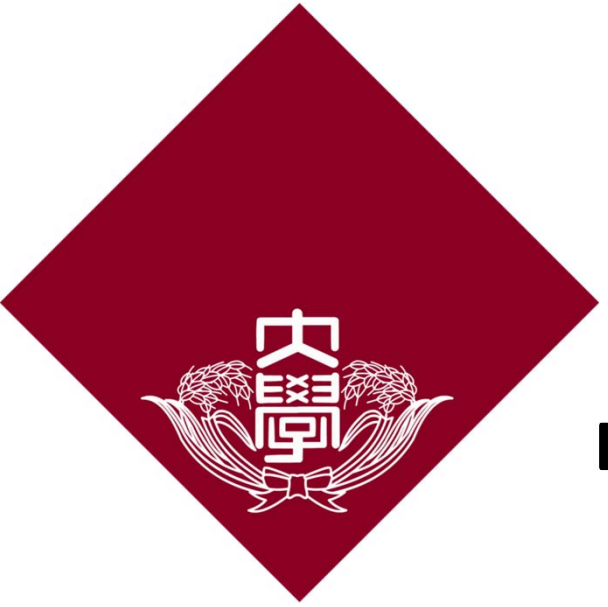
Multicore Cache Coherence Control by a Parallelizing Compiler



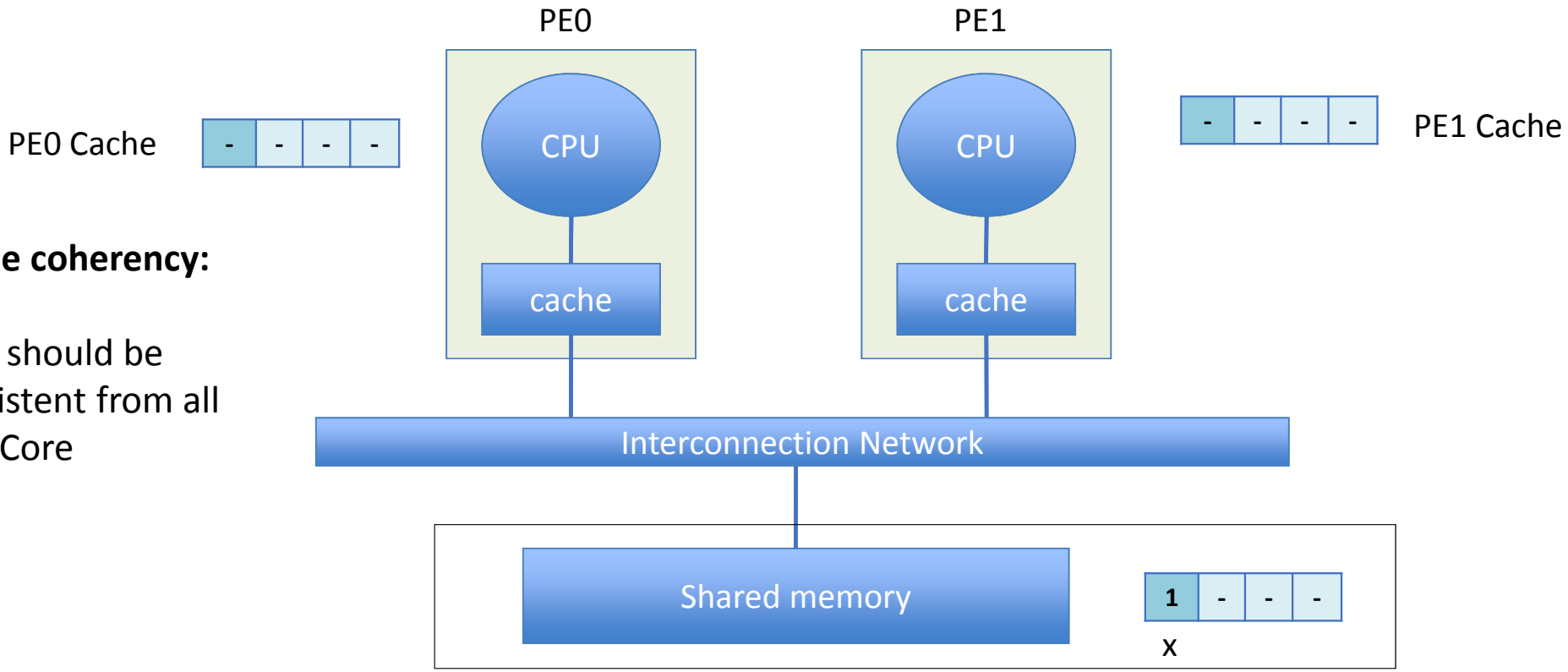
Hironori Kasahara, Keiji Kimura,
Boma A. Adhi, Yuhei Hosokawa
Yohei Kishimoto, Masayoshi Mase

Department of Computer Science and Engineering
Waseda University

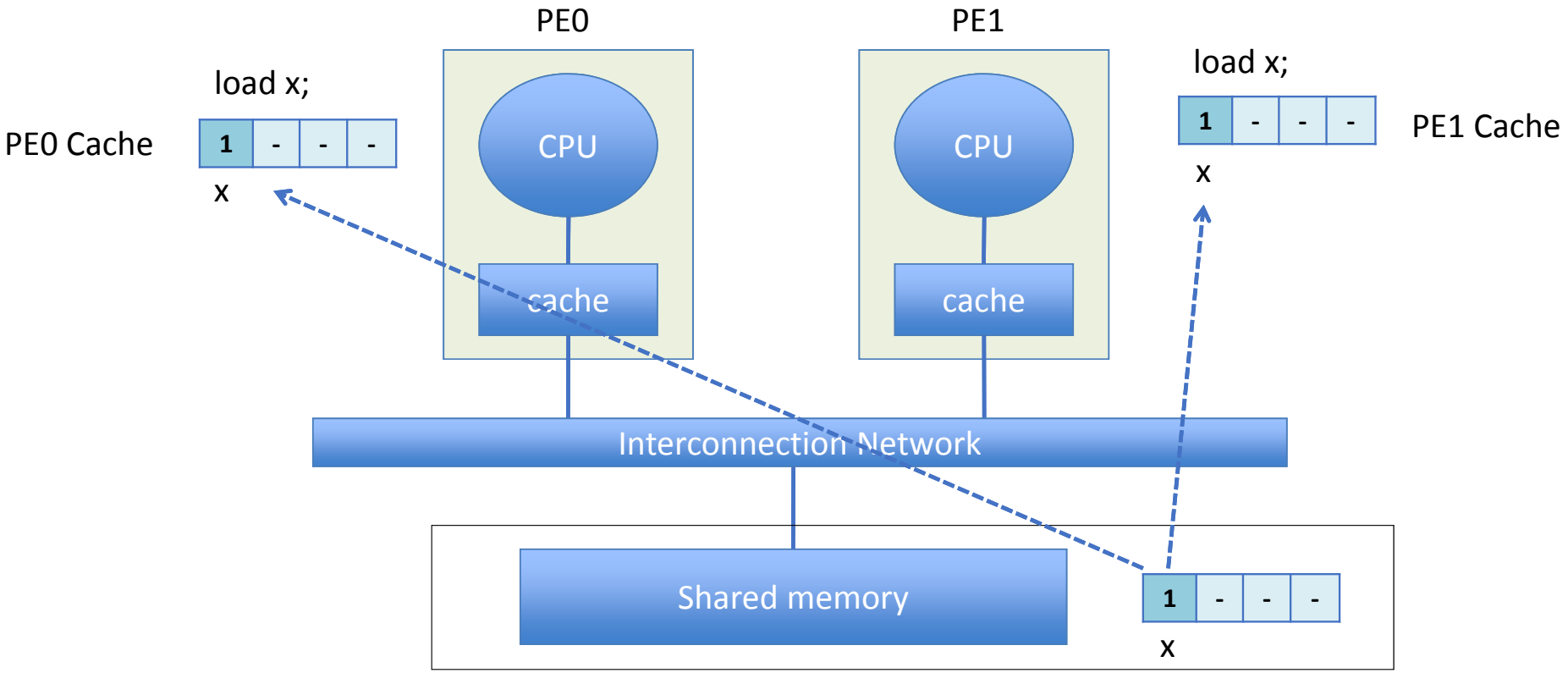
What is Cache Coherency?

Cache coherency:

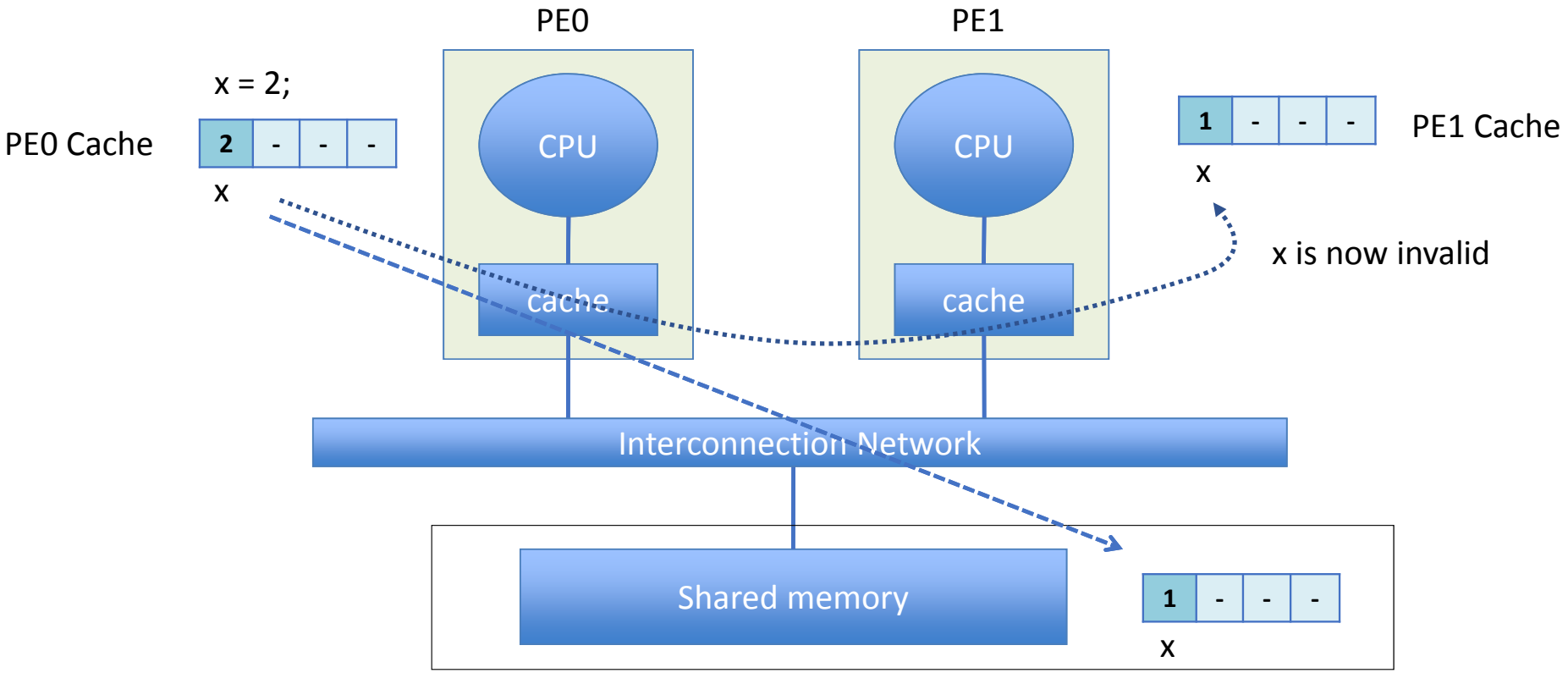
Data should be consistent from all CPU Core



What is Cache Coherency?



What is Cache Coherency?



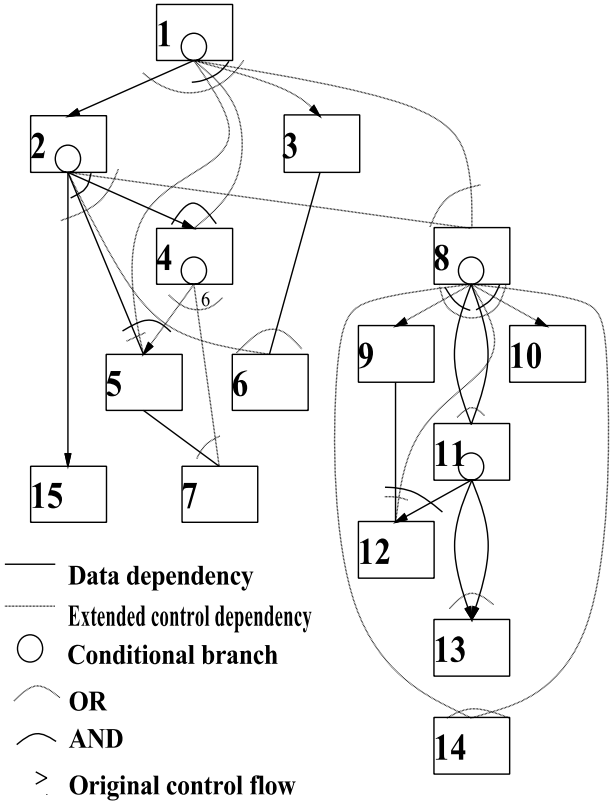
Why Compiler Controlled Cache Coherency?

Current many-cores CPU (Xeon Phi, TilePro 64, etc) uses hardware cache coherency mechanism.

- Hardware cache coherency for many-cores (hundreds to thousand cores) :
  - Hardware will be complex & occupy large area in silicon
  - Huge power consumption
  - Design takes long time & larger cost
- Software based coherency:
  - Small hardware, low power & scalable
  - No efficient software coherence control method has been proposed.

Proposed Software Coherence Method on OSCAR Parallelizing Compiler

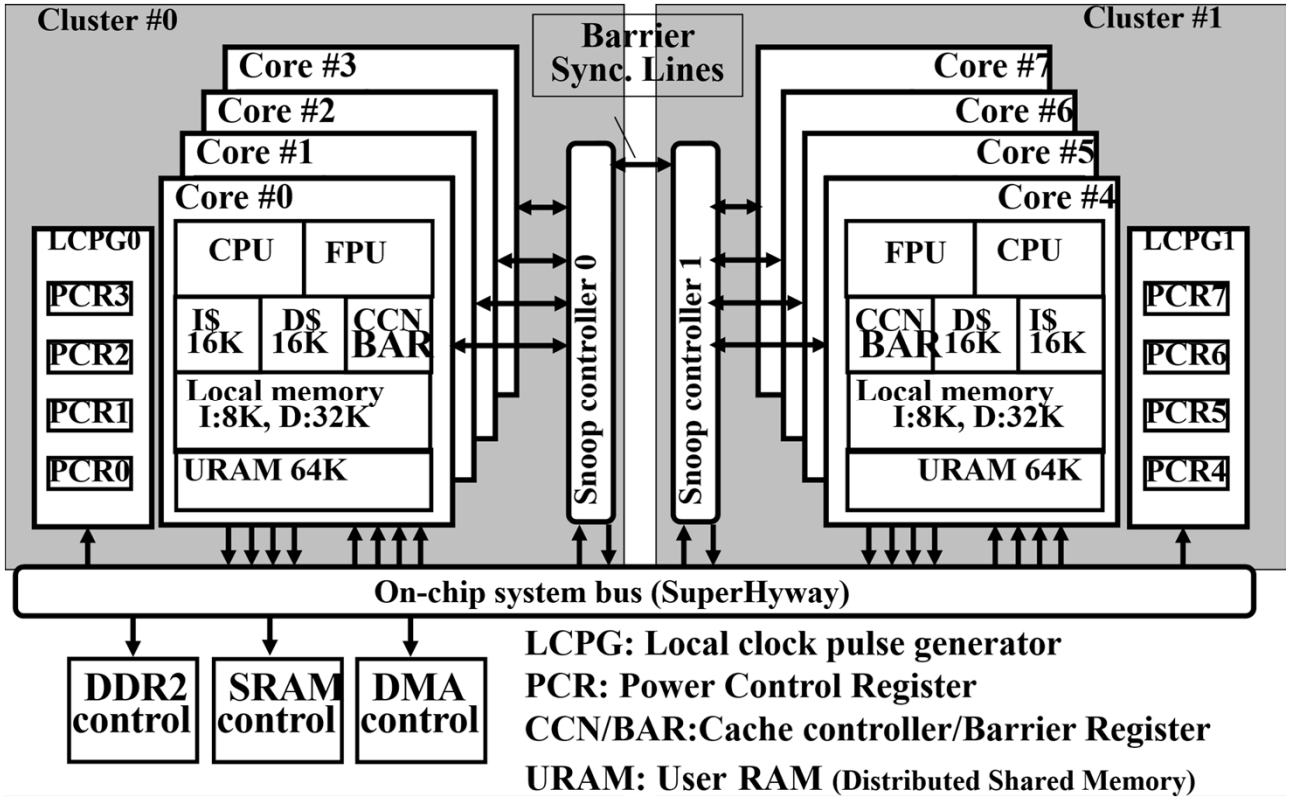
- Coarse grain task parallelization with earliest condition analysis (control and data dependency analysis).
- OSCAR compiler automatically controls coherence using following simple program restructuring methods:
  - To cope with stale data problems:
    - **Data synchronization by compilers**
  - To cope with false sharing problem:
    - **Data Alignment**
    - **Array Padding**
    - **Non-cacheable Buffer**



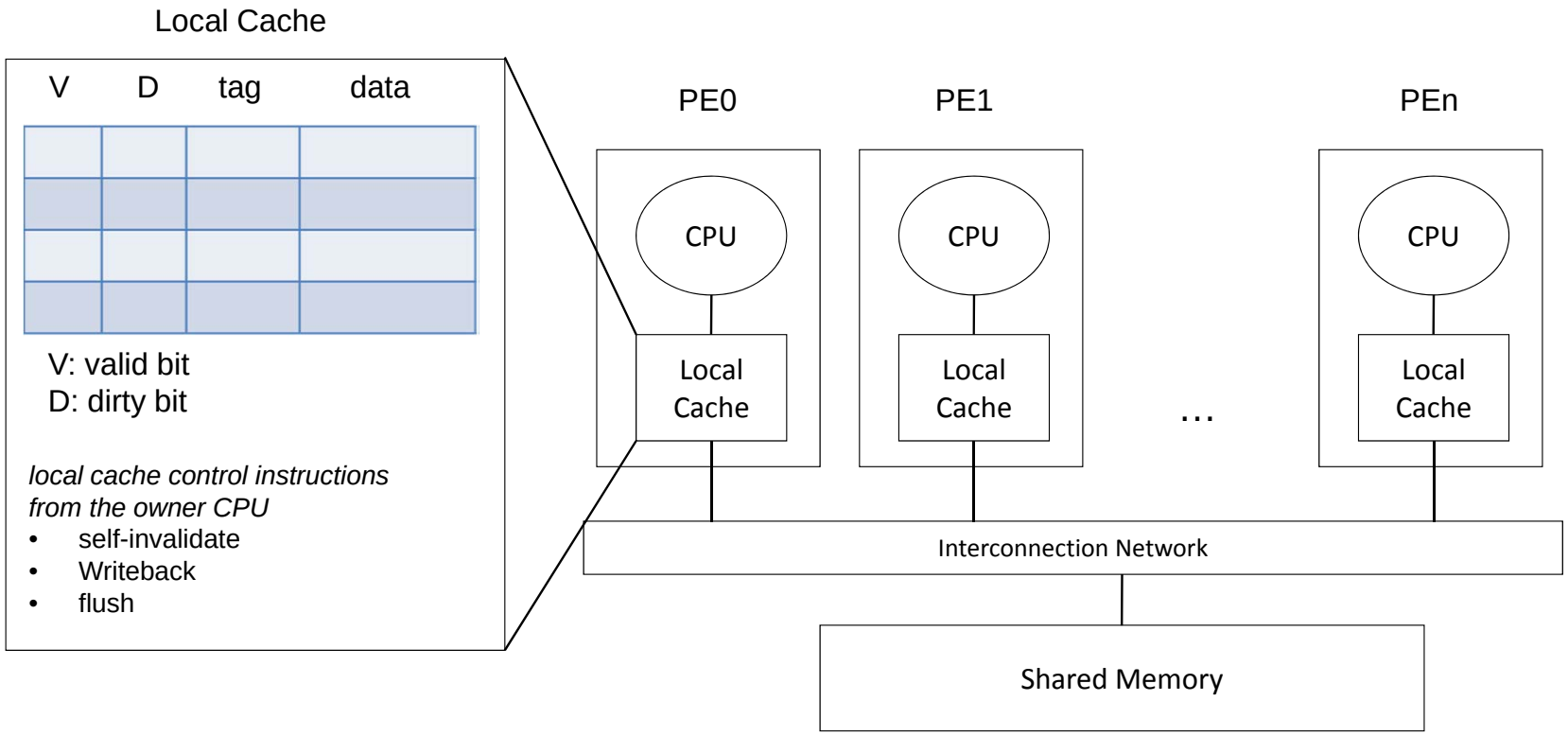
MTG generated by earliest executable condition analysis

The Renesas RP2 Embedded Multicore

2 clusters of 4 cores hardware cache coherent control SMP,
No coherence support for more than 5 cores



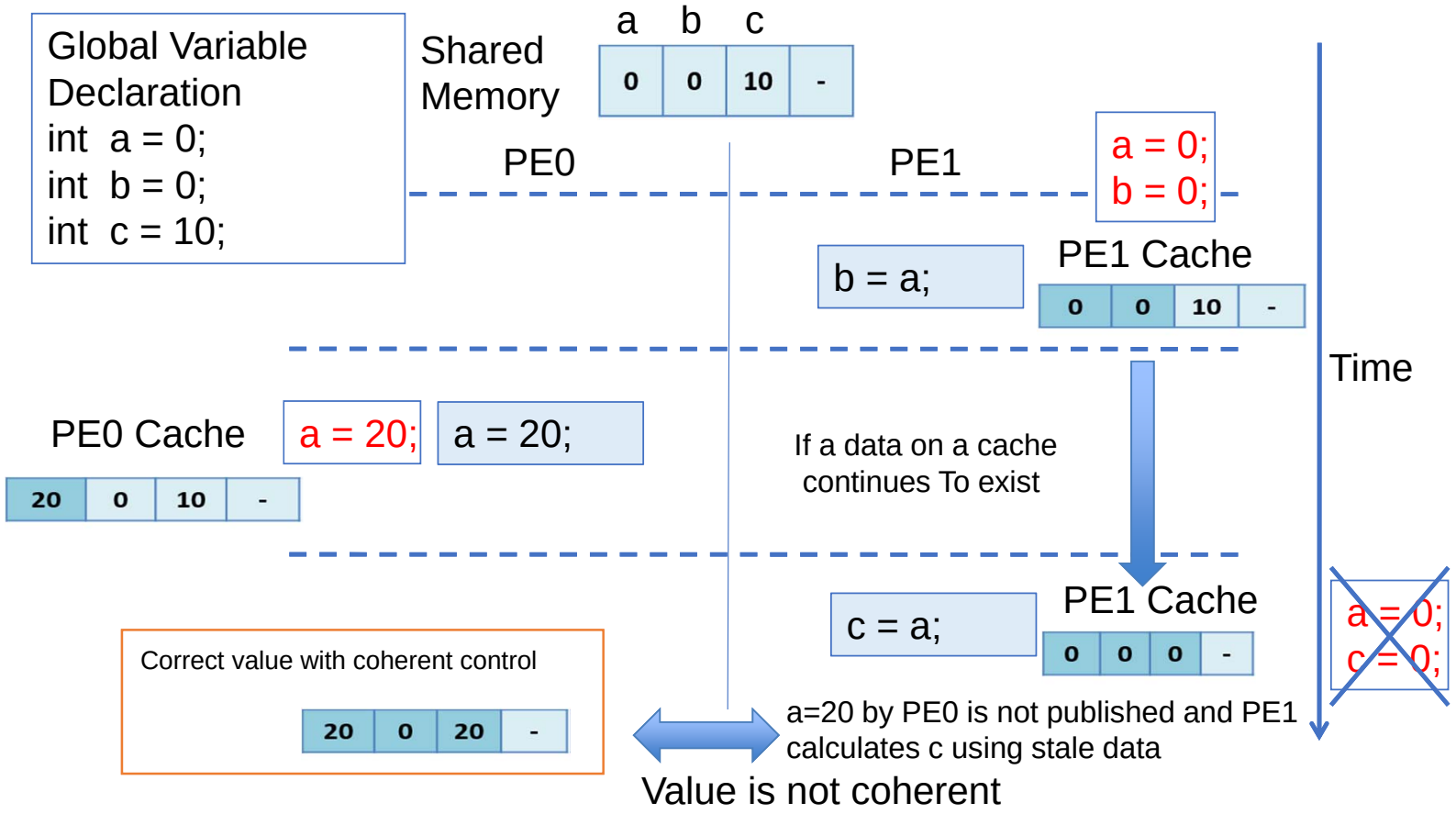
Non Coherent Cache Architecture



Problem in Non-coherent Architecture (1): Stale Data

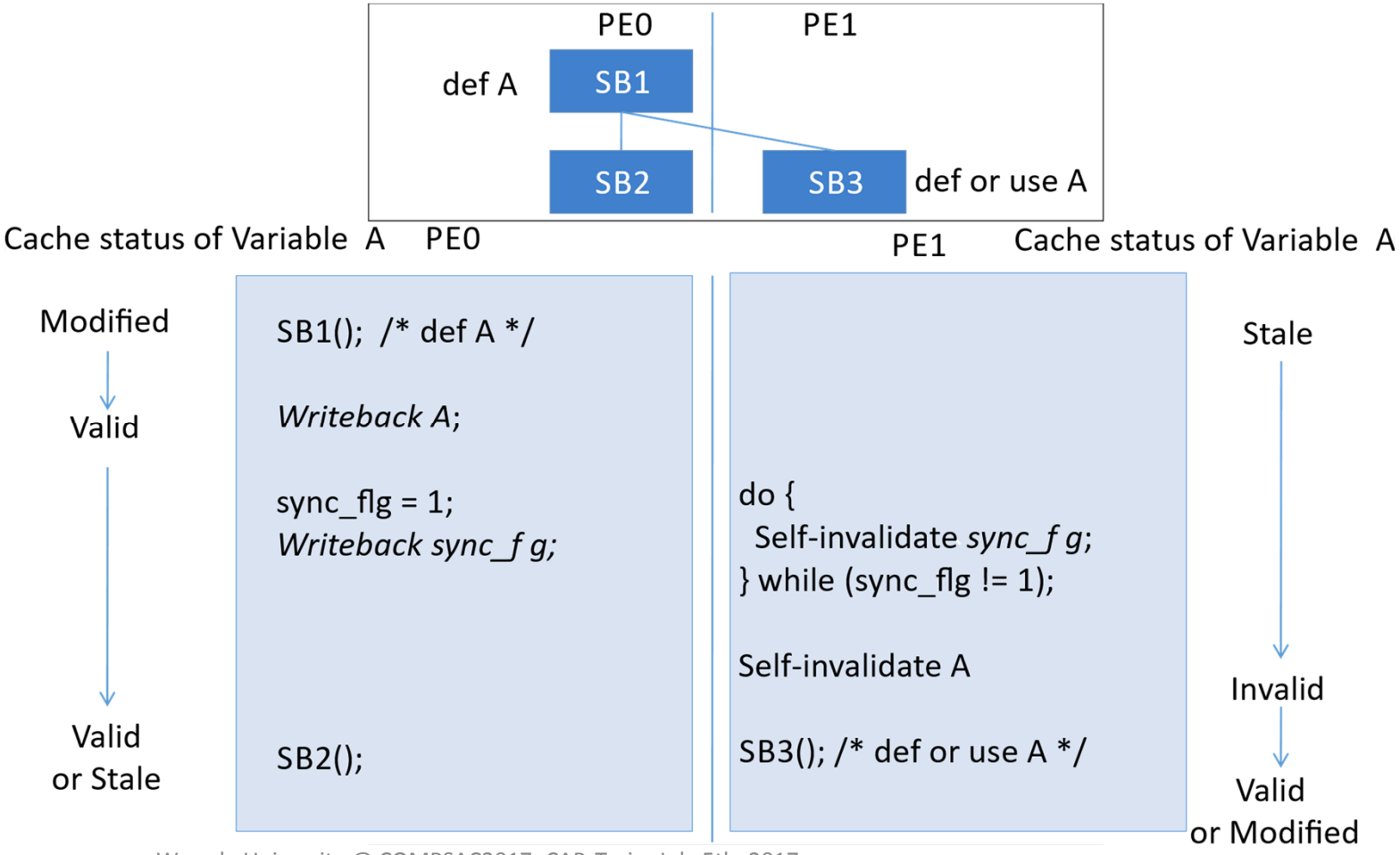
Stale Data:

An obsolete shared data exists on a processor cache after a new value is updated by another processor core



Solving Stale Data Problem

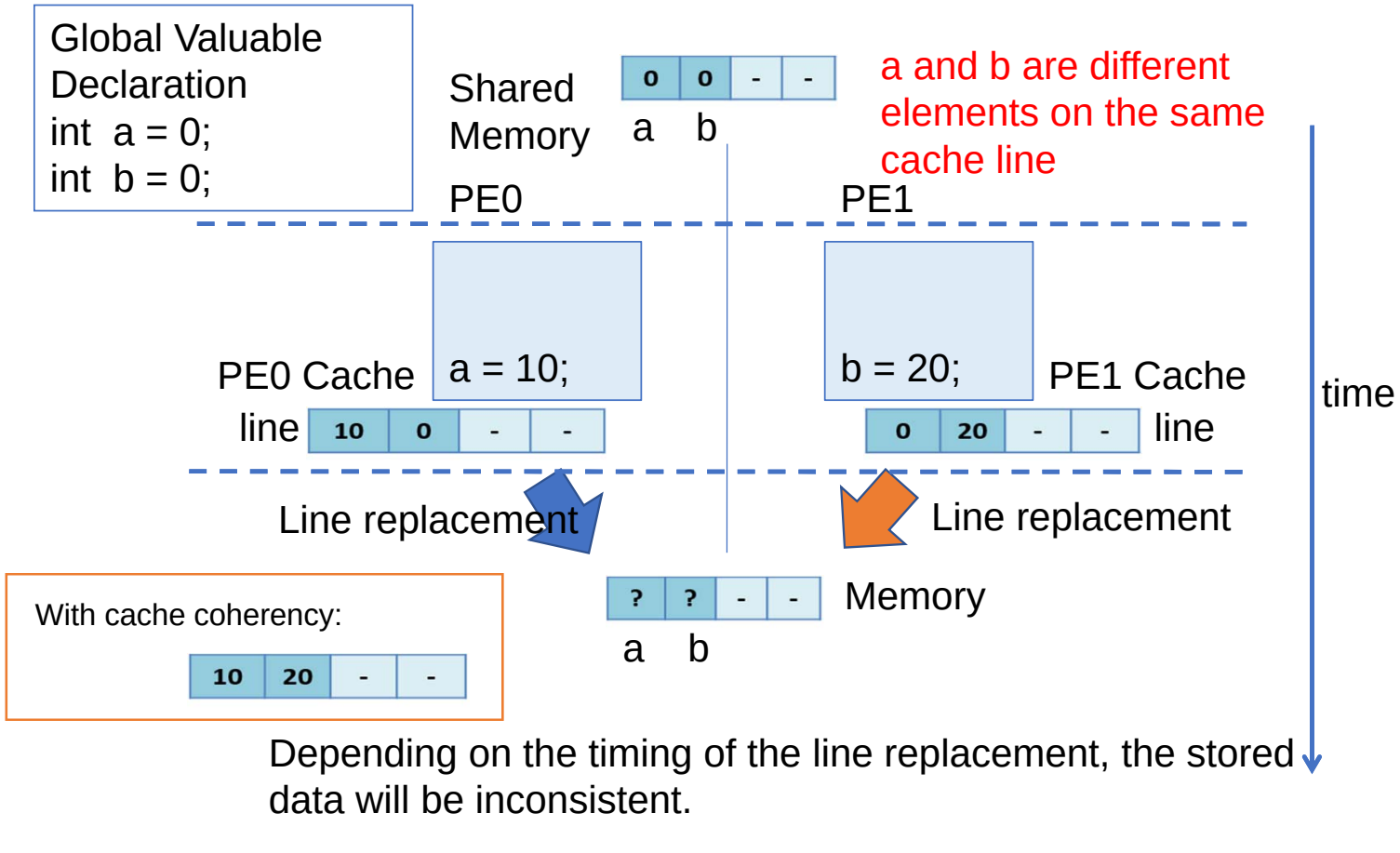
The compiler automatically inserts a writeback instruction and a synchronization barrier on PE0 code.
Then, it inserts a self-invalidate instruction on PE1 after the synchronization barrier.



Problem in Non-coherent Architecture (2): False Sharing

False sharing:

A condition which multiple cores share the same memory block or cache line. Then, each processor updates the different elements on the same block/cache line.



Solving False Sharing (1): Data Alignment

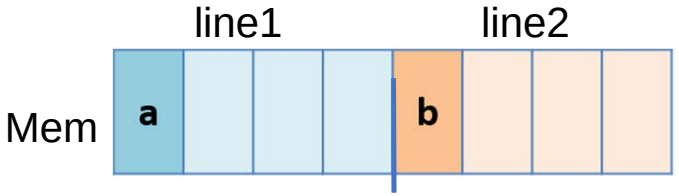
Data alignment solve the false sharing problem.

Different data accessed by different PE should **not** share a single cache line.

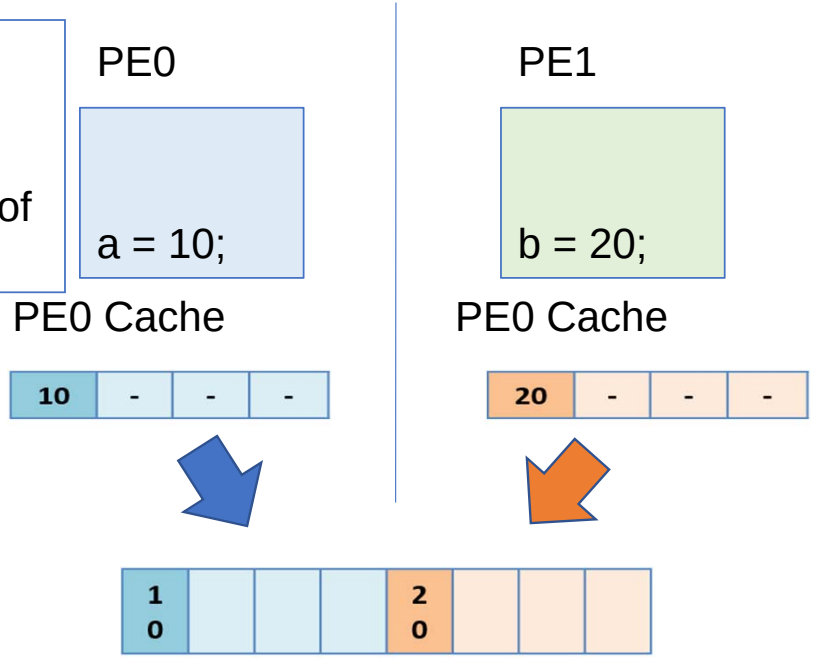
This approach works for scalar variables and small-sized one-dimensional array.

```
Variable Declaration
int  a __attribute__((aligned(16))) = 0;
int  b __attribute__((aligned(16))) = 0;
/* a, b are assigned to the first element of
each cache line */
```

a and b are assigned to the first elements of different cache lines

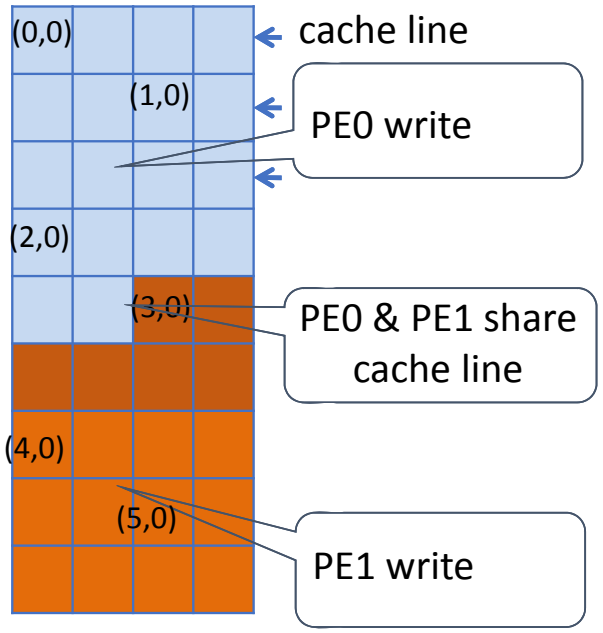


Updates by PE0 and PE1 are separately performed by write back



Two-dimensional Array Problem

- Splitting an array cleanly along the cache line is not always possible.
  - Lowest dimension is not integer multiply of cache line




```
int  a[6][6];
```

PE0

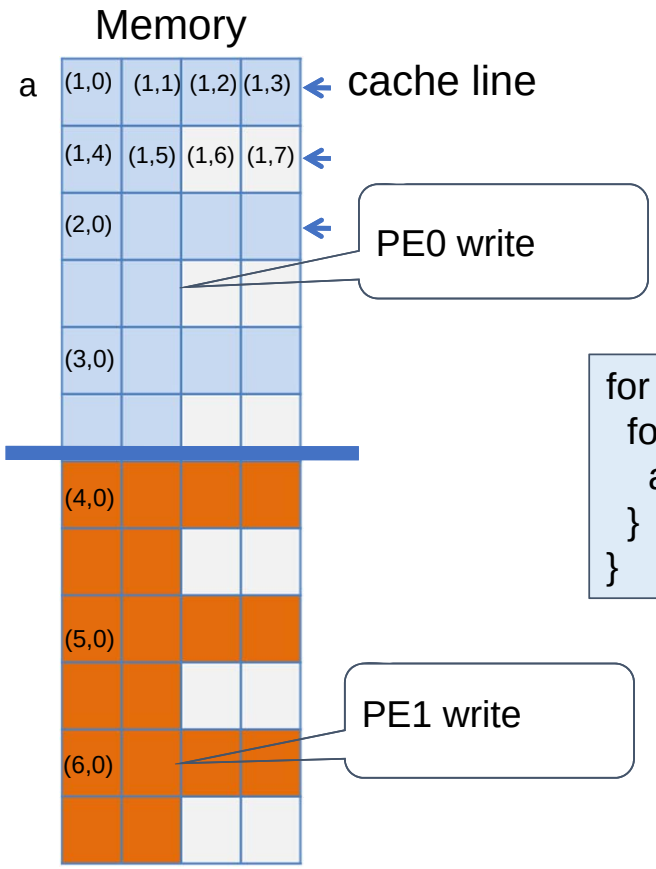
```
for (i = 0; i < 3; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = i * j;
  }
}
```

PE1

```
for (i = 3; i < 6; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = i * j;
  }
}
```

Solving False Sharing (2): Array Padding

The compiler inserts a padding (dummy data) to the end of the array to match the cache line size




```
int  a[6][8] __attribute__((aligned(16)));
/* a is aligned to the first element of the
cache line */
```

PE0

```
for (i = 0; i < 3; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = i * j;
  }
}
```

PE1

```
for (i = 3; i < 6; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = i * j;
  }
}
```

Solving False Sharing (3): Non-cacheable Buffer

Data transfer using non-cacheable buffer:
The idea is to put a small area in the main memory that should not be copied to the cache along the border between area modified by different processor core.

```
int  a[6][6] __attribute__((aligned(16)));
/* a is assigned to the first element of the cache line */
int nc_buf[1][6] __attribute((section("UNCACHE")));
/* nc_buf is prepared in non-cacheable area */
```

PE0

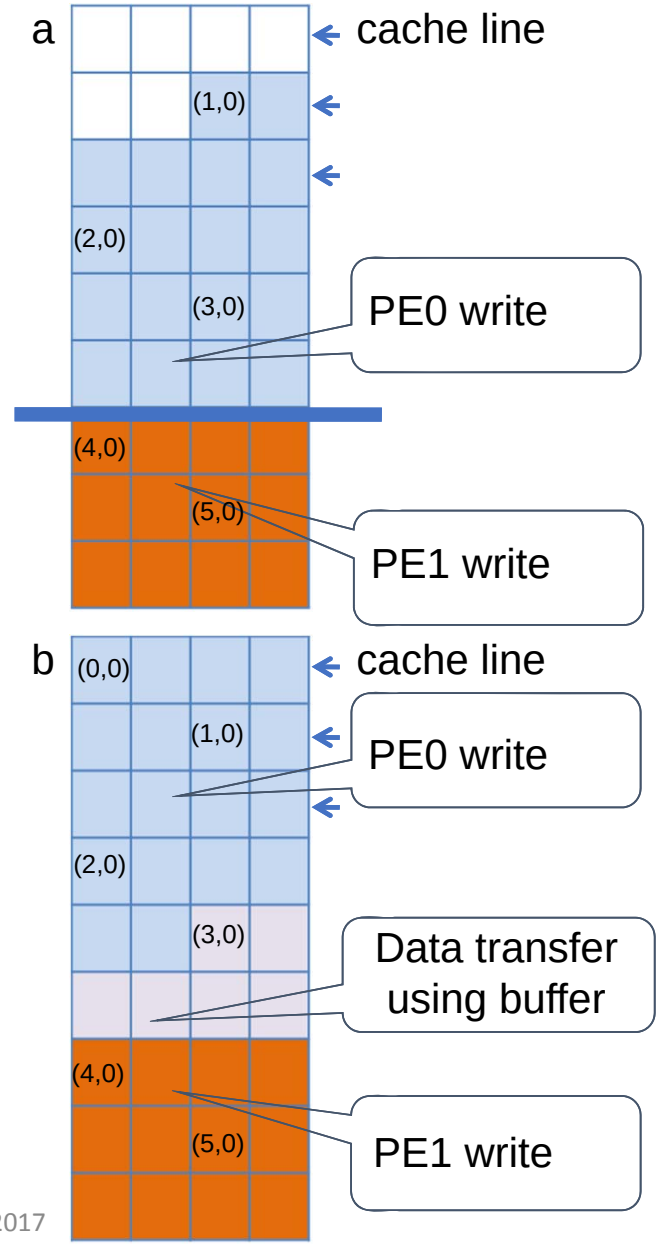
```
for (i = 1; i < 4; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = i * j;
    b[i-1][j] = i * j;
  }
}
```

```
for (i = 4; i < 5; i++) {
  for (j = 0; j < 6; j++) {
    b[i-1][j] = nc_buf[i-4][j];
  }
}
```

PE1

```
for (i = 4; i < 5; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = i * j;
    nc_buf[i-4][j] = i * j;
  }
}
```

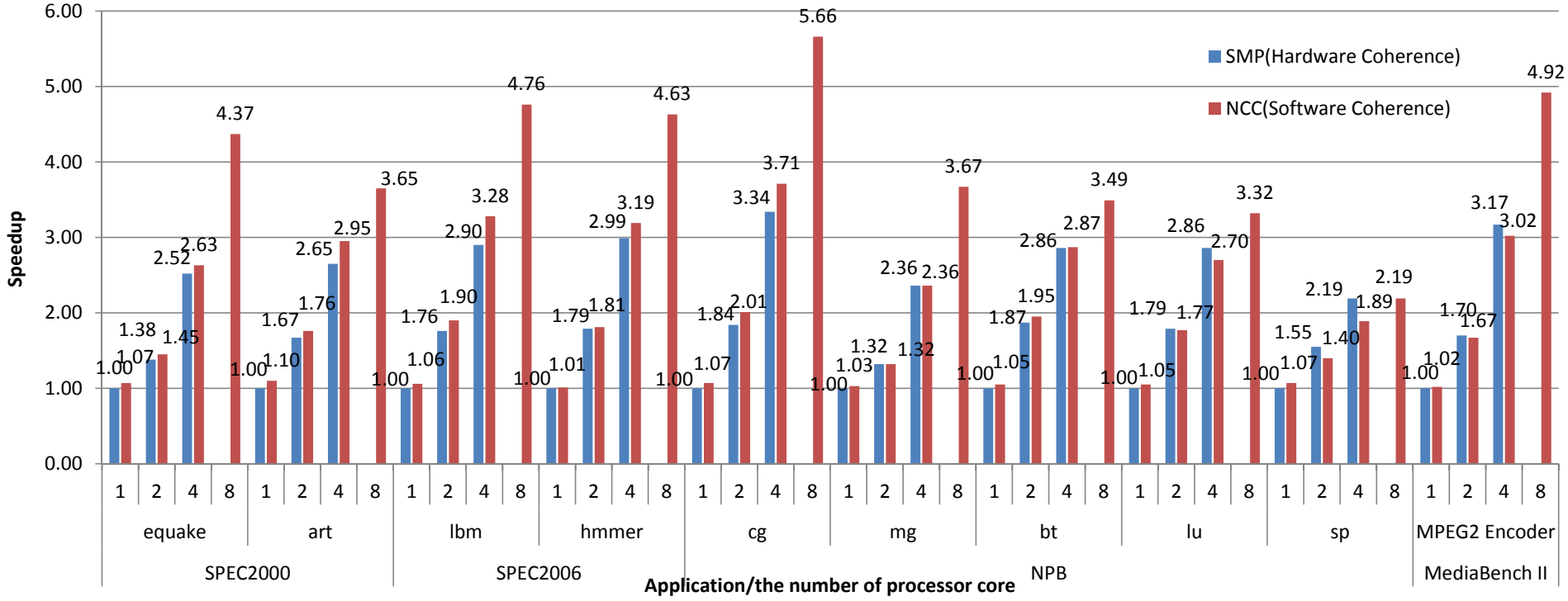
```
for (i = 5; i < 6; i++) {
  for (j = 0; j < 6; j++) {
    a[i][j] = I * j;
    b[i-1][j] = i * j;
  }
}
```



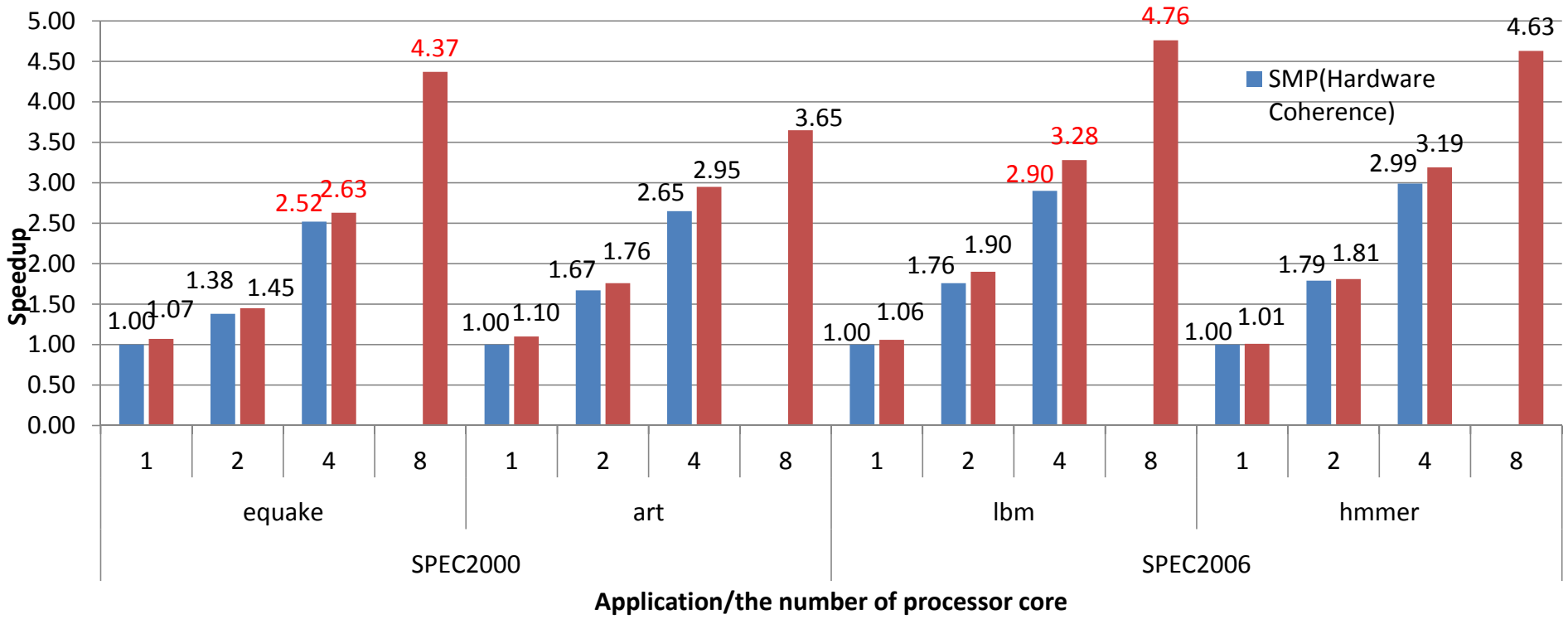
Performance Evaluation

- 10 Benchmark Application in C
  - In Parallelizable C format (Similar to Misra C in embedded field)
  - Fed into source to source automatic parallelization OSCAR Compiler with non-cache coherence control
  - Parallelized code then fed into Renesas C Compiler
- Evaluated in RP2 Processor
  - 8 core
  - Dual 4 core modules
  - Hardware cache coherency for up to 4 cores in the same module

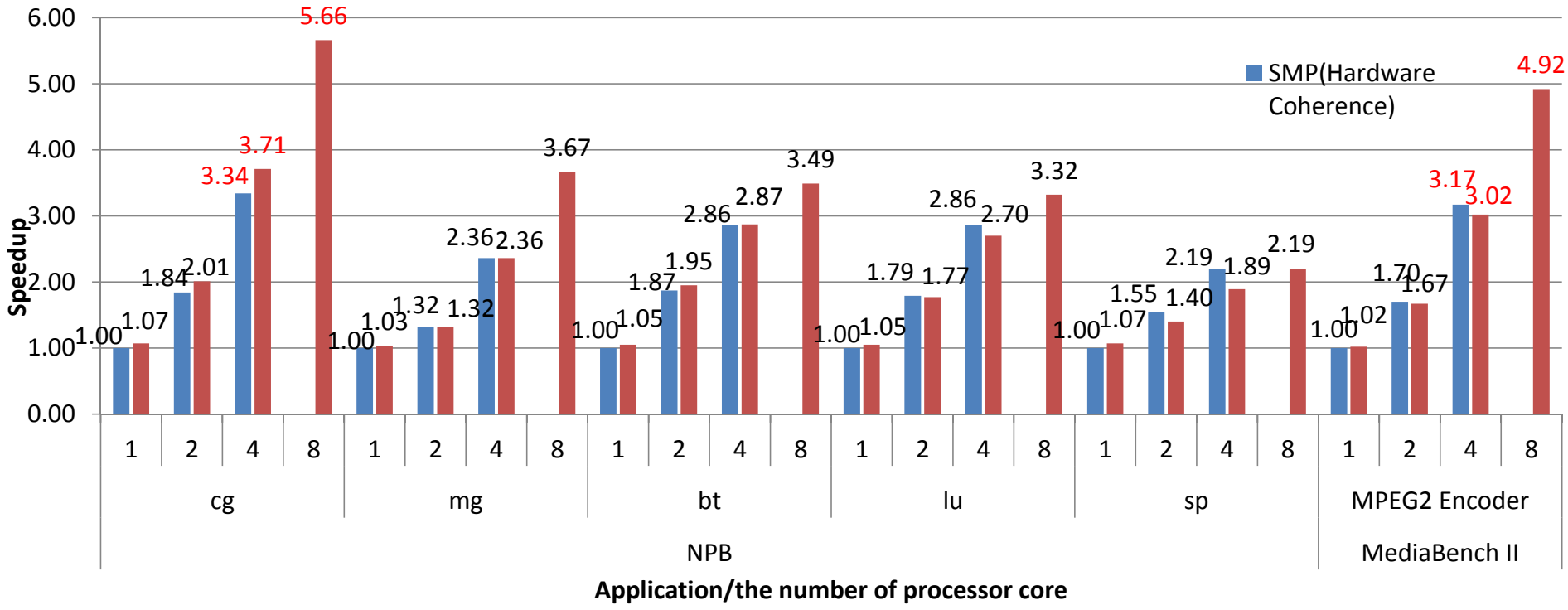
Performance of Hardware Coherence Control & Software Coherence Control on 8-core RP2



SPEC Result


5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
Speedup
SMP(Hardware Coherence)
1.00
1.07
1.38
1.45
2.52
2.63
4.37
1.00
1.10
1.67
1.76
2.65
2.95
3.65
1.00
1.06
1.76
1.90
2.90
3.28
4.76
1.00
1.01
1.79
1.81
2.99
3.19
4.63
1
2
4
8
equake
art
lbm
hmmer
SPEC2000
SPEC2006
Application/the number of processor core

NAS Parallel Benchmark and MediaBench II



Conclusions

- OSCAR compiler automatically controls coherence using following simple program restructuring methods:
  - To cope with stale data problems:
    - **Data synchronization by compilers**
  - To cope with false sharing problem:
    - **Data Alignment**
    - **Array Padding**
    - **Non-cacheable Buffer**
- Evaluated using **10 benchmark applications** Renesas RP2 8 core multicore processor.
  - SPEC2000
  - SPEC2006
  - NAS Parallel Benchmark
  - MediaBench II

Conclusion (cont'd)

- Provides **good speedup automatically**, few examples with 4 cores**:**
  - 2.63 times SPEC2000 equake (2.52 with hardware)
  - 3.28 times SPEC2009 lbm (2.9 with hardware)
  - 3.71 times on NPB cg (3.34 with hardware)
  - 3.02 times on MediaBench MPEG2 Encoder (3.17 with hardware)
- **Enable usage of 8 cores** automatically on RP2 with good speedup:
  - 4.37 times SPEC2000 equake
  - 4.76 times SPEC2009 lbm
  - 5.66 times on NPB cg
  - 4.92 times on MediaBench MPEG2 Encoder

Thank you