SAKURA internet

セルフホスト型ランタイムによる WebAssemblyインストルメンテーションの実現可能性検討

Feasibility Study of WebAssembly Binary Instrumentation with Self-hosted Runtime

2024/09/12　日本ソフトウェア科学会第 41 回大会

中田 裕貴 [1,2]　松原 克弥 [2]

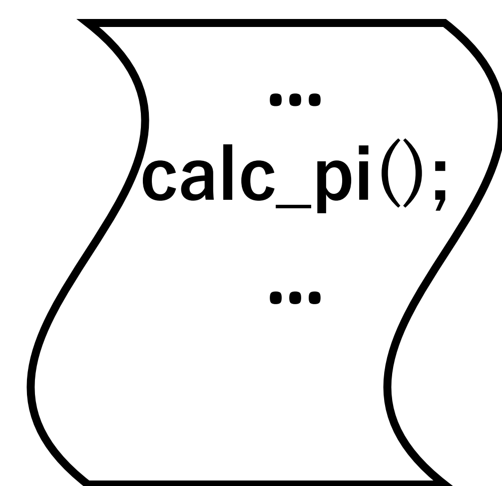
[1]:さくらインターネット株式会社　さくらインターネット研究所

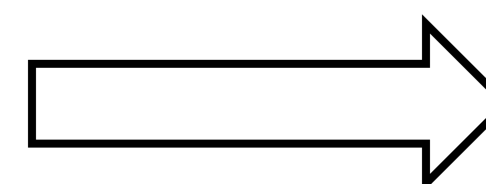
[2]:公立はこだて未来大学　システムソフトウェア研究室

インストルメンテーション（Binary / Code Instrumentation）

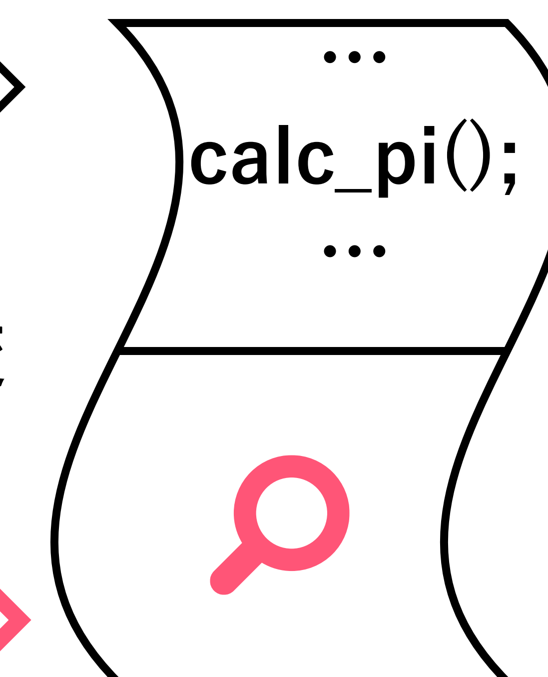
アプリケーション実行中の状態を把握するための技術

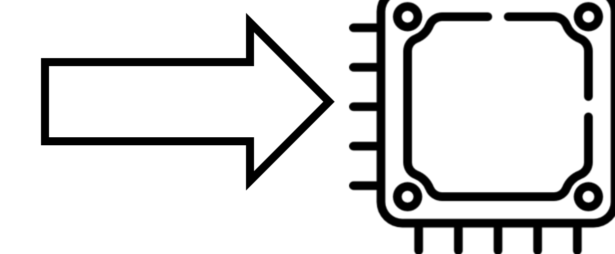
アプリケーションのコード・バイナリ

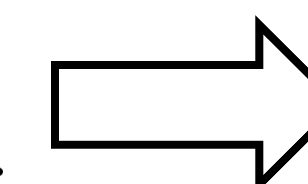


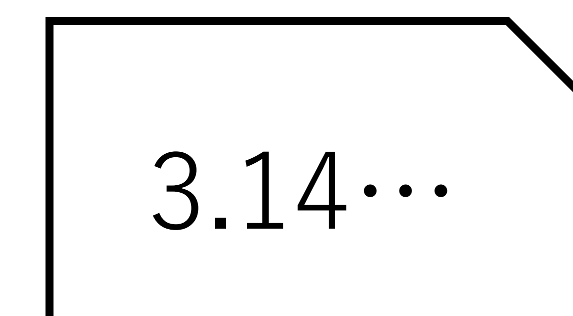


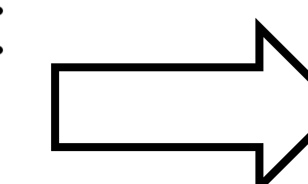
バイナリや
コードの改変











インストルメント
コード

- イベントログの記録
- 命令・操作時間の計測
- メモリ・スタック状態の追跡

分析結果

実行時間: …
メモリ使用量: …
コールグラフ: …

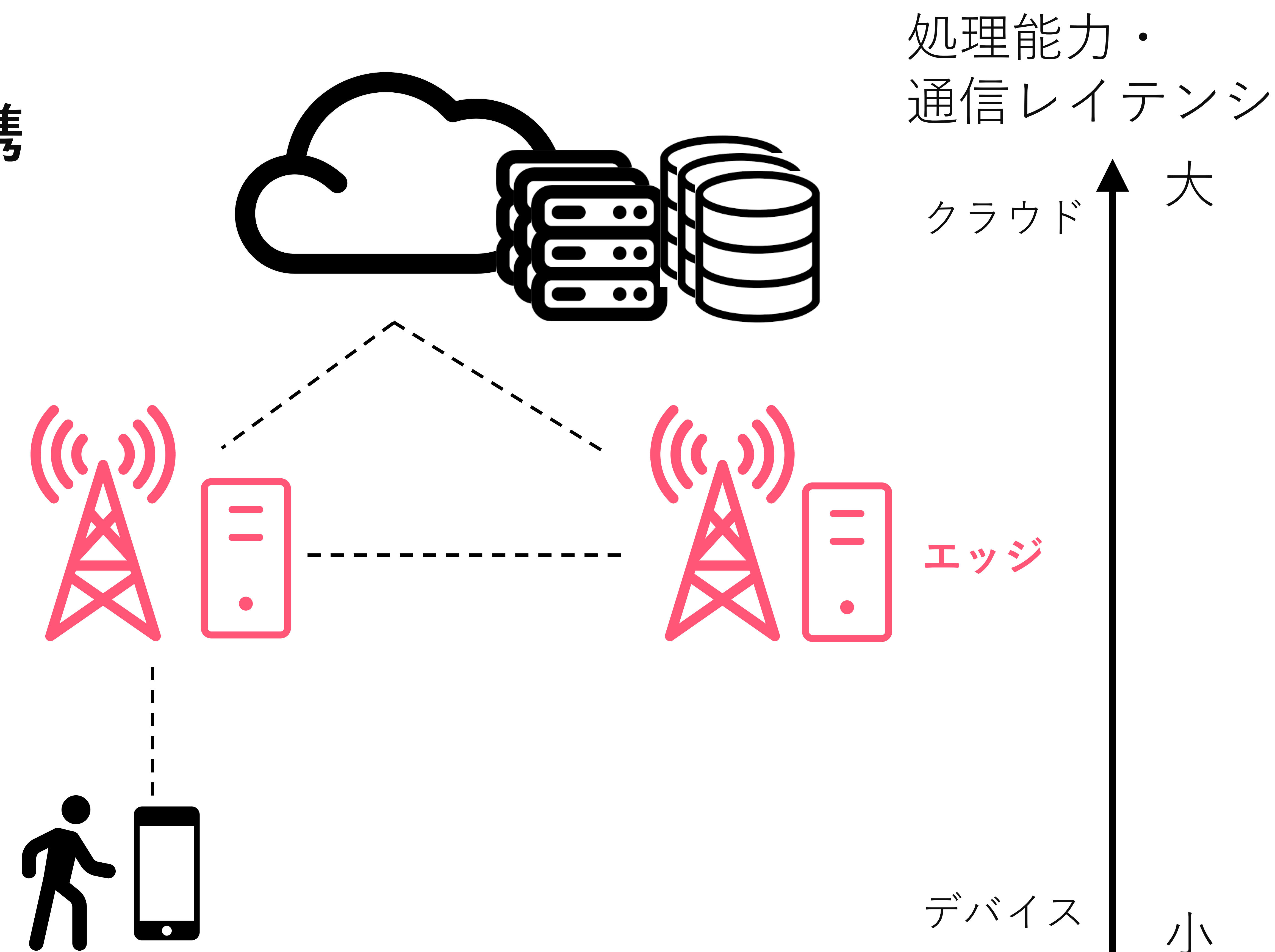
- 静的解析によるテストより詳細・正確な分析
  - バグの調査やフローの可視化、プロファイリング、リアルタイム監視に活用

SAKURA internet

エッジコンピューティングにおけるライブマイグレーションへの応用(1/2)

ユーザ近傍のエッジサーバを活用した計算環境[1]

- **デバイス・エッジサーバ・クラウドが連携**
  - **エッジサーバによる低レイテンシな処理**
  - **クラウドによる高速な処理**
- 連携処理手法
  - タスクのオフローディング
  - ユーザの移動や通信状況の変化による アプリケーションのハンドオーバ

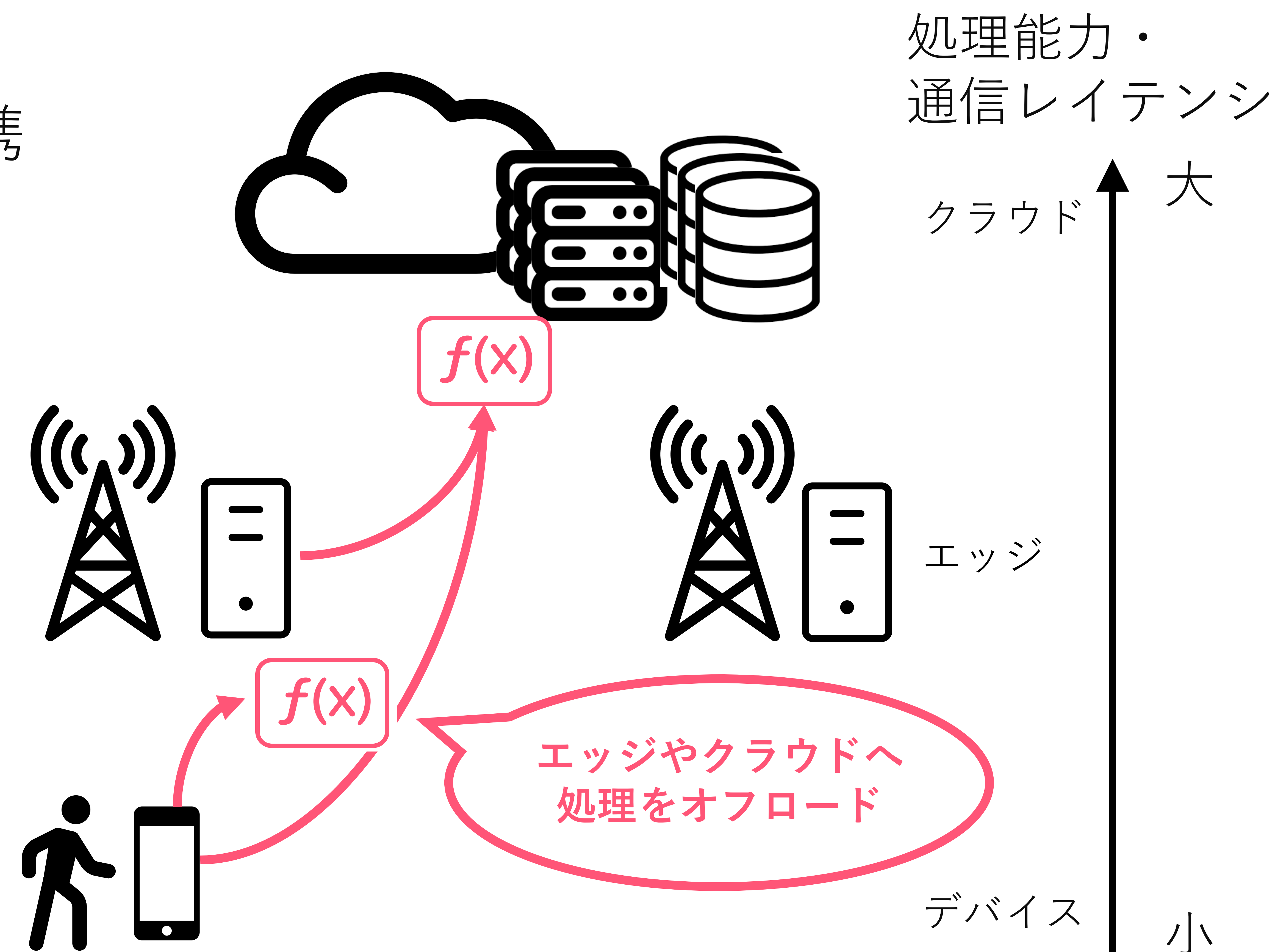


[1]Keyan Cao, Yefan Liu, Gongjie Meng, and Qi-meng Sun. An overview on edge computing research. IEEE Access, Vol. 8, pp. 85714–85728, 2020

エッジコンピューティングにおけるライブマイグレーションへの応用(1/2)

ユーザ近傍のエッジサーバを活用した計算環境[1]

- デバイス・エッジサーバ・クラウドが連携
  - エッジサーバによる低レイテンシな処理
  - クラウドによる高速な処理
- **連携処理手法**
  - **タスクのオフローディング**
  - **ユーザの移動や通信状況の変化による アプリケーションのハンドオーバ**

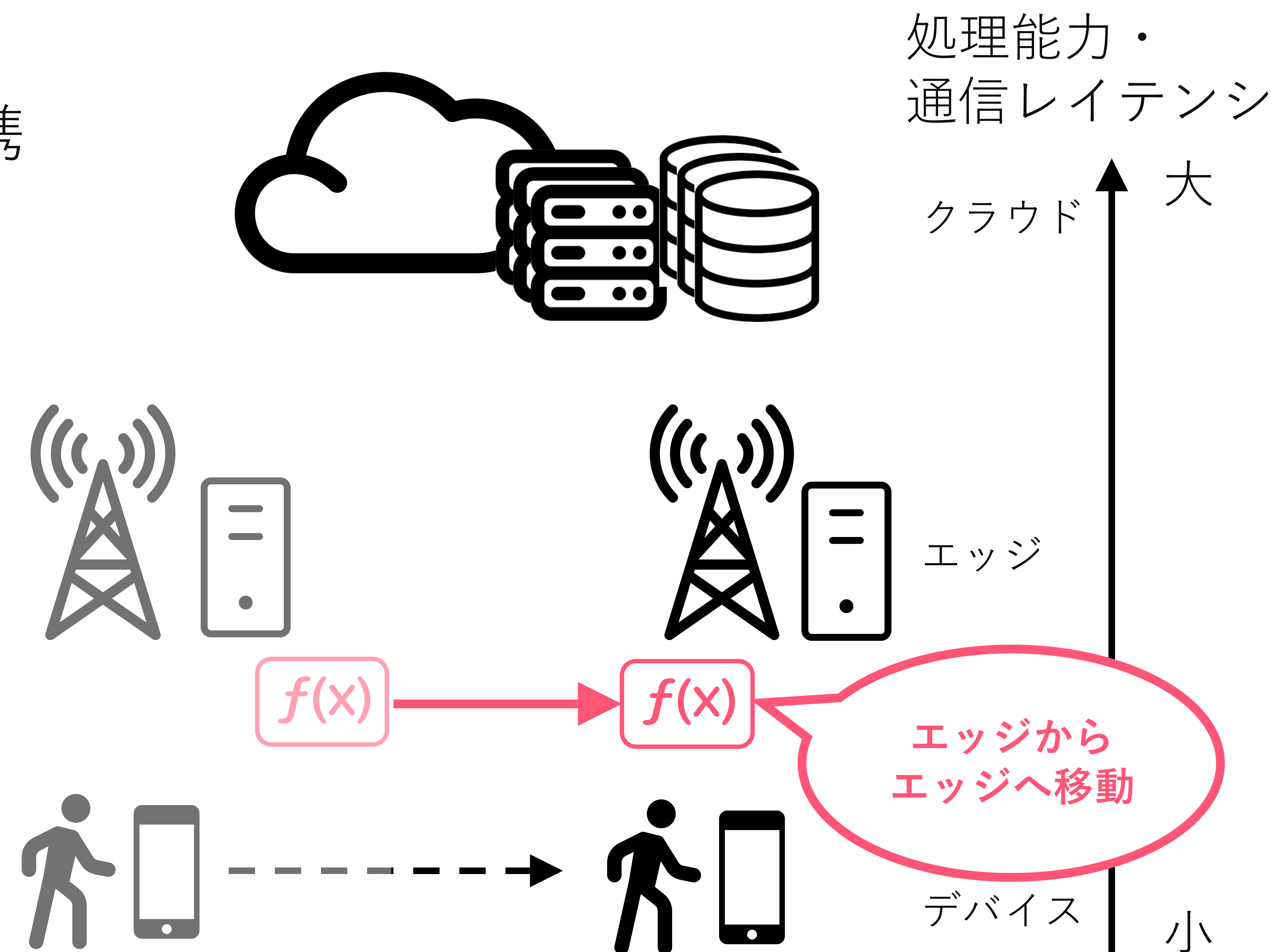


[1]Keyan Cao, Yefan Liu, Gongjie Meng, and Qi-meng Sun. An overview on edge computing research. IEEE Access, Vol. 8, pp. 85714–85728, 2020

エッジコンピューティングにおけるライブマイグレーションへの応用(1/2)

ユーザ近傍のエッジサーバを活用した計算環境[1]

- デバイス・エッジサーバ・クラウドが連携
  - エッジサーバによる低レイテンシな処理
  - クラウドによる高速な処理
- **連携処理手法**
  - **タスクのオフローディング**
  - **ユーザの移動や通信状況の変化による アプリケーションのハンドオーバ**



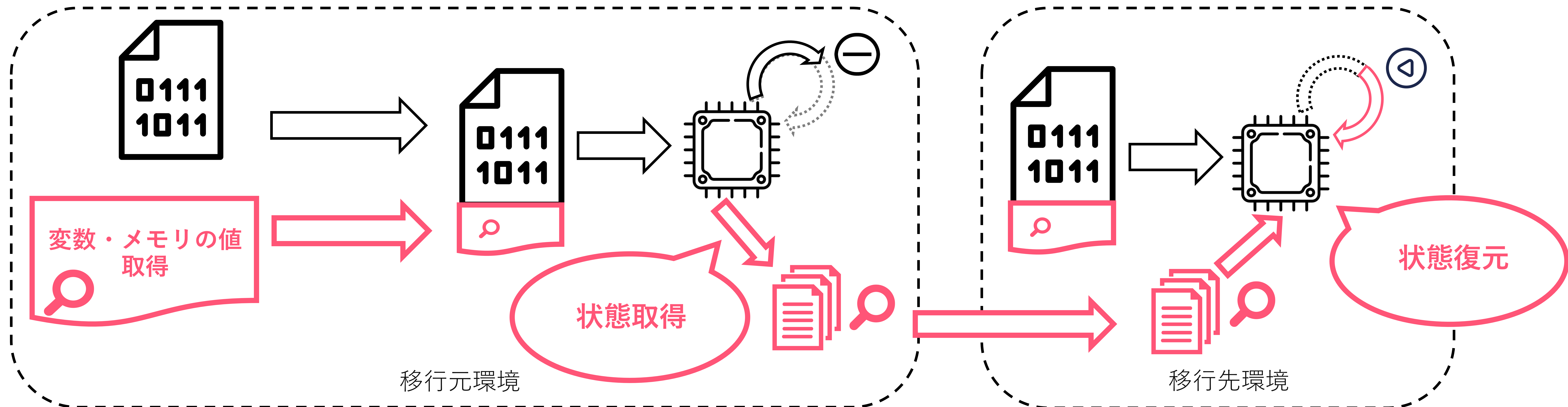
[1]Keyan Cao, Yefan Liu, Gongjie Meng, and Qi-meng Sun. An overview on edge computing research. IEEE Access, Vol. 8, pp. 85714–85728, 2020

SAKURA internet

エッジコンピューティングにおける**ライブマイグレーションへの応用**(2/2)

状態取得・復元によってオフローディングやハンドオーバを実現

- アプリケーションの処理状態を維持したまま別環境へ移動
  - 移行元で内部状態を取得、移行先で内部状態の復元

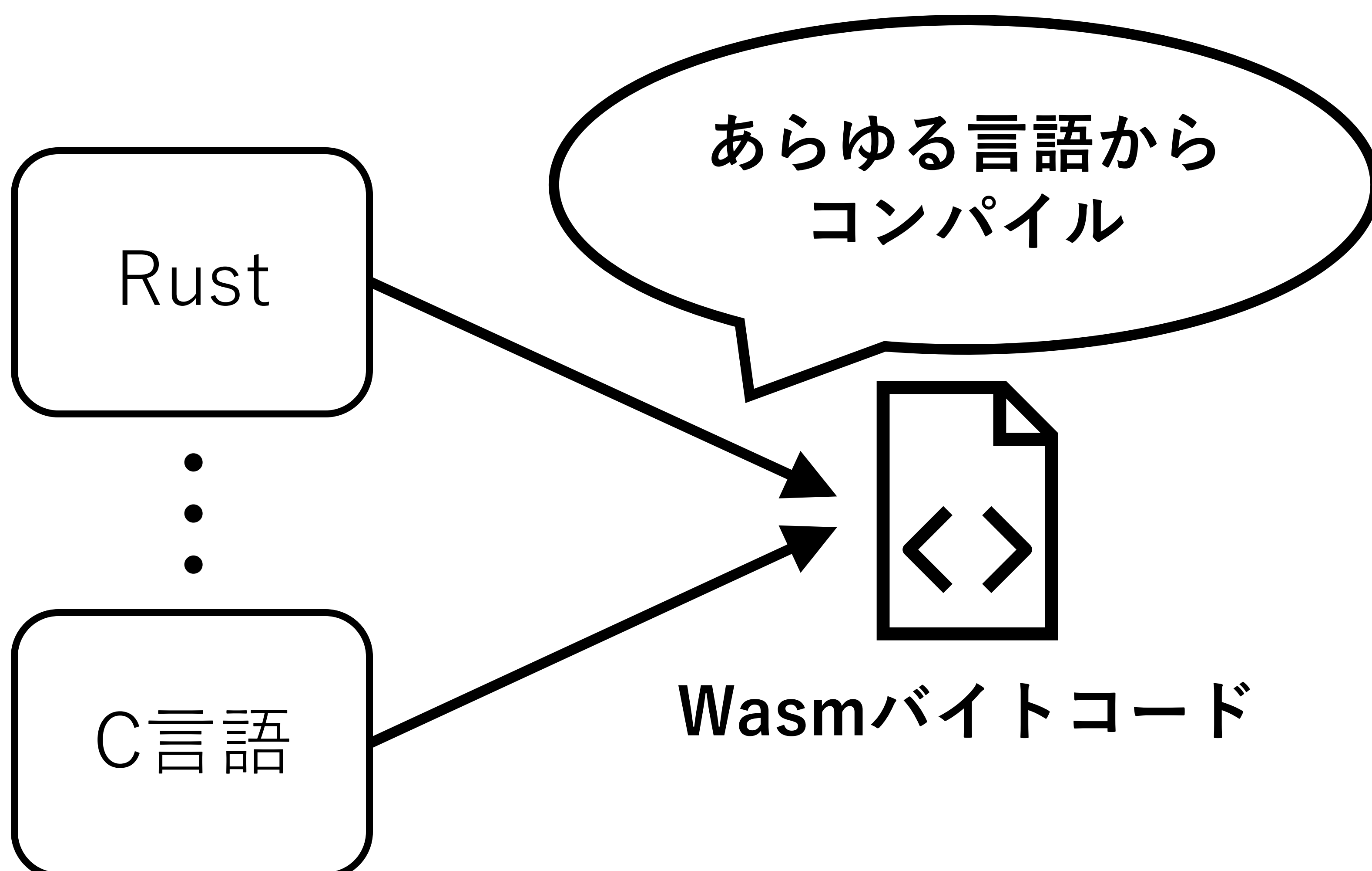


SAKURA internet

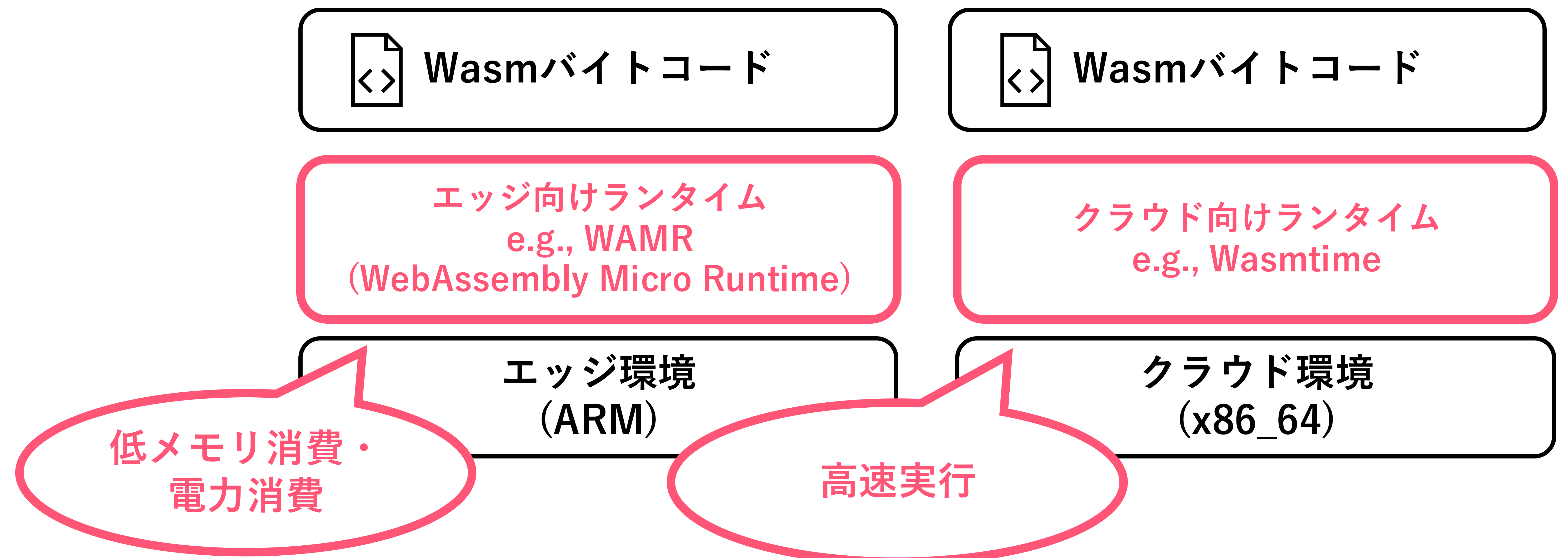
エッジコンピューティングにおけるアプリケーション実行環境

WebAssembly（Wasm）[2]
統一な実行環境によるシームレスなオフロード・ハンドオーバ

仮想命令セットアーキテクチャ



多様な環境で動作するVM・
各用途に特化したランタイム実装

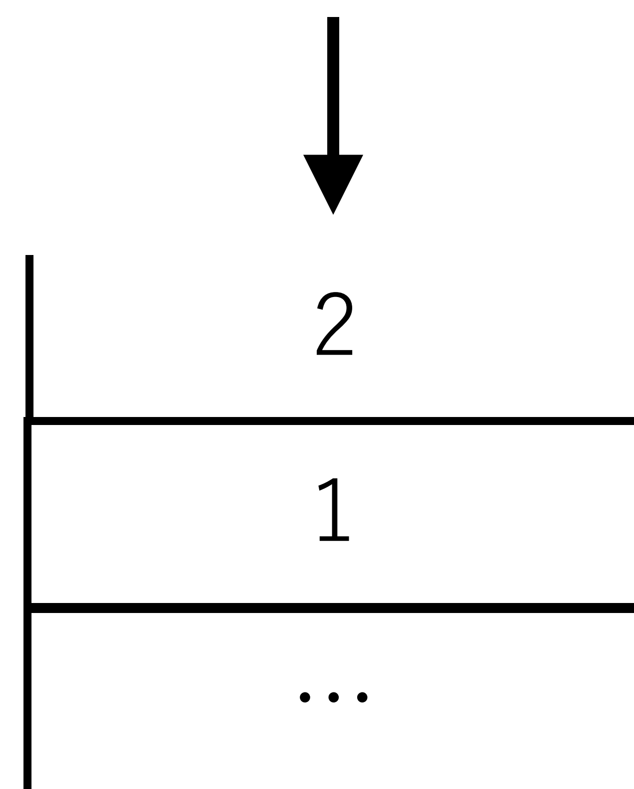


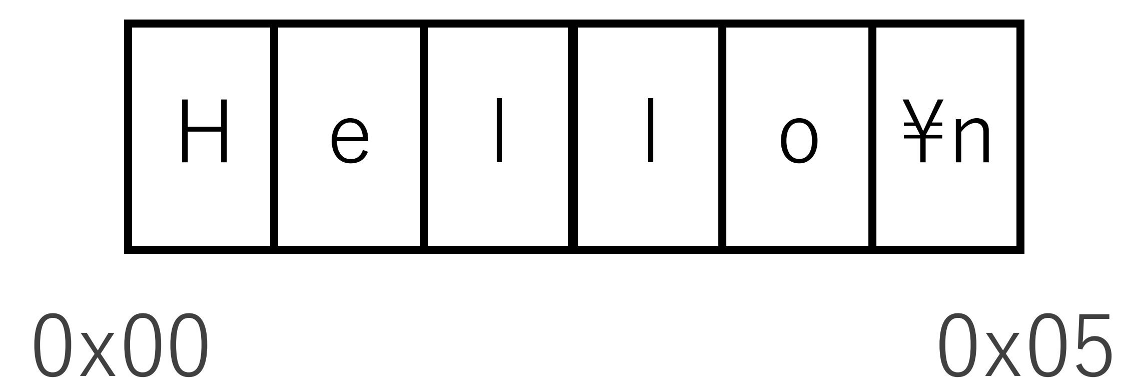
[2] WebAssembly Community Group. Webassembly specification — webassembly 2.0 (draft 2024-08- 11). https://webassembly.github.io/spec/core/, 2022. (Accessed on 08/12/2024).

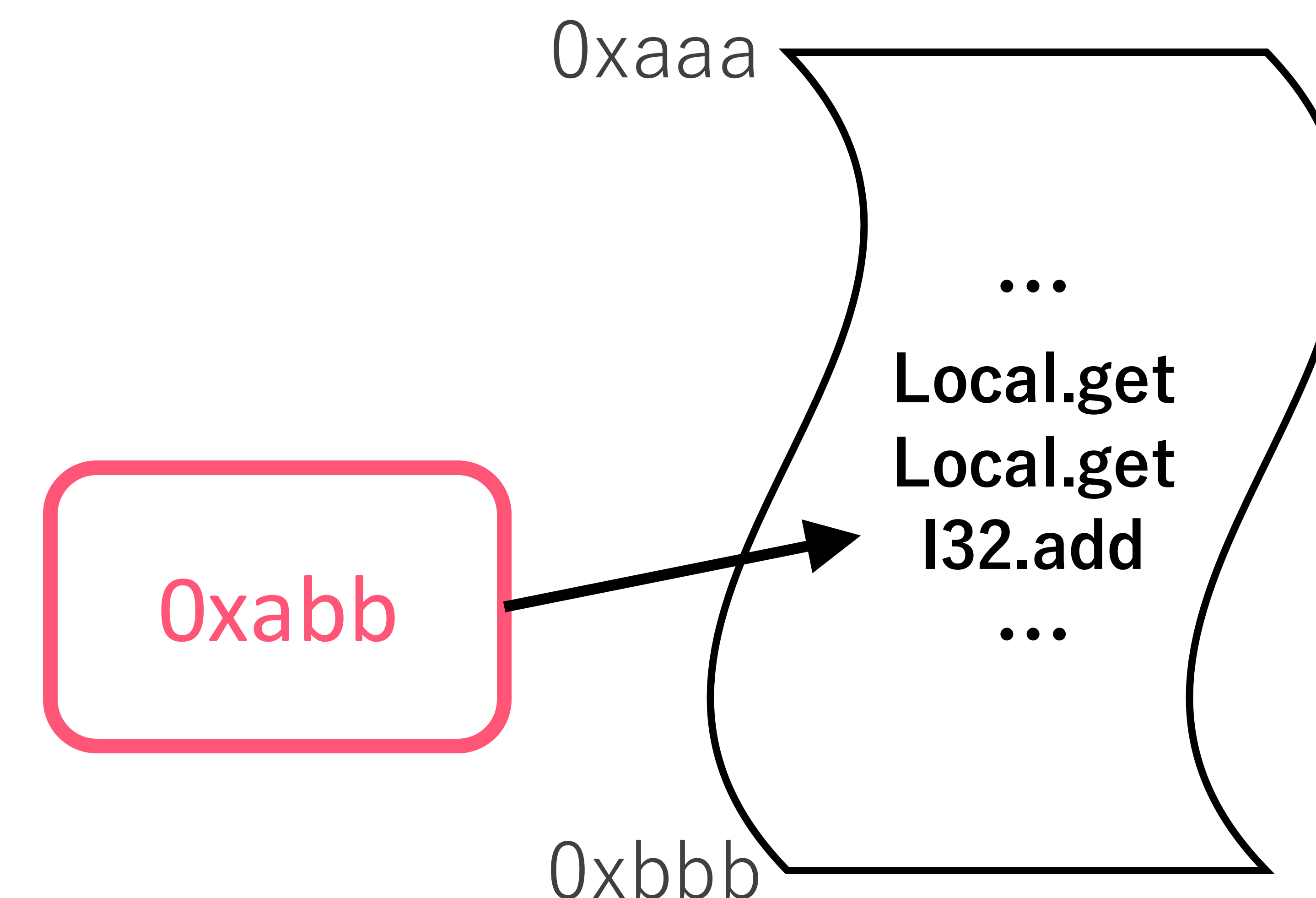
SAKURA internet

エッジVMマイグレーションに適した Wasmバイトコードインストルメンテーションの要件

R1. マイグレーションに必要な内部状態の取得







数値型を格納する
値スタック

関数の情報を格納する
フレームスタック

スタック

配列や文字列など
数値型以外のデータを格納

線形メモリ

次に実行する
バイトコードのアドレス

プログラムカウンタ

SAKURA internet

エッジVMマイグレーションに適した Wasmバイトコードインストルメンテーションの要件

R2. 多様なWasmランタイムや実行方式に非依存

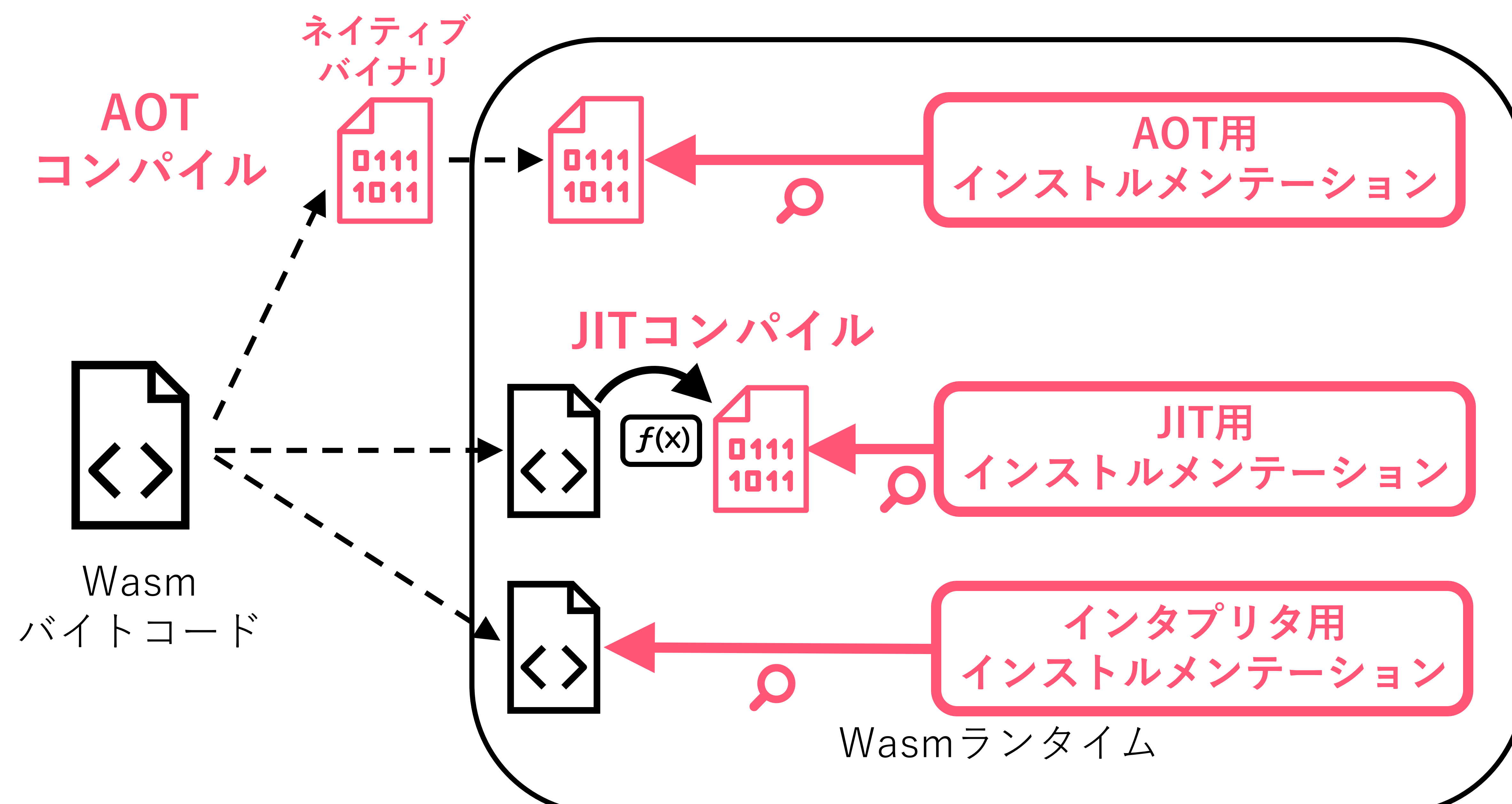
デバイスやエッジ・クラウドで 最適なWasmランタイムが異なる

各ランタイムに適した形での

インストルメンテーション実装は複雑

Wasmランタイムごとに採用する 実行方式が異なる

各実行方式の差異に合わせて実装するのは困難



エッジVMマイグレーションに適した Wasmバイトコードインストルメンテーションの要件

R3. アプリケーション性能に与える影響の軽減

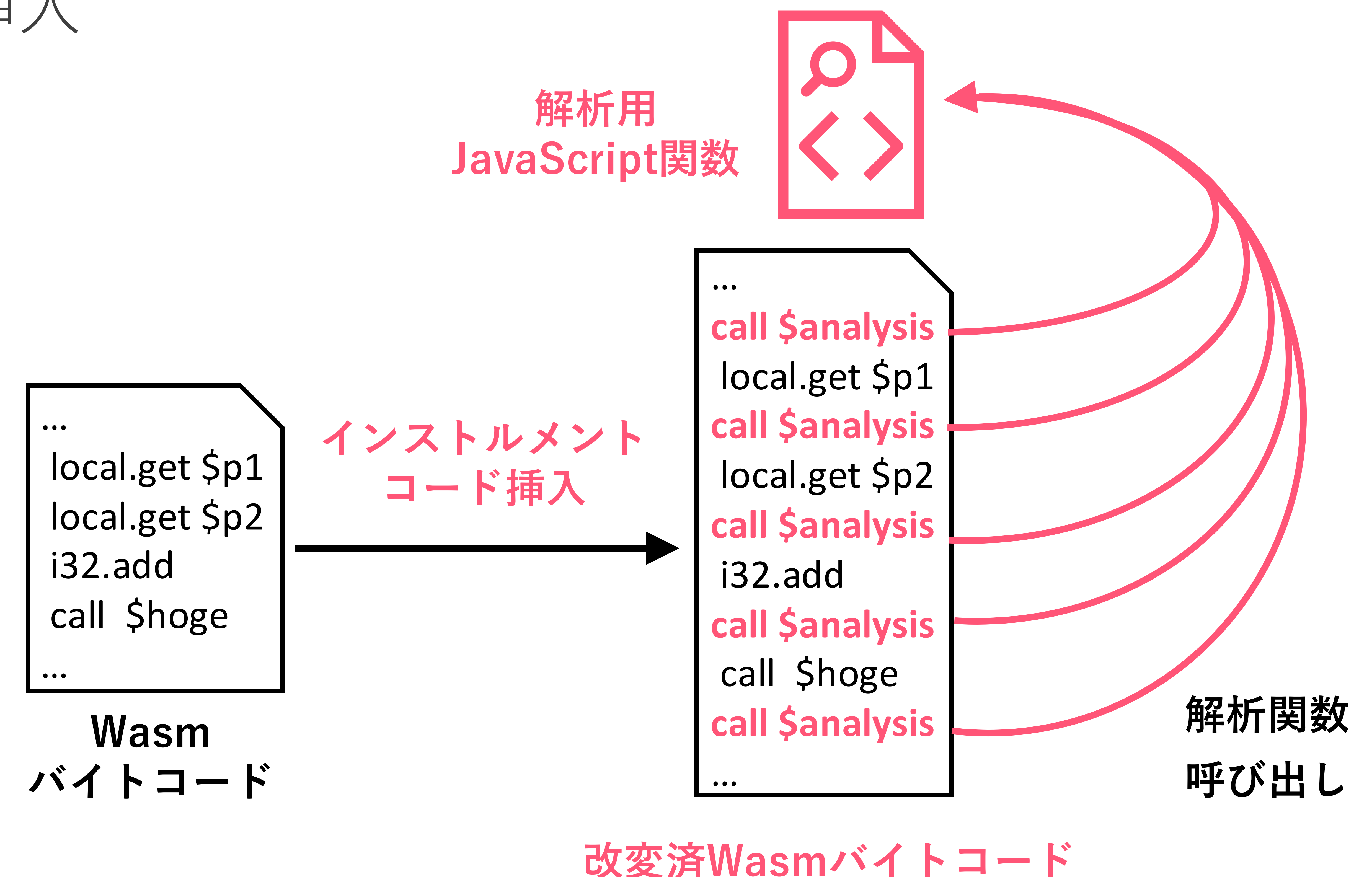
- 従来のバグの調査やパフォーマンス分析と目的が異なる
  - **テスト環境ではなく、実際のアプリケーションが動作する環境で使用を想定**
- **オフローディングやハンドオーバはアプリケーション処理の最適化のため**
  - インストルメンテーションによる性能低下は本来の目的達成を妨げる

SAKURA internet

Wasabi[3]：静的なインストルメントコード挿入手法

Pros: ランタイムや実行方式に非依存な内部状態の取得(R1,2)
Cons: インストルメンテーションによる大幅な性能低下(R3)

- 実行前にインストルメントコードを挿入
  - 命令単位や関数呼び出し前後
  - **ランタイムや実行方式に非依存**
- **アプリケーション性能が大幅に低下**
  - 解析処理をJavaScriptで記述し
    インストルメントコードから呼び出し
  - **大量のJavaScript関数の呼び出し**



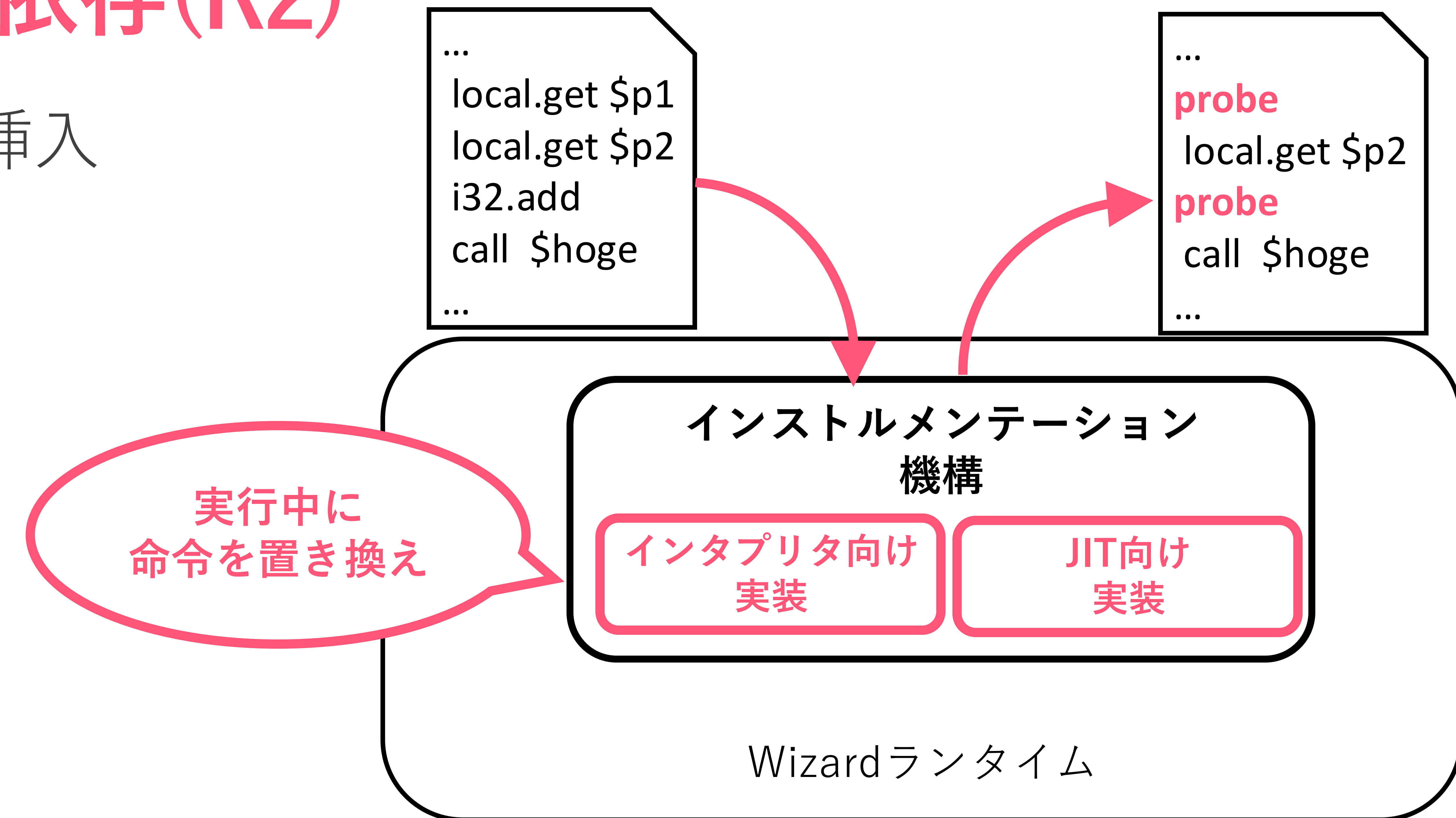
[3] Daniel Lehmann and Michael Pradel. Wasabi: A framework for dynamically analyzing webassembly. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’19, p. 1045–1058, New York, NY, USA, 2019. Association for Computing Machinery.

SAKURA internet

Wizard[4]：ランタイム内での動的なインストルメンテーション手法

Pros: 高性能なインストルメンテーション(R1, 3)
Cons: ランタイム・実行方式に依存(R2)

- 実行中にインストルメントコードを挿入
  - 計測対象の命令を置き換え
  - **対象外の命令はそのまま実行し
実行性能に与えるオーバヘッドを軽減**
- 実行方式に依存
  - インタプリタとJITコンパイルをサポート
  - **実行方式ごとに異なるインストルメンテーション機構**



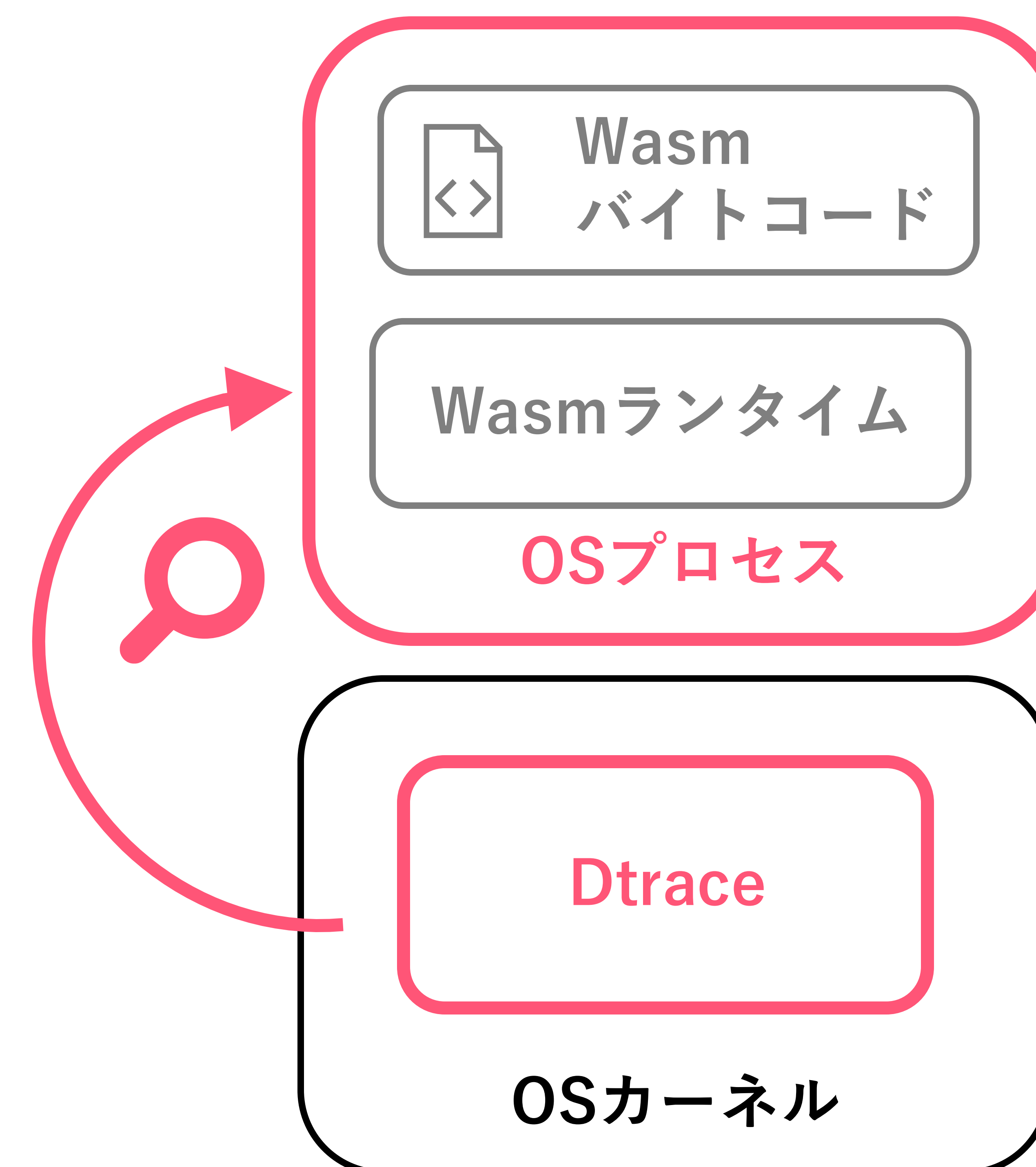
[4] Ben L. Titzer, Elizabeth Gilbert, Bradley Wei Jie Teo, Yash Anand, Kazuyuki Takayama, and Heather Miller. Flexible non-intrusive dynamic instrumentation for webassembly. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS '24, p. 398–415, New York, NY, USA, 2024. Association for Computing Machinery.

SAKURA internet

Dtrace[5]: Wasmランタイム外でのインストルメンテーション手法

Pros: OSカーネル内での高速なインストルメンテーション(R3)
Cons: 状態取得が複雑(R1)・実行方式への依存(R2)

- アプリケーションやOSカーネルなどの状態をリアルタイムに収集
- **カーネル内動作で性能に与えるオーバヘッドを軽減**
- **ランタイムとアプリケーションの状態が混在**
  - アプリケーションの状態のみを取得する必要
  - 実行方式に合わせて内部状態も変化
    - **アプリケーションの状態のみを取得するのは困難**



[5] Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Leventhal. Dynamic instrumentation of production systems. In 2004 USENIX Annual Technical Conference (USENIX ATC 04), Boston, MA, June 2004. USENIX Association.

SAKURA internet

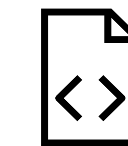
本研究の目的と提案

目的

- **エッジVMマイグレーションに適した Wasmバイトコードインストルメンテーション機構**
  - VMマイグレーションに適した内部状態の取得(R1)
  - ランタイム・実行方式非依存(R2)
  - アプリケーション性能に与えるオーバヘッドを軽減(R3)

提案

- **セルフホスト型Wasmランタイムによるインストルメンテーション**
  - **インストルメンテーション機構を備えたWasmランタイムをセルフホスト化**
  - **任意のWasmランタイム上でセルフホスト型Wasmランタイムを用いてアプリケーションを実行**

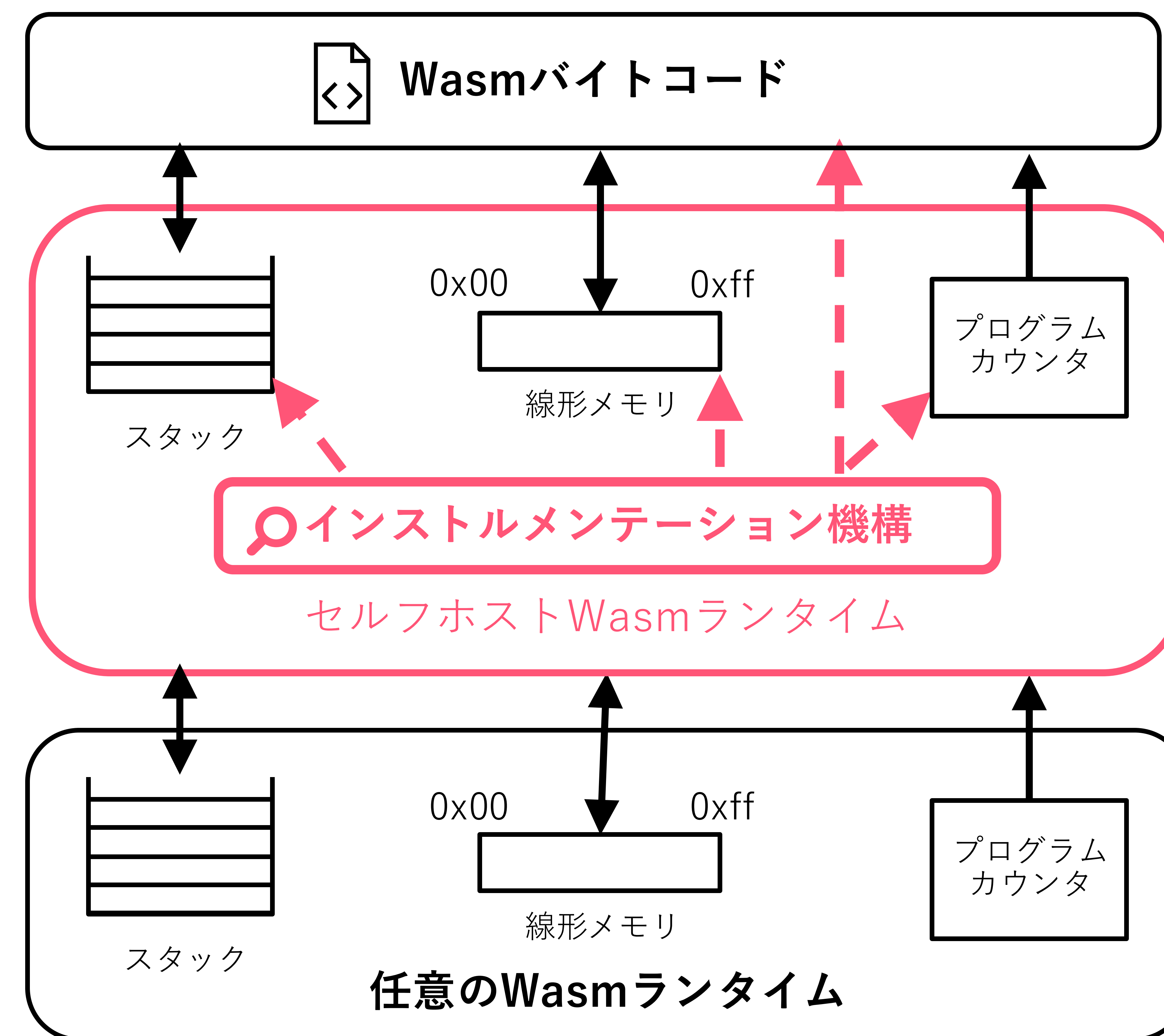


SAKURA internet

提案するセルフホスト型Wasmランタイムの概要

セルフホストWasmランタイムに
インストルメンテーション機構を実装

- ランタイムのためマイグレーションに必要な状態の取得が容易(R1)
- 任意のランタイムで実行
  - ランタイム・実行方式に依存したインストルメンテーション機構が不要(R2)
- **ランタイムを二重実行による性能低下の可能性がある**
  - **オーバヘッドを明らかにしてR3を評価する必要**



SAKURA internet

実現可能性評価1：セルフホスト環境におけるアプリケーション性能

アプリケーション性能へのオーバヘッドを明らかにする

- ベンチマークプログラム
  - The Computer Language24.06 Benchmarks Gameのマンデルブロ集合の計算と描画プログラム[6]
- セルフホスト化したWasm3で性能評価
  - セルフホストをサポート、高速インタプリタ方式
- **Wasm3、Wasmtime(JITコンパイル)でベンチマークプログラムを直接実行した場合と比較**

OS	Ubuntu 22.04.4 LTS(Linux 5.15.0)
CPU	Intel Xeon Silver 4208 8Core 2.10GHz
メモリ	32 GB
ストレージ	SSD 480GBx2 (RAID1)

実験環境

（さくらの専用サーバPHY RX2530 M5 8コア 1CPU）

[6] DaIsaac Gouy. mandelbrot rust #4 program (benchmarks game). https://benchmarksgame-team.pages.debian.net/benchmarksgame/program/mandelbrot-rust-4. html. (Accessed on 08/15/2024).

SAKURA internet

セルフホスト環境と通常実行環境でのアプリケーション性能比較結果

セルフホスト環境は
アプリケーション実行にかかる時間が大幅に増加

- セルフホスト環境におけるオーバヘッドの推測
  - **ベンチマークプログラムに加えて**
    セルフホストWasm3の命令も解釈・実行する必要
  - **実行しなければならない命令の数が増大**
  - WasmtimeのJITコンパイルが適用されるのは
    Wasm3のインタプリタ処理のみ
    - ベンチマークプログラムには適用されない

計測対象	実行時間（秒）
Wasm3	83.53
セルフホストWasm3 on Wasm3	**3125.61**
Wasmtime	14.02
セルフホストWasm3 on Wasmtime	**1579.32**

37倍

112倍

SAKURA internet

実現可能性評価2：セルフホスト型と静的なインストルメントコード挿入手法での性能比較

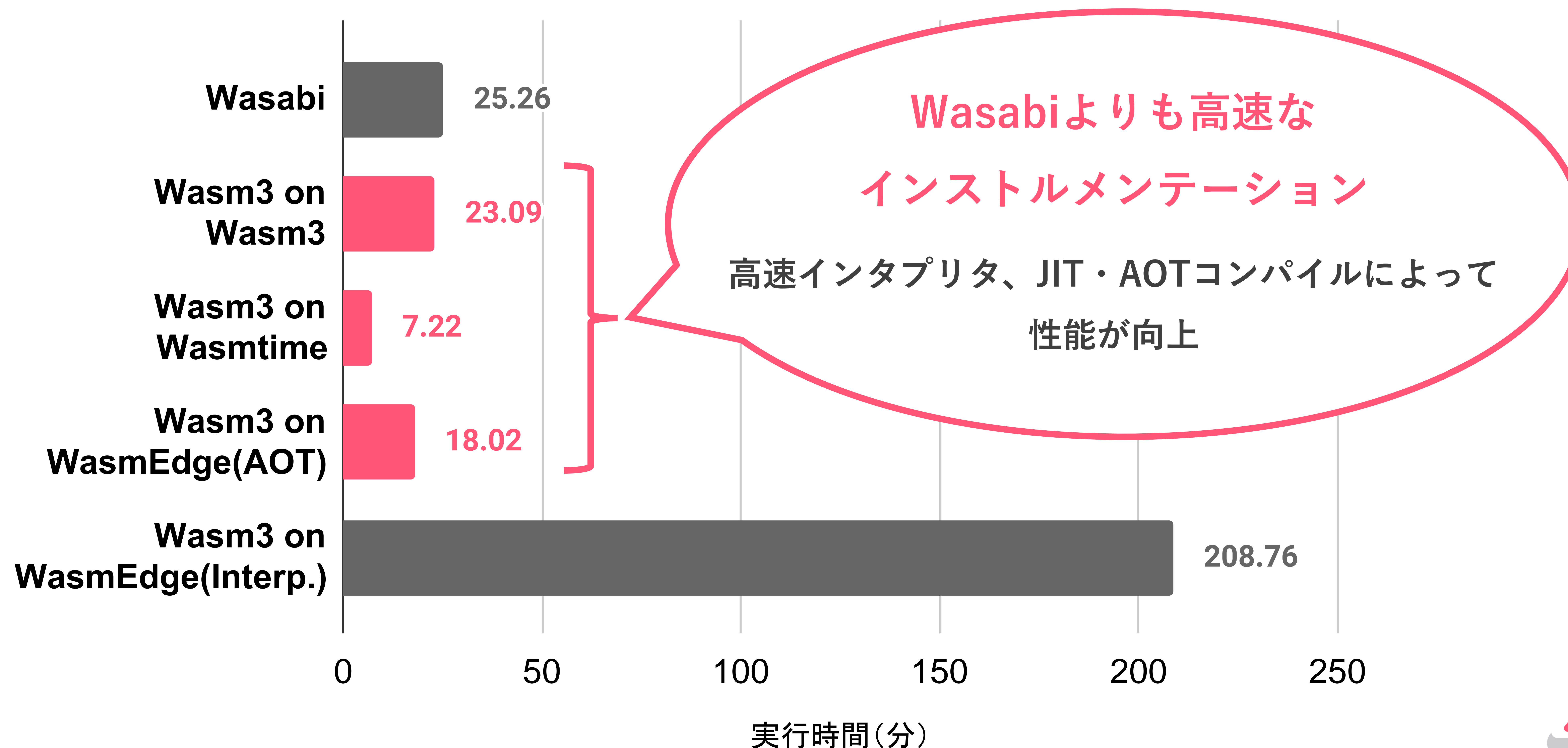
セルフホストのインストルメンテーション性能を明らかにする

- **セルフホストWasm3にインストルメンテーション機構を実装**
- **Wasabiとセルフホストで Wasm3でインストルメンテーションを実行**
  - 100万項のライプニッツ級数を用いた円周率計算
  - メモリアクセス命令を解析し使用したメモリのページサイズの合計を計算
- セルフホストWasm3の実行に使用したランタイム
  - Wasm3、Wasmtime、WasmEdge（インタプリタ・AOTコンパイル）
- Wasabiの実行にはNode.js（JITコンパイル）を使用

SAKURA internet

セルフホストWasm3とWasabiの性能比較結果

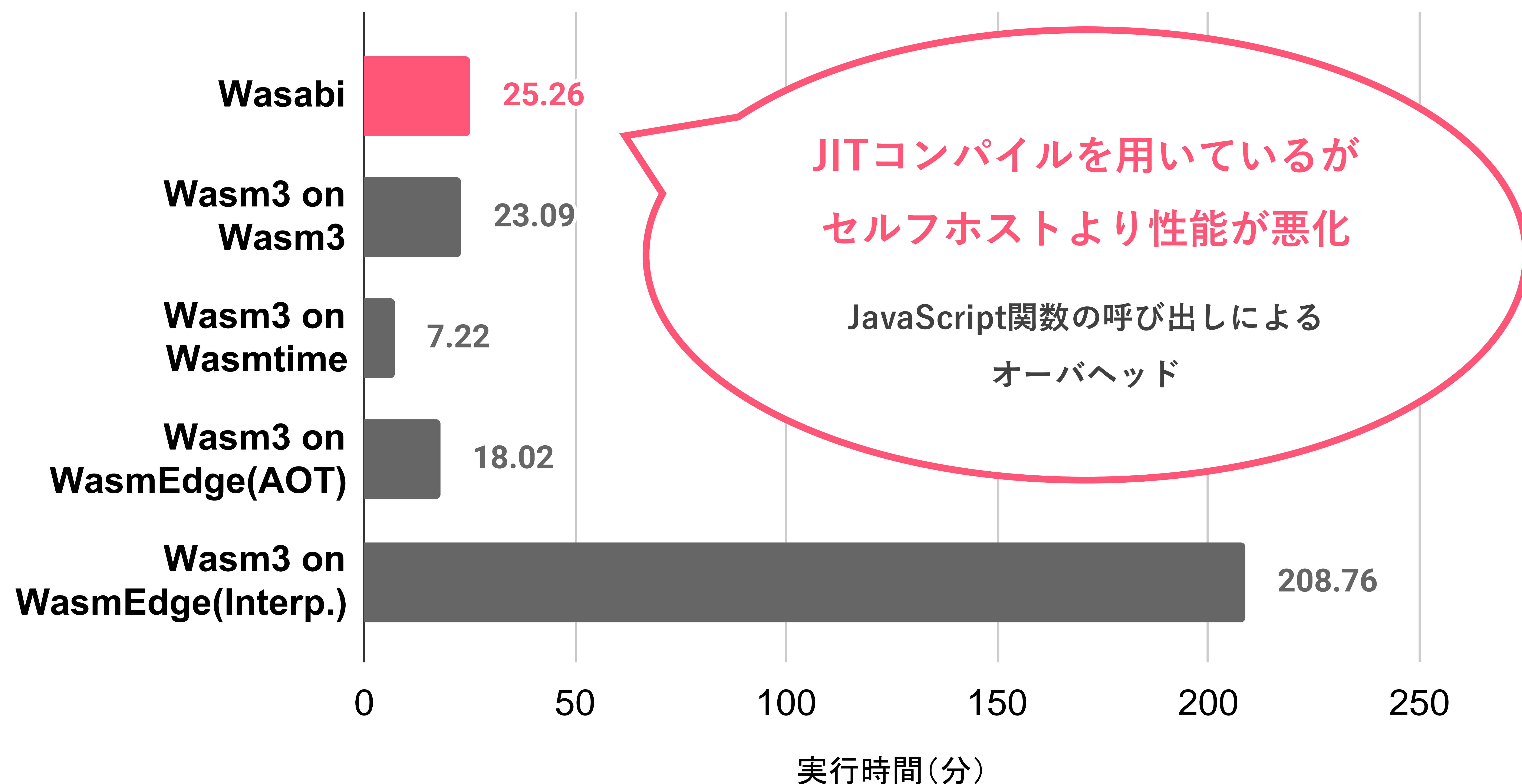
セルフホストWasm3でのインストルメンテーションは実行ランタイムに合わせて性能が変化



SAKURA internet

セルフホストWasm3とWasabiの性能比較結果

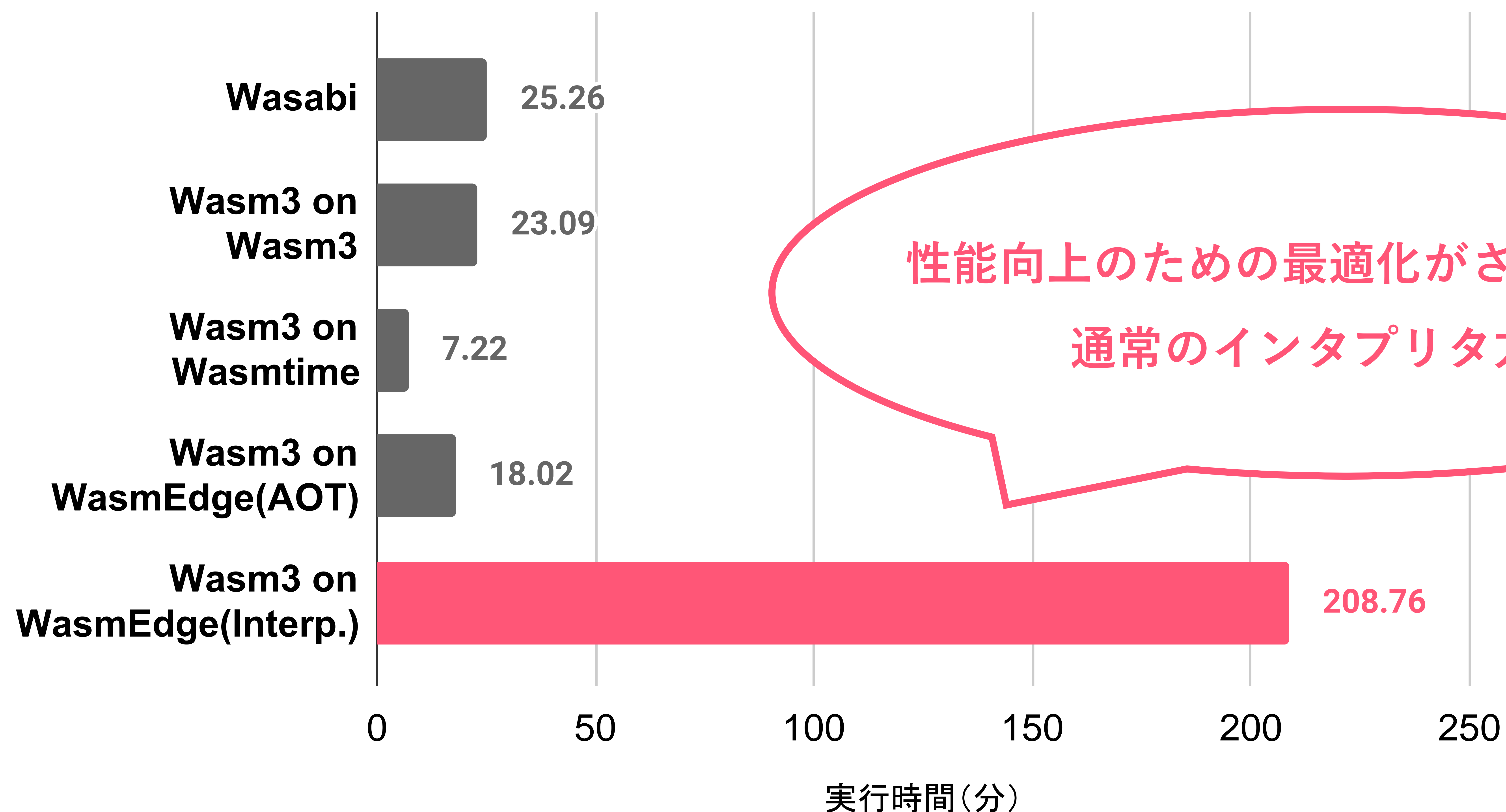
セルフホストWasm3でのインストルメンテーションは実行ランタイムに合わせて性能が変化



SAKURA internet

セルフホストWasm3とWasabiの性能比較結果

セルフホストWasm3でのインストルメンテーションは実行ランタイムに合わせて性能が変化



SAKURA internet

評価1･2のまとめ

既存手法に比べて高速であるが
インストルメンテーションなし場合に比べて大幅に性能が低下

- セルフホスト化した既存のWasmランタイムを用いた場合
  通常の実行環境に比べて大幅に低下
- Wasabiより良いインストルメンテーション性能
  - 使用するランタイムに合わせてセルフホスト型ランタイムの性能も変化

R3を満たすには

セルフホスト型を前提とした最適化ランタイムが必要

セルフホスト化による処理の差異に関する調査

オーバヘッドの要因を明らかにし
セルフホストを前提としたランタイム設計に活用

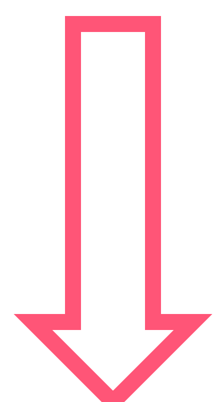
- **実行CPU総命令数と呼び出し関数の差を計測**
  - LinuxのパフォーマンスツールPerfを使用
  - 評価2で使用したベンチマークプログラムで計測
- 比較対象
  - Wasm3
  - セルフホストWasm3 on Wasm3

SAKURA internet

セルフホスト化による処理の差異に関する調査結果

実行CPU総命令数が大幅に増加し、呼び出す関数の傾向も変化

794倍



計測対象	総命令数
Wasm3	400,849,582,525
Wasm3 on Wasm3	**318,276,978,517,504**

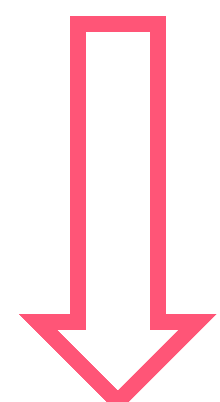
	Wasm3	Wasm3 on Wasm3
1	op_SetSlot_f64 (28.22%)	op_CopySlot_32(16.57%)
2	op_f64_Add_rs(22.15%)	op_i32_Load_i32_s(15.51%)
3	op_f64_Multiply_rs(21.53%)	op_CallIndirect(8.71%)
4	op_f64_Multiply_ss(7.22%)	op_SetSlot_i32(8.36%)
5	op_f64_Subtract_rs(6.41%)	op_Entry(6.89%)

SAKURA internet

セルフホスト化による処理の差異に関する調査結果

実行CPU総命令数が大幅に増加し、呼び出す関数の傾向も変化

794倍



計測対象
W
Wasm

算術演算処理が
全体の57.31%を占める

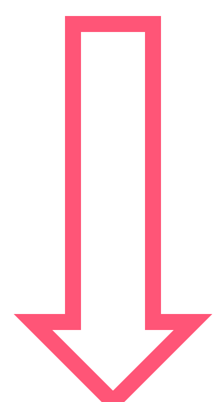
	Wasm3	Wasm3 on Wasm3
1	op_SetSlot_f64 (28.22%)	op_CopySlot_32(16.57%)
2	**op_f64_Add_rs(22.15%)**	op_i32_Load_i32_s(15.51%)
3	**op_f64_Multiply_rs(21.53%)**	op_CallIndirect(8.71%)
4	**op_f64_Multiply_ss(7.22%)**	op_SetSlot_i32(8.36%)
5	**op_f64_Subtract_rs(6.41%)**	op_Entry(6.89%)

SAKURA internet

セルフホスト化による処理の差異に関する調査結果

実行CPU総命令数が大幅に増加し、呼び出す関数の傾向も変化

794倍



計測対象	総命令数
Wasm3	400,849,582,525
Wasm3 on Wasm3	**318,276,978,517,504**

	Wasm3	Wasm3 on Wasm3
1	op_SetSlot_f64 (28.22%)	op_CopySlot_32(16.57%)
2	op_f64_Add_rs(22.15%)	**op_i32_Load_i32_s(15.51%)**
3	op_f64_Multiply_rs(21.53%)	**op_CallIndirect(8.71%)**
4	[illegible](7.22%)	op_SetSlot_i32(8.36%)
5	[illegible]	**op_Entry(6.89%)**

線形メモリや
制御命令処理が増加

SAKURA internet

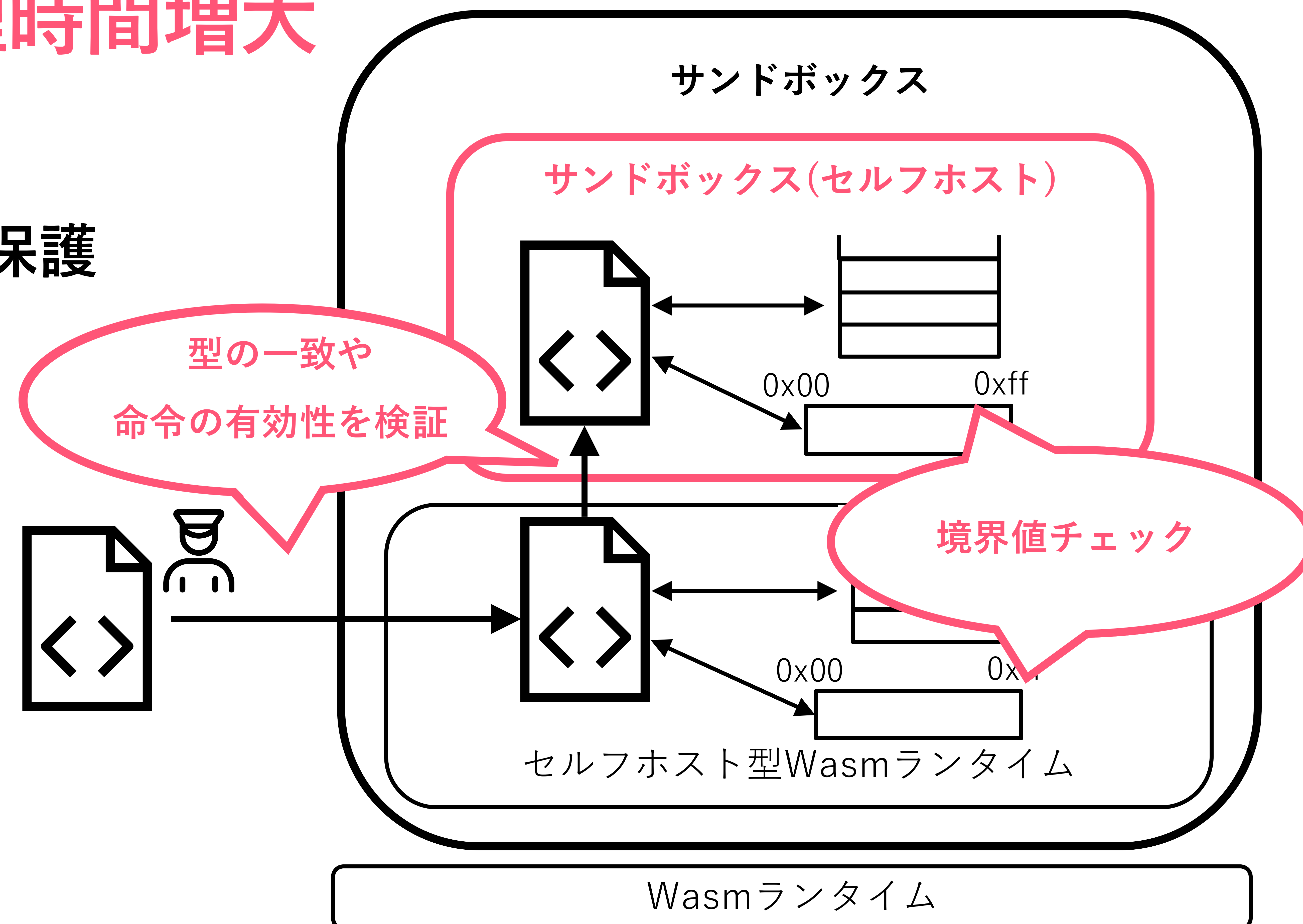
セルフホスト型を前提としたランタイム設計の議論(1/2)

処理変化の影響：
二重サンドボックスによる処理時間増大

サンドボックス：

悪意のあるプログラムから実行環境を保護

- 実行前：バイトコード検証と
  実行中：スタック・線形メモリの保護
- **既知の実行時性能のオーバヘッド**[7, 8]
- **メモリに関する処理が増加した結果**
  - **境界値チェック処理にかかる時間も増大**



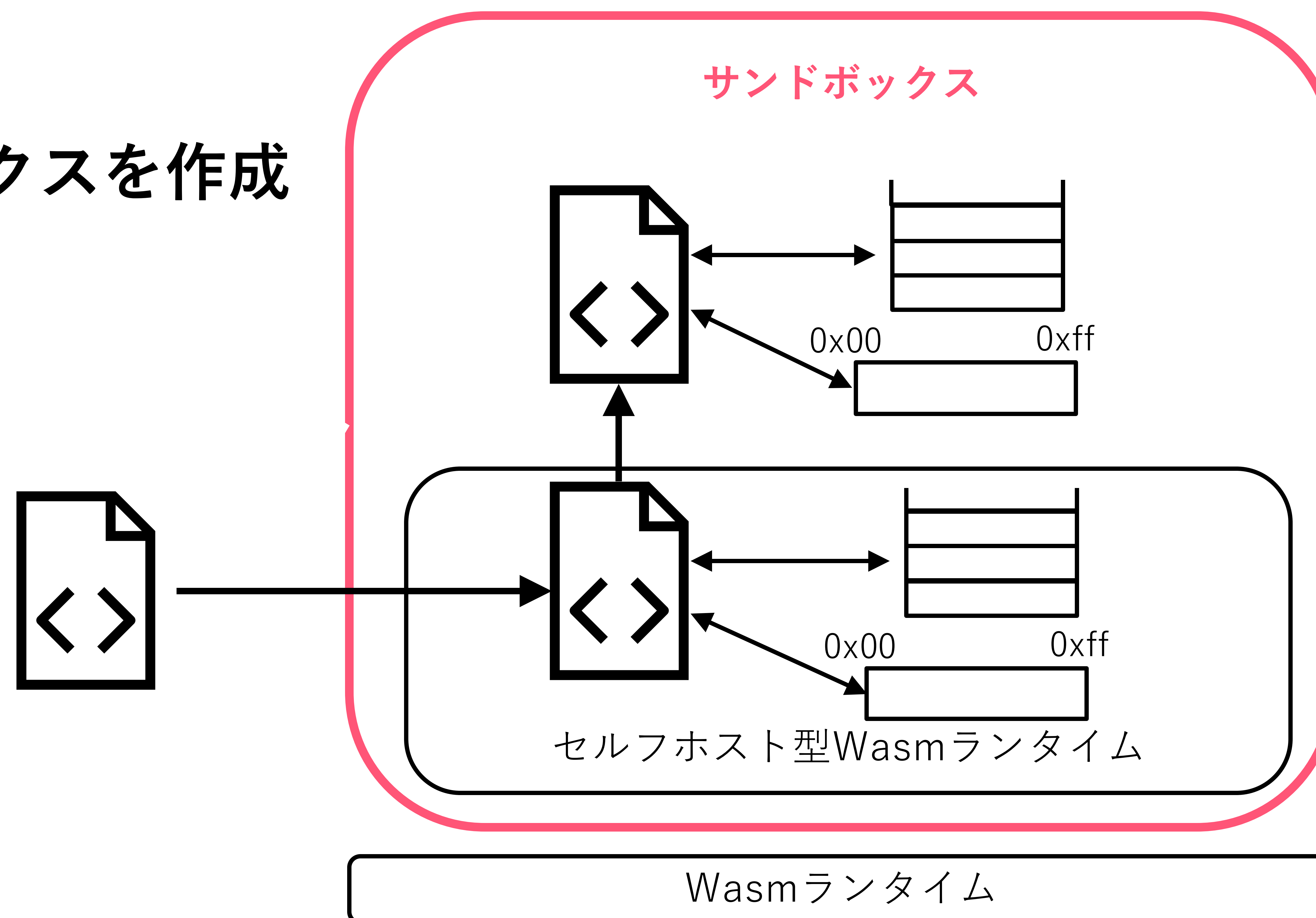
[7] Matthew Kolosick, Shravan Narayan, Evan Johnson, Conrad Watt, Michael LeMay, Deepak Garg, Ranjit Jhala, and Deian Stefan. Isolation without taxation: near-zero-cost transitions for webassembly and sfi. Proc. ACM Program. Lang., Vol. 6, No. POPL, jan 2022.
[8] Raven Szewczyk, Kimberley Stonehouse, Antonio Barbalace, and Tom Spink. Leaps and bounds: Analyzing webassembly 's performance with a fo- cus on bounds checking. In 2022 IEEE Interna- tional Symposium on Workload Characterization (IISWC), pp. 256–268, 2022.

SAKURA internet

セルフホスト型を前提としたランタイム設計の議論(2/2)

セルフホスト型ランタイムのサンドボックス機構の削減

- **セルフホスト環境を実行する Wasmランタイムのみサンドボックスを作成**
  - 二重処理を排除
- 実行するWasmランタイムのサンドボックス機構を活用して安全性を維持



まとめ

- **セルフホスト型Wasmランタイムによる Wasmバイトコードインストルメンテーション機構を提案**
  - ランタイムとして実装することでマイグレーションに必要な状態を容易に取得
  - ランタイム・実行方式に合わせたインストルメンテーション機構の実装が不要
- **既存の静的手法よりアプリケーション性能は高速であるが 通常実行時に比べて大きく低下**
- **ランタイム間で重複している機構の削減を検討**
- 今後の展望
  - 二重のサンドボックス機構を排除したセルフホスト最適化ランタイムの実装を進める
  - セルフホストを前提としたバイトコードの命令解釈・実行処理の高速化手法の検討

SAKURA internet