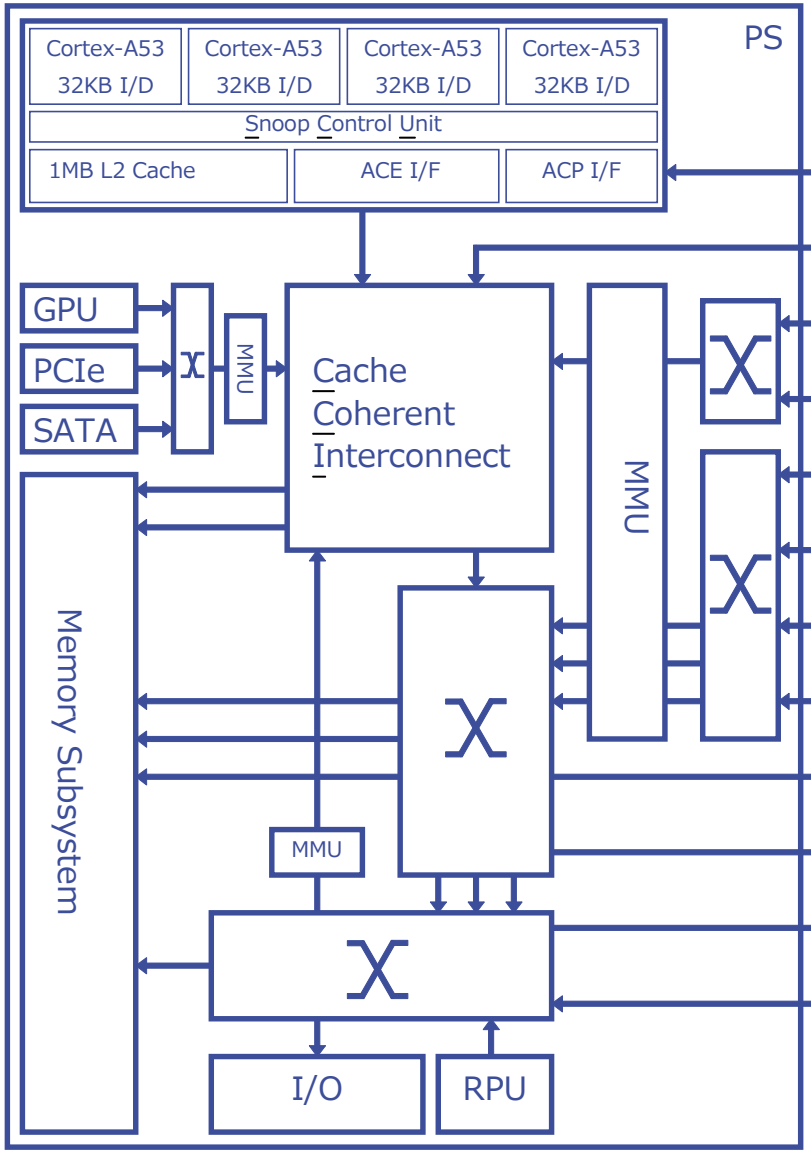
ZynqMP で
PL から PS へのアクセス

2016 年 2 月 20 日

@ikwzm

ZynqMP PS-PL Interface

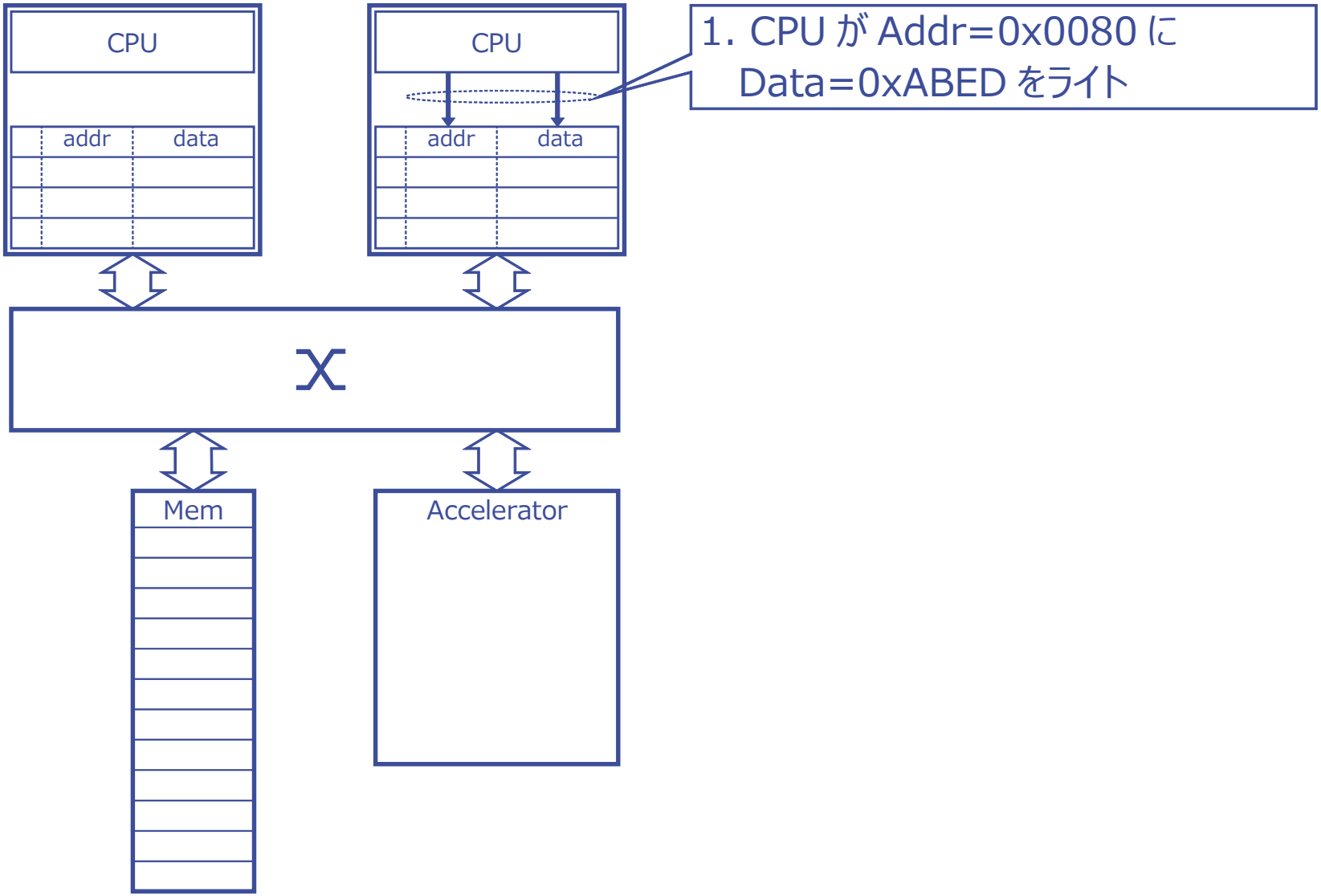


PL

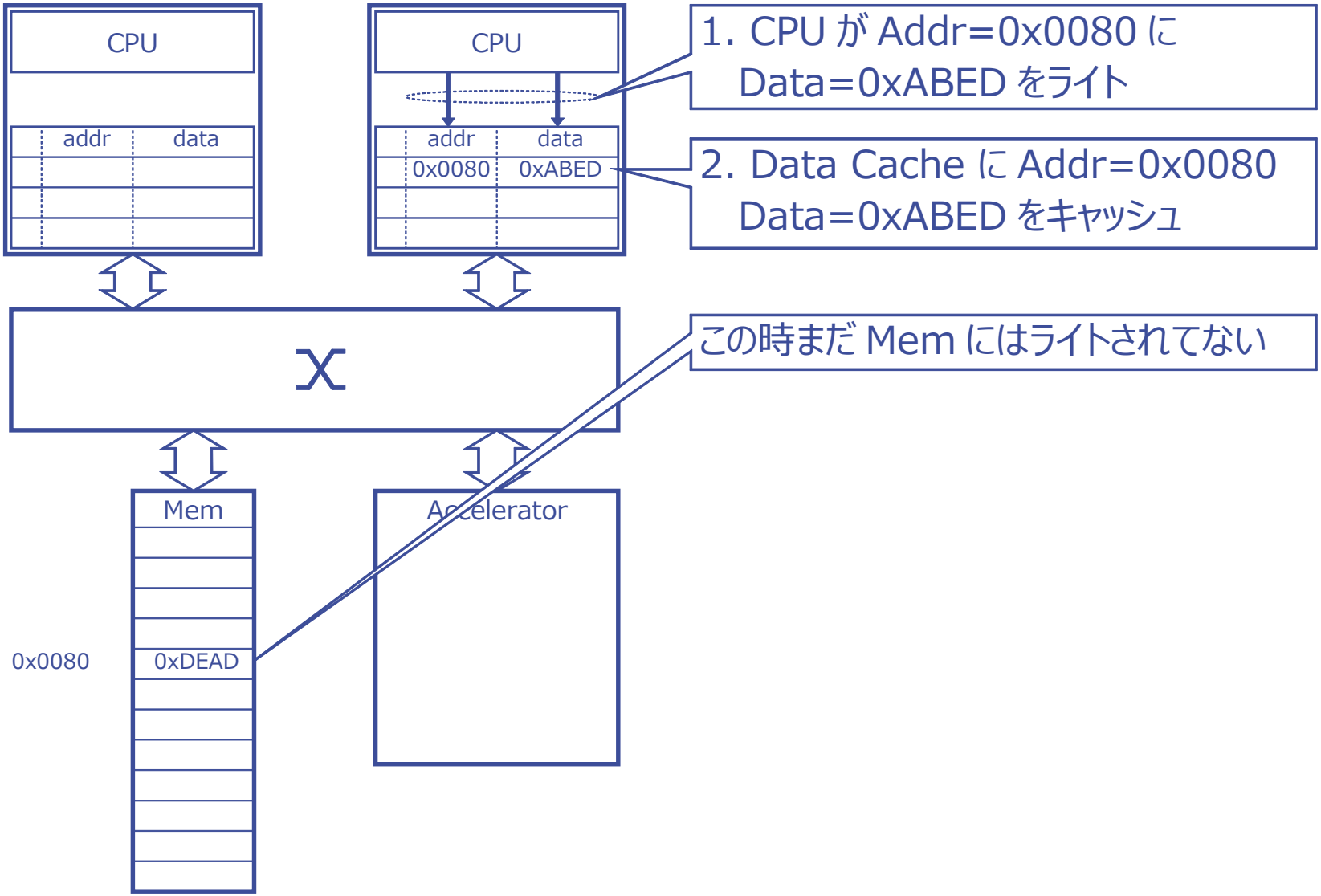
Interface Name	Addr Width	Data Width	MMU	Cache Coherence
S_AXI_ACP_FPD	40	32 or 64 or 128	No	I/O
S_AXI_ACE_FPD				Full
S_AXI_HPC0_FPD	49		Yes	I/O
S_AXI_HPC1_FPD				
S_AXI_HP0_FPD				No
S_AXI_HP1_FPD				
S_AXI_HP2_FPD				
S_AXI_HP3_FPD				
M_AXI_HPM0_FPD	40			
M_AXI_HPM1_FPD				
M_AXI_HPM0_LPD	32			
S_AXI_LPD	49		Yes / No	I/O / No

Cache Coherency のはなし

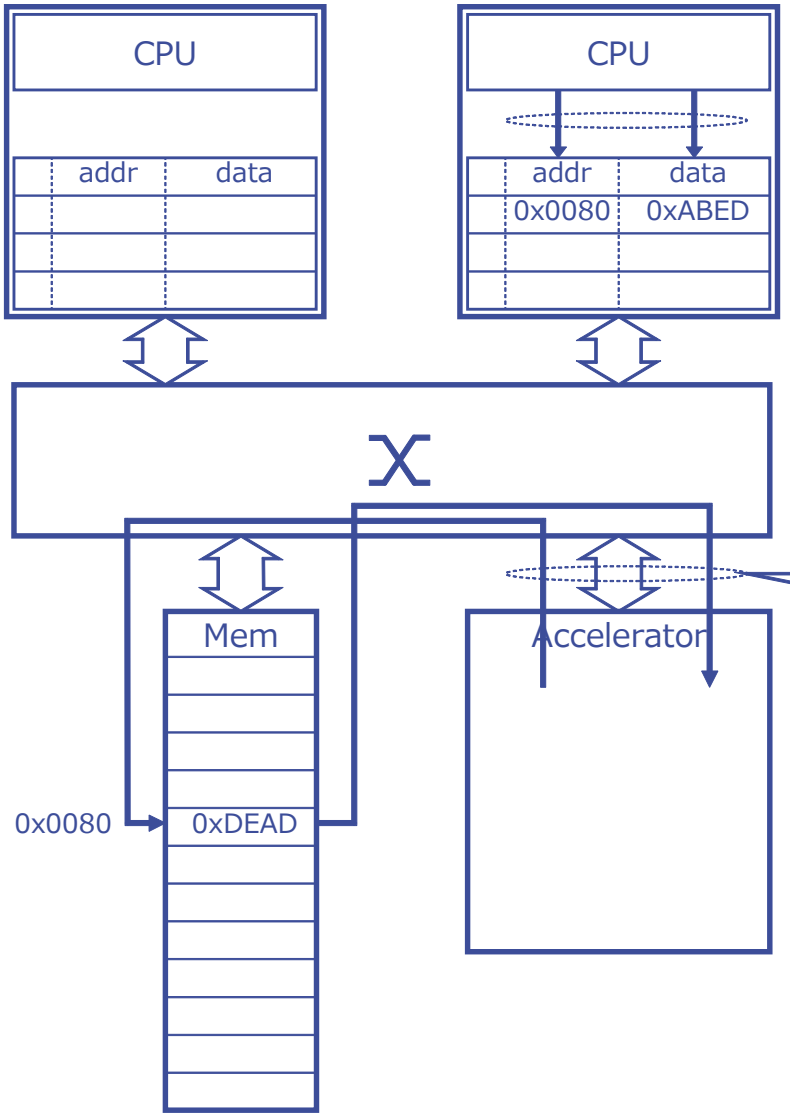
Cache Coherency でトラブルが発生するケース 1



Cache Coherency でトラブルが発生するケース 1



Cache Coherency でトラブルが発生するケース 1



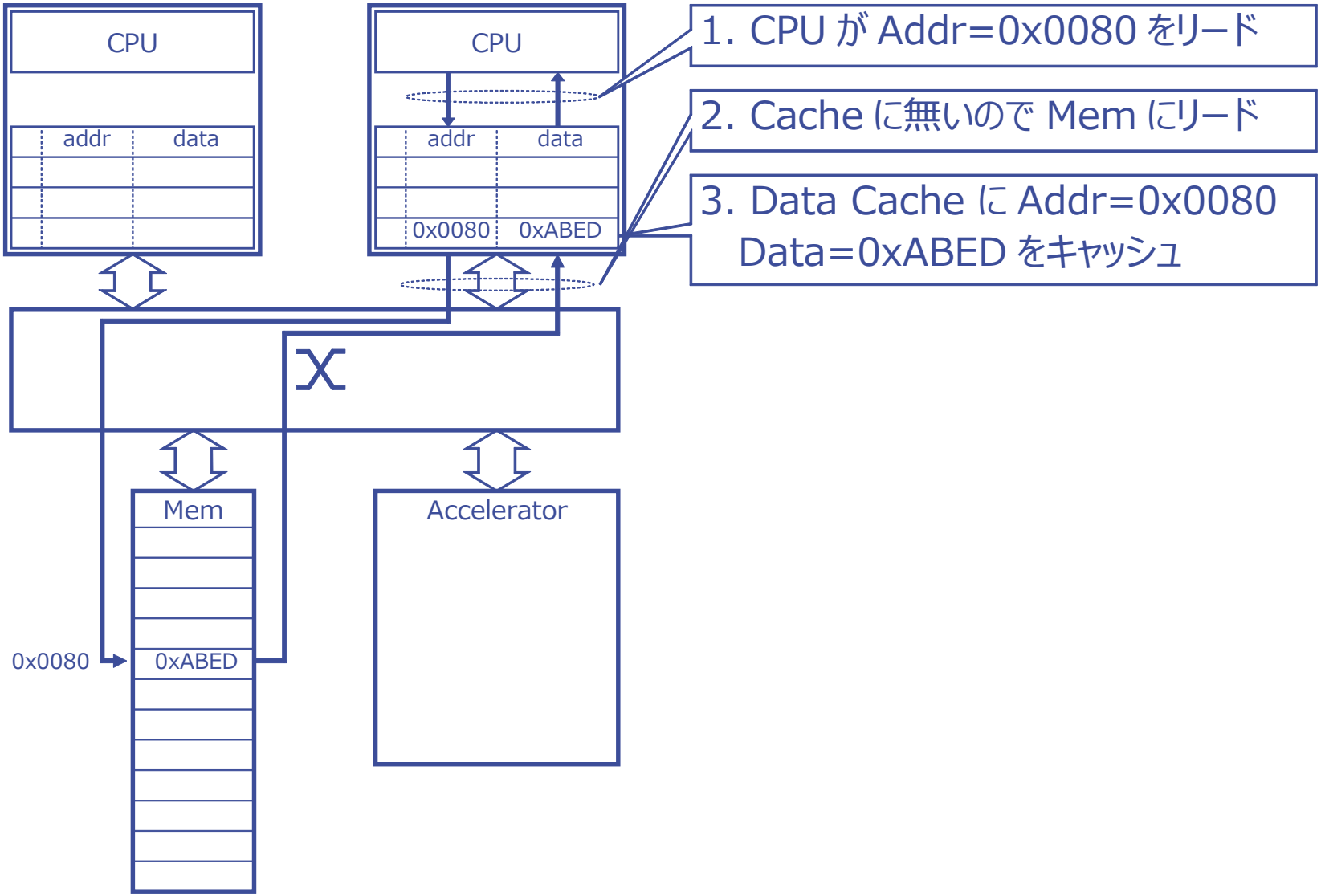
1. CPU が Addr=0x0080 に Data=0xABED をライト

2. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ

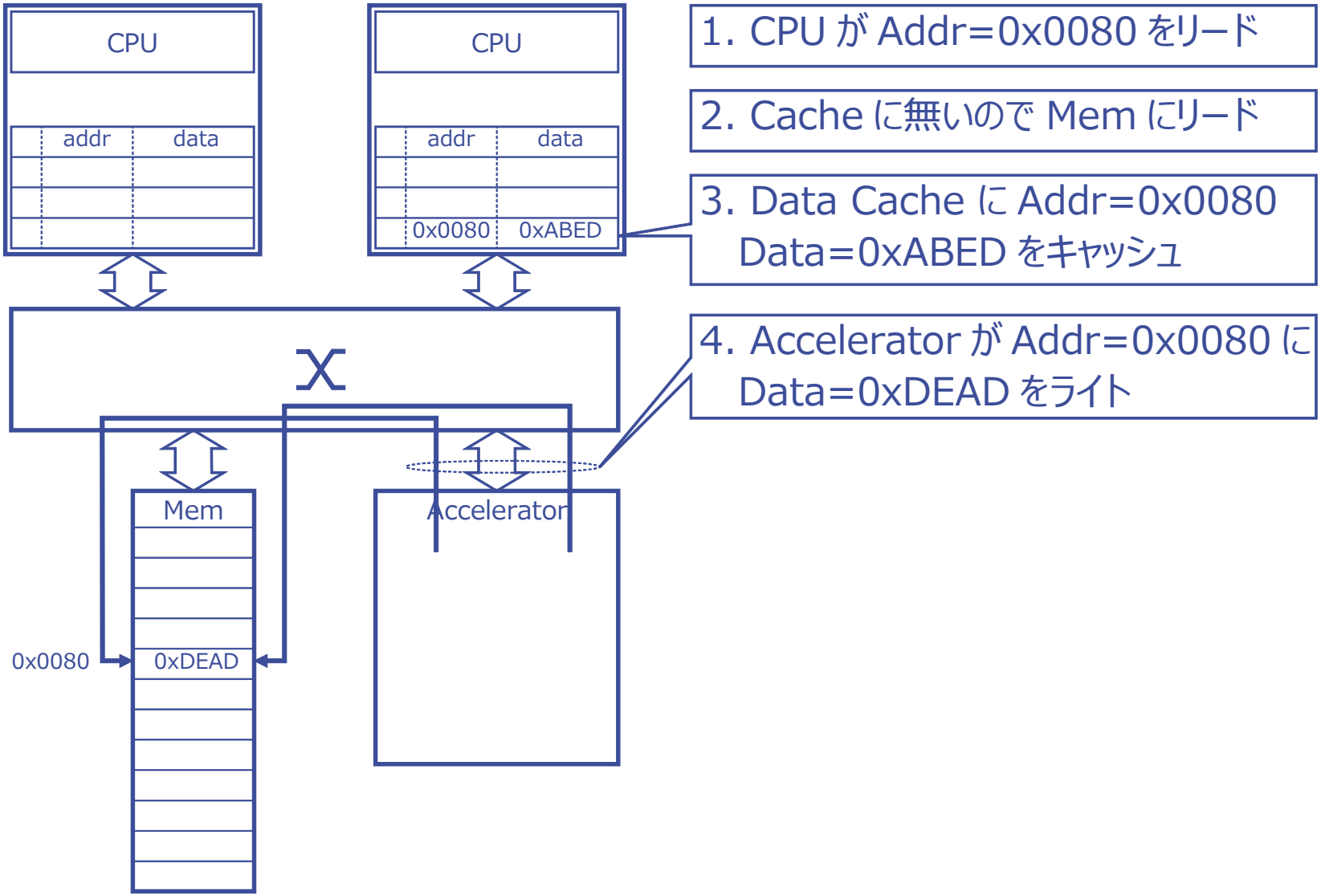
3. Accelerator が Addr=0x0080 をリード

4. 未だ Mem の Addr=0x0080 には Data=0xABED は書き込まれていないので、Accelerator は Data=0xDEAD を読んでしまう

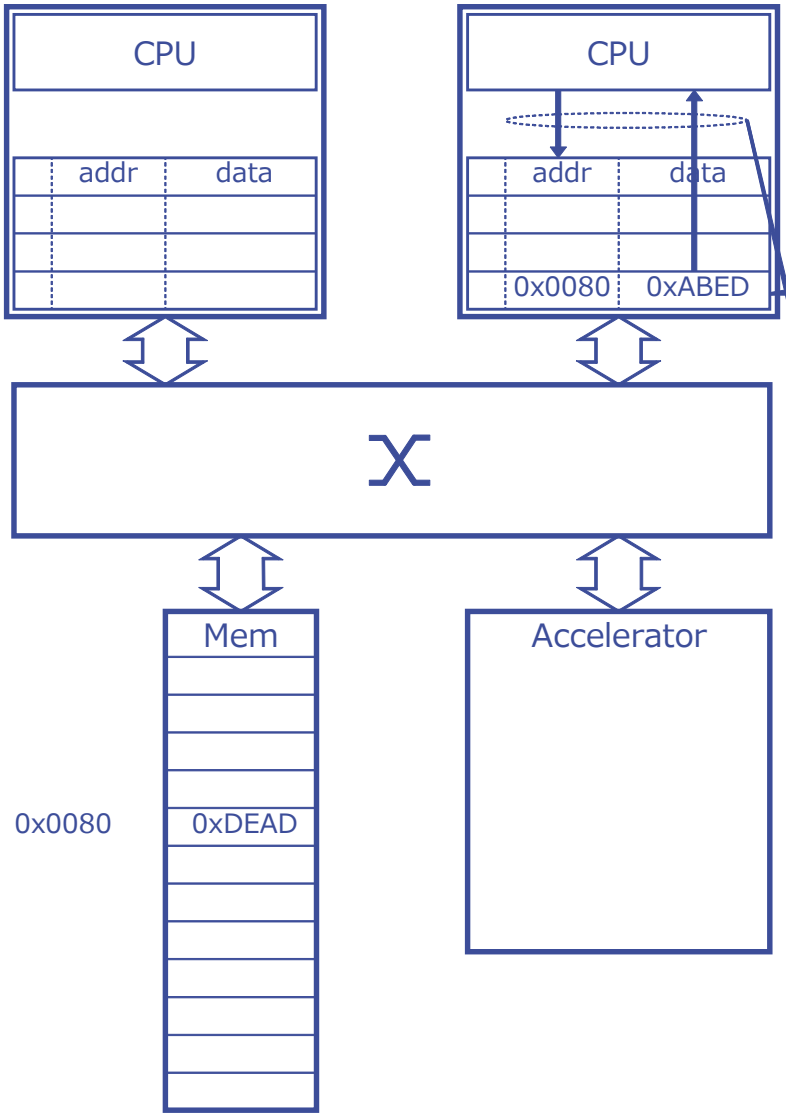
Cache Coherency でトラブルが発生するケース 2



Cache Coherency でトラブルが発生するケース 2



Cache Coherency でトラブルが発生するケース 2



1. CPU が Addr=0x0080 をリード
2. Cache に無いので Mem にリード
3. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ
4. Accelerator が Addr=0x0080 に Data=0xDEAD をライト
5. CPU が Addr=0x0080 をリード

Data Cache に Addr=0x0080 のデータ(0xABED)が残っているので、CPU は data=0xABED と誤った値をリード（Mem にはリードしない)

Cache Coherency トラブルの解決方法

・キャッシュを使わない

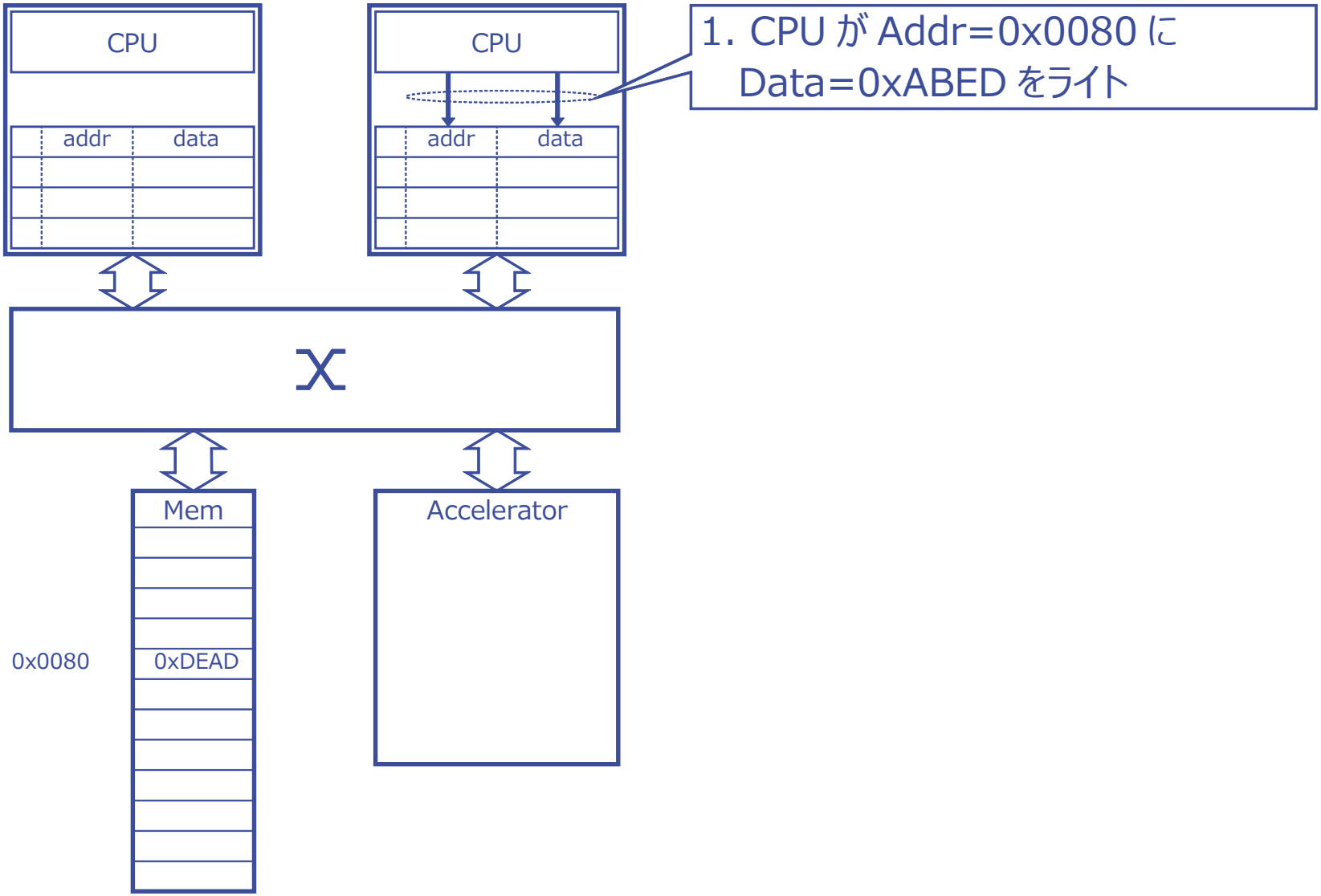
・ソフトウェアによる解決方法

　　ソフトウェアで Cache Flush/Invalidiate する

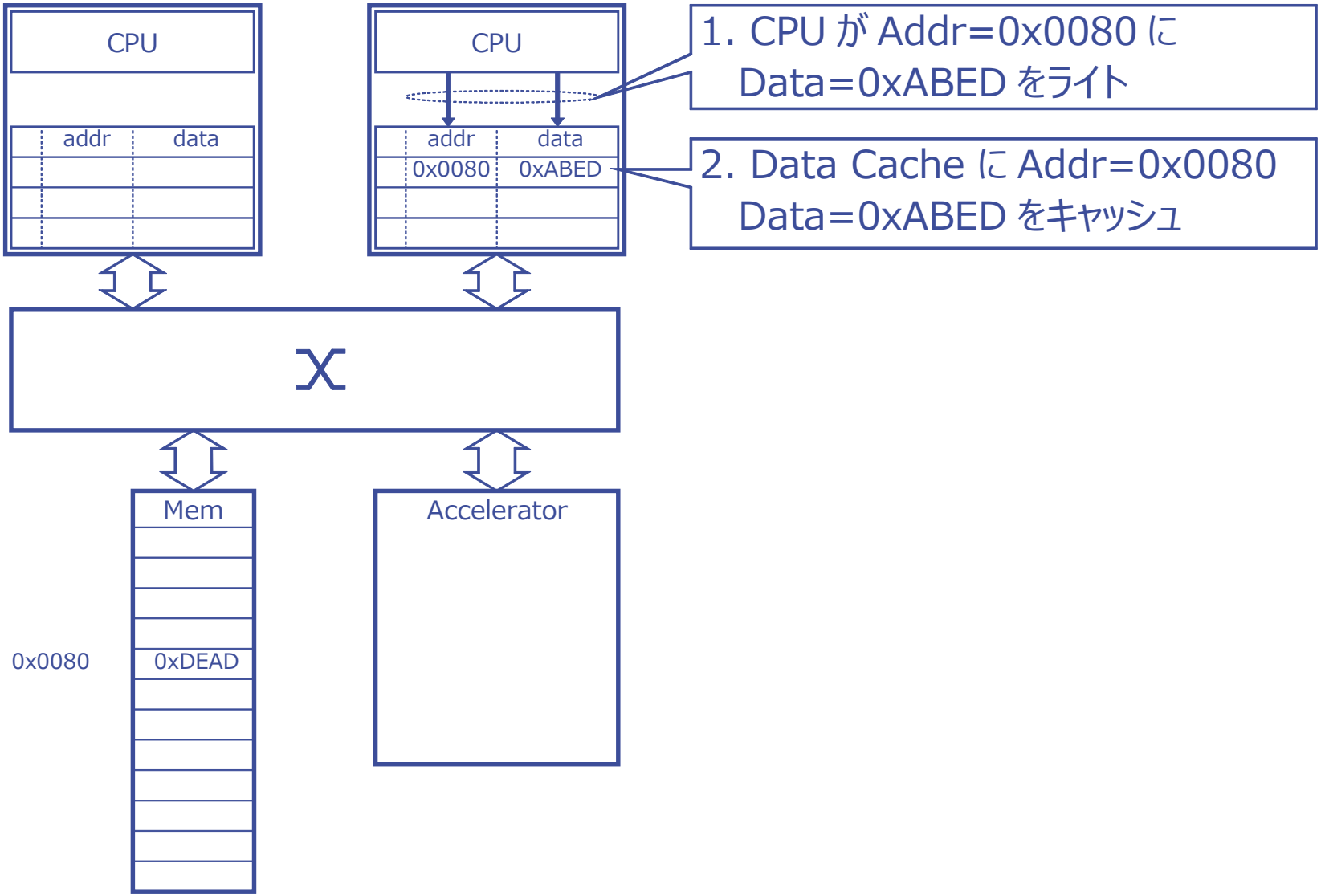
・ハードウェアによる解決方法

　　Cache Coherency に対応した Cache と Interconnect

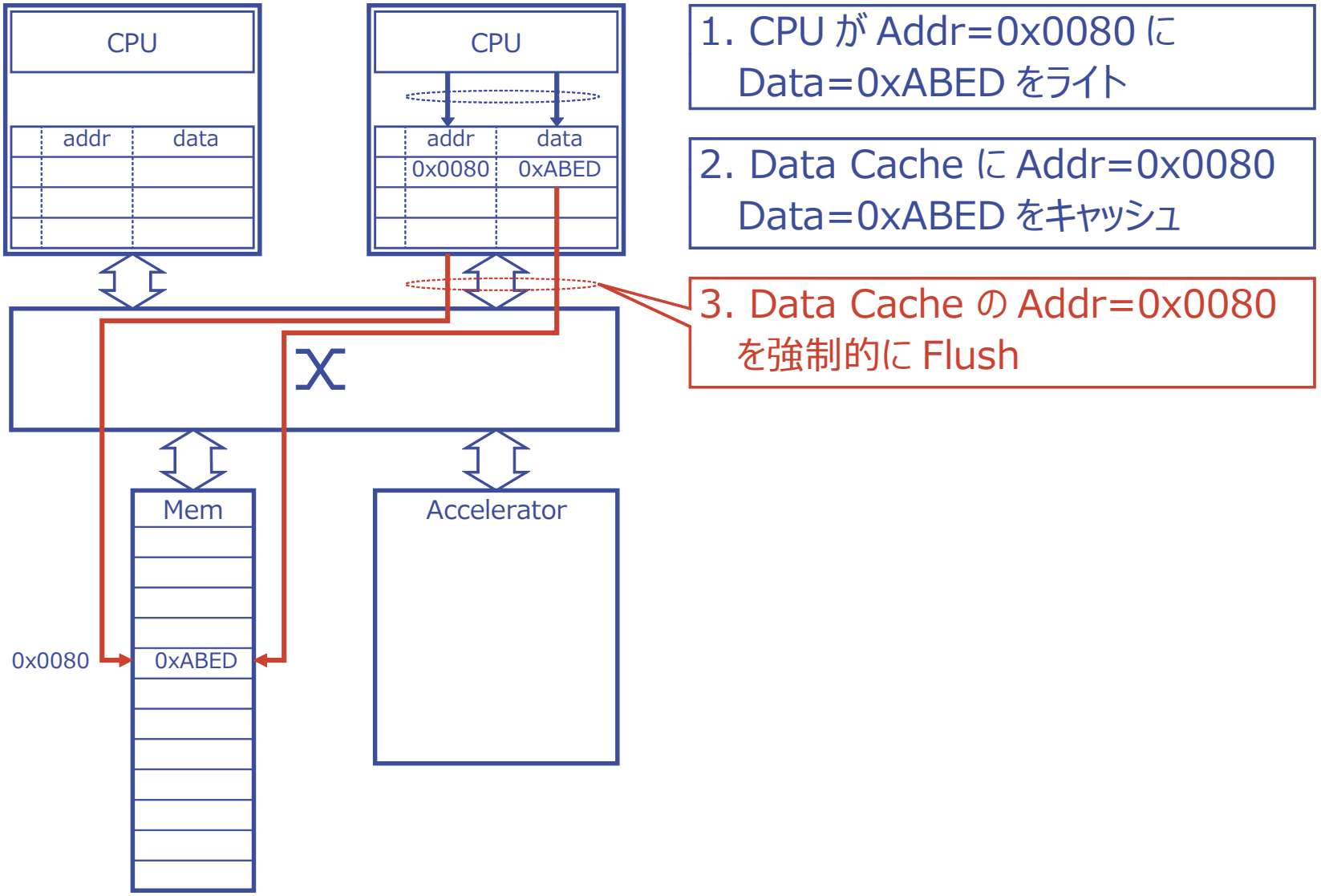
ソフトウェアで Cache を制御してトラブル回避　ケース１



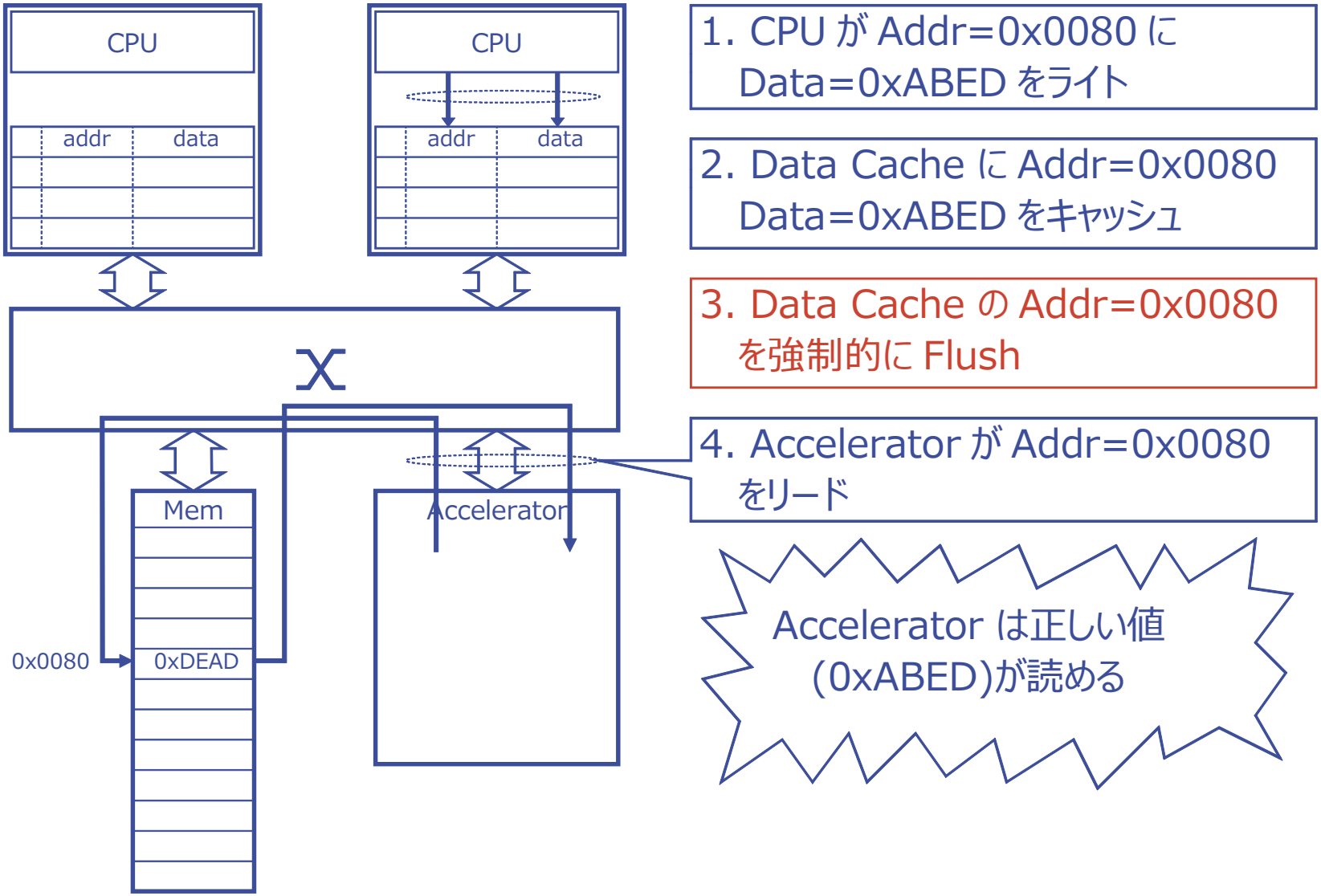
ソフトウェアで Cache を制御してトラブル回避　ケース１



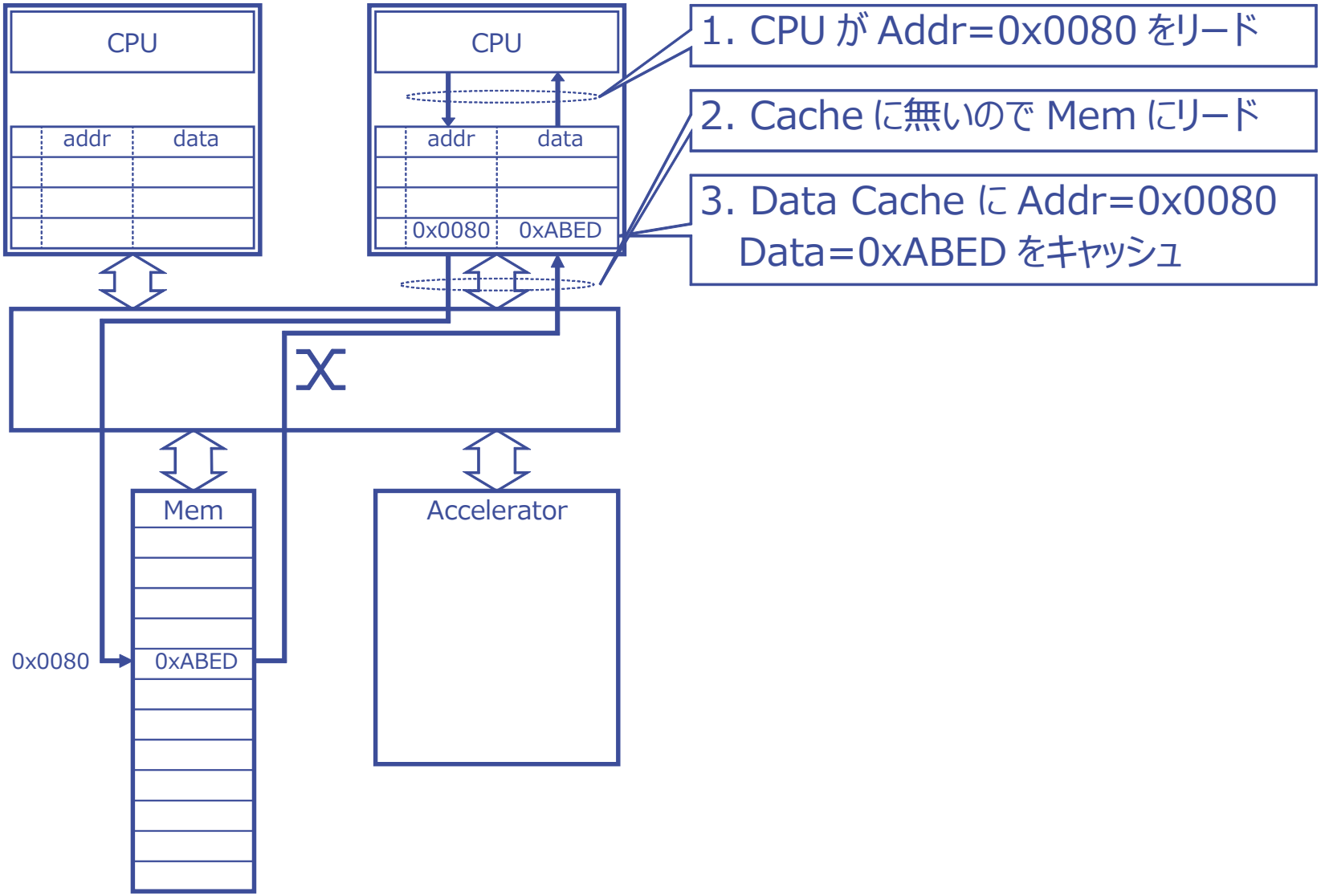
ソフトウェアで Cache を制御してトラブル回避　ケース 1



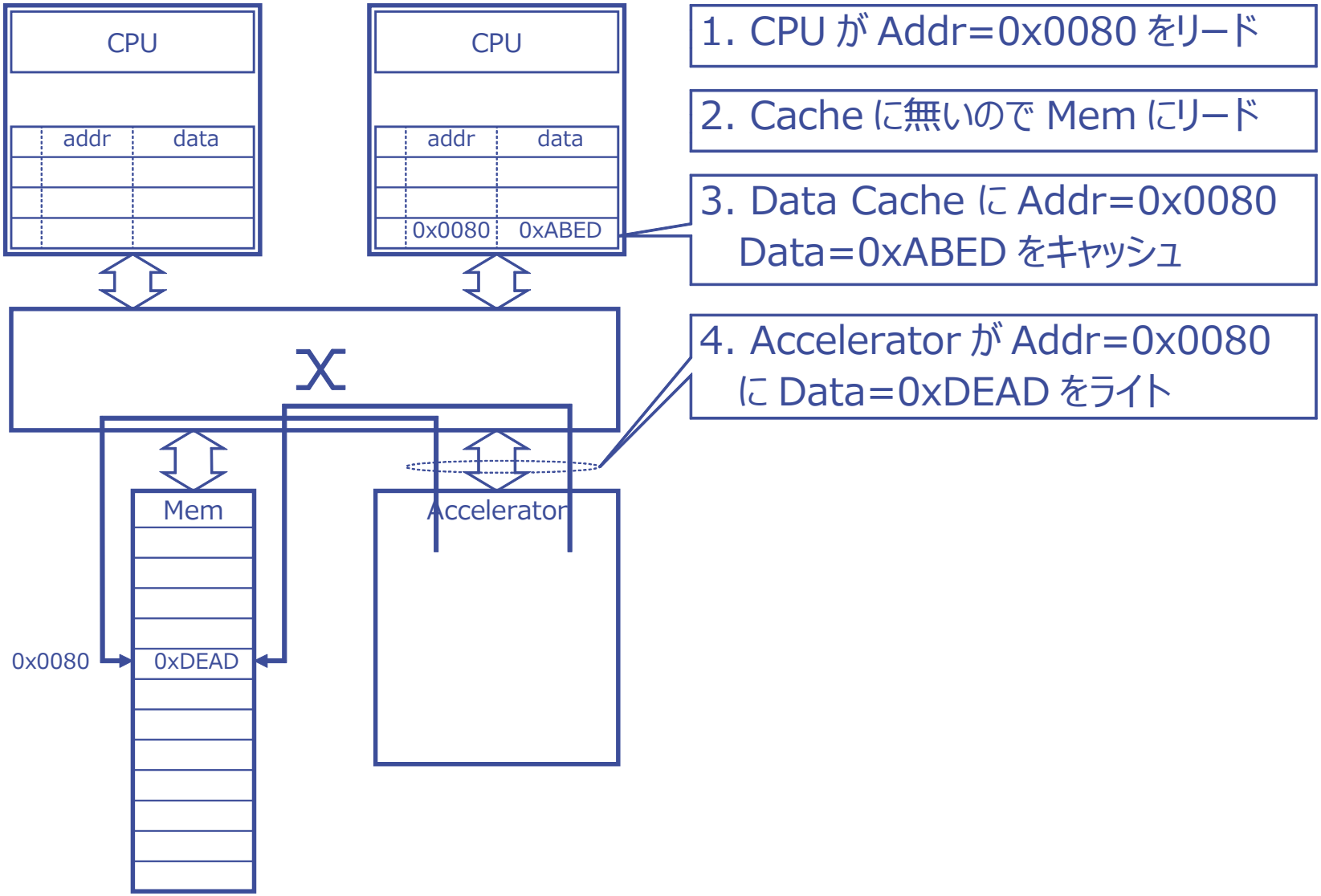
ソフトウェアで Cache を制御してトラブル回避　ケース１



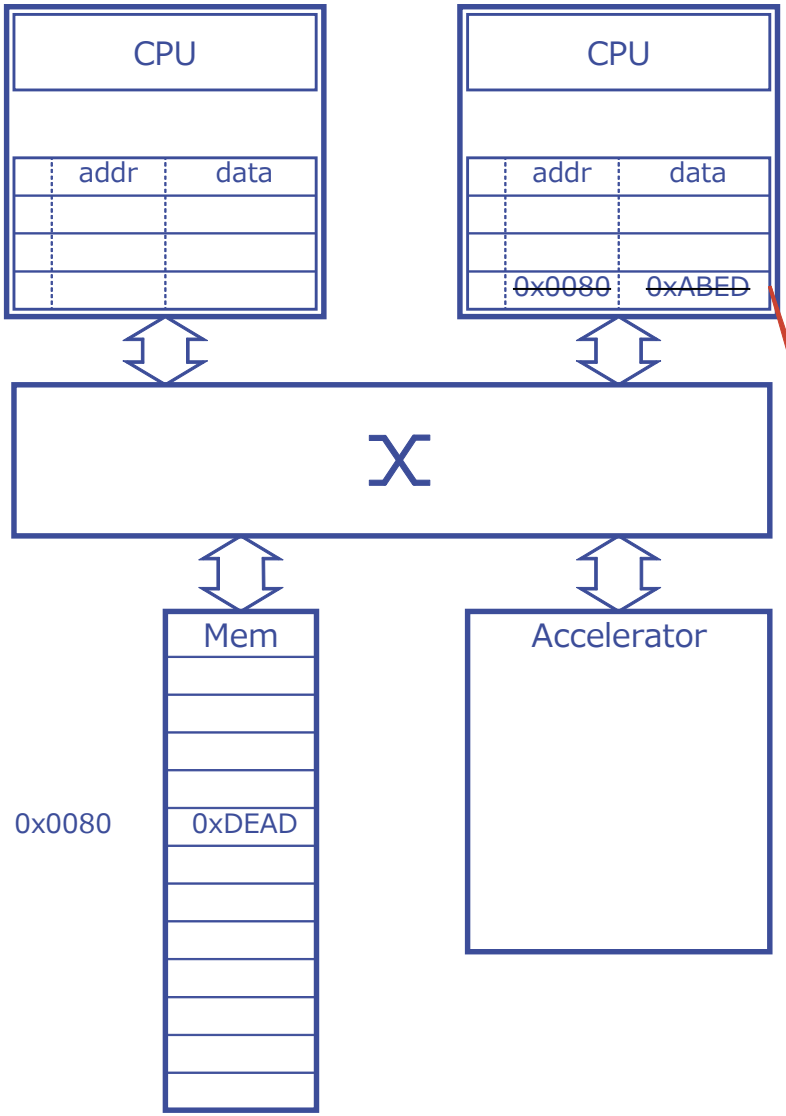
ソフトウェアで Cache を制御してトラブル回避　ケース２



ソフトウェアで Cache を制御してトラブル回避　ケース２

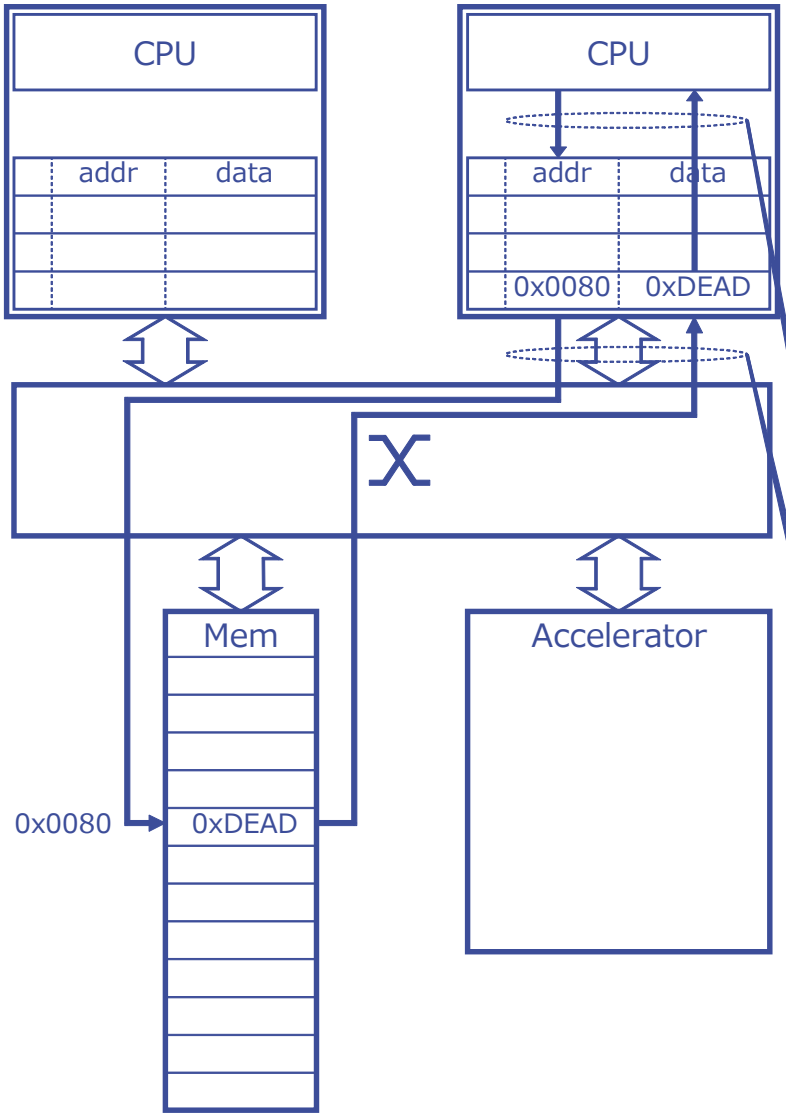


ソフトウェアで Cache を制御してトラブル回避　ケース２



1. CPU が Addr=0x0080 をリード
2. Cache に無いので Mem にリード
3. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ
4. Accelerator が Addr=0x0080 に Data=0xDEAD をライト
5. Data Cache の Addr=0x0080 を Invalidiate(無効化)

ソフトウェアで Cache を制御してトラブル回避　ケース２

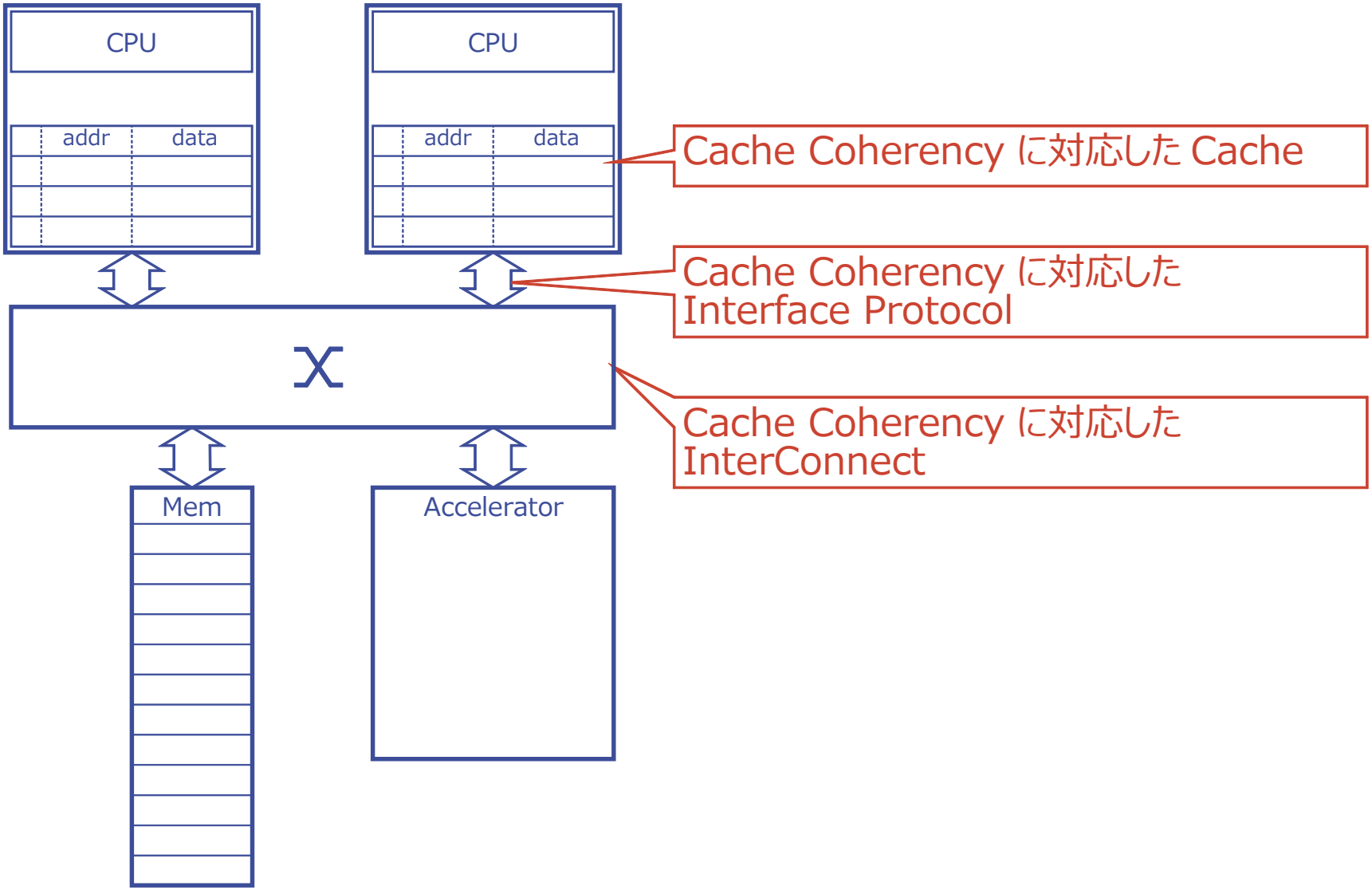


1. CPU が Addr=0x0080 をリード
2. Cache に無いので Mem にリード
3. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ
4. Accelerator が Addr=0x0080 に Data=0xDEAD をライト
5. Data Cache の Addr=0x0080 を Invalidiate(無効化)
6. CPU が Addr=0x0080 をリード
7. Cache に無いので Mem にリード

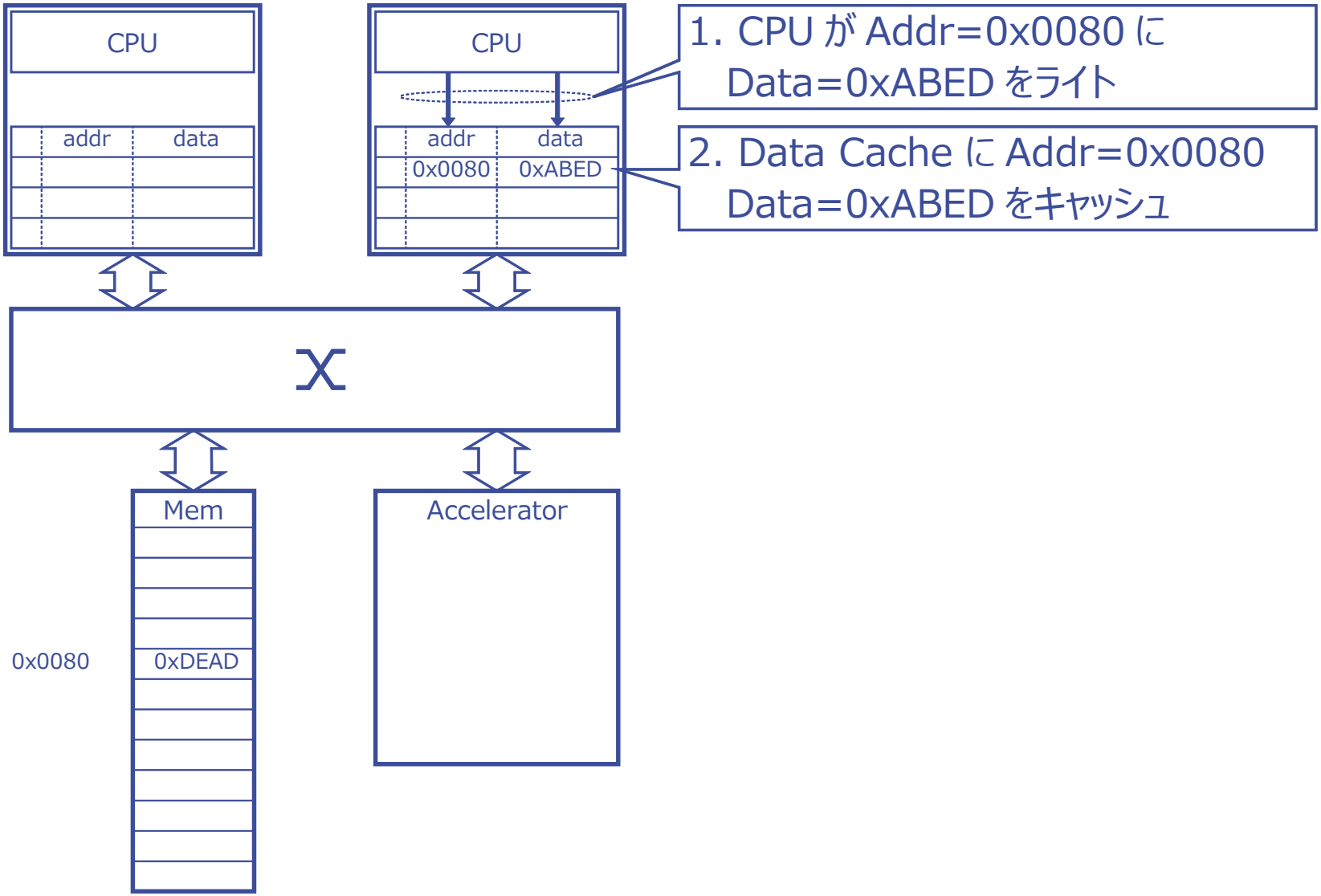
ソフトウェアで Cache を制御する方法の問題点

- 面倒くせ～よ
- 相手がリード/ライトするタイミングが判ってないと無理
  - CPU が DMA や Accelerator を制御する場合は可能だけど、
    マルチプロセッサ間では難しい
- 時間がかかる
  - キャッシュの操作は意外と時間がかかる
  - しかもクリティカルセクション(他の処理ができない)

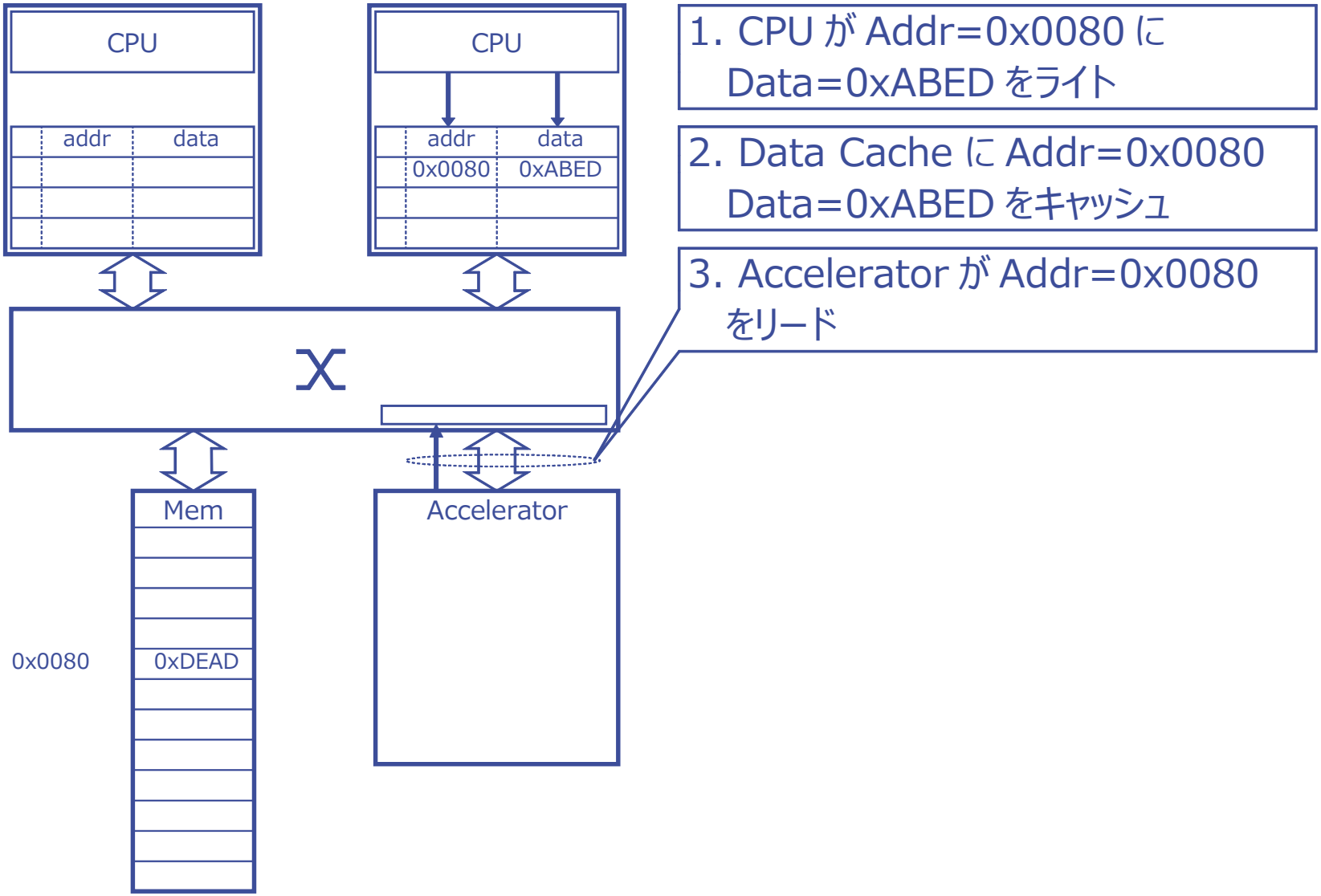
ハードウェアで Cache Coherency - 用意するもの



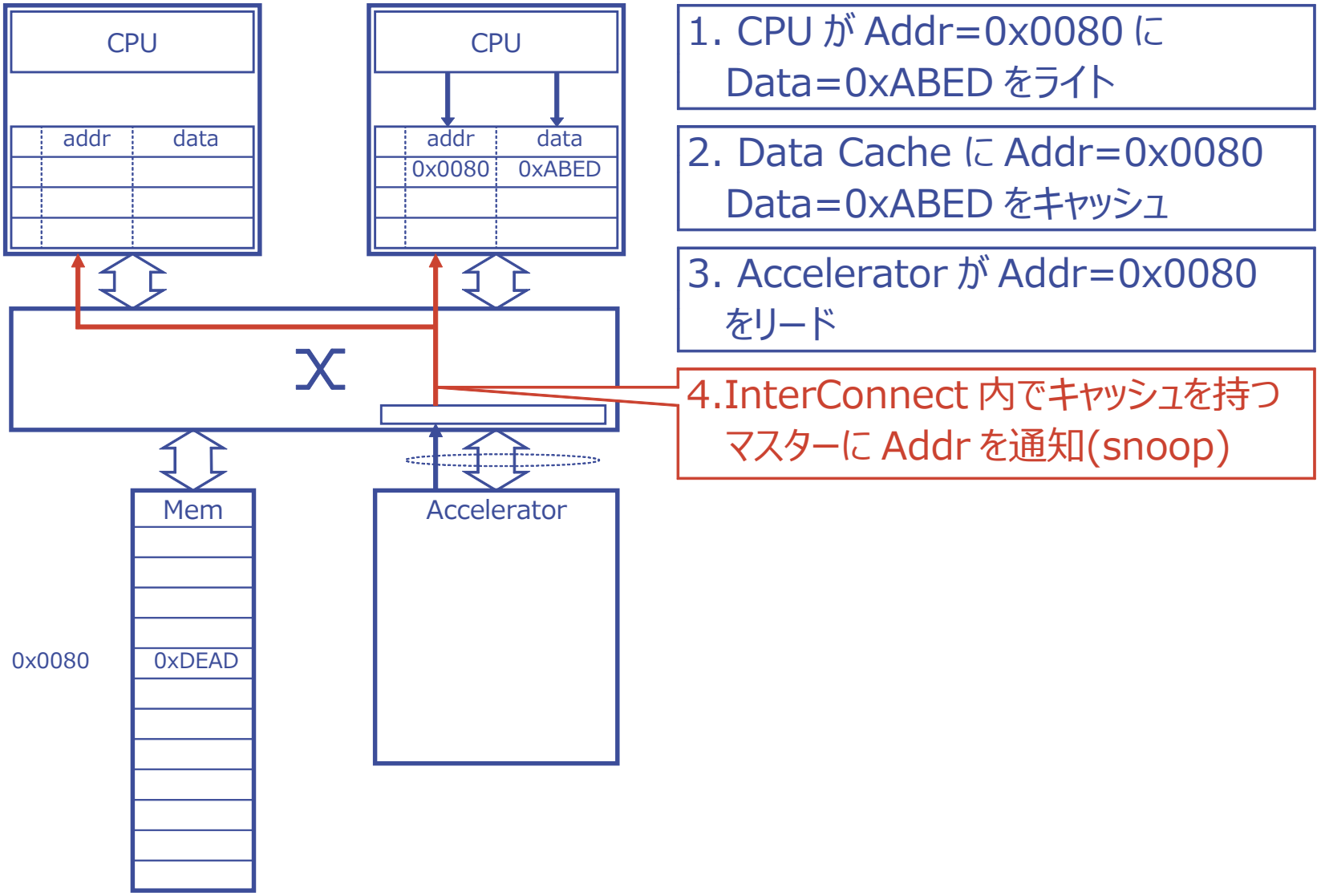
ハードウェアで Cache Coherency ケース 1



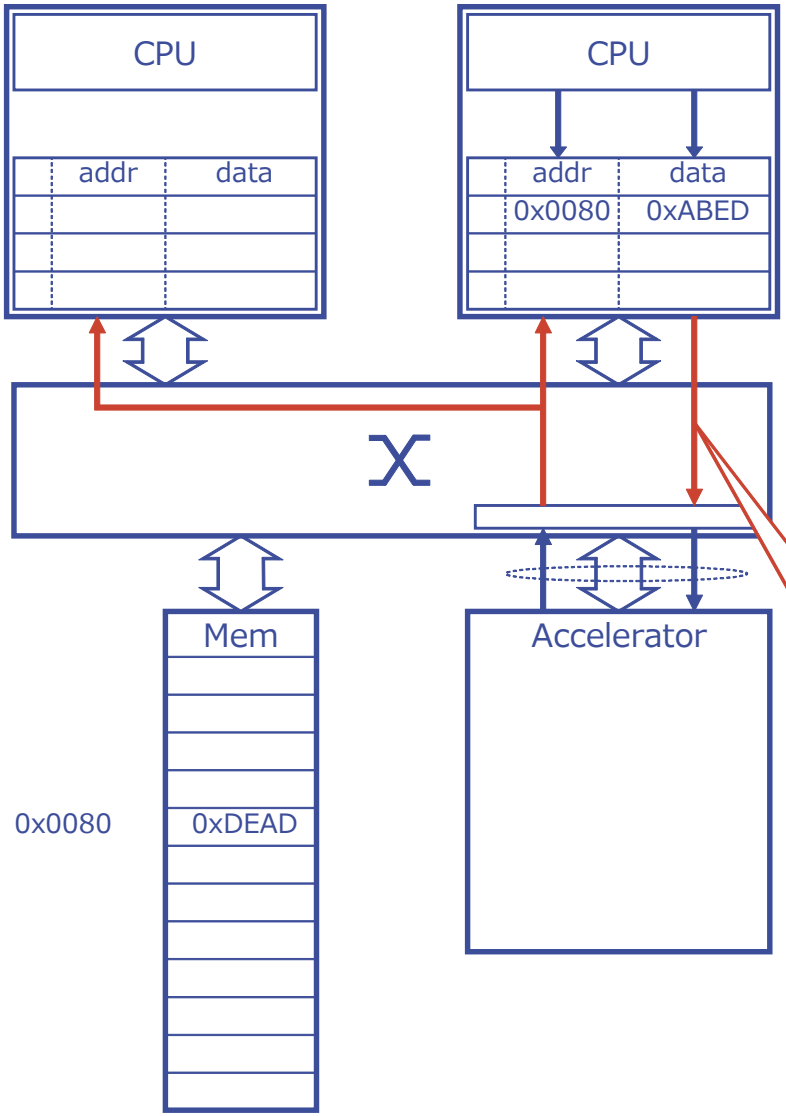
ハードウェアで Cache Coherency ケース 1



ハードウェアで Cache Coherency ケース 1



ハードウェアで Cache Coherency ケース 1



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

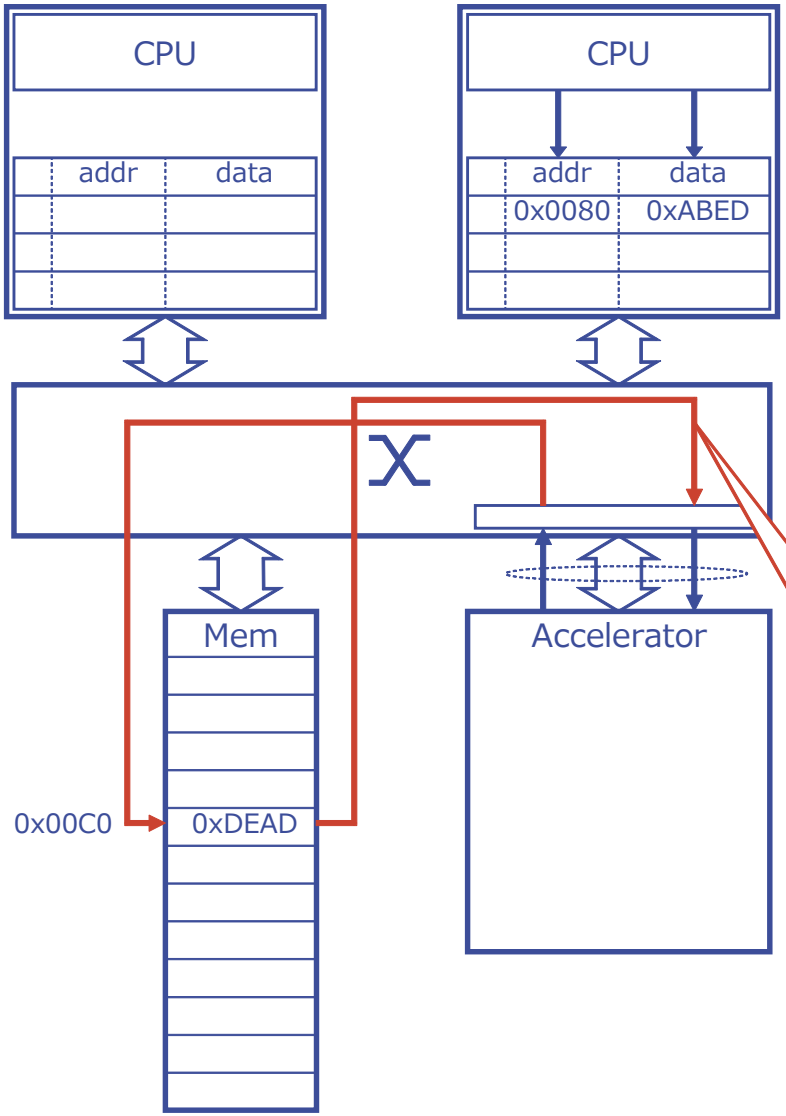
2. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ

3. Accelerator 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がデータを出力

ハードウェアで Cache Coherency ケース 1



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

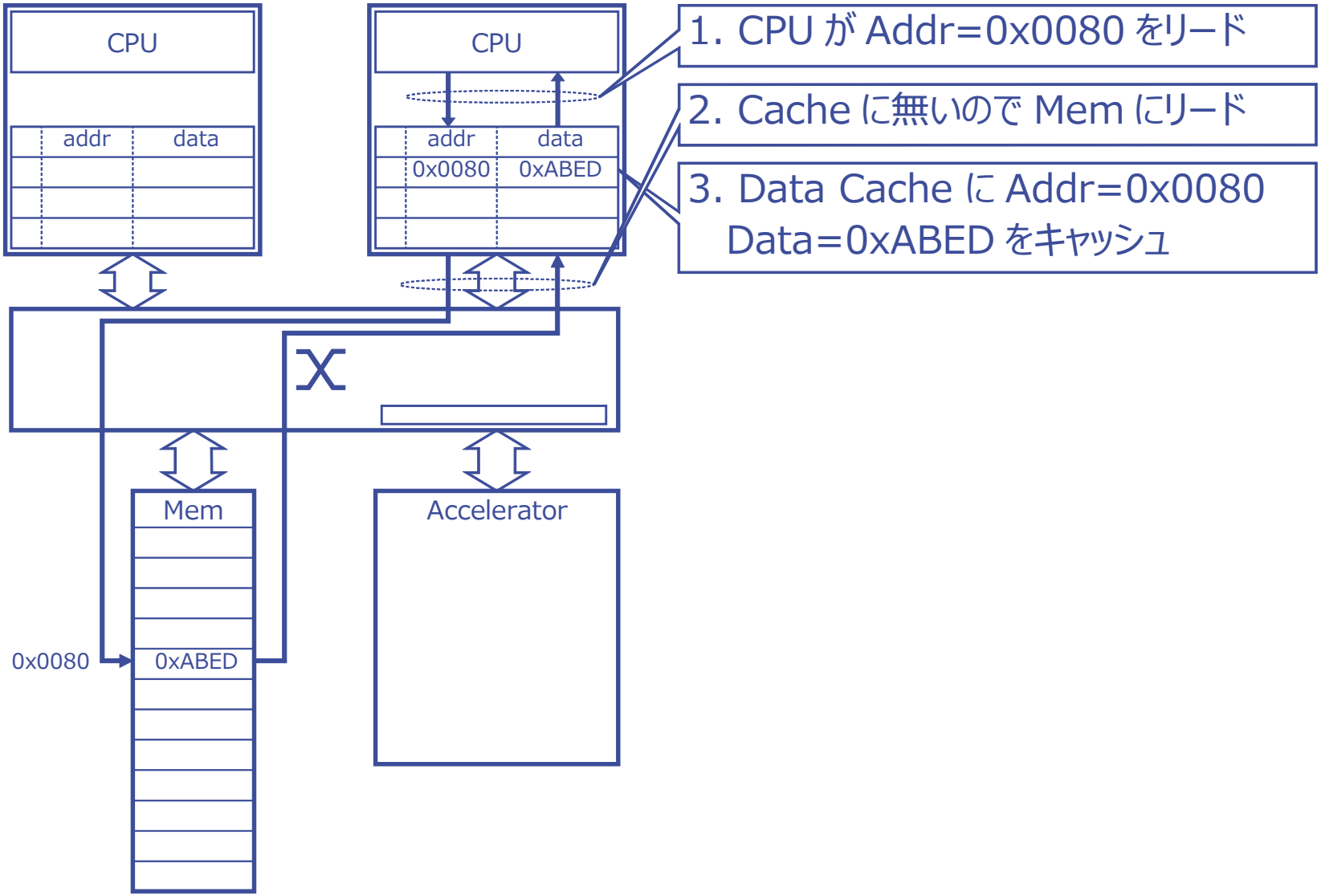
2. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ

3. Accelerator が Addr=0x00C0 をリード

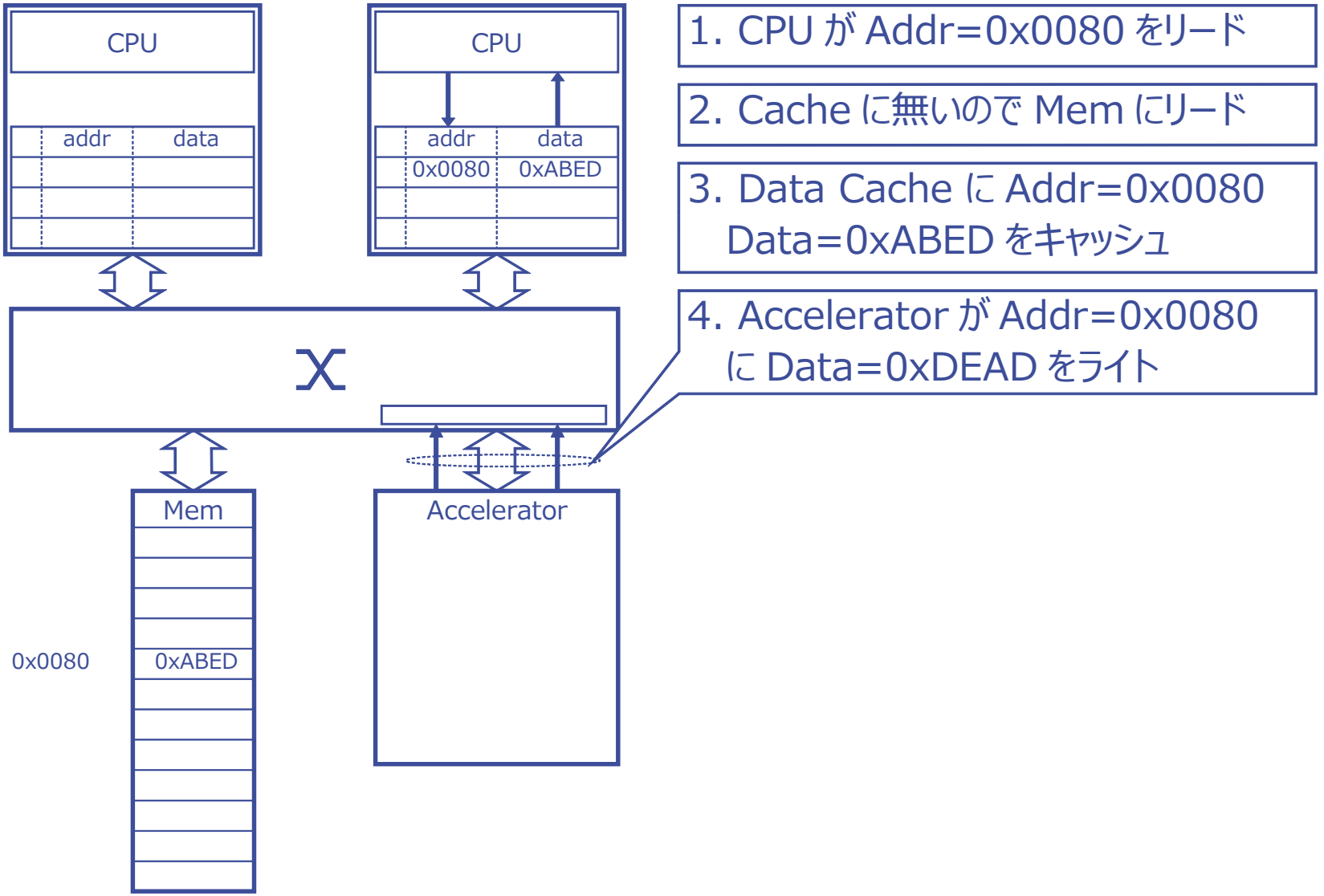
4.InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)

5.該当する Addr(0x00C0)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 Mem から読む

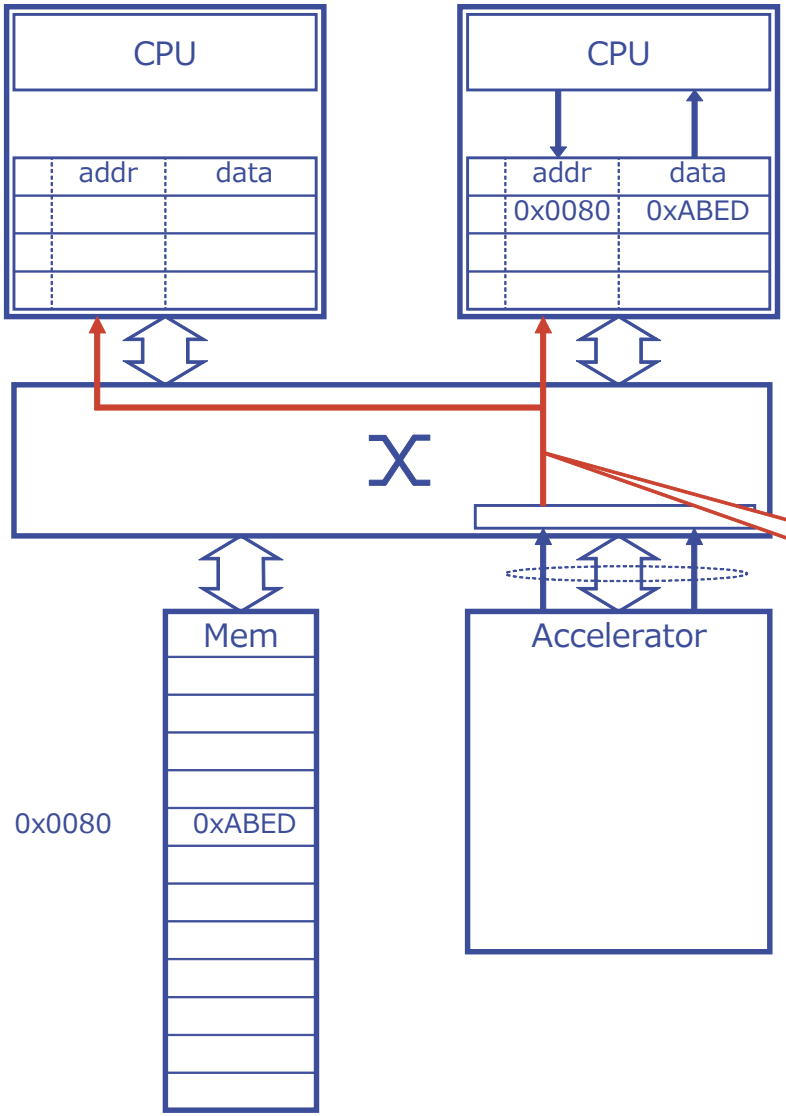
ハードウェアで Cache Coherency ケース 2



ハードウェアで Cache Coherency ケース 2

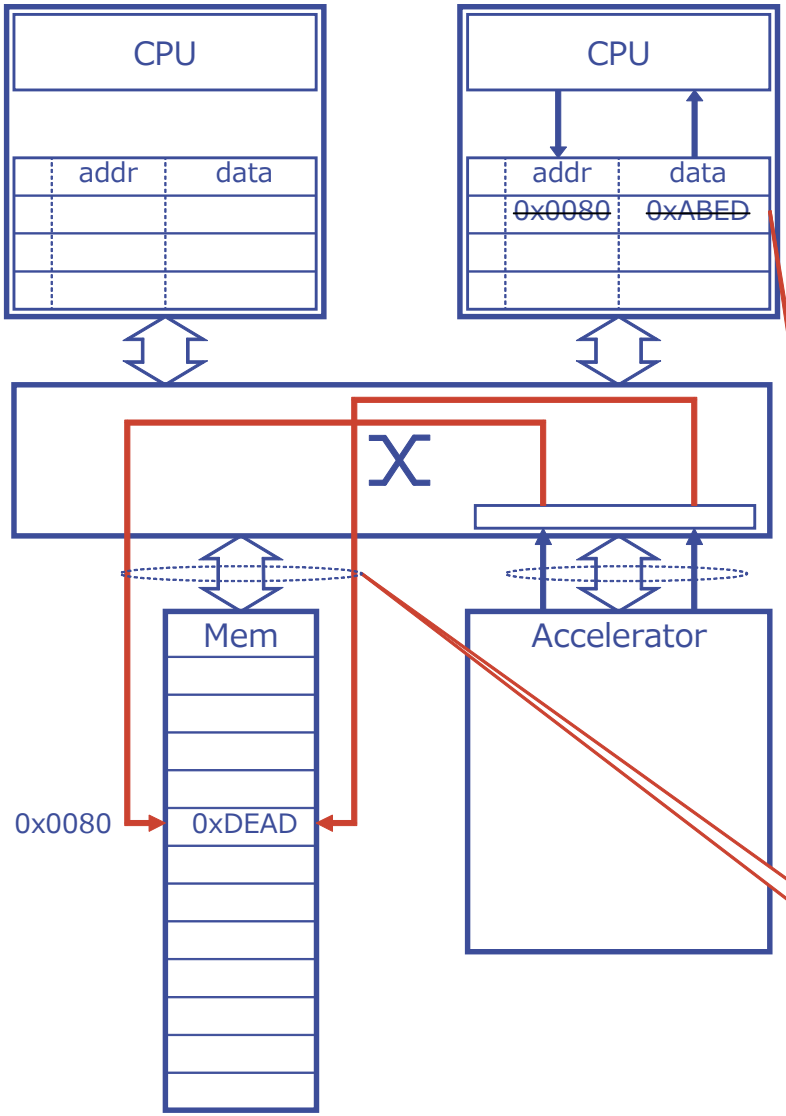


ハードウェアで Cache Coherency ケース 2



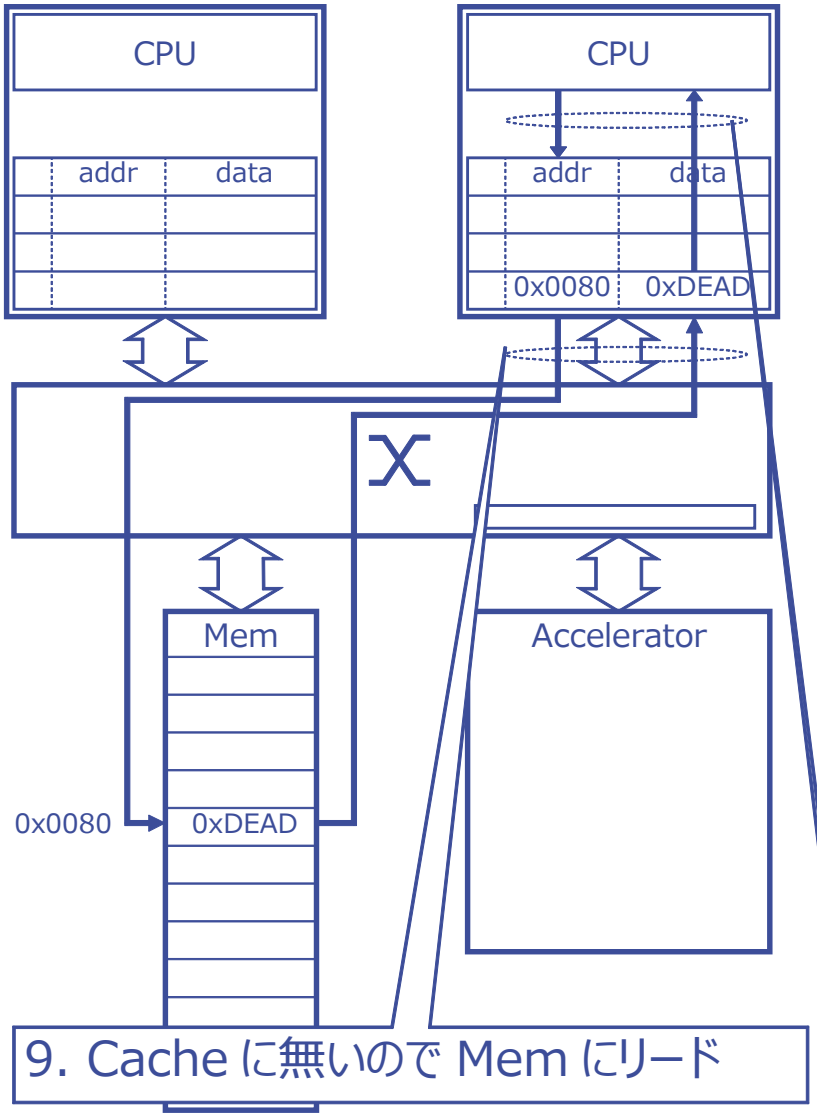
1. CPU が Addr=0x0080 をリード
2. Cache に無いので Mem にリード
3. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ
4. Accelerator が Addr=0x0080 に Data=0xDEAD をライト
5. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)

ハードウェアで Cache Coherency ケース 2 (キャッシュラインサイズの書き込みの場合)



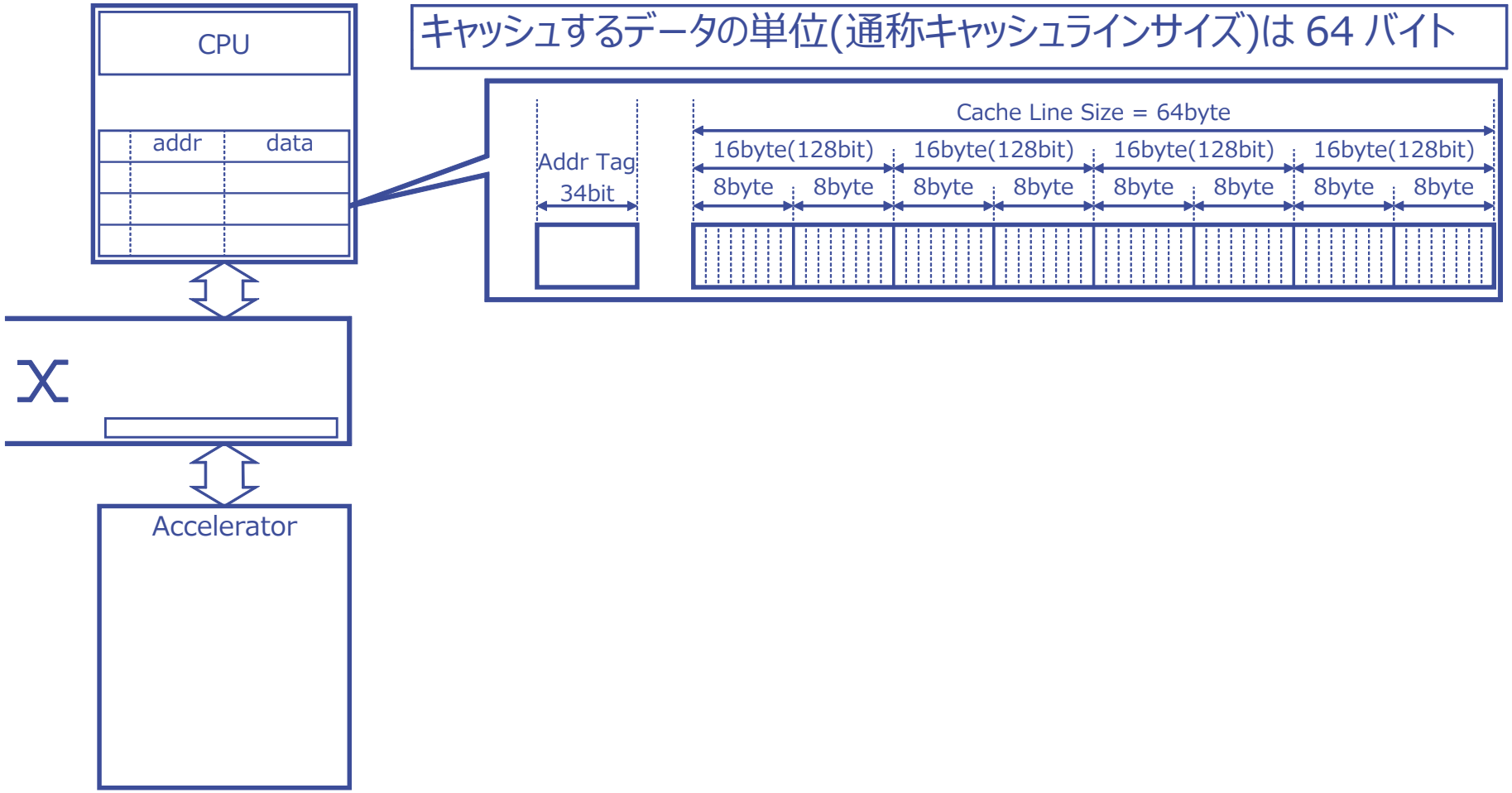
1. CPU が Addr=0x0080 をリード
2. Cache に無いので Mem にリード
3. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ
4. Accelerator が Addr=0x0080 に Data=0xDEAD をライト
5. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)
6. 該当する Addr(0x0080)のデータをキャッシュしているマスターはキャッシュを無効化(Invalidiate)
7. Mem の Addr=0x0080 に Data=0xDEAD をライト

ハードウェアで Cache Coherency ケース 2 （キャッシュラインサイズの書き込みの場合）

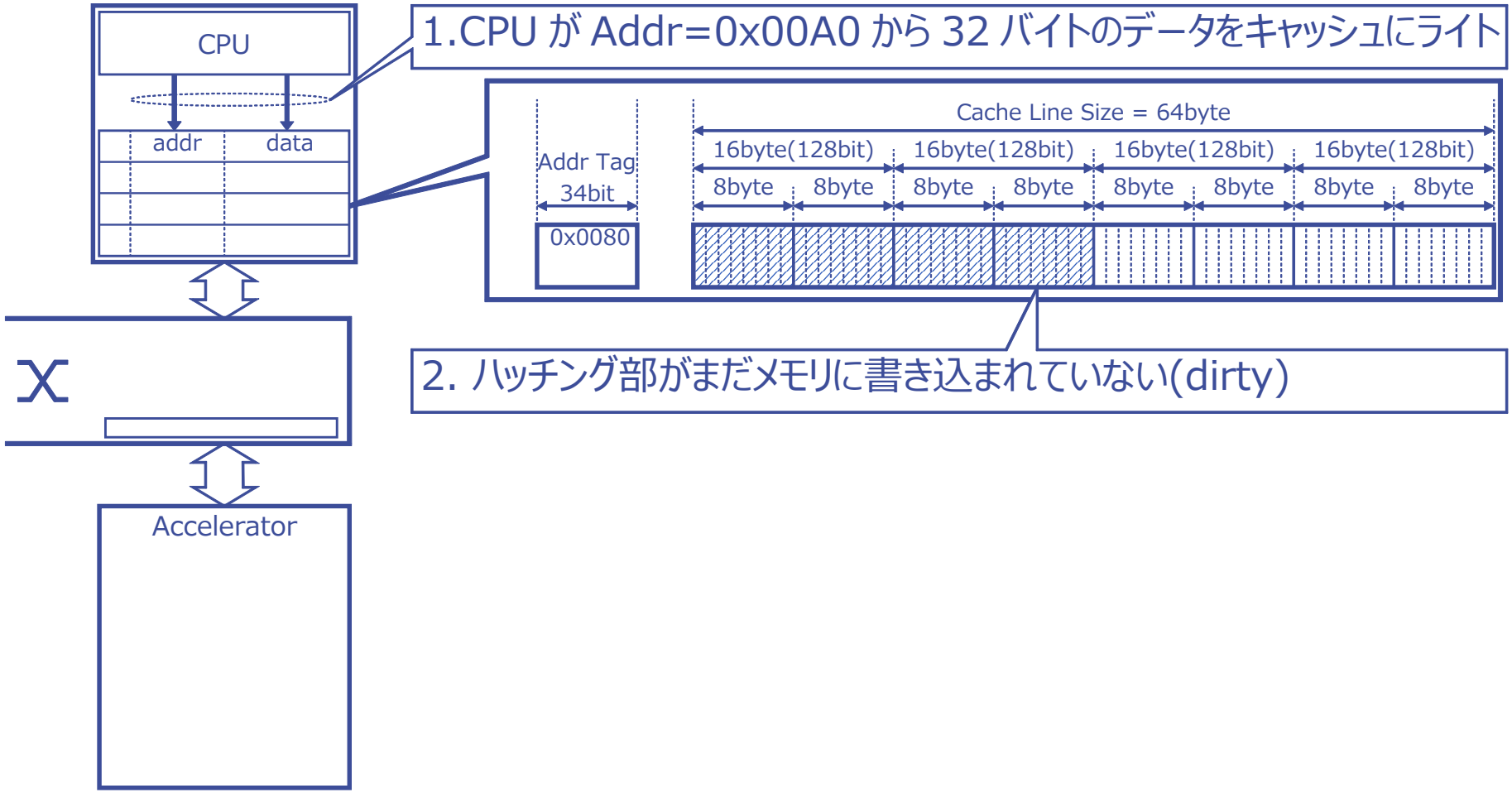


1. CPU が Addr=0x0080 をリード
2. Cache に無いので Mem にリード
3. Data Cache に Addr=0x0080 Data=0xABED をキャッシュ
4. Accelerator が Addr=0x0080 に Data=0xDEAD をライト
5. InterConnect 内でキャッシュを持つマスターに Addr を通知(snoop)
6. 該当する Addr(0x0080)のデータをキャッシュしているマスターはキャッシュを無効化(Invalidiate)
7. Mem の Addr=0x0080 に Data=0xDEAD をライト
8. CPU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

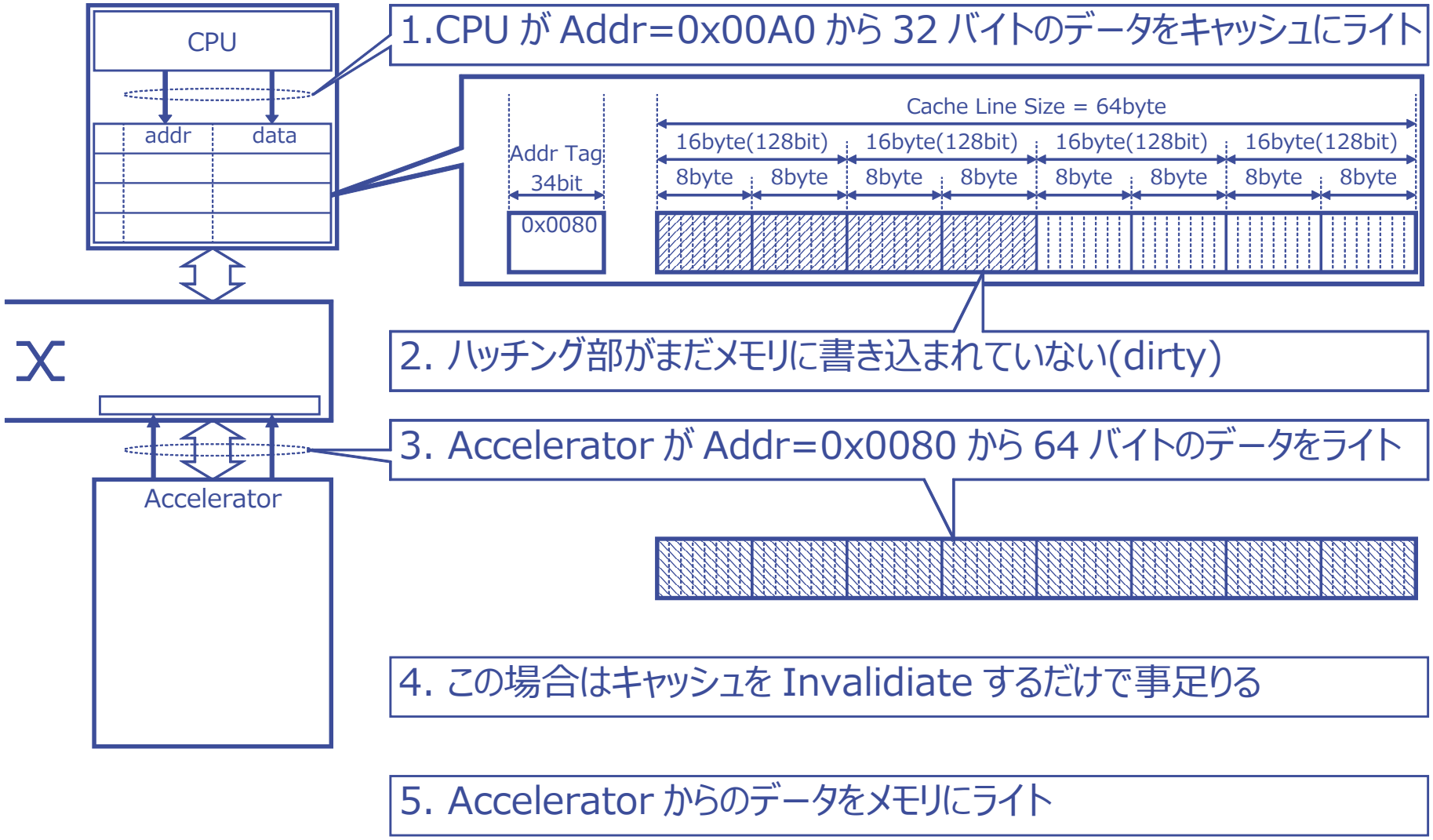
補足　キャッシュラインのアライメント問題



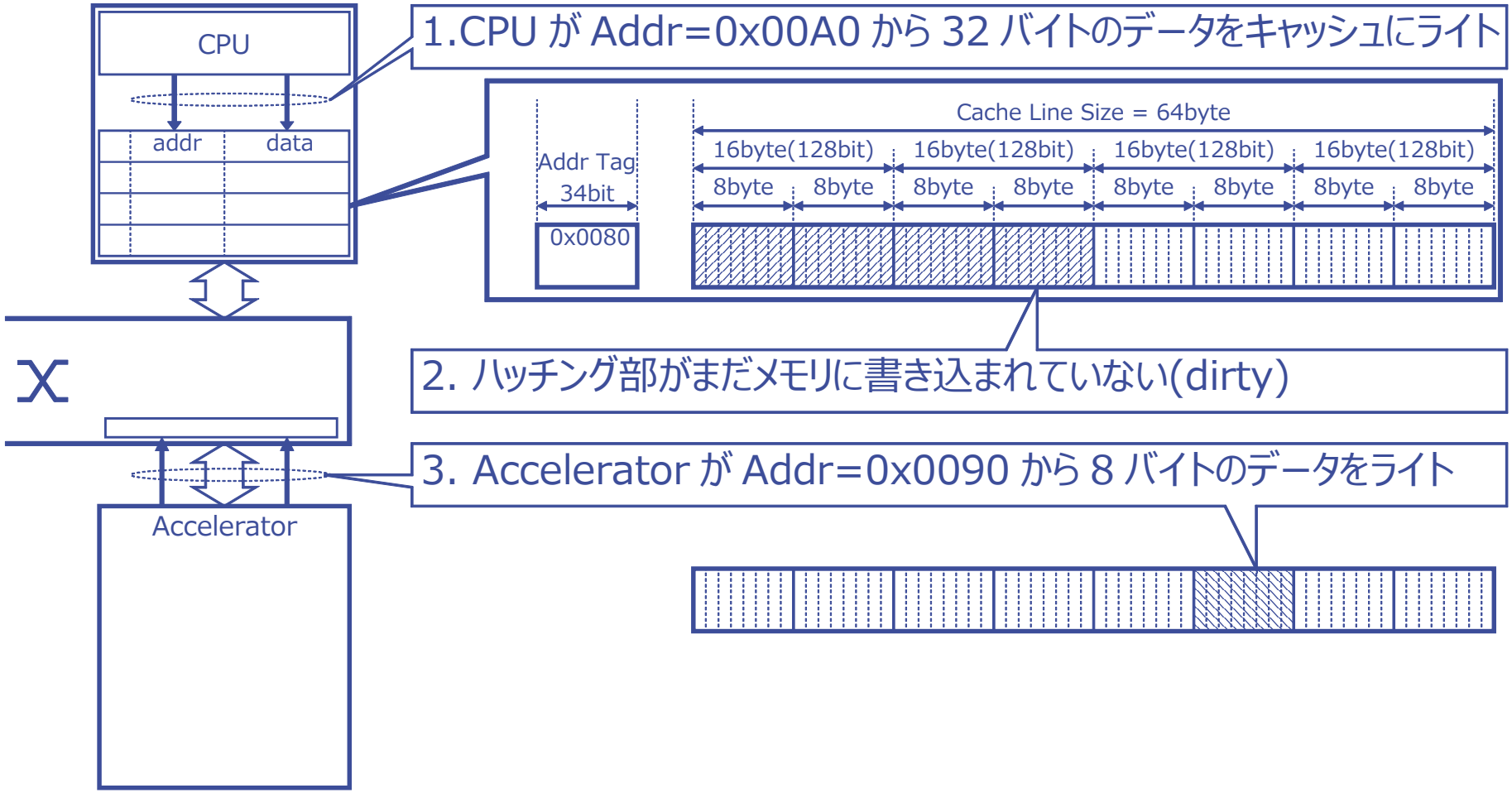
補足　キャッシュラインのアライメント問題



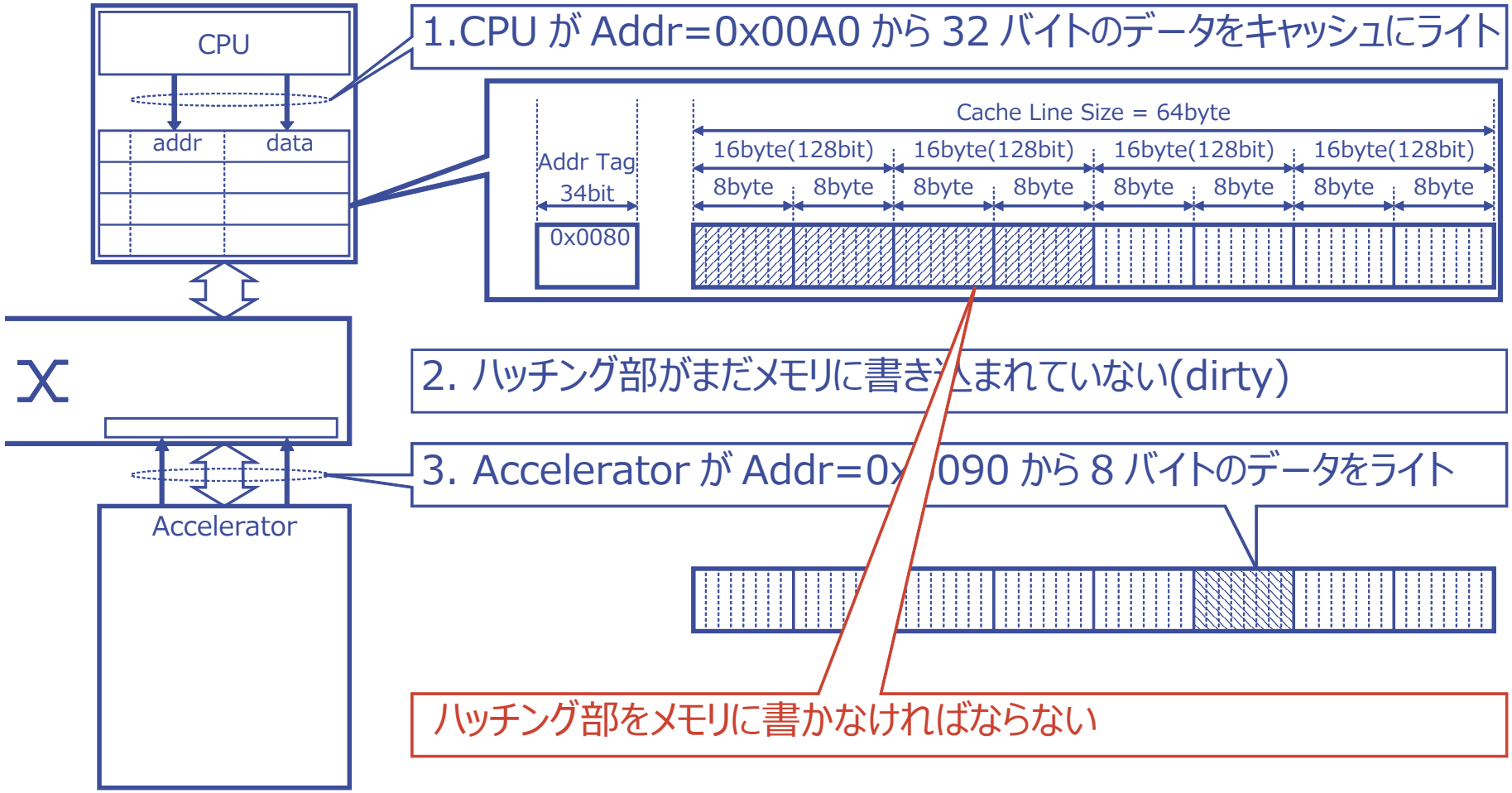
補足　キャッシュラインのアライメント問題



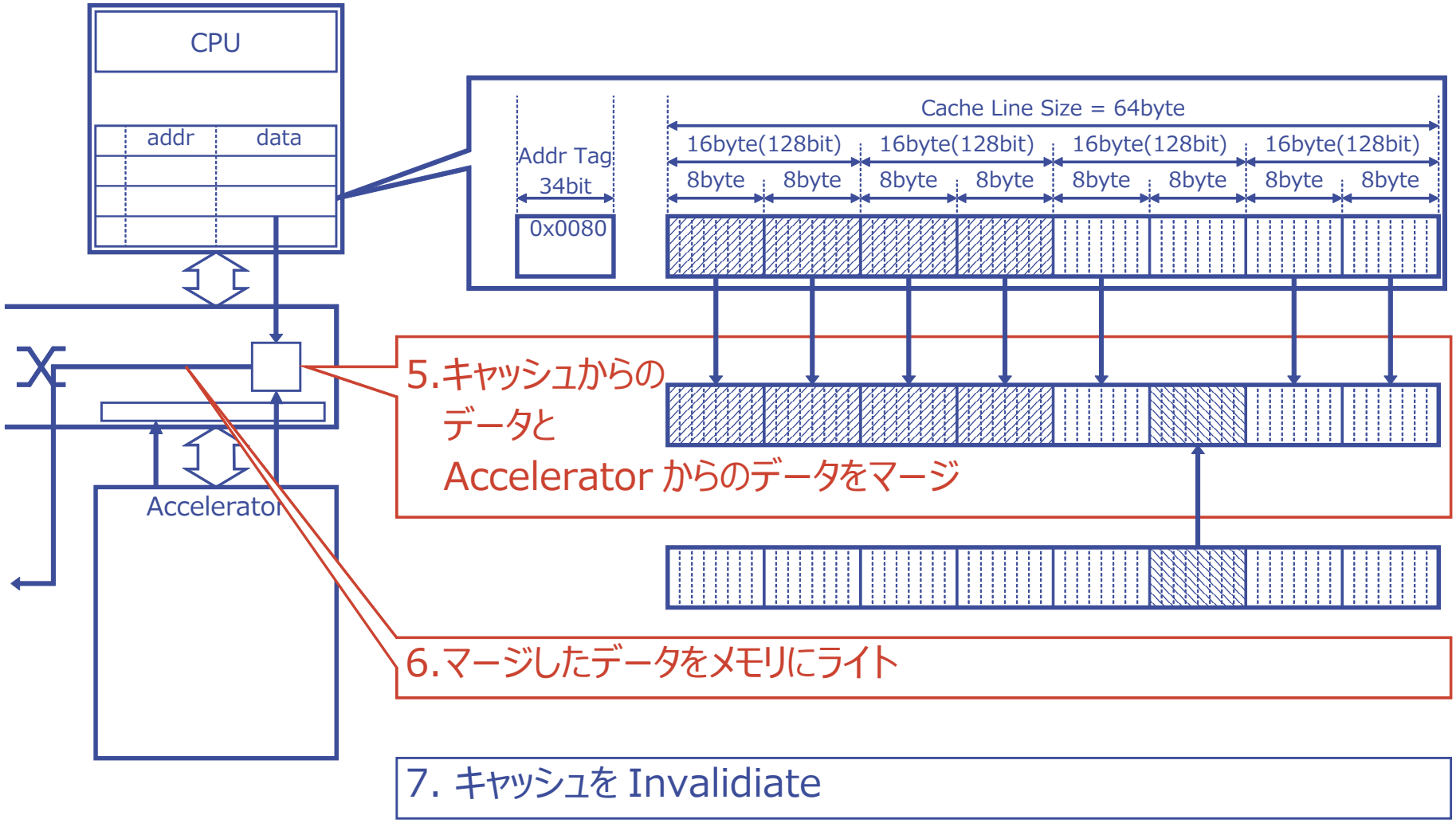
補足　キャッシュラインのアライメント問題



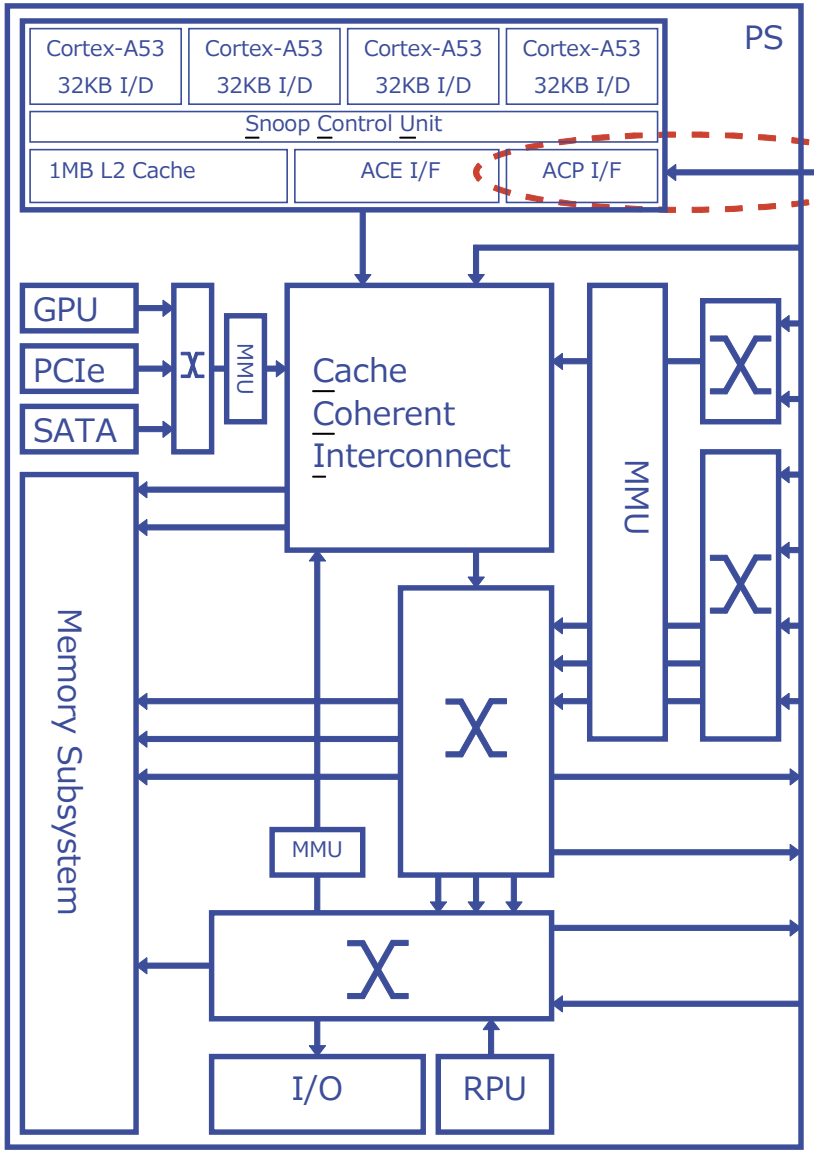
補足　キャッシュラインのアライメント問題



補足　キャッシュラインのアライメント問題



ACP(Accelerator Coherency Port)の制約



ACP Limitations

The ACP accepts only the following (cache-line friendly) transactions.

. 64-byte aligned (of 64-byte) read/write INCR transactions. All write-byte strobes must be same (either enabled or disabled).

. 16-byte aligned (of 16-byte) read/write INCR transactions. Write-byte strobes can have any value.

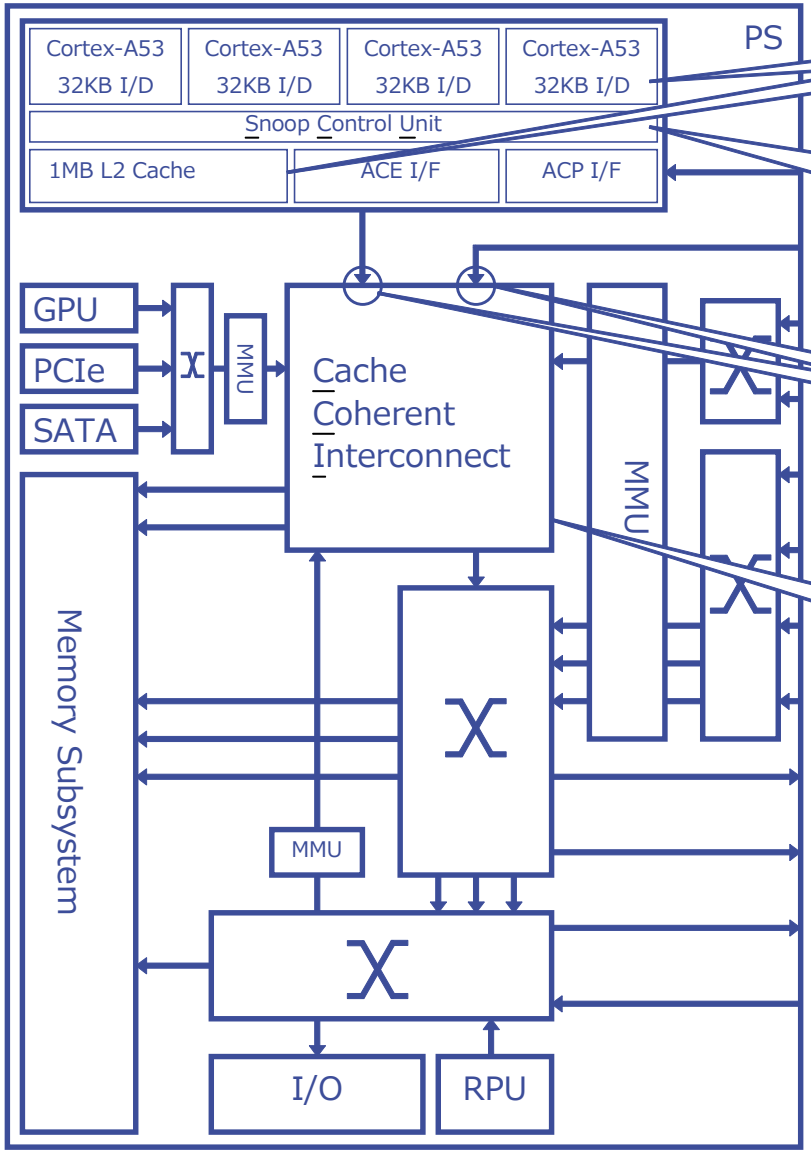
『 Zynq UltraScale+ MPSoC TRM UG1085 (v1.0) November 24,2015』826

ACP Coherency

The PL masters can also snoop APU caches through the APU’ s accelerator coherency port (ACP). The ACP accesses can be used to (read or write) allocate into L2 cache. However, the ACP only supports aligned 64-byte and 16-byte accesses. All other accesses get a SLVERR response.

『 Zynq UltraScale+ MPSoC TRM UG1085 (v1.0) November 24,2015』228

ZynqMP の Cache Coherency Architecture



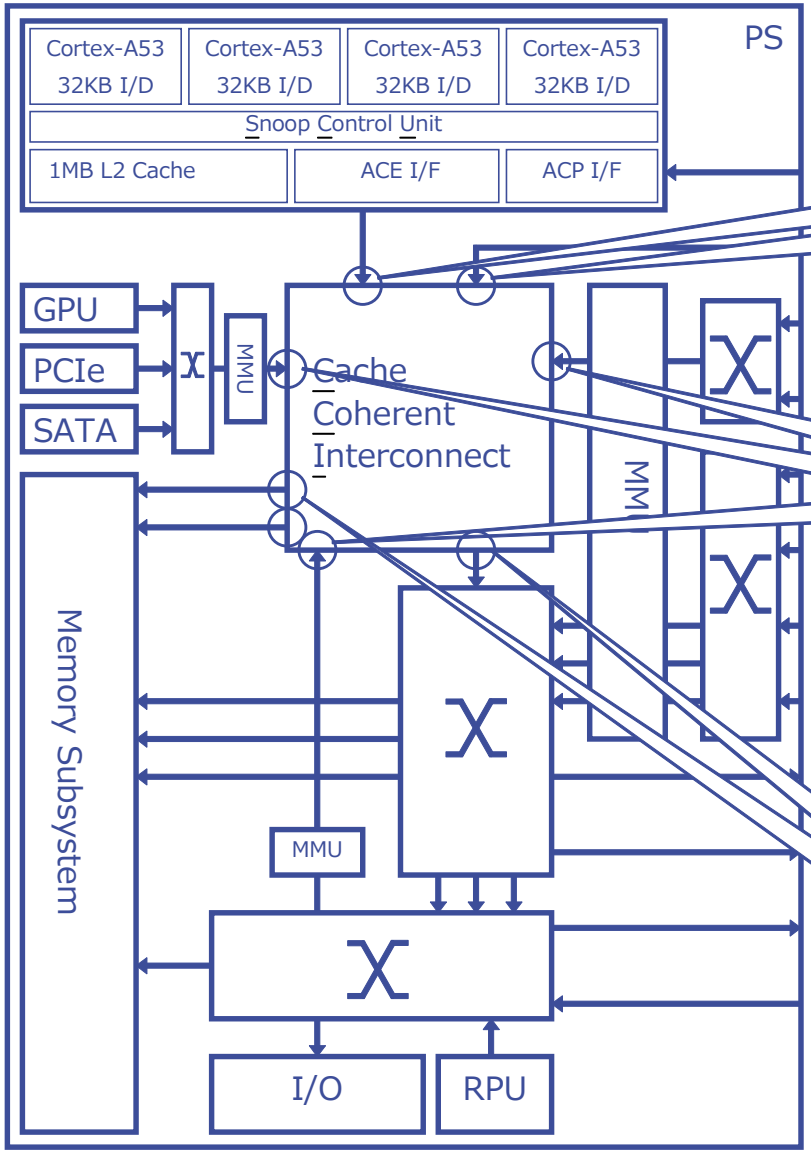
Cache Coherency に対応した Cache

Cache Coherency に対応した
InterConnect
SCU(Snoop Control Unit)

Cache Coherency に対応した
Interface Protocol
ACE(AXI Coherency Extension)

Cache Coherency に対応した
InterConnect
CCI(Cache Coherency Interconnect)

ZynqMP の Cache Coherency Interconnect I/F



Full Coherency (ACE Slave Port)
- Cache を持っているマスターとの I/F
- Cache Snoop Channel 有り
- APU から 1 本
- PL の S_AXI_ACE_FPD から 1 本

I/O Coherency (ACE-Lite Slave Port)
- Cache を持っていないマスターとの I/F
- Cache Snoop Channel 無し
- GPU/PCIe/SATA から 1 本
- PL の S_AXI_HPC[01]_FPD から 1 本
- LPD のスイッチから 1 本

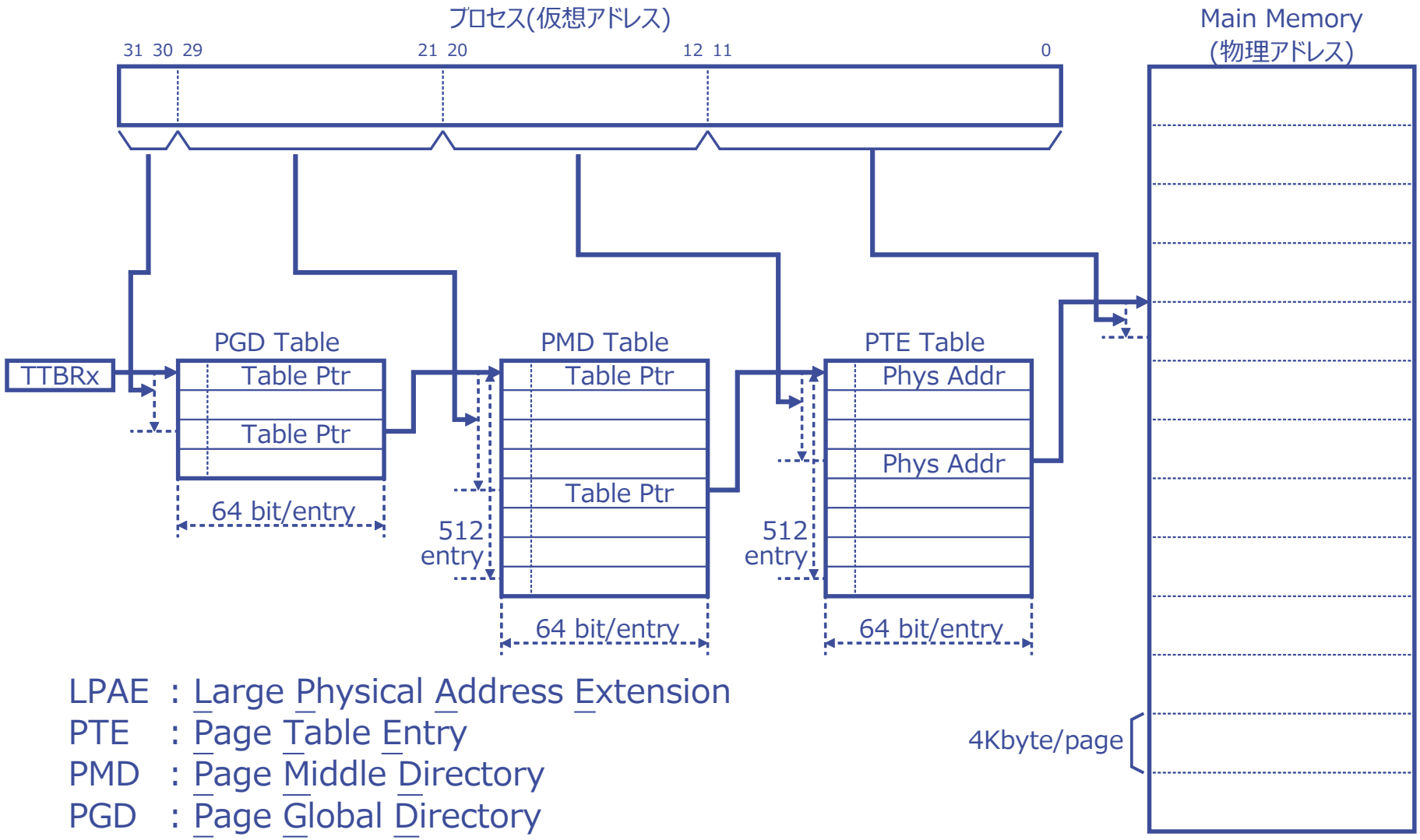
ACE-Lite Master Port
- Memory Subsystem へ 2 本
- その他の各種 I/O(FPGA 含む)へ 1 本

Memory Management Unit のはなし

Memory Management Unit の働き

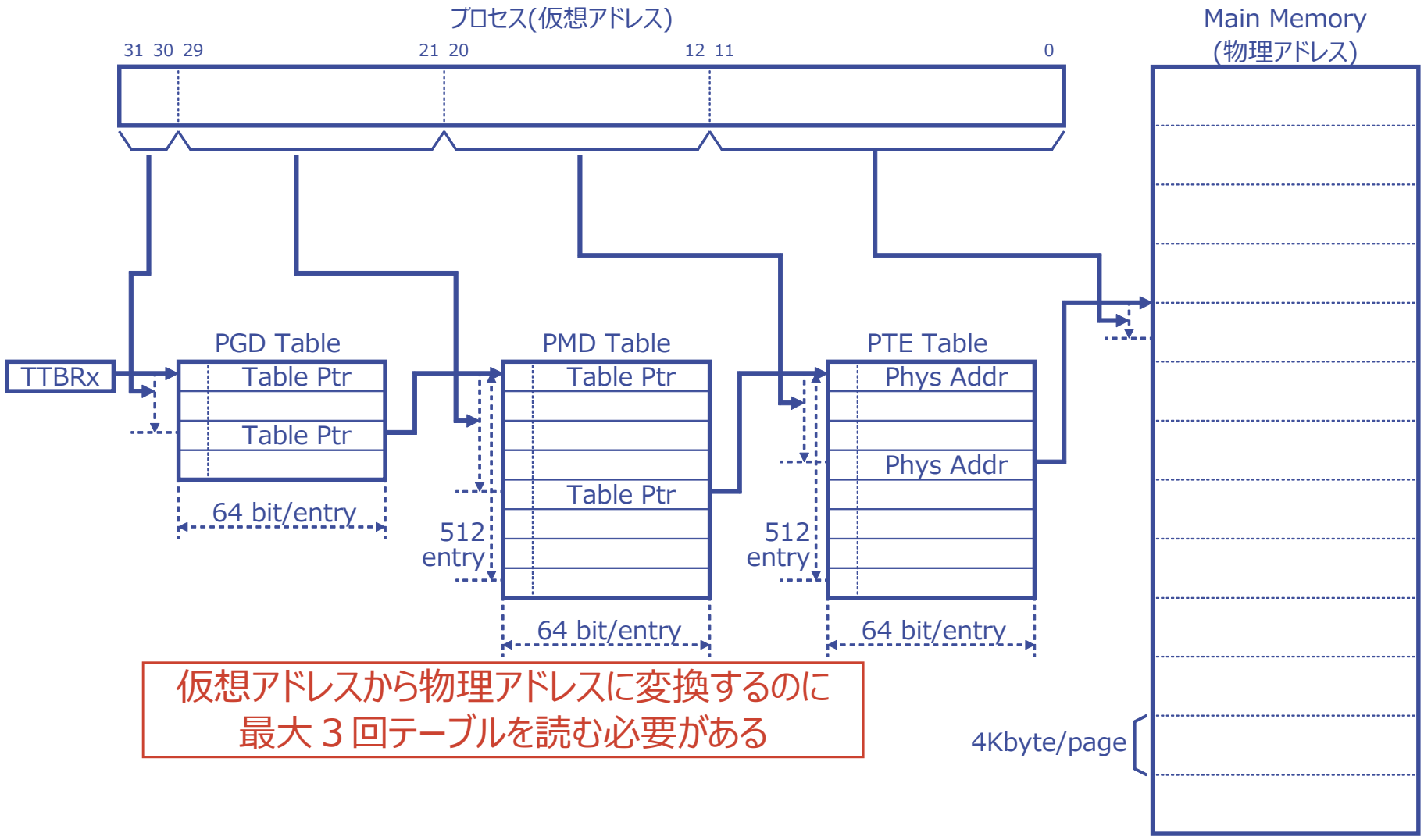
- 仮想アドレスから物理アドレスへの変換
  - 仮想記憶 - 個々のプロセスからは単一のメモリマップ
  - 物理メモリの有効利用
  - 物理アドレス空間と仮想アドレス空間の分離
- メモリ保護
- キャッシュ制御

仮想アドレスから物理アドレスへの変換(Aarch32-LPAE の例)

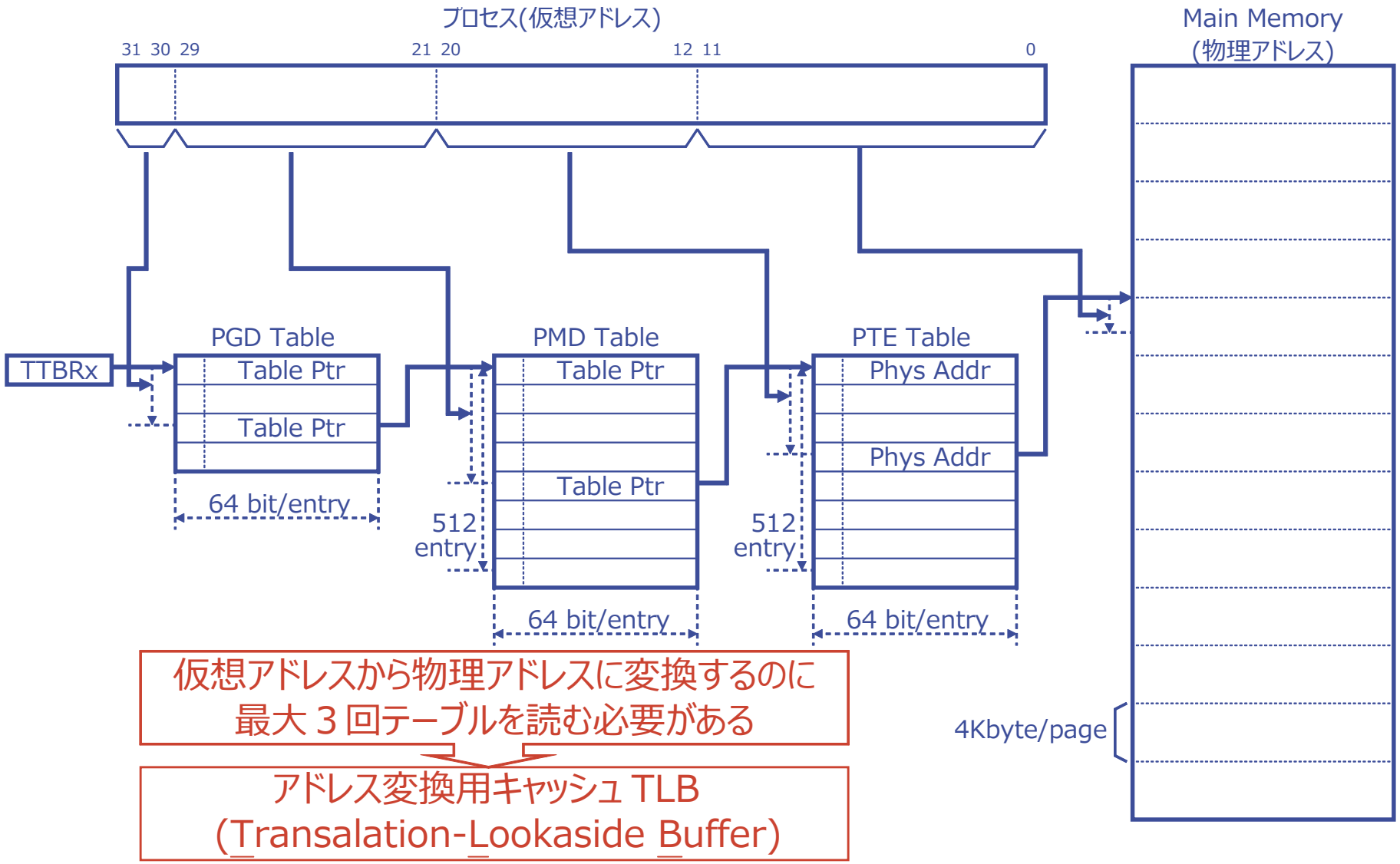


LPAE : Large Physical Address Extension
PTE : Page Table Entry
PMD : Page Middle Directory
PGD : Page Global Directory

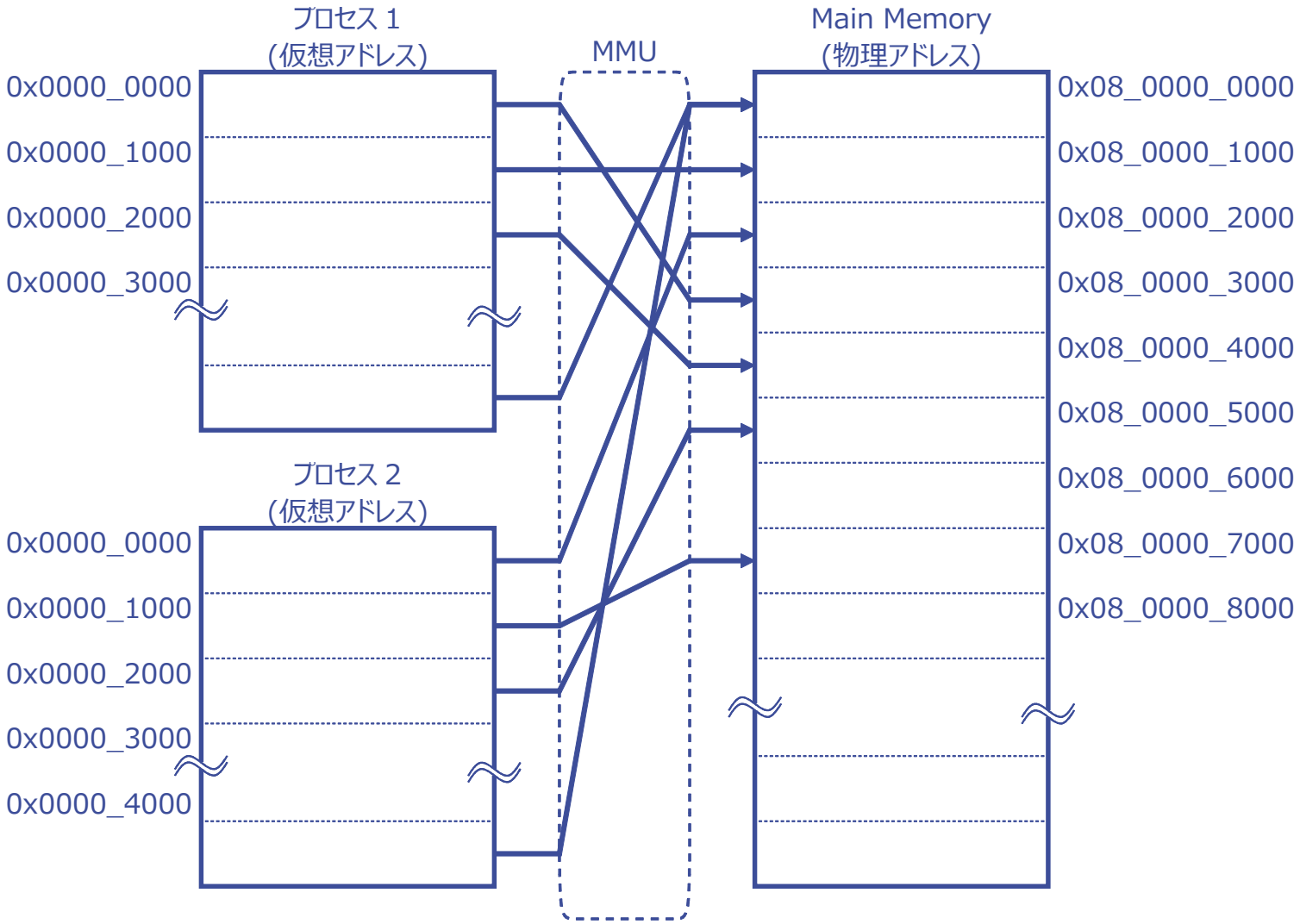
仮想アドレスから物理アドレスへの変換(Aarch32-LPAE の例)



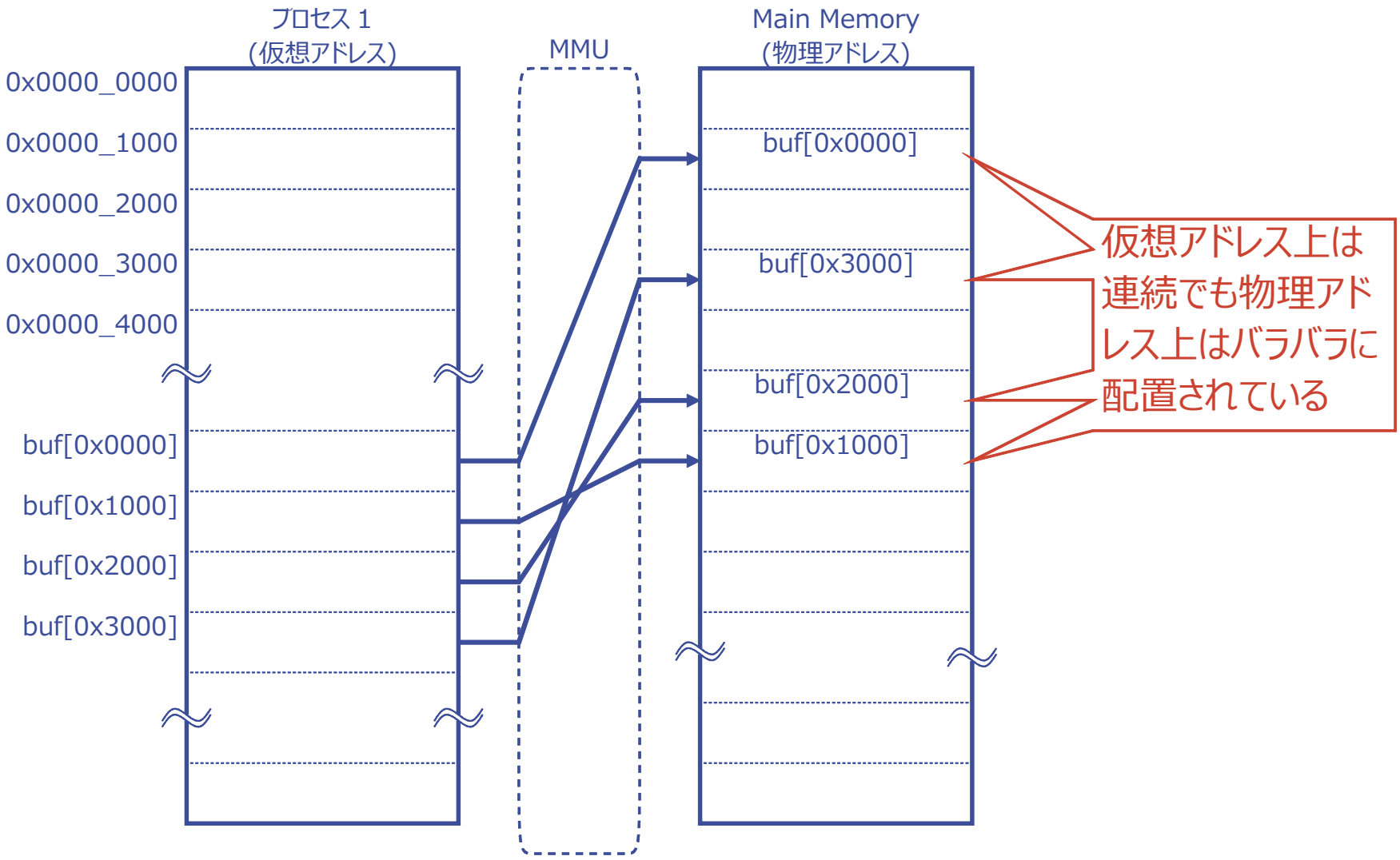
仮想アドレスから物理アドレスへの変換(Aarch32-LPAE の例)



仮想記憶 - 個々のプロセスからは単一のメモリマップ



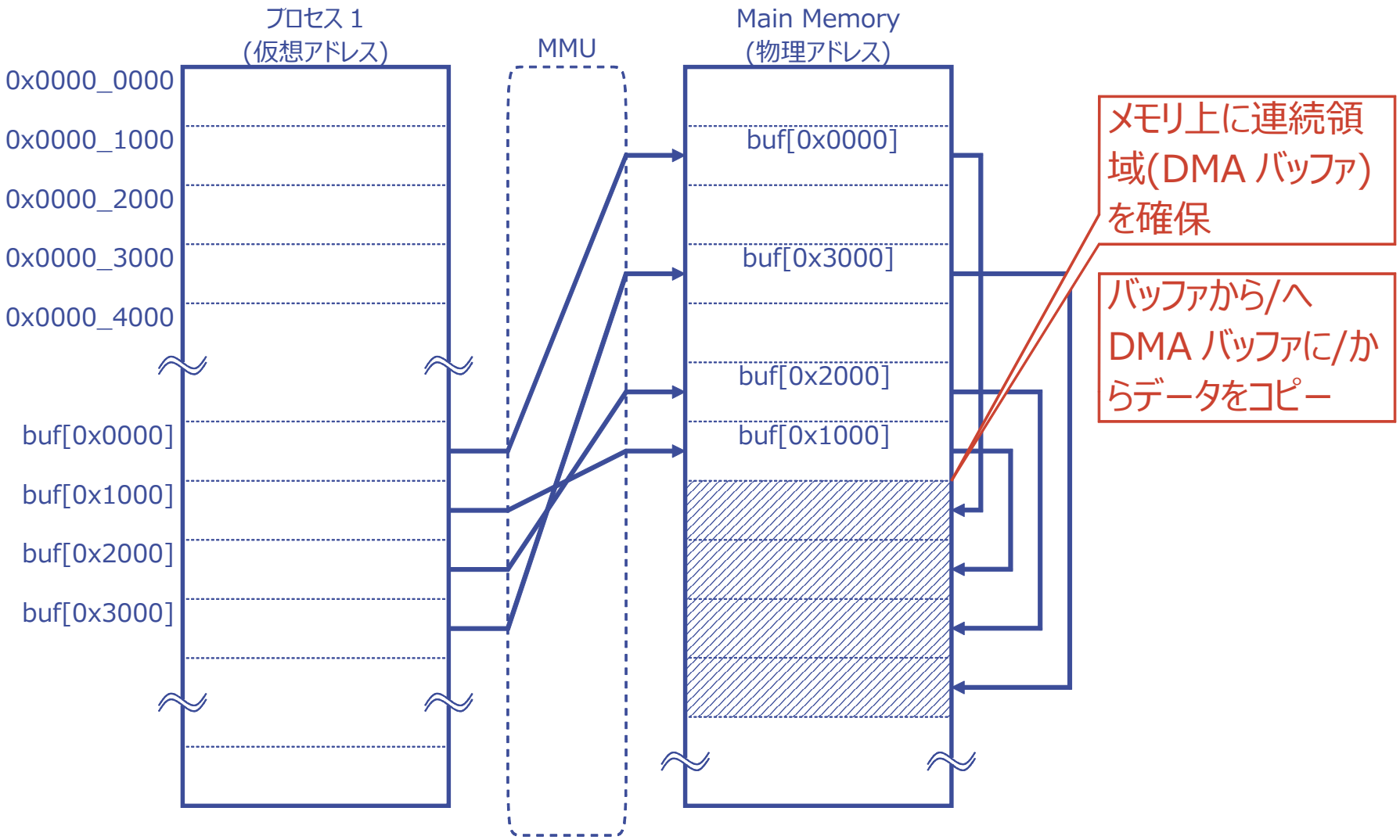
Accelerator から見た場合の問題



Accelerator からバラバラに配置されたバッファへのアクセス方法

- ハードウェアのアシスト無し
  - 物理メモリ上に連続領域(DMA バッファ)を確保
    - バッファから/へ DMA バッファへ/からデータをコピー
    - DMA バッファをユーザープロセスからアクセス
- ハードウェアのアシスト有り
    - Scatter-Gather DMA
    - IOMMU

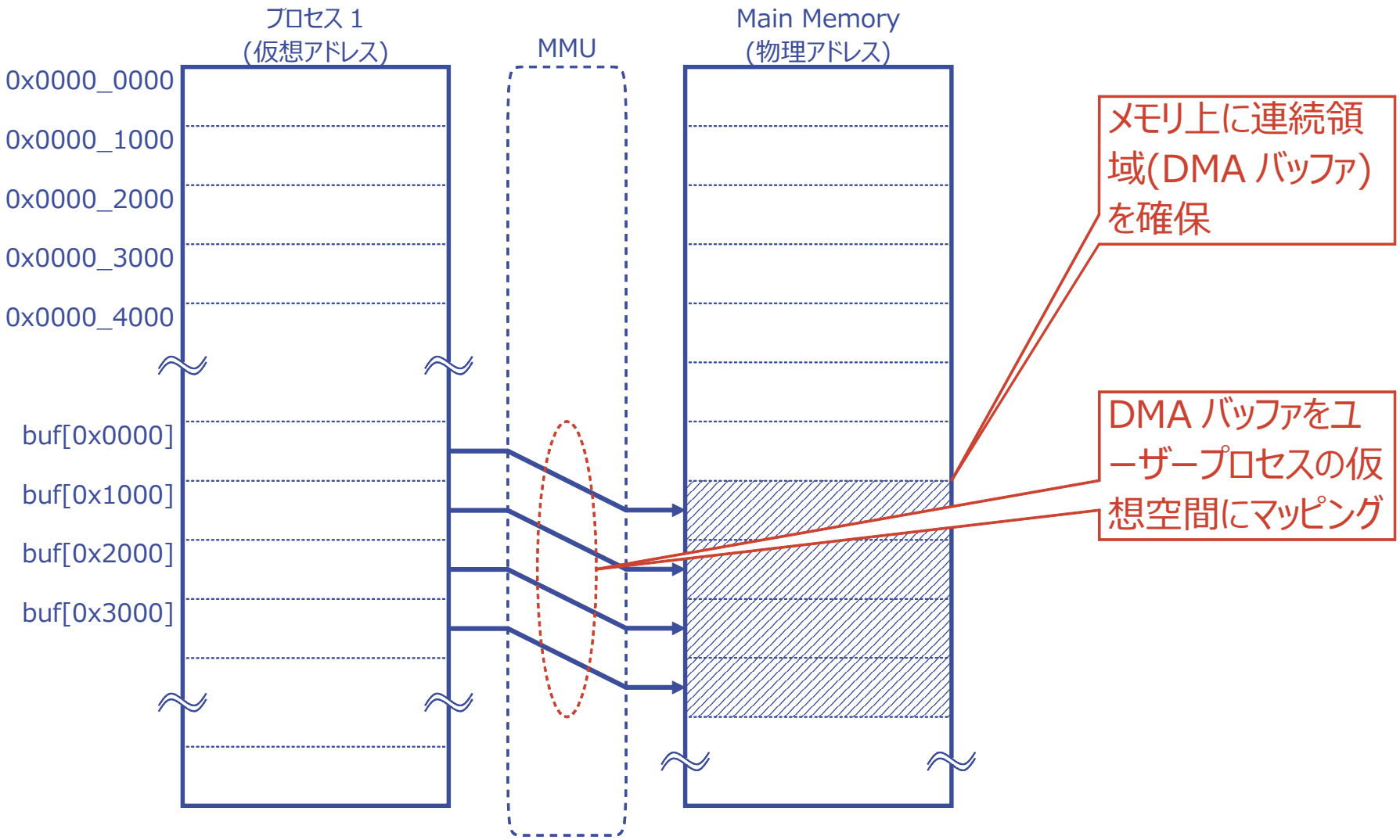
バッファから/へ DMA バッファへ/からデータをコピー



バッファから/へ DMA バッファへ/からデータをコピー

- 良い点
  - 特殊なハードウェアを必要としない
- 悪い点
  - データのコピーによる性能劣化
- 課題
  - 連続領域(DMA バッファ)の確保方法(後述)
- 例
  - レガシーなデバイス(一昔前はわりとメジャーな方法)
  - 今でも低速なデバイス(UART など)ではよく使われる

DMA バッファをユーザープロセスからアクセス



DMA バッファをユーザープロセスからアクセス

- 良い点
  - 特殊なハードウェアを必要としない
- 悪い点
  - ユーザープログラム側に対処が必要
- 課題
  - 連続領域(DMA バッファ)の確保方法(後述)
- 例
  - UIO(User space I/O)
  - udmabuf

連続領域(DMA バッファ)の確保方法 - Linux の場合

- Linux の管理外領域に連続領域を確保
  - メモリ管理をユーザープログラムが行う必要がある
  - 領域は Linux 起動時に確保
  - キャッシュ対象外(キャッシュが効かない)
- CMA(Contiguous Memory Allocator)を使う
  - メモリ管理は Linux のカーネルが行う
  - CMA 領域の最大値は Linux 起動時に指定
  - "ある程度"動的にバッファを確保できる(フラグメンテーション問題)
  - キャッシュの対象

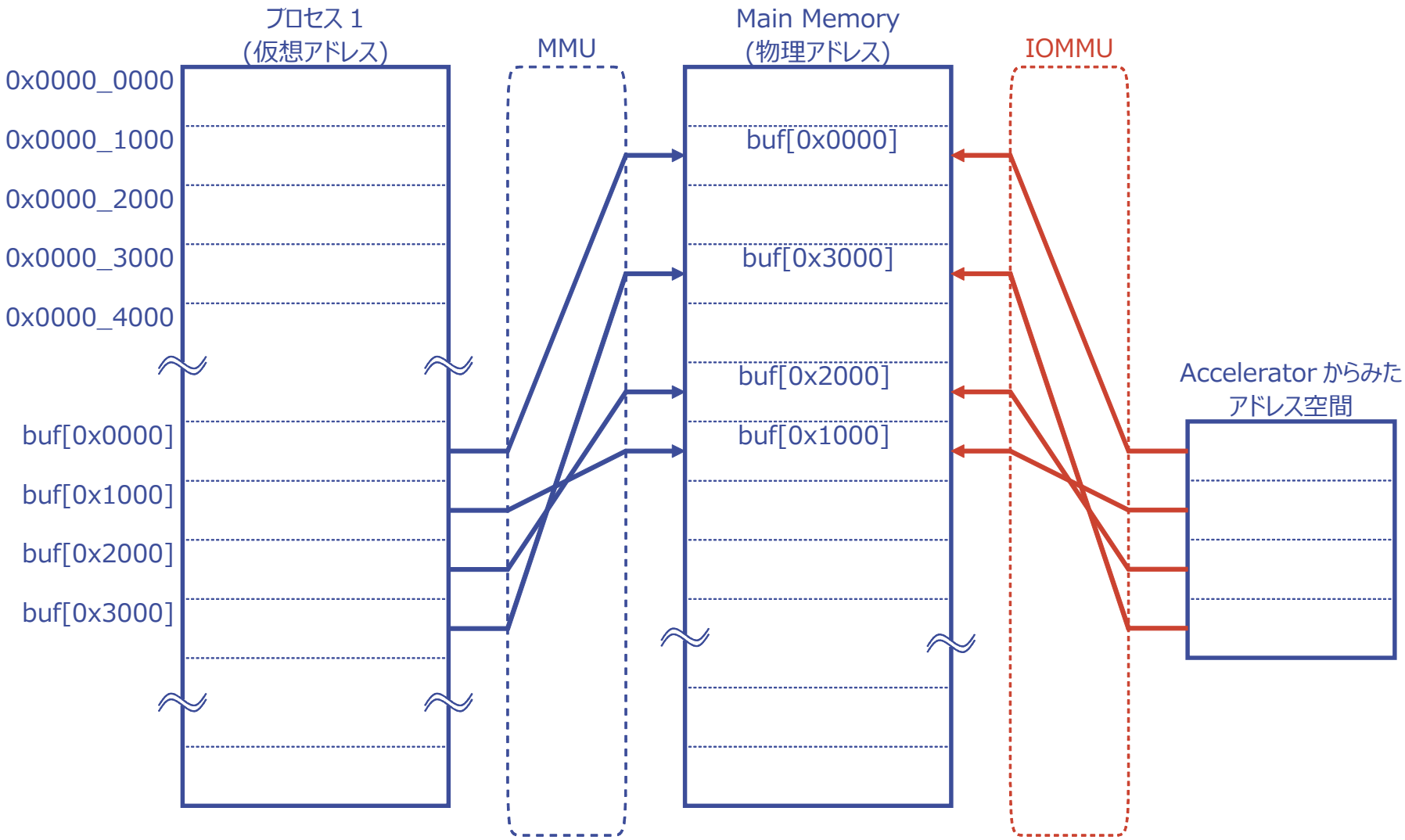
Scatter-Gather DMA

- 専用 DMA がバラバラに配置された物理領域を順番にアクセスする
- 例
  - USB ホストコントローラー(xHCI)
  - ネットワークコントローラー
  - SerialATA ホストコントローラー(AHCI)
  - メジャーかつ高スループットが必要なデバイスは大抵この方法

Scatter-Gather DMA

- 良い点
  - デバイスに特化した DMA による性能向上(高スループット)
  - バッファ間のデータコピーの削減
- 悪い点
  - 専用の DMA コントローラハードウェアが必要
  - 専用のデバイスドライバ(の開発とメンテナンス)が必要
  - ランダムアクセスに難がある(たいていはシーケンシャルアクセス)

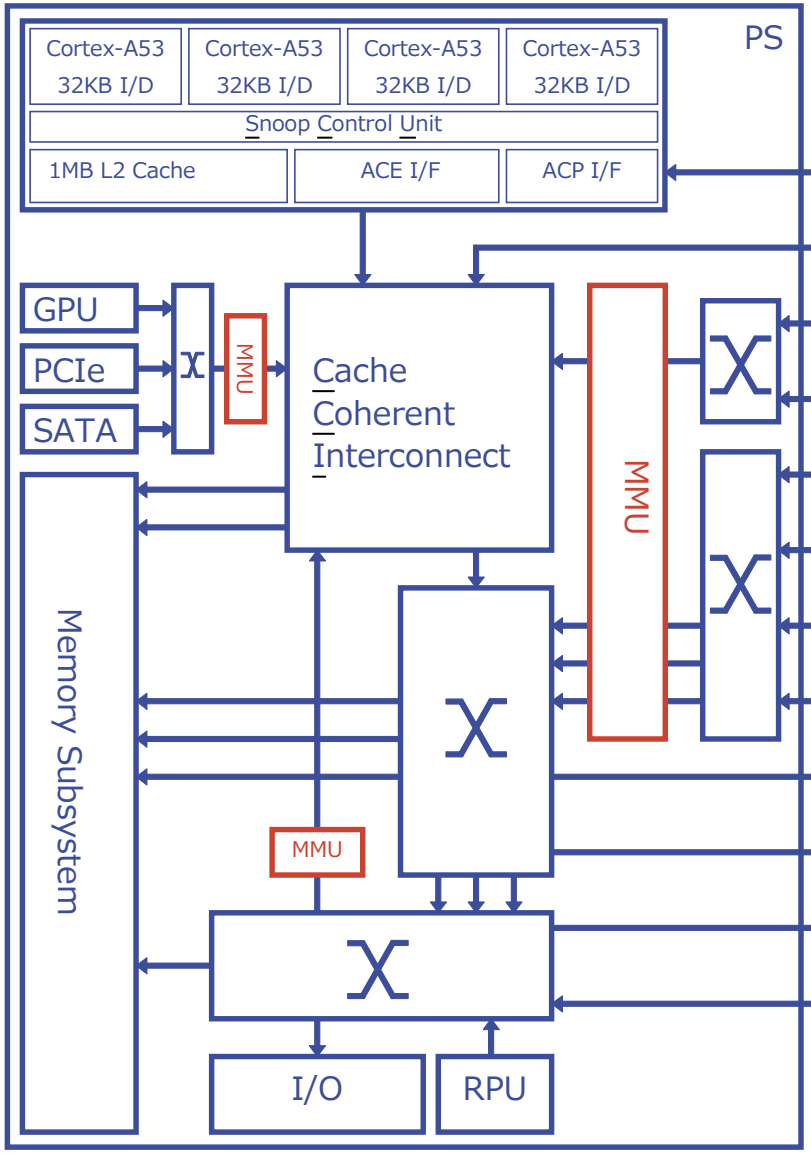
IOMMU


プロセス 1
(仮想アドレス)
MMU
Main Memory
(物理アドレス)
IOMMU
0x0000_0000
0x0000_1000
0x0000_2000
0x0000_3000
0x0000_4000
buf[0x0000]
buf[0x1000]
buf[0x2000]
buf[0x3000]
buf[0x0000]
buf[0x3000]
buf[0x2000]
buf[0x1000]
Accelerator からみた
アドレス空間

IOMMU

- 良い点
  - DMA の開発が楽(面倒は MMU にお任せ)
  - ドライバの開発が楽(MMU の管理は OS にお任せ)
  - バッファ間のデータコピーの削減
  - ランダムアクセスに強い
- 悪い点
  - IOMMU が必要
  - 仮想アドレスから物理アドレスに変換する際のオーバーヘッド
  - いまいち流行ってない(知見が少ない、これからに期待)

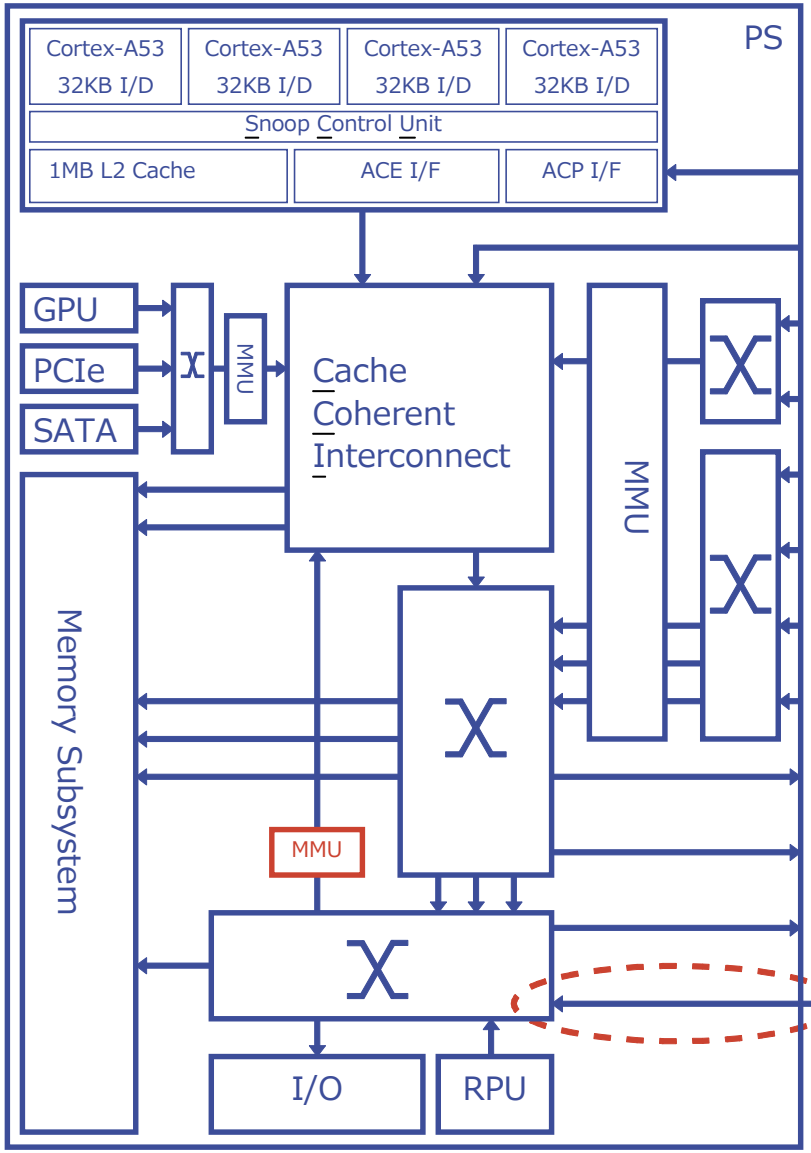
ZynqMP の IOMMU



PL

Interface Name	Addr Width	Data Width	IOMMU	Cache Coherence
S_AXI_ACP_FPD	40	32 or 64 or 128	No	I/O
S_AXI_ACE_FPD				Full
S_AXI_HPC0_FPD	49		Yes	I/O
S_AXI_HPC1_FPD				
S_AXI_HP0_FPD				No
S_AXI_HP1_FPD				
S_AXI_HP2_FPD				
S_AXI_HP3_FPD				
M_AXI_HPM0_FPD	40			
M_AXI_HPM1_FPD				
M_AXI_HPM0_LPD	32			
S_AXI_LPD	49		Yes / No	I/O / No

ZynqMP の S_AXI_LPD の注意点



LPD-PL Interface

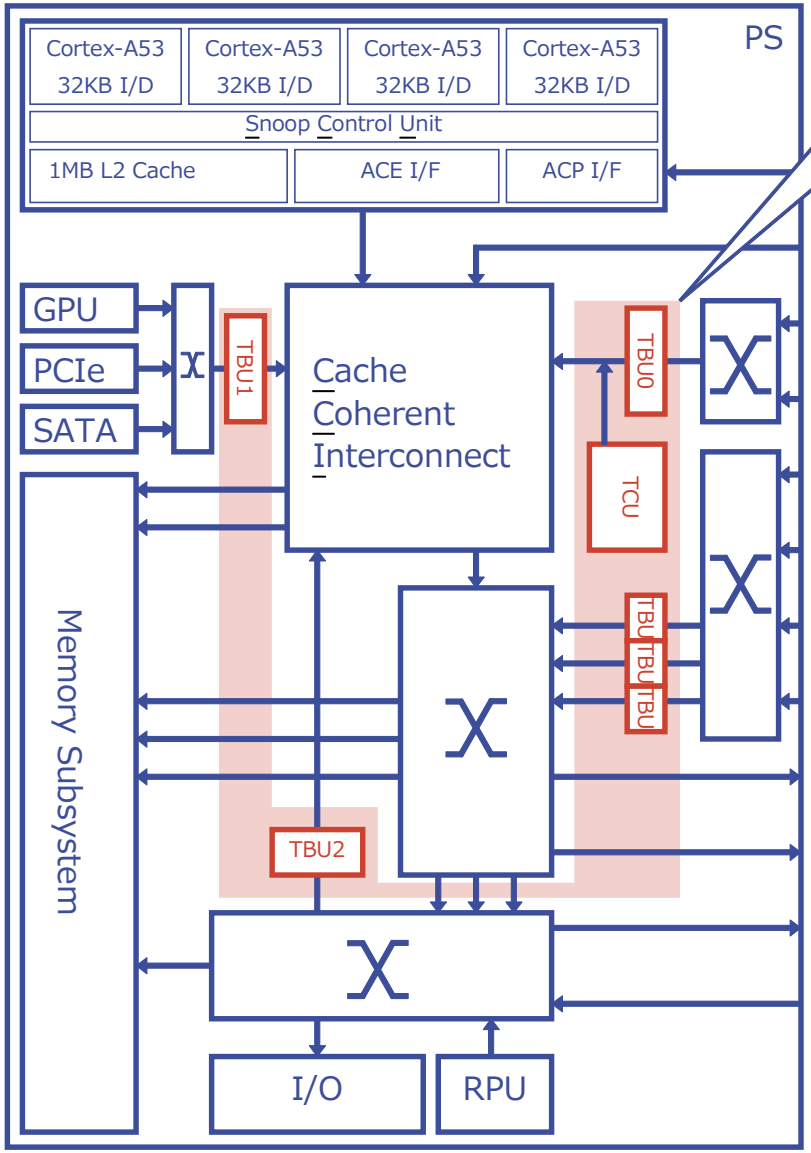
This port can be used in physical or virtual mode by setting the AxUSER bit. In virtual mode, it cannot directly access the LPD. Instead, virtual mode accesses are routed as follows.

PL → LPD → FPD (SMMU/CCI) → LPD

The S_AXI_PL_LPD is a PL interface that connects into the low-power domain. For situations where the FP domain is powered down, this interface provides a high-performance mastering capability from the PL. Due to the interconnect topology, this port has a relatively long latency to DDR.

『 Zynq UltraScale+ MPSoC TRM UG1085 (v1.0) November 24,2015』822

ZynqMP の IOMMU(ちょっと詳細に)



SMMU(System Memory Management Unit)

各 I/F 毎に MMU があるのではない

全体で IOMMU は一つ

次の 2 つのコンポーネントで構成

- TBU(Traslation Buffer Unit)
  - アドレス変換用キャッシュ TLB(Translation Lookaside Buffer)
  - TLB にヒットした時のみアドレス変換を行う
  - TLB ミスの時は TCU にテーブルウォークを依頼する
  - 各 I/F の近くに分散配置
- TCU(Translation Control Unit)
  - テーブルウォークを行い、各 TBU に変換結果を通知

IOMMU のアドレス変換のオーバヘッドはどのくらいだろう？

- オーバーヘッドの"平均"サイクル数
  - TLB ヒット率×1＋(1-TLB ヒット率)×TLB ミス時の Table Walk サイクル数
- TLB のヒット率は？
  - Accelerator のアクセスパターン
    - シーケンシャル/ランダム
    - アドレスの先出しする/しない
    - 1 トランザクションの転送量
  - TLB の構成(ZynqMP では 512entry 4-way)
  - TLB の追い出しアルゴリズム(LRU? Random?)
  - Table Walk 時に充填する TLB のエントリ数は？
- TLB ミス時の Table Walk の性能はどのくらい？

IOMMU のアドレス変換のオーバヘッドはどのくらいだろう？

- オーバーヘッドの"平均"サイクル数
  - TLB ヒット率×1＋(1-TLB ヒット率)×TLB ミス時の Table Walk サイクル数
- TLB のヒット率は？
  - Accelerator のアクセスパターン
    - シーケンシャル/ランダム
    - アドレスの先出しする/しない
    - 1 トランザクション
  - TLB のエントリ数は？
- TLB ミス時の Table Walk の性能はどのくらい？

やってみないとわからん

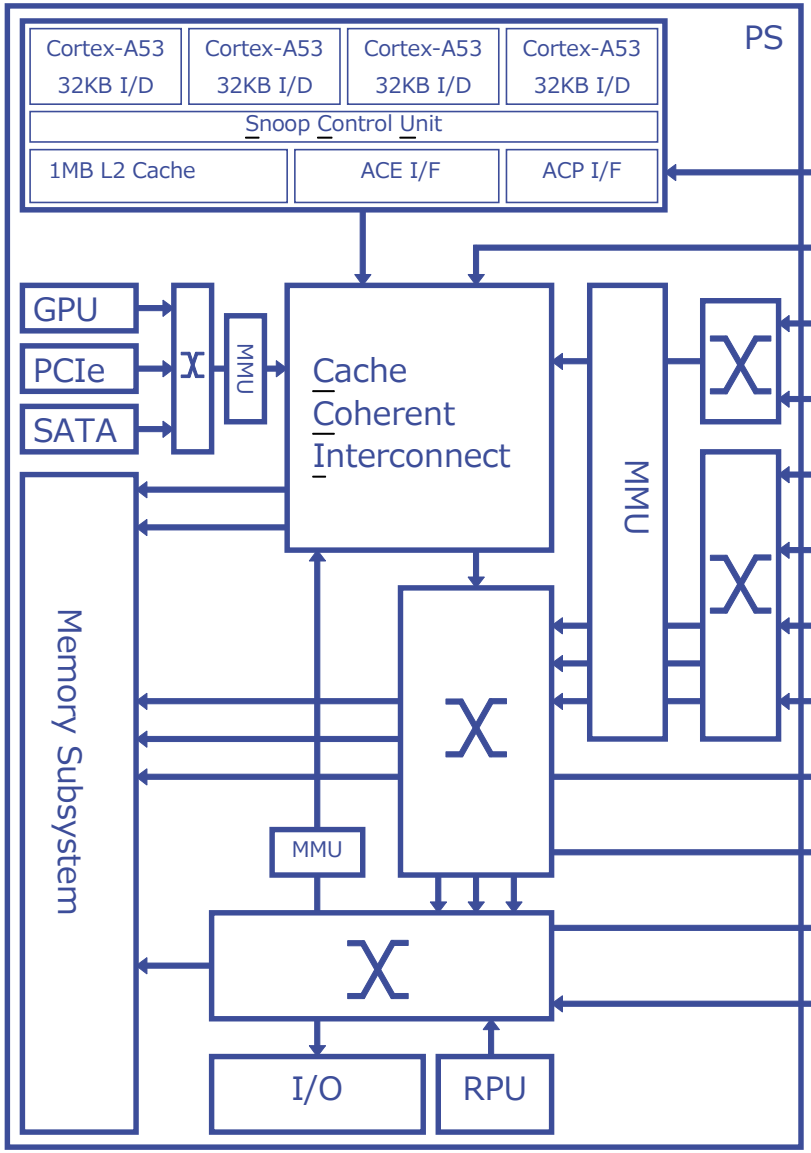
ZynqMPの Global System Address Map

Range Name	Size	Start Address	End Address	32bit addr	36bit addr	40bit addr
DDR SDRAM	2GB	0x00_0000_0000	0x00_7FFF_FFFF			
LPD-PL interface	512MB	0x00_8000_0000	0x00_9FFF_FFFF			
FPD-PL(HPM0) interface	256MB	0x00_A000_0000	0x00_AFFF_FFFF			
FPD-PL(HPM1) interface	256MB	0x00_B000_0000	0x00_BFFF_FFFF			
Quad-SPI	512MB	0x00_C000_0000	0x00_DFFF_FFFF			
PCIe Low	256MB	0x00_E000_0000	0x00_EFFF_FFFF			
Reserved	128MB	0x00_F000_0000	0x00_F7FF_FFFF			
Coresight System Trace	16MB	0x00_F800_0000	0x00_F8FF_FFFF			
RPU Low Latency Port	1MB	0x00_F900_0000	0x00_F90F_FFFF			
Reserved	63MB	0x00_F910_0000	0x00_FCFF_FFFF			
FPS Slave	16MB	0x00_FD00_0000	0x00_FDFF_FFFF			
Upper LPS Slave	16MB	0x00_FE00_0000	0x00_FEFF_FFFF			
Lower LPS Slave	12MB	0x00_FF00_0000	0x00_FFBF_FFFF			
CSU, PMU, TCM, OCM	4MB	0x00_FFC0_0000	0x00_FFFF_FFFF			
Reserved	12GB	0x01_0000_0000	0x03_FFFF_FFFF			
FPD-PL(HPM0) interface	4GB	0x04_0000_0000	0x04_FFFF_FFFF			
FPD-PL(HPM1) interface	4GB	0x05_0000_0000	0x05_FFFF_FFFF			
PCIe High	8GB	0x06_0000_0000	0x07_FFFF_FFFF			
DDR SDRAM	32GB	0x08_0000_0000	0x0F_FFFF_FFFF			
FPD-PL(HPM0) interface	224GB	0x10_0000_0000	0x47_FFFF_FFFF			
FPD-PL(HPM1) interface	224GB	0x48_0000_0000	0x7F_FFFF_FFFF			
PCIe High	256GB	0x80_0000_0000	0xBF_FFFF_FFFF			
DDR SDRAM	256GB	0xC0_0000_0000	0xFF_FFFF_FFFF			

32bit アドレスの注意事項

- MMU を使わないとアクセス可能なメインメモリは最大 2GB
  - APU が Aarch32(ARM 32bit mode)の場合
  - RPU(Cortex-R5)の場合
  - PL 側のマスターのアドレスが 32bit しかない場合

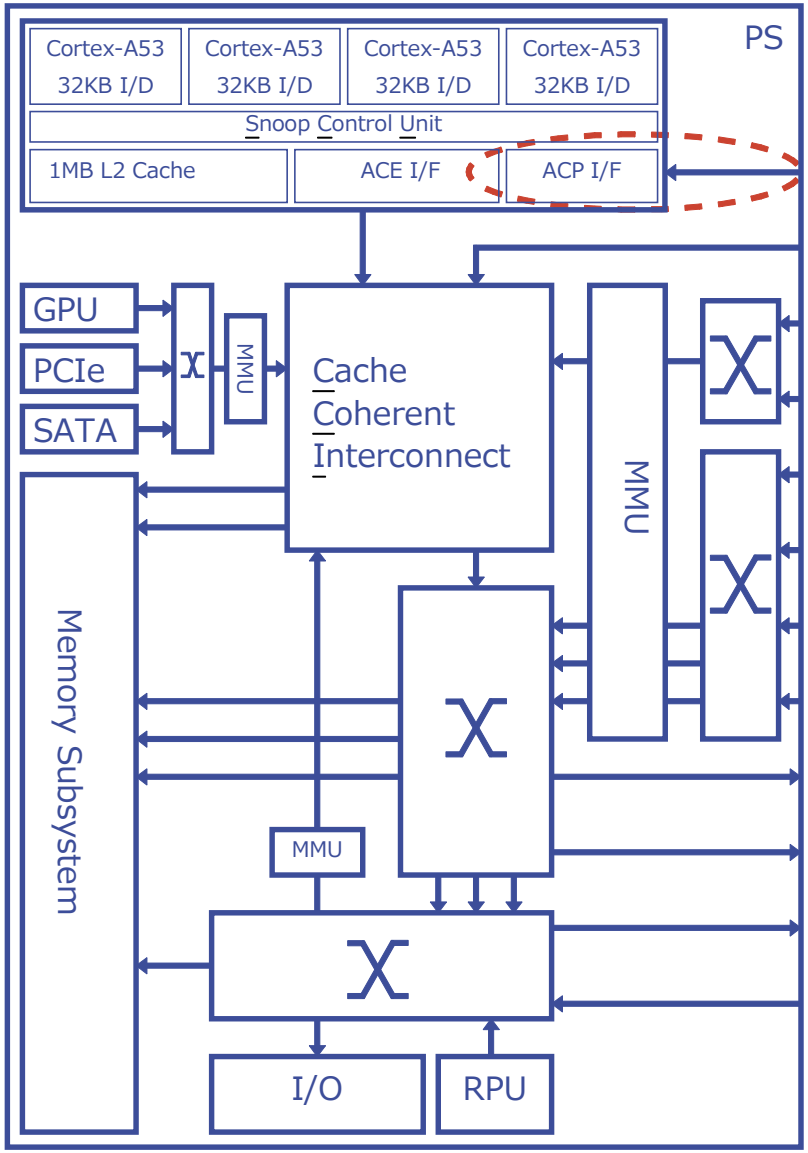
ZynqMP PS-PL Interface



PL

Interface Name	Addr Width	Data Width	MMU	Cache Coherence
S_AXI_ACP_FPD	40	32 or 64 or 128	No	I/O
S_AXI_ACE_FPD				Full
S_AXI_HPC0_FPD	49		Yes	I/O
S_AXI_HPC1_FPD				
S_AXI_HP0_FPD				No
S_AXI_HP1_FPD				
S_AXI_HP2_FPD				
S_AXI_HP3_FPD				
M_AXI_HPM0_FPD	40			
M_AXI_HPM1_FPD				
M_AXI_HPM0_LPD	32			
S_AXI_LPD	49		Yes / No	I/O / No

ZynqMP の S_AXI_ACP_FPD



S_AXI_ACP_FPD

・利点(Benefit)

- Lowest latency to L2 cache.
- Cache coherency.
- Ability to allocate into L2 cache.

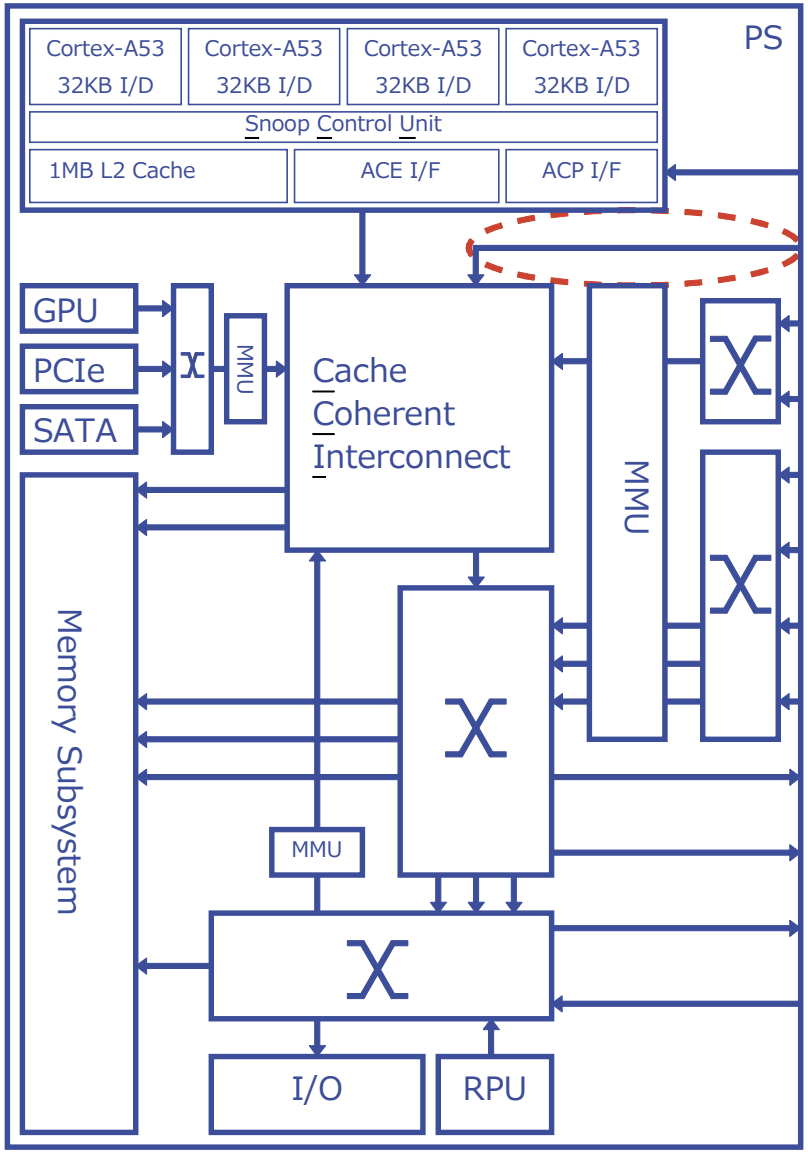
・検討事項(Considerations)

- Only 64B and 16B transactions supported
  - requires specially designed PL-DMA.
- Shares CPU interconnect bandwidth.
- More complex PL master design.

・推奨使用方法(Suggested Uses)

- PL logic tightly coupled with APU.
- Medium granularity CPU offload.

ZynqMP の S_AXI_ACE_FPD



S_AXI_ACE_FPD

・利点(Benefit)

- Optional cache coherency.
- APU can snoop into PL cached masters (two-way coherency).

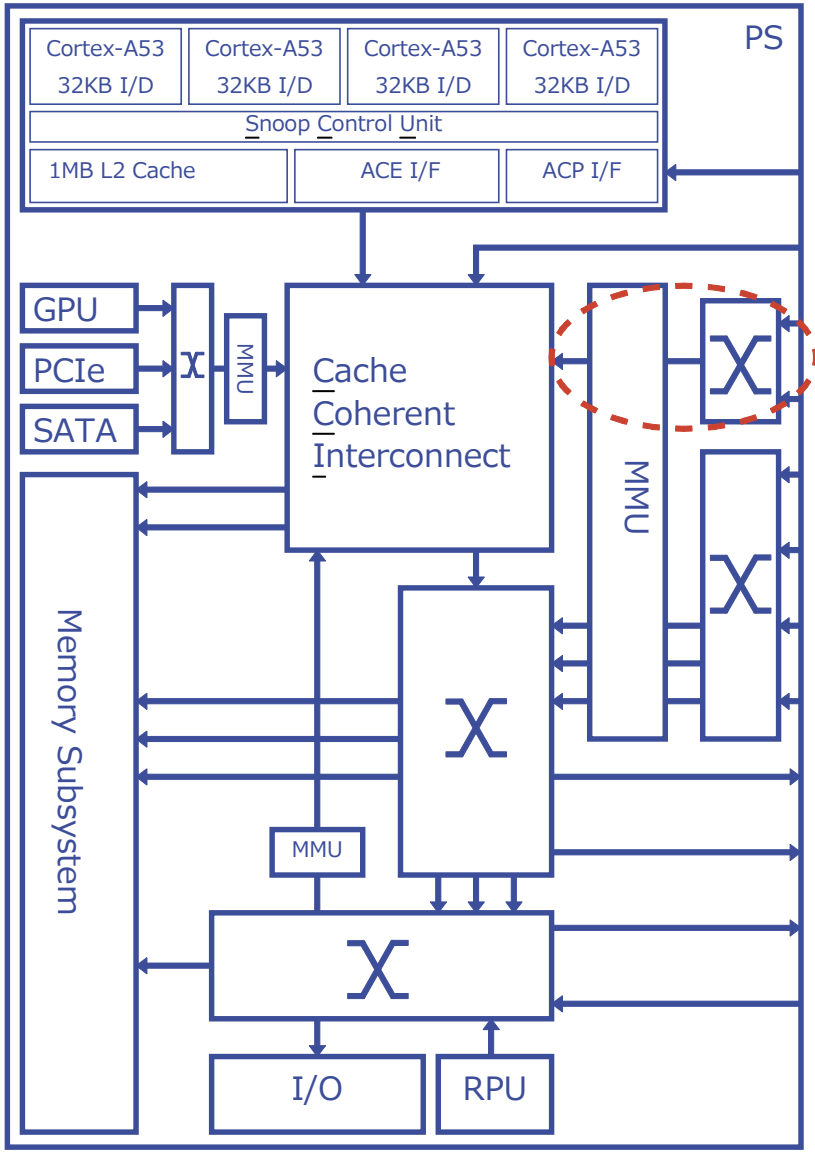
・検討事項(Considerations)

- Limited burst length (64B) support when snooping into cached master in PL (PS to PL).
- When used as ACELITE, larger burst length from PL to PS can cause the CPU to hang (direct AXI4 path between CCI and PS-DDR, which impacts other applications waiting to access DDR).
- Complex PL design that require support for ACE.

・推奨使用方法(Suggested Uses)

- Cached accelerators in PL.
- System cache in PL using block RAM.

ZynqMP の S_AXI_HPC[01]_FPD



S_AXI_HPC[01]_FPD

・利点(Benefit)

- High throughput (Best Effort).
- Multiple interfaces.
- AXI FIFO interface with QoS-400 traffic shaping.
- Hardware assisted coherency;
  no cache flush/invalidate in software driver.
- Virtualization support with SMMU in path.

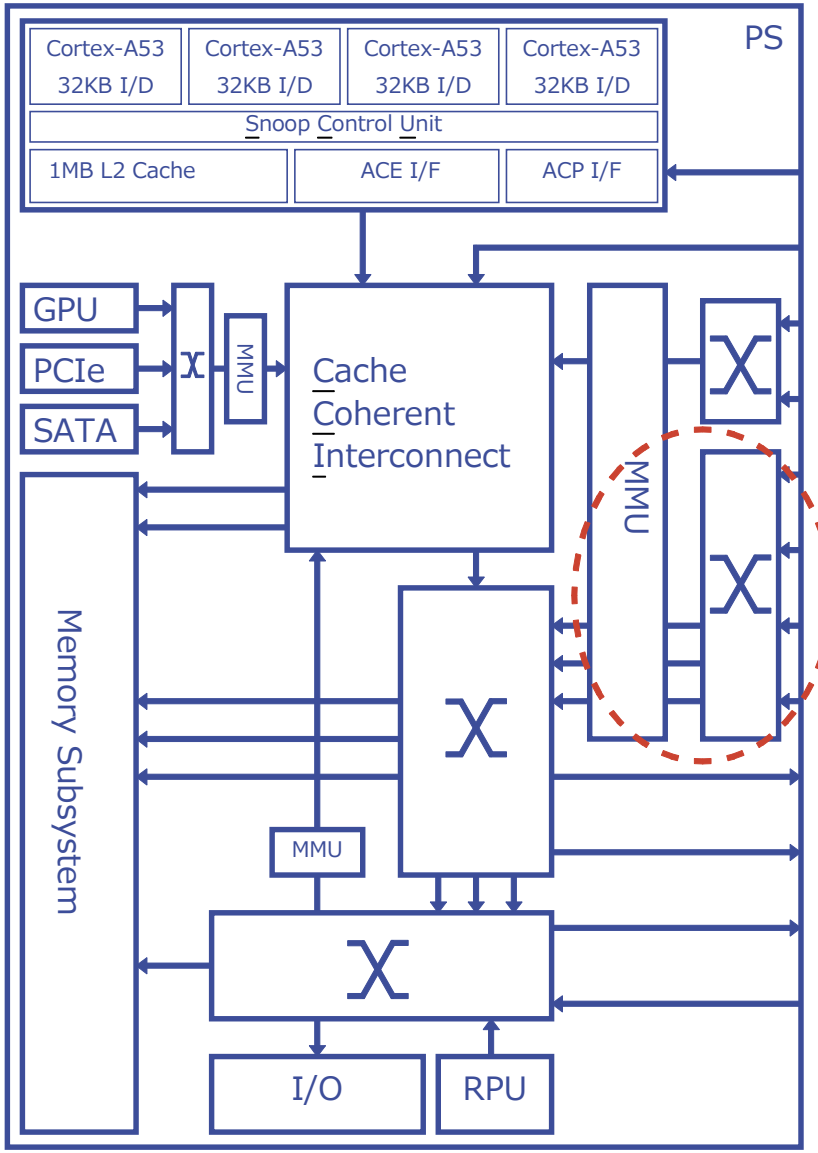
・検討事項(Considerations)

- More complex PL master design.
- PL design to drive AxCACHE as needed for coherency.

・推奨使用方法(Suggested Uses)

- High performance DMA for large datasets.

ZynqMP の S_AXI_HP[0123]_FPD



S_AXI_HP[0123]_FPD

・利点(Benefit)

- High throughput (Best Effort).
- Multiple interfaces.
- AXI FIFO interface with QoS-400 traffic shaping.
- Virtualization support with SMMU in path.

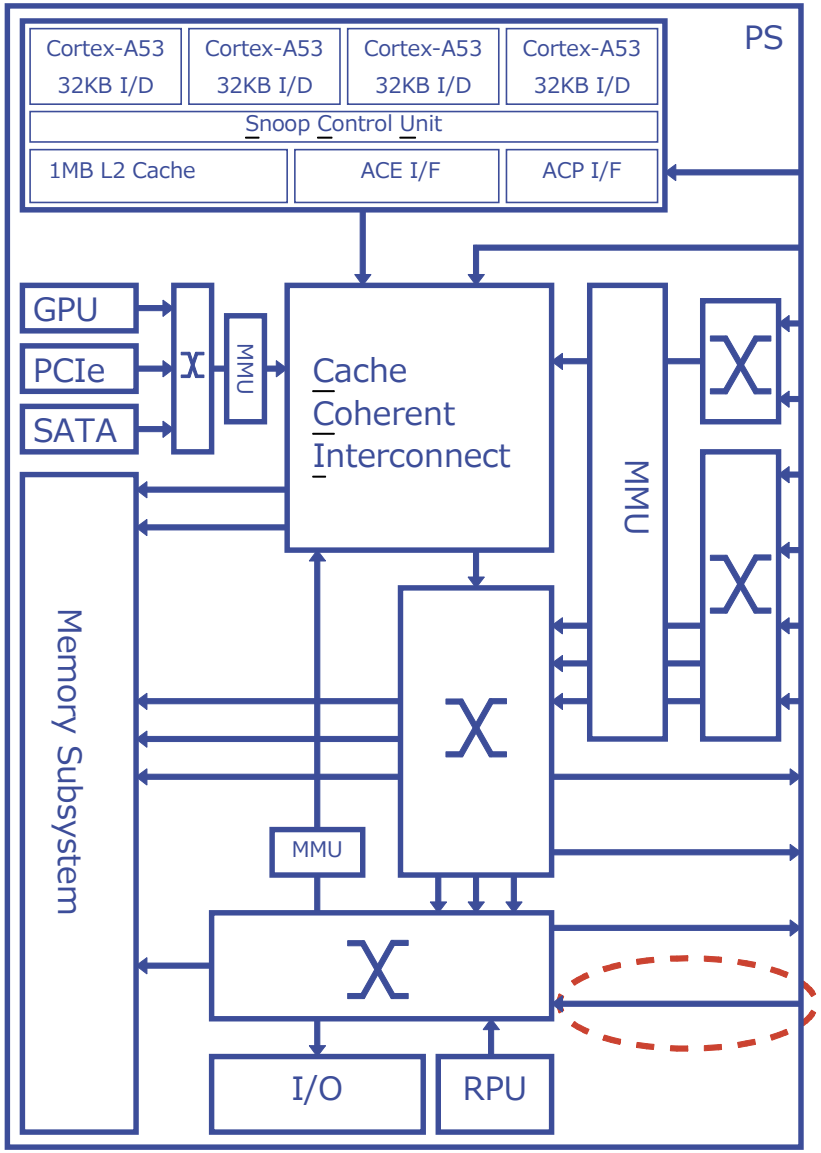
・検討事項(Considerations)

- Software driver to handle cache flush/invalidate.
- More complex PL master design.

・推奨使用方法(Suggested Uses)

- High performance DMA for large datasets.

ZynqMP の S_AXI_LPD



S_AXI_LPD

・利点(Benefit)

- Fastest, low latency path to the OCM and TCM.
- Optional coherency.
- SMMU in datapath provides option for virtualization.
- PL access to LPD when FPD is powered off.
- High throughput.

・推奨使用方法(Suggested Uses)

- Safety applications.