**User program**

- Pintos can run normal C programs, as long as they fit into memory and use only the system calls you implement. Notably, malloc() cannot be implemented because none of the system calls required for this project allow for memory allocation. Pintos also can't run programs that use floating point operations, since the kernel doesn't save and restore the processor's floating-point unit when switching threads.

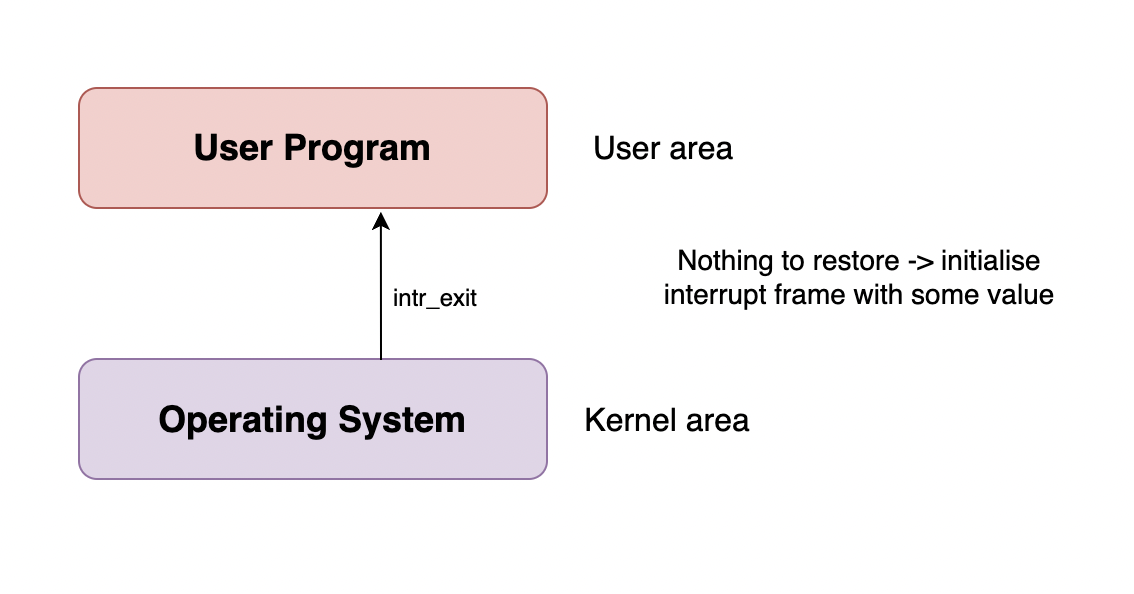
**Start process**

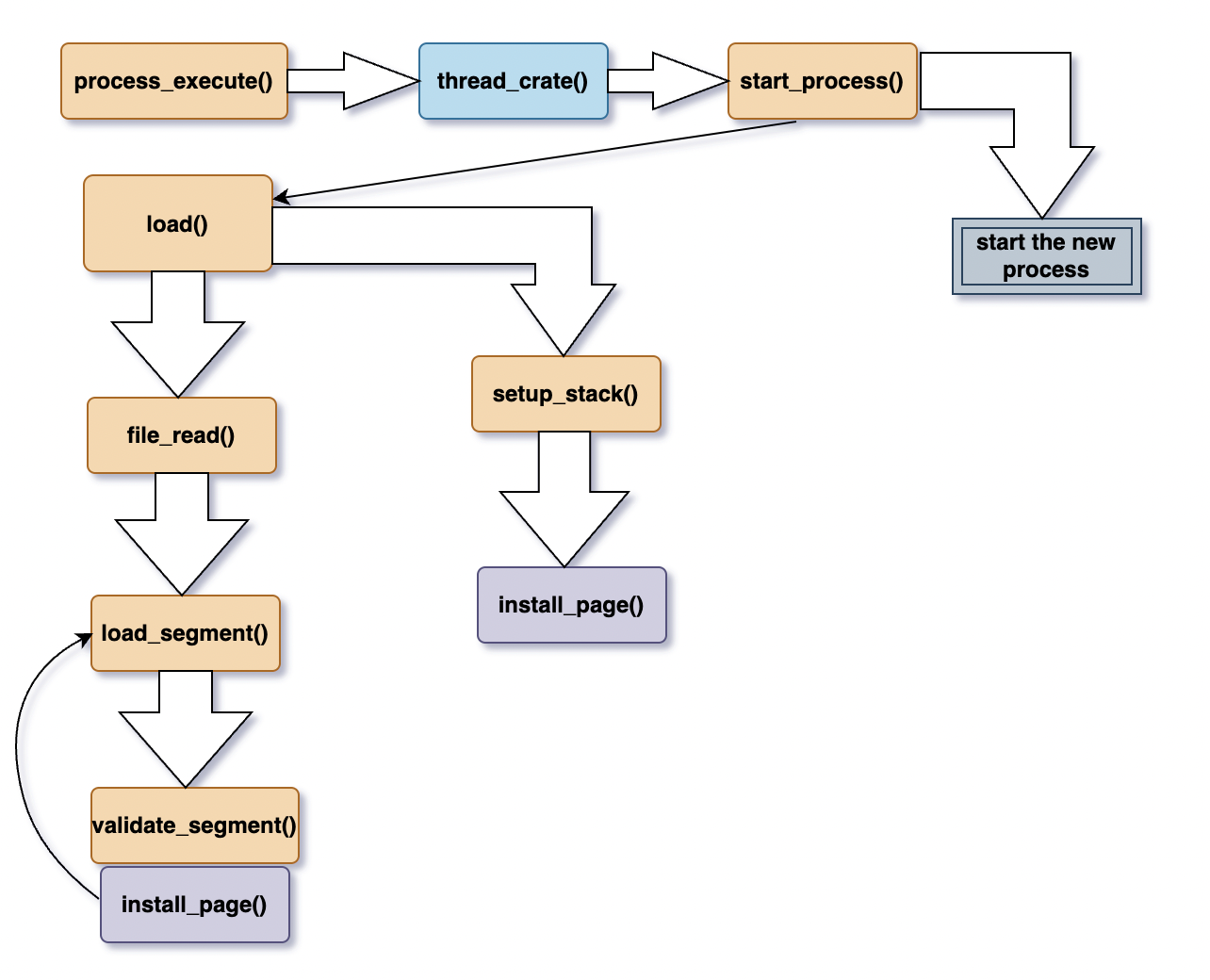
- Allocate interrupt frame.

- Load program and initialise interrupt frame and user stack.

- Setup arguments at the user stack.

- Jump to the user program through (interrupt\_exit).





**How User programs work**

Pintos can run normal C programs, as long as they fit into memory and use only the system calls you implement.

Pintos can load ELF executables with the loader provided for you in userprog/process.c. ELF is a file format used by Linux, Solaris, and many other operating systems for object files, shared libraries, and executables. You can actually use any compiler and linker that output 80x86 ELF executables to produce programs for Pintos.

**Virtual Memory Layout**

Kernel virtual memory is global. It is always mapped the same way, regardless of what user process or kernel thread is running. In Pintos, kernel virtual memory is mapped one-to-one to physical memory, starting at PHYS\_BASE. That is, virtual address PHYS\_BASE accesses physical address 0, virtual address PHYS\_BASE + 0x1234 accesses physical address 0x1234, and so on up to the size of the machine's physical memory.

A user program can only access its own user virtual memory. An attempt to access kernel virtual memory causes a page fault, and the process will be terminated. Kernel threads can access both kernel virtual memory and, if a user process is running, the user virtual memory of the running process. However, even in the kernel, an attempt to access memory at an unmapped user virtual address will cause a page fault.

**Accessing User Memory**

As part of a system call, the kernel must often access memory through pointers provided by a user program. The kernel must be very careful about doing so, because the user can pass a null pointer, a pointer to unmapped virtual memory, or a pointer to kernel virtual address space (above PHYS\_BASE). All of these types of invalid pointers must be rejected without harm to the kernel or other running processes, by terminating the offending process and freeing its resources.