計算機・LLMとの付き合い方

2026年4月16日（木）
スタートアップゼミ#03
博士1年　古橋 郁一

@田老, 2024年

目次

第一部: 計算機を使おう！

- 計算機の仕組み
- Pythonの仕組み
- プログラミング・パラダイム

第二部: LLMを使おう！

- 今日のLLMのトレンド
- ケーススタディ: 勉強時, 計算時, 発表時

前段
前回内容の確認

事前課題

事前準備

- uvのインストール
- Pythonの実行（Hello world）

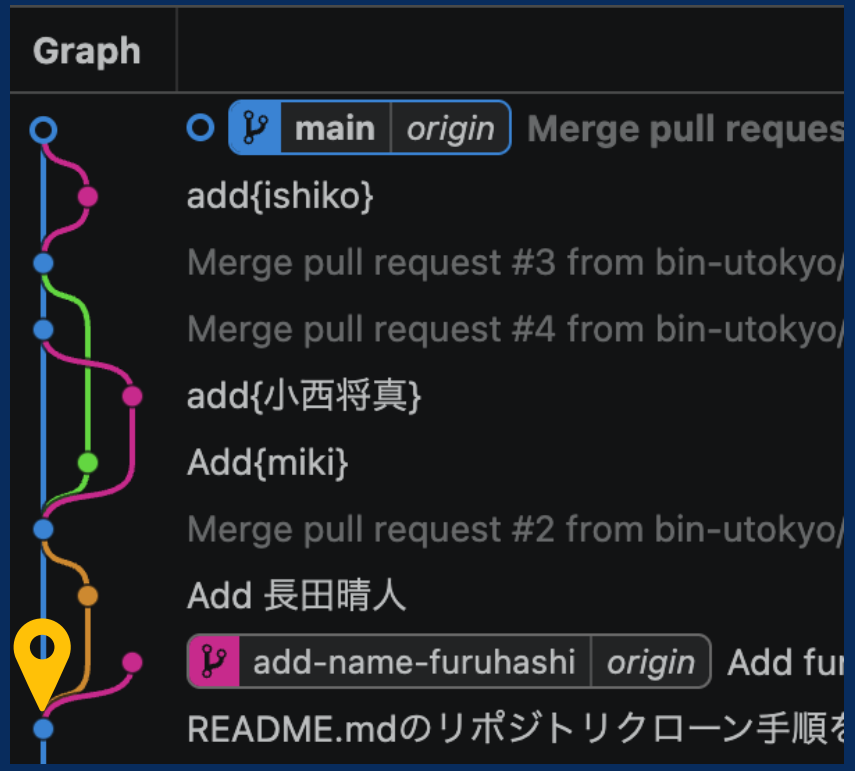
6. 自分の名前を追加して Pull Request を出す

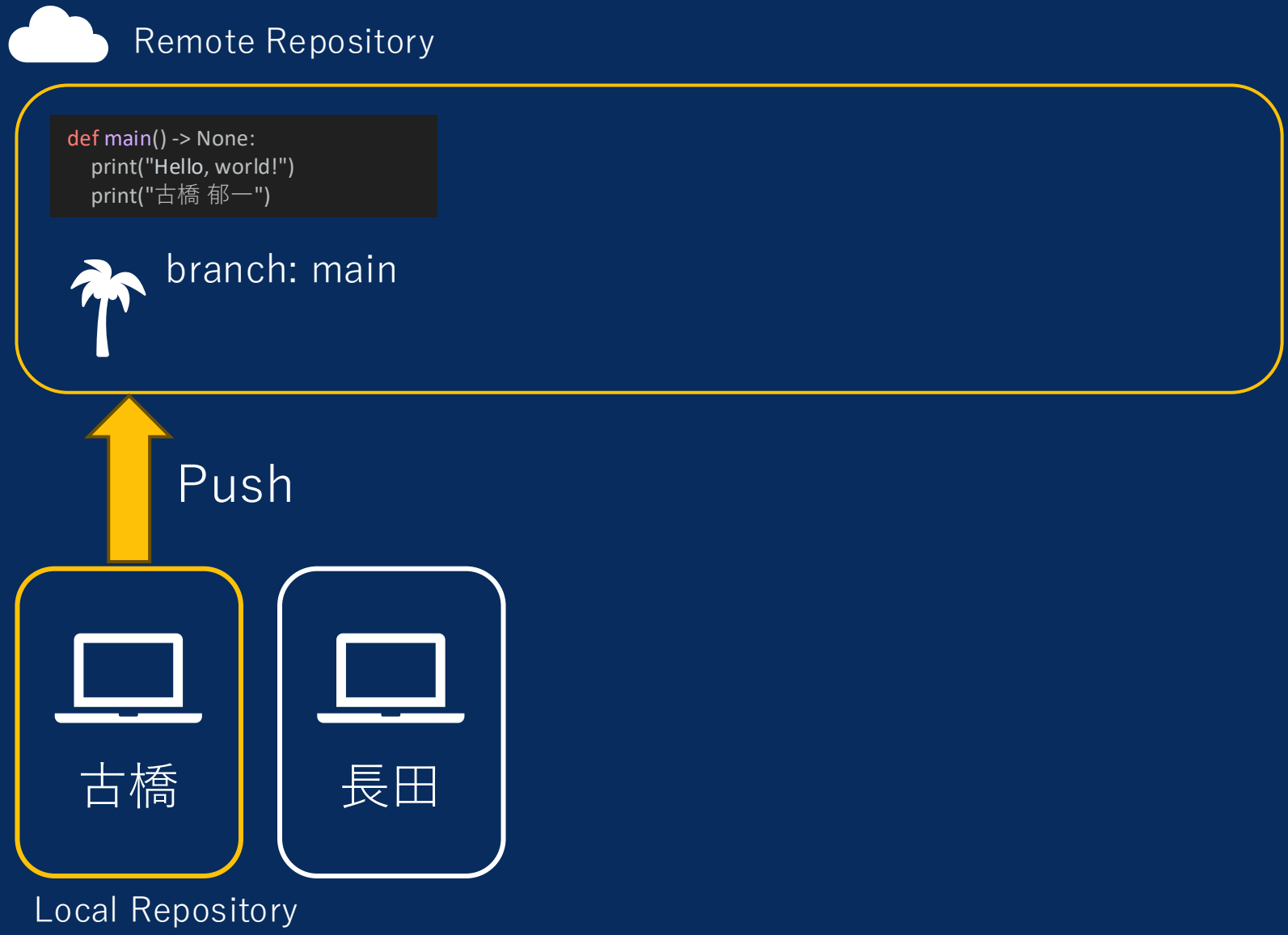
src/01_hello/main.py のコメントに沿って、以下を行ってください。

1. ブランチを作成する（ブランチ名: `add-name-{自分の名前}`）
2. `main.py` に自分の名前を追記する
3. `src/01_hello/main.py` のみをステージングしてコミットする（コミットメッセージ: `Add {自分の名前}`）
4. ブランチをリモートリポジトリにプッシュする
5. Pull Request を作成する（タイトル: `Add {自分の名前}`、内容: `{自分の名前}を追加しました。`）

Gitの復習

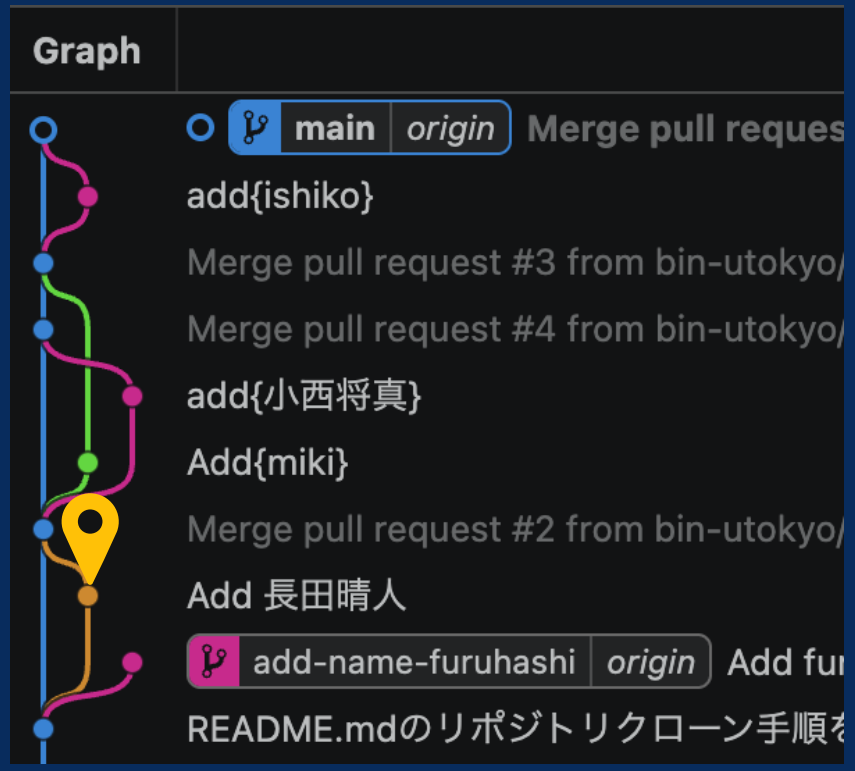
Git Graph





Gitの復習

Git Graph



Remote Repository

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
```

branch: main

Pull

古橋

長田

Local Repository

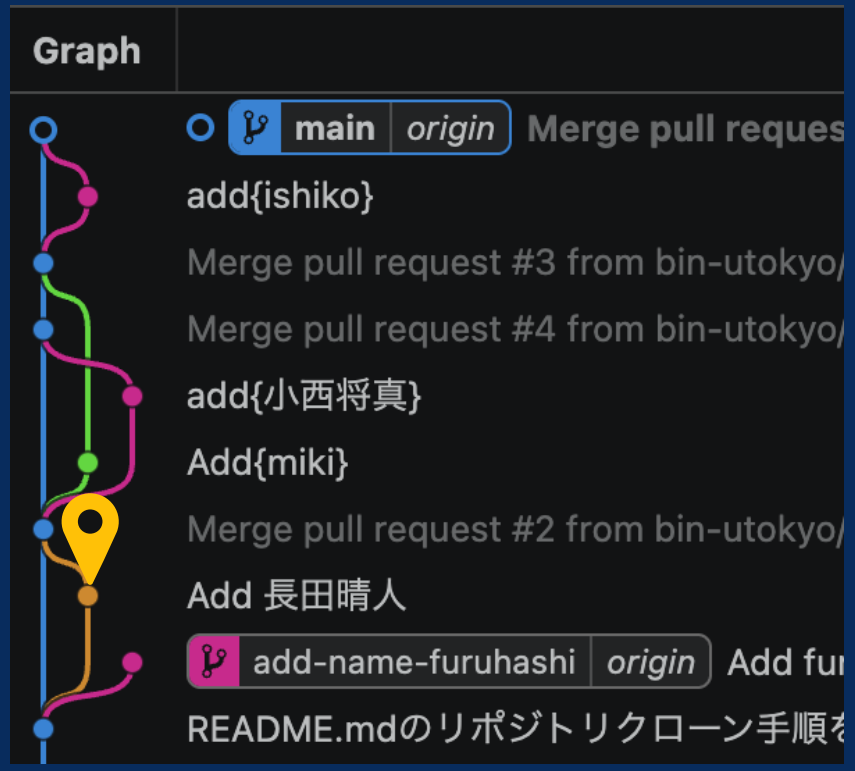
```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print(“長田 晴人")
```

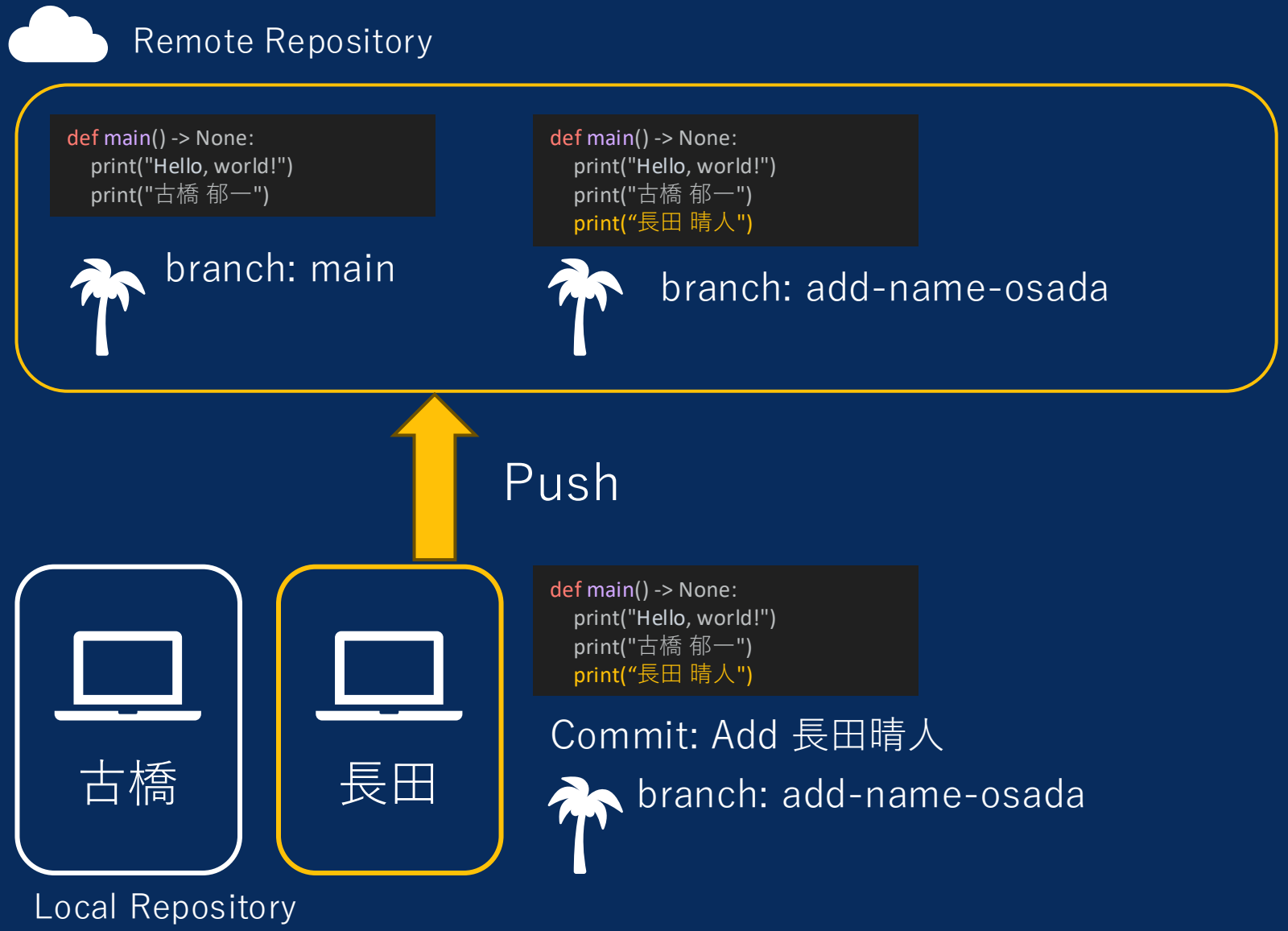
Commit: Add 長田晴人

local branch: add-name-osada

Gitの復習

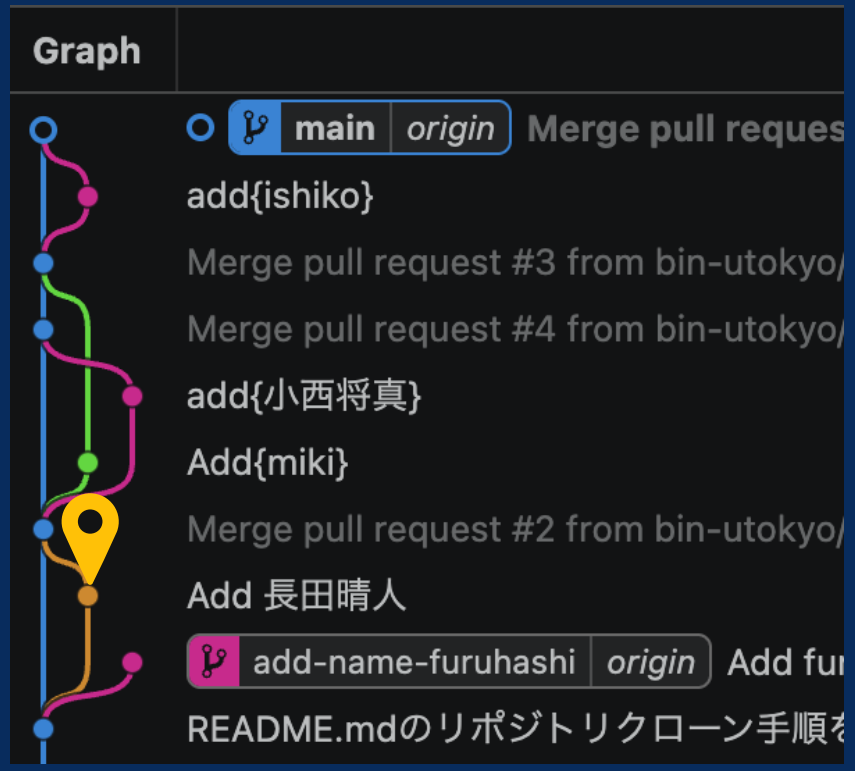
Git Graph





Gitの復習

Git Graph



Remote Repository

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
```

branch: main

PR

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print("長田 晴人")
```

branch: add-name-osada

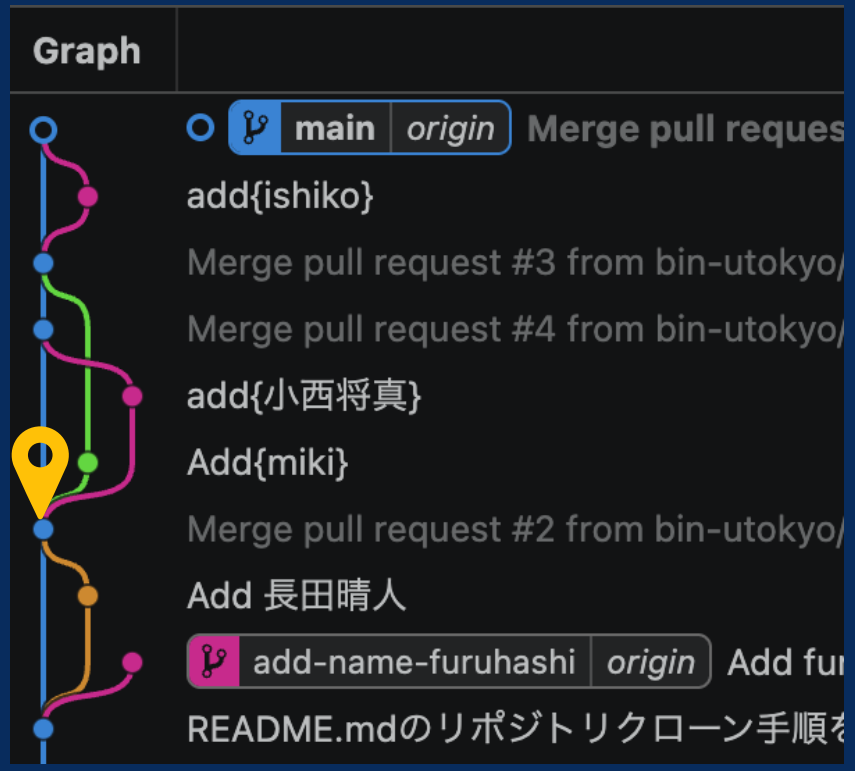
古橋

長田

Local Repository

Gitの復習

Git Graph



Remote Repository

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
```

branch: main

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print("長田 晴人")
```

branch: add-name-osada

Merge

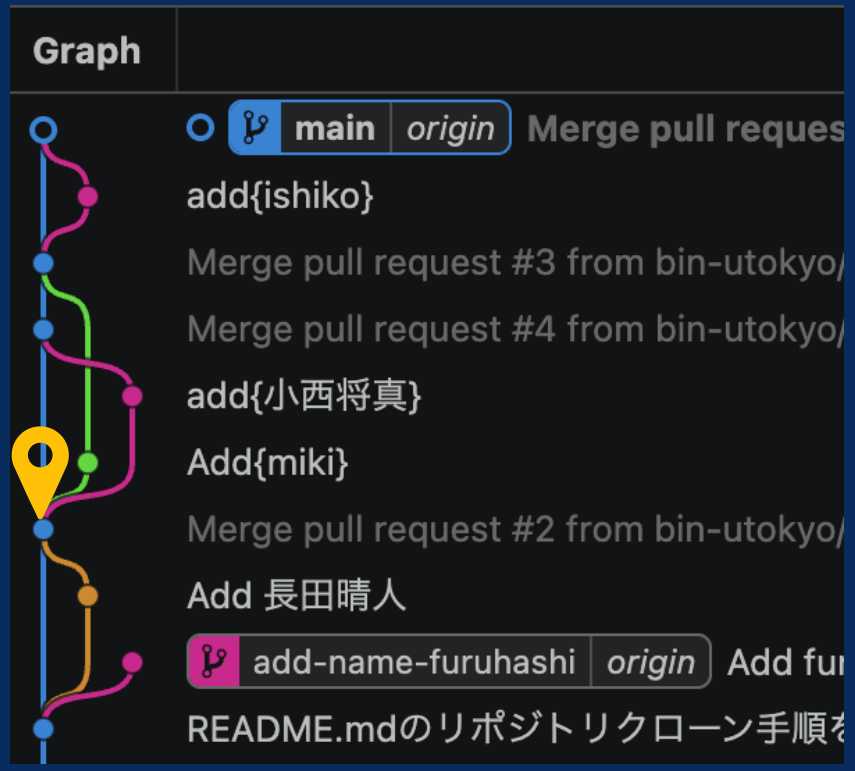
古橋

長田

Local Repository

Gitの復習

Git Graph



Remote Repository

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print(“長田 晴人")
```

branch: main

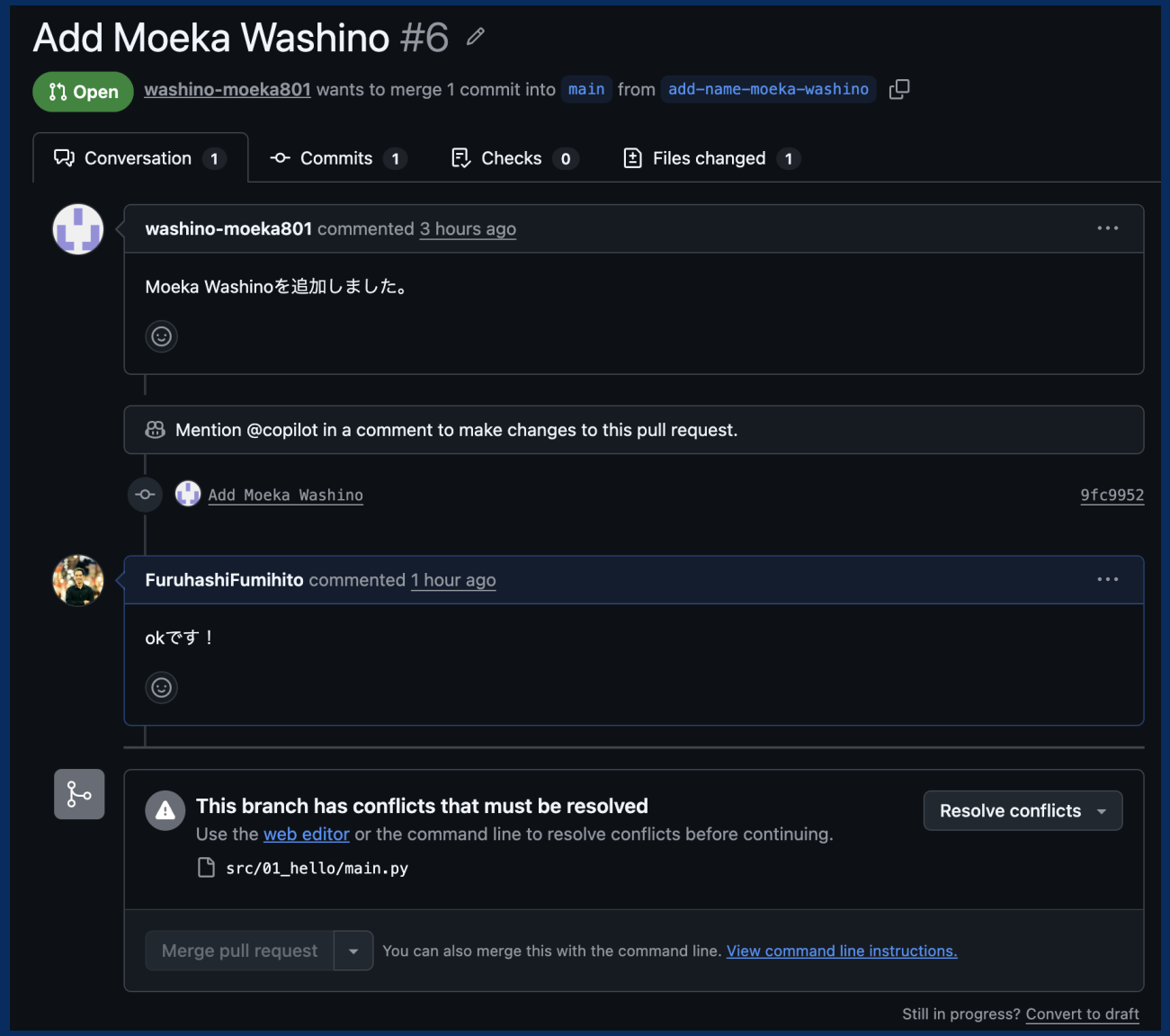
Merge

古橋

長田

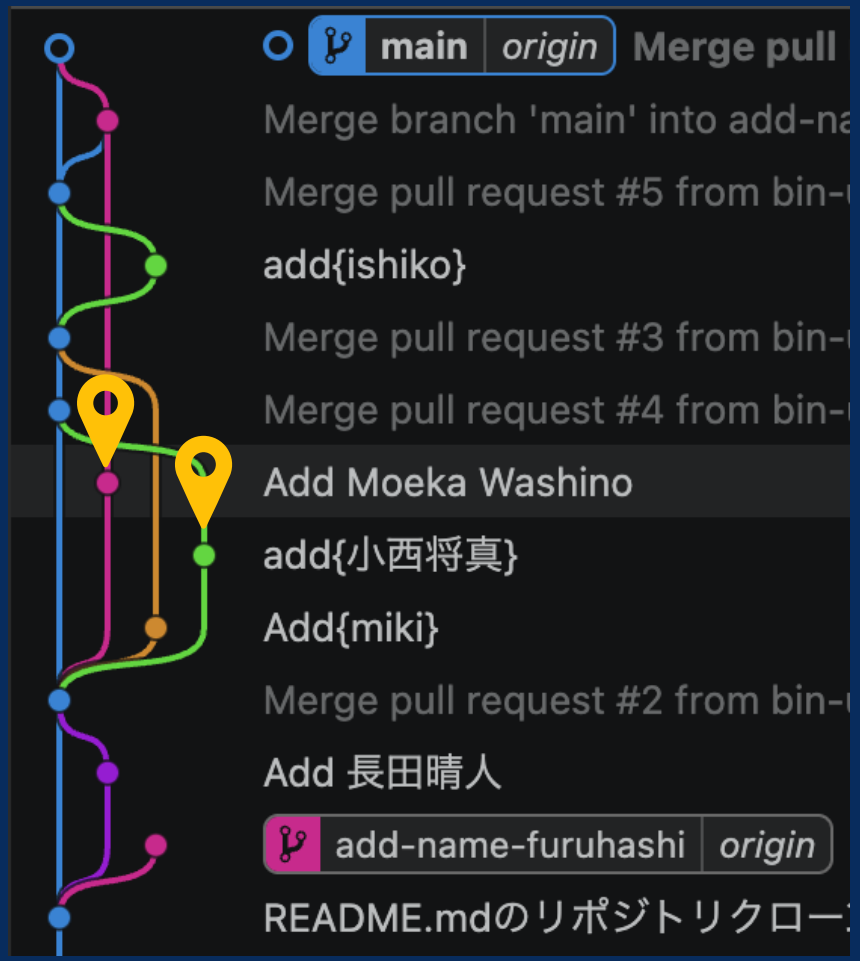
Local Repository

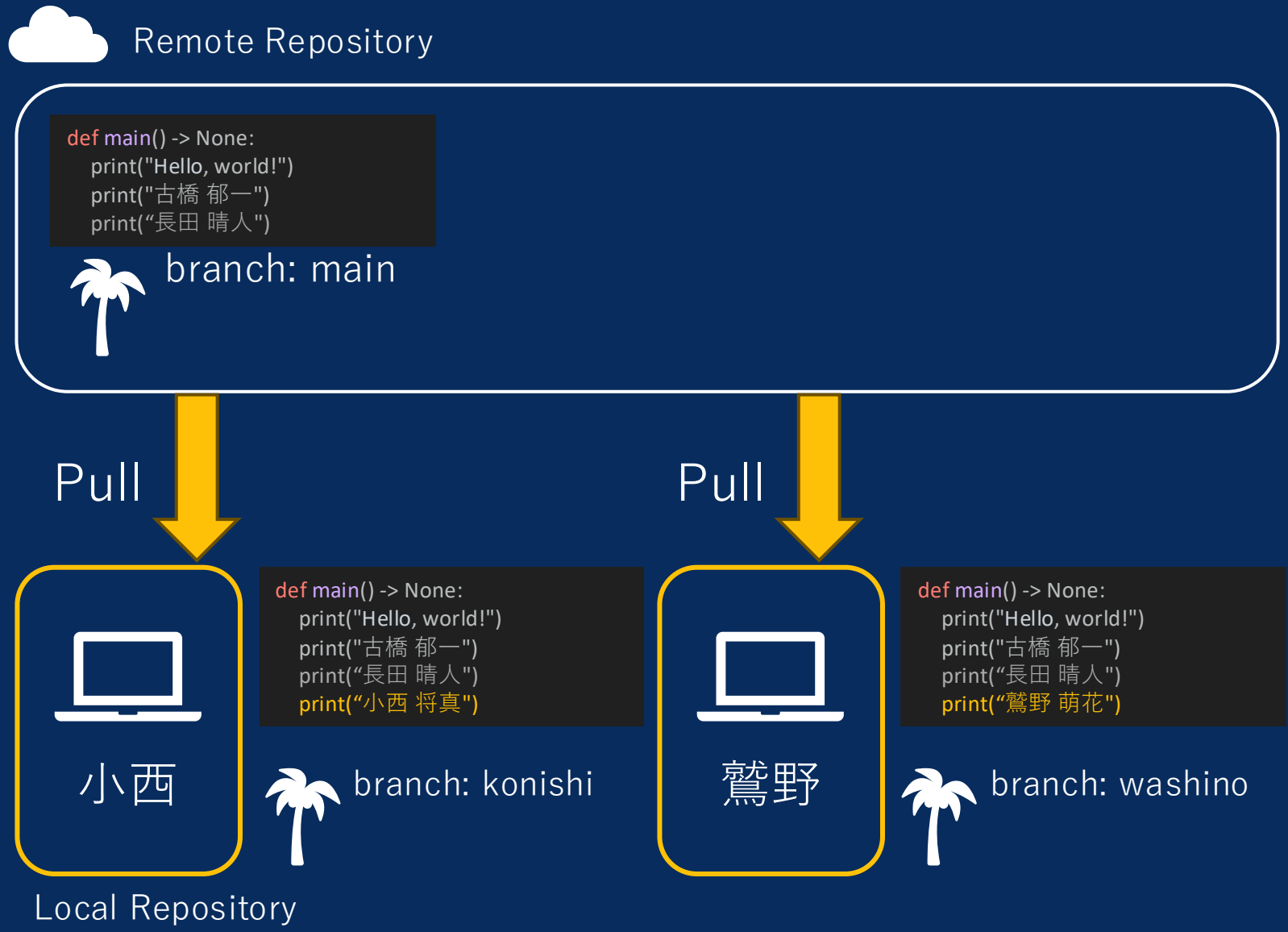
うまくいかないこともある



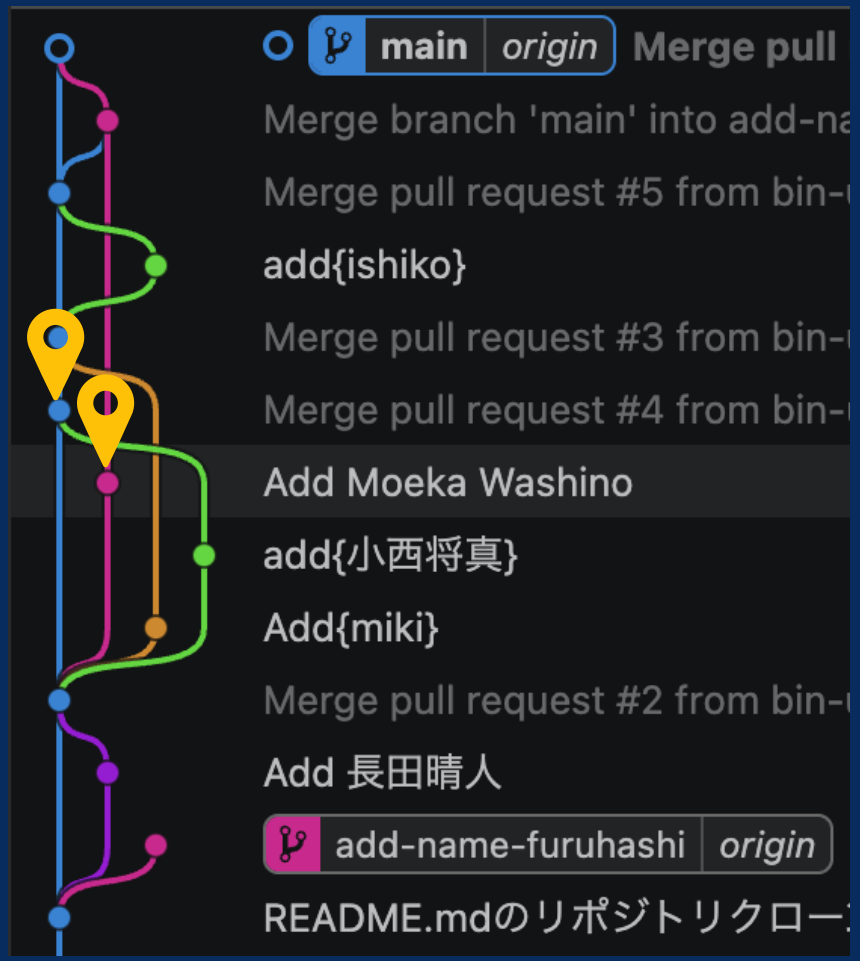
エラーっぽい...

うまくいかないこともある





うまくいかないこともある



Remote Repository

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print("長田 晴人")
    print("小西 将真")
```

branch: main

Push & Merge

小西

```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print("長田 晴人")
    print("小西 将真")
```

branch: konishi

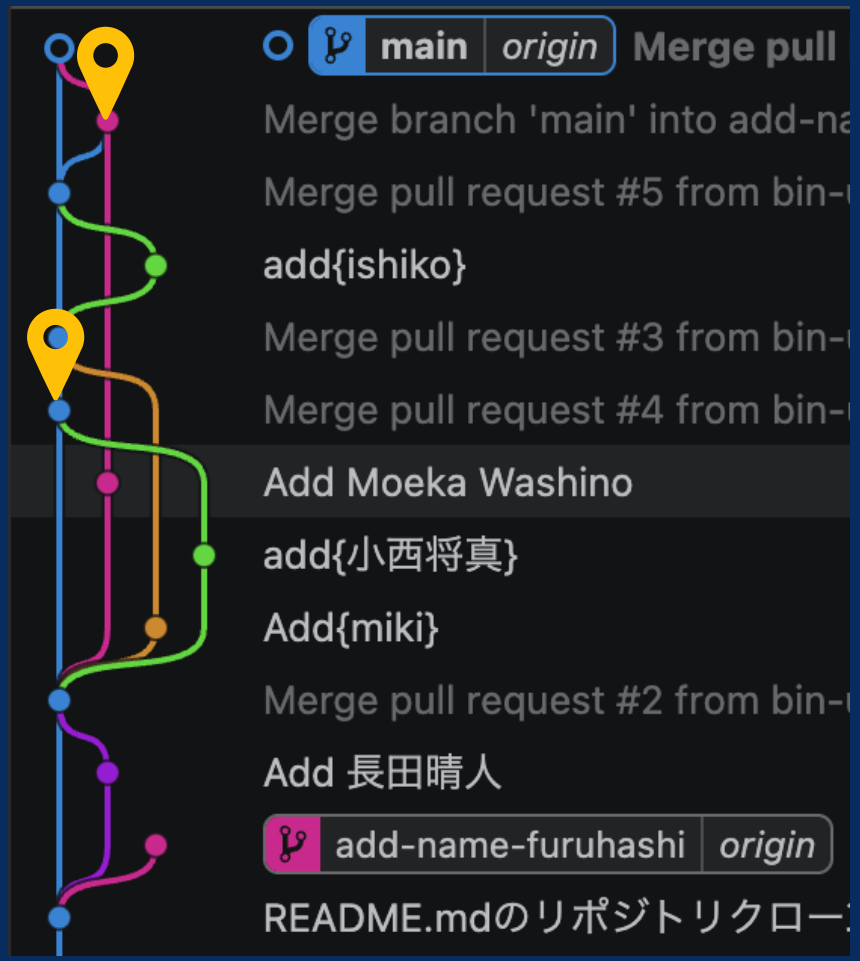
鷲野

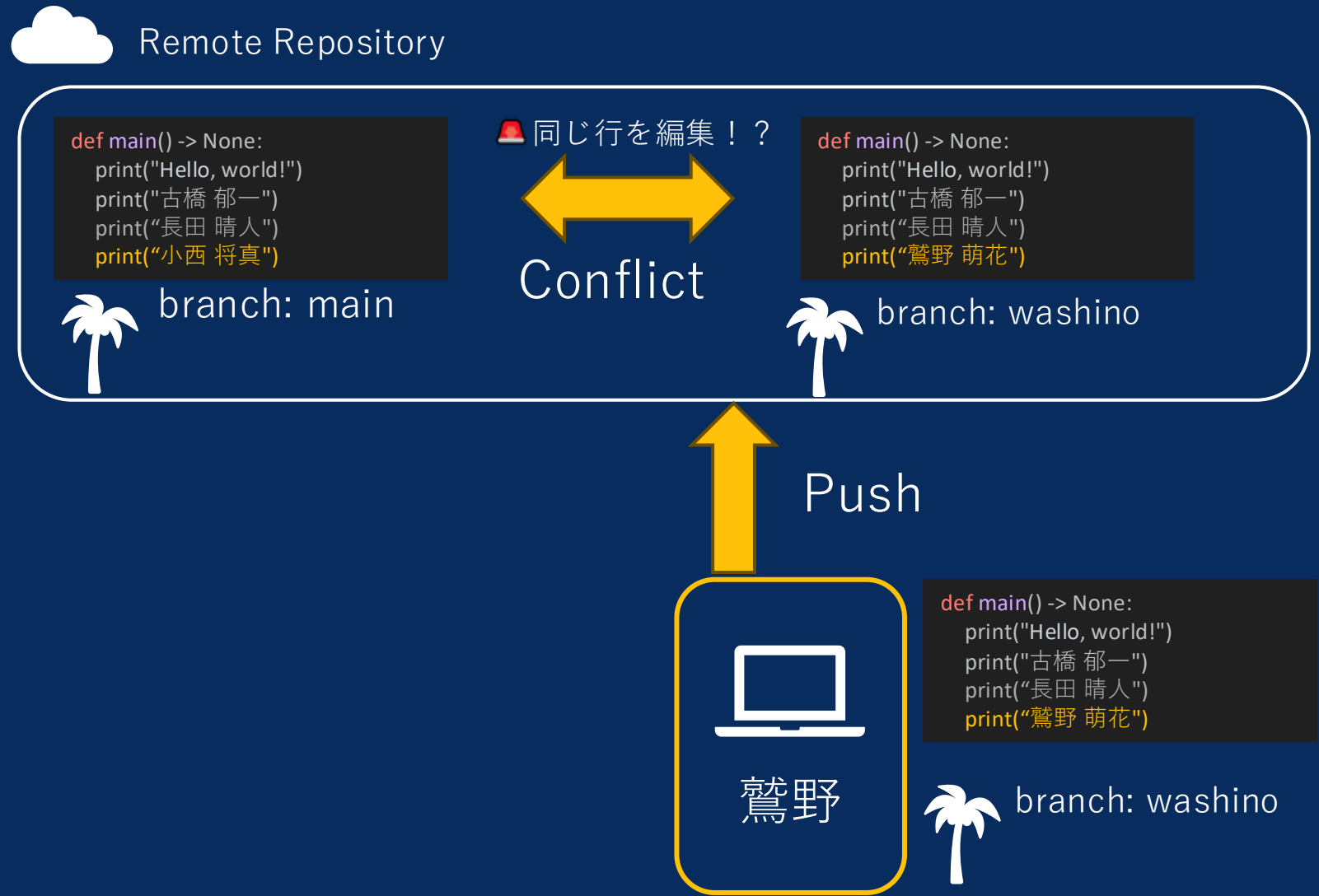
```
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    print("長田 晴人")
    print("鷲野 萌花")
```

branch: washino

Local Repository

うまくいかないこともある





完成物

疑問①: どういう原理でprint(“”)が出力されている？

python-training / src / 01_hello / main.py

hiko}

3 lines (18 loc) · 729 Bytes

Raw

```python
def main() -> None:
    print("Hello, world!")
    print("古橋 郁一")
    #
    # ここに名前を追加していく
    print("三木隆斗")
    #
    print("長田 晴人")
    print("小西将真")
    print("石河万衣")



    # Branchを作成: branch名は add-name-{自分の名前}
    # 名前を追記
    # ステージング: src/01_hello/main.py のみステージングする
    # コミット: コミットメッセージは Add {自分の名前}
    # プッシュ: ブランチをリモートリポジトリにプッシュする
    # Pull Requestを作成: タイトルは Add {自分の名前}、内容は「{自分の名前}を追加しました。」


if __name__ == "__main__":
    main()
```

疑問②: これ何？

```
(python-training) furuhashikaoruichi@ python-training % uv run python src/01_hello/main.py
Hello, world!
古橋 郁一
三木隆斗
長田 晴人
小西将真
(python-training) furuhashikaoruichi@ python-training %
```

細かい性格なので...

python-training / src / 01_hello / main.py

ishiko528 add{ishiko}

Code Blame 23 lines (18 loc) · 729 Bytes

```
1  def main() -> None:
2      print("Hello, world!")
3      print("古橋 郁一")
4      #
5      # ここに名前を追加していく
6      print("三木隆斗")
7      #
8      print("長田 晴人")
9      print("小西将真")
10     print("石河万衣")
```

気になる点①: 行間に#をつける？つけない？

気になる点②: “長田 晴人”？ or “長田晴人”？

- コードは「他人と自分にわかりやすく」
  - ＝ LLMにわかりやすく
- より大人数・長大なコーディングでは、適切なルールメイクが重要

疑問③: より良いコードの書き方とは？

プログラミング・パラダイム

第一部で学ぶ内容

疑問

- プログラムは、どうやって名前を出力している？
- uvとは何者？？なぜ必要？？
- より良い書き方はない？

学習内容

- 計算機の仕組み
- 仮想環境
- プログラミング・パラダイム（オブジェクト指向/テスト駆動開発）

第一部:
計算機を使おう！

今日の計算技術とは

LLM以前の内容:

- for文, if文, class, ...
- 今やLLMに指示するだけ

では何が重要か？

- LLMに適切な指示を出せるか？
- LLMの出力を理解できるか？
- 次の指示を出せるか？

今日学ぶ内容

- 計算機の仕組み
- Pythonの仕組み
- 現代の「良い」プログラミング規範

プログラムはなぜ動くのか？

python-training / src / 01_hello / main.py

ishiko528 add(ishiko)

Code Blame 23 Lines (18 loc) · 728 Bytes Raw

```python
def main() -> None:
    print("Hello, world!")
    print("永瀬 修一")
    #
    # ここに名前を追加していく
    print("三木陽斗")
    #
    print("長田 暖人")
    print("小西将真")
    print("石河万波")



    # Branchを作成: branch名は add-name-(自分の名前)
    # 名前を追記
    # ステージング: src/01_hello/main.py のみステージングする
    # コミット: コミットメッセージは Add (自分の名前)
    # プッシュ: ブランチをリモートリポジトリにプッシュする
    # Pull Requestを作成: タイトルは Add (自分の名前)、内容は「(自分の名前)を追加しました。」


if __name__ == "__main__":
    main()
```

計算機の構成

入力装置

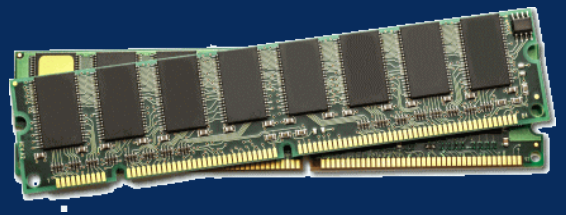
中央演算装置

Python

演算装置

出力装置

記憶装置



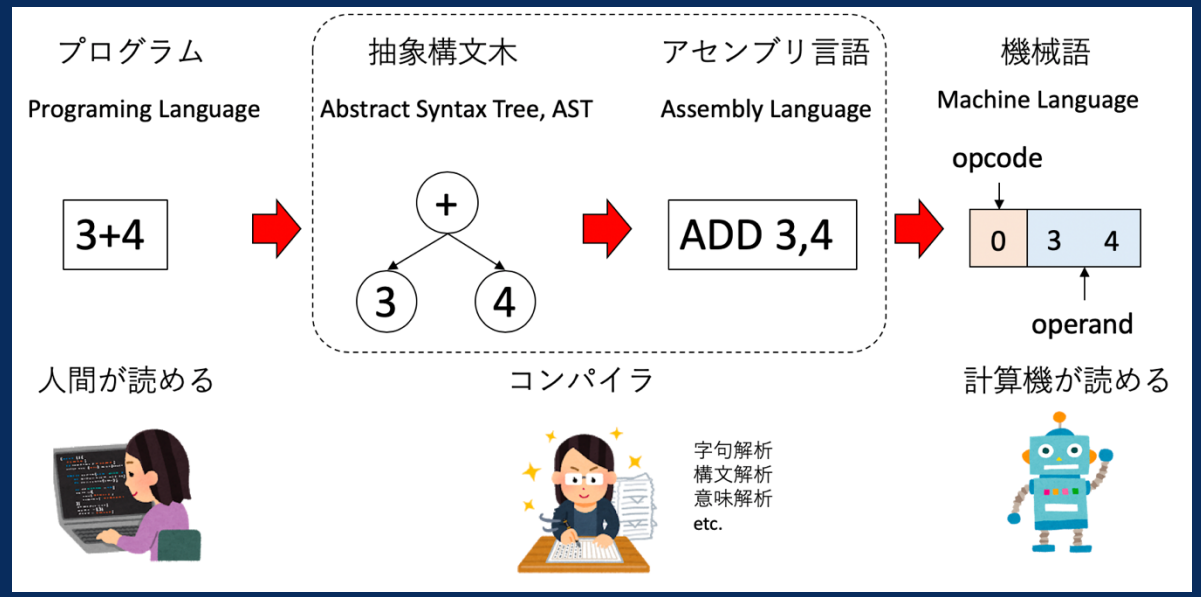
メモリ

ストレージ

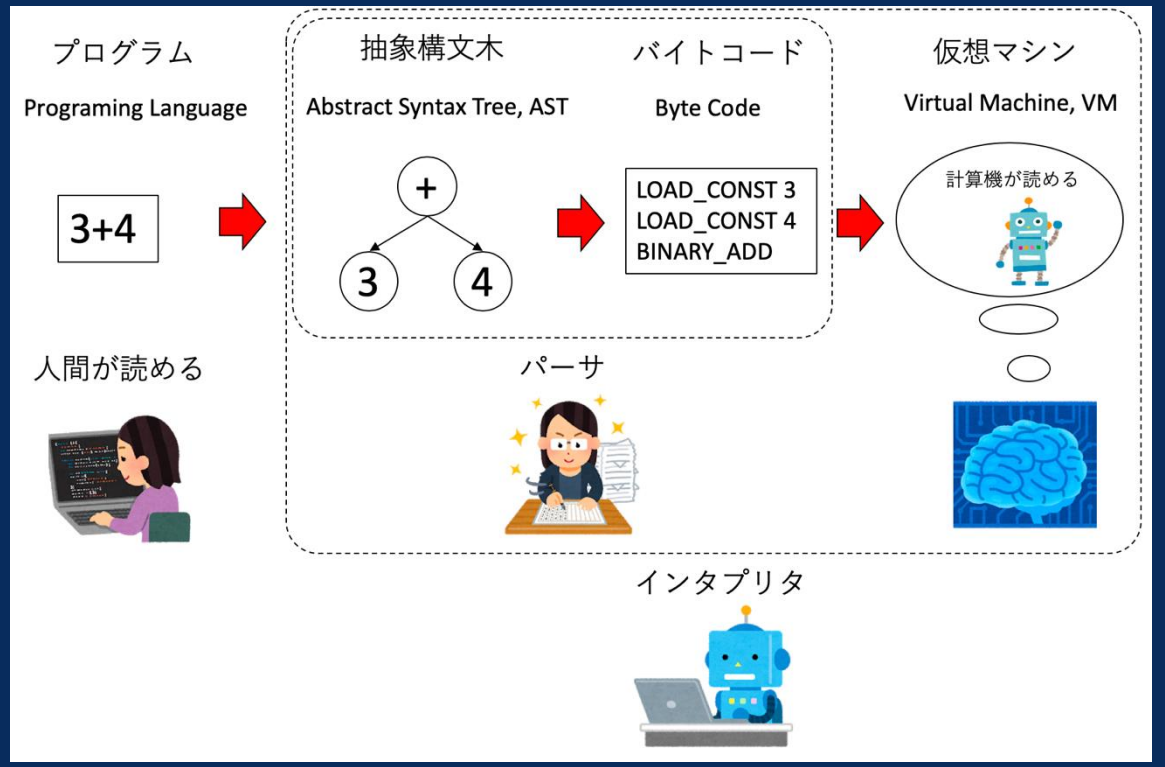
1 Billion nested loop iterations

プログラムが動く仕組み

コンパイル言語



インタプリタ言語



※実際には中間言語が多く存在

https://kaityo256.github.io/python_zero/howtowork/index.html

Pythonが動く仕組み

コンパイル言語

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

インタプリタ言語

```python
print("Hello, World!")
```

↓ バイトコード変換

```
0    RESUME      0

1    LOAD_NAME    0 (print)
     PUSH_NULL
     LOAD_CONST   0 ('Hello, World!')
     CALL         1
     RETURN_VALUE
```

↓ 仮想マシン内でC言語を実行

Zen of Python

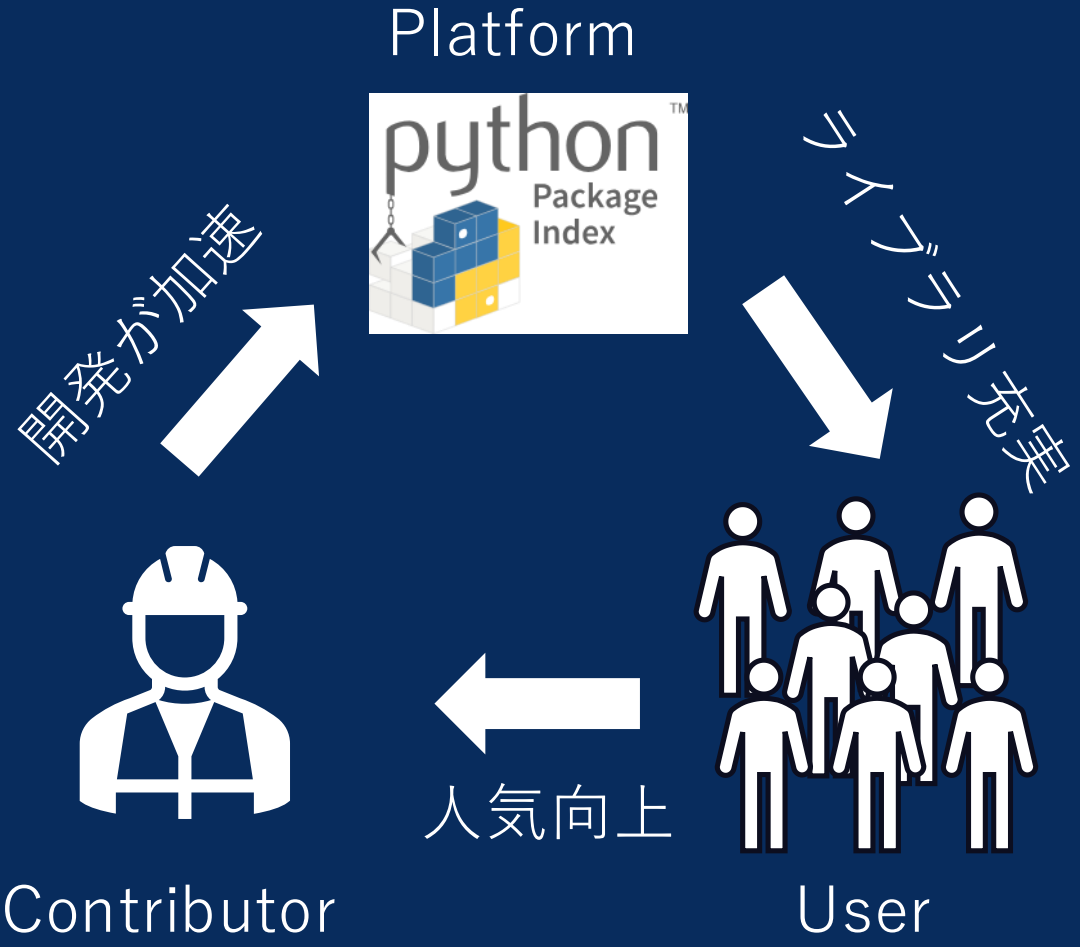
PEP (Python Enhancement Proposals)

...Pythonの設計文書

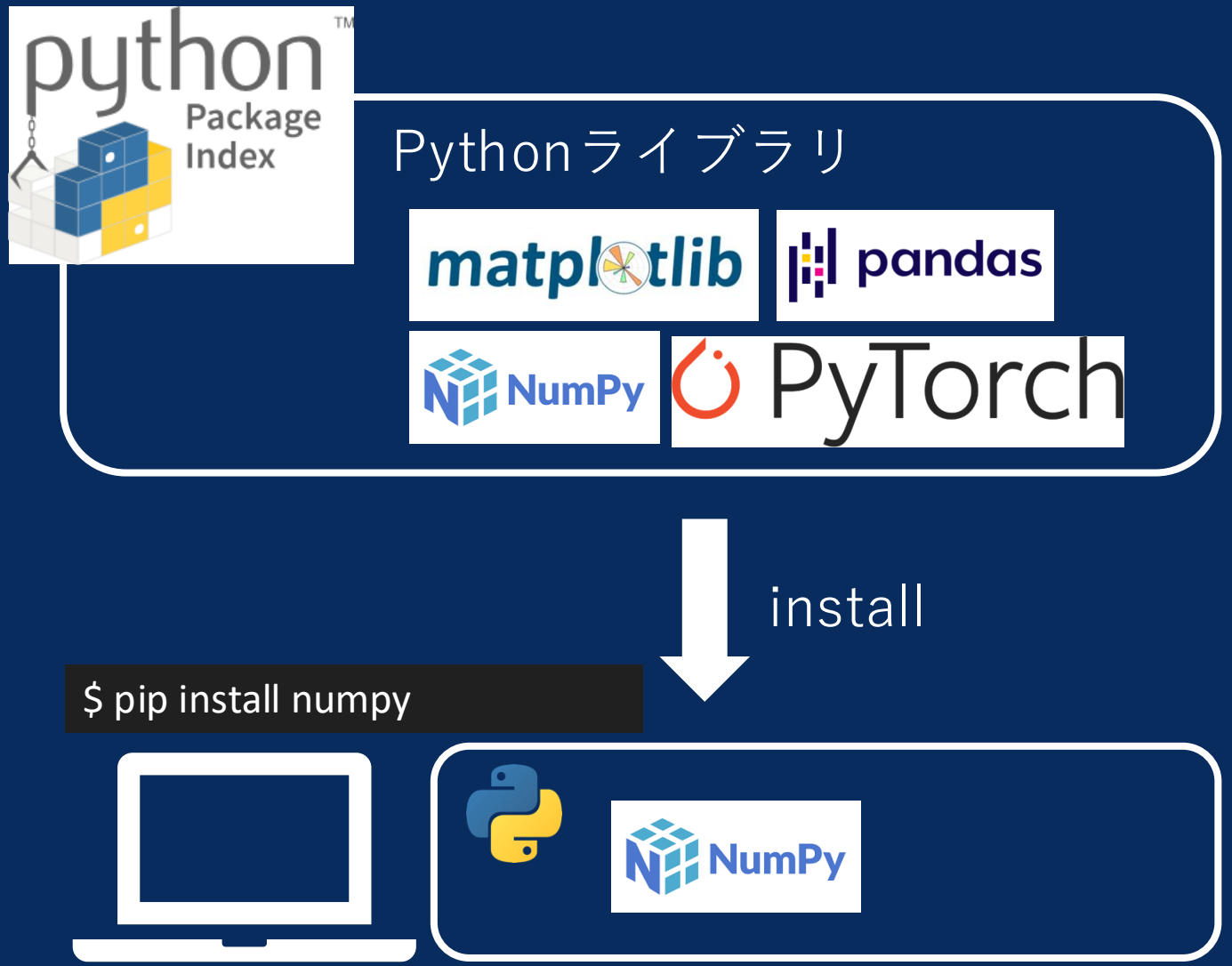
PEP20

```
>>> import this
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
...
```

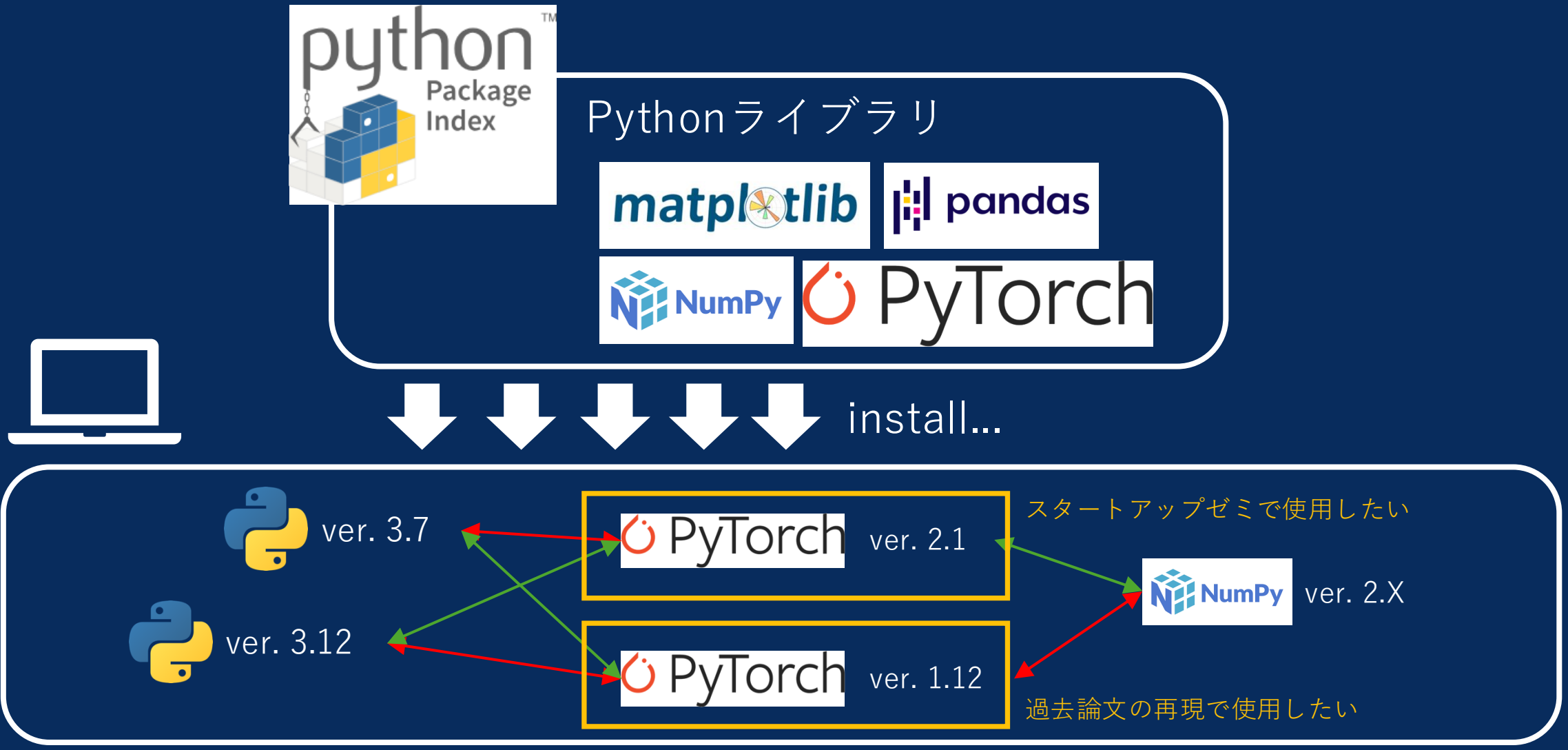
可読性を特に重要視



PyPIとpip


python
Package
Index
Pythonライブラリ
matplotlib
pandas
NumPy
PyTorch
install
$ pip install numpy
NumPy
Local Python Environment

PyPIとpip

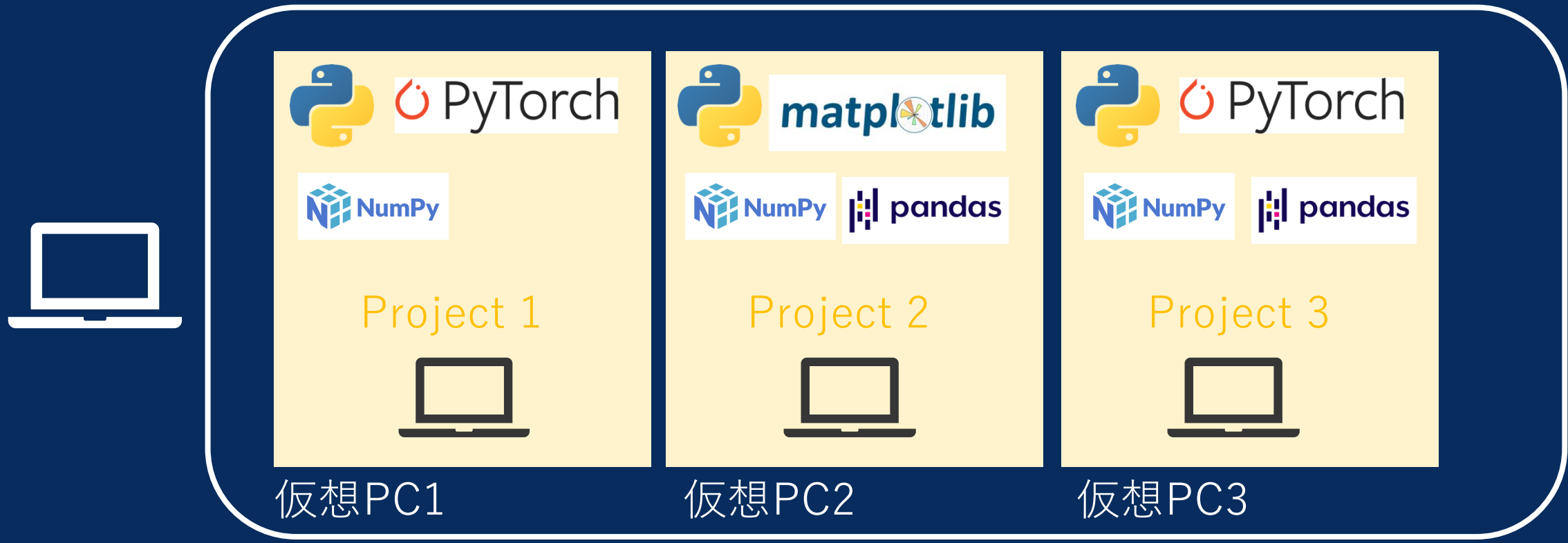

python
Package
Index
Pythonライブラリ
matplotlib
pandas
NumPy
PyTorch
install...
ver. 3.7
ver. 3.12
PyTorch
ver. 2.1
PyTorch
ver. 1.12
スタートアップゼミで使用したい
NumPy
ver. 2.X
過去論文の再現で使用したい
Local Python Environment

仮想環境

Q. 何がダメだった？

仮想環境

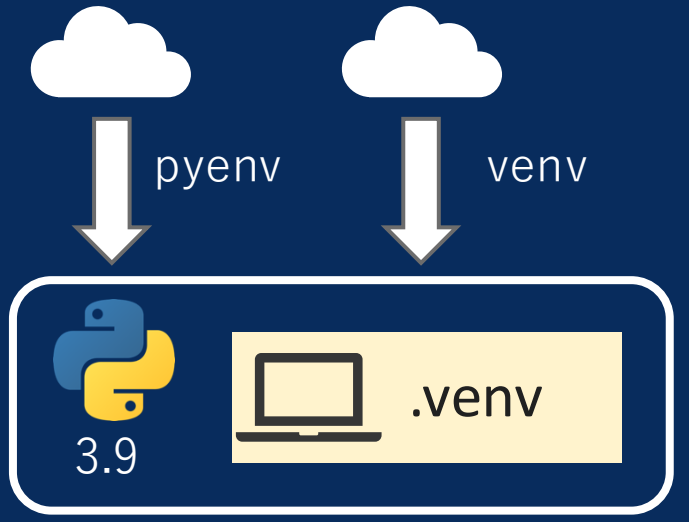
Q. 何がダメだった？→**仮想環境**



パッケージ管理

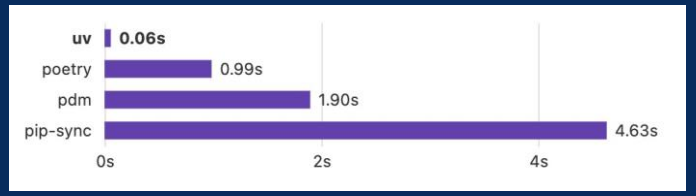
```
$ pyenv install 3.9
$ python -m venv .venv
$ source .venv/bin/activate
```

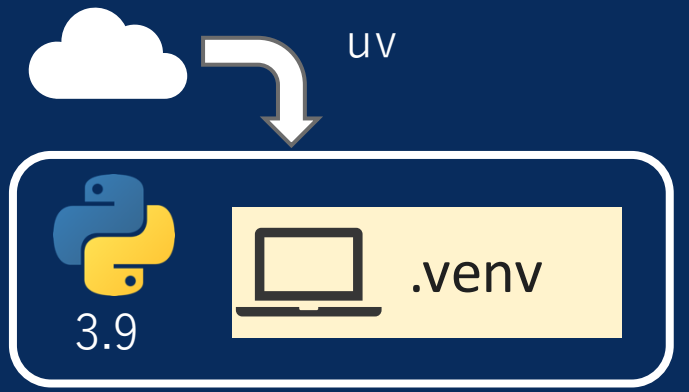
❌依存関係は自分で解決
❌遅い



```
$ uv sync
```

⭕依存関係児童解決
⭕早い

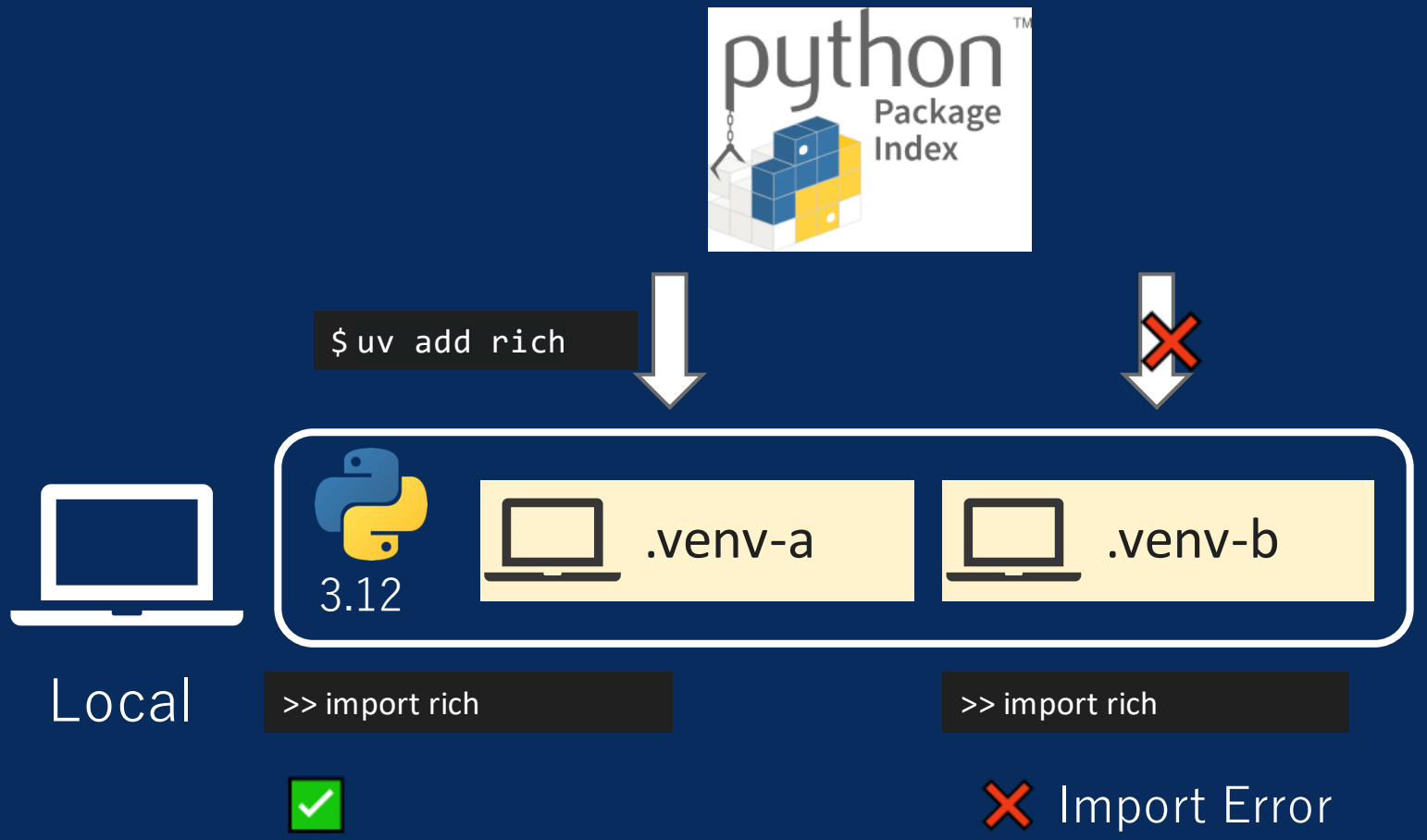




ハンズオン① 3分

- src/02_virtual-env/を確認
- PyPIからpip installしてみよう！

ハンズオン① 解説



Q1. 同プロジェクト内に複数のPython versionをinstallできる？
Q2. （例えば、）サーバで1からPython プロジェクトを設定する場合、どうする？

2026年3月19日　企業

OpenAI、Astral を買収へ

Codex の成長を加速し、次世代の Python 開発者ツールを支援

記事を音声で聴く　3:56　　共有

本日、OpenAI は Astral を買収し、強力なオープンソースの開発者向けツールを Codex のエコシステムに迎えることを発表します。

Astral は、**uv**、**Ruff**、および **ty** のような最新ツール群を通じて開発者の高速化を支える、最も広く使われているオープンソース Python ツールの一部を構築してきました。これらのツールは何百万もの開発ワークフローを支え、現代の Python 開発の基盤の一部となっています。開発者第一の理念のもと、買収完了後も OpenAI は Astral のオープンソース製品を支援する予定です。Astral のツール群とエンジニアリングの専門性を OpenAI に迎えることで、Codex に関する取り組みを加速し、ソフトウェア開発ライフサイクル全体で AI ができることを広げていきます。

ハンズオン② 5分

- 03_vending-machine/を確認
- 3つのコードの違いを観察しよう！

3つのコードは何が違う？

	Vanilla	Object	Test
特徴	場当たり的	役割（Object）に分割	テストを先に書く
強み	初速が速い	可読性が高い	バグが発生しにくい
強み	保守が難しい	設計が困難	初速が遅い

3つのコードは何が違う？

	Vanilla	Object	Test
特徴	場当たり的	役割（Object）に分割	テストを先に書く
強み	初速が速い	可読性が高い	バグが発生しにくい
強み	保守が難しい	設計が困難	初速が遅い

プログラミング・パラダイム

プログラムをどのような考え方や構造に基づいて設計するか

- オブジェクト指向（Object Oriented Programing）
- テスト駆動（Test Driven Development）

など多数

複数人・大規模プロジェクト開発において強力 → **LLMとの共同において必須**

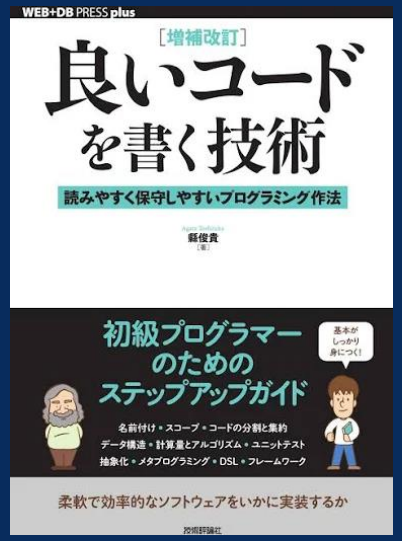
おすすめ

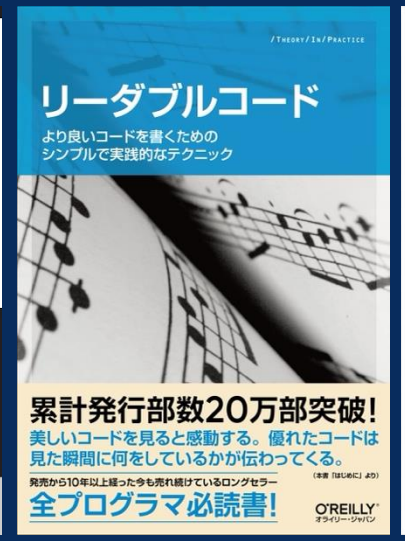
時間の都合から、詳しい説明は省略。

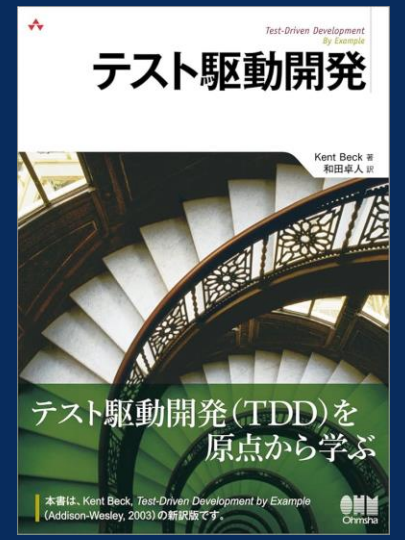
時間がない人
- オブジェクト指向: 例1, 例2
- テスト駆動開発: 例1, 例2

についてのWeb記事を読みましょう。

詳しく知りたい人
- 👉を読むのがいいです。
- 研究室にある本もあります。







LLMとプログラミングパラダイム

各.pyはLLMによって生成されたものです。

Vanilla

あなたは学部四年生のプログラミング初心者です。
添付のassignment.mdに回答してください。

オブジェクト指向

あなたは学部四年生のプログラミング初心者です。
添付のassignment.mdに回答してください。
その際、オブジェクト指向に則って開発を行なってください。

テスト駆動

あなたは学部四年生のプログラミング初心者です。
添付のassignment.mdに回答してください。
その際、テスト駆動開発に則って開発を行なってください。

教訓:
生成されるコードはプロンプトに強く依存する。
→生成者の技量

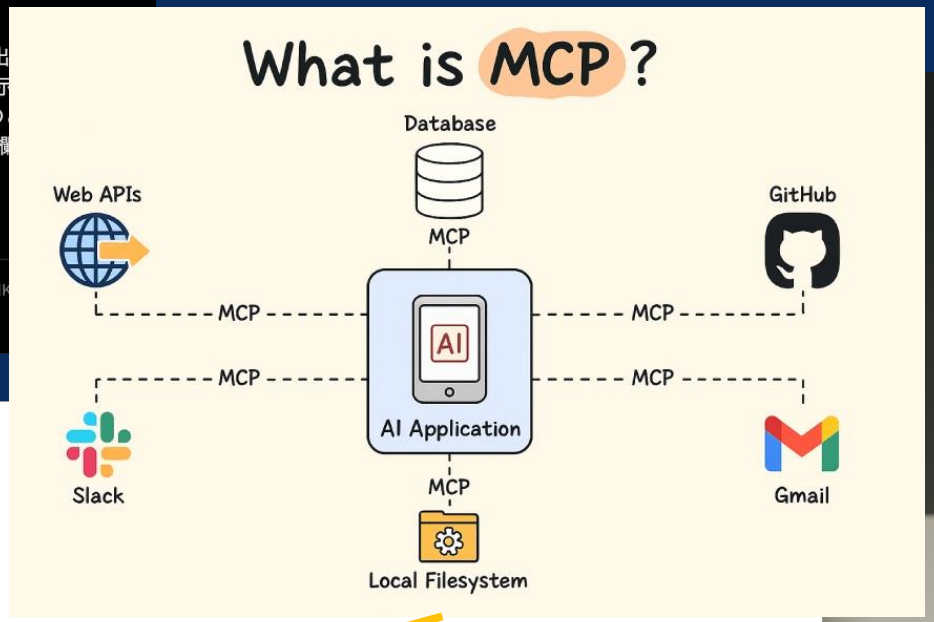
重要: ドメイン知識をLLMに制約する
=システム・アーキテクチャ

第二部:
LLMを使おう！

LLM使っていますか？

AIオールイン宣言

さとり
@satori_sz9
Subscribe
Show translation
ChatGPTを5倍賢くしたい人は全員これやった方が良い。出
対して「これを60点として100点の解答をください」と指示
い繰り返すと5倍賢い答えが返ってくる。とくに長文出力の
発揮する。論文でも検証済みで簡単な上に効果抜群。リプ欄
ので5秒でも見てみて。
1:38 PM · Oct 2, 2024 · 435.6K Views
6 383 5.9K 3.1K
Relevant



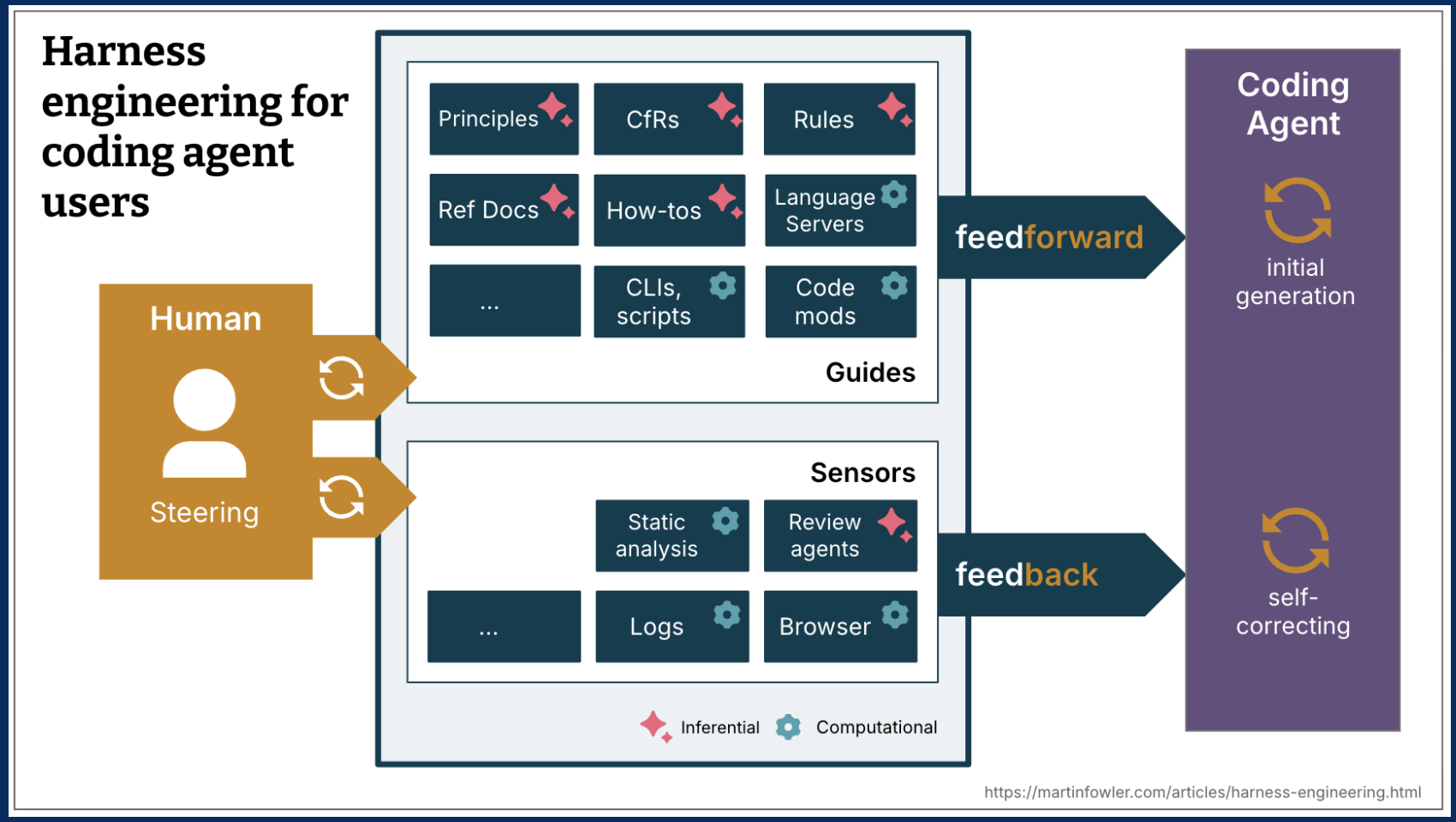
Prompt Engineering ⇒ Context Engineering ⇒ Environment Engineering

次はこれ？？

フルスイング by DNA

最近の潮流: Harness Eng.

コードレビューの自律化



https://martinfowler.com/articles/harness-engineering.html

とはいえ、新陳代謝は激しい

- 最新情報全てにキャッチアップする必要はない
  - 大半は淘汰される
- 月単位で革新的な技術が提供されうる
  - フットワークを軽くできるか？は重要

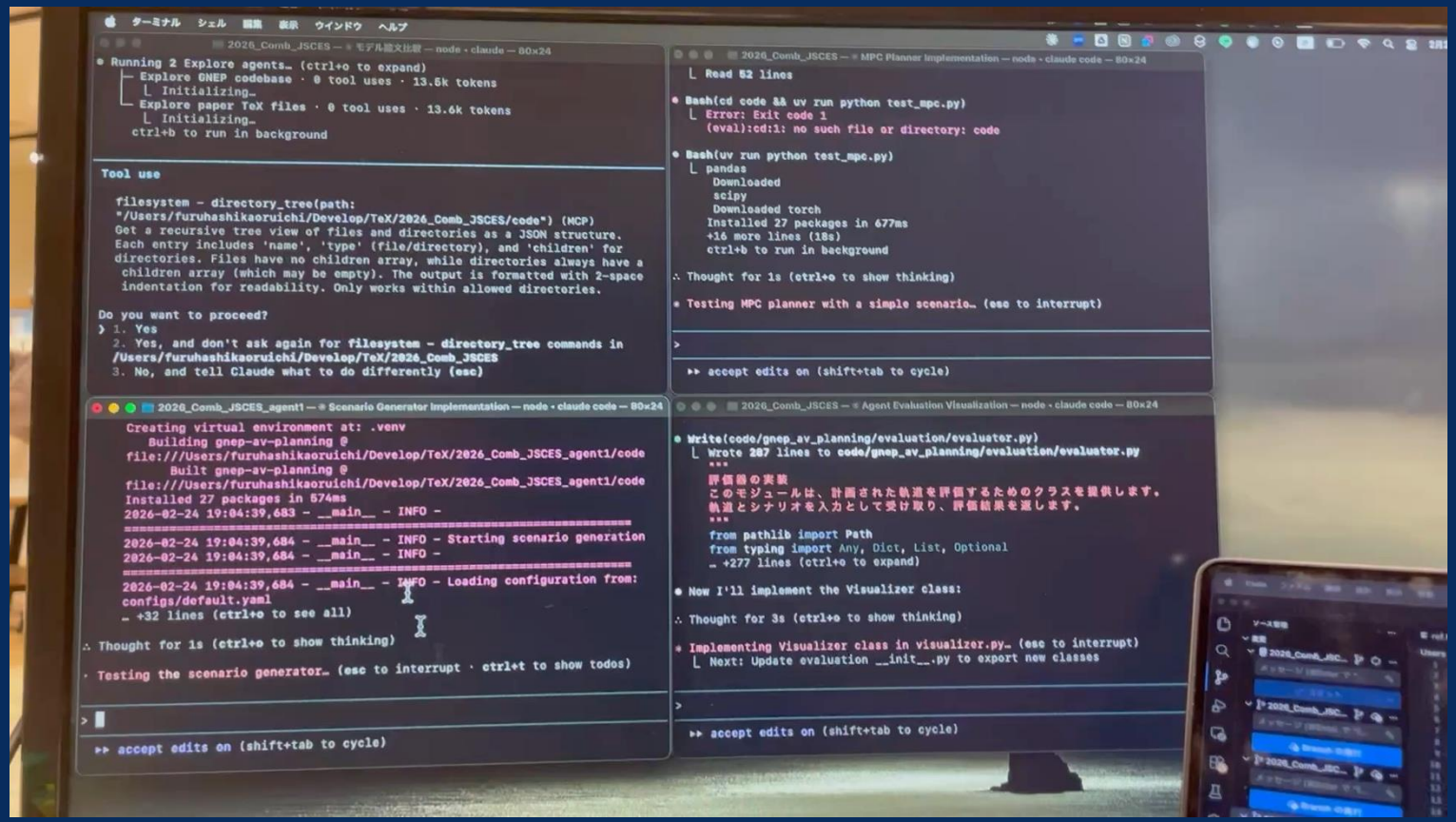
- 情報共有は重要
  - 古橋のケースを紹介します

利用しているサービス

- 全般
  - ChatGPT:
    - Pro: 執筆論文のレビューなど、論理検証してほしいとき
    - Thinking: デフォルト、即時で議論したいとき
    - Instant: プライベートなど（メールとか）
  - Claude: ほぼ使っていない
  - Gemini: 画像生成のみ
- Coding Agent
  - Codex: メインで使用（ChatGPTの有料プランに付随）
  - Claude Code: 予算都合で終了予定
    - Codex or Claude Codeはよく議論されるが、大して差はない。まず使うことが重要。
- VSCode
  - Copilot: TeXのコンパイルエラーなど、ちょっとした内容

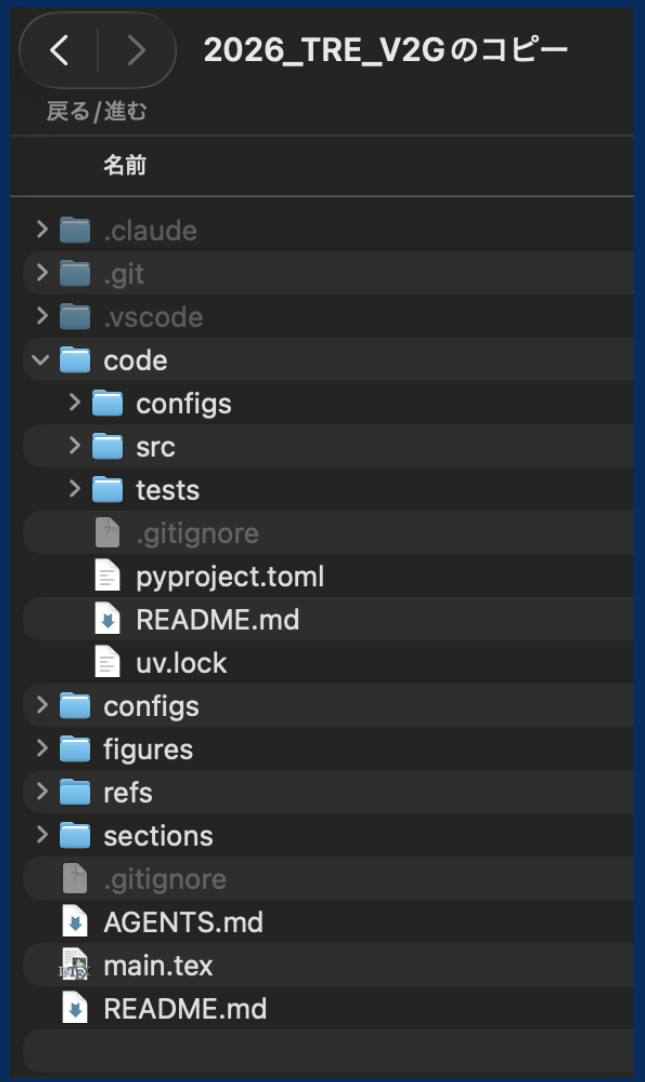
Coding Agent

Claude



論文を書くとき

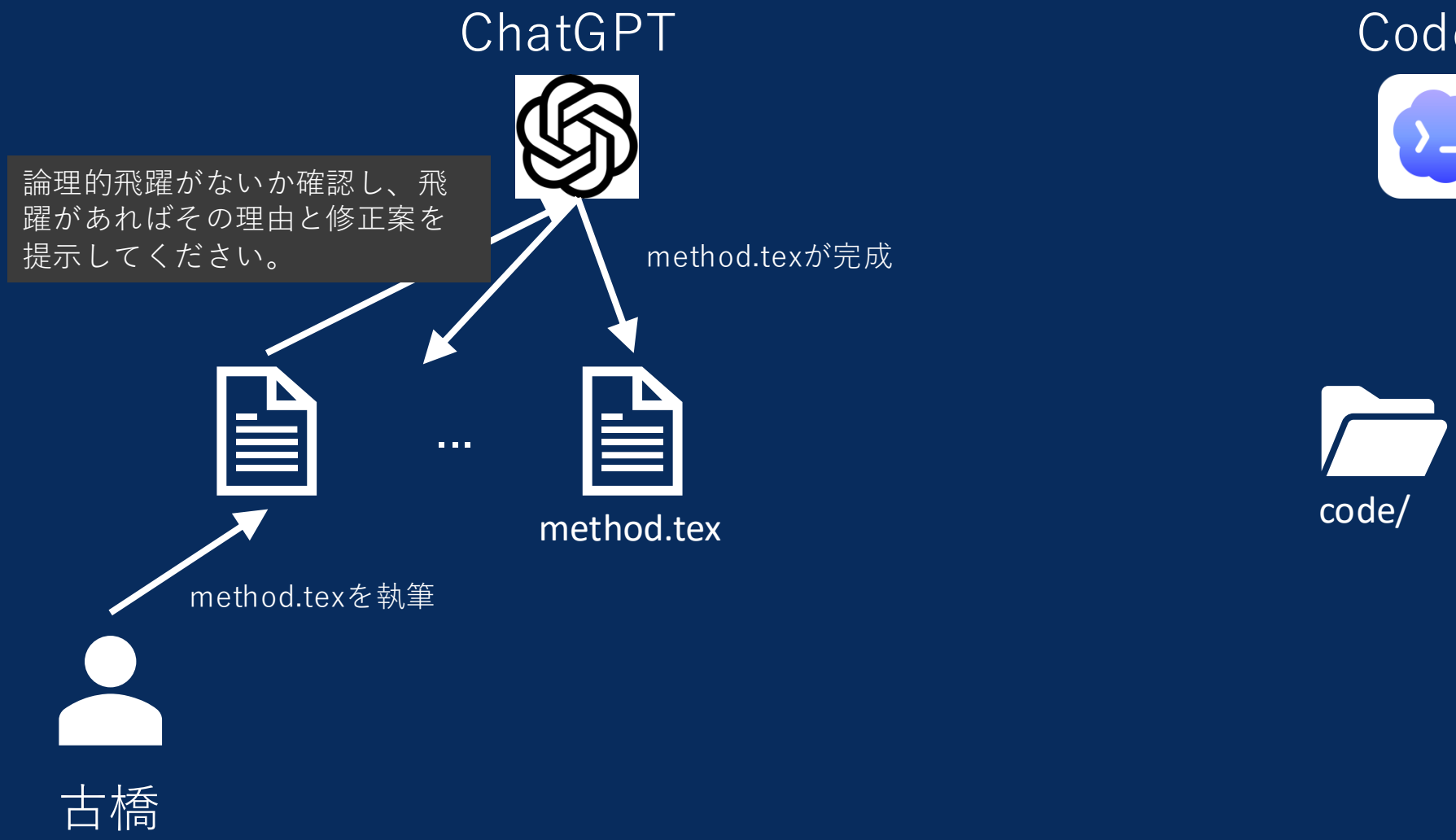
現在進行中の論文執筆フォルダ



.git/	Git管理は最も重要！ ex. 破壊的変更, 並列エージェント
code/	Pythonのソースファイルを格納 コーディング規則を書いたREADME.mdを作成
.claude/	Claude Codeの Agentの設定（Skillsなど）を格納
refs/	参考論文, 数値実験結果, 自分の考察メモを.mdで保存
sections/	.texによる原稿執筆
AGENTS.md	Coding Agentの一般的な設定ファイル

論文を書くとき

例えば、数値計算用のコード作成をするとき



論文を書くとき

例えば、数値計算用のコード作成をするとき

ChatGPT

Codex

method,texの計算をcodexに考えてもらいます。
指示文を整理した、instruction.md考えてください。

コード可読性のため、オブジェクト指向を採用してください。
例えば、
model/
・vehicle.py
・…
などを考えます。
適切にクラスを設定してください。
実験管理もしやすいように工夫してください。

method.tex

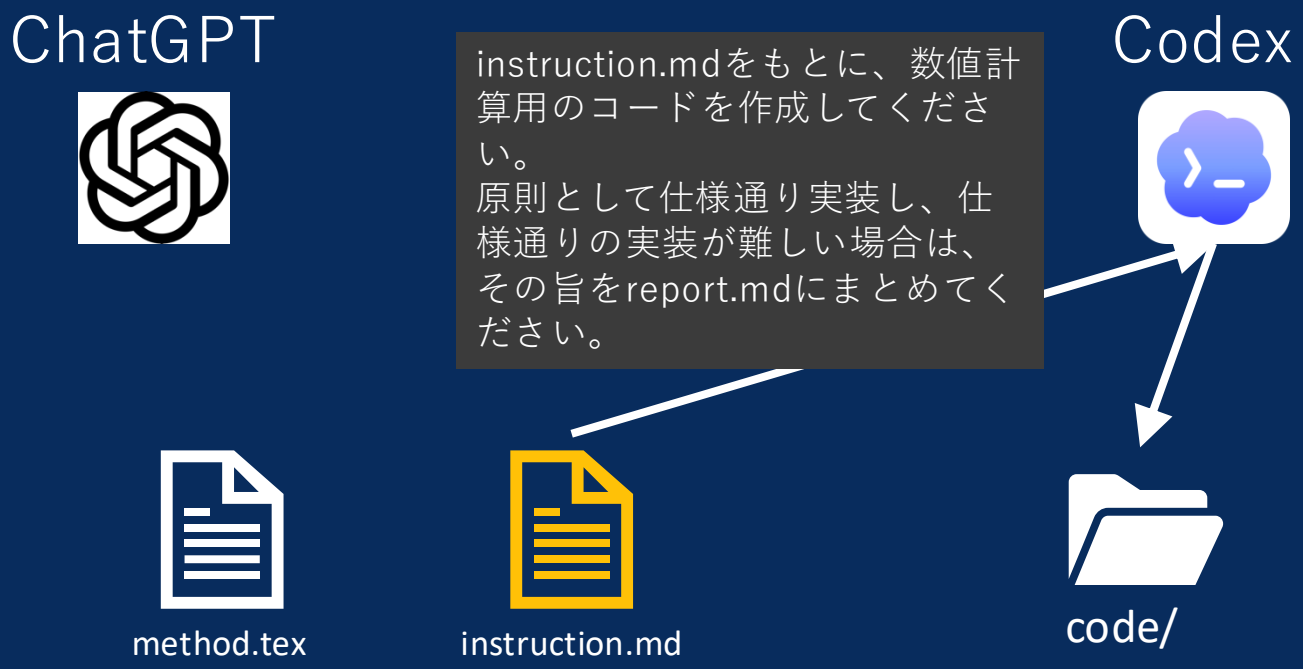
instruction.md

code/

古橋

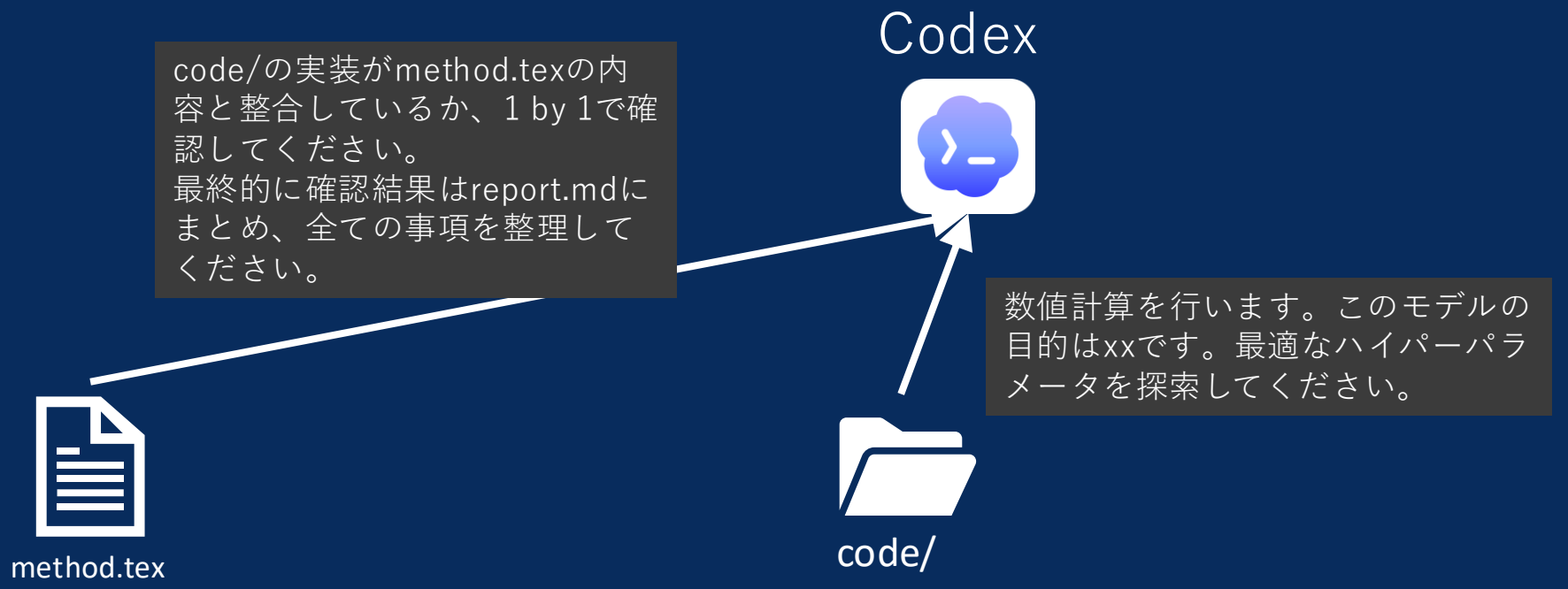
論文を書くとき

例えば、数値計算用のコード作成をするとき



論文を書くとき

例えば、数値計算用のコード作成をするとき



そのほかの事例

論文のサーベイ

類似の論文はありますか？
TR-B, TR-C, transportation scienceを軸にレビューしてください。
論文を提示する際には、要約と選定理由、doiを併記してください。

※TR: Transportation Research

発表資料の精査

添付資料の発表の流れを構造化してください。
特に論理的飛躍がある部分がないか、注意深く精査してください。
また、想定質問も10個作成してください

論文ドラフトの作成

添付論文の執筆スタイル、マクロな論文構成とミクロな文体のリズムを言語化してください。
最終的に500文字程度に圧縮してファイルに保存してください。

作成ファイルを使用

添付のファイルの文章執筆スタイルに合わせて、以下の内容の論文執筆を行なってください。
投稿先はTR-Cを想定しています。

おまけ: 私のLLM勉強法

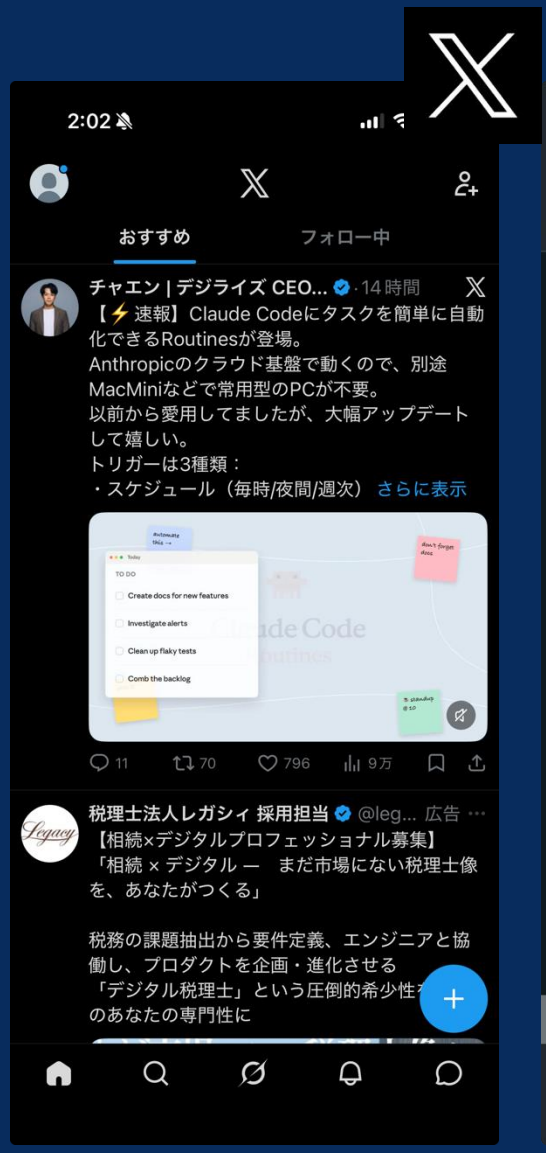
❌ 何もしない

🧐 詳しい人に聞く（他者依存）

Better:

- 自身の情報ソースを持つ
  - X
  - はてなブックマーク
  - Zenn, Qiita
  - など？
- 信頼できるソース
  - Claude Code Best Practices
  - Codex Best Practices

ただし、
最新情報のほとんどは無視してok



おわり