In the last class, we discussed how a C application could jump in the middle of an instruction to execute a privilege instruction even though compiler never generates a privileged instruction. Let us talk about whether this is possible in managed languages like Java, which offers memory safety.

In Java pointers do not exists and so we can't do pointer arithmetic on a function pointer. In Java, methods are the citizen of a class. An application can invoke a method defined in a class through the class object. In Java, we can't do unsafe typecast of an object to overwrite the method pointer with something else. In Java, the memory is garbage collected so we can't exploit use after free bugs. An out of bound array access throws a runtime exception in Java, so return address corruption is not possible. Because of the reasons listed above an application can never execute arbitrary code.

Let us talk about memory isolation. In Java memory can only be accessed via objects. A Java application cannot access memory outside an object. The virtual memory is not abstracted to the application. Because of these reasons we don't need additional hardware support (page tables/segmentation) for process isolation if the allocator is trusted.

If an OS is itself written in Java, and only allows Java applications, then there is no need for page tables or privilege rings. Let us talk about an OS called Java OS, which is purely written in Java, and only allows Java processes. The kernel itself is a Java process. In Java OS, both kernel and untrusted applications execute in ring-0.

How does a system call look like in Java OS?
To execute the system call we need to switch to the kernel process. Switching to a different process doesn't require page table switch. The system call handler is merely a context switch routine that copies system call arguments in kernel address space and directly jumps to the system call handler. Notice, that we have saved two ring transitions that we otherwise have to do in Linux or Windows.

How does IPC work in Java OS?
IPC is similar to the system call handler. The context switch method needs to invoke the receive routine in the target process after copying messages. This is better compared to Linux and Windows, where we have to do two ring transitions and one context switch (including page table switch) to schedule the target routine.

Device drivers still share the same address space with OS kernel.
Isolate device drivers to separate Java processes. Device drivers can talk to kernel via system calls.

Why are existing operating systems not written in Java?
Java is not an efficient language particularly due to garbage collection that can cause arbitrary latencies.

How can we design a secure OS in C?
One parameter for security comes from the amount of code we need to trust for the correct functioning of your system. The basic idea behind microkernel is kernel should remain as small as possible. In the microkernel design, kernel subsystems are isolated from each other. The microkernel runs each kernel subsystem in a separate process, and the OS mainly implements IPC. The system calls are replaced with IPC, e.g., a file read system call is replaced with an IPC request to the filesystem process. Microkernels are not good for performance because IPC is much slower than system calls.