Empowered by Innovation NEC

# 中堅企業向けシステム開発における コンパクトな性能検証の進め方

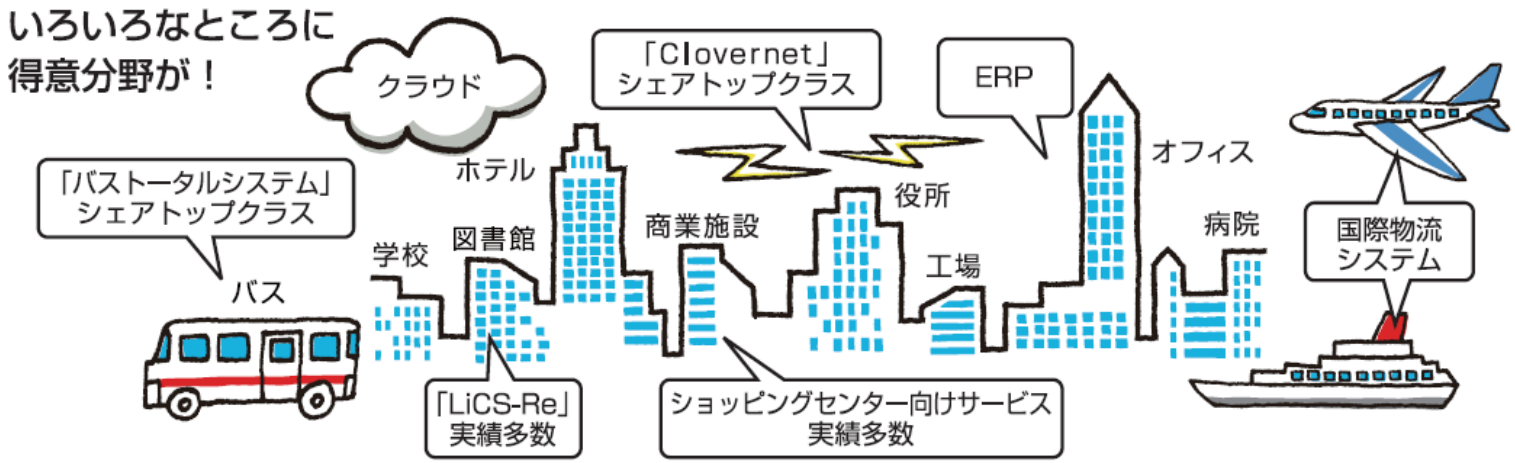
## 「このシステムは性能要件を98%達成しています」と言える検証プロセス

NECネクサソリューションズ
技術開発事業部
小池輝明

# NECネクサソリューションズ株式会社

## 豊富な事例・確かな技術をベースにした商品・サービス

豊富な事例や技術、ノウハウを集約した当社の商品・サービスを、お客様の課題にあわせ、最適な組み合わせで、ソリューションとして提供いたします。企業経営の基盤となるＥＲＰ。持たずに使う新しい時代の経営を支えるクラウドサービス。インターネットを利用して、安心の企業ネットワークを実現するマネージドＶＰＮサービス「Ｃｌｏｖｅｒｎｅｔ」などを代表に、私たちの商品・サービスは、様々な業種・ビジネスシーンでお客様から高い評価をいただいています。



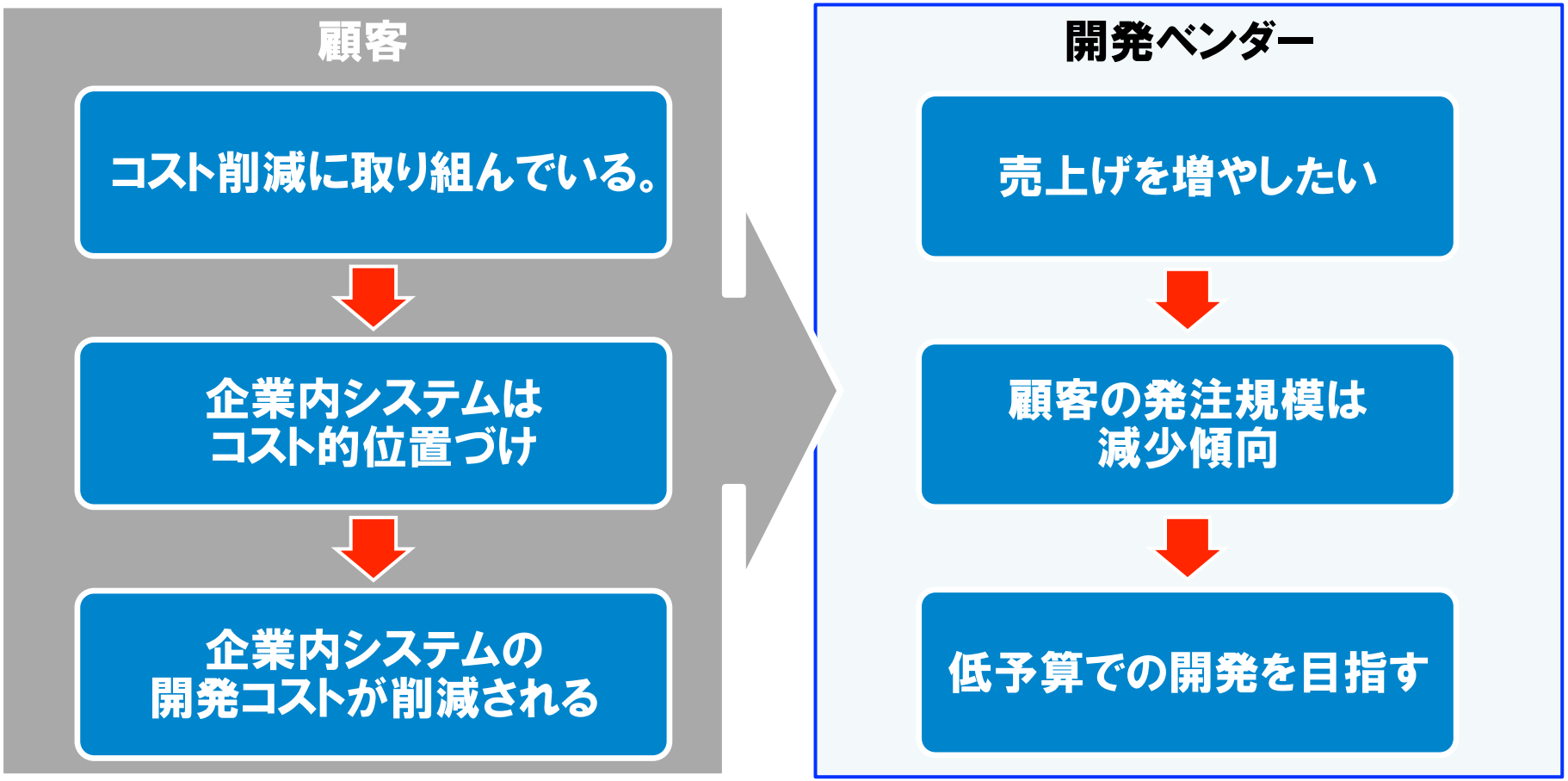
# SI開発のシステムの種類と今回のターゲット

| システムの種類 | 目的 | システム例 | 関連するシステム |
|---|---|---|---|
| 社会インフラ | 社会生活の基盤維持 | 交通、金融、通信など | 複数企業のシステムが密接に関連 |
| 顧客サービス | 売上拡大 | BtoC ショッピングサイト | 顧客向けシステムと企業内システム |
| 企業内システム | 業務効率化（コスト削減） | 基幹システム 業務システム、etc | 企業内システムと取引先企業 |

**今回性能検証のターゲットとしているシステム領域**
**顧客は中堅企業**

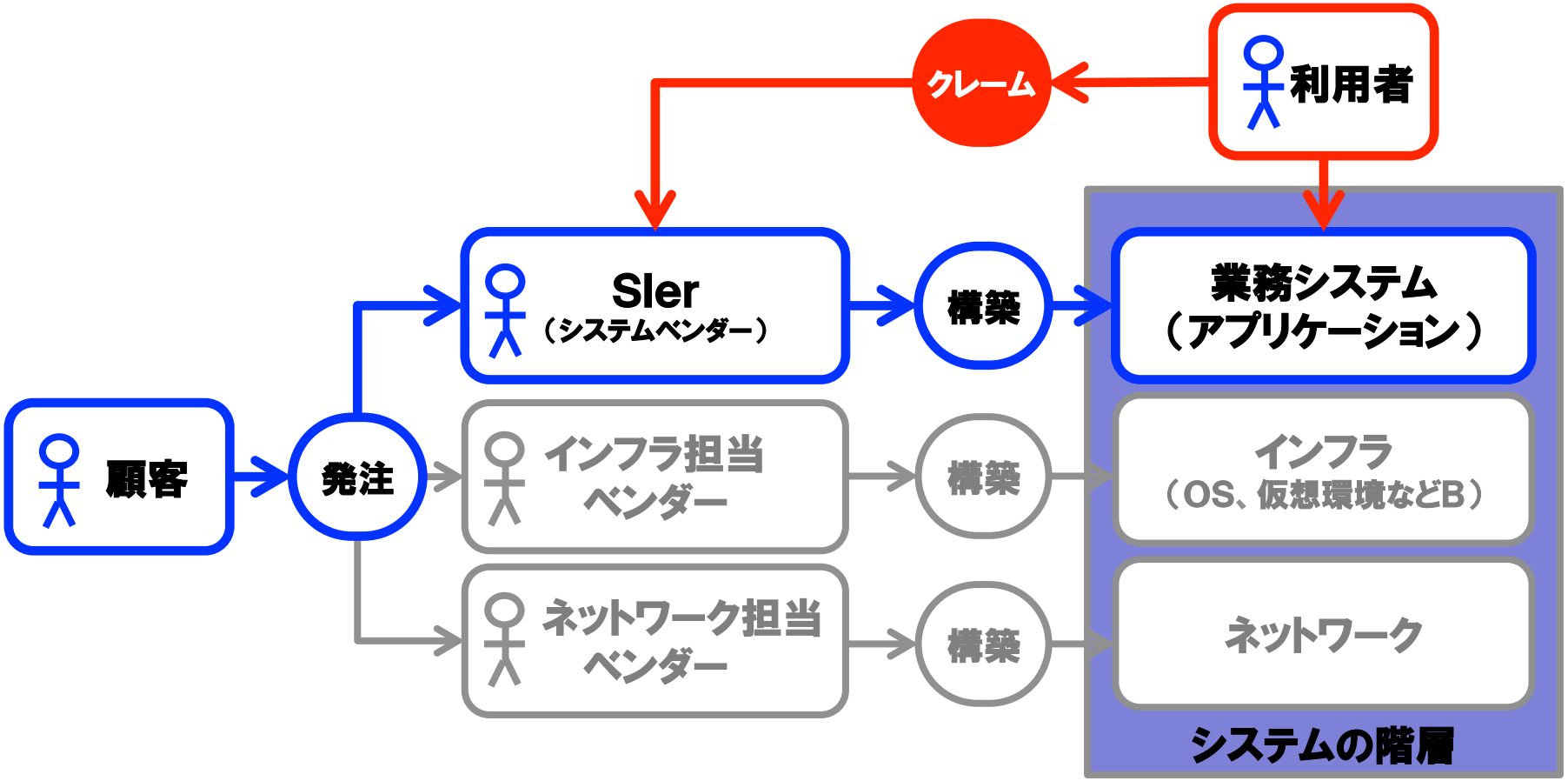
# 企業内システムにおける顧客とベンダーの課題(１)

**顧客は社内業務のコスト削減に取り組む一方で予算が縮小。**
**開発ベンダーは受注金額の縮小により利益確保が必要。**



# 企業内システムにおける顧客とベンダーの課題(２)

**システム構築における顧客の発注はマルチベンダー化が進む。**
**しかし、利用者のクレームはSIerに向けられる。**



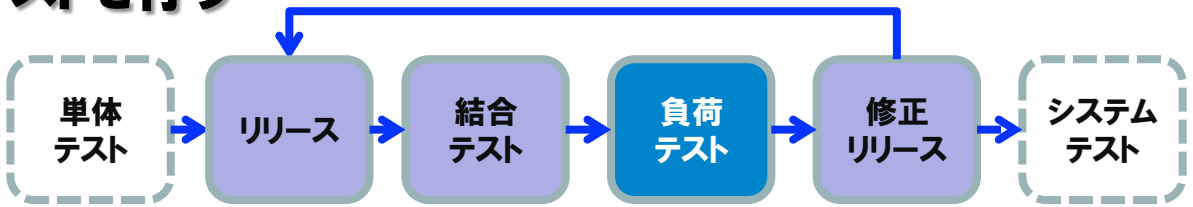
# 低予算の中で性能検証を実施するジレンマ

**低予算での開発であっても性能検証は実施したい。**
**しかし、開発スケジュールへの影響は最小限にしたい。**

## 結合テストフェーズで負荷テストを行う

○：環境コストが1つで済む

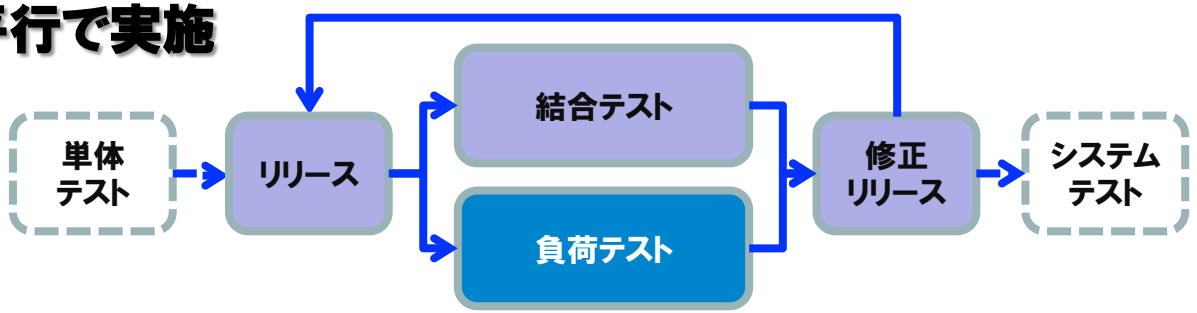
×：負荷テスト中は環境を占有
（テスト遅延リスク）




## 結合テストと負荷テストを平行で実施

○：テスト期間の短縮が可能

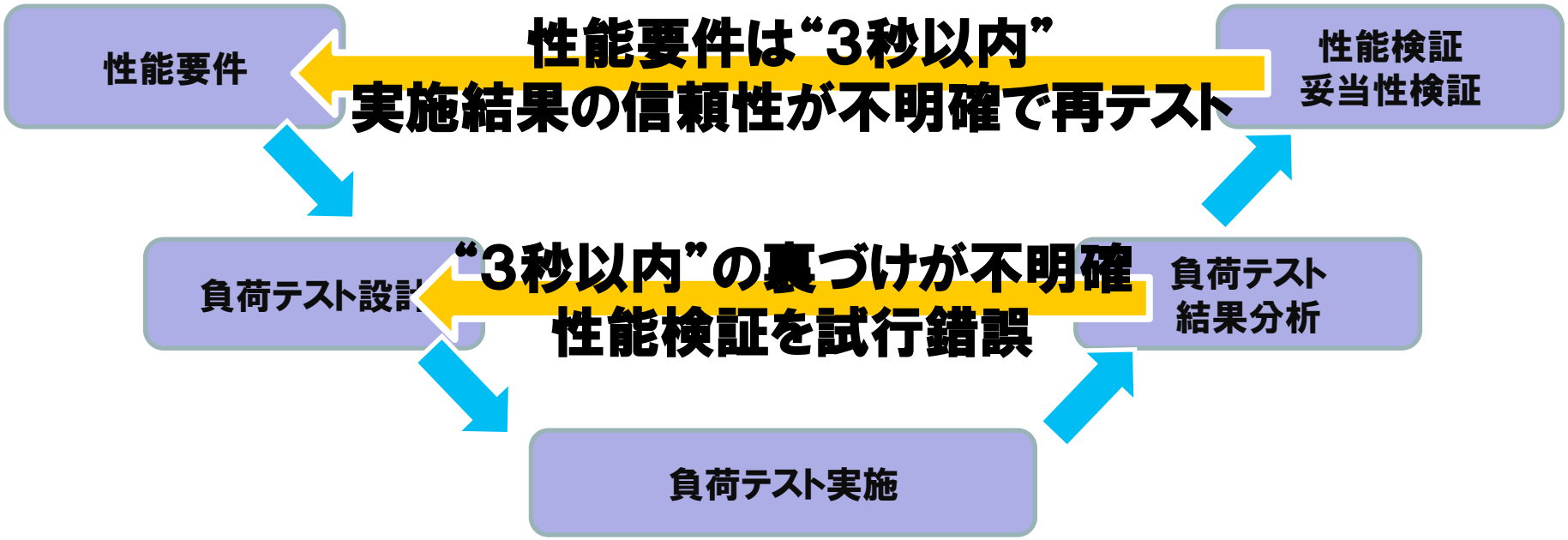
×：各々の環境が必要
（コスト増）




**結合テストを妨げないために、性能検証を効率的に行いたい。**

# 性能検証の効率化の阻害要因

性能要件や検証作業があいまいで、
手戻り作業の発生リスクが大きい。




全体的に場当たり的な検証作業

場当たり的な検証作業を見直し、性能検証のゴールを明確にする。

# 課題のまとめと対策

**開発予算が限られている中で効率的に性能検証を行う。**
**ただし、短期期間で行い開発を妨げない。**

## 課題

- “アドホック”な性能検証
- 性能検証作業の達成度があいまい
- 低予算での開発

**性能検証プロセスをコンパクト化し**
**効率的に性能検証を行う**

## 施策

- 【コンパクト化施策①】性能要件の見える化
- 【コンパクト化施策②】検証結果の見える化
- 【コンパクト化施策③】効率的な性能検証

## コンパクト化施策①　性能要件の見える化による性能検証の効率化

# 負荷テストシナリオのパターン分けによる性能要件の見える化

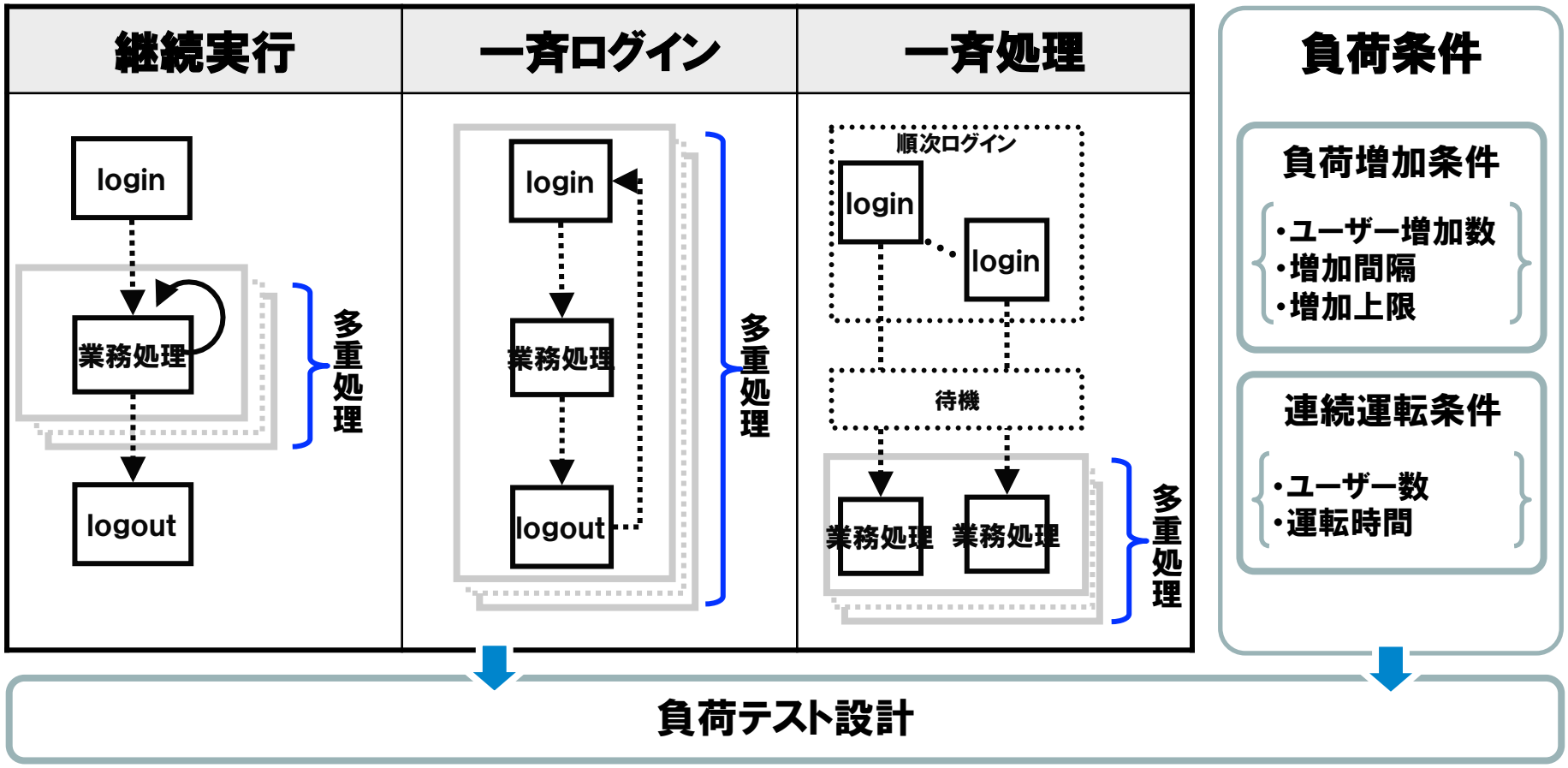
**業務面において性能検証ニーズの多い3パターンを策定。**
**性能検証結果イメージの具体化を狙う。**

| シナリオタイプ | 継続実行 | 一斉ログイン | 一斉処理 |
|---|---|---|---|
| 概要 | 特定業務を<br>繰り返し実行 | 複数ユーザーが<br>一斉ログインし<br>業務を実行 | 高負荷な業務処理を<br>一斉に実行 |
| 業務の例 | 伝票業務<br>店舗業務<br>受付業務 | 出退勤処理<br>商品検索<br>ワークフロー | 帳票出力<br>店舗業務<br>締め処理 |
| 性能要件<br>ポイント | 継続利用時の<br>リスクがないか | 短時間に<br>大量処理を行えるか | 全ての処理が<br>実行されるか |

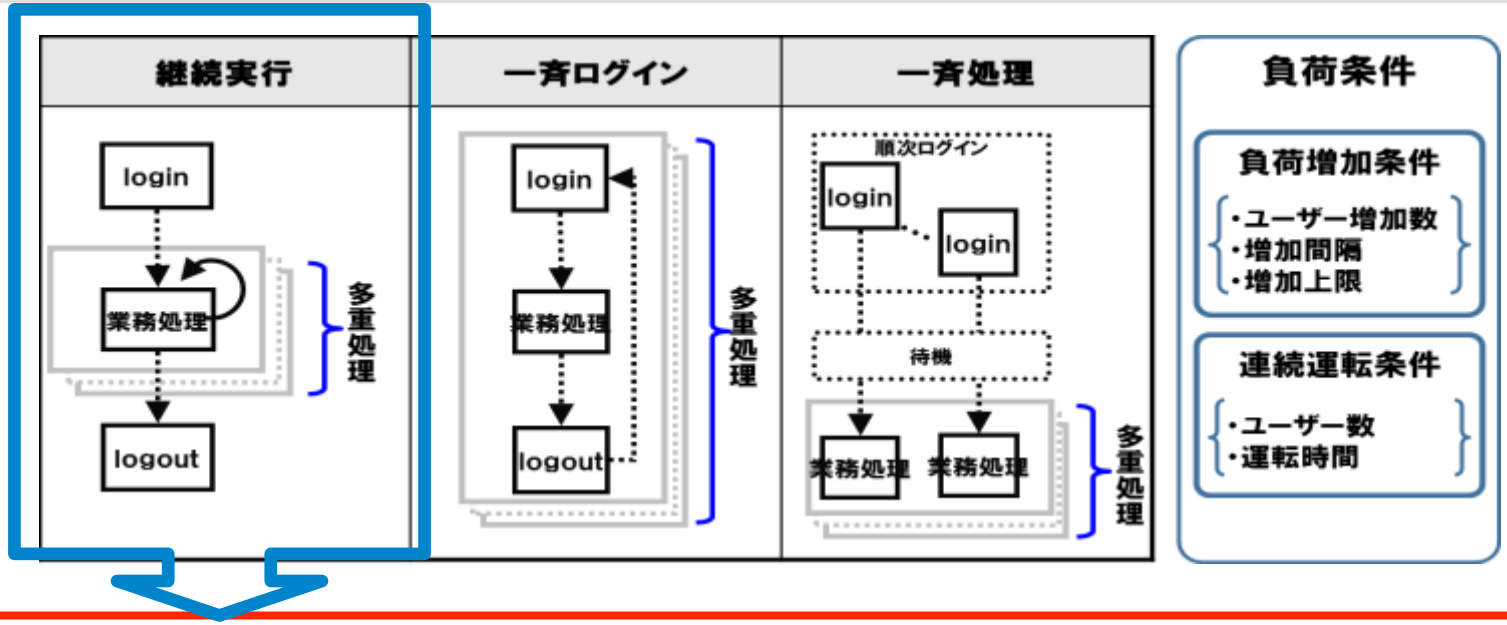
## コンパクト化施策①　性能要件の見える化による性能検証の効率化

# 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ

**3つのシナリオと負荷条件を組み合わせ、システム負荷状態による性能要件の達成度合いを確認する**



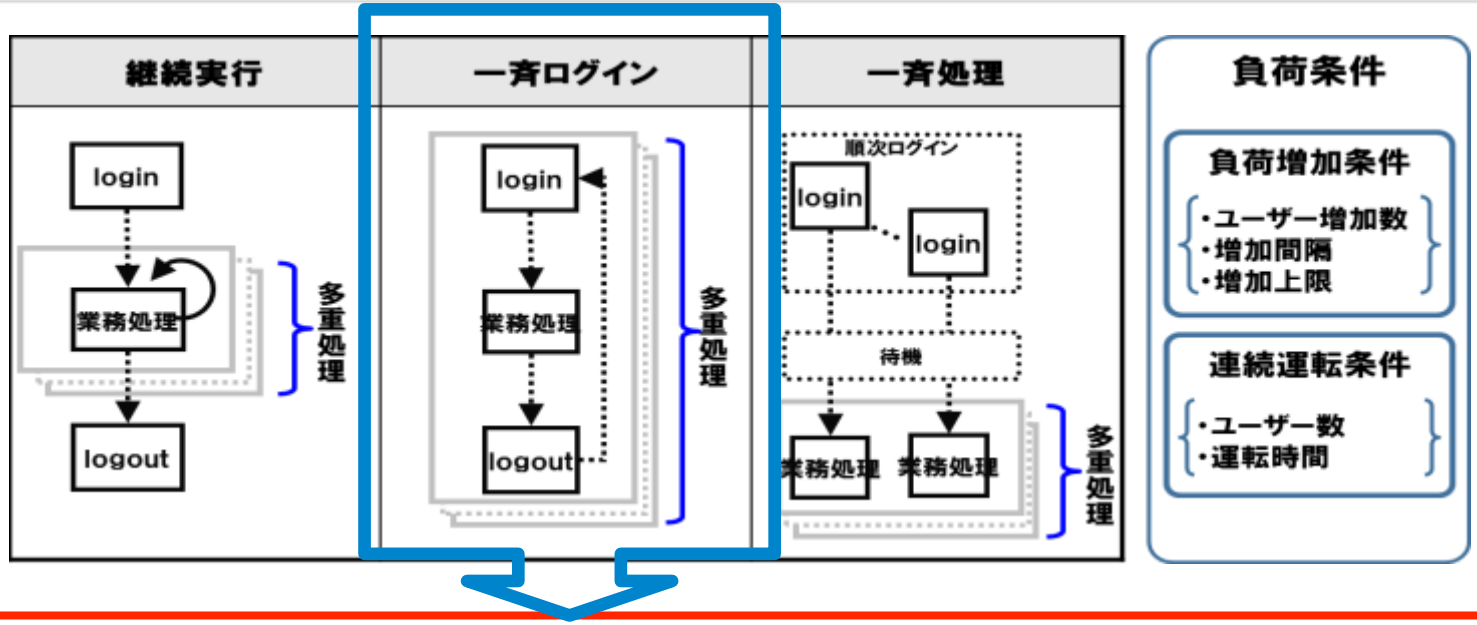
# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ




**シナリオタイプ「継続実行」**
**ログインしたユーザーは、一定期間、業務処理を繰り返す。**

**【想定業務】**
- **販売管理：受注伝票の入力**
- **店舗業務：レジ業務**
- **コールセンター：受付業務**

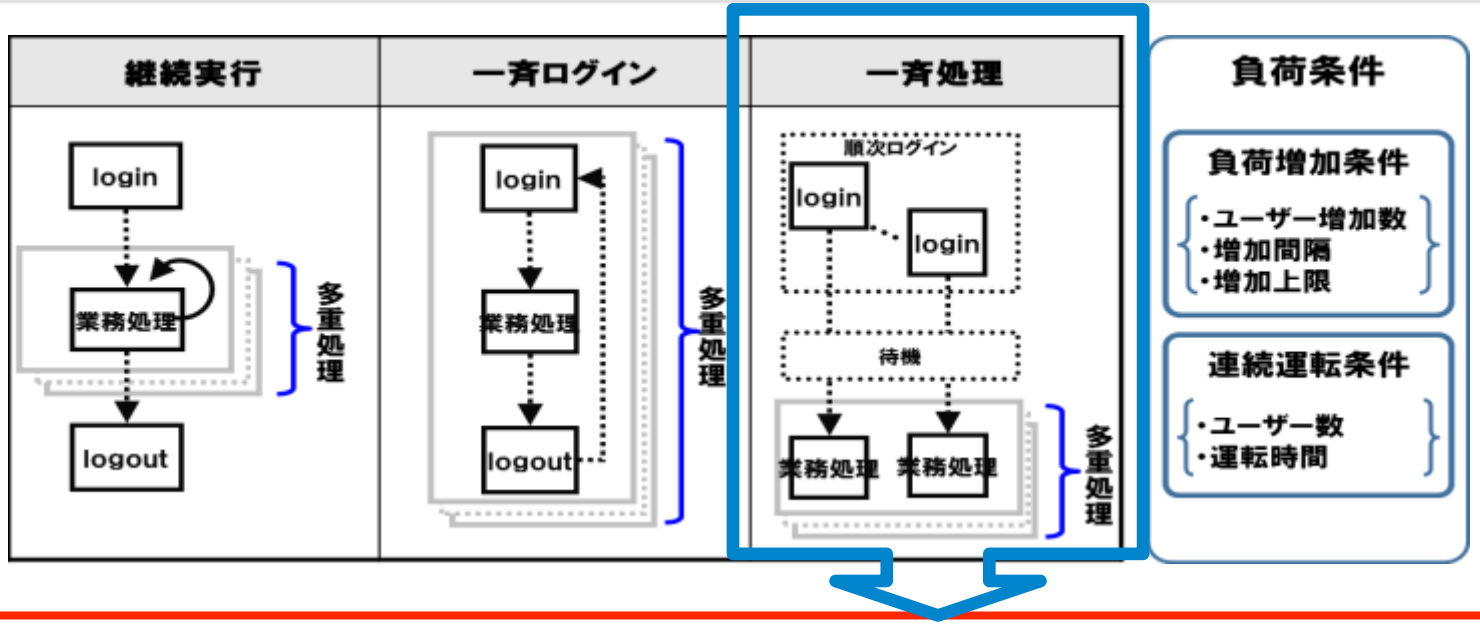
# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ




**シナリオタイプ「一斉ログイン」**
**ログイン後、ユーザーは業務処理を繰り返し、ログアウトする。**

**【想定業務】**
- **勤務管理：出退勤入力**
- **販売業務：商品検索**
- **ワークフロー：承認処理**

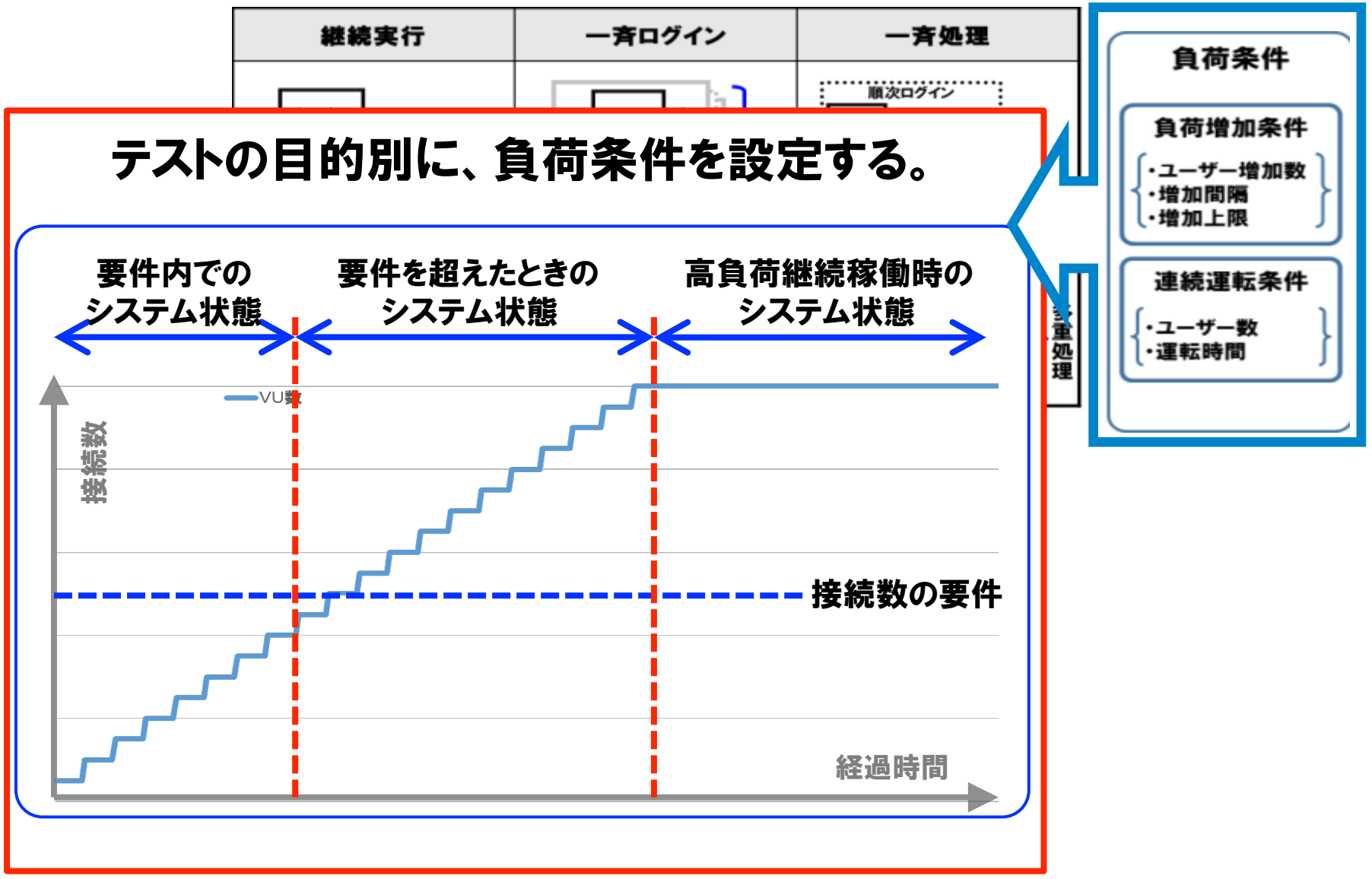
# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ




**シナリオタイプ「一斉処理」**
**特定業務の担当者が、比較的高負荷の業務を一斉に実施。**

**【想定業務】**
- **販売管理：帳票出力**
- **店舗業務：支店への通達**
- **業務共通：各種締め処理**

# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ

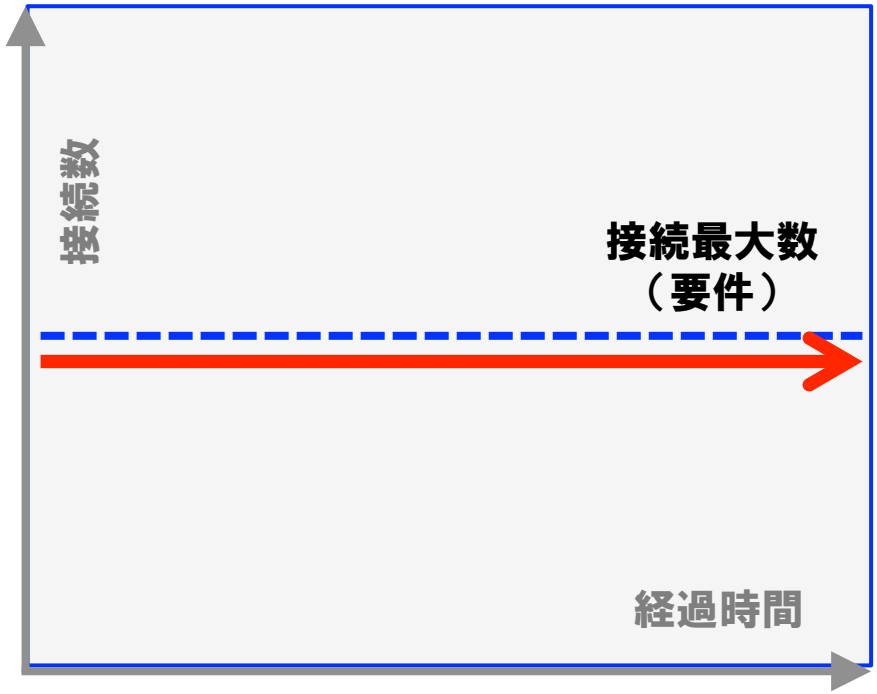


# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ

| 継続実行 | 一斉ログイン | 一斉処理 |
| --- | --- | --- |
| | 順次ログイン | |

**【要求された接続数での負荷】**

**要求された接続数で連続稼働させる。**
**連続稼働でリソースリークが発生しないか確認する。**

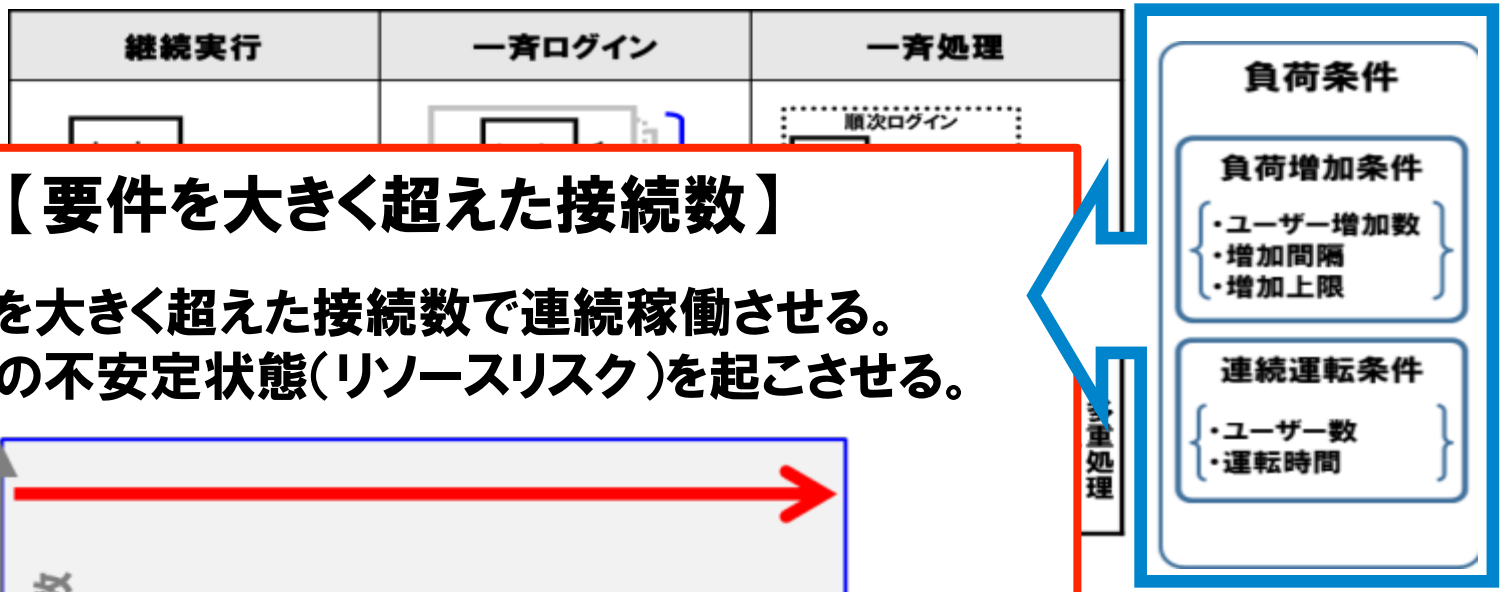



**負荷条件**

**負荷増加条件**
- ユーザー増加数
- 増加間隔
- 増加上限

**連続運転条件**
- ユーザー数
- 運転時間

# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ




**【要件を大きく超えた接続数】**

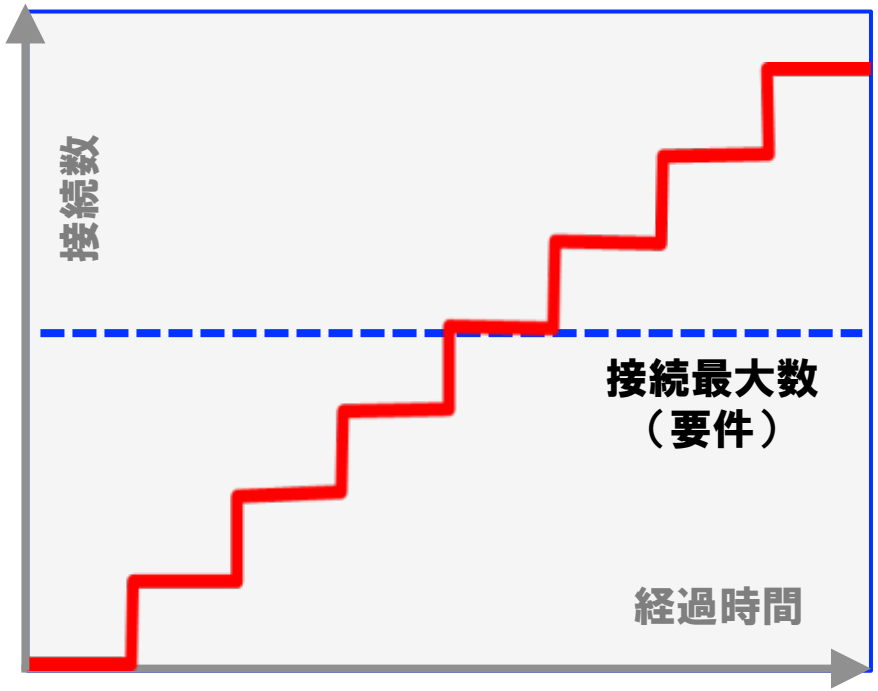
**要求を大きく超えた接続数で連続稼働させる。**
**システムの不安定状態（リソースリスク）を起こさせる。**

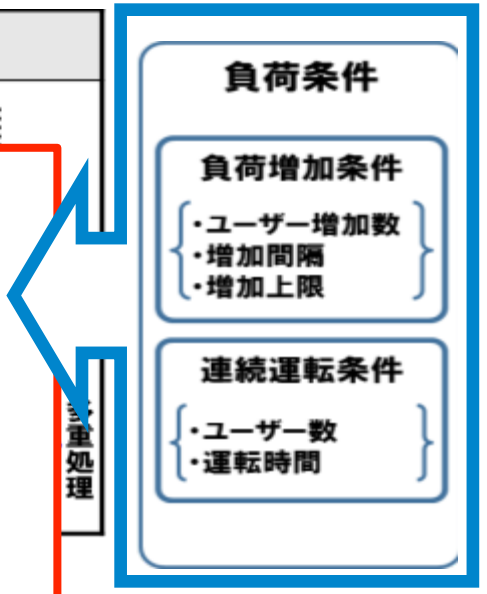
# コンパクト化施策①　性能要件の見える化による性能検証の効率化
## 負荷テスト設計に向けたシナリオと負荷状態の組み合わせ

| 継続実行 | 一斉ログイン | 一斉処理 |
| --- | --- | --- |
| | 順次ログイン | |

**【接続数を段階的に増やす】**

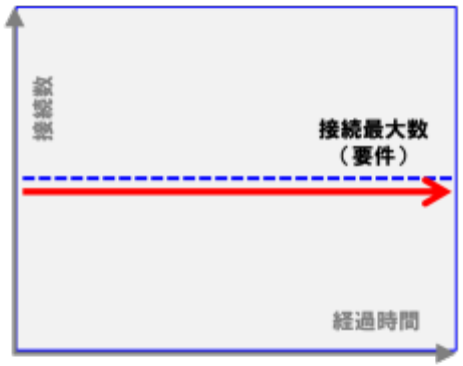
**接続数を一定間隔で増加させて、連続稼働させる。**
**増加によりシステムリソースの変化を見る。**



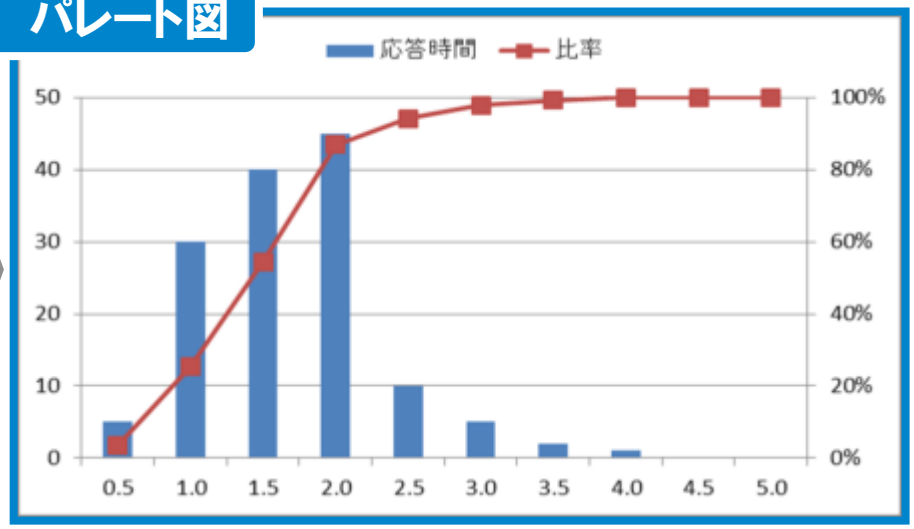
# 負荷条件と性能検証するグラフの関連付け

## 【最大接続数の一定負荷】

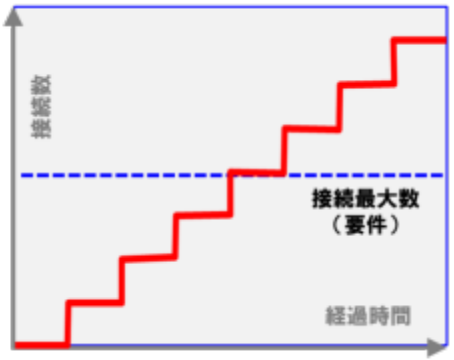



最大接続数で
一定時間稼働
↓
性能目標の
達成状況

**パレート図**

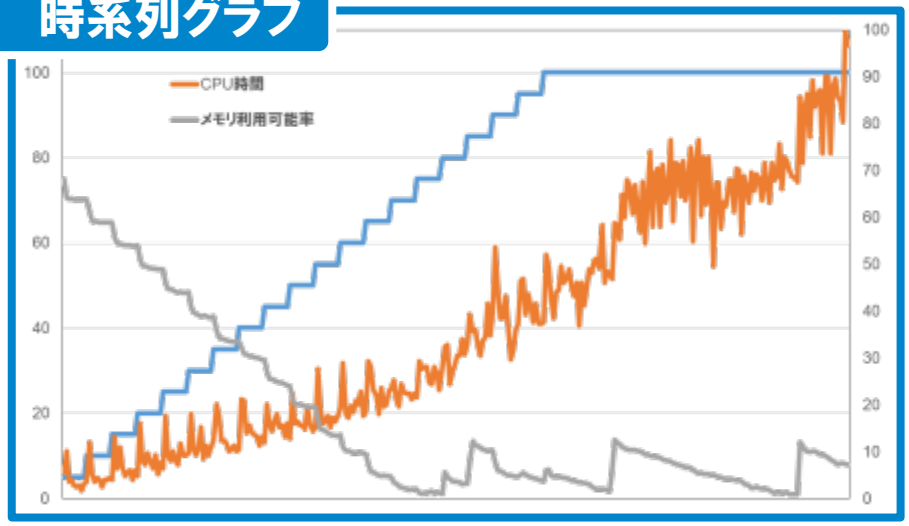



## 【接続数の段階的増加】




接続数増加で
一定時間稼働
↓
リソースリスク
を計測

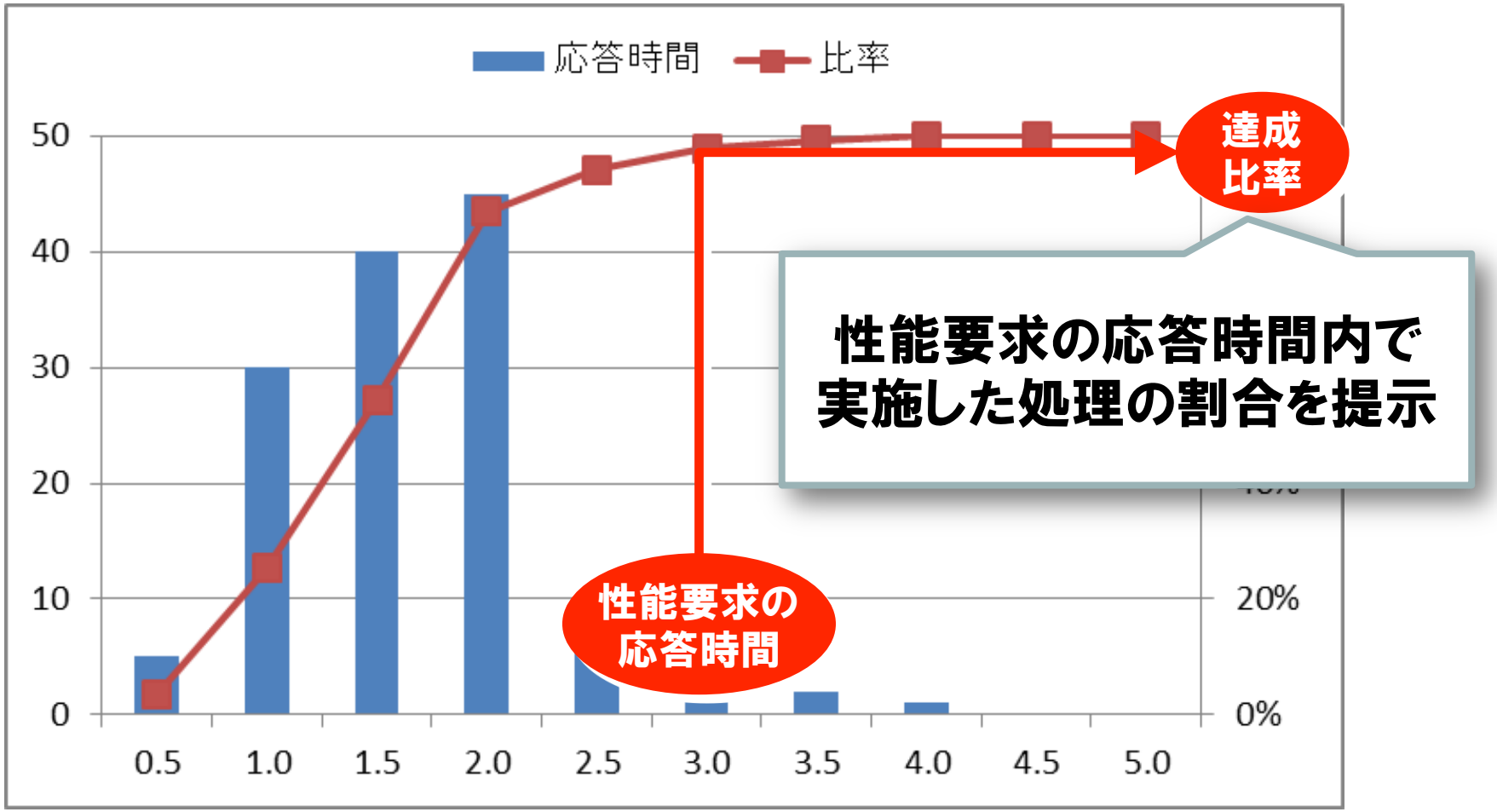
**時系列グラフ**



## コンパクト化施策②　検証結果の見える化による検証プロセスの短縮

# 「性能要件の達成度」の見える化

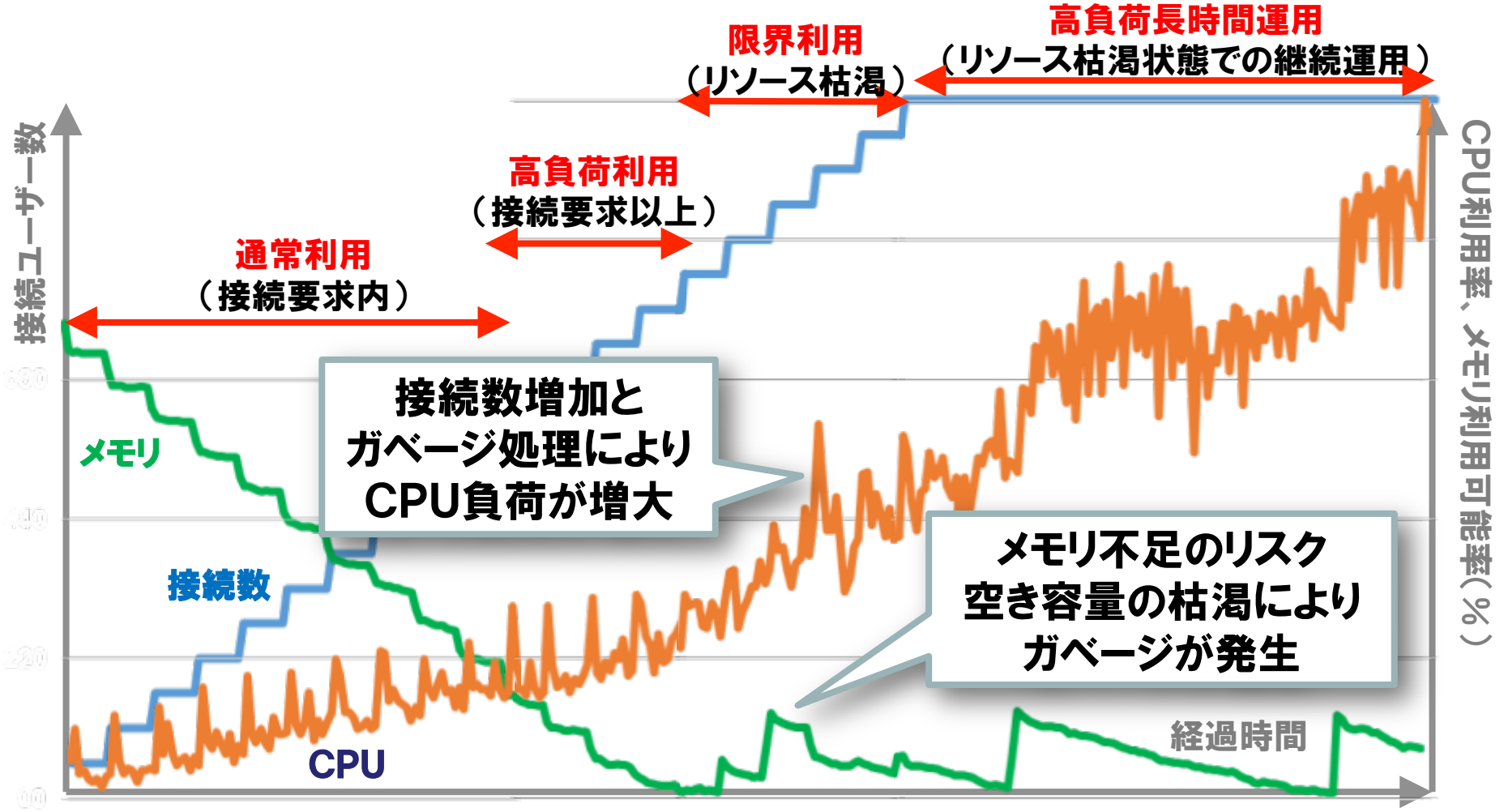
**パレート図により、**
**性能要件の達成度（未達成度）を定量的に判断可能に。**



## コンパクト化施策②　検証結果の見える化による検証プロセスの短縮

# 「システムのリソースリスク」の見える化

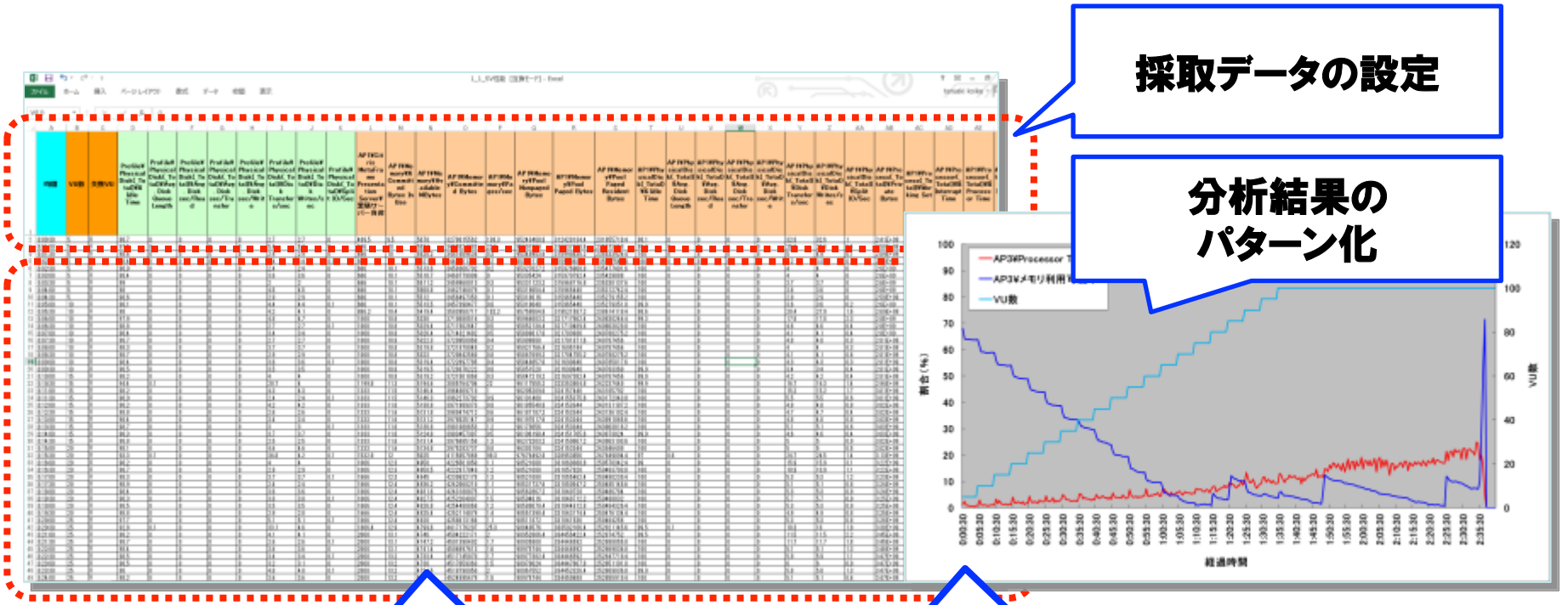
**時系列グラフにより、**
**負荷状況の変化によるリソースリスクを把握可能。**



## コンパクト化施策③　効率的な性能検証作業

# 分析パターンと採取データのテンプレート化

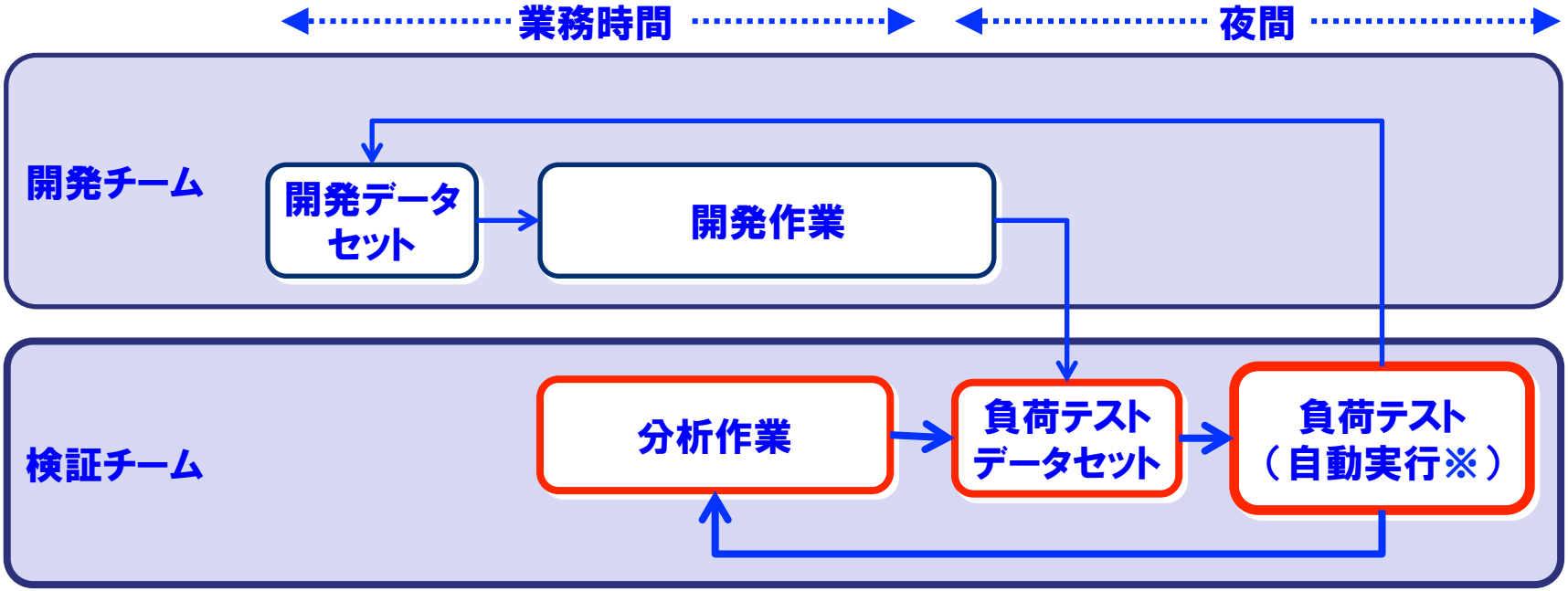
**負荷テストで採取したデータの集計作業効率化がポイント。**
**テンプレートを作成し、データのインポートでグラフを自動化**



実施にあたっての工夫点

# 負荷テストの自動実行化

**開発チームと同一の環境を使用した場合、負荷テストは限られる。**
**そこで、自動実行環境を構築し負荷テストを夜間に実施。**

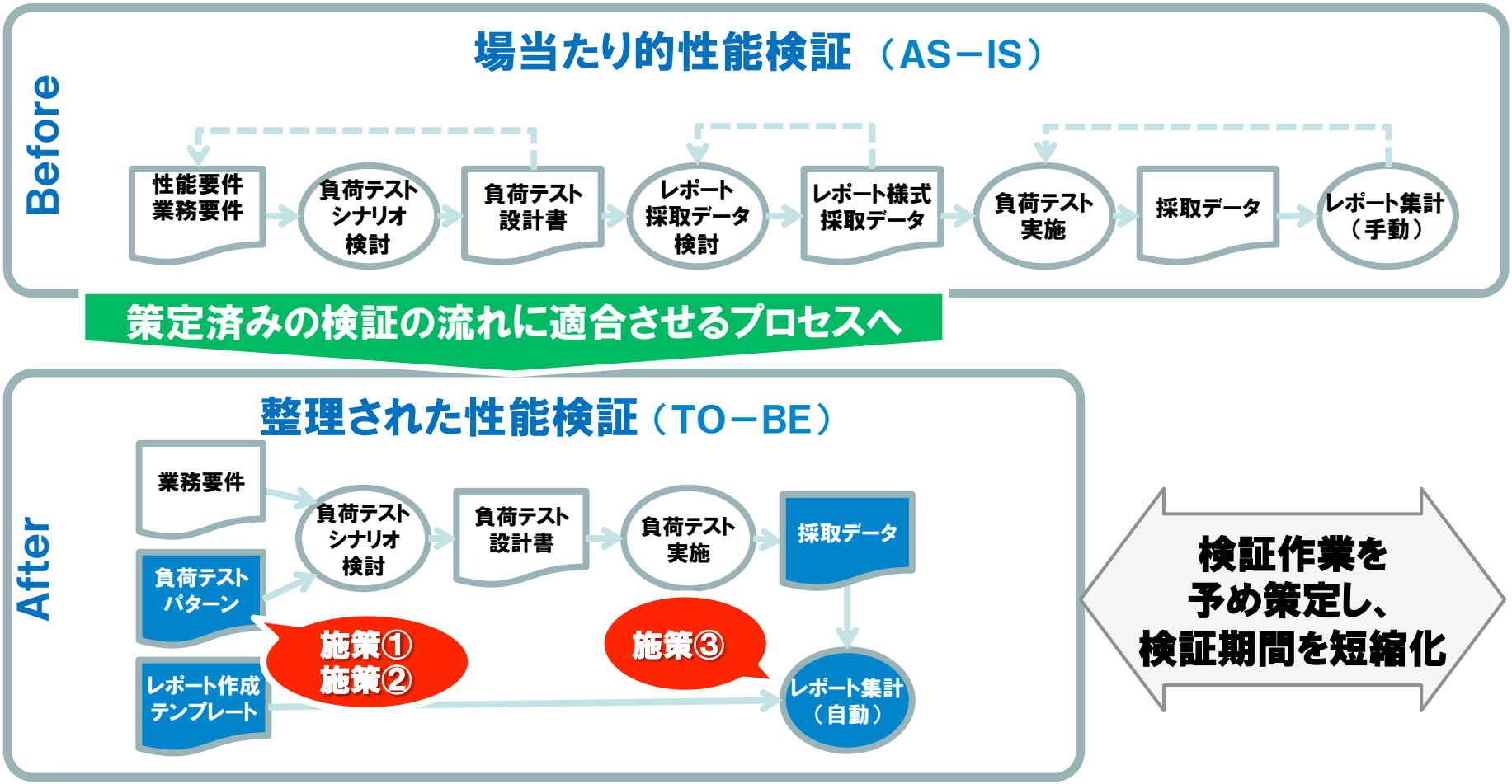



※負荷テストツールの起動をバッチ化しスケジュール起動で実現。

**夜間に負荷テストを自動で実行させることにより、**
**開発環境の占有時間を最短にし、分析作業に注力可能になった。**

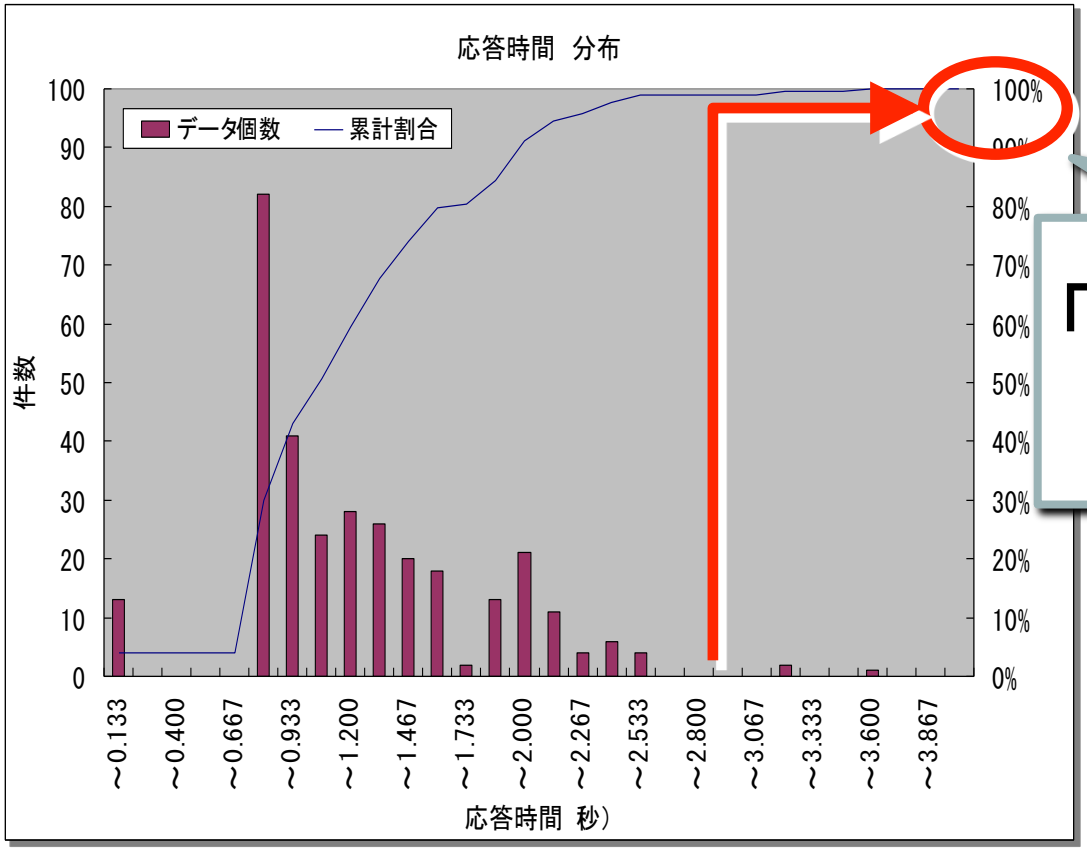
# 実施結果①
# 性能検証作業はコンパクト化されたか？

**開発案件毎に個別にアドホックに進めていた性能検証が短縮された**



# 実施結果②
# 「性能要件の達成度」の見える化

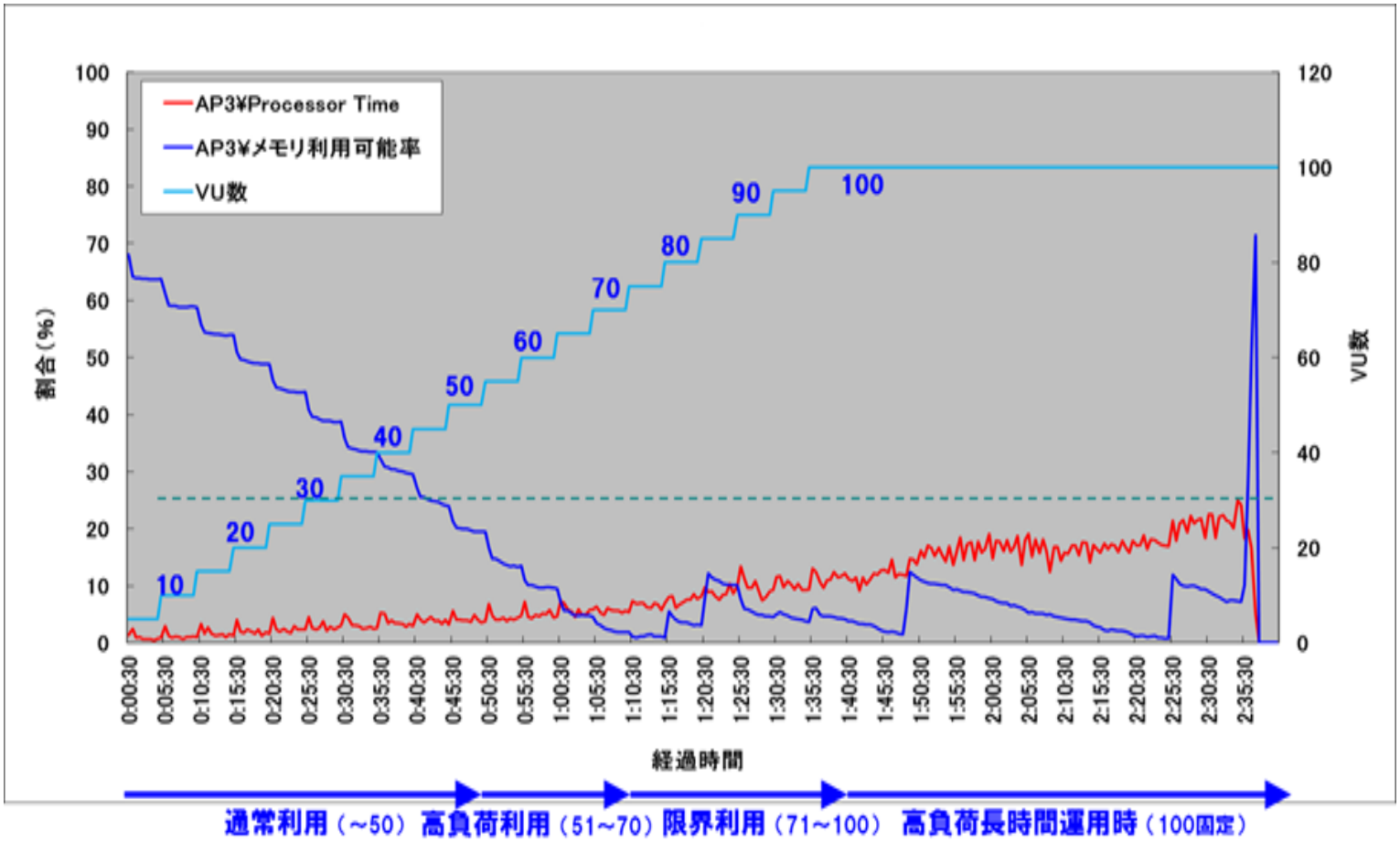
**同じ負荷をかけ続けた場合の性能要件の達成度をパレート図で確認。**
**達成度の判定だけでなく、応答時間の分布状況も把握可能。**



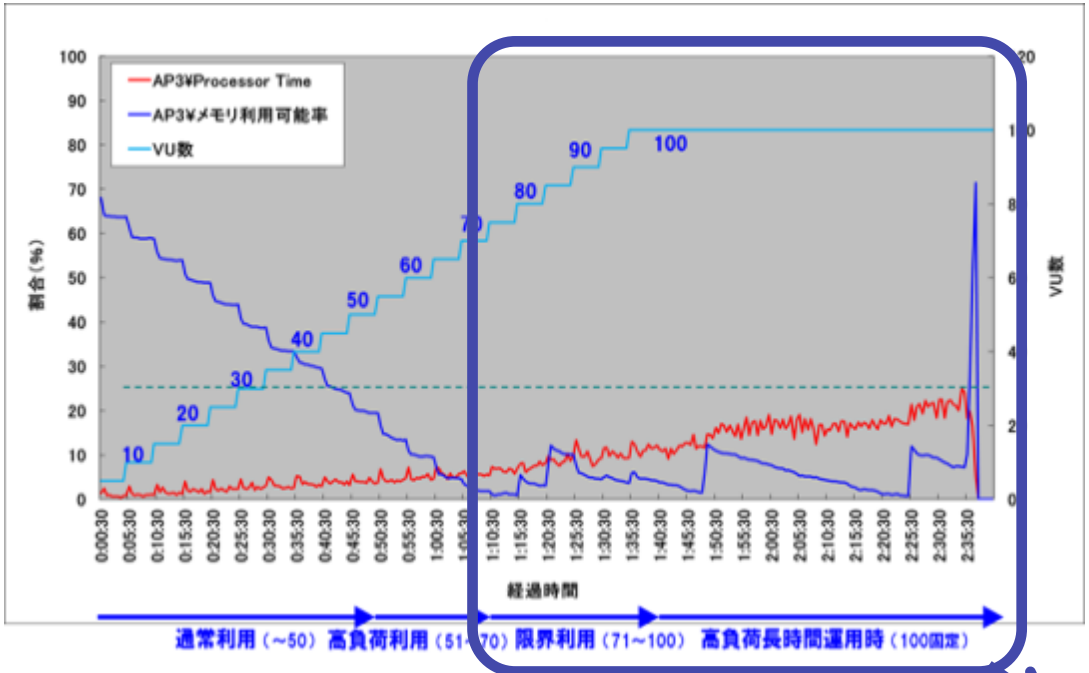

**「応答時間3秒以内」の達成率は“98％”**

# 実施結果③
# 「システムの限界」の見える化

**負荷を段階的に増加させたときのシステムの許容状態を把握**
**閾値を割り込むと急速にリソースが枯渇し、不安定になることを確認。**



# 実施結果③
# 「システムの限界」の見える化




負荷をかけた状態を
ステージとして定義することで、
システムの振る舞いを表現
↓
「何人まで使える？」に対する
回答が明瞭になった。

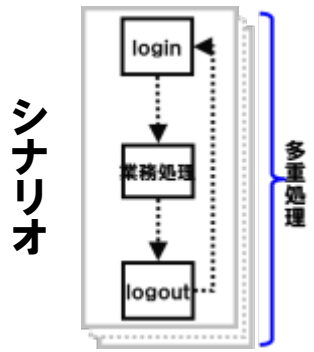
| ステージ | 通常利用 | 高負荷利用 | 限界利用 | 高負荷長時間運用 |
|---|---|---|---|---|
| ステージの説明 | 安定して利用可能 | 負荷が高くなり始める | 業務失敗のリスクが高い | システムが不安定になる |
| システム状況 | システムは安定、TATも安定。 | 応答時間は徐々に悪化。リソースも閾値近く。 | 業務を止めるなどの処置が必要。 | メモリが枯渇し、GCの発生が頻発 |

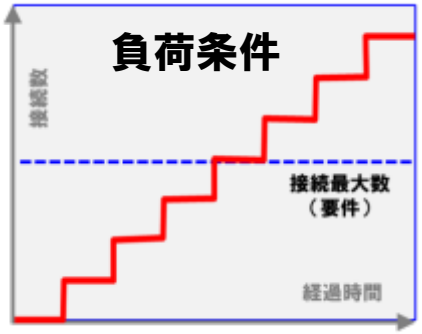
# 実践的な性能検証シナリオ

**シナリオ／負荷状態／検証のパターン化の組み合せにより**
**実業務を想定した負荷テストが可能になる。**

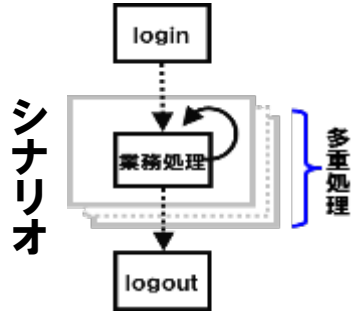
**例：小売店向けシステムで、セールによる売上が接続最大数を超えることが予想される場合。**

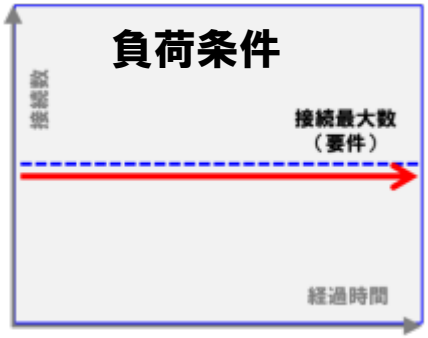
## 予想より多くの店舗から接続が増加



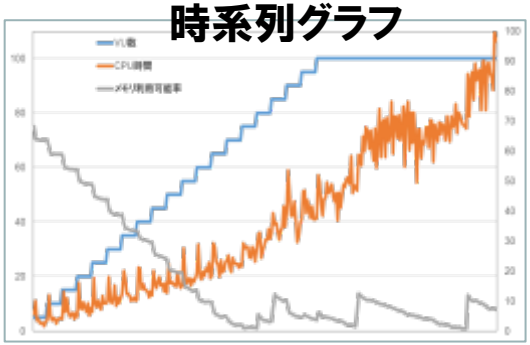




## 本店業務は担当スタッフが業務を実施



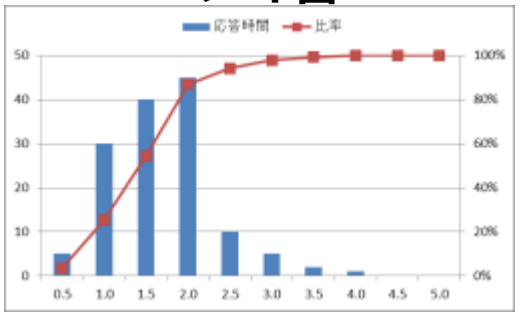




### 時系列グラフ



**要件を上回る接続数での**
**稼働による**
**システムリスクと**
**応答時間の達成率を把握**
**↓**
**対策立案**

### パレート図



# さいごに

- 従来、性能検証は対象のシステムに合わせて実施作業を個別に策定していた（AS－IS）。この方法では汎用性に乏しく、やり直しリスクも高い。
- そこで、本来あるべき性能検証作業を決め、実際の検証作業をこれにフィッティングさせる方法に変更した（TO－BE）
- このことにより、負荷テストで何をするのか、そのために性能要件をどのように設定するのかが見えてきた。
- また、集計作業の効率化や負荷テストの自動実行により、検証作業を実施するための制約が緩和された。
- 今後はシステム構成に合わせたバリエーションを増やし、効果を高めていきたい。

# 使用した検証ツールと参考文献

## 検証ツール

- XenApp**環境**：Citrix® EdgeSight for Load Testing
- Web**アプリケーション**：Oralce® Load Testing

## 参考文献

- **IIMパフォーマンス管理セミナー資料**
- **非機能要求グレード（IPA）**

Empowered by Innovation