MOTIVATION

0. Databases are cake, right?

--Many of you have *used* DBs. Some of you have even used transactions.
--But you probably took transactions for granted.
--Yet, if we reflect, transactions are magical:
   A set of unreliable components (disk, memory, software) give an
   extremely strong guarantee. This guarantee applies even when the DBMS
   is executing a complex, disruptive operation at the time of a crash.
--Today, we ask how DBMSes give this guarantee, i.e., how they _implement_
   atomicity.

1. Scenario and goals

--Scenario:
   --We assume only one writer (so ignore the problem of isolation)
   --Crashes are a fact of life.

--Goals: After a crash, we, the system designers, insist that:
   --Committed transactions are logically "in the database"
   --Uncommitted transactions are not

--Do these goals sound easy to you?
   --Observe, file systems do not actually have this property
   --And ARIES needs 67 pages to tell us how they solved the problem....

2. Why is this problem so hard?

Because DBMS insists on good performance, "DB" actually in three places:

--In write-ahead log (WAL)
--In structured form on "non-volatile storage" a/k/a "cell storage" a/k/a
   "backing store" (we call it "n.v.s." today)
--Cached in RAM (in "buffer pool"):  
   --"Both" WAL disk pages and n.v.s. disk pages may be cached in RAM
   --For today, we assume *only n.v.s.* disk pages are in buffer pool

ARIES

3. Before describing ARIES, compare to the course notes:

--Course notes (chapter 9C; also presented in lecture on 4/12):
   --First, undo losers
   --Then, redo winners

--ARIES
   --First, take analysis pass to see what DBMS must do to recover
   --Then, redo _everything_ that happened up until the crash_
   --Then, undo the losing transactions

4. Example (see blackboard)

5. What about crashes _during_ recovery? (Compare to protocol in Chapter 9C.)

NOTE: Concluding thoughts on reverse
CONCLUDING THOUGHTS

What is special about ARIES?
--The standard in logging and recovery protocols
--Gather in one place many of the details about how to do locking, logging,
and recovery in a high-performance way
--Some very neat ideas

Examples of neat things (not comprehensive list):
--Shows how to make recovery recoverable (i.e., avoid wasting work
  if a crash happens _during_ recovery).
--The identical mechanism that buys the above property also gives partial
  rollbacks more-or-less for free; for reference, the mechanism is CLRs.
--Uses logical UNDO logging ("remove XYZ from table ABC") rather than
  physical UNDO logging ("write this 30 character string to offset 79 of
  page 356")

Why is this paper so difficult to read?
--These ideas are subtle
--The paper is unclear in key places

With all of these moving parts (we have only scratched the surface; there is
locking, latching, partial rollbacks, etc.), how do the designers know that
their approach is correct?
--They maintain certain invariants
--They have conceptually simple models (e.g., three-pass recovery)
--But ultimately, maybe the designers _didn’t_ get it right!
--We can legitimately wonder about the source code that implements ARIES

If you’re wondering why ARIES needs a separate analysis pass.....
--They don’t actually need it for correctness. (See the text before 6.2.)
--But analysis pass can help in a few ways:
  --Most of these ways are listed on p.156, (1)-->(7), of the paper
  --There is a pedagogical benefit as well, which is that the analysis
    pass makes it easier for everyone to reason about how recovery
    works. After analysis, we know that DPT and transaction table will
    be in the same state that they were in at the time of the crash.

If you’re wondering about the purpose of CLRs.....
--CLRs offer bounded logging during restart. Thus, if crashes occur
  during restart, the log does not grow out of control (other recovery
  systems do not have this property).
--ARIES also uses CLRs to implement partial rollbacks (as mentioned above).
--The main idea behind CLRs: they ensure that an undo operation only moves
  forward. For example, say that to undo a transaction, the DBMS must undo
  LSNs 60, 50, 40, and 30. Now, assume that the DBMS gets as far as
  undoing 60 and 50 before crashing. After a crash, the DBMS will continue
  to undo, but it will begin to undo at 40.
--Confusingly, although CLRs are not undone (that is, undoes cannot be
  undone), they *can* be REDONE. In more detail, after a crash, the DBMS
  may need to reapply a CLR, expressing the idea that the undone action
  needs to be reapplied if the results of the undo did not make it to
  n.v.s. before the crash.