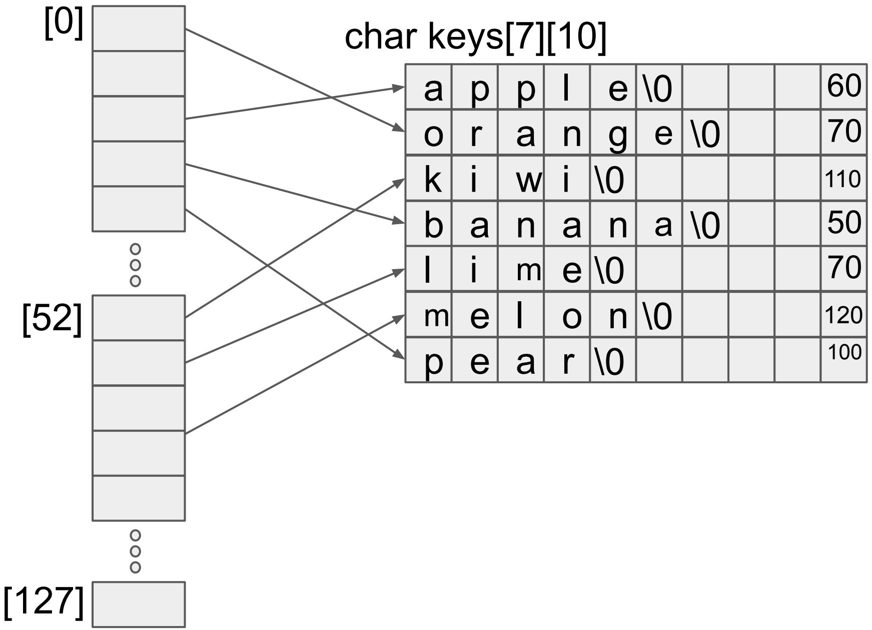
# File I/O

Michio Honda

# Recap

- Key Value Store
  - e.g., what is the price of the item “kiwi”?

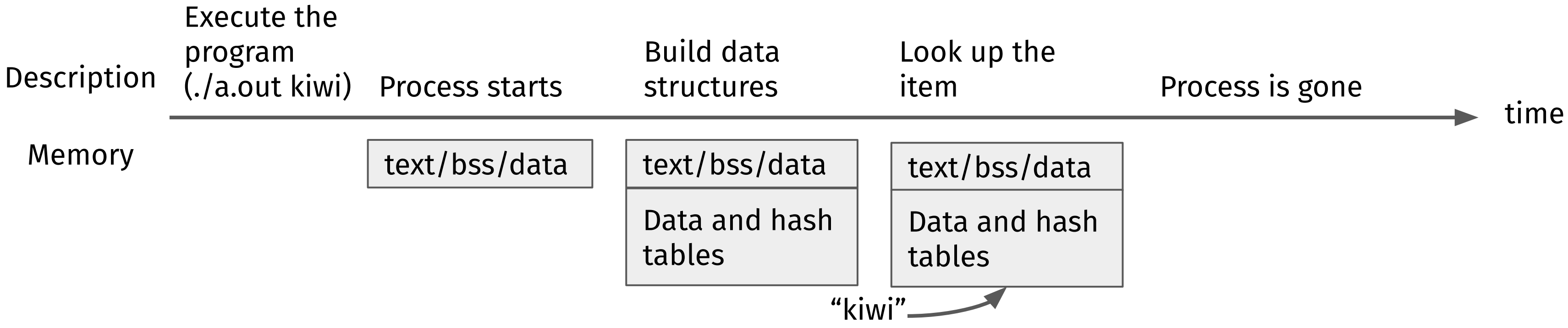


| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| a | p | p | l | e | \0 | | | | 60 |
| o | r | a | n | g | e | \0 | | | 70 |
| k | i | w | i | \0 | | | | | 110 |
| b | a | n | a | n | a | \0 | | | 50 |
| l | i | m | e | \0 | | | | | 70 |
| m | e | l | o | n | \0 | | | | 120 |
| p | e | a | r | \0 | | | | | 100 |

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

#define ITEMS   7
#define VALIDX 9
#define HASHSIZ 128

int
main(int argc, char **argv)
{
        int vals[ITEMS] = {60, 70, 110, 50, 90, 120, 100};
        char keys[ITEMS][VALIDX+1] = {"apple", "orange", "kiwi", "banana",
                                      "lime", "melon", "pear"};
        char *test = argv[1];
        char *hashtable[HASHSIZ];
        int i, j, hash;

        printf("%-8s %3s %4s\n", "item", "$/kg", "hash");
        for (i = 0; i < ITEMS; i++) {
                hash = 0;
                for (j = 0; keys[i][j] != '\0'; j++) {
                        hash += keys[i][j];
                }
                hashtable[hash % HASHSIZ] = keys[i];

                keys[i][VALIDX] = vals[i];
                printf("%-8s %3d %4d\n", keys[i], keys[i][VALIDX], hash);
        }

        // Now we want to find the price of something
        hash = 0;
        for (i = 0; test[i] != '\0'; i++) {
                hash += test[i];
        }
        printf("%s %d\n", test, hashtable[hash % HASHSIZ][VALIDX]);

        return 0;
}
```

# The lifetime of program execution

- All the data is gone when the program quits
  - Everything in the (virtual) memory



Description

Execute the program (./a.out kiwi)

Process starts

Build data structures

Look up the item

Process is gone

time

Memory

text/bss/data

text/bss/data

Data and hash tables

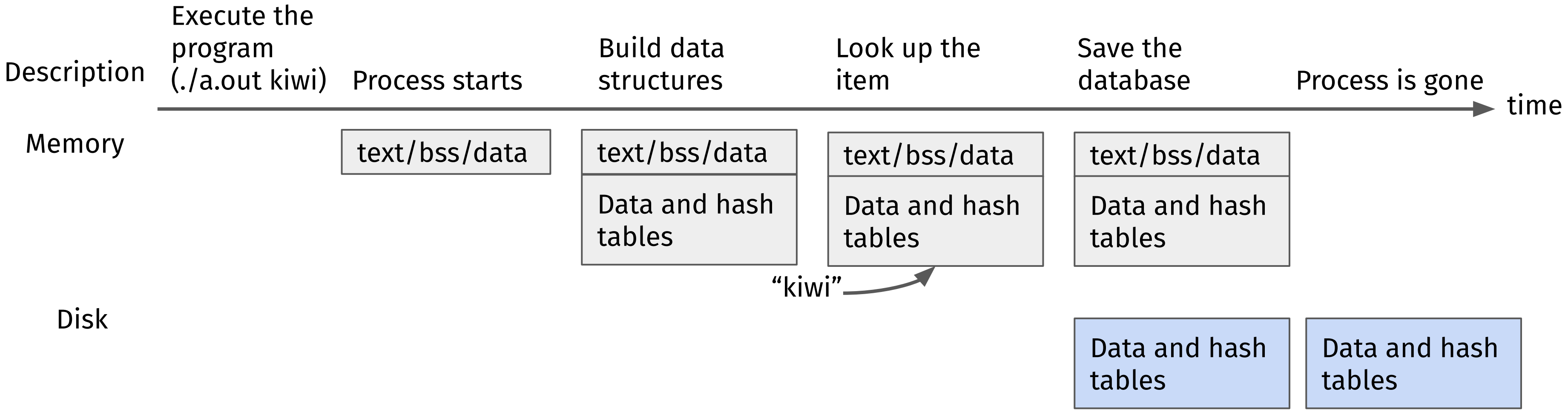
text/bss/data

Data and hash tables

“kiwi”

# What we want in this week

- Save the database in the disk
  - Avoid building the database every time it executes and the save the progress
  - Preserve the data over reboots (persistence)

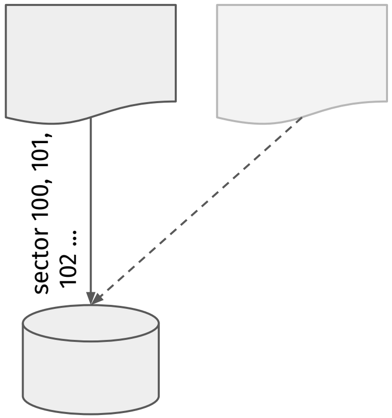


# Challenges

- Disk I/O
  - How to write data in the disk?
- On-disk data structures
  - What does the data written look like?

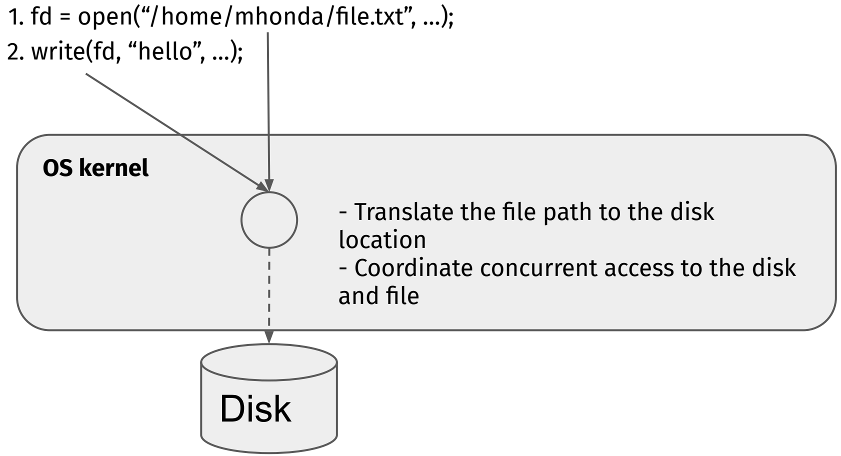
# Disk I/O

- How to store data in the disk?
  - Protection
    - What if another application writes to the same disk or same location?
    - If we simply lock the disk, what if the owning app dies while holding the lock?
  - Naming
    - Do we write data using disk's address (sector)?
    - Do we remember where (which sectors) we've written the data in the disk?



# Disk I/O (2)

- The OS kernel addresses these problems



1. fd = open(“/home/mhonda/file.txt”, ...);

2. write(fd, “hello”, ...);

**OS kernel**

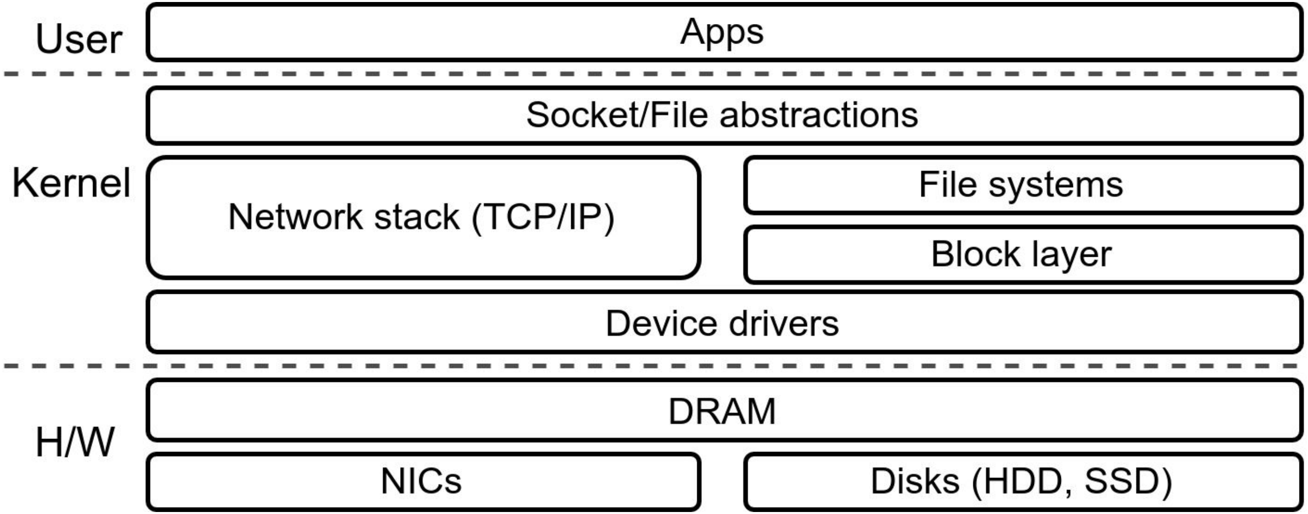
- Translate the file path to the disk location
- Coordinate concurrent access to the disk and file

# On-disk data structures

- You cannot write addresses
  - Open week3/3d.c
  - Add `printf(“%p\n”, hashtable);` before return;
  - Compile the code and check the output
  - Add or remove a single printf elsewhere
  - Compile and check again
- Exercise
  - Modify week3/3d2.c to store the data in a file

# Appendix: The OS Stack

- The OS addresses these problems


User
Apps
Kernel
Socket/File abstractions
Network stack (TCP/IP)
File systems
Block layer
Device drivers
H/W
DRAM
NICs
Disks (HDD, SSD)