Fundamental Abstractions in Computer Systems–Memory

Review:

1. Fundamental challenge to overcome with computer systems—complexity:
    a. Increases difficultly reasoning about correctness, debugging, and implementation.
    b. Created by large scope of systems problems
    c. Computers are probably the most complex things we've ever created!
2. Learned how to combat complexity—Modularity and Abstraction:
    a. Divide and Conquer lets us make the problem tractable
    b. The "box and arrows" pictures are easier to comprehend than the whole picture, provided that we draw the boxes well (i.e., use good abstractions)
3. But, what do each of the boxes do? How can we classify them? Can we say something general about the challenges and properties of different types of boxes?

Fundamental Abstractions:

1. Systems essentially do three things:
    a. Compute (Interpret) things (Interpreter)
    b. Store (remember) things (Memory)
    c. Communicate things (Communication channels)
2. Then, we'll explain how we use names to put these things together

Memory:

1. Storage, or a place to "remember" things.
2. Can be thought of as a map between a space of names and a space of values. (Alternatively, a table between the two spaces).
3. Fundamentally relies upon a "name" abstraction.
4. Interface
    a. Write (name, value): assign name to a new value. I.e., Add/replace an entry in a table. Create an arrow.
    b. Value = Read(name): get value of name. I.e., lookup in table, follow arrow.
5. What are some memories?
    a. Registers
    b. Caches (L1, L2, L3)
    c. DRAM
    d. SSDs

    e. HDDs
    f. Tape
6. We're going to look at these memories and think about what makes them similar and what makes them different. What are their properties?

Volatility: Does it need power to remember?

1. Many are non-volatile—they do not need power to remember
2. Some are volatile–they need power to remember

Coherency and Atomicity: What if there are multiple users?

1. Aside—Lamport "space-time" diagrams
    a. Used to understand behavior when we have multiple interacting components
    b. Each line is additional "space" (another module interacting)
    c. Time moves top to bottom
2. Read/Write Coherency—The result of a read to a named address is always the same as the "most recent" write.
3. Read/Write Atomicity—Result of any read or write is as if the read or write occurred completely before of completely after every other read write
4. Why are these things difficult?!?
    a. Reading/writing is actually multi-step:
    b. Data might be redundantly at multiple places at once.

Granularity: What values does it store? How do you name them?

1. Elements of the storage hierarchy are:
    a. Below registers: large arrays of bytes, addressable using numeric ranges
    b. byte-addressable—software can address each byte of storage
    c. Block-addressable—can address blocks of bytes of storage (often 1024)
2. Registers: Follow the pretty traditional name/value map
3. Generally, these devices have different error rates
    a. Usually, we think of block devices as failing frequently
    b. Byte storage interfaces do NOT expose potential error states:
4. "But, Dr. Q, I never write to a SSD or HDD using a block number!"
    a. Instead, we use file systems:
        i. Add a layer of indirection
        ii. Name files (readable names).
        iii. View files as contiguous streams rather than as blocks
        iv. Two ways of operating on file
            1. Explicitly treat file as stream of bytes (Open, Read, Write)
            2. Use Memory-Mapped I/O (mmap, munmap).
                a. When does the write actually happen?
        v. File Systems expose block device failures directly to the user
    b. Why did we build file systems, but not similar abstractions for memory?

Performance:

1. Drastic difference in latency across devices
2. Which seems more important, read latency or write latency?
3. Often Throughput is much closer than latency: use asynchronous I/O!

Tradeoffs

1. What problems arise with asynchronous I/O?!
    a. Non-volatility!
    b. Coherence, Atomicity!?