Linux で DMA バッファを mmap するとキャッシュが効かない場合がある

Kernel/VM 探検隊@北陸 Part 6

2023 年 12 月 2 日初版

@ikwzm

自己紹介みたいなもの

- ハンドルネーム: ikwzm
- 現在隠居中
- もうすぐ 58 才(けっこう年)
- 主に論理回路設計（回路図～VHDL)
- たまにプログラム(アセンブラ～ C/Ruby)
- https://github.com/ikwzm

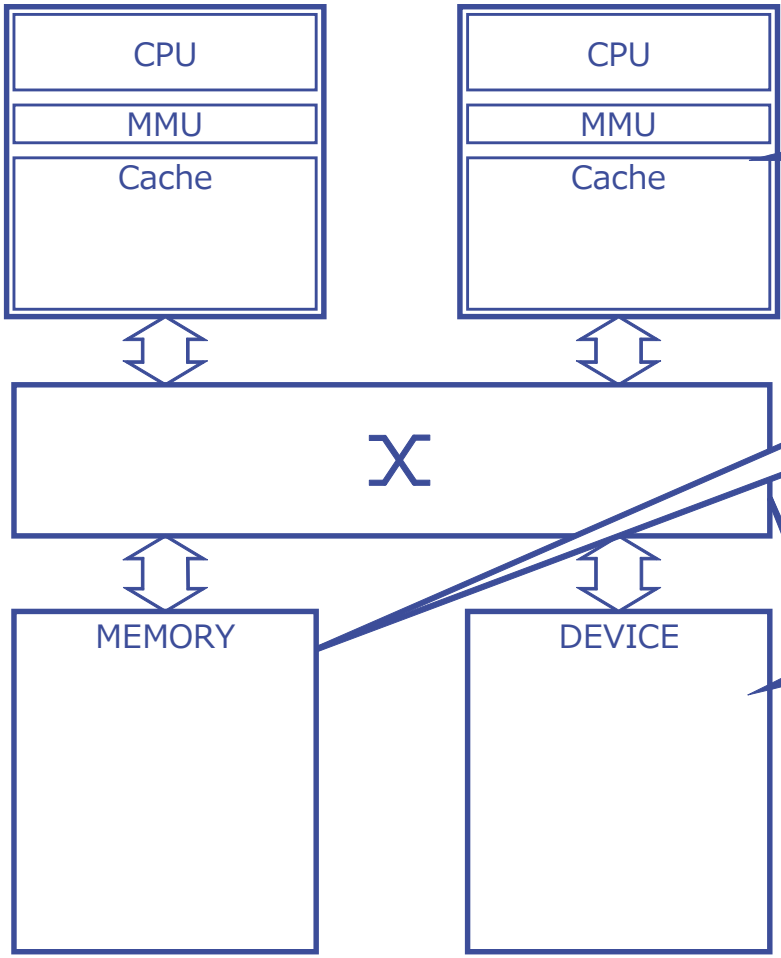
お品書き

- 背景
  - 主な登場人物の紹介
- Cache Coherency 問題
  - Cache Coherency 問題とは？
  - Cache Coherency 問題を解決する方法
- Cache Aliasing 問題
  - Cache Aliasing 問題とは？
  - Cache Aliasing 問題を解決する方法
- Linux で DMA バッファを mmap するとキャッシュが効かない仕組み
  - dev_is_dma_coherent() の役割
  - キャッシュが効かなくなる場合の例

お品書き

- 背景
  - 主な登場人物の紹介
- Cache Coherency 問題
  - Cache Coherency 問題とは？
  - Cache Coherency 問題を解決する方法
- Cache Aliasing 問題
  - Cache Aliasing 問題とは？
  - Cache Aliasing 問題を解決する方法
- Linux で DMA バッファを mmap するとキャッシュが効かない仕組み
  - dev_is_dma_coherent() の役割
  - キャッシュが効かなくなる場合の例

主な登場人物の紹介 -1-



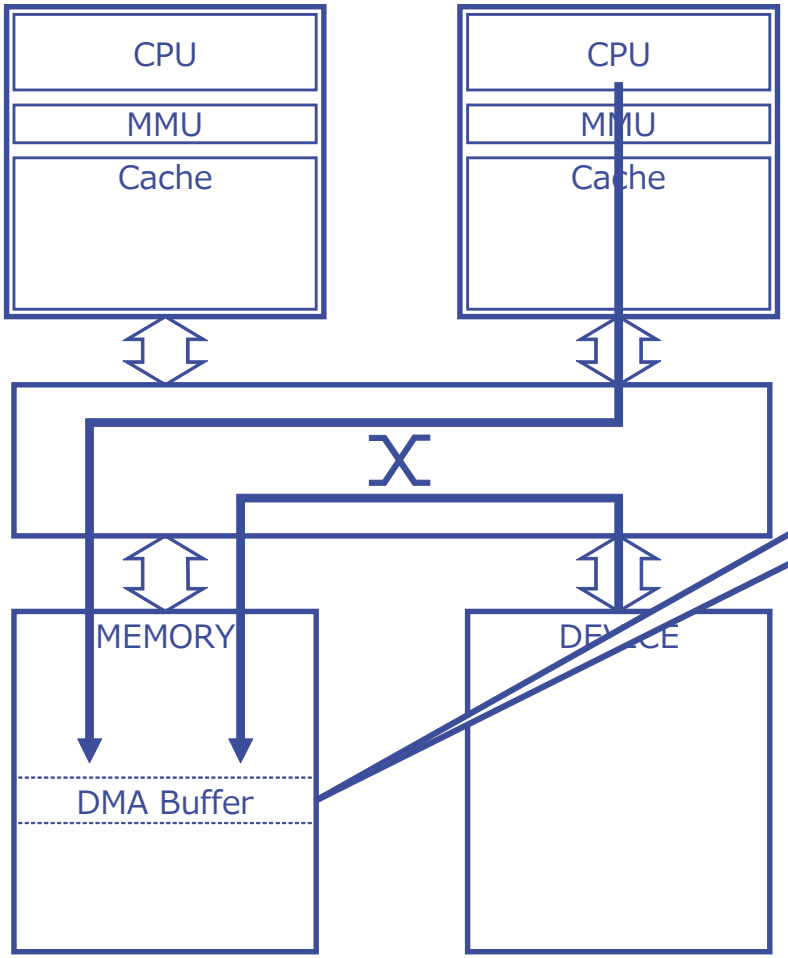
CPU
- Linux が動作する Processor
- 仮想メモリをサポート
- Cache を持ってる

MEMORY
- Linux が管理する主記憶

DEVICE
- CPU 以外の各種コンポーネント
- USB IF や Net IF 等の IO 系
- GPU や TPU 等のアクセラレーター系

InterConnect
- CPU Memory DEVICE を相互接続

主な登場人物の紹介 -2-



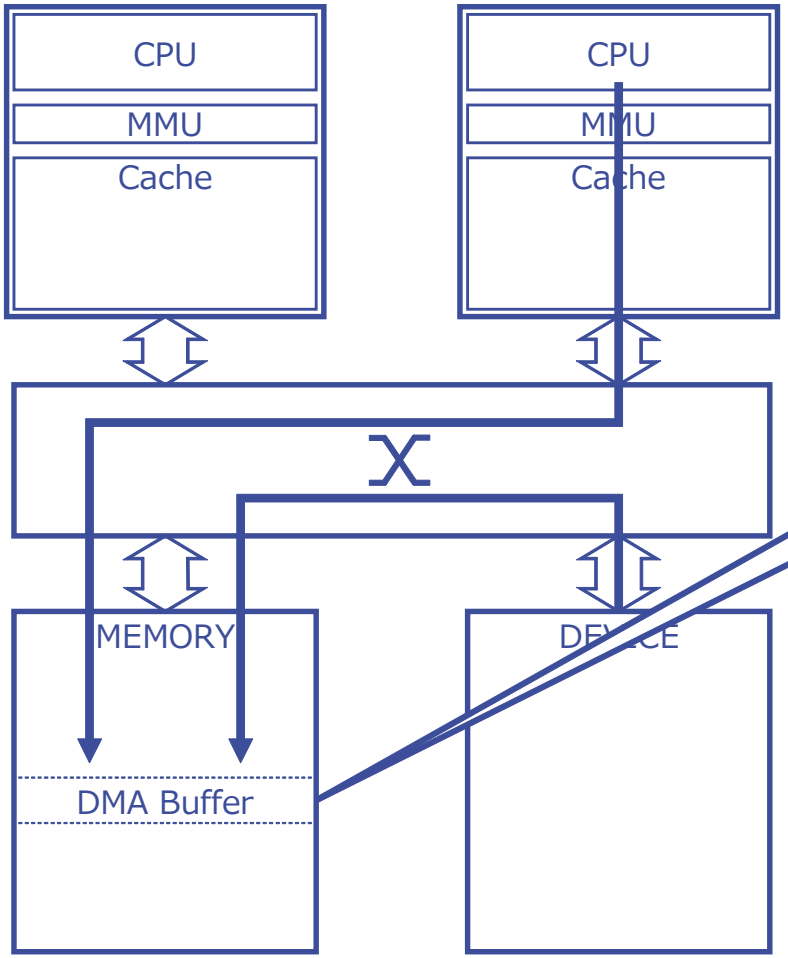
DMA (Direct Memory Access)

- DEVICE が MEMORY に直接アクセス (Direct Memory Access) する方法

DMA Buffer

- CPU と DEVICE からアクセス可能な MEMORY 上に確保した DMA のための領域

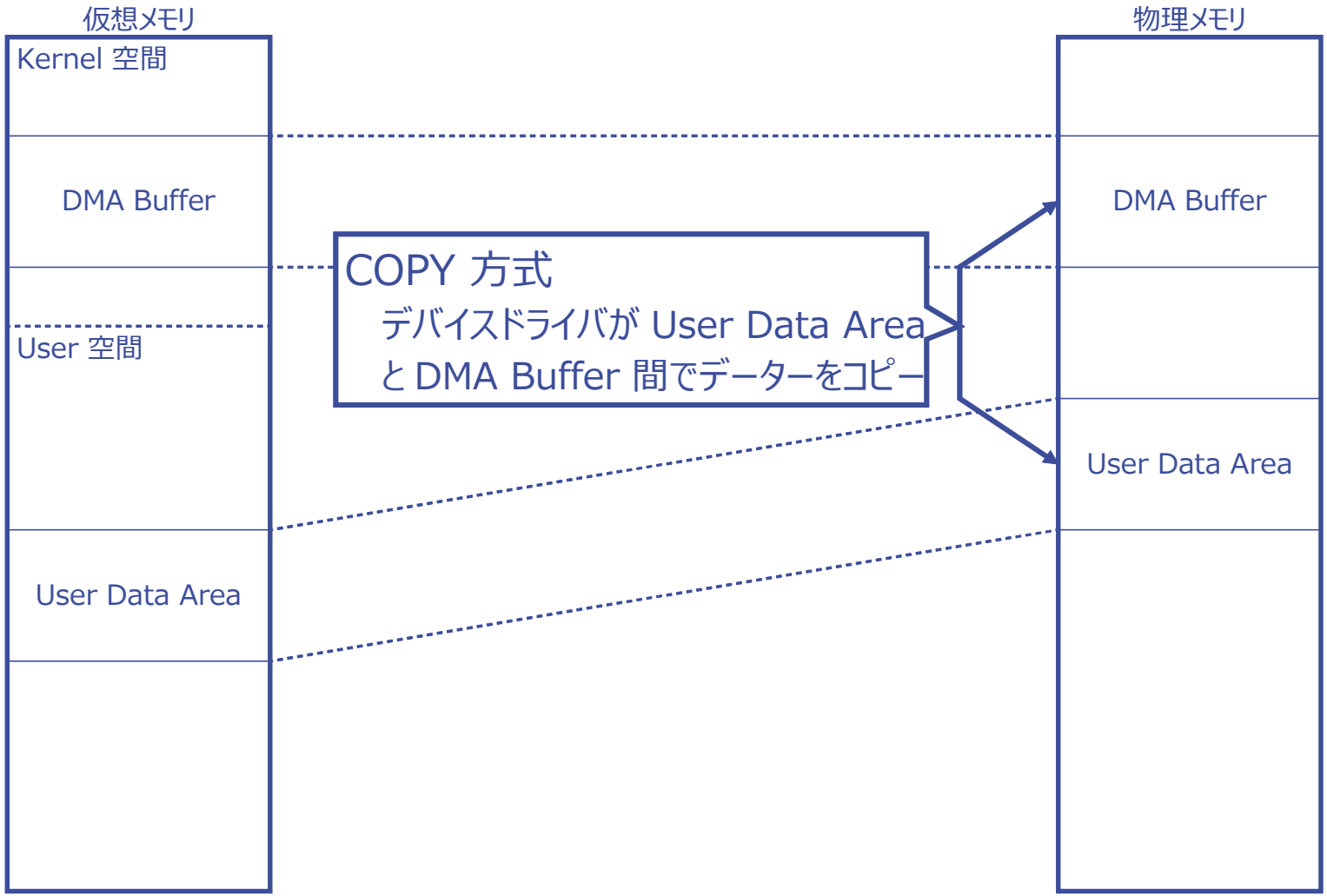
主な登場人物の紹介 -2-



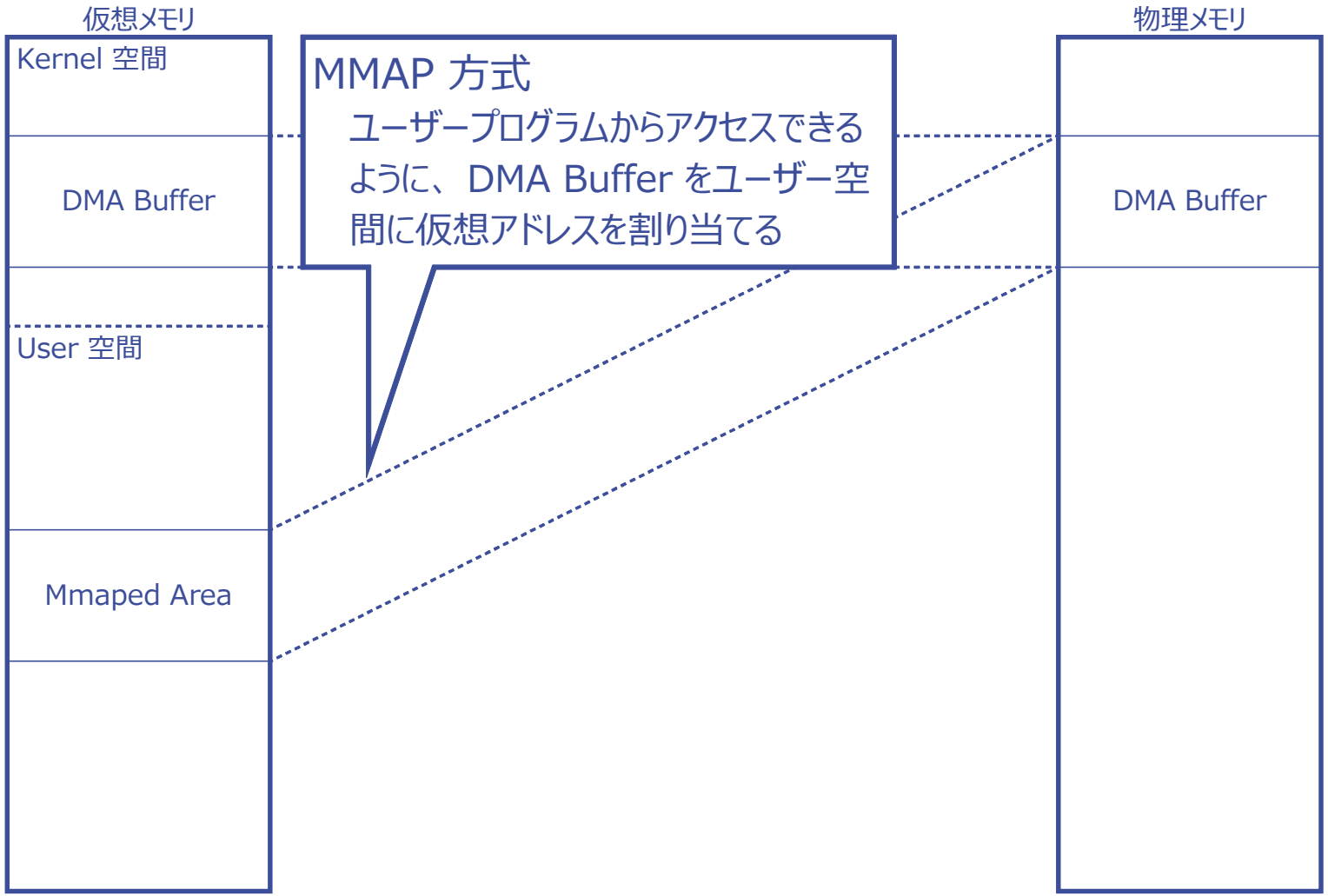
DMA (Direct Memory Access)
- DEVICE が MEMORY に直接アクセス (Direct Memory Access) する方法

DMA Buffer
- CPU と DEVICE からアクセス可能な MEMORY 上に確保した DMA のための領域
- Linux では Kernel の管理下にある
  - 仮想メモリと連携する
  - Kernel 空間に配置
  - DMA Mapping API

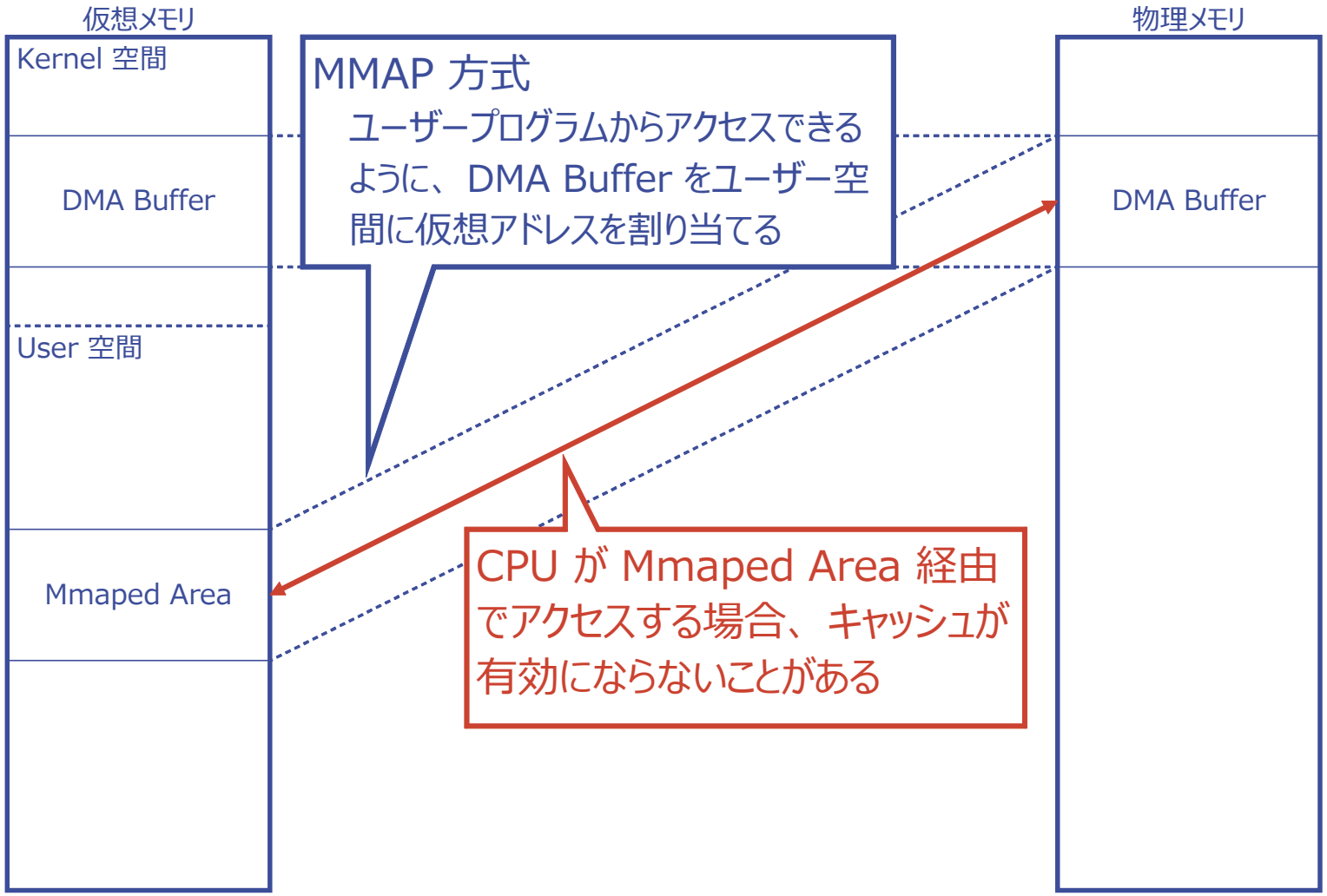
ユーザープログラムと DMA Buffer でデータを受け渡す方法 -1-



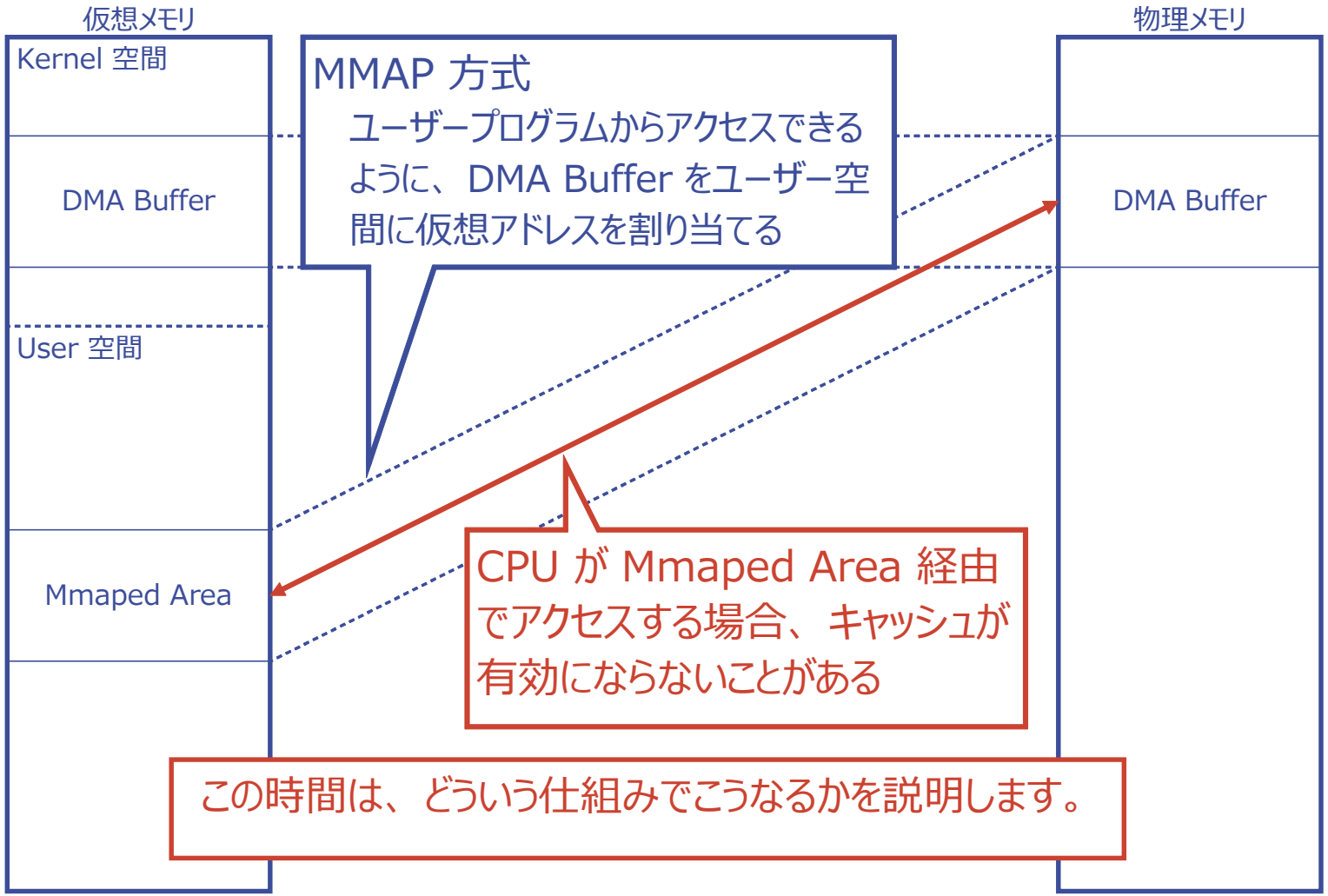
ユーザープログラムと DMA Buffer でデータを受け渡す方法 -2-



ユーザープログラムと DMA Buffer でデータを受け渡す方法 -2-



ユーザープログラムと DMA Buffer でデータを受け渡す方法 -2-



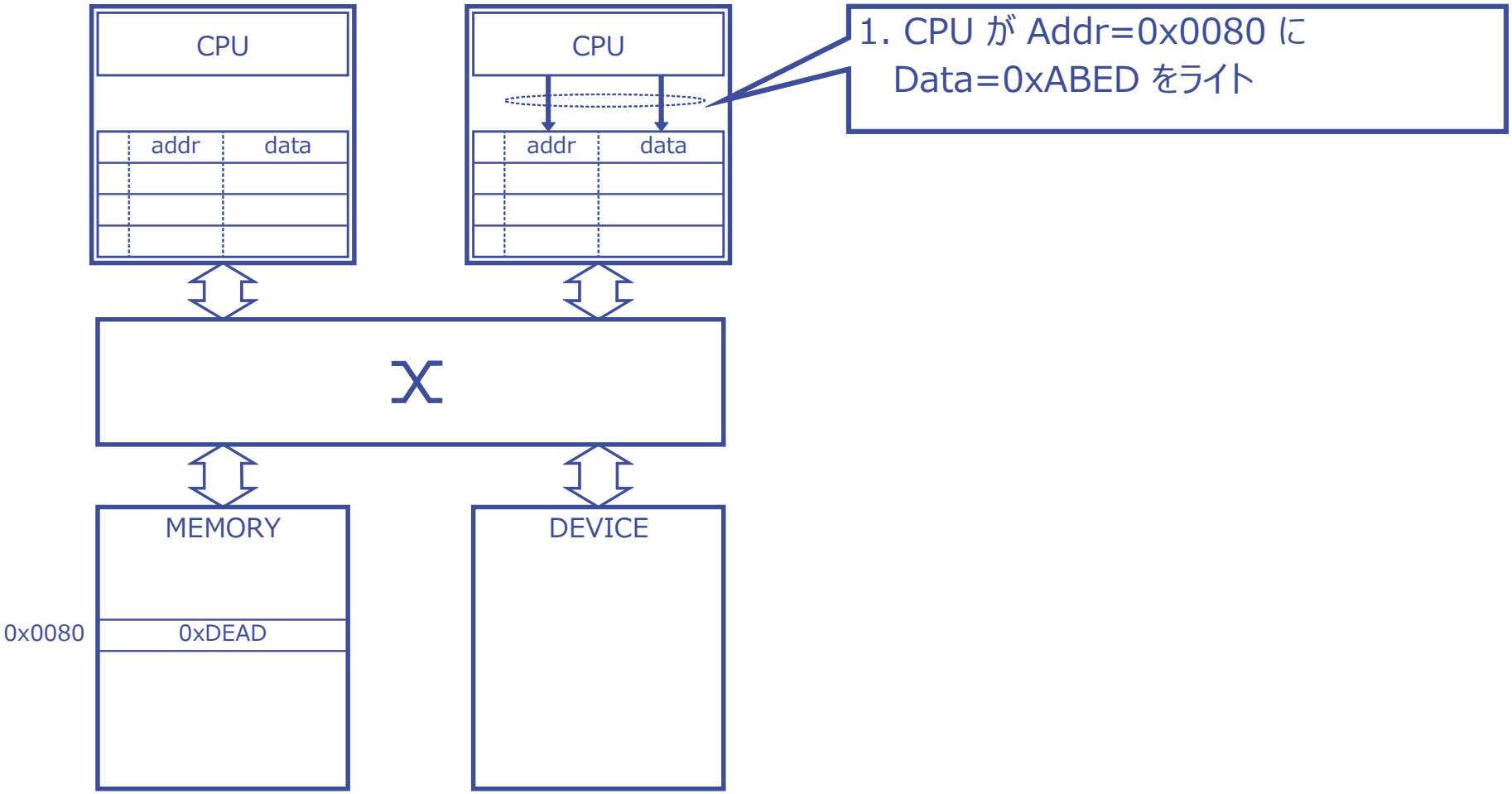
お品書き

- 背景
  - 主な登場人物の紹介
- Cache Coherency 問題
  - Cache Coherency 問題とは？
  - Cache Coherency 問題を解決する方法
- Cache Aliasing 問題
  - Cache Aliasing 問題とは？
  - Cache Aliasing 問題を解決する方法
- Linux で DMA バッファを mmap するとキャッシュが効かない仕組み
  - dev_is_dma_coherent() の役割
  - キャッシュが効かなくなる場合の例

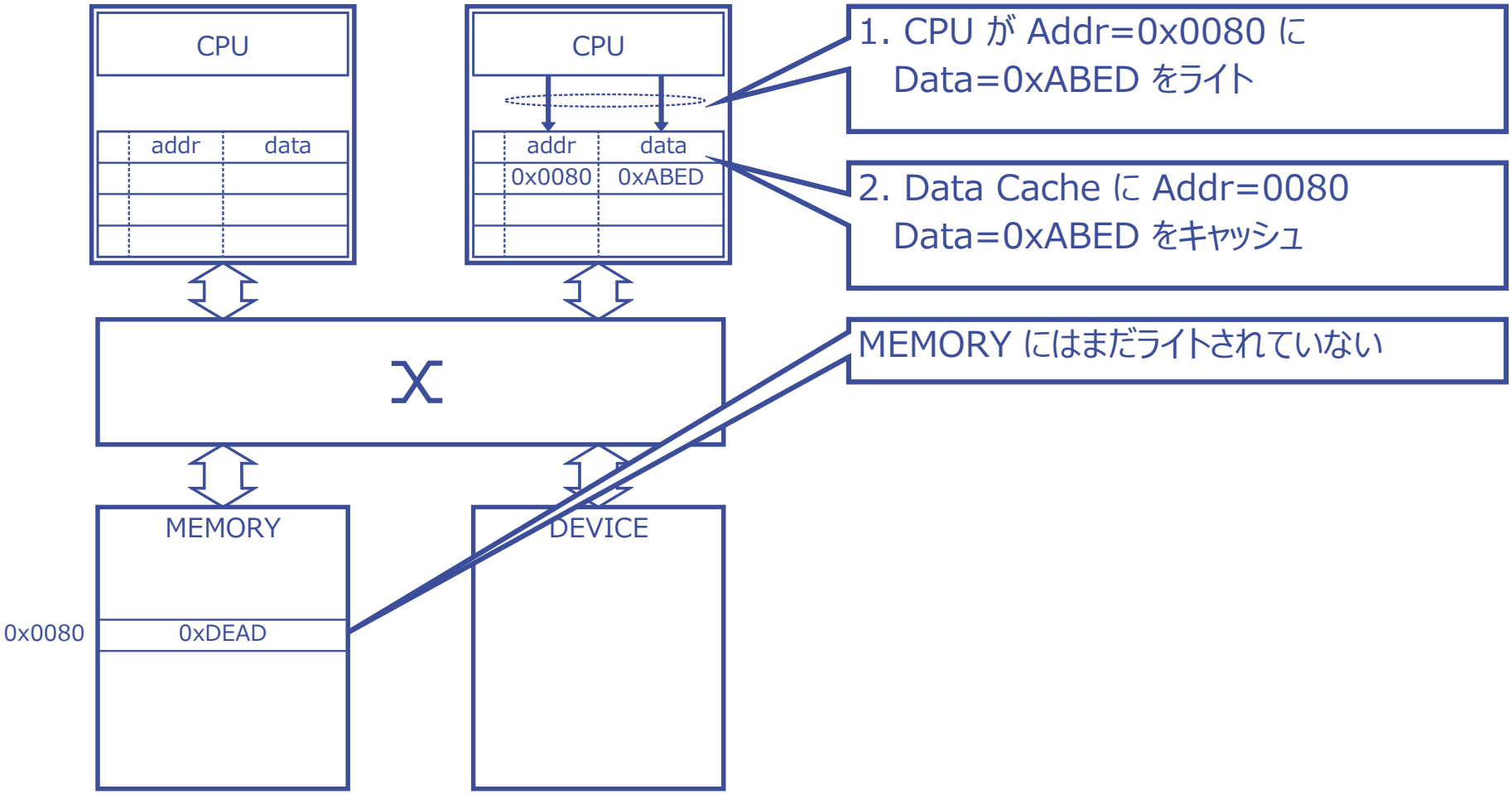
お品書き

- 背景
  - 主な登場人物の紹介
- Cache Coherency 問題
  - Cache Coherency 問題とは？
  - Cache Coherency 問題を解決する方法
- Cache Aliasing 問題
  - Cache Aliasing 問題とは？
  - Cache Aliasing 問題を解決する方法
- Linux で DMA バッファを mmap するとキャッシュが効かない仕組み
  - dev_is_dma_coherent() の役割
  - キャッシュが効かなくなる場合の例

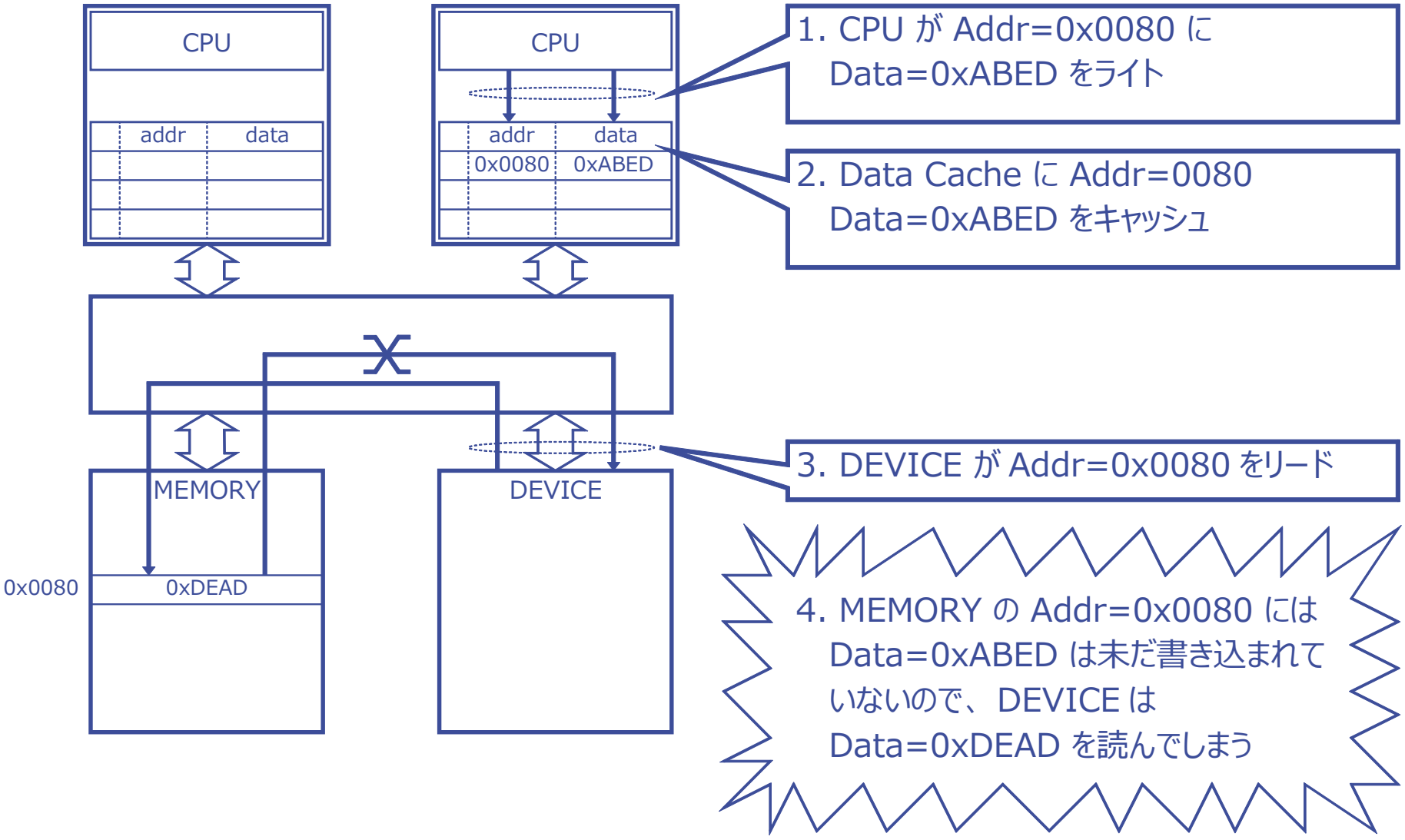
Cache Coherency で問題が発生するケース - 1 -



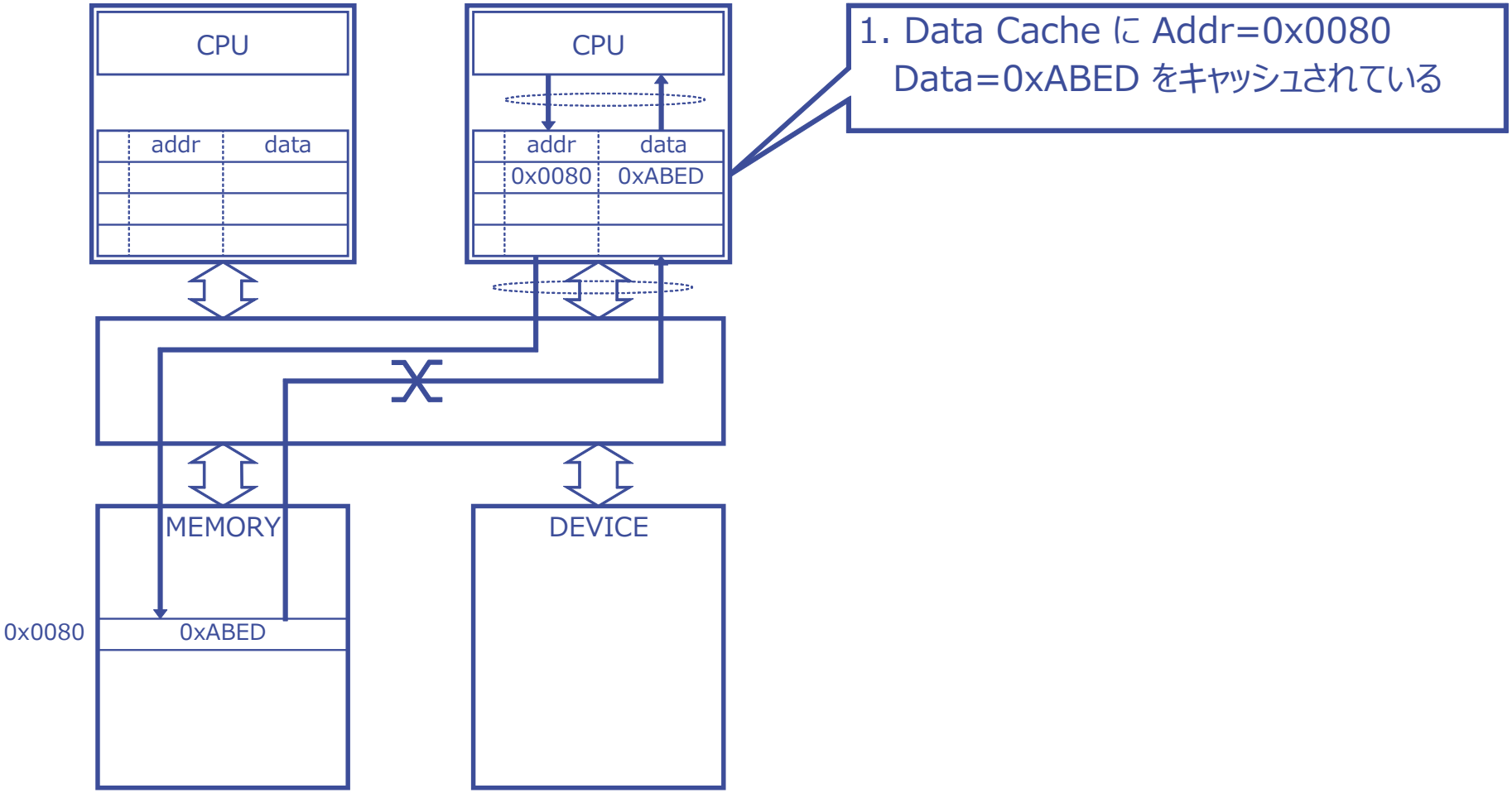
Cache Coherency で問題が発生するケース - 1 -



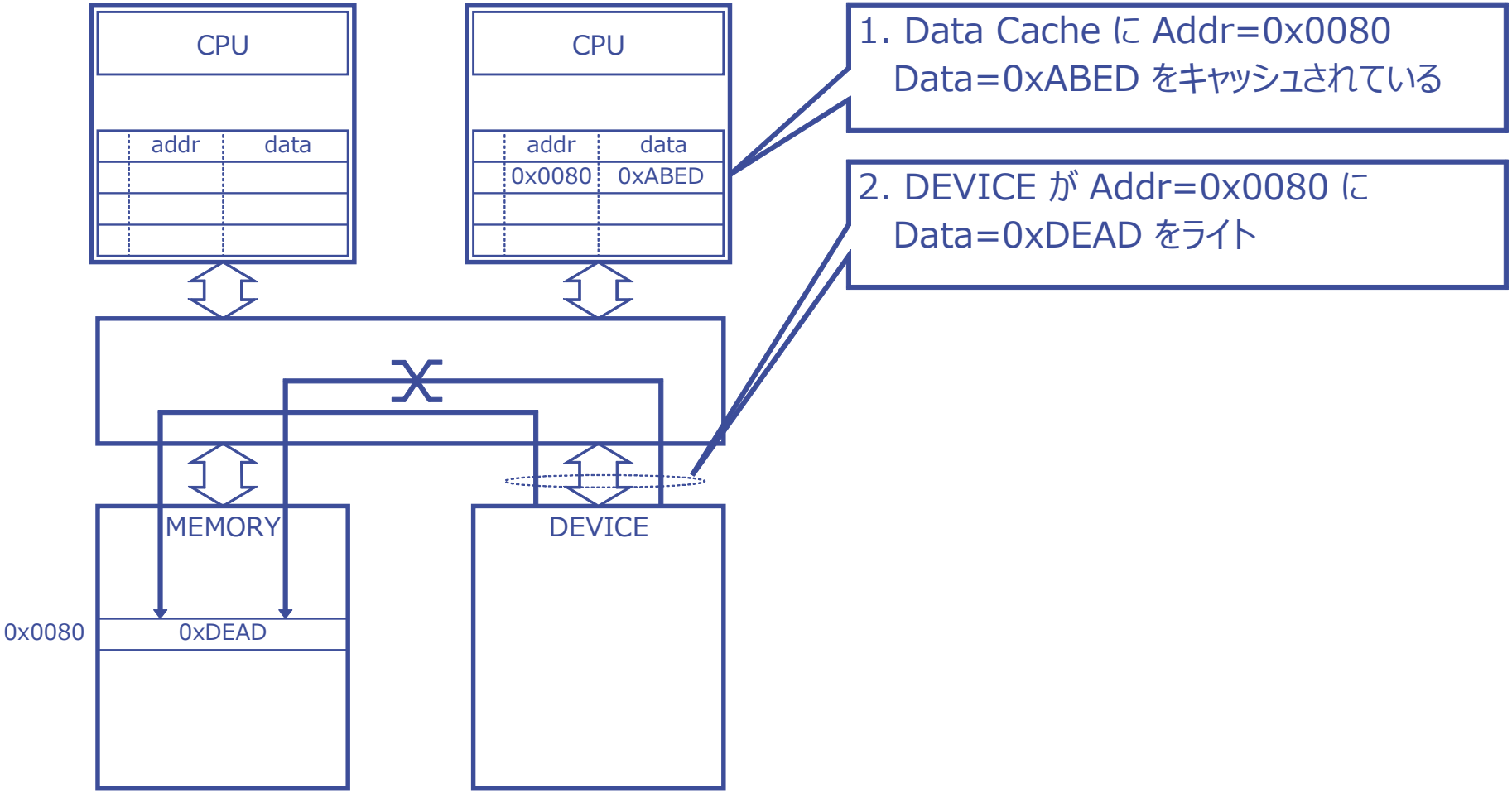
Cache Coherency で問題が発生するケース - 1 -



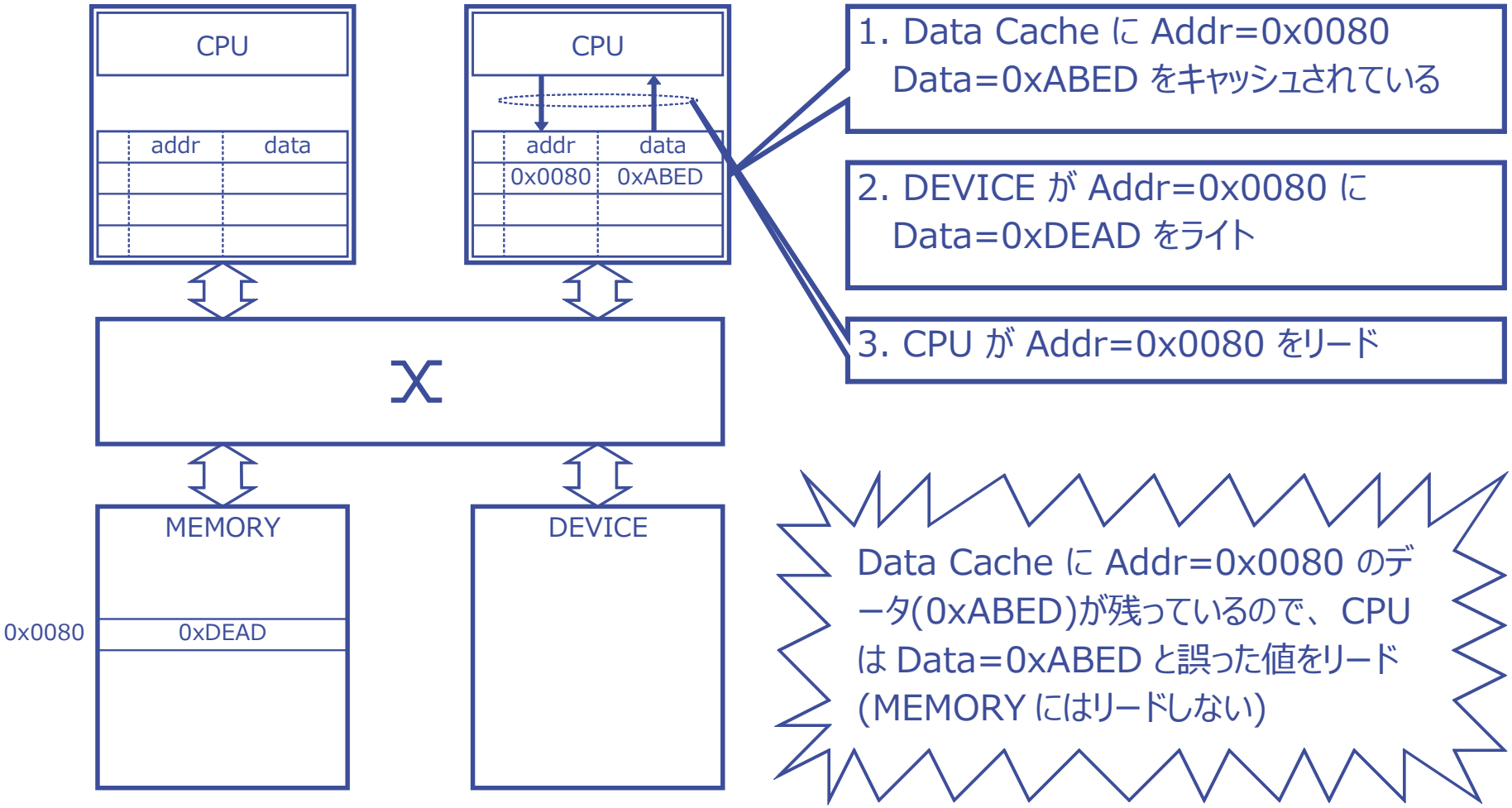
Cache Coherency で問題が発生するケース -2-



Cache Coherency で問題が発生するケース -2-



Cache Coherency で問題が発生するケース -2-



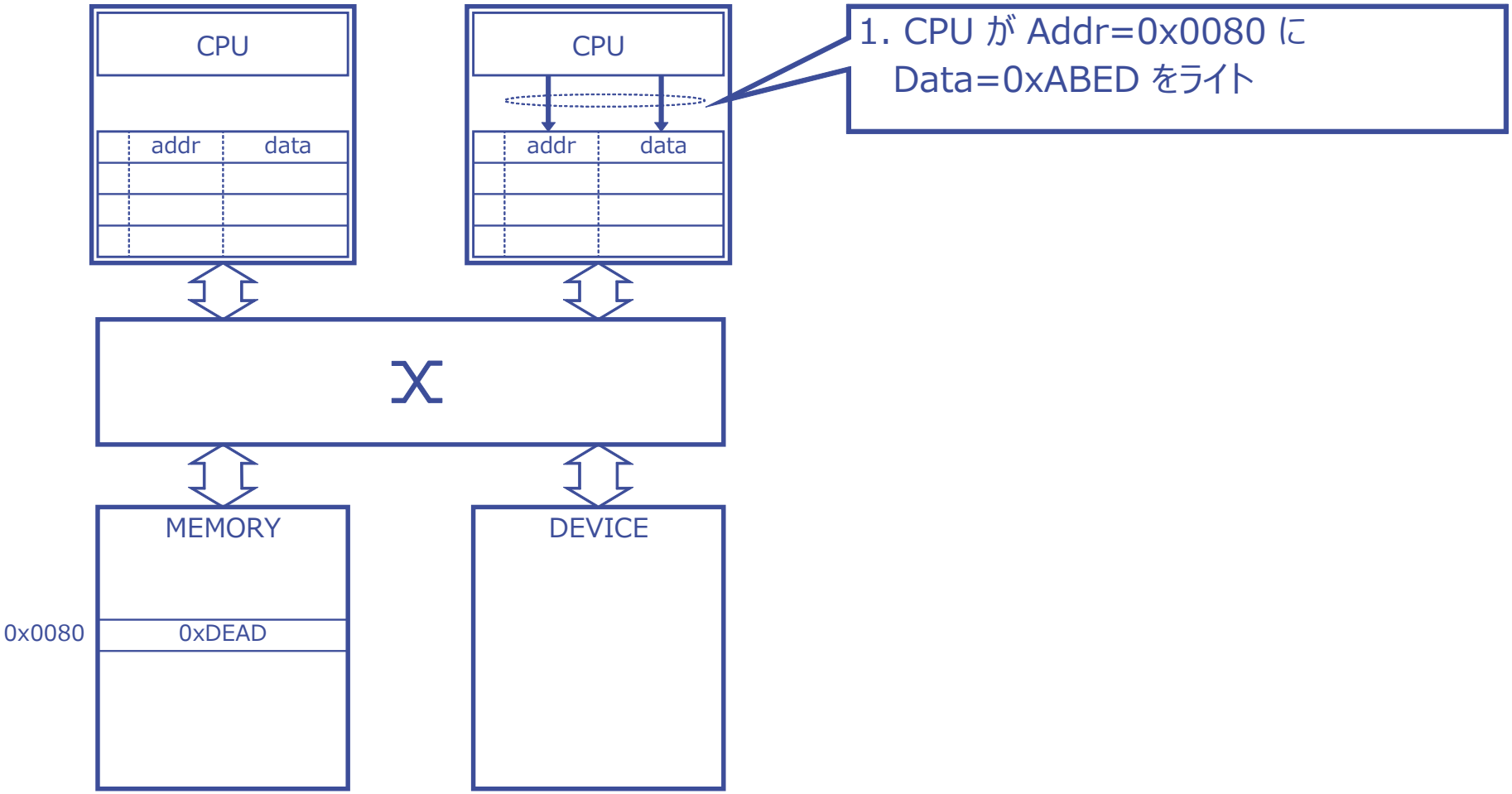
Cache Coherency 問題を解決する方法

- キャッシュを使わない
- ソフトウェアでなんとかする方法
  ソフトウェアで Cache Flush/Invalidiate する
- ハードウェアでなんとかする方法
  Cache Coherency に対応した Cache と Interconnect

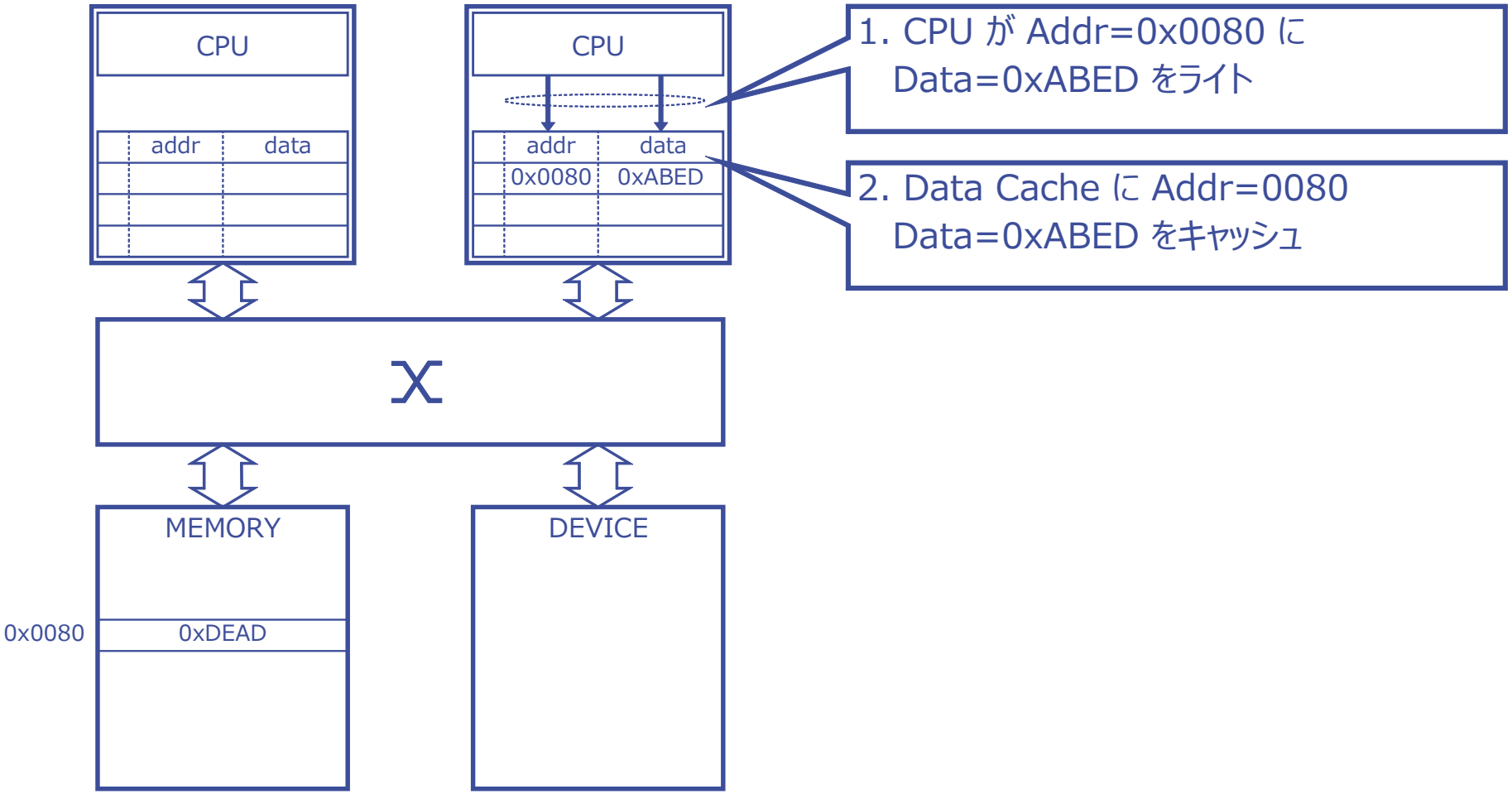
キャッシュを使わない方法

- ソフトウェアの実装負担: 軽い
  - 実質 MMU の設定のみ
- ハードウェアの実装負担: 軽い
  - MMU が必要だが、大抵の CPU は初期装備されている
- CPU からのアクセス速度: ちょ～遅い

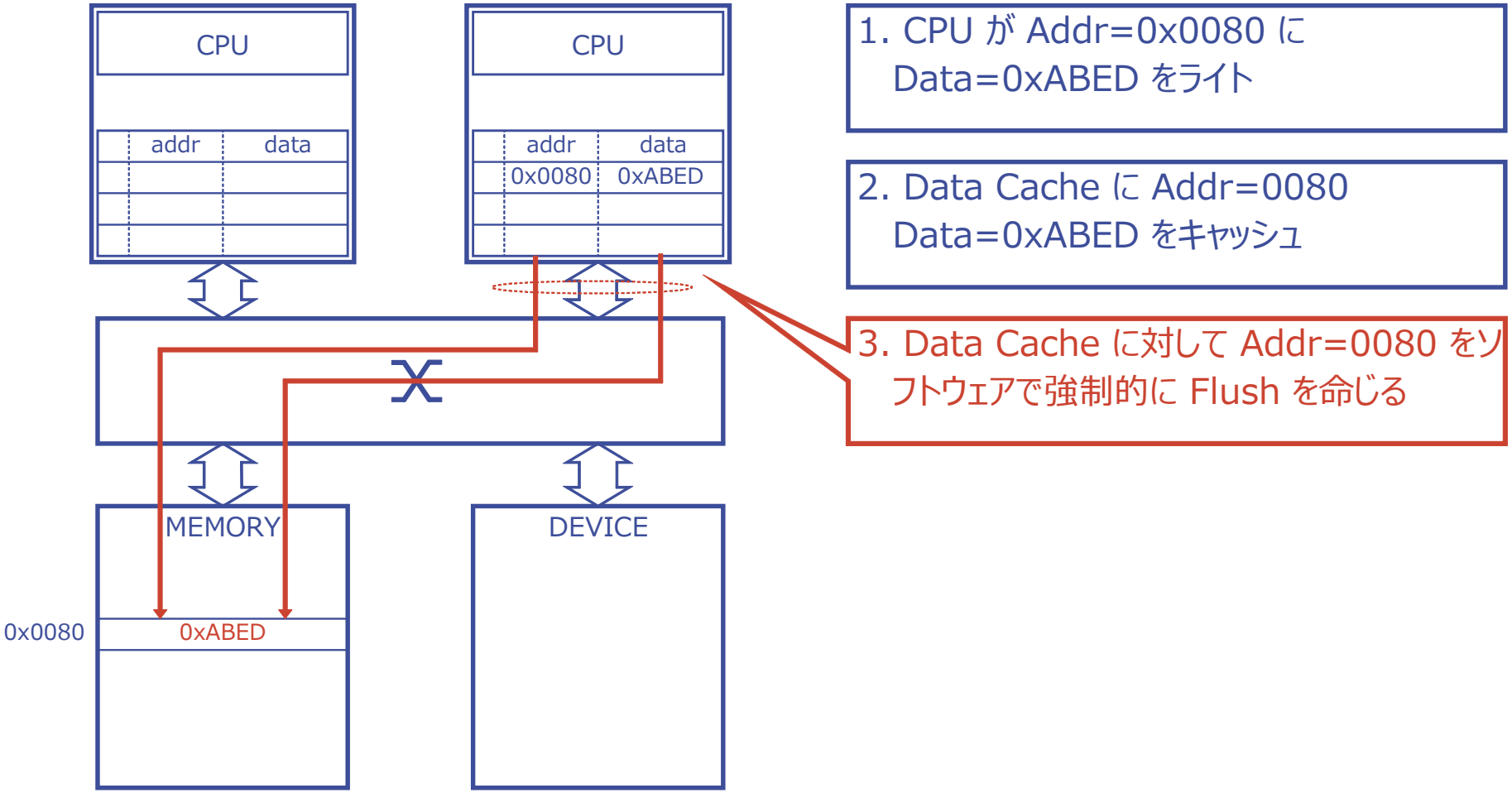
ソフトウェアでなんとかする方法　- 1 -



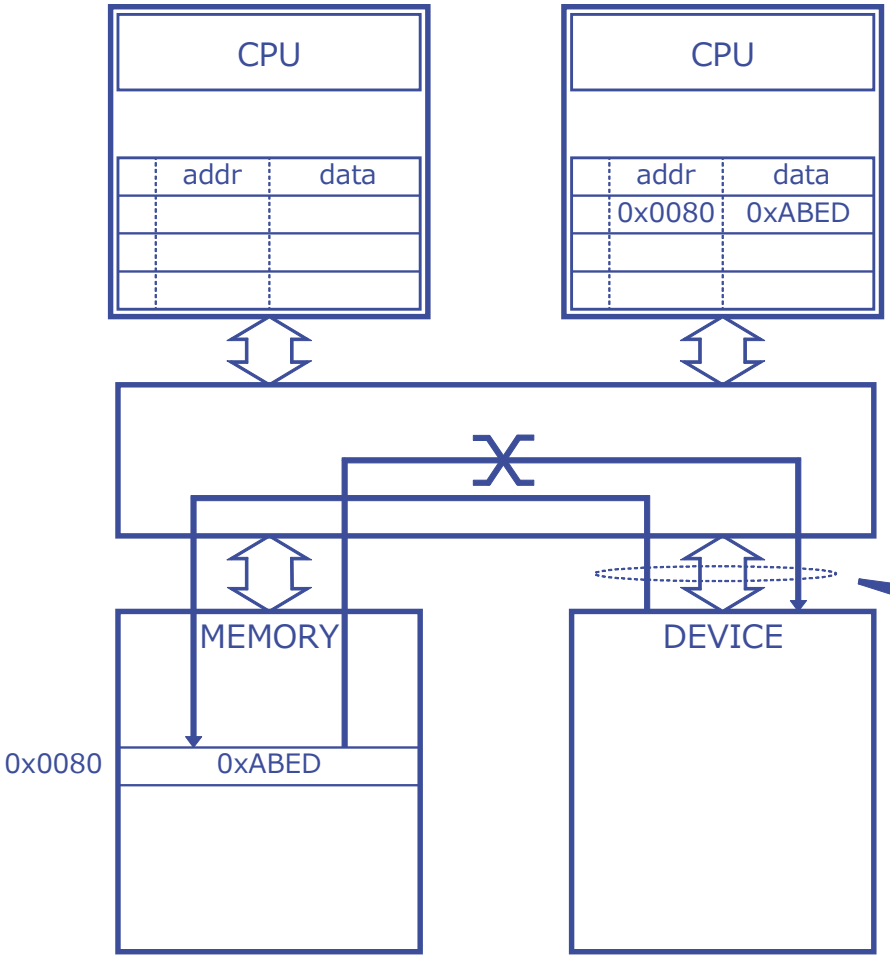
ソフトウェアでなんとかする方法　- 1 -



ソフトウェアでなんとかする方法　- 1 -



ソフトウェアでなんとかする方法　- 1 -



1. CPU が Addr=0x0080 に Data=0xABED をライト

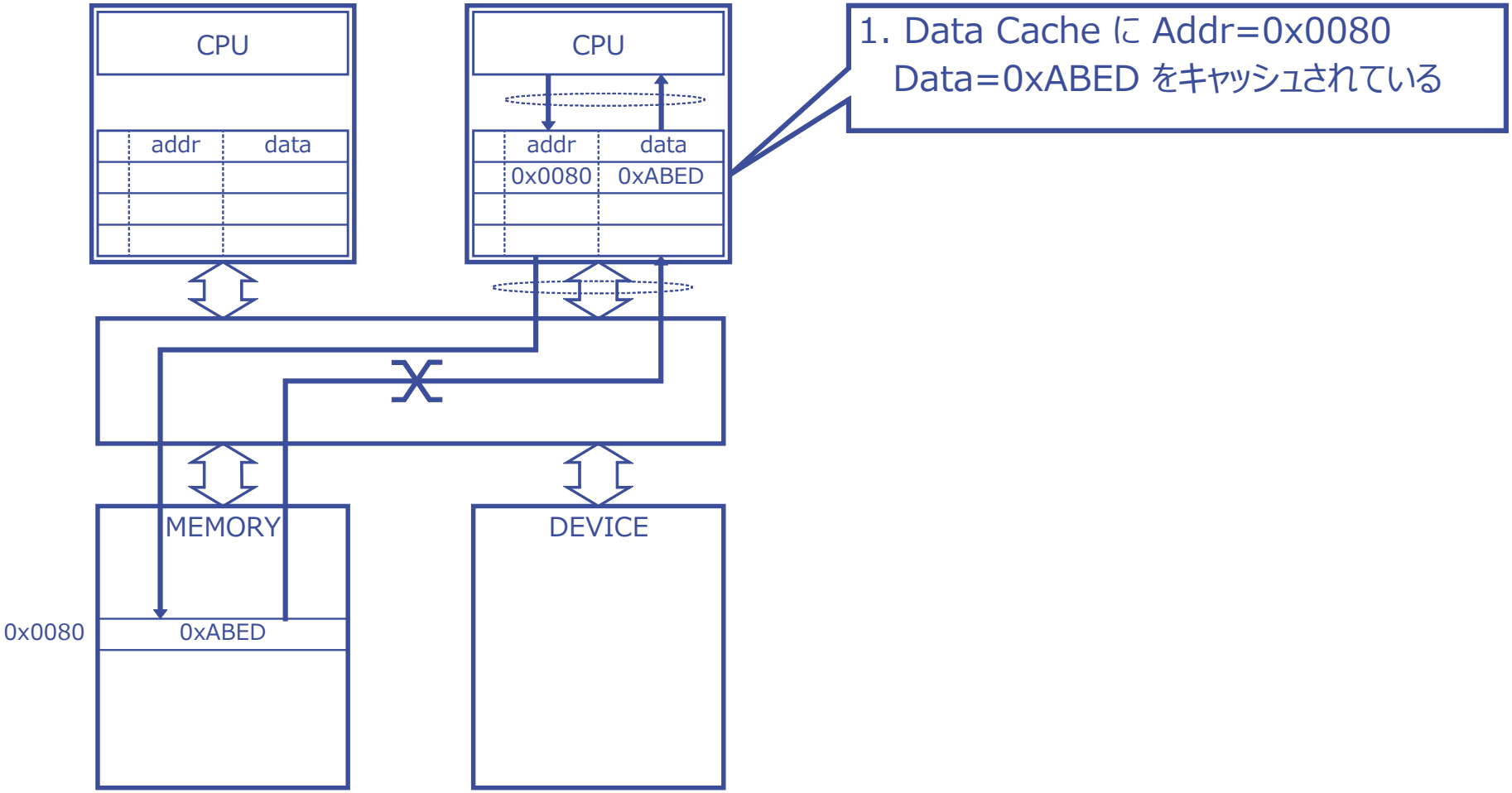
2. Data Cache に Addr=0080 Data=0xABED をキャッシュ

3. Data Cache に対して Addr=0080 をソフトウェアで強制的に Flush を命じる

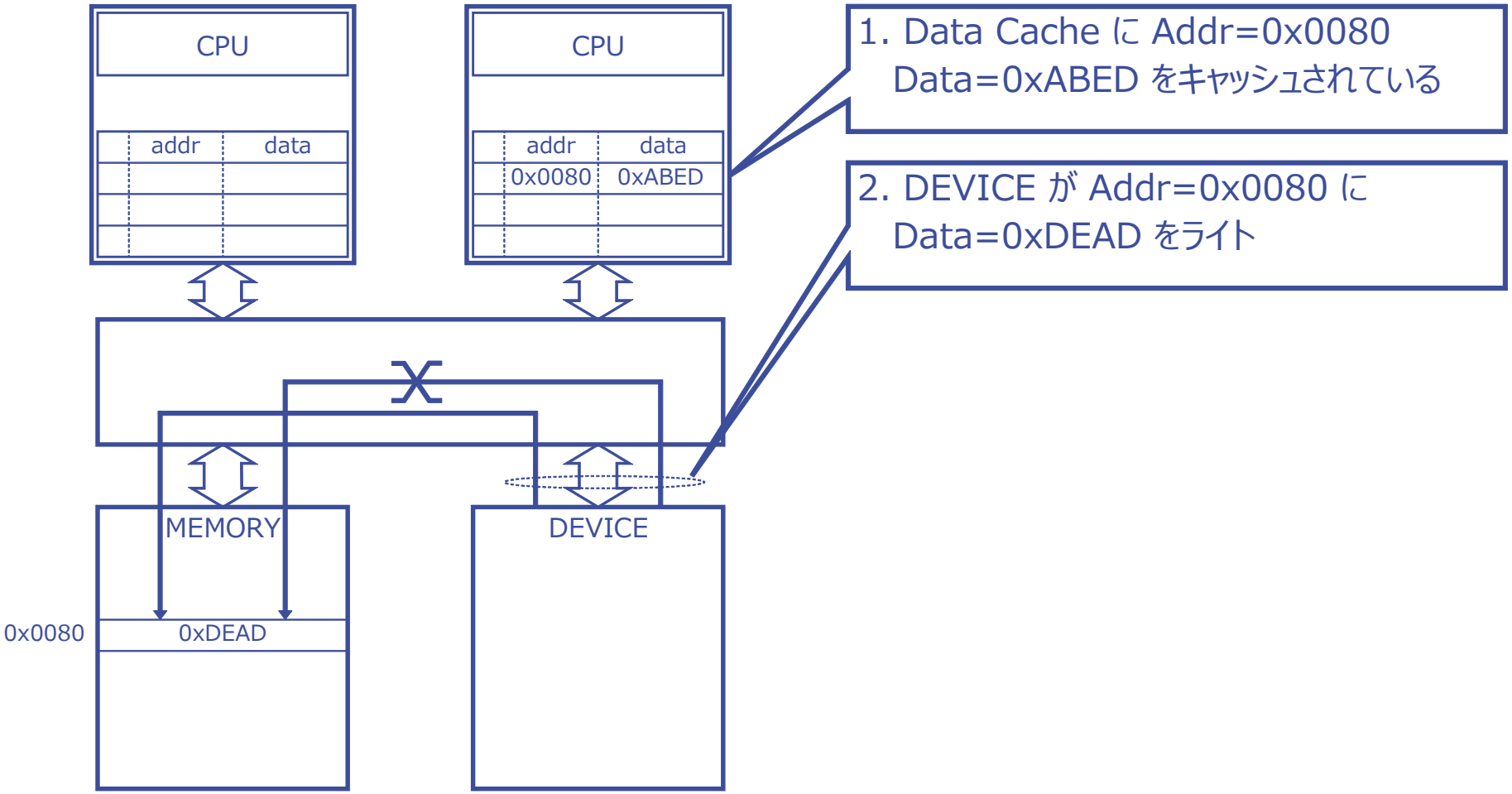
4. DEVICE が Addr=0x0080 をリード

DEVICE は正しい値 (0xABED)が読める

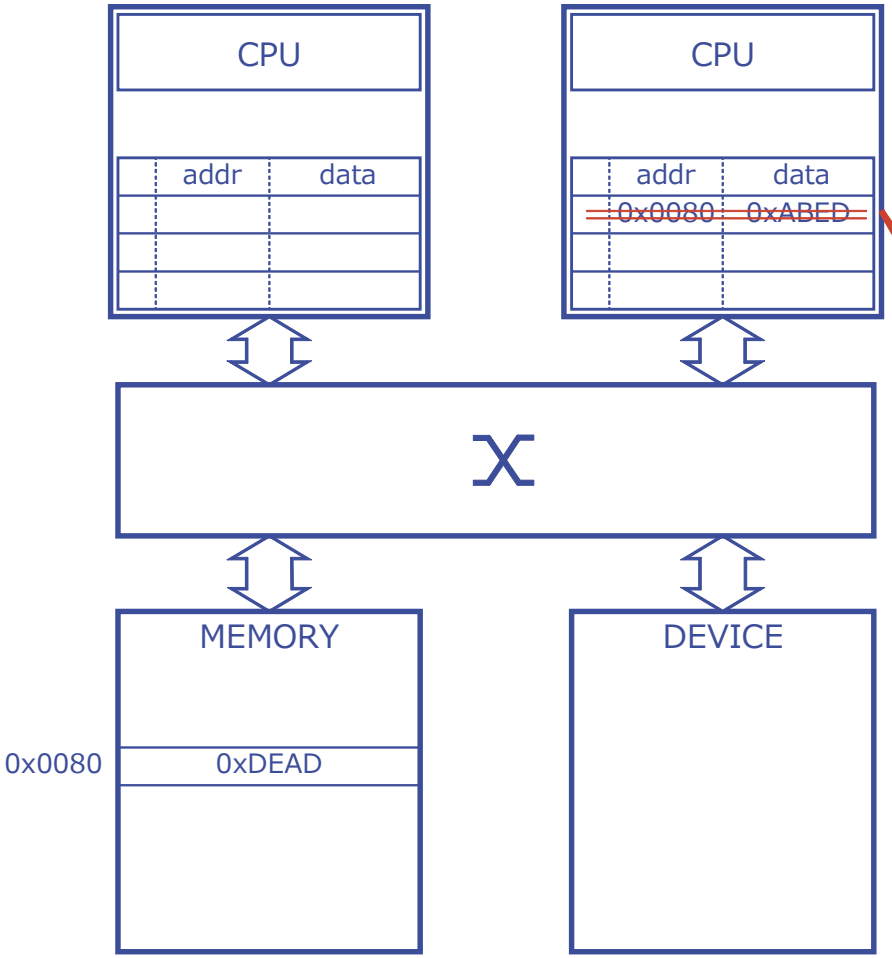
ソフトウェアでなんとかする方法　-2-



ソフトウェアでなんとかする方法　-2-



ソフトウェアでなんとかする方法　-2-

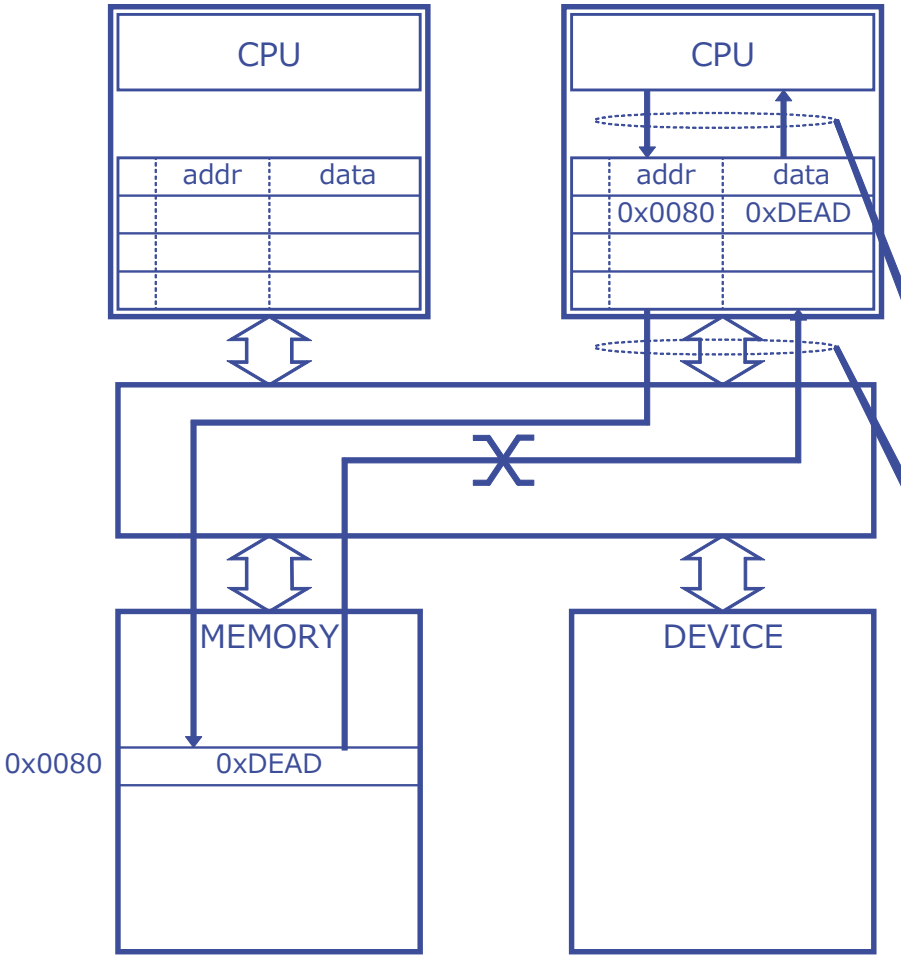


1. Data Cache に Addr=0x0080 Data=0xABED をキャッシュされている

2. DEVICE が Addr=0x0080 に Data=0xDEAD をライト

3. Data Cache の Addr=0x0080 をソフトウェアで Invalidiate(無効化)

ソフトウェアでなんとかする方法 -2-



1. Data Cache に Addr=0x0080 Data=0xABED をキャッシュされている

2. DEVICE が Addr=0x0080 に Data=0xDEAD をライト

3. Data Cache の Addr=0x0080 をソフトウェアで Invalidiate(無効化)

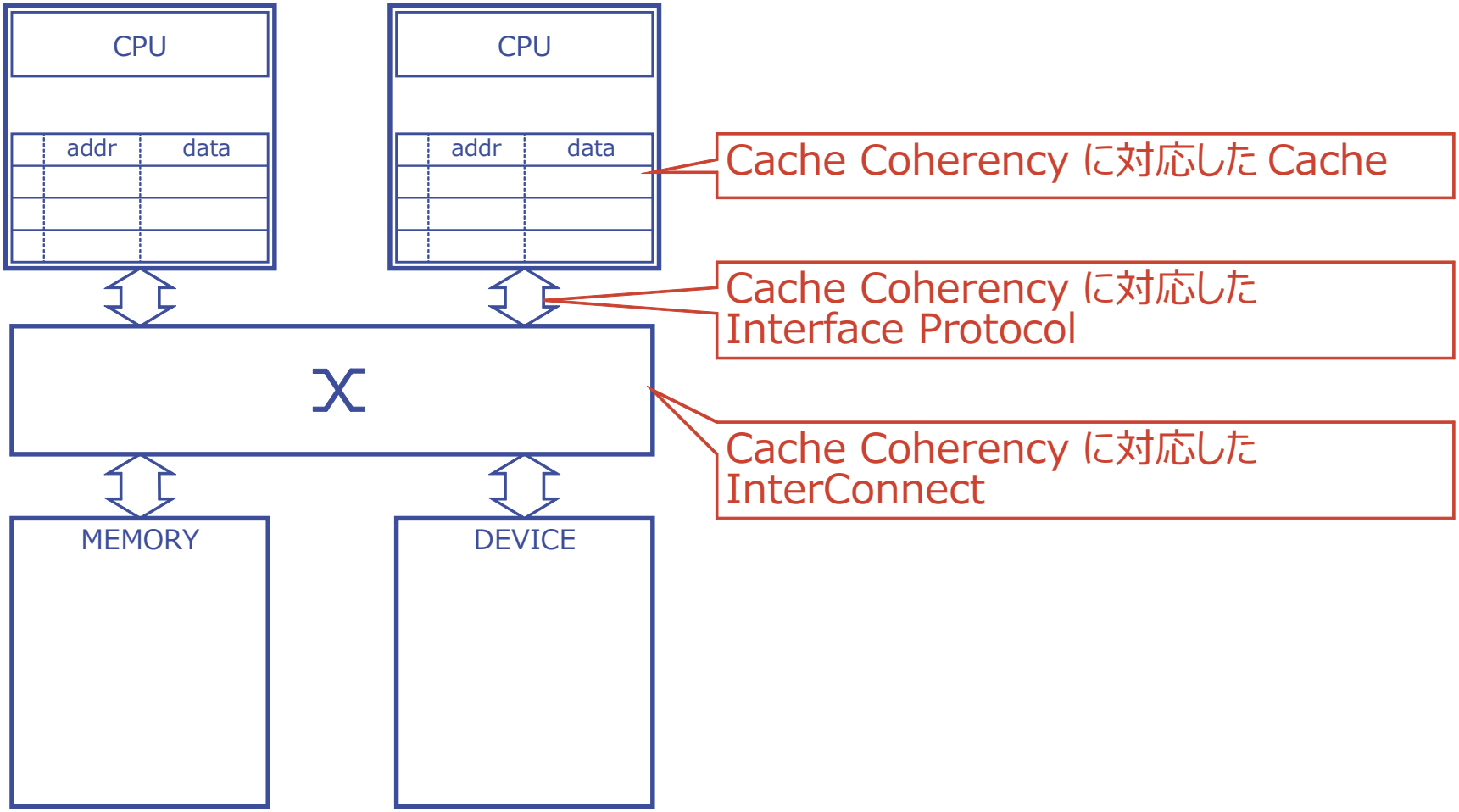
4. CPU が Addr=0x0080 をリード

5. Cache に無いので MEMORY からリード

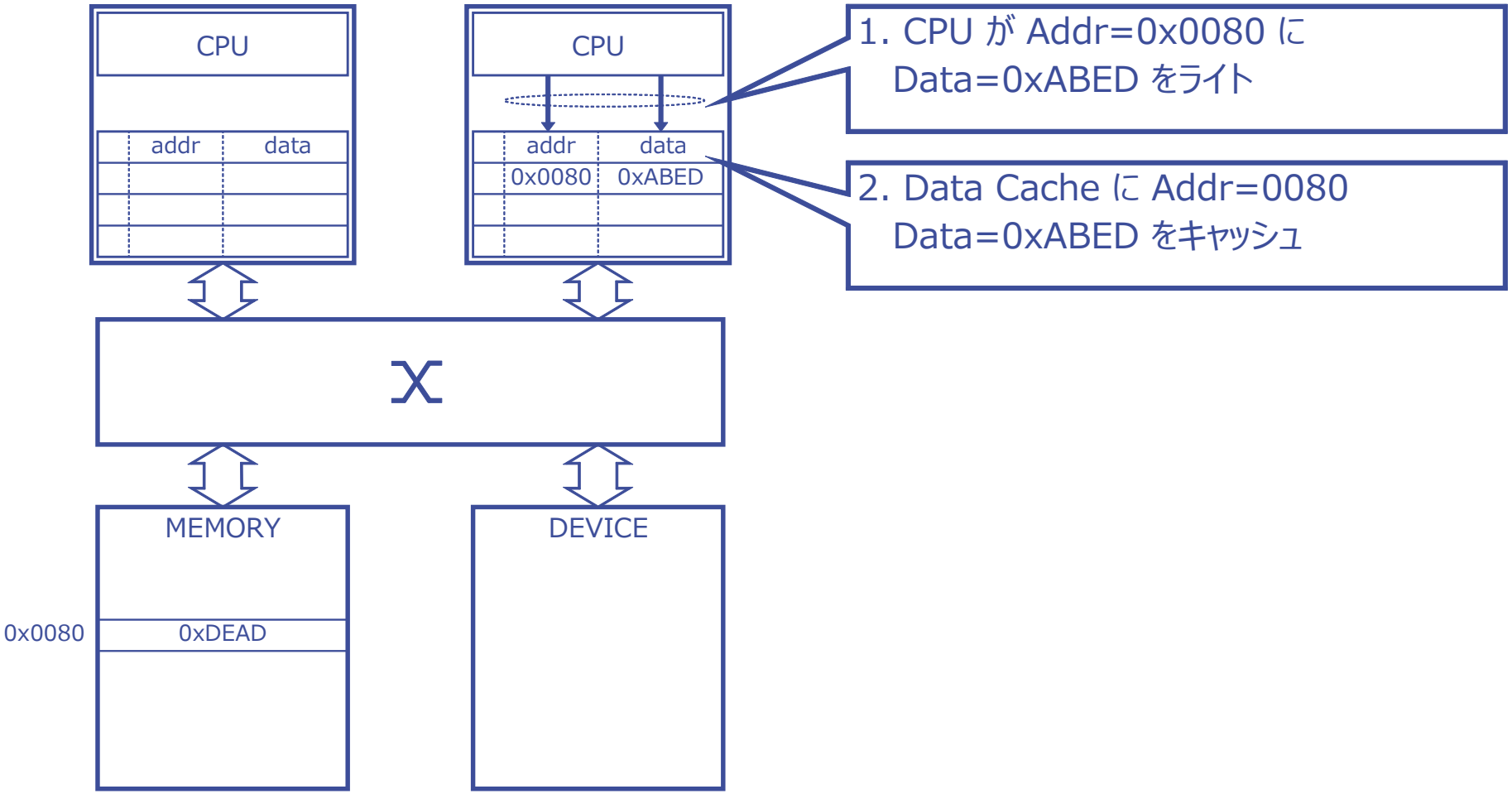
ソフトウェアでなんとかする方法の特徴

- ソフトウェアの実装負担: 重い
  - CPU からのアクセスとデバイスからのアクセスが排他的
  - デバイスのリード/ライトするタイミングが判ってないと無理
- ハードウェアの実装負担: 軽い
- CPU からのアクセス速度: 速い
  - 注意点: キャッシュの操作は意外と時間がかかる

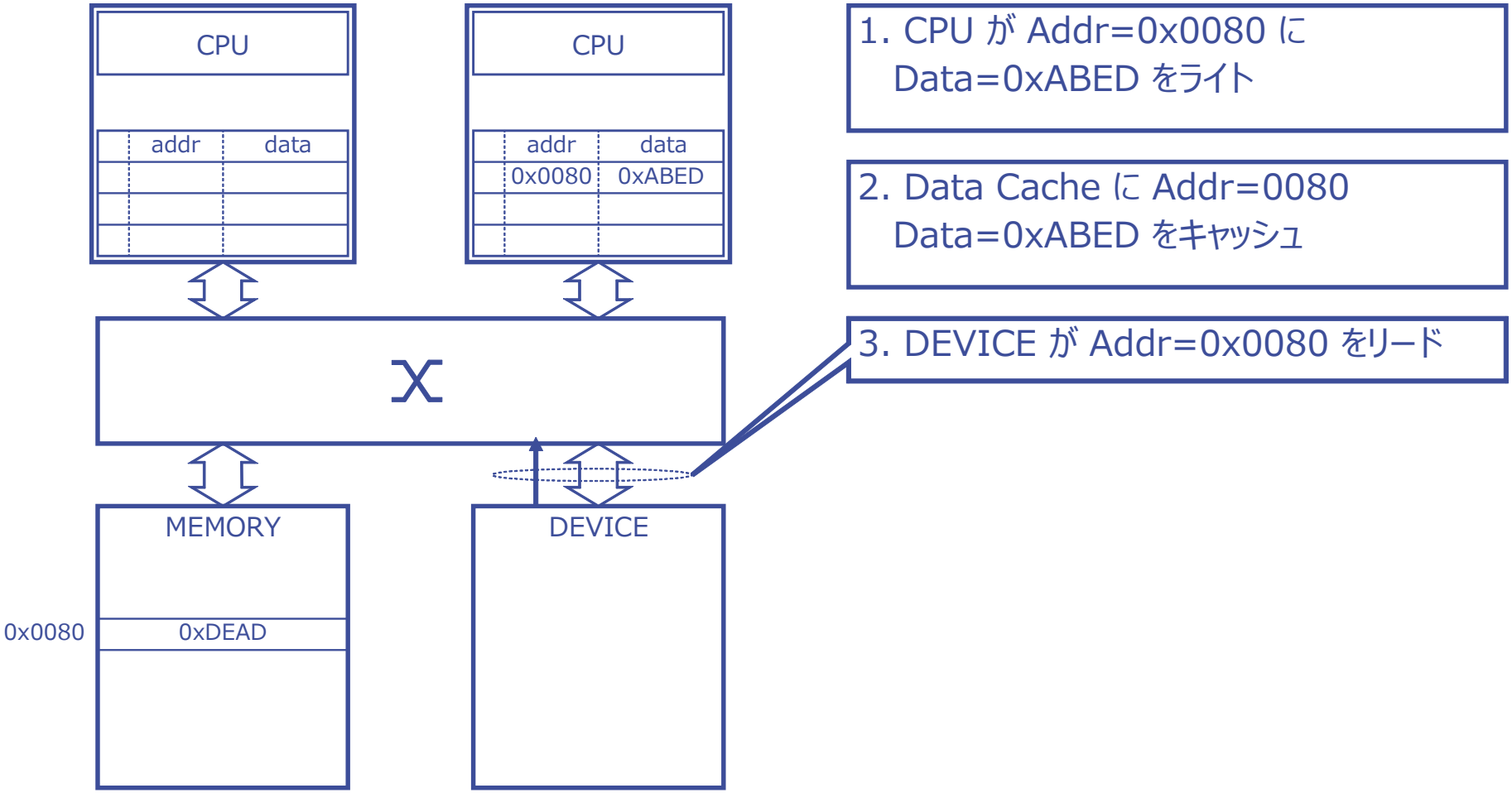
ハードウェアでなんとかする方法　- 用意するもの



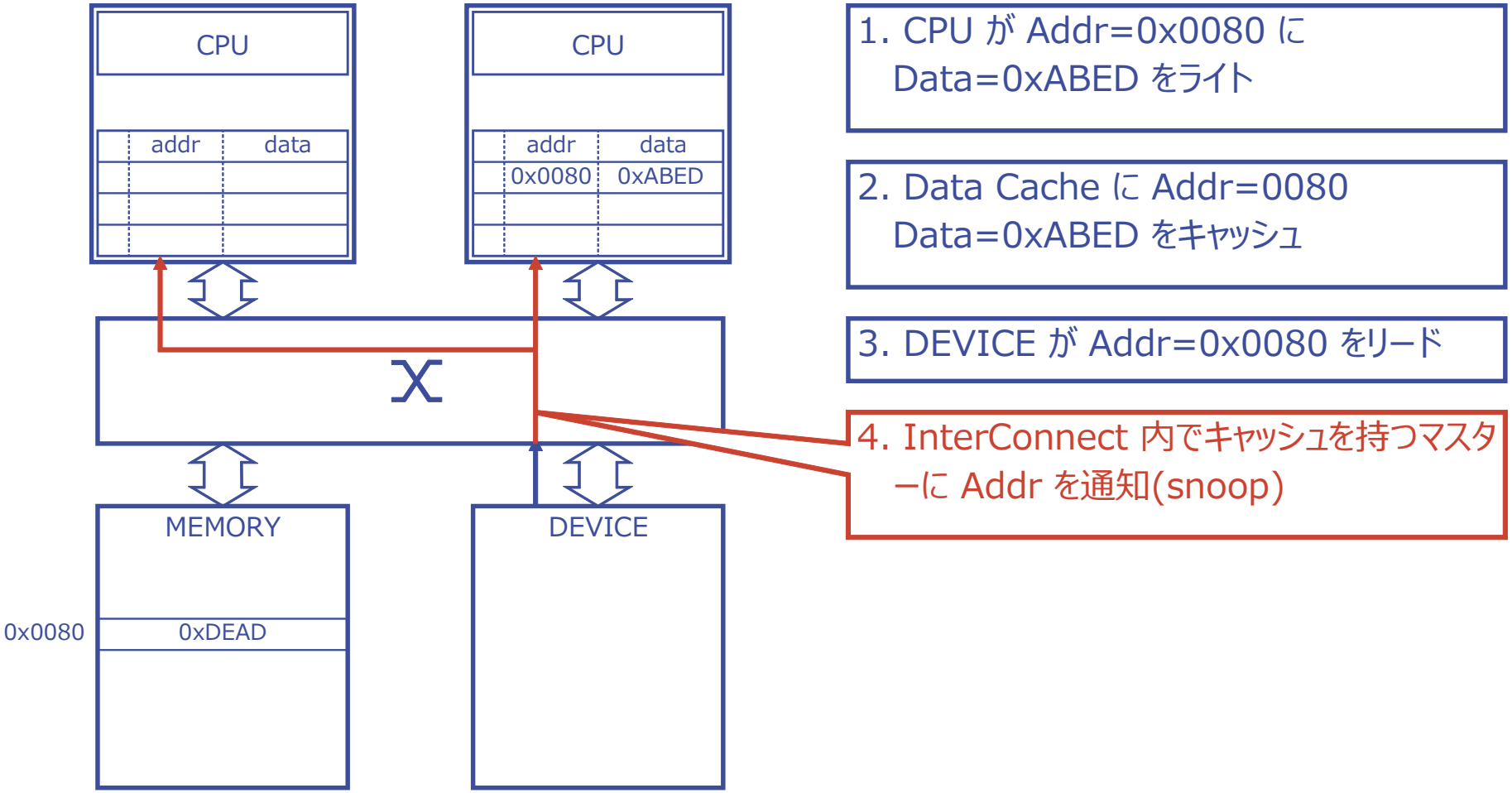
ハードウェアでなんとかする方法 - 1 -



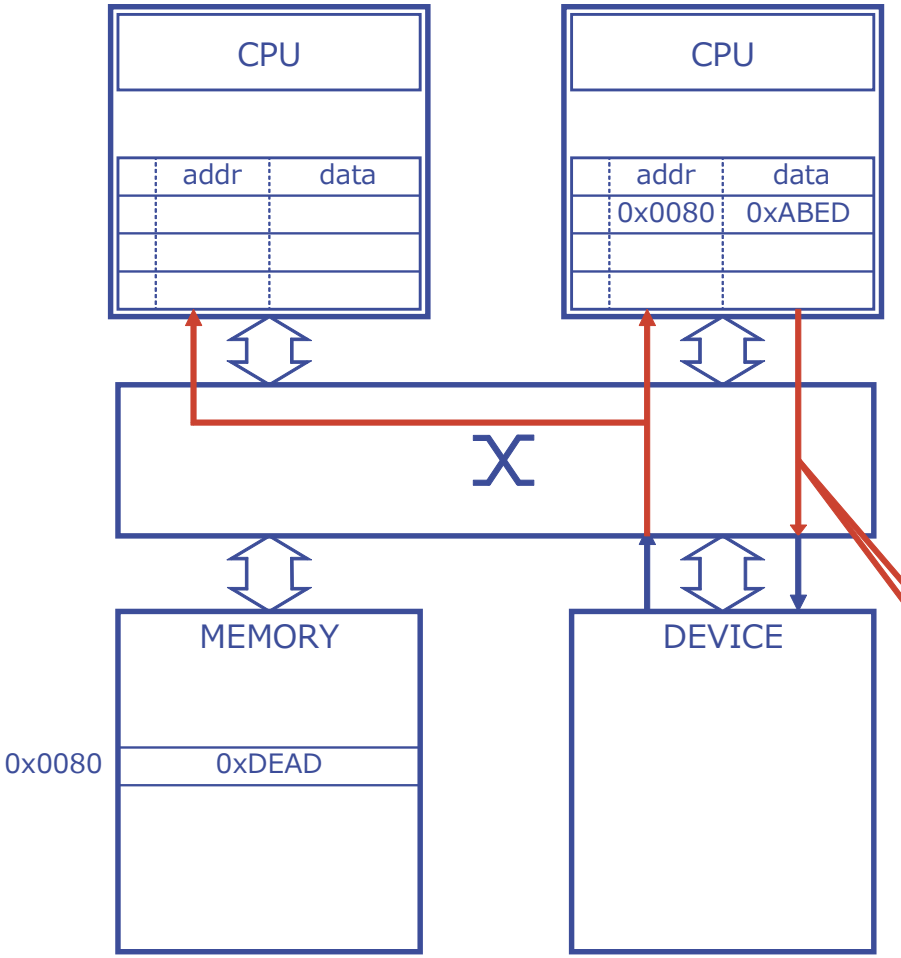
ハードウェアでなんとかする方法　- 1 -



ハードウェアでなんとかする方法 - 1 -



ハードウェアでなんとかする方法 - 1 -



1. CPU が Addr=0x0080 に Data=0xABED をライト

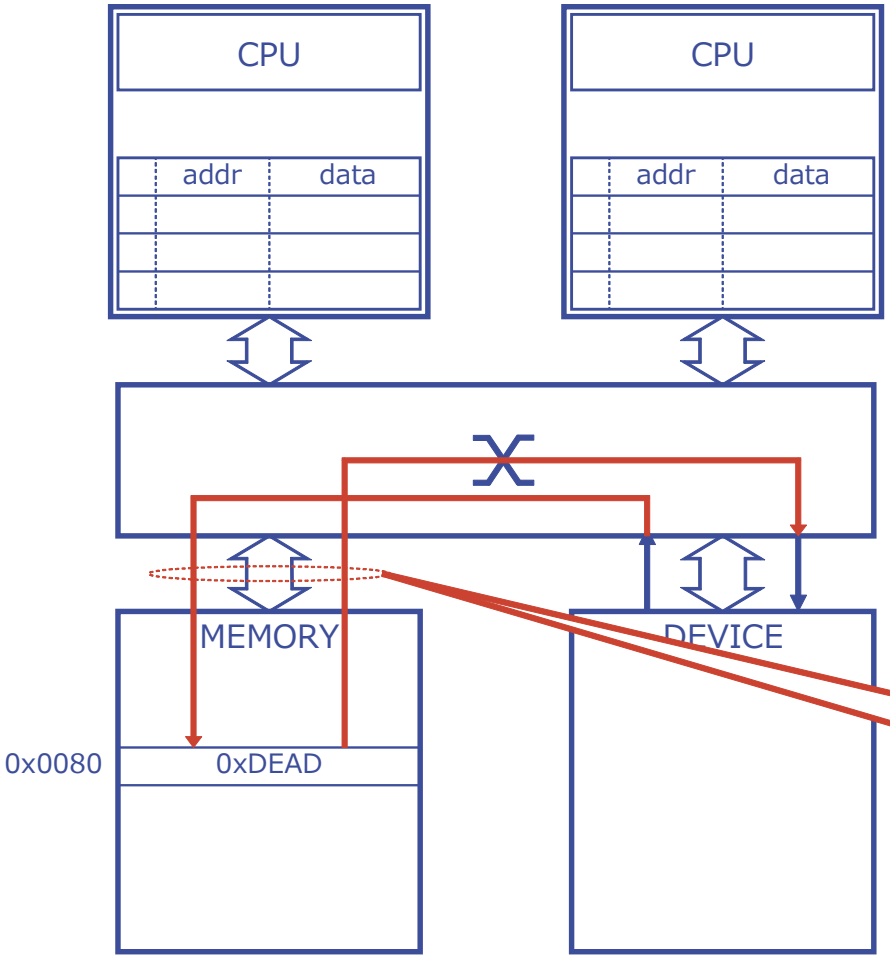
2. Data Cache に Addr=0080 Data=0xABED をキャッシュ

3. DEVICE が Addr=0x0080 をリード

4. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)

5. 該当する Addr(0x0080)のデータをキャッシュしているマスターがいる場合は、マスターのデータをもらう

ハードウェアでなんとかする方法 - 1 -

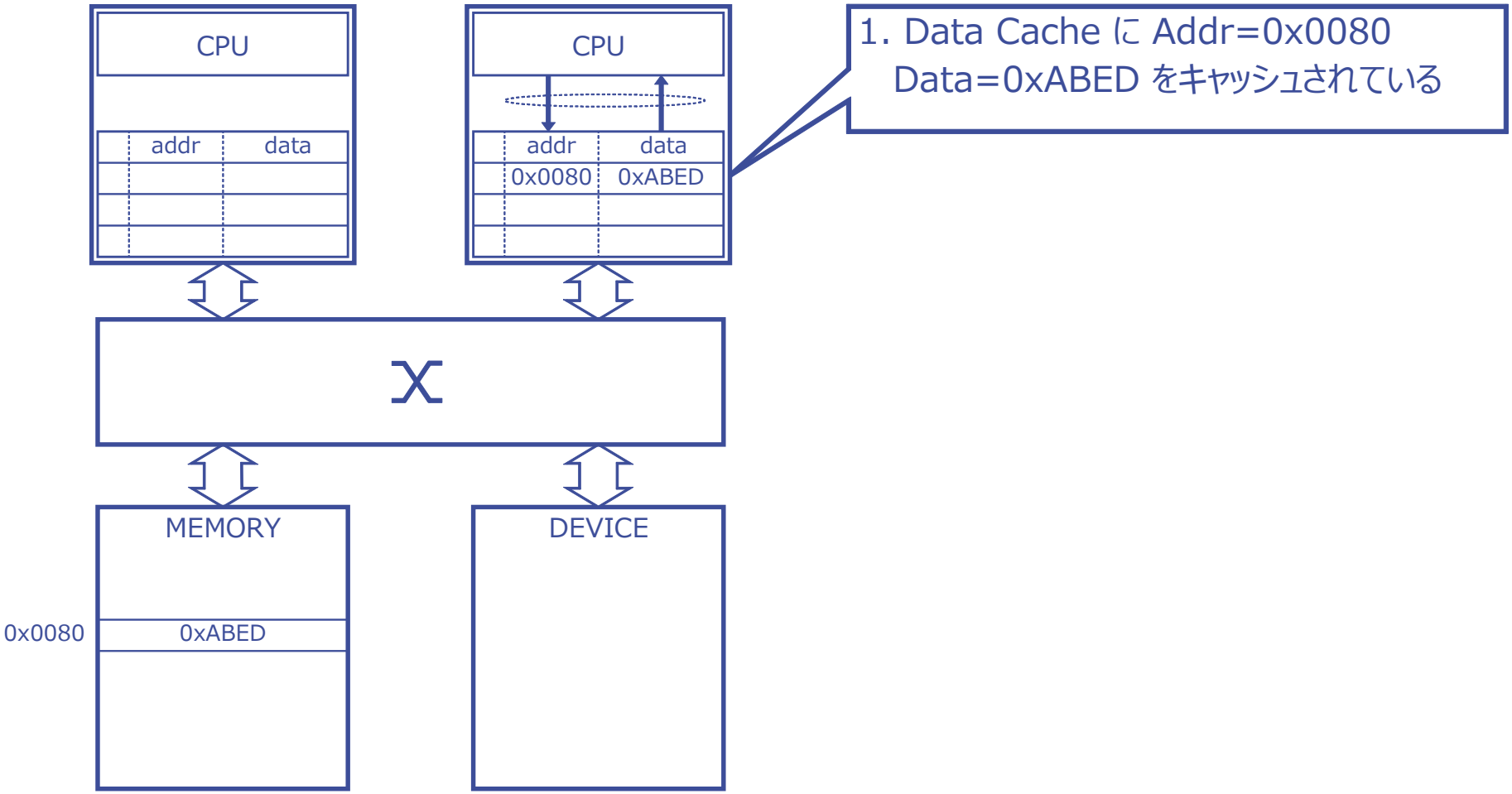


3. DEVICE が Addr=0x0080 をリード

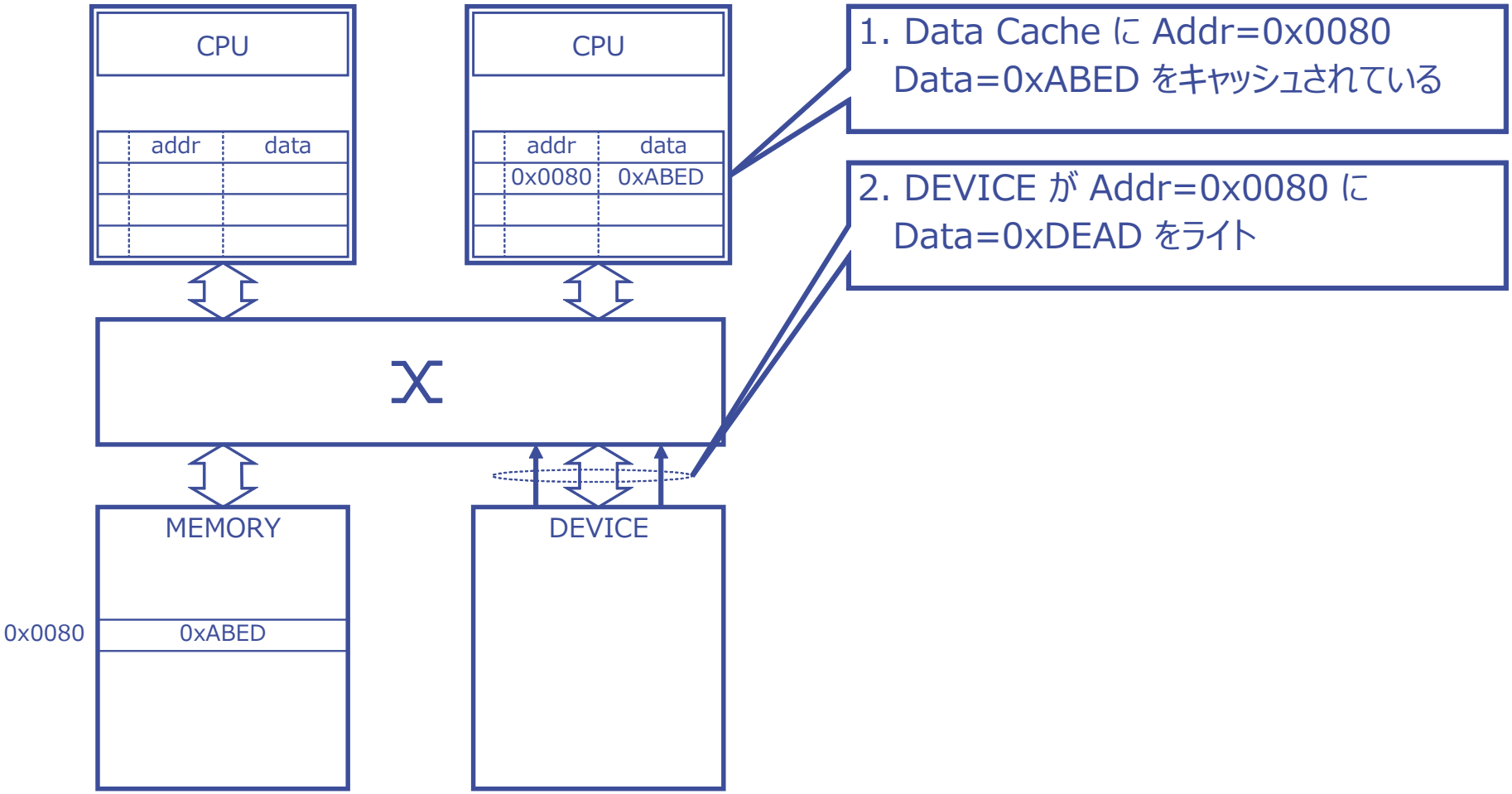
4. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)

5. 該当する Addr(0x0080)のデータをキャッシュしているマスターがいない場合は、MEMORY から読む

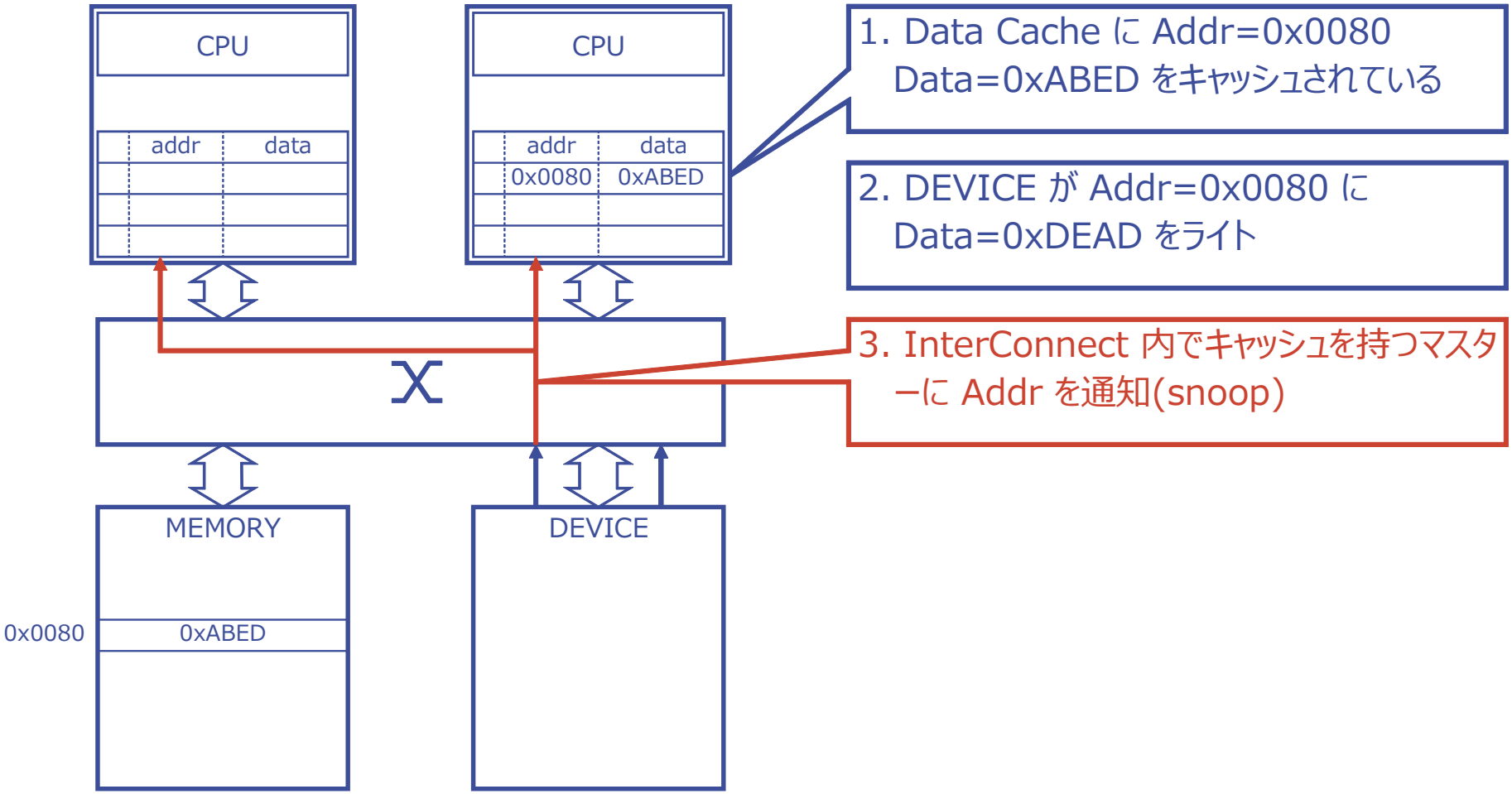
ハードウェアでなんとかする方法 -2-



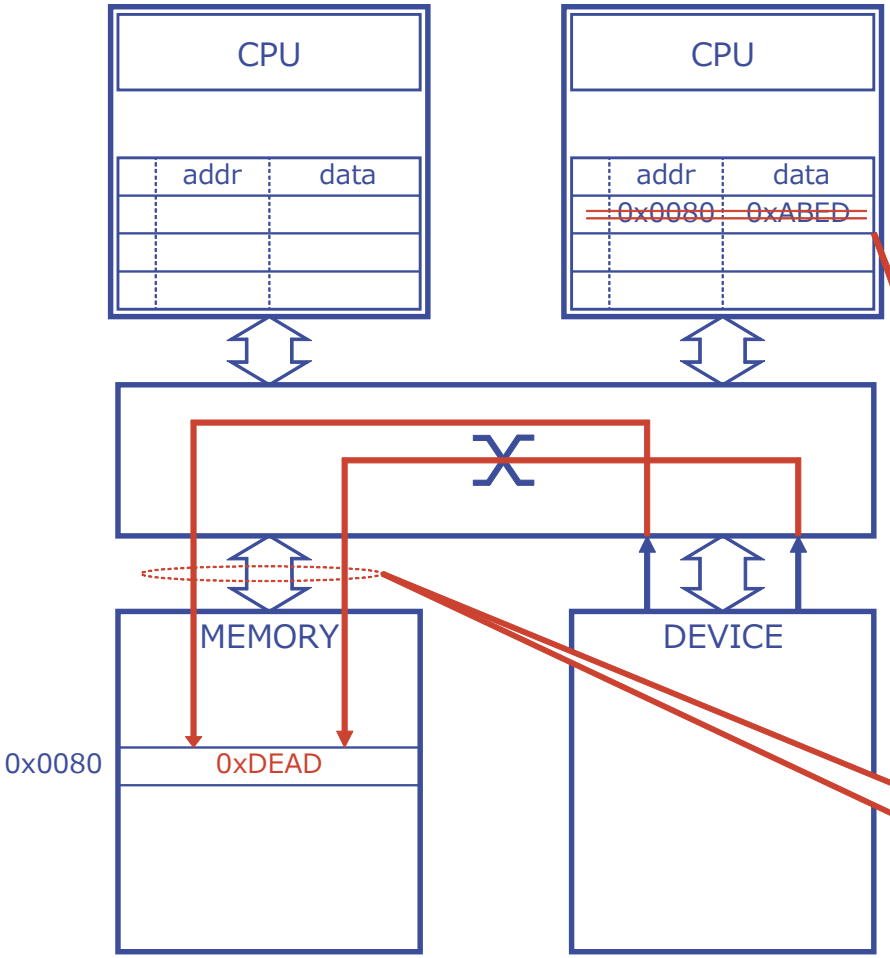
ハードウェアでなんとかする方法 -2-



ハードウェアでなんとかする方法　-2-



ハードウェアでなんとかする方法 -2-



1. Data Cache に Addr=0x0080 Data=0xABED をキャッシュされている

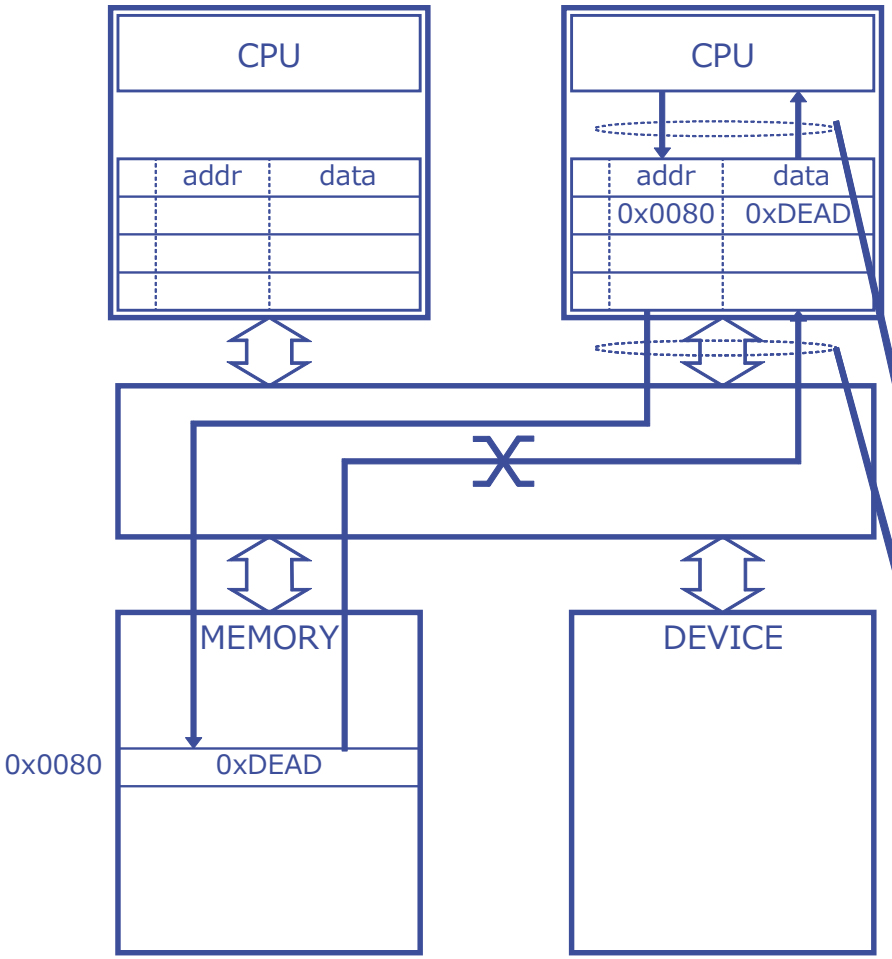
2. DEVICE が Addr=0x0080 に Data=0xDEAD をライト

3. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)

4. 該当する Addr(0x0080) のデータをキャッシュしているマスターはキャッシュを無効化 (Invalidiate)

5. MEMORY の Addr=0x0080 に Data=0xDEAD をライト

ハードウェアでなんとかする方法 -2-



1. Data Cache に Addr=0x0080 Data=0xABED をキャッシュされている
2. DEVICE が Addr=0x0080 に Data=0xDEAD をライト
3. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)
4. 該当する Addr(0x0080) のデータをキャッシュしているマスターはキャッシュを無効化 (Invalidiate)
5. MEMORY の Addr=0x0080 に Data=0xDEAD をライト
6. CPU が Addr=0x0080 をリード
7. Cache に無いので MEMORY からリード

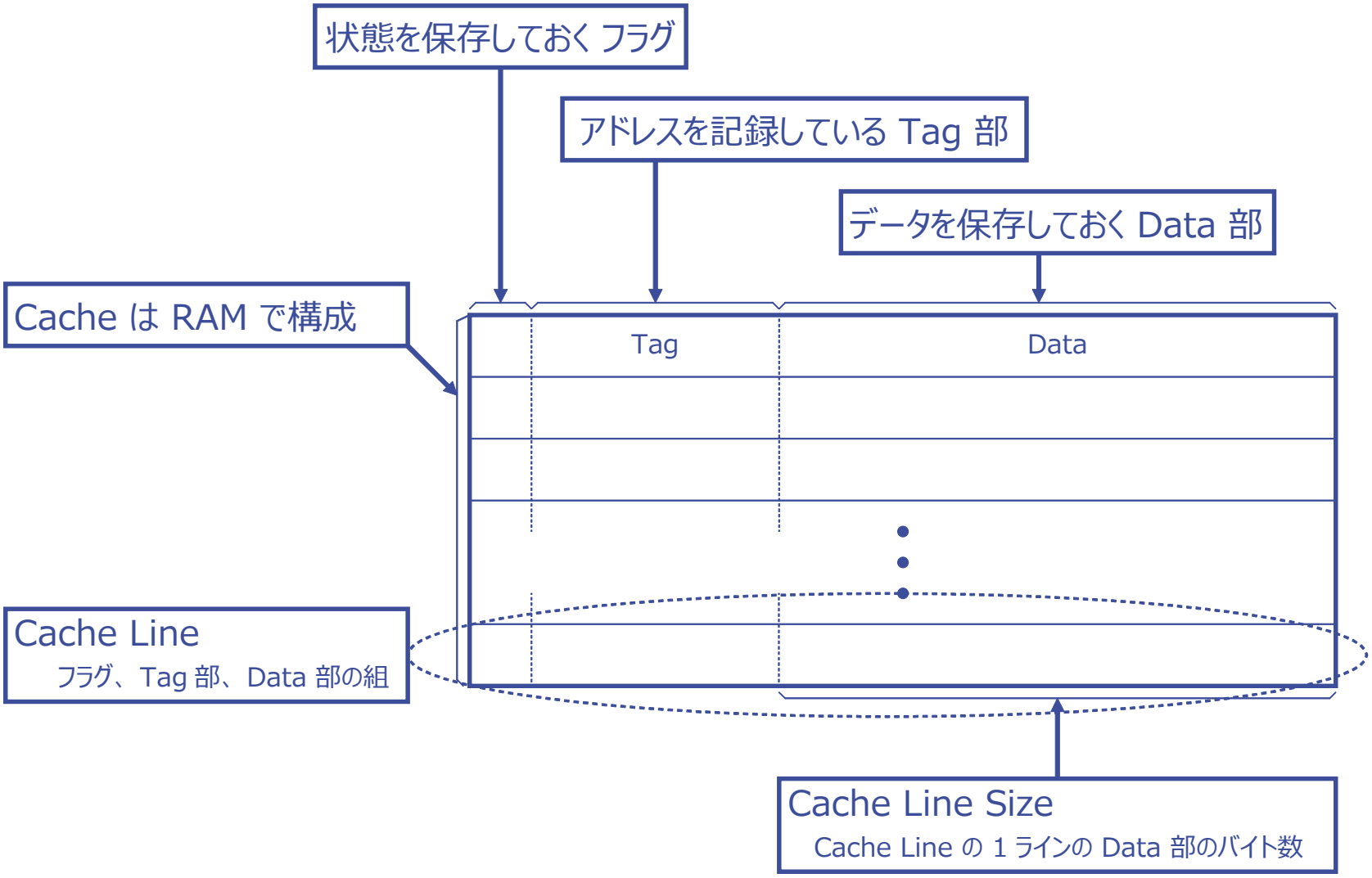
ハードウェアでなんとかする方法の特徴

- ソフトウェアの実装負担: 軽い
  - MMU の設定くらい
- ハードウェアの実装負担: 重い
  - Cache Coherency ハードウェアを 1 から実装するのは困難
  - Cache Coherency ハードウェアがすでに実装されている時は ありがた〜く使わせてもらってください
- CPU からのアクセス速度: 速い

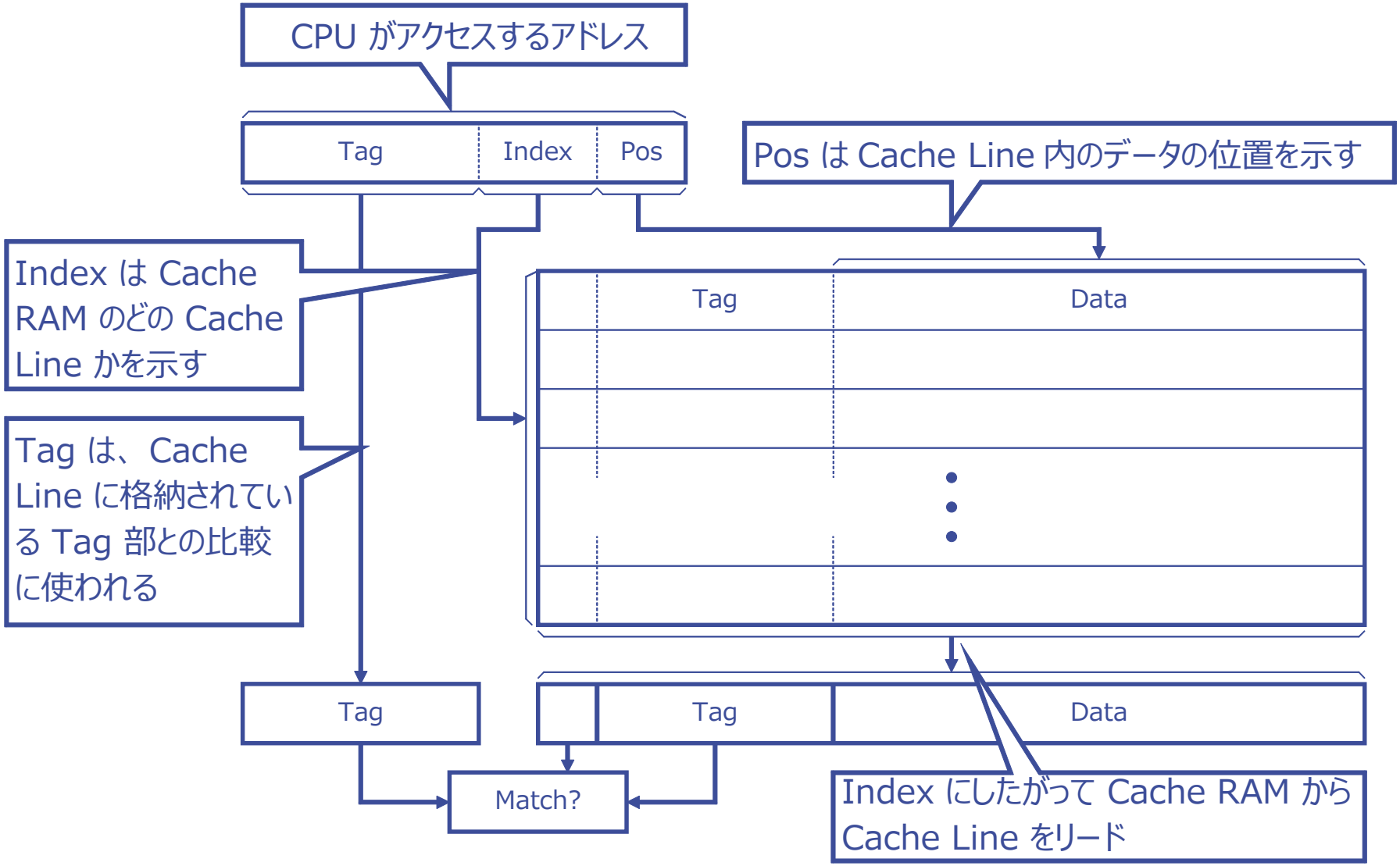
お品書き

- 背景
  - 主な登場人物の紹介
- Cache Coherency 問題
  - Cache Coherency 問題とは？
  - Cache Coherency 問題を解決する方法
- Cache Aliasing 問題
  - Cache Aliasing 問題とは？
  - Cache Aliasing 問題を解決する方法
- Linux で DMA バッファを mmap するとキャッシュが効かない仕組み
  - dev_is_dma_coherent() の役割
  - キャッシュが効かなくなる場合の例

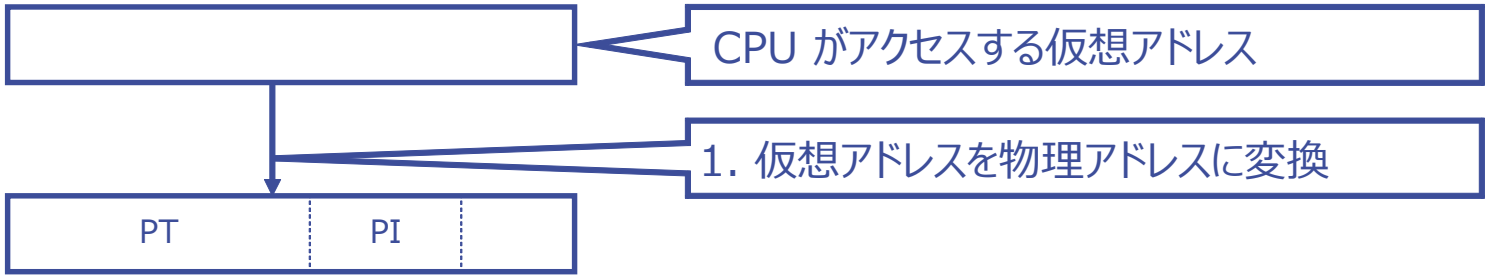
Cache の実装例 - Cache RAM の構成 -

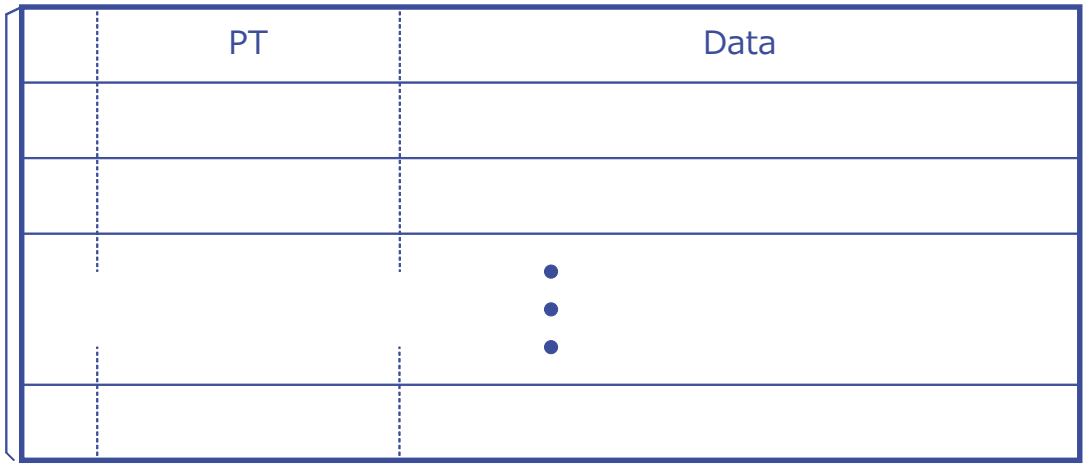


Cache の実装例 - Cache RAM へのアクセス -

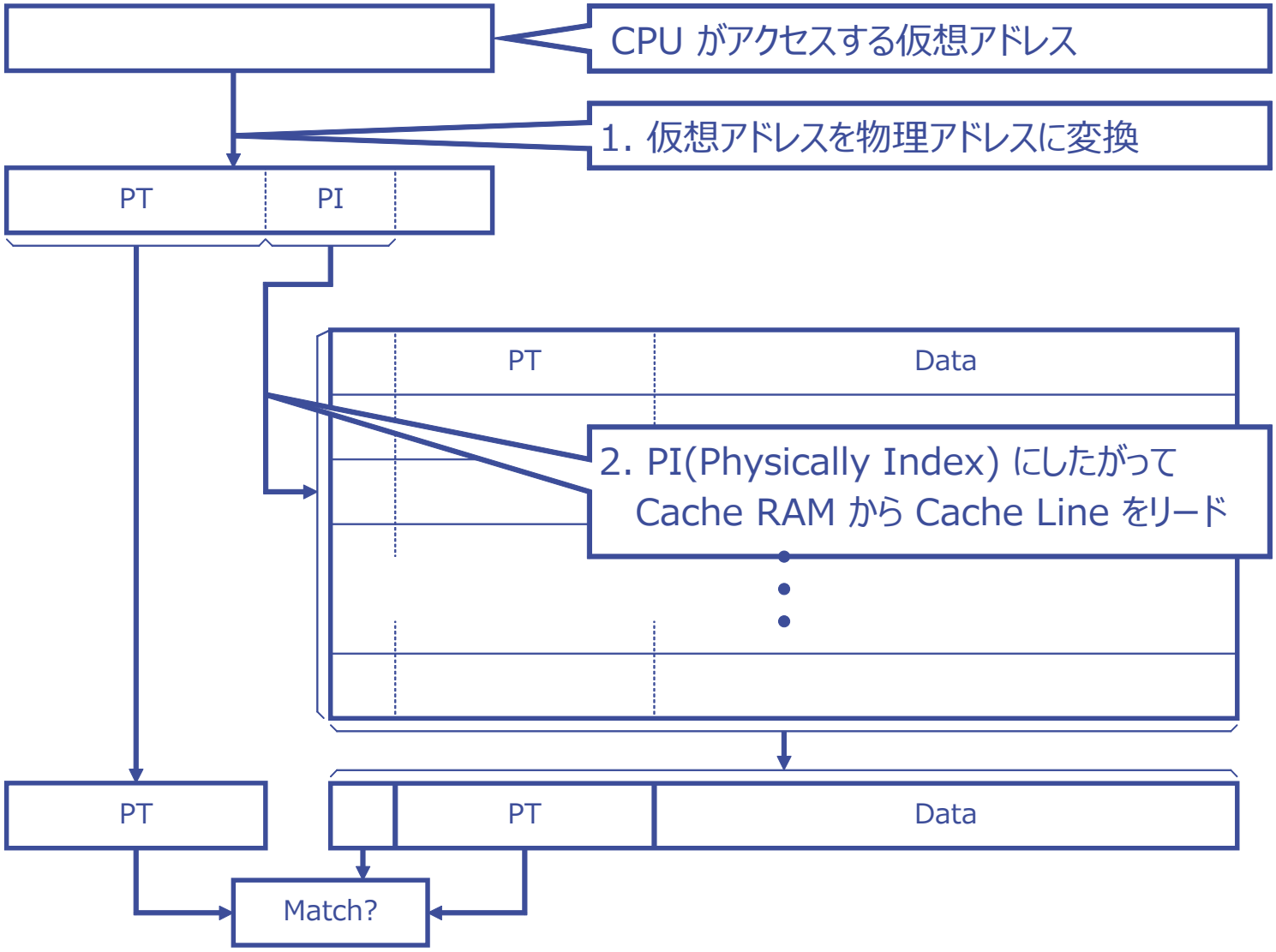


Cache の実装例 - PIPT(Physically Indexed Physically Tagged) -

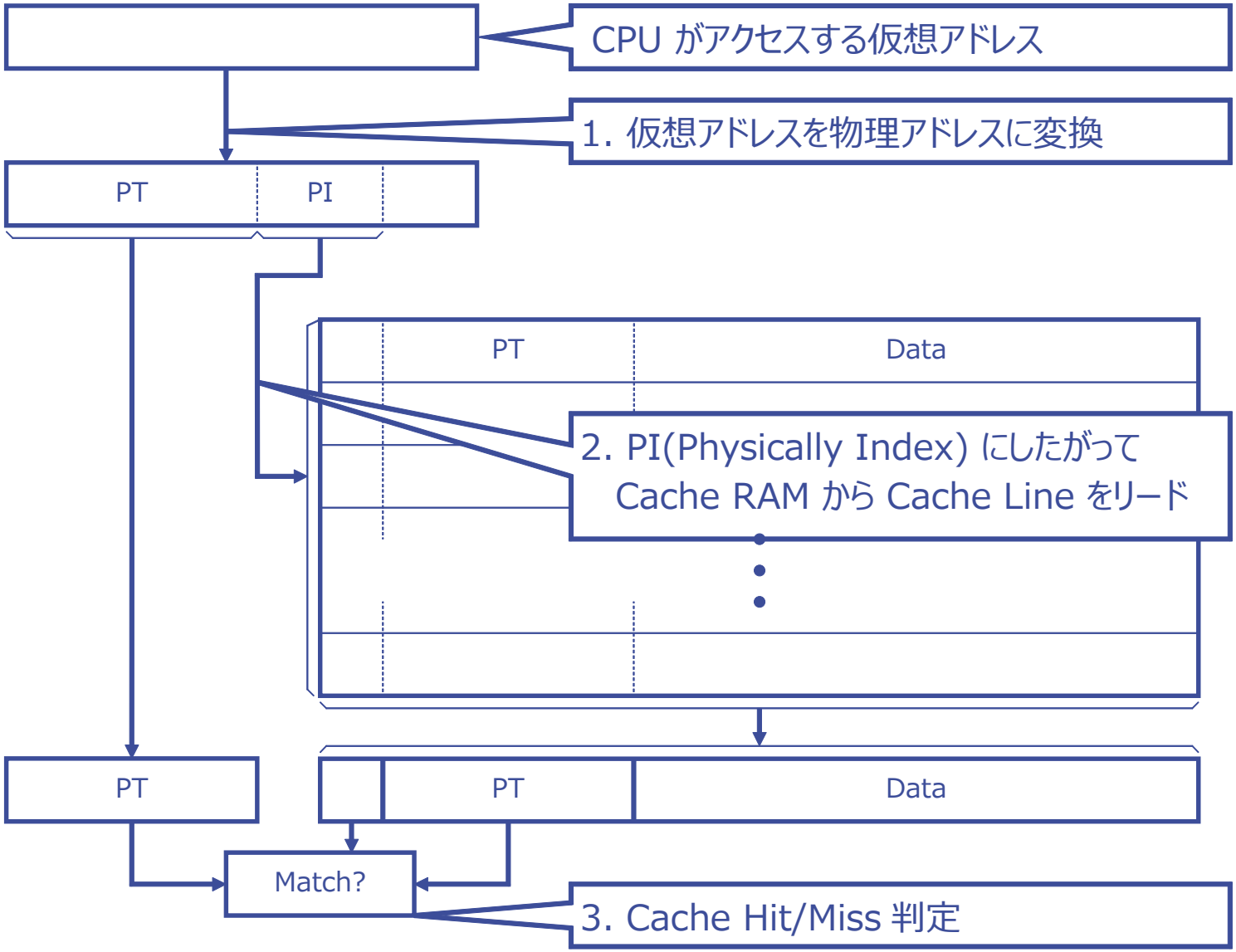




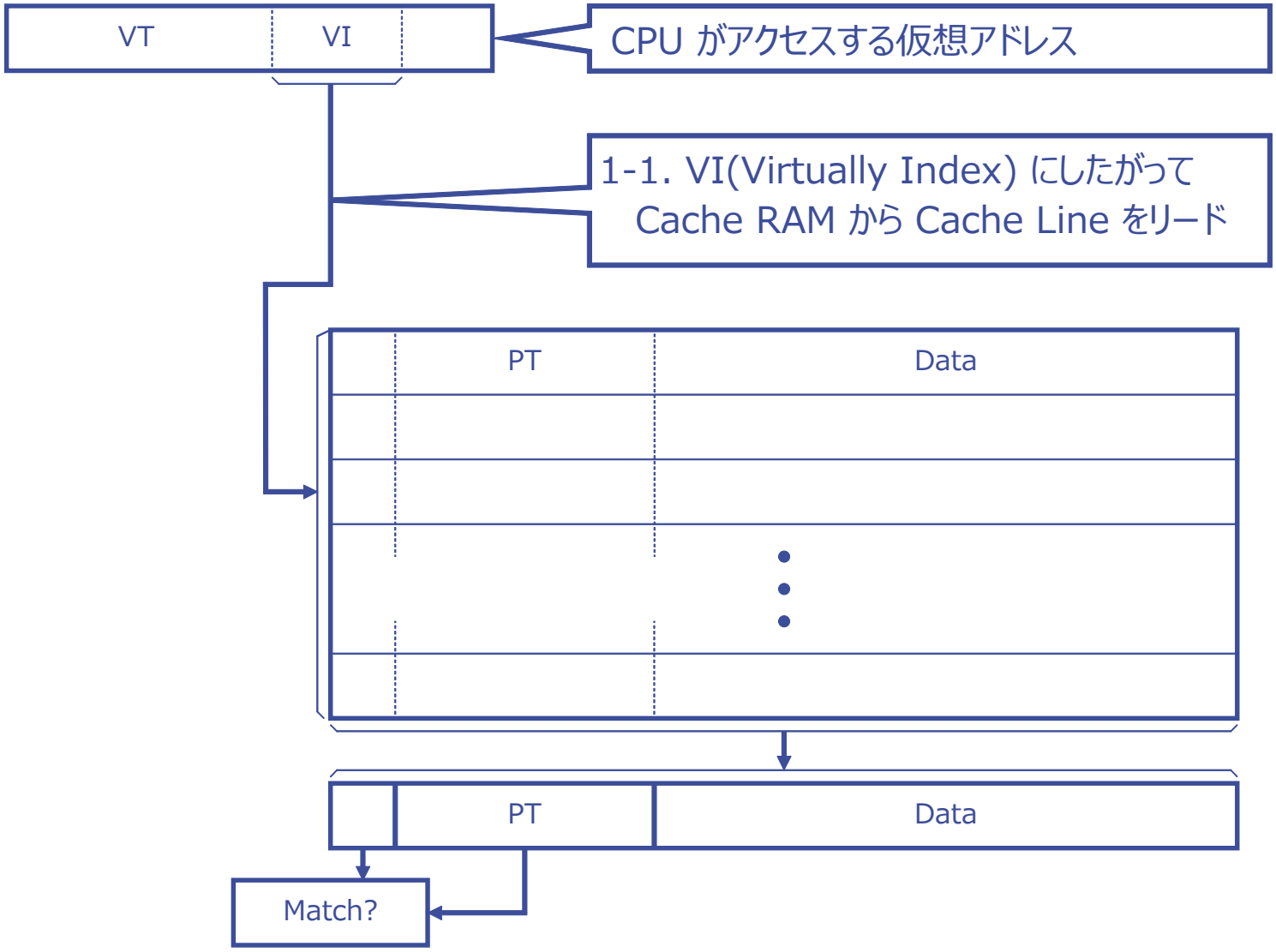
Cache の実装例 - PIPT(Physically Indexed Physically Tagged) -



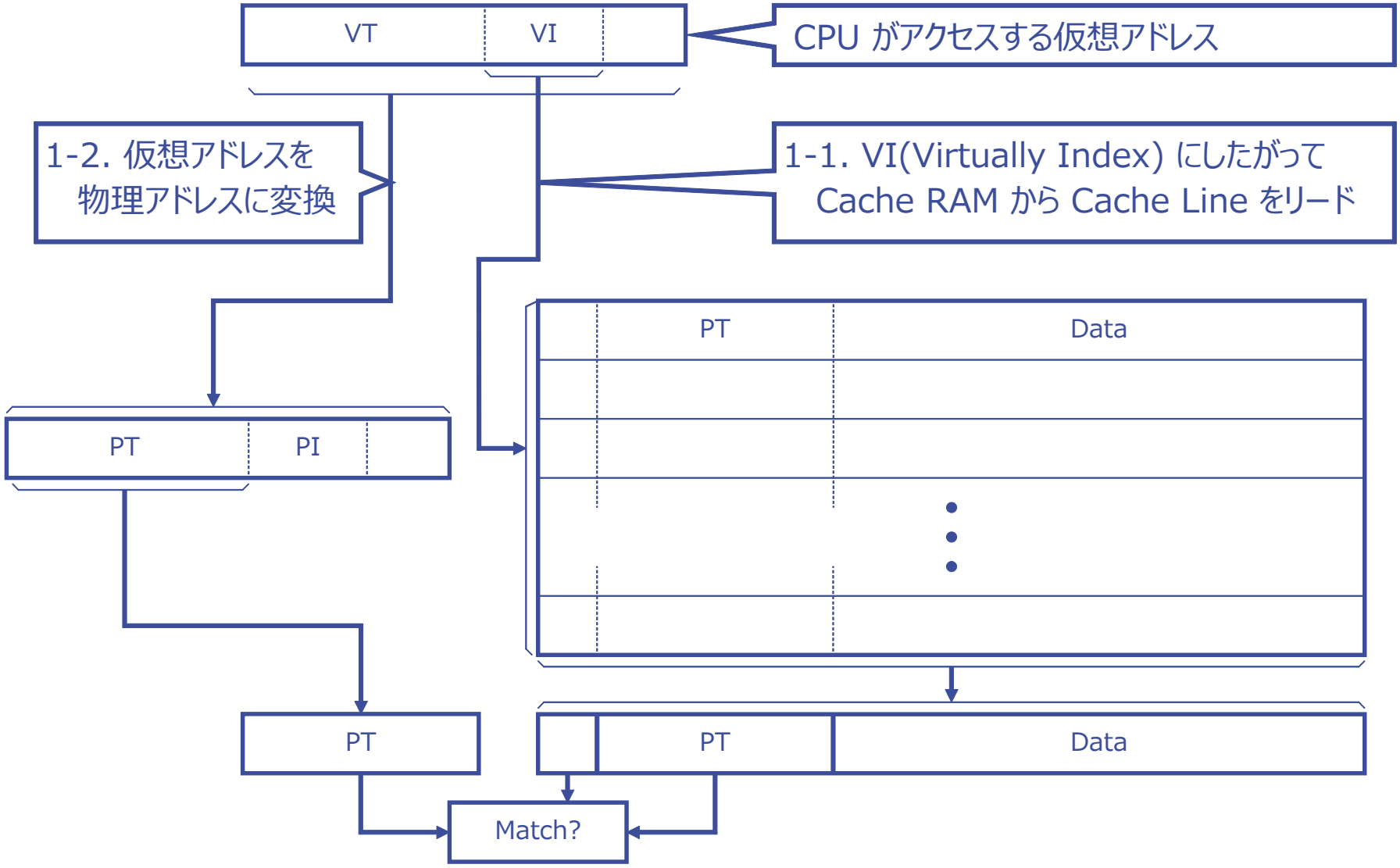
Cache の実装例 - PIPT(Physically Indexed Physically Tagged) -



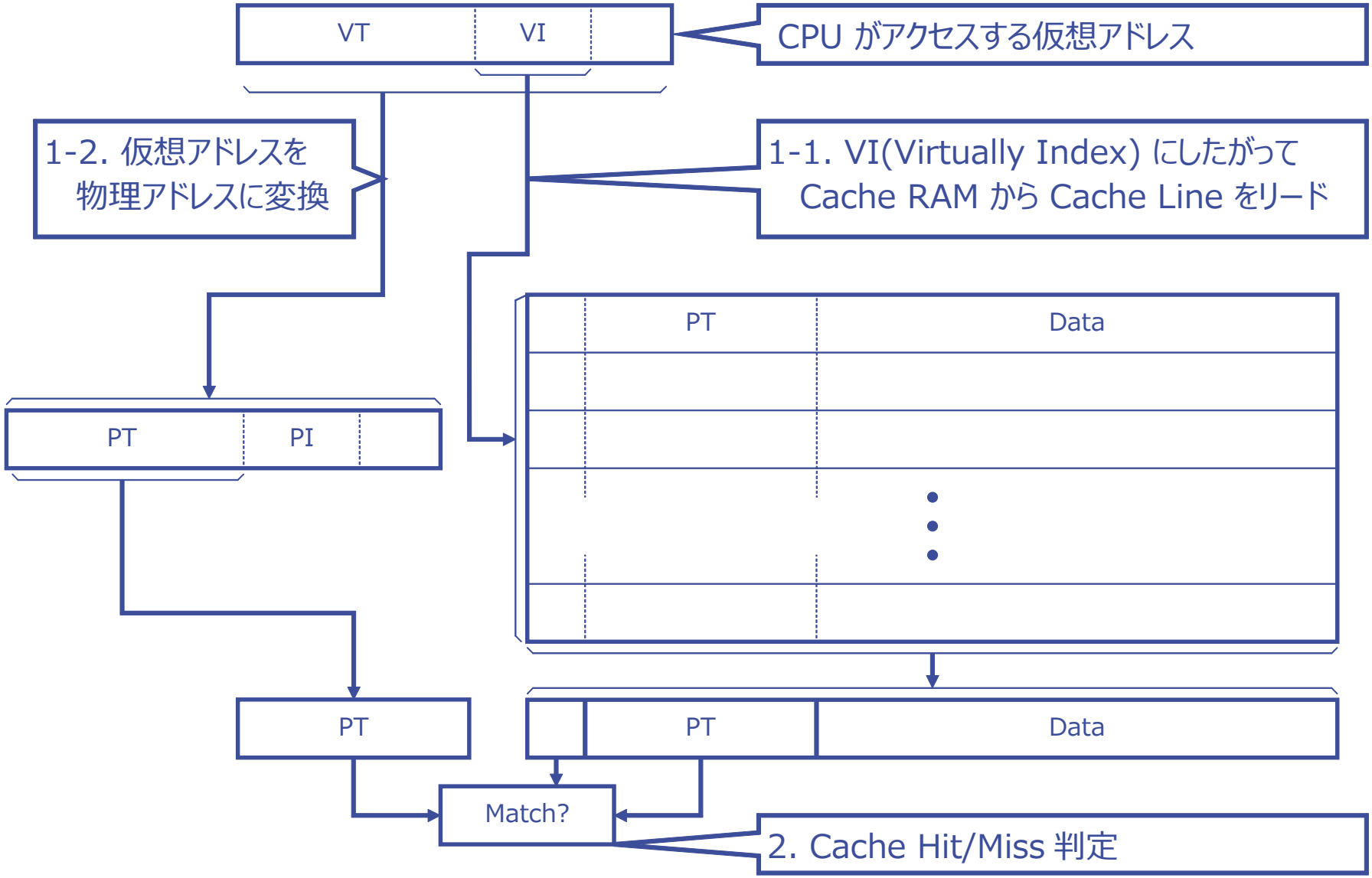
Cache の実装例 - VIPT(Virtually Indexed Physically Tagged) -



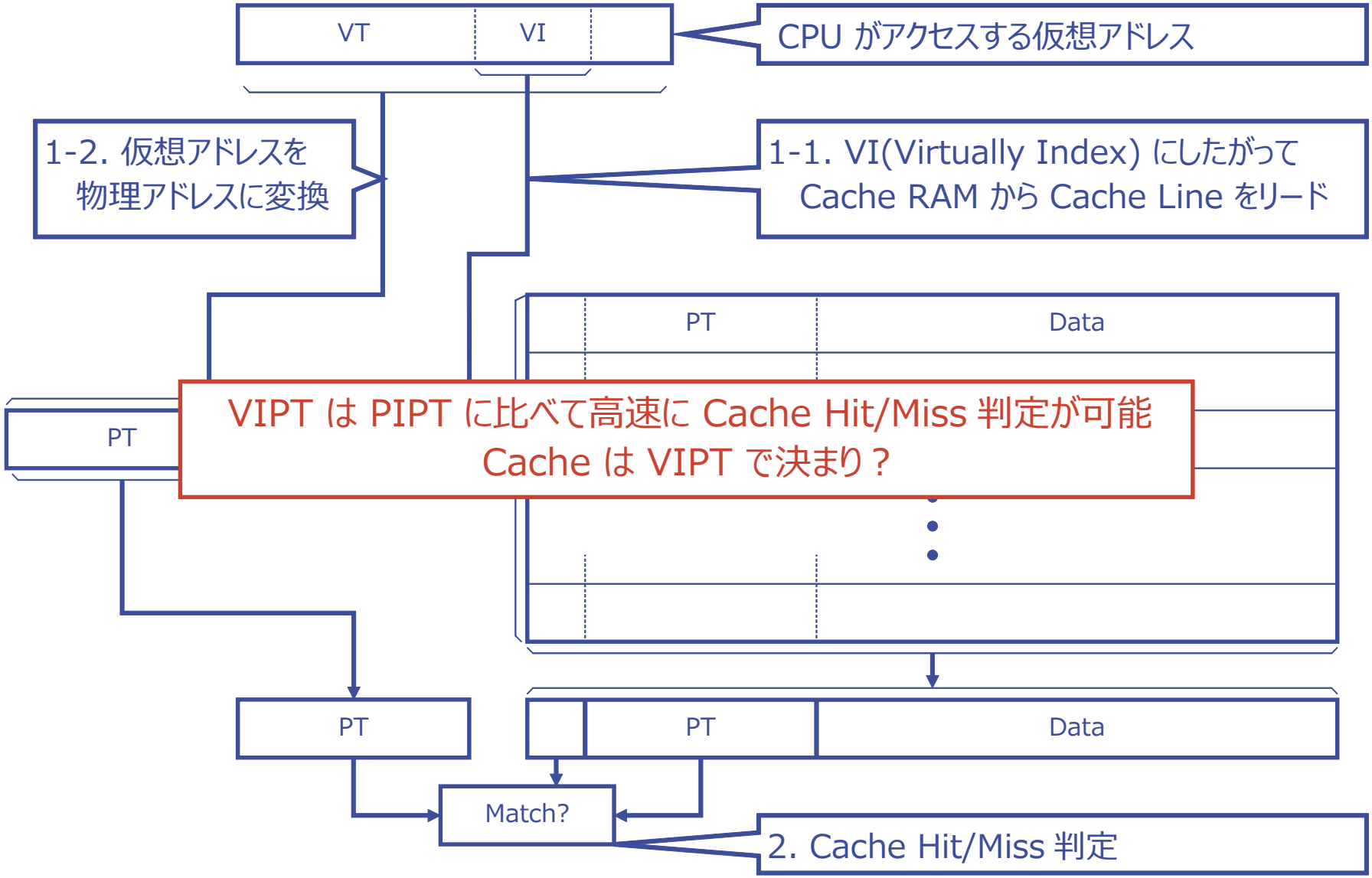
Cache の実装例 - VIPT(Virtually Indexed Physically Tagged) -



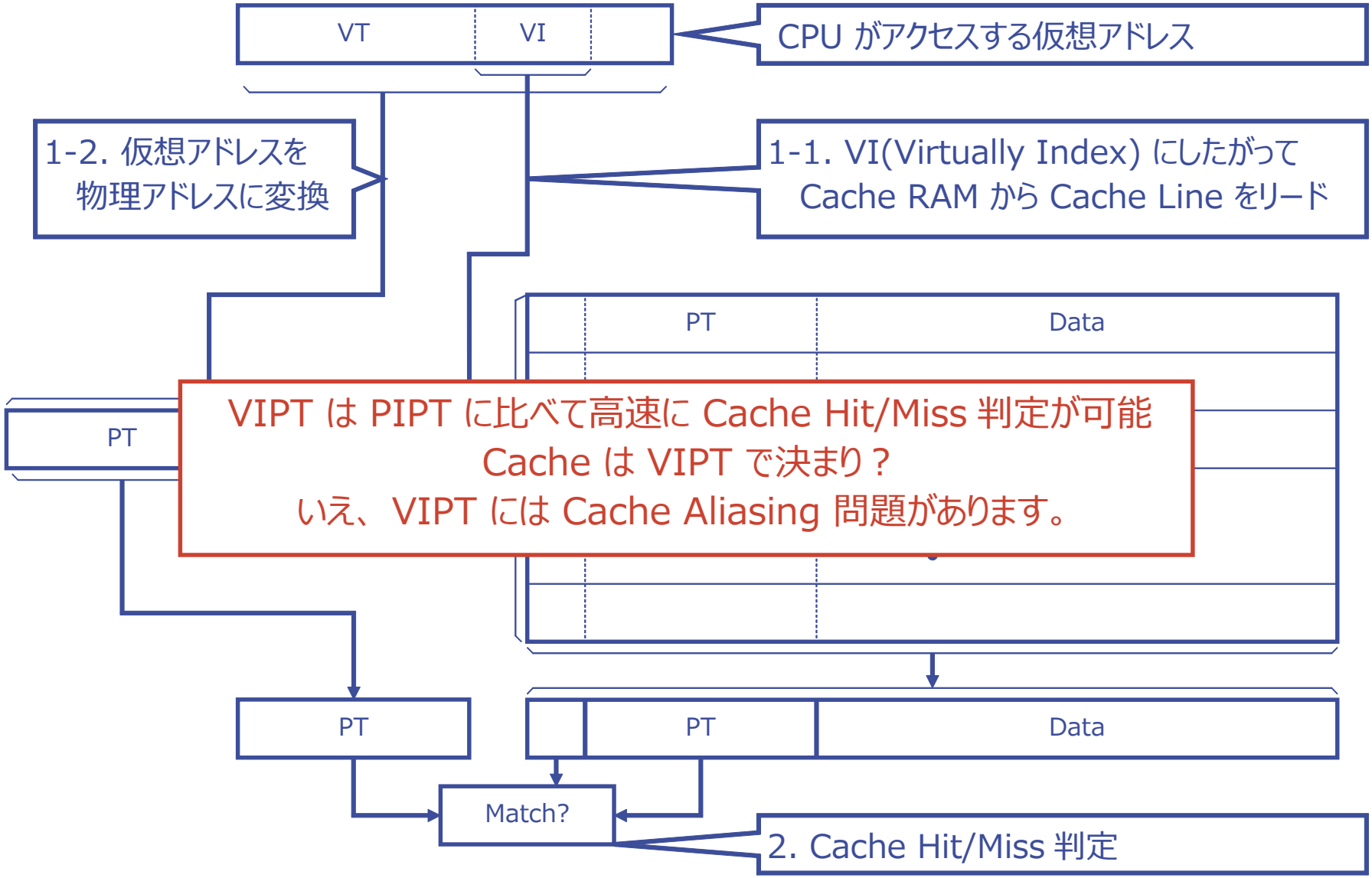
Cache の実装例 - VIPT(Virtually Indexed Physically Tagged) -



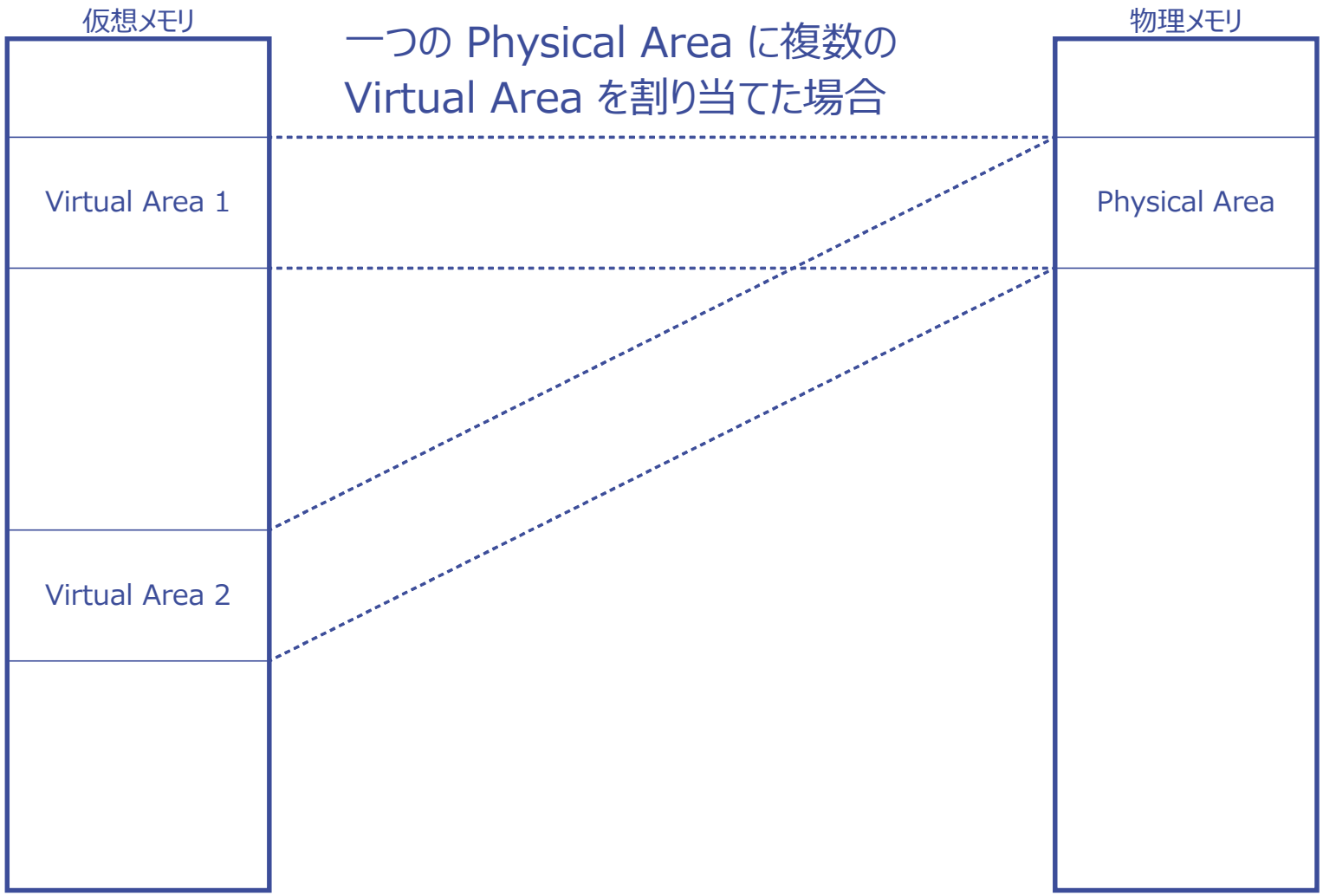
Cache の実装例 - VIPT(Virtually Indexed Physically Tagged) -



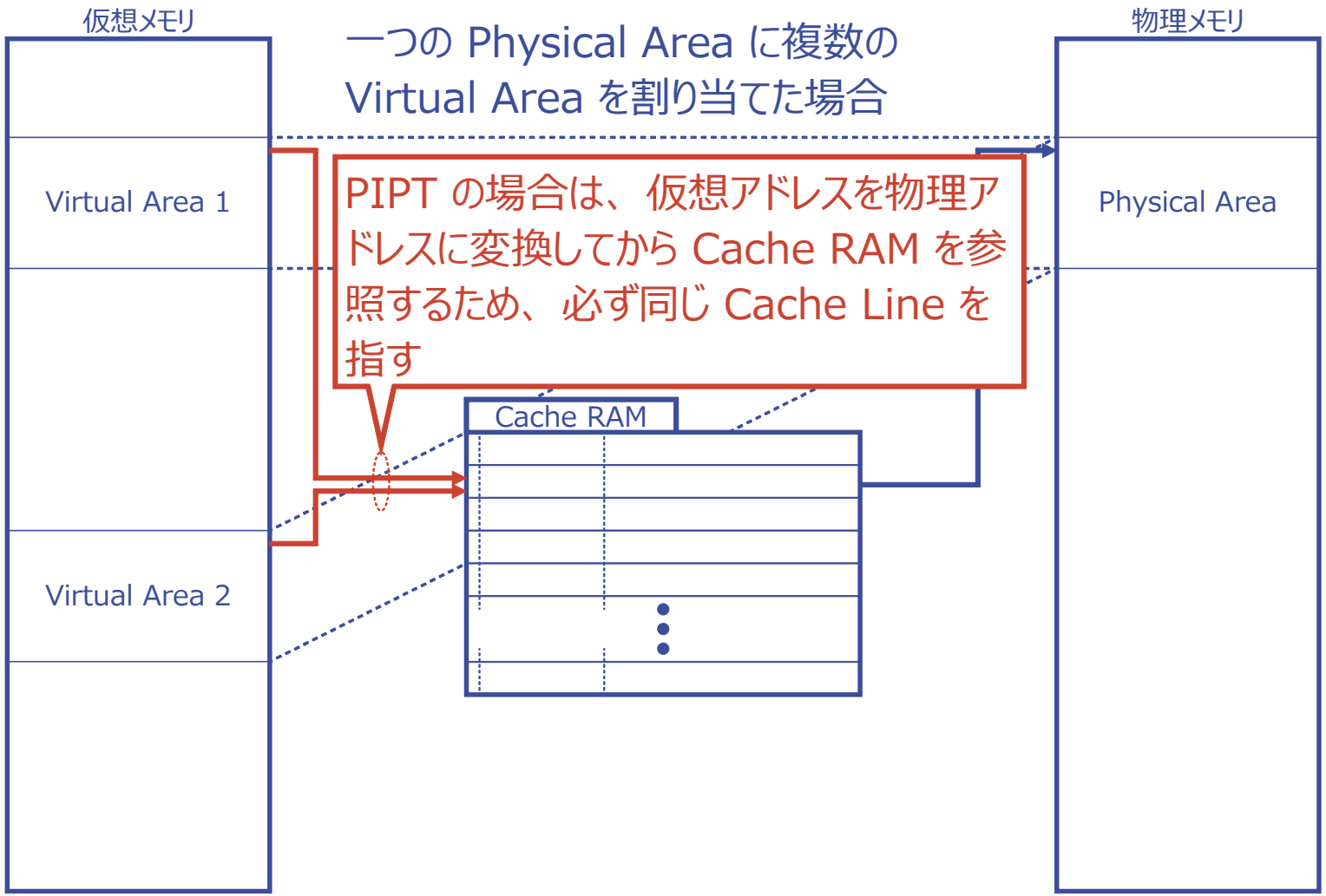
Cache の実装例 - VIPT(Virtually Indexed Physically Tagged) -



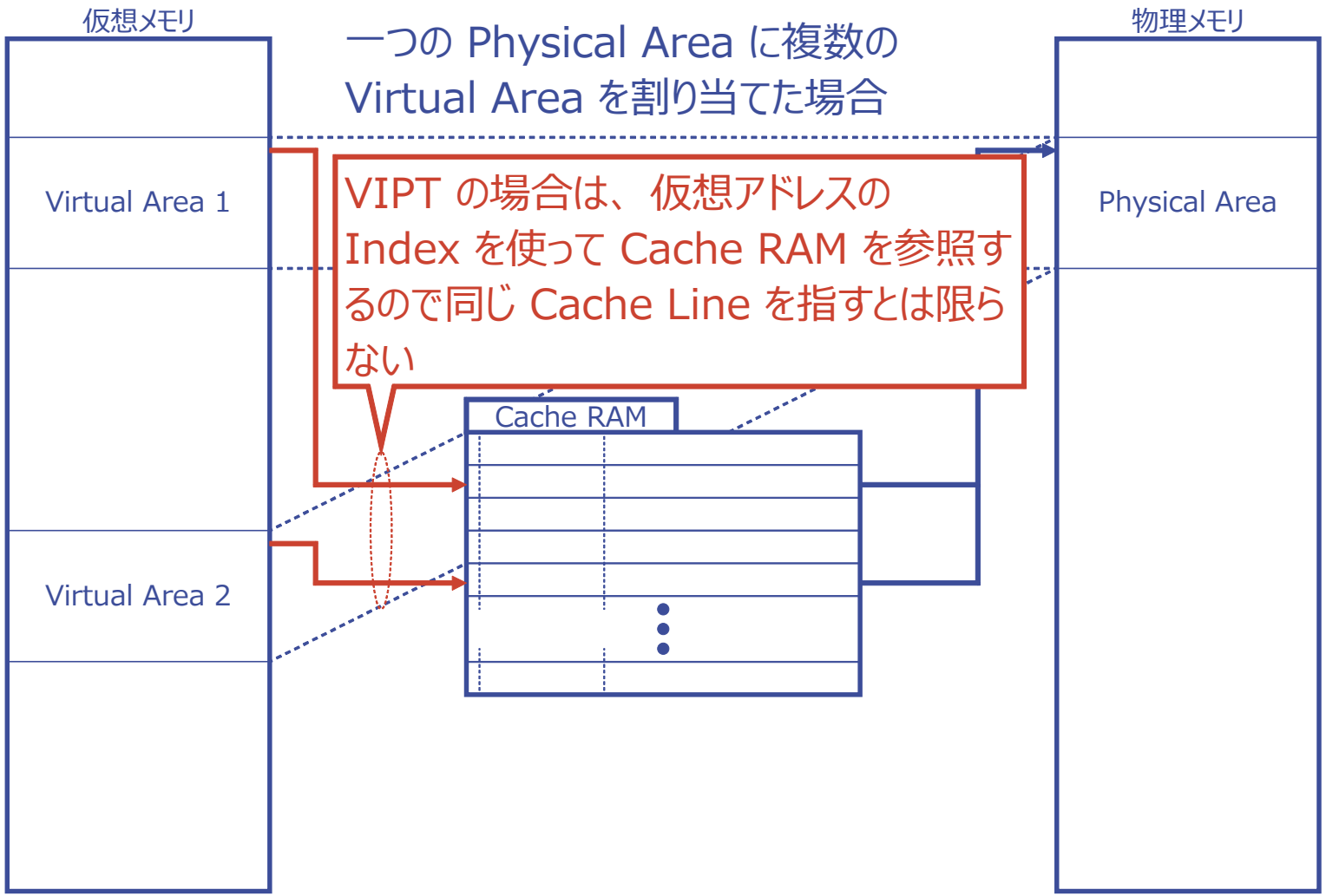
Cache Aliasing 問題とは？



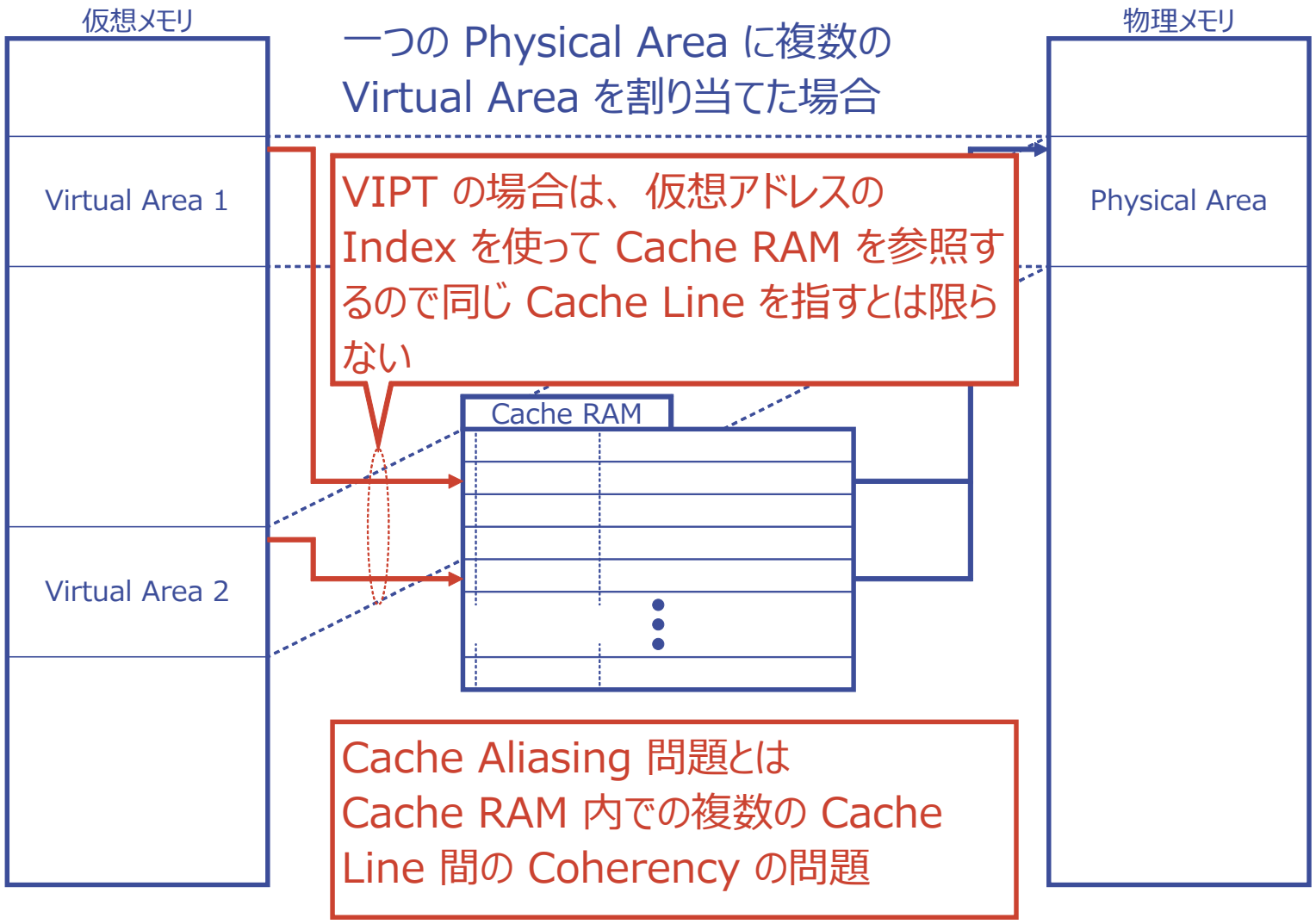
Cache Aliasing 問題とは？



Cache Aliasing 問題とは？



Cache Aliasing 問題とは？



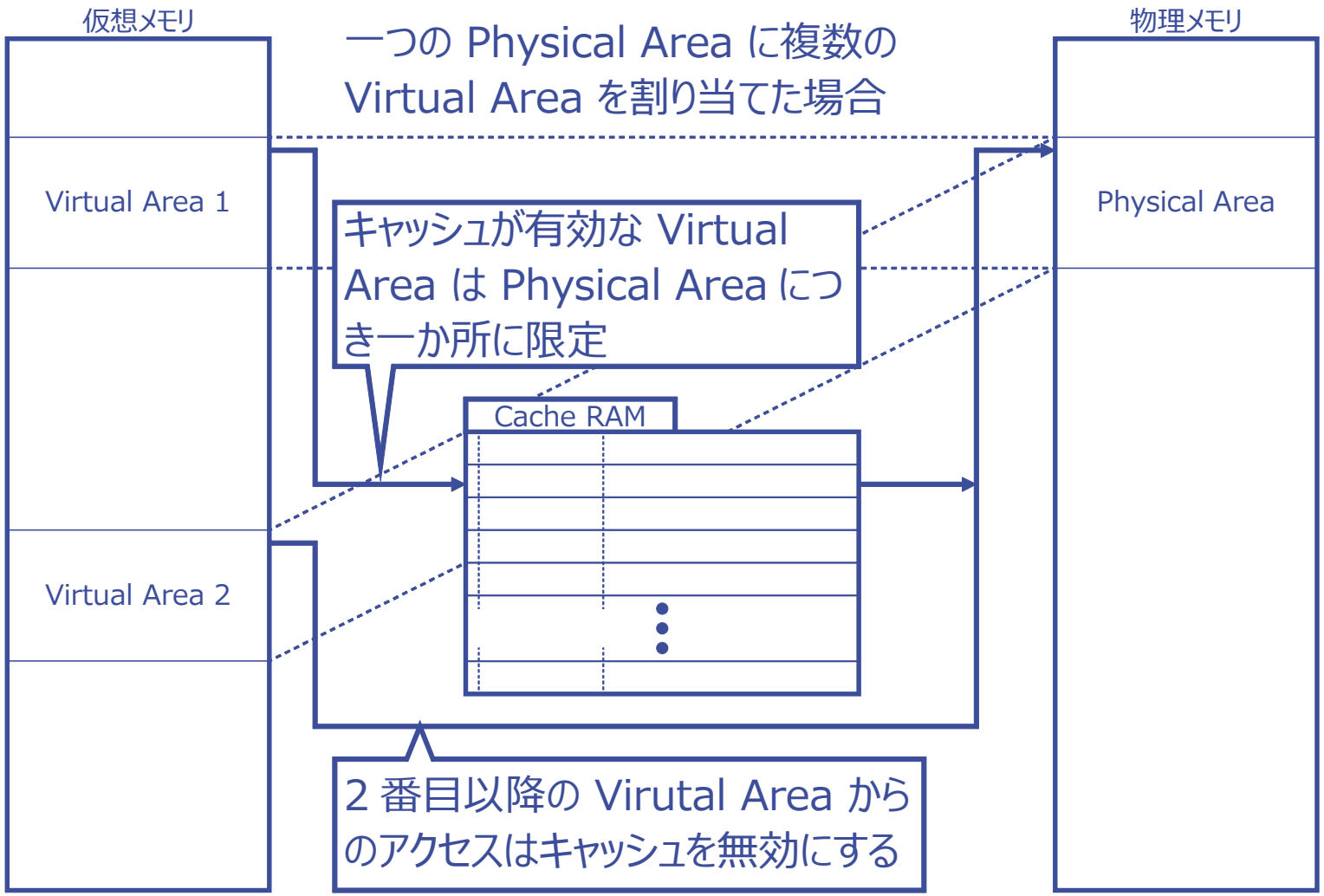
Cache Aliasing 問題を解決する方法

- ハードウェアでなんとかする方法
  - ハードウェアで頑張る？
  - VIPT をあきらめて PIPT にする
- ソフトウェアでなんとかする方法
  - Virtually Index が同じになるように仮想アドレスを割り当てる
    - ページサイズを大きくする
    - ページカラーリング手法
  - キャッシュを使わない

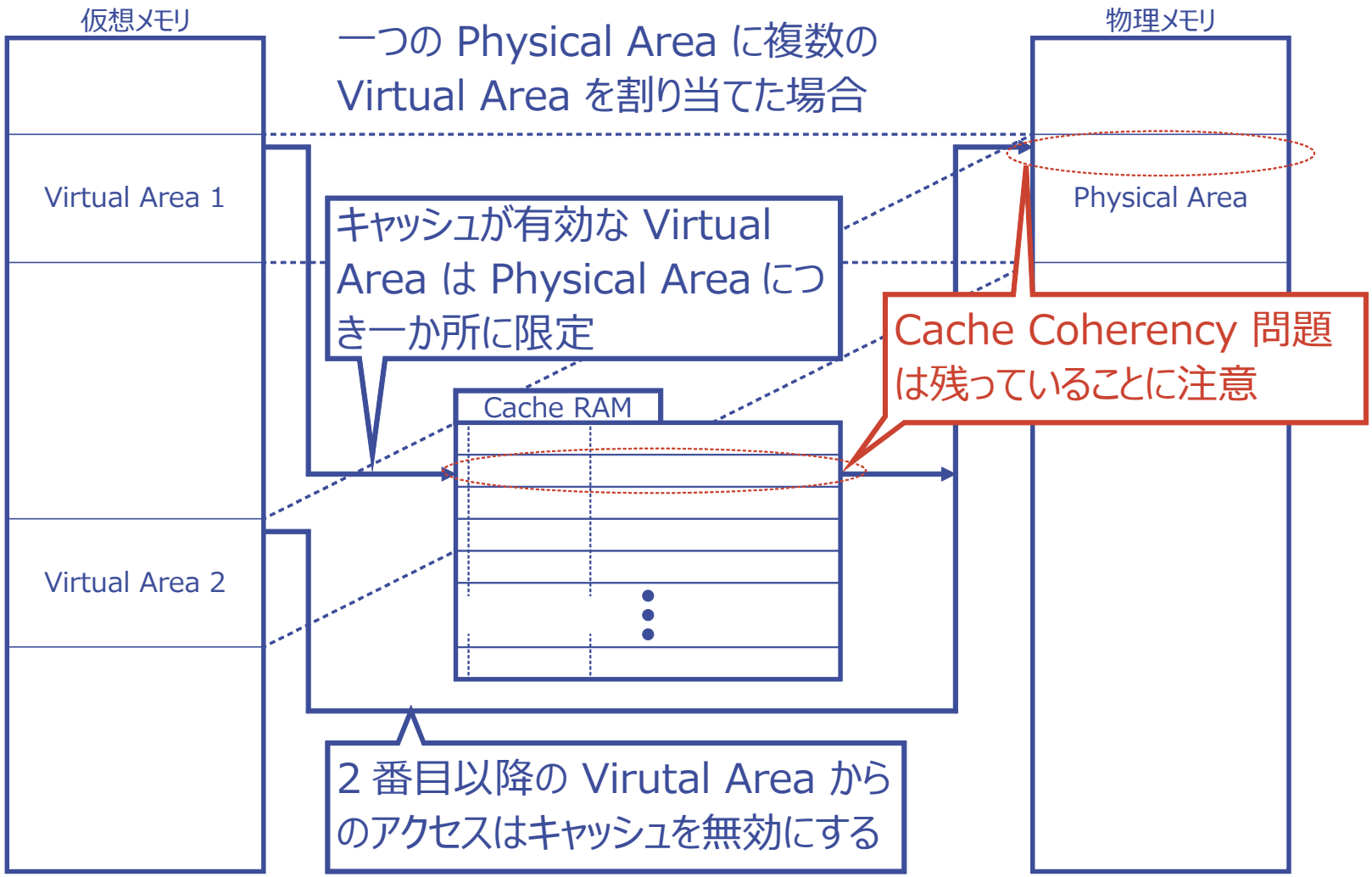
ソフトウェアでなんとかする方法 - ページカラーリング手法 -

- キャッシュカラーリングとも呼ばれる
- 仮想アドレスの割り当てを工夫する
- もともとは Cache Thrashing 対策
  - 頻繁に同じキャッシュラインを別の物理アドレスに対応する内容で置き換えてしまう現象
- Cache Aliasing 対策にも利用できる
- 実装がかなり複雑

ソフトウェアでなんとかする方法 - キャッシュを使わない -



ソフトウェアでなんとかする方法 - キャッシュを使わない -



お品書き

- 背景
  - 主な登場人物の紹介
- Cache Coherency 問題
  - Cache Coherency 問題とは？
  - Cache Coherency 問題を解決する方法
- Cache Aliasing 問題
  - Cache Aliasing 問題とは？
  - Cache Aliasing 問題を解決する方法
- Linux で DMA バッファを mmap するとキャッシュが効かない仕組み
  - dev_is_dma_coherent() の役割
  - キャッシュが効かなくなる場合の例

dma-direct - Linux の新しい DMA Mapping API の実装

- Linux では DMA バッファの管理は DMA Mapping API が窓口
- Linux Kernel 5.0 より DMA Mapping API の実装として新たに dma-direct が導入された
  - 4.20 以前はアーキテクチャごとに個別に実装されていた
  - 5.0 以降は dma-direct に統合されている（一部例外あり
  - arm(32bit)以外は 5.0 以降、dma-direct に統合
  - arm(32bit)は 6.0 以降、dma-direct に統合

- この話は dma-direct が対象

超重要キーワード: dev_is_dma_coherent

https://elixir.bootlin.com/linux/v6.1.62/source/include/linux/dma-map-ops.h#L263

```
#if defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_DEVICE) || ¥
        defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU) || ¥
        defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU_ALL)
extern bool dma_default_coherent;
static inline bool dev_is_dma_coherent(struct device *dev)
{
        return dev->dma_coherent;
}
#else
static inline bool dev_is_dma_coherent(struct device *dev)
{
        return true;
}
#endif /* CONFIG_ARCH_HAS_DMA_COHERENCE_H */
```

struct device の dma_coherent フィールドの値を返すアーキテクチャ

arc/arm/arm64/m68k/mips/parisc/powerpc/riscv(linux 6.0 以降)/sparc/sh 等

常に true を返すアーキテクチャ

x86/ia64/riscv(linux 5.19 以前)

Linux DMA Mapping API での Cache Coherency 制御例

```
static ssize_t device_file_read(struct file* file, char __user* buff, size_t count, loff_t* ppos)
{       struct object*  this      = file->private_data;
        int             result    = 0;
        size_t          xfer_size;
        size_t          remain_size;
        dma_addr_t      phys_addr;
        void*           virt_addr;

        phys_addr = this->phys_addr + *ppos;
        virt_addr = this->virt_addr + *ppos;
        xfer_size = (*ppos + count >= this->size) ? this->size - *ppos : count;

        dma_sync_single_for_cpu(this->dev, phys_addr, xfer_size, DMA_FROM_DEVICE);

        if ((remain_size = copy_to_user(buff, virt_addr, xfer_size)) != 0) {
                result = 0;
                goto return_unlock;
        }

        dma_sync_single_for_device(this->dev, phys_addr, xfer_size, DMA_FROM_DEVICE);
        :
        (中略)
        :
        return result;
}
```

この間は CPU のみがアクセスし、
DEVICE はアクセスしない

dma_sync_single_for_device()

https://elixir.bootlin.com/linux/v6.1.62/source/kernel/dma/mapping.c#L342

```
void dma_sync_single_for_device(struct device *dev, dma_addr_t addr,
                size_t size, enum dma_data_direction dir)
{
        const struct dma_map_ops *ops = get_dma_ops(dev);

        BUG_ON(!valid_dma_direction(dir));
        if (dma_map_direct(dev, ops))
                dma_direct_sync_single_for_device(dev, addr, size, dir);
        else if (ops->sync_single_for_device)
                ops->sync_single_for_device(dev, addr, size, dir);
        debug_dma_sync_single_for_device(dev, addr, size, dir);
}
EXPORT_SYMBOL(dma_sync_single_for_device);
```

dma_direct_sync_single_for_device()

https://elixir.bootlin.com/linux/v6.1.62/source/kernel/dma/direct.h#L55

```
static inline void dma_direct_sync_single_for_device(struct device *dev,
                dma_addr_t addr, size_t size, enum dma_data_direction dir)
{
        phys_addr_t paddr = dma_to_phys(dev, addr);

        if (unlikely(is_swiotlb_buffer(dev, paddr)))
                swiotlb_sync_single_for_device(dev, paddr, size, dir);

        if (!dev_is_dma_coherent(dev))
                arch_sync_dma_for_device(paddr, size, dir);
}
```

・dev_is_dma_coherent() が true の場合

　何もしない

・dev_is_dma_coherent() が false の場合

　arch_sync_dma_for_device() を呼び出す(アーキテクチャごとに実装されている Cache Flush を実行する)

dev_is_dma_coherent() の役割　その 1

- dev_is_dma_coherent() が true の場合
  - 事実:sync 時には何もしない
  - 推測:このデバイスの Cache Coherency はハードウェアで行う(または Cache が無効になっている)という事を示す

- dev_is_dma_coherent() が false の場合
  - 事実:sync 時には Cache の Flush/Invalidiate を行う
  - 推測:このデバイスの Cache Coherency はソフトウェアで行わなければならないことを示す

Linux の Cache Aliasing 問題対応

- ページカラーリングは採用しない
  - fa.linux.kernel というニュースグループで Linus さんが投稿

    https://yarchive.net/comp/linux/page_coloring.html
  - 実装コストが高すぎるということのようです

- ページのサイズを大きくするのは汎用 OS としては好ましくない

- 2 番目以降の仮想空間からのアクセスはキャッシュを無効にする

dma_mmap_attrs()

https://elixir.bootlin.com/linux/v6.1.62/source/kernel/dma/mapping.c#L457

```
int dma_mmap_attrs(struct device *dev, struct vm_area_struct *vma,
                   void *cpu_addr, dma_addr_t dma_addr, size_t size,
                   unsigned long attrs)
{
        const struct dma_map_ops *ops = get_dma_ops(dev);

        if (dma_alloc_direct(dev, ops))
                return dma_direct_mmap(dev, vma, cpu_addr, dma_addr, size,
                                       attrs);
        if (!ops->mmap)
                return -ENXIO;
        return ops->mmap(dev, vma, cpu_addr, dma_addr, size, attrs);
}
EXPORT_SYMBOL(dma_mmap_attrs);
```

dma_mmap_attrs() は DMA バッファに仮想アドレスを割り当てる際に呼ばれる DMA Mapping API

dma_direct_mmap()

https://elixir.bootlin.com/linux/v6.1.62/source/kernel/dma/direct.c#L555

```
int dma_direct_mmap(struct device *dev, struct vm_area_struct *vma,
                    void *cpu_addr, dma_addr_t dma_addr, size_t size,
                    unsigned long attrs)
{
        unsigned long user_count = vma_pages(vma);
        unsigned long count = PAGE_ALIGN(size) >> PAGE_SHIFT;
        unsigned long pfn = PHYS_PFN(dma_to_phys(dev, dma_addr));
        int ret = -ENXIO;

        vma->vm_page_prot = dma_pgprot(dev, vma->vm_page_prot, attrs);
        if (force_dma_unencrypted(dev))
                vma->vm_page_prot = pgprot_decrypted(vma->vm_page_prot);

        if (dma_mmap_from_dev_coherent(dev, vma, cpu_addr, size, &ret))
                return ret;
        if (dma_mmap_from_global_coherent(vma, cpu_addr, size, &ret))
                return ret;

        if (vma->vm_pgoff >= count || user_count > count - vma->vm_pgoff)
                return -ENXIO;
        return remap_pfn_range(vma, vma->vm_start, pfn + vma->vm_pgoff,
                               user_count << PAGE_SHIFT, vma->vm_page_prot);
}
```

vma->vm_page_prot の役割

- ページの属性を指定する
- 指定する値はアーキテクチャ毎に異なる
- 属性にはキャッシュの有効/無効も含む

dma_pgprot()

https://elixir.bootlin.com/linux/v6.1.62/source/kernel/dma/mapping.c#L415

```
/*
 * Return the page attributes used for mapping dma_alloc_* memory, either in
 * kernel space if remapping is needed, or to userspace through dma_mmap_*.
 */
pgprot_t dma_pgprot(struct device *dev, pgprot_t prot, unsigned long attrs)
{
        if (dev_is_dma_coherent(dev))
                return prot;
#ifdef CONFIG_ARCH_HAS_DMA_WRITE_COMBINE
        if (attrs & DMA_ATTR_WRITE_COMBINE)
                return pgprot_writecombine(prot);
#endif
        return pgprot_dmacoherent(prot);
}
```

dev_is_dma_coherent() が false の場合は pgprot_dmacoherent() を呼び出す

pgprot_dmacoherent() は何れのアーキテクチャでもキャッシュを無効にしている

- https://elixir.bootlin.com/linux/v6.1.62/source/arch/arm/include/asm/pgtable.h#L128
- https://elixir.bootlin.com/linux/v6.1.62/source/arch/arm64/include/asm/pgtable.h#L577
- https://elixir.bootlin.com/linux/v6.1.62/source/arch/m68k/include/asm/pgtable_mm.h#L182
- https://elixir.bootlin.com/linux/v6.1.62/source/include/linux/dma-map-ops.h#L291

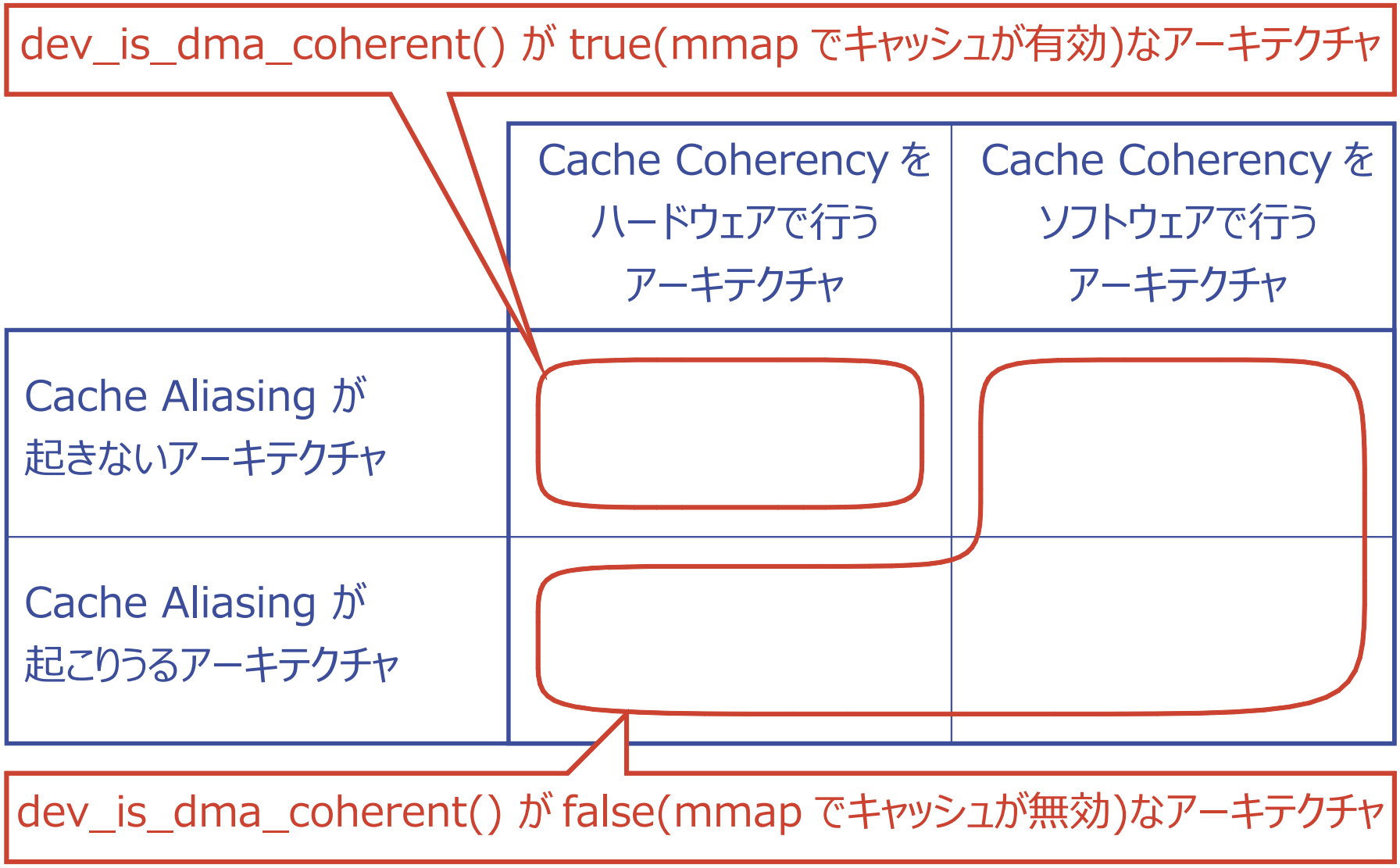
dev_is_dma_coherent() の役割　その 2

- dev_is_dma_coherent() が true の場合
  - 事実:mmap では何もしない(Cache の設定は変更しない)
  - 推測:このデバイスは Cache Aliasing 問題は起きないことを示す？
- dev_is_dma_coherent() が false の場合
  - 事実:mmap で Cache を無効にしている
  - 推測:このデバイスは Cache Aliasing 問題が起こりうることを示す？

dev_is_dma_coherent() の役割

	Cache Coherency を ハードウェアで行う アーキテクチャ	Cache Coherency を ソフトウェアで行う アーキテクチャ
Cache Aliasing が 起きないアーキテクチャ		
Cache Aliasing が 起こりうるアーキテクチャ		

dev_is_dma_coherent() の役割

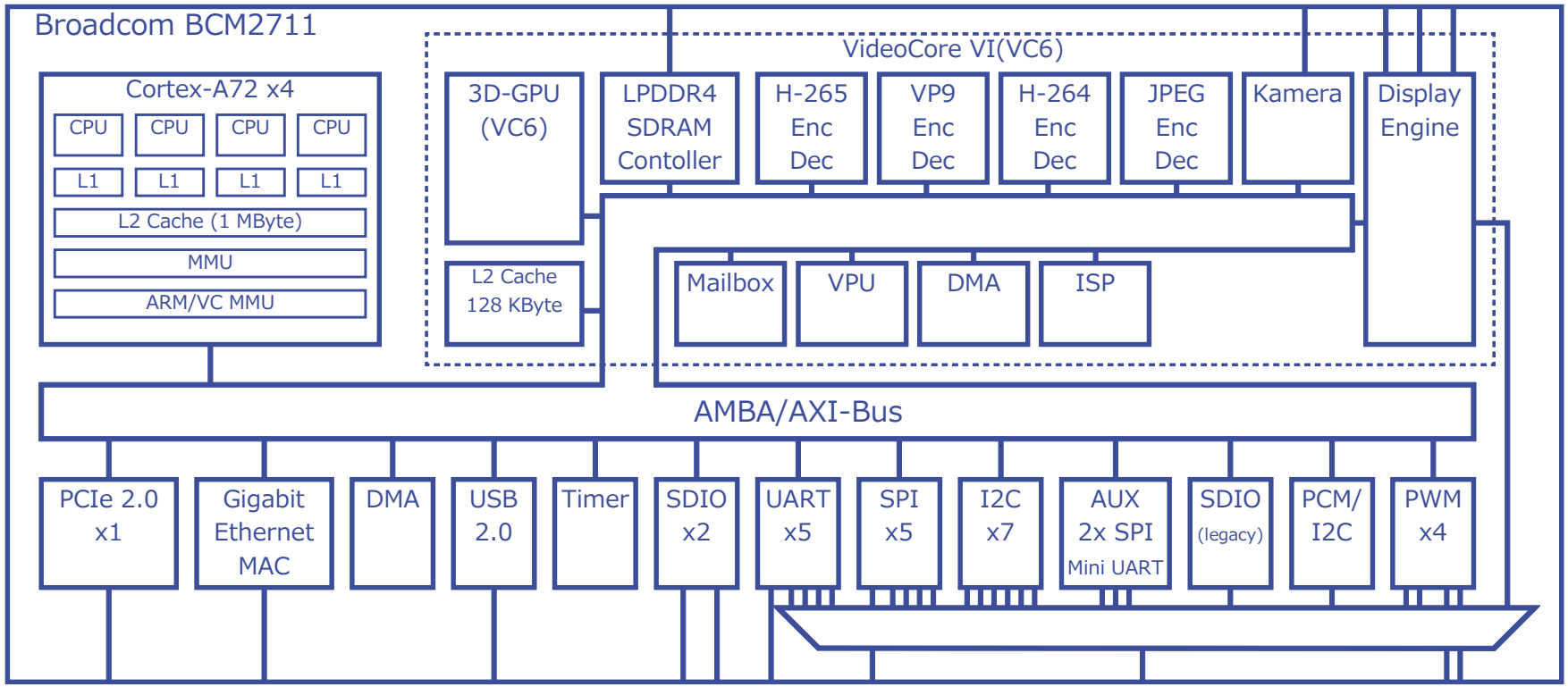


Linux Kernel での残念な点

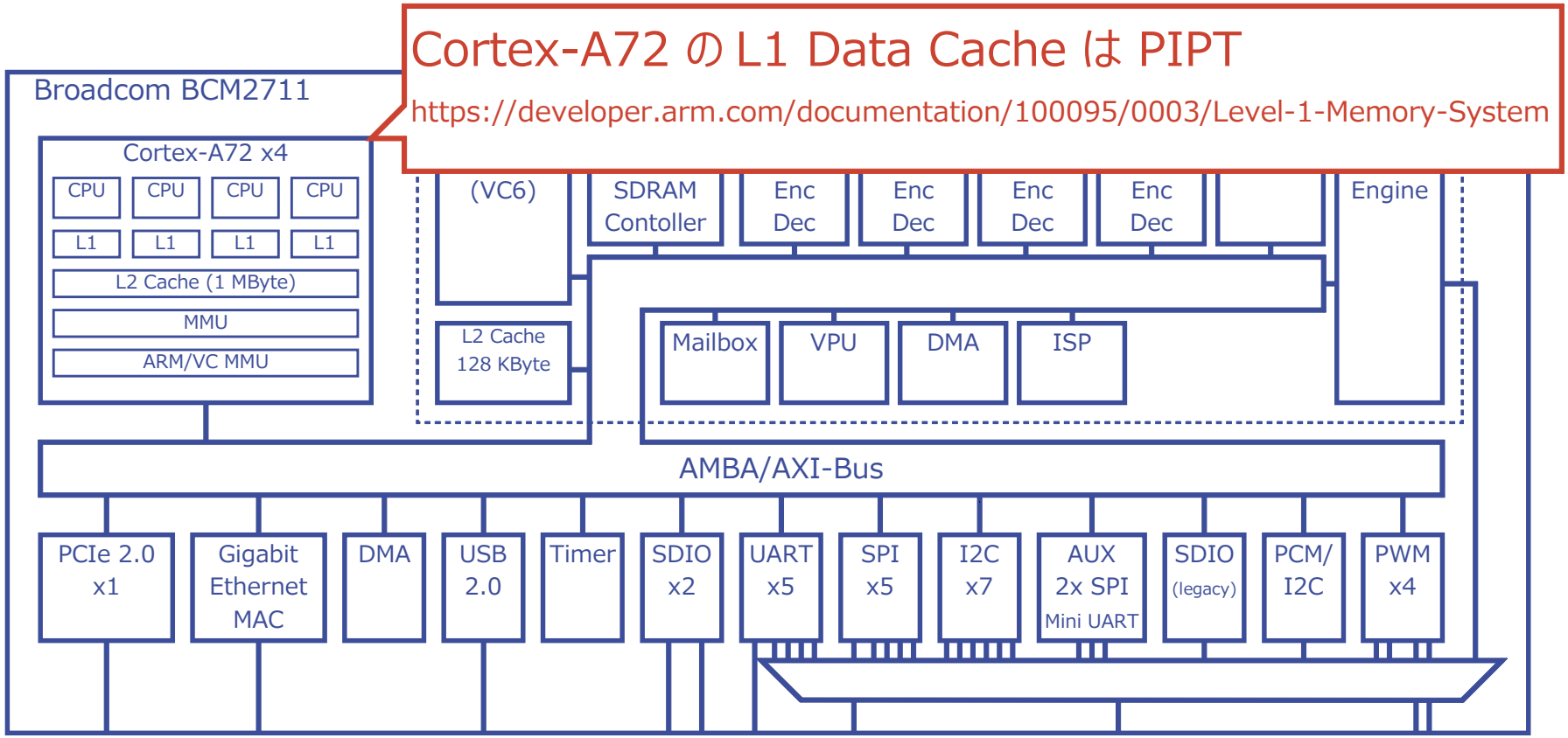
	Cache Coherency を ハードウェアで行う アーキテクチャ	Cache Coherency を ソフトウェアで行う アーキテクチャ
Cache Aliasing が 起きないアーキテクチャ		
Cache Aliasing が 起こりうるアーキテクチャ		

Cache Aliasing が起きなくても
Cache Coherency をソフトウェアで行うアーキテクチャの場合は、
mmap でキャッシュが無効になる

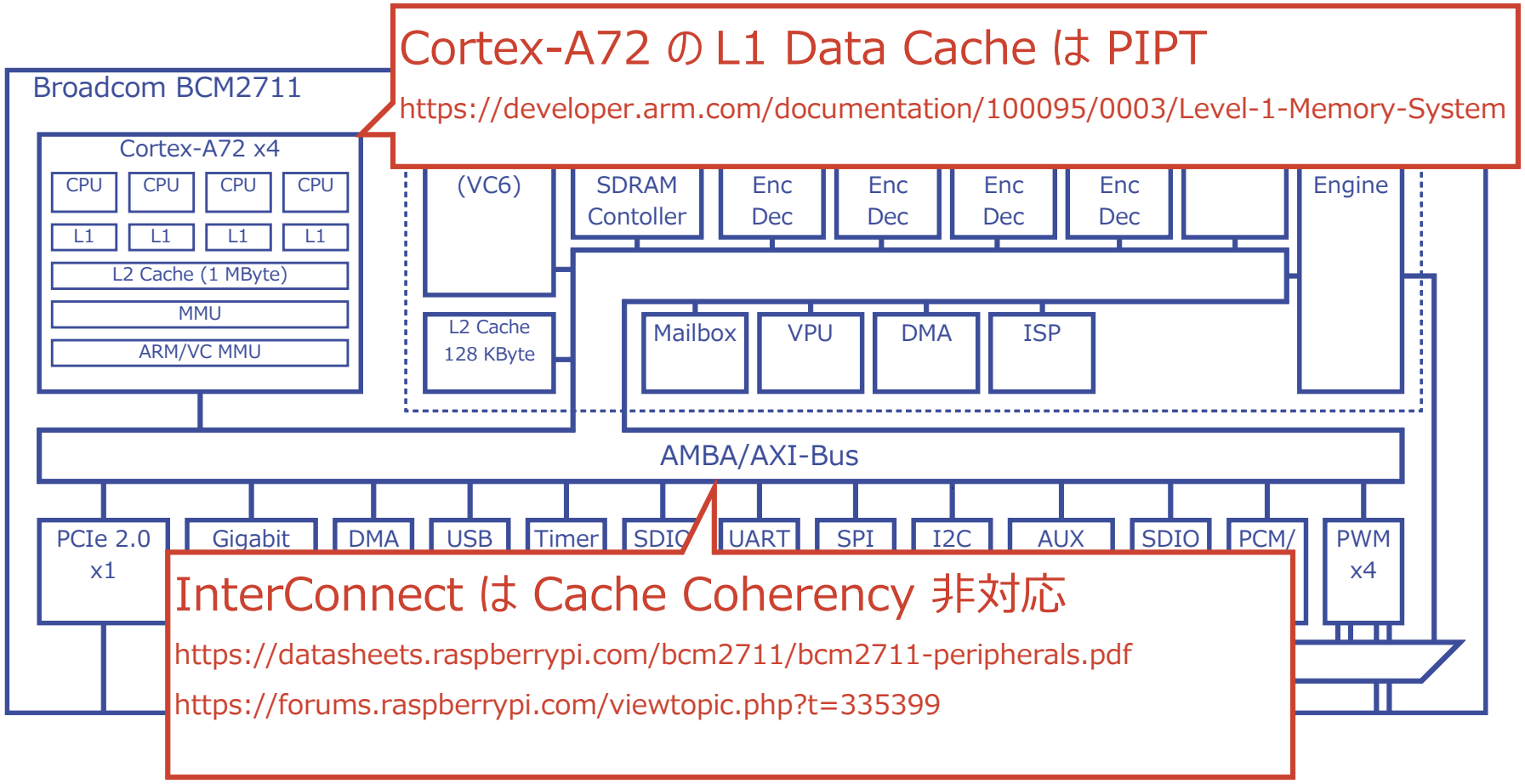
BCM2711 (Raspberry Pi 4 の SoC) の内部ブロック -1-



BCM2711 (Raspberry Pi 4 の SoC) の内部ブロック -2-



BCM2711 (Raspberry Pi 4 の SoC) の内部ブロック -2-



Raspberry Pi は mmap 時にキャッシュが無効になるアーキテクチャ

Raspberry Pi で mmap が遅くなる事例

- Libcamera memcpy from mmap'd region very slow (CM4)
  https://forums.raspberrypi.com/viewtopic.php?t=352554
- Why processing V4L2 Unicam buffers in userspace might be slow
  https://forums.raspberrypi.com/viewtopic.php?t=315272
- [HOW-TO] processing high resolution video at high framerate
  https://github.com/raspberrypi/picamera2/issues/740
- Could this be used with V4L2/libcamera buffers on the Raspberry Pi 4 (Arm A72)
  https://github.com/ikwzm/udmabuf/issues/107

RISC-V も他人事ではない -1-

https://elixir.bootlin.com/linux/v6.1.62/source/include/linux/dma-map-ops.h#L263

```
#if defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_DEVICE) || ¥
        defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU) || ¥
        defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU_ALL)
extern bool dma_default_coherent;
static inline bool dev_is_dma_coherent(struct device *dev)
{
        return dev->dma_coherent;
}
#else
static inline bool dev_is_dma_coherent(struct device *dev)
{
        return true;
}
#endif /* CONFIG_ARCH_HAS_DMA_COHERENCE_H */
```

struct device の dma_coherent フィールドの値を返すアーキテクチャ

arc/arm/arm64/m68k/mips/parisc/powerpc/riscv(linux 6.0 以降)/sparc/sh 等

常に true を返すアーキテクチャ

x86/ia64/riscv(linux 5.19 以前)

RISC-V も他人事ではない -2-

https://elixir.bootlin.com/linux/v6.1.62/source/include/linux/dma-map-ops.h#L263

```
#if defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_DEVICE) || ¥
        defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU) || ¥
        defined(CONFIG_ARCH_HAS_SYNC_DMA_FOR_CPU_ALL)
extern bool dma_default_coherent;
static inline bool dev_is_dma_coherent(struct device *dev)
{
        return dev->dma_coherent;
}
#else
static inline bool dev_is_dma_coherent(struct device *dev)
{
        return true;
}
#endif /* CONFIG_ARCH_HAS_DMA_COHERENCE_H */
```

https://elixir.bootlin.com/linux/v6.1.62/source/arch/riscv/Kconfig#L227

```
config RISCV_DMA_NONCOHERENT
        bool
        select ARCH_HAS_DMA_PREP_COHERENT
        select ARCH_HAS_SYNC_DMA_FOR_DEVICE
        select ARCH_HAS_SYNC_DMA_FOR_CPU
        select ARCH_HAS_SETUP_DMA_OPS
        select DMA_DIRECT_REMAP
```

Linux Kernel 6.0 にて追加

RISC-V も他人事ではない -3-

https://elixir.bootlin.com/linux/v6.1.62/source/arch/riscv/Kconfig.erratas#L61

```
config ERRATA_THEAD_CMO
        bool "Apply T-Head cache management errata"
        depends on ERRATA_THEAD && MMU
        select RISCV_DMA_NONCOHERENT
        default y
        help
          This will apply the cache management errata to handle the
          non-standard handling on non-coherent operations on T-Head SoCs.

          If you don't know what to do here, say "Y".
```

Linux Kernel 6.0 にて追加

RISC-V も他人事ではない -4-

https://elixir.bootlin.com/linux/v6.1.62/source/arch/riscv/Kconfig#L427

```
config RISCV_ISA_ZICBOM
        bool "Zicbom extension support for non-coherent DMA operation"
        depends on TOOLCHAIN_HAS_ZICBOM
        depends on !XIP_KERNEL && MMU
        select RISCV_DMA_NONCOHERENT
        select RISCV_ALTERNATIVE
        default y
        help
          Adds support to dynamically detect the presence of the ZICBOM
          extension (Cache Block Management Operations) and enable its
          usage.

          The Zicbom extension can be used to handle for example
          non-coherent DMA support on devices that need it.

          If you don't know what to do here, say Y.
```

Linux Kernel 6.0 にて追加

RISC-V も他人事ではない -4-

https://elixir.bootlin.com/linux/v6.1.62/source/arch/riscv/Kconfig#L427

```
config RISCV_ISA_ZICBOM
        bool "Zicbom extension support for non-coherent DMA operation"
        depends on TOOLCHAIN_HAS_ZICBOM
        depends on !XIP_KERNEL && MMU
        select RISCV_DMA_NONCOHERENT
        select RISCV_ALTERNATIVE
        default y
        help
          Adds support to dynamically detect the presence of the ZICBOM
          extension (Cache Block Management Operations) and enable its
          usage.

          The Zicbom extension can be used to handle for example
          non-coherent DMA support on devices that need it.

          If you don't know what to do here, say Y.
```

Linux Kernel 5.19 以前はデフォルトで dev_is_dma_coherent() は true

Linux Kernel 6.0 以降は 、キャッシュのコヒーレンシをハードウェアで行う場合は、dev_is_dma_coherent() は struct device の dma_coherent フィールドを明示的に設定しておかないと mmap 時に キャッシュが無効になる恐れがある

dev_is_dma_coherent() をデバイスツリーから true に設定する方法

・デバイスが dma_coherent か否かを、デバイスツリーから判定する関数がこちら

https://elixir.bootlin.com/linux/v6.1.62/source/drivers/of/address.c#L1062

```
bool of_dma_is_coherent(struct device_node *np)
{
	struct device_node *node;
	bool is_coherent = IS_ENABLED(CONFIG_OF_DMA_DEFAULT_COHERENT);

	node = of_node_get(np);

	while (node) {
		if (of_property_read_bool(node, "dma-coherent")) {
			is_coherent = true;
			break;
		}
		if (of_property_read_bool(node, "dma-noncoherent")) {
			is_coherent = false;
			break;
		}
		node = of_get_next_dma_parent(node);
	}
	of_node_put(node);
	return is_coherent;
}
EXPORT_SYMBOL_GPL(of_dma_is_coherent);
```

dev_is_dma_coherent() をデバイスツリーから true に設定する方法

- Linux Kernel のビルド時に OF_DMA_DEFAULT_COHERENT を指定する

https://elixir.bootlin.com/linux/v6.1.62/source/drivers/of/Kconfig#L93

```
config OF_DMA_DEFAULT_COHERENT
        # arches should select this if DMA is coherent by default for OF devices
        bool
```

- device tree に dma-coherent プロパティを指定する

https://elixir.bootlin.com/linux/v6.1.62/source/arch/arm64/boot/dts/nvidia/tegra234.dtsi#L63

```
                gpcdma: dma-controller@2600000 {
                        compatible = "nvidia,tegra234-gpcdma",
                                     "nvidia,tegra186-gpcdma";
                        reg = <0x2600000 0x210000>;
                        resets = <&bpmp TEGRA234_RESET_GPCDMA>;
                        reset-names = "gpcdma";
                        :
                        (中略)
                        :
                        #dma-cells = <1>;
                        iommus = <&smmu_niso0 TEGRA234_SID_GPCDMA>;
                        dma-coherent;
                };
```

まとめ

確定： Cache Coherency をソフトウェアで行うアーキテクチャでは、mmap でキャッシュが無効になる

- dev_is_dma_coherent() が false の場合は cache coherency をソフトウェアで行うと同時に mmap でキャッシュが無効化されている

推測： Cache Aliasing 問題の対策のため？

- Cache Aliasing 問題の解決方法として、複数の仮想アドレスを割り当てる際にキャッシュを有効にするのは 1 箇所のみにする方法がある

疑問： 何故 Cache Coherency 問題と Cache Aliasing 問題を一つのフラグで対応したのか？