Ethernet / TCP-IP - Training Suite

01 - LWIP Introduction

life.augmented

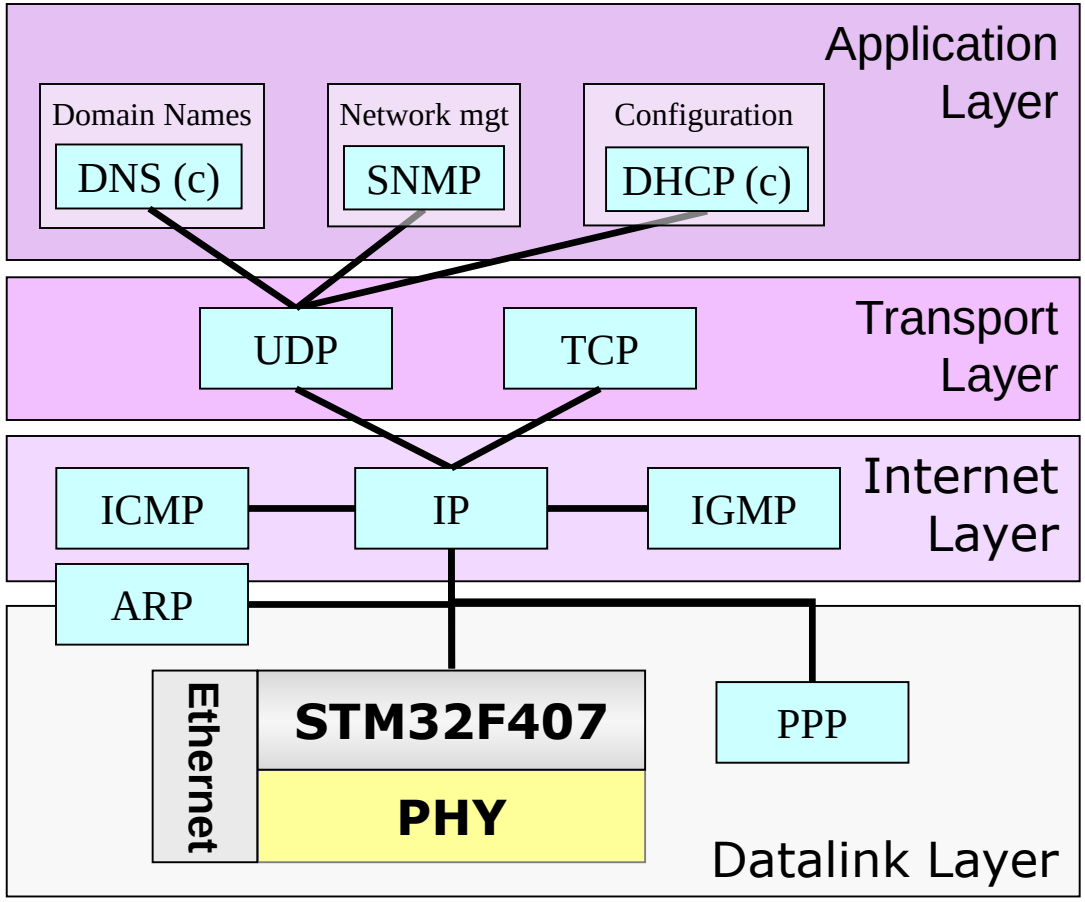
LwIP Distribution protocols

Application protocols
- SNMP,
- DNS client,
- DHCP client,

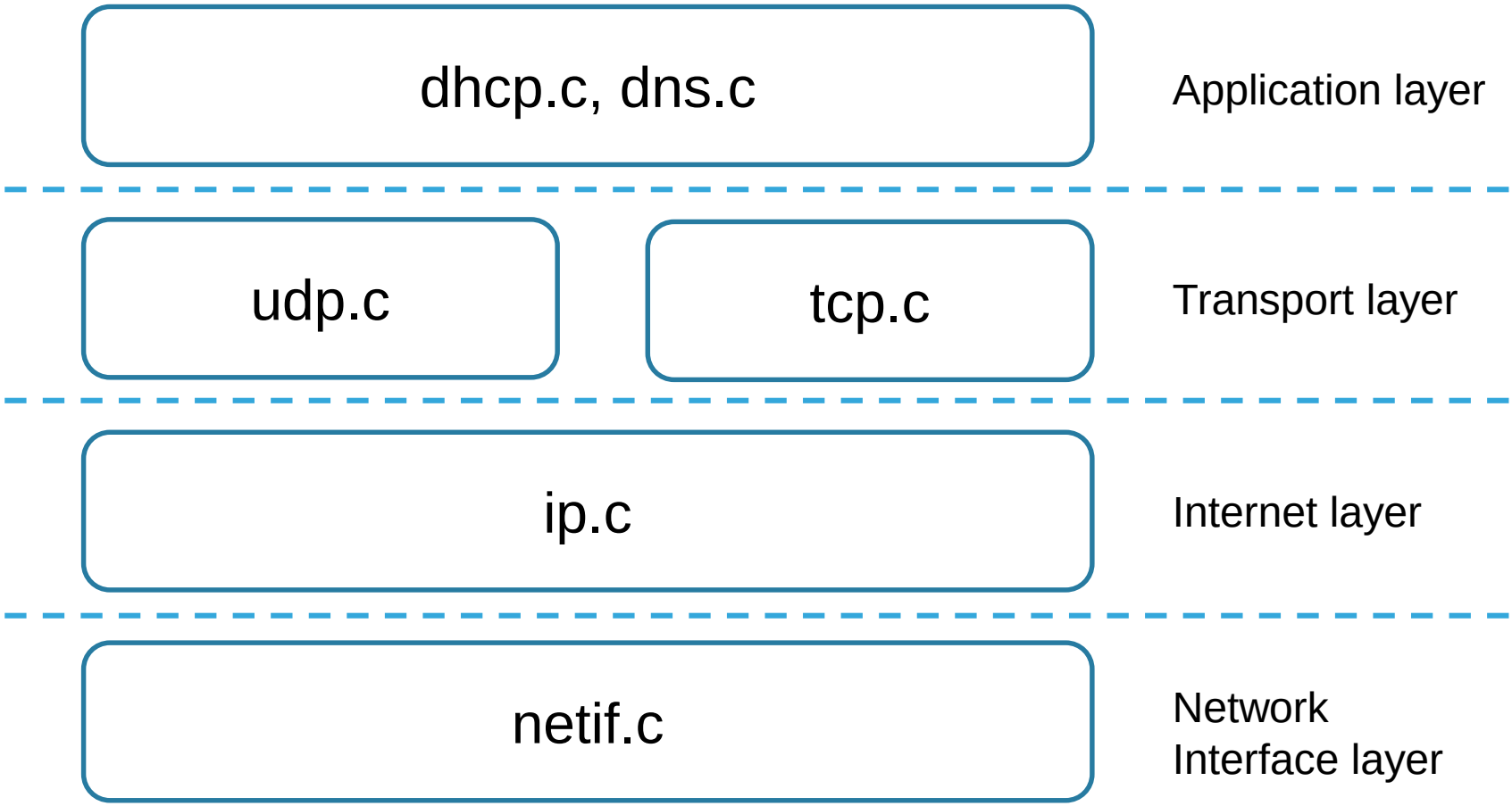
Transport protocols
- UDP,
- TCP,

Internet Protocols
- ICMP,
- IGMP,

Datalink Protocols
- ARP,
- PPP



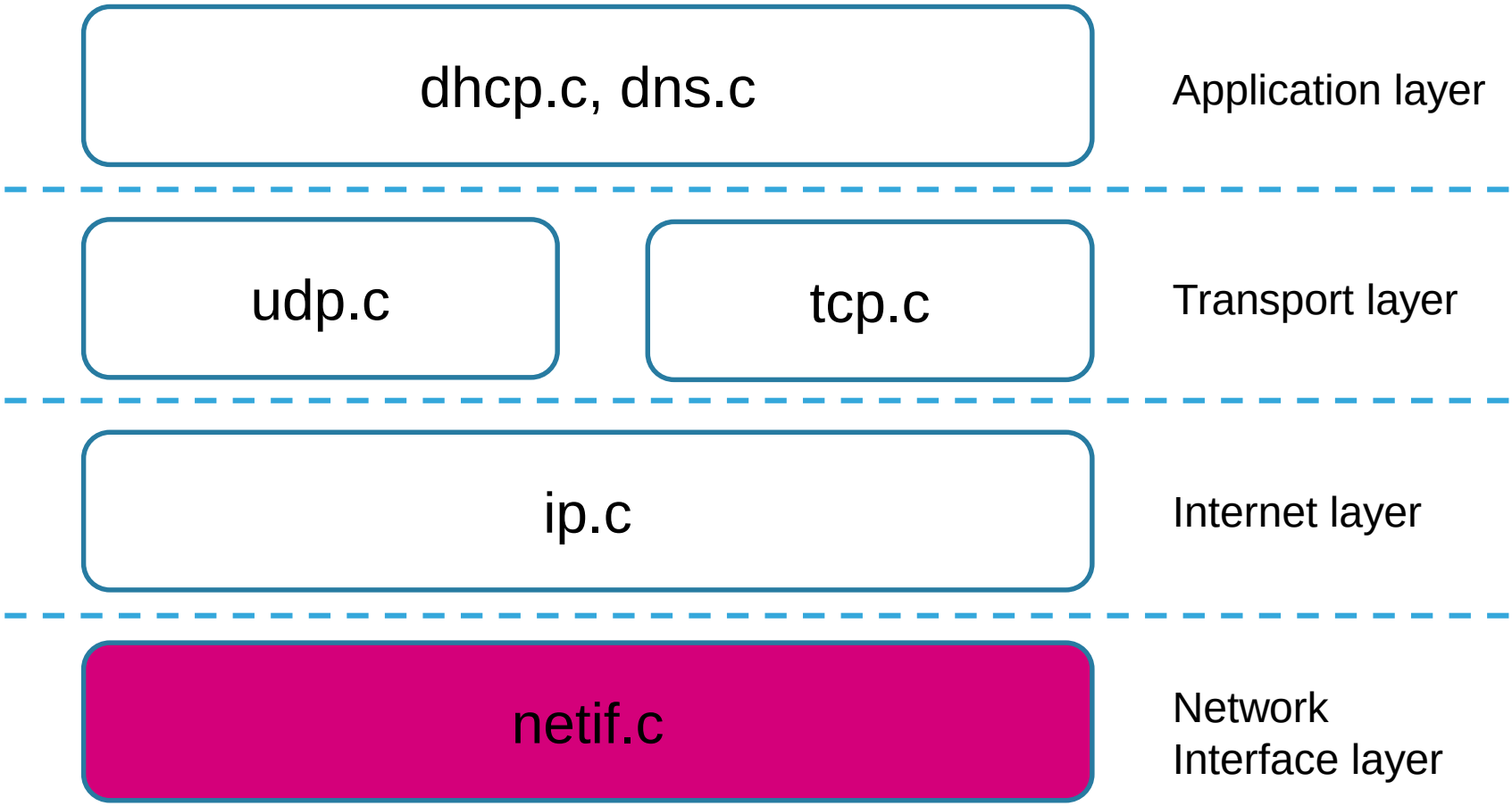
LwIP Architecture



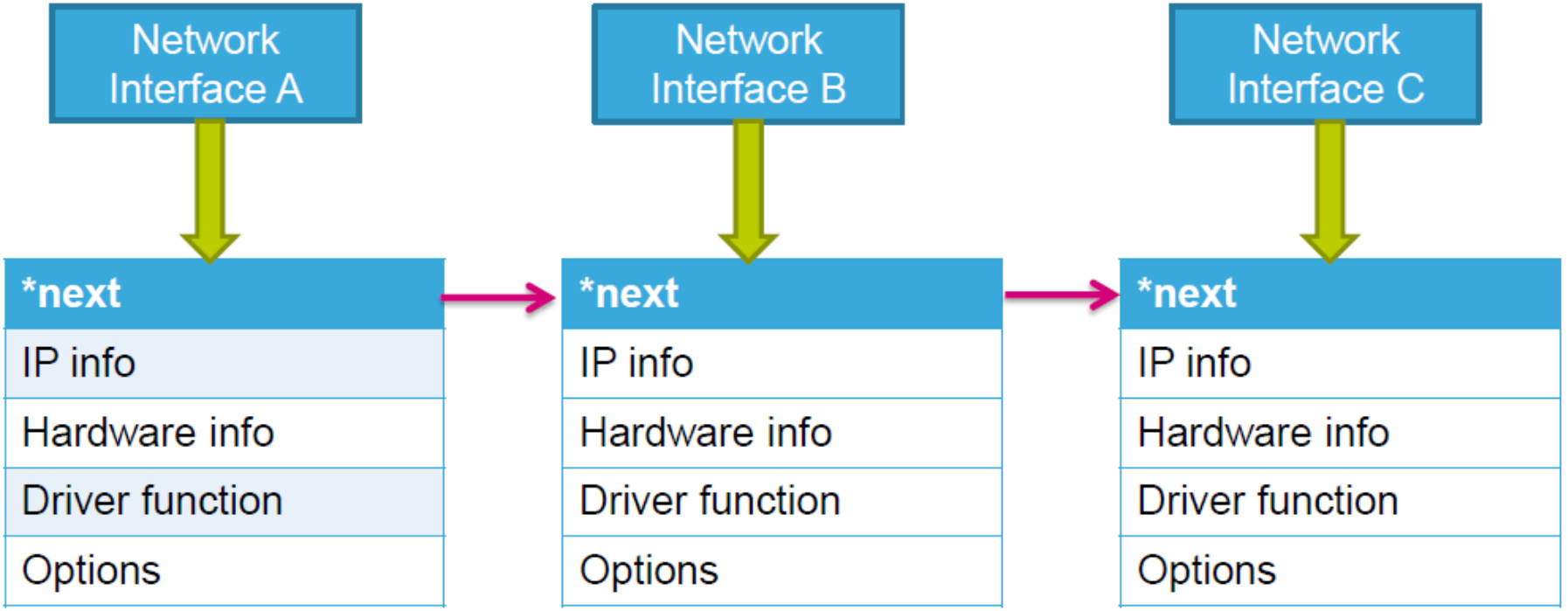
LwIP APIs

	RAW API	Netconn / Socket API
RTOS	No need	Need
Control based on	Pcb	socket
Calling methods	Callback	Close to the windows or Linux socket APIs
Structure	Core APIs	Higher level APIs
Application	➢ Lower memory devices ➢ Application without RTOS ➢ Developers has more control	➢ Higher memory devices ➢ Porting of protocols or application coming from Linux/windows
complexity	++	+
Memory	--	+

LwIP Architecture



Network Interface



Network Interface

- IP
  - IP address
  - Net mask
  - Gateway

Ensure network address and host address

- Different IP settings according to different cases
  - Static IP
  - DHCP
  - AutoIP

Network Interface

- Hardware info
  - MTU 1500bytes
  - Number of bytes used in hardware address
  - Hardware address

MAC address

  - Flags

Flags	Description
UP	Netif is up
BROADCAST	Broadcast
POINTTOPOINT	One end of a point-to-point connection
DHCP	DHCP
LINK_UP	Active link
ETHARP	ARP
ETHERNET	PPPOE
IGMP	IGMP

  - Name 2bytes
  - Number of this interface

Network Interface

- Driver Function
  - Input
  - linkoutput
  - link_callback

called by the network device driver to pass a packet up the TCP/IP stack

send a packet on the interface

This function is called when the netif link is set to up or down

Add Network Interface


```
struct netif *netif_add(struct netif *netif,
                        const ip4_addr_t *ipaddr,
                        const ip4_addr_t *netmask,
                        const ip4_addr_t *gw,
                        void *state,
                        netif_init_fn init,
                        netif_input_fn input)
```

```
int network_init(void)
{
  ip4_addr_t addr;
  ip4_addr_t netmask;
  ip4_addr_t gw;
  uint32_t start;

  tcpip_init(NULL, NULL);

  /* IP default settings, to be overridden by DHCP */

  IP4_ADDR(&addr, IP_ADDR0, IP_ADDR1, IP_ADDR2, IP_ADDR3);
  IP4_ADDR(&gw, GW_ADDR0, GW_ADDR1, GW_ADDR2,
GW_ADDR3);
  IP4_ADDR(&netmask, MASK_ADDR0, MASK_ADDR1,
MASK_ADDR2, MASK_ADDR3);

  /* add the network interface */
  netif_add(&Netif, &addr, &netmask, &gw, NULL, &ethernetif_init,
&ethernet_input);

  /* register the default network interface */
  netif_set_default(&Netif);

  netif_set_up(&Netif);

#ifdef USE_DHCP
  dhcp_start(&Netif);
#endif

  start = HAL_GetTick();
   ……
return 0;
}
```

Network Interface Driver

Function	Description
low_level_init	Calls the Ethernet driver functions to initialize the STM32F4xx Ethernet peripheral
low_level_output	Calls the Ethernet driver functions to send an Ethernet packet
low_level_input	Calls the Ethernet driver functions to receive an Ethernet packet.
ethernetif_init	Initializes the network interface structure (netif) and calls low_level_init to initialize the Ethernet peripheral
ethernetif_input	Calls low_level_input to receive a packet then provide it to the LwIP stack

Ethernet / TCP-IP - Training Suite

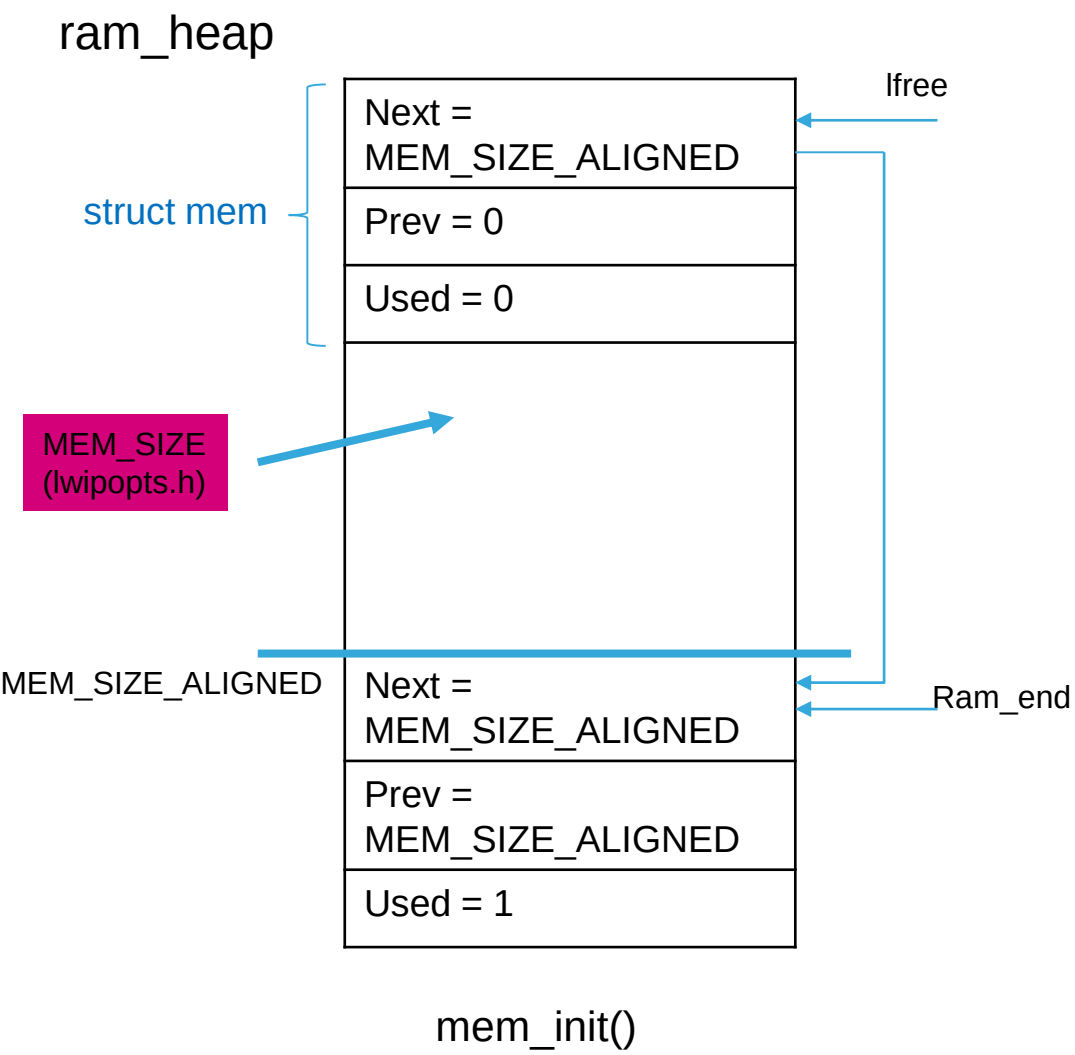
02 - lwIP Memory Management

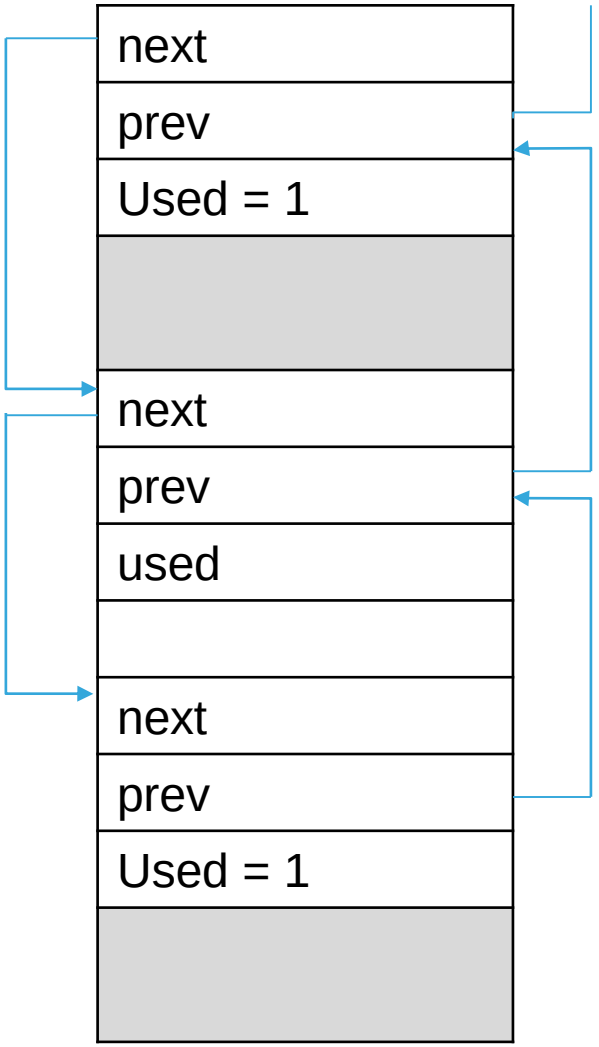
life.augmented

Dynamically allocated memory

- Lwip : Heap + Pools
- Heap (two options)
  - C standard library
  - lwIP’s custom heap-based (default), need to reverse some memory
  - Used for what (PBUF_RAM, tcp argument)
- Memp pools
  - make for fast and efficient memory allocation
  - Used for what (PCB, PBUF_POOLS & ROM…)
  - Need to reverse some memory

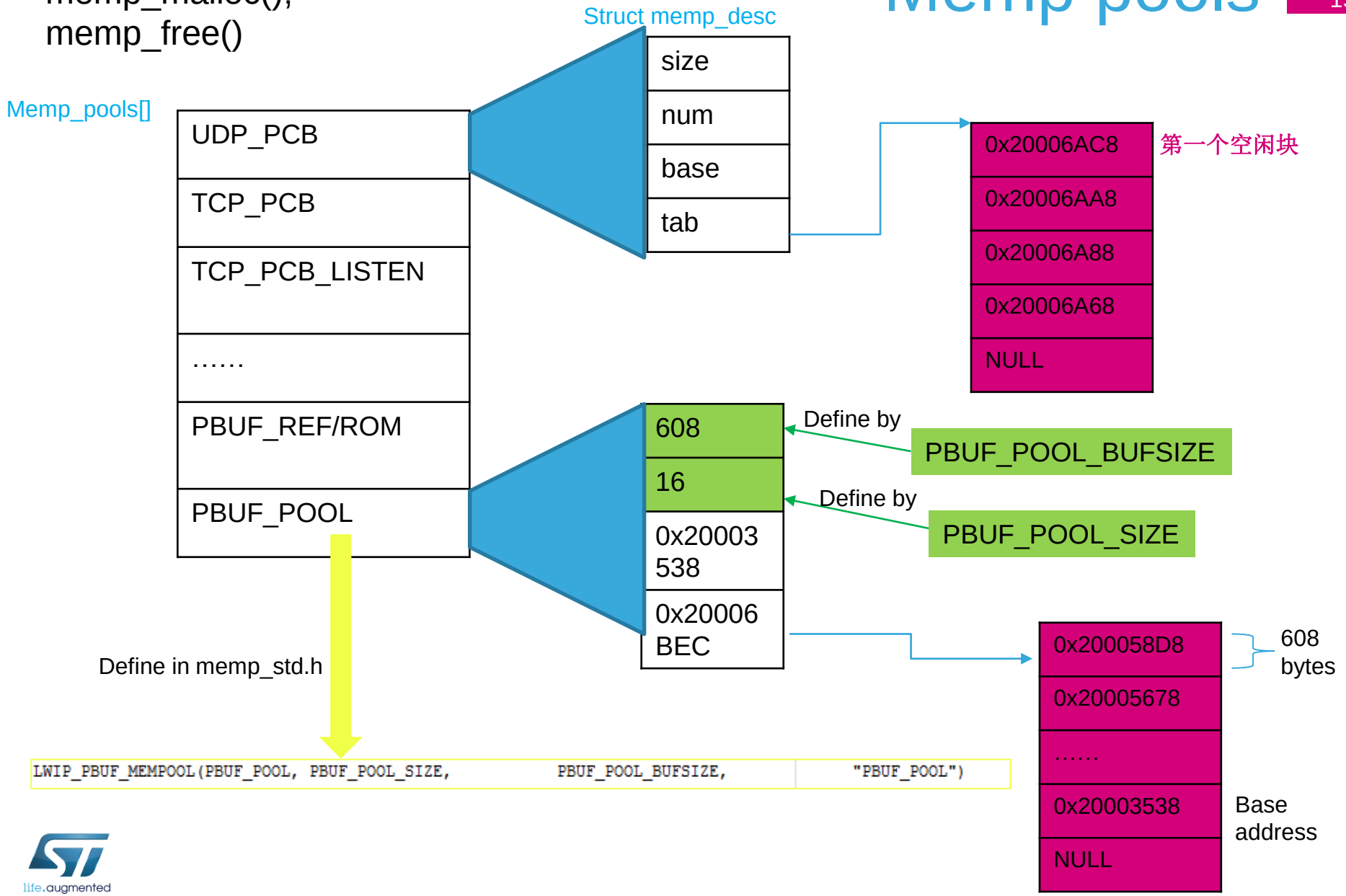
Heap


ram_heap
struct mem
Next = MEM_SIZE_ALIGNED
Prev = 0
Used = 0
lfree
MEM_SIZE (lwipopts.h)
MEM_SIZE_ALIGNED
Next = MEM_SIZE_ALIGNED
Prev = MEM_SIZE_ALIGNED
Used = 1
Ram_end
mem_init()


next
prev
Used = 1
next
prev
used
next
prev
Used = 1
mem_malloc()
mem_free()

Memp pools

memp_init();
memp_malloc();
memp_free()



life.augmented

LwIP Buffer management

- Requirements:
  - Can operate no fixed length buffer freely
  - Can add data to the head and tail (e.g. tcp to ip)
  - Can remove data from buffer (e.g. ip to tcp)
  - For performance, avoid copying

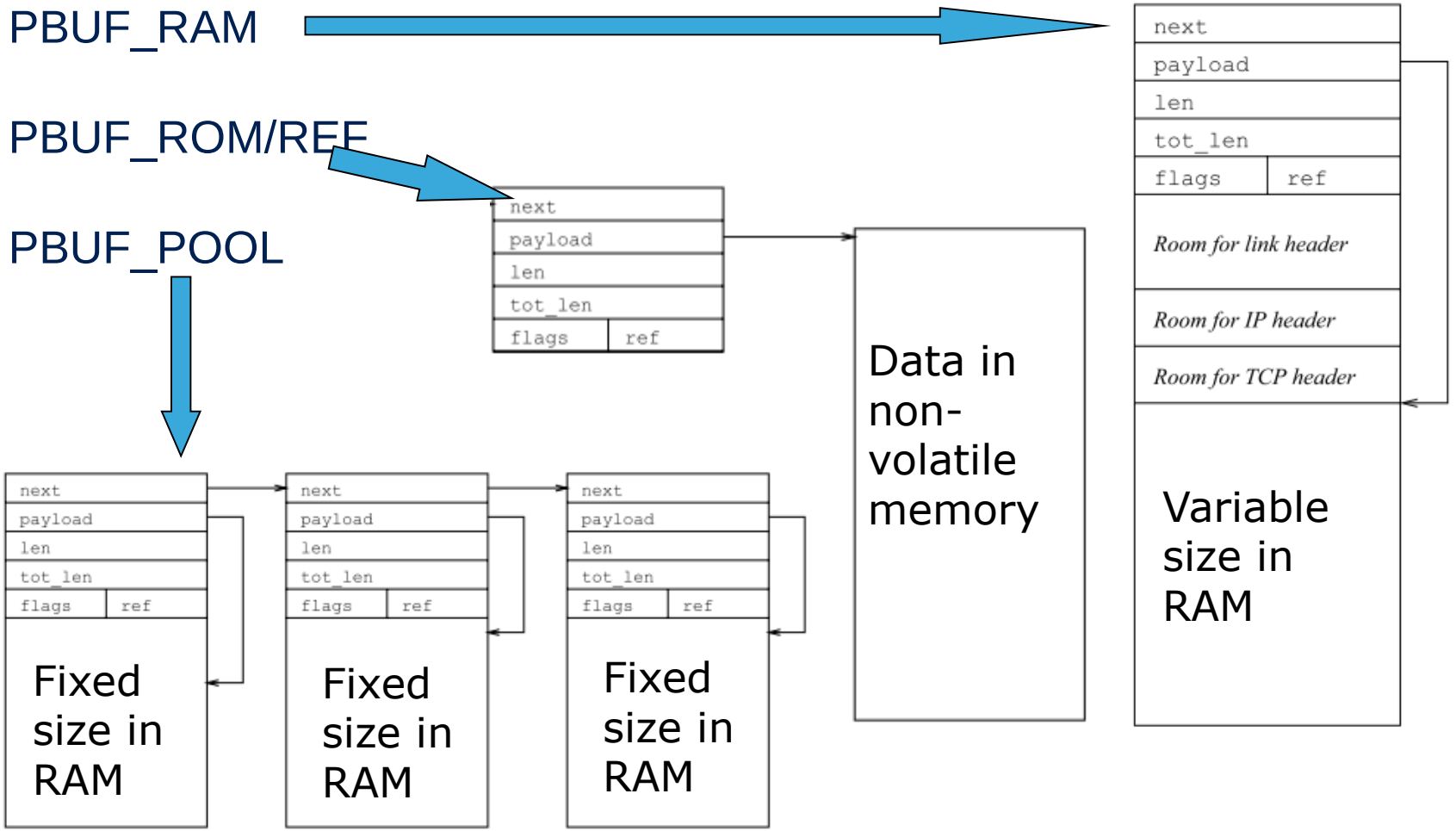
Pbuf struct

```
/** Main packet buffer struct */
struct pbuf {
  /** next pbuf in singly linked pbuf chain */
  struct pbuf *next;

  /** pointer to the actual data in the buffer */
  void *payload;

  /**
   * total length of this buffer and all next buffers in chain
   * belonging to the same packet.
   *
   * For non-queue packet chains this is the invariant:
   * p->tot_len == p->len + (p->next? p->next->tot_len: 0)
   */
  u16_t tot_len;

  /** length of this buffer */
  u16_t len;

  /** pbuf_type as u8_t instead of enum to save space */
  u8_t /*pbuf_type*/ type;

  /** misc flags */
  u8_t flags;

  /**
   * the reference count always equals the number of pointers
   * that refer to this pbuf. This can be pointers from an application,
   * the stack itself, or pbuf->next pointers from a chain.
   */
  u16_t ref;
};
```

LwIP Buffer management (Avoid copying)

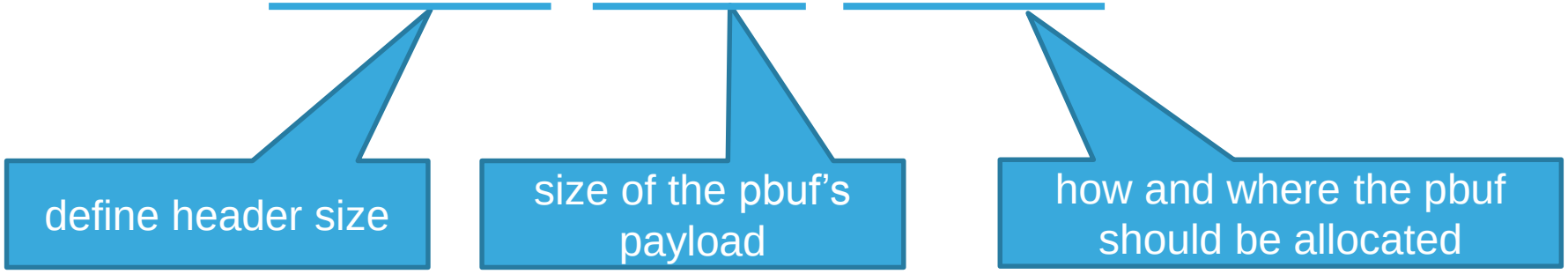
LwIP has 4 types of packet buffer structure :

- PBUF_RAM
- PBUF_ROM/REF
- PBUF_POOL



Pbuf function

- pbuf_alloc(pbuf_layer layer, u16_t length, pbuf_type type)

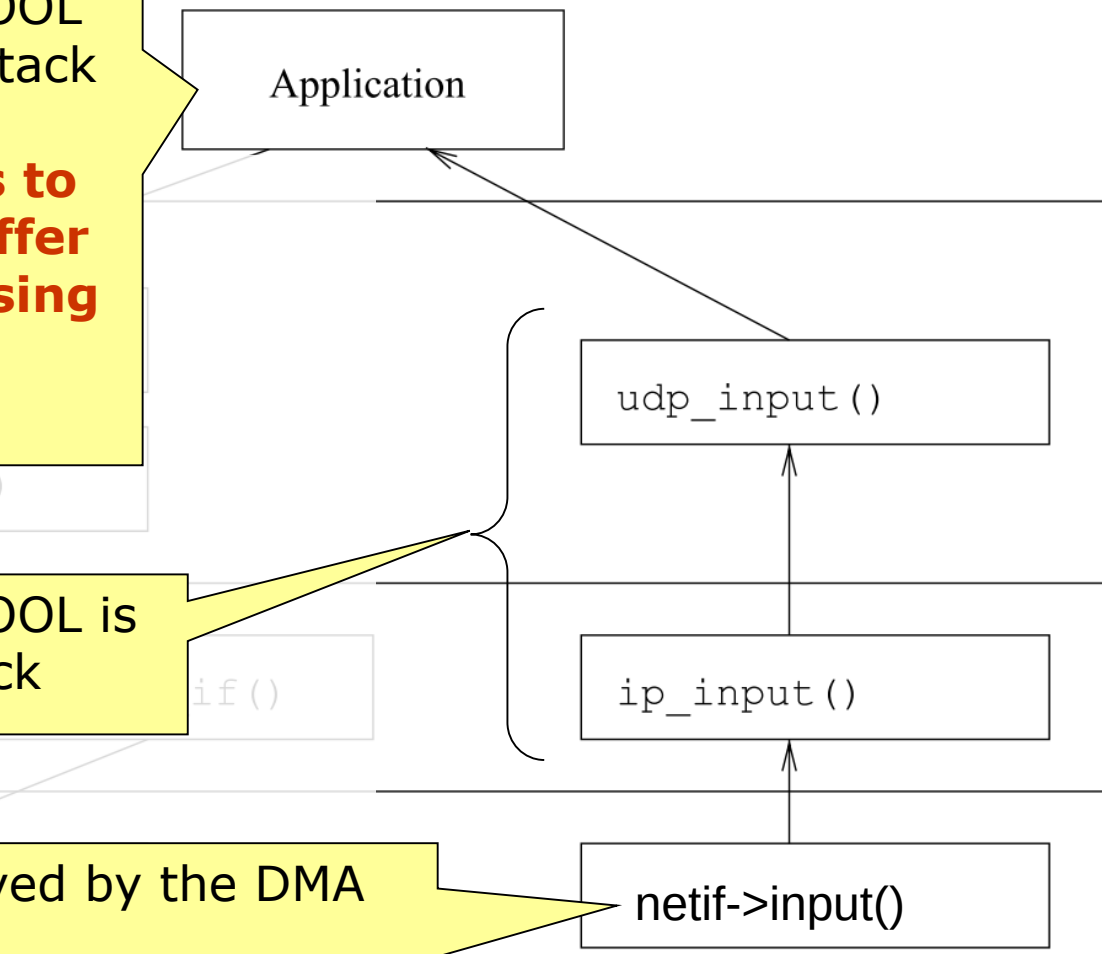


- pbuf_free(struct pbuf *p)

UDP Rx flow and Buffer Management example

The pointer to the PBUF_POOL is provided to the TCP-IP Stack

The User application has to Free the PBUF_POOL Buffer at the end of the processing



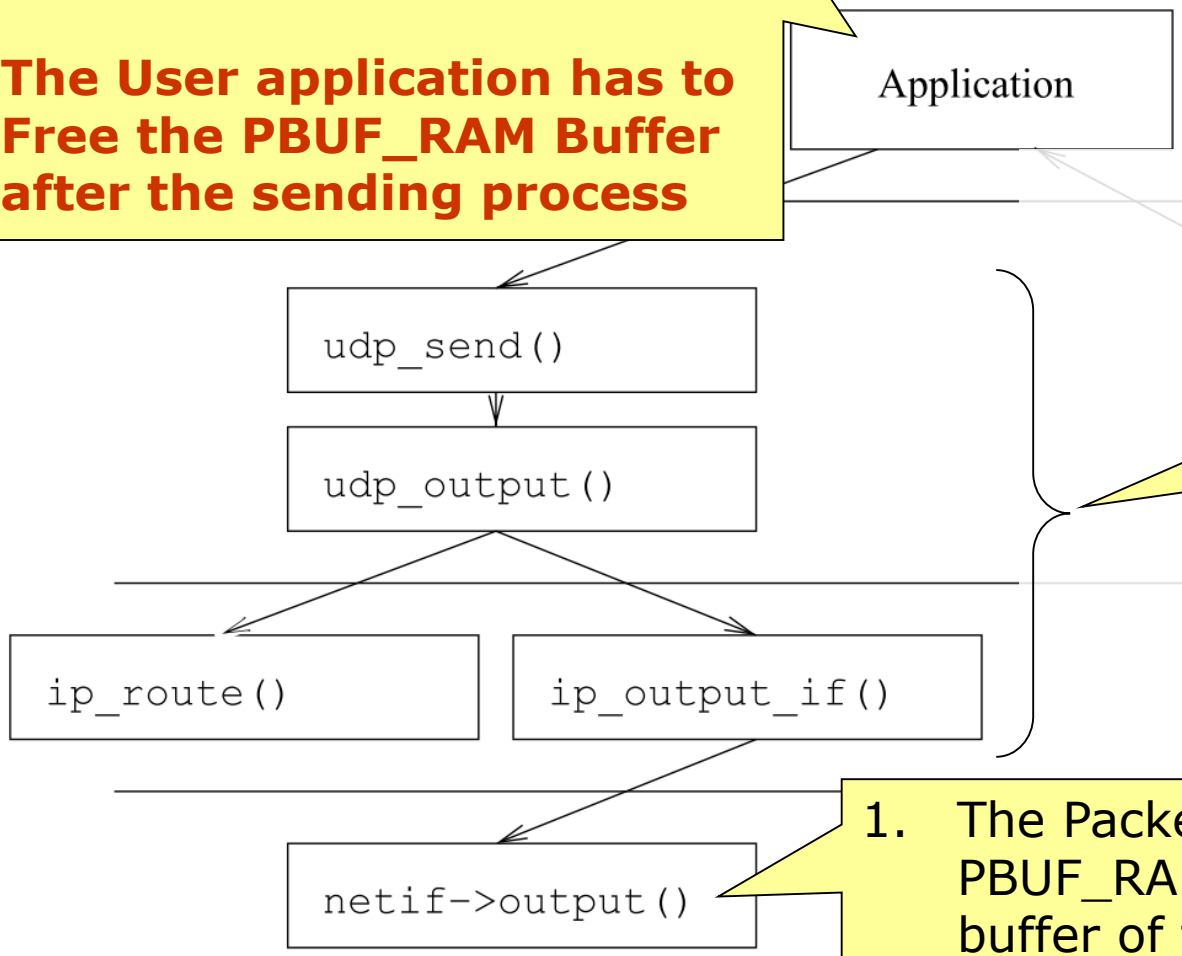
The pointer to the PBUF_POOL is provided to the TCP-IP Stack

1. The packet is first received by the DMA into dedicated buffers.
2. The Packet is the copied in a PBUF_POOL allocated by the driver

UDP Tx flow and Buffer Management example

A PBUF_RAM buffer is allocated by the user application

The User application has to Free the PBUF_RAM Buffer after the sending process



The pointer to the PBUF_RAM is provided to the TCP-IP Stack

1. The Packet is the copied from the PBUF_RAM buffer into a dedicated buffer of the driver.
2. The packet is send by the DMA

Memory footprint and module configuration

- Target: decide the number of your code size and RAM usage
- Principle: depending on your application
- Ways:
  - Simple configuration
  - Advanced configuration

Simple Configuration (lwipopt.h)

- Which API do you use? RAW / NETCONN / SOCKET
- Which protocols do you use? TCP / UDP / DHCP / SNMP
- How to configure the memory to fit your application?
  - Heap memory: mainly used to send

    #define MEM_SIZE
  - Memp pools memory: mainly used to receive

    #define PBUF_POOL_SIZE

    #define PBUF_POOL_BUFSIZE
  - Protocol control block

    #define MEMP_NUM_UDP_PCB //max number of simultaneous UDP structure

    #define MEMP_NUM_TCP_PCB ..

    #define MEMP_NUM_TCP_PCB_LISTEN …

Advanced configuration (opt.h)

- #include “lwipopt.h”
- Configurations:
  - Memory allocation algorithm
  - The options of all supported protocols
  - Timeout
  - Debug
  - If you need to configure these options, you must know what you do.

Ethernet / TCP-IP - Training Suite

03 - lwIP PCB Structure

life.augmented

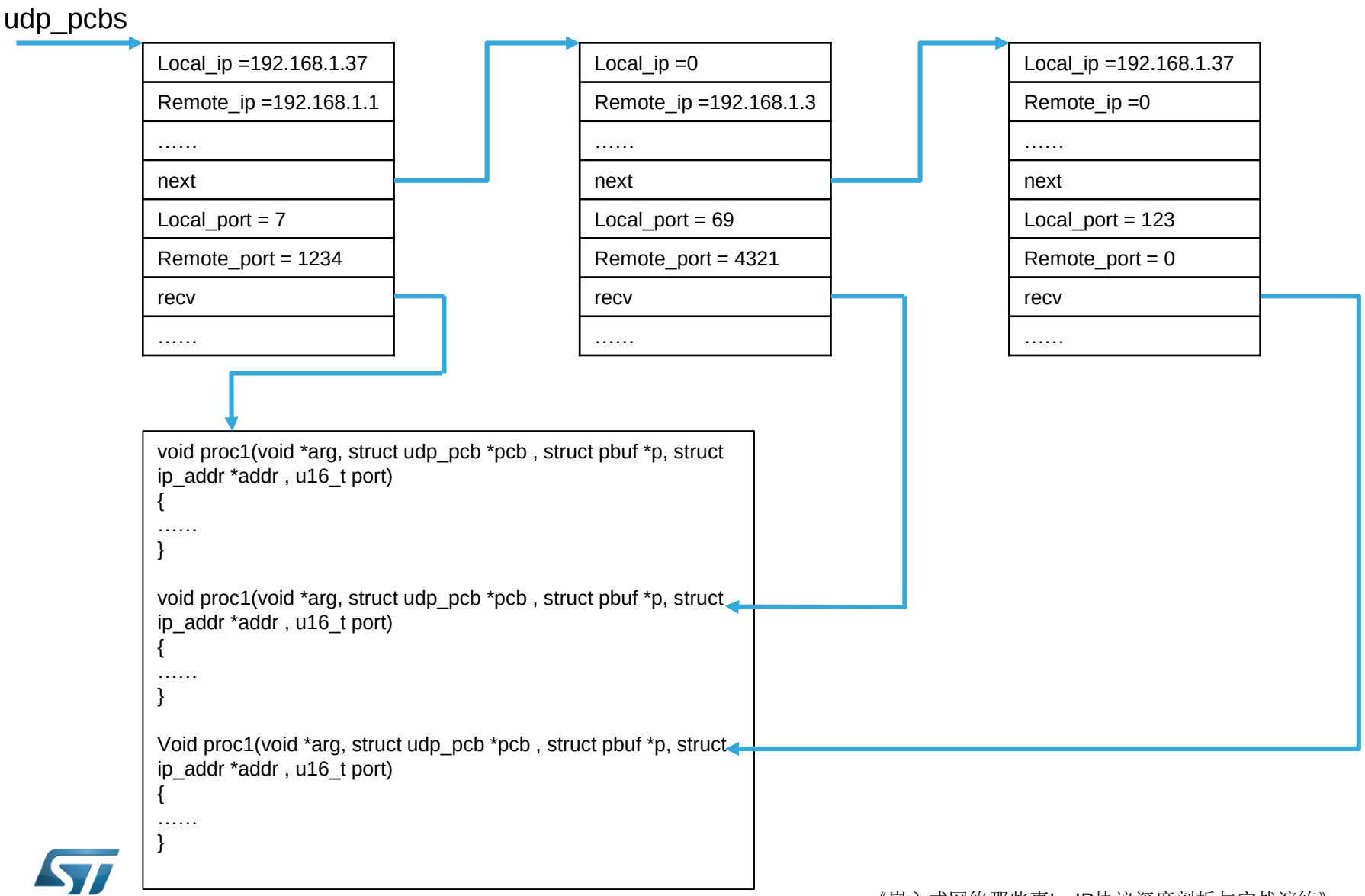
UDP Data Structure

udp.h

```
#define UDP_HLEN 8

struct udp_hdr {
  PACK_STRUCT_FIELD(u16_t src);
  PACK_STRUCT_FIELD(u16_t dest);  /* src/dest UDP ports */
  PACK_STRUCT_FIELD(u16_t len);
  PACK_STRUCT_FIELD(u16_t chksum);
} PACK_STRUCT_STRUCT;


struct udp_pcb {
IP_PCB; /* Common members of all PCB types */

struct udp_pcb *next; /* Protocol specific PCB members */

  u8_t flags;

  /** ports are in host byte order */
  u16_t local_port, remote_port;

  /** receive callback function */
  udp_recv_fn recv;
  /** user-supplied argument for the recv callback */
  void *recv_arg;
};
```

```
#define IP_PCB \
  IP_PCB_ISIPV6_MEMBER \
  /* ip addresses in network byte order */ \
  ip_addr_t local_ip; \
  ip_addr_t remote_ip; \
   /* Socket options */  \
  u8_t so_options;      \
   /* Type Of Service */ \
  u8_t tos;             \
  /* Time To Live */     \
  u8_t ttl              \
  /* link layer address resolution hint */ \
  IP_PCB_ADDRHINT

struct ip_pcb {
/* Common members of all PCB types */
  IP_PCB;
};
```

UDP PCB Chain



UDP Raw API presentation

…udp_new(…)

…udp_remove(…)

Create or remove an UDP Structure

…udp_bind(…)

…udp_connect(…)

…udp_disconnect(…)

Initialize the UDP Structure with
- the device **internal** IP address and port
- the remote (**external**) IP address and port

Clear remote IP address and port

…udp_recv(…)

Set a receive callback for a UDP PCB

…udp_send(…)

…udp_sendto(…)

…udp_sendto_if(…)

Function to call when sending a Packet
- Using the default target (**external**) IP address and the default interface
- Using a specific target (**external**) IP address but the default interface
- Using a specific target (**external**) IP address and a specific interface

UDP Connection setup example


```
Demo_upcb = udp_new();
if (Demo_upcb ==NULL)
{ //the PCB data structure cannot be allocated.
   return UDP_MemoryError;}
```

Create an UDP Structure

Error Management

```
Error = udp_bind(Demo_upcb, IP_ADDR_ANY, UDP_Listen_PORT);
if (Error != ERR_OK)
{ // the UDP Structure is a
   return DP_AlreadyBinded
```

Initialize the UDP Structure with
- the device **internal** IP address (any)
- the user application **internal** port

Error Management

```
udp_recv(Demo_upcb, &udp_client_callback, NULL);
```

Initialize the UDP Structure with the callback function (`udp_client_callback`) to use when a packet is received

Now your application is ready to receive data

UDP Connection CallBack example


```
void udp_client_callback(void *arg,
struct udp_pcb *upcb,
struct pbuf *pcallback,
struct ip_addr *addr,
u16_t port)
{
…
```

Callback function parameter
upcb : the UDP connection used
pcallback : the pointer to the received buffer
addr : the source IP address
port : the source port number

```
Error = pbuf_copy_partial(pcallback, UDP Data, 4, 0);
if (Error == 0)
{ CallBackError = UDP_MemoryError;
  pbuf_free(pcallback);
  return; }
```

Copy data from a PBUF_POOL, the pbuf API has to be used

If an error occurs, the user application has to free the buffer

Error Management

```
pbuf_free(pcallback);
…
}
```

After processing the receivefata, the user application has to free the packet

UDP Connection : send data example


```
Error = udp_connect(Demo_upcb, addr, port);
if( Error != ERR_OK)
{ return UDP_ConnectionE
```

Error Management

Initialize the UDP Structure with
- the target (**external**) IP address (any)
- the target application (**external**) port number

```
psend = pbuf_alloc(PBUF_TRANSPORT,len, PBUF_RAM);
if (psend==NULL)
{ return UDP_MemoryError; }
```

Allocate a PBUF_RAM to send the data

Error Management

```
memcpy (psend->payload, data, len);
```

Copy data into the PBUF_RAM

Send the data

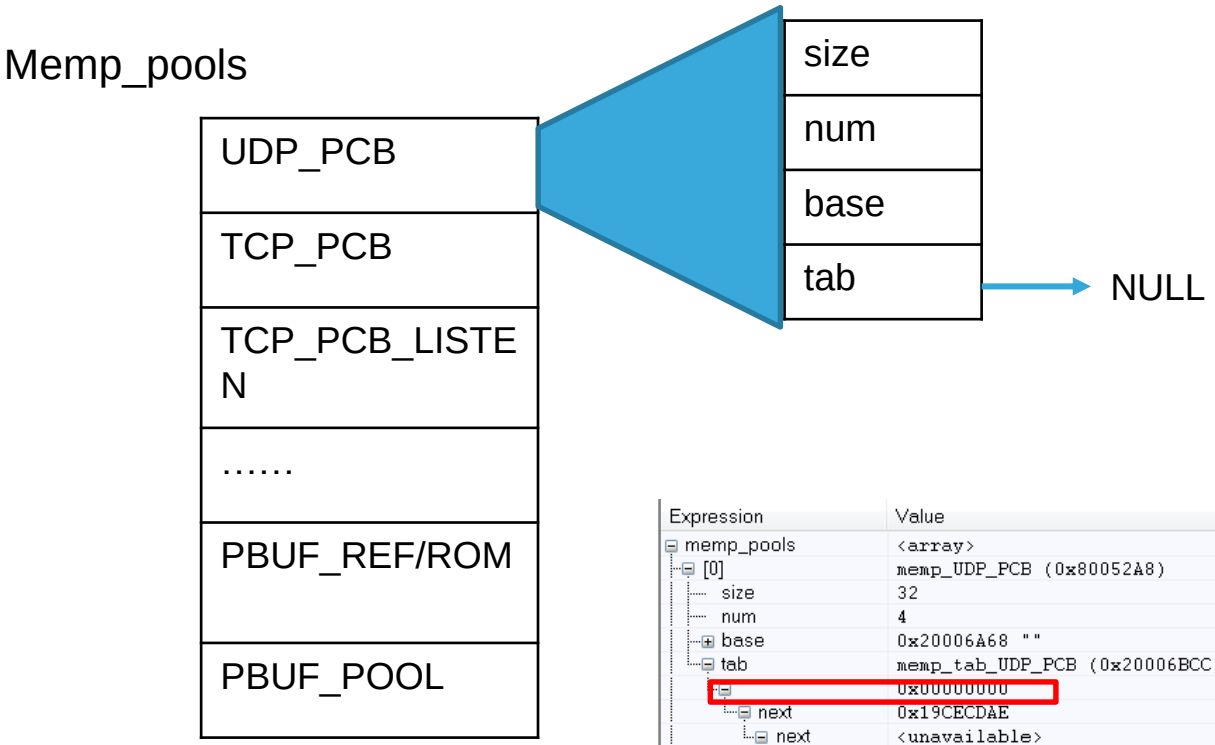
```
Error = udp_send(Demo_upcb, psend);
if (Error != ERR_OK)
{pbuf_free(psend);
return UDP_SendNOK;     }
```

If an error occurs, the user application has to free the packet

```
pbuf_free(psend);
```

After sending the Data, the user application has to free the packet

Build UDP Connection

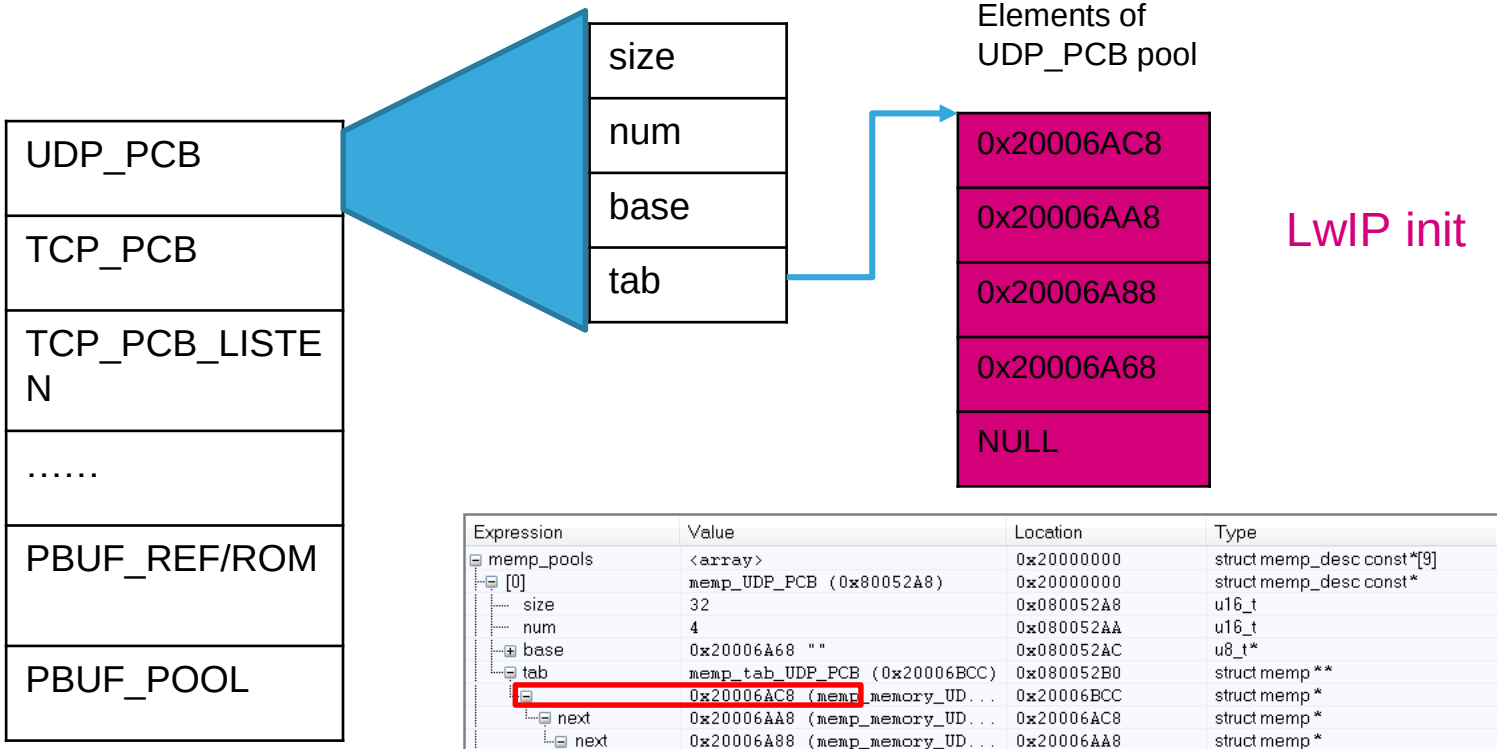


System Reset

Expression	Value	Location	Type
memp_pools	<array>	0x20000000	struct memp_desc const *[9]
[0]	memp_UDP_PCB (0x80052A8)	0x20000000	struct memp_desc const *
size	32	0x080052A8	u16_t
num	4	0x080052AA	u16_t
base	0x20006A68 ""	0x080052AC	u8_t*
tab	memp_tab_UDP_PCB (0x20006BCC)	0x080052B0	struct memp **
	0x00000000	0x20006BCC	struct memp *
next	0x19CECDAE	0x00000000	struct memp *
next	<unavailable>	0x19CECDAE	struct memp *
	<error>		
[1]	memp_TCP_PCB (0x80052B4)	0x20000004	struct memp_desc const *
[2]	memp_TCP_PCB_LISTEN (0x800...	0x20000008	struct memp_desc const *
[3]	memp_TCP_SEG (0x80052CC)	0x2000000C	struct memp_desc const *
[4]	memp_REASSDATA (0x80052D8)	0x20000010	struct memp_desc const *
[5]	memp_FRAG_PBUF (0x80052E4)	0x20000014	struct memp_desc const *
[6]	memp_SYS_TIMEOUT (0x80052F0)	0x20000018	struct memp_desc const *
[7]	memp_PBUF (0x80052FC)	0x2000001C	struct memp_desc const *
[8]	memp_PBUF_POOL (0x8005308)	0x20000020	struct memp_desc const *
udp_pcbs	0x00000000	0x20000048	struct udp_pcb *
pcb	Error (col 1): Unknown or ...		
<click to edit>			

Build UDP Connection

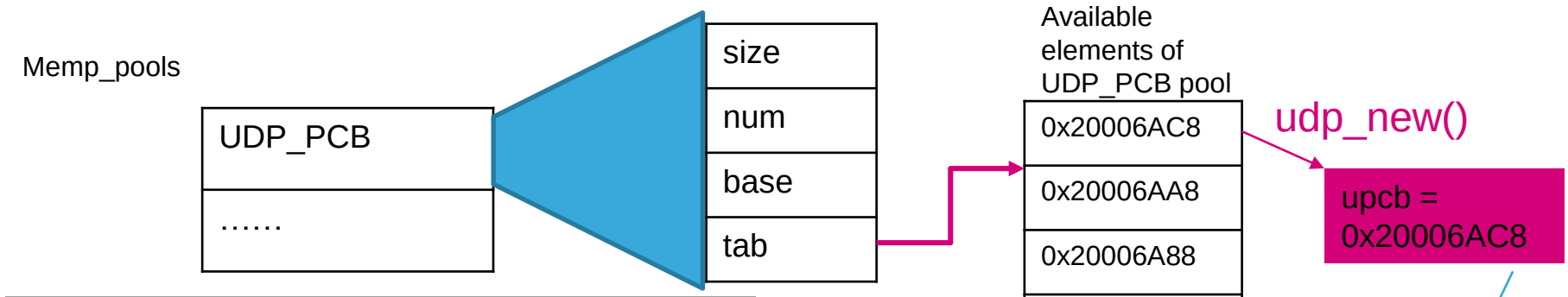
Memp_pools



Memp_std.h

Expression	Value	Location	Type
memp_pools	<array>	0x20000000	struct memp_desc const *[9]
[0]	memp_UDP_PCB (0x80052A8)	0x20000000	struct memp_desc const *
size	32	0x080052A8	u16_t
num	4	0x080052AA	u16_t
base	0x20006A68 ""	0x080052AC	u8_t*
tab	memp_tab_UDP_PCB (0x20006BCC)	0x080052B0	struct memp **
	0x20006AC8 (memp_memory_UD...	0x20006BCC	struct memp *
next	0x20006AA8 (memp_memory_UD...	0x20006AC8	struct memp *
next	0x20006A88 (memp_memory_UD...	0x20006AA8	struct memp *
next	memp_memory_UDP_PCB_base (...	0x20006A88	struct memp *
next	0x00000000	0x20006A68	struct memp *
[1]	memp_TCP_PCB (0x80052B4)	0x20000004	struct memp_desc const *
[2]	memp_TCP_PCB_LISTEN (0x800...	0x20000008	struct memp_desc const *
[3]	memp_TCP_SEG (0x80052CC)	0x2000000C	struct memp_desc const *
[4]	memp_REASSDATA (0x80052D8)	0x20000010	struct memp_desc const *
[5]	memp_FRAG_PBUF (0x80052E4)	0x20000014	struct memp_desc const *
[6]	memp_SYS_TIMEOUT (0x80052F0)	0x20000018	struct memp_desc const *
[7]	memp_PBUF (0x80052FC)	0x2000001C	struct memp_desc const *
[8]	memp_PBUF_POOL (0x8005308)	0x20000020	struct memp_desc const *
udp_pcbs	0x00000000	0x20000048	struct udp_pcb *
<click to edit>			

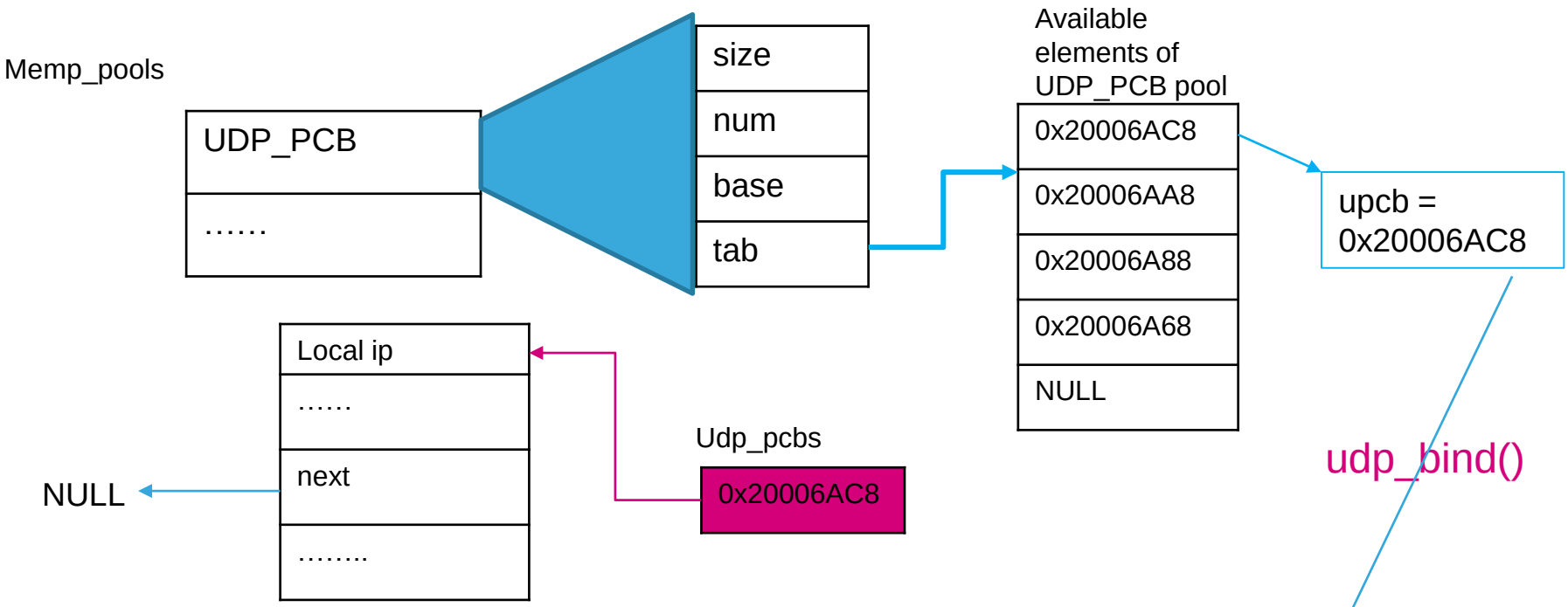
Build UDP Connection



Expression	Value	Location	Type
memp_pools	<array>	0x20000000	struct memp_desc const *[9]
[0]	memp_UDP_PCB (0x80052A8)	0x20000000	struct memp_desc const *
size	32	0x080052A8	u16_t
num	4	0x080052AA	u16_t
base	0x20006A68 ""	0x080052AC	u8_t*
tab	memp_tab_UDP_PCB (0x20006BCC)	0x080052B0	struct memp **
	0x20006AA8 (memp_memory_UD...	0x20006BCC	struct memp *
next	0x20006A88 (memp_memory_UD...	0x20006AA8	struct memp *
next	memp_memory_UDP_PCB_base (...	0x20006A88	struct memp *
next	0x00000000	0x20006A68	struct memp *
next	0x19CECDAE	0x00000000	struct memp *
[1]	memp_TCP_PCB (0x80052B4)	0x20000004	struct memp_desc const *
[2]	memp_TCP_PCB_LISTEN (0x800...	0x20000008	struct memp_desc const *
[3]	memp_TCP_SEG (0x80052CC)	0x2000000C	struct memp_desc const *
[4]	memp_REASSDATA (0x80052D8)	0x20000010	struct memp_desc const *
[5]	memp_FRAG_PBUF (0x80052E4)	0x20000014	struct memp_desc const *
[6]	memp_SYS_TIMEOUT (0x80052F0)	0x20000018	struct memp_desc const *
[7]	memp_PBUF (0x80052FC)	0x2000001C	struct memp_desc const *
[8]	memp_PBUF_POOL (0x8005308)	0x20000020	struct memp_desc const *
udp_pcbs	0x00000000	0x20000048	struct udp_pcb *
pcb	0x20006AC8 (memp_memory_UD...	R0	struct udp_pcb *

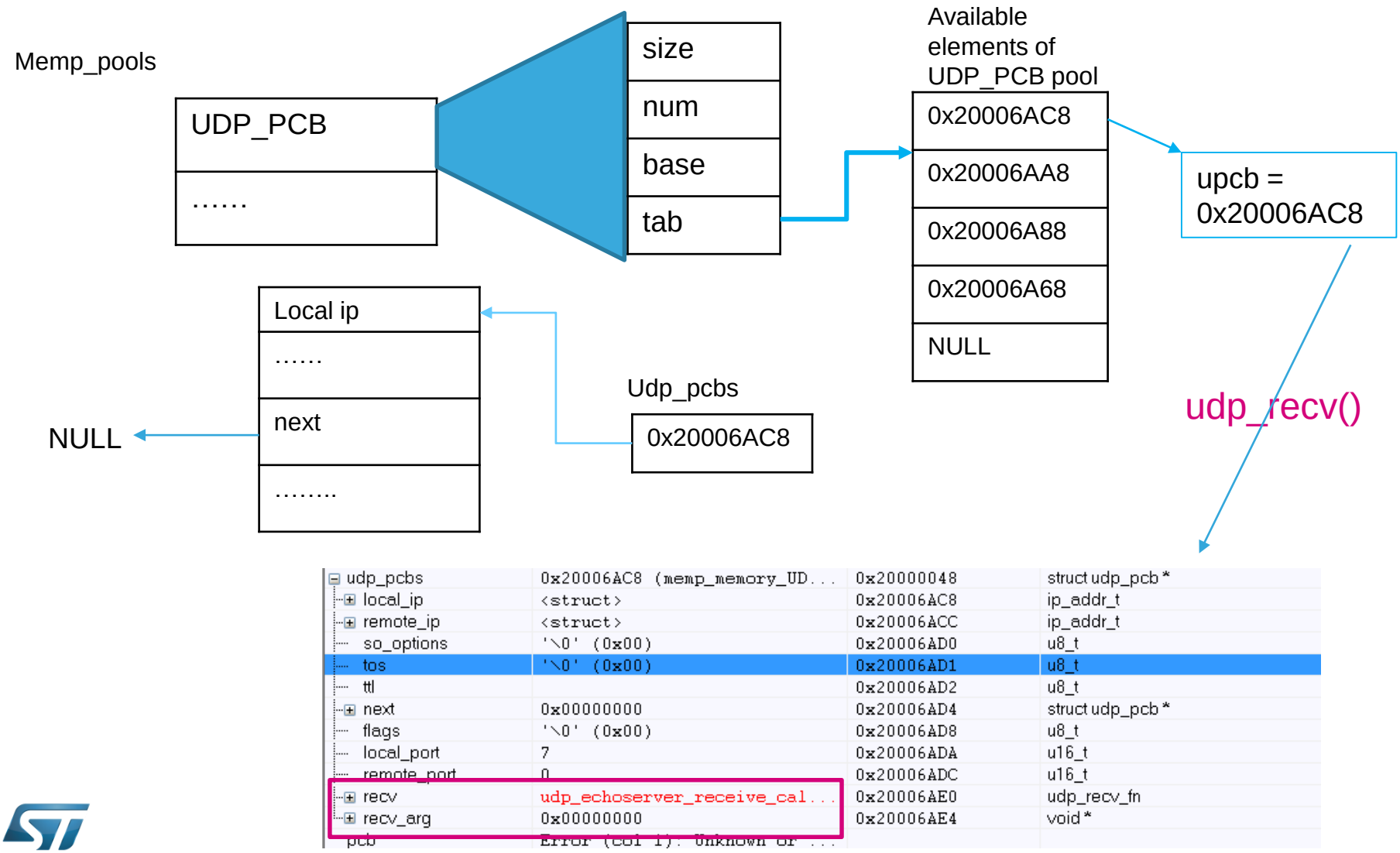
upcb	0x20006AC8 (memp_memory_UD...	R4	struct udp_pcb *
local_ip	<struct>	0x20006AC8	ip_addr_t
remote_ip	<struct>	0x20006ACC	ip_addr_t
so_options	'\0' (0x00)	0x20006AD0	u8_t
tos	'\0' (0x00)	0x20006AD1	u8_t
ttl		0x20006AD2	u8_t
next	0x00000000	0x20006AD4	struct udp_pcb *
flags	'\0' (0x00)	0x20006AD8	u8_t
local_port	0	0x20006ADA	u16_t
remote_port	0	0x20006ADC	u16_t
recv	HAL_ETH_ConfigDMA (0x0)	0x20006AE0	udp_recv_fn
recv_arg	0x00000000	0x20006AE4	void *

Build UDP Connection



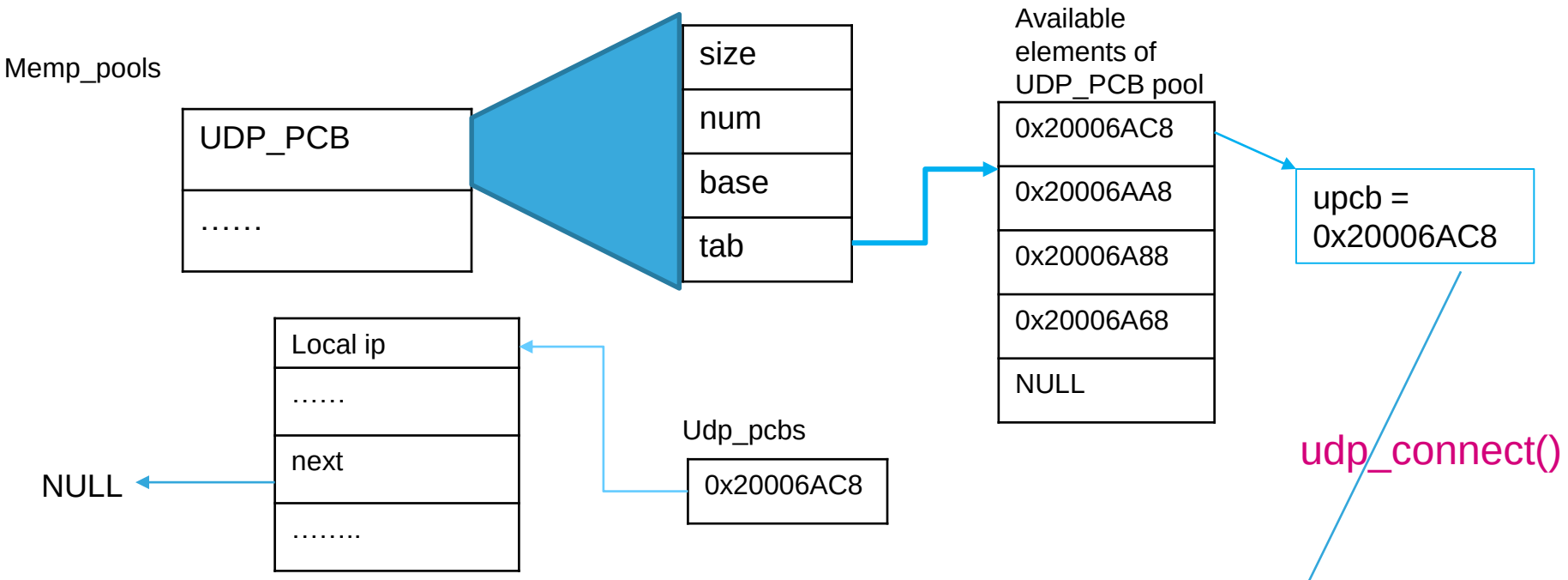
udp_pcbs	0x20006AC8 (memp_memory_UD...	0x20000048	struct udp_pcb *
upcb	0x20006AC8 (memp_memory_UD...	R4	struct udp_pcb *
local_ip	<struct>	0x20006AC8	ip_addr_t
remote_ip	<struct>	0x20006ACC	ip_addr_t
so_options	'\0' (0x00)	0x20006AD0	u8_t
tos	'\0' (0x00)	0x20006AD1	u8_t
ttl		0x20006AD2	u8_t
next	0x00000000	0x20006AD4	struct udp_pcb *
flags	'\0' (0x00)	0x20006AD8	u8_t
local_port	7	0x20006ADA	u16_t
remote_port	0	0x20006ADC	u16_t
recv	HAL_ETH_ConfigDMA (0x0)	0x20006AE0	udp_recv_fn
recv_arg	0x00000000	0x20006AE4	void *
<click to edit>			

Build UDP Connection



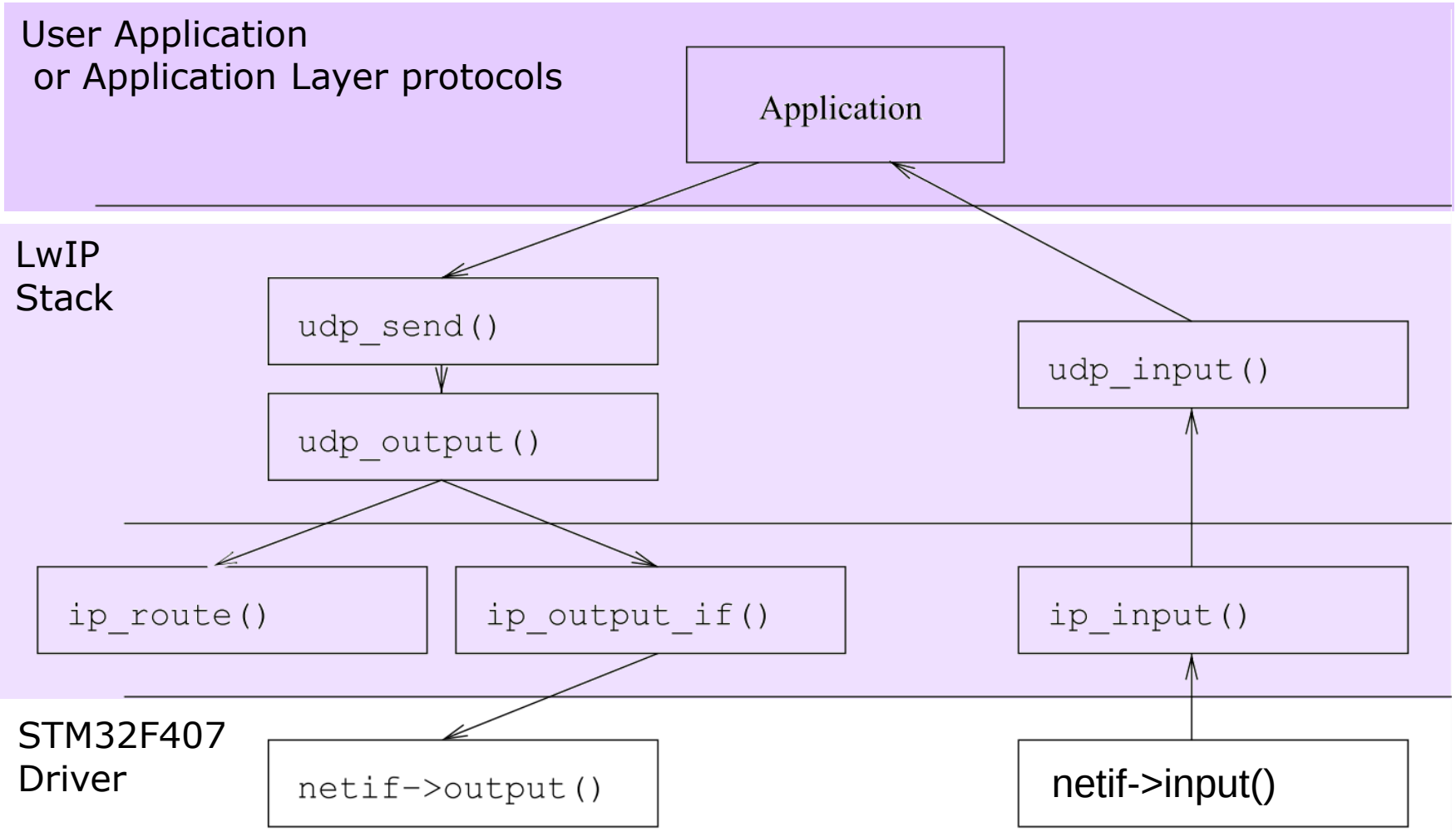
udp_pcbs	0x20006AC8 (memp_memory_UD...	0x20000048	struct udp_pcb *
local_ip	<struct>	0x20006AC8	ip_addr_t
remote_ip	<struct>	0x20006ACC	ip_addr_t
so_options	'\0' (0x00)	0x20006AD0	u8_t
tos	'\0' (0x00)	0x20006AD1	u8_t
ttl		0x20006AD2	u8_t
next	0x00000000	0x20006AD4	struct udp_pcb *
flags	'\0' (0x00)	0x20006AD8	u8_t
local_port	7	0x20006ADA	u16_t
remote_port	0	0x20006ADC	u16_t
recv	udp_echoserver_receive_cal...	0x20006AE0	udp_recv_fn
recv_arg	0x00000000	0x20006AE4	void *
pcb	Error (col 1): Unknown or ...		

Build UDP Connection

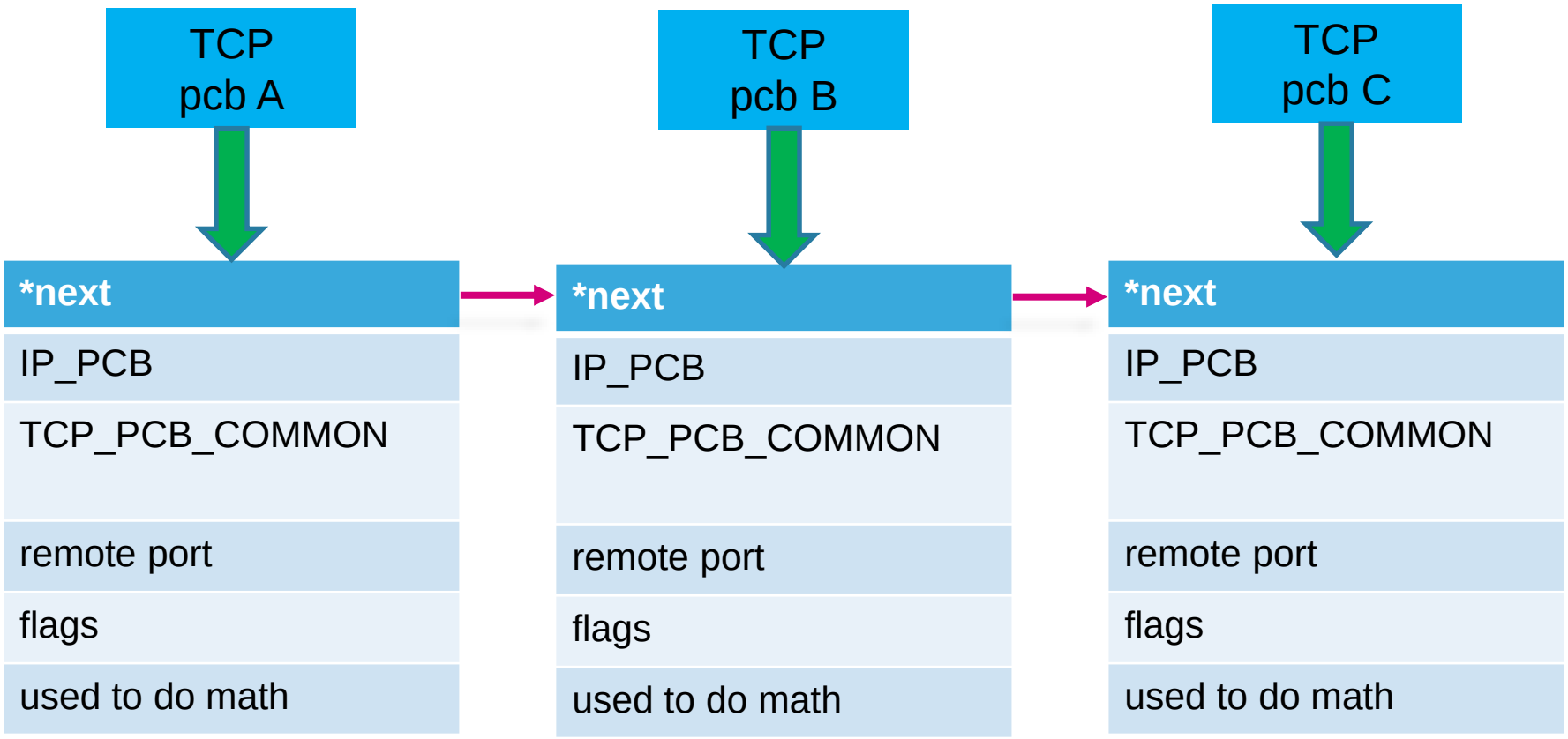


upcb	0x20006AC8 (memp_memory_UD...	R4	struct udp_pcb *
local_ip	<struct>	0x20006AC8	ip_addr_t
addr	0	0x20006AC8	u32_t
remote_ip	<struct>	0x20006ACC	ip_addr_t
addr	184592576	0x20006ACC	u32_t
so_options	'\0' (0x00)	0x20006AD0	u8_t
tos	'\0' (0x00)	0x20006AD1	u8_t
ttl		0x20006AD2	u8_t
next	0x00000000	0x20006AD4	struct udp_pcb *
flags	'.' (0x04)	0x20006AD8	u8_t
local_port	7	0x20006ADA	u16_t
remote_port	50032	0x20006ADC	u16_t
recv	udp_echoserver_receive_cal...	0x20006AE0	udp_recv_fn
recv_arg	0x00000000	0x20006AE4	void *
<click to edit>			

LwIP UDP connection flow



TCP pcb



The single-chained lists

TCP Raw API : connection functions

… tcp_new(void)

Create a TCP Structure

… tcp_bind(…)

Initialize the TCP Structure with the device **internal** IP address and port

… tcp_listen(…)

… tcp_accept(…)

These functions :
- sets up the local port to listen for incoming connections
- initialize the callback function to use when a new connection occurs.

… tcp_connect(…)

Connect to the remote host and sends the initial SYN segment which opens the connection.

LwIP Server setup

lwIP Stack Action	Application actions
Create TCP PCB	tcp_new() 1
Bind port number	tcp_bind() 2
Create listening endpoint (new PCB allocated)	tcp_listen() 3
Set accept callback	tcp_accept() 4

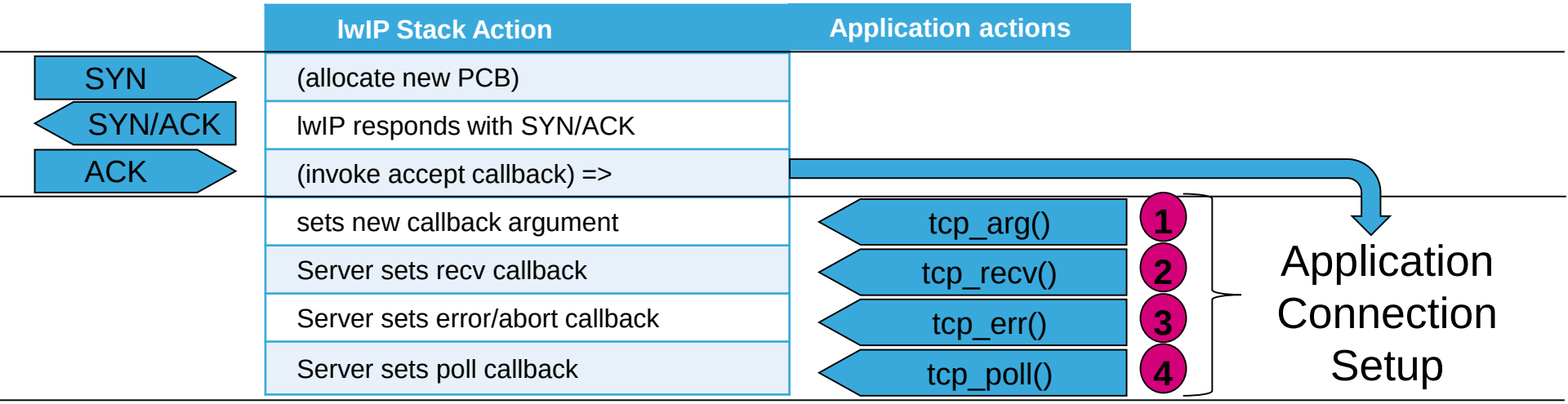
Application Listener Setup (server)

```
void tcp_echoserver_init(void)
{
  /* create new tcp pcb */
  tcp_echoserver_pcb = tcp_new();
  if (tcp_echoserver_pcb != NULL)
  {
    err_t err;
    /* bind echo_pcb to port 7 (ECHO protocol) */
    err = tcp_bind(tcp_echoserver_pcb, IP_ADDR_ANY, 7);
    if (err == ERR_OK)
    {
      /* start tcp listening for echo_pcb */
      tcp_echoserver_pcb = tcp_listen(tcp_echoserver_pcb);
      /* initialize LwIP tcp_accept callback function */
      tcp_accept(tcp_echoserver_pcb, tcp_echoserver_accept);
    }
    else
    {
      /* deallocate the pcb */
      memp_free(MEMP_TCP_PCB, tcp_echoserver_pcb);
    }
  }
```

1 2 3 4

Now the application is ready to handle incoming connections

LwIP Client incoming(1)



Now the application is ready to send/receive data to/from the client

LwIP Client incoming(2)


```
static err_t tcp_echoserver_accept(void *arg, struct tcp_pcb *newpcb, err_t err)
{
  ...
  /* allocate structure es to maintain tcp connection informations */
  es = (struct tcp_echoserver_struct *)mem_malloc(sizeof(struct tcp_echoserver_struct));
  if (es != NULL)
  {
    es->state = ES_ACCEPTED;
    es->pcb = newpcb;
    es->retries = 0;
    es->p = NULL;

    /* pass newly allocated es structure as argument to newpcb */
1   tcp_arg(newpcb, es);

    /* initialize lwip tcp_recv callback function for newpcb  */
2   tcp_recv(newpcb, tcp_echoserver_recv);

    /* initialize lwip tcp_err callback function for newpcb  */
3   tcp_err(newpcb, tcp_echoserver_error);
    /* initialize lwip tcp_poll callback function for newpcb */
4   tcp_poll(newpcb, tcp_echoserver_poll, 0);
    ret_err = ERR_OK;
  }
  else
  {
   ...
  }
  return ret_err;
}
```

TCP Raw API : Receive & send functions

… tcp_recv(…)

… tcp_recved(…)

Callback function

These functions are used to receive data :
- Specify a callback function
- inform LwIP that the data was handled

… tcp_sent(…)

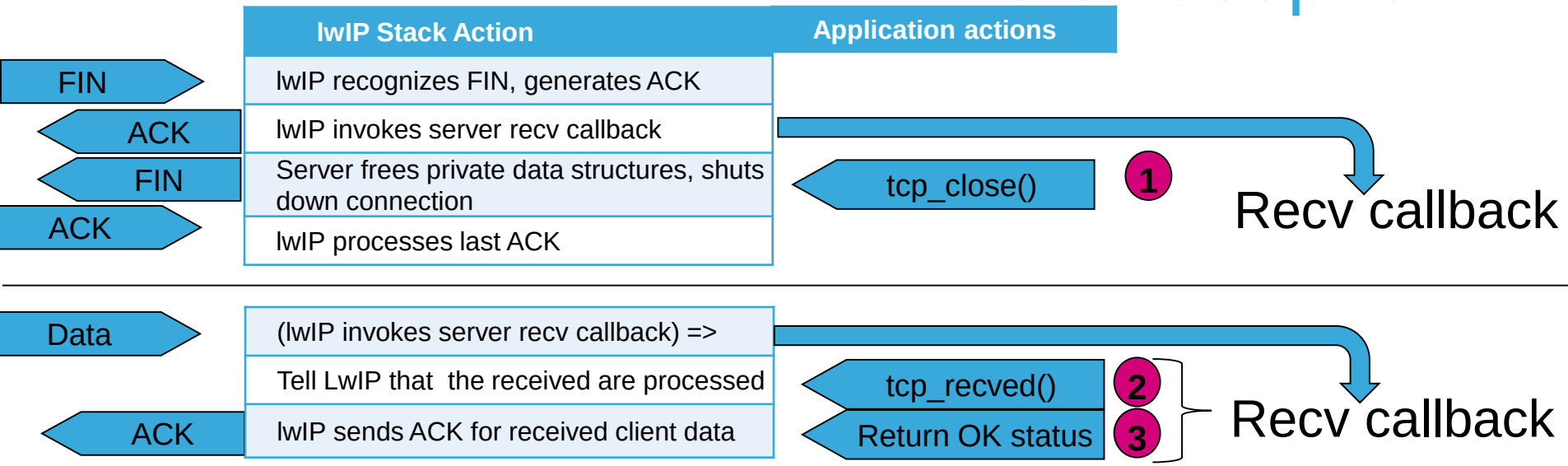
… tcp_sndbuf(…)

… tcp_write(…)

… tcp_output(…)

These functions are used to send out data :
- Specify a callback function
- Get the maximum amount of data that can be sent.
- to enqueue the data.
- to force the data to be sent.

LwIP Data reception



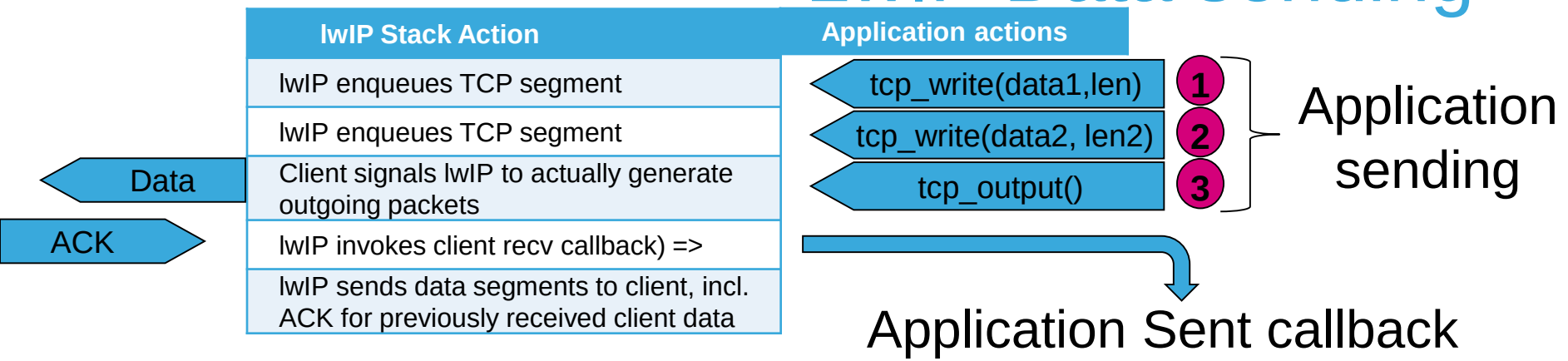

```
err_t Recv_callback(void *arg, struct tcp_pcb *tpcb,
    struct pbuf *pcallback, err_t err)
{…

 if(pcallback == NULL) {
1 tcp_close(tpcb);
  return ERR_OK;
 }
 Else{
2 tcp_recved(tpcb, pcallback->tot_len);
3 return ERR_OK;
 }
}
```

pcallback = NULL
when receiving a FIN

The receive callback
send the ACK on exit

life.augmented

LwIP Data sending



Example with Recopy

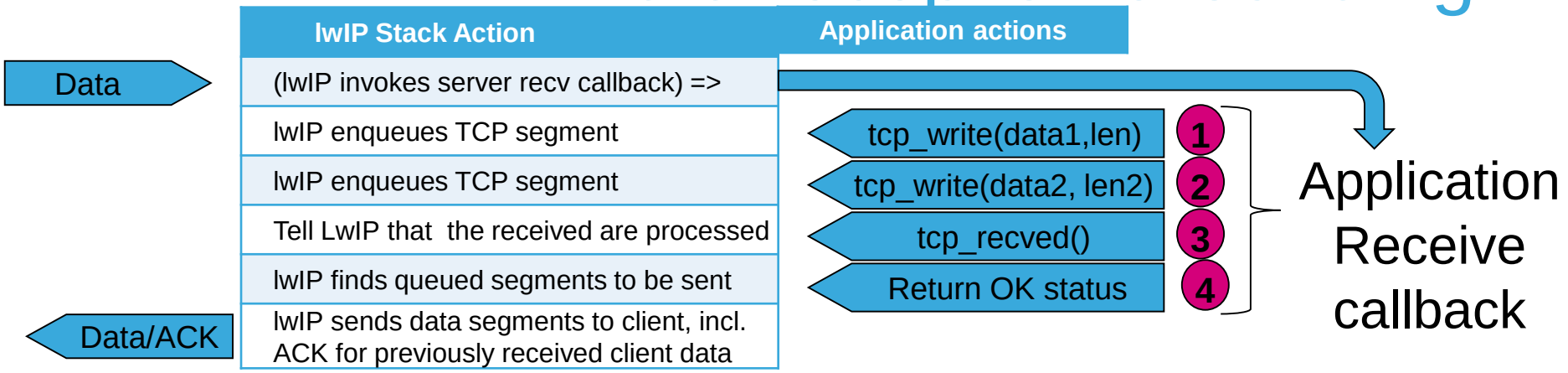
1
```
Error = tcp_write(tpcb, "LED2", 4, TCP_WRITE_FLAG_COPY|TCP_WRITE_FLAG_MORE);
if(Error != ERR_OK) return Error;
```

2
```
Error = tcp_write(tpcb, " TOGGLED", 8, TCP_WRITE_FLAG_COPY);
 if(Error != ERR_OK) return Error;
```

3
```
 Error = tcp_output(tpcb);
 if(Error != ERR_OK) return Error;
```

The application is informed of the data reception By the sent callback

LwIP Data reception & sending




```
err_t Recv_callback(void *arg, struct tcp_pcb *tpcb, struct pbuf *pcallback, err_t err)
{…
 if(pcallback == NULL)
 {
          …
 }
 Else{
  Error = tcp_write(tpcb, "LED2", 4, TCP_WRITE_FLAG_COPY|TCP_WRITE_FLAG_MORE);
  if(Error != ERR_OK) return Error;

  Error = tcp_write(tpcb, " TOGGLED", 8, TCP_WRITE_FLAG_COPY);
  if(Error != ERR_OK) return Error;

  tcp_recved(tpcb, pcallback->tot_len);
  return ERR_OK;
 }
}
```

1
2
3
4

reception of a FIN

No need to call tcp_ouput within the receive callback
It is done on exit

RAW API & BSD Sockets

Client Process Activity	BSD Sockets Call Used	lwIP RAW API Call Used	Server Process Activity	BSD Sockets Call Used	lwIP RAW API Call Used
create a socket	socket()	tcp_new()	create a socket	socket()	tcp_new()
bind a socket address(optional)	bind()	tcp_bind()	bind a socket address	bind()	tcp_bind()
			listen for incoming connections	listen()	tcp_listen()
request a connection	connect()	tcp_connect()			
			accept connection	accept()	tcp_accept()
send data	write() send()	tcp_write() tcp_sent()			
			receive data	read() recv()	tcp_recv()
			send data	write() send()	tcp_write() tcp_sent()
receive data	read() recv()	tcp_recv()			
Disconnect socket (optional)	shutdown() close()	tcp_abort() tcp_close()	Disconnect socket (optional)	shutdown() close()	tcp_abort() tcp_close()

Resource

AN3966	LwIP TCP/IP stack demonstration for STM32F4x7xx microcontrollers
AN3384	LwIP TCP/IP stack demonstration for STM32F2x7xx microcontrollers
AN3102	LwIP TCP/IP stack demonstration for STM32F107xx connectivity line microcontrollers
AN3968	STM32F407/STM32F417 in-application programming (IAP) over Ethernet
AN3226	STM32F107 In-Application Programming (IAP) over Ethernet
AN3376	STM32F2x7 In-Application Programming (IAP) over Ethernet
UM1709	STM32Cube Ethernet IAP example
UM1713	Developing applications on STM32Cube with LwIP TCP/IP stack

Ethernet / TCP-IP - Training Suite

04 – TCP/IP solution & Tools

life.augmented

TCP/IP solutions (1/3)

Provider	Solution name	Model	Cost	Availability			
				F107	F2	F4	F7
CMX	CMX-TCP/IP, CMX-MicroNet, CMX-Inet-Plus	Source	License	Y	Y	Y	Y
Cypherbridge	uSSH	Source	License	N	Y	Y	Y
EUROS	TCP/IP stack	Binaries	License	N	Y	Y	Y
Express Logic	NetX and NetX Duo IPv4/IPv6	Source	License	Y	Y	Y	Y
eCosCentric	SecureSockets, SecureShell eCosPro stacks	Source	License	Y	Y	Y	Y
eForce	µNet3	Source	License	Y	Y	Y	Y
EmCraft	Linux TCP/IP stack	Open source (GPL)	Free	N	Y	Y	Y
GreenHills	µ-velOSity TCP/IP v4/v6	Source	License	Y	Y	Y	Y
HCC	MISRA HCC-TCP/IP v4/v6	Source	License	Y	Y	Y	Y
Interniche	NicheStack	Source	License	Y	Y	Y	Y
Interniche	embTCP v4/v6	Binaries	License	N	Y	Y	Y
Keil/ARM	MDK-ARM TCPNET	Source	License	Y	Y	Y	Y
SICS	LwIP	Open source (BSD)	Free	Y[2]	Y[2]	Y[2]	Y[3]

TCP/IP solutions (2/3)

Provider	Solution name	Model	Cost	Availability			
				F107	F2	F4	F7
Mentor Embedded	Nucleus Network	Source	License	Y	Y	Y	Y
Micrium	µC/TCP-IP	Source	License	Y	Y	Y	Y
Micro Digital	smxNS and smxNS6 (Dual IPv6/v4)	Source	License	Y	Y	Y	Y
Oryx Emb.	CycloneTCP	Open source (GPL2) or source	Free or license	Y	Y	Y	Y
Quadros	RTXC Quadnet	Source	License	Y	Y	Y	Y
Rowebots	Unison TCP-IP/v4-v6	Source	License	Y	Y	Y	Y
SEGGER	embOS/IP	Source	License	Y	Y	Y	Y
ST	STM32Cube - LwIP	Open source (BSD)	Free	Q1/15	Y	Y	Q2/15

TCP/IP solutions (3/3)

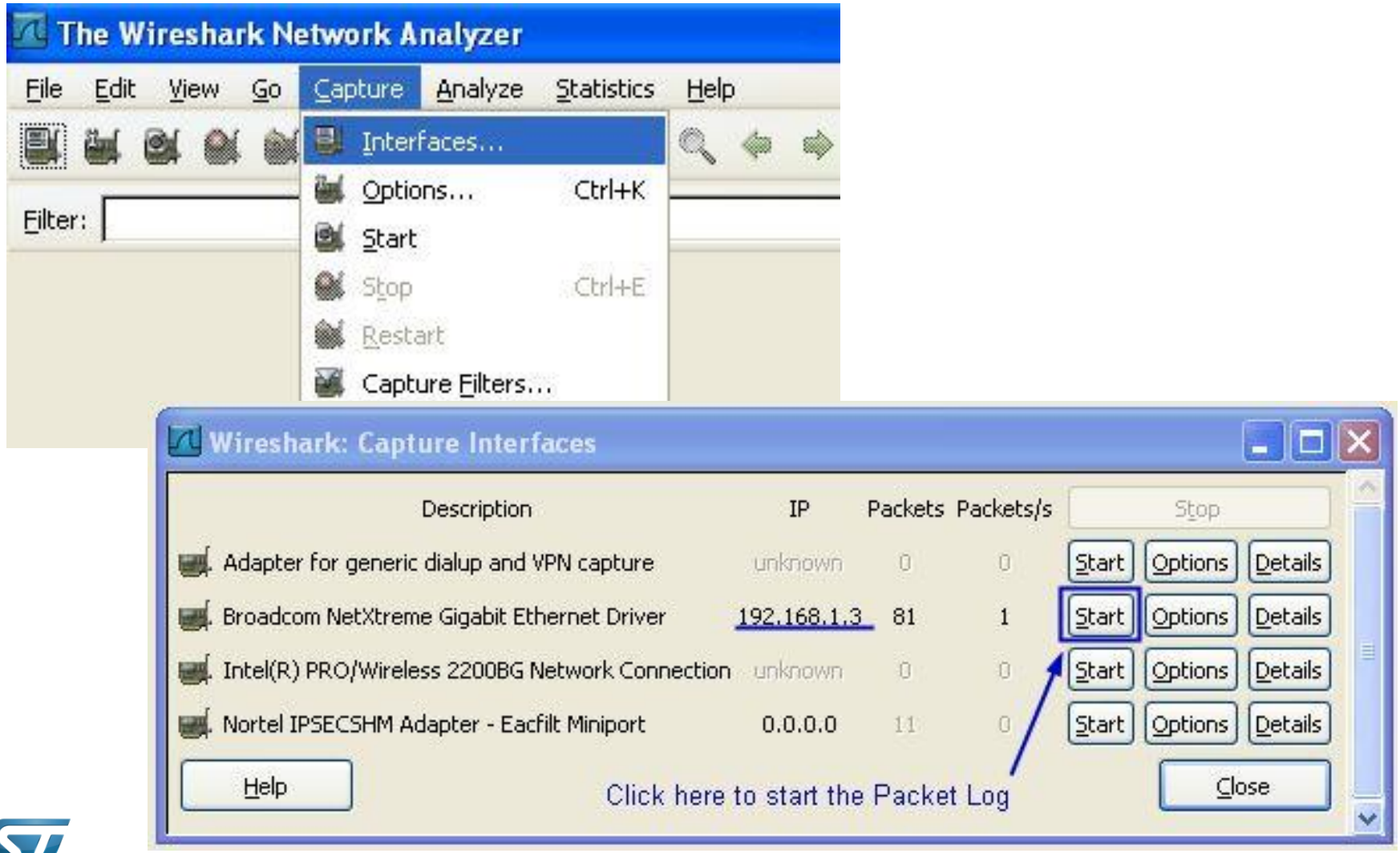
Provider	Solution name	Model	Cost	Availability			
				F107	F2	F4	F7
CypherBridge	uSSL/TLS	Source	License	N	Y	Y	Y
HCC	Verifiable SSL/TLS	Source	License	Y	Y	Y	Y
Oryx Emb.	CycloneSSL	Open source (GPL2) or Source	Free or license	Y	Y	Y	Y
PolarSSL	PolarSSL	Open source (GPL2) or Source	Free or license	Y[1]	Y[1]	Y[1]	Y[2]
ST	STM32Cube - PolarSSL	Open source (GPL2) or Source	Free or license	Q1 15	Y	Y	Q2 15
wolfSSL	CyaSSL	Open source (GPL2) or Source	Free or license	N	Y	Y	Y
SEGGER	emSSL	Source	License	Y	Y	Y	Y

Useful tools: WireShark

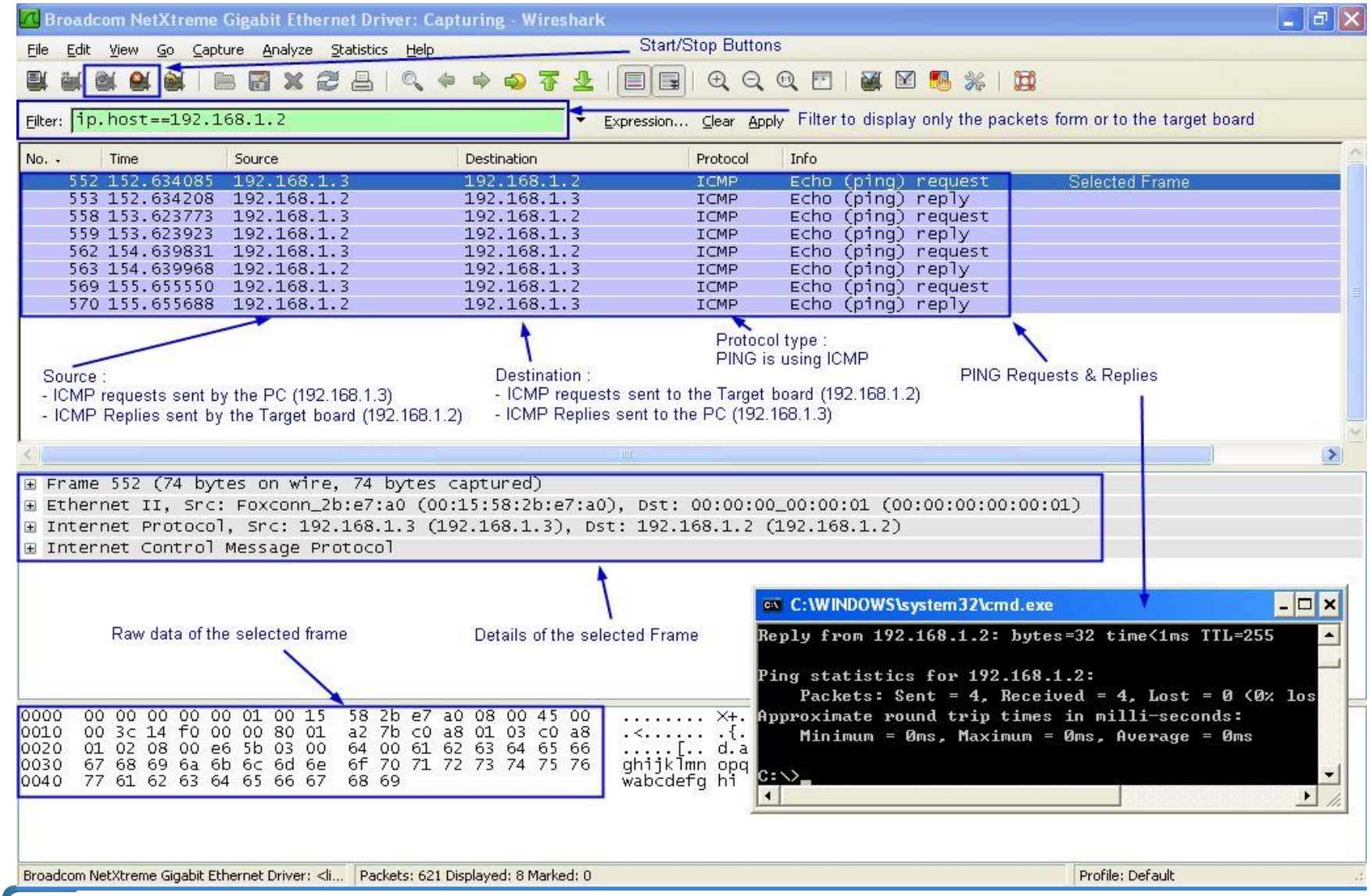
- WireShark is
  - a network monitoring tool
  - It uses WinPcap that interfaces directly with the Network card
- Wireshark allows you to
  - See all the packets sent or received by the PC
  - Filter the packets to display only the relevant information.
  - The packets content is formatted for easy reading
  - This Software cannot send any data

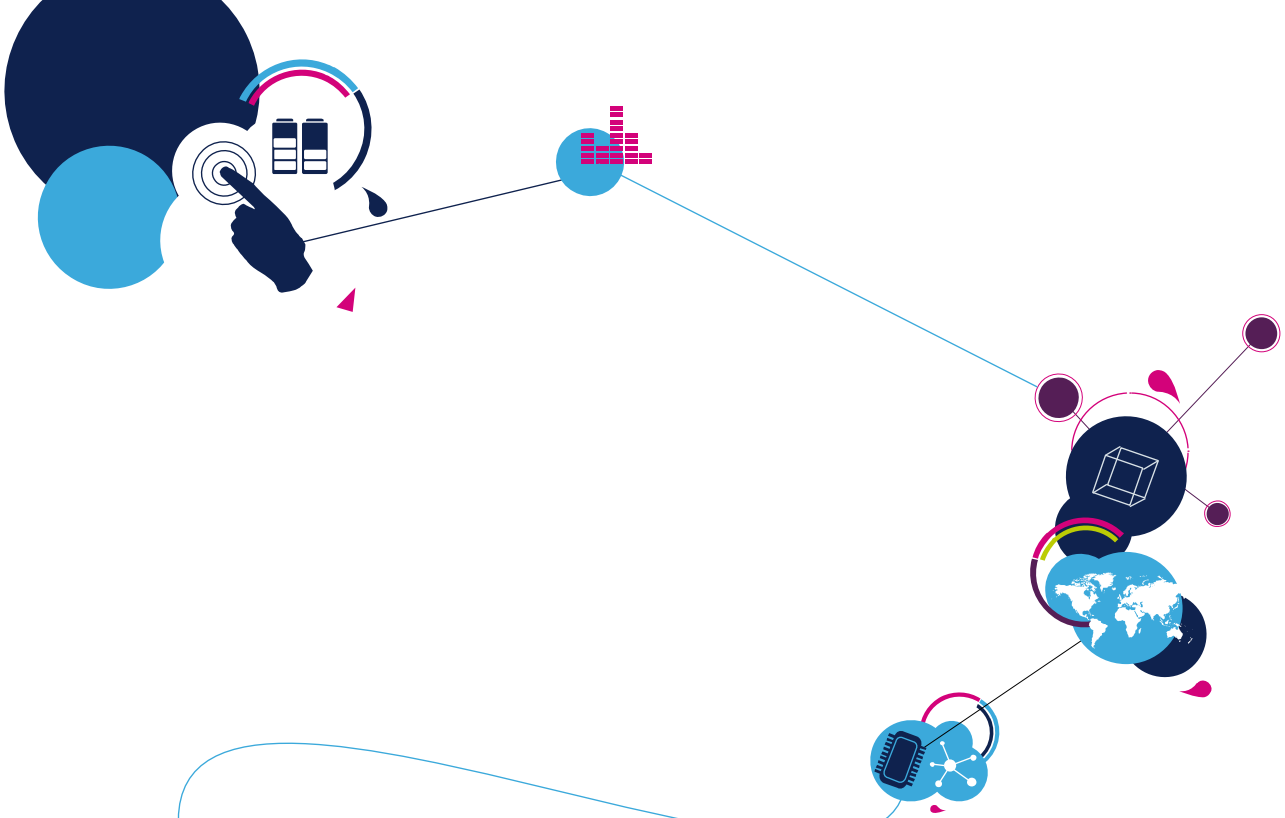
WireShark : how to use it

- Select the network interface you want to monitor



WireShark : ICMP Echo Requests & Replies





Thank you !

life.augmented