

The background of the slide is a white canvas covered with a pattern of teal-colored dots. The dots are of two shades, a darker teal and a lighter, minty green. They are arranged in a way that creates a sense of depth and movement, with some areas being more densely packed than others, particularly on the right side of the slide.

CS 449 REC

Cache lab

8-way set	Access order
A	7
B	3
C	2
D	6
E	5
F	1
G	4
H	0

LRU (eviction policy) quiz

- LRU cache scheme is actually a page replacement algorithm
- Access sequence: A -> B -> A -> D -> K

Quiz from Georgia Tech HPCA

8-way set	Access order	Current order
A	7	
B	3	
C	2	
D	6	
E	5	
F	1	
G	4	
H	0	

After A: hit or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	7	7
B	3	3
C	2	2
D	6	6
E	5	5
F	1	1
G	4	4
H	0	0

After A: **hit** or miss?

- Access sequence: **A** -> B -> A -> D -> K

8-way set	Previous order	Current order
A	7	
B	3	
C	2	
D	6	
E	5	
F	1	
G	4	
H	0	

After B: hit or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	7	6
B	3	7
C	2	2
D	6	5
E	5	4
F	1	1
G	4	3
H	0	0

After B: **hit** or miss?

- Access sequence: **A** -> **B** -> A -> D -> K

8-way set	Previous order	Current order
A	6	
B	7	
C	2	
D	5	
E	4	
F	1	
G	3	
H	0	

After A: hit or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	6	7
B	7	6
C	2	2
D	5	5
E	4	4
F	1	1
G	3	3
H	0	0

After A: **hit** or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	7	
B	6	
C	2	
D	5	
E	4	
F	1	
G	3	
H	0	

After D: hit or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	7	6
B	6	5
C	2	2
D	5	7
E	4	4
F	1	1
G	3	3
H	0	0

After D: **hit** or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	6	
B	5	
C	2	
D	7	
E	4	
F	1	
G	3	
H	0	

After K: hit or miss?

- Access sequence: A -> B -> A -> D -> K

8-way set	Previous order	Current order
A	6	5
B	5	4
C	2	1
D	7	6
E	4	3
F	1	0
G	3	2
H K	0	7

After K: hit or miss?

- Access sequence: A -> B -> A -> D -> K

Next to be evicted

Overview

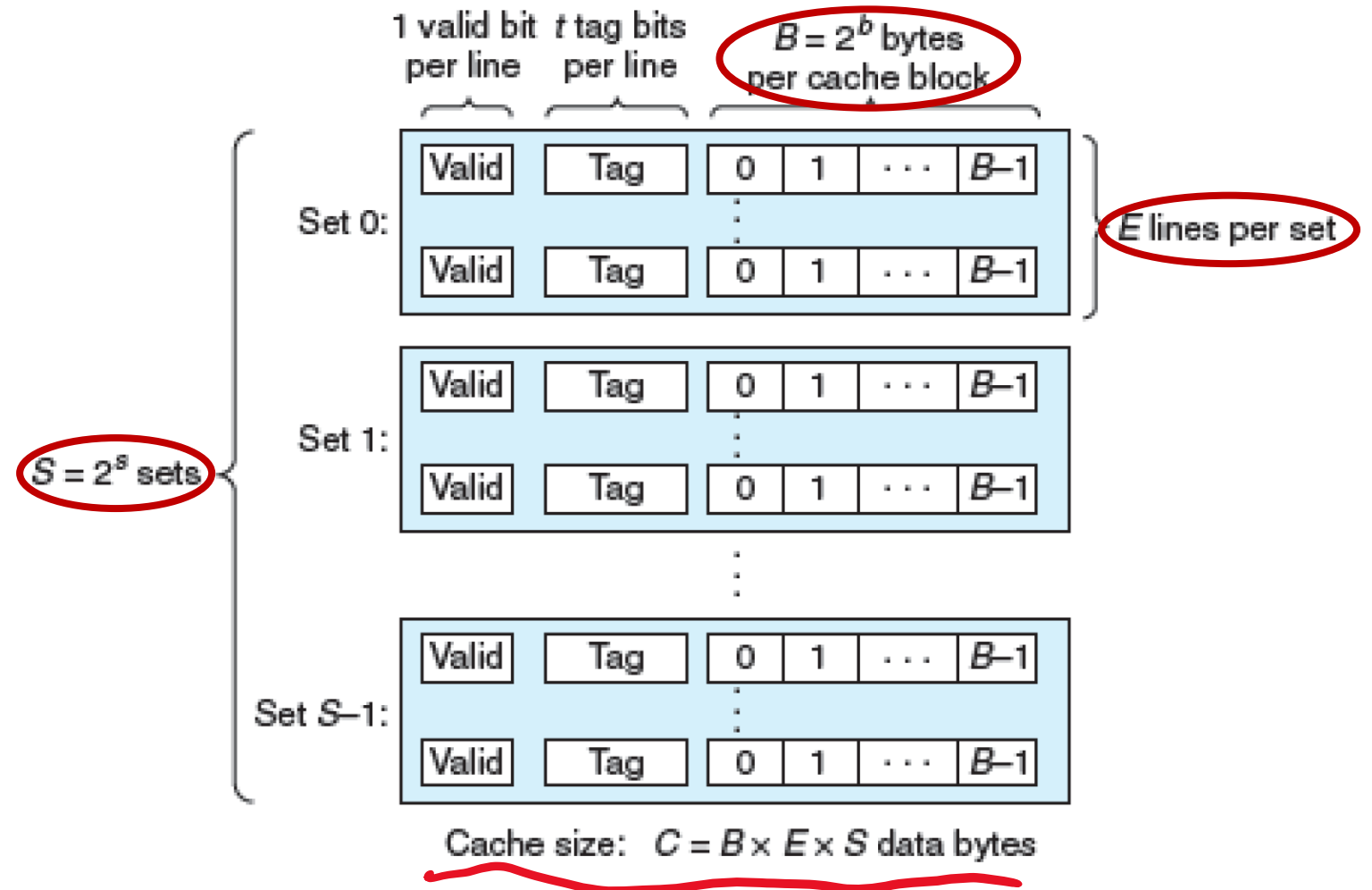
- Write a set associative cache simulator (starter code: csim.c)
- The trace file is generated by valgrind
- Your program should take the same command line arguments and produce identical output to the reference simulator (csim-ref)
- **BONUS:** write a C program that when linked with one of these cache object files, will determine and then output the cache size, associativity, and block size (starter code: cache-test-skel.c)

Usage (for your program & csim-ref)

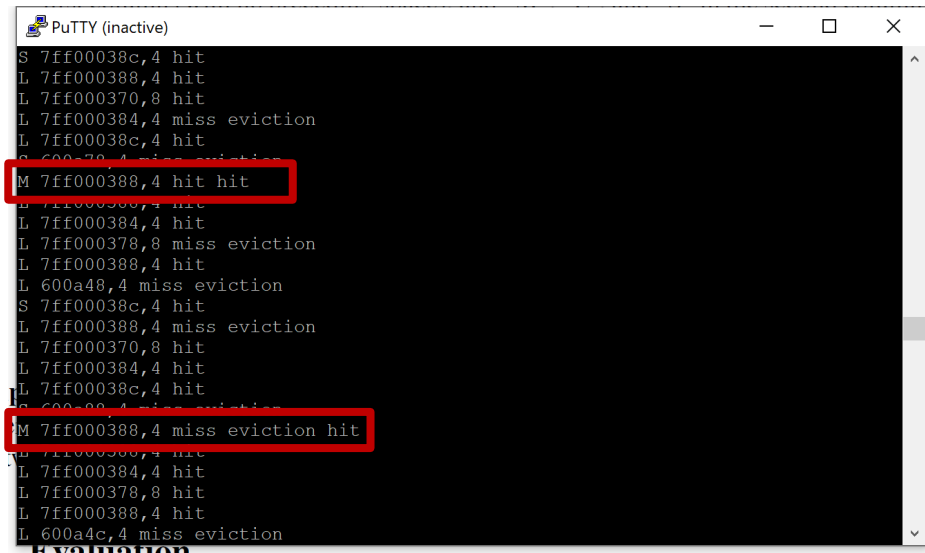
- -h: Optional help flag that prints usage info
- -v: Optional verbose flag that displays trace info
- -s <s>: Number of set index bits ($S = 2^s$ is the number of sets)
- -E <E>: Associativity (number of lines per set)
- -b : Number of block bits ($B = 2^b$ is the block size)
- -t <tracefile>: Name of the valgrind trace to replay

cache organization

- A cache is an array of sets
- The set index bits (as an unsigned int) tell us which set the word must be stored in
- The t tag bits in the address tell us which line (if any) in the set contains the word
 - A line in the set contains the word if and only if the valid bit is set and the tag bits in the line match the tag bits in the address A
- the b block offset bits give us the offset of the word in the B -byte data block



Implementation and Assumption 1



```
PutTY (inactive)
S 7ff00038c,4 hit
L 7ff000388,4 hit
L 7ff000370,8 hit
L 7ff000384,4 miss eviction
L 7ff00038c,4 hit
M 7ff000388,4 hit hit
L 7ff000384,4 hit
L 7ff000378,8 miss eviction
L 7ff000388,4 hit
L 600a48,4 miss eviction
S 7ff00038c,4 hit
L 7ff000388,4 miss eviction
L 7ff000370,8 hit
L 7ff000384,4 hit
L 7ff00038c,4 hit
M 7ff000388,4 miss eviction hit
L 7ff000384,4 hit
L 7ff000378,8 hit
L 7ff000388,4 hit
L 600a4c,4 miss eviction
```

- Each load/store can cause at most one cache miss (for all block sizes). The data modify operation (M) is treated as a load followed by a store to the same address.
- an M operation can result in two cache hits, or a miss and a hit plus a possible eviction (even when only 1 byte is modified).

Implementation and Assumption 2

```
/* Globals set by command line args */
int verbosity = 0; /* print trace if set */
int s = 0; /* set index bits */
int b = 0; /* block offset bits */
int E = 0; /* associativity */
char* trace_file = NULL;

/* Derived from command line args */
int S; /* number of sets */
int B; /* block size (bytes) */
```

- Your simulator must work correctly for arbitrary s , E , and b .
 - you will need to dynamically allocate storage for your simulator's data structures using the malloc (and free!) function.

Implementation and Assumption 3

- Instruction loads (I) are ignored, since we are interested in evaluating data cache performance.
 - You should parse the traces (what's the format?), ignoring the lines starting with "I" without preceding space
 - E.g.
 - I 0400d7d4,8
 - M 0421c7f0,4
 - L 04f6b868,8
 - S 7ff0005c8,8

Implementation and Assumption 4

- For this lab, you should assume that memory accesses are aligned properly, such that a single memory access never crosses block boundaries (when $2^b < \text{size}$)
 - you can ignore the request sizes in the valgrind traces

Implementation and Assumption 5

- Make sure to use `printSummary()`

Valgrind

- Covered in the early recitations
- **valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l**
 - --log-fd=1: tells valgrind to write to stdout
 - --tool=lackey: to use the Lackey tool (<https://www.valgrind.org/docs/manual/lk-manual.html>)
 - -v: verbose, display lots of info
 - --trace-mem=yes: one of the Lackey spec (see manual), to print the size & address of every memory access
 - ls -l: run valgrind on ls (list all) instruction with flag l (list the directory contents with detailed info)
- You don't need to try this line, since it's a very long output. For the homework you are provided with 4 trace files

What to do & how to do it

- Command line arguments & cache set up
- Read L/S/M from trace files
- Split the memory address (|tag|set index|block offset|)
- Use the “set” portion to select the correct set **(data structure)**
 - **Linked lists, arrays (how do you keep track of LRU?), ...**
- Use the “tag” portion to search for a line **(data structure)**
 - Set -> line -> tag
 - **Linked lists, arrays (how do you keep track of LRU?), ...**
- **Or combination of: queue/linked list, hashmap, 2d array (how do you keep track of LRU?), ...**

Notes

- Refer to previous labs if you have trouble freeing data structure (e.g. queue lab)
- You will need to consider how the cache looks like for all L/M/S operations with LRU (refer to last recitation cache tracing)
- `Atoi()`: convert string to int
- `sscanf`: to format string `%c %llx, %u`

Extra credit

- You will complete the starter code `cache-test-skel.c` such that when linked with one of these cache object files, will determine and then output the cache size, associativity, and block size.
- You will use the functions on the next page to perform cache accesses and use your observations of which ones hit and miss to determine the parameters of the caches.
- You can assume that the mystery caches have sizes that are powers of 2 and use a least recently used replacement policy.
- You cannot assume anything else about the cache parameters except what you can infer from the cache size.



FUNCTIONS YOU CAN USE

```
/** Initializes the cache. This function must be called so that the
    cache can initialize its data structures, though the mystery
    caches will ignore the provided arguments (as their parameters are
    hard-coded). */
void cache_init(int size, int block_size);

/** Lookup an address in the cache. Returns TRUE if the access hits,
    FALSE if it misses. */
bool_t access_cache(addr_t address);

/** Clears all words in the cache (and the victim buffer, if
    present). Useful for helping you reason about the cache
    transitions, by starting from a known state. */
void flush_cache(void);
```

```
echo
echo "Provided Cache TESTS: "
echo
make cache-test TEST_CACHE=caches/cache_1c_1e_1k.o
./cache-test
echo
rm cache-test
make cache-test TEST_CACHE=caches/cache_65536c_2e_16k.o
./cache-test
echo
rm cache-test
make cache-test TEST_CACHE=caches/cache_1048576c_256e_256k.o
./cache-test
echo
rm cache-test
make cache-test TEST_CACHE=caches/cache_16384c_4e_4k.o
./cache-test
echo
rm cache-test
make cache-test TEST_CACHE=caches/cache_32768c_8e_8k.o
./cache-test
echo
echo "Finished."
echo
rm cache-test
```

Evaluation

- “The test will make and run cache-test with all the cache object files”



START EARLY!