Paper title: Shared Memory Consistency Models: A Tutorial
Paper authors: Sarita V. Adve, Kourosh Gharachorloo
Your name: Parth Kansara (115135130)

What problem does the paper address? How does it relate to and improve upon previous work in its domain?
The paper talks about the various hardware-based shared memory models, used across uniprocessor and multiprocessor architectures. It addresses the differences in the behavior expected from the various memory consistency models, in particular, the various hardware and compiler optimizations that are enabled by each model. This model decides how a program runs on a particular system, and so it becomes significant to understand the effect of the model. The model affects *programmability*, *performance* and the *portability* of a program and thus has a huge impact on how parallel programs are developed.

For utilizing certain hardware and compiler optimizations without violating sequential consistency, several techniques have been discussed. **Kourosh et al.** provide two techniques for systems with hardware support for cache coherence - prefetching ownership of write operations delayed due to program order in cache-based systems using invalidation-based protocol & rolling back and reissuing read operations in dynamically scheduled processors. **Shasha and Snir** created an algorithm to detect the *safe* memory operations which do not violate sequential consistency when they are reordered, which can be used to implement the hardware and compiler optimizations. These works have been simplified and a clear understanding is imparted through the paper.

What are the key contributions of the paper?
To provide a thorough understanding of the various memory consistency models, the paper uses simple and uniform terminology, along with program examples to explain the caveats of each model. The paper also clears several misconceptions related to these models, by providing coherent examples. This can thus be used by computer science professionals to predict the behavior of these models and effectively write programs and optimizations based on them. Since most of the models have been explained in terms of the system optimizations enabled by them, this paper attempts to provide an alternative programmer-centric view that describes the models in terms of program behavior rather than hardware and compiler optimizations.

Uniprocessors require simple semantics for memory operations - the memory operations must occur sequentially as specified by the program order. This is supported by maintaining the uniprocessor and control dependencies.

In shared memory multiprocessors, sequential consistency is primarily used, which has two main requirements. Program order among operations from the same processor must be maintained (*program order*) and a single sequential order among operations from all the processors must be maintained (*atomicity*). In architectures that do not have a cache, the following hardware optimizations may violate sequential consistency - write buffers having bypassing capability, overlapping write operations and non-blocking read operations. Overlapping write operations can avoid violation of sequential consistency by enforcing a write operation from a processor to wait until the previous write operation from the same processor has reached its memory module by using an acknowledgement response. In architectures that do have a cache, there are three additional problems - ensuring cache coherence, detecting the completion of write operations and the maintenance of atomicity for the write operations. Furthermore, the compilers for shared memory parallel programs cannot directly apply several optimizations while also preserving sequential consistency.

Several *relaxed* memory consistency models have been suggested as an alternative to sequential consistency. One way to distinguish models that relax program order consistency is by the type of order they relax, which can be categorized as relaxation from a write to a subsequent read, between two writes, or from a read to a subsequent read or write. The paper also discusses a set of models that relax the program order between all the operations - weak ordering (WO) model, two flavors of the release consistency models namely RCsc and RCpc, Digital Alpha, SPARC V9 Relaxed Memory Order (RMO) and IBM PowerPC models. Models that relax the write atomicity requirement can be differentiated by their ability to permit a read operation to retrieve the value of a write operation from another processor prior to all cached copies of the accessed location receiving the invalidation or update message generated by the write.

The paper also suggests a programmer-centric specification that requires the programmer to provide information about the operations that may be involved in a race in a program which can be utilized by the system to decide the application of a particular optimization. This information can either be passed through the programming language or through the hardware.

Briefly describe how the paper's experimental methodology supports the paper's conclusions.
There is no explicit experimentation in this paper and thus, there is no elaborate answer for this question.

Write down one question you may want to ask the authors.
Can we eliminate the problem of portability of programs across hardwares with different memory consistency models by conveying the information about synchronization operations at the programming level?