Lecture 5: Behind Event-Driven Systems

Engineering Design with Embedded Systems

Patrick Lam
University of Waterloo

January 15, 2013

Today's Plan

Where do events come from?

Digression: Priorities

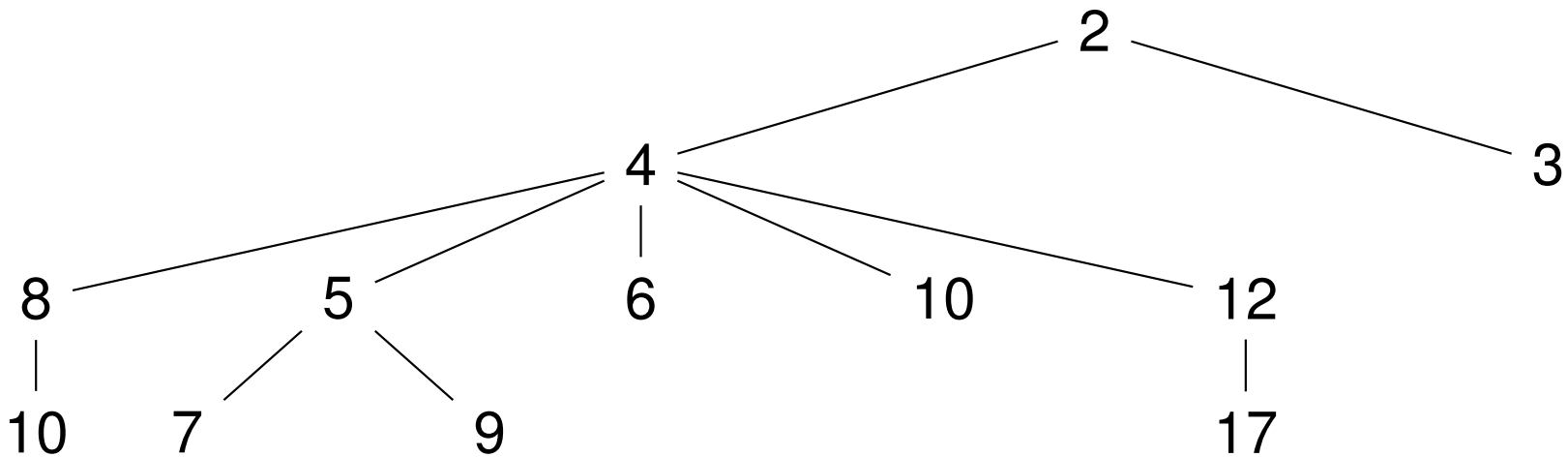
Imagine the following situation:

- your mom calls;
- supper is burning;
- laundry is done.

What do you do?

Implementing Priorities

Associate a priority with each event.
Use a *priority queue* data structure to get the highest-priority event.



Events and Finite-State Machines

Remember: reactive, not proactive.

How can the application do what it wants?

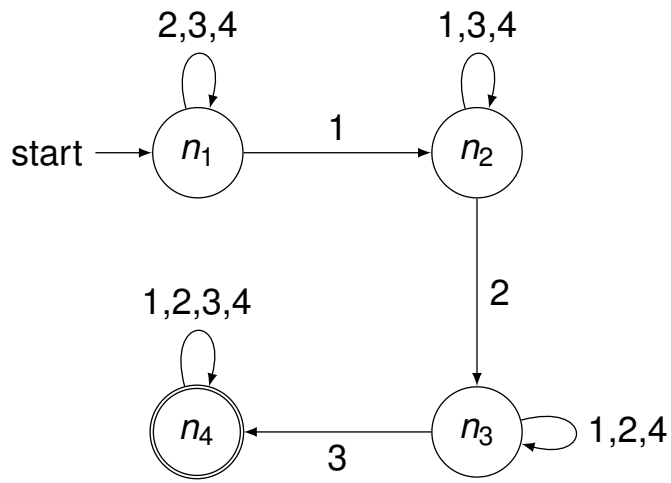
Use Finite-State Machines!

Events and Finite-State Machines

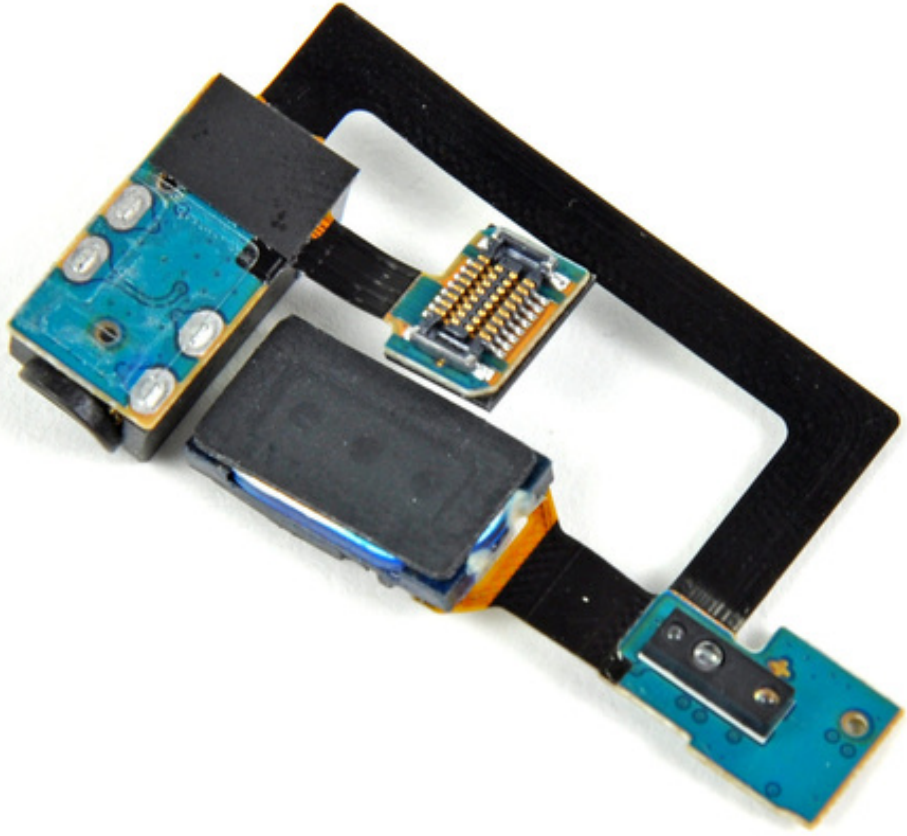
Remember: reactive, not proactive.

How can the application do what it wants?

Use Finite-State Machines!

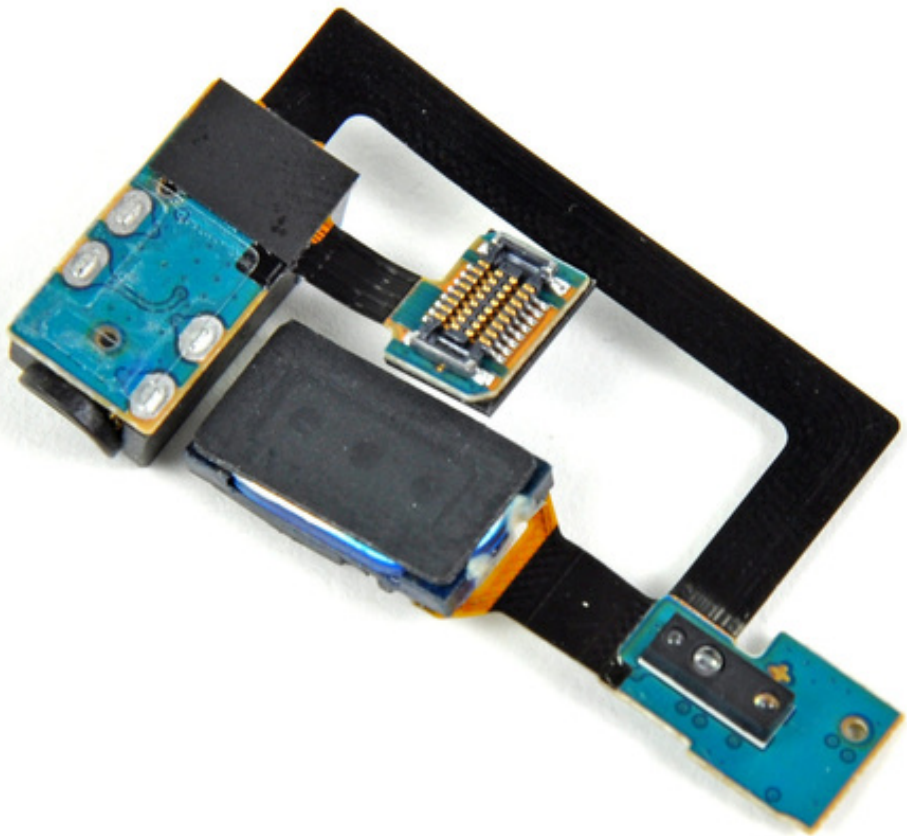


Where Events Come From



Analog-to-Digital Converter. Then what?

Where Events Come From



Analog-to-Digital Converter. Then what?

“Are we there yet?”

— example of polling.

“Are we there yet?”

— example of polling.

Polling

Polling: processor requests readings from the device at its convenience.

“What is the current light level?”

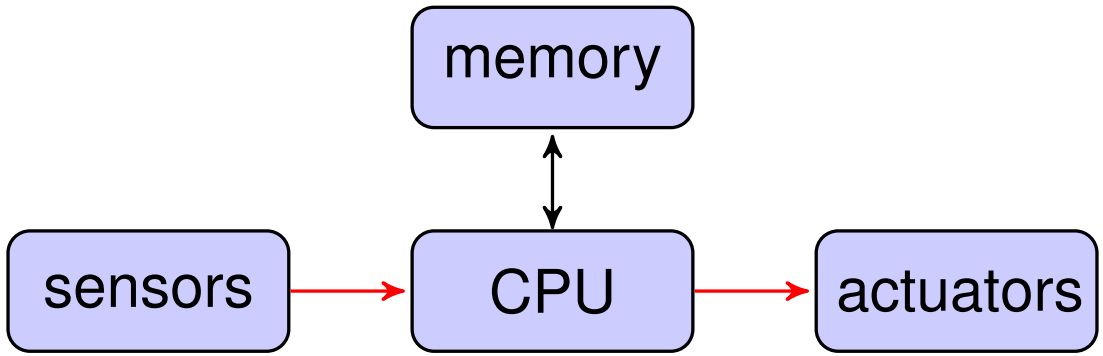
(also known as “passive synchronization”)

When to Poll?

It depends:

- whenever convenient (occasional polling);
- at fixed time intervals (periodic polling); or
- constantly (tight polling).

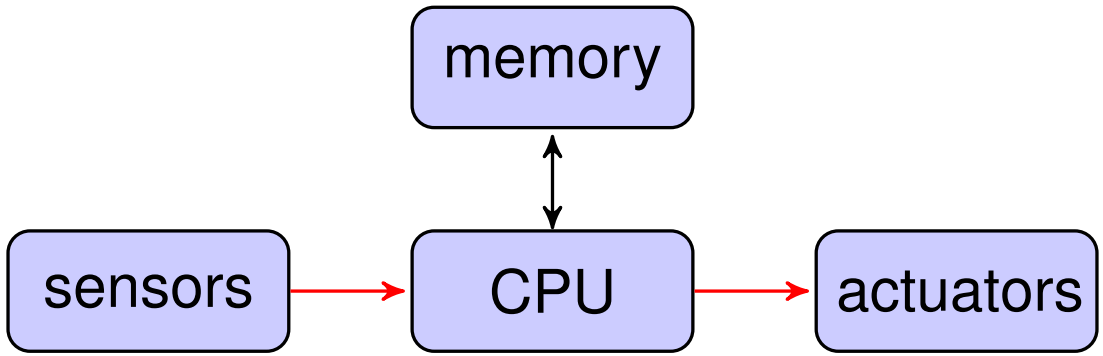
How to Get the Data



What's the mechanism?

- port-mapped I/O; or,
- memory-mapped I/O

How to Get the Data



What's the mechanism?

- port-mapped I/O; or,
- memory-mapped I/O

Port-mapped I/O: Special CPU Instructions

The Intel ia32 processors provide special I/O instructions:

```
outb    ax, 0x3f8
inw     dx, ax
```

May use a special bus, or set a specific signal on the bus.

Memory-mapped I/O Example

CPU just reads and writes to “memory”.

```
while (statusRegister == 0x0000) {
    // Do nothing until statusRegister changes value
}
// Read data that has changed from a dataRegister
// and store in memory
incomingData = dataRegister;
```

Devices listen on the bus and respond.

Memory-mapped I/O Example

```
while (statusRegister == 0x0000) {
    // Do nothing until statusRegister changes value
}
// Read data that has changed from a dataRegister
// and store in memory
incomingData = dataRegister;
```

This is a *tight polling loop*.

- Expect the hardware specification to promise that statusRegister eventually changes due to an external event.
- Data exchange occurs once device is ready: polling synchronization.

Interrupts: an alternative to polling

So far: processor controls when to read data from a device.

Instead: device may tell the processor when device is ready, using an interrupt.

This constitutes active synchronization.

How Interrupts Work

Interrupt tells the processor: “Something’s happening!”

Upon receipt of an interrupt, the processor:

- stops what it’s currently doing and saves its state;
- starts executing pre-defined interrupt handler, which:
  - reads the event information; and
  - stores it somewhere accessible.
- upon return from handler, resumes previous state.