University of Southern California

Viterbi School of Engineering

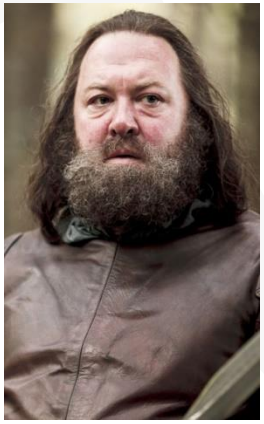
EE355
Software Design for Electrical Engineers

EE599 – Special Topics:
Software Design and Optimization for Electrical Engineers
Introduction

Robb Stark
King in the North

Aria Stark

Shahin Nazarian
SW for EE
in the South



Robert Baratheon

Objectives

- Enter the workplace with a literacy and fluency of the kinds of software development tools and processes used in industry
- Analyze, evaluate, and prove your ideas and concepts by creating software models
- Utilize available (open-source) software and apply to new programs
- Have the software and discrete math skills needed to take pertinent CS courses

Objectives (cont.)

- Understand the principles of program execution on modern computing hardware including memory management, subroutine calls, and efficiency issues
- Model a given problem as a set of objects and define their interaction
- Use object-oriented features of C++ to solve problems
- Utilize appropriate data structures from the C++ Standard Template Library (STL)

Objectives (cont.)

- Understand principles of discrete mathematics related to sets, graph theory, and certain numerical methods
- Use software development tools including IDEs, debuggers, makefiles, and source code management (version control) tools
- Perform a non-trivial software design project from conception to implementation
- Research?

Course Advice

- Catch the wave!
  - Overestimate the time you will need and your ability to get your work done
  - Start each assignment within 1-2 days of receiving it
  - Don't let shame or embarrassment keep you from the help you need
- Programming requires practice
- Expectations
  - Treat EE355 right!
    - Understand its goals, and what it takes to achieve them
    - Attendance, participation, asking questions, academic integrity, take an interest
- Let's make this fun

What Other Students Say

Is this class useful?

- This class was one of the most useful courses that I took in my undergrad. The reality is that as an electrical engineer you must be able to interface between hardware and software. Even if you don't intend to work in software development, you will almost certainly be asked to write or critique code at some point. I have used the skills I learned in this class during all of my Masters classes and at work at Boeing. Lately I've also been doing a lot of job interviews and I've seen that concepts learned in this class are very common in interview questions even in jobs outside of software development. Learning the material in this class well will set you up for success when you are looking for a job and give you a strong foundation for Masters level coursework or entry into industry. Plus it's fun!

What Other Students Say (cont.)

What advice would you give EE 355 students?

- Don't leave the programming assignments to the last few days before they are due. Even if you manage to finish them on time, you won't learn the material as well. If I could go back I would take more time with my work and try to think more critically about the assignments while I did them. Try to think about why one algorithm choice is better than another and don't just compile and then fix errors by trial and error. Make sure you understand why you make each choice that you do during your assignments because in a job interview people will ask you to justify your choices and you won't get a compiler to tell you what you need to fix

What Other Students Say (cont.)

Is this class useful?

- EE 355 was perhaps the most useful class I took at USC! I had no previous C++ experience, and in this class I learned the basics all the way up through more complex data structures and algorithms, and I got a job mainly because of the knowledge gained in the class

What advice would you give EE 355 students?

- I would advise anyone who takes EE 355 to not underestimate how much time the Programming Assignments will take you and start them with ample time to be able to finish them!

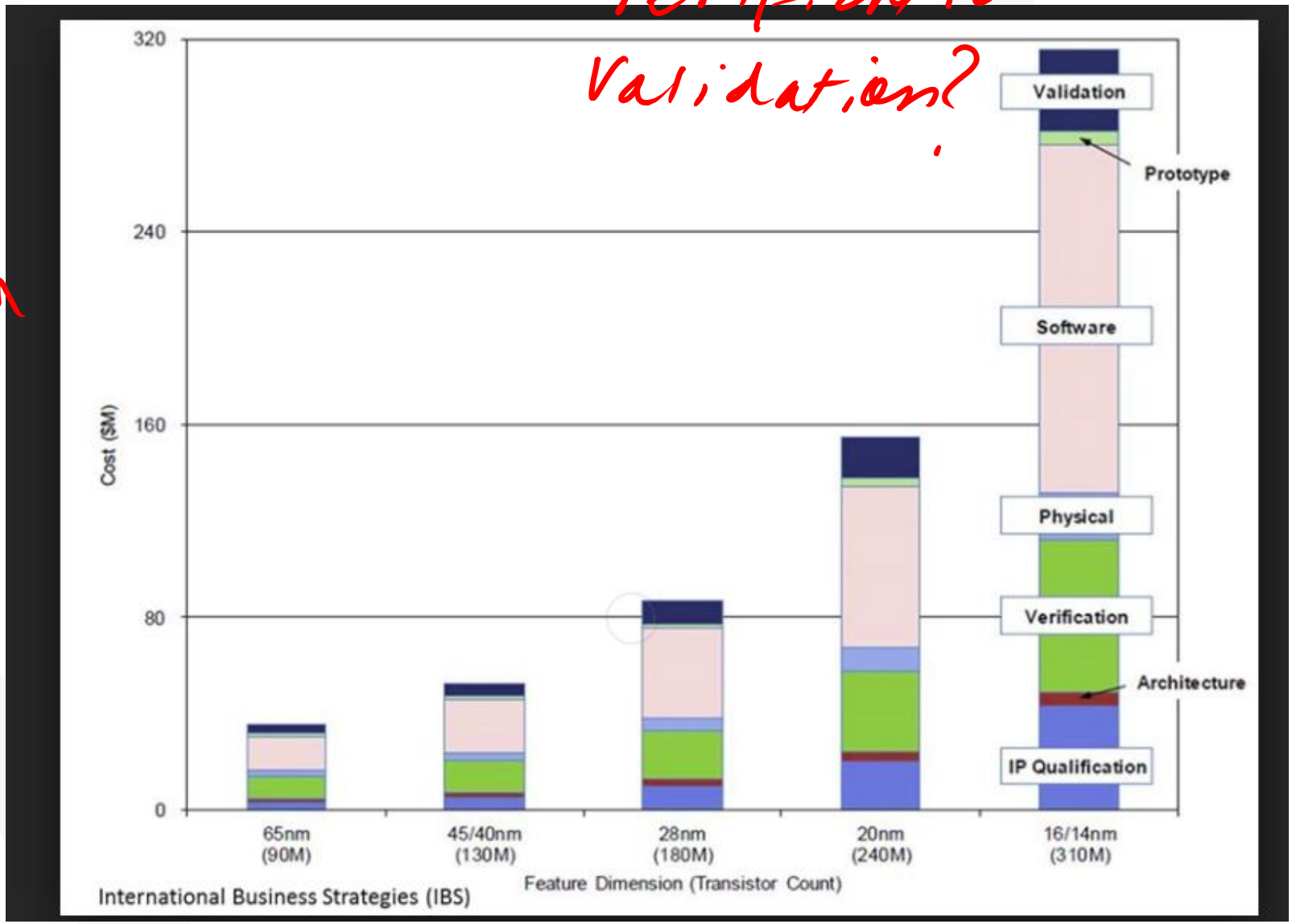
What Other Students Say (cont.)

Is this class useful?

- I just wanted to thank you for teaching us about machine learning and assigning us Lab 11. I had an interview with XX yesterday and sure enough, they asked me about machine learning (even though it wasn't for a programming position). I talked about what I did for Lab 11 and he seemed pretty impressed, not only because of the program but also the "data-finding" component. I'm sure you probably already are, but I'd highly recommend assigning that lab again to future classes.

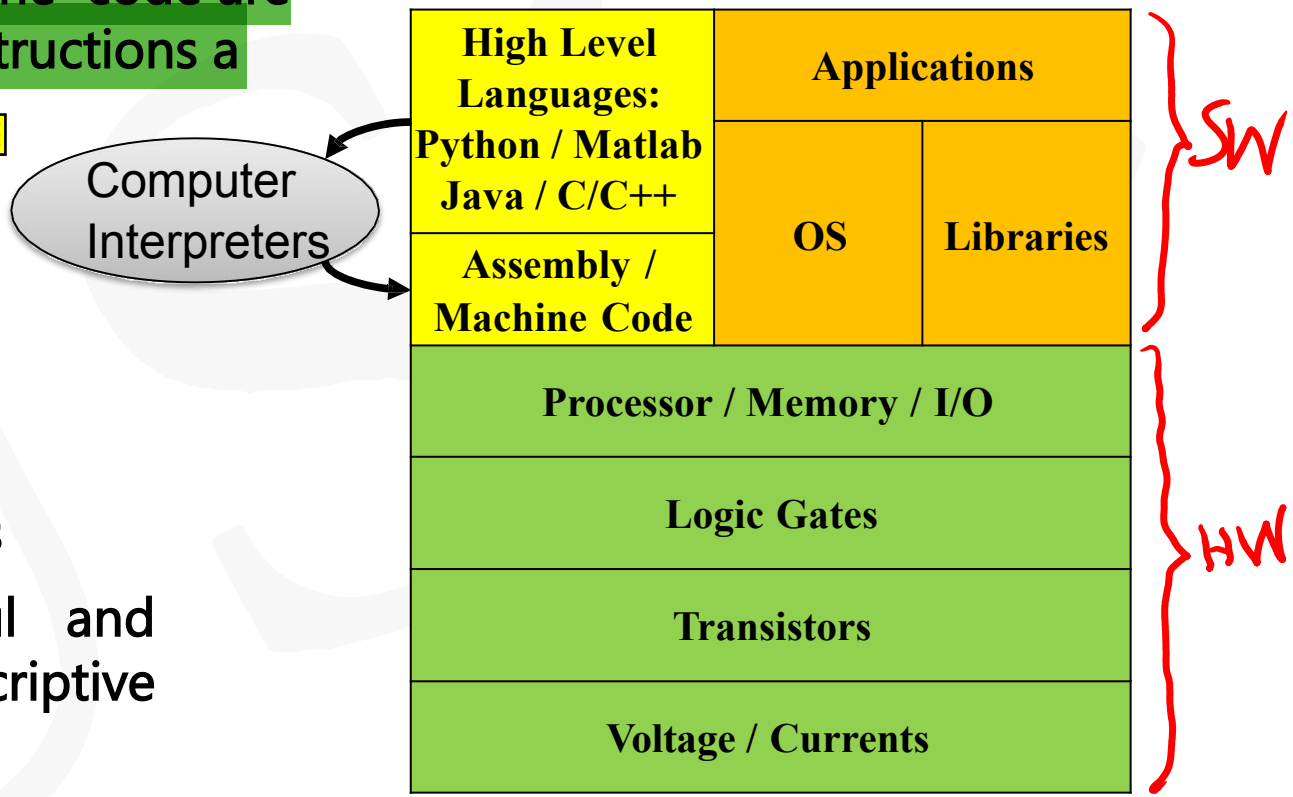
Question: which one: sw, hw, architecture, verification, validation?

SoC Design Cost

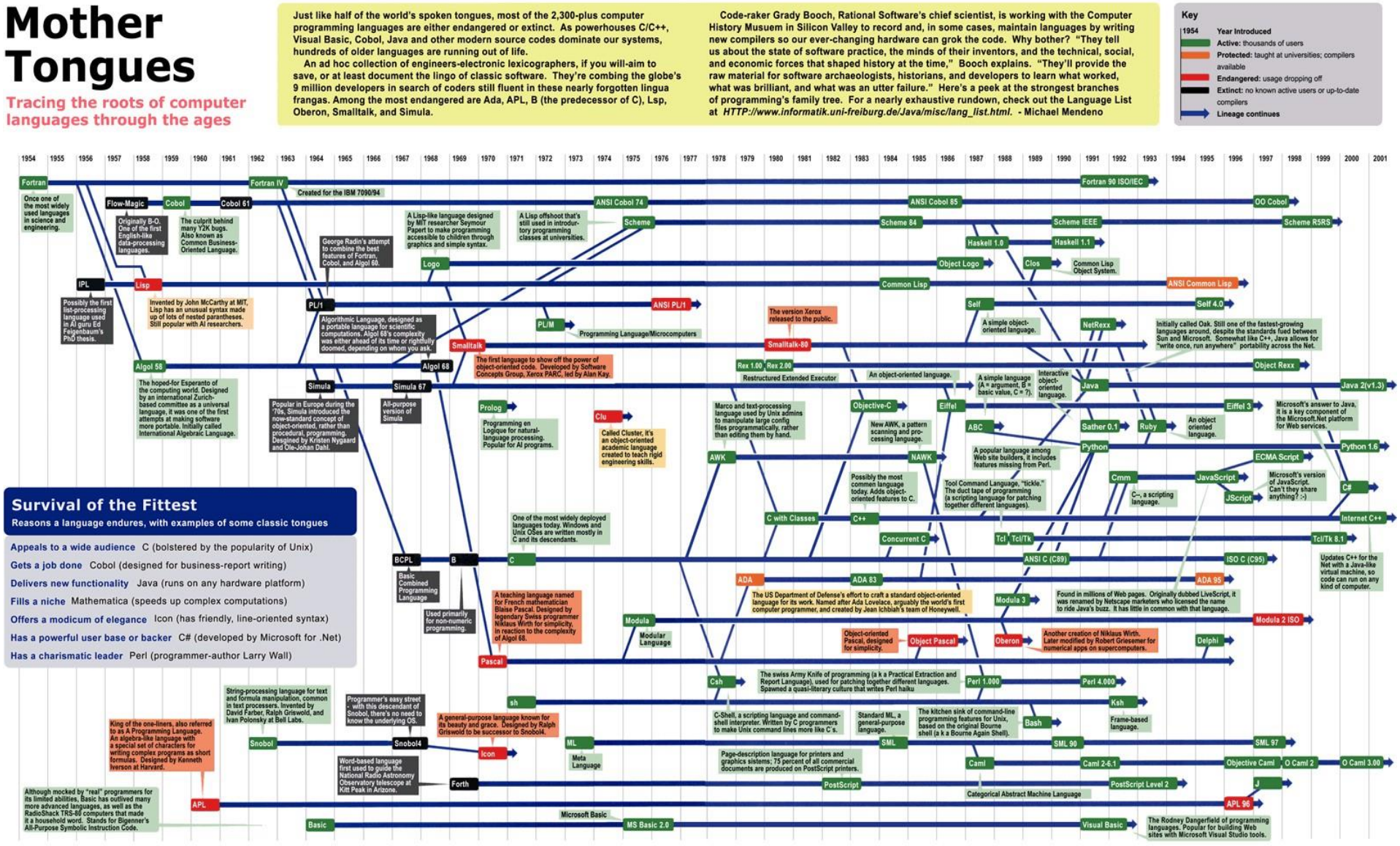


Computer Abstractions

- Computer systems can be viewed as a layered stack of abstractions from HW to SW
- Assembly and machine code are the fundamental instructions a CPU can execute
- Processor specific
- Too many details for programmer to track
- High level languages
  - More powerful and succinct descriptive abilities



Optional: High Level Languages



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

 http://www.digibarn.com/collections/posters/tongues/ComputerLanguagesChart-med.png

Compiled vs. Interpreted Languages

Compiled (C/C++)

- Requires code to be converted to the native machine language of the processor in the target system before it can be run
- Analogy: Taking a speech and translating it to a different language ahead of time so the speaker can just read it
- Faster
- Often allows programmer closer access to the hardware

Interpreted (Matlab / Python)

- Requires an interpreter program on the target system that will interpret the program source code command by command to the native system at run-time
- Analogy: Speaking through an interpreter where the speaker waits while the translator interprets
- Better portability to different systems
- Often abstracts HW functionality with built-in libraries (networking, file I/O, math routines, etc.)

https://www.youtube.com/watch?v=qaj7nO1HUqA

Why C++?

- C/C++ is close to the actual hardware
  - Makes it fast
  - Makes it flexible (Near direct control of the HW)
  - Makes it dangerous (Near direct control of the HW)
  - C became popular because it was the language used to implement Unix & then Linux (all Unix/Linux distributions came with a C / C++ compiler)
- C/C++ is ubiquitous
  - Used everywhere, even to implement other programming languages (i.e., Python, Matlab, etc.)
- Wide availability of compilers and development environments

Why Python

- Simple
- Broad
- Packages (libraries, data structures)

Remote Access, Editors, Compilers, IDE's

PROGRAMMING ENVIRONMENT

Development Environment

- To write and run software programs in C you will need
  - A text editor to write the code
  - A 'C/C++' compiler, linker, etc. to convert the code to a program
- Different OS and platform combinations have different compilers and produce "different version" of a program that can only run on that given OS/platform
  - GNU gcc/g++ (Unix/Linux/Mac)
  - Cygwin gcc/g++ (Windows)
  - Mac XCode (Mac only)
  - MS Visual Studio (Windows only)
  - CodeBlocks (cross-platform)

Ubuntu VM Image

- We are providing a virtual machine appliance (An Ubuntu Linux image that you can run on your Mac or Windows PC)
  - All instructions at the course website

Development Environment

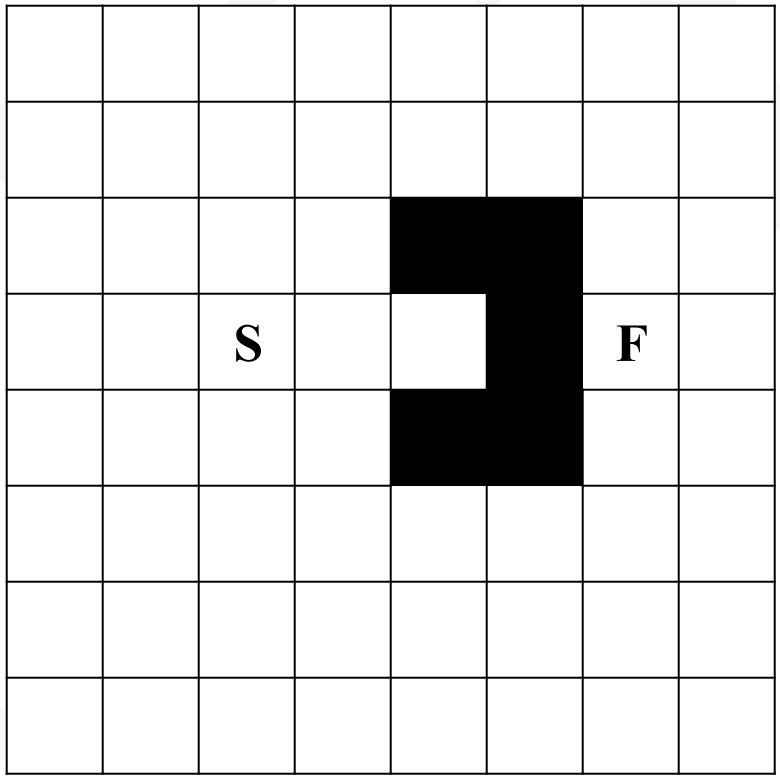
- Different OS and platform combinations have different compilers and produce"different versions" of a program that can only run on that given OS/platform
  - All assignments will be run/graded on our CS/EE Linux virtual machine or on our own Linux server
- You may develop code on any system but we recommend the VM we provide
  - Remember it must RUN CORRECTLY without any issues on the Linux VM thatwe provide

THINK LIKE A COMPUTER

Burst Length	Starting Column Address			Order of Accesses Within a Burst	
				Type = Sequential	Type = Interleaved
2	–	–	**A0**	–	–
	–	–	0	0-1	0-1
	–	–	1	1-0	1-0
4	–	**A1**	**A0**	–	–
	–	0	0	0-1-2-3	0-1-2-3
	–	0	1	1-2-3-0	1-0-3-2
	–	1	0	2-3-0-1	2-3-0-1
	–	1	1	3-0-1-2	3-2-1-0
8	**A2**	**A1**	**A0**	–	–
	0	0	0	0-1-2-3-4-5-6-7	0-1-2-3-4-5-6-7
	0	0	1	1-2-3-4-5-6-7-0	1-0-3-2-5-4-7-6
	0	1	0	2-3-4-5-6-7-0-1	2-3-0-1-6-7-4-5
	0	1	1	3-4-5-6-7-0-1-2	3-2-1-0-7-6-5-4
	1	0	0	4-5-6-7-0-1-2-3	4-5-6-7-0-1-2-3
	1	0	1	5-6-7-0-1-2-3-4	5-4-7-6-1-0-3-2
	1	1	0	6-7-0-1-2-3-4-5	6-7-4-5-2-3-0-1
	1	1	1	7-0-1-2-3-4-5-6	7-6-5-4-3-2-1-0

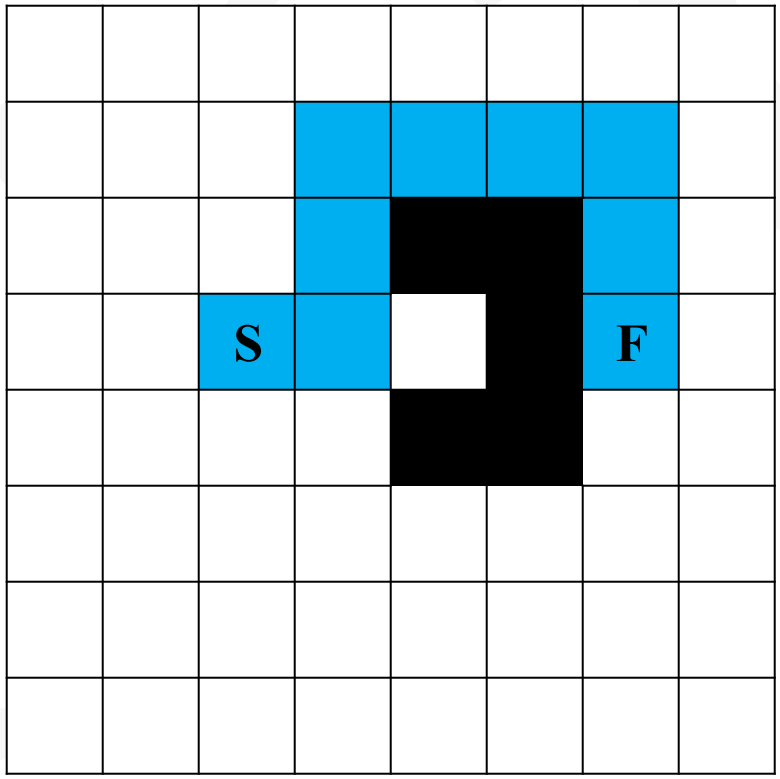
Path Planning

- Find shortest path fromS to F



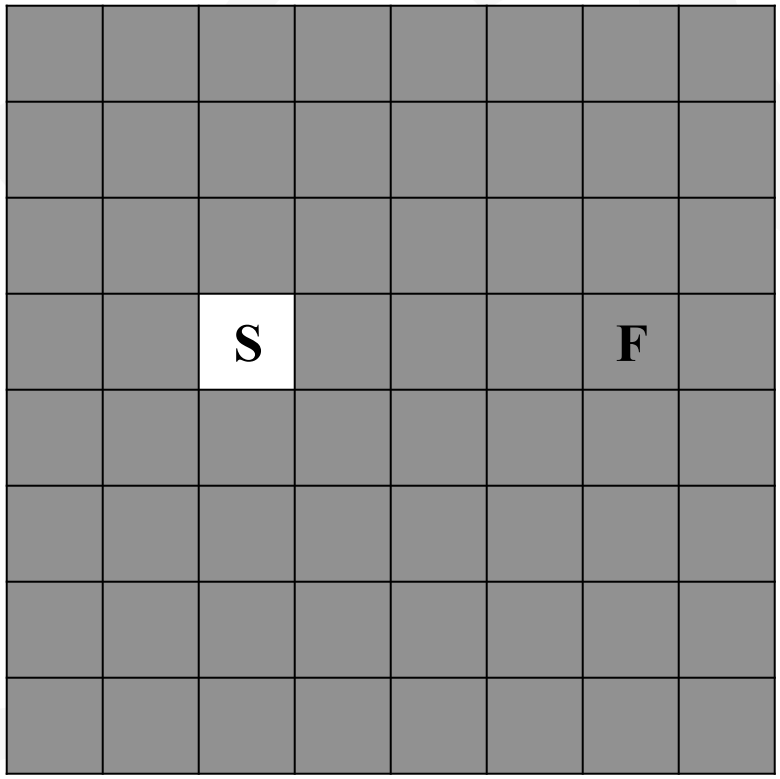
Path Planning (cont.)

- Find shortest path fromS to F



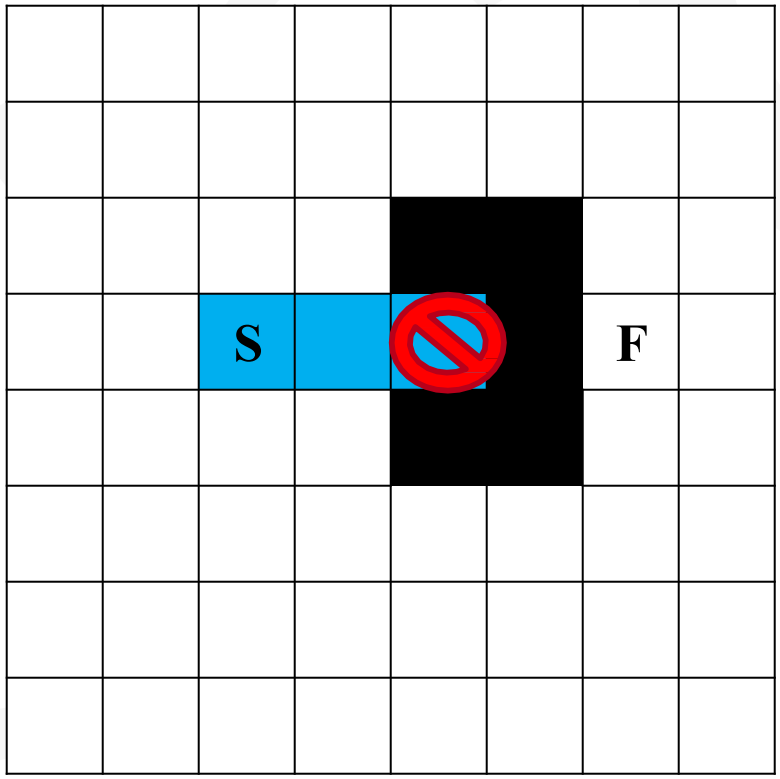
Path Planning (cont.)

- A computer usually can only process (or "see") one or two data items (a square) at a time

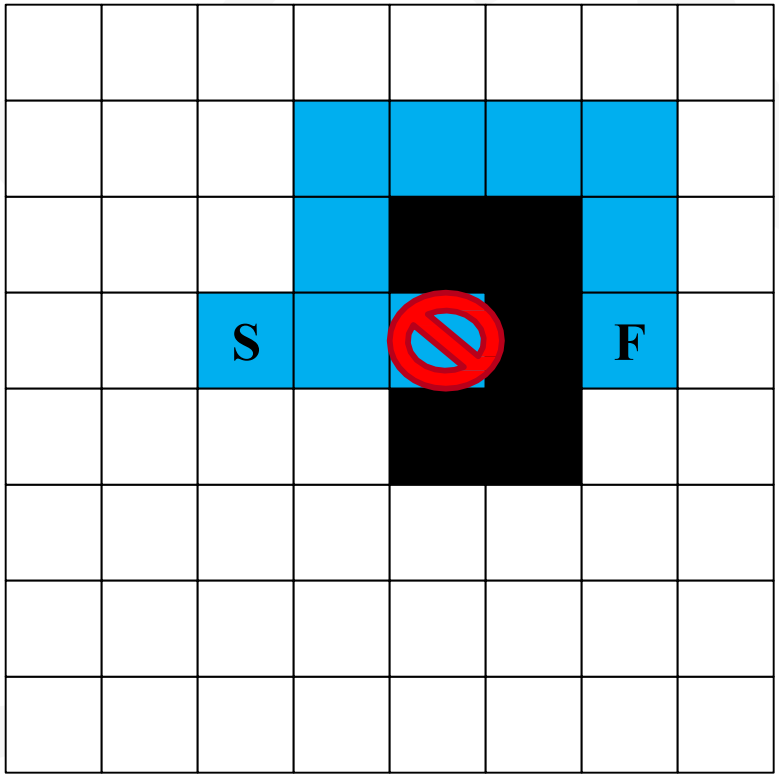


May just compute a straight line path from 'S' to 'F'

Path Planning (cont.)

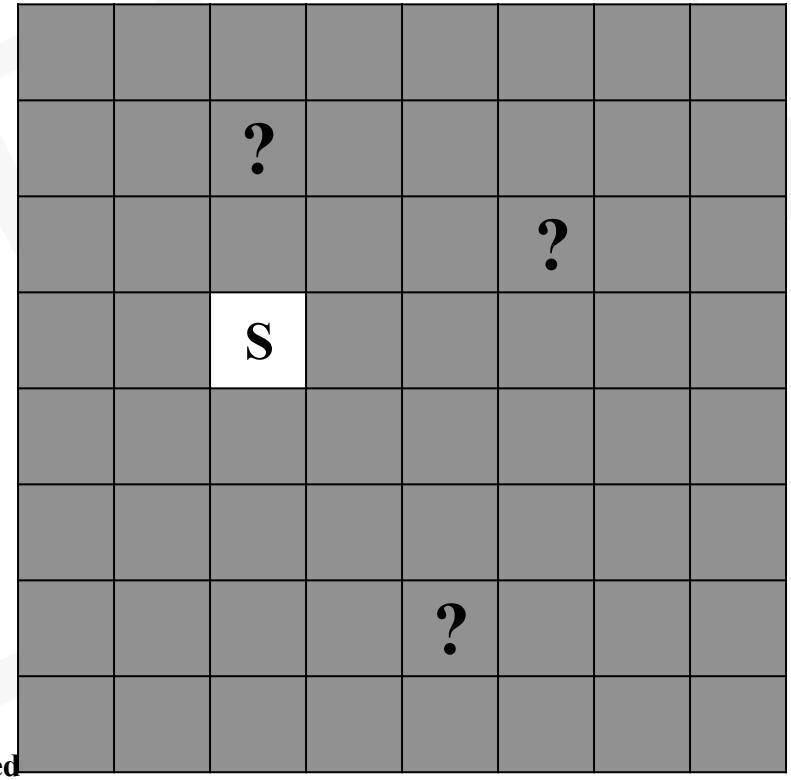


Path Planning (cont.)



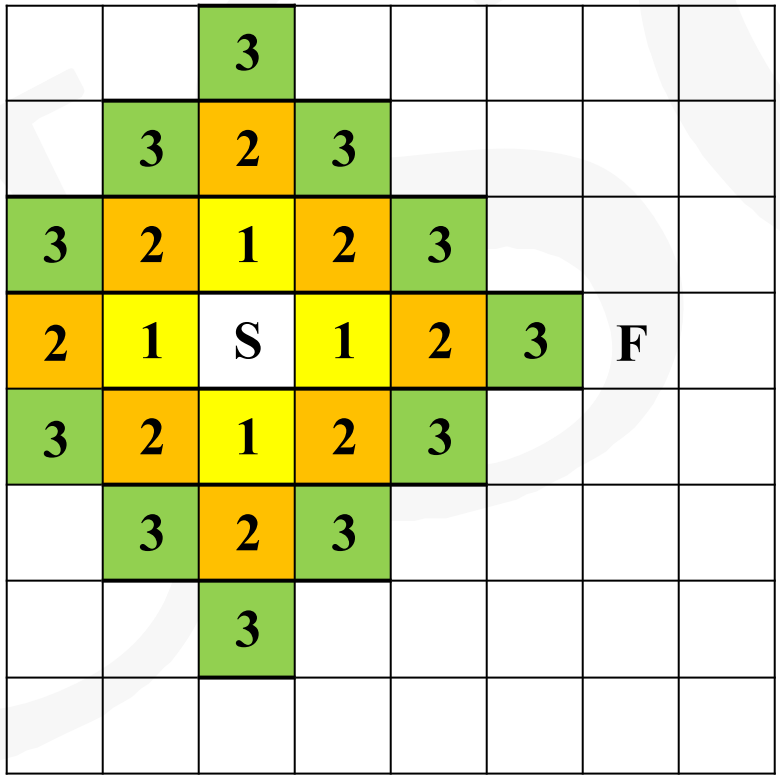
Path Planning (cont.)

- What if I don't know where the Finish square is? Can you devise a general search order to find the shortest path to 'F' while examining the minimum number of squares as possible



Path Planning (cont.)

- Examine all closer squares one at a time before progressing to further squares

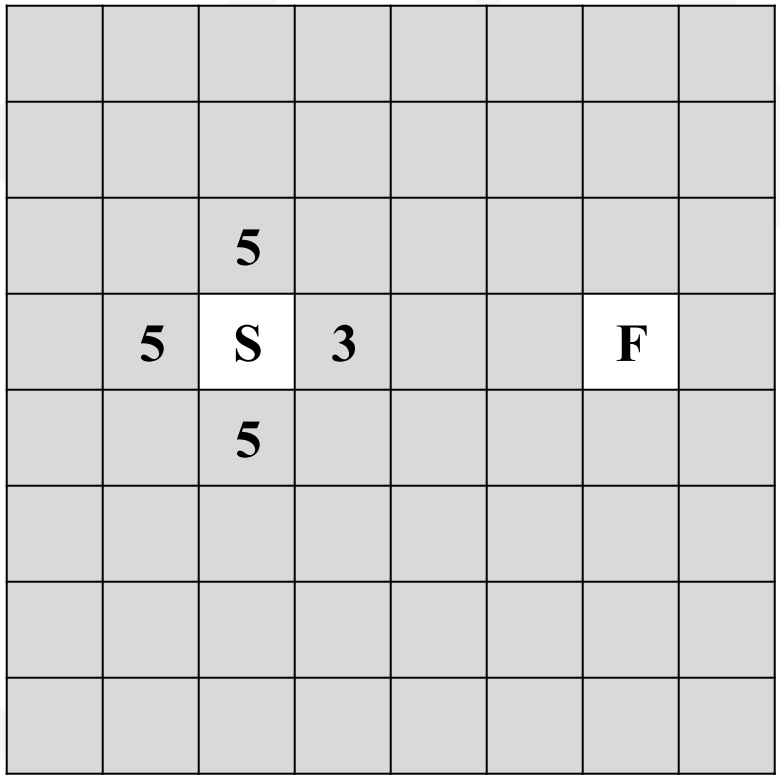


If you don't know where F is and want to find the shortest path, you have to do it this way

Uninformed search for shortest path:
Breadth-first

Path Planning (cont.)

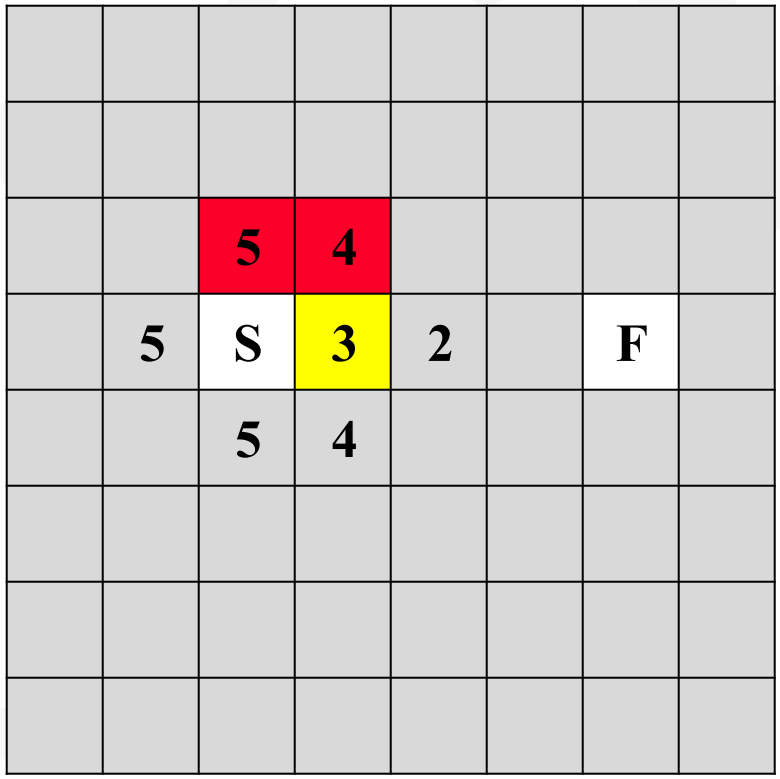
- Now I'll tell you where F is
- Can that help you reduce the number of squares explored?



Select a square to explore with minimum distance to the finish

Path Planning (cont.)

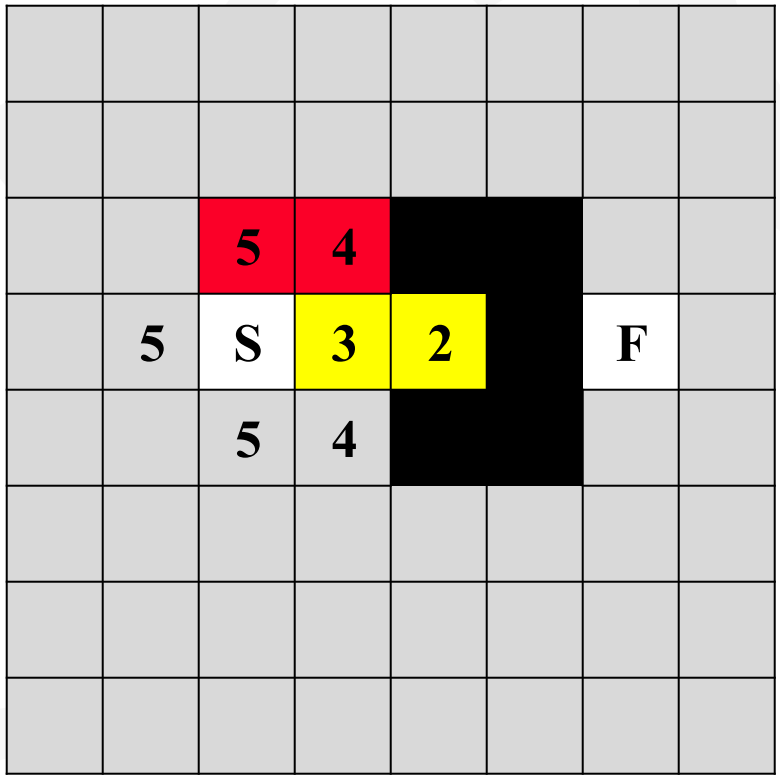
- Now I'll tell you where F is
- Can that help you reduce the number of squares explored?



Select a square to explore with minimum distance to the finish

Path Planning (cont.)

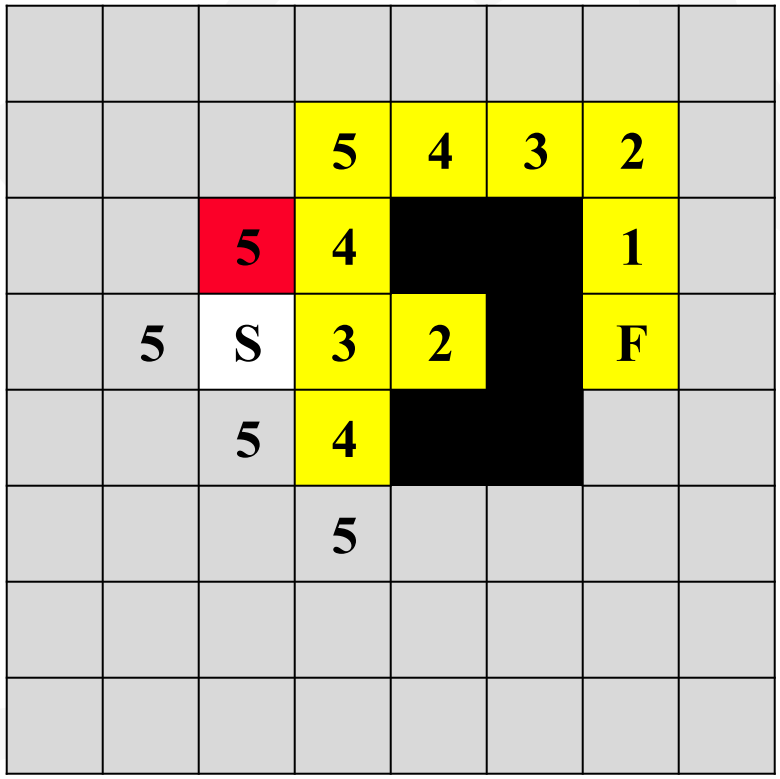
- But what if we run into a blockage?
  - Now we would pick the best among the remainder.



Select a square to explore with minimum distance to the finish

Path Planning (cont.)

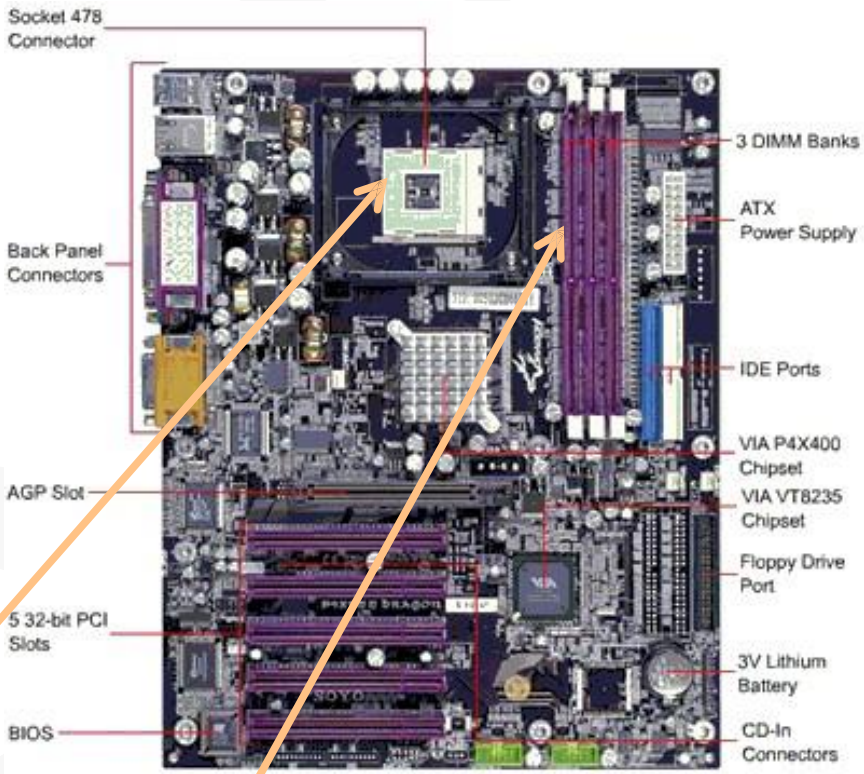
- But what if we run into a blockage?
  - Now we would pick the best among the remainder.

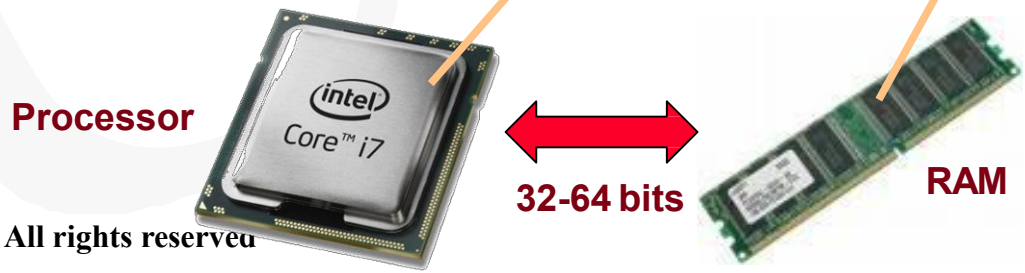


Select a square to explore with minimum distance to the finish

But Why?

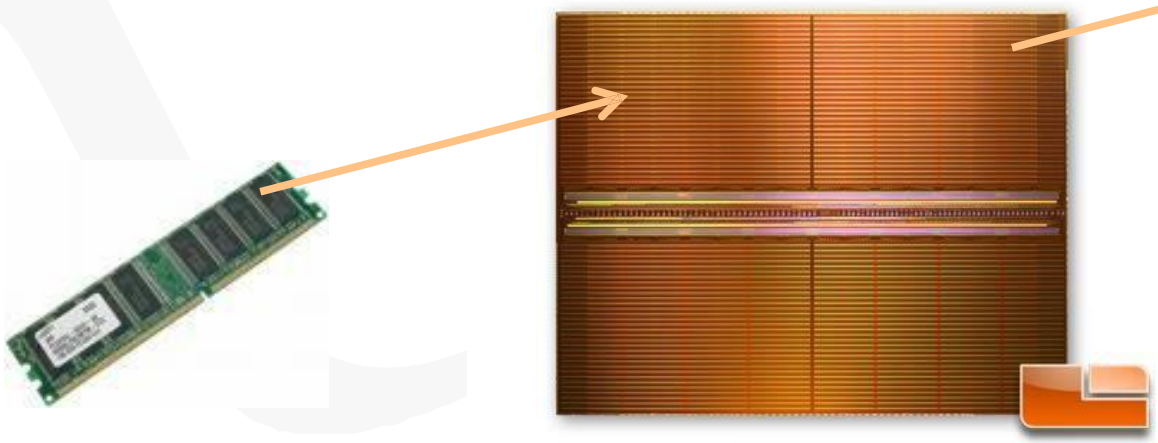
- Why can't a computer just "look" at the image
  - Computers store information as numbers
  - These numbers are stored as units of 8-, 32- or 64-bits and the processor is only capable to looking at 1 or 2 numbers simultaneously
  - Each pixel of the image is a separate piece of data





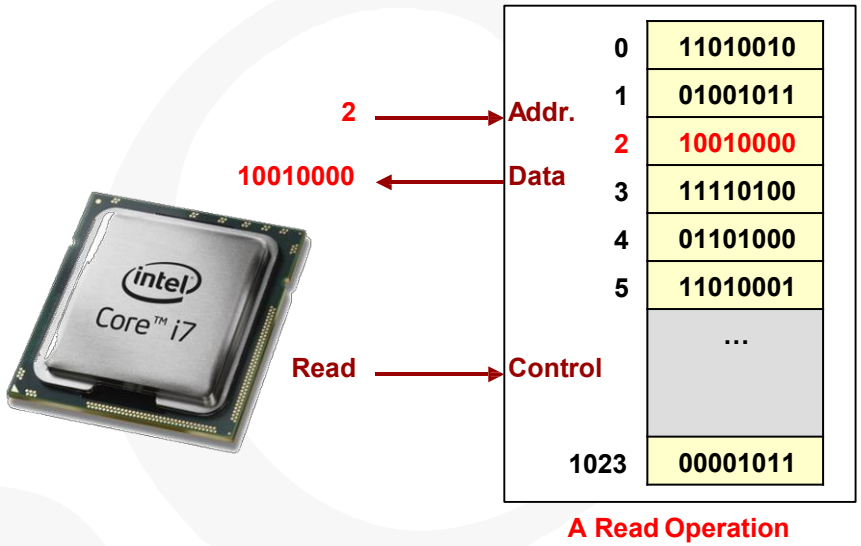
Memory

- Set of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
  - Used to reference the value in that location

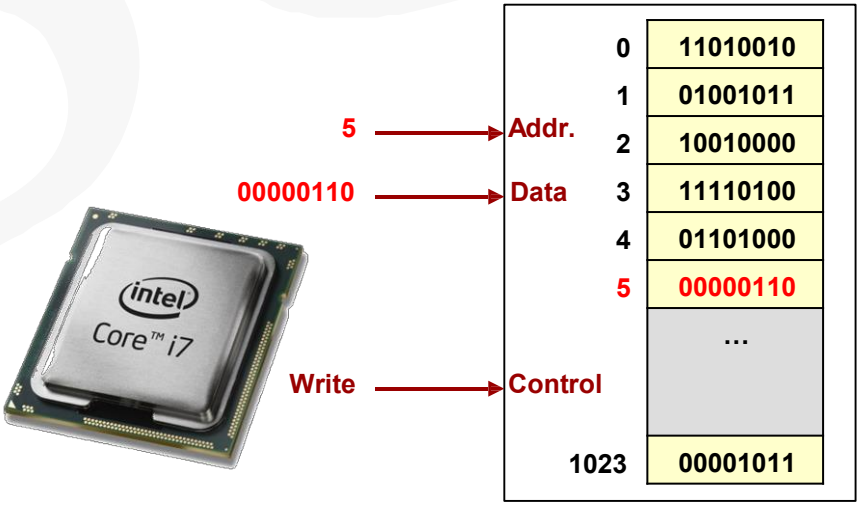


Memory Operations

- Memories perform 2 operations
  - Read: retrieves data value in a particular location (specified using the address)
  - Write: changes data in a location to a new value
- To perform these operations a set of address, data, and control inputs/outputs are used
  - Note: A group of wires/signals is referred to as a 'bus'
  - Thus, we say that memories have an address, data, and control bus



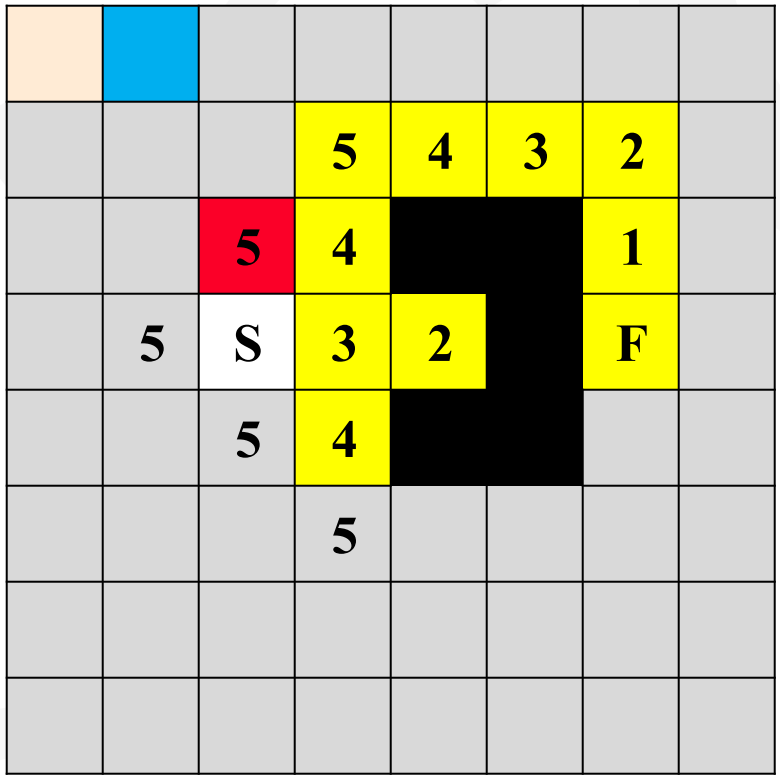
A Read Operation

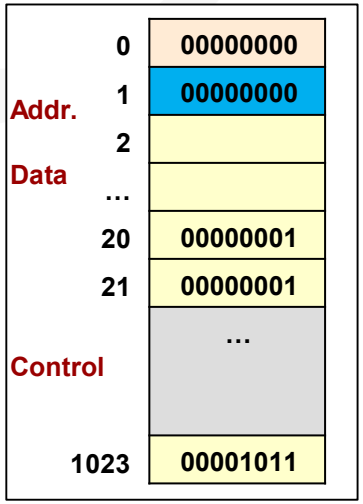


A Write Operation

Programming vs. Algorithms

- Programming entails converting an algorithm into a specific process that a computer can execute



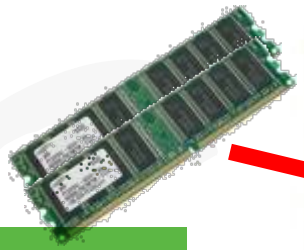


CORE CONCEPTS

Computer Organization Tour

- CPU
  - Logic circuits to execute instructions
  - Reads/writes instructions & data from RAM
- RAM
  - Stores data and program instructions (code) as it executes
- Hard Drives
  - Stores text and other data files that the program may want to read and write
  - Must be brought into RAM before being processed
- I/O Adaptors

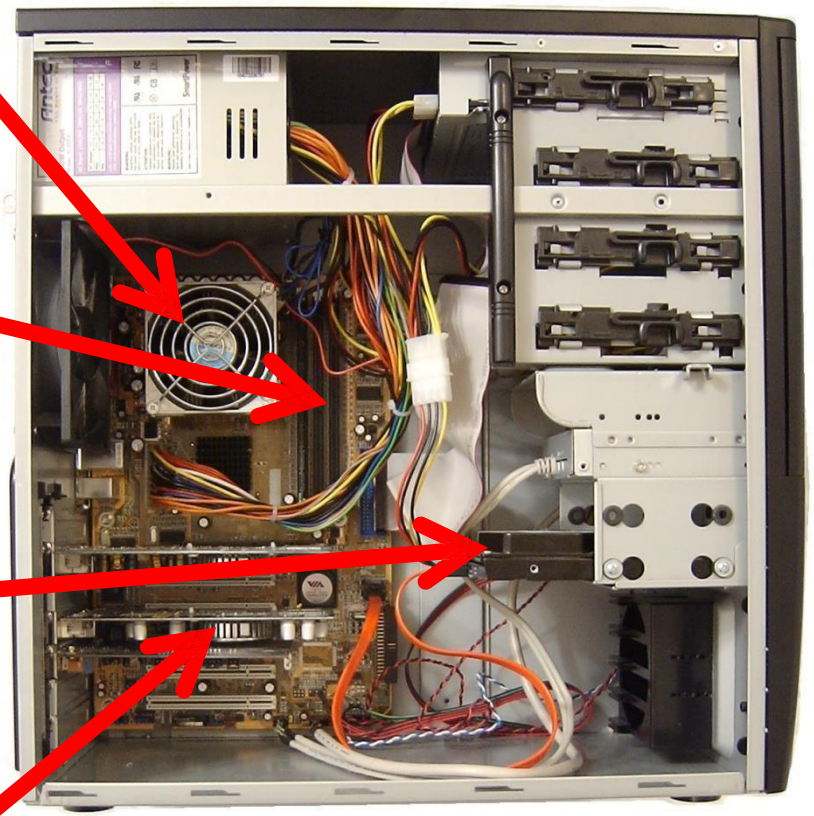
CPU



RAM

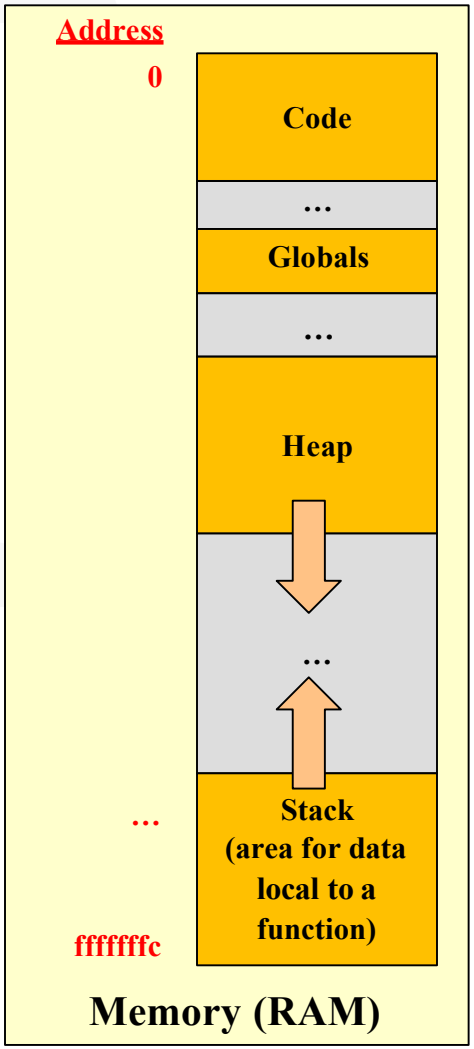
Hard Drive

I/O Adaptor
(Video Card)



Memory Organization

- 32-bit address range (0x0– 0xffffffff)
- Code usually sits at low addresses
- Global variables/data somewhere after code
- Heap: Area of memory that can be allocated and de-allocated during program execution (i.e. dynamically at run-time) based on the needs of the program
- System stack (memory for each function instance that is alive)
  - Local variables
  - Returnlink (where to return)
  - etc.
- Heap grows downward, stack grows upward…
  - In rare cases of large memory usage, they could collide and cause your program to fail or generate an exception/error

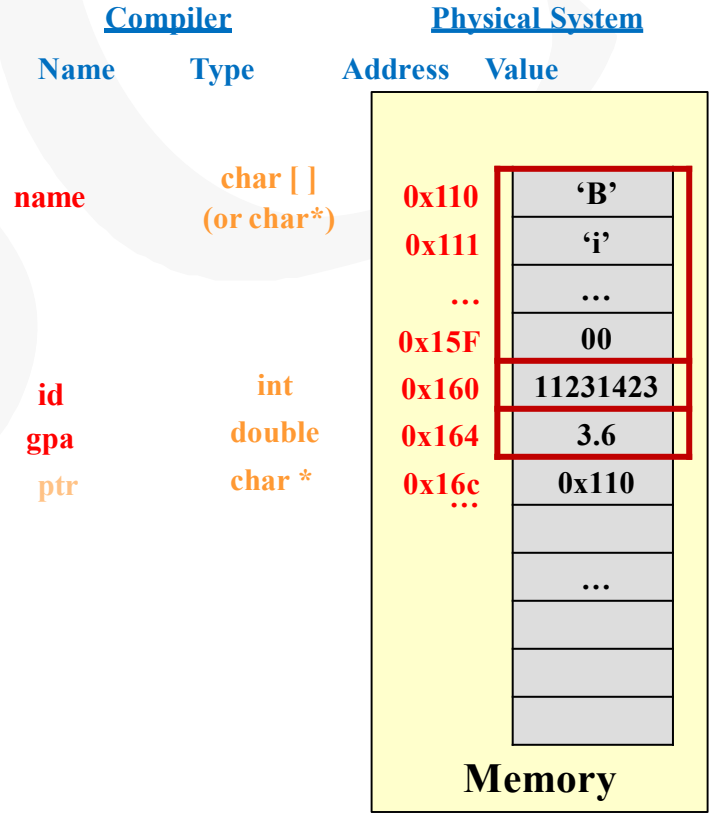


Variables & Memory

- Variables/objects are locations in memory where data is stored
  - Can be primitive data types: int, double, char, etc.
  - Or user-defined struct/class types
- Every variable/object has a name, type, address, & value

```
#include<iostream>
using namespace std;


int main(int argc, char *argv[])
{
  char name[80];
  int id;
  double gpa;
  strncpy(name,"Bill",80);
  id = 11231423;
  gpa = 3.6;
  char *ptr = name;
}
```



C Program Format/Structure

- Comments
  - Anywhere in the code
  - C-Style => "/*" and "*/"
  - C++ Style => "//"
- Compiler Directives
  - #includes tell compiler what other library functions you plan on using
  - 'using namespace std;' -- Just do it for now!
- Global variables (more on this later)
- main() function
  - Starting point of execution for the program
  - Variable declarations often appear at the start of a function
  - All code/statements in C must be inside a function
  - Statements execute one after the next
  - Ends with a 'return' statement
- Other functions

```
/* Anything between slash-star and
   star-slash is ignored even across
   multiple lines of text or code */
/*----Section 1: Compiler Directives  ----*/
#include <iostream>
#include <cmath>
using namespace std;









}
```

Software Process

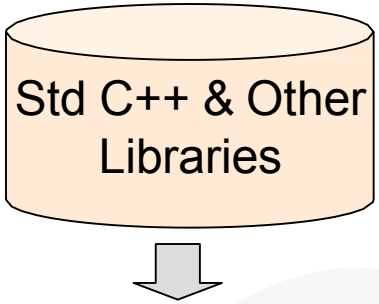
```
#include <iostream>
using namespace std;

int main()

{   int x = 5;
    cout << "Hello"
         << endl;
    cout << "x=" << x;
    return 0;

}
```

C++ file(s)
(test.cpp)



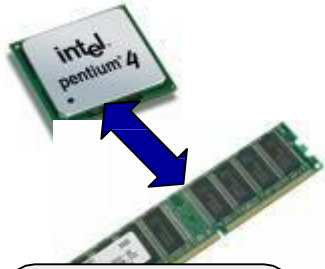
g++

Compiler

-g = Enable Debugging
-Wall =Show all warnings
-o test = Specify Output executable name

1110 0010 0101 1001
0110 1011 0000 1100
0100 1101 0111 1111
1010 1100 0010 1011
0001 0110 0011 1000

Executable Binary
Image
("test")



Load &
Execute

```
$ gedit test.cpp &
```

1 Edit & write code

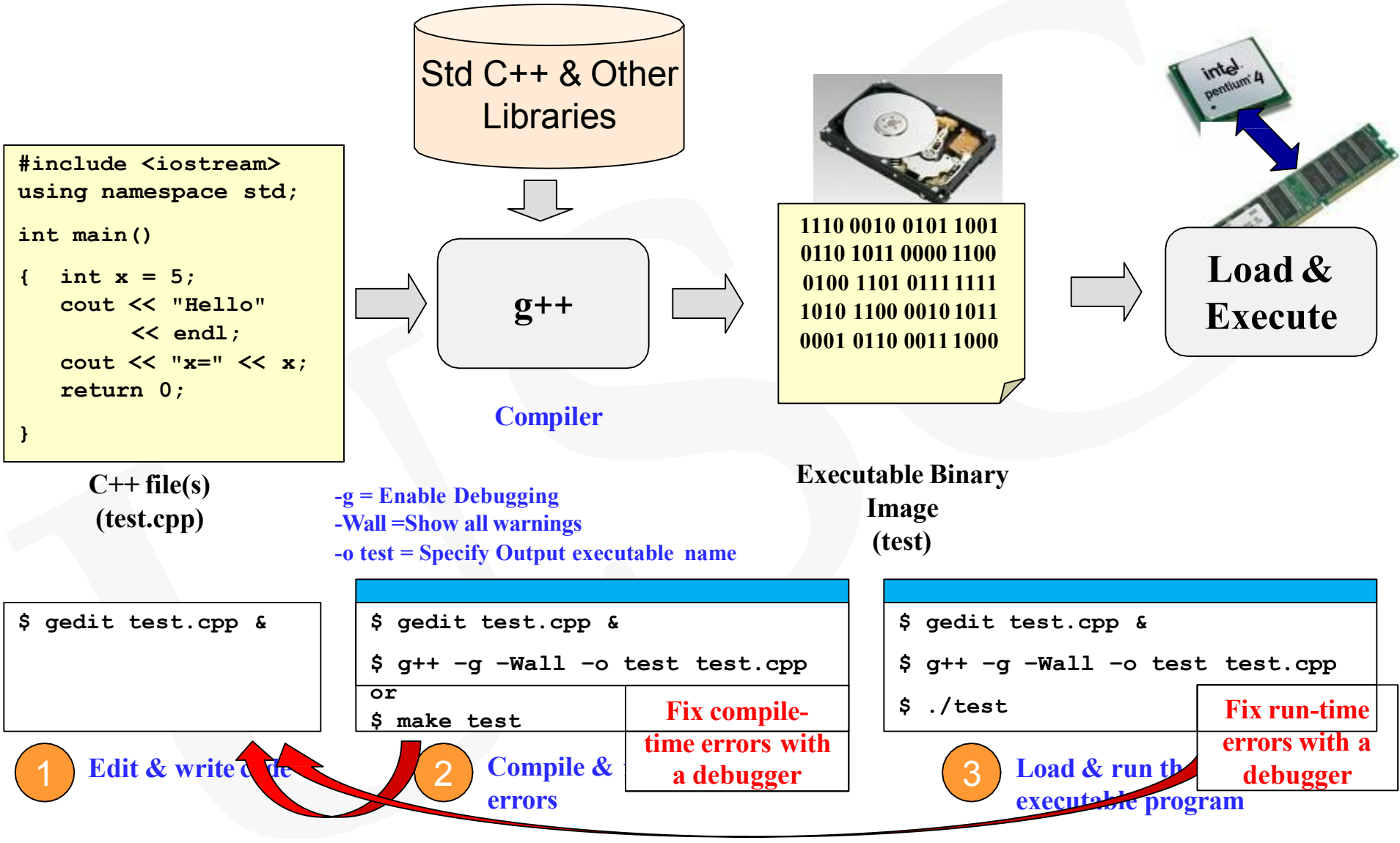
```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
or
$ make test
```

2 Compile & fix compiler errors

```
$ gedit test.cpp &
$ g++ -g -Wall -o test test.cpp
$ ./test
```

3 Load & run the executable program

Software Process (cont.)



COMPILE ERRORS & DEBUGGING

Compile Errors

gdb

- 'g++' will print any compile errors it encounters (and the associated line of the source code file)
  - Warnings will allow compilation to proceed and an executable to be produced, though they should be heeded by the programmer
  - Errors will prevent the executable from being produced and must be fixed
- Many errors may be output, start with the first
- Each error is accompanied by the line number and a "helpful" description of the cause of the error
  - Go back to the line number and see what may be wrong
  - If you have no idea about an error, try to "google" the error description provided by 'gcc' (many others have likely encountered that error)

Run-Time errors

- Your job is not done yet, when your program compiles
  - Though it is a triumph, it is only a minor one…the work is just beginning
- You now need to test your program
  - You should know what you expect it to do
  - What if it doesn't???
- Use a debugger
  - A program that will allow you to execute your program one (or many) steps at a time, then pause and examine variable values, etc.

Getting Help

- Feel free to get help from myself or the TAs, but…
- … before we will look at your code you MUST
  - Have properly indented code (astyle)
  - Know what you expected and localize the error to a line number where output diverges from your expectation
    - Use print statements
    - Use a debugger

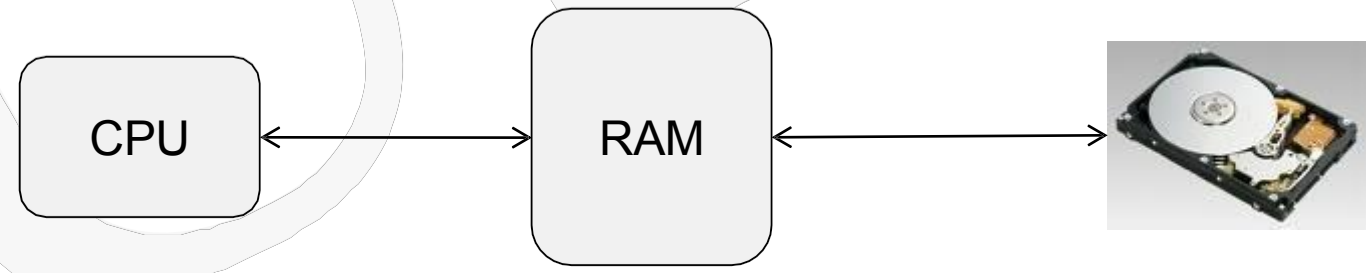
BACKUP

A Live Demo

- Sort an array of integers from 25,000 to 1
  - [N N-1 N-2 … 3 2 1] => [1 2 3 … N-2 N-1 N]
- Using an interpreted Matlab script
- Using C++
- Using a pre-compiled Matlab library function
  - a = 25000 – [0:24999]
  - sort(a) // built-in sort implementation
    // (non-interpreted)

In-Class Activity

- On the cartoon of a computer's processor, RAM, and hard drive shown below overlay the following items (if they exist)
  {Source Code, Executable Program, Program Variables, Data files produced by the program} at 3 points in time:
  - Before the program is compiled
  - After the program is compiled but before it is run
  - As the program is running



Example


- Eggs are packaged in cartons holding 1 dozen and shipped in boxes that hold 5 cartons
- Query the user for how many eggs they have and display the total number of cartons and boxes needed
- Concepts
  - cout and cin
  - Modulo operator (%) and Integer Division
  - eggs