In the last lecture, we saw the algorithm for Sequential Consistency. We proved that the sequential order is preserved in the permutation i.e. \( \forall i, \sigma|\pi = \pi|i. \) We will now prove that the permutation \( \pi|i \) is also legal. By legality we mean that read should return the value of the last write.

Consider any read \( r_i \) by \( P_i \) that reads \( v \) from object \( X \) and let \( w_j \) be the latest write that caused the latest update to \( P_i \)'s copy of \( X \) preceding the local read for \( r_i \).

By definition of \( \pi \), \( w_j < r_i \) in \( \pi \).

We need to prove that there is no other write in between \( w_j \) and \( r_i \) i.e. No \( w' \) exists such that \( w_j \prec \pi w' \prec \pi r_i \), where \( w' \) also writes to \( X \).

Suppose for contradiction that such a \( w' \) exists, where \( w' \) is a write by some processor \( P_k \).

By definition of \( \pi \), \( r_i \) is placed in \( \pi \) immediately after \( w_j (2^{nd \text{ condition}}) \) or immediately after the previous operation by \( P_i (1^{st \text{ condition}}) \).

Since \( w_j \prec \pi w' \prec r_i \), there has to be a previous operation by \( P_i \) between \( w' \) and \( r_i \). Let \( op_i \) be the earliest such operation, so we have,

\[
w_j \prec \pi w' \leq \pi op_i < \pi r_i.
\]

According to 1\(^{st} \) condition, \( op_i < r_i \) in \( \sigma|i \) as well. We now have 2 cases depending on whether \( op_i \) is a read or a write.

**Case 1:** \( op_i \) is a write. By Lemma 1, since \( w_j < \pi w' \), update for \( w_j \) precedes the update for \( w' \). By Lemma 2, since \( op_i \) precedes \( r_i \) in \( \sigma|i \), we know that the same ordering holds i.e. update of \( op_i \) precedes local read of \( r_i \) in \( P_i \)'s copy.

By Lemma 1, update for \( w_j < \text{update for } w' \leq \text{update for } op_i < \text{local read of } i \). Since \( w_j \) and \( w' \) write to the same object, read should read the value written by \( w' \).

**Case 2:** \( op_i \) is a read of object \( Y \).

By definition of \( \pi \), \( op_i \) is placed in \( \pi \) immediately after the previous operation by \( P_i (1^{st \text{ condition}}) \) or the latest write \( w'' \) of \( Y \) whose update precedes local read for \( op_i \).

Since \( op_i \) is the first operation by \( P_i \) that we picked after \( w' \), condition 1 cannot hold. Therefore 2\(^{nd \text{ condition}} \) holds i.e. \( op_i \) is placed immediately after the latest write \( w'' \) of \( Y \) whose update precedes local read for \( op_i \). Therefore,

\[
w_j < \pi w' \leq \pi w'' < \pi op_i < \pi r_i.
\]

By Lemma 1, update for \( w_j \) precedes update for \( w' \) which precedes update for \( w'' \).

By definition of \( w'' \), update for \( w'' \) precedes local read by \( op_i \). By Lemma 2, since \( op_i < r_i \) in \( \sigma|i \), local reads are also in the same order. Hence, \( r_i \) should have read the value of \( w' \).
1 Local Write Algorithm

We looked at the Local Read Algorithm previously, where read takes place with 0 delay and write has a delay \( d \) associated with it. Next we look at a Local Write Algorithm where write returns immediately with 0 delay and read operation has a delay \( d \) associated with it. This algorithm is suitable if the application requires lots of write’s and very few read’s.

Algorithm for Processor \( P_i \)

Algorithm 1 Local Write Algorithm

1. When \( read_i(X) \) occurs
   
   if \( num = 0 \) then
      return \((X, copy[X])\)
   end if

2. When \( write_i(X, v) \) occurs
   
   \( num := num + 1 \)
   \( tbc - send_i(X, v) \)
   \( ack_i(X) \)

3. When \( tbc - recv_i(X, v) \) from \( P_j \) occurs
   
   \( copy[X] = v \)
   if \( j = i \) then
      \( num := num - 1 \)
      if \( num = 0 \) and \( read \) on \( X \) is pending then
         return \((X, copy[X])\)
      end if
   end if

The proofs will be similar to the Local Read Algorithm case as discussed in the last lecture.

2 Models of complexity

1. Perfect clocks with known message delay = \( d \).
2. Perfect Clocks.
3. Message delay = \( d \), No perfect clocks.
4. No perfect clocks, message delay between \( (d - u, d) \)

It is quite clear that Model 4 is the weakest. The remaining three models however are quite identical i.e If we have an algorithm for either one of the first 3 models, the remaining 2 can be
simulated using clock synchronization. For example if we have a Model 3 system, where there is only the known message delay time = \( d \). The processors, can synchronize their clocks in the following way.

Processor \( P_1 \) assigns a time-stamp \( t_1 \) (\( P_1 \)'s local clock) with the messages that it sends. Now if \( P_2 \) receives this message at time \( t_2 \) (\( P_2 \)'s local clock), it can easily calculate that the message was sent at time \( t_2 - d \) (\( P_2 \)'s local clock). Therefore \( P_2 \) knows that \( t_2 - d \) on \( P_2 \)'s clock is the same as \( t_1 \) on \( P_1 \)'s clock i.e. they correspond to the same real time. Each processor can similarly calculate this offset for every other processor \( P_i \) thus simulating perfect clocks with known message delivery as in Model 2.

Because of the compatibility between models 1, 2, and 3, we can just focus on Model 1 and Model 4 since Models 2 and 3 are identical to Model 1 in complexity. In the following subsection we prove a Theorem which gives a lower bound on the Read and Write times for a sequential consistent implementation of shared memory using Model 1.

**Theorem 1** Suppose we have a distributed system with perfect clocks and an exact message delivery time of \( d \) (Model 1). For any sequentially consistent Memory Consistency System (MCS) that provides at least 2 read/write objects, \( t_{\text{read}} + t_{\text{write}} \geq d \).

**Proof:** We will prove the above theorem by contradiction. For contradiction, we have to come up with an execution where write precedes read and \( t_{\text{read}} + t_{\text{write}} < d \). Then we can argue that in less than \( d \) time, a message send did not make it to another processor.

Let initial values of objects \( X \) and \( Y \) be 0. Assume for contradiction that an algorithm exists such that \( t_{\text{read}} + t_{\text{write}} < d \).

Let \( \alpha_1 \) be the execution where processor \( P_1 \) executes the following code.

\[
[\text{Write}_1(X, 1), \text{Ack}_1(X)] \ [\text{Read}_1(Y), \text{Return}(Y, 0)]
\]

Also, let \( \alpha_2 \) be the execution where processor \( P_2 \) executes the following code.

\[
[\text{Write}_2(Y, 1), \text{Ack}_2(Y)] \ [\text{Read}_2(X), \text{Return}(X, 0)]
\]

Assuming both \( \alpha_1 \) and \( \alpha_2 \) start at \( t = 0 \), each of the two executions are valid since the write and the read are to different registers, and the read therefore returns the correct value 0 which is the initial value of the register. Now, by our assumption both \( \alpha_1 \) and \( \alpha_2 \) finish in less than \( d \) time. Therefore, if any message is sent at any time within \( \alpha_1 \) or \( \alpha_2 \), it won’t arrive before the end of the execution, since each message takes exactly \( d \) time to be delivered.

Now consider the new execution \( \alpha_3 \) that is the combination of both \( \alpha_1 \) and \( \alpha_2 \). We claim that this is a valid execution. Note that neither processor would receive any messages in \( \alpha_3 \) since it completes before time \( t = d \), and any message sent in the execution would take \( d \) time to be delivered which is after the execution has terminated. Therefore, \( \alpha_1 \overset{P_1}{\sim} \alpha_3 \), since \( P_1 \) cannot distinguish between \( \alpha_1 \) and \( \alpha_3 \). Similarly \( P_2 \) cannot distinguish between \( \alpha_2 \) and \( \alpha_3 \) i.e \( \alpha_2 \overset{P_2}{\sim} \alpha_3 \).

Hence the execution is correct because there is no way for one process to know about the other process’ write operations.
To show that $\alpha_3$ is a sequentially consistent execution, we need to give a serialization $\tau$ of all the operations. So we have $W_1 > R_2$ and $W_2 > R_1$ to satisfy legality. Also, we have $W_1 < R_1$ and $W_2 < R_2$ to satisfy program order at each processor.

This is a non-sequentially consistent execution because as per the above conditions,

$$W_1 < R_1 < W_2 < R_2 < W_1$$

which is a contradiction.

There is no way to construct $\tau$ to satisfy the constraints of Sequential Consistency giving a contradiction. This proves that $t_{\text{read}} + t_{\text{write}} \geq d$. $\blacksquare$