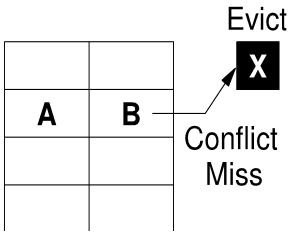
Lab 5: Identify Conflicting Address for a Cache Attack (2pts)

Cache attacks try to infer the access pattern by causing cache evictions. A popular form of cache attack is the **Prime+Probe** attack, where the attacker primes the cache with the set containing the Probe Address with the lines of the attacker.

When the Probe Address gets accessed, the cache will evict one of the lines used for priming the cache set and the attacker will "probe" the cache to learn that the particular cache set was accessed. For example, in the example beside, for a 2-way set associative cache, if the Probe Address is "X", the attacker needs to identify two lines (A and B) that maps to the same set as X.



For a cache that uses the bottom few bits of the line address for indexing the cache, finding the conflicting lines is easy. However, robust cache designs tend to use a complex hash (randomized) cache indexing, which makes it much harder to identify the conflicting address. Your job is to use cache hit/miss information to identify conflicting addresses. For an N-way cache, you will need to identify N conflicting lines, so that the pattern would be able to evict X (based on the replacement policy this may not happen 100% of the time).

Your assignment has the following constraints:

1. The cache contains 1024 sets
2. The index function is random (assume cache implementation is not visible to you).
3. Cache always installs on an access and uses random replacement for 2/4 way
4. The only function available to you is **`cache_access_install(addr)`** – which returns a HIT or MISS, depending on whether addr is present in the cache.
5. All addresses are line addresses (you do not have to worry about offset bits)
6. All line address MUST be less than 2^20 (approx. 1 million)
7. For N-way cache you must populate exactly N entries in **`conflict_list`**
8. We will test your design for 1-way (1 point) and 2-way (1 point). **You may attempt the 4-way design for 3 points of extra-credit (must run within 30 seconds)**
9. All simulations must end in less than 30 seconds on ecelinsrv
10. You are allowed to change only **conflict.cc**. For submission, you will upload only conflict.cc to T-square (so your changes to other files will not be visible to us).
11. We must be able to run your solution on the reference machine
12. We recommend you test your solutions for multiple cases (see run.sh for examples)

Download and run: Download the tarball from here **(updated: 11/19, 9:25 pm)** and do:

1. tar –xvzf lab5.tar.gz
2. cd lab5
3. type make
4. ./sim –h for help (see ./run.sh for information on how to run the simulator)
5. The simulator will print "OUTCOME: Failure" without implementing conflict.cc
6. Edit conflict.cc with your solution and run the simulator again

Submission instructions:

1. Test your solution on the reference machine to check if you meet the required time limit (for all cases – 1/2/4 way in case of extra credit).
2. Upload conflict.cc to T-square

FAQ (this is a new lab, so check here for updates):