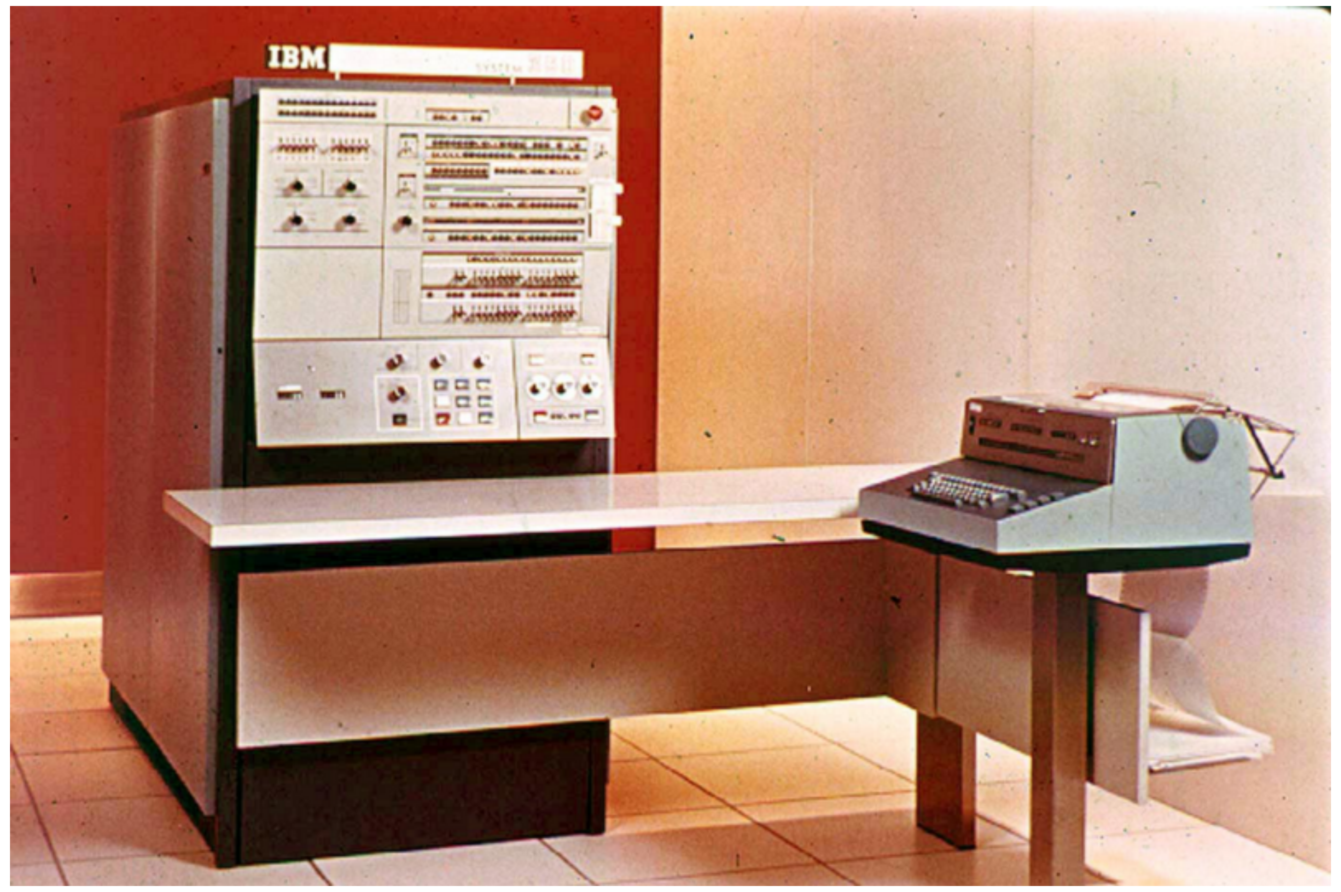
Review

- Running a program requires the code & stack segments in memory.
- Context = memory address space + stack pointer + instruction pointer
  - A CPU is in the context of a program if its instruction pointer and stack pointer registers point to the code & stack segments of the program.
- Context-switch means switching the context of a CPU to different programs by modifying its stack pointer and instruction pointer.

Big picture of context-switch



- The initial goal of operating systems is multi-tasking.
  - A naive way of multi-tasking is batch processing.
  - The concept of context-switch enables time-sharing multi-tasking.
    - There are different implementations of context-switch: user-level threads, kernel-level threads, processes

* Images from https://about.sourcegraph.com/blog/the-ibm-system-360-the-first-modular-general-purpose-computer/

Implementation comparison

		Switching stack pointer?	Switching instruction pointer?	Switching memory address space?	Switching kernel/user mode?
4411 P1	User-level Threads	Yes	No	No	No
Beyond 4411 P1	Kernel-level Threads	Yes	Yes	No	Yes
	Processes	Yes	Yes	Yes	Yes

Implementation comparison

		Switching stack pointer?	Switching instruction pointer?	Switching memory address space?	Switching kernel/user mode?
4411 P1	User-level Threads	Yes	No	No	No
Beyond 4411 P1	Kernel-level Threads	Yes	Yes	No	Yes
	Processes	Yes	Yes	Yes	Yes

Implementation comparison

		Switching stack pointer?	Switching instruction pointer?	Switching memory address space?	Switching kernel/user mode?
4411 P1	User-level Threads	Yes	No	No	No
Beyond 4411 P1	Kernel-level Threads	Yes	Yes	No	Yes
	Processes	Yes	Yes	Yes	Yes

Implementation comparison

		Switching stack pointer?	Switching instruction pointer?	Switching memory address space?	Switching kernel/user mode?
4411 P1	User-level Threads	Yes	No	No	No
Beyond 4411 P1	Kernel-level Threads	Yes	Yes	No	Yes
	Processes	Yes	Yes	Yes	Yes

Context-switch solves the problem of time-sharing multi-tasking.

Processes and threads implement the concept of context-switch.

Context-switch solves the problem of time-sharing multi-tasking.

Processes and threads implement the concept of context-switch.

Next problem: how do different processes/threads communicate?

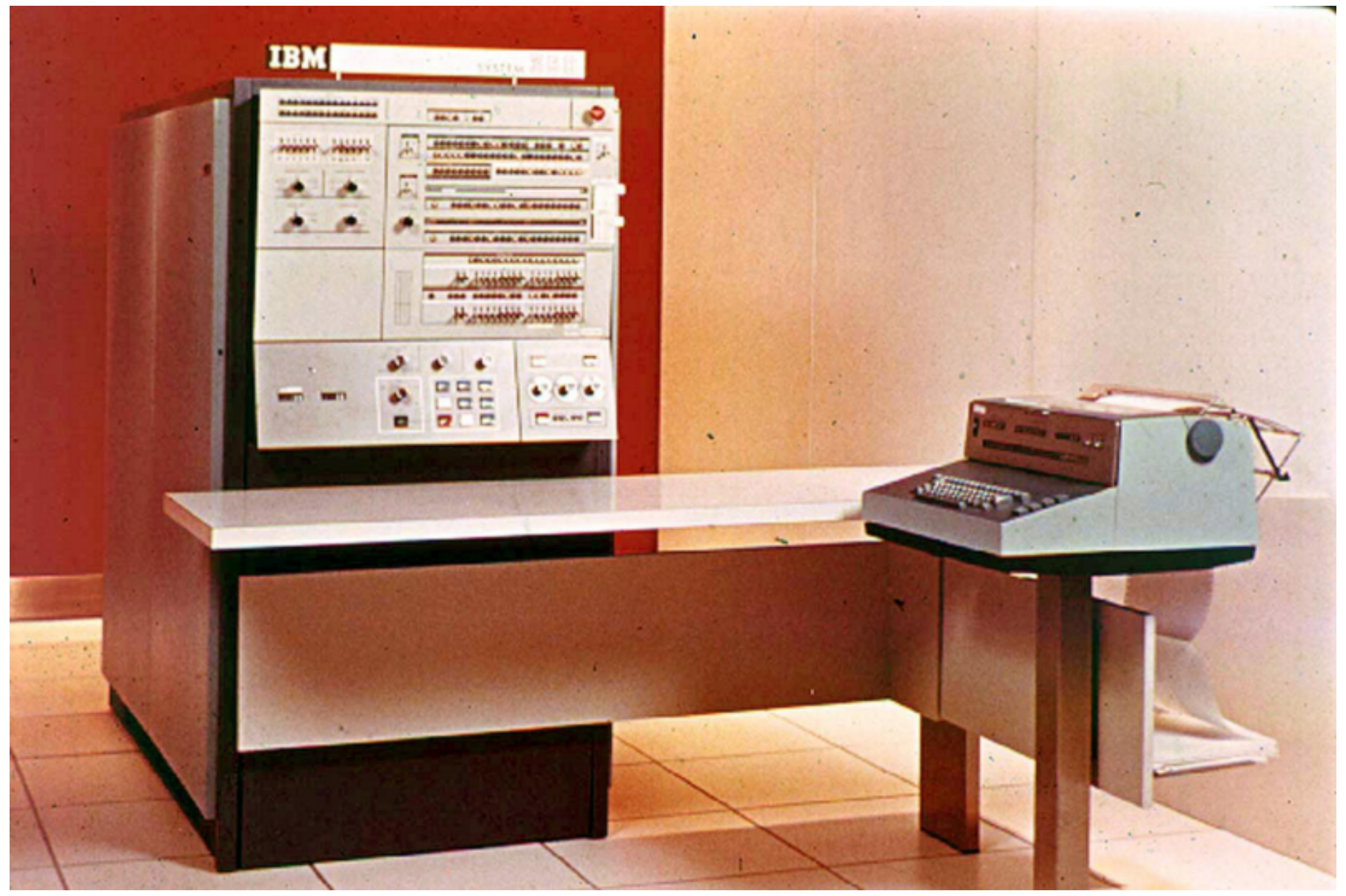
Next problem: how do different processes/threads communicate?

- For example, say there are 3 threads running my zoom together: one for user interface, one for microphone and one for camera.
  - When I click the “mute” button, the user interface thread should tell the microphone thread to stop recording.
  - The camera thread should continuously transfer video data to the user interface thread.

Interprocess communication (IPC)

- The terminology of this problem is IPC which is extensively studied in the operating systems literature. Performance is the key!
    - If IPC has poor performance, the camera thread cannot transfer video data to the user interface thread in time, leading to poor experience.
- Note: we will use the general term IPC to represent both communications among processes and communications among threads.

Historical representatives



- IBM 360 is a representative of time-sharing multi-tasking with context-switch.
- 1960s

- AT&T UNIX System V is a representative of interprocess communication (IPC).
- 1980s

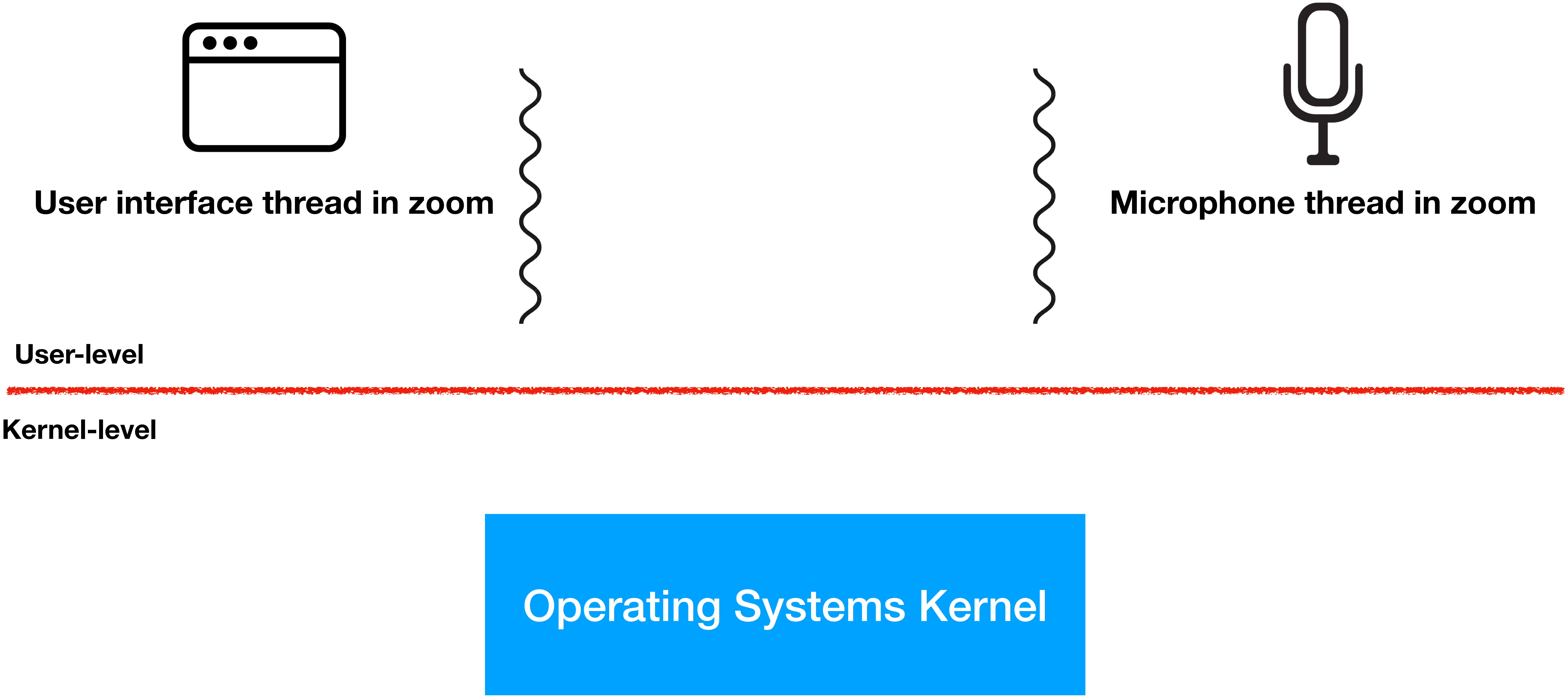
UNIX System V IPC

`System V IPC is the name given to three interprocess communication mechanisms that are widely available on UNIX systems: message queues, semaphore, and shared memory.`

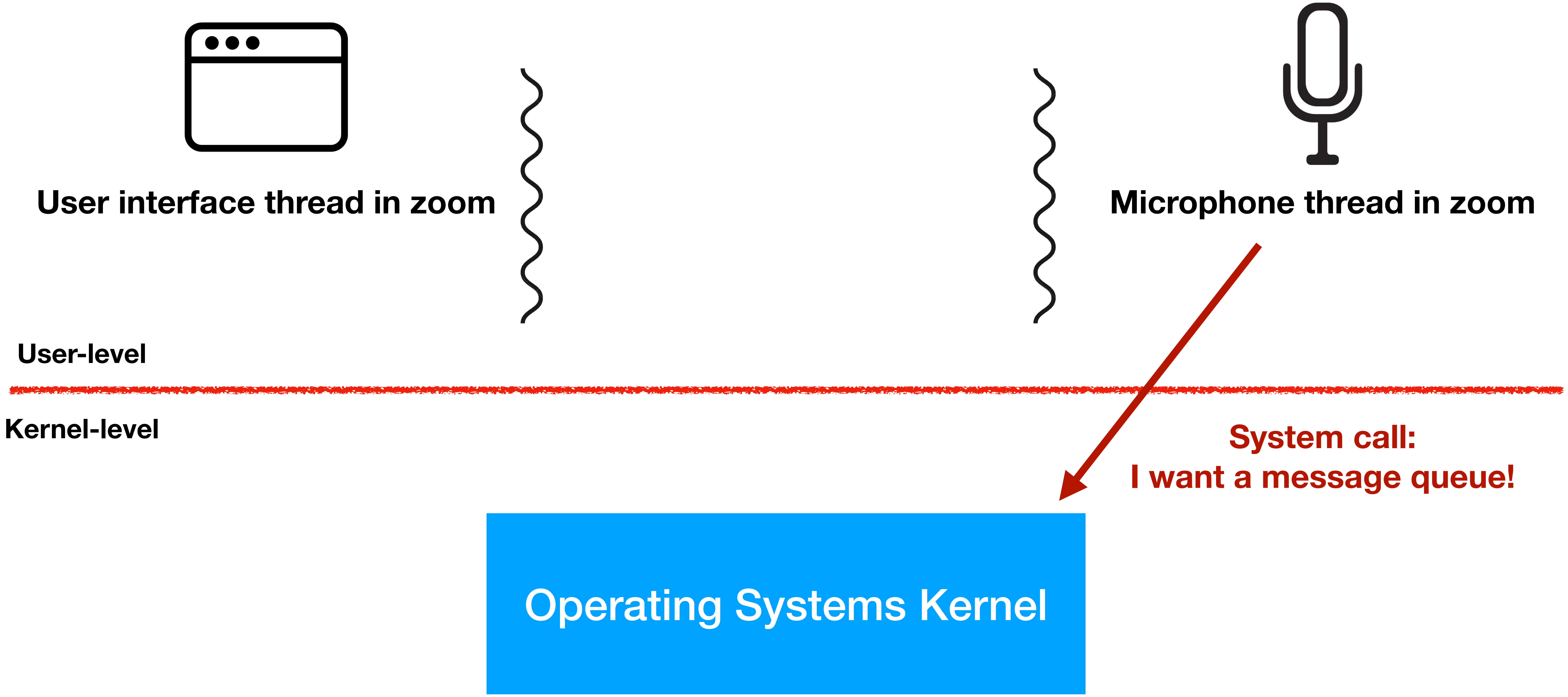
what you need to implement in 4411 P1

https://man7.org/linux/man-pages/man7/svipc.7.html

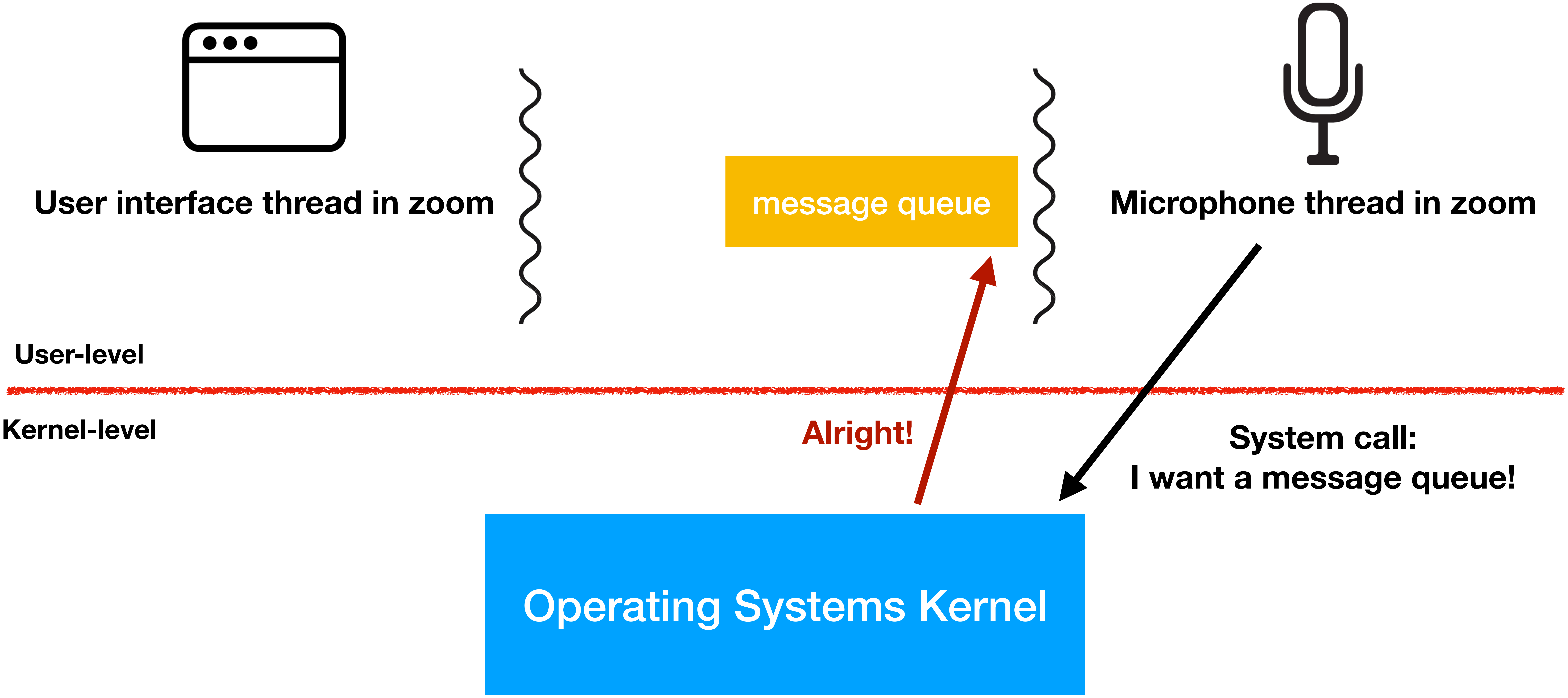
Message queue example



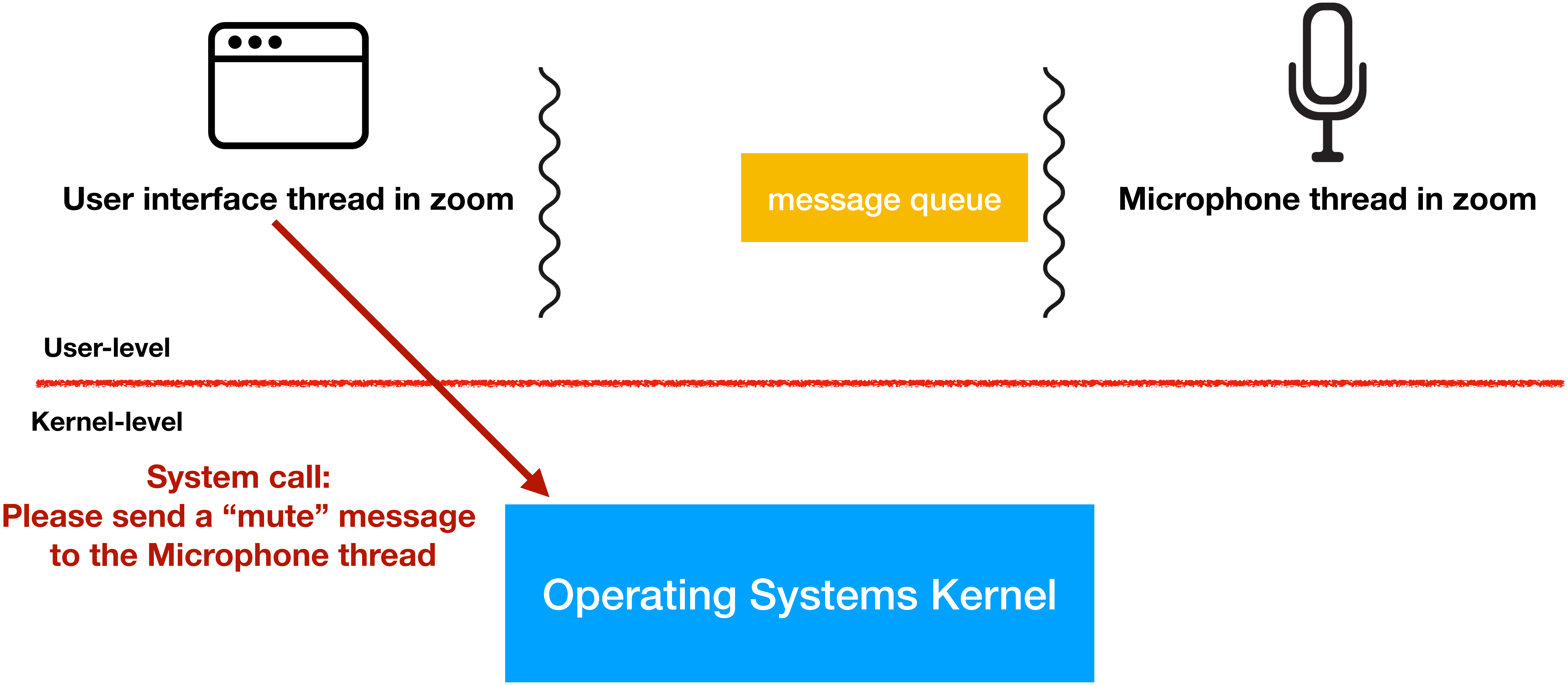
Message queue example



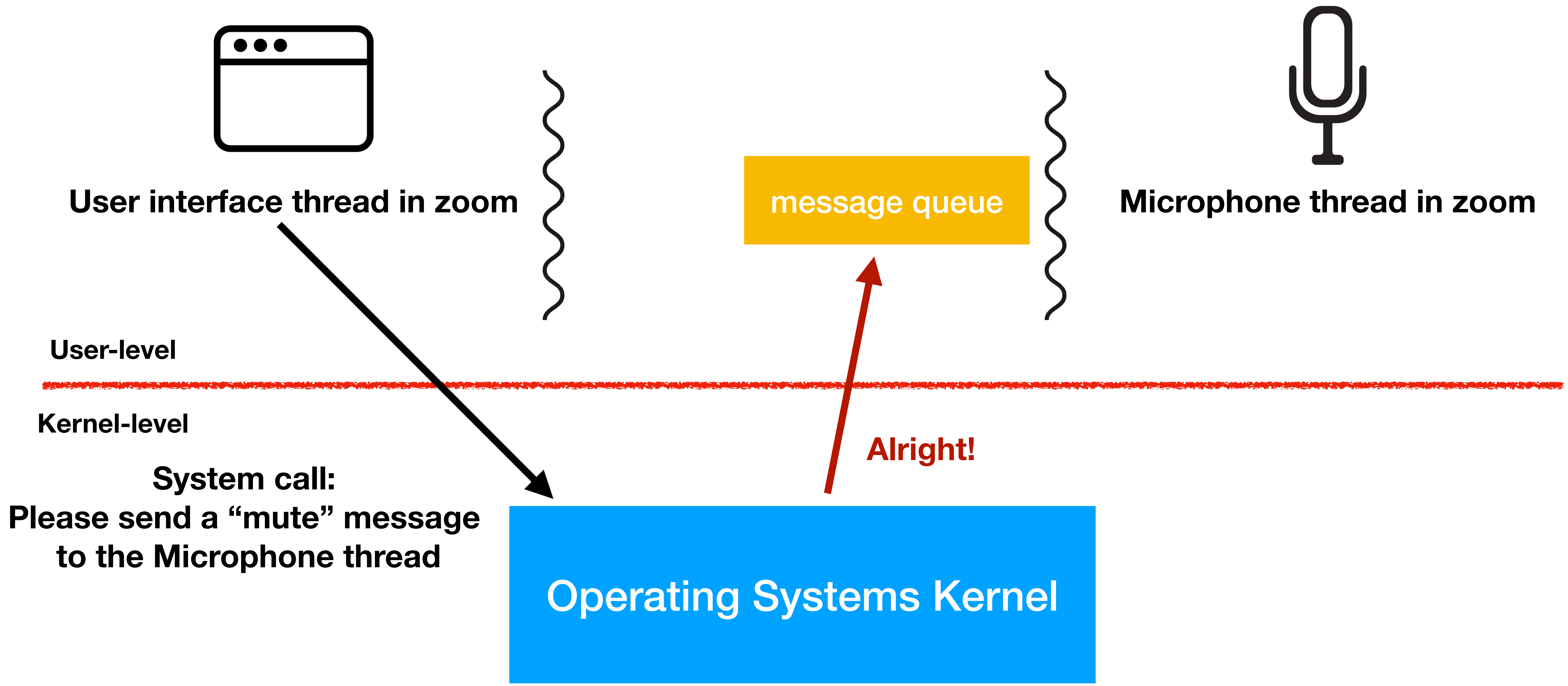
Message queue example



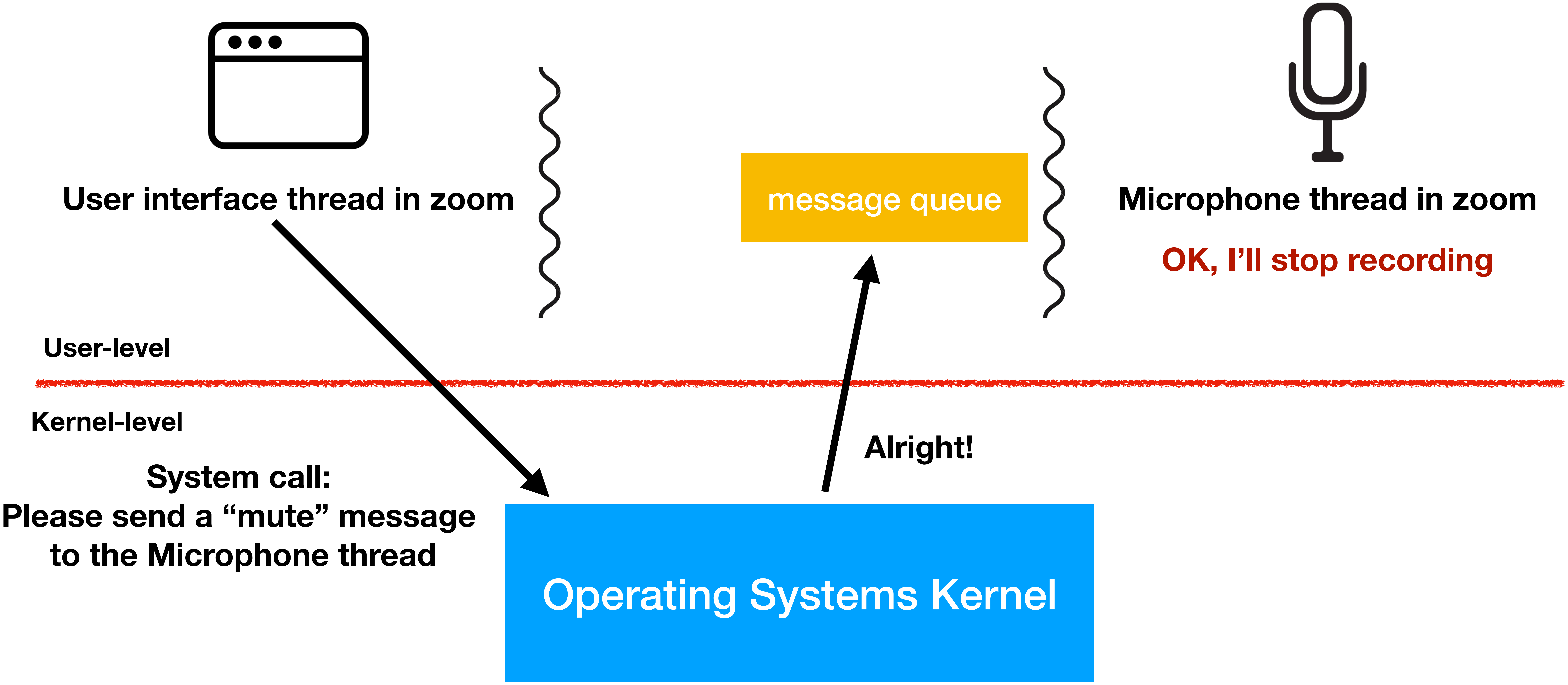
Message queue example



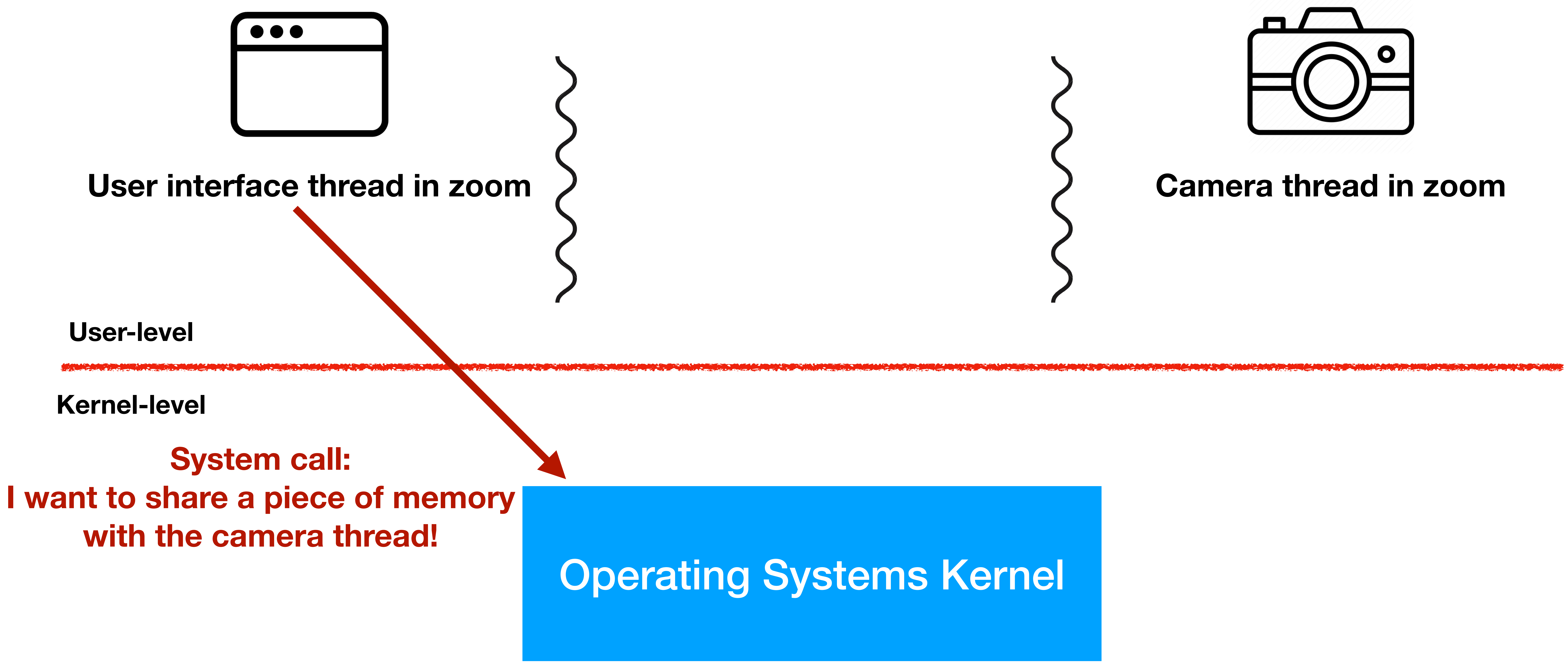
Message queue example



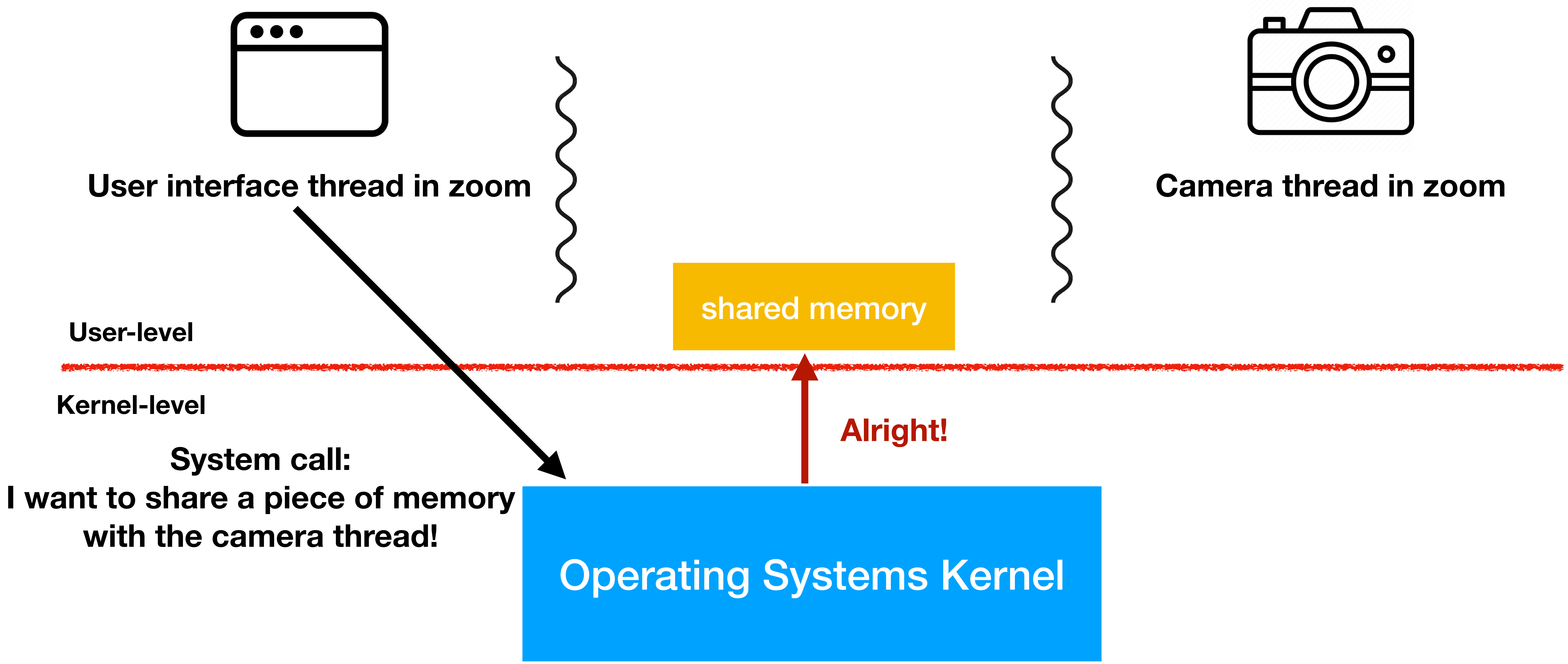
Message queue example



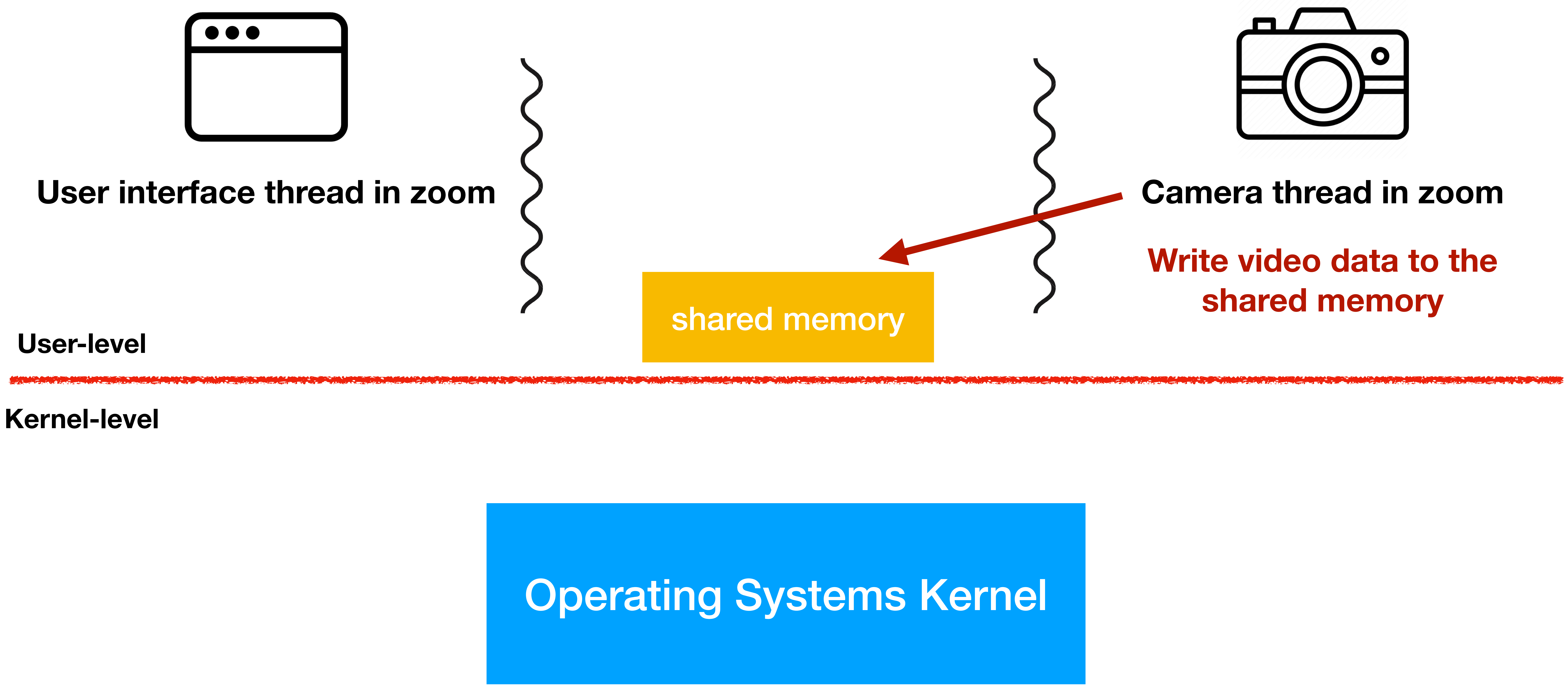
Shared memory example



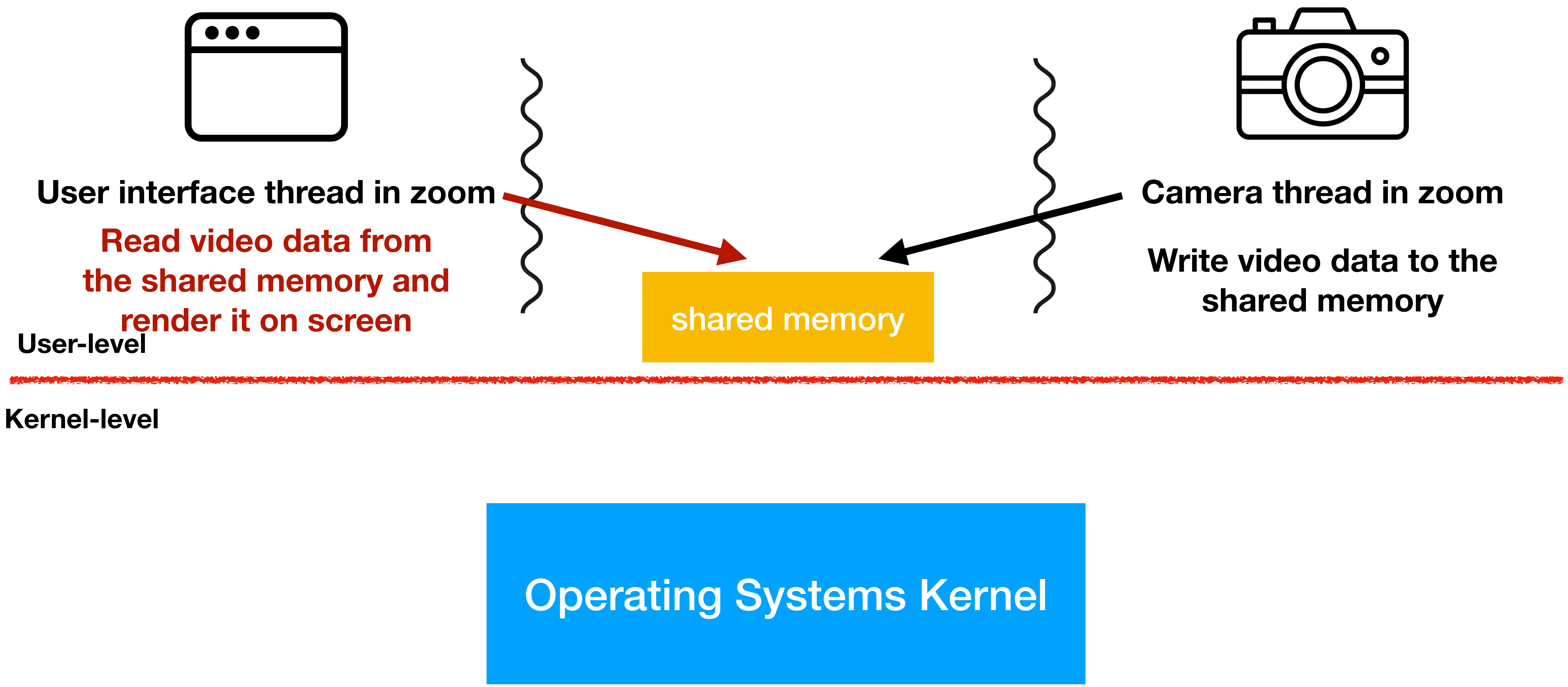
Shared memory example



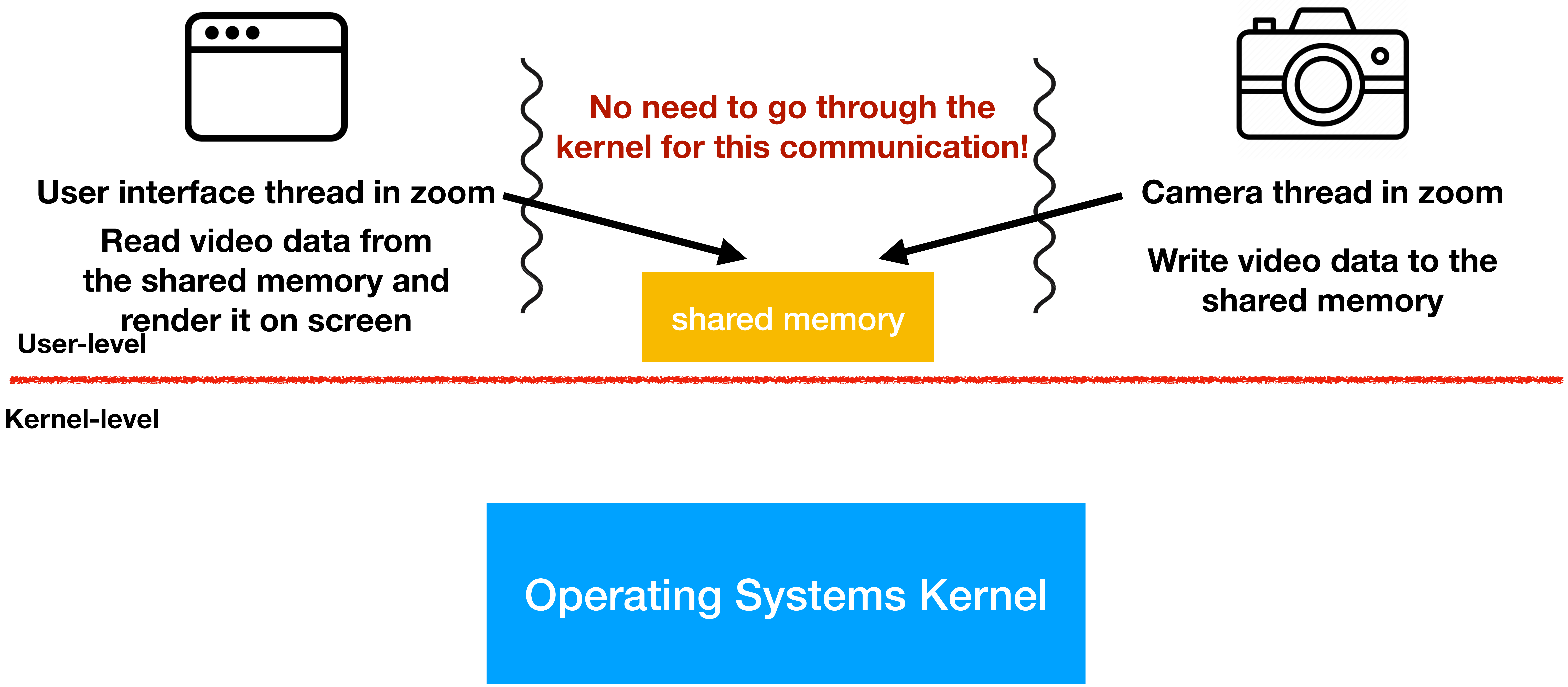
Shared memory example



Shared memory example



Shared memory example



Lesson: shared memory has better performance than message queues because communications get around the kernel.

The third IPC mechanism: semaphores

```
System V IPC is the name given to three interprocess
communication mechanisms that are widely available on
UNIX systems: message queues, semaphore, and shared
memory.
```

✅ ✅

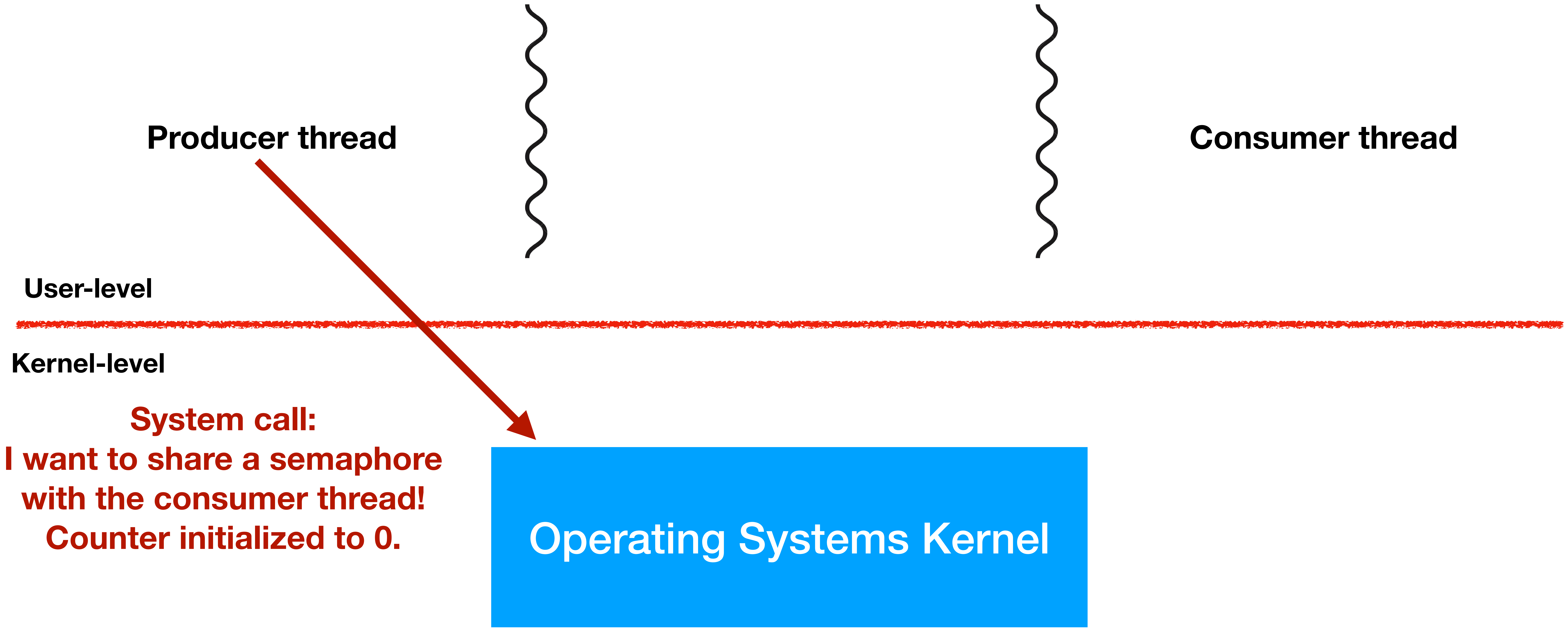
↑ what you need to implement in 4411 P1

https://man7.org/linux/man-pages/man7/svipc.7.html

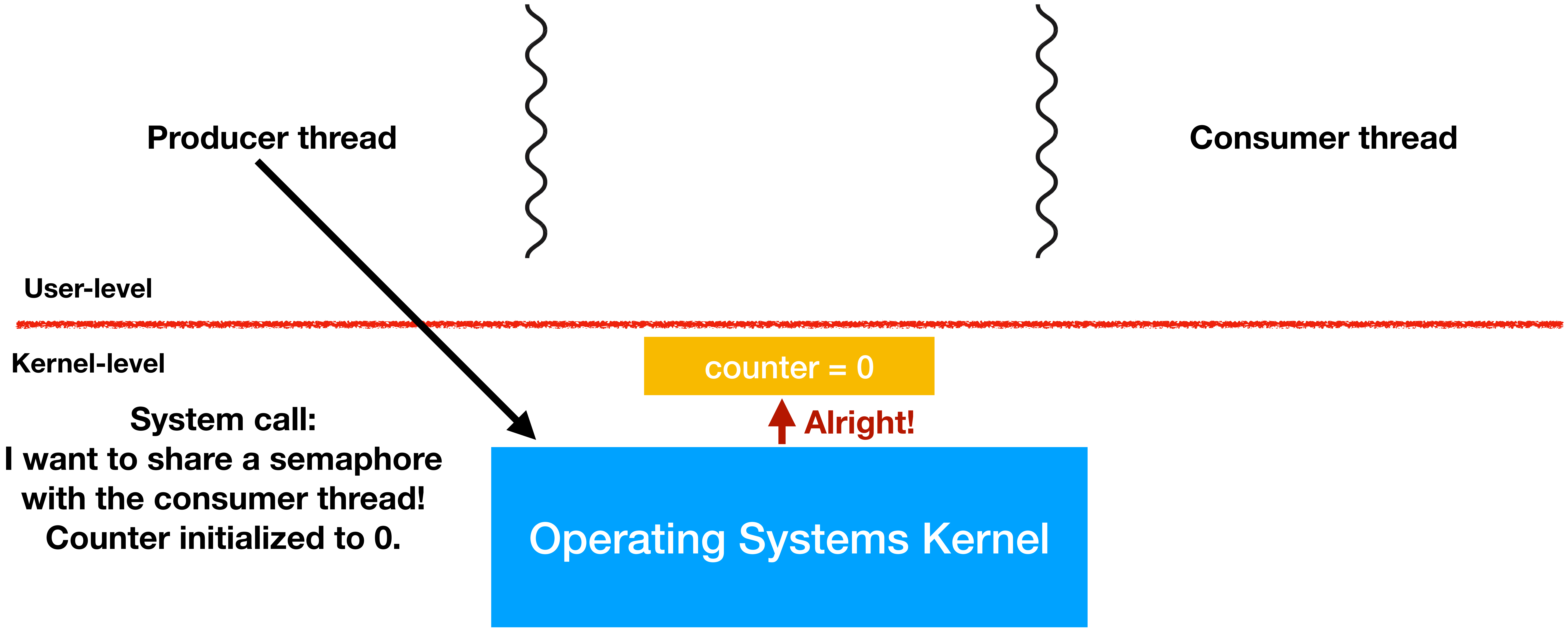
The producer-consumer problem

- There are two types of threads (or processes): producer and consumer.
  - Producer produces some kind of resources (e.g., HTTP web request) and consumer consume the resources (e.g., process the request).
- Goal: consumer should only be scheduled when some resource produced by the producer is available (i.e., has not been consumed).
- The core of semaphore is a counter of such available resources. If counter is greater than 0, a consumer thread will be scheduled.

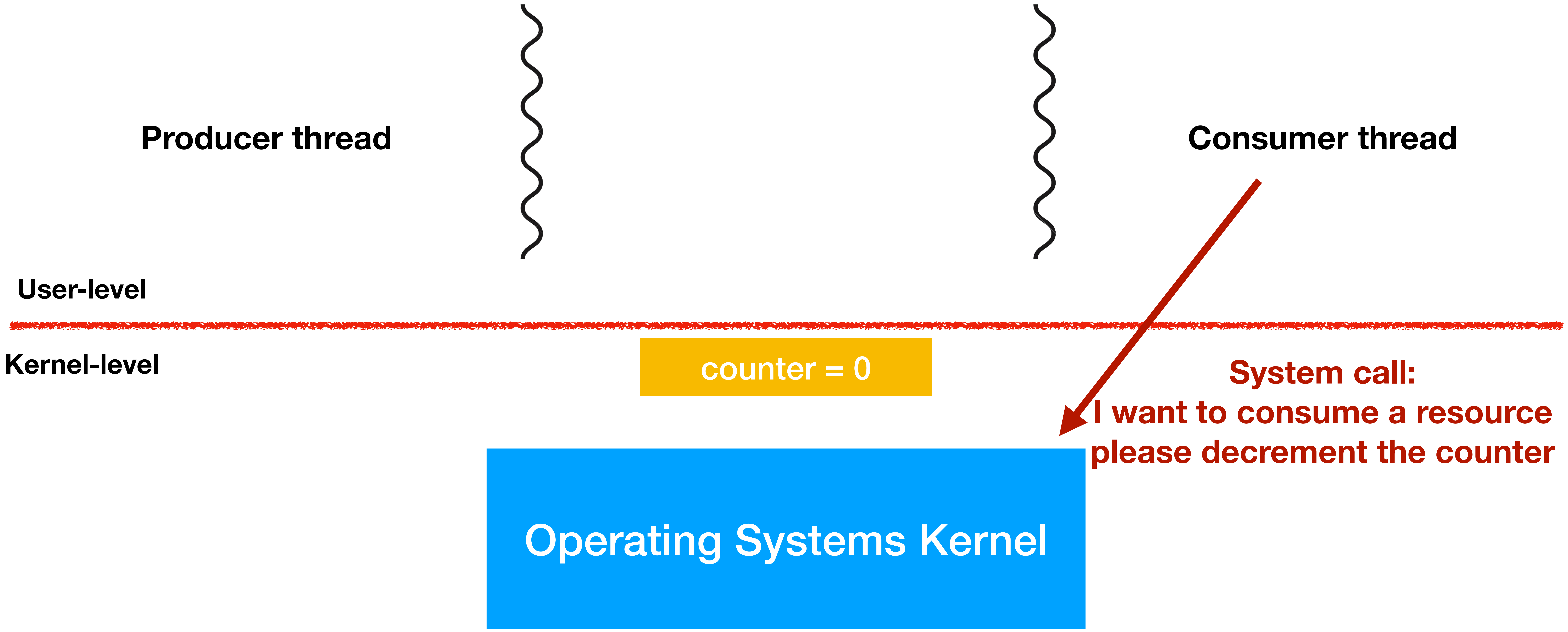
Producer-consumer example



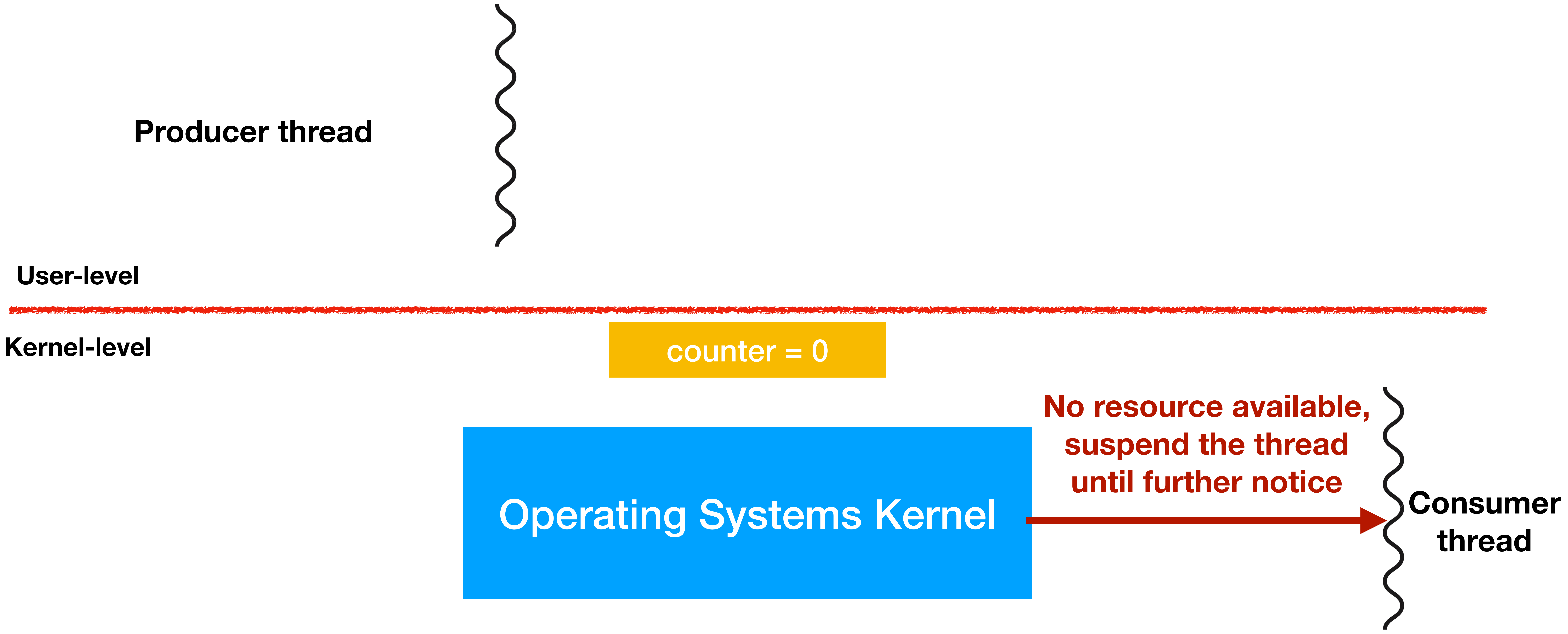
Producer-consumer example



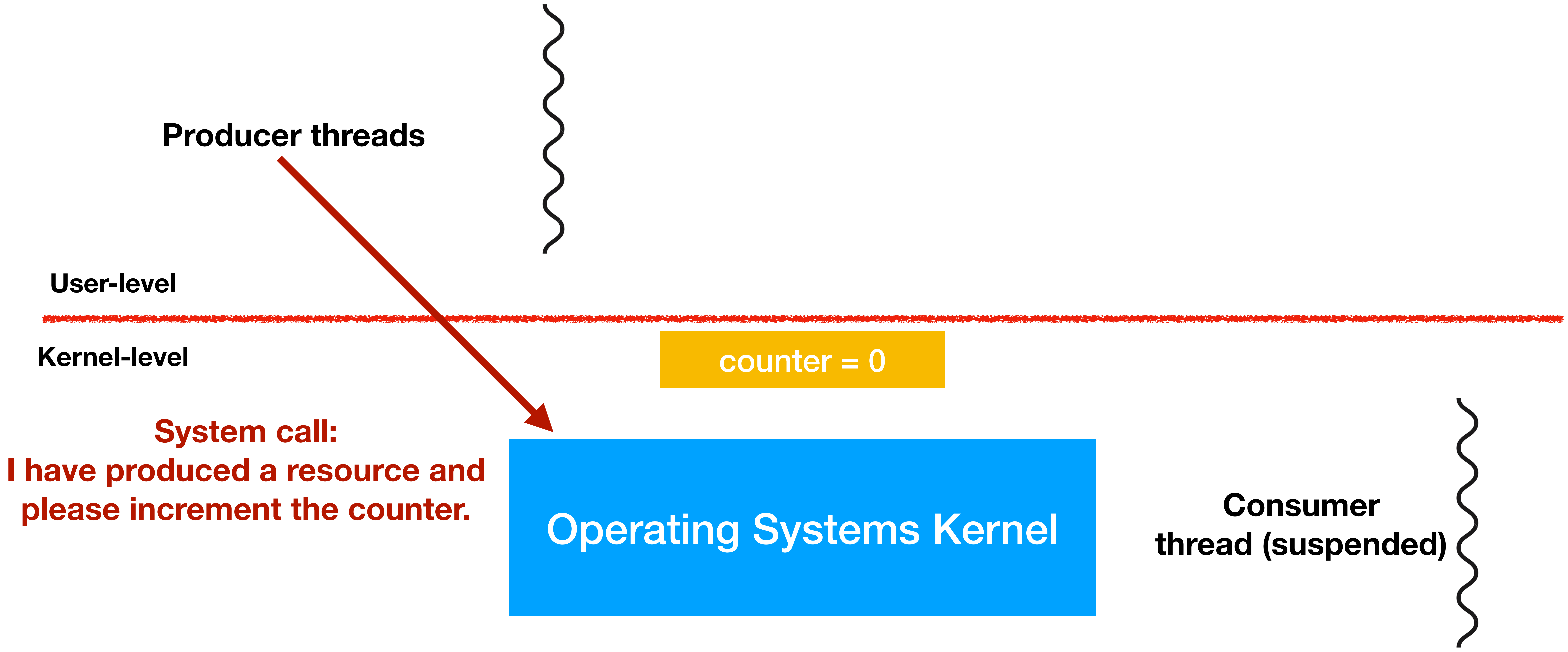
Producer-consumer example



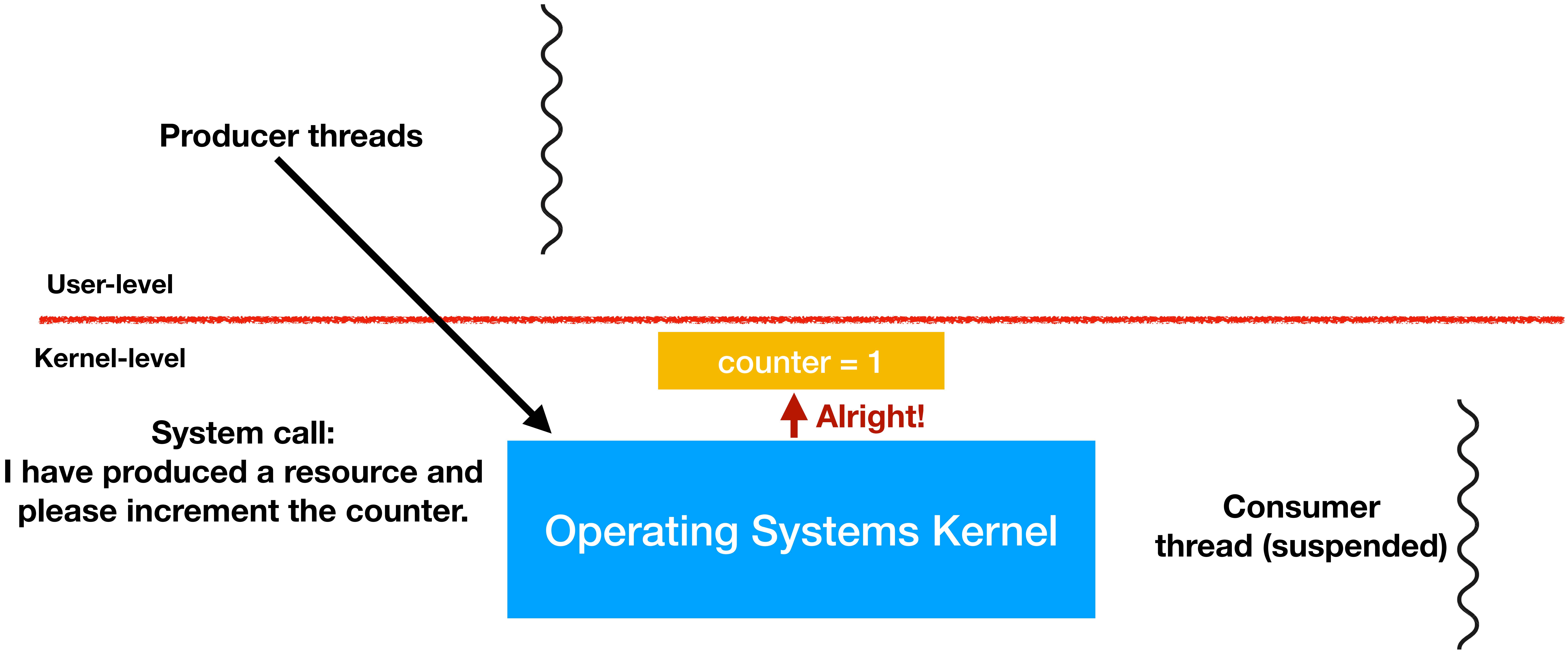
Producer-consumer example



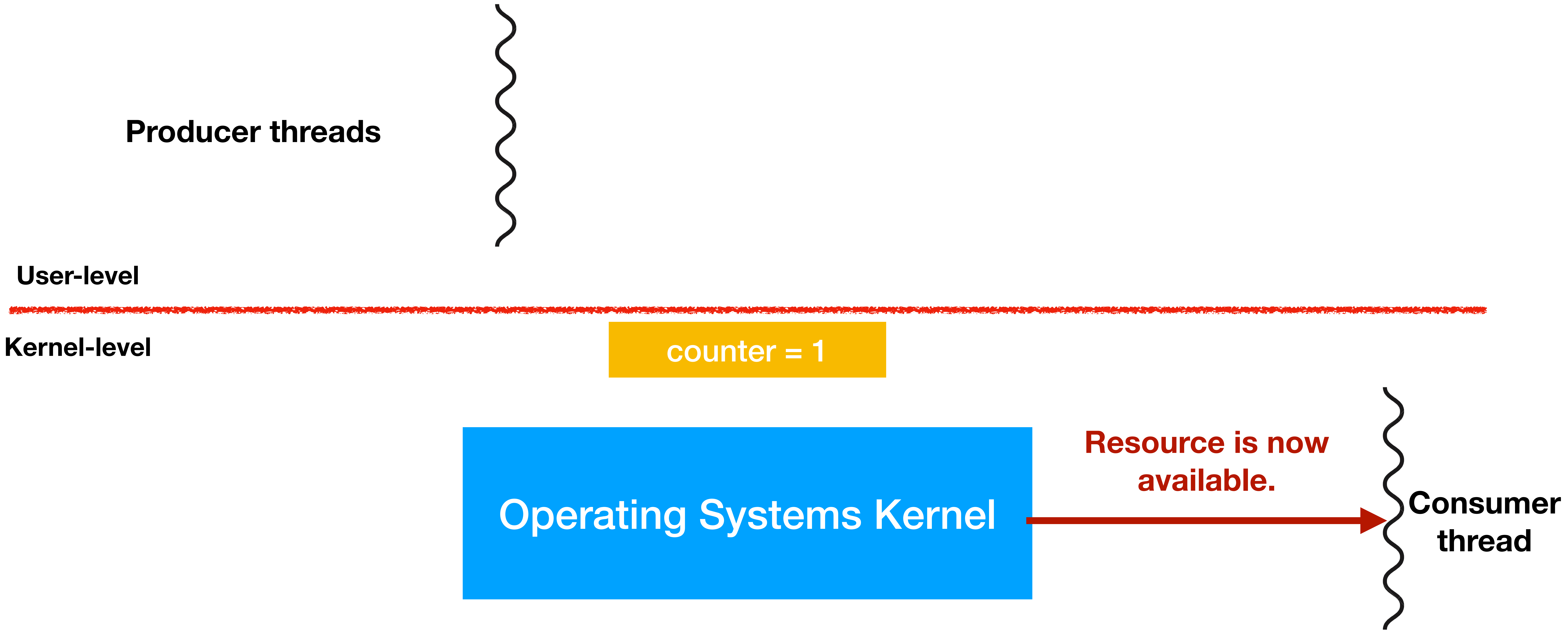
Producer-consumer example



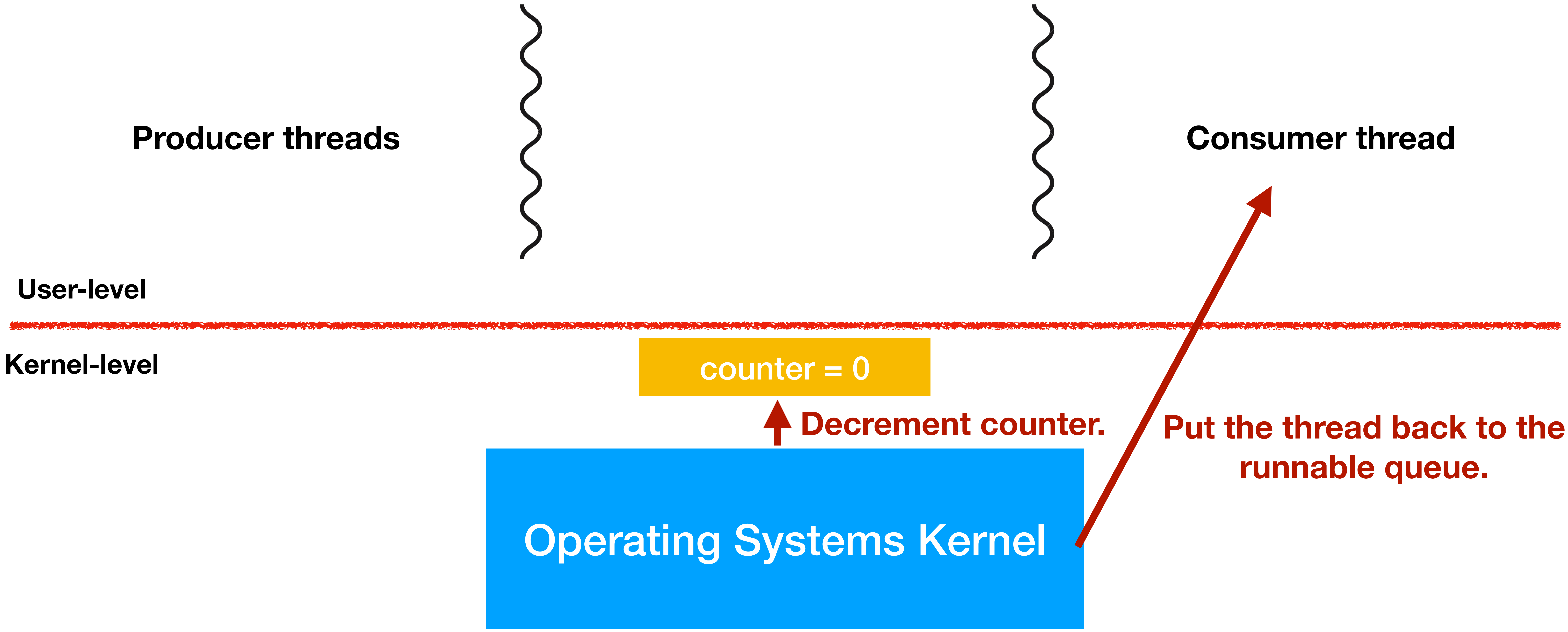
Producer-consumer example



Producer-consumer example



Producer-consumer example



Variants of producer-consumer

- There can be multiple producer threads and consumer threads.
- Bounded buffer: a producer can only produce if the number of available resources (the value of the counter) is not greater to a given number.
- …

Semaphores in P1

```
struct sema {
    // counter
    // queue of threads that are put to sleep
    // feel free to add other fields that you need
};

// initialize a semaphore
void sema_init(struct sema *sema, unsigned int count)

// produce a resource by incrementing the semaphore
void sema_inc(struct sema *sema)

// consume a resource by decrementing the semaphore
void sema_dec(struct sema *sema)
```

Lesson: semaphore is easy to implement, but it is not very useful and one should try to avoid using it.

Refer to “12 Commandments of Synchronization” by Emin Gün Sirer

Homework

- P1: implement semaphores and test your semaphore with producer-consumer and other synchronization problems.
- Read the Linux manual of System V IPC: https://man7.org/linux/man-pages/man7/svipc.7.html
- Next lecture: introduce the concepts of timer interrupt and scheduling for P2.

Concepts in the real-world (Unix/Linux)

- User-level threads
  - `getcontext, setcontext, …`
- Kernel-level threads
  - `pthread_create, pthread_join, …`
- Processes
  - `fork, …`

Concepts in the real-world (Unix/Linux)

- Message queues
  - `msgget, msgsnd, msgrcv, …`
- Semaphores
  - `semget, semop, …`
- Shared memory
  - `shmget, shmdt, …`