プログラミングを**科学**する
―表現する、処理する、証明する―

🎤田邉 裕大

東京科学大学
情報理工学院 数理・計算科学系　助教

ようこそ情報理工学院へ！

情報化社会の未来を創造する

https://www.isct.ac.jp/ja/001/about/organizations/school-of-computing

ようこそ情報理工学院へ！

情報化社会の未来を創造する

https://www.isct.ac.jp/ja/001/about/organizations/school-of-computing

https://educ.titech.ac.jp/cs/

https://educ.titech.ac.jp/is/

ようこそ情報理工学院へ！

情報化社会の未来を創造する

https://www.isct.ac.jp/ja/001/about/organizations/school-of-computing

数理科学と計算機科学を学修し、
情報化社会における複雑な課題の本質を論理的・数学的に追究します。

数理科学？

現代社会は情報化社会といわれ、多種多様な情報が社会のすみずみに深い影響を及ぼしています。数理・計算科学系では、そのような情報を科学的なアプローチで扱う方法を学修します。具体的には、コンピュータを使った新しい数学を駆使するアプローチ、現実の諸問題を数理モデルに基づいて解決するアプローチ、そしてコンピュータ・サイエンス、つまり情報処理を「計算」としてとらえるアプローチと、実際にそれを実行するコンピュータ・システムの設計方法を学びます。これら専門知識に裏付けられた手法を駆使して課題を解決することによって、国際的に活躍できる人材を養成することを目的としています。

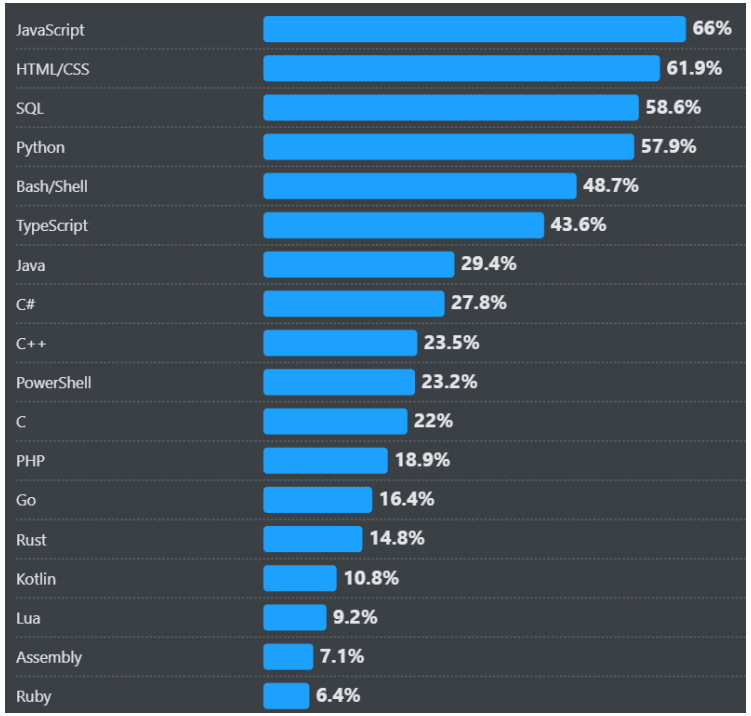
https://educ.titech.ac.jp/is/about_us/

https://educ.titech.ac.jp/cs/

https://educ.titech.ac.jp/is/

私の専門分野：
プログラミングおよびプログラミング言語(PL)

2025 Developer Survey 2025.07.30



https://survey.stackoverflow.co/2025/

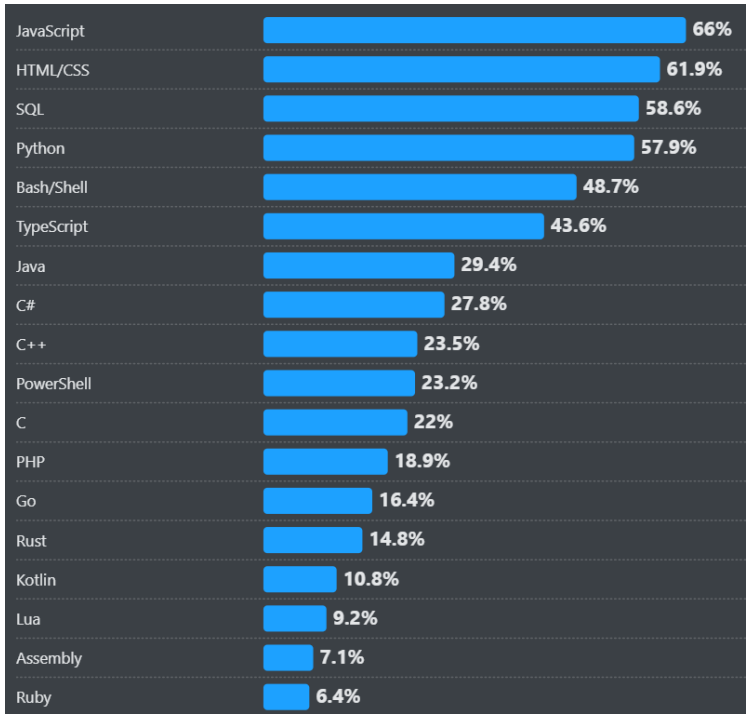
TIOBE Index for July 2025

Jul 2025	Jul 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.98%	+10.85%
2	2		C++	9.80%	-0.53%
3	3		C	9.65%	+0.16%
4	4		Java	8.76%	+0.17%
5	5		C#	4.87%	-1.85%
6	6		JavaScript	3.36%	-0.43%
7	7		Go	2.04%	-0.14%
8	8		Visual Basic	1.94%	-0.13%
9	24	⇈	Ada	1.77%	+0.99%
10	11	↑	Delphi/Object Pascal	1.77%	-0.12%

https://www.tiobe.com/tiobe-index/

私の専門分野：
プログラミングおよびプログラミング言語(PL)

2025 Developer Survey 2025.07.30



https://survey.stackoverflow.co/2025/

TIOBE Index for July 2025

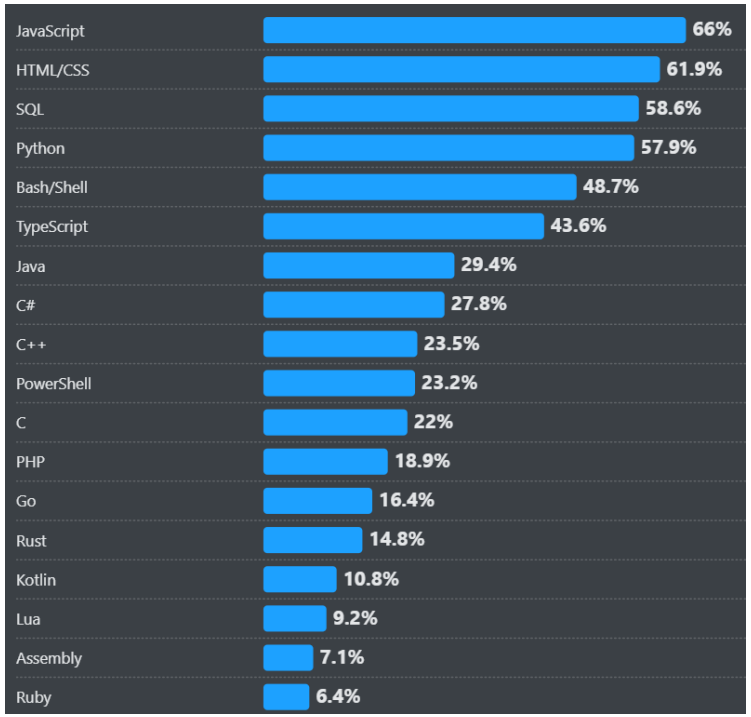
Jul 2025	Jul 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.98%	+10.85%
2	2		C++	9.80%	-0.53%
3	3		C	9.65%	+0.16%
4	4		Java	8.76%	+0.17%
5	5		C#	4.87%	-1.85%
6	6		JavaScript	3.36%	-0.43%
7	7		Go	2.04%	-0.14%
8	8		Visual Basic	1.94%	-0.13%
9	24	⇈	Ada	1.77%	+0.99%
10	11	↑	Delphi/Object Pascal	1.77%	-0.12%

https://www.tiobe.com/tiobe-index/

Q. 超絶技巧
プログラマーなの？

私の専門分野：
プログラミングおよびプログラミング言語(PL)

2025 Developer Survey 2025.07.30



https://survey.stackoverflow.co/2025/

TIOBE Index for July 2025

Jul 2025	Jul 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.98%	+10.85%
2	2		C++	9.80%	-0.53%
3	3		C	9.65%	+0.16%
4	4		Java	8.76%	+0.17%
5	5		C#	4.87%	-1.85%
6	6		JavaScript	3.36%	-0.43%
7	7		Go	2.04%	-0.14%
8	8		Visual Basic	1.94%	-0.13%
9	24	⇈	Ada	1.77%	+0.99%
10	11	↑	Delphi/Object Pascal	1.77%	-0.12%

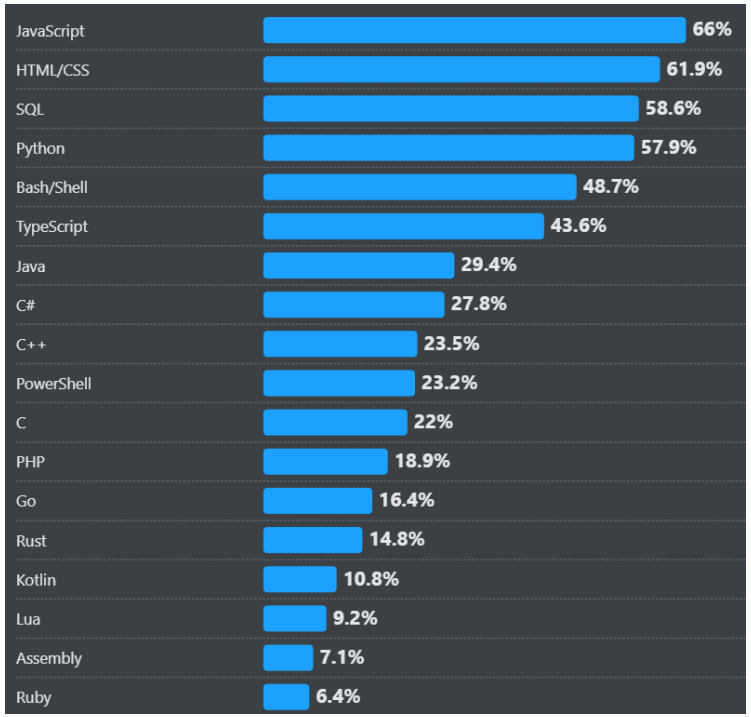
https://www.tiobe.com/tiobe-index/

Q. 超絶技巧プログラマーなの？

A. いいえ。
（※そういう研究者もいます。）

私の専門分野：
プログラミングおよびプログラミング言語(PL)

2025 Developer Survey 2025.07.30



https://survey.stackoverflow.co/2025/

TIOBE Index for July 2025

Jul 2025	Jul 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.98%	+10.85%
2	2		C++	9.80%	-0.53%
3	3		C	9.65%	+0.16%
4	4		Java	8.76%	+0.17%
5	5		C#	4.87%	-1.85%
6	6		JavaScript	3.36%	-0.43%
7	7		Go	2.04%	-0.14%
8	8		Visual Basic	1.94%	-0.13%
9	24	⇈	Ada	1.77%	+0.99%
10	11	↑	Delphi/Object Pascal	1.77%	-0.12%

https://www.tiobe.com/tiobe-index/

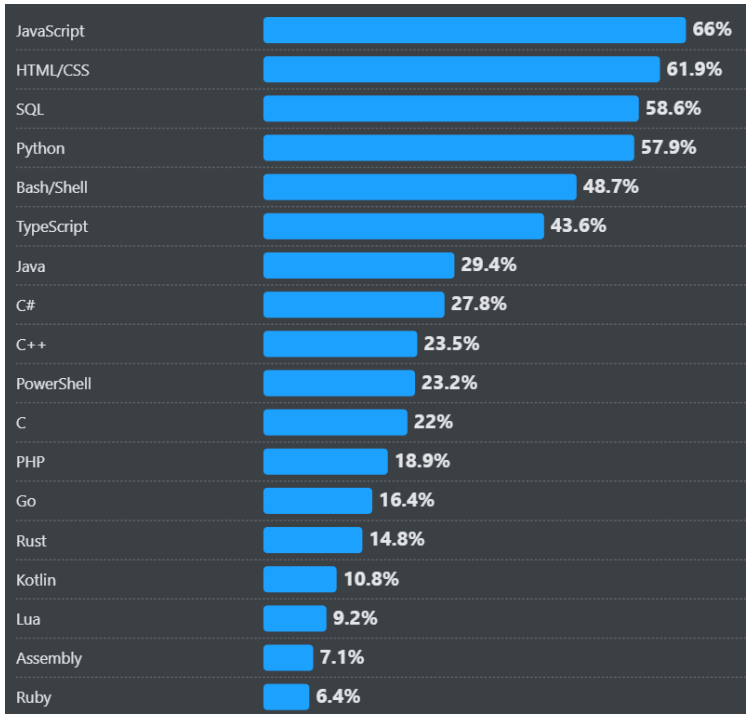
Q. 超絶技巧プログラマーなの？

A. いいえ。
（※そういう研究者もいます。）

Q. ←の言語を開発しているの？

私の専門分野：
プログラミングおよびプログラミング言語(PL)

2025 Developer Survey 2025.07.30



https://survey.stackoverflow.co/2025/

TIOBE Index for July 2025

Jul 2025	Jul 2024	Change	Programming Language	Ratings	Change
1	1		Python	26.98%	+10.85%
2	2		C++	9.80%	-0.53%
3	3		C	9.65%	+0.16%
4	4		Java	8.76%	+0.17%
5	5		C#	4.87%	-1.85%
6	6		JavaScript	3.36%	-0.43%
7	7		Go	2.04%	-0.14%
8	8		Visual Basic	1.94%	-0.13%
9	24	⇈	Ada	1.77%	+0.99%
10	11	↑	Delphi/Object Pascal	1.77%	-0.12%

https://www.tiobe.com/tiobe-index/

Q. 超絶技巧
プログラマーなの？

A. いいえ。
（※そういう研究者もいます。）

Q. ←の言語を開発
しているの？

A. いいえ。
（※そういう研究者も沢山います。）

Q. じゃあPL研究って何なの？

（私の考えでは）

“PLの視座”から

コンピューティングの問題

に迫る研究

Q. じゃあPL研究って何なの？

（私の考えでは）

“PLの視座”から

コンピューティングの問題

に迫る研究

ある問題に対する解決策の
簡潔で汎用的な表現
を見つけたい！

Q. じゃあPL研究って何なの？

（私の考えでは）

“PLの視座”から

コンピューティングの問題

に迫る研究

言語それ自体が研究動機ではないPL研究も沢山ある

過去の実例：OS, インターフェース, ソフトウェア開発環境, セキュリティ, 分散計算, データベース, 機械学習, ネットワーク, 画像処理, ハードウェア設計, ブロックチェーン, 量子計算 etc.

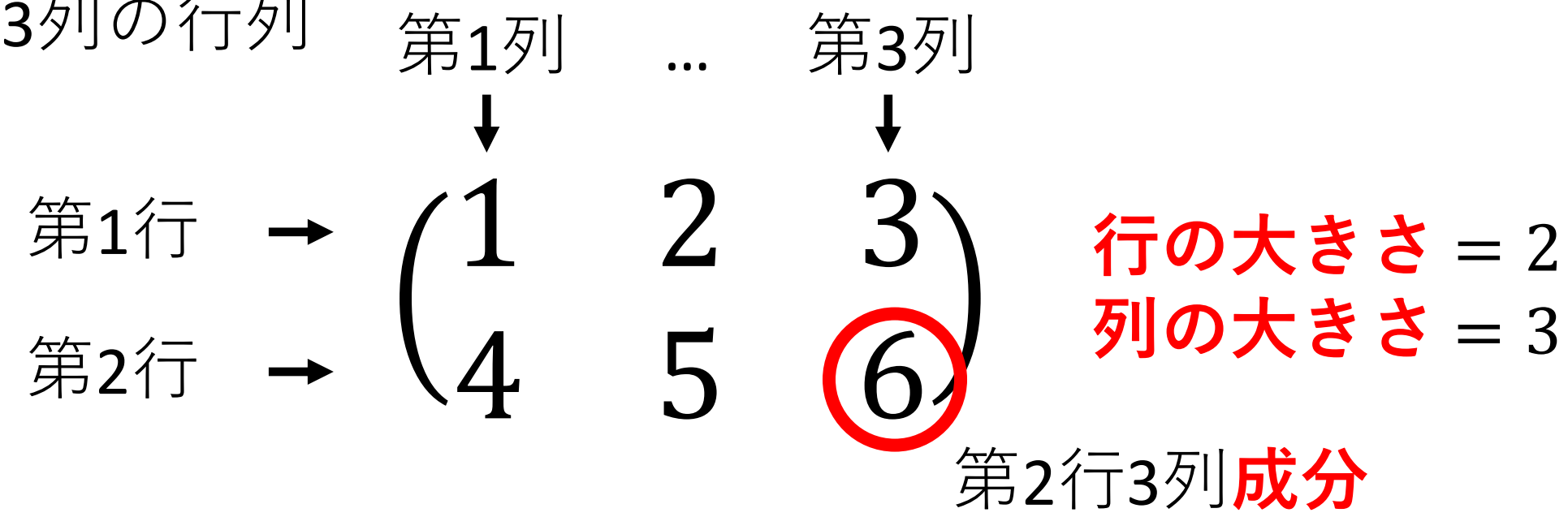
ある問題に対する解決策の
簡潔で汎用的な表現
を見つけたい！

高校数学で
プログラミングしてみよう！

数学的準備：行列

高校数学における定義：**行(横方向)と列(縦方向)を持つ数の配置**

例: 2行3列の行列



数学的準備：行列の和・差・実数倍

相当$A = B$

は以下を共に満たすことをいう。

- AとBの行・列の大きさがそれぞれ等しい。
- 各成分がそれぞれ等しい。

和$A + B$, 差$A - B$

- 各成分の和と差。

実数倍kA

- Aの各成分をk倍。

数学的準備：行列の和・差・実数倍

相当$A = B$

は以下を共に満たすことをいう。

- AとBの行・列の大きさがそれぞれ等しい。
- 各成分がそれぞれ等しい。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \neq \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

和$A + B$, 差$A - B$

- 各成分の和と差。

実数倍kA

- Aの各成分をk倍。

数学的準備：行列の和・差・実数倍

相当$A = B$

は以下を共に満たすことをいう。

- AとBの行・列の大きさがそれぞれ等しい。
- 各成分がそれぞれ等しい。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \neq \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

和$A + B$, 差$A - B$

- 各成分の和と差。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} - \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}$$

実数倍kA

- Aの各成分をk倍。

数学的準備：行列の和・差・実数倍

相当$A = B$

は以下を共に満たすことをいう。

- AとBの行・列の大きさがそれぞれ等しい。
- 各成分がそれぞれ等しい。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \neq \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

和$A + B$, 差$A - B$

- 各成分の和と差。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} - \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}$$

実数倍kA

- Aの各成分をk倍。

$$2\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$$

数学的準備：行列の和・差・実数倍

相当$A = B$

は以下を共に満たすことをいう。

- AとBの行・列の大きさがそれぞれ等しい。
- 各成分がそれぞれ等しい。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \neq \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

和$A + B$, 差$A - B$

- 各成分の和と差。

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} - \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 3 \end{pmatrix}$$

実数倍kA

- Aの各成分をk倍。

$$2\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$$

今日の本題

行列積$\boldsymbol{A \cdot B}$の定義はやや複雑

=> 次のページ

例：$N \times N$行列積を**人力**  で計算

2×2の行列積の場合…

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

例：$N \times N$行列積を**人力**  で計算

2×2の行列積の場合…

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

例：$N \times N$行列積を**人力**  で計算

2×2の行列積の場合…

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

乗算

乗算

$$= \begin{pmatrix} 0 \times 1 + (-1) \times 3 & 0 \times 2 + (-1) \times 4 \\ 1 \times 1 + 0 \times 3 & 1 \times 2 + 0 \times 4 \end{pmatrix}$$

例：$N \times N$行列積を**人力**  で計算

2×2の行列積の場合…

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$= \begin{pmatrix} 0 \times 1 + (-1) \times 3 & 0 \times 2 + (-1) \times 4 \\ 1 \times 1 + 0 \times 3 & 1 \times 2 + 0 \times 4 \end{pmatrix}$$

$$= \begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$$

例：$N \times N$行列積を**人力**  で計算

2×2の行列積の場合…

$\pi/2$回転行列

$$\begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -\sin\left(\frac{\pi}{2}\right) \\ \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix}$$

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$= \begin{pmatrix} 0 \times 1 + (-1) \times 3 & 0 \times 2 + (-1) \times 4 \\ 1 \times 1 + 0 \times 3 & 1 \times 2 + 0 \times 4 \end{pmatrix}$$

$$= \begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$$

例：$N \times N$行列積を**人力**で計算

2×2の行列積の場合…

$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

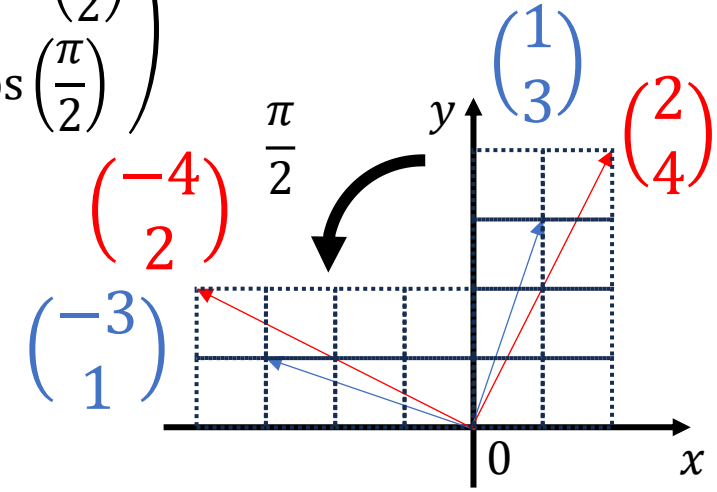
$\pi/2$回転行列

$$\begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -\sin\left(\frac{\pi}{2}\right) \\ \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$= \begin{pmatrix} 0 \times 1 + (-1) \times 3 & 0 \times 2 + (-1) \times 4 \\ 1 \times 1 + 0 \times 3 & 1 \times 2 + 0 \times 4 \end{pmatrix}$$

$$= \begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$$



$$AB = R_{\frac{\pi}{2}} B$$

$$= \begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$$

例：$N \times N$行列積を**計算機**  **で**計算

例：$N \times N$行列積を**計算機**  **で**計算

まずプログラムの**仕様**を考える：
どんな条件を満たすプログラムを書けばいいか？

例：$N \times N$行列積を**計算機** で計算

まずプログラムの**仕様**を考える：
どんな条件を満たすプログラムを書けばいいか？

① 計算手順を与える

乗算

$$c_{11} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & \cdots \end{pmatrix}$$

➡の総和を求める。これを全要素分繰り返す

例：$N \times N$行列積を**計算機** で計算

まずプログラムの**仕様**を考える：
どんな条件を満たすプログラムを書けばいいか？

① 計算手順を与える

乗算

$$c_{11} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & \cdots \end{pmatrix}$$

➡の総和を求める。これを全要素分繰り返す

② 入出力を沢山与える

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

例：$N \times N$行列積を**計算機** で計算

まずプログラムの**仕様**を考える：
どんな条件を満たすプログラムを書けばいいか？

① 計算手順を与える

乗算

$$c_{11} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & \cdots \end{pmatrix}$$

➡の総和を求める。これを全要素分繰り返す

② 入出力を沢山与える

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

③ 性質を数学的に述べる

行列$A = (a_{ij}) \in \mathbb{R}^{l \times m}, B = (b_{ij}) \in \mathbb{R}^{m \times n}$
に対し、積$(c_{ij}) \in \mathbb{R}^{l \times n}$ は以下を満たす。

$$\forall i \in [1, l], \forall j \in [1, n]. c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj} \right)$$

例：$N \times N$行列積を**計算機** で計算

まずプログラムの**仕様**を考える：
どんな条件を満たすプログラムを書けばいいか？

① 計算手順を与える

乗算

$$c_{11} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & \cdots \end{pmatrix}$$

→の総和を求める。これを全要素分繰り返す

② 入出力を沢山与える

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

仕様?: 自然言語でプロンプトを与える

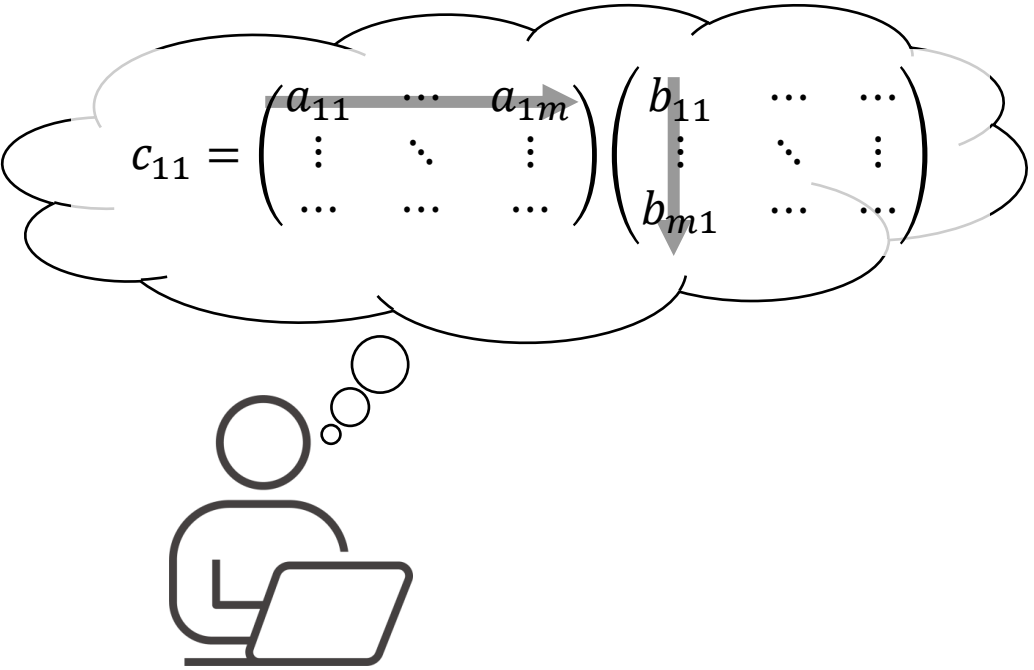
“Pythonで二次元行列の行列積を
求める関数を実装してください”

③ 性質を数学的に述べる

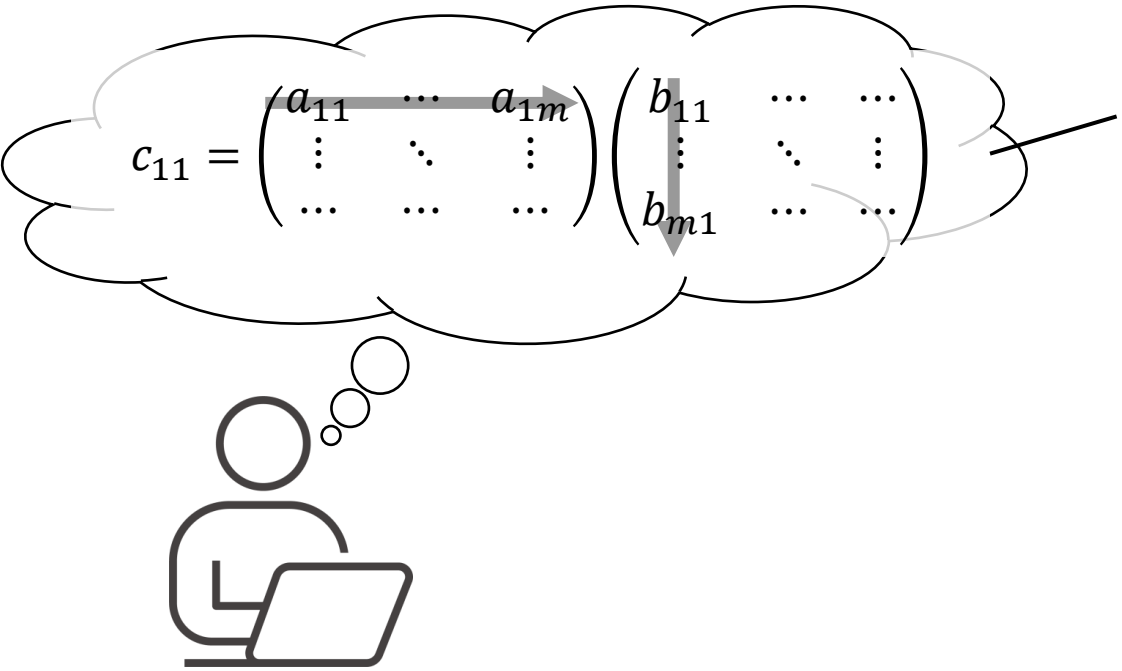
行列$A = (a_{ij}) \in \mathbb{R}^{l \times m}, B = (b_{ij}) \in \mathbb{R}^{m \times n}$
に対し、積$(c_{ij}) \in \mathbb{R}^{l \times n}$ は以下を満たす。

$$\forall i \in [1, l], \forall j \in [1, n]. c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj} \right)$$

プログラミング
＝ 計算や仕様を**表現する**行為

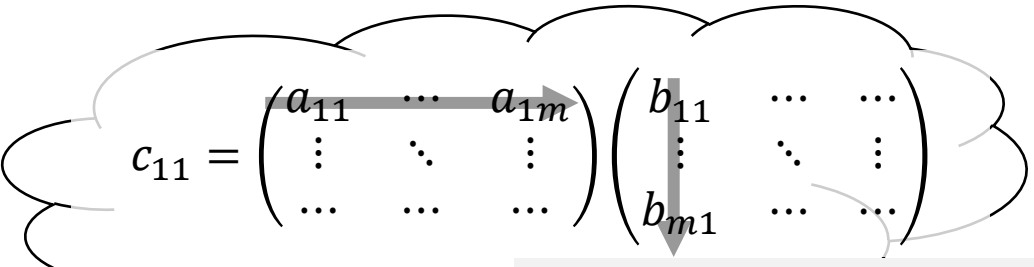


プログラミング
＝ **計算や仕様を表現する**行為



行列は**数値の二重リスト**にしよう

プログラミング
＝ 計算や仕様を表現する行為



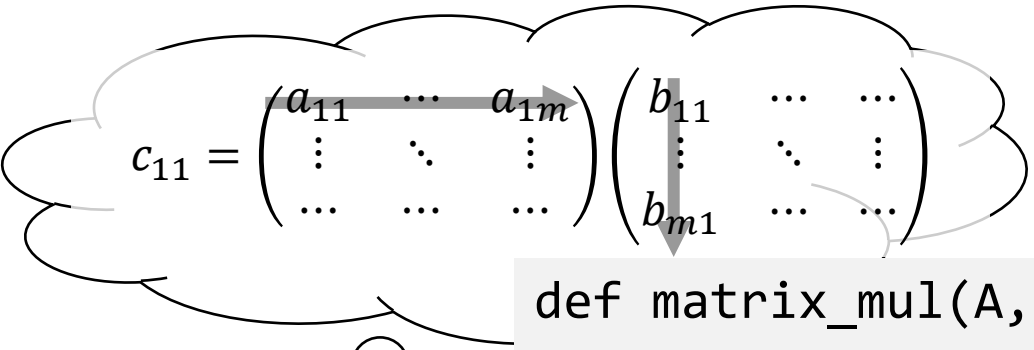
```
def matrix_mul(A, B):


  C = … # 各要素0初期化



  return C
```

まず関数定義

結果行列が束縛する
変数を用意、0初期化

変数Cを束縛する値
を結果として返す

プログラミング
＝ 計算や仕様を表現する行為

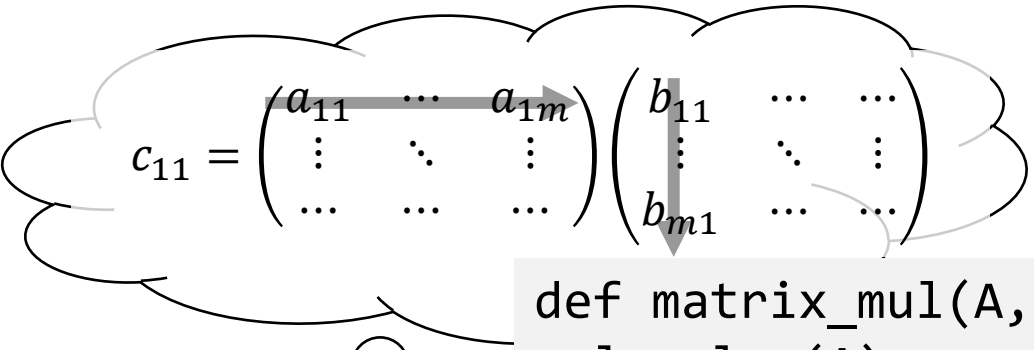


```
def matrix_mul(A, B):


  C = … # 各要素0初期化
  for i in range(_):      # Aの行
    for j in range(_):    # Bの列
      for k in range(_): # Aの列=Bの行
        C[i][j] += A[i][k] * B[k][j]
  return C
```

forループで積ABの各要素を求める

$$c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj} \right)$$

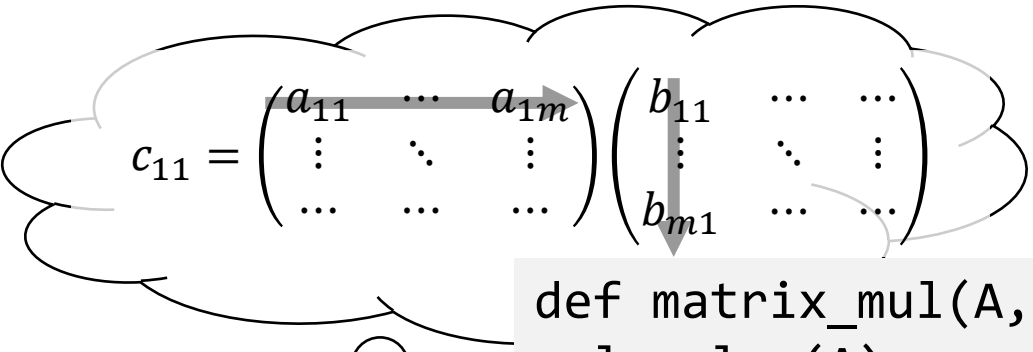
プログラミング
＝ 計算や仕様を表現する行為



```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = … # 各要素0初期化
  for i in range(l):      # Aの行
    for j in range(n):    # Bの列
      for k in range(m): # Aの列=Bの行
        C[i][j] += A[i][k] * B[k][j]
  return C
```

ループ境界の
行・列数を計算

プログラミング
＝ 計算や仕様を表現する行為

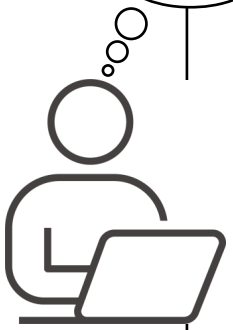


```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = … # 各要素0初期化
  for i in range(l):      # Aの行
    for j in range(n):    # Bの列
      for k in range(m): # Aの列=Bの行
        C[i][j] += A[i][k] * B[k][j]
  return C
```

無事に計算できていれば
AとBの行列積を表すリストのはず!

プログラミング言語処理系は
表現のギャップを埋める

01でプログラム
書くのは困難



```
$ python3 matrix.py
```

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = … # 結果行列Cの要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k] * B[k][j]
  return C
```

3重ループの実装

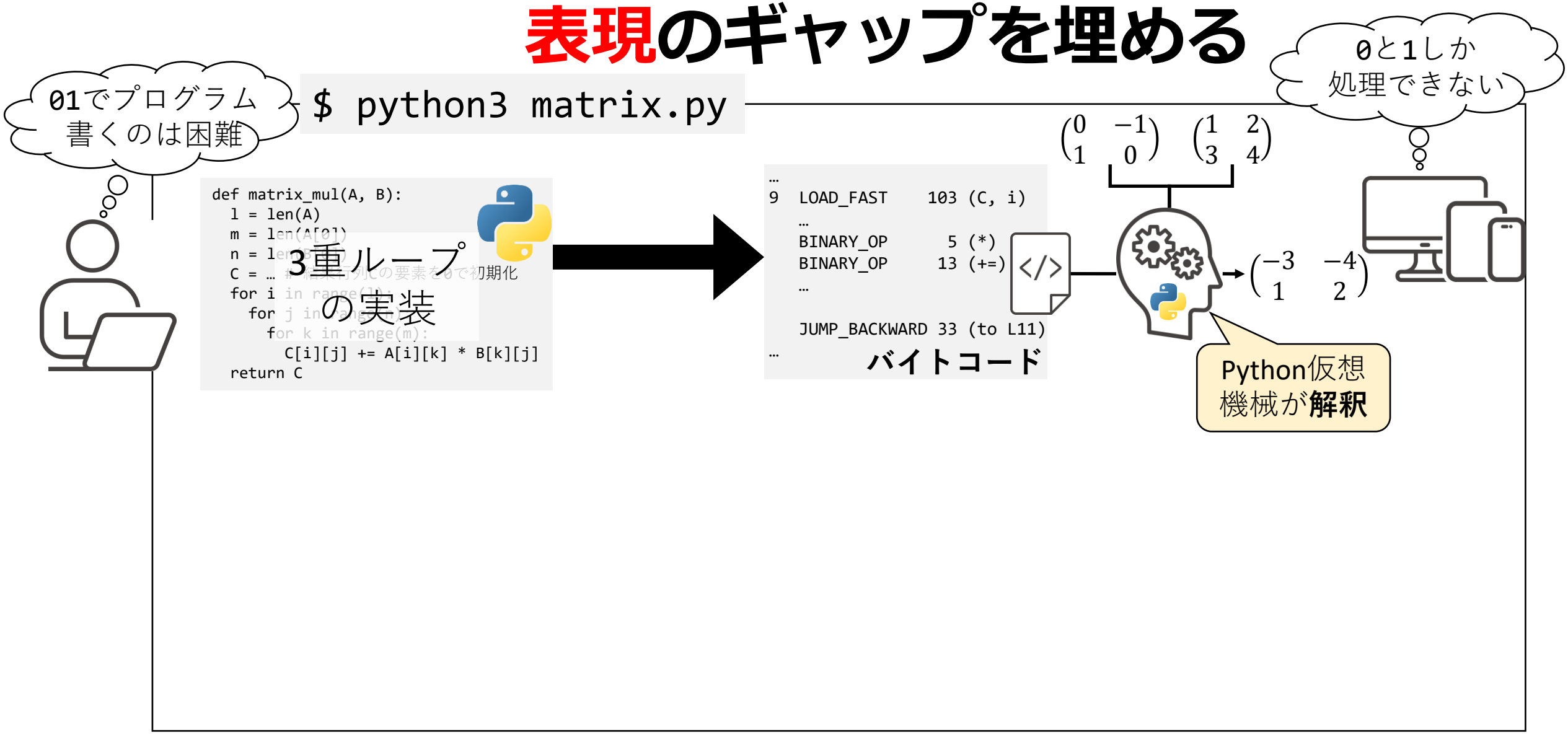
コンパイラで**変換**

```
…
9   LOAD_FAST     103 (C, i)
    …
    BINARY_OP       5 (*)
    BINARY_OP      13 (+=)
    …

    JUMP_BACKWARD 33 (to L11)
…
```

バイトコード

0と1しか
処理できない

プログラミング言語処理系は
表現のギャップを埋める



プログラミング言語処理系は
表現のギャップを埋める

01でプログラム
書くのは困難

```
$ python3 matrix.py
```

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = … # 結果行列Cの要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k] * B[k][j]
  return C
```

3重ループの実装

```
…
9   LOAD_FAST     103 (C, i)
    …
    BINARY_OP       5 (*)
    BINARY_OP      13 (+=)
    …

    JUMP_BACKWARD 33 (to L11)
…
```

バイトコード

$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ → $\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$

0と1しか
処理できない

Python仮想
機械が**解釈**

```
BINARY_OP      5 (*)
```

「BINARY_OP(*)は
C言語の関数
long_mul
で解釈せよ」

```
4319    static PyLongObject*
4320    long_mul(PyLongObject *a, PyLongObject *b)
4321    {
4322        /* fast path for single-di
4323        if (_PyLong_BothAreCompact
4324            stwodigits v = medium_value(a) * m
```

https://github.com/python/cpython/blob/main/Objects/longobject.c#L4319-L4334

プログラミング言語処理系は
表現のギャップを埋める

01でプログラム書くのは困難

```
$ python3 matrix.py
```

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = … # 結果行列Cの要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k] * B[k][j]
  return C
```

3重ループの実装

```
…
9   LOAD_FAST     103 (C, i)
    …
    BINARY_OP       5 (*)
    BINARY_OP      13 (+=)
    …
    JUMP_BACKWARD 33 (to L11)
…
```

バイトコード

$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ → $\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$

0と1しか処理できない

Python仮想機械が**解釈**

```
BINARY_OP      5 (*)
```

「BINARY_OP(*)はC言語の関数long_mulで解釈せよ」

コンパイラで**変換**

```
4319    static PyLongObject*
4320    long_mul(PyLongObject *a, PyLongObject *b)
4321    {
4322        /* fast path for single-di
4323        if (_PyLong_BothAreCompact
4324            stwodigits v = medium_value
```

```
01010101
01001000 10001001 11100101
…
01011101
11000011
```

01

https://github.com/python/cpython/blob/main/Objects/longobject.c#L4319-L4334

cpythonに内蔵

実行時は呼ぶだけ

単体テストでプログラムが
仕様を満たしているかテスト

簡単な方法：単体テスト

```
# test_matrix.py
A = [[0, -1], [1, 0]]
B = [[1,  2], [3, 4]]
assert \
 matrix_mul(A, B) == [[-3,-4],[1,2]],\
 "テスト1失敗"
print("テスト1成功！")
```

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

単体テストでプログラムが
仕様を満たしているかテスト

簡単な方法：単体テスト

```
# test_matrix.py
A = [[0, -1], [1, 0]]
B = [[1,  2], [3, 4]]
assert \
 matrix_mul(A, B) == [[-3,-4],[1,2]],\
 "テスト1失敗"
print("テスト1成功！")
```

```
B’= [[5, 6], [7, 8]]
assert \
  matrix_mulmul(B, B’) == \
    [[19,22],[43,50]],\
  “テスト2失敗"
print(“テスト2成功！")
```

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

単体テストでプログラムが
仕様を満たしているかテスト

簡単な方法：単体テスト

```
# test_matrix.py
A = [[0, -1], [1, 0]]
B = [[1,  2], [3, 4]]
assert \
 matrix_mul(A, B) == [[-3,-4],[1,2]],\
 "テスト1失敗"
print("テスト1成功！")
```

```
$ python3 test_matrix.py
テスト1成功！
テスト2成功！
```

```
B’= [[5, 6], [7, 8]]
assert \
  matrix_mulmul(B, B’) == \
    [[19,22],[43,50]],\
  “テスト2失敗"
print(“テスト2成功！")
```

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

単体テストでプログラムが **仕様を満たしているかテスト**

簡単な方法：単体テスト

```
# test_matrix.py
A = [[0, -1], [1, 0]]
B = [[1,  2], [3, 4]]
assert \
 matrix_mul(A, B) == [[-3,-4],[1,2]],\
 "テスト1失敗"
print("テスト1成功！")
```

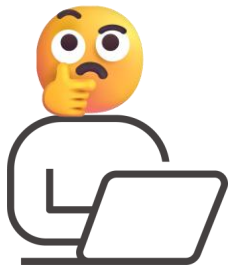
```
$ python3 test_matrix.py
テスト1成功！
テスト2成功！
```

```
B’= [[5, 6], [7, 8]]
assert \
  matrix_mulmul(B, B’) == \
    [[19,22],[43,50]],\
  “テスト2失敗"
print(“テスト2成功！")
```

A	B	$A \cdot B$
$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ 1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

一通り
実装完了
（？）

素朴な疑問①：もっと簡単に書けないか？



Strassenのアルゴリズム[Strassen'69]

出典: フリー百科事典『ウィキペディア（Wikipedia）』。 https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%A5%E3%83%88%E3%83%A9%E3%83%83%E3%82%BB%E3%83%B3%E3%81%AE%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0

もっと高速な
アルゴリズムを見つけた！

シュトラッセンのアルゴリズム（Strassen algorithm）は、$N \times N$の積を高速に計算するアルゴリズムである。通常の$N \times N$行列同士の積を計算するには$O(N^3)$の時間が必要だが、このアルゴリズムを用いると$O(N^{\log_2 7})$（$\log_2 7 \approx 2.807$）の時間で計算できる[1]。1969年、フォルカー・シュトラッセンが開発した[1][2]。

便宜上、Nを偶数と考えて、以下のように$\frac{N}{2} \times \frac{N}{2}$部分行列に分解する。

行列を分割し…

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

そして、以下の七つの行列をつくる。

同じCを得られる
ようなPを構成すると…

$$P_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$P_2 = (A_{21} + A_{22})B_{11}$$
$$P_3 = A_{11}(B_{12} - B_{22})$$
$$P_4 = A_{22}(B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{12})B_{22}$$
$$P_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$P_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

乗算＊7

掛け算
が1回減る

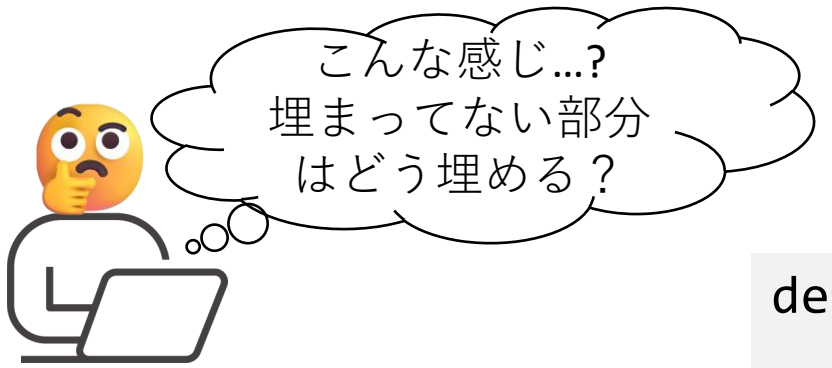
このとき、

$$C_{11} = P_1 + P_4 - P_5 + P_7$$
$$C_{12} = P_3 + P_5$$
$$C_{21} = P_2 + P_4$$
$$C_{22} = P_1 + P_3 - P_2 + P_6$$

の関係が成り立つ。

この関係を利用して計算すると、部分行列同士の乗算が、通常の方法では8回必要なのに、この方法では7回ですむようになり、計算時間が削減される。部分行列への分割を再帰的に行うことにより、さらに計算時間を削減することができる。

素朴な疑問①：もっと簡単に書けないか？




```
def strassen(A, B):
    …
    (AとBを分割する???)
    (Pを7つ作る???)
    …
    C = (PでC作る?)
    …
    return C?
```

Strassenのアルゴリズム[Strassen'69]

出典: フリー百科事典『ウィキペディア（Wikipedia）』 https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%A5%E3%83%88%E3%83%A9%E3%83%83%E3%82%BB%E3%83%B3%E3%81%AE%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0

シュトラッセンのアルゴリズム（Strassen algorithm）は、行列の積を高速に計算するアルゴリズムである。通常、$N \times N$行列同士の積を計算するには$O(N^3)$の時間が必要だが、このアルゴリズムを用いると、$O(N^{\log_2 7}) \approx O(N^{2.807})$の時間で計算できる[1]。1969年、フォルカー・シュトラッセンが開発した[1][2]。

便宜上、Nを偶数と考えて、以下のように$\frac{N}{2} \times \frac{N}{2}$部分行列に分解する。

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

そして、以下の七つの行列をつくる。

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

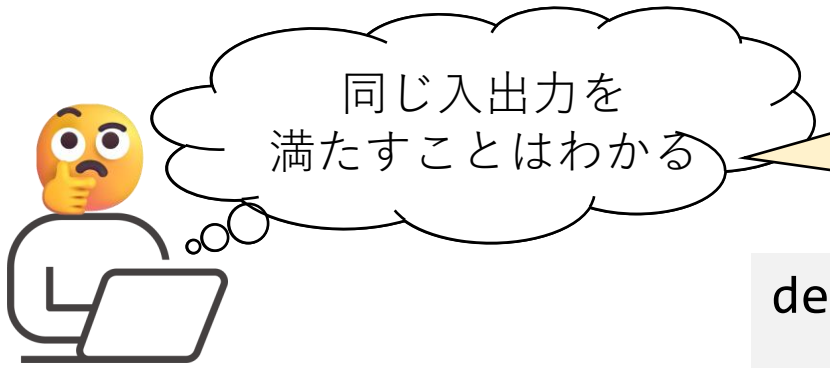
このとき、

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned}$$

の関係が成り立つ。

この関係を利用して計算すると、部分行列同士の乗算が、通常の方法では8回必要なのに、この方法では7回ですむようになり、計算時間が削減される。部分行列への分割を再帰的に行うことにより、さらに計算時間を削減することができる。

素朴な疑問①：もっと簡単に書けないか？



A	B	$A \cdot B$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}0 & -1\\1 & 0\end{pmatrix}$	$\begin{pmatrix}2 & -1\\4 & -3\end{pmatrix}$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}$	$\begin{pmatrix}19 & 22\\43 & 50\end{pmatrix}$
⋮	⋮	⋮

```
def matrix_multiplication(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # 行列Cの要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k] * B[k][j]
  return C
```

3重ループの実装

```
def strassen(A, B):
    …
    (AとBを分割する???)
    (Pを7つ作る???)
    …
    C = (PでC作る?)
    …
    return C?
```

Strassenのアルゴリズム[Strassen'69]

出典: フリー百科事典『ウィキペディア（Wikipedia）』 https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%A5%E3%83%88%E3%83%A9%E3%83%83%E3%82%BB%E3%83%B3%E3%81%AE%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0

シュトラッセンのアルゴリズム（Strassen algorithm）は、行列の積を高速に計算するアルゴリズムである。通常、$N \times N$行列同士の積を計算するには$O(N^3)$の時間が必要だが、このアルゴリズムを用いると、$O(N^{\log_2 7}) \approx O(N^{2.807})$の時間で計算できる[1]。1969年、フォルカー・シュトラッセンが開発した[1][2]。

便宜上、Nを偶数と考えて、以下のように$\frac{N}{2} \times \frac{N}{2}$部分行列に分解する。

$$\begin{pmatrix}C_{11} & C_{12}\\C_{21} & C_{22}\end{pmatrix} = \begin{pmatrix}A_{11} & A_{12}\\A_{21} & A_{22}\end{pmatrix}\begin{pmatrix}B_{11} & B_{12}\\B_{21} & B_{22}\end{pmatrix}$$

そして、以下の七つの行列をつくる。

$$\begin{aligned}P_1 &= (A_{11} + A_{22})(B_{11} + B_{22})\\P_2 &= (A_{21} + A_{22})B_{11}\\P_3 &= A_{11}(B_{12} - B_{22})\\P_4 &= A_{22}(B_{21} - B_{11})\\P_5 &= (A_{11} + A_{12})B_{22}\\P_6 &= (A_{21} - A_{11})(B_{11} + B_{12})\\P_7 &= (A_{12} - A_{22})(B_{21} + B_{22})\end{aligned}$$

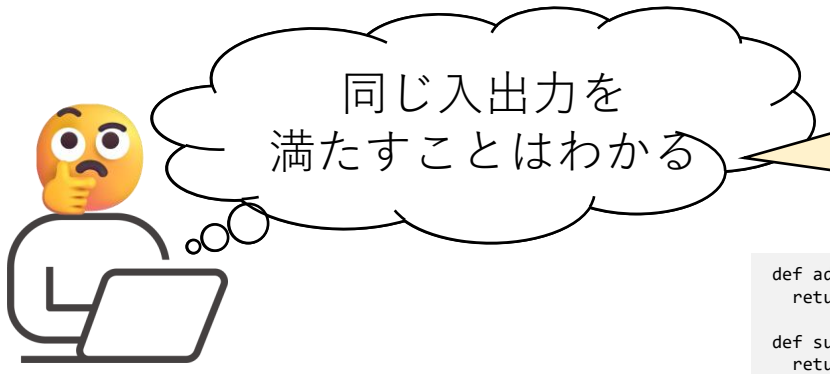
このとき、

$$\begin{aligned}C_{11} &= P_1 + P_4 - P_5 + P_7\\C_{12} &= P_3 + P_5\\C_{21} &= P_2 + P_4\\C_{22} &= P_1 + P_3 - P_2 + P_6\end{aligned}$$

の関係が成り立つ。

この関係を利用して計算すると、部分行列同士の乗算が、通常の方法では8回必要なのに、この方法では7回ですむようになり、計算時間が削減される。部分行列への分割を再帰的に行うことにより、さらに計算時間を削減することができる。

素朴な疑問①：もっと簡単に書けないか？



A	B	$A \cdot B$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}0 & -1\\1 & 0\end{pmatrix}$	$\begin{pmatrix}2 & -1\\4 & -3\end{pmatrix}$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}$	$\begin{pmatrix}19 & 22\\43 & 50\end{pmatrix}$
⋮	⋮	⋮

```
def matrix_multiplication(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = …行列Cの要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k] * B[k][j]
  return C
```

3重ループの実装

```
def add(A, B):
  return [ [a + b for a, b in zip(r1, r2)]
           for r1, r2 in zip(A, B) ]
def sub(A, B):
  return [ [a - b for a, b in zip(r1, r2)]
           for r1, r2 in zip(A, B) ]
def split(M):
  m = len(M) // 2
  return [ [row[:m] for row in M[:m]],
           [row[m:] for row in M[:m]],
           [row[:m] for row in M[m:]],
           [row[m:] for row in M[m:]] ]

def join(C11, C12, C21, C22):
  top = [r1 + r2 for r1, r2 in zip(C11, C12)]
  bot = [r1 + r2 for r1, r2 in zip(C21, C22)]
  return top + bot

def strassen(A, B):
  if len(A) == 1: return [[A[0][0] * B[0][0]]]
  A11, A12, A21, A22 = split(A)
  B11, B12, B21, B22 = split(B)
  P1 = strassen(add(A11, A22), add(B11, B22))
  P2 = strassen(add(A21, A22), B11)
  P3 = strassen(A11, sub(B12, B22))
  P4 = strassen(A22, sub(B21, B11))
  P5 = strassen(add(A11, A12), B22)
  P6 = strassen(sub(A21, A11), add(B11, B12))
  P7 = strassen(sub(A12, A22), add(B21, B22))
  C11 = add(sub(add(P1, P4), P5), P7)
  C12 = add(P3, P5)
  C21 = add(P2, P4)
  C22 = add(sub(add(P1, P3), P2), P6)
  return join(C11, C12, C21, C22)
```

大変

Strassenのアルゴリズム[Strassen'69]

出典: フリー百科事典『ウィキペディア（Wikipedia）』 https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%A5%E3%83%88%E3%83%A9%E3%83%83%E3%82%BB%E3%83%B3%E3%81%AE%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0

シュトラッセンのアルゴリズム（Strassen algorithm）は、行列の積を高速に計算するアルゴリズムである。通常、$N \times N$行列同士の積を計算するには$O(N^3)$の時間が必要だが、このアルゴリズムを用いると、$O(N^{\log_2 7}) \approx O(N^{2.807})$の時間で計算できる[1]。1969年、フォルカー・シュトラッセンが開発した[1][2]。

便宜上、Nを偶数と考えて、以下のように$\frac{N}{2} \times \frac{N}{2}$部分行列に分解する。

$$\begin{pmatrix}C_{11} & C_{12}\\C_{21} & C_{22}\end{pmatrix} = \begin{pmatrix}A_{11} & A_{12}\\A_{21} & A_{22}\end{pmatrix}\begin{pmatrix}B_{11} & B_{12}\\B_{21} & B_{22}\end{pmatrix}$$

そして、以下の七つの行列をつくる。

$$\begin{aligned}P_1 &= (A_{11} + A_{22})(B_{11} + B_{22})\\P_2 &= (A_{21} + A_{22})B_{11}\\P_3 &= A_{11}(B_{12} - B_{22})\\P_4 &= A_{22}(B_{21} - B_{11})\\P_5 &= (A_{11} + A_{12})B_{22}\\P_6 &= (A_{21} - A_{11})(B_{11} + B_{12})\\P_7 &= (A_{12} - A_{22})(B_{21} + B_{22})\end{aligned}$$

このとき、

$$\begin{aligned}C_{11} &= P_1 + P_4 - P_5 + P_7\\C_{12} &= P_3 + P_5\\C_{21} &= P_2 + P_4\\C_{22} &= P_1 + P_3 - P_2 + P_6\end{aligned}$$

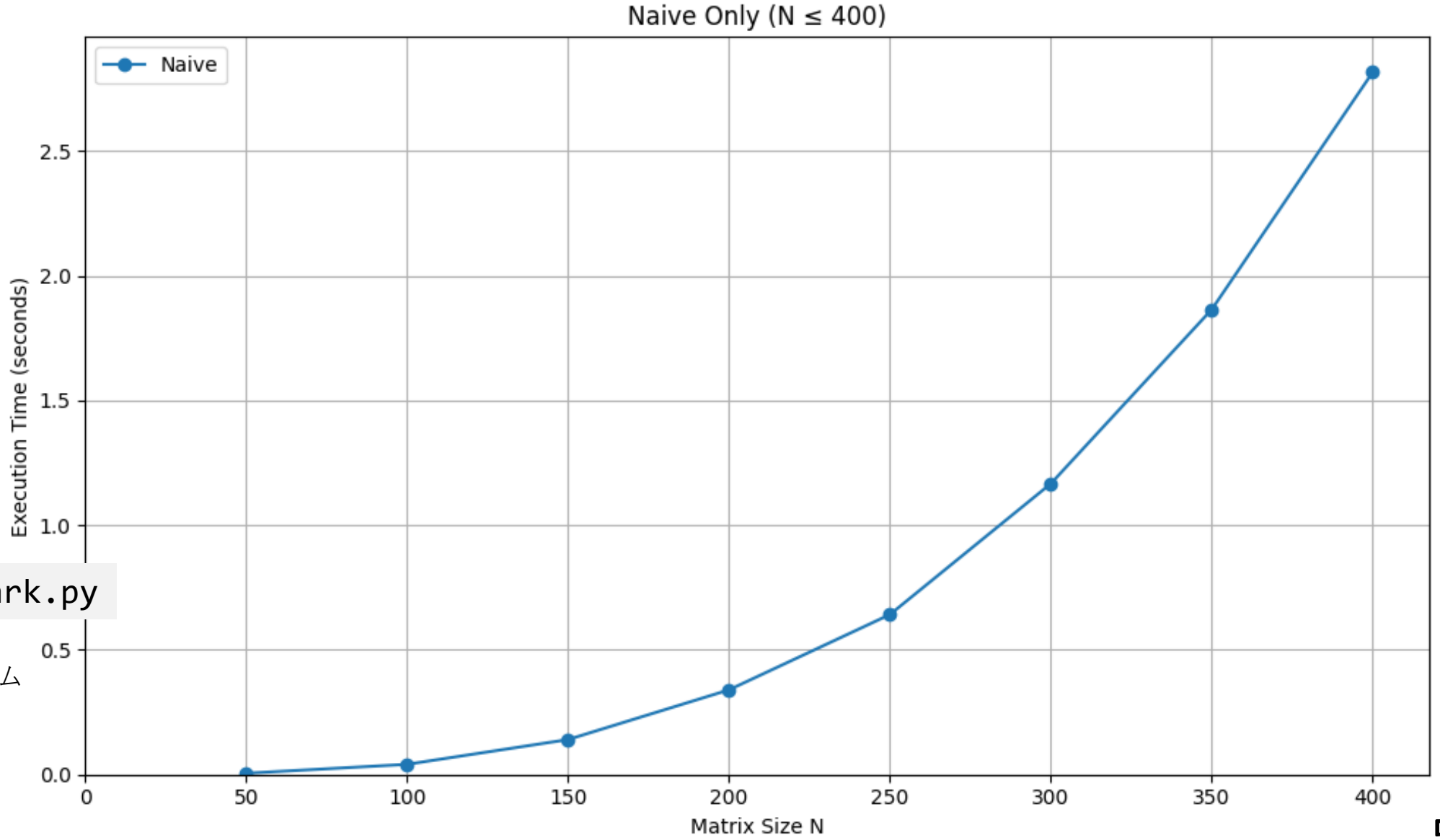
の関係が成り立つ。

この関係を利用して計算すると、部分行列同士の乗算が、通常の方法では8回必要なのに、この方法では7回ですむようになり、計算時間が削減される。部分行列への分割を再帰的に行うことにより、さらに計算時間を削減することができる。

素朴な疑問②：効率的に処理できてる？

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0]*n for _ in range(l)]  # C の要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装



Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3

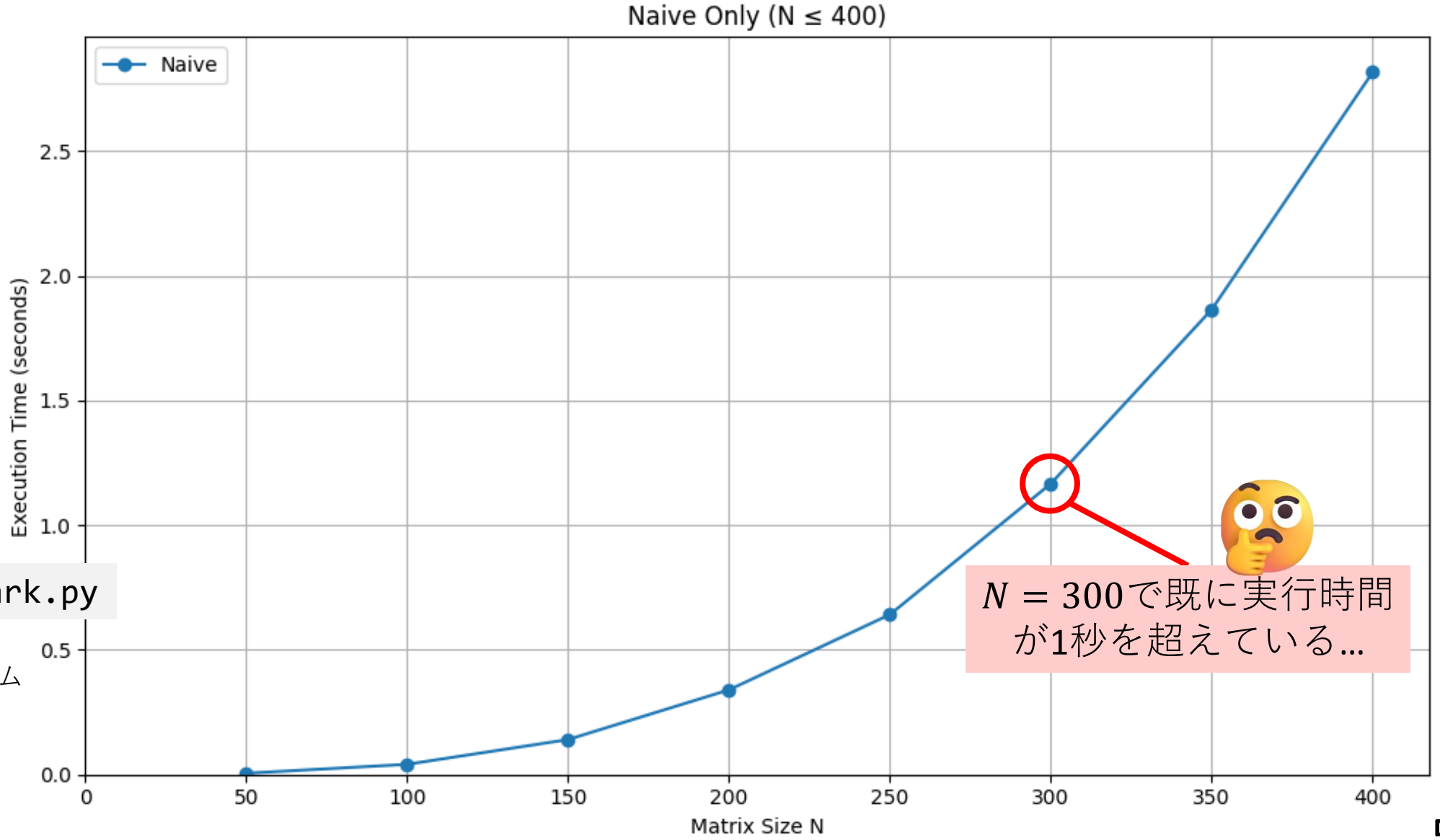
```
$ python benchmark.py
```

入力AとB：
浮動小数点の一様ランダム
バイナリ$N \times N$行列

素朴な疑問②：効率的に処理できてる？

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0]*n for _ in range(l)]  # C の要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装



Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3

```
$ python benchmark.py
```

入力AとB：
浮動小数点の一様ランダム
バイナリ$N \times N$行列

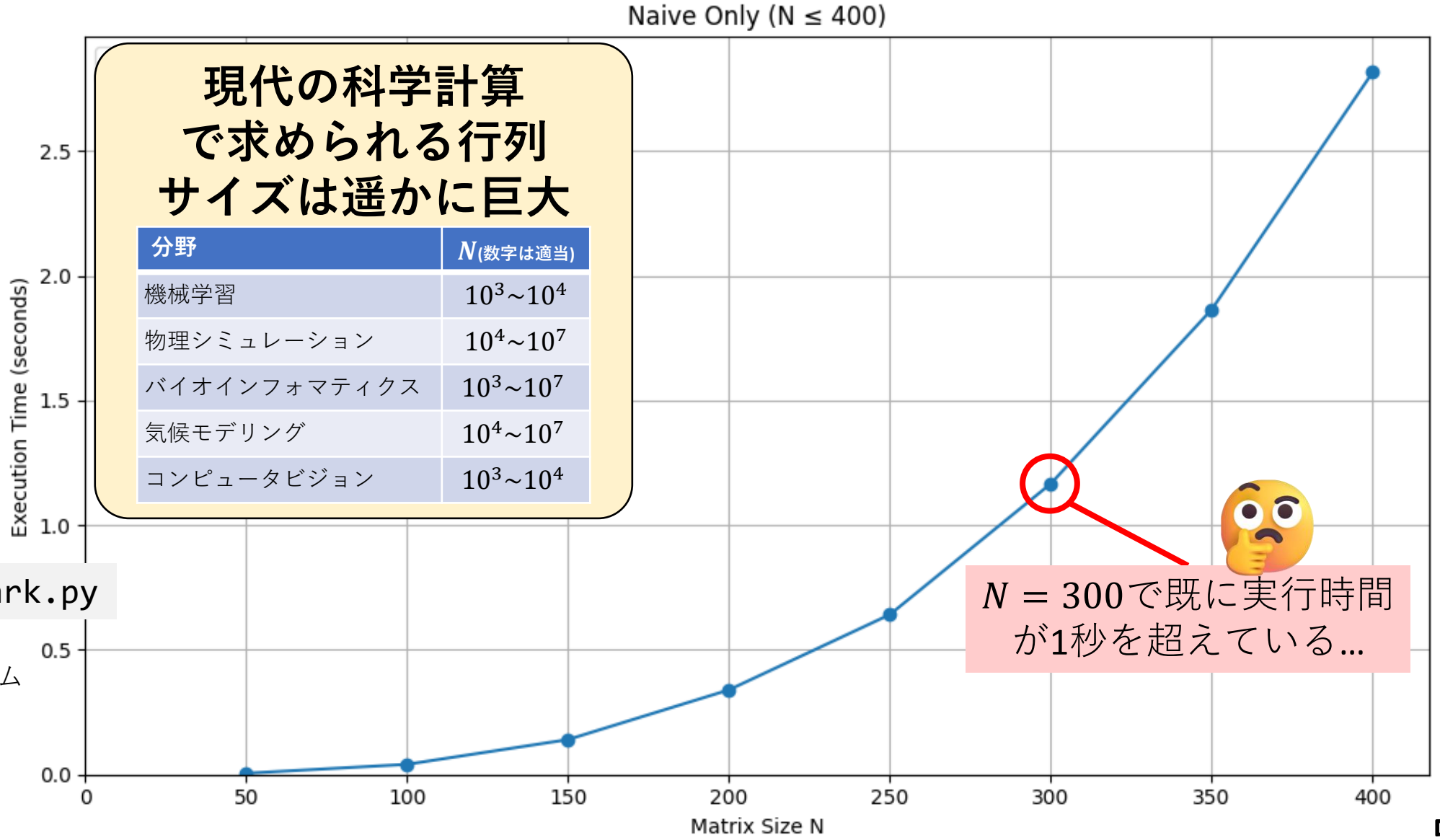
素朴な疑問②：効率的に処理できてる？

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0]*n for _ in range(l)]  # Cの要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

現代の科学計算で求められる行列サイズは遥かに巨大

分野	N(数字は適当)
機械学習	10^3~10^4
物理シミュレーション	10^4~10^7
バイオインフォマティクス	10^3~10^7
気候モデリング	10^4~10^7
コンピュータビジョン	10^3~10^4



$N = 300$で既に実行時間が1秒を超えている…

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3

```
$ python benchmark.py
```

入力AとB：
浮動小数点の一様ランダム
バイナリ$N \times N$行列

素朴な疑問②：効率的に処理できてる？

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [0]*l*n  # C の要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

現代の科学計算で求められる行列サイズは遥かに巨大

分野	N(数字は適当)
機械学習	10^3~10^4
物理シミュレーション	10^4~10^7
バイオインフォマティクス	10^3~10^7
気候モデリング	10^4~10^7
コンピュータビジョン	10^3~10^4

極端な例

Amazon Web Services ブログ

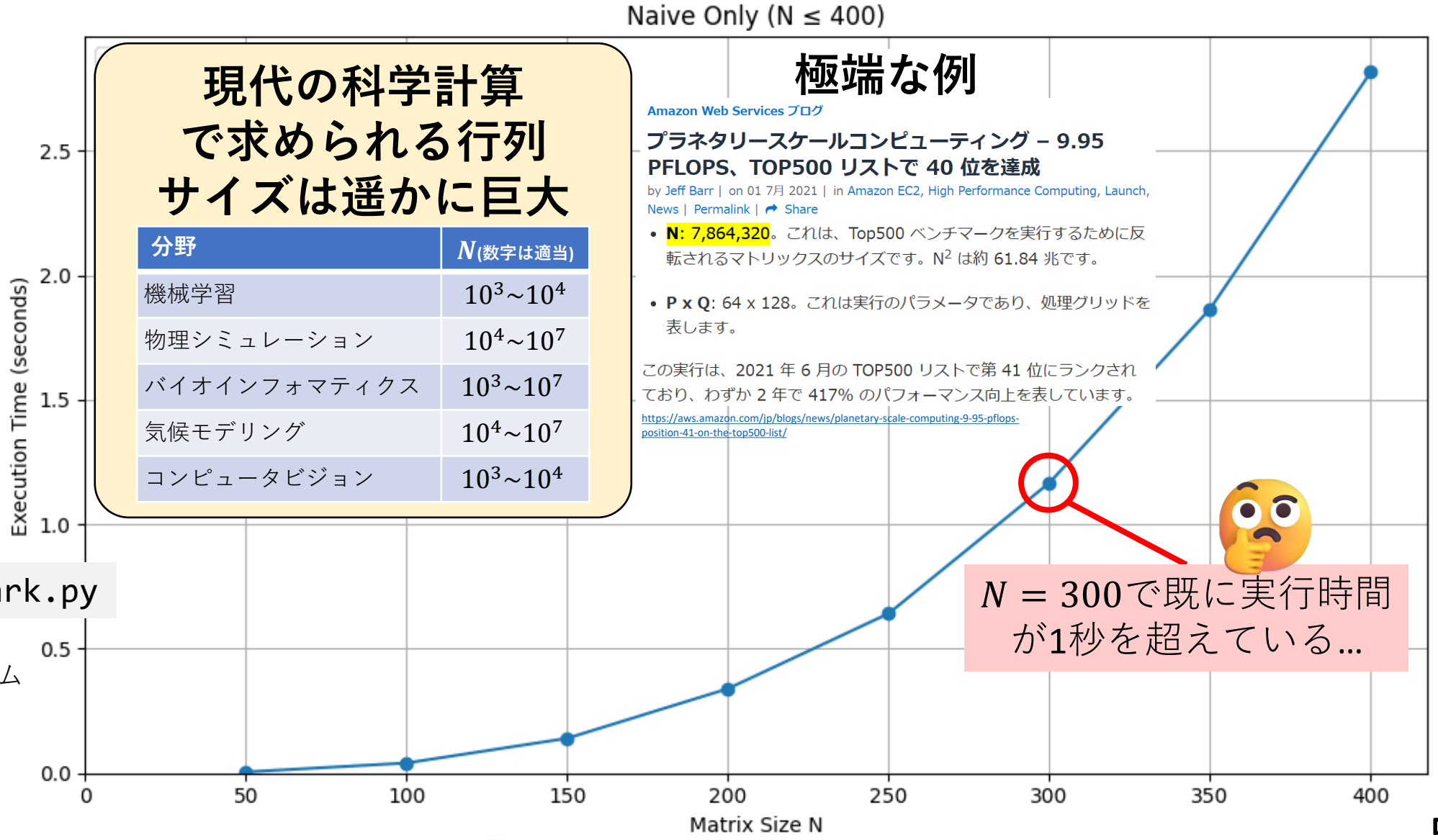
プラネタリースケールコンピューティング – 9.95 PFLOPS、TOP500 リストで 40 位を達成

by Jeff Barr | on 01 7月 2021 | in Amazon EC2, High Performance Computing, Launch, News | Permalink | Share

- **N**: 7,864,320。これは、Top500 ベンチマークを実行するために反転されるマトリックスのサイズです。N^2 は約 61.84 兆です。
- **P x Q**: 64 x 128。これは実行のパラメータであり、処理グリッドを表します。

この実行は、2021 年 6 月の TOP500 リストで第 41 位にランクされており、わずか 2 年で 417% のパフォーマンス向上を表しています。

https://aws.amazon.com/jp/blogs/news/planetary-scale-computing-9-95-pflops-position-41-on-the-top500-list/



$N = 300$で既に実行時間が1秒を超えている…

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3

```
$ python benchmark.py
```

入力AとB：
浮動小数点の一様ランダム
バイナリ$N \times N$行列

素朴な疑問③: プログラムは“**正しい**”？

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj} \right)$$

単体テスト

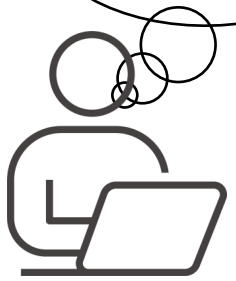
```
# test_matrix.py
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
assert \
 matmul(A, B) == [[19,22],[43,50]],\
 "テスト1失敗"
print("テスト1成功！")
…
```

```
$ python3 test_matrix.py
テスト1成功！
…
```

素朴な疑問③: プログラムは“**正しい**”？

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$



単体テスト

```
# test_matrix.py
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
assert \
 matmul(A, B) == [[19,22],[43,50]],\
 "テスト1失敗"
print("テスト1成功！")
…
```

```
$ python3 test_matrix.py
テスト1成功！
…
```

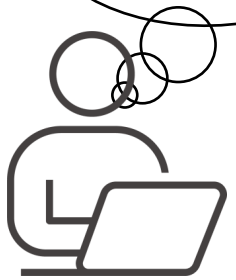
いくつかの例で
手計算と一致する
ことは確かめた。

A	B	$A \cdot B$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}$	$\begin{pmatrix}19 & 22\\43 & 50\end{pmatrix}$
⋮	⋮	⋮

素朴な疑問③: プログラムは“**正しい**”？

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$



単体テスト

```
# test_matrix.py
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
assert \
 matmul(A, B) == [[19,22],[43,50]],\
 "テスト1失敗"
print("テスト1成功！")
…
```

```
$ python3 test_matrix.py
テスト1成功！
…
```

いくつかの例で
手計算と一致する
ことは確かめた。

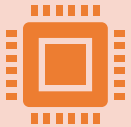
常にmatmalが③を満たすと
を確信するにはどうすれば？

A	B	$A \cdot B$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}$	$\begin{pmatrix}19 & 22\\43 & 50\end{pmatrix}$
⋮	⋮	⋮

プログラミングを**科学**する

表現する、処理する、証明する

表現する：人間の意図や仕様を計算機に伝える方法



処理する：プログラムを自動で最適化する方法

証明する：プログラムが正しく動作することを検証する方法

プログラミングを**科学**する

表現する、処理する、証明する

表現する：人間の意図や仕様を計算機に伝える方法

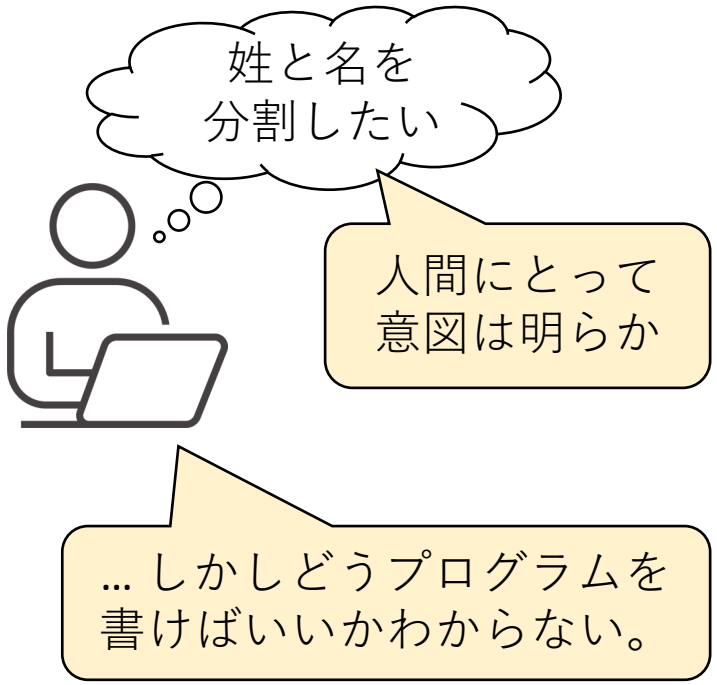
処理する：プログラムを自動で最適化する方法

証明する：プログラムが正しく動作することを検証する方法

動機：非プログラマーも自動化したい！

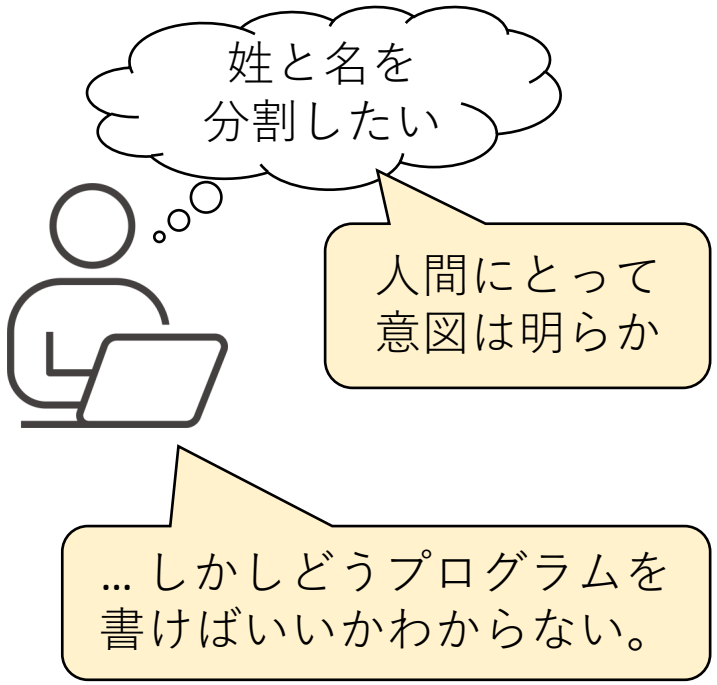
	A	B	C	D
1	Name	First	Last	
2	Yudai Tanabe	Yudai	Tanabe	
3	Taro Science			
4	Hanako Tokyo			
5	Ichiro Tanaka			
6	Yuki Sato			
7	Haruka Kobayashi			
8	Kenta Watanabe			
9	Aiko Yamamoto			
10	Daiki Nakamura			
11	Emiko Takahashi			
12	Shota Ito			
13	Mika Kondo			
14	Riku Fujita			
15	Ayaka Maeda			
16	Kazuya Inoue			
17	Reina Shimizu			

動機：非プログラマーも自動化したい！



	A	B	C	D
1	Name	First	Last	
2	Yudai Tanabe	Yudai	Tanabe	
3	Taro Science			
4	Hanako Tokyo			
5	Ichiro Tanaka			
6	Yuki Sato			
7	Haruka Kobayashi			
8	Kenta Watanabe			
9	Aiko Yamamoto			
10	Daiki Nakamura			
11	Emiko Takahashi			
12	Shota Ito			
13	Mika Kondo			
14	Riku Fujita			
15	Ayaka Maeda			
16	Kazuya Inoue			
17	Reina Shimizu			

動機：非プログラマーも自動化したい！



	A	B	C	D
1	Name	First	Last	
2	Yudai Tanabe	Yudai	Tanabe	
3	Taro Science			
4	Hanako Tokyo			
5	Ichiro Tanaka			
6	Yuki Sato			
7	Haruka Kobayashi			
8	Kenta Watanabe			
9	Aiko Yamamoto			
10	Daiki Nakamura			
11	Emiko Takahashi			
12	Shota Ito			
13	Mika Kondo			
14	Riku Fujita			
15	Ayaka Maeda			
16	Kazuya Inoue			
17	Reina Shimizu			

解: Excelの関数による実装

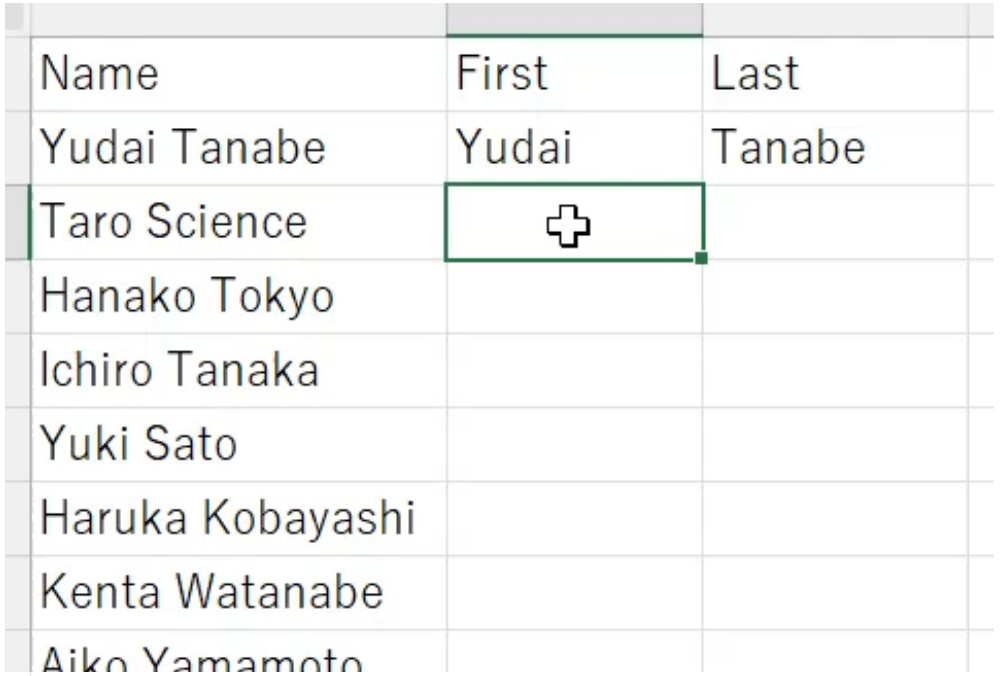
```
=BYROW(A2:A17,
   LAMBDA(a,
      LEFT(a, FIND(" ", a) - 1)))
=BYROW(A2:A17,
   LAMBDA(a,
      RIGHT(a, LEN(a)-FIND(" ", a))))
```

FlashFill[Gulwani et al., POPL'11]

FlashFill[Gulwani et al., POPL'11]

Name	First	Last
Yudai Tanabe	Yudai	Tanabe
Taro Science		
Hanako Tokyo		
Ichiro Tanaka		
Yuki Sato		
Haruka Kobayashi		
Kenta Watanabe		
Aiko Yamamoto		
Daiki Nakamura		
Emiko Takahashi		
Shota Ito		
Mika Kondo		
Riku Fujita		
Ayaka Maeda		
Kazuya Inoue		
Reina Shimizu		

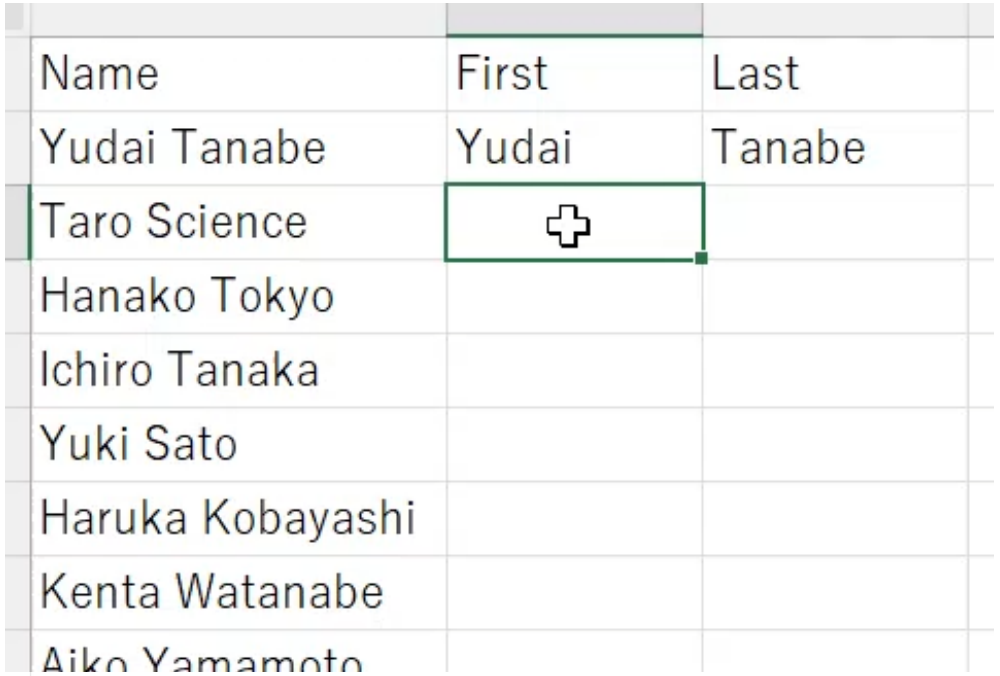
FlashFill[Gulwani et al., POPL'11]



Name	First	Last
Yudai Tanabe	Yudai	Tanabe
Taro Science		
Hanako Tokyo		
Ichiro Tanaka		
Yuki Sato		
Haruka Kobayashi		
Kenta Watanabe		
Aiko Yamamoto		

Date	Time	Organizer	Title	Message
2025/08/02	9:00	Alice Johnson	Project Kickoff	[Aug 2, 9:00] Project Kickoff by Johnson
2025/08/03	15:30	Bob Smith	Weekly Sync	[Aug 3, 15:30] Weekly Sync by Smith
2025/08/04	9:00	Charlie Tanaka	Client Presentation	
2025/08/05	14:15	Alice Johnson	Internal Design Review	
2025/08/06	19:30	Yudai Tanabe	Job Interview	
2025/08/07	0:45	Charlie Tanaka	Internal Design Review	
2025/08/08	6:00	Yudai Tanabe	Client Presentation	
2025/08/09	11:15	Alice Johnson	Client Presentation	

FlashFill[Gulwani et al., POPL'11]

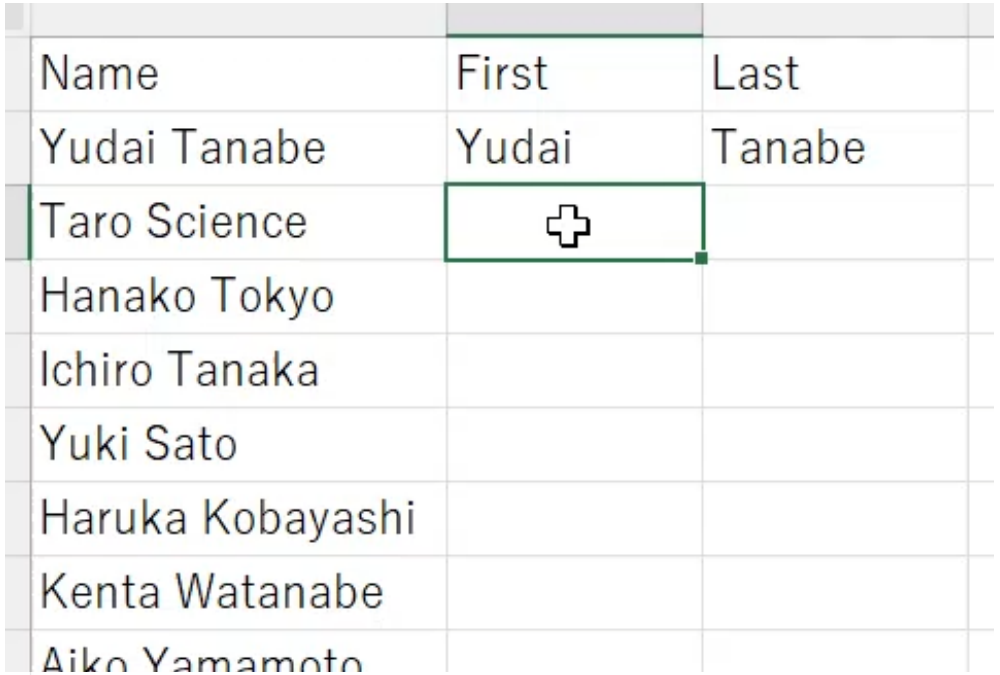


Name	First	Last
Yudai Tanabe	Yudai	Tanabe
Taro Science		
Hanako Tokyo		
Ichiro Tanaka		
Yuki Sato		
Haruka Kobayashi		
Kenta Watanabe		
Aiko Yamamoto		

Date	Time	Organizer	Title	Message
2025/08/02	9:00	Alice Johnson	Project Kickoff	[Aug 2, 9:00] Project Kickoff by Johnson
2025/08/03	15:30	Bob Smith	Weekly Sync	[Aug 3, 15:30] Weekly Sync by Smith
2025/08/04	9:00	Charlie Tanaka	Client Presentation	
2025/08/05	14:15	Alice Johnson	Internal Design Review	
2025/08/06	19:30	Yudai Tanabe	Job Interview	
2025/08/07	0:45	Charlie Tanaka	Internal Design Review	
2025/08/08	6:00	Yudai Tanabe	Client Presentation	
2025/08/09	11:15	Alice Johnson	Client Presentation	

FlashFill[Gulwani et al., POPL'11]

アイディア
入出力例から**プログラム**
を自動発見する



Name	First	Last
Yudai Tanabe	Yudai	Tanabe
Taro Science		
Hanako Tokyo		
Ichiro Tanaka		
Yuki Sato		
Haruka Kobayashi		
Kenta Watanabe		
Aiko Yamamoto		

Date	Time	Organizer	Title	Message
2025/08/02	9:00	Alice Johnson	Project Kickoff	[Aug 2, 9:00] Project Kickoff by Johnson
2025/08/03	15:30	Bob Smith	Weekly Sync	[Aug 3, 15:30] Weekly Sync by Smith
2025/08/04	9:00	Charlie Tanaka	Client Presentation	
2025/08/05	14:15	Alice Johnson	Internal Design Review	
2025/08/06	19:30	Yudai Tanabe	Job Interview	
2025/08/07	0:45	Charlie Tanaka	Internal Design Review	
2025/08/08	6:00	Yudai Tanabe	Client Presentation	
2025/08/09	11:15	Alice Johnson	Client Presentation	

FlashFillの中身 ー プログラム合成

仕様：入出力

入力	出力
“Yudai Tanabe”	**“Yudai”**
“Taro Science”	?

Ctrl + E

FlashFill

FlashFillの中身 ー プログラム合成

仕様：入出力

入力	出力
“Yudai Tanabe”	**“Yudai”**
“Taro Science”	?

Ctrl + E

FlashFill

仕様を満たす
プログラム

発見！

$$Sub(Input, 0, pos(input, "", " ", 1))$$

or

時間切れ

FlashFillの中身 ー プログラム合成

仕様：入出力

入力	出力
“Yudai Tanabe”	**“Yudai”**
“Taro Science”	?

Ctrl + E

FlashFill

仕様を満たす
プログラム

発見！

$Sub(Input, 0, pos(input, "", " ", 1))$

or

時間切れ

自動
適用

入力	出力
“Yudai Tanabe”	“Yudai”
“Taro Science”	**“Taro”**

FlashFillの中身 ー プログラム合成

仕様：入出力

入力	出力
“Yudai Tanabe”	**“Yudai”**
“Taro Science”	?

Ctrl + E

FlashFill

プログラマは
入出力を書くだけ

仕様を満たす
プログラム

発見！

$Sub(Input, 0, pos(input, "", " ", 1))$

or

時間切れ

自動
適用

入力	出力
“Yudai Tanabe”	“Yudai”
“Taro Science”	**“Taro”**

FlashFillの奥義：
Excelの問題を記述する**簡潔な表現**の発見

FlashFill

FlashFillの奥義：
Excelの問題を記述する**簡潔な表現**の発見

FlashFill

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \ ::= \ & Input \\
& \mid Str\ (\text{“}\ldots\text{”}) \\
& \mid Concat(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid Sub\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid (\text{他多数のExcel関数})
\end{aligned}
$$

FlashFillの奥義：Excelの問題を記述する**簡潔な表現**の発見

FlashFill

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \ ::= \ & Input \\
& \mid Str\ (\text{“}\ldots\text{”}) \\
& \mid Concat(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid Sub\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid (\text{他多数のExcel関数})
\end{aligned}
$$

Step 2.
候補の生成と評価

候補プログラム	計算結果
$Sub(Input, 0, 5)$	“Yudai”
$Concat$ $\quad Sub(Input, 0, 1)$ $\quad Const(\text{"さん"})$	“Yさん”

FlashFillの奥義：
Excelの問題を記述する**簡潔な表現**の発見

FlashFill

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \ ::= \ & Input \\
& \mid Str\ (\text{“ ...”}) \\
& \mid Concat(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid Sub\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid (\text{他多数のExcel関数})
\end{aligned}
$$

Step 2.
候補の生成と評価

候補プログラム	計算結果
$Sub(Input, 0, 5)$	“Yudai” ✅
$Concat$ $Sub(Input, 0, 1)$ $Const(\text{"さん"})$	“Yさん” ❌

Step 3.
入出力一致？

期待の出力
“Yudai”

FlashFillの奥義：Excelの問題を記述する**簡潔な表現**の発見

FlashFill

課題
現実的な時間で終わらない

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \ ::= \ & Input \\
& \mid Str\ (\text{“ ...”}) \\
& \mid Concat(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid Sub\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid (\text{他多数のExcel関数})
\end{aligned}
$$

Step 2.
候補の生成と評価

候補プログラム	計算結果
$Sub(Input, 0, 5)$	“Yudai” ✅
$Concat$ $\quad Sub(Input, 0, 1)$ $\quad Const(\text{"さん"})$	“Yさん” ❌

Step 3.
入出力一致？

期待の出力
“Yudai”

FlashFillの奥義：Excelの問題を記述する**簡潔な表現**の発見

FlashFill

課題
現実的な時間で終わらない

解決
Excelフォーラムの観察から
必要な機能を同定

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \ ::= \ & Input \\
& \mid Str\ (\text{“ ...”}) \\
& \mid Concat(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid Sub\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid (\text{他多数のExcel関数})
\end{aligned}
$$

Step 2.
候補の生成と評価

候補プログラム	計算結果
$Sub(Input, 0, 5)$	“Yudai” ✅
$Concat$ $Sub(Input, 0, 1)$ $Const(\text{"さん"})$	“Yさん” ❌

Step 3.
入出力一致？

期待の出力
“Yudai”

FlashFillの奥義：Excelの問題を記述する**簡潔な表現**の発見

FlashFill

課題
現実的な時間で終わらない

解決
Excelフォーラムの観察から
必要な機能を同定

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \;::=\; & \mathit{Input} \\
& \mid \mathit{Str}\ (\text{“}\ldots\text{”}) \\
& \mid \mathit{Concat}(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid \mathit{Sub}\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid \sout{\text{(他多数のExcel関数)}}
\end{aligned}
$$

Step 2.
候補の生成と評価

候補プログラム	計算結果
$\mathit{Sub}(\mathit{Input}, 0, 5)$	“Yudai” ✅
Concat $\quad \mathit{Sub}(\mathit{Input}, 0, 1)$ $\quad \mathit{Const}(\text{"さん"})$	“Yさん” ✕

Step 3.
入出力一致？

期待の出力
“Yudai”

洞察①文字列処理特化で十分機能する

FlashFillの奥義：Excelの問題を記述する**簡潔な表現**の発見

FlashFill

課題
現実的な時間で終わらない

解決
Excelフォーラムの観察から
必要な機能を同定

Step 1.
探索範囲を定義

$$
\begin{aligned}
\boldsymbol{e} \;::=\; & \mathit{Input} \\
& \mid \mathit{Str}\ (\text{“}\ldots\text{”}) \\
& \mid \mathit{Concat}(\boldsymbol{e_1}, \boldsymbol{e_2}) \\
& \mid \mathit{Sub}\ (\boldsymbol{e}, pos_1, pos_2) \\
& \mid \sout{(他多数のExcel関数)}
\end{aligned}
$$

Step 2.
候補の生成と評価

候補プログラム	計算結果
$\mathit{Sub}(\mathit{Input}, 0, 5)$	“Yudai” ✅
Concat $\mathit{Sub}(\mathit{Input}, 0, 1)$ $\mathit{Const}(\text{"さん"})$	“Yさん” ❌

Step 3.
入出力一致？

期待の出力
“Yudai”

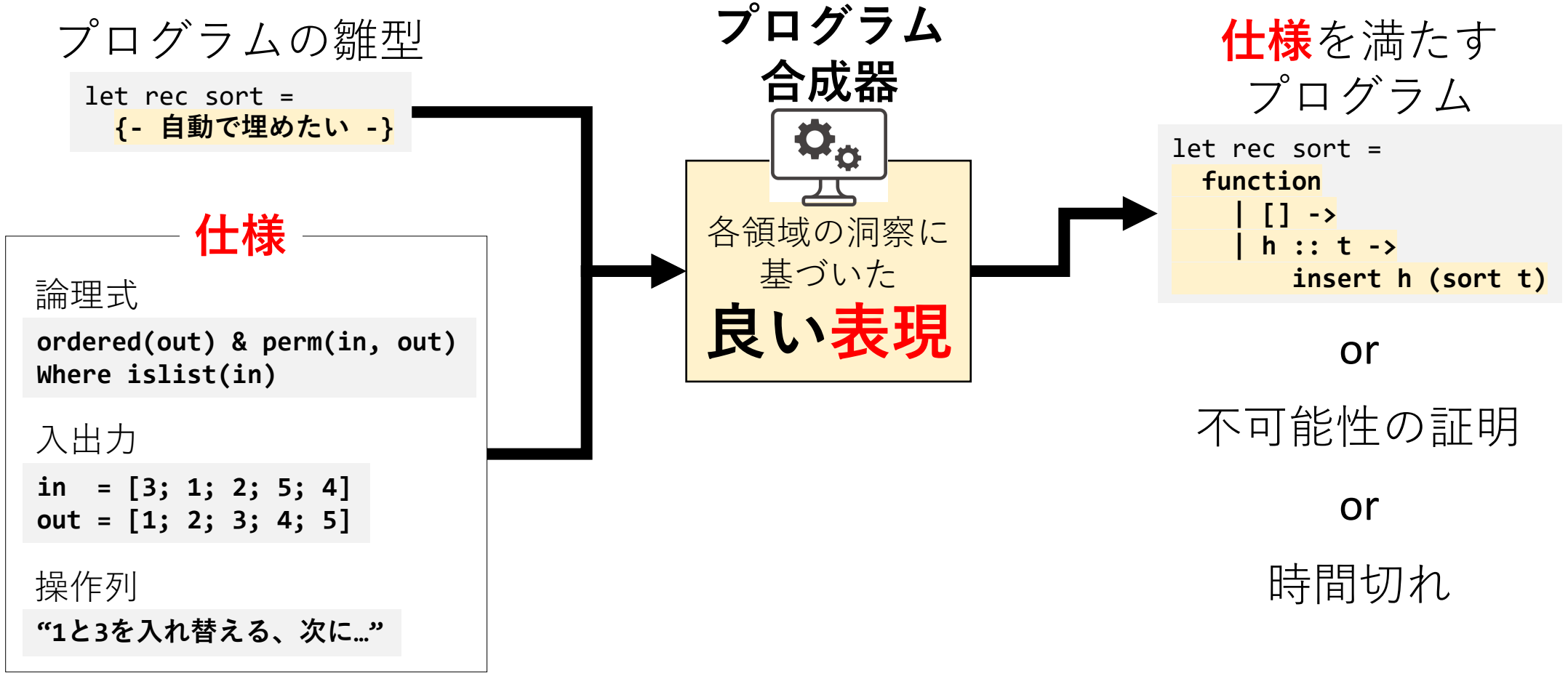
洞察①文字列処理特化で十分機能する

洞察②Excelで頻出の処理を優先探索

- 定数文字列は少ない
- プログラムは短い
- Subの後Concatが頻出

etc.

プログラム合成の精神：
良い表現は自動化の恩恵を民主化する



プログラム合成の精神：
良い表現は自動化の恩恵を民主化する

プログラムの雛型

```
let rec sort =
  {- 自動で埋めたい -}
```

仕様

論理式

```
ordered(out) & perm(in, out)
Where islist(in)
```

入出力

```
in  = [3; 1; 2; 5; 4]
out = [1; 2; 3; 4; 5]
```

操作列

```
“1と3を入れ替える、次に…”
```

プログラム合成器

各領域の洞察に基づいた
良い表現

仕様を満たすプログラム

```
let rec sort =
  function
    | [] ->
    | h :: t ->
        insert h (sort t)
```

or

不可能性の証明

or

時間切れ

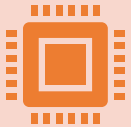
非専門家でも自動化可能に

- コンパイラ最適化
- データベースクエリの自動合成
- 自動デバッグとコード修復
- ロボット制御への応用

プログラミングを**科学**する

表現する、処理する、証明する

表現する：人間の意図や仕様を計算機に伝える方法



処理する：プログラムを自動で最適化する方法

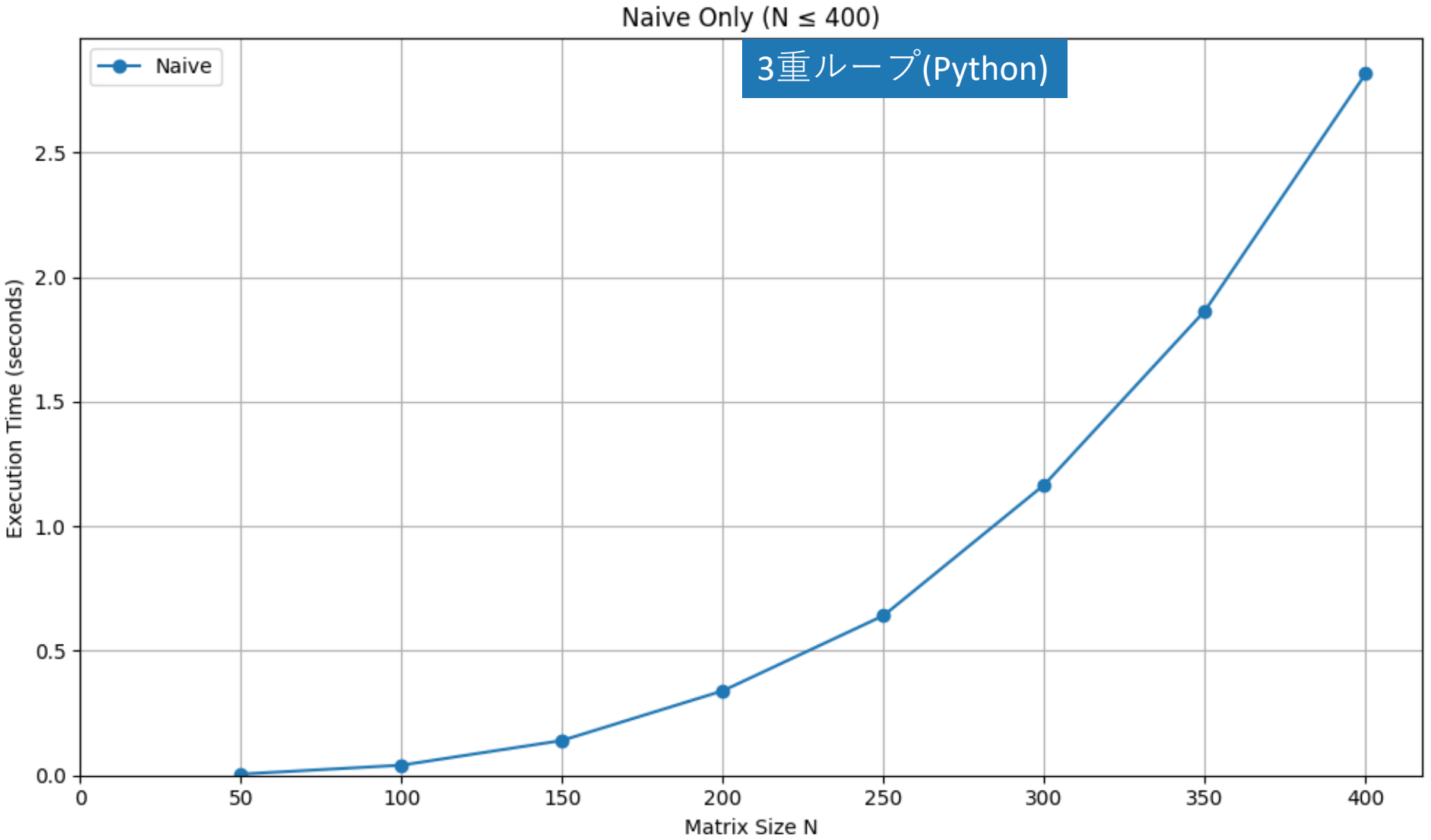
証明する：プログラムが正しく動作することを検証する方法

動機：もっと高速化できないか？

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0] * n for _ in range(l)]
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

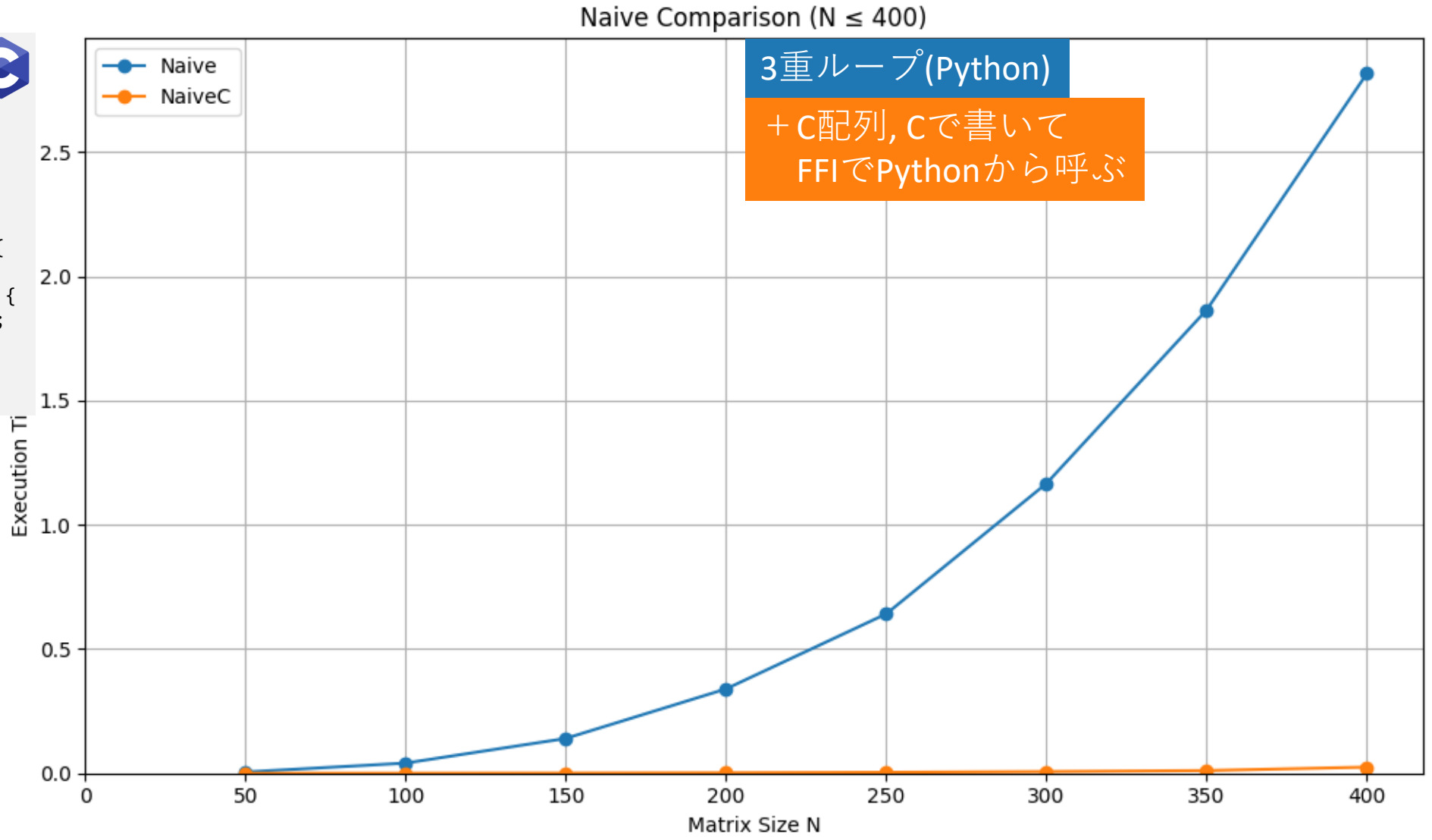
3重ループの実装



動機：もっと高速化できないか？

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
gcc 13.3.0

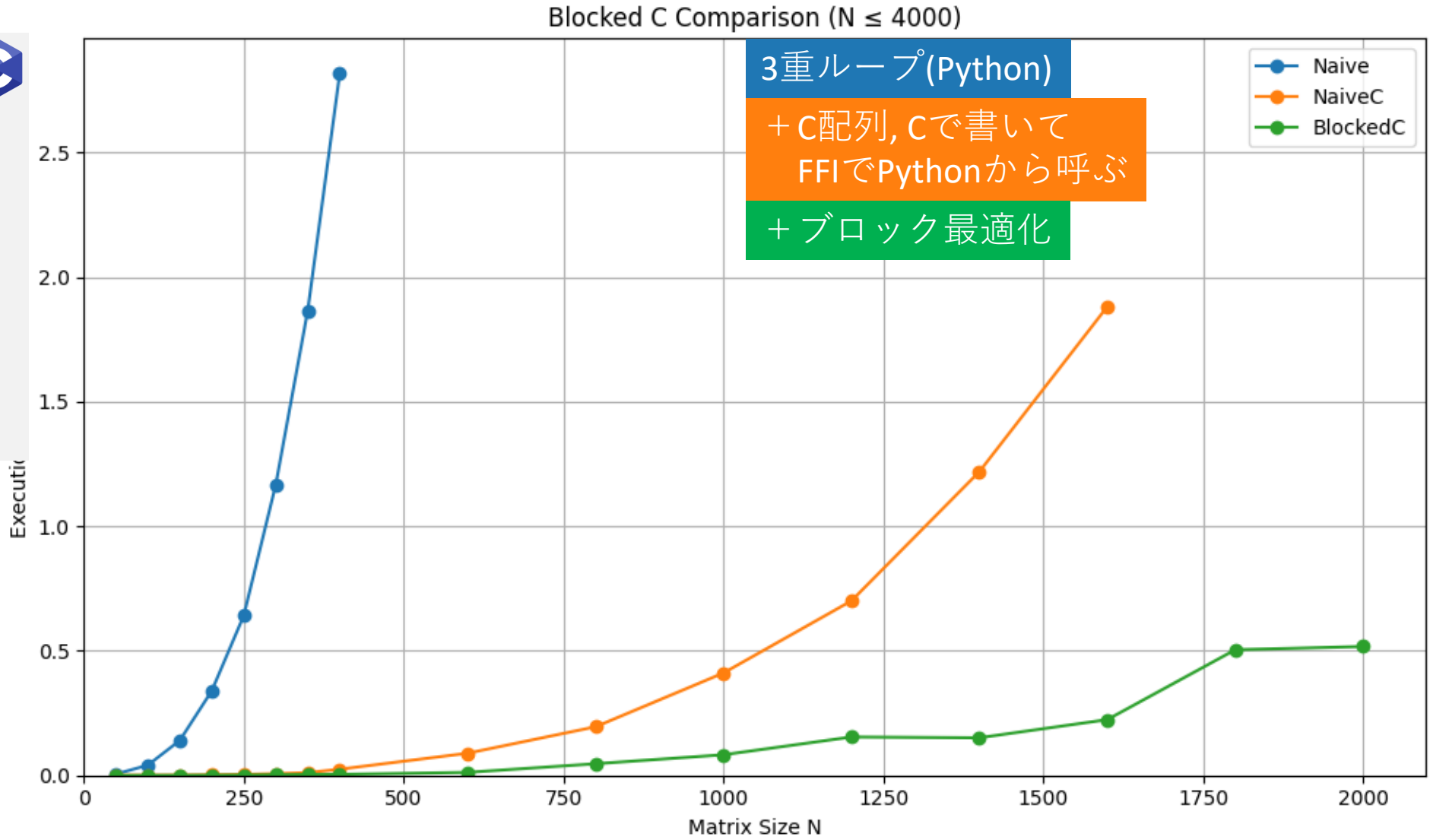
```c
#include <stddef.h>
void matmul_naive_c(
 const double* restrict A,
 const double* restrict B,
 double* restrict C,
 int N
) {
 for (int i = 0; i < N; ++i) {
  for (int j = 0; j < N; ++j) {
   double sum = 0.0;
   for (int k = 0; k < N; ++k) {
    sum += A[i*N+k] * B[k*N+j];
   }
   C[i * N + j] = sum;
}}}
```



動機：もっと高速化できないか？

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
gcc 13.3.0

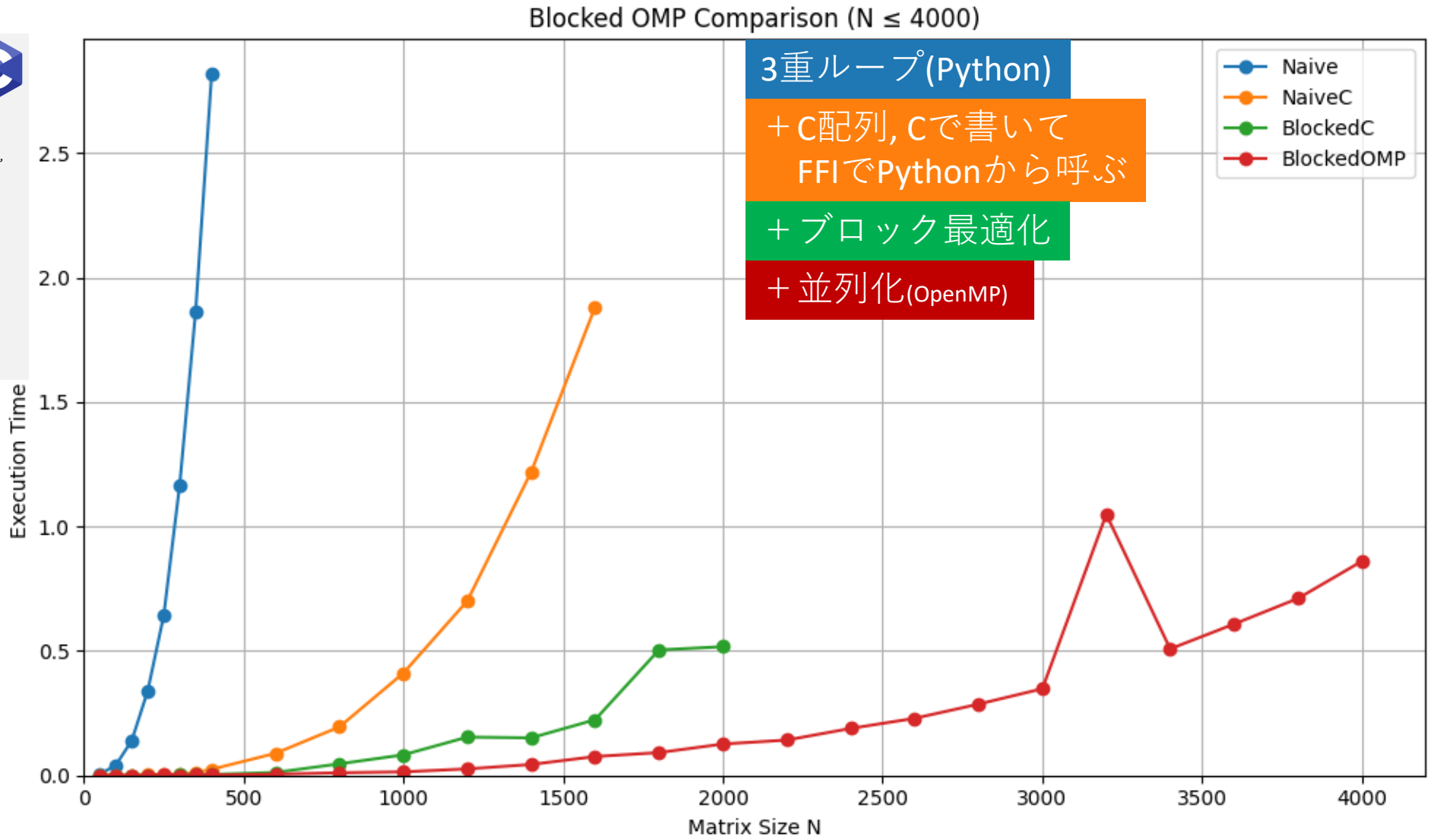
```c
#include <stdint.h>
#include <stdlib.h>
void matmul_blocked_c(
 const double* restrict A,
 const double* restrict B,
 double* restrict C, int N,
 int block_size
) {
 int i, j, k, ii, jj, kk;
 for (ii = 0; ii < N; ii += block_size) {
  for (jj = 0; jj < N; jj += block_size) {
   for (kk = 0; kk < N; kk += block_size) {
    int i_max = ii + block_size < N ? ii +
block_size : N;
    int j_max = jj + block_size < N ? jj +
block_size : N;
    int k_max = kk + block_size < N ? kk +
block_size : N;
    for (i = ii; i < i_max; ++i) {
     for (k = kk; k < k_max; ++k) {
      double a_ik = A[i * N + k];
      for (j = jj; j < j_max; ++j) {
       C[i * N + j] += a_ik * B[k * N + j];
}}}}}}}
```



動機：もっと高速化できないか？

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
gcc 13.3.0
GNU OpenMP (libgomp, OpenMP 4.5)
OMP_NUM_THREADS=32

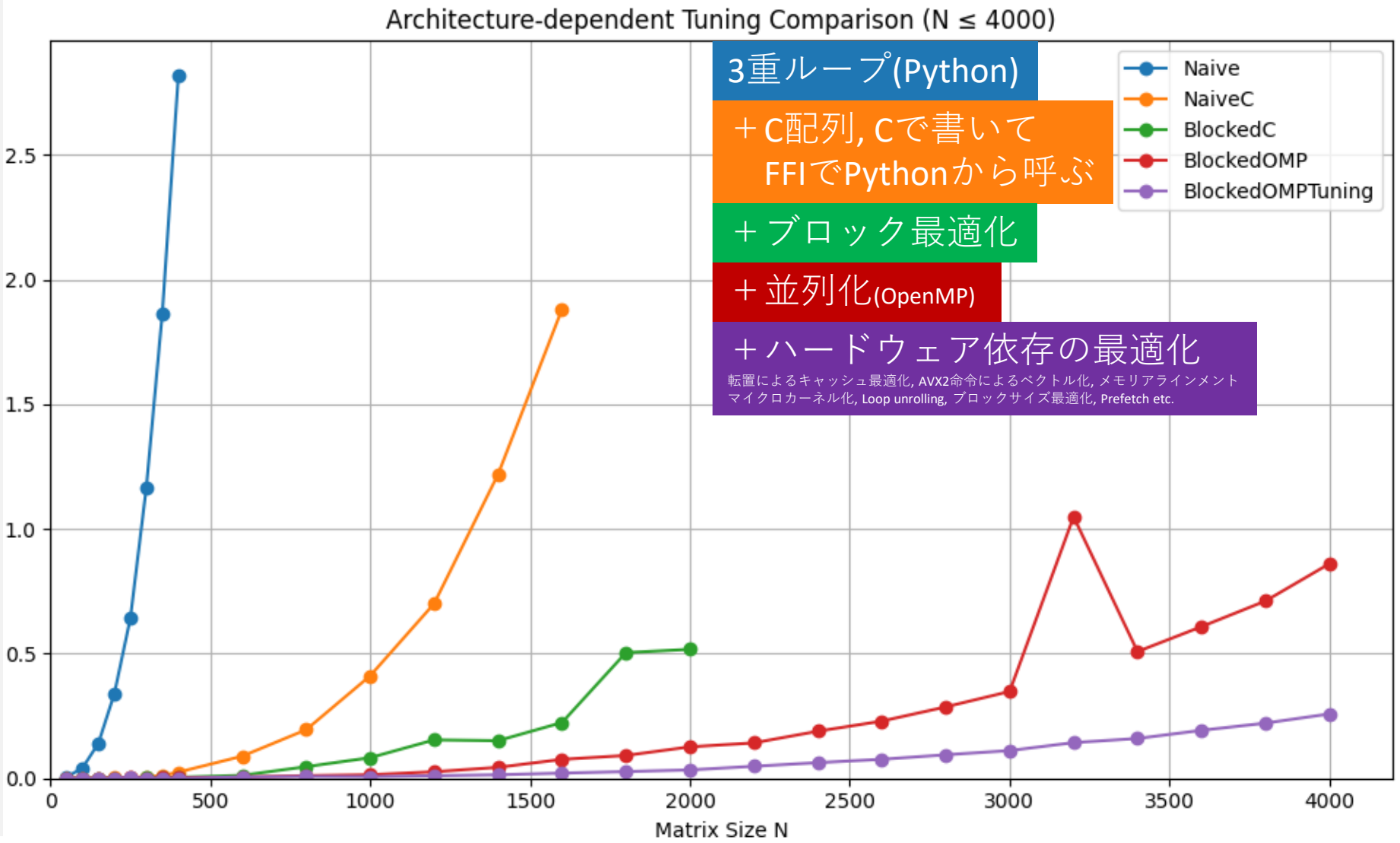
```c
#include <omp.h>
void matmul_blocked_omp_c(
 const double* restrict A,
 const double* restrict B,
 double* restrict C,
 int N,
 int block_size
) {
 int i, j, k, ii, jj, kk;
 #pragma omp parallel for collapse(2) private(ii, jj, kk, i, j,
k) schedule(static)
 for (ii = 0; ii < N; ii += block_size) {
  for (jj = 0; jj < N; jj += block_size) {
   for (kk = 0; kk < N; kk += block_size) {
   int i_max = ii + block_size < N ? ii + block_size : N;
   int j_max = jj + block_size < N ? jj + block_size : N;
   int k_max = kk + block_size < N ? kk + block_size : N;

   for (i = ii; i < i_max; ++i) {
    for (j = jj; j < j_max; ++j) {
     double sum = C[i * N + j];
     #pragma omp simd reduction(+:sum)
     for (k = kk; k < k_max; ++k) {
      sum += A[i * N + k] * B[k * N + j];
     }
     C[i * N + j] = sum;
}}}}}}
```



動機：もっと高速化できないか？

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
gcc 13.3.0
GNU OpenMP (libgomp, OpenMP 4.5)
OMP_NUM_THREADS=32

```c
#include <omp.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <immintrin.h>
#include <assert.h>
#define MICRO_M 8
#define MICRO_N 4
void transpose_for_b_kj(const double* B, double* B_T, int N, int stride, int block_size) {
 #pragma omp parallel for collapse(2) schedule(static)
 for (int kk = 0; kk < N; kk += block_size) {
  for (int jj = 0; jj < N; jj += block_size) {
   int k_max = (kk + block_size < N) ? kk + block_size : N;
   int j_max = (jj + block_size < N) ? jj + block_size : N;
   for (int k = kk; k < k_max; ++k)
    for (int j = jj; j < j_max; ++j)
     B_T[k * stride + j] = B[j * stride + k];
}}}
static inline void microkernel_8x4_unrolled_dynamic(
 const double* A,
 const double* B_T,
 double* C,
 int stride,
 int K,
 int M_actual
) {
 __m256d c[MICRO_M];
 for (int i = 0; i < M_actual; ++i)
  c[i] = _mm256_load_pd(&C[i * stride]);

 for (int p = 0; p < K; ++p) {
  __m256d b = _mm256_load_pd(&B_T[p * stride]);
  for (int i = 0; i < M_actual; ++i) {
   __m256d a = _mm256_set1_pd(A[i * stride + p]);
   c[i] = _mm256_fmadd_pd(a, b, c[i]);
 }}

 for (int i = 0; i < M_actual; ++i)
  _mm256_store_pd(&C[i * stride], c[i]);
}
void matmul_blocked_omp_tuning_c(
 const double* restrict A,
 const double* restrict B,
 double* restrict C,
 int N,
 int stride,
 int block_size
) {
 double* B_T;
 if (posix_memalign((void**)&B_T, 64, sizeof(double) * stride * stride) != 0 || B_T == NULL) {
  fprintf(stderr, "Failed to allocate memory for B_T\n");
  return;
 }
 transpose_for_b_kj(B, B_T, N, stride, block_size);
 #pragma omp parallel for collapse(2) schedule(dynamic, 1)
 for (int jj = 0; jj < N; jj += block_size) {
  for (int ii = 0; ii < N; ii += block_size) {
   for (int kk = 0; kk < N; kk += block_size) {
    int i_max = (ii + block_size < N) ? ii + block_size : N;
    int j_max = (jj + block_size < N) ? jj + block_size : N;
    int k_max = (kk + block_size < N) ? kk + block_size : N;
    int i_tail = i_max % MICRO_M;
    if (i_tail > 0) {
     for (int i = i_max - i_tail; i < i_max; i += i_tail) {
      for (int j = jj; j <= j_max - MICRO_N; j += MICRO_N) {
       for (int k = kk; k < k_max; k += MICRO_N) {
        int k_tail = k_max - k;
        int k_chunk = (k_tail >= MICRO_N) ? MICRO_N : k_tail;
        int prefetch_k = k + 4;
        if (prefetch_k < k_max) {
         _mm_prefetch((const char*)(&A[i * stride + prefetch_k]), _MM_HINT_T0);
         _mm_prefetch((const char*)(&B_T[prefetch_k * stride + j]), _MM_HINT_T0);
        }
        microkernel_8x4_unrolled_dynamic(
         &A[i * stride + k],
         &B_T[k * stride + j],
         &C[i * stride + j],
         stride,
         k_chunk,
         i_tail
        );
      }}
      int j_tail = j_max % MICRO_N;
      if (j_tail > 0) {
       for (int j = j_max - j_tail; j < j_max; ++j) {
        for (int ii_inner = 0; ii_inner < i_tail; ++ii_inner) {
         double sum = 0.0;
         for (int k = kk; k < k_max; ++k) {
          sum += A[(i + ii_inner) * stride + k] * B_T[k * stride + j];
         }
         C[(i + ii_inner) * stride + j] = sum;
     }}}}}
     for (int i = ii; i <= i_max - MICRO_M; i += MICRO_M) {
      for (int j = jj; j <= j_max - MICRO_N; j += MICRO_N) {
       for (int k = kk; k < k_max; k += MICRO_N) {
        int k_tail = k_max - k;
        int k_chunk = (k_tail >= MICRO_N) ? MICRO_N : k_tail;
        microkernel_8x4_unrolled_dynamic(
         &A[i * stride + k],
         &B_T[k * stride + j],
         &C[i * stride + j],
         stride,
         k_chunk,
         MICRO_M
        );
      }}
      int j_tail = j_max % MICRO_N;
      if (j_tail > 0) {
       for (int j = j_max - j_tail; j < j_max; ++j) {
        for (int ii_inner = 0; ii_inner < MICRO_M; ++ii_inner) {
         double sum = 0.0;
         for (int k = kk; k < k_max; ++k) {
          sum += A[(i + ii_inner) * stride + k] * B_T[k * stride + j];
         }
         C[(i + ii_inner) * stride + j] = sum;
}}}}}}}
 free(B_T);
}
```

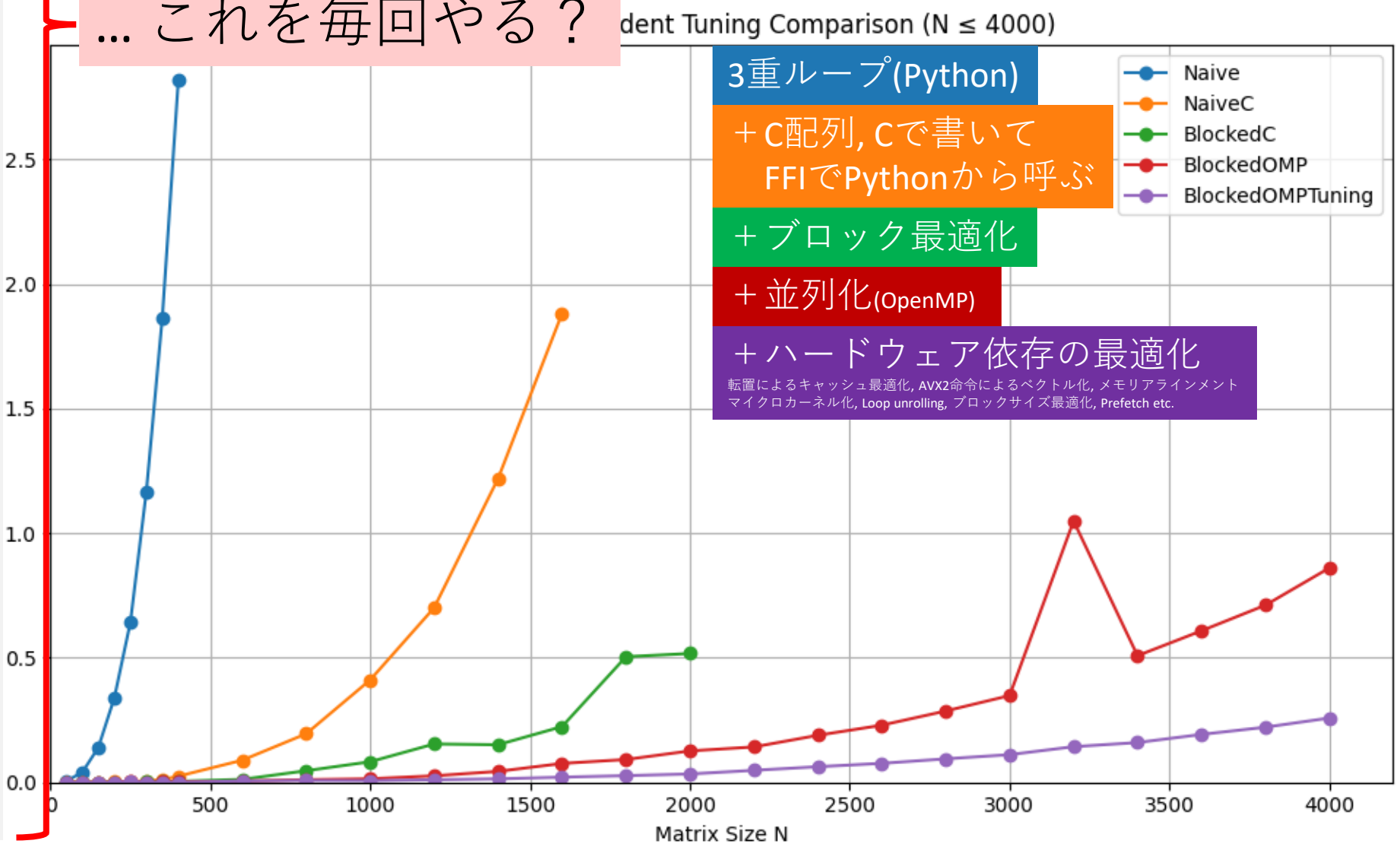


動機：もっと高速化できないか？

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
gcc 13.3.0
GNU OpenMP (libgomp, OpenMP 4.5)
OMP_NUM_THREADS=32

```c
#include <omp.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <immintrin.h>
#include <assert.h>
#define MICRO_M 8
#define MICRO_N 4
void transpose_for_b_kj(const double* B, double* B_T, int N, int stride, int block_size) {
 #pragma omp parallel for collapse(2) schedule(static)
 for (int kk = 0; kk < N; kk += block_size) {
  for (int jj = 0; jj < N; jj += block_size) {
   int k_max = (kk + block_size < N) ? kk + block_size : N;
   int j_max = (jj + block_size < N) ? jj + block_size : N;
   for (int k = kk; k < k_max; ++k)
    for (int j = jj; j < j_max; ++j)
     B_T[k * stride + j] = B[j * stride + k];
}}}
static inline void microkernel_8x4_unrolled_dynamic(
 const double* A,
 const double* B_T,
 double* C,
 int stride,
 int K,
 int M_actual
) {
 __m256d c[MICRO_M];
 for (int i = 0; i < M_actual; ++i)
  c[i] = _mm256_load_pd(&C[i * stride]);

 for (int p = 0; p < K; ++p) {
  __m256d b = _mm256_load_pd(&B_T[p * stride]);
  for (int i = 0; i < M_actual; ++i) {
   __m256d a = _mm256_set1_pd(A[i * stride + p]);
   c[i] = _mm256_fmadd_pd(a, b, c[i]);
 }}

 for (int i = 0; i < M_actual; ++i)
  _mm256_store_pd(&C[i * stride], c[i]);
}
void matmul_blocked_omp_tuning_c(
 const double* restrict A,
 const double* restrict B,
 double* restrict C,
 int N,
 int stride,
 int block_size
) {
 double* B_T;
 if (posix_memalign((void**)&B_T, 64, sizeof(double) * stride * stride) != 0 || B_T == NULL) {
  fprintf(stderr, "Failed to allocate memory for B_T\n");
  return;
 }
 transpose_for_b_kj(B, B_T, N, stride, block_size);
 #pragma omp parallel for collapse(2) schedule(dynamic, 1)
 for (int jj = 0; jj < N; jj += block_size) {
  for (int ii = 0; ii < N; ii += block_size) {
   for (int kk = 0; kk < N; kk += block_size) {
    int i_max = (ii + block_size < N) ? ii + block_size : N;
    int j_max = (jj + block_size < N) ? jj + block_size : N;
    int k_max = (kk + block_size < N) ? kk + block_size : N;
    int i_tail = i_max % MICRO_M;
    if (i_tail > 0) {
     for (int i = i_max - i_tail; i < i_max; i += i_tail) {
      for (int j = jj; j <= j_max - MICRO_N; j += MICRO_N) {
       for (int k = kk; k < k_max; k += MICRO_N) {
        int k_tail = k_max - k;
        int k_chunk = (k_tail >= MICRO_N) ? MICRO_N : k_tail;
        int prefetch_k = k + 4;
        if (prefetch_k < k_max) {
         _mm_prefetch((const char*)(&A[i * stride + prefetch_k]), _MM_HINT_T0);
         _mm_prefetch((const char*)(&B_T[prefetch_k * stride + j]), _MM_HINT_T0);
        }
        microkernel_8x4_unrolled_dynamic(
         &A[i * stride + k],
         &B_T[k * stride + j],
         &C[i * stride + j],
         stride,
         k_chunk,
         i_tail
        );
      }}
      int j_tail = j_max % MICRO_N;
      if (j_tail > 0) {
       for (int j = j_max - j_tail; j < j_max; ++j) {
        for (int ii_inner = 0; ii_inner < i_tail; ++ii_inner) {
         double sum = 0.0;
         for (int k = kk; k < k_max; ++k) {
          sum += A[(i + ii_inner) * stride + k] * B_T[k * stride + j];
         }
         C[(i + ii_inner) * stride + j] = sum;
     }}}}}
     for (int i = ii; i <= i_max - MICRO_M; i += MICRO_M) {
      for (int j = jj; j <= j_max - MICRO_N; j += MICRO_N) {
       for (int k = kk; k < k_max; k += MICRO_N) {
        int k_tail = k_max - k;
        int k_chunk = (k_tail >= MICRO_N) ? MICRO_N : k_tail;
        microkernel_8x4_unrolled_dynamic(
         &A[i * stride + k],
         &B_T[k * stride + j],
         &C[i * stride + j],
         stride,
         k_chunk,
         MICRO_M
        );
      }}
      int j_tail = j_max % MICRO_N;
      if (j_tail > 0) {
       for (int j = j_max - j_tail; j < j_max; ++j) {
        for (int ii_inner = 0; ii_inner < MICRO_M; ++ii_inner) {
         double sum = 0.0;
         for (int k = kk; k < k_max; ++k) {
          sum += A[(i + ii_inner) * stride + k] * B_T[k * stride + j];
         }
         C[(i + ii_inner) * stride + j] = sum;
}}}}}}}
 free(B_T);
}
```

… これを毎回やる？

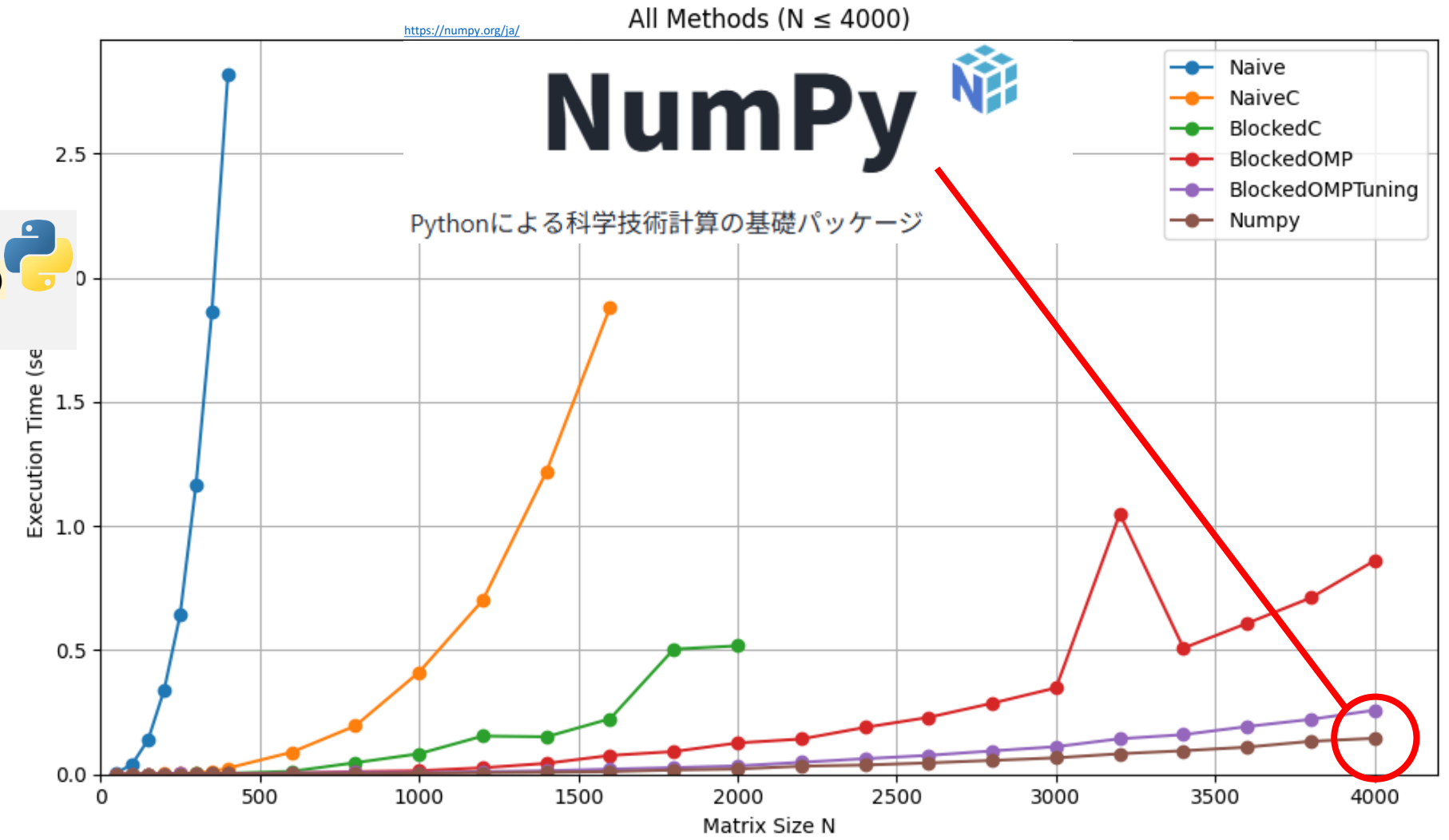


NumPyは高性能計算の為の **最適化をプログラマから隠蔽**

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
NumPy 2.3.2

NumPyなら
$A \cdot B$に近い記法
を書くだけでOK!

```
def matrix_mul(A, B):
  C = np.dot(A_np,B_np)
  return C
```



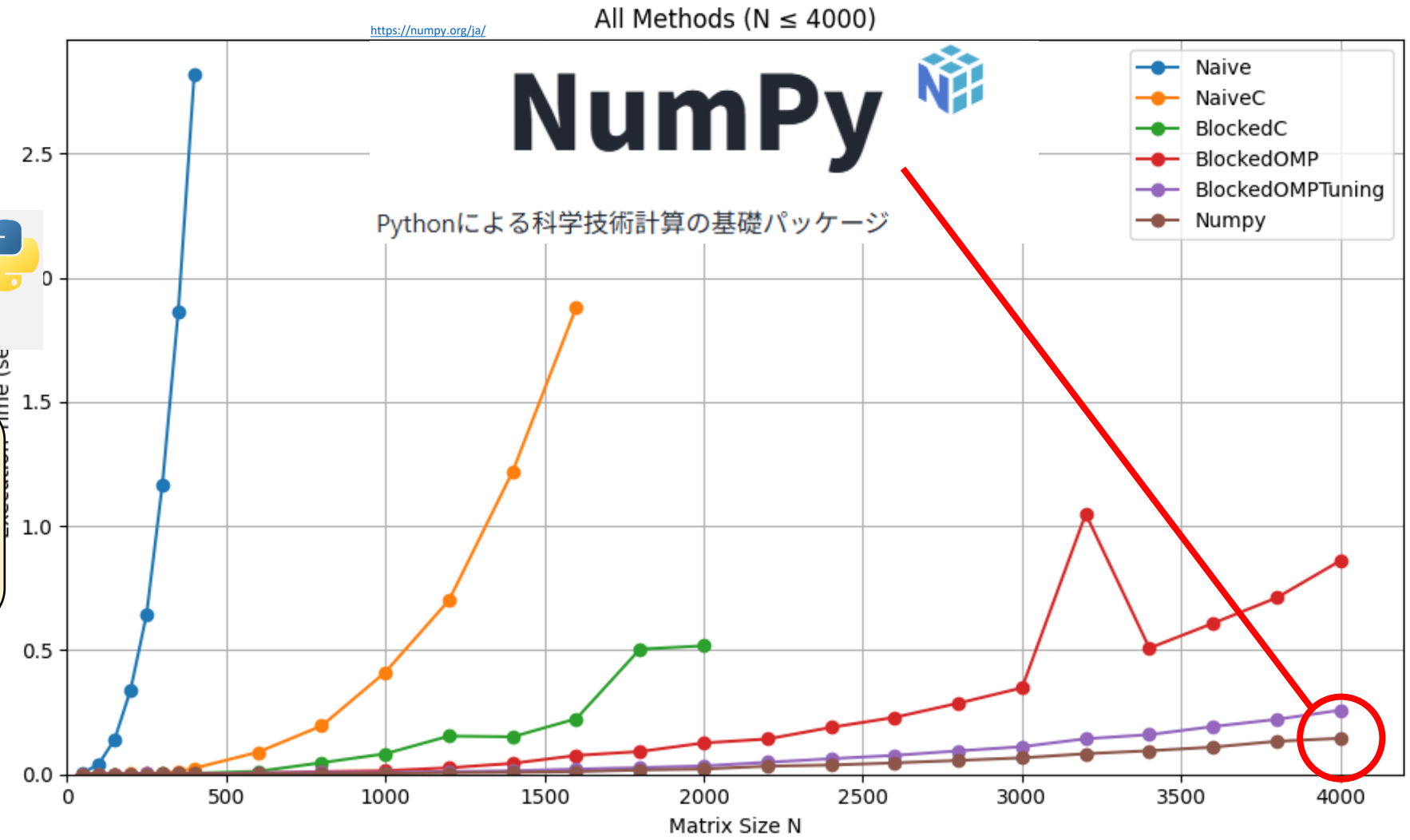
NumPyは高性能計算の為の **最適化をプログラマから隠蔽**

Ubuntu 24.04.2 LTS
Ryzen 9 7950X 16-Core
MemTotal: 130986880 kB
Python 3.12.3
NumPy 2.3.2

NumPyなら
$A \cdot B$に近い記法
を書くだけでOK!

```
def matrix_mul(A, B):
    C = np.dot(A_np,B_np)
    return C
```

必要な最適化や
チューニングは
NumPyの裏で実施



コンパイラ技術の精神：
簡潔な表現から実行時性能を引き出す

領域特化言語(DSL)
による**容易な記述**

コンパイル技術
による**最適化**

コンパイラ技術の精神：
簡潔な表現から実行時性能を引き出す

領域特化言語(DSL)
による**容易な記述**

コンパイル技術
による**最適化**

積 `np.dot(A,B)`	大きさ `np.size(A,1)`
内積 `np.inner(A,B)`	正規化 `np.linarg.norm(A)`
行列式 `np.det(A)`	ランク `np.matrix_rank(A)`

… 他多数

コンパイラ技術の精神：
簡潔な表現から実行時性能を引き出す

領域特化言語(DSL)
による**容易な記述**

積
`np.dot(A,B)`

大きさ
`np.size(A,1)`

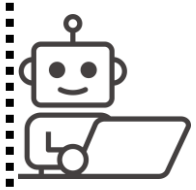
内積
`np.inner(A,B)`

正規化
`np.linarg.norm(A)`

行列式
`np.det(A)`

ランク
`np.matrix_rank(A)`

… 他多数



計算機に応じて
自動選択

コンパイル技術
による**最適化**

業界標準の線形代数
ライブラリ

http://www.openmathlib.org/OpenBLAS/docs/

OpenBLAS

https://www.netlib.org/lapack/

Accelerate

https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

Intel® oneAPI Math Kernel Library (oneMKL)

30 years of math performance excellence: Accelerate math processing routines and reduce development time.

https://developer.apple.com/jp/accelerate/

CやFortran実装
ベクトル命令化
自動並列化
キャッシュ最適化
カーネル最適化

コンパイラ技術の精神：
簡潔な表現から実行時性能を引き出す

領域特化言語(DSL)による**容易な記述**

NumPyは「Pythonの顔をしたC/Fortranへのコンパイラ」

コンパイル技術による**最適化**

積
`np.dot(A,B)`

大きさ
`np.size(A,1)`

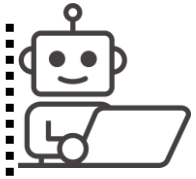
内積
`np.inner(A,B)`

正規化
`np.linarg.norm(A)`

行列式
`np.det(A)`

ランク
`np.matrix_rank(A)`

… 他多数



計算機に応じて自動選択

業界標準の線形代数ライブラリ

http://www.openmathlib.org/OpenBLAS/docs/

OpenBLAS

https://www.netlib.org/lapack/

Accelerate

https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

Intel® oneAPI Math Kernel Library (oneMKL)

30 years of math performance excellence: Accelerate math processing routines and reduce development time.

https://developer.apple.com/jp/accelerate/

CやFortran実装
ベクトル命令化
自動並列化
キャッシュ最適化
カーネル最適化

プログラミングを**科学**する

表現する、処理する、証明する

表現する：人間の意図や仕様を計算機に伝える方法

処理する：プログラムを自動で最適化する方法

証明する：プログラムが正しく動作することを検証する方法

動機：“**正しい**”ことを確信するには？

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj} \right)$$

≠

テストでは**他の入力で失敗する可能性を排除できない！**

A	B	$A \cdot B$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$	$\begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
⋮	⋮	⋮

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0]*n for _ in range(l)]
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k] * B[k][j]
  return C
```

3重ループの実装

```
# test_matrix.py
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
assert matmul(A, B) == [[19, 22], [43, 50]],\
 "テスト1失敗"
print("テスト1成功！")
…
```

単体テスト

シンボリック実行で**反例探索**

③ 性質を数学的に述べる(略記)

$$\forall i, j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj} \right)$$

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0] * n for _ in range(l)]  # C の要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

シンボリック実行で**反例探索**

CrossHair: https://crosshair-web.org/

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

≒

「変数のまま」計算する

```
def spec_matmul
(A: List[List[int]], B: List[List[int]]) -> List[List[int]]:
  l, m, n = len(A), len(A[0]), len(B[0])

  return [[sum(A[i][k] * B[k][j]
           for k in range(m))
                    for j in range(n)]
               for i in range(l)]

"""
pre: len(A) > 0 and len(B) > 0
pre: all(len(row) == len(A[0]) for row in A)
pre: all(len(row) == len(B[0]) for row in B)
pre: len(A[0]) == len(B)
post: _ == spec_matmul(A, B)
"""
```

オラクル的**仕様**

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

シンボリック実行で**反例探索**

CrossHair: https://crosshair-web.org/

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

≒

```
def spec_matmul
(A: List[List[int]], B: List[List[int]]) -> List[List[int]]:
  l, m, n = len(A), len(A[0]), len(B[0])

  return [[sum(A[i][k] * B[k][j]
           for k in range(m))
                  for j in range(n)]
              for i in range(l)]

"""
pre: len(A) > 0 and len(B) > 0
pre: all(len(row) == len(A[0]) for row in A)
pre: all(len(row) == len(B[0]) for row in B)
pre: len(A[0]) == len(B)
post: _ == spec_matmul(A, B)
"""
```

オラクル的
仕様

「変数のまま」計算する

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$
$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Step 1
変数を用意

シンボリック実行で**反例探索**

CrossHair: https://crosshair-web.org/

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

≒

```
def spec_matmul
(A: List[List[int]], B: List[List[int]]) -> List[List[int]]:
  l, m, n = len(A), len(A[0]), len(B[0])

  return [[sum(A[i][k] * B[k][j]
           for k in range(m))
                for j in range(n)]
            for i in range(l)]

"""
pre: len(A) > 0 and len(B) > 0
pre: all(len(row) == len(A[0]) for row in A)
pre: all(len(row) == len(B[0]) for row in B)
pre: len(A[0]) == len(B)
post: _ == spec_matmul(A, B)
"""
```

オラクル的
仕様

「変数のまま」計算する

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$
$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Step 1
変数を用意

Step 2
プログラムから変数の関係を収集

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

シンボリック実行で**反例探索**

CrossHair: https://crosshair-web.org/

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

≒

```
def spec_matmul
(A: List[List[int]], B: List[List[int]]) -> List[List[int]]:
  l, m, n = len(A), len(A[0]), len(B[0])

  return [[sum(A[i][k] * B[k][j]
           for k in range(m))
                  for j in range(n)]
              for i in range(l)]

"""
pre: len(A) > 0 and len(B) > 0
pre: all(len(row) == len(A[0]) for row in A)
pre: all(len(row) == len(B[0]) for row in B)
pre: len(A[0]) == len(B)
post: _ == spec_matmul(A, B)
"""
```

オラクル的
仕様

「変数のまま」計算する

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$
$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Step 1
変数を用意

Step 2
プログラムから変数の関係を収集

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

Step 3
反例探索

Check “¬post”

Z3
https://www.microsoft.com/en-us/research/project/z3-3

反例発見
or
見つからず

シンボリック実行で**反例探索**

CrossHair: https://crosshair-web.org/

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

≒

```
def spec_matmul
(A: List[List[int]], B: List[List[int]]) -> List[List[int]]:
  l, m, n = len(A), len(A[0]), len(B[0])

  return [[sum(A[i][k] * B[k][j]
           for k in range(m))
                  for j in range(n)]
              for i in range(l)]

"""
pre: len(A) > 0 and len(B) > 0
pre: all(len(row) == len(A[0]) for row in A)
pre: all(len(row) == len(B[0]) for row in B)
pre: len(A[0]) == len(B)
post: _ == spec_matmul(A, B)
"""
```

オラクル的**仕様**

```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C の要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

「変数のまま」計算する

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$
$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Step 1
変数を用意

Step 2
プログラムから変数の関係を収集

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

Step 3
反例探索

Check “¬post”

Z3
https://www.microsoft.com/en-us/research/project/z3-3

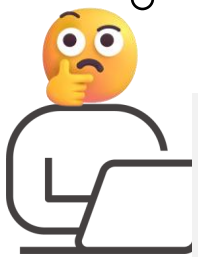
反例発見
or
見つからず

🤔**反例が見つからなかっただけでは？**

プログラム検証による**数学的証明**

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$



```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0] * n for _ in range(l)]  # C の要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

プログラム検証による**数学的証明**

Dafny: https://dafny.org/

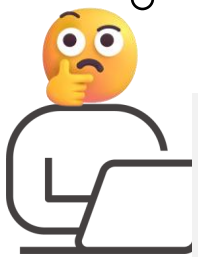
③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

=

```
ensures |C| == |A|
ensures forall i,
  0 <= i < |A|
  |C[i]| == |B[0]|
ensures forall i, j,
  0 <= i < |A| && 0 <= j < |B[0]| ==>
  C[i][j] == inpro(i, j, A, B, |A[0]|)
```

数学的仕様



```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C の要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

等価なプログラムに変換

```
function inpro(i: int, j: int,
  A: seq<seq<int>>, B: seq<seq<int>>, k: int
): int {
  if k == 0 then 0
  else inpro(i, j, A, B, k - 1)
       + A[i][k - 1] * B[k - 1][j]
}
method matmul(A: seq<seq<int>>, B: seq<seq<int>>)
returns (C: seq<seq<int>>) {
  var l := |A|;       // 行数（Aの行）
  var m := |A[0]|;    // Aの列数 = Bの行数
  var n := |B[0]|;    // 列数（Bの列）
  C := [];
  var i := 0;
  while i < l {
    var row := [];
    var j := 0;
    while j < n {
      var s := 0;
      var k := 0;
      while k < m {
        s := s + A[i][k] * B[k][j];
        k := k + 1;
      }
      row := row + [s];
      j := j + 1;
    }
    C := C + [row];
    i := i + 1;
  }
}
```

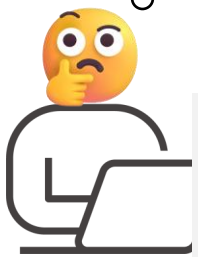
プログラムの実装

プログラム検証による**数学的証明**

Dafny: https://dafny.org/

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$



```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0] * n for _ in range(l)]  # C の要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

＝

等価性を証明

```
ensures |C| == |A|
ensures forall i,
  0 <= i < |A|
  |C[i]| == |B[0]|
ensures forall i,
  0 <= i < |A| && 0 <= j < |B[0]| ==>
  C[i][j] == inpro(i, j, A, B, |A[0]|)
```

数学的仕様

```
function inpro(i: int, j: int,
  A: seq<seq<int>>, B: seq<seq<int>>, k: int
): int {
  if k == 0 then 0
  else inpro(i, j, A, B, k - 1)
       + A[i][k - 1] * B[k - 1][j]
}
method matmul(A: seq<seq<int>>, B: seq<seq<int>>)
returns (C: seq<seq<int>>) {
  var l := |A|;       // 行数（Aの行）
  var m := |A[0]|;    // Aの列数 = Bの行数
  var n := |B[0]|;    // 列数（Bの列）
  C := [];
  var i := 0;
  while i < l {
    var row := [];
    var j := 0;
    while j < n {
      var s := 0;
      var k := 0;
      while k < m {
        s := s + A[i][k] * B[k][j];
        k := k + 1;
      }
      row := row + [s];
      j := j + 1;
    }
    C := C + [row];
    i := i + 1;
  }
}
```

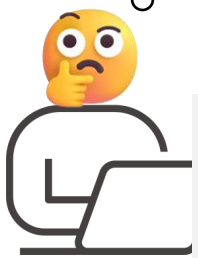
プログラムの実装

等価な
プログラム
に変換

プログラム検証による**数学的証明**

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$



```
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C要素を0初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

=

Dafny: https://dafny.org/

```
function inpro(
  i: int, j: int, A: seq<seq<int>>,
  B: seq<seq<int>>, k: int
): int {
  if k == 0 then 0
  else inpro(i,j,A,B,k-1) +
         A[i][k-1] * B[k-1][j]
}

method matmul(
  A: seq<seq<int>>, B: seq<seq<int>>
) returns (C: seq<seq<int>>)
  ensures |C| == |A|
  ensures forall i ::
    0 <=
  ensure
    0 <=
      C[
{
  var l
  var m := |A[0]|;
  var n := |B[0]|;
  C := [];
  var i := 0;
  while i < l {
    var row := [];
    var j := 0;
    while j < n {
      var s := 0;
      var k := 0;
      while k < m {
        s := s + A[i][k] * B[k][j];
        k := k + 1;
      }
      row := row + [s];
      j := j + 1;
    }
    C := C + [row];
    i := i + 1;
  }
}
```

```
requires |A| > 0 && |B| > 0
requires 0 <= i < |A| && 0 <= j < |B[0]|
requires forall row ::
  row in A ==> |row| == |A[0]|
requires forall row ::
  row in B ==> |row| == |B[0]|
requires |A[0]| == |B|
requires 0 <= k <= |A[0]|
decreases k
```

```
requires |A| > 0 && |B| > 0
requires forall row ::
  row in A ==> |row| == |A[0]|
requires forall row ::
  row in B ==> |row| == |B[0]|
requires |A[0]| == |B|
```

条件を付けて証明を誘導

```
invariant 0 <= i <= l
invariant |C| == i
invariant forall ii ::
  0 <= ii < i ==> |C[ii]| == n
invariant forall ii, jj ::
  0 <= ii < i && 0 <= jj < n ==>
    C[ii][jj] == inpro(ii, jj, A, B, m)
decreases l - i
```

```
invariant 0 <= j <= n
invariant |row| == j
invariant forall jj :: 0 <= jj < j ==>
  row[jj] == inpro(i, jj, A, B, m)
decreases n - j
```

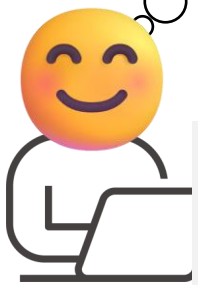
```
invariant 0 <= k <= m
invariant s == inpro(i, j, A, B, k)
decreases m - k
```

プログラム検証による**数学的証明**

③ 性質を数学的に述べる(略記)

$$\forall i,j.\ c_{ij} = \left(\sum_{k=1}^{m} a_{ik} \times b_{kj}\right)$$

=



```python
def matrix_mul(A, B):
  l = len(A)
  m = len(A[0])
  n = len(B[0])
  C = [[0 for _ in range(n)] for _ in range(l)]  # C の要素を0で初期化
  for i in range(l):
    for j in range(n):
      for k in range(m):
        C[i][j] += A[i][k]
                 * B[k][j]
  return C
```

3重ループの実装

条件が全経路・入力で成立することを証明

✅ ← Z3

https://www.microsoft.com/en-us/research/project/z3-3

Dafny: https://dafny.org/

```
function inpro(
  i: int, j: int, A: seq<seq<int>>,
  B: seq<seq<int>>, k: int
): int {
  if k == 0 then 0
  else inpro(i,j,A,B,k-1) +
         A[i][k-1] * B[k-1][j]
}

method matmul(
  A: seq<seq<int>>, B: seq<seq<int>>
) returns (C: seq<seq<int>>)
  ensures |C| == |A|
  ensures forall i ::
    0 <=
  ensure
    0 <=
      C[

{
  var l
  var m := |A[0]|;
  var n := |B[0]|;
  C := [];
  var i := 0;
  while i < l {
    var row := [];
    var j := 0;
    while j < n {
      var s := 0;
      var k := 0;
      while k < m {
        s := s + A[i][k] * B[k][j];
        k := k + 1;
      }
      row := row + [s];
      j := j + 1;
    }
    C := C + [row];
    i := i + 1;
  }
}
```

```
requires |A| > 0 && |B| > 0
requires 0 <= i < |A| && 0 <= j < |B[0]|
requires forall row ::
  row in A ==> |row| == |A[0]|
requires forall row ::
  row in B ==> |row| == |B[0]|
requires |A[0]| == |B|
requires 0 <= k <= |A[0]|
decreases k
```

```
requires |A| > 0 && |B| > 0
requires forall row ::
  row in A ==> |row| == |A[0]|
requires forall row ::
  row in B ==> |row| == |B[0]|
requires |A[0]| == |B|
```

条件を付けて証明を誘導

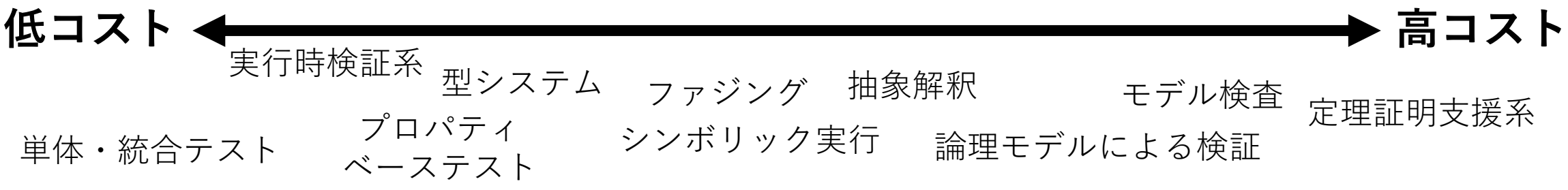
```
invariant 0 <= i <= l
invariant |C| == i
invariant forall ii ::
  0 <= ii < i ==> |C[ii]| == n
invariant forall ii, jj ::
  0 <= ii < i && 0 <= jj < n ==>
    C[ii][jj] == inpro(ii, jj, A, B, m)
decreases l - i
```

```
invariant 0 <= j <= n
invariant |row| == j
invariant forall jj :: 0 <= jj < j ==>
  row[jj] == inpro(i, jj, A, B, m)
decreases n - j
```

```
invariant 0 <= k <= m
invariant s == inpro(i, j, A, B, k)
decreases m - k
```

Dafny

プログラム検証の精神：
良い近似的表現の発見は検証を容易にする

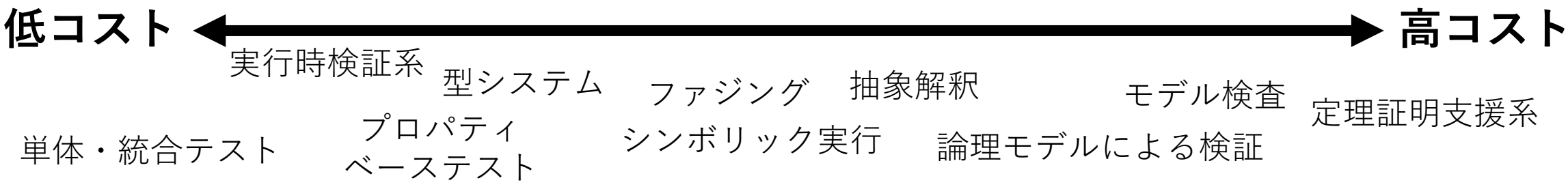


実装の把握

検証

仕様の記述

プログラム検証の精神：
良い近似的表現の発見は検証を容易にする



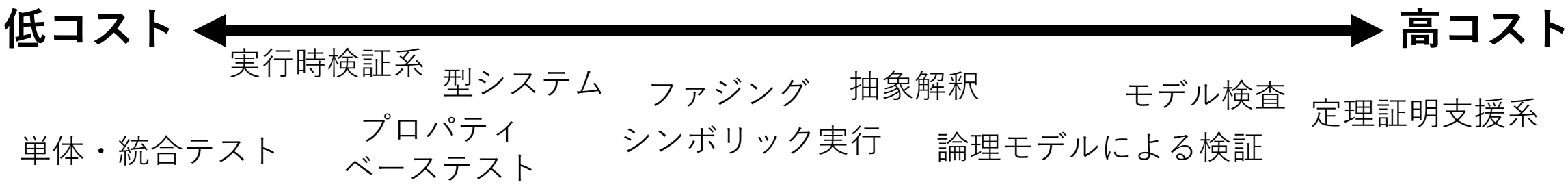
単体テスト

実装の把握　実行観察

検証
- 👍 自動的
- 👎 網羅性無し

仕様の記述　入出力例

プログラム検証の精神：
良い近似的表現の発見は検証を容易にする



	単体テスト	型システム
実装の把握	実行観察	型(構文的近似)
検証	👍 自動的 👎 網羅性無し	👍 (半)自動的 👎 仕様記述に限界
仕様の記述	入出力例	型の上で書く ルールや論理式

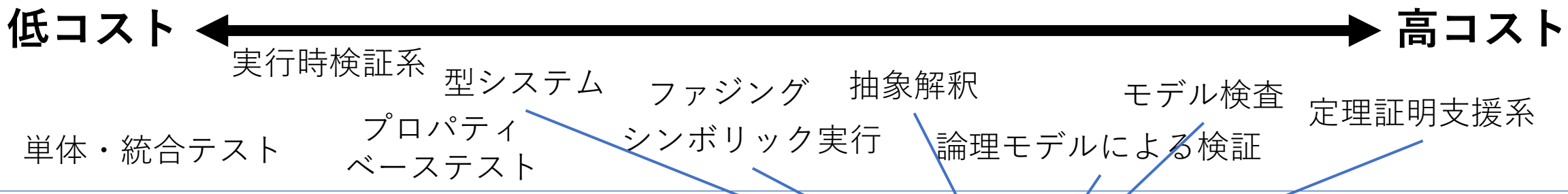
プログラム検証の精神：
良い近似的表現の発見は検証を容易にする

低コスト ⟷ **高コスト**

単体・統合テスト
実行時検証系
プロパティベーステスト
型システム
ファジング
シンボリック実行
抽象解釈
論理モデルによる検証
モデル検査
定理証明支援系

	単体テスト	型システム	プログラム検証による数学的証明
実装の把握	実行観察	型(構文的近似)	🥴厳密な意味定義
検証	👍 自動的 👎 網羅性無し	👍 (半)自動的 👎 仕様記述に限界	👍 数学的保証 👎 証明地獄(not 自動)
仕様の記述	入出力例	型の上で書く ルールや論理式	🥴数学的仕様

プログラム検証の精神：
良い近似的表現の発見は検証を容易にする



	単体テスト	型システム		プログラム検証による数学的証明
実装の把握	実行観察	型(構文的近似)	**良い近似的表現**	厳密な意味定義
検証	👍 自動的 👎 網羅性無し	👍 (半)自動的 👎 仕様記述に限界	👍 **(半)自動的？** (注：万能な自動検証器は存在しない)	👍 数学的保証 👎 証明地獄(not 自動)
仕様の記述	入出力例	型の上で書くルールや論理式	良い表現の上で記述可能な **強力な仕様**	数学的仕様

最近のプログラミング言語研究と社会

表現する

LLM(ある種のプログラム合成)の
プログラミングへの本格採用

処理する

機械学習向けDSLと
コンパイラの熾烈な競争

証明する

学術の歴史を汲む
検証技術に社会的注目

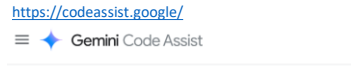
最近のプログラミング言語研究と社会

表現する

LLM(ある種のプログラム合成)の
プログラミングへの本格採用

https://github.com/features/copilot?locale=ja

https://codeassist.google/



Gemini Code Assist: AI-first
coding in your natural language

OpenAI

OpenAI Codex

Move through your backlog faster with AI coding
tools powered by our most capable models.

https://openai.com/codex/

https://claude.ai/

https://replit.com/

処理する

機械学習向けDSLと
コンパイラの熾烈な競争

証明する

学術の歴史を汲む
検証技術に社会的注目

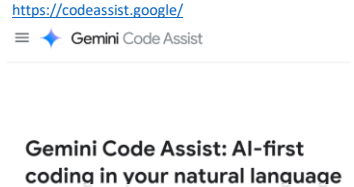
最近のプログラミング言語研究と社会

表現する

LLM(ある種のプログラム合成)の
プログラミングへの本格採用

https://github.com/features/copilot?locale=ja

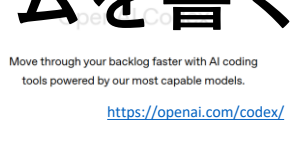
https://codeassist.google/



(2025年8月6日時点では高頻度で)
誤ったプログラムを書く

https://claude.ai/

https://replit.com/



https://openai.com/codex/

依然として重要なこと：
仕様は誰がどう書くの？

処理する

機械学習向けDSLと
コンパイラの熾烈な競争

証明する

学術の歴史を汲む
検証技術に社会的注目

最近のプログラミング言語研究と社会

表現する

LLM(ある種のプログラム合成)の
プログラミングへの本格採用

(2025年8月6日時点では高頻度で)
誤ったプログラムを書く

依然として重要なこと：
仕様は誰がどう書くの？

処理する

機械学習向けDSLと
コンパイラの熾烈な競争

企業	クラウド	DSL
Google	Cloud TPU	JAX/XLA
Amazon	AWS	SagaMaker
Microsoft	Azure	ONNX

企業	フレームワーク
NVIDIA	Cuda
Apple	CoreML
Intel	OpenVINO
AMD	Vitis AI

自社製品を「使わせる」為
コンパイラ開発は
産業競争力に直結

証明する

学術の歴史を汲む
検証技術に社会的注目

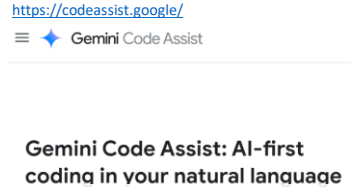
最近のプログラミング言語研究と社会

表現する

LLM(ある種のプログラム合成)の
プログラミングへの本格採用

https://github.com/features/copilot?locale=ja

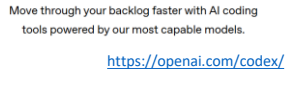
https://codeassist.google/



(2025年8月6日時点では高頻度で)
誤ったプログラムを書く

https://claude.ai/

https://replit.com/



https://openai.com/codex/

依然として重要なこと：
仕様は誰がどう書くの？

処理する

機械学習向けDSLと
コンパイラの熾烈な競争

企業	クラウド	DSL
Google	Cloud TPU	JAX/XLA
Amazon	AWS	SagaMaker
Microsoft	Azure	ONNX

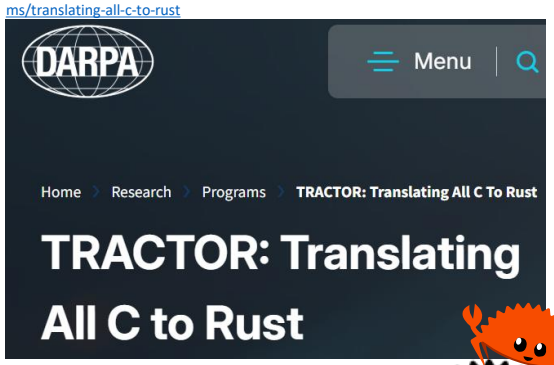
企業	フレームワーク
NVIDIA	Cuda
Apple	CoreML
Intel	OpenVINO
AMD	Vitis AI

自社製品を「使わせる」為
コンパイラ開発は
産業競争力に直結

証明する

学術の歴史を汲む
検証技術に社会的注目

https://www.darpa.mil/research/programs/translating-all-c-to-rust



https://bidenwhitehouse.archives.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/

https://github.com/rust-lang/rust-artwork/blob/master/logo/rust-logo-blk.svg
https://rustacean.net/

Q. プログラミング(言語)を**科学**するとは？

（私の考えでは）

“PLの視座”から

コンピューティングの問題

に迫ること

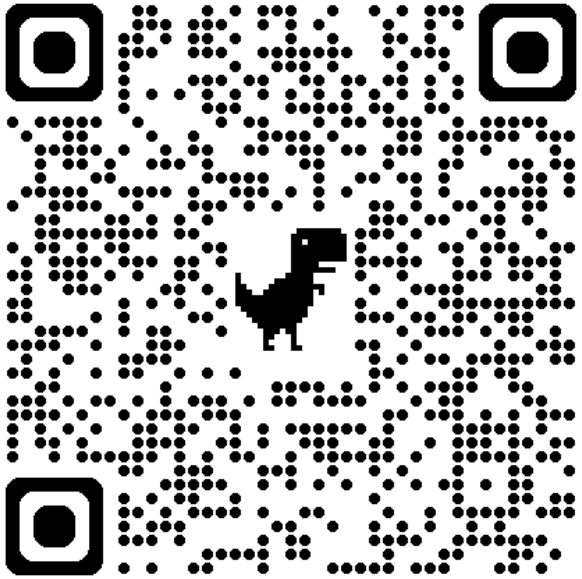
OS, インターフェース, ソフトウェア開発環境, セキュリティ, 分散計算, データベース, 機械学習, ネットワーク, 画像処理, ハードウェア設計, ブロックチェーン, 量子計算 etc.

ある問題に対する解決策の
簡潔で汎用的な表現
を見つけたい！

表現する、処理する、証明する

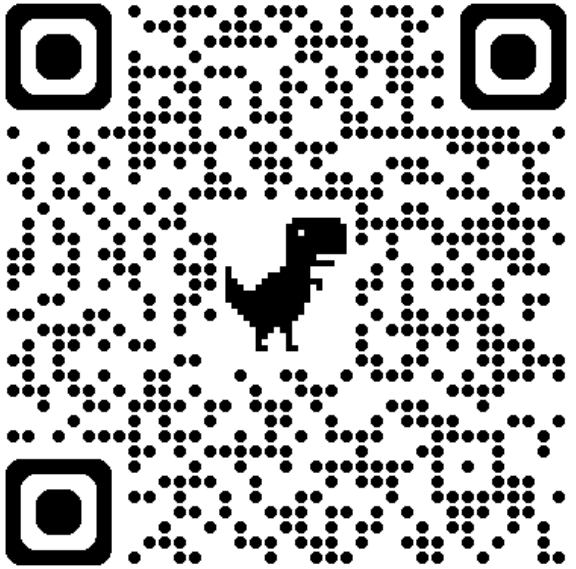
今日のスライドとプログラム

スライド



https://prg.is.titech.ac.jp/ja/news/tanabe-presents-research-introduction-at-open-campus-2025/

プログラム



https://github.com/yudaitnb/matrix-benchmark-oc/tree/main