デグレ防止の自動テストと即時リリースの取り組み

株式会社パラミックス
中村　幸太
2025/10/23

Agenda

PARAMIX

１．背景と改善目標
２．自動テスト導入
３．24時間いつでもリリース
４．改善による変化や効果

１．背景と改善目標

1.1 背景

PARAMIX

📈 ソースコードの増加に伴う課題

- 年々、ソースコード量が増加
- 機能追加・改修時に**既存機能へのデグレが頻発**
- デグレの**修正に多くの工数がかかり、新機能開発の時間が取れない**

🚫 リリース運用の課題

- 不具合修正後も、**定時後でしかリリースできない**

😡 利用者満足度の低下

- 機能改善の遅れ
- 不具合対応の遅延

1.2 改善したいこと

PARAMIX

☑ デグレの未然防止

- リリース前に**既存機能の不具合を検出**
- **デグレ件数を7割削減**を目指す

🚀 24時間いつでもリリース

- 従来：**定時後のみリリース可能**
- 改善後：**24時間いつでもリリース可能**に

🛠 保守性の向上

- バグ修正時間を削減して、機能追加時間を増やす
- 機能追加の生産性向上

1.3 改善策を導き出した経緯

PARAMIX

4つのキーメトリクス: 2023年の調査（3万6千件以上の回答をクラスター分析）

2023

	エリート	ハイパフォーマー	ミディアム パフォーマー	ローパフォーマー
リードタイム	1日未満	1日から1週間	1週間から1ヵ月	1週間から1ヵ月
デプロイ頻度	オンデマンド (1日複数回)	1日1回から週1回	週1回から月1回	週1回から月1回
MTTR	1時間未満	1日未満	1日から1週間	1ヶ月から半年
変更失敗率	5%	10%	15%	64%
構成比	18%	31%	33%	17%

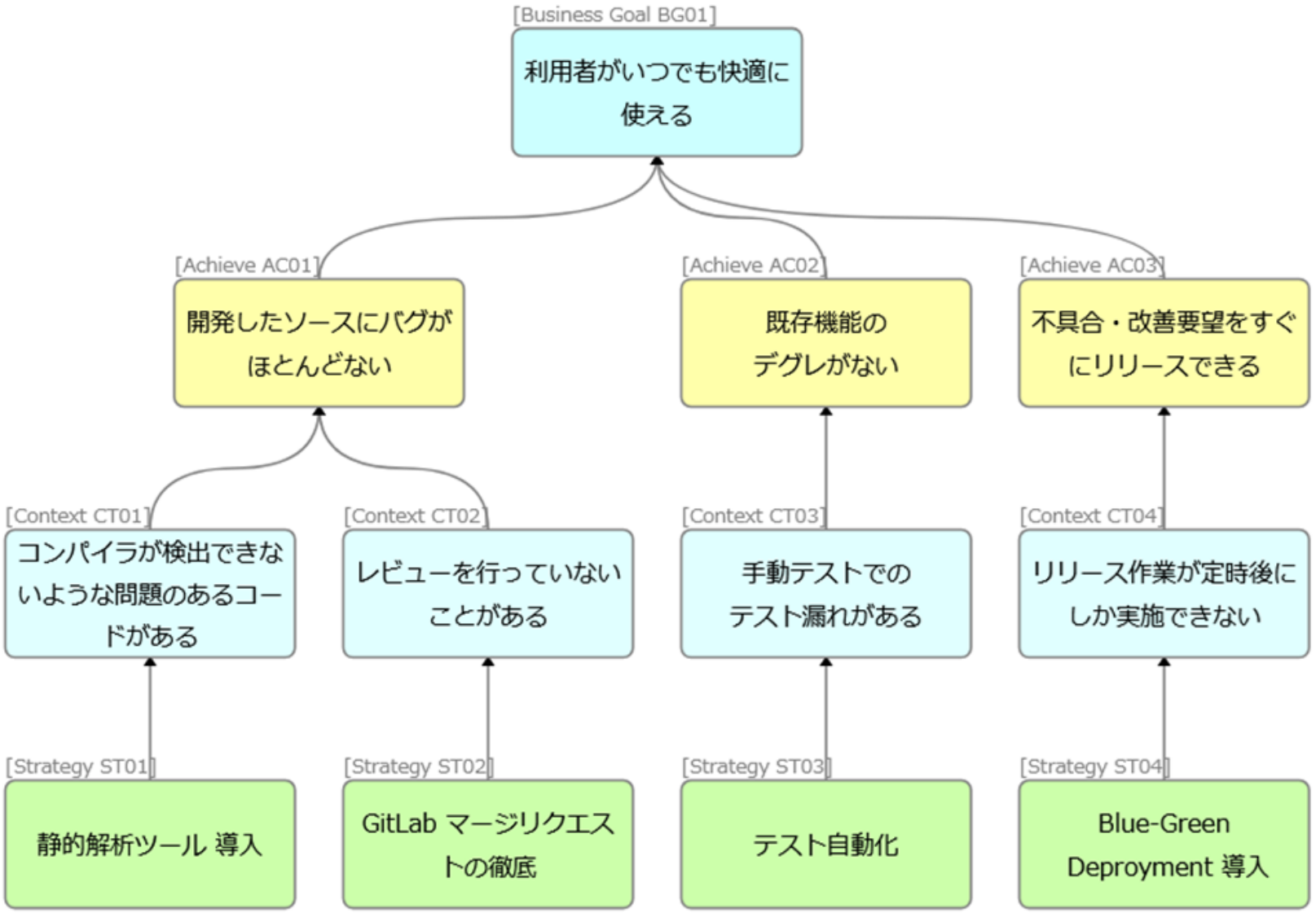
https://cloud.google.com/devops/state-of-devops/

引用元：和田卓人, “組織に自動テストを書く文化を根付かせる戦略 2024秋”, SPI Japan 2024, 2024

1.4 目標に対する施策を抽出

PARAMIX

目標施策の全体像を図式化



PARAMIX

２．自動テスト導入

2.1 自動テスト戦略の見直し

PARAMIX

🎯 背景と課題

- 当初は **Selenium による E2E テスト**を導入
  ➤ リリース後のデグレ対策を目的
- しかし…
  - バグの早期検知につながらない
  - 自動テストの品質評価も困難

📏 見直しの方針

- **自動テストの品質指標**として
  ➤ コードカバレッジ（ Coverage ）を採用
- Selenium では JavaScript の **Coverage が測定困難**
- Coverage が測れる **Jest** に移行

2.2 Jest の特徴

PARAMIX

🎯 特徴

- Facebook 製の JavaScript テストフレームワーク
- **高速実行**：テストの並列実行 & 差分検知で無駄を削減
- **カバレッジ測定が標準搭載**
- **モック機能が強力**：依存関係を簡単に置き換え可能
- **設定が少なく導入が容易**（Zero Config）
- **スナップショット**：UI や HTML 出力の変化検知が可能

2.3 Jest のサンプルコード

PARAMIX

```
// sum.ts
export function sum(a: number, b: number) {
  return a + b;
}

// sum.test.ts
import { sum } from './sum';

 test('sumテスト', () => {
  expect(sum(1, 2)).toBe(3);
});
```

2.4 HTML表示テスト効率化

PARAMIX

HTML 上の テーブルをコピー

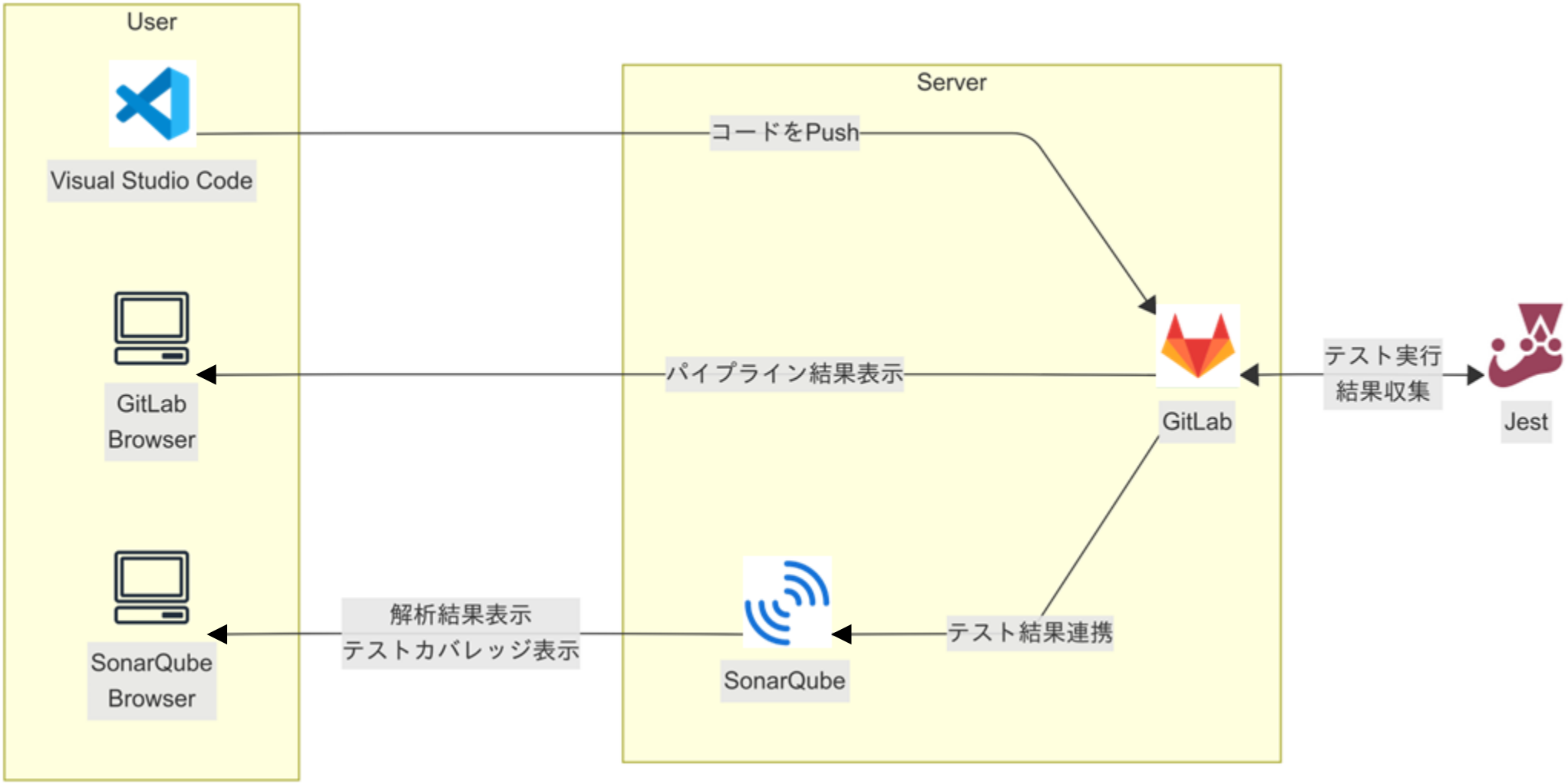
No.	テーブル	テーブル名称	CRUD
1	team	チーム	CRUD
2	h_userlang	言語区分・社員	CR
3	h_syain	ユーザー	R
4	kpt_item	KPT項目	R

エディタに貼り付けてマクロを実行

```
const p_table = new dadvJestTable('テーブルID')
const p_expect = [
    [0, 'No.', 'テーブル', 'テーブル名称', 'CRUD'],
    [1, '1', 'team', 'チーム', 'CRUD'],
    [2, '2', 'h_userlang', '言語区分・社員', 'CR'],
    [3, '3', 'h_syain', 'ユーザー', 'R'],
    [4, '4', 'kpt_item', 'KPT項目', 'R’],
]
p_table.expectTable(p_expect)
```

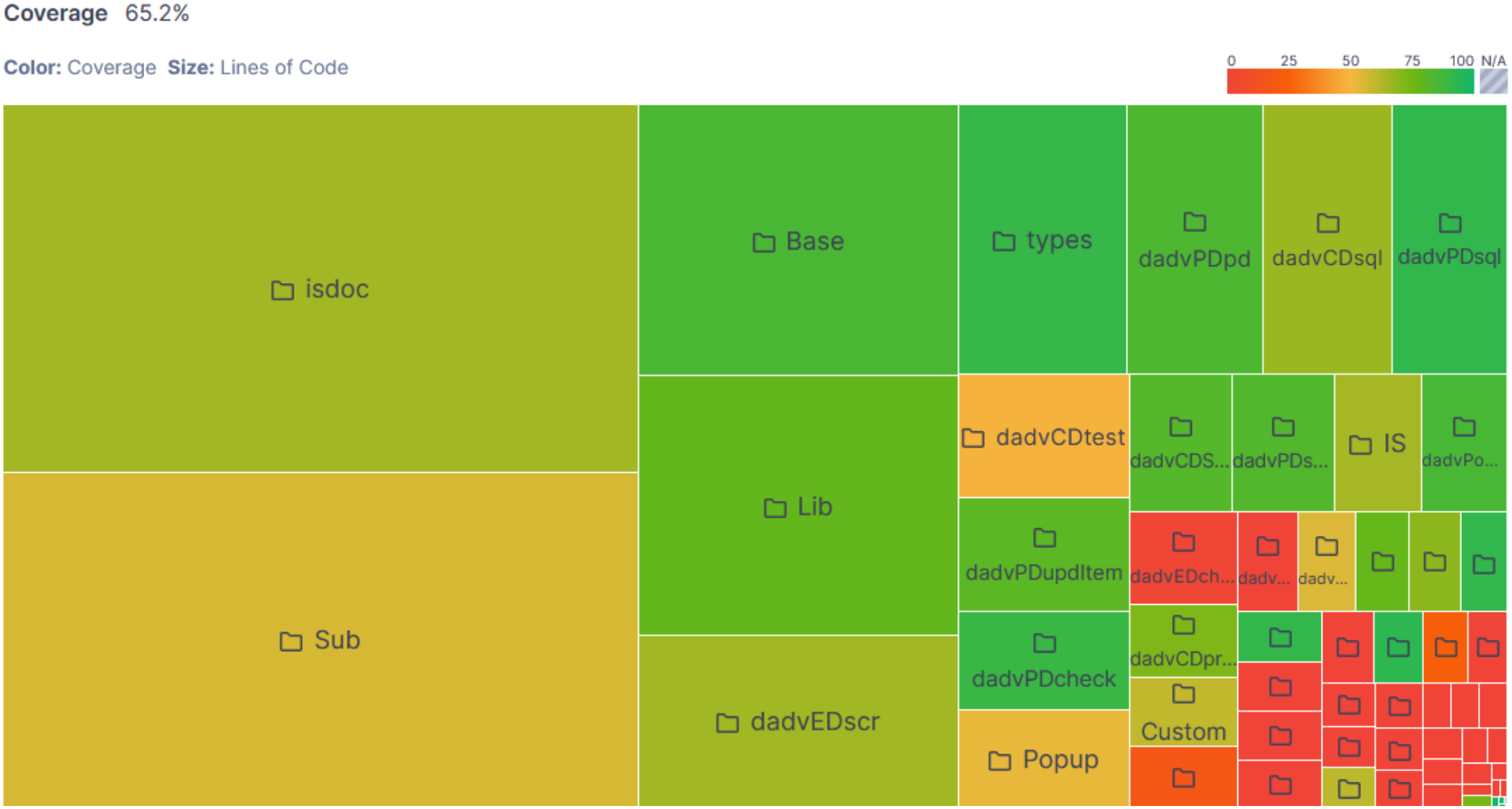
2.5 自動テストの自動実行

継続的な品質チェックの構成を導入



2.6 自動テスト Coverage

PARAMIX



2.7 自動テスト Coverage

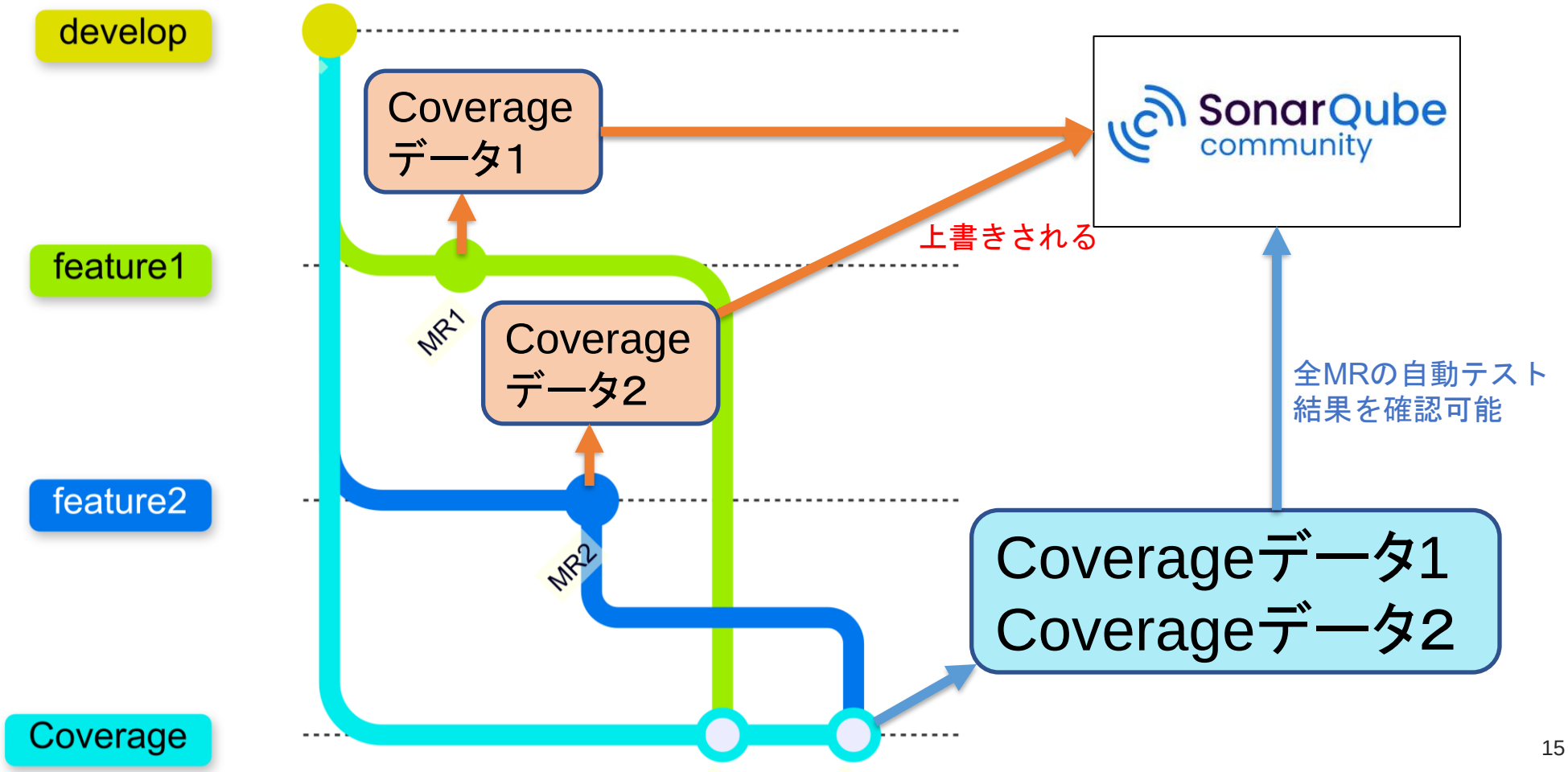
緑:テストが通った行
赤:テストが通っていない行

```
                SELECT句生成時の設定を取得する

electConf(x_mode: string) {
let p_conf: typeConf = {}
switch (x_mode) {
    case SQL_MODE.xpd:
        p_conf.select = '\t<Select>\n'
        p_conf.sent_start = '\t\t'
        p_conf.sent_end = '\n'
        p_conf.select_end = '\n\t</Select>'
        break
    case SQL_MODE.java:
        p_conf.select = '//SELECT\nString p_select = "" +\n'
        p_conf.sent_start = '\t"'
        p_conf.sent_end = '\\n" +\n'
        p_conf.select_end = '";\np_sql.setField(p_select);'
        break
    case SQL_MODE.pure:
    default:
        p_conf.select = 'SELECT\n'
        p_conf.sent_start = '\t'
        p_conf.sent_end = '\n'
        p_conf.select_end = ''
        break
}
return p_conf
}
```

2.8 自動テストのレビュー

課題：複数マージリクエスト(MR)があるとCoverage データが最後のMRで上書きされレビューが正しくできない
解決策：Coverageブランチで管理



PARAMIX

３．24時間いつでもリリース

3.1 Blue-Green Deployment

PARAMIX

☑ Blue-Green Deploymentとは？

- **2つの環境（Blue環境／Green環境）** を用意
  - 片方（例：Blue）が本番稼働中
  - もう片方（例：Green）に新バージョンをデプロイ
- 環境切り替えでサービスを停止せずにバージョンUP

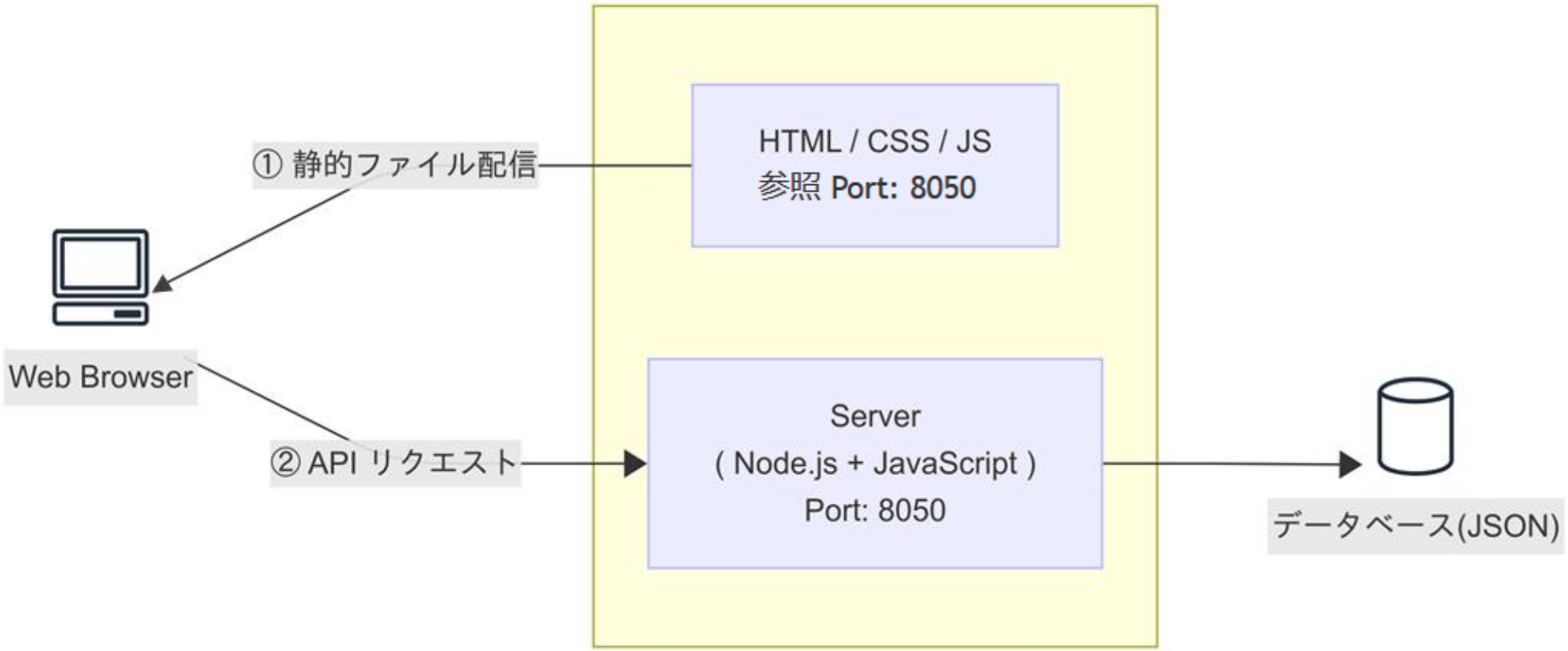
🎯 Blue-Green Deploymentのメリット

- 無停止デプロイが可能
- 万が一の際、即座にロールバックが容易

💡 今回、この手法を採用！

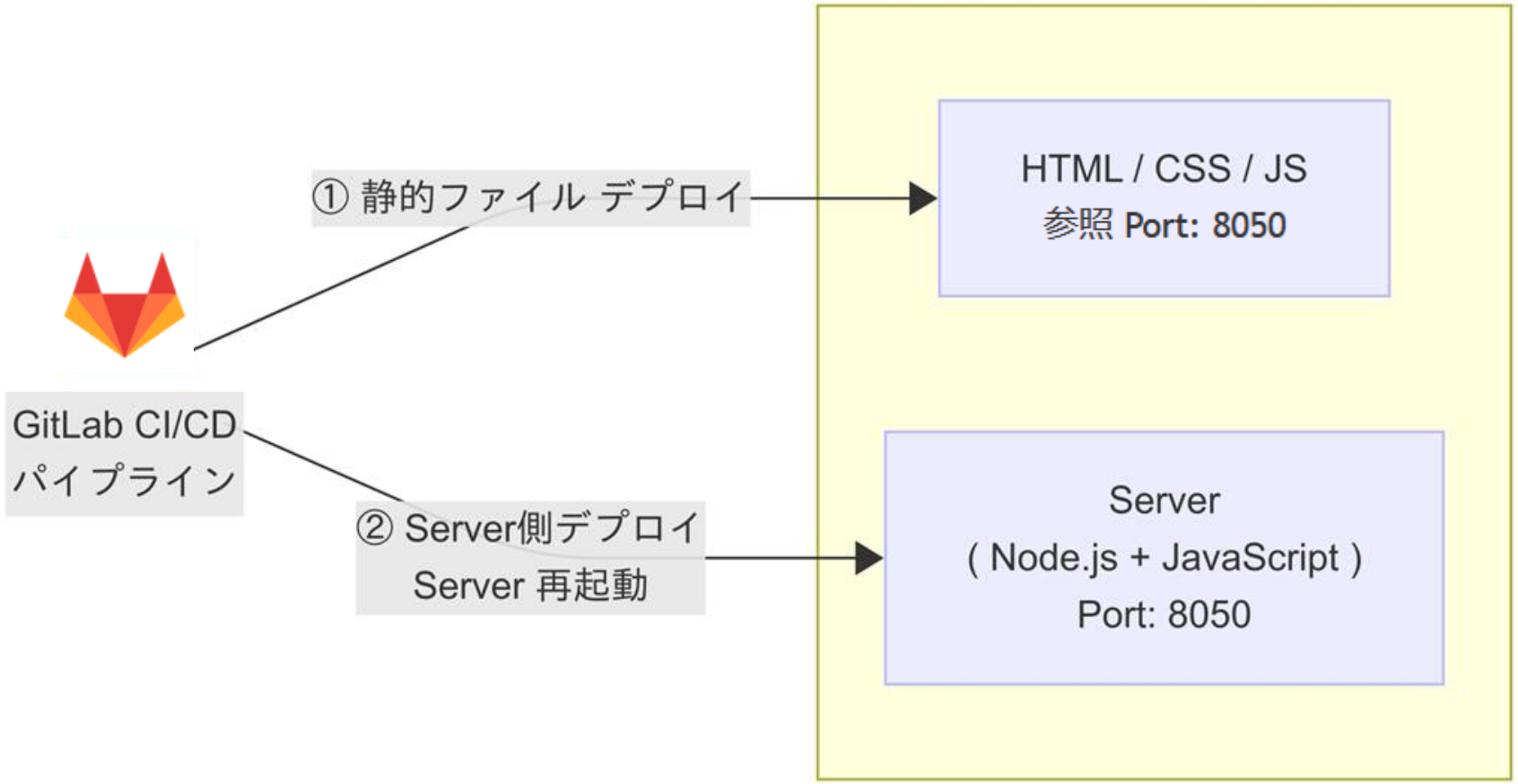
3.2 従来のシステム構成

- 当システムは **Client & Server アーキテクチャ**で構成
- Client は ブラウザ上のJavaScript によってフロントを担当
- Server は Node.js 上で JavaScript によりデータ処理を担当



3.3 従来のリリース課題

- Node.js Server は ポート 8050 固定で稼働
- リリース時には 稼働中プロセスの再起動が必要
- サービス一時停止を伴うため、業務時間内のリリース不可



3.4 24時間いつでもリリース導入

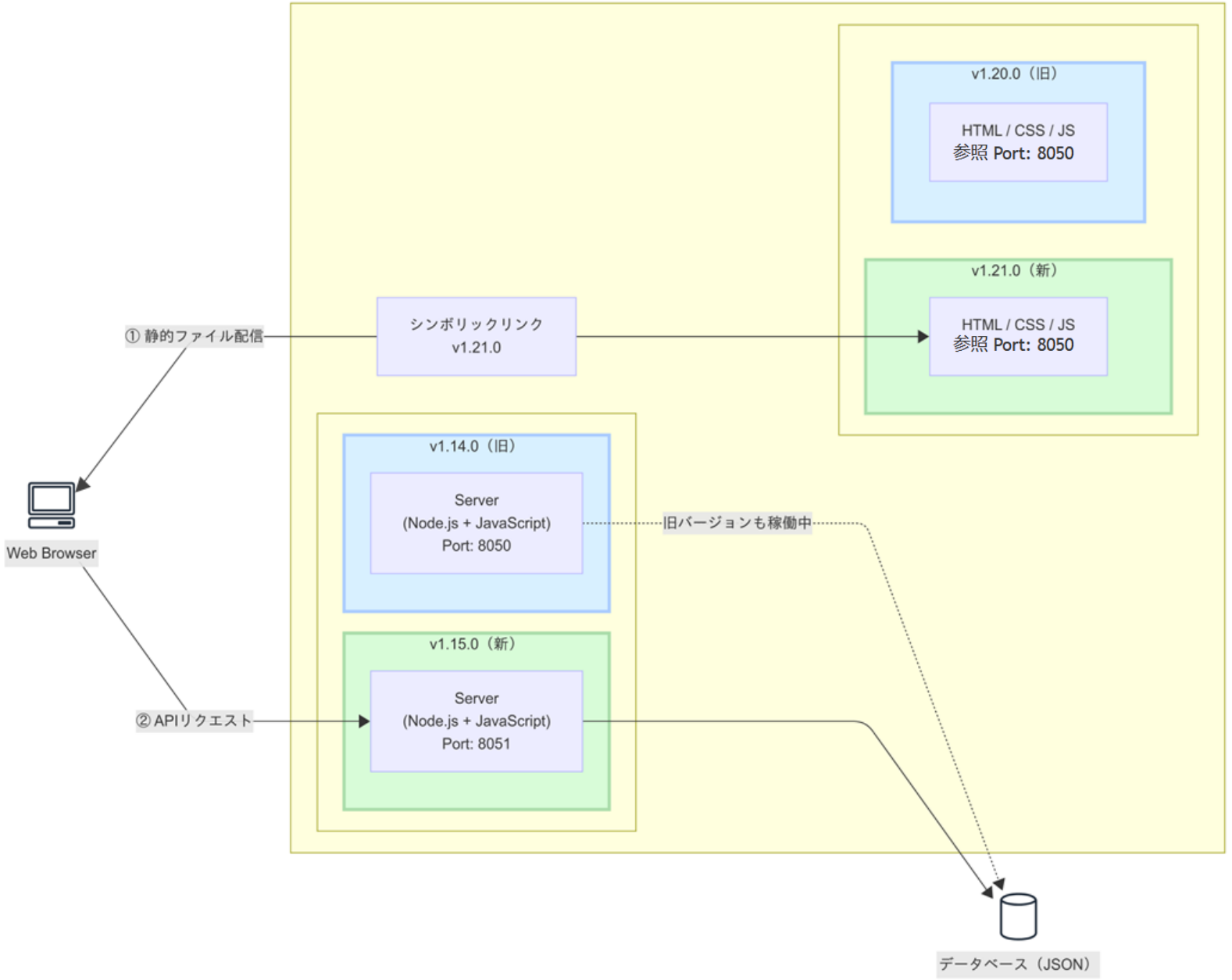
⚙ Blue-Green Deployment 導入後のシステム構成

- Blue／Green 環境はそれぞれ**ポート8050～8059のいずれか** で待機
- **新バージョンは空きポートへ配置**
- 切り替え時に **シンボリックリンクで参照先を更新**

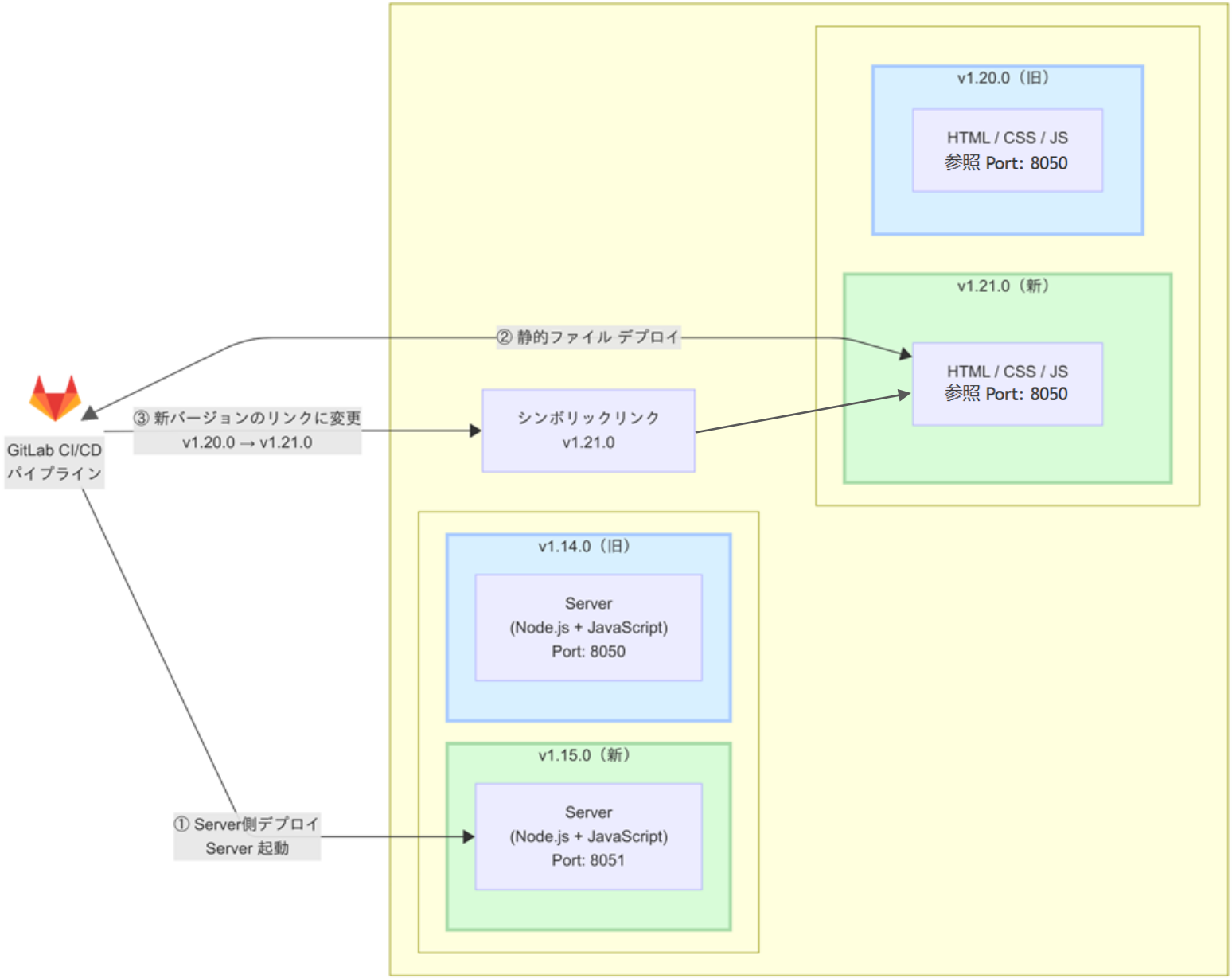
📊 導入効果：

- **24時間いつでリリース可能**
- **デプロイ中も既存サービスに影響なし**
- **人的負担・夜間作業の削減**

3.5 導入後の利用イメージ



3.5 導入後のリリース方法



４．改善による変化や効果

4.1 自動テストによるデグレ削減

自動テストを作成した機能のデグレは1件のみ

期間	自動テスト導入	デグレ件数	テストカバレッジ
2025年2月	✕ 未導入	15件	-
2025年3月	🔄 導入中	11件	約30%
2025年4月	🔄 導入中	7件	約40%
2025年5月	🔄 導入中	5件	約50%
2025年6月	☑ 運用定着	0件	約60%
2025年7月	☑ 運用定着	1件 (*1)	約60%
2025年8月	☑ 運用定着	1件 (*1)	約65%
2025年9月	☑ 運用定着	2件 (*1)	約65%

(＊1) 4件中3件は自動テスト未実装の機能

4.2 コード理解性の向上

PARAMIX

🔧 自動テストによるコード理解性の向上

- テストコードから**引数・返値の仕様を即座に把握**可能に
- テストコードが**利用例の役割**を果たすことで

→ **実装意図の理解速度が向上**

4.3 24時間いつでもリリース

PARAMIX

Blue-Green Deployment の導入効果

- サービスを停止せずにバージョンアップが可能に
- ユーザからのバグ報告に対し、定時まで待たずに修正リリースを完了
- 開発チームの対応負荷も大幅に軽減
- ユーザ満足度の向上にも貢献


PARAMIX