INTRODUCTION TO TRANSACTION PROCESSING

CHAPTER 21 (6/E)

CHAPTER 17 (5/E)

LECTURE OUTLINE

- Introduction to Transaction Processing
- Desirable Properties of Transactions
- Transaction Support in SQL

DEFINITIONS

- **Transaction**: an executing program (process) that includes one or more database access operations
  - A logical unit of database processing
  - Example from banking database: Transfer of $100 dollars from a chequing account to a savings account
  - Characteristic operations
    - **Reads** (database retrieval, such as SQL SELECT)
    - **Writes** (modify database, such as SQL INSERT, UPDATE, DELETE)
- *Note*: Each execution of a program is a distinct transaction with different parameters
  - Bank transfer program parameters: savings account number, chequing account number, transfer amount
- **Online Transaction Processing (OLTP) Systems**: Large multi-user database systems supporting thousands of concurrent transactions (user processes) per minute

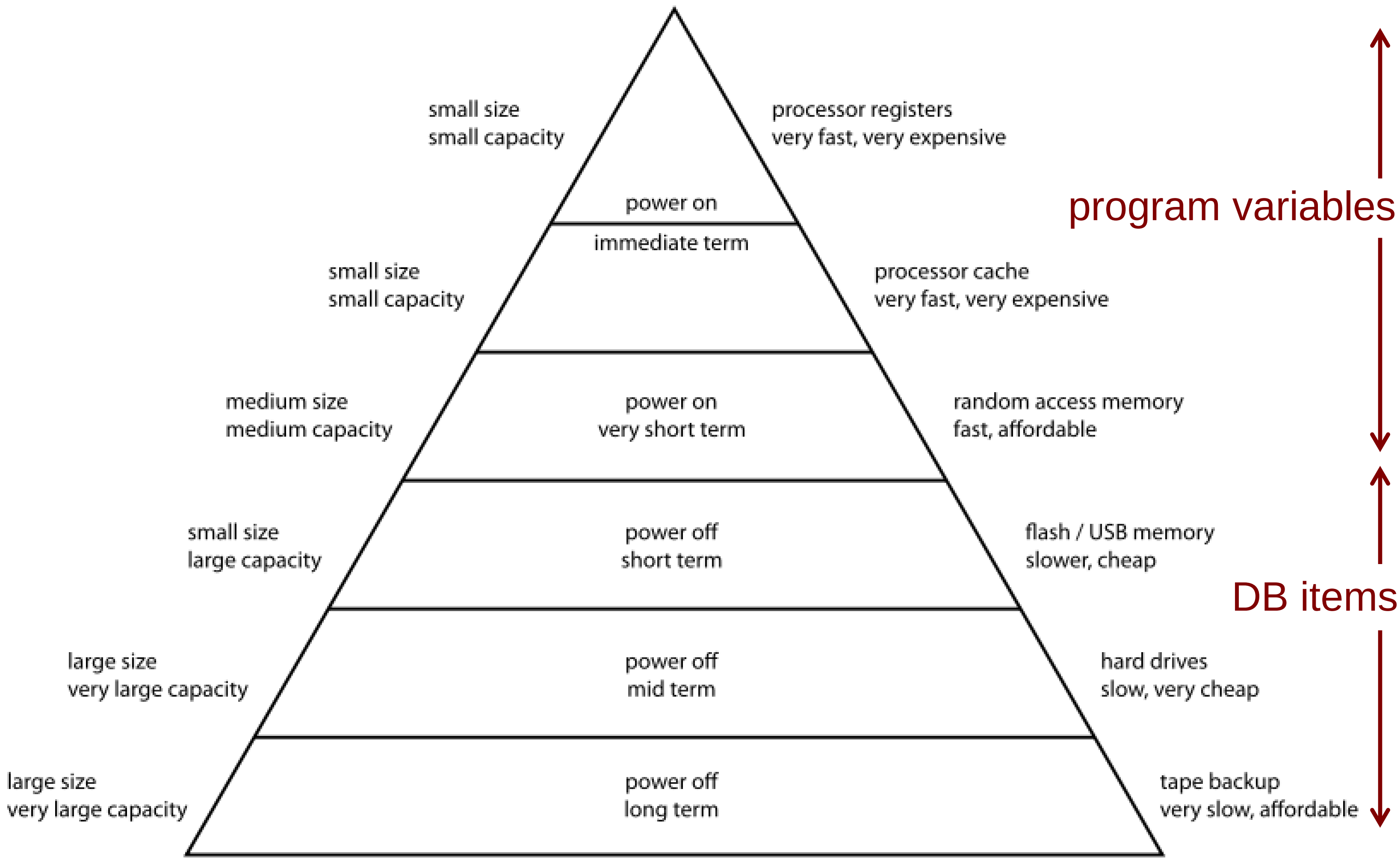
WHY WE NEED TRANSACTIONS

- A database is a shared resource accessed by many users and processes concurrently.
- Not managing this concurrent access to a shared resource will cause problems (not unlike in operating systems)
  - Problems due to concurrency
  - Problems due to failures

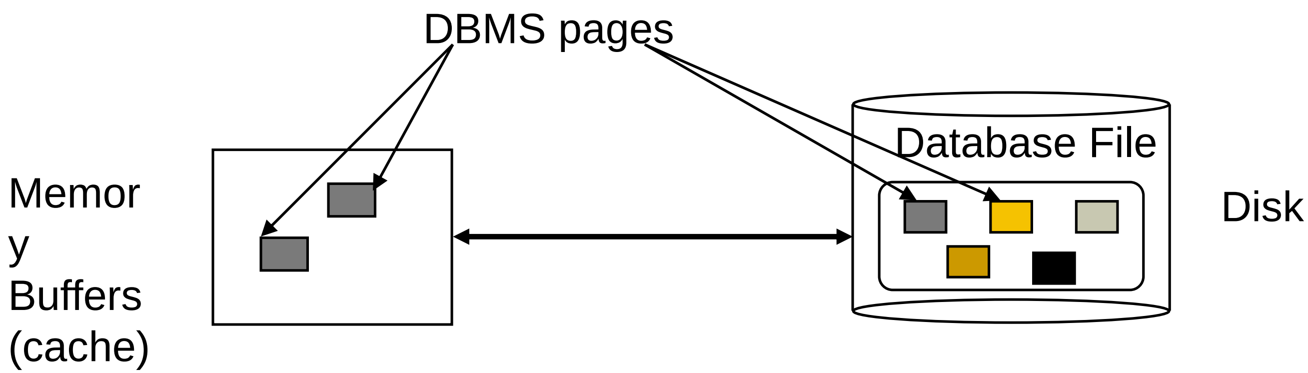
TRANSACTION PROCESSING MODEL

- Simple database model:
  - Database: collection of named data items
  - **Granularity** (size) of each data item immaterial
    - A field (data item value), a record, or a disk block
    - TP concepts are independent of granularity
- Basic operations on an item X:
  - **read_item(X)**: Reads a database item X into a program variable
    - For simplicity, assume that the program variable is also named X
  - **write_item(X)**: Writes the value of program variable X into the database item named X
- Read and write operations take some amount of time to execute

COMPUTER STORAGE HIERARCHY



READ AND WRITE OPERATIONS



- Basic unit of data transfer from the disk to the computer main memory is one disk block (or page).
- **read_item(X)** includes the following steps:
  1. Find the address of the disk block that contains item X.
  2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
  3. Copy item X from the buffer to the program variable named X.
- **write_item(X)** includes the following steps:
  1. Find the address of the disk block that contains item X.
  2. Copy that disk block into a buffer in main memory (if it is not already in some main memory buffer).
  3. Copy item X from the program variable named X into its correct location in the buffer.
  4. Store the updated block from the buffer back to disk
     - either immediately or, more typically, at some later point in time

BACK TO TRANSACTIONS

- Transaction (sequence of executing operations) may be:
  - *Stand-alone*, specified in a high level language like SQL submitted interactively, or
  - More typically, *embedded* within application program
- Transaction boundaries: *Begin_transaction* and *End_transaction*
  - Application program may include specification of several transactions separated by Begin and End transaction boundaries
  - Transaction code can be executed several times (in a loop), spawning multiple transactions
  - Transactions can end in two states:
    - Commit: transaction successfully completes and its results are committed (made permanent)
    - Abort: transaction does not complete and none of its actions are reflected in the database

TRANSACTION NOTATION

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);
	read_item(X);
	$X := X + M$;
	write_item(X);

- Focus on read and write operations
  - T_1: b_1; $r_1(X)$; $w_1(X)$; $r_1(Y)$; $w_1(Y)$; e_1;
  - T_2: b_2; $r_2(Y)$; $w_2(Y)$; e_2;
- b_i and e_i specify transaction boundaries (begin and end)
- i specifies a unique transaction identifier (Tid)
  - $w_5(Z)$ means *transaction 5 writes out the value for data item Z*

MODES OF CONCURRENCY

- Interleaved processing: concurrent execution of processes is interleaved on a single CPU
- Parallel processing: processes are concurrently executed on multiple CPUs

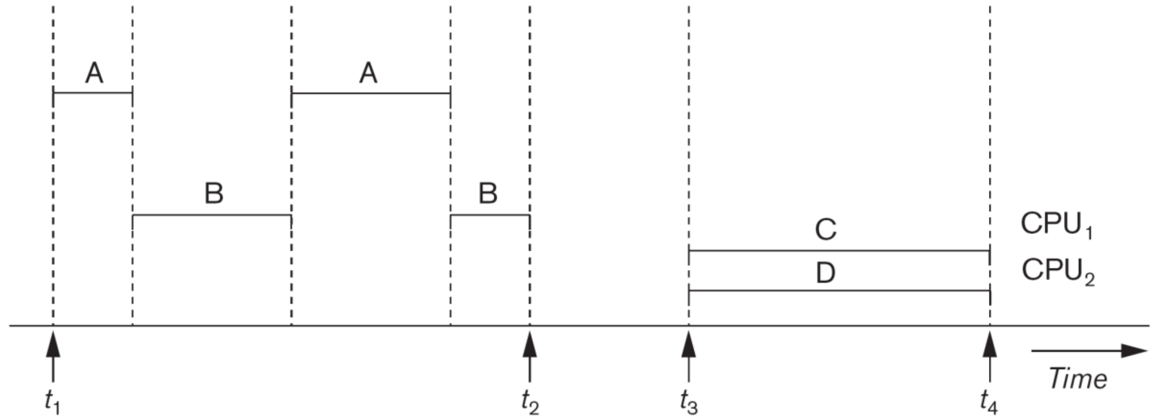


Figure 21.1
Interleaved processing versus parallel processing of concurrent transactions.

- Basic transaction processing theory assumes interleaving

WHAT CAN GO WRONG?

- Consider two concurrently executing transactions:

	at ATM window #1		at ATM window #2
1	read_item(savings);	a	read_item(chequing);
2	savings = savings - $100;	b	chequing = chequing - $20;
3	write_item(savings);	c	write_item(chequing);
4	read_item(chequing);	d	dispense $20 to customer;
5	chequing = chequing + $100;		
6	write_item(chequing);		

- System might crash after transaction begins and before it ends.
  - Money lost if between 3 and 6 or between c and d
  - Updates lost if write to disk not performed before crash
- Chequing account might have incorrect amount recorded:
  - $20 withdrawal might be lost if T2 executed between 4 and 6
  - $100 deposit might be lost if T1 executed between a and c
    - In fact, same problem if just 6 executed between a and c

ACID PROPERTIES

- **Atomicity**: A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **Consistency preservation**: A correct execution of the transaction must take the database from one consistent state to another.
- **Isolation**: Even though transactions are executing concurrently, they should appear to be executed in isolation – that is, their final effect should be as if each transaction was executed in isolation from start to finish.
- **Durability**: Once a transaction is committed, its changes (writes) applied to the database must never be lost because of subsequent failure.
- Enforcement of ACID properties:
  - Database constraint system (and application program correctness) responsible for C (introduced in previous classes)
  - Concurrency control responsible for I (more in next class)
  - Recovery system responsible for A and D (more in next class)

TRANSACTION SUPPORT IN SQL

- A single SQL statement is always considered to be atomic.
  - Either the statement completes execution without error or it fails and leaves the database unchanged.
- No explicit Begin_Transaction statement.
  - Transaction initiation implicit at first SQL statement and at next SQL statement after previous transaction terminates
- Every transaction must have an explicit end statement
  - **COMMIT**: the DB must assure that the effects are permanent
  - **ROLLBACK**: the DB must assure that the effects are as if the transaction had not yet begun

SAMPLE SQL TRANSACTION

```
update_proc() {
   EXEC SQL WHENEVER SQLERROR GO TO error;
   EXEC SQL INSERT
                INTO EMPLOYEE
                VALUES ('Robert','Smith','991004321',2,35000);
   EXEC SQL UPDATE EMPLOYEE
                SET SALARY = SALARY * 1.1
                WHERE DNO = 2;
   EXEC SQL COMMIT;
   return(0);
error:           /* continue if error on rollback */
   EXEC SQL WHENEVER SQLERROR CONTINUE;
   EXEC SQL ROLLBACK;
   return(1);
}
```

LECTURE SUMMARY

- Transaction concepts
- ACID properties for transactions
- Transaction support in SQL