Porting applications & DNS issues

socket interface extensions for IPv6

Eva M. Castro

ecastro@dit.upm.es

Contents

* Introduction

* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
  - Data structures
  - Conversion functions
  - Socket calls

* IPv4 and IPv6 Interoperability
  - Dual-stack

* Study case: ISABEL

Introduction

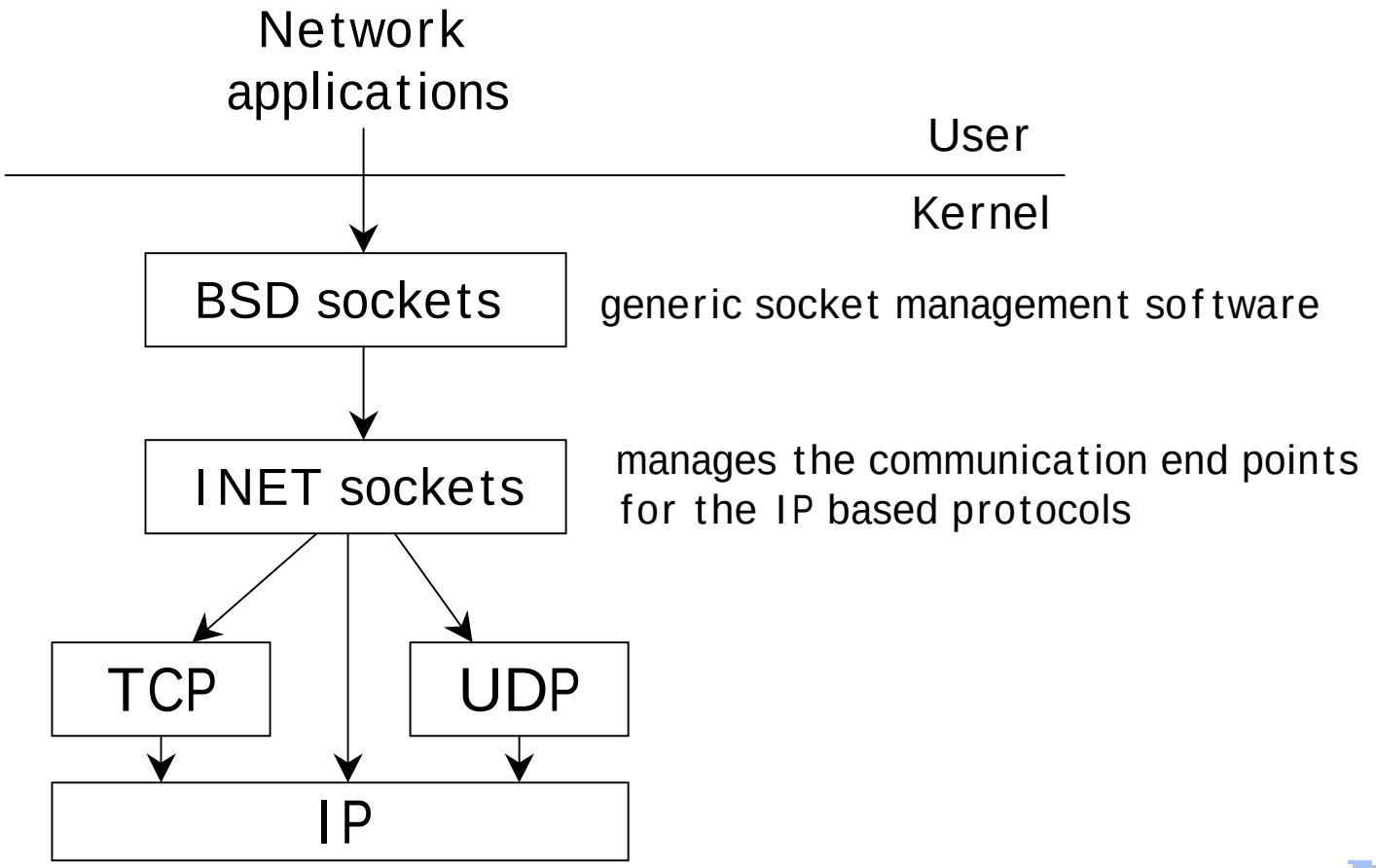
Existing IPv4 applications can work on IPv6 :

1. Porting the application networking part to manage IPv6 addresses.
Using socket interface:
    * RFC 2553: Basic socket interface extensions for IPv6 (Draft Oct 2000).
    * RFC 2292: Advanced sockets API for IPv6 (Draft Nov 2000).

2. Reengineering the application to use new features in addition to larger address space:
    * QoS: Flow labels and priorities.
    * Mobility.
    * Multihoming.
    * Anycast communication.
    * Security: authentication and encryption.
    * ...

Introduction

BSD socket model => abstraction of the communication details to a common interface



Contents

* Introduction

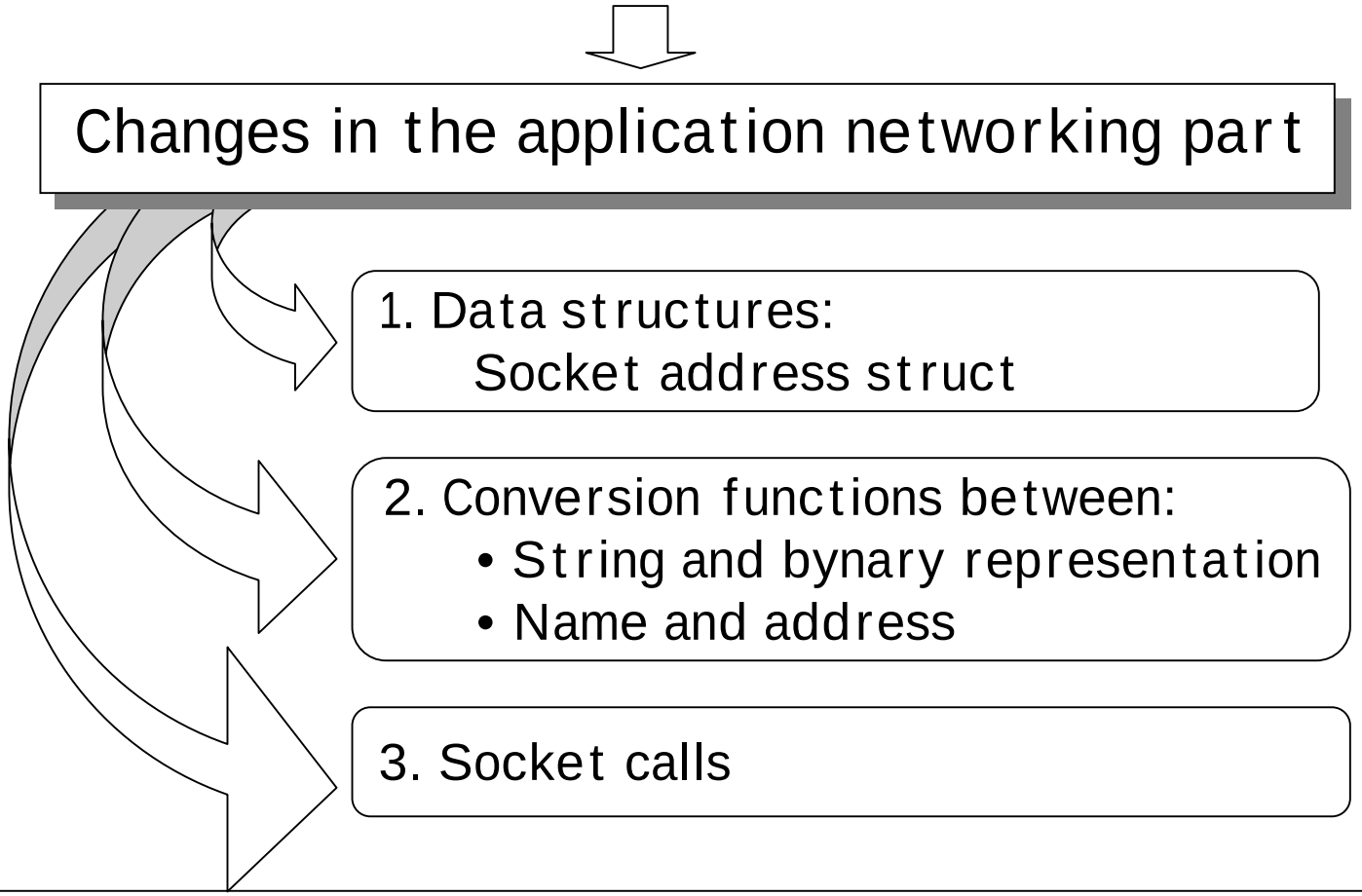
* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
    - Data structures
    - Conversion functions
    - Socket calls

* IPv4 and IPv6 Interoperability
    - Dual-stack

* Study case: ISABEL

Porting IPv4 applications to IPv6, using socket interface extensions for IPv6

IPv6 addresses are 128 bit long.



Contents

* Introduction

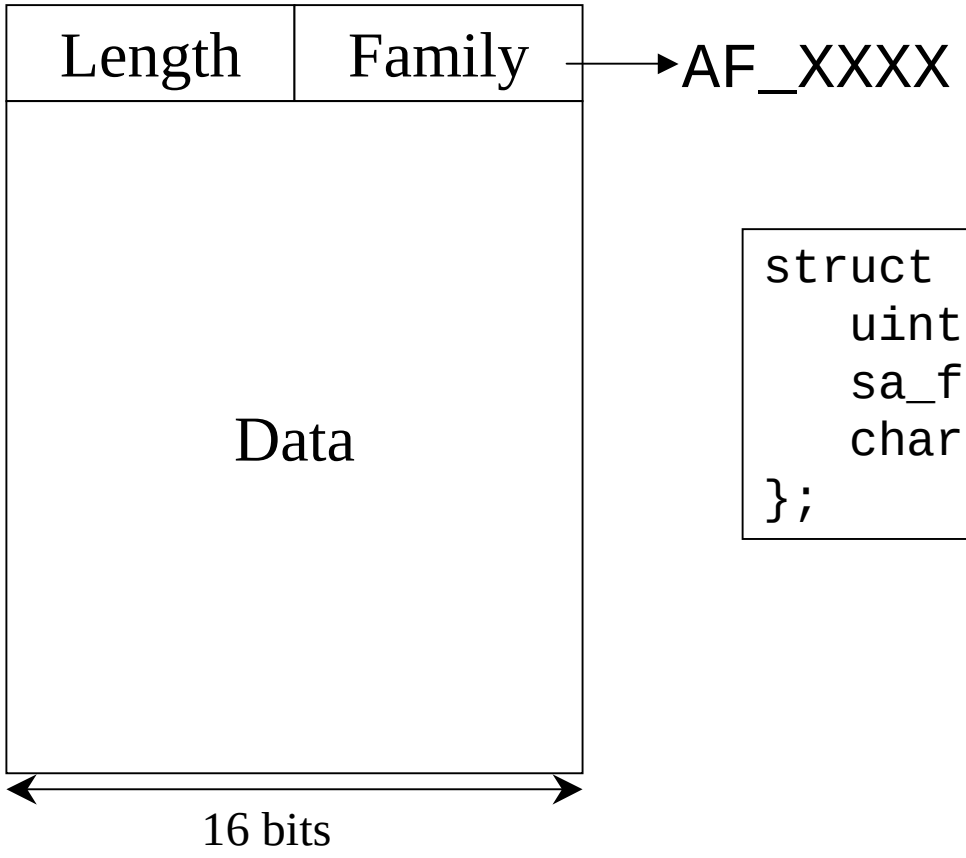
* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
  - Data structures
  - Conversion functions
  - Socket calls

* IPv4 and IPv6 Interoperability
  - Dual-stack

* Study case: ISABEL

Generic Socket Address Struct

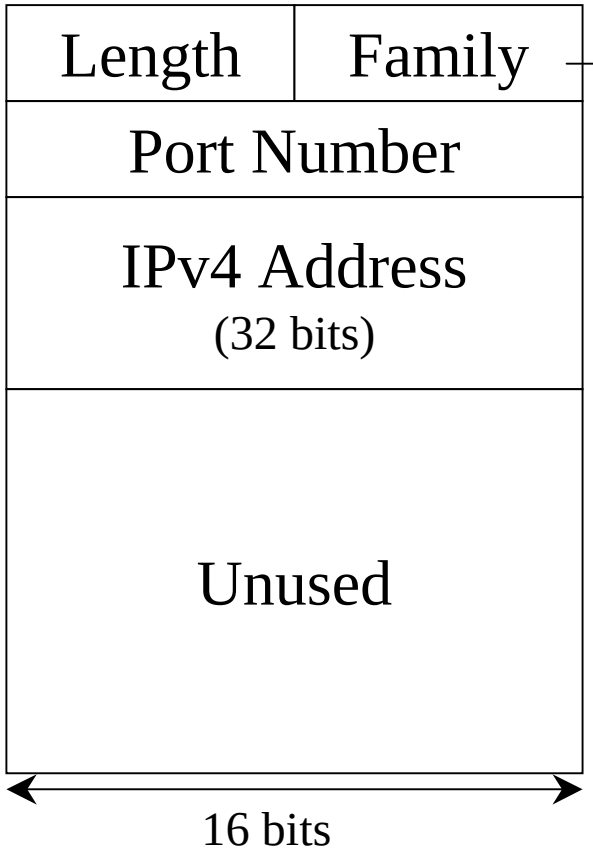
sockaddr




```
struct sockaddr {
   uint8_t       sa_len;
   sa_family_t   sa_family;
   char          sa_data[14];
};
```

IPv4 Socket Address Struct

sockaddr_in

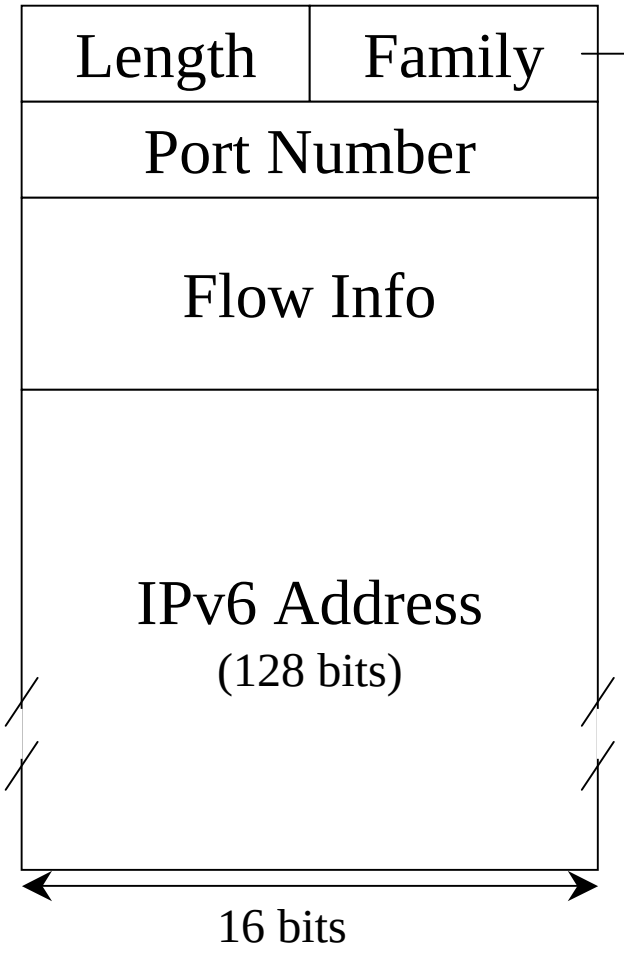



```
struct sockaddr_in {
   uint8_t          sin_len;
   sa_family_t      sin_family;
   in_port_t        sin_port;
   struct in_addr   sin_addr;
   char             sin_zero[8];
};

struct in_addr {
   uint32_t         s_addr;
};
```

IPv6 Socket Address Struct

sockaddr_in6




```
struct sockaddr_in6 {
   uint8_t           sin6_len;
   sa_family_t       sin6_family;
   in_port_t         sin6_port;
   uint32_t          sin6_flowinfo;
   struct in6_addr   sin6_addr;
};

struct in6_addr {
      uint8_t        s6_addr[16];
};
```

Contents

* Introduction

* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
  - Data structures
  - Conversion functions
  - Socket calls

* IPv4 and IPv6 Interoperability
  - Dual-stack

* Study case: ISABEL

String and binary Address Conversions

Conversions between the text representation and the binary value in network byte ordered (socket address structure).

1. String to 32-bit network byte ordered "a.b.c.d" → a.b.c.d

IPv4 only

```
int         inet_aton (const char *strptr,
                       struct in_addr *addrptr);
in_addr_t   inet_addr (const char *strptr); /*deprecated*/
```

IPv4 & IPv6

```
int         inet_pton (int family, const char *strptr,
                       void *addrptr);
```

2. 32-bit network byte ordered to string a.b.c.d → "a.b.c.d"

IPv4 only

```
int         inet_ntoa (struct in_addr inaddr);
```

IPv4 & IPv6

```
int         inet_ntop (int family, const char *strptr,
                       void *addrptr);
```

Conversions from Host Name to IP Address

```
struct hostent *gethostbyname(const char *hostname);
```

* Query for a DNS A record

"hostname.domain" → a.b.c.d IPv4 address

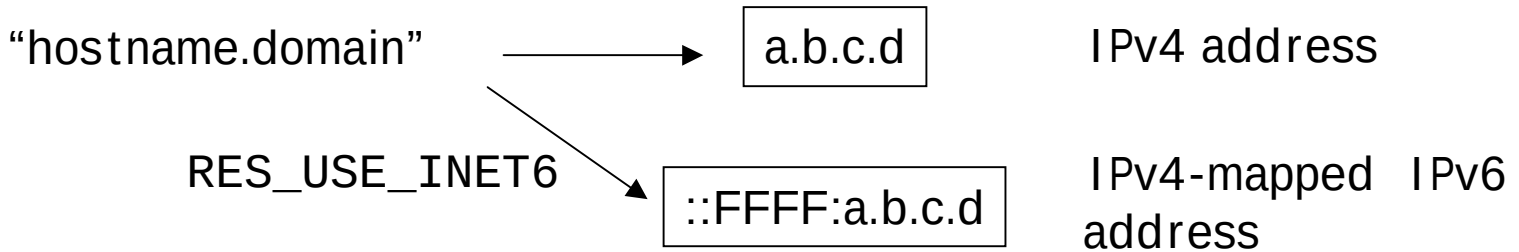
* Query for a DNS AAAA record, **RES_USE_INET6** resolver option is required:

"hostname.domain" → x:x:x:x:x:x:x:x IPv6 address (if not found, IPv4-mapped IPv6 address = ::FFFF:x.x.x.x)

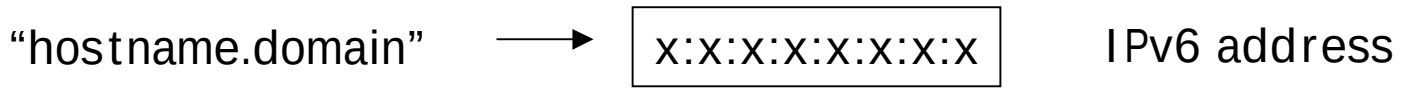
Conversions from Host Name to IP Address

```
struct hostent *gethostbyname2(const char *hostname, int family);
```

* Query for a DNS A record, **family=AF_INET** is required:



* Query for a DNS AAAA record, **family=AF_INET6** is required:

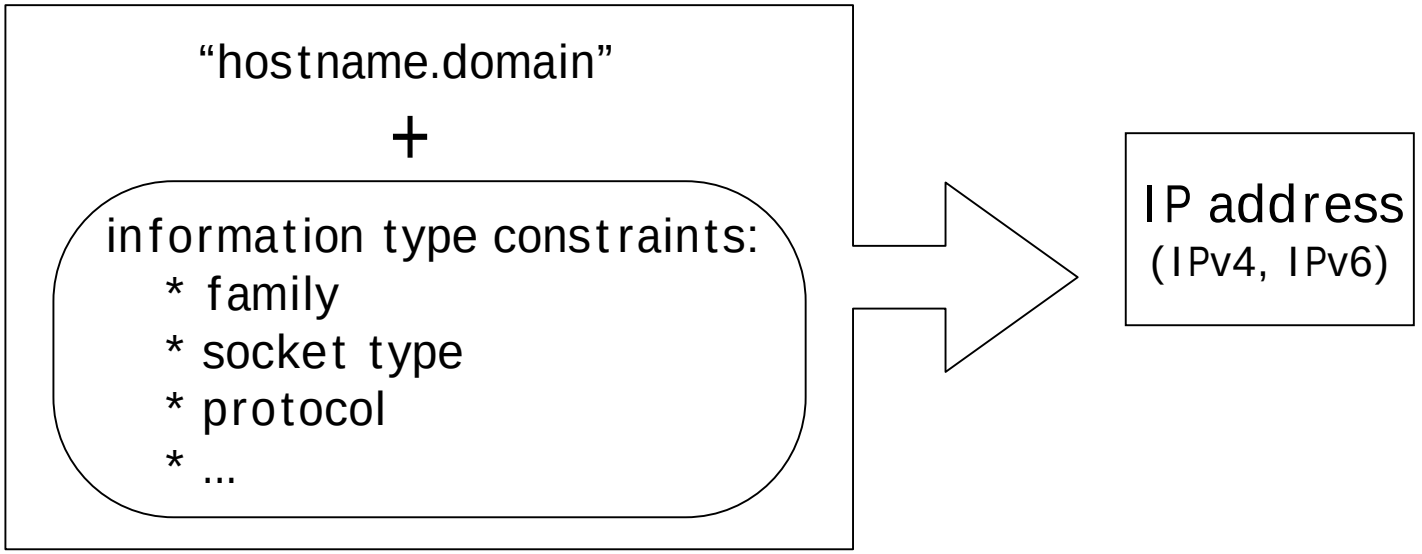


dit UPM

Conversions from Host Name to IP Address

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

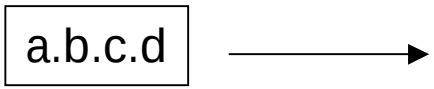
* Protocol independent function.



Conversions from IP Address to Host Name

```
struct hostent *gethostbyaddr(const char *addr,
                              size_t len,
                              int family);
```

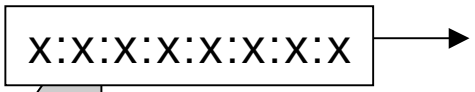
* From IPv4 address:



“hostname.domain”

```
addr   => in_addr
len    => sizeof(in_addr)
family => AF_INET
```

* From IPv6 address:



“hostname.domain”

```
addr   => in6_addr
len    => sizeof(in6_addr)
family => AF_INET6
```

=> query in the domain for IPv4 address

Conversions from to IP Address to Host Name

```
int getnameinfo (const struct sockaddr *sockaddr,
                 socklen_t addrlen,
                 char *host, size_t hostlen,
                 char *serv, size_t servlen, int flags);
```

* Protocol independent function.

* From IPv4 address:

a.b.c.d → "hostname.domain"

```
sockaddr => sockaddr_in
len => sizeof(sockaddr_in)
```

* From IPv6 address:

x:x:x:x:x:x:x:x → "hostname.domain"

```
sockaddr => sockaddr_in6
len => sizeof(sockaddr_in6)
```

Contents

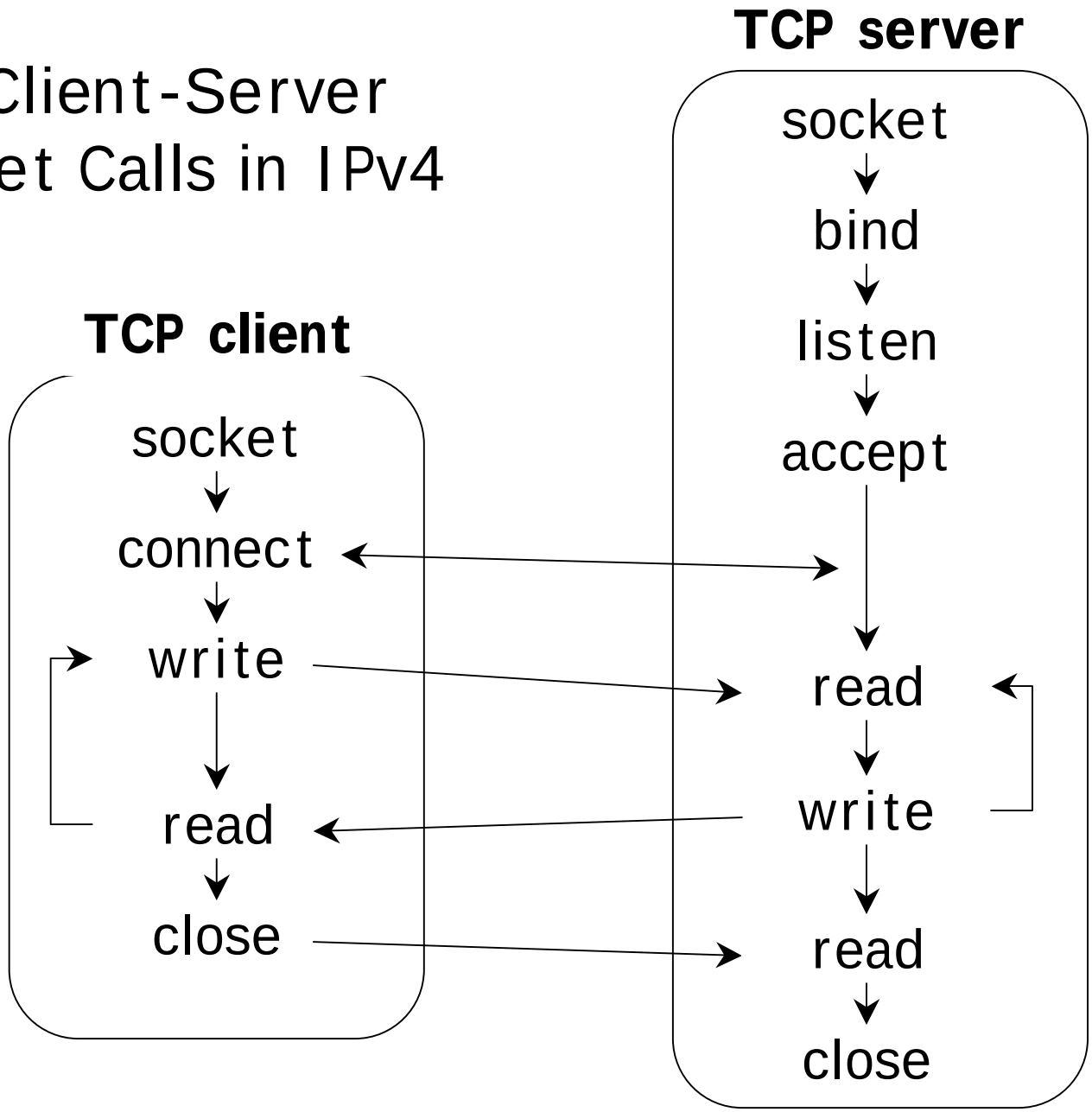
* Introduction

* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
  - Data structures
  - Conversion functions
  - Socket calls

* IPv4 and IPv6 Interoperability
  - Dual-stack

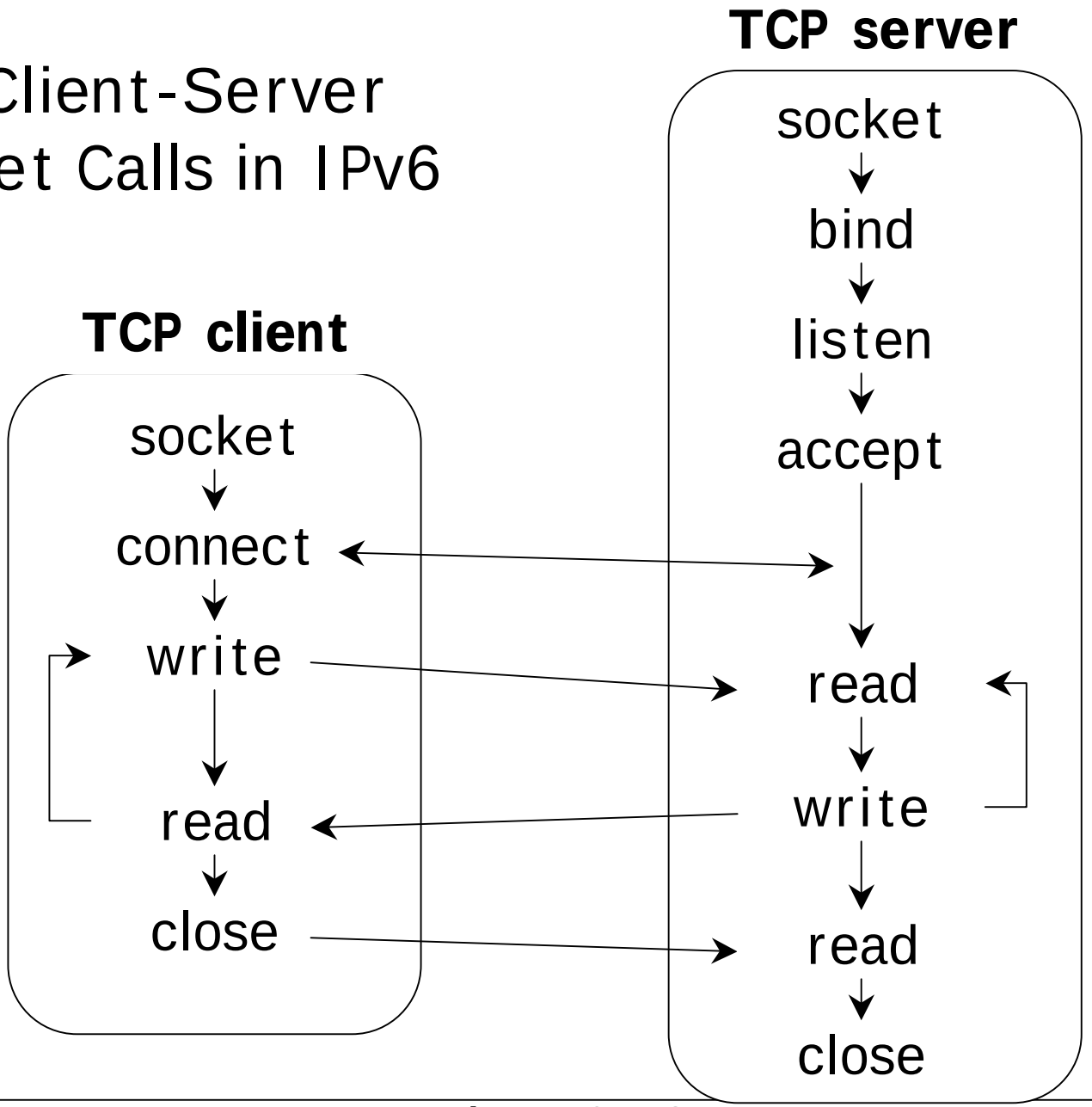
* Study case: ISABEL

TCP Client-Server Socket Calls in IPv4



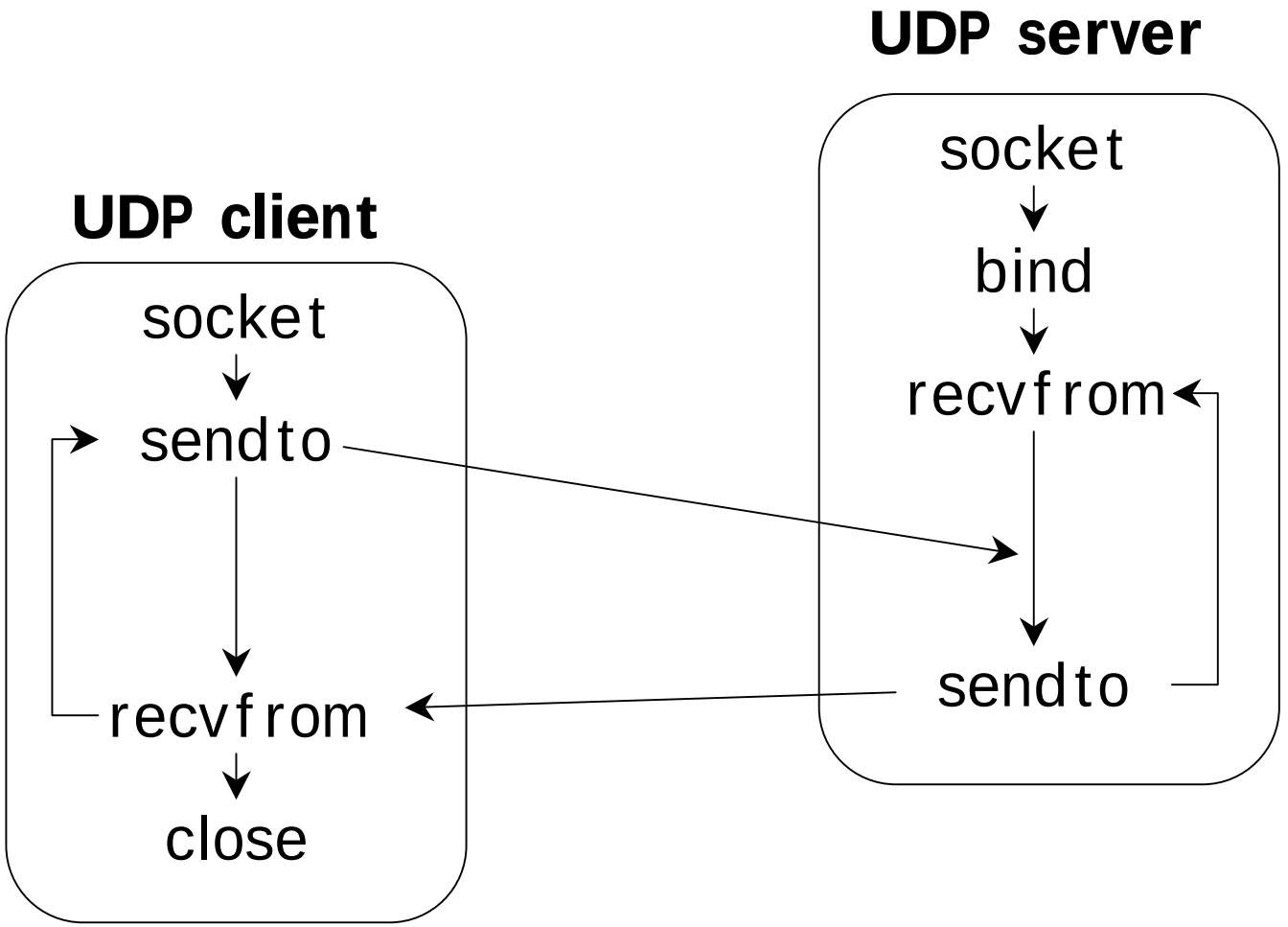
dit UPM

TCP Client-Server Socket Calls in IPv6

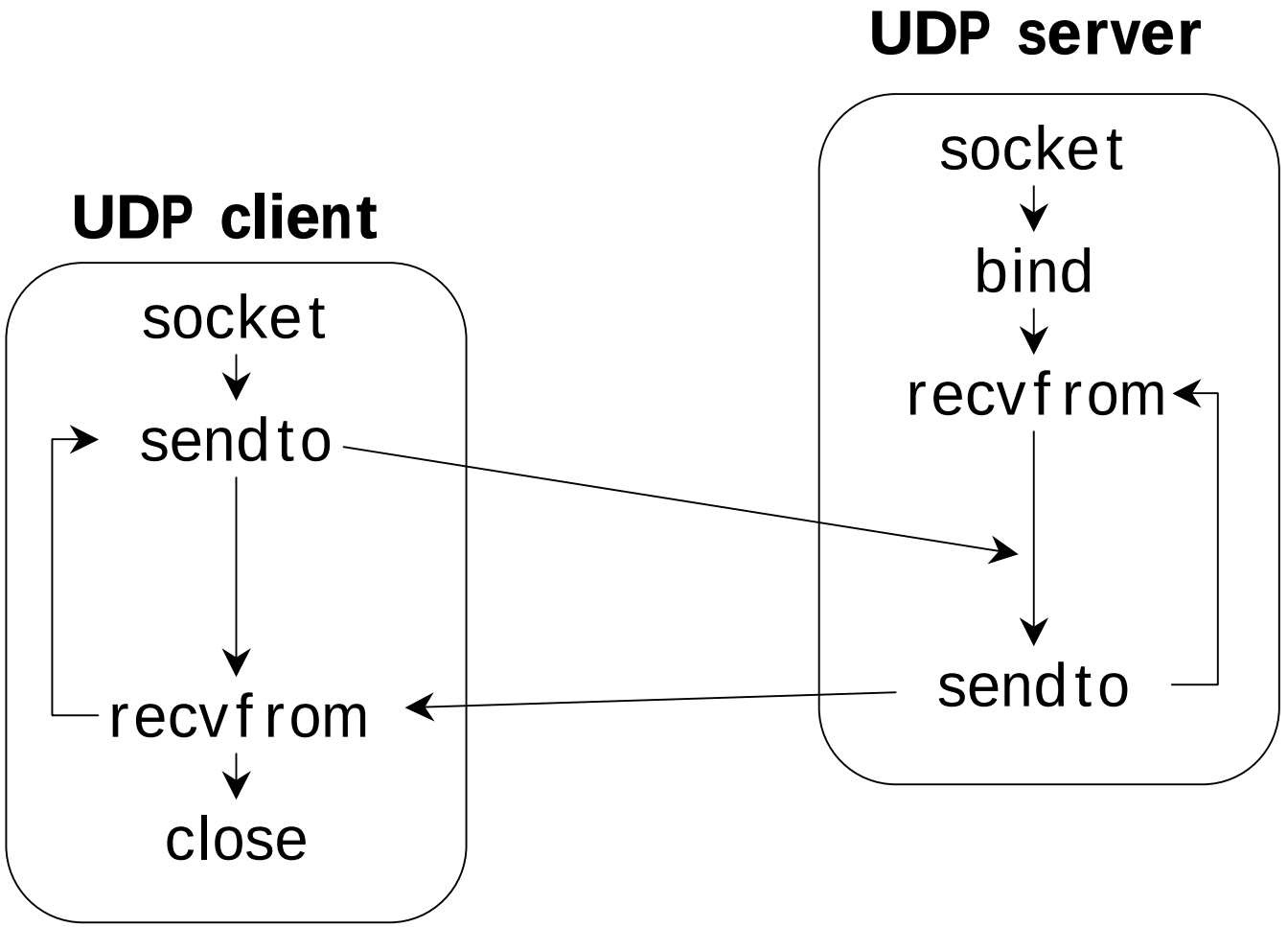


dit UPM

UDP Client-Server
Socket Calls in IPv4



UDP Client-Server
Socket Calls in IPv6



Same socket calls in IPv4 and IPv6

Some values in the API arguments change when using IPv4 or IPv6.

* Address family, socket type and protocol in the **socket** call:

```
int socket (int family, int type, int protocol);
```

family \ type	SOCK_STREAM	SOCK_DGRAM	SOCK_RAW
AF_INET	TCP	UDP	IPv4
AF_INET6	TCP	UDP	IPv6

protocol

* A pointer to a **sockaddr** struct and its length are required in some calls:

IPv4:
- Casting `(sockaddr_in *)` to `(sockaddr *)`
- Length => `sizeof(sockaddr_in)`

IPv6:
- Casting `(sockaddr_in6 *)` to `(sockaddr *)`
- Length => `sizeof(sockaddr_in6)`

Contents

* Introduction

* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
  - Data structures
  - Conversion functions
  - Socket calls

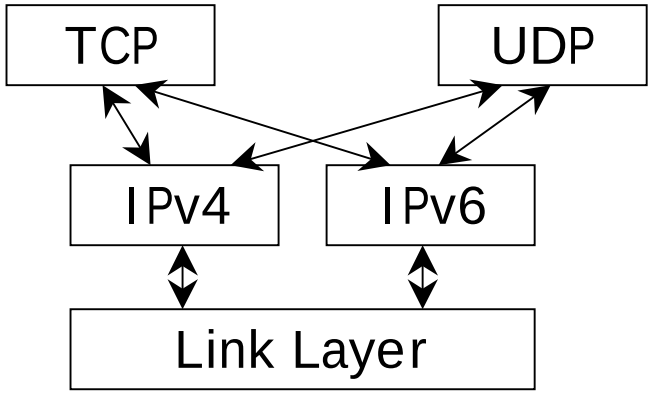
* IPv4 and IPv6 Interoperability
  - Dual-stack

* Study case: ISABEL

IPv4 and IPv6 Interoperability

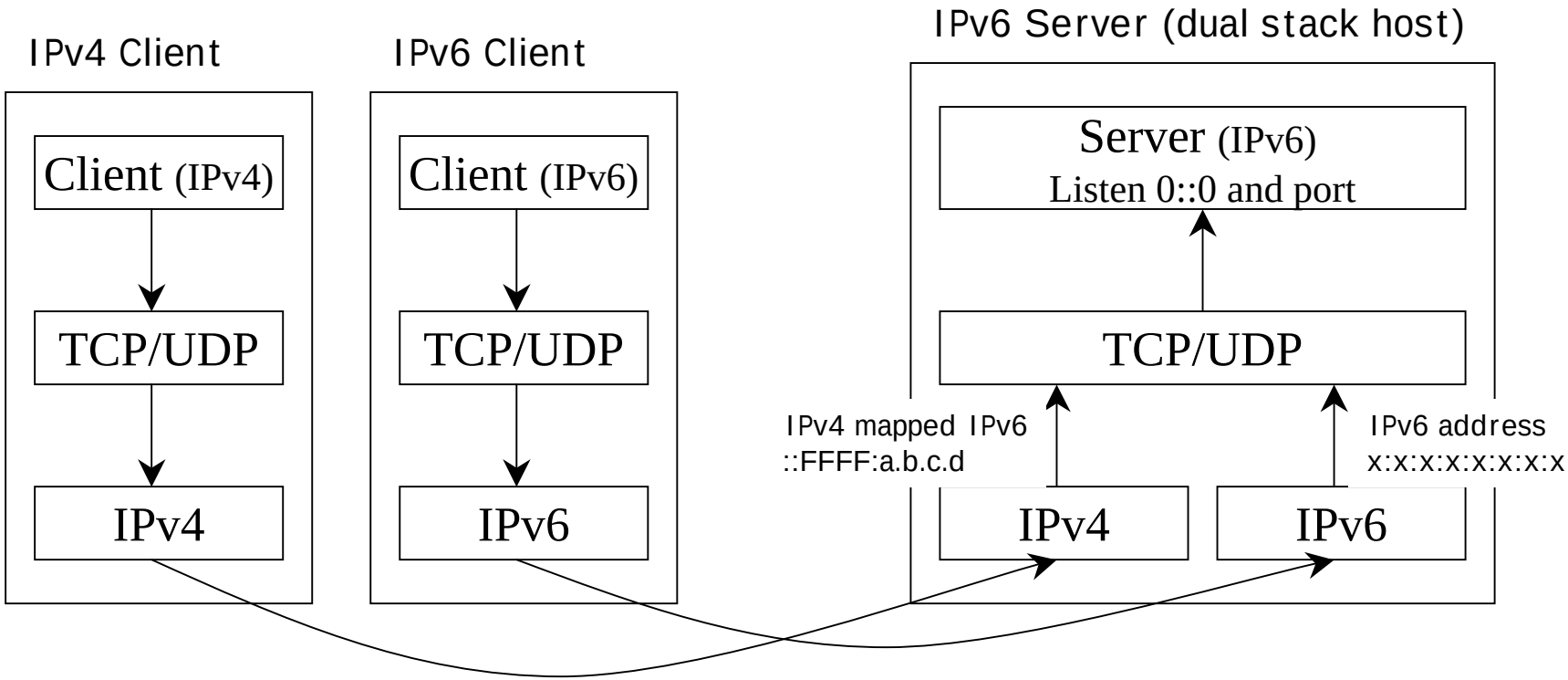
* During transition existing IPv4 applications should work with new IPv6 applications.

Support IPv4 and IPv6 layers (Dual stack)



IPv4 client - IPv6 server

* IPv6 servers running in a dual stack host accept connections from IPv4 and IPv6 clients.

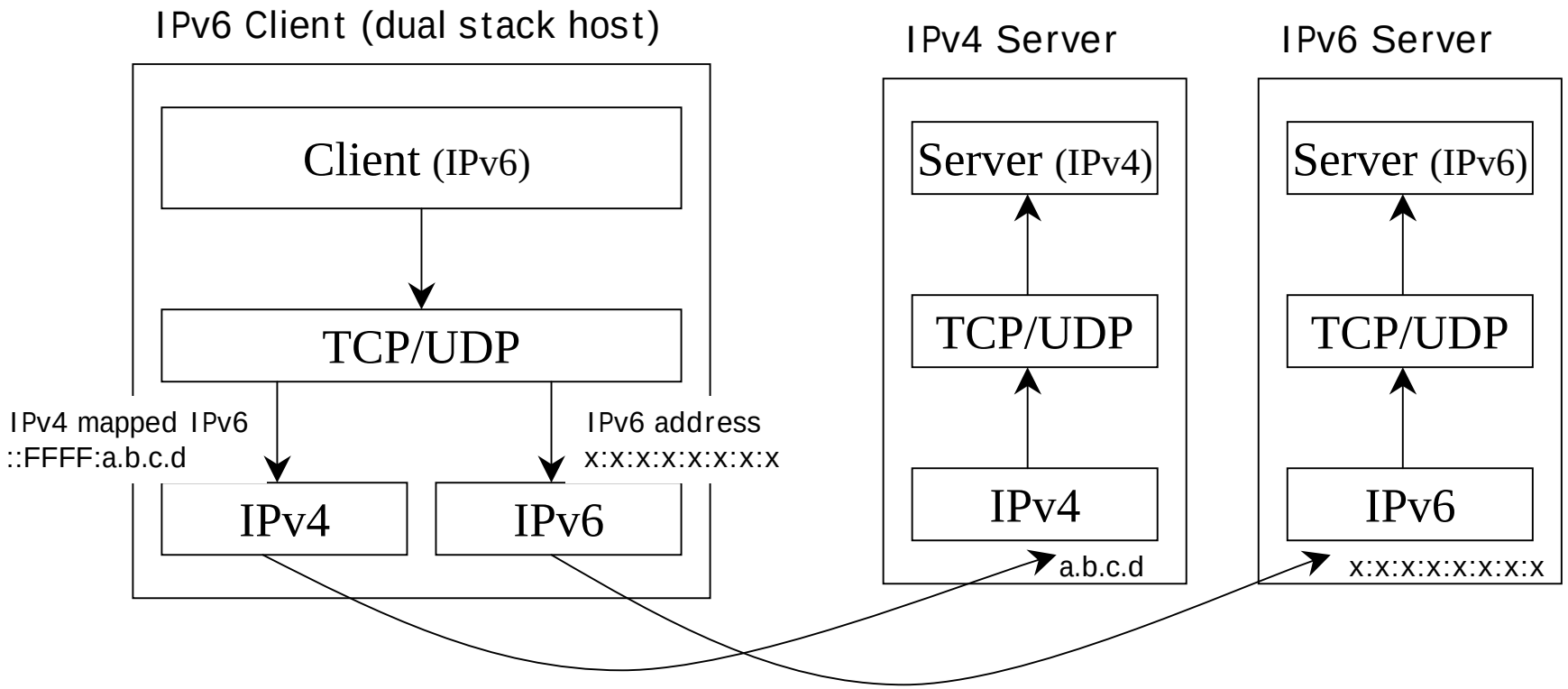


* Kernel converts from IPv4 to IPv4-mapped IPv6 address.

dit UPM

IPv6 client – IPv4 server

* IPv6 clients running in a dual stack host connect to IPv4 and IPv6 servers.



* Resolver converts from IPv4 to IPv4-mapped IPv6 address.

Portability

* Make applications protocol independet of IPv4 or IPv6:
  Define a library to handle sockets, hiding protocol dependency and making application code simpler.

* Use **`getaddrinfo`**, **`getnameinfo`** instead of **`gethostbyname`** and **`gethostbyaddr`**.

* Special Addressess: 127.x.y.z and localhost

Contents

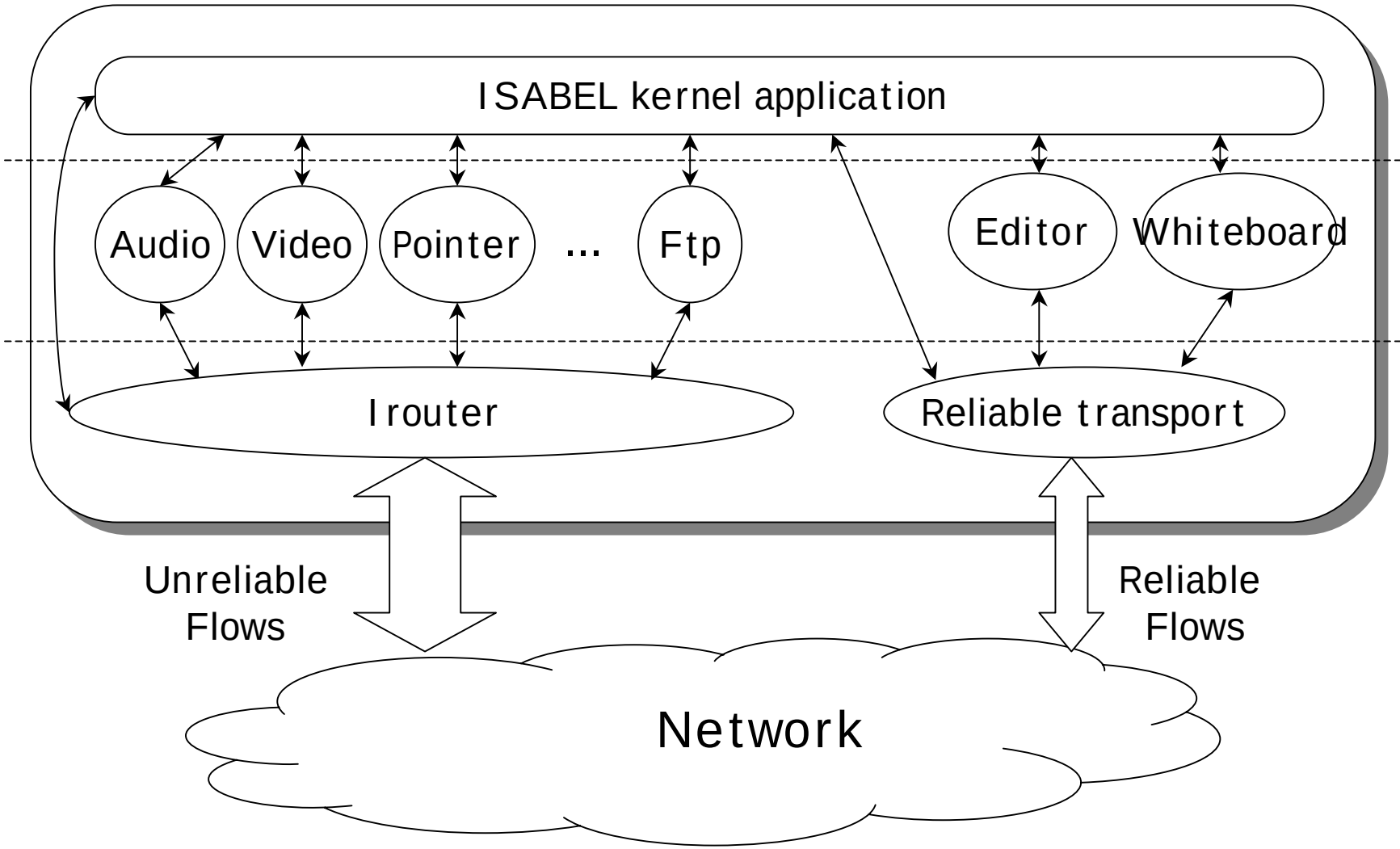
* Introduction

* Porting IPv4 applications to IPv6, using socket interface extensions to IPv6.
  - Data structures
  - Conversion functions
  - Socket calls

* IPv4 and IPv6 Interoperability
  - Dual-stack

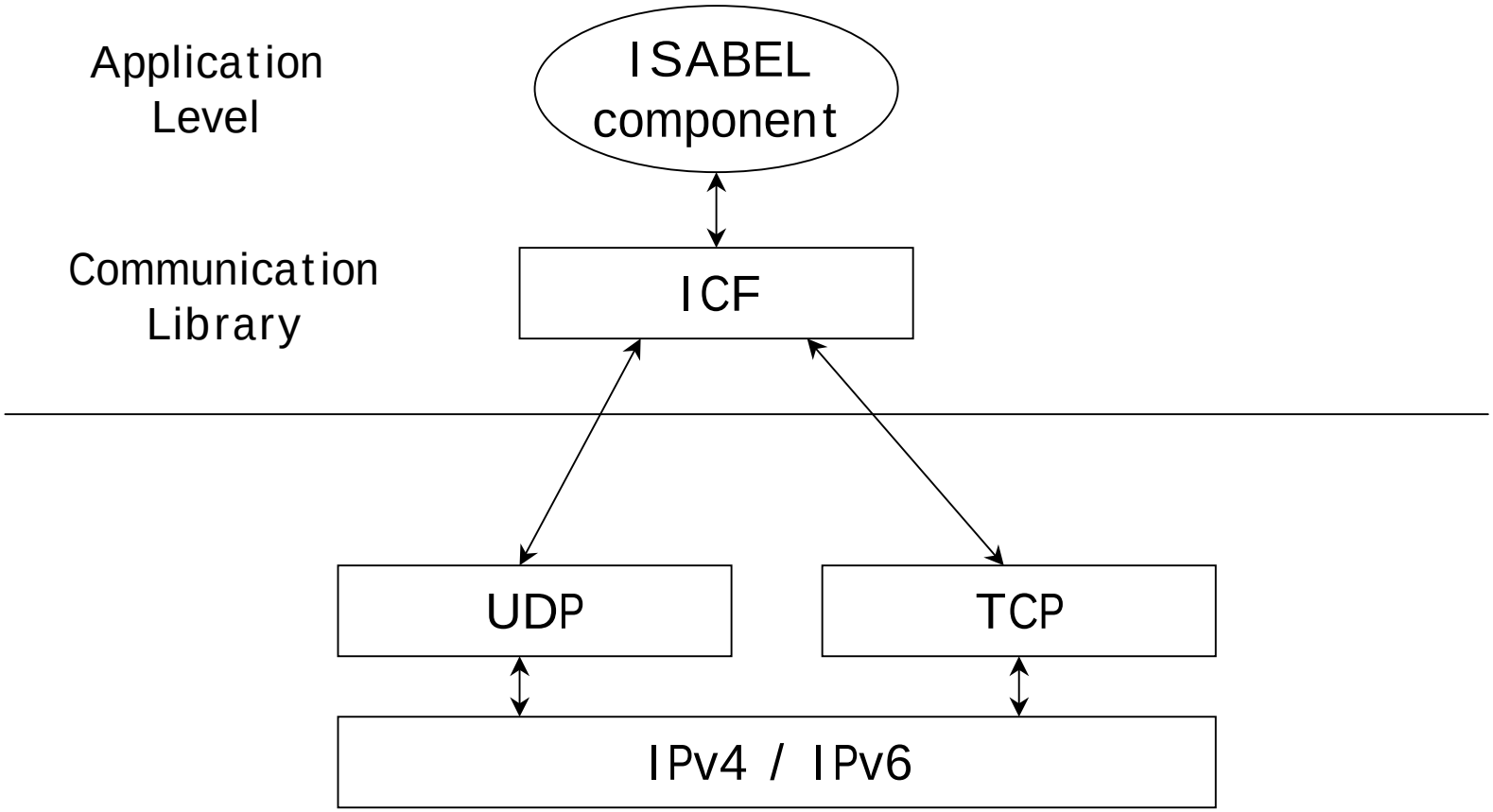
* Study case: ISABEL

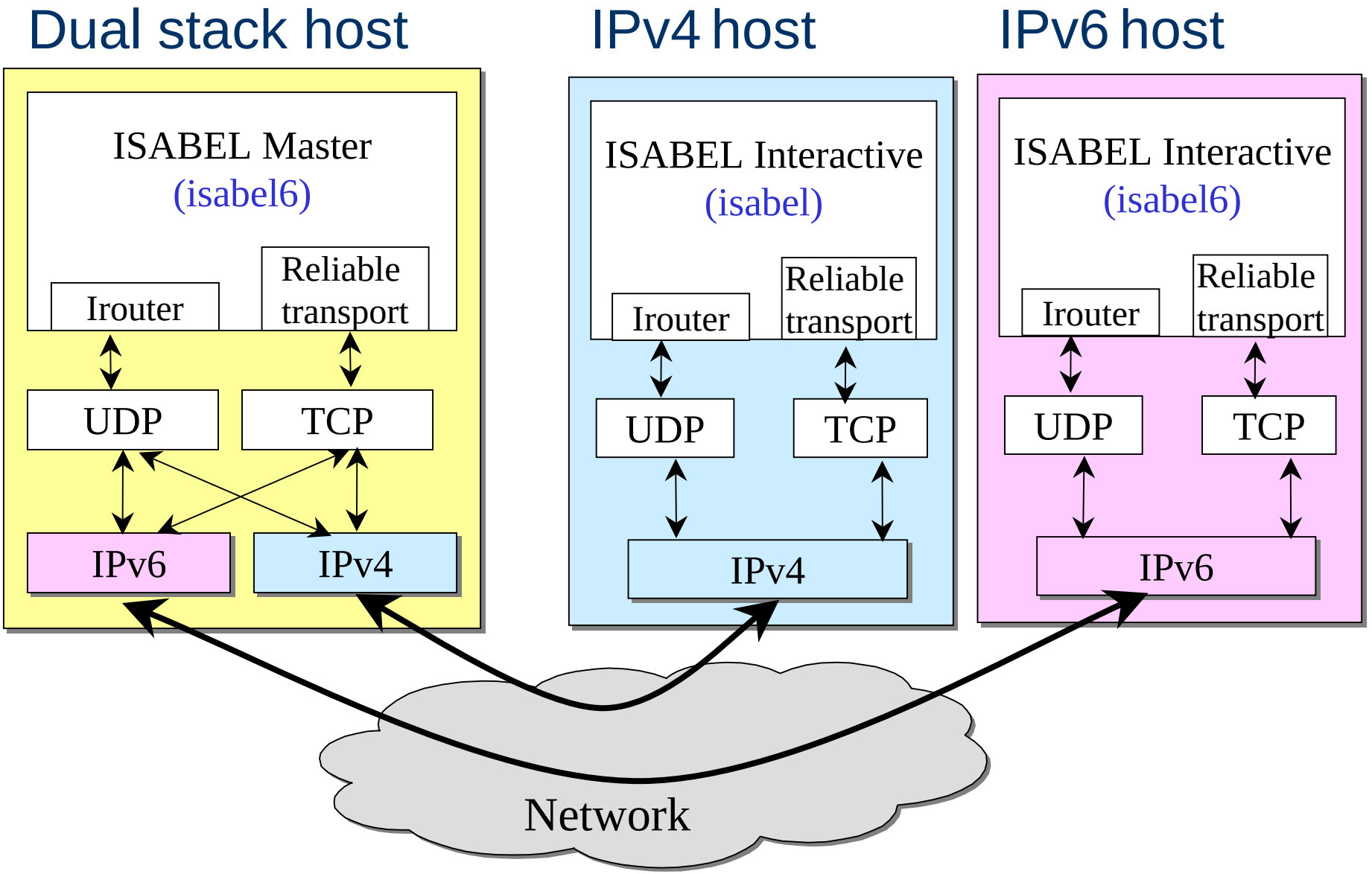
Study case: ISABEL



dit UPM

Study case: ISABEL




Dual stack host
IPv4 host
IPv6 host
ISABEL Master
(isabel6)
Irouter
Reliable transport
UDP
TCP
IPv6
IPv4
ISABEL Interactive
(isabel)
Irouter
Reliable transport
UDP
TCP
IPv4
ISABEL Interactive
(isabel6)
Irouter
Reliable transport
UDP
TCP
IPv6
Network