CS1300: **Introduction to CSE**

A computer is a programmable machine designed to sequentially and automatically carry out a sequence of arithmetic or logical operations (computations).

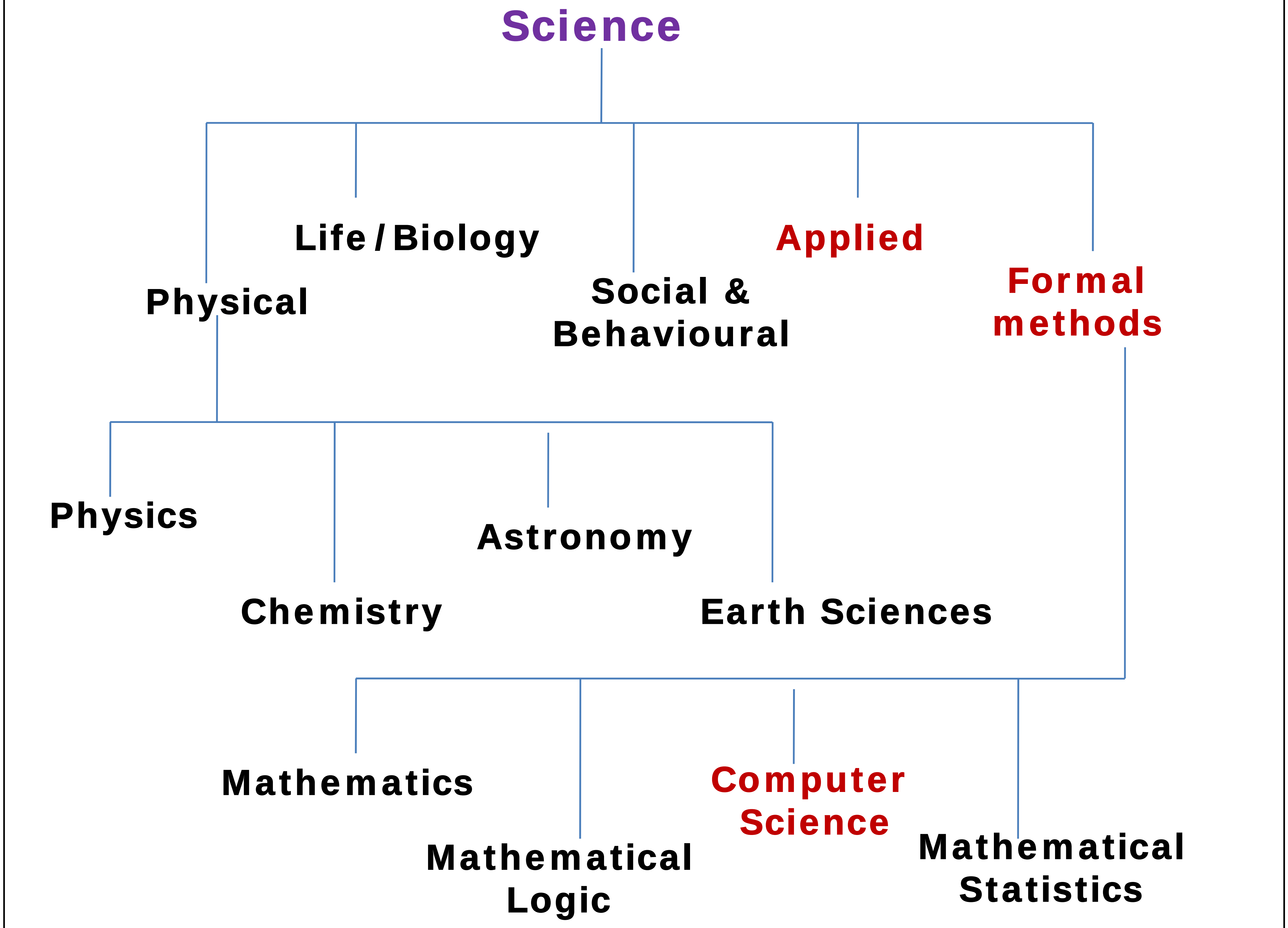
The particular sequence of operations can be changed readily, allowing the computer to solve more than one kind of problem.

Science (Latin: scientia means "knowledge") is a systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the world.

Scientific concepts is a body of reliable knowledge that can be logically and rationally explained.

Engineering is the discipline, art, skill and profession of acquiring and applying scientific, mathematical, economic, social, and practical knowledge, in order to design and build structures, machines, devices, systems, materials and processes that safely realize improvements to the lives of people.

It is a creative application of scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to forecast their behavior under specific operating conditions;


Science
Life / Biology
Applied
Physical
Social &
Behavioural
Formal
methods
Physics
Astronomy
Chemistry
Earth Sciences
Mathematics
Computer
Science
Mathematical
Logic
Mathematical
Statistics

REFERENCES

1. **G. Polya, "How to Solve It**", 2nd ed., Princeton University Press, 1957, ISBN: 0-691-08097-6.

2. **Jon Bentley, "Programming Pearls**", Addison-Wesley, Inc., 2000, ISBN: 0-201-65788-0.

3. **David Reed; A balanced Introduction to Computer Science**; Prentice Hall, 2004.

What is Computer Science?

Computer science (CS) is the systematic study of algorithmic methods for representing and transforming information, including their theory, design, implementation, application and efficiency.

Development ranges from computability theory, the invention of the stored-program electronic computer to modern VLSI and cloud computing. The roots of cs extend deeply into mathematics and engineering. Mathematics imparts analysis to the field; engineering imparts design.

What is Computer Engineering?

Computer engineering (CENG) is the design and prototyping of computing devices and systems.

While sharing much history and many areas of interest with CS, CENG concentrates its effort on the ways in which computing ideas are mapped into working physical systems. Emerging equally from the disciplines of CS and EE, CENG rests on the intellectual foundations of these disciplines, the basic physical sciences and mathematics.

Computer science is the study of the theoretical foundations of information and computation and of practical techniques for their implementation and application in computer systems.

Computer scientists invent algorithmic processes that create, describe, and transform information and formulate suitable abstractions to model complex systems.

The body of knowledge of computing is frequently described as the systematic study of algorithmic processes that describe and transform information: covering theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is:

What can be (efficiently) automated?

The profession contains various specialities such as computer science, computer engineering, software engineering, information systems, domain-specific applications, and computer systems.

It’s a rich discipline, with (often) nothing connected between designers/experts and users of the CS systems.

Computer Applications vs. Systems

Applications: Study of Information Processing tasks and their related data representations, to solve real-life problems in domains of real-life sciences and other branches of engineering and Applied Sciences.

Computing Systems: Efficient Structures, mechanisms and schemes for processing information. The real-life problem domain is often not considered here.

To be an expert, one must have knowledge of both. Computer Education attempts to elucidate the relationship between application and computer systems.

Line of division may not be clean always. Overlap exists over areas/problems such as:

Languages, O/S, Networks, HCI, etc.

Relationship with Other Science disciplines

- CS is more close to MATHS, than Physics (comes next, necessary for hardware design), Chem and Biology;

  - Mathematical logic, Theorems of Turing, Godel; Boolean Algebra, maths in algorithm analysis etc.

- EE is more close to CSE than other disciplines of Engg,. (its stronger bond than Chemistry vs. Chem. Engg, aircraft design and CFD, pharmacy and biology, material science and nano etc.

- Some Algos., were specifically designed to solve Engg. problems – FFT, Flight simulators, CAD/CAM, VLSI, spatial data analysis and display (GIS, Graphics etc.)

- Computing is called an "Engineering Science".

- Of course, it has brought in a few scientists/engineers from diversified fields together (?); - in Biocomputing, Robotics; GIS etc.

- Most importantly, many other fields have benefited due to CSE, where attempts to solve problems of "grand challenge" demand massively high-speed parallel computations.

What do we study in INTRO of “INTRO to CSE” ??

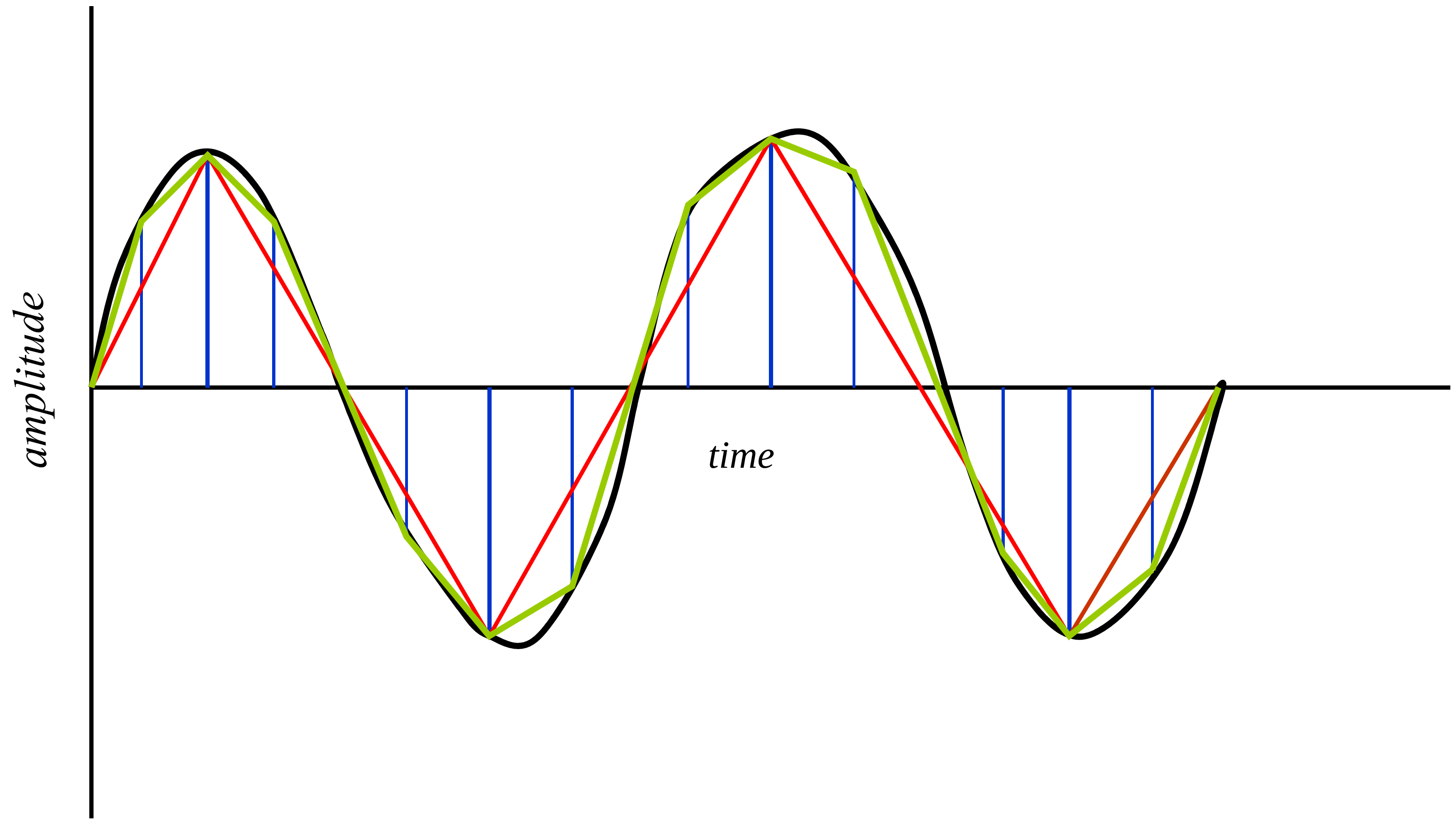
- Digital Systems
- Practical USAGE of Computers
- History of developments
- Domain and Applications
- Your courses in next 4-5 years
- Areas and Sub-areas
- Issues and Concerns
- Programming/Algorithms

Digital versus Analog Systems

- In a *digital* system information is represented and processed in *discrete* rather than continuous forms.

- Systems based on *continuous* forms of information are called *analog* systems

Continuous and Discrete



Advantages of Digital Systems

- More flexibility
- Can design high speed circuits
- Better precision
- Same results for same inputs
  - Analog circuits vary with respect to temperature, voltage etc.
- Information storage and retrieval functions are easier.
- Built in error detection and correction are possible
- Can build smaller systems

Disadvantages of Digital Systems

- World outside a digital,
  while computer is analog!

- Solution:
  - Input: Analog-to-Digital (A/D) converter at the input end.
  - Processing: Done using a digital system.
  - Output: Digital-to-Analog (D/A) converter at the output end.

What are computers used for?

- Traditional:
  - Automation
  - Control
- New Age:
  - Communication
  - Entertainment
  - Socializing
- Pervades all aspects of life

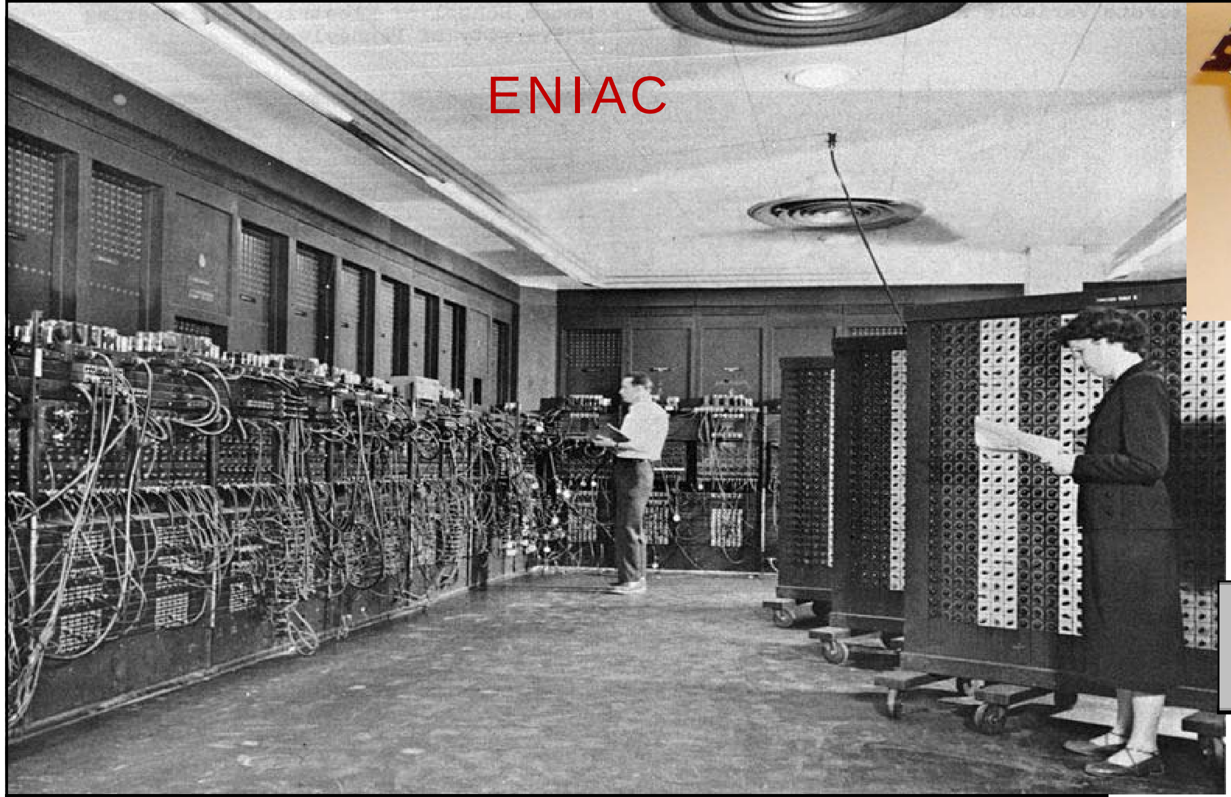
Information Representation

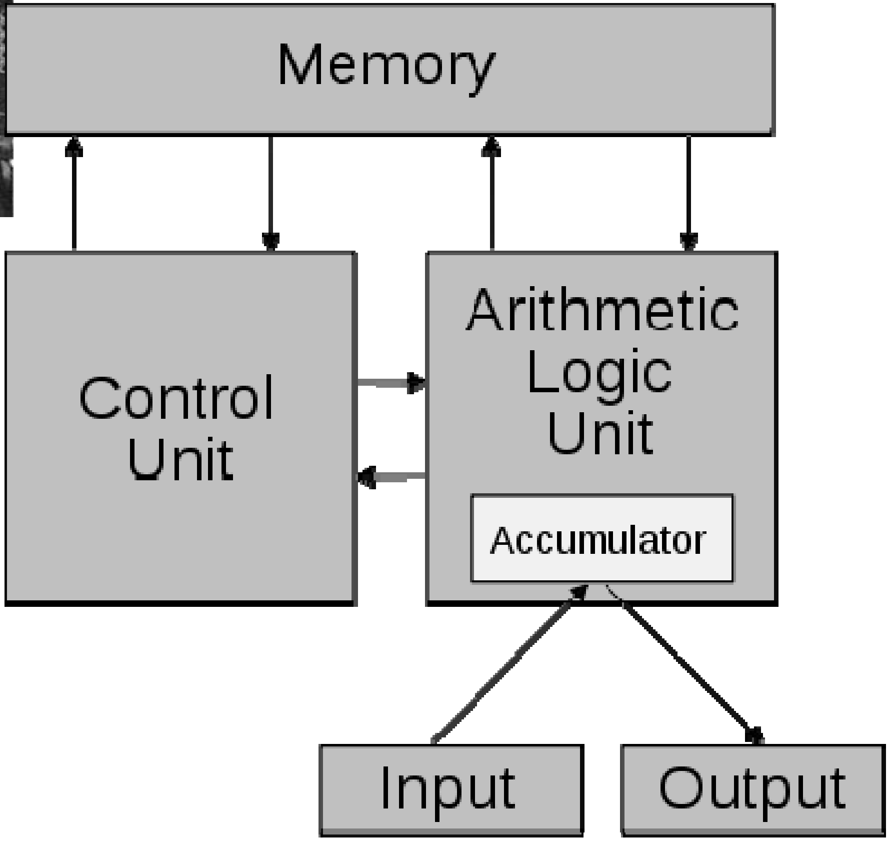
- Humans use languages
  - Alphabet
  - Words
  - Sentences
  - Punctuations, Numbers
  - All are symbolic
  - All these have to be agreed upon prior to communication

- Computer has a limited alphabet size

Brief History of CSE

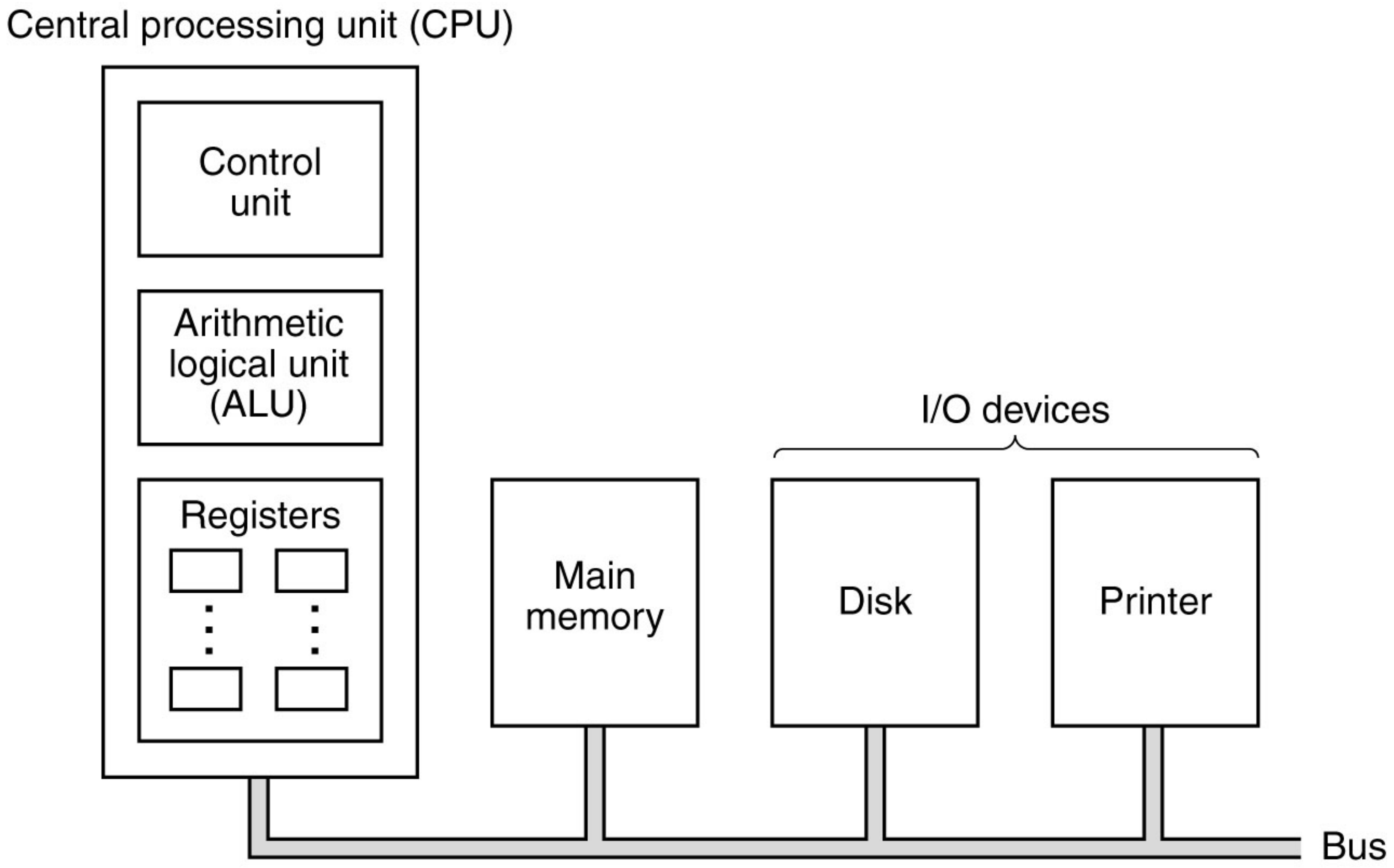
1800+	Charles Babbage	Difference Engine
1920	V. Bush	Analog Computers



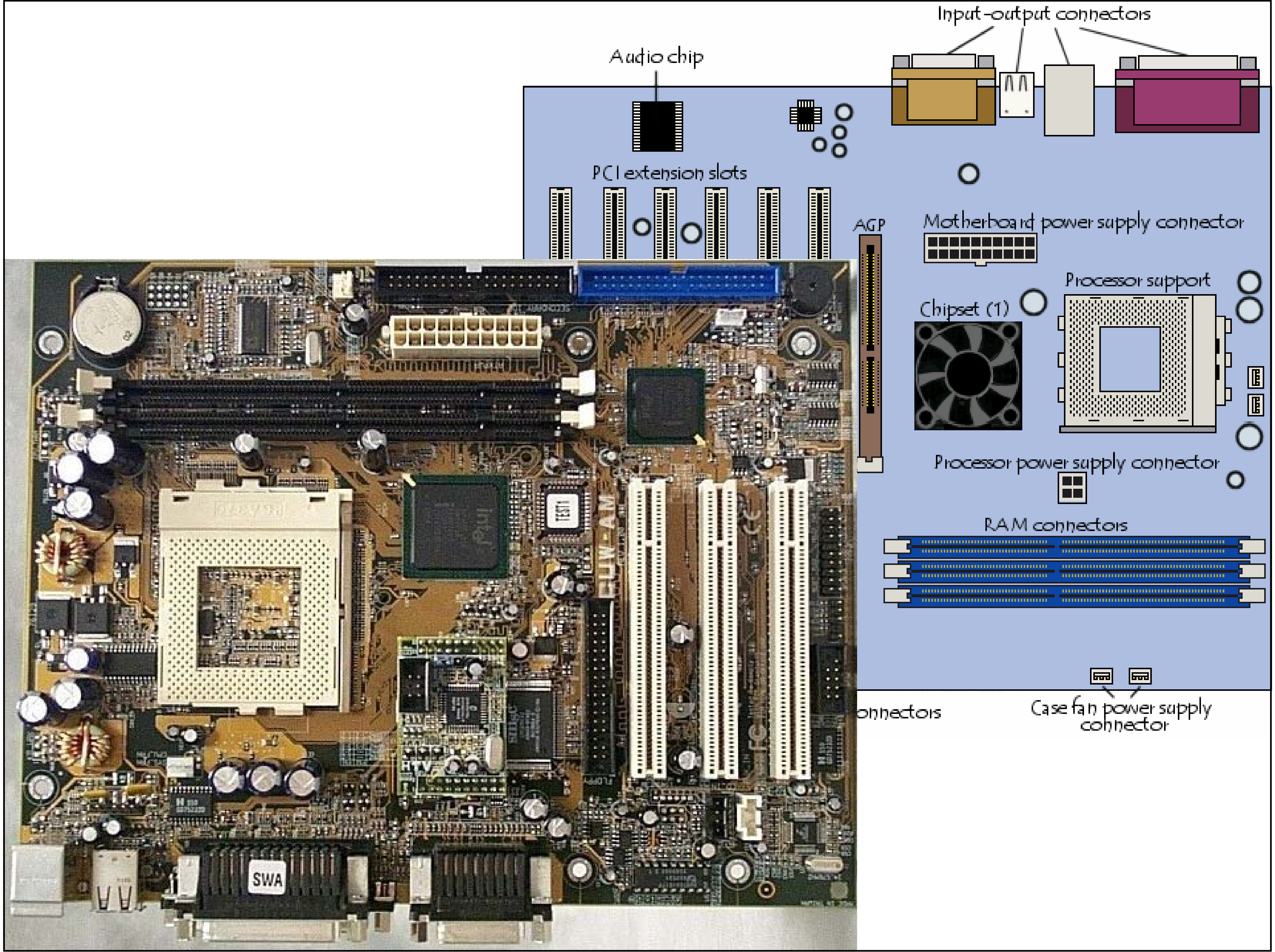


Design of the von Neumann architecture (1947)

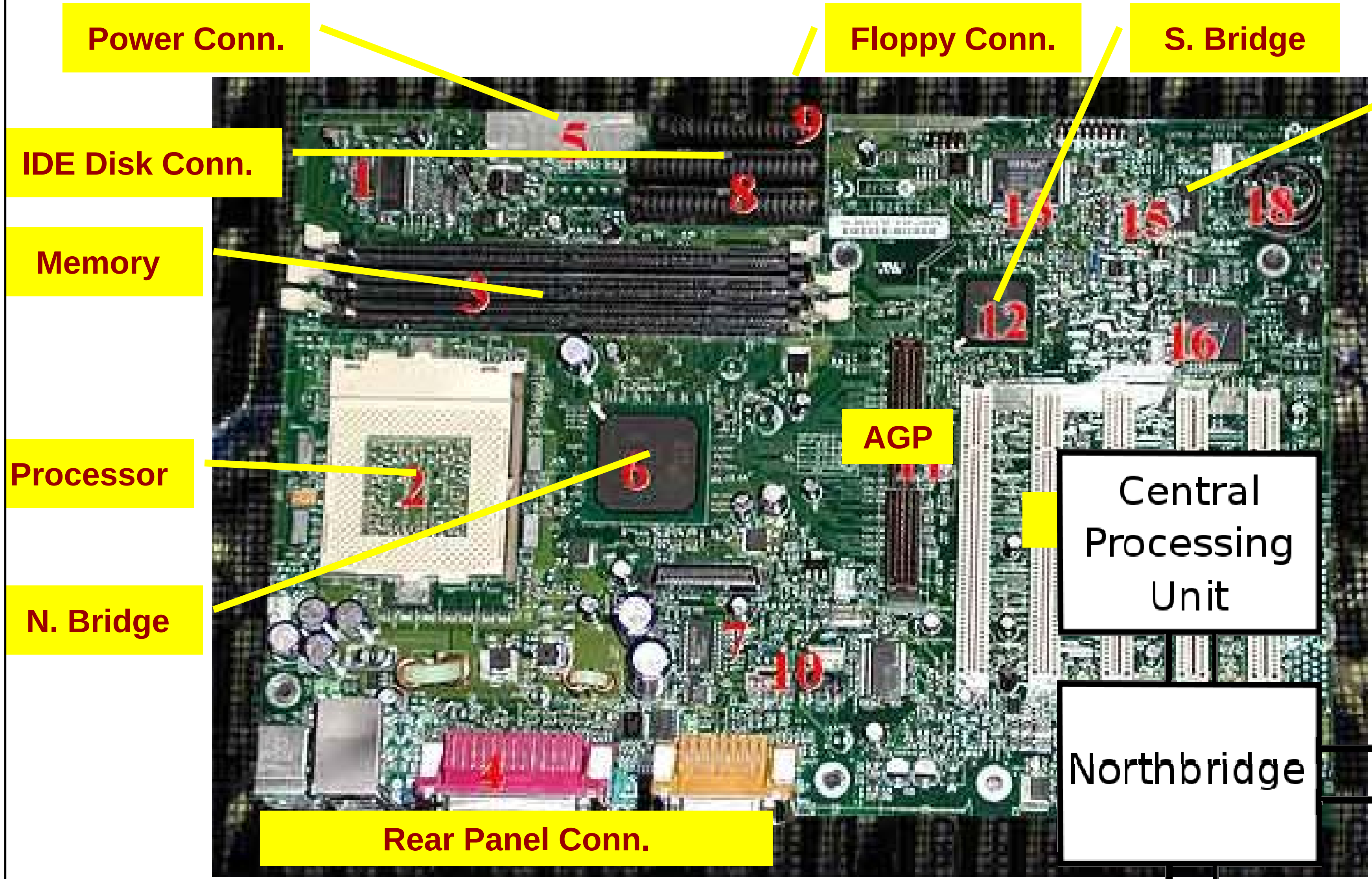
Central Processing Unit (CPU) based CO

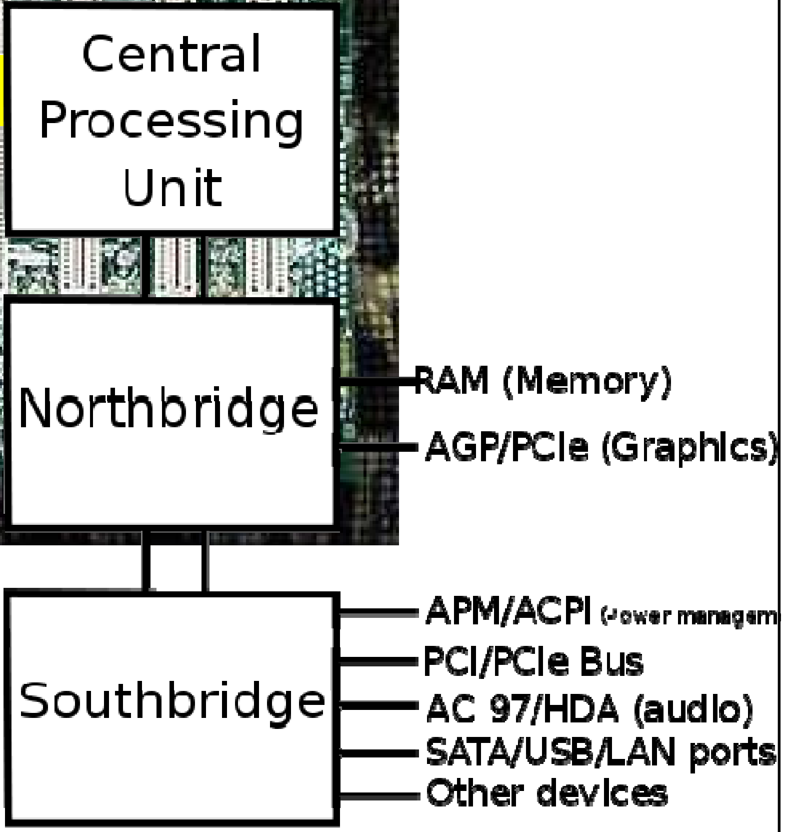


The organization of a simple computer with one CPU and two I/O devices


Input-output connectors
Audio chip
PCI extension slots
AGP
Motherboard power supply connector
Processor support
Chipset (1)
Processor power supply connector
RAM connectors
onnectors
Case fan power supply connector
SWA

Typical Motherboard (Pentium III)





AGP -
PCI -
IDE –
BIOS -

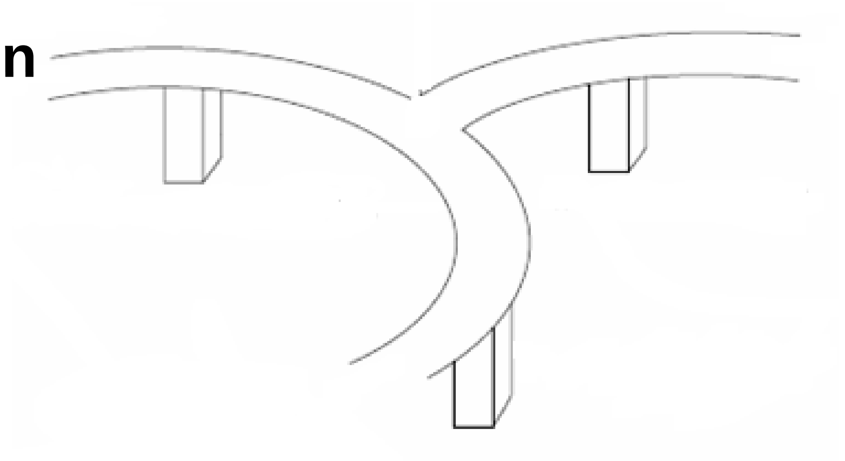
Three Pillars in Computing

Operating Systems

- OS
- Systems Design
- Performance Evaluation

Architecture

- Organization
- Architecture
- Network Design



Applications + Theory

- Algorithms, Compilers
- Software Engineering
- Databases, Artificial Intelligence etc
- Speech, Signal Processing

Four Areas in Computer Science & Engg.

Operating Systems

- OS
- Systems Design
- Performance Evaluation

Architecture

- Organization
- Architecture
- Network Design

Applications

- Software Engineering, Networks
- Artificial Intelligence – PR, ANN, SC
- Speech, Signal/Image/Video Processing
- Graphics, Multimedia
- Databases, VLSI, Robotics

Theory

- Algorithms, Compilers
- Cryptosystems
- Graph Theory
- Biocomputing

Hardware	Software	Theory	Applications
Switching Theory and Digital Design	Computational Engineering	Discrete Mathematics for Computer Science	Introduction to Database Systems, Data Mining
Computer Organization	Paradigms of Programming	Mathematical Concepts for Computer Science	*AI*, *PR*, *CV*, SPEECH, ANN, DVP, AVP, SC, RL, MBS, CBR, ML…
Computer System Design	*Operating Systems*	*Principles of Communication*	*Computer Graphics; Multimedia*
	Language Translators	*Data Structures and Algorithms*	VLSI, Digital System Testing
	Principles of Software Engineering	Languages, Machines and Computations	Computer Networks, Optical Networks, Cloud Computing
		Cyptography & N/W Sec.; Unconv. Models of Computing	Modern Compilers

$$\begin{tikzcd}
\underline{\text{Problem}} \arrow[d] \\
\text{Identify, Input, Output,} \\[-2em]
\text{Analytical Solution \& Modular Tasks} \arrow[d] \\
\text{Flowchart Design} \arrow[d] \\
\text{Convert to Algorithmic Steps} \arrow[d] \\
\text{Write program / code} \arrow[d] \\
\text{Debug / Compile} \arrow[d] \\
\underline{\text{Execute program}} \text{ to get results} \\[-2em]
\text{on Machine}
\end{tikzcd}$$

Typical Flow-diagram From Problem To Solution, Using Computers

Domain of CS(E)

- Till 1980s, mostly used for number crunching, data processing;

- PCs and internet, changed focus to coordination, communication and entertainment

- Knowledge involved efficient design and programming, information processing, theoretical studies and limitations.

- Usability, standardization, reliability, safety and transparency are other issues.

- Overlap with other (recent) fields:
  - DNA encoding (biology);
  - Cognitive models – psychology, neuro-biology etc.
  - Smaller and faster chips, faster communication media;
  - Biological memories;
  - Super-parallel computation;
  - Cryptography.

Issues and Concerns in the field

Given a (logically correct) algorithm, a digital computer is always available with enough (almost infinite) resources to accomplish the task. In fact, it should be possible then to execute a large set of such simulation processes.

Practitioner is expected to be specialized/skilled in four basic areas:

- Algorithmic thinking
- Data Structure and Representation
- Programming
- Design of systems

CS is a combination of all these four sub-areas/skills; not just 1 or 2 of them.

- **Algorithmic thinking**

  - Interpretation of the object under study, understand and formulate action as step-by-step procedures to give unambiguous results, when carried out by anyone, on any machine, anytime.

    Emphasizes standard procedure and scientific thinking to analyze and reproduce physical effects and observe them under various (valid) environments.

- **Data Structure and Representation**

  - Efficient way to store data, so that the problem could be solved (most) efficiently. Efficiency – retrieval, processing time and memory; inventing encoding phenomena for rich algorithms;

- **Programming**

  - Algorithms to programs/software, so that the machine can execute; requires knowledge of languages, tools, O/S etc.

- **Design of systems**

  - Practical considerations, Engg. Tradeoffs, integration of parts, meeting cost/time constraints, reliability and safety requirements etc.

Common misconceptions about computer science :

- Computer science is the study of computers
- Computer science is the study of how to write computer programs
- Computer science is the study of the uses and applications of computers and software

The general public sometimes confuses computer science with careers that deal with computers (such as information technology), or think that it relates to their own experience of computers, which typically involves activities such as gaming, web-browsing, and word-processing.

However, the focus of computer science is more on understanding the properties of the programs used to implement software such as games, banking and web-browsers, and using that understanding to create new programs or improve existing ones.

Standardized notions of computer science

– The study of algorithms
  - Formal and mathematical properties
  - Efficient design and analysis

- Hardware Design
  - Formal methods
  - Logical blocks to RTL

- Linguistic realizations and Applications
  - NLP, AI, ANN, CG, CN, DB…….

- Programming and Software Engg.
  - Coding
  - Testing, re-engineering,
  - Code maintenance

- **Computer scientist designs and develops algorithms to solve problems**

- **Operations involved in designing algorithms**

  - **Formal and mathematical properties**
    - **Studying the behavior of algorithms to determine whether they are correct and efficient**

  - **Hardware realizations**
    - **Designing and building computer systems that are able to execute algorithms**

- **Linguistic realizations**
  - **Designing programming languages and translating algorithms into these languages**
- **Applications**
  - **Identifying important problems and designing correct and efficient software packages to solve these problems**

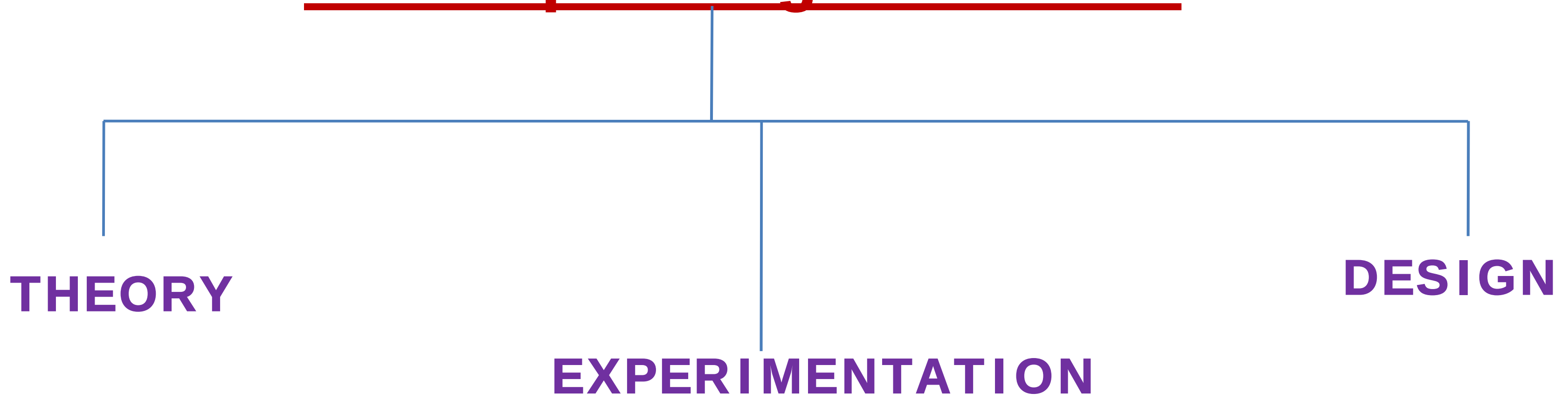
Relationship with Other Science disciplines (contd.)

- COMPUTATION (Computing or Computational science) has emerged as a third paradigm of science;

- Large problems solved in the fields of: Crystalline structure, quantum thermodynamics, Schroedinger Eqn., Navier-Stokes Eqn., in CFD, global climate modeling, weather forecasting, Stock market prediction / analysis, human genome sequencing, drug discovery and forensics.

- Other non (pure-) science disciplines which benefit – Library Science, Management Science, Economics, Medicine, Psychology and Cognitive science, Linguistics, Philosophy, Humanities and Social sciences (behavior, law, politics etc.)

Three paradigms in CSE



THEORY: Building conceptual frameworks and notations for understanding relationships among objects in a domain and the logical consequences of axioms and laws.

EXPERIMENTATION: Exploring models of systems and architectures within given application domains and testing whether those models can predict new behaviors accurately.

DESIGN: Constructing computer systems that support work in given organizations or application domains.

The three paradigms constantly interact in the work of computer scientists; the interaction brings in the vigor and richness of the field.

In areas of rapidly developing technology, such as databases, human interfaces, and Web-based systems, theoreticians aim mainly at bringing order into a rapidly accumulating mass of experience through broad conceptual frameworks, taxonomies, and analytic methods.

In mature areas such as computational complexity, algorithms, data structures, automata, formal languages, switching theory, graph theory, combinatorics, and formal languages, theoreticians focus on deeper, comprehensive analyses of phenomena for which formal models exist.

With a few notable exceptions including logic design, graphics, algorithm analysis, and compilers, theory has had limited impact on the complex problems of practical systems and applications.

Experimenters construct models of phenomena or of possible systems. Examples are measurement of programs and systems, validation of hypotheses, prototyping to extend abstractions to practice, logic simulation, simulations of systems and of physical processes, testing of protocols, system performance analysis, and comparisons of different architectures.

Designers are concerned with building systems that meet clear specifications and satisfy their customers - performance analysis, reliability, safety, security, and ethics.

Significant accomplishments include program development systems, simulators, microchip design systems, VLSI, CAD, CAM, graphics, databases, and supercomputers.

Unsuccessful design - unreliable, undependable, unsafe, too costly, too difficult to change, and too complex to understand.

Subareas of the Field

Computer science can be divided into a number of coherent subareas, each with substantial theoretical, experimental, and design issues,

Sub-areas Of CS:

1 Algorithms & Data Structures
2 Programming Languages
3 Architecture
4 Operating Systems and Networks
5 Software Engineering
6 Databases & Information Retrieval
7 Artificial Intelligence & Robotics
8 Graphics
9 Human Computer Interaction
10 Computational Science
11 Organizational Informatics
12 Bioinformatics

New and emerging fields of CS (few):

- DNA sequence encoding and analysis
- Modeling cognitive process of the human brain
- New materials for faster chip and communication
- Bio-sensors and bio-memories
- Quantum processes for super-computers and cryptography
- Obiquous computing (HCI)
- Large scale systems for complex analysis
- Pervasive and Cloud Computing
- Multi-core, pipelined, super-scalar architectures
- ???

Algorithms & Data Structures

Theory of algorithms encompasses computability theory, computational complexity theory, concurrency theory, probabilistic algorithm theory, database theory, randomized algorithms, pattern-matching algorithms, graph and network algorithms, algebraic algorithms, combinatorial optimization and cryptography.

It is supported by discrete mathematics (graph theory, recursive functions, recurrence relations, combinatorics), calculus, induction, predicate logic, temporal logic (a calculus of time dependent events), semantics, probability, and statistics.

Testing has yielded valuable characterizations (e.g. performance) of certain methods such as divide-and-conquer, greedy algorithms, dynamic programming, finite state machine interpreters, stack machine interpreters, heuristic searches, randomized algorithms and parallel and distributed algorithms.

Program libraries for theoretical formulations, e.g. mathematical software, searching, sorting, random number generation, textual pattern matching, hashing, graphs, trees, communication network protocols, distributed-data updates, semaphores, deadlock detectors, synchronizers, storage managers, lists, tables, and paging algorithms. Theoretical results translated into useful/practical systems: RSA public key cryptosystem, production-quality compilers, and VLSI circuit layout.

Programming languages

Medium for virtual machines that execute algorithms and with notations for algorithms and data; It also deals with efficient translations from high-level languages into machine codes.

Language involve data types, operations, control structures, mechanisms for introducing new types and operations. The sets of strings of symbols that are generated by such notations are called languages.

How are these abstractions implemented on computers?

What notation (syntax) can be used effectively and efficiently to specify what the computer should do?

How are functions (semantics) associated with language notations?

How can machines translate between languages?

The theory of programming languages studies models of machines that generate and translate languages and of grammars for expressing valid strings in the languages.

Examples include models of formal languages and automata, Turing machines, Post systems, lambda-calculus, pi-calculus, and propositional logic.

The theory deals with semantics, the study of the relationships between strings of the language and states of the underlying virtual machines.

It deals with types, which are classes of objects. Related mathematics is predicate logic, temporal logic, modern algebra and mathematical induction.

Examples: procedural languages (Cobol, Fortran, Algol, Pascal, Ada, and C),
- object-oriented languages (Clu, Smalltalk, C++, Eiffel, Java),
- functional languages (Lisp, ML, Haskell),
- dataflow languages (Sisal, Val, Id Nouveau),
- logic (Prolog),
- string (Snobol, Icon), and concurrency (Concurrent Pascal, Occam, SR, Modula-3).

Classification of languages based on their syntactic and semantic models, for example,
static typing, dynamic typing, functional, procedural, object-oriented, logic specification, message-passing, and dataflow.

Classification by application, for example, business data processing, simulation, list processing, and graphics.

Classification by functional structure, for example, procedure hierarchies, functional composition, abstract data types, and communicating sequential processes.

Abstract implementation models have been developed for each major type of language, including:
imperative, object-oriented, logic and constraint, concurrent, and distributed.

Run-time models have been implemented, for static and dynamic execution models, type checking, storage and register allocation, compilers, cross compilers, interpreters, analyzers that find parallelism in programs, and programming environments that aid users with tools for efficient syntactic and semantic error checking, profiling, debugging, and tracing.

A crowning achievement has been programs that take the description of a language and automatically produce a compiler that will translate programs in that language into machine code (examples include YACC and LEX in Unix environments);

Often designers create a mini-language and a parser.

Operating Systems and Networks

Control mechanisms that allow multiple resources to be efficiently coordinated in computations distributed over many computer systems connected by local and wide-area networks.

It has yielded well-known operating systems such as Unix, Multics, Mach, VMS, Mac-OS, OS/2, MS-DOS, and Windows NT.

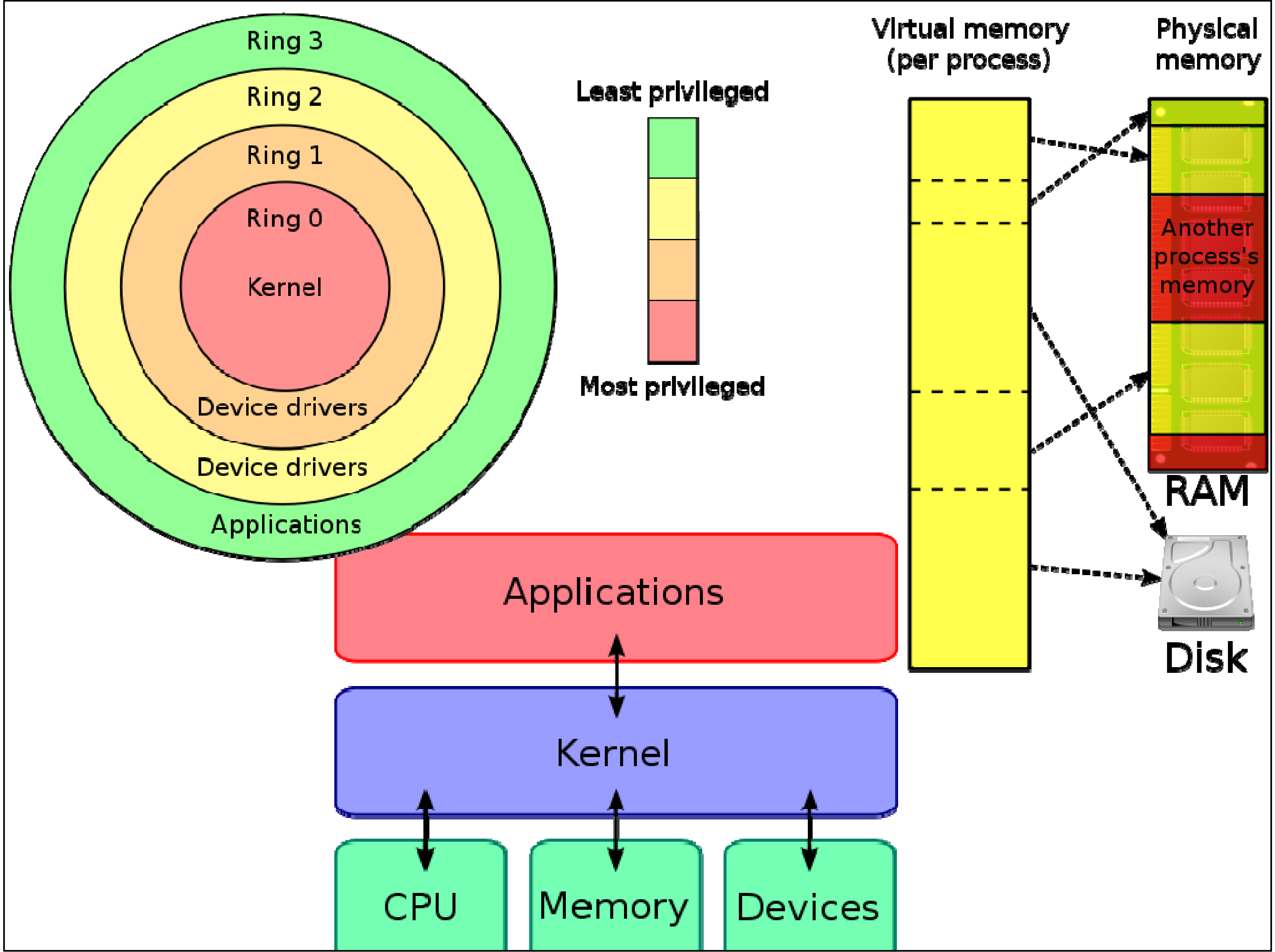
The field has yielded efficient standard methods including time sharing systems, automatic storage allocators, multilevel schedulers, memory managers, hierarchical file systems. It has produced standard utilities including editors, document formatters, compilers, linkers, and device drivers. It has produced standard approaches to files and file systems.

Also network architectures such as Ethernet, FDDI, token ring nets, SNA, and DECNET. It has produced protocol techniques embodied in the US Department of Defense protocol suite (TCP/IP), virtual circuit protocols, Internet, real time conferencing, and X.25. Considerable attention has been devoted to security and privacy issues in the Internet.

What are effective control strategies for job scheduling, memory management, communications, access to software resources, communication among concurrent tasks, reliability, and security?

Concepts taught: Abstraction and information-hiding principles; binding of user-defined objects to internal computational structures; process and thread management; memory management; job scheduling; secondary storage and file management; performance analysis; distributed computation; remote procedure calls; real-time systems; secure computing; and networking, including layered protocols, Internet protocols, naming, remote resource usage, help services, and local network routing protocols such as token passing and shared buses.

Also concurrency theory (synchronization, determinacy, and deadlocks); scheduling theory; program behavior and memory management theory; network flow theory; performance modeling and analysis. Supporting mathematics include bin packing, probability, queuing theory, queuing networks, communication and information theory, temporal logic, and cryptography.


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged
Virtual memory (per process)
Physical memory
Another process's memory
RAM
Disk
Applications
Kernel
CPU
Memory
Devices

ARCHITECTURES

The theory of architecture includes: digital logic, Boolean algebra, coding theory, and finite-state machine theory.

Supporting mathematics include statistics, probability, queuing theory, reliability theory, discrete mathematics, number theory, and arithmetic in different number systems.

Methods involve efficient design of CPU, memory, interface circuitry (I / O), etc. Multi-processor design, large storage etc. are also considered.

What about performance ?? How does one measure that ? Can you design digital sensors as used by humans ??

Difference between architecture and organization ??

From low level transistors → GATES → functional units → processing units → interconnections for a large system

<u>ARCHITECTS</u> in CSE use:

- finite state machines,
- general methods of synthesizing systems from basic components,
- models of circuits and finite state machines for computing arithmetic functions,
- models for data path and control structures, optimizing instruction sets for various models and workloads,
- hardware reliability, space, time,
- organization of machines for various computational models, and
- identification of “levels of abstraction” at which the design can be viewed -- e.g., configuration, program, instruction set, registers, and gates.

Designs involve - arithmetic function units, cache, von-Neumann machine, RISCs (Reduced Instruction Set Computers), CISCs (Complex Instruction Set Computers), error recovery, computer aided design (CAD); systems and logic simulations for the design of VLSI circuits, reduction programs for layout and fault diagnosis, silicon compilers (compilers that produce instructions for manufacturing a silicon chip).

Software Engineering

This area deals with the design of programs and large software systems that meet specifications and are safe, secure, reliable, and dependable.

Fundamental questions include:

What are the principles behind the development of programs and programming systems?

How does one prove that a program or system meets its specifications?

How does one develop specifications that do not omit important cases and can be analyzed for safety?

By what processes do software systems evolve through different generations?

By what processes can software be designed for understandability and modifiability?

What methods reduce complexity in designing very large software systems?

Three kinds of theory are used for software engineering:

- Program verification and proof (which treats forms of proofs and efficient algorithms for constructing them),

- Temporal logic (which is predicate calculus extended to allow statements about time-ordered events), and

- Reliability theory (which relates the overall failure probability of a system to the failure probabilities of its components over time).

Nine major categories of Models and measurements in software engineering :

(1) Specification of the input-output functions of a system: predicate transformers, programming calculi, abstract data types, object-oriented notations, and Floyd-Hoare axiomatic notations.

(2) The process by which a programmer constructs software: stepwise refinement, modular design, separate compilation, information-hiding, dataflow, software lifecycle models, layers of abstraction.

(3) Processes to develop software systems: specification-driven, evolutionary, iterative, formal, and cleanroom.

(4) Processes to assist programmers in avoiding or removing bugs in their programs: syntax-directed text editors, stepwise program execution tracers, programming environments, and software tools.

5) **Methods to improve the reliability of programs**: software fault tolerance, N-version programming, multiple-way redundancy, check pointing, recovery, information flow security, testing, and quality assurance.

(6) **Measurement and evaluation of programs**.

7) **Matching software systems with machine architectures** (the more specialized high-performance computers are not general-purpose).

(8) **Organizational strategies and project management**.

(9) **Software tools and environments**.

Software projects use version control systems to track versions of the modules of the emerging system; examples are RCS and SCCS.

Many syntax directed editors, line editors, screen editors, and programming environments have been implemented; examples are Turbo C and Turbo Pascal.

Database and Information Retrieval Systems

This area deals with the organization of large sets of persistent, shared data for efficient query and update.

The term database is used for a collection of records that can be updated and queried in various ways.

The term retrieval system is used for a collection of documents that will be searched and correlated; updates and modifications of documents are infrequent in a retrieval system.

Fundamental questions include:

What models are useful for representing data elements and their relationships?

How can basic operations such as store, locate, match, and retrieve be combined into effective transactions?

How can the user interact effectively with these transactions?

Fundamental questions (Contd.) –

How can high-level queries be translated into high performance programs?

What machine architectures lead to efficient retrieval and update?

How can data be protected against unauthorized access, disclosure, or destruction?

How can large databases be protected from inconsistencies due to simultaneous update?

How can protection and performance be achieved when the data are distributed among many machines?

How can text be indexed and classified for efficient retrieval?

A variety of theories have been devised and used to study and design database and information retrieval systems.

These include relational algebra and relational calculus, concurrency theory, serializable transactions, deadlock prevention, synchronized updates, statistical inference, rule-based inference, sorting, searching, indexing, performance analysis, and cryptography (ensuring privacy of information and authentication of persons who stored it or attempt to retrieve it).

Models and associated measurements:
(1) Data models for the logical structure of data and relations among data elements: object-based, record-based. (2) Storing files for fast retrieval, notably indexes, trees, inversions, and associative stores. (3) Access methods. (4) Query optimization. (5) Concurrency control and recovery. (6) Integrity (consistency) of a database under repeated updates, including concurrent updates of multiple copies. (7) Database security and privacy, including protection from unauthorized disclosure or alteration of data and minimizing statistical inference. (8) Virtual machines associated with query languages (e.g., text, spatial data, pictures, images, rule-sets). (9) Hypertext and multimedia integration of different kinds of data (text, video, graphics, voice).

Artificial Intelligence and Robotics