Make TrustZone Great Again

高野 祐輝[1]、金谷 延幸[2]、津田 侑[2]

1 大阪大学

2 情報通信研究機構

発表の流れ

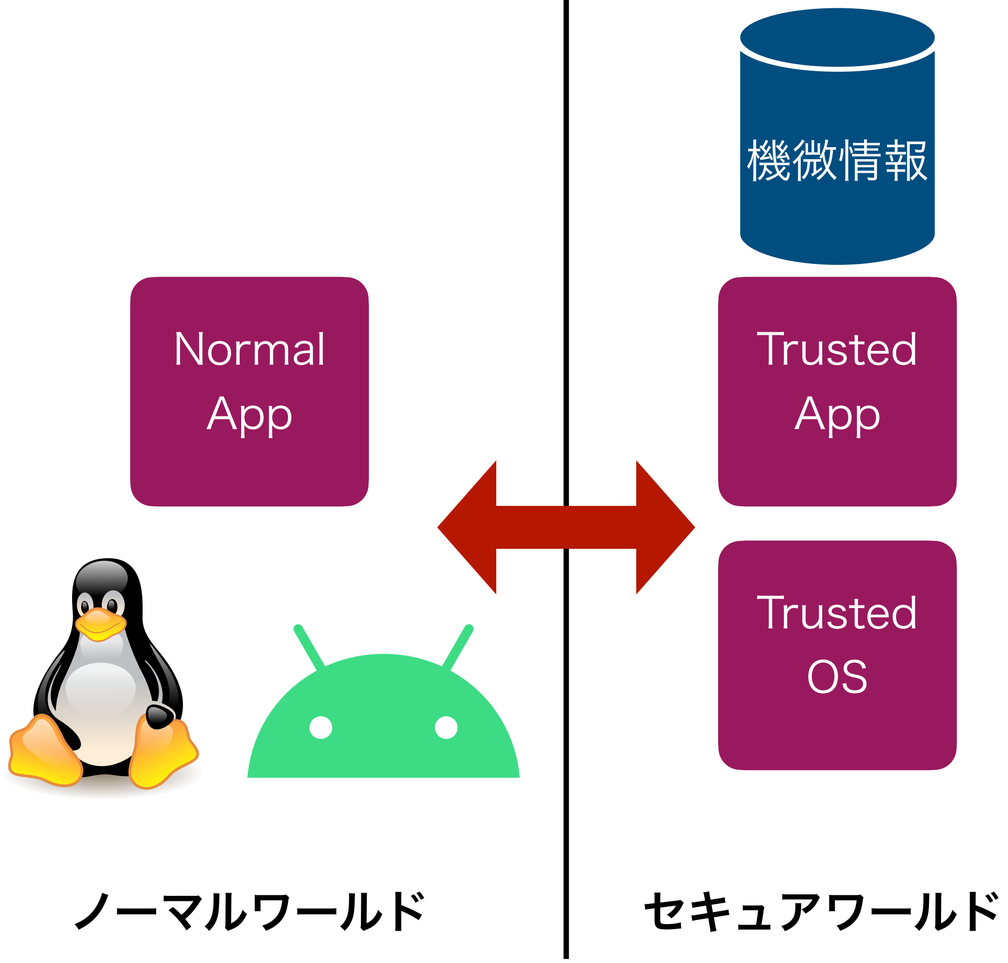
- 研究概要：目的と結果
- 背景と目的：Trusted Execution Environment (TEE)と問題点
- 技術的背景：TrustZoneの説明
- 提案システム：Baremetalispの設計と実装
- 関連研究：TrustZoneと周辺技術
- 評価：Baremetalispと他のTEE技術との比較
- まとめと今後の予定

研究概要

- 背景：既存TEEソフトウェアの問題点
  - そもそものTEE用ソフトウェアのバグ
  - TEE用ソフトウェアの更新が難しいため、外部から任意の計算を注入することが難しく、edge computing基盤の利用にハードル
- 本研究の目的と新規性
  - 目的：型安全性を中核に据えたTEEソフトウェアの実現
  - 提案システム：Baremetalisp
  - 新規性：
    - Baremetalisp TEE：RustによるTrustZone用のファームウェアとOSの設計と実装
    - BLisp：外部からの計算を安全に行えるように、効果系を適用したプログラミング言語

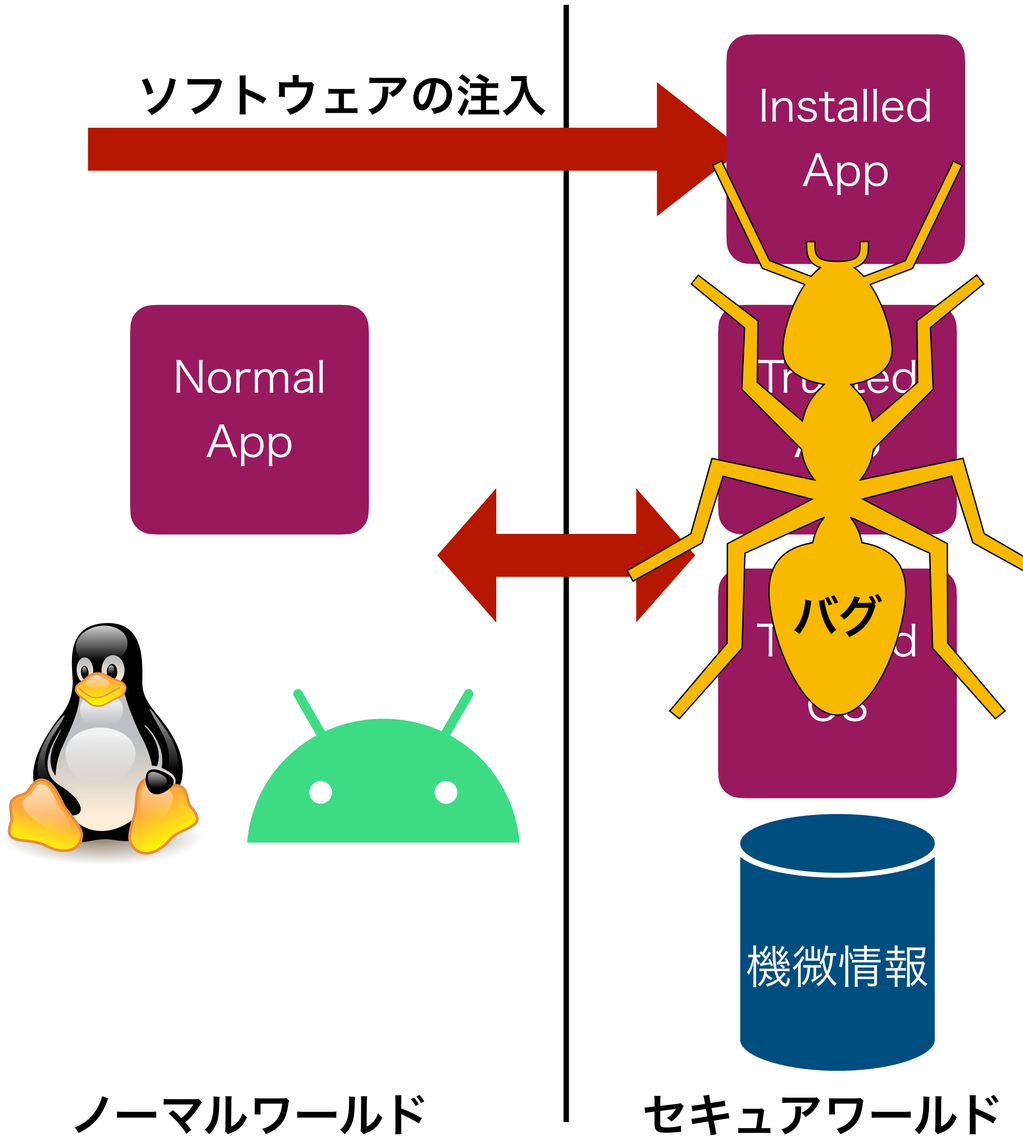
背景（1/2）

- Trusted Execution Environment (TEE)
  - CPUのコンテキストを論理的に分離
  - ノーマルワールド
    - 汎用OSやアプリケーションが動作する世界
    - AndroidやLinux、その上で動くアプリケーション
  - セキュアワールド
    - ノーマルワールドから隔離された世界
    - 専用のOSやアプリケーションが動作
    - 機微情報が置かれる
- TEEのハードウェアアーキテクチャ
  - ArmのTrustZone、RISC-VのKeystone、IntelのSGXなど



背景（2/2）

- TEEソフトウェアのバグ
  - D. Cerdeiraらによると2013〜2018年のTEEソフトウェアにおける、脅威度が中程度以上の脆弱性は80以上 [1]
  - 情報漏洩、ノーマルワールド内ソフトウェアへの不正アクセスの原因に
- TEEソフトウェアの更新の難しさ
  - TEEの外からプログラムを注入して実行することが困難（edge computingの計算基盤利用に障害）
  - 正しいソフトウェアでないと、上記問題が発生（これが難しい）

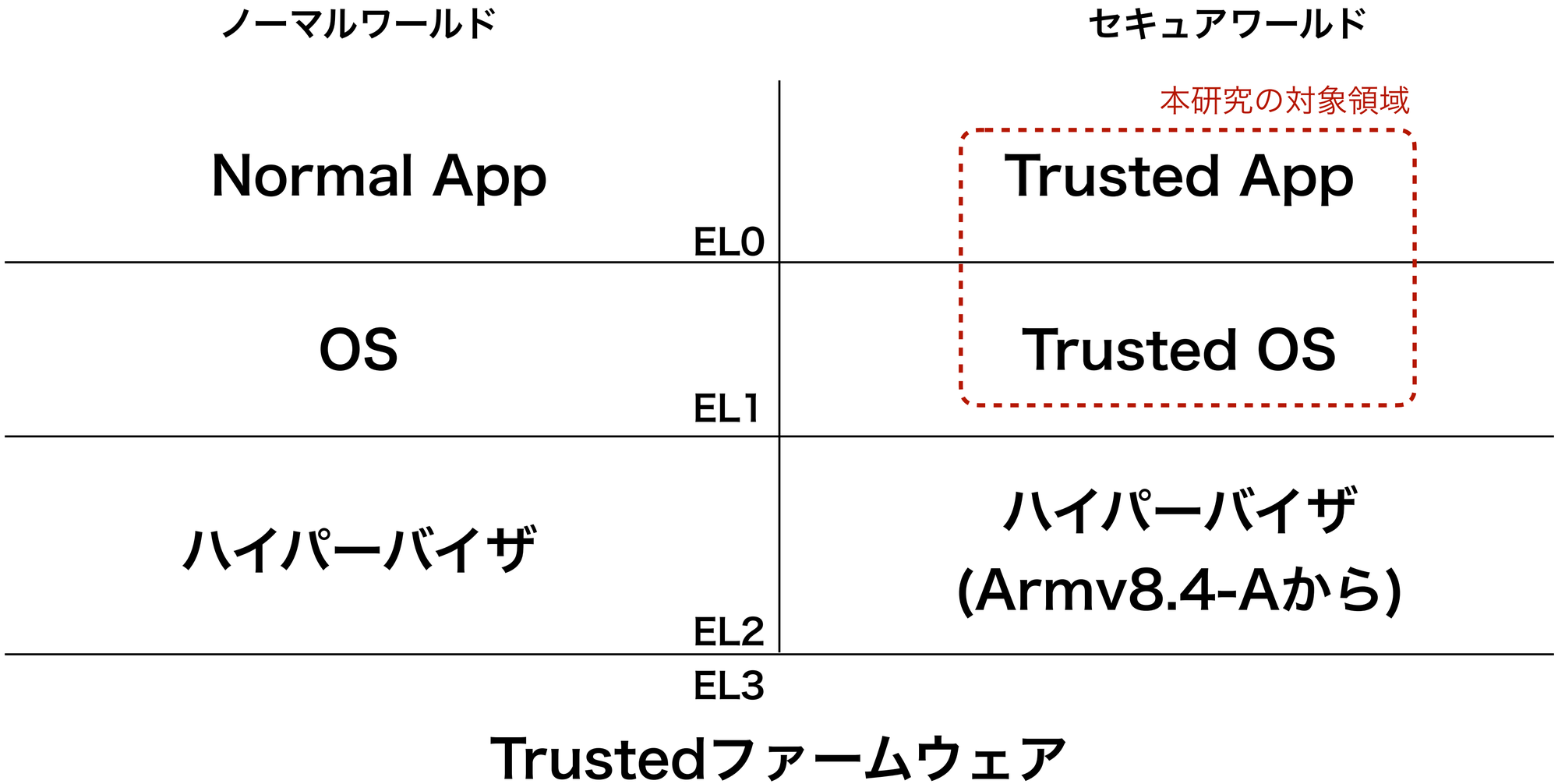


[1] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In 2019 IEEE Symposium on Security and Privacy, S&P 2019, San Francisco, CA, USA, May 19-23, 2019. IEEE, 2020.

本研究の目的

- 型安全なTEEソフトウェアの実現
  - メモリ安全
  - 確実なエラーハンドリング
  - → 致命的な脆弱性の低減
- TEE内への安全なプログラム注入の実現
  - どんなコードを注入しても、情報漏洩は起きないように
  - → edge computingのTEE利用可能に

Armv8-A TrustZone



注：例外レベルは、Exception LevelのELと省略して呼称

提案システム：Baremetalisp

- Baremetalisp：API記述言語のBLispと、TEE実現ソフトウェアのBaremetalisp TEEからなる
- BLisp
  - 型安全なプログラミング言語
  - 効果系を適用し、TEE内への安全な関数の注入を可能に
- Baremetalisp TEE
  - Rust言語で設計・実装された、型安全なファームウェアとOS
  - メモリ安全、確実なエラーハンドリング

BLisp

- 構文はLispに近く、意味論はOCaml、Haskellに近い
- 静的型付け言語
- 一般的な関数型言語の機能を完備：代数的データ型、ジェネリクス、末尾呼び出し最適化、パターンマッチ、型推論
- 効果系を適用
  - IOのある関数と純粋な関数を完全に分離して扱うように
  - セキュアワールドへの安全な関数の注入が可能に

BLispの構文

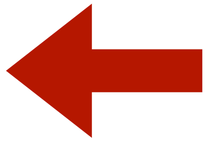
$T	:=		型
		Int	整数型
	|	Bool	真偽値型
	|	'($T)	リスト型
	|	$TFUN	関数型
	|	$TDT	代数的データ型
	|	$ID	型変数
$TFUN	:=	($EF (→ ($T*) $T))	関数型
$EF	:=	Pure | IO	効果
$TDT	:=		代数的データ型
		$TID	型引数なし
	|	($TID $T+)	型引数あり
$ADT	:=	(data $DDEF $MEM*)	代数的データ型定義
$DDEF	:=	$TID | ($TID $ID*)	
$MEM	:=	$TID | ($TID $T*)	
$DEFUN	:=	($FEX $ID ($ID*) $TFUN $E)	関数定義
$FEX	:=	export | defun	

$E	:=		式
		$LITERAL	リテラル
	|	$ID	変数
	|	$TID	代数的データ型
	|	($TID $E*)	代数的データ型
	|	(if $E $E $E)	if 式
	|	(lambda ($ID*) $E)	λ 式
	|	($E+)	関数適用
	|	'($E*)	リスト
	|	$L	let 式
	|	$M	パターンマッチ
$L	:=	(let ($V+) $E)	**let 式**
$V	:=	($LP $E)	
$LP	:=	$ID | ($TID $LP+)	
$M	:=	(match $E $C+)	パターンマッチ
$C	:=	($MP $E)	
$MP	:=	$LITERAL | $ID	
	|	$TID | ($TID $MP*)	

BLispの利用例

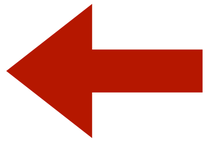
API定義の例

```
(export callback-test (x)
    (IO (-> (Int) Int))
    (call-rust x 0 0))
```



callback-test関数は、IOのある関数として定義

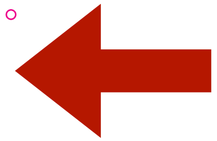
```
(export lambda-test (f)
    (Pure (-> ((Pure (-> (Int) Int))) Int))
    (mul2 (f 2)))
```



lambda-test関数の、引数fには、
Pureな関数のみとるとしてとる

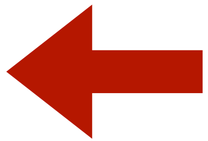
API呼び出し例

```
; 型付けエラー。IO 関数を渡している。
(lambda-test callback-test)
```



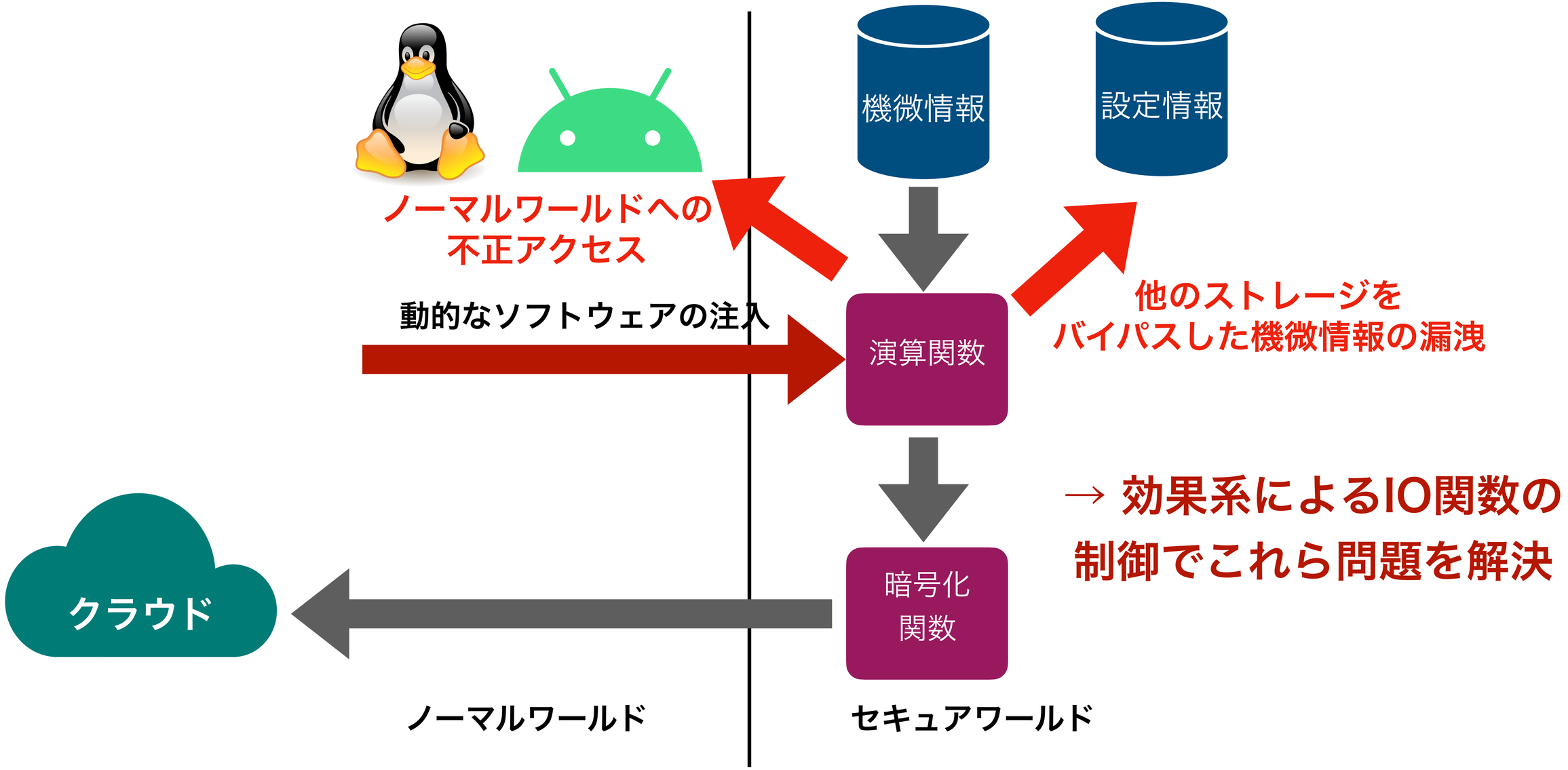
callback-testの効果はIOのためエラー

```
; 型付けエラー。IO 関数をλ式の中で呼んでいる。
(lambda-test (lambda (x)
    (let ((_ (callback-test 1)))
        x)))
```



同じく、callback-testの効果はIOのためエラー

なぜIOの制御が重要か？



Baremetalisp TEE

- Rust言語による設計と実装を行い、型安全なTEEを実現
- ファームウェア：OSの初期化と起動、セカンダリCPUの起動、電源管理（Power State Coordination Interface（PSCI）の一部を実装）、ワールド間のコンテキストスイッチ、メモリマネージメントユニットの初期化など
- OS：BLispランタイムの初期化と起動、動的メモリ管理（Buddy Allocator & Slab Allocator）など

関連研究・TEE関連技術

- TrustZone用TEEソフトウェア：OP-TEE、QSEE、Trustonic TEE、Huawei TEE、NVIDIA TEE、などがある
- RustZone：RustでTrusted Appを実装する提案
- Trusted Language Runtime (TLR)：.NETランタイムをTrustZone内で動作させる提案

評価

	Trusted OS の型安全性	Trusted App の型安全性	動的計算コード 注入の安全性
Baremetalisp （提案システム）	○	○	○
OP-TEE	×	×	×
RustZone	×	○	×
TLR	×	△	×

まとめと今後の予定

- まとめ
  - Armv8-A TrustZone向けの、新たなTEE実現基盤であるBaremetalispを提案
  - Rust言語で設計・実装することにより、Trsuted OSとAppの型安全性を実現し、メモリ安全性と、確実なエラーハンドリングを達成
  - さらに、API定義用言語のBLisp言語の提案を行い、効果系による安全な動的な計算コード注入を実現
  - 基礎部分の実装はほぼ終了
- 今後の予定
  - Linuxなどとの連携は、まだ行えないため、今後対応予定
  - BLispとRustの連携をよりよくしていく
  - unsafeなコードの検証
  - RISC-Vでの動作も確認済みで、他のアーキテクチャへも対応予定