Theorem: Consider a program P that is data-race-free. Then an execution E of program P that obeys SC- must be E SC.

Proof: We say an access X is before (respectively after) an access Y in an execution if X is ordered before (respectively after) Y by the serialization order of the execution. Similarly, we use “last” and “first” to implicitly refer to the serialization order.

We first prove some simple lemmas.

Lemma 1. A synchronization read in E returns the value of the last conflicting write ordered before it in E.

Proof: A validated synchronization read must return the value of a write that is race-consistent for it in E. By definition, this is the last conflicting write ordered before the read in E.

Lemma 2. Let Ev be an execution that is used to validate some read in E. Then a synchronization read in Ev returns the value of the last conflicting write ordered before it in Ev.

Proof: Only data reads can return non-SC values in Ev. Therefore, by definition, a synchronization read must return the value of the last conflicting write ordered before it in Ev.

Lemma 3. Let Ev be an execution that is used to validate some read in E. Then if X hb Y in Ev, then X is before Y in Ev. Similarly, if X hb Y in E, then X is before Y in E.

Proof: Recall that I hb J if (1) I is before J by program order, or (2) I is a synchronization write, J is a synchronization read, J returns the value written by I, or (3) I hb K hb J for some K.

Thus, if X hb Y in Ev due to a po edge, then X is before Y in Ev since the serialization order of Ev is consistent with program order by definition. If X hb Y in Ev due to a synch edge, then again X is before Y in Ev by Lemma 2. If X hb Y due to the transitive closure of po and synch edges, then transitivity applying the above observations, again X is before Y in Ev.

We refer to the first type of edge as a po edge and the second type as a synch edge (for synchronization order).

Lemma 4. Let Ev be an execution that is used to validate some read in E. Let R’ be a read that was previously validated or is validated by Ev. Then if R’ is a synchronization read, then R’ returns the same value in E and Ev.

Proof: The write whose value R’ returns in E must be race-consistent for Ev. Thus, this write must be the last conflicting write before R’ in Ev. It follows from Lemma 2 that R’ returns the value of this write in Ev.

Let Evl be the execution used to validate the last read in E. Evl contains all reads of E. There are two cases.

Case 1. Evl does not have a non-SC read.

Then Evl is an SC execution and cannot have data races. If all reads in Ervl return the same value as in E, then E is SC and we are done.

So assume for a contradiction that all reads in Evl do not return the same value as in E. Then there must be some read in Evl that occurs in both E and Evl and returns a different value. Let R be the first such read in Evl (according to its serialization order).

By Lemma 4, R must be a data read. Let R return the value of W in E. Since W is hb-consistent for R in Evl, it follows that we cannot have R hb W in Evl. Since Evl does not have data races, it must be that W hb R in Evl. By Lemma 3, W must be before R in Evl.
Case 2. Evl has a non-SC read.

Consider the first execution used for validation that has a non-SC read. Call it Ev. Consider the first non-SC read R in Ev. R returns the value of the same write W in Ev and E. Consider the execution Evr that was used to validate R. Evr cannot have any non-SC reads (because Ev is the first such execution used for validation). So Evr is SC and so has no data races. So W hb R in Evr. So W hb R in E. So W hb R in Ev as well. By the previous lemma, W is before R in Ev.

Since R is non-SC in Ev, there must be a conflicting write W' between W and R in Ev. Either W and W' form a data race in Ev or W' and R form a data race in Ev, since W is hb-consistent for R. Since there are no non-SC reads before R, we can take the execution Ev until R, make R read the value of the last conflicting write before it, and continue the execution without any SC reads. This is an SC execution of program P that has a data race (between W and W' or between W' and R). This is a contradiction.