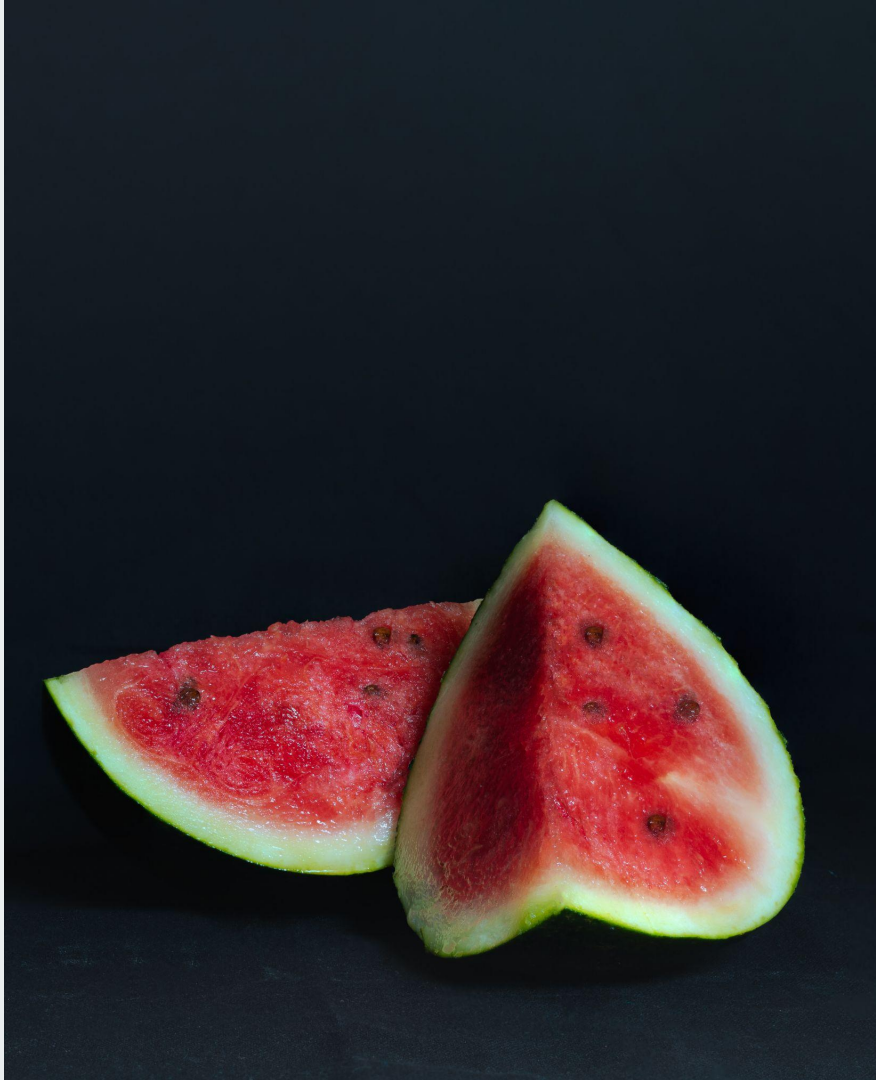
そのプロダクト、「正常に動いてます！」と言えますか？

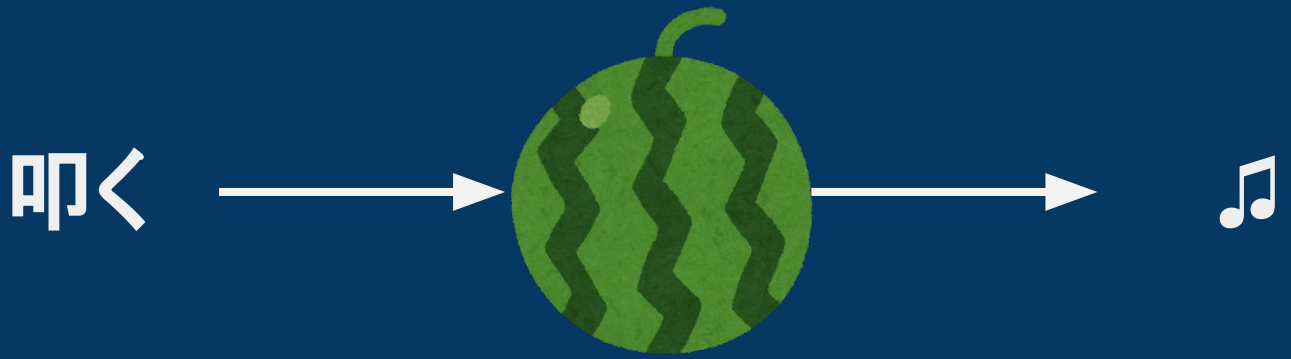
QA視点からのObservability入門

リンクアンドモチベーション
SREユニット QAエンジニア
小島 直毅



スイカの食べ頃は

Observable?



高音

叩く

中音

低音

叩く

高音

中音

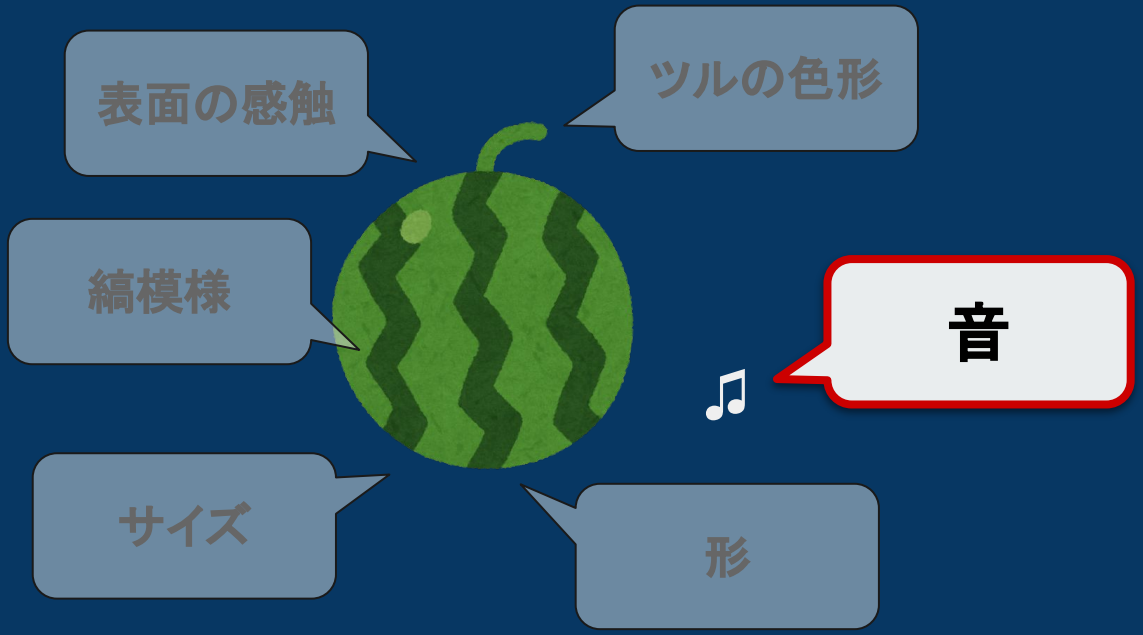
⬆️食べ頃！

低音

水分量


表面の感触
ツルの色形
縞模様
音
サイズ
形

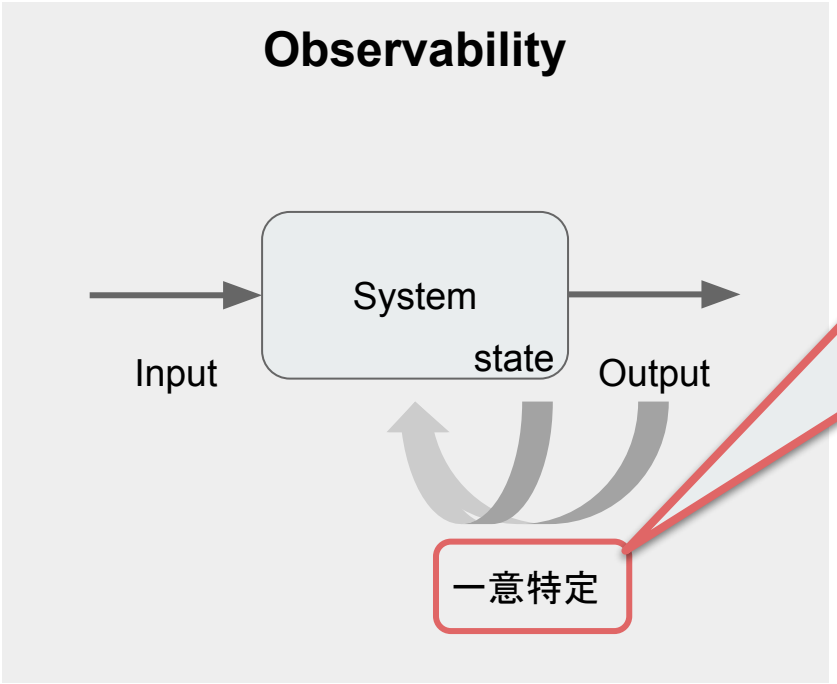
食べ頃？



スイカの食べ頃は

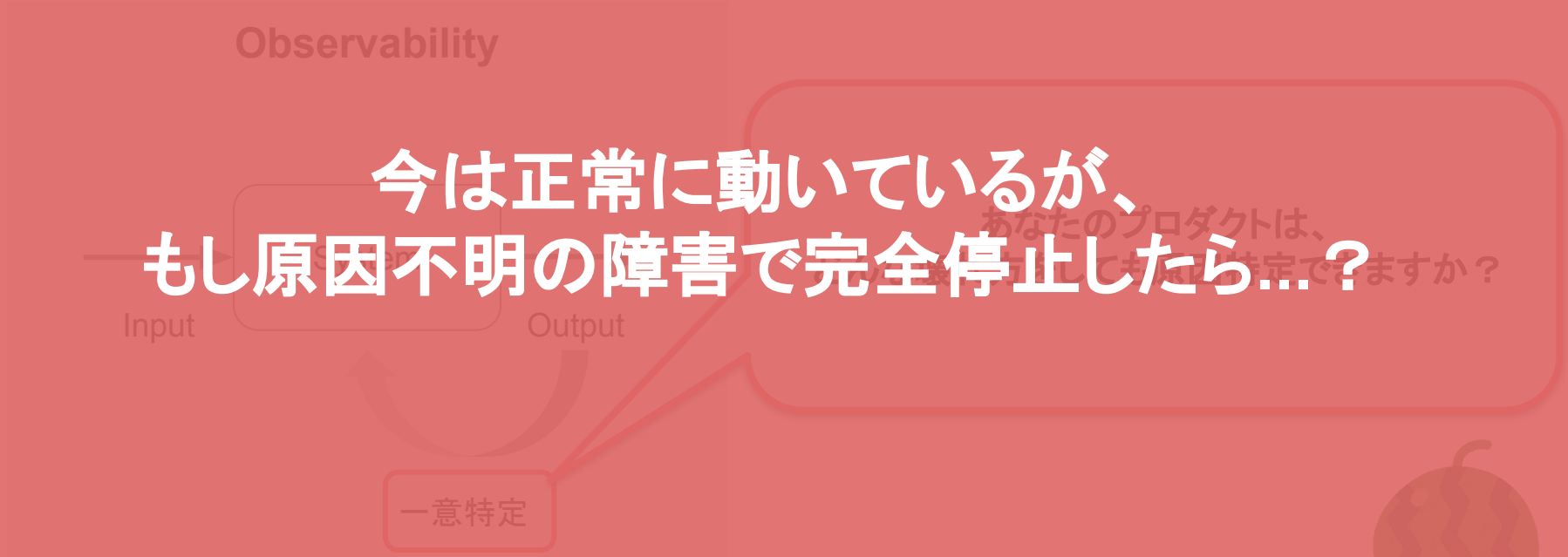
Observable!!

おさらい



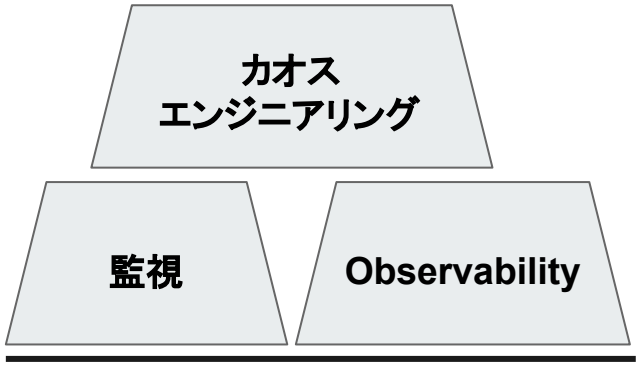
あなたのプロダクトは、
どんな壊れ方をしても原因特定できますか？

おさらい



今は正常に動いているが、
もし原因不明の障害で完全停止したら...？

今日のSRE



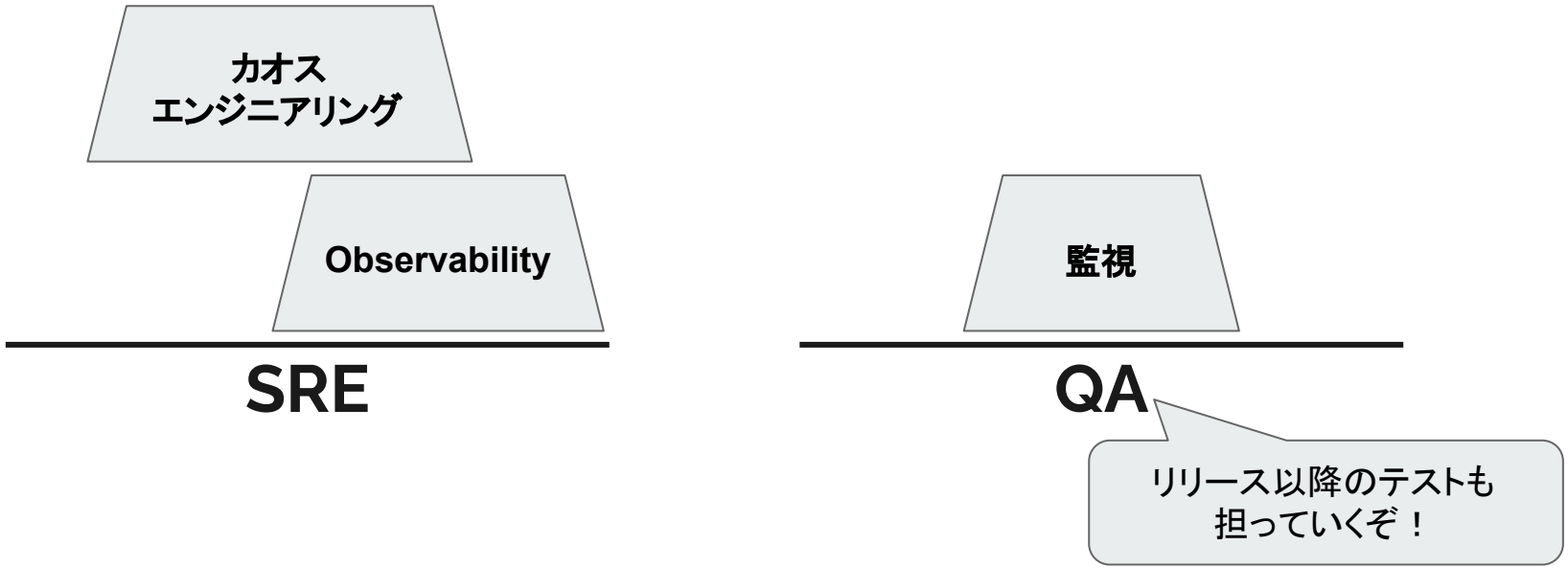
SRE

SREの勉強もしなきゃ 💦

助けられることは
ないだろうか？

QA

結論: 監視はQA(開発チーム)で持とう！



Today's Message:

プロダクトが複雑化すれば、組織も複雑化する
組織が複雑化すれば、「既知」を扱う監視はチームに依存する

QAはまず監視こそ学び、
チーム全体がObservabilityを実現できる体制を後押ししよう

自己紹介

小島 直毅 Naoki Kojima

所属・活動:

Link and Motivation Group

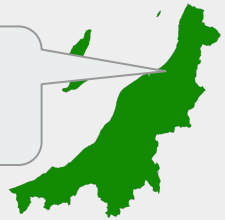
SREユニット QAエンジニア

SigSQA

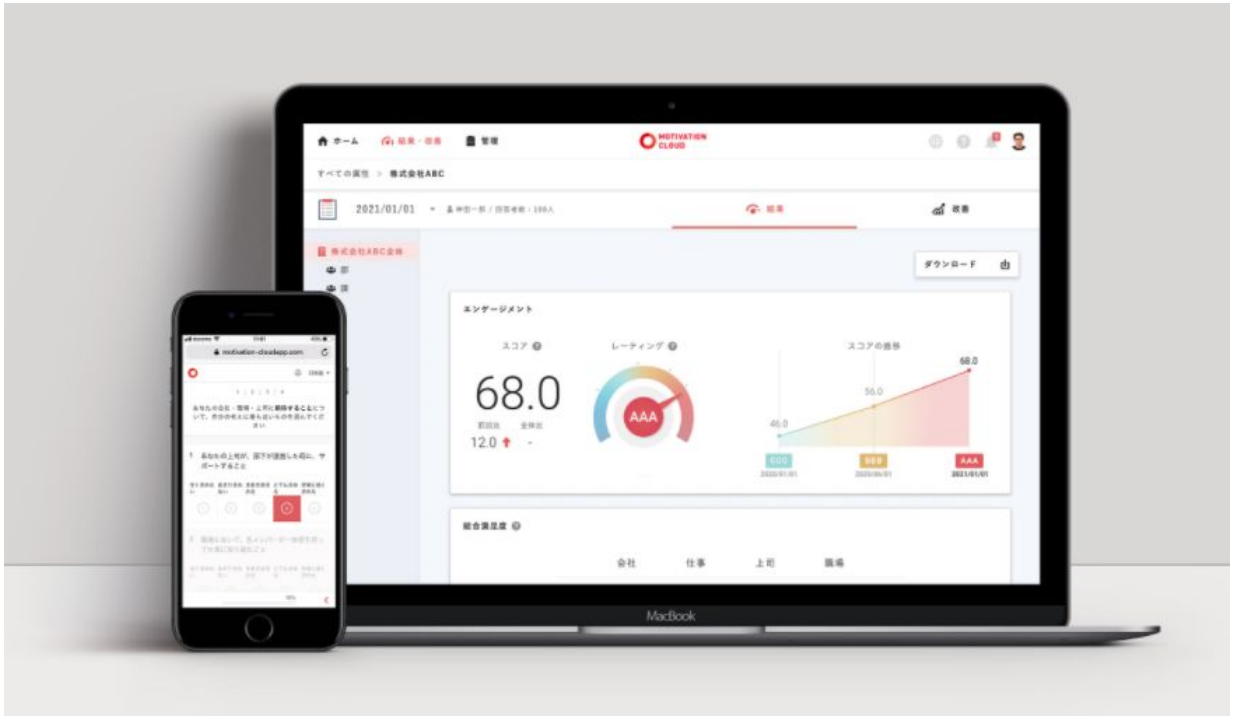
QAファンネルなど

新潟要素:

祖父が
新発田市出身！



趣味:



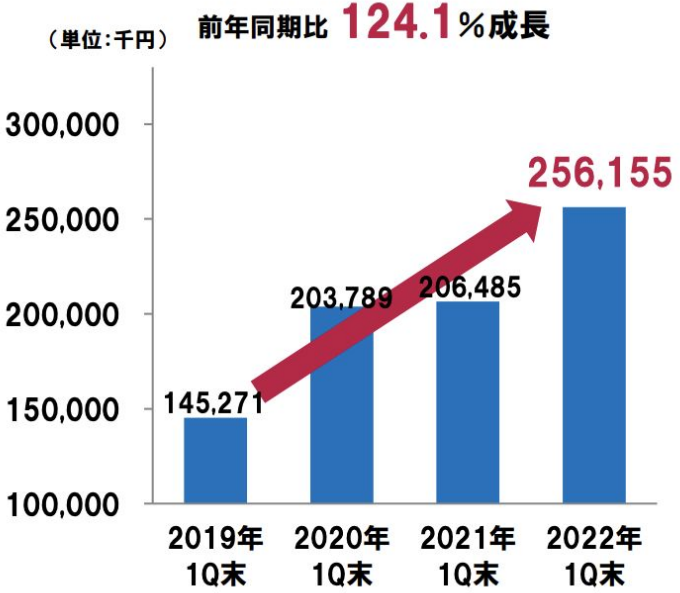
MOTIVATION CLOUD

by Link and Motivation Group

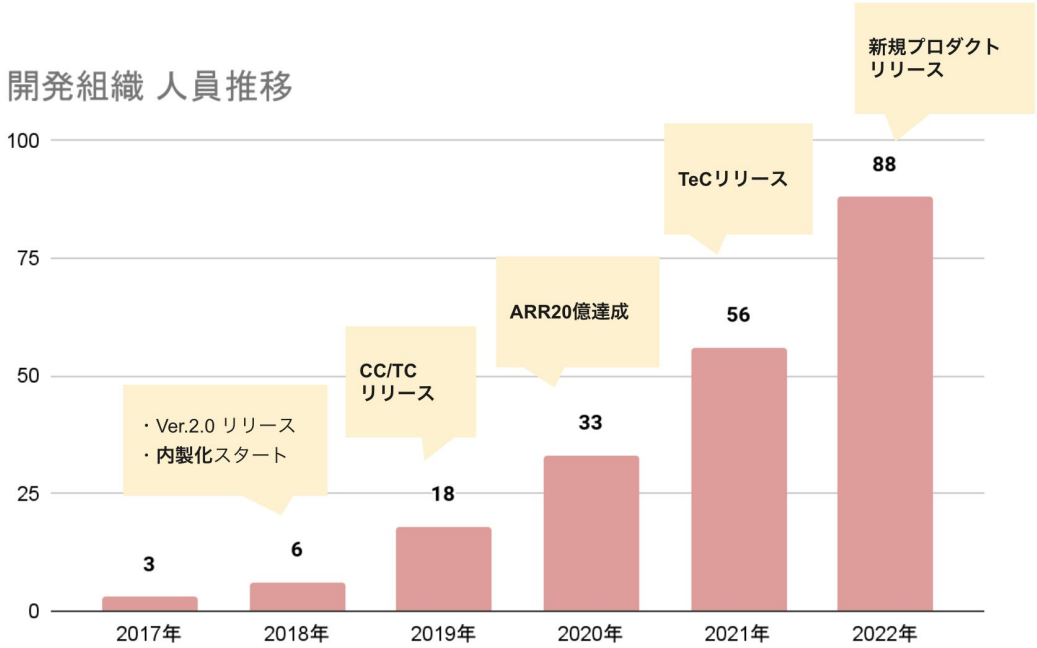
組織もObservableにしていきたい！

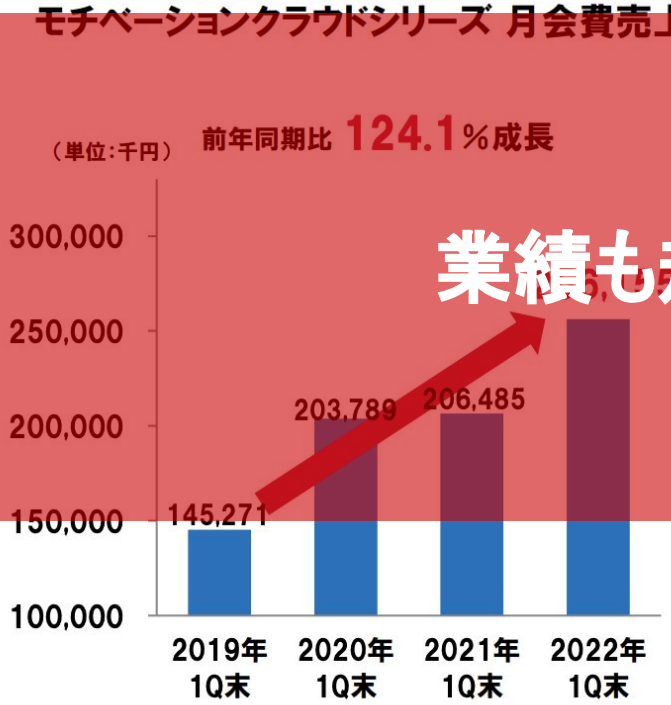
MOTIVATION CLOUD
by Link and Motivation Group

モチベーションクラウドシリーズ 月会費売上

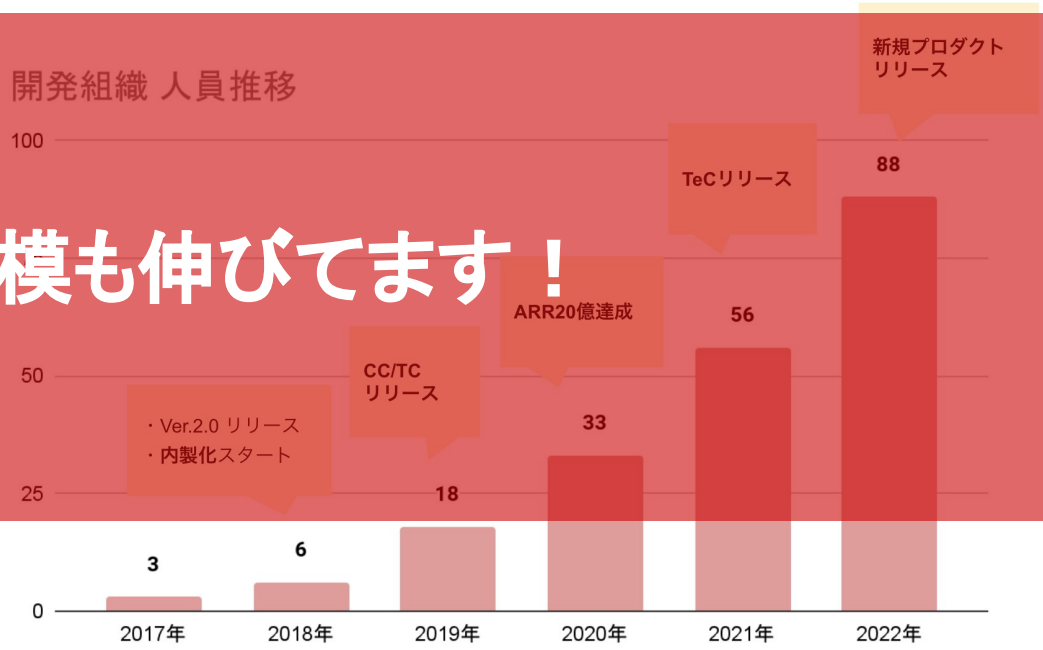


開発組織 人員推移





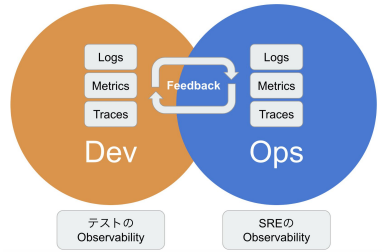
業績も規模も伸びてます！



本題はここから！

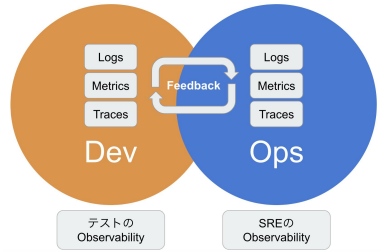
Agenda

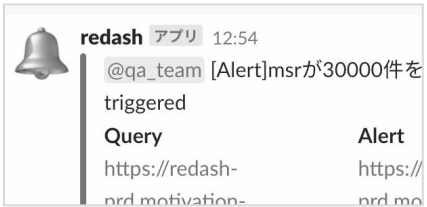
概念 → 理想 → 事例



Agenda

概念 → 理想 → 事例



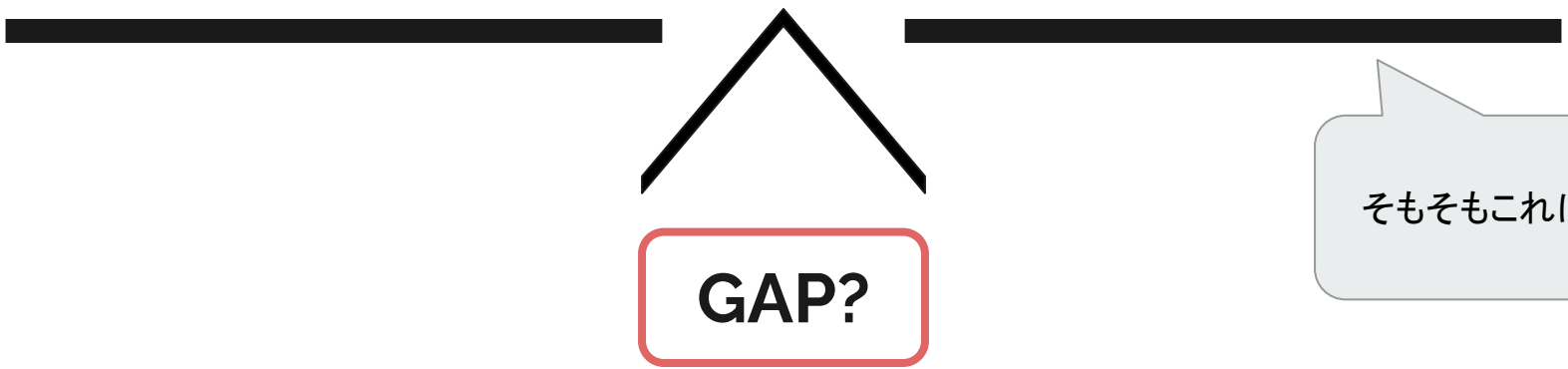


Observability?

色々な視点がありそうです

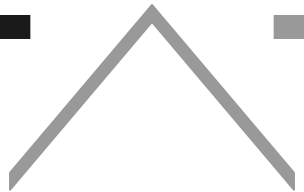
SRE視点の
Observability

QA視点の
Observability



SRE視点の
Observability

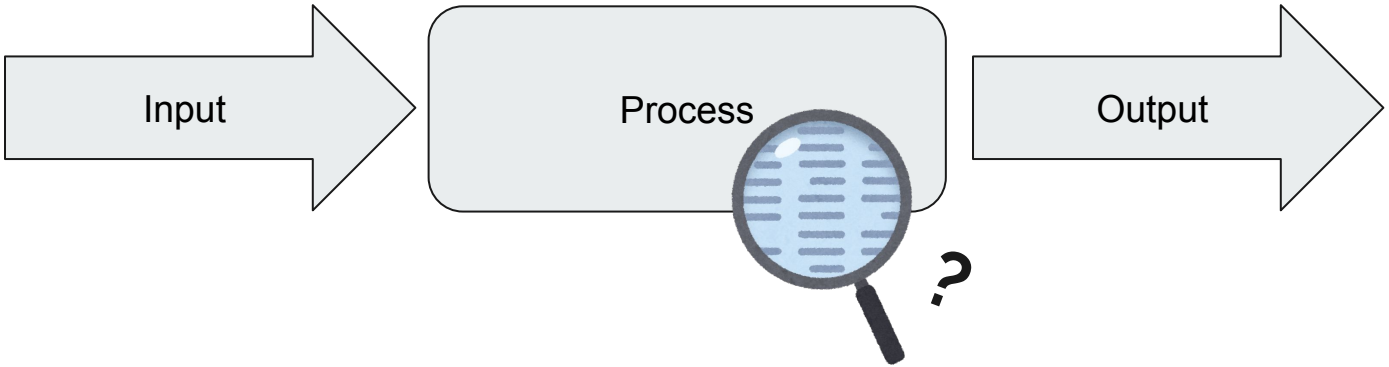
QA視点の
Observability



GAP?

そもそもこれは何？

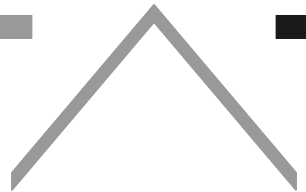
SRE視点のObservability（ここでは詳細省略！）



- 大規模かつ複雑化するシステムは、障害要因を事前推測するのに限界がある
  - 有事に原因を特定できる仕組みが重要に

SRE視点の
Observability

QA視点の
Observability

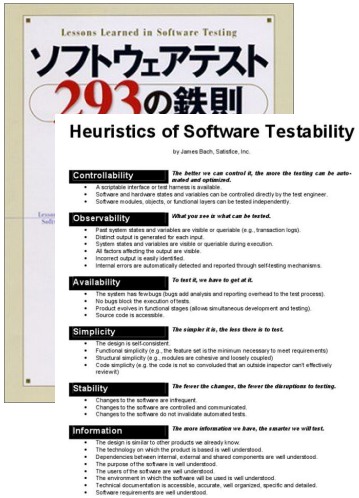


GAP?

そもそもこれは何？

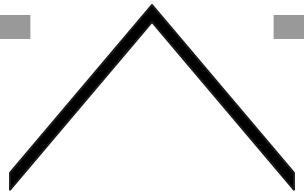
テストのObservability概要

- テスタビリティ(テスト容易性)の1つ
  - Operability
  - **Observability**
    - **「見えるものしかテストできない (から可視化しよう)」という概念**
      - クエリログやサーバー稼働率などが見れるか？
      - Inputが正確に把握できるか？
      - Outputが正確に把握できるか？
      - Outputの正誤判断が容易にできるか？
      - 内部エラーを自動的に出力できるか？
  - Controllability
  - Decomposability
  - Simplicity
  - Stability
  - Understandability



SRE視点の
Observability

テスト視点の
Observability



GAP?

概念まとめ

	SRE	QA
Observability	可観測性	観測容易性
Controllability	可制御性	制御容易性

未知の障害に**備える**

テストで**既知を増やす**

まとめ

	SRE	QA
Observability	可観測性	観測容易性
Controllability	可制御性	制御容易性

「プロダクトを観測し正常にする」という目的は同じ

「既知にできるか？」が2つを分ける

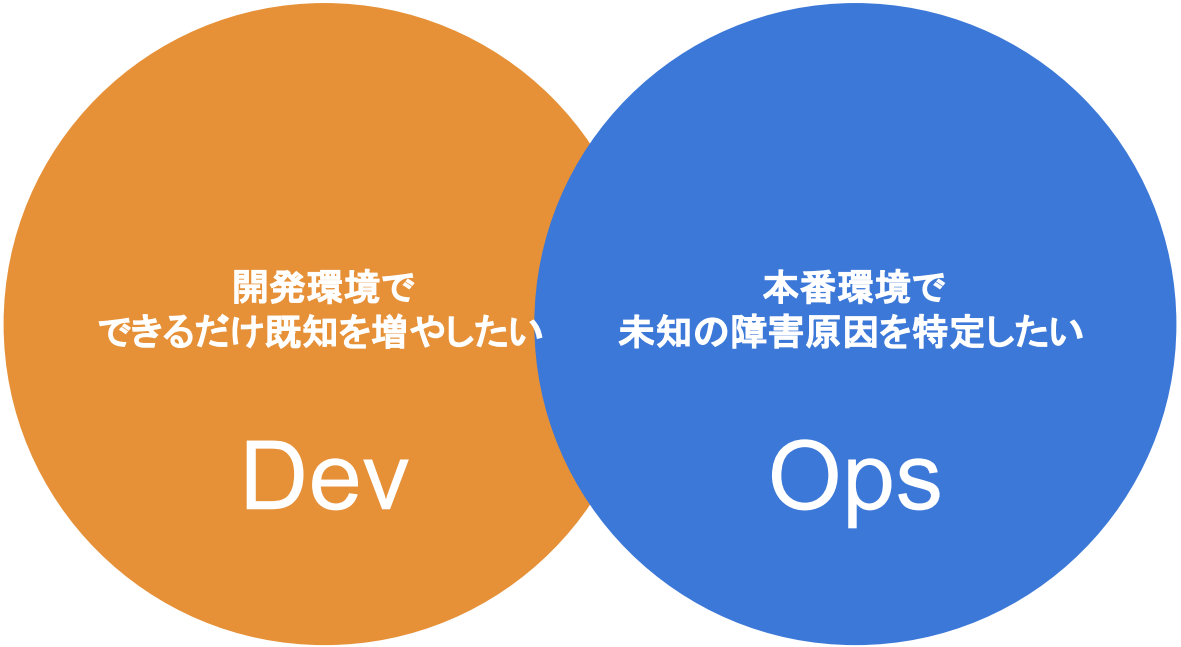
観測/制御できる状態を**提供**する

いざの時に**備える**

観測/制御できる状態を**改善**する

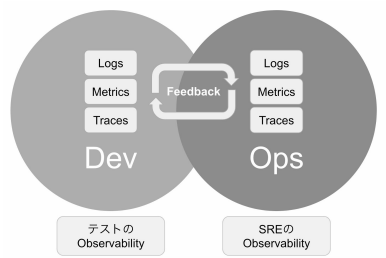
テストで観測/制御を繰り返す

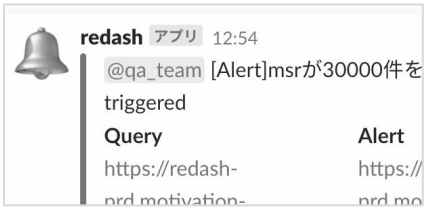
まとめると、


開発環境で
できるだけ既知を増やしたい
Dev
本番環境で
未知の障害原因を特定したい
Ops
QA視点の
Observability
SRE視点の
Observability

Agenda

概念 → **理想** → 事例

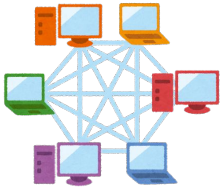




Case: LM

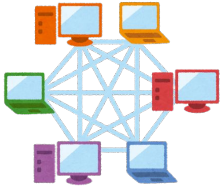
プロダクト・組織の複雑化と、
SREの認知負荷を時系列に追っていきましょう

システムの複雑化は、組織の複雑化と同時に起きる



SRE/Opsの
認知負荷

システムの複雑化は、組織の複雑化と同時に起きる

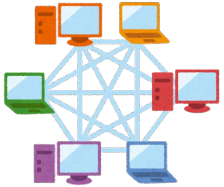


SRE/Opsの
認知負荷

システム構成はシンプル
SREは多くて1人
インフラ担当が監視もする

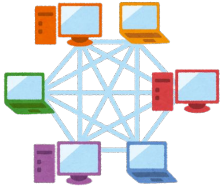
システムの複雑化は、組織の複雑化と同時に起きる

システム構成は1プロダクト4サーバー
SREは多くて2人
SREや運用担当がまばらに監視する

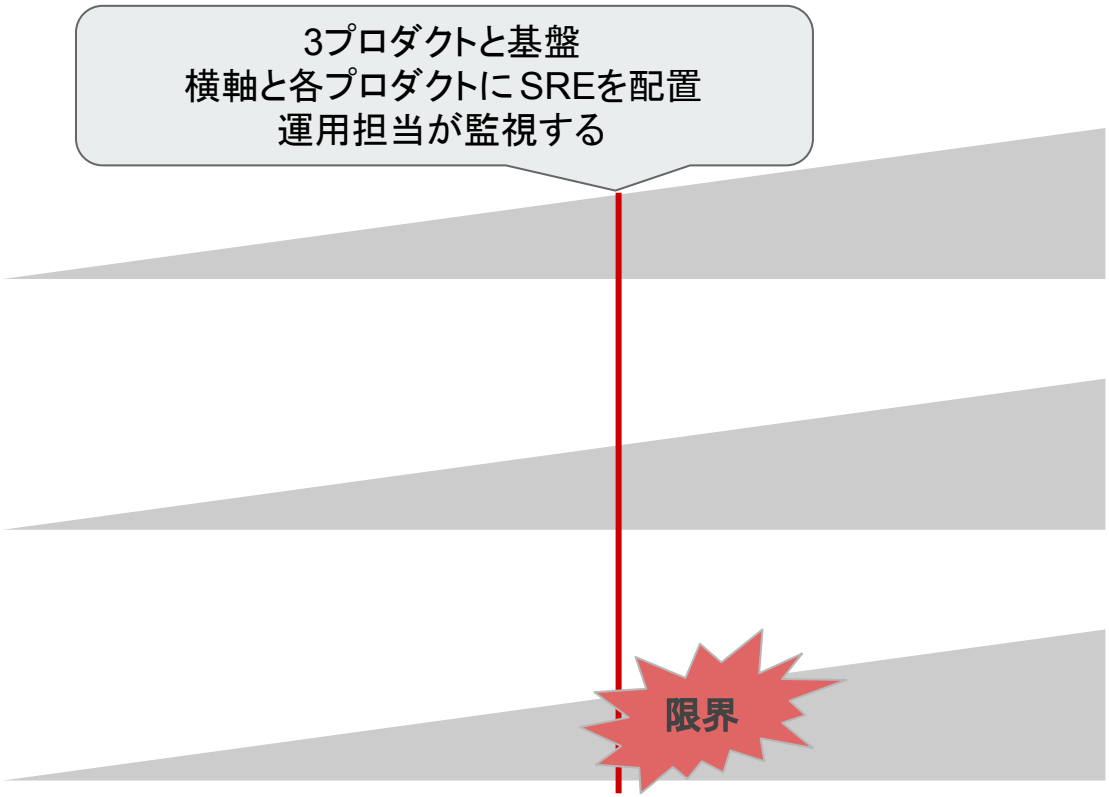


SRE/Opsの
認知負荷

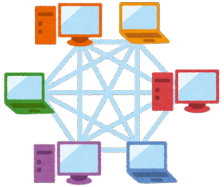
システムの複雑化は、組織の複雑化と同時に起きる



SRE/Opsの
認知負荷



システムの複雑化は、組織の複雑化と同時に起きる



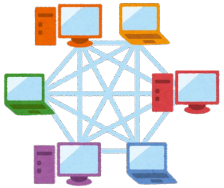
SRE/Opsの
認知負荷

5プロダクトと複数の基盤
横軸と各プロダクトにSREを配置
役割を明確に決めて監視する

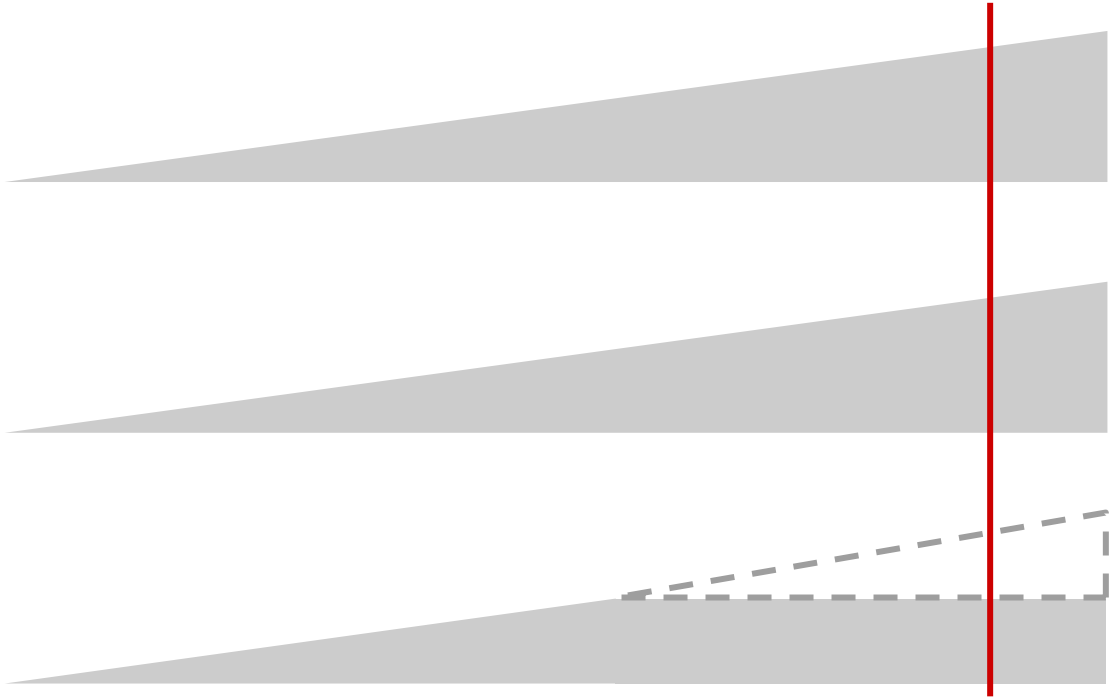
限界

理想

理想: 規模拡大により認知負荷が高まらない組織

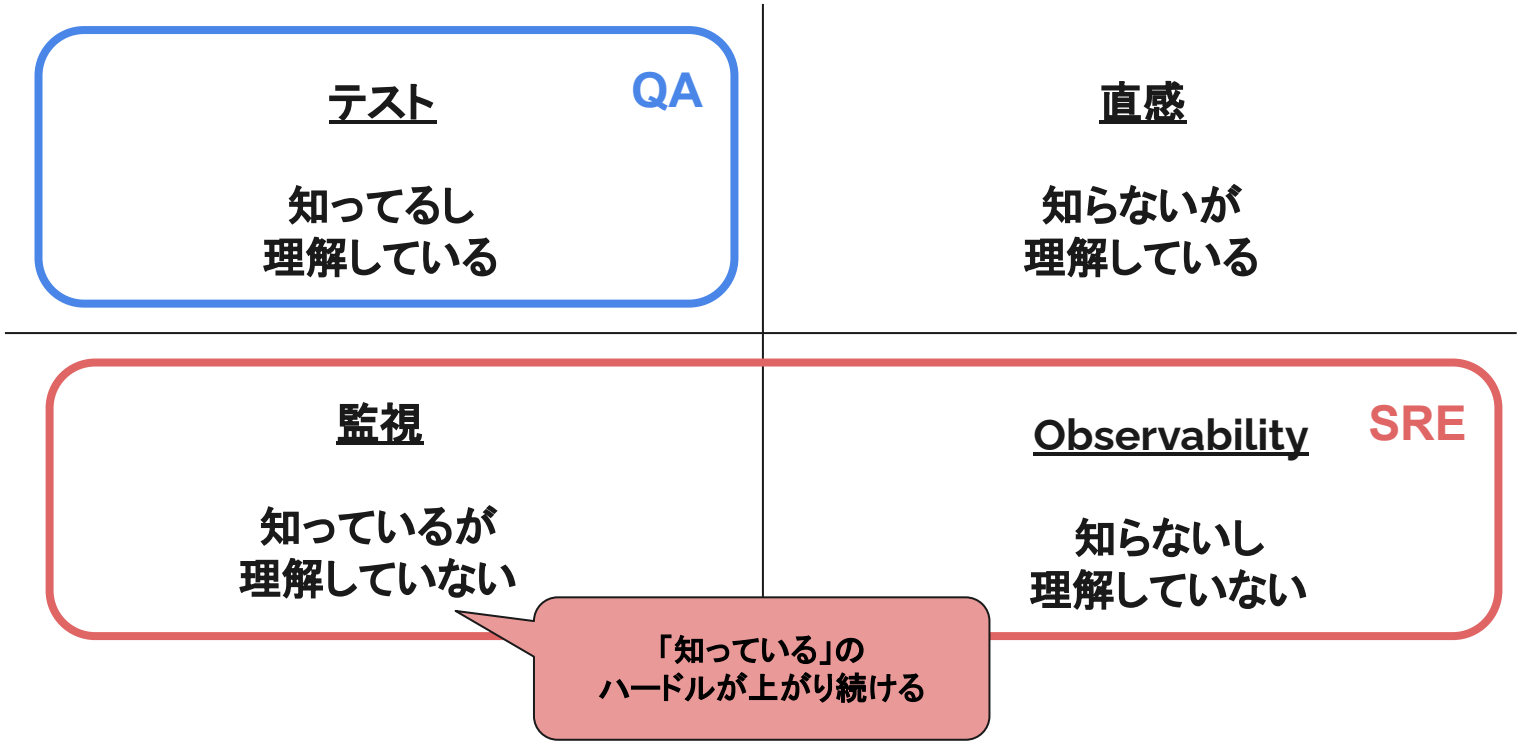


SRE/Opsの
認知負荷

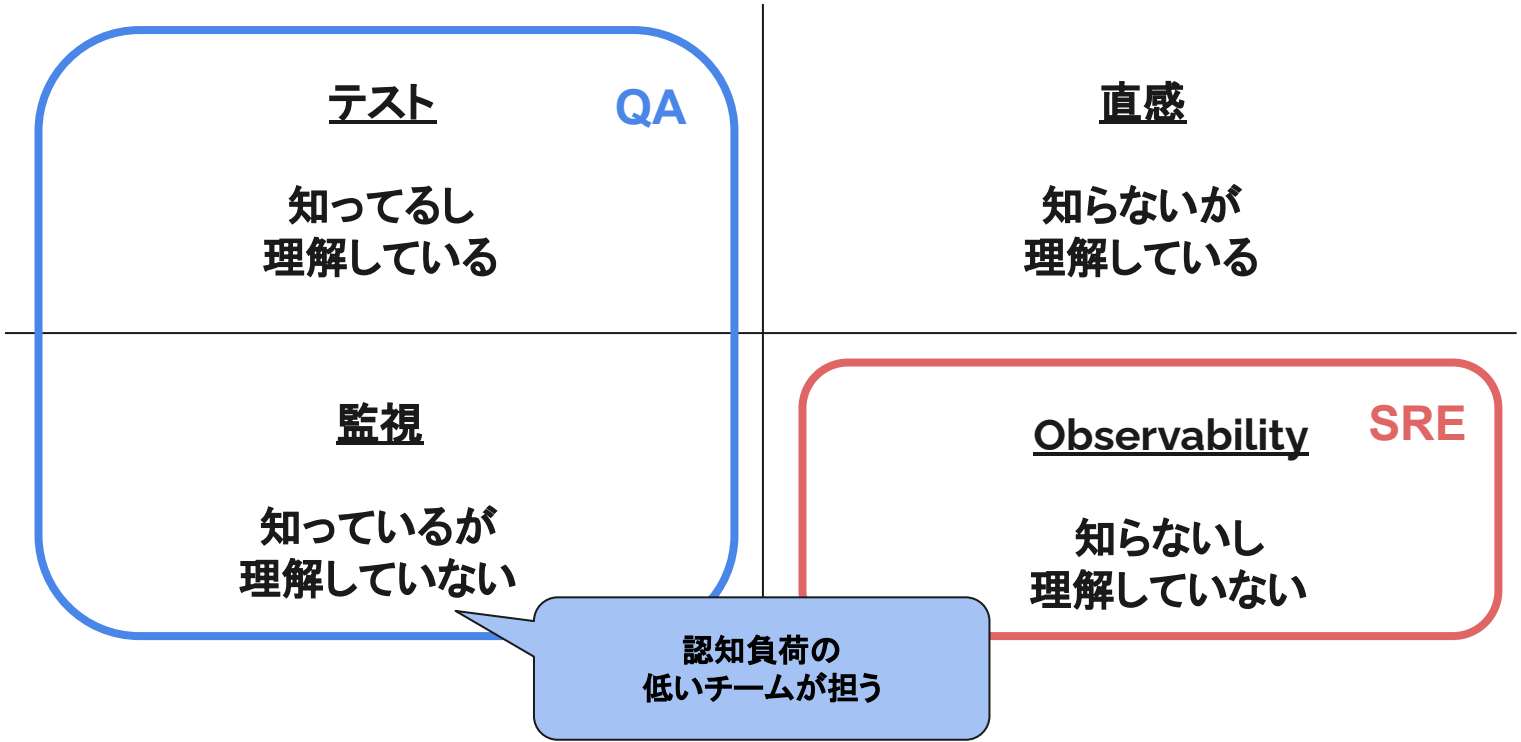


何が限界を生むのか？

SREの認知負荷はアプリ依存する領域から高まる



理想: QAはObservabilityの前に監視をやろう！

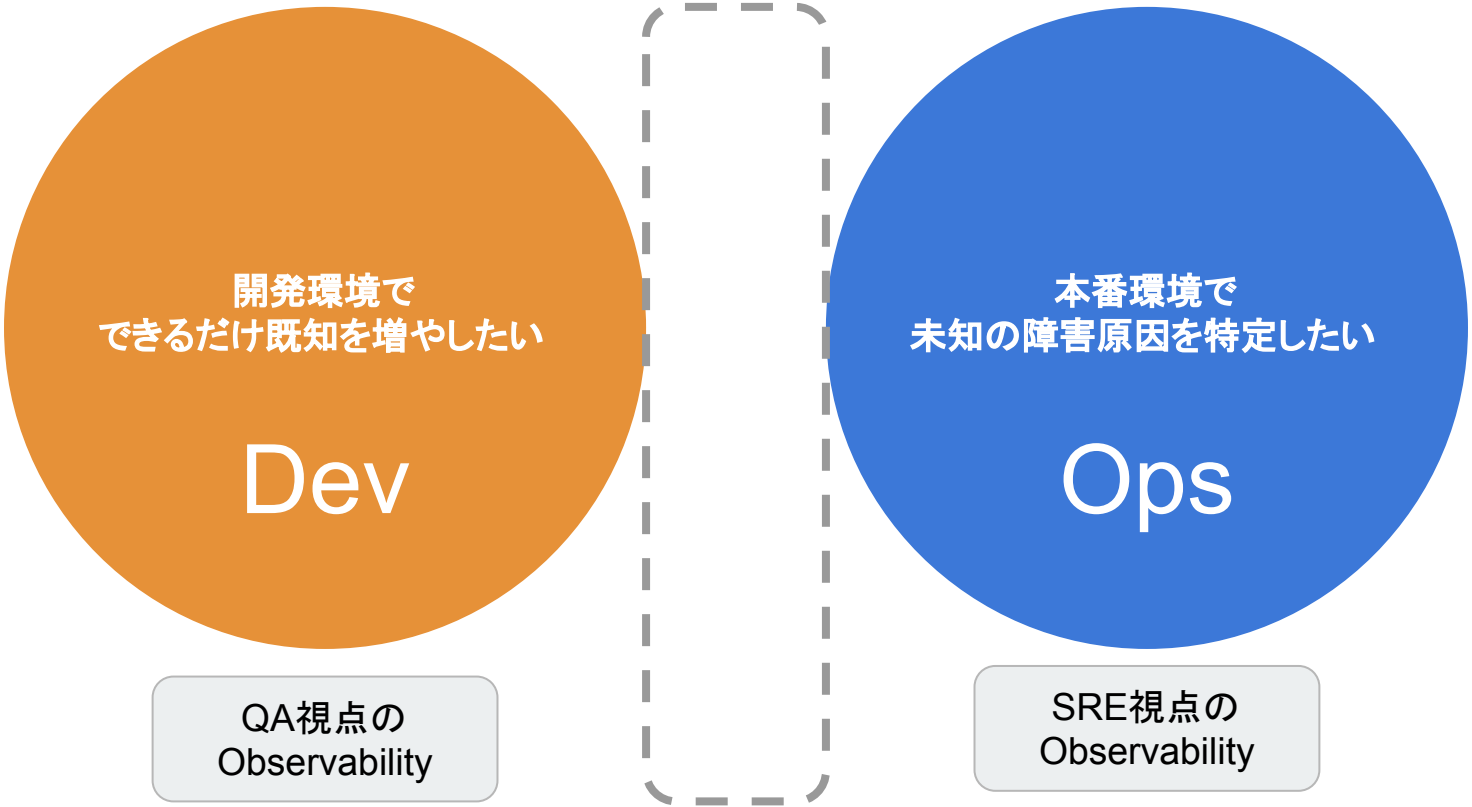


アンチパターン:

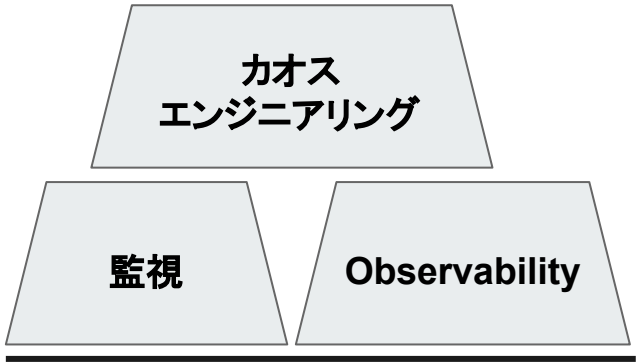
監視の認知負荷が高まりきるとどうなるか？

「知っている」の
ハードルが上がり続けると

実際は、いろいろなものが間に落ちていく



監視をSREが持っている場合



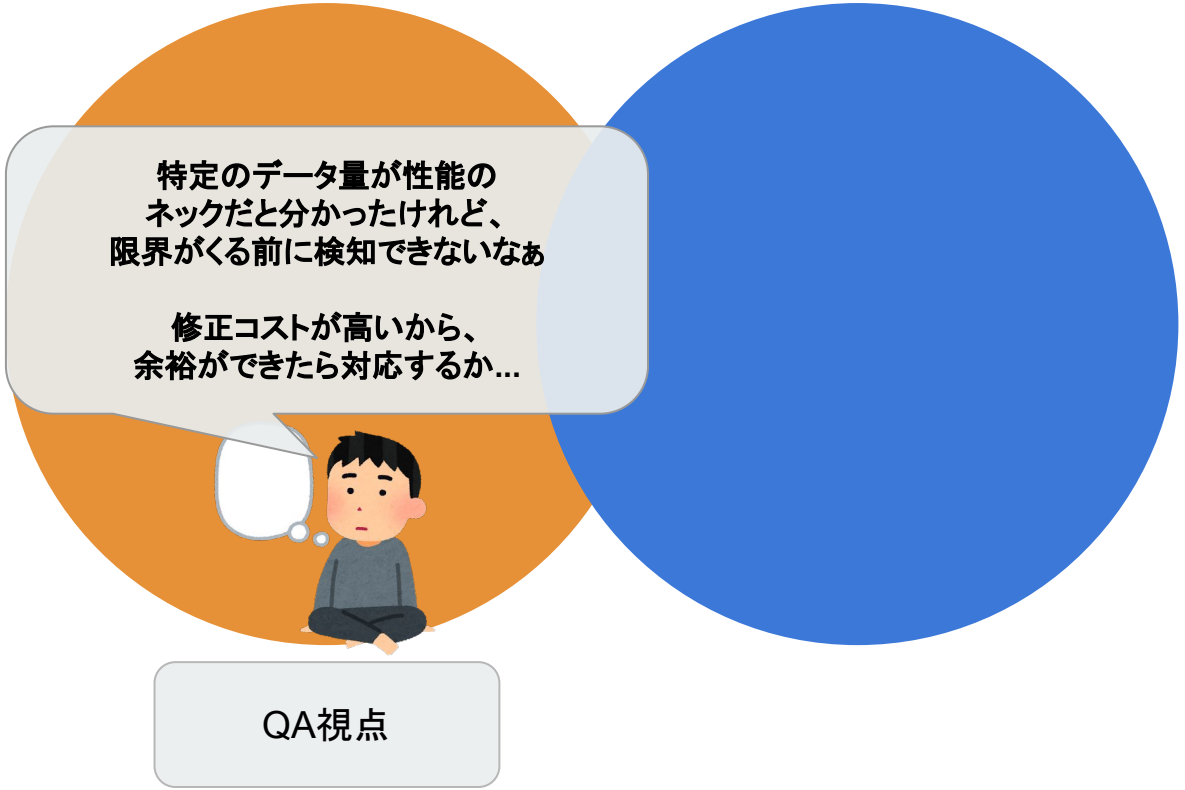
SRE

SREの勉強もしなきゃ💦

次の開発進めなきゃ！

QA

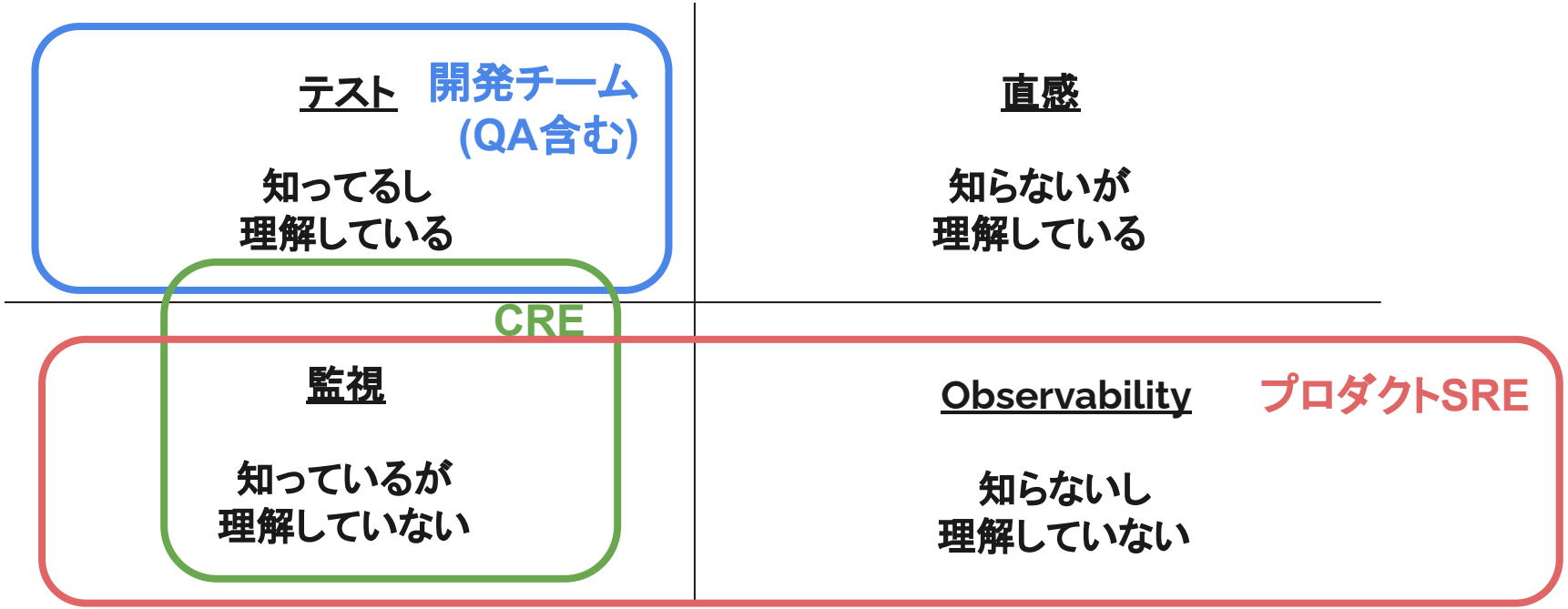
SRExQA連携できないケース(実例)



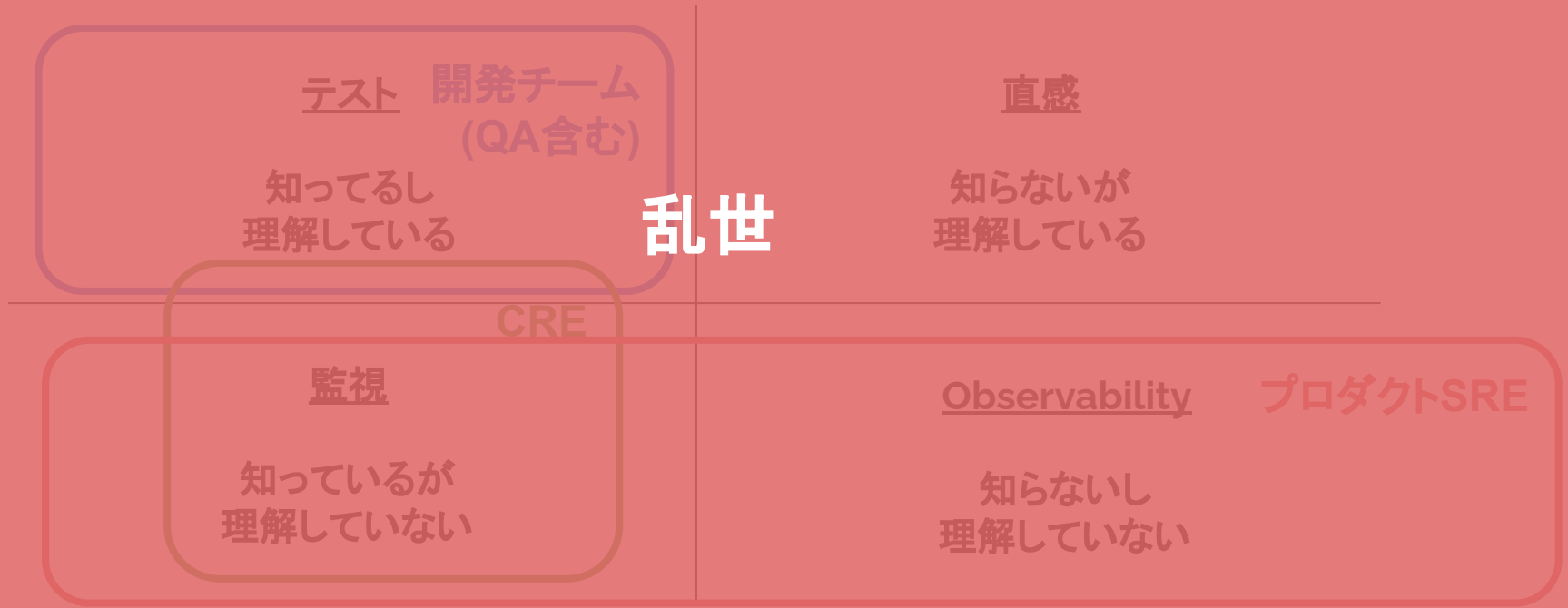
SRExQA連携できないケース(実例)

SRExQA連携できないケース(実例)

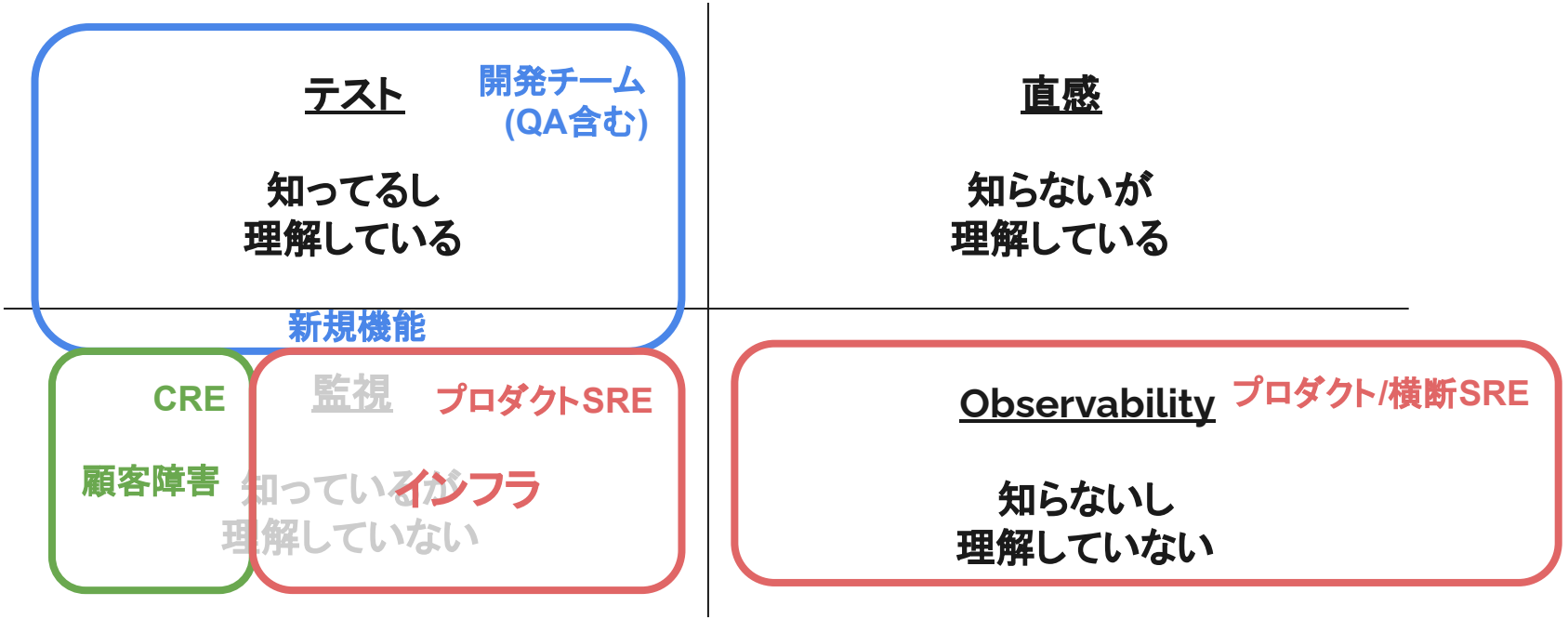
LMの場合: 1年前くらい



LMの場合: 1年前くらい



LMの場合: 現在


テスト
開発チーム
(QA含む)
知ってるし
理解している
直感
知らないが
理解している
新規機能
CRE
監視
プロダクトSRE
Observability
プロダクト/横断SRE
顧客障害
知っているが
理解していない
インフラ
知らないし
理解していない

もしアプリ開発チームが監視できていたら？

SRExQA連携できる例

XXテーブルのデータ量がネックになる場合を見越して、本番環境で3000レコード/ユーザーを超えたらアラート出すようにします！

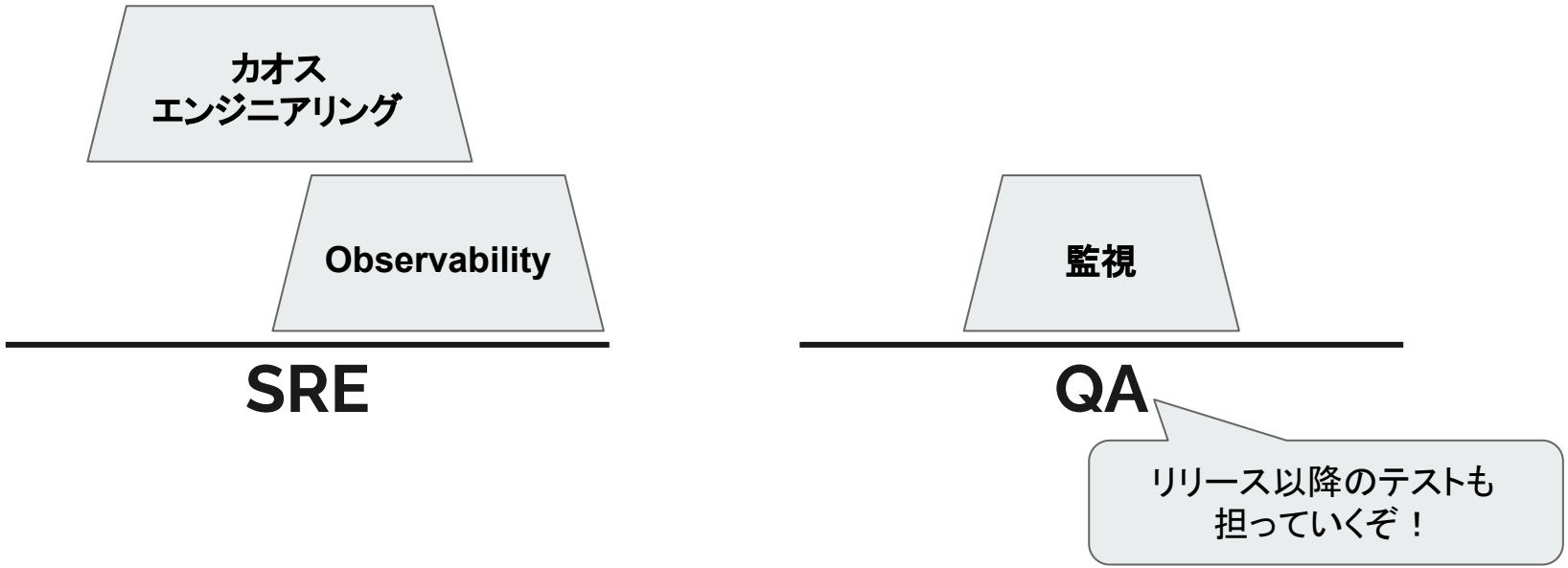
SRExQA連携できる例

XXテーブルのデータ量がネックになる場合を見越して、本番環境で3000レコード/ユーザーを超えたらアラート出すようにします！

ツール使ってくれてありがとうございます！

開発チームが自走してくれるので、
未知の障害予防に集中できます！

結論: 監視はQA(開発チーム)で持とう！



「そもそもObservabilityに着手できない…」

と言わせてしまわないよう、QAから組織を動かす

SREとQAの間に落ちやすい課題(例)

- 特定のパターンで障害が発生しやすいとわかっているもの
- ユーザーにより、正常/異常が変わりうるもの(負荷・性能など)
- 外部要因により、正常/異常が変わりうるもの(外部ツールなど)
- 開発環境のテスタビリティ

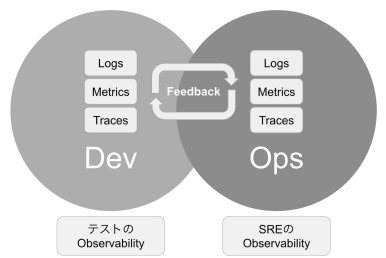
多くは**「SREが知らない状態で本番に入りうるリスク」**(アプリの裁量)

なので、以下のどちらがか必要

- **アプリからSREと連携してObservabilityを保つ**
- **アプリが自立してObservabilityを担保できる仕組み・体制を運用する**

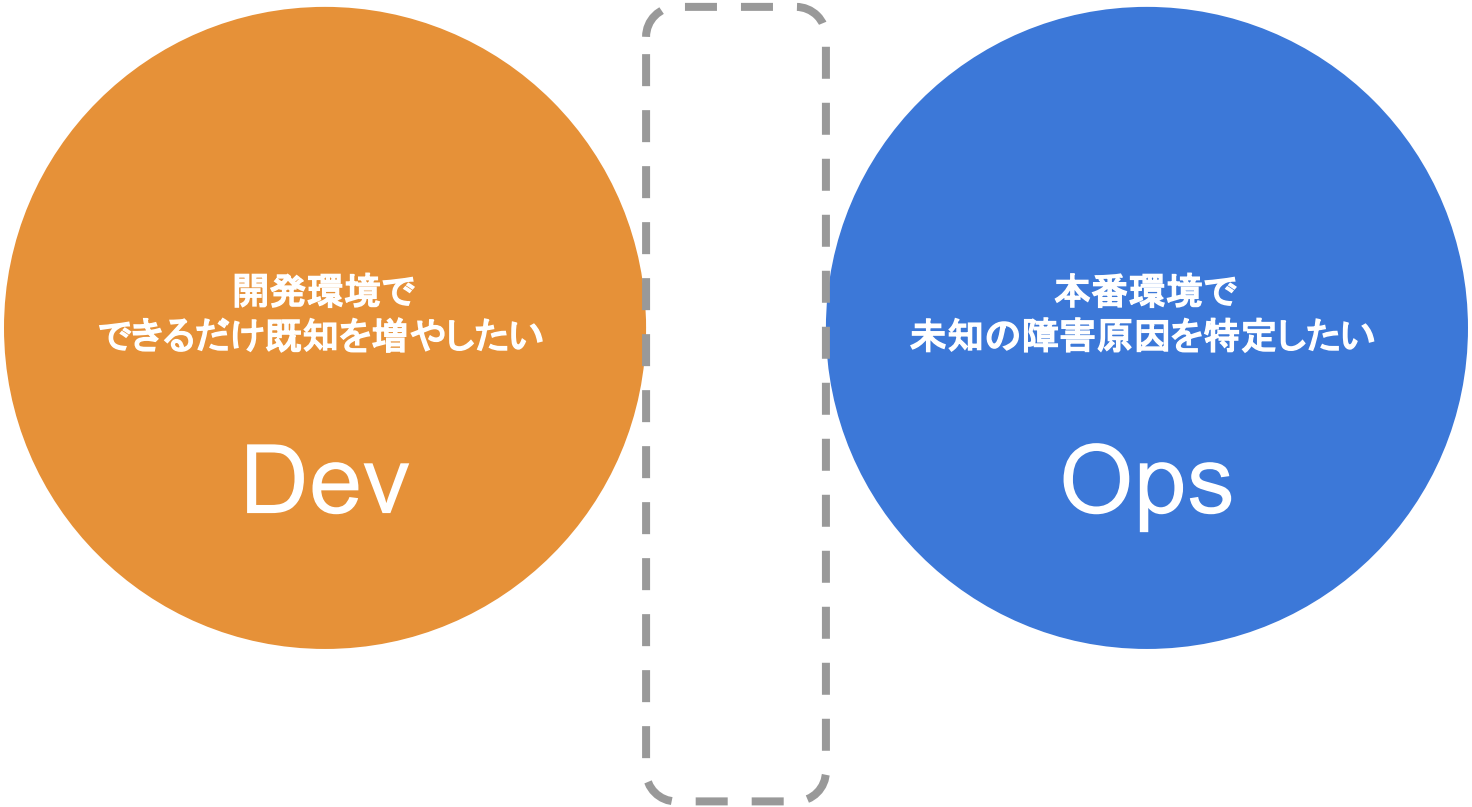
Agenda

概念 → 理想 → **事例**



「知っている」の
ハードルが上がり続けると

どれだけ事例を知っても、間に落ちていく



「知っている」の
ハードルが上がり続けると

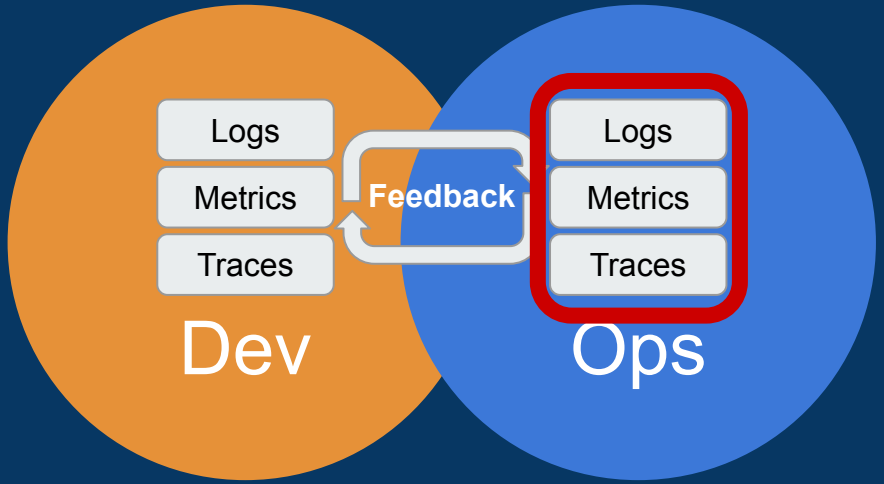
どれだけ事例を知っても、間に落ちていく

ただし、大事なのは事例数ではなく

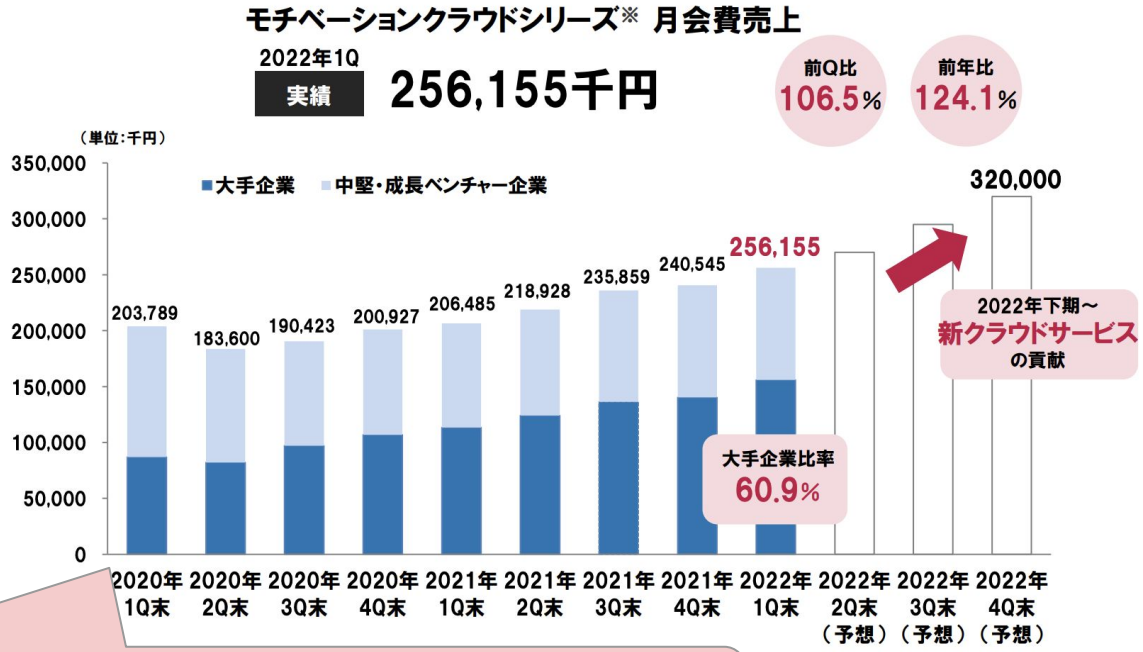
リスクに対するオーナーシップ！

なので、「やり方を知る」より「役割の変化を捉える」視点で聞いてください！

Case 1. 性能アラート

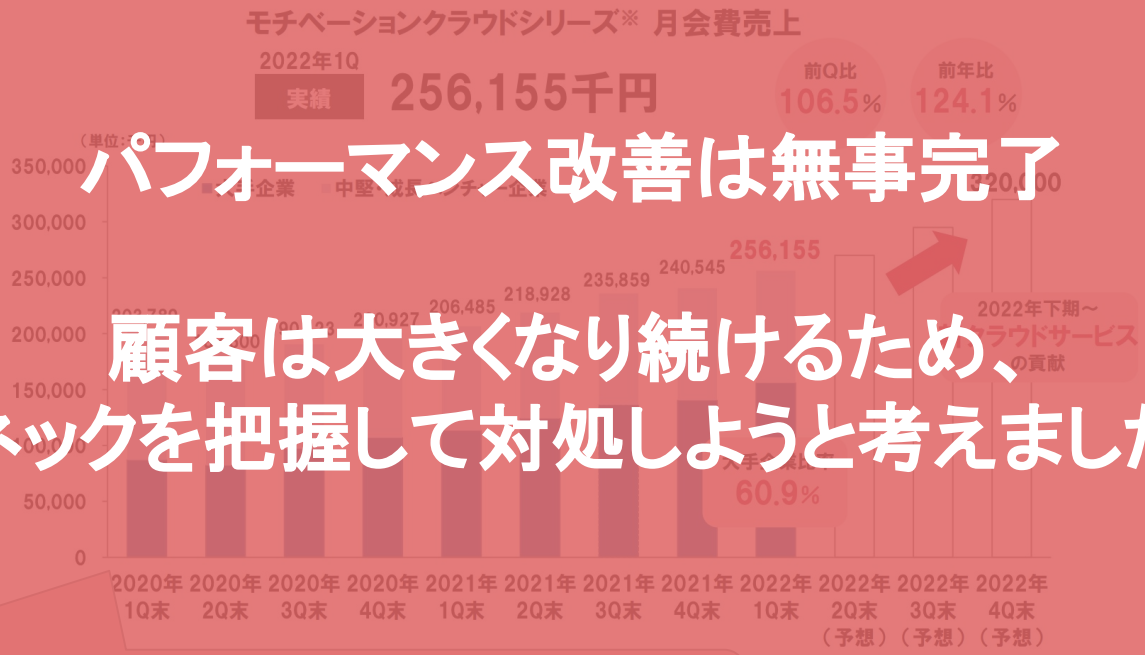


背景



エンタープライズ化するためには、性能改善が必要！

背景



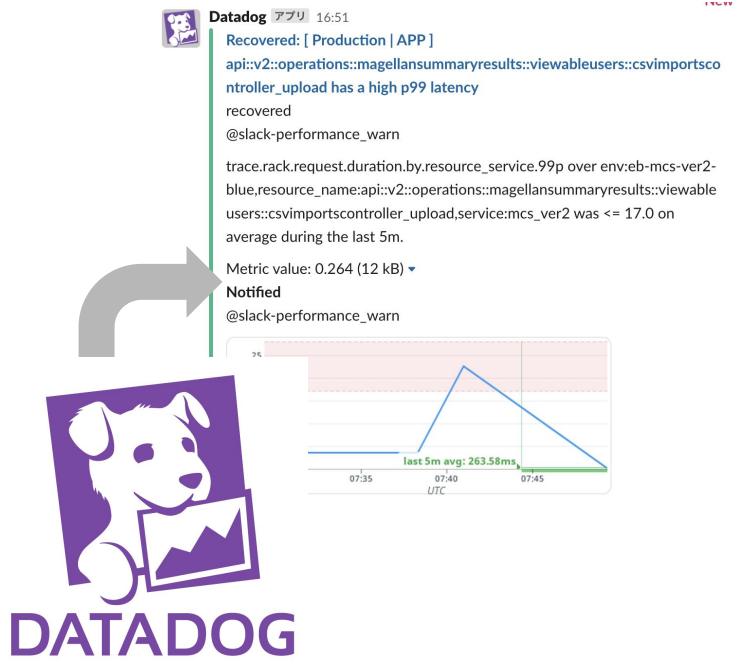
パフォーマンス改善は無事完了

顧客は大きくなり続けるため、
ネックを把握して対処しようと考えました

エンタープライズ化するためには、性能改善が必要！

打ち手

機能名	測定値(5万人)
api/v2/operations/magellan_viewable_users#csv_upload,1800.0	設定値1800s
api/v2/operations/magellan_viewable_users#recommend_csv_download,10	設定値10s
api/v2/surveys/statuses#show,5.0	設定値5s
api/v2/surveys/attributes#index,5.0	設定値5s
api/v2/magellan_summary_results#download_survey_summary,10.0	設定値10s



ネックになりうる各 APIや処理に閾値を設定

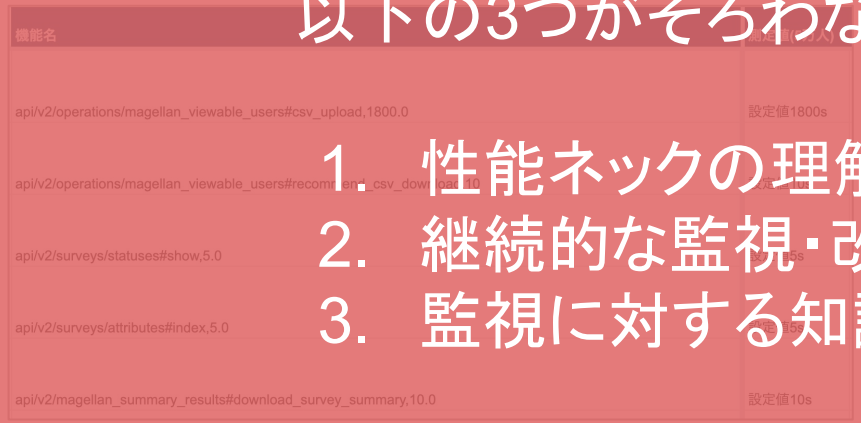
Datadogを連携し、閾値を超えたら slack通知・改修

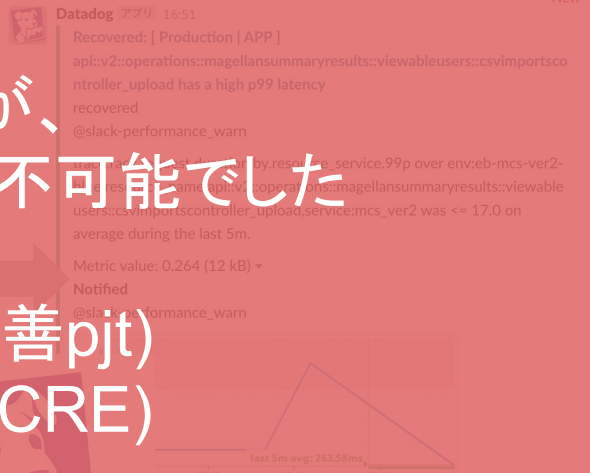
打ち手

シンプルに見えますが、
以下の3つがそろわないと実現不可能でした

1. 性能ネックの理解 (性能改善pjt)
2. 継続的な監視・改善体制 (CRE)
3. 監視に対する知識 (SRE)

=> 意志を持って役割設計すること
さもなくば間に落ちます

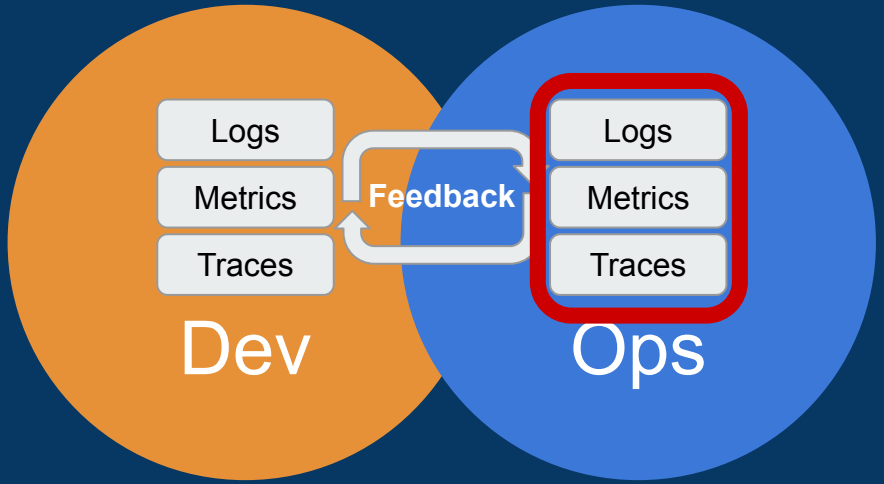




ネックになりうる各 API で監視・閾値を設定

Datadogを連携し、閾値を超えたら slack通知・改修

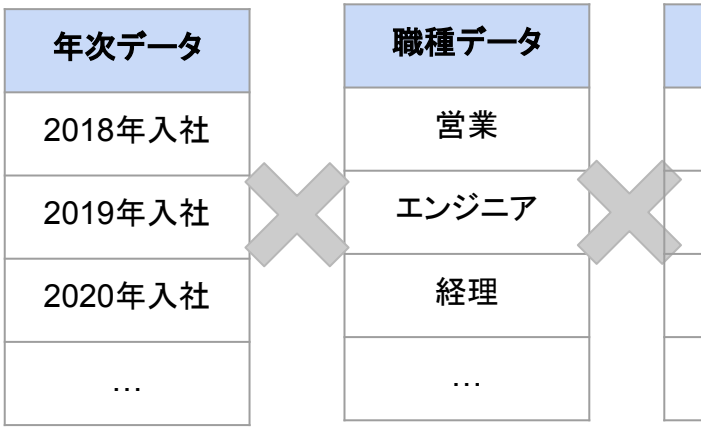
Case 2. 負荷アラート



背景

一部のデータが肥大化すると、
改修では間に合わないボトルネックが生まれる！

実際に発生し、データを削除頂くケースも



打ち手

[暫定]データの量を監視し、最低限対応する

[恒久]リアルタイム算出の実装で負荷の抜本低減

打ち手

愚直に全てを監視するのではなく、
その先を見据えたコスパの良い監視にとどめる

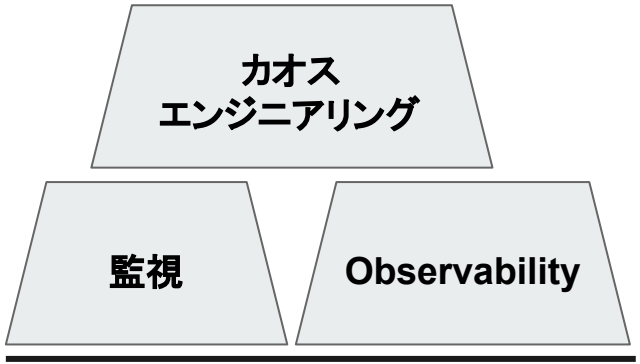
=> 監視が必要なリスクの排除を優先

[暫定]データの量を監視し、最低限対応する

[恒久]リアルタイム算出の実装で負荷の抜本低減

まとめ

監視をSREが持っている場合



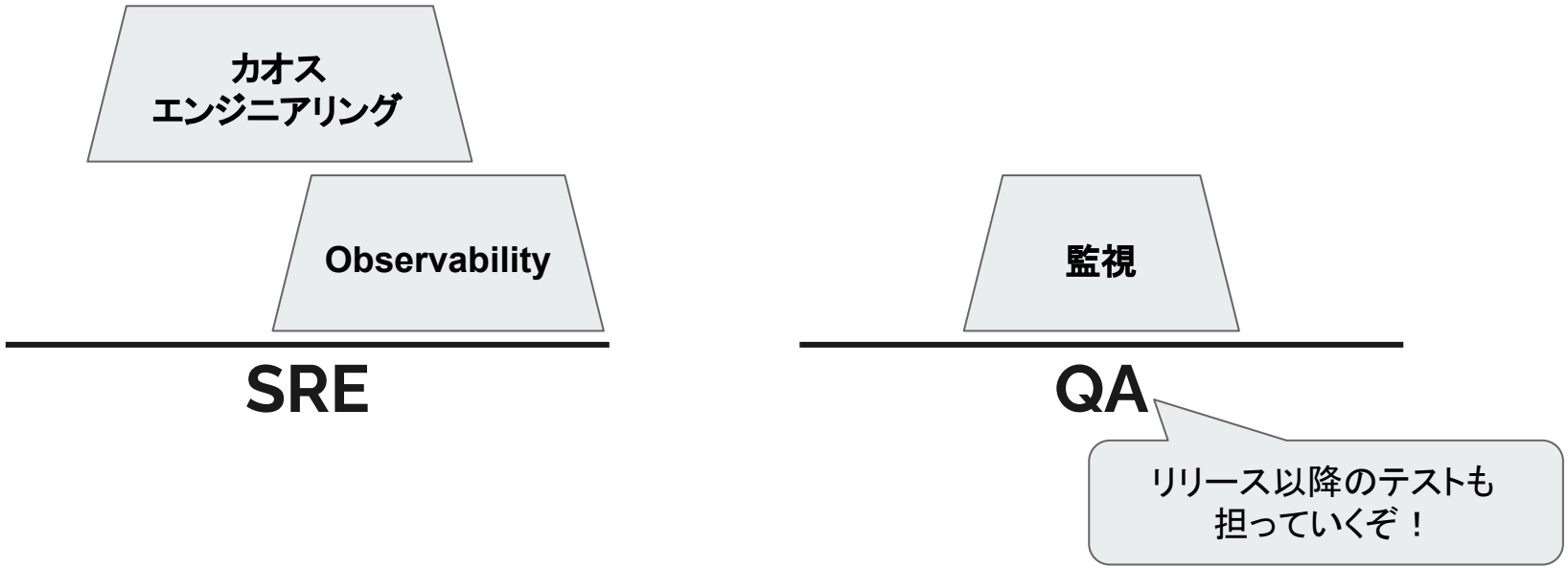
SRE

SREの勉強もしなきゃ💦

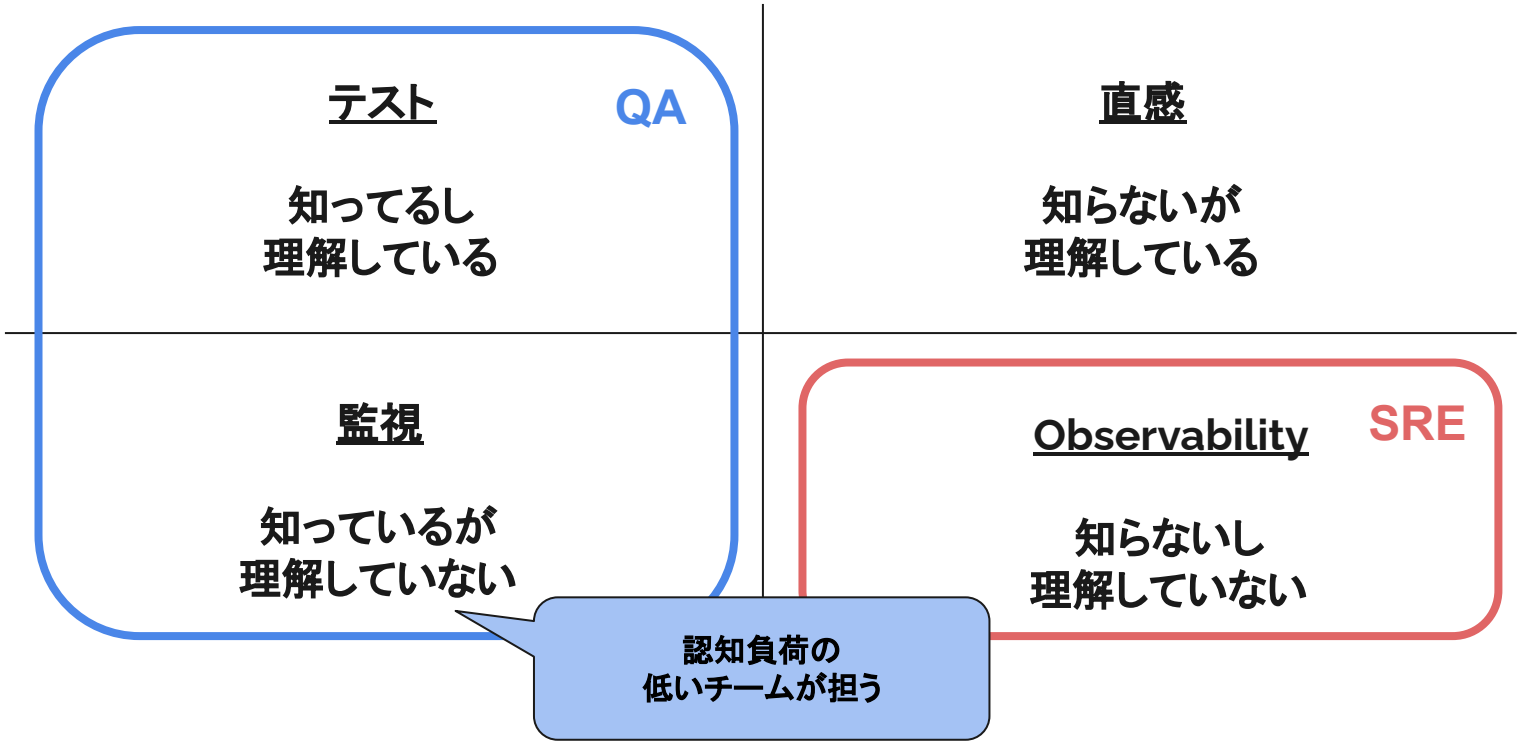
次の開発進めなきゃ！

QA

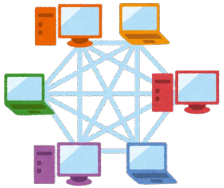
結論: 監視はQA(開発チーム)で持とう！



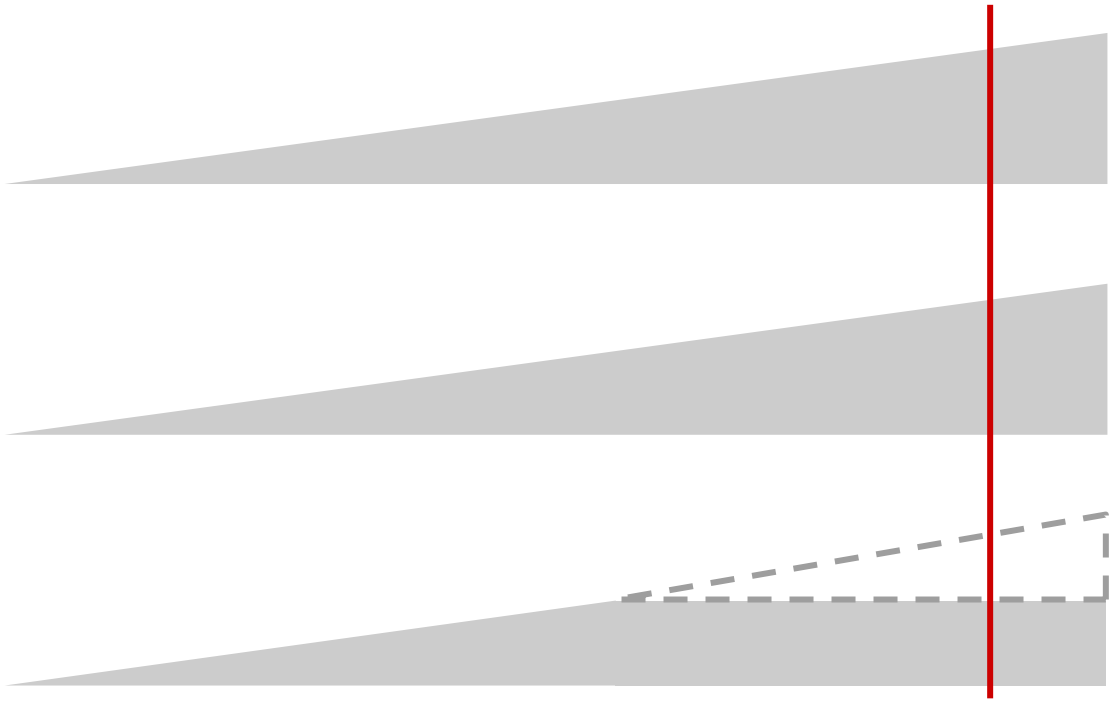
理想: QAはObservabilityの前に監視をやろう！



システムの複雑化は、組織の複雑化と同時に起きる



SRE/Opsの
認知負荷



その先に、

Observabilityやカオスエンジニアリングが

待っている！

Today's Message:

プロダクトが複雑化すれば、組織も複雑化する
組織が複雑化すれば、「既知」を扱う監視はチームに依存する

QAはまず監視こそ学び、
チーム全体がObservabilityを実現できる体制を後押ししよう

最後に

実現したいのは
「顧客に安定・迅速・確実に価値を届けること」です

監視をすることも、
Observableなプロダクトをつくることも、
その1つの手段でしかありません