**Systems Fundamentals (SF)**

A computer system is a set of hardware and software infrastructure upon which applications are constructed. Computer systems have become a pillar of people's daily life. As such, learning the knowledge about computer systems, grasping the skills to use and design these systems, and understanding the fundamental rationale and principles in computer systems are essential to equip students with the necessary competency toward a career related to computer science.

In the curriculum of computer science, the study of computer systems typically spans across multiple courses, including, but not limited to, operating systems, parallel and distributed systems, communications networks, computer architecture and organization and software engineering. The System Fundamentals knowledge area, as suggested by its name, focuses on the fundamental concepts in computer systems that are shared by these courses within their respective cores. The goal of this knowledge area is to present an integrative view of these fundamental concepts in a unified albeit simplified fashion, providing a common foundation for the different specialized mechanisms and policies appropriate to the particular domain area. These concepts include an overview of computer systems, basic concepts such as state and state transition, parallelism, resource allocation and scheduling, and so on. Compared to CS2013, the SF knowledge area makes the following major changes to the knowledge units:

1. Added two new units: system security and system design;
2. Added a new unit of system performance, which includes the topics from the deprecated unit of proximity and the deprecated unit of virtualization and isolation;
3. Added a new unit of performance evaluation, which includes the topics from the deprecated unit of evaluation and the deprecated unit of quantitative evaluation;
4. Changed the unit of computational paradigms to overview of computer systems, deprecated some topics in the unit, and added topics from the deprecated unit of cross-layer communications;
5. Changed the unit of state and state transition to basic concepts, and added topics such as finite state machines;
6. Changed some topics in the unit of parallelism, such as simple application-level parallel processing;
7. Deprecated the unit of cross-layer communications, and moved parts of its topics to the unit of overview of computer systems;
8. Deprecated the units of evaluation and quantitative evaluation, and moved parts of their topics to the unit of performance evaluation;
9. Deprecated the units of proximity and virtualization and isolation, and moved parts of their topics to the unit of system performance;
10. Renamed the unit of reliability through redundancy to system reliability.

**SF. Systems Fundamentals. [18 CS Core hours, 9 KA Core hours]**

<table>
<thead>
<tr>
<th>Course Description</th>
<th>CS Core hours</th>
<th>KA Core hours</th>
<th>Includes Electives</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF/Overview of Computer Systems</td>
<td>3</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>SF/Basic Concepts</td>
<td>4</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>SF/Parallelism</td>
<td>3</td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>SF/Resource Allocation and Scheduling</td>
<td>1</td>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>SF/System Performance</td>
<td>2</td>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>SF/Performance Evaluation</td>
<td>2</td>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>SF/System Reliability</td>
<td>1</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>SF/System Security</td>
<td>1</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>SF/System Design</td>
<td>1</td>
<td>1</td>
<td>Y</td>
</tr>
</tbody>
</table>
SF/Overview of Computer Systems

[3 CS Core hours]

Topics:

- Basic building blocks and components of a computer (gates, flip-flops, registers, interconnections; datapath + control + memory)
- Hardware as a computational paradigm: Fundamental logic building blocks; Logic expressions, minimization, sum of product forms
- Programming abstractions, interfaces, use of libraries
- Distinction between application and OS services, remote procedure call
- Application-OS interaction
- Basic concept of pipelining, overlapped processing stages
- Basic concept of scaling: going faster vs. handling larger problems

Learning Outcomes:

1. Describe the basic building blocks of computers and their role in the historical development of computer architecture.
2. Design a simple logic circuit using the fundamental building blocks of logic design to solve a simple problem (e.g., adder).
3. Use tools for capture, synthesis, and simulation to evaluate a logic circuit design.
4. Describe how computing systems are constructed of layers upon layers, based on separation of concerns, with well-defined interfaces, hiding details of low layers from the higher layers.
5. Describe that hardware, OS, VM, application are additional layers of interpretation/processing.
6. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers.
7. Construct a simple program (e.g., a TCP client/server) using methods of layering, error detection and recovery, and reflection of error status across layers.
8. Find bugs in a layered program by using tools for program tracing, single stepping, and debugging.
9. Understand the concept of strong vs. weak scaling, i.e., how performance is affected by scale of problem vs. scale of resources to solve the problem. This can be motivated by simple, real-world examples.

SF/Basic Concepts

[4 CS Core hours]

Topics:

- Digital vs. Analog/Discrete vs. Continuous Systems
- Simple logic gates, logical expressions, Boolean logic simplification
- Clocks, State, Sequencing
- State and state transition (e.g., starting state, final state, life cycle of states)
- Finite state machines (e.g., NFA, DFA)
- Combinational Logic, Sequential Logic, Registers, Memories
- Computers and Network Protocols as examples of State Machines

Learning Outcomes:

1. Describe the differences between digital and analog systems, and between discrete and continuous systems. Can give real-world examples of these systems.
2. Describe computations as a system characterized by a known set of configurations with transitions from one unique configuration (state) to another (state).
3. Describe the distinction between systems whose output is only a function of their input (stateless) and those with memory/history (stateful).
4. Develop state machine descriptions for simple problem statement solutions (e.g., traffic light sequencing, pattern recognizers).
5. Describe a computer as a state machine that interprets machine instructions.
6. Explain how a program or network protocol can also be expressed as a state machine, and that alternative representations for the same computation can exist.
7. Derive time-series behavior of a state machine from its state machine representation (e.g., TCP connection management state machine).

SF/Parallelism

[3 CS Core hours]

[Cross-reference: PD/Parallelism Fundamentals]

Topics:

- Sequential vs. parallel processing
- Application-level sequential processing: single thread
- Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers, pipelining
- Parallel programming vs. concurrent programming
- Request parallelism vs. Task parallelism
- Multicore architectures and hardware support for synchronization

Learning Outcomes:

1. Write a simple sequential problem and a simple parallel version of the same program.
2. Evaluate the performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved.
3. Demonstrate on an execution timeline that parallelism events and operations can take place simultaneously (i.e., at the same time). Explain how work can be performed in less elapsed time if this can be exploited.
4. Explain other uses of parallelism, such as for reliability/redundancy of execution.
5. Describe the differences between the concepts of instruction parallelism, data parallelism, thread parallelism/multitasking, task/request parallelism.
6. Write one simple parallel program (e.g., PageRank) in more than one parallel programming paradigm (e.g., a simple parallel program that manages shared resources through synchronization primitives; and a simple parallel program that performs simultaneous operations on partitioned data through task parallelization).
7. Use performance tools to measure speed-up achieved by parallel programs in terms of both problem size and number of resources.

SF/Resource Allocation and Scheduling

[1 CS Core hour and 2 KA Core hours]

Topics:

- Different types of resources (e.g., processor share, memory, disk, net bandwidth)
- Common scheduling algorithms (e.g., first-come-first-serve scheduling, priority-based scheduling, fair scheduling and preemptive scheduling)
Advantages and disadvantages of common scheduling algorithms

**Learning Outcomes:**

1. Define how finite computer resources (e.g., processor share, memory, storage and network bandwidth) are managed by their careful allocation to existing entities.
2. Describe how common scheduling algorithms work.
3. Describe the pros and cons of common scheduling algorithms.
4. Implement common scheduling algorithms, and evaluate their performances.

**SF/System Performance**

**[2 CS Core hours and 2 KA Core hours]**

[Cross-reference: AR/Memory Management, OS/Virtual Memory]

**Topics:**

- Latencies in computer systems
  - Speed of light and computers (one foot per nanosecond vs. one GHz clocks)
  - Memory vs. disk latencies vs. across the network memory
- Caches and the effects of spatial and temporal locality on performance in processors and systems
- Caches and cache coherency in databases, operating systems, distributed systems, and computer architecture
- Introduction into the processor memory hierarchy and the formula for average memory access time
- Rationale of virtualization and isolation: protection and predictable performance
- Levels of indirection, illustrated by virtual memory for managing physical memory resources
- Methods for implementing virtual memory and virtual machines

**Learning Outcomes:**

1. Explain the importance of locality in determining system performance.
2. Calculate average memory access time and describe the tradeoffs in memory hierarchy performance in terms of capacity, miss/hit rate, and access time.
3. Explain why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources.
4. Describe how the concept of indirection can create the illusion of a dedicated machine and its resources even when physically shared among multiple programs and environments.
5. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation.

**SF/Performance Evaluation**

**[2 CS Core hours and 2 KA Core hours]**

**Topics:**

- Performance figures of merit
- Workloads and representative benchmarks, and methods of collecting and analyzing performance figures of merit
- CPI (Cycles per Instruction) equation as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations.
- Amdahl’s Law: the part of the computation that cannot be sped up limits the effect of the parts that can
Analytical tools to guide quantitative evaluation
- Order of magnitude analysis (Big O notation)
- Analysis of slow and fast paths of a system
- Events on their effect on performance (e.g., instruction stalls, cache misses, page faults)
- Understanding layered systems, workloads, and platforms, their implications for performance, and the challenges they represent for evaluation
- Microbenchmarking pitfalls

**Learning Outcomes:**
1. Explain how the components of system architecture contribute to improving its performance.
2. Explain the circumstances in which a given figure of system performance metric is useful.
3. Explain the usage and inadequacies of benchmarks as a measure of system performance.
4. Describe Amdahl’s law and discuss its limitations.
5. Use limit studies or simple calculations to produce order-of-magnitude estimates for a given performance metric in a given context.
6. Use software tools to profile and measure program performance.
7. Design and conduct a performance-oriented experiment of a common system (e.g., an OS and Spark).
8. Conduct a performance experiment on a layered system to determine the effect of a system parameter on figure of system performance.

**SF/System Reliability**

**[1 CS Core hour and 1 KA Core hour]**

**Topics:**
- Distinction between bugs and faults
- Reliability through redundancy: check and retry
- Reliability through redundancy: redundant encoding (error correction codes, CRC, FEC)
- Reliability through redundancy: duplication/mirroring/replicas
- Other approaches to reliability

**Learning Outcomes:**
1. Explain the distinction between program errors, system errors, and hardware faults (e.g., bad memory) and exceptions (e.g., attempt to divide by zero).
2. Articulate the distinction between detecting, handling, and recovering from faults, and the methods for their implementation.
3. Describe the role of error correction codes in providing error checking and correction techniques in memories, storage, and networks.
4. Apply simple algorithms for exploiting redundant information for the purposes of data correction.
5. Compare different error detection and correction methods for their data overhead, implementation complexity, and relative execution time for encoding, detecting, and correcting errors.

**SF/System Security**

**[1 CS Core hour and 1 KA Core hour]**

**Topics:**
- Common system security issues (e.g., virus, denial-of-service attack and eavesdropping)
- Countermeasures
  - Cryptography
  - Security architecture
Intrusion detection systems, firewalls

Learning Outcomes:

1. Describe some common system security issues and give examples.
2. Describe some countermeasures against system security issues.

SF/System Design

[1 CS Core hour and 1 KA Core hour]

Topics:

● Common criteria of system design (e.g., liveness, safety, robustness, scalability and security)
● Designs of representative systems (e.g., Apache web server, Spark and Linux)

Learning Outcomes:

1. Describe common criteria of system design.
2. Given the functionality requirements of a system and its key design criteria, provide a high-level design of this system.
3. Describe the design of some representative systems.

Desirable Professional Dispositions

Professional dispositions are part of the competency model promised in our vision statement. Professional dispositions are “cultivated behaviors” desirable in the workplace. They are malleable and observable. The dispositions listed in earlier curricular volumes include:

● Proactive: take initiative, self-starter, independent
● Self-directed: self-learner, self-motivated
● Goal-driven
● Professional: exercise discretion, behave ethically
● Responsible: act appropriately
● Adaptable: Flexible
● Collaborative: team player
● Responsive
● Meticulous: attentive to detail, thorough
● Inventive: look beyond simple solutions

This is not a comprehensive list. Please feel free to add your own, e.g., “Follow instructions” is important for beginning programmers. Professional dispositions differ from skills in that they involve willingness and agency, e.g., it is not sufficient for one to have the skills to collaborate with others. One must also have the willingness to collaborate, the latter being a disposition.

The most desirable professional dispositions for this knowledge area are:

● Meticulousness: students must pay attention to details of different perspectives when learning about and evaluating systems.
Adaptiveness: students must be flexible and adaptive when designing systems. Different systems have different requirements, constraints and working scenarios. As such, they require different designs. Students must be able to make appropriate design decisions correspondingly.

**Necessary and Desirable Math**

The Math necessary for this knowledge area includes:

- Discrete Math:
  - Sets and relations
  - Basic graph theory
  - Basic logic
- Linear Algebra:
  - Basic matrix operations
- Probability and Statistics
  - Random variable
  - Bayes theorem
  - Expectation and Variation
  - Cumulative distribution function and probability density function

The Math desirable for this knowledge area includes:

- Queueing theory
- Stochastic process

**Shared and Crosscutting Concepts**

*Working definition*: Shared concepts are specific concepts shared with one or some other knowledge areas. Crosscutting concepts are abstract concepts that are shared with most of the knowledge areas. Crosscutting concepts include abstraction, encapsulation, generalization, specialization, reliability, efficiency, tradeoff, modeling, simulation, composition, usability, and approximation.

This knowledge area shares multiple concepts with knowledge areas of networking and communication (NC), operating system (OS), architecture and organization (AR) and parallel and distributed computing (PDC).

**Subcommittee**

Chair: Qiao Xiang, Xiamen University, Xiamen, China

Subcommittee members:

- Doug Lea, State University of New York at Oswego, Oswego, NY, United States
- Monica D. Anderson, University of Alabama, Tuscaloosa, AL, United States
- Matthias Hauswirth, University of Lugano, Lugano, Switzerland
● Ennan Zhai, Alibaba Group, Hangzhou, China

Other contributors:

● Michael S. Kirkpatrick, James Madison University, Harrisonburg, VA, United States
● Linghe Kong, Shanghai JiaoTong University, Shanghai, China