EEL-4736/5737
Principles of Computer System Design

Prof. Renato J. Figueiredo

LAR 336 (ACIS)

http://byron.acis.ufl.edu

renato@acis.ufl.edu

What is this class about?

- Computer science and engineering
  - Linguistics constructs describing computation (software)
  - Physical constructs for realizing computation (hardware)
- Computer systems encompass both
- Design of computer systems - “specialties”
  - Computer architecture
  - Operating systems
  - Database and transaction systems
  - Computer networks
  - Compilers
  - Reliability
  - …
- Rather than focus on a particular specialty
  - What are principles of system design that broadly apply across all of them?

What is this class about?

- Example – a Google search
  - You interact with multiple systems
    - Your computer is a system
      - Takes and processes your input, initiates a network connection
    - The Internet is a system
      - Routes your data to Google and back
    - www.google.com is a system
      - Parses your request, looks up database for hits, returns response

Design principles

- Various engineering principles recur in the design of these systems
  - Modularity –boundaries to manage complexity
    - Laptop
      - I/O subsystems – keyboard, network interface
      - Processor and memory subsystems
    - Internet
      - Switches, routers
      - Each a computer with its own subsystems
    - www.google.com
      - Load balancers, web servers, file/data bases

Design principles

- Abstraction – how modules interact; separation of interface from internals
  - Laptop
    - Processor – a billion-transistor system that is abstracted as a state machine indexed by a program counter with interface exposed by instruction set
  - Internet
    - A system that communicates data packets according to the IP protocol
  - www.google.com
    - A distributed system that takes as input a search query and returns as output a search result using the HTTP protocol

Design principles

- Layering – widely-used pattern of organizing modules
  - Laptop
    - O/S layer provides core services – schedule processor, allocate memory, deal with I/O
    - Application layer provides the Web browser the user interacts with
  - Internet
    - IP layer provides core functionality of packet transmission, on best-effort basis
    - TCP layer provides in-order delivery, retransmission, congestion control
  - www.google.com
    - Presentation layer, query processing layer, data layer

Design principles

- Hierarchy – widely-used design approach
  - Laptop
    - Memory hierarchy
  - Internet
    - Domain name servers
  - www.google.com
    - File system structure

Design principles

- Reliability – tolerance to faults
  - Laptop/desktop
    - ECC-protected memory
    - RAID disks
  - Internet
    - Error detection codes in packets
    - Redundant hardware in packet routers
  - www.google.com
    - Multiple web servers
    - Geographical distribution

Class overview

- Recurring themes
  - System design principles
    - Principles – not immutable laws, but guidelines that capture wisdom and experience from previous designs
  - Abstraction, layering and enforcing modularity
- Applications, case studies in computer systems design
  - Processors, operating systems, networks, client/server systems, storage

Career paths

- Systems design at the core of IT industry
  - Pervasive: IoT, mobile, cloud computing
  - HW/SW interfaces are increasingly blurred
  - Intel, Google, Amazon, Facebook, Microsoft, IBM, HP, VMware, Nvidia, Qualcomm, Arista, Audible, Citrix, …
- PoCSD provides you with foundational knowledge invaluable for the design of systems, across the HW/SW stack

Tentative outline

- Introduction to computer systems
  - Components, interfaces, environments
  - Coping with complexity
- Elements of computer system organization
  - Modularity, abstraction, naming, layers
- Naming systems
- Modularity via clients/servers, virtualization
- Performance – resource scheduling and management

Tentative outline (cont'd)

- Network as a system and as a system component
- Fault tolerance
- Atomicity
- (Consistency)

Course overview

- Who is this class intended for?
  - Senior undergraduates broadly interested in computer systems
    - Expand upon basic knowledge of hardware design and software programming/algorithms
  - Graduate students who are interested in pursuing in-depth topics in computer systems.
- This class builds a foundation helpful in careers where you'll design hw/sw
  - And for a variety of advanced computer engineering classes in our program:
  - Virtual computers, distributed systems, software-defined systems, fault-tolerant computer architecture

Example - project

- Design a distributed storage system
  - Tolerates server failures, or
  - Supports multiple concurrent clients
- Progressively more complex:
  - Where data is initially stored in local memory
  - Then on a server
  - Then on distributed servers
- Design principles:
  - naming; layering; modularity through client/server; networks; performance; fault tolerance

Administrative matters

- Textbook:
  - J. Saltzer, M. Frans Kaashoek
    - "Principles of Computer System Design"
    - Morgan Kaufmann, 2009
- Primary Web site:
  - Canvas (UF's course management system)
- Instructor and assistants
  - Renato Figueiredo, LAR 336 (ACIS Lab)
    - http://byron.acis.ufl.edu
    - Office hours posted on my Google calendar (linked from my Web site and Canvas)
  - TA: Mr. Cody Rigby
    - Hours will be posted on Canvas

Academic honesty policies

- An “F” grade will be given for academic dishonesty
  - Cheating, plagiarism, non-independent work...
- All work submitted in this course must be your own and produced exclusively for this course. The use of sources (ideas, quotations, paraphrases) must be properly acknowledged and documented. Violations will be noted on student disciplinary records.
- Each student must read the UF student honor code at:
  - https://www.dso.ufl.edu/sccr/process/student-conduct-honor-code/
  - Read also:
  - http://www.acm.org/about/code-of-ethics
  - http://www.ieee.org/portal/pages/iportals/aboutus/ethics/code.html

Grading

- Assignments and approximate weights
  - Homeworks and project: 35%
    - Include substantial programming
  - Two midterms, final exam: 65%
  - Canvas quizzes:
    - Will not count towards your final grade
    - A mechanism to help you practice, keep up with material, and self-assess your course knowledge

Disclaimer

- The instructor reserves the right to change the contents, outline, and grading scheme of the course for the good of the students' learning experience and success of the course

Pre-requisites

- Requirements:
  - Digital design (e.g. EEL4712)
    - You need a good understanding of how hardware subsystems are designed, such as a simple controller
  - Data structures, programming (e.g. EEL-4834)
    - You must have a good understanding of how to program applications using a high-level language
- Desirable:
  - Basic understanding of operating systems and computer organization
  - Experience with Linux
  - Experience with Python
    - Will be important for assignments

Poll

- Digital design class?
- Data structures and programming class?
- Computer architecture class?
- Operating systems class?
- Unix/Linux?
- C?
- Python or other scripting language?

Project and programming

- This is a computer systems design class
  - And you will design a (software) system
  - Help solidify concepts learned in class
  - Will be challenging – and rewarding
- Development environment
  - Most will be done on your own laptop
  - Virtual machine with Linux
  - Programming language – Python
    - See Homework #1

Programming/assignments

- We will focus on the design of software systems of progressive complexity
  - Covering various aspects of the material discussed in class
- Why Python?
  - It is simple to learn and use
  - Ideal for rapid prototyping
  - It is widely used in practice for the programming of various systems
    - YouTube, Google, ..

Programming/assignments

- Not familiar with Unix/Linux? Python?
  - I don't want to discourage you, but you *will* need to invest extra time to ramp up
- Quotes:
  - "Now working in industry I feel how great it (PoCSD) was"
  - "The assignments, quizzes and exams are very challenging."
  - "I feel this is a challenging but rewarding course."
  - "The programming assignments were challenging."
  - "Brilliant. Made me learn Python."
  - "This course may be one of the most useful courses in the department."
  - "Course work is excellent. Fundamental course for all the computer engineers as it covers all the aspects of computer designing."

Textbook and reading

- Come prepared to class – read in advance
  - I will provide pointers to reading assignments in advance of lectures
  - Topics that I have not have had time to cover in lecture – still assume you have read it
  - Reading may be complemented by technical papers
- Start with Chapters 1 and 2

Homework #1

- Posted on class web site (Canvas)
- Goals:
  - Get prepared with your basic environment to use in class
  - Get acquainted with the core code we'll build upon throughout the semester
  - Self-assessment of your ability to follow the course

Systems - defined

- What is a system?
  - Webster unabridged: "*A complex unity formed of many often diverse parts subject to a common plan or serving a common purpose*"
  - Textbook: "*A set of interconnected components that has an expected behavior at the interface with its environment*"
    - *Computer* or *information system:* system intended to store, process or communicate information – predominantly digital – under automatic control

Systems - idea

- Divide and conquer
  - Things under discussion are part of the *system*
  - Things not under discussion are part of the *environment*
- Interactions between system and environment is the *interface*
- What constitutes the system (components, interactions) depends on one's point of view
  - *Purpose* and *granularity*
    - Example of jet airplane as a system on textbook

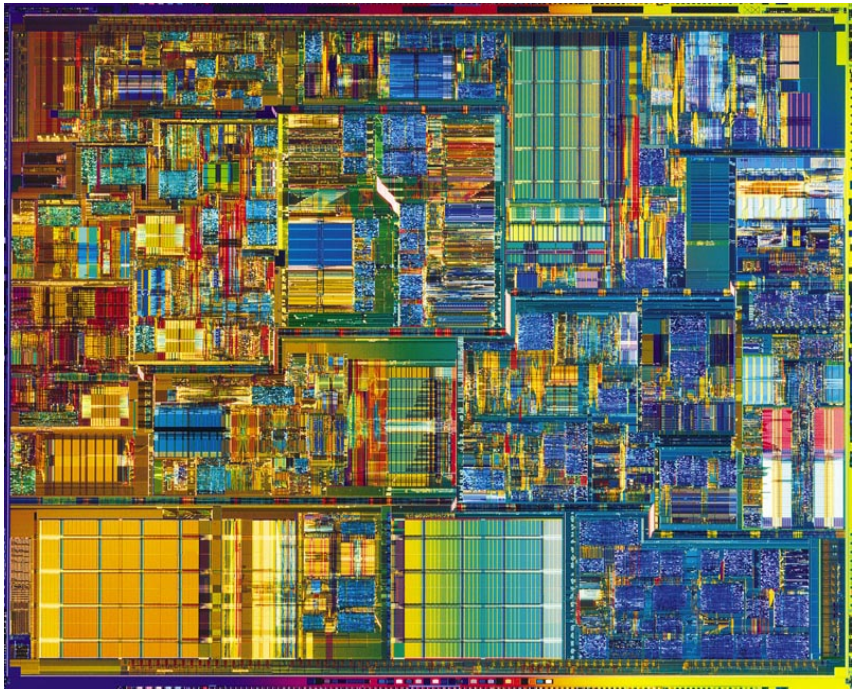
Systems and subsystems

- When a system in one context is a component in another, it is usually referred to as a *subsystem*
- Decomposition of systems is recursive
- Decomposition/composition is an integral part of the design process of non-trivial systems

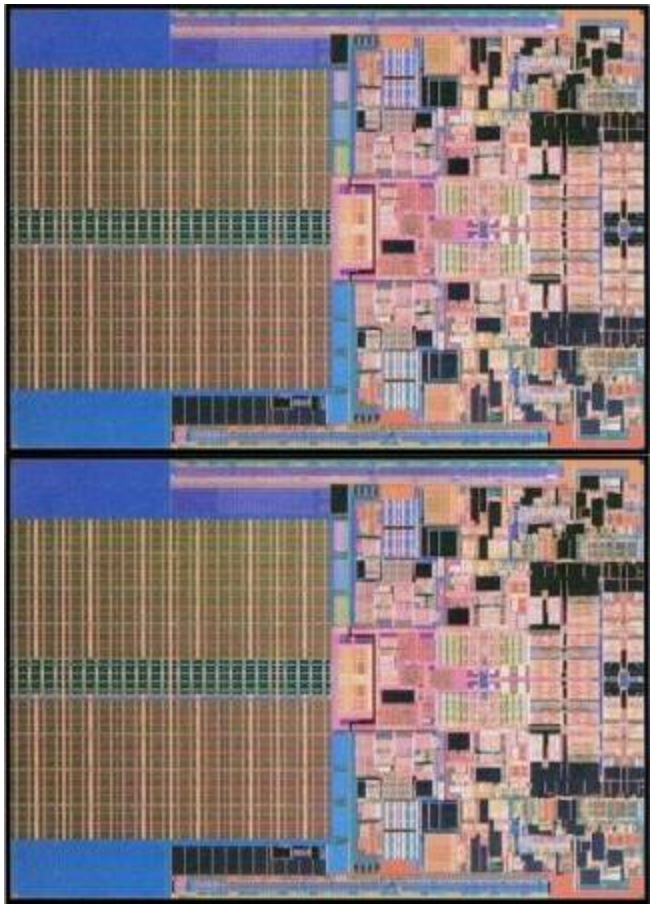
Complexity

- Complex: difficult to understand
  - A key underpinning aspect of design of computer systems is managing complexity
- Difficult to measure; signs of complexity:
  - Large number of components
  - Large number of interconnections
  - Many irregularities
  - A long description
  - A team of designers, implementers, or maintainers

Complexity



Pentium 4
(40M transistors)



Yorkfield
(2x2-core dies,
820M transistors)

Complexity

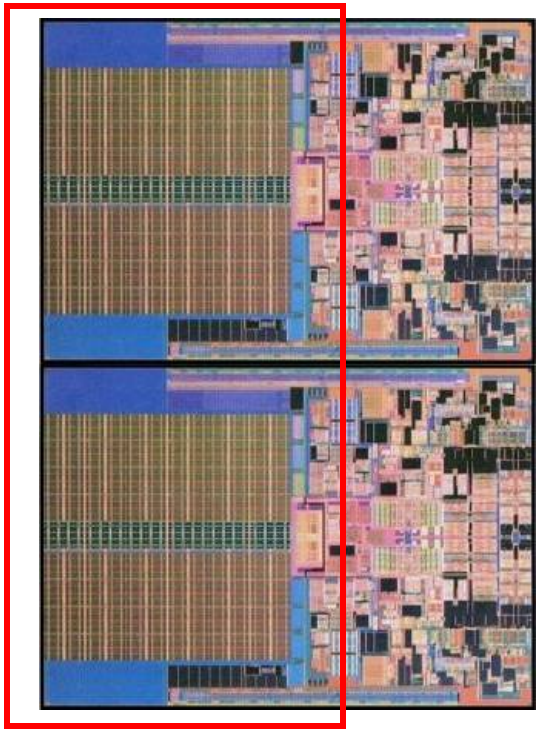
- main(int c,char**v){return!m(v[1],v[2]);}m(char*s,char*t){return*t-42?*s?63==*t|*s==*t&&m(s+1,t+1):!*t:m(s,t+1)||*s&&m(s+1,t);}
  - http://en.wikipedia.org/wiki/One-liner_program
  - Pattern matching for “*” and “?”

- Is the code modular? Well-documented?

Sources of complexity

- Maintaining high utilization
  - Requirement – desire for high performance or efficiency
  - Increases complexity in scheduling, control

Caches and controllers are needed for high performance memory access



Common problems

- Propagation of effects
  - Changes that are small (locally) can propagate effects to many components
  - See textbook's example – changing the tire size of mass-produced car
- *There are no small changes in a large system*

Coping with Complexity

- How to build useful computer systems in the face of sources of complexity?
  - Will study principles, case studies; systematic approaches; insights into how designers create real systems
    - O/S kernel, file system; client/server; virtualization; networks, fault tolerance
  - Modularity, Abstraction, Layering, Hierarchy

Modularity

- A divide-and-conquer approach
- Both large number of components and interconnections are sources of complexity
  - Modularity seeks to reduce interactions
  - Can consider interactions *within* a module without simultaneously thinking about the components *inside other modules*

Modularity

- A key benefit of modularity is that modules can be designed, analyzed replaced, independently
  - Provided their behavior and interfaces remain the same
- *It is easier to change a module than to change the modularity*

Abstraction

- For modularity to be effective
  - There should be little to no propagation of effects among modules
- There are many ways to divide a system
  - There are ways that prove to be better than others (quantitatively or qualitatively)
  - Better divisions are characterized by fewer module interactions and less propagation of effects
- *Abstraction*:
  - separation of interface from internals
  - separation of specification from implementation
- Nearly always associated with modularity

Example abstractions

- State in sequential logic abstracted as registers
- UNIX device abstracted as a file
- A software object

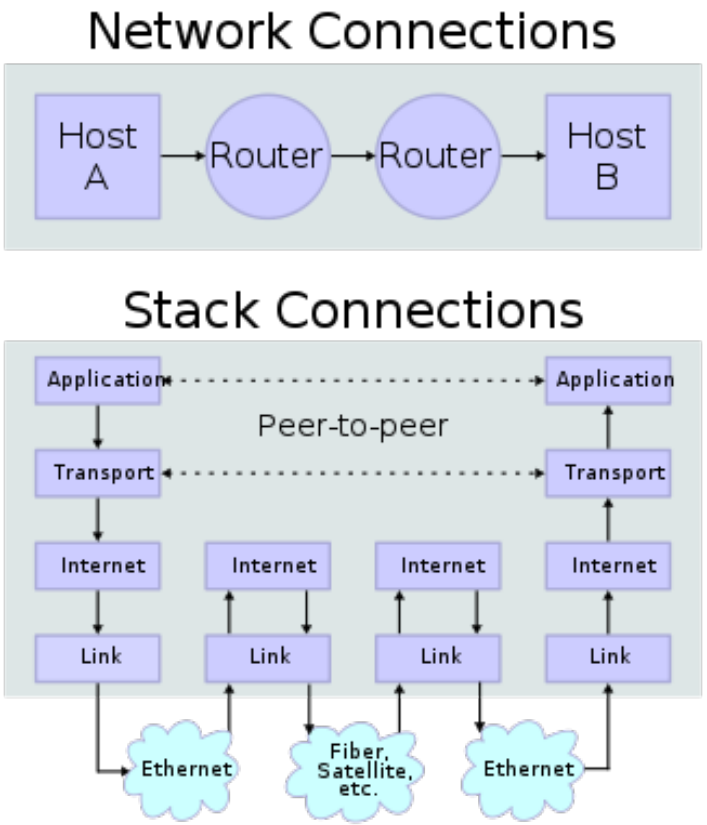
Abstraction and modularity

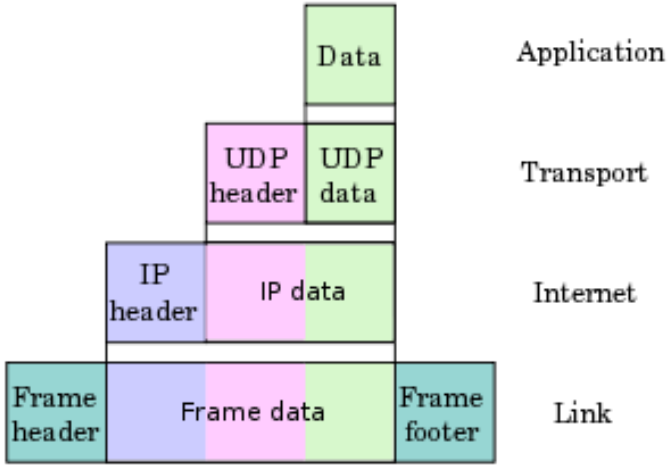
- Abstraction and modularity intended to minimize interconnections
  - Implementation errors or well-meaning design attempts to bypass module boundaries may defeat this purpose
  - Software, in particular
- Techniques that *enforce* modular abstractions are important
  - Client/service; virtualization
  - Containment of faults for fault-tolerance

Layering

- A particular approach of module organization
  - Build a complete set of mechanisms as a layer (lower layer)
  - Use lower layer to build different complete set of mechanisms (upper layer)
  - A module in a layer only interacts with peers in the same layer, or with adjacent (lower or upper) layer

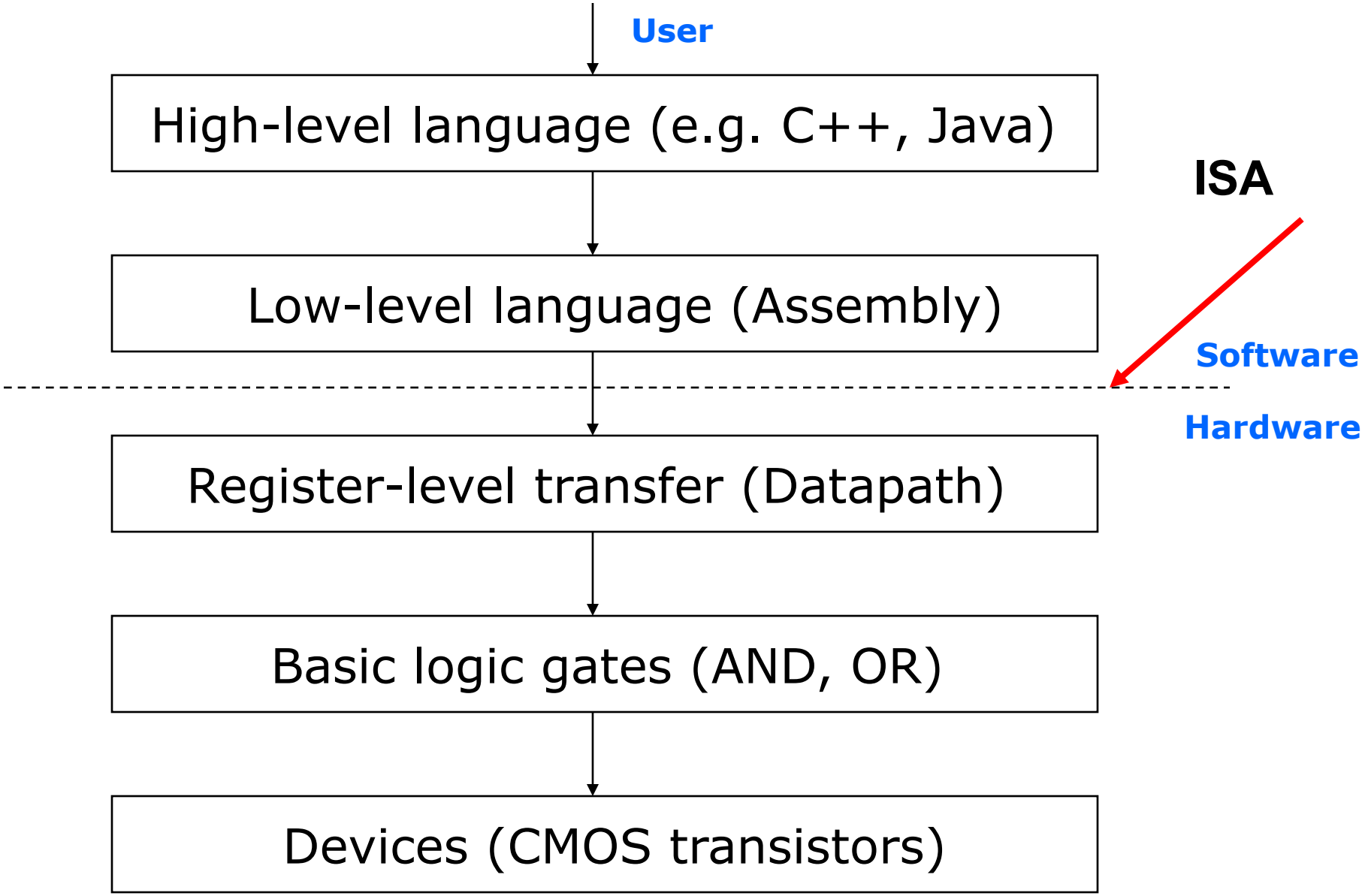
Layering





Source: http://en.wikipedia.org/wiki/TCP/IP_model

Abstraction layers



Hierarchy

- Another well-used specialized approach to organize modules
  - Small set of modules assembled in a subsystem with a well-defined interface
  - Assemble a small set of subsystems to produce a larger subsystem
  - Continue until the final system has been constructed from a small number of (large) subsystems
- E.g. memory hierarchy

How to apply?

- What is the right (abstraction, modularity, layering, hierarchy) from a vast number of plausible alternatives?
  - Only real guidance comes from experience from the design of previous systems
  - In this class we will study many cases, and you will work on a project to help build this experience

Naming

- Modularity, abstraction, layering, hierarchy allow decomposition of the design into manageable modules
- *Names* are needed to make connections among modules
  - Wires/buses in a hardware schematic
  - Register identifiers in a processor
  - Memory addresses in DRAM
  - Sector numbers in a disk
  - Object names in a program
  - Server names on the Internet

Next

- Elements of computer system organization
  - Fundamental abstractions
  - Naming in computer systems
  - Names and layers
  - Example: UNIX file system
- Reading: Chapter 2