When a system crash occurs, we must consult the log to determine those transactions that need to be redone and those that need to be undone. In principle, we need to search the entire log to determine this information. There are two major difficulties with this approach:

1. The search process is time-consuming.
2. Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.

To reduce these types of overhead, we introduce checkpoints. We describe below a simple checkpoint scheme that

(a) does not permit any updates to be performed while the checkpoint operation is in progress, and

(b) outputs all modified buffer blocks to disk when the checkpoint is performed.

A checkpoint is performed as follows:

1. Output onto stable storage all log records currently residing in main memory.
2. Output to the disk all modified buffer blocks.
3. Output onto stable storage a log record of the form <checkpoint L>, where L is a list of transactions active at the time of the checkpoint.

Transactions are not allowed to perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in progress. The presence of a <checkpoint L> record in the log allows the system to streamline its recovery procedure. Consider transaction Ti that completed prior to the checkpoint. For such a transaction, the <Ti commit>record (or < Ti abort> record) appears in the log before the <checkpoint> record. Any database modifications made by Ti must have been written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on Ti.

After a system crash has occurred, the system examines the log to find the last <checkpoint L> record (this can be done by searching the log backward, from the end of the log, until the first <checkpoint L> record is found).

The redo or undo operations need to be applied only to transactions in L, and to all transactions that started execution after the <checkpoint L> record was written to the log. Let us denote this set of transactions as T.

* For all transactions Tk in T that have no <Tk commit> record or <Tk abort> record in the log, execute undo(Tk).
* For all transactions Tk in T such that either the record <Tk commit> or the record <Tk abort> appears in the log, execute redo(Tk).

Note that we need only examine the part of the log starting with the last check point log record to find the set of transactions T, and to find out whether a commit or abort record occurs in the log for each transaction in T.

As an illustration, consider the set of transactions {T0, T1,...,T100}. Suppose that the most recent checkpoint took place during the execution of transaction T67 and T69, while T68 and all transactions with subscripts lower than 67 completed before the checkpoint. Thus, only transactions T67, T69,...,T100 need to be considered during the recovery scheme. Each of them needs to be redone if it has completed (that is, either committed or aborted); otherwise, it was incomplete, and needs to be undone.

Consider the set of transactions L in a checkpoint log record. For each transaction Ti in L, log records of the transaction that occur prior to the checkpoint log record may be needed to undo the transaction, in case it does not commit.

However, all log records prior to the earliest of the < Ti start>log records, among transactions Ti in L, are not needed once the checkpoint has completed. These log records can be erased whenever the database system needs to reclaim the space occupied by these records.

The requirement that transactions must not perform any updates to buffer blocks or to the log during checkpointing can be bothersome, since transaction processing has to halt while a checkpoint is in progress. A fuzzy checkpoint is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written out. Section 16.5.4 describes fuzzy-checkpointing schemes. Later in Section 16.8 we describe a checkpoint scheme that is not only fuzzy, but does not even require all modified buffer blocks to be output to disk at

the time of the checkpoint.