Systems Fundamentals (SF)

The underlying hardware and software infrastructure upon which applications are constructed is collectively described by the term "computer systems." Computer systems broadly span the sub-disciplines of operating systems, parallel and distributed systems, communications networks, and computer architecture. Traditionally, these areas are taught in a non-integrated way through independent courses. However these sub-disciplines increasingly share important common fundamental concepts within their respective cores. These include computational paradigms, parallelism, cross-layer communications, state and state transition, resource allocation and scheduling, and so on. This knowledge area presents an integrative cross-layer 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 knowledge areas that it underlies. An organizing principle is “programming for performance”: what does a programmer need to know about the underlying system in order to achieve high performance in an application being developed.

SF. Systems Fundamentals [18 core Tier 1, 9 core Tier 2 hours, 27 total]

	Core-Tier 1 hours	Core-Tier 2 hours	Includes Electives
SF/Computational Paradigms	**3**		
SF/Cross-Layer Communications	**3**		
SF/State-State Transition-State Machines	**6**		
SF/System Support for Parallelism	**3**		
SF/Performance	**3**		
SF/Resource Allocation and Scheduling		**2**	
SF/Proximity		**3**	
SF/Virtualization and Isolation		**2**	
SF/Reliability through Redundancy		**2**	

SF/Computational Paradigms

[3 Core-Tier 1 hours]

[Cross-reference PD/parallelism fundamentals: The view presented here is the multiple representations of a system across layers, from hardware building blocks to application components, and the parallelism available in each representation; PD/parallelism fundamentals focuses on the application structuring concepts for parallelism.]

Topics:

- A computing system as a layered collection of representations
- 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 gates, flip-flops, counters, registers, PL); Logic expressions, minimization, sum of product forms
- Application-level sequential processing: single thread [xref PF/]
- Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers
- Basic concept of pipelining, overlapped processing stages
- Basic concept of scaling: going faster vs. handling larger problems

Learning Outcomes:

1. List commonly encountered patterns of how computations are organized [Knowledge].
2. Describe the basic building blocks of computers and their role in the historical development of computer architecture [Knowledge].
3. Articulate the differences between single thread vs. multiple thread, single server vs. multiple server models, motivated by real world examples (e.g., cooking recipes, lines for multiple teller machines, couple shopping for food, wash-dry-fold, etc.) [Knowledge].
4. Articulate 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 the simple, real-world examples [Knowledge].
5. Design and simulate a simple logic circuit using the fundamental building blocks of logic design [Application].
6. Write a simple sequential problem and a simple parallel version of the same program [Application].
7. Evaluate performance of simple sequential and parallel versions of a program with different problem sizes, and be able to describe the speed-ups achieved [Evaluation].

SF/Cross-Layer Communications

[3 Core-Tier 1 hours]

Topics:

- Programming abstractions, interfaces, use of libraries
- Distinction between application and OS services, remote procedure call
- Interactions between applications and virtual machines
- Reliability

***Learning Outcomes**:*

1. 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 [knowledge].
2. Recognize that hardware, VM, OS, application are just additional layers of interpretation/processing [knowledge].
3. Describe the mechanisms of how errors are detected, signaled back, and handled through the layers [knowledge]
4. Construct a simple program using methods of layering, error detection and recovery, and reflection of error status across layers [application].
5. Find bugs in a layered program by using tools for program tracing, single stepping, and debugging [evaluation].

SF/State-State Transition-State Machines

[6 Core-Tier 1 hours]

[Cross-reference AL/Basic Computability and Complexity, OS/state and state diagrams, NC/protocols]

Topics:

- Digital vs. analog/discrete vs. continuous systems
- Simple logic gates, logical expressions, Boolean logic simplification
- Clocks, state, sequencing
- Combinational Logic, Sequential Logic, Registers, Memories
- Computers and Network Protocols as examples of State Machines

Learning Outcomes:

1. Describe computations as a system with a known set of configurations, and a byproduct of the computation is to transition from one unique configuration (state) to another (state) [Knowledge].
2. Recognize the distinction between systems whose output is only a function of their input (Combinational) and those with memory or history (Sequential) [Knowledge].
3. Describe a computer as a state machine that interprets machine instructions [Knowledge].
4. 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 [Knowledge].
5. Develop state machine descriptions for simple problem statement solutions (e.g., traffic light sequencing, pattern recognizers) [Application].
6. Derive time-series behavior of a state machine from its state machine representation [Evaluation].

SF/System Support for Parallelism

[3 Core-Tier1 hours]

[Cross-reference: PD/Parallelism Fundamentals]

***Topics**:*

- Execution and runtime models that distinguish Sequential vs. Parallel processing
- System organizations that support Request and Task parallelism and other parallel processing paradigms, such as Client-Server/Web Services, Thread parallelism(Fork-Join), and Pipelining
- Multicore architectures and hardware support for parallelism

***Learning Outcomes**:*

1. For a given program, distinguish between its sequential and parallel execution, and the performance implications thereof [knowledge].
2. Demonstrate on an execution time line that parallel 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 [knowledge].
3. Explain other uses of parallelism, such as for reliability/redundancy of execution [knowledge].
4. Define the differences between the concepts of Instruction Parallelism, Data Parallelism, Thread Parallelism/Multitasking, Task/Request Parallelism.
5. Write a simple parallel program in more than one paradigm so as to be able to compare and contrast ease of expression and performance in solving a given problem [application].
6. Use performance tools to measure speed-up achieved by parallel programs in terms of both problem size and number of resources [evaluation].

SF/Performance

[3 Core-Tier 1 hours]

[Cross-reference PD/Parallel Performance]

***Topics**:*

- Figures of performance merit (e.g., speed of execution, energy consumption, bandwidth vs. latency, resource cost)
- Benchmarks (e.g., SPEC) and measurement methods
- CPI equation (Execution time = # of instructions * cycles/instruction * time/cycle) 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

***Learning Outcomes**:*

1. Explain how the components of system architecture contribute to improving its performance [Knowledge].
2. Describe Amdahl's law and its implications for parallel system speed-up when limited by sequential portions, e.g., in processing pipelines [Knowledge].
3. Benchmark a parallel program with different data sets in order to iteratively improve its performance [Application].
4. Use software tools to profile and measure program performance [Evaluation].

SF/Resource Allocation and Scheduling

[2 Core-Tier 2 hours]

***Topics**:*

- Kinds of resources: processor share, memory, disk, net bandwidth
- Kinds of scheduling: first-come, priority
- Advantages of fair scheduling, preemptive scheduling

***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 [Knowledge].
2. Describe the scheduling algorithms by which resources are allocated to competing entities, and the figures of merit by which these algorithms are evaluated, such as fairness [Knowledge].
3. Implement simple scheduling algorithms [Application].

4. Measure figures of merit of different scheduler implementations [Evaluation].

SF/Proximity

[3 Core-Tier 2 hours]

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

Topics:

- Speed of light and computers (one foot per nanosecond vs. one GHz clocks)
- Latencies in computer systems: memory vs. disk latencies vs. across the network memory
- Caches, spatial and temporal locality, in processors and systems
- Elementary introduction into the processor memory hierarchy: registers and multi-level caches, and the formula for average memory access time

Learning Outcomes:

1. Explain the importance of locality in determining performance [Knowledge].
2. Describe why things that are close in space take less time to access [Knowledge].
3. Calculate average memory access time and describe the tradeoffs in memory hierarchy performance in terms of capacity, miss/hit rate, and access time [Evaluation].

SF/Virtualization and Isolation

[2 Core-Tier 2 hours]

Topics:

- Rationale for 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 why it is important to isolate and protect the execution of individual programs and environments that share common underlying resources, including the processor, memory, storage, and network access [Knowledge].
2. 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 [Knowledge].
3. Measure the performance of two application instances running on separate virtual machines, and determine the effect of performance isolation [Evaluation].

SF/Reliability through Redundancy

[2 Core-Tier 2 hours]

***Topics**:*

- Distinction between bugs and faults, and how they arise in hardware vs. software
- How errors increase the longer the distance between the communicating entities; the end-to-end principle as it applies to systems and networks
- Redundancy through check and retry
- Redundancy through redundant encoding (error correcting codes, CRC/Cyclic Redundancy Codes, FEC/Forward Error Correction)
- Duplication/mirroring/replicas

***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) [Knowledge].
2. Articulate the distinction between detecting, handling, and recovering from faults, and the methods for their implementation [Knowledge].
3. Describe the role of error correcting codes in providing error checking and correction techniques in memories, storage, and networks [Knowledge].
4. Apply simple algorithms for exploiting redundant information for the purposes of data correction [Application].
5. Compare different error detection and correction methods for their data overhead, implementation complexity, and relative execution time for encoding, detecting, and correcting errors [Evaluation].