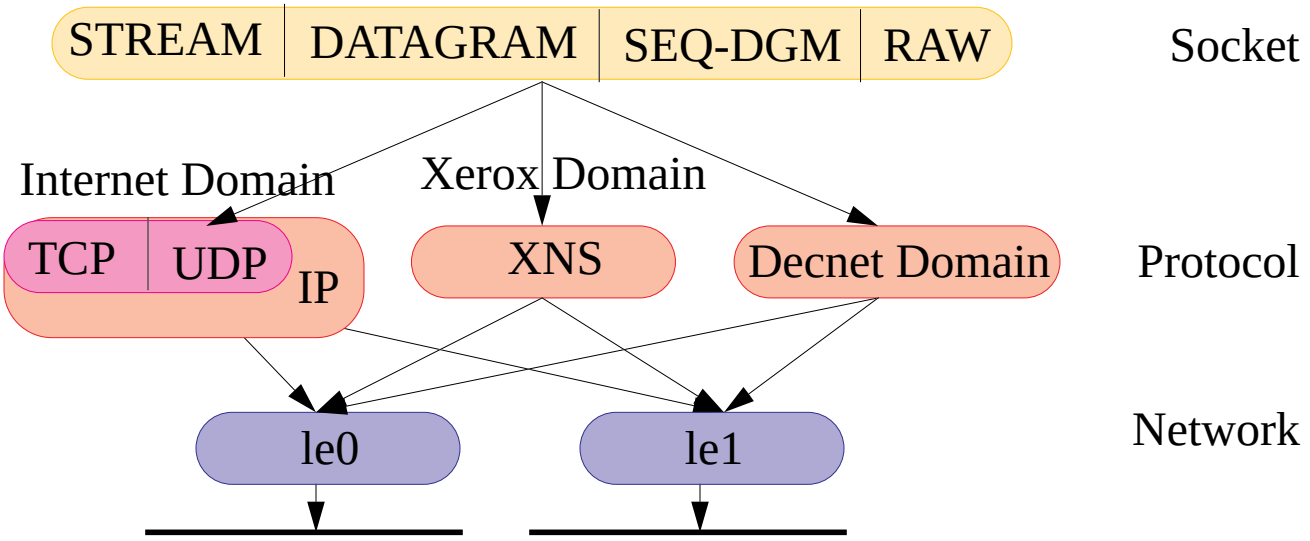
Socket Service Types

The following socket types are defined:

1. **SOCK_STREAM**: stream socket
2. **SOCK_DGRAM**: datagram socket
3. **SOCK_RAW**: raw-protocol interface
4. **SOCK_RDM**: reliably-delivered message
5. **SOCK_SEQPACKET**: sequenced packet stream

More types may be defined in the future

IPC Implementation in UNIX



Socket Layer (specifies required service)

- Abstract objects that provide distinct endpoints of communication
- Buffering

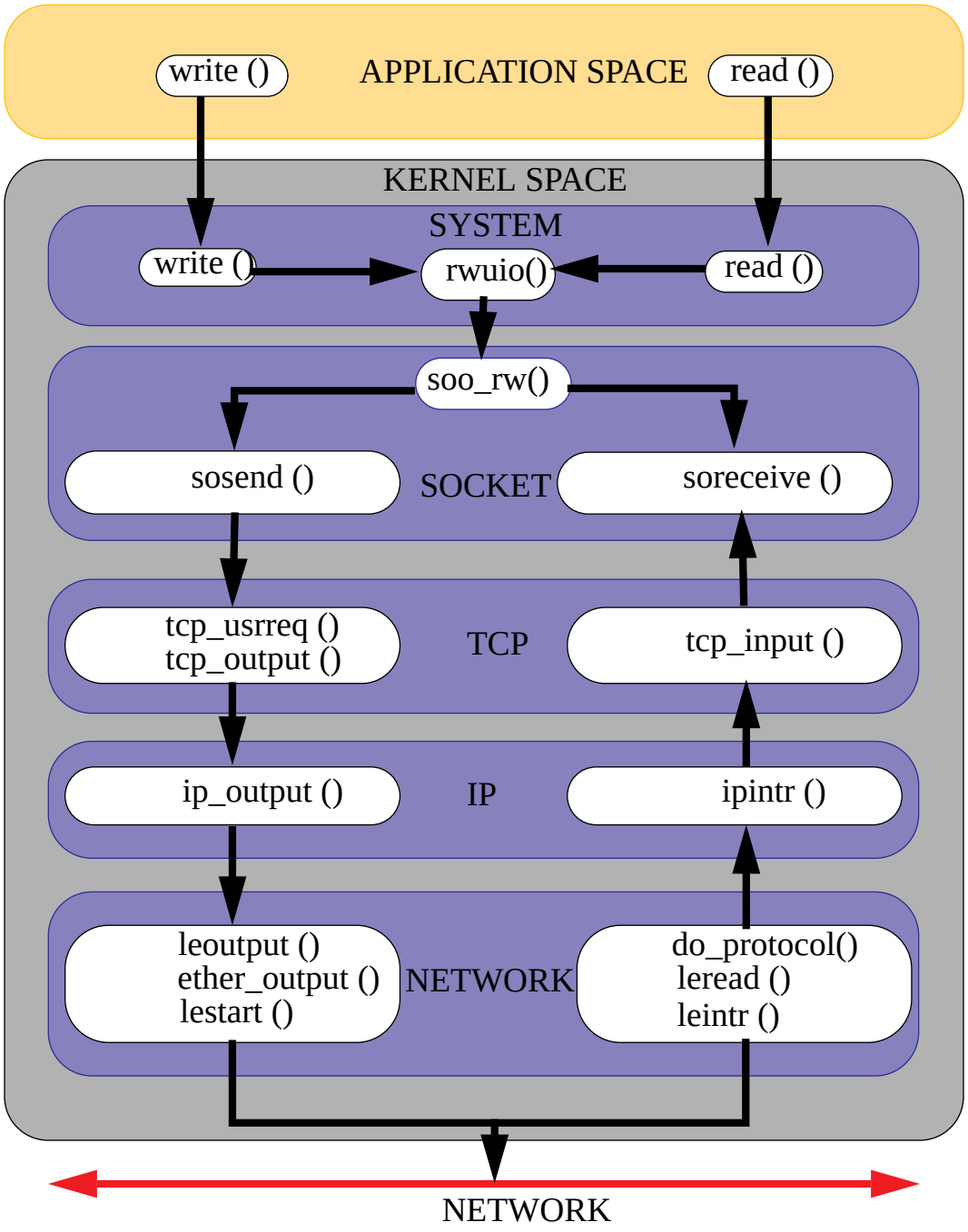
Protocol Layer (implements the service)

- Domains and their protocols

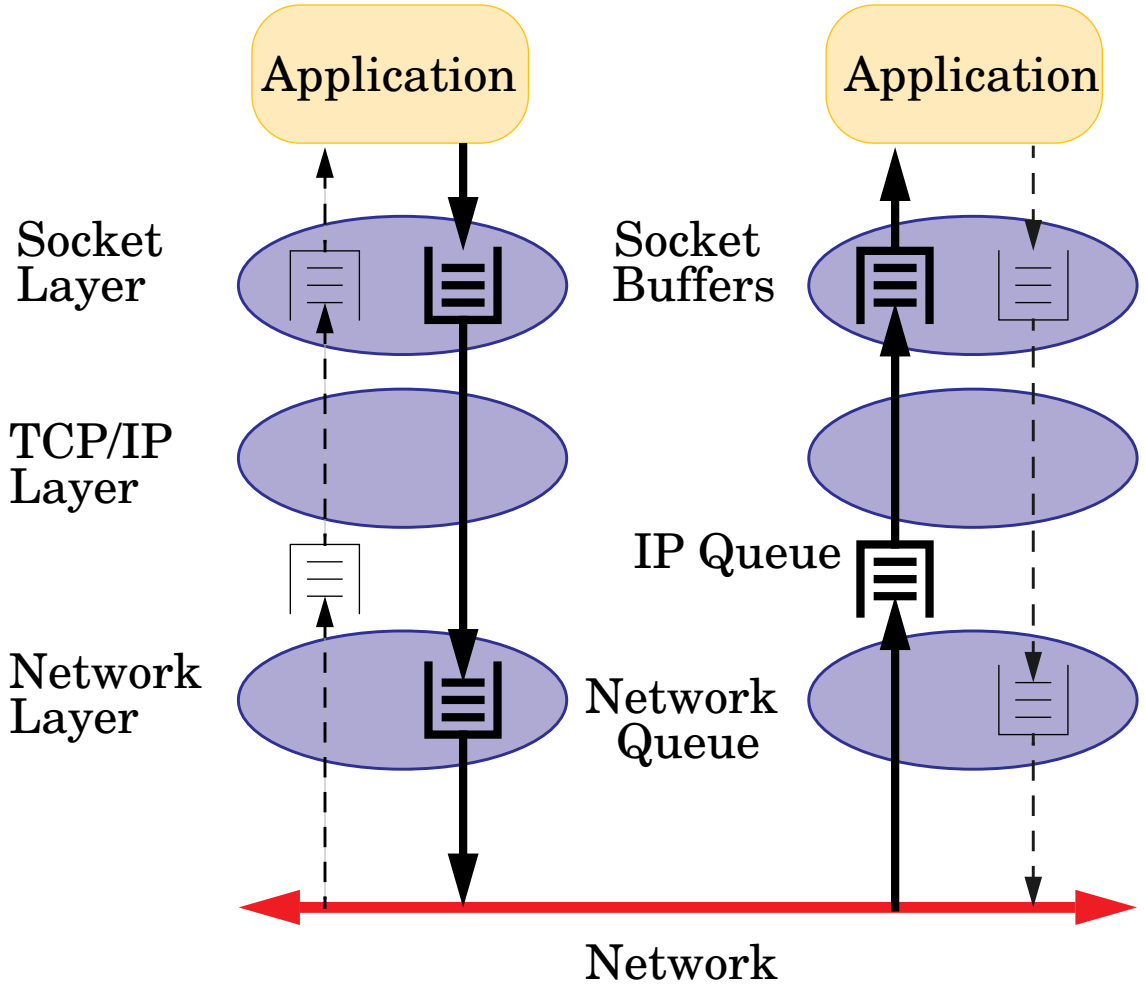
Network Layer:

- Interface to the network hardware

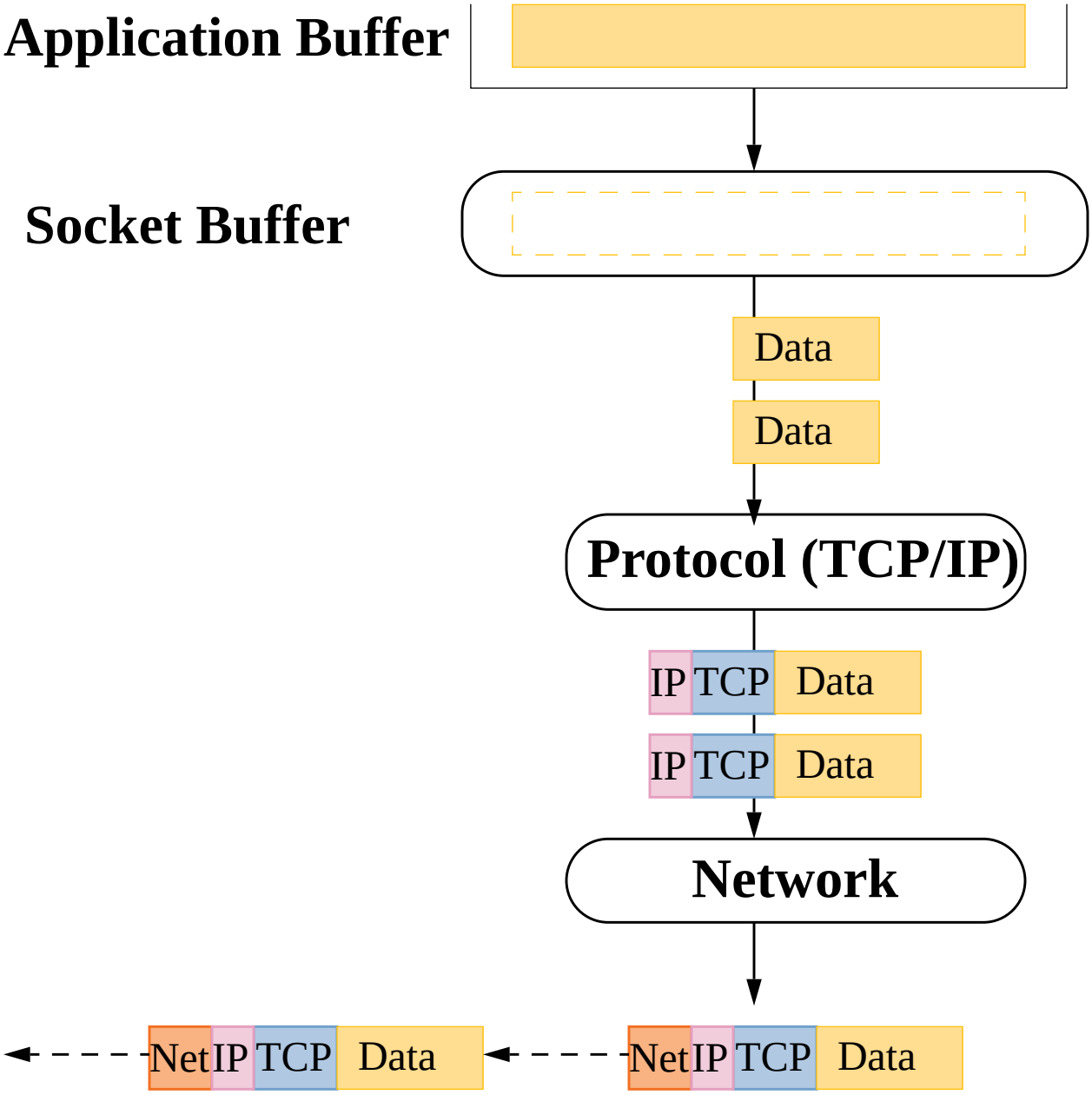
FUNCTION CALLS FOR TCP/IP



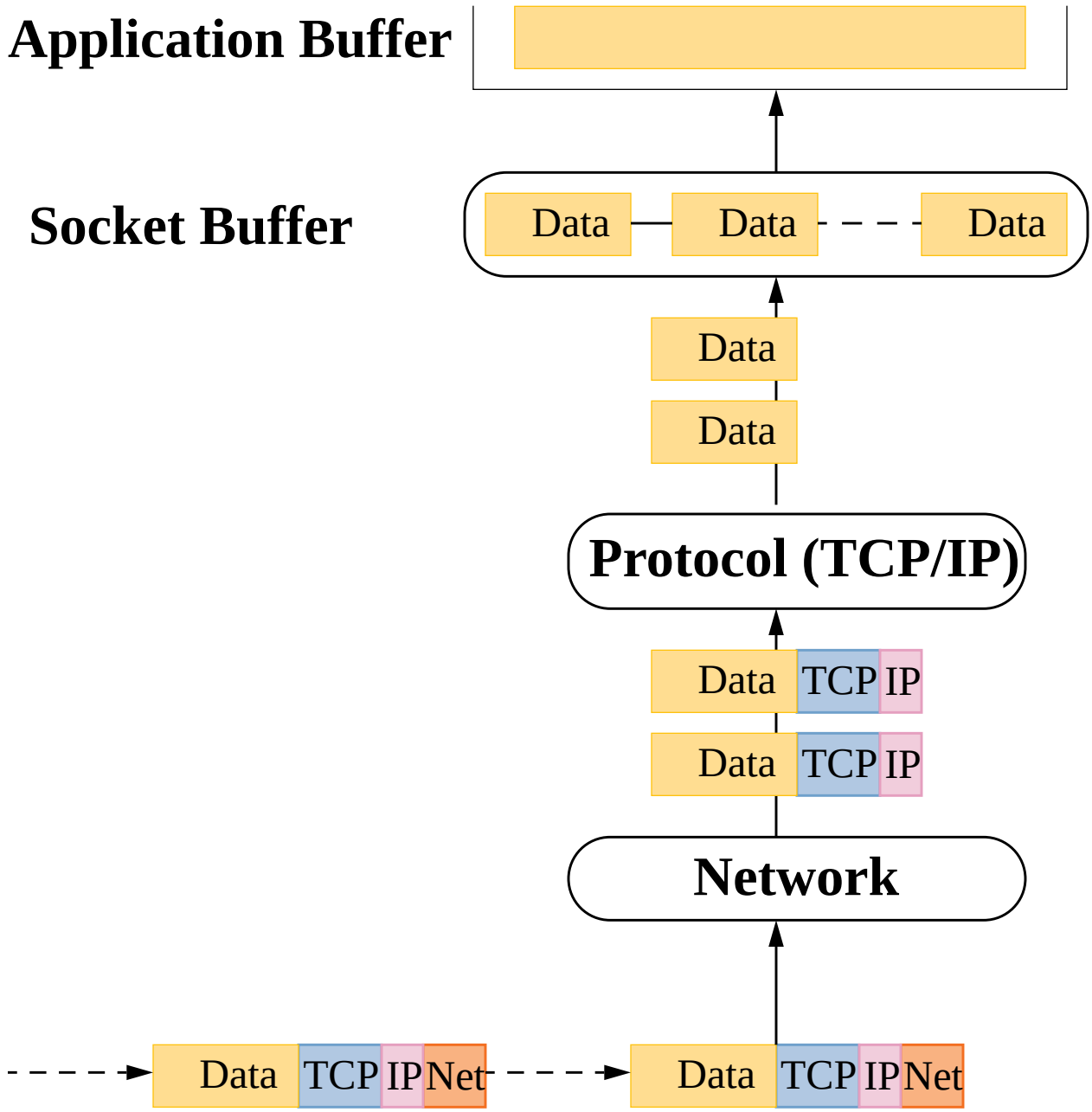
IPC Packet/Data Queues



Data Transmission



Data Reception

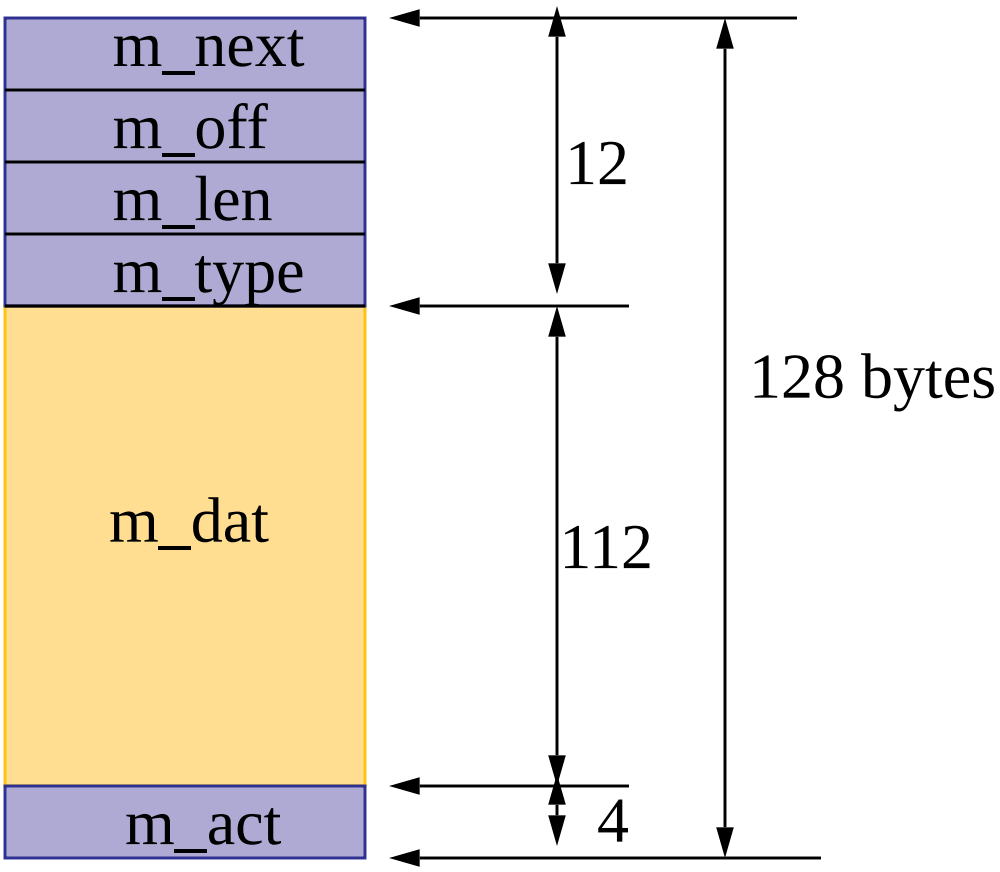


Memory Management Requirements

- **Allocate**/**deallocate** memory fast and efficiently
- Handle both **small** and **large** packets efficiently
- **Copy** packets efficiently
- **Trim** data from **front** (headers) or **back** (trailers) of packets efficiently (without copying)
- Preserve **packet sequence** (streams, sequenced datagrams)
- Preserve **packet boundaries** (datagrams)

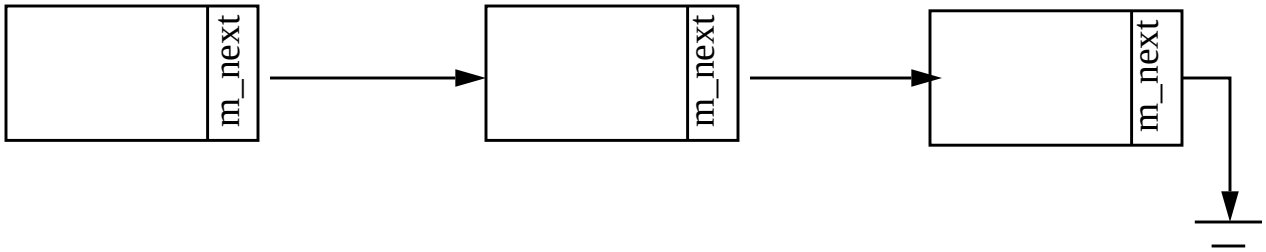
Solution: Mbufs

What is an mbuf (memory buffer)?

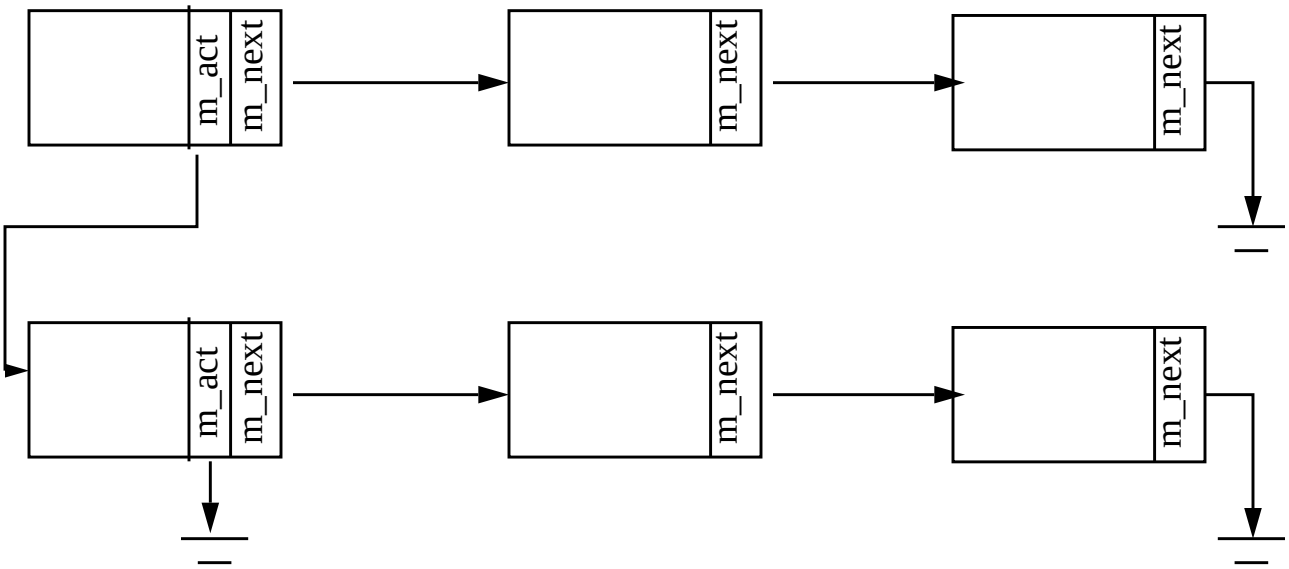


Mbuf chains and lists

Mbufs can be linked to form **Chains** using the **m_next** field:

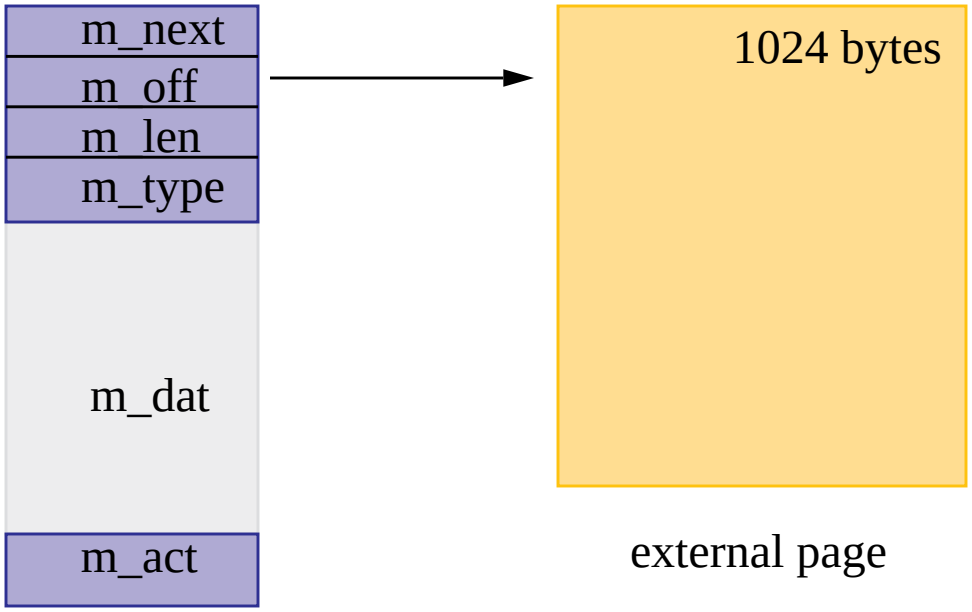


or linked to form **Lists** using the **m_act** field:



Storing Large packets

Instead of storing data in the mbuf itself, data can be stored in an external page:



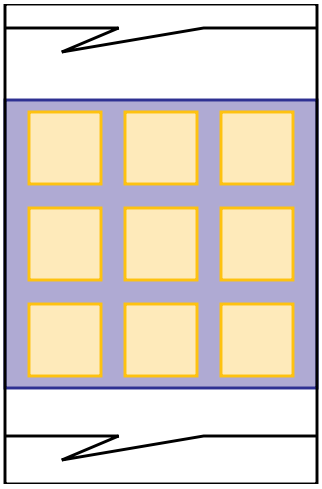
The mbufs are called **Cluster Mbufs**

In a cluster mbuf the m_dat field is **always** empty

Mbuf Allocation/Deallocation

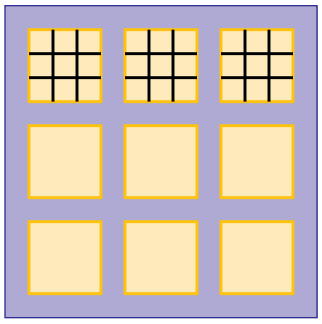
During system initialization a pool of memory pages is allocated to the mbuf store:

kernel memory



Mbuf Store:
Never paged out!

Some of these pages are used for mbufs. The rest are used as external store for cluster mbufs:



Pages containing mbufs

Pages for
cluster mbufs

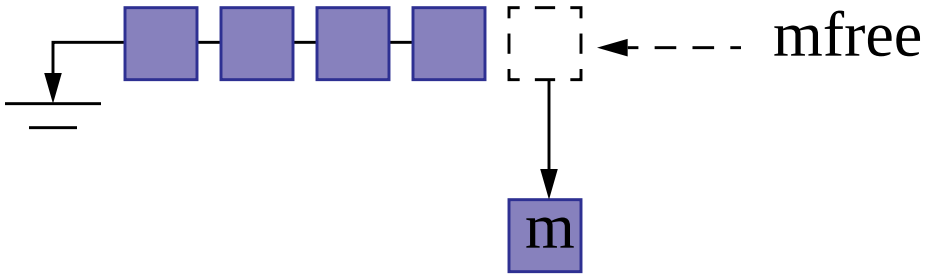
All free mbufs are on the mbuf free list

All free pages are on the page free list

Mbuf allocation

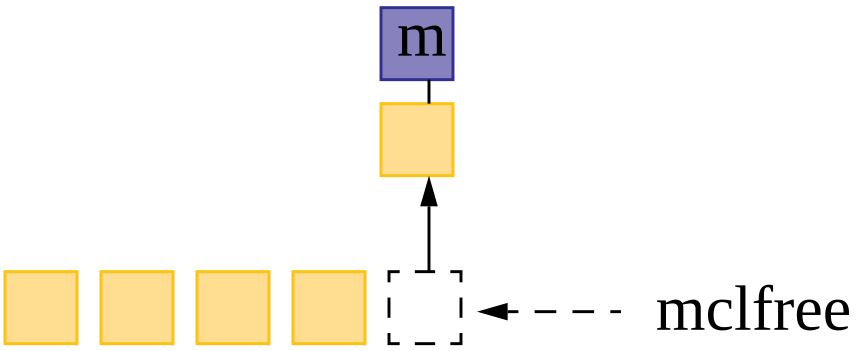
struct mbuf *m = m_get ()

Returns a pointer to an mbuf from the free list:



int mclget (m):

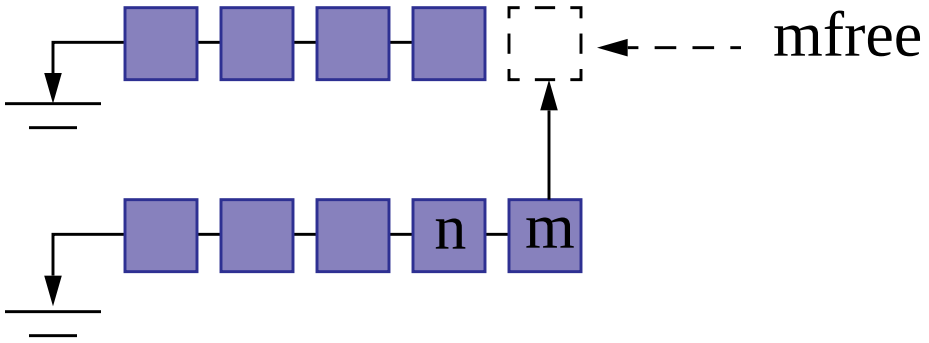
Given an mbuf attach a page to create a cluster mbuf:



Mbuf Deallocation

struct mbuf *m_free (m):

Given an mbuf chain, free the first mbuf and return a pointer to the next mbuf:



m_free () works on cluster mbufs too

int m_freem (m):

Free a chain of mbufs

Mbuf Copy

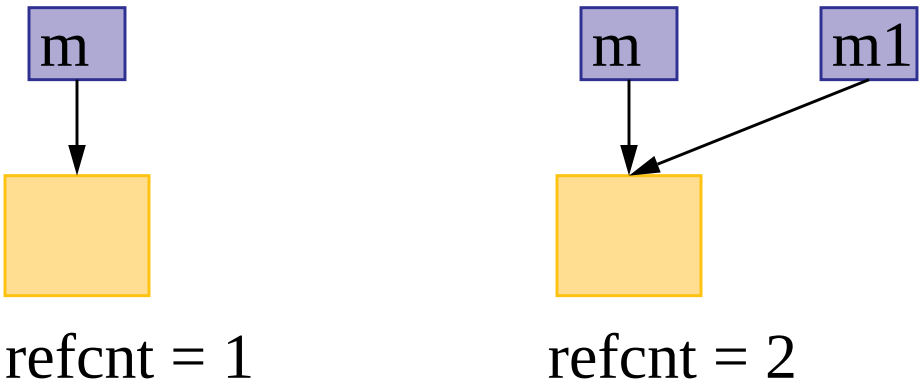
struct mbuf *m1 = m_copy (m, off, len):

Copies *len* data bytes from *m* starting at offset *off*
Returns a new mbuf chain m1

Copying a **cluster** mbuf is accomplished by:

- allocating a new mbuf
- making m_off to point to the external page
- incrementing the reference count of the external page

m1 = m_copy (m) results to:

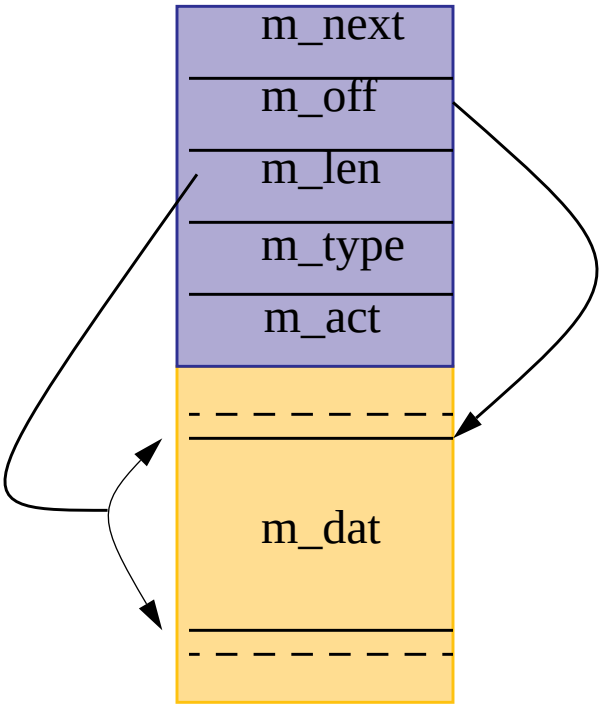


Trimming an Mbuf

int m_adj (m, len):

if len is positive, trim len data from head

if len is negative, trim len data from tail



Mbuf to Data and Data to Mbuf

data = mtod (m, type):

Return the address of the data in the mbuf casting it to *type*

e.g.:

```
int *p;
p = mtod (m, int *)
```

struct mbuf *m = dtom (ptr):

The inverse of mtod ()

Return the address of the mbuf where *ptr* resides

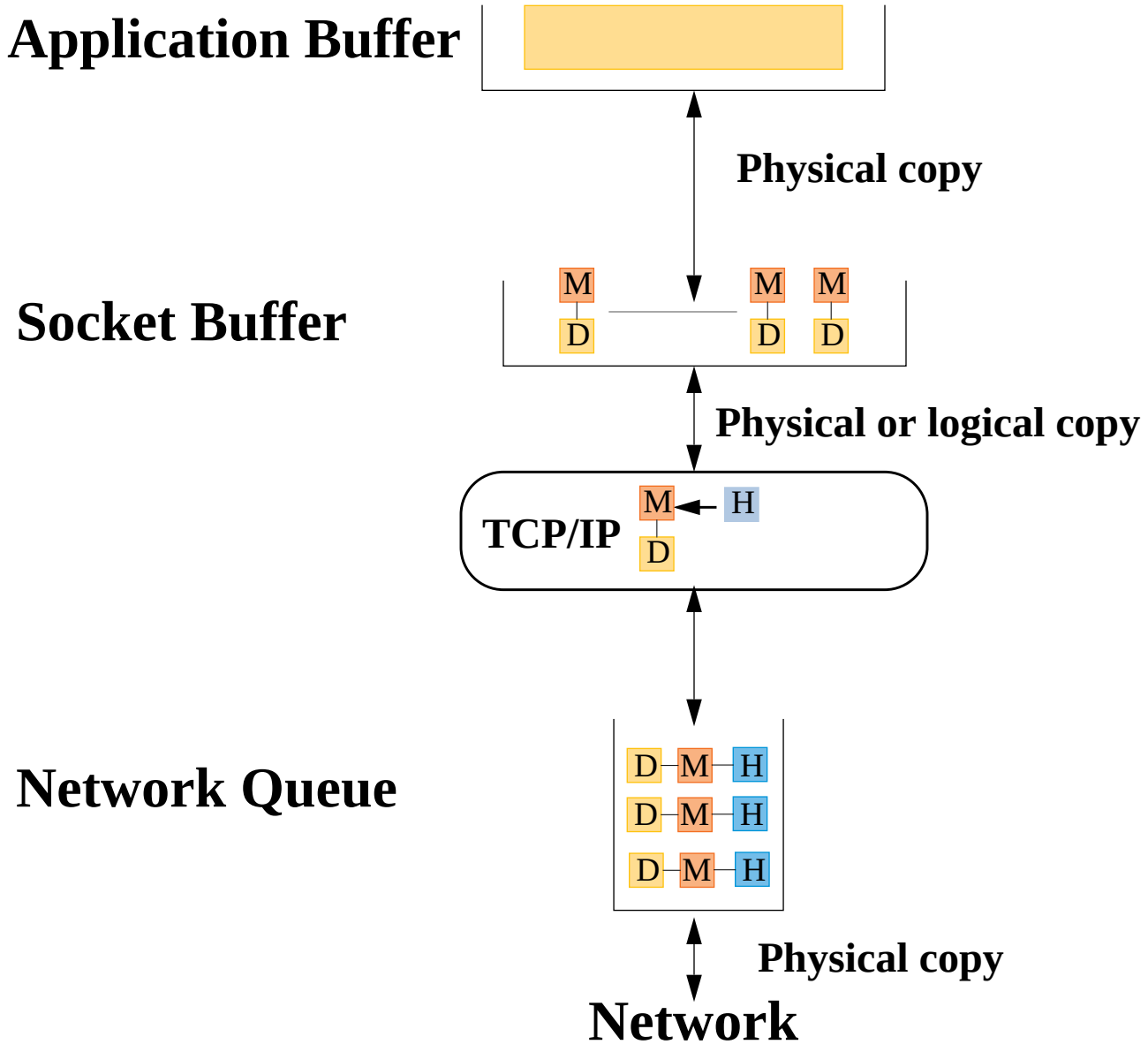
Making mbuf data contiguous

struct mbuf *m1 = m_pullup (m, len):

Rearrange mbuf chain *m* so that *len* bytes are contiguous in the data area. Return resulting mbuf chain. Copies data if necessary.

Needed so that *mtod ()* and *dtom ()* will work (e.g., when accessing protocol headers)

Data Movement with Mbufs



M — Cluster mbuf containing data

D — Page containing data

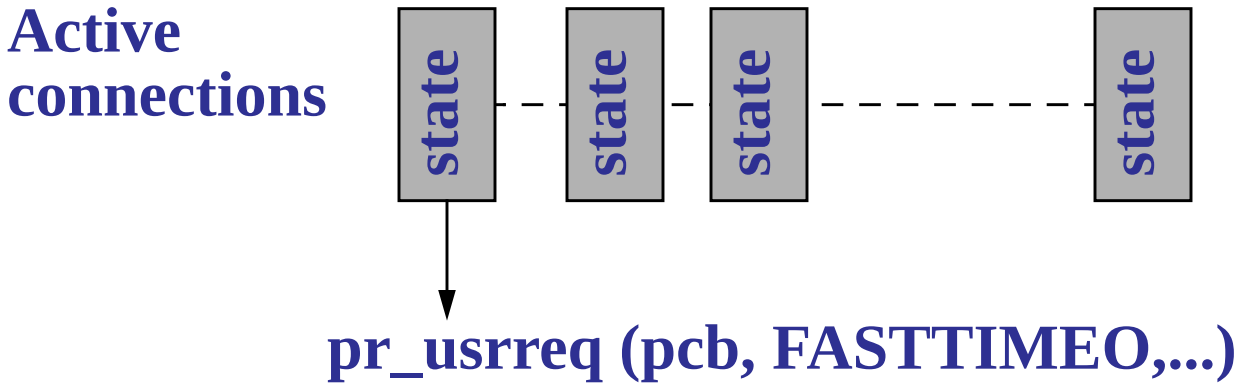
H — Mbuf containing header

Timers

Two types of timers are provided by the OS:

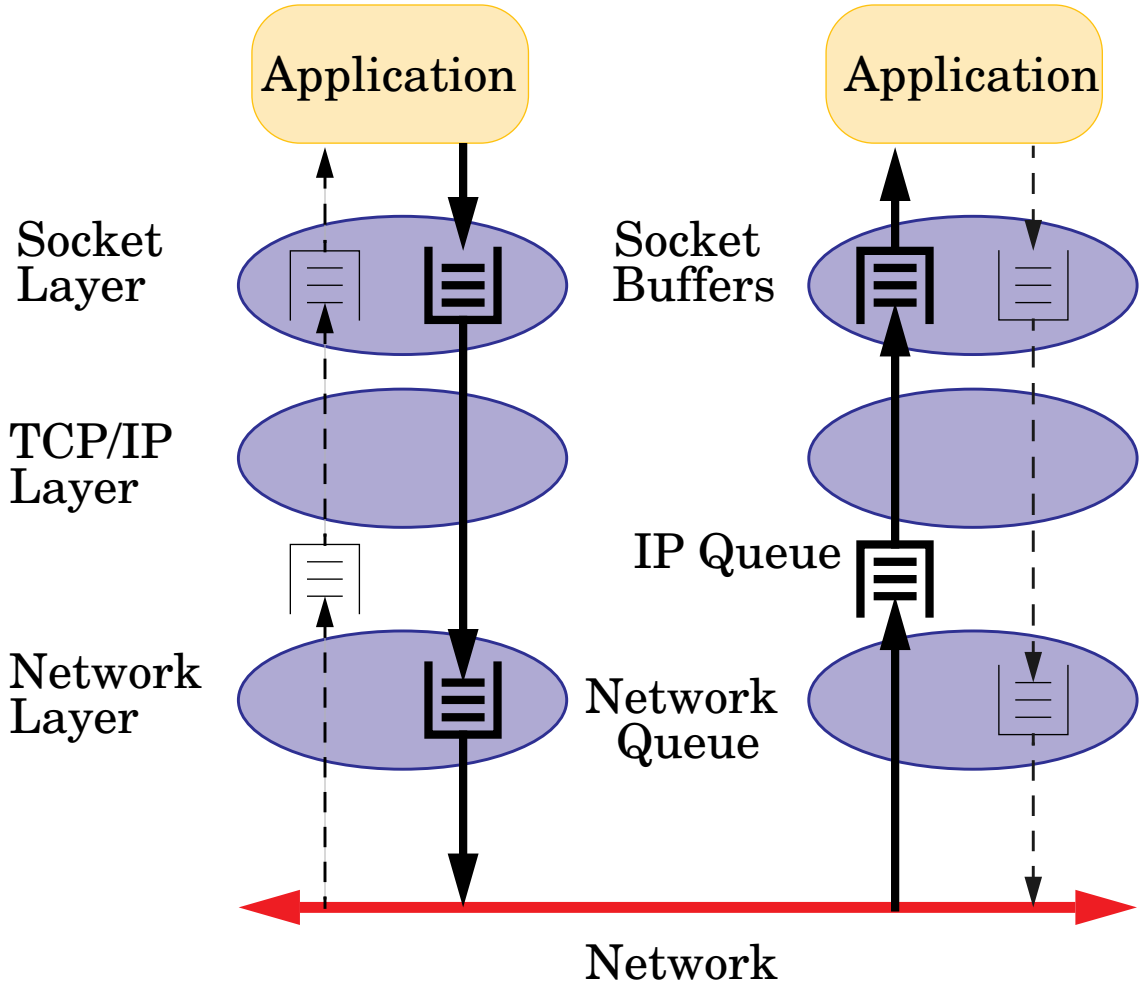
- Fast timer: every 200 ms
- Slow timer: every 500 ms

Every time a timer fires, the list of active connections is traversed and the protocol user request function is called for each connection:



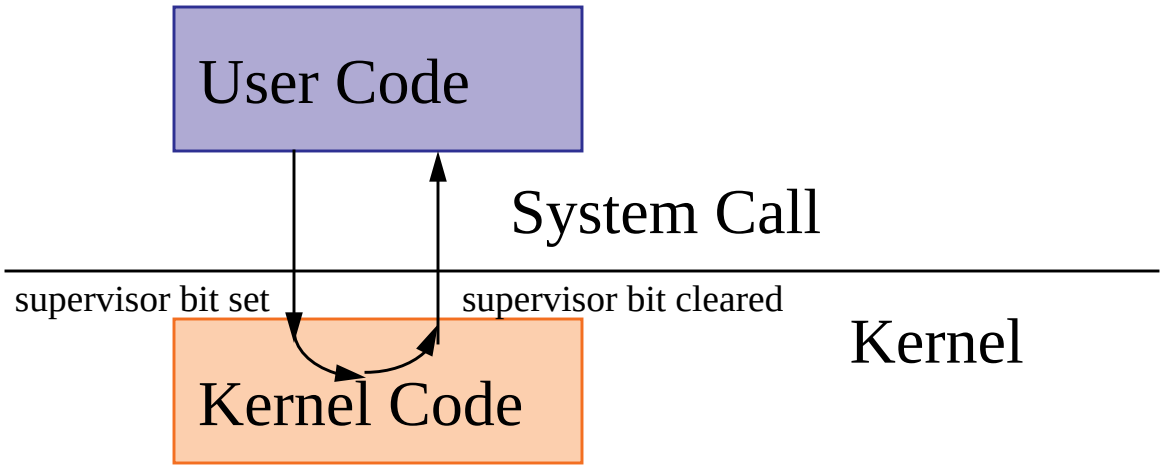
Protocol processing (like retransmissions or delayed ack) can now take place

IPC Packet/Data Queues



Kernel Scheduling

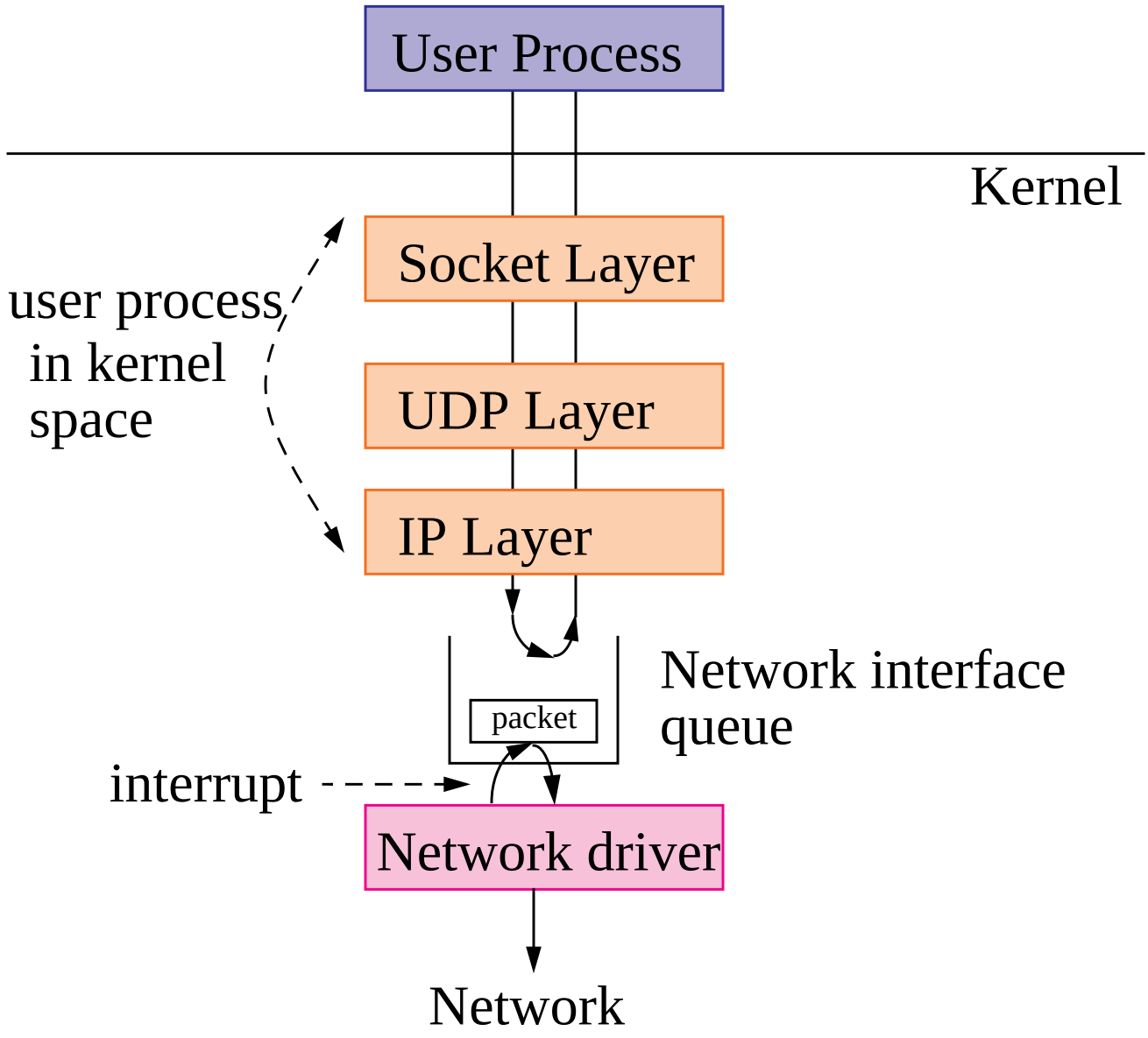
What is a system call?



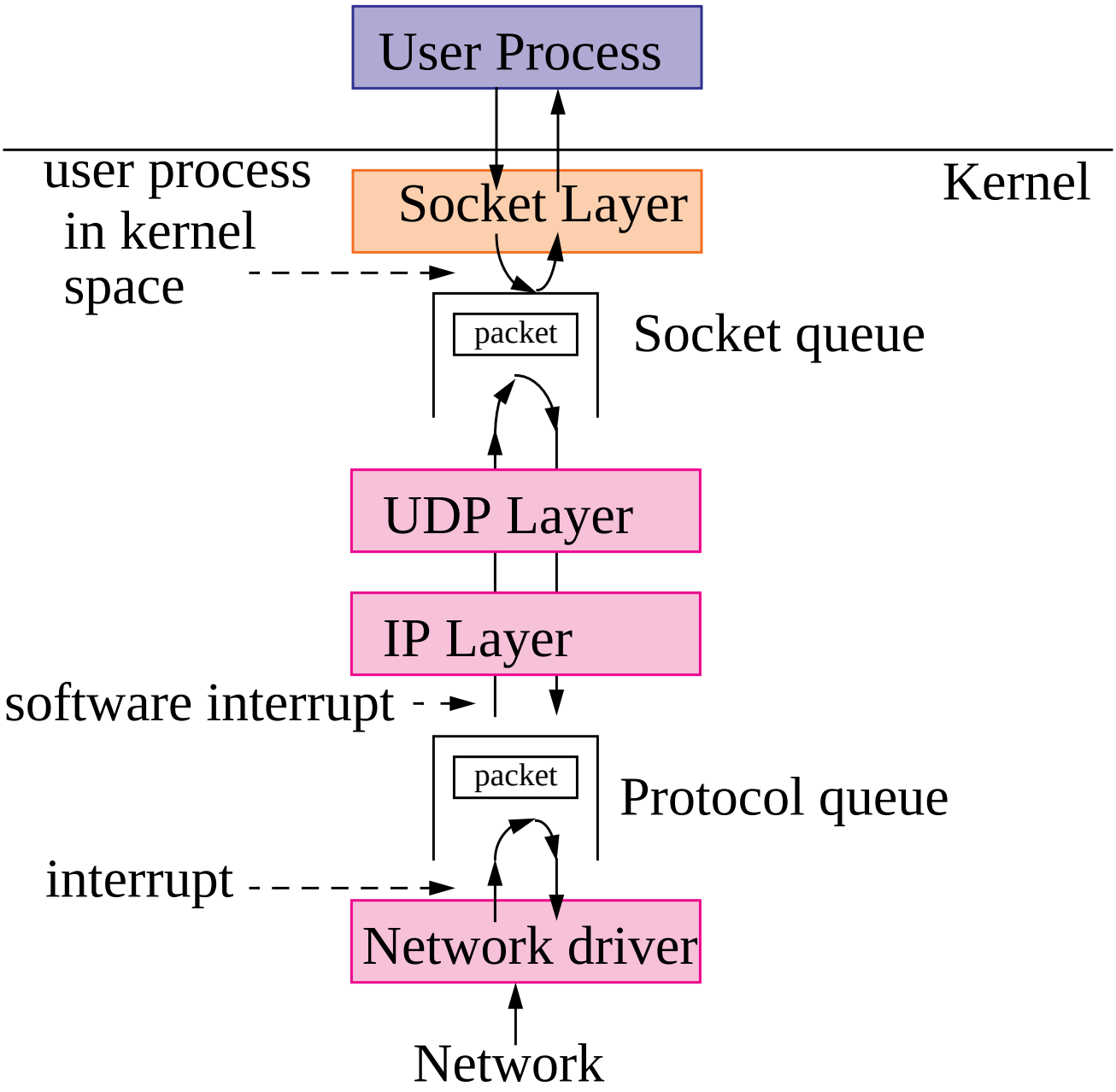
The kernel address space is always mapped into each process' address space. During a system call, a user process enters the kernel (supervisor bit is set) and executes kernel code. Once processing is done, the supervisor bit is cleared and the process returns to user space.

Any normal kernel processing (e.g., scheduling, interrupt processing) is therefore not affected (unless the system call masks interrupts)

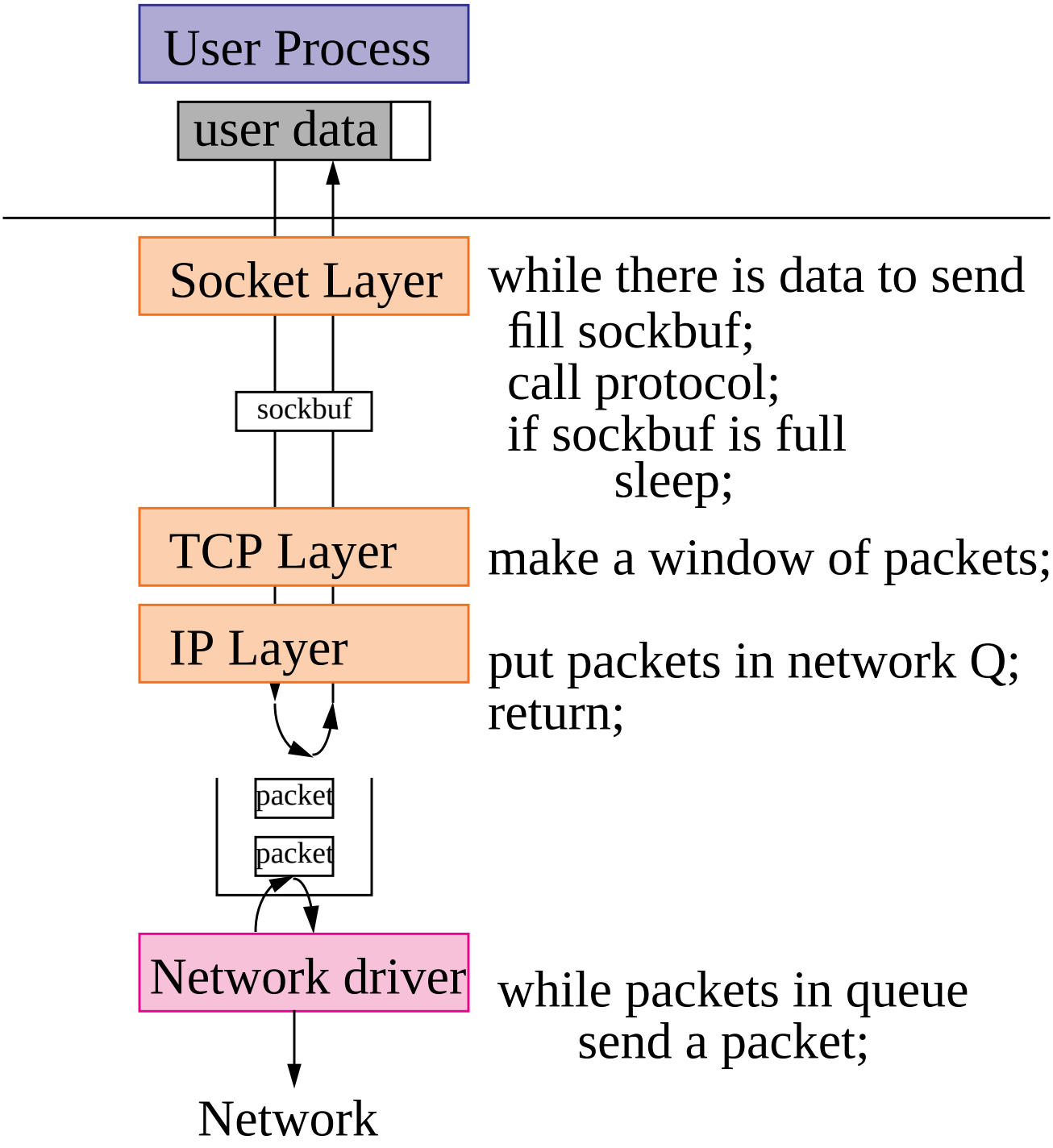
Sending Data: Datagrams



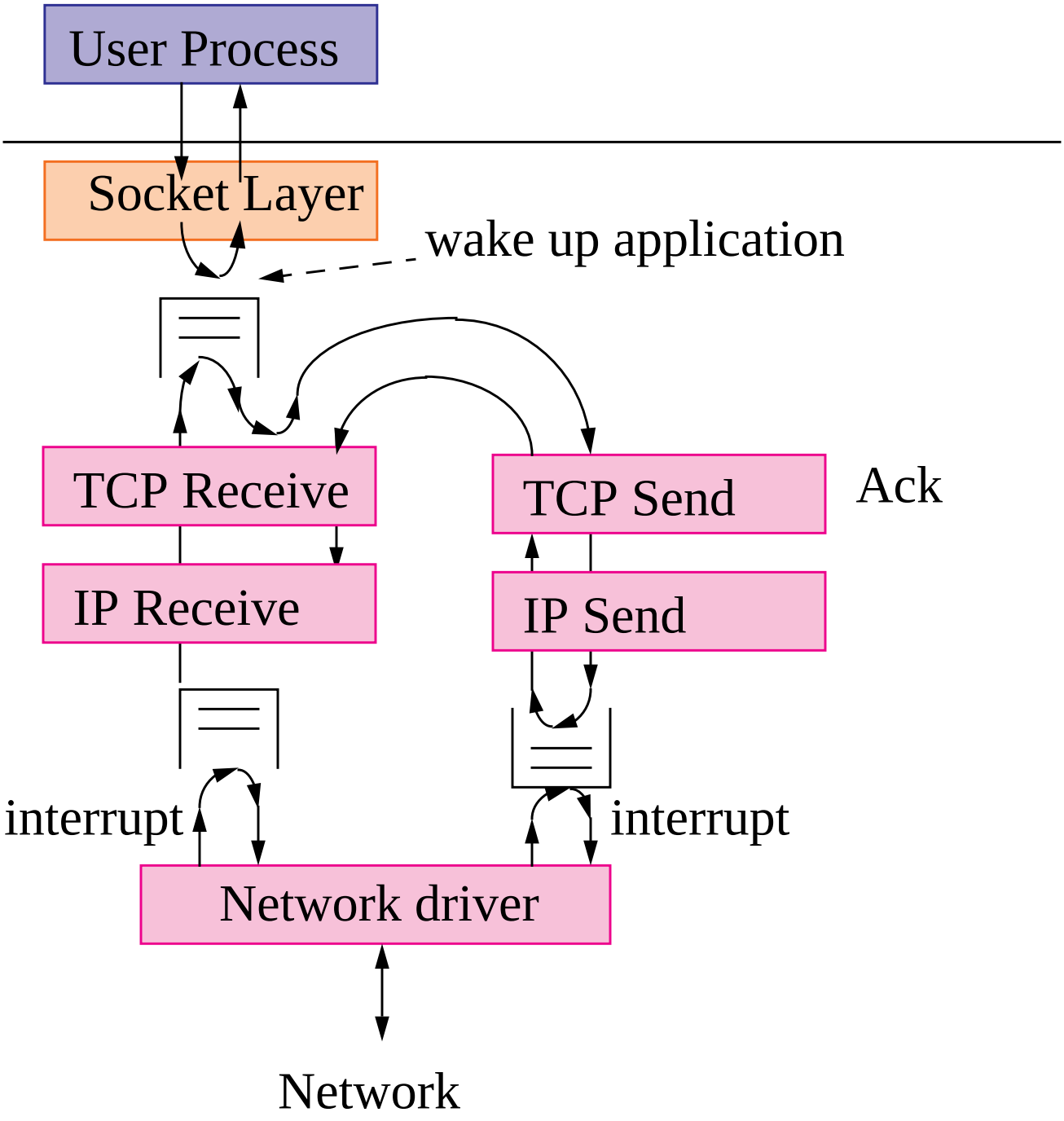
Receiving Datagrams



Sending Data: Streams

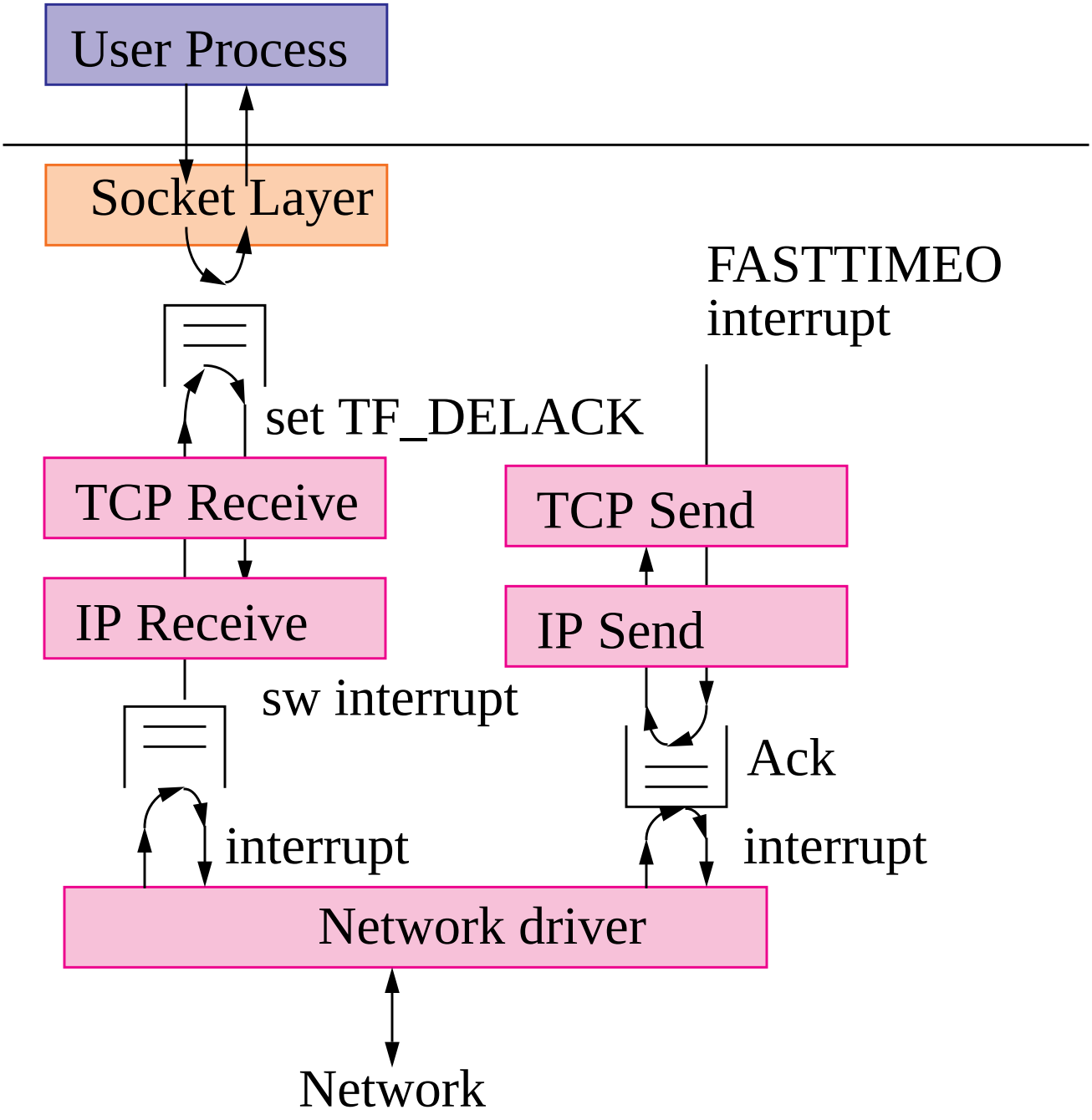


Receiving Stream Data



Delaying Acknowledgment

To avoid “silly window”, ack may be delayed:



Receiving Acks