

CSE 333

Section 8

Client-side Networking

Logistics

Due Today:

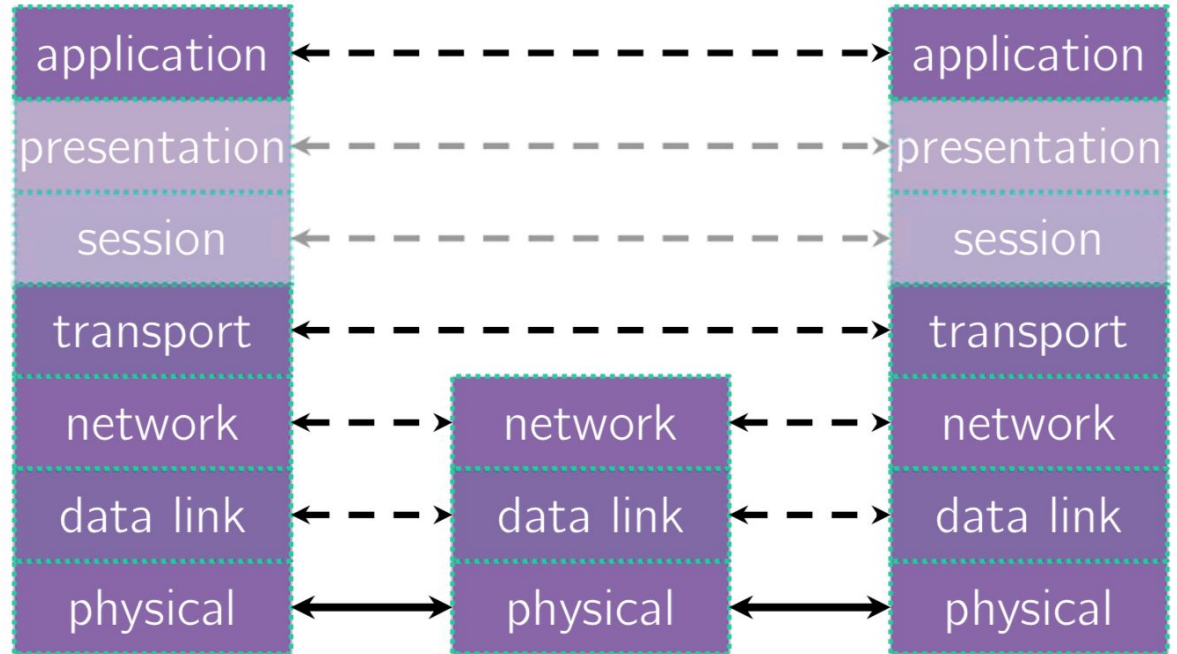
Homework 3 @ 11:59 pm

Due Monday:

Exercise 10 @ 11 am

Networking - At a High Level

Computer Networks: A 7-ish Layer Cake



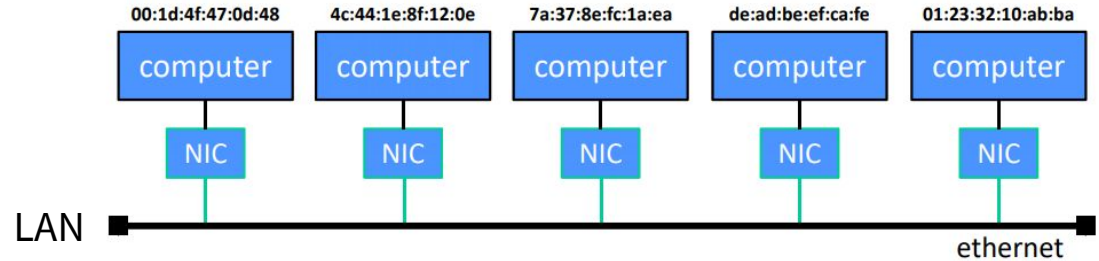
Computer Networks: A 7-ish Layer Cake



bit encoding at signal level

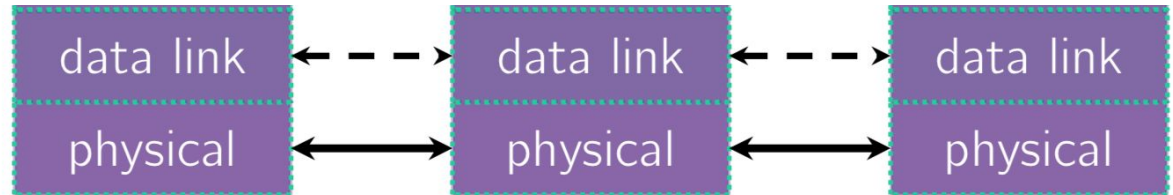


Computer Networks: A 7-ish Layer Cake

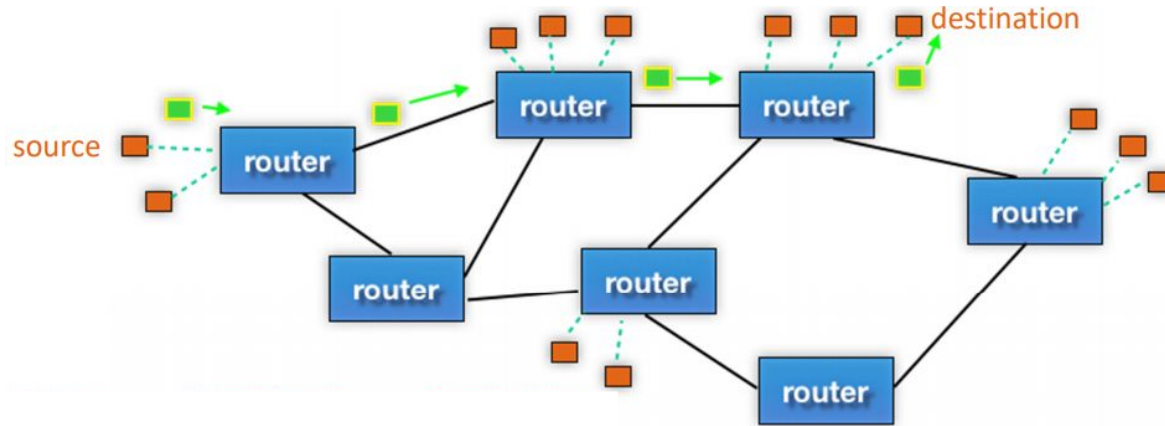


multiple computers on a local network

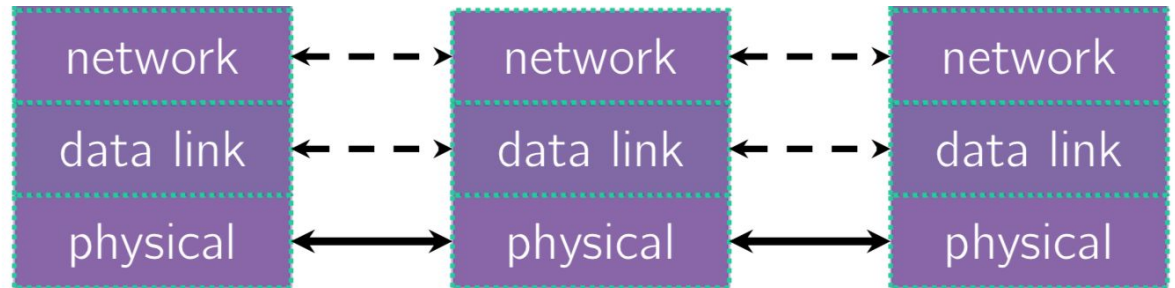
bit encoding at signal level



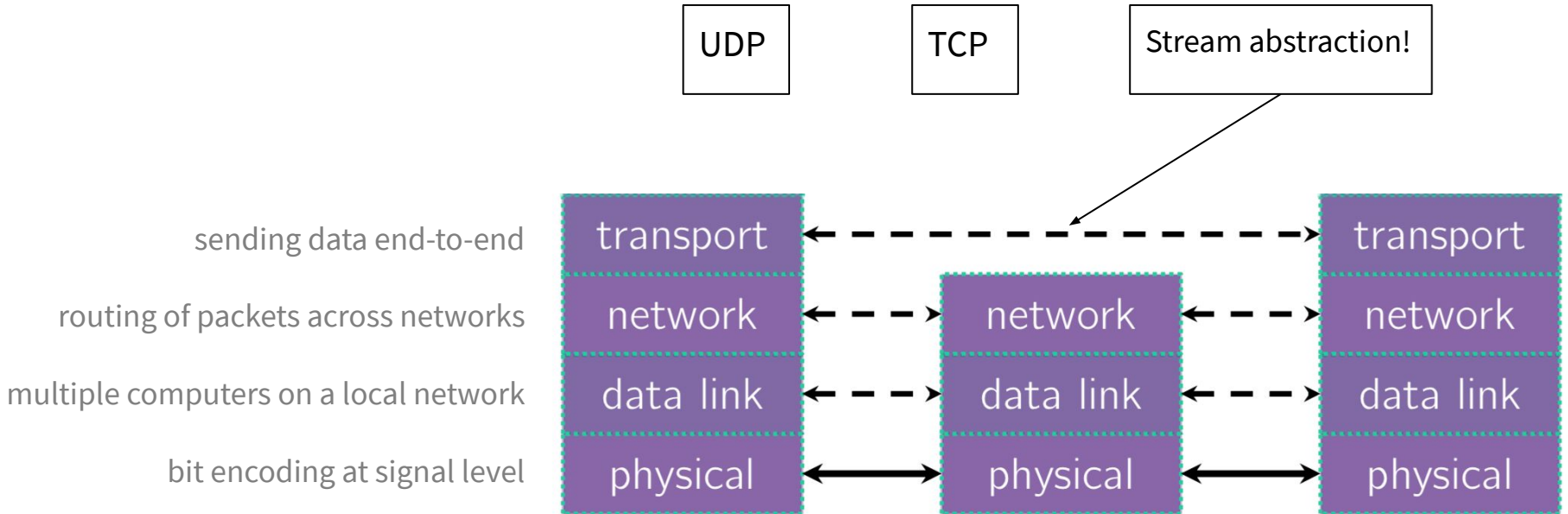
Computer Networks: A 7-ish Layer Cake



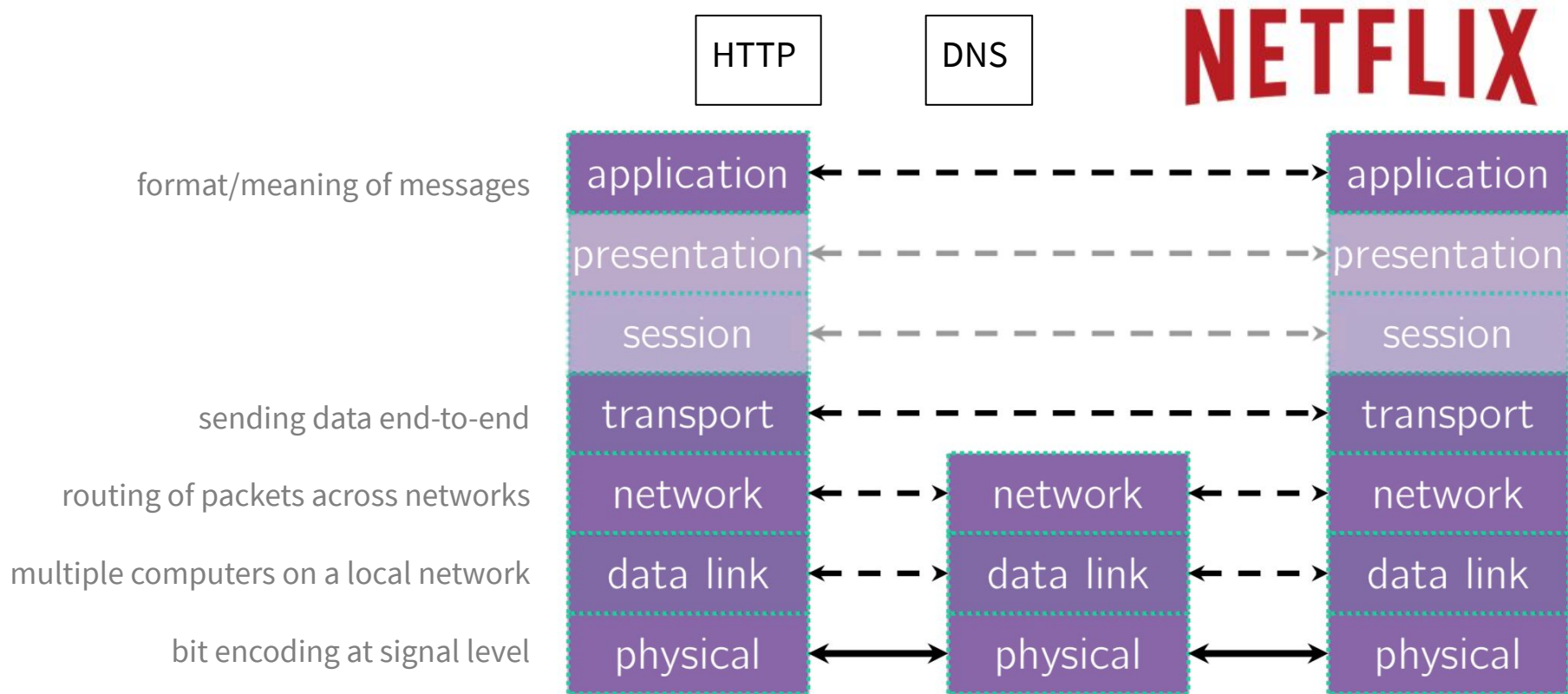
- routing of packets across networks
- multiple computers on a local network
- bit encoding at signal level



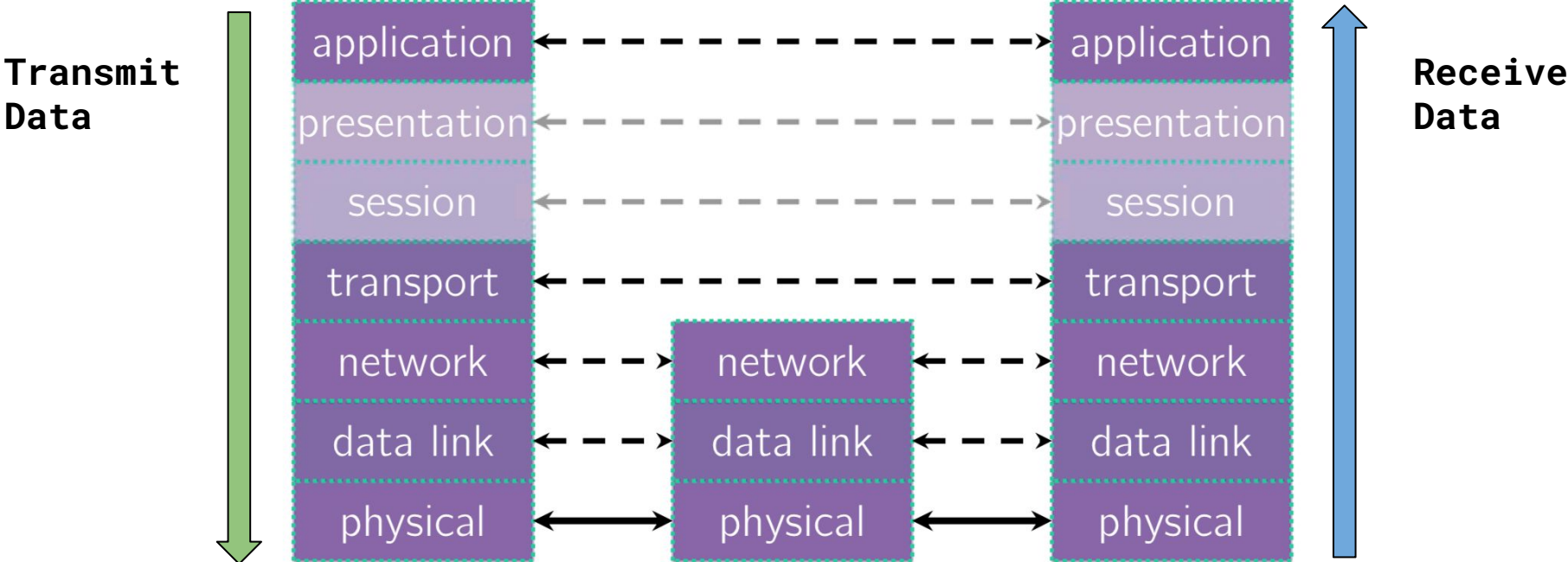
Computer Networks: A 7-ish Layer Cake



Computer Networks: A 7-ish Layer Cake



Data flow

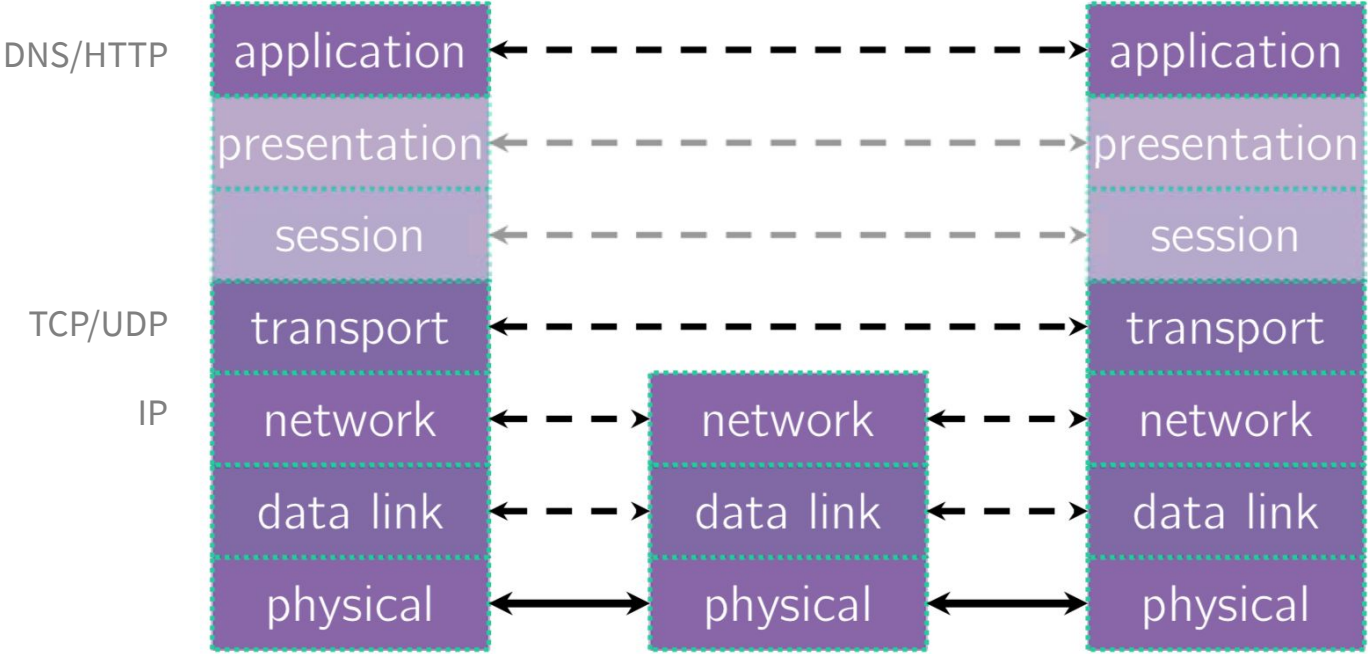


Exercise 1

Exercise 1

- DNS: Translating between IP addresses and host names. (Application Layer)
- IP: Routing packets across the Internet. (Network Layer)
- TCP: Reliable, stream-based networking on top of IP. (Transport Layer)
- UDP: Unreliable, packet-based networking on top of IP. (Transport Layer)
- HTTP: Sending websites and data over the Internet. (Application Layer)

Exercise 1



TCP versus UDP

Transmission Control Protocol(TCP)

- Connection oriented Service
- Reliable and Ordered
- Flow control

User Datagram Protocol(UDP)

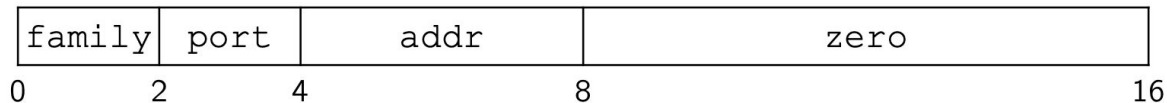
- Connectionless service
- Unreliable packet delivery
- Faster
- No feedback

Client-side Networking

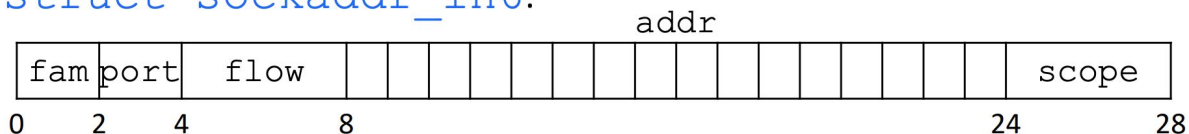
Sockets

- Just a file descriptor for network communication
- Types of Sockets
 - Stream sockets (TCP)
 - Datagram sockets (UDP)
- Each socket is associated with **a port number** and **an IP address**
 - Both port and address are stored in network byte order (big endian)

```
struct sockaddr_in:
```



```
struct sockaddr_in6:
```



Sockets

`struct sockaddr` (pointer to this struct is used as parameter type in system calls)

fam	????
-----	------

....

`struct sockaddr_in` (IPv4)

fam	port	addr	zero
-----	------	------	------

16

`struct sockaddr_in6` (IPv6)

fam	port	flow	addr	scope
-----	------	------	------	-------

28

`struct sockaddr_storage`

fam	
-----	--

Big enough to hold either

Byte Ordering and Endianness

- **Network Byte Order (Big Endian)**
 - The most significant byte is stored in the highest address
- **Host byte order**
 - Might be big or little endian, depending on the hardware
- **To convert between orderings, we can use**
 - `uint16_t htons (uint16_t hostlong);`
 - `uint16_t ntohs (uint16_t hostlong);`

 - `uint32_t htonl (uint32_t hostlong);`
 - `uint32_t ntohl (uint32_t hostlong);`

Networking methods

```
// Figure out what IP address and port to talk to
// returns 0 on success, negative number on failure
int getaddrinfo(const char *hostname,      // hostname to lookup
               const char *servname,     // service name
               const struct addrinfo *hints, // desired output (optional)
               struct addrinfo **res);    // results structure

// Frees memory allocated by getaddrinfo()
void freeaddrinfo(struct addrinfo *ai);
```

Networking methods

```
struct addrinfo {
    int ai_flags;           // additional flags
    int ai_family;         // AF_INET, AF_INET6, AF_UNSPEC
    int ai_socktype;       // SOCK_STREAM, SOCK_DGRAM, 0
    int ai_protocol;       // IPPROTO_TCP, IPPROTO_UDP, 0
    size_t ai_addrlen;     // length of socket addr in bytes
    struct sockaddr* ai_addr; // pointer to socket addr
    char* ai_canonname;     // canonical name
    struct addrinfo* ai_next; // can have linked list of records
}
```

- `ai_addr` points to a `struct sockaddr` describing a socket address, can be IPv4 or IPv6

Networking methods

```
// Creates a socket
// returns file descriptor on success, -1 on failure (errno set)
int socket(int domain,          // AF_INET, AF_INET6, etc.
           int type,           // SOCK_STREAM, SOCK_DGRAM, etc.
           int protocol);      // usually 0

// Connects to the server
// returns 0 on success, -1 on failure (errno set)
int connect(int sockfd,          // socket file descriptor
            struct sockaddr *serv_addr, // socket addr of server
            socklen_t addrlen);   // size of serv_addr
```

Networking methods

```
// returns amount read, 0 for EOF, -1 on failure (errno set)
ssize_t read(int fd, void *buf, size_t count);
```

```
// returns amount written, -1 on failure (errno set)
ssize_t write(int fd, void *buf, size_t count);
```

```
// returns 0 for success, -1 on failure (errno set)
int close(int fd);
```

- Same POSIX methods we used for file I/O!
(so they require the same error checking...)

Exercise 2

Input param ->

specify lookup hints

Dotted boxes should be filled in with the data type.

① _____ (hostname, servname, _____, _____)

<- Output param

extract fields from result (IPv4 vs IPv6)

② _____ (_____, type, protocol)

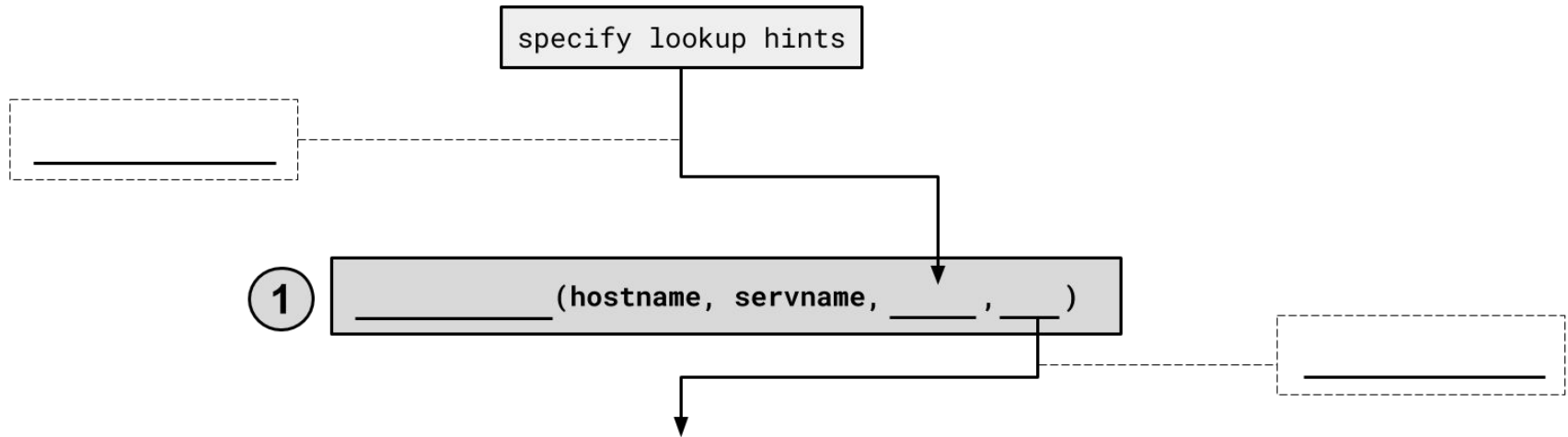
③ _____ (_____, _____, addrlen)

④ read(_____, buf, count)

write(_____, buf, count)

⑤ close(_____)

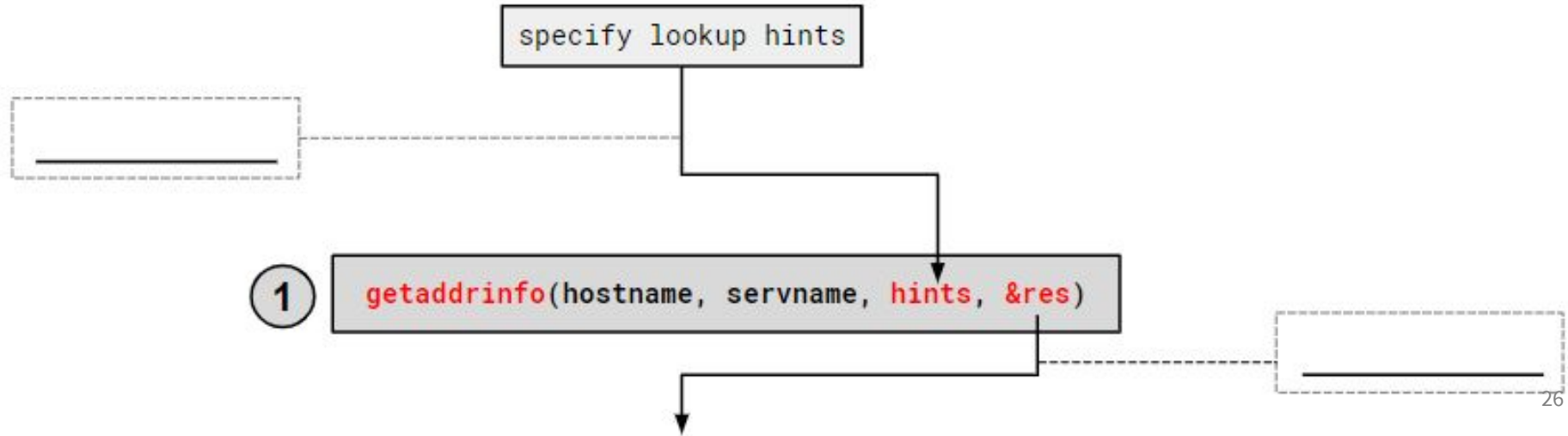
1.



1. getaddrinfo()

```
int getaddrinfo(const char *hostname,  
               const char *service,  
               const struct addrinfo *hints,  
               struct addrinfo **res);
```

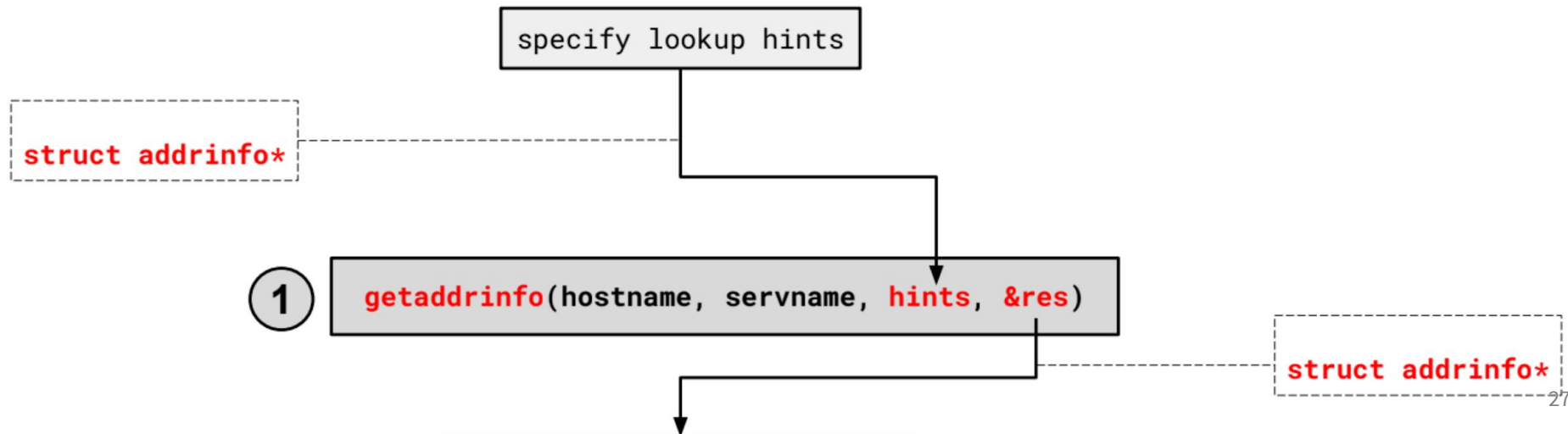
- Performs a **DNS Lookup** for a hostname



1. getaddrinfo()

```
int getaddrinfo(const char *hostname,  
               const char *service,  
               const struct addrinfo *hints,  
               struct addrinfo **res);
```

- Performs a **DNS Lookup** for a hostname
- Use “hints” to specify constraints (`struct addrinfo *`)
- Get back a linked list of `struct addrinfo` results



1. getaddrinfo() - Interpreting Results

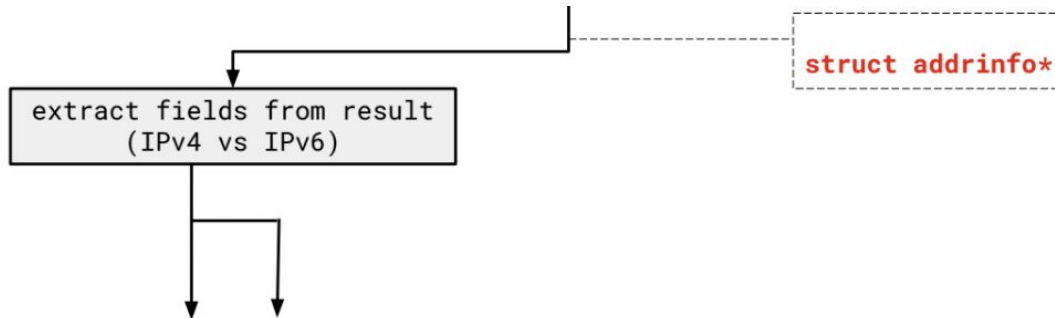
```
struct addrinfo {
    int ai_flags; // additional flags
    int ai_family; // AF_INET, AF_INET6, AF_UNSPEC
    int ai_socktype; // SOCK_STREAM, SOCK_DGRAM, 0
    int ai_protocol; // IPPROTO_TCP, IPPROTO_UDP, 0
    size_t ai_addrlen; // length of socket addr in bytes
    struct sockaddr* ai_addr; // pointer to socket addr
    char* ai_canonname; // canonical name
    struct addrinfo* ai_next; // can form a linked list
};
```

- ai_addr points to a struct sockaddr describing the socket address

1. getaddrinfo() - Interpreting Results

With a `struct sockaddr*`:

- The field `sa_family` describes if it is IPv4 or IPv6
- Cast to `struct sockaddr_in*` (v4) or `struct sockaddr_in6*` (v6) to access/modify specific fields
- Store results in a `struct sockaddr_storage` to have a space big enough for either



1. getaddrinfo() - Interpreting Results

`struct sockaddr` (pointer to this struct is used as parameter type in system calls)

fam	????
-----	------

.....

`struct sockaddr_in` (IPv4)

fam	port	addr	zero
-----	------	------	------

16

`struct sockaddr_in6` (IPv6)

fam	port	flow	addr	scope
-----	------	------	------	-------

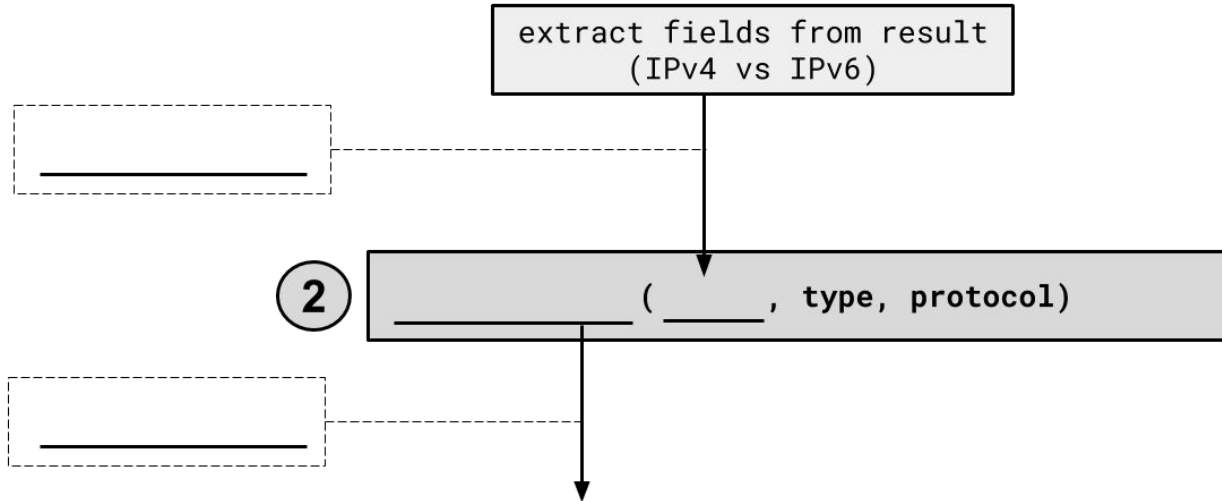
28

`struct sockaddr_storage`

fam	
-----	--

Big enough to hold either

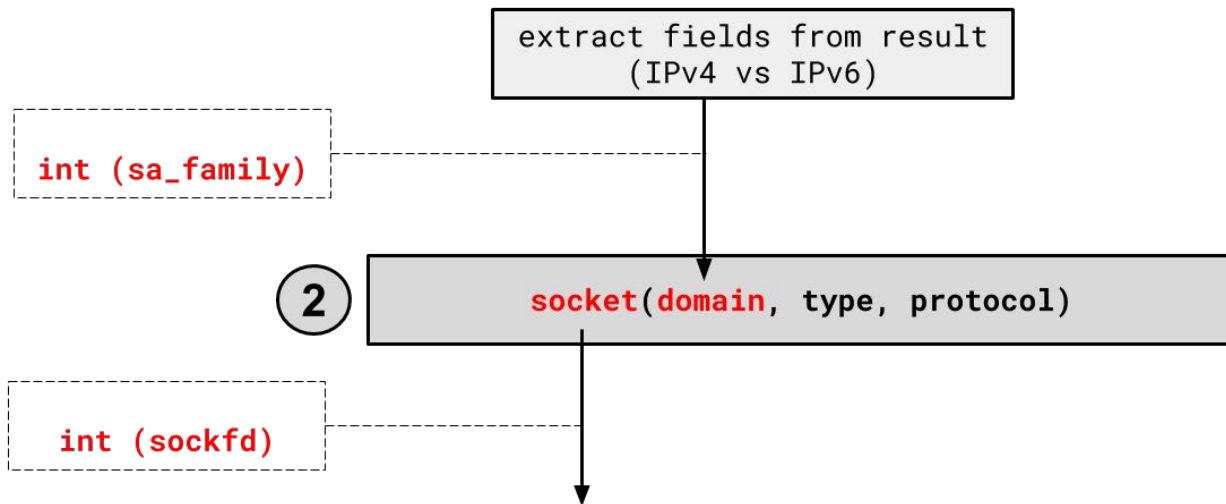
2.



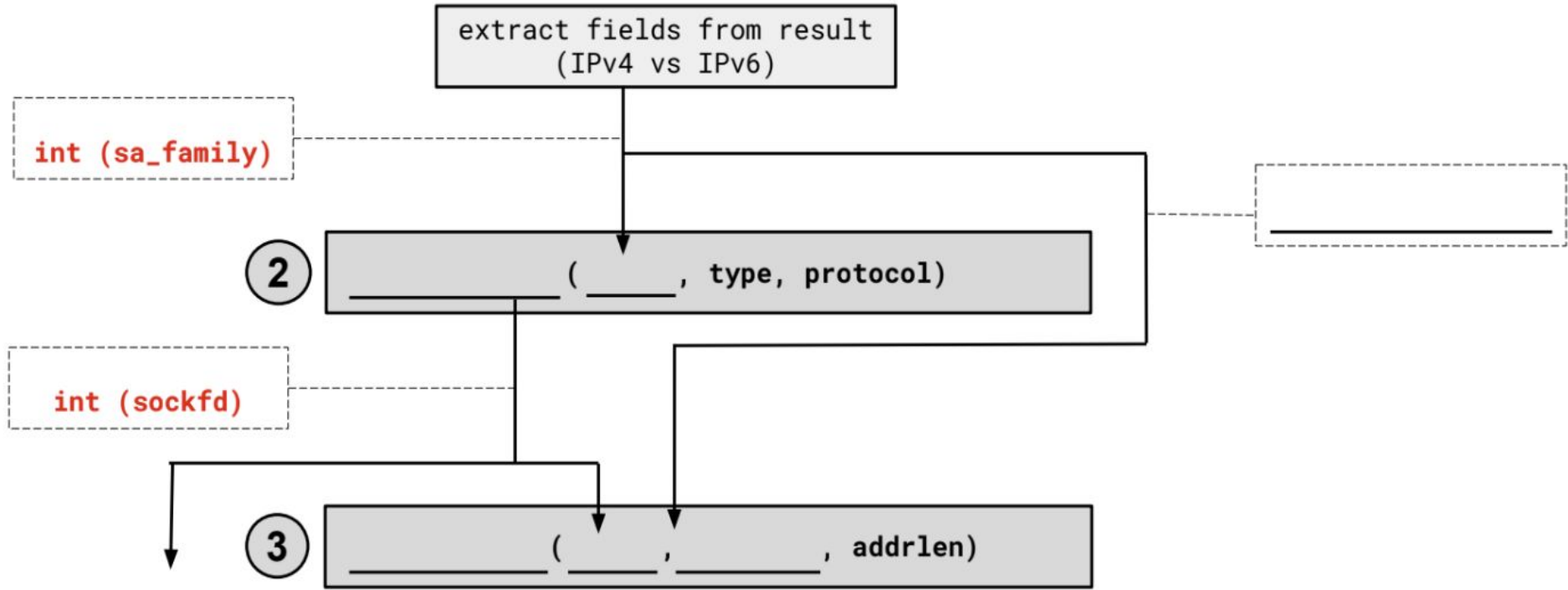
2. socket()

```
int socket(int domain,    // AF_INET, AF_INET6
           int type,     // SOCK_STREAM (TCP)
           int protocol); // 0
```

- Creates a “raw” socket, ready to be bound
- Returns file descriptor (`sockfd`) on success, `-1` on failure



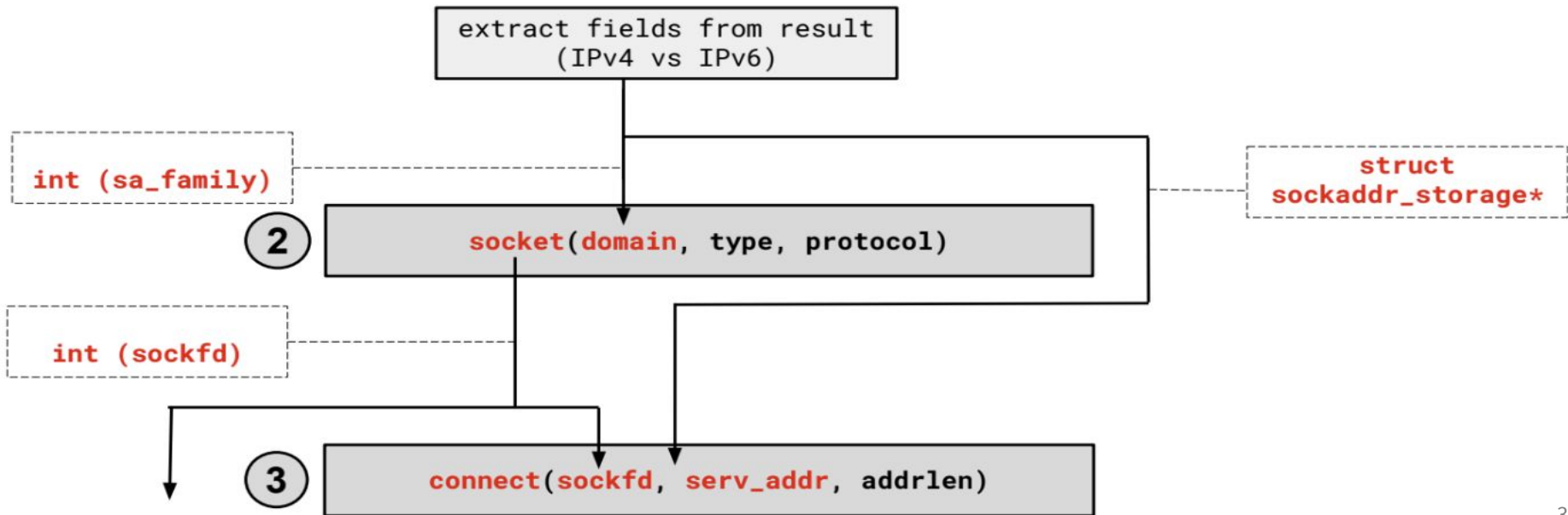
3.



3. connect()

```
int connect (int sockfd,                // from 2  
            const struct sockaddr *serv_addr, // from 1  
            socklen_t addrlen);        // size of serv_addr
```

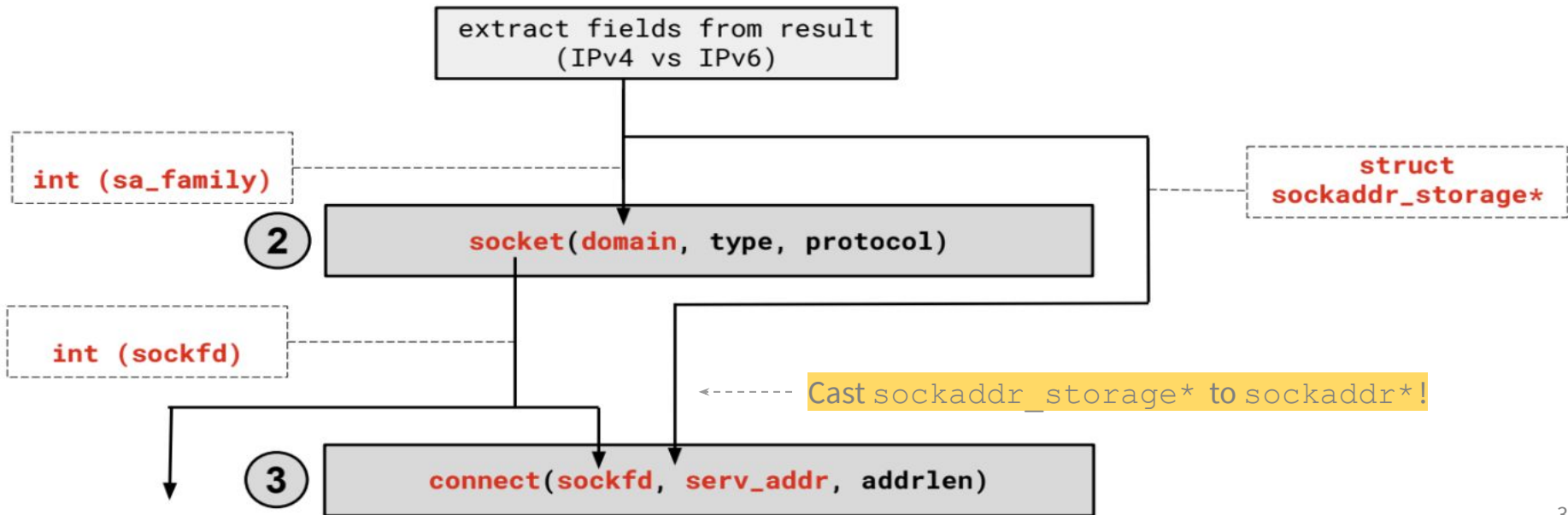
- Connects an available socket to a specified address
- Returns 0 on success, -1 on failure



3. connect()

```
int connect (int sockfd,                // from 2
             const struct sockaddr *serv_addr, // from 1
             socklen_t addrlen);        // size of serv_addr
```

- Connects an available socket to a specified address
- Returns 0 on success, -1 on failure



4. read/write and 5. close

- Thanks to the file descriptor abstraction, use as normal!
- `read` from and `write` to a buffer, the OS will take care of sending/receiving data across the network
- Make sure to `close` the fd afterward

