Lab7

Cache (Part A)

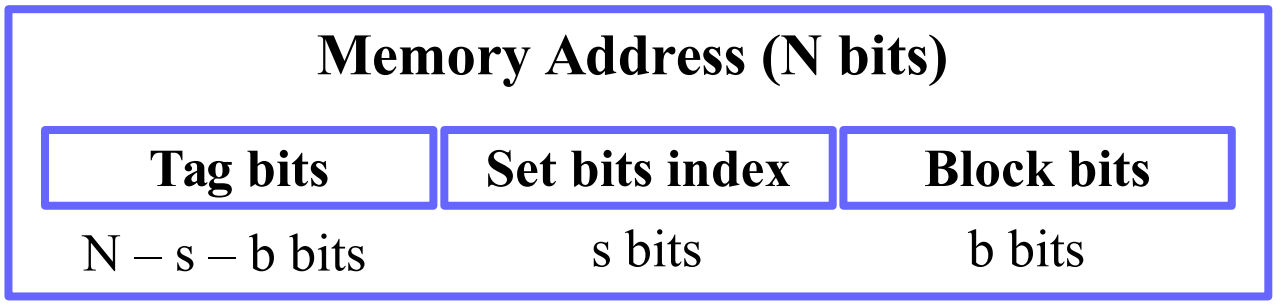
- Writing Cache Simulator -

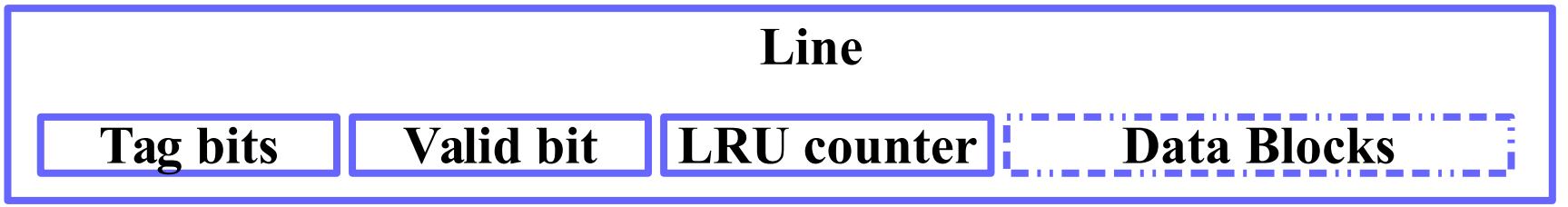
In this lab...

- Cache memory revisited
  - Basic building blocks to write cache simulator
    - Notations, Address(tag bit, set index bit), Cache memory(set, line, ...), LRU
    - Calculate hit/miss/eviction count
- You will learn how to write a simple command line based application
  - Specifying Input / Output
  - File I/O APIs
  - Dynamic memory allocation & de-allocation
  - Option parsing

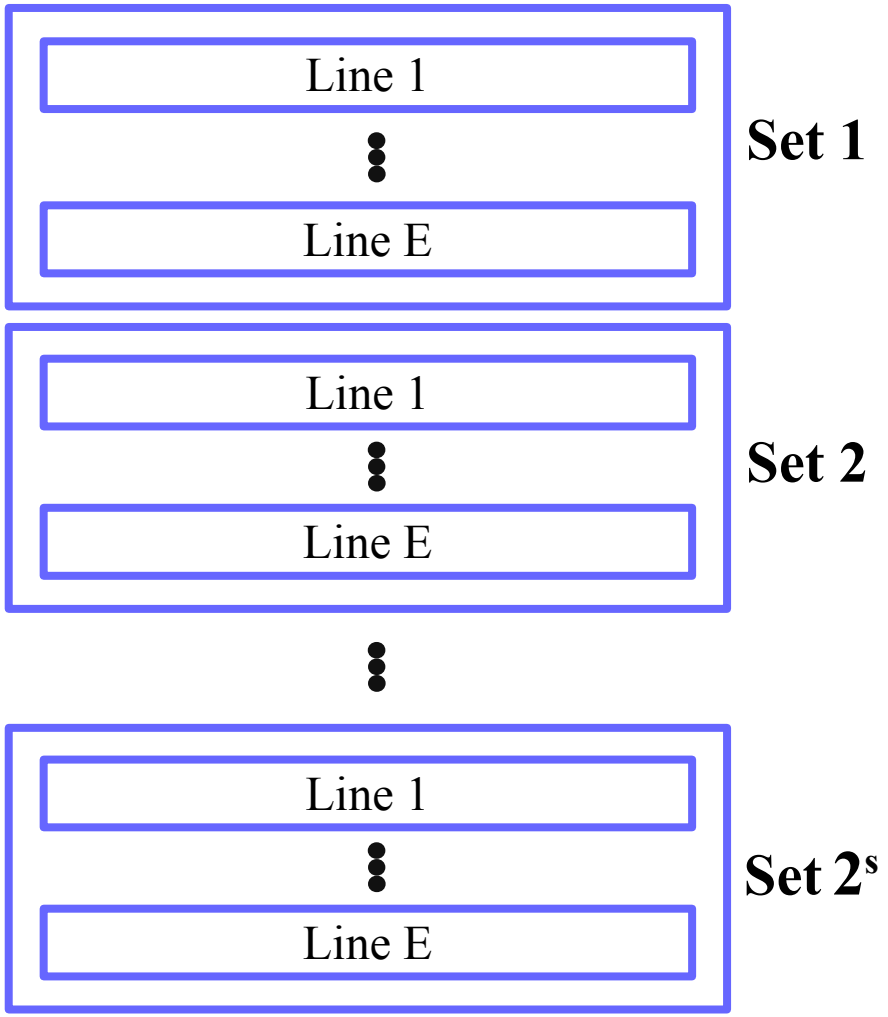
Cache Memory

- Store data or instruction which satisfies *principle of locality*(spatial, temporal)
- Address (tag bit, set index bit, block bits)
- Cache Memory(set, line, valid, block)





Cache Memory (Associativity)



Number of lines in a set determines **associativity**

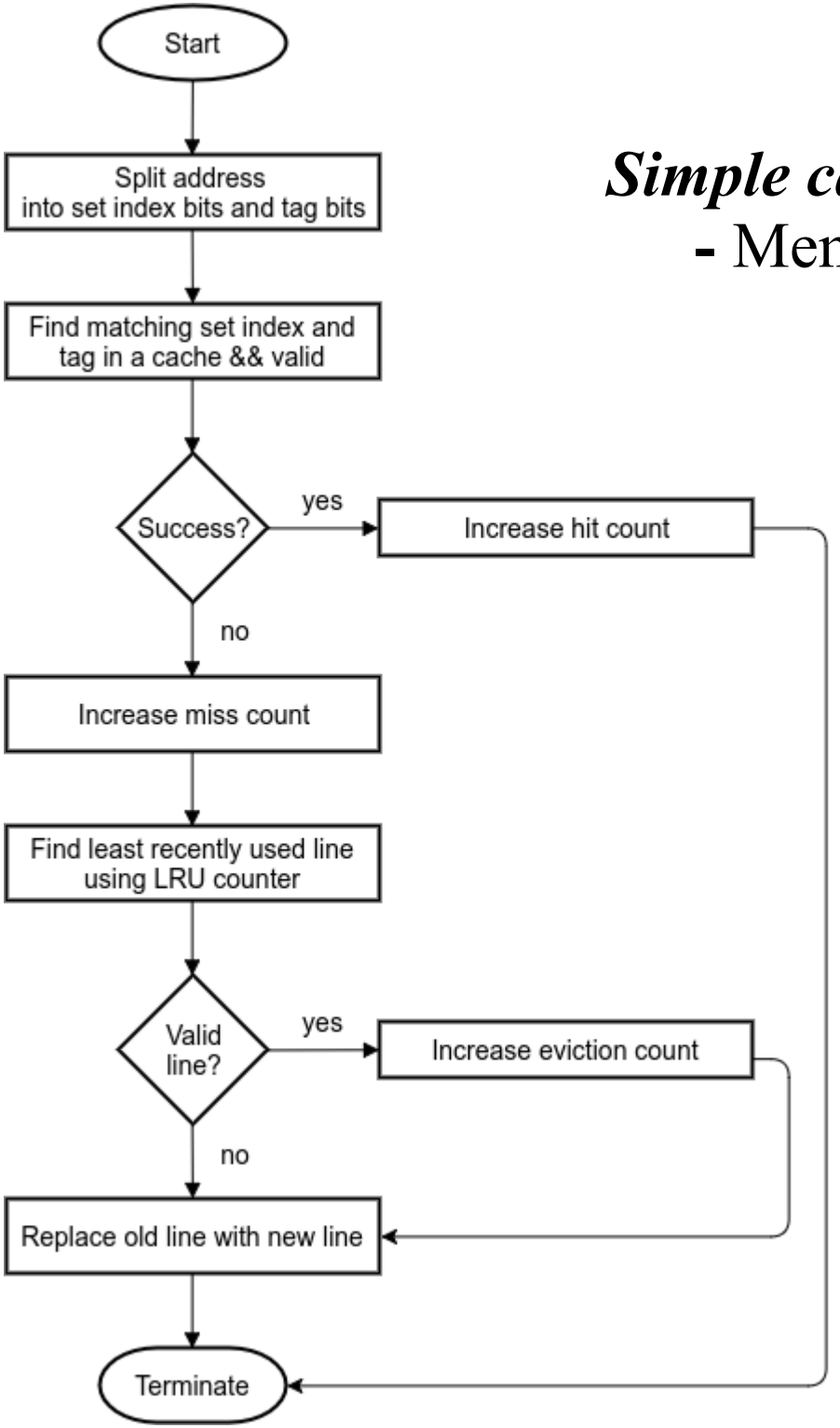
Associativity	Mapping
1	Fully Associative Mapping
$1 < n < 2^s$	Set Associative Mapping
2^s	Direct Mapping

Cache Hit / Miss / Eviction

- If a program attempts to access block X *and block X is in cache*, then this access is called '*hit*'
- If a program attempts to access block X and *block X is not in a cache*, then this access is called '*miss*'
  - If the address accessed by a program is already filled with another different block (not empty block, called *valid*), then this need to be given '*eviction*'
  - Need replacement policy
- (In this lab, we don't care about data, so only line mtaching is enough)

Cache Replacement Policy

- Size of cache memory is limited, thus not all blocks from upper layer are stored in a cache
- So cache memory needs to replace block in a cache with the new block to be used in the near future
  - How to pick the block to be evicted? (*victim block*)
- Replacement policy algorithms
  - **LRU:** Replace **L**east **R**ecently **U**sed line with the new one
    - Use counter to trace recently accessed line
  - FIFO, LIFO, ...



Simple cache simulator

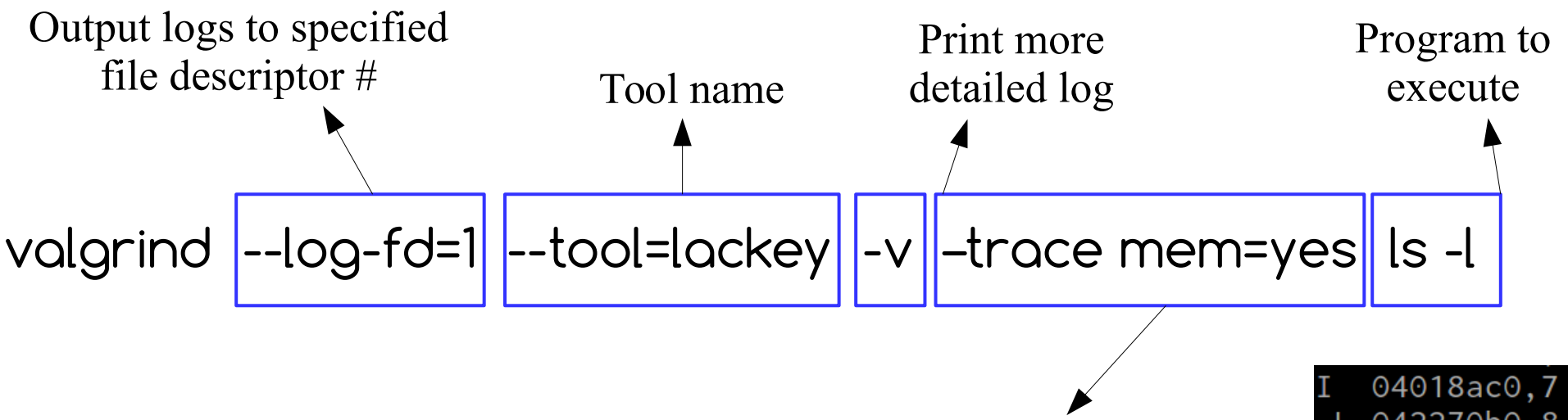
- Memory access

Valgrind

- A suite of tools for debugging and profiling programs to detect memory management problems
- Provided as a command line tool
- Usage
  - valgrind –tool=[tool-name] [options] [program to execute]
  - valgrind -tool=lackey ls
- Install (ubuntu assumed)
  - sudo apt install valgrind
- Reference valgrind linux manpage

Tools	Function
Memcheck	Memory Profiler
Cachegrind	Cache Profiler
Callgrind	Extension of Cachgrind
Massif	Heap Profiler
Helgrind	Multi-threaded Program Debugger
Lackey	Simple profiler

Valgrind Example



File Descriptor	#
standard input	0
standard output	1
standard error	2

prints the size and address of almost every memory access made by the program.

```
I  04018ac0,7
 L 042270b0,8
I  04018ac7,4
I  04018acb,4
I  04018acf,2
I  04018ad8,4
I  04018adc,7
 L 042270a0,8
I  04018ae3,5
 S ffefff7e8,8
I  0401b4f0,5
I  0401b4f5,2
I  0401b4f7,6
```

Check more file descriptors by “lsof”

See more info at file descriptor

Semantics of the output

[space] operation address, size

Operation	Meaning
I	Instruction load
L	Data load
S	Data store
M	Data modify

Virtual memory address that program accessed in hexadecimal format

number of bytes

For example,
"**I 04018ac0, 8**" means memory access for loading an instruction at memory address of 0x0400d7d4 which has a size of 8 bytes

```
I  04018ac0,7
 L 042270b0,8
I  04018ac7,4
I  04018acb,4
I  04018acf,2
I  04018ad8,4
I  04018adc,7
 L 042270a0,8
I  04018ae3,5
 S ffefff7e8,8
I  0401b4f0,5
I  0401b4f5,2
I  0401b4f7,6
```

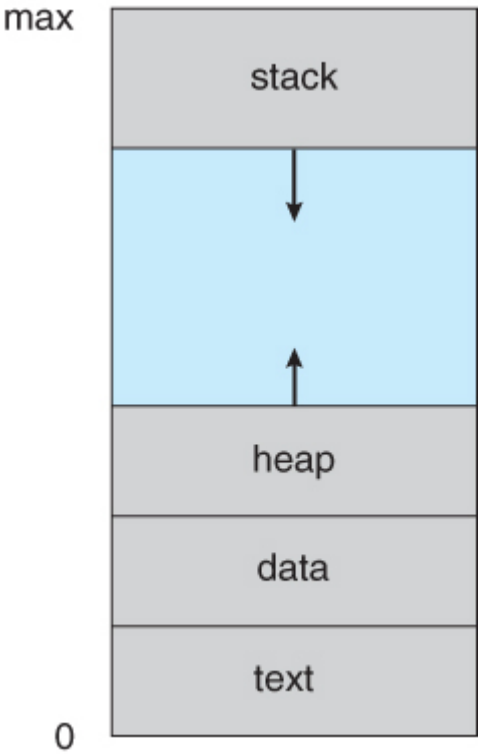
Important! 'M' means that it needs two memory accesses: One for loading and modifying, second for storing the result

File I/O APIs

- Low Level File Access
  - `open(),read(),write(),close()`
- High-level Standard I/O Library
  - `fopen(),fclose(),fread(),fwrite()`
  - `fscanf(),sscanf()` : **Formatted input**
  - `fgets()` : read a string with n bytes
- Usage pattern
  - `fp = fopen("text.txt", "r");`
  - Read text from "text.txt"
  - Do something with the file
  - `fclose(fp);`

Dynamic Memory Allocation/Deallocation

- `malloc()` allocates consecutive dynamic memory space in heap
- `free()` de-allocates specified memory space allocated by malloc()
- Example
  - Allocate int array with size of 16
    - `int *p = (int *)malloc(16);`
  - Free dynamically allocated space
    - `free(p);`



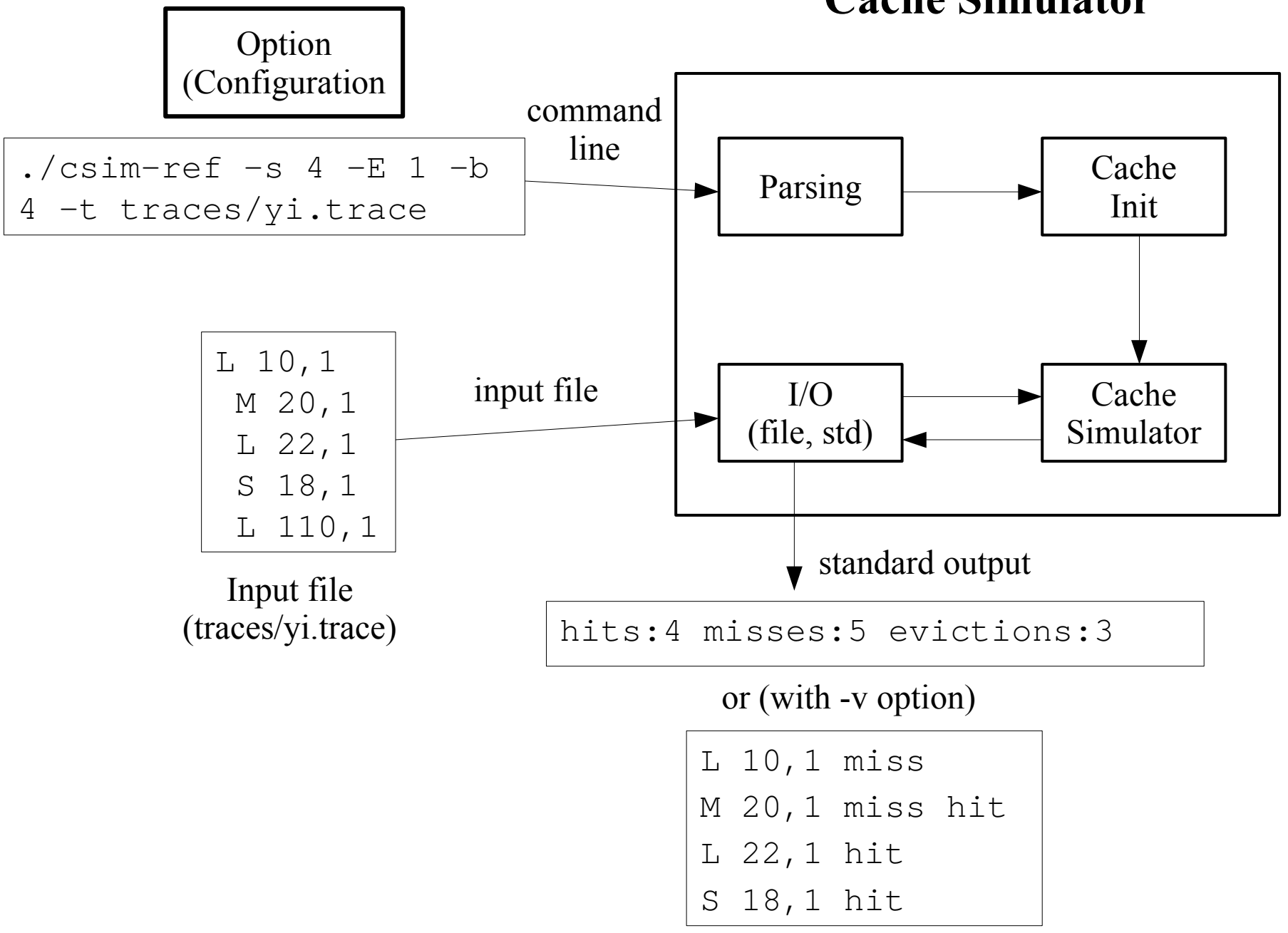
Parsing command line options

- A program usually need to react differently to different given configuration
- Short options (with/without argument)
  - “`ls -l`”,“`df -h`”,“`gcc -o [arg]` ”
- Long options (with/without argument)
  - “`ls --help`”,“`gcc --version`”,“`gcc -std=c11`”
- Two common option parsing library in POSIX implementation
  - `getopt()` : parse short options
  - `getopt_long()` : parse short + long options

Assignment #1 (Part A)
Writing a Cache Simulator

- Understand the mechanism of cache memory system and implement simplified cache simulator
- Write a program which simulates cache memory access and count hit/miss/eviction
- Input file is pre-generated by Valgrind tool
- Output contains hit/miss/eviction for each instruction in input file
- It doesn’t care about memory content. Only cares about tag bit and set index bit to find matching line

Cache Simulator



Today

- Write a program which allocates two dimensional int array with `r` rows and `c` columns
- Both `r` and `c` are from command line options which can be parsed using `getopt()`
- Options
  - `-r` : Number of row
  - `-c` : Number of column
  - `-h` : Print program usage
  - `./prog -r [number of row] -c [number of col]`

Assignment #2 (Part B)
Optimizing Matrix Transpose (next week)

- Writing matrix transpose operation function
- As few cache misses as possible using locality
- ...