

Porting Applications

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

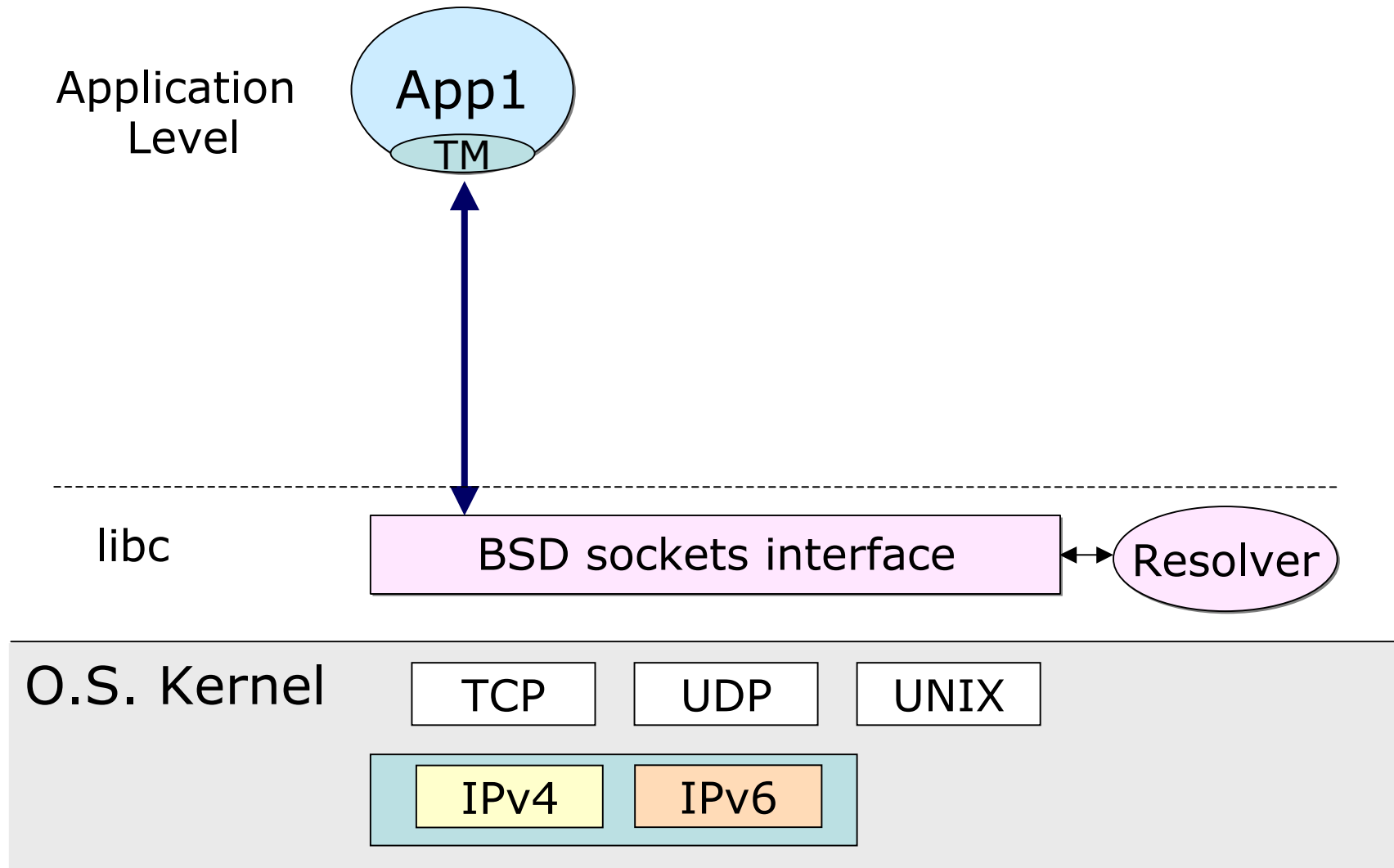
Systems and Communications Group (GSyC)
Experimental Sciences and Technology
Department (ESCET)
Rey Juan Carlos University (URJC)



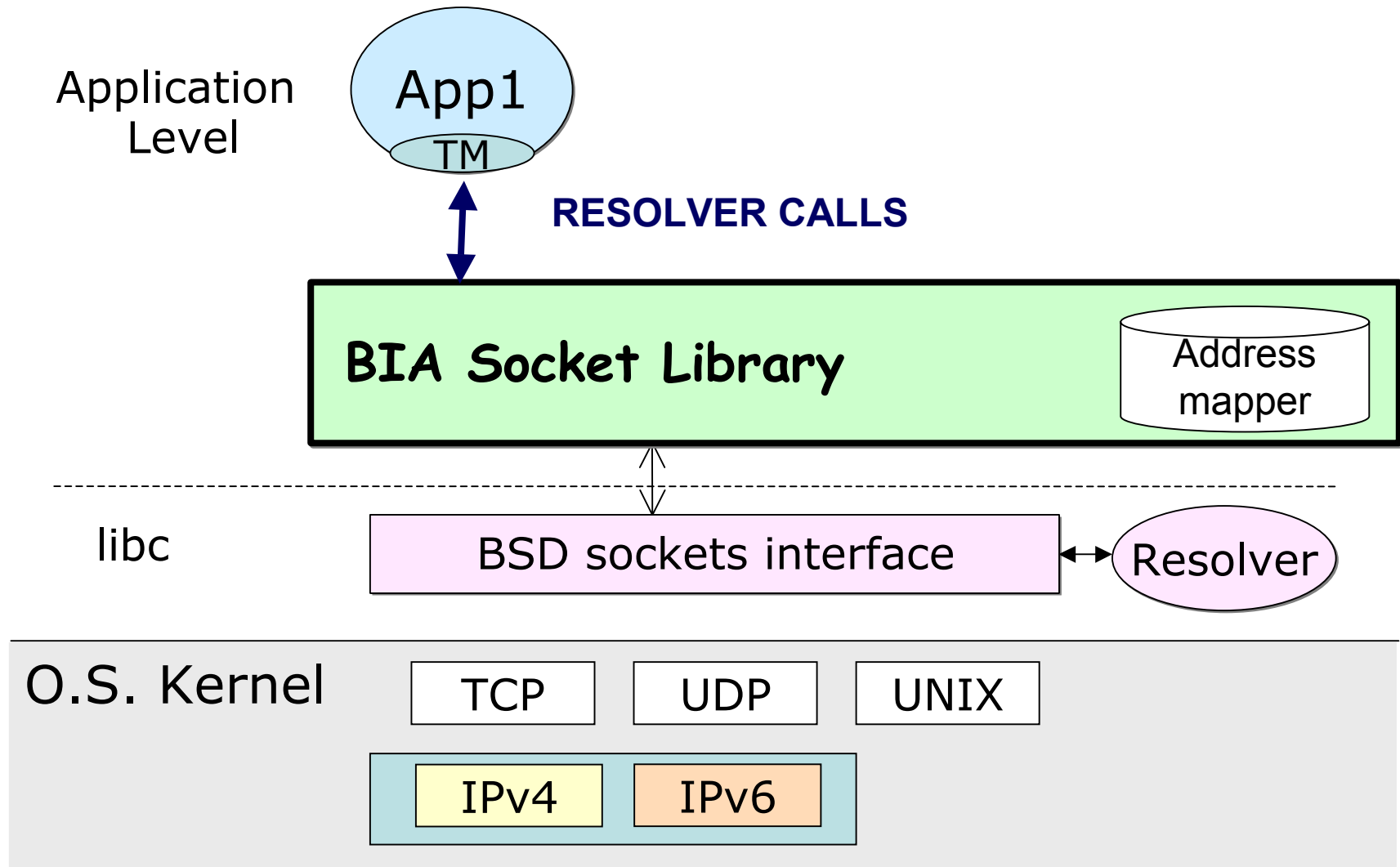
Agenda

- Not change applications, nor kernel.
- Porting applications
 - Identifying IP version dependencies
 - Porting source code
- IPv4/IPv6 Interoperability
- Recommendations

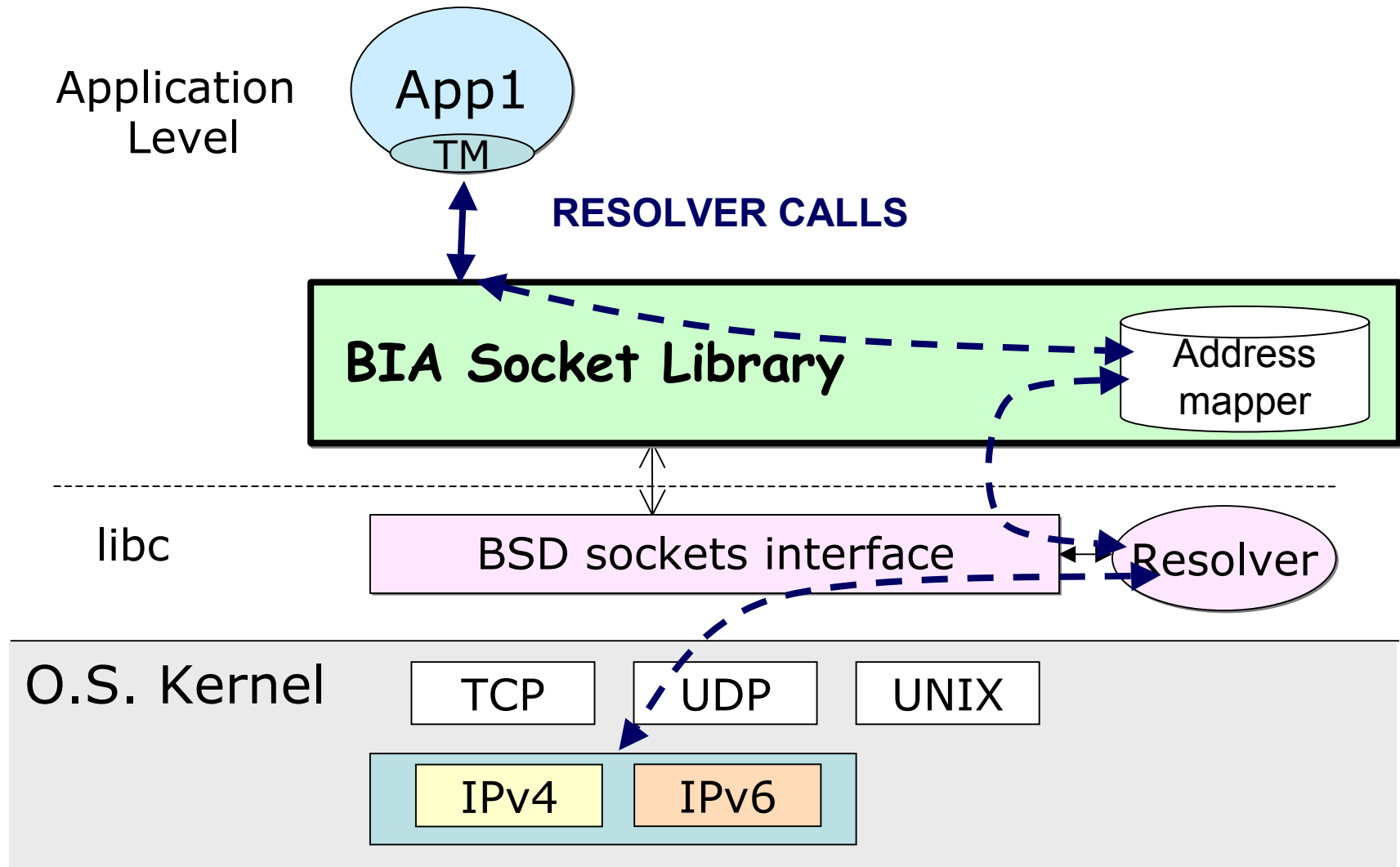
Analyzing existing applications



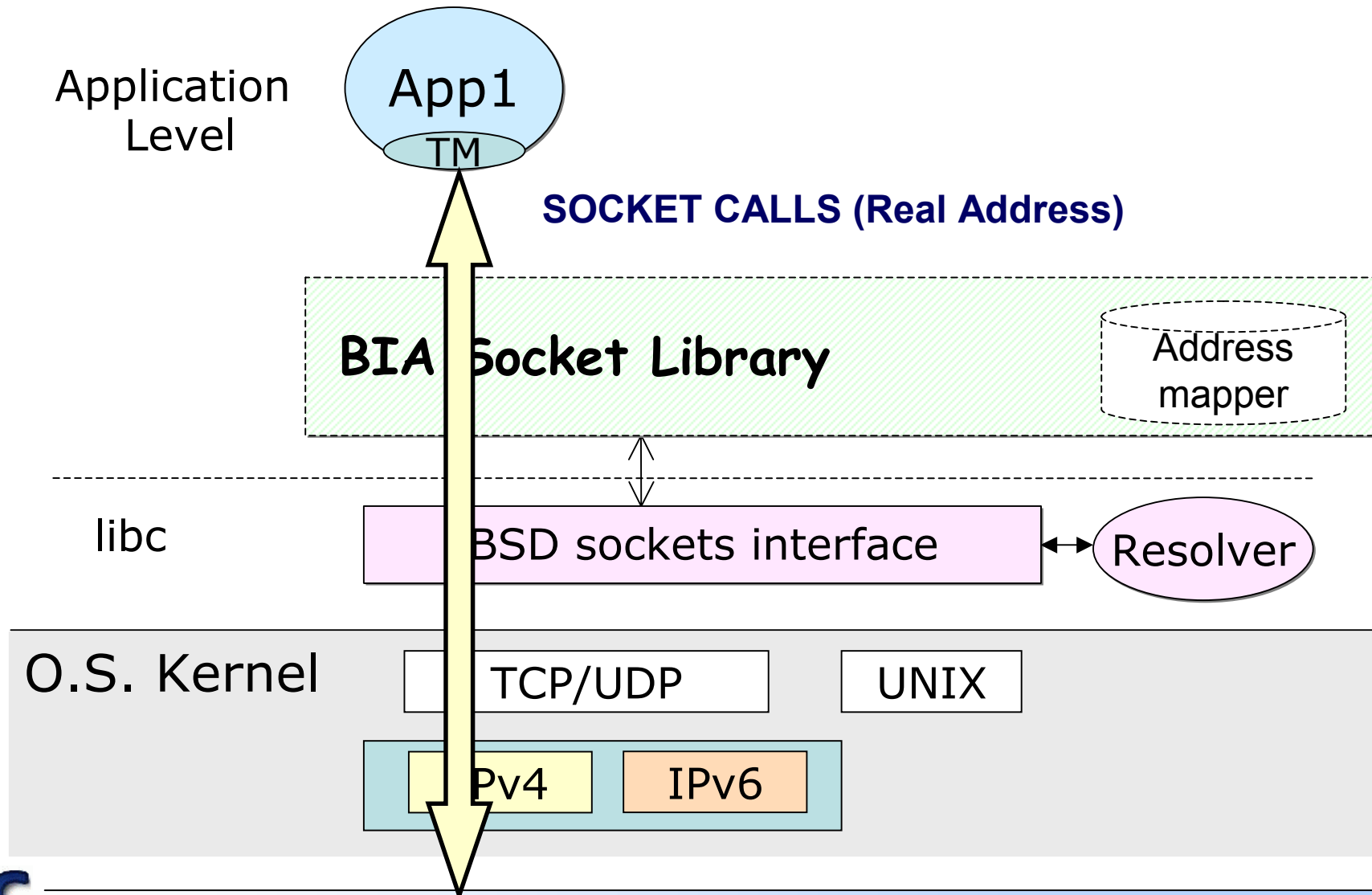
Not change applications, nor kernel



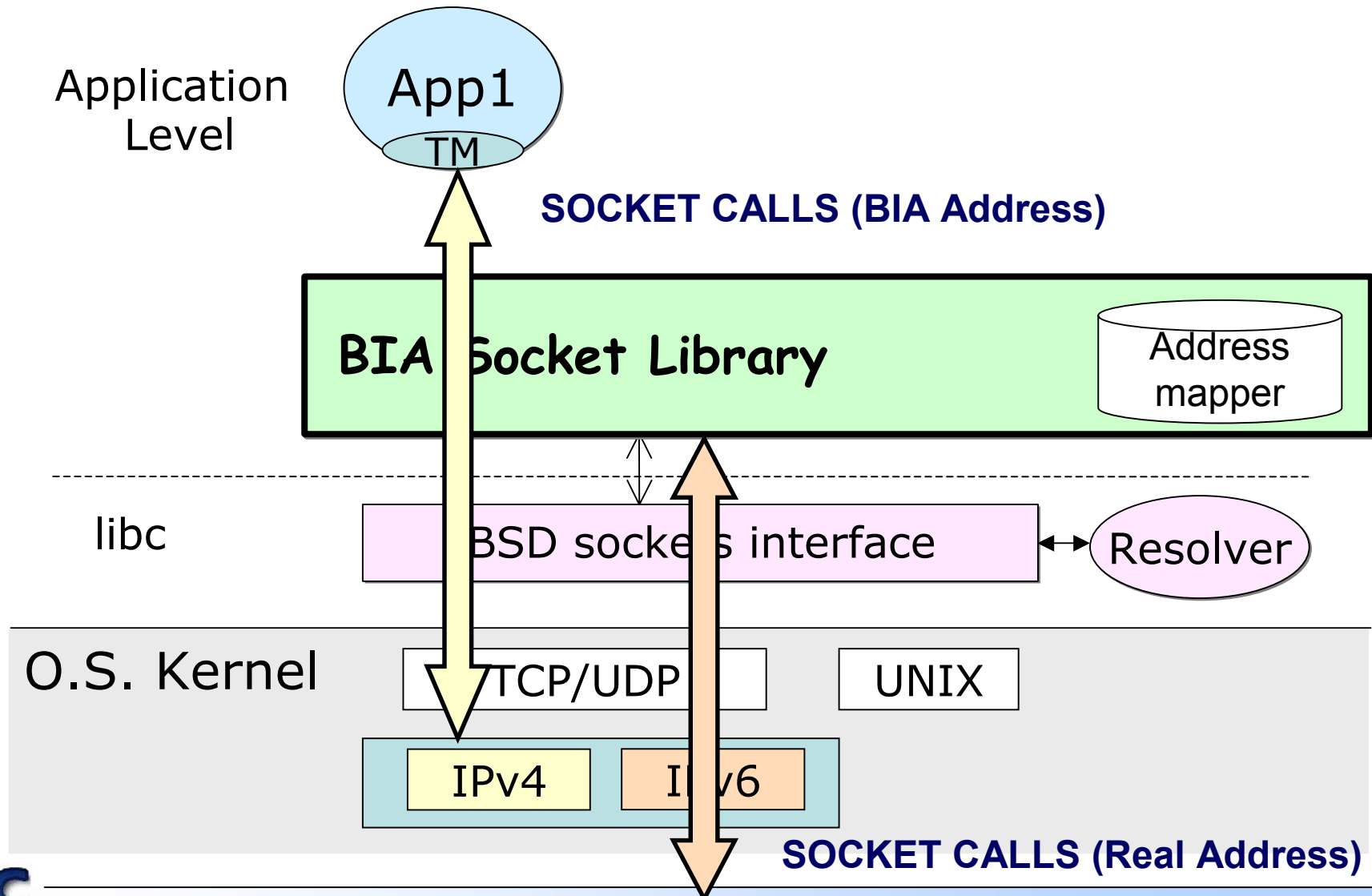
Resolver calls



IPv4 communication



IPv6 communication



Agenda

- Not change applications, nor kernel.
- Porting applications
 - Identifying IP version dependencies
 - Porting source code
- IPv4/IPv6 Interoperability
- Recommendations

Identifying IP dependencies

1. Network addresses (FQDN & IP address)
2. Application transport module
 - Data structures
 - Conversion functions
 - Socket calls & options
 - Special addresses
 - Default address selection
3. ADU Fragmentation
4. Register systems based on IP addresses

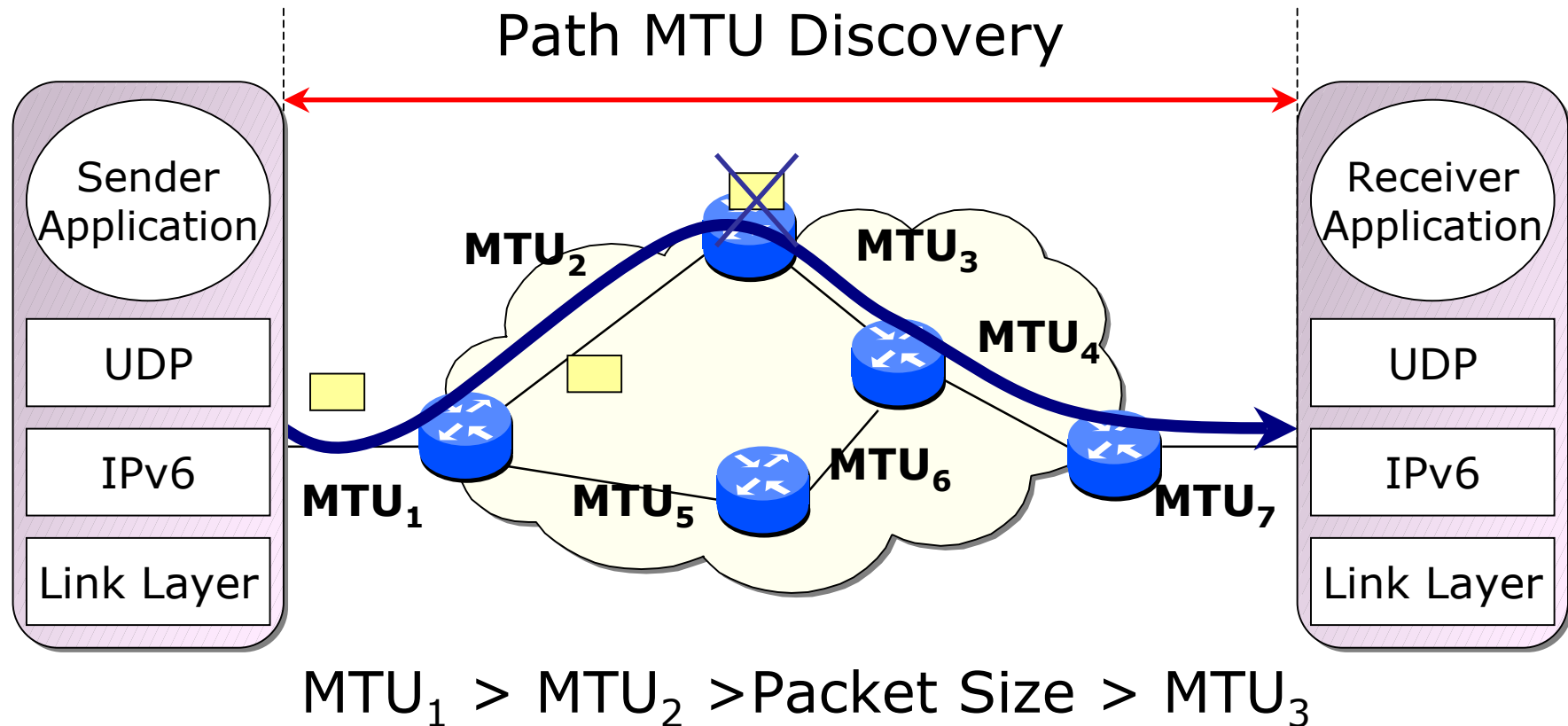
1. Network addresses (FQDN & IP address)

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

2. Application transport module

- Data structures
- Resolution & conversion functions
- Socket calls & options
- Special addresses:
 - Wildcard Address: `INADDR_ANY` / `IN6ADDR_ANY_INIT`
 - Loopback Address: `INADDR_LOOPBACK` / `IN6_ADDR_LOOPBACK_INIT`
 - Broadcast Address: `INADDR_BROADCAST` / `MULTICAST?`
- Default address selection (RFC 3484)
 - Source
 - Destination

3. ADU fragmentation



4. Register Systems

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.

Agenda

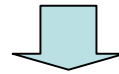
- Not change applications, nor kernel.
- Porting applications
 - Identifying IP version dependencies
 - Porting source code
- IPv4/IPv6 Interoperability
- Recommendations

Porting source code

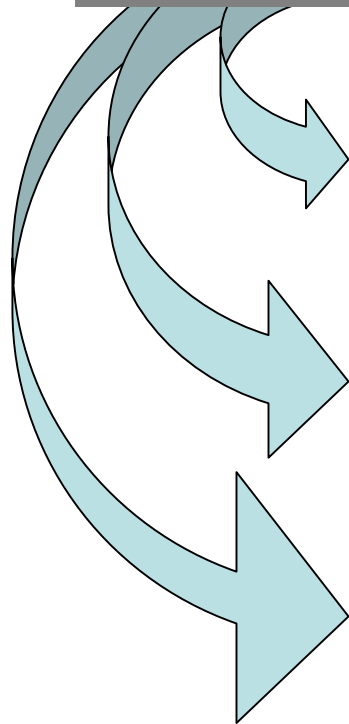
1. Porting the application networking part to manage IPv6 addresses. Using socket interface:
 - RFC 3493** (Feb 2003): Basic socket interface extensions for IPv6 (2553 obsoleted).
 - RFC 2292**: Advanced sockets API for IPv6.
2. Reengineering the application to use new features in addition to larger address space:
 - QoS: Flow labels and priorities.
 - 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

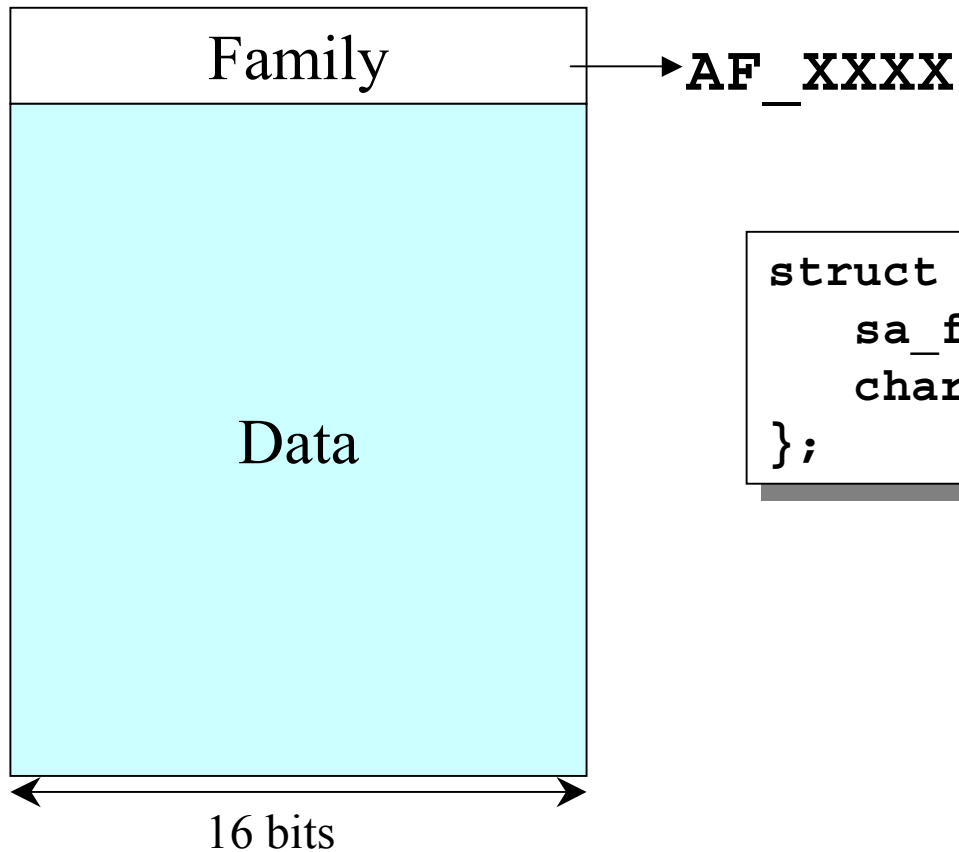
2. Conversion functions between:

- String and binary representation
- Name and address

3. Socket calls

Generic socket address

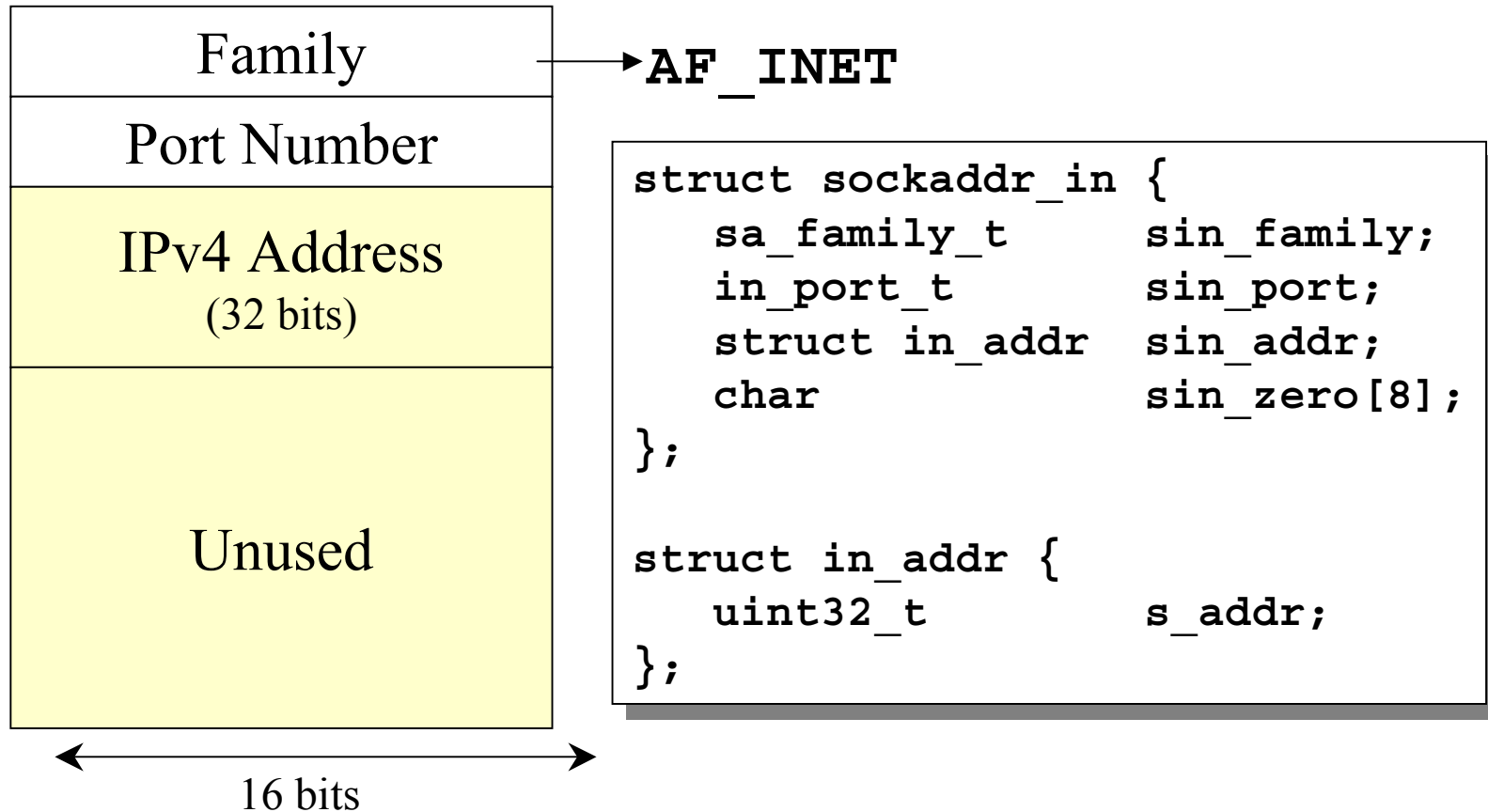
sockaddr



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

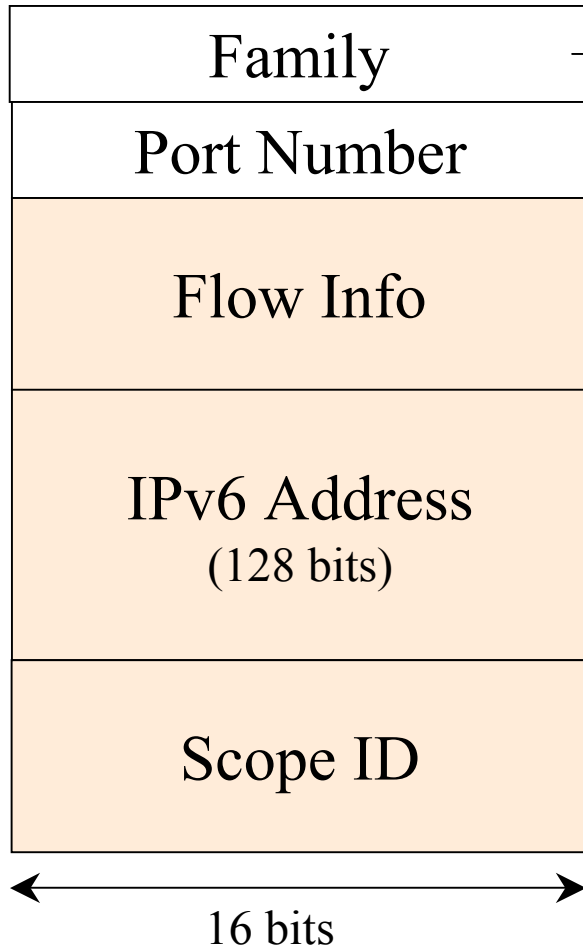
IPv4 socket address structure

sockaddr_in



IPv6 socket address structure

sockaddr_in6

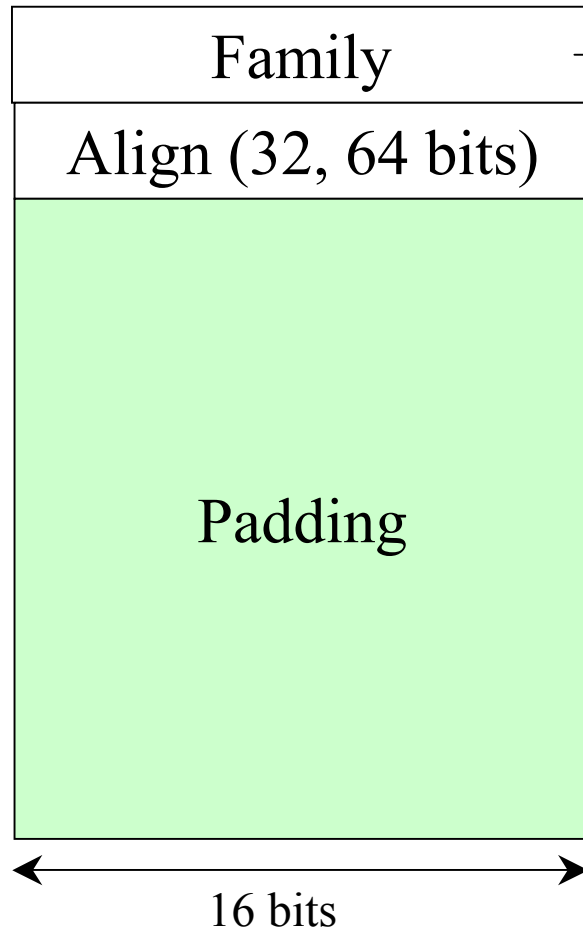


```
struct sockaddr_in6 {
    sa_family_t      sin6_family;
    in_port_t        sin6_port;
    uint32_t         sin6_flowinfo;
    struct in6_addr  sin6_addr;
    uint32_t         sin6_scope_id;
};

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

Protocol independent structure

sockaddr_storage



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

Structure Allocation

IPv4-only code

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

IPv6-only code

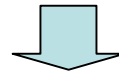
```
struct sockaddr_in6 serverAddr;  
  
/* ... */  
  
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

Protocol independent code

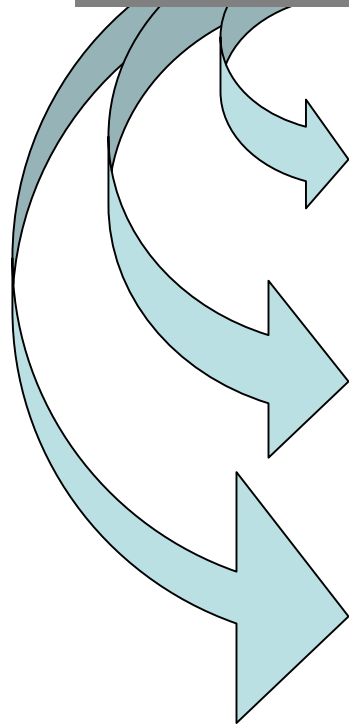
```
struct sockaddr_storage serverAddr;  
  
/* ... */  
  
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

BSD socket interface extensions

IPv6 addresses are 128 bit long.



Changes in the application networking part



1. Data structures:
Socket address struct

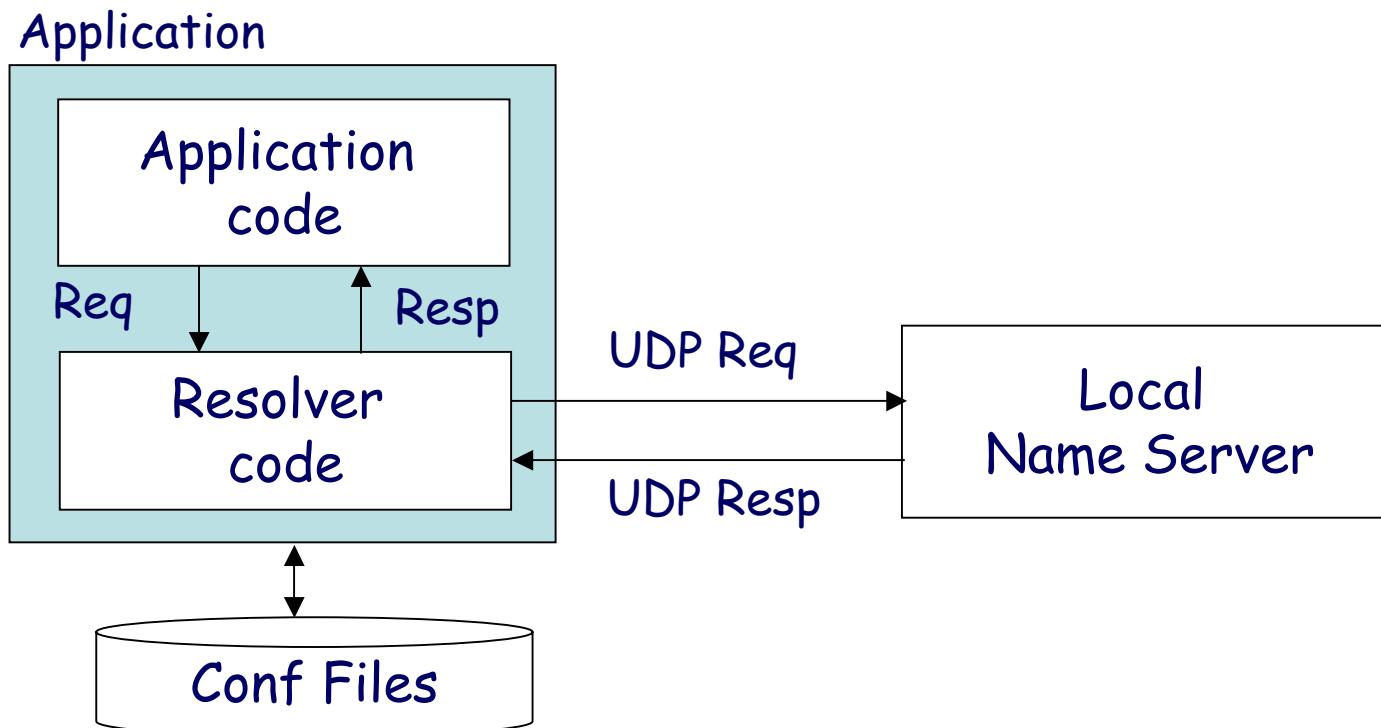
2. Conversion functions between:

- String and 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)
<code>gethostbyname()</code>	<code>getaddrinfo()</code> <code>getnameinfo()</code>
<code>gethostbyaddr()</code>	

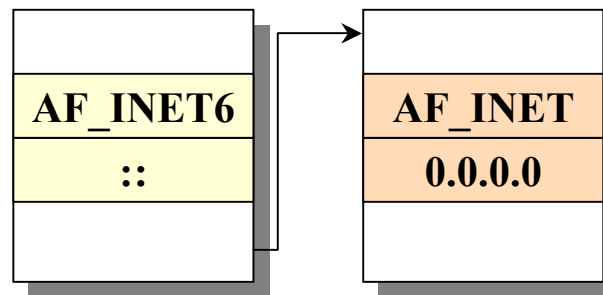
Protocol
Independent
Functions

Getaddrinfo

* Protocol independent function.

```
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);
```

res:



Getnameinfo

* Protocol independent function.

```
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	<code>inet_aton()</code> <code>inet_addr()</code>	<code>inet_pton()</code>
Binary -> String	<code>inet_ntoa()</code>	<code>inet_ntop()</code>

inet_pton

IPv4-only code

```
struct in_addr addr4;  
char *addrStr4="127.0.0.1";  
  
/* ... */
```

```
inet_pton(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);
```

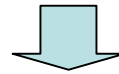
IPv4/IPv6 code

```
struct sockaddr_storage addr;  
char addrStr[ADDRSTRLEN];  
  
/* ... */
```

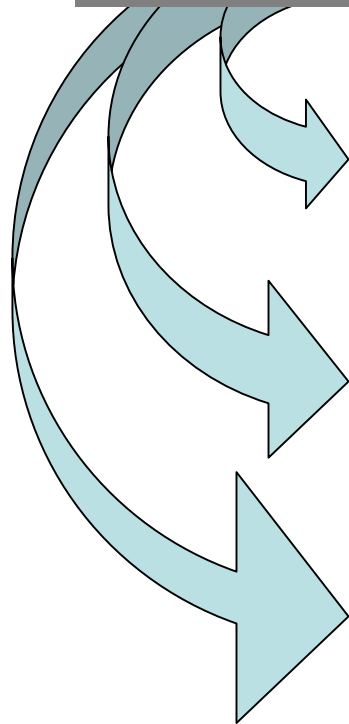
```
inet_ntop(addr.ss_family, addr, addrStr, sizeof(addrStr));
```

BSD socket interface extensions

IPv6 addresses are 128 bit long.



Changes in the application networking part



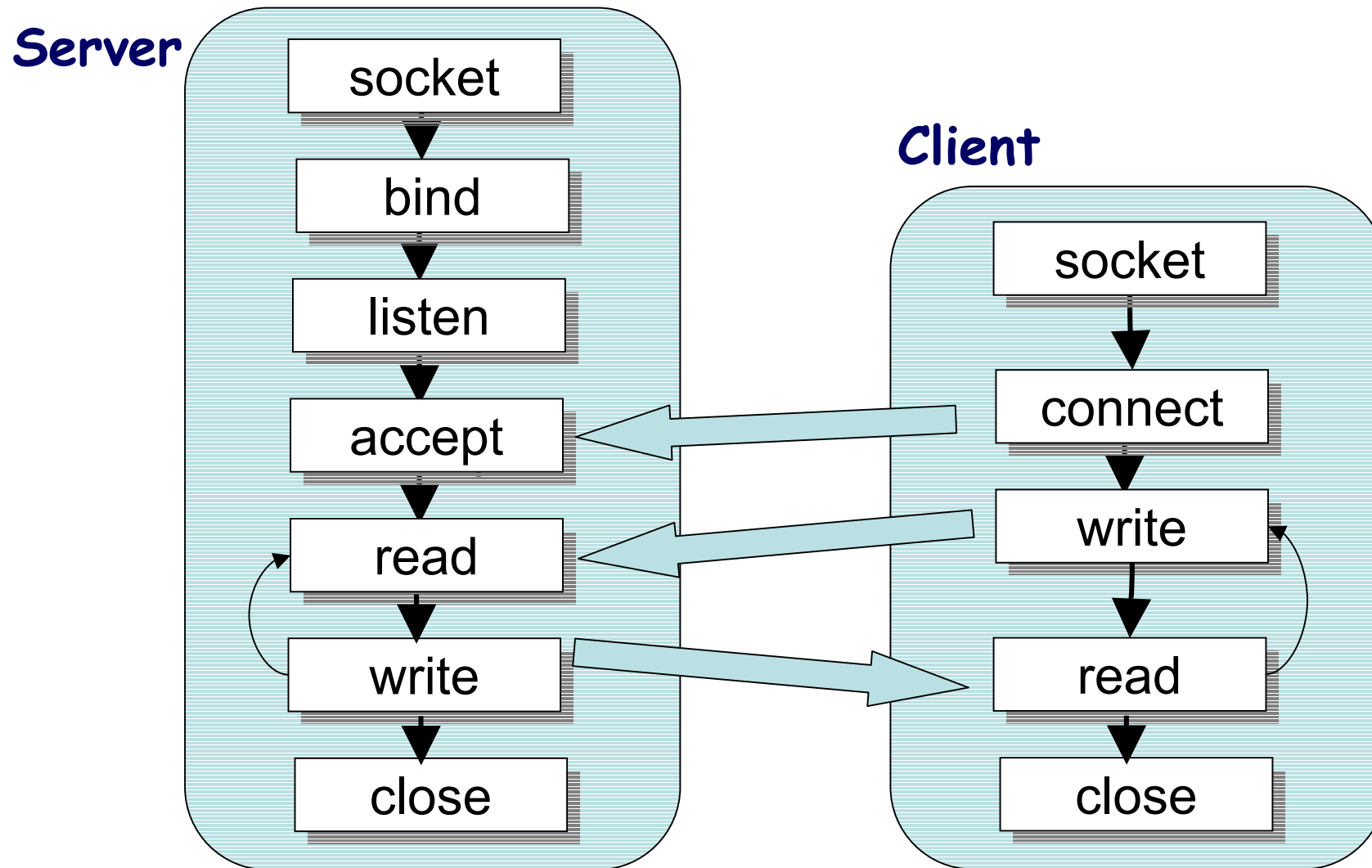
1. Data structures:
Socket address struct

2. Conversion functions between:

- String and binary representation
- Name and address

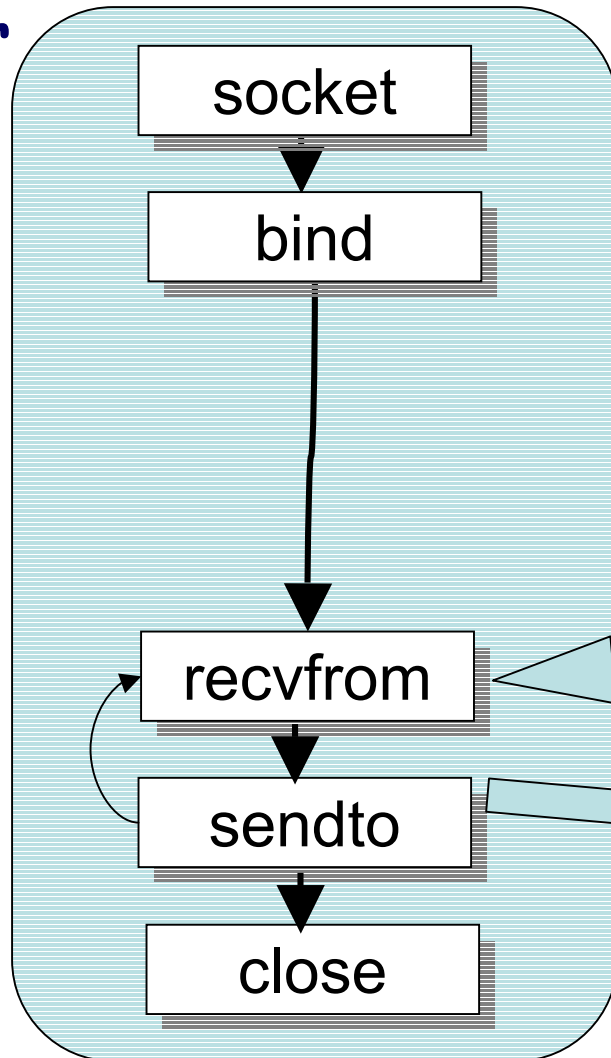
3. Socket calls

Socket calls (TCP)

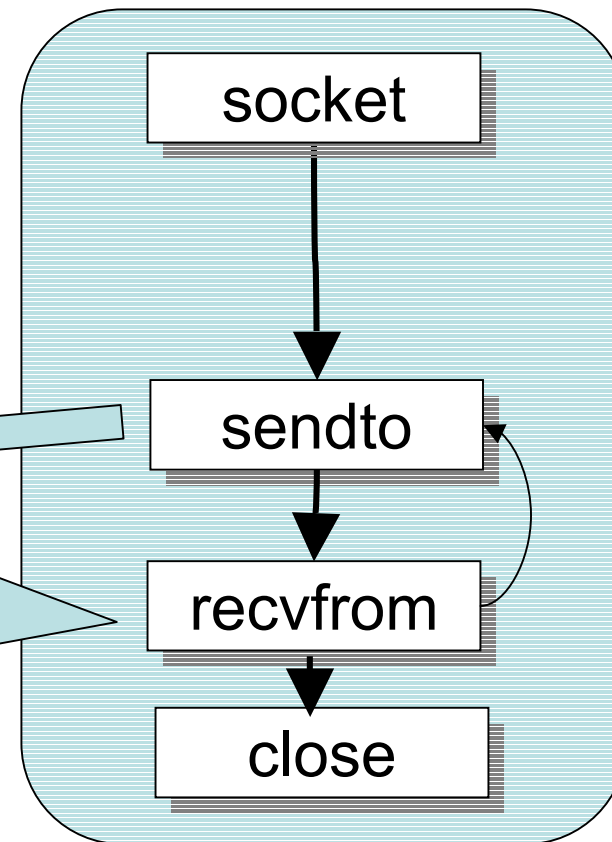


Socket calls (UDP)

Server



Client



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)

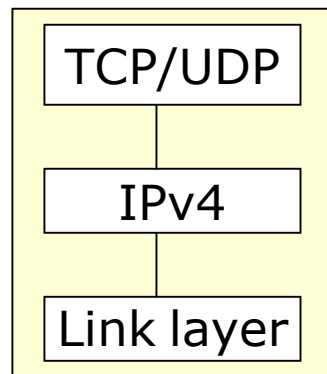
- IPV6_UNICAST_HOPS
- Multicast:
 - IPV6_MULTICAST_IF
 - IPV6_MULTICAST_HOPS
 - IPV6_MULTICAST_LOOP
 - IPV6_JOIN_GROUP
 - IPV6_LEAVE_GROUP
- IPV6_V6ONLY for AF_INET6

Agenda

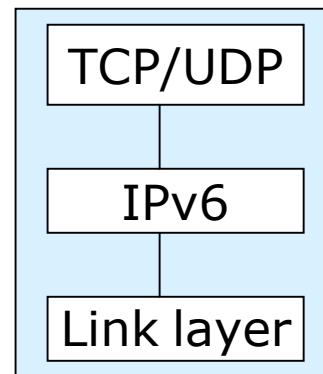
- Not change applications, nor kernel.
- Porting applications
 - Identifying IP version dependencies
 - Porting source code
- IPv4/IPv6 Interoperability
- Recommendations

IPv4/IPv6 Interoperability

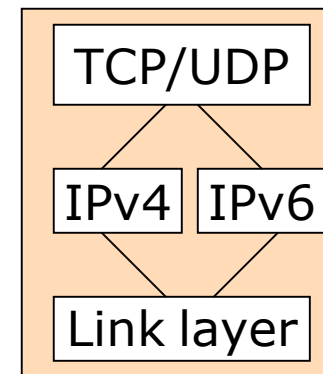
- Dual stack will be the mechanism used during transition period.
- Not all combinations between IPv4/IPv6 only-nodes and dual stack nodes are allowed to interact.



IPv4-only node



IPv6-only node

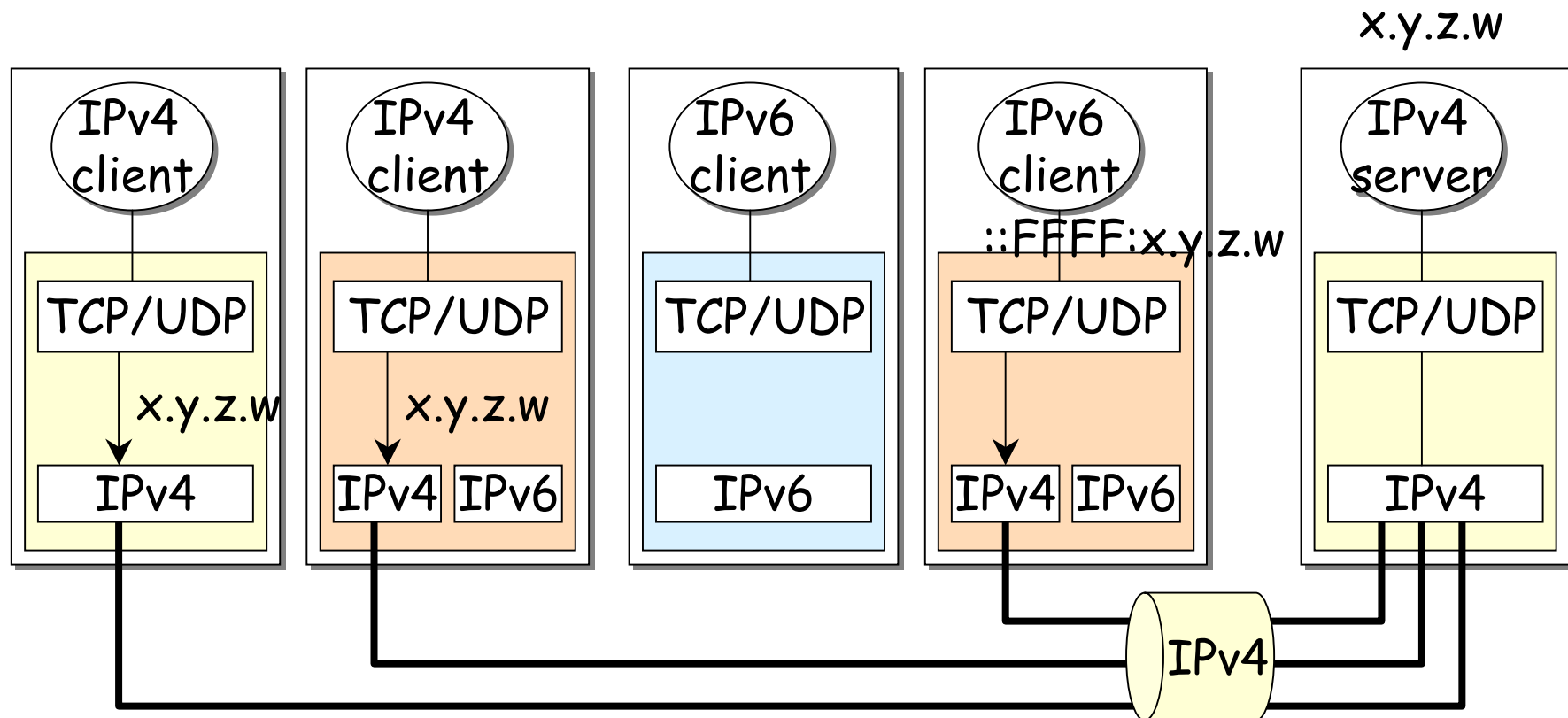


Dual stack node

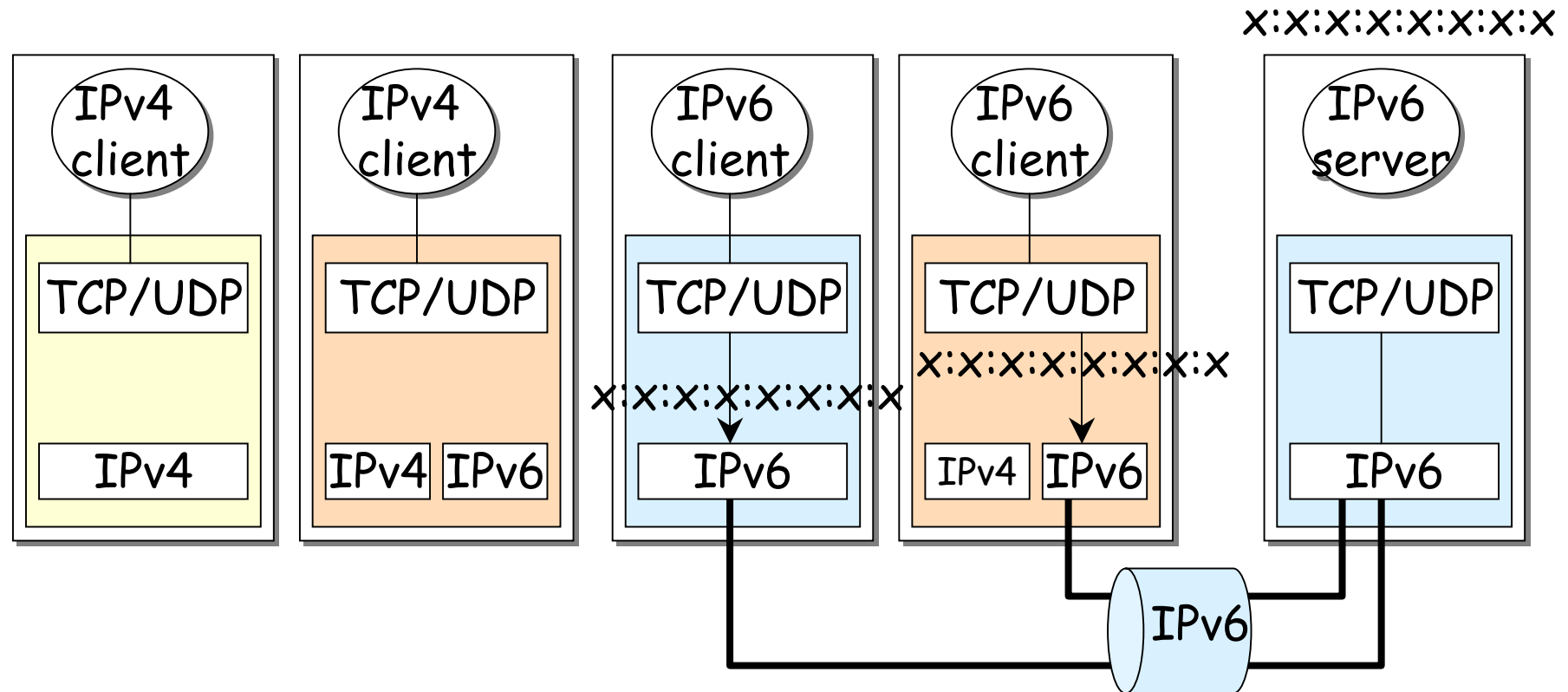
Interoperability cases

1. IPv4 server at IPv4-only node.
2. IPv6 server at IPv6-only node.
3. IPv4 server at dual stack node.
4. IPv6 server at dual stack node.
5. IPv6-only server at dual stack node.

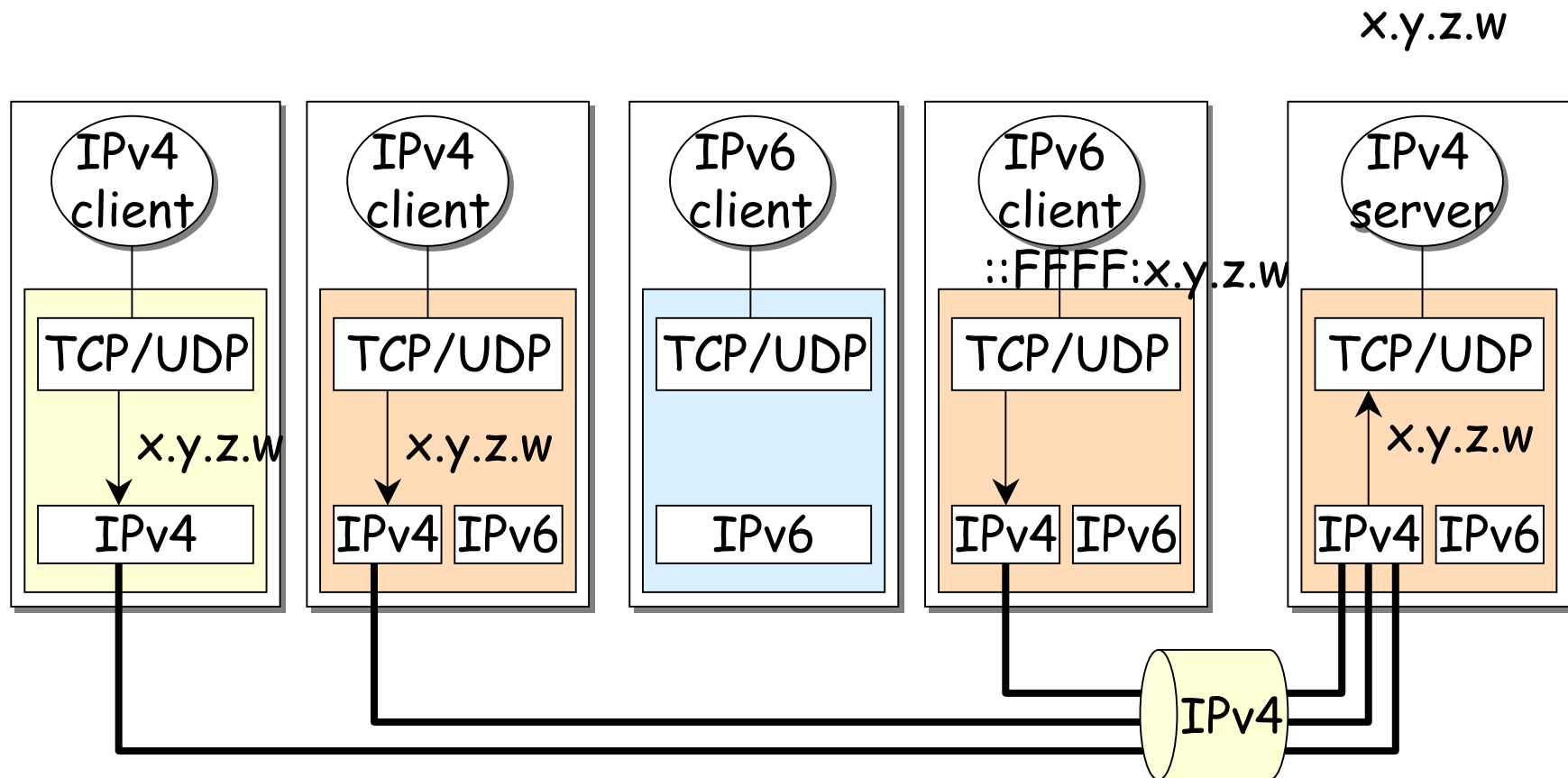
1. IPv4 server at IPv4-only node



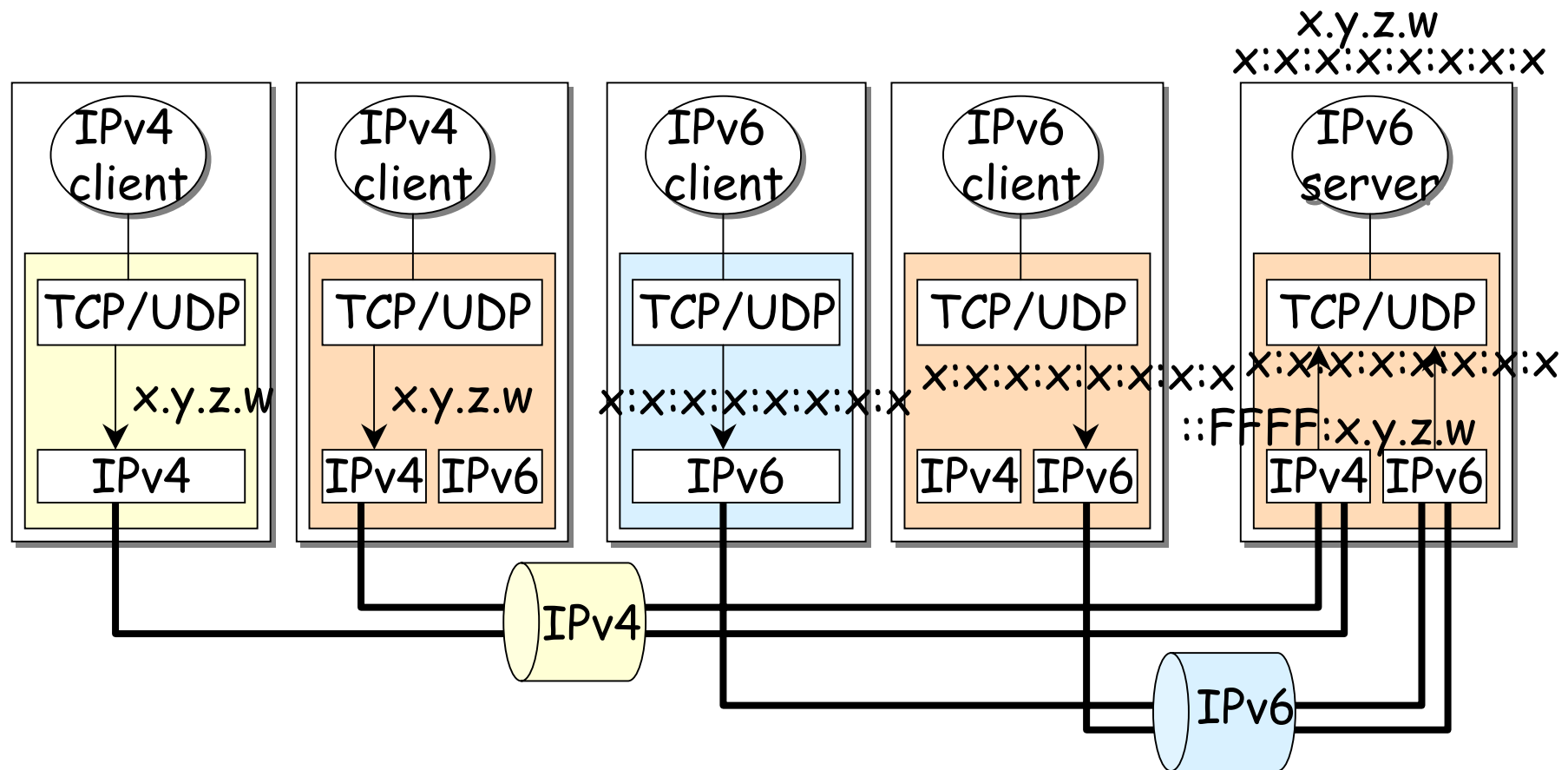
2. IPv6 server at IPv6-only node



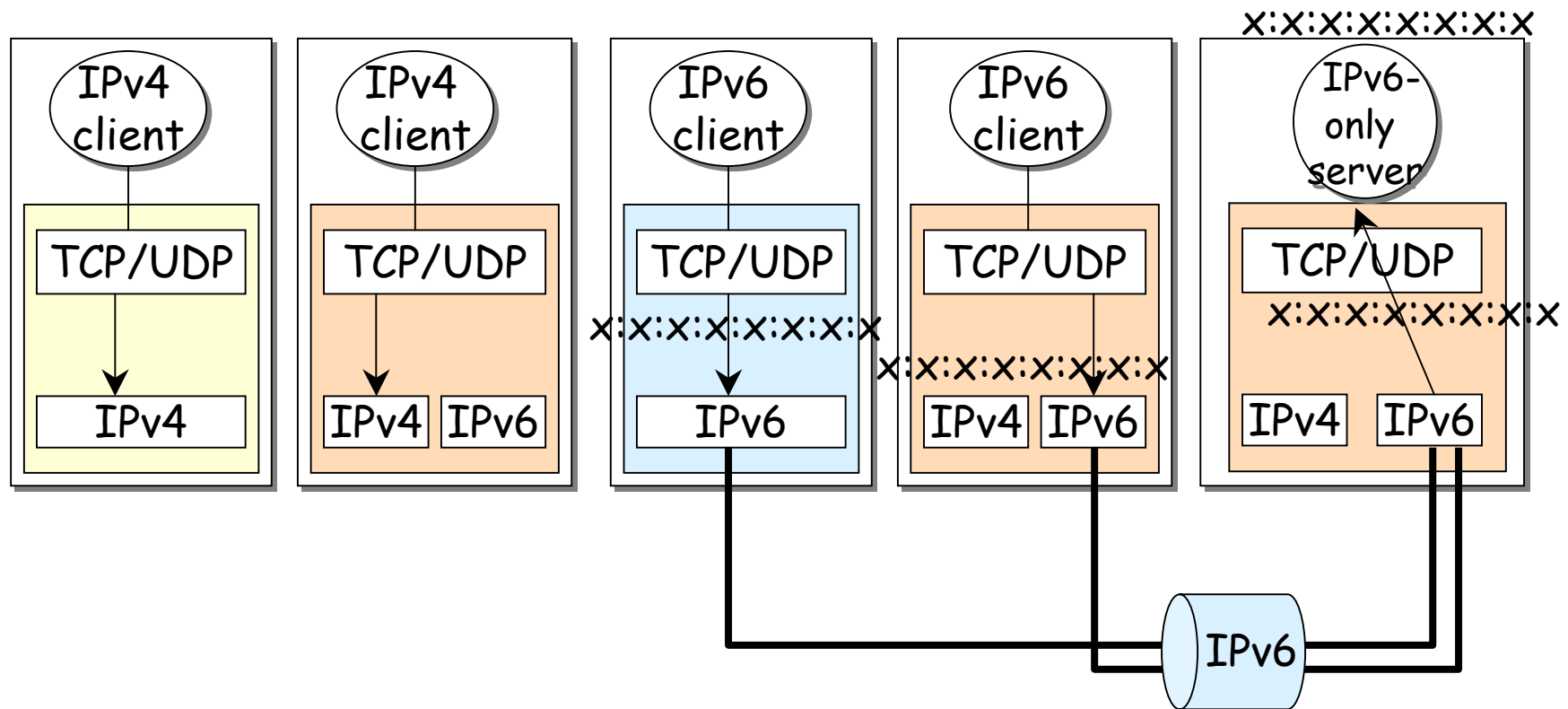
3. IPv4 server at dual stack node



4. IPv6 server at dual stack node



5. IPv6-only servers at dual stack



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

- Not change applications, nor kernel.
- Porting applications
 - Identifying IP version dependencies
 - Porting source code
- IPv4/IPv6 Interoperability
- Recommendations

Recommendations

- Separate application transport module.
- Write protocol independent code (dual app):
 - struct sockaddr_storage
 - getaddrinfo/getnameinfo
- Use FQDN instead IP address.
- If IP address needed, refresh periodically.