

Processes

REFRESH TYPE	EXAMPLE SHORTCUTS	EFFECT
SOFT REFRESH	EMAIL <input type="button" value="REFRESH"/> BUTTON	REQUESTS UPDATE WITHIN JAVASCRIPT
NORMAL REFRESH	F5, CTRL-R, ⌘R	REFRESHES PAGE
HARD REFRESH	CTRL-F5, CTRL-⇧, ⌘⇧R	REFRESHES PAGE INCLUDING CACHED FILES
HARDER REFRESH	CTRL-⇧-HYPER-ESC-R-F5	REMOTE CYCLES POWER TO DATACENTER
HARDEST REFRESH	CTRL-⌘-⇧-#-R-F5-F5-ESC-O-O-Ø-▲-SCROLL LOCK	INTERNET STARTS OVER FROM ARPANET

Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Memory & data
Arrays & structs
Integers & floats
RISC V assembly
Procedures & stacks

Executables

Memory & caches
Processor Pipeline
Performance
Parallelism

Assembly
language:

```
get_mpg(car*):
    lw    a5,0(a0)
    lw    a4,4(a0)
    divw  a5,a5,a4
    fcvt.s.w    fa0,a5
    ret
```

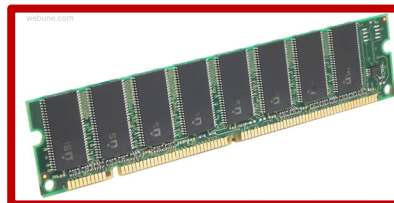
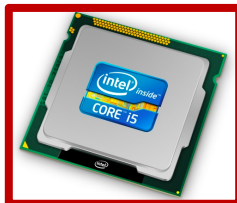
Machine
code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

OS:



Computer
system:



Leading Up to Processes

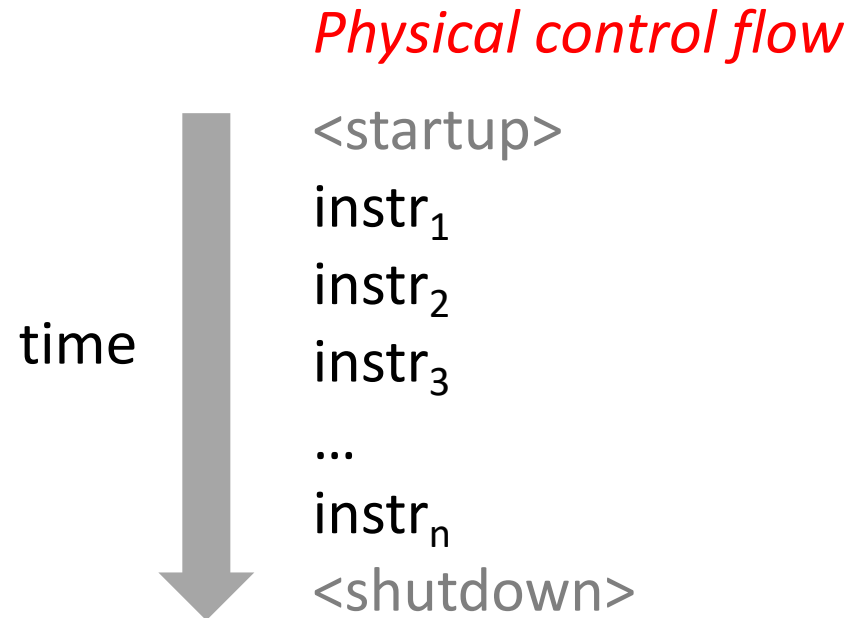
- ❖ System Control Flow
 - **Control flow**
 - **Exceptional control flow**
 - Asynchronous exceptions (interrupts)
 - Synchronous exceptions (traps & faults)

Control Flow

- ❖ **So far:** we've seen how the flow of control changes as a *single program* executes
- ❖ **Reality:** multiple programs running *concurrently*
 - How does control flow across the many components of the system?
 - In particular: More programs running than CPUs
- ❖ **Exceptional control flow** is basic mechanism used for:
 - Transferring control between *processes* and OS
 - Handling *I/O* and *virtual memory* within the OS
 - Implementing multi-process apps like shells and web servers
 - Implementing concurrency

Control Flow

- ❖ Processors do only one thing:
 - From startup to shutdown, a CPU simply reads and executes (interprets) a sequence of instructions, one at a time
 - This sequence is the CPU's *control flow* (or *flow of control*)



Altering the Control Flow

- ❖ Up to now, two ways to change control flow:
 - Jumps (conditional and unconditional)
 - Call and return
 - Both react to changes in *program state*

- ❖ Processor also needs to react to changes in *system state*
 - Unix/Linux user hits “Ctrl-C” at the keyboard
 - User clicks on a different application’s window on the screen
 - Data arrives from a disk or a network adapter
 - Instruction divides by zero
 - System timer expires

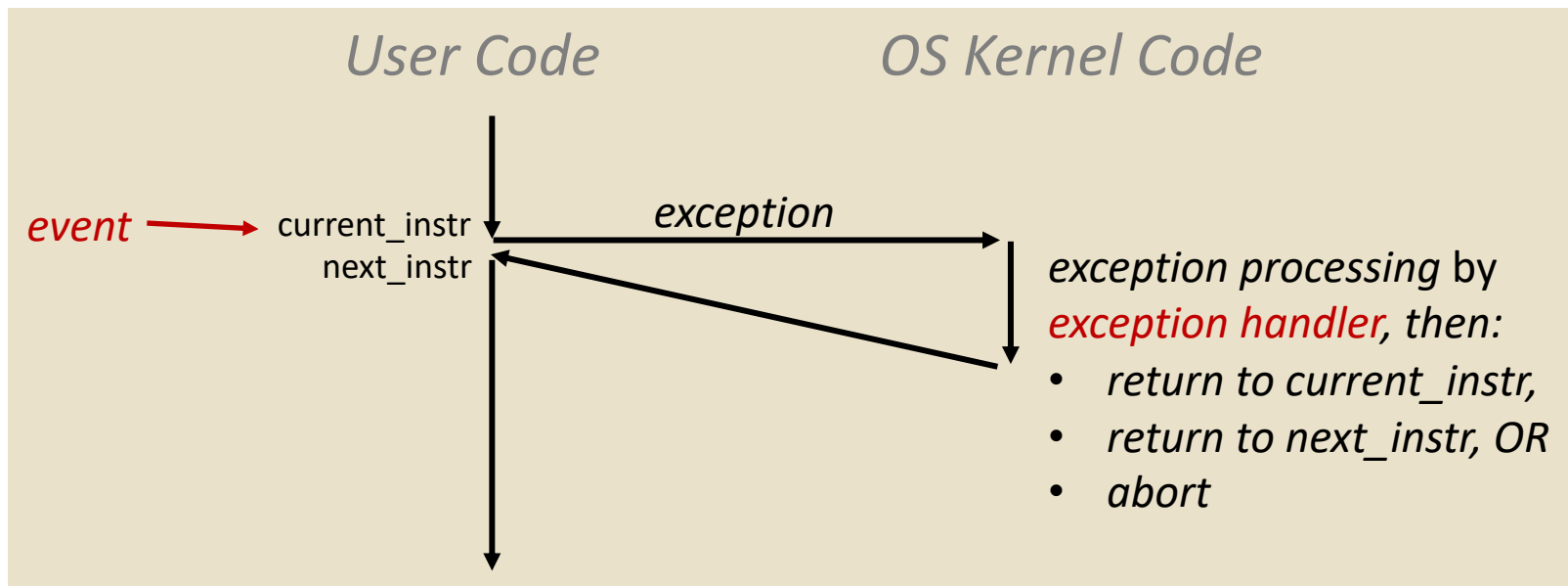
- ❖ Can jumps and procedure calls achieve this?
 - No – the system needs mechanisms for *“exceptional”* control flow!

Exceptional Control Flow

- ❖ Exists at all levels of a computer system
- ❖ Low level mechanisms
 - **Exceptions**
 - Change in processor's control flow in response to a system event (*i.e.* change in system state, user-generated interrupt)
 - Implemented using a combination of hardware and OS software
- ❖ Higher level mechanisms
 - **Process context switch**
 - Implemented by OS software and hardware timer
 - **Signals**
 - Implemented by OS software
 - We won't cover these – see CSE451 and CSE/EE474

Exceptions

- ❖ An *exception* is transfer of control to the operating system (OS) kernel in response to some *event* (i.e. change in processor state)
 - Kernel is the memory-resident part of the OS
 - Examples: division by 0, page fault, I/O request completes, Ctrl-C

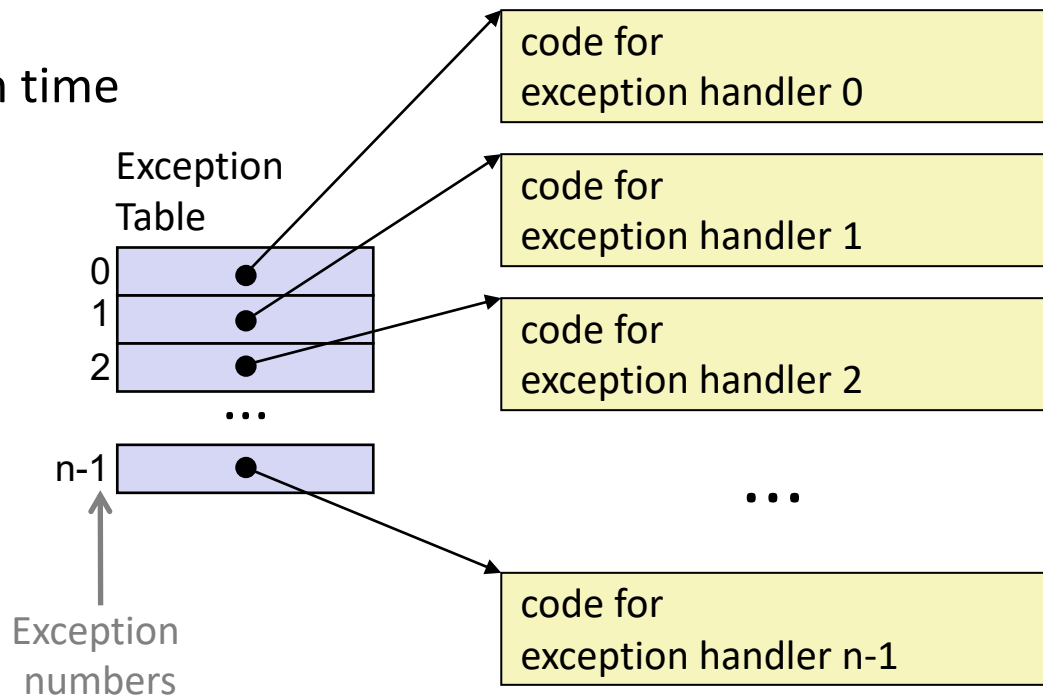


- ❖ *How does the system know where to jump to in the OS?*

Exception Table

This is extra
(non-testable)
material

- ❖ A jump table for exceptions (also called *Interrupt Vector Table*)
 - Each type of event has a unique exception number k
 - k = index into exception table (a.k.a interrupt vector)
 - Handler k is called each time exception k occurs



Exception Table (Excerpt)

This is extra
(non-testable)
material

<i>Exception Number</i>	<i>Description</i>	<i>Exception Class</i>
0	Divide error	Fault
13	General protection fault	Fault
14	Page fault	Fault
18	Machine check	Abort
32-255	OS-defined	Interrupt or trap

Leading Up to Processes

- ❖ System Control Flow
 - Control flow
 - Exceptional control flow
 - **Asynchronous exceptions (interrupts)**
 - **Synchronous exceptions (traps & faults)**

Asynchronous Exceptions (Interrupts)

- ❖ Caused by events external to the processor
 - Indicated by setting the processor's interrupt pin(s) (wire into CPU)
 - After interrupt handler runs, the handler returns to “next” instruction

- ❖ Examples:
 - I/O interrupts
 - Hitting Ctrl-C on the keyboard
 - Clicking a mouse button or tapping a touchscreen
 - Arrival of a packet from a network
 - Arrival of data from a disk
 - Timer interrupt
 - Every few milliseconds, an external timer chip triggers an interrupt
 - Used by the OS kernel to take back control from user programs

Synchronous Exceptions

- ❖ Caused by events that occur as a result of executing an instruction:
 - **Traps**
 - **Intentional**: transfer control to OS to perform some function
 - Examples: *system calls*, breakpoint traps, special instructions
 - Returns control to “next” instruction
 - **Faults**
 - **Unintentional** but possibly recoverable
 - Examples: *page faults*, segment protection faults, integer divide-by-zero exceptions
 - Either re-executes faulting (“current”) instruction or aborts
 - **Aborts**
 - **Unintentional** and unrecoverable
 - Examples: parity error, machine check (hardware failure detected)
 - Aborts current program

Summary

❖ Exceptions

- Events that require non-standard control flow
- Generated externally (interrupts) or internally (traps and faults)
- After an exception is handled, one of three things may happen:
 - Re-execute the current instruction
 - Resume execution with the next instruction
 - Abort the process that caused the exception

Processes

- ❖ **Processes and context switching**
- ❖ Creating new processes
 - `fork()`, `exec*()`, and `wait()`
- ❖ Zombies

What is a process?

It's an *illusion!*

