

# Project 2 – Main-Memory (DRAM) Access and Hybrid Manager

## Goal and Description

The objective of this project is to design and implement a **main-memory performance simulator** that models a simplified **DRAM subsystem**. The simulator should process memory-access requests generated by the cache simulator from Project 1 or by synthetic traces. It will capture key aspects of DRAM operation such as **row-buffer hits and misses**, **access queuing**, and **periodic refresh cycles**, which together influence system latency and throughput.

Students will begin by building a single-level DRAM model that reflects the behavior of a realistic memory controller. Each memory request should be mapped to a specific bank and row, and access latency must depend on whether the target row is already active (row hit) or needs to be precharged and reactivated (row miss). The simulator should also incorporate the effect of **refresh operations**, which temporarily pause servicing requests and increase average access time.

After developing the baseline DRAM model, students will extend it to include a **hybrid memory configuration** that combines DRAM with **Flash storage**. In this model, frequently accessed (“hot”) data should remain in DRAM, while less frequently used (“cold”) data is migrated to Flash. The Flash layer introduces higher access latency but offers higher density and lower power consumption, allowing students to explore the trade-offs between **speed**, **energy efficiency**, and **endurance**.

The overall purpose of this project is to help students understand how memory hierarchy design decisions affect system-level performance. Through experimentation, students will measure average latency, throughput, and endurance across multiple policies and configurations. The results will demonstrate how factors like **row-buffer locality** and **hybrid memory management** influence main-memory performance.

## Tools and Environment

This project should be implemented using **Python** or **C++**, as these languages are well-suited for algorithmic modeling and numerical computation. The **NumPy** library should be used for efficient mathematical operations and latency calculations, while **Matplotlib** (or an equivalent plotting library) should be used to visualize performance

metrics. Students may reuse cache-miss traces from Project 1 as input data, or generate synthetic workloads that mimic sequential and random access patterns.

## Deliverables

1. **Functional Source Code:** A complete simulator that models DRAM operation with configurable timing parameters, including row activation delay, column access delay, and refresh interval.
2. **Latency and Throughput Measurements:** Tables showing average access latency, throughput, and queue wait times under different workloads and configurations.
3. **Hybrid Memory Evaluation:** Comparative results between pure DRAM and hybrid DRAM + Flash systems, showing how migration and replacement policies affect latency and endurance.
4. **Row-Buffer Analysis:** Graphs illustrating row-buffer hit ratios, highlighting how access locality reduces latency.
5. **Technical Report :** A clear explanation of the simulator design, tested policies, experimental results, and key conclusions on how memory hierarchy organization impacts system performance.

## Learning Outcomes

Upon completing this project, students will:

- Gain a detailed understanding of **DRAM organization**, including banks, rows, and timing constraints.
- Learn how **row-buffer locality** and **refresh mechanisms** affect average access latency.
- Understand the performance–endurance trade-offs in **hybrid DRAM–Flash memory systems**.
- Develop practical skills in **performance modeling**, **numerical simulation**, and **data visualization**.
- Build a solid foundation for future advanced projects on **DRAM reliability**, such as RowHammer and error mitigation studies.