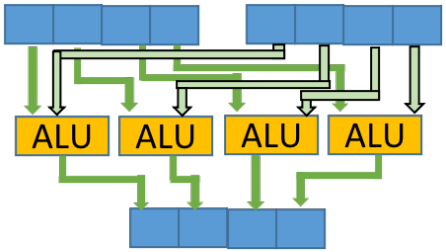
Lecture 23: End Semester Review

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

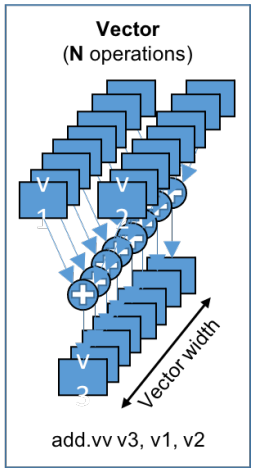
Today's Class

- End semester review
- Quiz-4

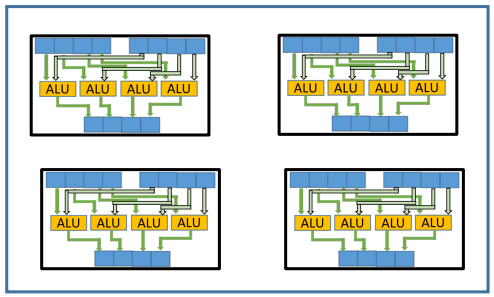
SIMD Vector Units (1/2)



SIMD operation on vectors



- Special registers that support instructions to operate upon **vectors** than **scalar** values
- Each core can operate on more than one 32-bit value in each cycle



Multicore processor supporting SIMD operations

SIMD Vector Units (2/2)

```
40 void mul_vector() {
41   //initialize
42   for(int i=0; i<size; i++) {
43     Vec8f _A(1.0f);
44     Vec8f _B(1.0f);
45     Vec8f _C(0);
46     for(int j=0; j<size; j+=VECTOR_WIDTH) {
47       _A.store(&A[i*size+j]);
48       _B.store(&B[i*size+j]);
49       _C.store(&C[i*size+j]);
50     }
51   }
52
53   for(int i=0; i<size; i++) {
54     for(int k=0; k<size; k++) {
55       Vec8f _A = A[i*size + k];
56       for(int j=0; j<size; j+=8) {
57         // C[i*size+j] = A[i*size+k] * B[k*size+j] + C[i*size+j];
58         Vec8f _B = Vec8f().load(&B[k*size + j]);
59         Vec8f _C = Vec8f().load(&C[i*size + j]);
60         _C = _A*_B + _C;
61         _C.store(&C[i*size + j]);
62       }
63     }
64   }
65 }
```

Matrix Multiplication using VCL

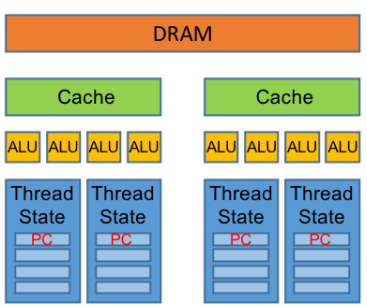
```
53   for(int i=0; i<size; i++) {
54     for(int k=0; k<size; k++) {
55       VEC_Xf _A = A[i*size + k];
56       VEC_Xf _B, _C;
57       for(int j=0; j<size; j+=VECTOR_WIDTH) {
58         // C[i*size+j] = A[i*size+k] * B[k*siz
59         _B.load(&B[k*size + j]);
60         _C.load(&C[i*size + j]);
61         _C = _A*_B + _C;
62         _C.store(&C[i*size + j]);
63       }
64     }
65   }
66 }
```

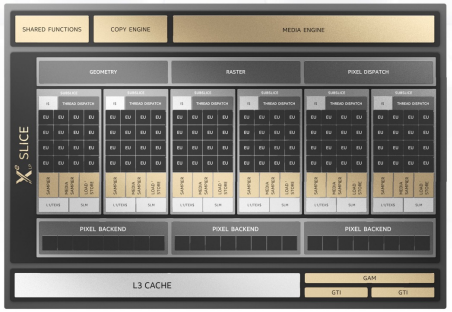
Alternative approach for loading

GPU Computing using boost::compute

```
32  int* a = my_svm::alloc<int>(context, size);
33  int* b = my_svm::alloc<int>(context, size);
34  int* c = my_svm::alloc<int>(context, size);
35  std::fill(a, a+size, 1);
36  std::fill(b, b+size, 2);
37  std::fill(c, c+size, 0);
38  for(int i=0; i<size; i++) assert(a[i] == 1);
39  for(int i=0; i<size; i++) assert(b[i] == 2);
40
41  // source code for the add kernel
42  const char source[] = BOOST_COMPUTE_STRINGIZE_SOURCE(
43      __kernel void add(__global const int *a,
44                        __global const int *b,
45                        __global int *c)
46      {
47          const uint i = get_global_id(0);
48          c[i] = a[i] + b[i];
49      }
50  );
51  // create the program with the source
52  compute::program program = compute::program::build_with_source(source, context);
53  // create the kernel
54  compute::kernel kernel(program, "add");
55  // set the kernel arguments
56  kernel.set_arg_svm_ptr(0, a);
57  kernel.set_arg_svm_ptr(1, b);
58  kernel.set_arg_svm_ptr(2, c);
59  // run the add kernel
60  timer::kernel("GPU kernel", [&]() {
61    queue.enqueue_1d_range_kernel(kernel, 0, size, vector_width);
62    queue.finish();
63  });
64  time_gpu+=timer::duration();
```

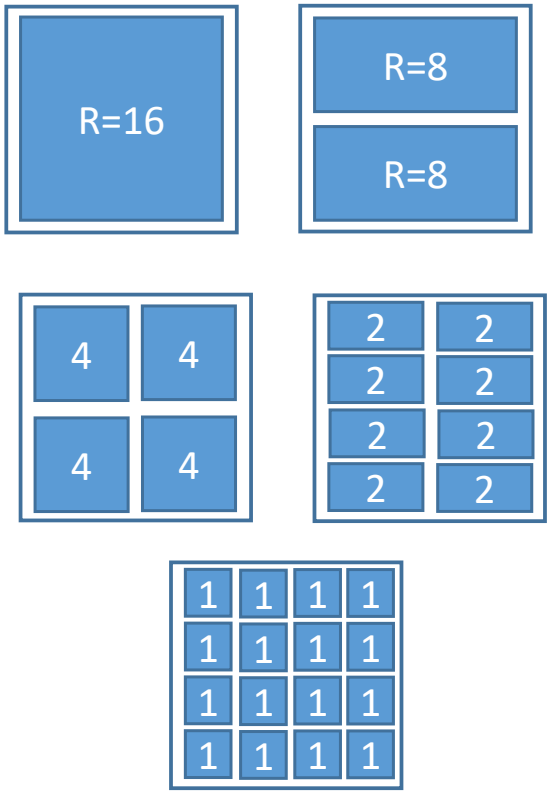
w





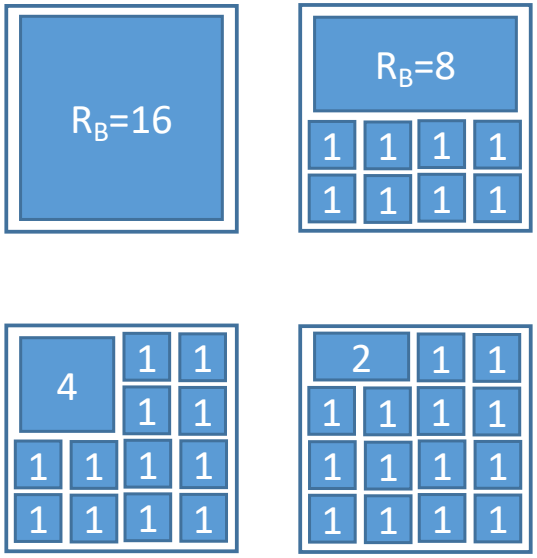
- Multicore processors are latency oriented, whereas GPUs are throughput oriented
- Steps for using Intel SVM based boost::compute programming
  - Create a context and device queue
  - Create programs to execute on the GPU
  - Allocate SVM memory on the host as it is also accessible by the GPU
  - Select the GPU kernel and set its arguments
  - Provide kernel to the command queue for execution
  - Free the SVM memory

Heterogeneous Computing (1/2)



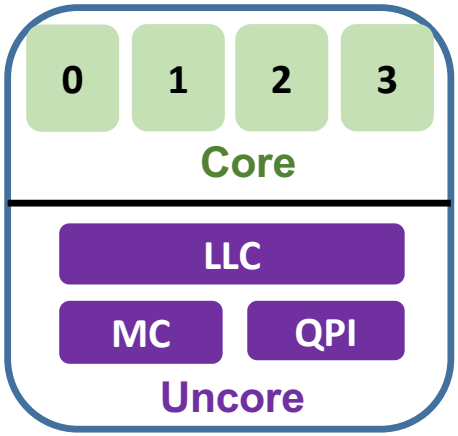
- Amdahl's law for **symmetric** multicores
  - Serial fraction of the program would run on a single core with performance as:
    - (1-f) / $Perf_R$
      - $Perf_R$ is the performance of each of the single core of the processor type R shown below
  - Parallel fraction use N/R cores at the rate Perf(R) each
    - f / (Perf(R) * (N/R)) = f*R / Perf(R)*N
  - Speedup(f, R, N) = 1 / ({(1-f)/Perf(R)} + {f*R/(Perf(R)*N)})

Heterogeneous Computing (2/2)



- Amdahl's law for **asymmetric** multicores
  - Each processor is using the same number of total resources (**N=16**)
    - One **B**ig core with R_B resources would leave N-R_B resources for little cores
    - Assuming each little core has R=1, total number of little cores are N-R_B
  - Serial fraction would still be represented as in symmetric
    - (1-f) / Perf(R_B)
  - Parallel fraction using one big core with Perf(R_B) performance and N-R_B little cores with Perf(1)=1 performance would now be
    - f / (Perf(R_B) + N – R_B)
  - Speedup(f, R_B, N) = 1 / ({(1-f) / Perf(R_B)} + {f/(Perf(R_B) + N –R_B)})

Power Management



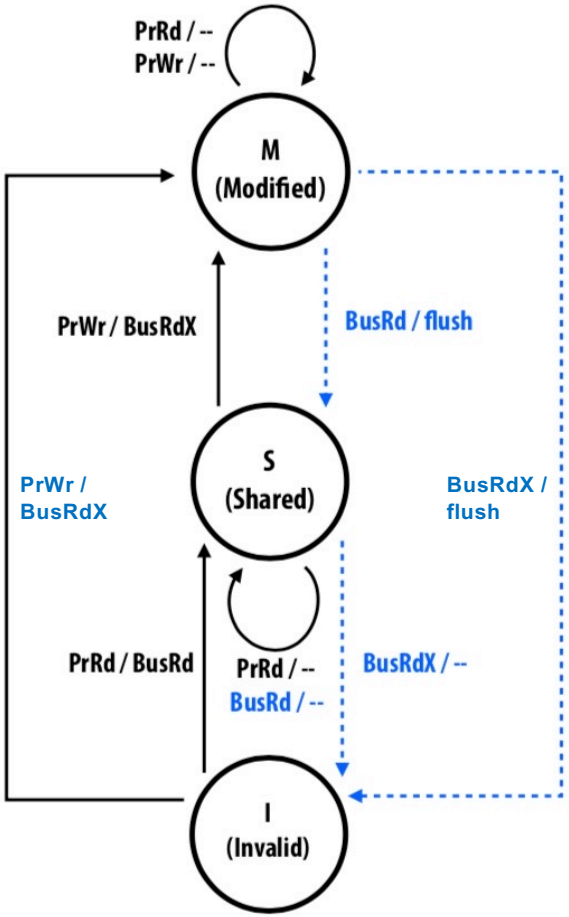
Multicore Processor

- P-states
  - Dynamic Voltage and Frequency Scaling (DVFS) is used by the processor to operate the core at a specific frequency and voltage
  - Each P-states have an associated frequency
- C-states
  - Power states used by the CPUs to reduce the power at **C**ore level or on a CPU **P**ackage **C**ore level (core, private caches, etc.)
  - Core goes to sleep
    - Used when some of the cores are not being used at all
- Uncore frequency scaling
  - Changes the frequency of the uncore elements in the processor
  - Can be set in the userspace similar to DVFS
  - Currently supported only by Intel processor

Cache Coherence (1/3)

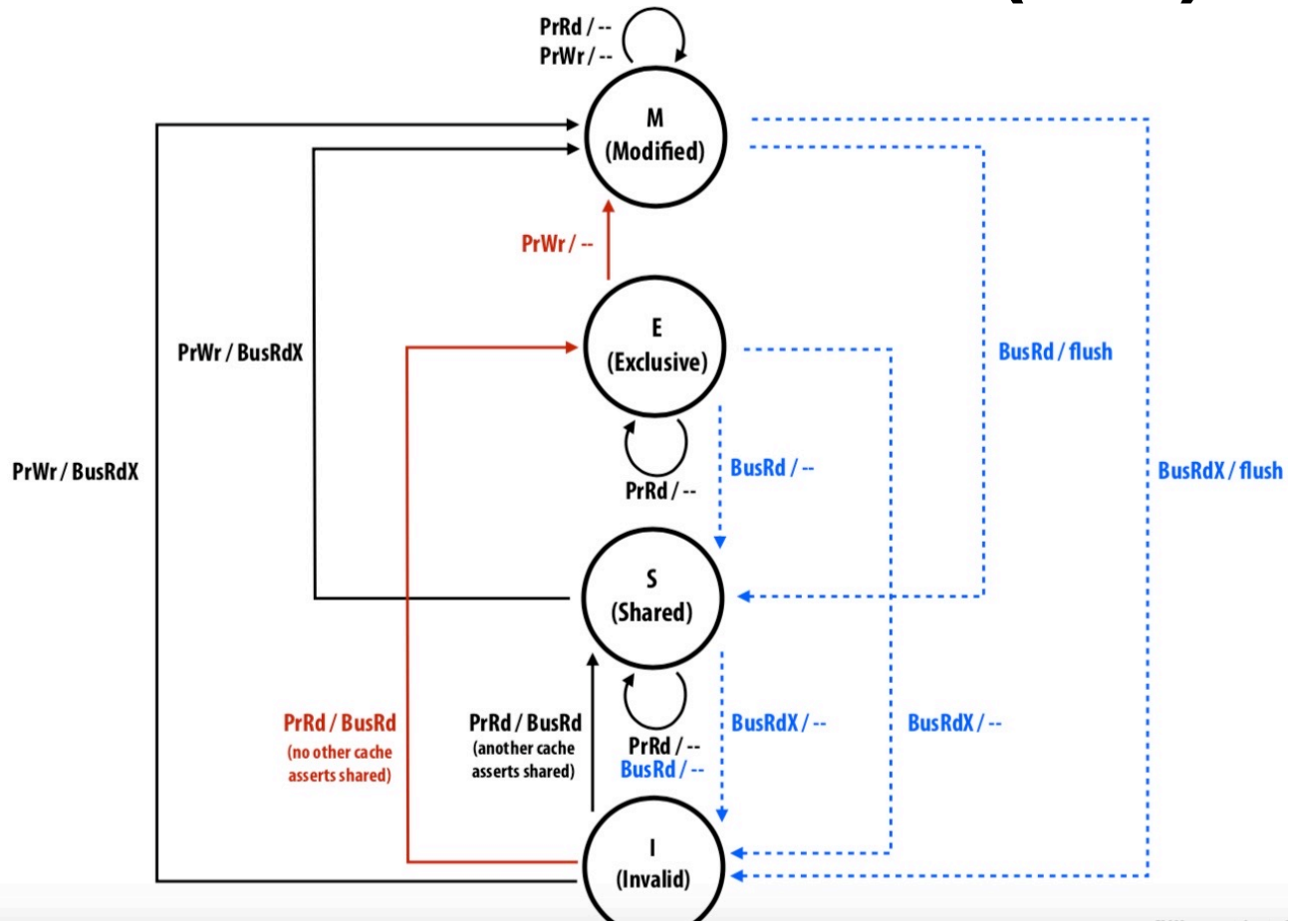
- **Program order** must be maintained at a single processor
  - A read by processor P to address X that follows a write by P to address X, should return the value of the write by P
    - Assuming no other processor wrote to X in between
- **Write propagation** to other processors
  - A read by processor P1 to address X that follows a write by processor P2 to X returns the written value... if the read and write are "sufficiently separated" in time (store buffers!)
    - Assuming no other writes to X occurs in between
- **Write serialization**
  - Writes to the same address are serialized: two writes to address X by any two processors are observed in the same order by all processors
    - E.g., if values 1 and then 2 are written to address X, no processor observes X having value 2 before value 1

Cache Coherence (2/3)



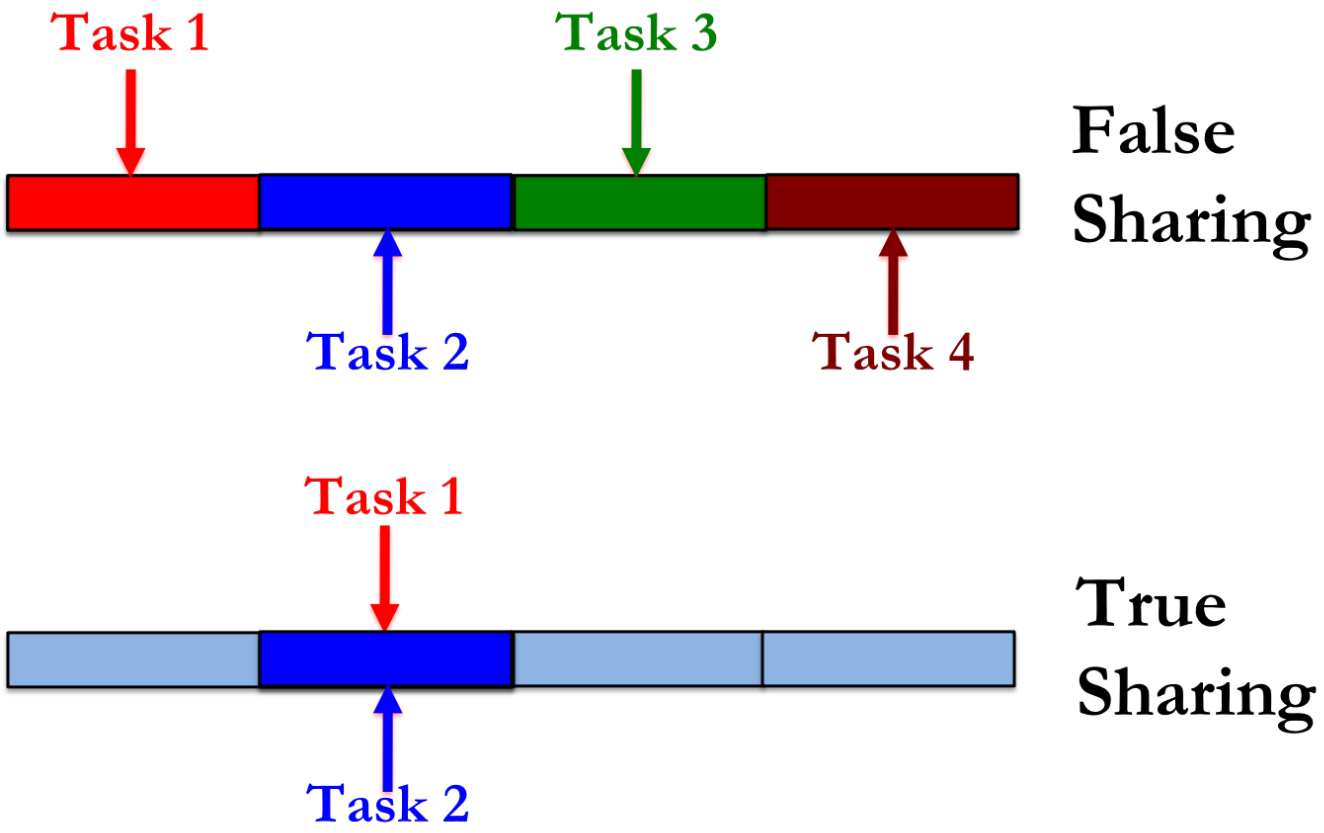
- **MSI** protocol
  - A line in the **M** state can be modified without notifying other caches
  - Processor can only write to lines only in the **M** state
    - If line is not already exclusive in cache, cache controller must first broadcast a **read-exclusive** transaction to move the line into that state
      - Required even if the line is in **S**hared state
  - When other processor's cache controller snoops a read exclusive for a line it contains
    - Must invalidate the line in its cache
    - Because if it didn't, then multiple caches will have the same line

Cache Coherence (3/3)



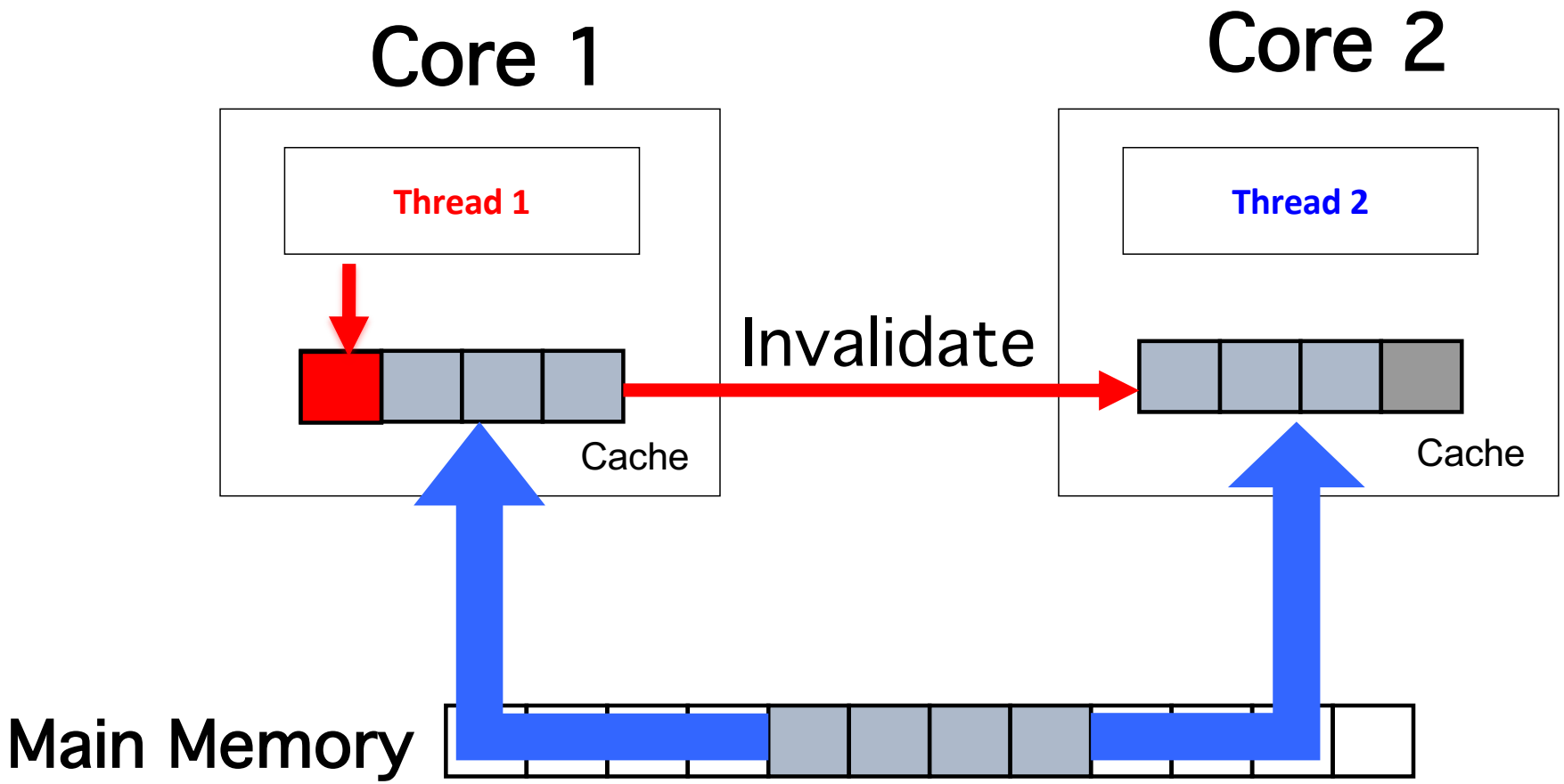
- **MESI** protocol
  - Removes the redundant BusRdX when only one cache holds a line in S state and wants to move it into M state
  - Cache loads a line directly into **E**xclusive state instead of **S**hared state

False Sharing (1/3)



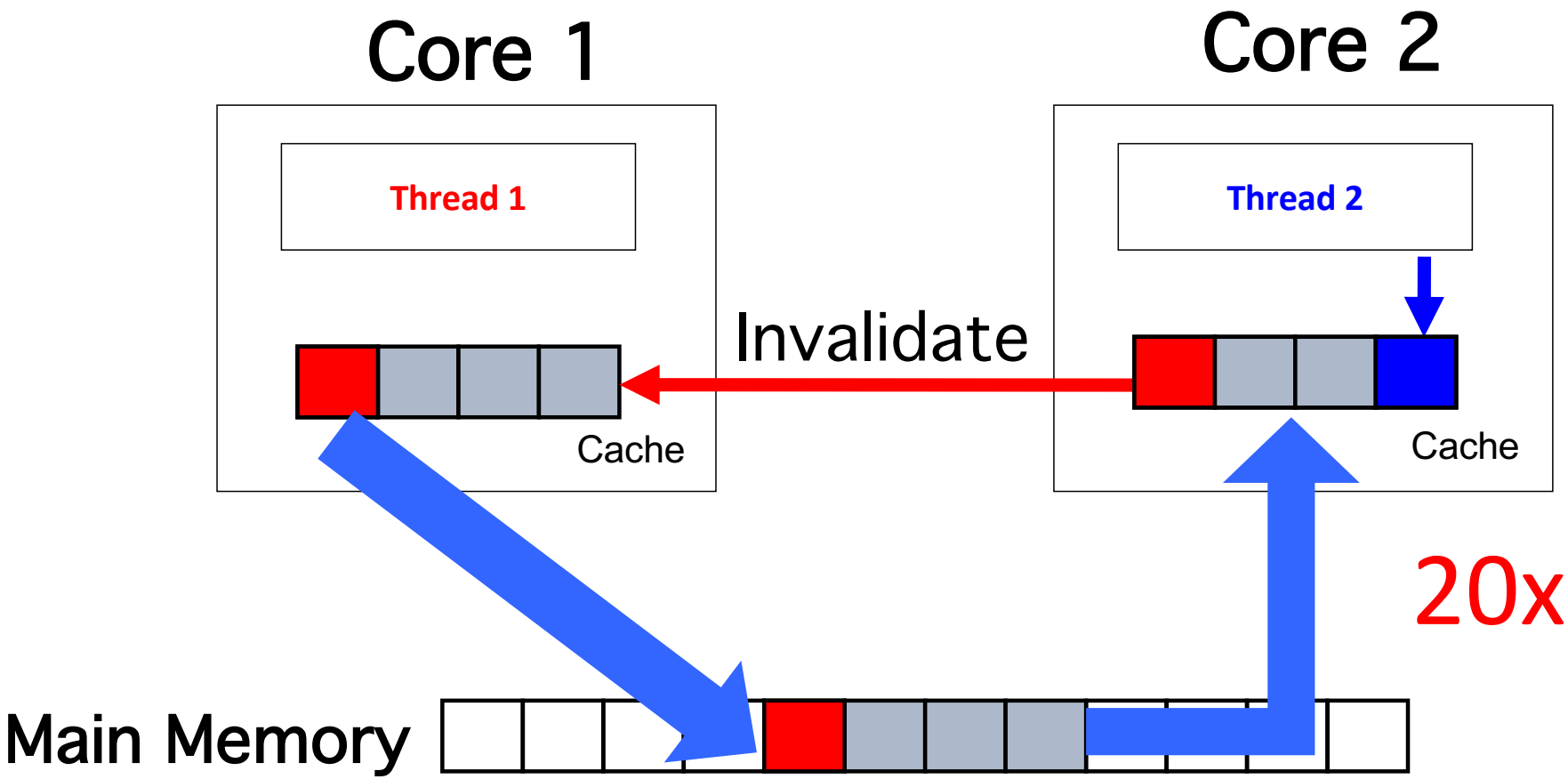
Sources: https://people.umass.edu/tongping/pubs/dthreads-final.pptx, and https://people.umass.edu/tongping/pubs/sheriff-final.pptx

False Sharing (2/3)



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

False Sharing (3/3)



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

Supporting Resilience

- Checkpointing
  - Rollback recovery approach using checkpoint/restart
  - The programmer adds some specific functions in the application to save essential state and restore from this state in case of failure
  - Hurts productivity and leads to I/O bottleneck
- Forward recovery
  - Application can handle the error and execute some some specific recovery procedure without relying on classic rollback recovery
- Replication
  - Each process/thread/task is replicated such that the probability that all replicas would fail is acceptably small
  - Amount of computational resources is a major challenge
- Failure prediction
  - Draw conclusions about upcoming failures from the occurrence of previous failures

Next Two Lectures

- Remaining two lectures will be student seminar
- Each project group will give a presentation as notified earlier