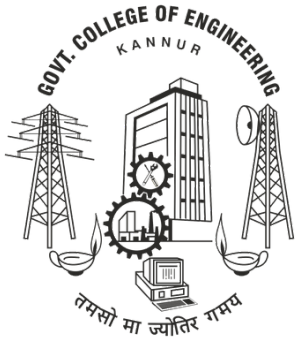
Government College of Engineering, Kannur

Kerala-670567



Project report on

CACHE REPLACEMENT POLICY PERFORMANCE

Project carried out
BY:

Group - 4

Adith Surendran	***KNR24CS006***
Adwaith K Ratheesh	***KNR24CS007***
Alan Joseph	***KNR24CS009***
Amal N K	***KNR24CS011***
Anandhu E V	***KNR24CS012***

Under the Supervision of

Professor Priya J

Assistant Professor

Department of Computer Science and Engineering

Table of Contents

Introduction

About gem5

gem5 is a powerful, open-source computer architecture simulation platform widely used in academia and industry for modeling, experimenting with, and evaluating computer systems. It enables researchers, students, and engineers to study how different hardware designs behave without requiring physical processors or prototype hardware. By providing a **flexible** and **modular** environment, gem5 allows users to construct **detailed models of CPUs, memory hierarchies, caches, interconnects, and peripheral devices**, making it possible to explore architectural ideas, analyze system performance, and evaluate design trade-offs in a controlled and repeatable manner. Because modern processors are highly complex and expensive to prototype, gem5 plays a crucial role in enabling **early-stage architectural** exploration and performance analysis before real hardware is manufactured.

The internal implementation of gem5 combines high-performance system modeling with flexible user-level configuration. The core simulation engine and performance-**critical components** are written in **C++**, which allows gem5 to efficiently model detailed microarchitectural behavior such as instruction pipelines, cache hierarchies, memory controllers, and timing interactions between system components. At the same time, gem5 uses **Python** as its **configuration** and orchestration **language.** Python scripts are used to describe the system being simulated, including the type of processor, cache sizes, memory configuration, clock frequency, and peripheral devices. This design choice provides both performance and usability: C++ ensures that the simulation engine runs efficiently, while Python enables users to easily construct, modify, and automate experiments without modifying the core simulator code. As a result, gem5 can be extended and customized to support new architectural ideas, making it a popular research platform for experimental computer architecture studies.

Users primarily interact with gem5 through the **command-line interface**, where simulations are launched using Python configuration scripts and runtime parameters. gem5 is most commonly built and used on **Ubuntu and other Linux-based operating systems**, as the development environment and build tools are well supported on Linux platforms. Although gem5 can be compiled on other Unix-like systems, Linux remains the standard and most stable environment for development and experimentation. Interaction with gem5 is typically done through terminal commands rather than graphical interfaces, reflecting its **research-oriented nature.**

While this workflow may appear complex for beginners, it provides **precise control** over experiments and makes gem5 suitable for reproducible scientific research. gem5 supports two primary modes of simulation that define how users configure and interact with the system. In **System-Call Emulation (SE) mode**, gem5 runs user-level applications without simulating a complete operating system. In this mode, system calls made by the application are handled by the host operating system, allowing faster simulation speeds and simpler setup. SE mode is commonly used for microarchitectural studies where the focus is on CPU pipelines, cache behavior, branch prediction, or memory access patterns of individual programs. In contrast, **Full-System (FS) mode** simulates an entire computer system, including the processor, memory, devices, and a full operating system such as Linux. This mode enables the simulation of realistic workloads that involve operating system behavior, device drivers, file systems, and multitasking environments. FS mode is particularly valuable for studying system-level interactions, virtualization, operating system scheduling, and the performance impact of hardware changes on complete software stacks. Together, these two modes allow gem5 to support both low-level architectural research and full-system experimentation.

One of the major strengths of gem5 is its support for multiple instruction set architectures (ISAs), including x86, ARM, and RISC-V. This makes gem5 highly **versatile** and suitable for studying a wide range of computing platforms, from desktop and server processors to mobile and embedded systems. Researchers can model existing processors or design experimental microarchitectures to evaluate how architectural changes affect performance, energy efficiency, and scalability. Compared to simple emulators such as QEMU, gem5 provides much more detailed timing and microarchitectural modeling, enabling fine-grained performance analysis. This makes gem5 particularly **suitable for research and education**, where understanding architectural behavior is more important than achieving high emulation speed. gem5 is widely used for **applications** such as **processor design research**, **cache** and **memory hierarchy optimization**, evaluation of new **architectural techniques**, **operating system research**, and teaching computer architecture concepts.

Topic

1. Introduction

Cache memory is a critical component in modern computer architecture, bridging the speed gap between the fast CPU and the slower main memory. When the cache is full, a **Cache Replacement Policy** determines which existing block must be evicted to make room for new data. This project evaluates the performance of four policies—**LRU** (Least Recently Used), **FIFO** (First-In-First-Out), **Random** and **MRU** (Most Recently Used) using the gem5 simulator.

2. Replacement Policies

Least Recently Used (LRU)

Mechanism: LRU tracks the access history of cache blocks. When eviction is necessary, it selects the block that has not been accessed for the longest period.
Rationale: Relies on the principle of temporal locality, assuming that data accessed recently is likely to be accessed again soon.
Complexity: High overhead; requires updating timestamps or position bits on every memory access.

First-In, First-Out (FIFO)

Mechanism: FIFO treats the cache as a circular queue. Blocks are evicted in the exact order they were inserted, regardless of how frequently or recently they have been accessed.
Rationale: Simplifies hardware design by decoupling eviction decisions from access patterns.
Complexity: Low overhead; requires only a counter or pointer to track the insertion order.

Random Replacement

Mechanism: The policy selects a victim block for eviction completely at random, without regarding access history or insertion order.
Rationale: Avoids "pathological cases" where structured algorithms (like LRU) consistently evict data just before it is needed (thrashing).
Complexity: Minimal overhead; requires a pseudo-random number generator but no state tracking for individual blocks.

Most Recently Used (MRU)

Mechanism: MRU evicts the block that was most recently accessed or inserted. Ideally, it keeps older data in the cache while discarding new data quickly.
Rationale: Effective for cyclic scanning workloads (e.g., repeated loops over large datasets) where the most recently used data is the least likely to be reused immediately.
Complexity: Moderate overhead; similar to LRU but with inverted logic.

3. Methodology

Simulation Environment

We utilized gem5 v25.1 in Syscall Emulation (SE) mode. The system was configured using a custom Python script to allow dynamic switching of replacement policies at runtime while keeping all hardware parameters constant.

- CPU: Single-core TimingSimpleCPU running at 1GHz.
- L1 Data Cache: 32kB size, 2-way set associative, with 4 MSHRs (Miss Status Holding Registers).
- Memory: 512MB DDR3-1600 RAM.
- Workload: A synthetic C benchmark performing sequential access on a 128kB array (4x larger than the cache) to force capacity and conflict misses.

Experimental Design

The workload was compiled with gcc -O0 -static to prevent compiler optimizations from altering memory access patterns. We ran four distinct simulations:

1. LRU: Baseline industry standard.
2. FIFO: Simplest to implement hardware-wise.
3. Random: Non-deterministic baseline.
4. MRU: Experimental policy (evicts the most recently used block).

Experimental Methodology

To evaluate the impact of cache replacement policies on system performance, we conducted a series of controlled simulations using the gem5 simulator. The experimental workflow was designed to isolate the replacement policy as the single variable while keeping all other architectural parameters constant.

The following steps were performed:

1. **Workload Compilation:**
   - The synthetic benchmark (workload.c) was compiled using gcc with the -O0 flag to disable compiler optimizations and the -static flag for compatibility with gem5's syscall emulation mode.
   - This ensured that the memory access pattern (a linear scan of a 128KB array) was preserved exactly as written in the source code.

2. **Simulation Configuration:**
   - A custom Python configuration script (run_cache.py) was developed to instantiate a single-core system with a 1GHz TimingSimpleCPU and a 32KB L1 Data Cache.
   - The script accepted a command-line argument (--policy) to dynamically switch the cache replacement policy at runtime.

3. **Execution of Experiments:**
   - Four distinct simulations were executed, one for each replacement policy: LRU, FIFO, Random, and MRU.
   - Output directories were separated (e.g., m5out/LRU, m5out/FIFO) to prevent data overwrites.

4. **Data Extraction:**
   - The stats.txt file generated for each simulation was analyzed to extract key performance metrics.
   - We utilized grep to parse specific statistics, including:
     - system.cpu.dcache.overallMissRate::total (Cache Miss Rate)
     - simSeconds (Total Execution Time)
     - system.cpu.dcache.writebacks::total (Memory Traffic)

4. Analysis of Results

The quantitative results obtained from the stats.txt files reveal distinct performance characteristics for each replacement policy.

- **Unexpected Efficiency of Random Replacement:** The Random policy achieved the lowest miss rate (12.19%) and the fastest execution time. This result is counter-intuitive, as LRU is typically expected to outperform random selection. However, the workload (a sequential loop over an array significantly larger than the cache) represents a "pathological case" for LRU. In such scenarios, LRU consistently evicts the oldest block just before it is needed again (thrashing). Random replacement, by virtue of its non-deterministic nature, statistically preserves some useful blocks that structured policies like LRU would systematically evict.

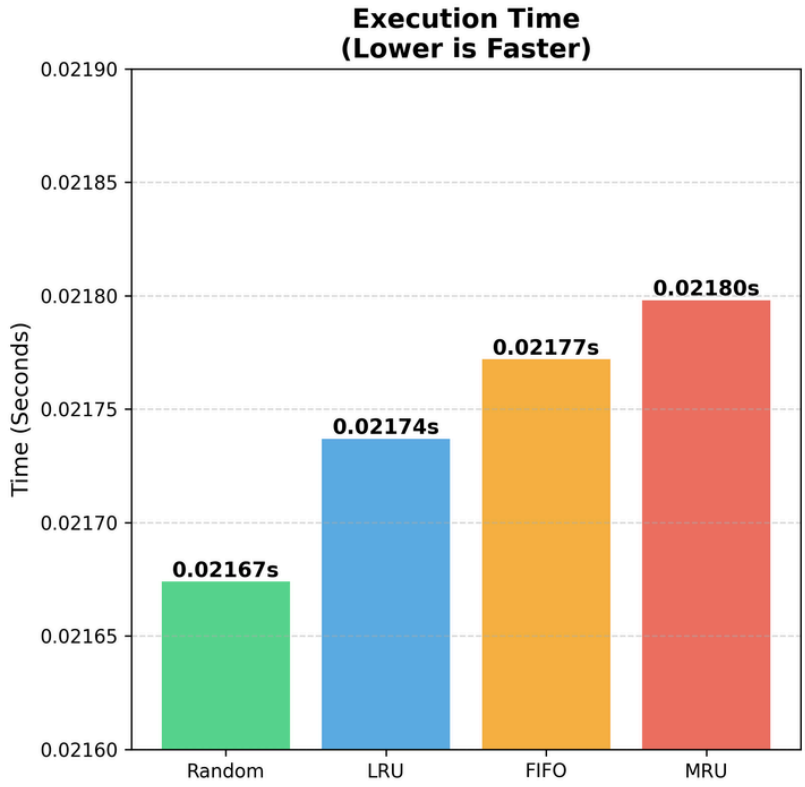
- **Trade-off Between Speed and Traffic:** While Random was faster, LRU demonstrated superior efficiency in memory traffic, generating the fewest writebacks (2,235). This indicates that while LRU incurred slightly more misses, it was more effective at retaining "dirty" blocks (data that has been written to), thereby reducing the frequency of expensive write operations to main memory.

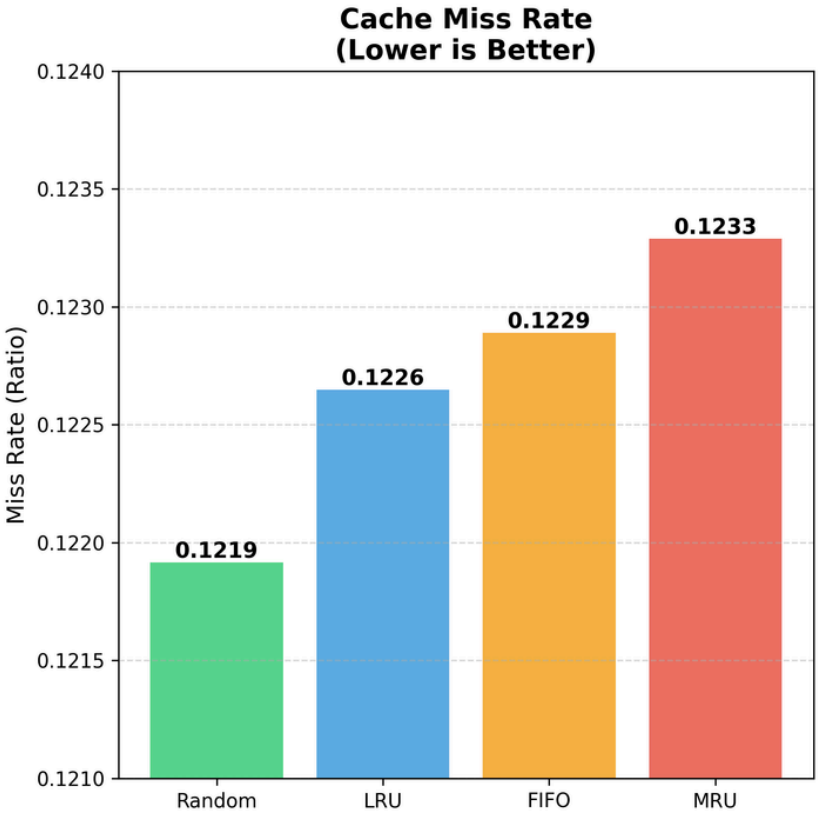
- **Poor Performance of MRU:** The MRU policy performed the worst across all metrics. By prioritizing the eviction of the most recently used data, MRU effectively discarded the exact blocks required for the immediate next iteration of the loop, preventing the cache from establishing a useful working set.

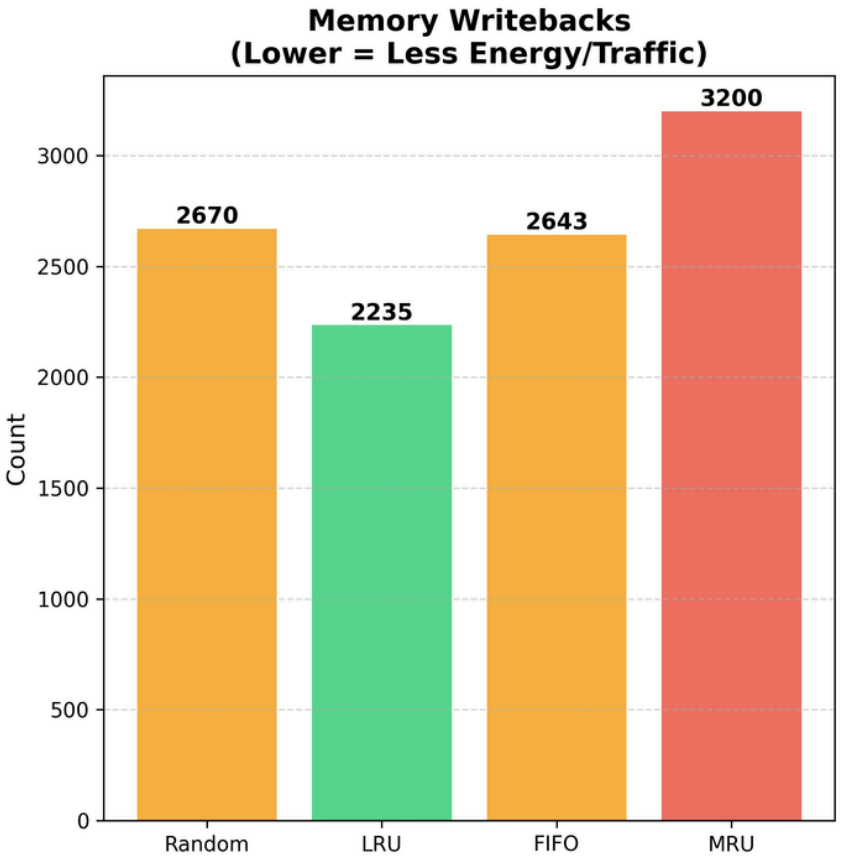
- The **Random policy** achieved the **lowest CPI (8.639)**, while **MRU** recorded the **highest (8.689).** A lower CPI indicates that the CPU spent less time "stalled" or waiting for data from main memory. Since Random resulted in fewer cache misses, the CPU could execute instructions more continuously without frequent idle periods, whereas MRU's higher miss rate forced the CPU to wait more often, driving the CPI up.

- Despite **LRU generating fewer writebacks**, Random utilized the least total memory bandwidth (~619 MB/s vs. ~620 MB/s for LRU). This reveals a crucial insight: bandwidth consumption is the sum of both reads (fetching data) and writes (saving data). Since the workload is read-heavy, Random's superior hit rate (fewer reads from memory) outweighed LRU's advantage in reducing writebacks. Consequently, Random placed the least total pressure on the system's memory bus.

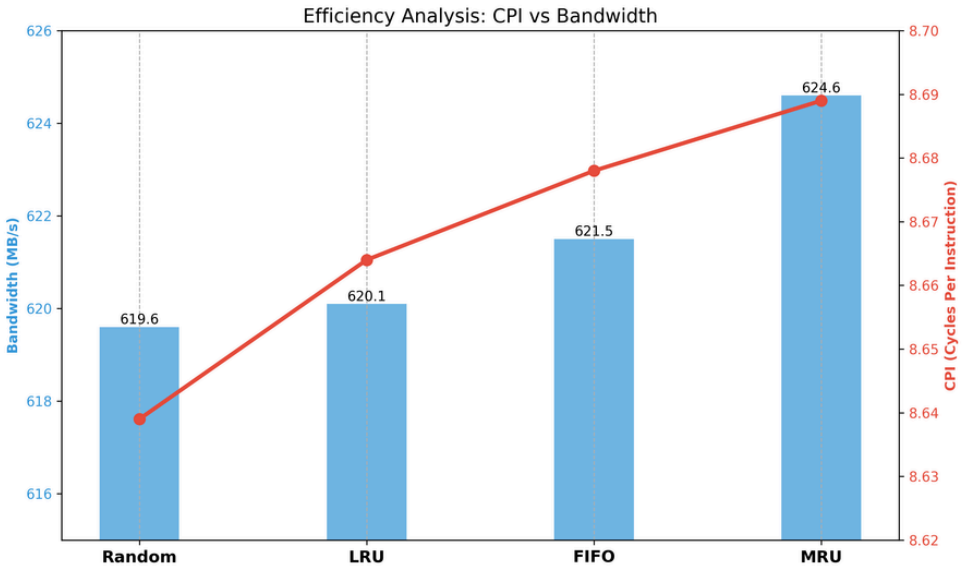
5. Bar Plots


Execution Time
(Lower is Faster)
Time (Seconds)
0.02190
0.02185
0.02180
0.02175
0.02170
0.02165
0.02160
0.02167s
0.02174s
0.02177s
0.02180s
Random
LRU
FIFO
MRU

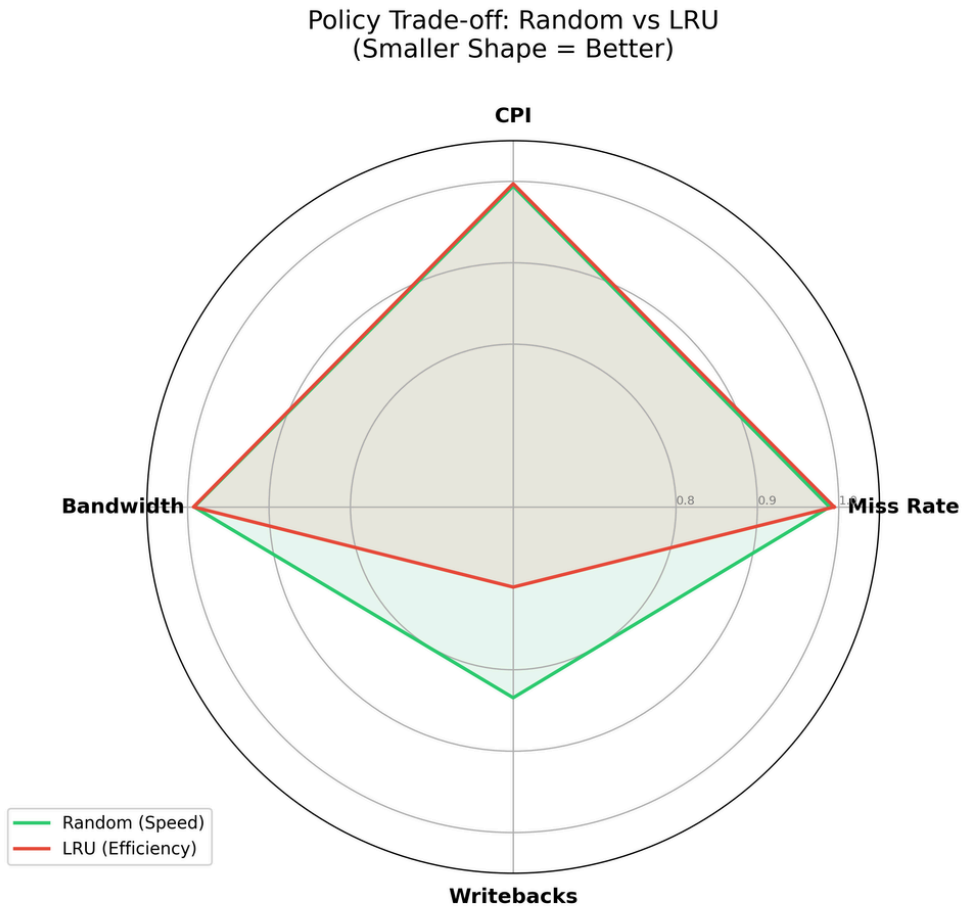

Cache Miss Rate
(Lower is Better)
Miss Rate (Ratio)
0.1240
0.1235
0.1230
0.1225
0.1220
0.1215
0.1210
0.1219
0.1226
0.1229
0.1233
Random
LRU
FIFO
MRU



6. CPI vs Bandwidth



7. Radar Plot (LRU vs Random)



Screenshots

FIFO Simulation

```
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ ~/gem5/build/X86/gem5.opt --outdir=m5out/FIFO run_cache.py --policy FIFO
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Feb 20 2026 01:20:09
gem5 started Feb 25 2026 21:39:57
gem5 executing on amal-n-k-HP-245-G7-Notebook-PC, pid 5102
command line: /home/amal-n-k/gem5/build/X86/gem5.opt --outdir=m5out/FIFO run_cache.py --policy FIFO

warn: Base 10 memory/cache size 512MB will be cast to base 2 size 512MiB.
warn: Base 10 memory/cache size 32kB will be cast to base 2 size 32KiB.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Grou
p. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation with policy: FIFO...
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.hh:1127: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/amal-n-k/Desktop/workload'
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Starting Cache Stress Test...
Done.
Exiting @ tick 21772342000 because exiting with last active thread context
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ _
```

FIFO Stats

```
  GNU nano 7.2                                        stats.txt
_
---------- Begin Simulation Statistics ----------
simSeconds                                   0.021772                       # Number of seconds simulated (Second)
simTicks                                 21772342000                        # Number of ticks simulated (Tick)
finalTick                                21772342000                        # Number of ticks from beginning of simulation (restored f
simFreq                                  1000000000000                       # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                      9.45                       # Real time elapsed on the host (Second)
hostTickRate                             2303647550                         # The number of ticks simulated per host second (ticks/s)
hostMemory                                     661204                       # Number of bytes of host memory used (Byte)
simInsts                                      2508682                       # Number of instructions simulated (Count)
simOps                                        4281458                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                   265426                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                     452991                       # Simulator op (including micro ops) rate (op/s) ((Count/S
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu.numCycles                         21772342                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                               8.678797                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                               0.115223                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                       0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                     0                       # Number of work items this cpu completed (Count)
system.cpu.issuedInstType_0::No_OpClass            332      0.01%      0.01% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntAlu            2586014     60.40%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntMult                41      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntDiv                 28      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatAdd              151      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCmp                0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCvt                0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
                                                [ Read 840 lines ]
^G Help         ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo        M-A Set Mark
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^/ Go To Line   M-E Redo        M-6 Copy
```

LRU Simulation

```
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ ~/gem5/build/X86/gem5.opt --outdir=m5out/LRU run_cache.py --policy LRU
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Feb 20 2026 01:20:09
gem5 started Feb 25 2026 21:38:22
gem5 executing on amal-n-k-HP-245-G7-Notebook-PC, pid 5017
command line: /home/amal-n-k/gem5/build/X86/gem5.opt --outdir=m5out/LRU run_cache.py --policy LRU

warn: Base 10 memory/cache size 512MB will be cast to base 2 size 512MiB.
warn: Base 10 memory/cache size 32kB will be cast to base 2 size 32KiB.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Grou
p. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation with policy: LRU...
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.hh:1127: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/amal-n-k/Desktop/workload'
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Starting Cache Stress Test...
Done.
Exiting @ tick 21737174000 because exiting with last active thread context
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ _
```

LRU Stats

```
  GNU nano 7.2                                          stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.021737                       # Number of seconds simulated (Second)
simTicks                                 21737174000                        # Number of ticks simulated (Tick)
finalTick                                21737174000                        # Number of ticks from beginning of simulation (restored f
simFreq                                1000000000000                         # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  6.19                       # Real time elapsed on the host (Second)
hostTickRate                           3513956050                         # The number of ticks simulated per host second (ticks/s)
hostMemory                                 661208                         # Number of bytes of host memory used (Byte)
simInsts                                  2508682                         # Number of instructions simulated (Count)
simOps                                    4281458                         # Number of ops (including micro ops) simulated (Count)
hostInstRate                               405526                         # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                 692090                         # Simulator op (including micro ops) rate (op/s) ((Count/S
system.clk_domain.clock                      1000                         # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage        1                         # Voltage in Volts (Volt)
system.cpu.numCycles                      21737174                         # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                            8.664779                         # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                            0.115410                         # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                   0                         # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                 0                         # Number of work items this cpu completed (Count)
system.cpu.issuedInstType_0::No_OpClass        332      0.01%      0.01% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntAlu        2586014     60.40%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntMult            41      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntDiv             28      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatAdd          151      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCmp            0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCvt            0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
                                [ Read 840 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

Random Simulation

```
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ ~/gem5/build/X86/gem5.opt --outdir=m5out/Random run_cache.py --policy Random
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Feb 20 2026 01:20:09
gem5 started Feb 25 2026 21:41:02
gem5 executing on amal-n-k-HP-245-G7-Notebook-PC, pid 5151
command line: /home/amal-n-k/gem5/build/X86/gem5.opt --outdir=m5out/Random run_cache.py --policy Random

warn: Base 10 memory/cache size 512MB will be cast to base 2 size 512MiB.
warn: Base 10 memory/cache size 32kB will be cast to base 2 size 32KiB.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Grou
p. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation with policy: Random...
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.hh:1127: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/amal-n-k/Desktop/workload'
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Starting Cache Stress Test...
Done.
Exiting @ tick 21674248000 because exiting with last active thread context
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ _
```

Random Stats

```
  GNU nano 7.2                                        stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.021674                       # Number of seconds simulated (Second)
simTicks                                 21674248000                        # Number of ticks simulated (Tick)
finalTick                                21674248000                        # Number of ticks from beginning of simulation (restored f>
simFreq                                1000000000000                         # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  9.00                       # Real time elapsed on the host (Second)
hostTickRate                           2407071650                         # The number of ticks simulated per host second (ticks/s) >
hostMemory                                661208                          # Number of bytes of host memory used (Byte)
simInsts                                 2508682                          # Number of instructions simulated (Count)
simOps                                   4281458                          # Number of ops (including micro ops) simulated (Count)
hostInstRate                              278600                          # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                475473                          # Simulator op (including micro ops) rate (op/s) ((Count/S>
system.clk_domain.clock                     1000                          # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage       1                          # Voltage in Volts (Volt)
system.cpu.numCycles                     21674248                          # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                           8.639695                          # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                           0.115745                          # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                 0                           # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted               0                           # Number of work items this cpu completed (Count)
system.cpu.issuedInstType_0::No_OpClass      332      0.01%      0.01% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntAlu      2586014     60.40%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntMult          41      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntDiv           28      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatAdd        151      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCmp          0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCvt          0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
                                                [ Read 845 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

MRU Simulation

```
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ ~/gem5/build/X86/gem5.opt --outdir=m5out/MRU run_cache.py --policy MRU
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Feb 20 2026 01:20:09
gem5 started Feb 25 2026 21:41:46
gem5 executing on amal-n-k-HP-245-G7-Notebook-PC, pid 5194
command line: /home/amal-n-k/gem5/build/X86/gem5.opt --outdir=m5out/MRU run_cache.py --policy MRU

warn: Base 10 memory/cache size 512MB will be cast to base 2 size 512MiB.
warn: Base 10 memory/cache size 32kB will be cast to base 2 size 32KiB.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Grou
p. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation with policy: MRU...
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.hh:1127: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/amal-n-k/Desktop/workload'
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Starting Cache Stress Test...
Done.
Exiting @ tick 21797740000 because exiting with last active thread context
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$
```

MRU Stats

```
 GNU nano 7.2                                        stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.021798                       # Number of seconds simulated (Second)
simTicks                                 21797740000                        # Number of ticks simulated (Tick)
finalTick                                21797740000                        # Number of ticks from beginning of simulation (restored fr
simFreq                                1000000000000                         # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  8.79                       # Real time elapsed on the host (Second)
hostTickRate                           2479513455                       # The number of ticks simulated per host second (ticks/s) >
hostMemory                                661208                       # Number of bytes of host memory used (Byte)
simInsts                                  2508682                       # Number of instructions simulated (Count)
simOps                                    4281458                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                               285358                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                 487008                       # Simulator op (including micro ops) rate (op/s) ((Count/S>
system.clk_domain.clock                      1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu.numCycles                      21797740                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                            8.688921                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                            0.115089                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                   0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                 0                       # Number of work items this cpu completed (Count)
system.cpu.issuedInstType_0::No_OpClass        332      0.01%      0.01% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntAlu        2586014     60.40%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntMult            41      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::IntDiv             28      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatAdd          151      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCmp            0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
system.cpu.issuedInstType_0::FloatCvt            0      0.00%     60.41% # Number of insts issued per FU, per thread (Count)
                                                [ Read 840 lines ]
^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo         M-A Set Mark
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo         M-6 Copy
```

Combined Outputs Using grep Command

```
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ grep -E "simSeconds|system.cpu.dcache.overallMissRate::total|system.cpu.dcache.write
backs::total" m5out/*/stats.txt
m5out/FIFO/stats.txt:simSeconds                                 0.021772                       # Number of seconds simulated (Second)
m5out/FIFO/stats.txt:system.cpu.dcache.overallMissRate::total     0.122890                       # miss rate for overall accesses (Rati
o)
m5out/FIFO/stats.txt:system.cpu.dcache.writebacks::total             2643                       # number of writebacks (Count)
m5out/LRU/stats.txt:simSeconds                                  0.021737                       # Number of seconds simulated (Second)
m5out/LRU/stats.txt:system.cpu.dcache.overallMissRate::total     0.122648                       # miss rate for overall accesses (Ratio
)
m5out/LRU/stats.txt:system.cpu.dcache.writebacks::total             2235                       # number of writebacks (Count)
m5out/MRU/stats.txt:simSeconds                                  0.021798                       # Number of seconds simulated (Second)
m5out/MRU/stats.txt:system.cpu.dcache.overallMissRate::total     0.123289                       # miss rate for overall accesses (Ratio
)
m5out/MRU/stats.txt:system.cpu.dcache.writebacks::total             3200                       # number of writebacks (Count)
m5out/Random/stats.txt:simSeconds                                 0.021674                       # Number of seconds simulated (Secon
d)
m5out/Random/stats.txt:system.cpu.dcache.overallMissRate::total     0.121915                       # miss rate for overall accesses (Ra
tio)
m5out/Random/stats.txt:system.cpu.dcache.writebacks::total             2670                       # number of writebacks (Count)
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ _
```

```
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ grep "system.cpu.dcache.overallMissRate::total" m5out/*/stats.txt
m5out/FIFO/stats.txt:system.cpu.dcache.overallMissRate::total     0.122890                       # miss rate for overall accesses (Rati
o)
m5out/LRU/stats.txt:system.cpu.dcache.overallMissRate::total     0.122648                       # miss rate for overall accesses (Ratio
)
m5out/MRU/stats.txt:system.cpu.dcache.overallMissRate::total     0.123289                       # miss rate for overall accesses (Ratio
)
m5out/Random/stats.txt:system.cpu.dcache.overallMissRate::total     0.121915                       # miss rate for overall accesses (Ra
tio)
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ _
```

```
tio)
amal-n-k@amal-n-k-HP-245-G7-Notebook-PC:~/Desktop$ grep -E "system.cpu.cpi|system.mem_ctrl.dram.bwTotal::total" m5out/*/stats.txt
m5out/FIFO/stats.txt:system.cpu.cpi                               8.678797                       # CPI: cycles per instruction (core le
vel) ((Cycle/Count))
m5out/FIFO/stats.txt:system.mem_ctrl.dram.bwTotal::total        621544527                       # Total bandwidth to/from this memory
((Byte/Second))
m5out/LRU/stats.txt:system.cpu.cpi                                8.664779                       # CPI: cycles per instruction (core lev
el) ((Cycle/Count))
m5out/LRU/stats.txt:system.mem_ctrl.dram.bwTotal::total         620129921                       # Total bandwidth to/from this memory (
(Byte/Second))
m5out/MRU/stats.txt:system.cpu.cpi                                8.688921                       # CPI: cycles per instruction (core lev
el) ((Cycle/Count))
m5out/MRU/stats.txt:system.mem_ctrl.dram.bwTotal::total         624563831                       # Total bandwidth to/from this memory (
(Byte/Second))
m5out/Random/stats.txt:system.cpu.cpi                             8.639695                       # CPI: cycles per instruction (core
level) ((Cycle/Count))
m5out/Random/stats.txt:system.mem_ctrl.dram.bwTotal::total        619576928                       # Total bandwidth to/from this memor
y ((Byte/Second))
```

Programs

Workload Program

workload.c

```c
#include <stdio.h>
#include <stdlib.h>

// Array size: 128KB (Assuming we simulate a 32KB Cache, this guarantees conflict misses)
#define ARRAY_SIZE (32 * 1024)
#define NUM_LOOPS 100

int main() {
    printf("Starting Cache Stress Test...\n");

    // Allocate memory
    int *arr = (int*)malloc(ARRAY_SIZE * sizeof(int));
    if (arr == NULL) return 1;

    // Initialize (Cold Misses)
    for (int i = 0; i < ARRAY_SIZE; i++) {
        arr[i] = i;
    }

    // Access Pattern Loop
    // We stride by 16 (64 bytes) to hit a new cache line every access
    volatile int sink;
    for (int k = 0; k < NUM_LOOPS; k++) {
        for (int i = 0; i < ARRAY_SIZE; i += 16) {
            sink = arr[i]; // Read access
        }
    }

    printf("Done.\n");
    return 0;
}
```

Python Script

run_cache.py 9+

run_cache.py > ...

```python
import m5
from m5.objects import *
import argparse

# 1. Parse Command Line Arguments for Policy
parser = argparse.ArgumentParser()
parser.add_argument('--policy', type=str, default='LRU',
                    choices=['LRU', 'FIFO', 'Random', 'MRU'],
                    help='Replacement policy to use')
args = parser.parse_args()

# 2. Define the System
system = System()
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()

system.mem_mode = 'timing' # Required for gathering cache stats
system.mem_ranges = [AddrRange('512MB')] # Small memory for low-end laptop

# 3. CPU Setup (TimingSimple is faster than O3)
system.cpu = TimingSimpleCPU()

# 4. Cache Configuration
# We create a simple L1 Data Cache.
# We ignore L1 Instruction Cache for simplicity (connect it to membus directly)
class MyL1Cache(Cache):
    assoc = 2               # 2-way set associative
    tag_latency = 2
    data_latency = 2
    response_latency = 2
    mshrs = 4
    tgts_per_mshr = 20
    size = '32kB'           # Small cache to force misses

    # DYNAMIC POLICY SWITCHING HERE
    def __init__(self, policy_name):
        super().__init__()
        if policy_name == 'LRU':
            self.replacement_policy = LRURP()
        elif policy_name == 'FIFO':
            self.replacement_policy = FIFORP()
        elif policy_name == 'Random':
            self.replacement_policy = RandomRP()
        elif policy_name == 'MRU':
            self.replacement_policy = MRURP() # Extra credit policy!

system.cpu.icache = MyL1Cache(args.policy)
system.cpu.dcache = MyL1Cache(args.policy)

# Connect Caches to CPU
system.cpu.icache.cpu_side = system.cpu.icache_port
system.cpu.dcache.cpu_side = system.cpu.dcache_port

# 5. Interconnect (Memory Bus)
system.membus = SystemXBar()

# Connect caches to memory bus
system.cpu.icache.mem_side = system.membus.cpu_side_ports
system.cpu.dcache.mem_side = system.membus.cpu_side_ports

# 6. Interrupt Controller (Required for x86)
system.cpu.createInterruptController()
system.cpu.interrupts[0].pio = system.membus.mem_side_ports
system.cpu.interrupts[0].int_requestor = system.membus.cpu_side_ports
system.cpu.interrupts[0].int_responder = system.membus.mem_side_ports

```

```
67
68  # 7. Memory Controller
69  system.mem_ctrl = MemCtrl()
70  system.mem_ctrl.dram = DDR3_1600_8x8()
71  system.mem_ctrl.dram.range = system.mem_ranges[0]
72  system.mem_ctrl.port = system.membus.mem_side_ports
73
74  # 8. Set the Workload (Binary to run)
75  binary = 'workload' # Path to your compiled C binary
76  system.workload = SEWorkload.init_compatible(binary)
77
78  process = Process()
79  process.cmd = [binary]
80  system.cpu.workload = process
81  system.cpu.createThreads()
82
83  # 9. Instantiate and Run
84  root = Root(full_system=False, system=system)
85  m5.instantiate()
86
87  print(f"Beginning simulation with policy: {args.policy}...")
88  exit_event = m5.simulate()
89  print(f"Exiting @ tick {m5.curTick()} because {exit_event.getCause()}")
```

Conclusion

This project successfully evaluated LRU, FIFO, Random, and MRU cache replacement policies using the gem5 simulator. By subjecting a 32KB L1 Data Cache to a workload larger than its capacity (128KB), we uncovered a significant divergence between theoretical complexity and real-world performance.

Key Findings:

- Simplicity Wins on Speed (CPI & Miss Rate): Contrary to the expectation that "smarter" algorithms perform better, the simple Random policy achieved the lowest miss rate (12.19%) and the lowest CPI (8.639). This indicates that Random stalled the CPU the least, effectively breaking the synchronized "thrashing" cycles that plagued deterministic algorithms like LRU.

- The Traffic Paradox (Bandwidth vs. Writebacks): A nuanced trade-off emerged regarding memory traffic. While LRU was superior at retaining "dirty" data (generating the fewest writebacks: 2,235), Random actually consumed the least total memory bandwidth (~619 MB/s). Because the workload was read-heavy, Random's ability to minimize read fetches outweighed LRU's ability to minimize writebacks.

- The Failure of MRU: The MRU policy performed consistently worse across all metrics (Highest CPI: 8.689). By prioritizing the eviction of the most recently used data, it effectively discarded the working set immediately after use, proving unsuitable for sequential scanning workloads.

Final Verdict:

There is no single "best" replacement policy; the choice depends on the specific optimization goal:

- For Raw Speed & Total Bandwidth: Random replacement is superior when the working set significantly exceeds cache capacity.

- For Write Efficiency: LRU remains the standard for minimizing costly write-back operations, which is critical for battery-constrained systems.

Future work could explore hybrid policies (like Segmented LRU) that attempt to combine the scan-resistance of Random with the write-efficiency of LRU.