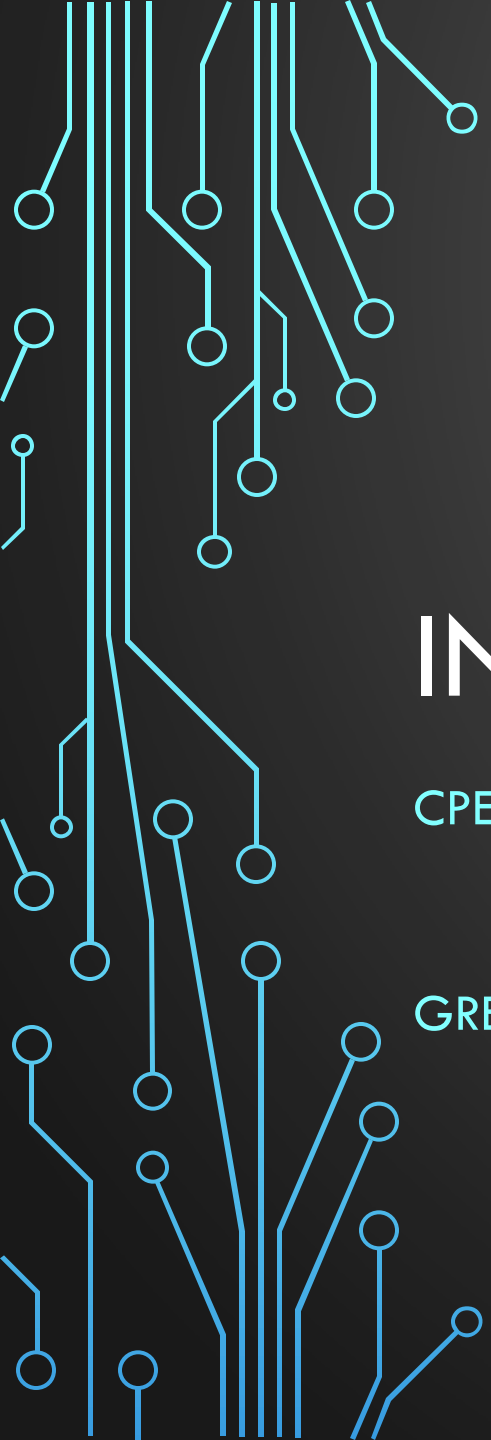


INTRODUCTION TO SPEDE

CPE / CSC 159: OPERATING SYSTEM PRAGMATICS

GREG CRIST (CRIST@CSUS.EDU)

A PHYSICAL DEVELOPMENT ENVIRONMENT

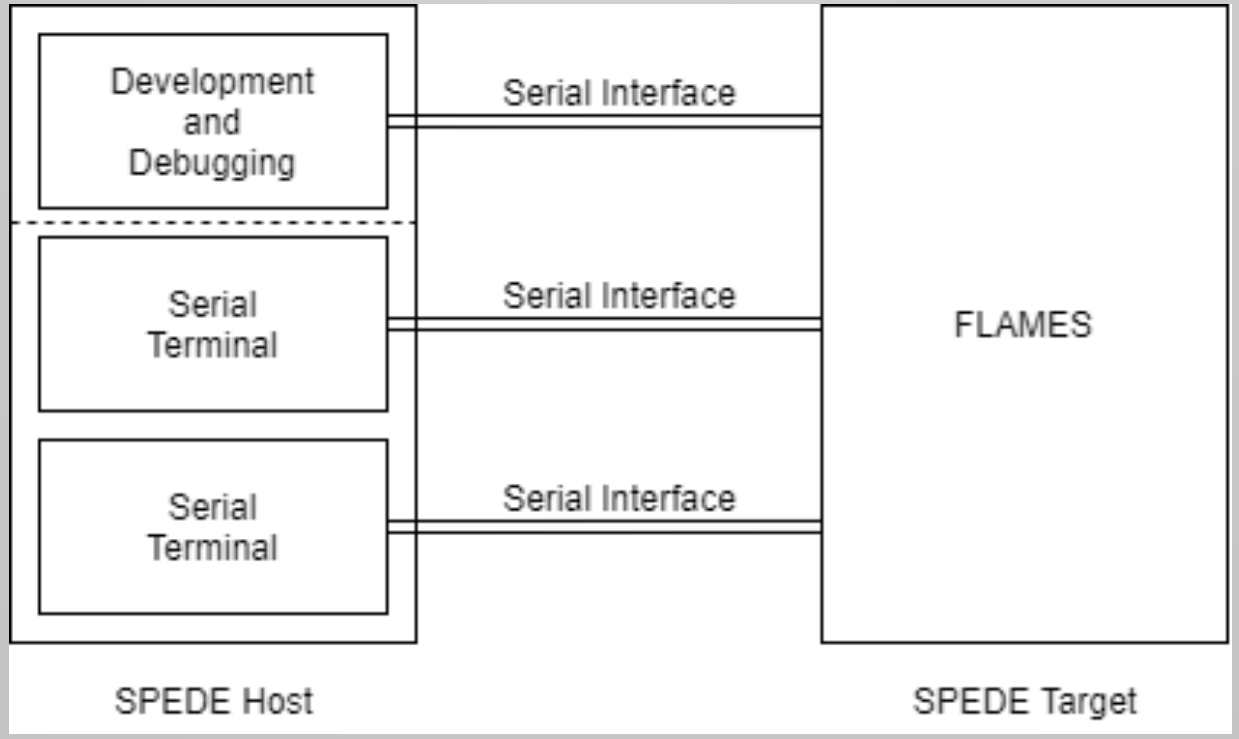
WHY SPEDE?

- SPEDE: System Programmers Educational Development Environment
- SPEDE provides a framework that consists of:
  - a set of basic functionalities to get up and running *fast*
    - c libraries, compiler, debugger
  - a bootloader to load our operating system
- SPEDE traditionally ran on physical hardware but also runs within a Virtual Machine environment
  - Keeps it portable and accessible

THE SPEDE VIRTUAL MACHINES

- SPEDE: System Programmers Educational Development Environment
- Two Virtual Machines:
  - SPEDEHost
    - Our development, compilation, and debugging environment
    - Modern ubuntu Linux with various tools, editors, and more
  - SPEDETarget
    - MS DOS based environment with a bootloader to launch our “operating system” kernel

OUR VIRTUAL ENVIRONMENT



THE SPEDEHOST SYSTEM

- Includes general tools and utilities
- Includes a compiler to build an image for our target system
- Includes a debugger (GDB) to perform live debug on our target system
- Includes the SPEDE/FLAMES header files, libraries, and source code
- Includes the SPEDE/FLAMES “flash programmer” to download our operating system image

THE SPEDETARGET SYSTEM

- Includes the FLAMES bootloader
- Includes the FLAMES runtime environment
- Otherwise, full “hardware” access is available!

BASIC REQUIREMENTS

- Computer with the ability to run VirtualBox Virtual Machines
  - Generally: “modern” x86 (Intel or AMD) CPU; 2GB+ RAM; 10GB+ Disk Space

- Please communicate ASAP if this is NOT possible for you!

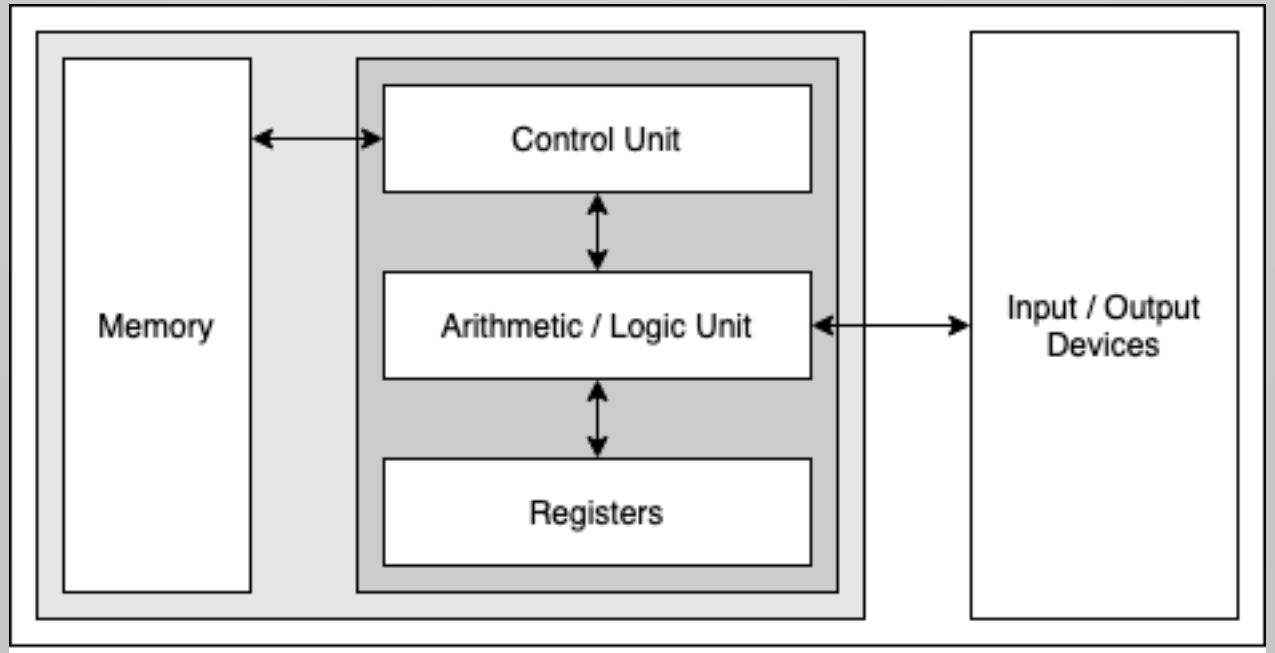
GETTING STARTED

- Your first assignment will be to get up and running with the SPEDE environment
  - Downloading and installing the SPEDE virtual machines
  - Configuring your local git credentials
  - Loading a pre-built demo image from the SPEDE Host to the SPEDE Target
  - Modifying and compiling code to be run on the SPEDE Target
  - Walk through the process of debugging code that is running on the SPEDE Target
  - Practice submitting your assignment

- This is an individual assignment!
  - You need to know how to develop, run, and debug to support your team

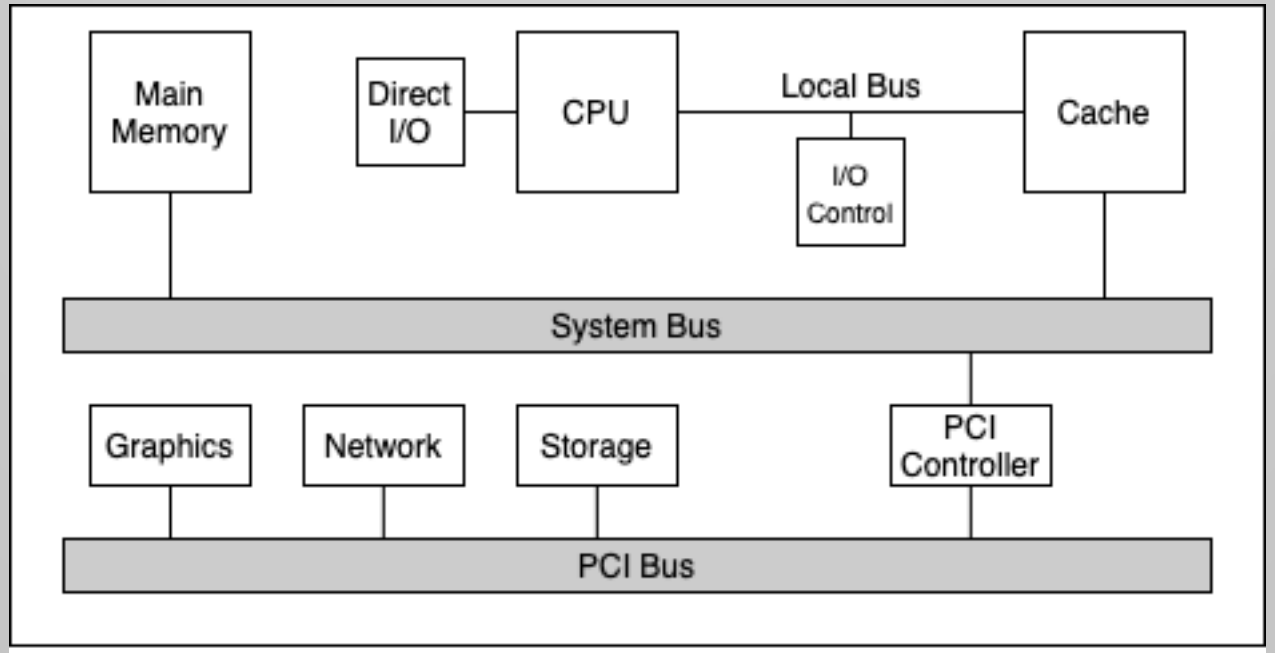
SPEDETARGET

SYSTEM ARCHITECTURE



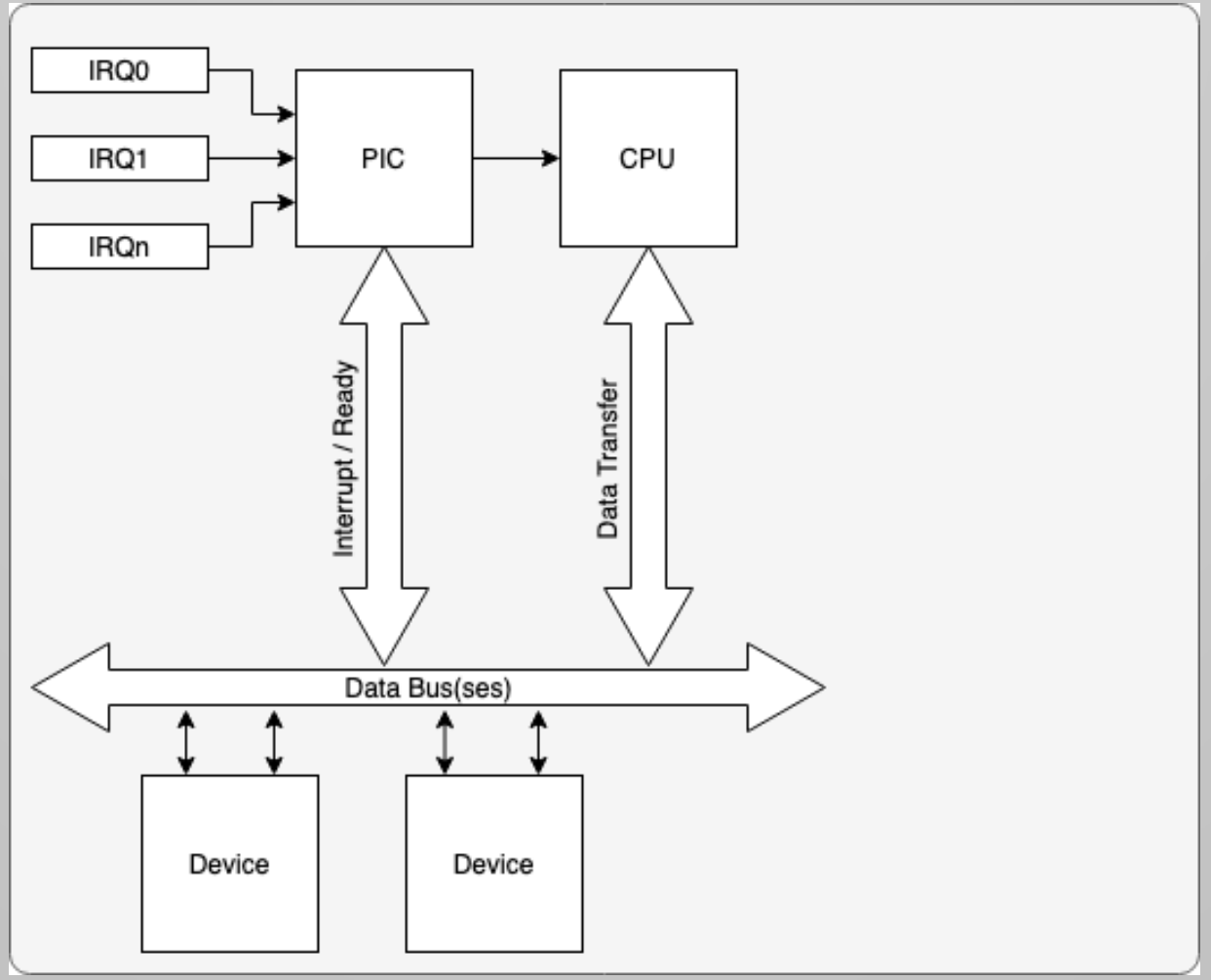
Von Neumann System Architecture

INTEL ARCHITECTURE (SIMPLIFIED)



Intel x86 PC Architecture (Simplified)

“REFERENCE” ARCHITECTURE



TERMINOLOGY AND CONCEPTS (TO EXPLORE)

- Registers
  - Control Registers
  - Segment Registers
  - General Registers
  - Eflags registers
- Instructions
- Data

- Components
  - CPU
  - PIC
  - RAM
  - MMU
  - BUS

REGISTERS

- Control Registers
  - CR0: State Information
  - CR1: Intel Reserved
  - CR2: Page fault address
  - CR3: Virtual memory page directory base address

REGISTERS

- Segment Registers
  - CS: Code Segment - Indicates the code segment where your code runs
  - DS: Data Segment - Indicates the data segment that your code may access
  - ES, FS, GS: Extra Segment(s) – far pointer addressing
  - SS: Stack Segment – Indicates the stack segment where your program's stack is located

REGISTERS

- General Registers
  - EAX, AX, AH, AL – Accumulator Register
    - I/O, arithmetic, interrupt calls
  - EBX, BX, BH, BL – Base Register
    - Base pointer for memory access, interrupt return values
  - ECX, CX, CH, CL – Counter Register
    - Loop counter, interrupt values
  - EDX, DX, DH, DL – Data Register
    - I/O, Arithmetic, interrupt calls

DEMO TIME!