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

KONNICHIWA, Ìm 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っているので、

しょーへーさん or **すろっくさん**って呼んでください

KONNICHIWA, Ìm Shohei Kobayashi@srockstyle

Site Reliability Engineer

in Platform Engineering Unit at Studist

KONNICHIWA, Ìm Shohei Kobayashi@srockstyle

Site Reliability Engineer

in Platform Engineering Unit at Studist

Engineer history 2005 ～

KONNICHIWA, Ìm Shohei Kobayashi@srockstyle

Site Reliability Engineer

in Platform Engineering Unit at Studist

Engineer history 2005 ～

Rails3 ～

KONNICHIWA, Ìm Shohei Kobayashi@srockstyle

Site Reliability Engineer

in Platform Engineering Unit at Studist

Engineer history 2005 ～

Rails3 ～

Vine Linux 2.0 ～

Fedora Core 1 ～

KONNICHIWA, Ìm 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てます

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

スタディスト公式note
2025年8月7日 10:39

「合意形成」

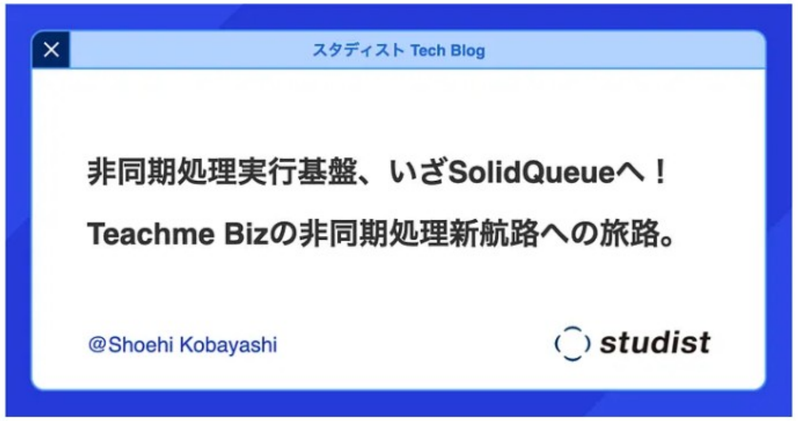
言葉にするのは簡単ですが、日々の業務においてこれほどまでに難しく、ま

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

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

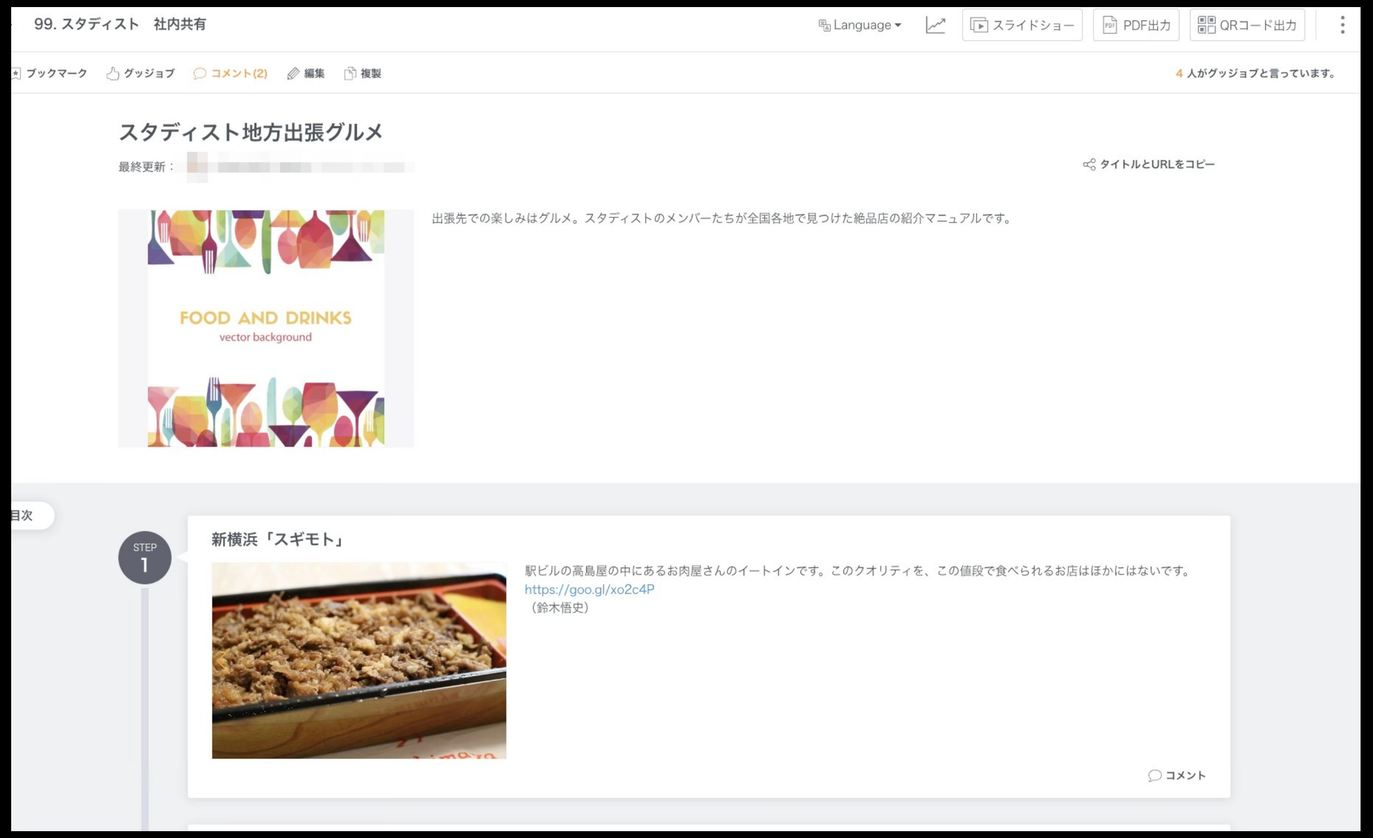
アジェンダ

- 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

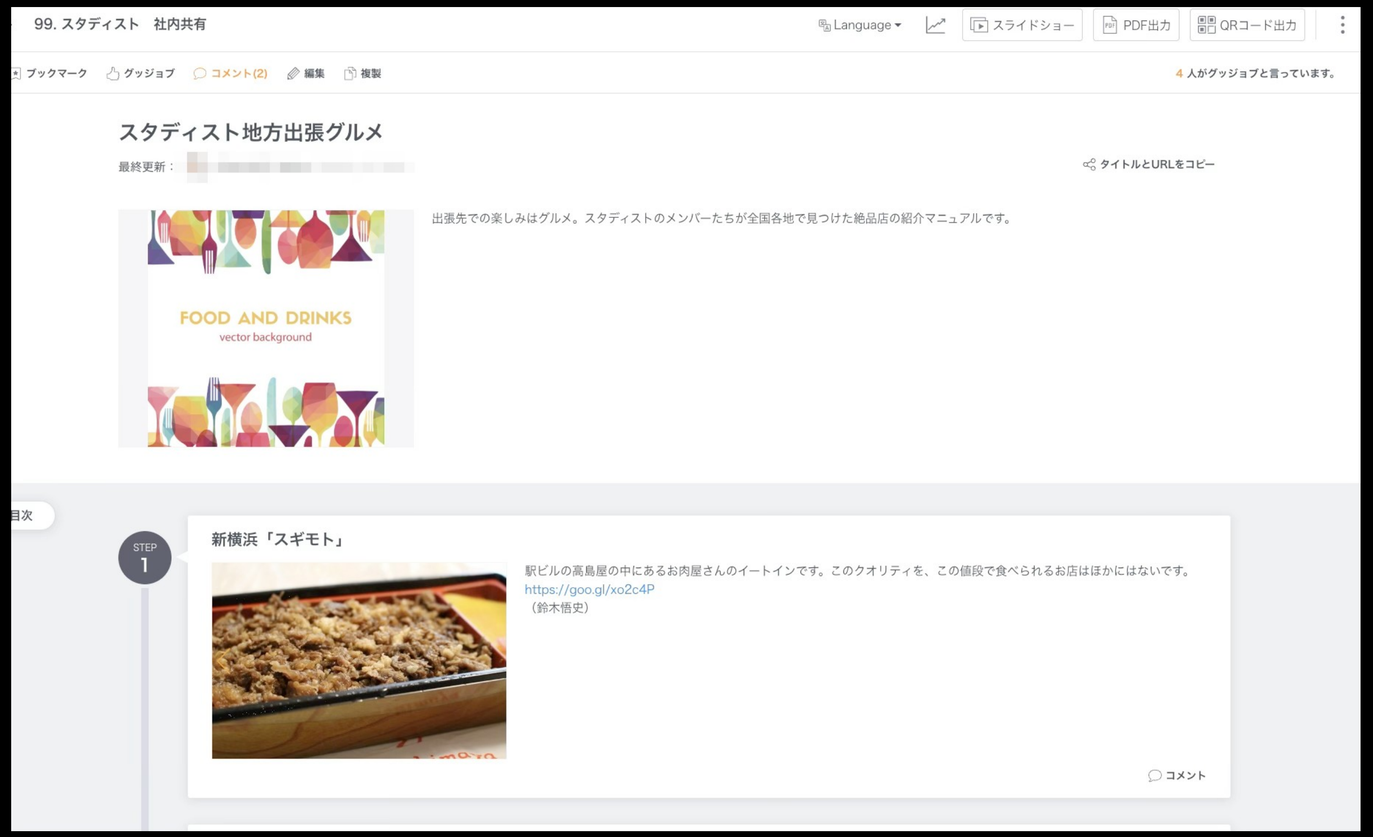
- はじめに： Teachme Biz について
- 課題 Part その 1
- 選定
- 設計と実装そして移行
- 課題 Part その 2
- まとめ


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

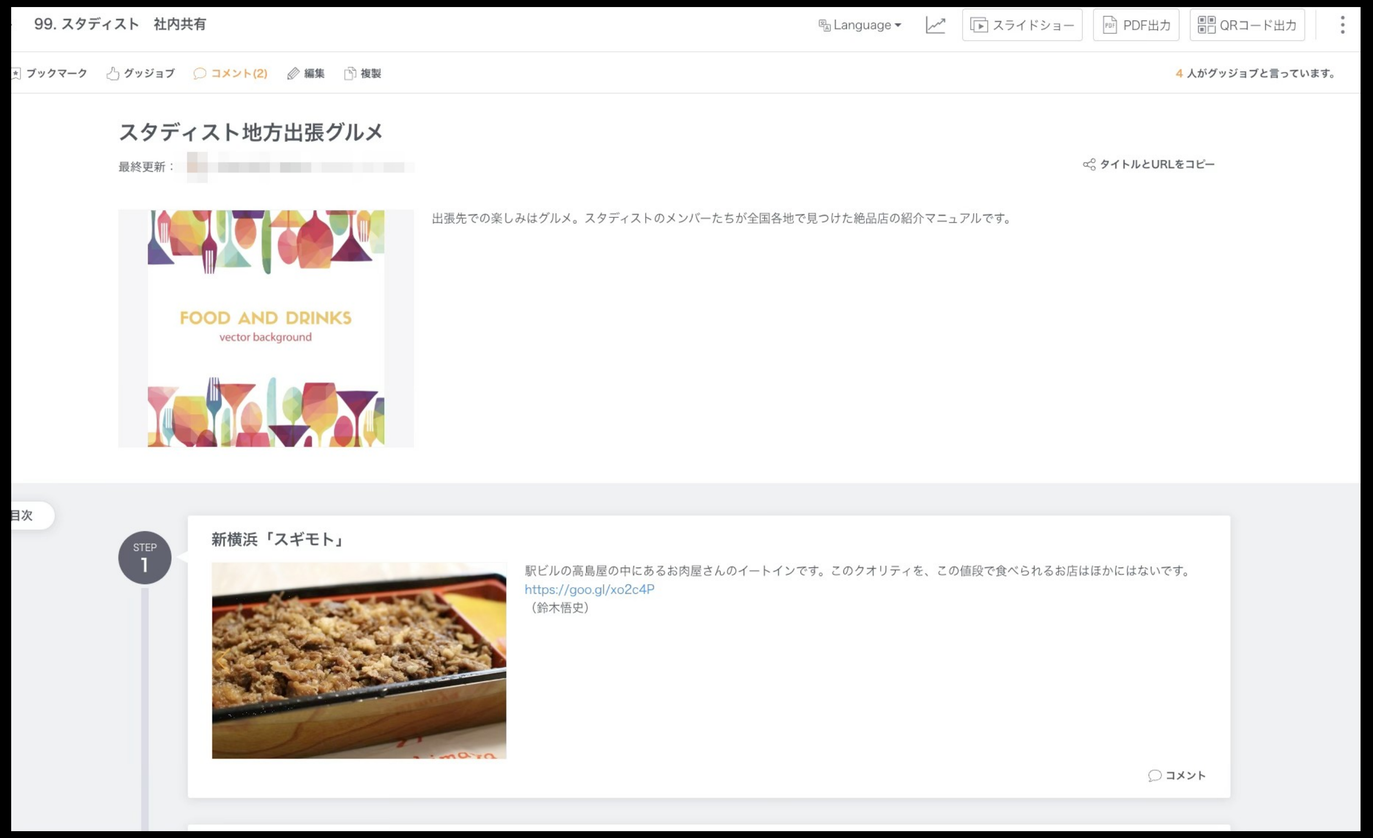


リリース 10 年以上の Rails アプリケーション



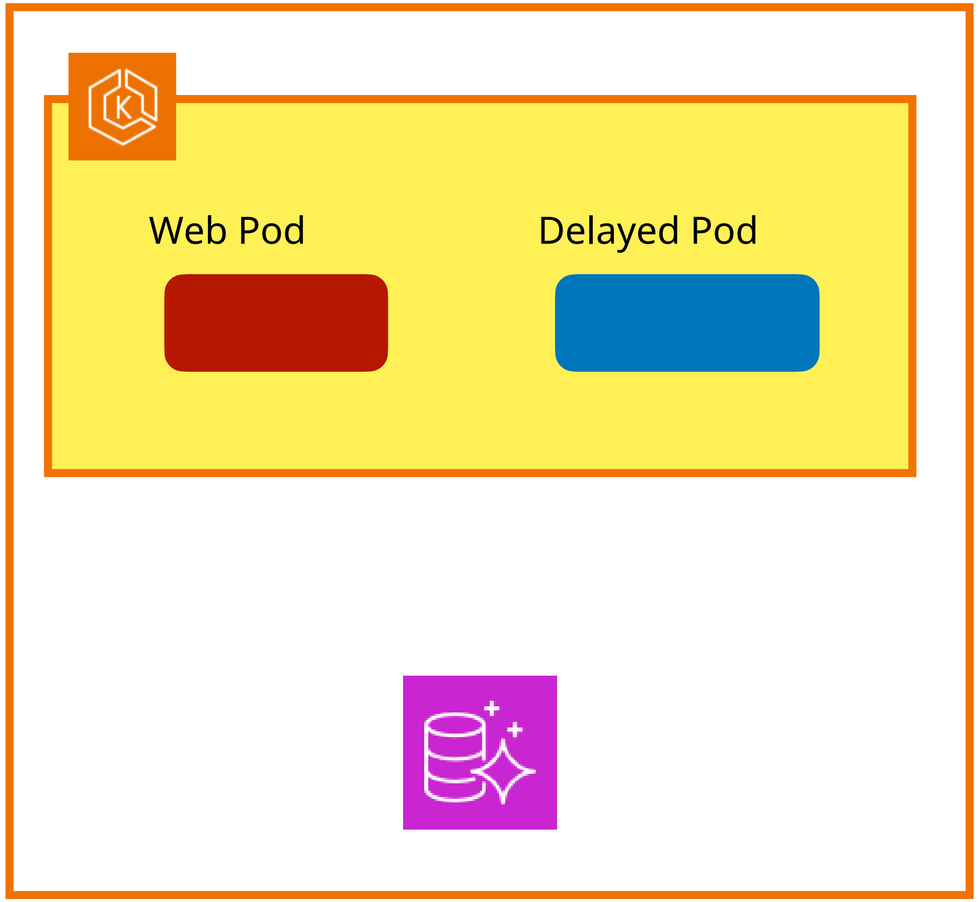
2012 年リリース Rails3.2.1

リリース 10 年以上の Rails アプリケーション

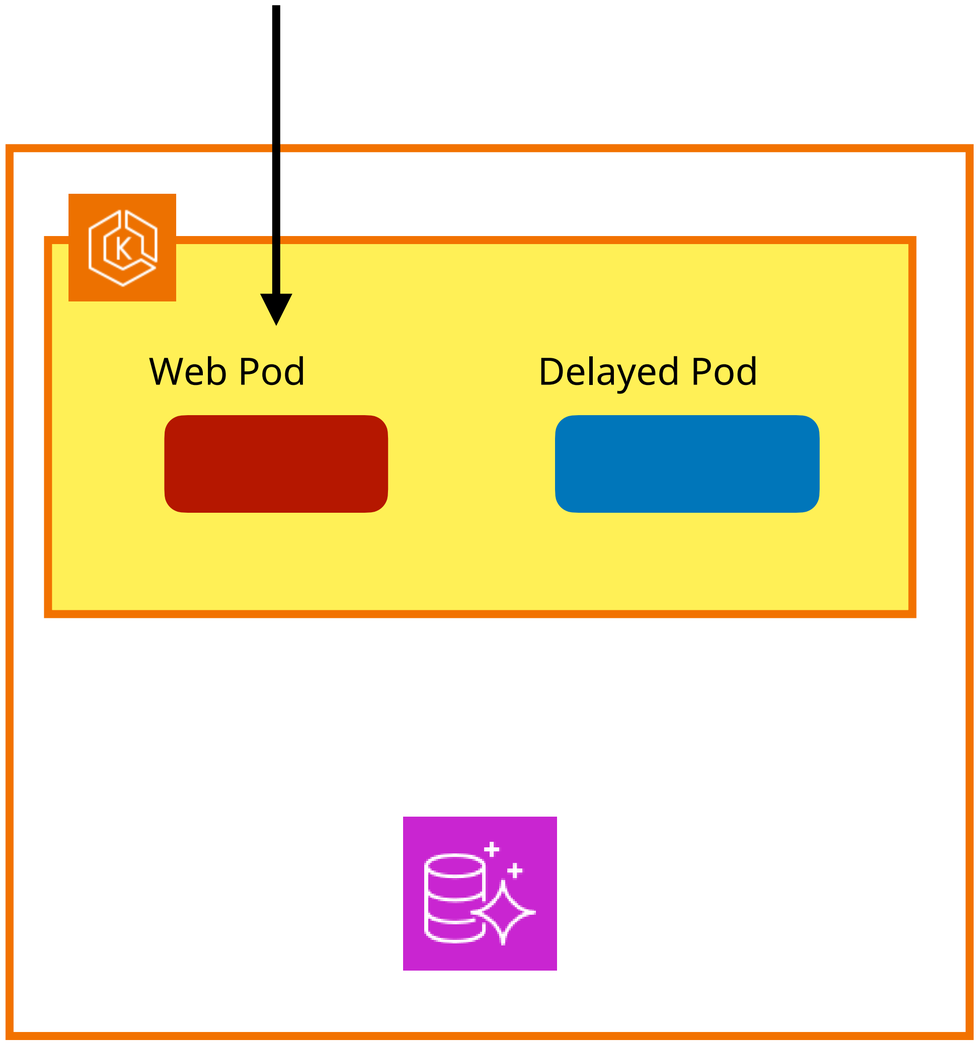


2012 年リリース Rails3.2.1 -> 2024 年 10 月当時 Rails7.1

非同期処理の簡単なおさらい

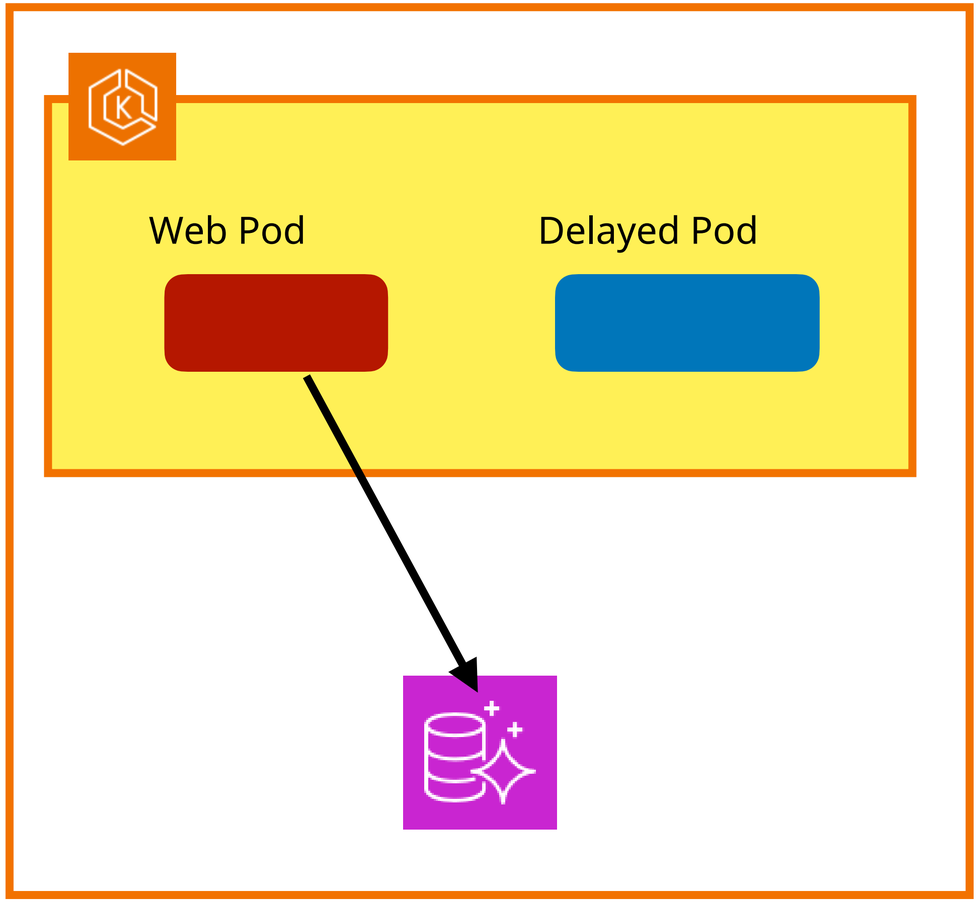


非同期処理の簡単なおさらい



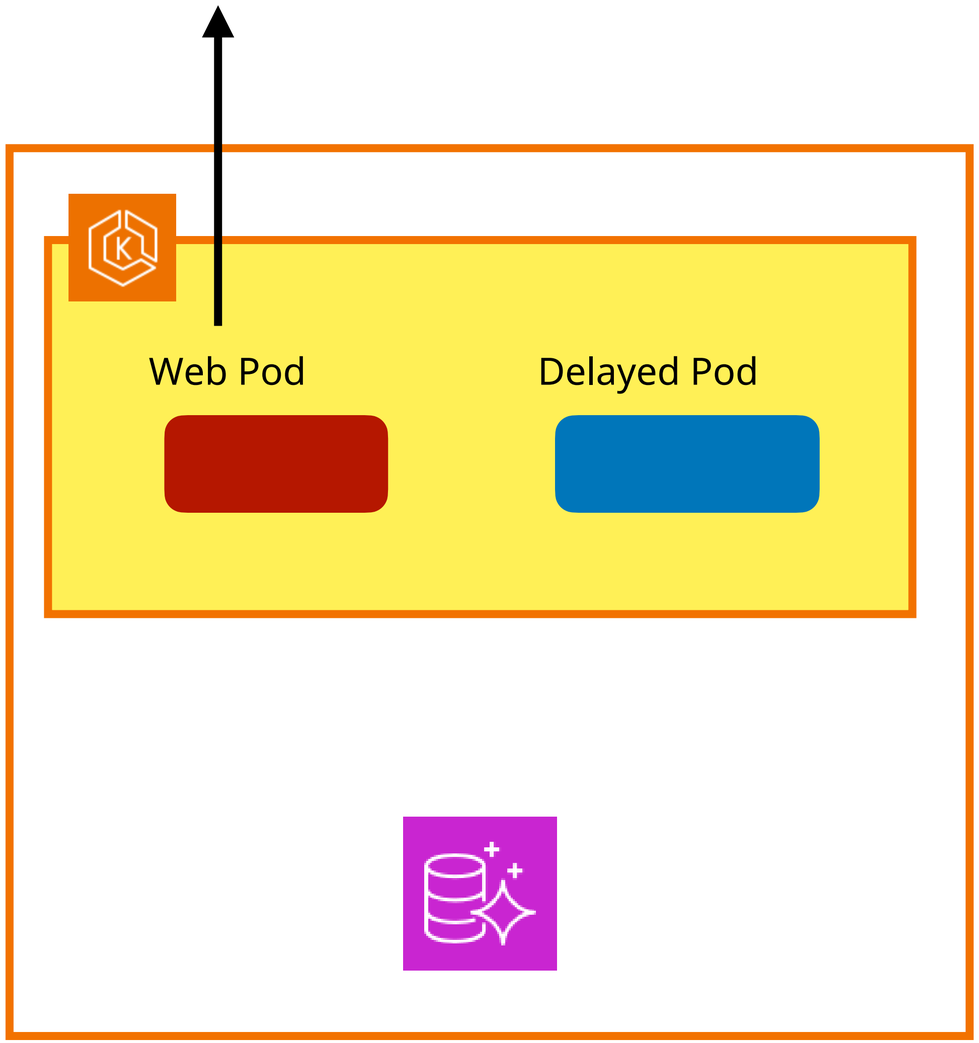
1. ユーザが操作する

非同期処理の簡単なおさらい



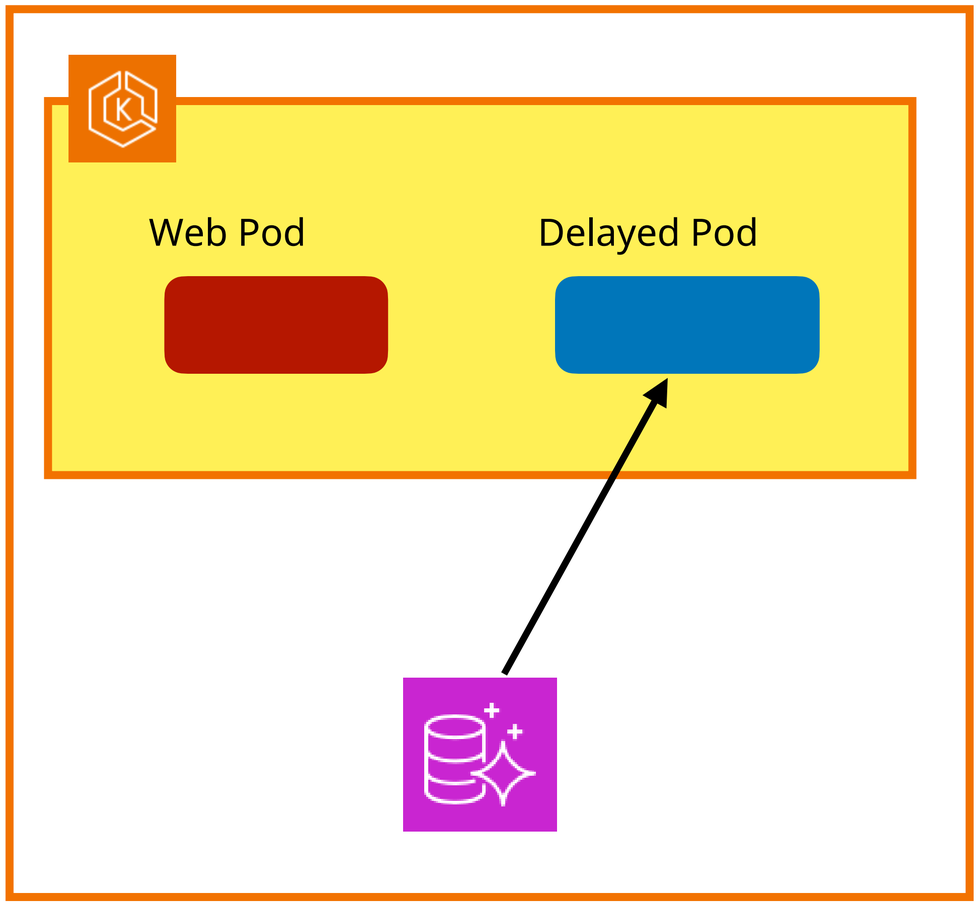
1. ユーザが操作する
2. DB に処理がエンキューされる

非同期処理の簡単なおさらい



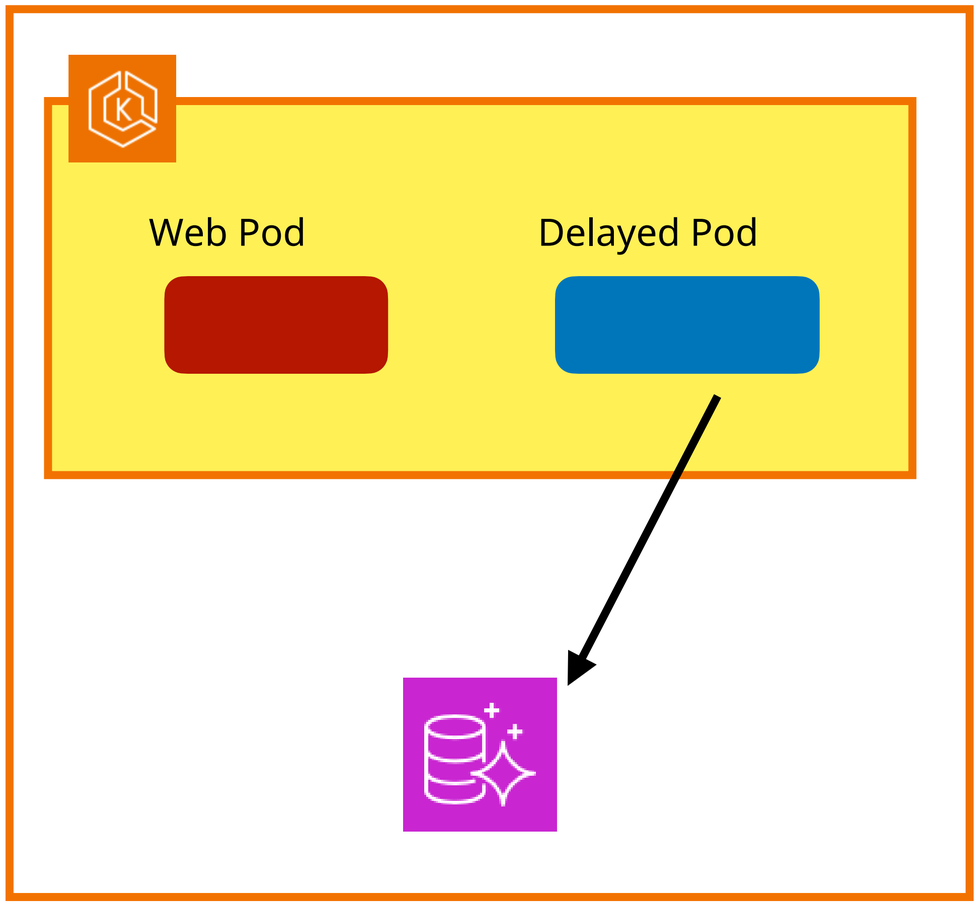
1. ユーザが操作する
2. DB に処理がエンキューされる
3. ユーザにレスポンスが返される

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

非同期処理基盤

- 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 | +

August 7th, 2018

katsuhisa_ / Katsuhisa Kitano / 北野 勝久 10:29 AM

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

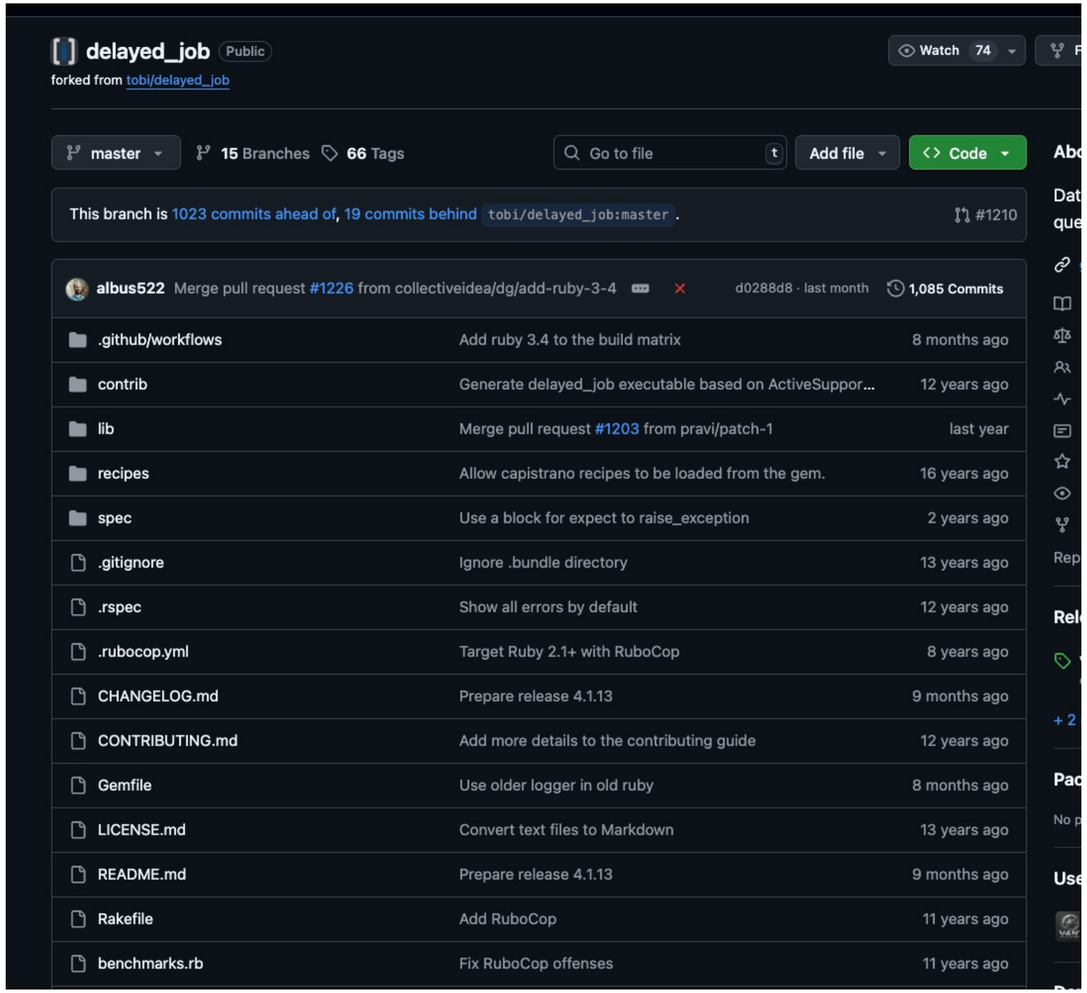
またデッドロックかぁ

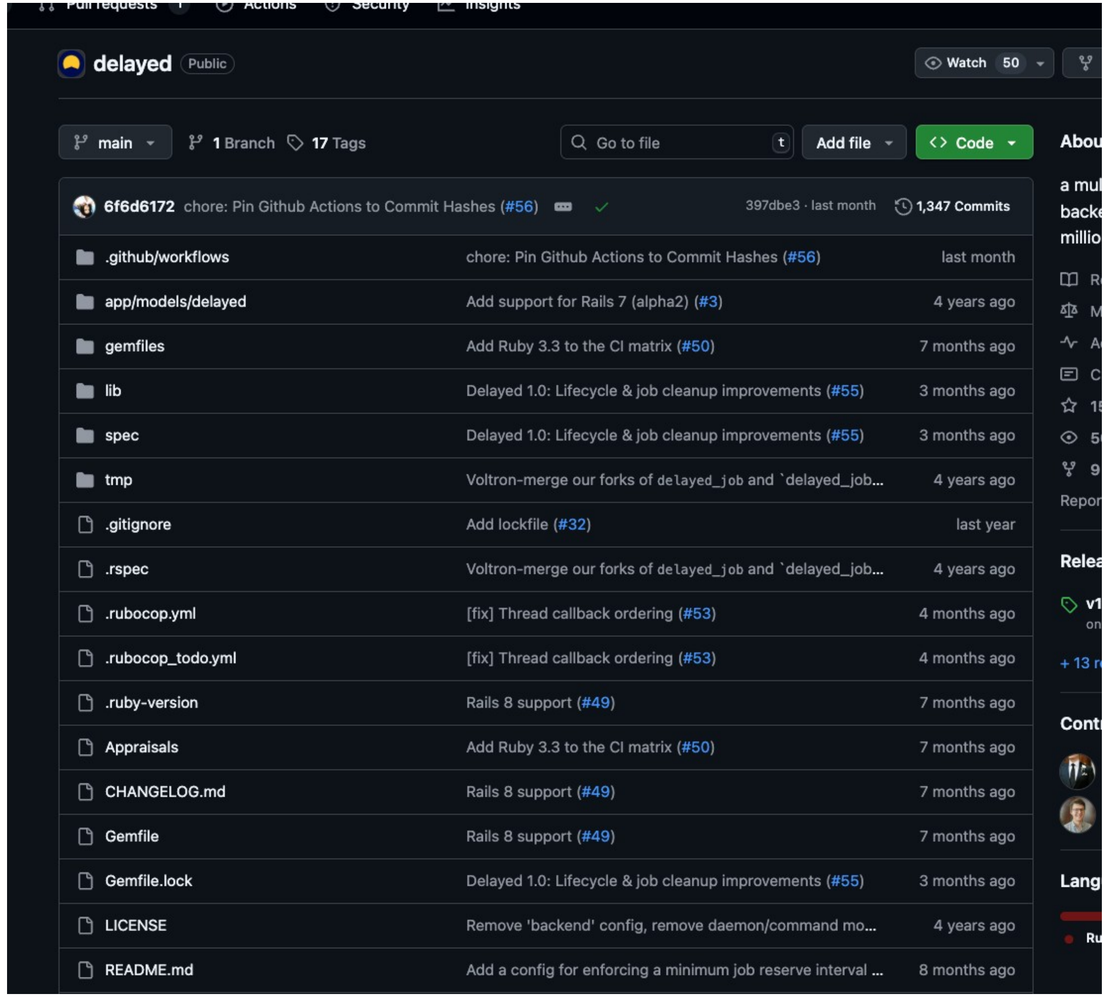
katsuhisa_ / Katsuhisa Kitano / 北野 勝久 10:31 AM

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





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、なる早で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、なる早で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 人にしたよ！

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 人にしたよ！

2 人にしたからお客さん（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人にしたのに…

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 人にしたのに…

電卓は一個しかねえぞ！！！

2 人に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 人にしたのに…

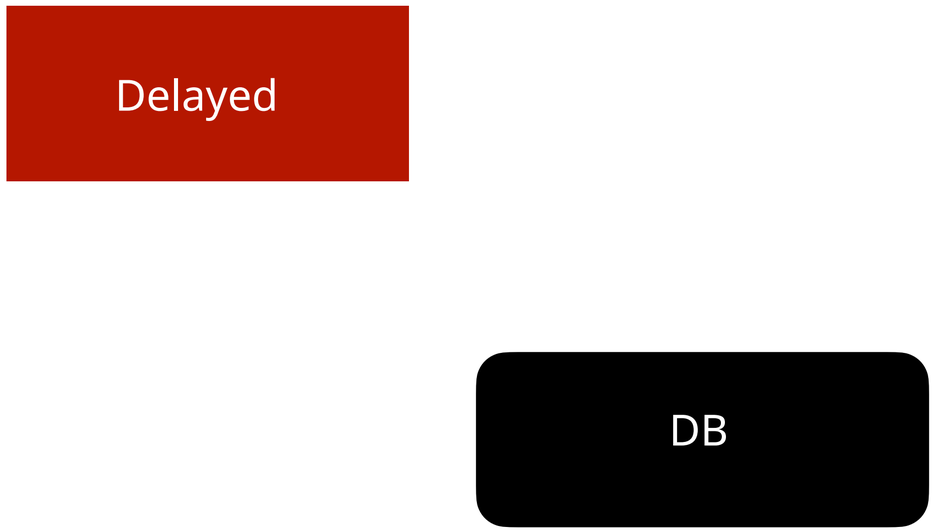
電卓は一個しかねえぞ！！！

2 人に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

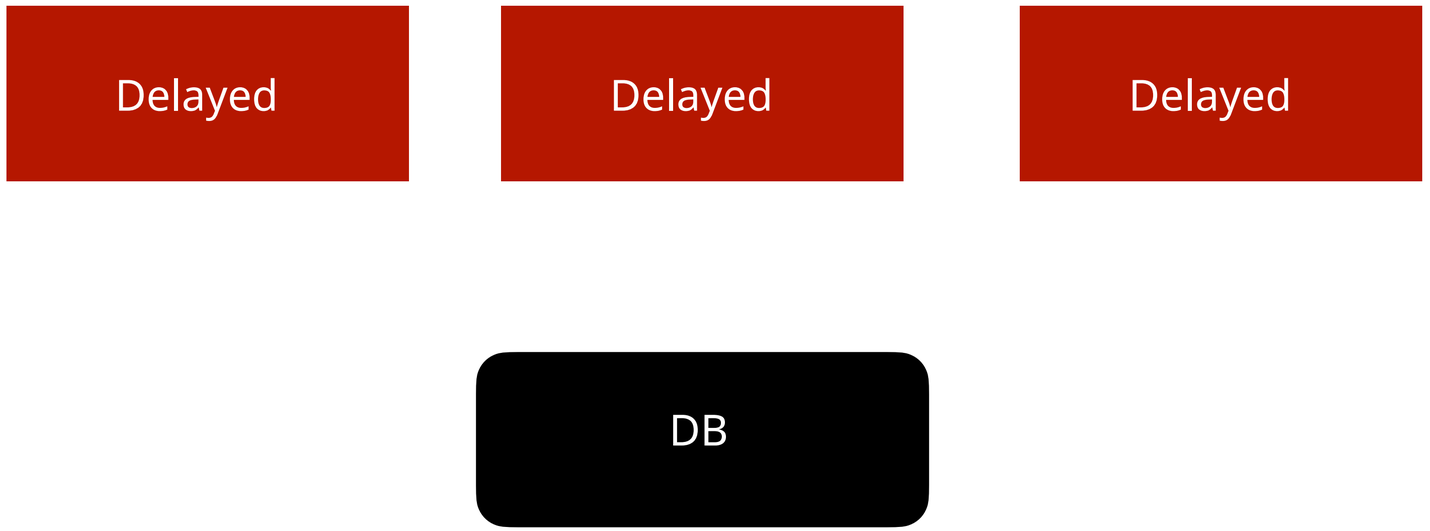
ジョブがいっぱいあるのでいっぱい処理したい



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

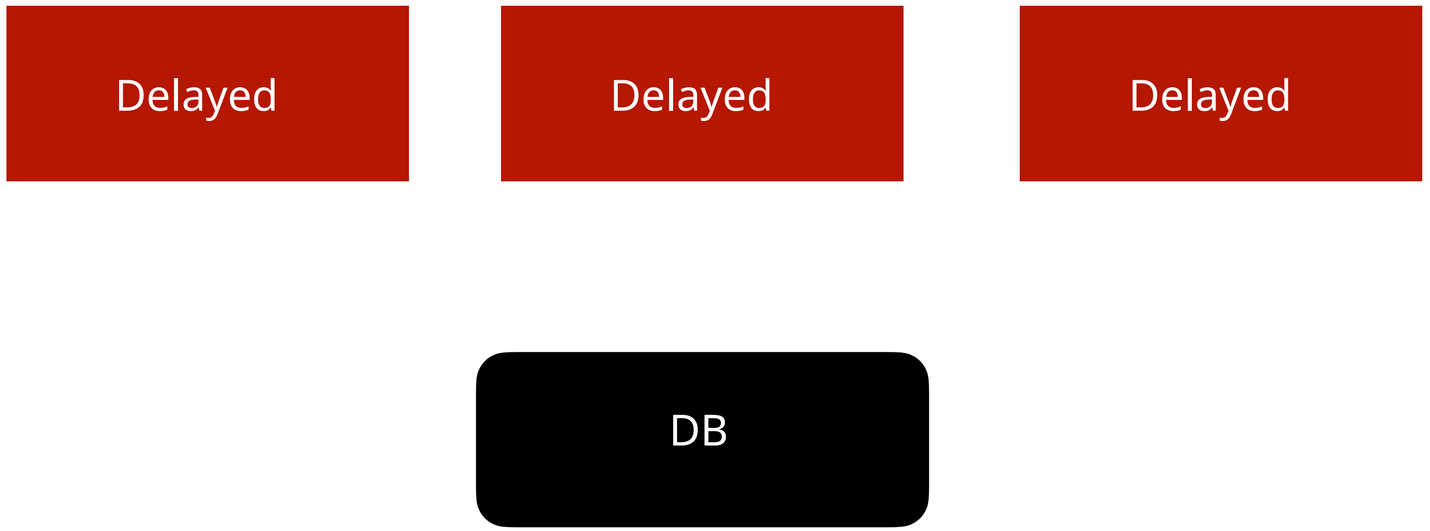
処理 Worker を増やす = 処理がもっと捌けるはず



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!

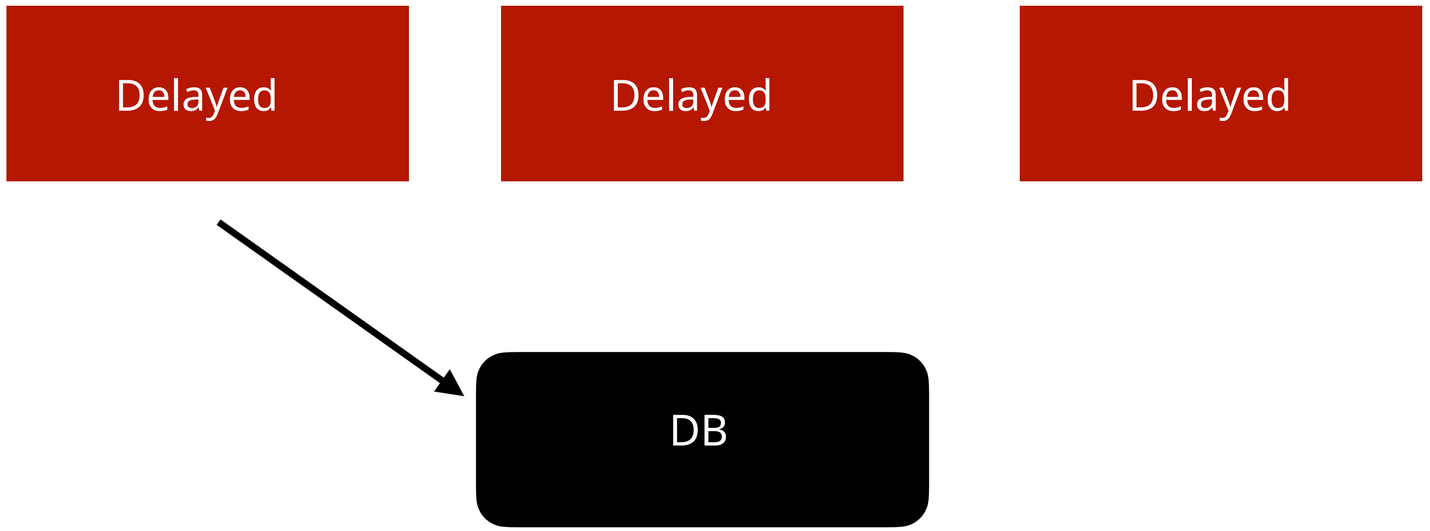


線形的なスケールアウト不可能問題

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

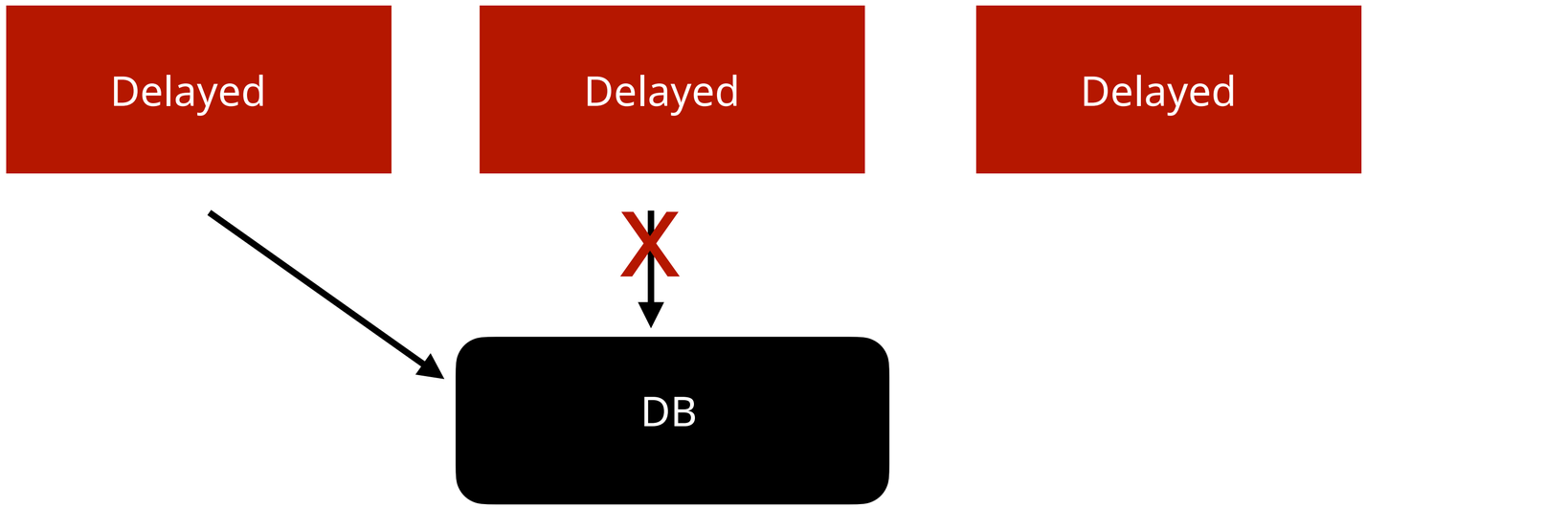


線形的なスケールアウト不可能問題

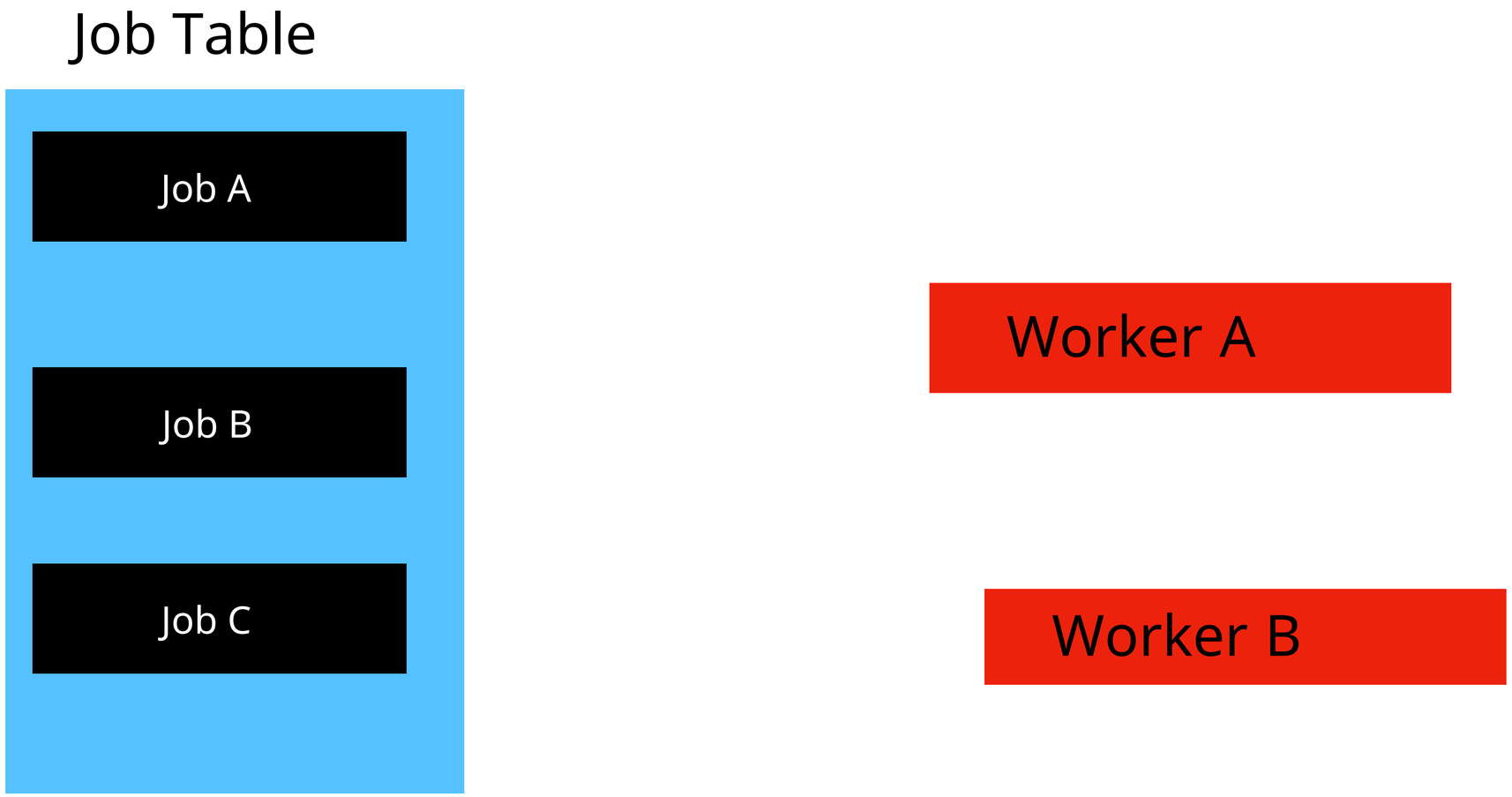
ジョブがいっぱいあるのでいっぱい処理したい

処理実行!

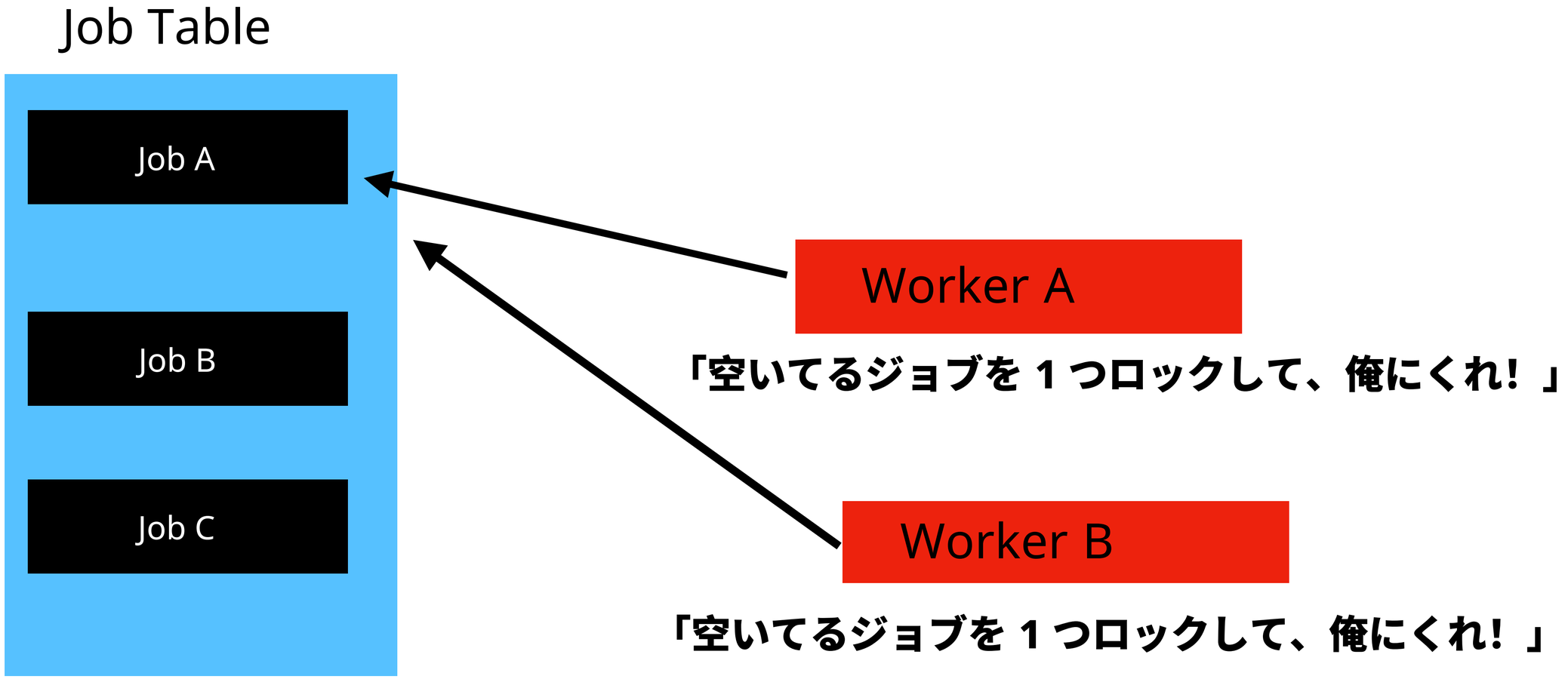
他の worker は待つ。



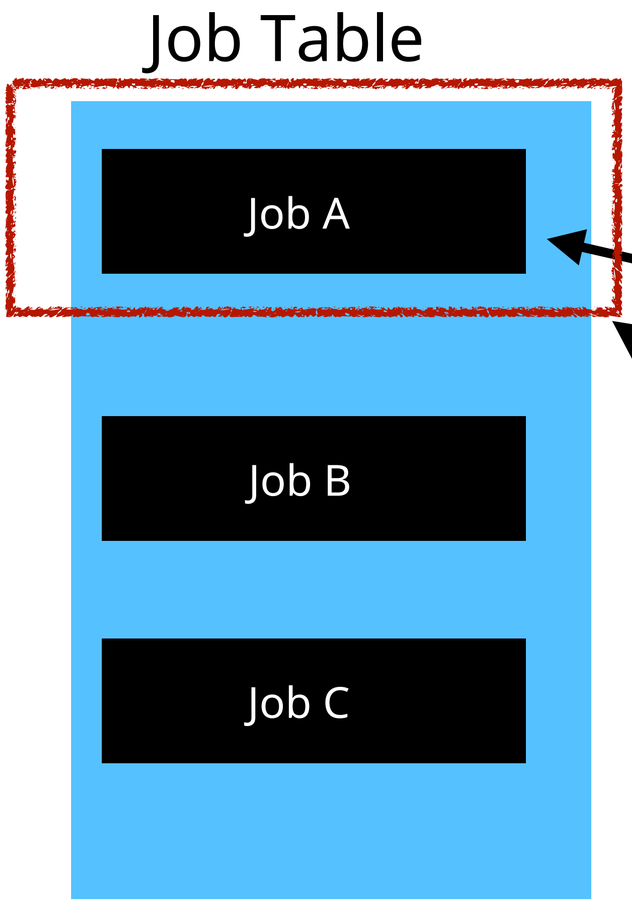
線形的なスケールアウト不可能問題



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




```
START TRANSACTION;
SELECT * FROM JOBS WHERE processed='no' FOR UPDATE LIMIT 1;
-- xxxxx
COMMIT
```

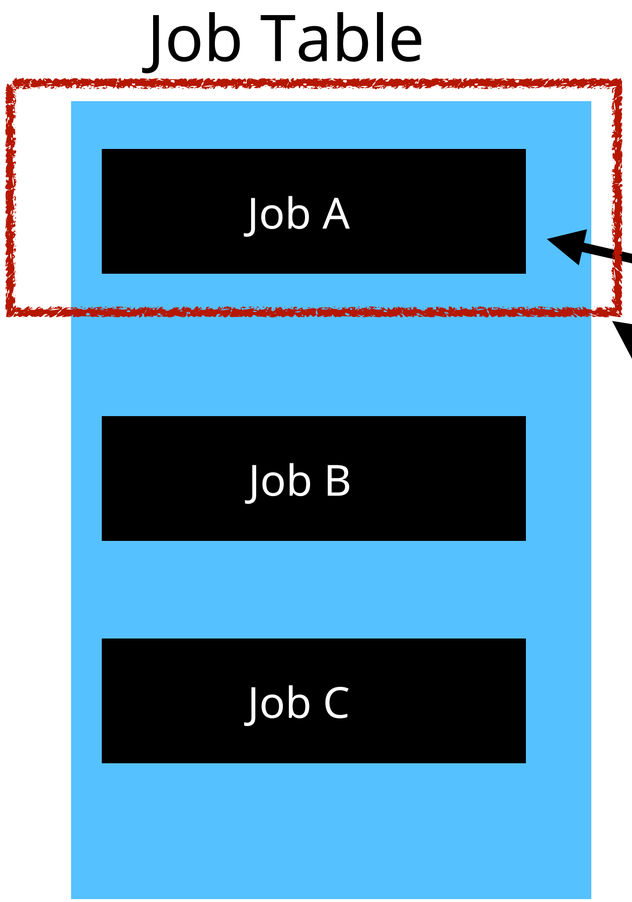
Worker A

「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！」

線形的なスケールアウト不可能問題




```
START TRANSACTION;
SELECT * FROM JOBS WHERE processed='no' FOR UPDATE LIMIT 1;
-- xxxxx
COMMIT
```

Worker A

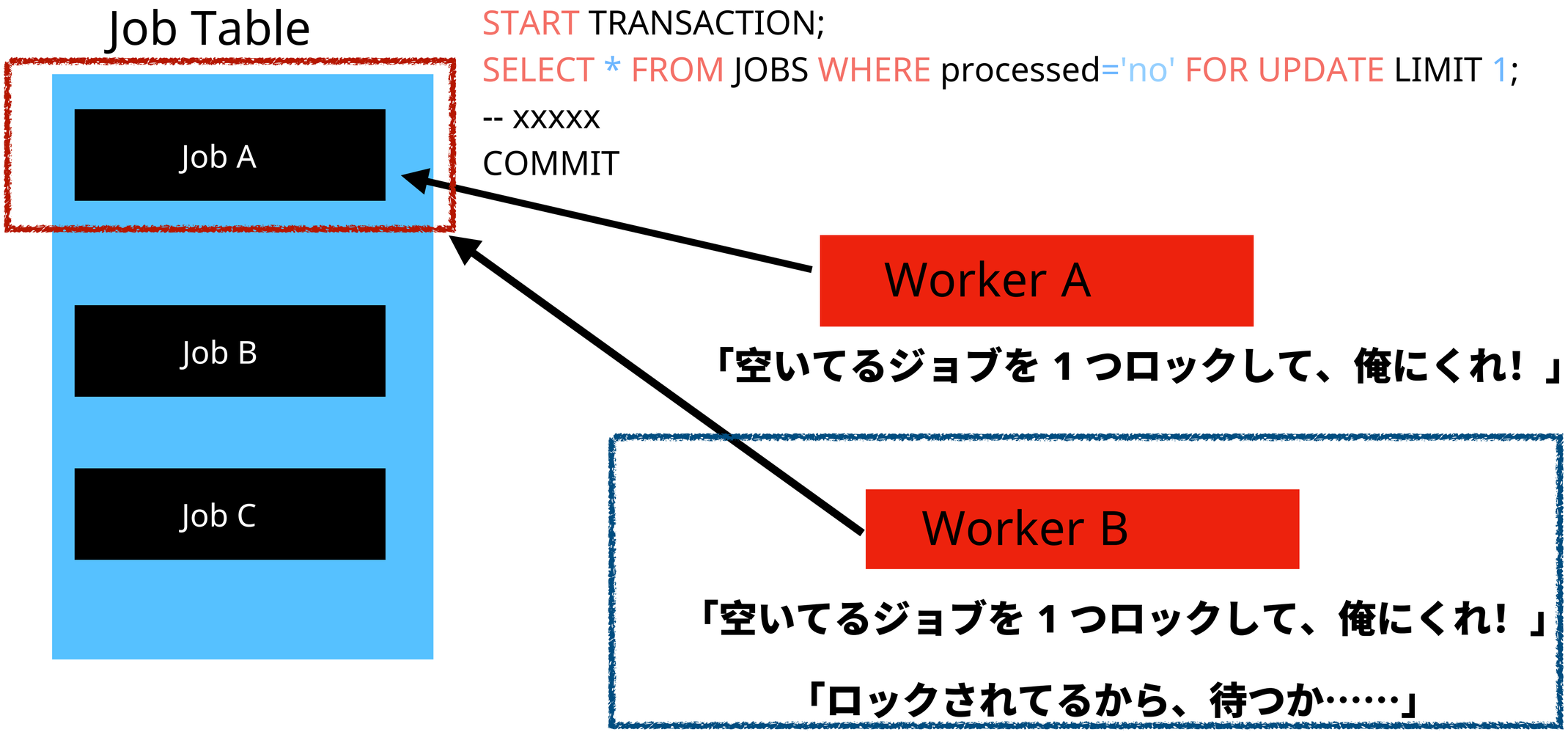
「空いてるジョブを 1 つロックして、俺にくれ！」

Worker B

「空いてるジョブを 1 つロックして、俺にくれ！」

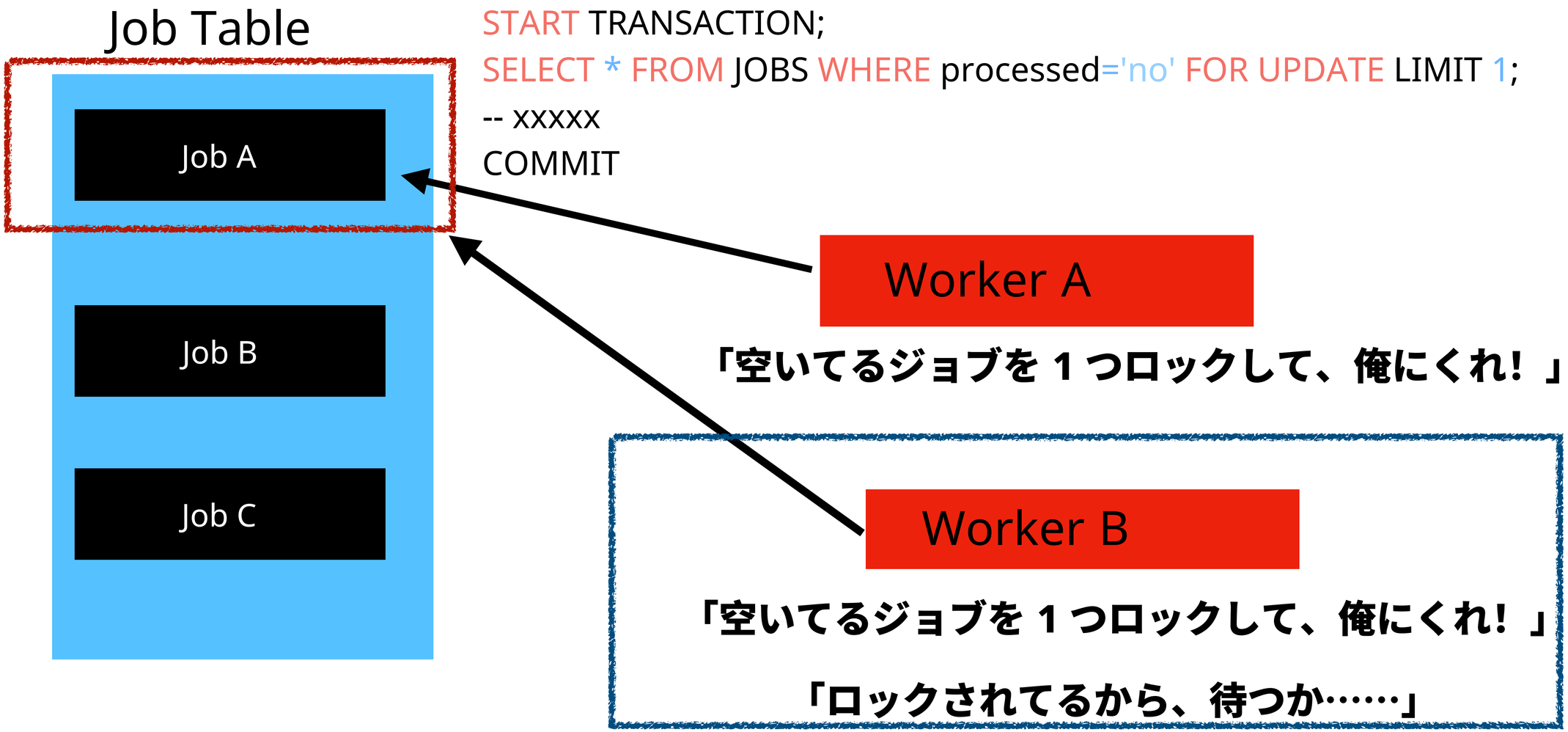
「ロックされてるから、待つか……」

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

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

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

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こと
- ⬜ 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こと
- ⬜ 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こと
- ⬜ 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こと
- ⬜ 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こと
- ⬜ 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

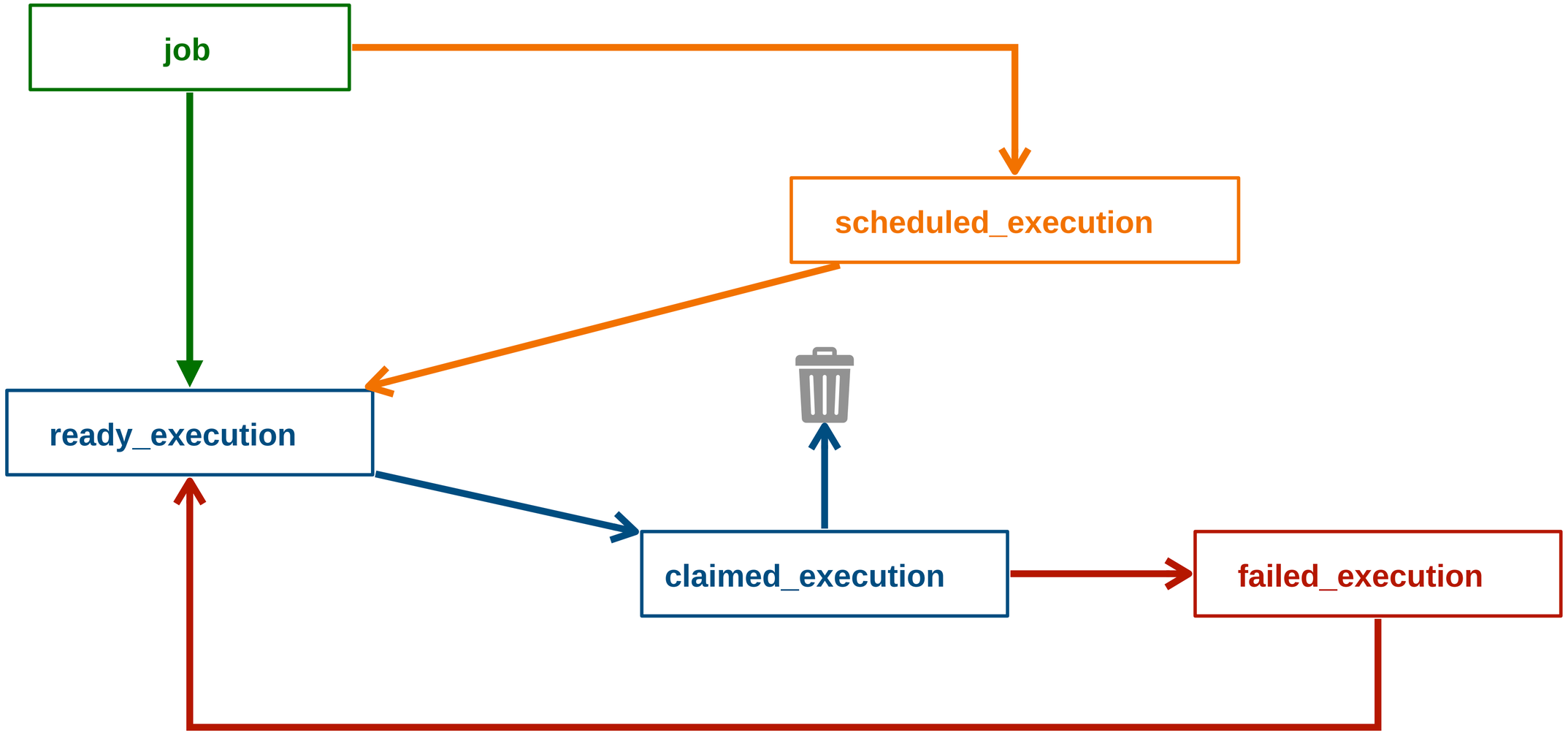
障害対応とトラブルシューティング

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

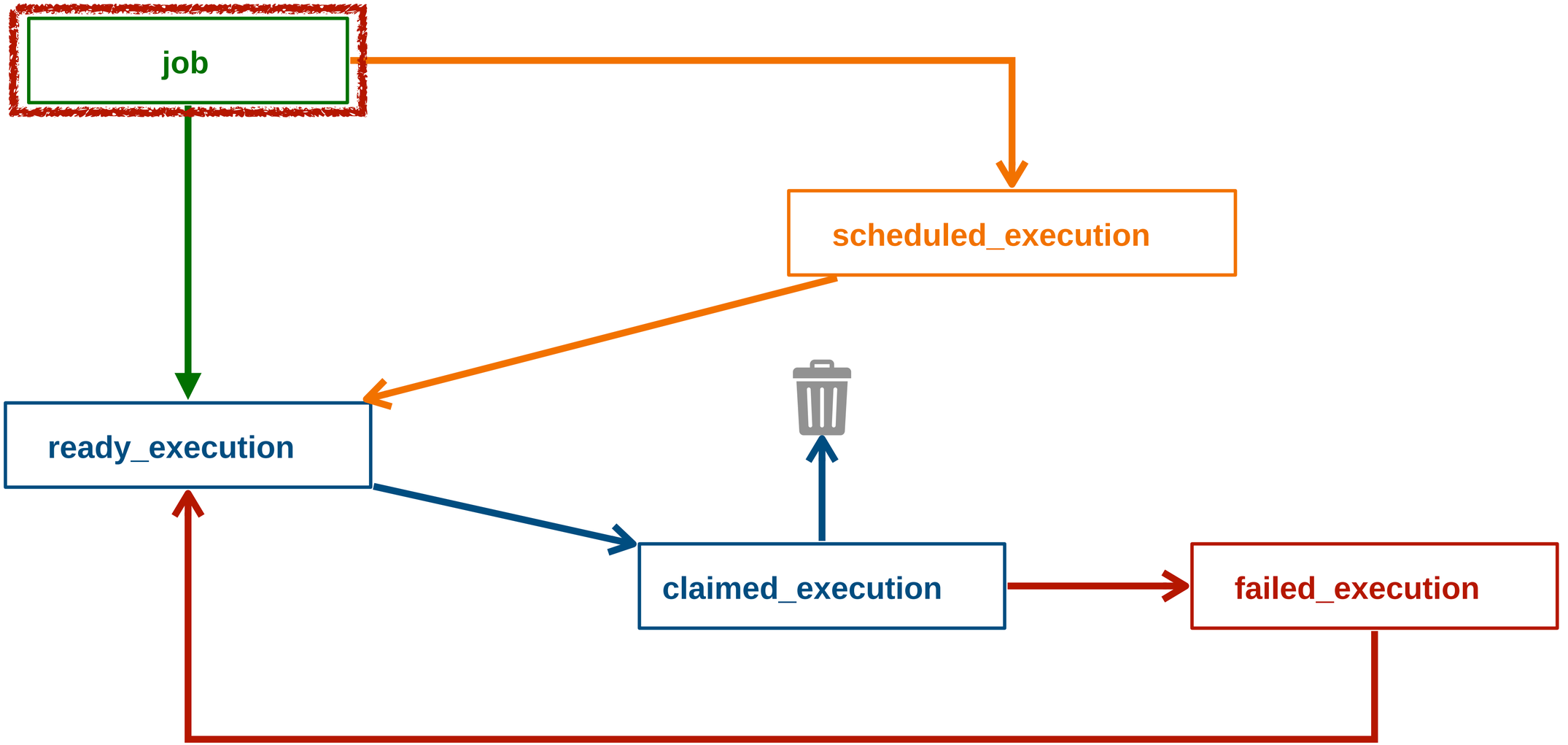
全体図（間に色々あるけど主要なものだけ）



Solid Queue の内部実装の理解

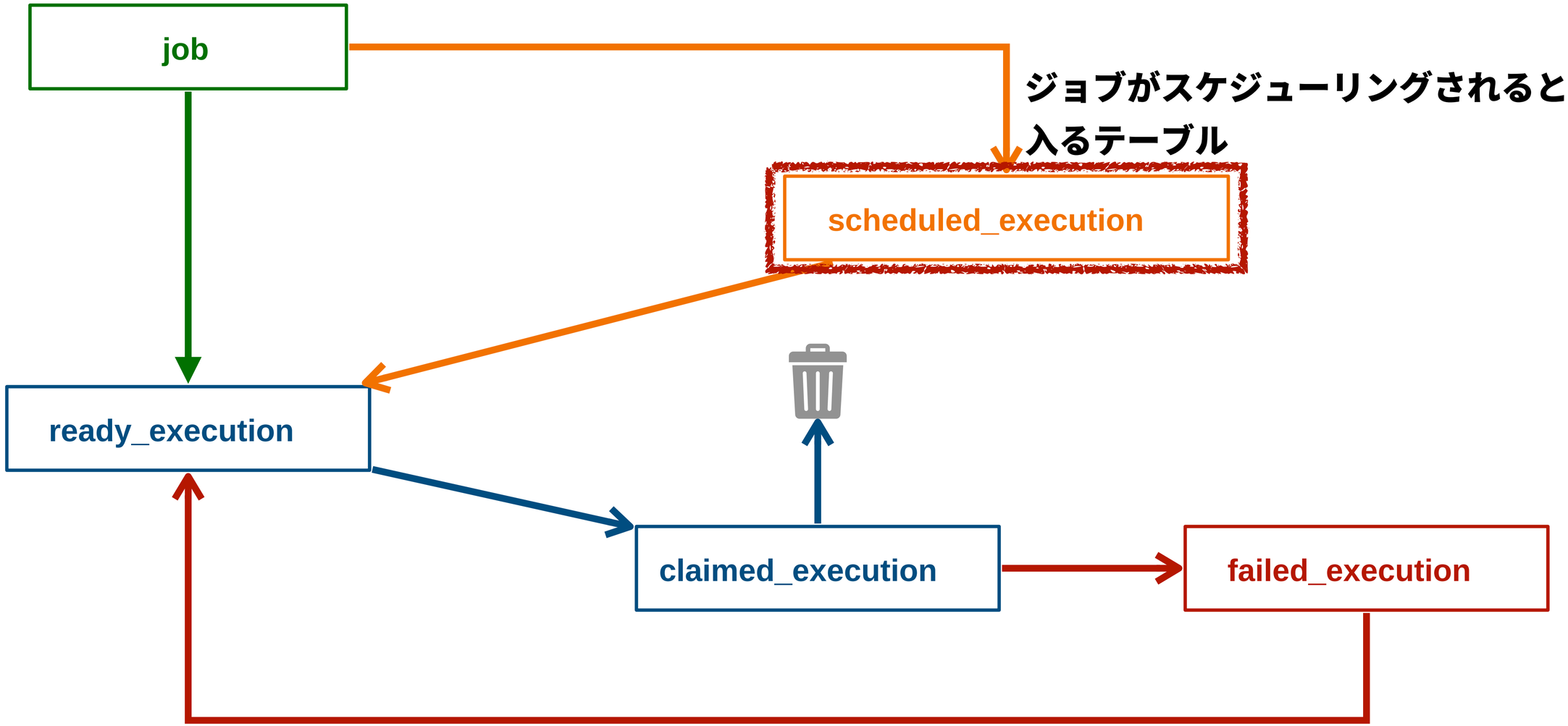
全体図（間に色々あるけど主要なものだけ）

ジョブがエンキューされると入るテーブル



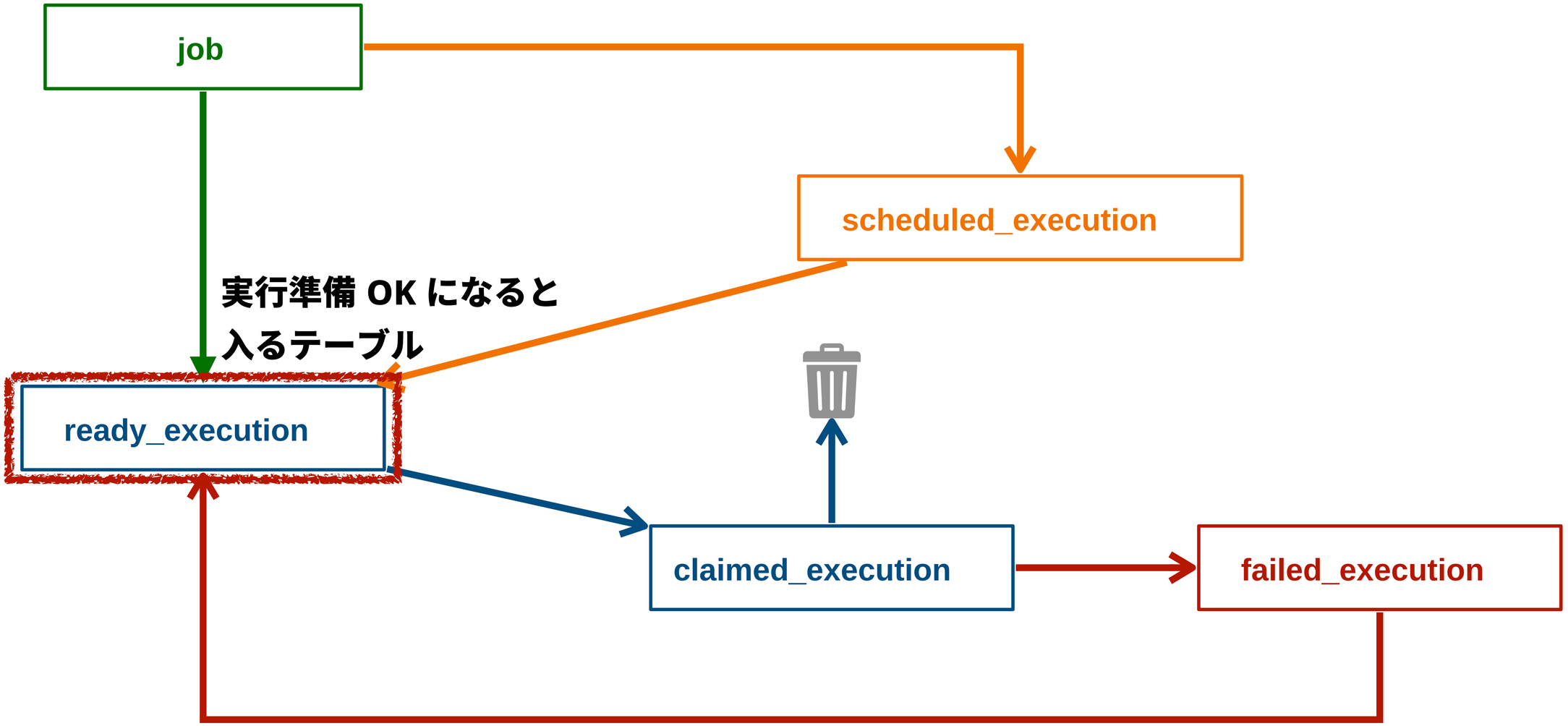
Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なものだけ）



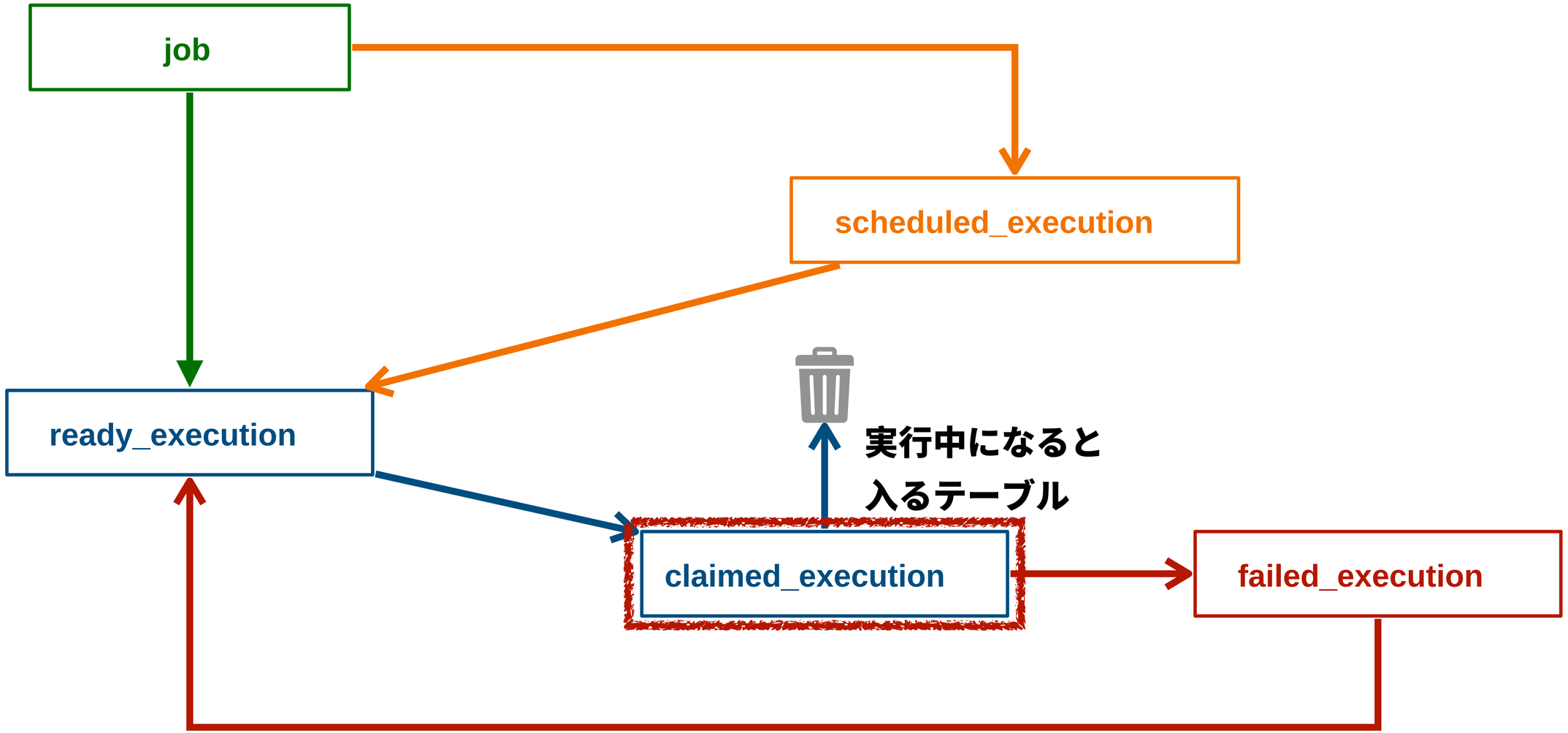
Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なものだけ）



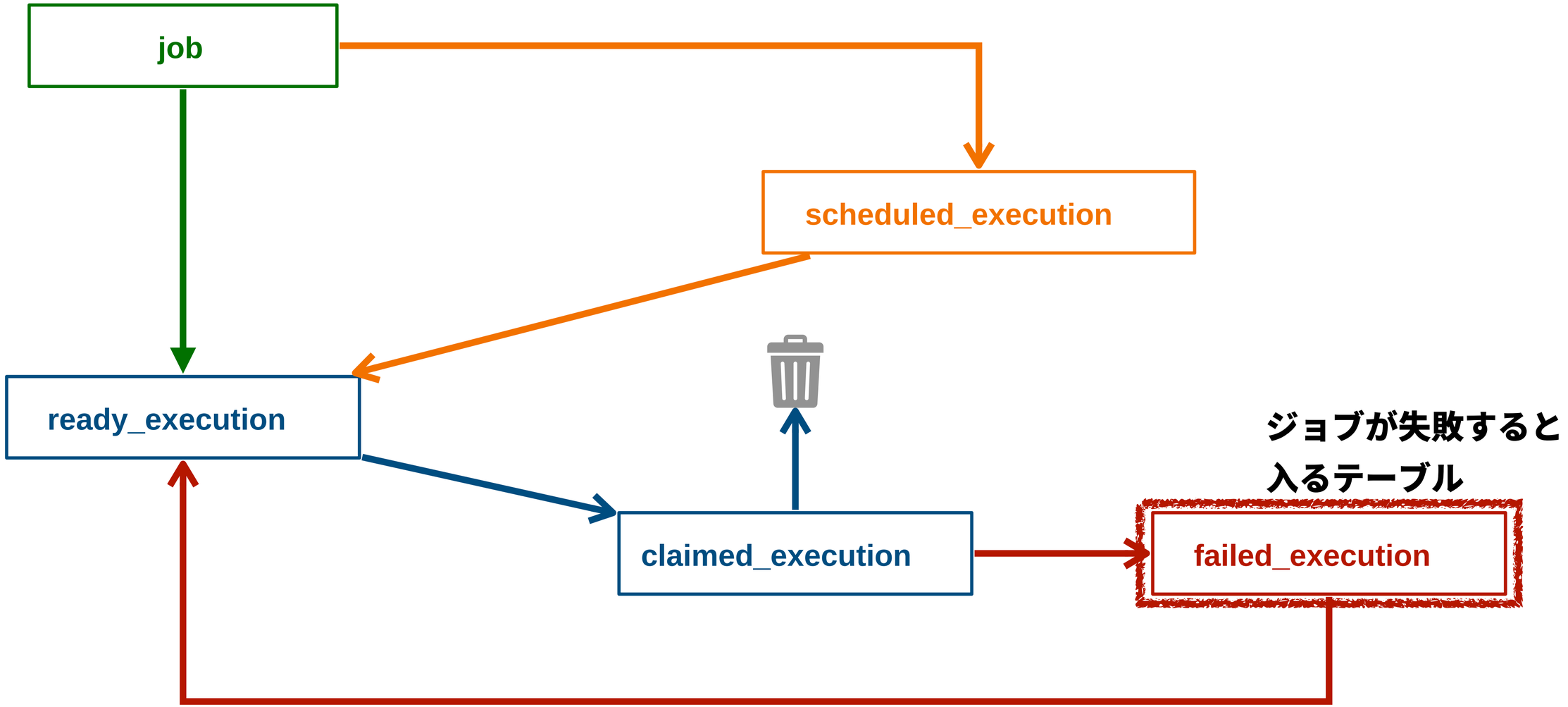
Solid Queue の内部実装の理解

全体図（間に色々あるけど主要なものだけ）



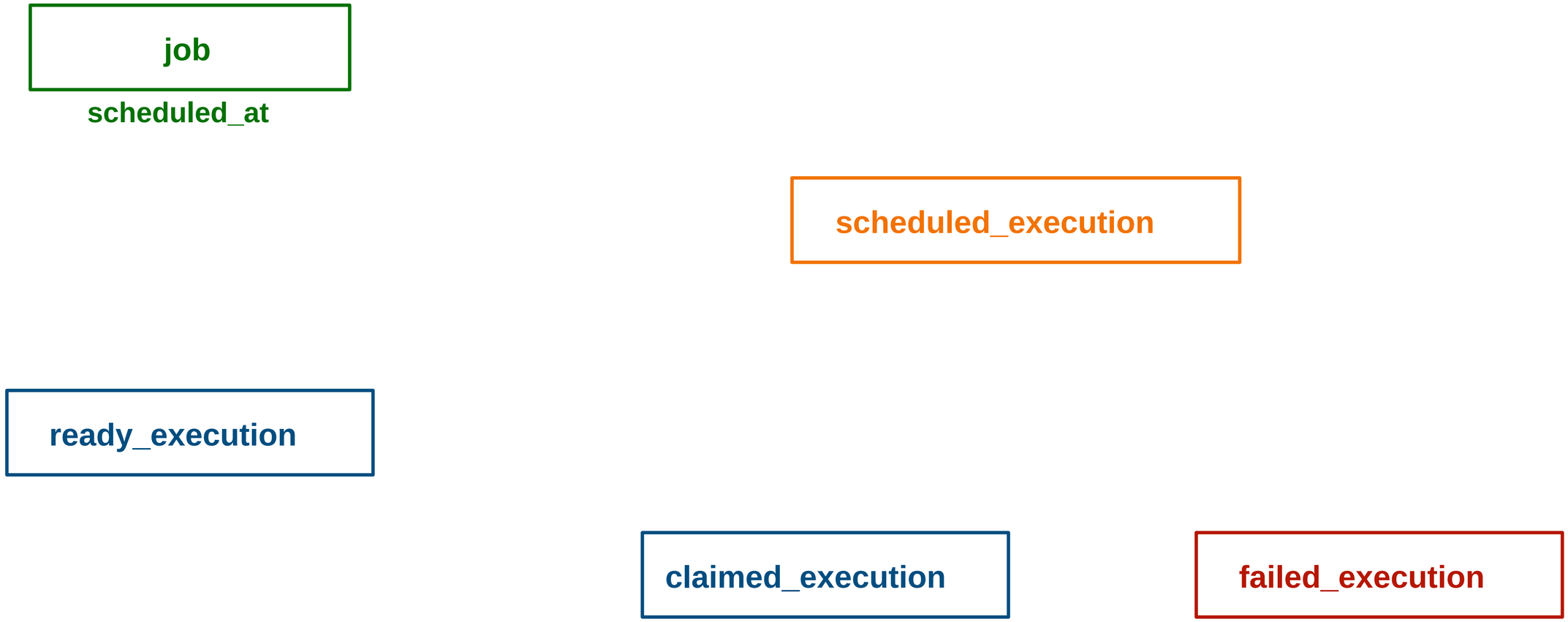
Solid Queue の内部実装の理解

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



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

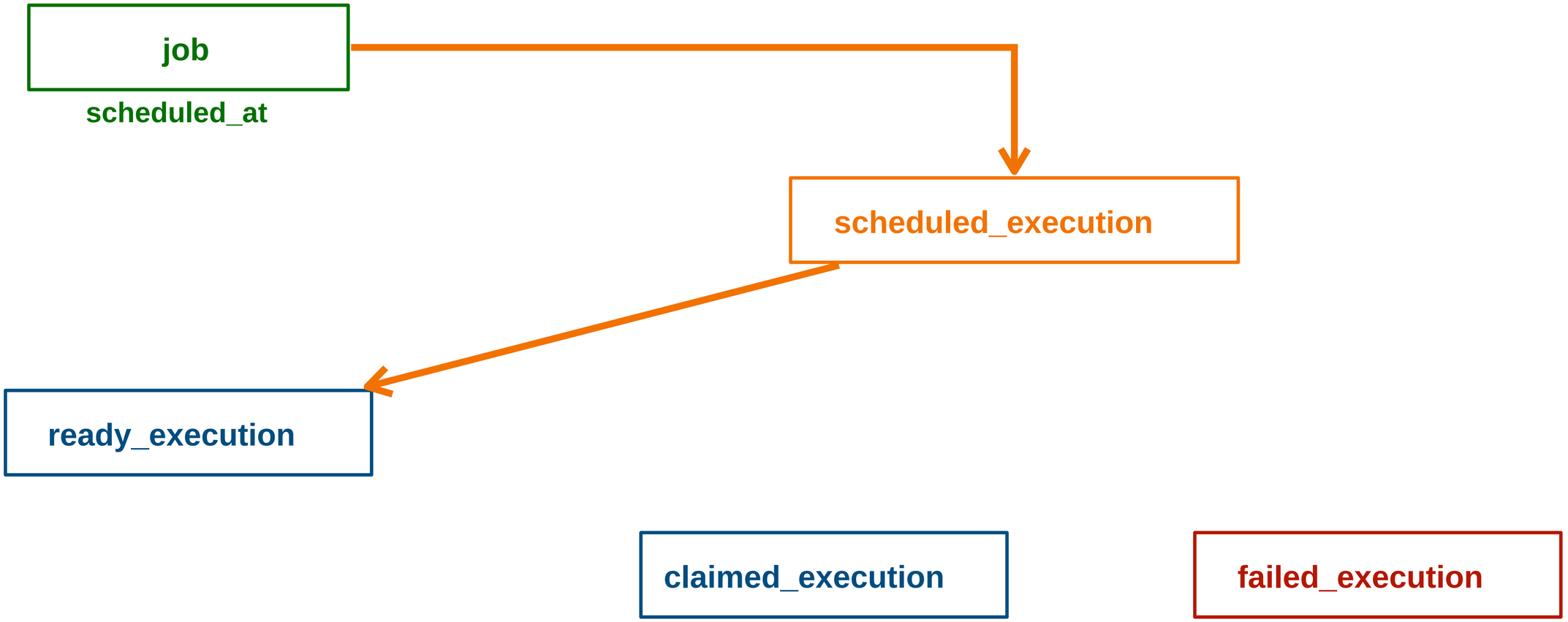
ready_execution

claimed_execution

failed_execution

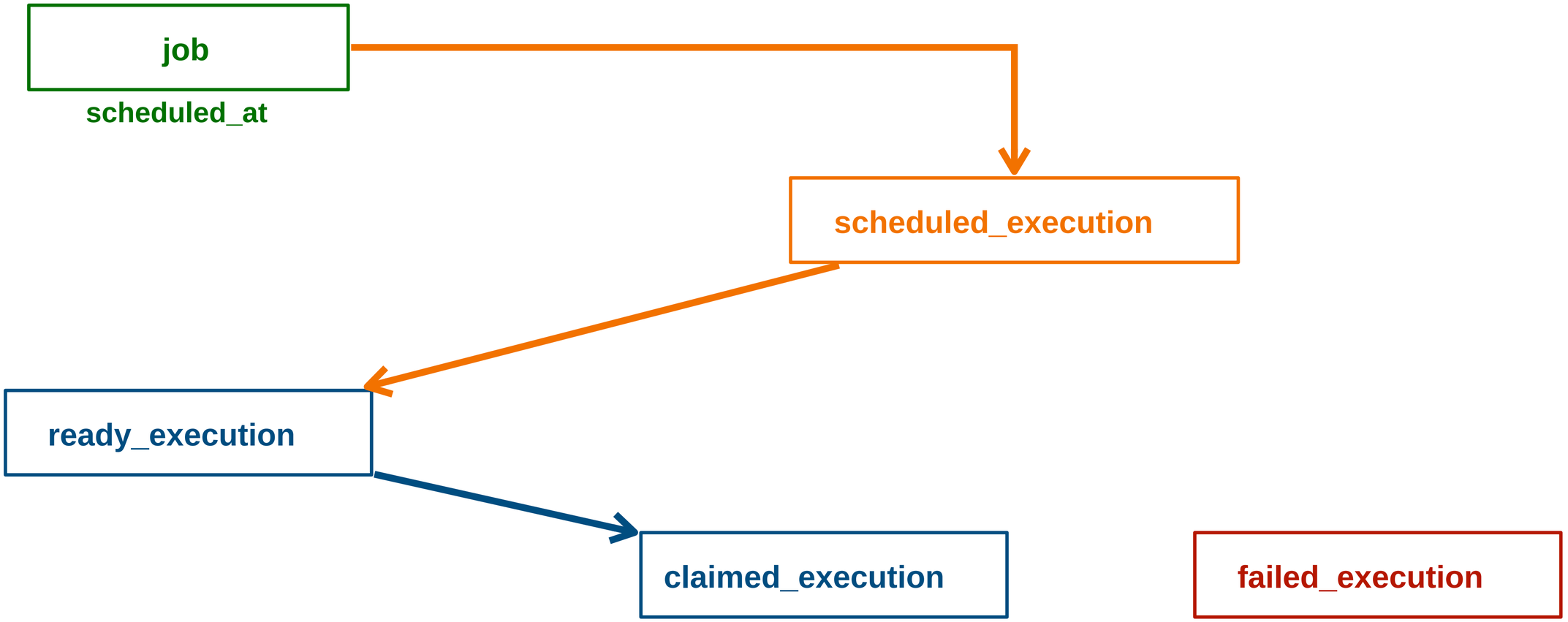
Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



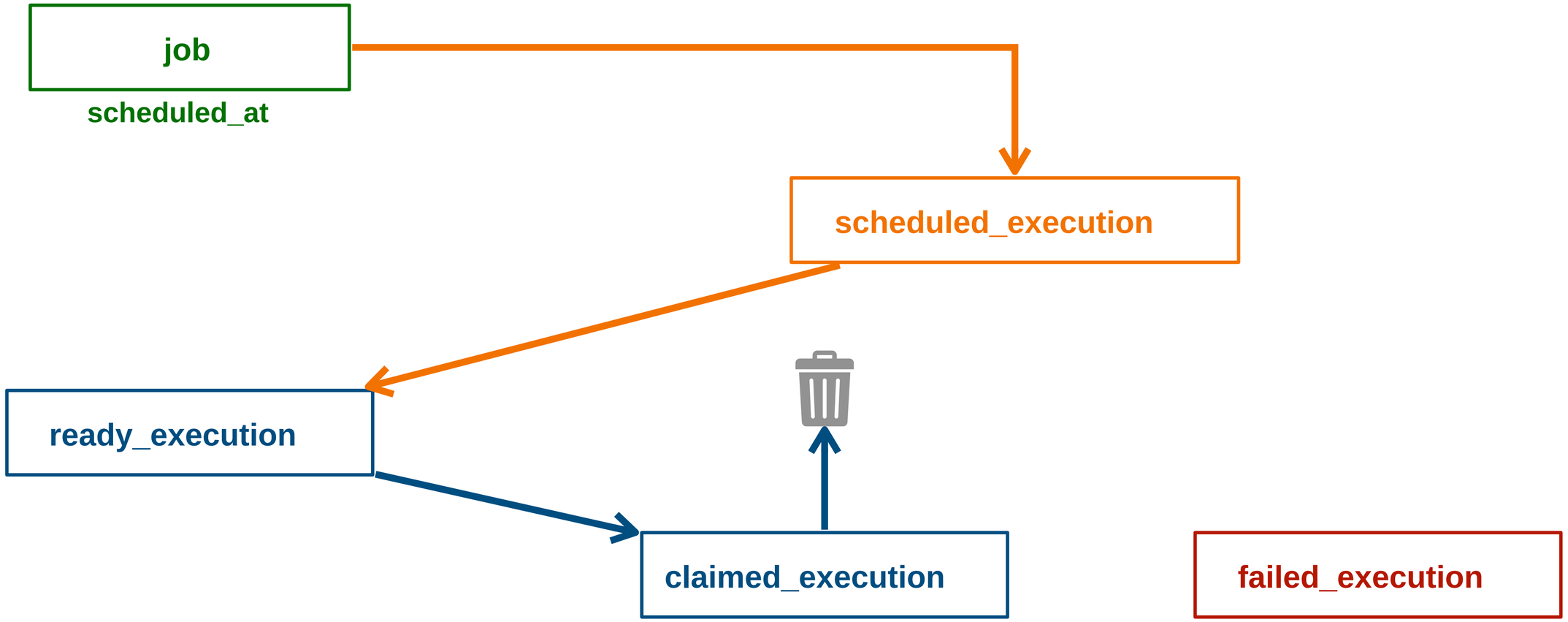
Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



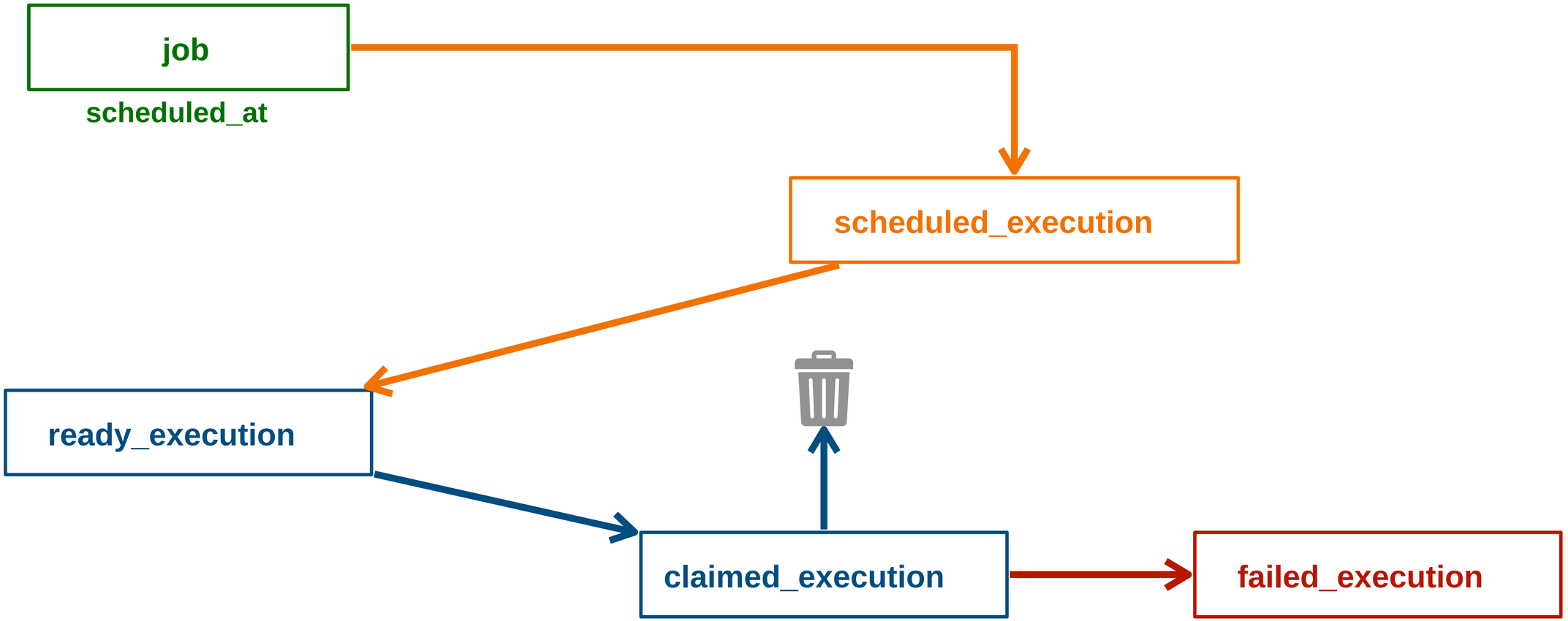
Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



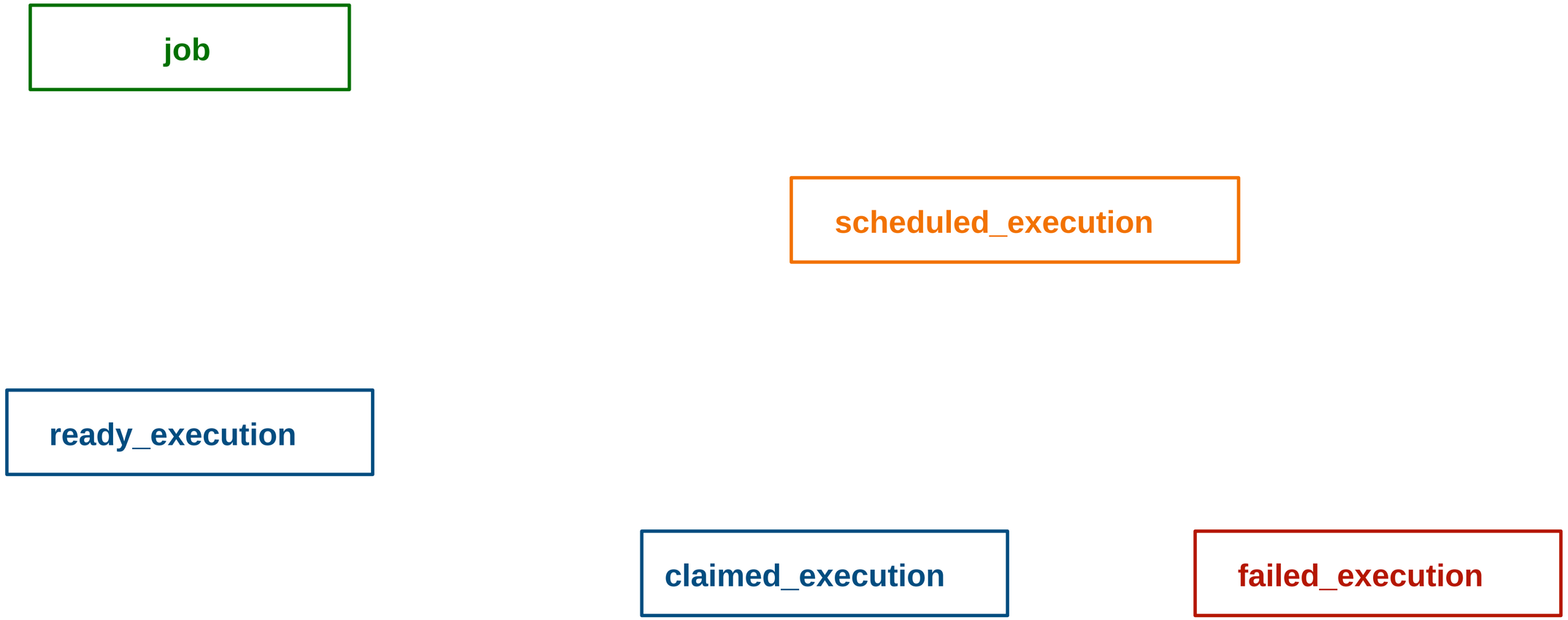
Solid Queue の内部実装の理解

次の実行時刻が決まっているジョブの場合



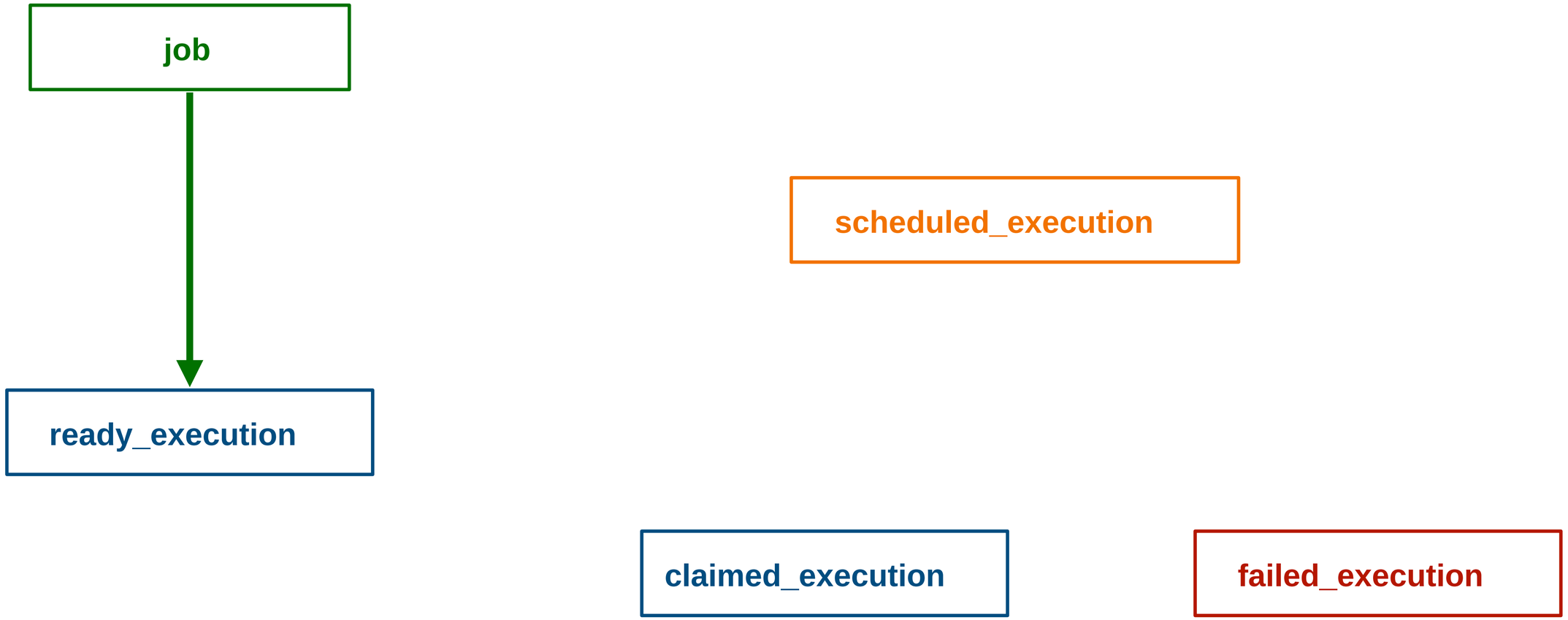
Solid Queue の内部実装の理解

すぐ実行したいジョブの場合



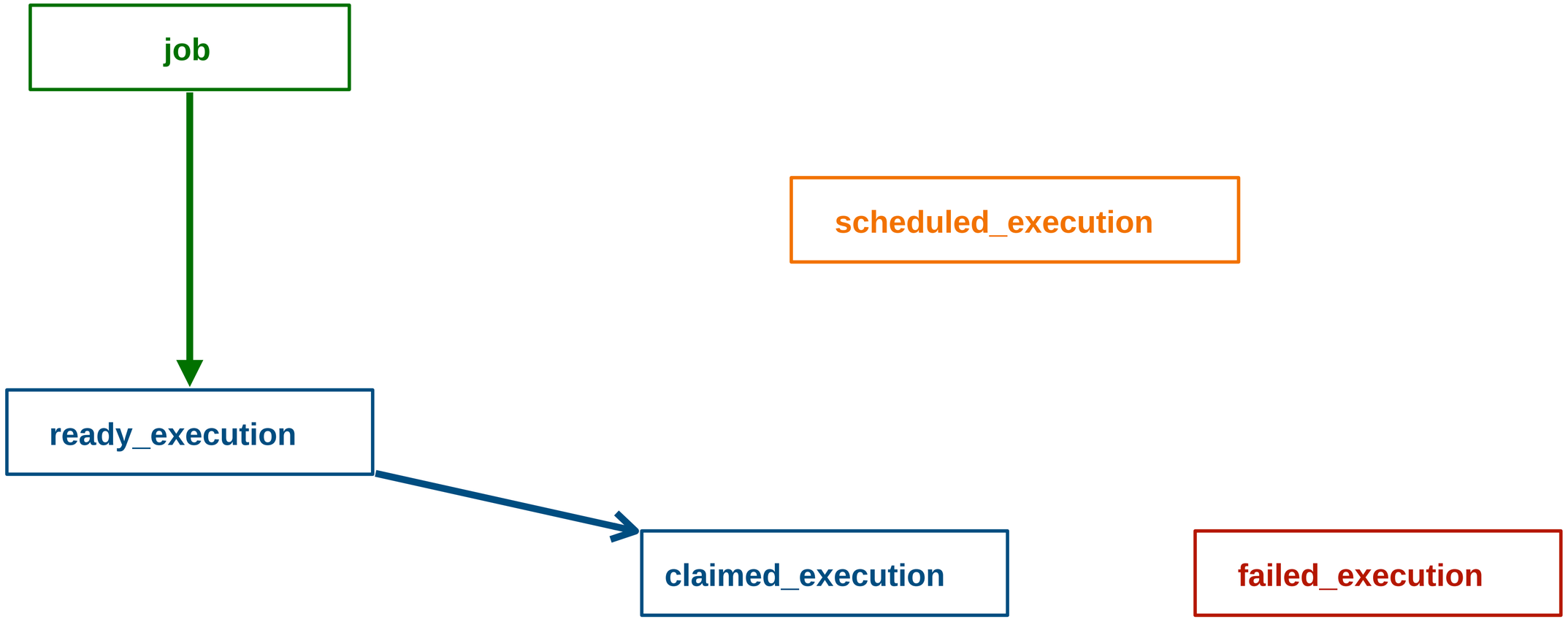
Solid Queue の内部実装の理解

すぐ実行したいジョブの場合



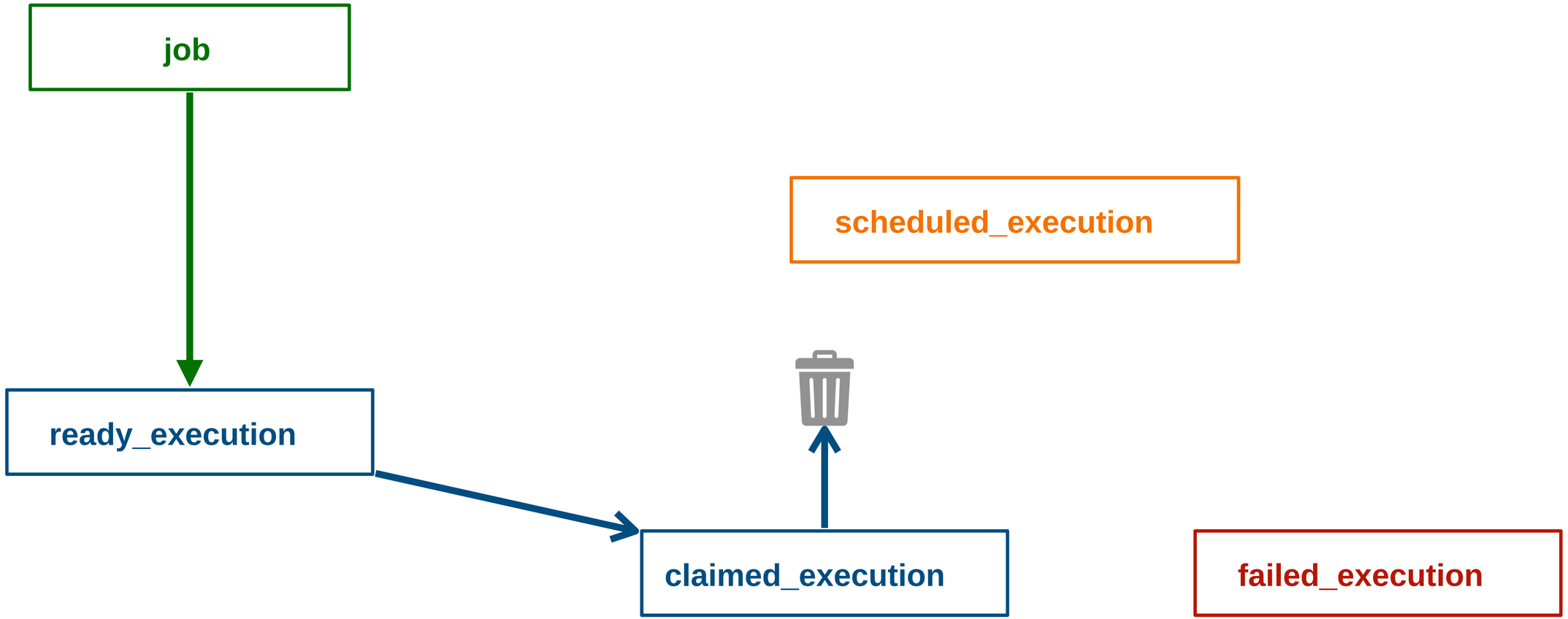
Solid Queue の内部実装の理解

すぐ実行したいジョブの場合



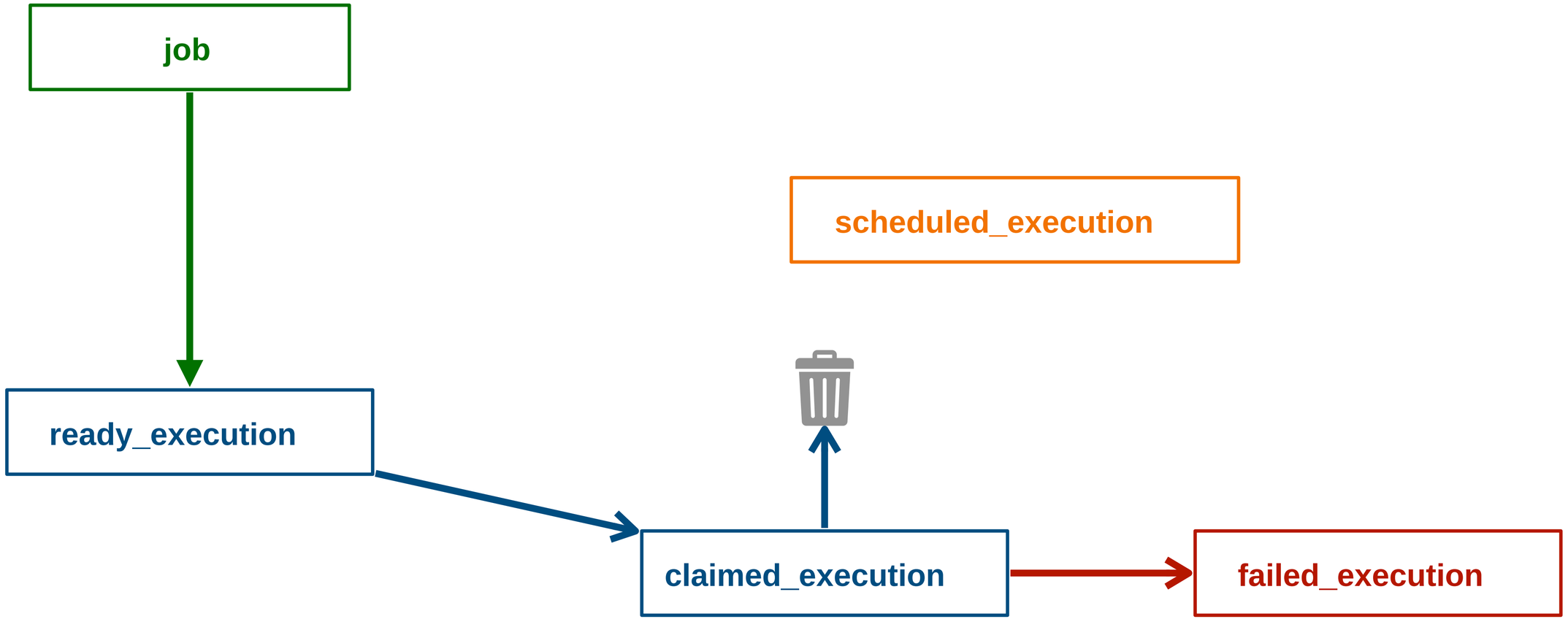
Solid Queue の内部実装の理解

すぐ実行したいジョブの場合

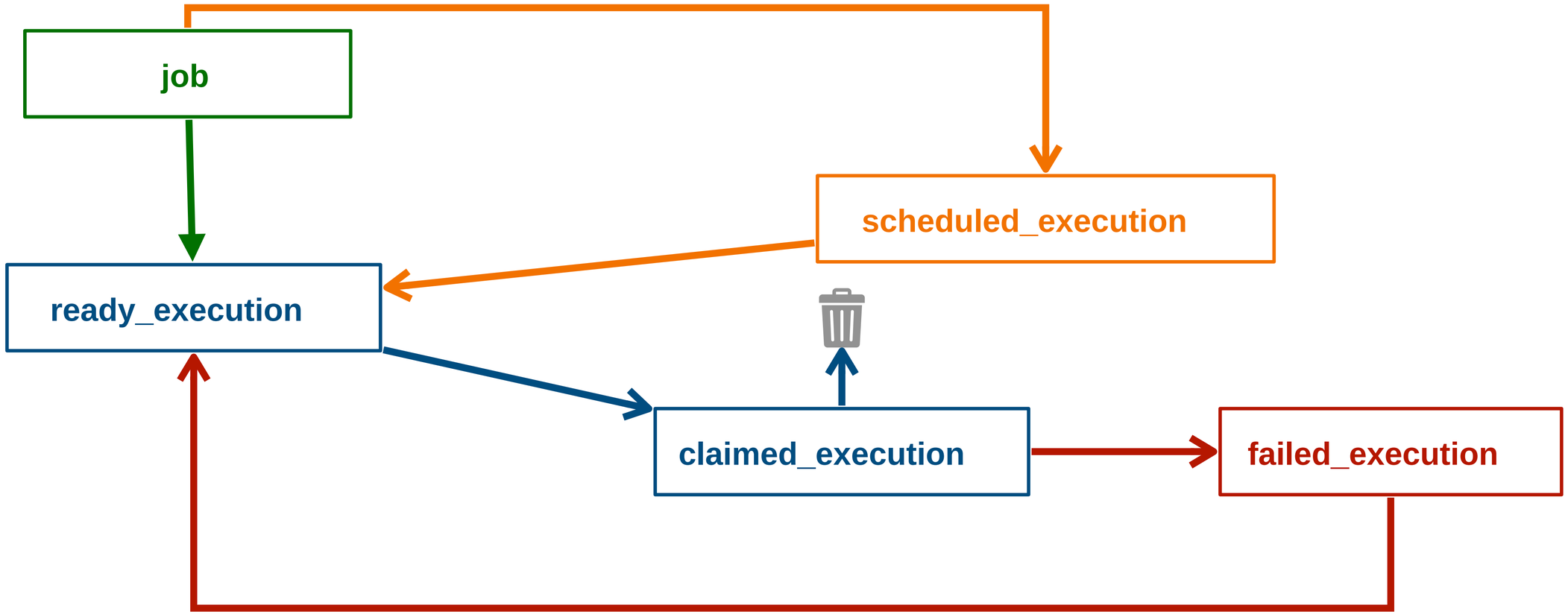


Solid Queue の内部実装の理解

すぐ実行したいジョブの場合

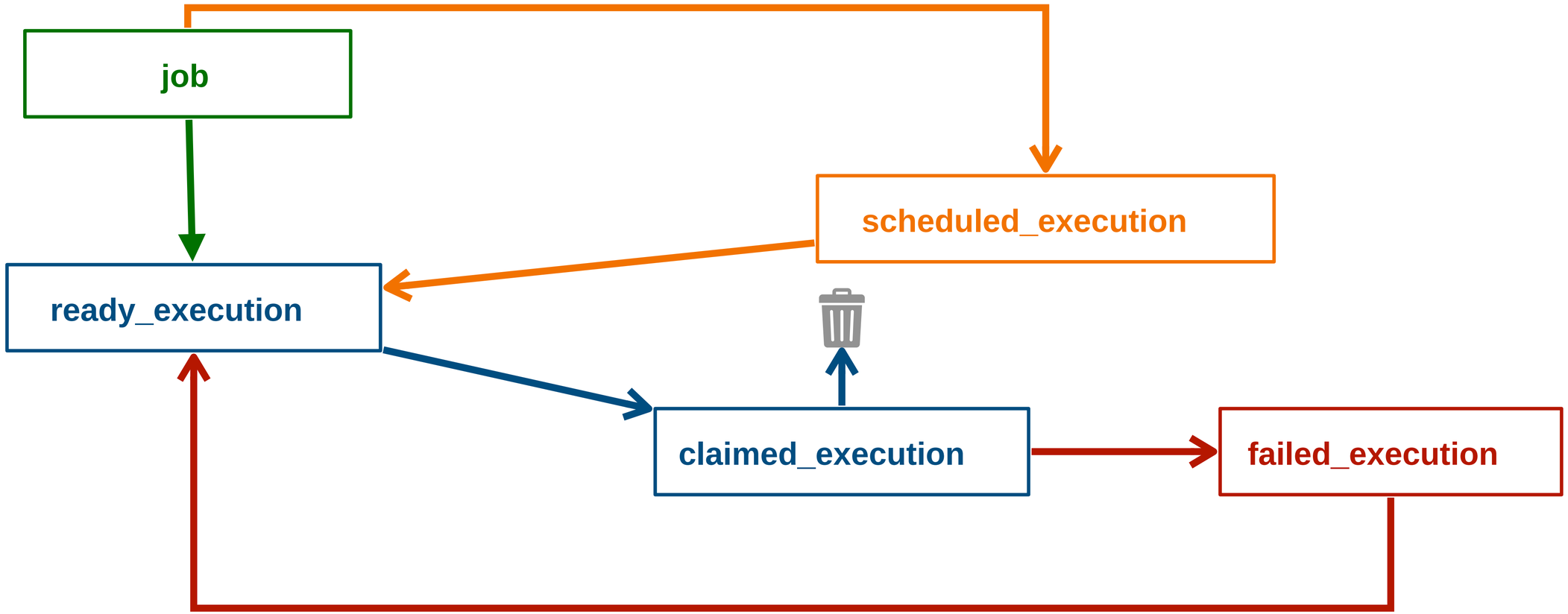


Solid Queue の内部実装の理解



Solid Queue の内部実装の理解

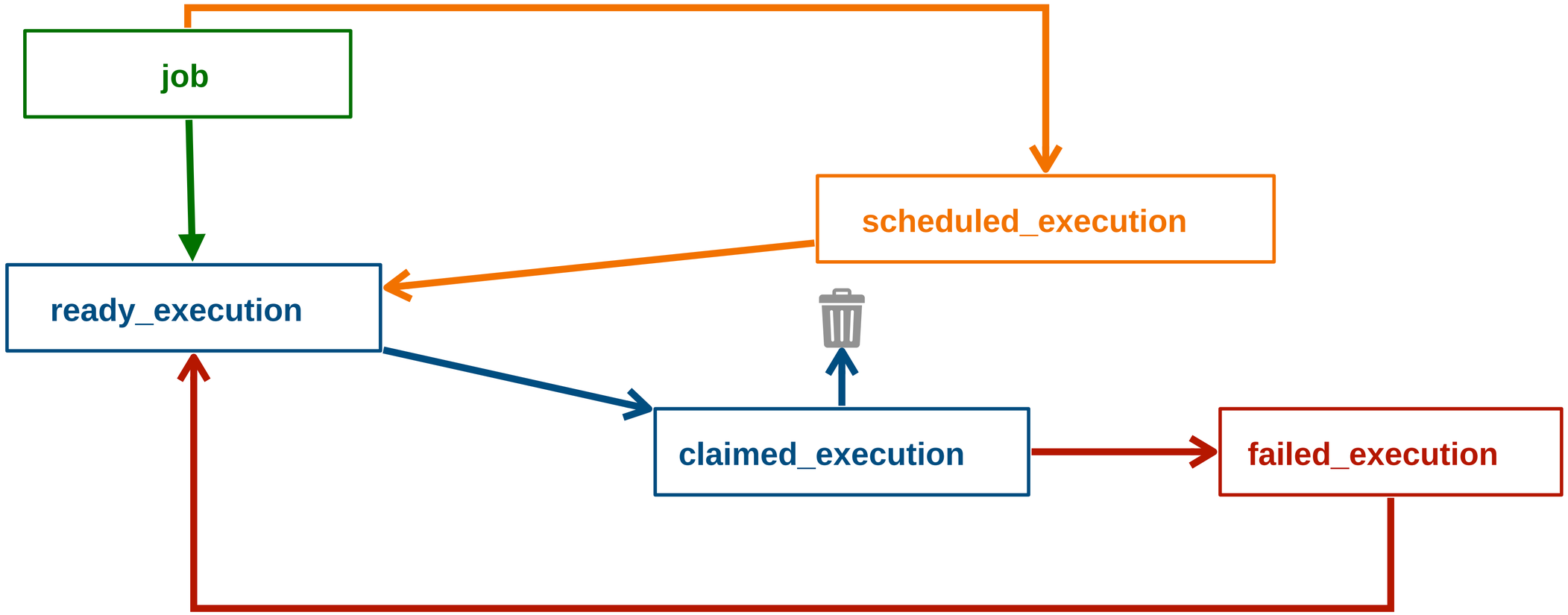
監視時はそれぞれのテーブルを頭に入れてクエリを書く



Solid Queue の内部実装の理解

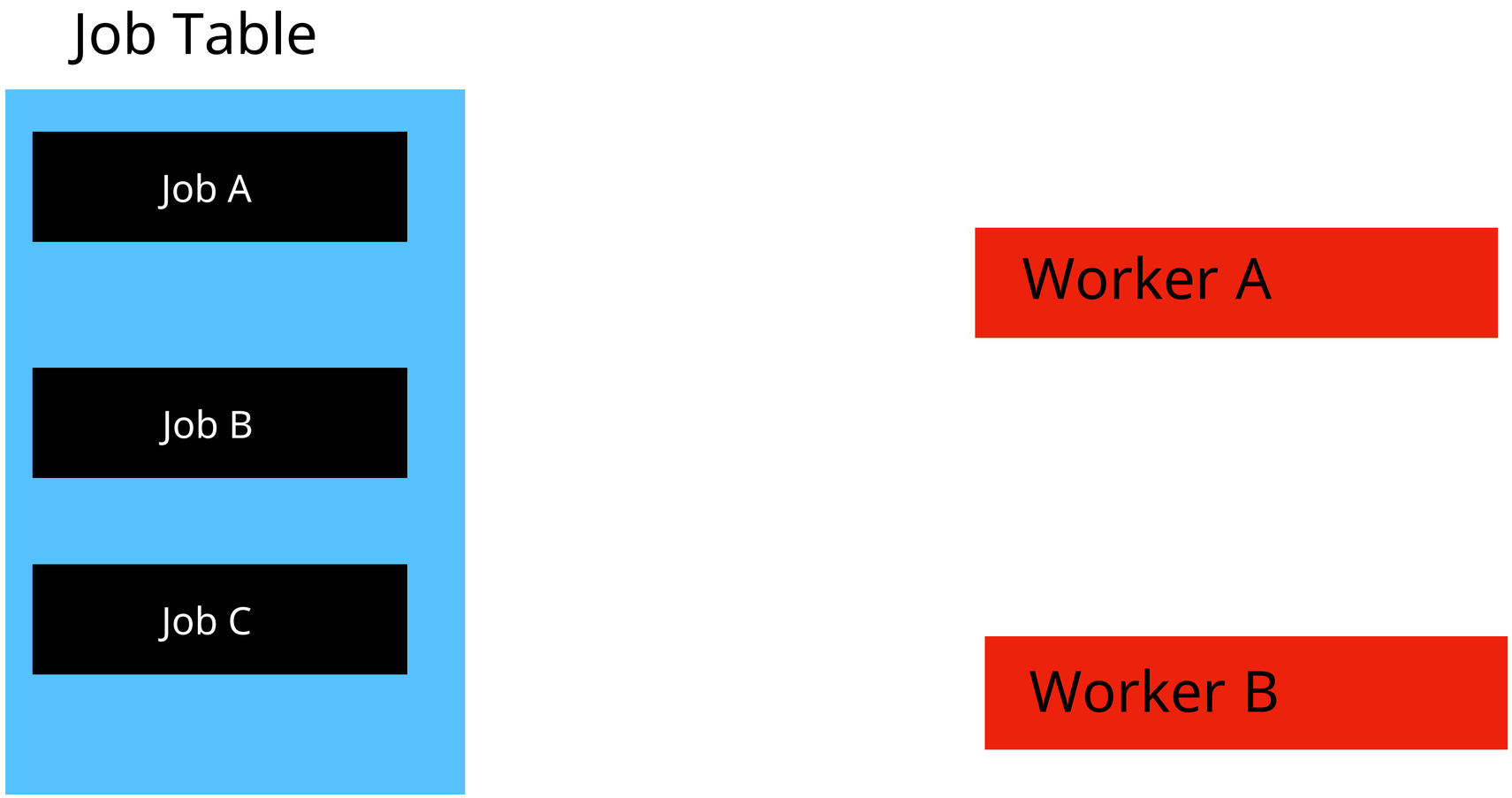
監視時はそれぞれのテーブルを頭に入れてクエリを書く

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

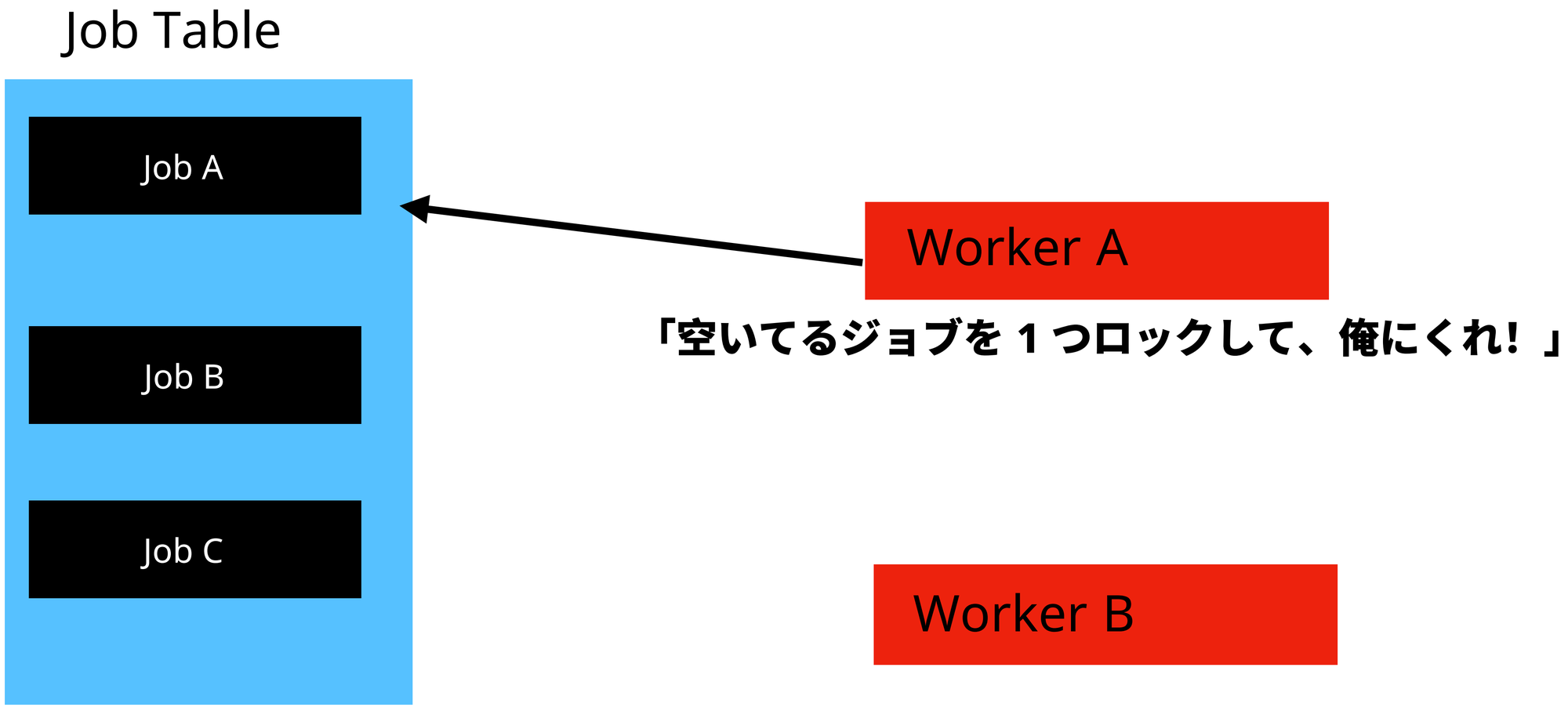


Solid Queue で解決する線形スケールアウト

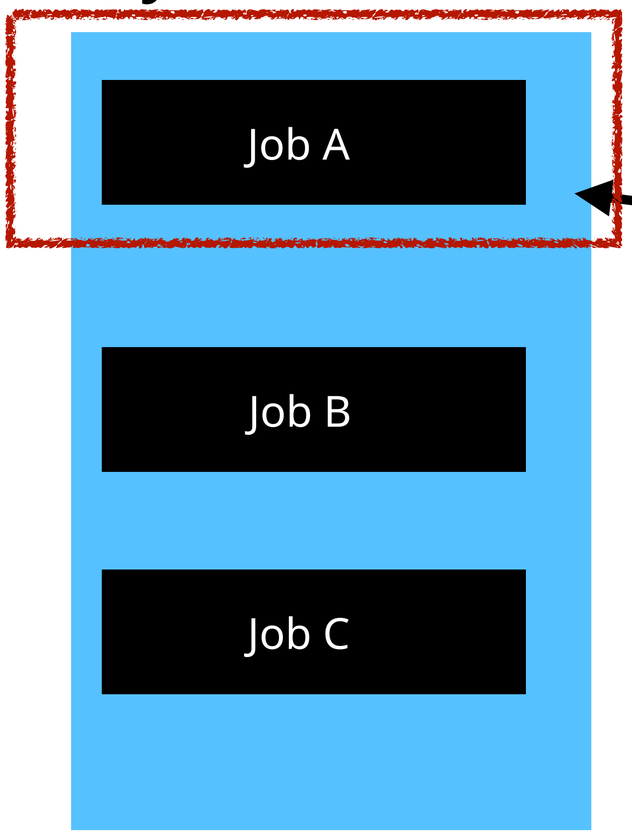
Solid Queue で解決する線形スケールアウト



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



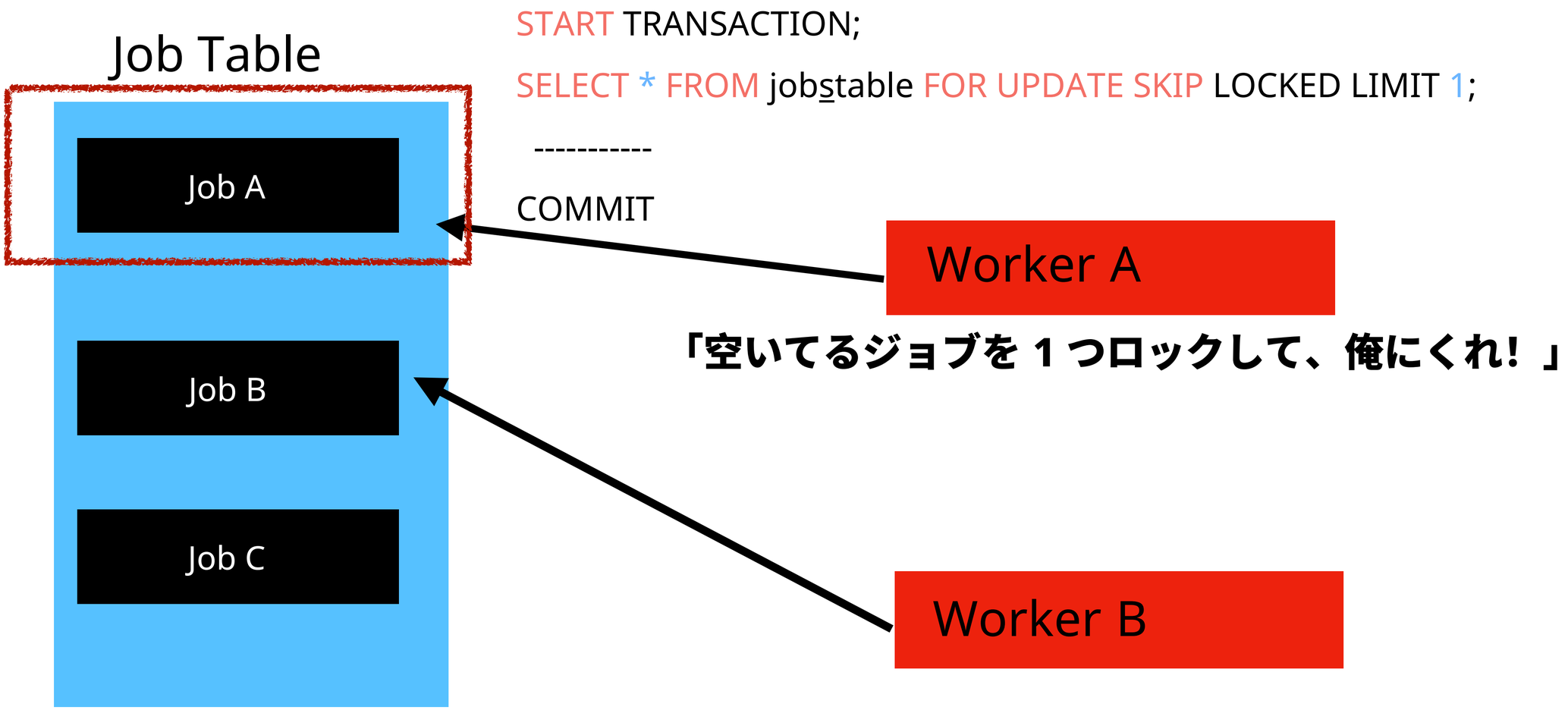

```
START TRANSACTION;
SELECT * FROM jobstable FOR UPDATE SKIP LOCKED LIMIT 1;
  ----------
COMMIT
```

Worker A

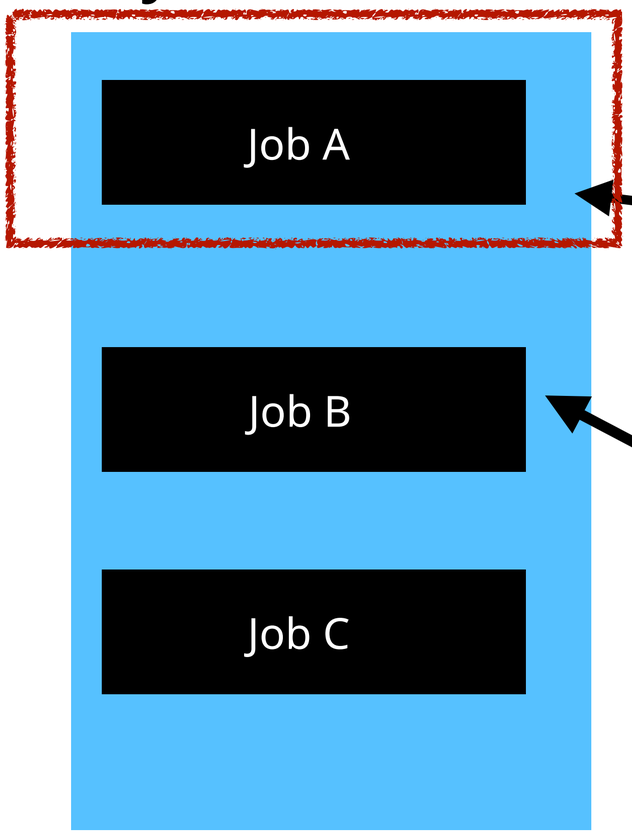
「空いてるジョブを 1 つロックして、俺にくれ！」

Worker B

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 人にした

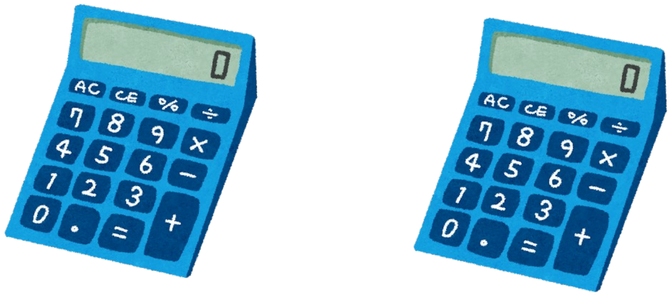
Solid Queue の線形的なスケールアウト

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、お客さんは

空いている方に行ってもらうよ



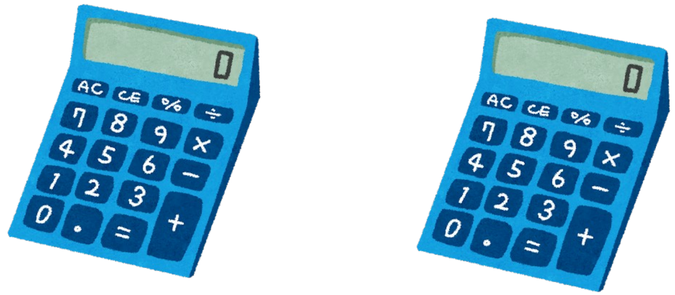
Solid Queue の線形的なスケールアウト

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、お客さんは

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さんを捌くスピードが…

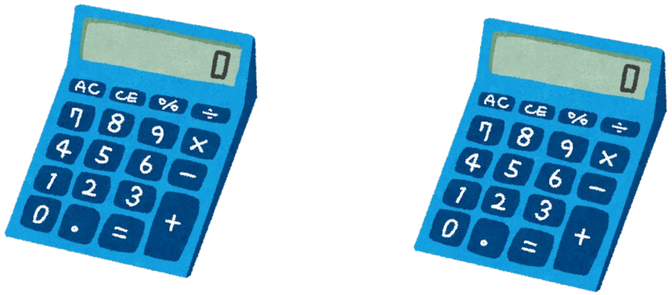
Solid Queue の線形的なスケールアウト

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、お客さんは

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さんを捌くスピードが…

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

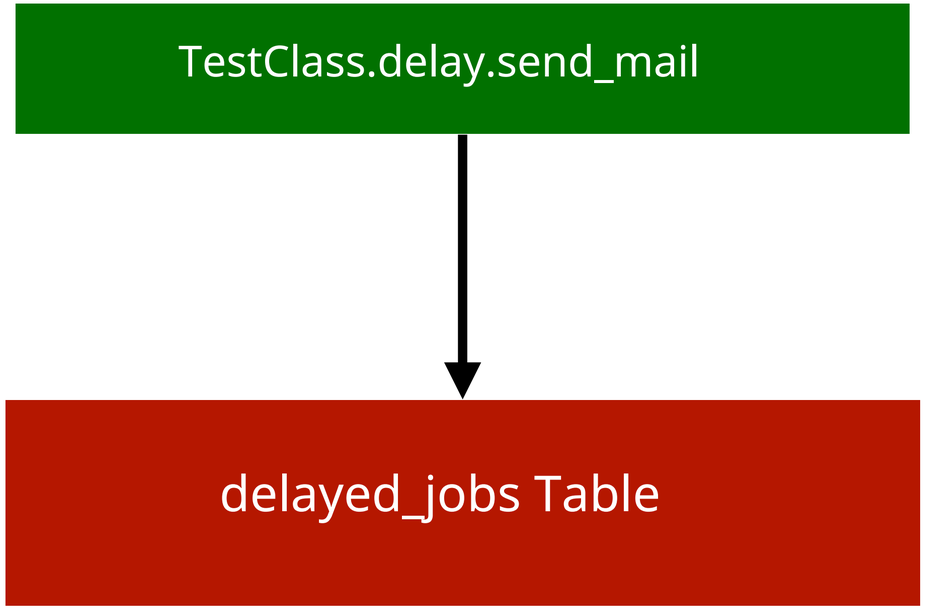
```
TestClass.delay.send_mail( user_id: user.id)
```

TestClass.delay.send_mail

delayed_jobs Table

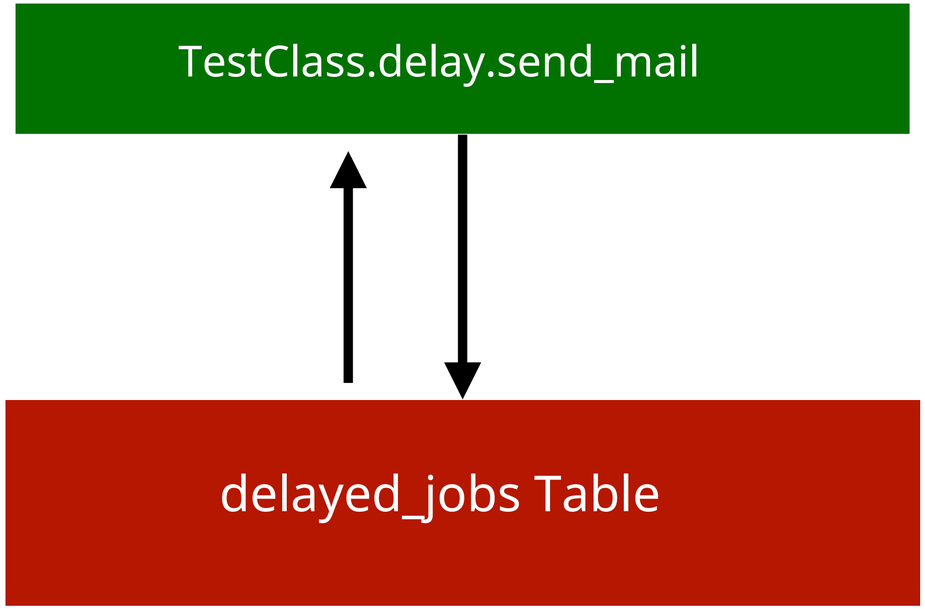
".delay" Method

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



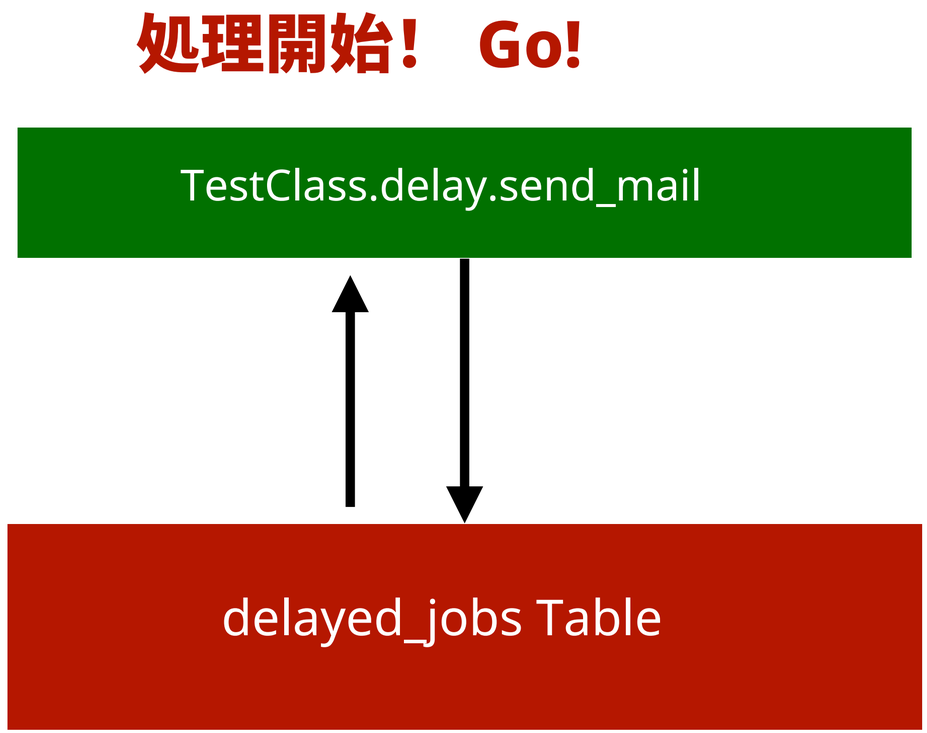
".delay" Method

```
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)
```

".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
)
```

SolidQueue を install しても Delayed で動作

```
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

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！

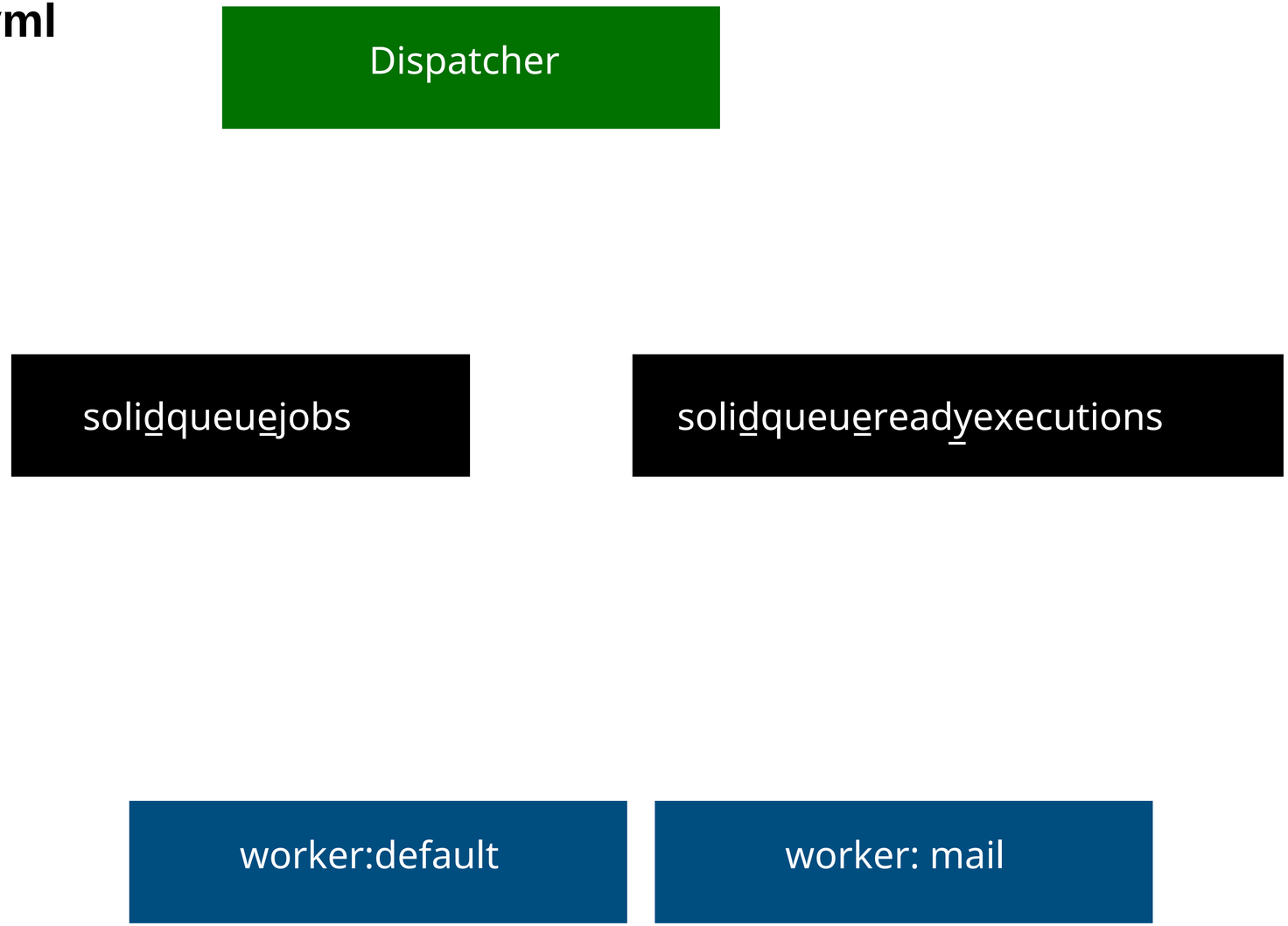
移行手順

- ✅ 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

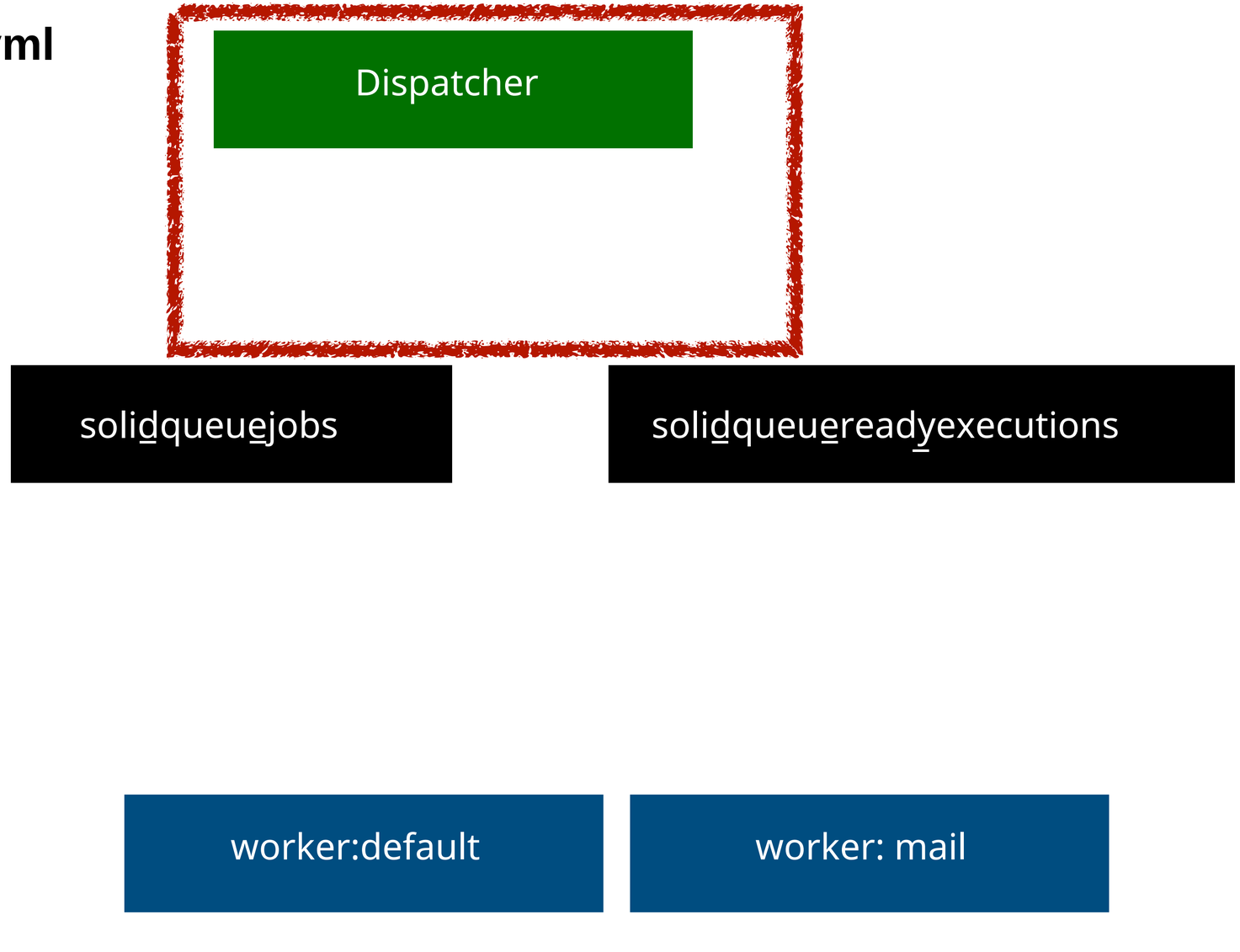
```yaml
production:
  dispatchers:
    - polling_interval: 1
      batch_size: 500
  workers:
    - queues: "default"
      threads: 2
      processes: 5
      polling_interval: 5
    - queues: "mail"
      threads: 2
      processes: 5
      polling_interval: 1
```



Solid Queue Configuration

config/solid_queue/queue_config.yml

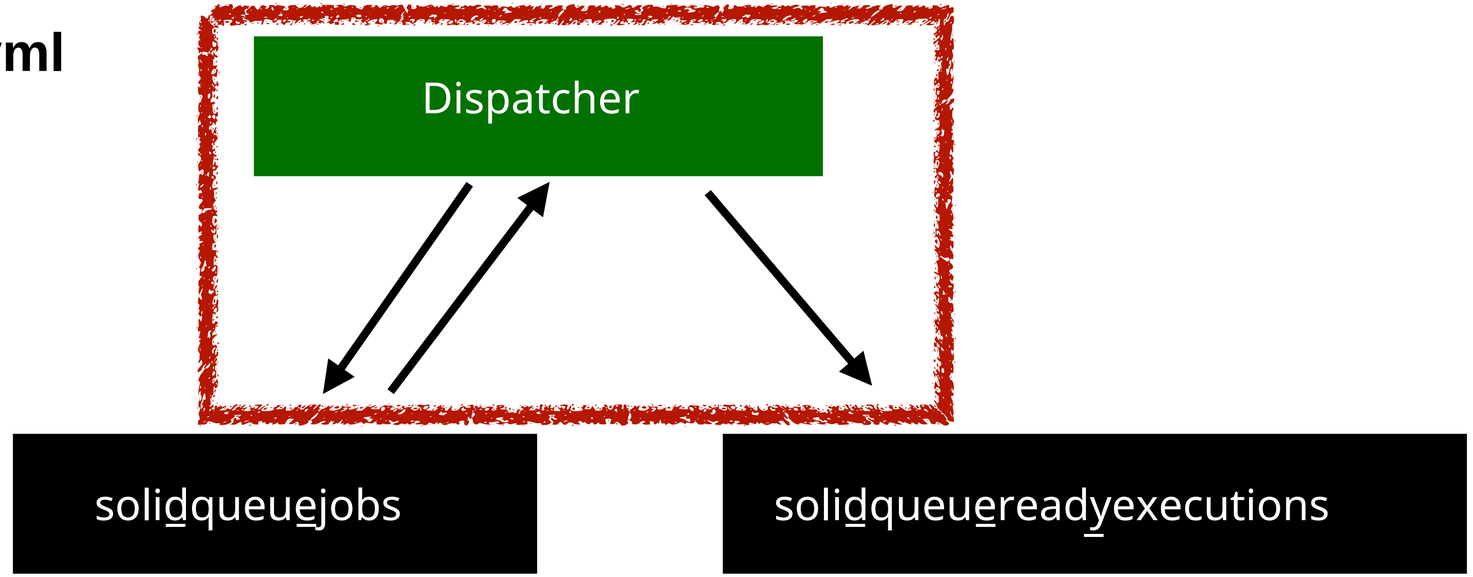
```yaml
production:
  dispatchers:
    - polling_interval: 1
      batch_size: 500
  workers:
    - queues: "default"
      threads: 2
      processes: 5
      polling_interval: 5
    - queues: "mail"
      threads: 2
      processes: 5
      polling_interval: 1
```



Solid Queue Configuration

config/solid_queue/queue_config.yml

```yaml
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
```



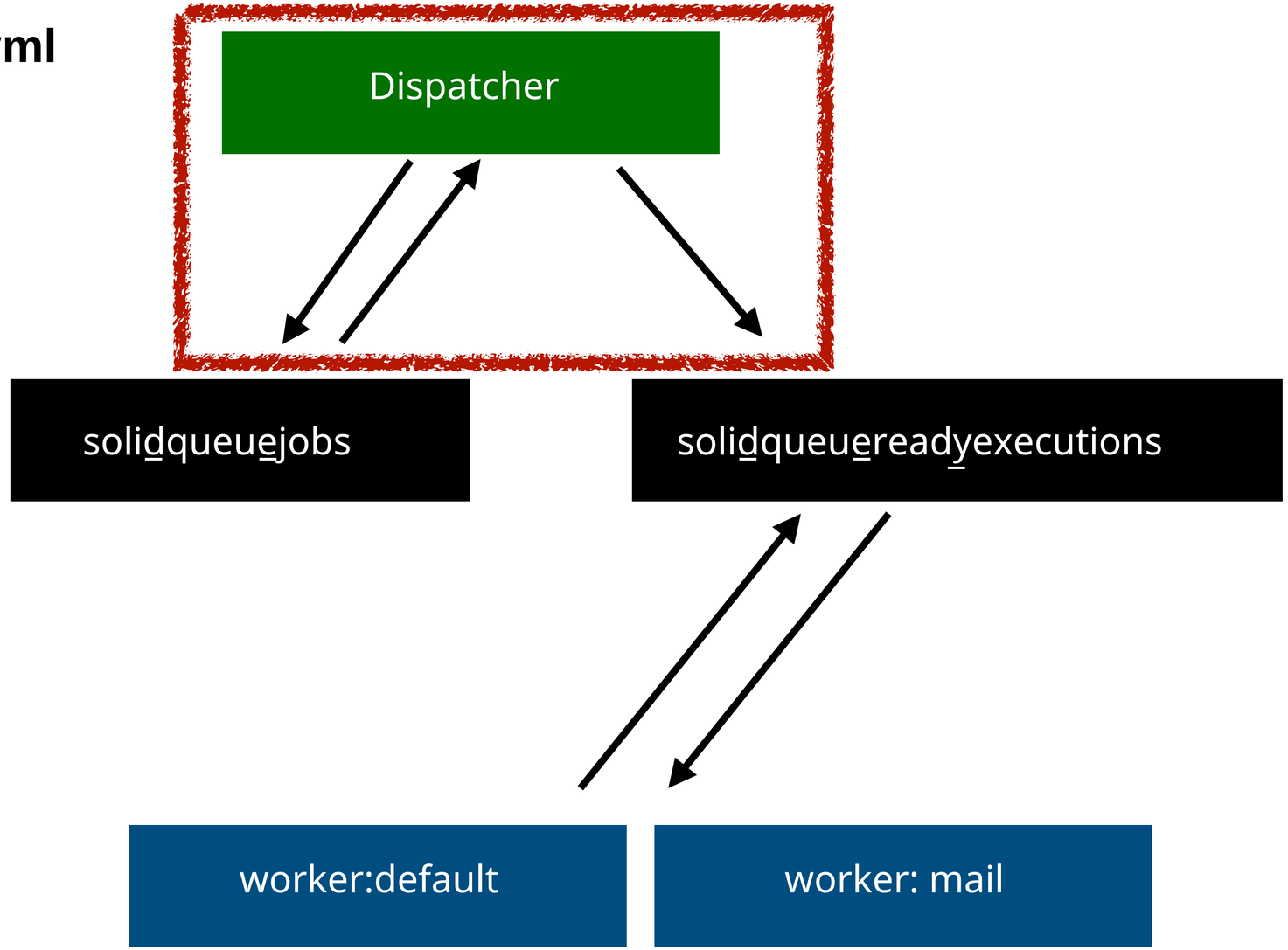
worker:default

worker: mail

Solid Queue Configuration

config/solid_queue/queue_config.yml

```yaml
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

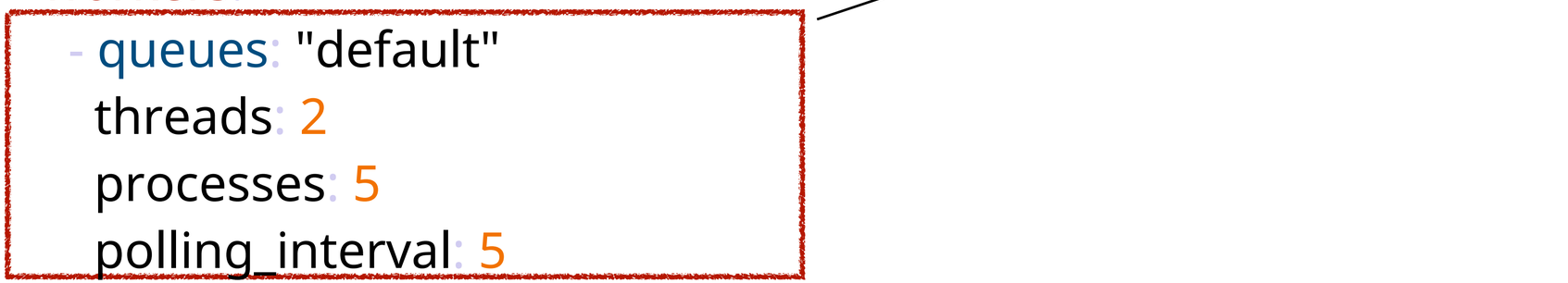
config/solid_queue/queue_config.yml

```yaml
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

```yaml
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

```yaml
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
```

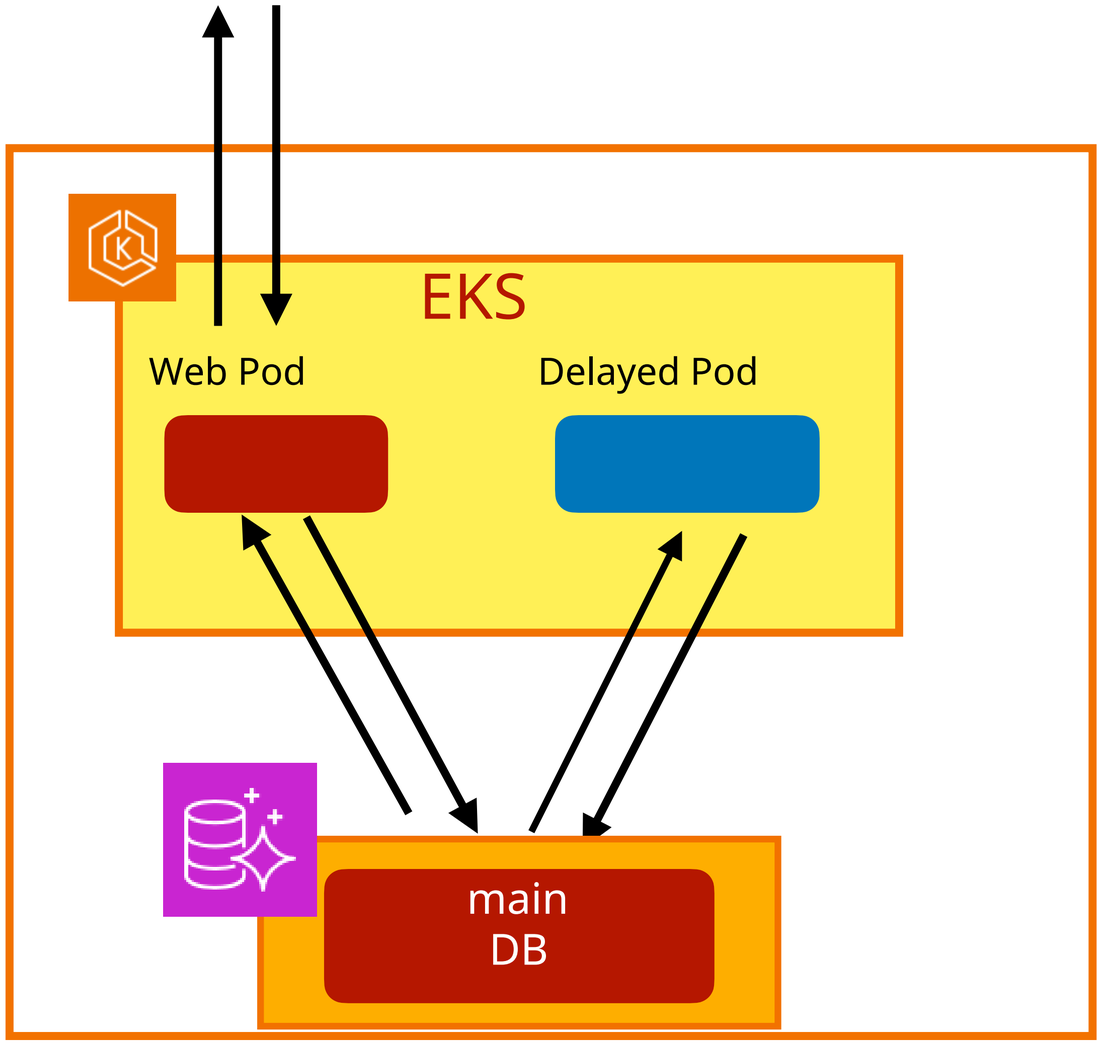
ワーカーとディスパッチャをポーリングするときの Active Record ログを抑制する

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

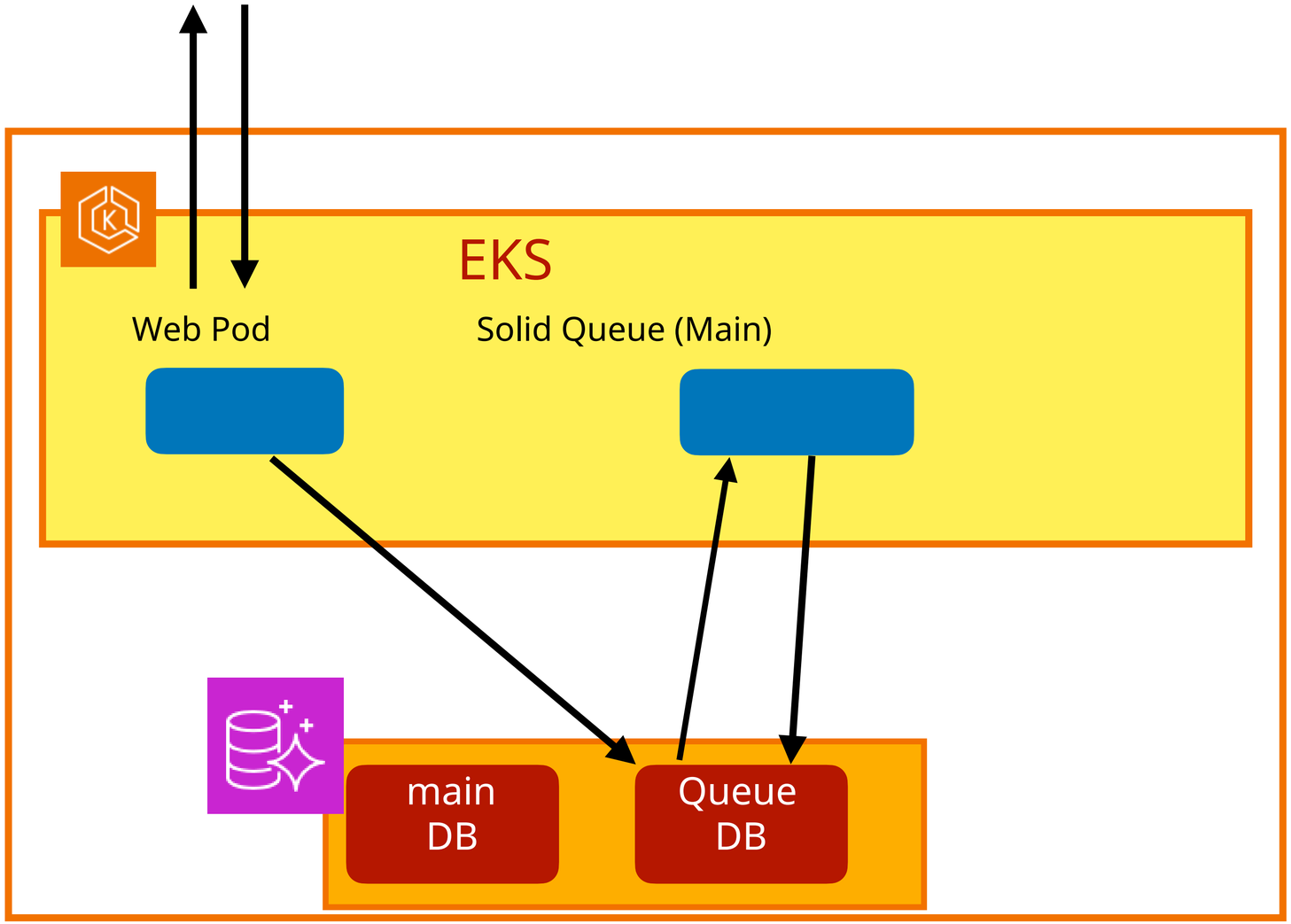
完了したジョブをテーブルに残さない

```
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 の機能として

提供されてるんだぜ!

- 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

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)

自前実装しました!

/ 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得するときに分単位になってしまうのを秒単位に (#16699)

4d419a9 · 4 months ago History

Code Blame 157 lines (139 loc) · 5.75 KB Raw

```ruby
# frozen_string_literal: true

# 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
class Monitoring::ReportingSolidQueueJob < ApplicationJob
  include Monitoring::SolidQueueMetricsHelper
  include SolidQueueLogger
  include SolidQueueRetryHandler
  self.queue_adapter = :solid_queue unless Rails.env.test?
  around_perform :solid_queue_logger
  queue_as :teachme_monitoring

  METRICS = %w[
    future_count
    erroring_count
    failed_count
    working_count
    workable_count
    max_age
  ].freeze

  # 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
  # max_lock_age
  # alert_age_percent

  IGNORE_MONITORING_CLASS = %w[
    Monitoring::ReportingSolidQueue
  ].freeze

  AGE_ALERT_THRESHOLD = 30.minutes

  def perform
    queues = fetch_queues
    ActiveSupport::Notifications.instrument('solid_queue.monitor.run', default_tags) do
      METRICS.each { |metric| emit_metric!(metric, queues) }
    end
```

監視するぞ！ (Solid Queue 編)

自前実装しました!

/ 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得するときに分単位になってしまうのを秒単位に (#16699)

4d419a9 · 4 months ago History

Code Blame 157 lines (139 loc) · 5.75 KB Raw

```ruby
# frozen_string_literal: true

# 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
class Monitoring::ReportingSolidQueueJob < ApplicationJob
  include Monitoring::SolidQueueMetricsHelper
  include SolidQueueLogger
  include SolidQueueRetryHandler
  self.queue_adapter = :solid_queue unless Rails.env.test?
  around_perform :solid_queue_logger
  queue_as :teachme_monitoring

  METRICS = %w[
    future_count
    erroring_count
    failed_count
    working_count
    workable_count
    max_age
  ].freeze

  # 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
  # max_lock_age
  # alert_age_percent

  IGNORE_MONITORING_CLASS = %w[
    Monitoring::ReportingSolidQueue
  ].freeze

  AGE_ALERT_THRESHOLD = 30.minutes

  def perform
    queues = fetch_queues
    ActiveSupport::Notifications.instrument('solid_queue.monitor.run', default_tags) do
      METRICS.each { |metric| emit_metric!(metric, queues) }
    end
```

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

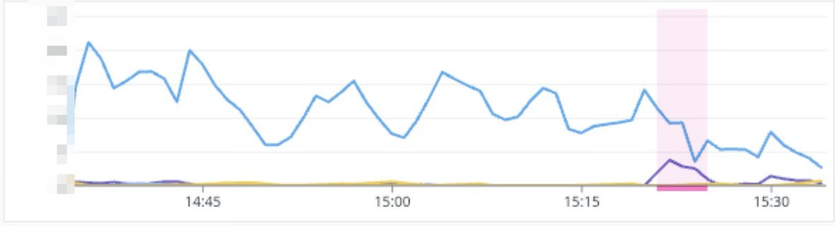
Share | Anomalies | Show Overlays | Configure | Add Widgets

Saved Views | Filter by: rails_env: production | backend: * | queue: *

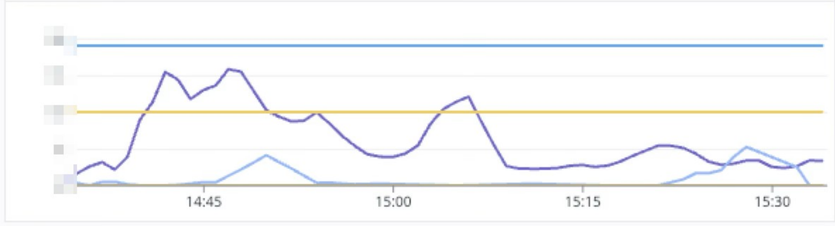
UTC+09:00 | 1h | Past 1 Hour

処理中のジョブ数

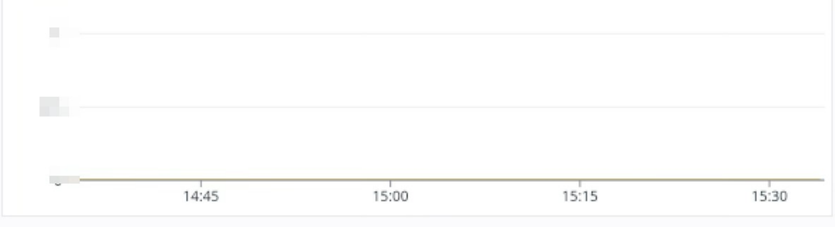
処理待ちのジョブ数



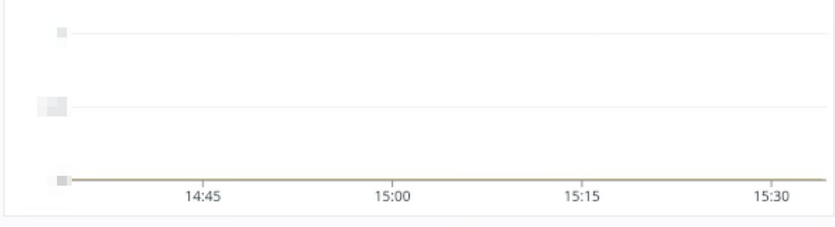
実行予定のジョブ数



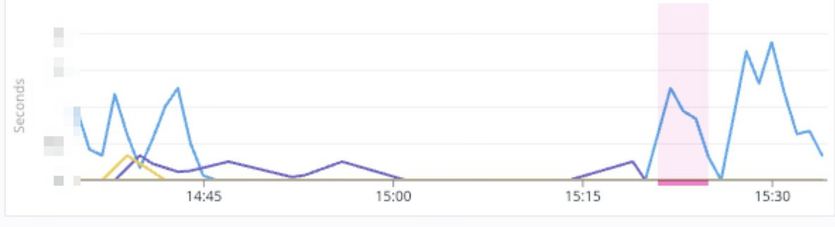
エラー発生回数



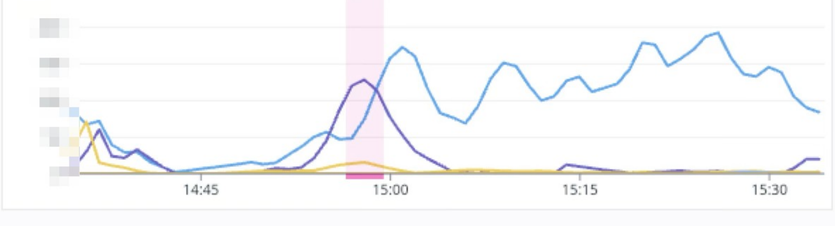
失敗したジョブ数



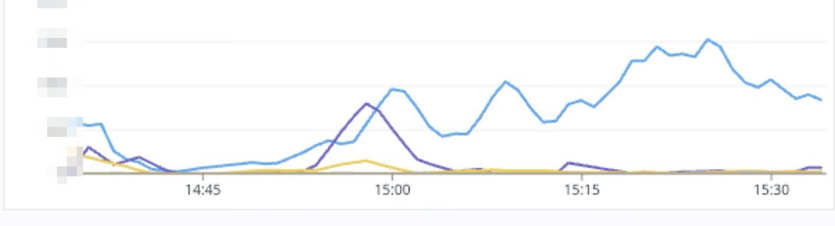
最も古い処理待ちジョブの経過時間（秒）



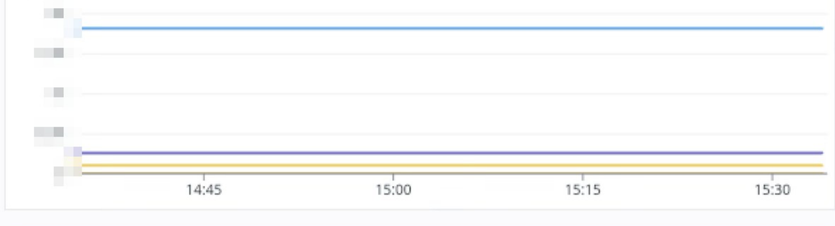
最も長く実行されているジョブの経過時間（秒）



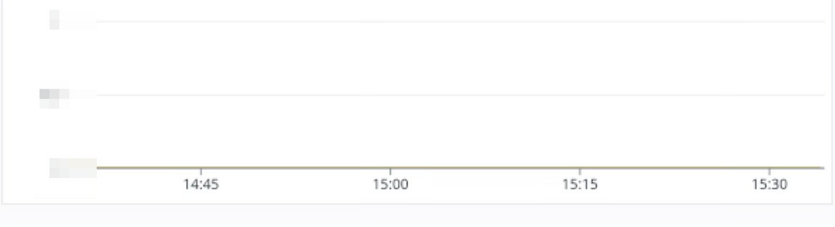
平均実行時間（秒）



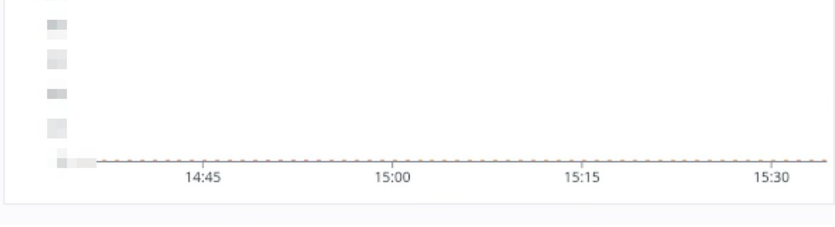
リトライ待機中ジョブ数



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!

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

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

Delayed -> SolidQueue!

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

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

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

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

Delayed -> SolidQueue!

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

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

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

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

delayed -> solidqueue に変更!

Delayed -> SolidQueue!

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

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

↓

```ruby
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
- [redacted]

Input

notion

[redacted]

duvel RR

基盤設計と導入

apps

[redacted]

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
- [redacted]

Input

notion

[redacted]

duvel RR

基盤設計と導入

apps

[redacted]

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
- [redacted]

Input

notion

[redacted]

duvel RR

基盤設計と導入

apps

[redacted]

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

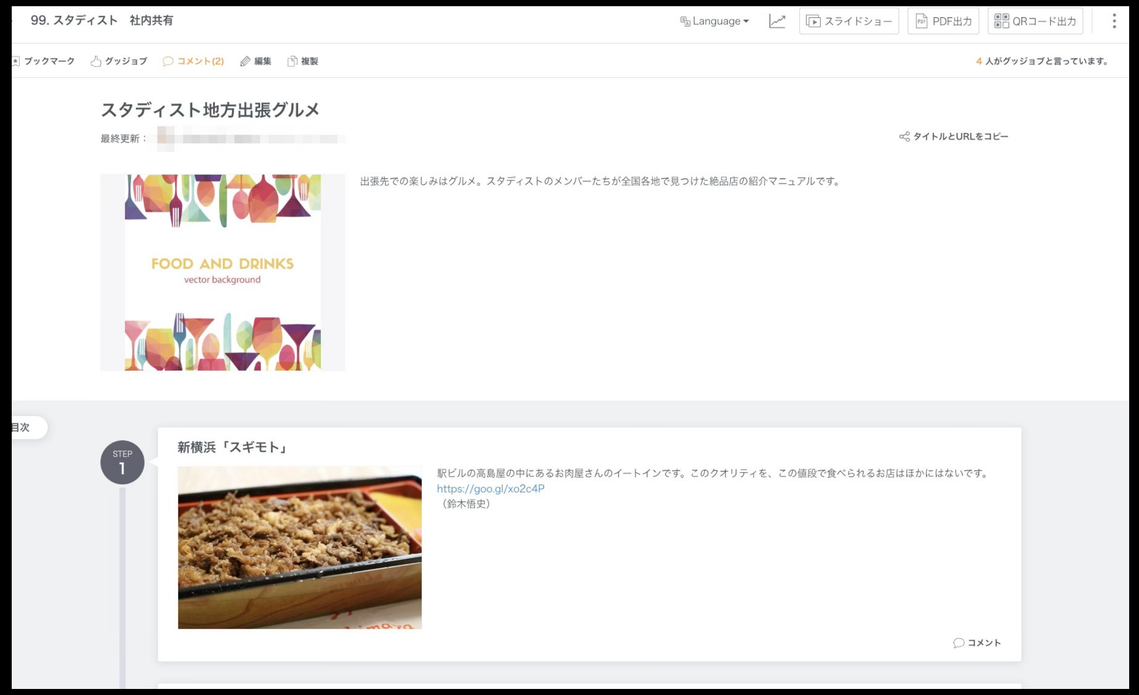
- ✅ .delay メソッドの撲滅
- ✅ ActiveJob で明示的に Delayed を指定
- ✅ SolidQueue 設定
- ✅ 監視
- ⬜ SolidQueue を利用するように全 Job を変更
- ⬜ gem uninstall delayed

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

enqueueaftertransactioncommit 効いてない問題

enqueue_after_transaction_commit 設定されてなくない?



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

enqueueaftertransactioncommit 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。

enqueueaftertransactioncommit とは

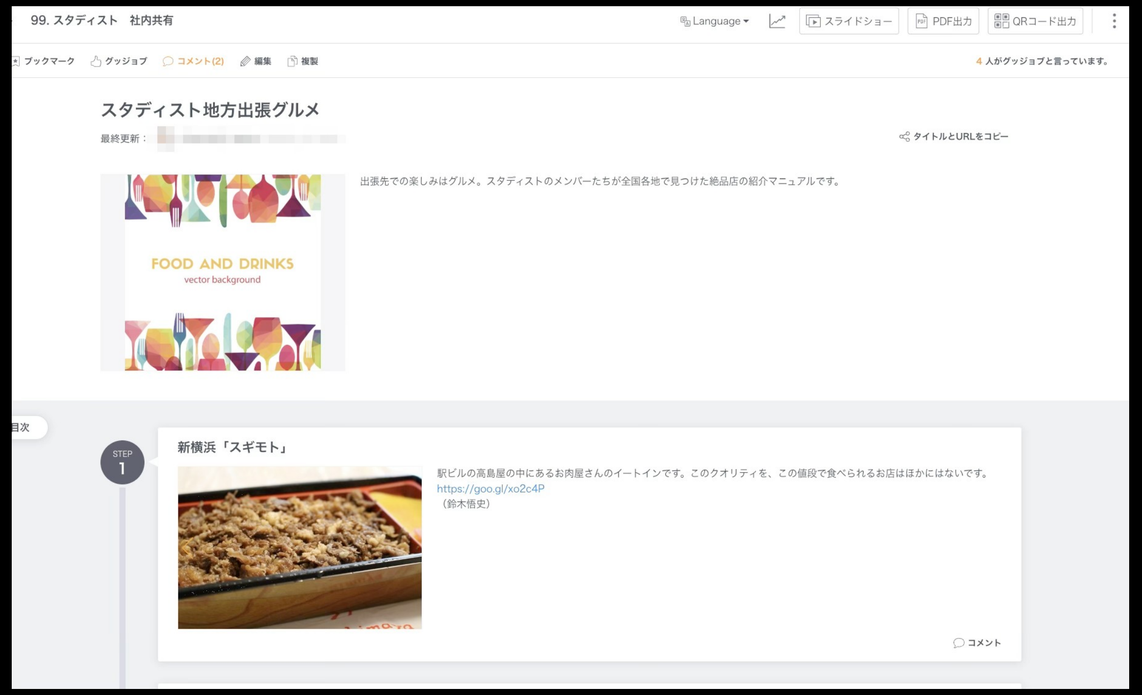
DB のトランザクション整合性を有効にできるオプション。

```
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
```

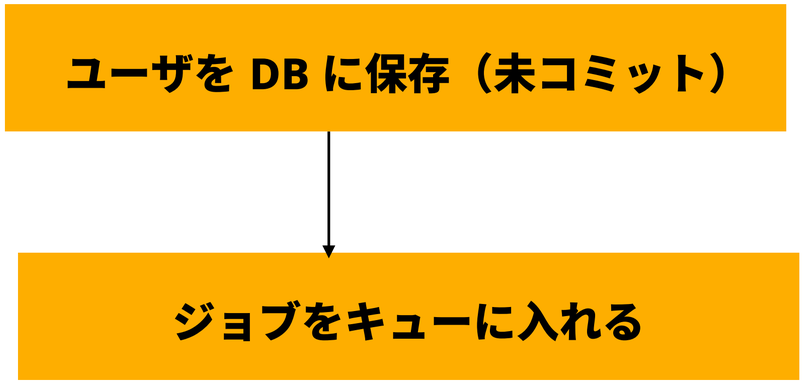
トランザクション内部に書かれたジョブ問題 1

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
```

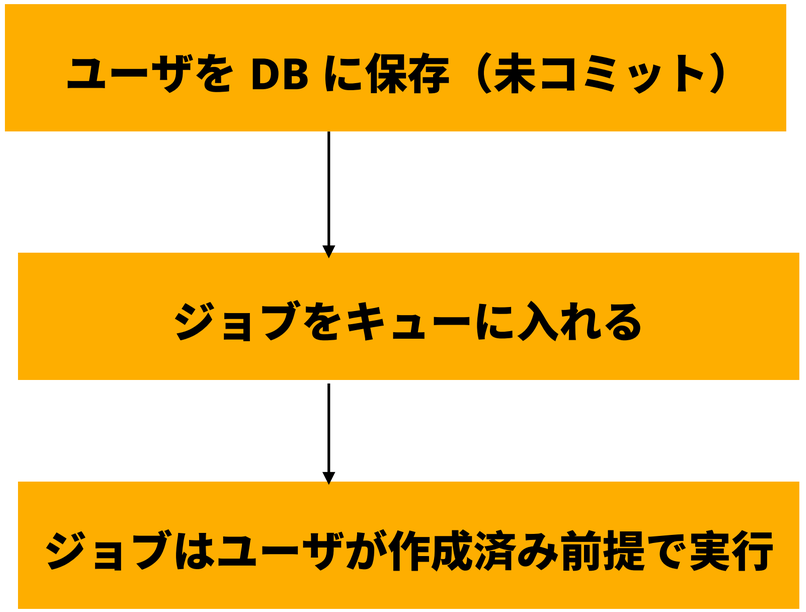
トランザクション内部に書かれたジョブ問題 1

```
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
```



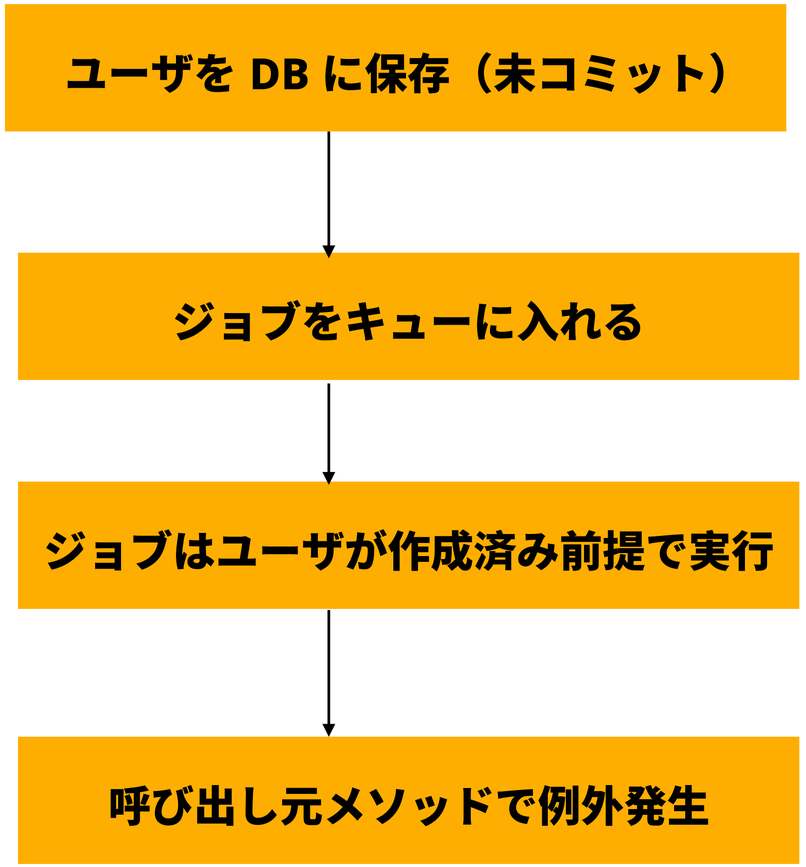
トランザクション内部に書かれたジョブ問題 1

```
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

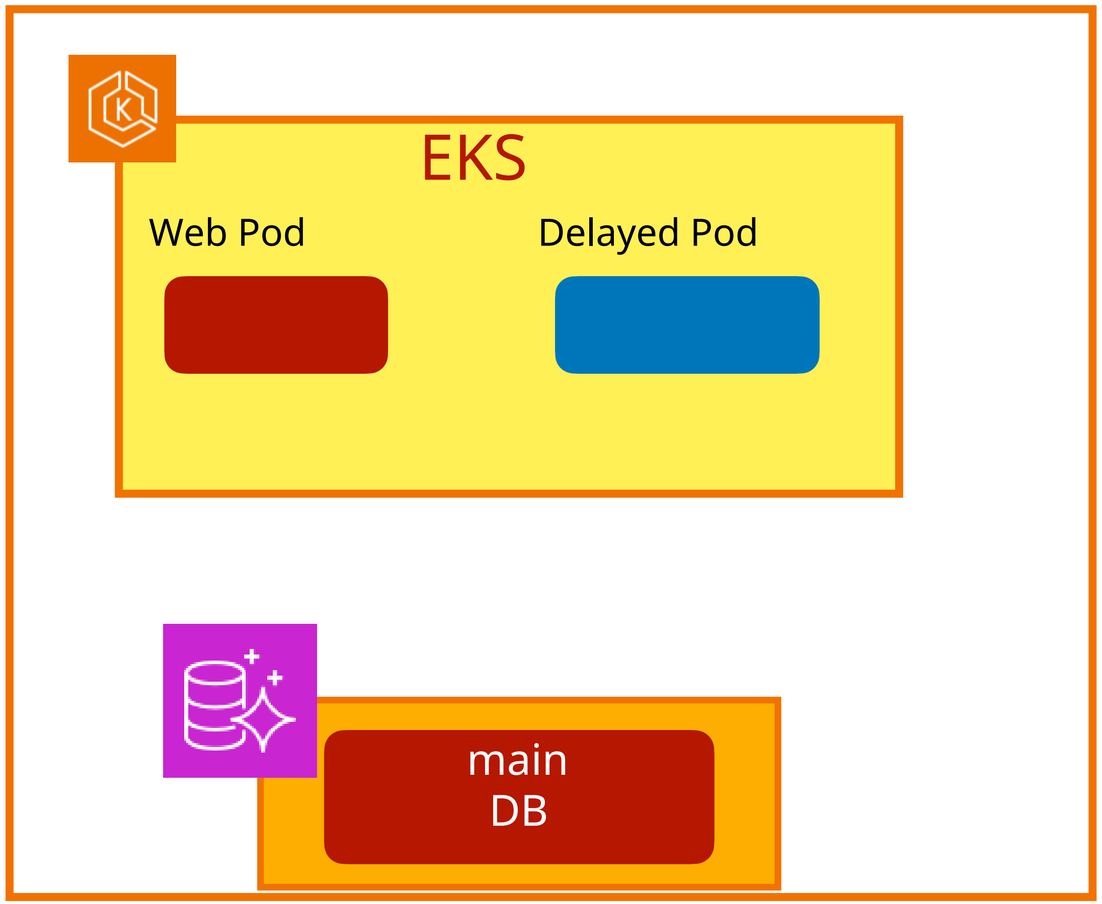
```
ActiveRecord::Base.transaction do
  user = User.create!(name: " 新しいユーザー ")
  WelcomeEmailJob.perform_later(user.id)

  raise "Something went wrong!"
end
```



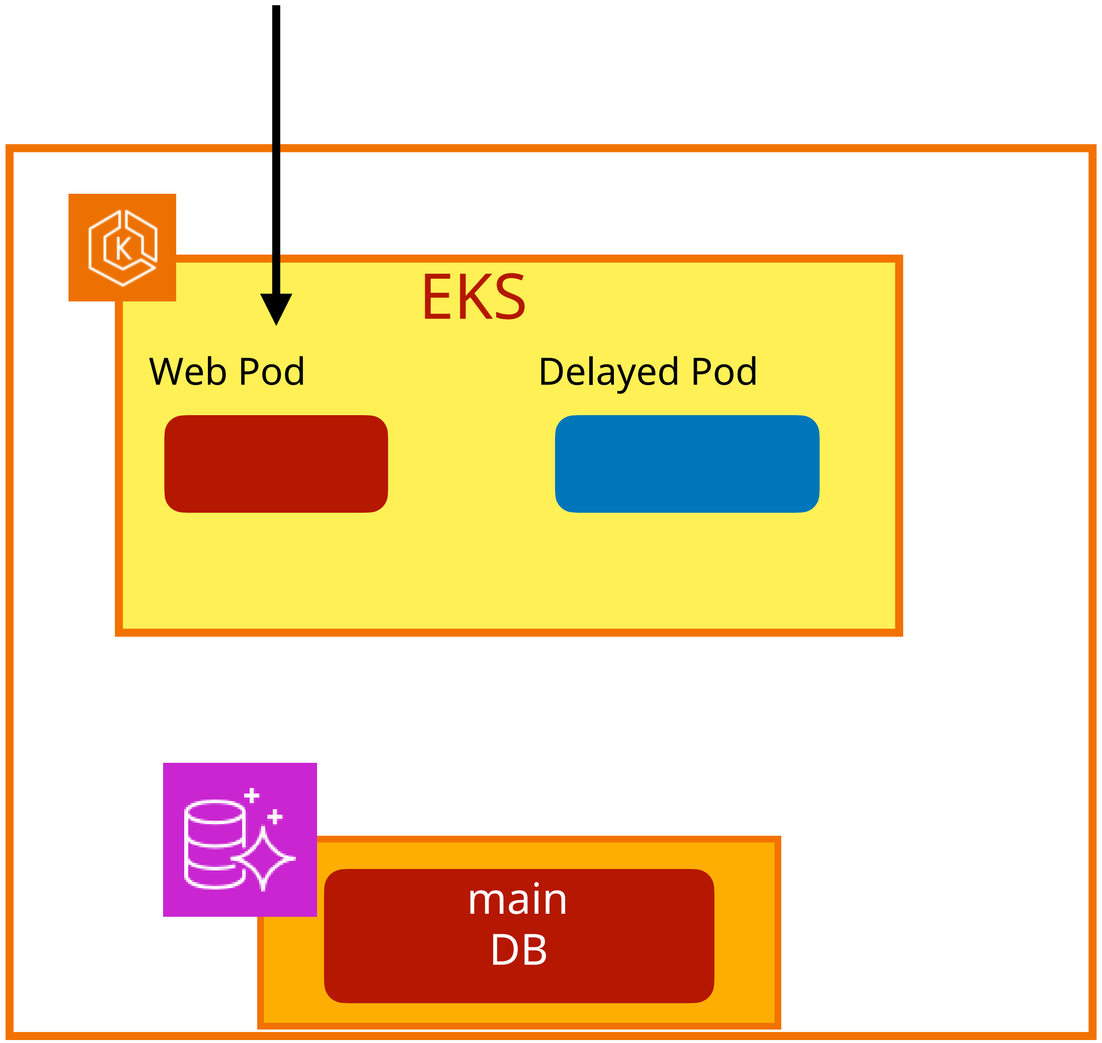
トランザクション内部に書かれたジョブ問題 1

```
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

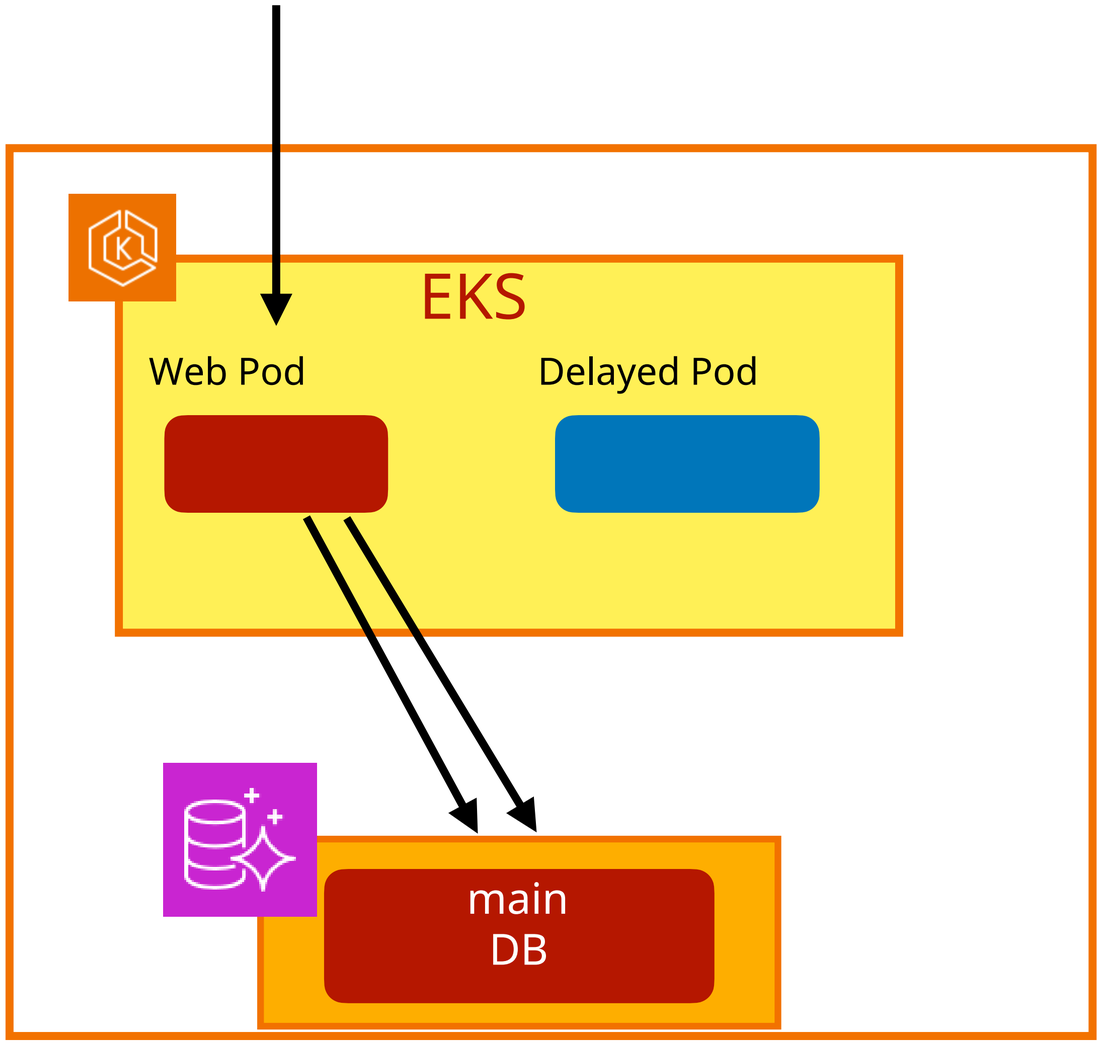


トランザクション内部に書かれたジョブ問題 2



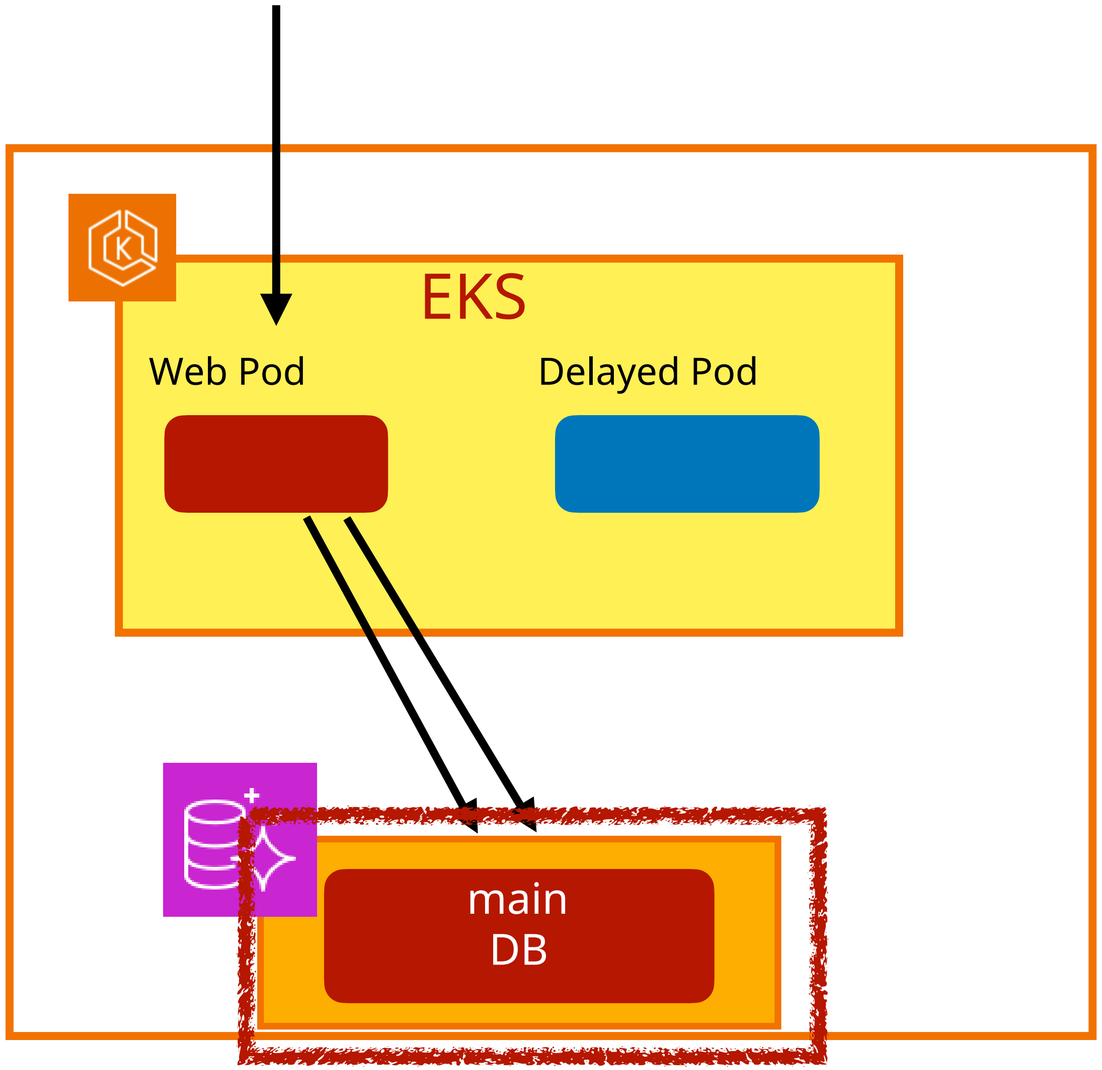
1. Web にアクセスがある

トランザクション内部に書かれたジョブ問題 2



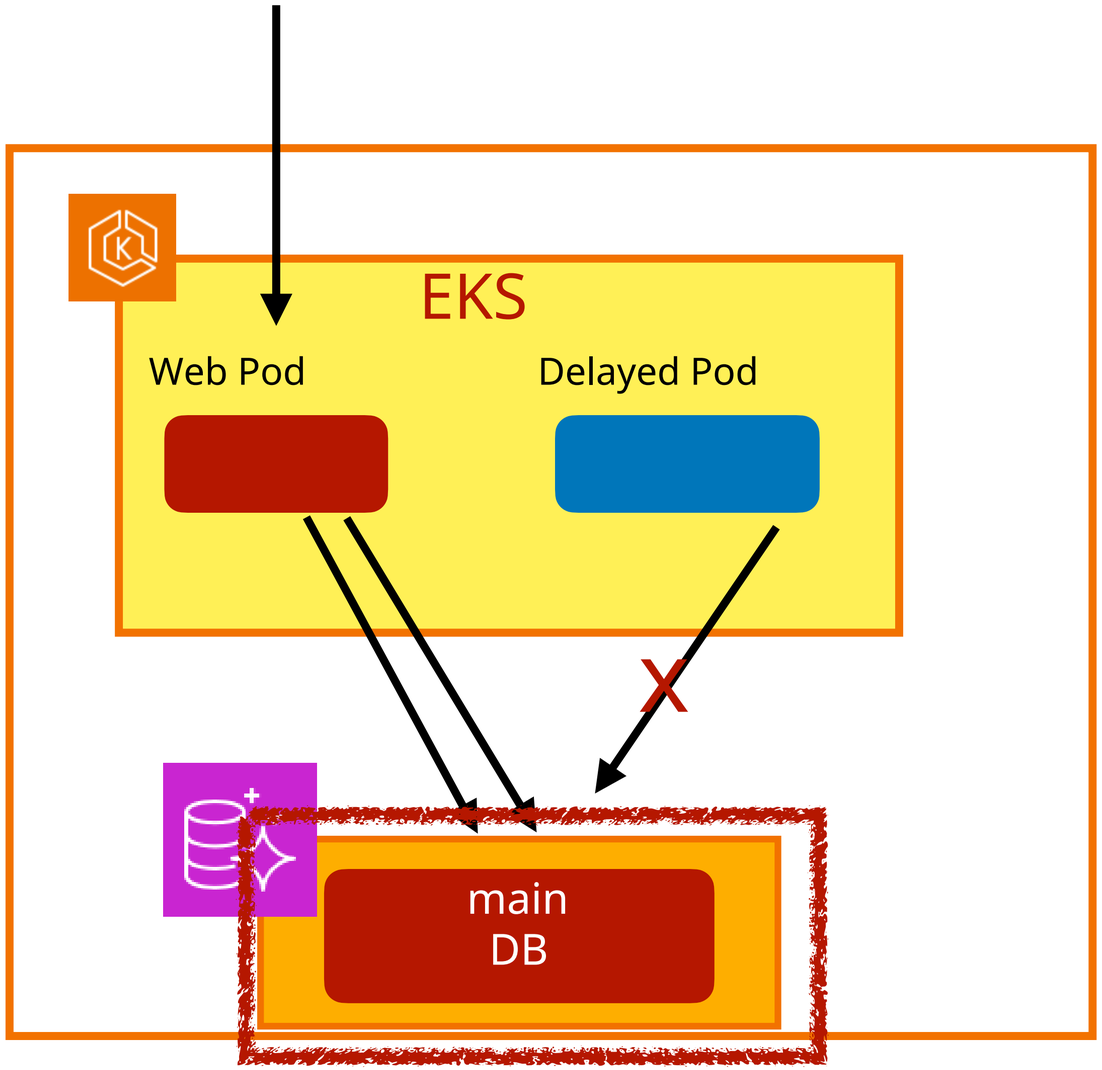
1. Web にアクセスがある
2. トランザクション開始＆ユーザ作成

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）

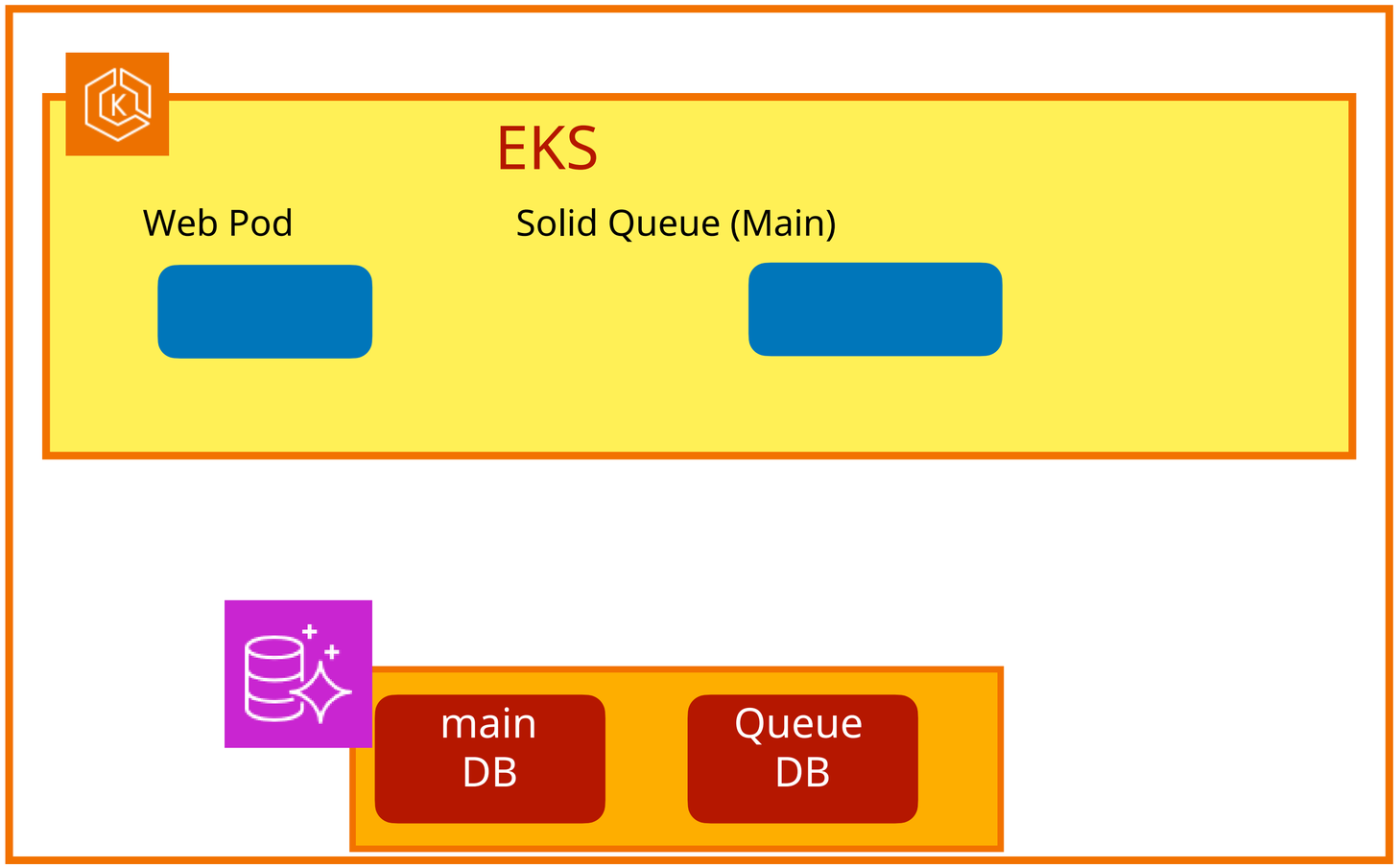
トランザクション内部に書かれたジョブ問題 2



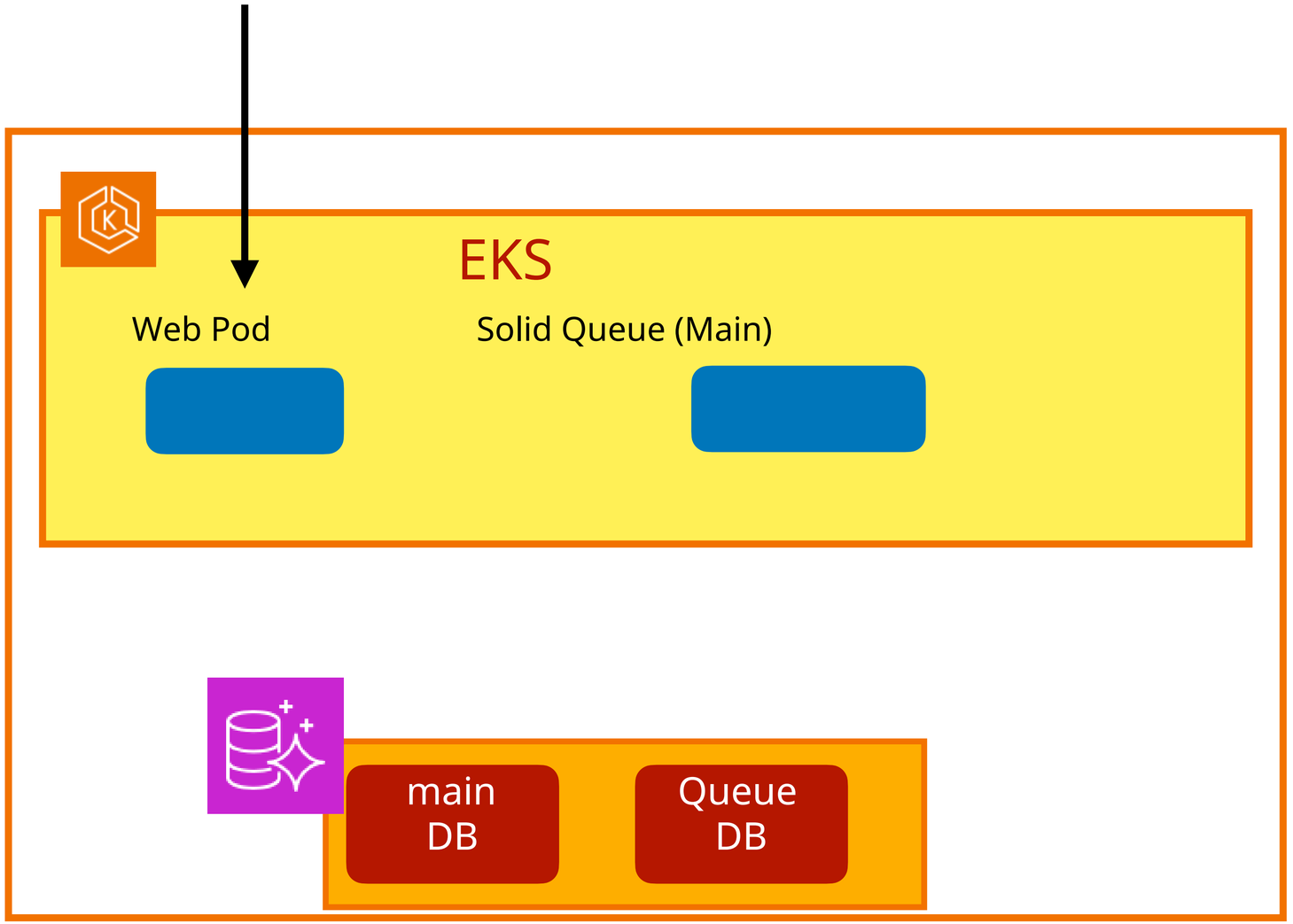
1. Web にアクセスがある
2. トランザクション開始＆ユーザ作成
3. ジョブをエンキュー（未コミット）

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

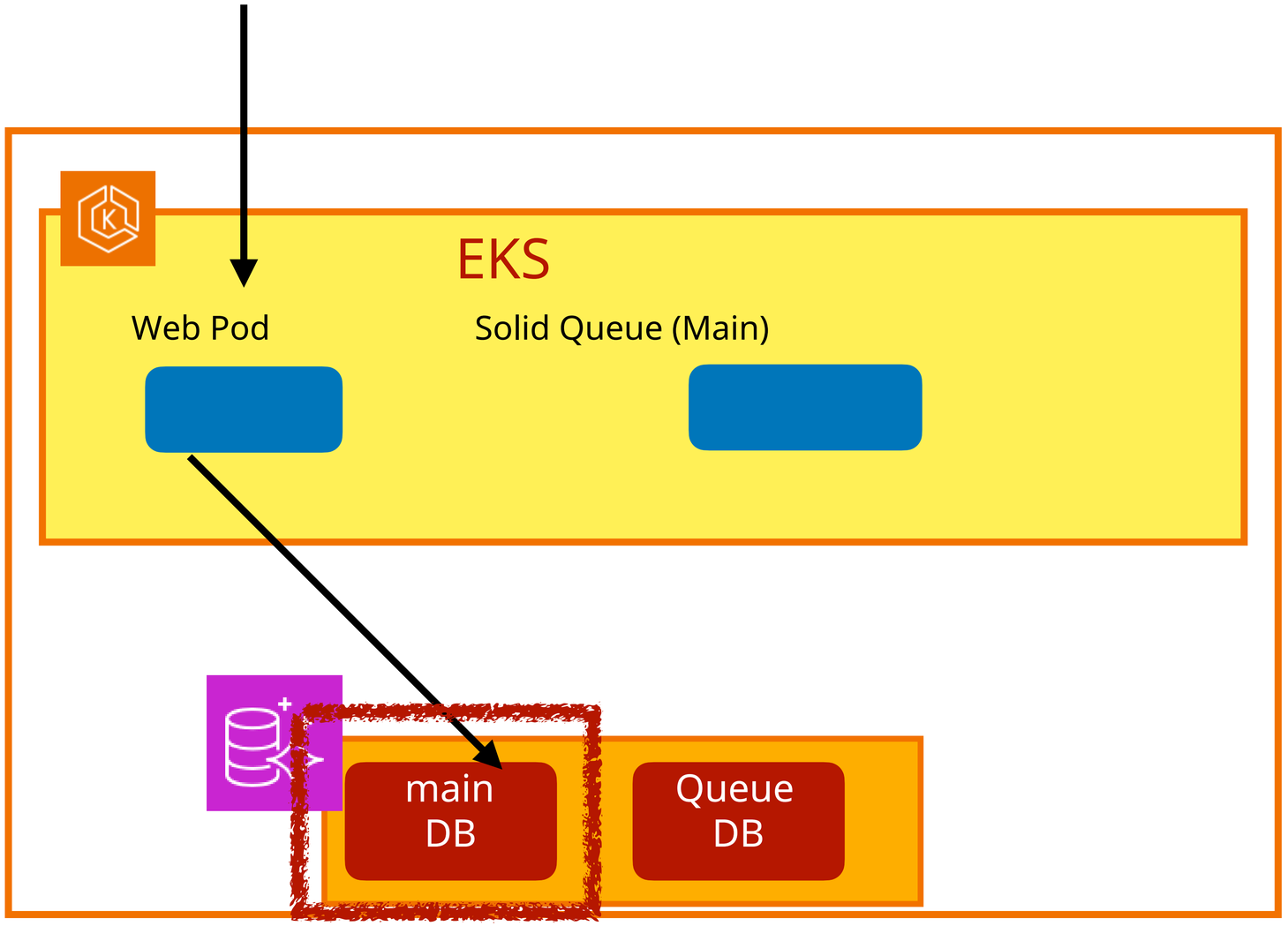


トランザクション内部に書かれたジョブ問題 2



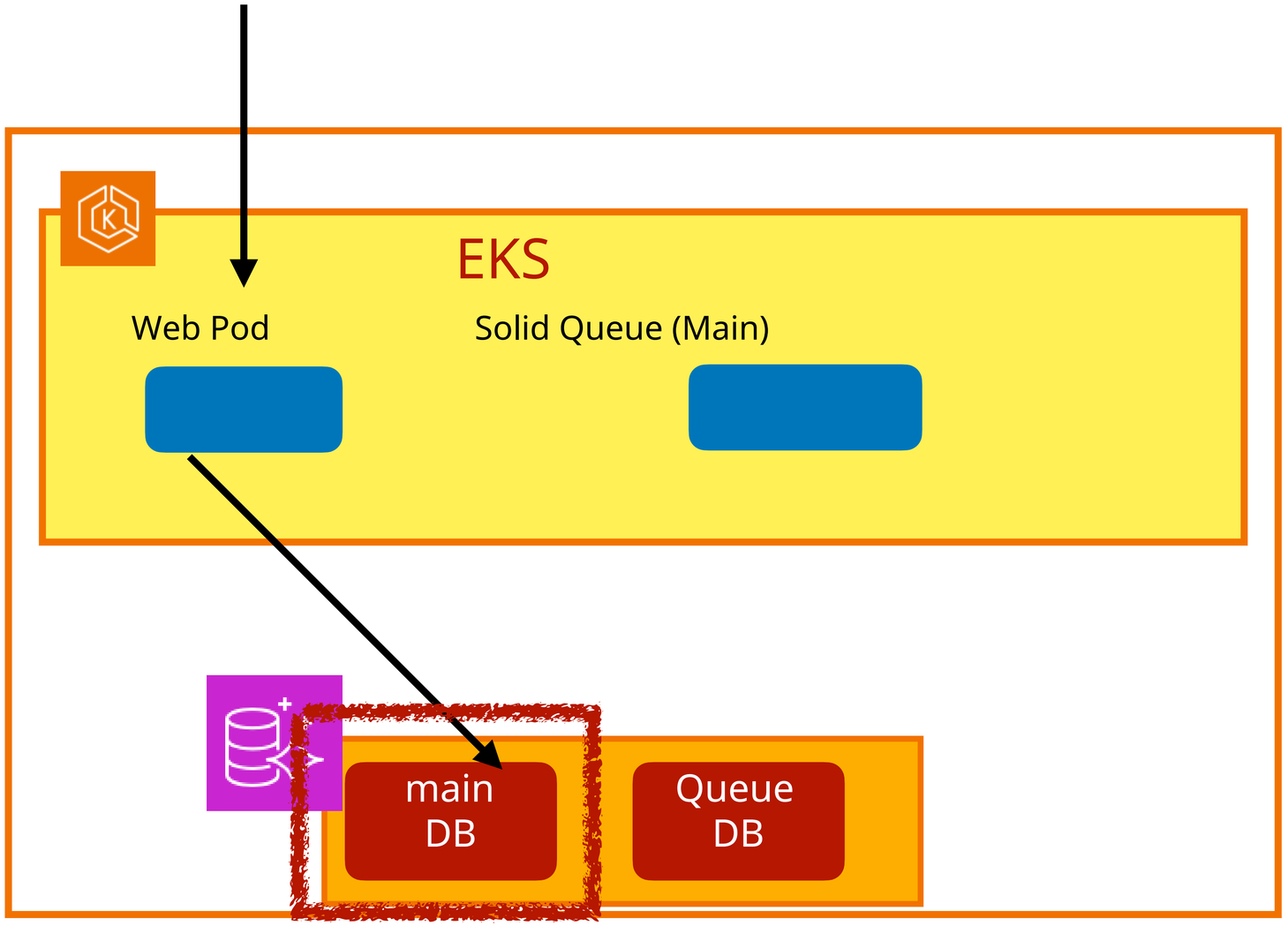
1. Web にアクセスがある

トランザクション内部に書かれたジョブ問題 2



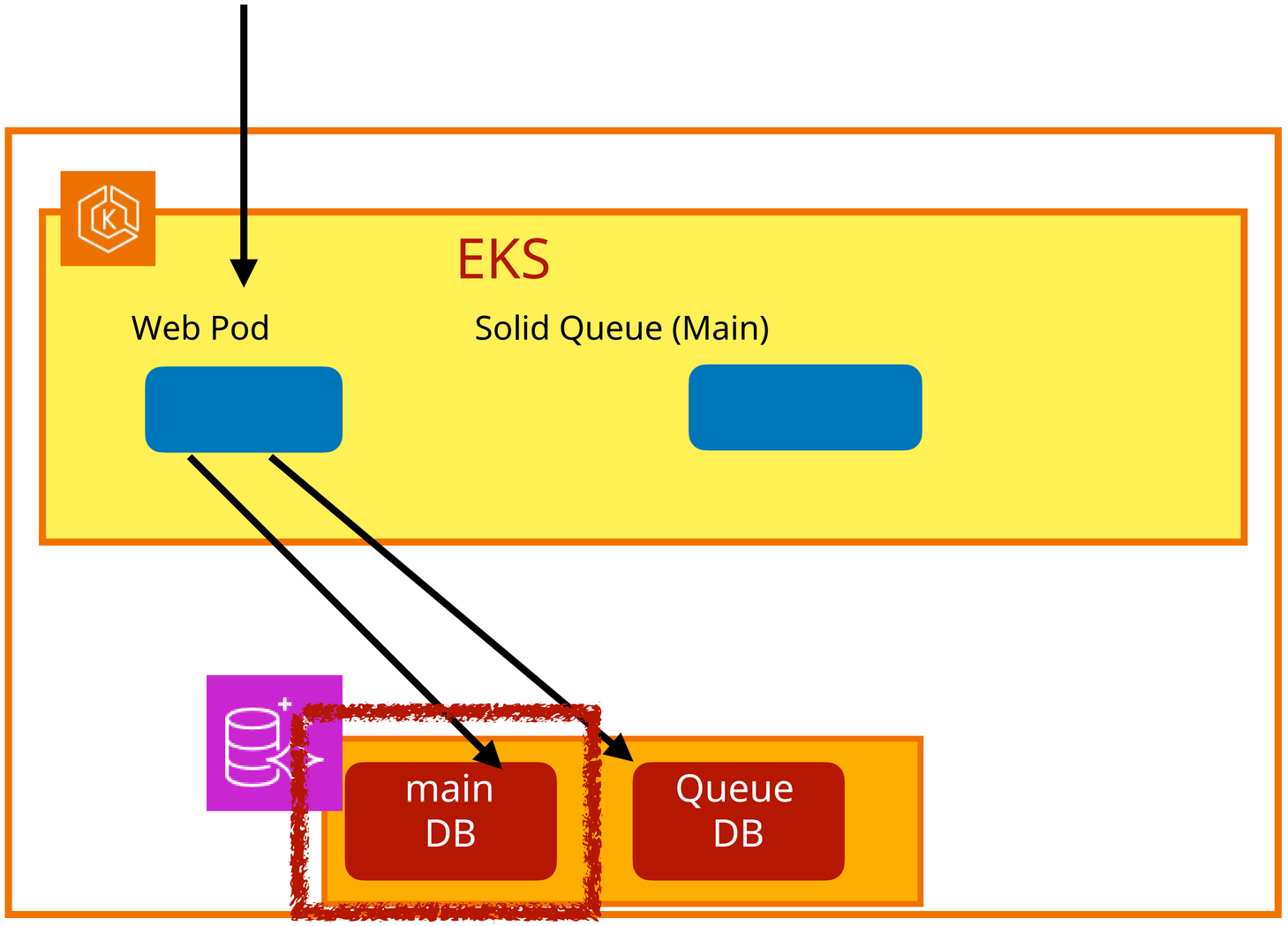
1. Web にアクセスがある
2. トランザクション開始

トランザクション内部に書かれたジョブ問題 2



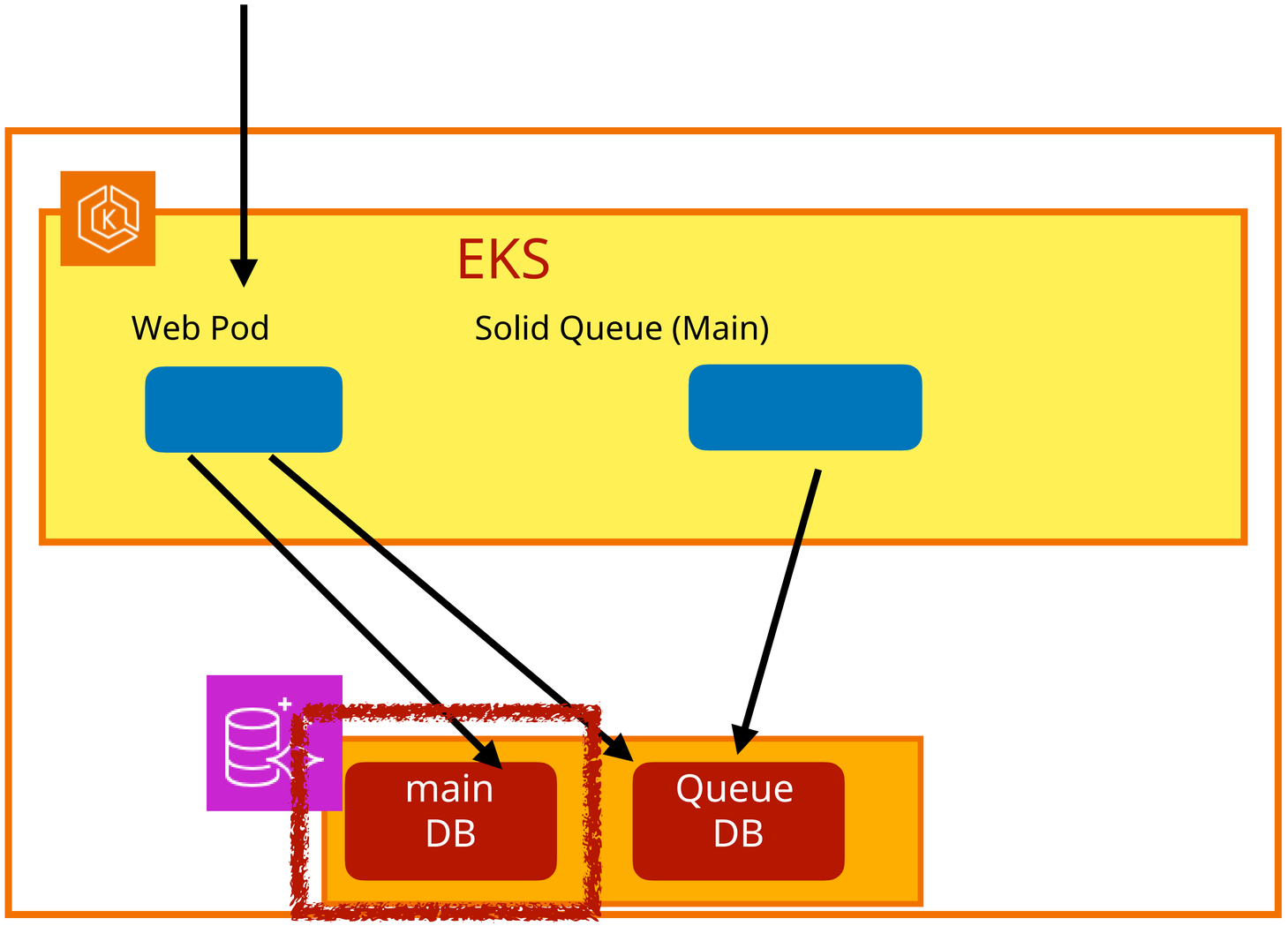
1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）

トランザクション内部に書かれたジョブ問題 2



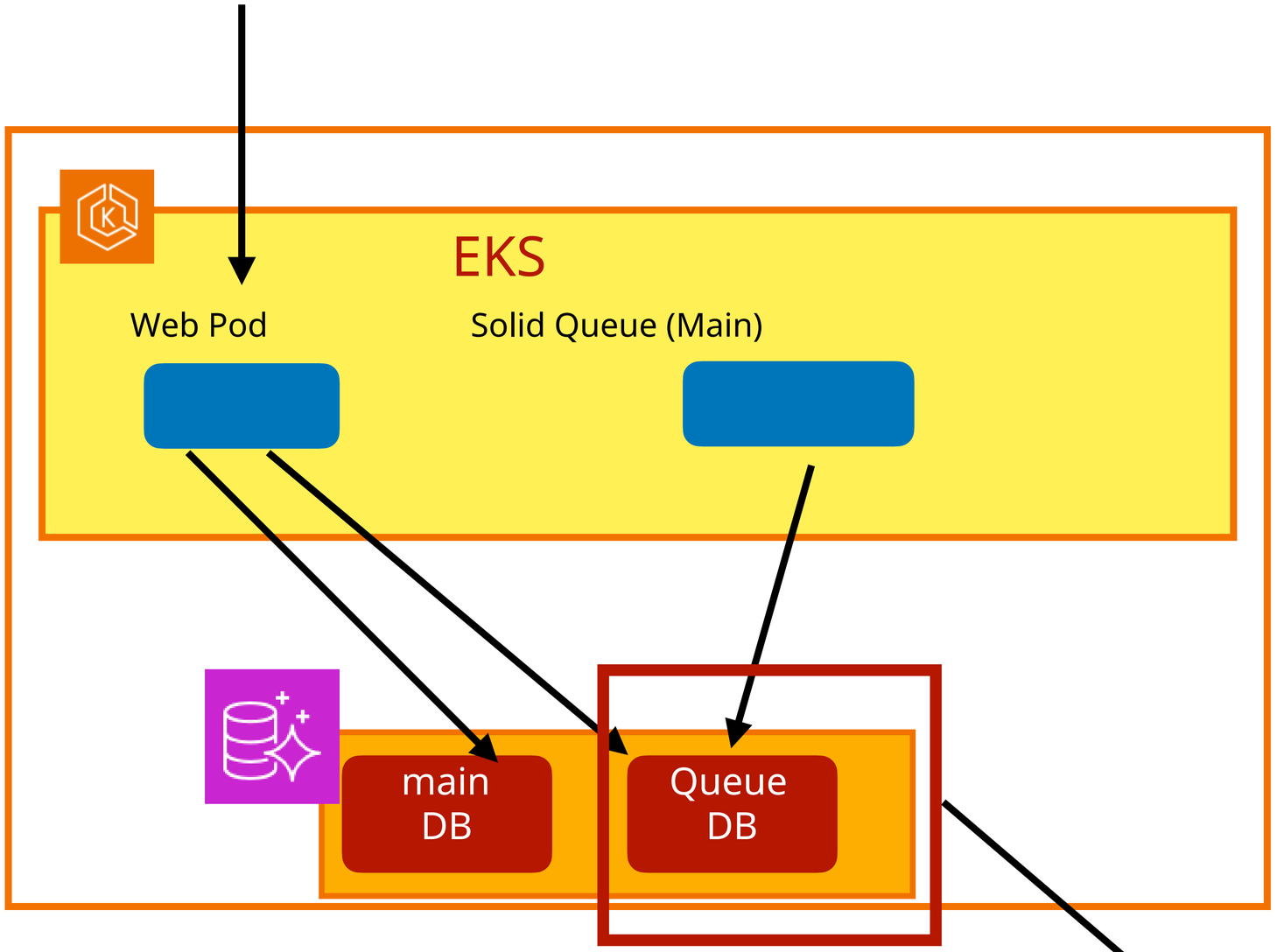
1. Web にアクセスがある
2. トランザクション開始
3. 処理の DB 保存処理（未コミット）
4. キュー DB にキューがはいる

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

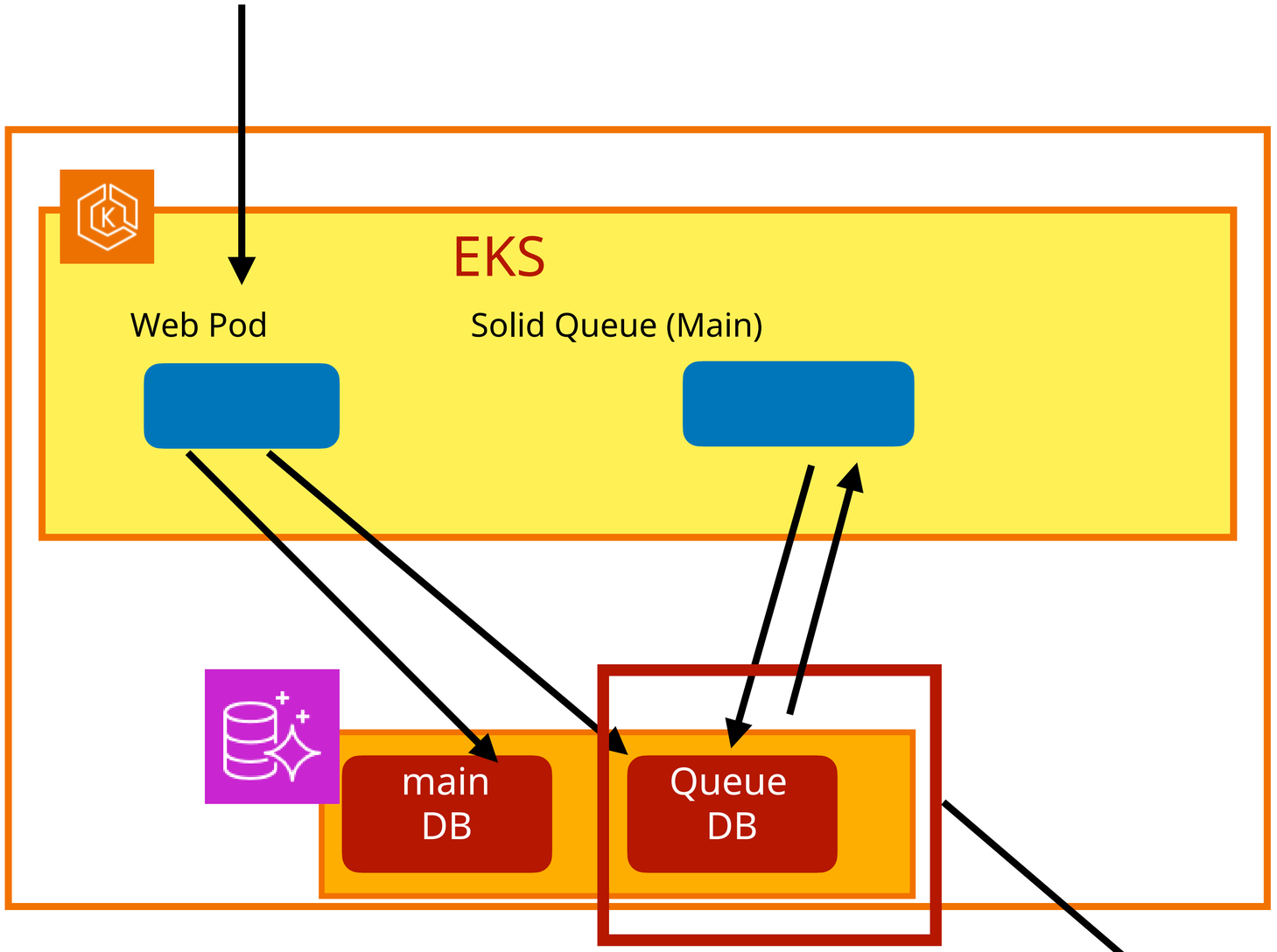
トランザクション内部に書かれたジョブ問題 2



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

enqueueaftertransactioncommit 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

enqueueaftertransactioncommit 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

enqueueaftertransactioncommit を使えるようにする！

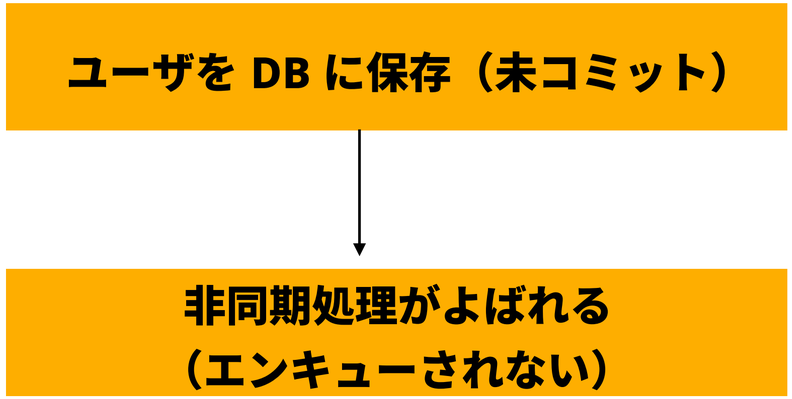
ユーザを DB に保存（未コミット）

```
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

enqueueaftertransactioncommit を使えるようにする！

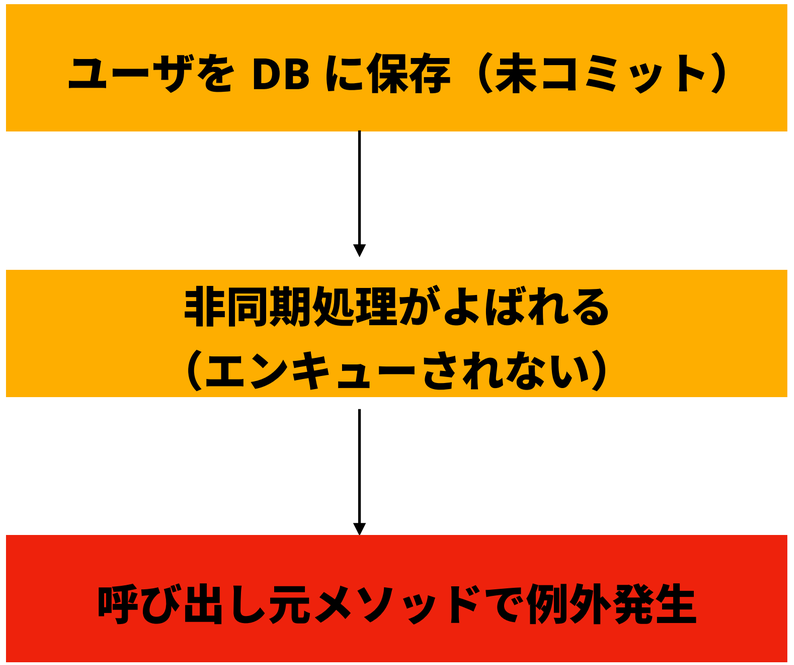
```
ActiveRecord::Base.transaction do

  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

enqueueaftertransactioncommit を使えるようにする！

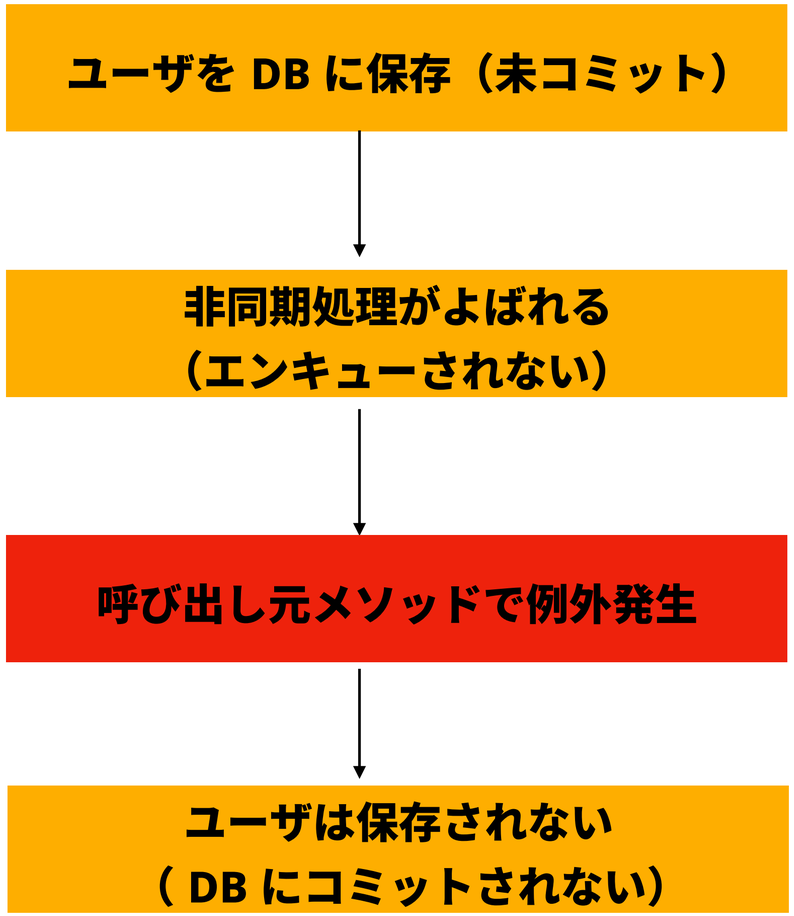
```
ActiveRecord::Base.transaction do
  user = User.create!(name: " 新しいユーザー ")
  WelcomeEmailJob.perform_later(user.id)
  raise "Something went wrong!"
end
```



解決法トランザクション内部に書かれたジョブ問題

enqueueaftertransactioncommit を使えるようにする！

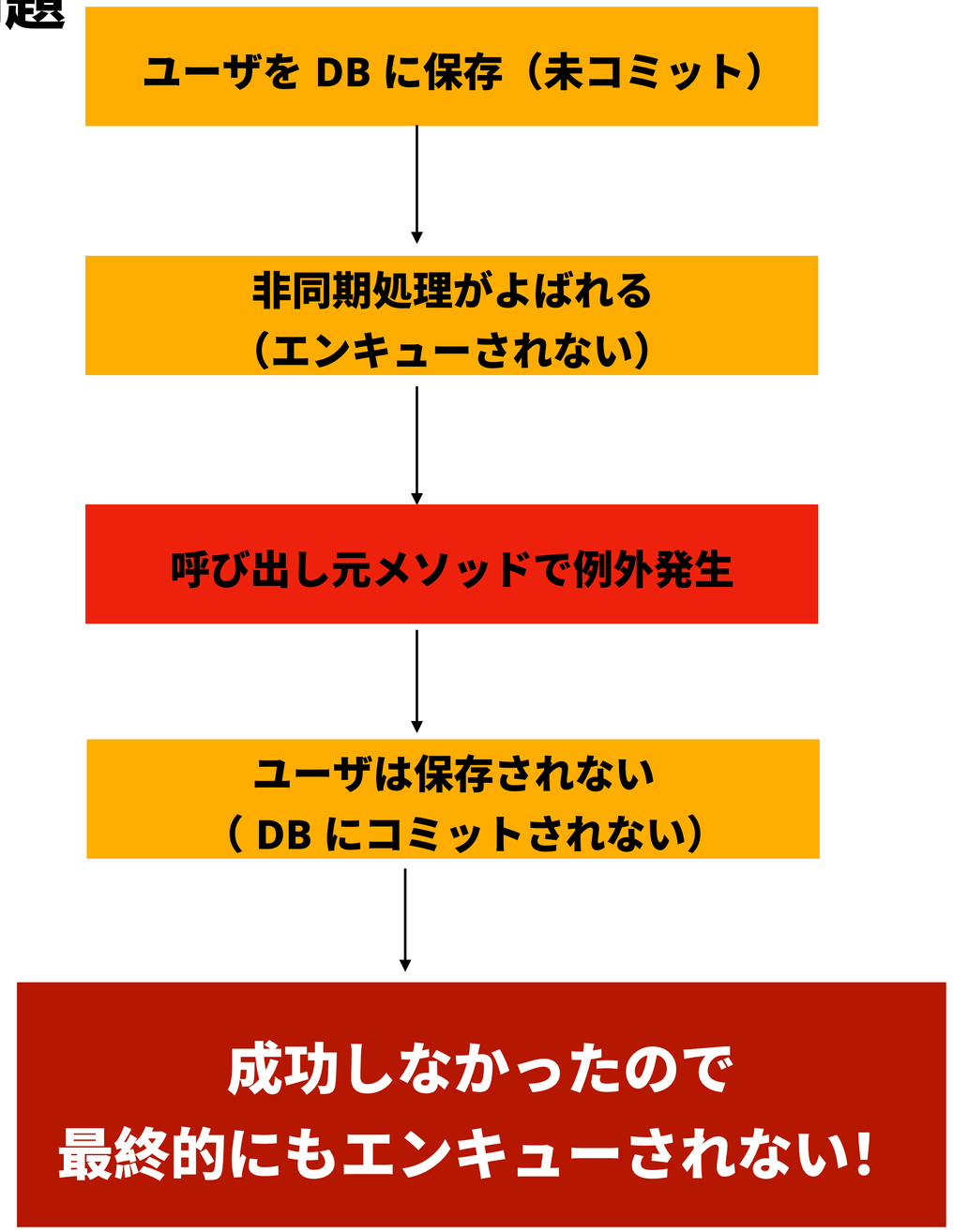
```
ActiveRecord::Base.transaction do
  user = User.create!(name: " 新しいユーザー ")
  WelcomeEmailJob.perform_later(user.id)

  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

enqueueaftertransactioncommit 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
```



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?

beforexxx / afterxxxx 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?

beforexxx / afterxxxx 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

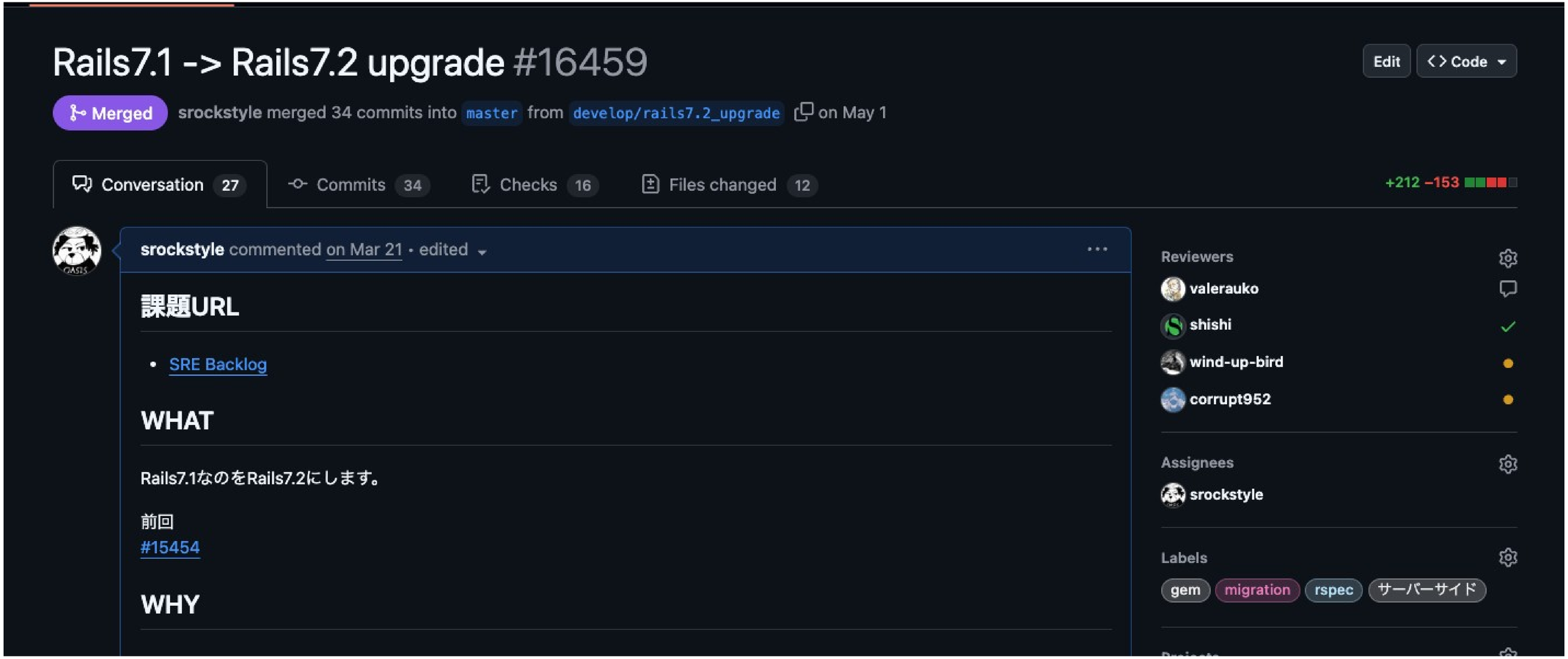
Teachme Biz にはあるのか?

beforexxx / afterxxxx 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なのを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

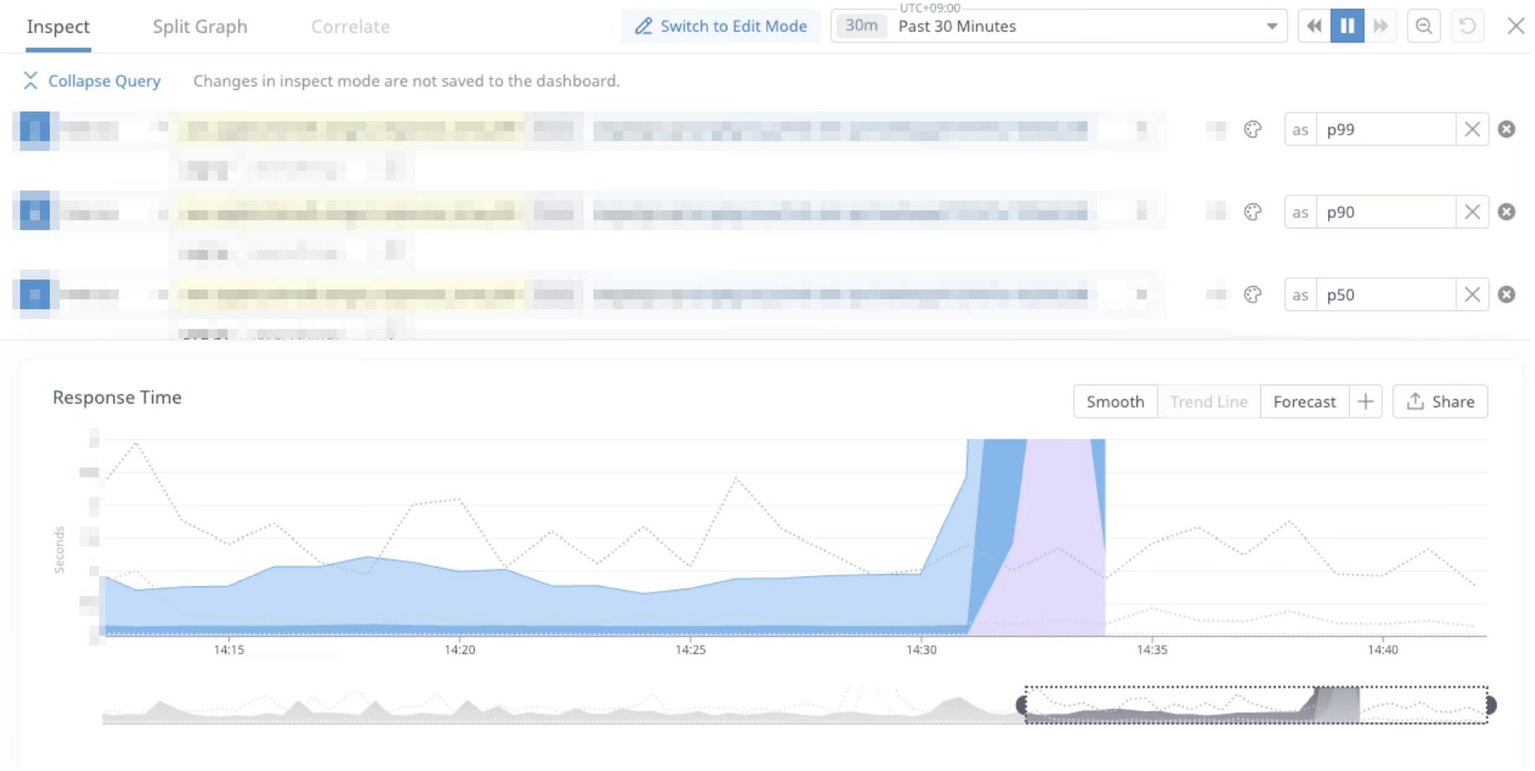
実行して欲しい場合はどうするの?

設計を見直して

実行して欲しい場合はどうするの？

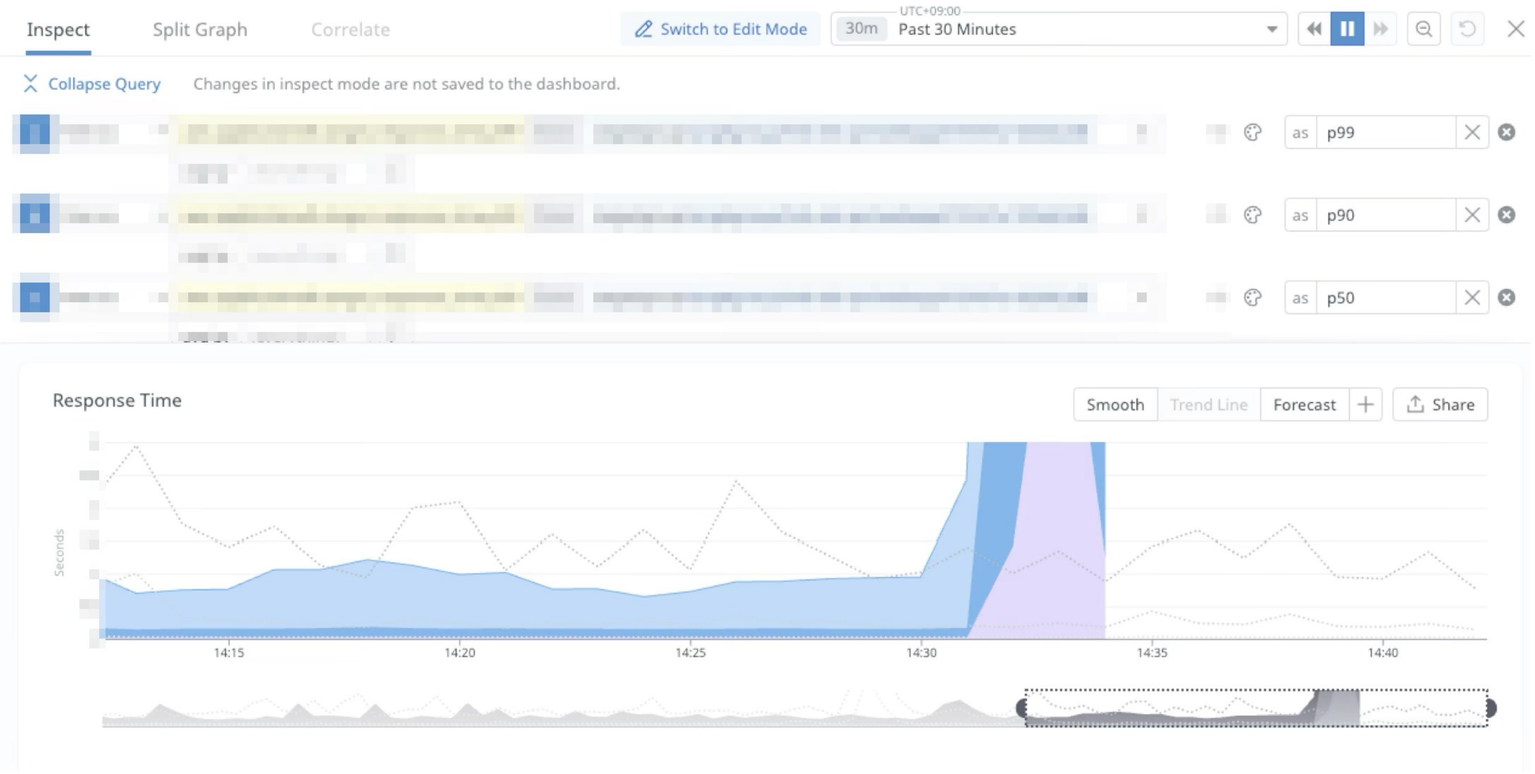
Solid Queue 速すぎて過負荷になった問題

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

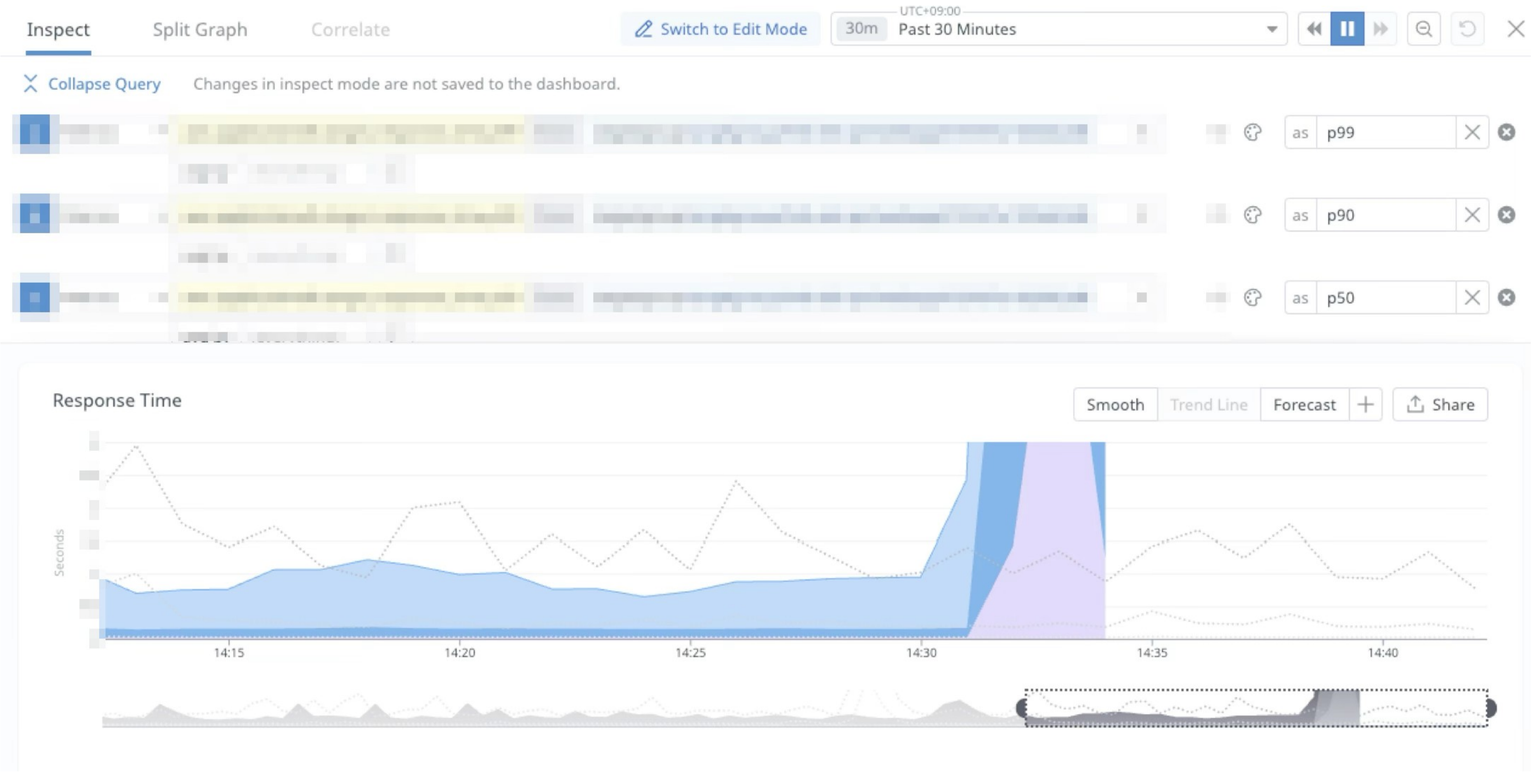
SolidQueue リリースから数日後



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

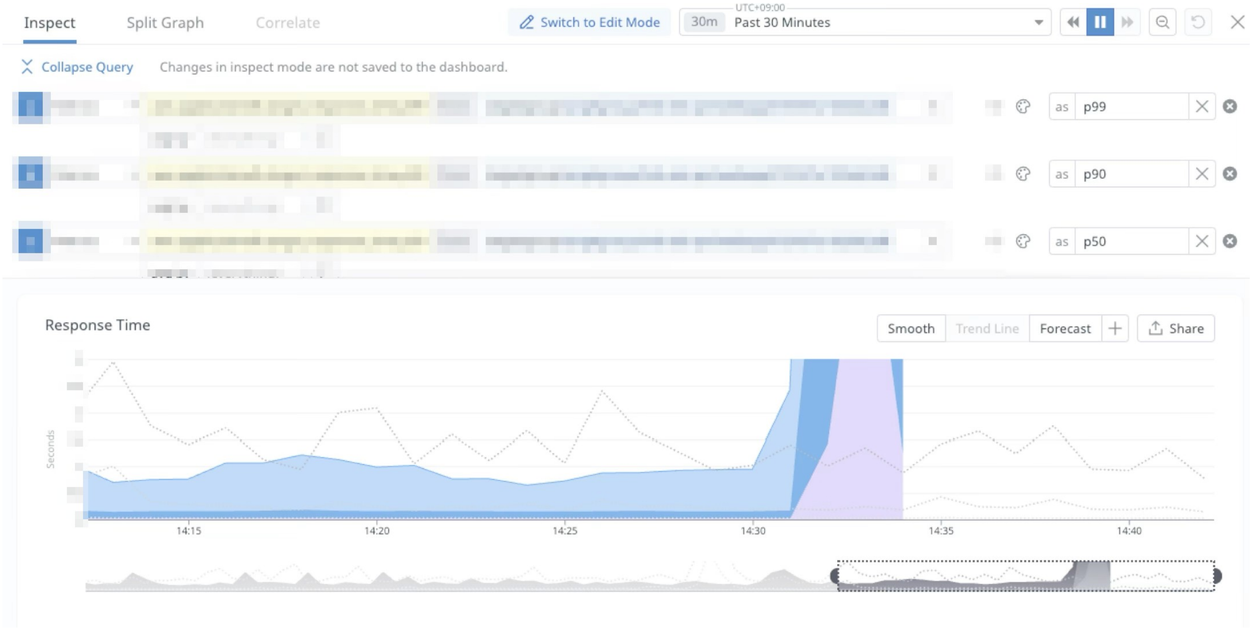


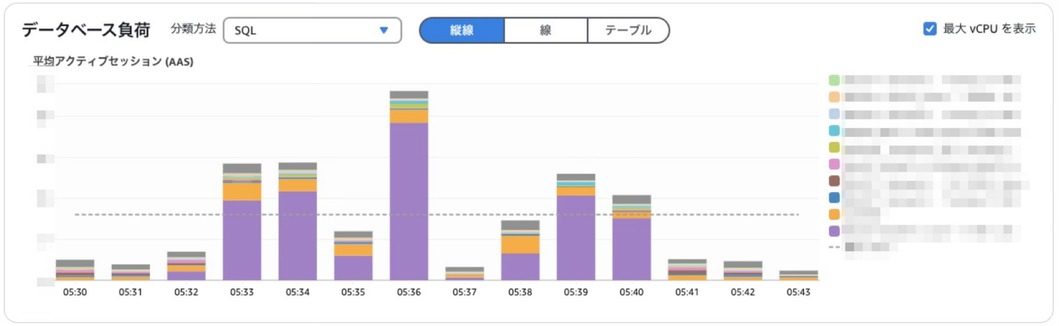
重くない?

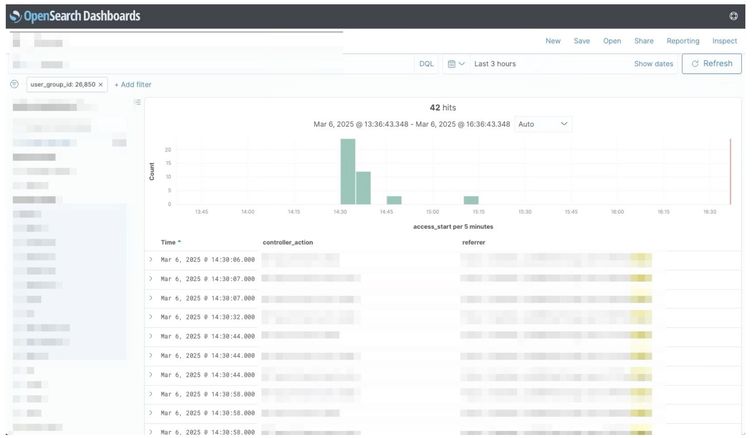
というかアクセスすると重くない?

Solid Queue 速すぎて過負荷になった問題

原因は DB の高負荷

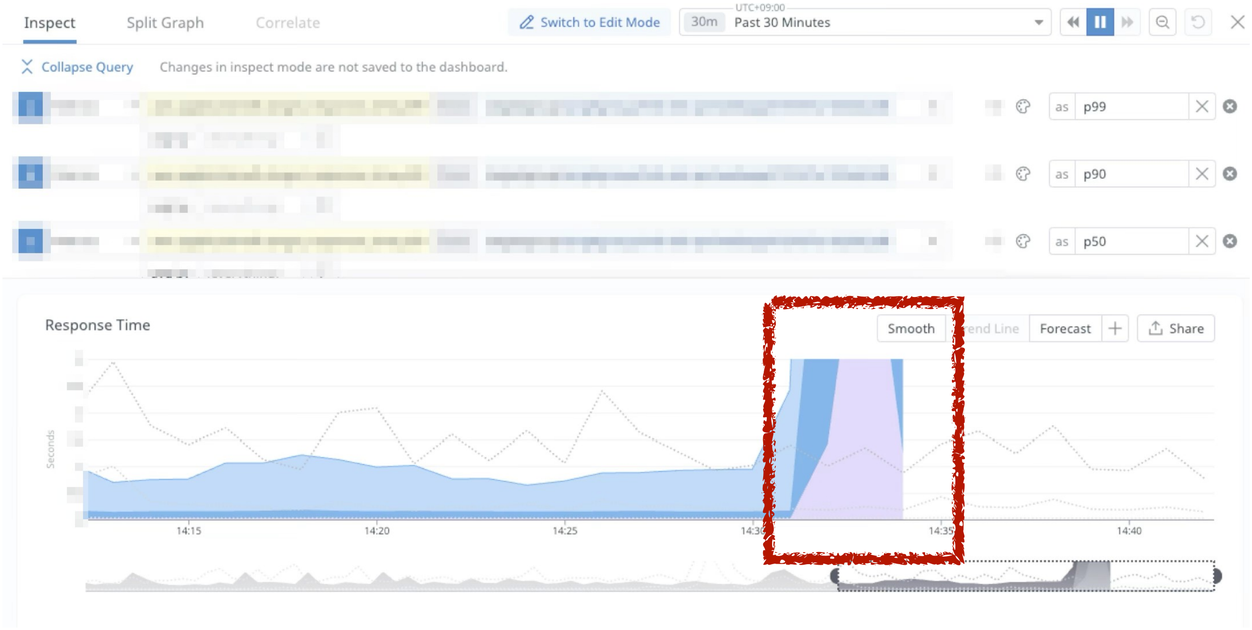


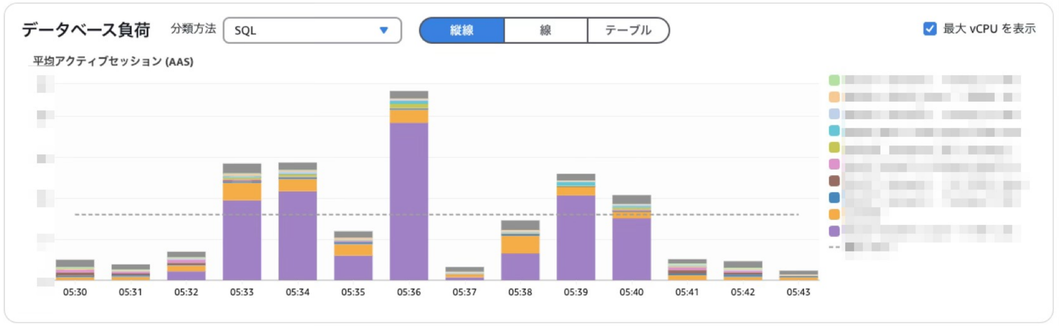


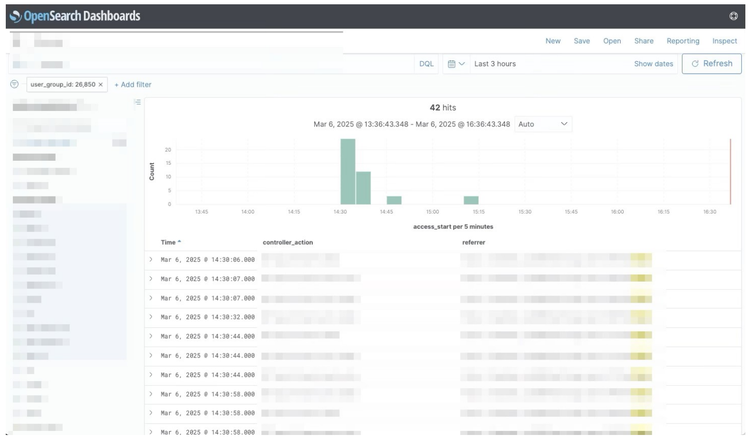


Solid Queue 速すぎて過負荷になった問題

原因は DB の高負荷

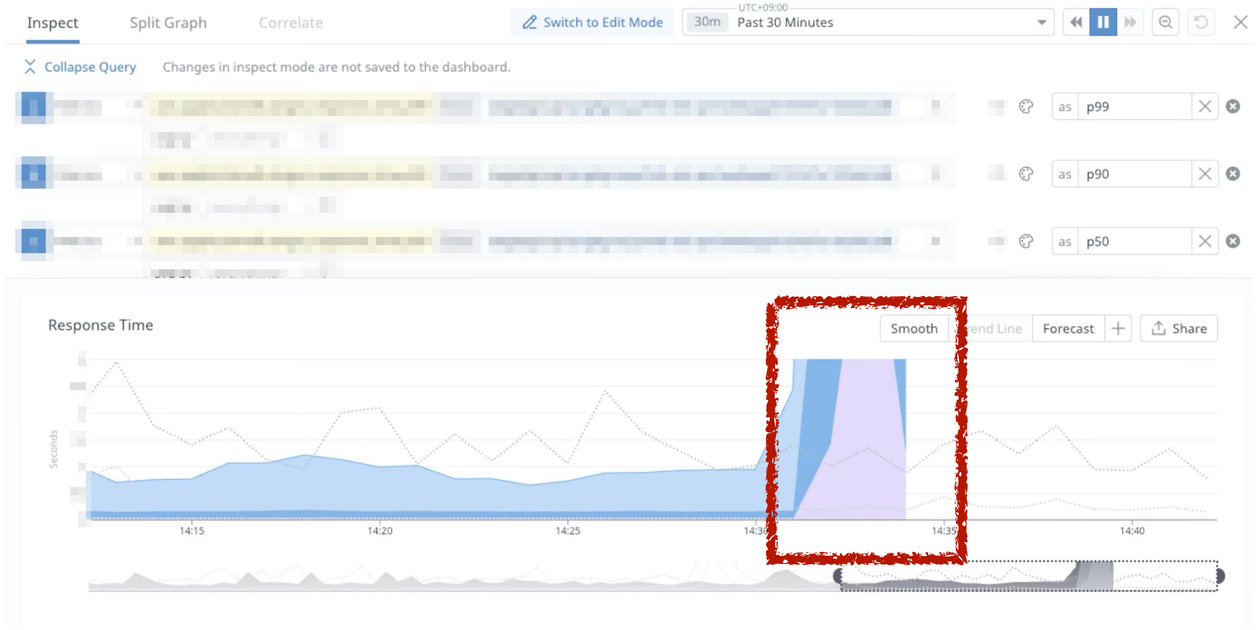


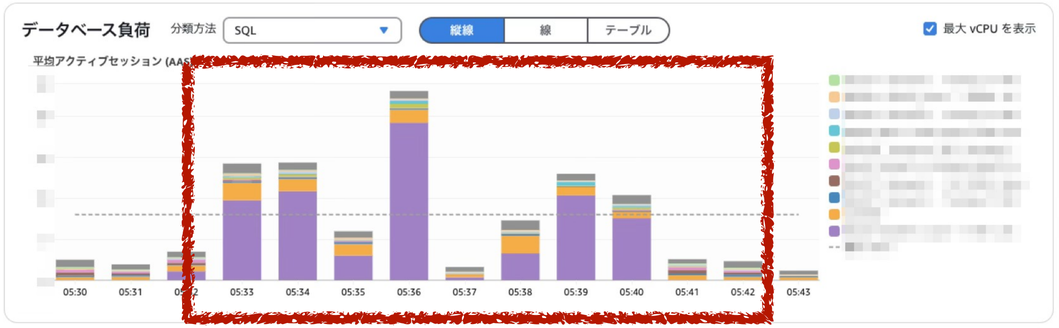


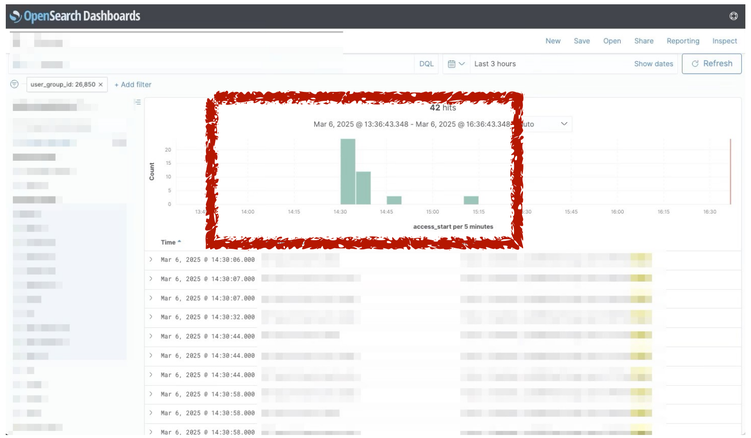


Solid Queue 速すぎて過負荷になった問題

原因は DB の高負荷

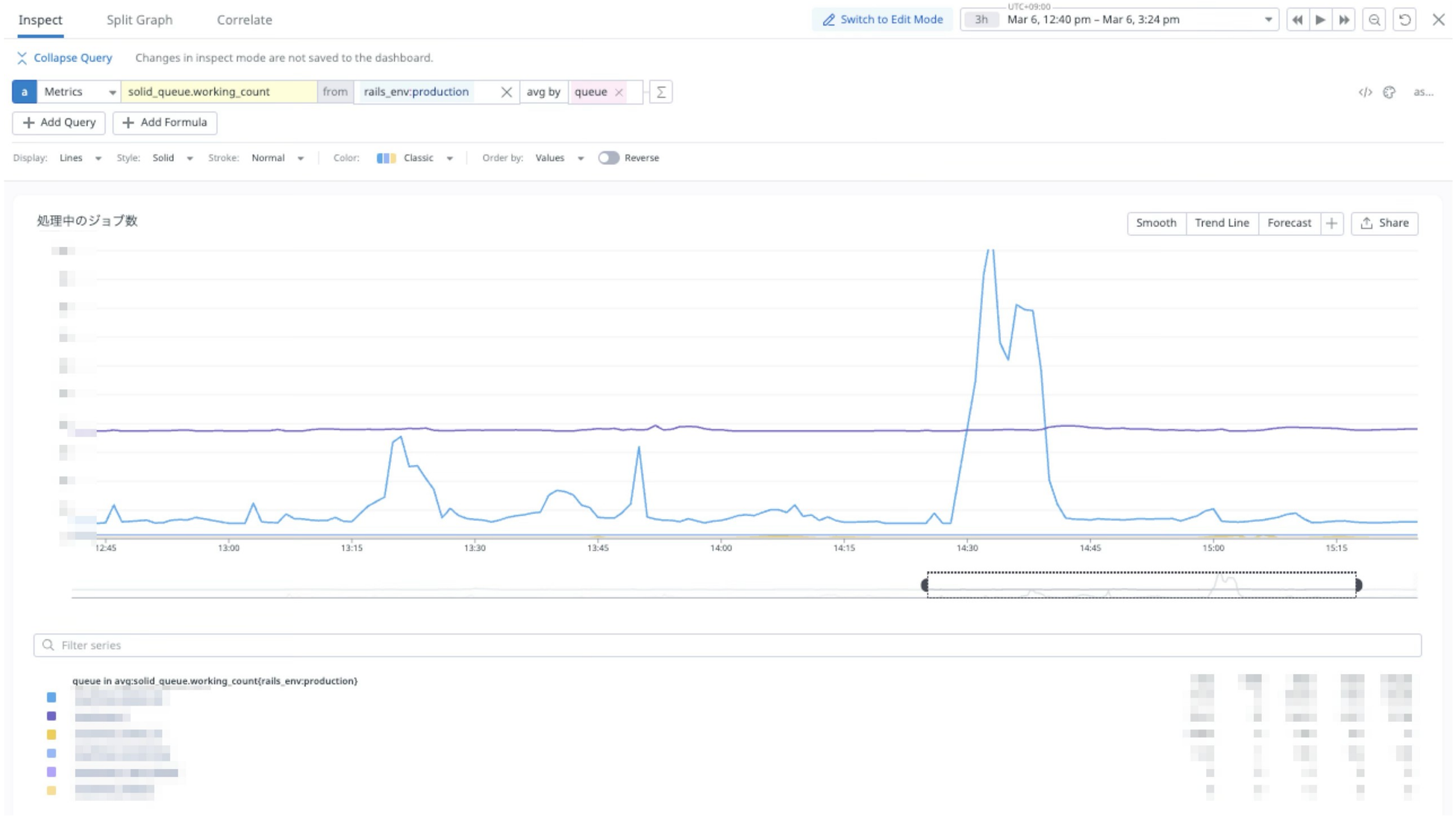






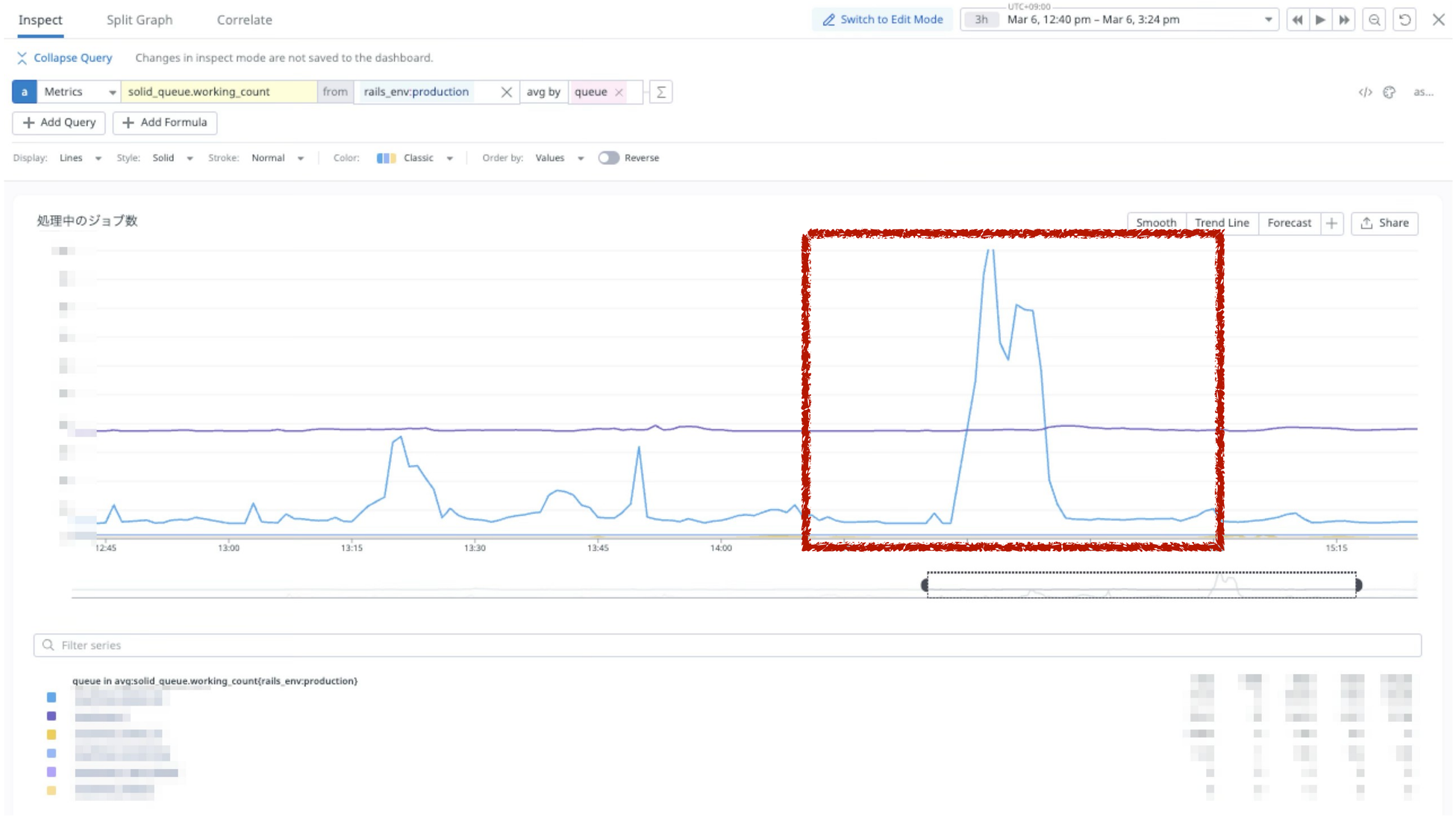
Solid Queue 速すぎて過負荷になった問題

原因は ....

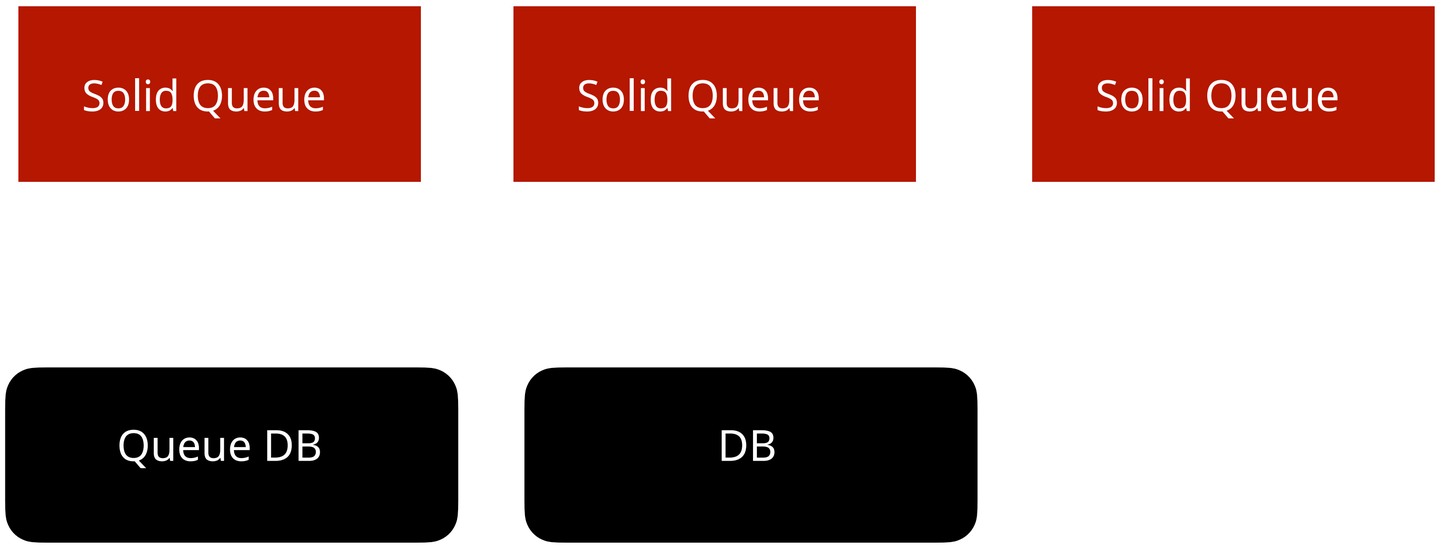


Solid Queue 速すぎて過負荷になった問題

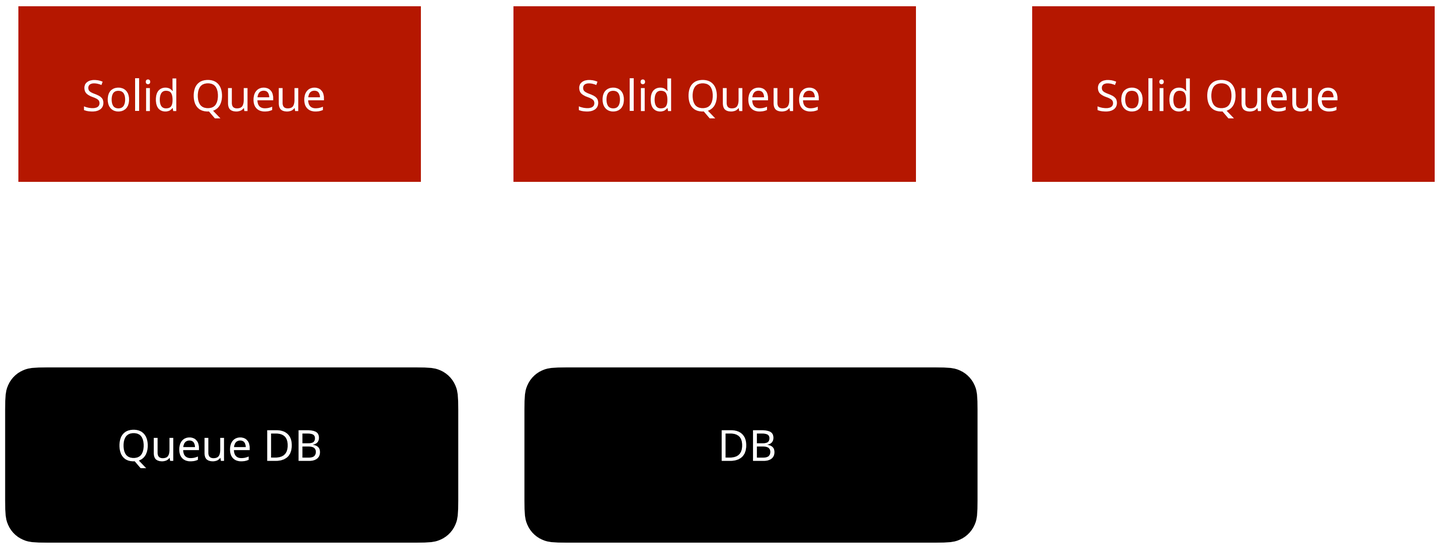
原因は ....



Solid Queue 速すぎて過負荷になった問題



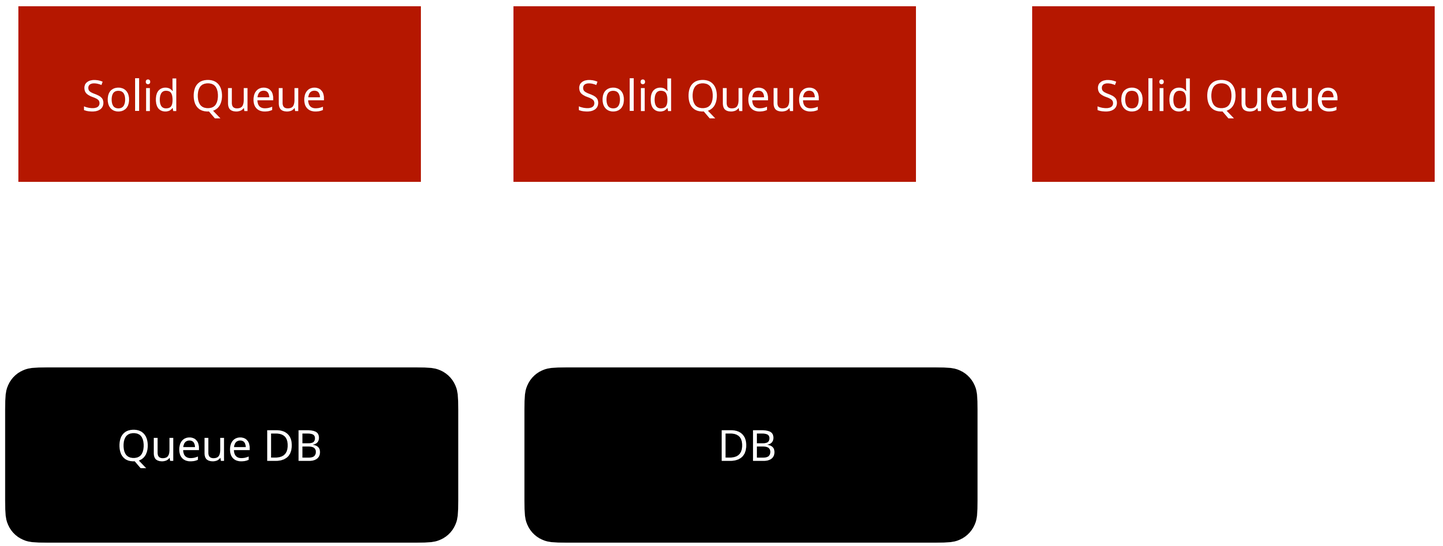
Solid Queue 速すぎて過負荷になった問題
原因は重量級ジョブが大量に並列実行されるようになったこと



Solid Queue 速すぎて過負荷になった問題

原因は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

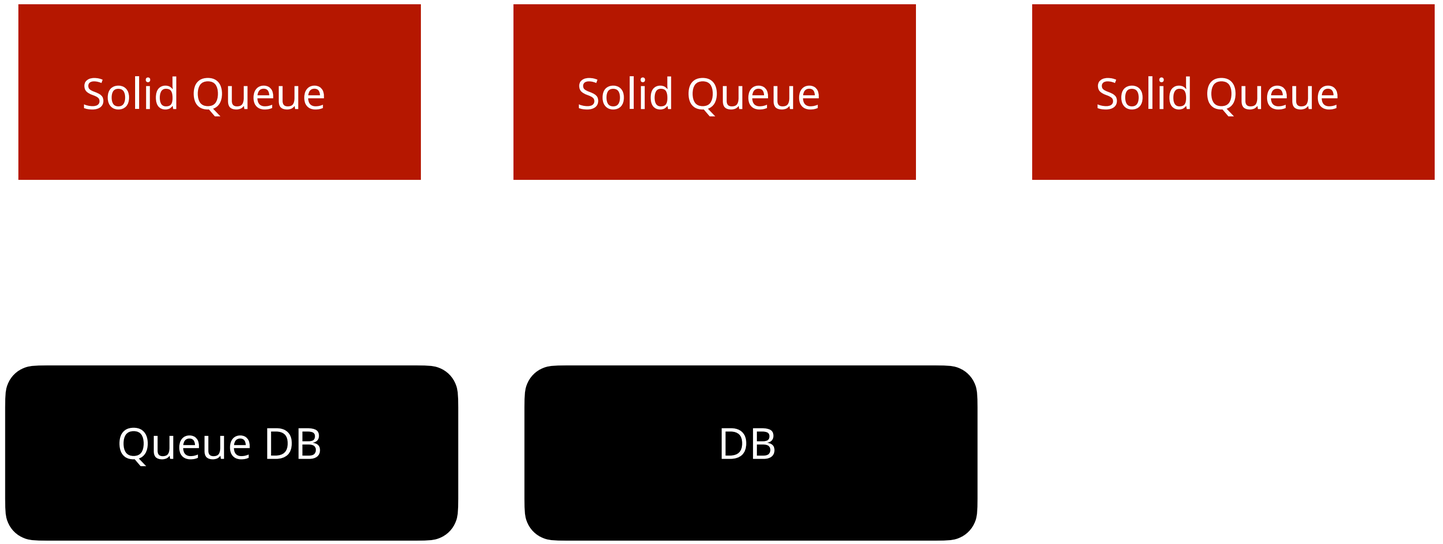


Solid Queue 速すぎて過負荷になった問題

原因は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!

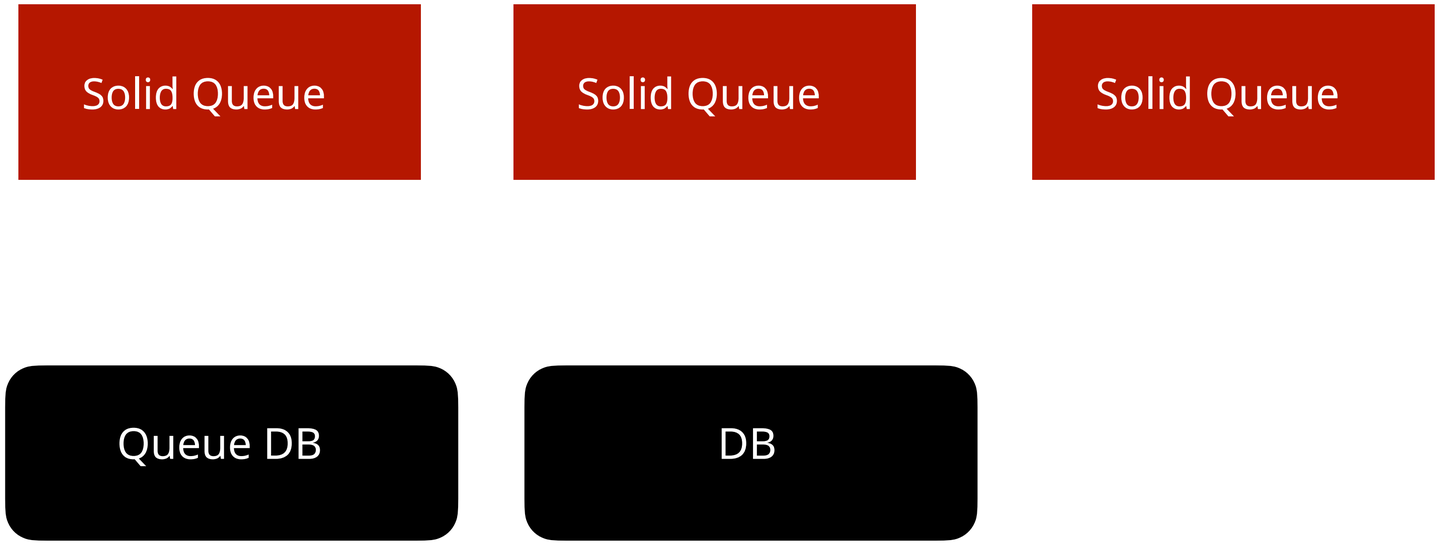


Solid Queue 速すぎて過負荷になった問題

原因は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!

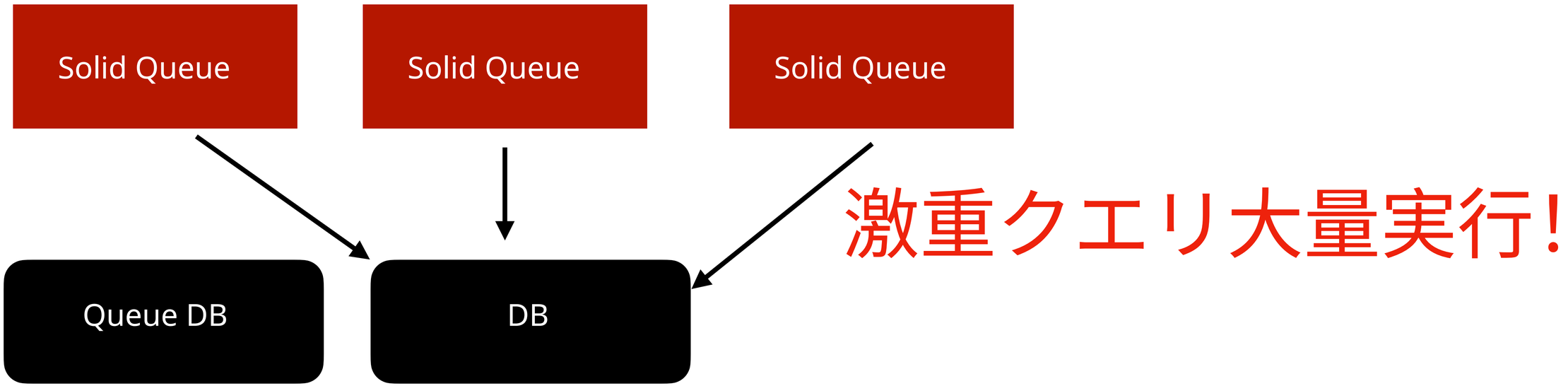


Solid Queue 速すぎて過負荷になった問題

原因は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！

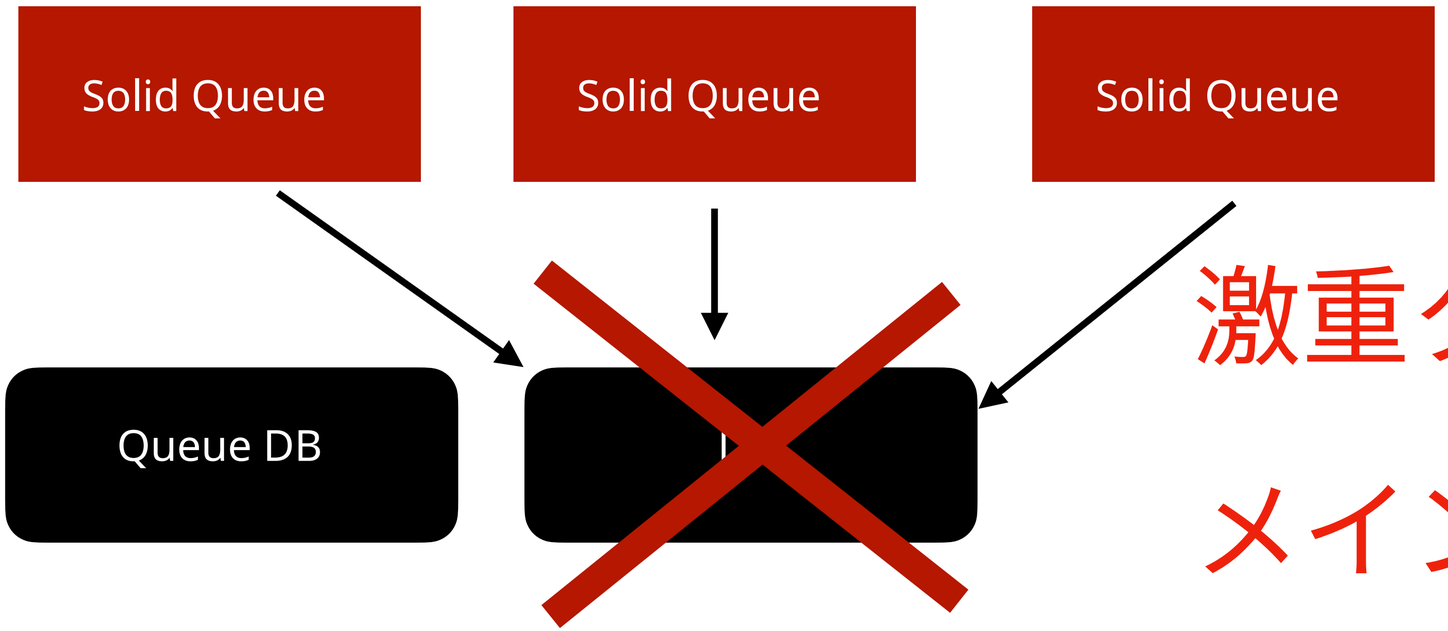


Solid Queue 速すぎて過負荷になった問題

原因は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

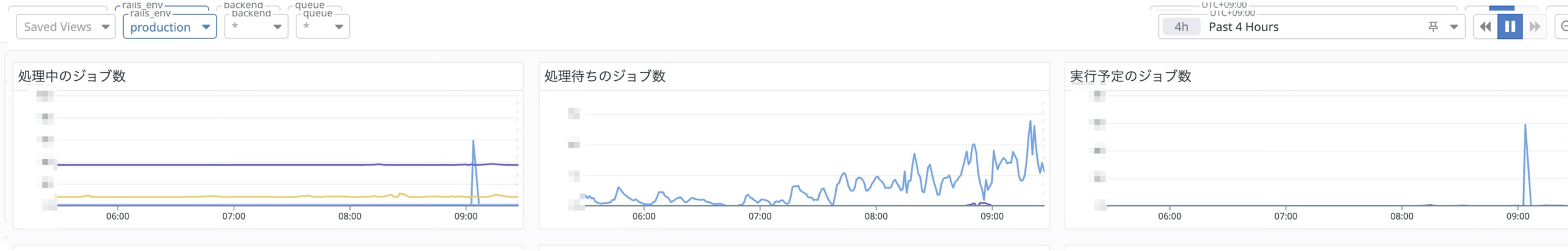
処理実行!



激重クエリ大量実行!

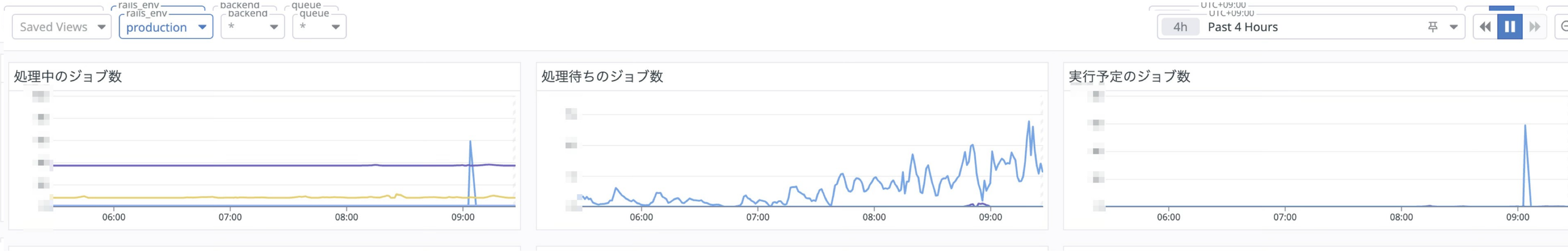
メインの DB が過負荷

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

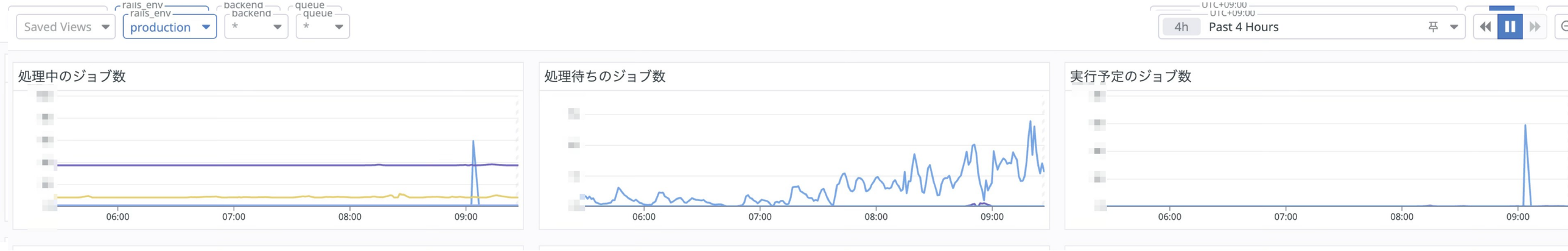
復旧優先のため、重いやつ専用 Worker / 専用キューを作った



シングルプロセス / シングルスレッドの流れてきても並列しない用

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

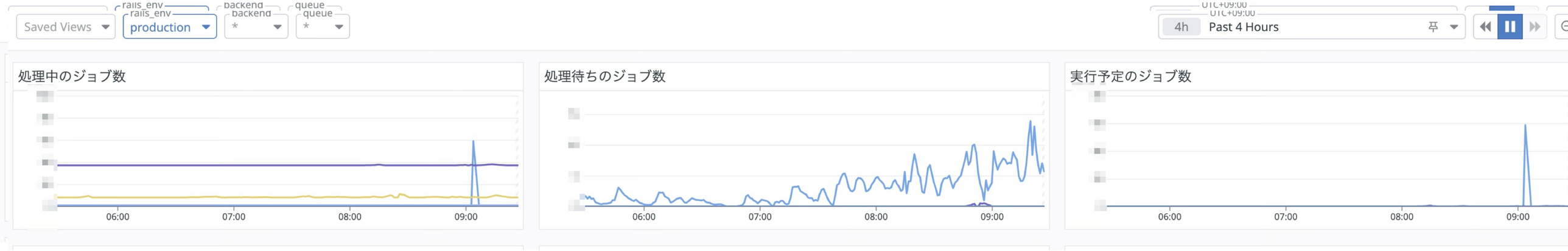


シングルプロセス / シングルスレッドの流れてきても並列しない用

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

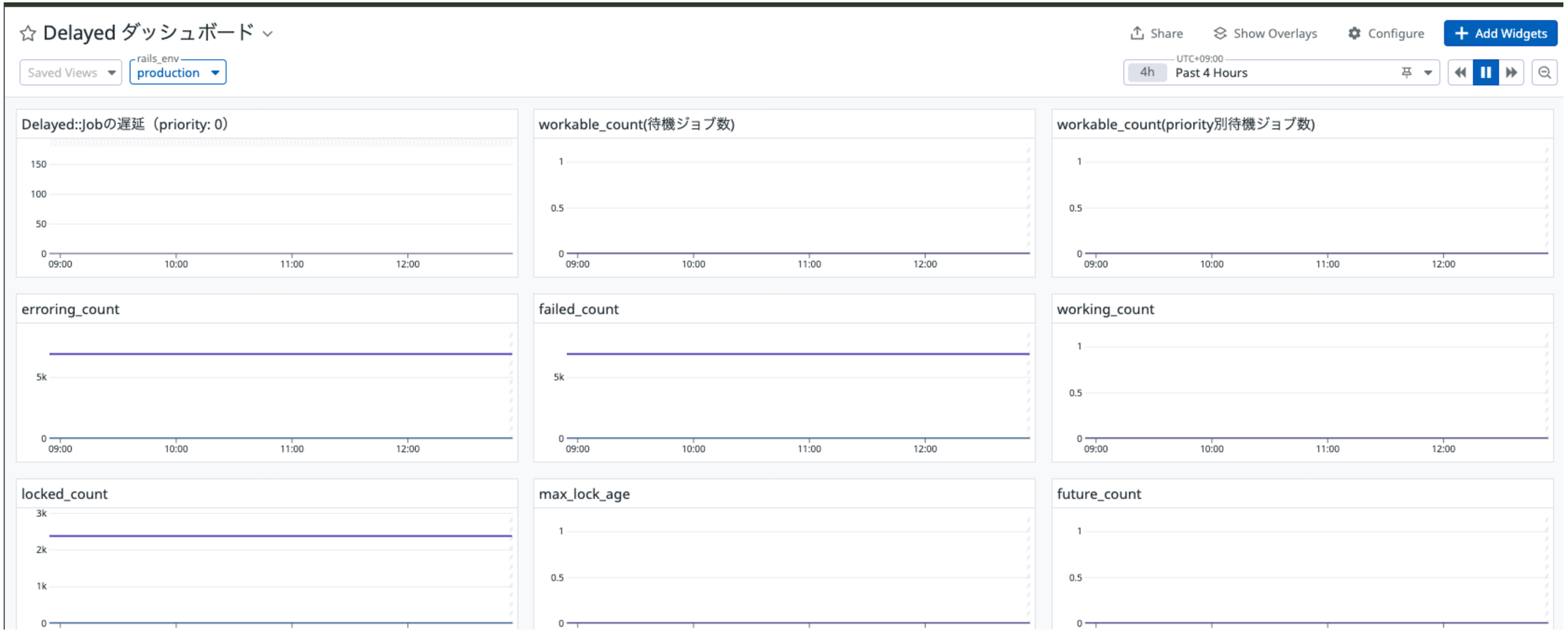
Rosa さんごめんなさい

Migration Requirements

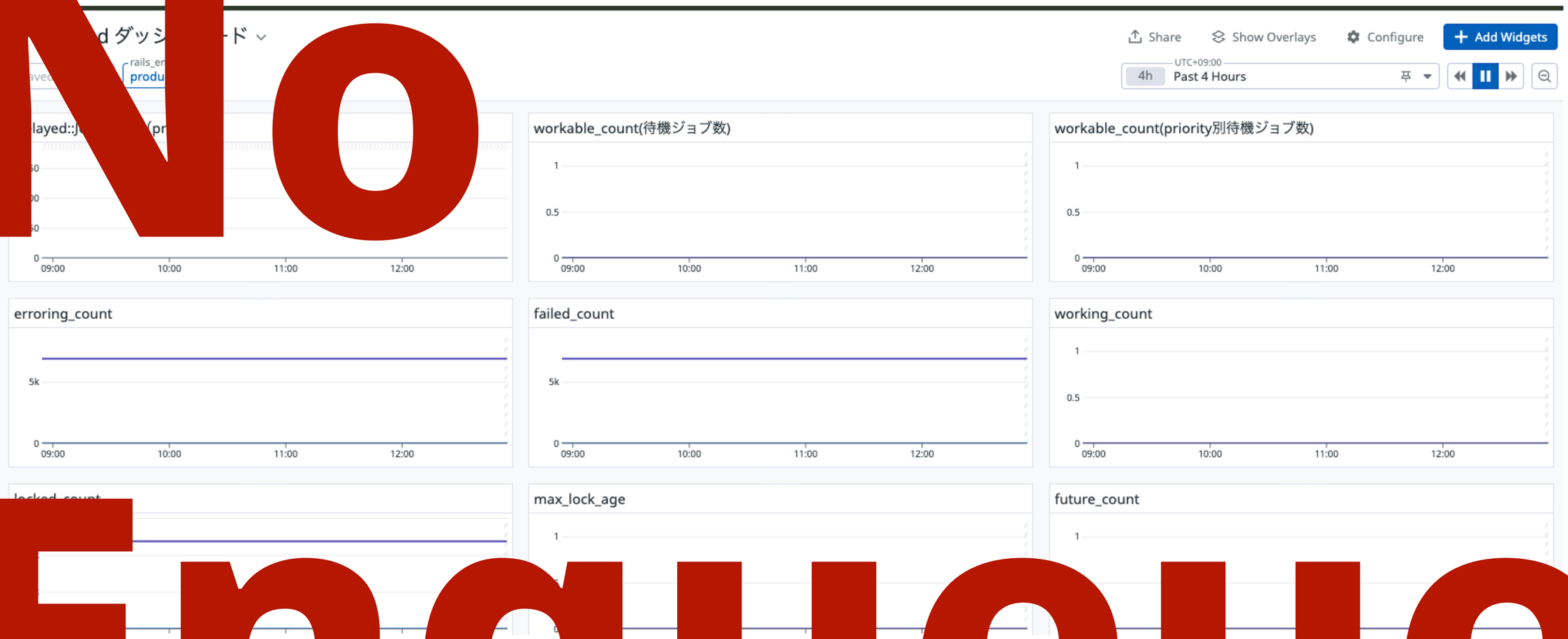
- ✅ .delay メソッドの撲滅
- ✅ ActiveJob で明示的に Delayed を指定
- ✅ SolidQueue 設定
- ✅ 監視
- ✅ SolidQueue を利用するように全 Job を変更
- ⬜ gem uninstall delayed

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



No Enqueue!

【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

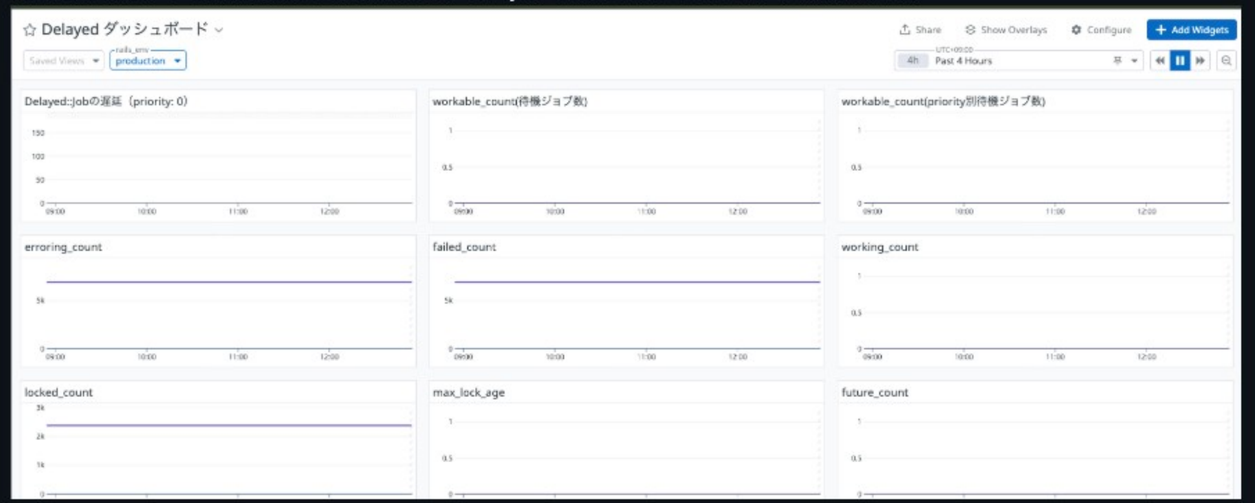
WHAT

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

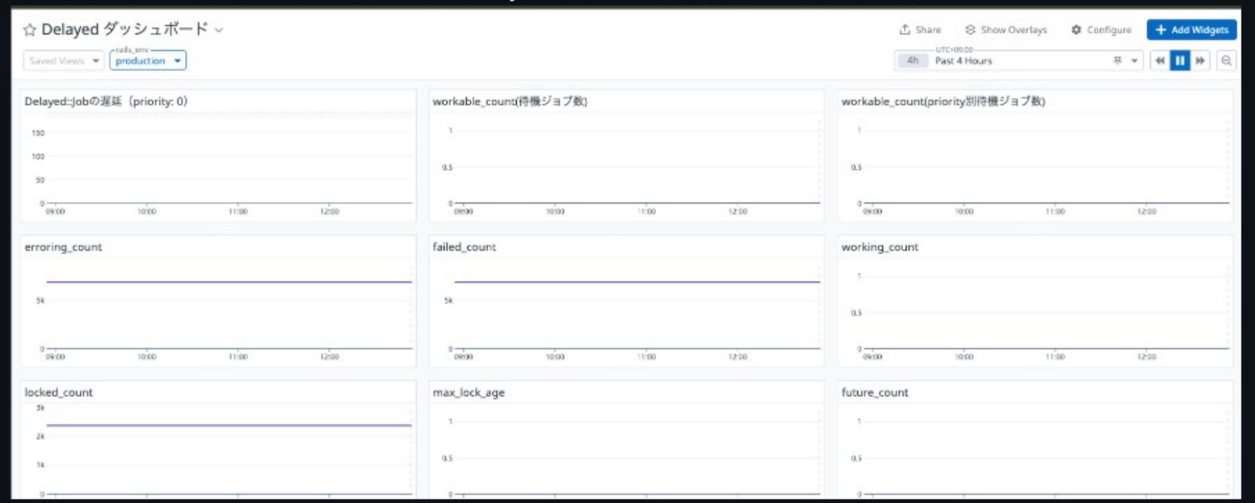
WHAT

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 ✓
- Kodaishikawa ✓

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

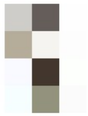
7 years ago...

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ぁ

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/分析基盤の開発・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。

100秒でわかる「Teachme Biz」

後で見る　共有

手順が見える、伝わる、拡がる

Teachme Biz

見る YouTube

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の、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。

スタディストのSREは君を待ってるぜ!

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

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！