agent-browser

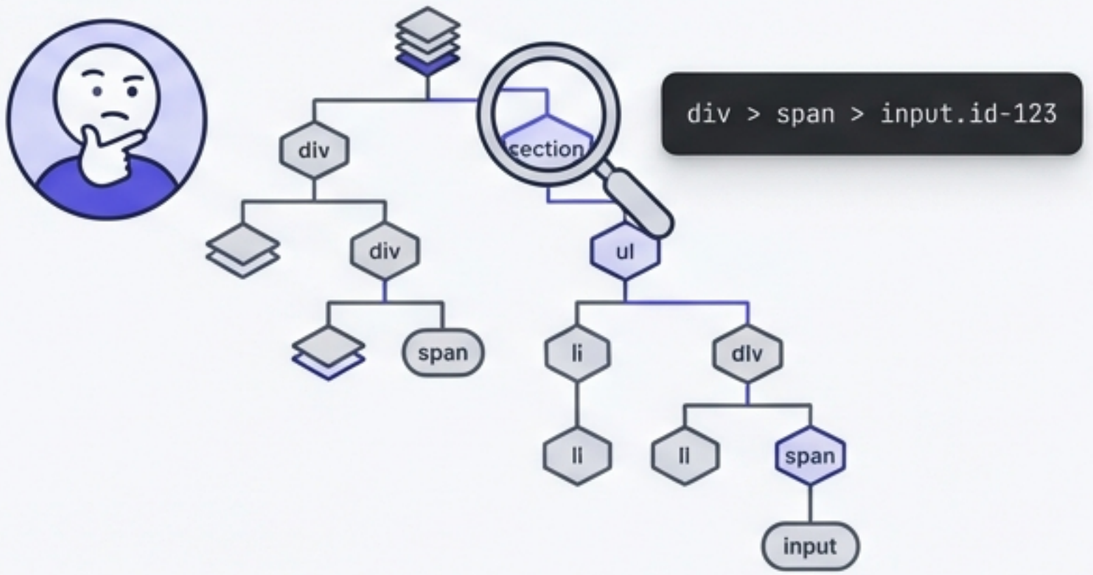
AIエージェントのためのブラウザ操作インターフェース

なぜ Vercel Labs は Playwright を再構築したのか?
「人間が書く」から「AIが叩く」への転換点。

Developed by Vercel Labs | Rust-based CLI | GitHub Stars 11k+

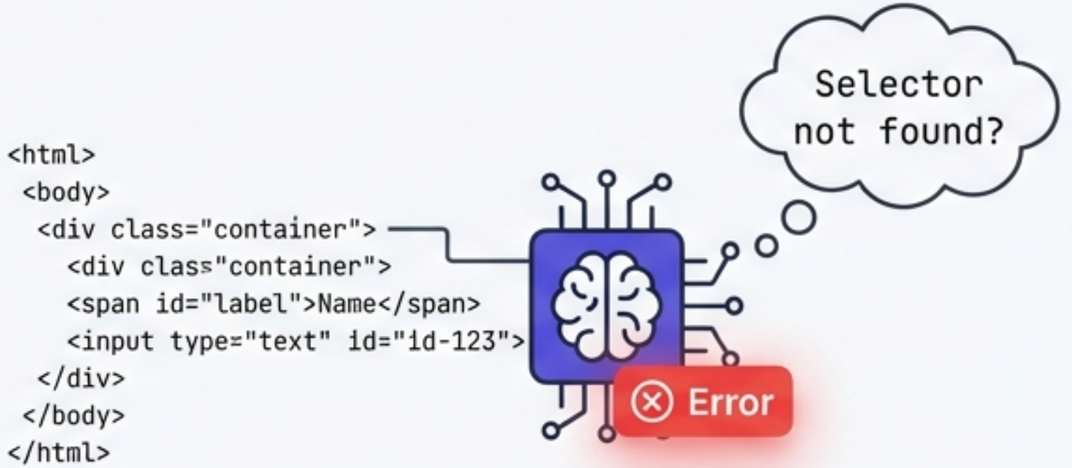
自動化のボトルネックは「人間」から「AIの翻訳コスト」へ

Era 1: Human Manual Coding



従来の自動化は、人間がDOM構造を解析し、コードを書くことを前提としていました。

Era 2: AI Automation



AIに従来のツールを使わせると、APIが複雑すぎ、かつセレクタの推論が不安定であるため、実行エラーが頻発します。

結論：AIには、人間用のツールではなく、AI専用の「抽象化レイヤー」が必要です。

解決策：Webページを「AIが読める形式」に変換する

Complex Web Page
Raw HTML / DOM

agent-browser
Inter / JetBrains Mono

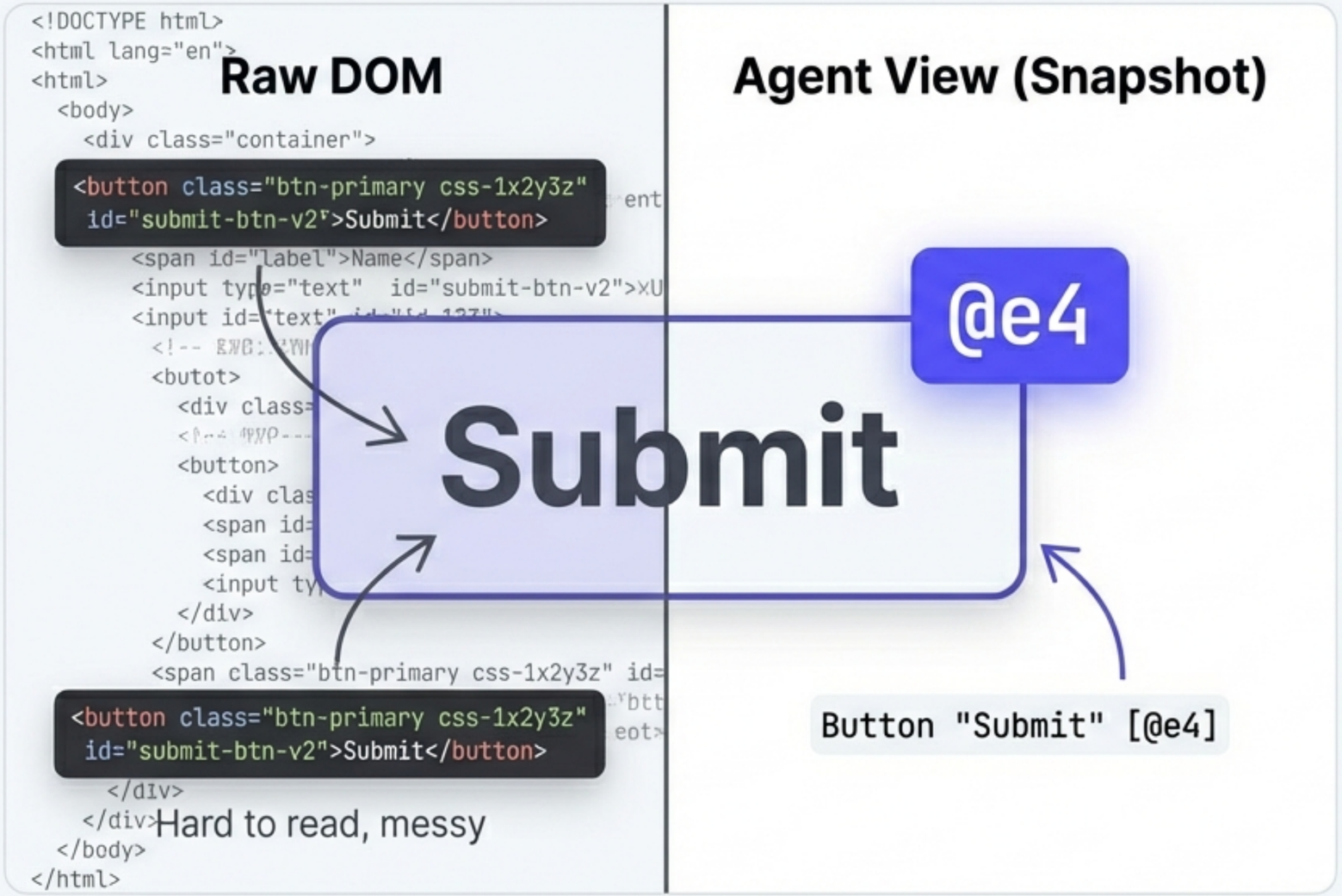
Structured JSON for AI
Inter

AIエージェント専用 CLI レイヤー (Playwright Wrapper)
Carbon Gray Inter

学習コストと 実行エラーを最小化
Carbon Gray Inter

公開18日で 100,000 DL突破 / GitHub Stars 11,670+
Carbon Gray Inter

最大の発明：抽象化された「Refシステム」



Snapshot: ページ構造をスナップショット化し、一時的なID（Ref）を付与します。

Stability: AIは `div.main...` ではなく `click @e4` と指示するだけで操作可能です。

Benefit: デザイン変更や動的クラス名の影響を受けず、AIが迷わずに要素を特定できます。

比較：Playwright vs agent-browser

Playwright / Human Context

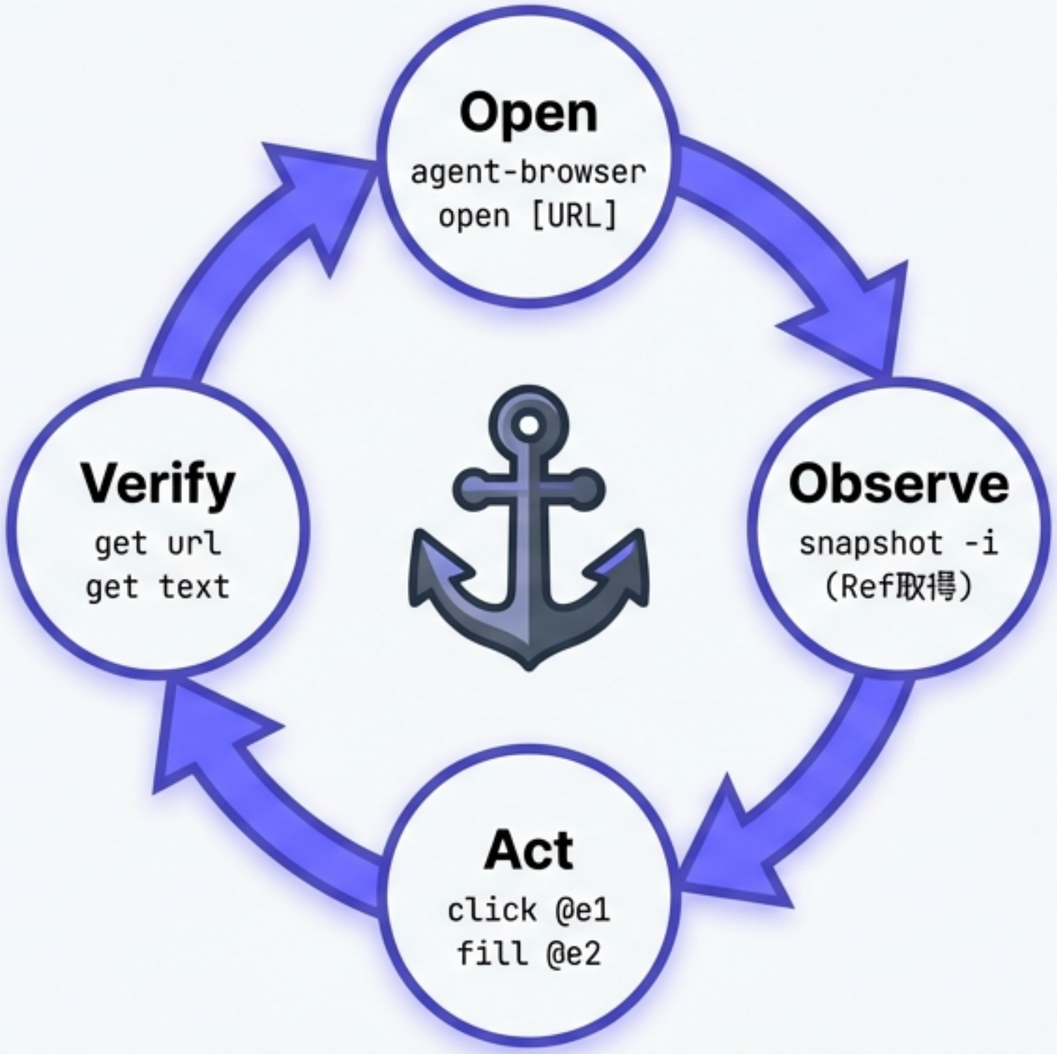
```
// 構造解析とセレクタ特定が必要
import { chromium } from 'playwright';
const page = await browser.newPage();
await page.goto('https://example.com');
await page.locator('input[name="email"]')
.fill('test@ex.com');
await page.locator('button[type="submit"]')
.click();
```

agent-browser / AI Context

```
# Refを見るだけで操作可能
agent-browser open example.com
agent-browser snapshot -i
# -> Output: textbox "Email" [@e3], button
"Submit" [@e4]
agent-browser fill @e3 "test@ex.com"
agent-browser click @e4
```

コード量の削減だけでなく、「DOM構造への依存」を排除した点が本質的な違いです。

スナップショット駆動のワークフロー

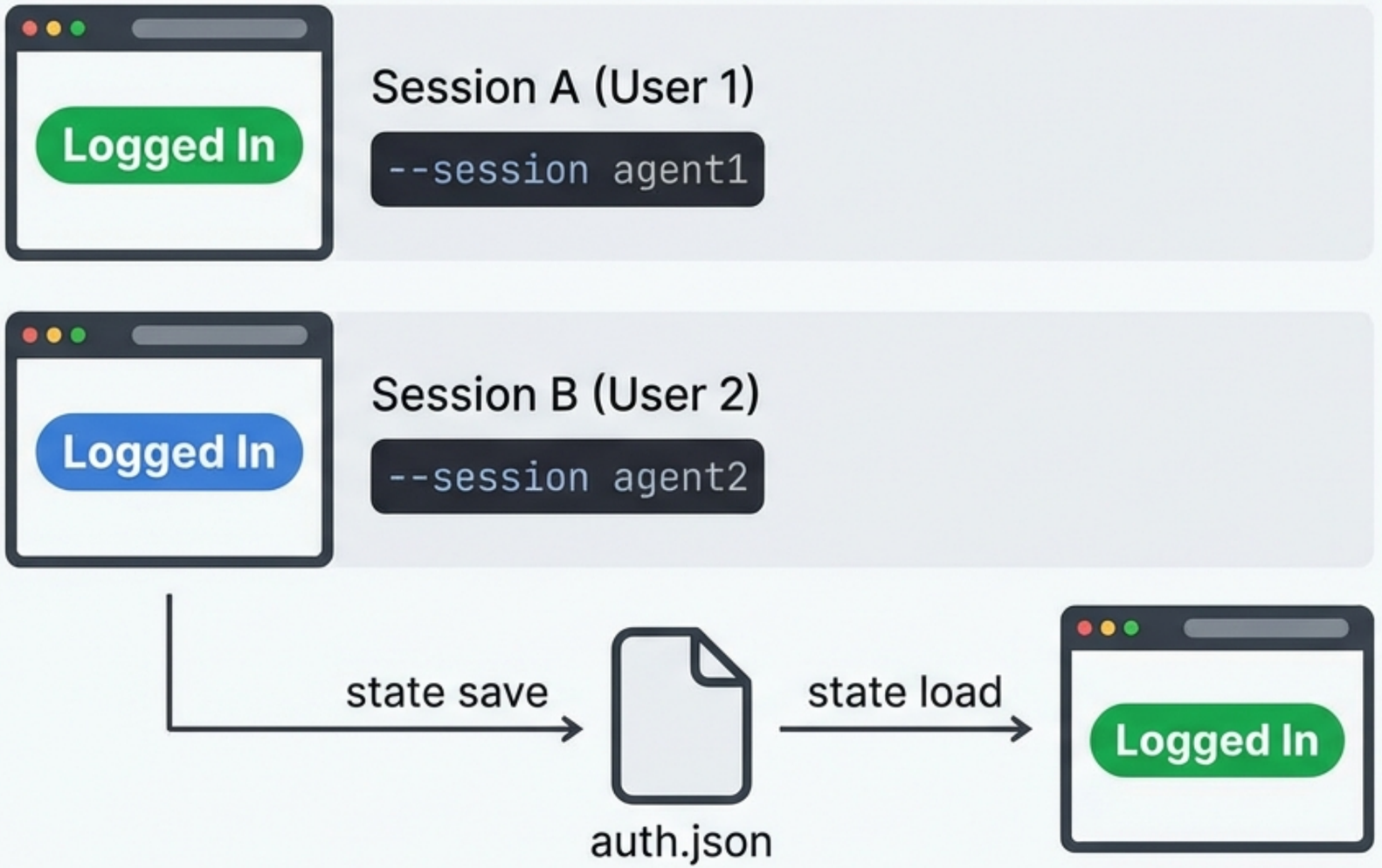


決定的（Deterministic）な自動化

このループを回すことで、非同期処理やDOM変化に強い自動化を実現します。

AIは常に最新の状態（Snapshot）に基づいて判断するため、ハルシネーションのリスクが低下します。

認証とセッション管理の革命



機能一覧

- **Session Isolation*: フラグ一つで、CookieやLocalStorageが分離されたコンテキストを起動。
- ***State Persistence*: ログイン状態を永続化し、毎回のログイン処理をスキップ。テストを高速化。

人間とAIの協業「ストリーミング・デバッグ」

ペアブラウジング

- **Problem**: 通常のヘッドレスブラウザは動作が見えず、AIがどこで詰まっているか判断しにくい。
- **Solution**: 環境変数を指定するだけで、WebSocket経由でブラウザ画面をリアルタイム配信。
- **Benefit**: AIの操作を人間が監視し、必要なら介入するハイブリッド運用が可能。

アーキテクチャとパフォーマンス

主要な仕様

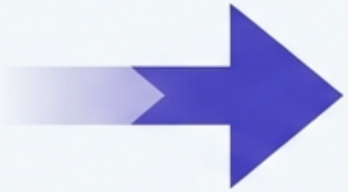
- **Rust Implementation**: CLI部分はRustで記述され、高速な起動と実行を実現。
- **JSON Native**: `snapshot --json`でLLMに最適な構造化データを提供。
- **Compatibility**: あらゆるAIコーディングツールと即座に連携可能。

ユースケース１：E2Eテストの「脱・コード化」

Maintenance Heavy

Maintenance Heavy

厳密なコードベースのテスト



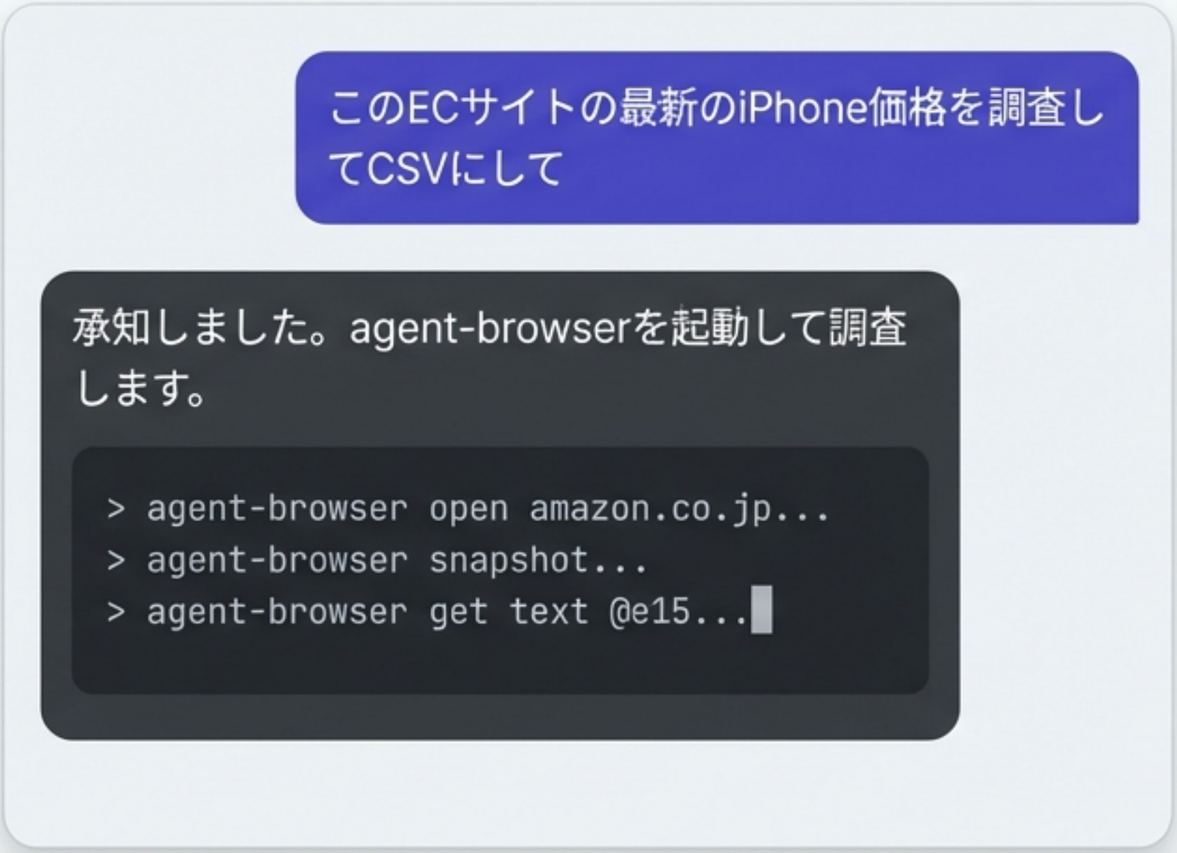
Lightweight Validation

Lightweight Validation

AIによるスモークテスト

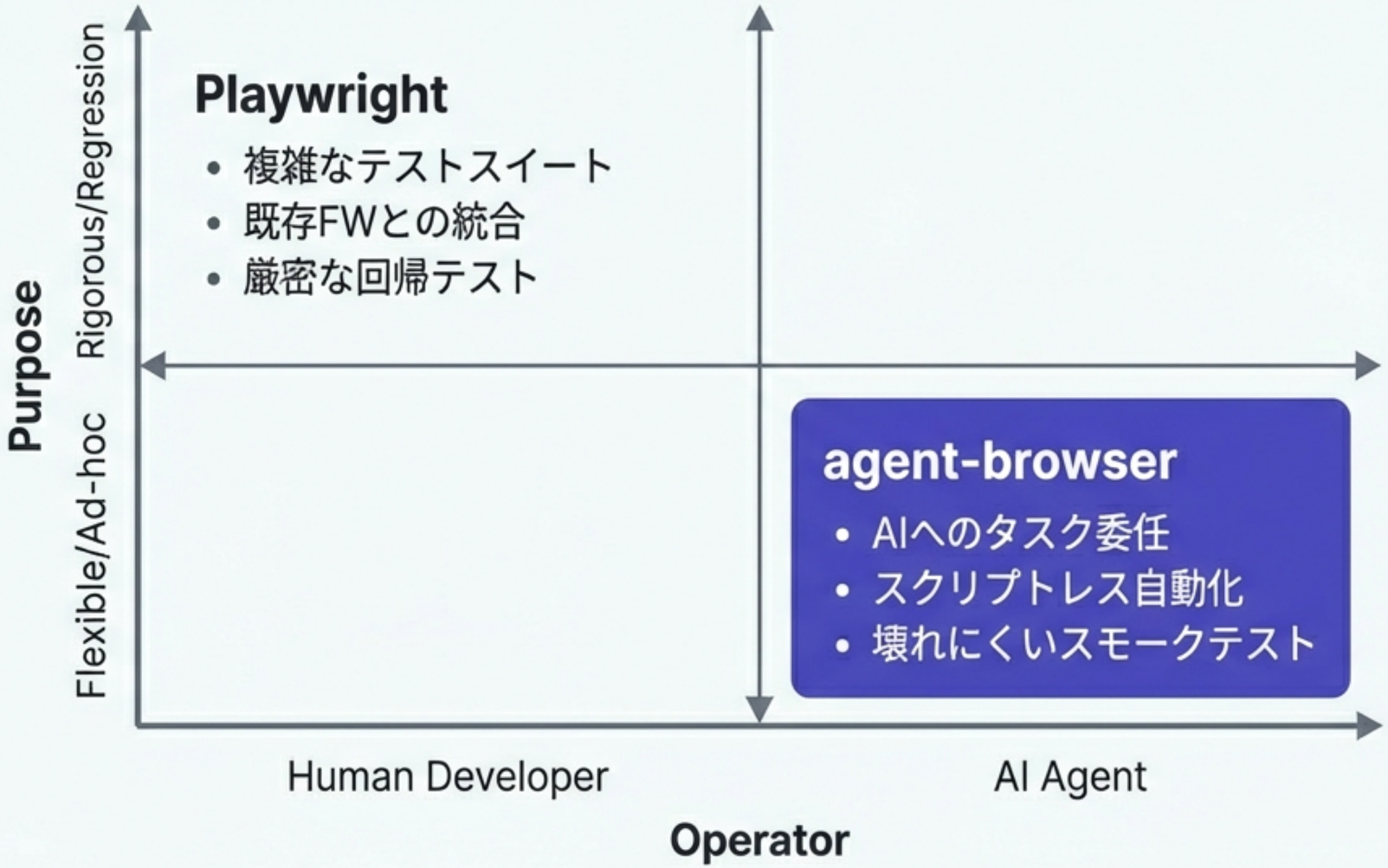
- Target: 主要導線の動作確認（ログイン、フォーム送信など）。
- Approach: 複雑なセレクタ管理を廃止。
- Advantage: UI変更でセレクタが変わっても、AIが柔軟に対応するため、テストが「壊れにくい」。

ユースケース２：自律型AIエージェントへの組み込み



- **Concept**
  - AIコーディングアシスタントに自然言語で指示を出し、ブラウザ操作を丸投げ。
- **Tasks**
  - スクレイピング、競合調査、定期巡回。
- **Why**
  - 手順を定義するのではなく、AIがその場でページを見て判断・実行。

Playwright vs agent-browser：どちらを選ぶべきか？



今すぐ始める（Quick Start）

```
# 1. インストール
npm install -g agent-browser

# 2. 実行（インタラクティブモード）
agent-browser open https://example.com
agent-browser snapshot -i

# 3. AIに指示させる場合（例：Claude Code）
# "agent-browserを使ってログインフローを確認して"
```

設定ファイル不要。`npm` 一発で、その場ですぐにブラウザ自動化を開始できます。

ブラウザ自動化の未来：AIのための標準プロトコルへ

`agent-browser` は単なる便利ツールではありません。
AIがWebを「見て、理解し、操作する」ための、新しい標準インターフェースです。

```
> npm install -g agent-browser
```

https://agent-browser.dev　　github.com/vercel-labs/agent-browser