检查点（checkpoint）

undo日志中的检查点

- 上述恢复过程需要扫描整个undo日志文件，为了降低数据库故障恢复的开销，可以定期地在日志文件中插入检查点。
- 在日志中插入检查点的处理过程包括：

1. 系统停止接受‘启动新事务的请求’；
2. 等到所有当前活跃的事务被提交或中止，并且在日志中写入了<Commit T>或<Abort T>记录；
3. 将日志记录刷新到磁盘；
4. 写入日志记录<CKPT>，并再次刷新日志；
5. 重新开始接受新的事务。

undo日志中的检查点

- 在进行故障恢复时，只要逆向扫描到第一条<CKPT>记录(最后一个被记入日志的检查点)就可以结束故障恢复工作。

设置检查点的例子

- 日志的开始情况如图 **1**所示

undo日志
<Start T_1>
<T_1, A, 5>
<Start T_2>
<T_2, B, 10>

图 **1**

- 假设现在需要插入一个检查点，则插入检查点后的日志可能如图 **2**所示

undo日志
<Start T_1>
<T_1, A, 5>
<Start T_2>
<T_2, B, 10>
<T_2, C, 15>
<T_1, D, 20>
<Commit T_1>
<Commit T_2>
<CKPT>
<Start T_3>
<T_3, E, 25>

图 **2**

- 非静止检查点
  - 在设置检查点的过程中，允许新的事务进入系统。
  - 设置‘非静止检查点’的步骤包括

1. 写入日志记录<Start CKPT($T_1, \ldots, T_k$)>，并刷新日志；
   - $T_1, \ldots, T_k$是当前所有活跃事务的标识符
2. 等待$T_1, \ldots, T_k$中的每一个事务的提交或中止，但允许开始执行其它新的事务；
3. 当$T_1, \ldots, T_k$都已经完成时，写入日志记录<End CKPT>并刷新日志。

设置非静止检查点的例子

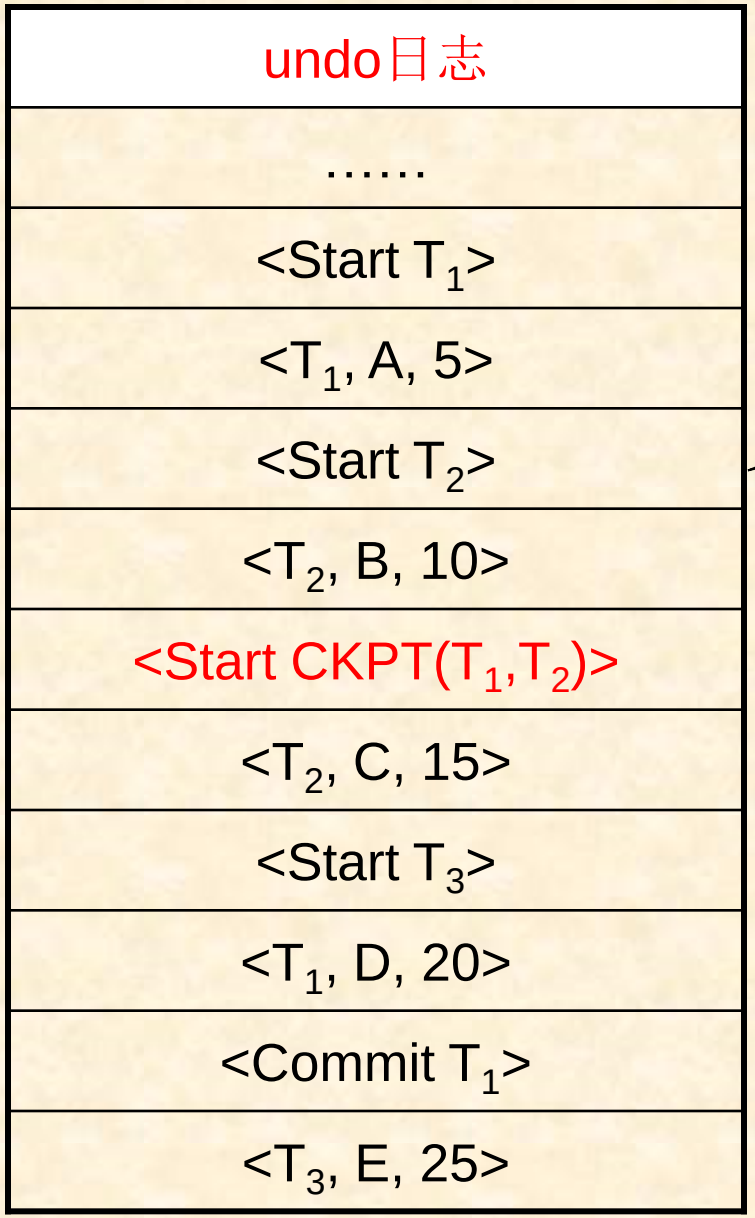
图 3

undo日志
<Start T_1>
<T_1, A, 5>
<Start T_2>
<T_2, B, 10>
<Start CKPT(T_1, T_2)>
<T_2, C, 15>
<Start T_3>
<T_1, D, 20>
<Commit T_1>
<T_3, E, 25>
<Commit T_2>
<End CKPT>
<T_3, F, 30>

带有非静止检查点日志的恢复

- 从日志尾部向后扫描日志文件进行故障恢复。
- 如果先遇到<End CKPT>记录，则继续向后扫描，直到出现对应的<Start CKPT(...)>记录就可以结束故障恢复工作，在这之后的日志记录是没有用处的，可以被抛弃。
- 如果先遇到<Start CKPT(T_1, ..., T_k)>记录，此种情况下的故障恢复工作需要撤消两类事务的操作：
  - 在<Start CKPT(T_1, ..., T_k)>记录之后启动的事务
    - 在扫描到<Start CKPT(T_1, ..., T_k)>记录时，这类事务的操作已经被撤消。
  - T_1, ..., T_k中在系统崩溃前尚未完成的事务
    - 继续向后扫描日志，直至其中未完成事务的访问操作被全部撤消。

带有非静止检查点的故障恢复（图 4）



redo日志的检查点

- 在redo日志中插入检查点时，由于已提交事务所做的修改被写入数据库磁盘的时间可能比事务提交的时间要晚得多，因此在插入检查点时，不仅仅需要考虑当前有哪些事务是活跃的，还要确保当前已提交事务的所有修改被写入到数据库的磁盘中去。

- 为了做到这一点，系统必须知道：
  1. 有哪些内存缓冲区被修改过，但还没有将修改结果写入磁盘？
  2. 每一个内存缓冲区都被哪些事务修改过？每个事务修改后的结果是什么？

- 在redo日志中插入(非静止)检查点的步骤

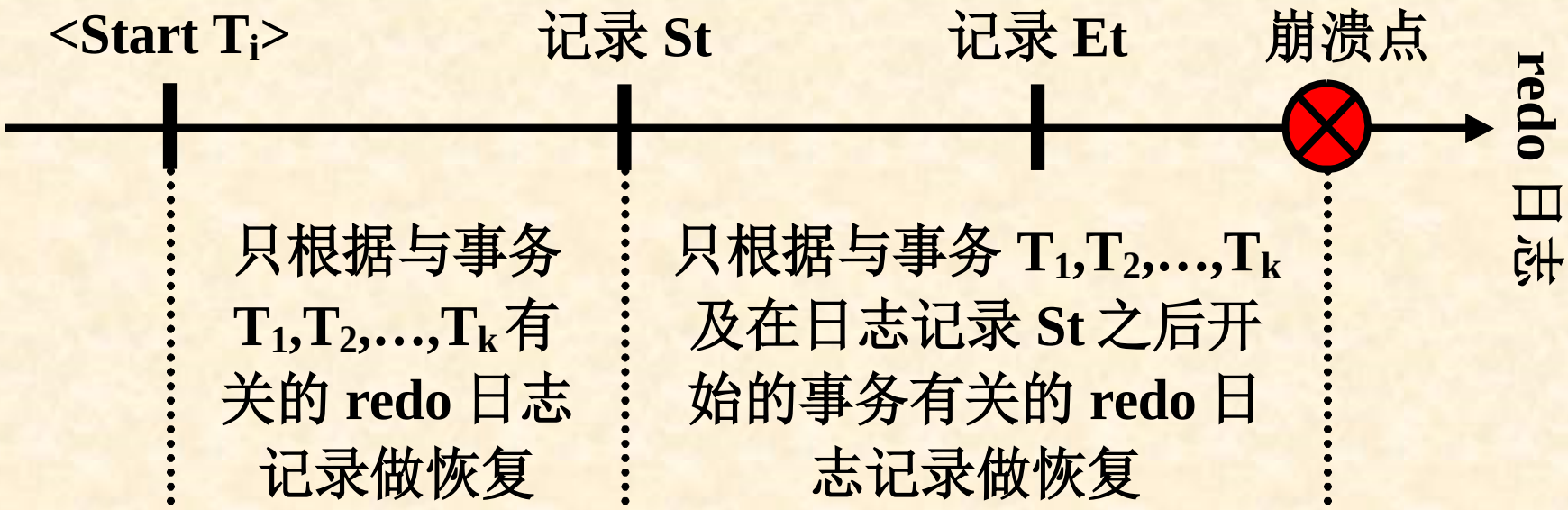
1. 写入日志记录<Start CKPT(T_1, …, T_k)>，并刷新日志；
   - 其中：T_1, …, T_k是当前所有活跃事务的标识符
   - 同时获得当时所有已提交事务的标识符集合S
2. 将集合S中的事务已经写到内存缓冲区但还没有写到数据库磁盘的数据写入磁盘；
3. 写入日志记录<End CKPT>并刷新日志。
   - 不必等待事务T_1, …, T_k或新开始事务的结束

图 5　带有非静止检查点的redo日志

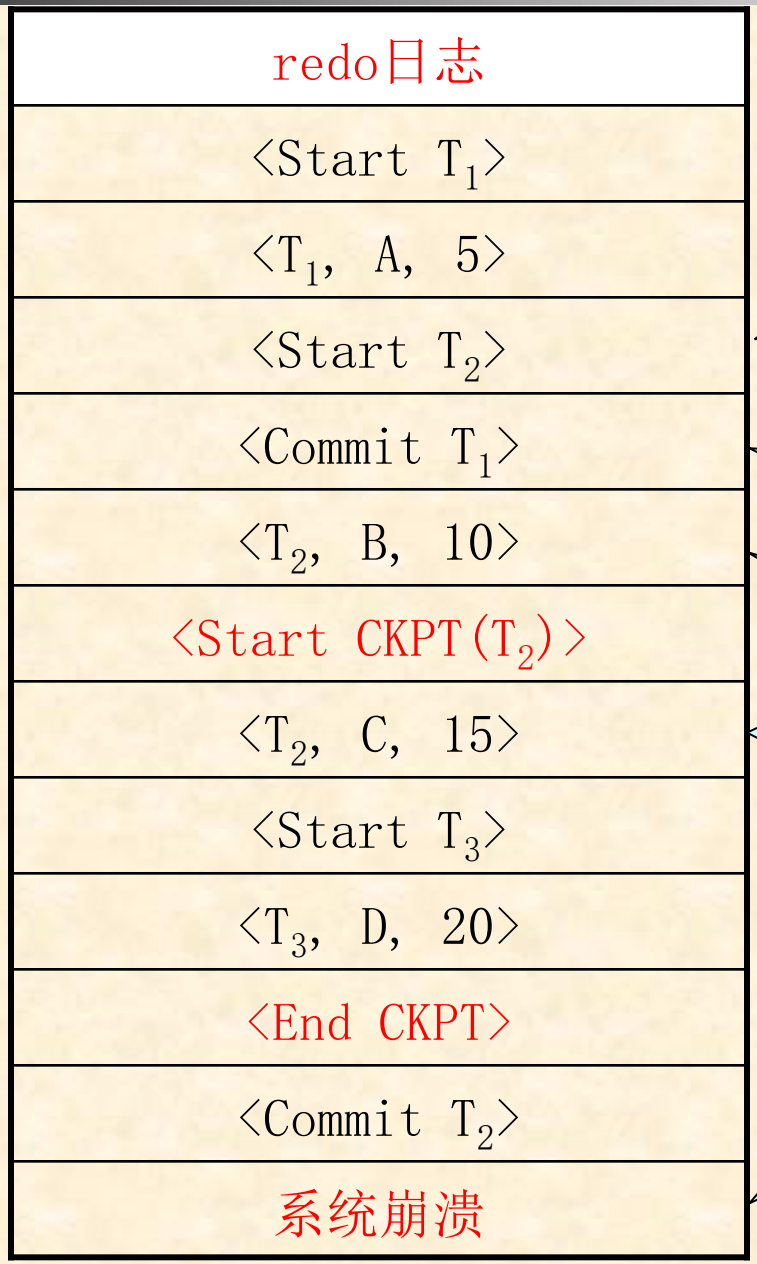
redo日志
<Start T_1>
<T_1, A, 5>
<Start T_2>
<Commit T_1>
<T_2, B, 10>
<Start CKPT(T_2)>
<T_2, C, 15>
<Start T_3>
<T_3, D, 20>
<End CKPT>
<Commit T_2>
<Commit T_3>

使用带检查点的redo日志的恢复

- 寻找最后一个被记入日志的检查点记录：
  - 如果是<End CKPT>(记为记录Et)，假设与之相对应的检查点记录是<Start CKPT(T_1, …, T_k)>(记为记录St)，并找到最早出现的<Start T_i>(记为记录t_i)，则故障恢复方法如下：针对事务T_1, …, T_k以及在St之后开始的那些事务，重做其中已经被提交的事务



例：使用带有检查点的redo日志进行恢复



从这里开始redo操作

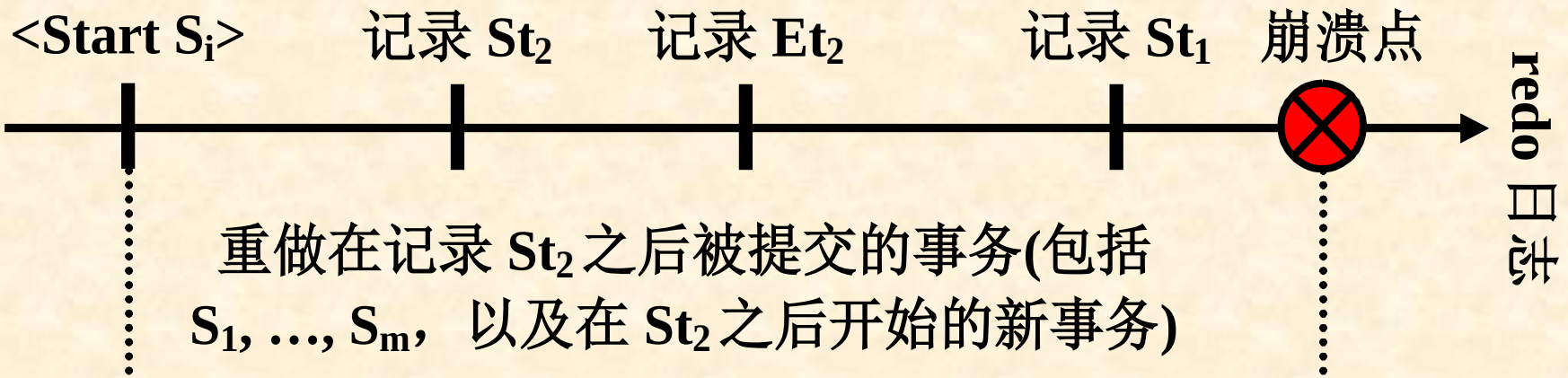
在<Start CKPT(T_2)>之前已经结束的事务(T_1)不必处理

重做在<Start CKPT(T_2)>之后，崩溃发生之前已经被提交的事务(T_2)

最后为尚未结束事务(T_3)加入结束标志<Abort T_3>

使用带检查点的redo日志的恢复（续）

- 寻找最后一个被记入日志的检查点记录：
  - 如果是<StartCKPT(T_1, …, T_k)>（记为记录St_1），我们继续寻找前一个<EndCKPT>（记为记录Et_2），以及与Et_2相对应的<StartCKPT(S_1, …, S_m)>（记为记录St_2）；
  - 该情况就如同日志记录Et_2是日志文件中的最后一条检查点（<EndCKPT>）记录一样进行恢复。



- 在undo/redo日志中插入检查点

1. 写入日志记录<Start CKPT(T_1, ..., T_k)>，并刷新日志。其中：T_1, ..., T_k是当前所有活跃事务的标识符；
2. 将所有被修改过的缓冲区写到数据库的磁盘中去；
3. 写入日志记录<End CKPT>并刷新日志。

带有检查点的undo/redo日

	undo/redo日志
1	<Start T_1>
2	<T_1, A, 4, 5>
3	<Start T_2>
4	<Commit T_1>
5	<T_2, B, 9, 10>
6	<Start CKPT(T_2)>
7	<T_2, C, 14, 15>
8	<Start T_3>
9	<T_3, D, 19, 20>
10	<End CKPT>
11	<Commit T_2>
12	<Commit T_3>

❑在下述情况下如何进行数据库的故障恢复？（哪些事务将被重做，哪些事务将被撤消，从哪一步开始，到哪一步结束）

1. 在第12步之后发生故障
2. 在第11步与第12步之间发生故障
3. 在第10步与第11步之间发生故障
4. 在第9步与第10步之间发生故障