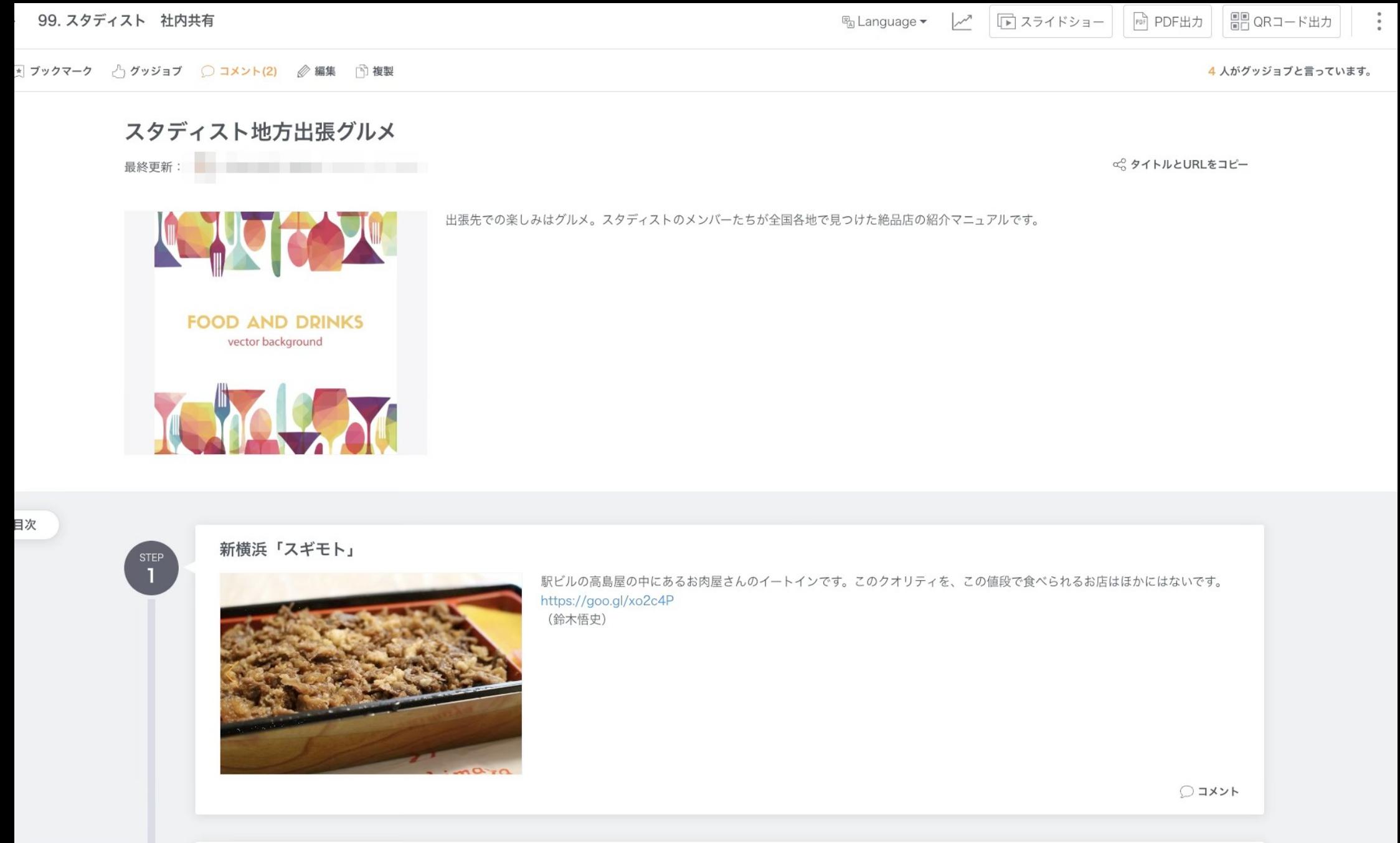
```
  self.enqueue_after_transaction_commit = true
```

```
end
```

# enqueue\_after\_transaction\_commit は Rails7.2 から

- ✓ 用途に応じて複数のキュー管理が可能
  - ✓ 新しい AWS リソースが不要
  - ✓ 線形スケールアウト可能
  - ✓ DB ロックでパフォーマンス劣化しない
- ✓ トランザクション内呼び出し対応
- ✓ Rails との親和性
  - ✓ 定期実行処理基盤がある
  - ✓ 無課金で使いたい

# enqueue\_after\_transaction\_commit... ?



2012年リリース Rails3.2.1 -> 2025年3月当時 Rails7.1

# トランザクション内部に書かれたジョブ問題 1

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

# トランザクション内部に書かれたジョブ問題 1

ユーザを DB に保存（未コミット）

```
 ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

# トランザクション内部に書かれたジョブ問題 1

```
ActiveRecord::Base.transaction do
  user = User.create!(name: "新しいユーザー")
  WelcomeEmailJob.perform_later(user.id)
  raise "Something went wrong!"
end
```

ユーザを DB に保存 (未コミット)

ジョブをキューに入れる

# トランザクション内部に書かれたジョブ問題 1

```
ActiveRecord::Base.transaction do
  user = User.create!(name: "新しいユーザー")
  WelcomeEmailJob.perform_later(user.id)
  raise "Something went wrong!"
end
```

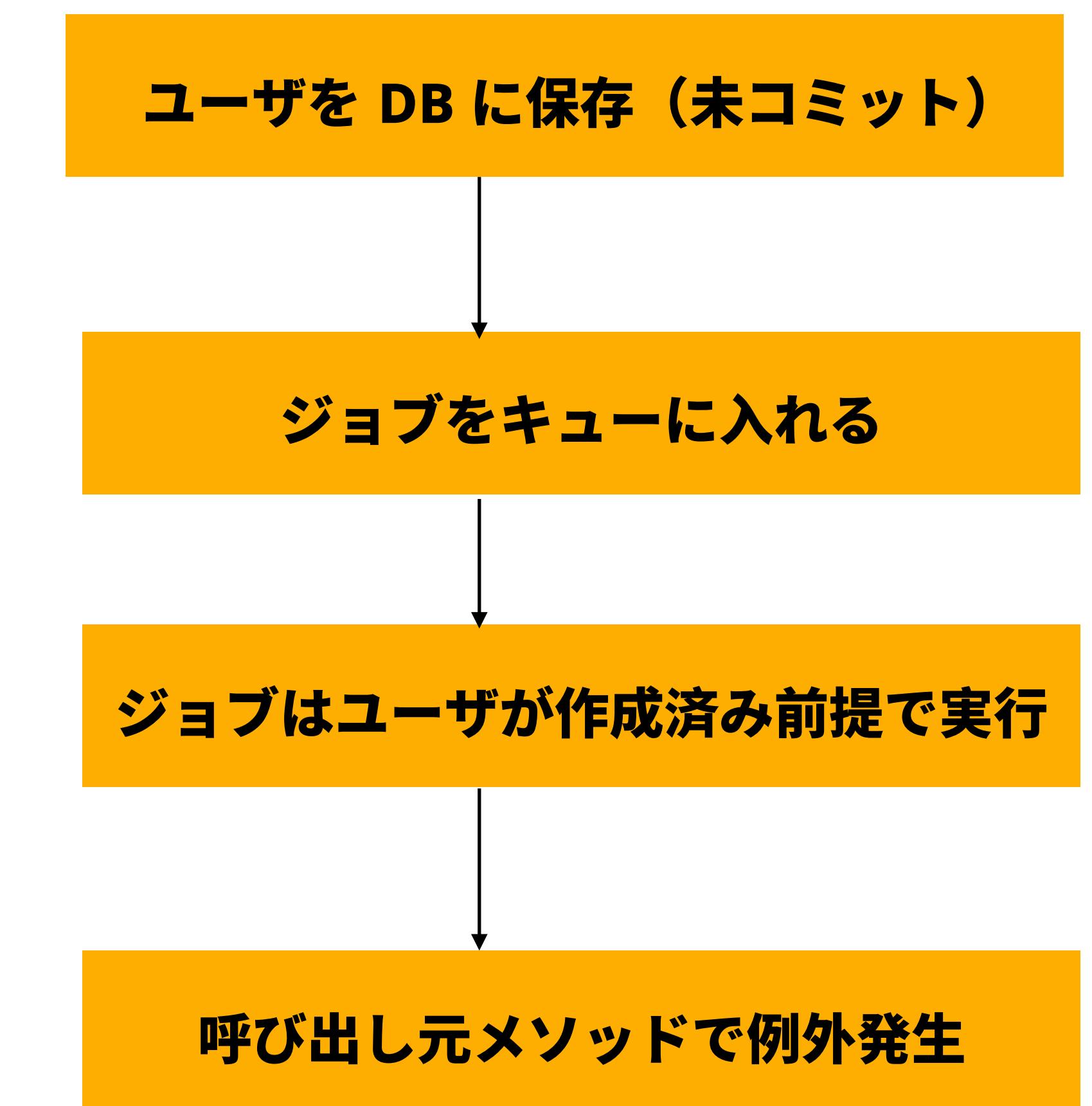
ユーザを DB に保存 (未コミット)

ジョブをキューに入れる

ジョブはユーザが作成済み前提で実行

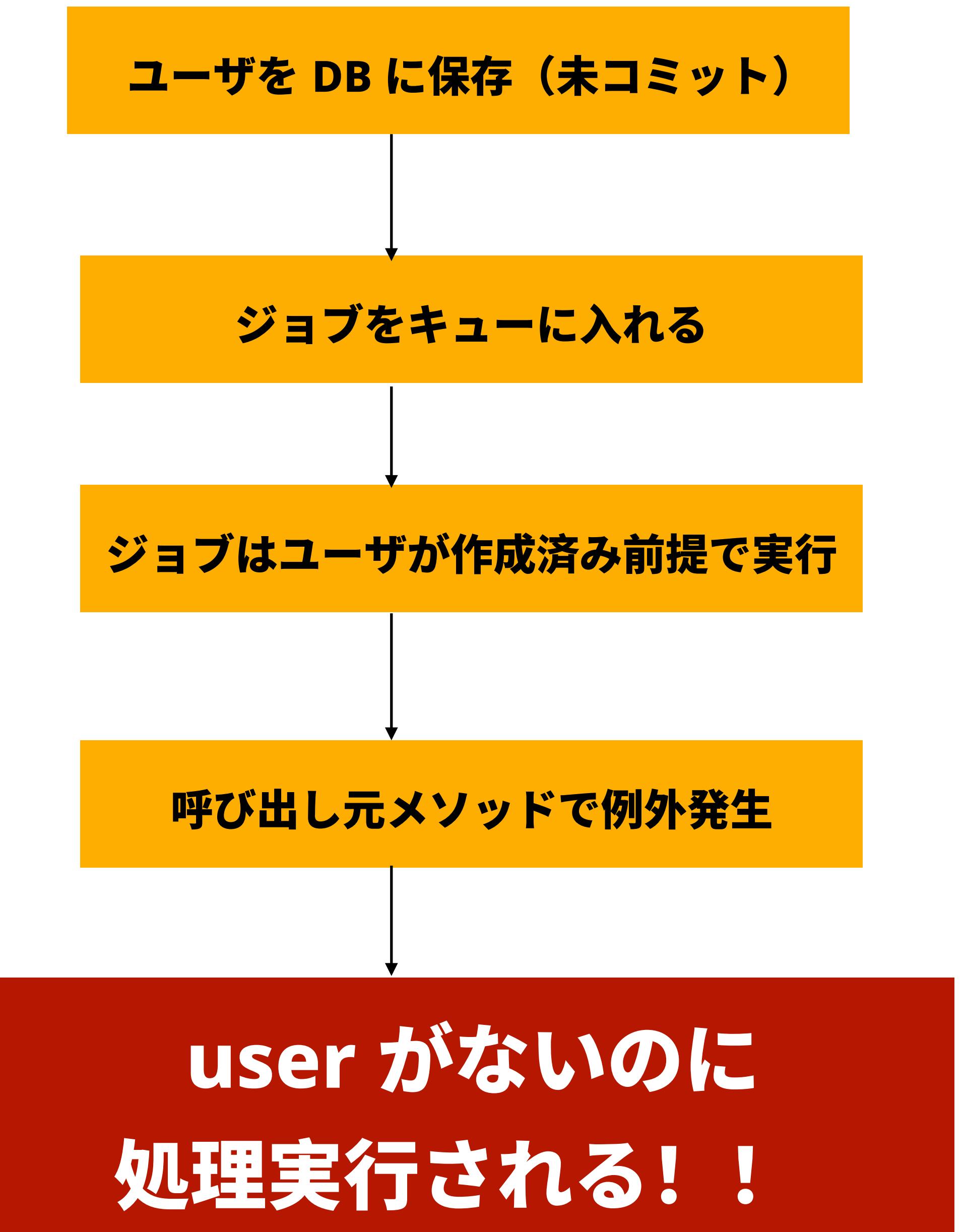
# トランザクション内部に書かれたジョブ問題 1

```
ActiveRecord::Base.transaction do
  user = User.create!(name: "新しいユーザー")
  WelcomeEmailJob.perform_later(user.id)
  raise "Something went wrong!"
end
```

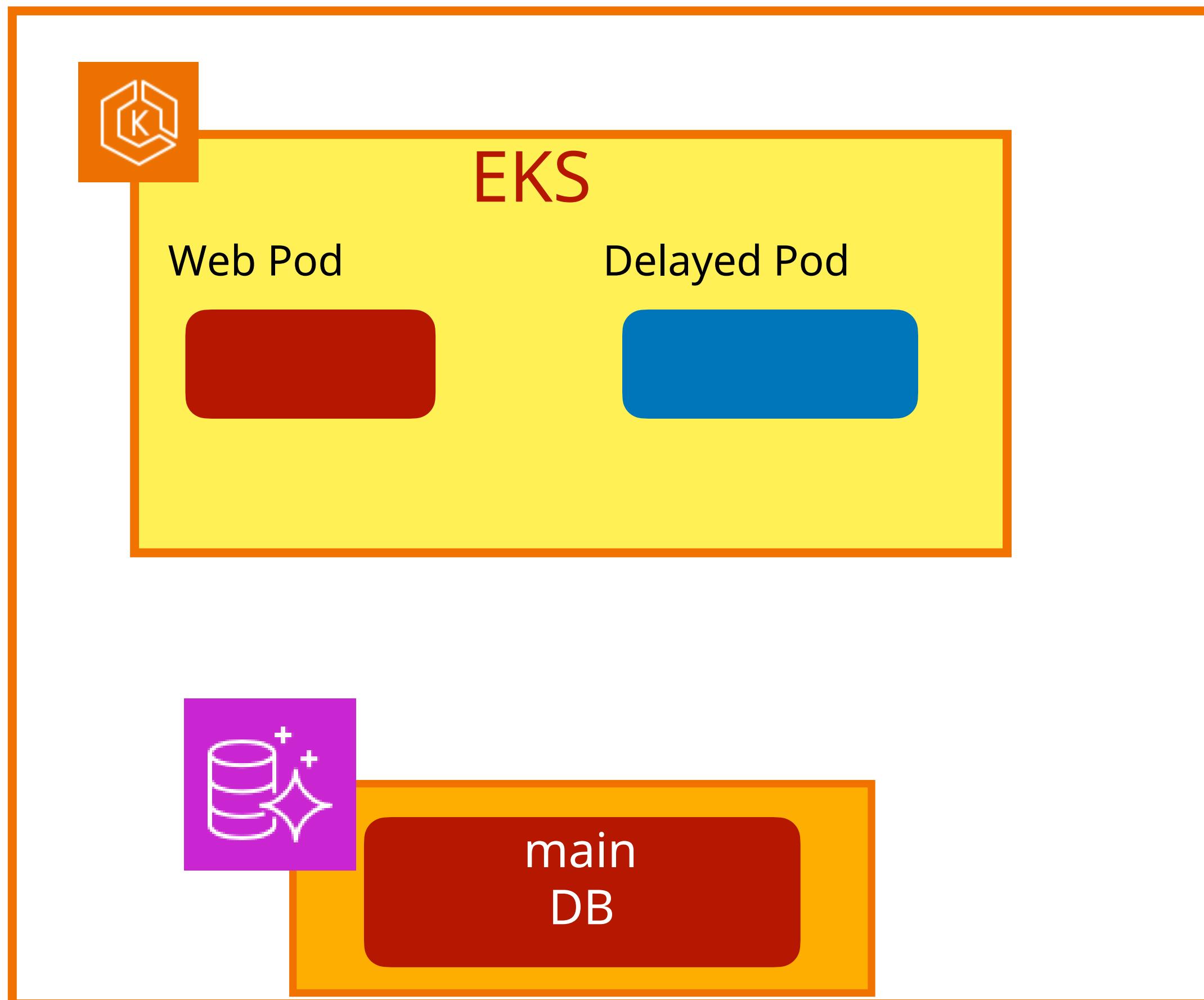


# トランザクション内部に書かれたジョブ問題 1

```
ActiveRecord::Base.transaction do
  user = User.create!(name: "新しいユーザー")
  WelcomeEmailJob.perform_later(user.id)
  raise "Something went wrong!"
end
```

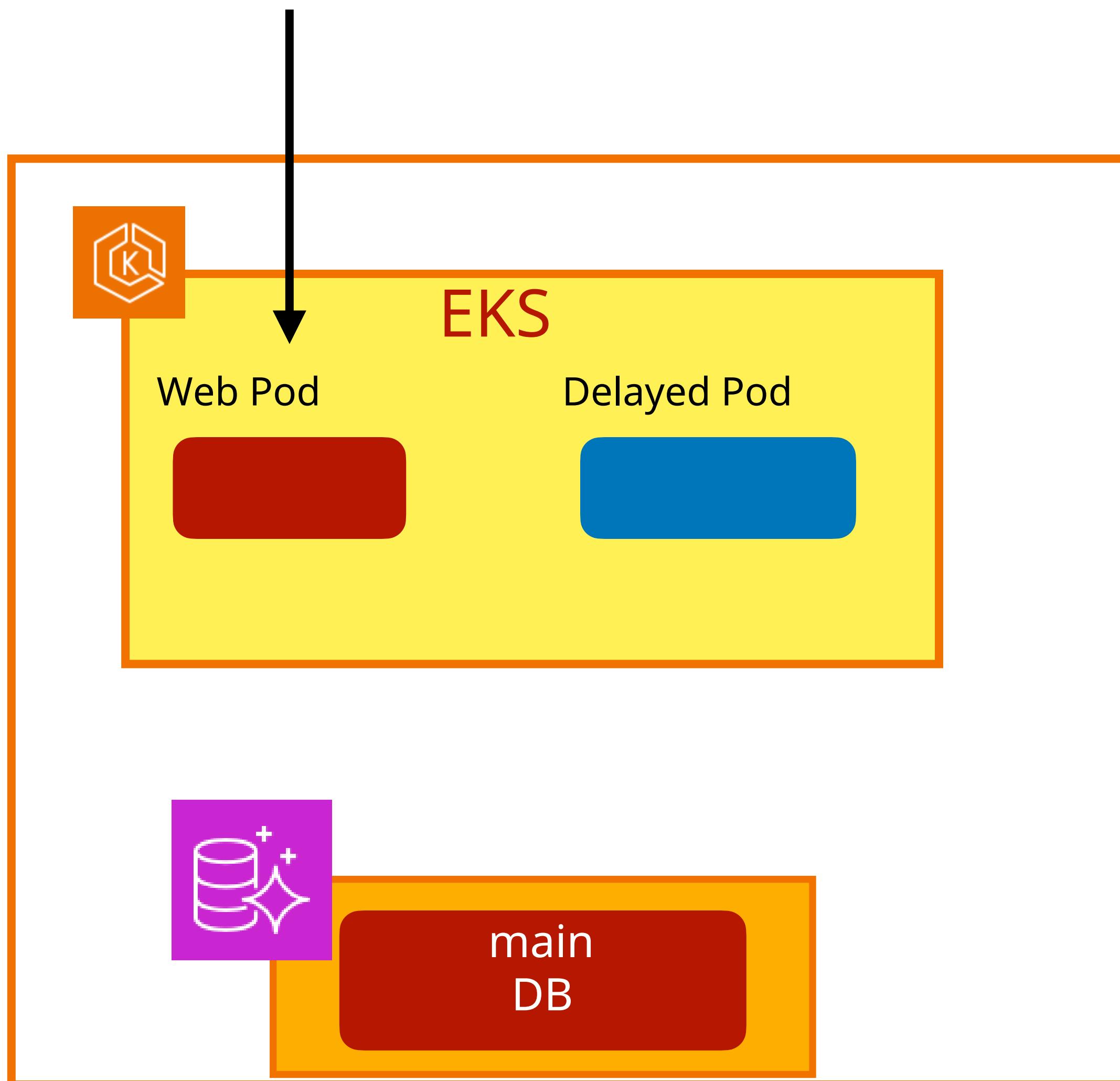


# トランザクション内部に書かれたジョブ問題 2

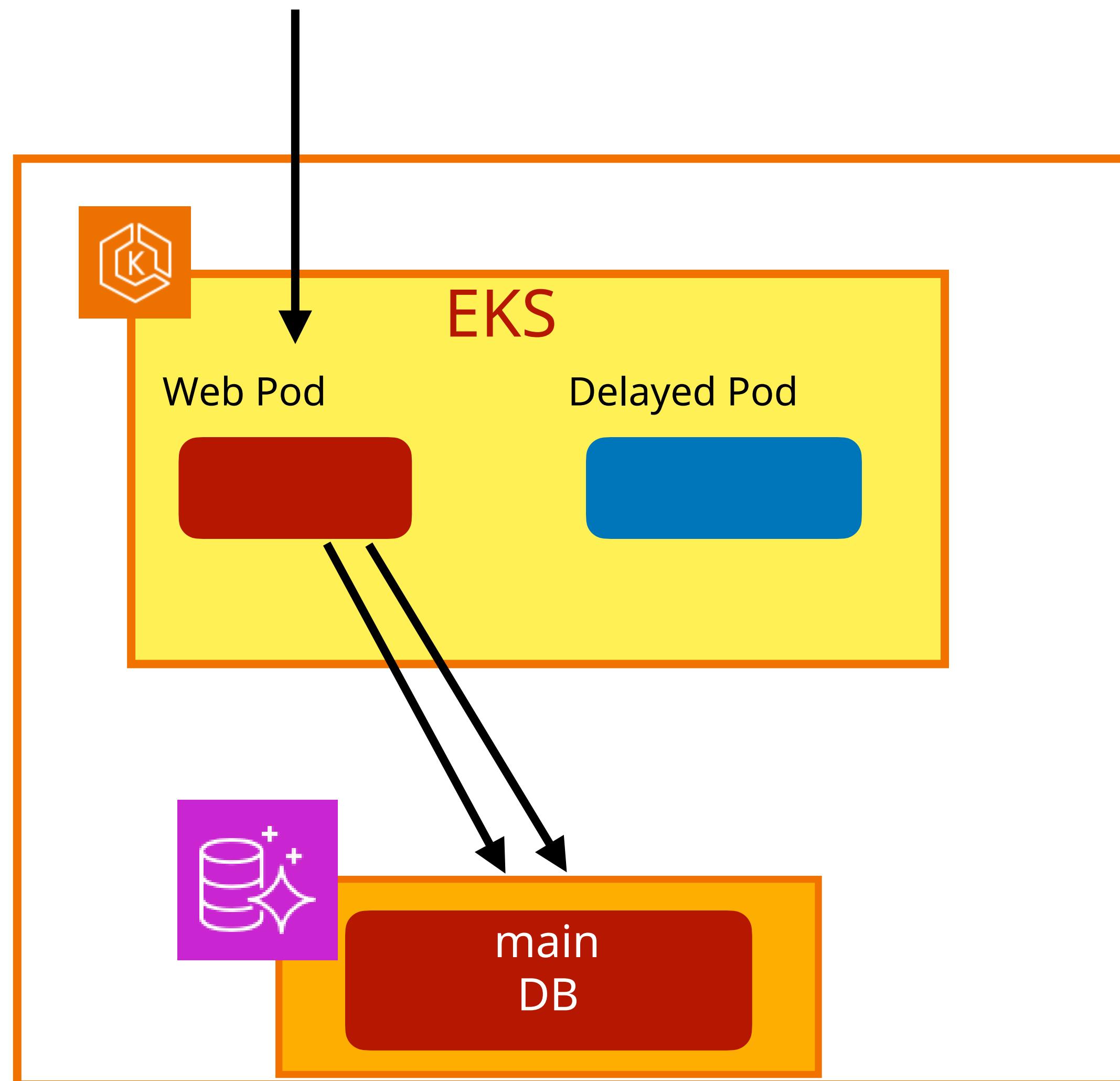


# トランザクション内部に書かれたジョブ問題 2

1. Web にアクセスがある

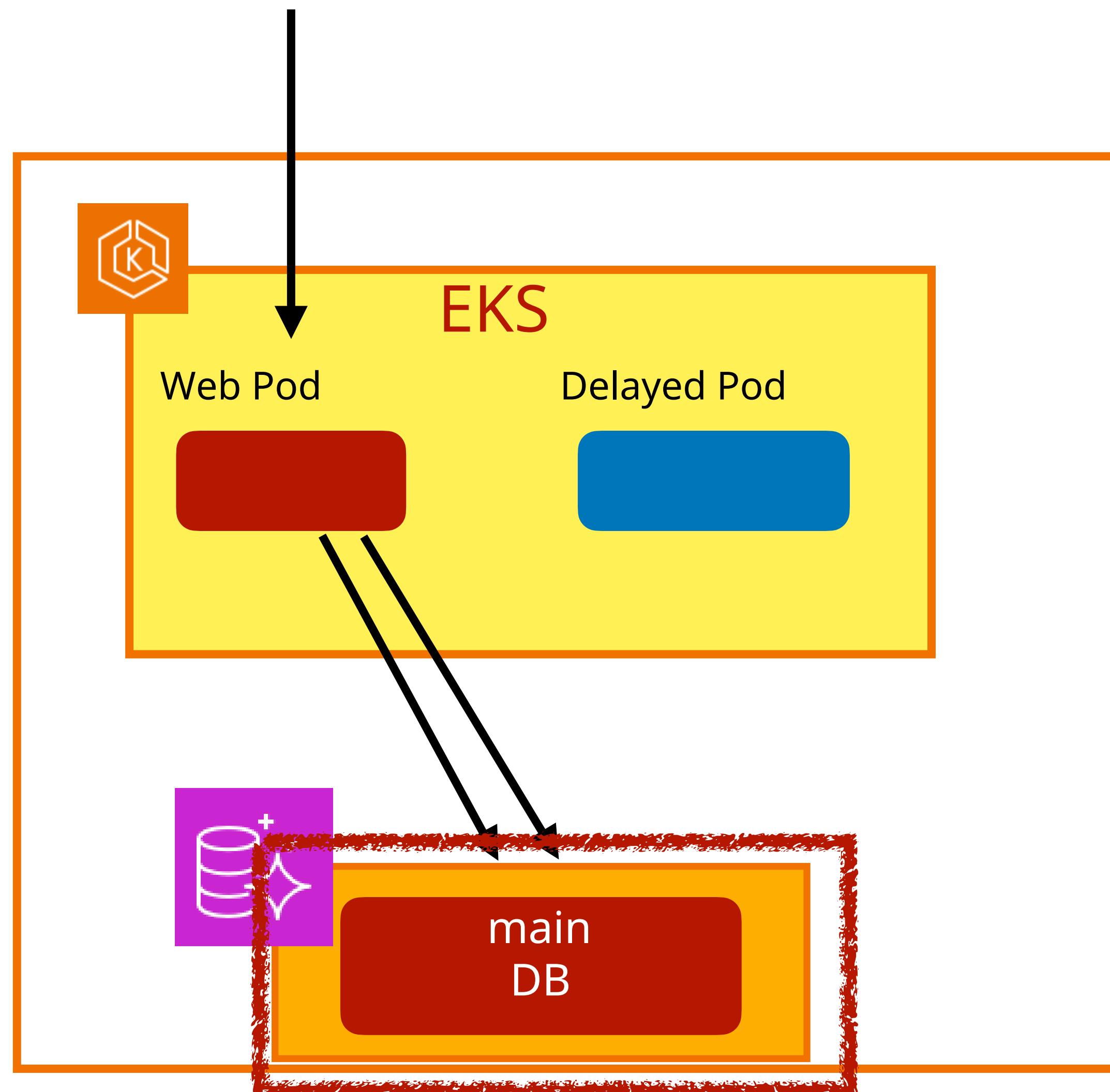


# トランザクション内部に書かれたジョブ問題 2



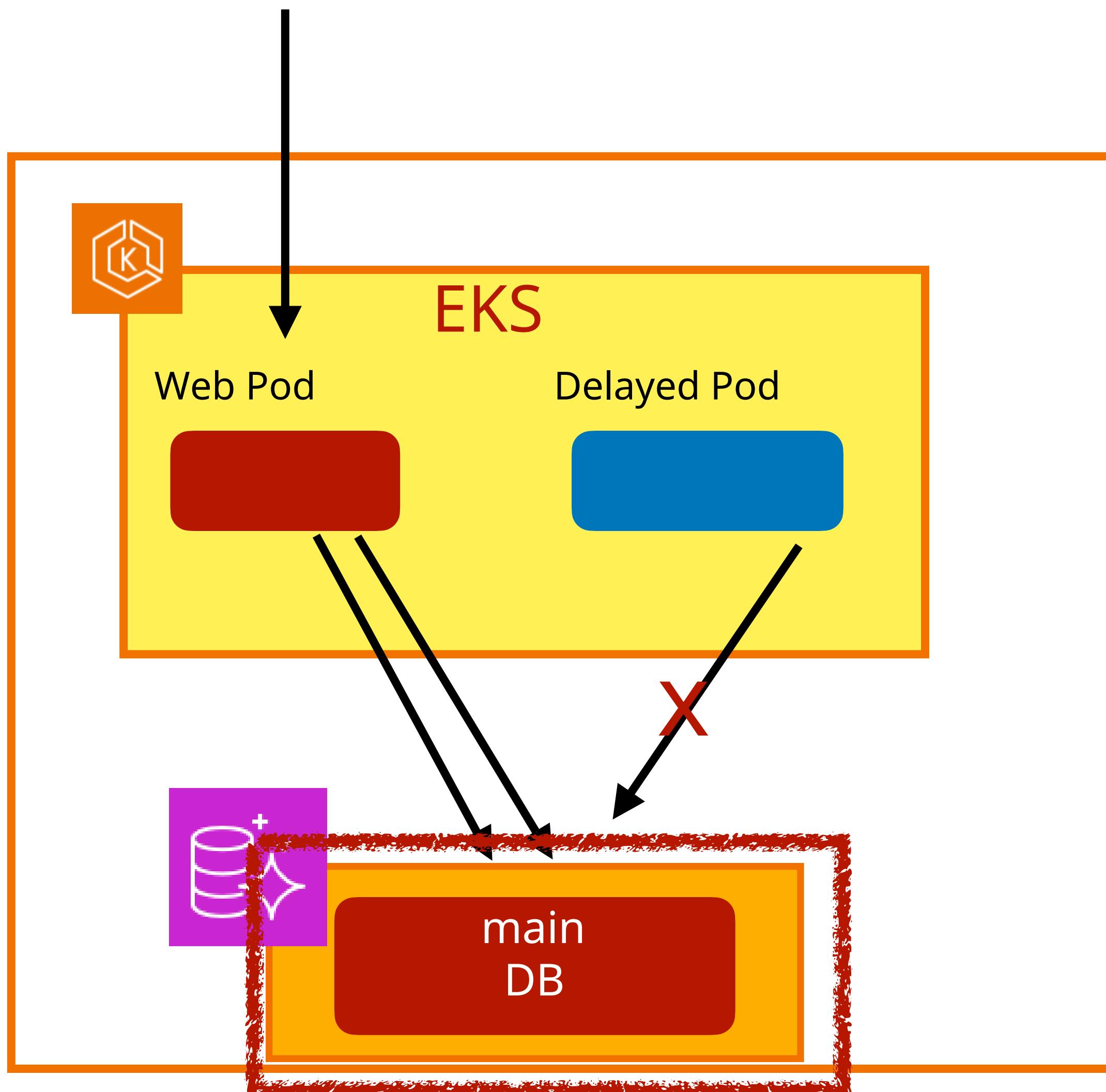
1. Web にアクセスがある
2. トランザクション開始 & ユーザ作成

# トランザクション内部に書かれたジョブ問題 2



1. Web にアクセスがある
2. トランザクション開始 & ユーザ作成
3. ジョブをエンキュー (未コミット)

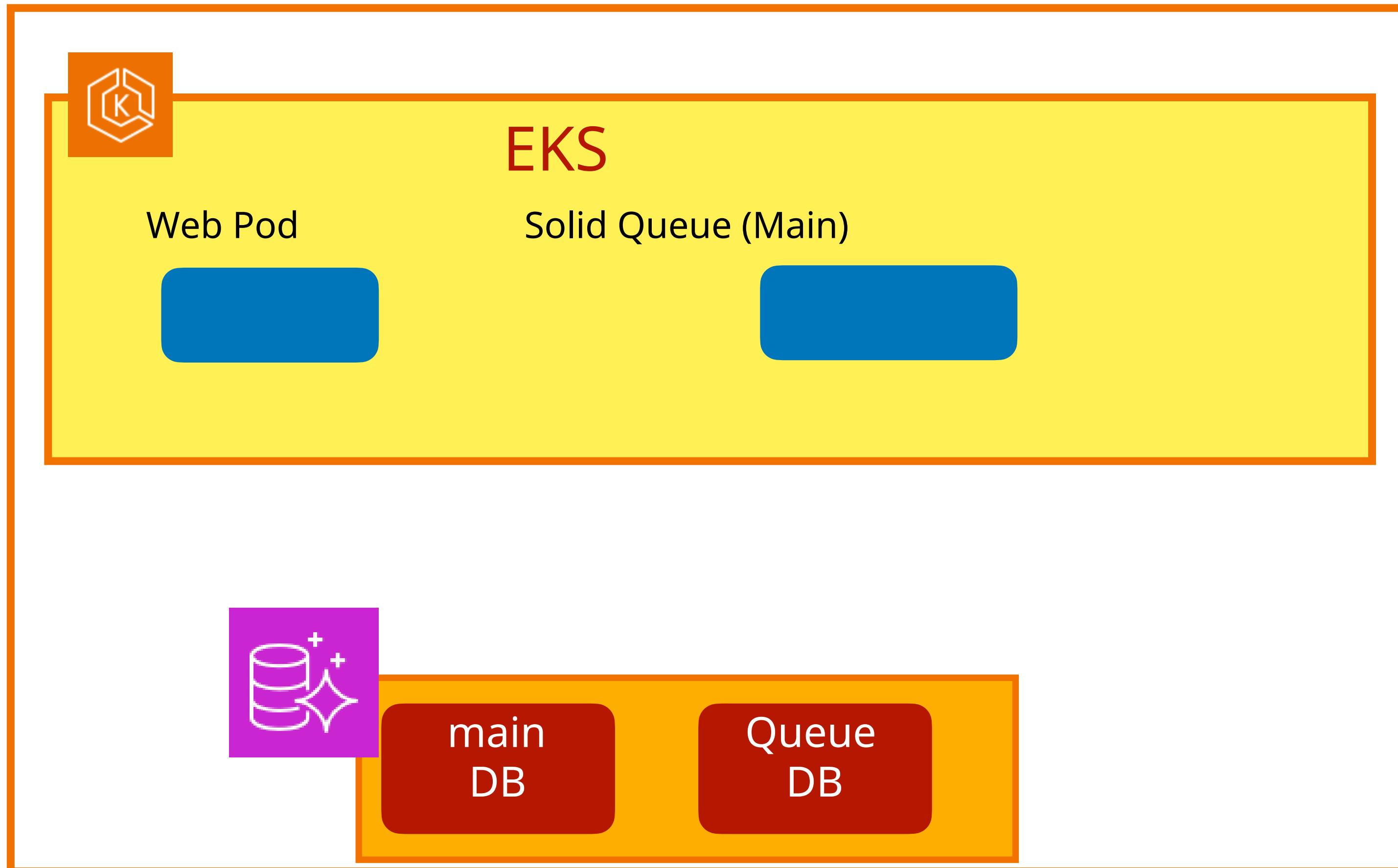
# トランザクション内部に書かれたジョブ問題 2



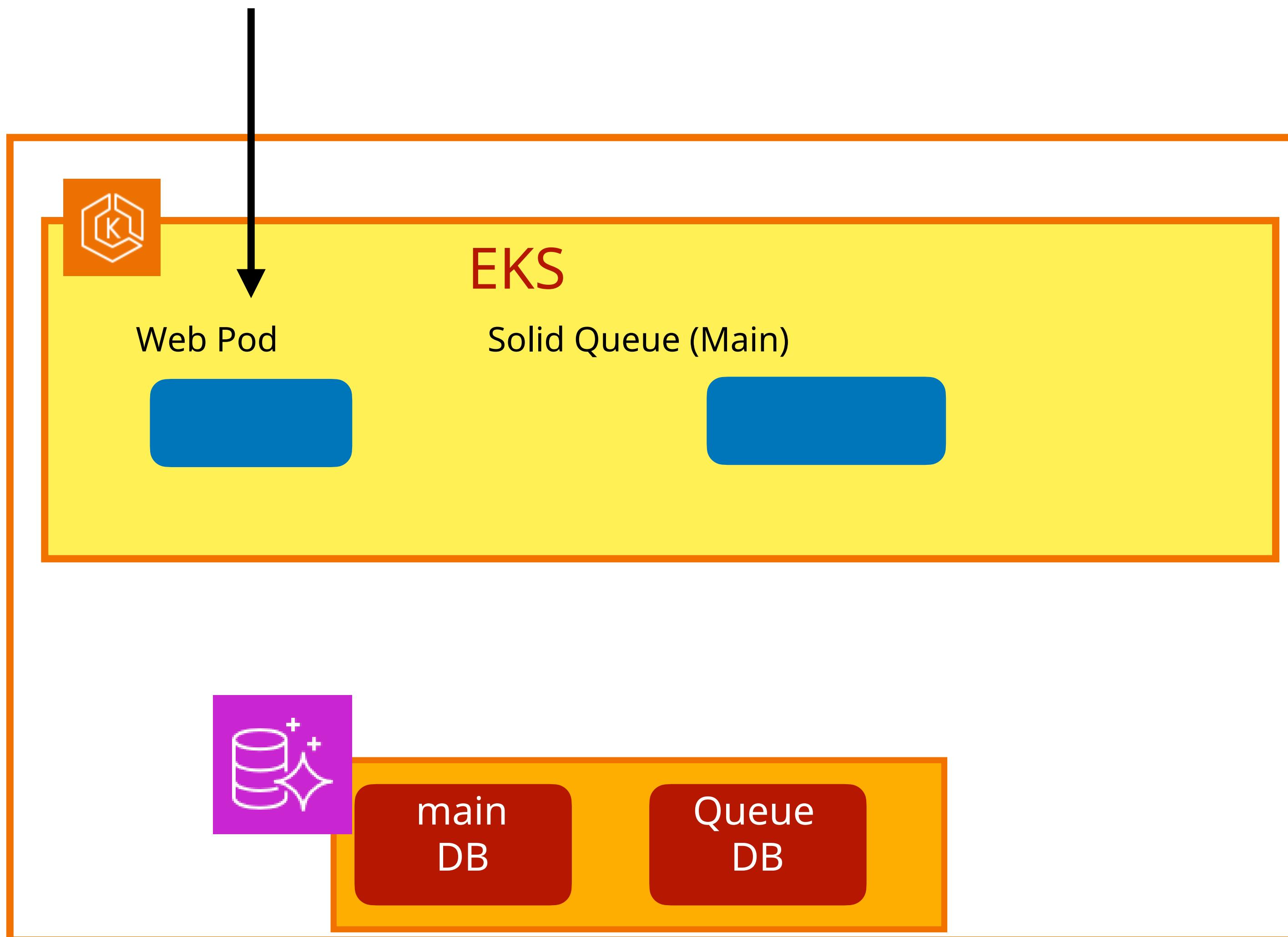
1. Web にアクセスがある
2. トランザクション開始 & ユーザ作成
3. ジョブをエンキュー (未コミット)

コミットしない限り worker は  
ジョブを取得不可能

# トランザクション内部に書かれたジョブ問題 2

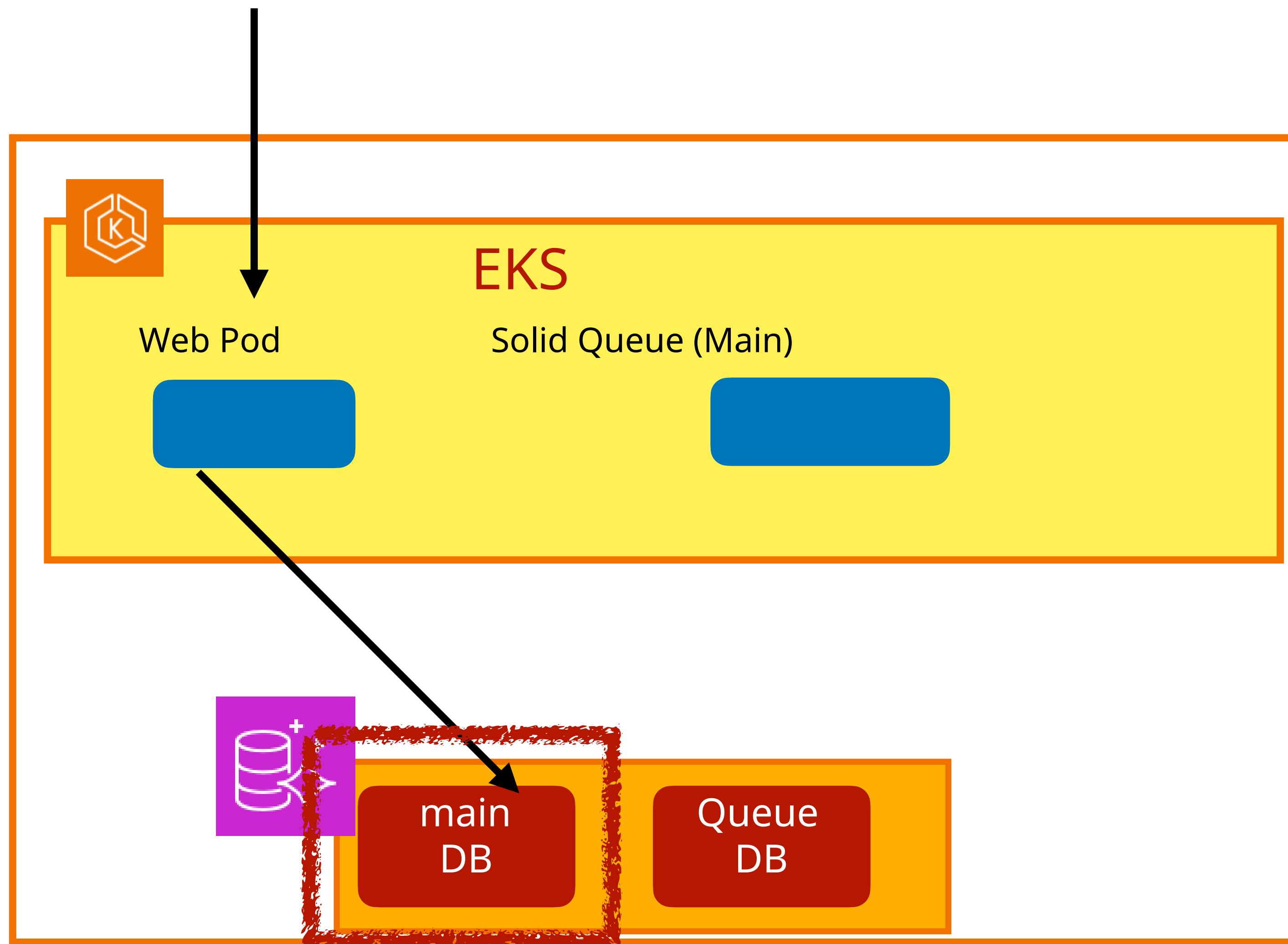


# トランザクション内部に書かれたジョブ問題 2



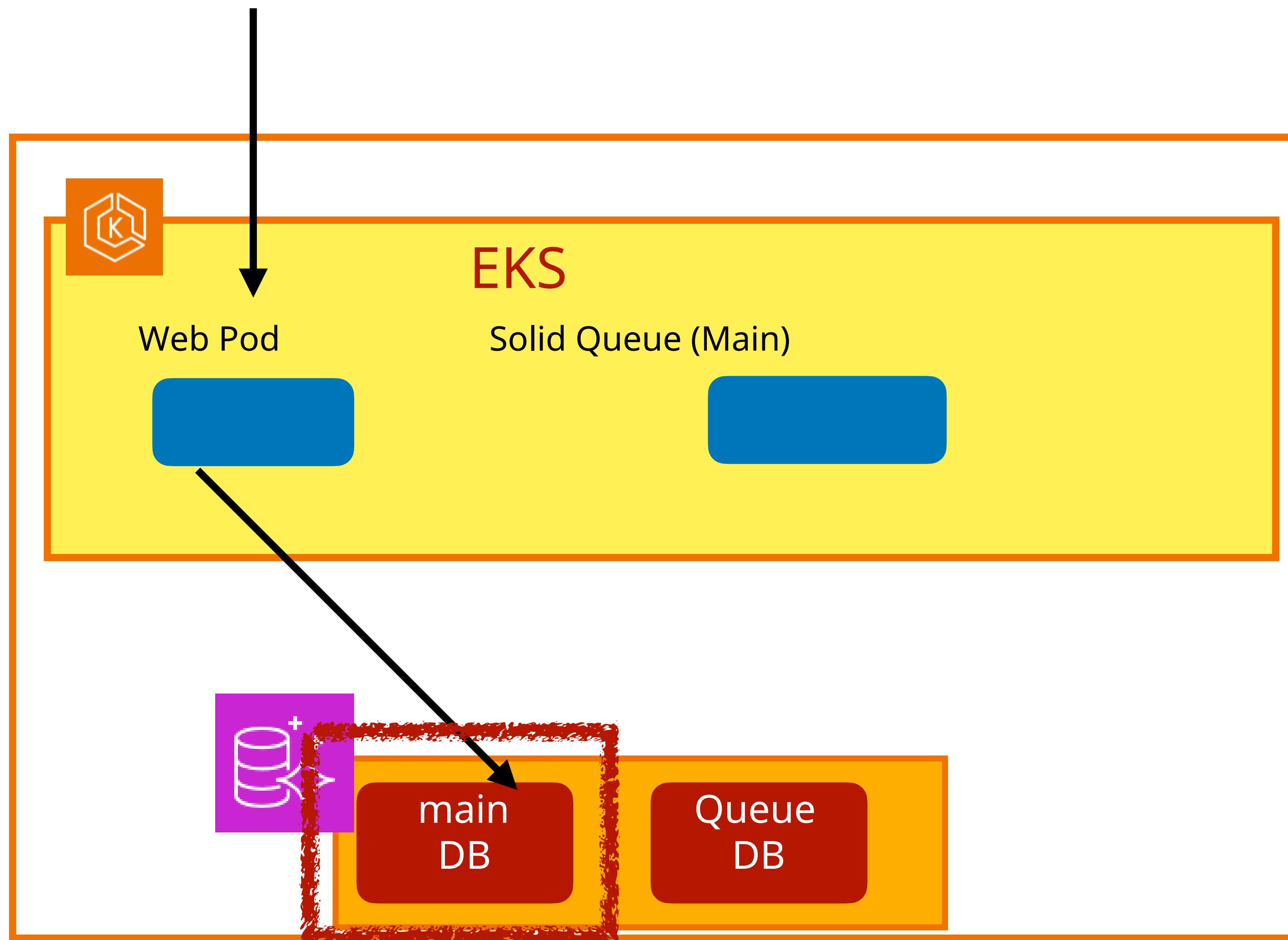
1. Web にアクセスがある

# トランザクション内部に書かれたジョブ問題 2



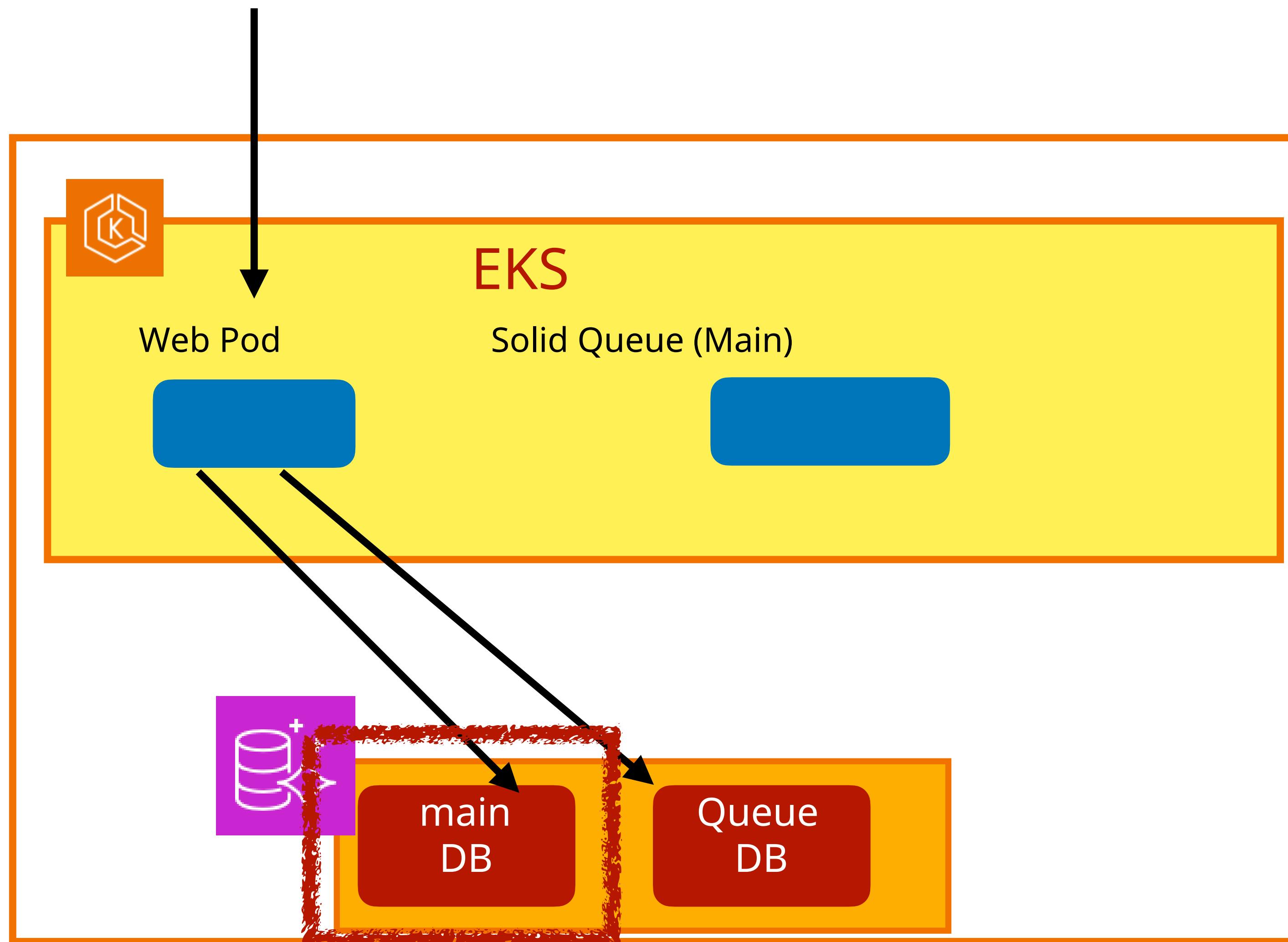
1. Web にアクセスがある
2. トランザクション開始

# トランザクション内部に書かれたジョブ問題 2



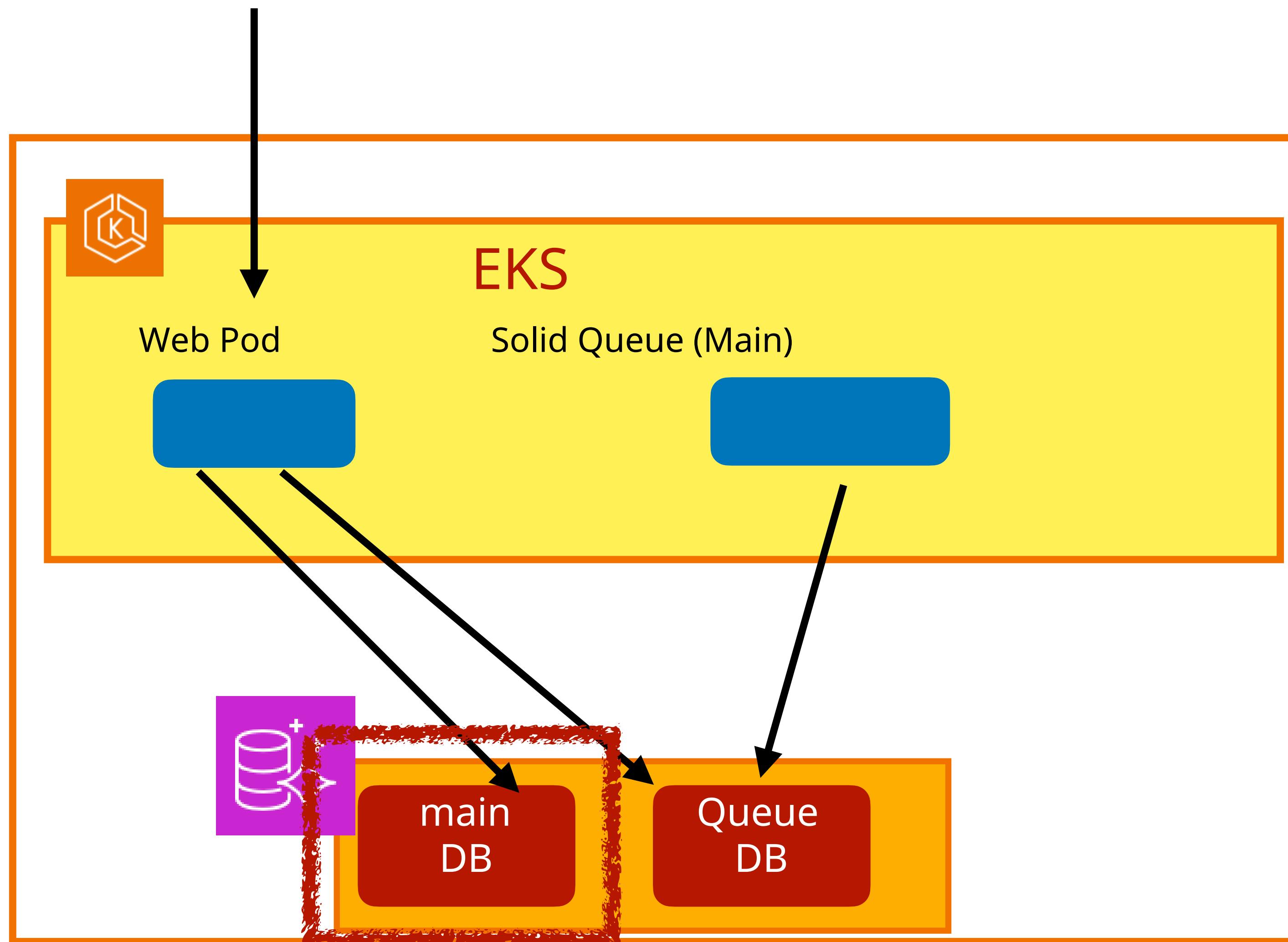
1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）

# トランザクション内部に書かれたジョブ問題 2



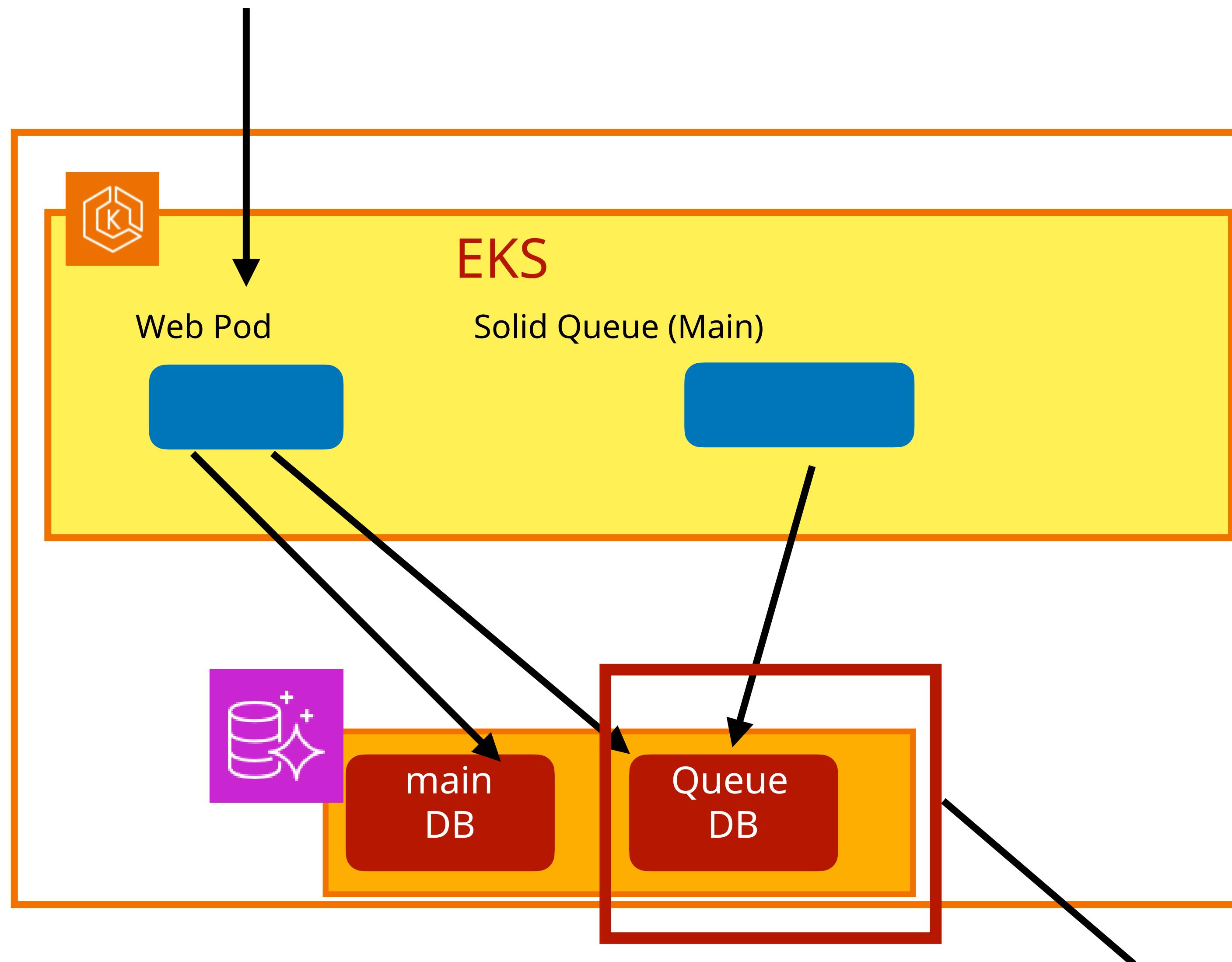
1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）
4. キュー DB にキューがはいる

# トランザクション内部に書かれたジョブ問題 2



1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）
4. キュー DB にキューがはいる
5. Worker が取得に行く

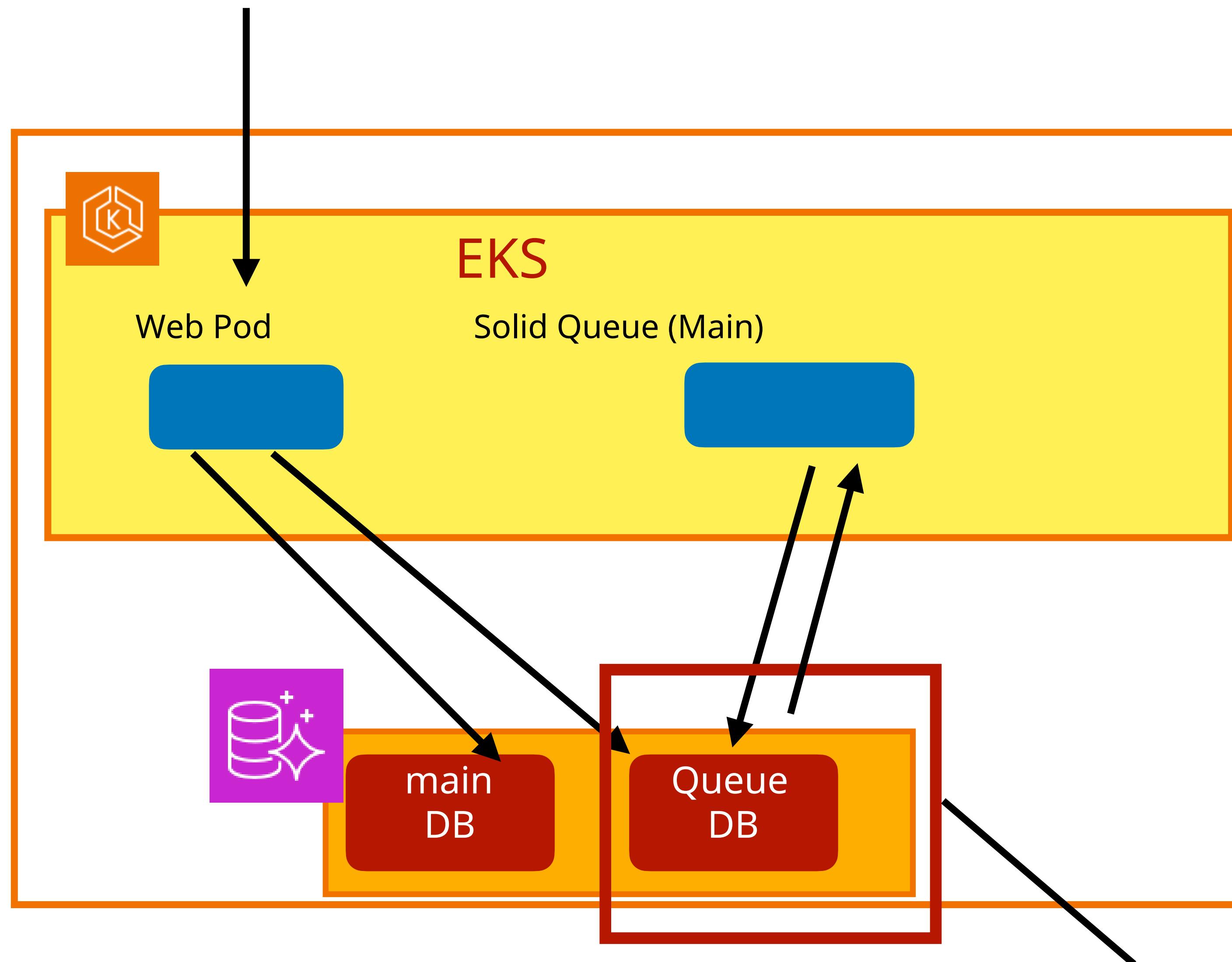
# トランザクション内部に書かれたジョブ問題 2



1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）
4. キュー DB にキューがはいる
5. Worker が取得に行く

main DB のコミット状況は QueueDB は知ったことではない

# トランザクション内部に書かれたジョブ問題 2



1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）
4. キュー DB にキューがはいる
5. Worker が取得に行く
6. コミット前にジョブが実行される

main DB のコミット状況は QueueDB は知ったことではない

# 解決法トランザクション内部に書かれたジョブ問題

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue`after`transaction`commit` を使えるようにする!

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue~~after~~transaction~~commit~~ を使えるようにする!

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue~~after~~transaction~~commit~~ を使えるようにする!

ユーザを DB に保存（未コミット）

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue~~after~~transaction~~commit~~ を使えるようにする!

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

ユーザを DB に保存 (未コミット)

非同期処理がよばれる  
(エンキューされない)

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue~~after~~transaction~~commit~~ を使えるようにする!

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

ユーザを DB に保存 (未コミット)

非同期処理がよばれる  
(エンキューされない)

呼び出し元メソッドで例外発生

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue~~after~~transaction~~commit~~ を使えるようにする!

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

ユーザを DB に保存 (未コミット)

非同期処理がよばれる  
(エンキューされない)

呼び出し元メソッドで例外発生

ユーザは保存されない  
(DB にコミットされない)

# 解決法 トランザクション内部に書かれたジョブ問題

enqueue~~after~~transaction~~commit~~ を使えるようにする!

```
ActiveRecord::Base.transaction do
```

```
  user = User.create!(name: "新しいユーザー")
```

```
  WelcomeEmailJob.perform_later(user.id)
```

```
  raise "Something went wrong!"
```

```
end
```

ユーザを DB に保存 (未コミット)

非同期処理がよばれる  
(エンキューされない)

呼び出し元メソッドで例外発生

ユーザは保存されない  
(DB にコミットされない)

成功しなかったので  
最終的にもエンキューされない!

# トランザクション内部に書かれたジョブ問題

# トランザクション内部に書かれたジョブ問題

Teachme Biz にはあるのか？

## トランザクション内部に書かれたジョブ問題

Teachme Biz にはあるのか？

before<sub>xxx</sub> / after<sub>xxxx</sub> に書かれたもの

# トランザクション内部に書かれたジョブ問題

Teachme Biz にはあるのか?

before<sub>xxx</sub> / after<sub>xxxx</sub> に書かれたもの

明示的に transaction で囲まれたもの

## トランザクション内部に書かれたジョブ問題

Teachme Biz にはあるのか？

before<sub>xxx</sub> / after<sub>xxxx</sub> に書かれたもの

明示的に transaction で囲まれたもの

→ **Rails あげるか！！**

# あげました!

## Rails7.1 -> Rails7.2 upgrade #16459

[Edit](#)[Code](#)

 Merged srockstyle merged 34 commits into master from develop/rails7.2\_upgrade  on May 1

 Conversation 27

 Commits 34

 Checks 16

 Files changed 12

+212 -153 



srockstyle commented on Mar 21 · edited

### 課題URL

- [SRE Backlog](#)

### WHAT

Rails7.1などをRails7.2にします。

前回

[#15454](#)

### WHY

#### Reviewers

 valerauko

 shishi

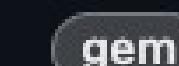
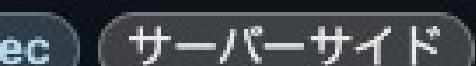
 wind-up-bird

 corrupt952

#### Assignees

 srockstyle

#### Labels

 gem  migration  rspec  サーバーサイド



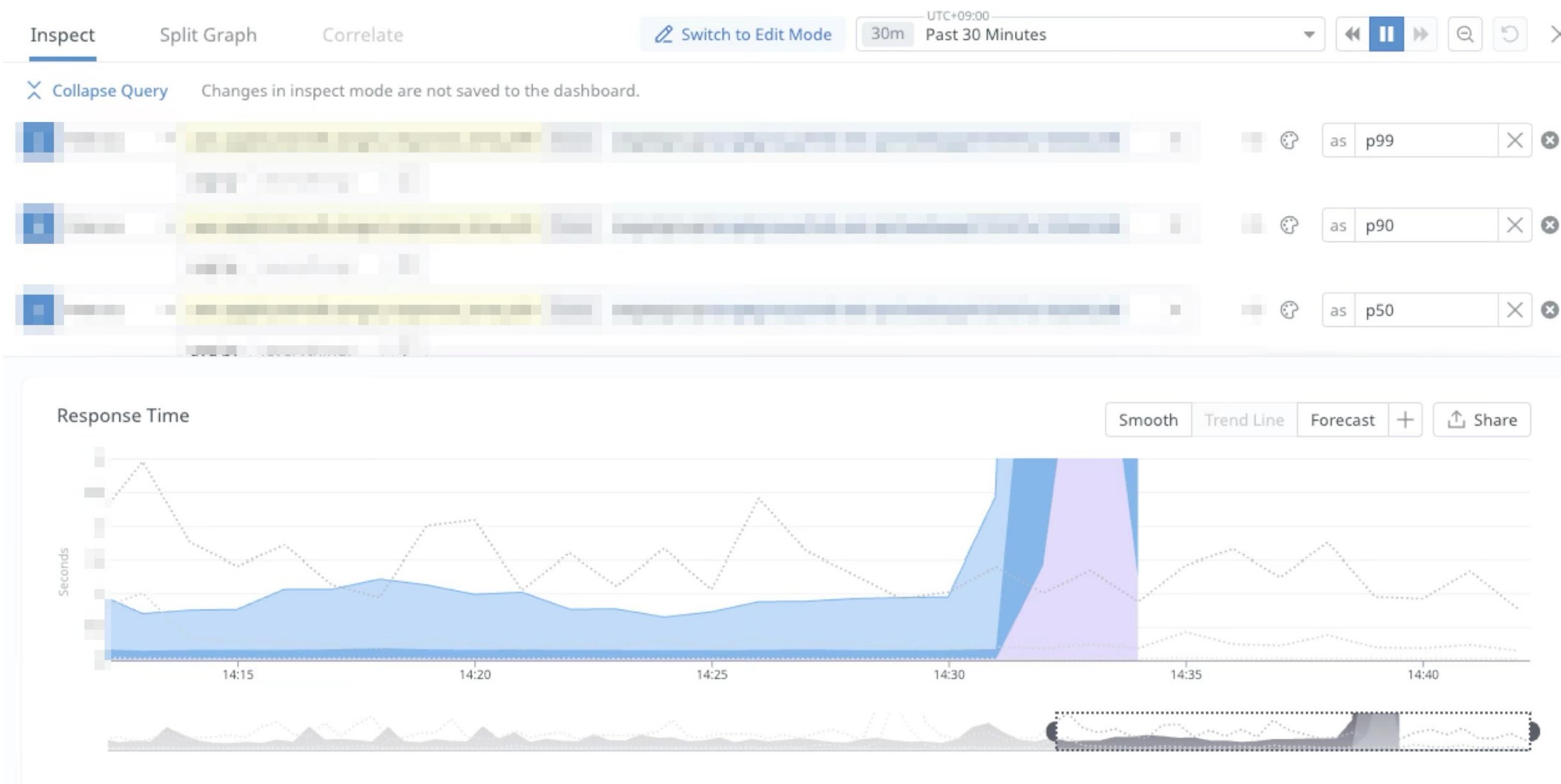
Transaction 完了前に  
エンキューさせて  
実行して欲しい場合はどうするの?

設計を見直して

実行して欲しい場合はどうするの?

# **Solid Queue 速すぎて過負荷になった問題**

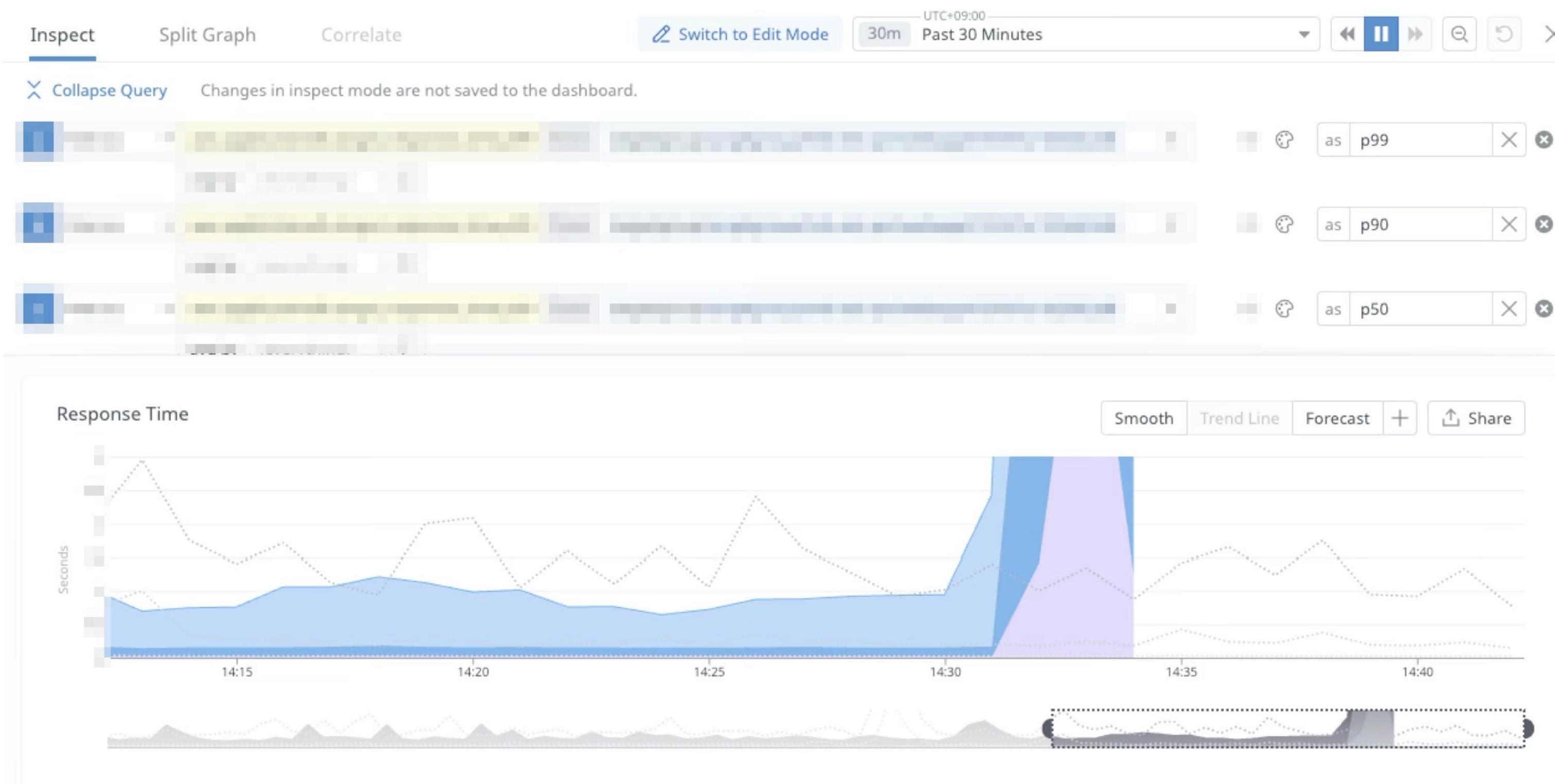
# Solid Queue 速すぎて過負荷になった問題



# Solid Queue 速すぎて過負荷になった問題

## SolidQueue リリースから数日後

重くない?



# Solid Queue 速すぎて過負荷になった問題

SolidQueue リリースから数日後

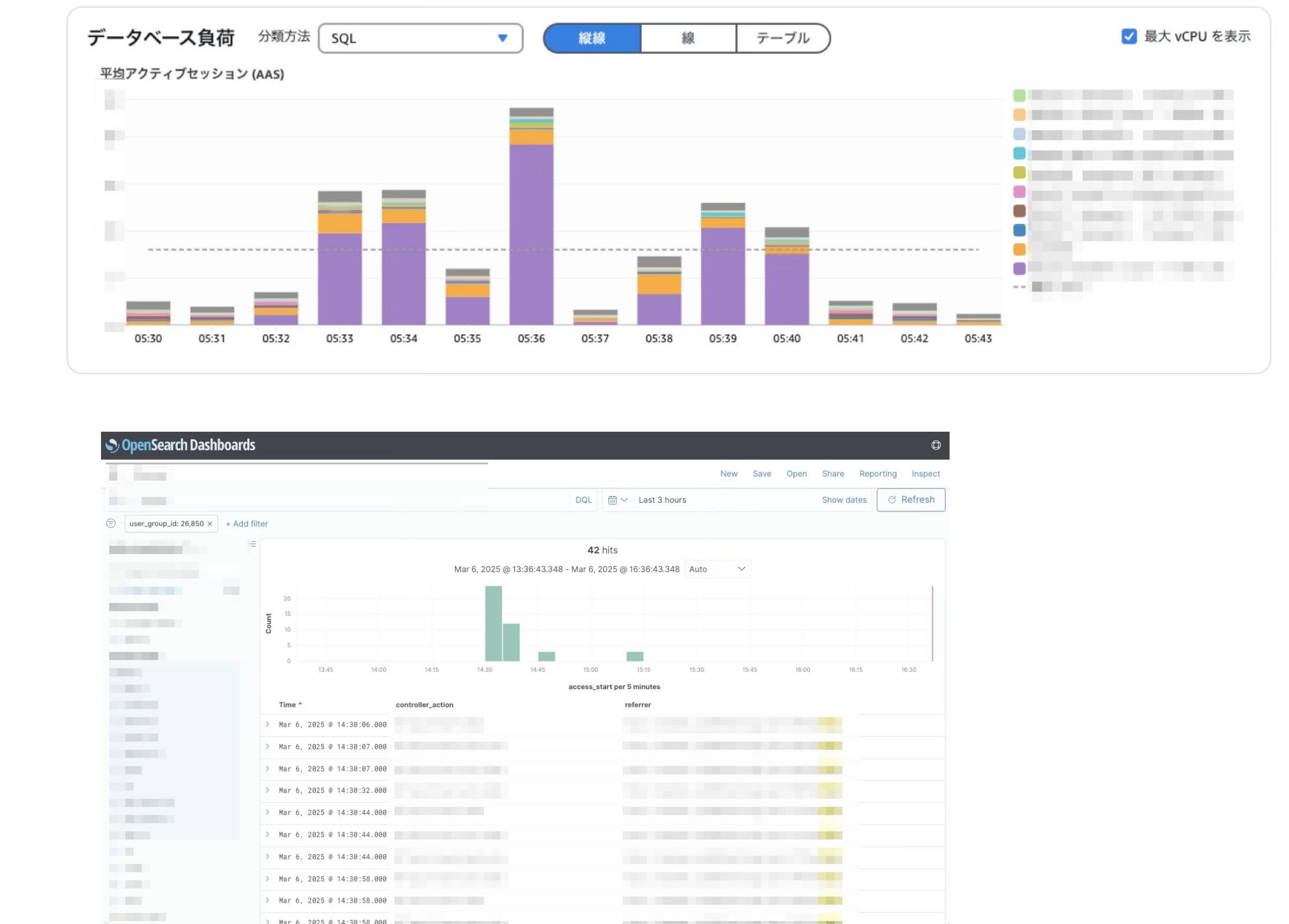
重くない?



というかアクセスすると重くない?

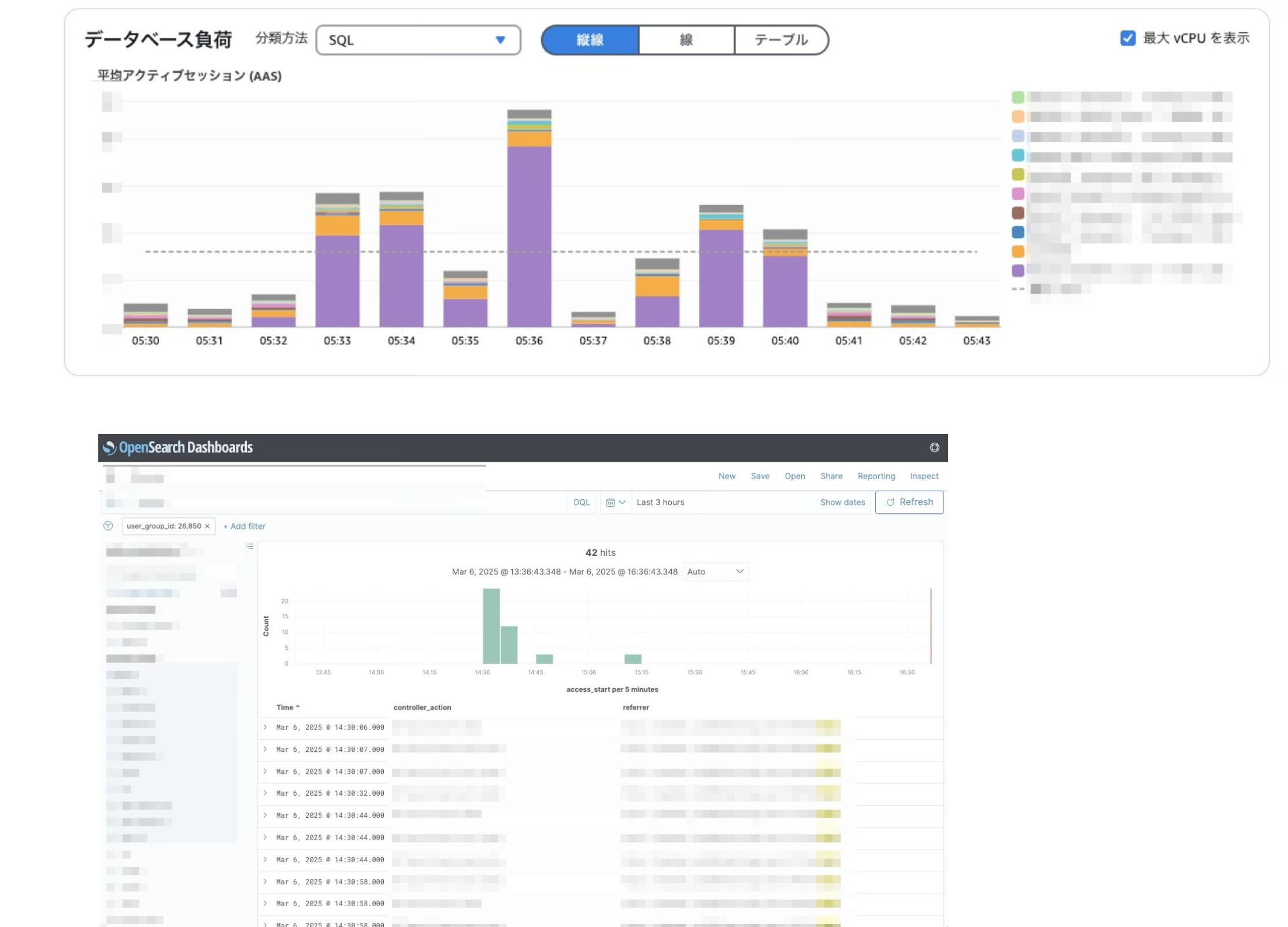
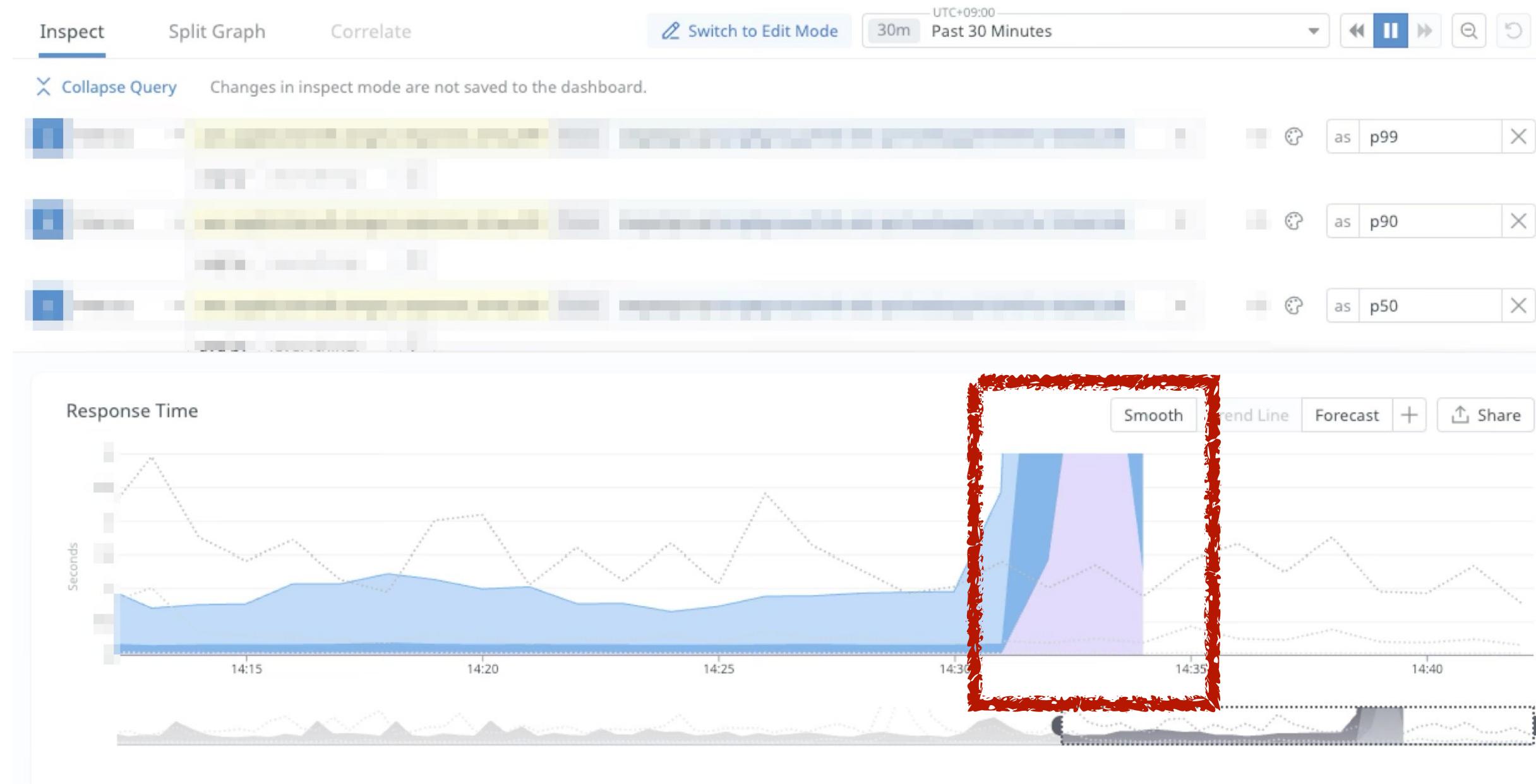
# Solid Queue 速すぎて過負荷になった問題

## 原因是 DB の高負荷



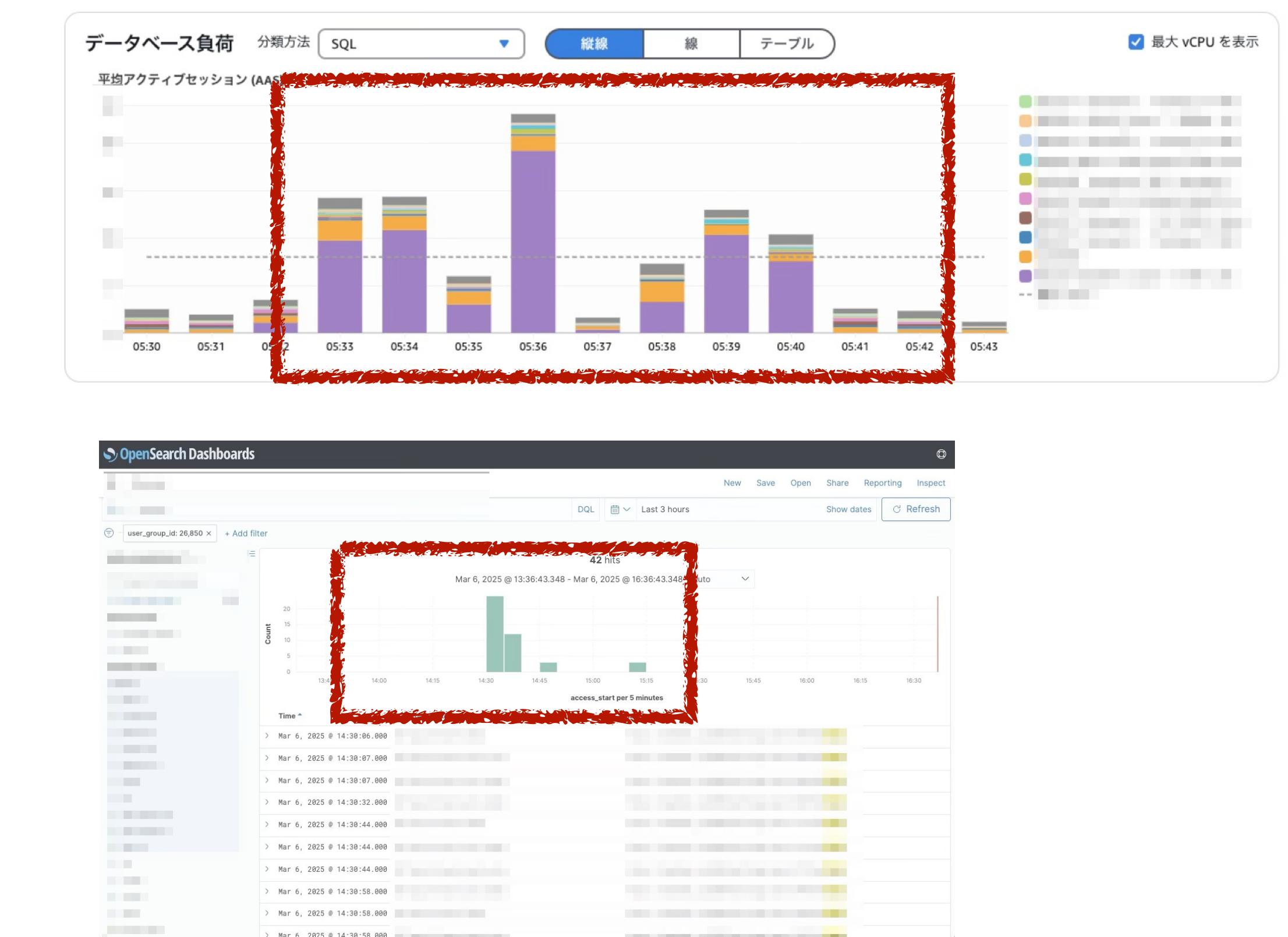
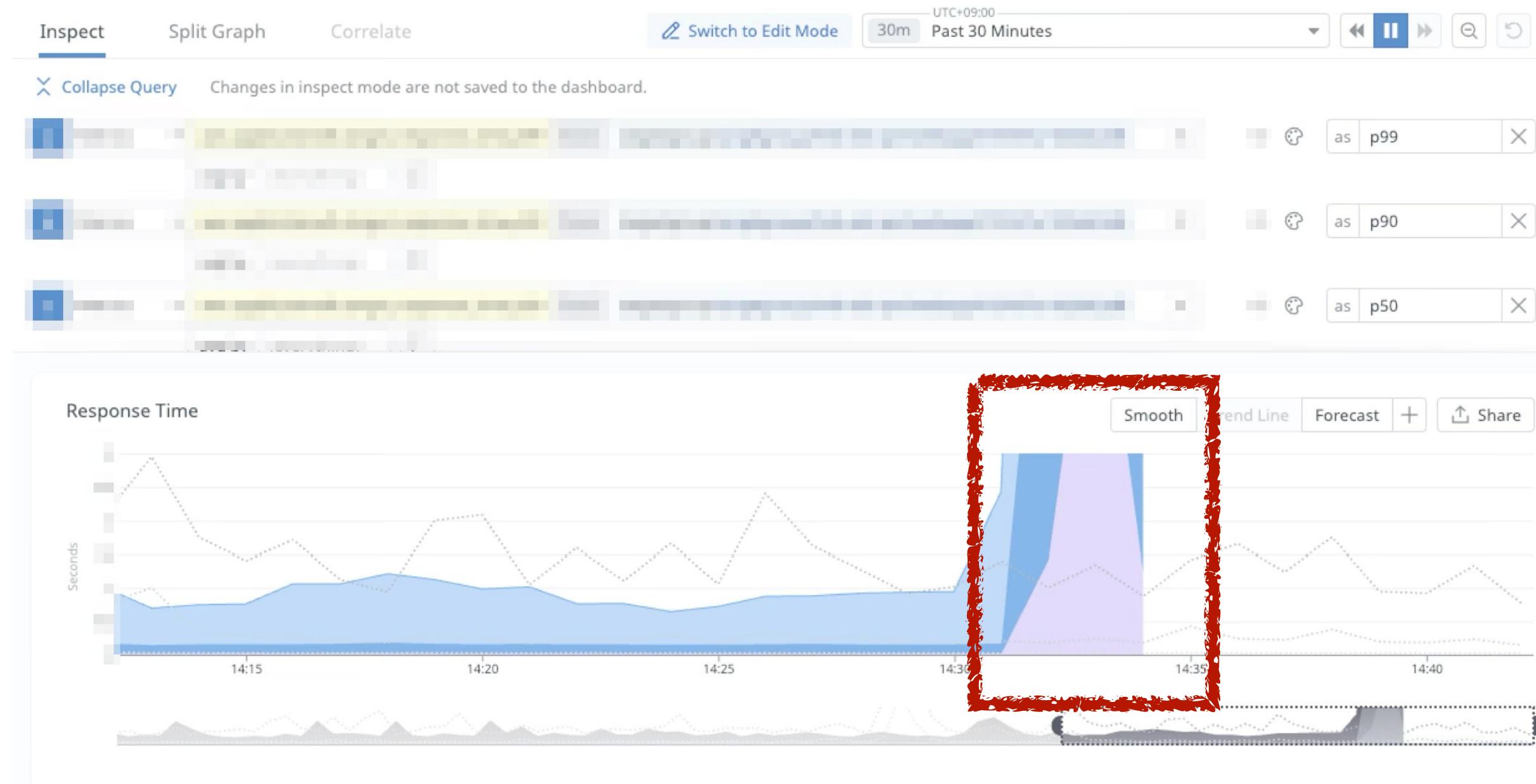
# Solid Queue 速すぎて過負荷になった問題

## 原因是 DB の高負荷



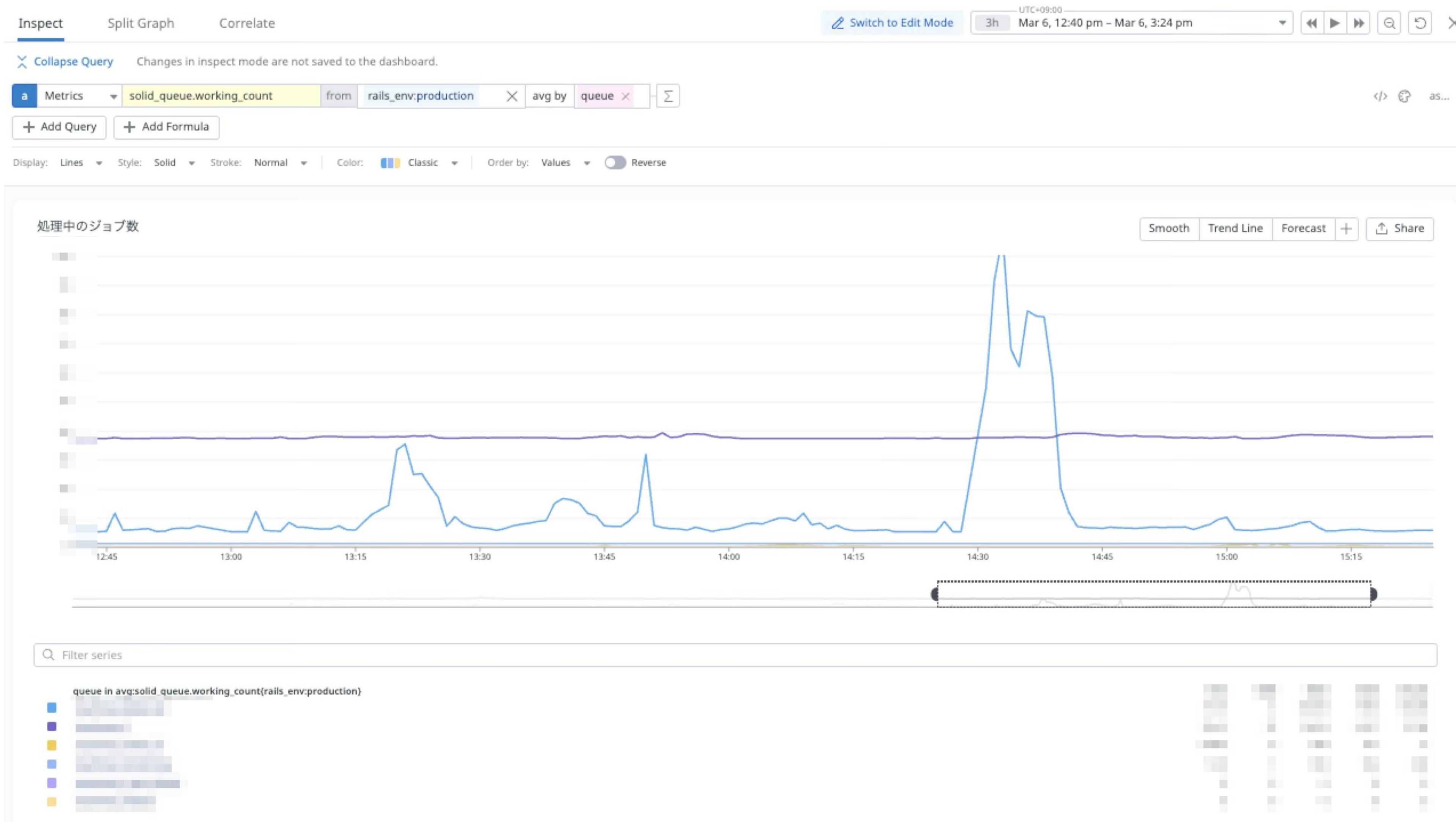
# Solid Queue 速すぎて過負荷になった問題

## 原因是 DB の高負荷



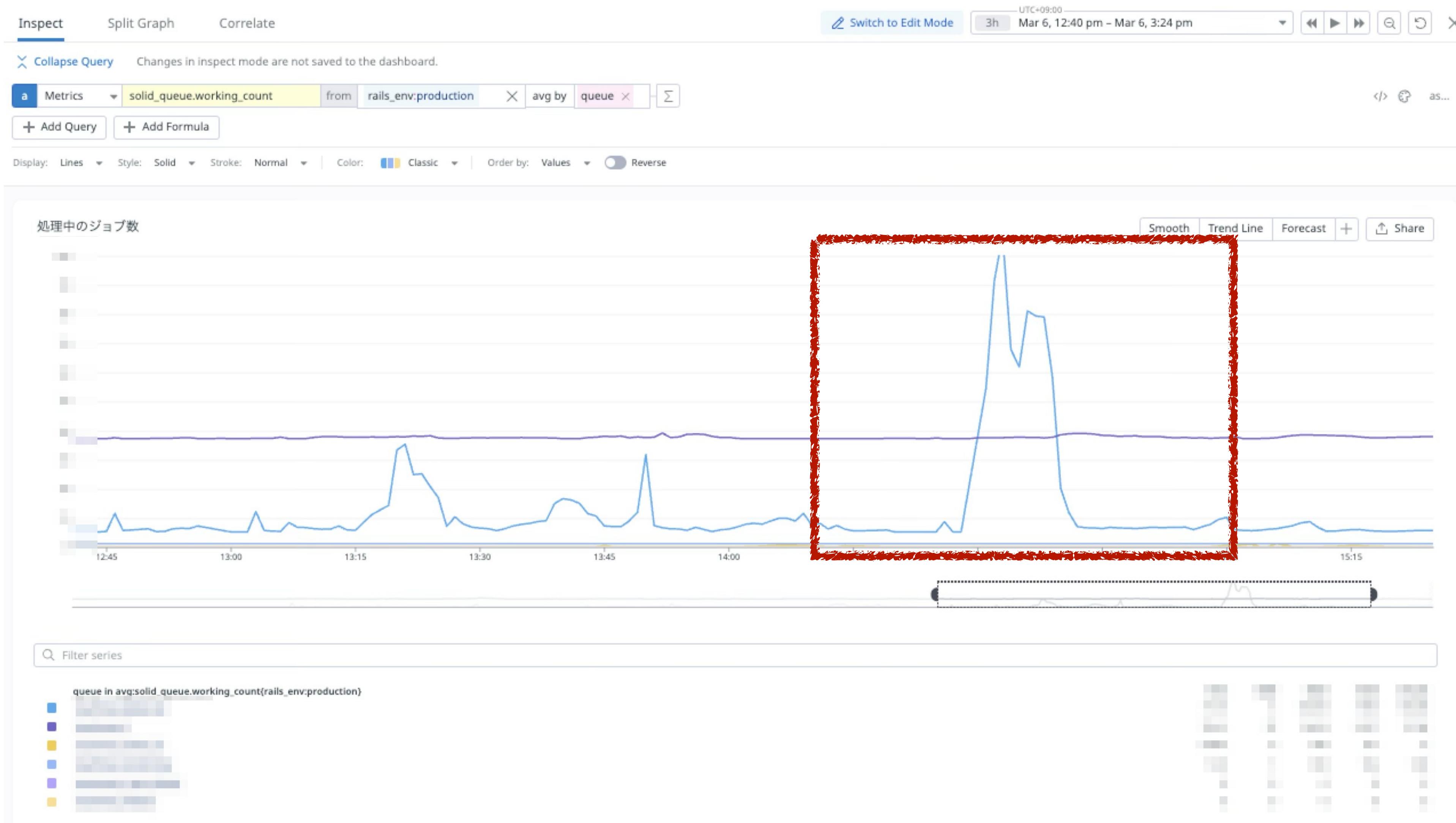
# Solid Queue 速すぎて過負荷になった問題

# 原因は ...

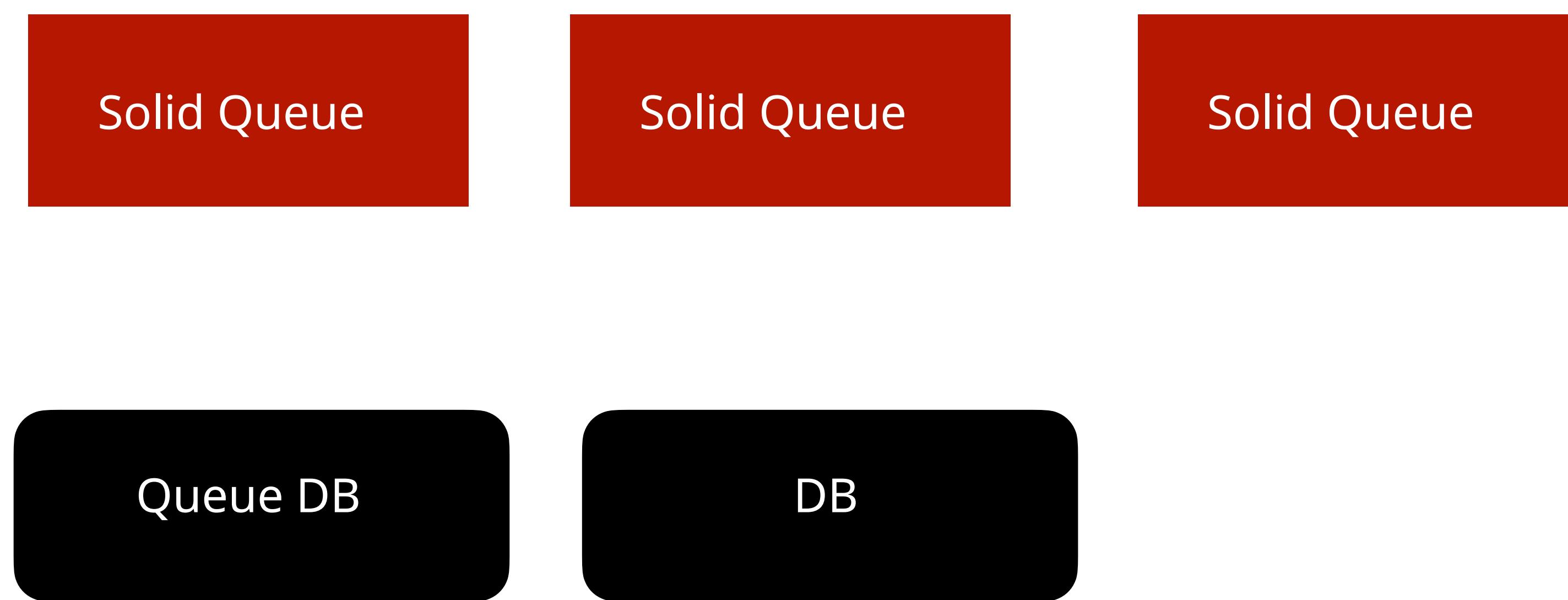


# Solid Queue 速すぎて過負荷になった問題

# 原因は ...

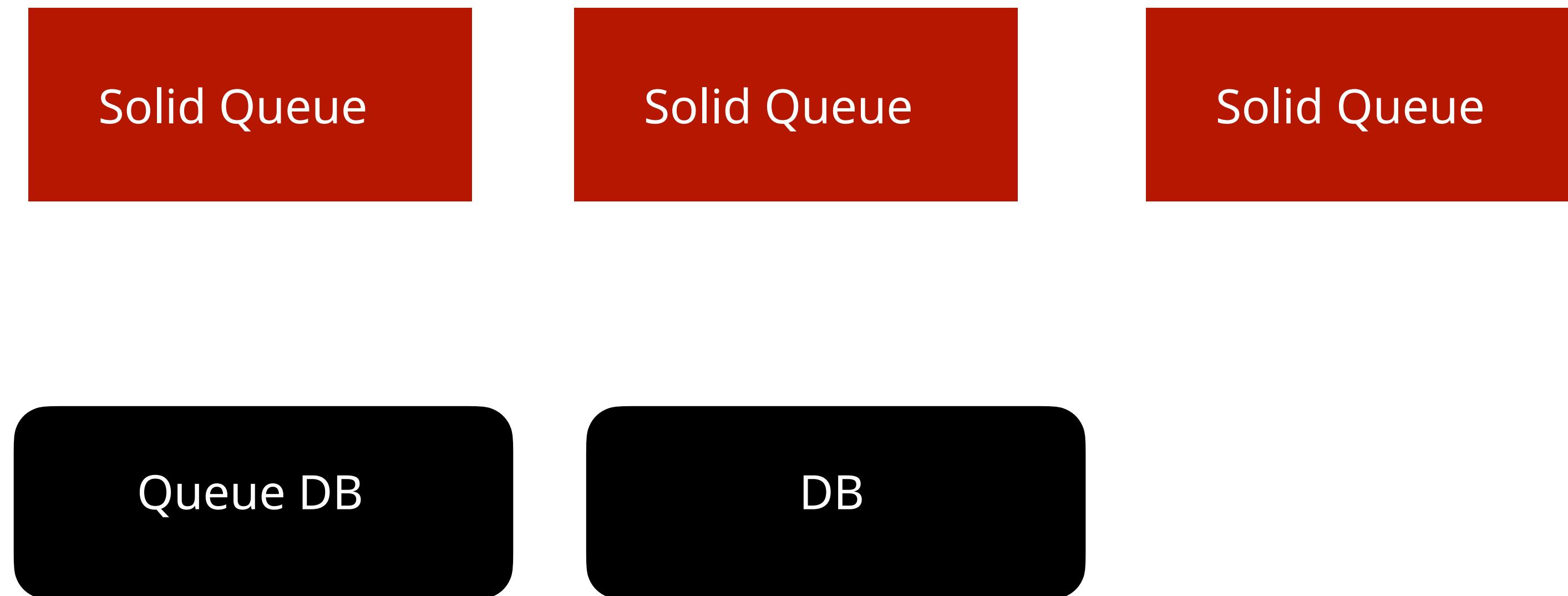


# Solid Queue 速すぎて過負荷になった問題



# Solid Queue 速すぎて過負荷になった問題

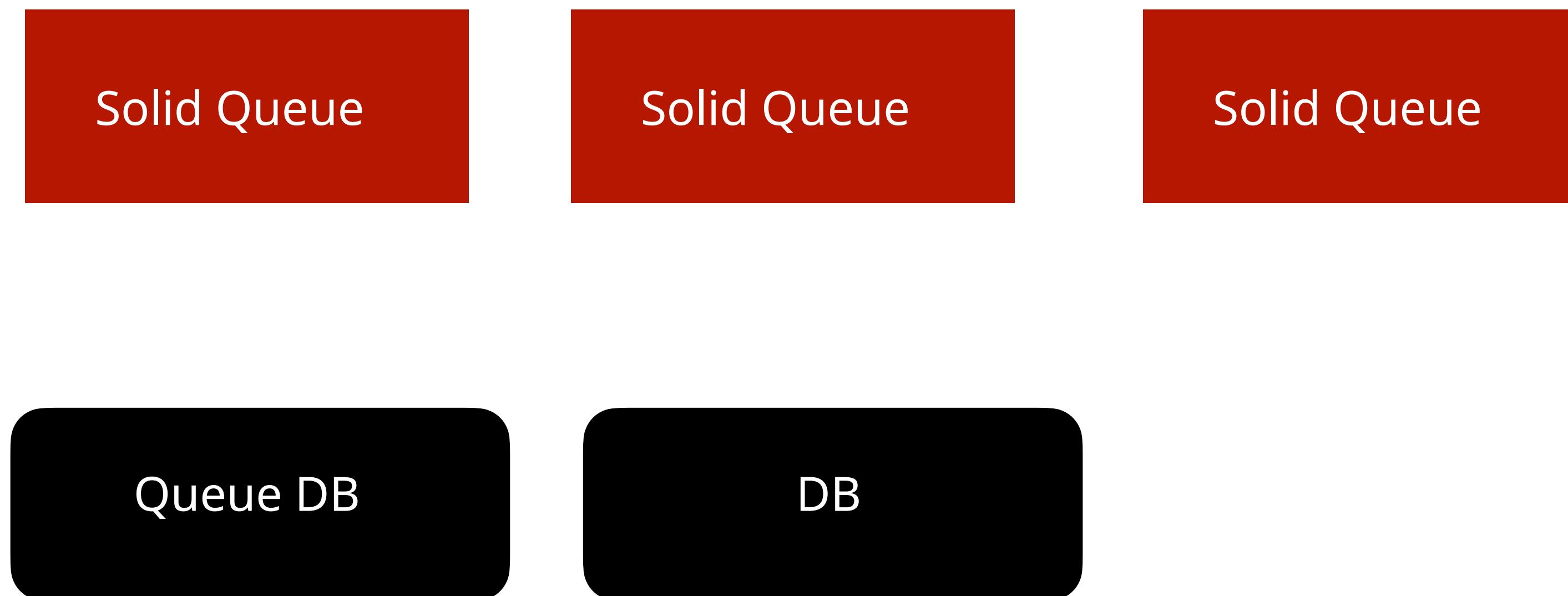
原因是重量級ジョブが大量に並列実行されるようになったこと



# Solid Queue 速すぎて過負荷になった問題

原因是重量級ジョブが大量に並列実行されるようになったこと

すごいクエリのジョブがいっぱいきたので処理したろか

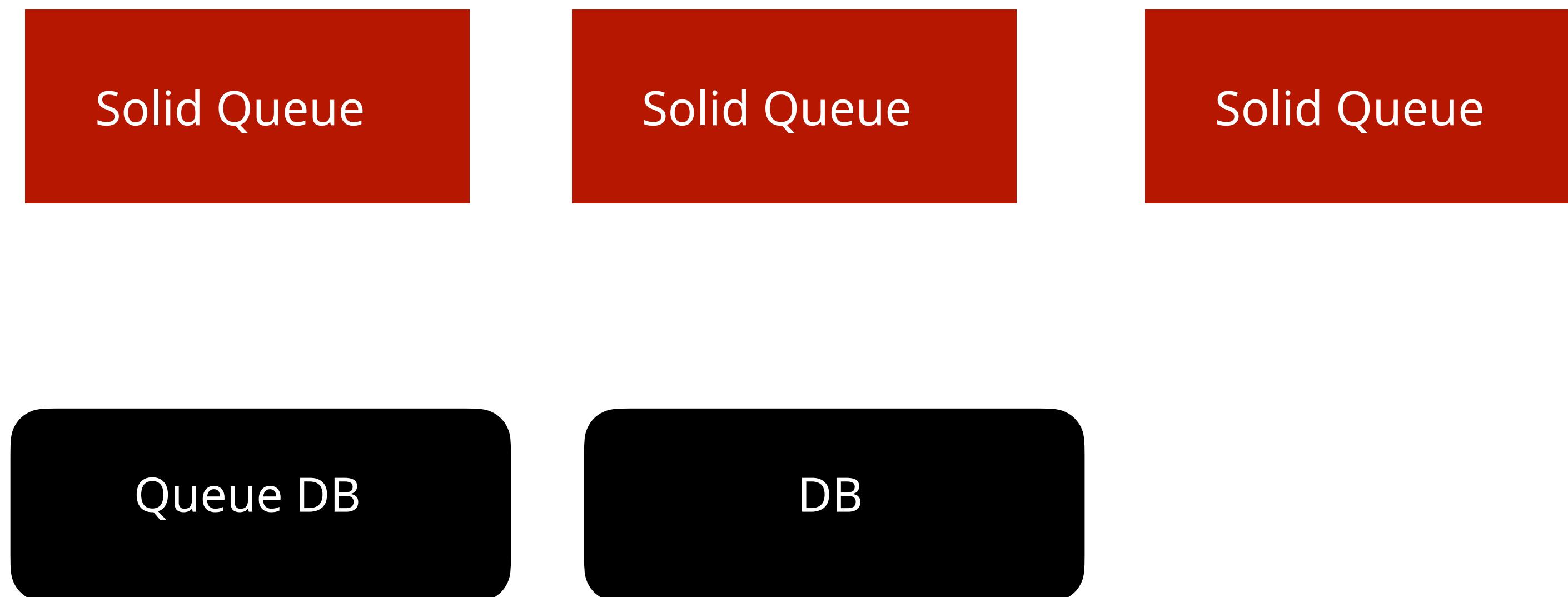


# Solid Queue 速すぎて過負荷になった問題

原因是重量級ジョブが大量に並列実行されるようになったこと

すごいクエリのジョブがいっぱいきたので処理したろか

処理 Worker とプロセスも上がり Solid Queue 処理速度は Delayed の 40 倍や!



# Solid Queue 速すぎて過負荷になった問題

## 原因是重量級ジョブが大量に並列実行されるようになったこと

すごいクエリのジョブがいっぱいきたので処理したろか

## 処理実行!

Solid Queue

Solid Queue

Solid Queue

Queue DB

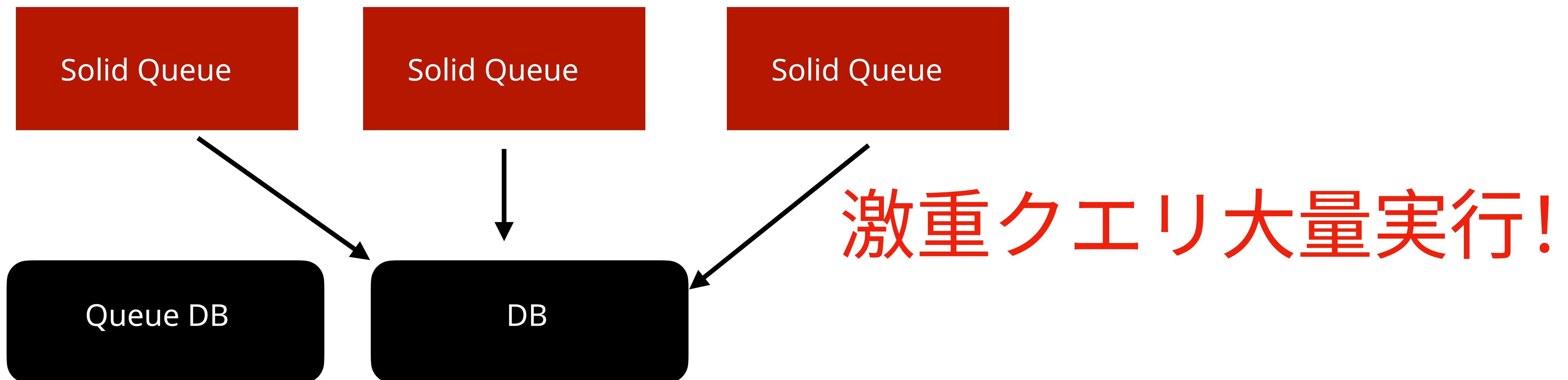
DB

# Solid Queue 速すぎて過負荷になった問題

原因是重量級ジョブが大量に並列実行されたようになったこと

すごいクエリのジョブがいっぱいきたので処理したろか

処理実行!

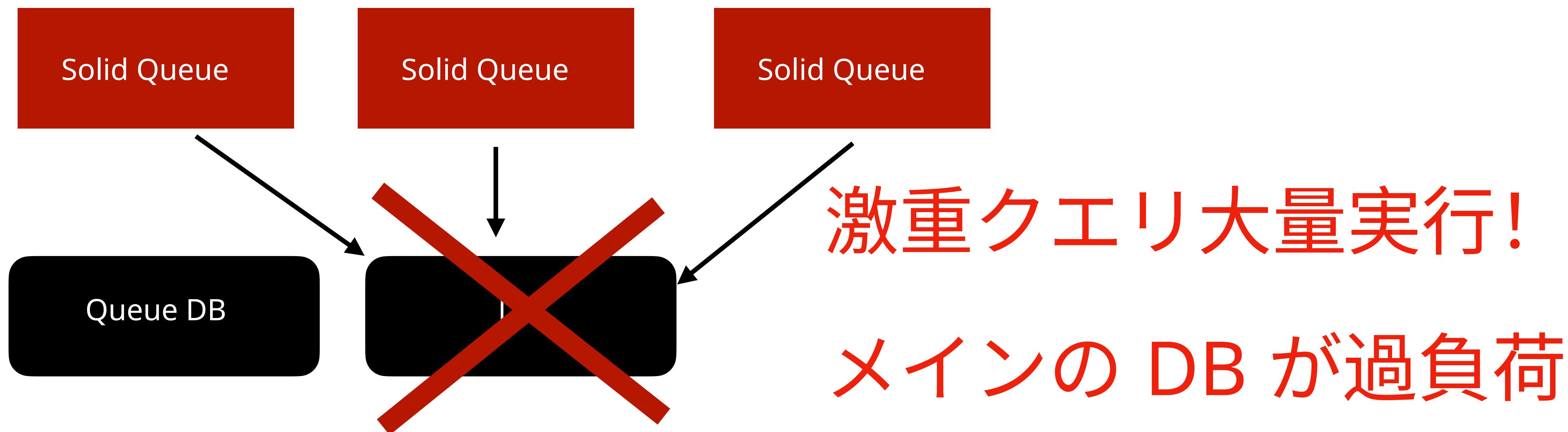


# Solid Queue 速すぎて過負荷になった問題

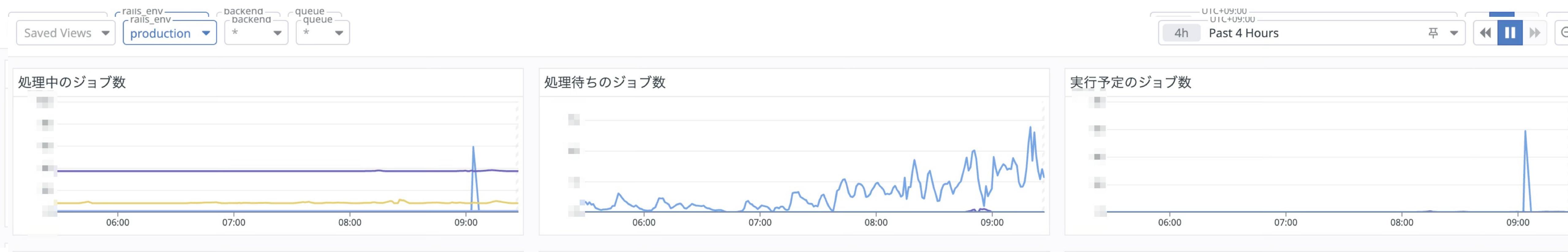
原因是重量級ジョブが大量に並列実行されたようになったこと

すごいクエリのジョブがいっぱいきたので処理したろか

処理実行!

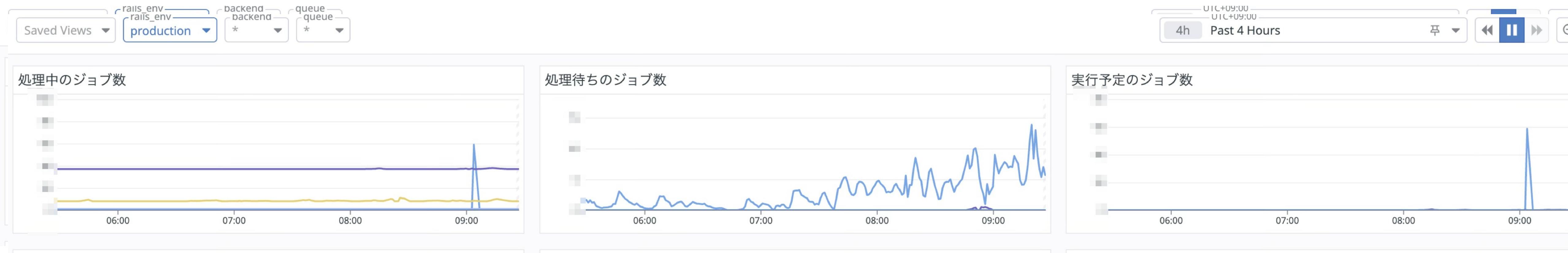


# Solid Queue 速すぎて過負荷になった問題



# Solid Queue 速すぎて過負荷になった問題

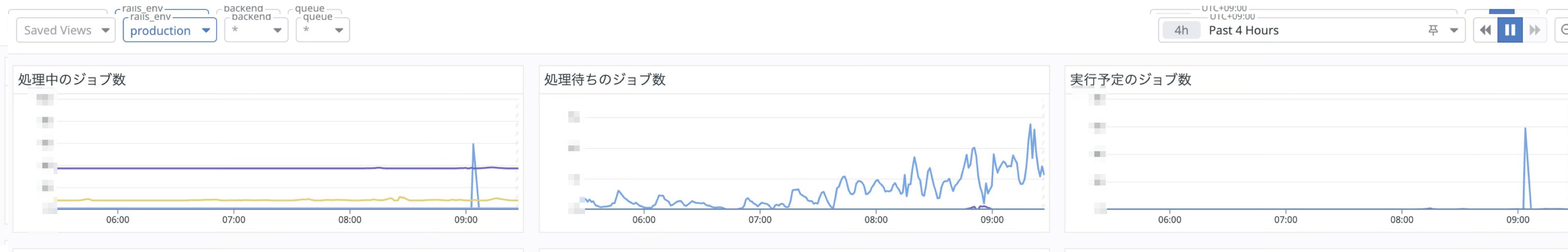
## 復旧優先のため、重いやつ専用 Worker / 専用キューを作った



シングルプロセス / シングルスレッドの流れにあっても並列しない用

# Solid Queue 速すぎて過負荷になった問題

復旧優先のため、重いやつ専用 Worker / 専用キューを作った

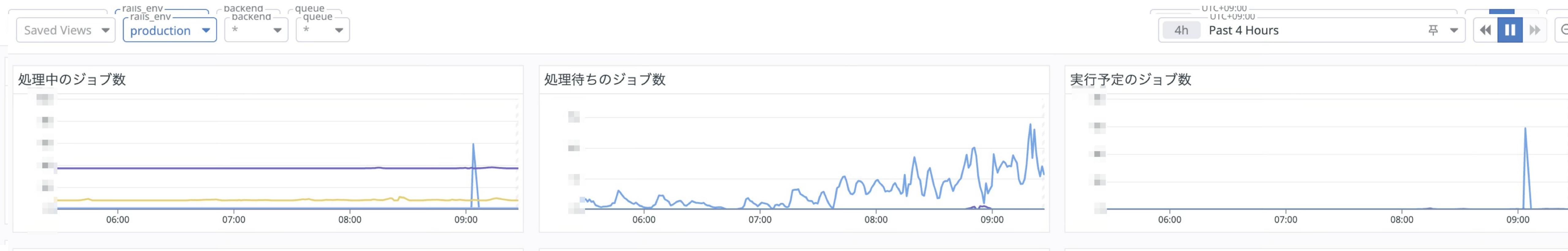


シングルプロセス / シングルスレッドの流れてきても並列しない用

Solid Queue のいいところを全部潰した対応をしました

# Solid Queue 速すぎて過負荷になった問題

復旧優先のため、重いやつ専用 Worker / 専用キューを作った



シングルプロセス / シングルスレッドの流れてきても並列しない用

Solid Queue のいいところを全部潰した対応をしました

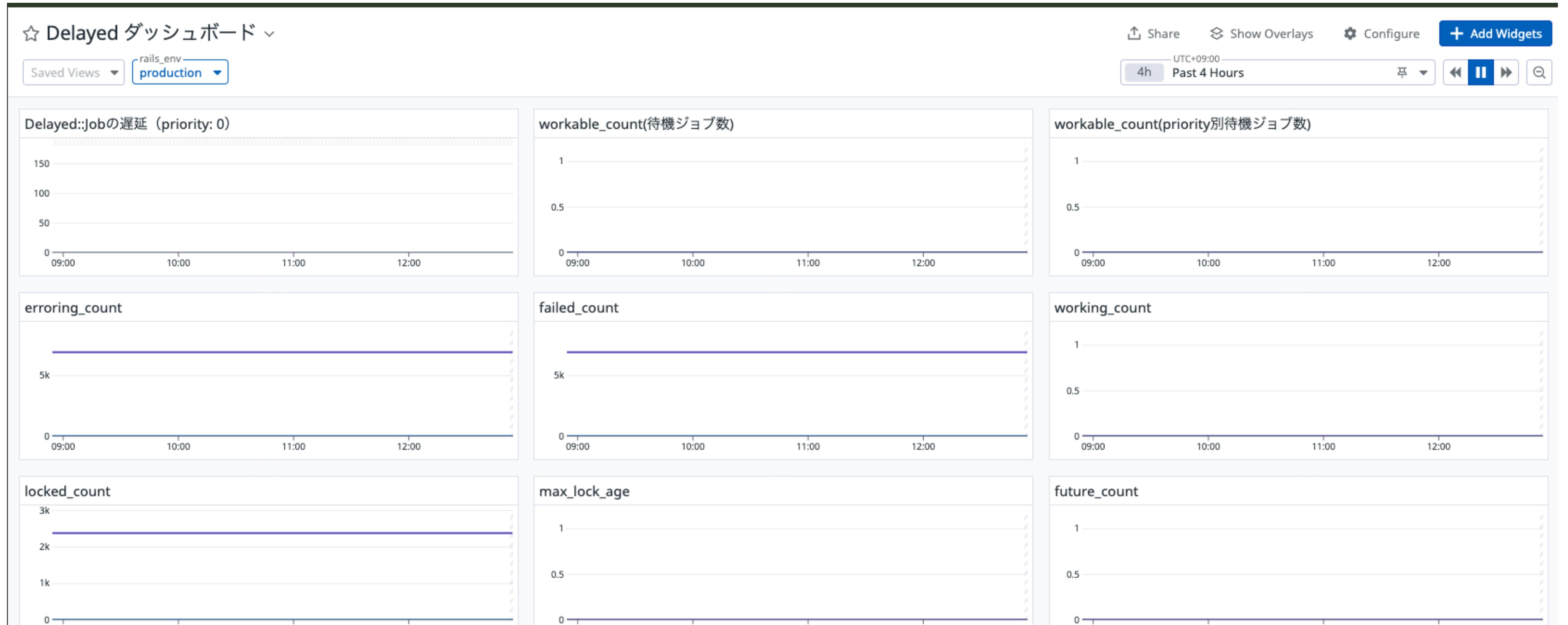
**Rosa さんごめんなさい**

# Migration Requirements

-  **.delay メソッドの撲滅**
-  **ActiveJob で明示的に Delayed を指定**
-  **SolidQueue 設定**
-  **監視**
-  **SolidQueue を利用するように全 Job を変更**
-  **gem uninstall delayed**

# Datadog / Delayed 監視ダッシュボード

# Datadog / Delayed 監視ダッシュボード



# Datadog / Delayed 監視ダッシュボード



# 【非同期バックエンド移行】Delayedとその関連基盤コードを削除 #17389

Edit `Code`

↳ Merged srockstyle merged 10 commits into master from develop/remove-delayed-job on Jun 23

Conversation 10

Commits 10

Checks 33

Files changed 17

+302 -304



srockstyle commented on Jun 18 · edited

## 課題URL

[StudistCorporation/teachme\\_infrastructure#8524](https://StudistCorporation/teachme_infrastructure#8524)

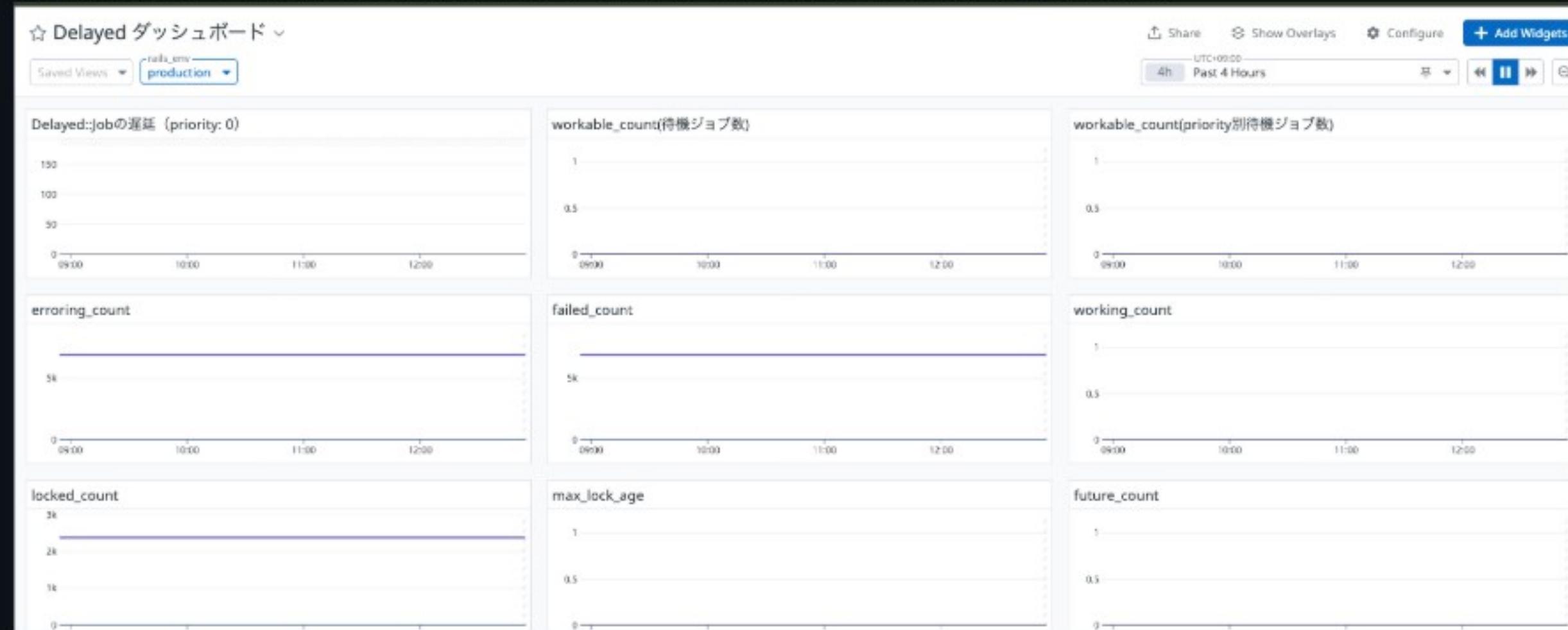
## WHAT

なんやかんやあってDelayed使わなくなったので削除する。

## WHY

### ここまでの大雑把なあらすじ

なんやかんやあってSolidQueueに移行完了し、Delayedで実行されているジョブが無くなった。



### Reviewers

masuzzk ✓

katsuhisa91 ✓

wind-up-bird ✓

fumi23 ✓

Kodalshikawa ✓

### Assignees

srockstyle

### Labels

gem rspec サーバーサイド

### Projects

None yet

### Milestone

No milestone

### Development

Successfully merging this pull request may close these issues.

None yet

### Notifications

Customize

Unsubscribe

# 【非同期バックエンド移行】Delayedとその関連基盤コードを削除 #17389

Edit `Code`

↳ Merged srockstyle merged 10 commits into master from develop/remove-delayed-job on Jun 23

Conversation 10

Commits 10

Checks 33

Files changed 17

+302 -304



srockstyle commented on Jun 18 · edited

## 課題URL

[StudistCorporation/teachme\\_infrastructure#8524](https://StudistCorporation/teachme_infrastructure#8524)

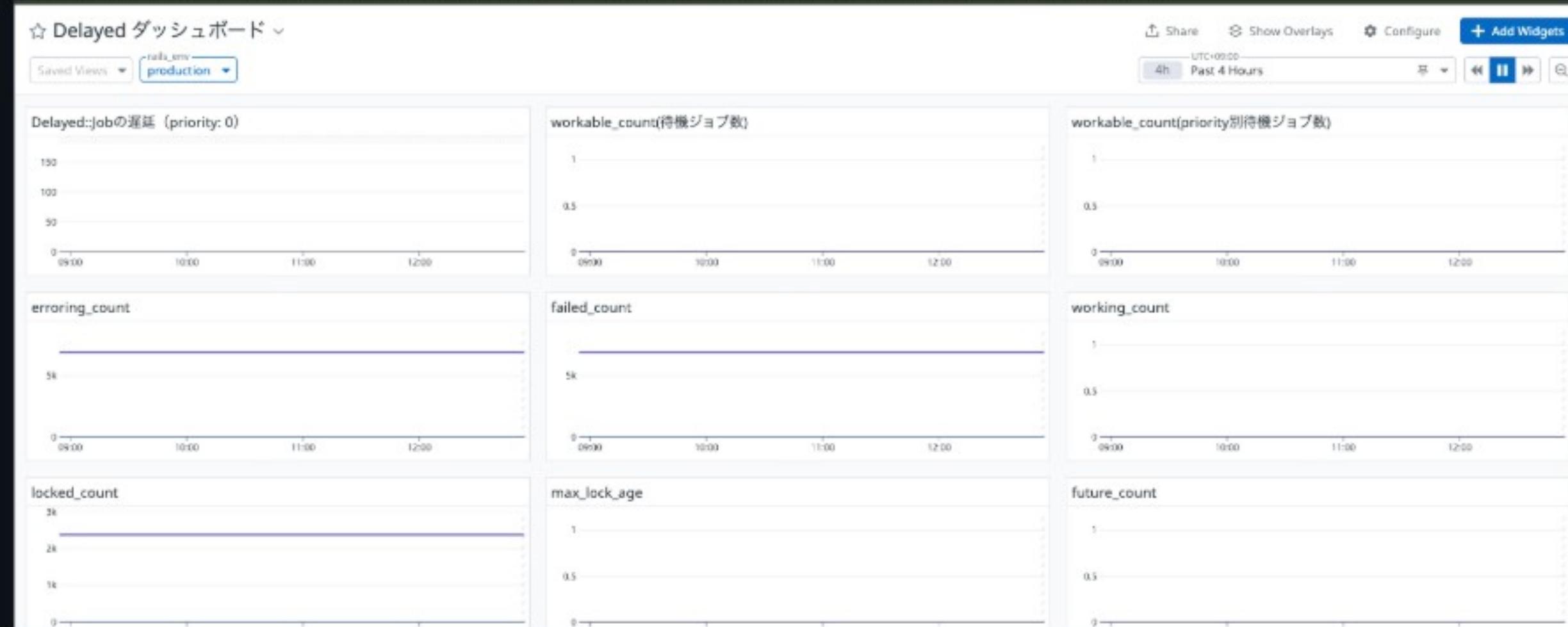
## WHAT

なんやかんやあってDelayed使わなくなったので削除する。

## WHY

### ここまでの大雑把なあらすじ

なんやかんやあってSolidQueueに移行完了し、Delayedで実行されているジョブが無くなった。



### Reviewers

masuzzk

katsuhisa91

fumi23

Kodalshikawa

### Assignees

srockstyle

### Labels

gem rspec サーバーサイド

### Projects

None yet

### Milestone

No milestone

### Development

Successfully merging this pull request may close these issues.

None yet

### Notifications

Customize

Unsubscribe

# 7 years ago...

Messages Add canvas Files Bookmarks +



**katsuhisa\_ / Katsuhisa Kitano / 北野 勝久** 10:29 AM

August 7th, 2018

こんな感じでERROR が発生していることはわかってるんだけど、  
それがプロセスを殺したかどうかまでは出てない感じ。  
おそらく、このERROR が出たタイミングでプロセスが死んでる。(と推測している。)



10:30 AM

またデッドロックかあ



**katsuhisa\_ / Katsuhisa Kitano / 北野 勝久** 10:31 AM

そうなんだよねえ

# Migration Requirements

- ✓ **.delay メソッドの撲滅**
- ✓ **ActiveJob で明示的に Delayed を指定**
- ✓ **SolidQueue 設定**
- ✓ **監視**
- ✓ **SolidQueue を利用するように全 Job を変更**
- ✓ **gem uninstall delayed**

# Migration Requirements

- ✓ .delay メソッドの撲滅
- ✓ ActiveJob で明示的に Delayed を指定
- ✓ SolidQueue 設定
- ✓ 監視
- ✓ SolidQueue を利用するように全 Job を変更
- ✓ gem uninstall delayed

完了だぜ！



# 結果！

## パフォーマンスとスケーラビリティの向上

- 綺麗に線形にスケールアウトできるようにしたい
- 処理できるジョブの数を増やしたい
- リソースを有効活用したい

## 柔軟なジョブ管理と優先制御

- 優先度を定義したら優先度を守ってほしい
- ジョブごとにキューと Worker を分けたい
- 定期実行処理をまとめたい (←これは今年やる予定！！)

## 開発と運用の効率化

- ActiveJob に移行したい (Delayed 専用の書き方と ActiveJob の混在廃止)
- できるだけ Rails Way に乗せたい

- ・はじめに： Teachme Biz について
- ・移行前の課題
- ・選定
- ・設計と実装
- ・移行後の課題
- ・まとめ

実装だけは多分楽

Claude Code 使えばもっと早かっ (略)

**AI で効率のよいエンジニアリングができる現代だからこそ**

AI で効率のよいエンジニアリングができる現代だからこそ

**Solid Queue の内部実装を理解すること**

**Solid Queue に合わせた設計 ~ リリース計画が超大事**

AI で効率のよいエンジニアリングができる現代だからこそ

**Solid Queue の内部実装を理解すること**

**Solid Queue に合わせた設計 ~ リリース計画が超大事**

**Delayed にあって Solid Queue にないものはあるか**

AI で効率のよいエンジニアリングができる現代だからこそ

**Solid Queue の内部実装を理解すること**

**Solid Queue に合わせた設計 ~ リリース計画が超大事**

**Delayed にあって Solid Queue にないものはあるか**

**Solid Queue に変更することで影響ある機能・チームはどこか**

AI で効率のよいエンジニアリングができる現代だからこそ

**Solid Queue の内部実装を理解すること**

**Solid Queue に合わせた設計 ~ リリース計画が超大事**

**Delayed にあって Solid Queue にないものはあるか**

**Solid Queue に変更することで影響ある機能・チームはどこか**

**現状の課題を解決できるものかどうか**

AI で効率のよいエンジニアリングができる現代だからこそ

**Solid Queue の内部実装を理解すること**

**Solid Queue に合わせた設計 ~ リリース計画が超大事**

**Delayed にあって Solid Queue にないものはあるか**

**Solid Queue に変更することで影響ある機能・チームはどこか**

**現状の課題を解決できるものかどうか**

**監視の計画する**

AI で効率のよいエンジニアリングができる現代だからこそ

**Solid Queue の内部実装を理解すること**

**Solid Queue に合わせた設計 ~ リリース計画が超大事**

**Delayed にあって Solid Queue にないものはあるか**

**Solid Queue に変更することで影響ある機能・チームはどこか**

**現状の課題を解決できるものかどうか**

**監視の計画する**

**障害を起こさないように段階的なリリース計画をする**

# 最後に弊社 EM より

「全ての技術的負債と呼ばれるものは  
その時の全力であることは間違いないし、  
そう信じて前進していきたい」

# One More Thing

仕事探しているそこの君!

# 仕事探しているそこの君!

## SRE (Site Reliability Engineer)

応募する

### 採用情報

#### 職務内容

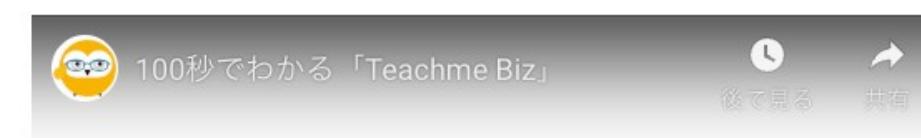
手順書作成・共有のB2B SaaS 「[Teachme Biz](#)」のSRE

具体的には

- プロダクト開発者自身でのシステム運用を行うためのプラットフォームおよび周辺ツールの構築等
- SREの思想に基づいたチームプラクティスの導入・運用・改善 (e.g. Postmortem, SLI/SLO, Production Readiness Checklist ...)
- AWS, Terraform 等を活用した Infra基盤の設計・構築およびレビュー
- ログ収集/解析基盤の開発・運用
- モニタリング/アラートシステムの構築・運用
- プロダクト開発者自身でのシステム運用をサポートする周辺ツールの開発・運用

をお任せします。

Teachme Biz についてはこちらをご覧ください。



見る 

#### 募集背景

現在、継続的な機能開発スピードの向上を目指すために、アーキテクチャ刷新を行っています。開発者体験を継続的に向上していくため、プラットフォームの構築・運用に興味があるエンジニアの方を募集しています。

<https://open.talentio.com/r/1/c/studist-recruit/pages/29489>

# スタディストのSRE は君を待ってるぜ!

私たちの旅路は  
続くんだぜ！



私たちの旅路は  
続くんだぜ！

ご清聴ありがとうございました！

