Thread, Processes, and Address Spaces

- Topics
  - What is a process?
- Learning Objectives:
  - Explain the manifestation of a thread, process, and address space within the operating system.
  - Compare and contrast the different possible mappings between user and kernel level threads.
  - Explain all the pieces that comprise an address space.

Process = Address Space + Thread(s) (1)

- A process is composed of two parts:
  - A part that keeps track of "stuff": Address space
  - A dynamic part: Thread
- Address space:
  - A "place" in which execution happens.
  - The set of addresses (e.g., memory locations) to which a running computation has access.
  - An address space can be physical (addresses map directly to locations in the hardware) or virtual (addresses are "make believe" but get translated into locations in hardware).
  - Address spaces provide protection boundaries.

Process = Address Space + Thread(s) (2)

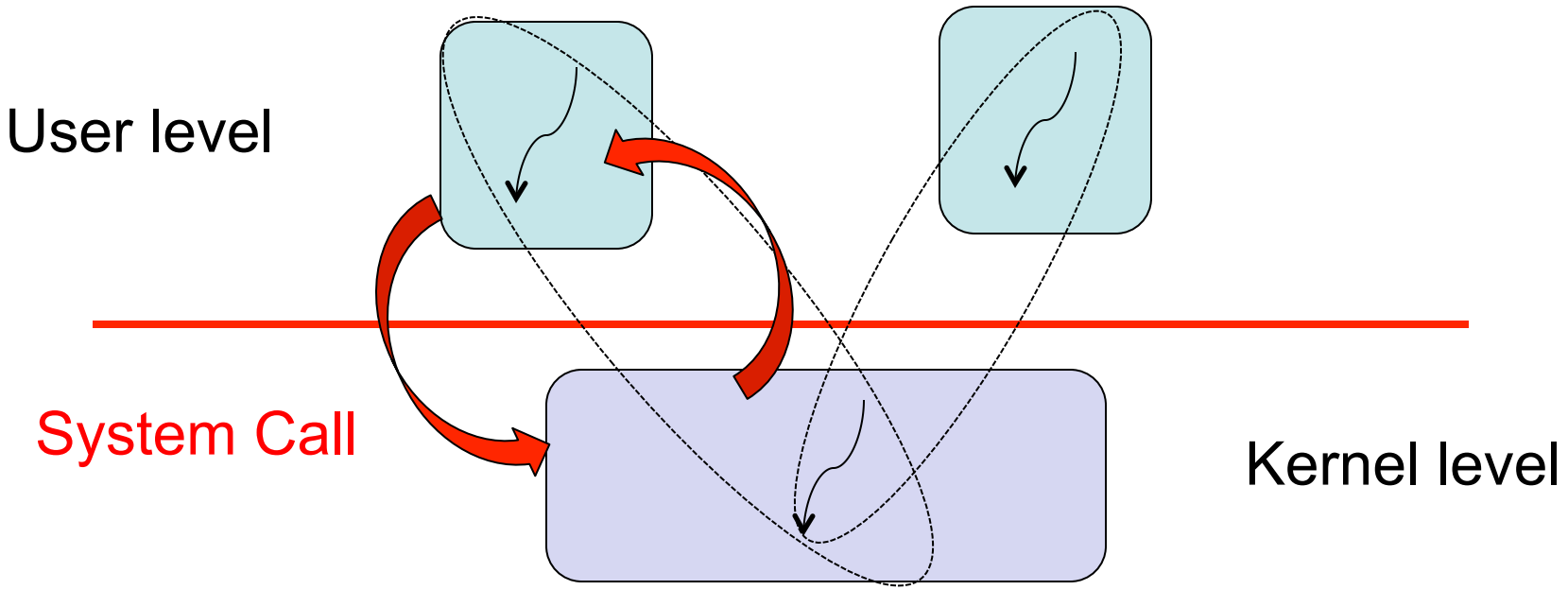
- A process is composed of two parts:
  - A static part: Address space
  - A dynamic part: Thread
- Thread:
  - A logical flow of control
  - Execution state
- A process has one address space and one or more threads in it.
- Threads share the address space, i.e., memory – that is why you need to synchronize access to memory between threads and (unless you go to great length) do not need to synchronize access to memory across processes.

User/Kernel Thread/Address Space

- User level processes can contain one or more threads.
- The operating system (kernel) can contain one or more threads.
- You can think of user processes and the kernel as running in their own address spaces.
  - The details sometimes get a little fuzzy and we'll talk more about that later, but for now, this is a reasonable model.
- There are a variety of ways to map user threads to kernel threads.

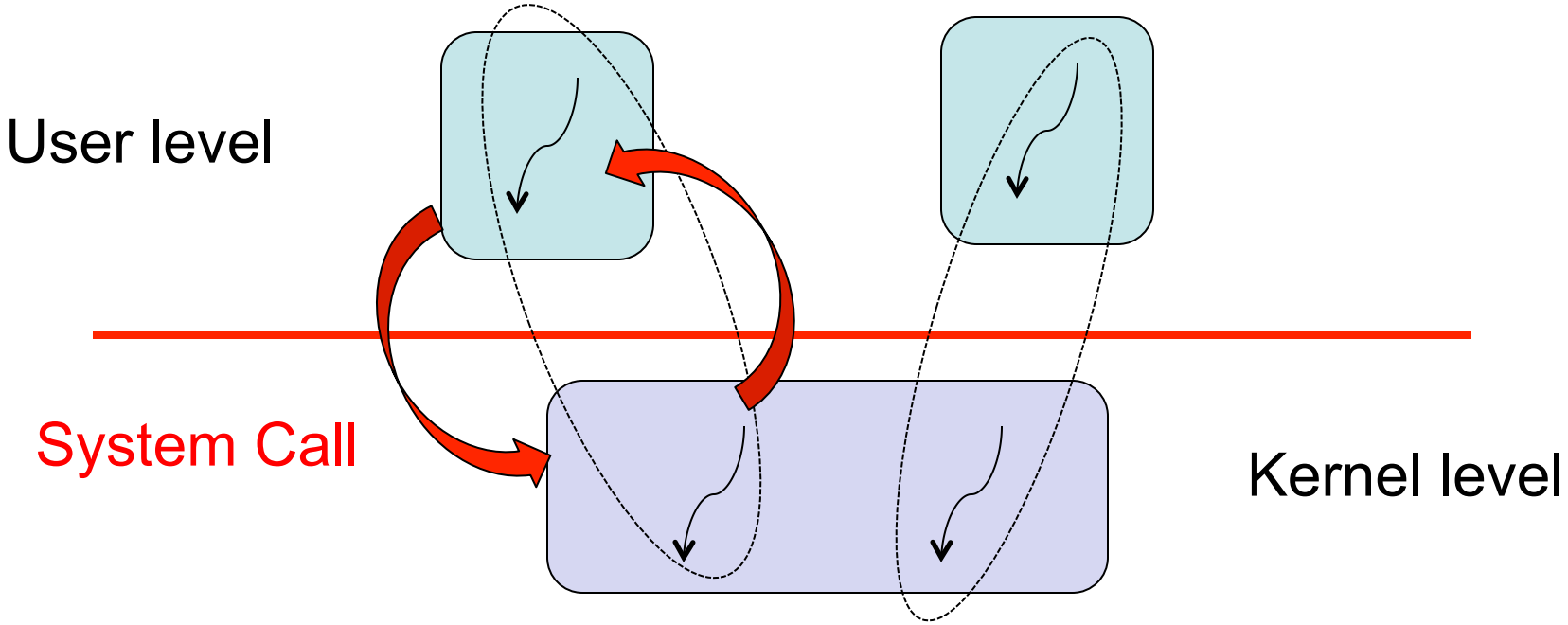
Architecture 1: Single-threaded processes and kernel

- Historically, both the OS and user processes were single-threaded.
- Design is easy!



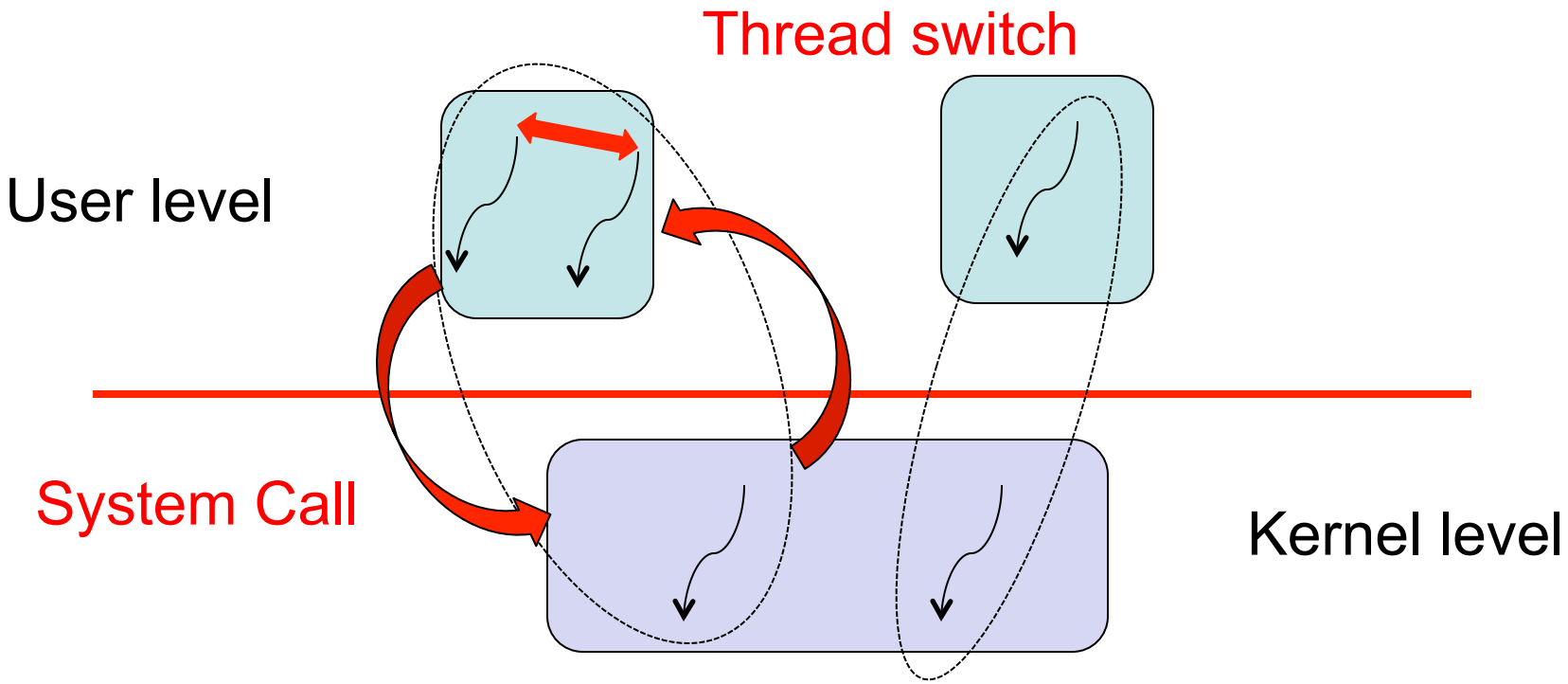
Architecture 2: Single-threaded processes; multi-threaded kernel

- Processes are still single-threaded.
- Design is easy: a process maps to a single OS thread.



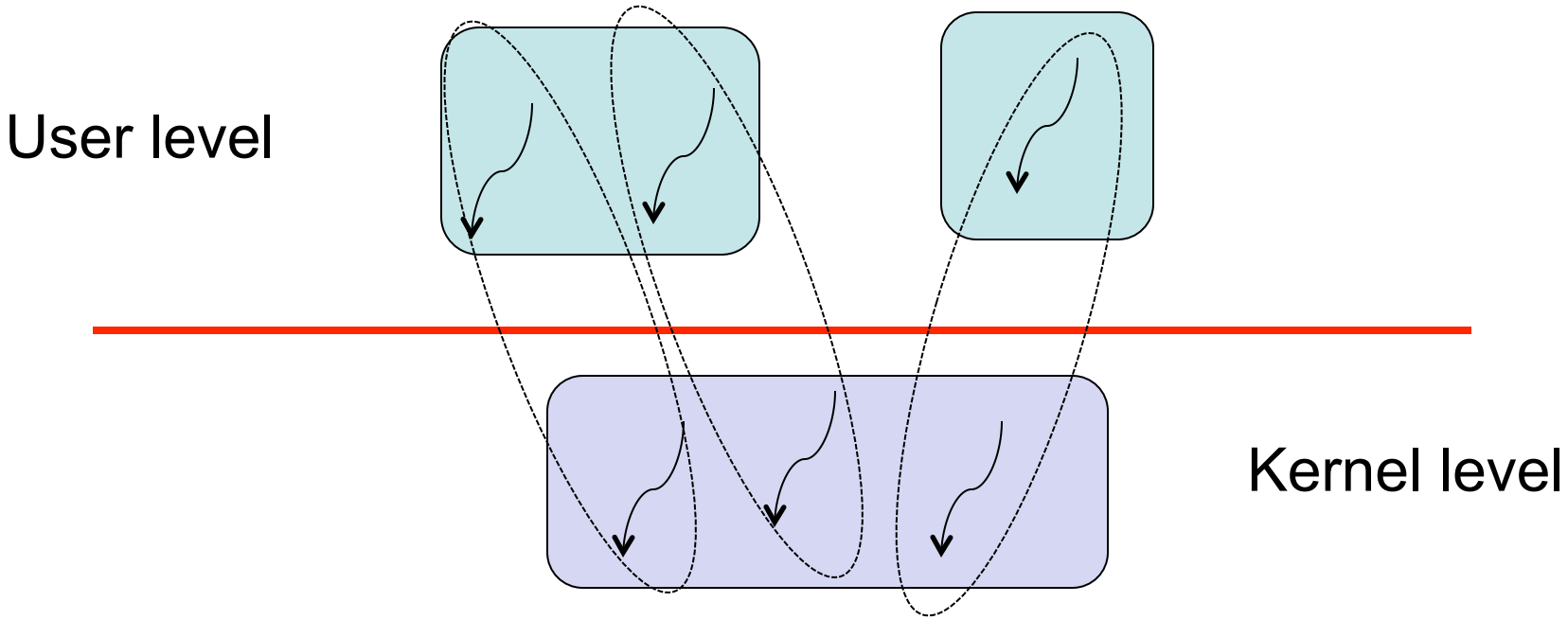
Architecture 3: Multithreaded process per kernel thread

- Process may be multi-threaded
- Each process maps to a single kernel thread
- Sometimes called user-level threads



Architecture 4: Multithreaded 1:1

- Processes may be multi-threaded
- Every user-level thread maps to a kernel thread



Architecture 5: Multithreaded N:M

- User processes are multi-threaded
- N user threads map to M kernel threads