金融の基幹システムを
1年半かけて .NET 6に移行した話

Atsushi Nakamura

.NET 6 migration of a financial company's mission-critical system.

About Me

About Me

中村 充志 / Atsushi Nakamura

- リコージャパン株式会社 所属
- Enterprise（おもに金融）系SIerのITアーキテクト
- 「持続可能なソフトウェア」の探求がライフワーク
- .NET界 朝活部 日本支部長

- Blog　http://www.nuits.jp
  https://zenn.dev/nuits_jp
- Twitter　@nuits_jp

.NET 6 migration of a financial company's mission-critical system.

Today's Materials

Today's Materials

金融の基幹システムを1年半かけて .NET 6に移行した話
https://zenn.dev/nuits_jp/articles/2022-08-26-migration-to-net6

Twitterの@nuits_jpアカウントから#csharptokyoハッシュタグをつけてリンクを投稿します。

ついでにフォローしてってください！

.NET 6 migration of a financial company's mission-critical system.

Agenda

Agenda

1. プロダクト概要
2. プロジェクト概要
3. 顧客をどう説得したか
4. テスト戦略
5. 炎上

Agenda

1. プロダクト概要
2. プロジェクト概要
3. 顧客をどう説得したか
4. テスト戦略
5. 炎上

Agenda

1. 炎上
2. プロダクト概要
3. プロジェクト概要
4. 顧客をどう説得したか
5. テスト戦略

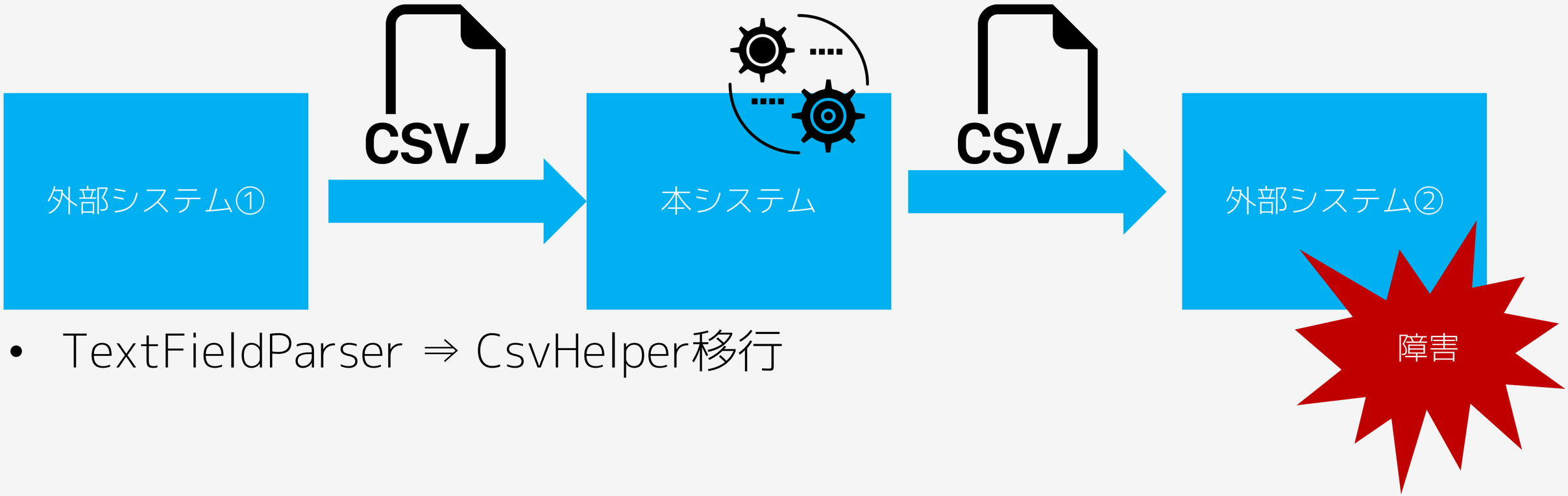
.NET 6 migration of a financial company's mission-critical system.

炎上

炎上

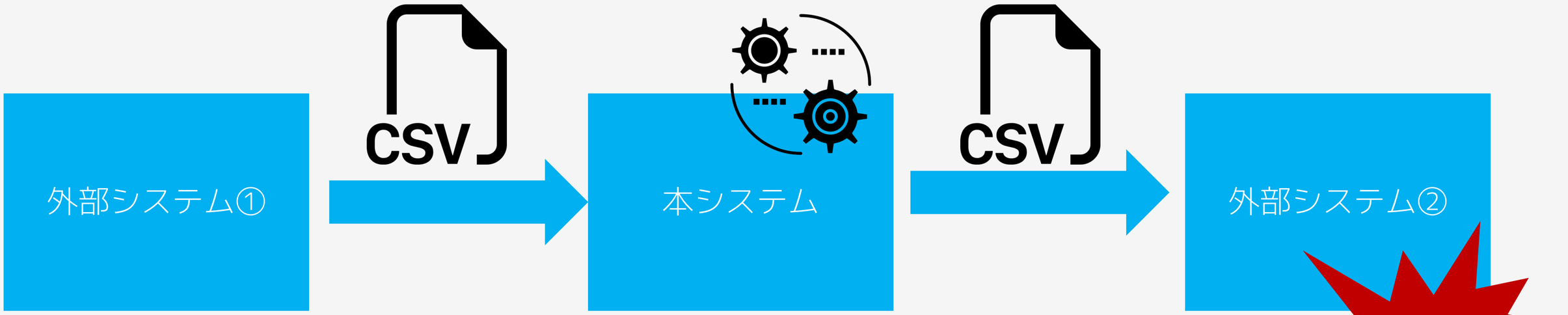
- TextFieldParser ⇒ CsvHelper移行

炎上



- TextFieldParser ⇒ CsvHelper移行

炎上



- TextFieldParser ⇒ CsvHelper移行
- デフォルトの挙動に差異があることに気が付かなかった
  - TextFieldParser　：Trim有効
  - CsvHelper　　：Trim無効
- 外部システムからの連携CSVの取込結果に差異が発生
- 幸い業務影響はなかった

え？つまらん？

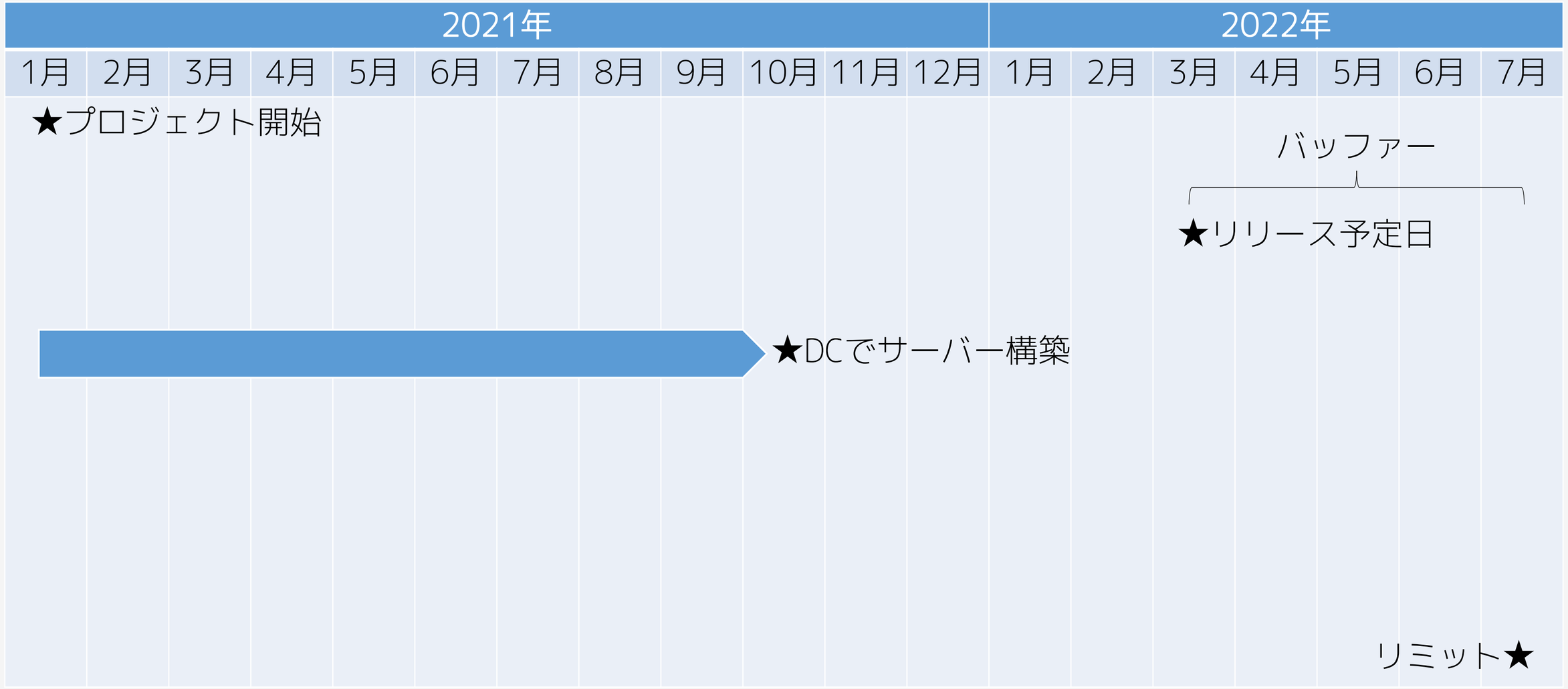
Agenda

1. プロダクト概要
2. プロジェクト概要
3. 顧客をどう説得したか
4. テスト戦略
5. 炎上

本当の炎上を見せてあげますよ

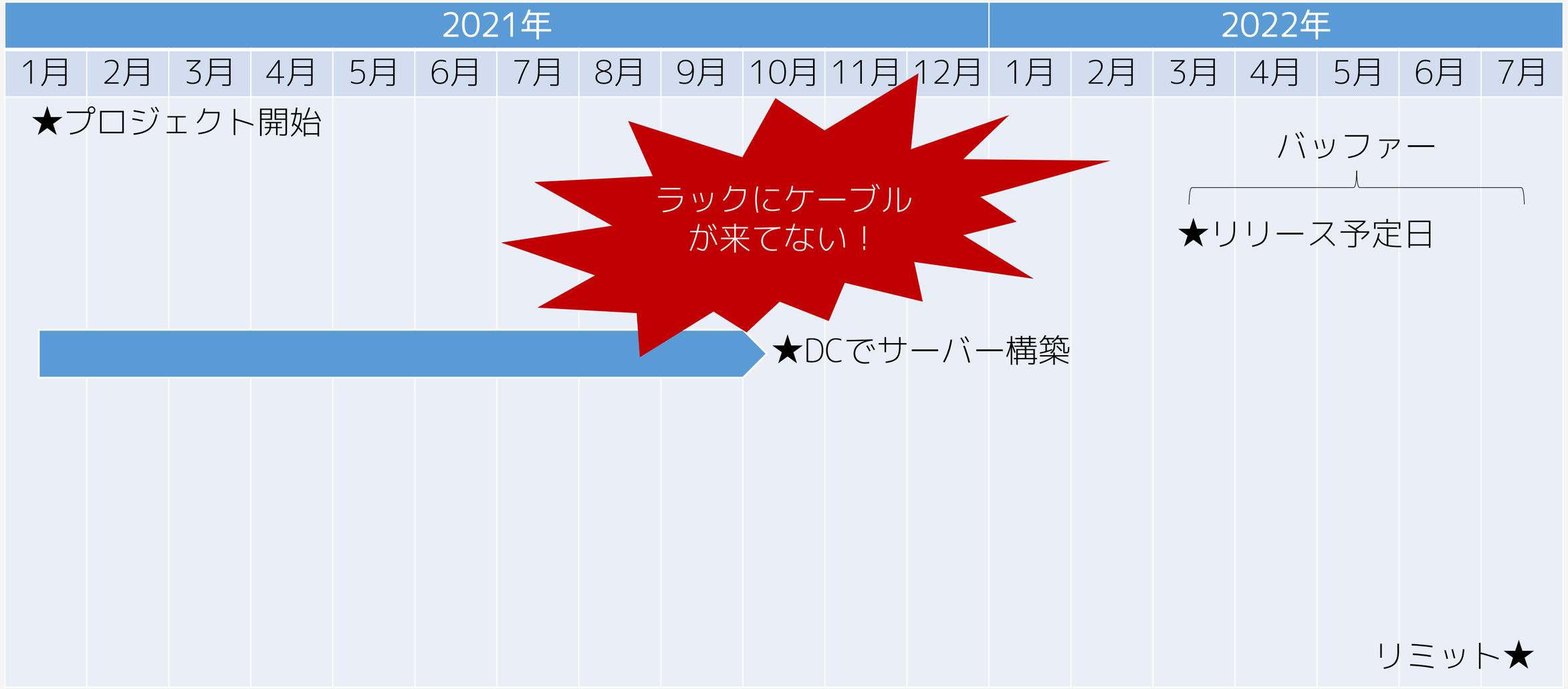
大炎上

- Windows Server 2008R2延長サポート終了：2022年7月12日



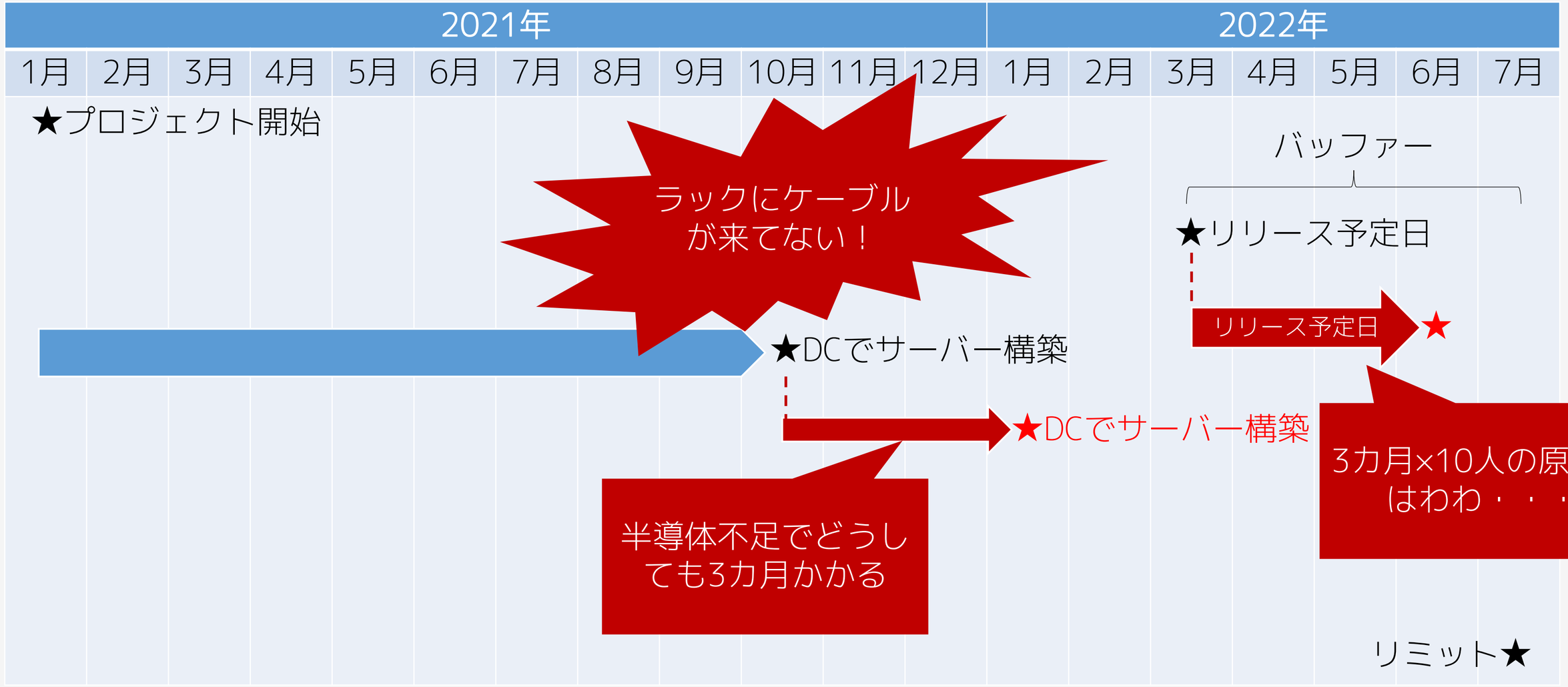
大炎上

- Windows Server 2008R2延長サポート終了：2022年7月12日



大炎上

- Windows Server 2008R2延長サポート終了：2022年7月12日



光ケーブルは半導体

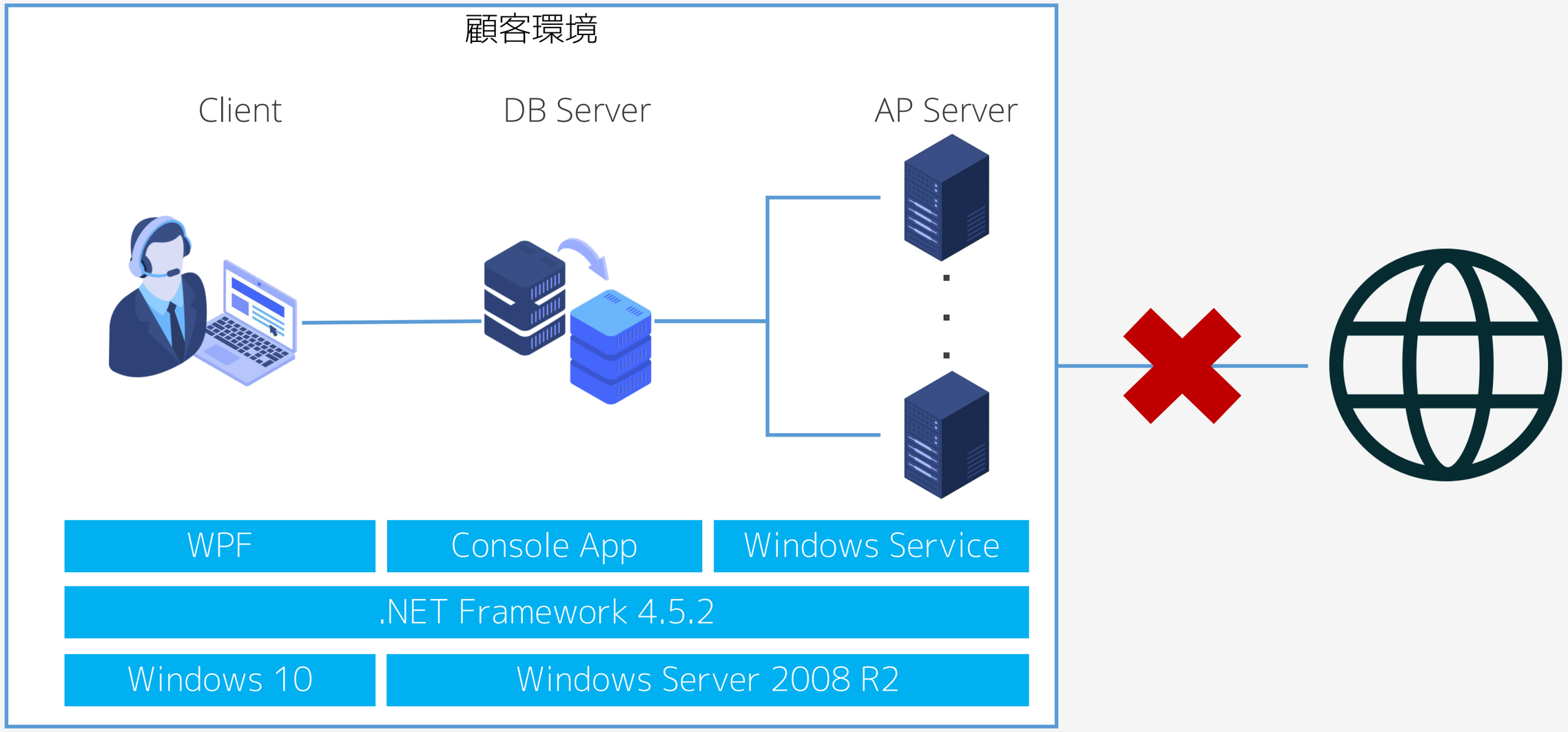
.NET 6 migration of a financial company's mission-critical system.

プロダクト概要

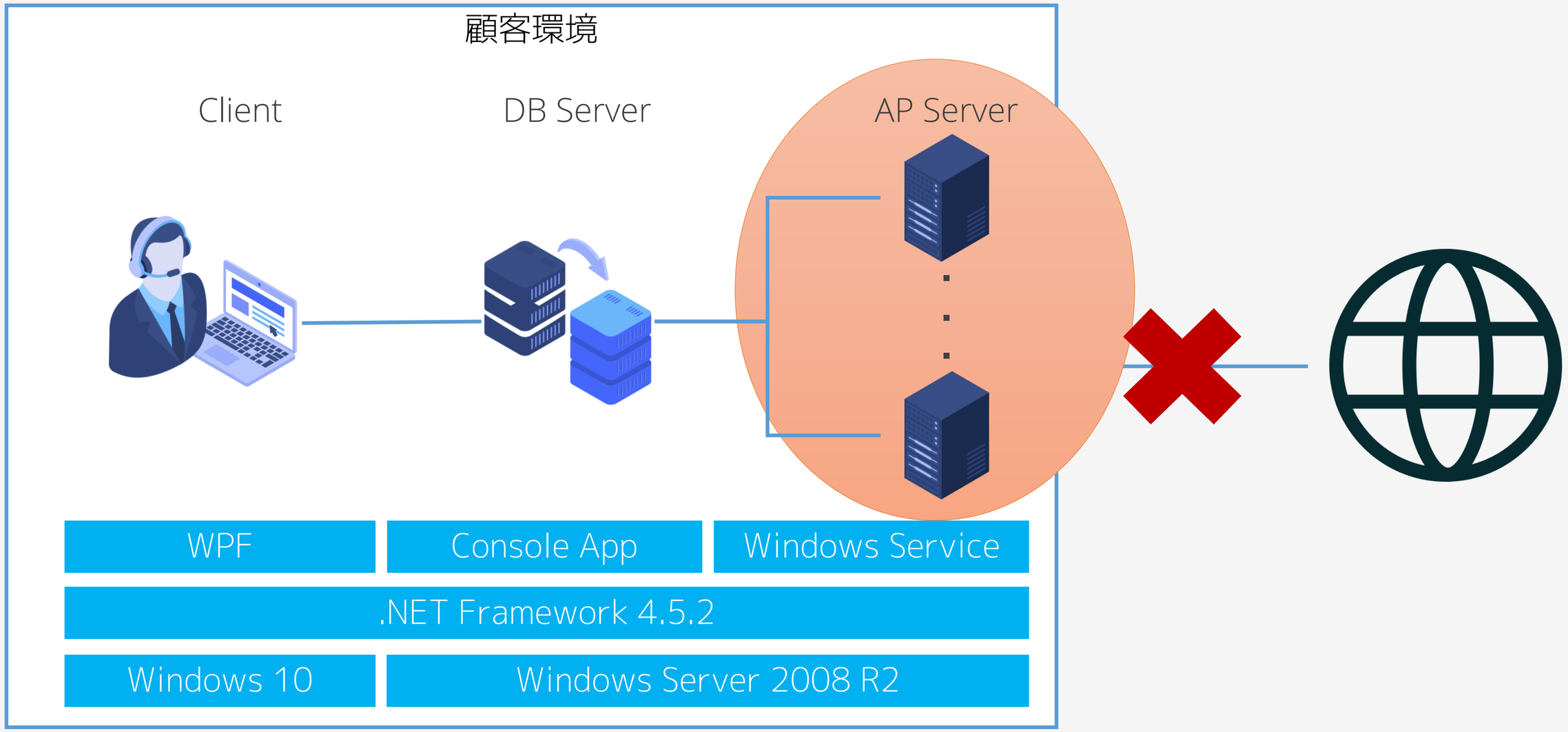
Product Overview

- 業種：金融
- 仮称：Image Document Workplace（以降IDW）
- 概要：画像化された、手書きの書類を扱うワークプレイス
- 略歴
  - 2013年4月に初期バージョンの開発開始
  - 当初の利用者数200人
  - 以後継続的に、年50～100人月程度開発
  - 徐々に適用業務も広がった
  - 2020年12月に基幹システムに昇格
  - 全国の営業店で受け取った書類はその場で画像化されIDW上へ

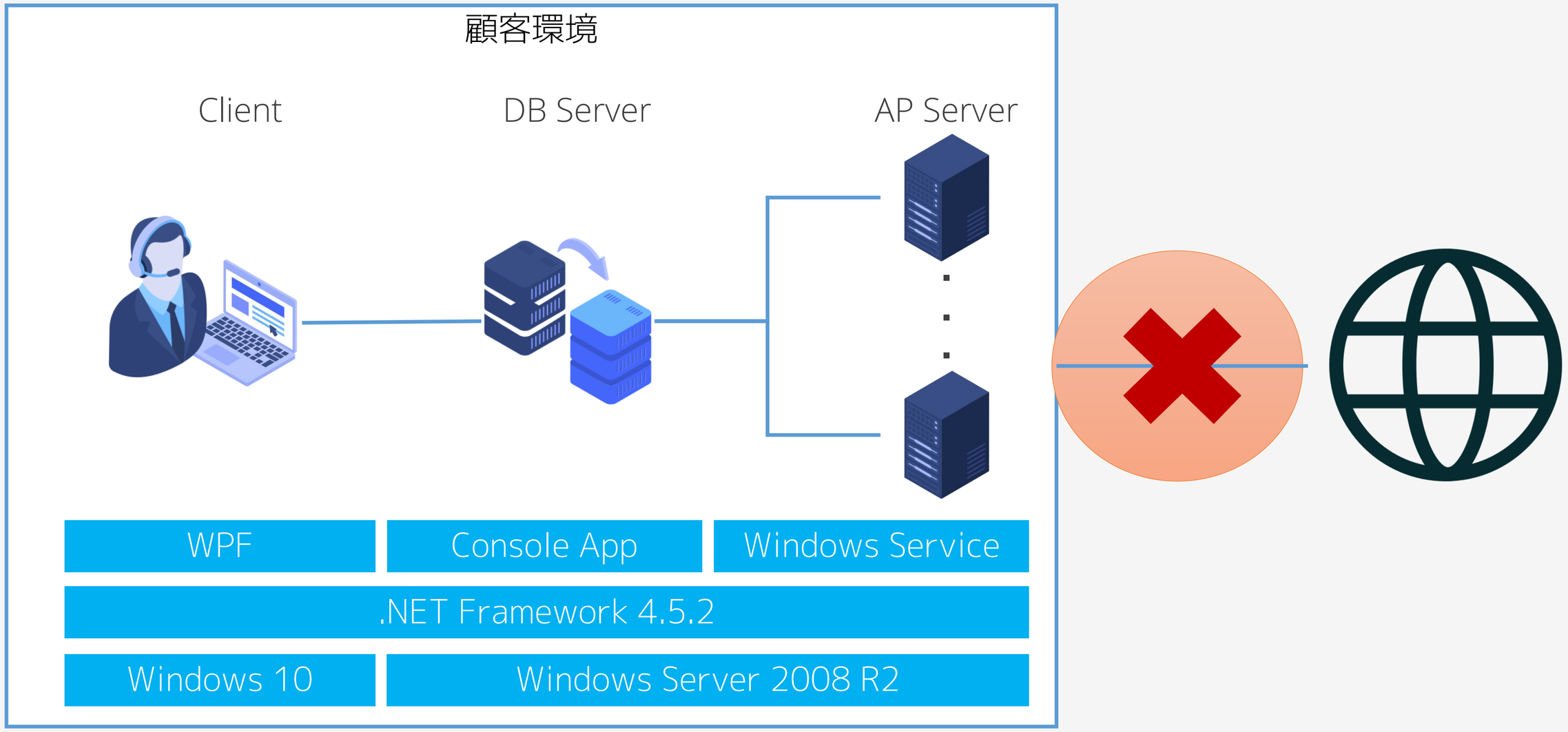
Product Overview（Before）



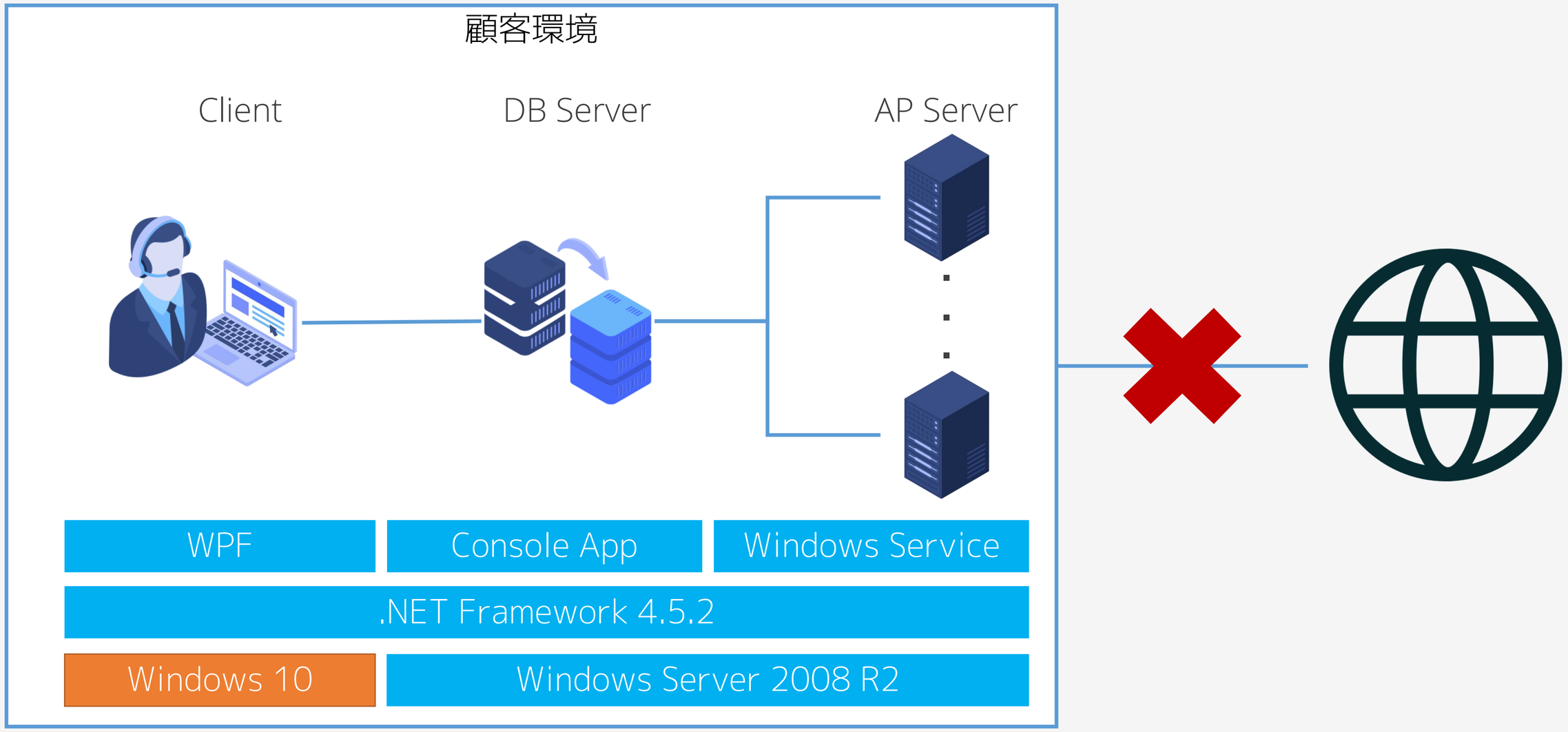
Product Overview（Before）



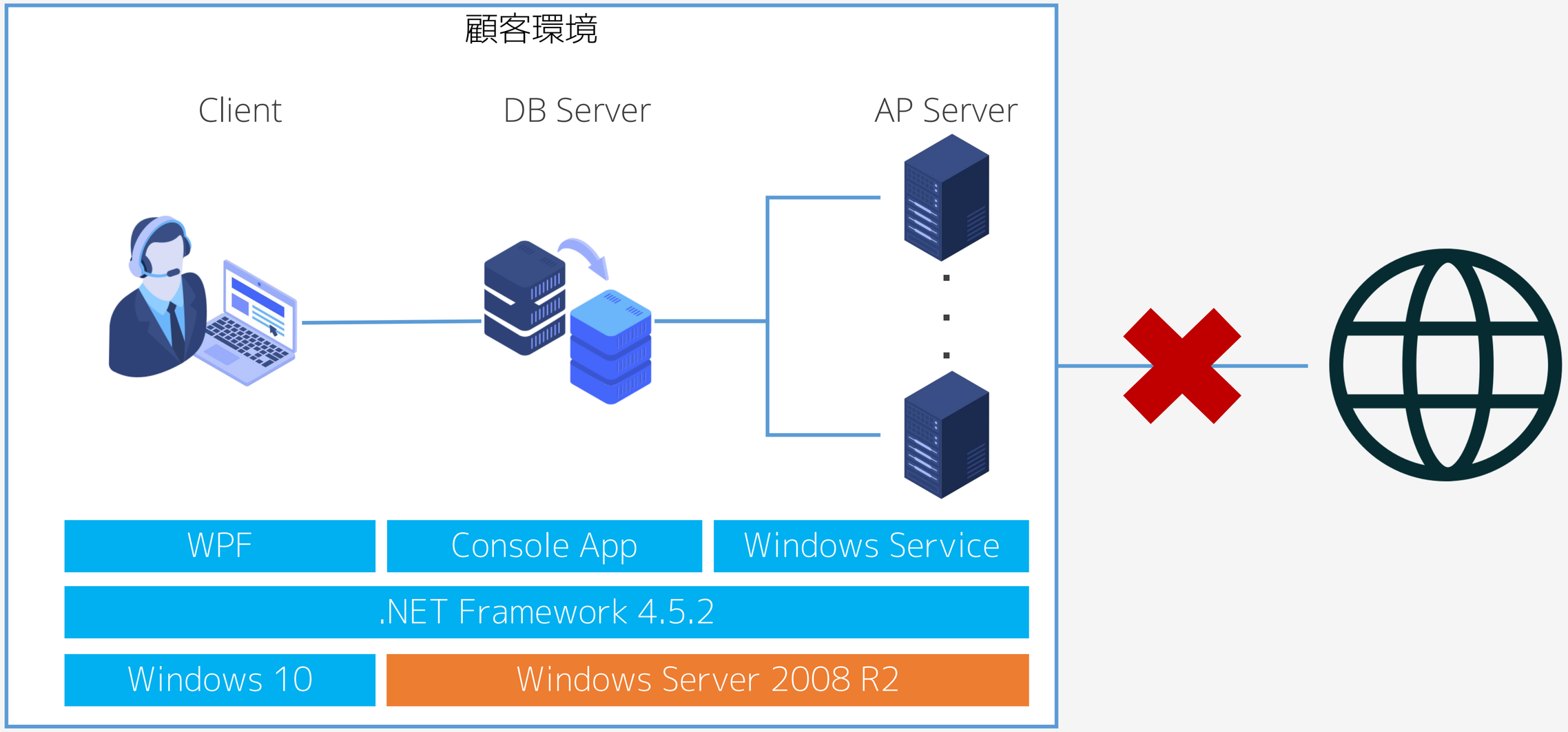
Product Overview（Before）



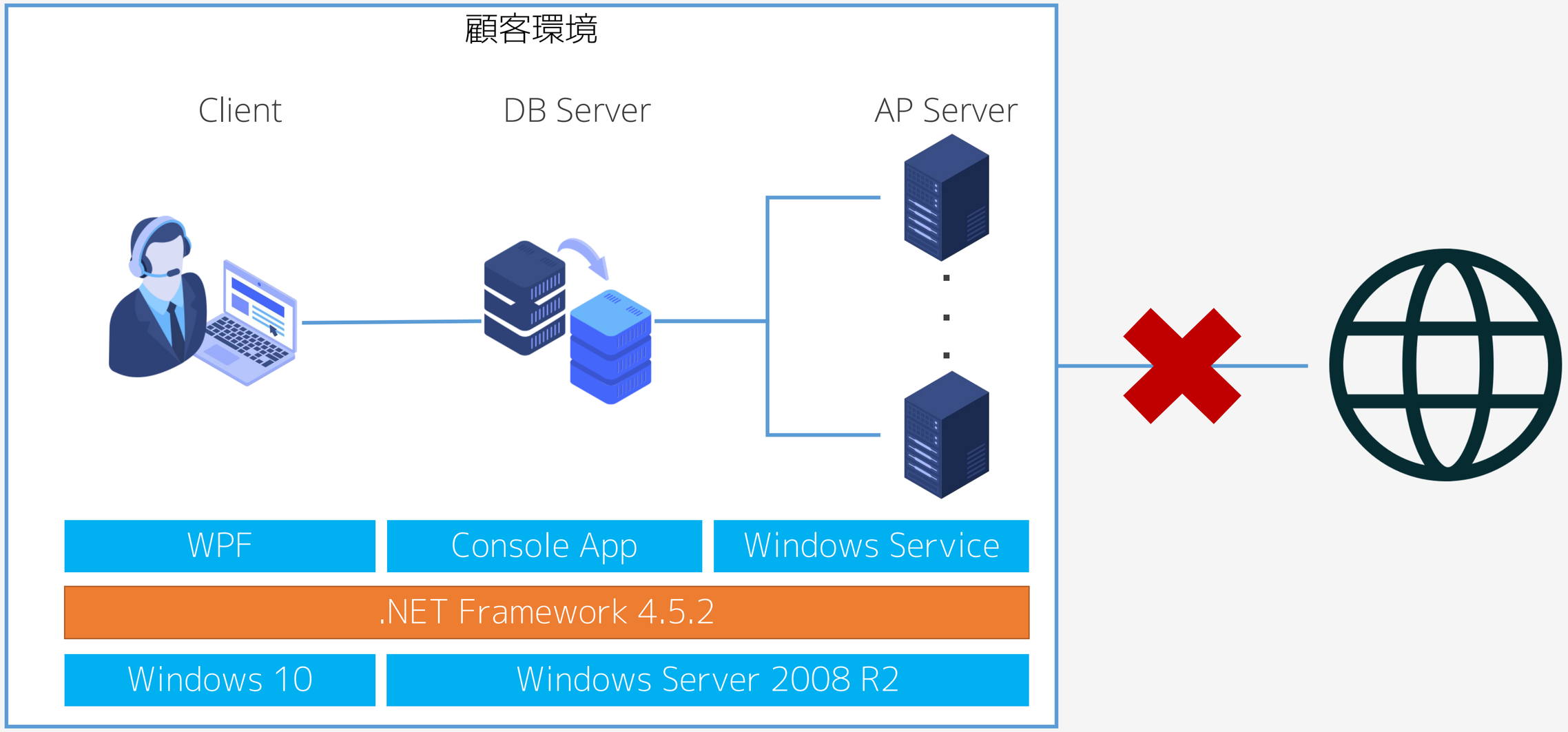
Product Overview（Before）



Product Overview（Before）



Product Overview（Before）



.NET 6 migration of a financial company's mission-critical system.

プロジェクト概要

Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施
6. クラウド移行

Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
   a. テスト品質の改善
   b. 業務・システム
4. 非機能要件を見直し
5. 生産性向上のため
6. ~~クラウド移行~~

マイナンバーを扱うシステムでコンプラ問題が解決できる確証がなかったこと

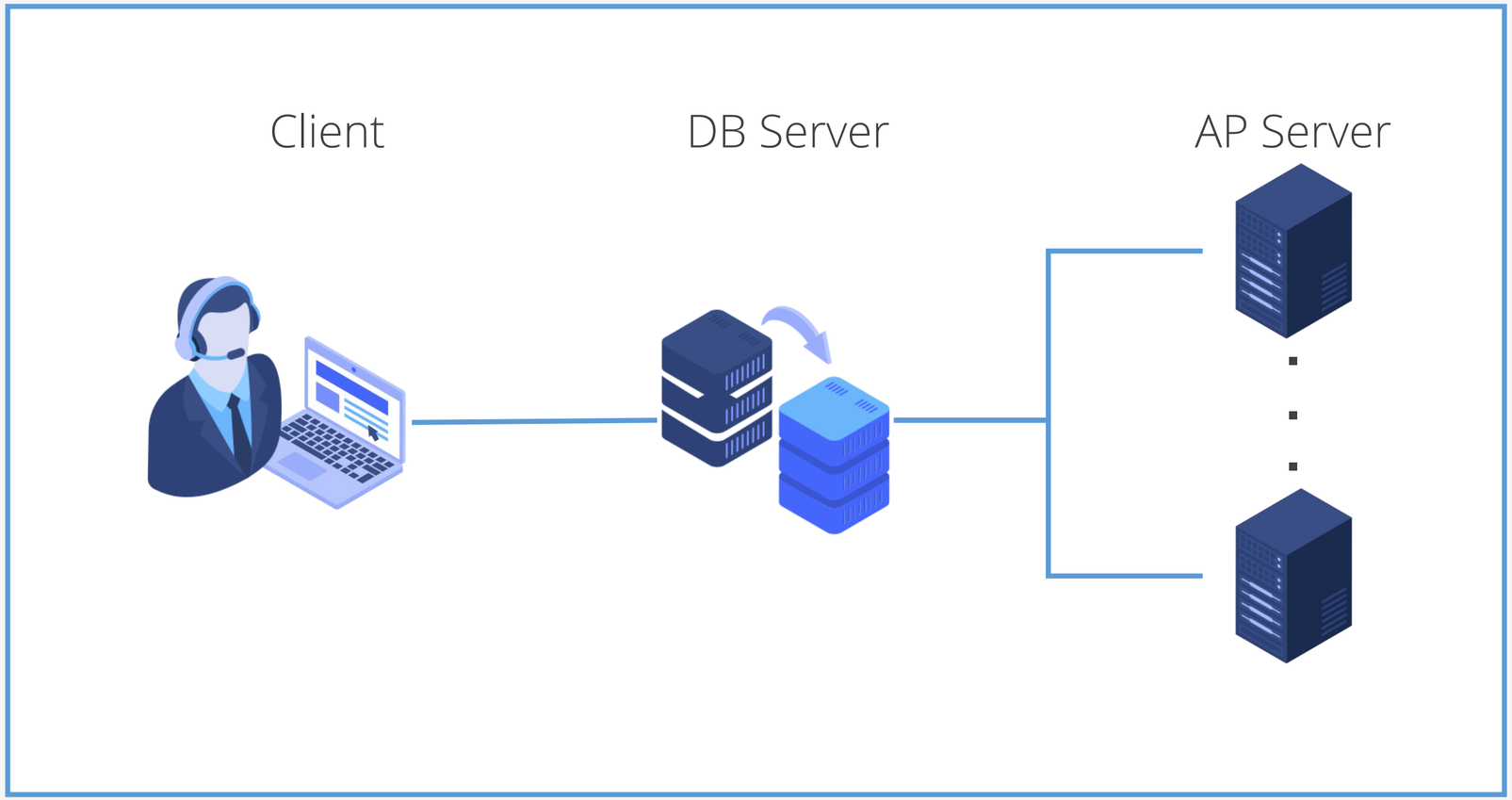
Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非
5. 生
6. ~~ク~~

実施

2020年の基幹システムリリース時に、「ガチ炎上」した
そのため顧客を含めたチーム全体にとって悲願

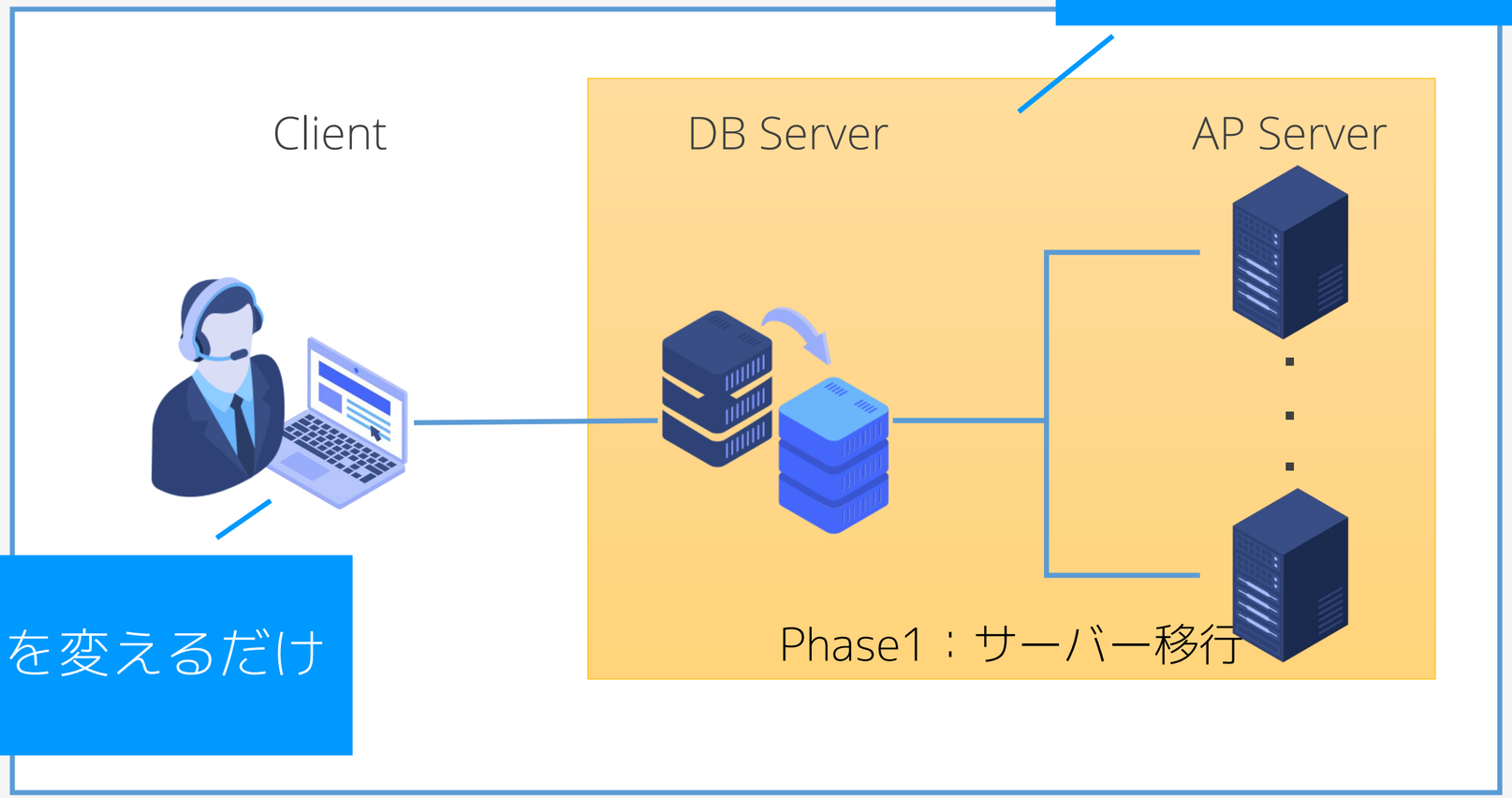
Project Schedule



フェーズ	2021年												2022年											
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12
サーバー移行																								
機能追加・改修																								

Project Schedule

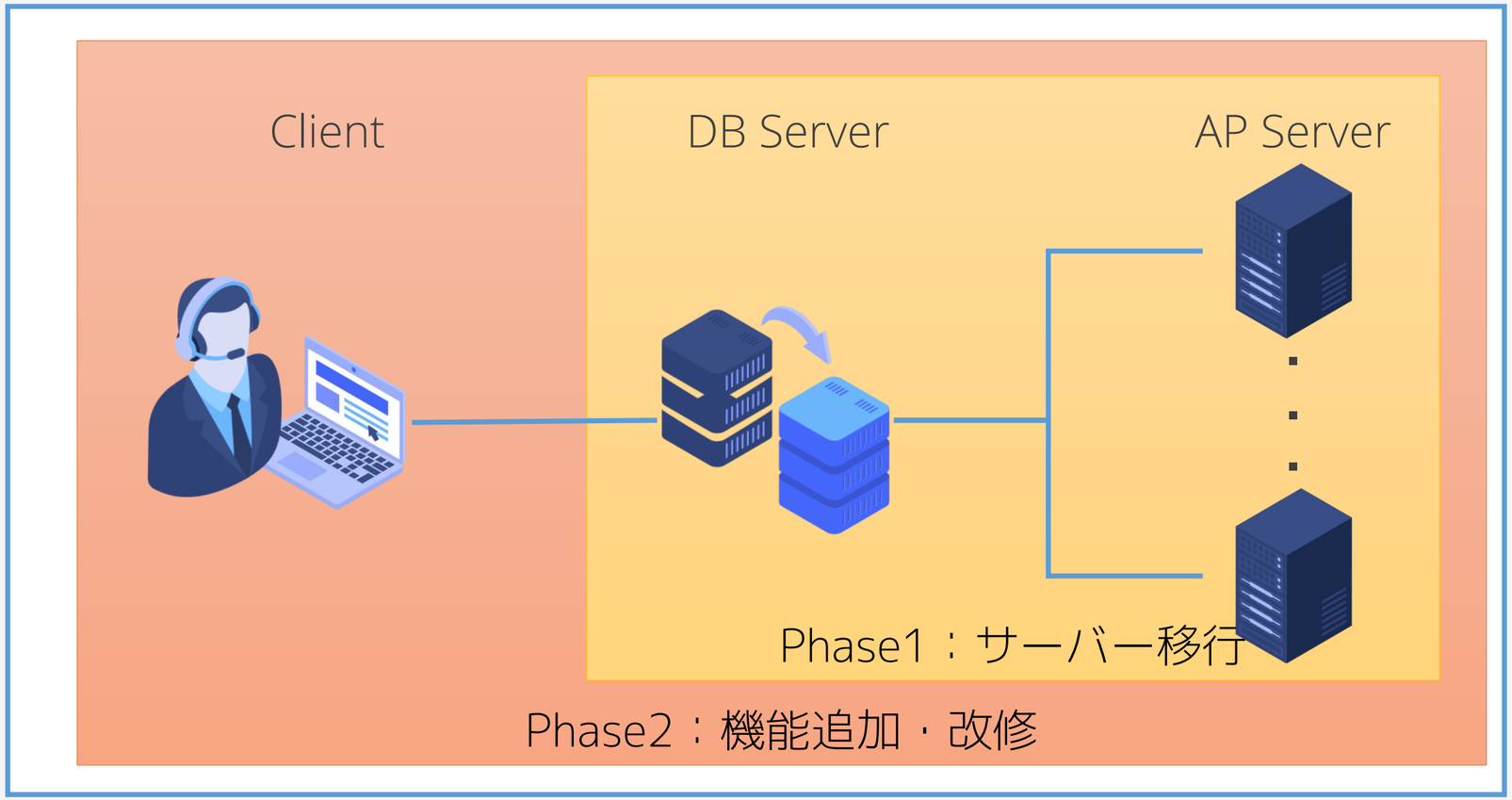
サーバー移行フェーズではOS・ミドルウェア・Runtimeの変更



接続の向きを変えるだけ

フェーズ	2021年												2022年											
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12
サーバー移行																								
機能追加・改修																								

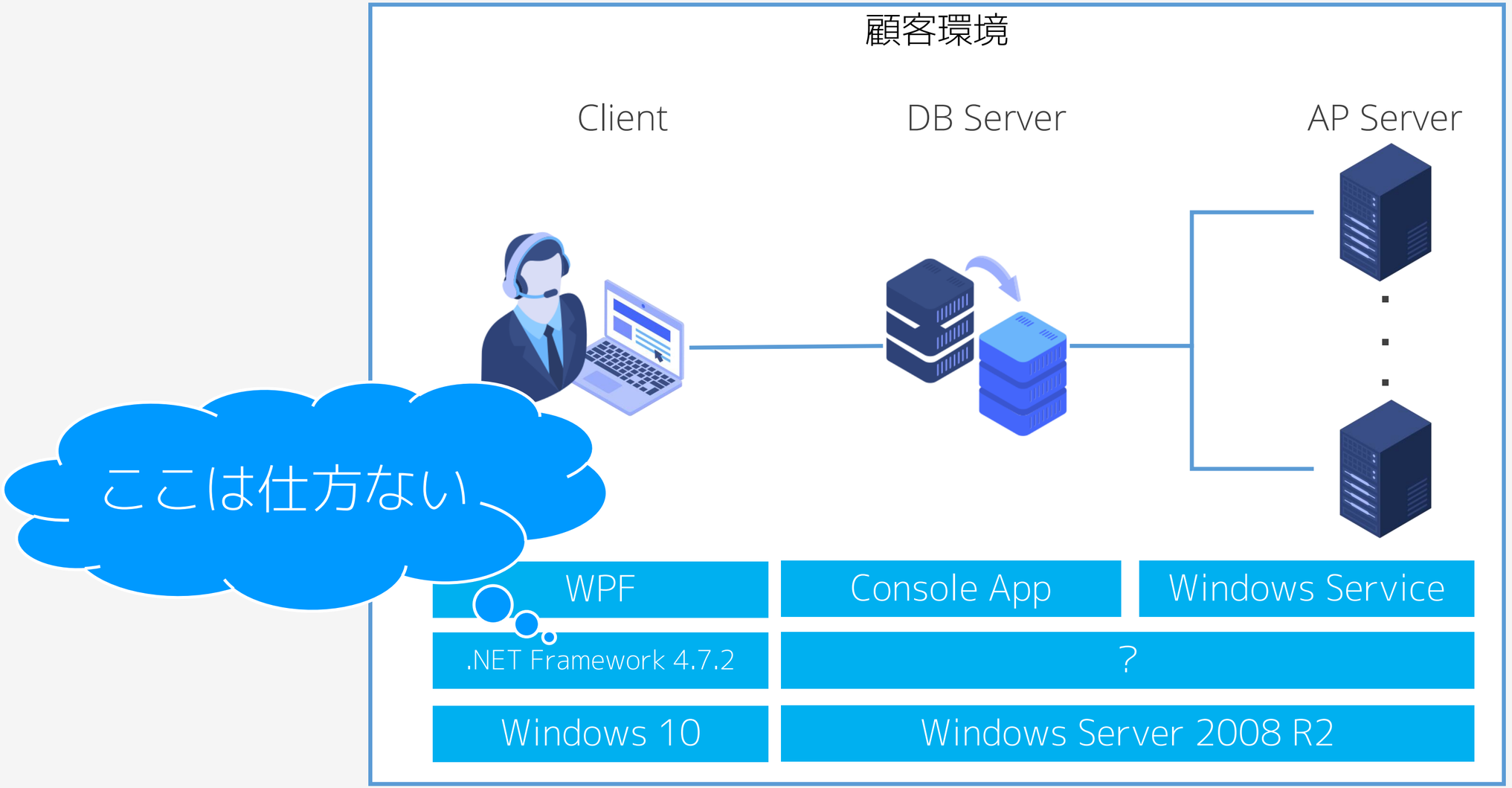
Project Schedule



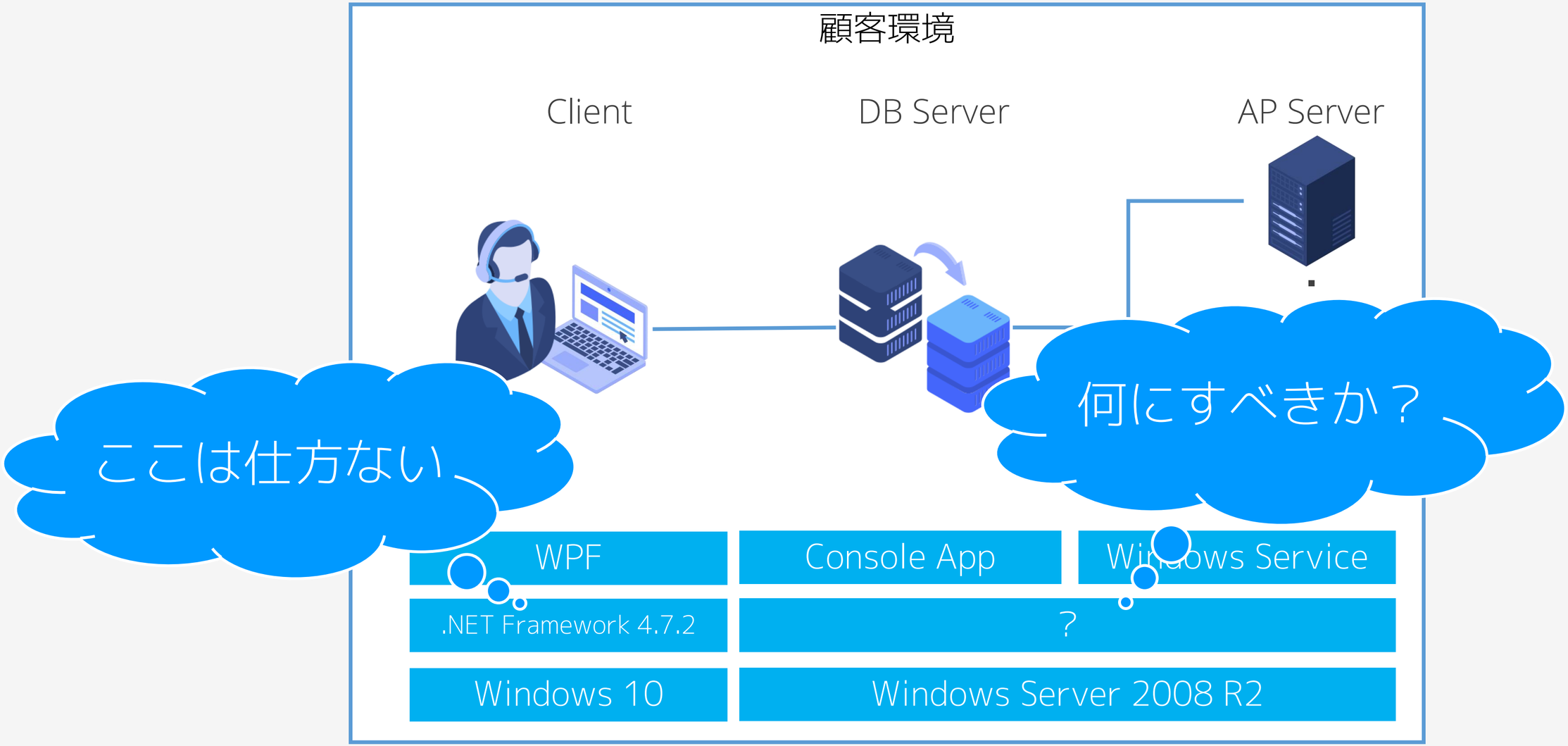
フェーズ	2021年												2022年											
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12
サーバー移行																								
機能追加・改修																								

で、.NETなに使おう？

.NETどうする？



.NETどうする？



悩ましいスケジュール

項目	2021年												2022年		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
.NET 6 Release											★				
物理移行フェーズ															
要件定義～結合テスト	■	■	■	■	■	■	■	■	■	■	■	■			
システムテスト													■	■	
受入テスト														■	■
リリース															★

２つの選択肢

1. .NET Framework 4.8で永久に塩漬けする
2. .NET 6で２年おきにリプレースし続ける

ここまでの流れから
4.8じゃないの？

.NETと一緒にキャリアの塩付けはいやだ！

最新.NETを適用していくメリット

1. 最新.NET自体の強み
    1. 性能向上
    2. 品質向上（nullableとか）
    3. 生産性向上（便利機能は今後.NET向けに）
2. 将来的なクラウド利用に対する布石
3. エンジニアのモチベーションの向上
    1. 生産性向上
    2. 品質向上

最新.NETを適用していくメリット

1. 最新.NET自体の強み
    1. 性能向上
    2. 生産性向上
    3. 品質向上
2. 将来的なクラウド利用に対する布石
3. エンジニアのモチベーションの向上

「幸福感の高い社員の創造性は3倍、生産性は31％、売上は37％高い上に、欠勤率が41％、離職率が59％低く、業務上の事故が70％少ない」

慶應義塾大学大学院 - 前野隆司教授
https://www.lifehacker.jp/article/245162great_place_to_work

弊社「働き方変革」

https://kagayakutelework.jp/seminar/2020/pdf/tokyo04/ricoh201223.pdf

弊社「働き方変革」

https://kagayakutelework.jp/seminar/2020/pdf/tokyo04/ricoh201223.pdf

弊社「働き方変革」

https://kagayakutelework.jp/seminar/2020/pdf/tokyo04/ricoh201223.pdf

エンジニアの悪癖

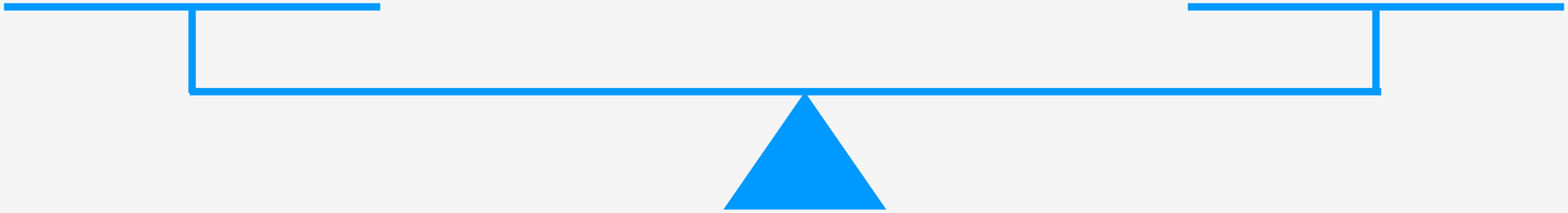
これでは顧客を説得できない

.NET 6 migration of a financial company's mission-critical system.

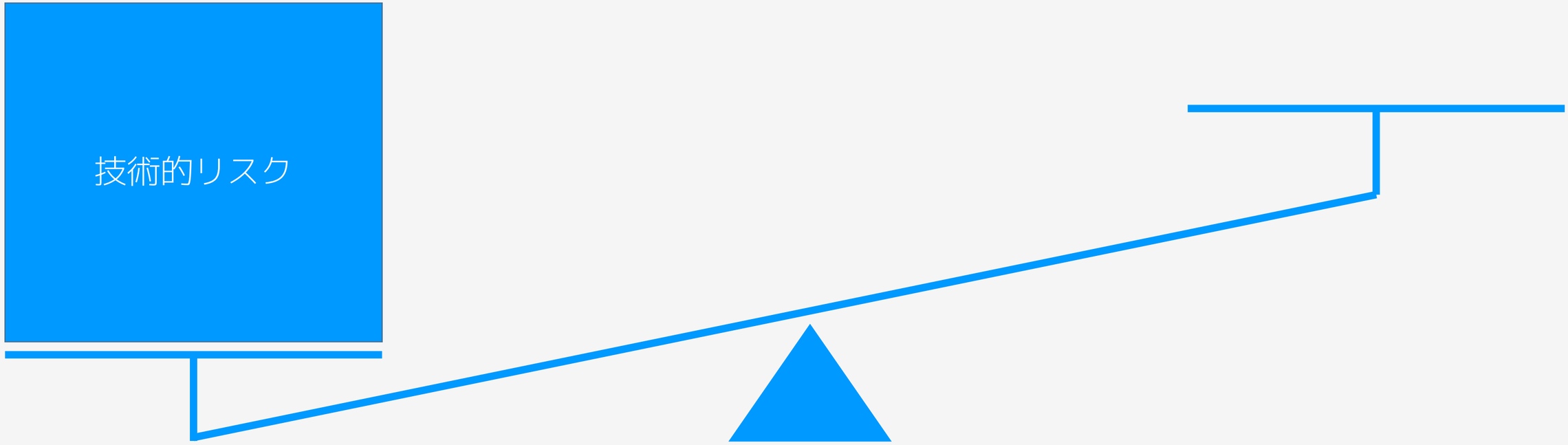
顧客をどう説得したか

顧客の視点から考えてみる

顧客から見た.NET6



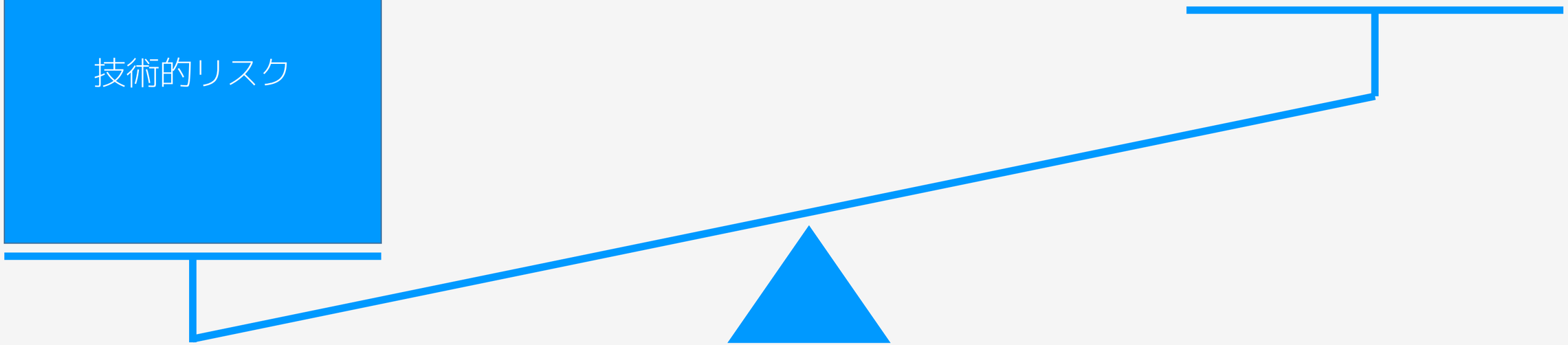
顧客から見た.NET6



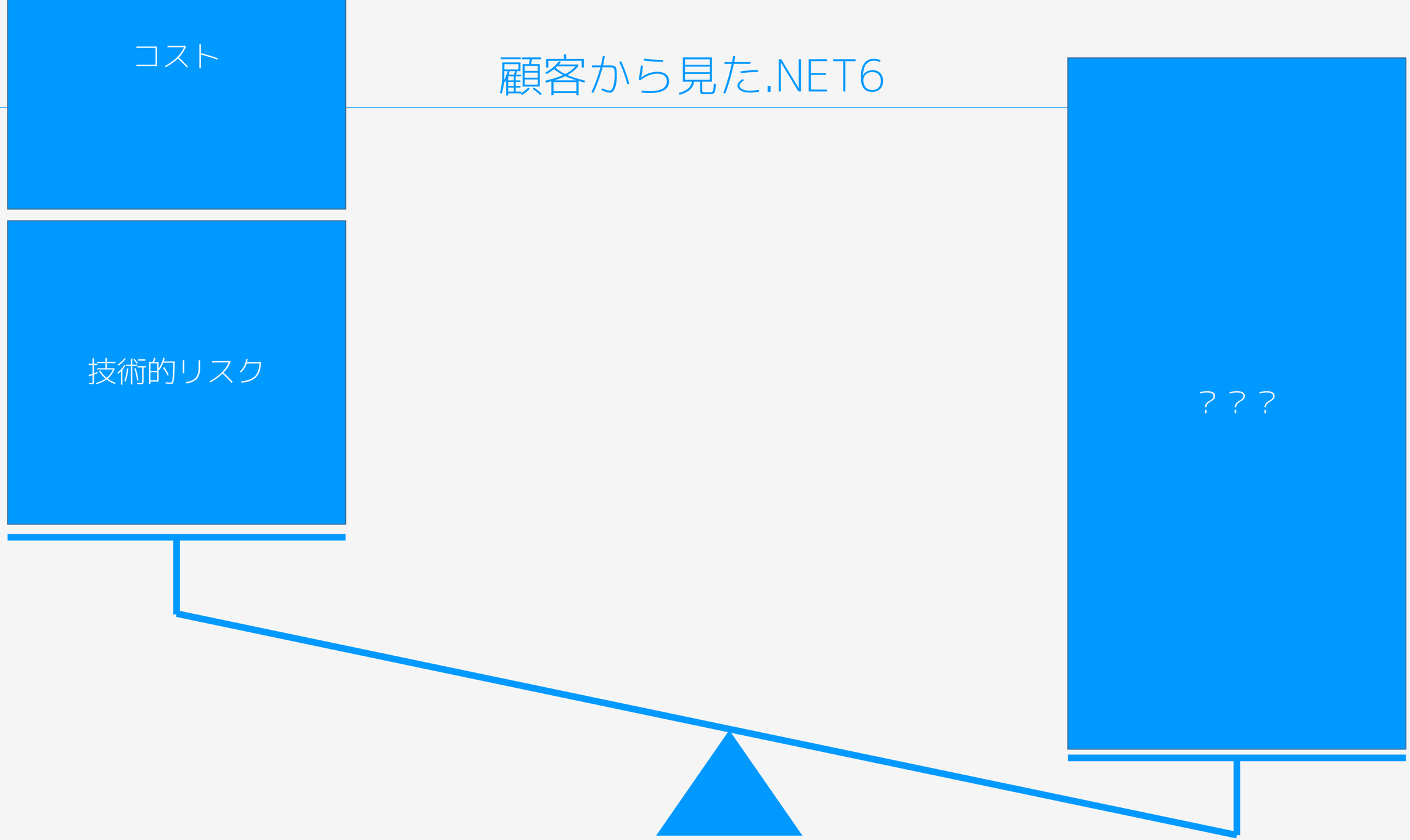
顧客から見た.NET6

コスト

技術的リスク



顧客から見た.NET6

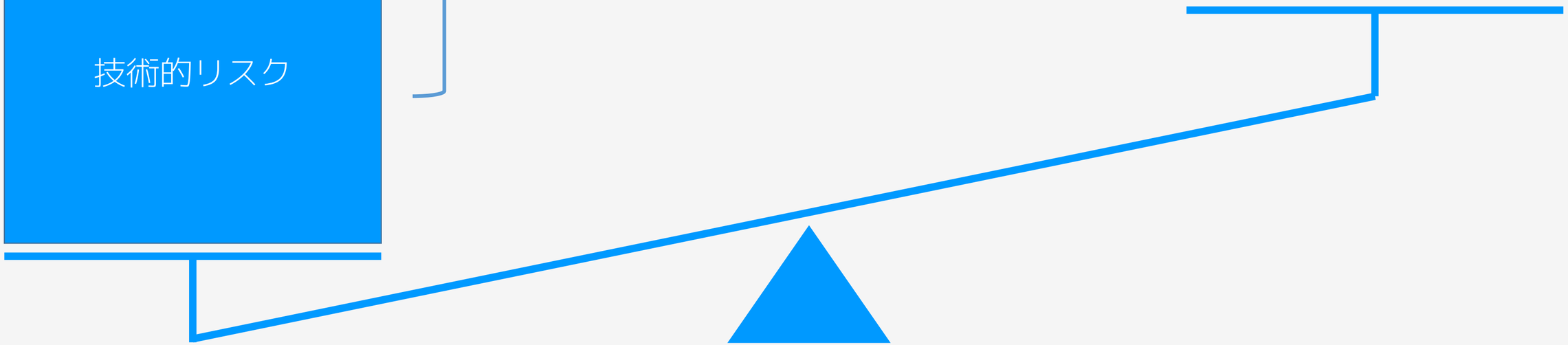


顧客から見た.NET6

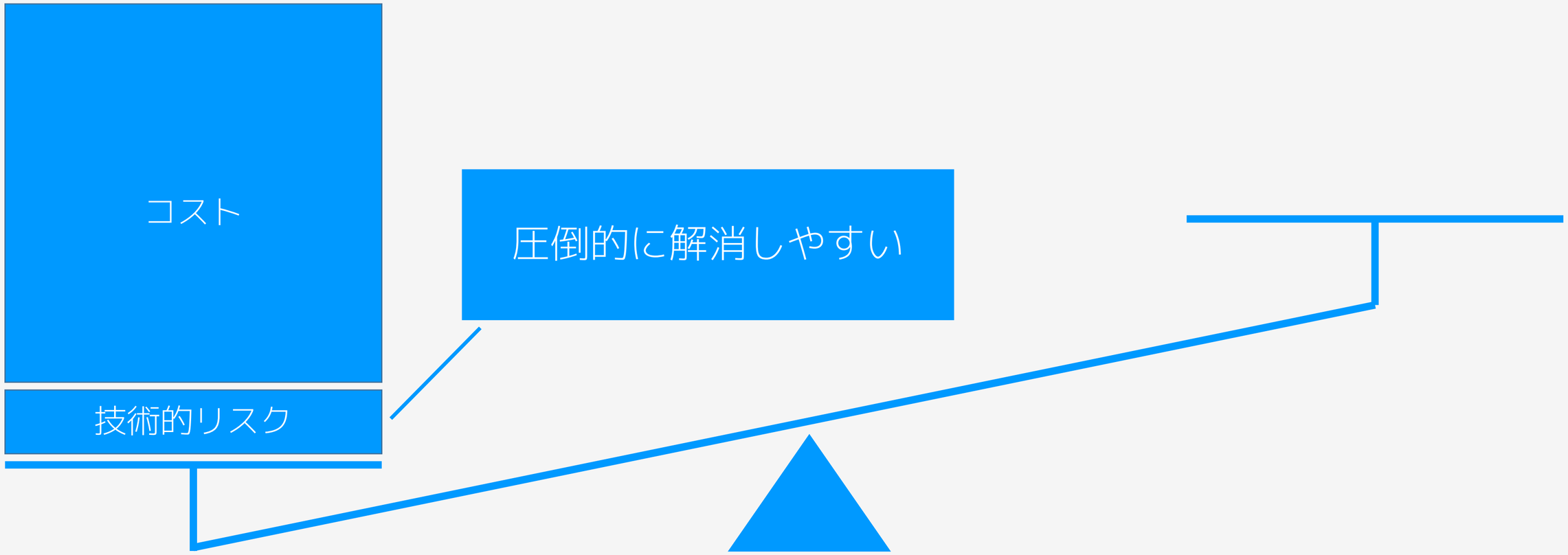
コスト

技術的リスク

同程度のウェイトか？



顧客から見た.NET6



前提として

前提

- 今回.NET 6にアップデート することは、大きなリスクはない
  1. .NET Frameworkに依存する機能は「ほぼ」無い
  2. 十分な体制
  3. 十分な時間
  4. 十分なコスト
- 2年おきに更新しつづけることがリスク

そのうえで・・・

技術的リスクが低いと考える理由

1. .NET Frameworkから.NETへの移行は破壊的変更が多い
2. .NET 6から.NET 8への移行は基本的に後方互換性を期待できる
3. リスクの大きさ
   net4.5.2⇒net6.0 ＞＞＞超えられない壁＞＞＞ net6.0⇒net8.0
4. つぎの2つが用意できれば、今後のリスクは非常に小さい
   1. 再現性あるテスト⇒自動テスト
   2. 十分な品質のテスト

Project Goal

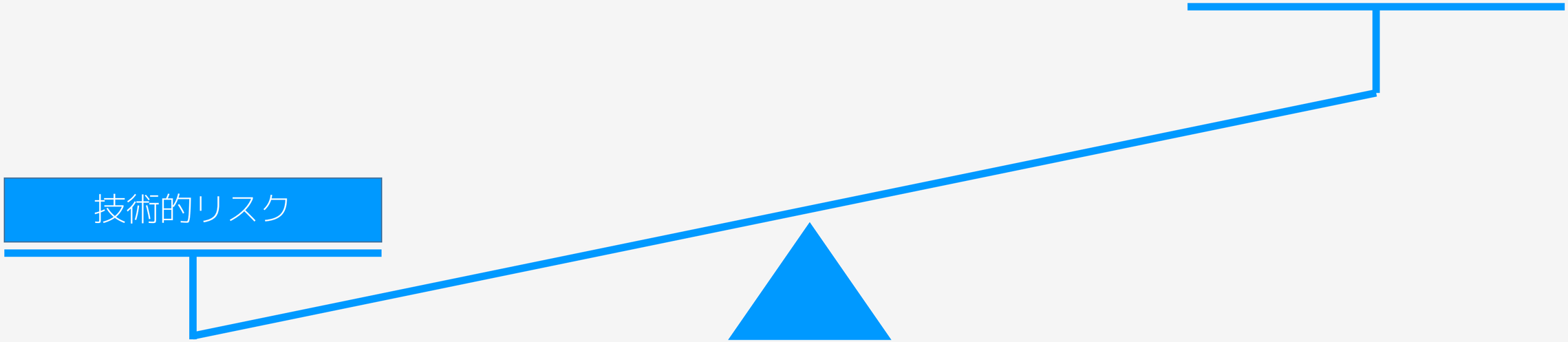
1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

Project Goal

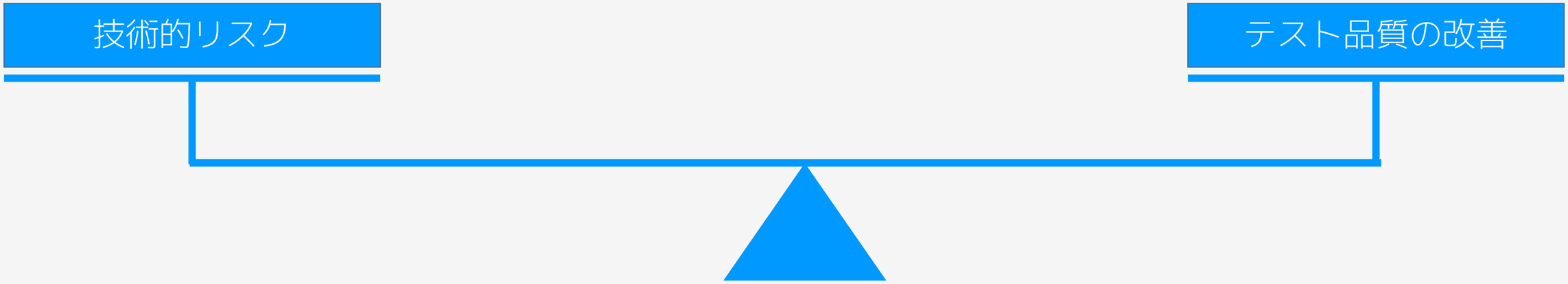
1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

※ テストの詳細は後半で

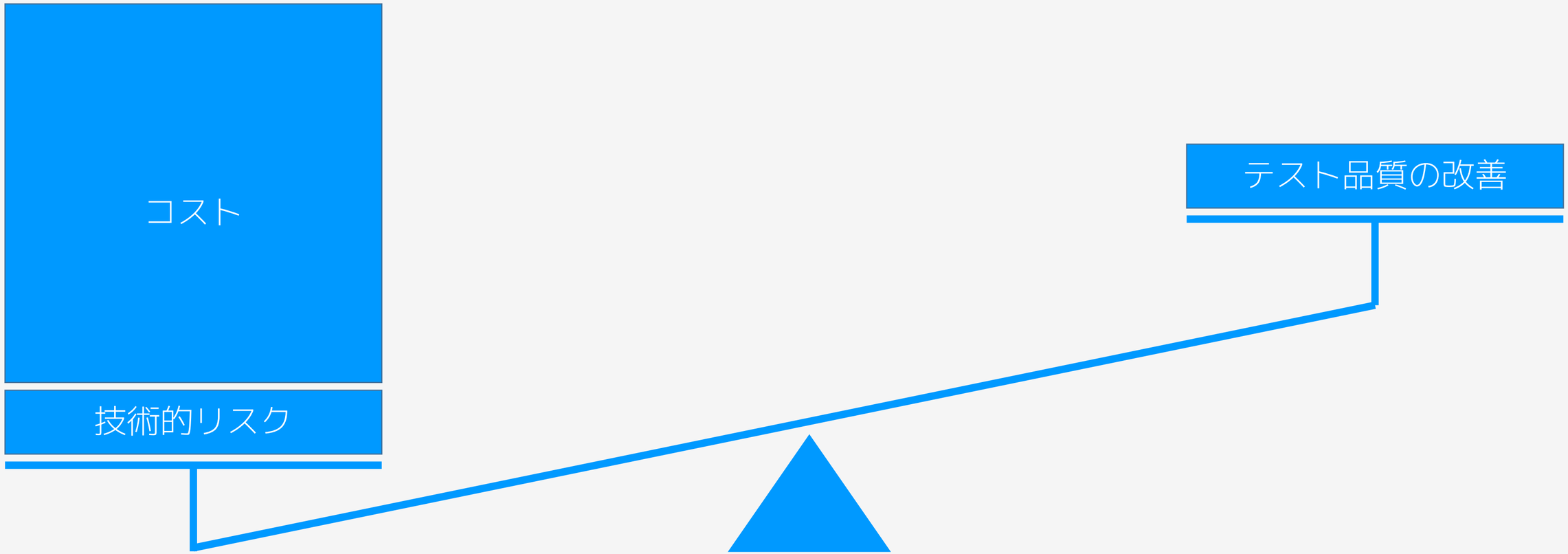
顧客から見た.NET6



顧客から見た.NET6

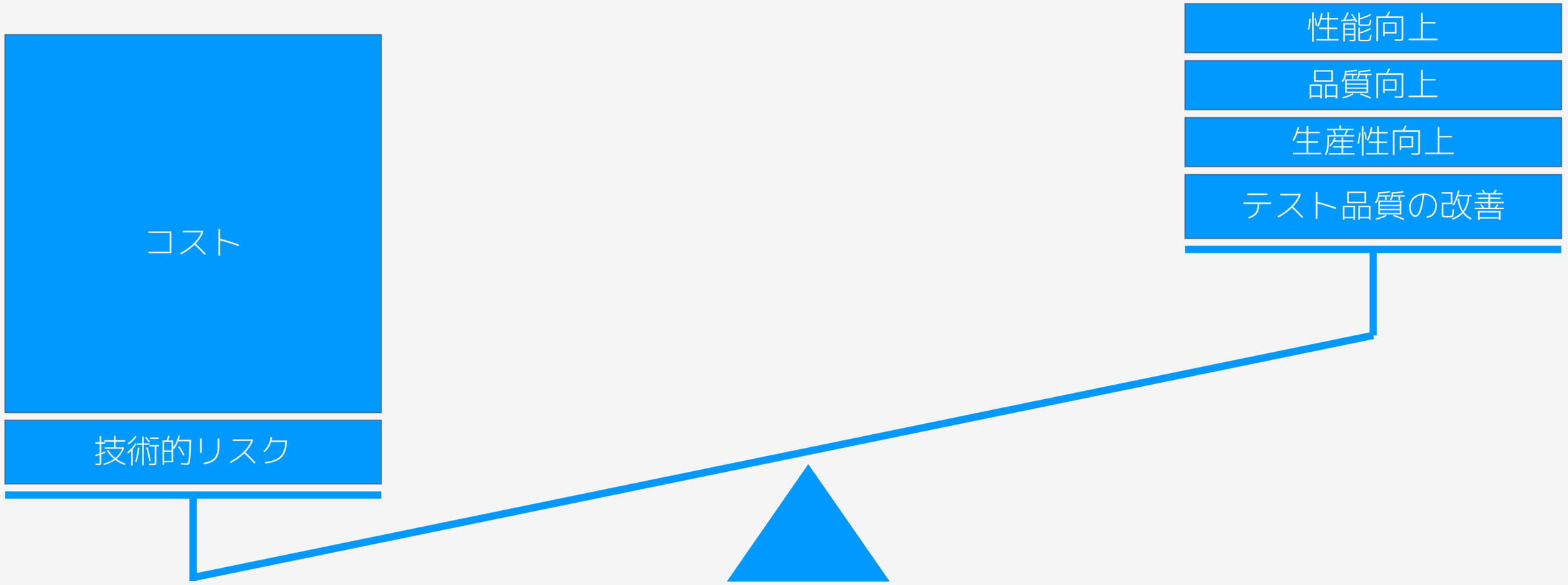


顧客から見た.NET6

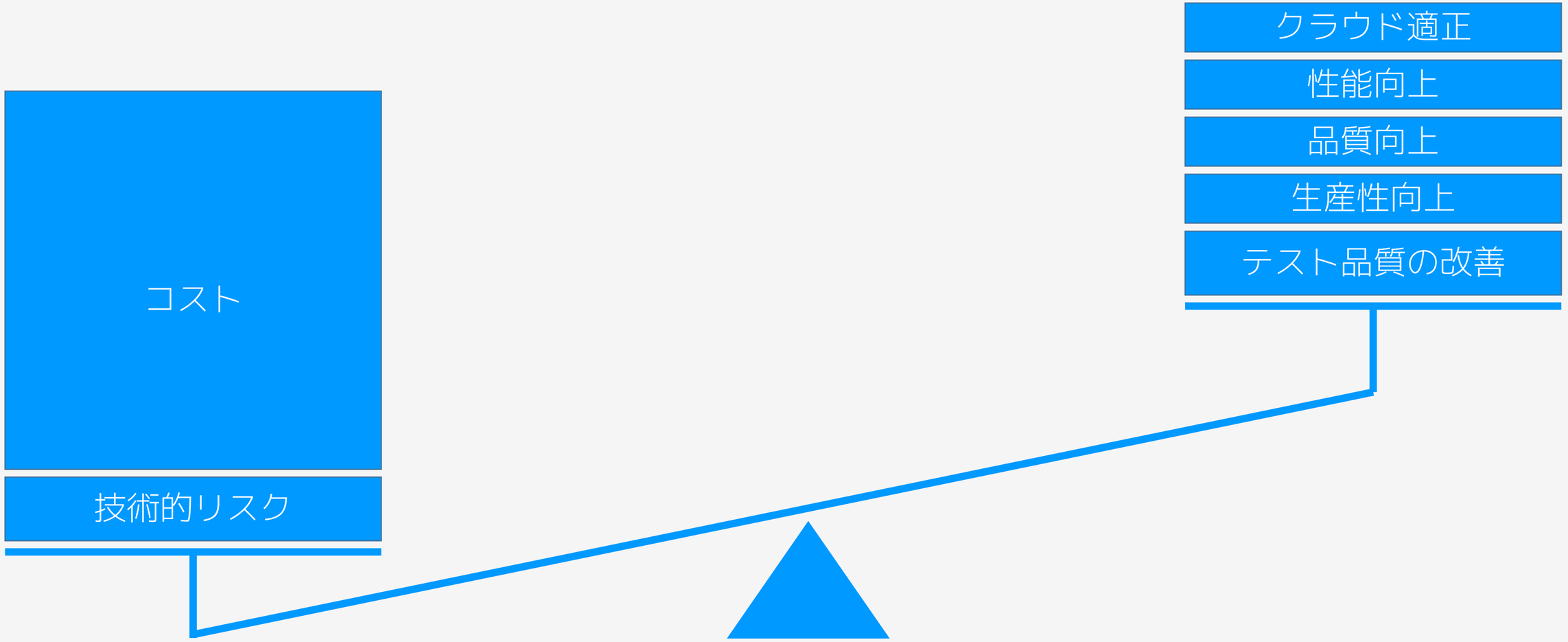


何をもって倒す？

顧客から見た.NET6

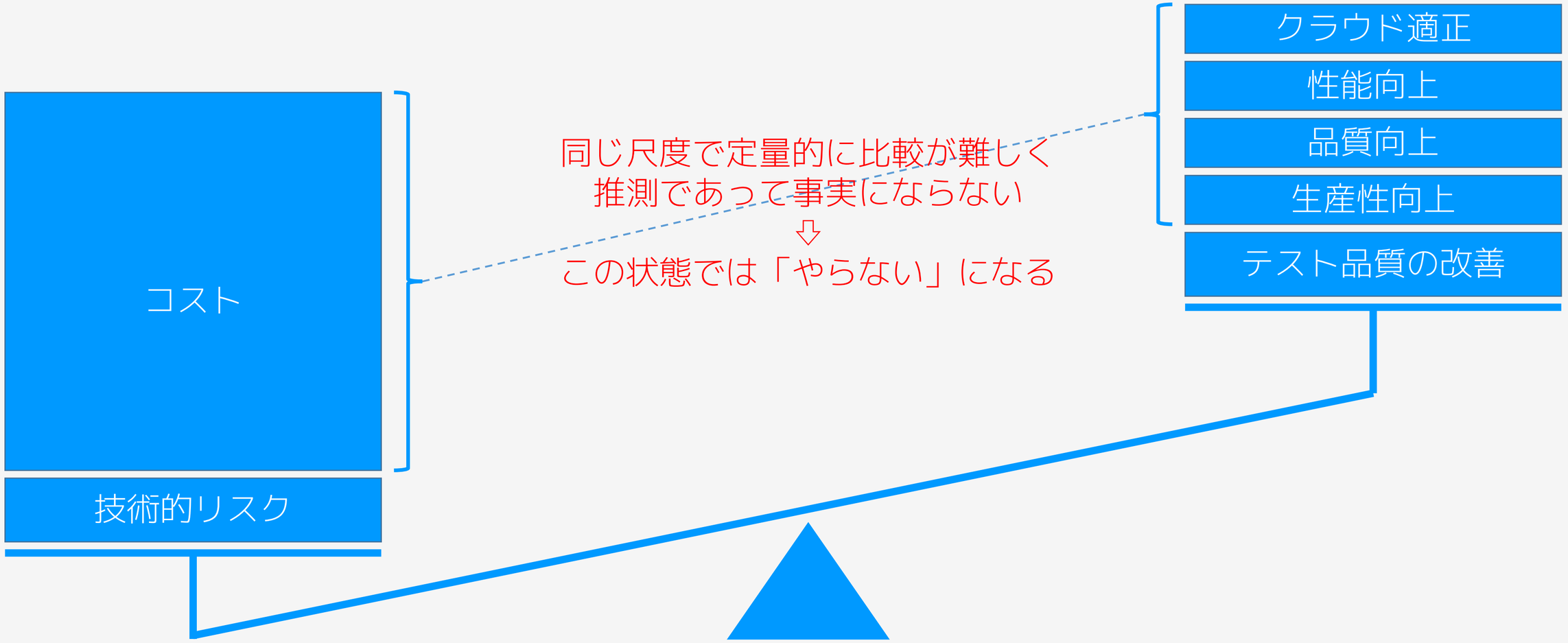


顧客から見た.NET6

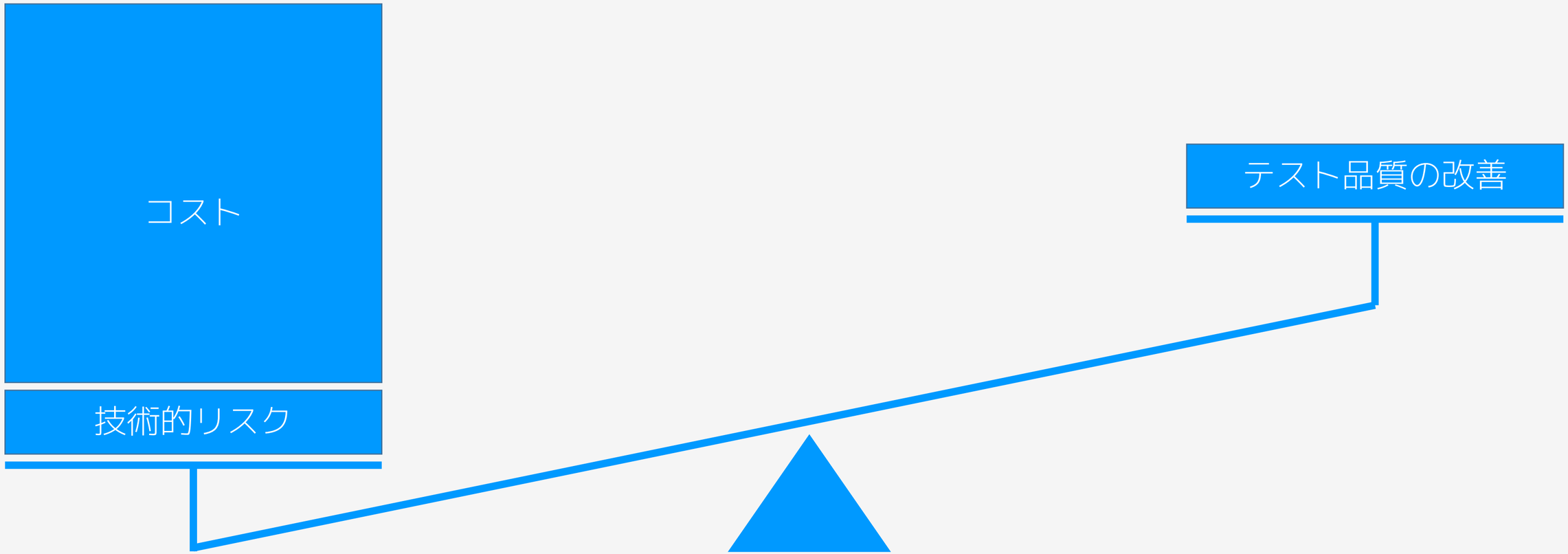


右に傾いたりしない

顧客から見た.NET6



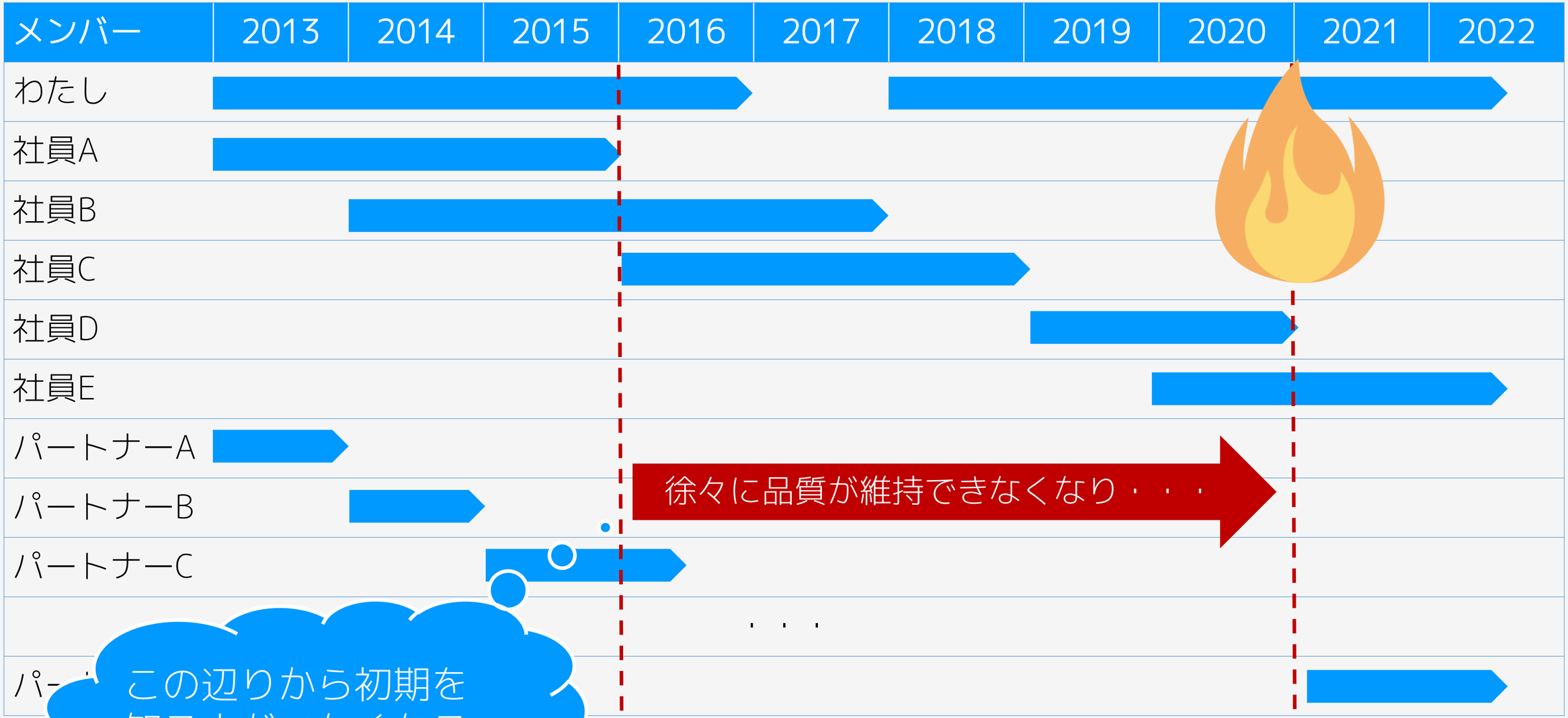
顧客から見た.NET6



Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

メンバー育成がひとつのGoalとなった理由



炎上の要因

- 開発サイドと顧客サイドの双方に問題があった
- 開発サイドの問題
  - 未改修の機能が停止した
  - 改修箇所に依存する機能への影響想定が不十分だった
  - そもそも想定可能なメンバーが私一人しかいなかった

業務・システムともに習熟したメンバーの育成

Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

一度育成して終わりではない
育成後は維持する必要がある

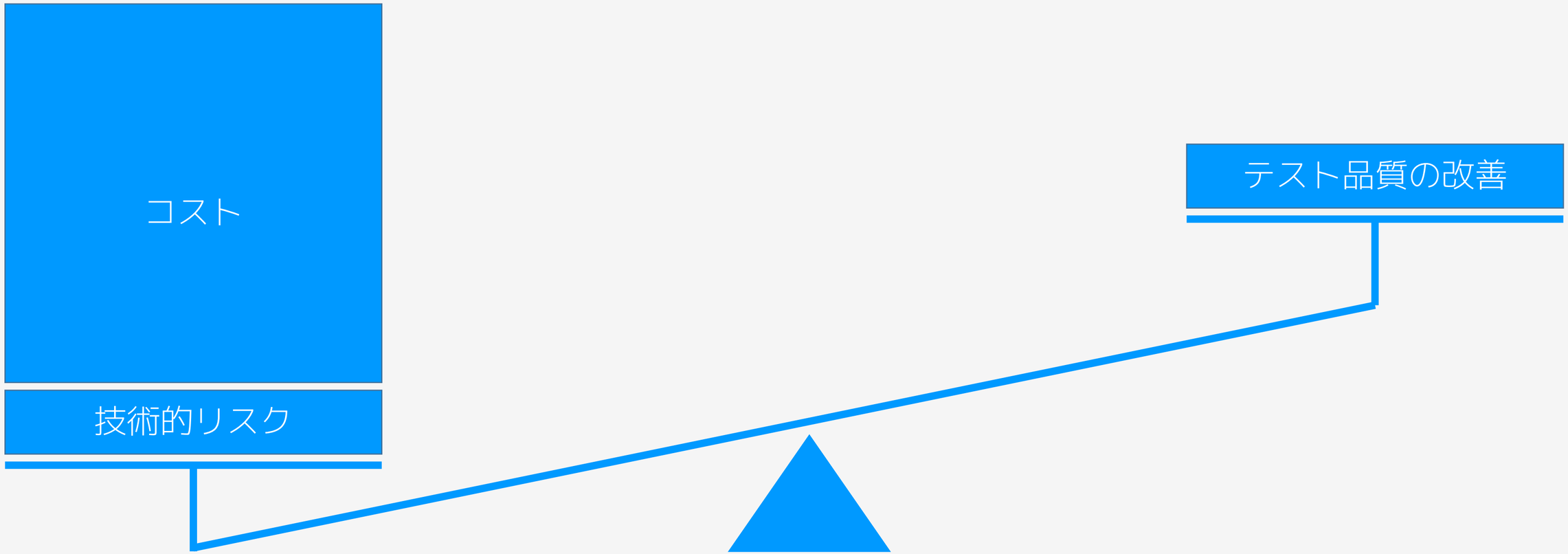
定期的なアップデートで維持できないか？

コストの解消とメンバー育成

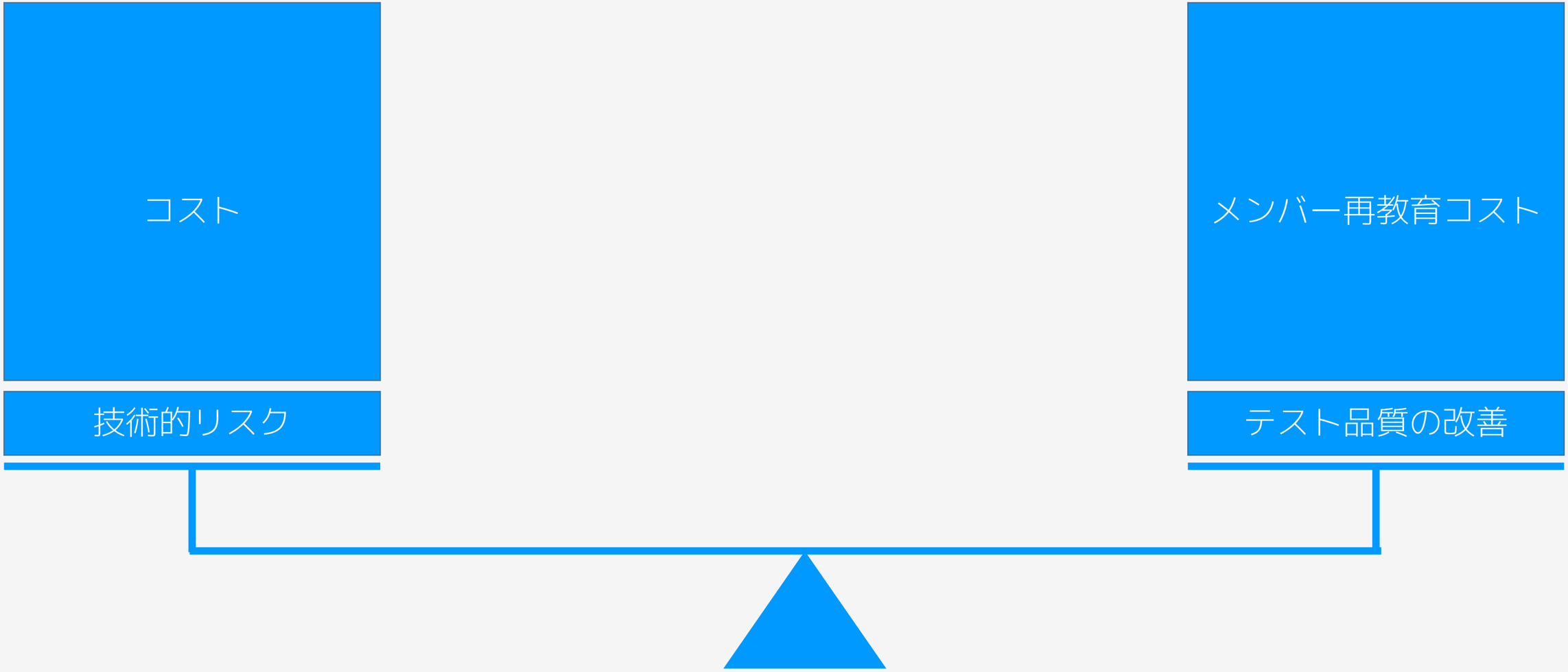
- およそ2年～3年でメンバーが入れ替わる
- 2年おきに継続的にメンバーを再教育する
- もっとも最適な教育はなにか？

- ビジネス シナリオ テストの実施
- .NET のライフサイクルは3年
- LTSのリリース間隔は2年
- 同時に.NETをアップデートすることで実質ノーコストでテスト可能

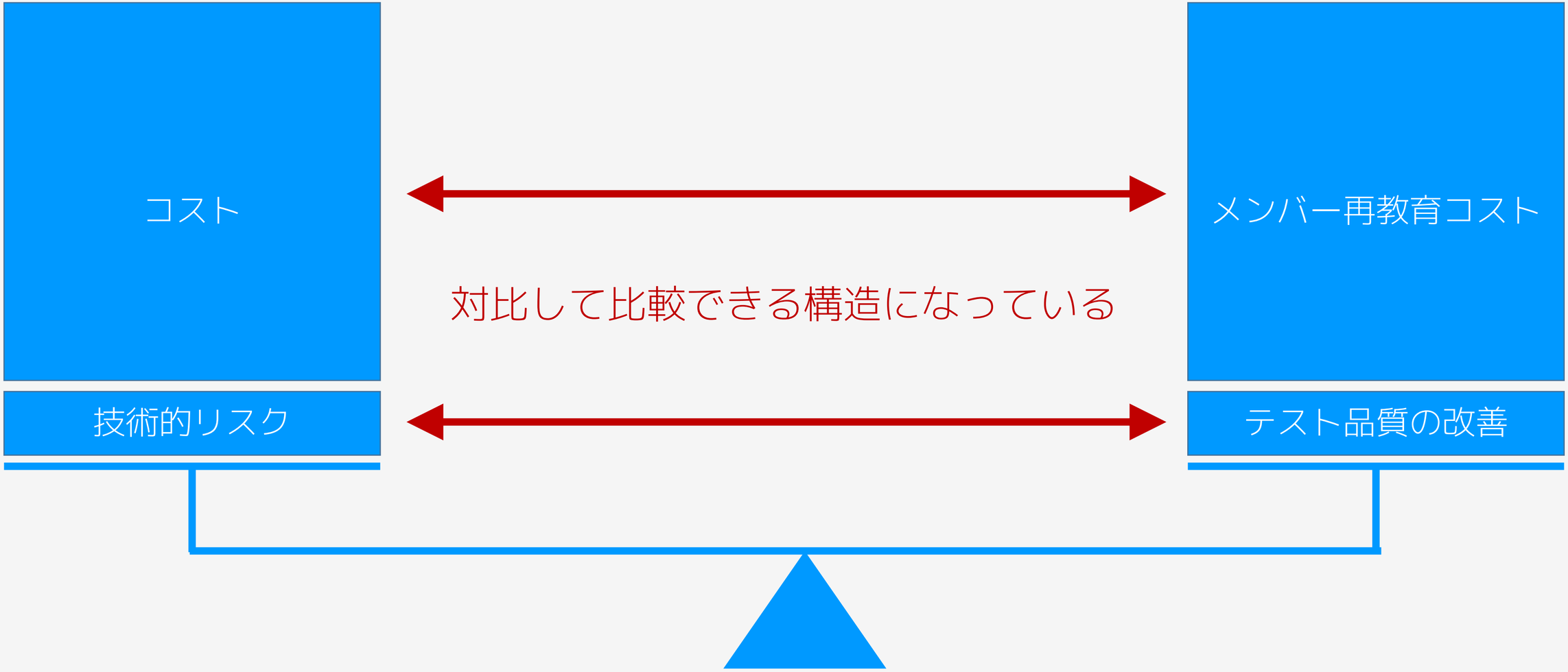
顧客から見た.NET6



顧客から見た.NET6

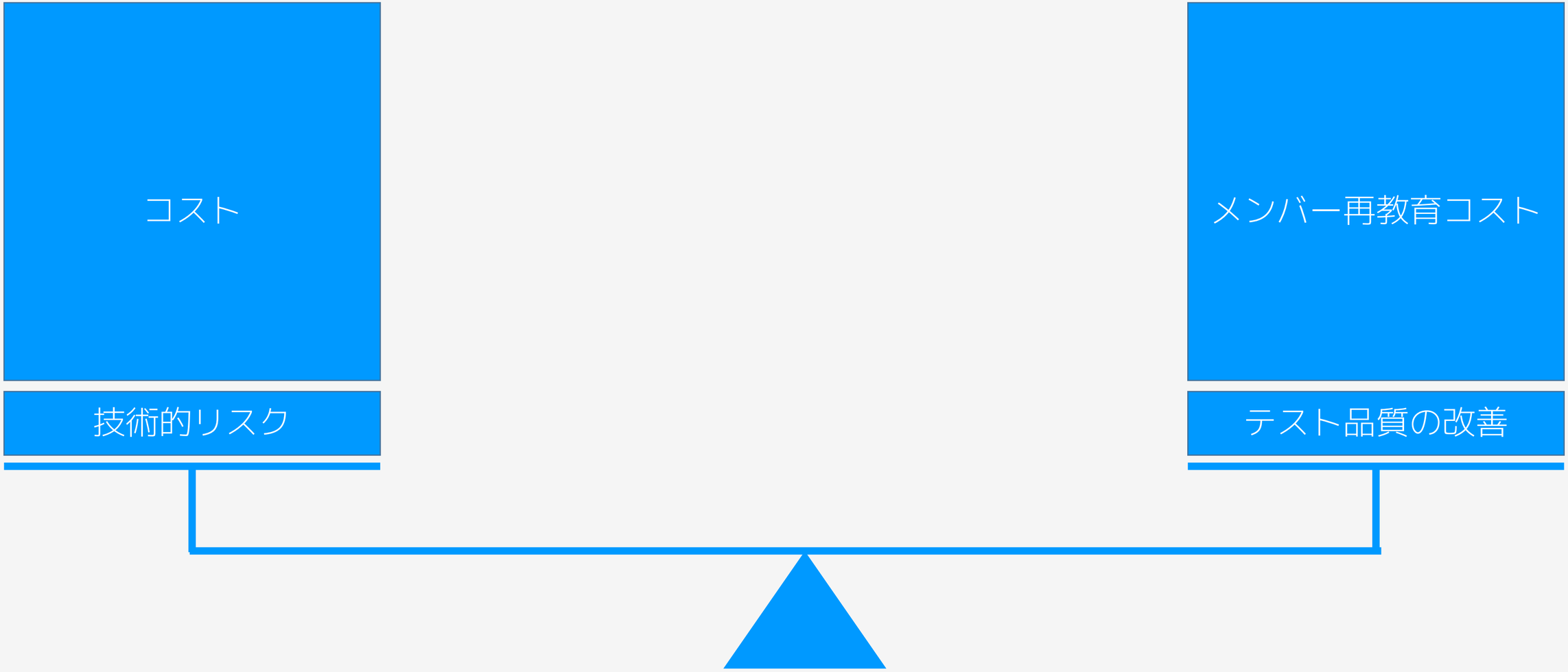


顧客から見た.NET6

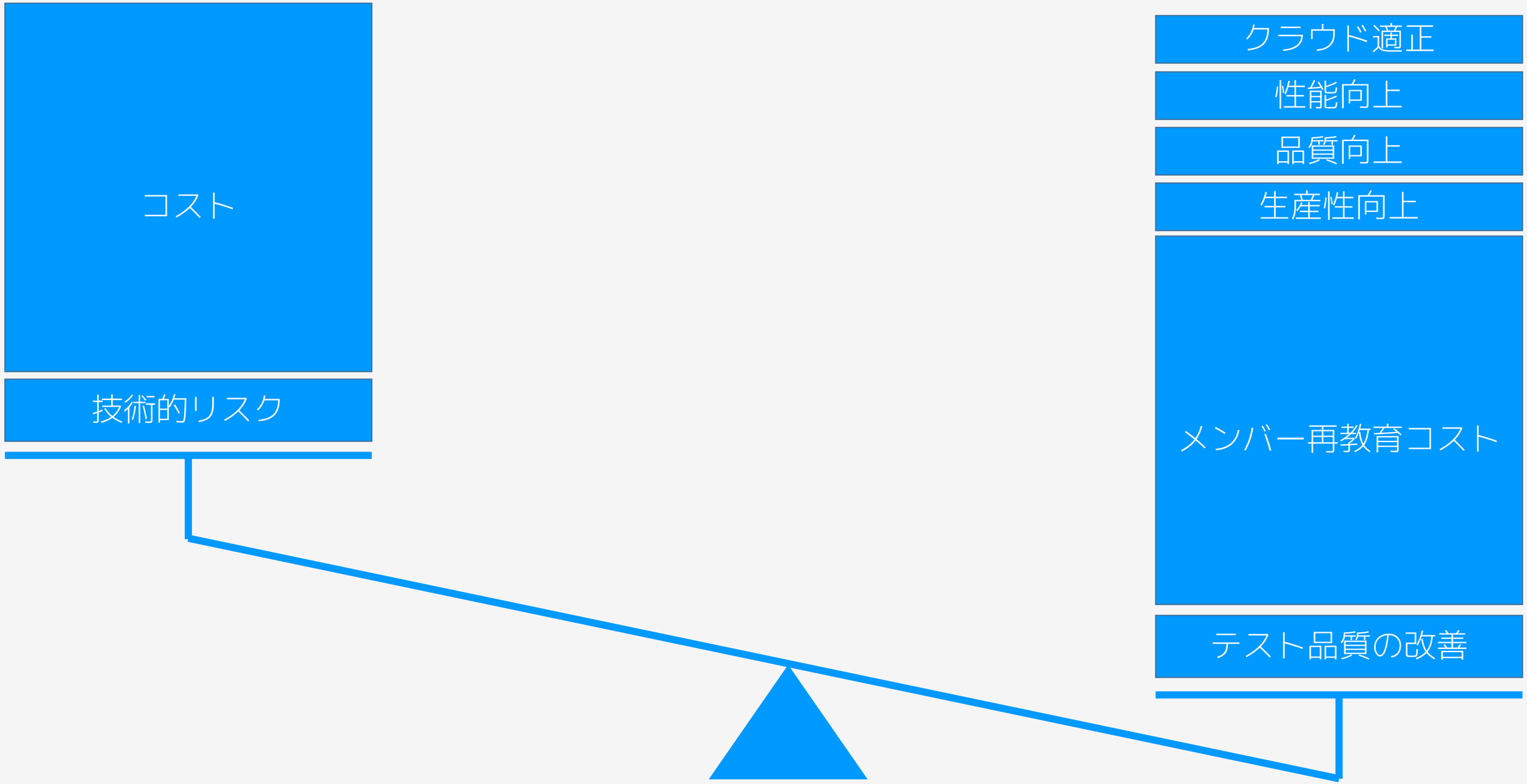


あとはシンプルです

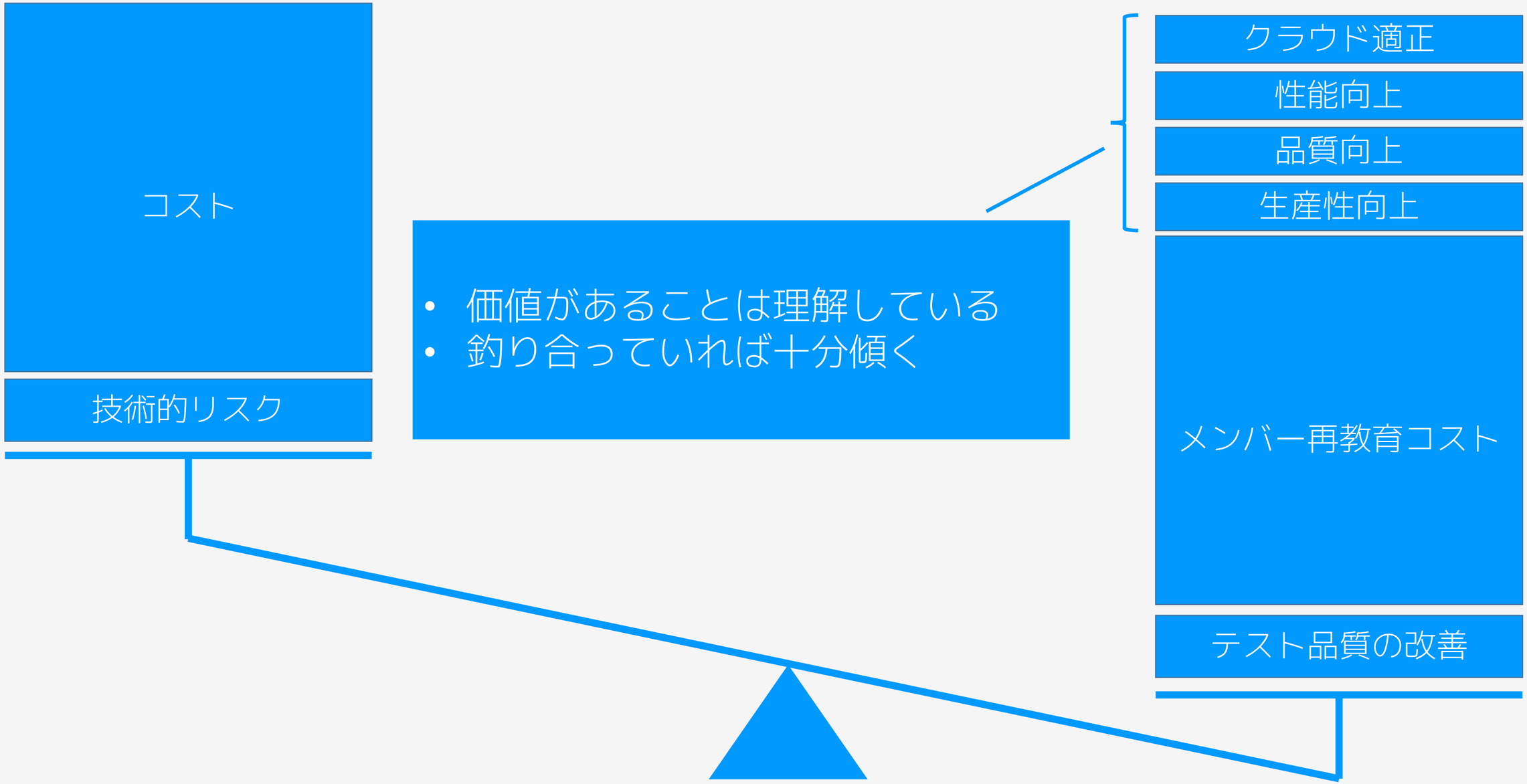
顧客から見た.NET6



顧客から見た.NET6



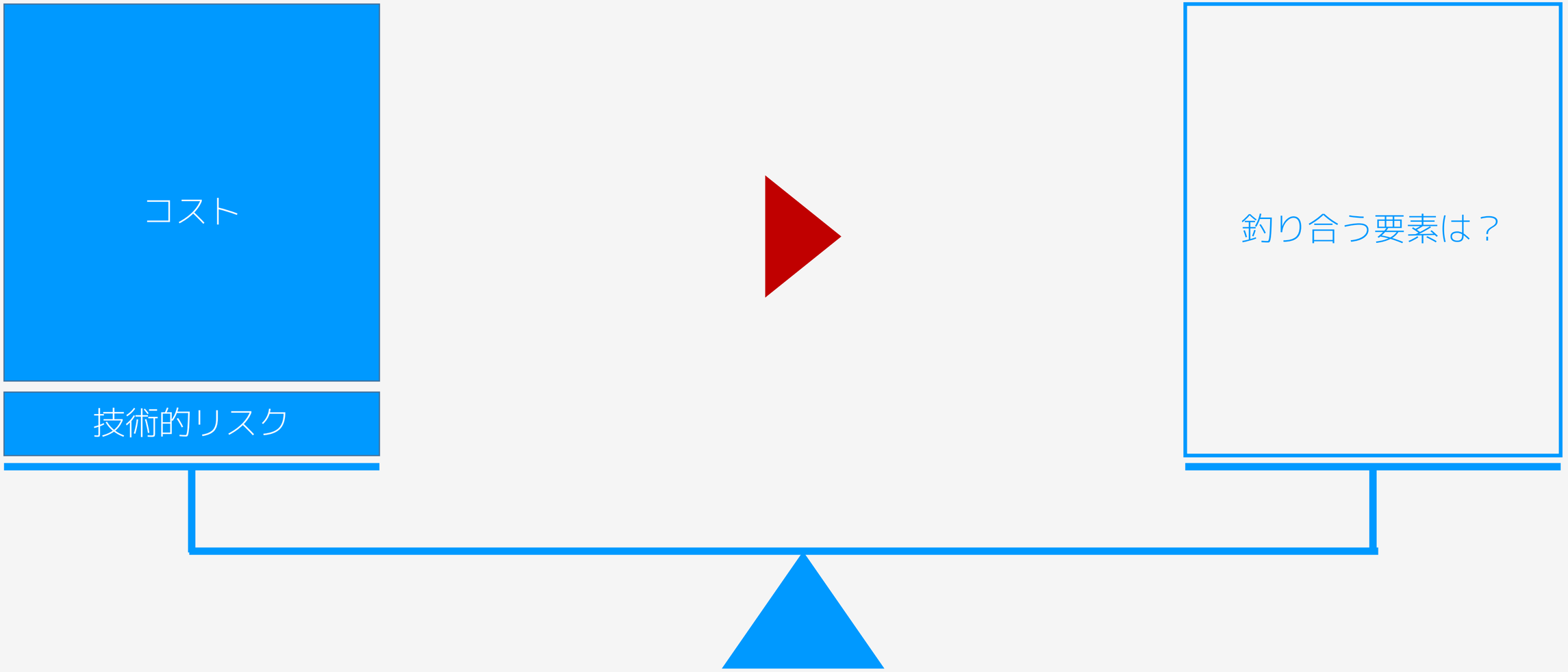
顧客から見た.NET6



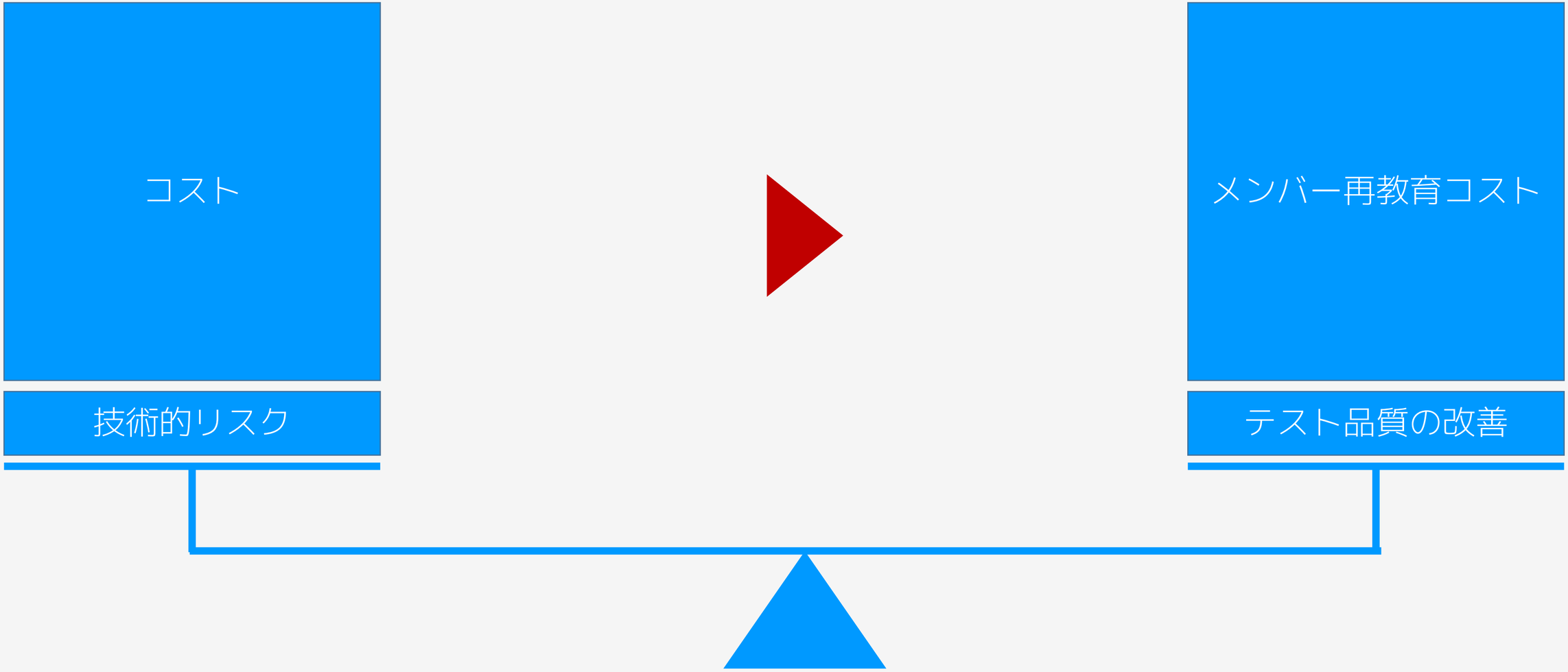
こうして.NET 6移行の理解を得ました

ところで話が上手すぎると思った人？

スタートは実はここ



釣り合う要素を逆算し・・・



Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

当初の顧客要求は少し異なった

当初の顧客要求

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発~~/運用~~の実現
    a. ~~テスト品質の改善~~
    b. ~~業務・システムともに習熟したメンバーの育成~~
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

保守・運用側の課題

1. ・・・
2. 業務とシステムの全体を把握している人材が不足している
3. ・・・
4. ・・・

マージして・・・

逆提案した

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

逆提案した

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機
5. 生産

これらの具体的な施策もすべて提案に含めました

実施

逆提案した

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機
5. 生産

実施

これらの具体的な施策もすべて提案に含めました
あえて伝えなかったのは・・・

私が.NET 6にしたくて仕方なかったことだけ

Thank You!

おしまい！

だと技術的な話が、ほぼ無くてさみしいので・・・

Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

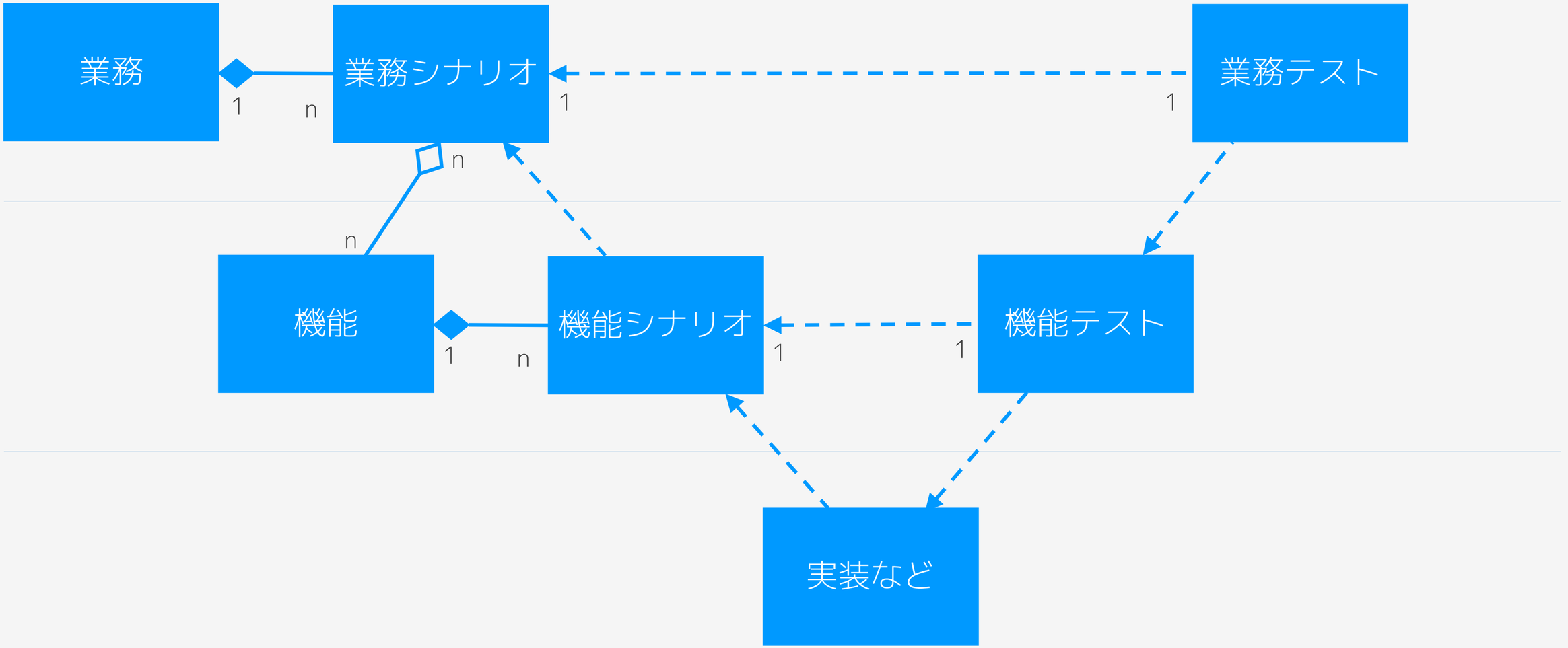
テスト品質の改善について
今回練り直した・・・

.NET 6 migration of a financial company's mission-critical system.

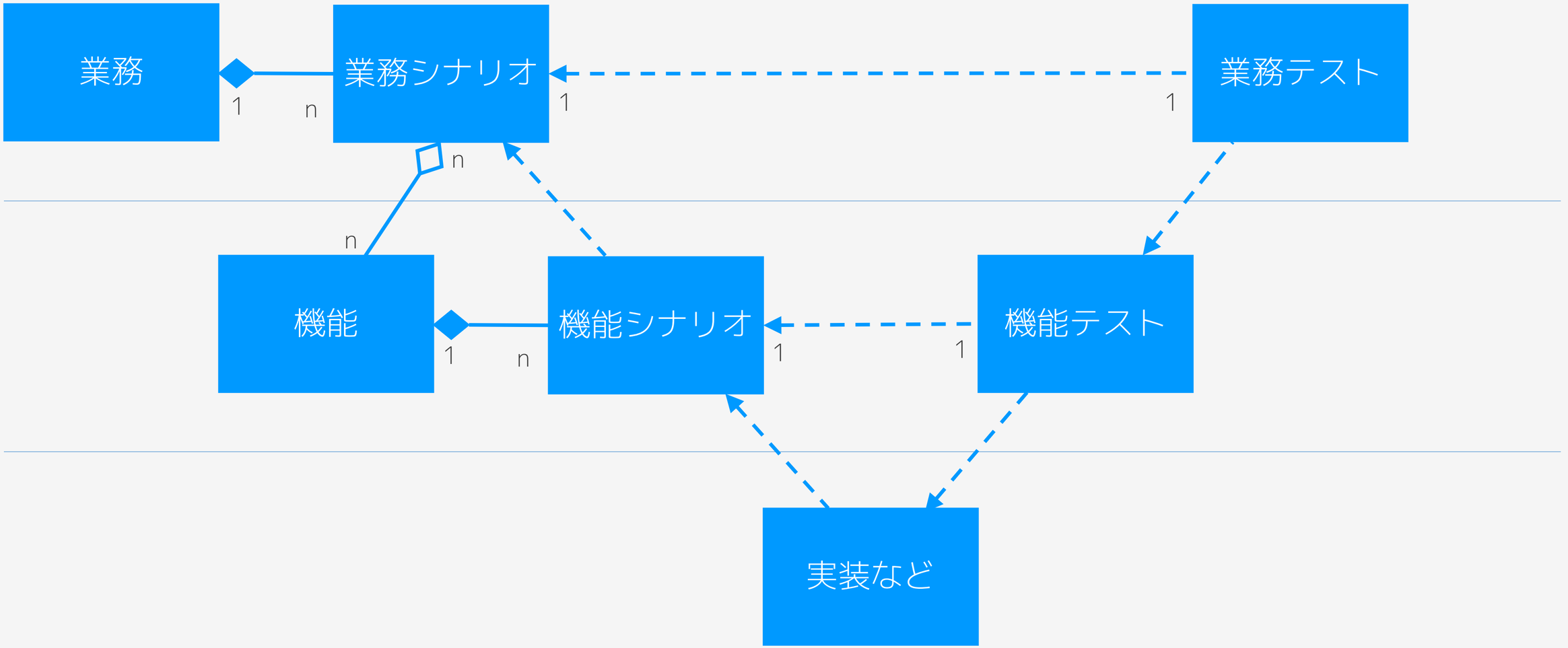
テスト戦略

ドキュメント・テスト体系

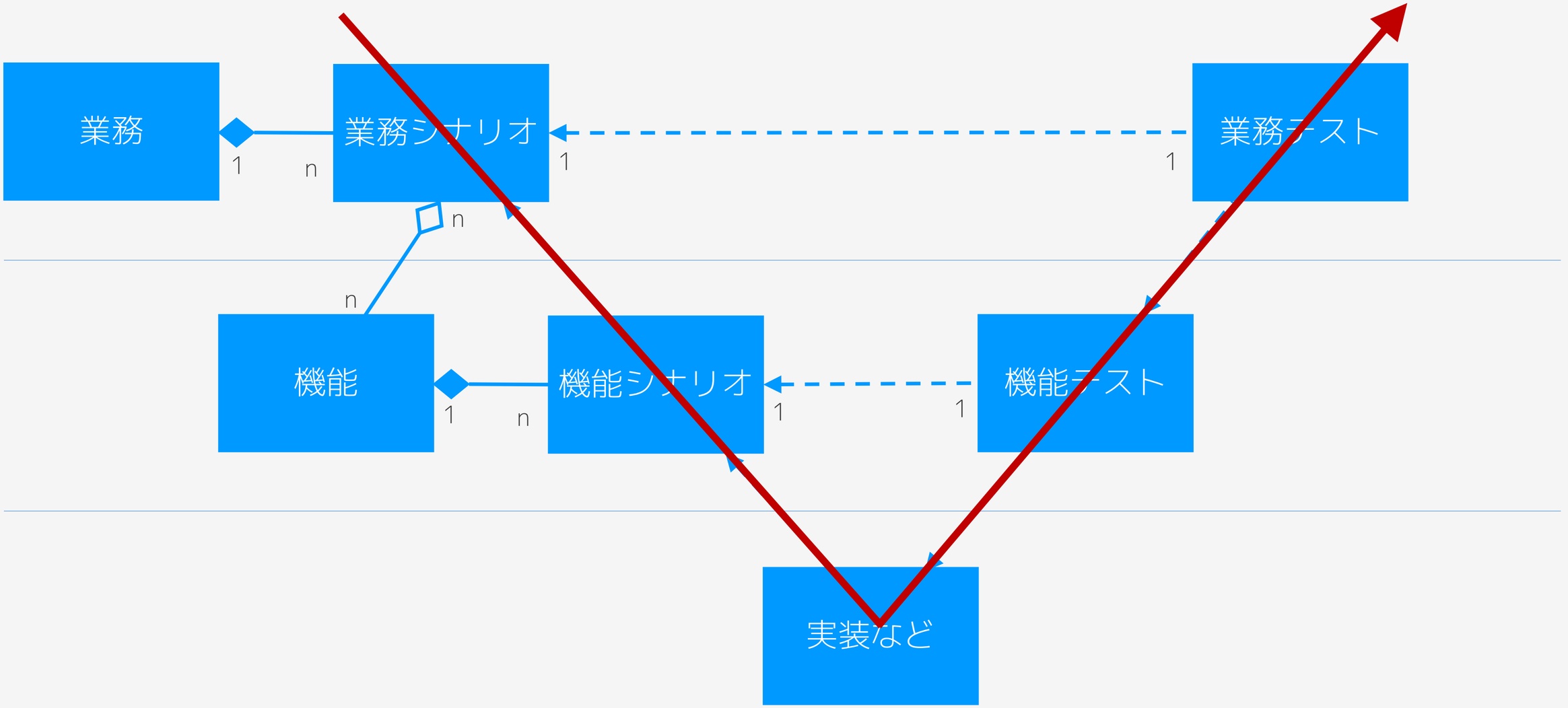
設計とテスト



設計とテスト



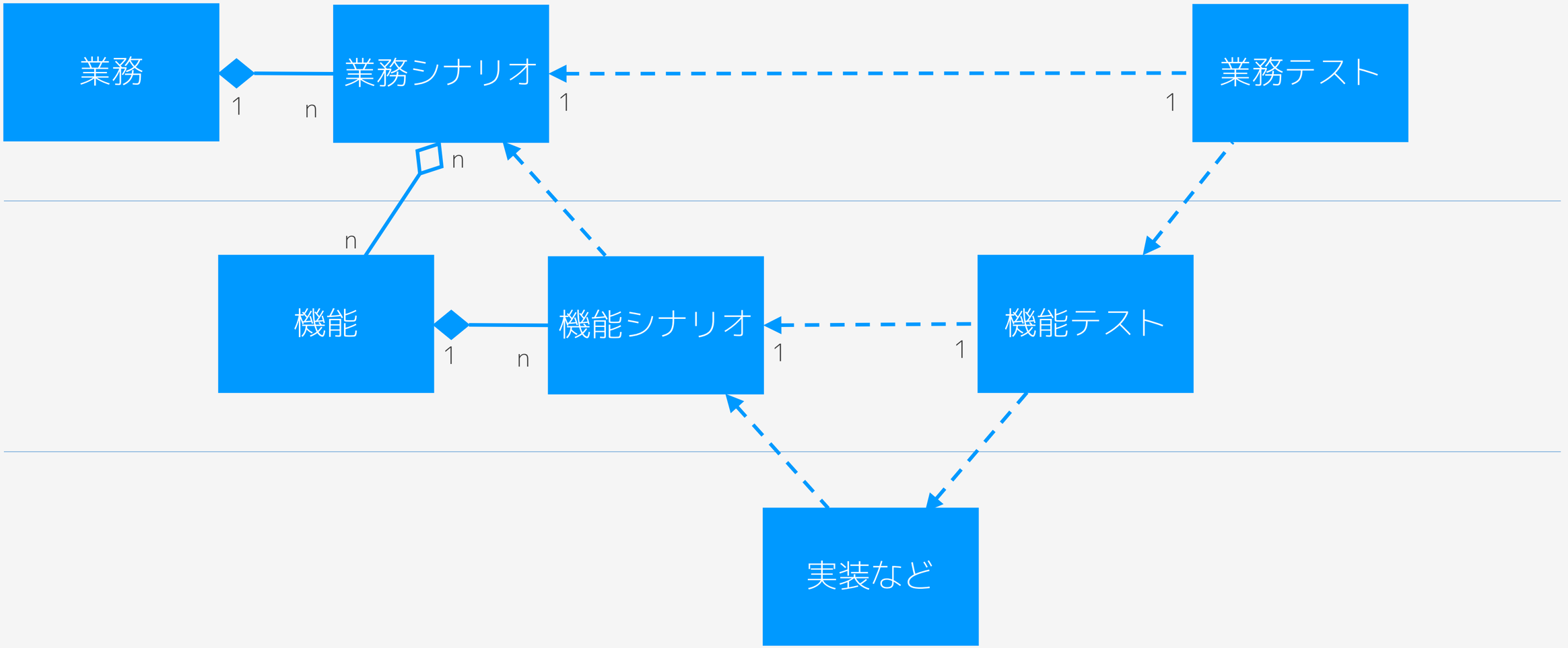
設計とテスト



Project Goal

1. OSのESU期限となる2022年7月までに新環境へ移行する
2. 移行に際し、業務に影響を与えず移行をする
3. 基幹システムに見合った、安定した開発/運用の実現
    a. テスト品質の改善
    b. 業務・システムともに習熟したメンバーの育成
4. 非機能要件を見直し、見合った性能を確保する
5. 生産性向上のため機能追加・改修（性能改善含む）の実施

設計とテスト

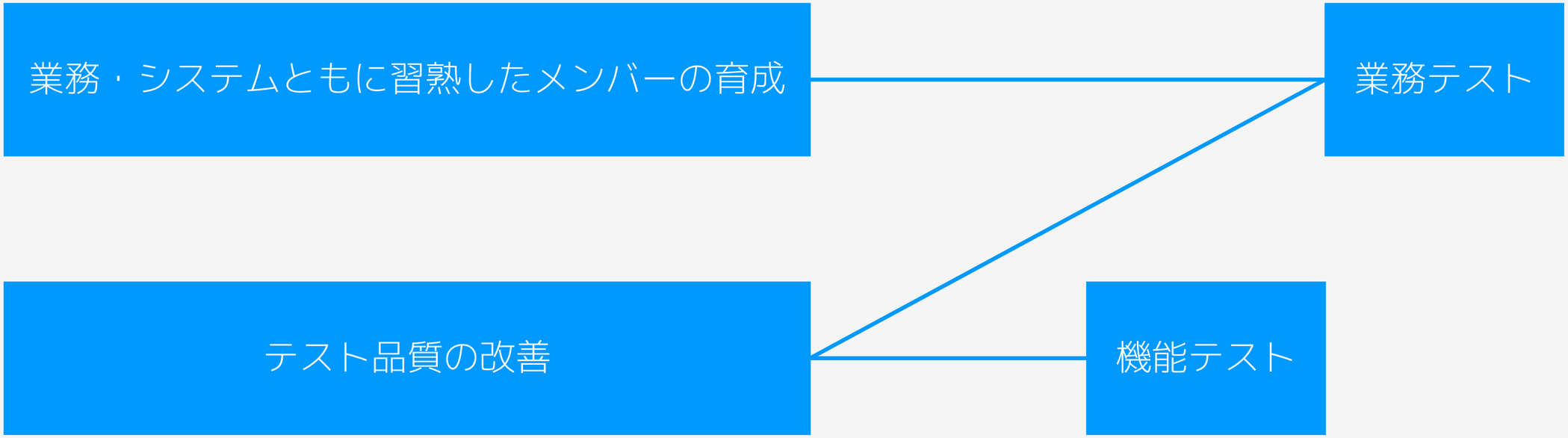


設計とテスト

業務テスト

機能テスト

設計とテスト



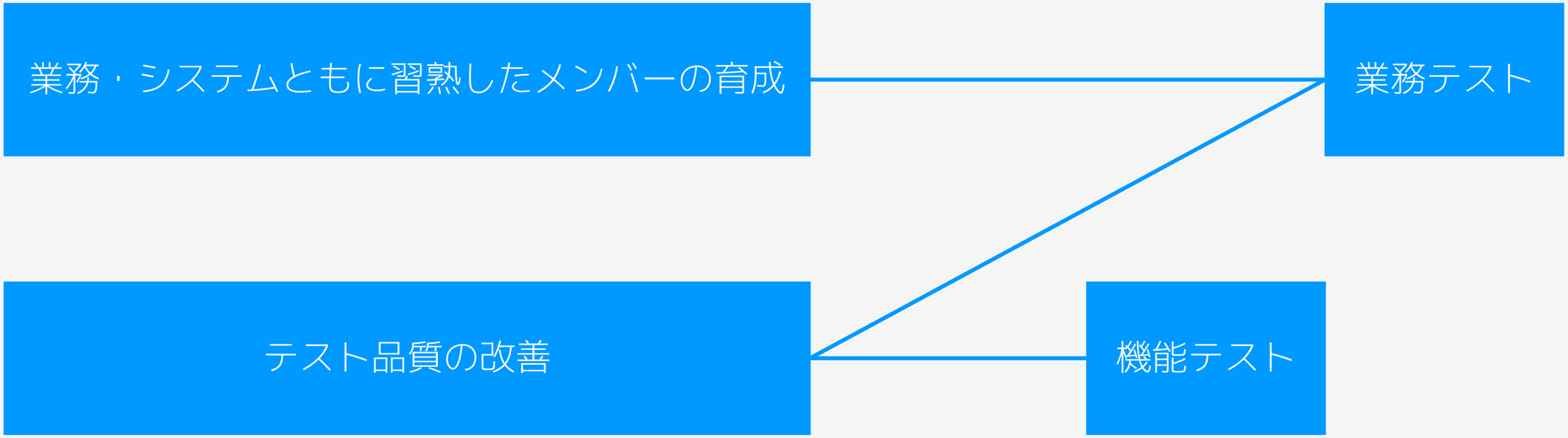
設計とテスト

業務・システムともに習熟したメンバーの育成

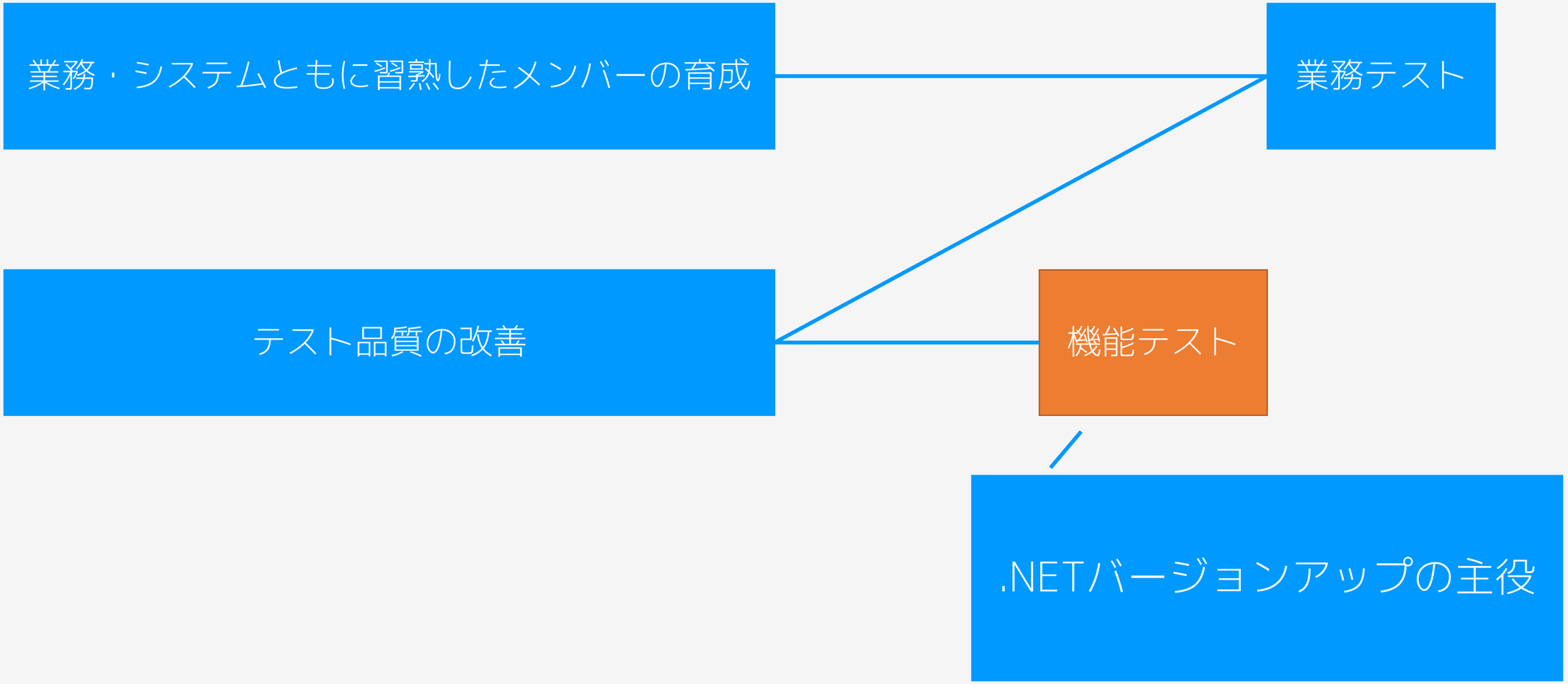
業務テスト

機能テスト

設計とテスト



設計とテスト



機能テストの状態

1. 概ね十分なテストケース
2. 再現性が担保されたテストデータ
3. テストの実施は手動
4. 期待結果との比較は目視

これらの自動化がゴール

機能テスト戦略

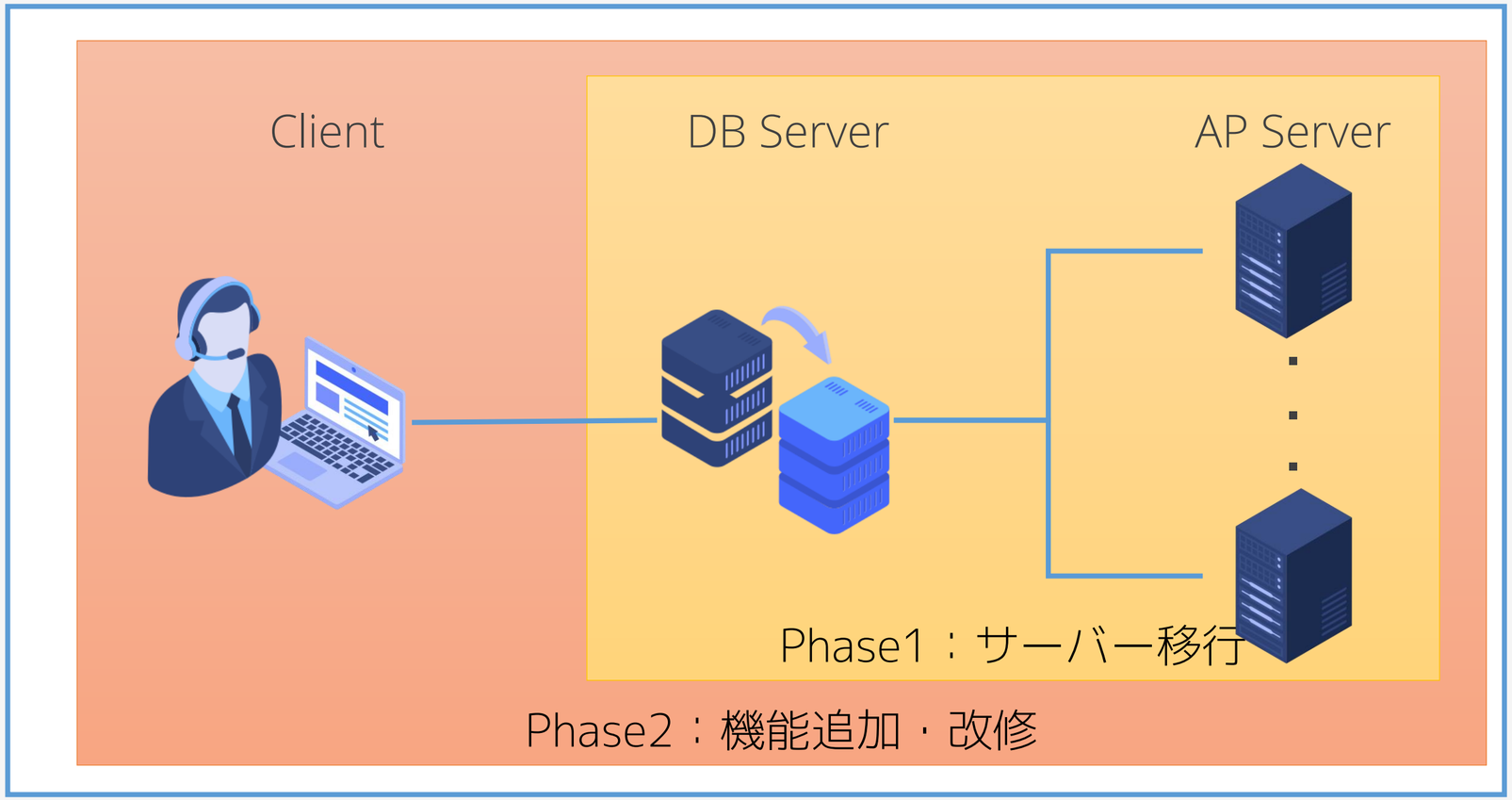
1. .NET Frameworkバージョンで手動テスト実施
2. Databaseの全テーブルをダンプして期待値を作成
3. Microsoft公式のガイドに従って.NETへ移行
   - .NET Framework から .NET への移植の概要
     https://docs.microsoft.com/ja-jp/dotnet/core/porting/
4. テストの自動化
5. 実行結果をダンプしてNo.2の期待値と比較

上記はDatabaseに限定して説明しています

バグも含めて現行動作を担保

10年動いているシステムのバグは、業務的にはとっくに仕様

Project Schedule



フェーズ	2021年												2022年											
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12
サーバー移行																								
機能追加・改修																								

機能テスト戦略

1. .NET Frameworkバージョンで手動テスト実施
2. Databaseの全テーブルをダンプして期待値を作成
3. Microsoft公式のガイドに従って.NETへ移行
   - .NET Framework から .NET への移植の概要
     https://docs.microsoft.com/ja-jp/dotnet/core/porting/
4. .NET 6でテストの自動化
5. 実行結果をダンプしてNo.2の期待値と比較

機能テスト戦略

1. .NET Frameworkバージョンで手動テスト実施
2. Databaseの全テーブルをダンプして期待値を作成
3. Microsoft公式のガイドに従って.NETへ移行
   - .NET Framework から .NET への移植の概要
     https://docs.microsoft.com/ja-jp/dotnet/core/porting/
4. .NET 6でテストの自動化
5. 実行結果をダンプしてNo.2の期待値と比較

めんどくさい。たとえば時間の扱いとか

時間のパターン

No.	期待値のパターン	説明
1	固定値	おもにテスト開始前に登録され、意図せず変更されないことを評価する。
2		
3		

時間のパターン

No.	期待値のパターン	説明
1	固定値	おもにテスト対象外のデータ。意図せず変更されないことを評価する。
2	可変 – DB初期化時刻以前	テスト開始前で、5分以内の時刻が必要といったケース
3		

時間のパターン

No.	期待値のパターン	説明
1	固定値	おもにテスト対象外のデータ。意図せず変更されないことを評価する。
2	可変 – DB初期化時刻以前	たとえば、前回の実行から、所定の時間を経過していなかった場合、処理をパスするような振る舞い
3	可変 - DB初期化時刻以後	UpdateAtのような列

テストが難しくなる要素

1. 時間
2. 環境（例えばログのホスト名）
3. 冪等性の担保されていないサービス・ライブラリへの依存

3は、そっちを直せよって気分になりますが、例えば・・・

個別に対応する内容じゃない

「PDFのメタデータには作成日時や更新日時が含まれている
　同じ条件でPDFを生成しても、同じバイナリーにならない」

ツール＆ライブラリー作りました

DbAssertions

提供機能

1. DatabaseからCSVでエクスポートして期待値を作成するツール
2. テスト実行後にDatabaseとCSVを比較するライブラリ

https://github.com/nuitsjp/DbAssertions

リポジトリをのぞいてみると・・・

No.	ディレクトリ	説明
1	DbAssertions	
2	DbAssertions.SqlServer	
3	DbAssertions.SqlServer.App	
4	DbAssertions.Test	
5	Sample	

リポジトリをのぞいてみると・・・

No.	ディレクトリ	説明
1	DbAssertions	コアライブラリ。一応DB非依存で、No.2・3から依存されている。
2	DbAssertions.SqlServer	
3	DbAssertions.SqlServer.App	
4	DbAssertions.Test	
5	Sample	

リポジトリをのぞいてみると・・・

No.	ディレクトリ	説明
1	DbAssertions	コアライブラリ。一応DB非依存で、No.2・3から依存されている。
2	DbAssertions.SqlServer	DbAssertionsのSQL Server向けの実装 • DBの値をエクスポートして期待値ファイルの作成 • ユニットテストへ期待値との比較するAssertionの提供 Appおよびテストコードから利用される
3	DbAssertions.SqlServer.App	
4	DbAssertions.Test	
5	Sample	

リポジトリをのぞいてみると・・・

No.	ディレクトリ	説明
1	DbAssertions	コアライブラリ。一応DB非依存で、No.2・3から依存されている。
2	DbAssertions.SqlServer	DbAssertionsのSQL Server向けの実装 • DBの値をエクスポートして期待値ファイルの作成 • ユニットテストへ期待値との比較処理の提供 Appおよびテストコードから利用される
3	DbAssertions.SqlServer.App	期待値ファイルを作成するため、No.2をコンソールアプリケーションとして提供する
4	DbAssertions.Test	
5	Sample	

リポジトリをのぞいてみると・・・

No.	ディレクトリ	説明
1	DbAssertions	コアライブラリ。一応DB非依存で、No.2・3から依存されている。
2	DbAssertions.SqlServer	DbAssertionsのSQL Server向けの実装 • DBの値をエクスポートして期待値ファイルの作成 • ユニットテストへ期待値との比較処理の提供 Appおよびテストコードから利用される
3	DbAssertions.SqlServer.App	SQL Server用の期待値ファイルを作成するためのアプリケーション
4	DbAssertions.Test	No.1・2のテストコード
5	Sample	

リポジトリをのぞいてみると・・・

No.	ディレクトリ	説明
1	DbAssertions	コアライブラリ。一応DB非依存で、No.2・3から依存されている。
2	DbAssertions.SqlServer	DbAssertionsのSQL Server向けの実装 • DBの値をエクスポートして期待値ファイルの作成 • ユニットテストへ期待値との比較処理の提供 Appおよびテストコードから利用される
3	DbAssertions.SqlServer.App	SQL Server用の期待値ファイルを作成するためのアプリケーション
4	DbAssertions.Test	No.1・2のテストコード
5	Sample	デモで利用するサンプルリソース置き場

もうコード見てたって人、手を挙げて！

お題

1. AdventureWorks　- いつもいつもお世話になります
2. PersonスキーマのPersonテーブルを利用
3. 初期化時にID＝1の行を更新
    1. ModifiedDateを現在時刻に
4. テストでは、ID=2の行を更新
    1. TITLEを「Mr.」に
    2. ModifiedDateを現在時刻に

先ほどのリポジトリのSampleプロジェクトを参照

- https://github.com/nuitsjp/DbAssertions

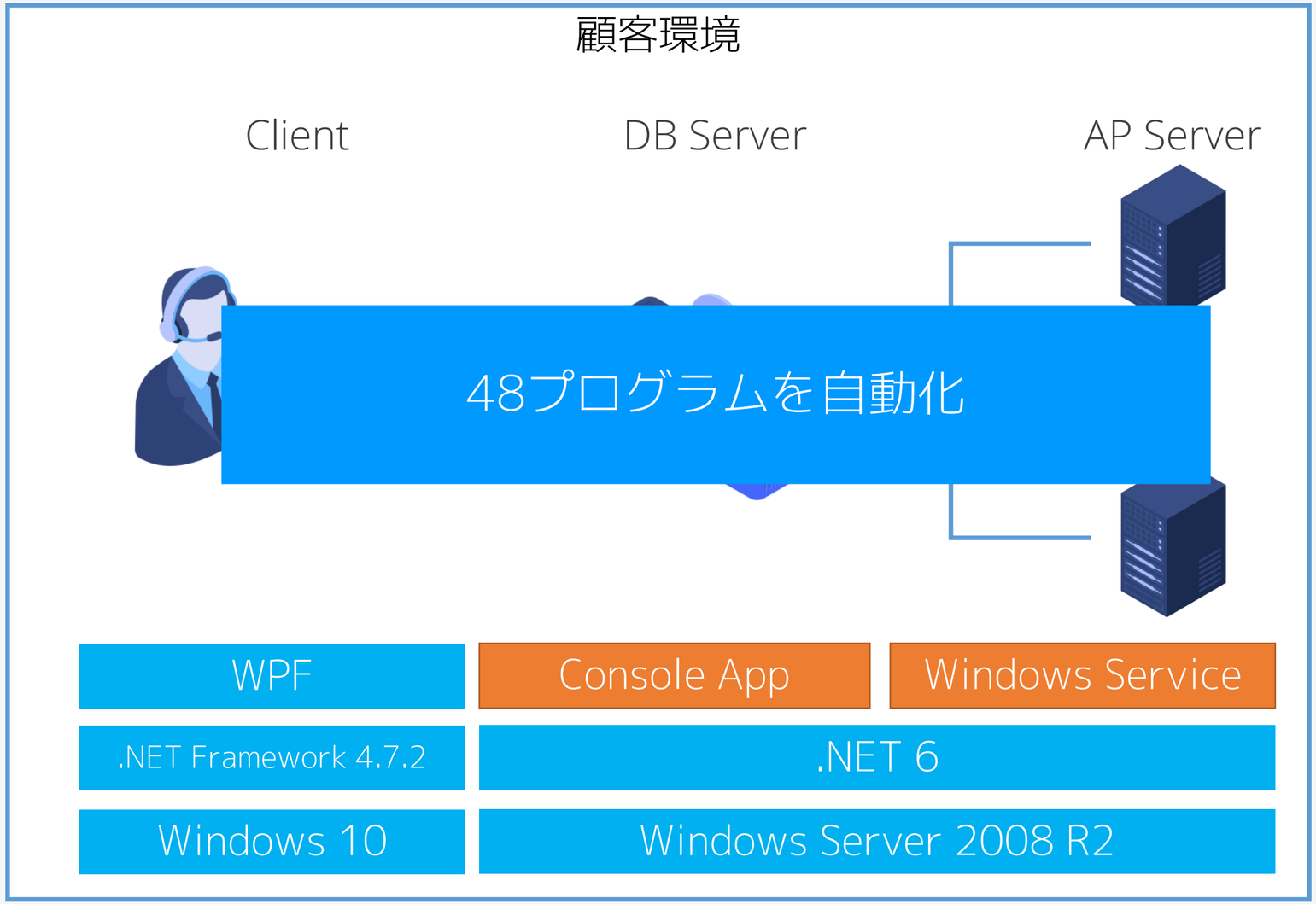
テスト自動化の手順

1. AdventureWorksをきれいな状態で起動する
2. テスト前の状態にデータを更新する（Initialize.sql）
3. 自動化対象を実行する（UpdateTitle.sql）
4. ツール(DbAssertions.SqlServer.App.exe)でDBをダンプする
5. テストを自動化する（Sample.csproj）

2回実行する

実際に見てみよう！

.NETどうする？



使いたい人は、ドキュメント書けってIssue建ててplz.

.NET 6 migration of a financial company's mission-critical system.

まとめ

まとめ

プロジェクトスポンサーから見た場合、.NET Frameworkが選びやすいのは事実だと思います。

今回は私が顧客を説得できたシナリオを紹介しましたが、背景に強く依存した内容です。

これをそのまま流用するというわけにはいかないでしょう。

技術者として、よりモダンな環境に身を置けるようにするため

皆さん自身のwin-winのシナリオを見つけよう！

Thank You!

おしまい！