Understanding Amdahl's law with paper and scissors
Consistency and atomic blocks - interleaving multiple bank account transfers and 'making money'
Divide and Conquer algorithms - e.g. merge sort or mapreduce word count

Amdahl's Law

What is it?

Amdahl's Law is a way of showing that unless a program (or part of a program) is 100% efficient at using multiple CPU cores, you will receive less and less of a benefit by adding more cores.

The speed of a program is the time it takes the program to excecute. This could be measured in any increment of time. Speedup is defined as the time it takes a program to execute in serial (with one processor) divided by the time it takes to execute in parallel (with many processors).

Let a task T being processed by only one CPU, e.g., sorting a list of number, or calculating the sum of a set of numbers. Say this task requires 2 minutes to be executed in one CPU. The conjecture is: If task T splits into T1 and T2 that can run in parallel & we have two CPUs, then the task T will require (approx.) 1 minute to be executed! The speedup (latency): the ratio of the system performance before improvement out of the system performance after improvement:

S = (2 minutes) / (1 minute) = 2 times faster!

The IBM Fellow G. Amdahl gives the theoretical bound A(s) in latency of the execution of a task that can be expected by a system given that the resources are improved (1967).

Amdahl's law is A(s) = 1 / (1 - p + (p/s))

- Where A(s) is the theoretical speedup of the whole task
- s is the speed up of part of the task that benefits from system improvement
- p is the percentage of the whole execution time that part of the task benefits from system improvement; envisage: the percentage of the task parallelizability; parallelization factor.

What concepts do we need to teach in order to teach Amdahl effectively?

- Introduce the point that by splitting up tasks to multiple cores it will increase the time taken in order to complete the task, but once efficiency reaches 100% more cores will be useless.
- Due to parallel overhead, adding more cores to an already 100% efficient system will be pointless.
- Introduce and explain Amdahl.

How do we do it?

- Perhaps by first introducing a serial program, one that goes in order one by one? Using one student to do an activity?
- Then introduce a parallel program - split the task in two - one student doing some work, while another does some other work maybe?
- Introduce more and more students, but at some point figuring out the work that could be split takes more time than doing the work with say only two people doing it.
- Suggest that Amdahl's law helps figure this out?

Using paper and scissors - unsure?

Consistency and Atomic Blocks

What is it?

Concept of ACID - Atomicity, Consistency, Isolation, Durability, is a set of properties of database transactions, where a transaction is a unit of work performed within a database management system.

Consistency in database systems refers to the requirement that any given database transaction must change affected data only in allowed ways.

Atomic blocks are segments of a transaction that are done in atomic conditions?

An example of an atomic transaction is a monetary transfer from bank account A to account B. It consists of two operations, withdrawing the money from account A and saving it to account B. Performing these operations in an atomic transaction ensures that the database remains in a consistent state, that is, money is neither lost nor created if either of those two operations fail.

What concepts do we need to teach?

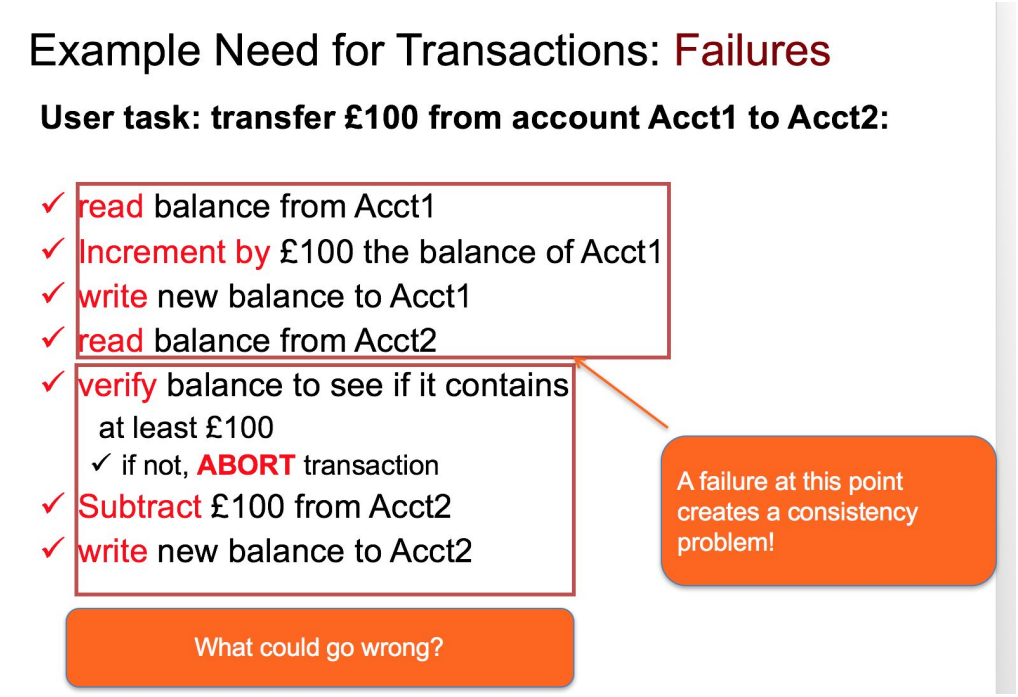
- Do students need to have an understanding of a DBMS first? The only thing they will have dealt with in school is spreadsheets rather than a fully fledged databases
- Why it's important that concepts are run consistently and in order, due to problems highlighted with the bank accounts in the example below.
- Do they need to know about integrity constraints? Or just generally understand that database transactions have to occur properly or problems will occur - then outline the problems.
- Will students need to know about:

The user of a DBMS needs mechanisms for grouping operations into transactions. Five different operations are usually provided:

- **begin transaction** - this starts a sequence of queries which will be treated atomically
- **commit** - end the transaction making all changes permanent and public.
- **abort** or **rollback** - undo all the changes of the current transaction.
- **checkpoint**, **validate** or **savepoint** *SpointName* - mark the current point as a potential point to roll back to
- **rollback** *SpointName* - roll back only as far as the check point

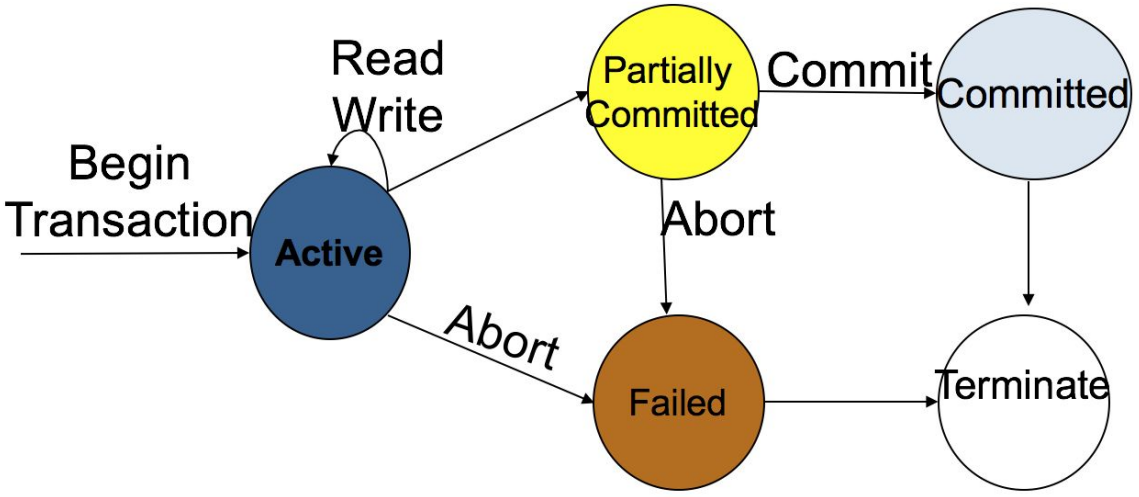
How do we teach it?

- Using cut out money and allowing a student to be a bank account and following the instructions a la this:



(DB Chp 7)

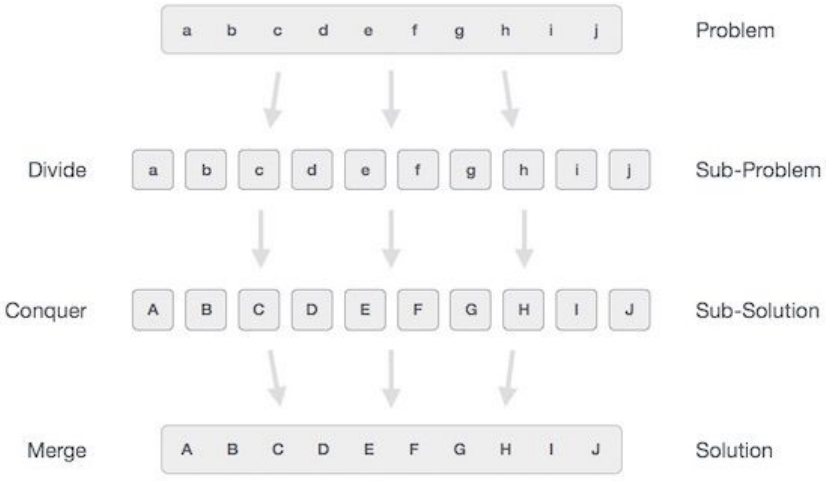
- Perhaps start off by saying that what are the problems when you run a bank account concurrently, show them that things can go wrong - or ask them - show them the failed bank account example.
- Then go on and show them that by running things properly the bank transactions can be done properly.
- Then lead on and say that this is what databases are required to do through the ACID properties.



Divide and Conquer Algorithm

What is it?

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.



Divide/Break

This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage,

sub-problems become atomic in nature but still represent some part of the actual problem.

Conquer/Solve
This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

Merge/Combine
When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.

Merge sort is a good example of a divide and conquer algorithm.

What concepts do we need to teach?

- Why divide and conquer algorithms are important - help us solve bigger problems more easily by breaking it down into smaller problems
- What is merge sort? How is sorting this way helpful - one of the fastest algorithms

How do we teach it?

- Perhaps cutting up a list of numbers and sticking them up, then following the algorithm on the board put all the numbers in order,
- Or have students interact by having them hold the numbers then getting them to stand in front of the class and swap around and separate into smaller groups in order to swap around and solve the sort - may be more confusing than just having on person up moving the numbers