AWS オンラインセミナー

ちがいからみる Platform Engineering

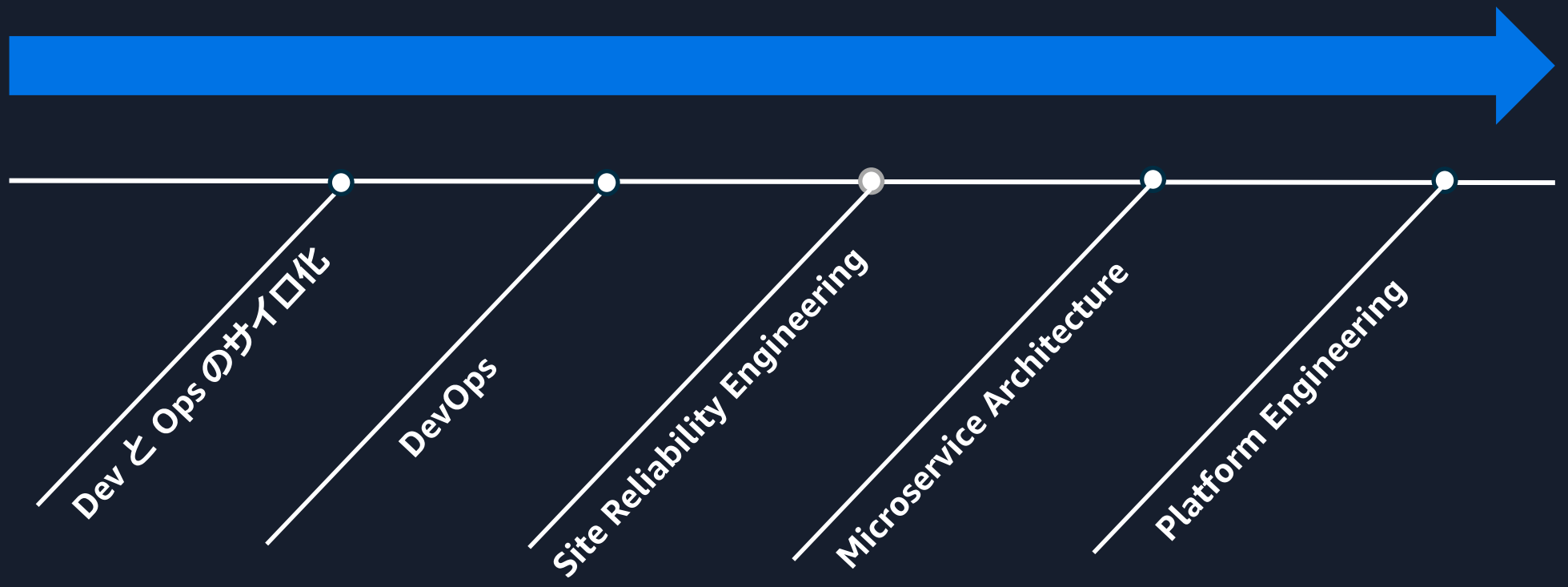
クラウドネイティブ化に伴って生じた新たなチームトポロジー

Ryota Yamada (riita@)

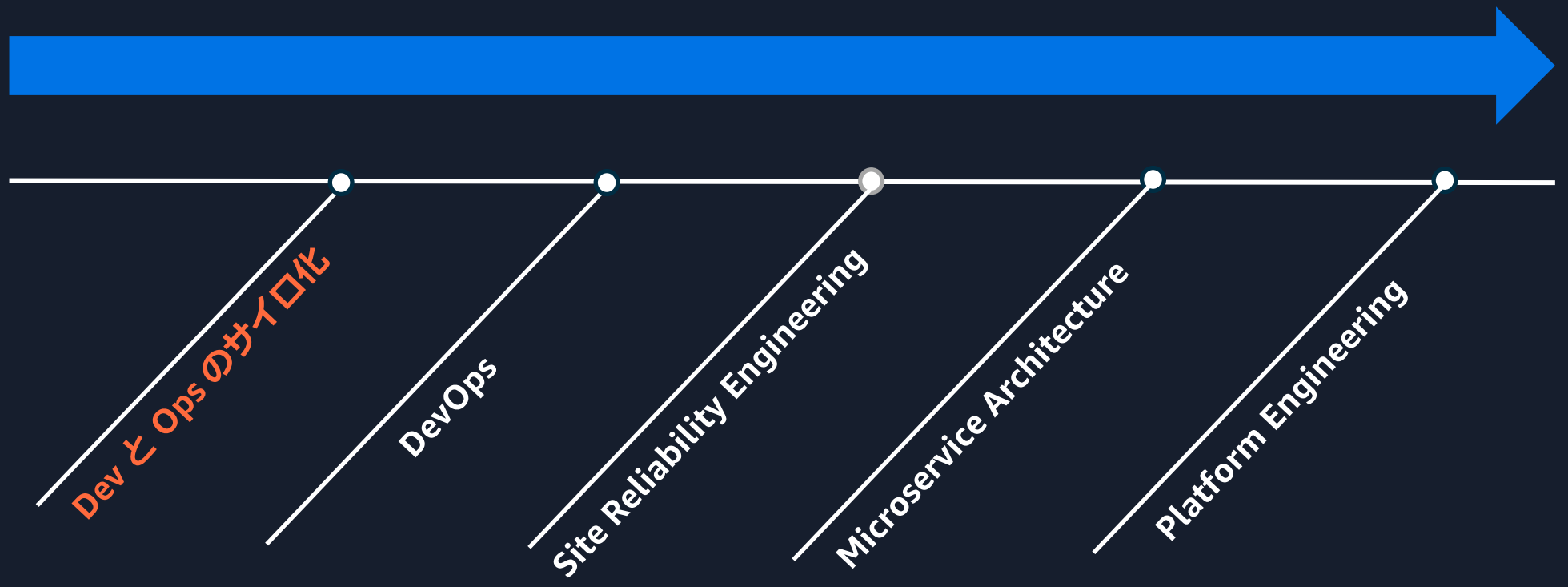
Amazon Web Services Japan G.K.

Platform Engineering に
どんなイメージをお持ちですか？

クラウドネイティブ化の背景



クラウドネイティブ化の背景



従来の組織体系には、歪みが生じている

開発チーム

運用チーム

従来の組織体系には、歪みが生じている

開発チーム

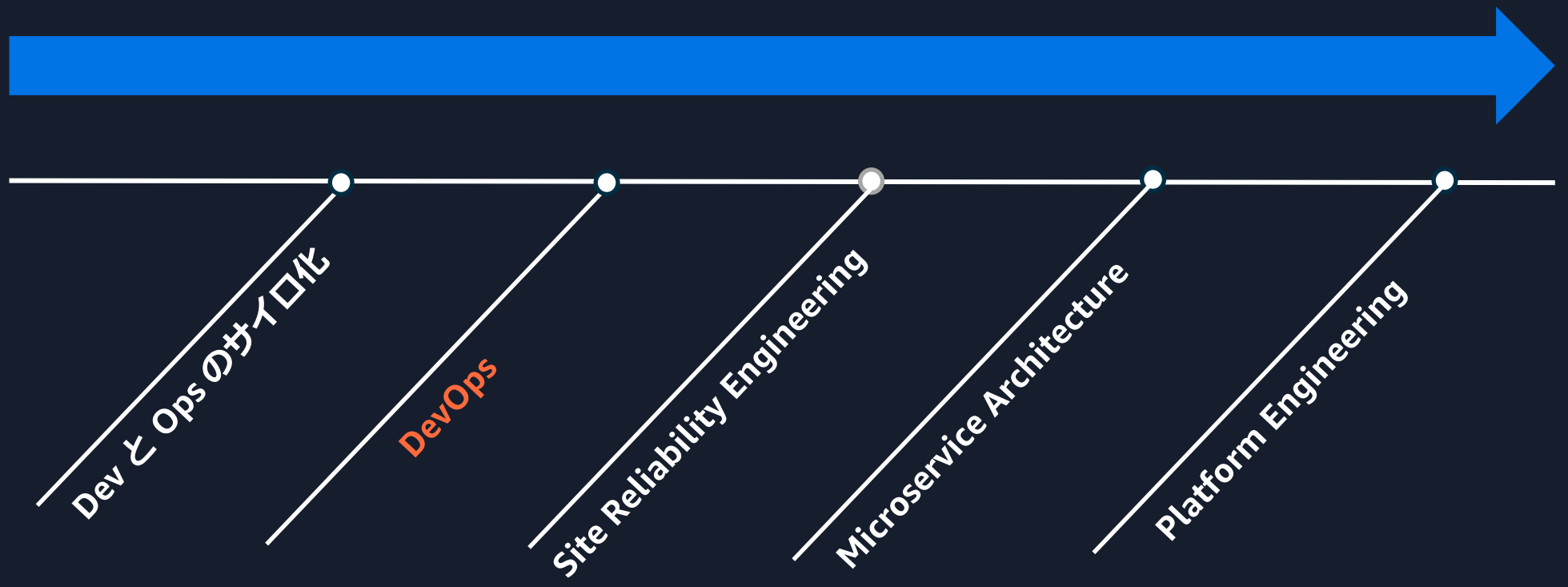
（機能開発に責任を持つ）

運用チーム

（稼働率に責任を持つ）

開発チームとインフラチームの組織の歪みをなくすには？

クラウドネイティブ化の背景



DevOps という解決策

You build it, you run it.

DevOps エンジニア

（開発と運用の両面からビジネス
全体への寄与を優先している）

DevOps という解決策

You build it, you run it.

できるだけ稼働率を
高めながらリリース

できるだけ頻繁に
新機能をリリース

DevOps エンジニア

（開発と運用の両面からビジネス
全体への寄与を優先している）

DevOps とは文化

頻繁にリリースする

稼働率を高める

相反する両方の側面を達成できるように考えていく組織の文化

DevOps ではない手法の例

手作業での一貫性のない
ソフトウェアのデプロイ

不十分な手作業での
品質保証

障害時のロールバック
計画がない

リリースを行うのに
管理者の手動での承認必須

DevOps ではない手法の例

頻繁にリリースできない

手作業での一貫性のない
ソフトウェアのデプロイ

稼働率が下がる可能性高

不十分な手作業での
品質保証

稼働率が下がる可能性高

障害時のロールバック
計画がない

稼働率が下がる可能性高

頻繁にリリースできない

リリースを行うのに
管理者の手動での承認必須

DevOps な手法の例

継続的インテグレーション

継続的デリバリー

マイクロサービス

Infrastructure as Code

Observability

コミュニケーションと連携

DevOps な手法の例

継続的インテグレーション

継続的デリバリー

マイクロサービス

Infrastructure as Code

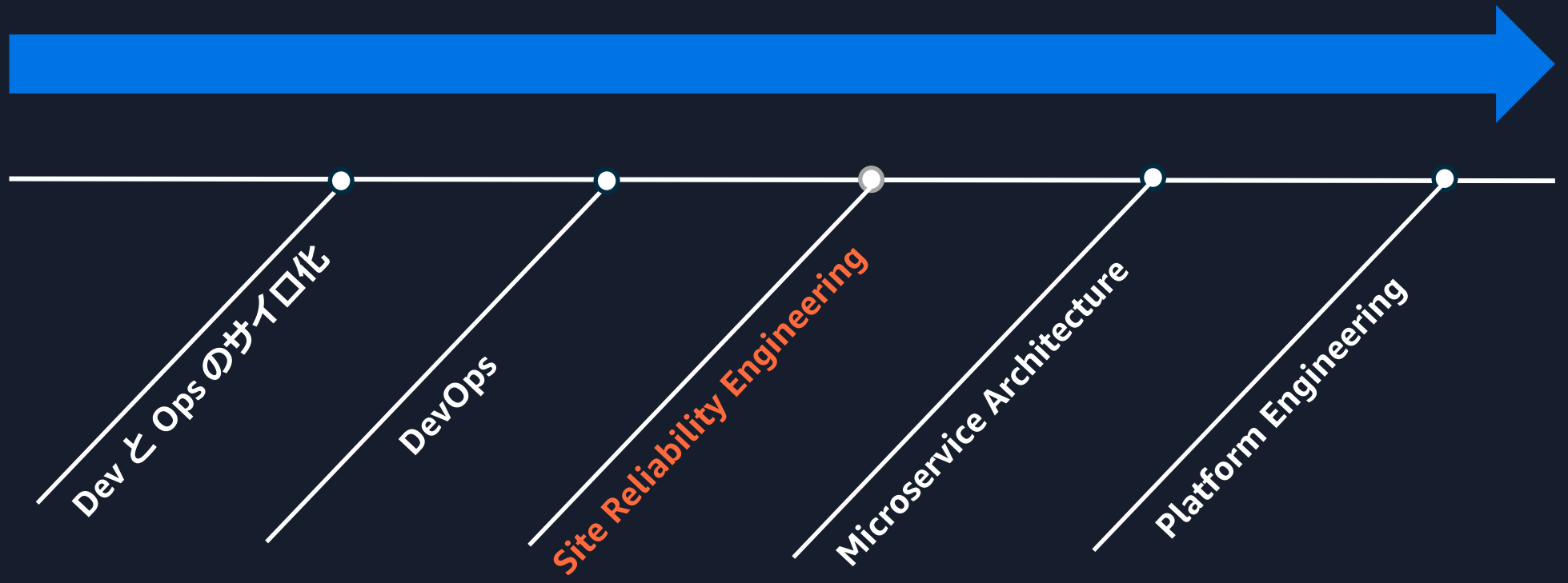
Observability

コミュニケーションと連携

稼働率を下げずに、頻繁にリリースできるようになる！

リリースの頻度を下げずに
サービスの稼働率を高めるには？

クラウドネイティブ化の背景



Site Reliability Engineering

SRE は、 DevOps を実践するための手法のひとつ

リリース頻度を下げずに、サービスの稼働率を高めるために
オペレーションをソフトウェアの問題として扱う手法

言い換えると、運用の問題をソフトウェアで解決する手法

参考：https://medium.com/@bruce_25864/the-road-to-sre-ad4c73df78b8

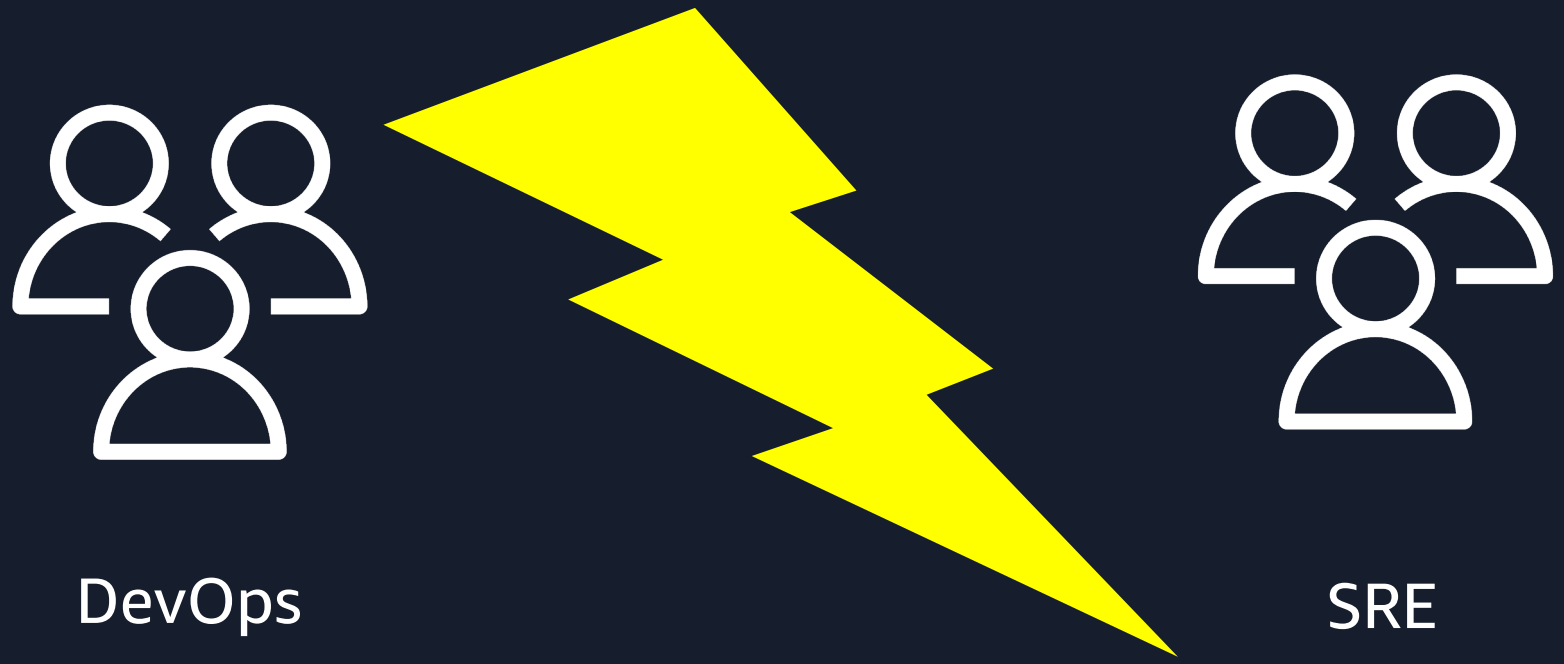
The ROAD to SRE

SRE で取り入れられている4つの実践

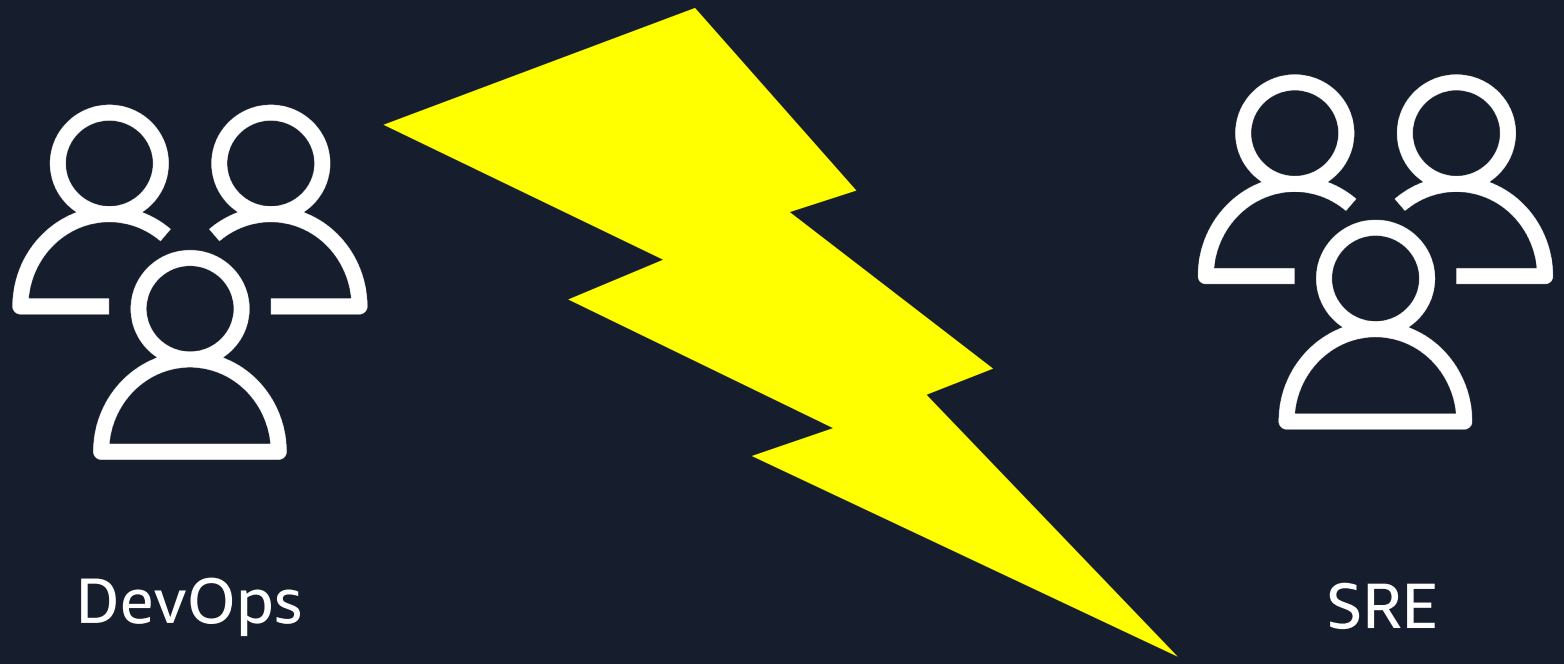
- ***Response*** – 障害発生時の適切なオンコール体制、対処方法 (Playbook / Runbook) を提供する。
  - Severity ごとの MTTD (平均検出時間)、MTTA (平均確認時間)、MTTR (平均解決時間)、MTBF (平均故障間隔) などの主要な指標を改善する。
- ***Observability*** – サービスの状態に関して詳細なテレメトリ (TEMPLE etc...) を提供する。
  - 実用的なアラートを用意し、インシデントの SLI への影響を最小限に抑える。
- ***Availability*** – 顧客の観点から測定する意味のある SLI/SLO を提供する。
  - ワークロードのボトルネックや障害シナリオを特定し、回復力を向上させる。
- ***Delivery*** – 一貫した自動化されたビルド、プロビジョニング、デプロイメントを提供する。
  - カナリアリリースなどのプログレッシブデリバリー戦略により、より安全にデプロイを行う。

サービスが巨大化したら
何が起きるだろうか

DevOps と SRE の矛盾？



DevOps と SRE の矛盾？



DevOps の内側の概念としてあったはずの SRE というプラクティスが、
巨大化とともに別チームに切り離され、再びサイロが生じてしまう。

クラウドネイティブ化に伴って
チーム構成もクラウドネイティブに

ライフサイクルによって分割された巨大な組織

ソフトウェア開発ライフサイクル毎に分割された組織に起きるサイロ

ライフサイクルによって分割された巨大な組織

ソフトウェア開発ライフサイクル毎に分割された組織に起きるサイロ

一つの機能をリリースするのに、全てのチームが関与する必要がある。

ライフサイクルごとの組織で起きる課題

- 特定の機能を開発するのに、すべてのチームが関わる必要がある
  - ステークホルダーが多いので、コミュニケーションコストが大きくなる。
  - **リンゲルマン効果**によりメンバーの士気やパフォーマンスが低下する。
  - 開発 → QA → インフラとリリースのたびに依頼が発生するので、リリースに余分な時間がかかる。また、チーム間で優先順位の認識が共有されなくなる。
  - チーム間でフィードバックが共有されず改善されにくい。
- チーム毎に別々の責務を担っているのでサイロが生じる。
  - 開発チームはどんどんリリースしたい。
  - 運用チームは安全にリリースしたい。
  - QAチームはまとめてリリースしたい。

リンゲルマン効果

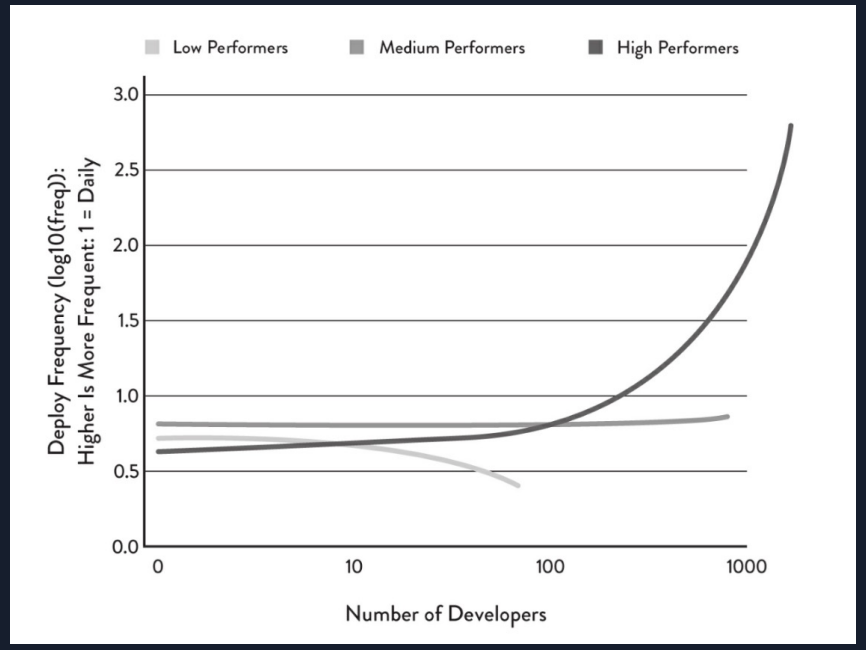
- 集団で共同作業をすると、人数の増加とともに 1 人あたりの仕事効率が低下する。
- 以下には、具体例として調査結果を掲載する。

Table 2
Relative Performance as a Function of Group Size

No. of workers	Work usable in practice (relative figures) Furnished per worker	Total
1	1.00	1.00
2	0.93	1.86
3	0.85	2.55
4	0.77	3.08
5	0.70	3.50
6	0.63	3.78
7	0.56	3.92
8	0.49	3.92

Note. This table is a translation and copy of the table given on page 9 of Ringelmann (1913b).

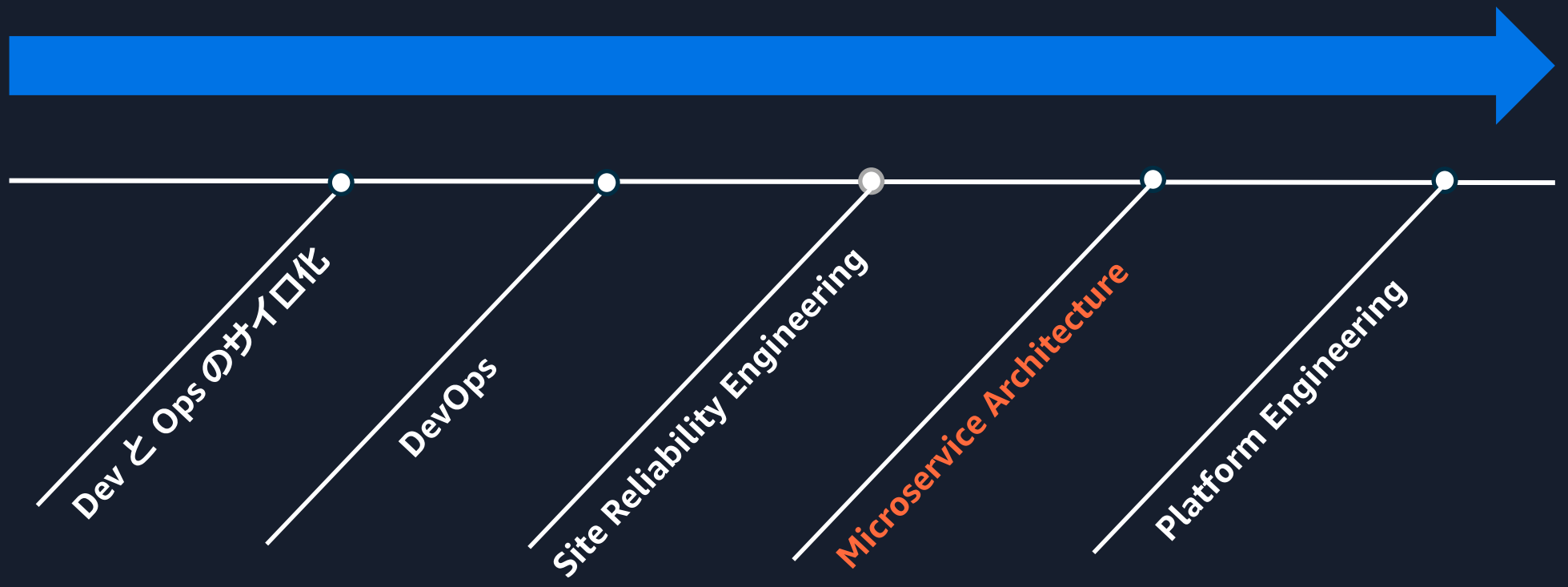
Kravitz, D. A. And Martin. B.(1986) Ringelmann Rediscovered : The original article. Journal of Personality and Social Psychology. 50(5) pp936-941



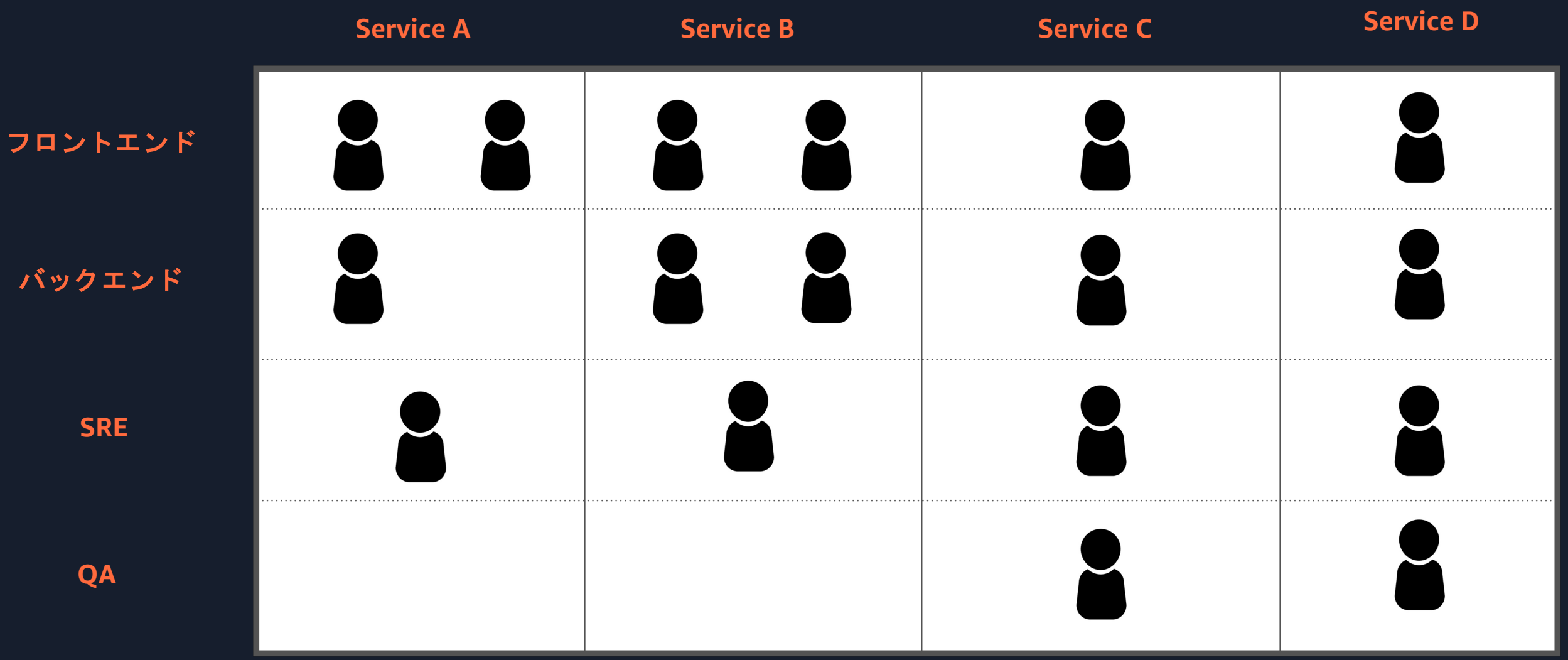
ACCELERATE: The Science of Learn Software and DevOps
https://itrevolution.com/product/accelerate/

リンゲルマン効果が起きる原因

クラウドネイティブ化の背景



自律的に機能する主体的な組織へ



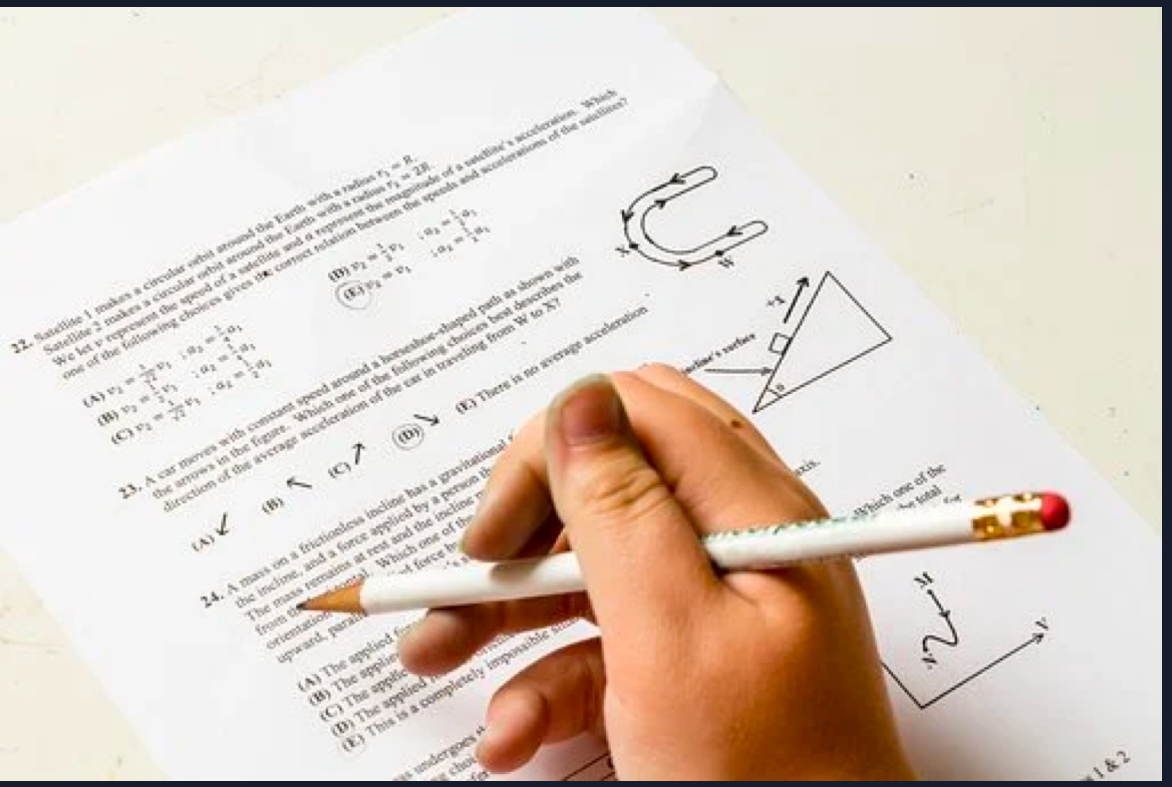
マイクロサービスアーキテクチャにおける究極的な成果物は新たな組織

Amazon でのツー・ピザ・チーム

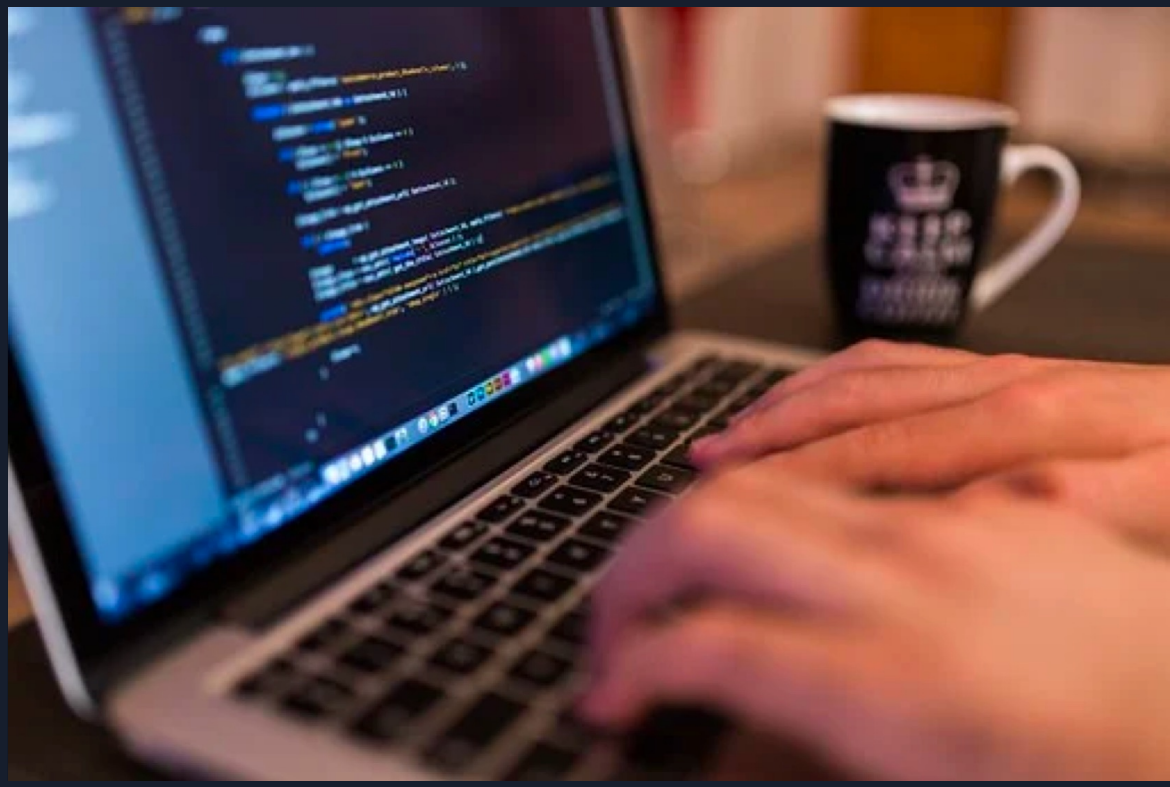
- Amazon では、ピザが2枚で足りるくらい小さなチーム (通常 5 – 10 人) にする。

ただ、チームの人数を小さくするだけではリンゲルマン効果は低減できない。

これらの違いは？



宿題



趣味

言われたことを
言われた通りに

自分で考えて
自分で実行する

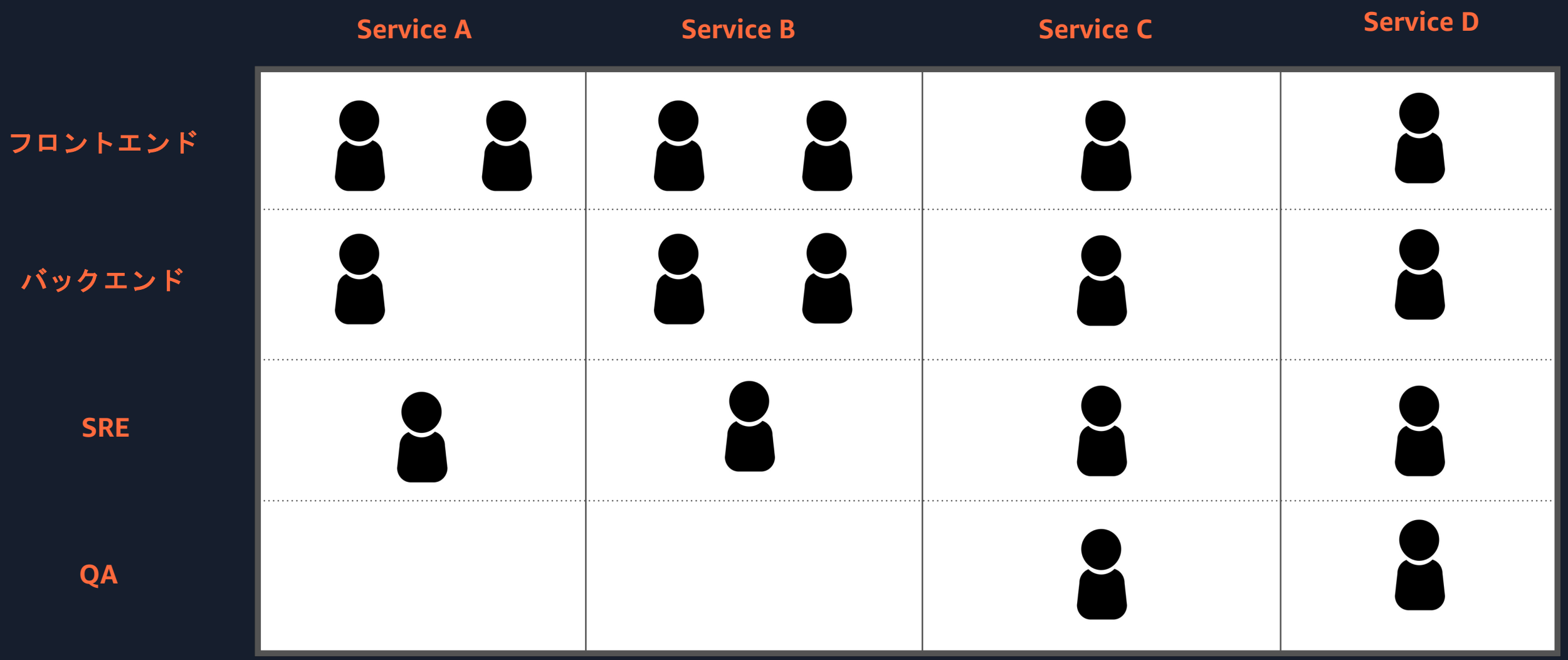
Amazon でのツー・ピザ・チーム

- Amazon では、ピザが2枚で足りるくらい小さなチーム (通常 5 – 10 人) にする。
  - ツー・ピザ・チームは、明確な線引きのされた説明責任を持つ。
  - ツー・ピザ・チームには、自主的に機能するための権限と環境を提供する。
  - ツー・ピザ・チームが自ら課題を見つけ解決することを可能とする。
  - リーダーは料金所ではなくガードレールを提供する。
- ツー・ピザ・チームというネーミングためか、チームの人数に意識が向けられることが多いが、**より重要なことは、自律性と説明責任があること**である。

ツー・ピザ・チームは、シングルスレッドリーダーとして、お客様のために独立してイノベーションを起こす権限を持ちます。

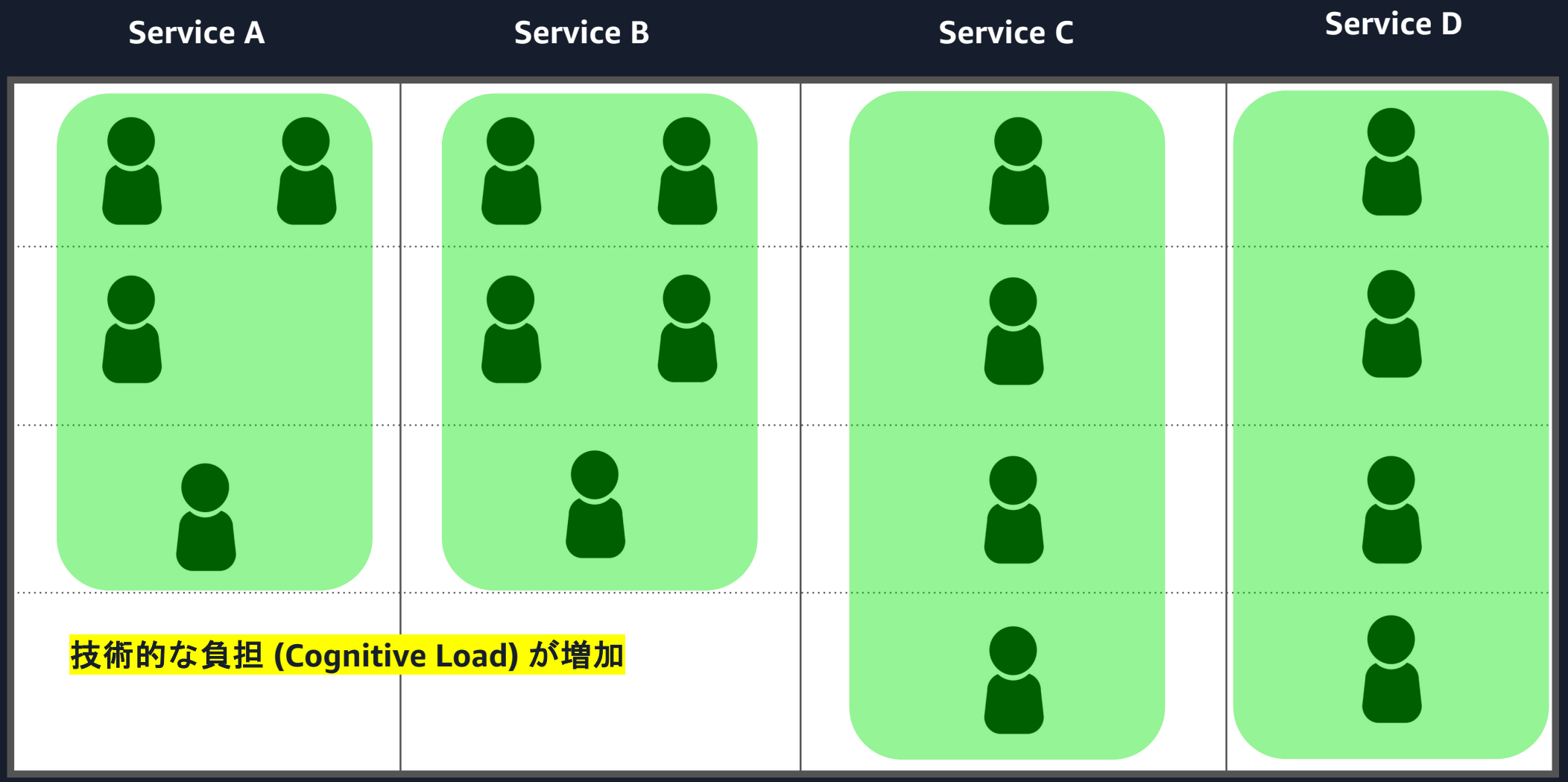
ツー・ピザ・チームは、規模だけを問題としているのではなく、オーナーシップや独立性を育み、チームレベルにまで落とし込むことも重要視しています。

自律的に機能する主体的な組織へ



ツー・ピザ・チームの負荷が大きくなる

至る所で行われる車輪の再発明



チームごとの技術レベルが揃わない

Cognitive Load（認知負荷）

- Cognitive Load とはユーザーが情報を処理したり理解する際にかかる負荷
- 認知心理学の言葉で、人間の脳にワーキングメモリがあり、その容量は限られていることから発生して作られた言葉

1. **Intrinsic cognitive load**
   - 学習対象そのものの複雑さによる負荷
2. **Extraneous cognitive load**
   - 業務や学習に無関係な余分な負荷
3. **Germane cognitive load**
   - 理解や学習が進行する際に発生する適切な負荷

3 つの Cognitive Load

Intrinsic cognitive load

学習対象の難解さによる負荷

- マイクロサービスアーキテクチャで複雑な分散合意の実装
- 利用しているアルゴリズムを変更し、計算量を log オーダーに減らす
- ライブラリのバグを直す
- Kubernetes のマニフェストを学習
- 大規模すぎるコードベースの保守
- 難しすぎるセキュリティガイドラインを理解し、適切に実装する

Extraneous cognitive load

学習に無関係な余分な負荷

- 情報がまとまっておらず開発に必要な情報を収集するところから始まる
- プログラムをビルドするためのコマンドが毎回わからなくなる
- 無関係のアラートでメンションされる
- デプロイのたびに上司の承認が必要
- ツールへのログインが頻繁すぎる
- 業務で利用するパソコンが重い

Germane cognitive load

適切な学習のための必要な負荷

- お客様が求めている機能は何か考える
- サービスの正確な挙動を把握する
- 新機能を実装するにあたってアーキテクチャを考え、Design Doc を書く
- 今進めているプロジェクトの背景や目的について整理し、提案を進める

3 つの Cognitive Load

Intrinsic cognitive load

学習対象の難解さによる負荷

- マイクロサービスアーキテクチャで複雑な分散合意の実装
- 利用しているアルゴリズムを変更し、計算量を log オーダーに減らす
- ライブラリのバグを直す
- Kubernetes のマニフェストを学習
- 大規模すぎるコードベースの保守
- 難しすぎるセキュリティガイドラインを理解し、適切に実装する

Extraneous cognitive load

学習に無関係な余分な負荷

- 情報がまとまっておらず開発に必要な情報を収集するところから始まる
- プログラムをビルドするためのコマンドが毎回わからなくなる
- 無関係のアラートでメンションされる
- デプロイのたびに上司の承認が必要
- ツールへのログインが頻繁すぎる
- 業務で利用するパソコンが重い

Germane cognitive load

適切な学習のための必要な負荷

- お客様が求めている機能は何か考える
- サービスの正確な挙動を把握する
- 新機能を実装するにあたってアーキテクチャを考え、Design Doc を書く
- 今進めているプロジェクトの背景や目的について整理し、提案を進める

ツー・ピザ・チームは
ここに認知負荷を寄せるべき！

3 つの Cognitive Load

Intrinsic cognitive load

学習対象の難解さによる負荷

- マイクロサービスアーキテクチャで複雑な分散合意の実装
- 利用しているアルゴリズムを変更し、計算量を log オーダーに減らす
- ライブラリのバグを直す
- Kubernetes のマニフェストを学習
- 大規模すぎるコードベースの保守
- 難しすぎるセキュリティガイドラインを理解し、適切に実装する

仕組みによって解決すべき

Extraneous cognitive load

学習に無関係な余分な負荷

- 情報がまとまっておらず開発に必要な情報を収集するところから始まる
- プログラムをビルドするためのコマンドが毎回わからなくなる
- 無関係のアラートでメンションされる
- デプロイのたびに上司の承認が必要
- ツールへのログインが頻繁すぎる
- 業務で利用するパソコンが重い

仕組みによって解決すべき

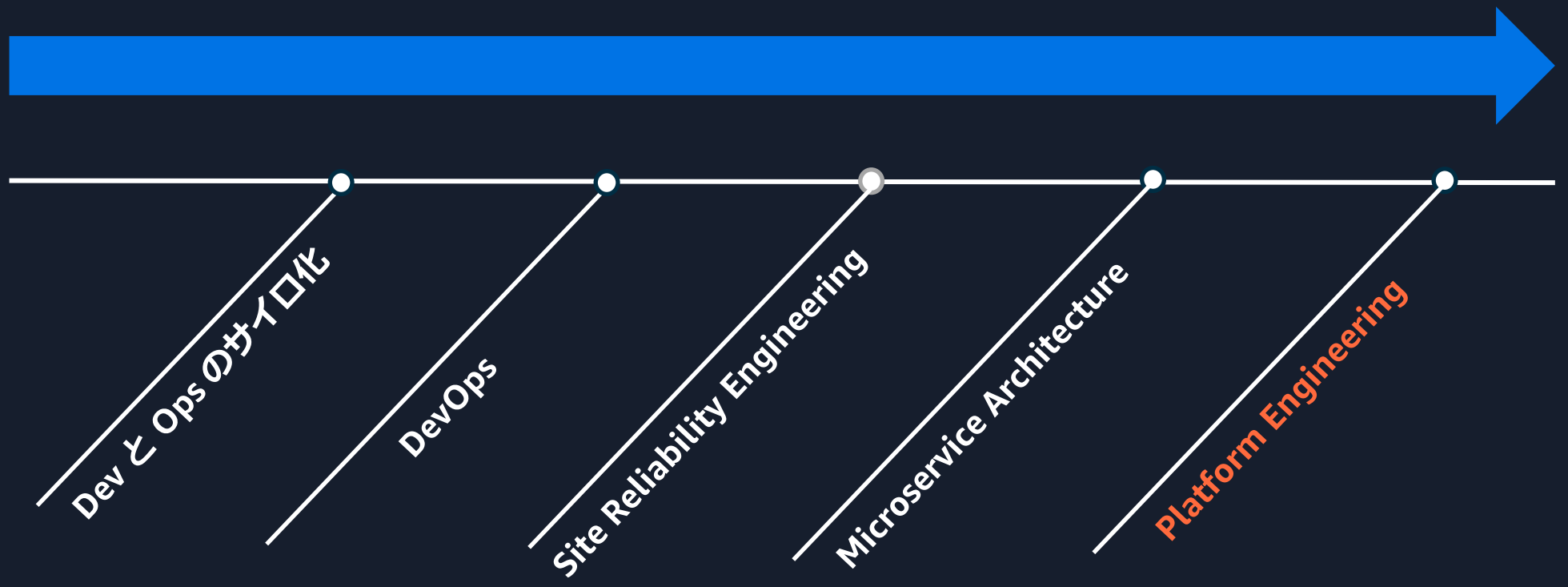
Germane cognitive load

適切な学習のための必要な負荷

- お客様が求めている機能は何か考える
- サービスの正確な挙動を把握する
- 新機能を実装するにあたってアーキテクチャを考え、Design Doc を書く
- 今進めているプロジェクトの背景や目的について整理し、提案を進める

ツー・ピザ・チームは
ここに認知負荷を寄せるべき！

クラウドネイティブ化の背景



Platform Engineering が提供するもの

ツー・ピザ・チームの Intrinsic cognitive load と Extraneous cognitive load を低減するための仕組みとして以下のものなどを提供する。

ウェブポータル
プラットフォームの機能を提供するためのポータル

API
プラットフォームの機能を自動的に提供するためのAPI

ゴールデンパス
プラットフォームの能力を最適に利用するためのガイド

開発環境
ホストされたIDEやリモート接続ツールなどの開発環境

Observability
ダッシュボードを使用したサービスの観測

セキュリティ
コードとアーティファクトの静的解析、ランタイム解析、ポリシーの強制などのセキュリティサービス

アーティファクト
コンテナイメージやライブラリ、ソースコードなどのアーティファクトの保存

インフラストラクチャ
コンピューティングリソース、ネットワーク、ブロックとボリュームストレージなどのインフラストラクチャ

引用：https://tag-app-delivery.cncf.io/whitepapers/platforms/

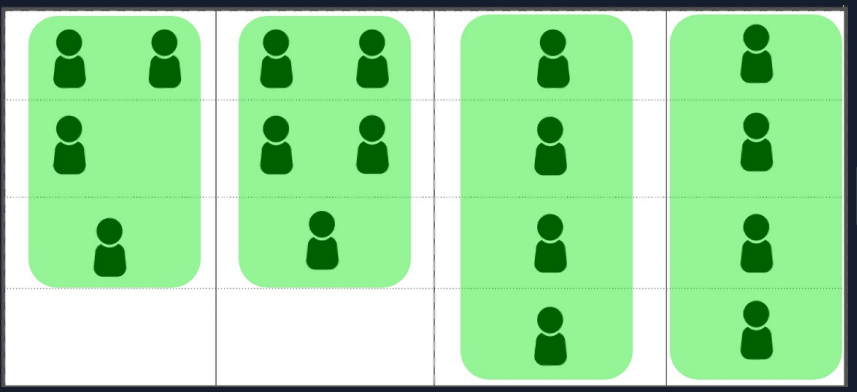
共通基盤を作ってみたが
失敗した経験がある方も多いのでは

Platform as a Product

- Platform Engineering では、ツー・ピザ・チームの Intrinsic cognitive load と Extraneous cognitive load を低減するための仕組みを Product として提供する。
- Platform はツー・ピザ・チームからのフィードバックをもとに開発される。
  - 従来型の共通基盤は、単なる共通化であり、利用が強制されるものであった。
  - ツー・ピザ・チーム利用を強制されることはなく、他の方法を選択できる。
  - プラットフォームチームは、局所的で特殊な要件に対応する必要はない。

認知負荷を減らしながら実現する DevOps

Intrinsic cognitive load と Extraneous cognitive loadを
Platform を利用することによって低減する。
開発するサービスについては、
運用も含め End-to-End で説明責任と権限を持つ
つまり、Platfrom を利用しないという権限も持つ



ツー・ピザ・チーム

API 利用

Platform

DevOps

Platform Team

Platform という 1 つの Productに
End-to-End で説明責任と権限を持つ

Platform as a Product

- Platform Engineering では、ツー・ピザ・チームの Intrinsic cognitive load と Extraneous cognitive load を低減するための仕組みを Product として提供する。
- Platform はツー・ピザ・チームからのフィードバックをもとに開発される。
  - ツー・ピザ・チームは利用を強制されることはなく、他の方法を選択できる。
  - 従来型の共通基盤は、単なる共通化であり、利用が強制されるものであった。
  - プラットフォームチームは、局所的で特殊な要件に対応する必要はない。

局所的で特殊な要件には、どう対応するのか？

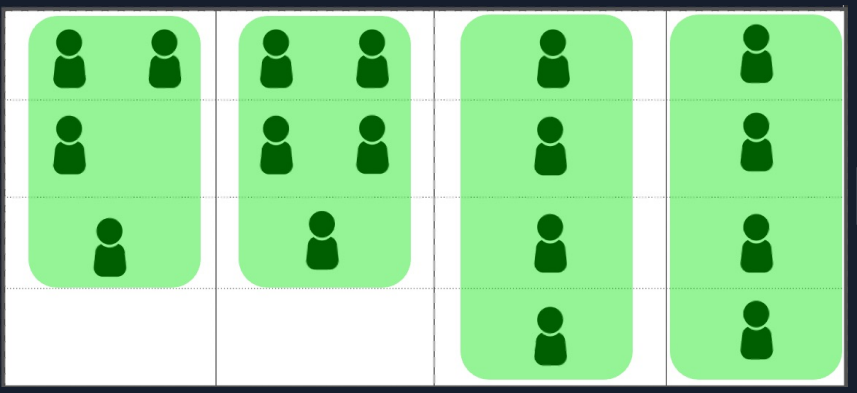
ツー・ピザ・チームだけでは解決できない高難易度の
信頼性に関する技術課題を解決するための組織構成が必要に

Embedded SRE

- Embedded SRE はツー・ピザ・チームの一員ではなく、独立したチーム。
- サイロを生まないようツー・ピザ・チームに一時的に加わり
  専門家としてシステムの信頼性の向上に関わるプロジェクトに取り組む。
- 一定期間おきにローテーションを行う。信頼性に関わる技術の専門家である SRE はチームを離れ、また別のチームに加わる。
- プラットフォームチームが提供する Platform という Product を利用することによって全体の 80% 程度の生産性や信頼性に関わる認知負荷を解決する場合には、残りの 20% のドメインに局所的な問題を Embedded SRE が解決する。

認知負荷を減らしながら実現する DevOps

Intrinsic cognitive load と Extraneous cognitive loadを
Platform を利用することによって低減する。
開発するサービスについては、
運用も含め End-to-End で説明責任と権限を持つ
つまり、Platfrom を利用しないという権限も持つ



ツー・ピザ・チーム

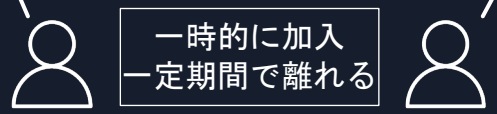
API 利用

Platform

DevOps

Platform Team

Platform という 1 つの Productに
End-to-End で説明責任と権限を持つ



Embedded SRE

ツー・ピザ・チームと SRE チームの間にサイロは生まれない。
ツー・ピザ・チームが、運用も含め End-to-End で説明責任と
権限を持つ組織構成であるが、
Intrinsic cognitive load を Embedded SRE によって低減できる。

Embedded SRE と Platform Engineering の違い

	Embedded SRE	Platform Engineering
低減する認知負荷	Intrinsic cognitive load	Intrinsic cognitive load と Extraneous cognitive load
責任範囲	サービスの信頼性、稼働率	認知負荷を低減するためのプロダクトを開発運用すること
ゴールとなる指標	SLI/SLO	SPACE, DORA
チームトポロジー	Enabling	Platform
コミュニケーションモード	Collaboration	X as a Service
解決する課題の粒度	配属したチームに固有の課題	多くの利用者を目指す

Platform Engineering は
大規模な組織でしか成り立たない？

Platform Engineering へのよくある誤解

- Platform は、すべてのツールを Self Service 化しなければならない
- Platform はInternal Developer Platform を運営 ・提供しなければならない
- Platform Engineering をするには Kubernetes を使わなければならない

Platform Engineering へのよくある誤解

- Platform は、すべてのツールを Self Service 化しなければならない
- Platform はInternal Developer Platform を運営 ・提供しなければならない
- Platform Engineering をするには Kubernetes を使わなければならない

もちろん、上記のような Platform Engineering をやっている組織もあるだろう。

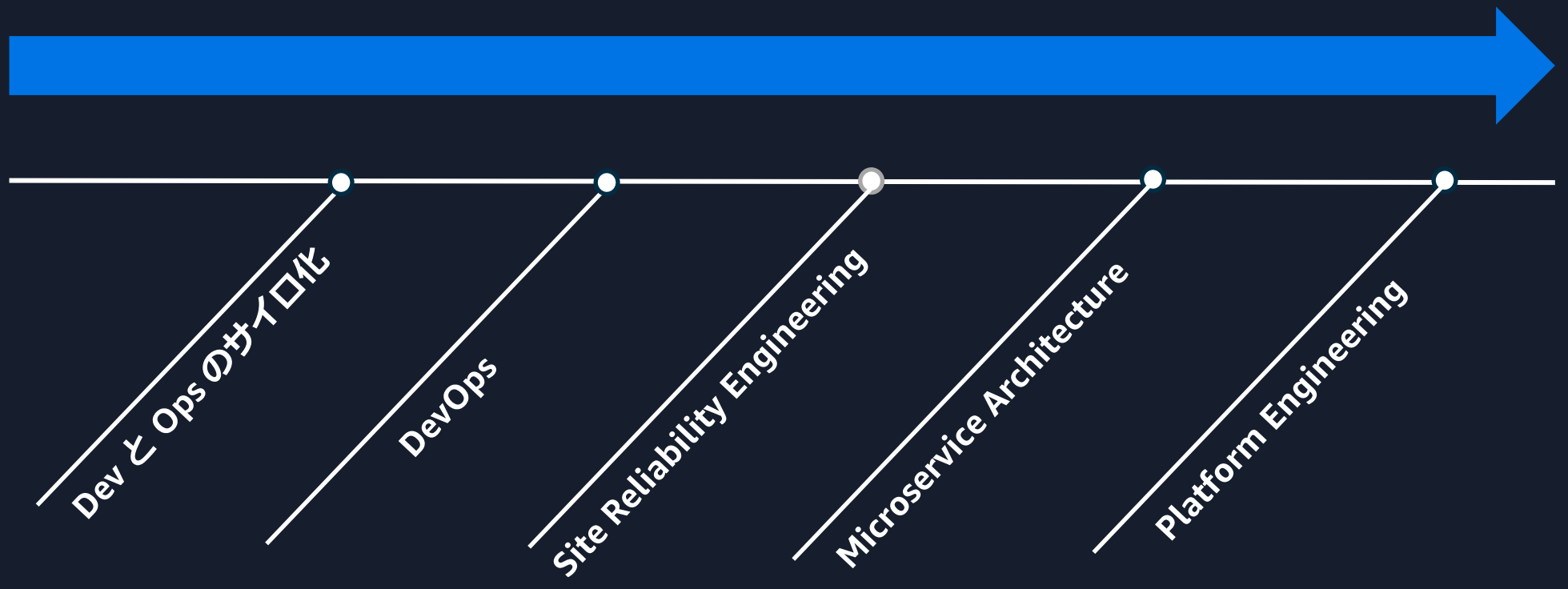
重要なのは、How (Internal Developer Platform, Kubernetes etc...)ではなくて Why(なぜやるのか)

クラウドネイティブ化の推進などによって
ツー・ピザ・チームの認知負荷が大きくなっているとき
Platform Engineering は価値を発揮する。

Thinnest Viable Platform

- 価値提供を実現できる最小限の Platform
  - Self Service 化せずに、むしろ積極的に既存の OSS や AWS サービスを活用
  - AWS サービスを利用することで、Platform チームの運用負荷を低減
  - Internal Developer Platform を提供せずとも、まずは標準手順書を提供するだけ
- Thinnest Viable Platform によって認知負荷を低減できているかを計測
  - SPACE, DORA を計測して数値として変化を追う
  - ニーズに応じて Platform の提供するものを選択

クラウドネイティブ化の背景



Thank you!

Ryota Yamada

Tech Training Specialist