

第31回コンピュータシステム・シンポジウム (ComSys2019)

# SX-Aurora TSUBASA のアーキテクチャと オペレーティングシステム

日本電気株式会社 AIプラットフォーム事業部

今井照之 <t-imai-cp@nec.com>

2019年12月10日

# Orchestrating a brighter world

未来に向かい、人が生きる、豊かに生きるために欠かせないもの。  
それは「安全」「安心」「効率」「公平」という価値が実現された社会です。

NECは、ネットワーク技術とコンピューティング技術をあわせ持つ  
類のないインテグレーターとしてリーダーシップを発揮し、  
卓越した技術とさまざまな知見やアイデアを融合することで、  
世界の国々や地域の人々と協奏しながら、  
明るく希望に満ちた暮らしと社会を実現し、未来につなげていきます。

# 目次

## 自己紹介

SX-Aurora TSUBASA のアーキテクチャ

SX-Aurora TSUBASA のオペレーティングシステム

ヘテロジニアスコンピューティング

# 自己紹介

名前: 今井 照之 (いまい てるゆき)

所属: 日本電気株式会社 AIプラットフォーム事業部

● NEC 府中事業場 (東京都府中市) 勤務

業務: SX-Aurora TSUBASA のオペレーティングシステム (VEOS) 開発

● Aurora の実現性検討、アーキテクチャ検討、OS 開発まで一通り関わってきました

## 経歴

● 2007年: 某大学 コンピュータ科学専攻 修士卒

・ OS と分散並列処理の研究室で、リモートスワップについて研究

● 2007年～2010年: ストリームデータ処理の研究

● 2010年～2011年: ストレージ用オペレーティングシステムの開発

・ iStorage M シリーズ (SAN ストレージ) の OS 開発

● 2011年～現在: HPC システムソフトウェアの開発

・ ベクトルスーパーコンピュータの I/O の開発

– SX-ACE の I/O (ファイルシステム・ネットワーク) の検討・評価など

● SX-Aurora TSUBASA のオペレーティングシステム開発

– SX-ACE後継機の検討から始まり、出荷まで一通り従事

## システムアーキテクチャの構築・実現性検証

- OS-less のデバイスからのシステムコール機構
- OS-less のアクセラレータ風のデバイス上での「丸ごと実行」プログラミングモデル

## 仮想記憶の設計

- UVA 風のプロセスメモリ管理

## ヘテロジニアスコンピューティングを実現する機構の設計・実装

- VH call: VEプログラムから x86 ライブラリ関数の呼び出し
  - ・ ホスト上での任意の処理をシステムコールとして追加
- VE offloading: x86 プログラムからVE関数の呼び出し
  - ・ OpenCL / CUDA 風のプログラミングで VE を使用

# SX-Aurora TSUBASA

## 製品紹介

コンセプト  
モデル一覧  
性能



# NEC HPC Product



SX-9



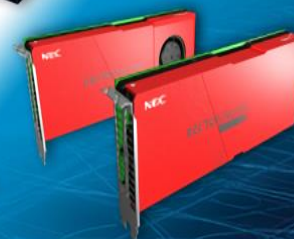
SX-ACE



SX-Aurora TSUBASA



Next Generation



**SX** Vector  
Supercomputer



**LX** x86/Linux  
Cluster

2008

2010

2012

2014

2016

2018

2020

2022



## Design Concept

POINT  
1

### Memory Bandwidth

- ✓ 1.35TB/s per processor

POINT  
2

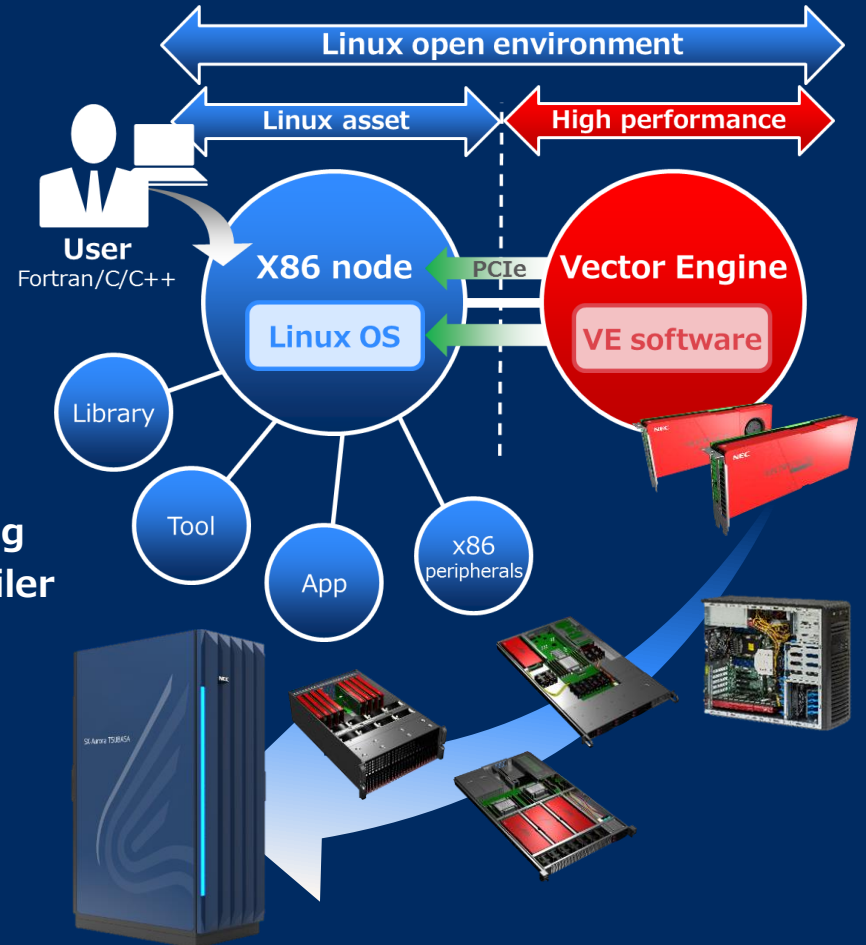
### Easy to Use

- ✓ x86/Linux open environment
- ✓ Fortran/C/C++ standard programming
- ✓ Automatic vectorization by NEC compiler

POINT  
3

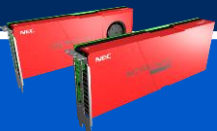
### Flexible and Scalable

- ✓ From Tower to Supercomputer model
- ✓ InfiniBand interconnect for MPI

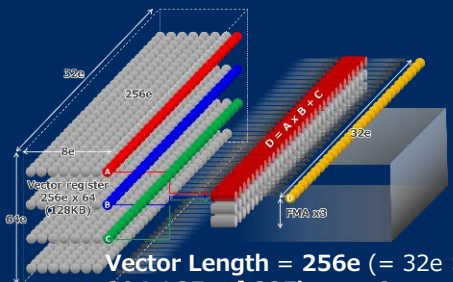


## Vector Engine

### Vector Engine 1.0E Specification



Cores/CPU	8
Core Performance	~304GF(DP) ~608GF(SP)
CPU Performance	~2.43TF(DP) ~4.86TF(SP)
Cache Capacity	16MB shared
Memory Bandwidth	~1.35TB/s
Memory Capacity	24, 48GB



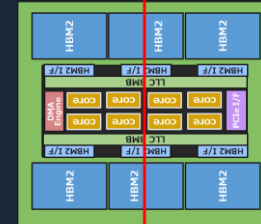
Vector Length = 256e (= 32e × 8 cycle)  
 304.1GF = [ 32Flops × 2 (FMA) ] /cycle × 3 × 1.58GHz

### Card Implementation

- PCIe Gen3 x16 interface
- Full-length full-height card
- Dual slot
- < 300W power

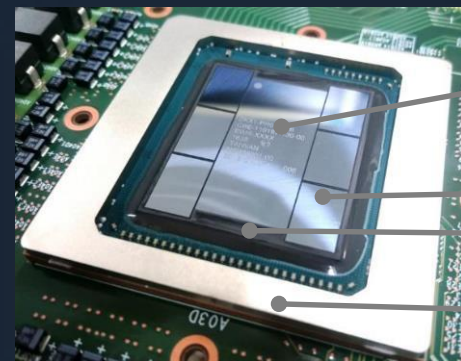
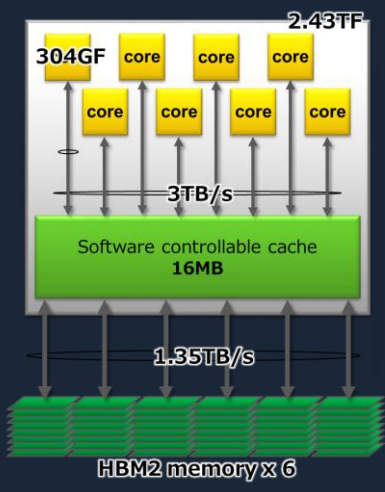


### NUMA(Partitioning) mode support



- Virtually divide into two domains.
- Avoid/relax LLC competition/congestion.

### Vector processor module



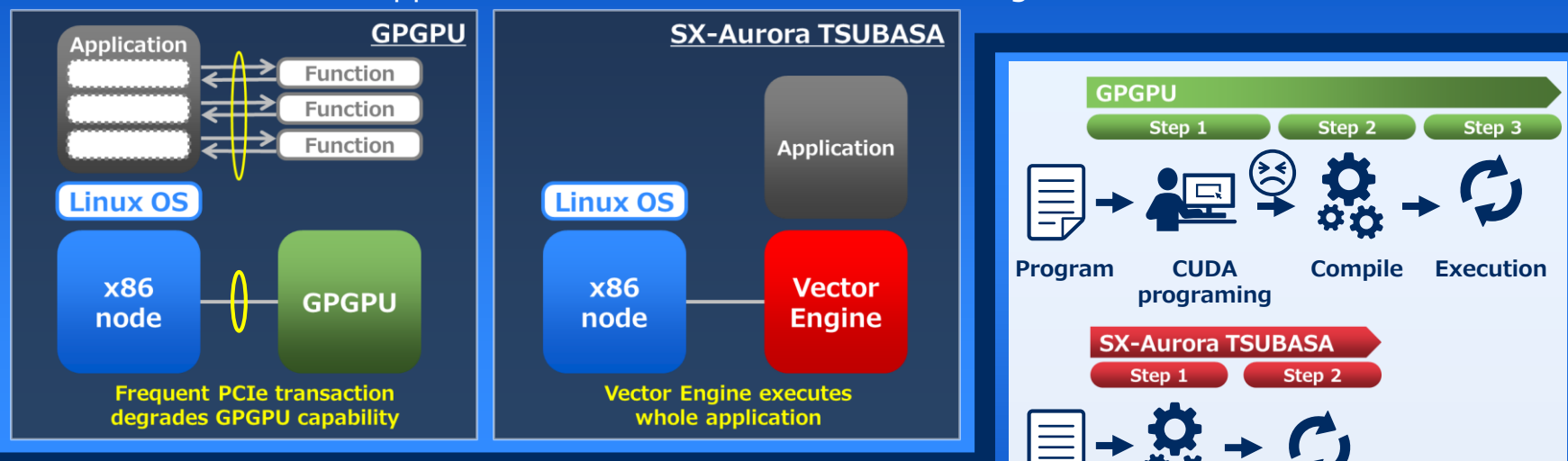
### 2.5D Implementation

- Vector processor  
16nm FF  
15mm x 33mm
- HBM2 x6
- Silicon interposer  
32.5mm x 38mm
- Packgae  
60mm x 60 mm

**World's first implementation of HBM2 x6 /processor**  
 Developed by TSMC/Broadcom/NEC

## Avoiding PCIe Bottleneck



- GPGPU : Function acceleration model
- SX-Aurora : Whole application is executed on the Vector Engine








## Standard programming environment

- GPGPU : Difficult to Use , CUDA programming
- SX-Aurora : Easy to Use , Fortran/C/C++ standard programming

## Type of Vector Engine

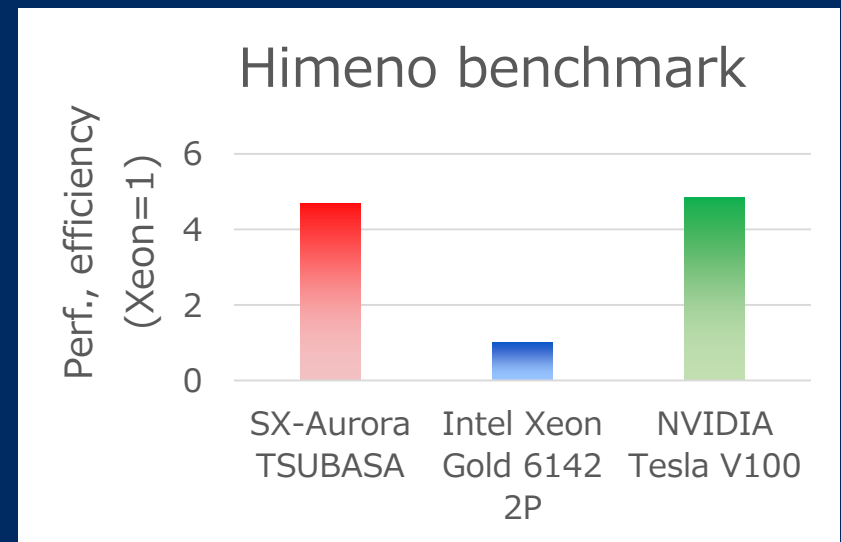
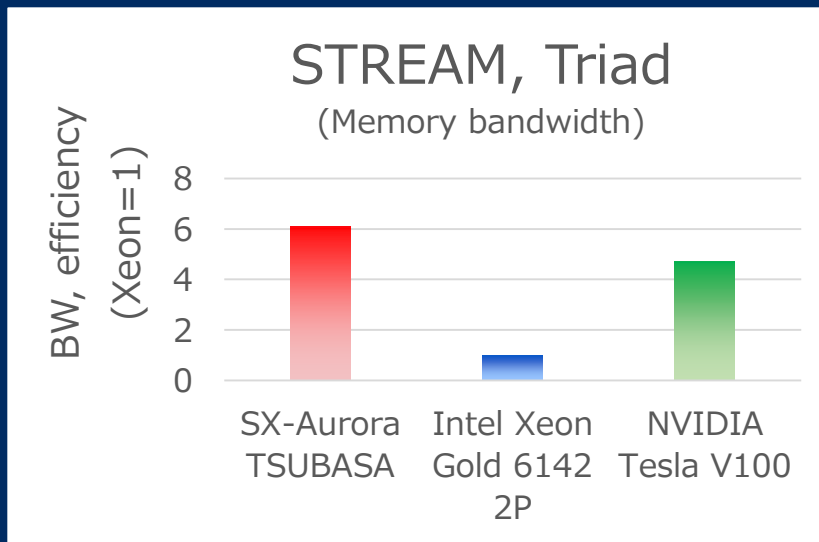
	Type 10AE	Type 10BE	Type 10CE
Vector cores	8	8	8
Frequency	1.584GHz	1.408GHz	1.400GHz
Flops/core	304GF	270GF	268GF
Flops/processor	2.43TF	2.16TF	2.15TF
Memory Bandwidth	1.35TB/s	1.35TB/s	1.00TB/s
Memory Capacity	48GB	48GB	24GB
Interface	PCIe Gen3	PCIe Gen3	PCIe Gen3
Cooling	Liquid	Liquid or Air	Air
Form factor			

## Product Lineup

	Tower model	Rack mount model			Supercomputer
<b>Series</b>	A101-1 A111-1	A311-2	A311-4	A311-8	A511-64
<b>Models</b>					
<b>Form Factor</b>	Tower	1U Rack mount		4U Rack mount	Proprietary Rack
<b># of VE</b>	1	2	2 / 4	8	64
<b>Supported VE</b>	Type 10B Type 10CE	Type 10BE			Type 10AE
<b>VE TFlops</b>	2.15	4.32	8.64	17.28	155.52
<b>VE TB/s</b>	1.22 / 1.0	2.7	5.4	10.8	86.4
<b>Xeon®</b>	1	1	2	2	16
	Gold 6200 series, Silver 4200 series				
<b>System cooling</b>	Air cooling				Liquid cooling
<b>Production</b>	Jan., 2020 launch				

## Basic Performance & Benchmark

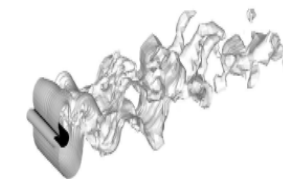
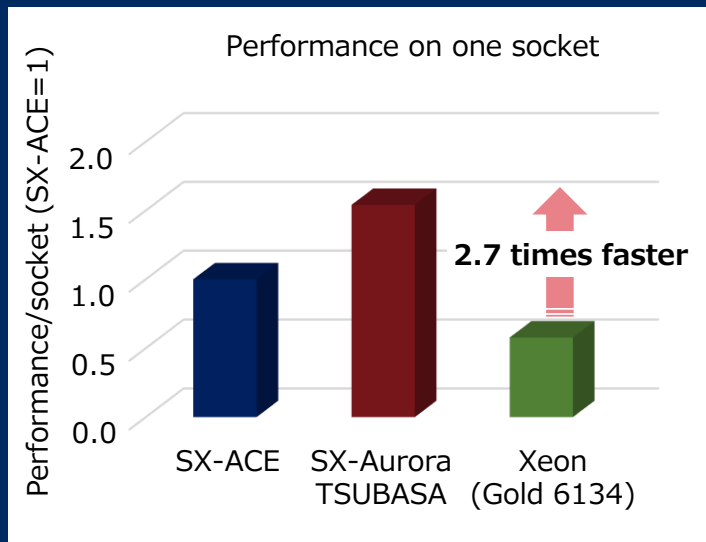
- **STREAM** : Industry leading memory access performance and efficiency
- **Himeno benchmark** : Competitive performance and efficiency



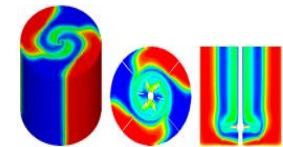
- The **STREAM** benchmark is a benchmark designed to measure the sustainable memory bandwidth.
- The **Himeno** benchmark is a linear solver of pressure Poisson using a point-Jacobi method and known to be highly memory intensive and bound by memory bandwidth.

## Simulation - Incompressible Flow Solver FASTEST

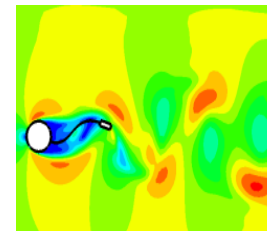
- SX-Aurora TSUBASA delivers **2.7x higher Performance** compared to Xeon/Skylake



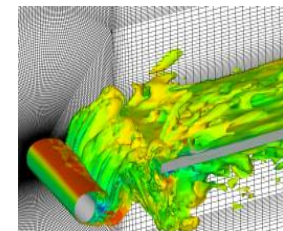
Turbulent flow around a cylinder



Mixing simulation



Fluid-structure interactions



3D flow around cylinder-plate configuration

Source: Technische Universität Darmstadt

- This simulation is ...
  - ✓ FASTEST is a solver of the three-dimensional incompressible Navier-Stokes equations.
  - ✓ The Navier-Stokes equations help with the design of aircraft and cars, the study of blood flow, the design of power stations, the analysis of pollution, and more.



# SX-Aurora TSUBASA の アーキテクチャ

SX-Aurora TSUBASA の生い立ち – SX-ACE 後継機として  
Linux 化の課題

SX-Aurora TSUBASA のアーキテクチャ

# SX-ACE 後継機としての生い立ち

SX-ACE 後継機は、AI・ビッグデータ解析など、大規模計算機センター以外へと適応領域を広げるために、3つのコンセプトに基づき開発開始。

## 従来からのベクトルプロセッサの良いところを踏襲 (1) (2)

- 慣れ親しんだプログラミングモデルの Fortran や C プログラムをコンパイルするだけで自動ベクトル化 / 自動並列化により高速化したい
- × CUDA / GPU のように、プログラムを書き換えて動かす

## 使いやすさ: Linux への移行 (2)

- 従来: System V Unix を NEC が HPC 向けに独自強化した SUPER-UX
- 次世代: de-facto standard な Linux

## デスクサイド・小規模サーバから大規模計算機センターまでをカバー (3)

- シングルノードから大規模クラスタまでを1つのアーキテクチャで実現
- 従来のプライスでは民間 AI・BD 領域には受け入れられない  
→ 原価・開発費の低減が必須

**Design Concept**

- POINT 1 Memory Bandwidth**
  - ✓ 1.35TB/s per processor
- POINT 2 Easy to Use**
  - ✓ x86/Linux open environment
  - ✓ Fortran/C/C++ standard program
  - ✓ Automatic vectorization by NEC
- POINT 3 Flexible and Scalable**
  - ✓ From Tower to Supercomputer
  - ✓ InfiniBand interconnect for MPI

ベクトル機への Linux 移植の障壁は正確な例外を前提とした仮想記憶

- おさらい: 現代的な OS の仮想記憶
- ベクトルプロセッサにおける例外処理の課題
- Linux 環境実現のためのアプローチ

## Demand Paging や Copy-on-Write によるメモリ管理

**Demand Paging:** アクセスするまで実ページの割り当てを遅延

- メモリ領域の割り当て時にはプロセスの仮想アドレス領域のみ割り当て
- ページフォルト発生を契機に、実ページをマップ
- メモリ不足時、ファイルやスワップにデータを退避し、実ページを解放

**Copy-on-Write:** 実際に実ページのコピーが必要になるまで、実メモリの割り当てと内容のコピーを遅延

- 書き込まない限りは、コピーではなく同じ実体で構わない
  - ・ 同一の実ページへのマッピングを設定
  - ・ ただし、書き込みは禁止しておく
- 実際に書き込まれたときに、実ページを割り当て、新しい実ページへマップ
  - ・ ページフォルト発生を契機に、実ページを割り当て、元のページから内容を複製
  - ・ ページフォルトから復帰後、書き込み命令から再開

ページフォルトから、正確に復帰できなければならない

# ベクトルプロセッサにおける例外処理の課題 (1/2)

## 正確な例外とは？

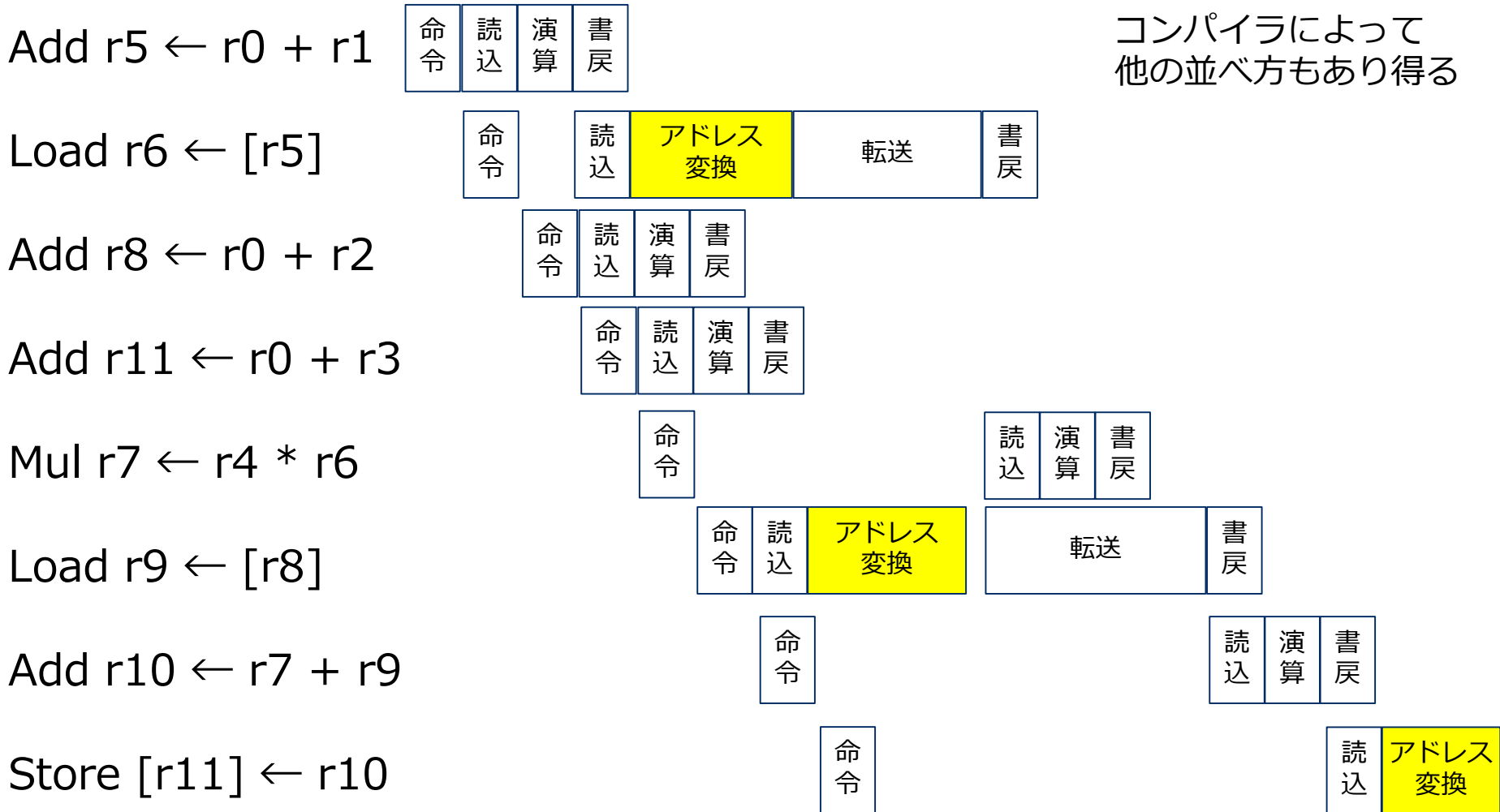
- 例外の原因となった命令が正確に判明
- 例外ハンドラから、例外を発生させた命令の直前の状態に正確に復帰し、再実行可能  
→ あたかもプログラムに記述された順序で実行されたかのように見えること

## Out-of-order (OoO) 実行

- 命令の (プログラム上の) 順序に依らず、オペランドの内容が利用可能になった順に処理を実行 → 依存関係のない命令間では入れ替え可能
  - 空いている演算器で処理できる命令をどんどん実行
  - メモリアクセスは遅い、特にキャッシュに乗らないと遅い
    - レジスタオペランド間: < 数 nsec
    - メモリアクセス: 数十~数百 nsec
  - 例: 次のページ参照
- 現代のプロセッサの多くは OoO 実行
  - もちろん SX のプロセッサも OoO
    - HPC 領域では当然、性能競争があるので
  - 一部の組込プロセッサなどではインオーダー
    - ピーク性能の要求が高くない、電力効率が重要

例:  $c[i] = k * a[i] + b[i]$

パイプライン・アウトオブオーダー実行の RISC における一例



## Out-of-order 実行の性能優先のため、システムコール例外のみ正確

### 正確な例外サポートのために必要なリソース

- 例外から復帰する正確に復帰する = CPUのアーキテクチャ状態を例外発生直前に戻せる  
→ 状態の履歴を保存しておくことが必要
- 前頁の例では？

### ベクトルプロセッサの場合、必要なリソースが特に大きくなる

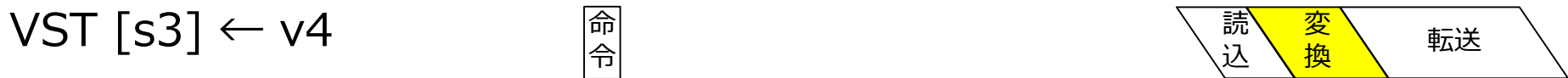
- アーキテクチャ状態のサイズが大きい: 1命令あたり状態変化のサイズ
  - ・ スカラレジスタ (Integer / Floating Point): 8 byte 変化する
  - ・ SX のベクトルレジスタ: 8 byte/要素 × 256 要素 = 2KB 変化
  - ・ (参考) 他社の SIMD は数十 byte 程度
    - 例: AVX-512 は 512 bit = 64 byte
- より多くの履歴を取っておかなければならない
  - ・ ベクトル命令とスカラ命令のレイテンシ・スループットの差
    - 数百命令分のスカラ命令がベクトル命令より先行することがある
  - ・ 対象となるアドレスが多い → 例外が発生しないことが確定するまで時間がかかる
    - 特に間接アクセス (scatter / gather)

Note: システムコール命令は例外発生が確定 → 以降は実行しない (正確な例外実装可能)



# ベクトルプロセッサにおける実行例

```
for (i = 0; i < SIZE; ++i) {  
    c[i] = k * a[i] + b[i]  
}
```



# ベクトル機への Linux 移植のためのアプローチ

✗ CPU が正確なページフォルト例外をサポート  
→ プロセッサの性能が劣化

- ページフォルト例外から復帰のため、OoO 実行の長さに制約

✗ Linux カーネル改造: カーネルが不正確なページフォルトに対応  
→ カーネルの中でも中核となる仮想記憶の再実装

- 仮想記憶の再実装

- 領域をマップする mmap で、ただちに実ページを割り当てる
- プロセスの fork の際、メモリの内容も複製
- スタックの管理

- ファイルシステムの改造

- ページキャッシュが課題
  - ファイルをマップした領域にただちに内容が読まれること
  - マップ中のメモリは破棄してはならない

Linux カーネルの移植はしない

…… では、どのように “Linux” を実現するか？

そもそも “Linux” に見える環境とは？

→ システムコールと (主に GNU の) ツール

OS機能を外部から提供し、Linux を移植せず Linux と同等機能を実現

PCI Express カードにベクトルプロセッサを搭載 = Vector Engine (VE)

- フレキシブルな構成を実現
- 標準に準拠することで開発量も削減

VE に搭載する CPU = 従来の SX の特徴を継承

- ベクトル長の長い (64 bit × 256 要素) のベクトルプロセッサ
- HBM2 採用による高メモリバンド幅

VE は演算に特化

- VE CPU から OS 動作に必要な機能の削減
  - ・ 割り込みハンドラ・特権モードを持たない
- ホストから OS 機能を提供するための機能をVEに実装
  - ・ ホスト側の OS 機能が仮想記憶の機能も提供
    - 従来の正確なページフォルトを前提としない設計 + mmap (メモリマップトファイル・共有メモリ) の機能追加



ハードウェアとソフトウェアのコーデザインによる開発

## SX-ACE 以前と同様、仮想アドレスの割り当てと同時に実メモリをマップ

### 仮想アドレスの割り当て (ヒープ・スタックの伸長、ファイルのマップ)

- 対応する実ページをOSが割り当て
- ページテーブルエントリを登録

### プロセスの fork

- ページテーブルを複製
- 複製に使用する実ページを新たに割り当て
- 内容をコピー

### ファイルのマップ **New**

- 既にマップしているプロセスがある → 当該実ページに対してマップ
- まだマップしているプロセスがない → 実ページを割り当て、ファイルの内容をロード

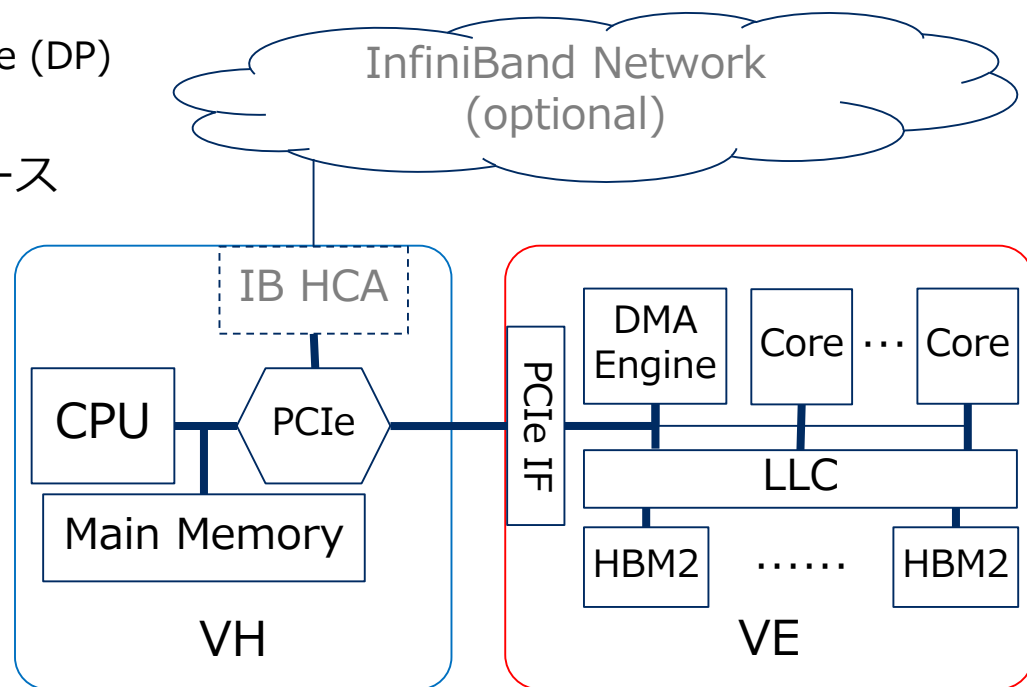
# SX-Aurora TSUBASA のアーキテクチャ

## Vector Engine (VE)

- VE CPU LSI
  - ・ 8 コアのベクトルプロセッサ 307GF/core (DP)
  - ・ 6 HBM で 48 GB、1.2TB/s (\*)
- PCI Express Gen3 x16 インタフェース
  - ・ カード形状・電力 (300W) とともに規格準拠
- 冷却機構
  - ・ 空冷 (active / passive) / 水冷

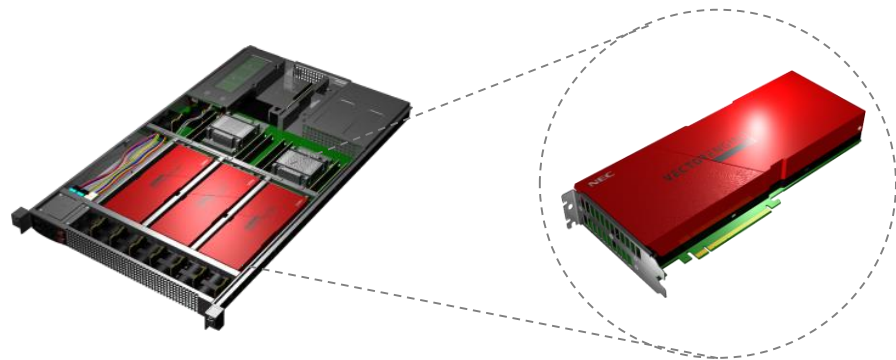
## Vector Host (VH)

- 通常の x86 / Linux サーバ
- 1台の VH に 1 ~ 8 VE を接続
- VE に対して OS 機能を提供



## InfiniBand クラスタ構成可能

- 複数のVHを IB で接続
- VEからのOSバイパス通信をサポート



(\*): 2018年当時の世代の値

# VE CPU の特徴的な機能

## PCI メモリ空間から CPU のすべてのレジスタにアクセスできる

- 実行・停止の制御も可能
- コアの停止中、汎用レジスタ・ベクトルレジスタ・命令カウンタ等の読み書きが可能  
→ プロセスの開始・終了・コンテキストスイッチをホストから制御できる
- 特権レジスタ (ページテーブルなど) は PCI 空間からのみアクセス可能
  - ・ コアの命令では読み書き不可

## 例外をホストに割込で通知する

- 例外が発生すると、コアの実行が停止する
  - ・ 特権モードに入る代わり
  - ・ 例外を発生させる monitor call (MONC) 命令をサポート
    - システムコール用
- 例外の原因は Exception Status (EXS) レジスタに提示
- EXS の内容を例外発生時に PCI メモリ空間上で指定されたアドレスに書き込む
  - ・ OSの効率的な実装のため

## VEから安全に PCI 空間にアクセスする方法がある

- PCI空間にマップできる (DMA用) 仮想アドレス空間とアドレス変換機構
- PIO: Load Host Memory (LHM) / Store Host Memory (SHM) 命令
- DMA ディスクリプタ表

# SX-Aurora TSUBASA の オペレーティングシステム VEOS

VEOS のアーキテクチャ

- (1) プログラムのロード・システムコール
- (2) メモリ管理
- (3) スケジューラ
- (4) ユーザレベル・ゼロコピー通信



# VEOS のアーキテクチャ

## Pseudo Process: ve\_exec

- VE プロセスと 1:1 で動作する Linux プロセス
  - ・実行ファイルのロード
  - ・VE のシステムコールの代理実行

## VEOS service: veos

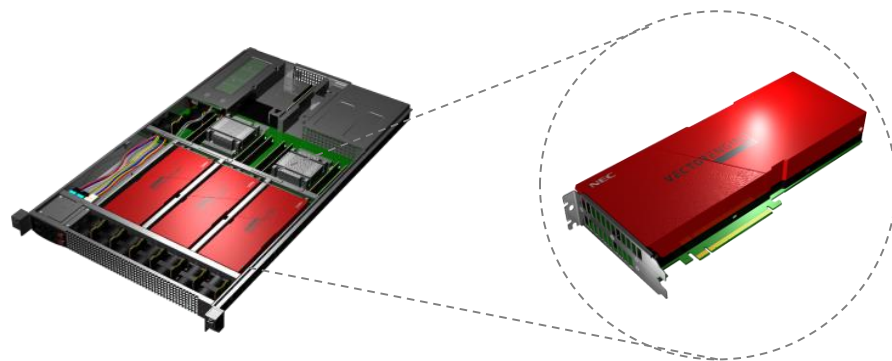
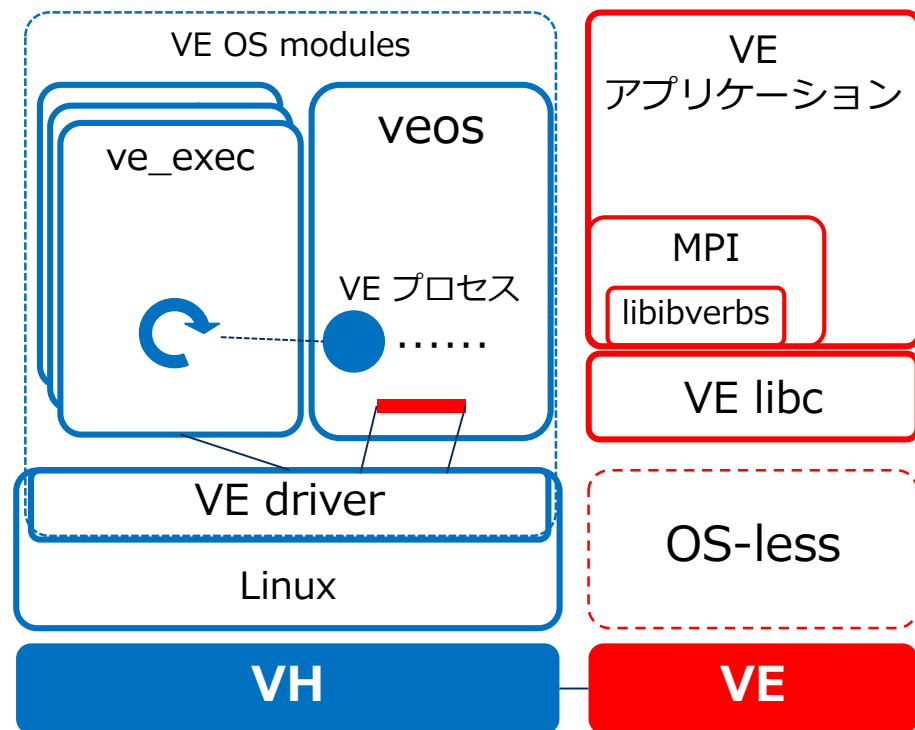
- VE 毎にVE 固有の資源管理を行うRPC サーバ
  - ・VEの物理メモリ
  - ・VEコアのスケジューリング
  - ・システムコールのデータ転送 (DMA)

## VE driver (カーネルモジュール)

- PCI デバイスドライバ
  - ・割込を待つ機能を Pseudo Process に提供
  - ・レジスタ領域のマッピングを VEOS に提供

## VE 側に OS なし

- VE 版 GNU libc を提供



# 1.1 プログラムのロード

## VH上で ve\_exec プロセスを起動

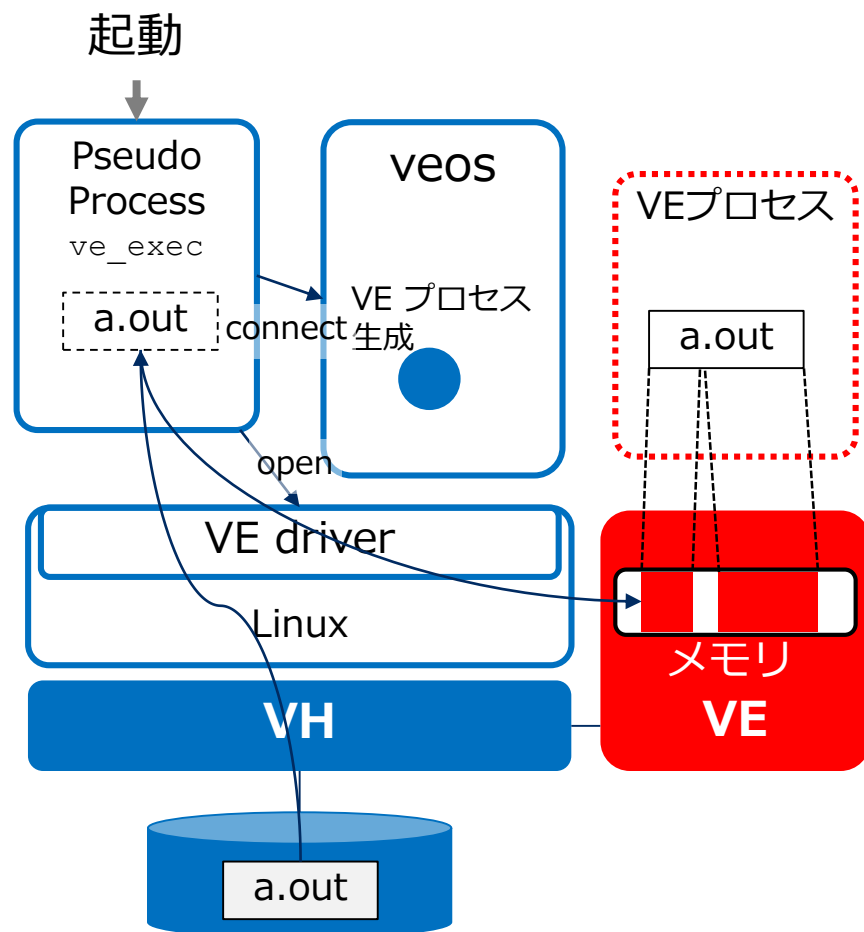
```
$ ve_exec ./a.out
```

## VEOS への VE プロセス生成要求

- VEOS ハンドルの準備
  - ・ VEデバイスを開く
  - ・ VEOS に socket に接続
- システムコール引数用領域をドライバに登録
- プロセス生成の要求 (ve\_exec → veos)
- VEプロセス構造体の生成・初期化 (veos)

## 実行バイナリのロード

- ELFヘッダの読込・解析
- セグメントのロード
  - ・ テキスト・データセグメント (VEメモリ) 割当
  - ・ セグメントのイメージ転送
- 初期スタックイメージの構築
  - ・ コマンド引数 / 環境変数 / auxiliary vector
- プロセスの開始要求 (ve\_exec → veos)  
→ veos のスケジューリングで動作開始



# 1.2 システムコール

## 例外待ち (pseudo process)

- VEドライバの ioctl() 内 wait queue に入る

## システムコールの呼び出し

VE libc が以下の処理

- システムコール番号と引数を SHM で書く
- MONC命令を発行

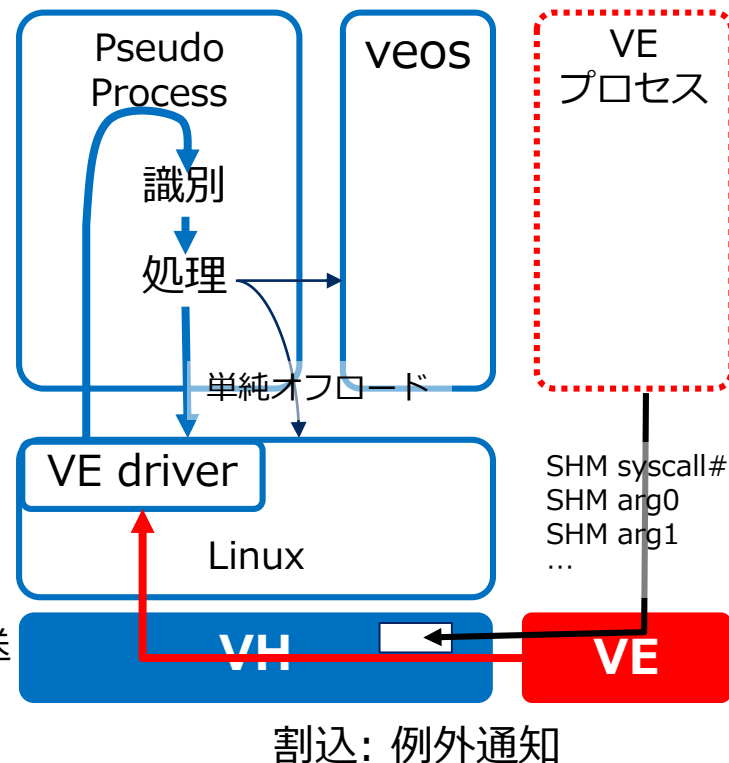
## 例外の通知を受信 (pseudo process)

- 割込ハンドラが待ち状態のプロセスを起こす
- 例外の種別を識別

## システムコールの処理

- 単純オフロード: Linux の同じシステムコールを呼ぶ
  - 例: open, read, write, socket, getpid, getuid, ...
  - バッファの copy-in / copy-out が必要なものはDMAで転送
- VE固有実装
  - Pseudo Process 内の処理 + RPC で veos に処理依頼
  - 例: mmap, brk, fork, clone, ...

ライブラリ libvepseudo.so に実装: ve\_exec は例外待ちループからライブラリ関数を呼出



## 2.1 VEのメモリ管理

仮想アドレス空間は Pseudo Process が、物理メモリは VEOS が管理

### 仮想アドレス空間

Pseudo Process の空間の一部として埋め込み

#### ● 固定領域

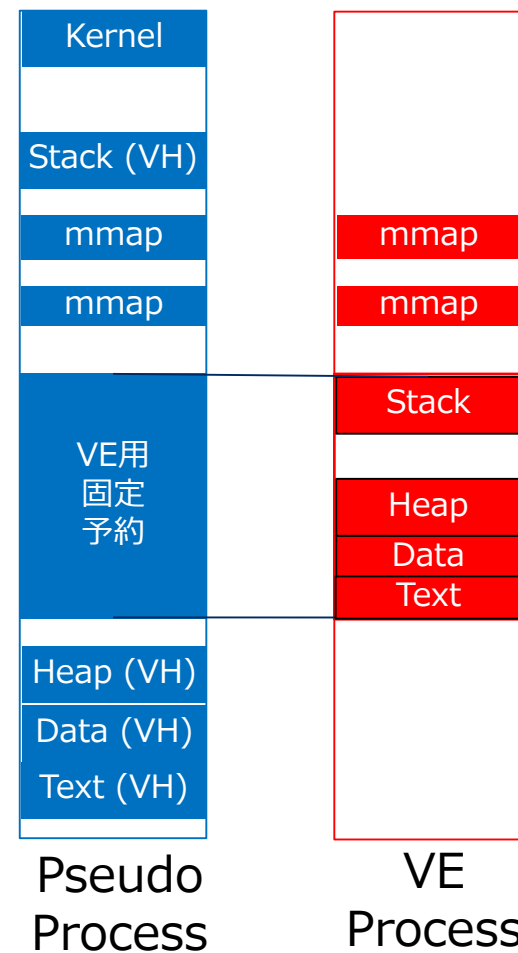
- 予約領域 1TB: 0x6000000000000 - 0x60fffffffff
- VE 用テキスト・ヒープ・スタック

#### ● 動的に決定される領域

- 256 ページ分の “chunk” 単位で Linux の mmap で割り当て
  - VEのアドレス変換機構・ページテーブルの構造上の制限のため
    - » 2MB / 64MB のページサイズ
    - » ページサイズに依らず2段 (32ディレクトリ×256エントリ)
    - » 同一ディレクトリのアドレス領域は同一ページサイズでなければならない
- 割り当てた chunk の使用を Pseudo Process が管理

### 物理メモリ

- VEOS の Buddy System が管理
- VEOS が RPC を受けてマップ / アンマップを処理
  - Pseudo Process が仮想アドレス決定し、RPCで指定



## 2.2 ファイル mmap

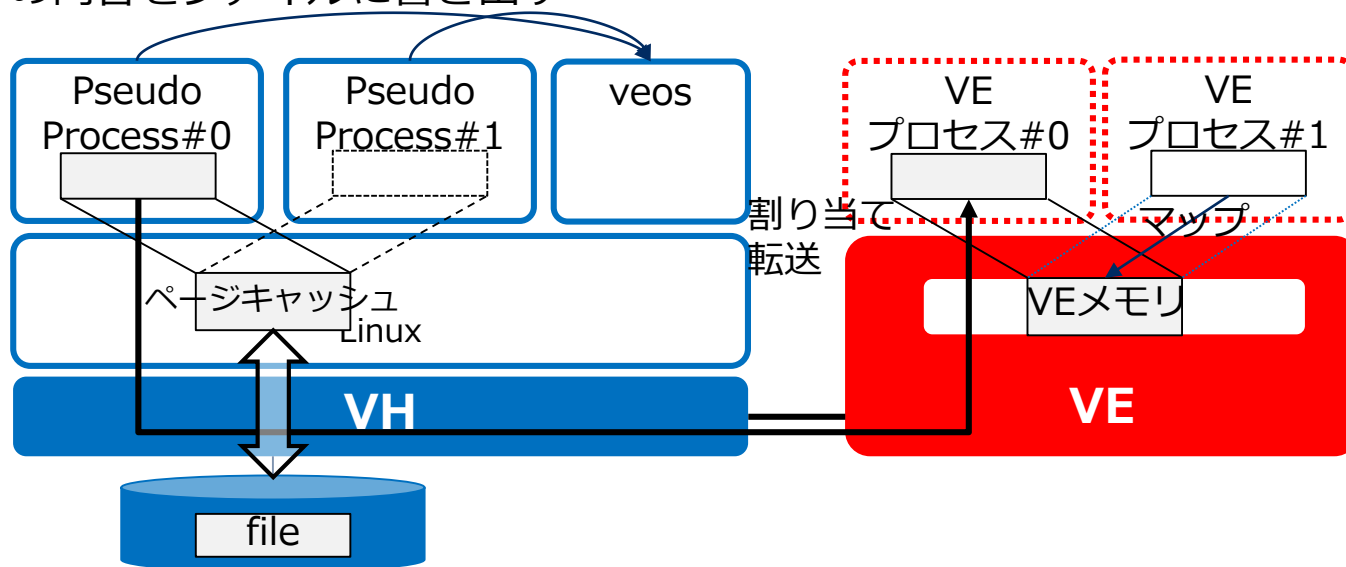
ページキャッシュと独立したVEのページを割り当て、共有

### マップ (mmap)

- VEメモリをただちに割り当て、ファイルから内容を読み込む  
→ 直ちに割り当てるので、デマンドページングに必要な正確な例外不要
- 共有マッピング (MAP\_SHARED) でマップ済みのメモリがあれば、新たなメモリを割り当てず、マップ済みのメモリへマップ

### アンマップ (munmap) ・ 同期 (msync)

- VEメモリの内容をファイルに書き出す



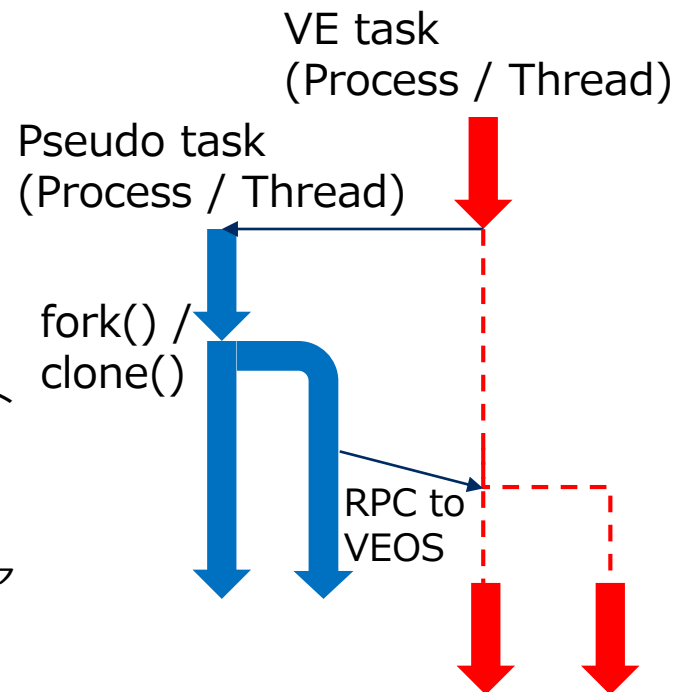
# 3.1 マルチタスクのサポート

## プロセス・スレッド

- VEOS が Linux 同様「タスク」として管理
- fork と clone をサポート
  - Pseudo Process が VH の fork / clone で子を生成
  - 子用の VEOS ハンドルを作成
  - VEOS に子プロセス/スレッド生成を要求

## スケジューリング

- 一般的な OS と同様にコンテキストスイッチをサポート
- 特徴 / 差異
  - 一般的な OS: タイマ割込により特権モードに推移
  - VEOS: VH Linux のタイマ (シグナル) で起動され、VEのコアを停止し外部からコンテキストを転送



## 3.2 シグナル

### VHとの透過性のため、Pseudo Process と VEOS が連携

#### シグナルの配送

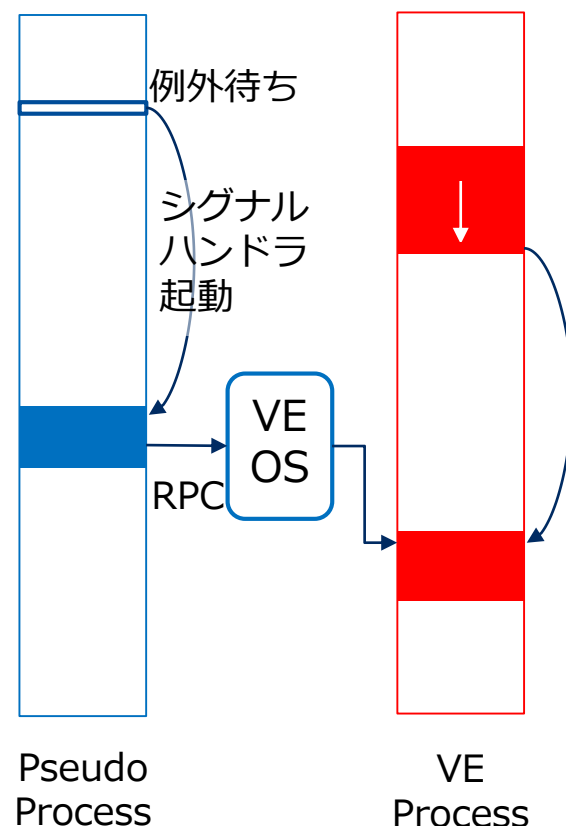
- Pseudo Process のシグナルハンドラがキャッチ
- VEOSにシグナル配送を要求
- 次のVEプロセスを実行するときにVEOS がシグナル配送
  - ・シグナルスタックの構築
  - ・命令ポインタを移動

#### 例外による(同期)シグナル

- 例外種別を識別し、対応するシグナル番号を決定
- シグナル配送をVEOSに要求
- VEOS がVEプロセスにシグナル配送
- 回復不能な例外の場合、プロセス終了
  - ・VEではほとんどの例外は回復不能

#### キャッチできないシグナル

- Pseudo Process 終了を検知するとVEプロセスを終了
- Pseudo Process が停止するとVEプロセスも停止





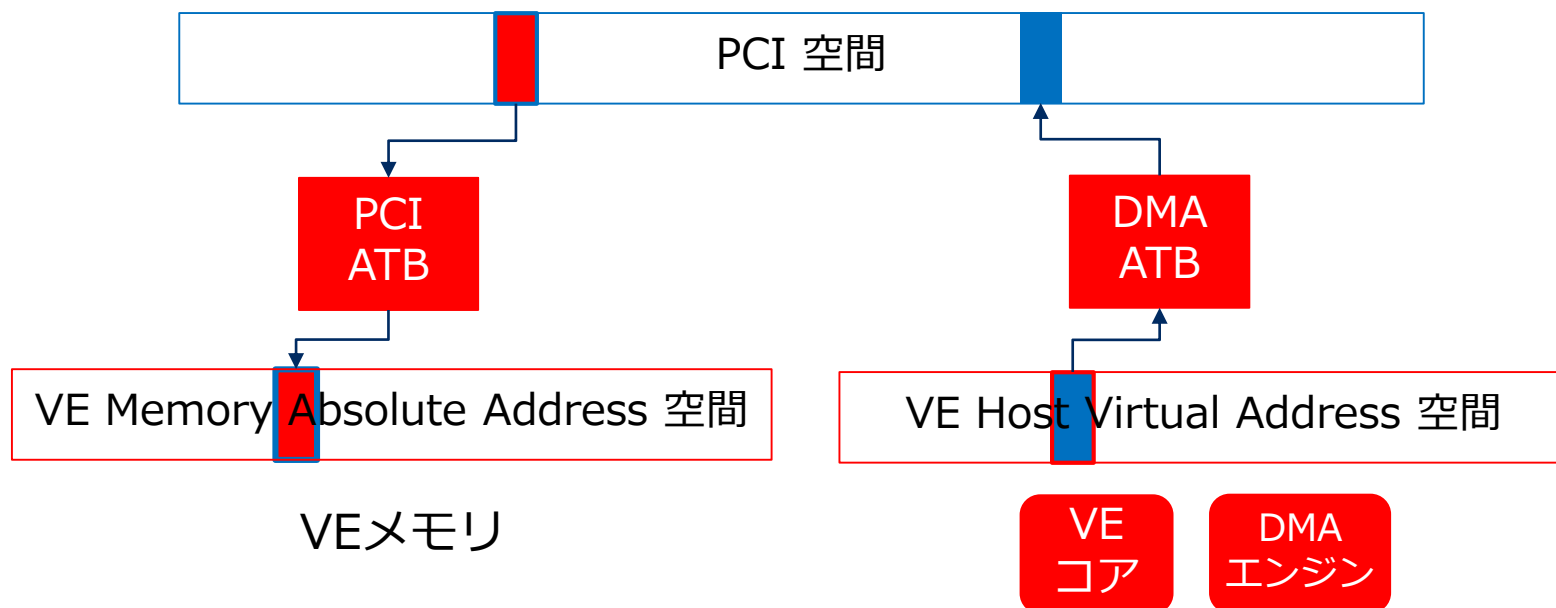
# 4.1 OSバイパス通信のための VE のハードウェア機構

## PCI Address Translation Buffer (PCIATB)

- PCIデバイス (他のVEを含む)、ホストから VE の主記憶へのアクセスに使用するアドレス変換機構
- PCI 空間 → VEMAA (VEメモリの物理アドレス) のマッピング

## DMA Address Translation Buffer (DMAATB)

- DMAエンジン・LHM/SHM 命令が使用するアドレス変換機構
- VE Host Virtual Address (VEHVA, DMA 用仮想アドレス) → PCI 空間のマッピング



## 4.2 VE の OS バイパス通信 (1/2): VE Shared Memory

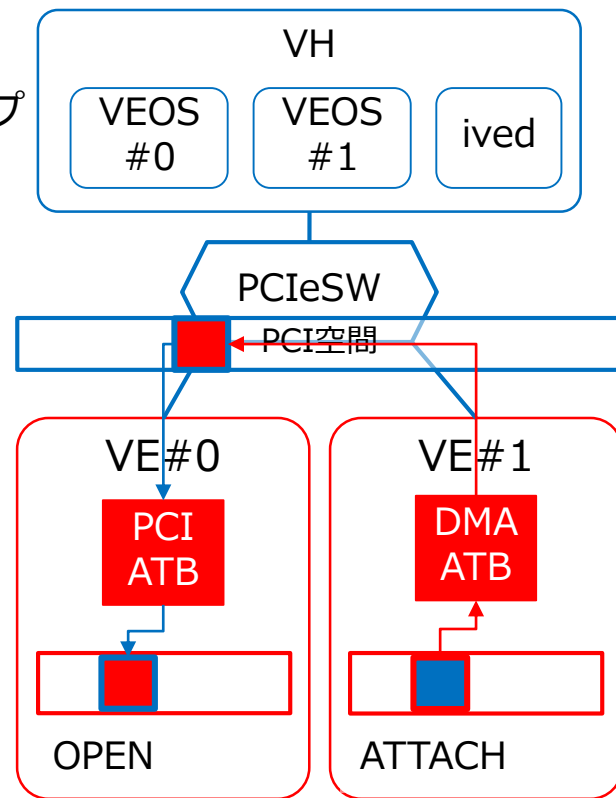
### VEメモリを他の VE から RDMA でアクセス可能にする VEOS の独自機能

#### VE Shared Memory (VESHM) の API

- 自プロセスのメモリを RDMA アクセス可能にする / 不可にする
  - ve\_shared\_mem\_open()
  - ve\_shared\_mem\_close()
- RDMA アクセス可能なリモートメモリをマップ / アンマップ
  - ve\_shared\_mem\_attach()
  - ve\_shared\_mem\_detach()

#### 設計・実装

- Inter VE Daemon (IVED): VH 内の VESHM を集中管理
  - PID と仮想アドレスと対応する VEMAA の組・当該領域の UUID
- Pseudo Process: API に対応するシステムコールを提供
- VEOS: VE 単位の資源管理
  - PCIATB と DMAATB のエントリを管理
  - Open / Close したときに、IVED に当該領域を登録/削除
  - Attach するときは、IVED に問い合わせ



## 4.3 VE の OS バイパス通信 (2/2): InfiniBand 通信

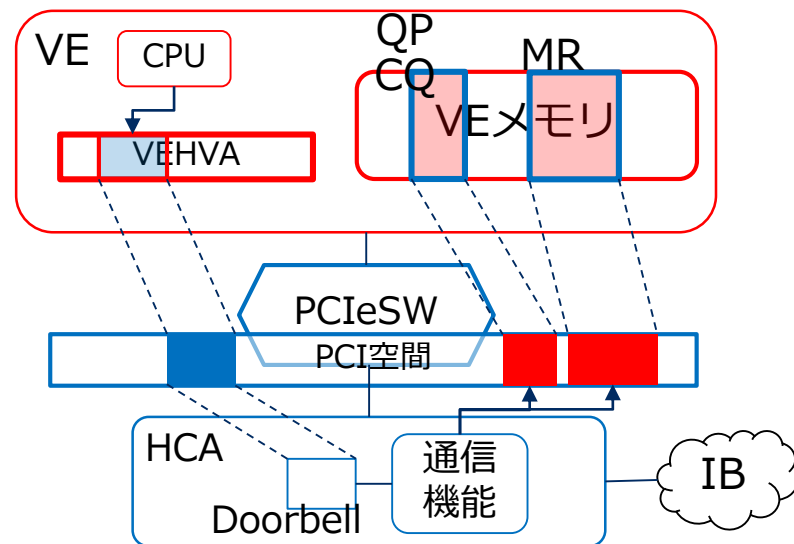
### PeerDirect のサポートにより、VE から直接 InfiniBand 通信

PeerDirect: Mellanox InfiniBand HCA がサポートする PCIe デバイスが直接IB通信するための技術

- RDMA 対象のメモリや Queue Pair、Completion Queueなどを PCIメモリ空間にマップ
- GPU の RDMA 通信 (GPUDirect RDMA) も本機能を使用

### VE 用 PeerDirect 機能

- HCA が直接 VE メモリにアクセス
  - ・ゼロコピー通信を実現
- HCA のレジスタ (doorbell) を VE にマップ
  - ・OSをバイパスして VE から IB 通信を制御
  - ・VHのプロセス仮想アドレスを指定してマップする機能



# SX-Aurora TSUBASA の ヘテロジニアスコンピューティング

VH call

VE offloading

# VEとVHの両方を使うための機能

VEからx86の使用、x86からVEの使用の両方をサポート

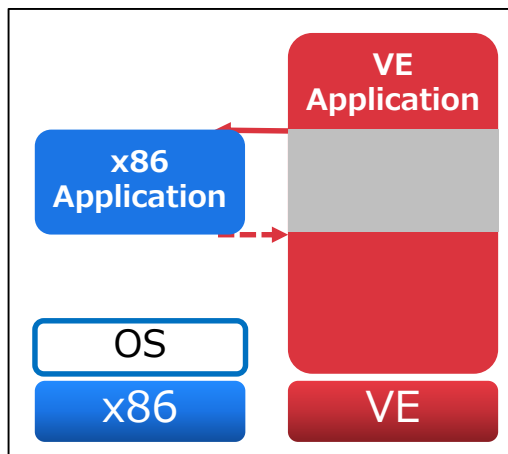
基本的には VE 上でプログラム全体を実行するが、それに加えて  
ヘテロジニアスコンピューティングをサポートする機能を提供

## VH call

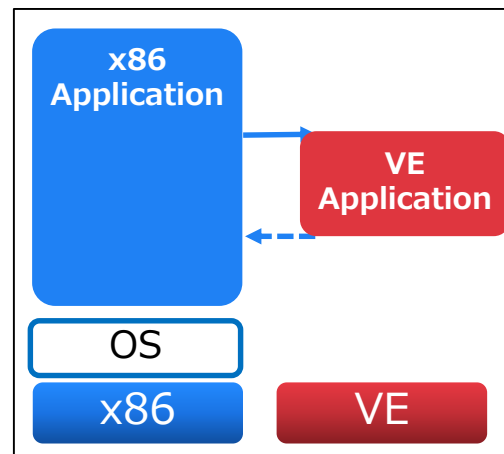
- VEのプログラムから VH 用の共有ライブラリの関数を呼び出す

## VE Offloading (VEO)

- VH のプログラムから VE 用の共有ライブラリの関数を呼び出す
  - ・アクセラレータ風のプログラミングモデルを提供



VH call



VEO

## VE プログラムから VH のライブラリの関数を呼び出す

### API

- `vhcall_install()` / `vhcall_uninstall()`: ライブラリのロード・アンロード
- `vacall_find()`: シンボルの探索
- `vhcall_invoke()`: 関数の呼び出し

### 実装

- VE のシステムコール処理機構上に、(ほぼ単純オフロードのシステムコールとして実装
  - `vhcall_install()`: `dlopen()`
  - `vhcall_uninstall()`: `dlclose()`
  - `vhcall_find()`: `dlsym()`
  - `Vhcall_invoke()`: `vhcall_find()` で取得した値を受け取り、関数ポインタにキャストして呼ぶ
    - 引数としてバッファの先頭ポインタと長さを取り、呼ぶ前後でバッファのデータを `copy-in / copy-out`

# VE Offloading (VEO): 設計

## VEO Process Handle: VEO プログラムを動かすための VE プロセス

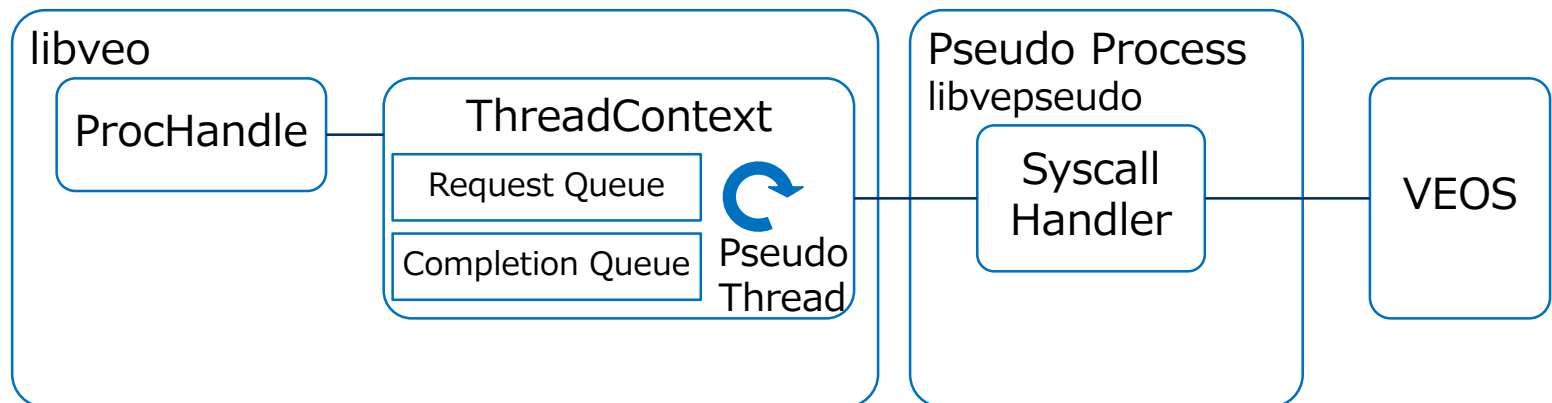
- プロセスの生成
- VEプログラムのロード
- データ転送

## VEO Thread Context: VE上の関数を実行するためのスレッド

- VEO Process Handle から生成
- 非同期な関数呼び出し / 完了確認機能
- 処理のリクエストキュー・完了キューを持つ

## OpenCL 風の API を提供

- 互換性はないが、対応する機能を概ね提供



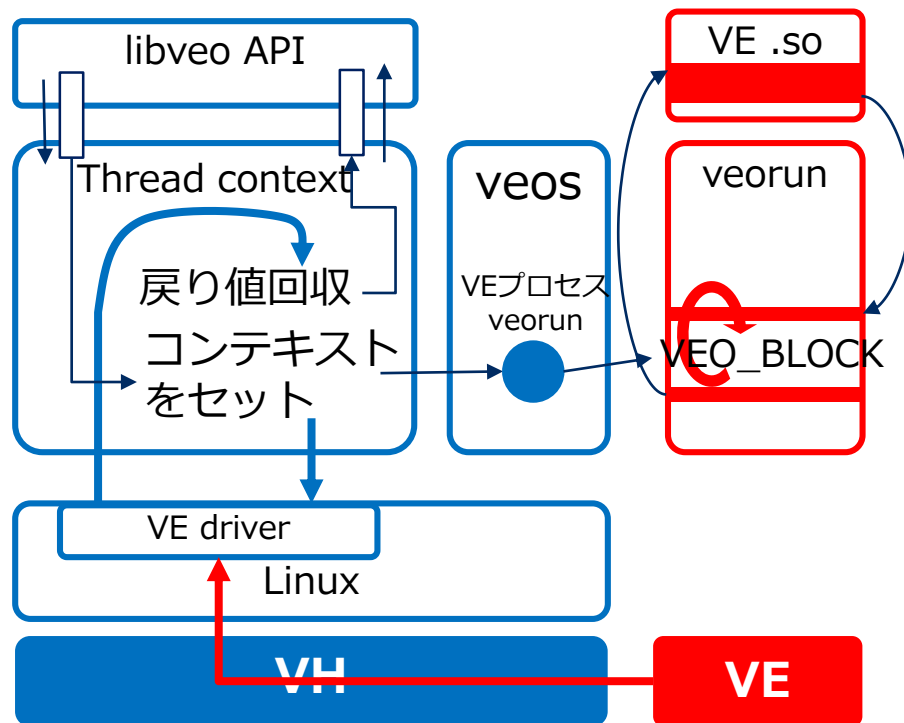
# VE Offloading (VEO): 実装

## 実装

- VH側: libveo.so
  - libvepseudo.so のプロセス生成・プログラムロード・システムコール処理機能を使用
  - 独自システムコール (VEO\_BLOCK) を追加
  - VEO スレッドコンテキスト: VEO用の例外処理ループを実行し続けるスレッド
- VE側: veorun
  - 関数の戻り値レジスタの内容を引数として、VEO\_BLOCKを繰り返す

## 処理

- Process Handle 生成
  - VEプログラム veorun を実行
  - 最初の VEO\_BLOCK まで実行
- Thread Context 生成
  - veorun に pthread\_create() を呼ばせる
  - 子スレッドは例外待ちループ
  - 親は戻る
- 関数呼び出し～完了確認
  - レジスタ・スタックをセットしてVE実行再開
  - VEO\_BLOCK で戻り値を回収





おわりに

## Frovedis (FRamework Of VEctorized and DIstributed data-analytics)

<https://github.com/frovedis/frovedis>

- 機械学習フレームワーク: Spark MLlib / Python scikit-learn とほぼ互換 interface
- 実装: ネイティブの VE プログラム
  - VE 上で動作するMPIプログラムが RPC サーバ
  - Spark (Java) / Python の RPC クライアント

## TensorFlow for SX-Aurora TSUBASA

<https://github.com/sx-aurora-dev/tensorflow>

- TensorFlow の SX-Aurora TSUBASA への移植
- Keras (Neural Network)、vetfkernel (カーネル)、vednn (DNNライブラリ)
- VEO を使用

## HAM – Heterogeneous Active Messages for Offloading

<https://github.com/noma/ham>

- Ham-Offload: C++ の RPC ライブラリ HAM によるオフロードプログラミングモデルの実装
  - X86-64, Xeon Phi (KNC/KNL), **NEC SX-Aurora TSUBASA** and ARM64

## SX-Aurora TSUBASA のアーキテクチャ

- SX-ACE 後継機における Linux 移植の課題
- Vector Host + Vector Engines

## VEOS

- Pseudo Process / VEOS service / VE driver

## VH call と VE Offloading によるヘテロジニアスコンピューティング

## 参考

- NEC Aurora Forum 内 ドキュメンテーション  
<https://www.hpc.nec/documents/>
- GitHub: VEOS for SX-Aurora TSUBASA  
<https://github.com/veos-sxarr-nec>
- GitHub: VE関連 OSS (非公式)  
<https://github.com/SX-Aurora>
  - py-veo (VEO の Python binding): <https://github.com/SX-Aurora/py-veo>



Aurora Forum

検索

 **Orchestrating** a brighter world

**NEC**