# Lecture 20: False Sharing

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

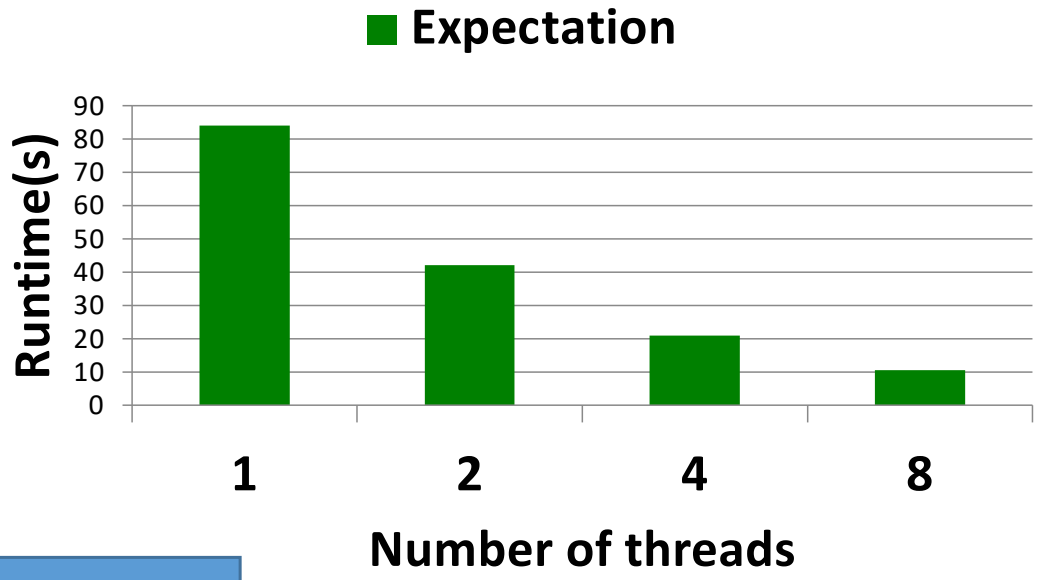
# Today's Class

- False sharing
- Runtime solutions for detecting/repairing false sharing
  - Sheriff
  - Featherlight

Acknowledgement: Today's lectures slides are adapted from several conference presentation slides available online on false sharing

# False Sharing

```
int count[8]; //Global array

thread_func(int id) {
    for(i = 0; i < M; i++)
        count[id]++;
}
```



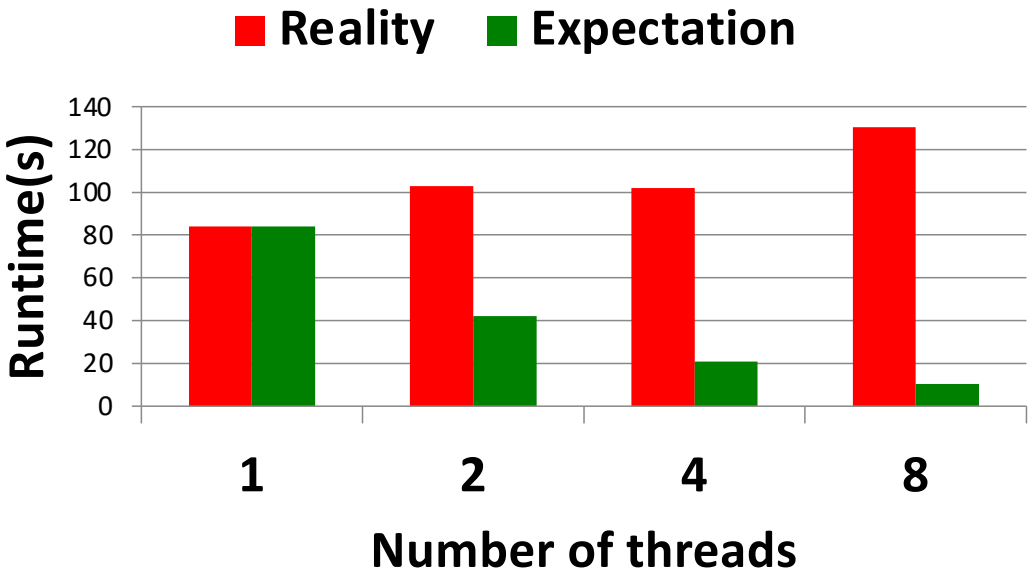

Let's try to understand the problem in this code using the MESI coherence protocol



Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# False Sharing

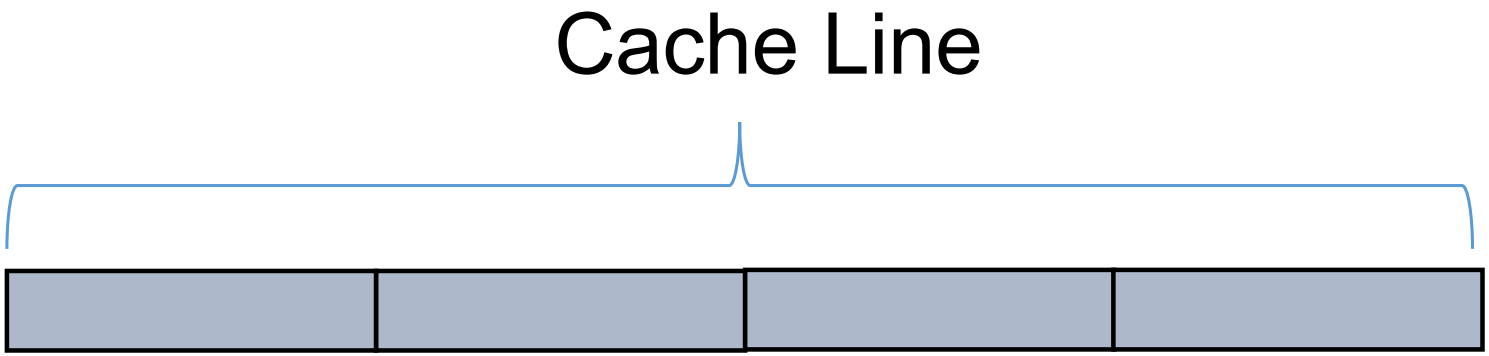
```
int count[8]; //Global array

thread_func(int id) {
    for(i = 0; i < M; i++)
        count[id]++;
}
```






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

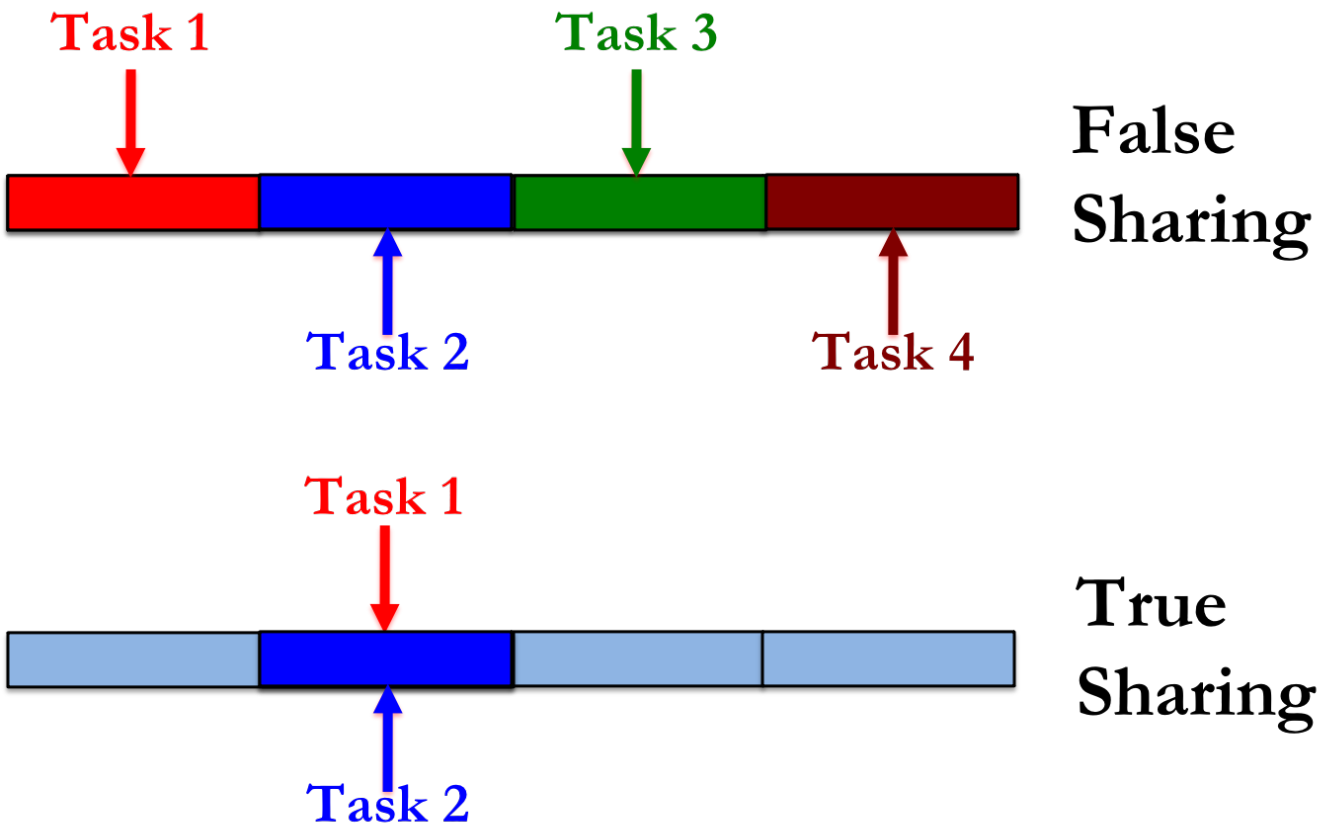
# False Sharing vs. True Sharing






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

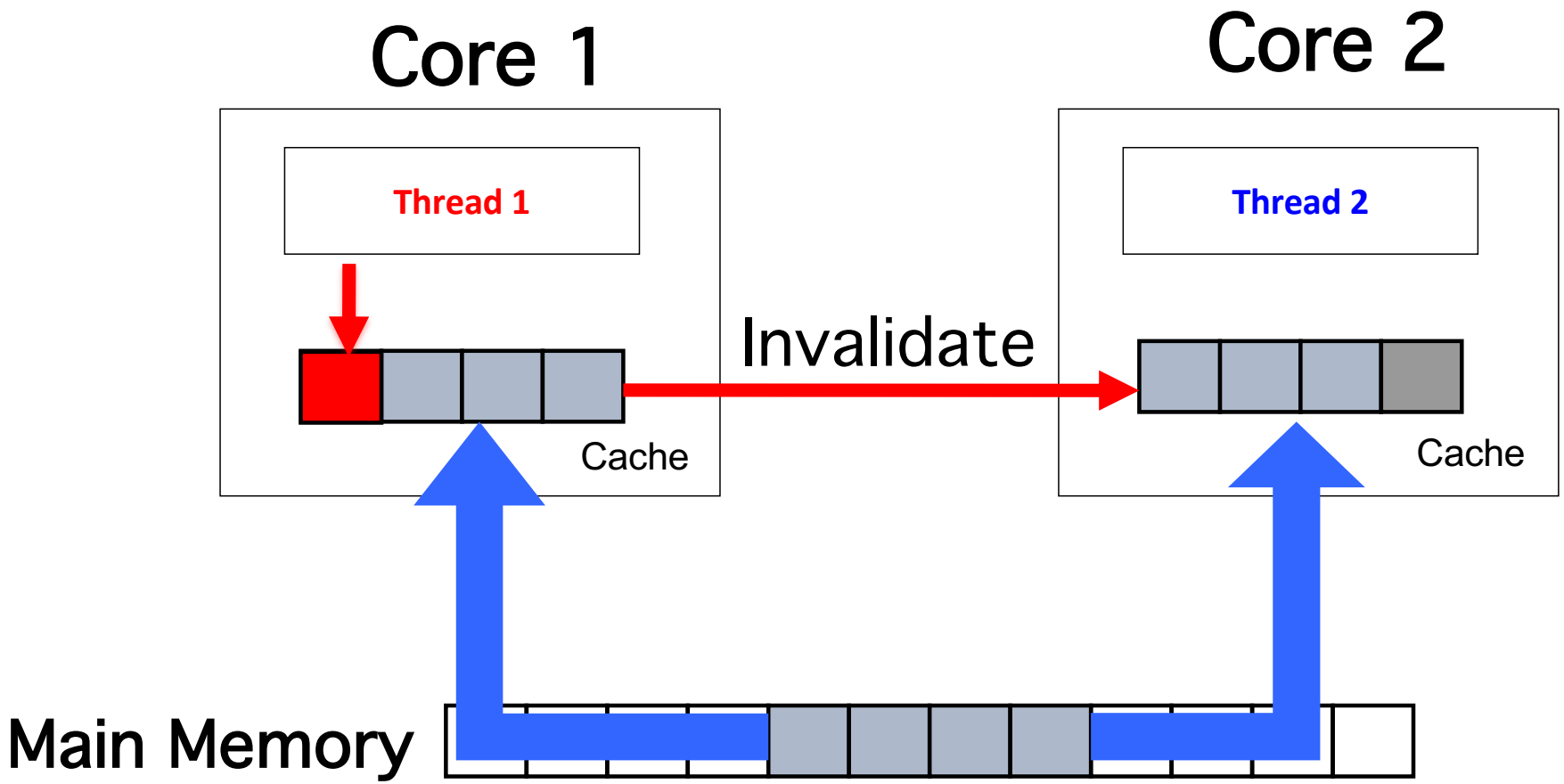
# False Sharing vs. True Sharing






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

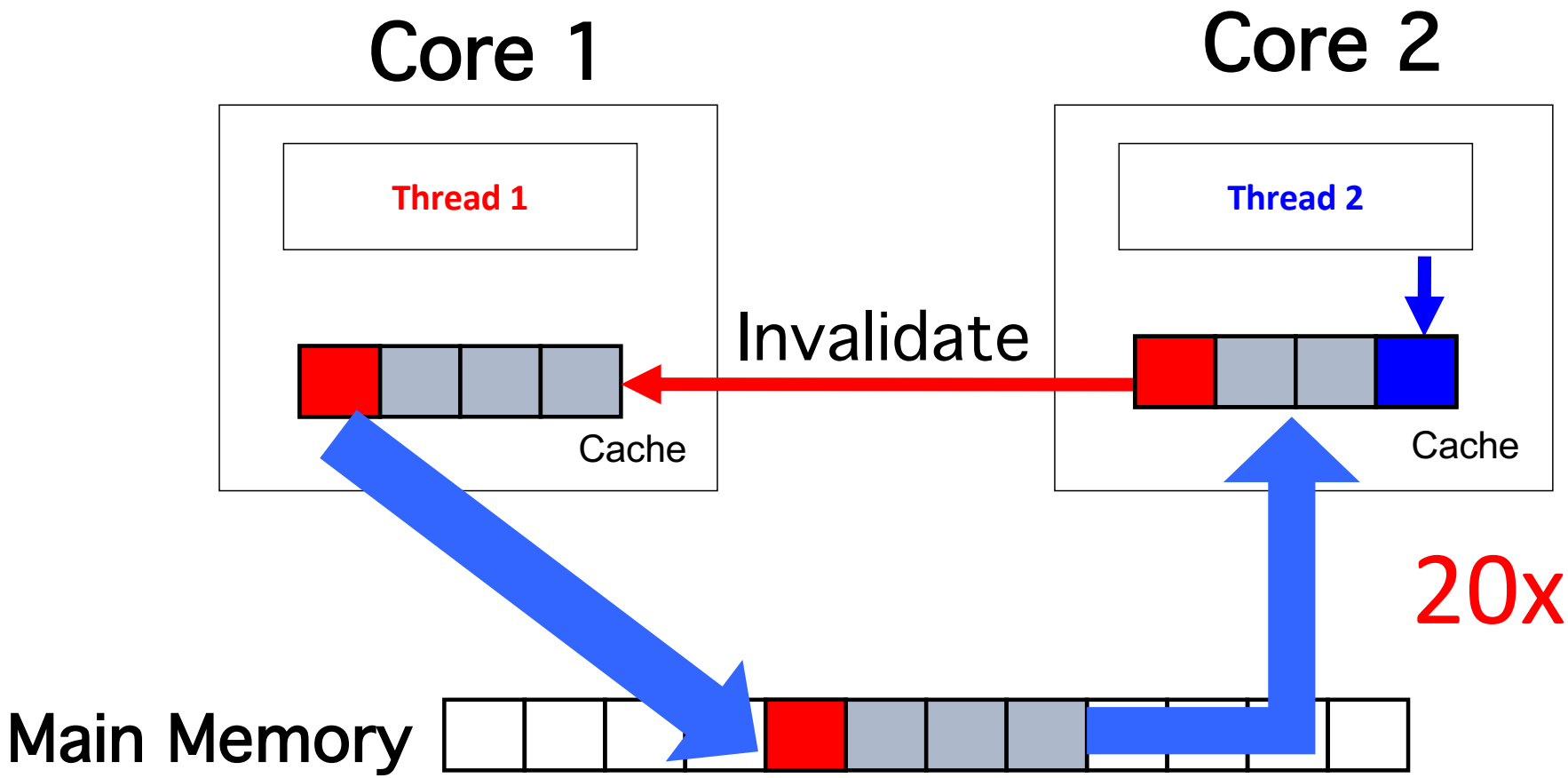
# False Sharing






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# False Sharing






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Resource Contention at Cache Line Level



Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# False Sharing is Everywhere

```
me = 1;
you = 1; // globals
```

```
me = new Foo;
you = new Bar; // heap
```

```
class X {
  int me;
  int you;
}; // fields
```

```
arr[me] = 12;
arr[you] = 13; // array indices
```

Two different threads
T1 & T2 are involved



Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Detecting / Removing False Sharing

- Solutions based on instrumenting memory access and O.S.
  - Sheriff
    - It can both detect, and resolve false sharing during runtime
    - Fakes threads with processes
    - Uses page protection mechanism to track false sharing
  - *and many more…*
- Solutions based on hardware Performance Monitoring Units (PMUs)
  - Featherlight
    - Uses lightweight profiling of hardware Performance Monitoring Units (PMUs) and debug registers
    - Addresses several shortcomings of prior implementations
      - Doesn't require instrumenting memory accesses of O.S.
      - Extremely low overheads
  - *and many more…*

# Walkthrough of Sheriff Execution

**1. Initialization** – creating mapping of global and heap variables for processes

**2. Process creation** – creating processes instead of threads

**3. Execution** – copies of memory pages per process for local updates

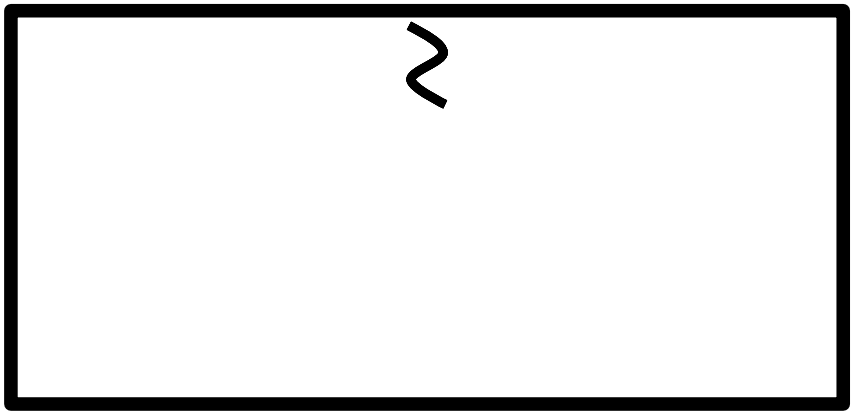
**4. Synchronization** – merging diffs in per-process pages into the global copy

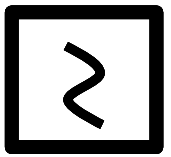
```
1.  /* global variables */
2.  int sum[2];
3.  int main() {
4.    /* heap allocations */
5.    int a = new int[size]; //initialized
6.    T1 = new thread([=]() {
7.      for(int i=0; i<size/2; i++) sum[0]+=a[i];
8.    });
9.    T2 = new thread([=]() {
10.     for(int i=size/2; i<size; i++) sum[1]+=a[i];
11.   });
12.   T1.join(); T2.join();
13.   //Cleanups
14. }
```

# Sheriff Execution: Process Creation

*shared address space*

*disjoint address spaces*





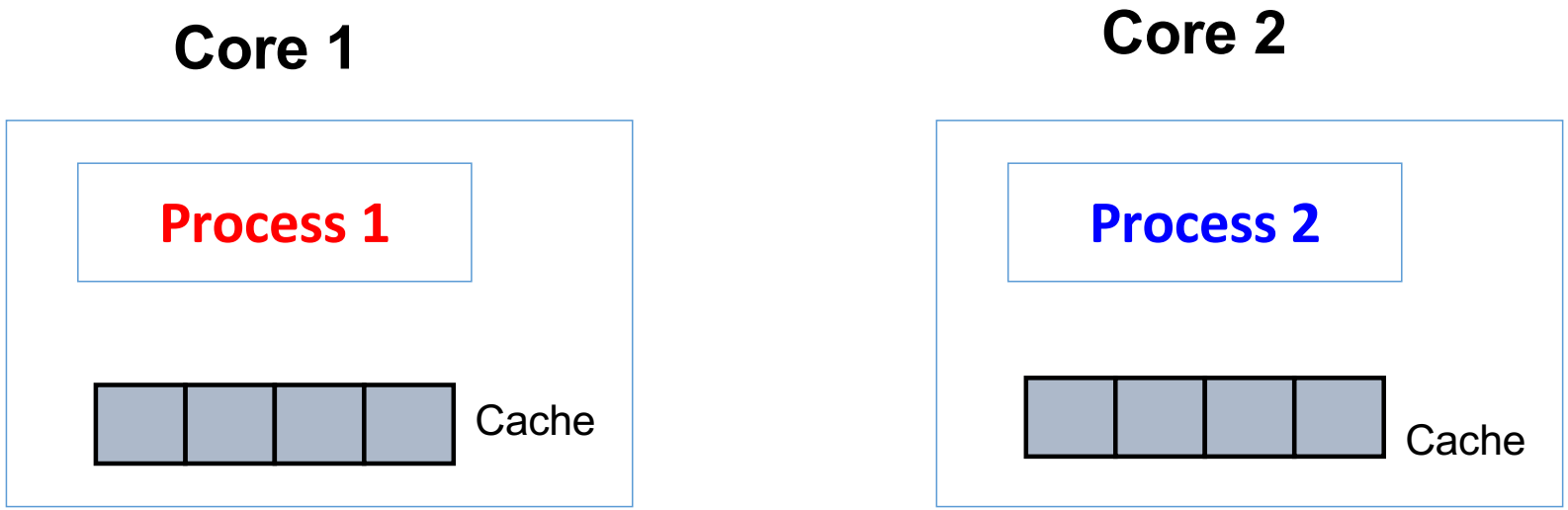


Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Sheriff Execution: Process Creation

- In Linux, both pthreads and processes are essentially a KLT, and are created using the same API (do_fork)
- Threads are created on the same CPU to improve locality, whereas processes are created on different CPUs

# Sheriff Execution: Initialization





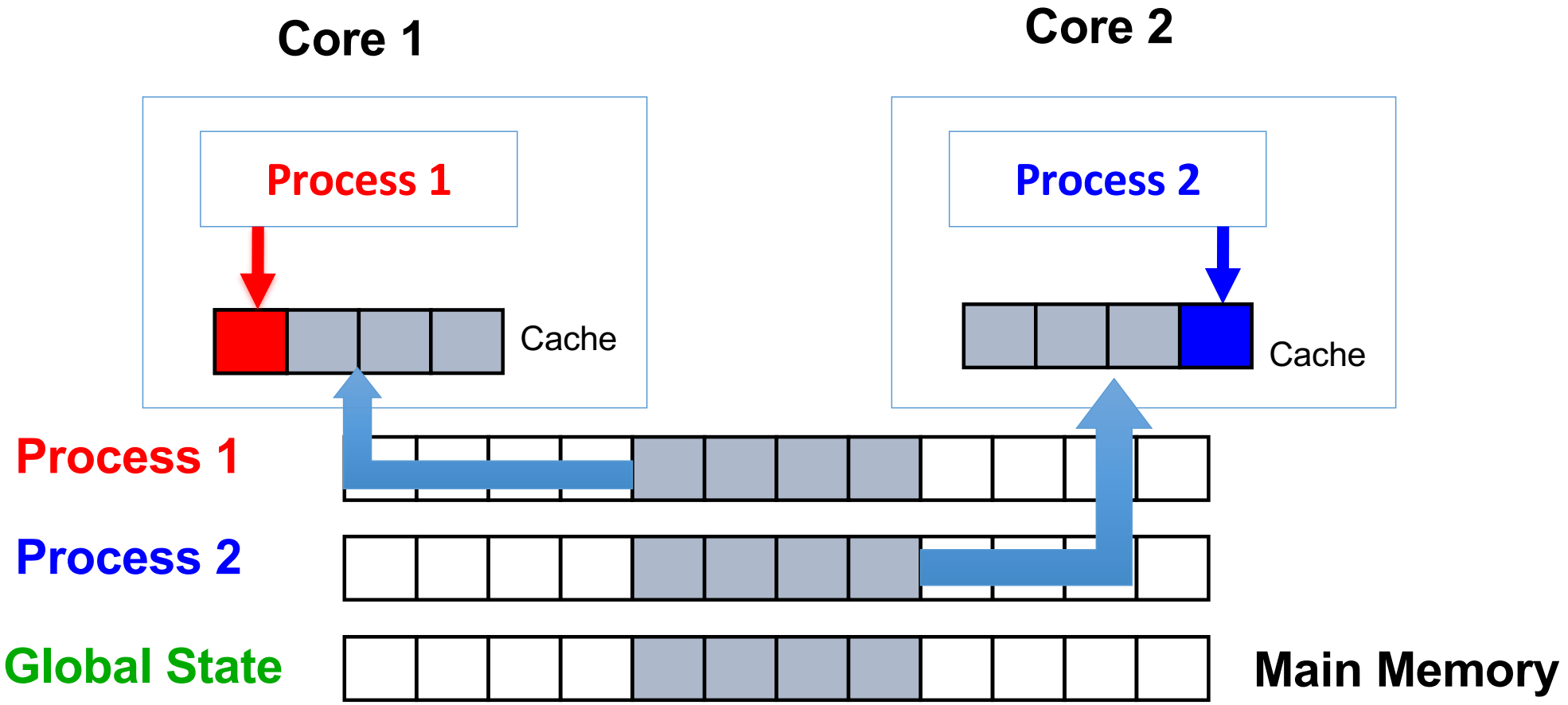


Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Sheriff Execution: Initialization

- Advantages of converting threads into processes
  - Enables the use of per-thread page protection, allowing Sheriff to track memory accesses by different threads (processes)
  - Each thread's (process) memory access are isolated, hence they would not update the same cache line
    - No false sharing!
- Memory mapped files are used to share global and heaps across different processes
- Twin copies of the pages for storing the global and heaps
  - Shared mapping for holding shared states
    - Pages storing these shared states are marked copy-on-write
  - Private mapping for per-process updates
    - Private copy of of the above shared pages are created whenever a process would attempt to update a page for the first time

# Sheriff Execution: Execution






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Sheriff Execution: Synchronization

- There are two different types of synchronization points
  - Thread termination
  - End of the critical section (mutex unlock), barriers, etc.
- At each synchronization point, Sheriff commits changes from private pages to the shared pages
  - It commits only the differences between the twin and the modified pages

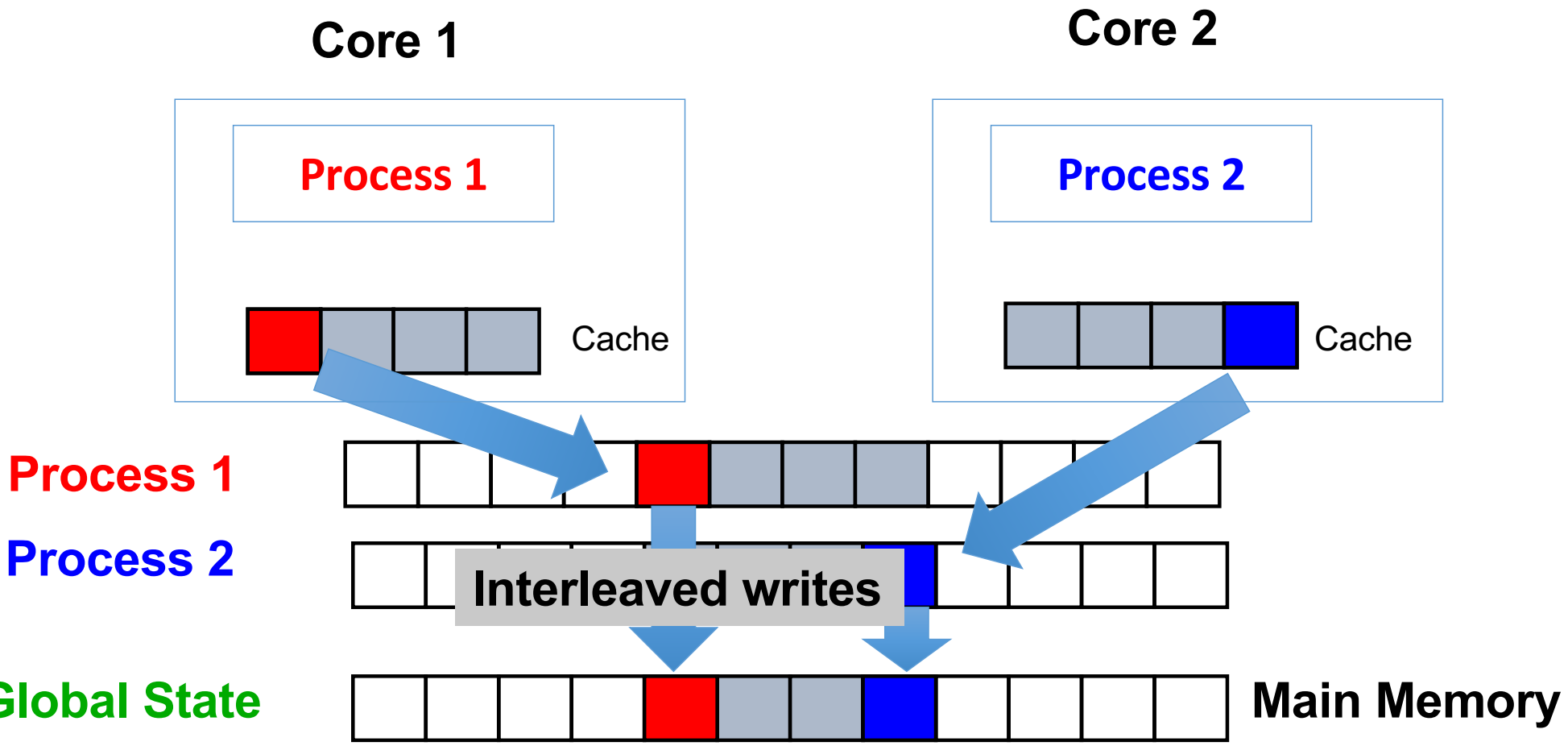
# Sheriff Execution: Synchronization


Snapshot and diffing the local changes




Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Sheriff Execution: Synchronization






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Sheriff Execution: Synchronization

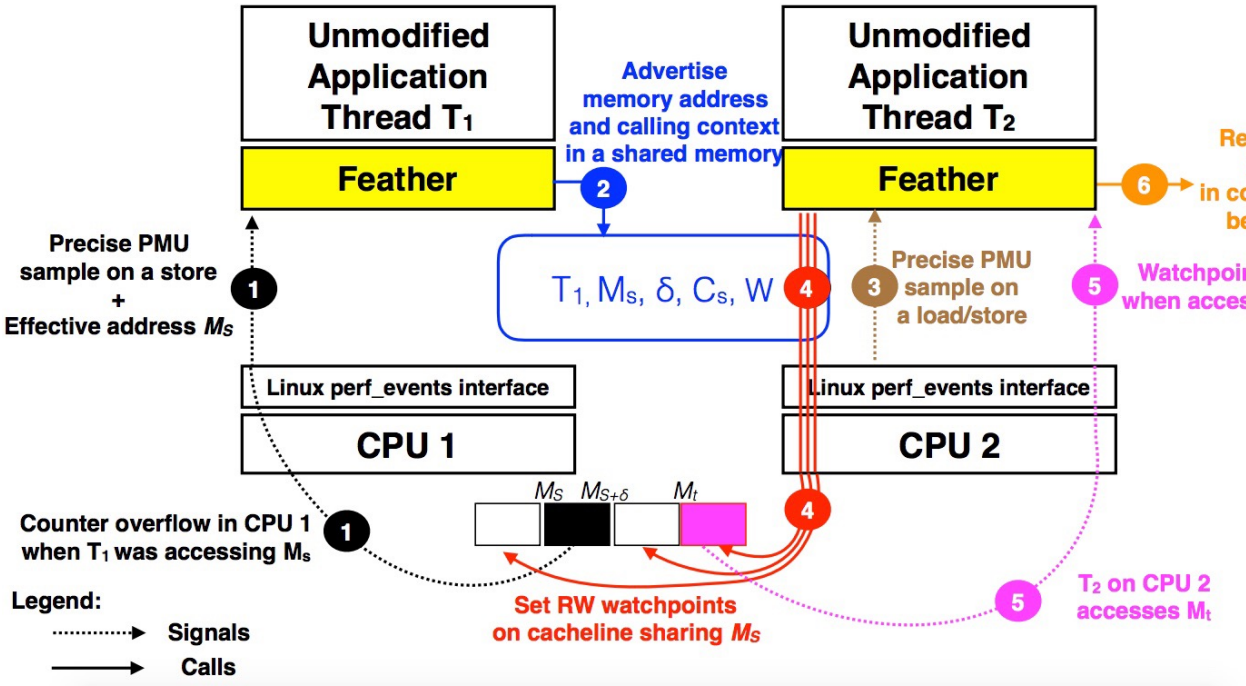






Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

# Featherlight: High Level Overview




- PMUs sample memory address **M** accessed by each thread (or process)
- A thread publishes its sampled address at a common location visible to other threads
- Other threads use hardware debug registers to monitor the addresses sharing the same cache line as **M**, excluding **M** itself
- If a thread accesses another variable in the same cache line, the debug register traps, which indicates false sharing

Debug registers enable trapping CPU execution for debugging when the PC (program counter) reaches an address (breakpoint) or an instruction accesses a designated address (watchpoint)

# Reading Materials

- Sheriff
  - https://people.umass.edu/tongping/pubs/sheriff-oopsla11.pdf
- Featherlight
  - https://dl.acm.org/doi/10.1145/3178487.3178499

# Next Lecture (L #21)

- Data race detection in task parallel programs