Intro to System Design Theory

Kay Ashaolu - Instructor

Aishwarya Sriram - TA

All Digital Data: Zeros and Ones

Fact:

All digital data is made up of zeros and ones.

- Every computer, smartphone, and digital device operates on binary.
- This simple foundation supports all complex systems.

The Binary System Fundamentals

- **Binary Digits:** 0 and 1 represent two states.
  - **0:** Off / No current
  - **1:** On / Current flowing
- These states form the basic language of computers.

Transistors: The Building Blocks

- **What are Transistors?**
  Semiconductor devices that act as switches.
- **Role:**
  - Allow or block the flow of electrical current.
  - Represent binary data (0 for off, 1 for on).

Early Computers & Binary

- Early systems used binary principles directly.
- **Punch Cards:**
  - Physical cards with holes to represent 1s (punched) and 0s (not punched).
  - Time-consuming and error-prone, but foundational.

Evolution: Low-Level to High-Level Languages

- **Low-Level:**
  - Assembly language, closer to machine code.
- **High-Level:**
  - Languages like Fortran, C++, Python.
- **Key Insight:**
  Regardless of abstraction, all code ultimately becomes binary.

Building Blocks: From Code to Systems

- **Concept:**
  Use modular, reusable components to build complex systems.
- **Analogy:**
  Just as functions build applications, system components combine to form robust architectures.

Core Computer Components

- **Task-Focused:**
  - **CPU:** Executes instructions.
  - **GPU:** Renders graphics and performs complex computations.
- **Storage-Focused:**
  - **RAM:** Fast, volatile memory for temporary data.
  - **Disk Storage:** Non-volatile memory for long-term data retention.

Task-Focused Components: CPU & GPU

- **CPU (Central Processing Unit):**
  - Performs arithmetic and logic operations.
  - Executes binary instructions.
- **GPU (Graphics Processing Unit):**
  - Highly efficient at mathematical calculations.
  - Key for rendering and machine learning tasks.

Storage-Focused Components: RAM & Disk

- **RAM (Random Access Memory):**
  - Volatile memory: Loses data when powered off.
  - Fast read/write for intermediate processing.
- **Disk Storage:**
  - Non-volatile: Retains data without power.
  - Larger capacity but slower access compared to RAM.

Data Representation: Storage vs. Tasks

- **Storage:**
  Binary data organized as numbers, text, images.
- **Tasks:**
  Binary instructions that tell the computer what to do.
- **Core Idea:**
  Both storage and tasks are based on zeros and ones.

Storage

Introduction to Data Structures

- **Purpose:**
  Organize and manage binary data effectively.
- **Key Benefit:**
  Transform raw bits into meaningful information.
- **Examples:**
  Arrays, dictionaries, trees, and graphs.

Arrays, Lists, & Vectors

- **Definition:**
  Linear sequences of data, indexed by numbers.
- **Key Points:**
  - Easy indexing (e.g., accessing the 0th element).
  - Efficient for ordered data.
- **Visual Example:**
  `[A, B, C, D]` where index 0 = A, index 1 = B, etc.

Dictionaries & Objects

- **Definition:**
  Key-value paired data structures.
- **Advantages:**
  - Descriptive keys (like a real dictionary).
  - Easy access to complex data.
- **Example (Python):**

```python
person = {
    "name": "Alice",
    "age": 29,
    "height": "5ft 6in"
}
print(person["name"])  # Output: Alice
```

Trees & Graphs: Modeling Relationships

- **Trees:**
  Hierarchical data (e.g., family trees).
- **Graphs:**
  More general networks (e.g., social connections).
- **Use Case:**
  Efficiently representing relationships between entities.

Trade-offs in Data Structures

- **Arrays:**
  - Fast indexing, simple memory layout.
- **Dictionaries:**
  - Flexible, descriptive access.
- **Trees/Graphs:**
  - Ideal for modeling complex relationships.
- **Decision Criteria:**
  - Consider access patterns, memory usage, and scalability.

Tasks

Understanding Tasks in Computing

- **Definition:**
  A task is a unit of work that transforms data.
- **Key Concept:**
  Tasks are executed by the CPU and represented in binary.
- **Example:**
  A function performing a calculation.

Task Execution: From Instructions to Action

- **Process:**
  1. **Fetch:** CPU retrieves binary instructions.
  2. **Execute:** Performs arithmetic, logic, and data movement.
  3. **Output:** Produces a result.
- **Result:**
  Raw data is transformed into meaningful output.

Functions as Tasks

- **Functions encapsulate tasks.**
- **Characteristics:**
  - Receive input parameters.
  - Execute a set of instructions.
  - Return a result.
- **Key Insight:**
  Modular functions are the building blocks of complex systems.

Example: Addition Function in Python

```python
def add(x, y):
    result = x + y  # Perform addition
    return result

# Using the function
sum_value = add(3, 4)
print(sum_value)  # Output: 7
```

- **Explanation:**
  The function receives two inputs, processes them, and returns the sum.

Storage and Tasks as Building Blocks

Storage & Tasks: Integration in Systems

- **Storage:**
  Provides the data.
- **Tasks:**
  Operate on that data.
- **Combined Effect:**
  Creating building blocks for system design.
- **Example:**
  A function (task) that manipulates data stored in arrays or dictionaries.

Modularity in System Design

- **Key Principle:**
  Combine smaller tasks and data structures to build larger systems.
- **Benefits:**
  - Easier debugging.
  - Enhanced maintainability.
  - Scalable and extendable architectures.
- **Focus:**
  Designing systems with clear, well-defined building blocks.

Real-World Analogy: Recipes & Ingredients

- **Recipe:**
  Represents a task or function.
- **Ingredients:**
  Represent the data (storage).
- **Analogy:**
  Just as a recipe transforms ingredients into a dish, functions transform raw data into useful outputs.

Recap: Key Concepts Covered

- **Binary Foundation:**
  Everything is zeros and ones.
- **Data Structures:**
  Arrays, dictionaries, trees, and graphs organize data.
- **Tasks:**
  Functions and instructions transform data.
- **System Design:**
  Integrates storage and tasks into modular building blocks.

Questions?