IPv6 Porting Applications

US IPv6 Summit

Dec 8-11, 2003

Eva M. Castro – eva@gsyc.escet.urjc.es

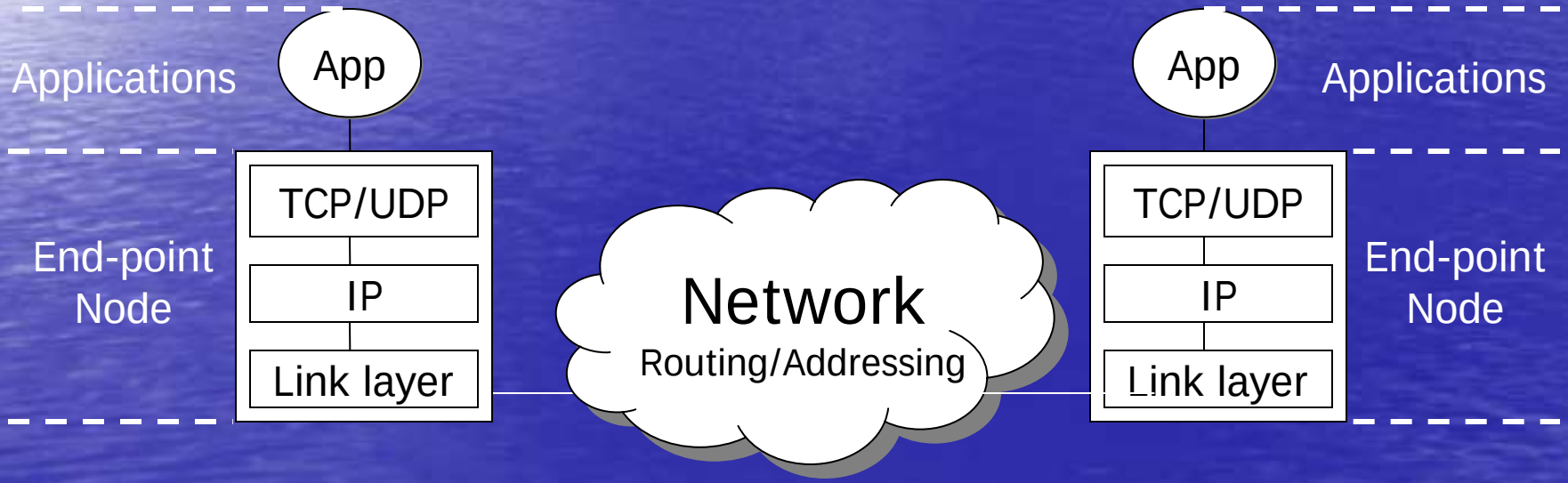
Systems and Communications Group (GSyC)
Experimental Sciences and Technology Department (ESCET)
Rey Juan Carlos University (URJC)

Agenda

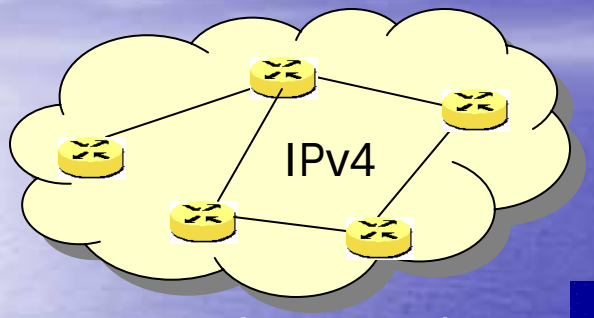
- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

Transition Architecture

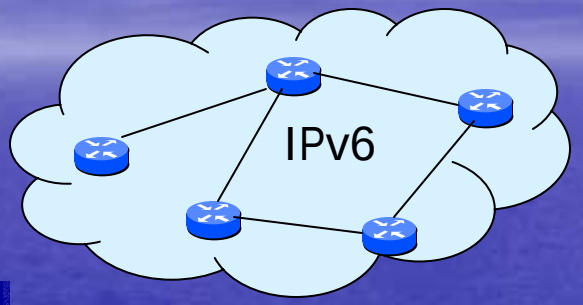
- Network
- End-point nodes
- Applications



Network (routing/addressing)

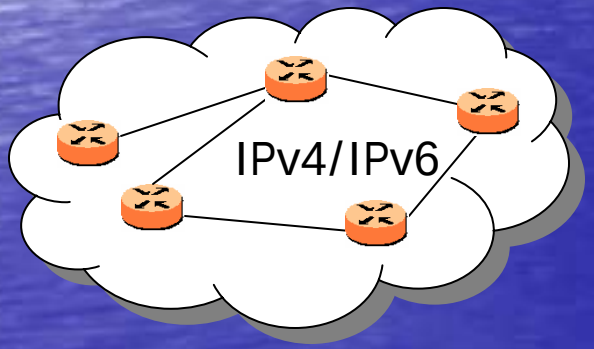


IPv4-only Network

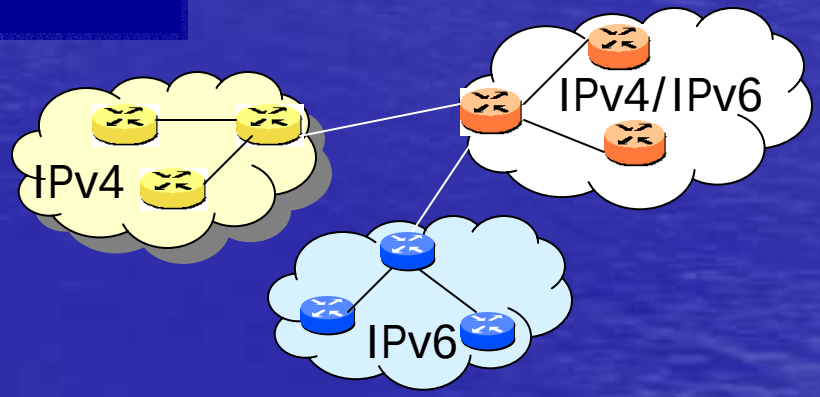


IPv6-only Network

Transition Mechanisms:
- Tunnels
- Protocol translation

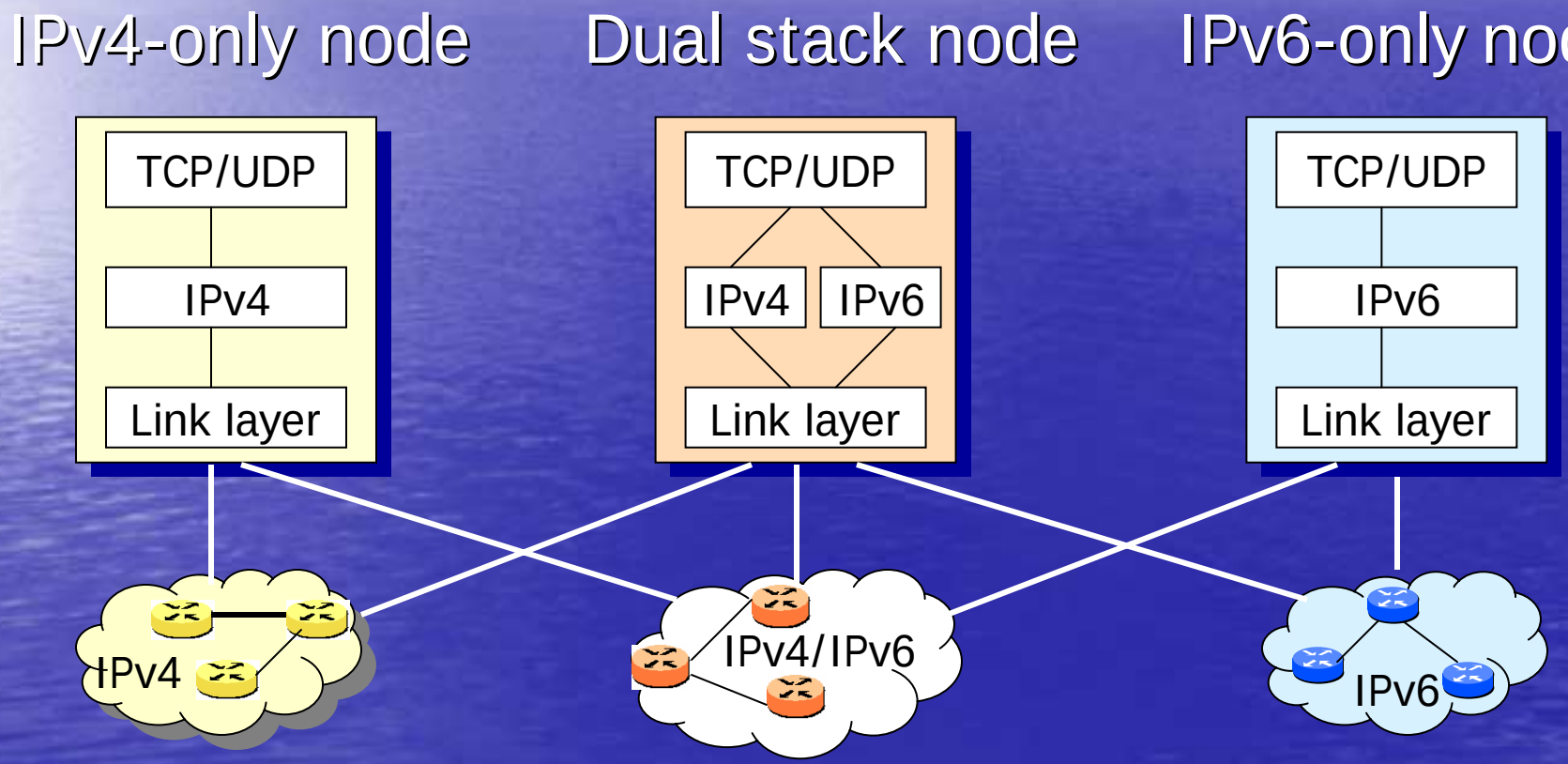


Dual Network



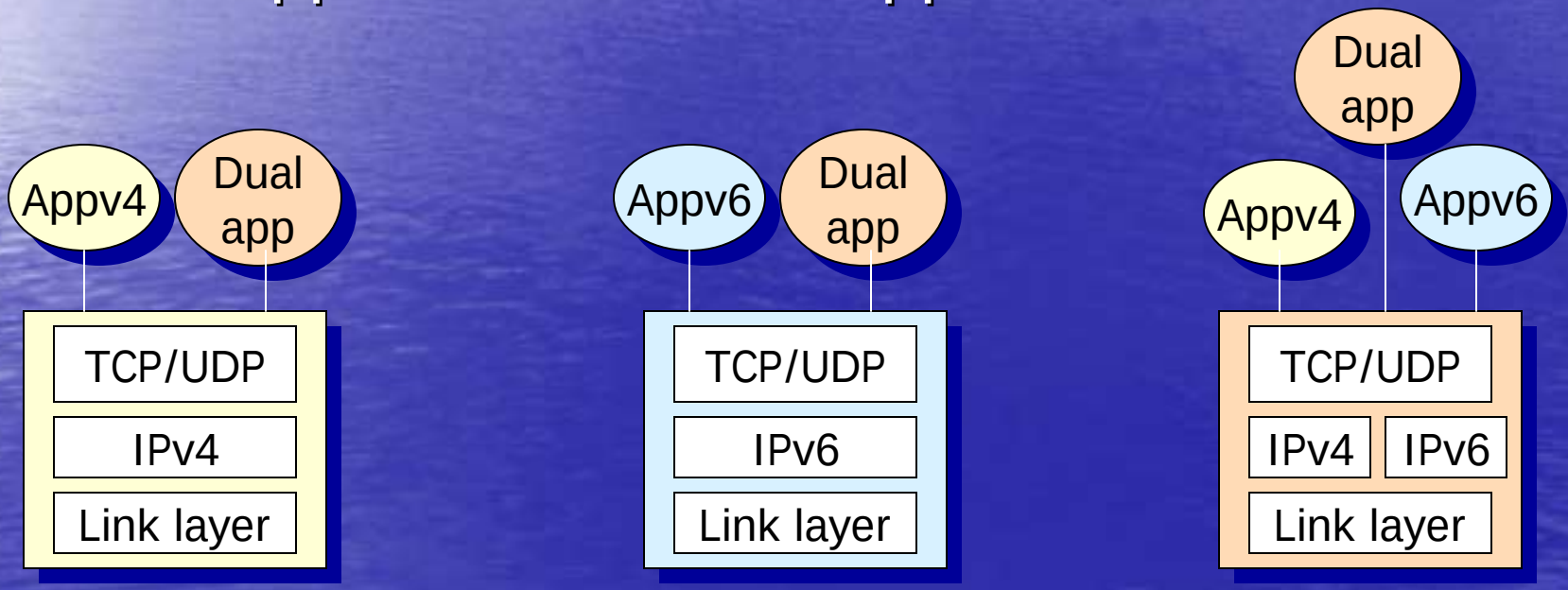
Heterogeneous Network

End-point Nodes (IP stack)

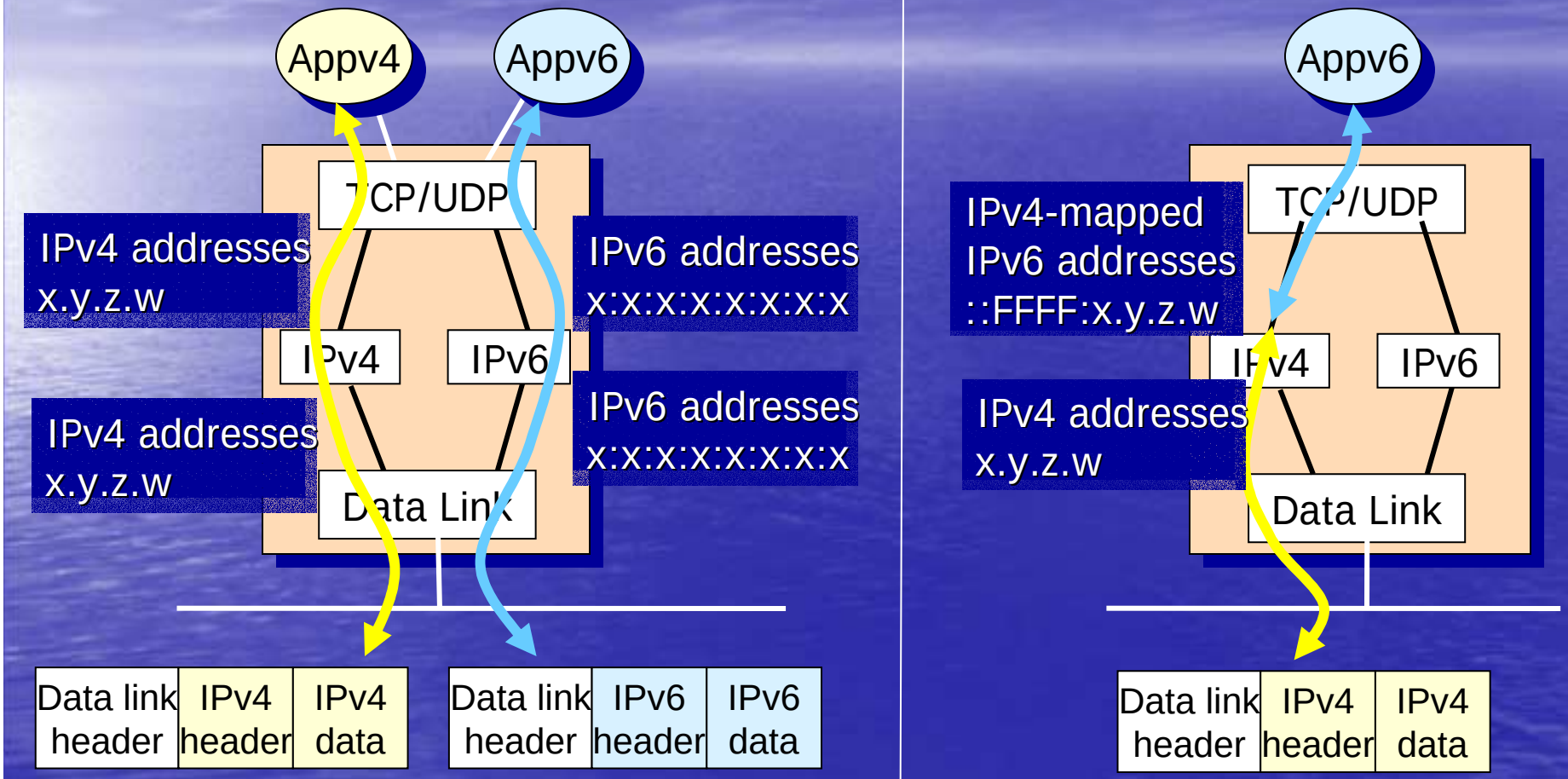


Applications (source code)

- IPv4-only Applications – Appv4
- IPv6-only Applications – Appv6
- Dual Applications – Dual app



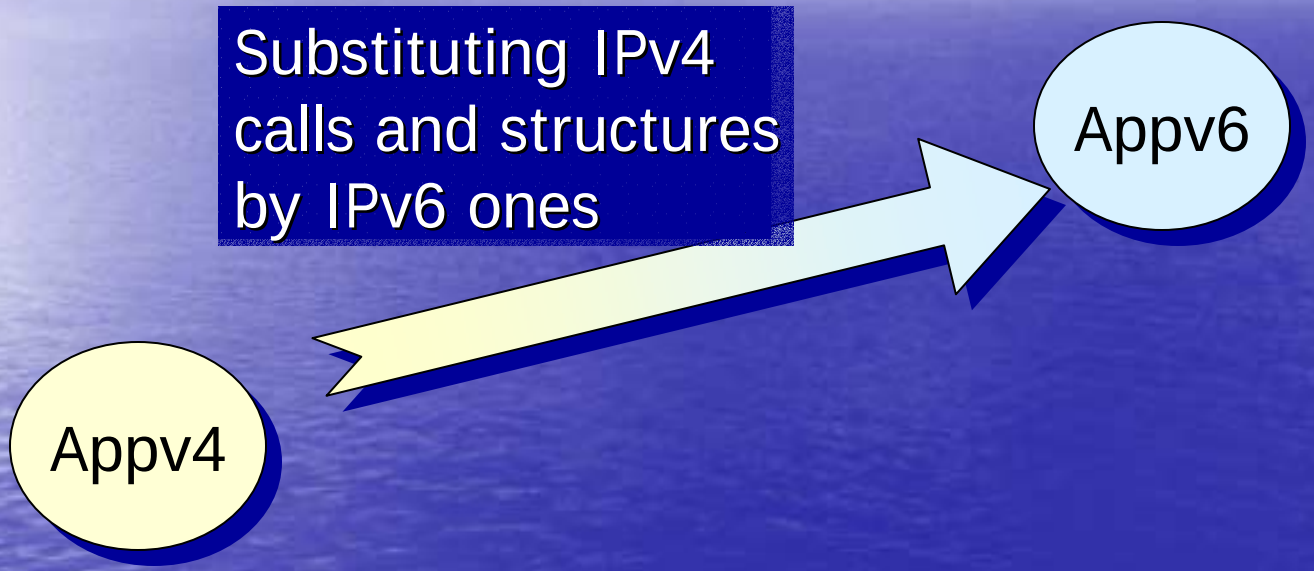
Applications - Dual Nodes



Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

Evolution of Applications



Evolution of Applications

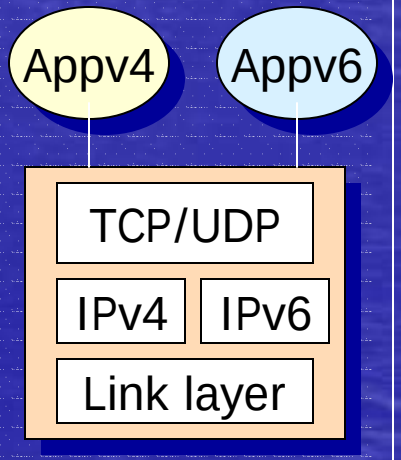
Substituting IPv4 calls and structures by IPv6 ones

Appv6

Appv4

ADVANTAGES:
- Easy task, short time

PROBLEMS:
- Application selection
- Source code maintainability

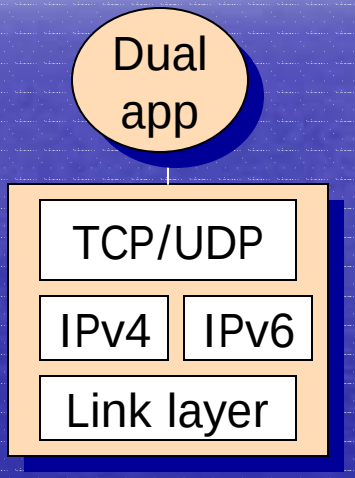


Evolution of Applications

Evolution of Applications

ADVANTAGES:
- One application for all kind of nodes
- Source code maintainability

DISADVANTAGES:
- More changes, long time

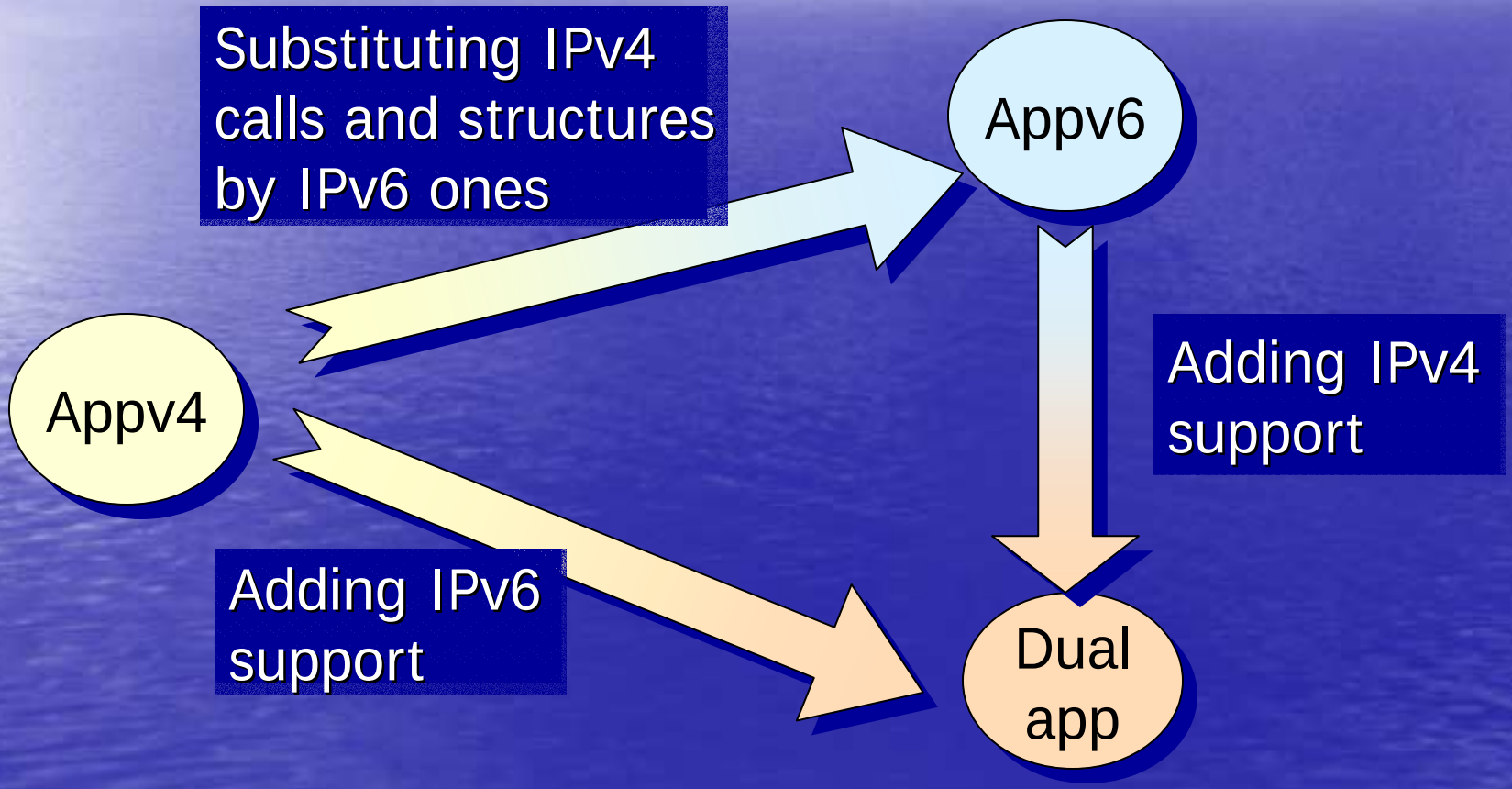


Appv4

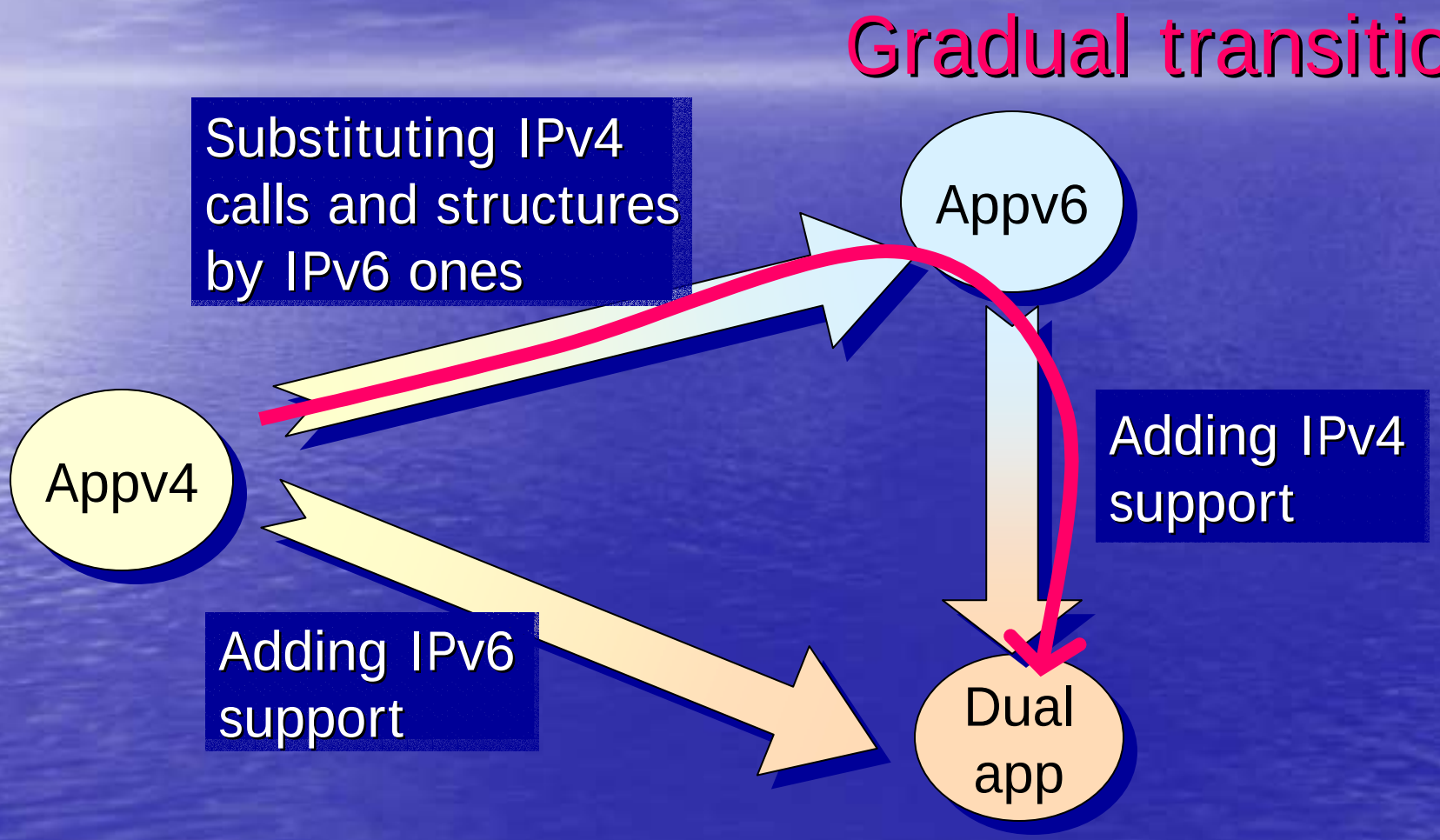
Adding IPv6 support

Dual app

Evolution of Applications



Evolution of Applications



Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

Application Transition Scenarios

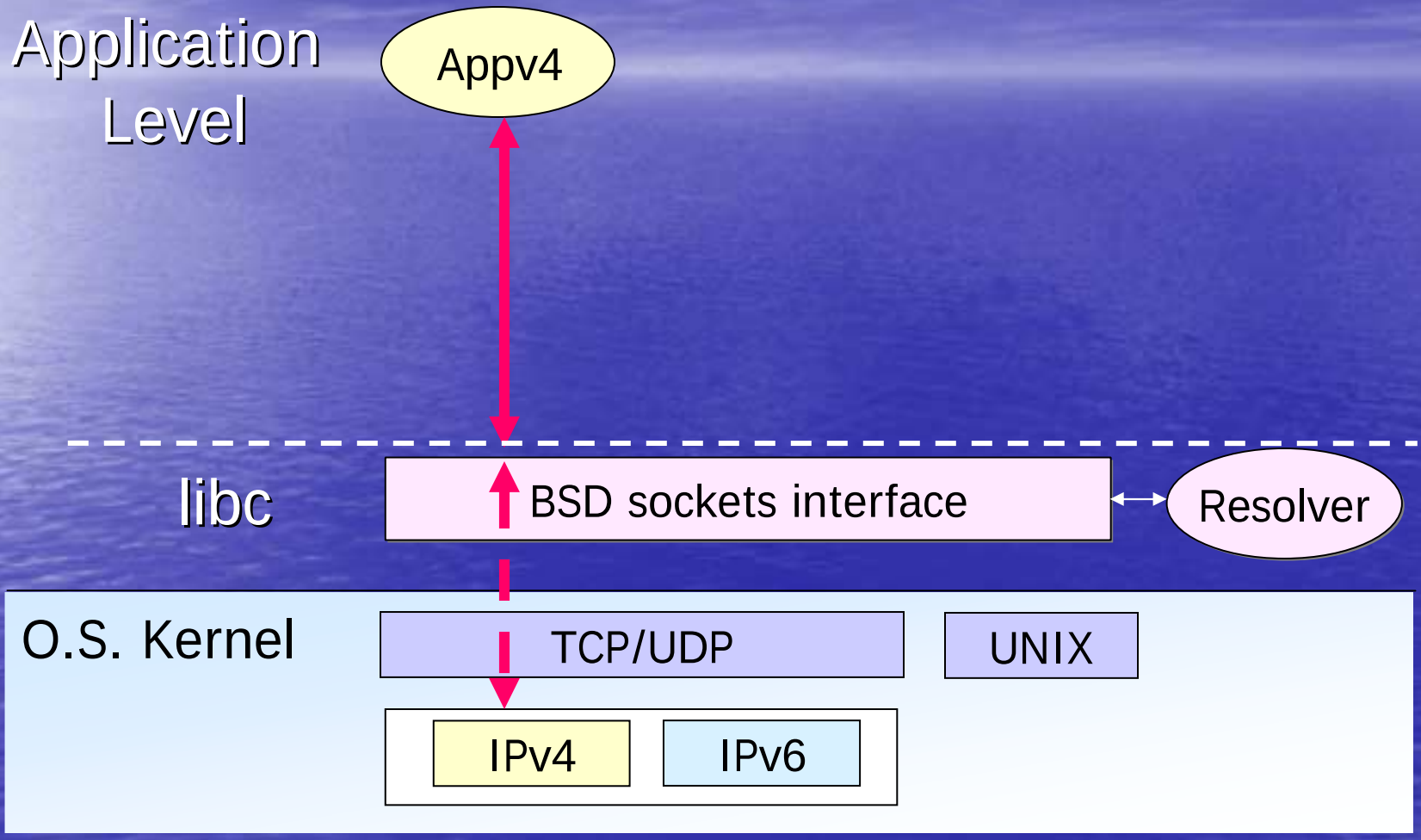
From existing IPv4 applications in IPv4 node:

1. IPv4 applications in dual-stack nodes
2. IPv6 applications in dual-stack nodes
3. Dual applications in dual-stack nodes
4. Dual applications in IPv4 nodes

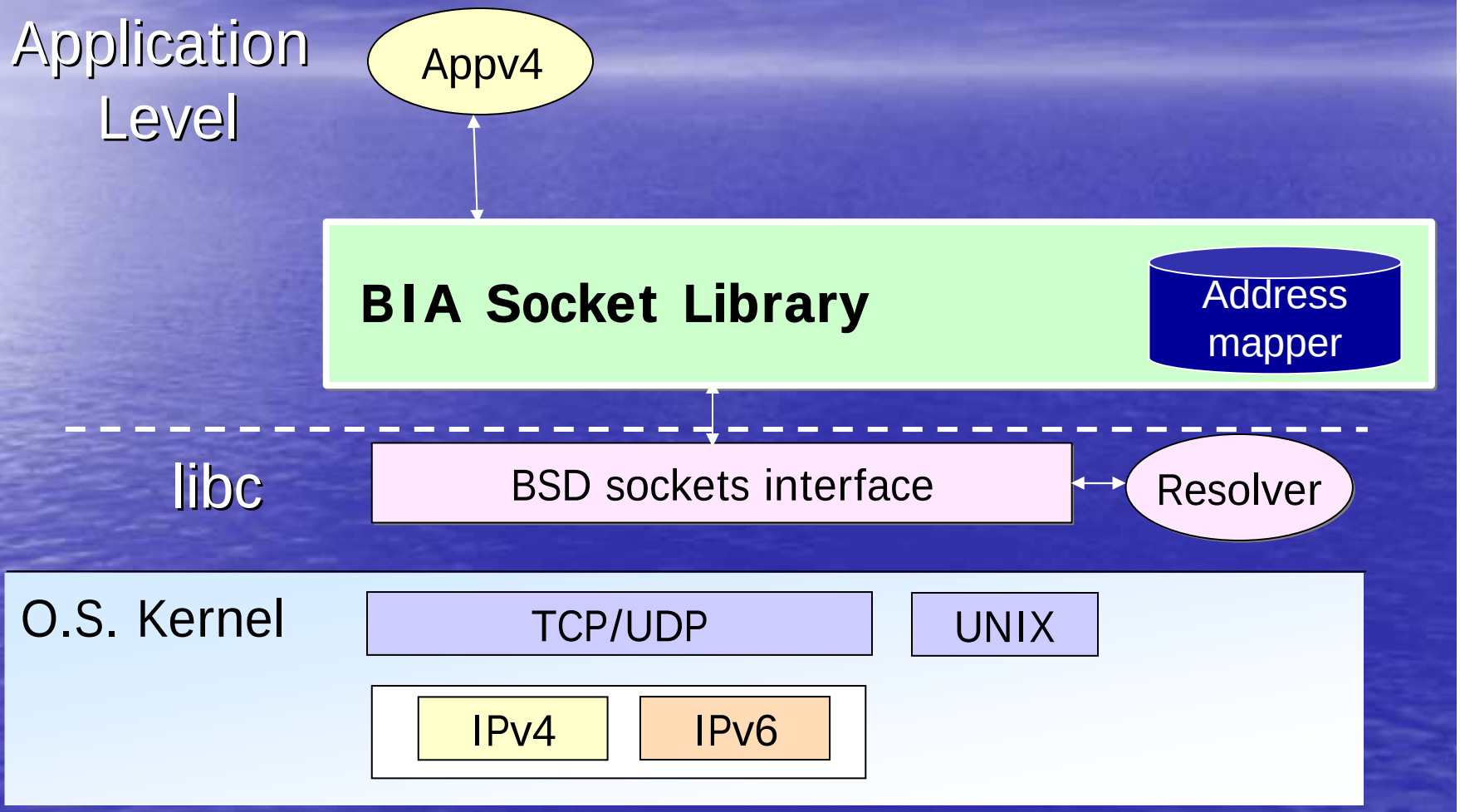
1. IPv4 applications in dual-stack nodes

- IPv4 source code dependencies
- Use IPv4
- If IPv6 communication is required:
  - port source code to IPv6, then applications use native IPv6, or
  - use transition mechanisms. Applications use IPv4, but IPv6 packets are exchanged:
    - **BIA**:
      - IPv4 applications + Dual stack nodes
    - **BIS**:
      - IPv4 applications + IPv4 nodes
        Dual stack nodes

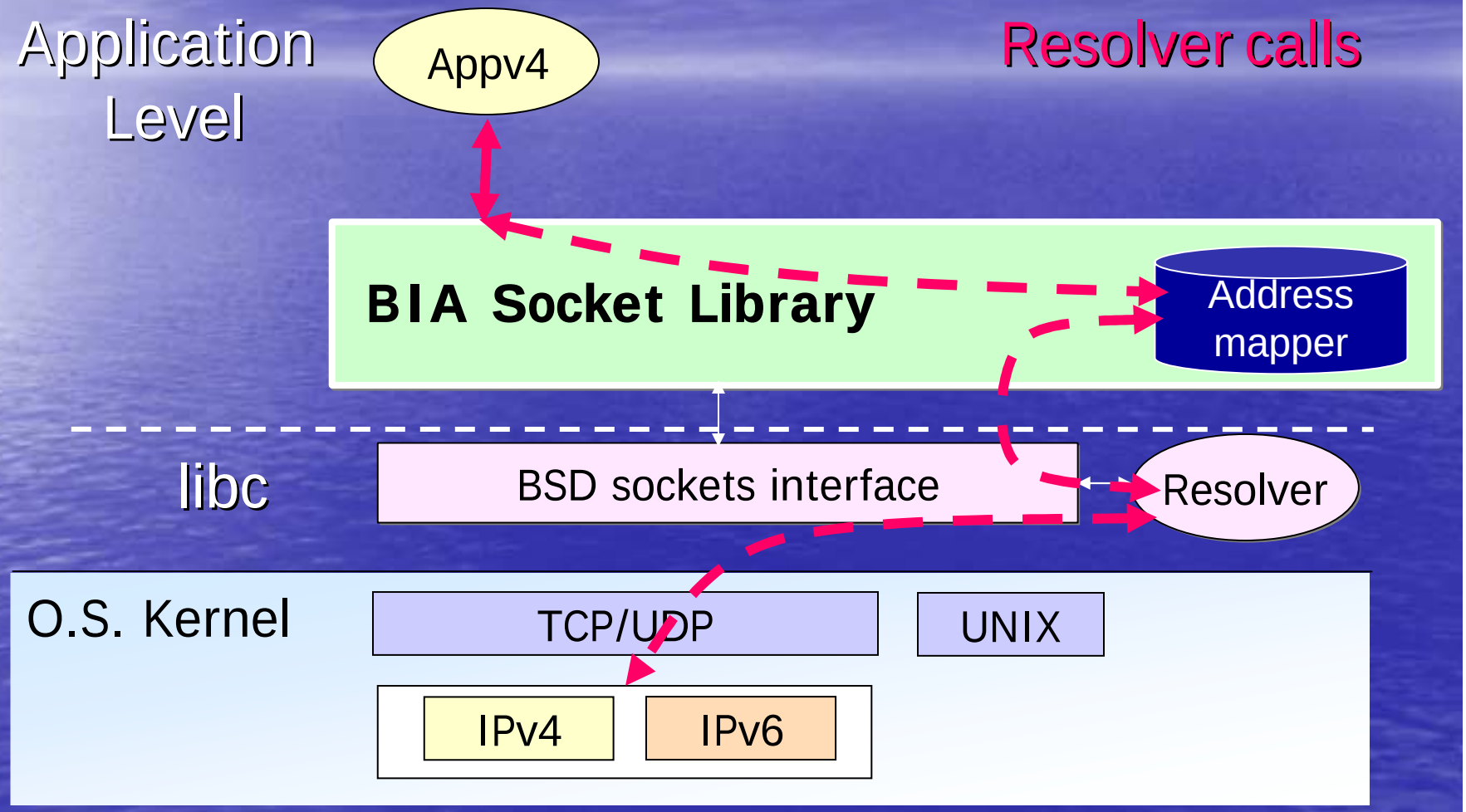
IPv4 Applications



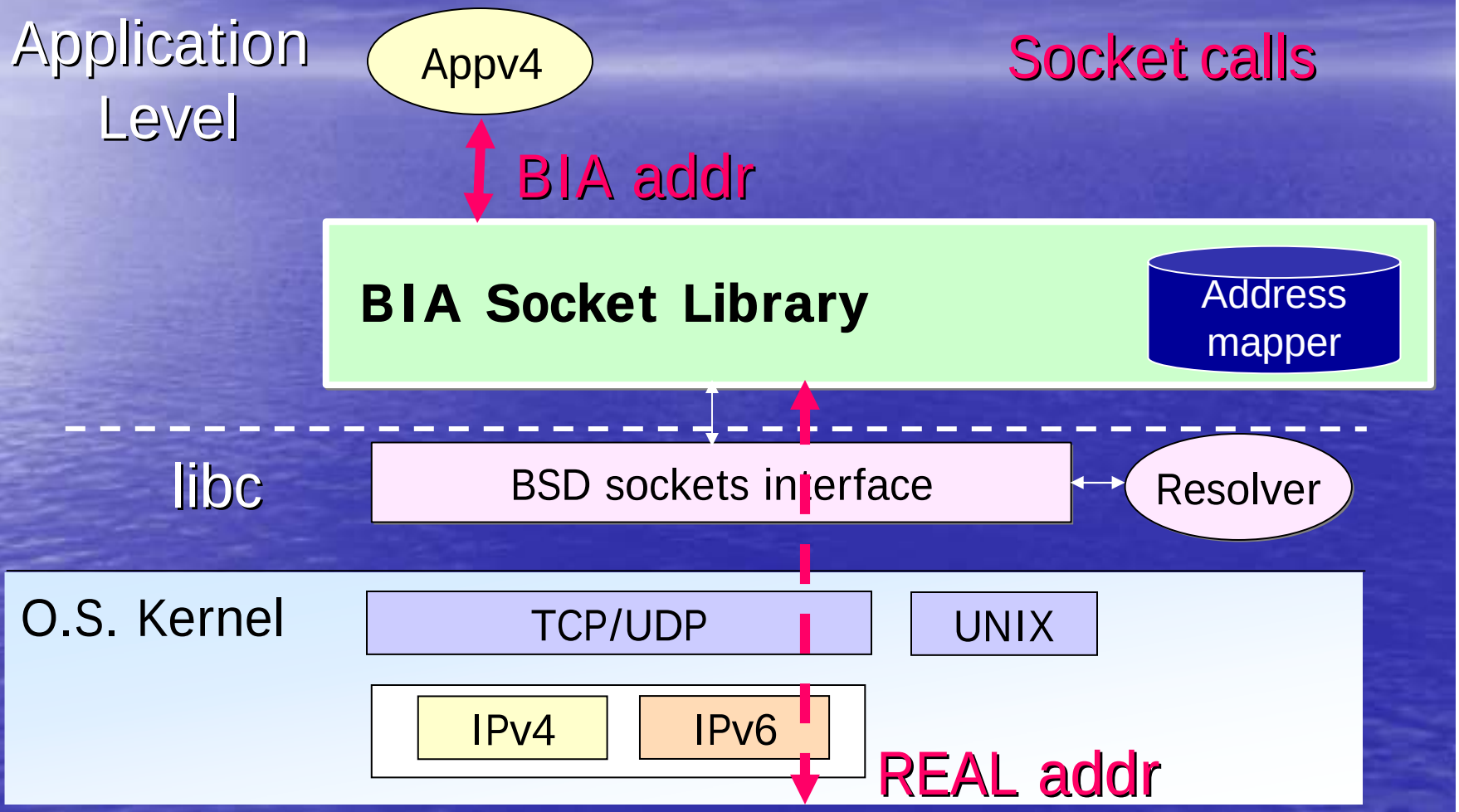
BIA – Bump In the API



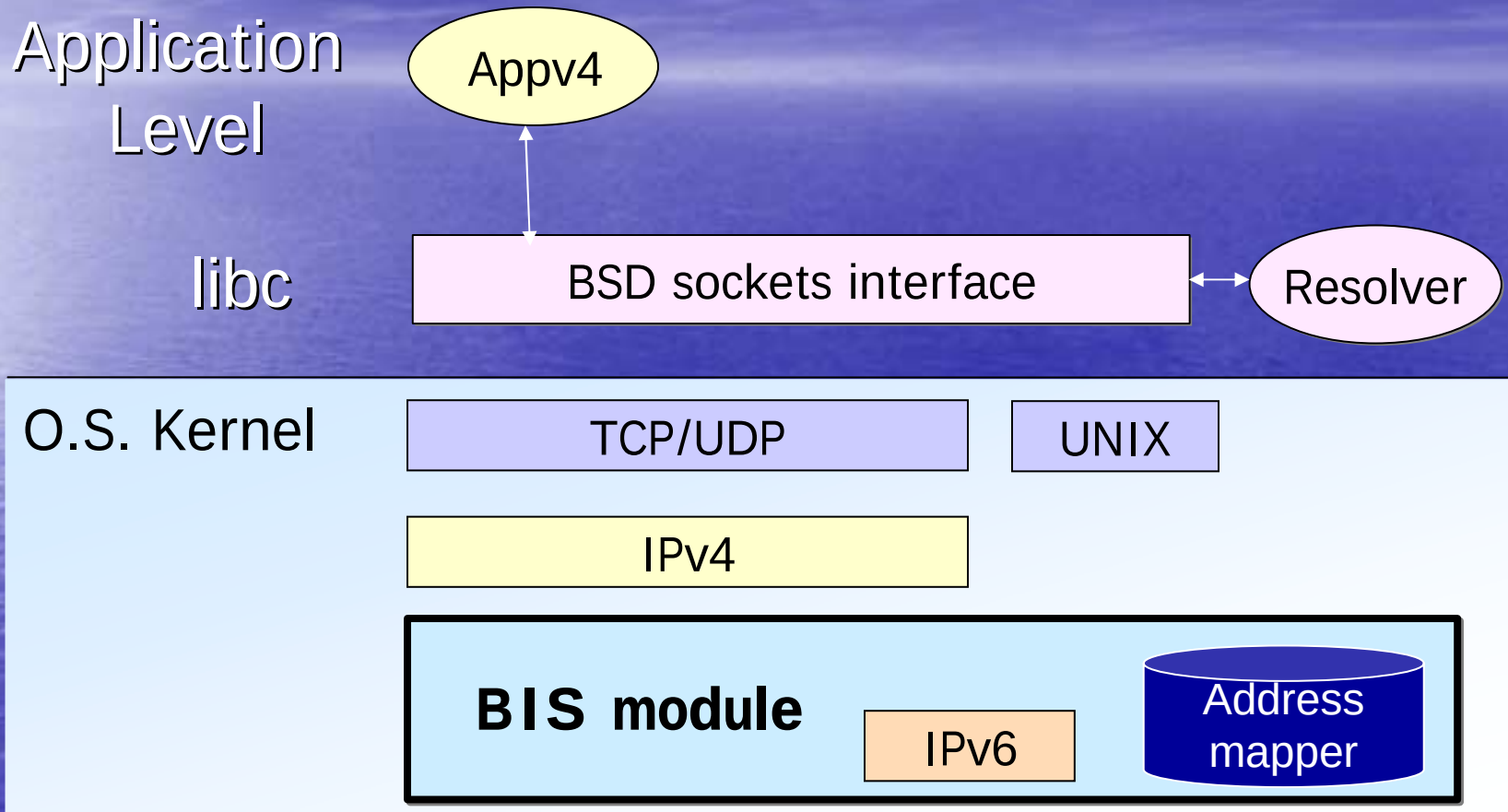
BIA – resolver calls



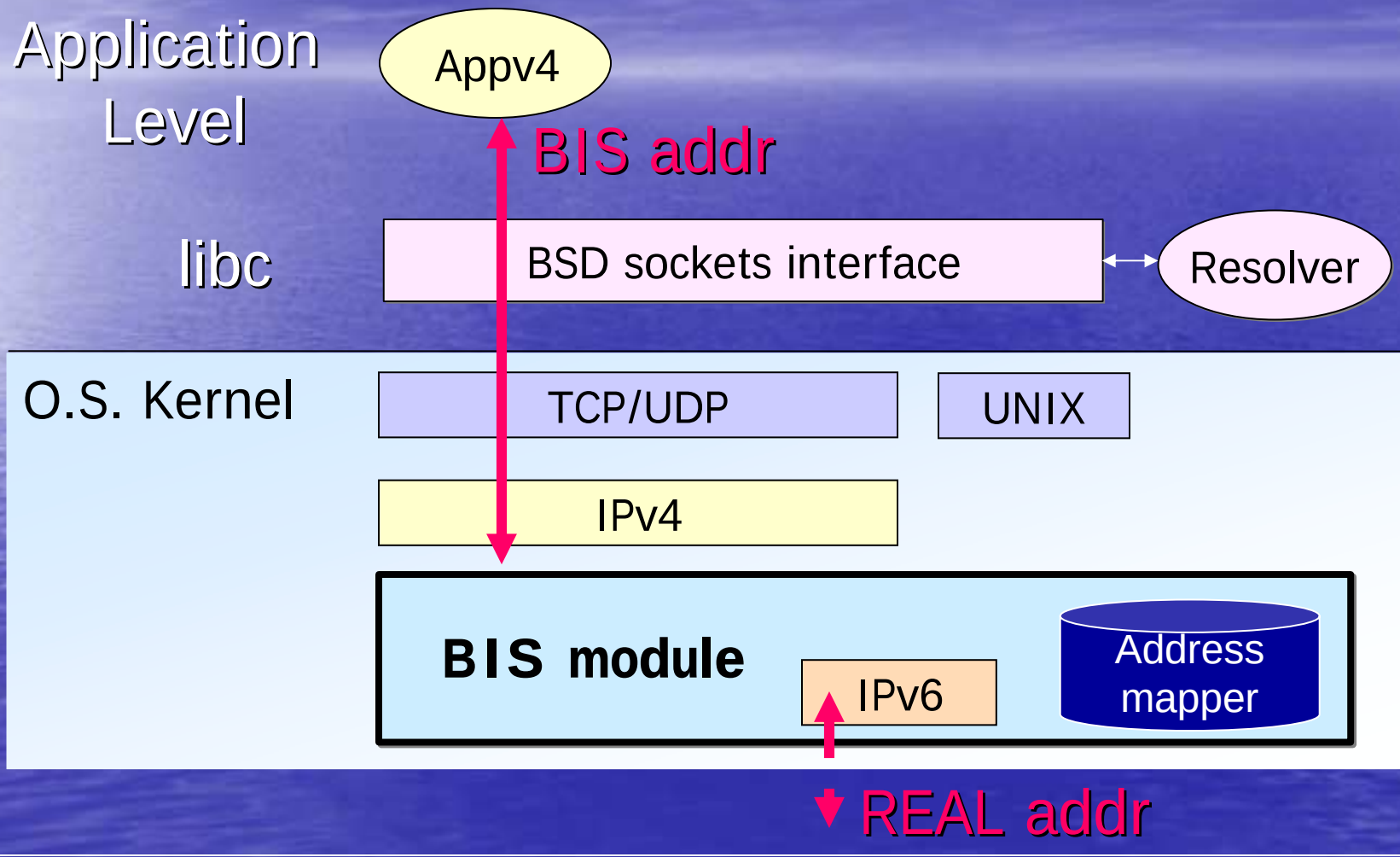
BIA – socket calls



BIS – Bump In the Stack

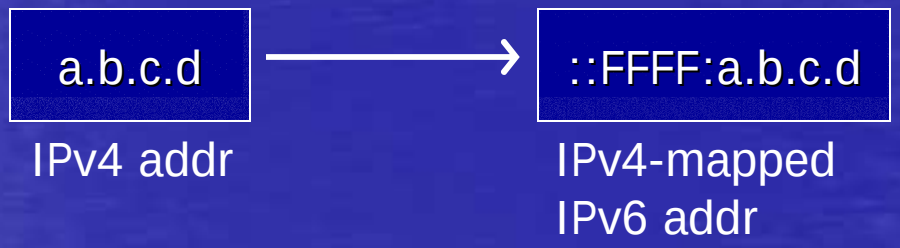


BIS – Bump In the Stack

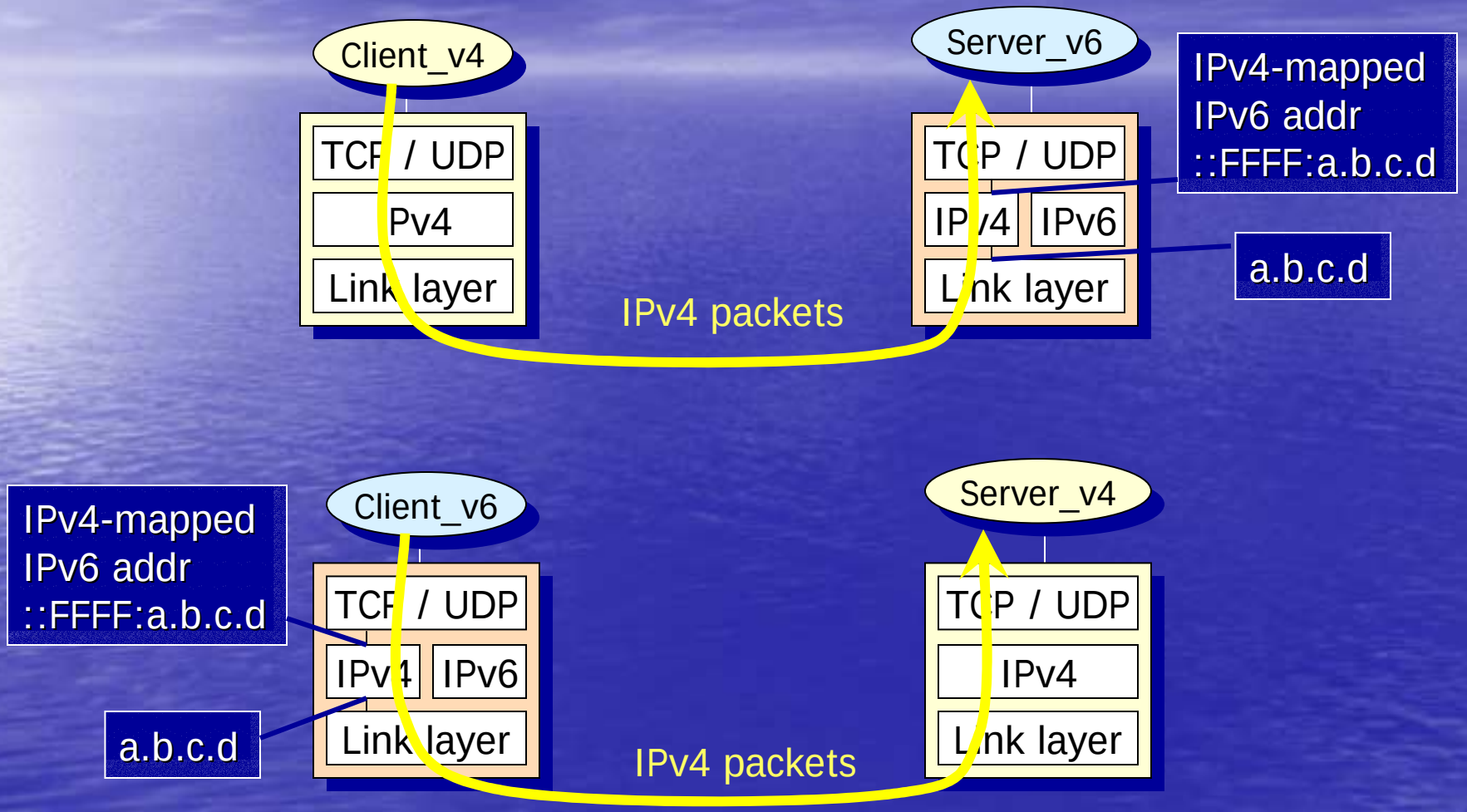


2. IPv6 applications in dual-stack nodes

- IPv6 source code dependencies
- Use IPv6
- If IPv4 communication is required:
  - change source to dual application, or
  - two different applications: appv4 & appv6, or
  - use IPv4-mapped IPv6 addresses
    - Built from IPv4 addresses



IPv6 Apps – IPv4-mapped IPv6 addr



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

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

3. Dual applications in dual-stack nodes

- Applications use both protocols.
- Ways of implementation:
  - Applications request IPv6 addresses related to a host name if not valid, applications request IPv4 addresses.

    IP VERSION DEPENDENT SOURCE CODE

  - Applications request all addresses related to a host name, and try connection with the list of addresses until connection succeeds.

    IP VERSION INDEPENDENT SOURCE CODE

4. Dual applications in IPv4 nodes

- Write dual applications which can also run in IPv4-only nodes:
  - Try first AF_INET6 socket and the AF_INET.
  - If kernel does not have IPv6 support, the socket call will return an error.
  - Try AF_INET even socket call returned an error.

Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

Application Porting Considerations

IP version dependencies in applications:

- Presentation format for an IP address
- Transport layer API
- Name and address resolution
- Specific IP dependencies

1. Presentation format for an IP addr

IP addresses are usually provided in a presentation format, string: "10.0.0.1"

PROBLEMS

- Allocated memory to store the string could be not enough.
- IPv4 uses "." as separator, IPv6 uses ":". Parsers should be complaint with both formats.
- Ambiguity using ":" character in URLs: http://[IPv6Address]:portNumber

RECOMMENDATION

- Use FQDN

2. Transport layer API

- Network information storage.
- Address conversion functions.
- Communication API functions.
- Network configuration options.

PROBLEMS

- IPv4 Network API makes visible IP version dependencies, functions and structures must be changed

RECOMMENDATION

- Develop IP version independent applications

3. Name and address resolution

- Two basic resolution functions
- DNS queries/responses are sent using IPv4/IPv6, regardless data records.

PROBLEMS

- Existing IPv4 name/address resolution calls are not valid for IPv6.

RECOMMENDATION

- Use new IP version independent structures and functions

4. Specific IP dependencies

- IP address selection.
- Application framing.
- Storage of IP addresses.

PROBLEMS

- There are more IP dependencies apart from network API calls and structures.

RECOMMENDATION

- Review source code in detail

Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

BSD Socket API

IPv6 addresses are 128 bit long.

Changes in the application networking part

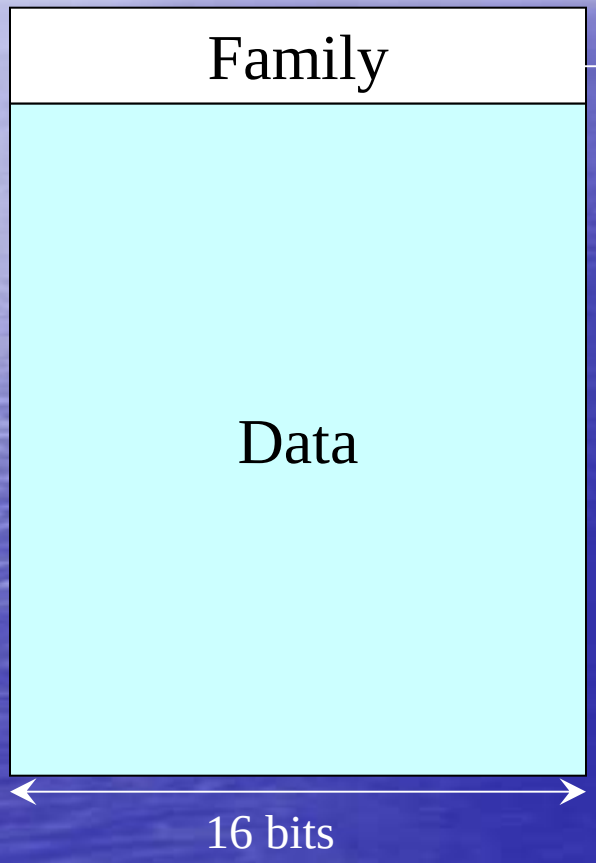
1. Data structures:
Socket address struct

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

3. Socket calls

Generic Socket Address

sockaddr

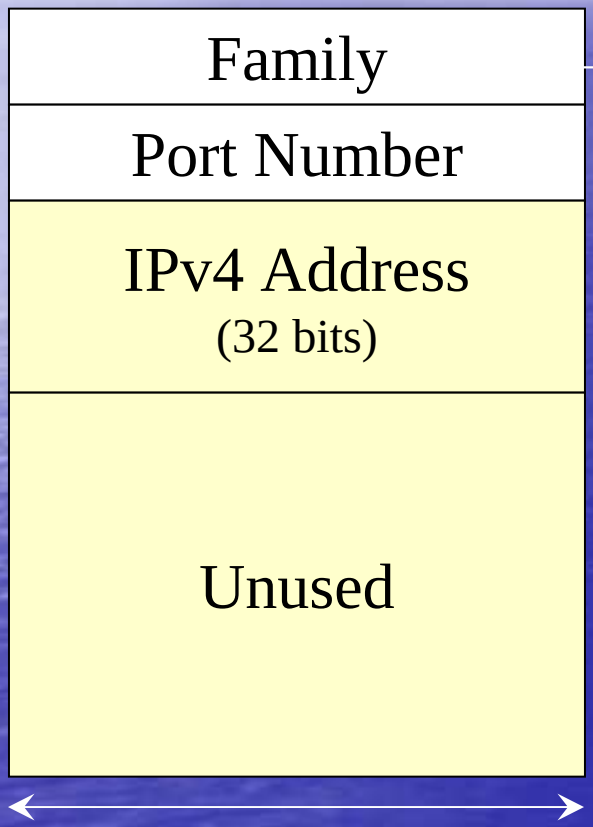


AF_XXXX

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

IPv4 socket address structure

sockaddr_in



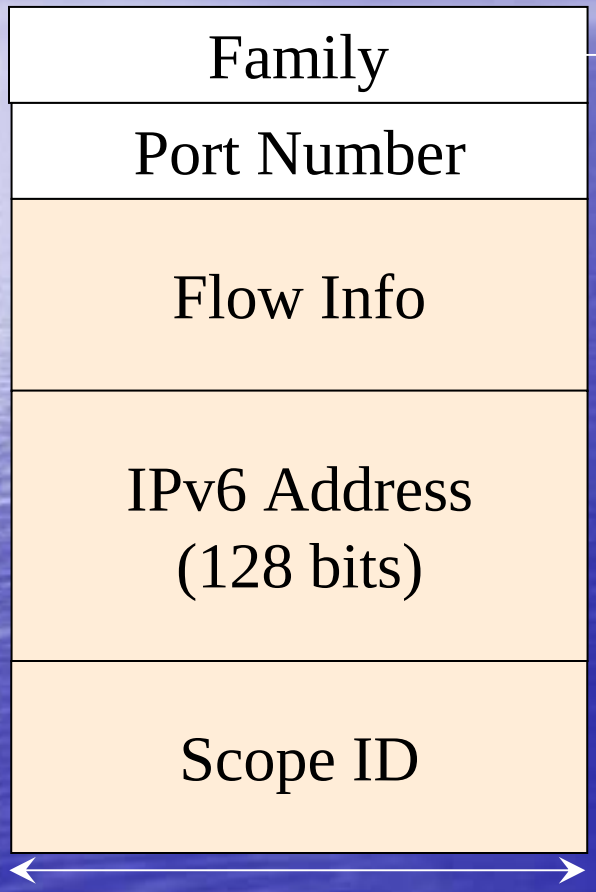
AF_INET

```
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



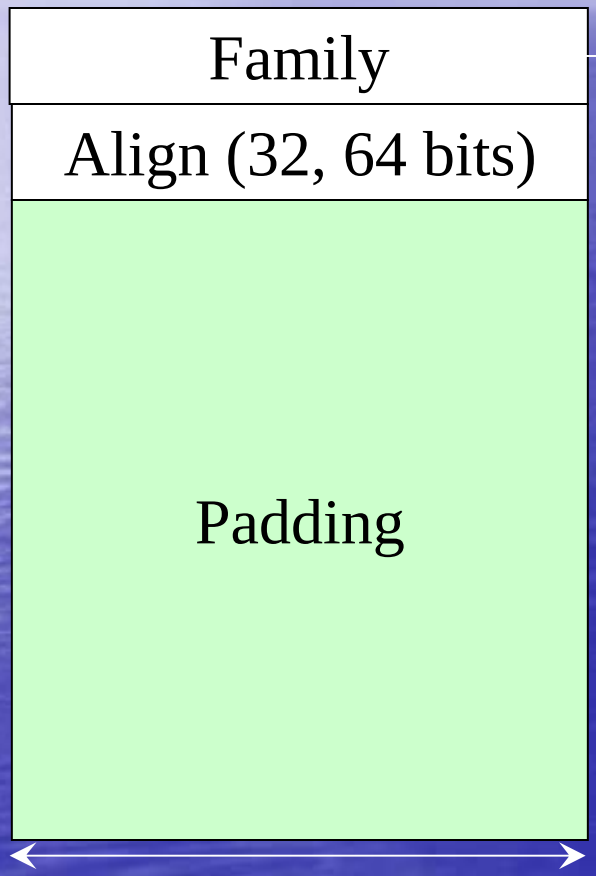
AF_INET6

```
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];
};
```

Structure Allocation

IPv4-only

```
struct sockaddr_in serverAddr;
/* … */
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

IPv6-only

```
struct sockaddr_in6 serverAddr;
/* … */
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

Protocol independent

```
struct sockaddr_storage serverAddr;
/* … */
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

BSD Socket API

IPv6 addresses are 128 bit long.

Changes in the application networking part

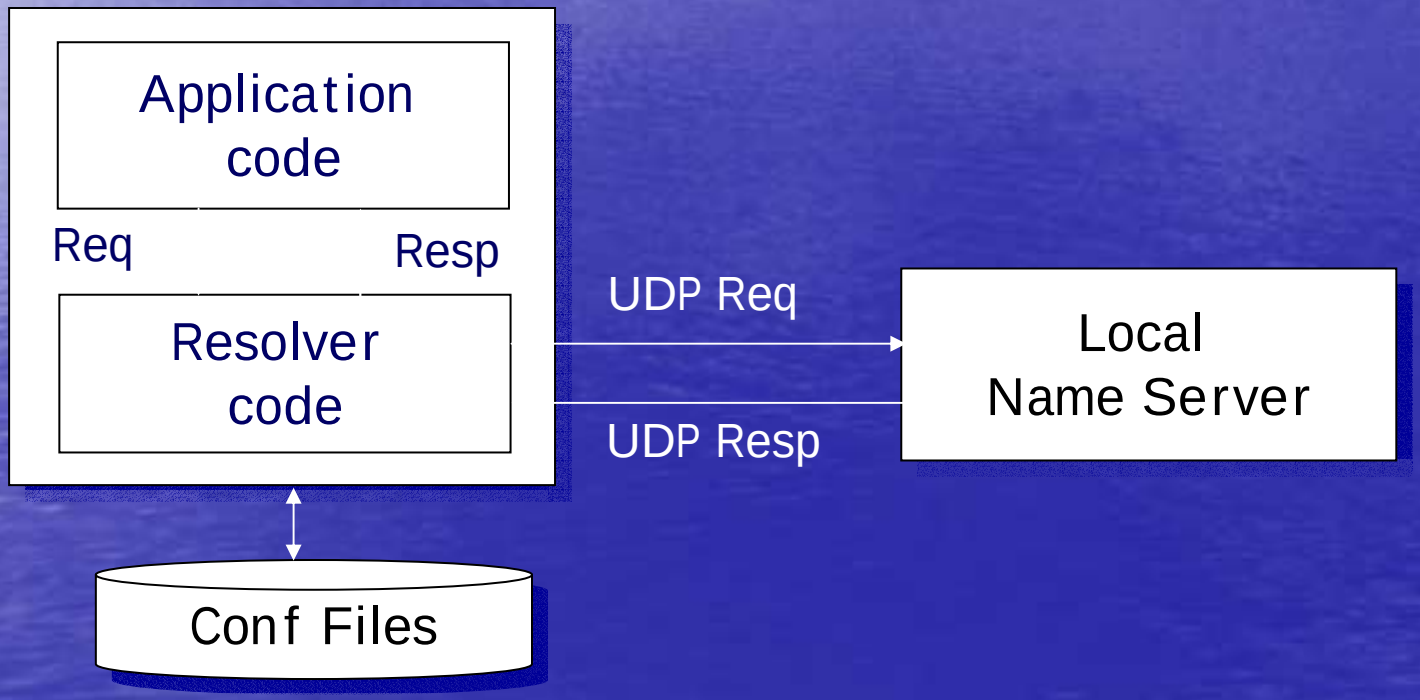
1. Data structures:
   Socket address struct

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

3. Socket calls

Conversion functions

- RESOLVER: returns IP address structure.



Conversion: name & IP addr

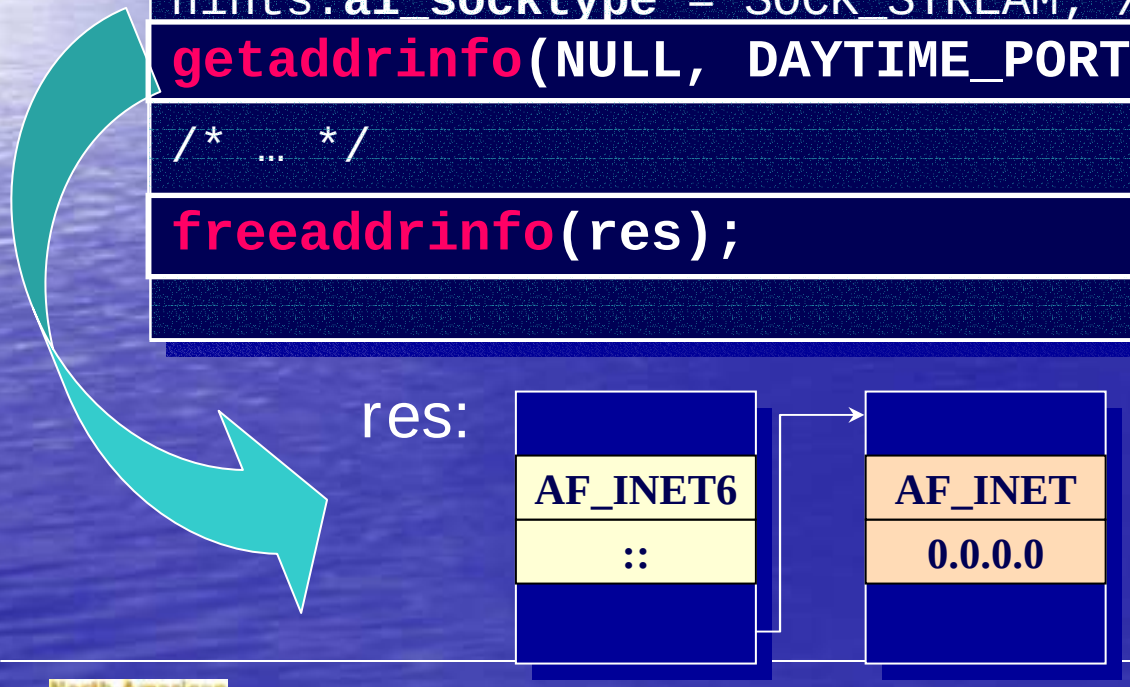
IPv4 only (v4 addr)	IPv4 & IPv6 (v4, v6, v4-map v6 addr)
`gethostbyname()`	`getaddrinfo()`
`gethostbyaddr()`	`getnameinfo()`

Protocol Independent Functions

Getaddrinfo

```
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);
/* … */
freeaddrinfo(res);
```



Getnameinfo

```
struct sockaddr_storage clientAddr;
char clientHost[ADDRLEN];
char clientPort[PORTLEN];
/* … */
connectedfd = accept(serverfd,
                     (struct sockaddr *)&clientAddr,
                     &alen);
getnameinfo((struct sockaddr *)&clientAddr, addrLen,
            clientHost, sizeof(clientHost),
            clientPort, sizeof(clientPort),
            NI_NUMERICHOST);

printf("Request from host=[%s] port=[%s]\n",
       clienthost, clientservice);
}
```

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()`

inet_pton

IPv4-only code

```
struct in_addr addr4;
char *addrStr4="127.0.0.1";
/* … */
inet_aton(addrStr4,
          &addr4);
```

IPv6-only code

```
struct sockaddr_in6 addr6;
char *addrStr6="::1";
/* … */
inet_pton(AF_INET6, addrStr6,
          &addr6);
```

IPv4/IPv6 code

```
struct sockaddr_storage addr;
int family = AF_INET6;
char *addrStr="::1"
/* … */
inet_pton(family, addrStr, &addr);
```

inet_ntop

Old IPv4-only code

```
struct in_addr addr4;
char *addrStr4;
/* … */
addrStr4 = inet_ntoa(addr4);
```

New IPv6-only code

```
struct sockaddr_in6 addr6;
char addrStr6[INET6_ADDRSTRLEN];
/* … */
inet_ntop(AF_INET6, addr6,
          addrStr6,
          INET6_ADDRSTRLEN);
```

IPv4/IPv6 code

```
struct sockaddr_storage addr;
char addrStr[ADDRSTRLEN];
/* … */
inet_ntop(addr.ss_family, addr, addrStr, sizeof(addrStr));
```

BSD Socket API

IPv6 addresses are 128 bit long.

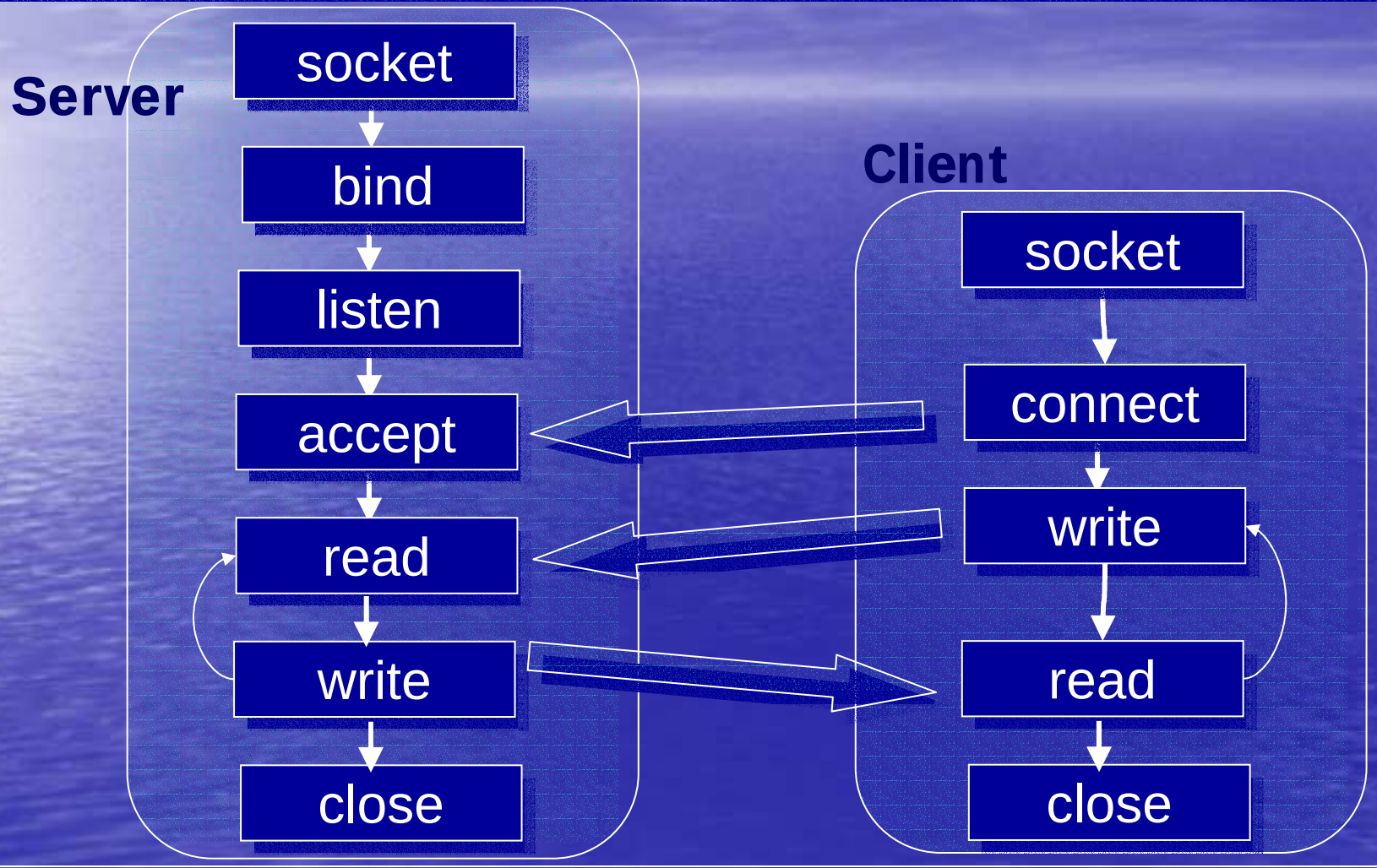
Changes in the application networking part

1. Data structures:
   Socket address struct

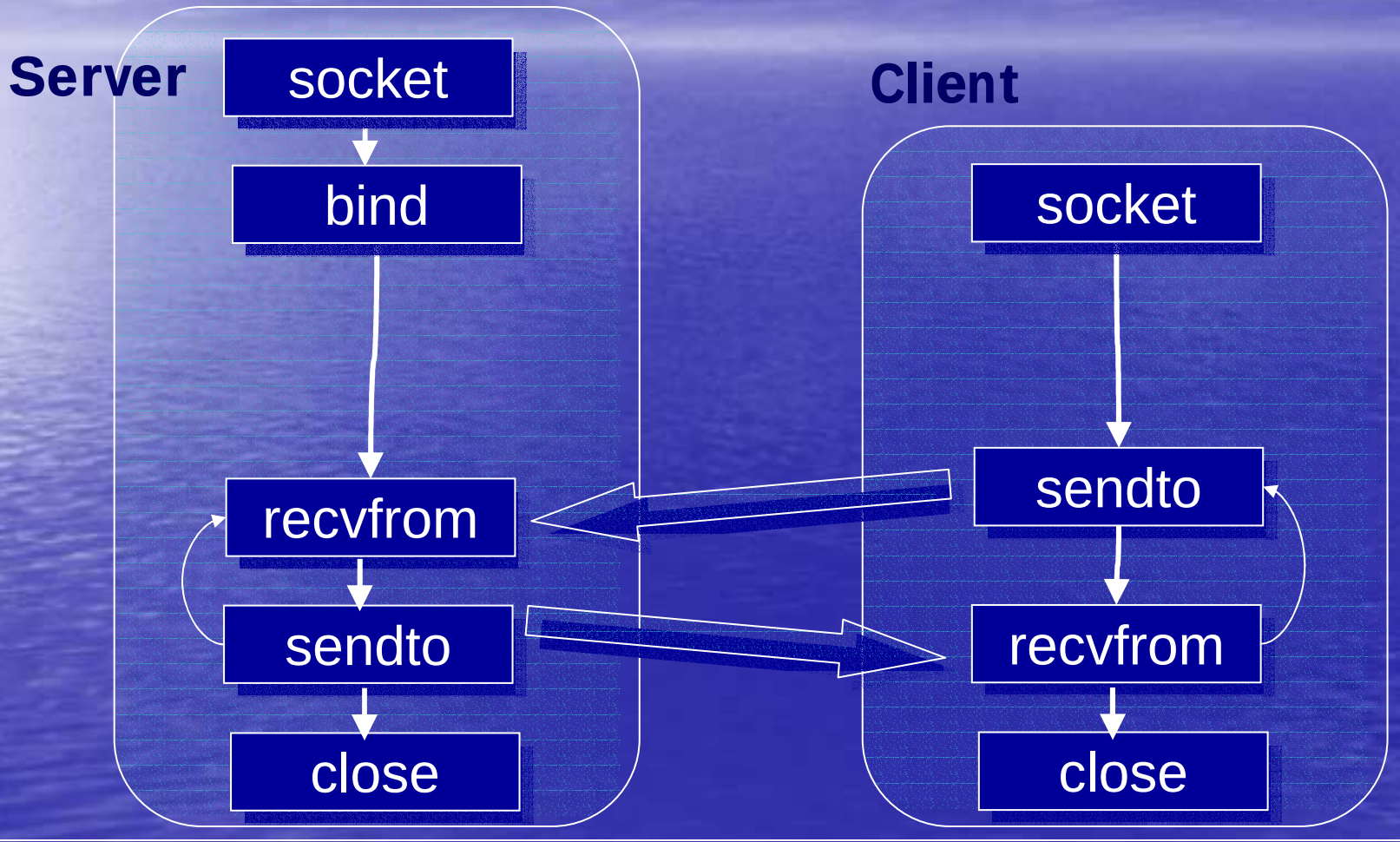
2. Conversion functions between:
   - String and binary representation
   - Name and address

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);
```

- Casting to generic socket structure **struct sockaddr *.**

```
IPv4:                  from (struct sockaddr_in *)
IPv6:                  from (struct sockaddr_in6 *)
Protocol independent:  from (struct sockaddr_storage *)
```

- Size of socket address structure.

Socket Options (setsockopt)

- IPV6_UNICAST_HOPS
- Multicast:
  - IPV6_MULTICAST_IF
  - IPV6_MULTICAST_HOPS
  - IPV6_MULTICAST_LOOP
  - IPV6_JOIN_GROUP
  - IPV6_LEAVE_GROUP
- IPV6_V6ONLY for AF_INET6

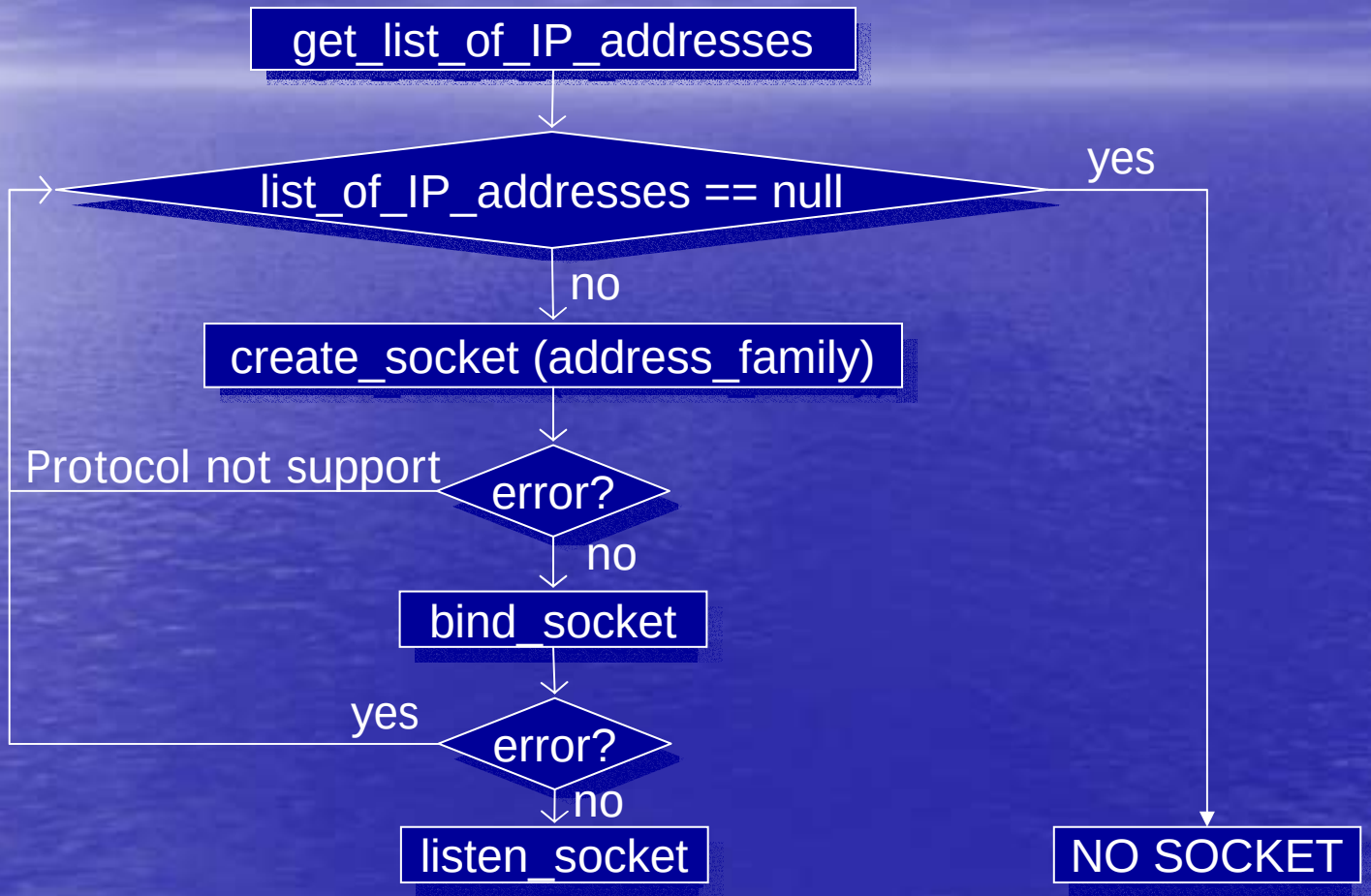
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

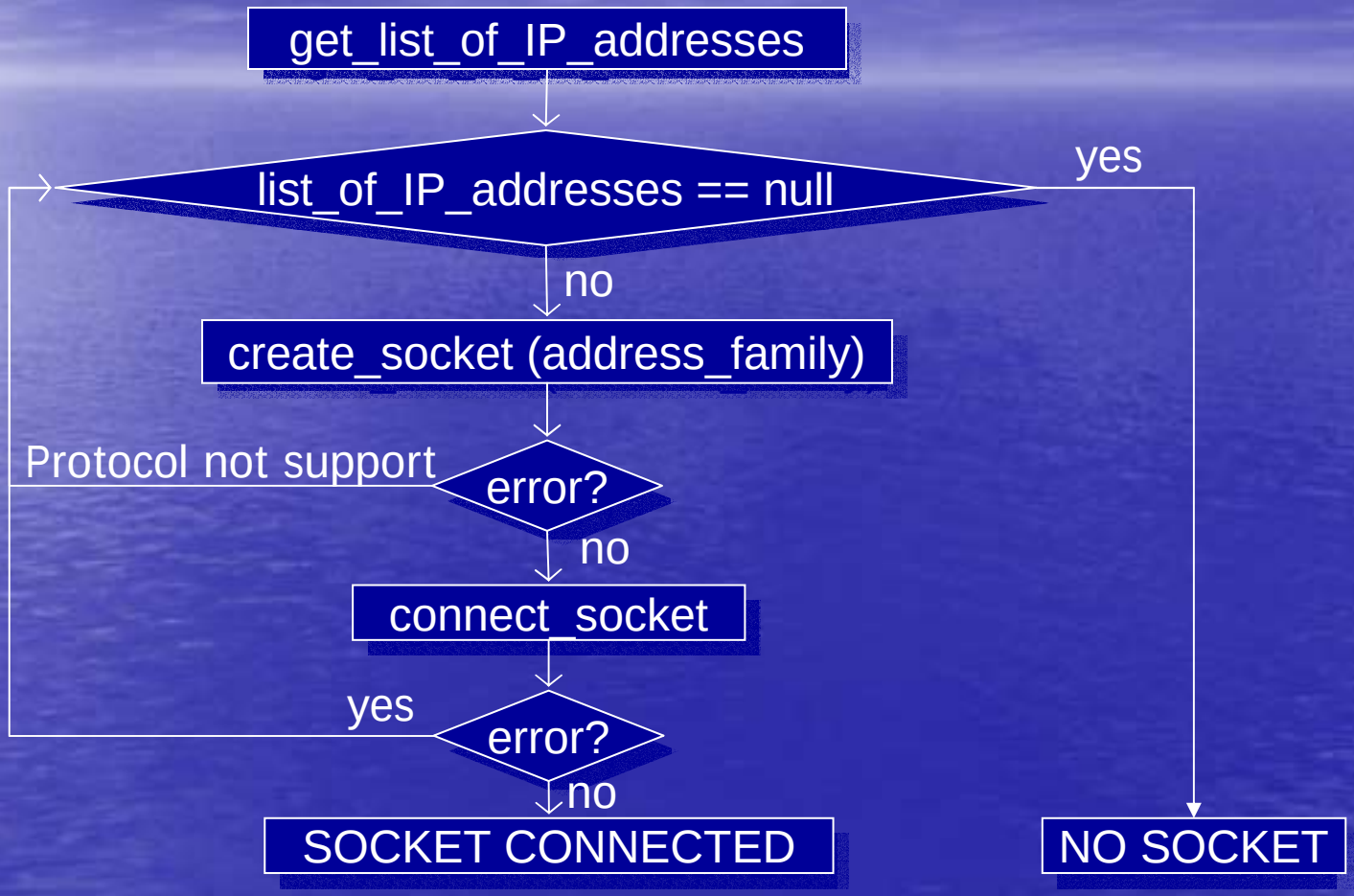
IP Version Independent Applications

- Use IP version independent structures:
  - sockaddr_storage
- Use IP version independent functions:
  - getaddrinfo()/getnameinfo()
- Not use inet_ntop()/inet_pton()
- Iterated jobs for finding the working address:
  - Server:
    - listening packets addressed to a specific port.
  - Clients:
    - connecting to one of the server addresses.

Server example



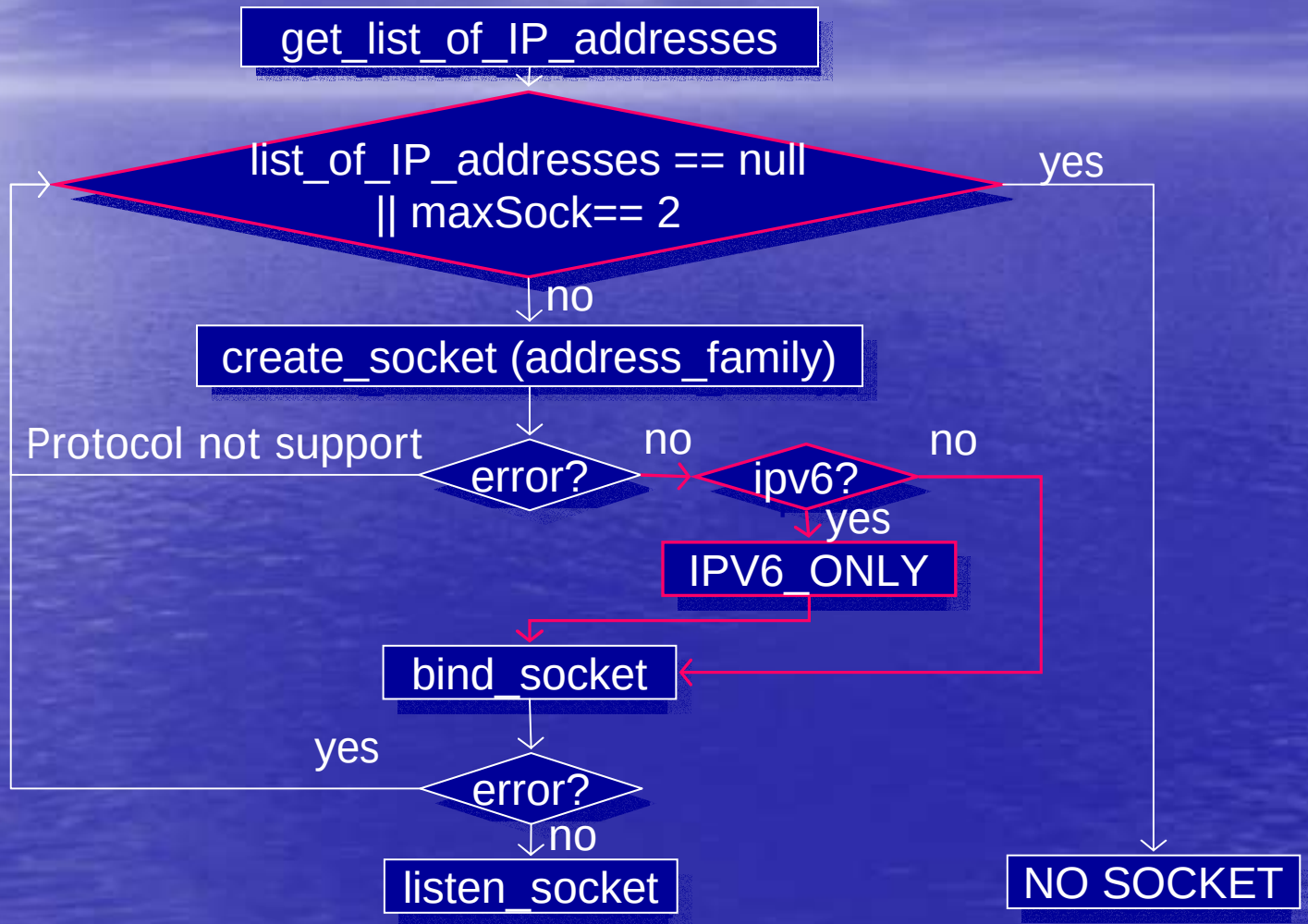
Client example



Server applications supporting IPv4 & IPv6

- IPv6 servers :
  - IPv4 clients connect using an IPv4-mapped IPv6 address
  - IPv6 clients connect using an IPv6 address
- Dual servers: servers use different sockets for IPv4 and IPv6 connections (IPV6_ONLY):
  - IPv4 clients connect using an IPv4 address
  - IPv6 clients connect using an IPv6 address

Dual servers: different IPv4 & IPv6 sockets



Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations

Recommendations for applications

- Not use hard coded IP addresses
- IP version independent applications
  - Structures & functions
  - Iterated jobs to find a valid connection
- Develop dual applications:
  - Valid for both protocols IPv4 & IPv6