The TCP/IP stack in the FreeBSD kernel: an overview of the implementation

Kevin Lo
msi
The FreeBSD project

Examples of operating systems use FreeBSD-based network stack

DragonflyBSD, a fork of FreeBSD

OS X from Apple

Os^{v} from Cloudius Systems

RTEMS

Peeking at the Linux kernel changelogs

Increase the initial cwnd to 10 (RFC 6928) : 2.6.39

Proportional Rate Reduction for TCP (RFC 6937) : 3.2

Early Retransmit for TCP (RFC 5827) : 3.5

TCP Fast Open : 3.6, 3.7

Memory buffers (mbufs)

struct mbuf: the most important data structure in the FreeBSD networking subsystem, which is defined in <sys/mbuf.h>

Every packet sent/received is handled using the mbuf structure

Mbufs are fixed size data buffers (256 bytes each)

mbuf structure

```
struct mbuf {
        struct m_hdr    m_hdr;
        union {
                struct {
                        struct pkthdr   MH_pkthdr;      /* M_PKTHDR set */
                        union {
                                struct m_ext    MH_ext; /* M_EXT set */
                                char            MH_databuf[MHLEN];
                        } MH_dat;
                } MH;
                char M_databuf[MLEN];           /* !M_PKTHDR, !M_EXT */
        } M_dat;
};
```

m_hdr structure

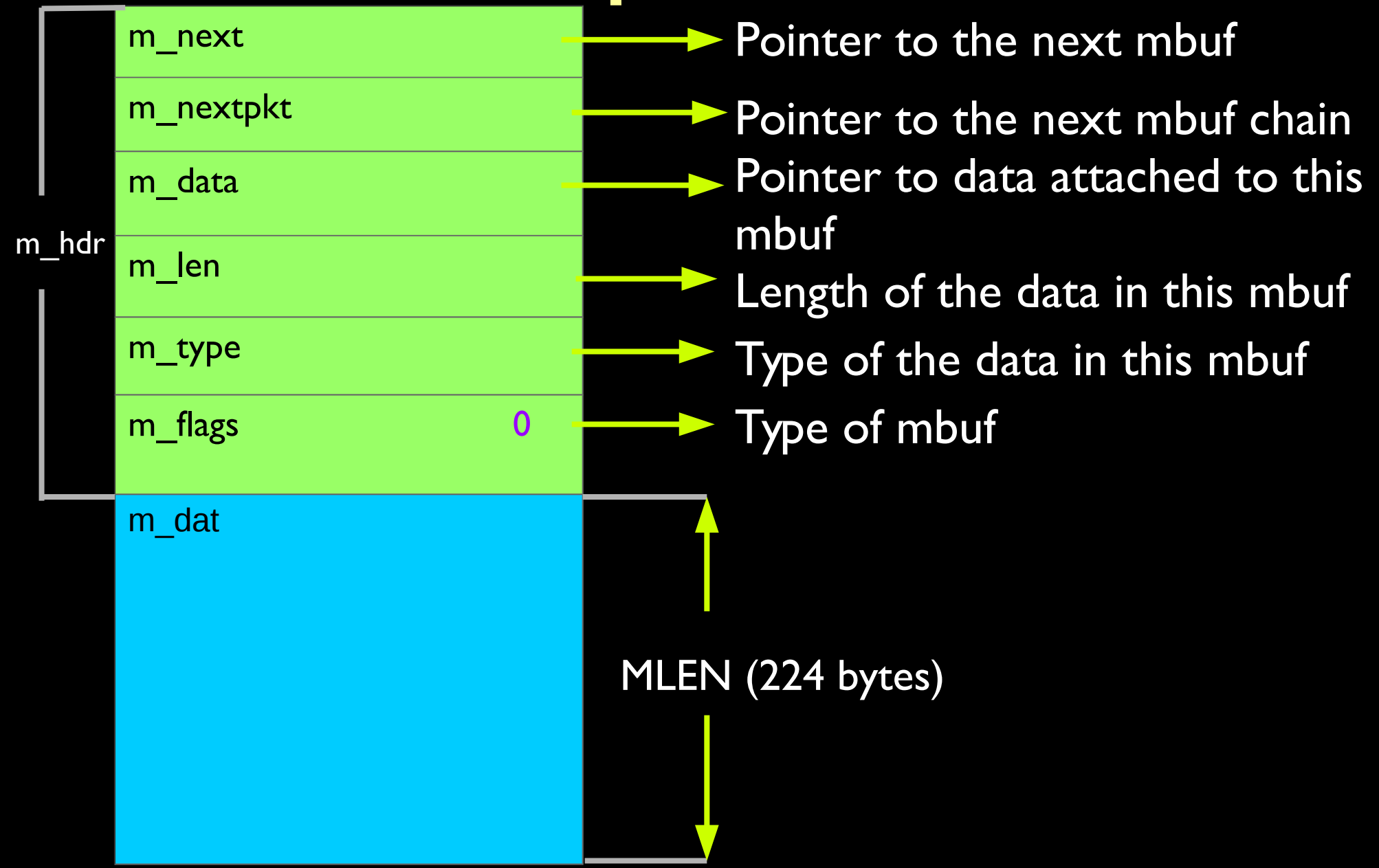
```
/*
 * Header present at the beginning of every mbuf.
 * Size ILP32: 24
 *         LP64: 32
 */
struct m_hdr {
        struct mbuf     *mh_next;       /* next buffer in chain */
        struct mbuf     *mh_nextpkt;  /* next chain in queue/record */
        caddr_t          mh_data;       /* location of data */
        int32_t          mh_len;        /* amount of data in this mbuf */
        uint32_t         mh_type:8,     /* type of data in this mbuf */
                         mh_flags:24;   /* flags; see below */
#if !defined(__LP64__)
        uint32_t        mh_pad;         /* pad for 64bit alignment */
#endif
};
```

On 64-bit platforms, this results into 3 * 8 bytes + 2 * 4 bytes = 32 bytes

Simple mbuf

```
struct mbuf {
        struct m_hdr    m_hdr;
        union {
                struct {
                        struct pkthdr   MH_pkthdr;      /* M_PKTHDR set */
                        union {
                                struct m_ext    MH_ext; /* M_EXT set */
                                char            MH_databuf[MHLEN];
                        } MH_dat;
                } MH;
                char M_databuf[MLEN];           /* !M_PKTHDR, !M_EXT */
        } M_dat;
};
```

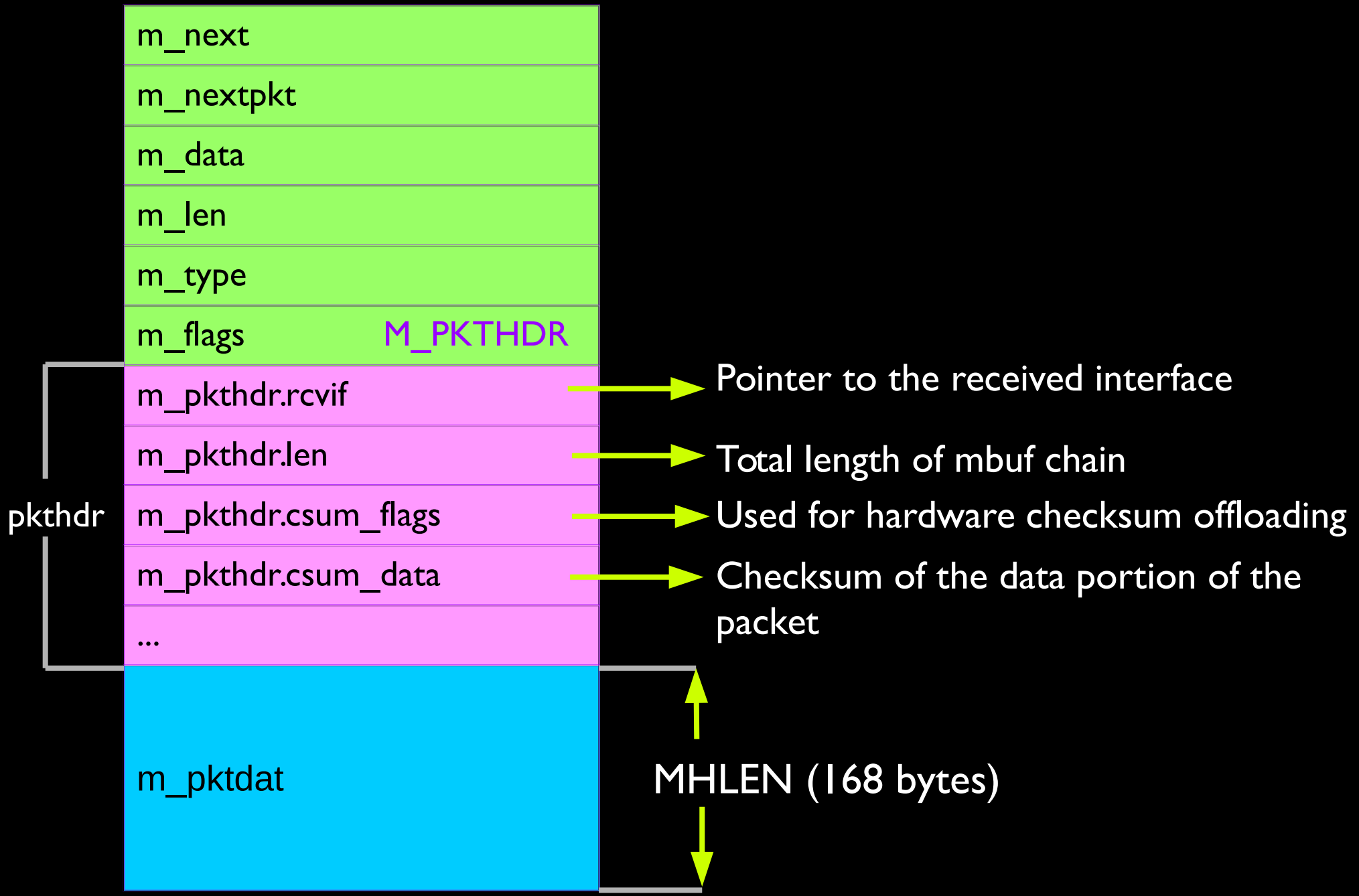
Simple mbuf


m_next
m_nextpkt
m_data
m_len
m_type
m_flags
0
m_hdr
m_dat
Pointer to the next mbuf
Pointer to the next mbuf chain
Pointer to data attached to this mbuf
Length of the data in this mbuf
Type of the data in this mbuf
Type of mbuf
MLEN (224 bytes)

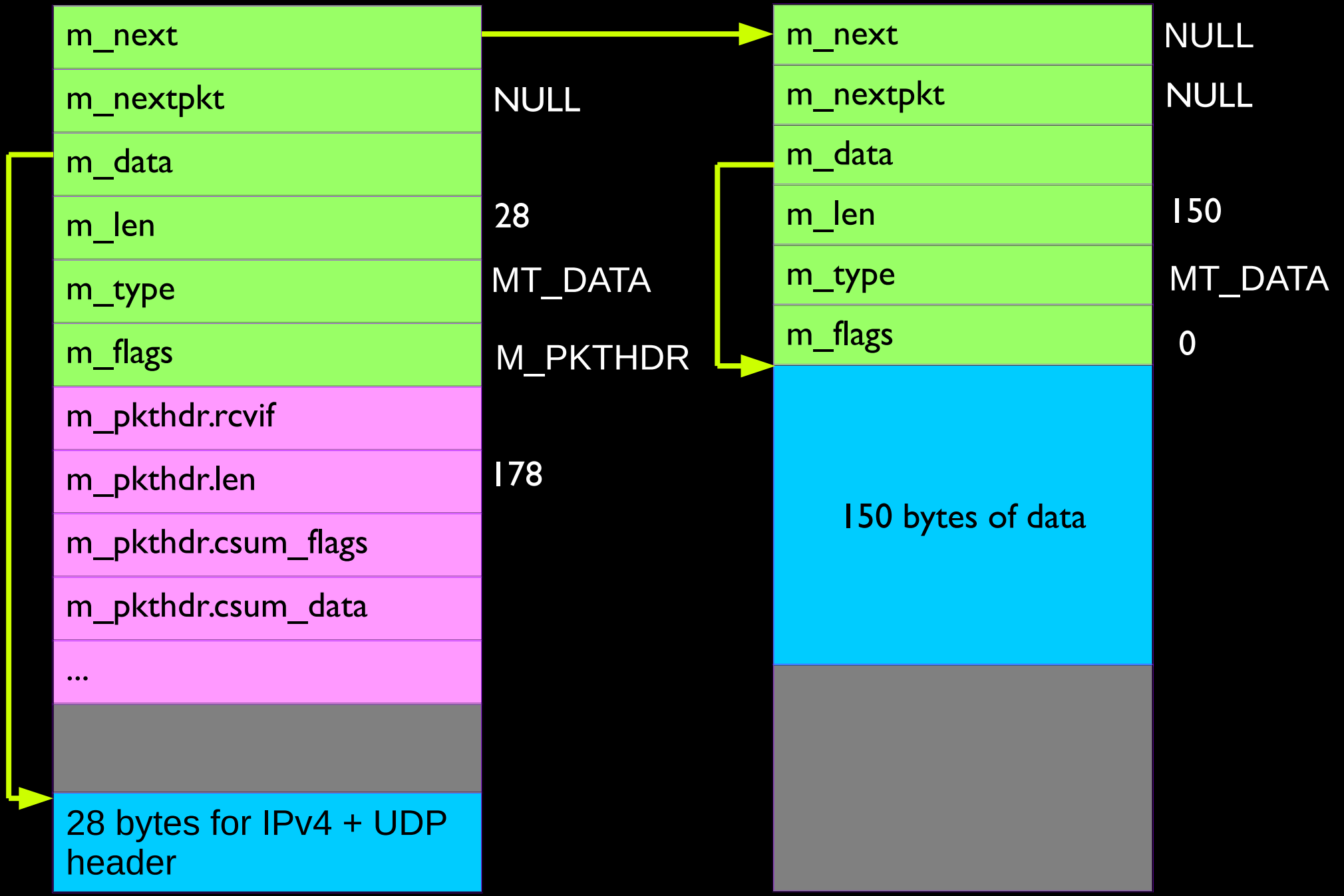
Packet header mbuf

```
struct mbuf {
        struct m_hdr    m_hdr;
        union {
                struct {
                        struct pkthdr   MH_pkthdr;      /* M_PKTHDR set */
                        union {
                                struct m_ext    MH_ext; /* M_EXT set */
                                char            MH_databuf[MHLEN];
                        } MH_dat;
                } MH;
                char M_databuf[MLEN];           /* !M_PKTHDR, !M_EXT */
        } M_dat;
};
```

Packet header mbuf



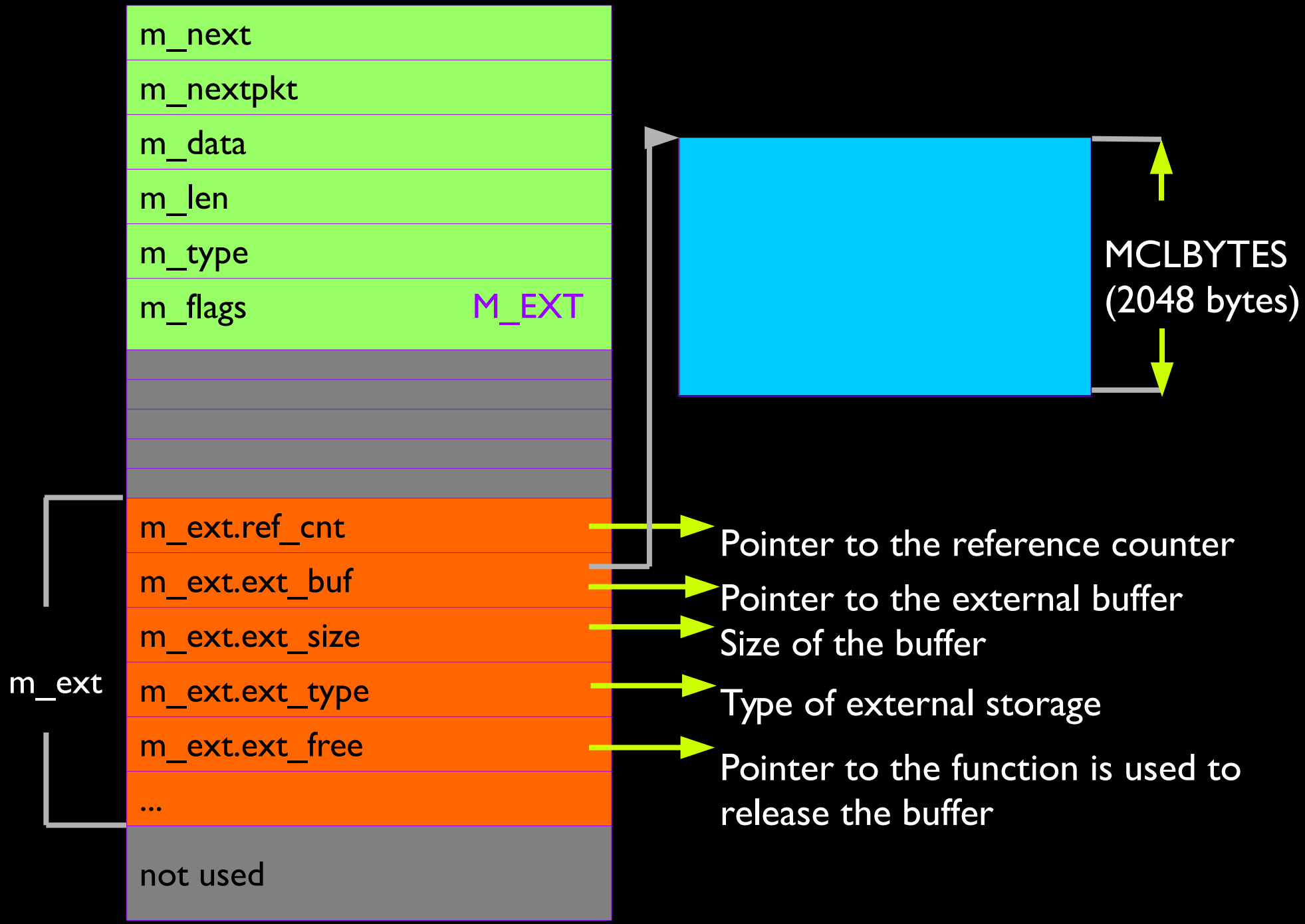
A typical UDP packet



Mbuf page cluster

```
struct mbuf {
        struct m_hdr    m_hdr;
        union {
                struct {
                        struct pkthdr   MH_pkthdr;      /* M_PKTHDR set */
                        union {
                                struct m_ext    MH_ext; /* M_EXT set */
                                char            MH_databuf[MHLEN];
                        } MH_dat;
                } MH;
                char M_databuf[MLEN];           /* !M_PKTHDR, !M_EXT */
        } M_dat;
};
```

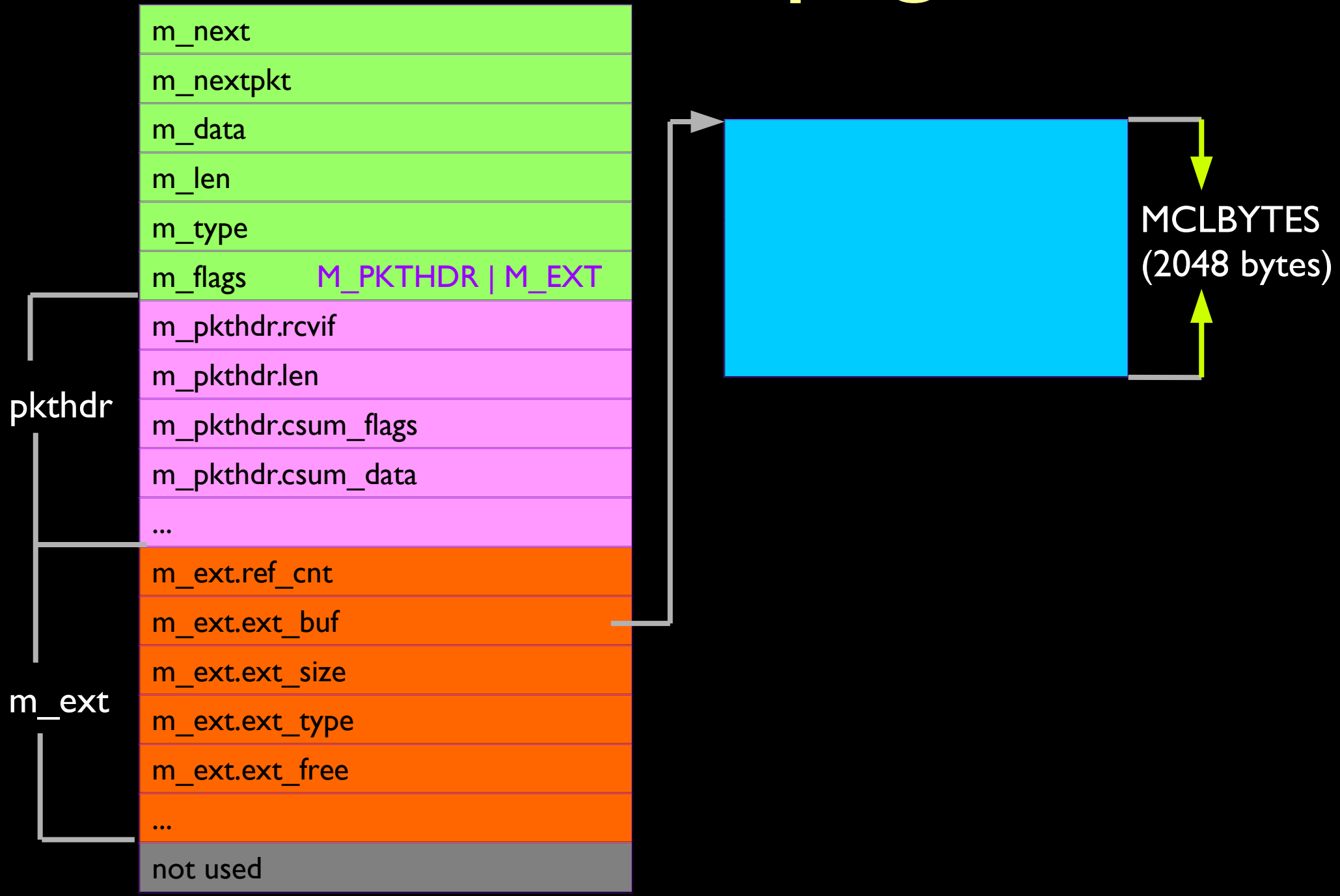
Mbuf page cluster



Packet header + page cluster

```
struct mbuf {
        struct m_hdr    m_hdr;
        union {
                struct {
                        struct pkthdr   MH_pkthdr;      /* M_PKTHDR set */
                        union {
                                struct m_ext    MH_ext; /* M_EXT set */
                                char            MH_databuf[MHLEN];
                        } MH_dat;
                } MH;
                char M_databuf[MLEN];           /* !M_PKTHDR, !M_EXT */
        } M_dat;
};
```

Packet header + page cluster



Mbuf utility routines

MGET() / m_get(): allocate an mbuf

MGETHDR() / m_gethdr(): allocate an mbuf with a packet header

MCLGET() / m_clget(): add an external cluster to an mbuf

m_free(): free a single mbuf

m_freem(): free a chain of mbufs

man mbuf

Protocol data structures

The protocol layer uses three main types of structures:

- domain structure
- protocol switch structure (protosw & ip6protosw)
- protocol control block (PCB)

Communication domains

Group of related protocols

Each has address family constant

domain structure

Defined in <sys/domain.h>

```
struct domain {
        int             dom_family;             /* AF_xxx */
        char            *dom_name;
         ...
        struct          protosw *dom_protosw, *dom_protoswNPROTOSW;
        struct          domain *dom_next;
        ...
        void            *(*dom_ifattach)(struct ifnet *);
        void            (*dom_ifdetach)(struct ifnet *, void *);
                                                /* af-dependent data on ifnet */
};
```

Supported address families

AF_LOCAL / AF_UNIX	Local communication
AF_INET	Internet version 4
AF_INET6	Internet version 6
AF_ROUTE	Link layer interface
PF_KEY	Internal key-management
AF_NATM	Asynchronous transfer mode
AF_NETGRAPH	Netgraph sockets
AF_BLUETOOTH	Bluetooth protocols
AF_INET_SDP	OFED socket direct protocol

RFC 2367 section 1.3

The PF_KEY protocol family (PF_KEY) symbol is defined in <sys/socket.h> in the same manner that other protocol families are defined. PF_KEY does not use any socket addresses.

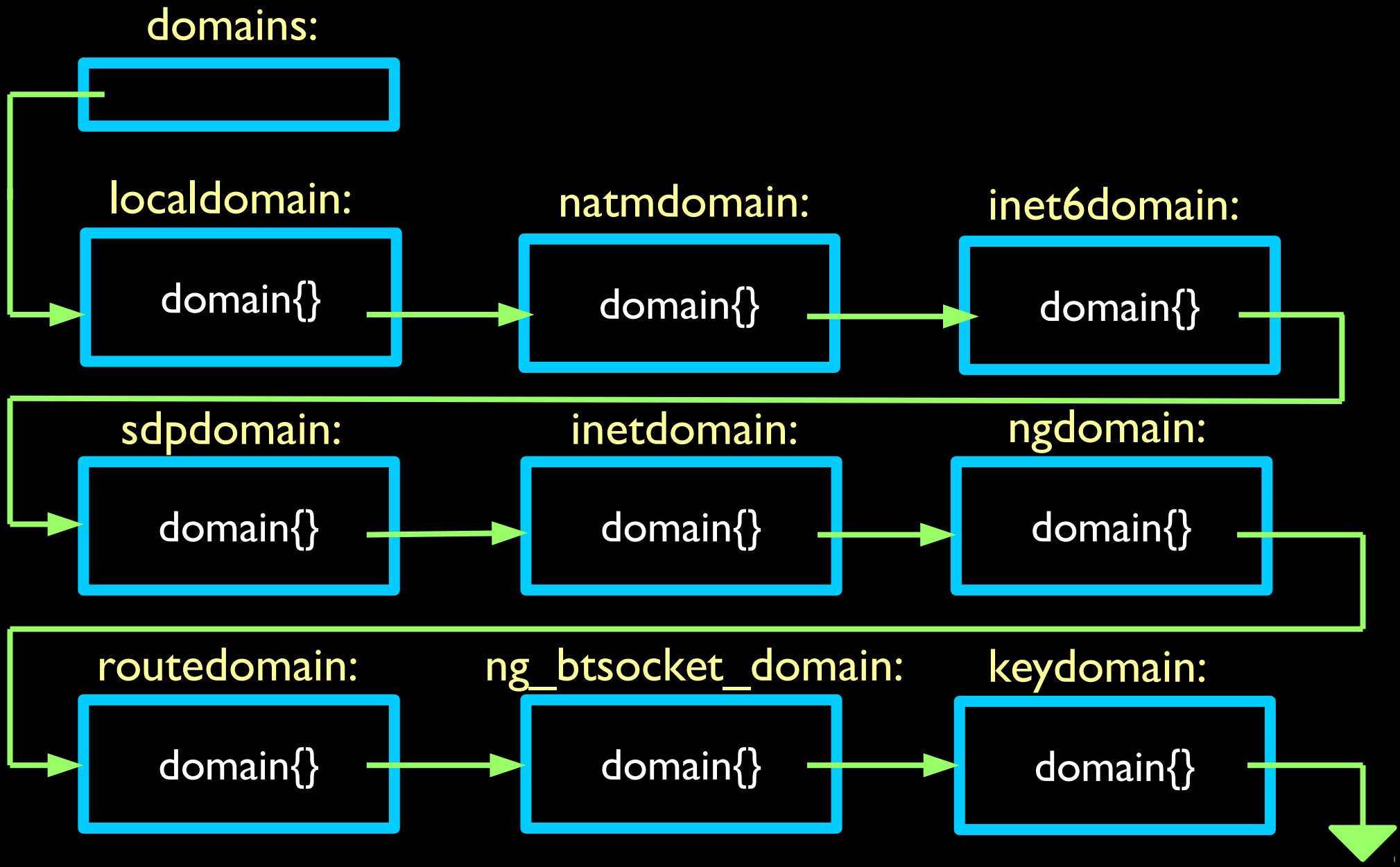
Applications using PF_KEY MUST NOT depend on the availability of a symbol named AF_KEY, but kernel implementations are encouraged to define that symbol for completeness.

```
int s;
s = socket(AF_KEY, SOCK_RAW, PF_KEY_V2);
```

inetdomain

```
struct domain inetdomain = {
        .dom_family =           AF_INET,
        .dom_name =             "internet",
        .dom_protosw =          inetsw,
        .dom_protoswNPROTOSW =  &inetsw[sizeof(inetsw)/sizeof(inetsw[0])],
#ifdef RADIX_MPATH
        .dom_rtattach =         rn4_mpath_inithead,
#else
        .dom_rtattach =         in_inithead,
#endif
#ifdef VIMAGE
        .dom_rtdetach =         in_detachhead,
#endif
        .dom_rtoffset =         32,
        .dom_maxrtkey =         sizeof(struct sockaddr_in),
        .dom_ifattach =         in_domifattach,
        .dom_ifdetach =         in_domifdetach
};
VNET_DOMAIN_SET(inet);
```

Domains list

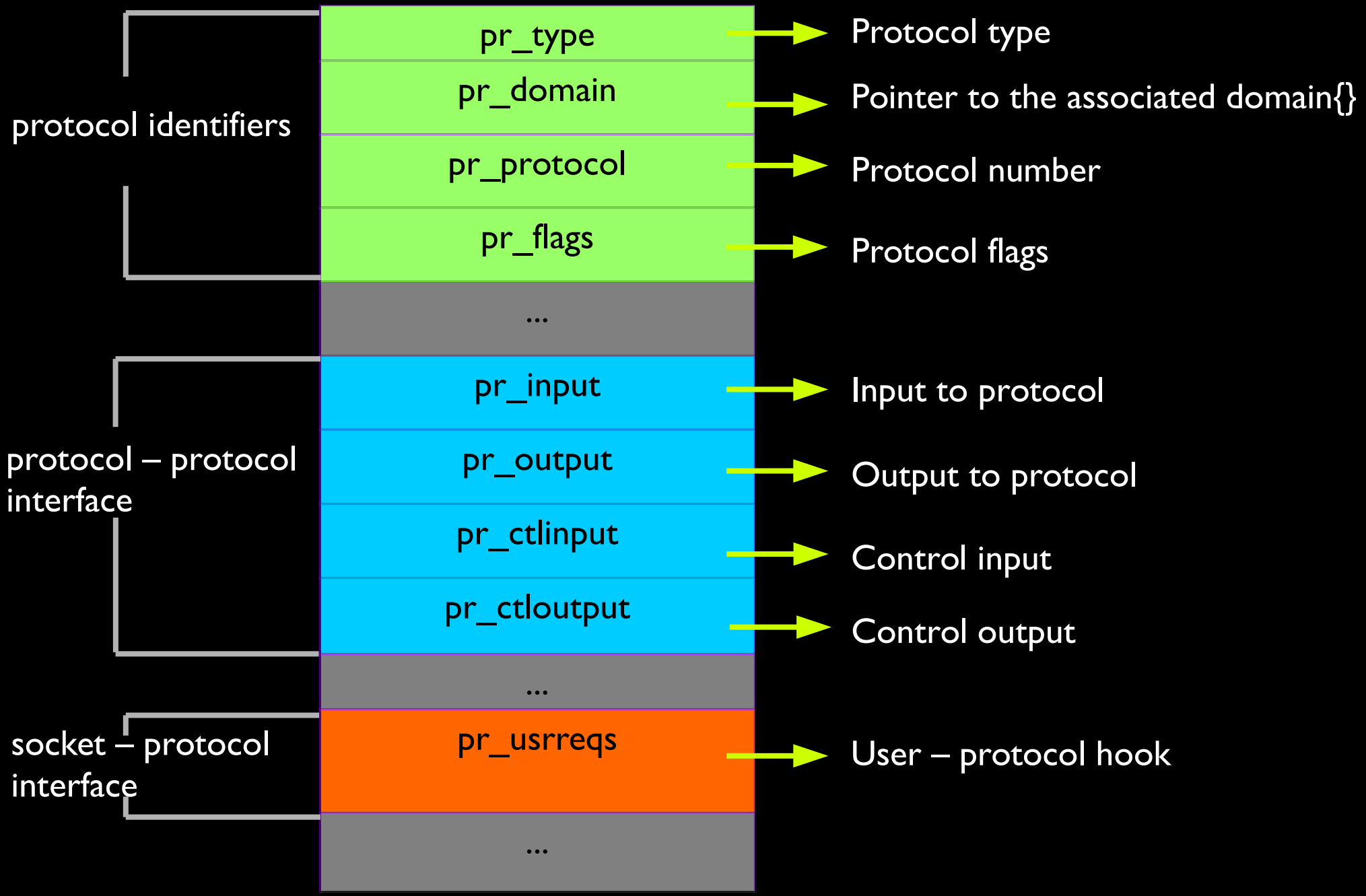


protosw structure

Defined in <sys/protosw.h>

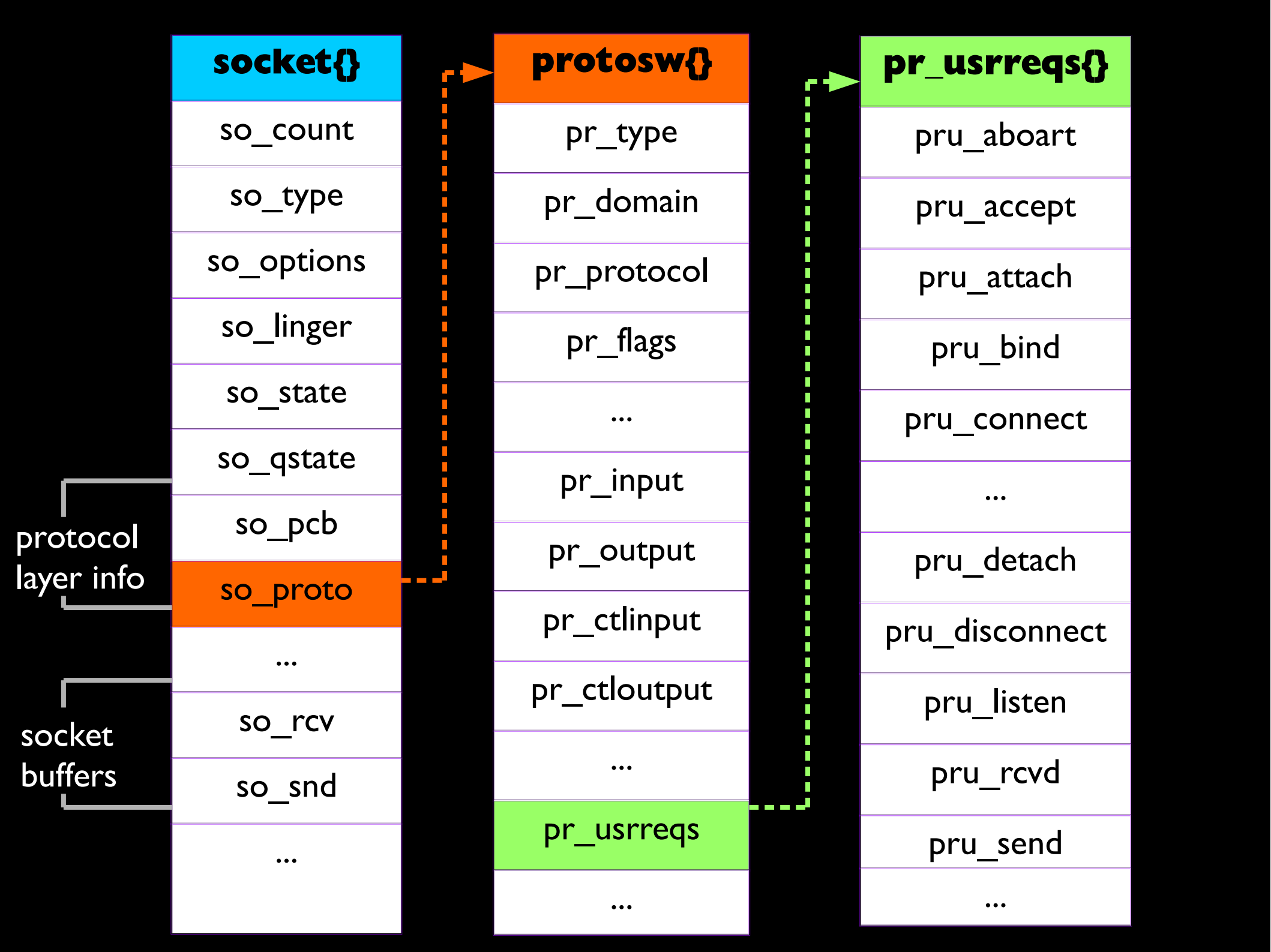
```
/* USE THESE FOR YOUR PROTOTYPES ! */
typedef void    pr_input_t (struct mbuf *, int);
typedef int     pr_input6_t (struct mbuf **, int*, int);  /* XXX FIX THIS */
typedef int     pr_output_t (struct mbuf *, struct socket *);
typedef void    pr_ctlinput_t (int, struct sockaddr *, void *);
typedef int     pr_ctloutput_t (struct socket *, struct sockopt *);
typedef void    pr_init_t (void);
typedef void    pr_destroy_t (void);
typedef void    pr_fasttimo_t (void);
typedef void    pr_slowtimo_t (void);
typedef void    pr_drain_t (void);
```

protosw structure (cont.)



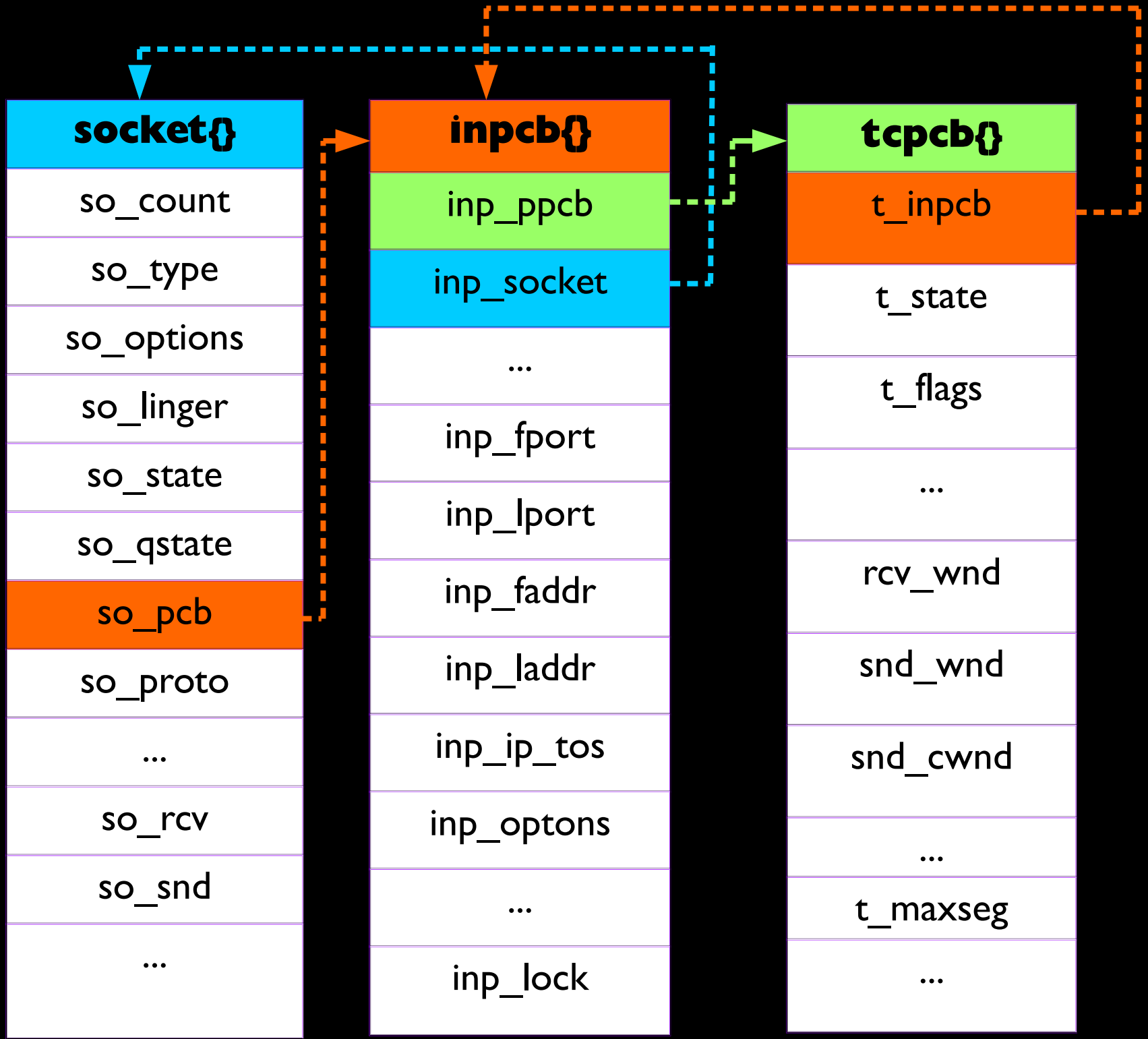
inetsw[] switch table

```
struct protosw inetsw[] = {
{
        .pr_type =              SOCK_DGRAM,
        .pr_domain =            &inetdomain,
        .pr_protocol =          IPPROTO_UDP,
        .pr_flags =             PR_ATOMIC|PR_ADDR,
        .pr_input =             udp_input,
        .pr_ctlinput =          udp_ctlinput,
        .pr_ctloutput =         udp_ctloutput,
        .pr_init =              udp_init,
#ifdef VIMAGE
        .pr_destroy =           udp_destroy,
#endif
        .pr_usrreqs =           &udp_usrreqs
},
```


socket{}
so_count
so_type
so_options
so_linger
so_state
so_qstate
so_pcb
so_proto
...
so_rcv
so_snd
...
protocol layer info
socket buffers
protosw{}
pr_type
pr_domain
pr_protocol
pr_flags
...
pr_input
pr_output
pr_ctlinput
pr_ctloutput
...
pr_usrreqs
...
pr_usrreqs{}
pru_aboart
pru_accept
pru_attach
pru_bind
pru_connect
...
pru_detach
pru_disconnect
pru_listen
pru_rcvd
pru_send
...

Protocol control block (pcb)

- Hold protocol information
- Stored as a doubly linked list
- Internet protocol control block (inpcb)
  - Foreign and local IP addresses
  - Foreign and local port numbers
  - Back pointer to socket
  - Per-protocol pcb
- TCP control block (tcpcb)
  - Protocol state information


socket{}
so_count
so_type
so_options
so_linger
so_state
so_qstate
so_pcb
so_proto
...
so_rcv
so_snd
...
inpcb{}
inp_ppcb
inp_socket
...
inp_fport
inp_lport
inp_faddr
inp_laddr
inp_ip_tos
inp_optons
...
inp_lock
tcpcb{}
t_inpcb
t_state
t_flags
...
rcv_wnd
snd_wnd
snd_cwnd
...
t_maxseg
...

Bibliography