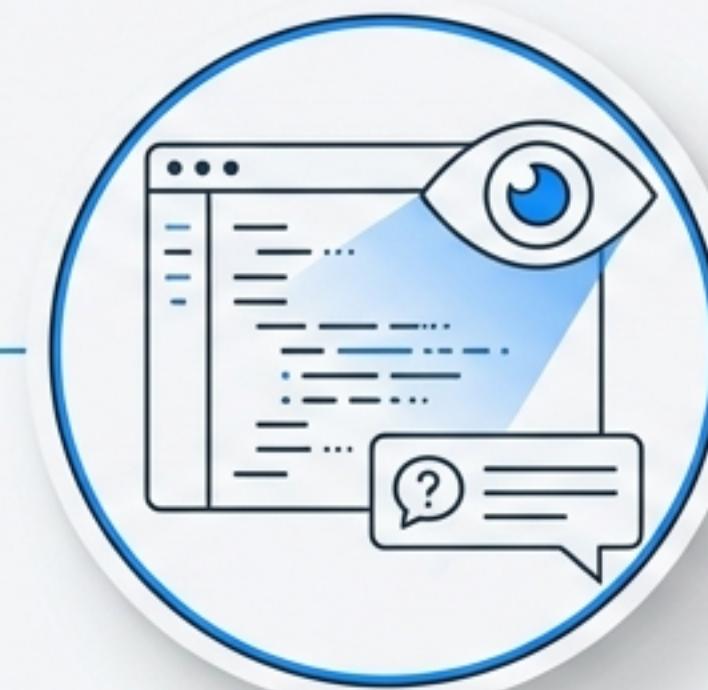


AIネイティブ・プログラミングの解剖学

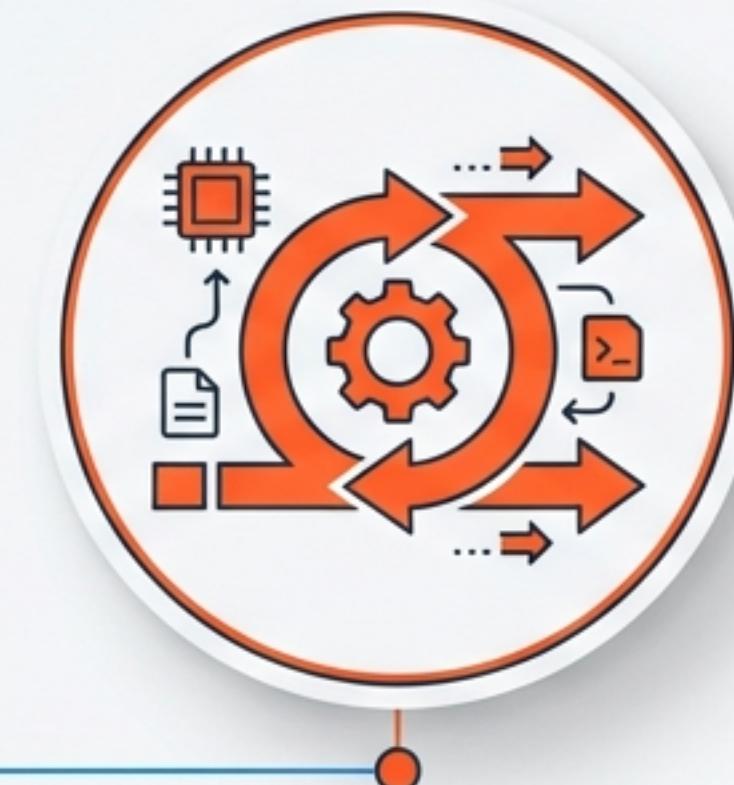
補助プラグインから「OS」レベルの統合、
そして自律型エージェントへ



**Code Completion
(Auto-complete)**



**AI-Integrated IDE
(Context Awareness)**



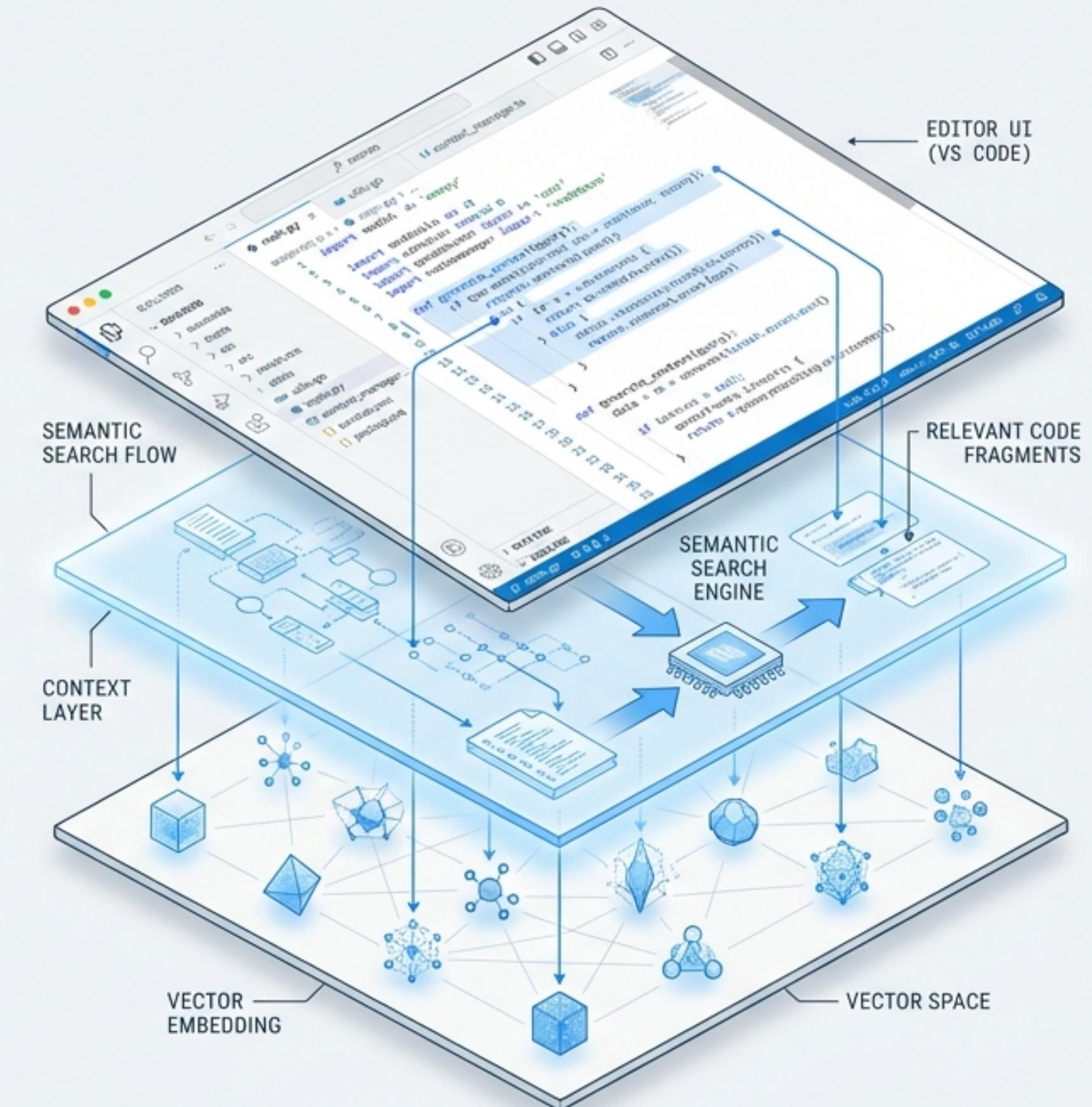
**Autonomous Agents
(Loop Execution)**

現在のAIは単なる『賢いIntelliSense』ではありません。それはオペレーティング層へと進化しました。本プレゼンテーションでは、IDE (Cursor) とエージェント (Claude Code) の対立と融合、そしてその動力を支えるトークンエコノミクスを解剖し、最適なワークフローを構築するための指針を示します。

Cursor : 単なるプラグインではなく、コンテキストを支配する「OS」

多くの開発者が抱く誤解：CursorはVS Codeにチャット画面を埋め込んだだけのものである。

真の強み：コンテキストレイヤー（Context Layer）
Cursorはエディタ全体をラップするOSとして機能します。特に「@codebase」を使用する際、プロジェクト全体を単純に読み込むのではなく、ベクトル埋め込み（Vector Embeddings）を用いた意味検索を行います。これにより、膨大なコードベースから関連性の高い断片だけを抽出し、モデルのメモリを圧迫することなく高精度な回答を生成します。



Tabキーの魔力と「Copilot++」の予測エンジン

Copilot++ (Cursor Tab) は、従来の単語予測とは一線を画します。ユーザーが変数名やロジックを変更した瞬間、その変更が数行先のコードにどう影響するかをリアルタイムで計算し、**未来の差分 (Diff)** を提案します。これは「次の単語」ではなく、「ユーザーの意図」を予測する機能です。

The screenshot shows a code editor window titled "main.js". The code is as follows:

```
7   ...
8 }
9
10 const userContext = loadUser(userId); |
11
12 // Updated logic based on new user context ← GHOST TEXT (SUGGESTION)
13 if (userContext && userContext.hasPermission('admin')) {
14   ...
15   initializeAdminDashboard(userContext.data); ← INTENT PREDICTION (DIFF)
16 }
17
18 }
```

Annotations explain the features:

- USER TYPING**: Points to the line "const userContext = loadUser(userId); |".
- GHOST TEXT (SUGGESTION)**: Points to the code block starting with "if (userContext && userContext.hasPermission('admin')) {".
- INTENT PREDICTION (DIFF)**: Points to the line "initializeAdminDashboard(userContext.data);".
- JetBrains Mono**: Points to the font name at the bottom left.

IDEの限界と「トークン消費」の壁

IDEアプローチの摩擦点：

修正すべきファイルが特定できない時、ユーザーは「@codebase」や全ファイル読み込みに頼りがちです。これは大量のトークンを浪費し、コストを増大させます。

また、既存のVS Code UIフレームワークに縛られているため、複数のファイルをまたぐ大規模なアーキテクチャ再構築のようなタスクでは、モデルが慎重になりすぎ、大胆な変更が難しいという制約があります。

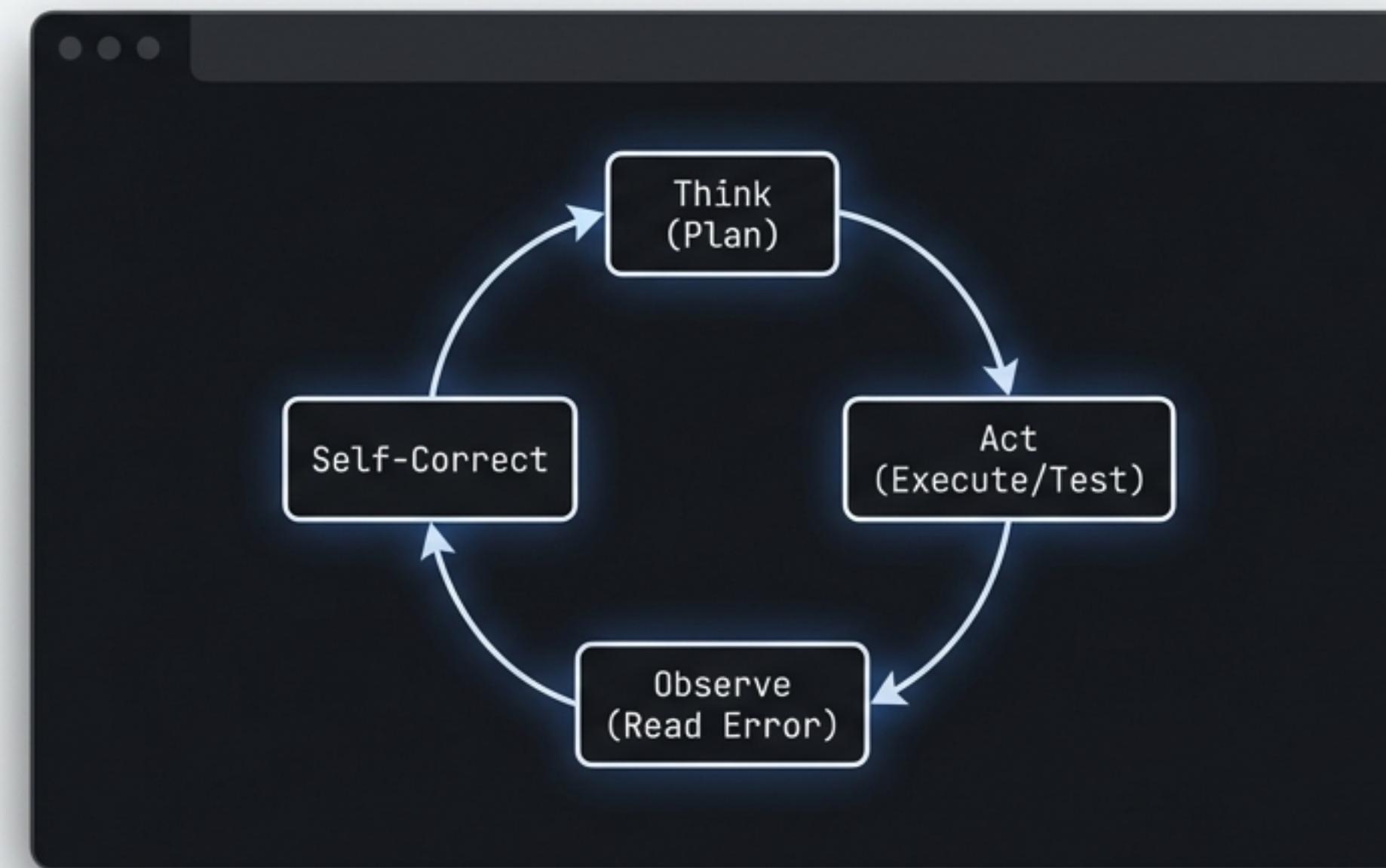


Context Window
/ Token Usage



Claude Code：ターミナルに住む「自律型万能戦士」

CLIファーストの哲学。IDEの「チャット→回答」という直線的な対話とは異なり、エージェントは自律的なループ構造を持ちます。自ら計画し（Think）、コマンドを実行し（Act）、エラーが出ればログを読み（Observe）、自己修正を行います。このループ構造こそが、複雑なタスクを完遂できる理由です。



比較：Cursor (IDE) vs. Claude Code (Agent)

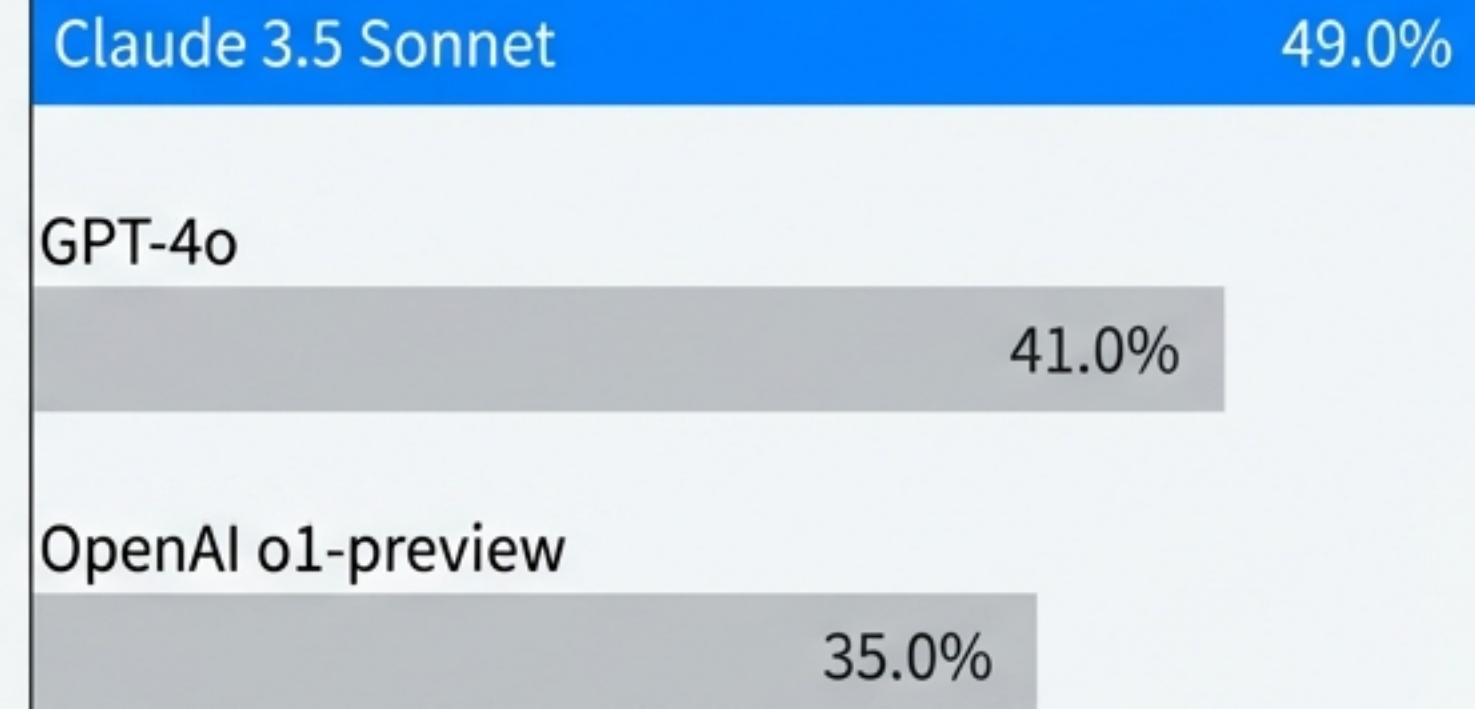
項目	Cursor (IDE Mode)	Claude Code (Agent Mode)
実行権限 (Permissions)	保守的 (コード編集中心)	Shell権限あり (テスト実行・ログ確認可)
意思決定 (Decision Making)	補完と対話 (Completion & Dialogue)	ループ構造 (Think → Act → Observe)
取り消し (Rollback)	GitやCmd+Zに依存	ネイティブなUndo機能 (ファイル操作の完全制御)
推奨ユースケース	日々のコーディング、ドラフト作成	リファクタリング、バグ修正、環境構築

エンジンとしての「Claude 3.5 Sonnet」の優位性

これらのツールが共通して採用するエンジン、
Claude 3.5 Sonnet。
なぜこのモデルが選ばれるのか？

1. 圧倒的な問題解決能力: SWE-bench Verifiedにおいて49.0%を記録し、競合を凌駕。
2. Vision Benchmarks: スクリーンショットの理解力において、パディングや色の微細な違いを捉える「目」の良さ。
3. Artifactsへの最適化: モダンなUIコード(React/Tailwind)を一発で出力するように調整されています。

SWE-bench Verified



なぜClaudeのコードは「美しく」感じるのか

「審美眼」と「スピード」の融合。

GPT-4oなどが時に見せる「怠慢（Lazy Code）」に対し、Claude 3.5 Sonnetはプロダクションレベルのフロントエンドコードを完結させる能力に長けています。

さらにClaude Codeでは、「書いて、実行して、自動修正する」というサイクル速度が極めて速いため、開発者はデザインのプロセスに「キレ」を感じるのである。

Other Models

Code Editor

```
sparse_component.js
// sparse_component.js

function BasicComponent() {
  return (
    <div>
      <h1>Welcome</h1>
      <p>Content goes here.</p>
      {/* ...rest of code */}
    </div>
  );
}
```

Welcome

Content goes here.

Claude 3.5 Sonnet

Code Editor

```
// polished_component.jsx
import { Card } from '@/components/ui';

export default function ModernCard() {
  return (
    <Card className="bg-slate-900 text-white p-4 rounded-lg shadow-md w-[auto] w-[auto]">
      <h2 className="text-xl font-bold mb-2">Elegant Design</h2>
      <p className="text-slate-300 text-sm mb-4">Complete with modern Tailwind styling.</p>
      <button className="bg-blue-500 hover:bg-blue-600 text-white py-2 px-4 rounded transition-colors">
        Explore More
      </button>
    </Card>
  );
}
```

Elegant Design

Complete with modern Tailwind styling.

Explore More

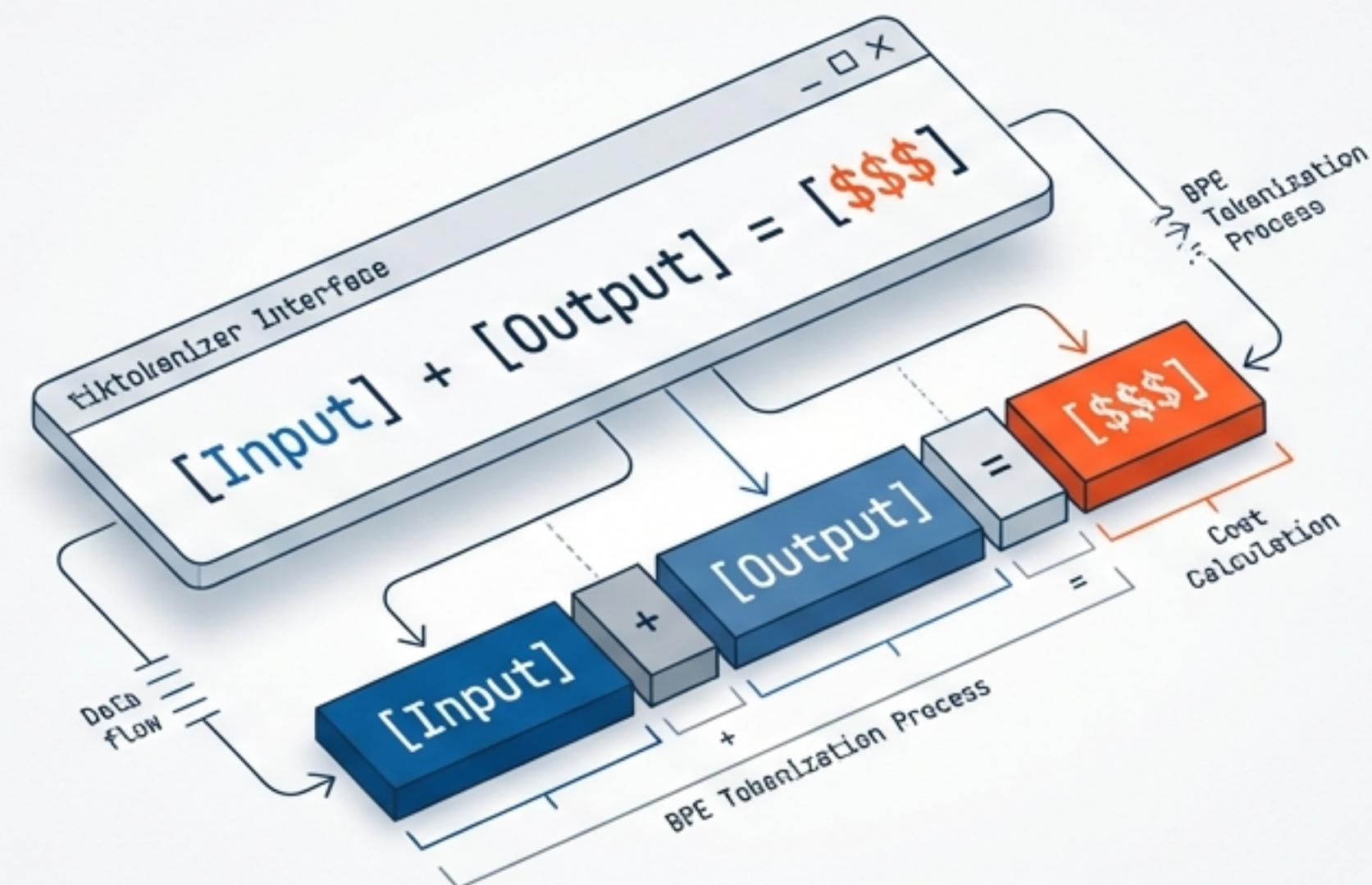
bg-slate-900 rounded-lg shadow-md
active state drop-nadog

トークン (Token) : 脳細胞を動かす「エネルギー」

AIネイティブ開発において、トークンは「燃料」であり「通貨」です。

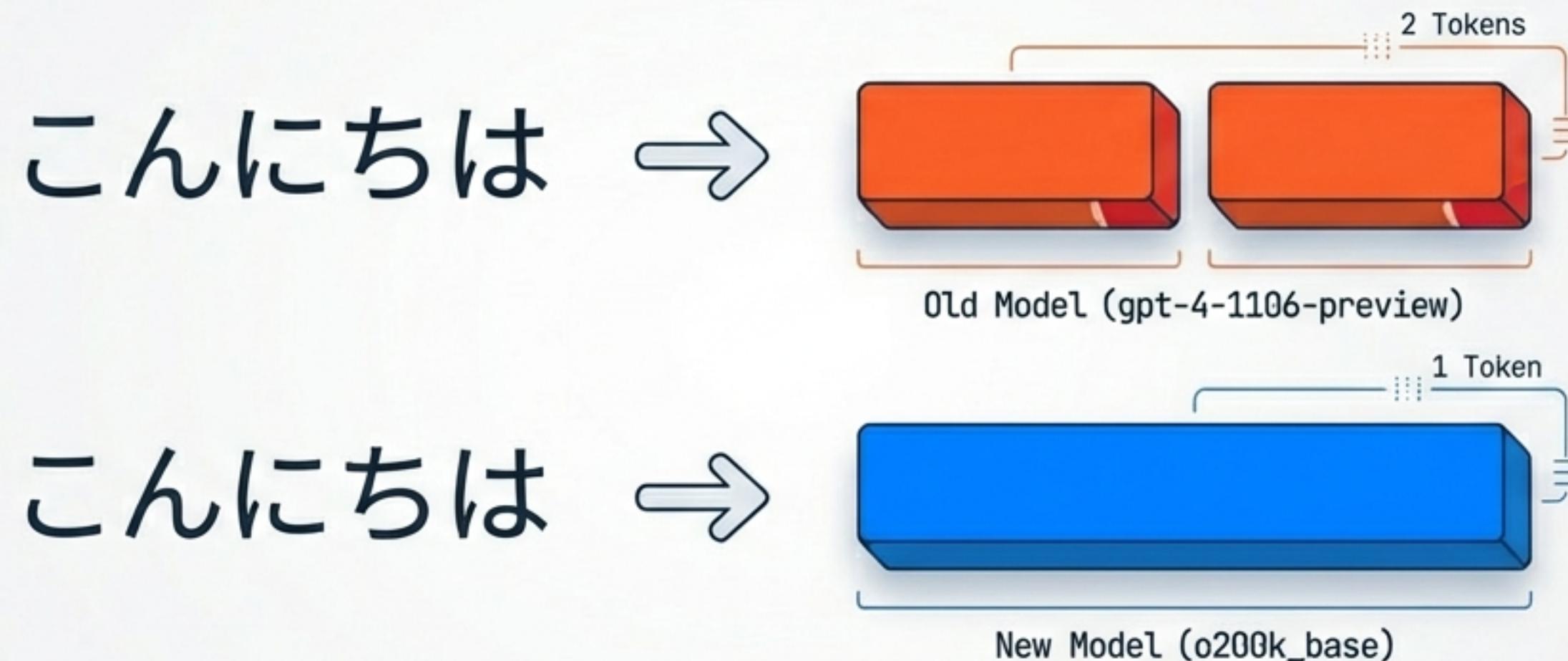
モデルは文字そのものではなく、BPE (Byte Pair Encoding) によって分割されたトークンを読み込みます。

商用AIの課金モデルは「入力 (Prompt) + 出力 (Completion)」の総量で決まるため、自律型エージェントを使う上で、このメカニズムの理解はコスト管理の核心となります。



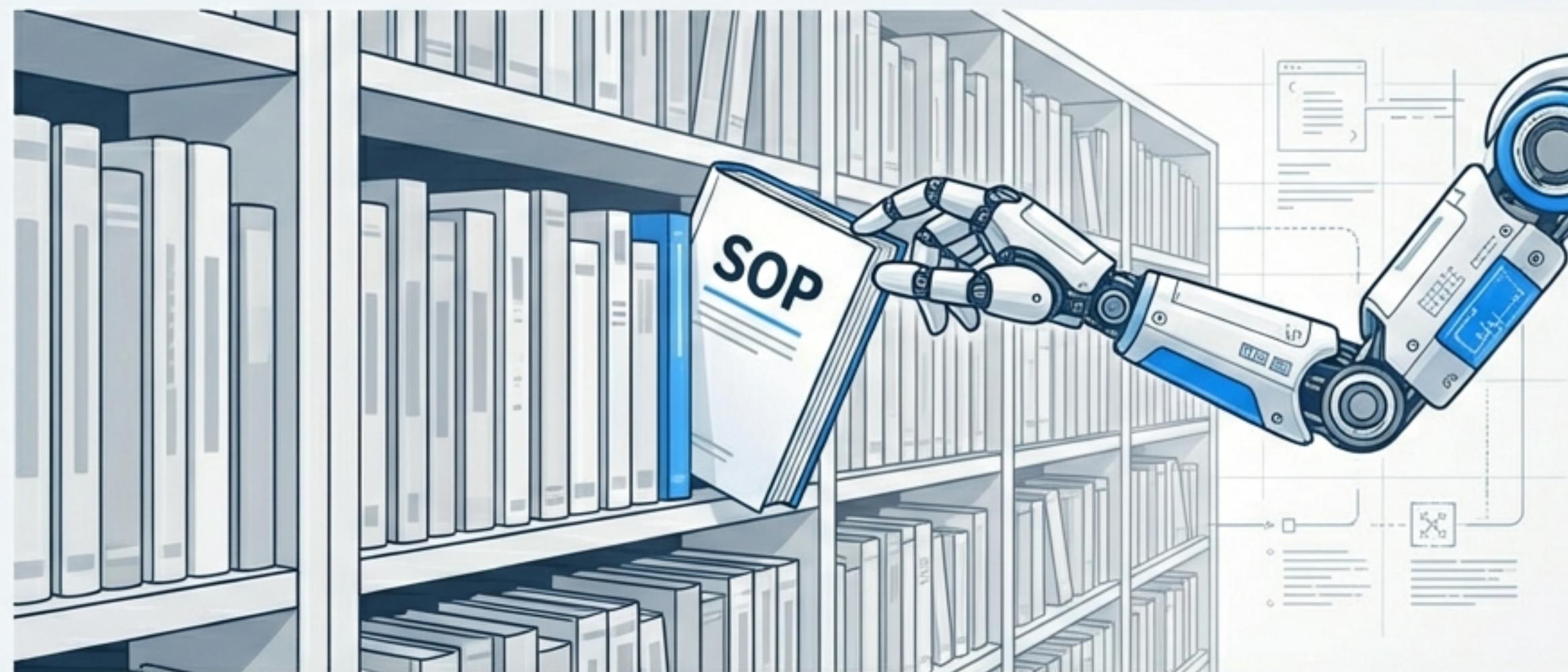
日本語ユーザーへの朗報：言語の不平等の解消

かつて、日本語や中国語は英語に比べて2~3倍のトークンを消費していました（言語の不平等）。しかし、GPT-4oや最新のモデルで採用されたトークナイザー「o200k_base」は、東アジア言語に対して大幅に最適化されています。これにより、日本語での長いコンテキスト処理が、より高速に、そして安価に行えるようになりました。

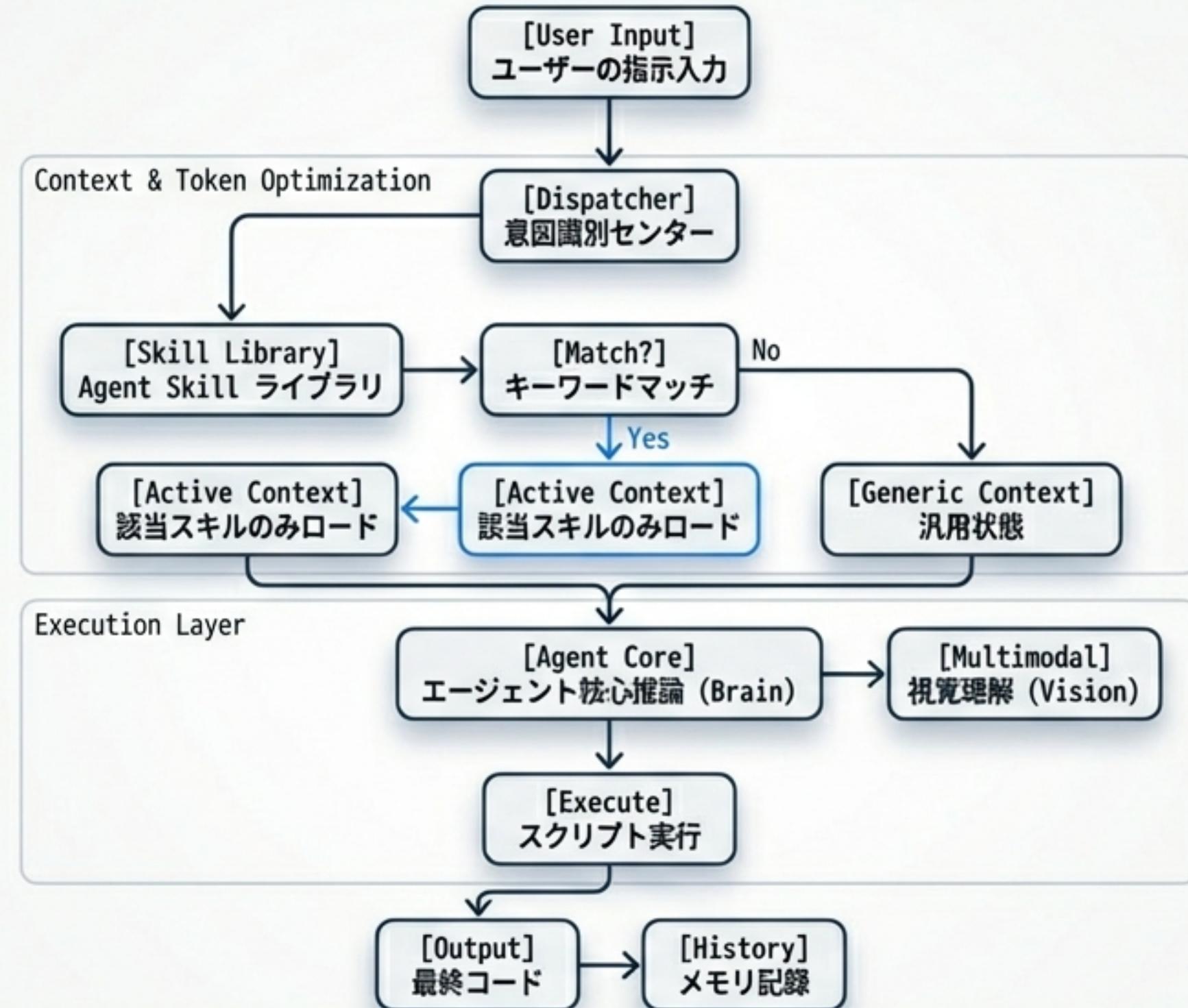


Agent Skill : AIに渡す「標準作業手順書（SOP）」

Agent Skill (MCP - Model Context Protocol) は、エージェントのためのガイドブックです。 「プロジェクト全体を読め」と指示する代わりに、「UIを修正するなら src/componentsだけを見ろ」「authフォルダは触るな」といった具体的な指示書（スキル）を渡します。これにより、探索を「全量検索」から「オンデマンド呼び出し」に変え、トークンを節約しつつ精度を劇的に向上させます。



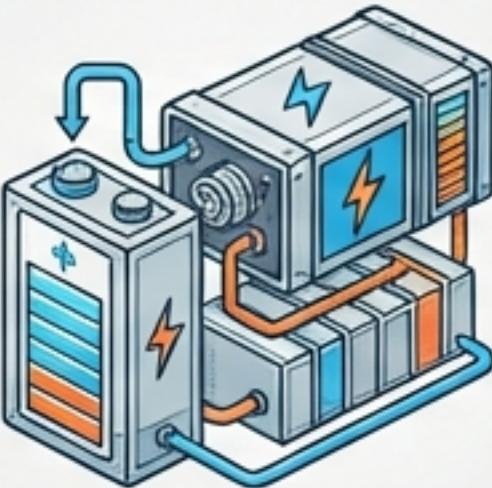
データフロー：意図から実行への最適化パス



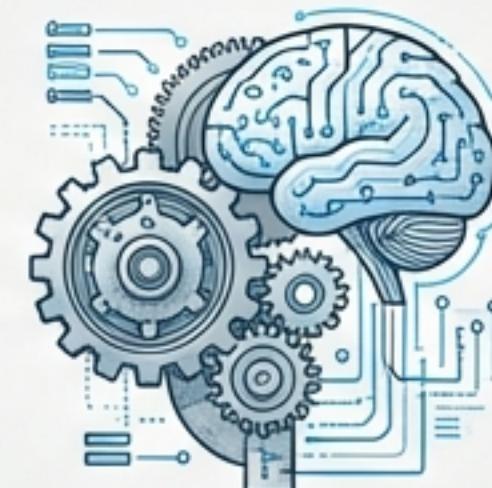
エコシステムの全体像：役割と関係性

これら4つの要素が噛み合うことで、AIネイティブ・プログラミングは成立します。エネルギー（トークン）を管理し、適切なマニュアル（スキル）を職人（エージェント）に渡すことが、アーキテクトであるあなたの役割です。

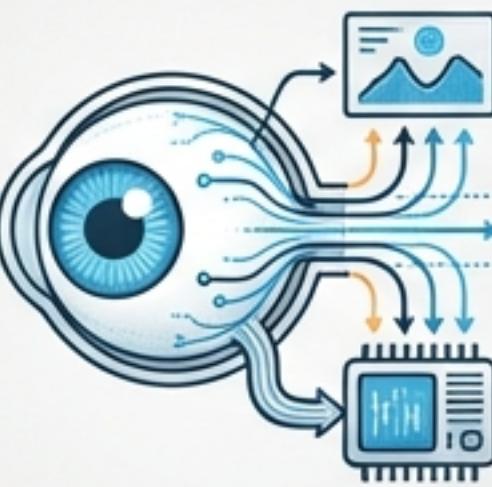
Token
エネルギー (Fuel)



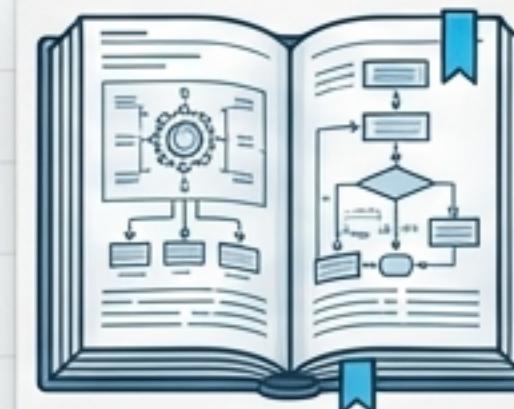
Agent
職人 (Artisan)



Multimodal Agent
目を持つ職人
(Artisan with Eyes)



Agent Skill
専門技能書 (SOP)



次の一步：あなたのプロジェクトを最適化する

今すぐできるアクション：プロジェクトルートに `.cursorrules` を配置する。これはエージェントに対する「行動規範」です。コーディング規約、禁止ディレクトリ、アーキテクチャの好みを明示することで、エージェントの迷いを無くし、トークン消費を抑え、手戻りを防ぎます。



```
→ .cursorrules
{
  "rules": [
    "Always use Tailwind CSS for styling.",
    "Do NOT modify files in /legacy directory.",
    "Prefer functional components over class components."
  ]
}
```



AIネイティブな開発者へ

私たちは、単なるコード補完の時代を終え、思考と行動のループを管理する時代に入りました。
適切なインターフェース（Cursor / Claude Code）を選び、燃料（Token）を最適化し、
手順書（Skill）を整備する。あなたはもう、ただのコーダーではありません。
インテリジェント・システムの設計者（Architect）なのです。