オンラインゲームの
インフラ検証、設計と運用

@POLYGON PICTURES Tech Seminar

~映像制作パイプラインとアーティストのテクニック 5~

株式会社Aiming
菅野 明洋

自己紹介

- 名前
  - 菅野明洋
- 所属
  - 株式会社Aiming
    - インフラチーム

免責事項

- 本発表における見解は、
  私自身の見解で行っております。

- 所属する会社・団体の見解を
  反映したものではありませんので、
  ご了承ください。

前置き

前置き

- ゲーム的な要素や他業種との違いを中心に発表します
  - 話す内容は主観にて選定しております
  - 気になった部分や分からない部分はその場で質問しても構いません
- インフラの一般的な部分の説明は割愛しております

要件・システムデザイン

ゲームではユーザ体験を第一に

ユーザにとって面白いか？

- 技術に縛られ面白さが減ると本末転倒
  - 本質大事
- 常に新しい価値を提供が必要
- 新しい価値のために
  リスクを取ることもある

システム矛盾・不満の削減

快適にプレイできるか？

- 遊べないことはユーザの不満に
  - 24時間ユーザが遊べることを考慮
  - 不慮の事故、多方面から攻撃があっても
    サービス維持しなくてはいけない
    - ~~どうしても落ちることはありますが。~~
  - ユーザにとっての機会損失は不満へ

ユーザから見た一貫性担保

- ユーザ視点での矛盾の発生の排除
  - ユーザ操作の反映遅れ・無効
    - 対戦ゲームにて操作が反映されなかった
    - etc
  - データ損失・ロールバックの排除
    - 取得したアイテムの損失等
    - 課金の反映失敗

システムへの不満

- 矛盾の削減
  - ユーザ視点で発生した矛盾は、
    事実関係なしに不満に
- 極力システムへのヘイトを軽減
  - ユーザにとっての不利は
    全てシステム側にヘイトへ
- 全ての不満がユーザ減・売上減へ直結

その他の要件

- 一般的な話なので割愛
  - セキュリティ
    - 定期的にシステム監査
      - 社内
      - ゲームシステム
  - コスト削減

インフラ検証：
ミドルウェア・クラウド選定

選定する際の苦悩

- システムの複雑化にて自前主義は困難
  - ~~納期がありますし~~
- OSSやシェアウェア、クラウドサービスを用いて開発費を軽減
- 本質はエンターテイメントであるため、ゲーム的に本質的でない機能は、
  他社製のソフトウェアを頼ることもある
  - ~~楽したい~~

選定までのアプローチ

- 要求機能からの検討
- 基本機能の性能向上

要求機能からの検討

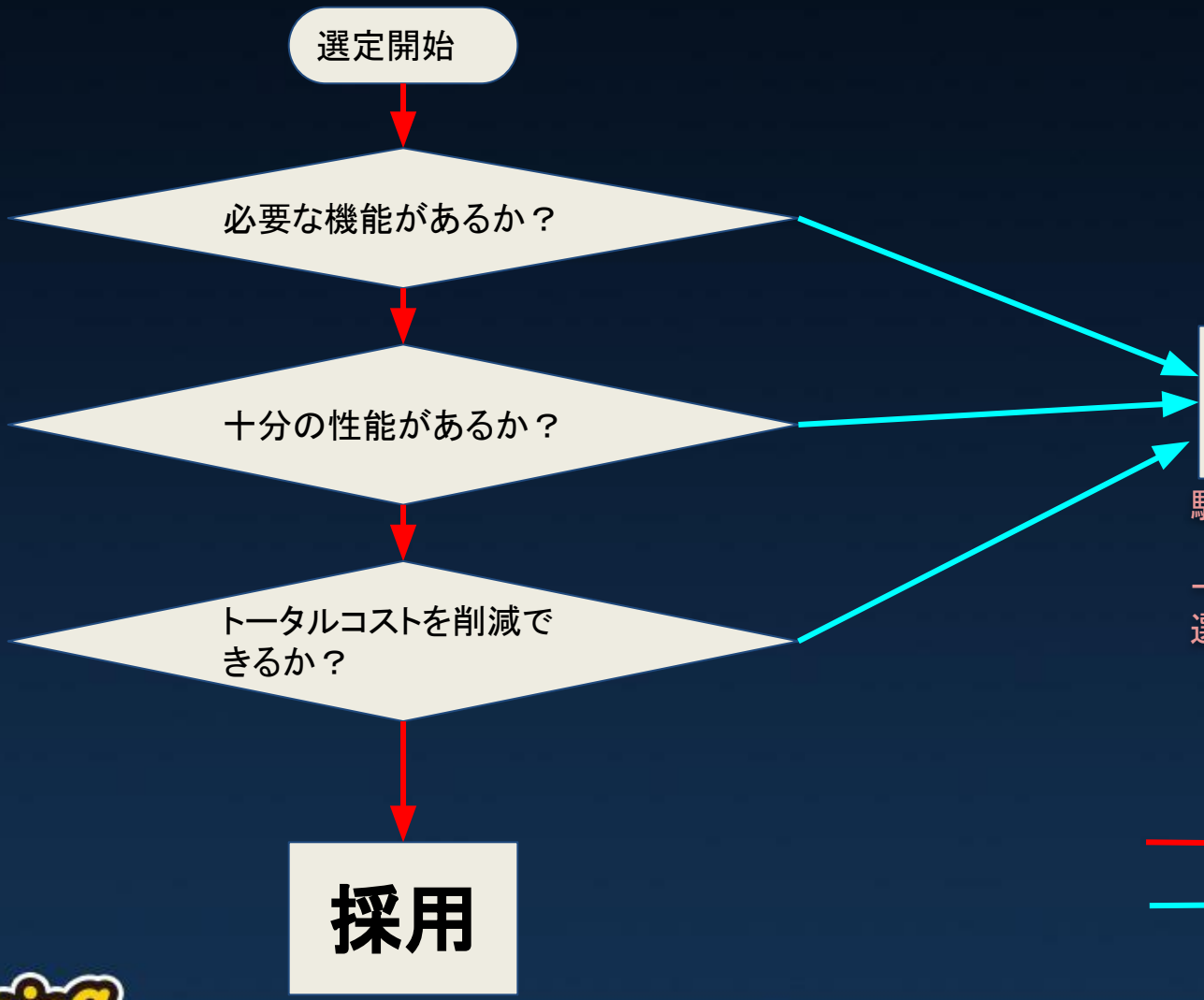
ギルドを200人までにするから
チャット上限を200人にしたい

わかりました。
検討します。

基本機能の性能向上

(例)バックエンドの性能が向上すれば、体感速度が向上する

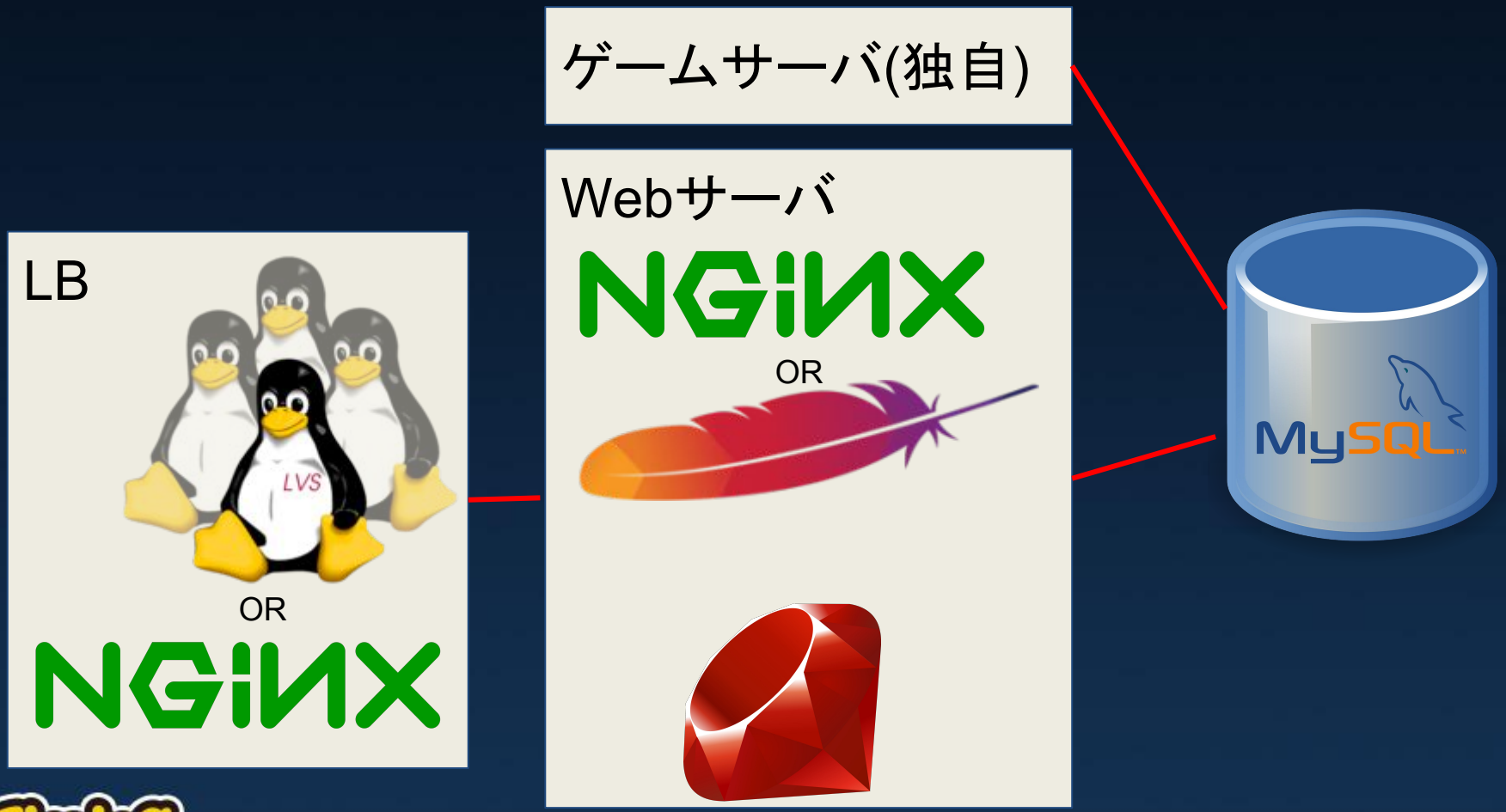
ミドルウェア/クラウド選定の流れ



自分で作る例(1)

- クラウド VS OSS、又はOSS VS 自社製
  - フルマネージドDBから自社管理のDBへ
  - OSSチャットツールVS自社製チャット
  - L4,L7 LB
    - 性能や障害時の切り替え等のハンドリング
- 自作
  - システム、監視プラグインの実装
- 特別な場合
  - クラウド、特定ソフトウェア等が使用不可

自分で作る例(2)



必要な機能の確認

- 要求仕様を満たす機能を有するか確認
- 要求に対して一部補助できることも確認
  - 複数のミドルウェアや独自実装で要求を実現

性能値の確認(1)

- 予想アクティブユーザ数から目標値算出
  - ゲームイベント中の最大同時接続数
  - メンテナンス終了後の最大瞬間風速
  - 平常時の同時処理数
  - テレビCMや広告宣伝後
    - アクセスが10倍以上来ることも・・・？
  - etc
- 数年先も見据えた拡張性や余力を確認
  - サービスの成長を支えるため
  - リリース後の改修はコストアップ

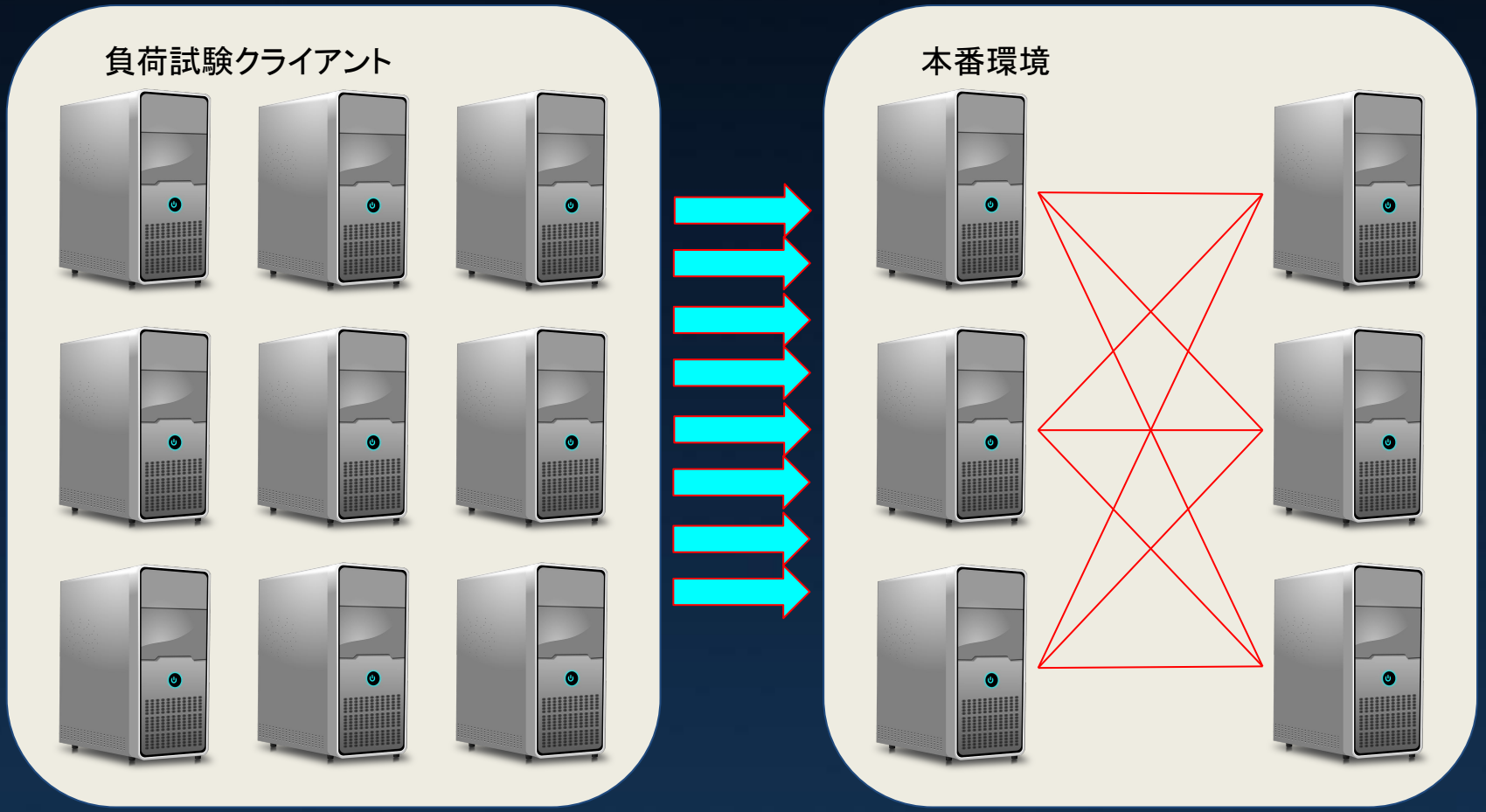
性能値の確認(2)

- どこまでのサービスを保証するか
  - 可用性(稼働率向上)
  - レイテンシ
  - レスポンス速度
- ノウハウ吸収
  - どう実装するのが良いか？
    - 得意な処理は？
    - 苦手な処理は？
    - アーキテクチャの強みの確認

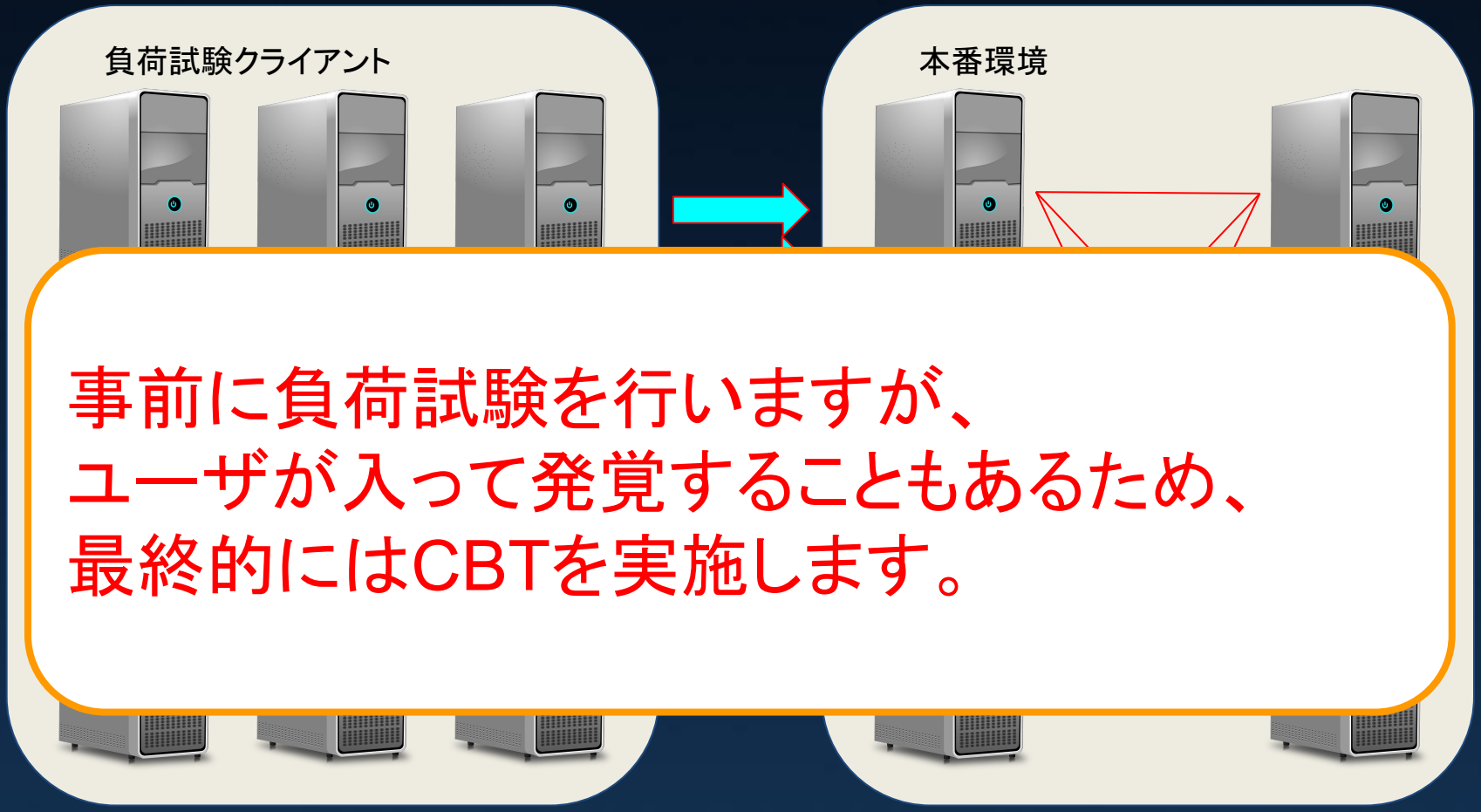
性能値の確認(3)

- 汎用ベンチマーク
  - WebならJMeterやGatling等々
  - DBならTPC系やSysbench、JDBC Runner等
    - 実装や仕様に合わせて試験を行うのが望ましい
- 複雑な要件は検証プログラムを実装
  - ステートフル通信の検証(ゲーム等)
  - 障害試験(整合性チェック等)

ステートフル通信での負荷試験

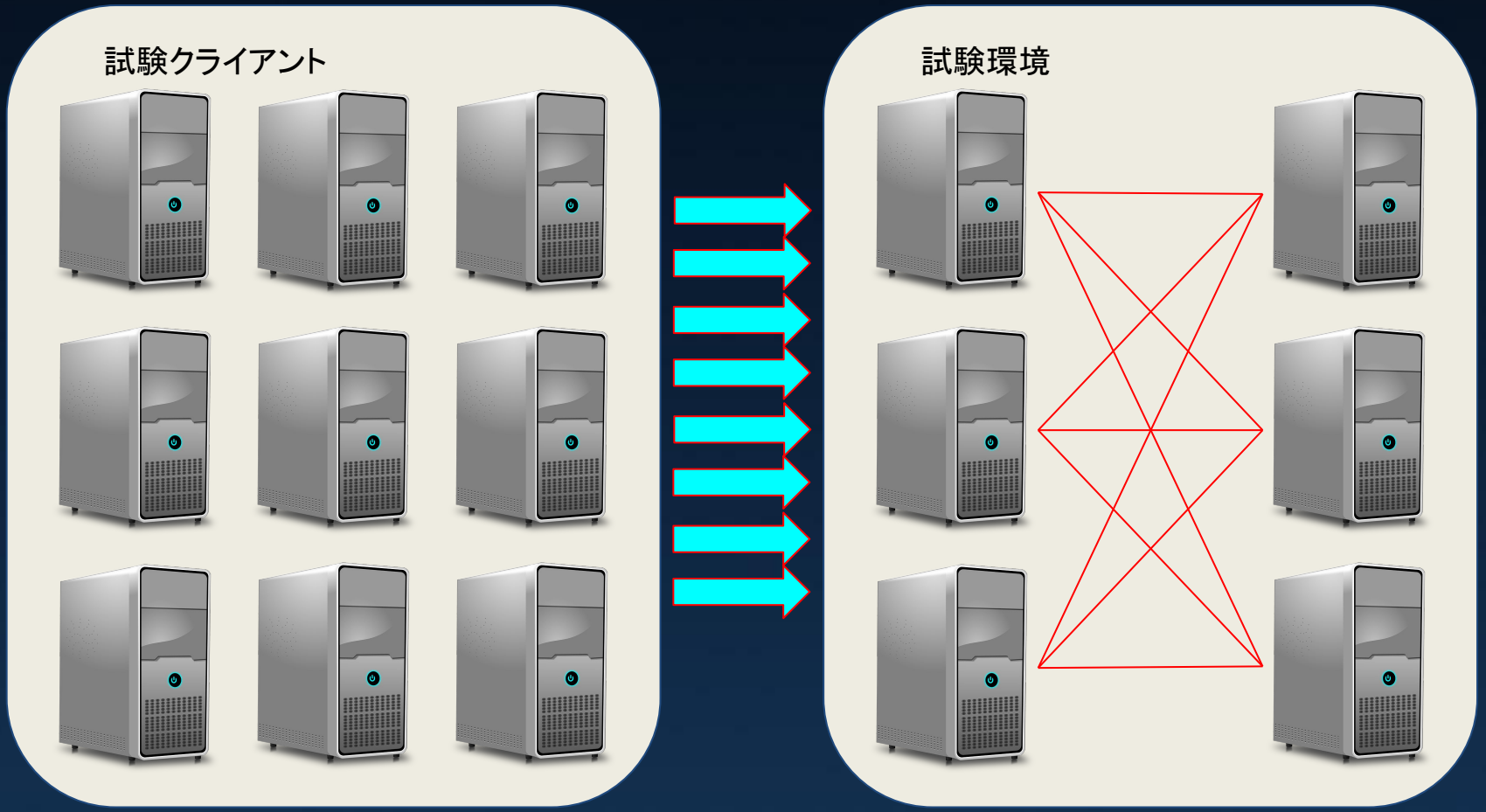


ステートフル通信での負荷試験



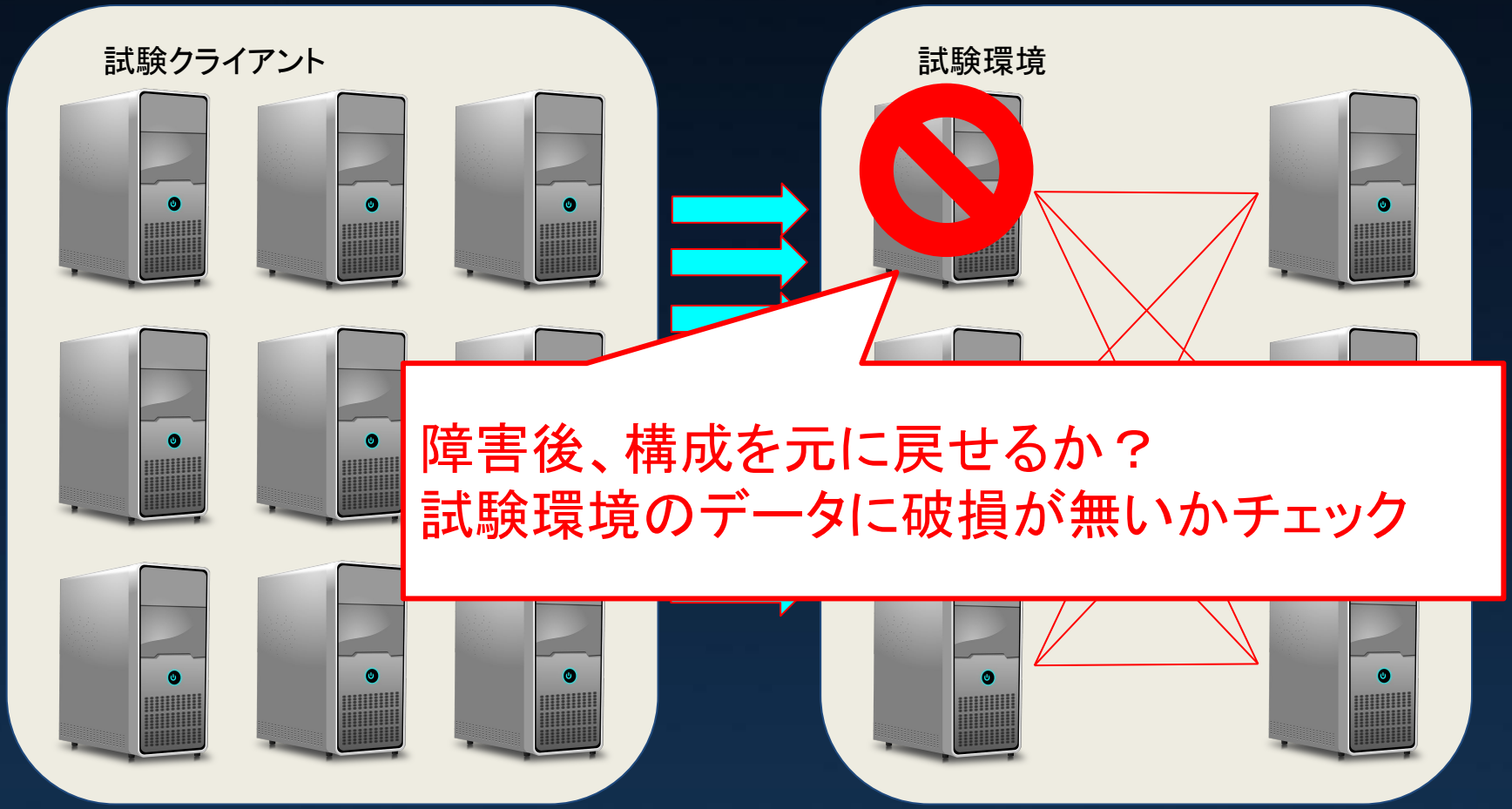
事前に負荷試験を行いますが、
ユーザが入って発覚することもあるため、
最終的にはCBTを実施します。

障害試験の例

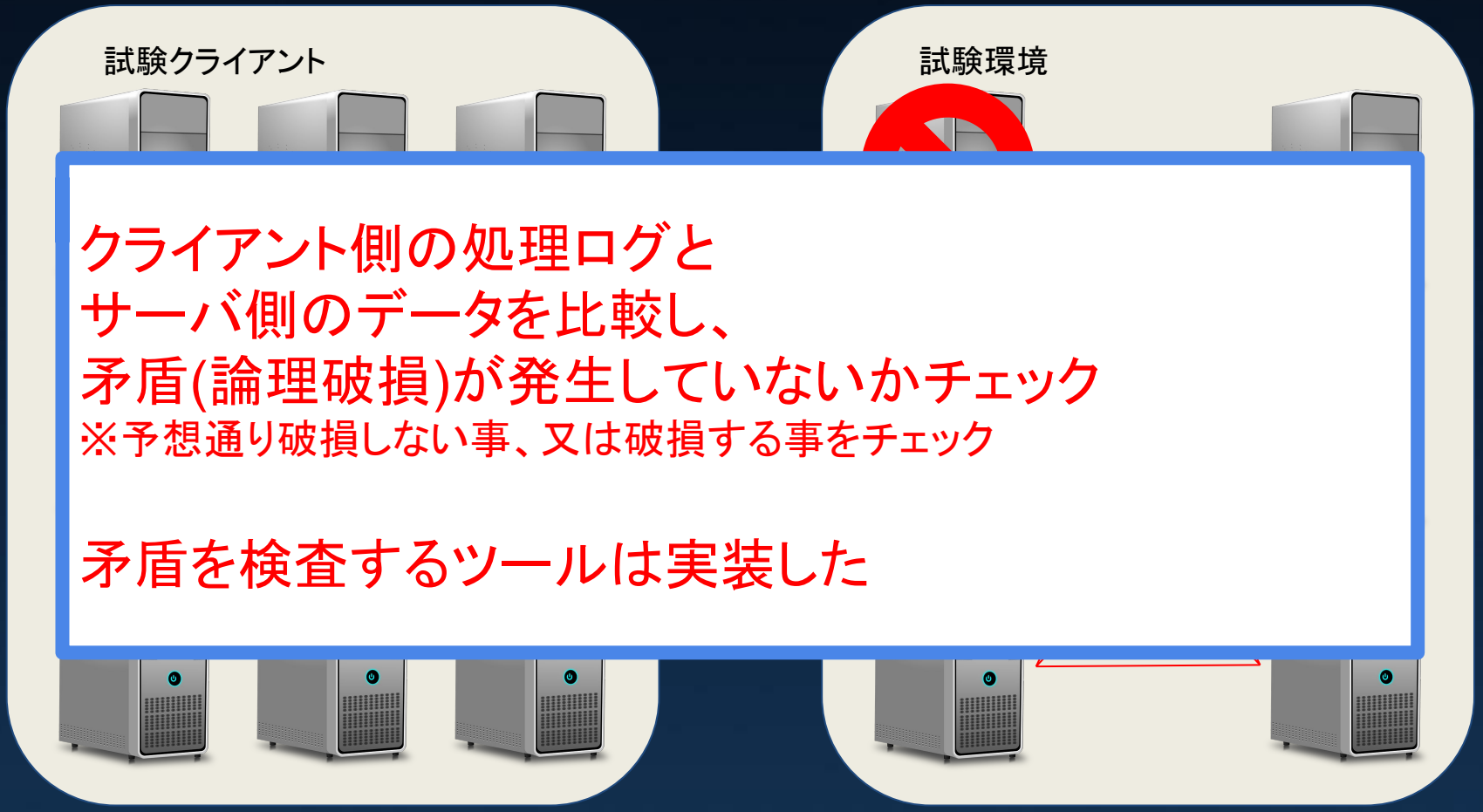


障害試験の例

障害試験の例



障害試験の例

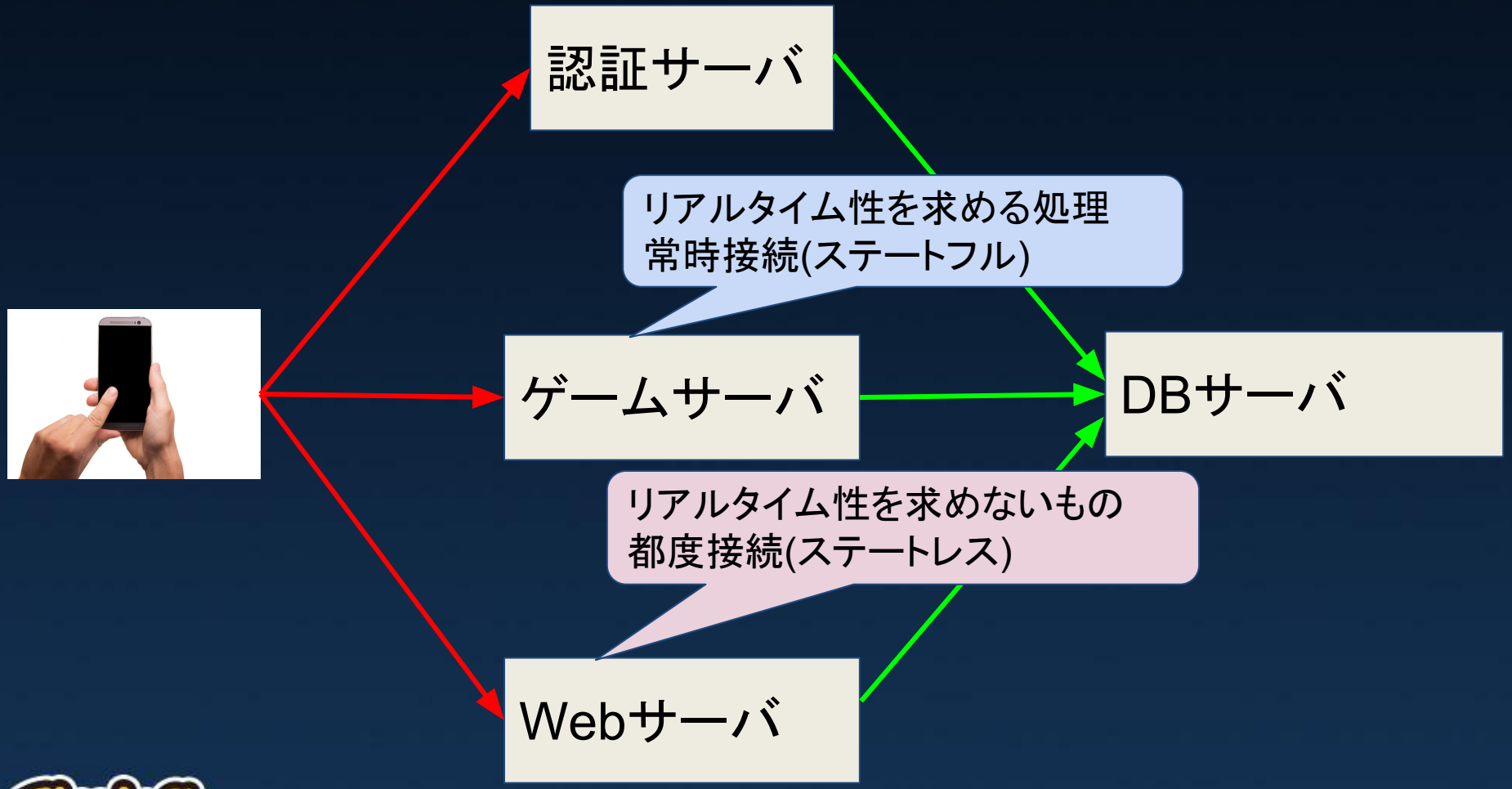


クライアント側の処理ログと
サーバ側のデータを比較し、
矛盾(論理破損)が発生していないかチェック
※予想通り破損しない事、又は破損する事をチェック

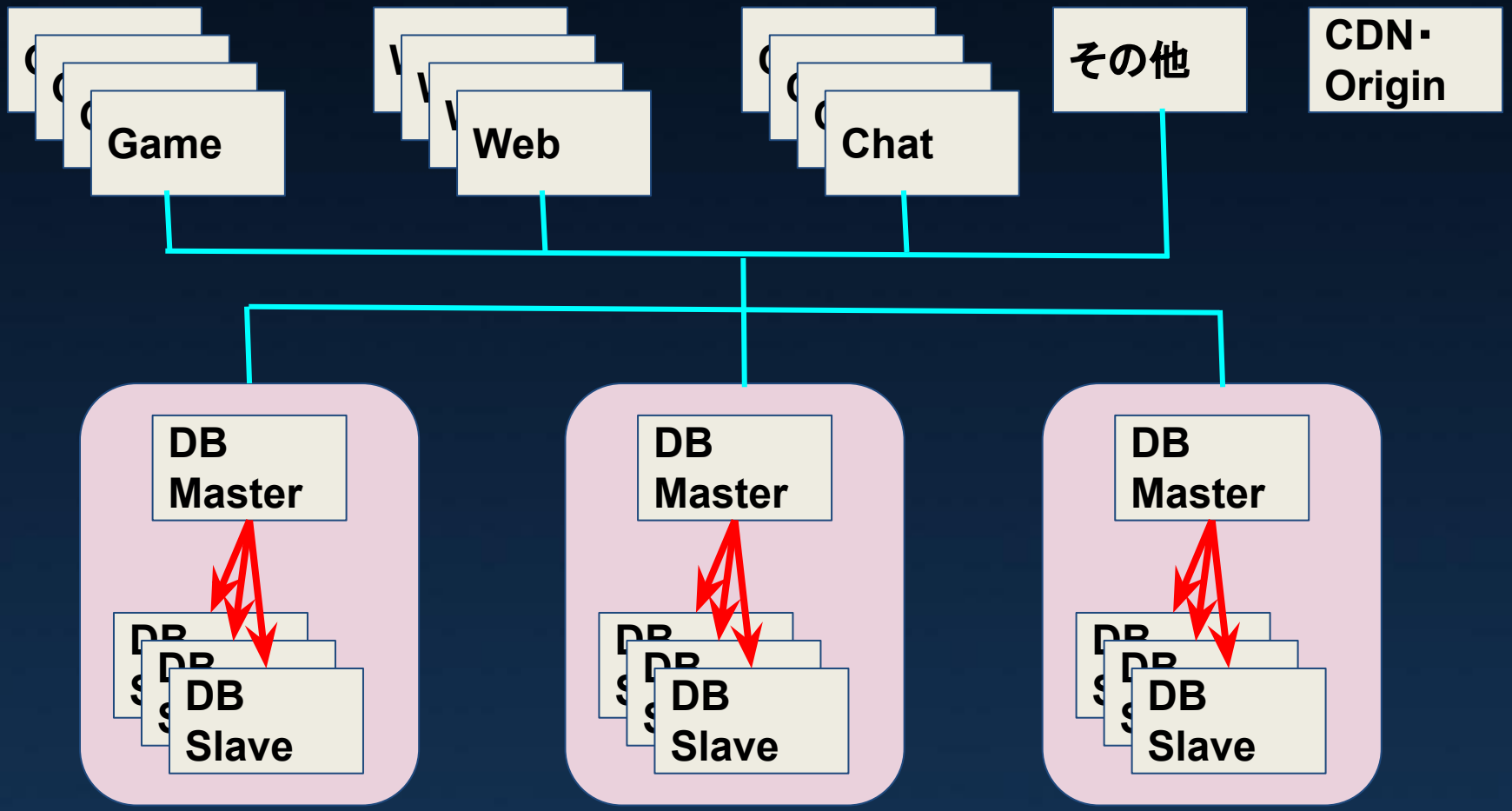
矛盾を検査するツールは実装した

インフラ設計
一般的なMMORPGの設計例

MMORPGの簡易構成

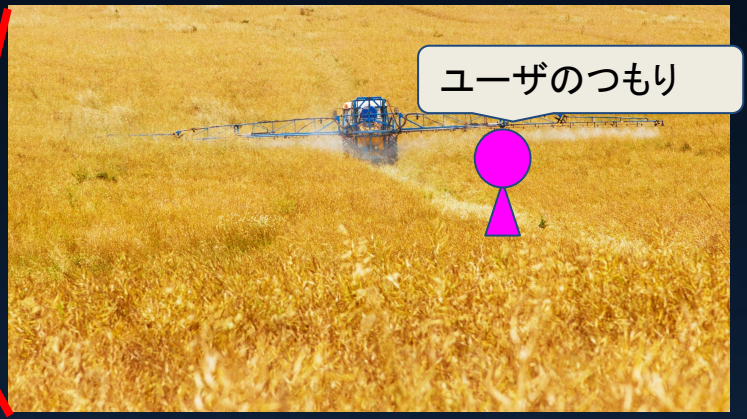


ゲームシステム構成例



MMORPGの処理例(1)

ゲームサーバ



ゲームサーバ

MMORPGの処理例(2)

ゲームサーバ

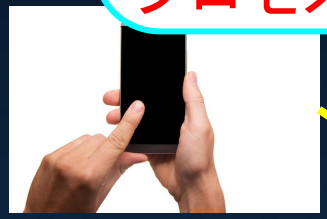
ゲームサーバ

フィールド切替時に
サーバのデータを移動する

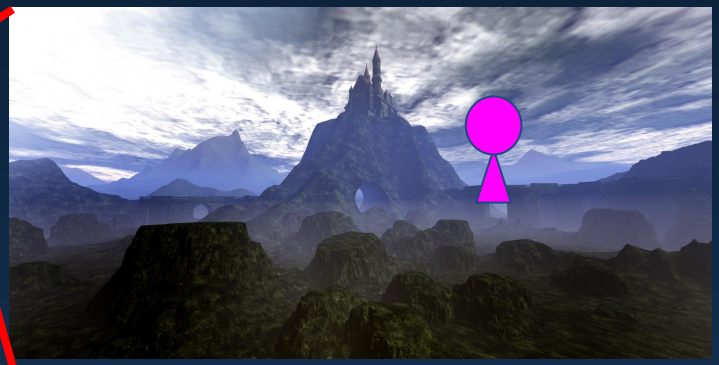
MMORPGの処理例(3)

ユーザ間のリアルタイム性を重視するため、
サーバを並べて冗長化することはしていない

ゲームフィールド上にいるユーザは同じサーバ上の
プロセスで処理を行う



ゲームサーバ



ゲームサーバ運用の問題点

- リソース管理が難しい
  - 負荷上昇時サーバ追加で対処ができない
    - ゲームサーバ上で動かしているプログラムの調整が必要
- 障害対応が難しい
  - 画一的なシステムでないため、オペレーションが複雑になり、非常時のオペレーションが属人化

インフラ設計
コンテナ型仮想化を用いた現状と課題

コンテナを用いたメリット

- デプロイ速度の向上
  - 開発環境から別の環境へのデプロイが容易
- 環境の標準化が容易に
  - 新しくJOINしたメンバーの導入が容易
  - 非エンジニアによるデプロイも容易に
- コンテナの増減設が容易

コンテナを用いた設計と課題(1)

- アーキテクチャの変更
  - モノリシックアーキテクチャから
    マイクロサービスアーキテクチャへの移行
  - 最初からコンテナを意識する必要がある
    - コンテナ型を想定してないシステムの
      移行コストが高い
      - 2年経過の開発案件では半年弱以上のコストを
        算出
        - » 実際は多くを妥協した
        - » 早期に検討した案件は対応が早かった

コンテナを用いた設計と課題(2)

- 性能が安定しない
  - 同じホストにあるコンテナ同士影響がある
    - コンテナ同士でリソースを食い合う
  - All コンテナ化は困難
    - スループットの面では物理が優位
      - 物理と比べハイパーバイザ型は80～70%
        コンテナ型は90～80%ぐらいの性能
        » ハイパーバイザ×コンテナ型は72～56%で見積もり
    - モノリシックなソフトは非コンテナが適切
      - DBやデータストア等は物理が良い

コンテナを用いた設計と課題(3)

- ログ収集
  - fluentdやrsyslogでかき集める?
    - GKEならStack Driver
- セキュリティ
  - セキュリティの担保をどうする？
    - サーバの役割毎にネットワークや
      コンテナを稼働させるホストを分離を検討

コンテナを用いた設計と課題(3)

- 監視
  - コンテナを動かすサーバの監視
  - コンテナの監視
    - コンテナのイベントをフックして死活監視
    - New Relic、Datadogなどの導入
    - コンテナ単位でヘルスチェック処理
    - etc
  - 監視データの管理
    - コンテナの入れ替わり・増減が激しい

ご静聴ありがとうございました。