SS研　HPCフォーラム

FUJITSU
shaping tomorrow with you

ポスト「京」開発の取り組み

次世代テクニカルコンピューティング開発本部
新庄　直樹

2016.8.26

アウトライン

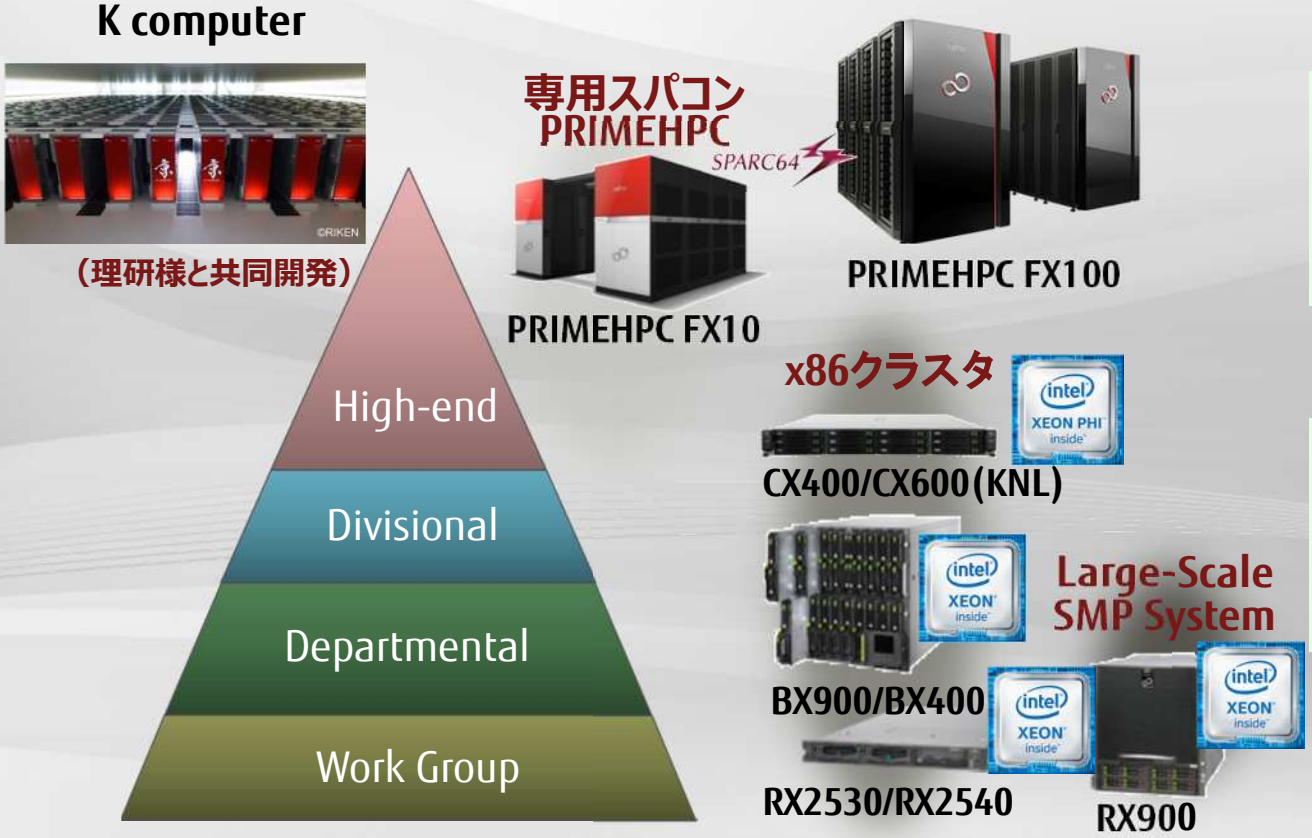
FUJITSU

- 富士通のHPCへの取り組み
- ポスト「京」開発の取り組み
- ARMv8-Aアーキテクチャについて
- HPC拡張命令セットSVEについて
- まとめ

富士通のHPCへの取り組み

FUJITSU

- お客様のニーズに合わせたHPCソリューションを提供
  - 独自CPU搭載の専用スパコンとx86クラスタシステムの両面サポート
  - シングルシステムイメージ運用を実現するシステムソフトの開発・提供
  - 高性能、高可用性、高信頼性の実現



CPUとインターコネクトを独自開発し、高いスケーラビリティを実現

PRIMERGYによるx86クラスタで、最新のCPUとアクセラレータに対応

PRIMEHPC FXシリーズとポスト「京」の開発

FUJITSU

Past	2015	2020

FUJITSU PRIMEHPC

FX10 FX100

- 1.8x CPU perf.
- Easier installation
- 4x(DP) / 8x(SP) CPU & Tofu2
- High-density & lower energy

Technical Computing Suite (TCS)

- Handles millions of parallel jobs
- FEFS: super scalable file system
- MPI: Ultra scalable collective communication libraries
- Lower OS jitter using assistant core
- Context-aware optimizing compiler
- Job-dedicated, scalable local file system

Japan's National Projects

Operation of K computer

HPCI strategic apps program → Priority Issues

Apps. review / FS → Post K computer development
FUJITSU developed HPC optimized ARM CPU w/ lower energy

Post-K

PRIMEHPC FXシリーズ

- 「京」のデザインコンセプトを継承し、Tofuをはじめシステムアーキテクチャを踏襲、強化
- アプリケーション互換を維持
- アプリケーション資産と利用ノウハウを蓄積

ポスト「京」

- 理研とともにポスト「京」システムの開発に取り組み中
- ARMv8+HPC拡張命令をサポートするCPUを開発中

ポスト「京」開発の取り組み

FUJITSU

- 要件
  - 「京」、PRIMEHPC FXシリーズとの連続性の堅持
  - 電力あたり演算性能の向上
  - オープンコミュニティとの連携と普及
- 施策
  - レイテンシコアの高い演算性能と高いメモリバンド幅＋高いスケーラビリティ

    【スケーラビリティ向上】 ハイブリッド並列（VISIMPACT）、アシスタントコア、

    【演算性能向上】 FMA、SIMD拡張、数学関数補助、

    【効率向上】 コア間バリア同期、セクタキャッシュ、ソフトウェア制御プリフェッチ、
  - 最先端テクノロジ使いこなしと制御
    - 半導体、メモリに最先端テクノロジの採用と電力制御
  - システムソフト、OSSの拡がるオープンアーキテクチャへの移行
    - ARM/Linuxのコミュニティとの協業を促進し、コミュニティとともにHPC技術、利用技術を磨き、利用者の利便性を高める

ARM ISAの採用と強化

FUJITSU

- ARMとともにHPC拡張命令（SVE）を開発
  - SVE(Scalable Vector Extension)の策定に“Lead partner”として参画
  - 富士通で実績を積んだHPC技術を盛り込み、ARM ISAのHPC分野での応用の拡大を目指す
- ARM SVEをポスト「京」向けCPUに実装
  - オープンアーキテクチャSVEの採用で、HPCの技術開発の機会を拡大し加速
    - コンパイラ、システムソフト、アプリケーション開発で富士通からARMコミュニティに技術貢献し、コミュニティとの協業を促進
    - 他社システム含めた、ARMサーバのインストールベースの拡大で、コミュニティが拡大、強化
  - PRIEMHPCシリーズで実績ある、独自拡張についても継続、強化する
    - コア間ハードウェアバリア、セクタキャッシュなど
- マイクロアーキテクチャの継続強化と展開
  - ARM ISAへの対応も糧にマイクロアーキテクチャの強化を継続
  - 継続するSPARC、メインフレームCPU開発にもシナジー

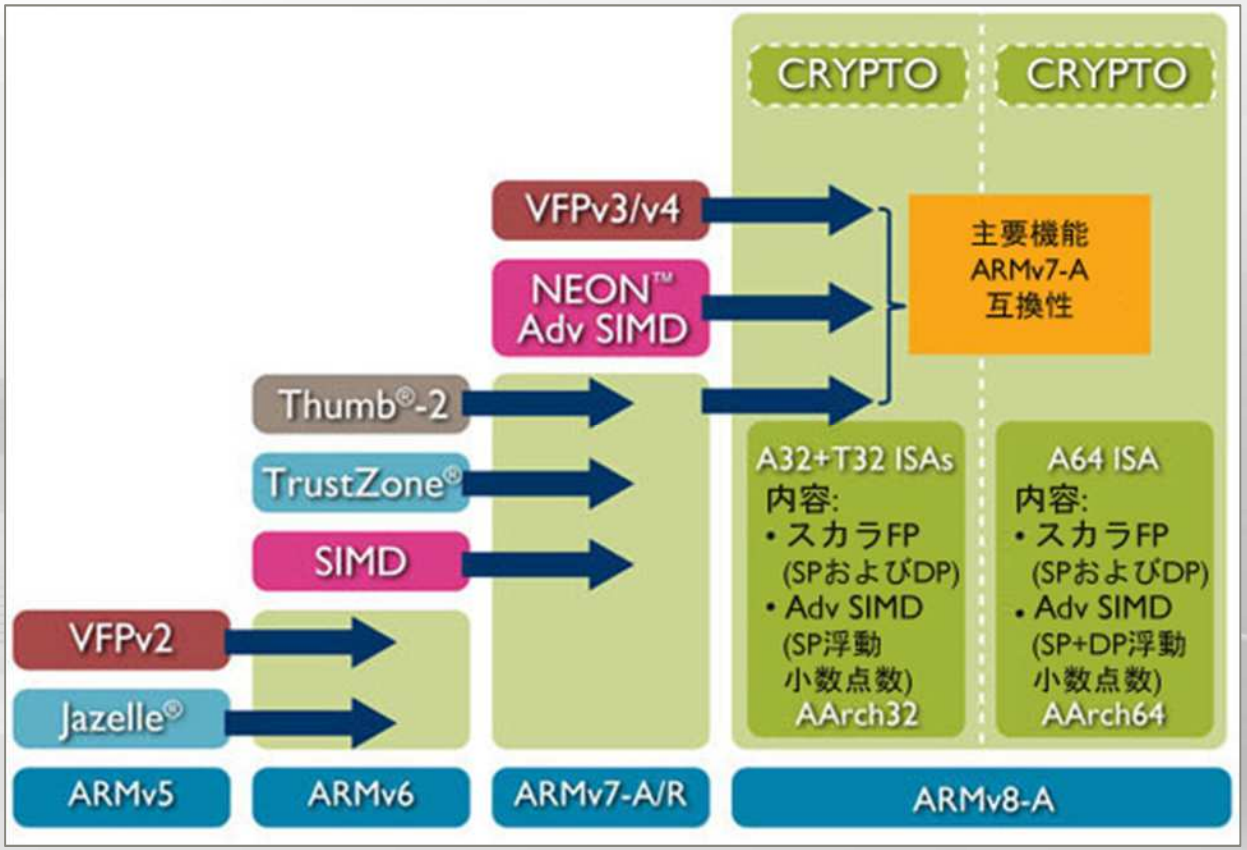
FUJITSU

ARMv8-Aアーキテクチャについて

ARMアーキテクチャの歴史

FUJITSU

- 25年に渡り電力効率の良いCPUを開発
- ～ARMv7：32bit命令セット。組込み分野でデファクト化
- ARMv8-A：従来の32bit命令セットに加え、新規に64bit命令セットを追加



引用元: https://www.arm.com/ja/products/processors/instruction-set-architectures/index.php

ARMv8-Aアーキテクチャ

FUJITSU

アーキテクチャの狙い

- 市場に浸透した32bit命令セットであるARMv7に対する互換性を保つ
- 64bitモードに対応して、サーバ市場にターゲットを拡大

2つの実行状態

- AArch32: 32bitの実行状態　(ARMv7互換)
  - 命令セット：A32, T32
- AArch64: 64bitの実行状態
  - 命令セット：A64

ARMv8-A CPU搭載製品例

- arrows NX F-02H(Snapdragon 808 MSM8992)
- MP30-AR0(X-Gene 1)

AArch64の命令セット

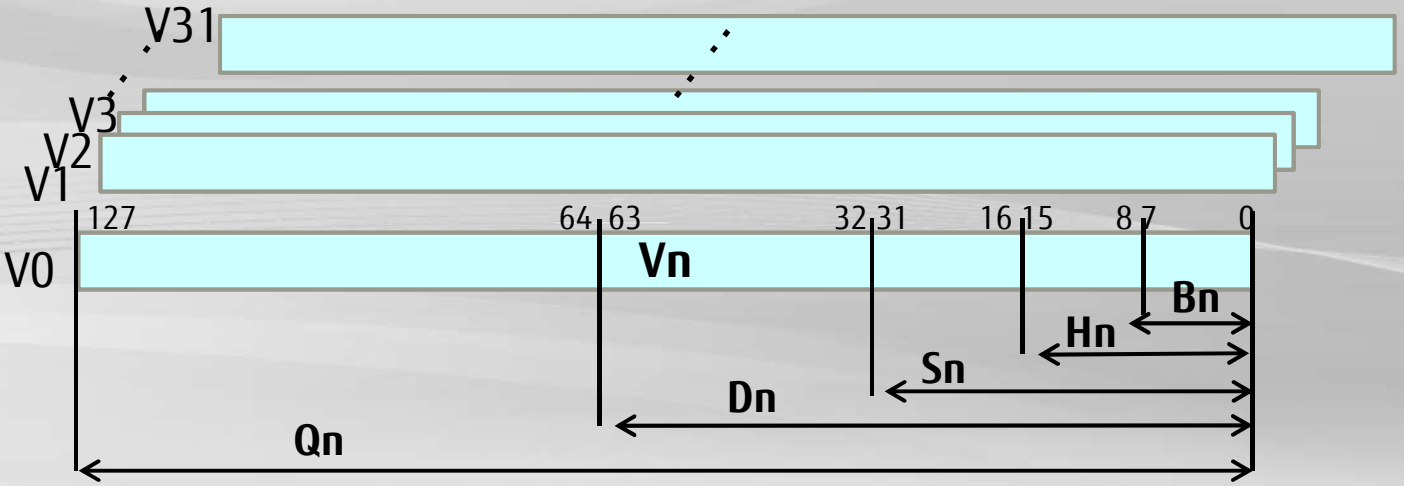
FUJITSU

- 概要
  - 64bitアドレス空間（物理メモリは48bitアドレス）
  - 32bit固定長命令
  - 汎用レジスタ 31個
  - 浮動小数点レジスタ32個（NEON用 128bit）
- 特徴
  - 使用頻度の高い命令列をより高速に実行
    - 多種類の命令サポート
    - 複合命令も多種類サポート
      - 例）シフト + 加算: add x1, x2, x3, LSL #2
    - 豊富なアドレッシングモード
  - NEON(Advanced SIMD)が128-bit SIMDをサポート
    - ARMv8-AのAArch64で、NEONが強化されIEEE754準拠化

NEON (Advanced SIMD)

FUJITSU

- 概要
  - 整数および単精度、倍精度浮動小数点数のSIMD命令を定義
  - 128ビットレジスタ 32本 V0～V31 （下図参照）
- 特徴
  - マルチメディア演算やデータ処理用途のSIMD拡張命令
  - 要素サイズ単位で、ベクトルレジスタを隙間なく利用できる
    例：8bit要素16個でVnを使用(Vn.16B)



FUJITSU

HPC拡張命令セットSVEについて

SVE(Scalable Vector Extension)

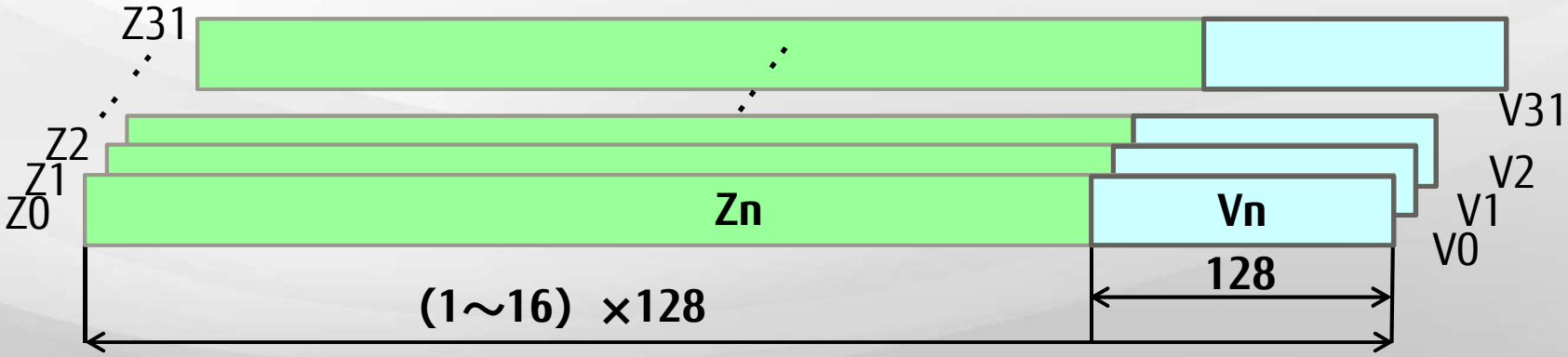
FUJITSU

- AArch64のHPC拡張命令セット
  - 富士通のHPCの経験・技術を活かし、ARMと共同設計
  - NEONとは別の拡張として、HPCにフォーカスしたSIMD命令を定義
  - SVE命令とNEON命令を同時に利用可能
  - 高度なSIMD化を実現する豊富な命令をサポート
- ポスト「京」では、SVEを採用する

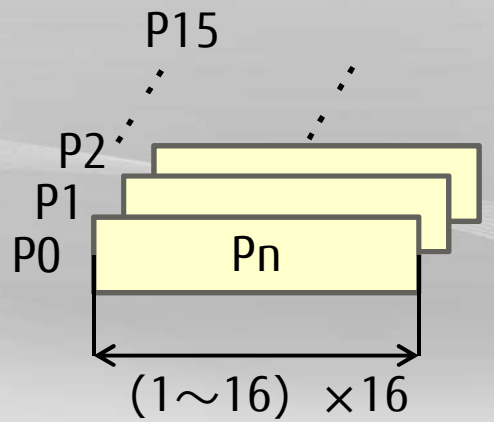
SVEのレジスタ構成

FUJITSU

- Scalable vector registers Z0-Z31
  - NEONのベクトルレジスタV0-V31の拡張
  - 単精度・倍精度浮動小数点と64,32,16,8ビット整数



- Scalable predicate registers
  - P0～P7：Load/Store, 算術演算制御
  - P8～P15：Predicate操作
  - FFR：投機的LoadによるFaultを記録



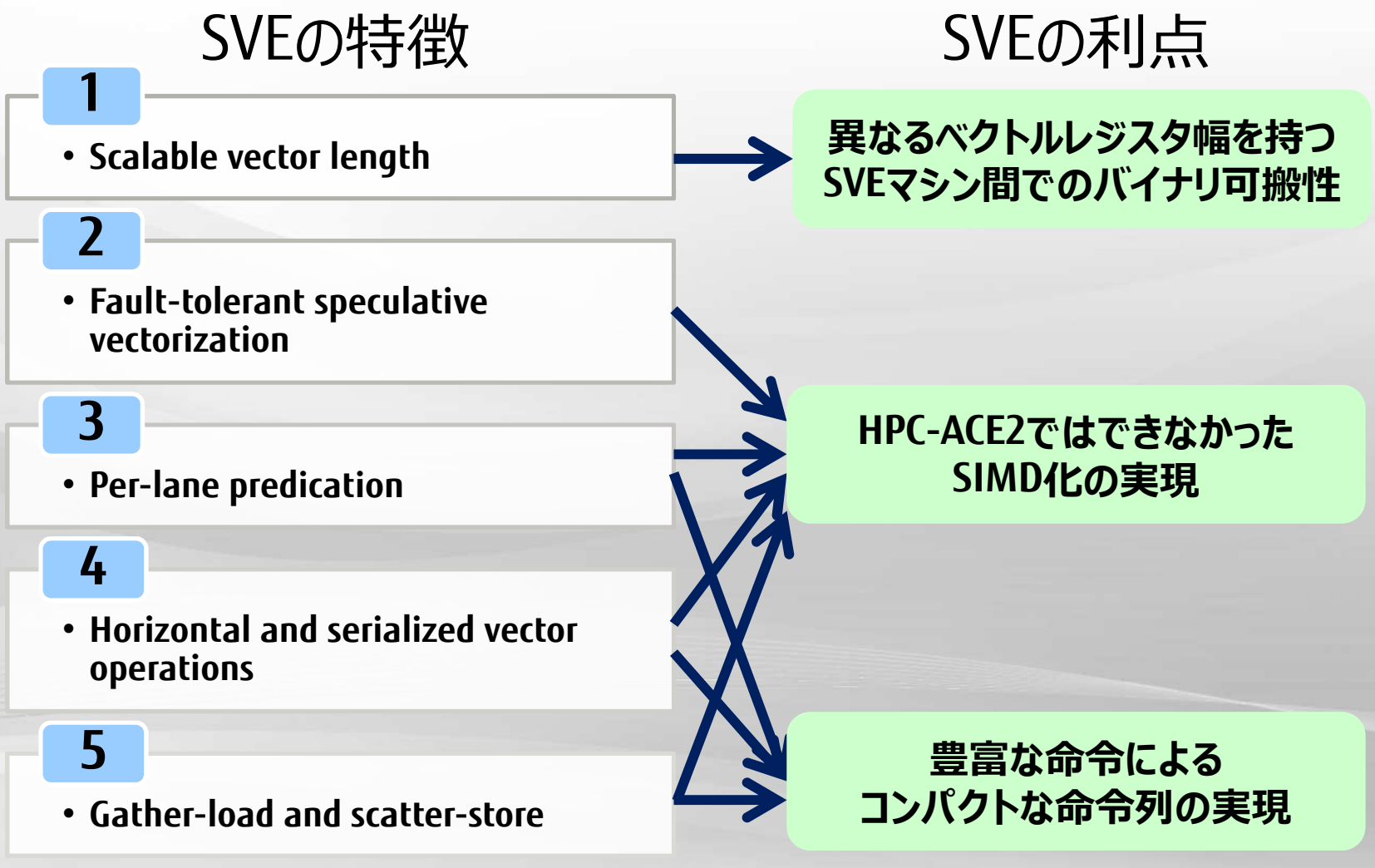
SIMD ISA拡張の比較

FUJITSU

		SVE	HPC-ACE2	AVX-512
ベースISA		ARMv8-A	SPARC V9	Intel 64
SIMD	bit幅	**128～2048**	256	512
	単精度要素数	**4～64**	8（命令に制限有）	16
	倍精度要素数	**2～32**	4	8
汎用整数レジスタ（本）		31+SP （v8-Aと同じ）	32+32 （g,l,i,o+xg）	16
ベクトルレジスタ（本）		32	128	32
プレディケートレジスタ（本）		**16**	（ベクトルレジスタを共用）	8

SVEの優位性

FUJITSU

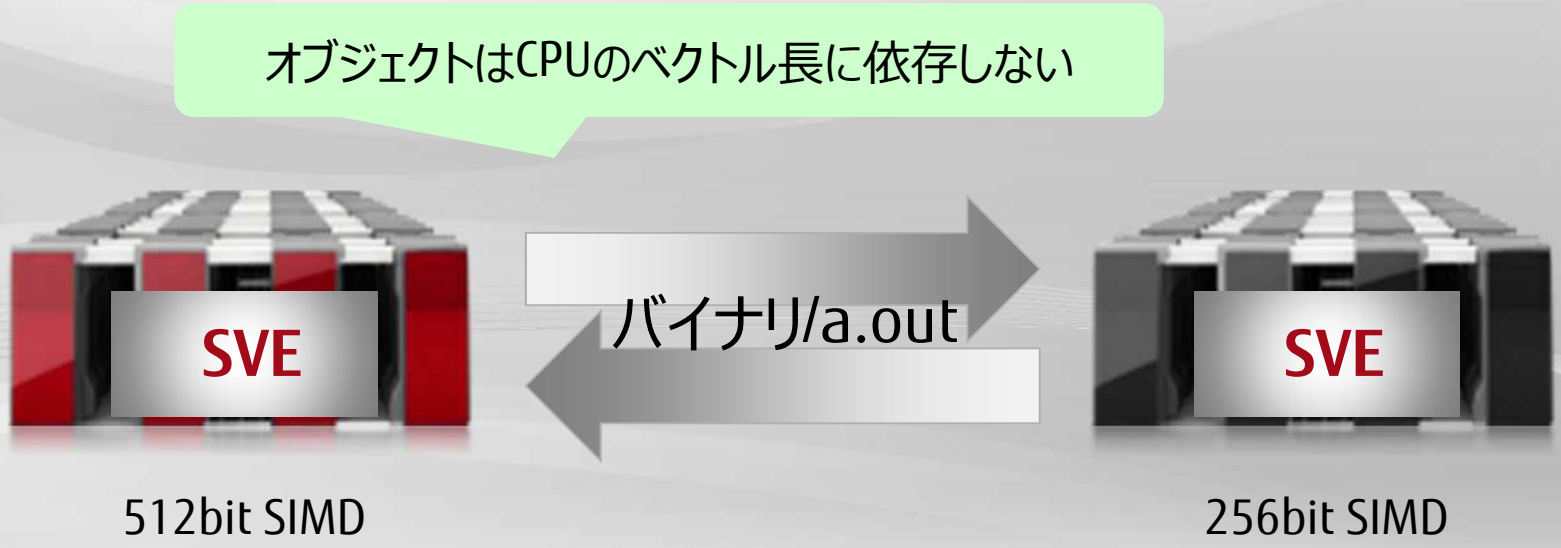


1. Scalable Vector Length

FUJITSU

- ISAがベクトル長を固定していない
  - SVEは128bitから2048bitまで、128の倍数のベクトル長(VL)をサポート
  - ベクトル長は、CPUがサポートする長さに、動的に決定される。
- 異なるベクトル長のCPUで実行可能なバイナリを生成可能
  - ベクトル長を動的に決定可能にするVector-Length Agnostic(VLA)プログラミング

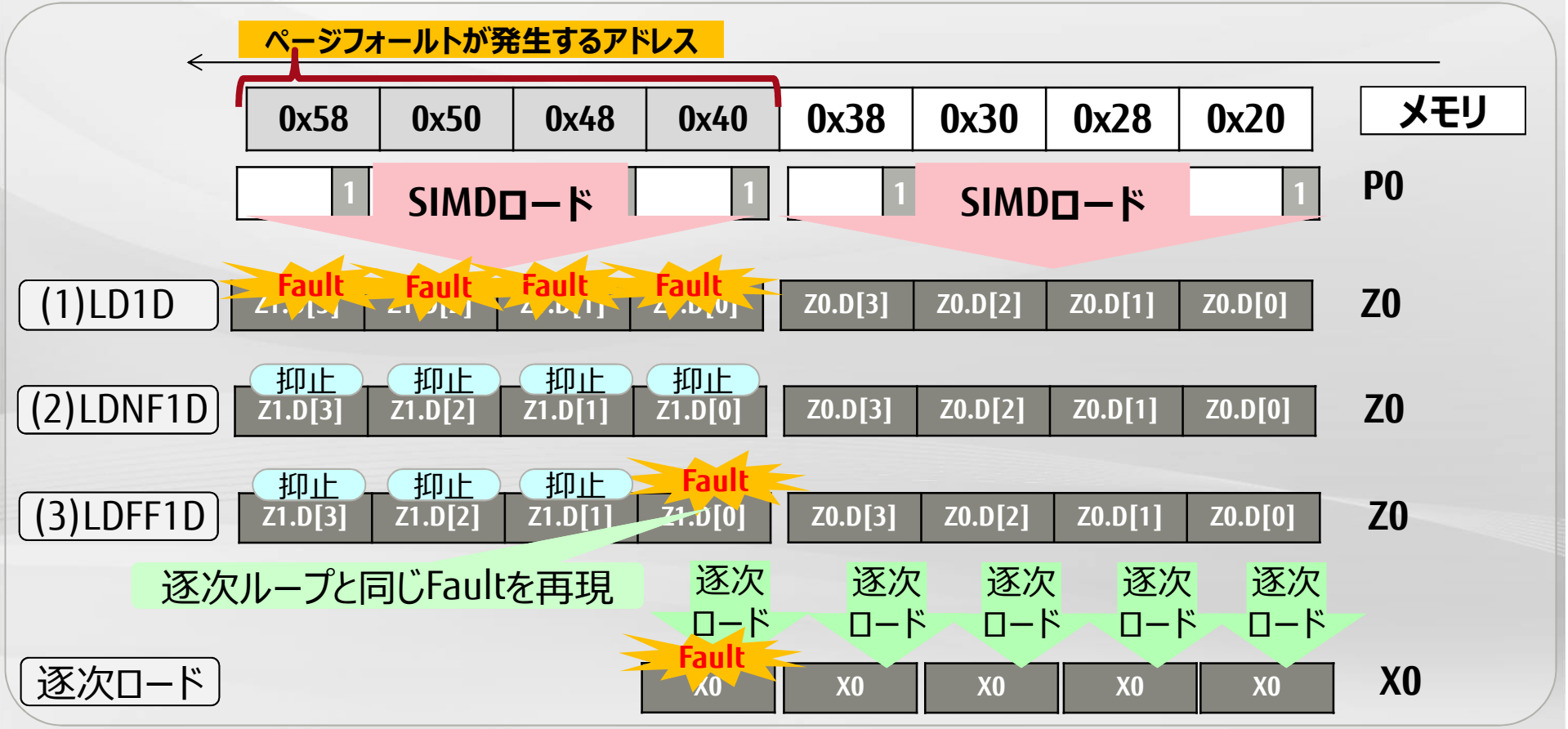
バイナリの可搬性



2. Fault-tolerant Speculative Vectorization

FUJITSU

- Faultの扱いが異なるベクトル化が可能
  - (1) LD1D　：通常SIMD load　⇒ 通常のFaultが発生
  - (2) LDNF1D：Non-faulting SIMD load ⇒ 全てのFaultを抑止
  - (3) LDFF1D：First-fault load　⇒ 最初の有効な要素のFault以外を抑止

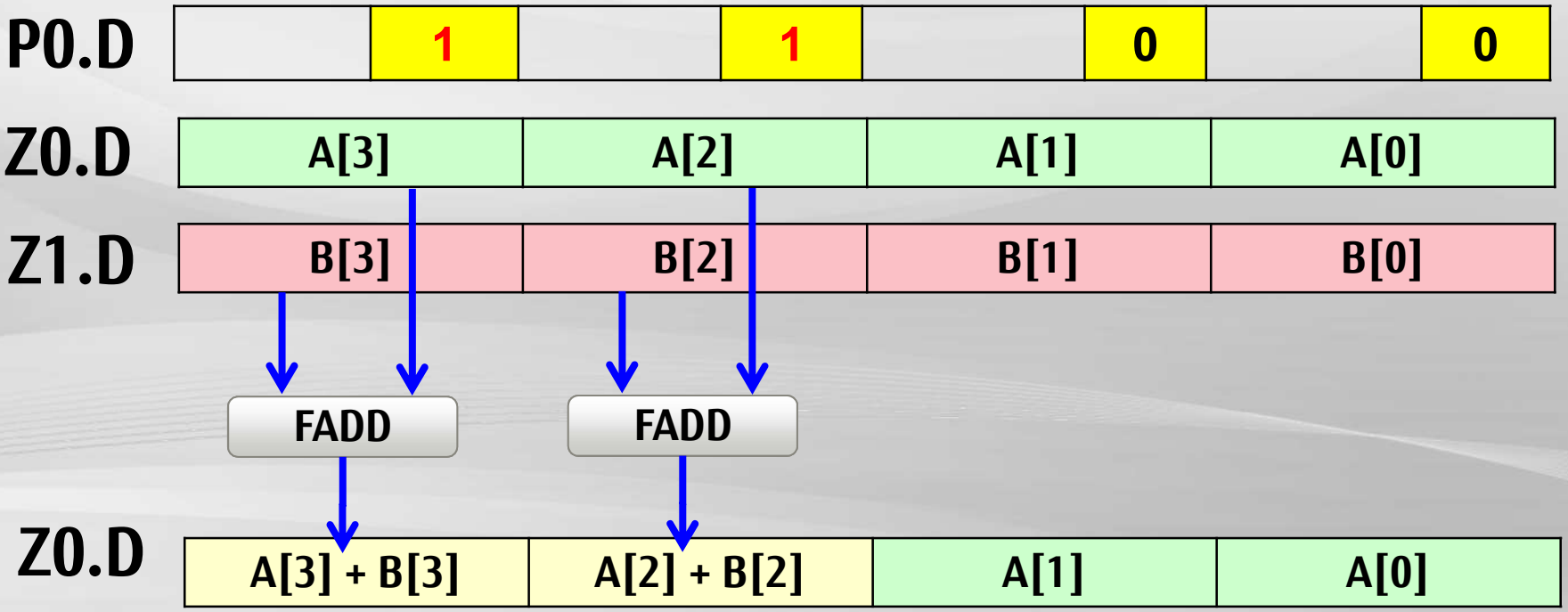


3. Per-lane Predication

FUJITSU

- Predicateレジスタによる要素（Lane)単位の演算制御

```
FADD  Z0.D,   P0/M,   Z0.D,   Z1.D
```



Predicationの活用例

FUJITSU

- if文を含むループのSIMD化
  - **PTRUE命令**で、ALL TRUEを生成
  - **CMPGT命令**で、条件分岐のマスクP1を生成
  - P1によるマスク付きで LD1D, FADD, ST1Dを実行

ソースプログラム

```
int a[n];
double b[n], c[n];
....
for (i=0; i<n; ++i) {
   if (a[i] > 1) {
      b[i] = b[i] + c[i];
   }
}
```

アセンブライメージ

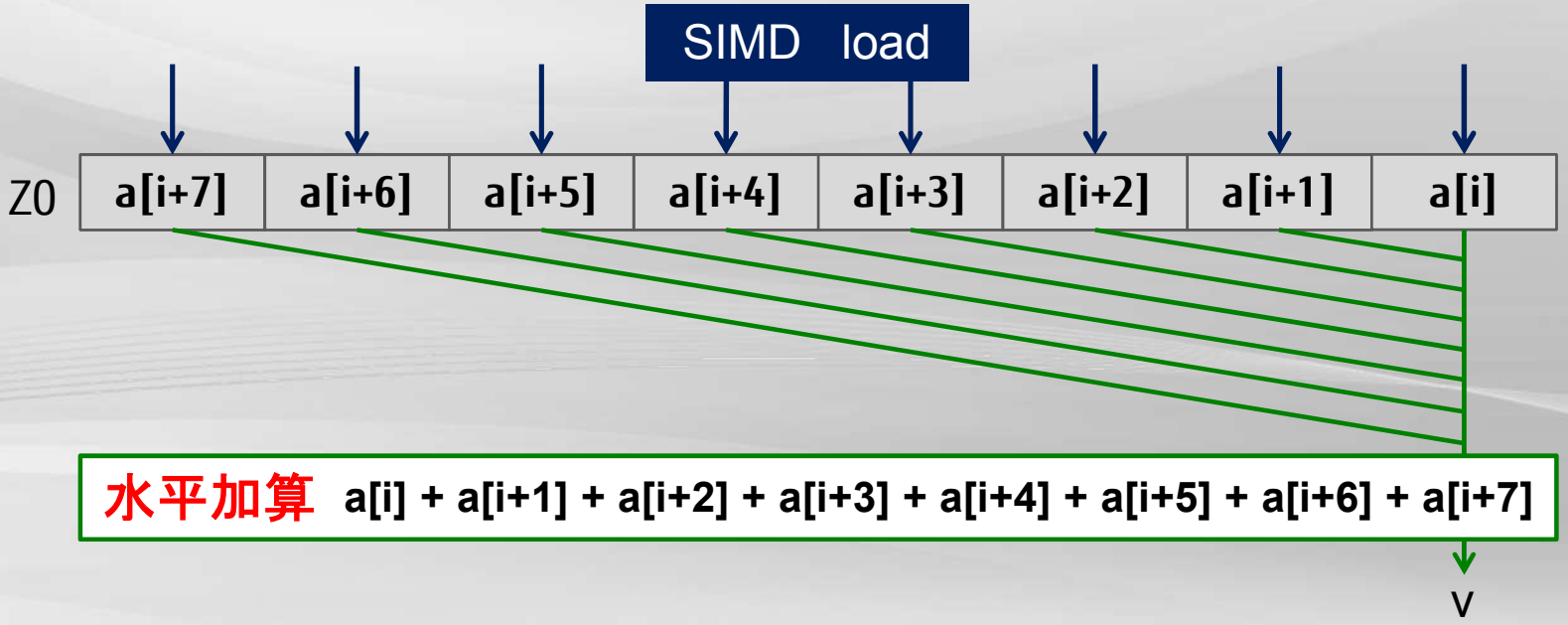
```
PTRUE  P0.D
LD1D   Z0.D, P0/Z, [X0]          // a
CMPGT  P1.D, P0/Z, Z0.D, #1
LD1D   Z1.D, P1/Z, [X1]          // b
LD1D   Z2.D, P1/Z, [X2]          // c
FADD   Z1.D, P1/M, Z1.D, Z2.D
ST1D   Z1.D, P1,   [X1]          // b
```

4. Horizontal and Serialized Vector Operations

◻水平加算命令を利用したSIMD化

```
// 総和プログラム
for (i=0; i<n; ++i) {
  s = s + a[i];
}
```

```
// 水平加算を用いたSIMD化イメージ
for (i=0; i<n; i+=8) {
  Z0 = a[i+7~i];  // SIMD load
  v = 水平加算(Z0);
  s = s + v
}
```



コンパイラの出力コード例

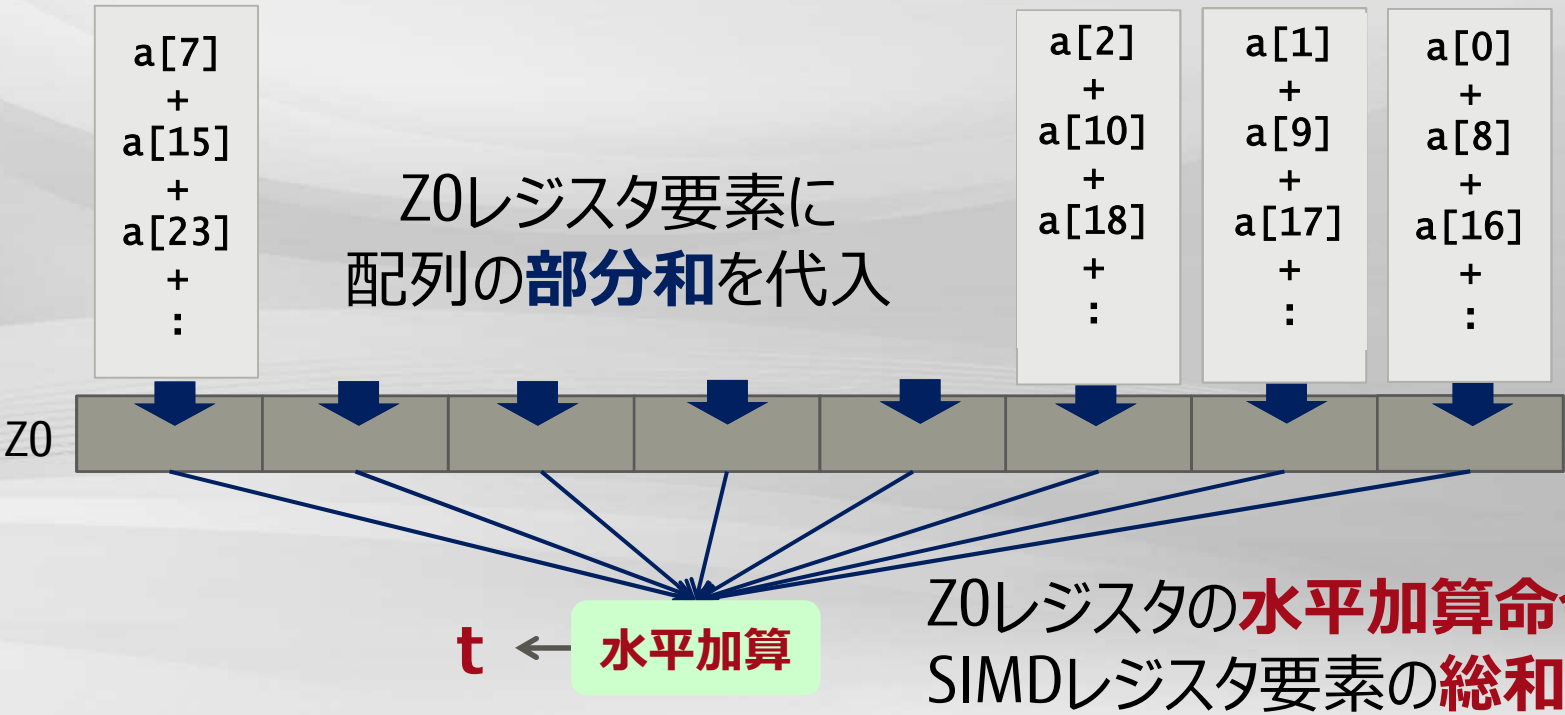
FUJITSU

演算順序の変更を伴うSIMD化

```
// 総和プログラム
for (i=0; i<n; ++i) {
  t = t + a[i];
}
```

```
// 部分和⇒総和
for (i=0; i<n; i+=8) {
  Z0[0] = Z0[0] + a[i];
  Z0[1] = Z0[1] + a[i+1];
     :
  Z0[7] = Z0[7] + a[i+7];
}
t = 水平加算(Z0)
```

SIMD実行

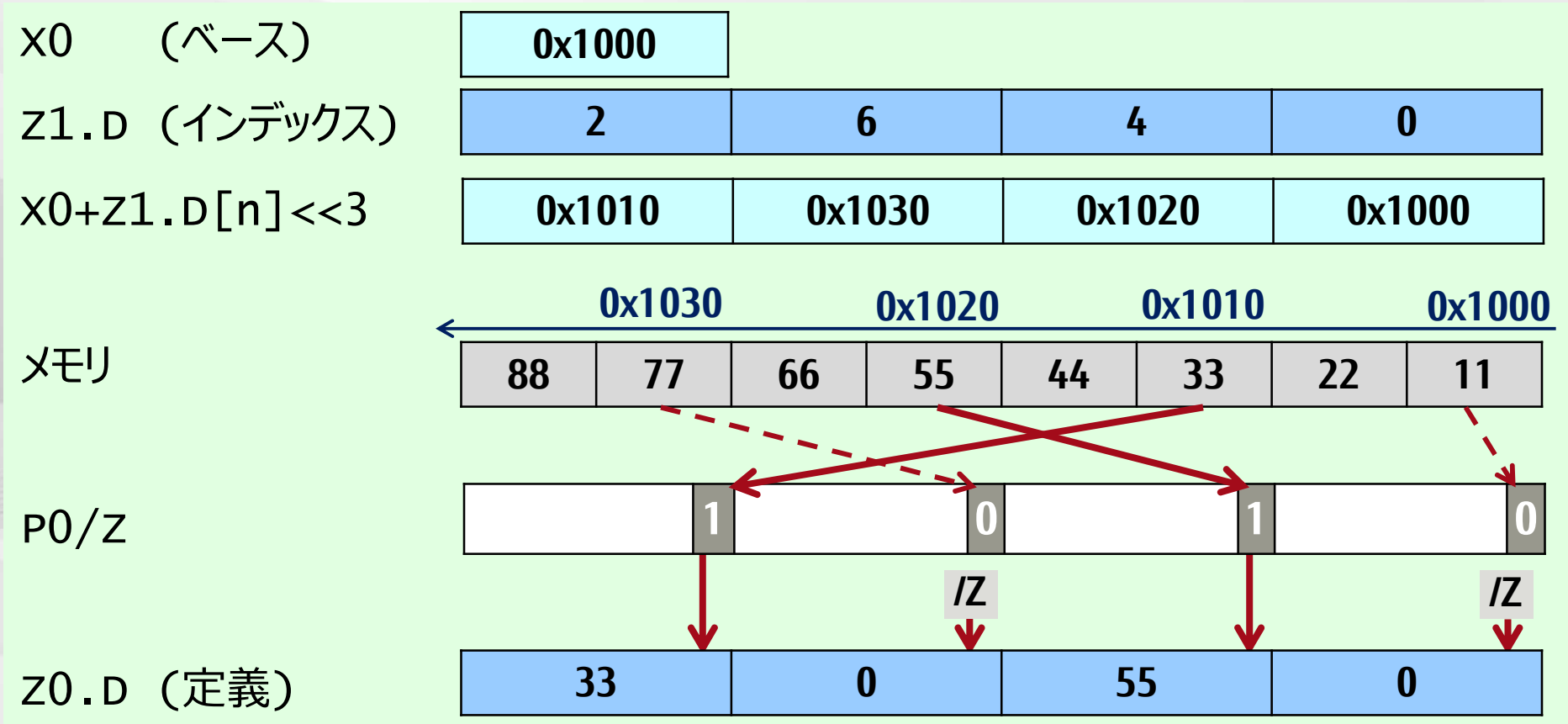


5. Gather-load and Scatter-store

FUJITSU

- Load/StoreともにPredicateによるマスク処理が可能
- ベクトルレジスタでインデックス指定可能なアドレッシングモードをサポート

例) `LD1D Z0.D, P0/Z, [X0, Z1.D, LSL #3]`



Gather Load/Scatter Storeのアドレッシングモード

FUJITSU

- 豊富なアドレッシングモードをサポート

(1) `LD1D Z0.D, P0/Z, [X0, Z0]`
　　ベース　　・・・スカラレジスタ
　　オフセット　・・・ベクトルレジスタ(scale無し)

(2) `LD1D Z0.D, P0/Z, [X0, Z0, LSL #3]`
　　ベース　　・・・スカラレジスタ
　　オフセット　・・・ベクトルレジスタ(scale有り)

(3) `LD1D Z0.D, P0/Z, [Z0, #<imm>]`
　　ベース　　・・・ベクトルレジスタ
　　オフセット　・・・即値

→ **アドレス演算のコンパクト化**

SVEが実現する新しいSIMD化

FUJITSU

SVEの特徴

1. Scalable vector length
2. Fault-tolerant speculative vectorization
3. Per-lane predication
4. Horizontal and serialized vector operations
5. Gather-load and scatter-store

新しくSIMD化できるループ

```
// WHILE ループ
while (A[i] != 0)  {
  i++;
}
```

```
// 飛び出しのあるループ
for (i=0; i<n; ++i) {
  if (x[i] !=0) {break;}
  dst[i] = src[i];
}
```

```
// データ依存のあるループ
for (i=0; i<N; ++i) {
  B[i] = B[A[i]];
}
```

WHILEループのSIMD化

FUJITSU

- First-fault loadとPredicateによるSIMD化
  - Predicateを用いた、ベクトル長に無依存な終端条件と繰り返しの実現
  - 投機的SIMDロードによる、配列AのFaultを再現

ソースプログラム

```
int i = 0;
int A[N];

while (A[i] != 0)  {
  i++;
}
```

スカラコード(ARMv8)

```
      adr   x0, A
      mov   w1, #0
loop: ldr   w1, [x0], #4
      add   w1, w1, 1
      cbnz  w1, loop
```

ベクトルコード(SVE)

```
      adr    x0,  A        // 配列Aのアドレス取得
      mov    x2,  #0       // 変数iの初期化
      dup    z0.s,#0       // 条件式の定数値0を定義
      ptrues p7.s          // 全要素TRUEのpredicate定義

loop:
      setffr               // FFRを初期化
      ldff1w z1.s,  p7/z, [x0, x2, LSL#2]// First-fault load
      rdffr  p0.b,  p7/z                 // FFR読み込み
      cmpeq  p1.s,  p0/z,  z1.s, z0.s // 条件式の比較
      brkbs  p1.b,  p0/z,  p1.b       // breakの設定
      incp   x2,  p1.s                // 添え字iをインクリメント
      b.last loop                     // breakの設定に従って分岐
```

データ依存があるループのSIMD化

FUJITSU

- ループ全体をSIMD化
- データ依存がある部分を、ベクトルレジスタの水平方向に繰返し
- 依存がない要素を動的に判定し、Predicateを用いてSIMD実行

ソースプログラム

```
int i;
int A[n], B[n];
for(i=0;i<n;++i) {
  B[i] = B[A[i]];
}
```

スカラコード(ARMv8)

```
   mov   w0, 0
   adr   x1, A
   adr   x2, B
loop:
   ldrsw x4, [x1,x0,lsl #2]
   ldr   w4, [x2,x4,lsl #2]
   str   w4, [x2,x0,lsl #2]
   add   w0, w0, 1
   cmp   w0, x5 // i<n
   blt   loop
```

ベクトル化したループの疑似コード(SVE)

```
// 16要素SIMDの場合
for (i=0; i<n; i+=16)
{
  vec1  = vload(A[i])      // A[i]をLoad
  pred1 = cmplt(vec1, n)   // i<nを表すPredicate
  pred3 = pred1            // Predicateをコピー

  // ベクトルの要素を水平方向に処理
  do {
     // 依存が無い要素を表すPredicateを生成
    pred2 = dependence_resolved(i, vec1, pred3)
     //  B[A[i]]をLoad
    vec2  = vgather_load(B[vec1]], pred2)
     //  B[i]をStore
    vstore(B[i], pred2) = vec2
     // 実行した要素を記録
     pred3 = bitclear(pred3, pred2)
  } while (pred3に未実行の要素がある);

  i = i + pcnt(pred1) // pred1の有効な要素の数だけ、iをインクリメント
}
```

ポスト「京」の機能比較

FUJITSU

	Post-K	FX100	K computer	Intel Xeon Phi
SIMD拡張命令セット	SVE	HPC-ACE2	HPC-ACE	AVX-512
SIMD幅（bit）	512	256	128	512
積和演算（4オペランド）	○	○	○	×
逆数近似（除算、SQRT）	○	○	○	○
三角関数補助命令	○	○	○	×
指数関数補助命令	○	○	×	○
パックド単精度演算※	○	△	×	○
ハードウェアバリア	○	○	○	×
セクタキャッシュ	○	○	○	×
ハードウェアプリフェッチ	○	○	○	○

△・・・対象命令が限定的

※・・・倍精度に対して二倍の要素数を実行する命令

まとめ

FUJITSU

FUJITSU

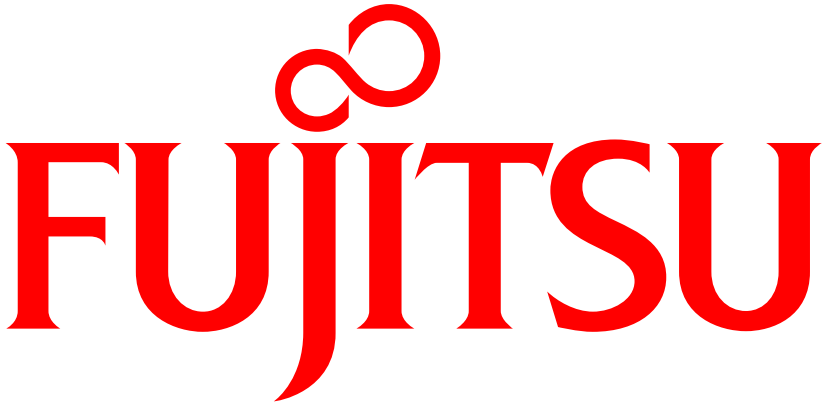
- スパコン向けに進化してきたISA
- 実績のあるマイクロアーキ
- コンパイラのコード生成技術

ARM

- アプリの効率的実行を追求したISA
- オープンなアーキテクチャ
- モバイル ⇒ サーバ ⇒ HPCへ拡大中

SIMD命令セットアーキテクチャ 「SVE」

先進ベンダとの積極的なコラボレーション、先端テクノロジの確実な実装を通して、ポスト「京」を成功させます。



shaping tomorrow with you