These tests cover the three main behaviors of a cache: **Reuse (Locality)**, **Eviction (Capacity)**, and **Hierarchy (L2 Rescue)**.

Test Scenario 1: Basic Temporal Locality (The "Happy Path")

Objective: Verify that the simulator correctly identifies data that is already present in the cache (Hit) and does not fetch it from memory again.

1. Configuration (In CacheHierarchy.cpp):

- **L1 Cache:** Size = 4096 (4KB), Block Size = 64, Associativity = 2
- **L2 Cache:** Size = 65536 (64KB), Block Size = 64, Associativity = 4
  (Standard configuration where L1 is large enough to hold a few items)

2. Input Trace (In main.cpp):

C++
```
std::vector<unsigned long> memoryTrace = {
   0x1000, // Access A
   0x2000, // Access B
   0x1000, // Access A (Again)
   0x2000, // Access B (Again)
   0x1000  // Access A (Again)
};
```

3. Expected Behavior:

- **Step 1 & 2:** Cold Misses. 0x1000 and 0x2000 are fetched from memory and stored in L1.
- **Step 3, 4, & 5: L1 Hits**. The simulator should find these addresses residing in L1.
- **L2 Activity:** L2 should have 0 Hits because L1 never missed after the initial load.

4. Expected Output Statistics:

- **L1 Hits:** 3
- **L1 Misses:** 2
- **L2 Hits:** 0
- **L2 Misses:** 2

Test Scenario 2: The "Tiny L1" Eviction (Hierarchy Validation)

Objective: Verify that when L1 runs out of space, it evicts the oldest item (FIFO) to L2, and subsequent requests for that item are serviced by L2 (L1 Miss -> L2 Hit).

1. Configuration (In CacheHierarchy.cpp):

- L1 Cache: Size = 128 (Bytes), Block Size = 64, Associativity = 2
  (CRITICAL: This limits L1 to exactly 2 cache lines)
- **L2 Cache:** Size = 4096 (4KB), Block Size = 64, Associativity = 4

2. Input Trace (In main.cpp):

C++
```
std::vector<unsigned long> memoryTrace = {
    0x1000, // 1. Fill Slot 1
    0x2000, // 2. Fill Slot 2 (L1 is now FULL)
    0x3000, // 3. New Data. Forces Eviction of 0x1000 (Oldest).
    0x1000  // 4. Retry 0x1000. It is gone from L1, must be found in L2.
};
```

3. Expected Behavior:

- **Step 3:** Causes a "Capacity Miss". L1 drops 0x1000 to make room for 0x3000.
- **Step 4:** The request for 0x1000 fails in L1 (**Miss**) but succeeds in L2 (**Hit**), proving the hierarchy works correctly.

4. Expected Output Statistics:

- **L1 Hits:** 0
- **L1 Misses:** 4
- **L2 Hits:** 1 (The rescue of 0x1000)
- **L2 Misses:** 3 (The initial cold loads)

Test Scenario 3: Worst-Case "Thrashing" (Conflict Misses)

Objective: Demonstrate the weakness of small associativity. If we alternate between 3 addresses that map to the same set in a 2-way cache, they will constantly kick each other out.

1. Configuration (In CacheHierarchy.cpp):

- L1 Cache: Size = 128 (Bytes), Block Size = 64, Associativity = 2
  (Capacity is 2 lines)
- **L2 Cache:** Size = 4096 (4KB), Block Size = 64, Associativity = 4

2. Input Trace (In main.cpp):

C++
```
std::vector<unsigned long> memoryTrace = {
    0x1000, // Load A
    0x2000, // Load B (L1 Full: [A, B])
    0x3000, // Load C (Evict A. L1: [B, C])
    0x1000, // Load A (Evict B. L1: [C, A])
    0x2000, // Load B (Evict C. L1: [A, B])
    0x3000  // Load C (Evict A. L1: [B, C])
};
```

3. Expected Behavior:

- Every single step triggers an eviction.
- The cache is "Thrashing"—spending all its time loading and evicting without ever getting a hit in L1.
- However, L2 (which is larger/4-way) should catch all the re-accesses.

4. Expected Output Statistics:

- **L1 Hits:** 0 (0% Hit Rate)
- **L1 Misses:** 6
- **L2 Hits:** 3 (The repeats of A, B, and C)
- **L2 Misses:** 3 (The first time A, B, and C were seen)