Lab Assignment

- Each team will independently implement the launch interceptor specification
- For this assignment, you’re writing portable C code
- We’ll worry about I/O later

Lab Assignment

- You are allowed to reuse code from the Internet
  - But you must cite the source for anything that is cut-and-pasted or retyped
- Correctness:
  - You get points for creating test cases that other teams cannot handle correctly
    - Including mine!
  - You lose points for not being able to handle test cases generated by other teams

Lab Assignment

- Each team should come up with a division of work for the Launch Interceptor
  - ~1 page of PDF, due in 1 week
  - Who will do what?
  - What are the interfaces between people?
    - Be specific! I want to see prototypes for C functions
  - Who is the test czar?
  - What is the test plan?
    - Again be specific
    - Where will test cases come from?
    - How will you know the answers are right?
    - Unit testing vs. system testing?
    - What does the test harness look like?

Lab Assignment

- You'll be using git, hosted on github
  - Has a bit of a learning curve!
  - I'll send out instructions and links to docs

Mars Curiosity Rover

- Duplicated computers, one is backup
- Each has a RAD750 CPU
  - PPC architecture
  - Up to 200 MHz
- 256 MB of DRAM, 2 GB of flash memory
- Runs VxWorks: a real-time OS
- Software written in C

Today

- Requirements
- Design
  - Architectures
  - Processors
  - Languages

Embedded System Requirements

- Two basic flavors
  - Functional – What the system does
    - We just talked about this
  - Non-functional (or para-functional) – Important properties not directly related to what the system does

Example Non-Functional Requirements

- Energy-efficient
- Real-time
- Safety critical
- Upgradeable
- Cost sensitive
- Highly available or fault-tolerant
- Secure

- These issues cut across system designs
  - Important (and difficult) to get them right
  - We’ll be spending a lot of time on these

Crosscutting Issues

- **Energy efficiency**
  - Must run for years on a tiny battery (hearing aid, pacemaker)
  - Unlimited power (ventilation control)
- **Real-time**
  - Great harm is done if deadlines are missed (process control, avionics, weapons)
  - Few time constraints (toy)

More Crosscutting Issues

- **Safety critical**
  - **Device is safety critical (nuclear plant)**
  - **Failure is largely irrelevant (toy, electric toothbrush)**
- **Upgradability**
  - **Impossible to update (spacecraft, pacemaker)**
  - **Easily updated (firmware in a PC network card)**

More Crosscutting Issues

- **Cost sensitivity**
  - **A few % in extra costs will kill profitability (many products)**
  - **Cost is largely irrelevant (military applications)**

- **Availability / fault-tolerance**
  - **Must be operating all the time (pacemaker, spacecraft control)**
  - **Can shutdown at any time (cell phone)**

More Crosscutting Issues

- **Secure**
  - **Security breach extremely bad (smart card, satellite, missile launch control)**
  - **Security irrelevant (many systems)**
- **Distributed**
  - **Single-node (many systems)**
  - **Fixed topology (car)**
  - **Variable topology (sensor network, bluetooth network)**

Software Architectures

- Important high-level decision when building an embedded system:
  - What does the "main loop" look like?
- How is control flow determined?
  - What computations can preempt others, and when?
- How is data flow determined?
- Options:
  - Cyclic executive
  - Event-driven
  - Threaded
  - Dataflow
  - Client-server

Cyclic Executive

```
main() {
  init();
  while (1) {
    a();
    b();
    c();
    d();
}}
```

Advantages?

Disadvantages?

Historically, most embedded systems are based on cyclic executives

Cyclic Exec. Variations

```
main() {
  init();
  while (1) {
    wait_on_clock();
    a();
    b();
    c();
}}
```

```
main() {
  init();
  while (1) {
    a();
    b();
    a();
    c();
    a();
}}
```

Interrupt Driven

```
main() {
  while (1) { }
}
```

```
interrupt_handler() {
  …
}
```

Or…

```
main() {
  while (1) {
    sleep();
  }
}
```

Advantages?

Disadvantages?

Event Driven

```
main() {
  while (1) {
    event_t e =
      get_event();
    if (e) {
      (e)();
    } else {
     sleep_cpu();
}}}
```

```
interrupt_handler() {
  time_critical_stuff();
  enqueue_event
    (non_time_critical);
}
```

Advantages?

Disadvantages?

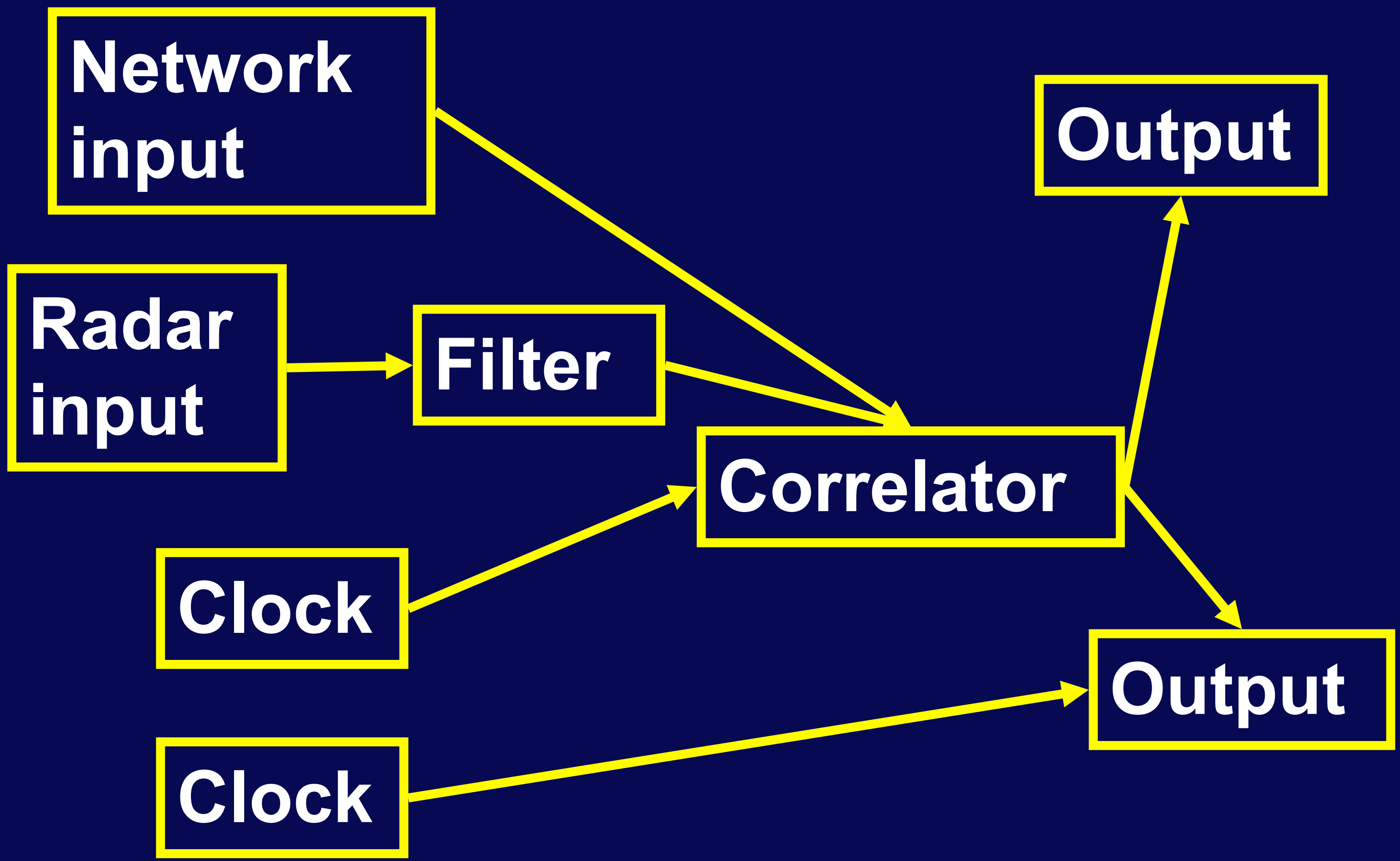
Threaded (using an RTOS)

- Threads are usually sleeping on events
- Highest priority thread runs except when:
  - It's blocked
  - An interrupt is running
  - It wakes up and another thread is executing in the kernel

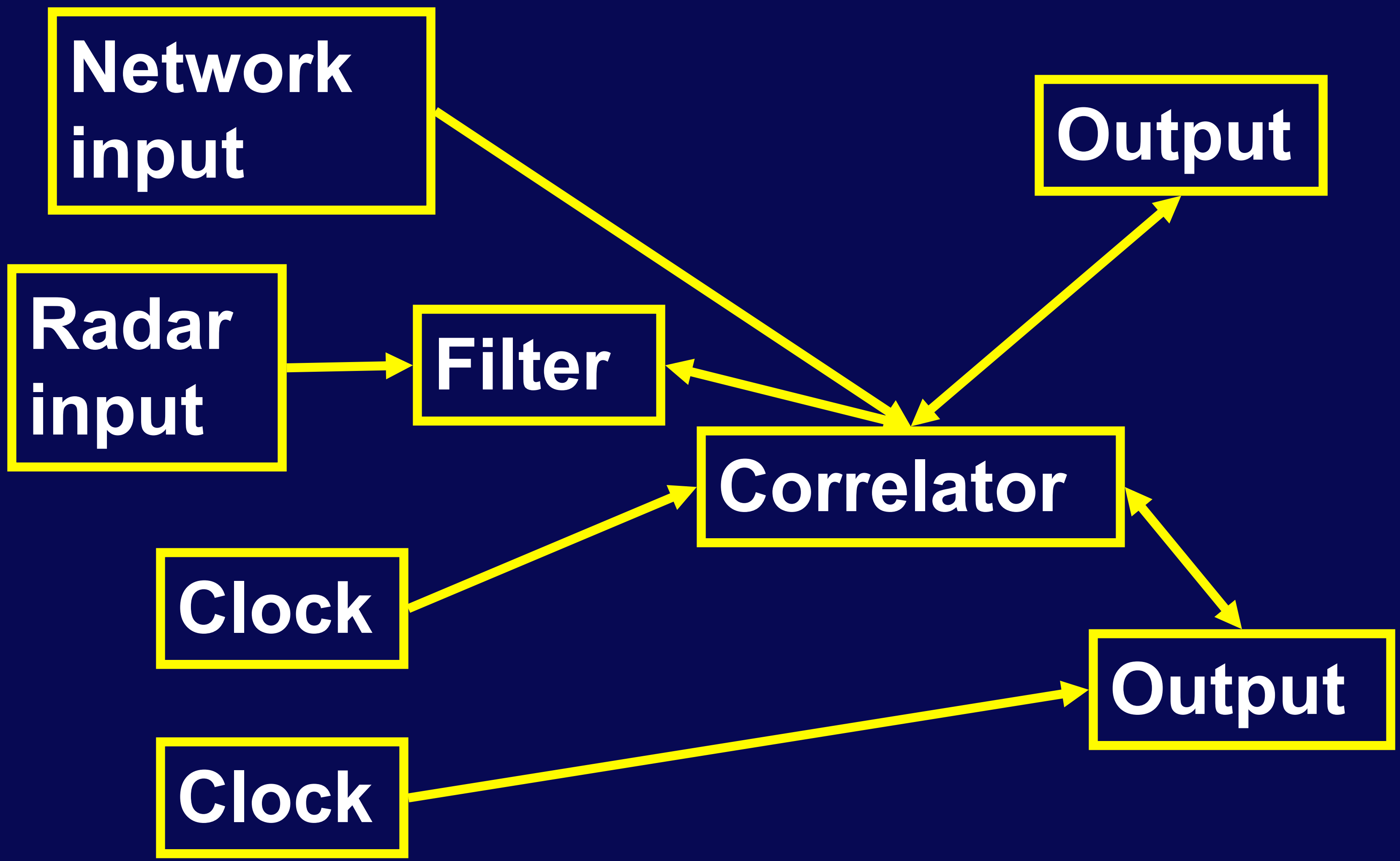
Advantages?

Disadvantages?

Pipeline-Driven (Dataflow)



Client-Server



Architecture Summary

- All of the architectures have significant advantages and disadvantages
  - Resource usage
  - Responsiveness
  - Safety
  - Fault tolerance
  - Maintainability
- Once an architecture is chosen, lots of other design decisions follow
- Very important to choose an appropriate architecture for a new system
- Architectures can be combined
  - But this is hard to get right

Choosing a CPU

- Issues:
  - Cost
  - Size
  - Pinout
  - Devices
  - Performance
  - Match to system workload
  - Memory protection
  - Address space size
  - Word size
  - User / kernel support
  - Floating point

CPU Options

- Create custom hardware
  - May not need any CPU at all!
- 4-bit microcontroller
  - Few nibbles of RAM
  - No OS
  - Software all in assembly
  - E.g. COP400, EM73201, W741E260, HD404358
  - Dying out?

More CPU Options

- 8-bit microcontroller
  - A few bytes to a few hundred KB of RAM
  - At the small end software is in asm, at the high end C, C++, Java
  - Might run a home-grown OS, might run a commercial RTOS
  - Still dominate both numbers and dollar volume
  - Two kinds:
    - Old style
      - CISC, designed for hand-written code
      - E.g. 68HC11, 6502, Z80, 8051
      - These are >20 years old and doing well
    - New style
      - RISC, designed as a compiler target
      - E.g. AVR, PIC

More CPU Options

- 16- and 32-bit microcontrollers
  - Few KB to many MB of RAM
  - Usually runs an RTOS
  - May or may not have caches
  - Wide range of costs
  - 16-bit: MSP430, 68HC16, H8
  - 32-bit: ARM, MIPS, MN10300, x86, PPC, ColdFire
  - Labs in this class will use ARM
  - Is 16-bit dying?
    - Has serious disadvantages compared to 32-bit but few advantages
  - New ARM "Cortex" processors designed to kill the 8-bit and 16-bit markets

More CPU Options

- 32- or 64-bit microprocessor
  - Basically a PC in a small package
  - Runs Win XP, Linux, or whatever
  - Relatively expensive in power and $$
- Many specialized processors exist
  - E.g. DSP – optimized for signal processing

Choosing a Language

- **Issues:**
  - Footprint
    - RAM, ROM
  - Efficiency
  - Debuggability
  - Predictability
  - Portability
  - Toolchain quality
  - Libraries
  - Level of abstraction
  - Developer availability
    - Anyone know Jovial? PL/1? Forth? BCPL?

Programming Languages

- Assembler
  - No space overhead
  - Good programmers write fast code
  - Non-portable
  - Very hard to debug
- C
  - Little space and time overhead
  - Somewhat portable
  - Good compilers exist

More Languages

- C++
  - Often used as a “better C”
  - Low space and time overhead if used carefully
  - Unbelievably complex, especially C++11
- Java
  - More portable
  - Full Java requires lots of RAM
  - J2ME popular on cell-phone types of devices
  - Bad for real-time!

Choosing an OS

- Issues very similar to languages
  - Footprint
    - RAM, ROM
  - Efficiency
  - Debuggability
  - Predictability
  - Portability
- Other issues
  - Process / thread model
  - Device support
  - Scheduling model
  - Price and licensing model

Real-Time OS

- Low end: Not much more than a threads library
- High end: Stripped-down version of Linux or WinXP
- Many, many RTOSs exist
  - They are quite easy to create
- Interesting RTOSs:
  - QNX
  - uClinux
  - uC/OS-II
  - VxWorks

Summary

- Embedded systems are highly diverse
- External requirements dictate
  - Choice of CPU, language, OS
  - Choice of software architecture
    - This is worth thinking about very carefully
- Very different experience developing embedded apps relative to desktop apps
- Embedded systems are:
  - Fun – They make stuff happen in the real world
  - Important – Your life depended on hundreds of them on the way to school today
  - Ubiquitous – More processors sold per year than people on earth

Assignment for Tuesday

- Find an embedded device that you can take apart such as an old…
  - Cell phone, home router or hub, MP3 player, printer, …
- Should be a device you don't care about
  - If you can't find one, talk to me
- Open the device so you can see the main circuit board(s)
- Identify as many parts as possible – search for part numbers on the web
- Estimate cost to produce the device
  - Part cost + assembly cost
  - Compare to purchase cost
- Talk about the device in class on Tues
  - Also hand in a short writeup – I'll send mail about this