ARIES

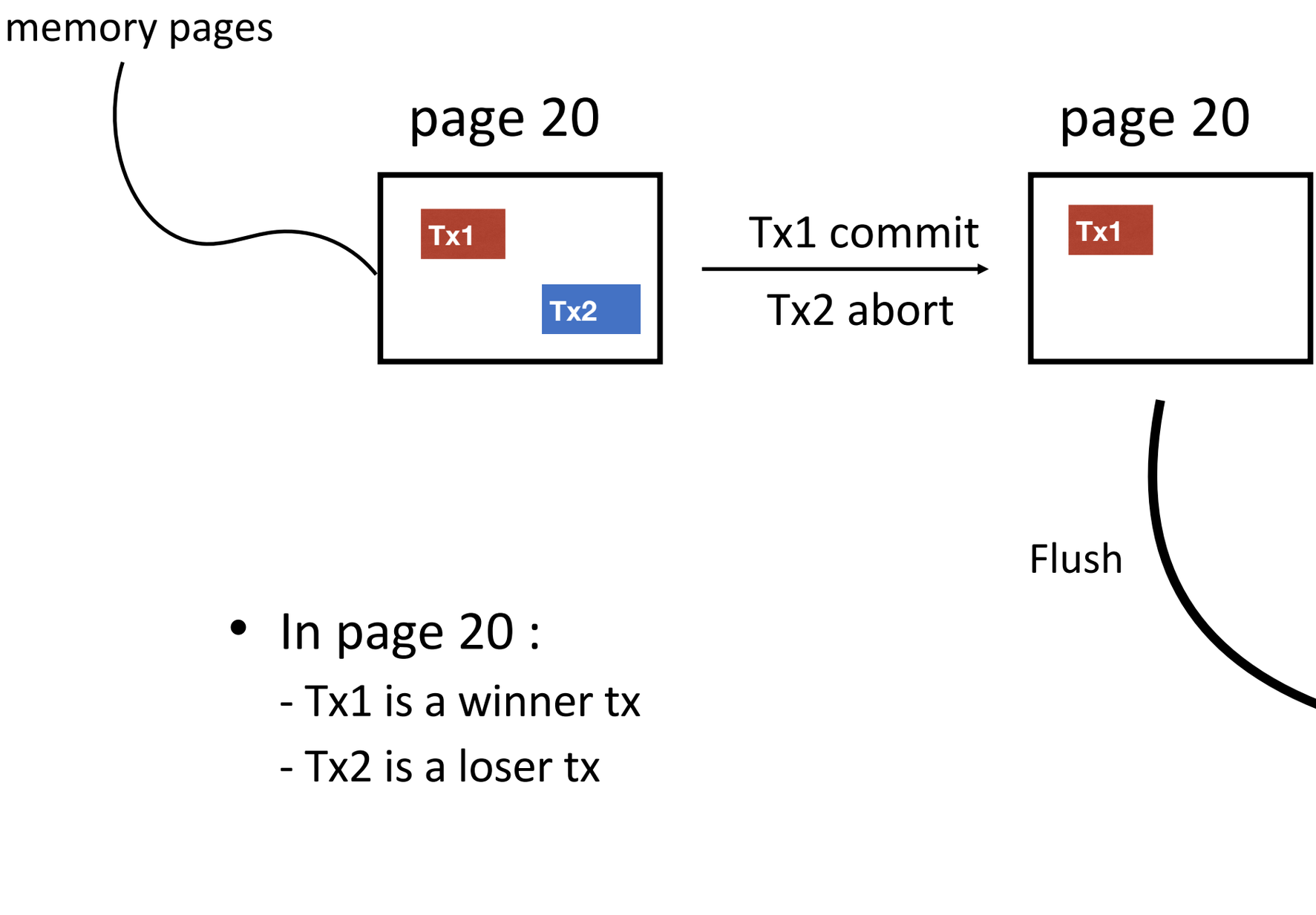
CS471000 Introduction to Database Systems

2023 Spring

What is ARIES ?

- A recovery algorithm.
- Published by IBM Research in early 1990s.
- Most commercial databases implement ARIES.

What we expected



- In page 20 :
  - Tx1 is a winner tx
  - Tx2 is a loser tx

However…

- Steal
  - Due to buffer management, dirty pages may be flushed to disk before txs commit
  - The changes made by **loser** txs must be UNDO
- No Force
  - Due to performance reason, dirty page won’t be flush immediately after txs commit
  - The changes made by **winner** txs must be REDO

Logs in ARIES

Physical Log Record

- Format:

  <**Op Code**, **txNum**, fileName, blockNum, offset, sqlType, **oldVal**, **newVal** >

- REDO:

  - Apply newVal to the page

- UNDO:

  - Apply oldVal to the page

  - Append its ***Compensation Log Record***

Compensation Log Record

- A **CLR** describes the actions taken to undo an operation.
- CLRs are added to log like any other record.
- CLRs is **Redo** ONLY. Why?
- It has all the fields of an update log record plus the **undoNext** pointer (the next-to-be-undone LSN).

Why CLR Needs UndoNext?

- UndoNext helps skip logs which have been Undone by Redo

[0] <Start 1>

[1] <SetVal , 1 , Page 20 , 0 , 1>

[2] <SetVal , 1 , Page 20 , 1 , 2>

[3] <SetVal , 1 , Page 20 , 2 , 3>

Crash Here !

[4]<SetValClr 1, Page 20 , 3 , 2 , [3] > // Append Undo [3] Redo log

Crash Again !

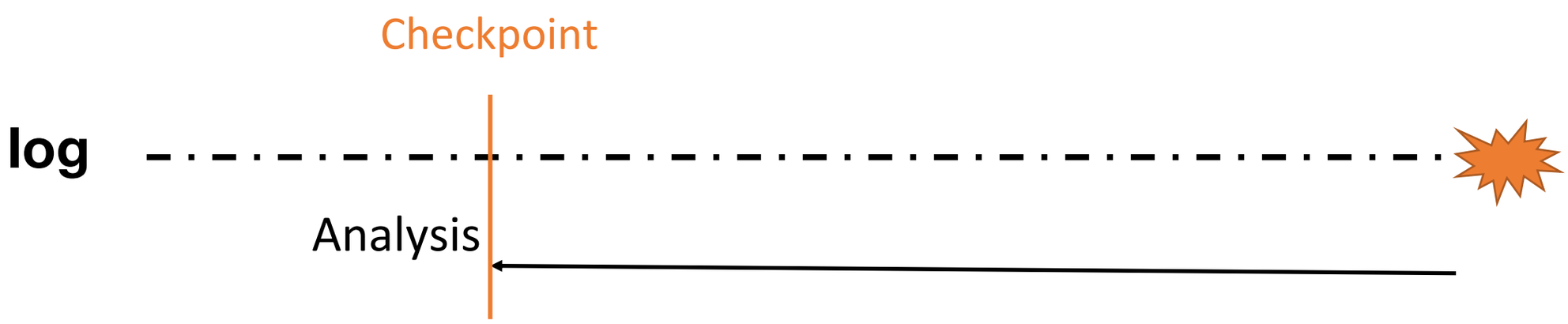
[5]<SetValClr 1, Page 20 , 2 , 1 , [2] > // Append Undo [2] Redo log

[6]<SetValClr 1, Page 20 , 1 , 0 , [1] >

Recovery Phases of ARIES

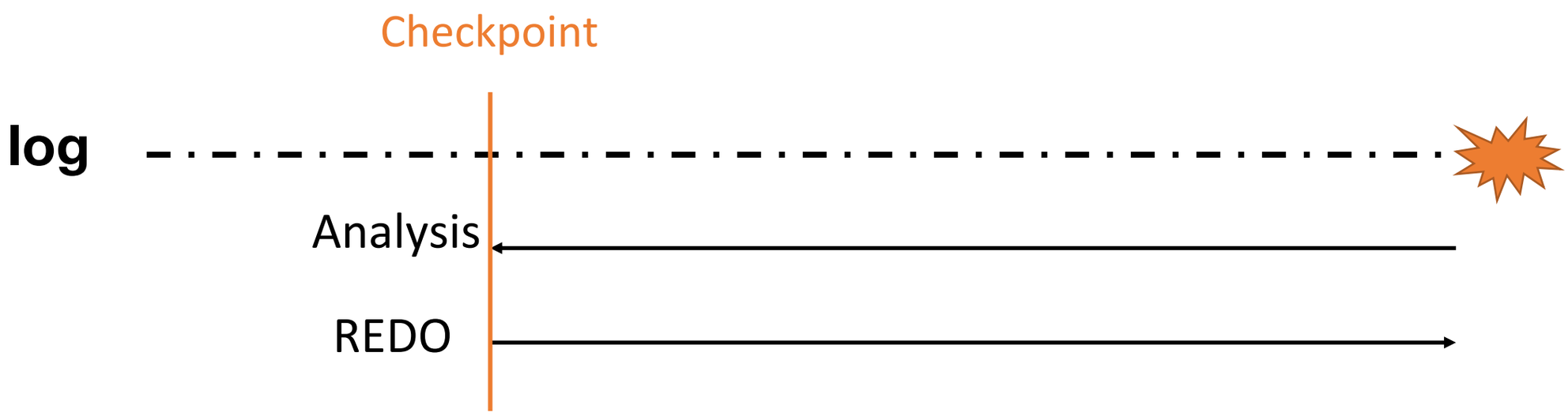
Recovery Phases of ARIES

- Analysis Phase
  - Find the earliest possibly start point of dirty page
  - Find loser txs



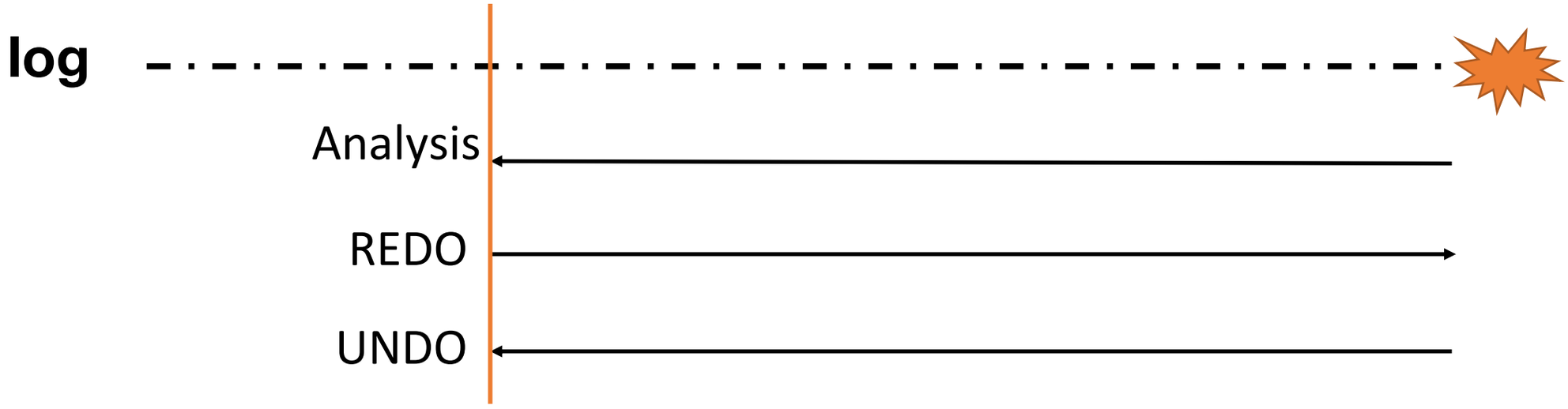
Recovery Phases of ARIES

- Analysis Phase
  - Find the earliest possibly start point of dirty page
  - Find loser txs
- REDO Phase
  - Repeat history (*both* winner and loser changes)
  - Recovery exact page status when the failure occurred

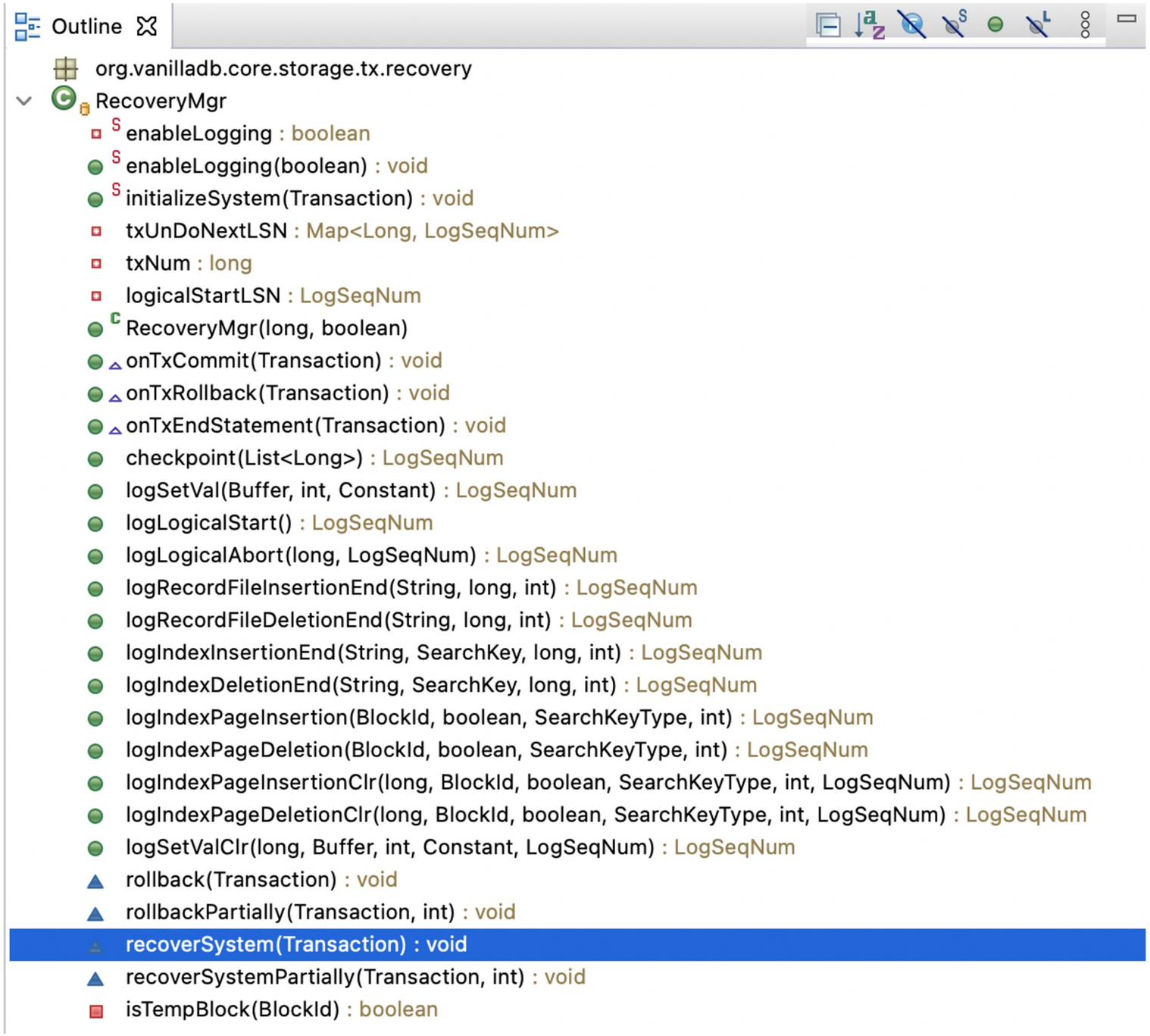


Recovery Phases of ARIES

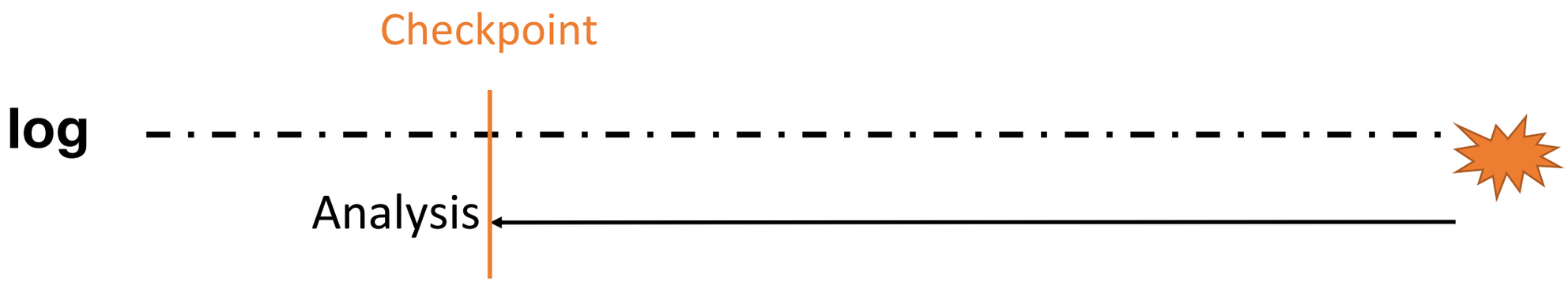
- Analysis Phase
  - Find the earliest possibly start point of dirty page
  - Find loser txs
- REDO Phase
  - Repeat history (*both* winner and loser changes)
  - Recovery exact page status when the failure occurred
- UNDO Phase
  - Rollback *loser* txs changes



Recovery in VanillaDB



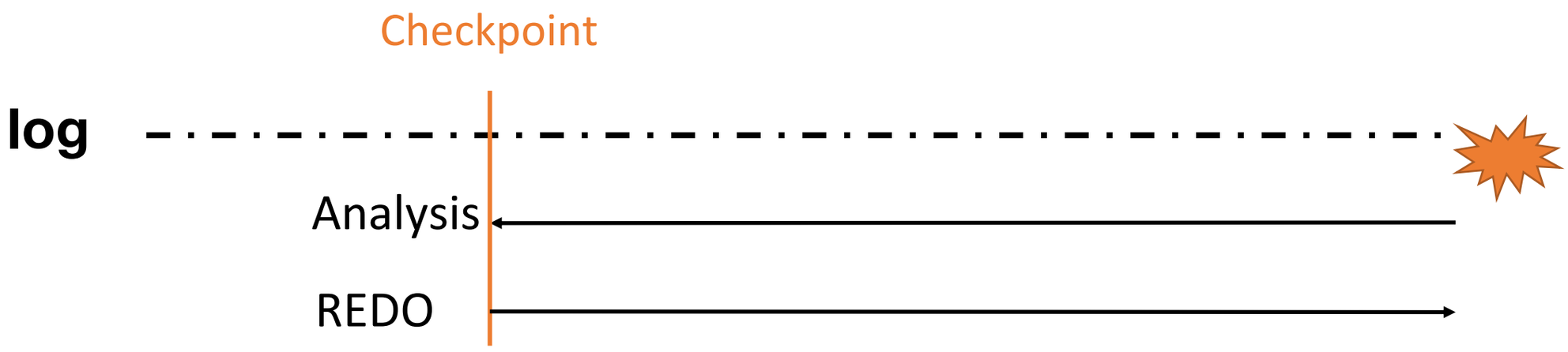
Analysis Phase




```
// analyze phase
while (iter.hasNext()) {
    LogRecord rec = iter.next();
    int op = rec.op();
    if (op == OP_CHECKPOINT) {
        // Since we flush all dirtyPage at checkpoint, therefore no need
        // to find the start record of active txNum
        txsOnCheckpointing = ((CheckpointRecord) rec).activeTxNums();
        for (long acTxn : txsOnCheckpointing) {
            // txNum give us info of possible unFinshedTxs,
            // Check if those weren't in finishedTxs, and add it to the
            // uncompletedTxs
            if (!finishedTxs.contains(acTxn))
                unCompletedTxs.add(acTxn);
        }
        // Start Redo From checkpoint
        break;
    }

    if (op == OP_COMMIT) {
        finishedTxs.add(rec.txNumber());
    } else if (op == OP_ROLLBACK) {
        finishedTxs.add(rec.txNumber());
    } else if (op == OP_START && !finishedTxs.contains(rec.txNumber())) {
        unCompletedTxs.add(rec.txNumber());
    }
}
```

Redo Phase




```
/*
 * redo phase: Repeating History
 */

while (iter.hasPrevious()) {
    LogRecord rec = iter.previous();

    rec.redo(tx);
}
```

Undo Phase

```
/*
 * undo phase: undo all actions performed by the active txs during last
 * crash
 */

while (iter.hasNext()) {
    LogRecord rec = iter.next();

    int op = rec.op();
    if (!unCompletedTxs.contains(rec.txNumber()) || op == OP_COMMIT || op == OP_ROLLBACK)
        continue;
    /*
     * Use UnDoNextLSN to skip unnecessary physical record which have
     * been redo its undo by CLR or records have been rolled back
     */

    if (txUnDoNextLSN.containsKey(rec.txNumber())) {
        if (txUnDoNextLSN.get(rec.txNumber()).compareTo(rec.getLSN()) != 1)
            continue;
    }
    if (op == OP_START)
        unCompletedTxs.remove(rec.txNumber());
    else if (rec instanceof LogicalEndRecord) {

        // Undo this Logical operation;
        rec.undo(tx);

        LogSeqNum logicalStartLSN = ((LogicalEndRecord) rec).getlogicalStartLSN();
        /*
         * Save the Logical Start LSN to skip the log records between
         * the end record and the start record
         */
        txUnDoNextLSN.put(rec.txNumber(), logicalStartLSN);

    } else if (rec instanceof CompesationLogRecord) {

        LogSeqNum undoNextLSN = ((CompesationLogRecord) rec).getUndoNextLSN();
        /*
         * Save the UndoNext LSN to skip the records have been rolled
         * back
         */
        txUnDoNextLSN.put(rec.txNumber(), undoNextLSN);
    } else
        rec.undo(tx);

    if (unCompletedTxs.size() == 0)
        break;
}
```

Reference

- ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging