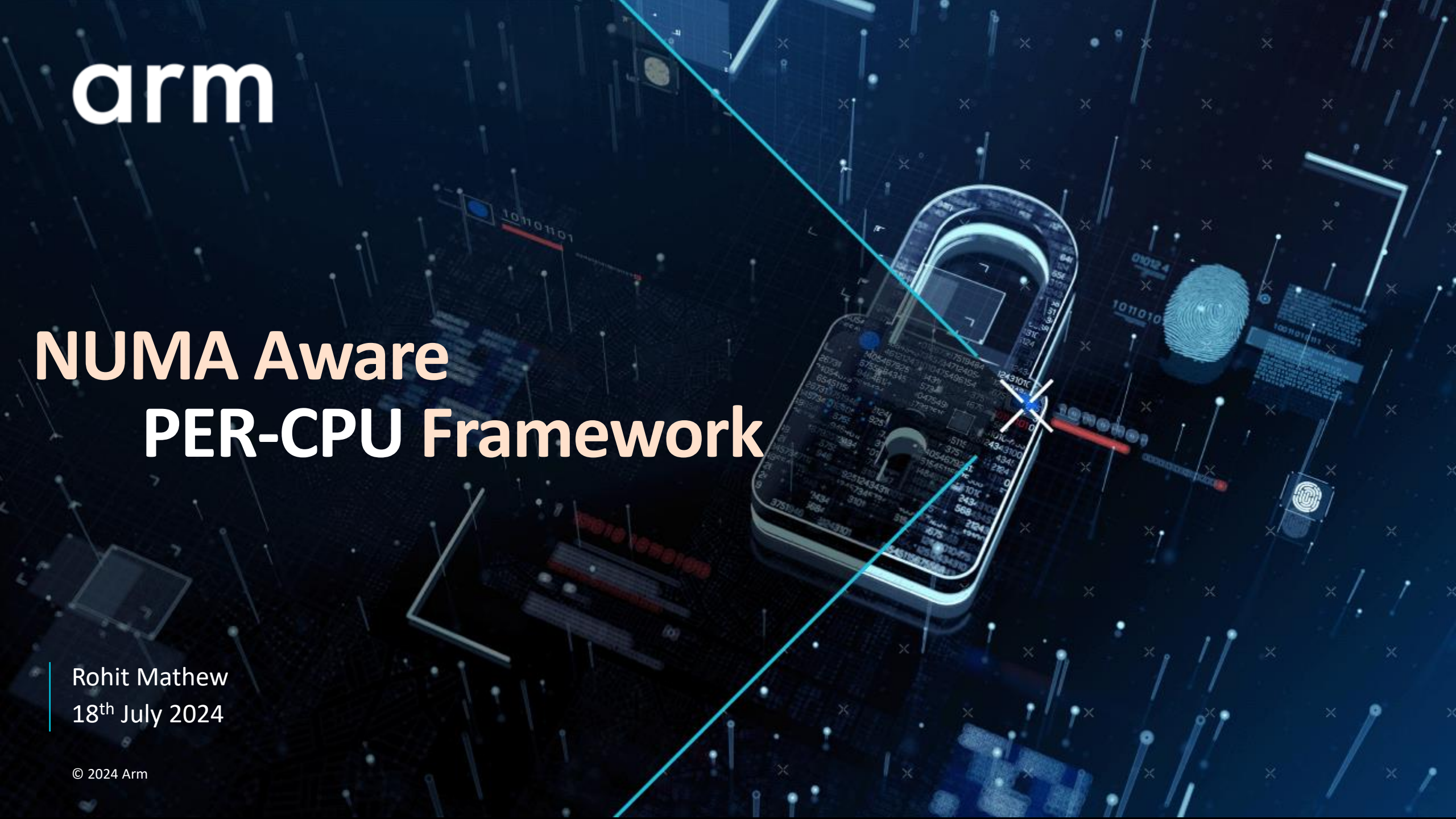


NUMA Aware PER-CPU Framework

Rohit Mathew

18th July 2024

Agenda

- Problem
  - Overview
  - PER-CPU Objects
- Proposal – NUMA Aware PER-CPU Framework
  - How do we do it?
  - Platform's responsibility
  - Definer Interface
  - Accessor Interface
  - Optimization 1 - tpidr_el3 magic
  - Optimization 2 – avoid cache thrashing
  - Stack migration
  - Interface variants

Problem

Overview

- Homogeneous multichip platforms have physically segregated SRAM in each chiplet
- TF-A runtime image size for multichip can exceeds a single SRAM size
  - Can we reduce the runtime Image size on the primary SRAM?
- Additionally, CPUs from non-primary chiplet deals with a NUMA latency due to cross chip access
  - Can we move parts of the Image to the SRAM local to CPU in context?

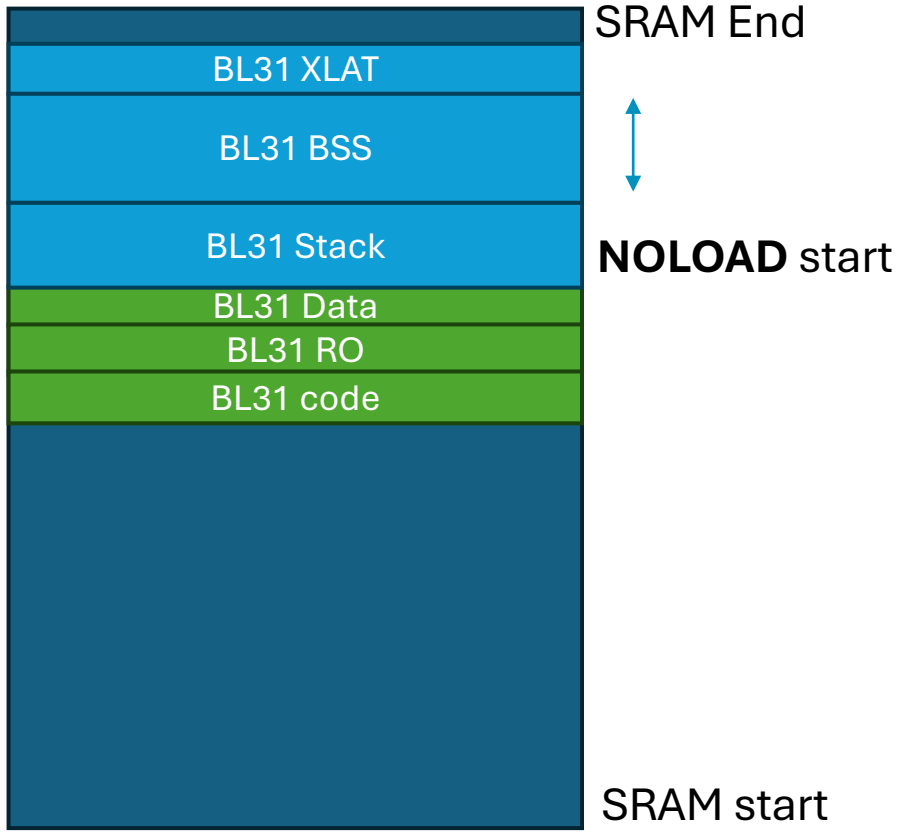
Problem

PER-CPU Objects

The NUMA problem

- TF-A has a *lot of* global objects that are per CPU.
  - RMM context, NS PSCI context, SPMD context etc
  - part of BSS which is not loaded explicitly, but forms part of the runtime.
- CPU objects are re-used through-out the lifetime of the system
  - Cross chip Read/write/snoops would add in **NUMA latency**

The Storage problem



Proposal : NUMA Aware PER-CPU Framework

How do we do it?

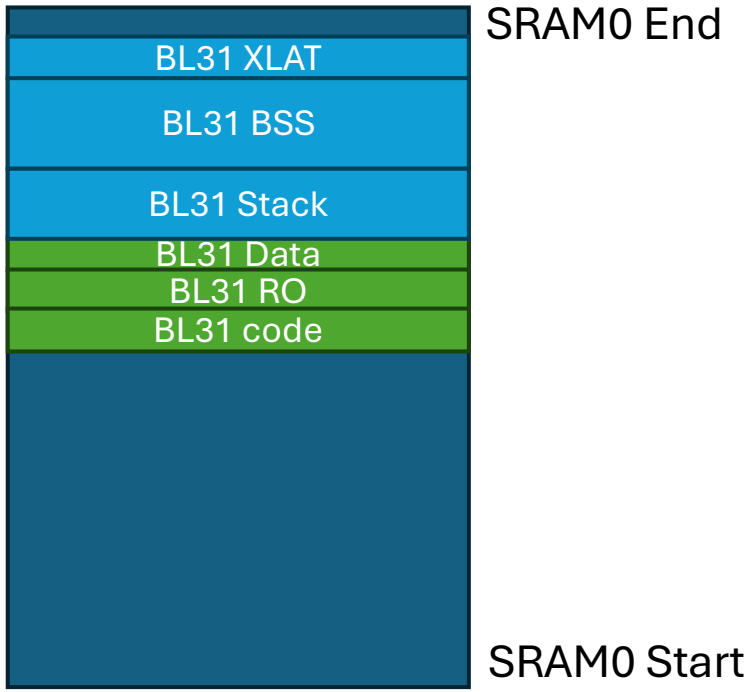
- Every PER-CPU object should *ideally* be defined using the framework's **Definer Interface**
- **Accessor Interface** would help with accessing these objects.
- For single chiplet systems, there is no change in how things works**
- For multi-chiplet systems, PER-CPU framework would deal with allocating globals spread across SRAMs
- A new section called ".per_cpu" would be introduced just for the multichip systems to tie PER-CPU globals in a single chip

**Certain optimizations can bring changes for single chip as well.

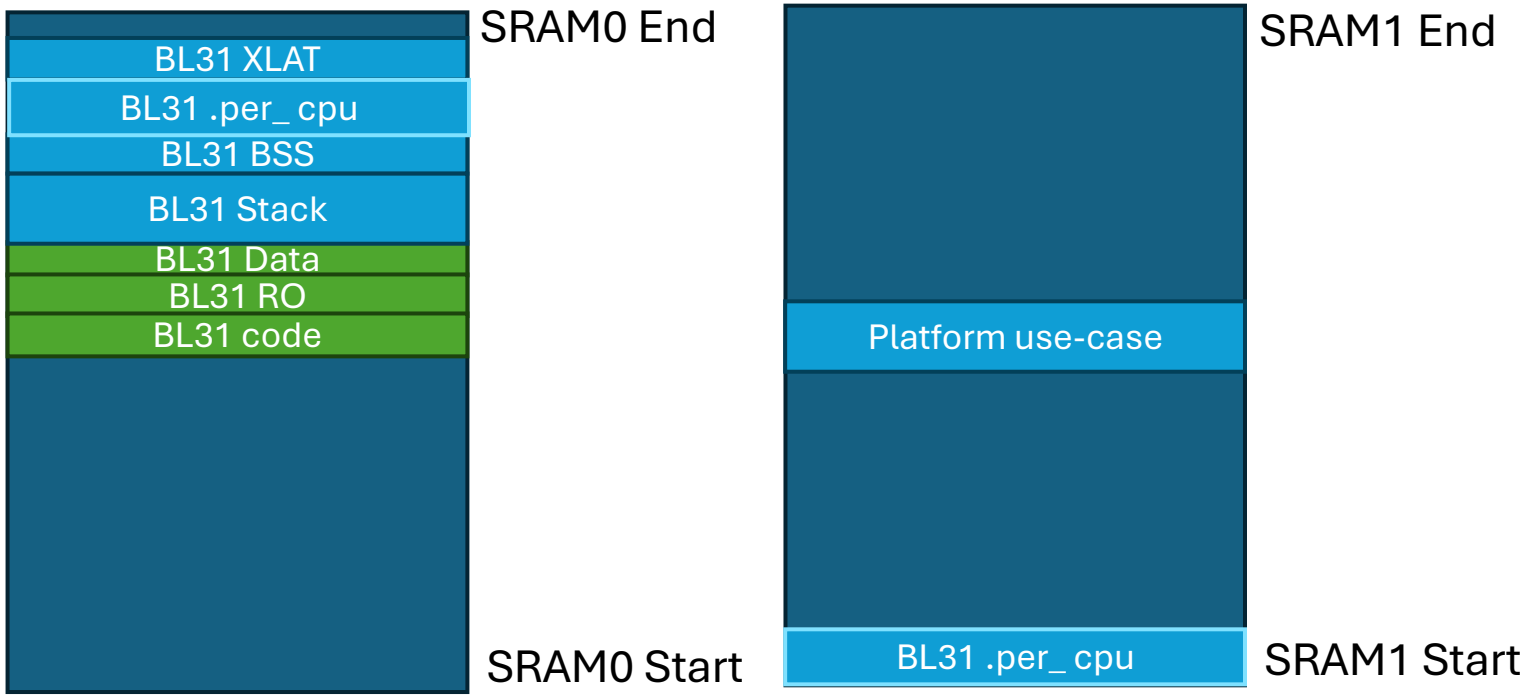
Proposal : NUMA Aware PER-CPU Framework

How does it look?

Single Chip



Multi-Chip



arm

Proposal : NUMA Aware PER-CPU Framework

Platform's Responsibility

Single chip

- Nothing to be done.

Multi-chip

- Set build option PER_CPU_MULTICHIP := 1
- Setup page tables for remote regions at desired locations.
- Implement
  - `uintptr_t plat_per_cpu_section_base(int cpu);`
  - This should return the address of the «.per_cpu » section corresponding to a CPU

Proposal : NUMA Aware PER-CPU Framework

Definer Interface

Single chip

```
#define DEFINE_PER_CPU(TYPE, NAME)          \
        TYPE NAME[PLATFORM_CORE_COUNT]
```

- No changes internally
- Here is an example use-case -

```
DEFINE_PER_CPU(rmmd_rmm_context_t, rmm_context);
```

Multi-chip

```
#define DEFINE_PER_CPU(TYPE, NAME)          \
        TYPE NAME[CHIPLET_CORE_COUNT]       \
        __section(PER_CPU_MULTICHIP_SECTION)
```

- The object is tied to a different section (.per_cpu)
- The number of cores have been reduced to cores per chiplet
- This region would be duplicated across each chiplets SRAM.

Proposal : NUMA Aware PER-CPU Framework

FOR_CPU_PTR accessor Interface

Single chip

```
#define FOR_CPU_PTR(NAME, CPU)      \
        &NAME[CPU]
```

- No changes internally
- Here is an example use-case -

```
rmmd_rmm_context_t *rmm_ctx = FOR_CPU_PTR(rmm_context,
linear_id);
```

- If used in an env where multi-CPUs can concurrently access, make sure to use proper locking primitives!

Multi-chip

```
#define PER_CPU_OFFSET(x) (x - PER_CPU_START)
#define FOR_CPU_PTR(NAME, CPU) __extension__    \
         ((__typeof__(&NAME[0]))                \
         (plat_per_cpu_section_base(CPU) +      \
 PER_CPU_OFFSET((uintptr_t)&NAME[CPU%CHIPLET_CORE_COUNT])))
```

- **`plat_per_cpu_section_base`** is implemented by the platform to return the section base for the CPU in context
- **`plat_per_cpu_section_base`** is one way of doing it; tpidr_el3 would be another way.

Proposal : NUMA Aware PER-CPU Framework

THIS_CPU_PTR accessor Interface

Single chip

```
#define THIS_CPU_PTR(NAME)          \
        &NAME[plat_my_core_pos()]
```

- No changes internally

Multi-chip

```
#define PER_CPU_OFFSET(x) (x - PER_CPU_START)
#define THIS_CPU_PTR(NAME) __extension__                    \
        ((__typeof__(&NAME[0]))                             \
        (plat_per_cpu_section_base(plat_my_core_pos()) +    \
        PER_CPU_OFFSET(                                     \
  (uintptr_t)&NAME[plat_my_core_pos()%CHIPLET_CORE_COUNT])))
```

- Here is an example use-case

```
rmmd_rmm_context_t *ctx = THIS_CPU_PTR(rmm_context);
```

Proposal : NUMA Aware PER-CPU Framework

Op1 - tpidr_el3 magic

```
bl      6e698 <plat_my_core_pos>
mov     w19, w0
bl      75d14 <plat_per_cpu_section_base>
and     w19, w19, #0x3
```

- Can be optimized to something as simple as

```
mrs      x0 ,tpidr_el3
add      x0, x0, #0x9f8
```

- This should be multi-folds faster, even faster than an access from a cached pointer in case of a cache miss.
  - we rely on a system register to get the offset for a particular CPU
  - The unoptimized variant relies multiple memory accesses to calculate the right offset

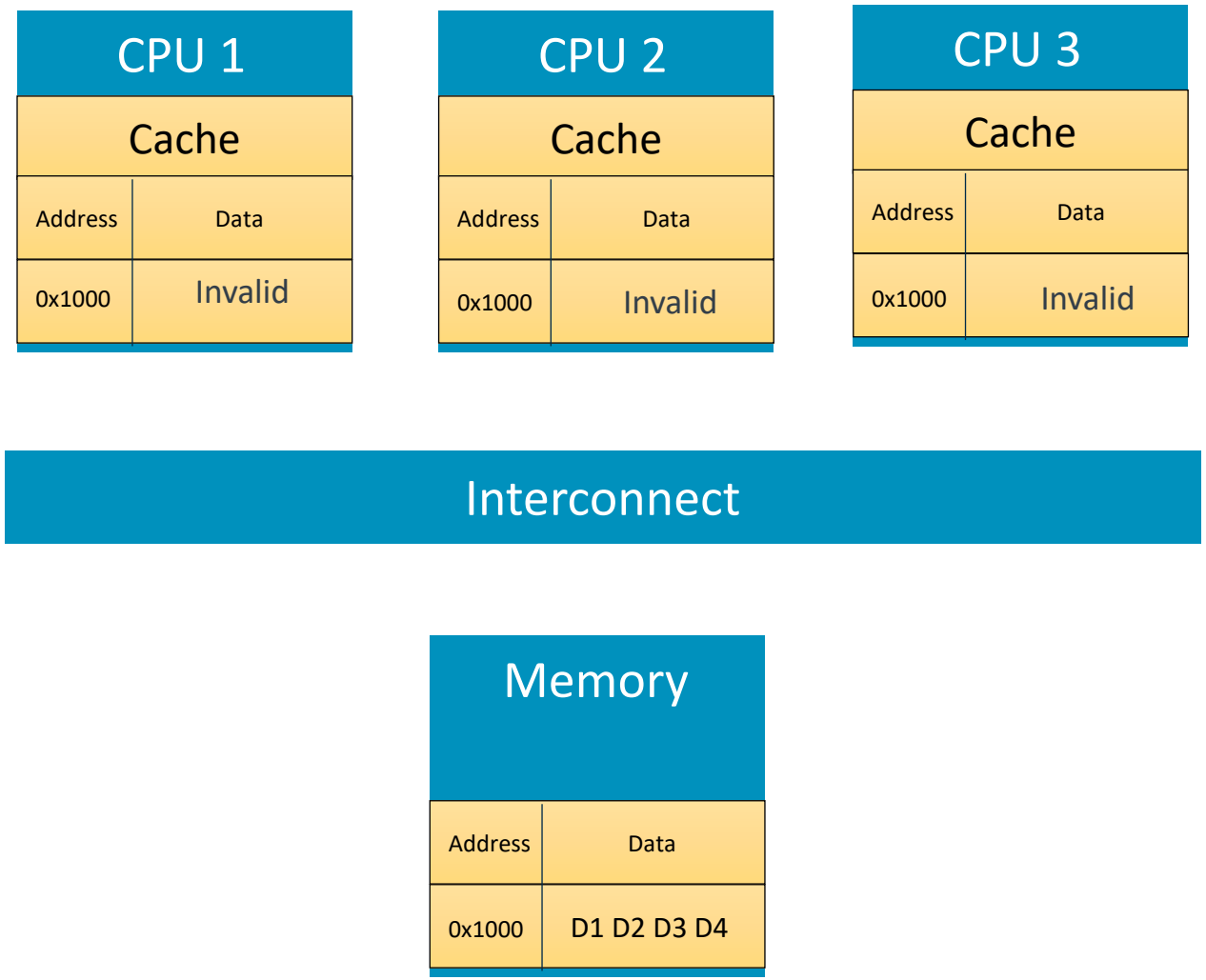
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



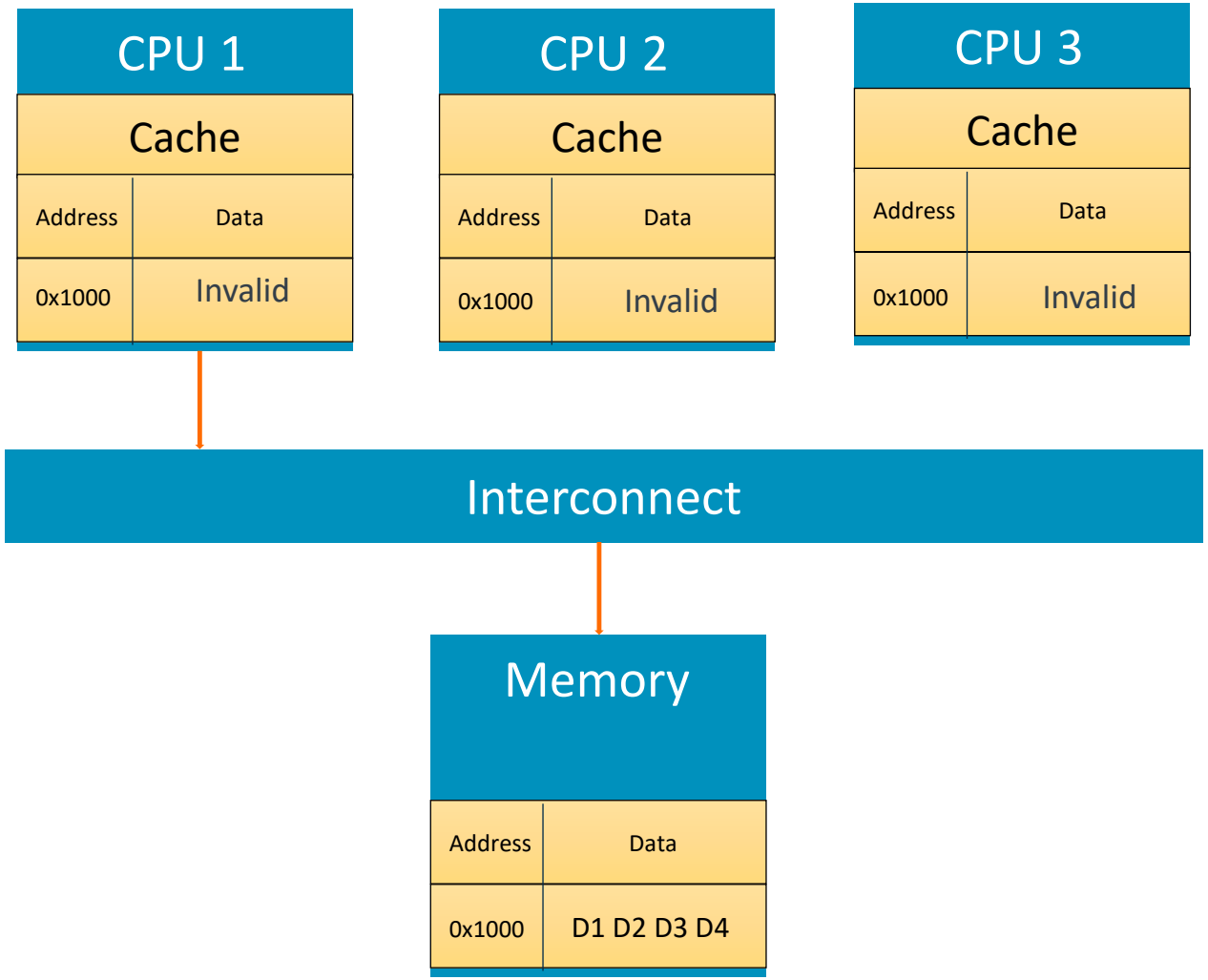
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



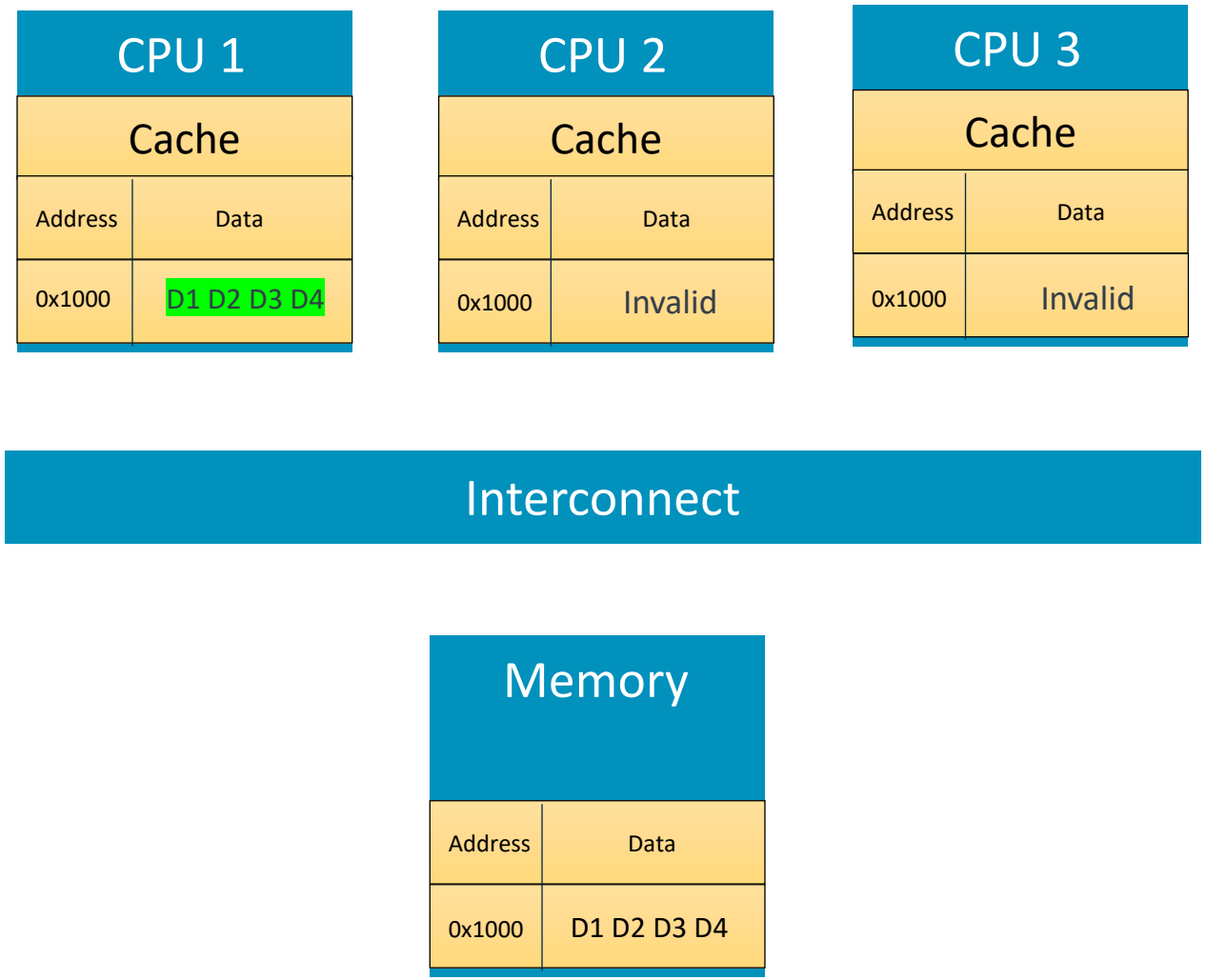
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



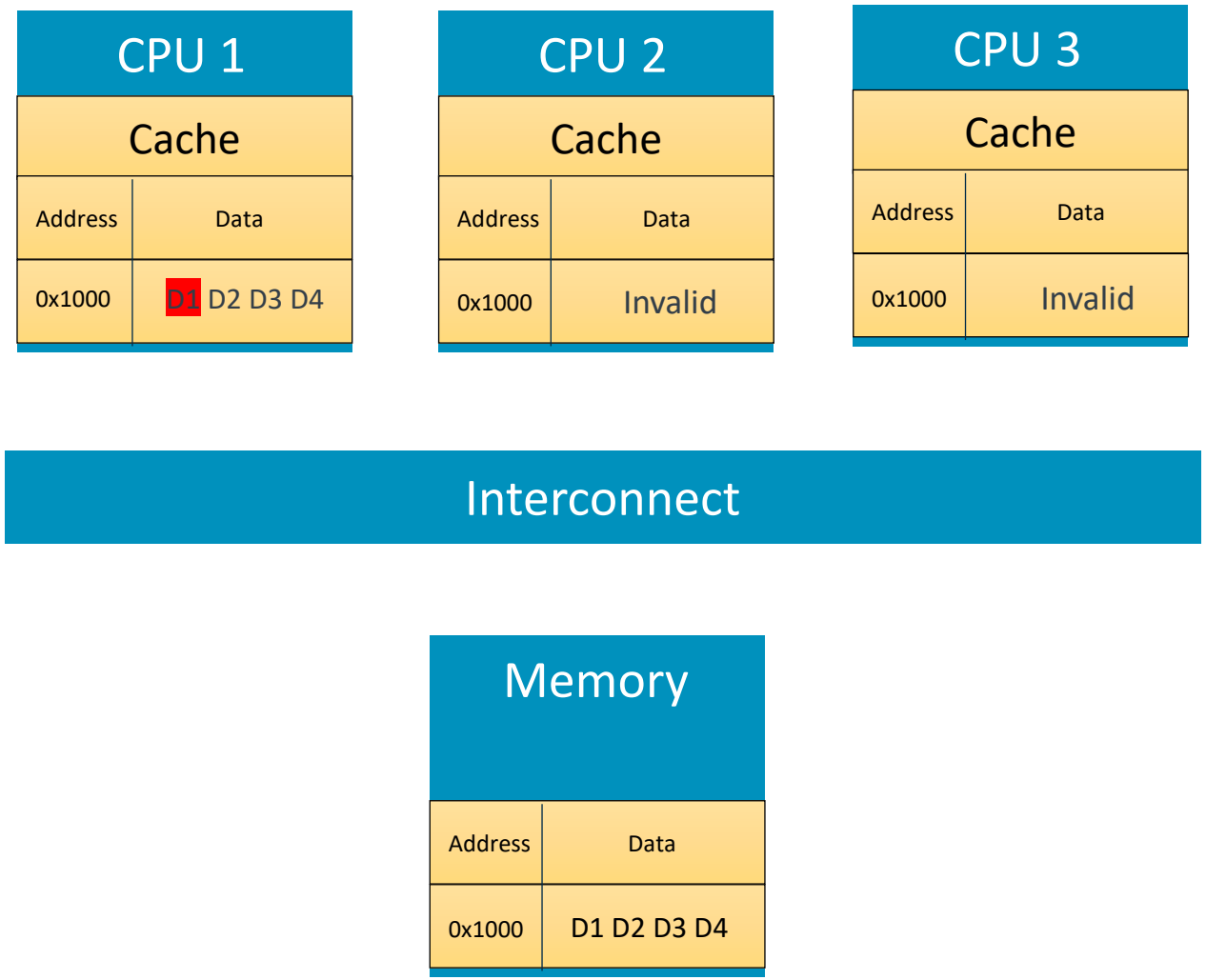
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



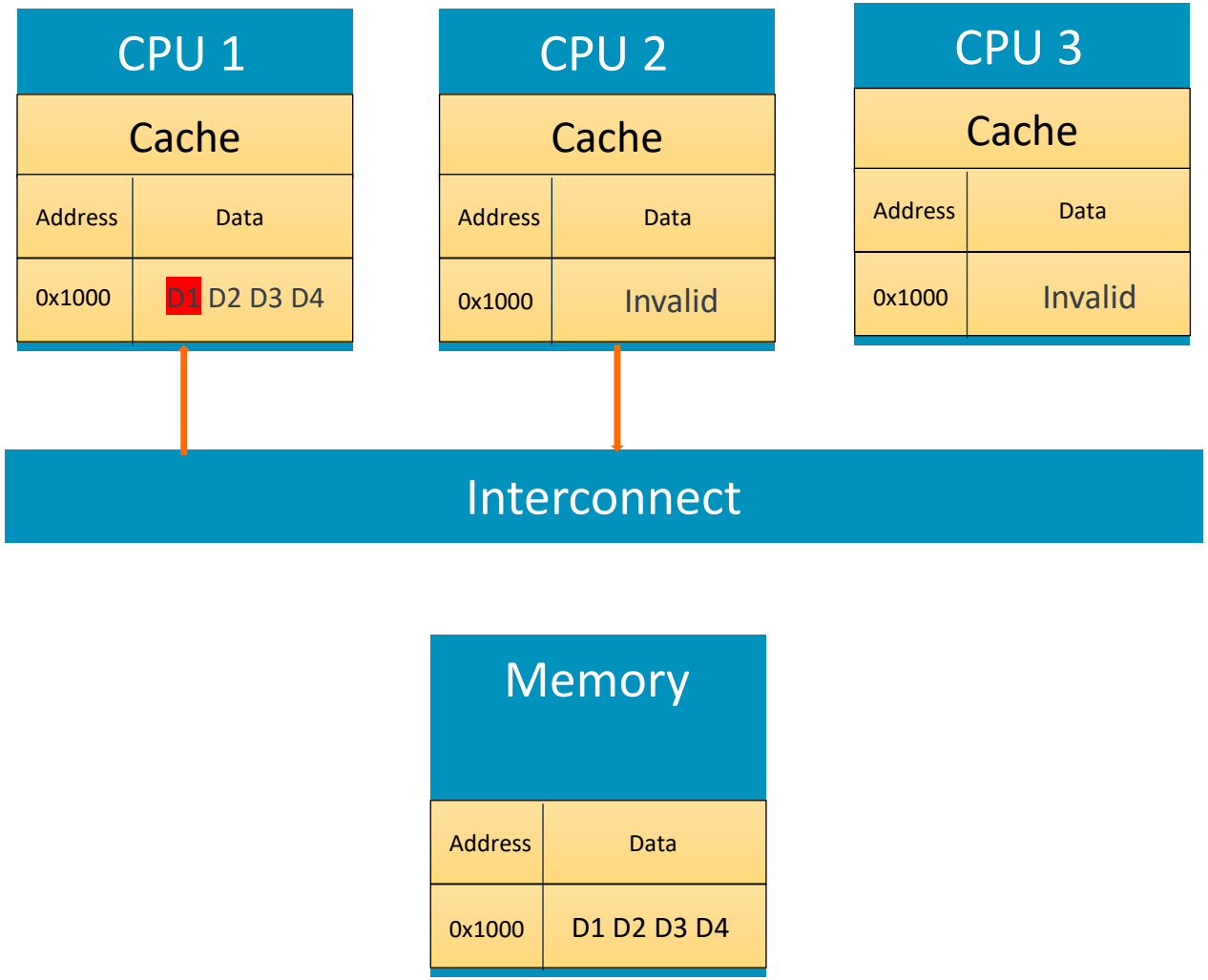
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



arm

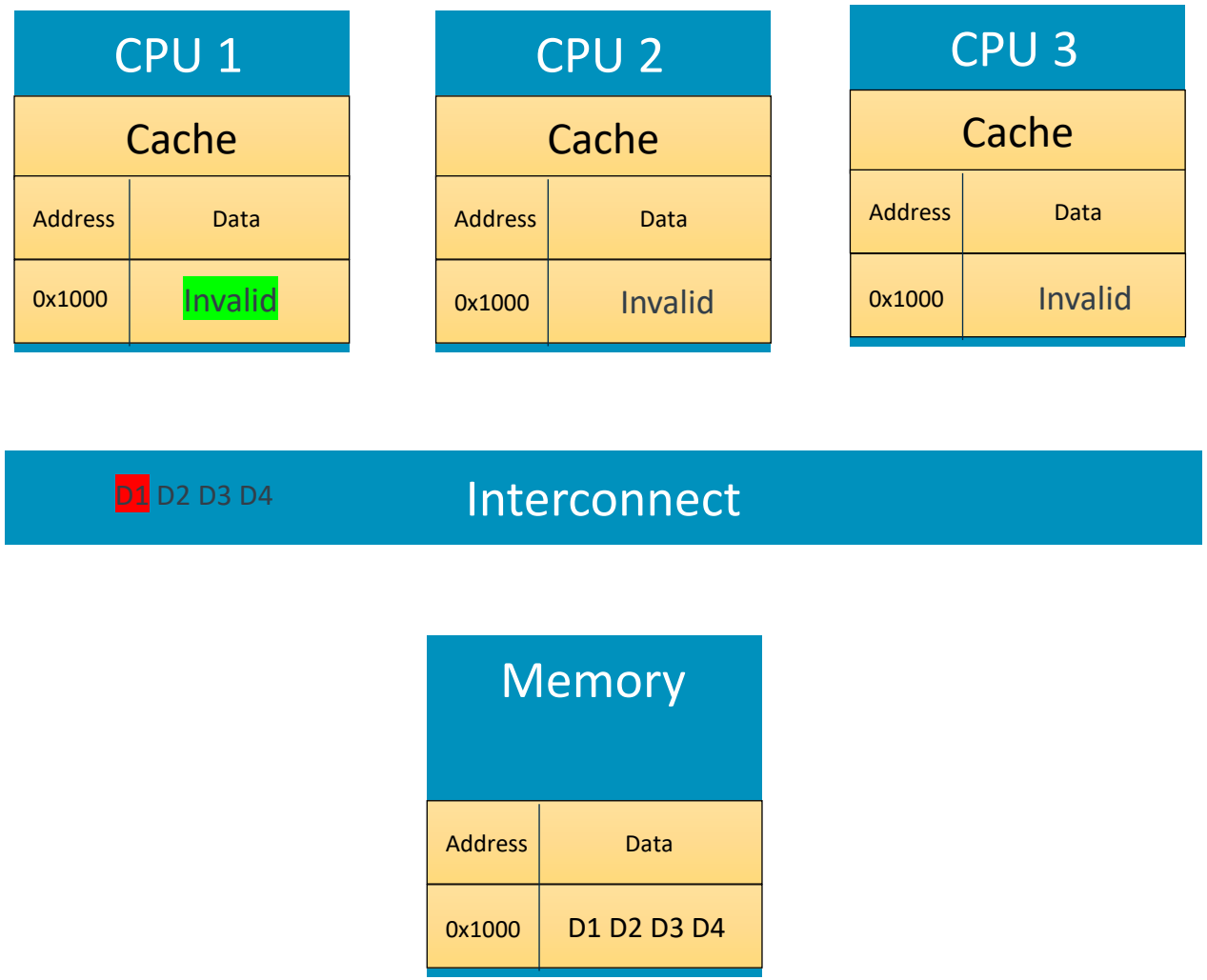
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



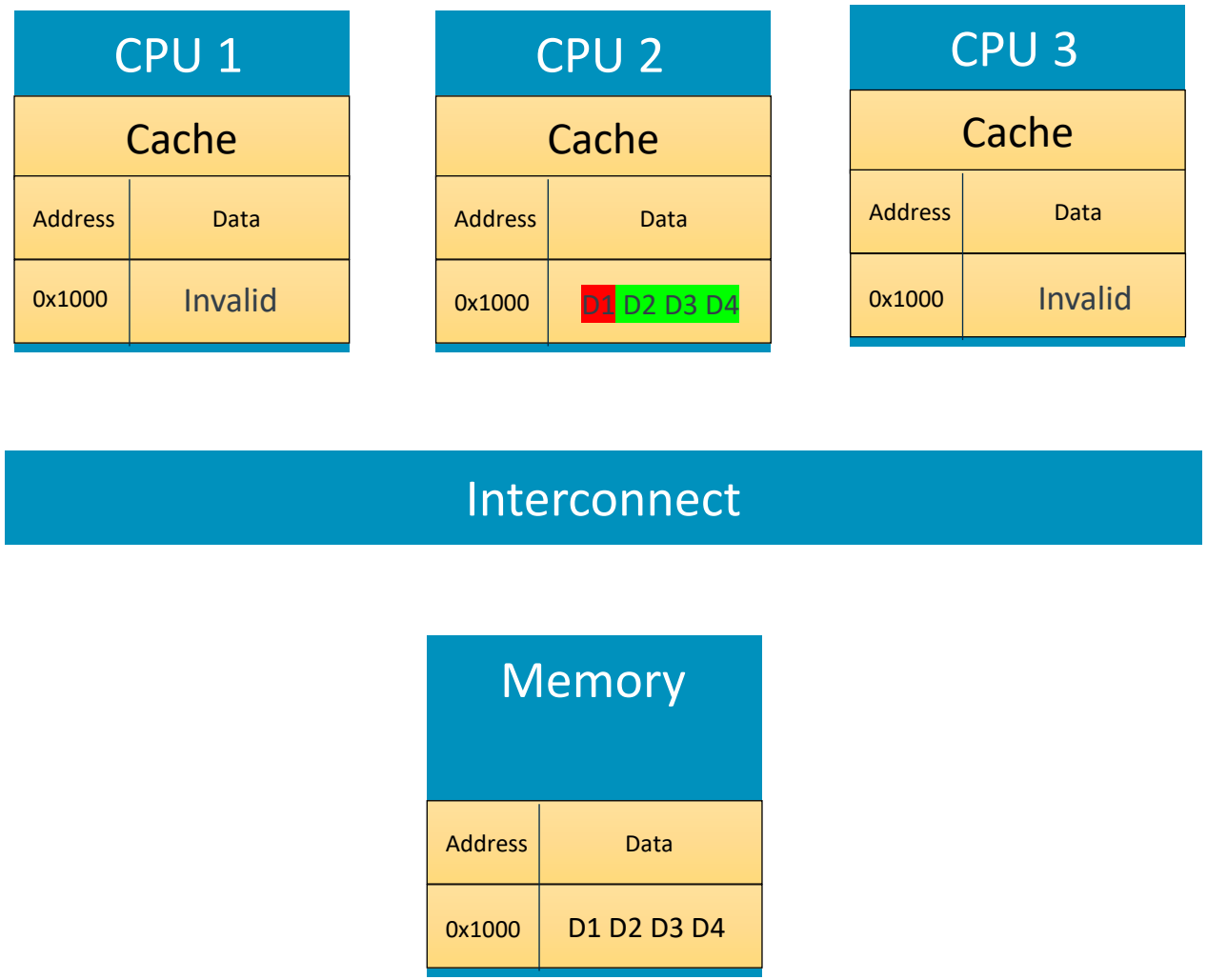
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



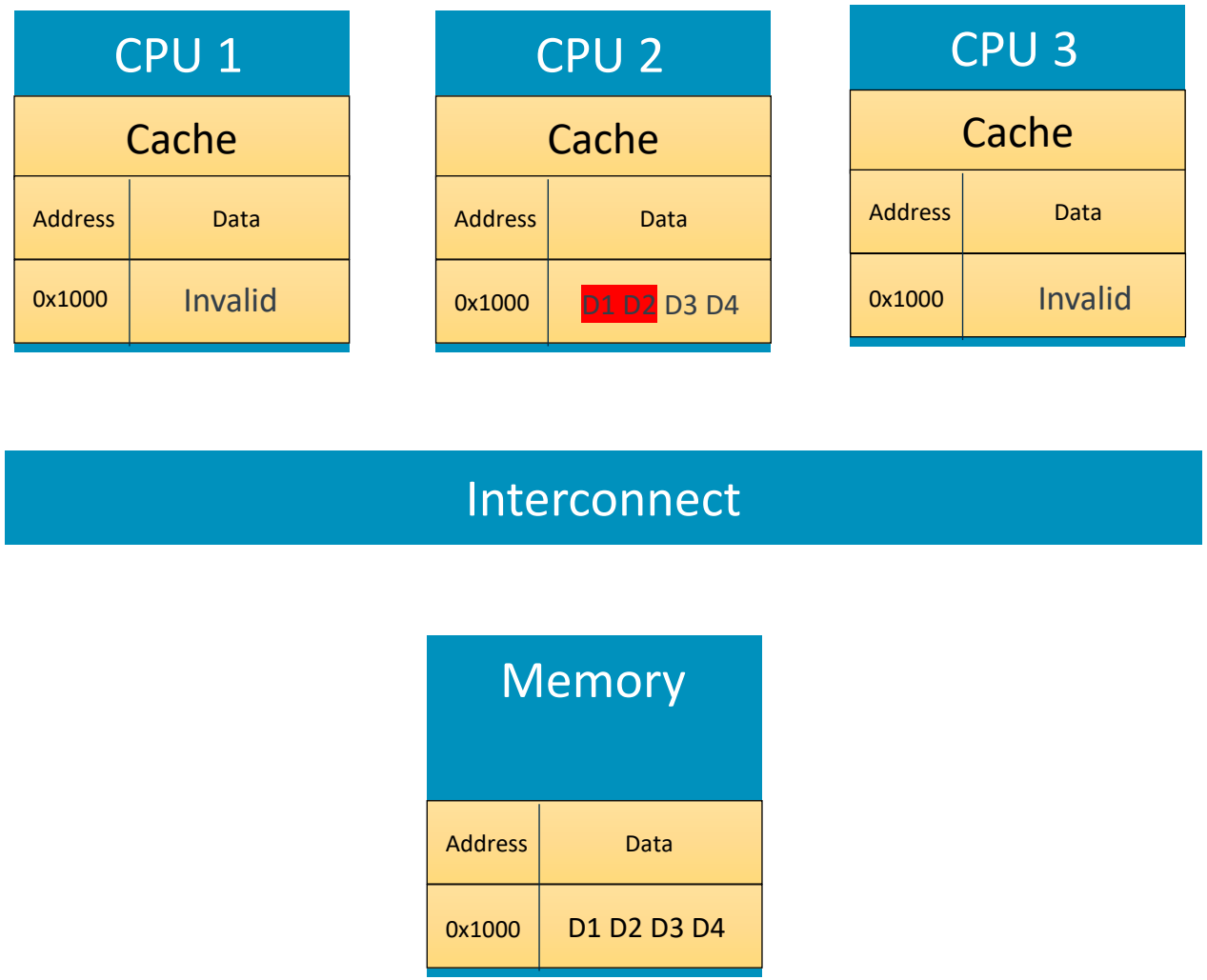
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



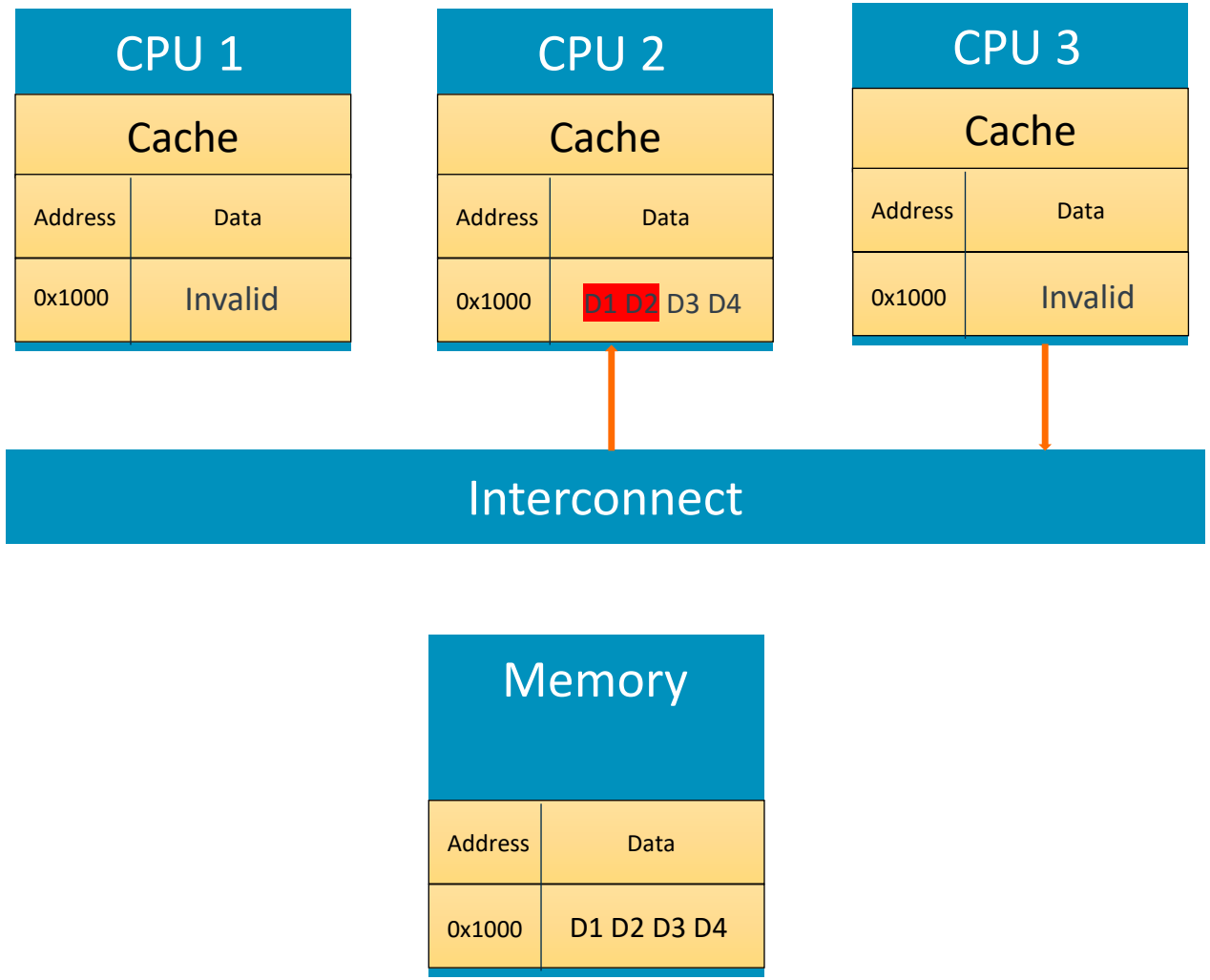
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



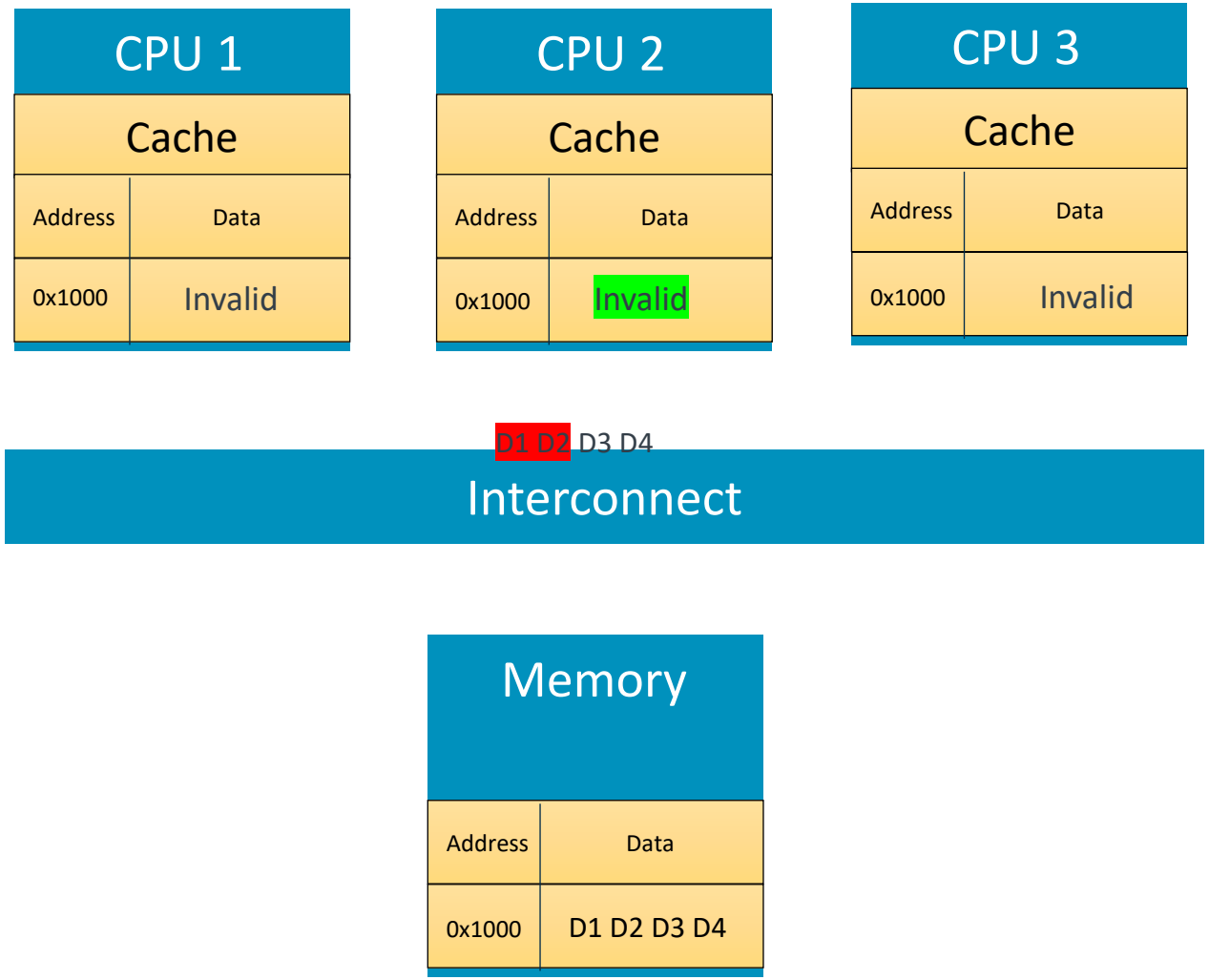
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



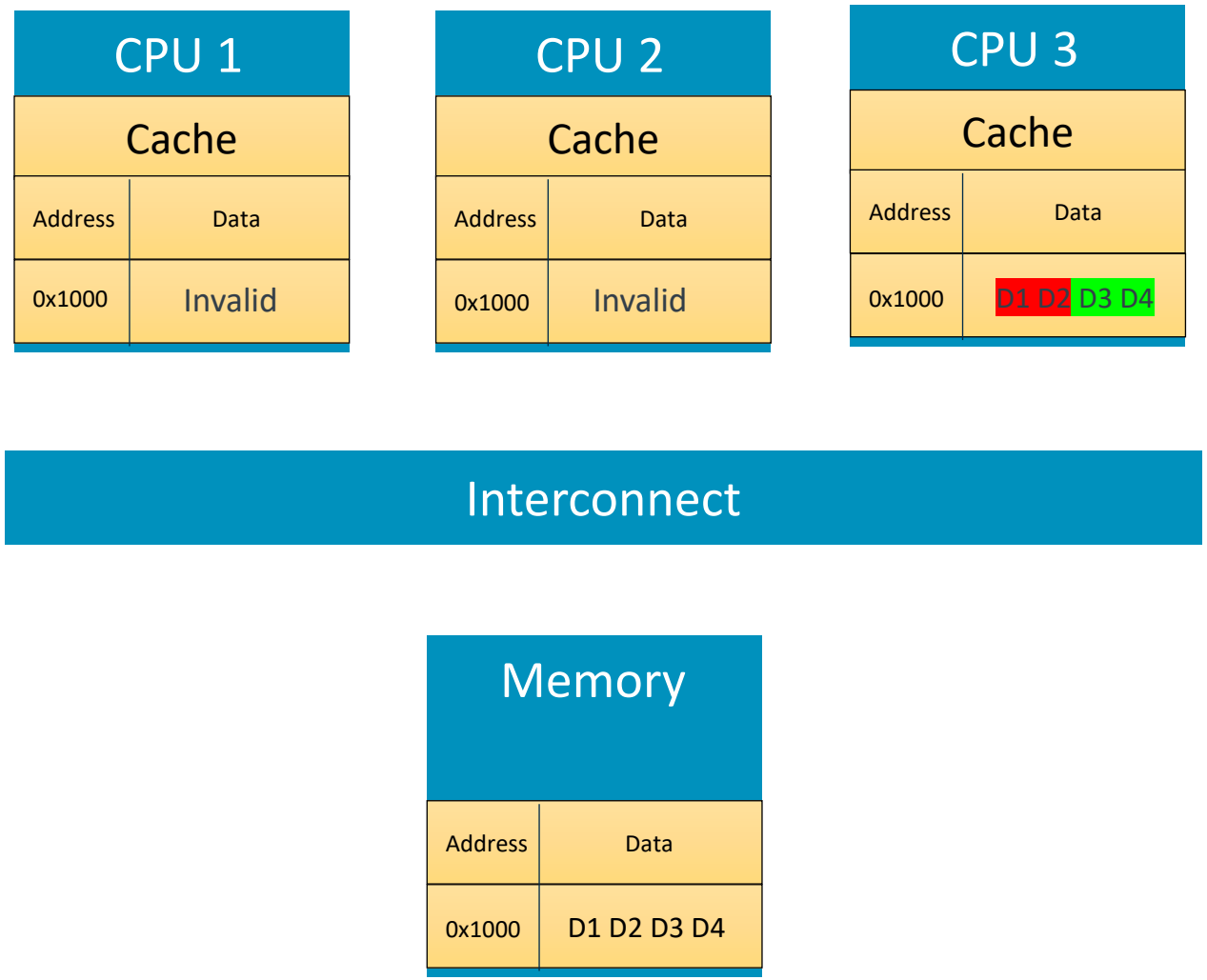
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



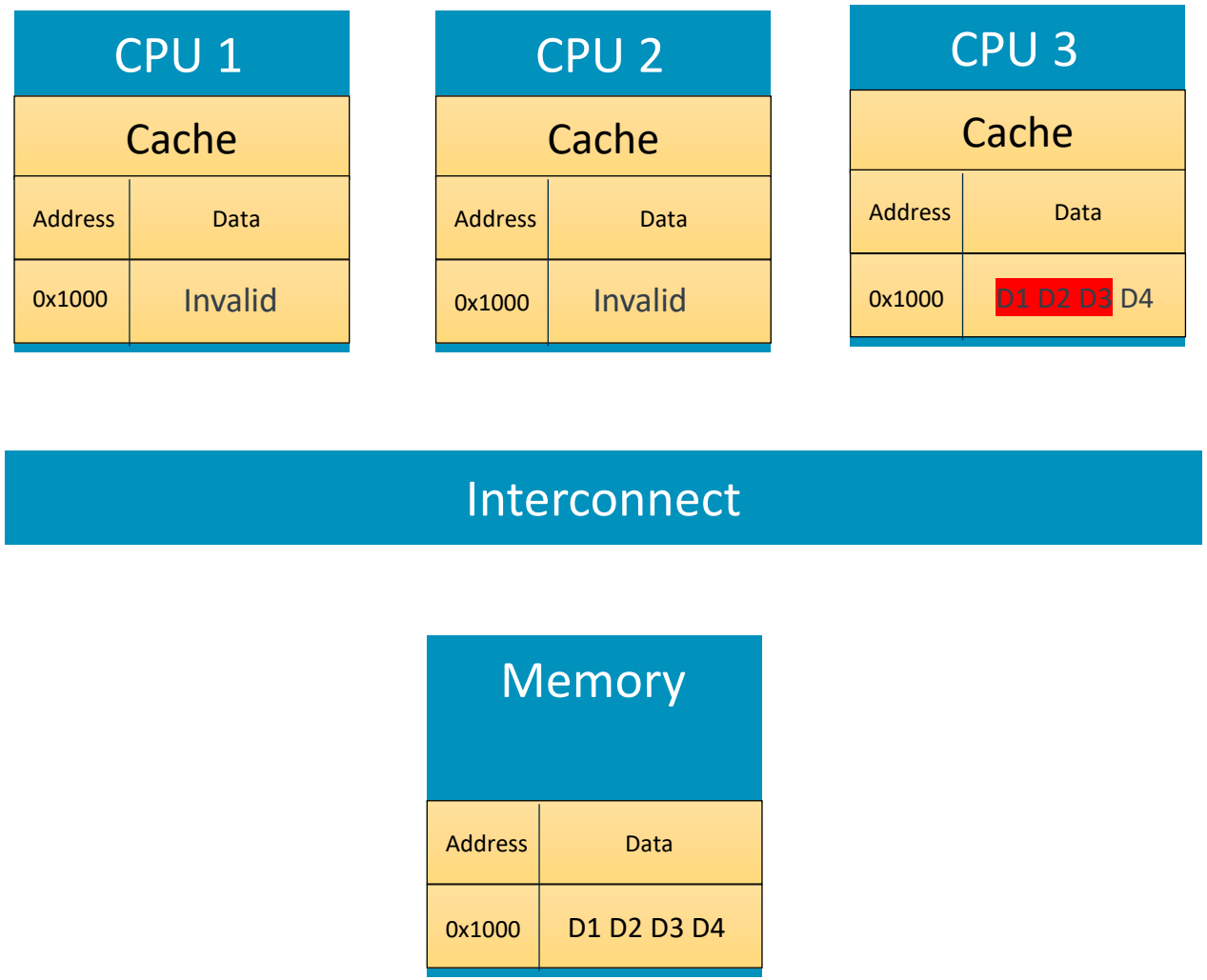
Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- Contiguous arrays *can* cause data for different CPUs to be residing on the same cache-line

```
TYPE NAME[CPU_MAX]
```

- This introduces false sharing or cache-thrashing where the ownership of the cache line keeps switching between different CPUs.



Proposal : NUMA Aware PER-CPU Framework

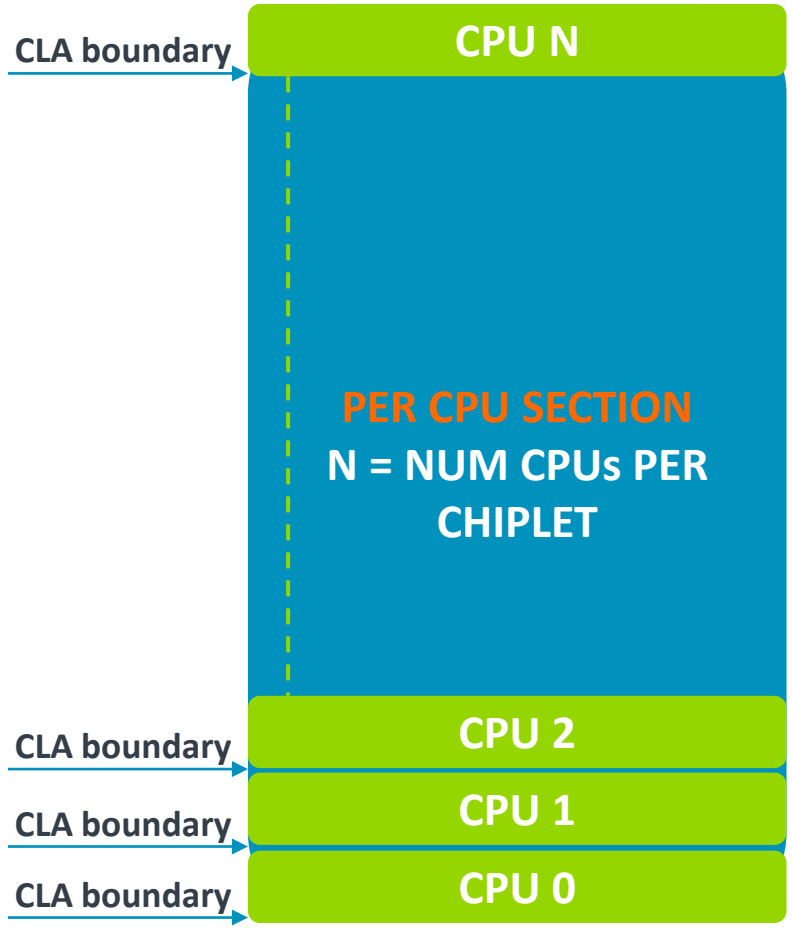
Op2 – Avoid Cache Thrashing

- If performance is key, modify the definition as follows

```
#define DEFINE_PER_CPU(TYPE, NAME)          \
        TYPE NAME                           \
        __section(PER_CPU_MULTICHIP_SECTION)
```

- In the linker script,

```
#define PER_CPU_SECTION                                  \
.per_cpu : ALIGN(CACHE_WRITEBACK_GRANULE) {              \
__PER_CPU_START_UNIT__ = .;                              \
*(SORT_BY_ALIGNMENT(.per_cpu*))                          \
__PER_CPU_END_UNIT__ = .;                                \
. = ALIGN(CACHE_WRITEBACK_GRANULE);                      \
__PER_CPU_END_UNIT_CLA = .;                              \
. = ((NUM_CPUS_PER_CHIPLET - 1)                          \
*(__PER_CPU_END_UNIT_CLA - __PER_CPU_START_UNIT__)       \
. = . + __PER_CPU_END_UNIT__ - __PER_CPU_START_UNIT__)\
}
```



Proposal : NUMA Aware PER-CPU Framework

Op2 – Avoid Cache Thrashing

- The extra cache-line alignment coupled with breaking down the array would avoid the same cache-line to exist in multiple CPUs.
- *Could* take up a bit more storage as alignment is costly.
  - Change in Definer and Accessor implementation. Interface should be same
- single-chip could be kept untouched; however, this would be a better design if performance is of priority. (Remember **)
  - **multi-CPU problem** and not a multi-chip one!

Proposal : NUMA Aware PER-CPU Framework

Migration of Stack

- Stack as of today is using its own section **“.tzfw_normal_stacks”** and is a big consumer like the other context globals.
- Plan to move the stack to the PER-CPU framework as we progress with the migration
- At a high level this would mean:
  - Removing the stack section from BL31 linker script
  - Stack would now be defined by the framework
  - SP is switched via the accessor interface

Proposal : NUMA Aware PER-CPU Framework

Interface variants

- For both `FOR_CPU_PTR` and `THIS_CPU_PTR`, it would be beneficial to have a non-pointer/object accessor interface (`FOR_CPU/THIS_CPU`).
  - Eg: `FOR_CPU(spm_core_context, core_id).state = SPMC_STATE_OFF;`

- Definer interface should also support **aligned definitions**, for definitions requiring tighter alignments.
  - Eg: `__aligned (64) some_struct_t some_struct[PLATFORM_CORE_COUNT];`
  - Could be defined as `DEFINE_PER_CPU_ALIGNED(some_struct_t, some_struct, 64)`
  - .per_cpu section has to be aligned to the max of (SORT_BY_ALIGNEMENT(.per_cpu))

- Support for arrays
  - Eg: `uint64_t shadow_registers[16][PLATFORM_CORE_COUNT];`

Proposal : NUMA Aware PER-CPU Framework

Interface variants

- Support for **initialized PER-CPU variables** could be a use-case we should support
  - Eg: `./plat/st/common/stm32mp_gic.c:static unsigned int target_mask_array[PLATFORM_CORE_COUNT] = {1, 2};`
- Possibly useful to add support in BL32
  - Eg:`./bl32/tsp/tsp_timer.c:static timer_context_t pcpu_timer_context[PLATFORM_CORE_COUNT];`
- This would be a long-term activity where less crucial objects can be migrated down the line.

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు