Interrupts

Peter J. Denning

What are Interrupts?

- Normal execution of process may be interrupted by events beyond its control
  - Errors – conditions **inside** the computation that invalidate it
  - External signals –conditions **outside** that require immediate attention

- Called Exceptional Conditions
  - Exception – the act of responding to an exceptional condition
  - Interrupt – the signal from a sensor telling the OS to take an exception

Examples

- Errors
  - divide by zero
  - overflow
  - array index out of bounds
  - memory protection violation
- External signals
  - time slice end
  - page fault
  - disk completion
  - microkernel system call
  - keyboard
  - packet arrival

Architecture

- **Sensors** – hardware that detects an exceptional condition
  - divide by zero -- detected in ALU
  - Overflow – detected in ALU
  - array index out of bounds – detected by MMU
  - memory protection violation – detected by MMU
  - time slide end – detected by timer in CPU
  - disk completion – detected by the disk controller
  - microkernel system call – detected in CPU by instruction decoder
  - Keyboard – detected when key is pressed
  - packet arrival – detected by the network card

Architecture (2)

- **Selector** – Circuit in the CPU that receives signals from all sensors and selects the signal of highest priority
  - Conditions numbered 0,1,2,…,S-1 (S=16 is common) with 0 the highest priority and S-1 the lowest
  - If signals i and j are both on, with i<j, selector chooses i and ignores j
  - Remembers ignored signals until they are selected and dispatched
  - When sensor i is selected, selector outputs the code sel=i and sets the interrupt ready bit irdy=1

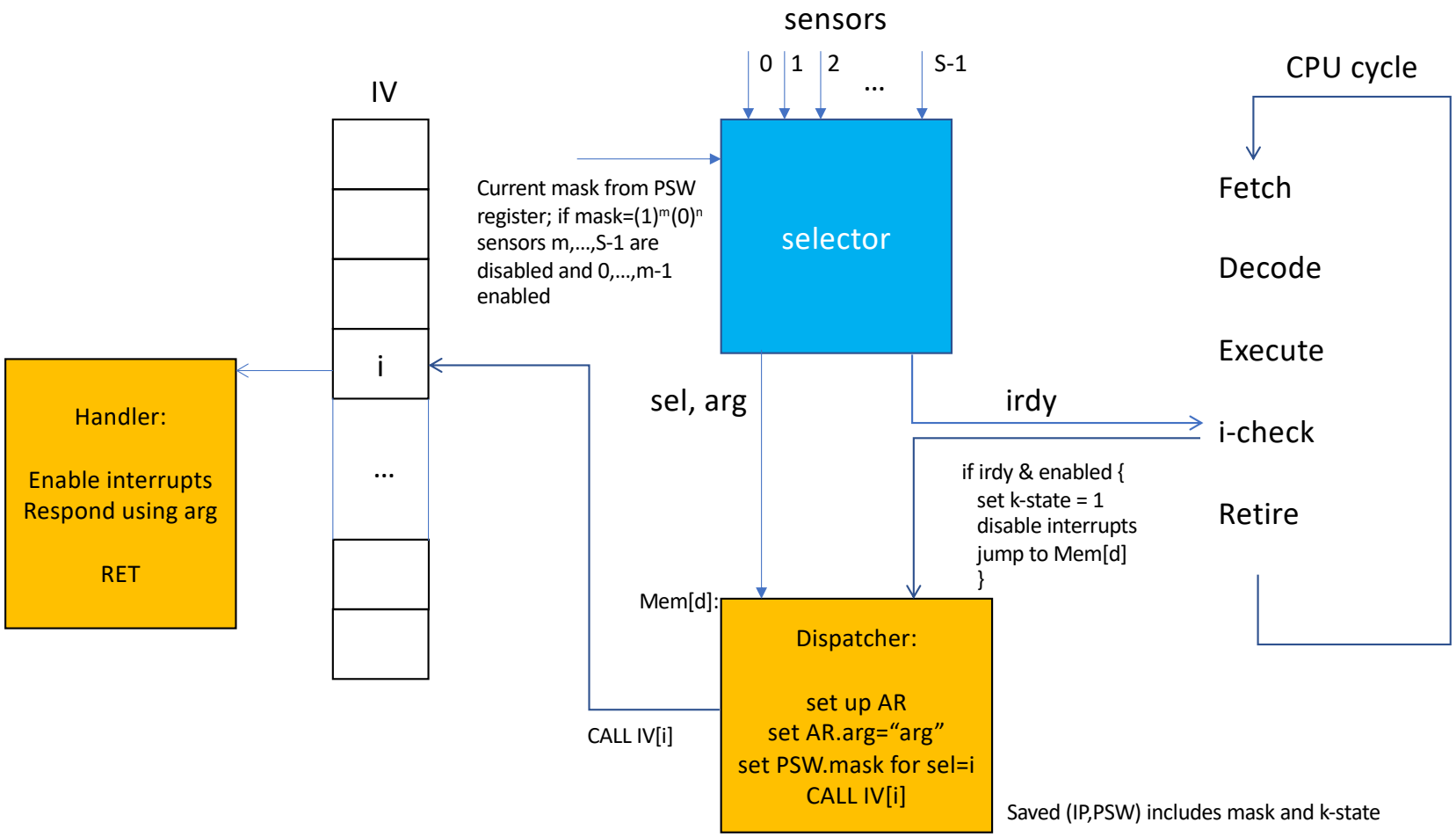
- **Interrupt check** – a step in each CPU fetch-execute cycle to see if irdy=1. If so, jump the CPU to run the dispatcher while turning off interrupts and entering kernel mode.

- **Dispatcher** – a short software routine that sets up activation record (AR) for the proper interrupt handler (IH) software and calls it.

- **Handling** –IH executes its function, completes with RET (return) instruction, which pops the AR from the stack and restores execution of the process originally interrupted. Most handlers turn interrupts back on, so that higher priority interrupts can take precedence.

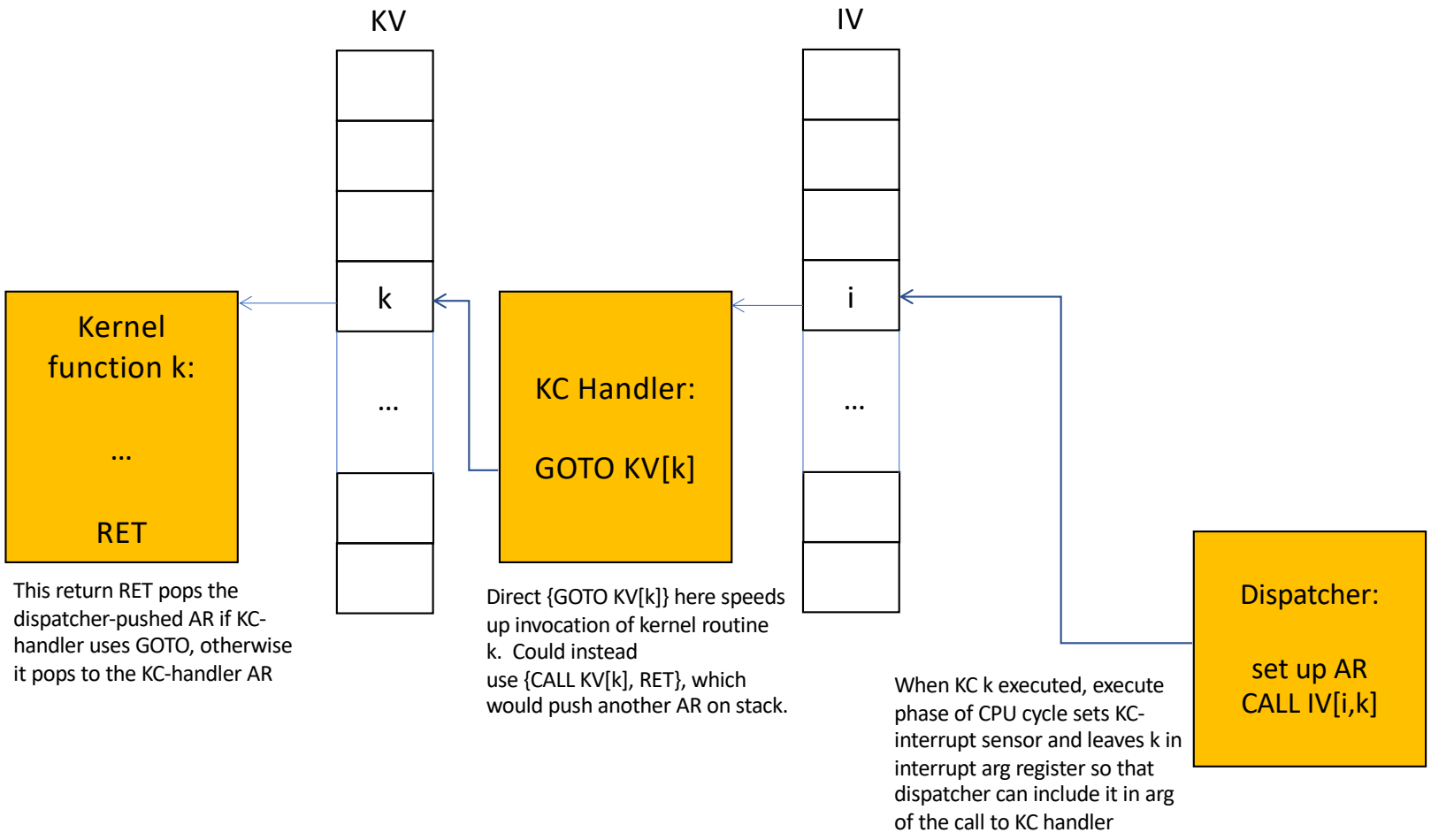
Architecture (3)

- **Priority (mask)** – A code indicating which of incoming sensor signals the selector should act on. Corresponds to the priority of the handler. Hides pending interrupt signals from sensors of lower priority than the executing handler.
  - Typically mask has S bits of the form prefix of 1s followed by suffix of 0s
  - When no interrupts are being handled, mask is all 1s (all sensors enabled)
  - User code operates with the all-1s mask
  - Example: Handler for sensor 3 has mask 111000000000000 meaning sensors 0, 1, 2, are enabled and all others ignored
  - Mask stored in the PSW of the CPU
- **Interrupt Argument (arg**) – a code generated by selector to be passed as a parameter to the handler
  - for example, the virtual address causing a page fault interrupt
- **Interrupt Vector (IV)** – list of entry points for IHs; stored in kernel memory



Kernel call – Microkernel

- Micro kernel routines require extra security – they are run in kernel model and have access to all of kernel memory
- Micro kernel routines, and only those routines, run in kernel mode
- Force entry to kernel routine at its entry point (protected entry)
- Let Ek = memory address of first instruction of a kernel routine k
- Ordinary CALL Ek not allowed because user could change Ek to something else
- Instead create kernel vector KV[k] = Ek meaning that kernel routine with ID code k has entry point Ek. KV private in kernel space.
- Instruction KC k implements “CALL KV[k]”



Supervisor call – User kernel

- User kernel routines run in user mode. Therefore there is no need to use the KC instruction to enter them.
- Instead, the entry points of the user kernel API can be made available to the compiler, which can use ordinary procedure calls for them.
- Some operating systems want to use protected entry for user kernel as well as micro kernel, as added protection against malware.
- In that case the system would be set up to use KC for all kernel calls. The KV would be extended to include both the entry point and the kernel mode of that routine. The KC-handler would set the kernel mode of the called kernel routine accordingly.