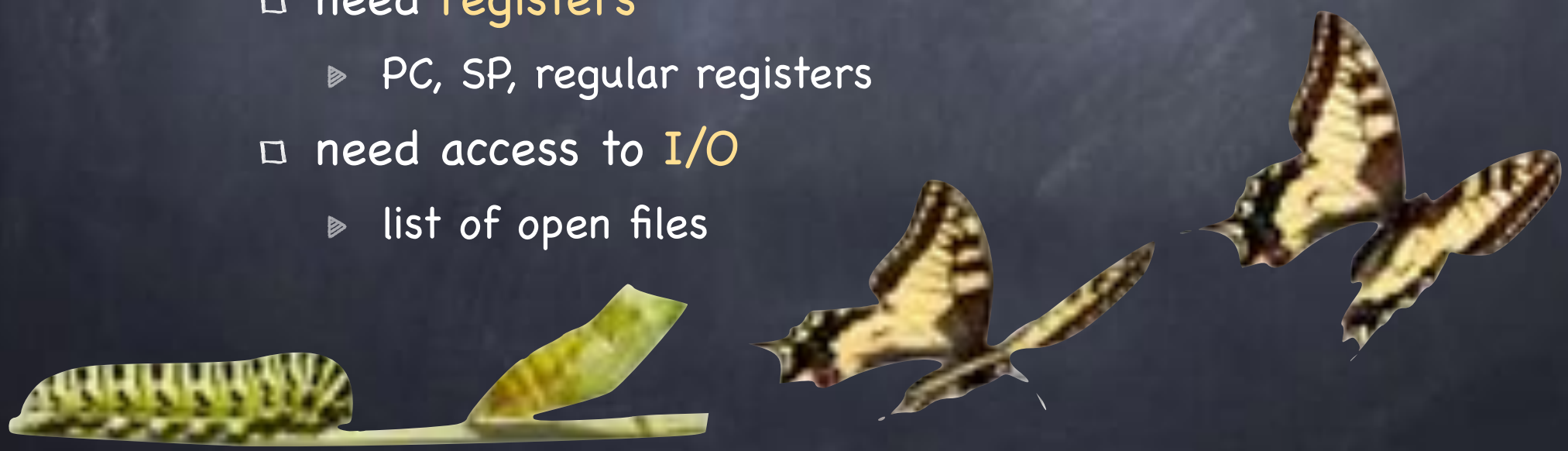
The Process

A running program

(Chapters 2-6)

From Program to Process

- To make the program's code and data come alive
  - need a CPU
  - need memory — the process' address space
    - for data, code, stack, heap
  - need registers
    - PC, SP, regular registers
  - need access to I/O
    - list of open files



A First Cut at the API

- Create
  - causes the OS to create a new process
- Destroy
  - forcefully terminates a process
- Wait (for the process to end)
- Other controls
  - e.g. to suspend or resume the process
- Status
  - running? suspended? blocked? for how long?

Process Magement by the OS

PCB

PC
Stack Ptr
Registers
PID
UID
Priority
List of open files
Process status
Kernel stack ptr
...

Process Control Block (PCB)

- A per-process data structure held by the OS
- Stores three types of information
  - Process identification
  - Process state (registers, PC, SP, MM Info...)
    - to suspend and restart process
  - Process control
    - scheduling status, priority, CPU time used

You'll Never Walk Alone

- Machines run (and thus OS must manage) multiple processes
  - how should the machine's resources be mapped to these processes?
- OS as a referee...

You'll Never Walk Alone

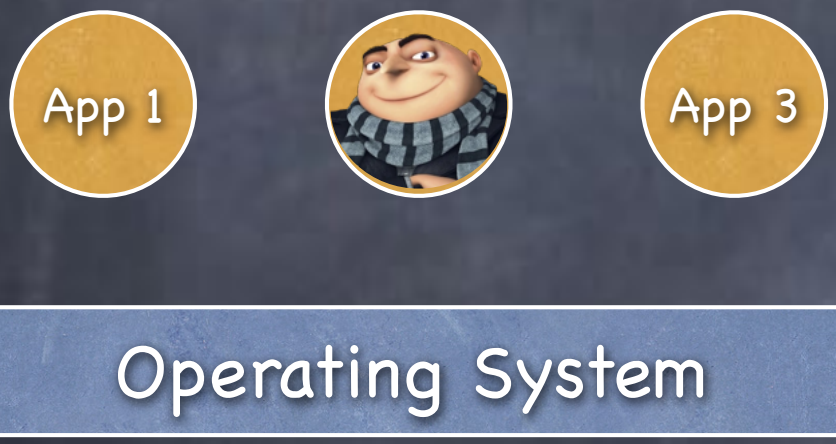
- Machines run (and thus OS must manage) multiple processes
  - how should the machine's resources be mapped to these processes?
- Enter the illusionist!

  - give every process the illusion of running on a private CPU
    - which appears slower than the machine's

    Virtualize the CPU
  - give every process the illusion of running on a private memory
    - which may appear larger (??) than the machine's

    Virtualize memory

Isolating Applications



- Buggy apps can crash other apps
- Buggy apps can crash OS
- Buggy apps can hog all resources
- Malicious apps can violate privacy of other apps

The bug that was a BUG

One of the worst bugs I ever had to deal with was in this game. Once the game player made it to the Colony, every so often the system would crash and burn at totally random times. You might be playing for ten minutes when it happened or ten hours, but it would just die in a totally random way

There was a slow-moving slug like creature that knew how to follow the game player's trail. When it came across another creature, rather than bouncing off and risk losing the trail, I made it so that it would destroy the other creature and stay on target to find you. This worked great, except that on some rare occasions, this slug could do to a wall what it did to the other creatures. That is, it could delete it. This meant that the virtual door was now open for this creature to explore the rest of the RAM on the Macintosh, deleting and modifying it as it went along. Of course, it was just a matter of time before it found some juicy code. In other words, the bug was a REAL bug.

David A. Smith

http://www.croquet.zone/2005/02/my-colony-memoir.html

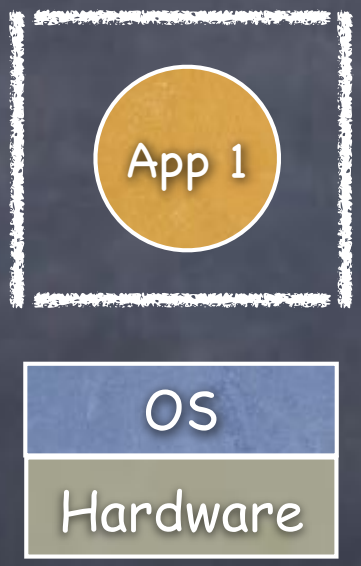
Super Mario Land 2

Isolating Applications

Reading and writing memory, managing resources, accessing I/O...

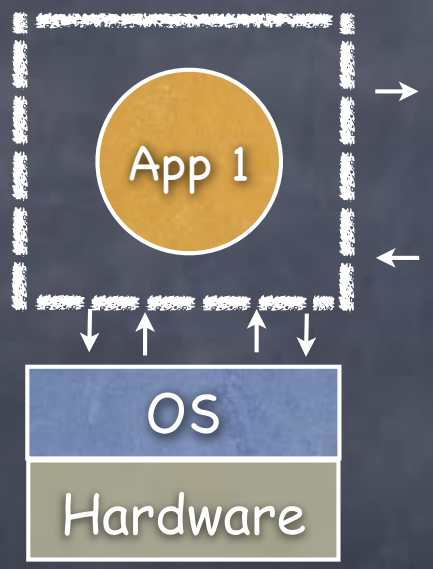
- Buggy apps can crash other apps
- Buggy apps can crash OS
- Buggy apps can hog all resources
- Malicious apps can violate privacy of other apps
- Malicious apps can change the OS

The Process, Refined



- A running program with restricted rights
- The enforcing mechanism must not hinder functionality
  - still efficient use of hardware
  - enable safe communication

The Process, Refined



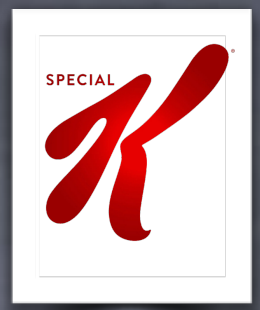
- A running program with restricted rights
- The enforcing mechanism must not hinder functionality
  - still efficient use of hardware
  - enable safe communication

Mechanism and Policy

- Mechanism
  - enables a functionality
- Policy
  - determines how that funtionality should be used

Mechanisms should not determine policies!

Enters the OS Kernel

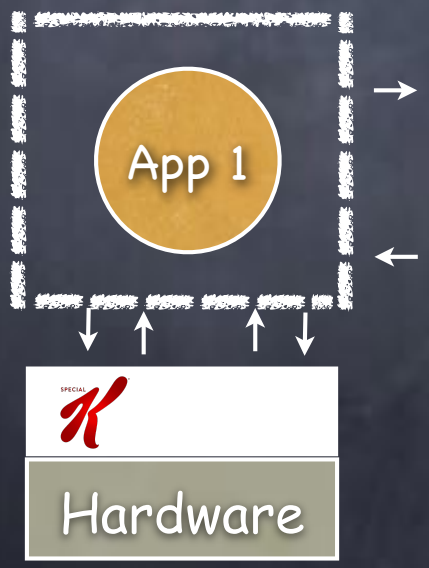


- Lowest level of OS code running on a system
- Kernel is trusted with full access to all hardware capability
- All other software (OS or applications) is considered untrusted

Untrusted	Applications
	Rest of the OS
Trusted	Kernel

How can the OS Enforce Restricted Rights?

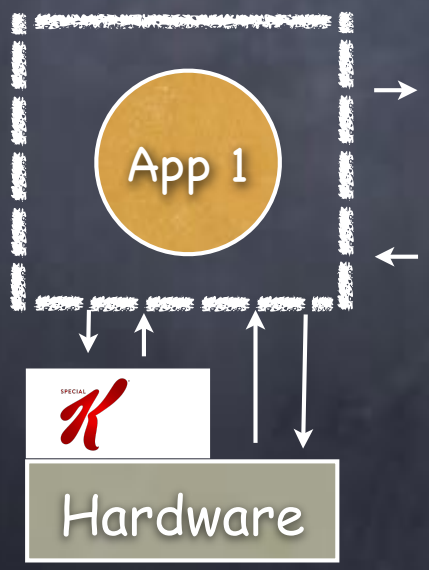
- Easy: kernel interprets and checks each instruction from apps (and untrusted OS)



  - slow
  - many instructions are safe: do we really need to involve the kernel?

How can the OS Enforce Restricted Rights?

Mechanism: Dual Mode Operation



- hardware to the rescue: use a bit to enable two modes of execution:
  - in user mode, processor only executes a limited (safe) set of instructions (checked by processor)
  - in kernel mode, no such restriction
- only OS kernel trusted to run in kernel mode

Amongst our weaponry are such diverse elements as...

- To support dual-mode operation:
  - Privileged instructions
    - in user mode, no way to execute potentially unsafe instructions
  - Memory isolation
    - in user mode, memory accesses outside a process' memory region are prohibited
  - Timer interrupts
    - ensure kernel will periodically regain control from running process

I. Privileged instructions

- Set mode bit
- I/O ops
- Memory management ops
- Disable interrupts
- Set timers
- Halt the processor

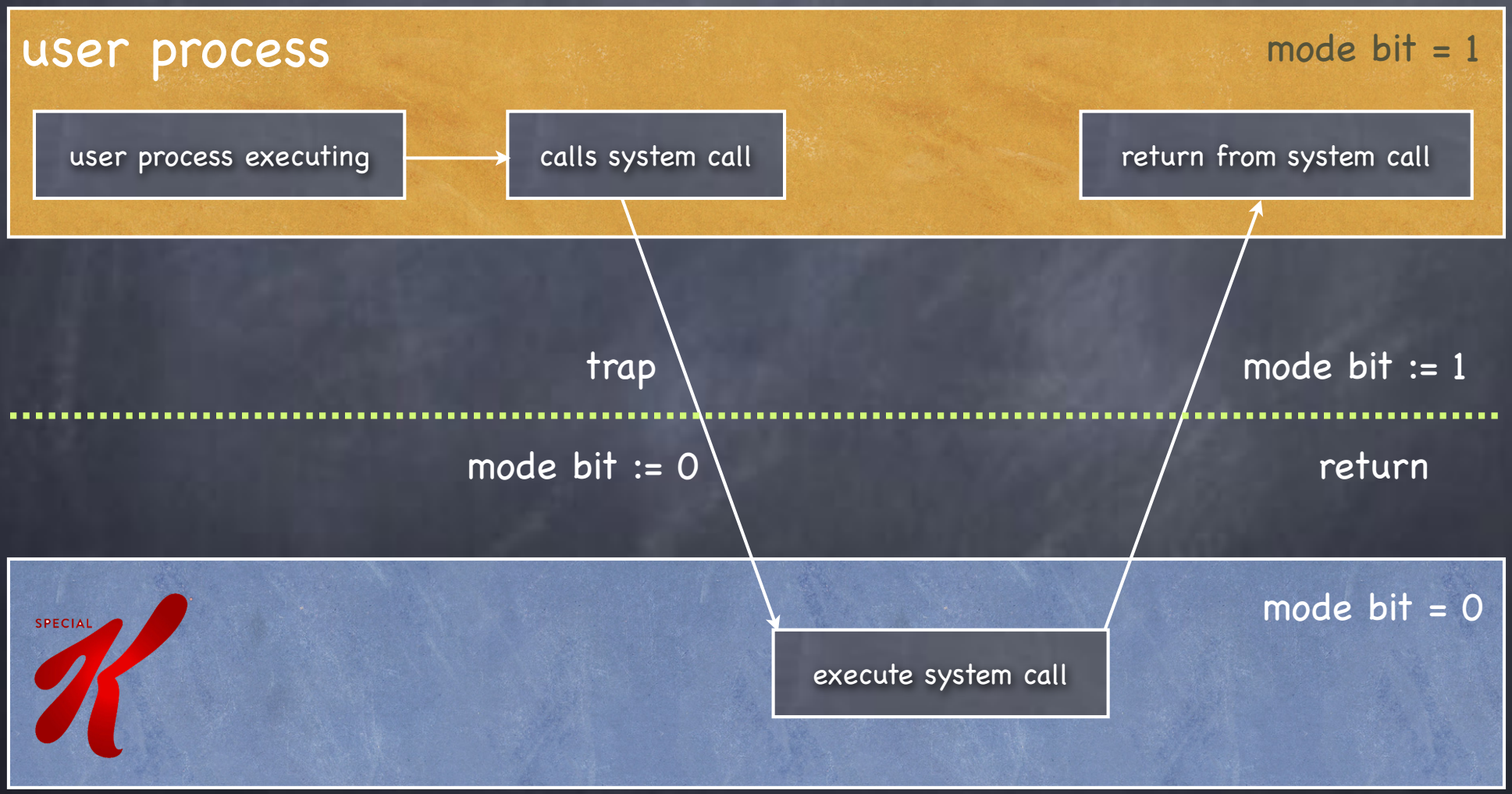
I. Privileged instructions

- But how can an app do I/O then?
  - ask for permission!
    - system calls achieve access to kernel mode only at specific locations specified by OS
- Executing a privileged instruction while in user mode (naughty naughty...) causes a processor exception....
  - ...which passes control to the kernel at specific locations (exception dependent) where appropriate handlers are invoked

I. Privileged instructions

- Set mode bit
- I/O ops
- Memory management ops
- Disable interrupts
- Set timers
- Halt the processor
- Set location of interrupt vector

Crossing the line



II. Memory Protection

Step 1: Virtualize Memory

- Virtual address space: set of memory addresses that process can name
  - CPU works with virtual addresses
- Physical address space: set of memory addresses supported by hardware

Virtual address space