Concepts and Models of Parallel and Data-centric Programming

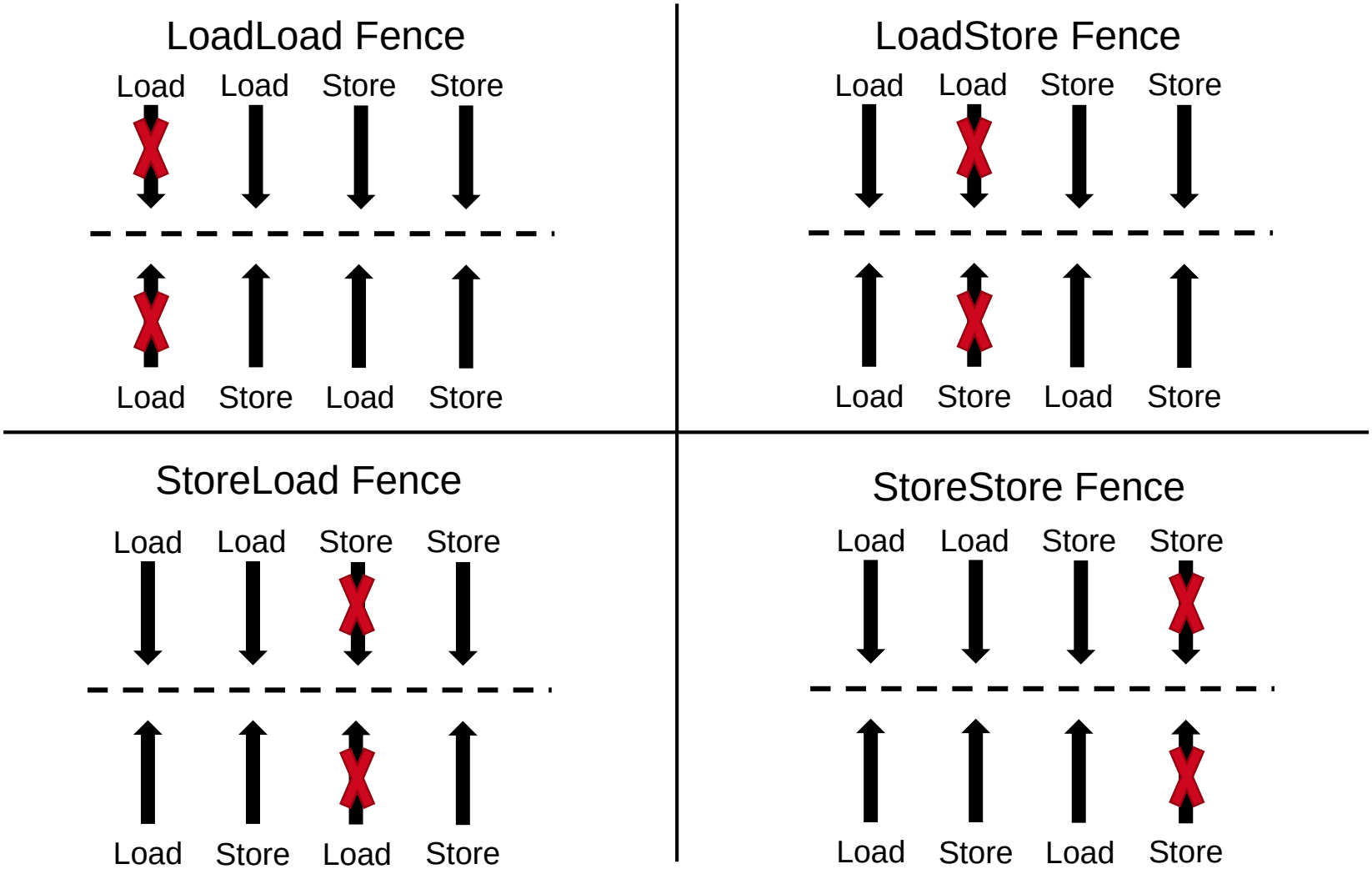
Shared Memory – C++ Memory Order Additions

Lecture, Summer 2020

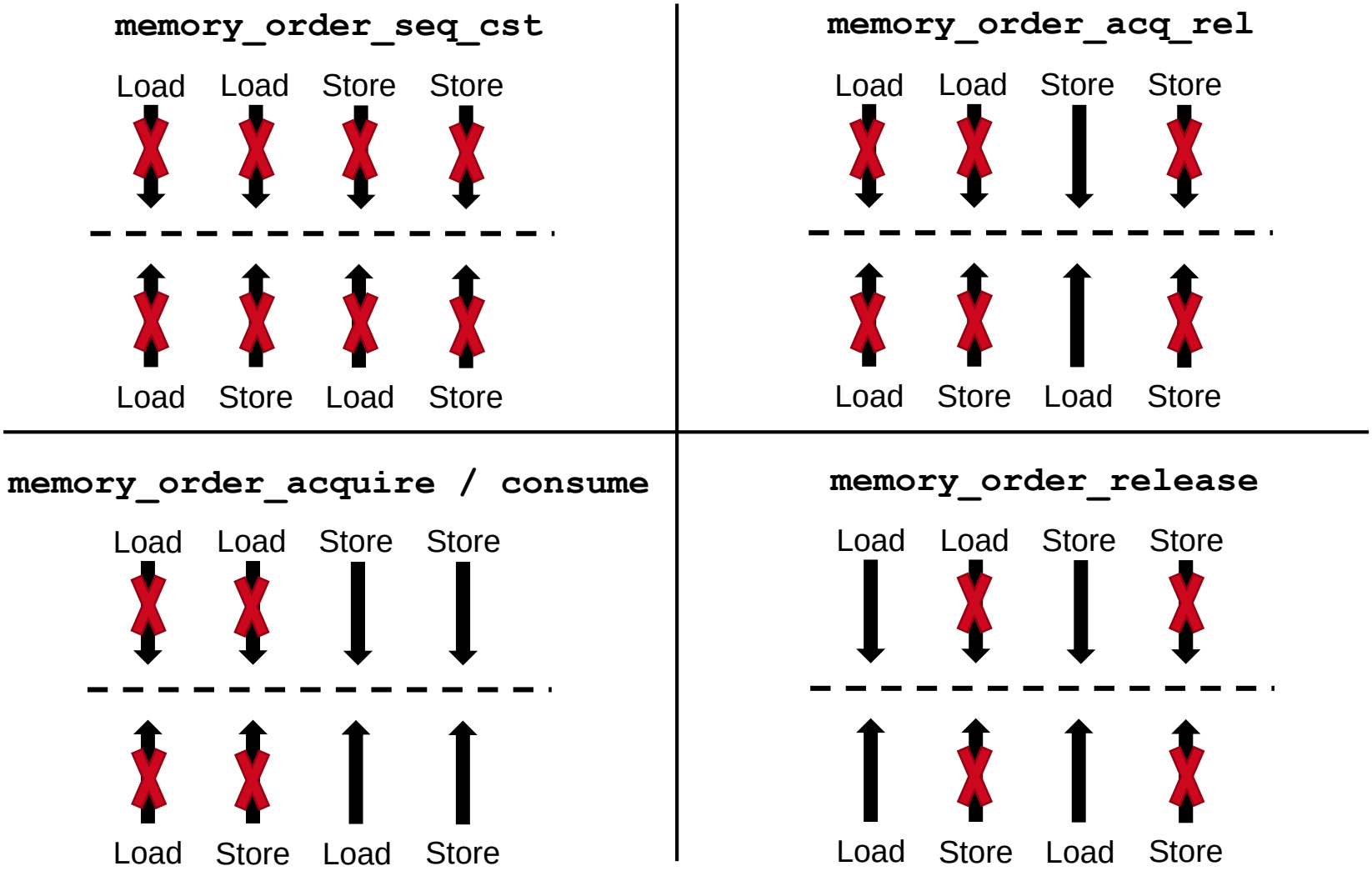
Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

i12 High Performance Computing | RWTH AACHEN UNIVERSITY

Recap: Memory Fences



Reacap: Memory Orderings C++



Recap: Atomic Operations

- Three categories of atomic operations supported with different memory ordering semantics:
  - **Load:** Atomically read from variable (`res = myvar.load()`)
  - **Store:** Atomically write to variable (`myvar.store(val)`)
  - **Read-Modify-Write:** Atomically read *and* write to variable (`res = myvar.exchange(val)`)
- Default memory order for all operations: `memory_order_seq_cst`
- Supported memory orders for different operations

	relaxed	consume	acquire	release	acq_rel	seq_cst
Load	✓	✓	✓			✓
Store	✓			✓		✓
Read-Modify-Write	✓	✓	✓		✓	✓

C++ Fences vs. Atomic Operations (1)

- There is a difference in memory reordering constraints between using atomic operations (load, store, read-modify-write) and calling a fence operation in C++.
- Atomic operations with a given memory order prohibit reorderings *only* with the atomic operation itself

```
 1  std::atomic<int> myatomic(42);
 2  int X = 0, Y = 0;
 3
 4  r1 = X; // Load of X
 5
 6  // Loads and Stores before this atomic operation
 7  // cannot be reordered with the atomic store
 8  myatomic.store(1, std::memory_order_release);
 9
10  // But: This store can be reordered with the atomic store and the load of X.
11  Y = 41; // Store to Y
```

C++ Fences vs. Atomic Operations (2)

- The fence operation `std::atomic_thread_fence` in C++ has stronger guarantees.
- In case of memory_order_release: No reordering with *any* store after the fence (LoadStore, StoreStore fence)

```
 1  std::atomic<int> myatomic(42);
 2  int X = 0, Y = 0, Z = 0;
 3
 4  r1 = X; // Load of X
 5
 6  // Loads and Stores before this fence
 7  // cannot be reordered with any store after the fence
 8  std::atomic_thread_fence(std::memory_order_release);
 9
10  // This store cannot be reordered with the load of X.
11  Y = 41; // Store to Y
12  // This load can be reordered with the load of X.
13  r2 = Z; // Load of Z
```

C++ Fences vs. Atomic Operations (3)

- Similar for atomic operations and fences with `std::memory_order_acquire`

```
1   std::atomic<int> myatomic(42);
2   int X = 0, Y = 0;
3
4   r1 = X; // Load of X, can be reordered with atomic load and store to Y
5
6   // Loads and Stores after this atomic operation
7   // cannot be reordered with the atomic load
8   int tmp = myatomic.load(1, std::memory_order_acquire);
9
10  Y = 41; // Store to Y, cannot be reordered with atomic load
```

```
1   std::atomic<int> myatomic(42);
2   int X = 0, Y = 0;
3
4   r1 = X; // Load of X, cannot be reordered with the store to Y
5
6   // Loads and Stores after this fence
7   // cannot be reordered with any load before the fence
8   std::atomic_thread_fence(std::memory_order_acquire);
9
10  Y = 41; // Store to Y
```