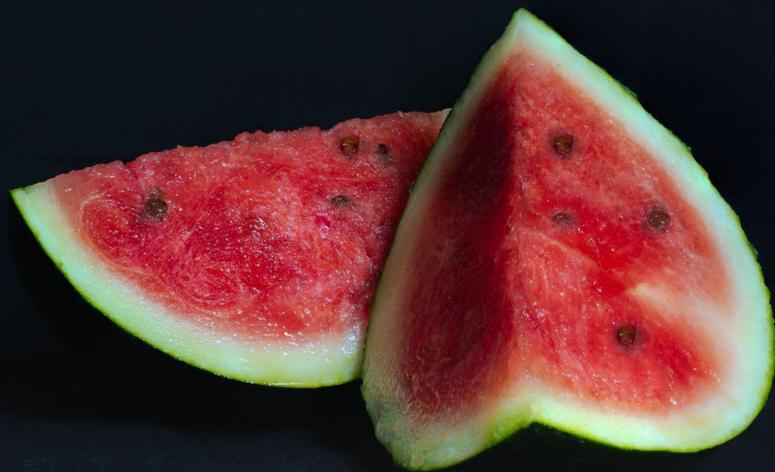


そのプロダクト、 「正常に動いてます！」 と言えますか？

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?

叩

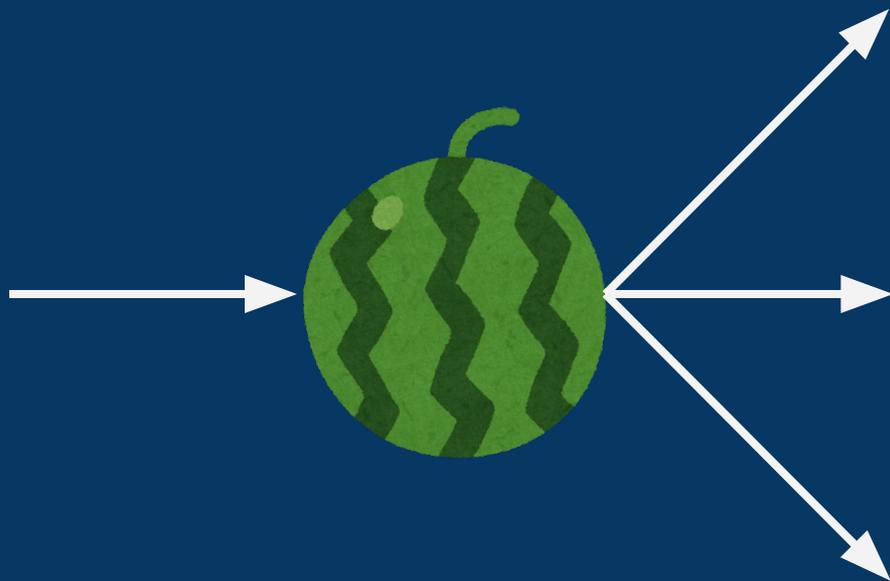


高音

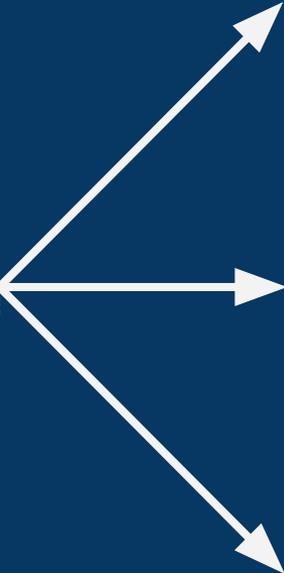
中音

低音

叩



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？

表面の感触

ツルの色形

縞模様

音

サイズ



形



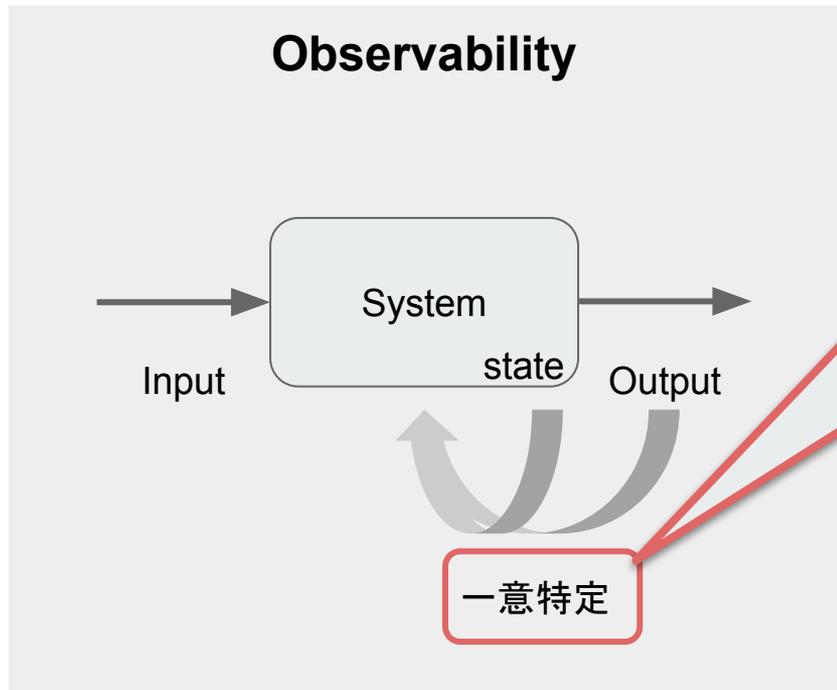
スイカの食べ頃は



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

Observability

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...?

Input

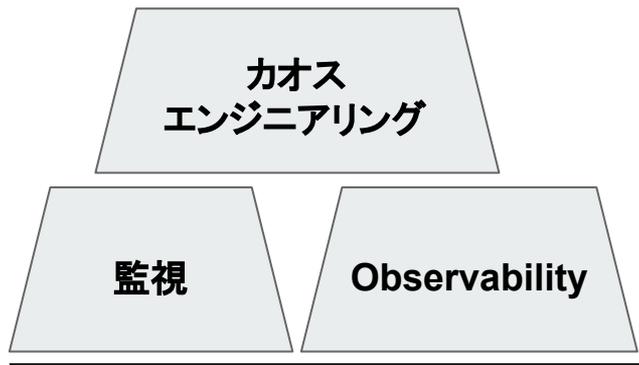
Output

一意特定

あなたのプロダクトは、
完全停止しても原因を特定できますか？



今日のSRE



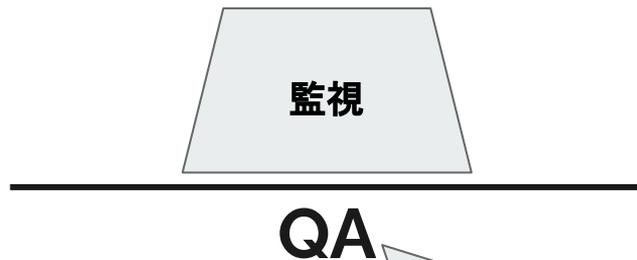
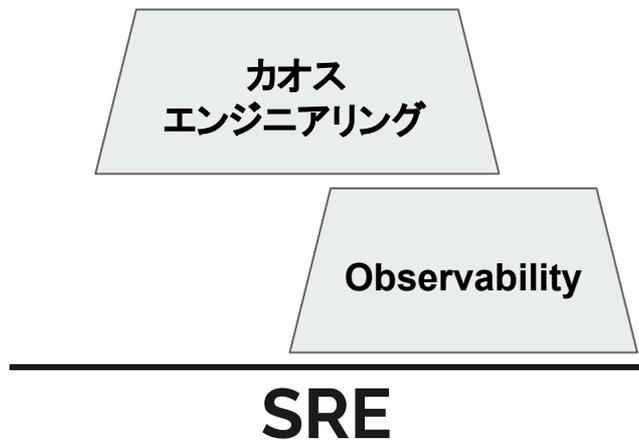
SRE

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!



リリース以降のテストも
担っていくぞ!

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

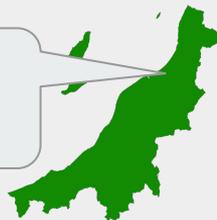
SREユニット QAエンジニア

SigSQA

QAファンネルなど

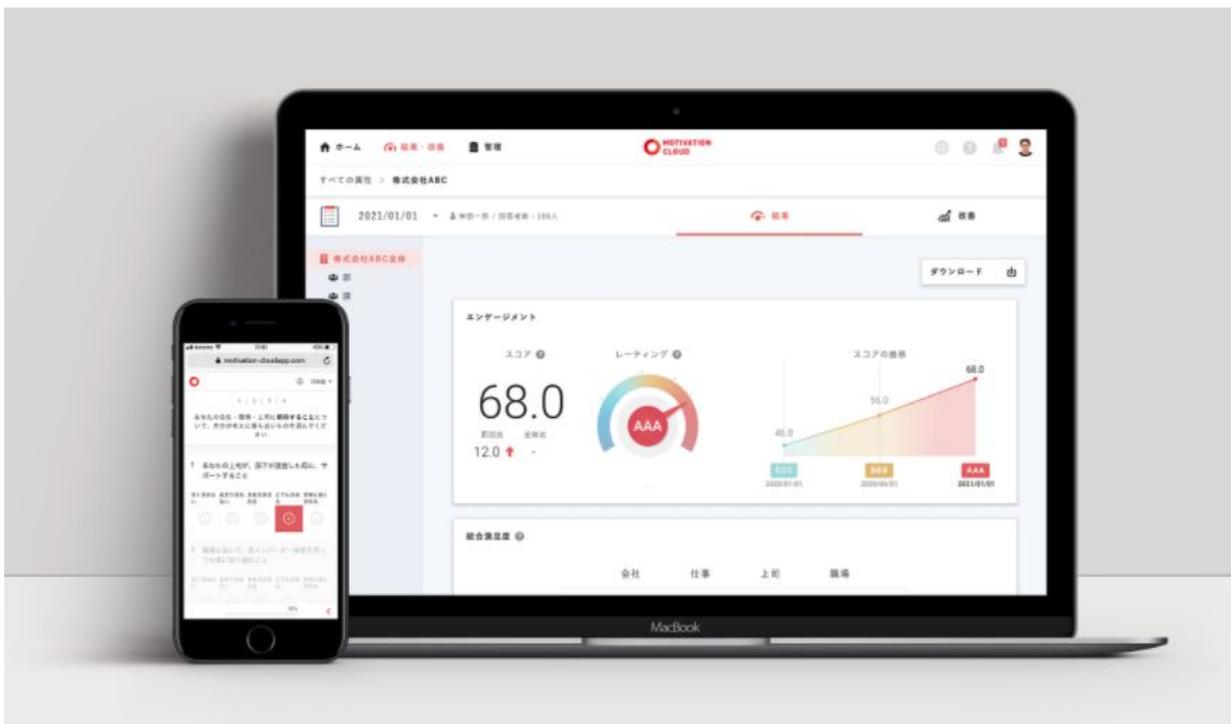
新潟要素:

祖父が
新発田市出身!



趣味:



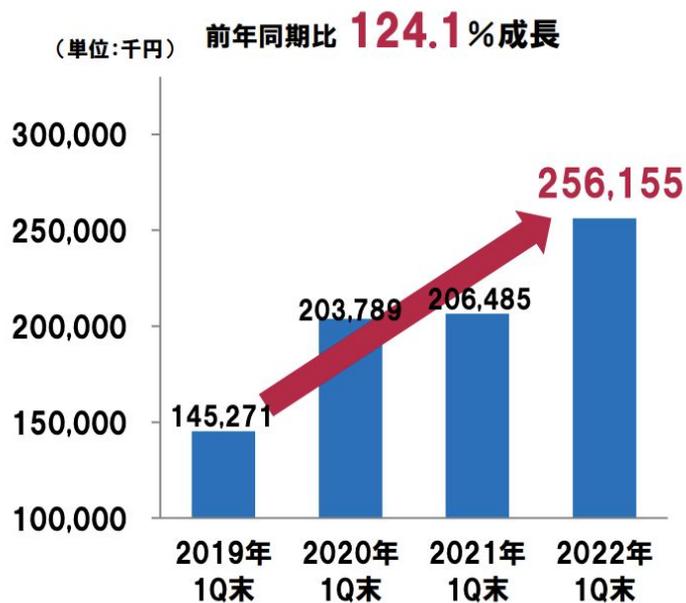


 **MOTIVATION
CLOUD**
by Link and Motivation Group

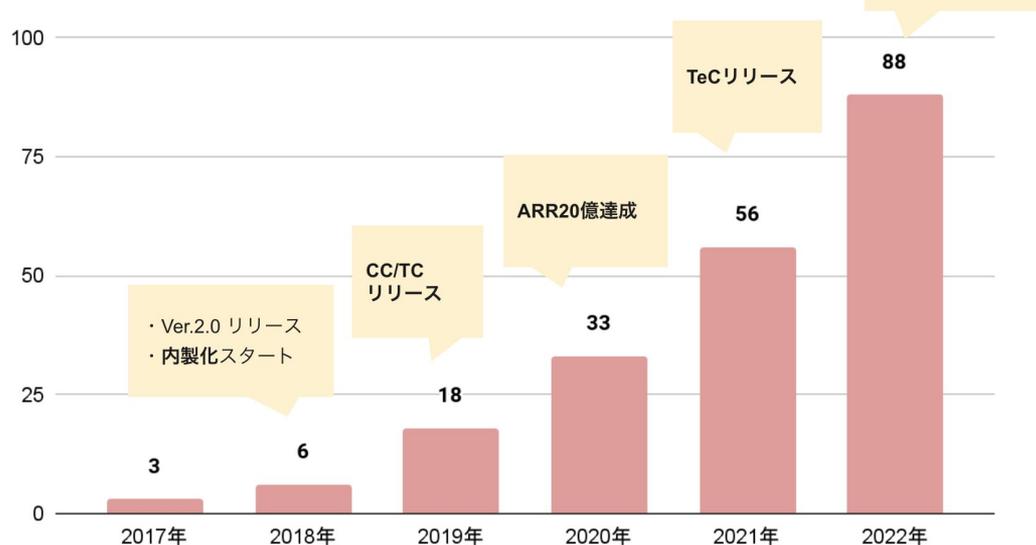
組織もObservableにしていきたい！

 **MOTIVATION
CLOUD**
by Link and Motivation Group

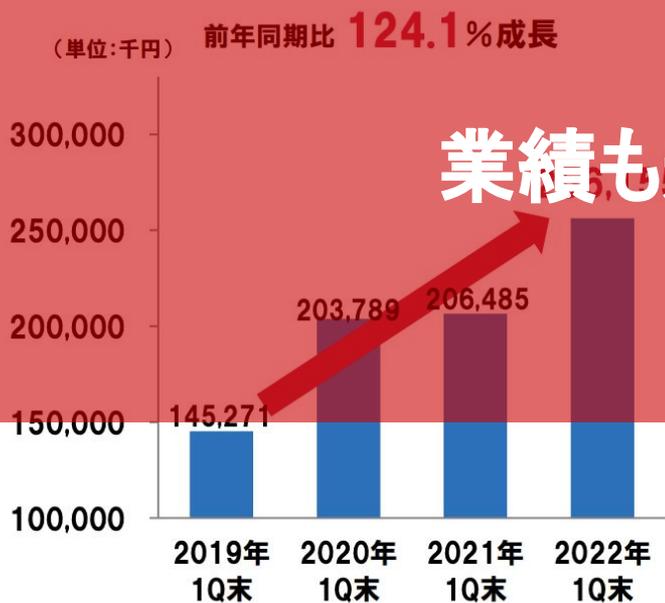
モチベーションクラウドシリーズ 月会費売上



開発組織 人員推移

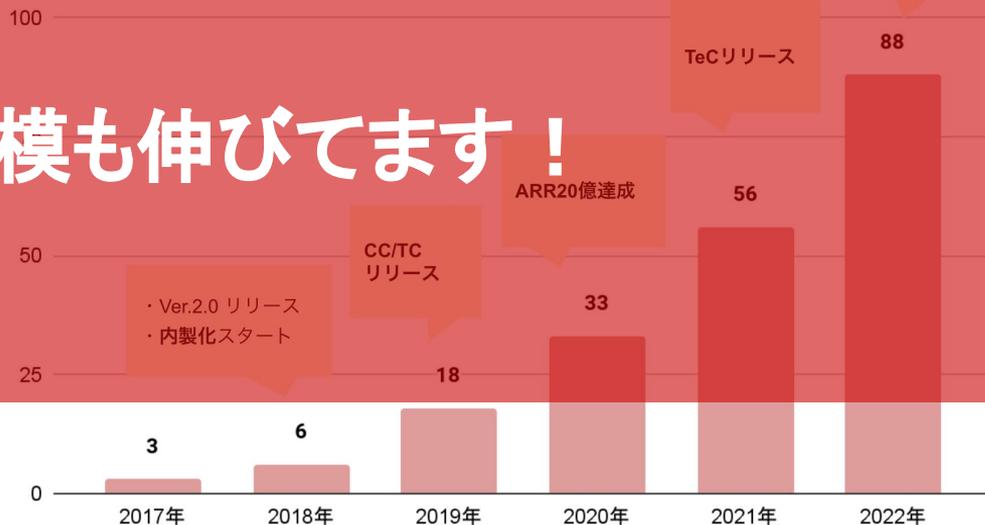


モチベーションクラウドシリーズ 月会費売上



業績も規模も伸びています！

開発組織 人員推移



本題はここから！

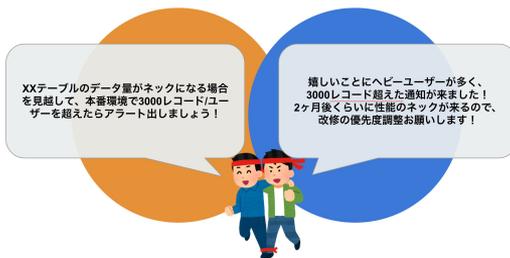
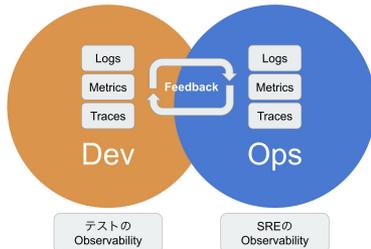


Agenda

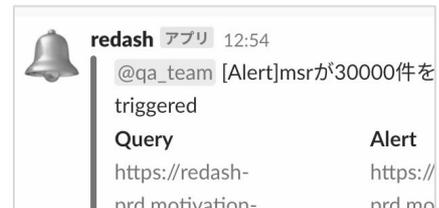
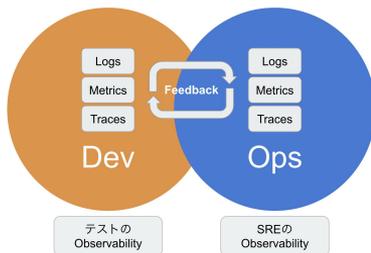
概念

理想

事例



Agenda





Observability?

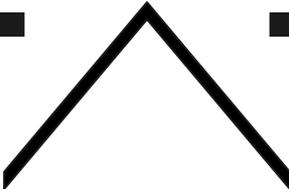
色々な視点がありそうです



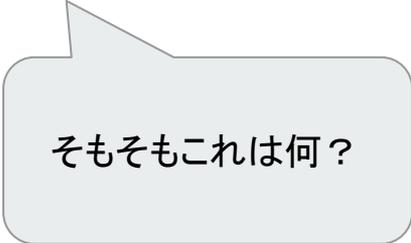


**SRE視点の
Observability**

**QA視点の
Observability**



GAP?

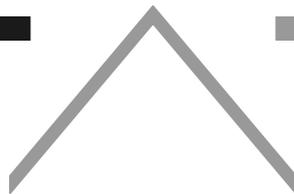


そもそもこれは何？



**SRE視点の
Observability**

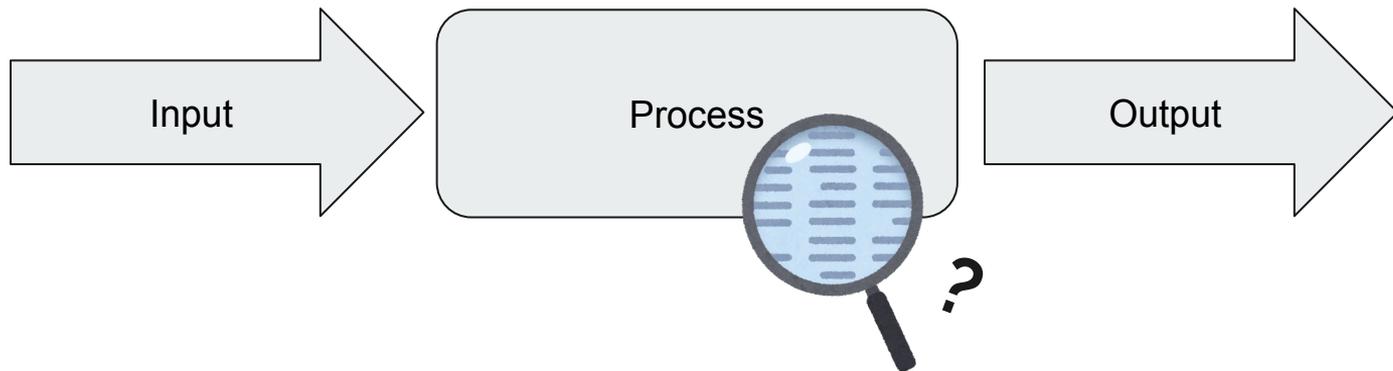
**QA視点の
Observability**



GAP?

そもそもこれは何？

SRE視点のObservability (ここでは詳細省略！)

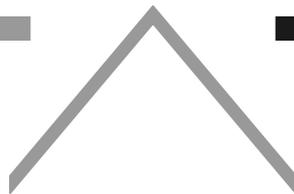


- 大規模かつ複雑化するシステムは、障害要因を事前推測するのに限界がある
 - 有事に原因を特定できる仕組みが重要に



SRE視点の
Observability

QA視点の
Observability

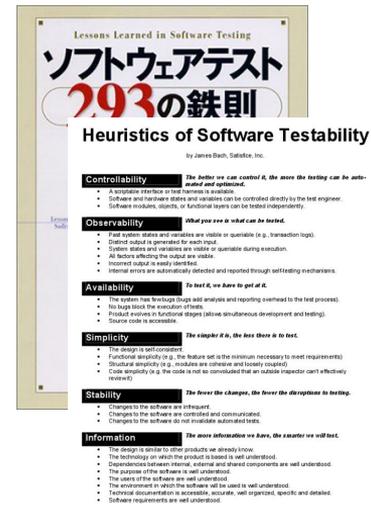


GAP?

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

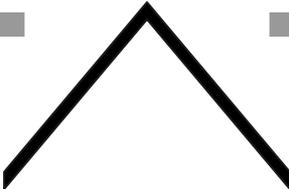




SRE視点の
Observability



テスト視点の
Observability



GAP?

概念まとめ

	SRE	QA
Observability	<u>可</u> 観測性	観測 <u>容</u> 易性
Controllability	<u>可</u> 制御性	制御 <u>容</u> 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

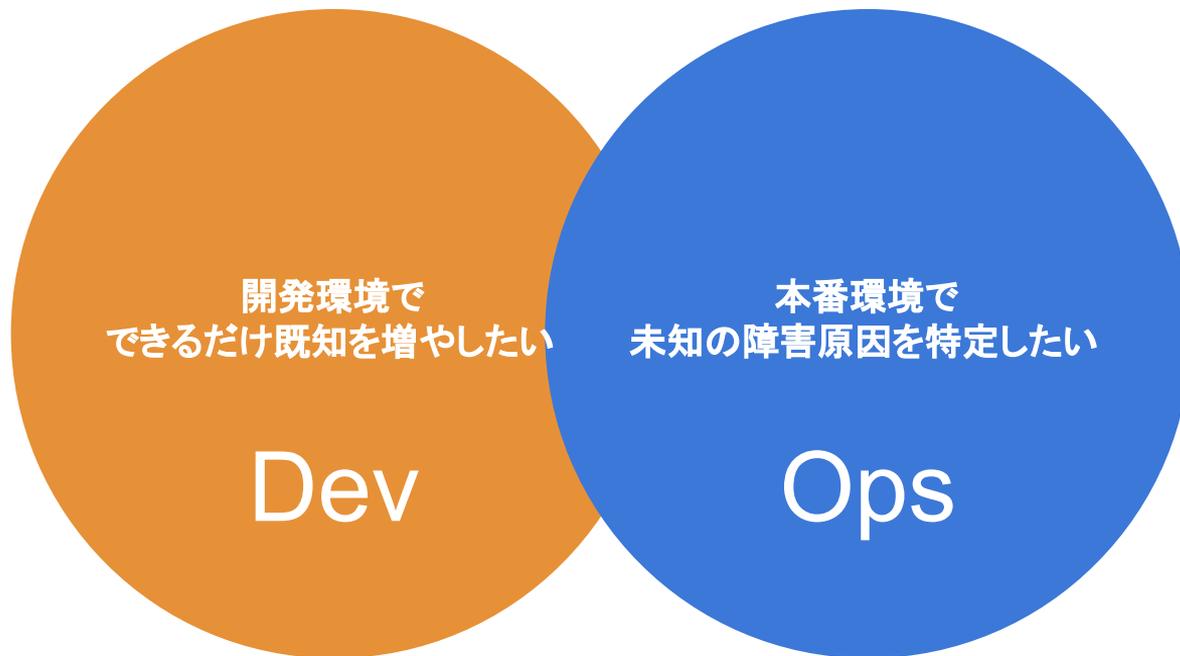
観測/制御できる状態を**提供する**

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**

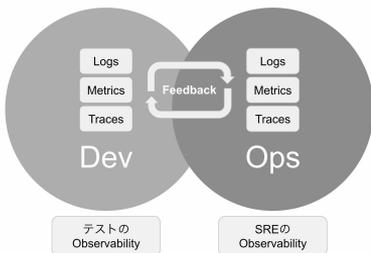
まとめると、



QA視点の
Observability

SRE視点の
Observability

Agenda



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しましょう！

嬉しいことにヘビーユーザーが多く、
3000レコードを超えた通知が来ました！
2ヶ月後くらいに性能のネックが来るので、
改修の優先度調整お願いします！

redash アプリ 12:54

@qa_team [Alert]msrが30000件を
triggered

Query Alert

https://redash- https://

prd.motivation- prd.mo

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

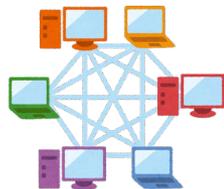
システムの複雑化は、組織の複雑化と同時に起きる



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

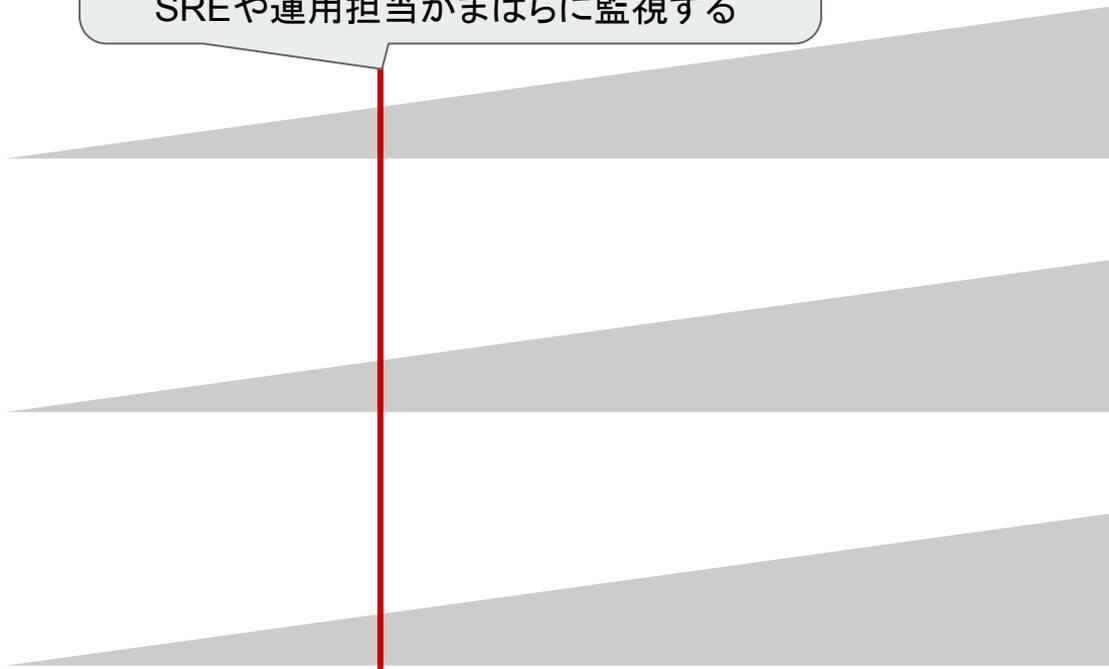


システムの複雑化は、組織の複雑化と同時に起きる

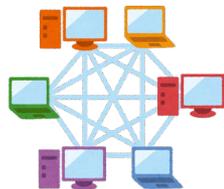


SRE/Opsの
認知負荷

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

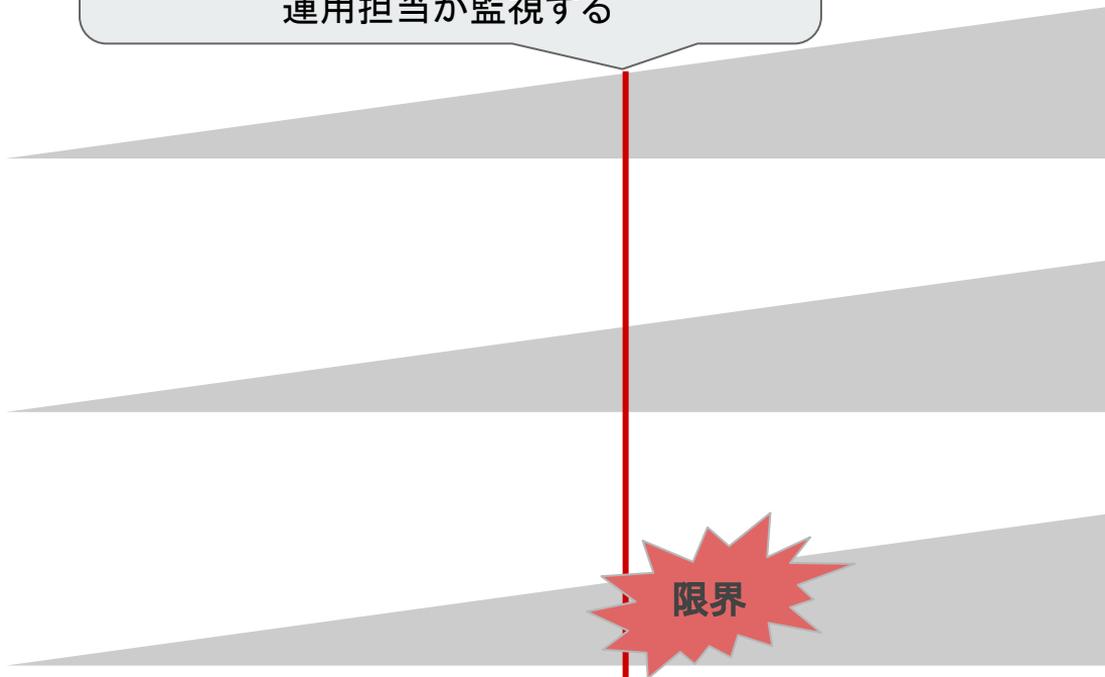


システムの複雑化は、組織の複雑化と同時に起きる



SRE/Opsの
認知負荷

3プロダクトと基盤
横軸と各プロダクトにSREを配置
運用担当が監視する

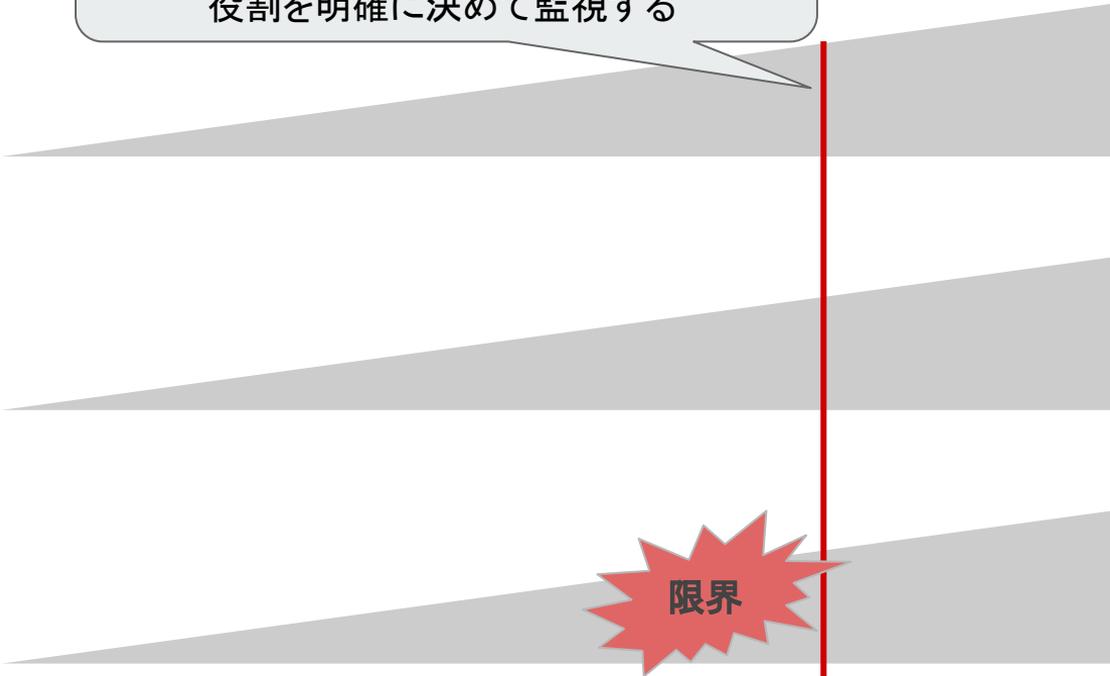


システムの複雑化は、組織の複雑化と同時に起きる



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



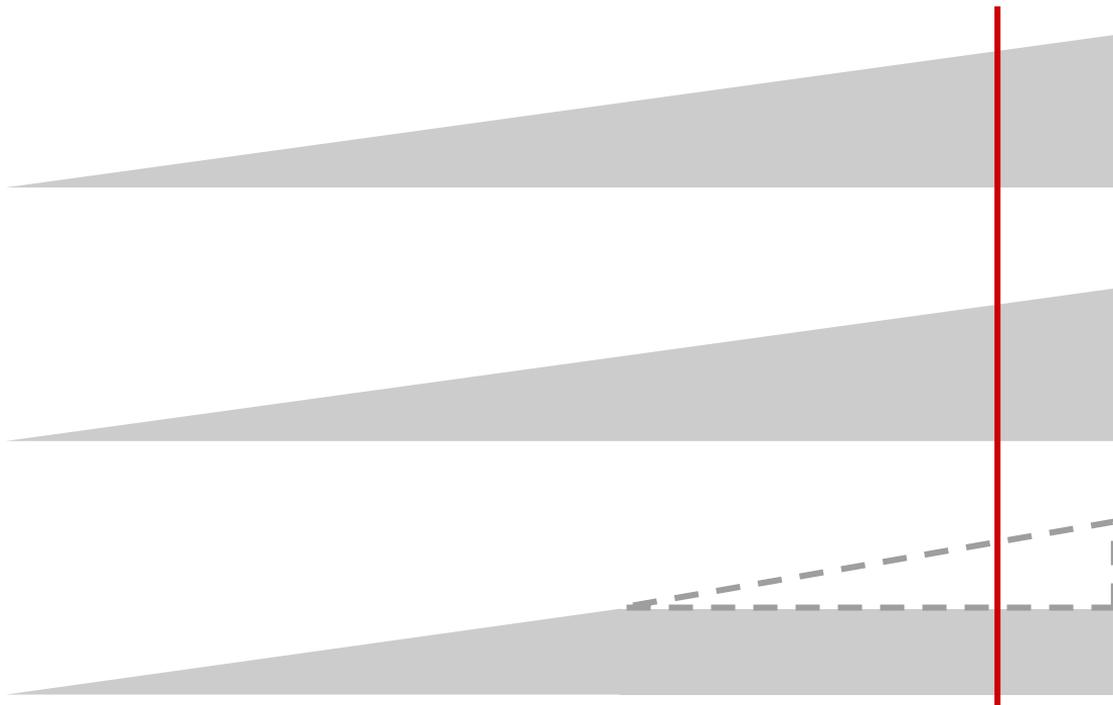
理想



理想: 規模拡大により認知負荷が高まらない組織



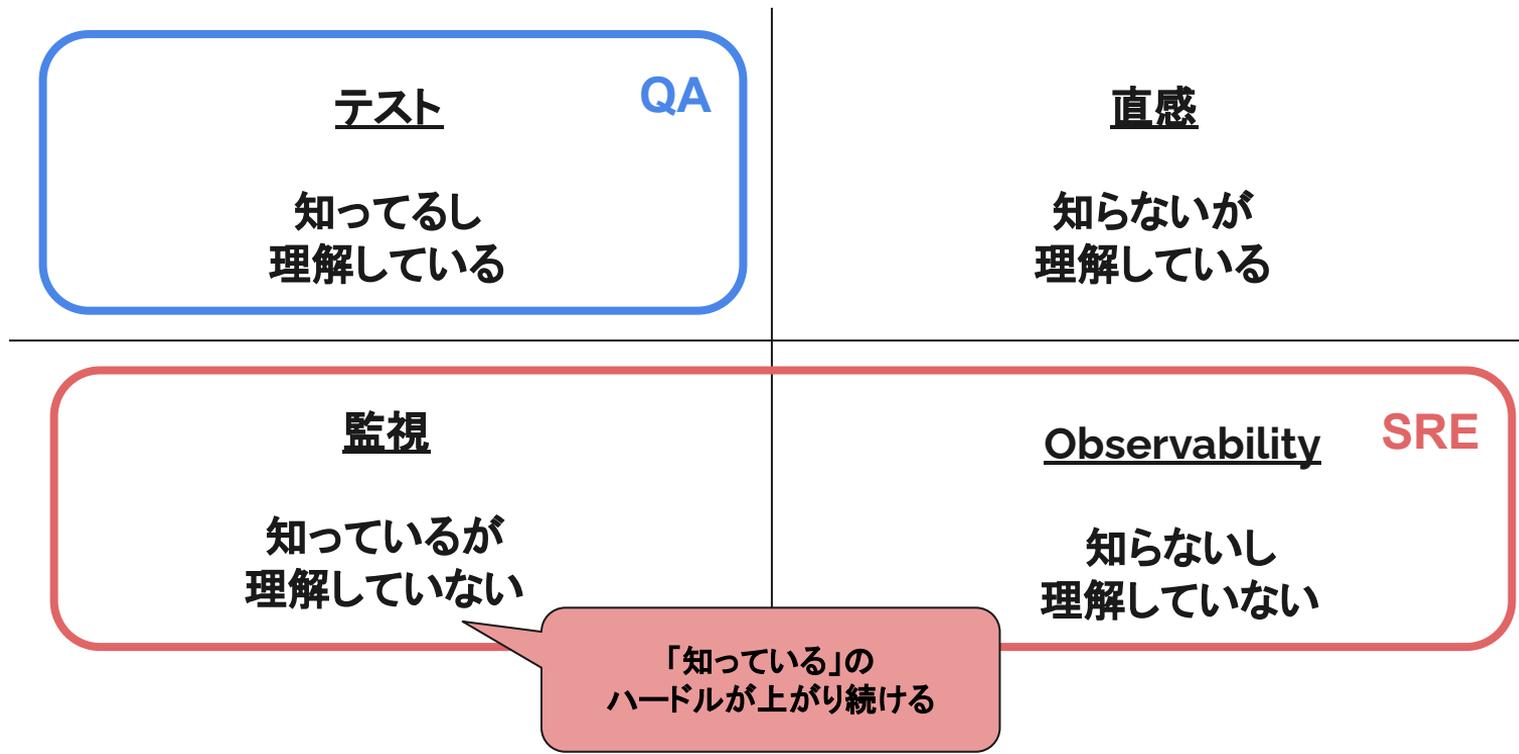
SRE/Opsの
認知負荷



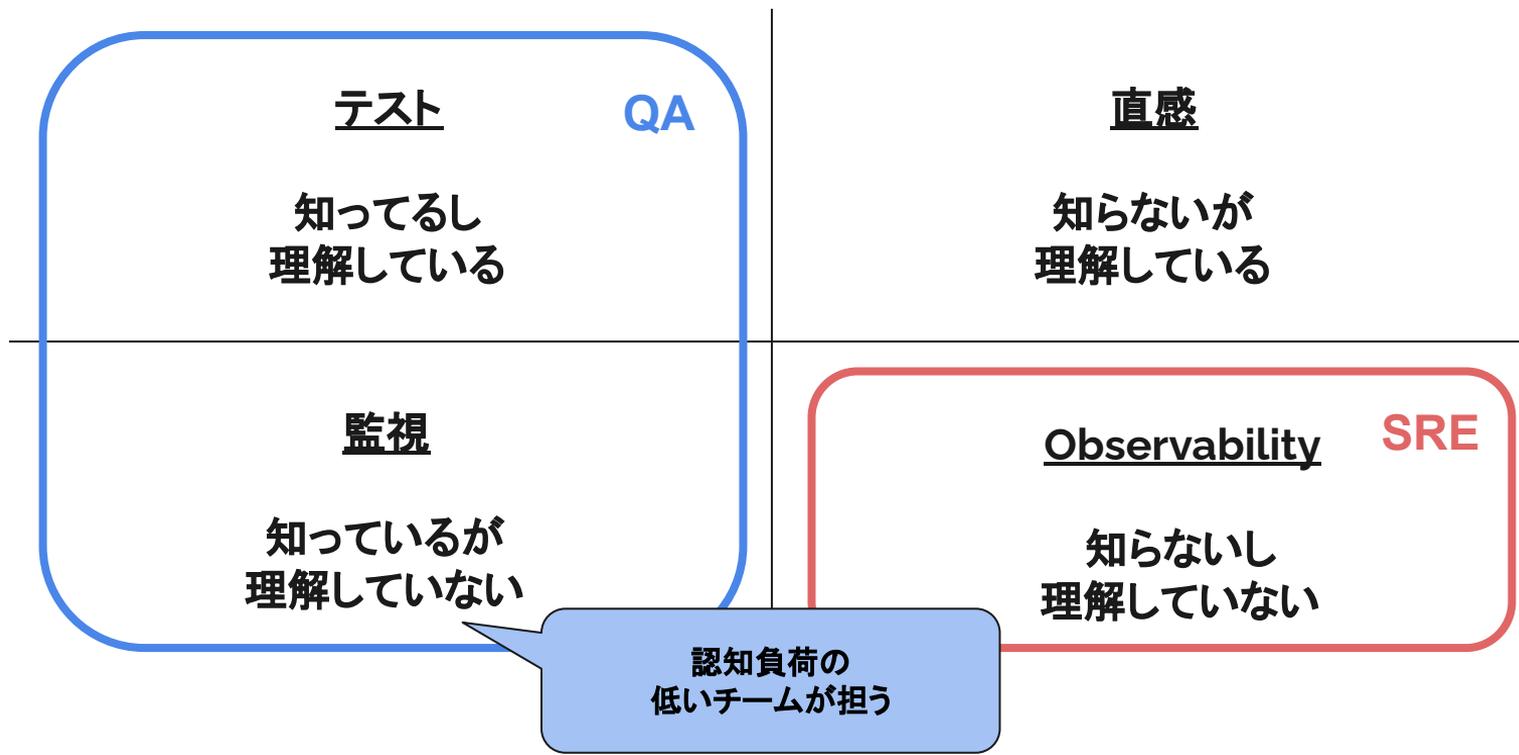
何が限界を生むのか？



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
ハードルが上がると

実際は、いろいろなものが間に落ちていく

開発環境で
できるだけ既知を増やしたい

Dev

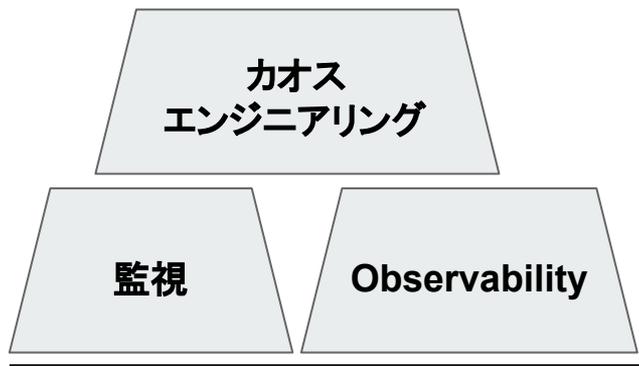
QA視点の
Observability

本番環境で
未知の障害原因を特定したい

Ops

SRE視点の
Observability

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)

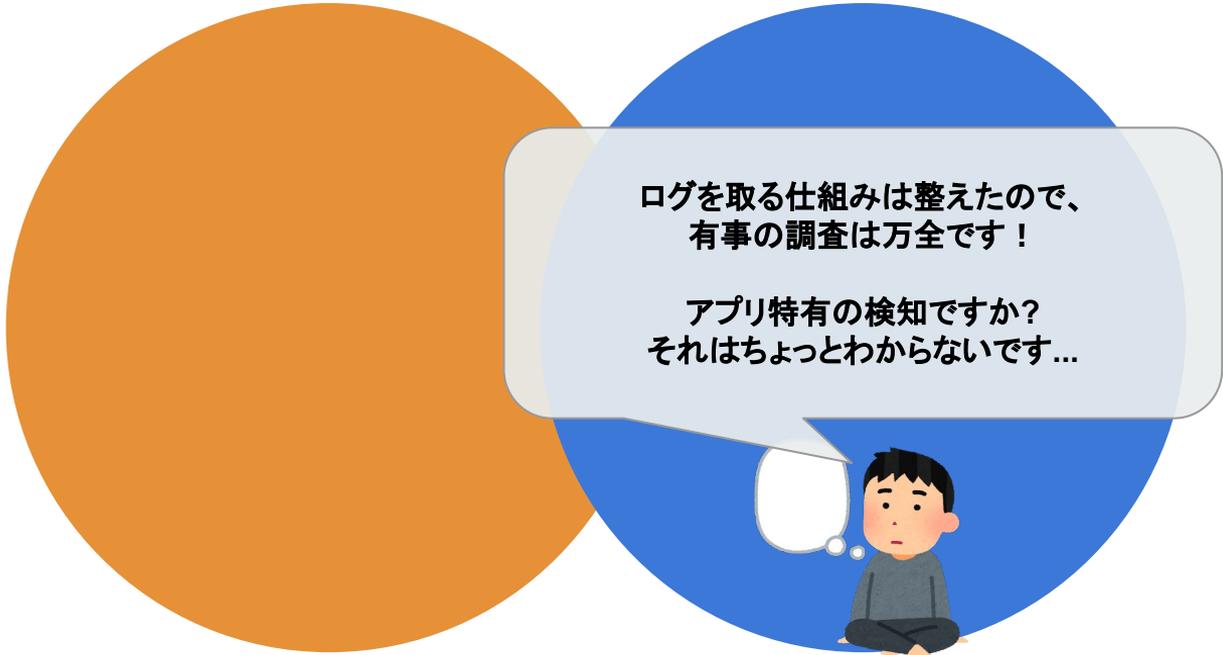
特定のデータ量が性能の
ネックだと分かったけれど、
限界がくる前に検知できないなあ

修正コストが高いから、
余裕ができたら対応するか...



QA視点

SRExQA連携できないケース(実例)



ログを取る仕組みは整えたので、
有事の調査は万全です！

アプリ特有の検知ですか？
それはちょっとわかりません...

SRE視点

SRExQA連携できないケース(実例)

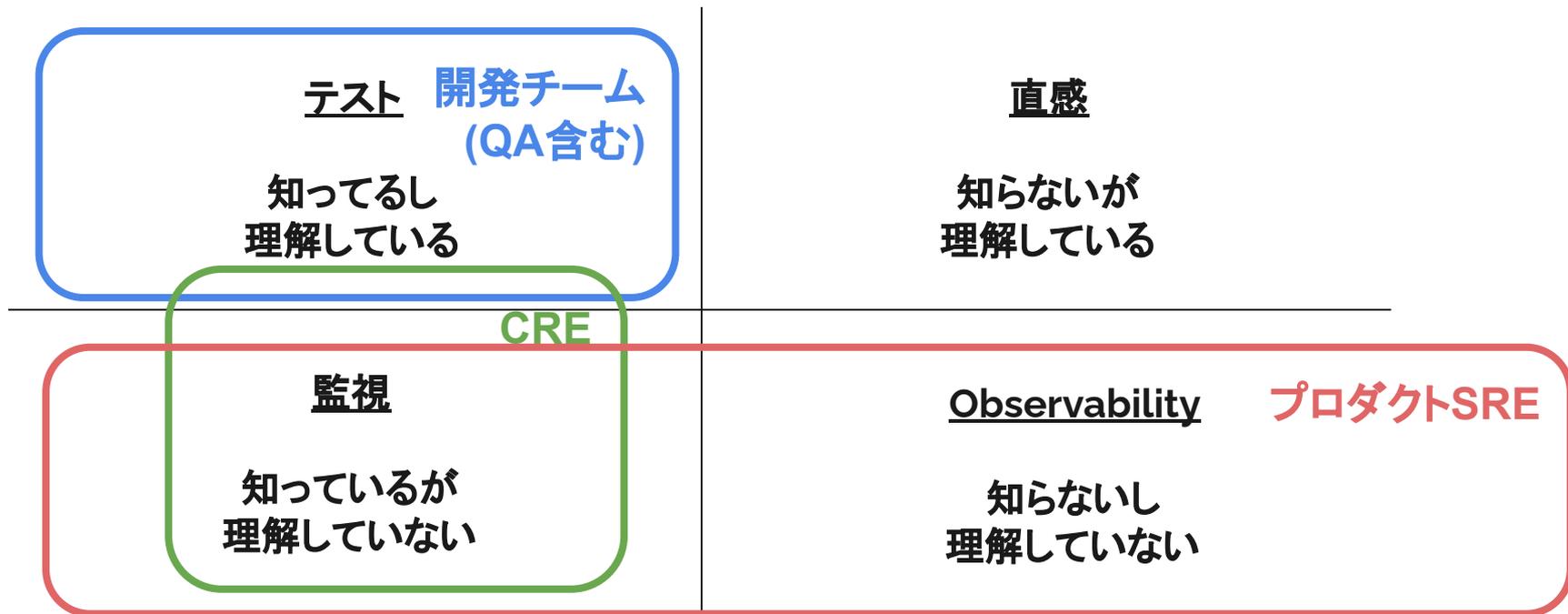


障害発生！
データ量がネックでサーバーが落ちてる！！
でも勝手に顧客データを消すわけには行かないし...
どうやらスケールアップしても限界がありそう。
SREのタスクは後ろ倒しにしないきゃ...

うっ...復旧には、1週間かかる改修、もしくは謝罪とデータ削除か...



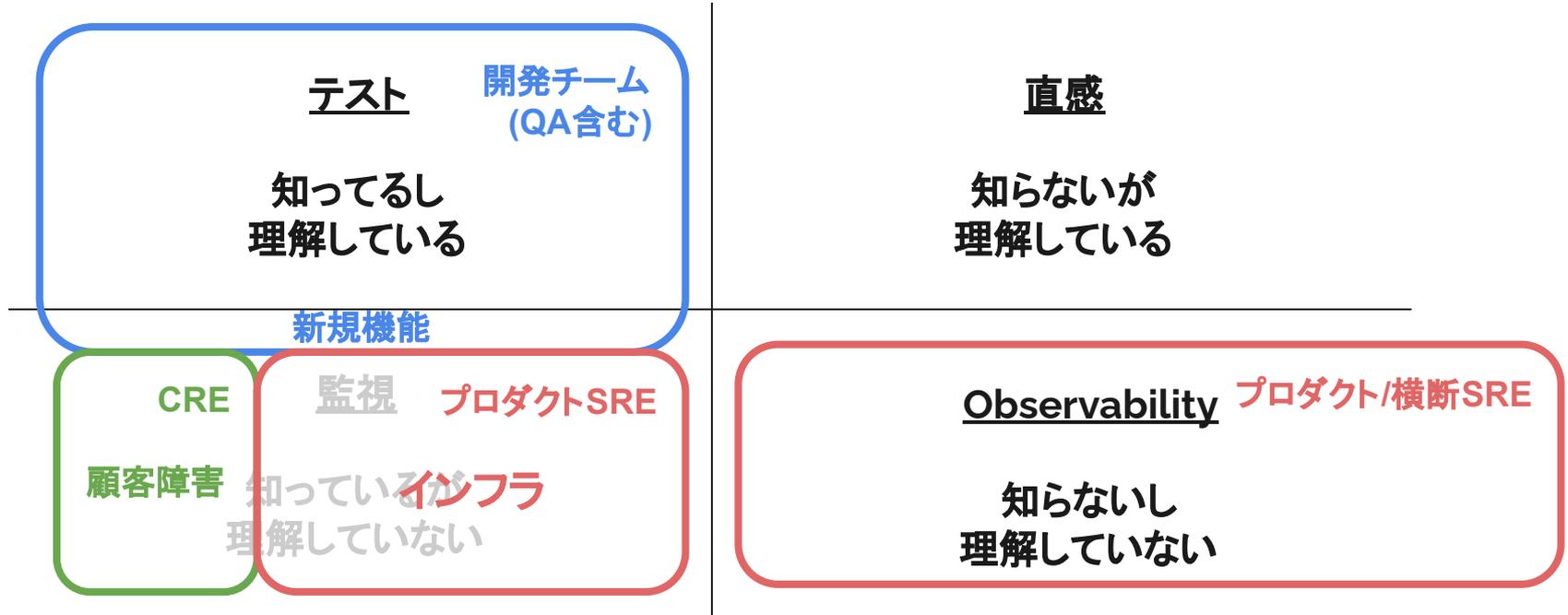
LMの場合: 1年前くらい



LMの場合: 1年前くらい



LMの場合: 現在



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

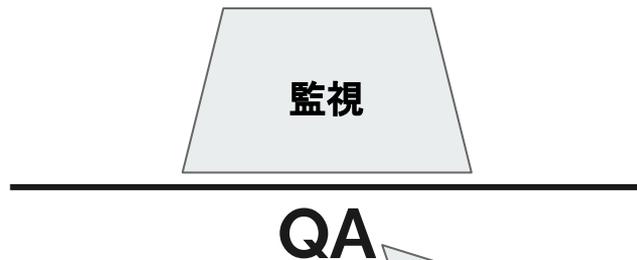
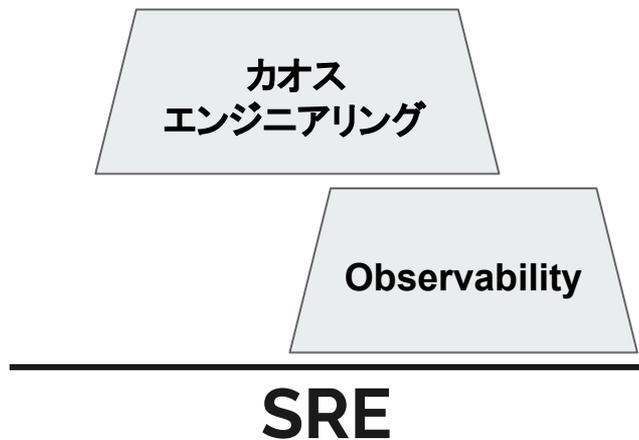
XXテーブルのデータ量がネックになる場合を見越して、本番環境で3000レコード/ユーザーを超えたらアラート出すようにします！

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!



リリース以降のテストも
担っていくぞ!

「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)

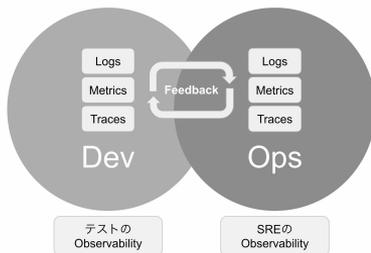
- 特定のパターンで障害が発生しやすいとわかっているもの
- ユーザーにより、正常/異常が変わりうるもの(負荷・性能など)
- 外部要因により、正常/異常が変わりうるもの(外部ツールなど)
- 開発環境のテストビリティ

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、以下のどちらがが必要

- 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



「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

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

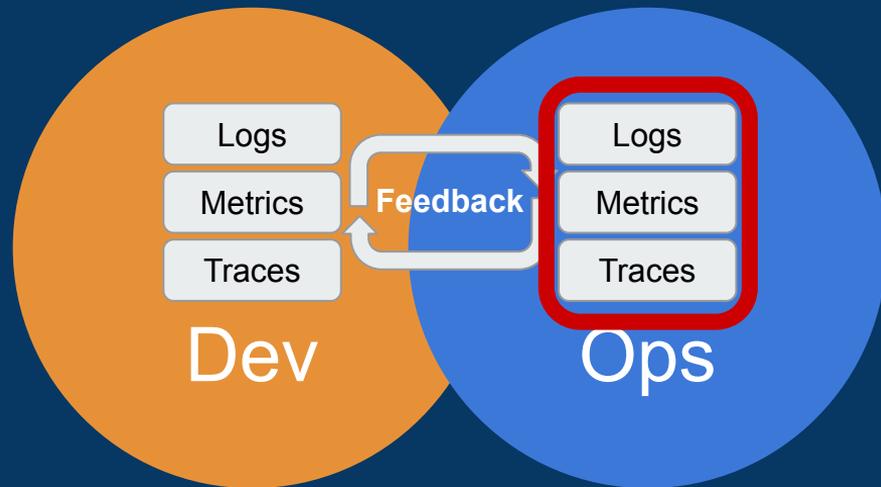
「知っている」の
ハードルが上がると

どれだけ事例を知っても、間に落ちていく

ただし、大事なことは事例数ではなく
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

モチベーションクラウドシリーズ※ 月会費売上

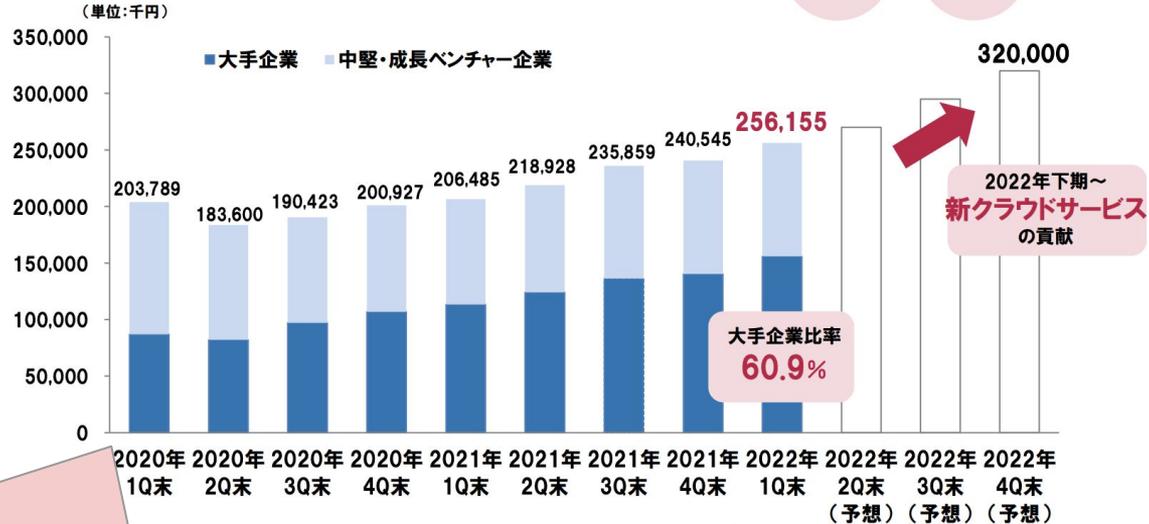
2022年1Q

実績

256,155千円

前Q比
106.5%

前年比
124.1%



エンタープライズ化するためには、性能改善が必要！

背景

モチベーションクラウドシリーズ※ 月会費売上

2022年1Q

実績

256,155千円

前Q比
106.5%

前年比
124.1%

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 アプリ 16:51

Recovered: [Production | APP]

api::v2::operations::magellansummaryresults::viewableusers::csvimportscontroller_upload has a high p99 latency

recovered

@slack-performance_warn

trace.rack.request.duration.by.resource_service.99p over env:eb-mcs-ver2-blue,resource_name:api::v2::operations::magellansummaryresults::viewableusers::csvimportscontroller_upload,service:mcs_ver2 was <= 17.0 on average during the last 5m.

Metric value: 0.264 (12 kB) ▾

Notified

@slack-performance_warn



DATADOG



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

機能名	設定値
api/v2/operations/magellan_viewable_users#csv_upload,1800.0	設定値1800s
api/v2/operations/magellan_viewable_users#recommend_csv_download,10.0	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

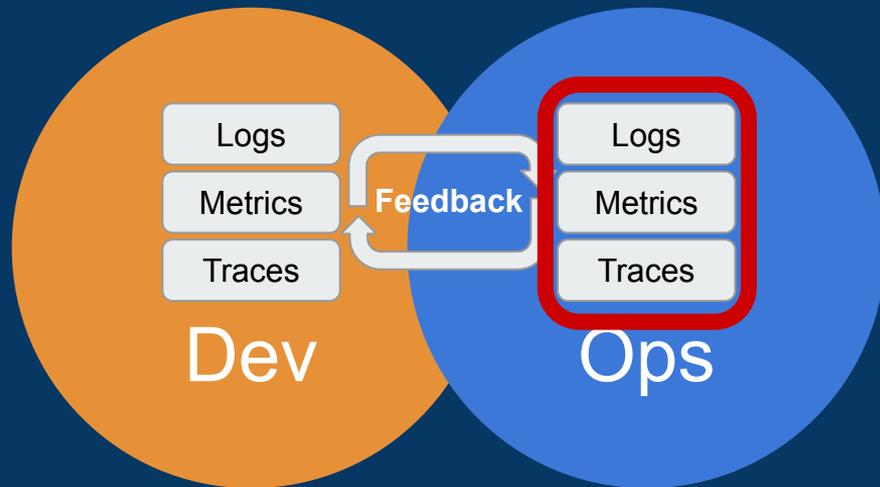
1. 性能ネックの理解 (性能改善pj)
2. 継続的な監視・改善体制 (CRE)
3. 監視に対する知識 (SRE)

=> 意志を持って役割設計すること
さもなければ間に落ちます



ネックになりうる各 API や 運用 関連サービスに 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！

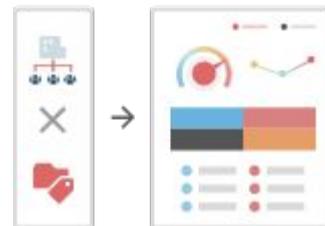
実際に発生し、データを削除頂くケースも

年次データ	職種データ	
2018年入社	営業	
2019年入社	エンジニア	
2020年入社	経理	
...	...	

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

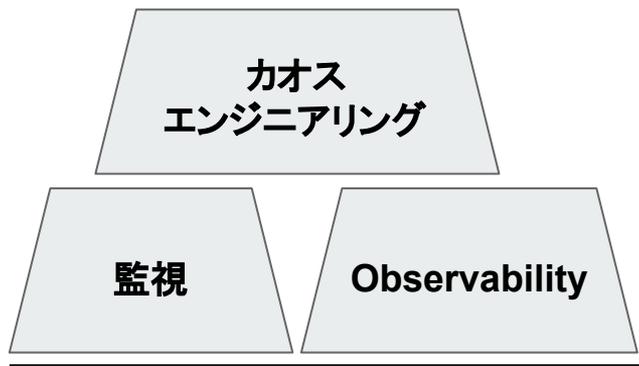
[暫定]データの量を監視し、最低限対応する

[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



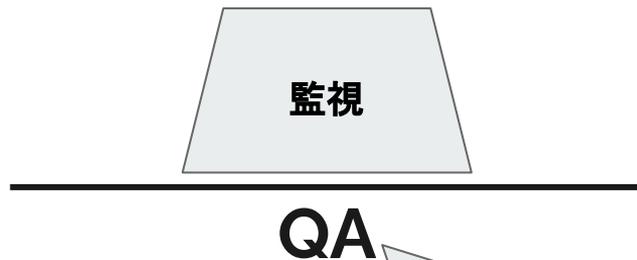
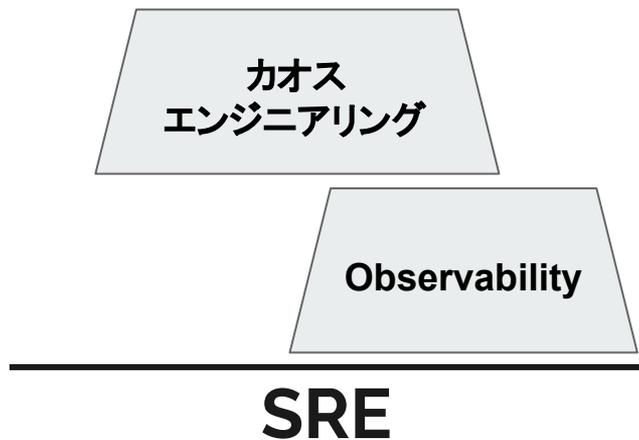
SRE

SREの勉強もしなきゃ💧

次の開発進めなきゃ！

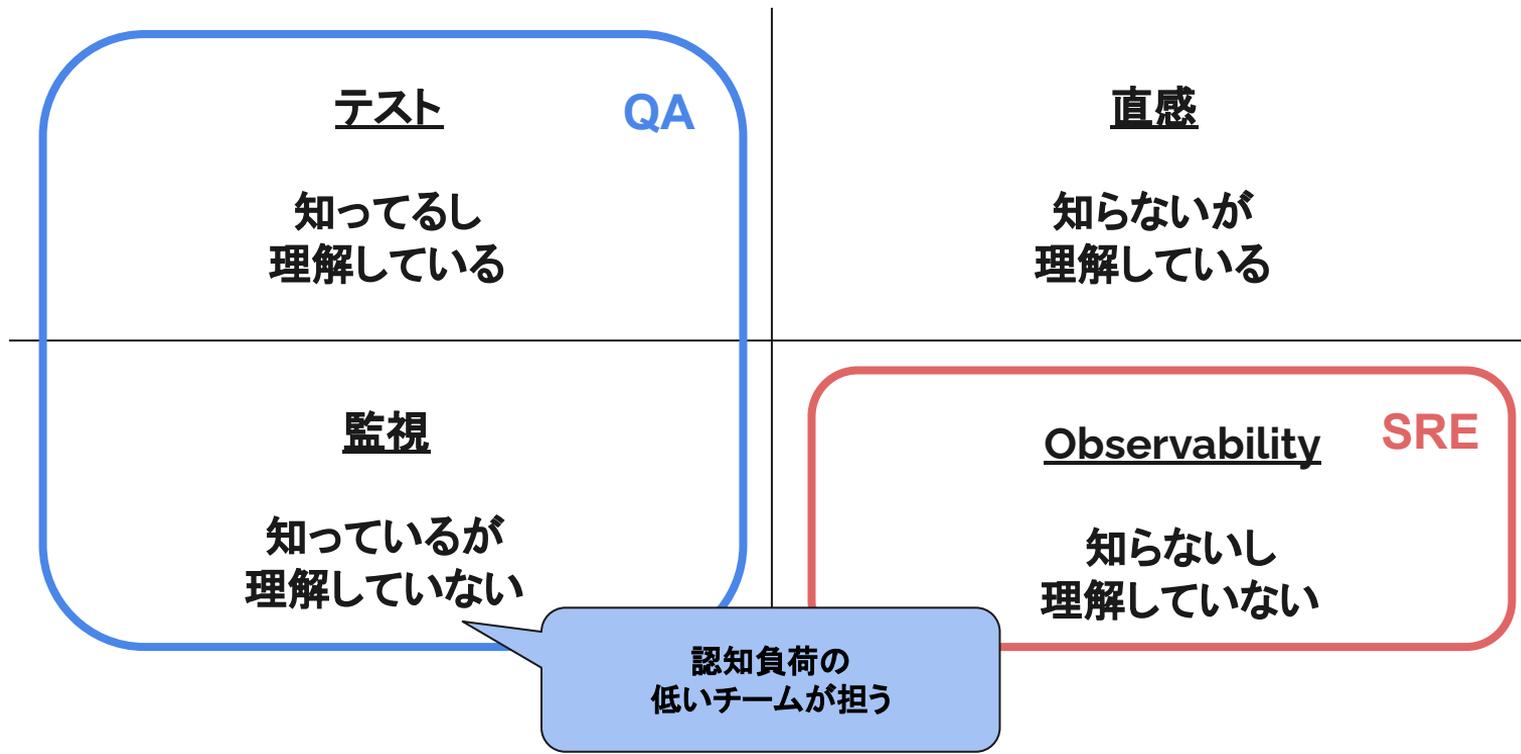
QA

結論: 監視はQA(開発チーム)で持とう!



リリース以降のテストも
担っていくぞ!

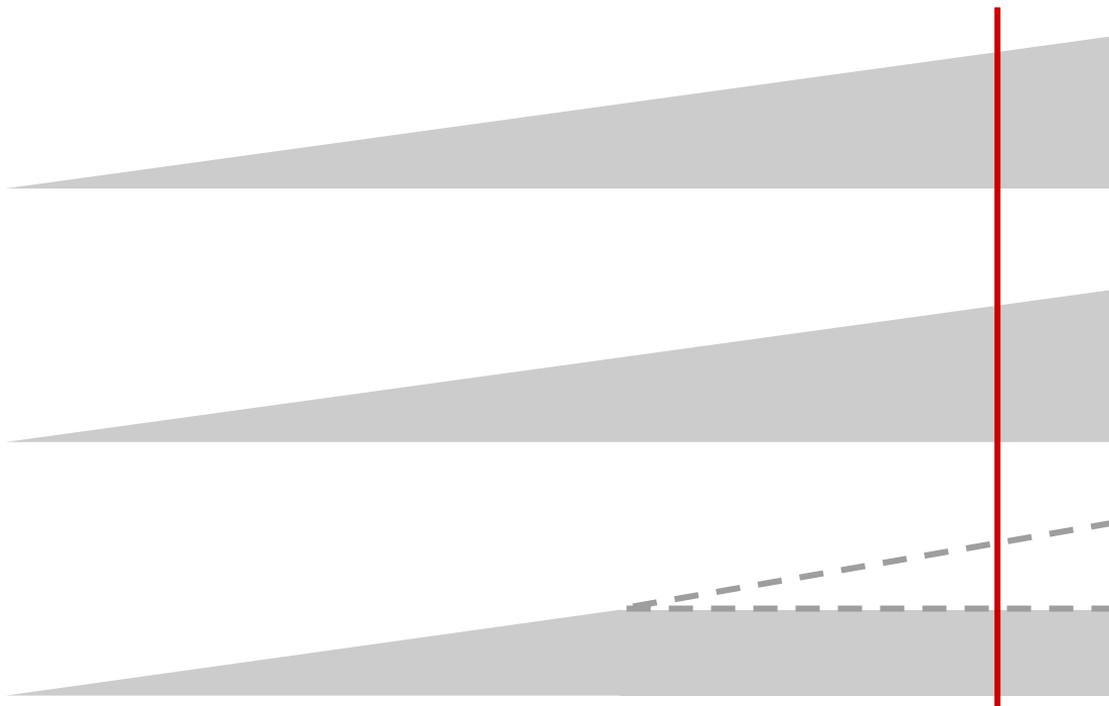
理想: QAはObservabilityの前に監視をやろう!



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