Levels of transformation in a computer system design create abstractions, which improves productivity (and also ease of designing and developing systems) because we do not need to worry about decisions made in other layers. However, computer architecture is the art and science of tradeoffs, it limits the efficiency because some effort is put to transform between these abstraction layers. Even the Von Neumann model builds an abstraction layer between the processors and memories.

When we moved toward multi-core systems, we wanted to gain N times performance for N times more cores, but the nature of the problems (Amdahl’s law) and also shared resources (like DRAM chip) confliction limited us. So, cores are not the only thing to enable us to get what we want in terms of performance. A good example for this one is that when we tried to run two applications on a dual-core system we faced 1.1X, and 3.3X slow down for them, which the DRAM controller was responsible for this because of its unfair scheduling accesses to the DRAM.

**Note**: It is a general bitter fact that everywhere in the world there is a shared resource and an arbiter is going to decide how to divide it fairly it is going to have an unfairness problem.

DRAM row size (typical just for having a sense): 8KB

**#** of rows in a DRAM bank: 64K

DRAM handles refreshes internally. In other words, controller does not send a command and does not tell the controller to refresh a set of rows.

DRAM refresh for data retention causes scalability problem because as the number of cells increases with regard to technology development the energy that going to be used to retain the data is going to increase too. Also, energy consumption for refresh is a downside. Additionally refreshing DRAM periodically cause performance degradation (rows get unavailable while refreshing them), and QoS/ predictability problem because long pauses for refreshes decreases the QoS.