Applications porting and development with IPv6

Dr. Tomás P. de Miguel
tmiguel@dit.upm.es
Department of Telematic Systems Engineering (DIT)
Technical University of Madrid (UPM)

Eva M. Castro
eva@gsyc.escet.urjc.es
Systems and Communications Group (GSyC)
Experimental Sciences and Technology Department (ESCET)
Rey Juan Carlos University (URJC)

Agenda

- Transition to IPv6 is only a network issue
- Transition to IPv6 implies application code porting
- Describe two code porting exercises

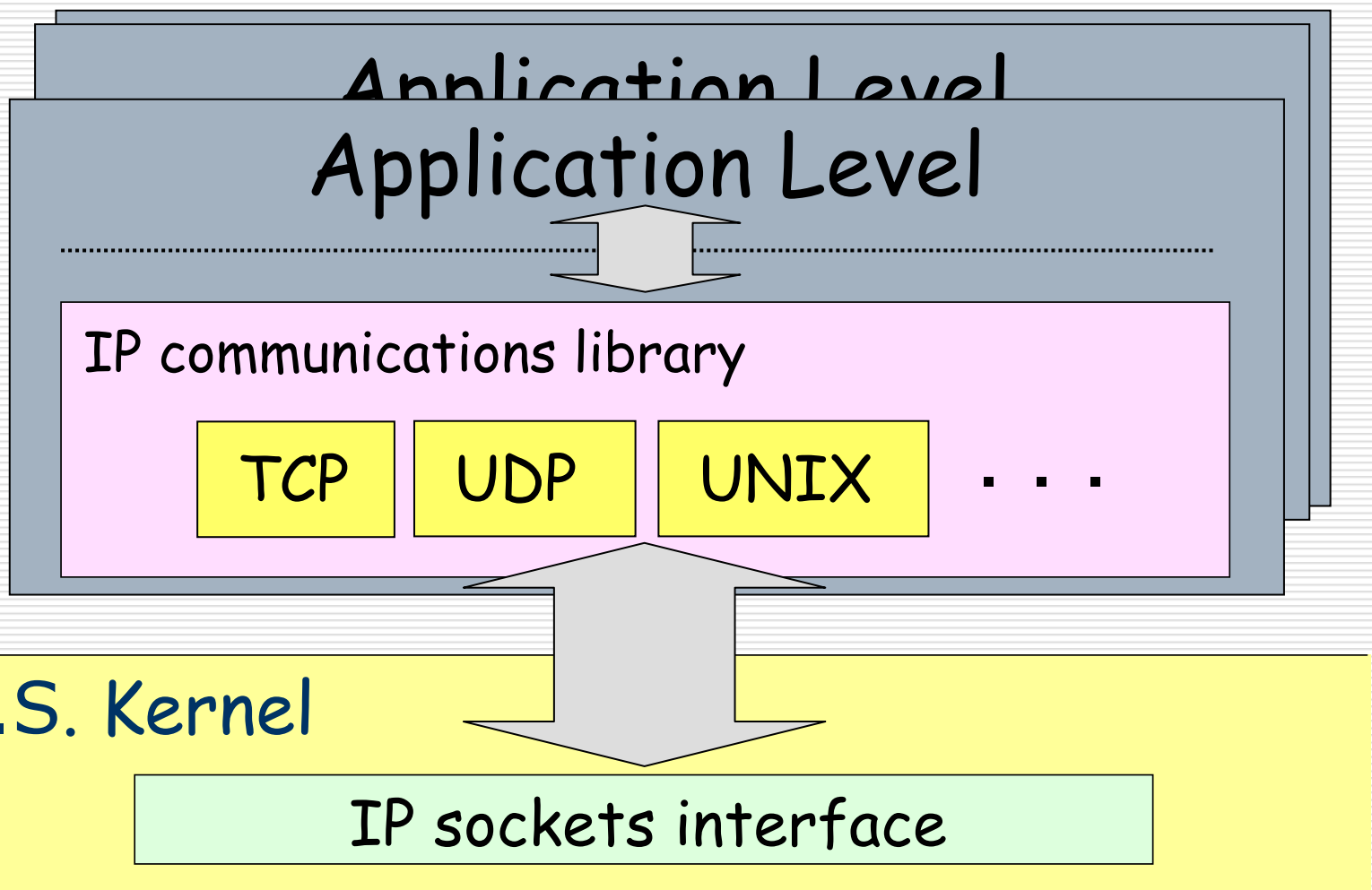
IPv6 transition criteria

- Incremental upgrade
  - Existing hosts can be upgraded at any time
  - Existing hosts can be upgraded at any order
- Incremental development
  - New hosts can be installed at any time
  - New hosts can be installed in any place
- Easy addressing
  - Continue to use existing address after existing networks migration
- Low start-up costs
  - Easy deployment of new networks

Transition impact on upper layers

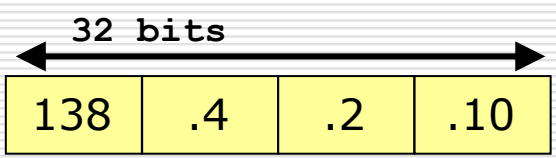
- TCP/IP network architecture is not perfectly layered
- Applications identify destination node
  - Using the IP address
  - Using DNS name
- In both cases application should be revised
  - IPv6 manages new IP addresses
  - Transport layer interface changes
- During transition will be necessary to support both IPv4 and IPv6 nodes.

Applications architecture

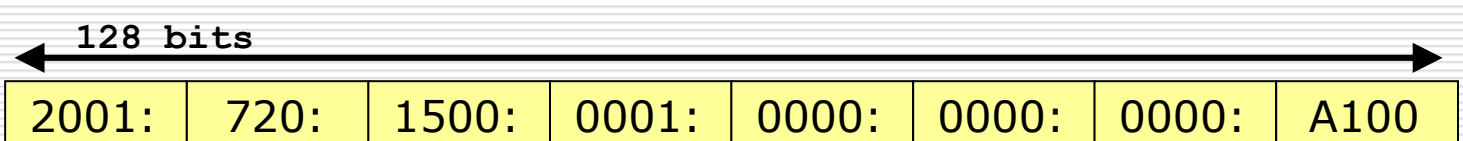


Transition interaction problems

- IP address management
  - IPv4 use 32 bit address

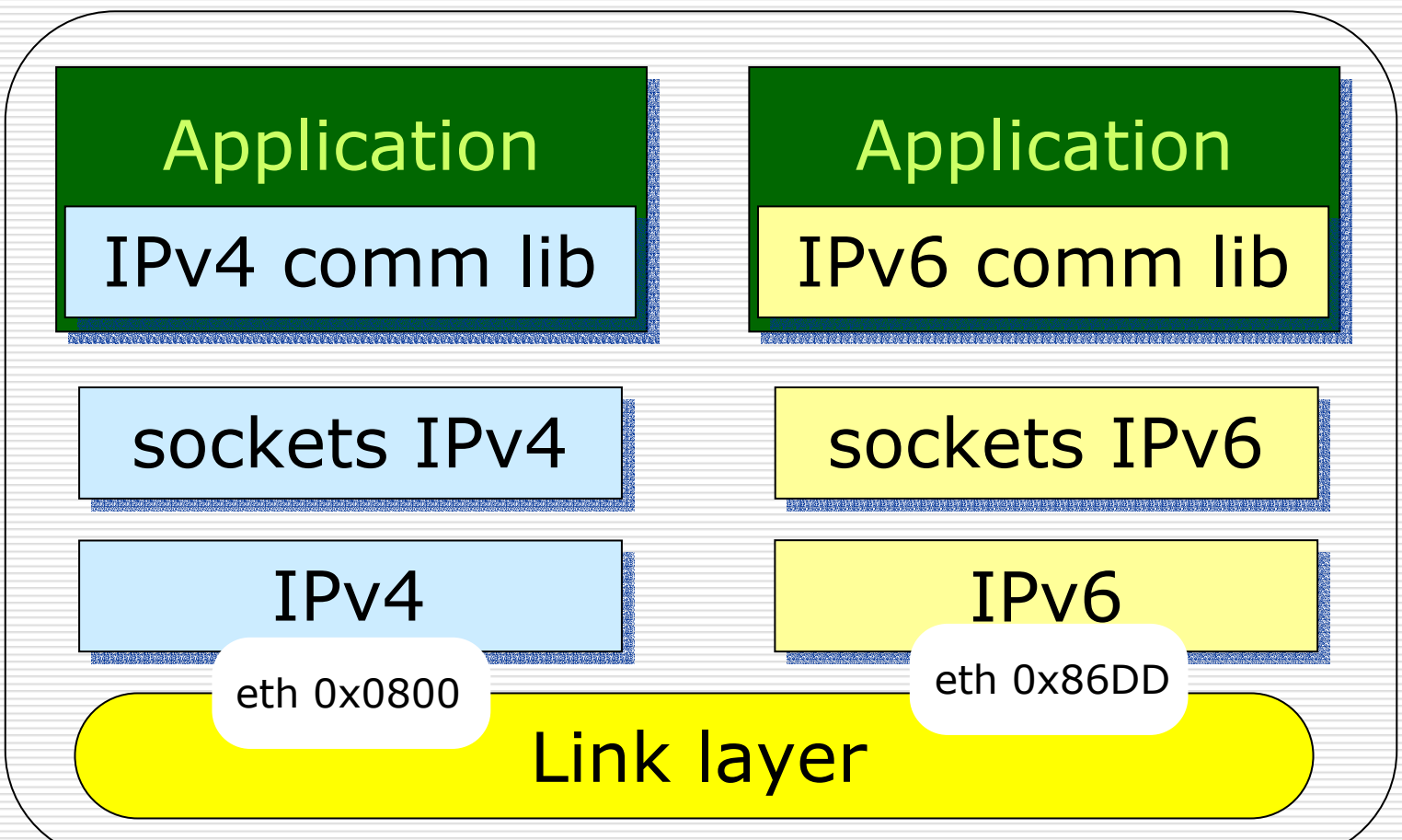


  - IPv6 use 128 bit address



- Socket network programming interface
  - Socket data structures
  - Socket function calls

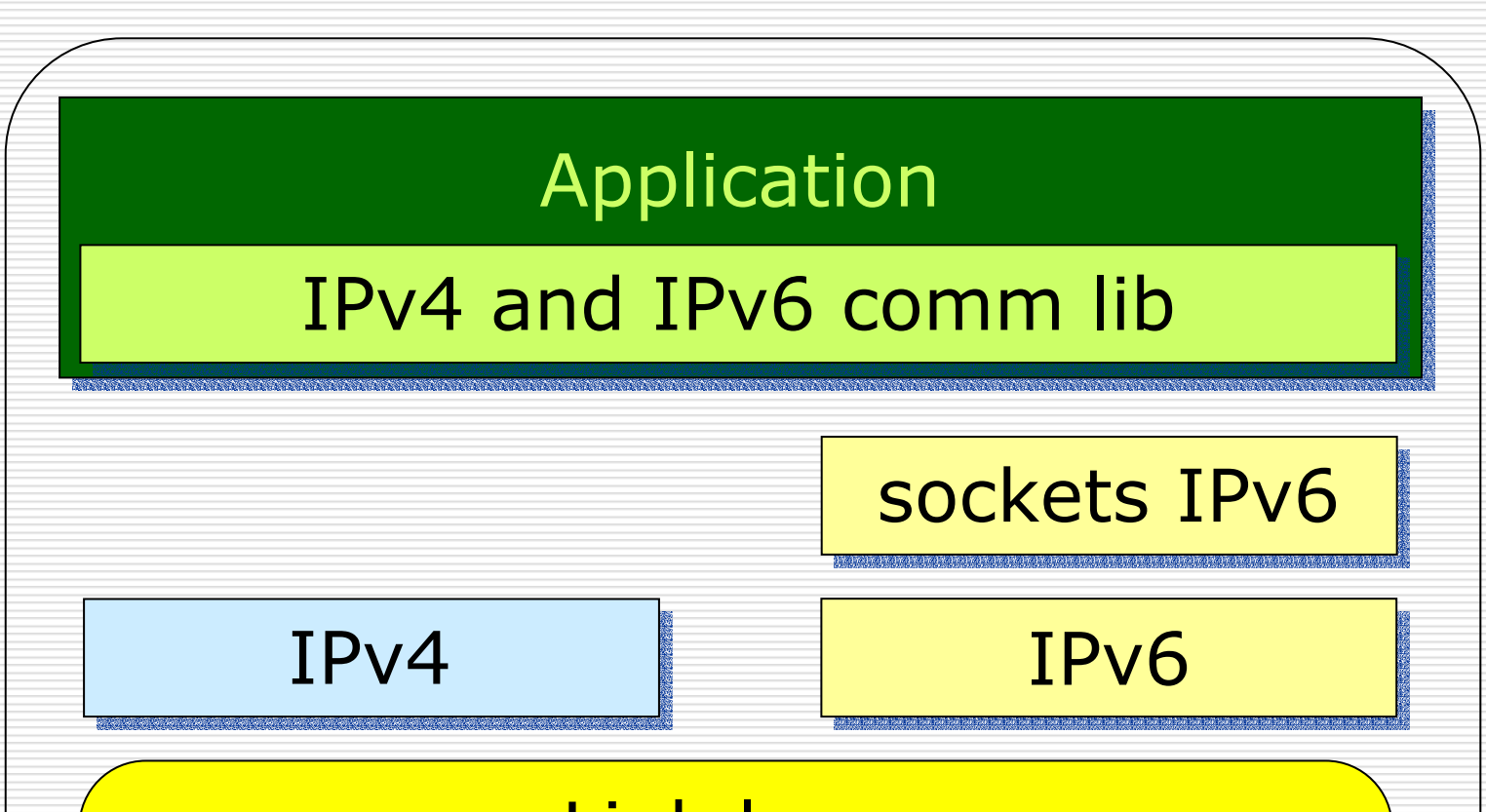
Dual stack platform



Dual stack node

- Have both IPv6 and IPv4 addresses
- Include resolver libraries capable of dealing with A and AAAA/A6 records
- When asking to DNS for a dual node, the order of the answers would normally define the protocol used
- Recommendation
  - do not register IPv6 address in DNS till they are configured and working in systems
- Application uses IPv6 or IPv4 depending on the answers received and their order

Dual stack application

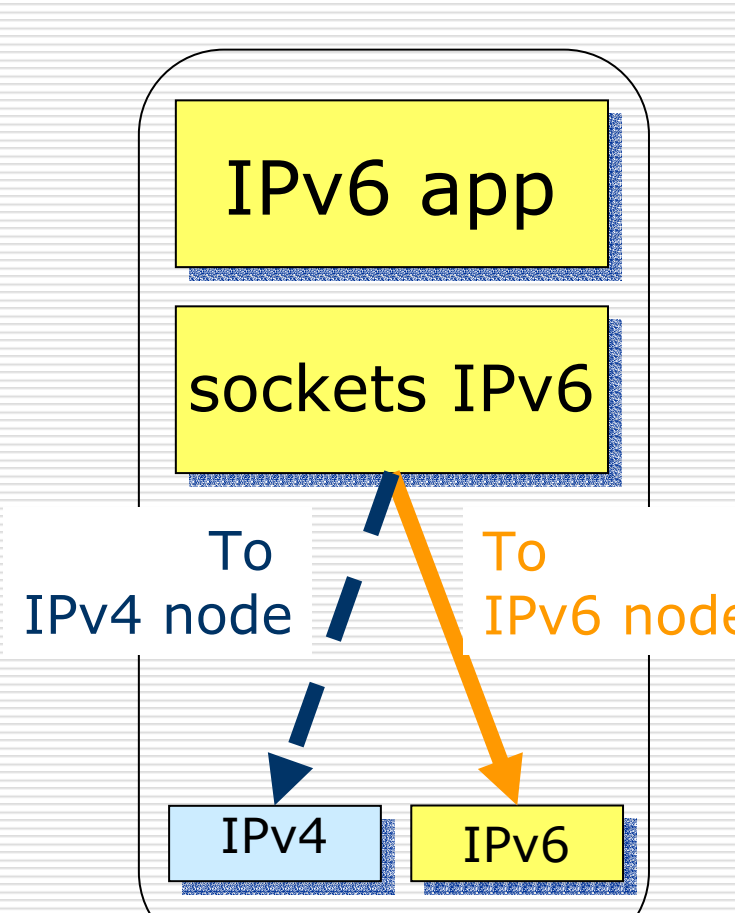


Transition scenarios

- Porting existing networks
  1. All applications are IPv4 only
  2. Provide two different applications
  3. Dual IPv4 and IPv6 applications
- Setup new networks IPv6 only
  1. Dual IPv4 and IPv6 applications
  2. Applications can be IPv6 only
     - If they are IP address independent

IPv6 only application → IPv4 node

- Dual stack host
- Application uses sockets INET6 API
  - It uses IPv6 addresses
- IPv4-mapped IPv6 address are used to represent an IPv4 address in an INET6 socket
  - ::FFFF:a.b.c.d
- Example
  - ::FFFF:138.4.2.10



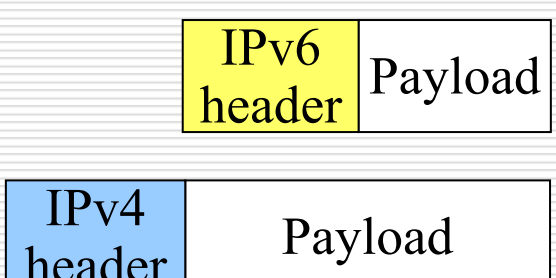
Transition scenarios

From application	using stack	to IPv4 node		to IPv6 node	
		IPv4 net	IPv6 net	IPv4 net	IPv6 net
IPv4	IPv4	IPv4	?	Fail	?
	IPv4/IPv6	IPv4	tunnel	Fail	translator
	IPv6	?	tunnel	?	translator
IPv4/IPv6	IPv4	IPv4	?	translator	?
	IPv4/IPv6	IPv4	translator	translator	IPv6
	IPv6	?	translator	?	IPv6
IPv6	IPv4	Fail	?	tunnel	?
	IPv4/IPv6	Fail	translator	tunnel	IPv6
	IPv6	?	translator	?	IPv6

? It has no sense

Tunneling

- It is a basic mechanism to transport IPv6 packets over IPv4 networks.
- IPv6 packets are encapsulated on IPv4 packets to traverse non IPv6 capable networks

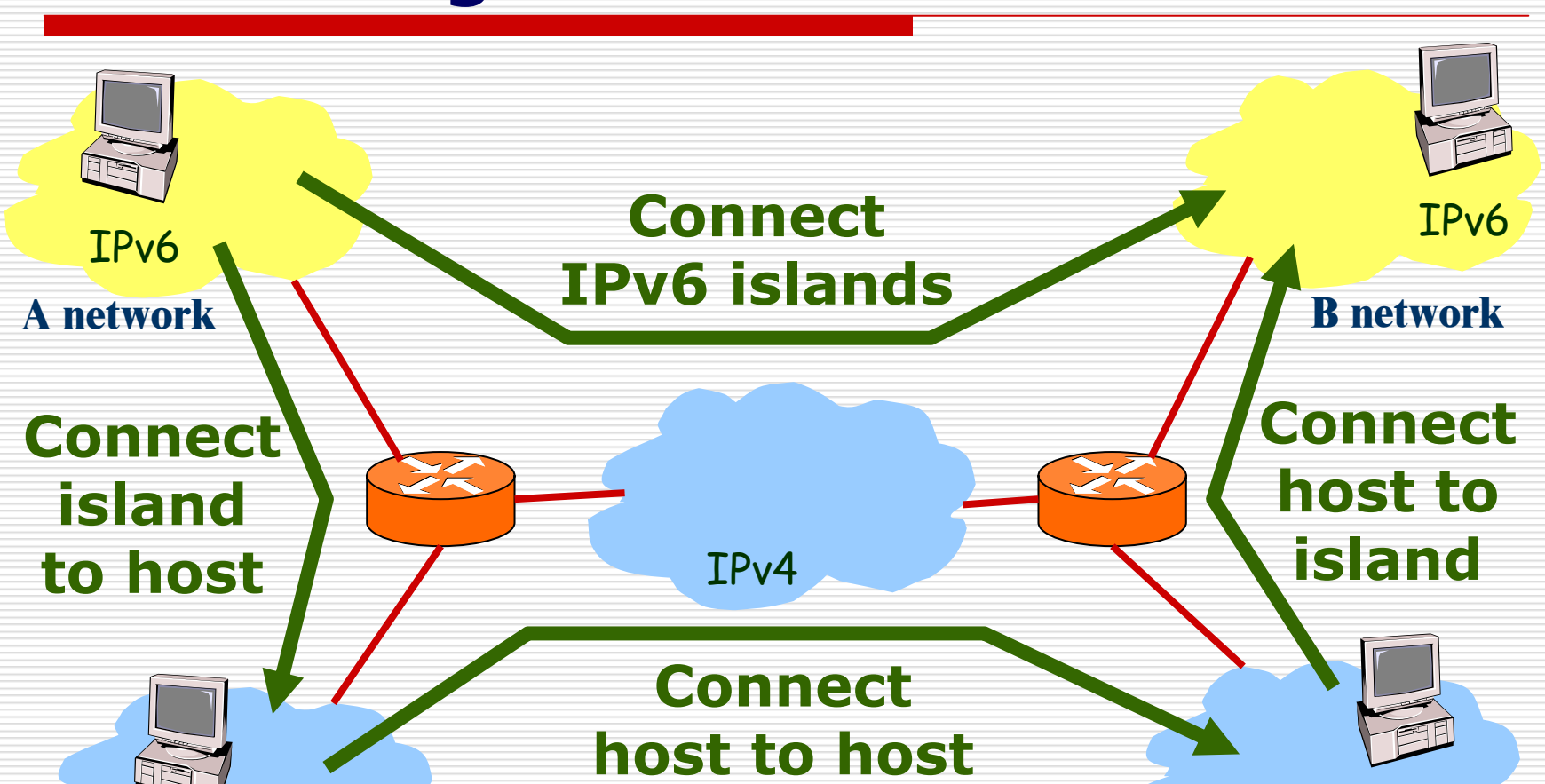


- Technique extensively used
  - MBONE
  - Multiprotocol (IPX, Appletalk) over IP
  - IP mobility

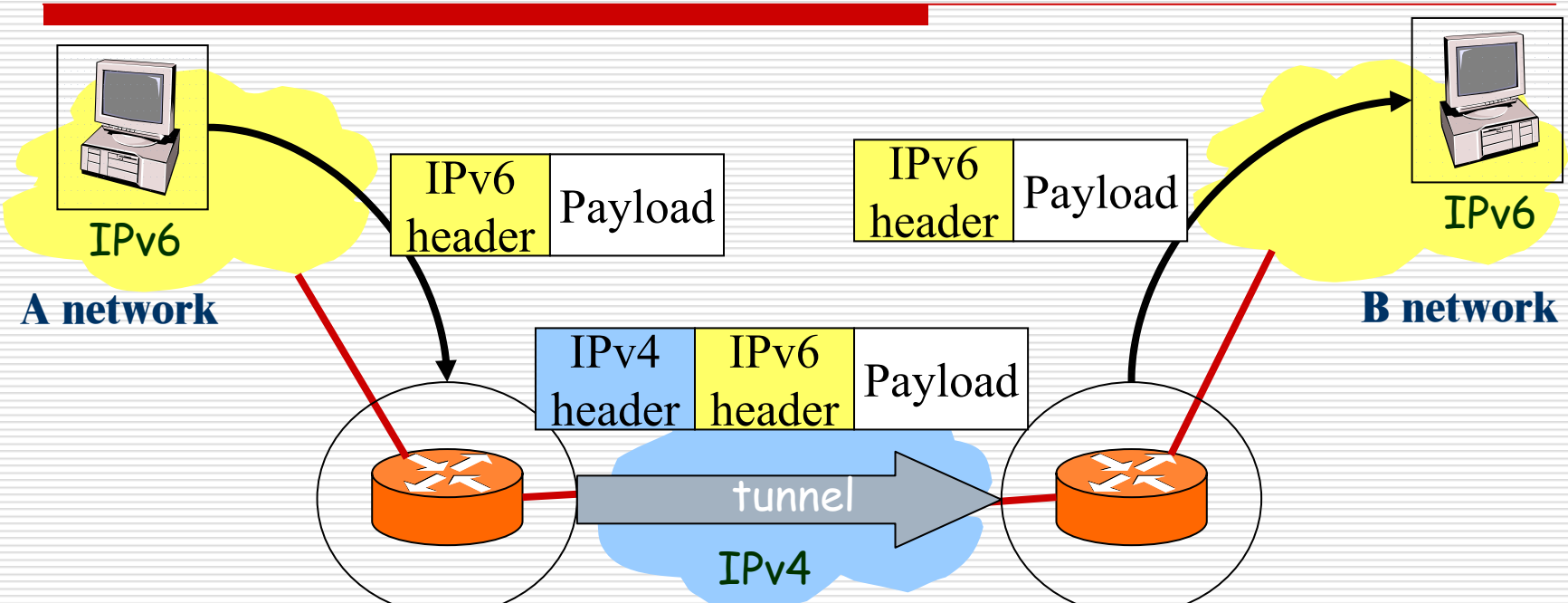
Tunneling

- Configured tunnel
  - Tunnel to a router
  - Destination host any IPv6 address
  - Types:
    - *Router-to-router*
    - *Host-to-router*
- Automatic tunnel
  - Tunnel to a concrete host
  - Destination host an IPv4-compatible address
  - Types:
    - *Host-to-host*
    - *Router-to-host*
- Multicast tunneling

Tunneling scenarios



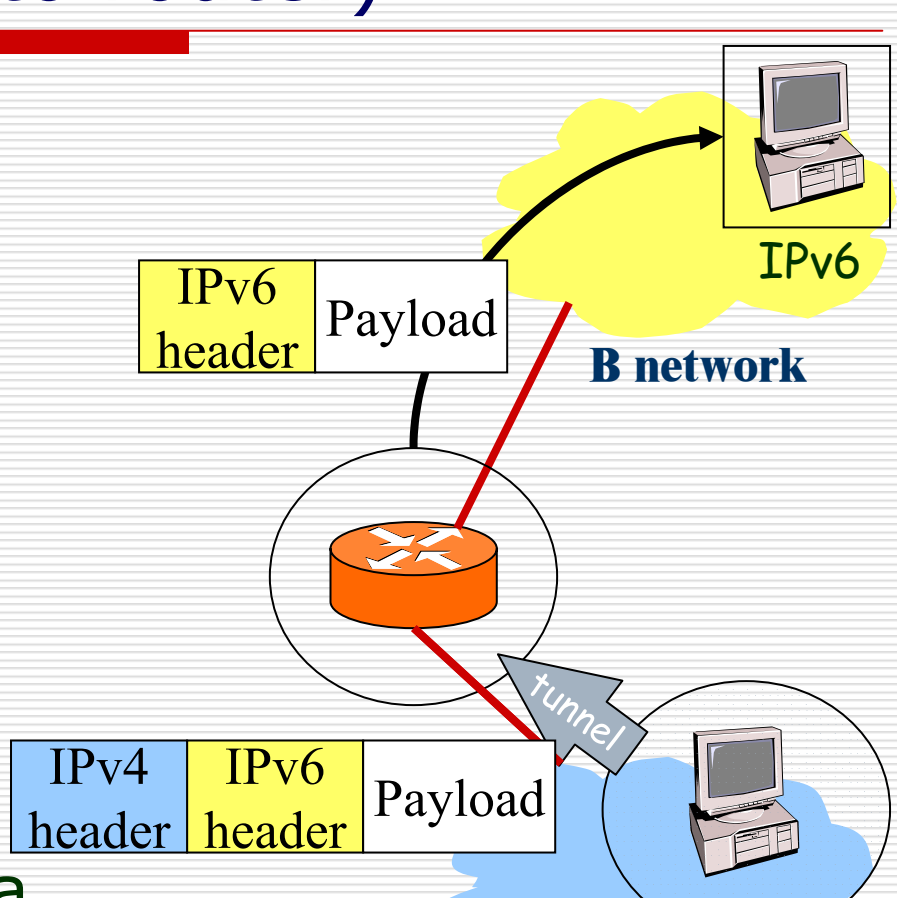
Tunneling (router to router)



- Configured tunnel
- IPv6/IPv4 routers interconnected by an IPv4 infrastructure
- Hosts are IPv6 only

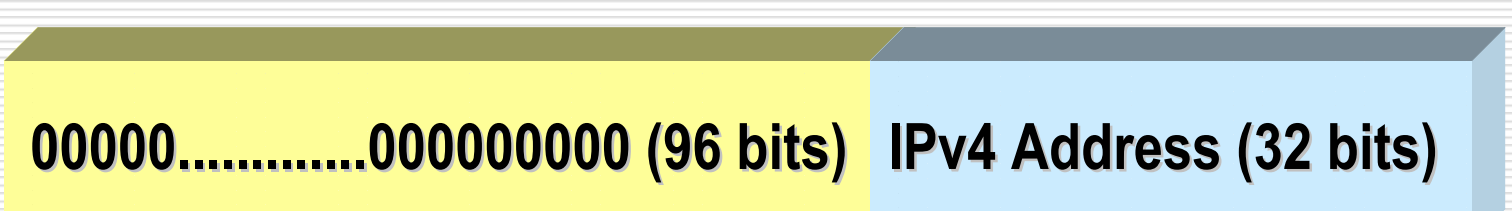
Tunneling (host to router)

- Configured tunnel
  - It can be setup as automatic tunnel
- Hosts
  - From: dual stack hosts
  - To: IPv6 only hosts
- Hosts tunnel IPv6 packets to an intermediary IPv6/IPv4 router via an IPv4 infrastructure



Automatic tunnels

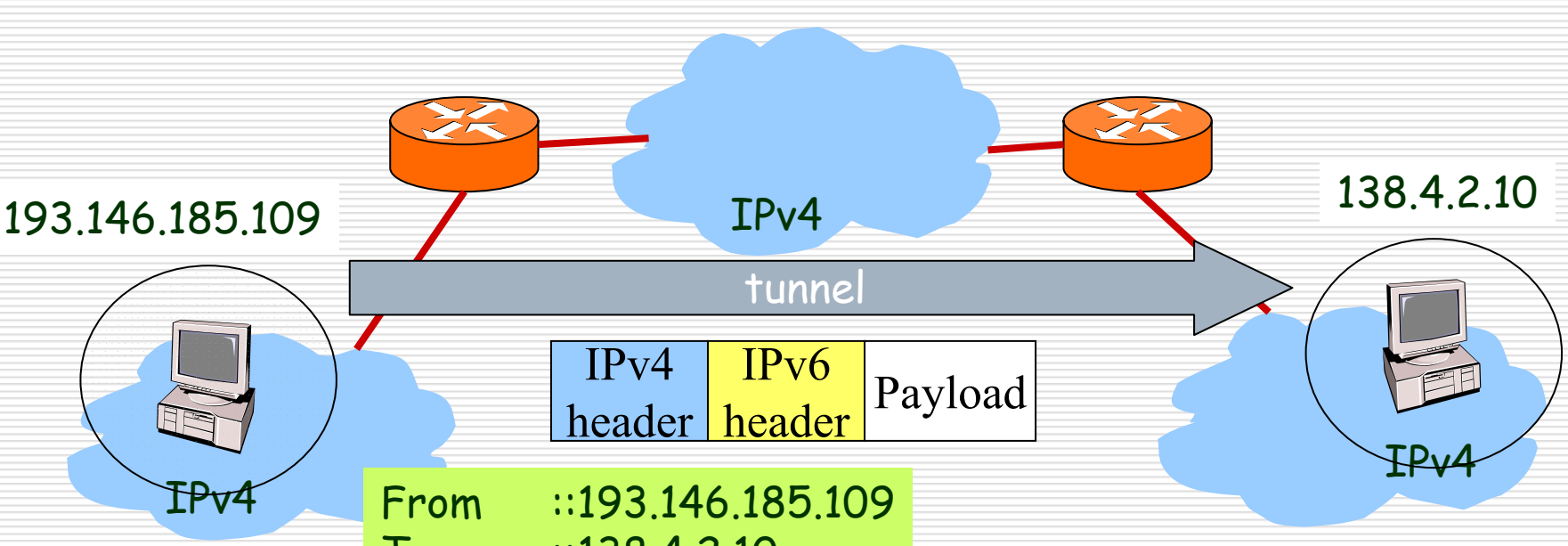
- How to obtain IPv6 address of an IPv4 end point?
- We can use IPv4-compatible IPv6 address



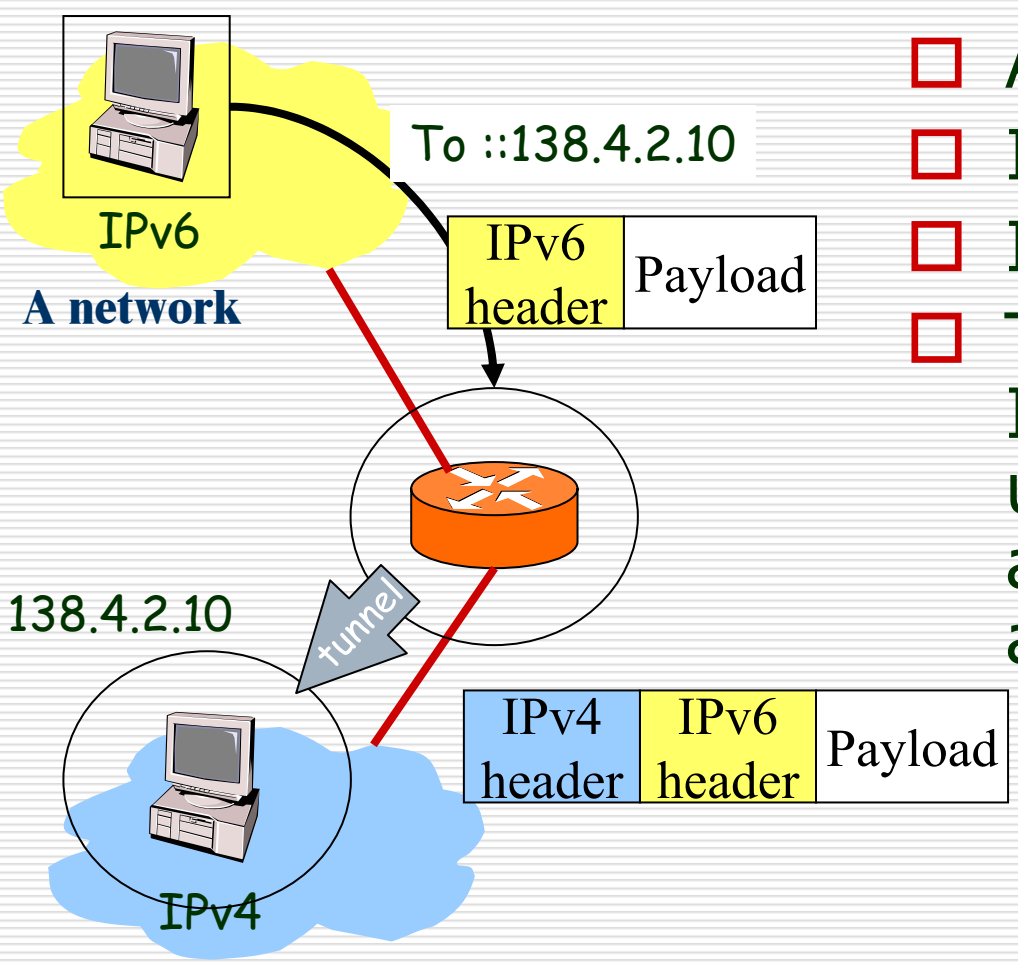
- Example: 0:0:0:0:0:0:138.4.2.10 or ::138.4.2.10
- IPv4-compatible addresses are assigned exclusively to nodes that support automatic tunneling

Tunneling (host to host)

- Automatic tunnel
- Dual stack IPv4/IPv6 hosts
- Interconnected by an IPv4 infrastructure



Tunneling (router to host)



- Automatic tunnel
- IPv4/IPv6 routers
- IPv4/IPv6 hosts
- The router translate IPv6 address host can use tunnels to reach an IPv4/IPv6 host via an IPv4 infrastructure

Translators

- Network level translators
  - SIIT (Stateless IP/ICMP Translator)
  - NAT-PT (Network Address Translation-Protocol Translation)
  - BIS (Bump in the stack)
    - MBIS (Multicast extensions to BIS)
- Transport level translators
  - Transport Relay Translator (TRT)
- Application level translators
  - BIA (Bump in the API)

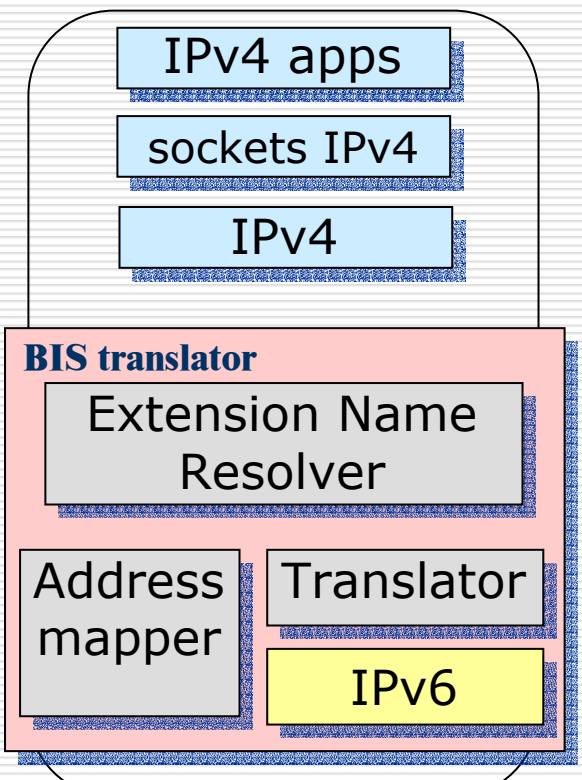
NAT-PT (Network Address Translation – Protocol Translation)

- It is a gateway, which translates an IPv6 into an equivalent IPv4 address and vice versa.
- The connection state is established when
  - First data packet traverses NAT-PT box
    - If IPv6 domain starts communication
  - First DNS response traverses NAT-PT box
    - IF IPv4 domain starts communication
- Heavy mechanism for each packet
  - Address state
  - Protocol translation
- Single failure point
- Problems when the application transfers IP addresses

Bump in the stack(BIS) technique

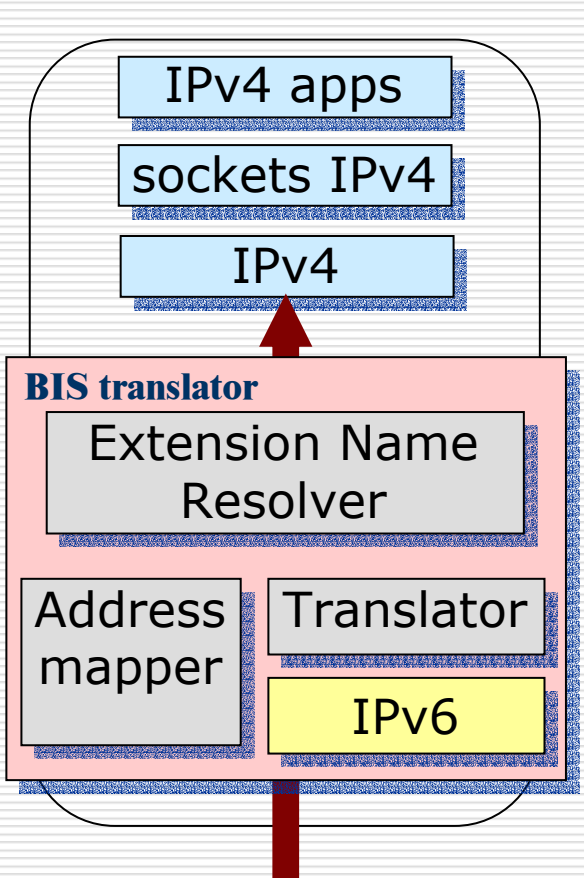
- BIS allows dual networks hosts to communicate with IPv6 hosts using existing IPv4 applications.
  - With IPv4 hosts using standard IPv4 stack
  - With IPv6 hosts using special translator stack
- IPv4 only applications
  - uses IPv4 only sockets API.
  - manages only IPv4 address
- It translates IPv4 packages into IPv6 ones and vice versa using the IP conversion mechanism SIIT.
- BIS can co-exist with other translators.

BIS dual stack structure



- Extension Name Resolver
  - Redefine DNS requests
  - Looks for 'A' and 'AAAA' records
  - Detect IPv6 addresses
- Address Mapper
  - Maintains a database Translate IPv6 into IPv4 addresses
- Translator
  - Translate IPv4 packages into IPv6 ones.

BIS operation (communication with an IPv4 host)

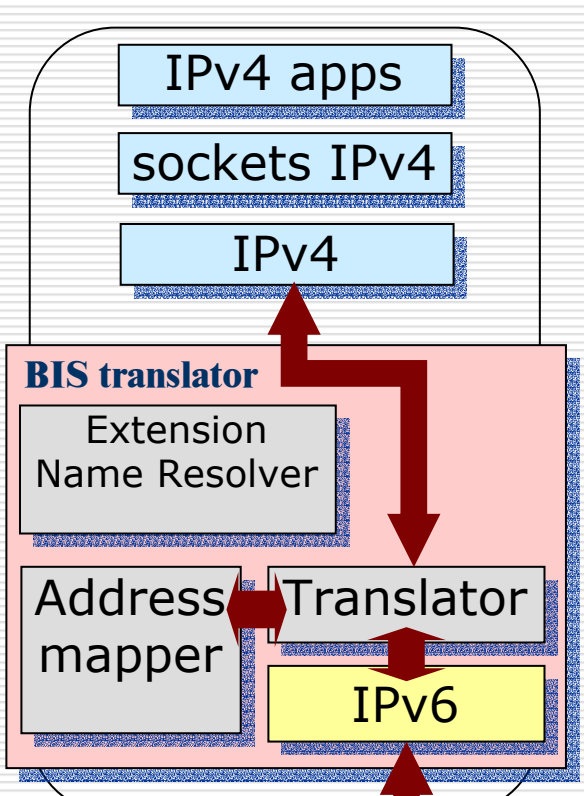


1. Application sends a query to resolve an 'A' record for the target host.
2. BIS **Extension Name Resolver** creates another query
   - 'A' record
   - 'AAAA' record
3. DNS returns an 'A' record
4. IPv4 stack is used without any translation

Host IPv4

IPv4 network

BIS operation (from BIS node to IPv6 host)

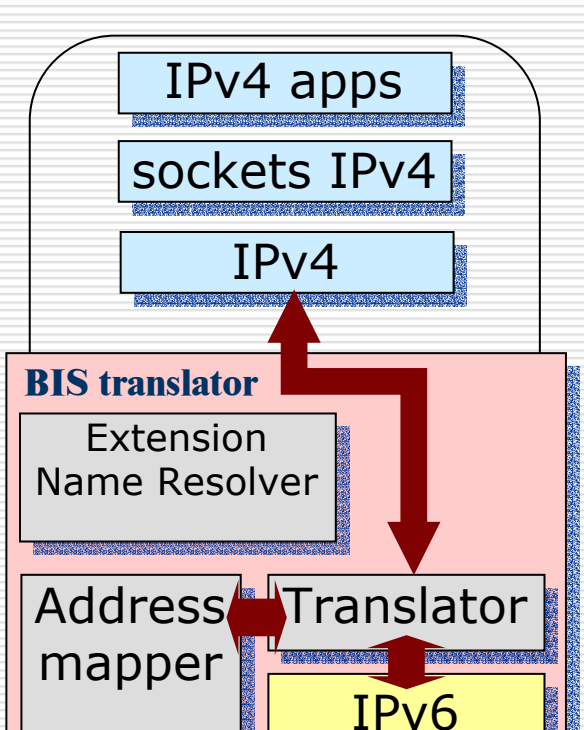


1. Application sends a query to resolve an 'A' record and BIS creates a query with both 'A' and 'AAAA'
2. DNS returns 'AAAA' record
3. BIS **Address Mapper** maps
   - IPv6 address into IPv4 one
4. Application sends an IPv4 packet
5. **Translator** transforms IPv4 into IPv6 packet and sends to the network
   - Using mapped addresses

Host
IPv6
(server)

IPv6 network

BIS operation (from IPv6 node to BIS node)



1. BIS node receives an IPv6 packet
2. The translator tries to produce an IPv4 packet
   - Requesting mapped entries
3. If IPv6 source address is not found
   - Selects an IPv4 address from the spool
4. Translator tosses IPv4 packet up to the application

Host
IPv6
(client)

IPv6 network

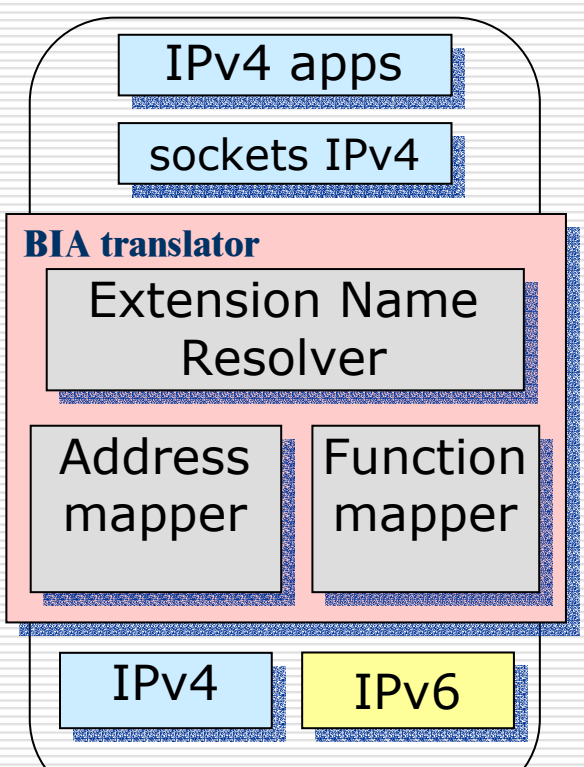
BIS limitations

- It has the same limitations of other header translation methods
  - Single failure point
  - Problems when the application transfers IP addresses
- Implementation is dependent upon network interface driver.
- This method is invalid when
  - Application manages network information.
  - Communication is not unicast
  - Application uses IPv4 options
  - Received IPv6 packets with options

Bump in the API (BIA) technique

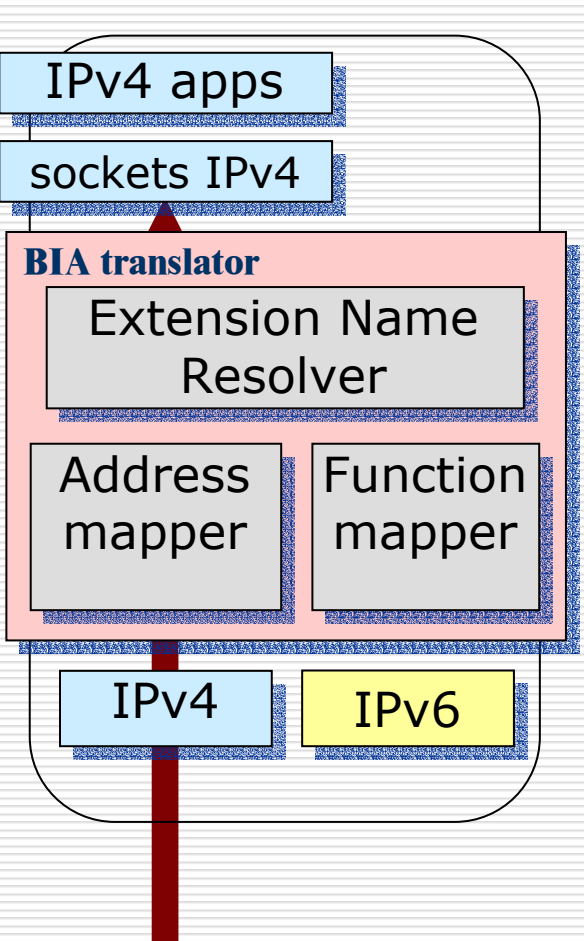
- It allows IPv4 applications communicate with IPv6 hosts
  - It works on dual stack hosts.
  - It works on IPv6 only hosts.
- Inserts an API translator between the socket API module and the TCP/IP module
- Translation is possible without header translation

BIA dual stack structure



- Extension Name Resolver
  - Redefine DNS requests
  - Looks for 'A' and 'AAAA' records
  - Detect IPv6 addresses
- Address Mapper
  - Maintains a database
  - Translate IPv6 into IPv4 addresses
- Function Mapper
  - Translate IPv4 sockets function calls into IPv6 ones.

BIA operation (from BIA node to IPv6 host)

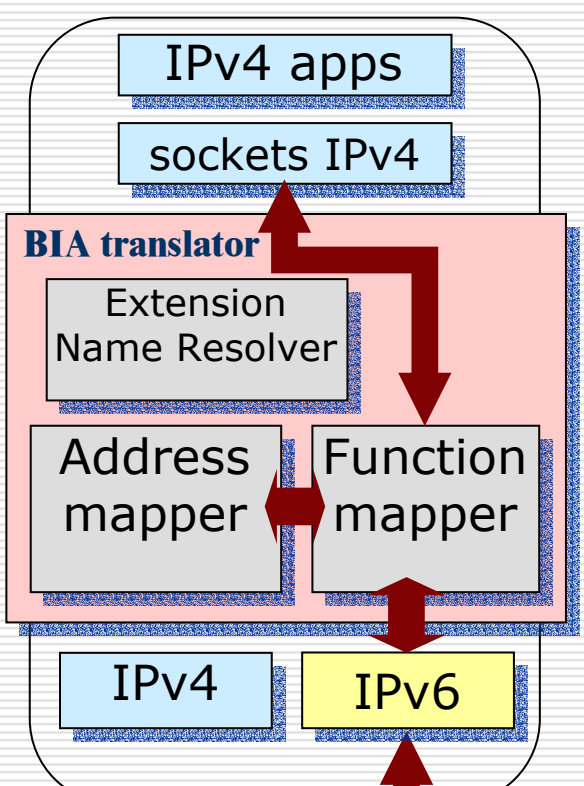


1. Application sends a query to resolve an 'A' record for the target host.
2. BIA **Extension Name Resolver** creates another query
   - 'A' record
   - 'AAAA' record
3. DNS returns an 'A' record
4. IPv4 stack is used without any translation

Host IPv4

IPv4 network

BIA operation (communication with an IPv6 host)

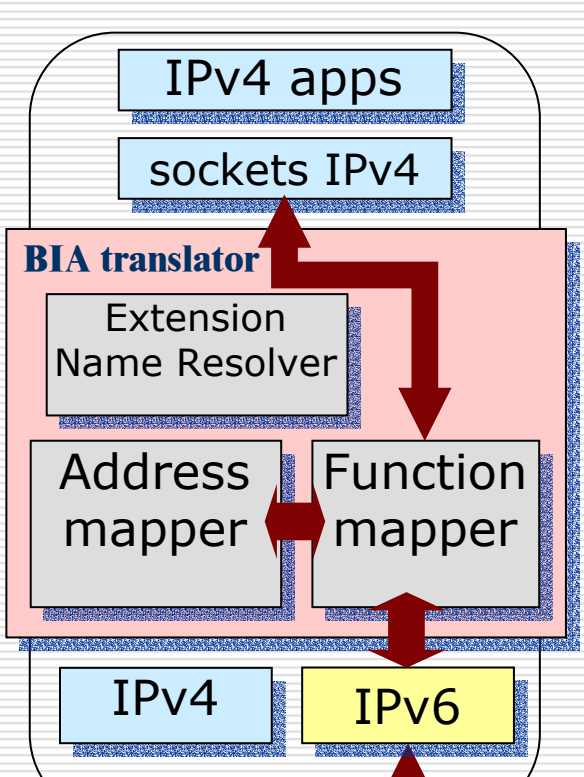


1. Application sends a query to resolve an 'A' record and BIS creates a query with both 'A' and 'AAAA'
2. DNS returns 'AAAA' record
3. BIA **Address Mapper** maps
   - IPv6 address into IPv4 one
4. Application sends an IPv4 packet
5. **Function Mapper** translates IPv4 sockets function calls into IPv6 ones
   - Using mapped addresses

Host IPv6 (server)

IPv6 network

BIA operation (from IPv6 node to BIA node)



1. BIA Function mapper receive a IPv6 packet
2. Function mapper request IPv4 addresses to the Address Mapper
   - Local address
   - Remote address
3. Invoke IPv4 socket function to connect with application

Host
IPv6
(client)

IPv6 network

BIA limitations

- It has the same limitations of other header translation methods
  - Single failure point
  - Problems when the application transfers IP addresses
- This method is invalid when
  - Application manages network information.
  - Application uses IPv4 options
  - Received IPv6 packets with options

Applications porting guidelines

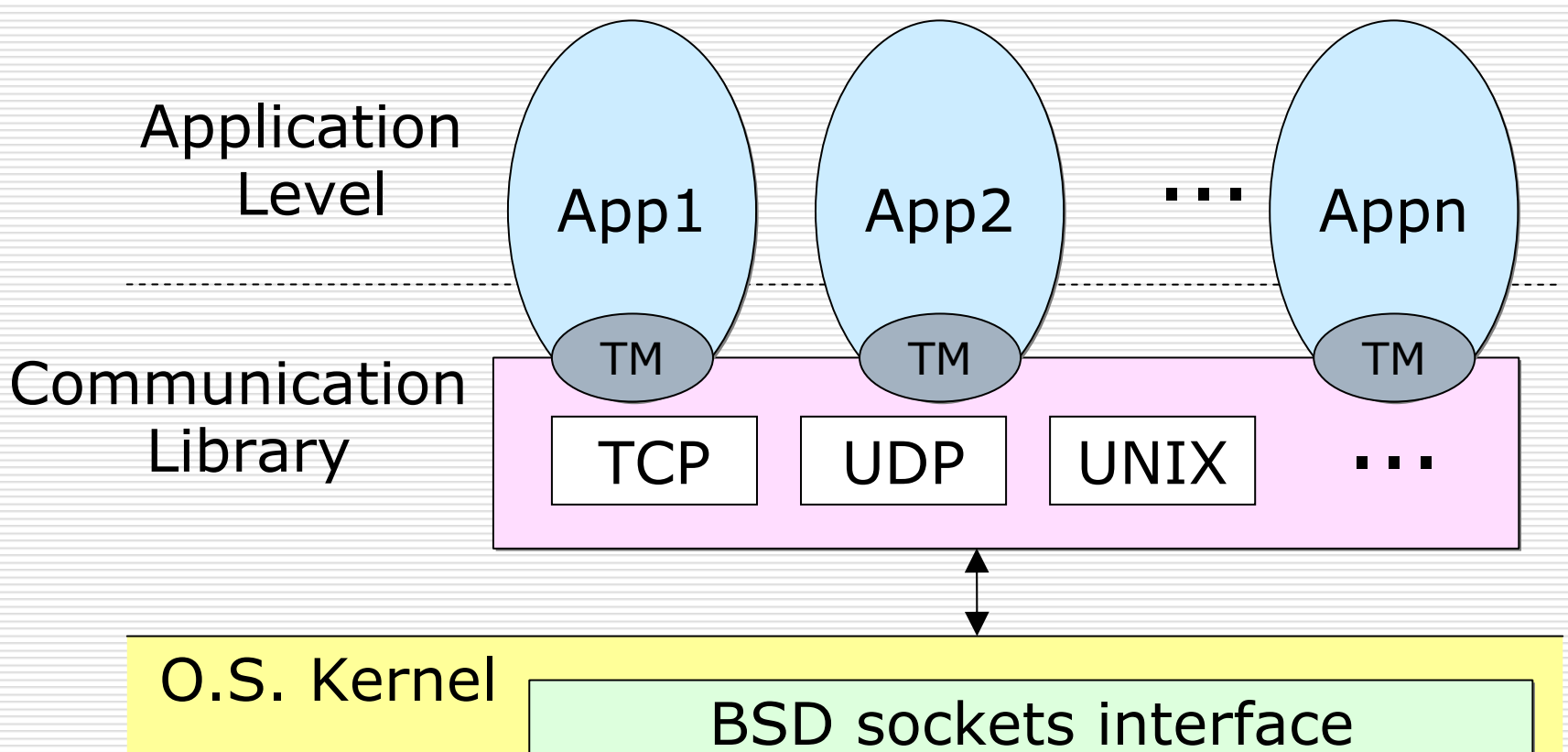
1. Use existing IPv4 only application
   - Using translators
   - Valid only with limitations
2. Porting existing application
   - Applicable only if source code is available
   - Porting communications libraries
     - Example: Java net library
3. Developing new application
   - Independent of IP addresses
   - Dependent of IP addresses
     - Not recommended
     - Developing IPv4/IPv6 dual code

Agenda

- Transition to IPv6 is only a network issue
- Transition to IPv6 implies application code porting
  1. Analyze existing programs
  2. Rewrite source code
- Describe two code porting exercises

Analyzing existing programs

- Application Transport Module (TM).



Porting to IPv6

- Communications library
- Other application modules

(with IP address dependencies)

1. IP address parsers.
2. Use of special addresses.
3. Local IP address selection.
4. ADU Fragmentation.
5. Register systems based on IP addresses.

1. IP address parsers

- Compliant with IPv4/IPv6 address format.
  - Separator: "." for IPv4 and ":" for IPv6.
- Ambiguity separating the address and the service port number.
  - 138.4.2.10:3333
- Recommendations:
  - Use FQDN.
  - Use literal IP address (RFC 2373).

    http://[2001:720:1500:1::a100]:80/index.html

2. Use of special addresses

Special addresses are found hard coded within the source instead of symbolic names.

Symbolic Name (IPv4/IPv6)	IPv4 Addr	IPv6 Addr
INADDR_ANY/IN6ADRR_ANY_INIT	0.0.0.0	::
INADDR_LOOPBACK/ IN6_ADDR_LOOPBACK_INIT	127.0.0.1	::1
INADDR_BROADCAST	255.255.255.255	not exist

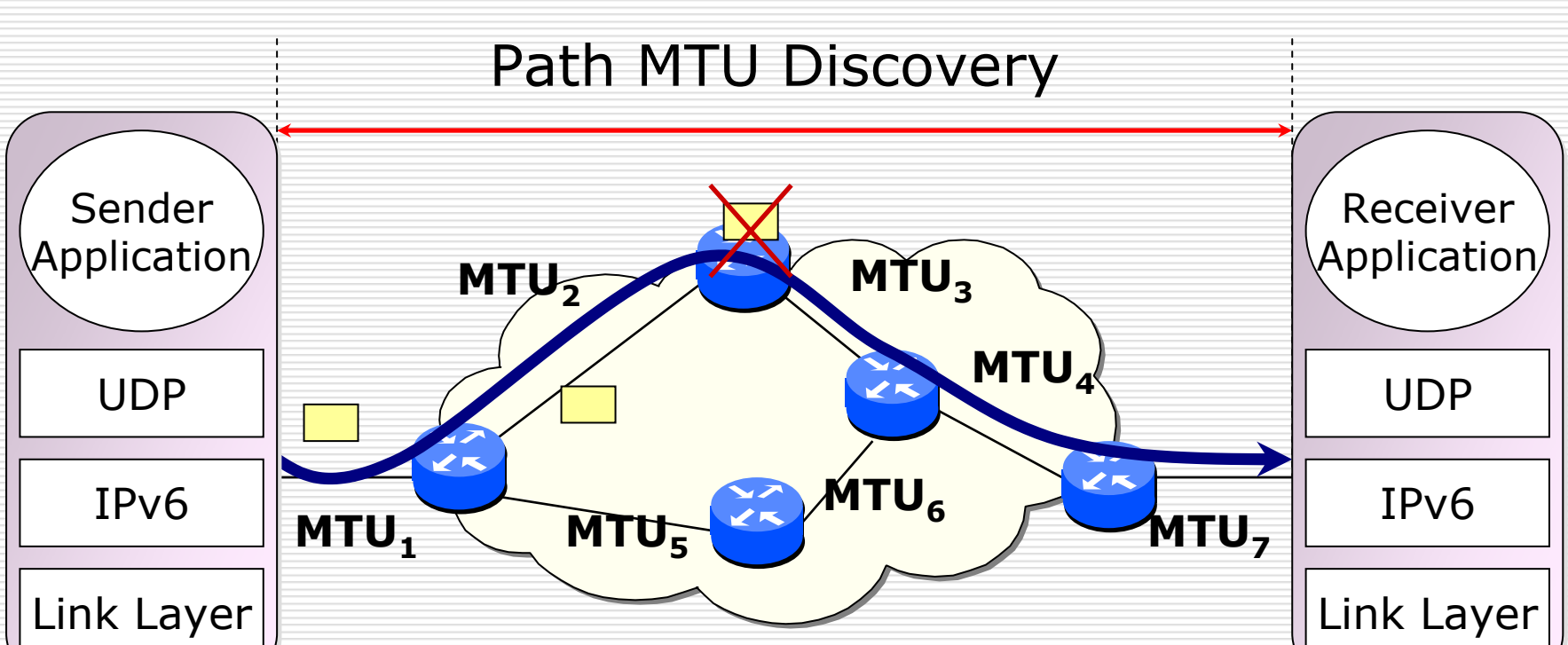
3. Local IP address selection

- Source address selection (multihoming):
  - Unspecified address

    (Default Address Selection).
  - Work in progress:

    Draft-ietf-ipv6-default-addr-select.txt
- Destination address selection:
  - Resolver returns a set of candidates
  - A selection algorithm is needed
    - based on the policy of “best choice”.

4. ADU fragmentation



$MTU_1 > MTU_2 >$Packet Size $> MTU_3$

5. Register Systems using IP addresses

- Group communication is often related to a group membership based on participant registry system.
- Problem:
  - IP address can change over time.
- Solution:
  - FQDN.
  - Refresh IP address value.

Porting guidelines

1. Porting the application networking part to manage IPv6 addresses.

   Using socket interface:
   - RFC 2553: Basic socket interface extensions for IPv6.
   - RFC 2292: Advanced sockets API for IPv6.
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.
   - etc.

BSD socket interface extensions

IPv6 addresses are 128 bit long.

Changes in the application networking part

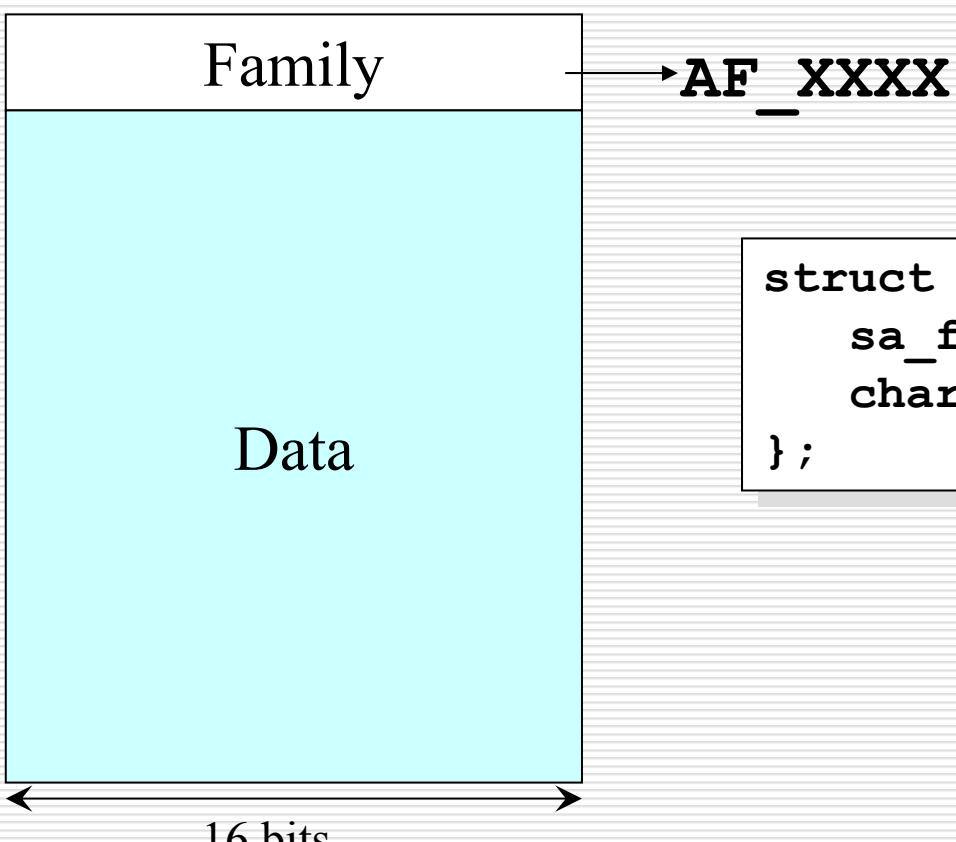
1. Data structures:
   Socket address struct

2. Conversion functions between:
   - String and bynary representation
   - Name and address

3. Socket calls

Generic socket address

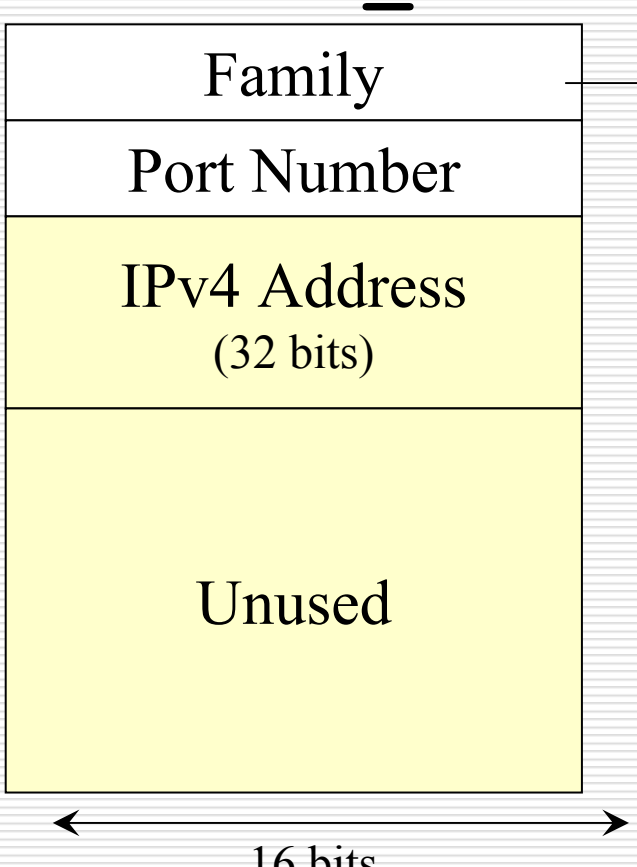
sockaddr




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

IPv4 socket address structure

sockaddr_in

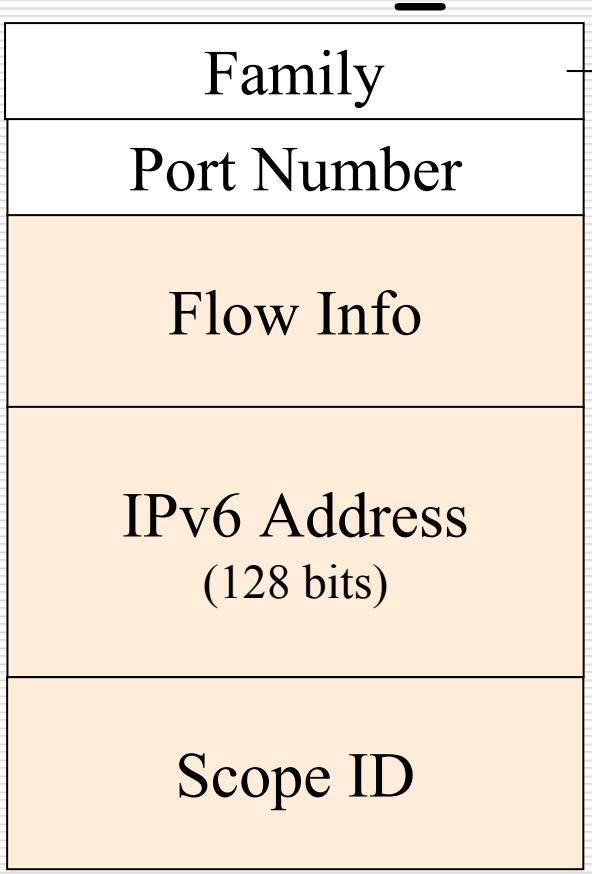


AF_INET

```
struct sockaddr_in {
   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 structure

sockaddr_in6

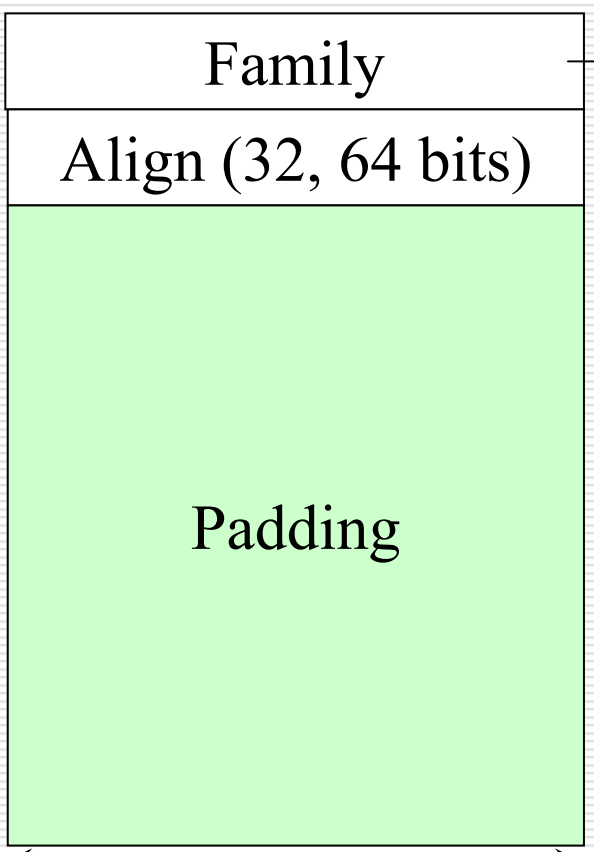


AF_INET6

```
struct sockaddr_in6 {
    sa_family_t         sin6_family;
    in_port_t           sin6_port;
    uint32_t            sin6_flowinfo;
    struct in6_addr     sin6_addr;
    uint32_t            sin6_scope_id;
};

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

Protocol independent structure

sockaddr_storage



→ **AF_XXXX**

```
struct sockaddr_storage {
    sa_family_t       sin6_family;
    __ss_aligntype __ss_align;
    char __ss_padding[_SS_PADSIZE];
};
```

BSD socket interface extensions

IPv6 addresses are 128 bit long.

Changes in the application networking part

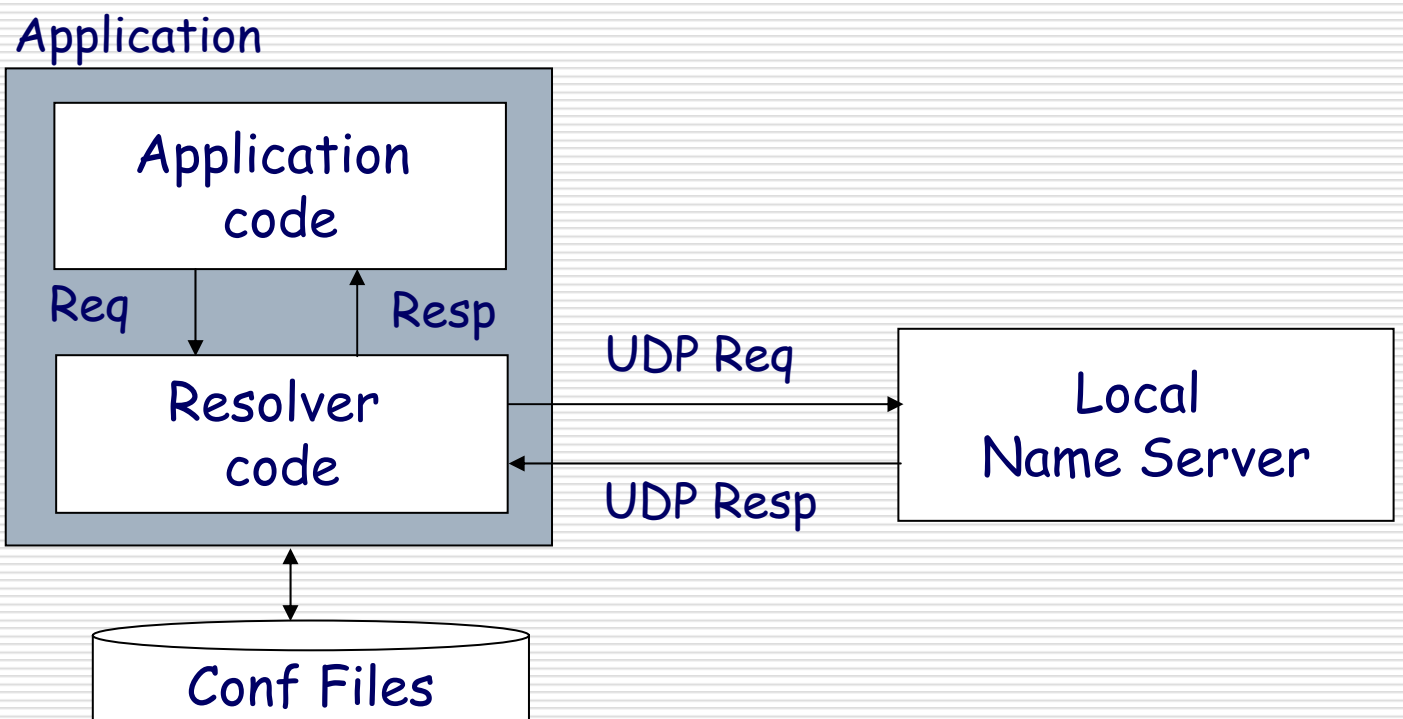
1. Data structures:
   Socket address struct

2. Conversion functions between:
   - Hostname and address
   - String and bynary representation

3. Socket calls

Conversion functions

- RESOLVER: returns IP address structure.



Conversion: hostname & IP addr

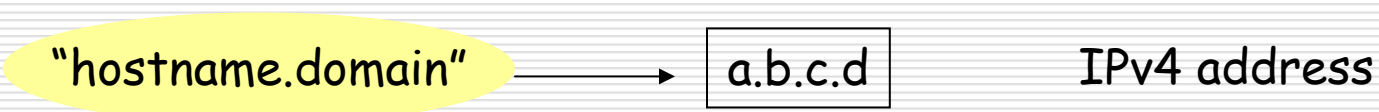
IPv4 only	IPv4 & IPv6
`gethostbyname()`	`gethostbyname2()`
	`getaddrinfo()`
`gethostbyaddr()`	`getnameinfo()`

Protocol Independent Functions

Gethostbyname

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

* Query for a DNS A record

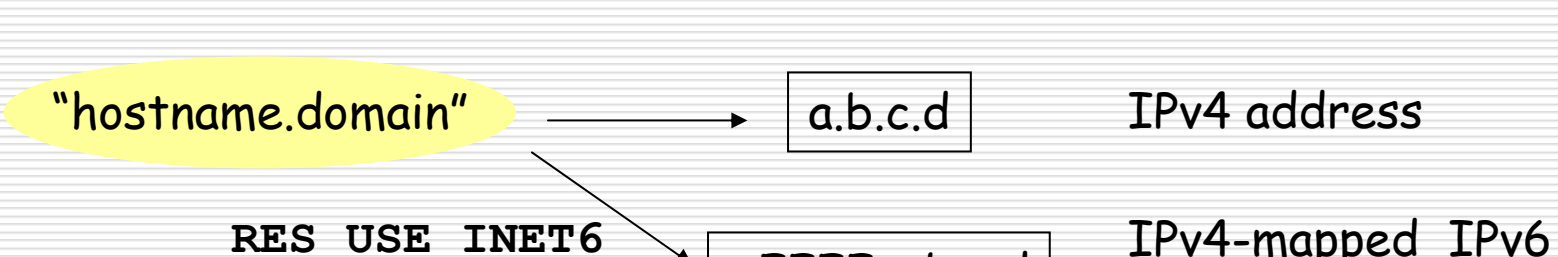


* Query for a DNS AAAA record, **RES_USE_INET6** resolver option is required:

Gethostbyname2

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

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

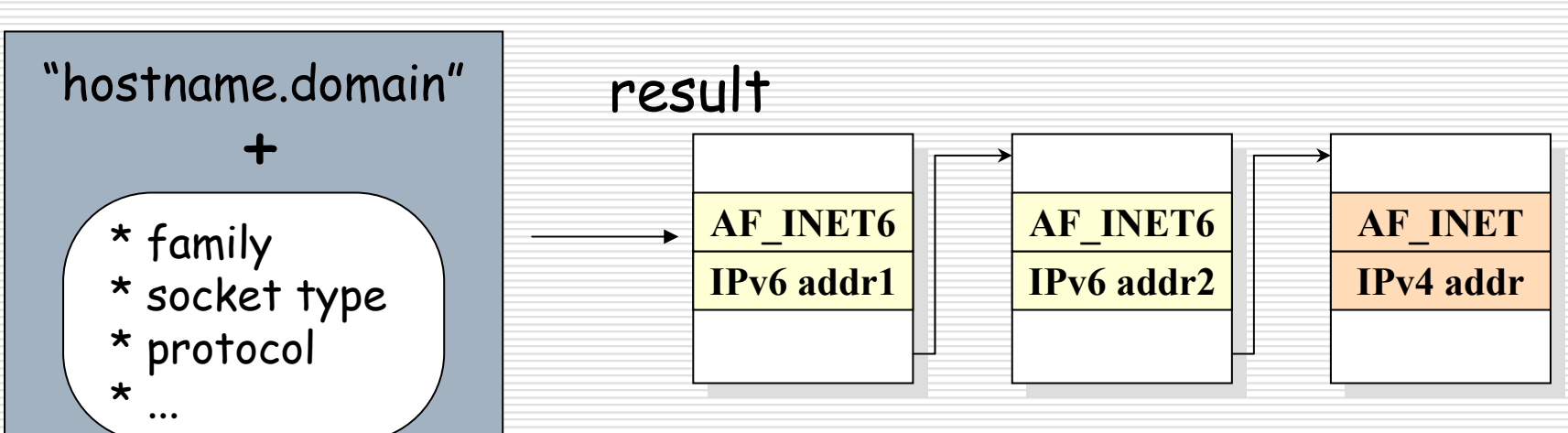


* Query for a DNS *AAAA* record, **family=AF_INET6** is required:

Getaddrinfo

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

* Protocol independent function.



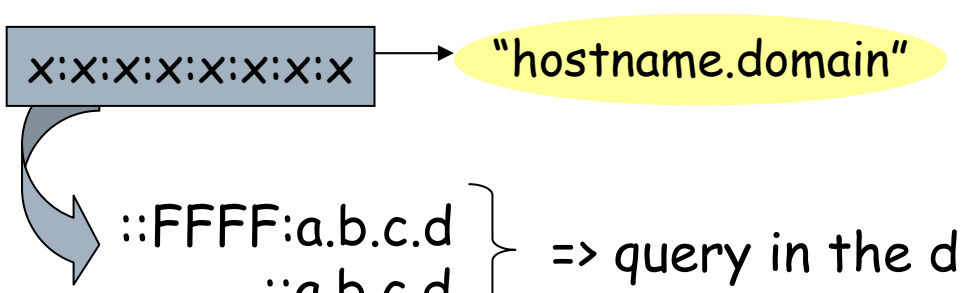
Gethostbyaddr

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

* From IPv4 address:

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

* From IPv6 address:



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

Getnameinfo

```
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)

Conversion: string & binary

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

	IPv4 only	IPv4 & IPv6
String -> Binary	`inet_aton()` `inet_addr()`	`inet_pton()`
Binary -> String	`inet_ntoa()`	`inet_ntop()`

Conversion: string & binary

1. String to 32-bit network byte ordered

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

IPv4 only

```
int    inet_ntoa (struct in_addr inaddr
                  const char *strptr);
```

IPv4 & IPv6

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

BSD socket interface extensions

IPv6 addresses are 128 bit long.

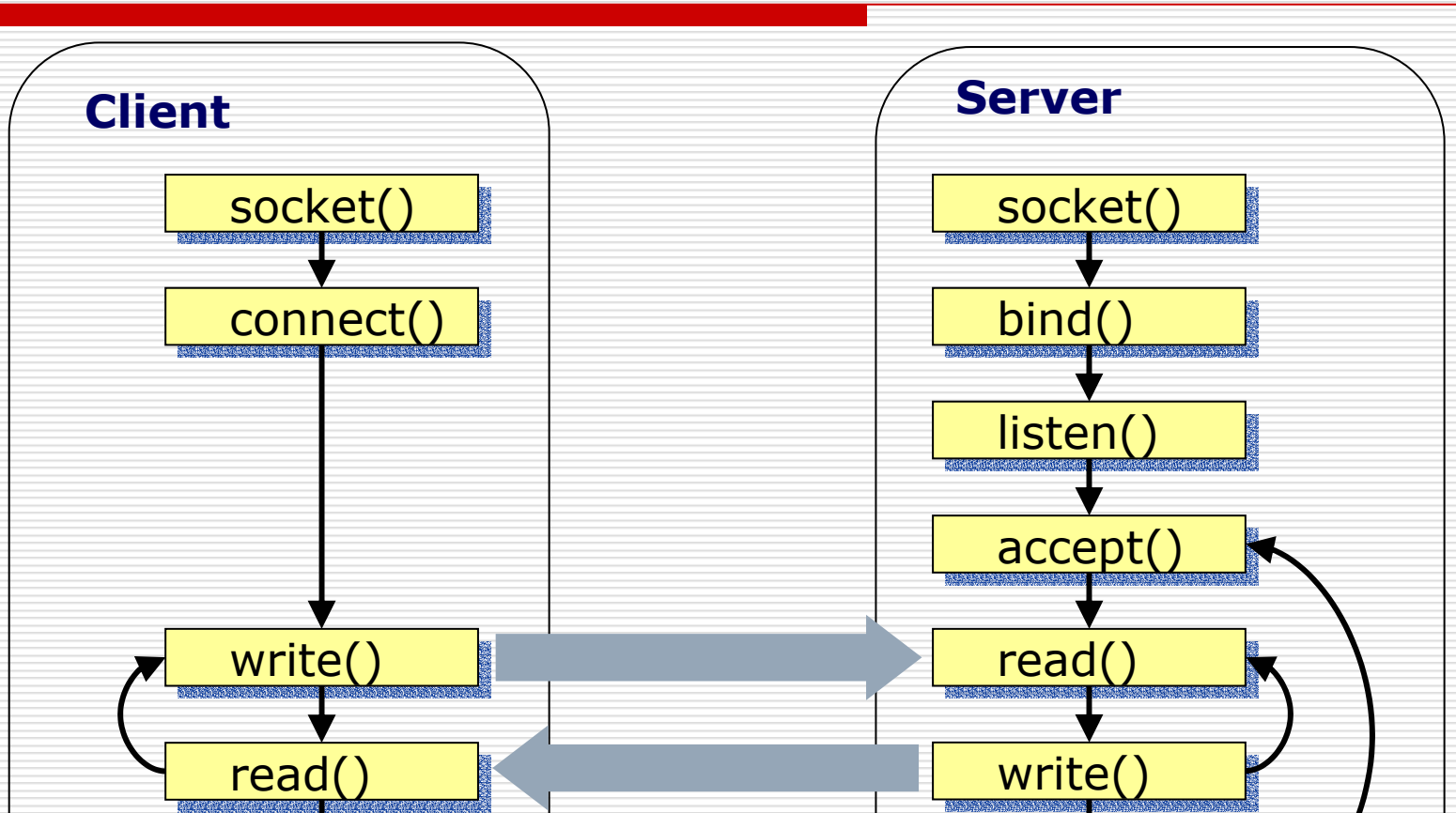
Changes in the application networking part

1. Data structures:
   Socket address struct

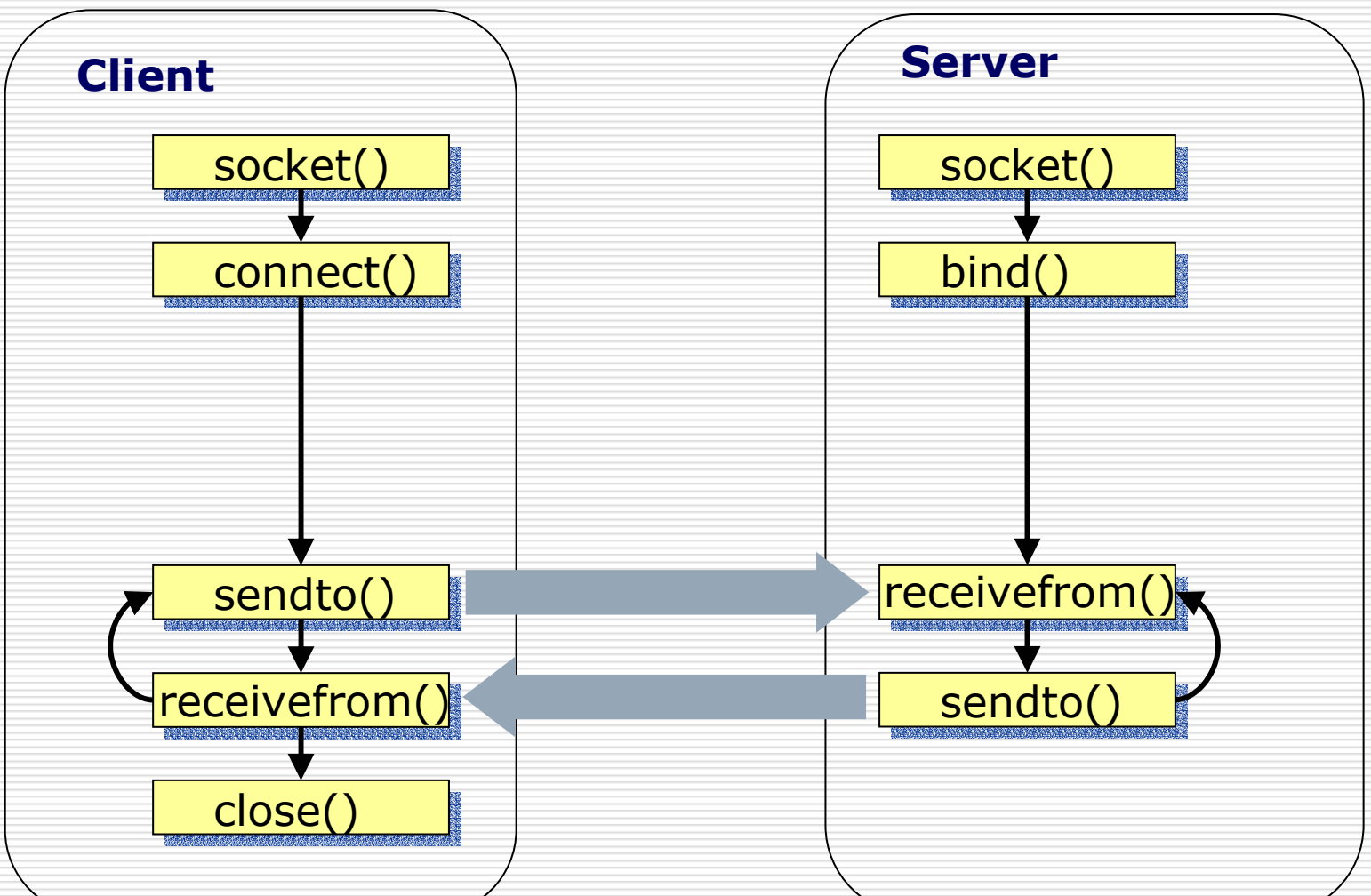
2. Conversion functions between:
   - Hostname and address
   - String and bynary representation

3. Socket calls

Socket calls (TCP)



Socket calls (UDP)



Same socket calls IPv4 & 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

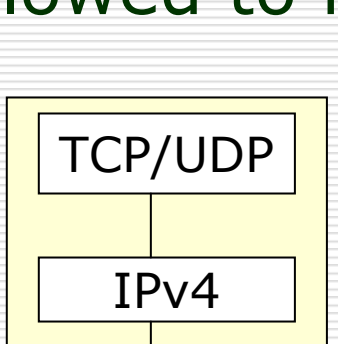
* A pointer to a **sockaddr** struct and its length are required in some calls:

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

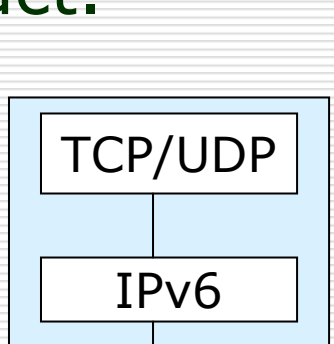
IPv6:
- Casting **(sockaddr_in6 *) to (sockaddr *)**
- **Length => sizeof(sockaddr_in6)**

IPv4/IPv6 Interoperability

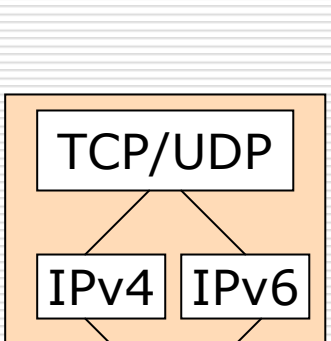
- Dual stack will be the mechanism used during transition period.
- Not all combinations between IPv4 o IPv6 only-nodes and dual stack nodes are allowed to interact.



IPv4-only node



IPv6-only node



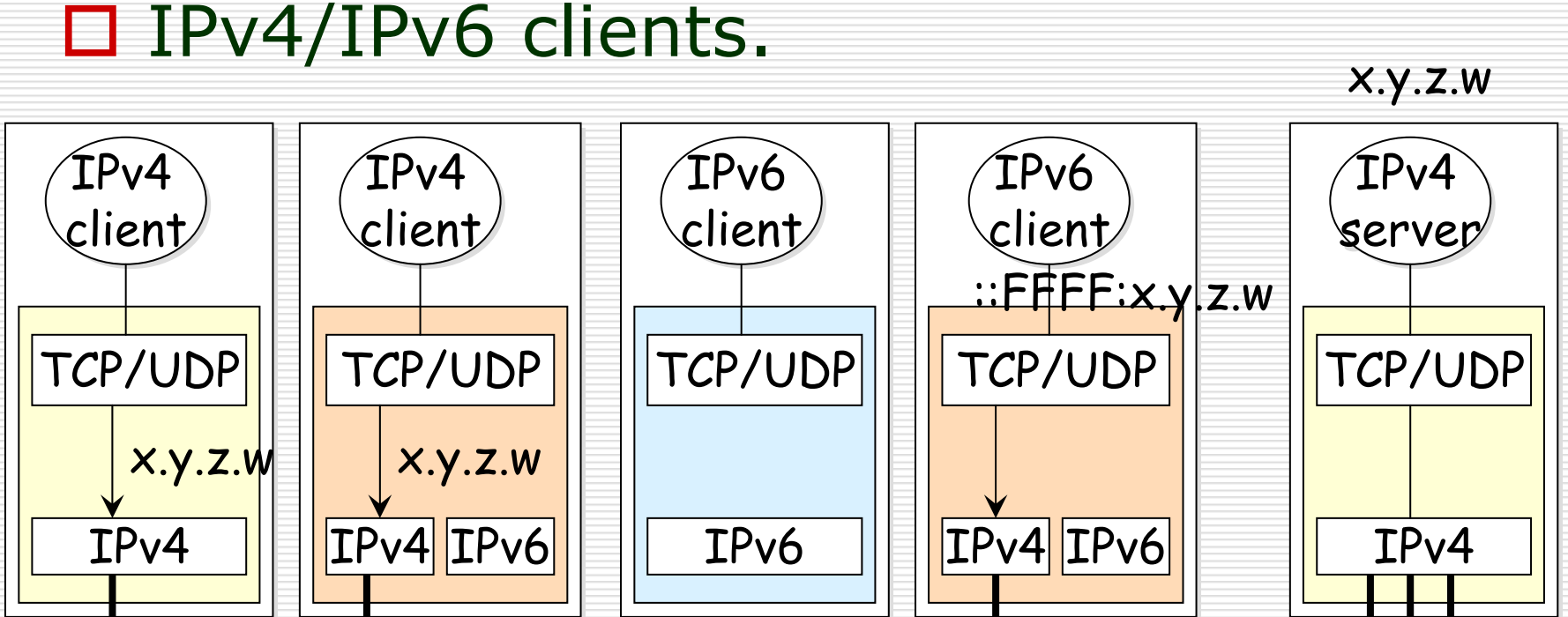
Dual stack node

Interoperability cases

- IPv4 server at IPv4-only node.
- IPv6 server at IPv6-only node.
- IPv4 server at dual stack node.
- IPv6 server at dual stack node.
- IPv6-only server at dual stack node.

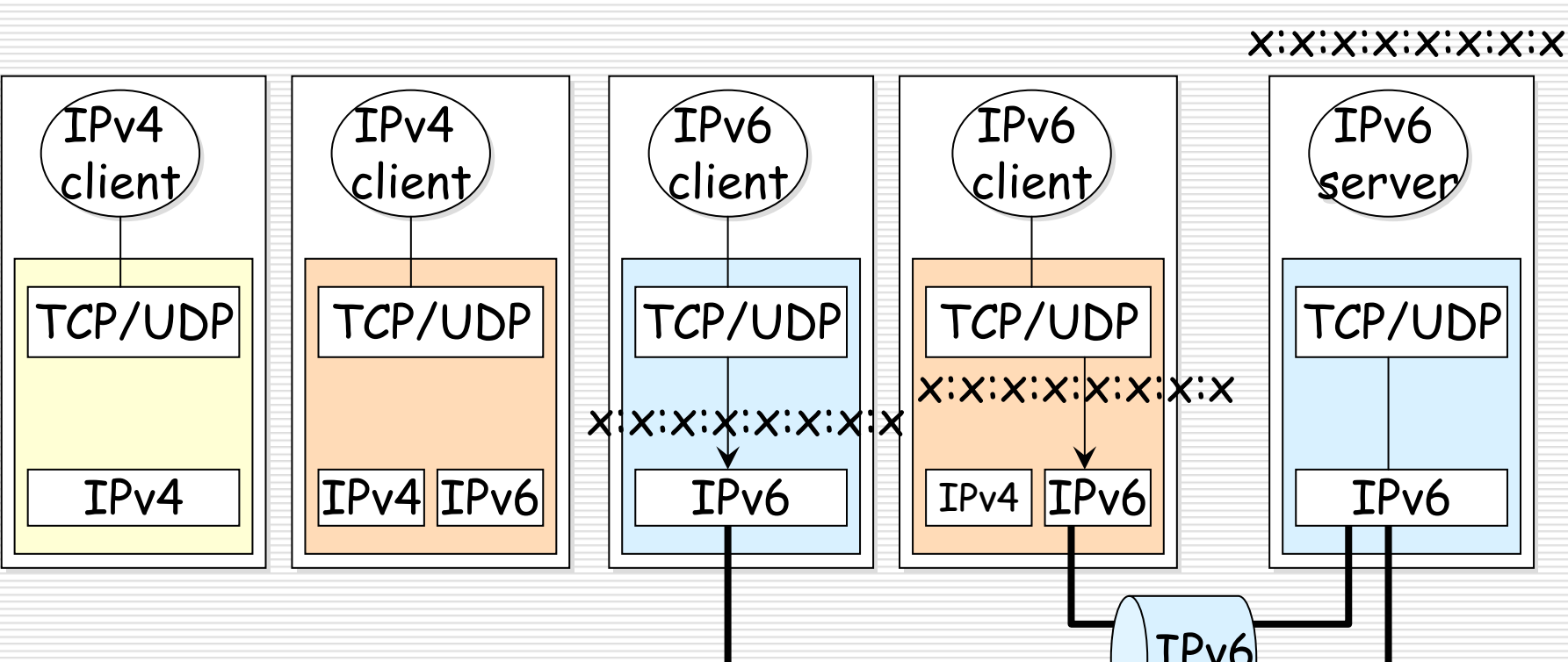
1.IPv4 server at IPv4-only node

- IPv4/IPv6 clients.



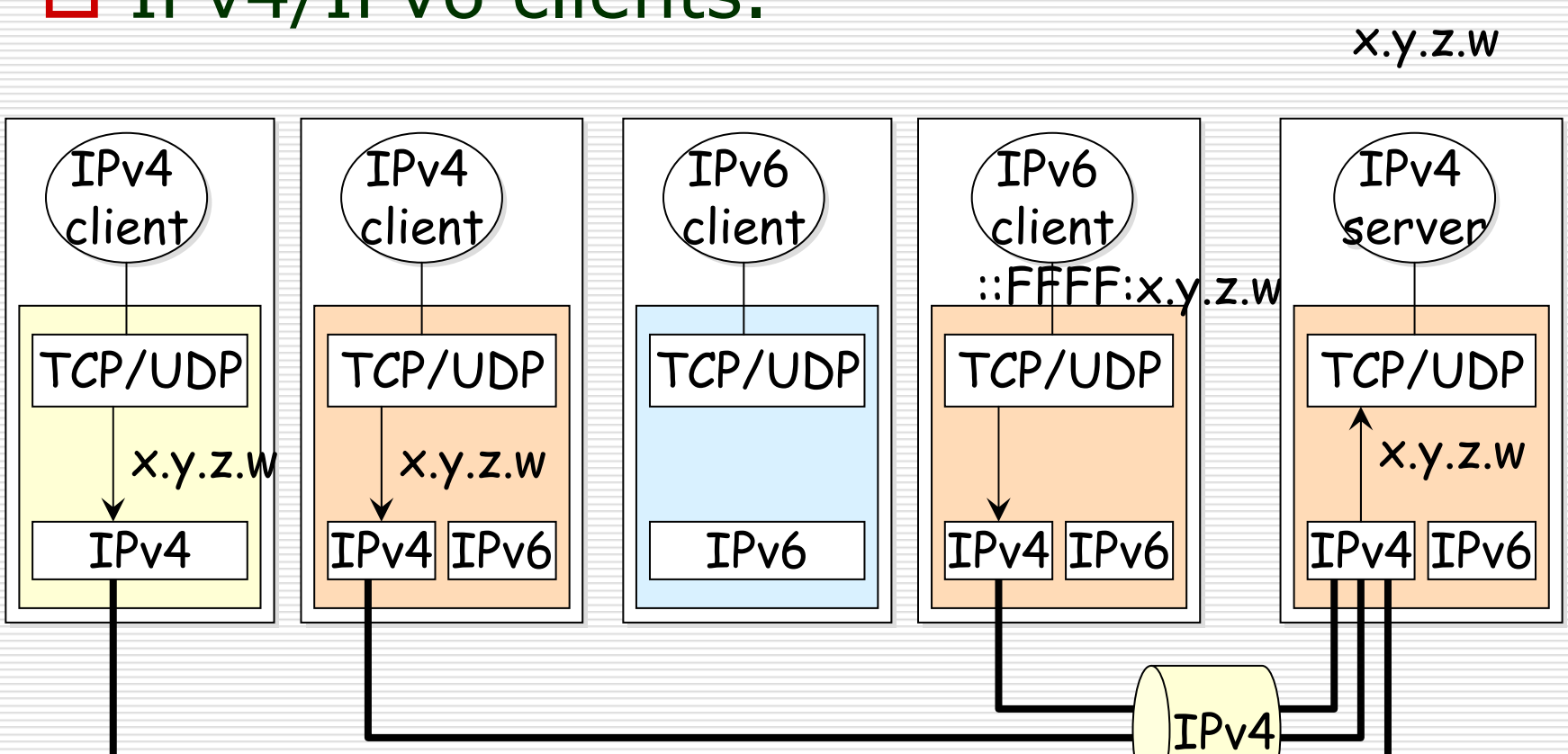
2.IPv6 server at IPv6-only node

- IPv4/IPv6 clients.



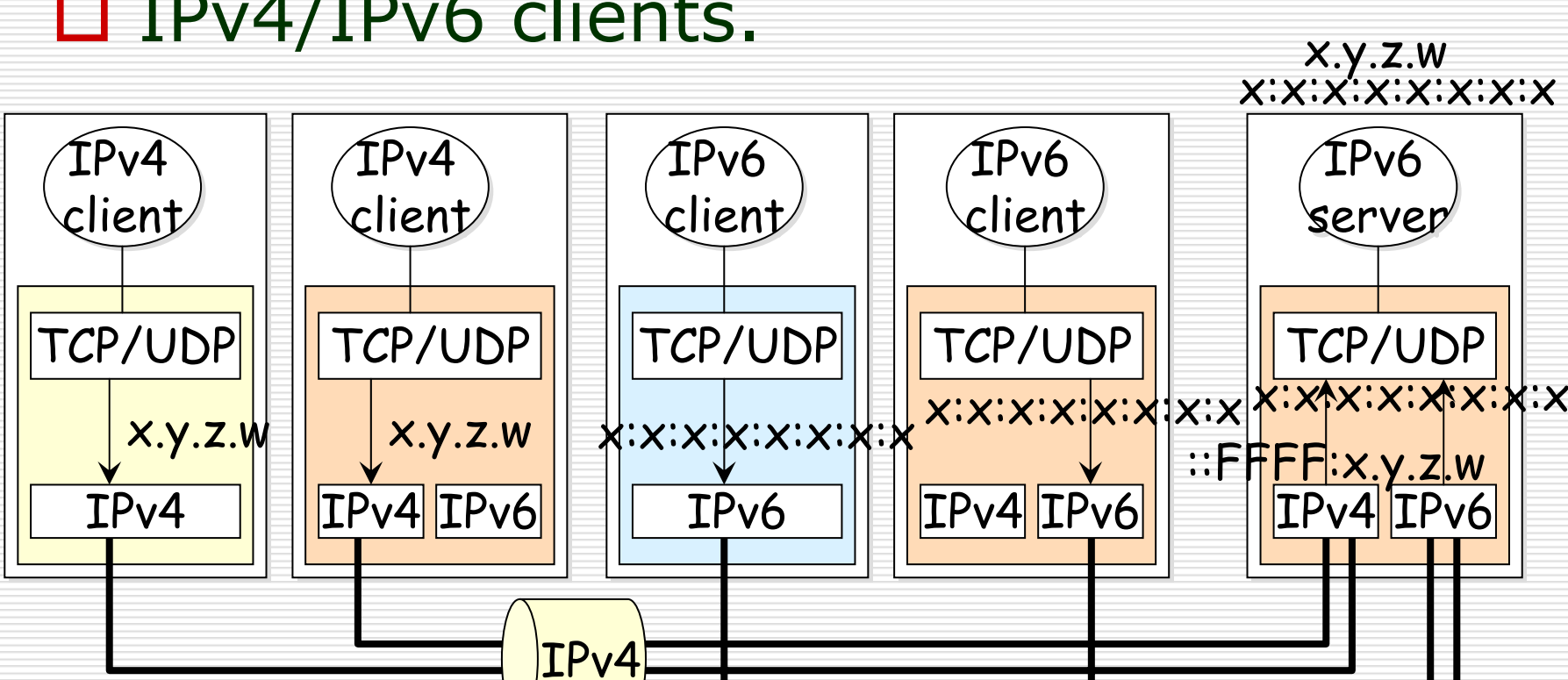
3.IPv4 server at dual stack node

- IPv4/IPv6 clients.



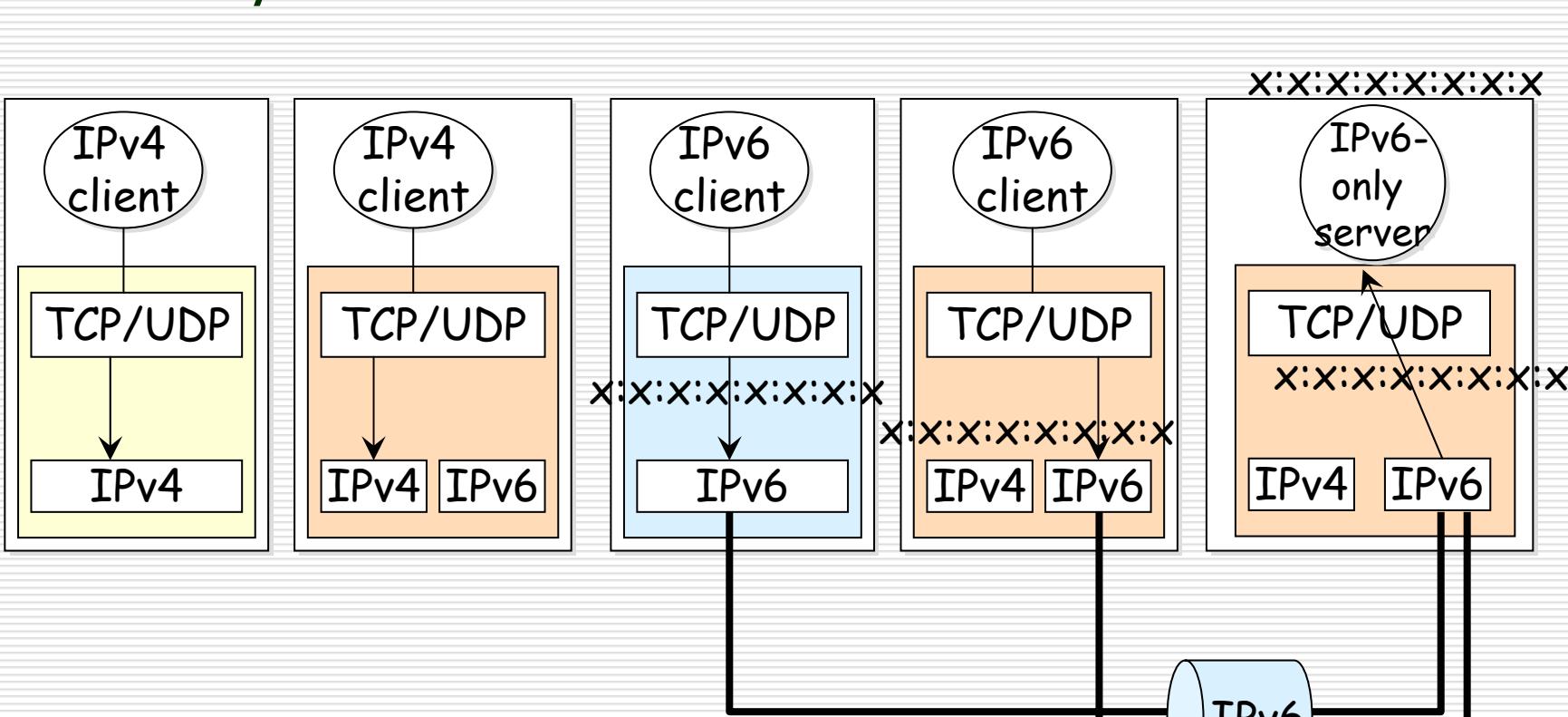
4.IPv6 server at dual stack node

- IPv4/IPv6 clients.



5. IPv6-only servers at dual stack

- IPv4/IPv6 clients.



Client server interoperability

		IPv4 server		IPv6 server	
		IPv4 node	Dual stack	IPv6 node	Dual stack
IPv4 client	IPv4 node	IPv4	IPv4		
	Dual stack	IPv4	IPv4		
IPv6 client	IPv6 node			IPv6	IPv6
	Dual stack			IPv6	IPv6

* Network is dual if it is necessary

Client server interoperability

		IPv4 server		IPv6 server	
		IPv4 node	Dual stack	IPv6 node	Dual stack
IPv4 client	IPv4 node	IPv4	IPv4	Fail	IPv4
	Dual stack	IPv4	IPv4	Fail	IPv4
IPv6 client	IPv6 node	Fail	Fail	IPv6	IPv6
	Dual stack	IPv4	IPv4	IPv6	IPv6

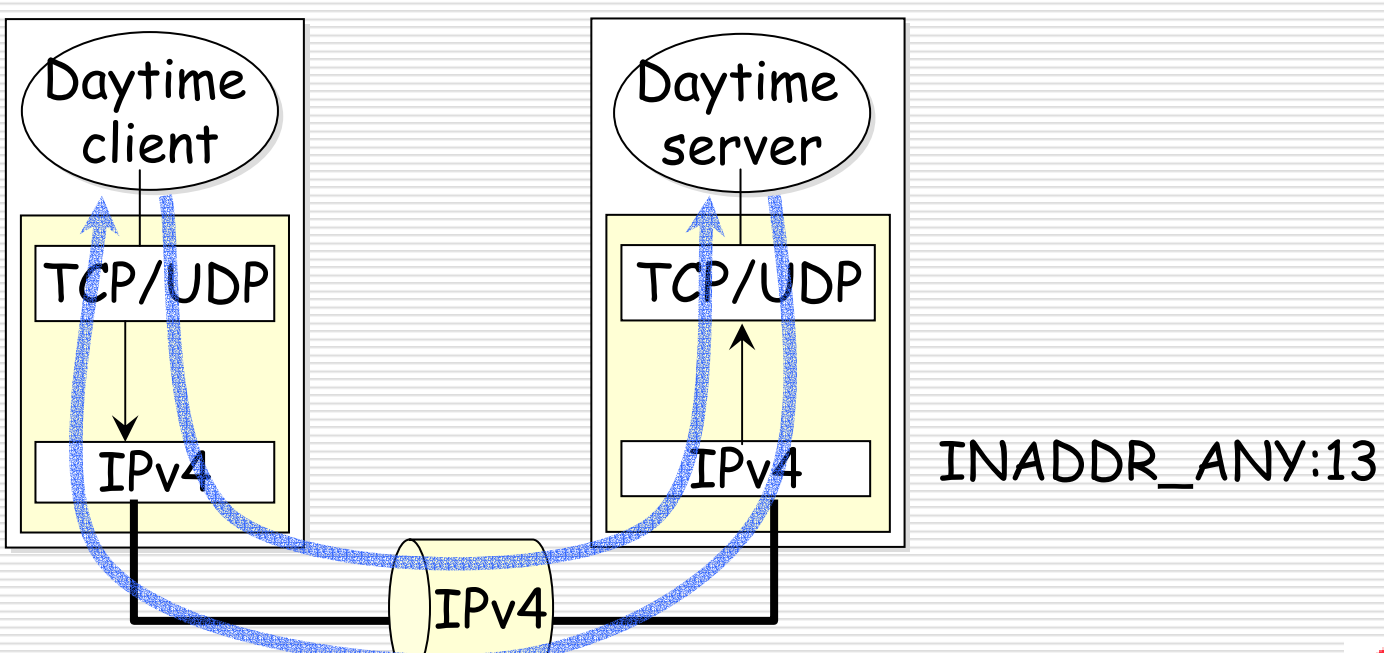
* Network is dual if it is necessary

Agenda

- Transition to IPv6 is only a network issue
- Transition to IPv6 implies application code porting
- Transition exercises
  - Simple Point to point : Daytime
  - Simple Multipoint: Daytime
  - Point to Multipoint : Isabel

Daytime

- A daytime service sends the current date and time as a character string without regard to the input (RFC 867).



IPv4: Unicast Daytime Server

1. Creating server socket.

```
struct sockaddr_in serverAddr;

alen      = sizeof(struct sockaddr_in);
socktype = SOCK_STREAM; /* SOCK_DGRAM */
serverAddr.sin_family      = AF_INET;
serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
serverAddr.sin_port        = htons(DAYTIME_PORT);

serverfd = socket(PF_INET, socktype, 0);
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
listen(serverfd, LISTEN_QUEUE);
```

IPv4: Unicast Daytime Server

2. Waiting for client connections

```
for (; ;) {
    struct sockaddr_in clientAddr;
    alen        = sizeof(struct sockaddr_in);
    connectedfd = accept(serverfd,
                         (struct sockaddr *)&clientAddr,
                         &alen);
    clienthost  = inet_ntoa(clientAddr.sin_addr);
    port        = ntohs(clientAddr.sin_port);
    printf("Request from host=[%s] port=[%d]\n",
           clienthost, port);
    myGetTimeFunction(timeStr);
    write(connectedfd, timeStr, strlen(timeStr));
    close(connfd);
}
```

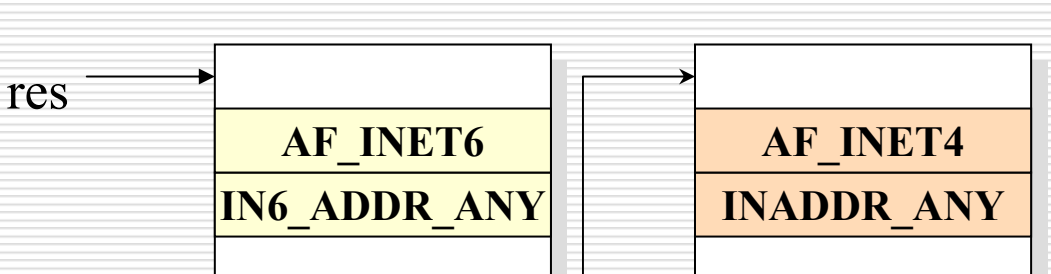
IPv4: Unicast Daytime Client

```
struct sockaddr_in serverAddr;
struct hostent *phe;

alen      = sizeof(struct sockaddr_in);
socktype = SOCK_STREAM; /* SOCK_DGRAM */
serverAddr.sin_family       = AF_INET;
serverAddr.sin_port         = htons(DAYTIME_PORT);
phe = gethostbyname(hostname);
memcpy(&serverAddr.sin_addr, phe->h_addr, phe->h_length);

clientfd = socket(PF_INET, socktype, 0);
connect(clientfd, (struct sockaddr *)&serverAddr, &alen);

while (read(connectedfd, timeStr, sizeof(timeStr)) > 0)
        printf("%s", timeStr);

close(connectedfd);
```

IPv6: Unicast Daytime Server

1. Obtaining server wildcard address

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags     = AI_PASSIVE;
hints.ai_family    = AF_UNSPEC;
hints.ai_socktype  = SOCK_STREAM; /* SOCK_DGRAM */
getaddrinfo(NULL, DAYTIME_PORT, &hints, &res);
```



IPv6: Unicast Daytime Server

- 2. Creating server socket.

```
serverfd = socket(res->family, res->ai_socktype,
                  res->ai_protocol);

bind(serverfd, res->ai_addr, res->ai_addrlen);

listen(serverfd, LISTEN_QUEUE);

freeaddrinfo(res);
```

IPv6: Unicast Daytime Server

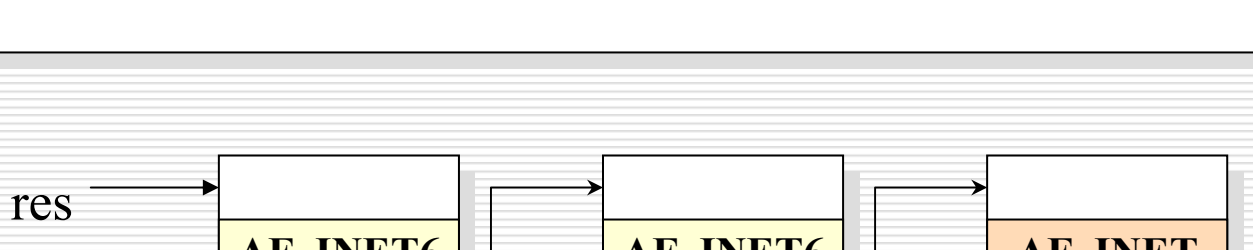
- 3. Waiting for client connections

```
for (; ;) {
    struct sockaddr_storage clientAddr;
    alen        = sizeof(struct sockaddr_storage);
    connectedfd = accept(serverfd,
                         (struct sockaddr *)&clientAddr,
                         &alen);
    getnameinfo((struct sockaddr *)&clientAddr, alen,
                clientHost, sizeof(clientHost),
                clientPort, sizeof(clientPort),
                NI_NUMERICHOST);
    printf("Request from host=[%s] port=[%s]\n",
           clienthost, clientservice);
    myGetTimeFunction(timeStr);
    write(connectedfd, timeStr, strlen(timeStr));
    close(connfd);
}
```

IPv6: Unicast Daytime Client

1. Obtaining server address

```
struct addrinfo hints, *res, *ressave;

memset(0, &hints, sizeof(hints);
hints.ai_family   = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM; /* SOCK_DGRAM */
getaddrinfo(serverHost, DAYTIME_PORT, &hints, &res);
```



IPv6: Unicast Daytime Client

2. Connecting server

```
ressave=res;
for (; res!=NULL; res=res->ai) {
    clienfd = socket(res->family, res->ai_socktype,
                     res->ai_protocol);
    if (clientfd <0)
        continue;
    if (connect(clientfd, res->ai_addr, res->ai_addrlen)==0)
        break;
    close(serverfd);
};
while (read(connectedfd, timeStr, sizeof(timeStr)) > 0)
        printf("%s", timeStr);
close(connectedfd);
freeaddrinfo(ressave);
```

Agenda

- Transition to IPv6 is only a network issue
- Transition to IPv6 implies application code porting
- Transition exercises
  - Simple Point to point : Daytime
  - Simple Multipoint: Daytime
  - Point to Multipoint : Isabel

Multicast Daytime

Server:

- Create socket
- Join multicast group

Client:

- Create socket
- Join multicast group

- Send time response

- Receive time response

IPv4: Multicast Daytime

Creating socket

```
struct sockaddr_in mcastAddr;
struct hostent *phe;

alen     = sizeof(struct sockaddr_in);
socktype = SOCK_DGRAM
mcastAddr.sin_family      = AF_INET;
mcastAddr.sin_port        = htons(DAYTIME_PORT);
phe = gethostbyname(multicastGroup);
memcpy(&mcastAddr.sin_addr, phe->h_addr, phe->h_length);

mcastfd = socket(PF_INET, socktype, 0);
bind(mcastfd, (struct sockaddr *)&mcastAddr, &alen);
```

IPv4: Multicast Daytime

- Joining multicast group

```
struct ip_mreq      mreq;

memcpy(&mreq.imr_multiaddr, mcastAddr)->sin_addr,
       sizeof(struct in_addr));
mreq.imr_interface.s_addr= htonl(INADDR_ANY);

n = setsockopt(sockfd, IPPROTO_IP,IP_ADD_MEMBERSHIP,
                &mreq, sizeof(mreq));
```

IPv4: Multicast Daytime Server

- Sending time response.

```
for ( ; ;) {
   myGetTimeFunction(timeStr);
   n = sendto(sockfd, timeStr, sizeof(timeStr), 0,
              (struct sockaddr *)&mcastAddr,
              sizeof(mcastAddr));
   sleep(1000);
}
```

IPv4: Multicast Daytime Client

- Receiving time response.

```
struct sockaddr_in serverAddr;


n = recvfrom(sockfd, buffer, sizeof(buffer), 0,
             (struct sockaddr *)&serverAddr,
             &addrlen);
printf("%s\n", buffer);
close(sockfd);
```

IPv6: Multicast Daytime Server

- Creating socket.

```
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_family   = AF_UNSPEC;
hints.ai_socktype = SOCK_DGRAM;
getaddrinfo(multicastGroup, DAYTIME_PORT, &hints, &res);

mcastfd = socket(res->ai_family, res->ai_socktype,
                 res->ai_protocol);
bind(mcastfd, res->ai_addr, res->ai_addrlen);
```

IPv6: Multicast Daytime

- Joining a multicast group (IPv4).

```
addr= res->ai_addr;
switch (addr->ai_family) {
    case AF_INET: {
        struct ip_mreq      mreq;
        memcpy(&mreq.imr_multiaddr,
               ((struct sockaddr_in *)addr)->sin_addr),
                sizeof(struct in_addr));
        mreq.imr_interface.s_addr= htonl(INADDR_ANY);
        n = setsockopt(mcastfd, IPPROTO_IP,
                       IP_ADD_MEMBERSHIP,
                       &mreq, sizeof(mreq));
    } break;
```

IPv6: Multicast Daytime

- Joining a multicast group (IPv6).

```
case AF_INET6: {
        struct ipv6_mreq    mreq6;
        memcpy(&mreq6.ipv6mr_multiaddr,
              &(((struct sockaddr_in6 *)addr)->sin6_addr),
              sizeof(struct in6_addr));

        mreq6.ipv6mr_interface= 0;

        n = setsockopt(mcastfd, IPPROTO_IPV6,
                       IPV6_ADD_MEMBERSHIP, &mreq6,
                       sizeof(mreq6));
    } break;
}
freeaddrinfo(res);
```

IPv6: Multicast Daytime Server

- Sending time response.

```
for ( ; ;) {
   myGetTimeFunction(timeStr);
   n = sendto(sockfd, timeStr, sizeof(timeStr), 0,
              (struct sockaddr *)&mcastAddr,
              sizeof(mcastAddr));
   sleep(1000);
}
```

IPv6: Multicast Daytime Client

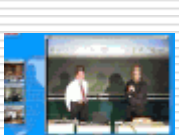
- Receiving time response.

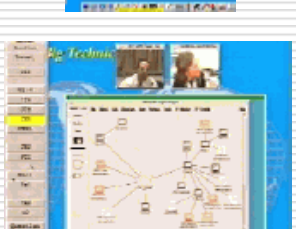
```
struct sockaddr_storage serverAddr;

n = recvfrom(sockfd, buffer, sizeof(buffer), 0,
             (struct sockaddr *)&serverAddr,
             &addrlen);

printf("%s\n", buffer);

close(sockfd);
```

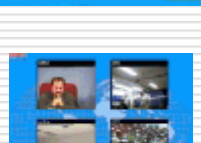
Agenda

- Transition to IPv6 is only a network issue
- Transition to IPv6 implies application code porting
- Transition exercises
  - Simple Point to point : Daytime
  - Simple Multipoint: Daytime
  - Point to Multipoint : Isabel

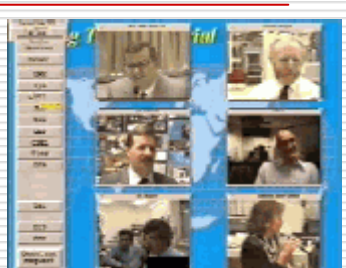
ISABEL

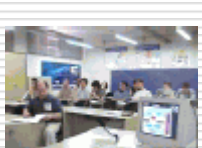






- Group collaboration tool for Internet
  - Developed at UPM
  - http://isabel.dit.upm.es
- To interconnect audiences and groups
  - With a large number of endpoints/users
    - Effective with up to 20 sites
- Introduce innovative service concepts:
  - Tele-meeting
  - Tele-training
  - Tele-conference

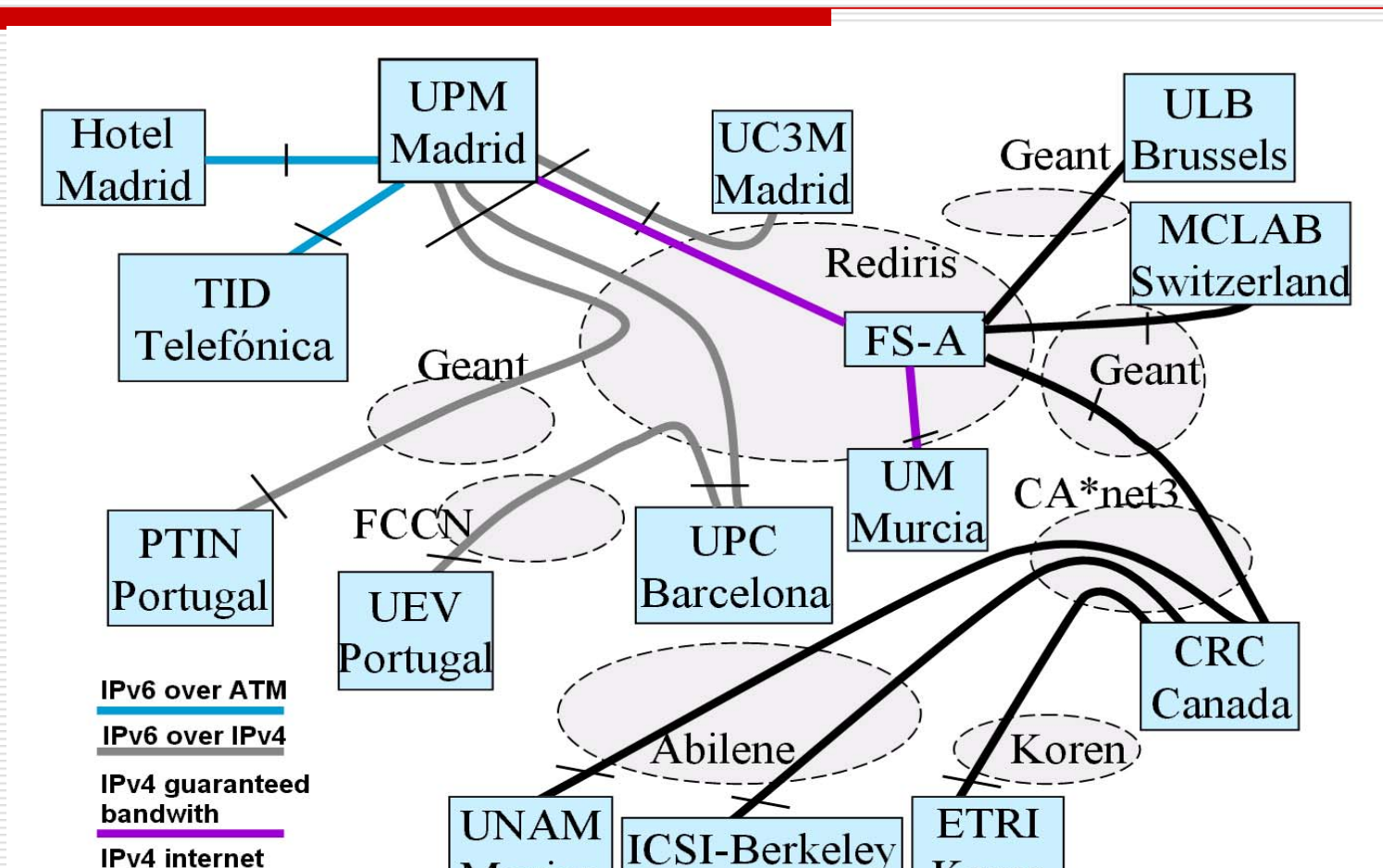




ISABEL usage

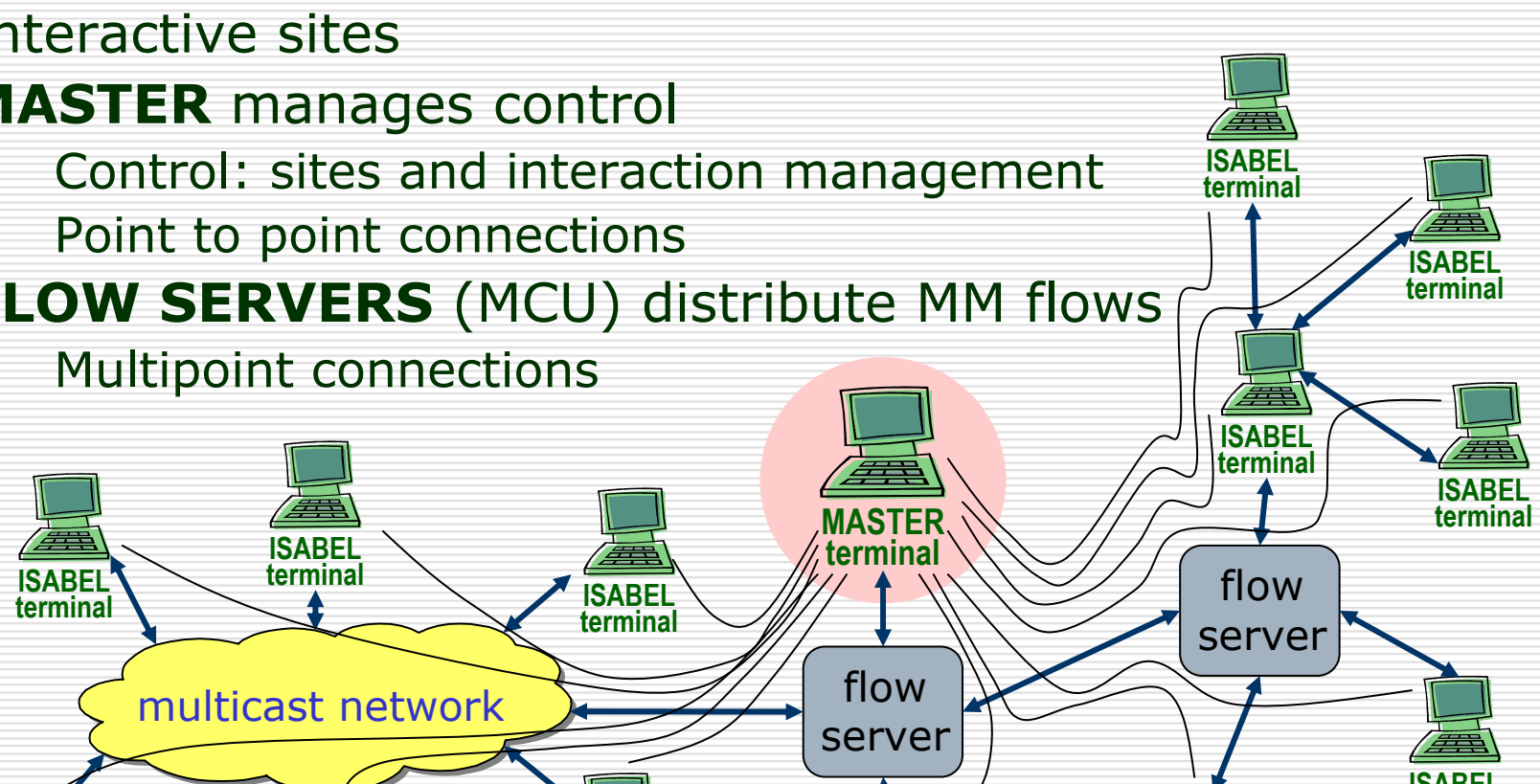
- EU RACE Summer Schools on ABC
  - ABC'93: 2 sites (Spain – Portugal)
  - ABC'94: 5 sites (Spain, Portugal, Switzerland)
  - ABC'95: 11 sites (Europe)
  - ABC'96: 20 sites (Europe & Canada)
- Distributed Congress (97-00)
  - Global 360, Telecom I+D, Internet NG
- Industrial usage
  - Telemeeting service
  - Ericsson, Vodafone, Telefónica, Portugal Telecom
- Distributed courses
  - PHD & graduate courses performed
    - Between Madrid, Barcelona, Valencia, Murcia
- Madrid Global IPv6 Summit 2002
  - First congress **distributed over IPv6** with ISABEL

IPv6 Summit 2002 Madrid

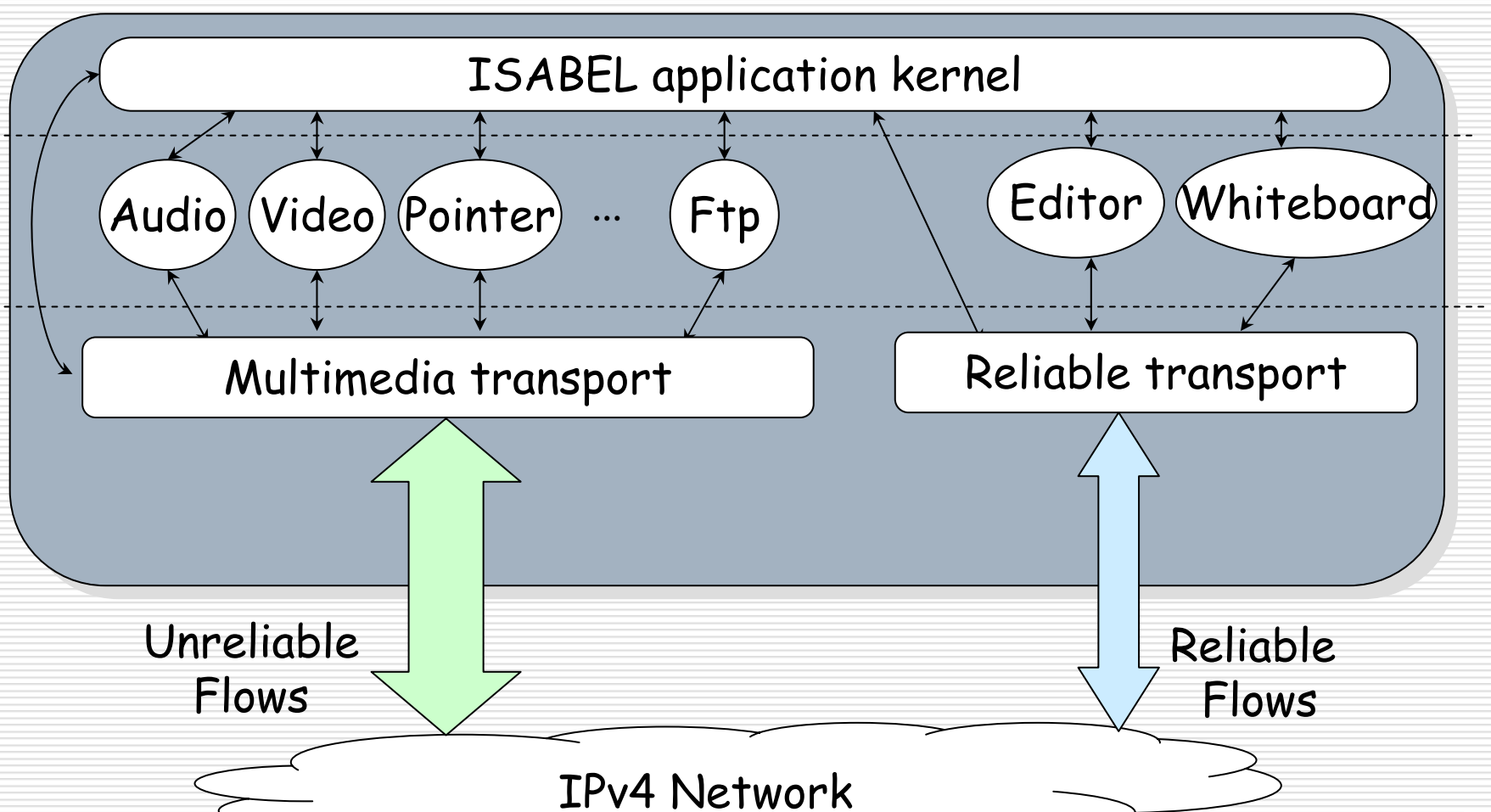


ISABEL network architecture

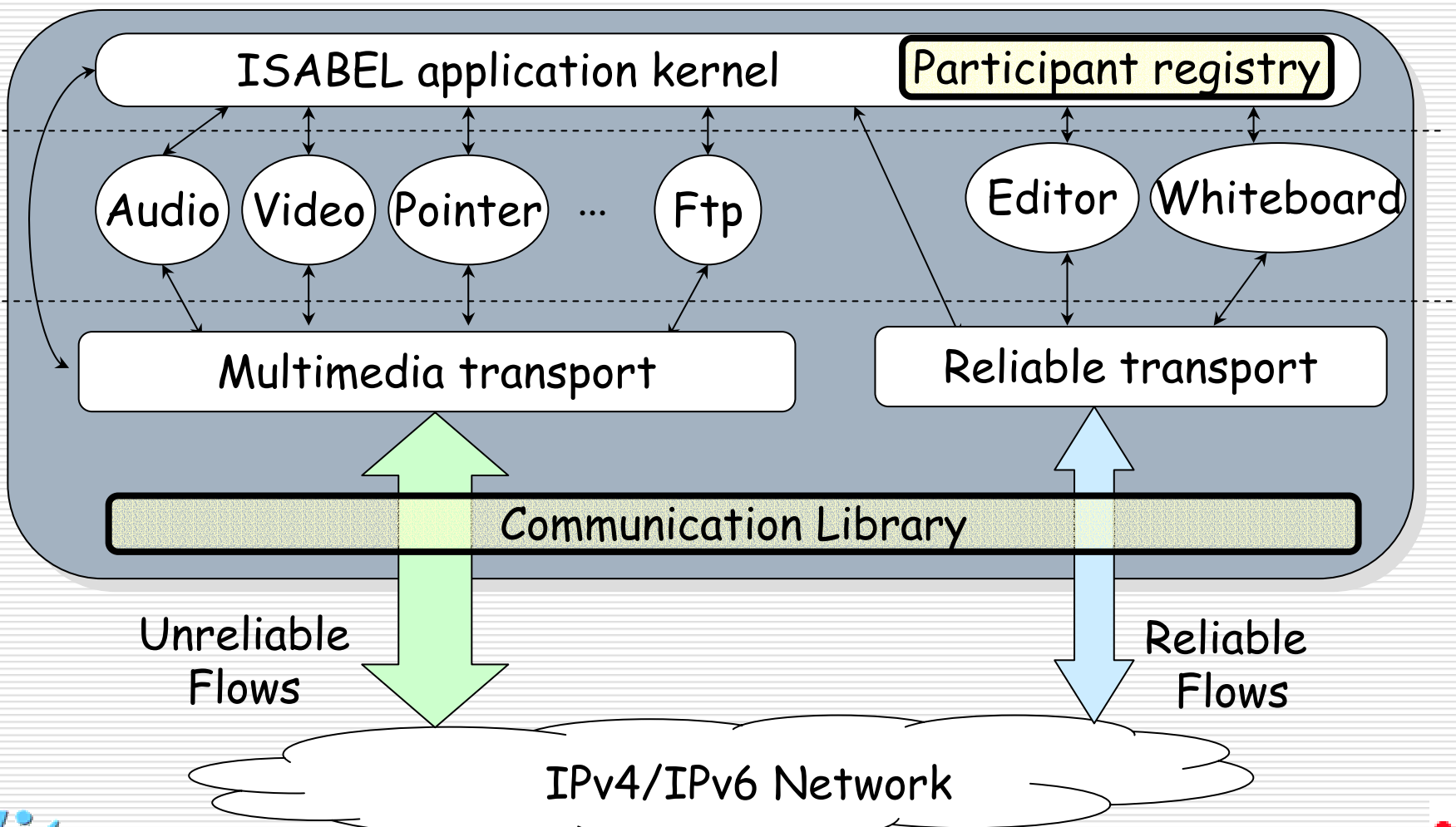
- Interactive sites
- **MASTER** manages control
  - Control: sites and interaction management
  - Point to point connections
- **FLOW SERVERS** (MCU) distribute MM flows
  - Multipoint connections



ISABEL application architecture



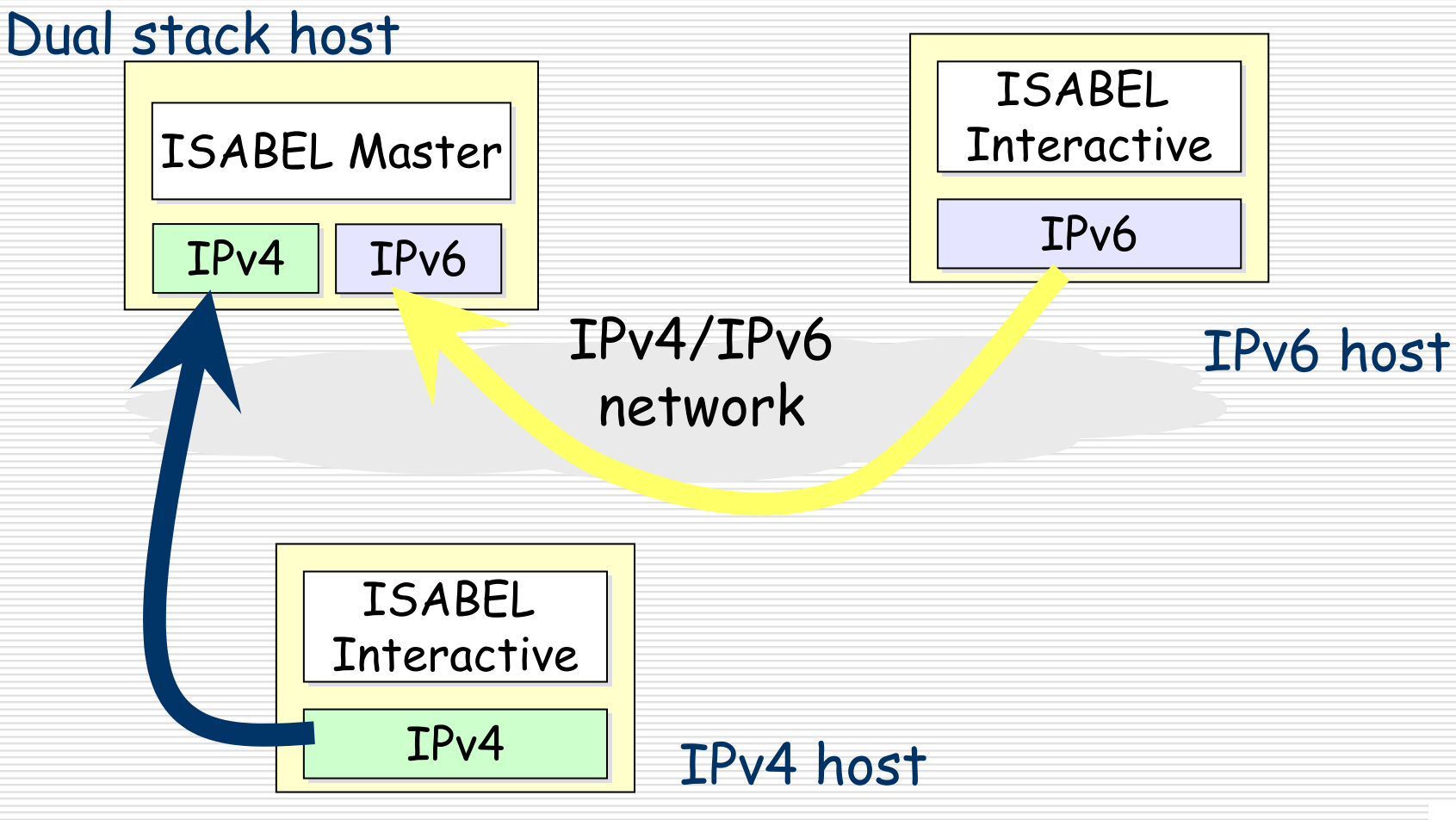
ISABEL porting to IPv6



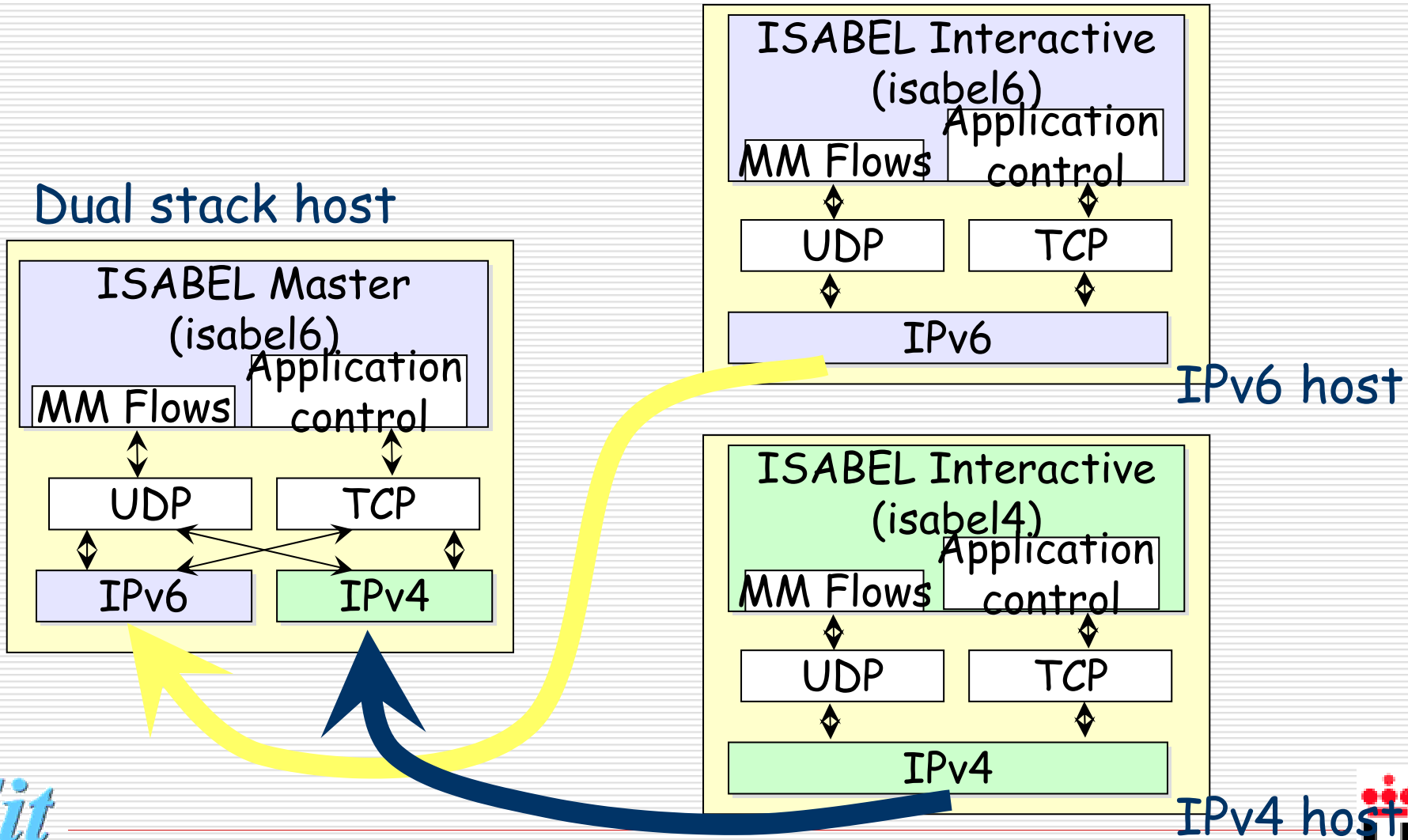
Isabel changes

- Communication library.
  - Dual stack library
  - Private communications interface
    - Object Wrapper Facade Pattern technique
- Participant System Registry.
  - Based on FQDN.
- IP address parser.
  - IPv4 & IPv6 compliant.
- Checking packets fragmentation
  - PMTU-D

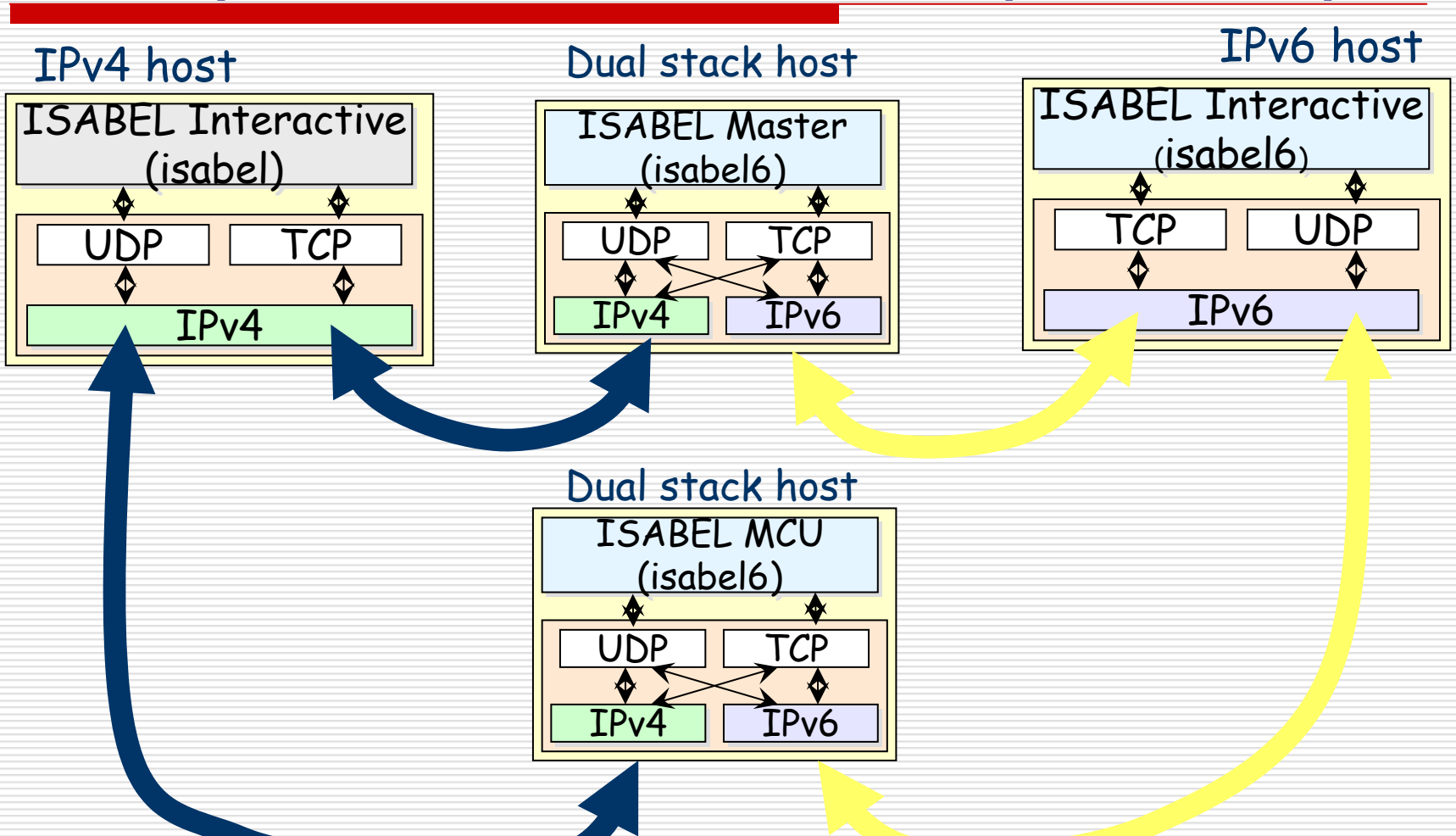
Study case: Isabel interoperability



Study case: Isabel interoperability



Study case: Isabel interoperability



Conclusions of ISABEL porting

- ISABEL was IP addresses dependant
- Redesign was mandatory
  - Introduce dual stack architecture
  - Redesign leads to a more consistent architecture
    - Participants registry is not IP dependant
  - Amount of work invested was reasonable
- Additional work has been done to take advantages of IPv6
  - Security
  - Mobile IP
- Work still needed: QoS

Acknowledgements

This work has been mainly funded by LONG Project (http://www.ist-long.com)

Questions?

- Thanks for your attention
- Any questions?