tianocore

UEFI & EDK II Base Training

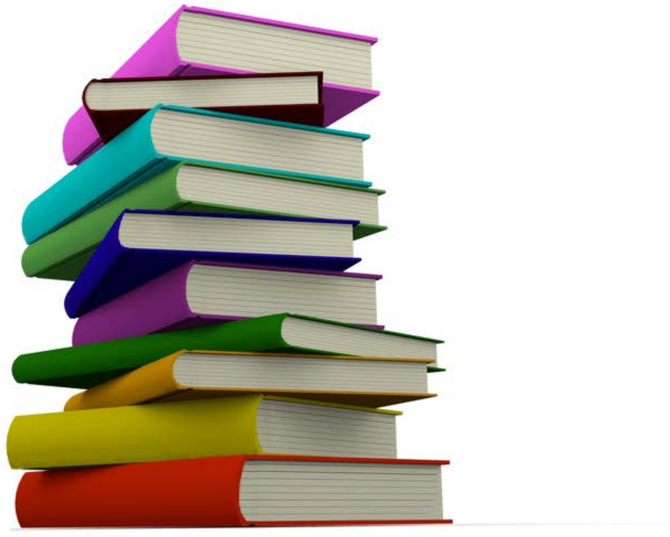
UEFI Network in EDK II

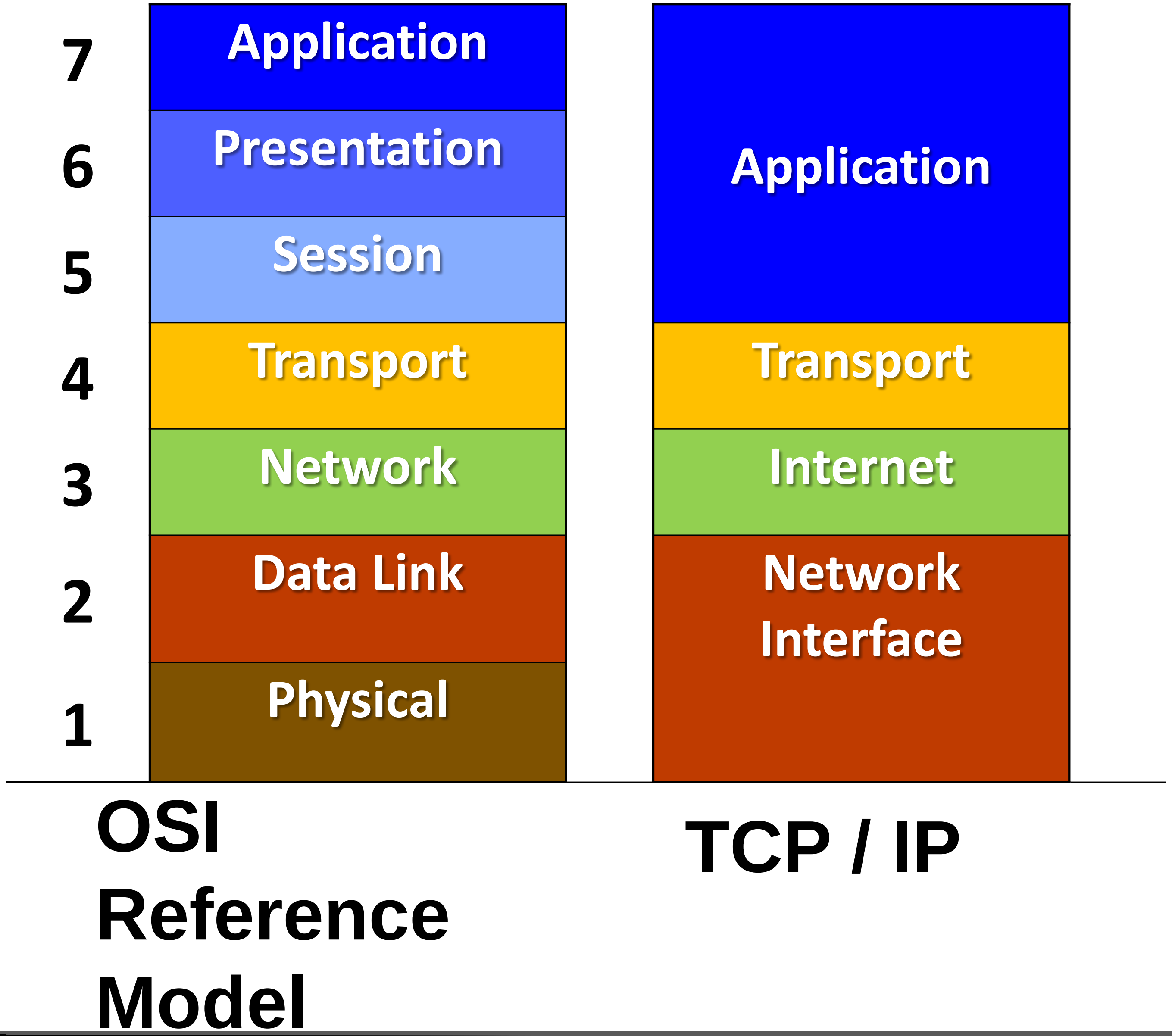
tianocore.org

Lesson Objective

- UEFI Network Stack Layers
- EDK II Network Features Overview
- What UEFI Protocols Make Network Work in EDK II
- UEFI HTTP(s) Boot Overview

Industry Standard – Network Stack Layers





1. Open Systems Interconnection(OSI) Model is a reference model.
2. TCP/IP Model roughly covers six layers of the OSI Reference Model with the TCP/IP Application layer as a compound of the Application, Presentation and Session layers in OSI Model.
3. TCP/IP Model doesn't cover the Physical/Hardware layer.
4. The TCP/IP Model is familiar, UEFI network stack also follows this model. We will have a big picture on what we have now in each of the TCP/IP Model layers.

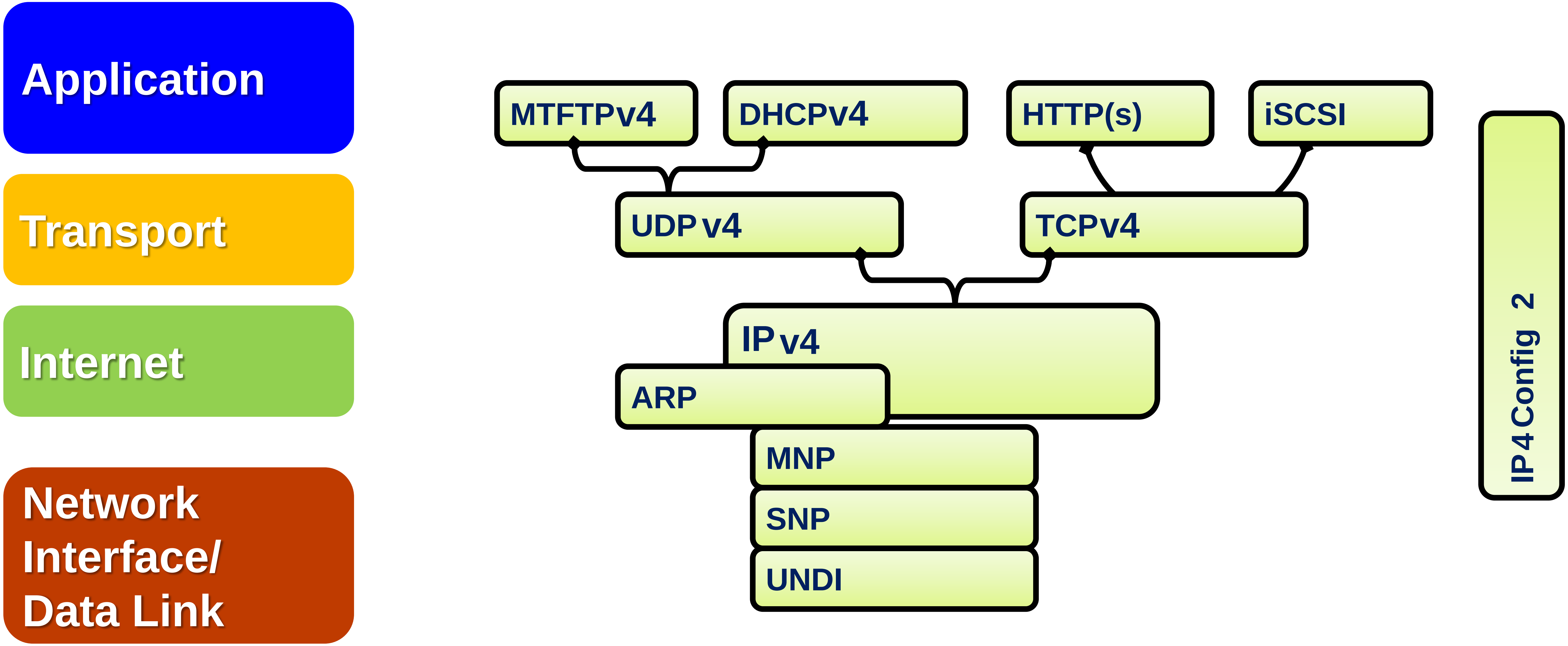
TCP / IP Network Stack for UEFI

Layer	TCP / IP	UEFI Drivers & Applications
7, 6, 5	Application	MTFTP, PXE, DHCP, ISCSI, HTTP(s), TLS, DNS
4	Transport	TCP, UDP
3	Internet	IP (4|6), ARP, ICMP, IGMP(4), ND(6), MLD(6)
2, 1	Network Interface	MNP, SNP, UNDI, DPC

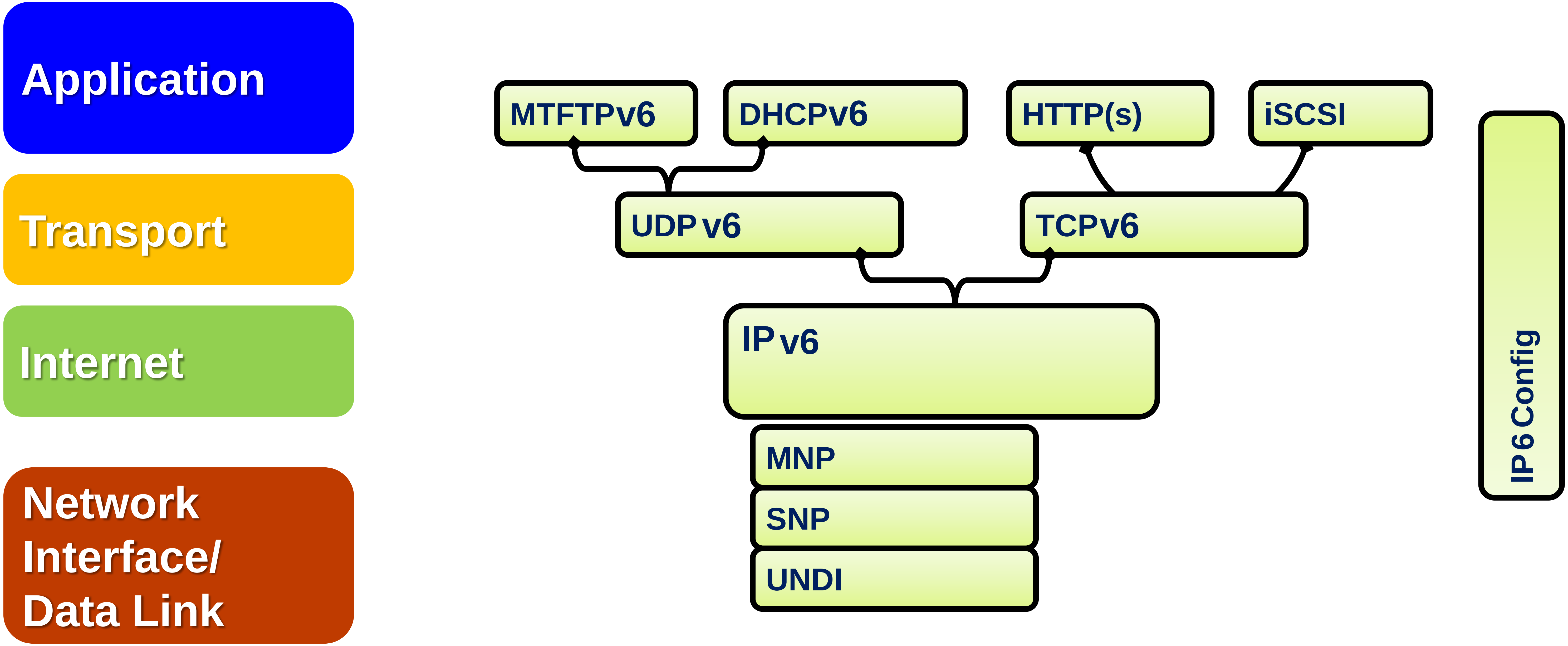
Network Acronyms

Layer	Acronym	Meaning
Application	MTFTP	- Multicast Trivial File Transfer Protocol
	PXE	– PreBoot eXecution Environment
	DHCP	- Dynamic Host Configuration Protocol
	iSCSI	- Internet Small Computer System Interface
	HTTP(s)	- HyperText Transfer Protocol w/ (s)-TLS
	TLS	- Transport Layer Security
Transport	DNS	- Domain Name Server
	TCP	- Transmission Control Protocol
Internet	UDP	- User Datagram Protocol
	IP	- Internet Protocol
Network Interface/ Data Link	ARP	- Address Resolution Protocol
	MNP	– Managed Network Protocol
	SNP	– Simple Network Protocol
	UNDI	- Universal Network Device Interface
	DPC	- Deferred Procedure Call

Network Stack IPv4



Network Stack IPv4 and/or IPv6



Network Stack IPv4 and/or IPv6

Network Interface/ Data Link

MNP

SNP

UNDI

Network Interface Card Device Driver – Provide interfaces to operate the NIC.

Network Stack IPv4 and/or IPv6

Network Interface/ Data Link

MNP

SNP

UNDI

Abstract interfaces of UNDI to packet level I/O & management interfaces.

Network Stack IPv4 and/or IPv6

Provide concurrent access to frame-level functions. Multiplexer and de-multiplexer.

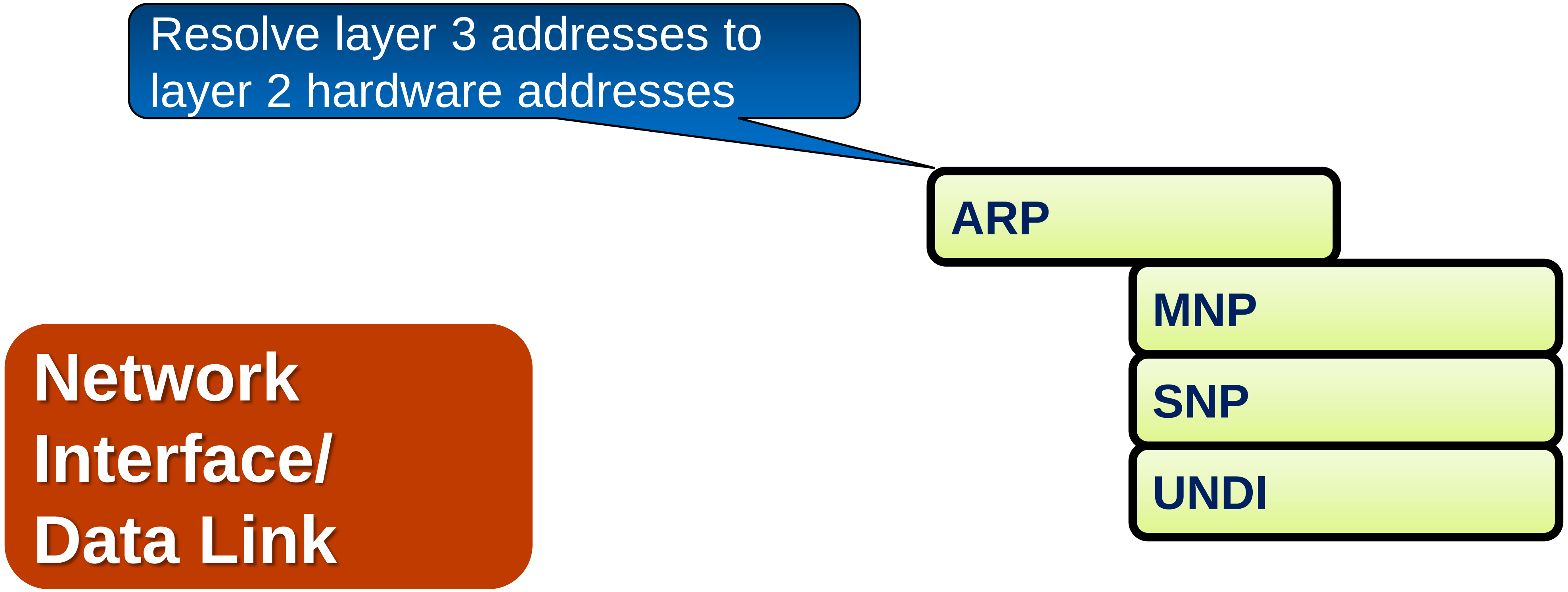
MNP

SNP

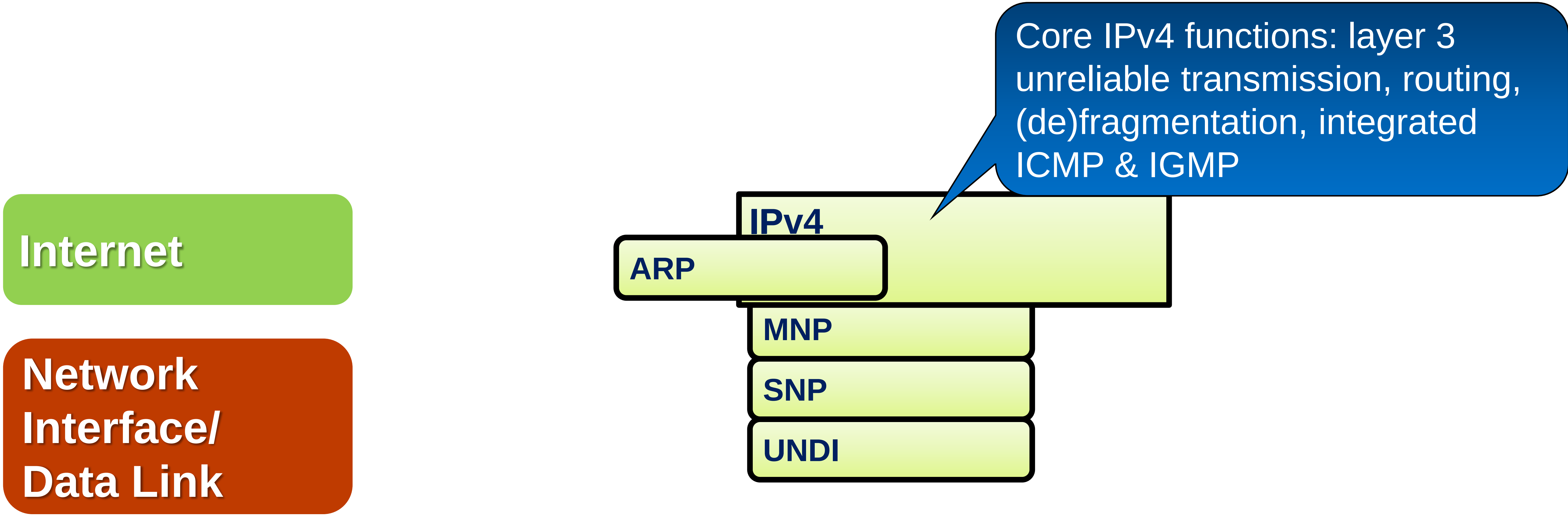
UNDI

Network Interface/ Data Link

Network Stack IPv4 and/or IPv6



Network Stack IPv4



Network Stack IPv4 and/or IPv6

Core IPv6 functions: layer 3 unreliable transmission, routing, (de)fragmentation, integrated ICMP & MLD & ND

Internet

IPv6

Network Interface/ Data Link

MNP

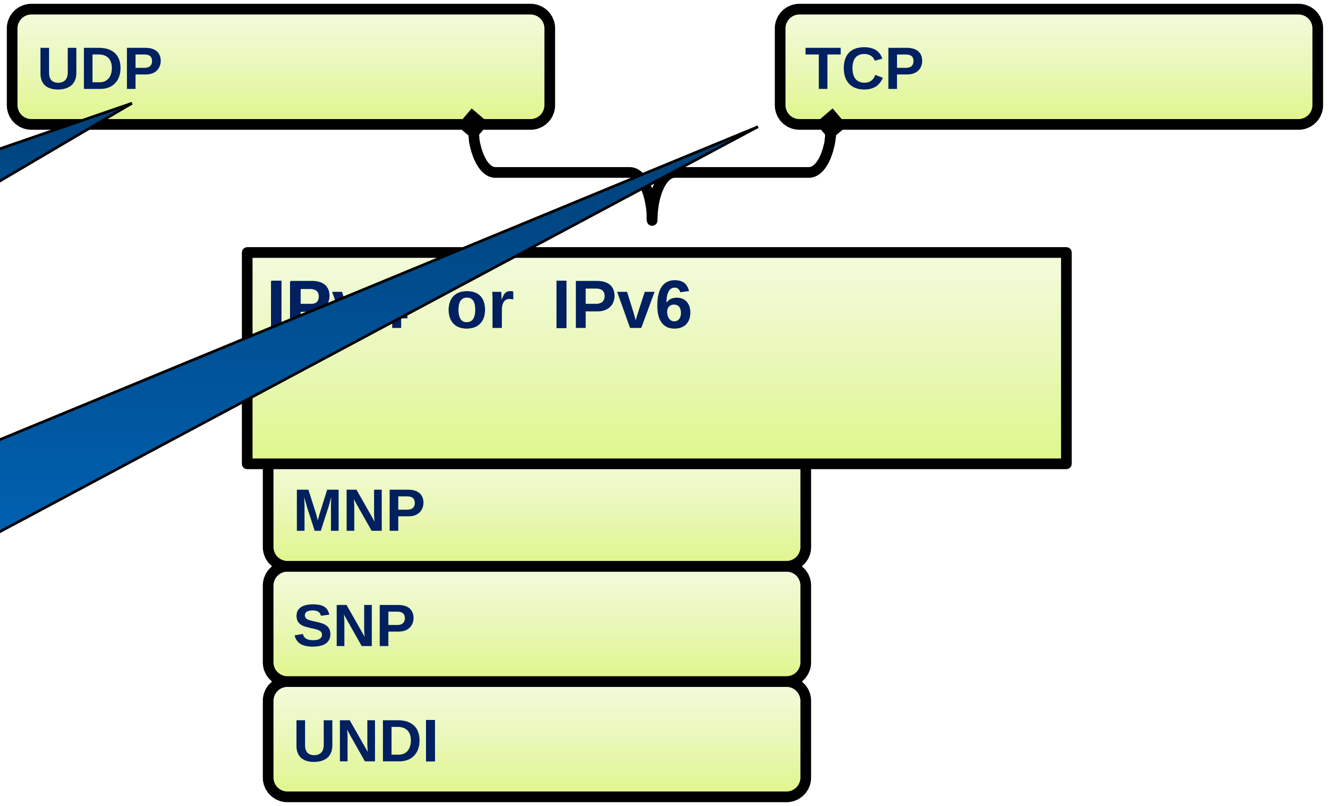
SNP

UNDI

Network Stack IPv4 and/or IPv6

Transport

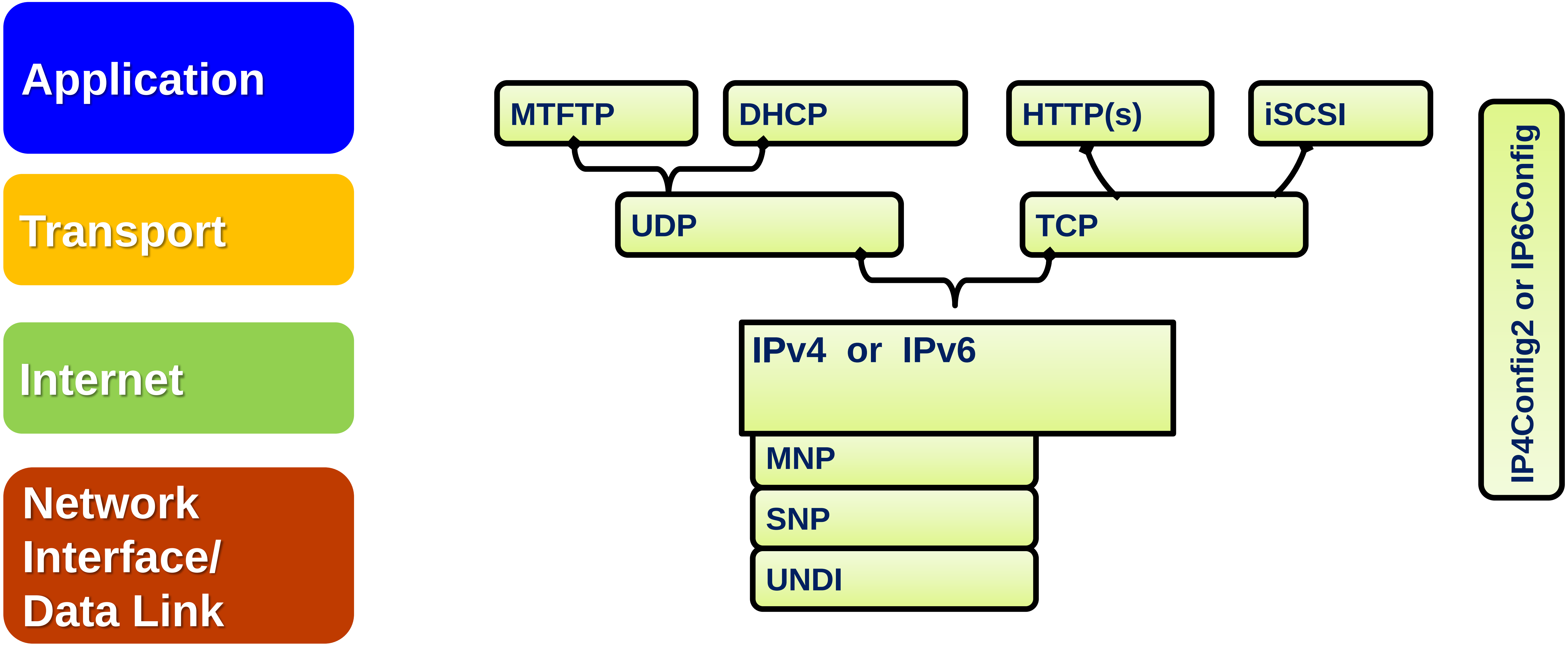
Network



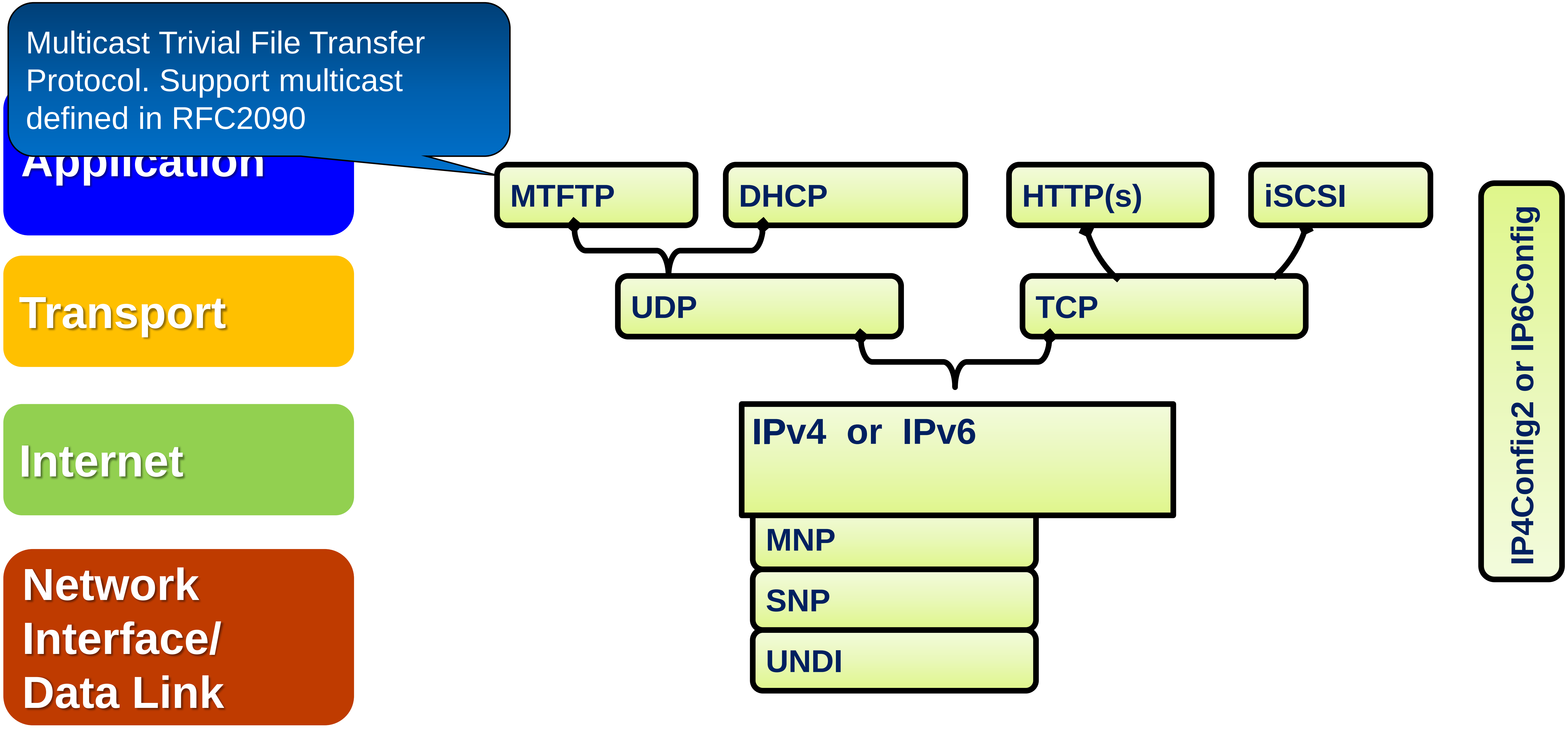
Unreliable transport layer service. Datagram – data is transmitted in blocks

Reliable transport layer service, more complex than UDP. Data is transmitted as a continuous stream.

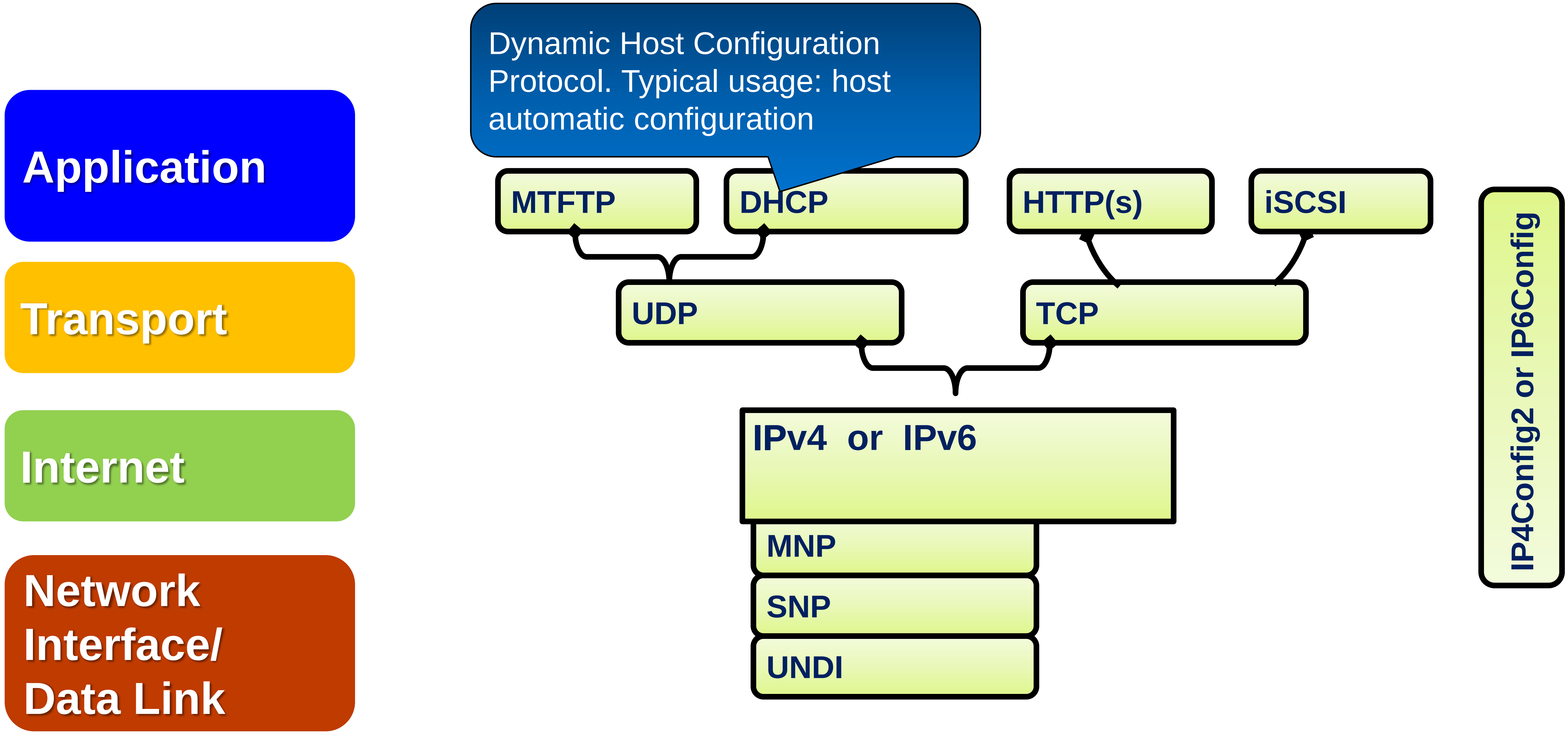
Network Stack IPv4 and/or IPv6



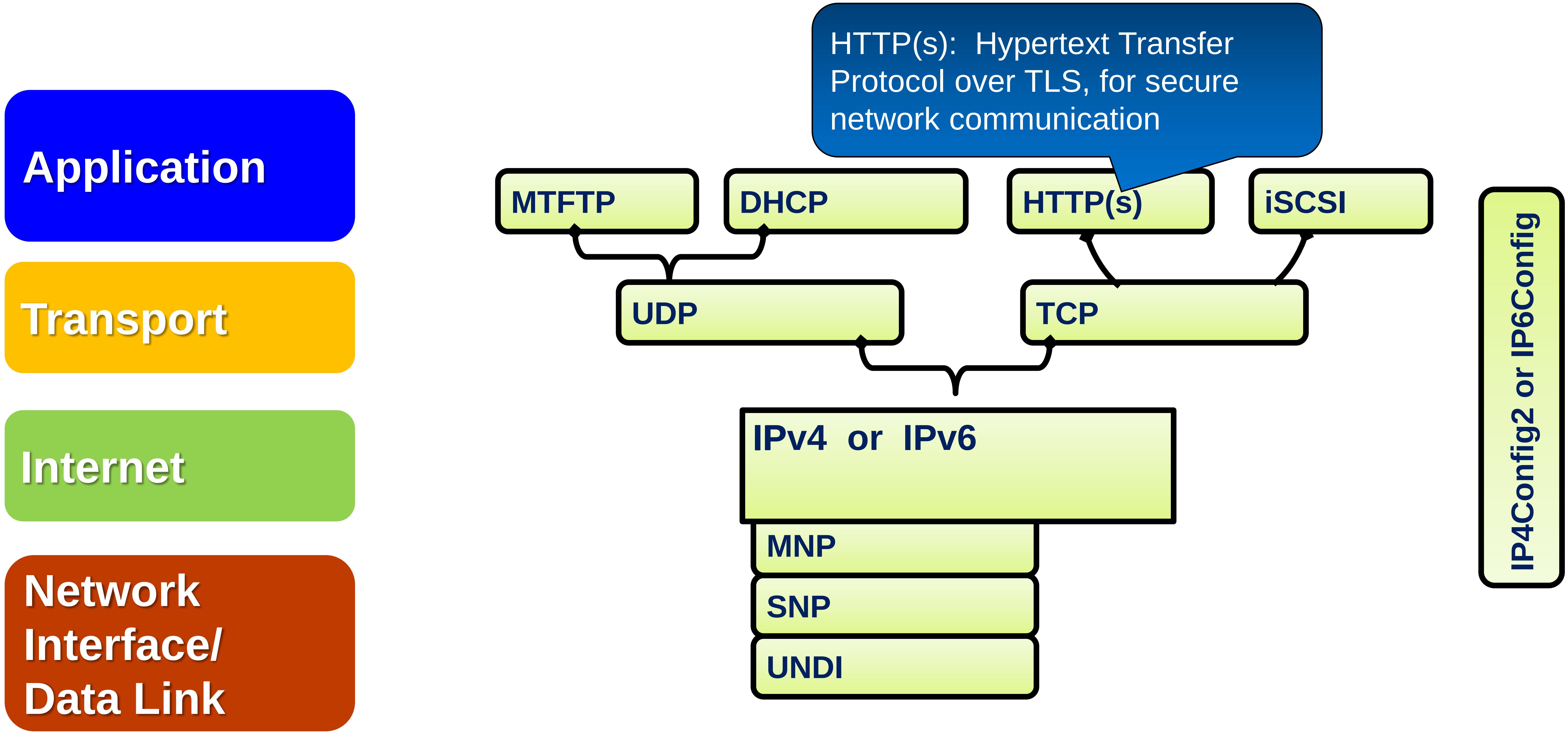
Network Stack IPv4 and/or IPv6



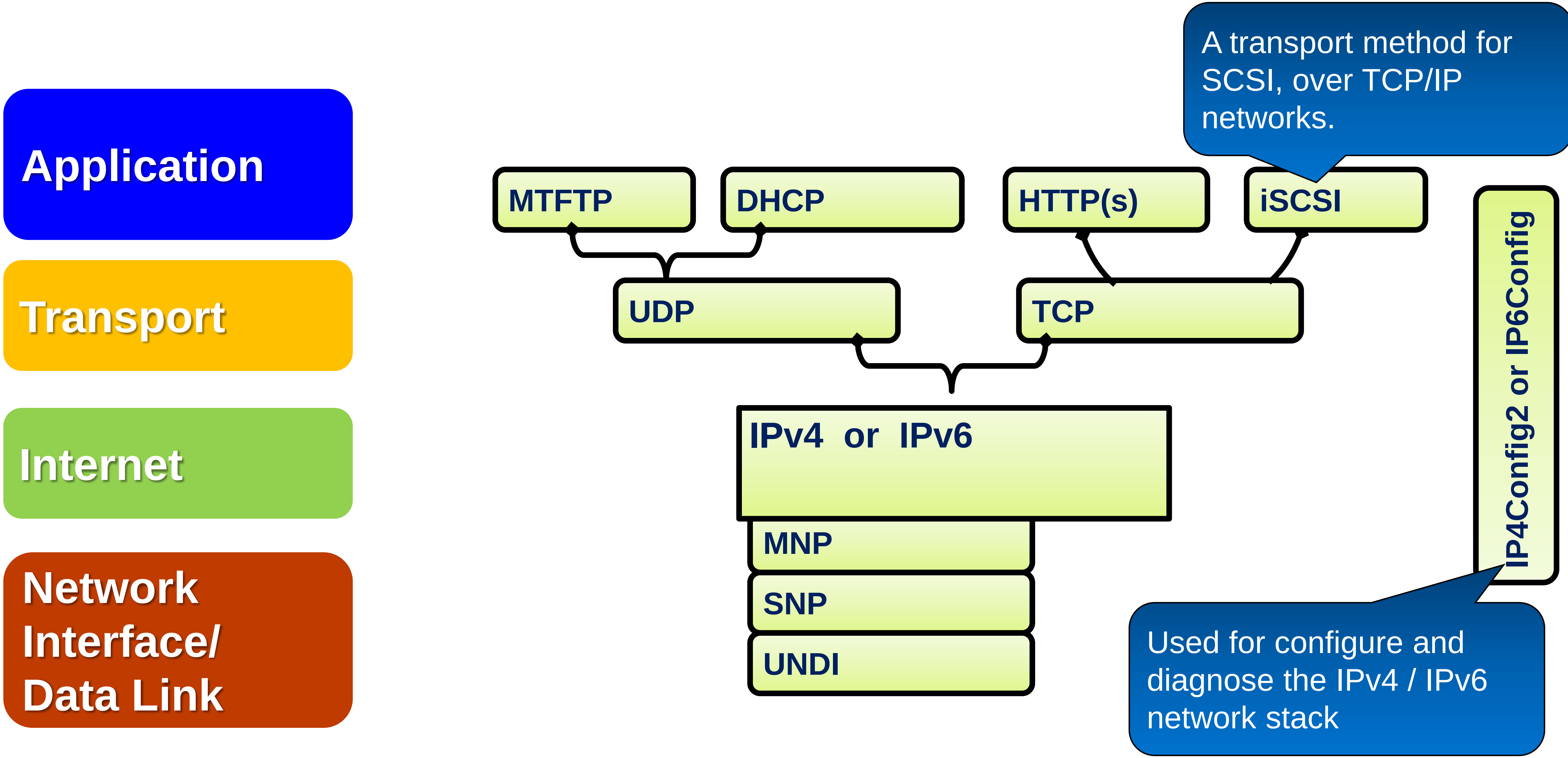
Network Stack IPv4 and/or IPv6



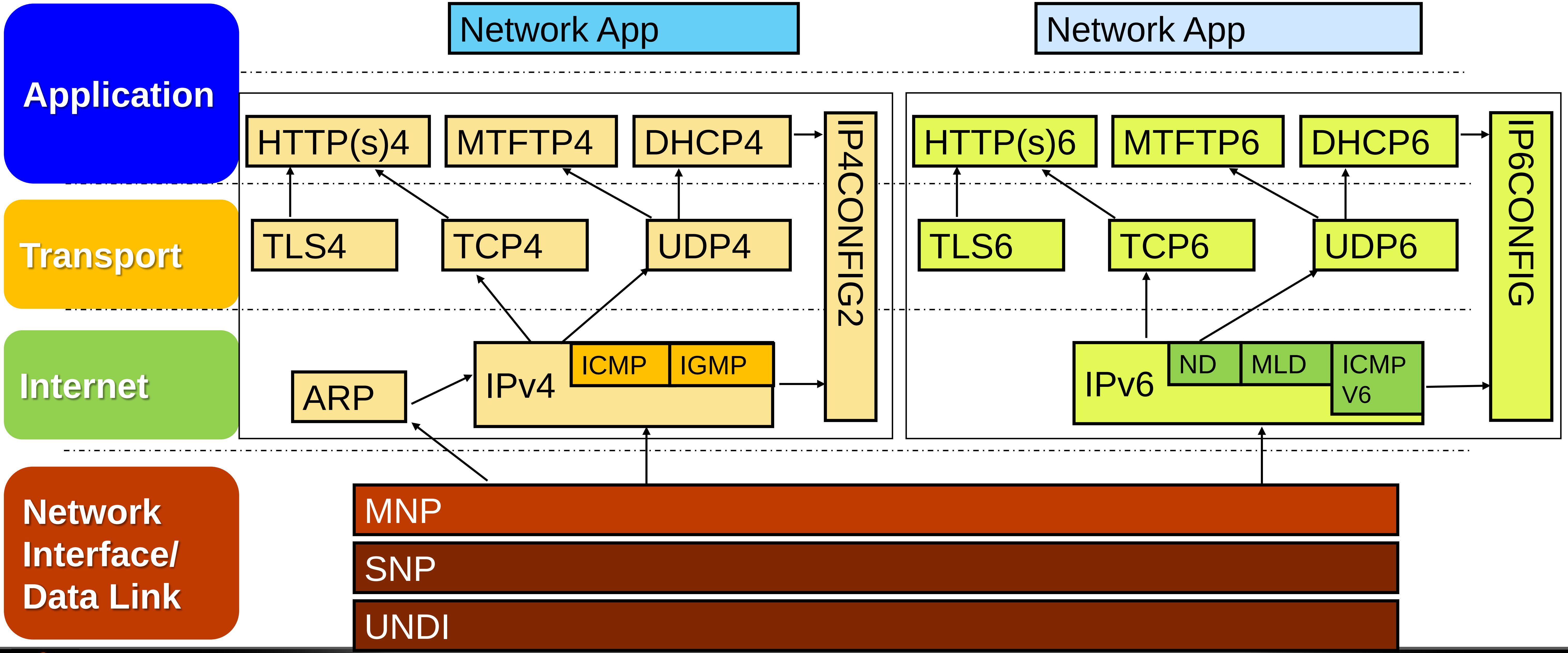
Network Stack IPv4 and/or IPv6



Network Stack IPv4 and/or IPv6



UEFI Network Stack: IPv4 vs IPv6



UEFI Network Stack w/ Protocols

PXEBaseCode
HTTP(s) Boot
iSCSI
HTTP(s) Boot
IPv6 PXE

Ping
IfConfig
Ping6
IfConfig6

DHCP4
DNS4
TLS4
MTFTP4
MTFTP6
TLS6
DNS6
DHCP6

UEFI IP4 Stack
-EFI_IP4_CONFIG2_PROTOCOL
-EFI_TCP4_PROTOCOL
-EFI_UDP4_PROTOCOL
-EFI_ARP_PROTOCOL
-EFI_IP4_PROTOCOL

EFI_IPSEC_CONFIG_PROTO

BaseCryptLib

UEFI IP6 Stack
-EFI_TCP6_PROTOCOL
-EFI_UDP6_PROTOCOL
-EFI_IP6_CONFIG_PROTOCOL
-EFI_IP6_PROTOCOL

MNP (IPv4 & IPv6 ready)
SNP
UNDI

UEFI 2.5 added HTTP(s) Boot 2015

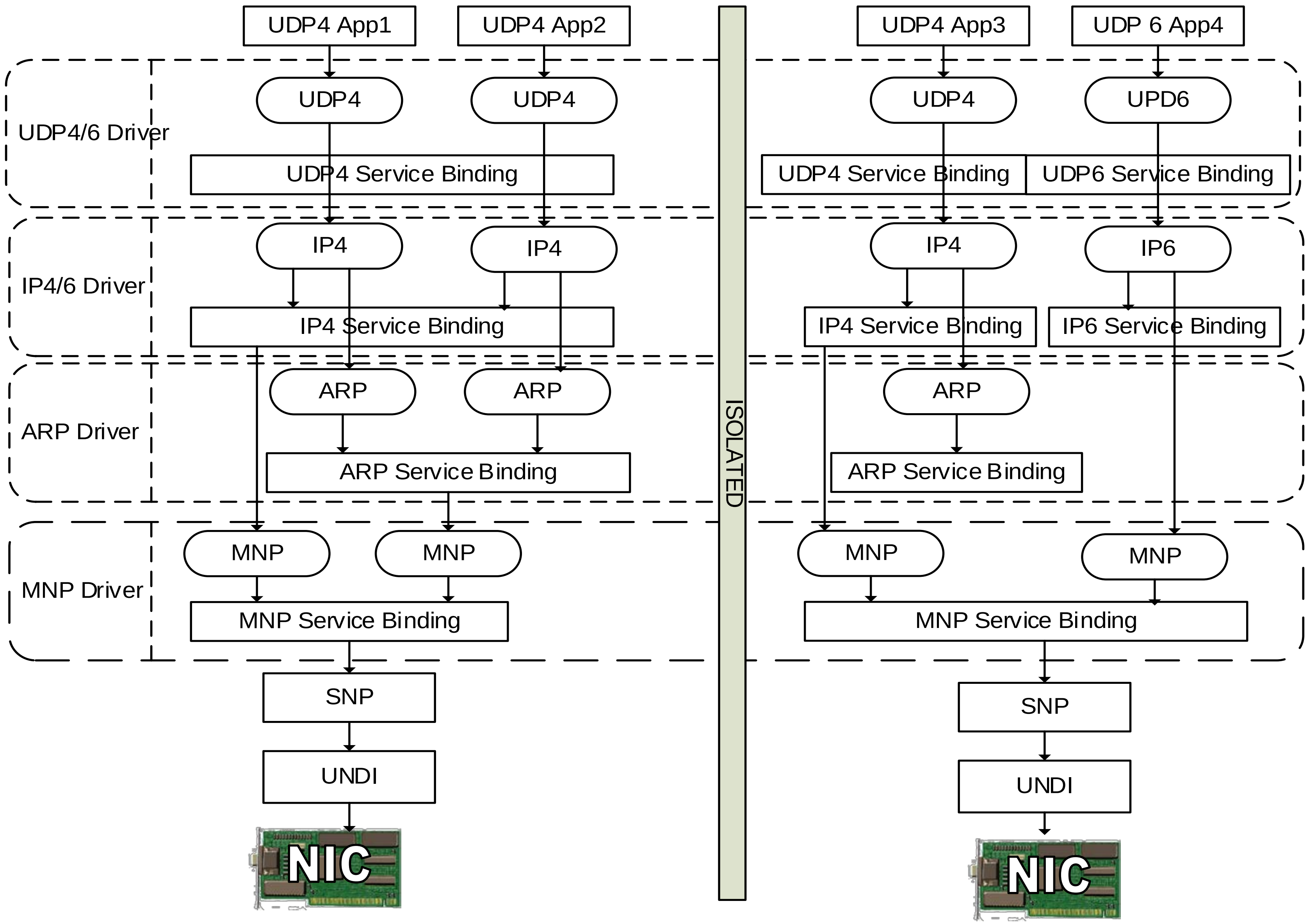
A Live Picture

Application

Transport

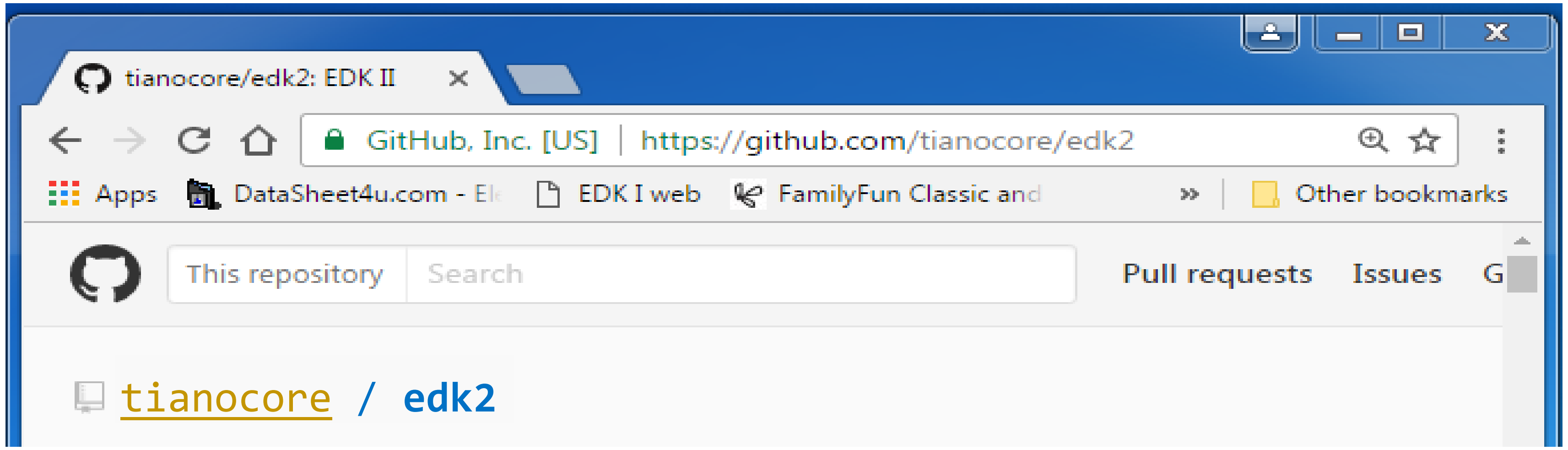
Internet

Network Interface/ Data Link



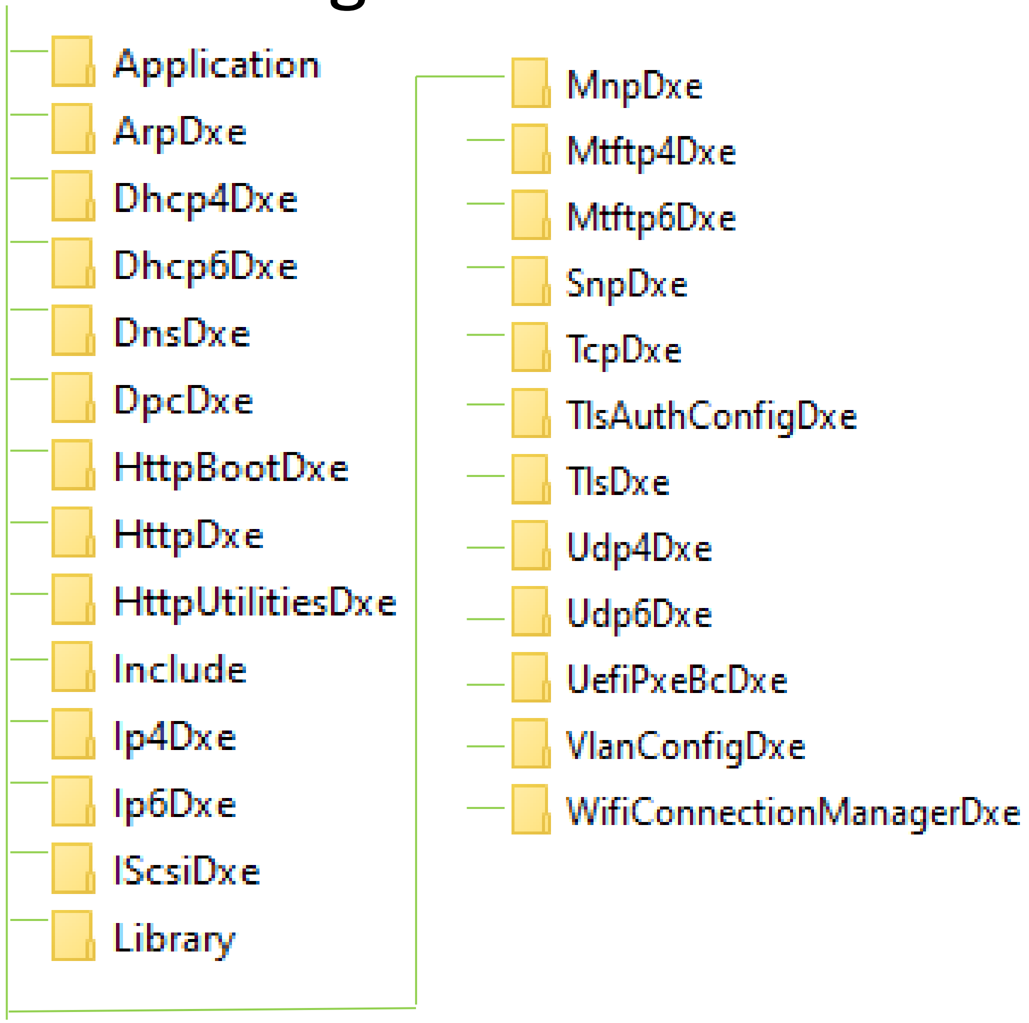
EDK II Network Features Overview

Where is the EDK II Network Stack located?



github.com/tianocore/edk2

NetworkPkg



Network Related Libraries

MdeModulePkg

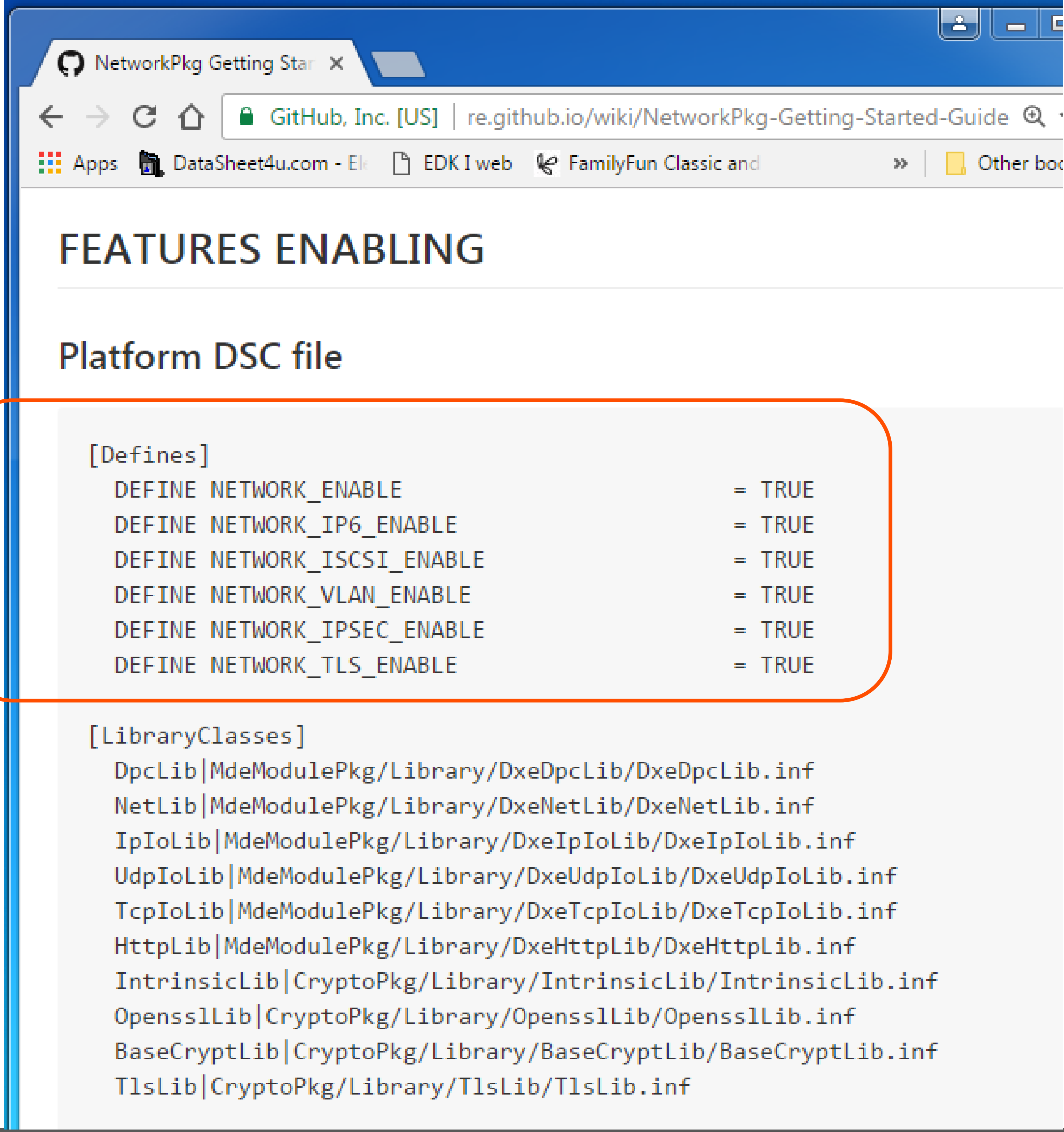
Library

How to Enable the EDK II Network Stack

Update the Platform DSC and FDF files

- Link: https://github.com/tianocore/tianocore.github.io/wiki/NetworkPkg-Getting-Started-Guide#features-enabling

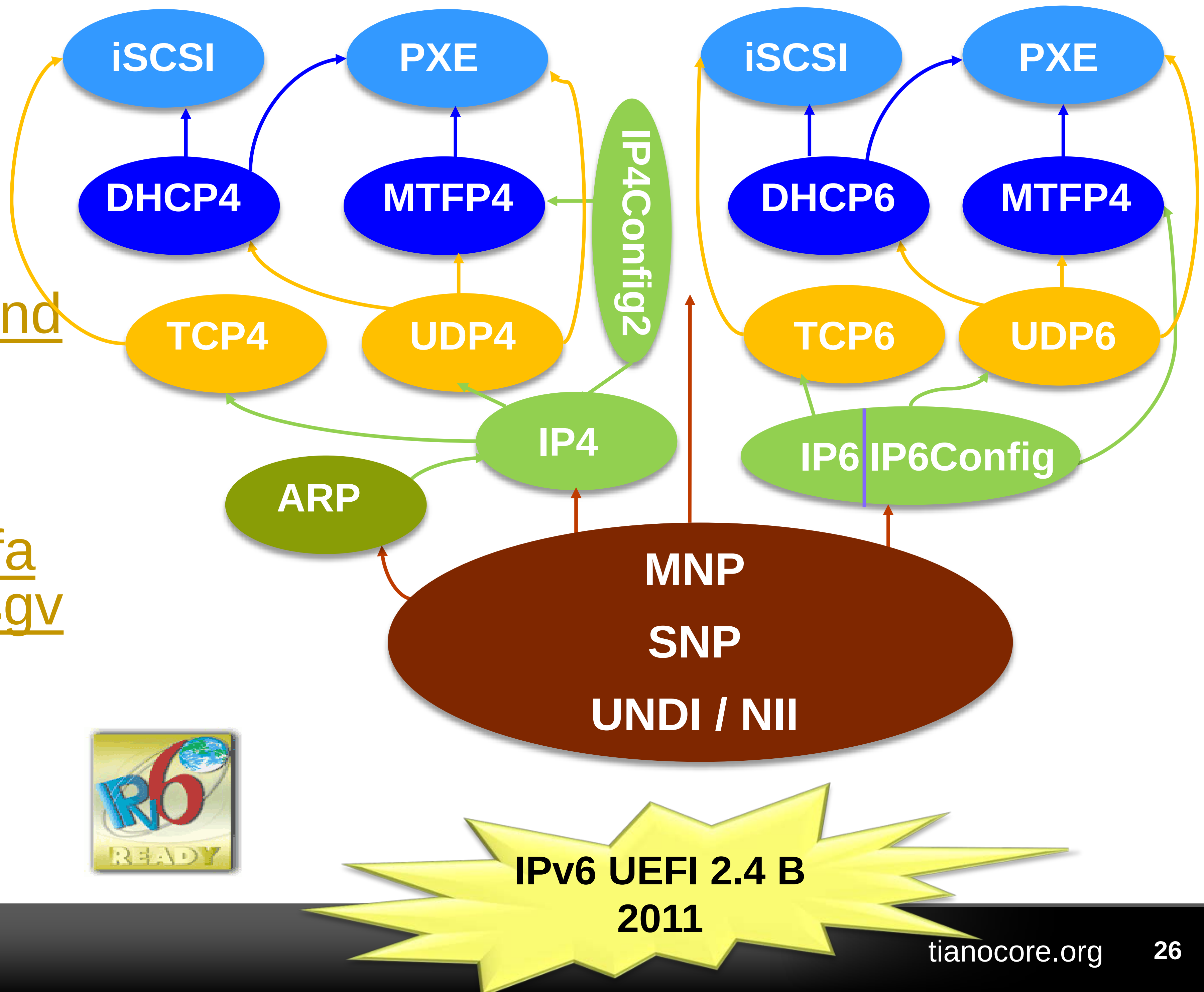
```
DEFINE NETWORK_ENABLE  = TRUE
. . .
```



IP6 Networking - Vendors

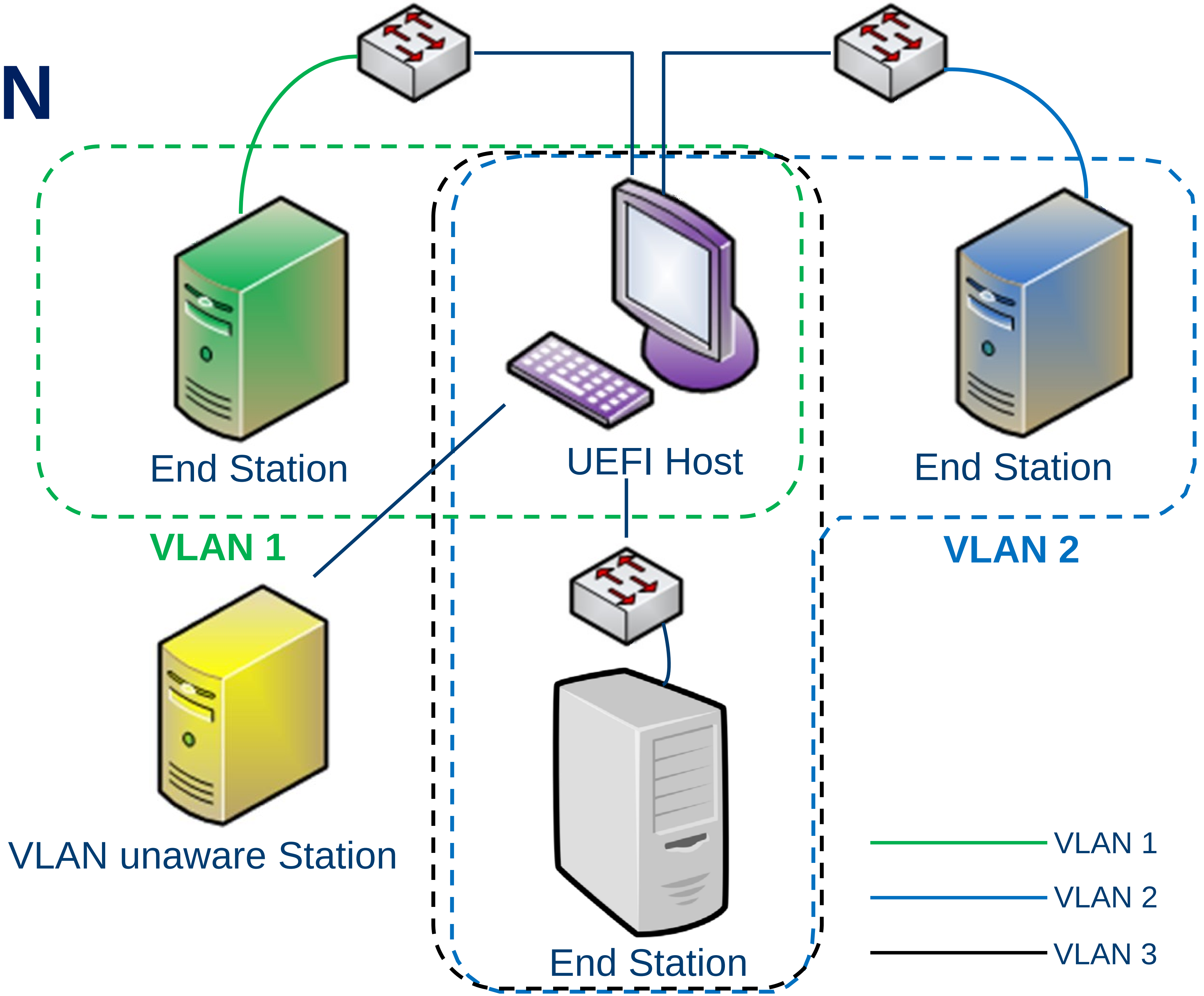
IPv6 protocols compliance

– IPv6 ready logo approved http://www.ipv6ready.org/db/index.php/public/
– Requirements for IPv6 transition https://www.nist.gov/sites/default/files/documents/itl/antd/usgv6-v1.pdf
– Vendor Testing: https://www-x.antd.nist.gov/usgv6/faq-c.html#vendors



Virtual LAN - VLAN

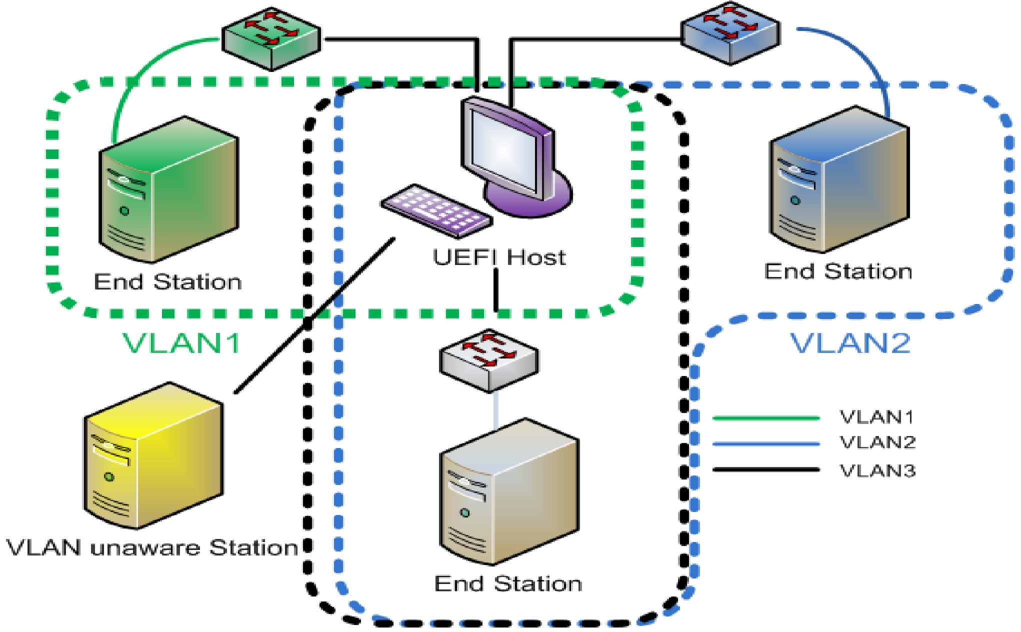
- Logical groups of Stations at the data link layer (Tagging)
  - VLAN's allow a network manager to logically segment a LAN into different broadcast domains [Link]
- Why VLAN?
  - Performance
  - Security
  - Formation of Virtual Workgroups
  - Simplified Administration
  - Cost
- VLAN Standard: IEEE 802.1Q

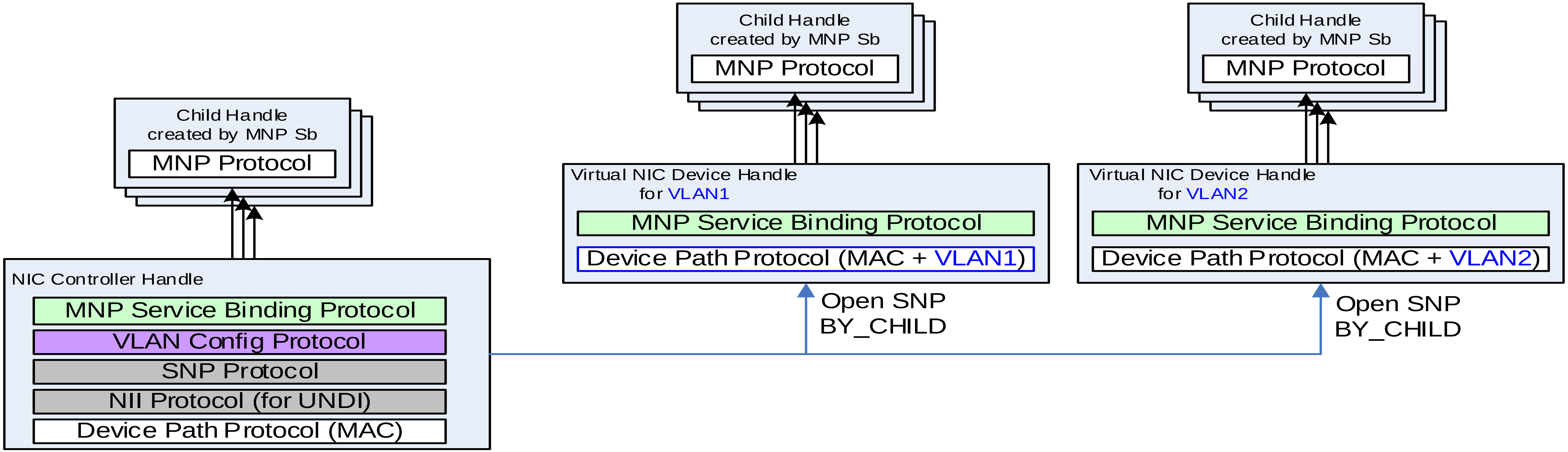


VLAN Support - EDK II

Technology includes

– Support Hybrid LAN topology
– Multiple VLAN for one station
– MNP and VLAN Configuration Protocol
– VLAN configuration by Shell Application `Vconfig`

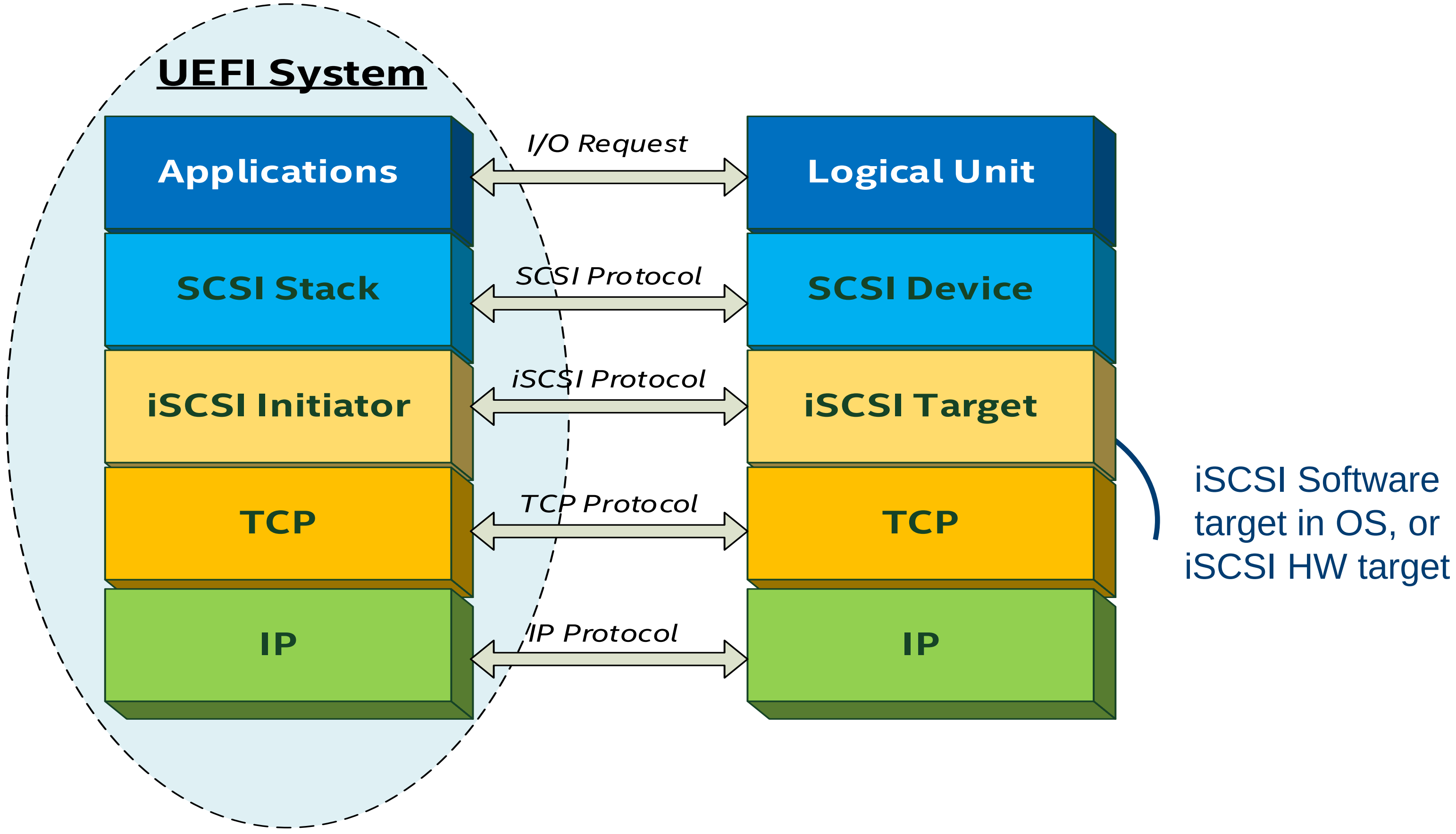


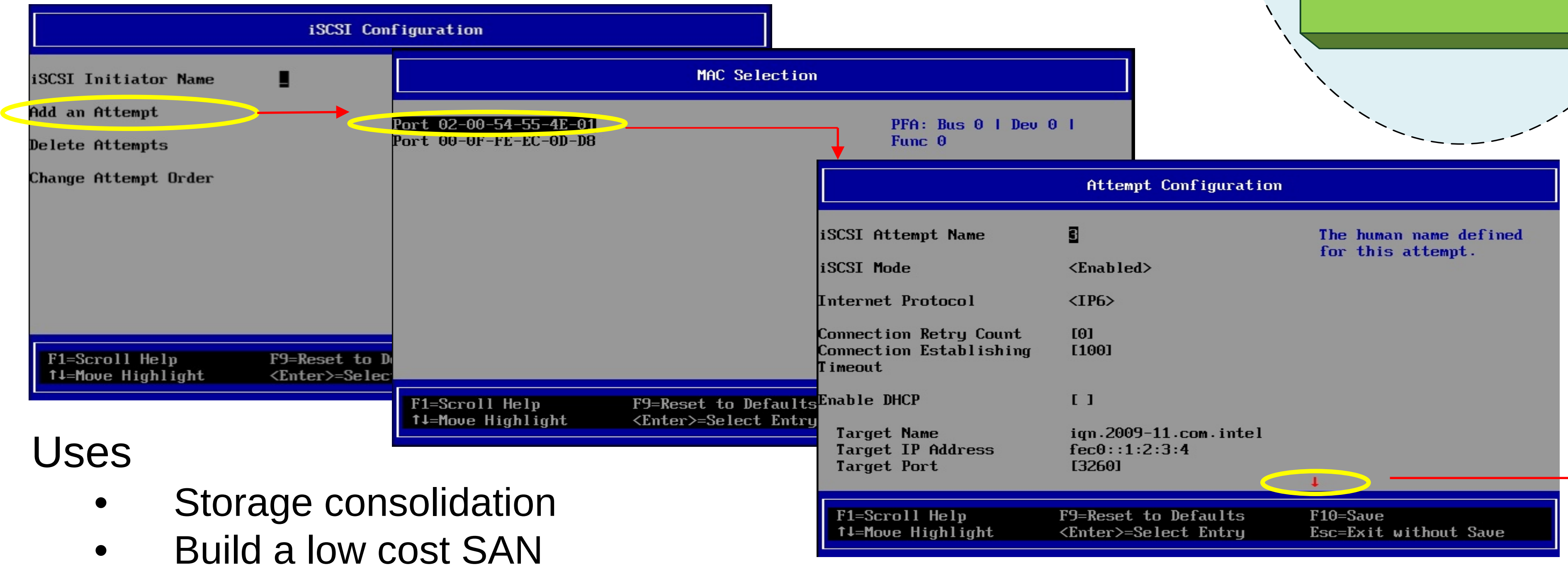


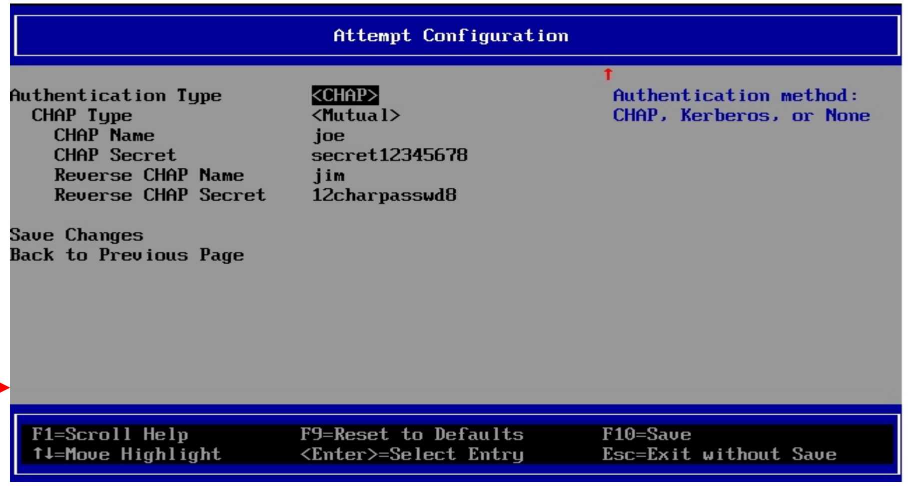
UEFI iSCSI Solutions

SAN/Data center boot over iSCSI

- Manual/DHCP based configuration allowed
- Cryptographic logon with CHAP
- Multi-path/fail-over capable
- User Interface using HII



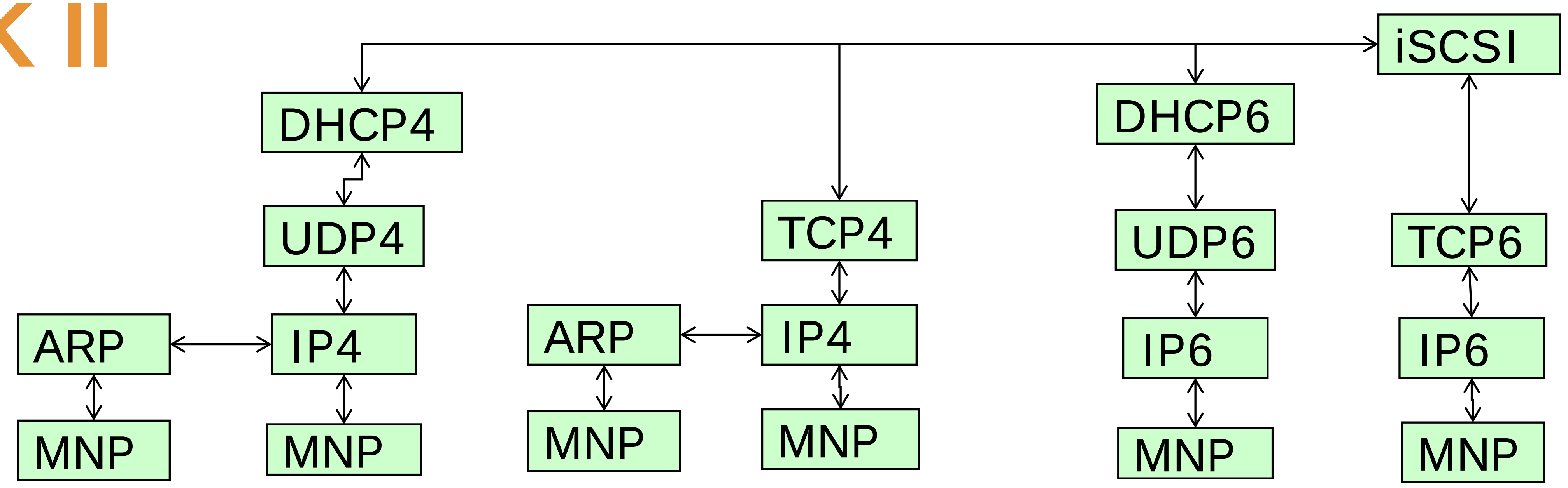


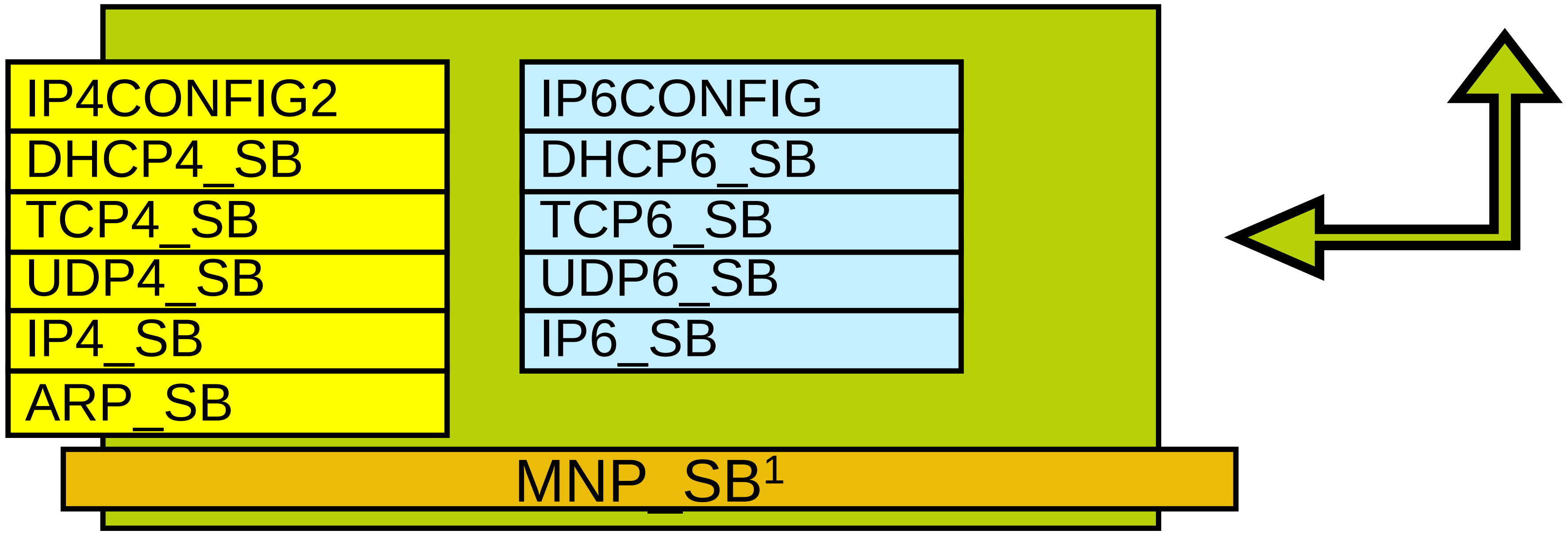


Uses

- Storage consolidation
- Build a low cost SAN
- Cluster Shared Volumes
- Diskless Booting

Dual-Stack Heritage – iSCSI usage model EDK II





[1] Service Binding Protocol

IPsec – Network Security

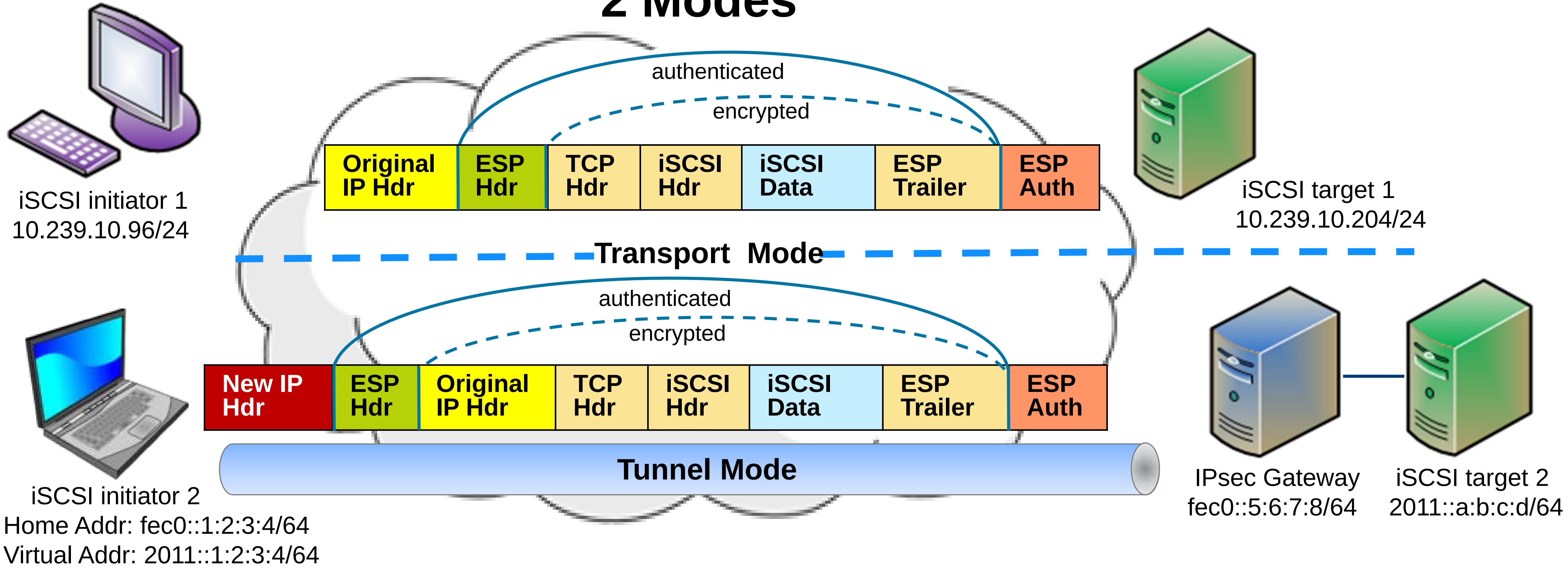
Secure Internet Protocol Communication

- Protects any application traffic across an IP network
- Mandatory for IPv6

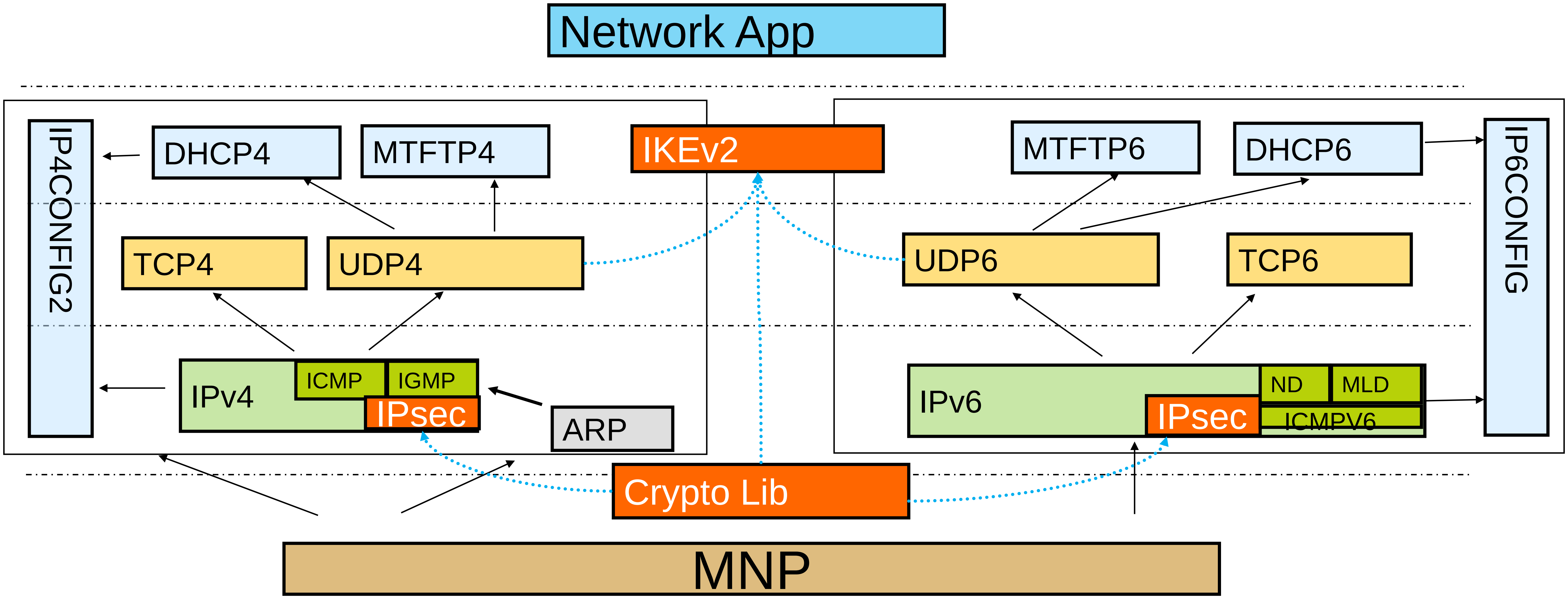
Features include

- AH, ESP, IKE version 2
- HMAC-SHA1, TripleDES-CBC, AES-CBC
- Modes of operation : Transport vs. Tunnel
- Pre shared Key/X.509 certificate authentication

2 Modes

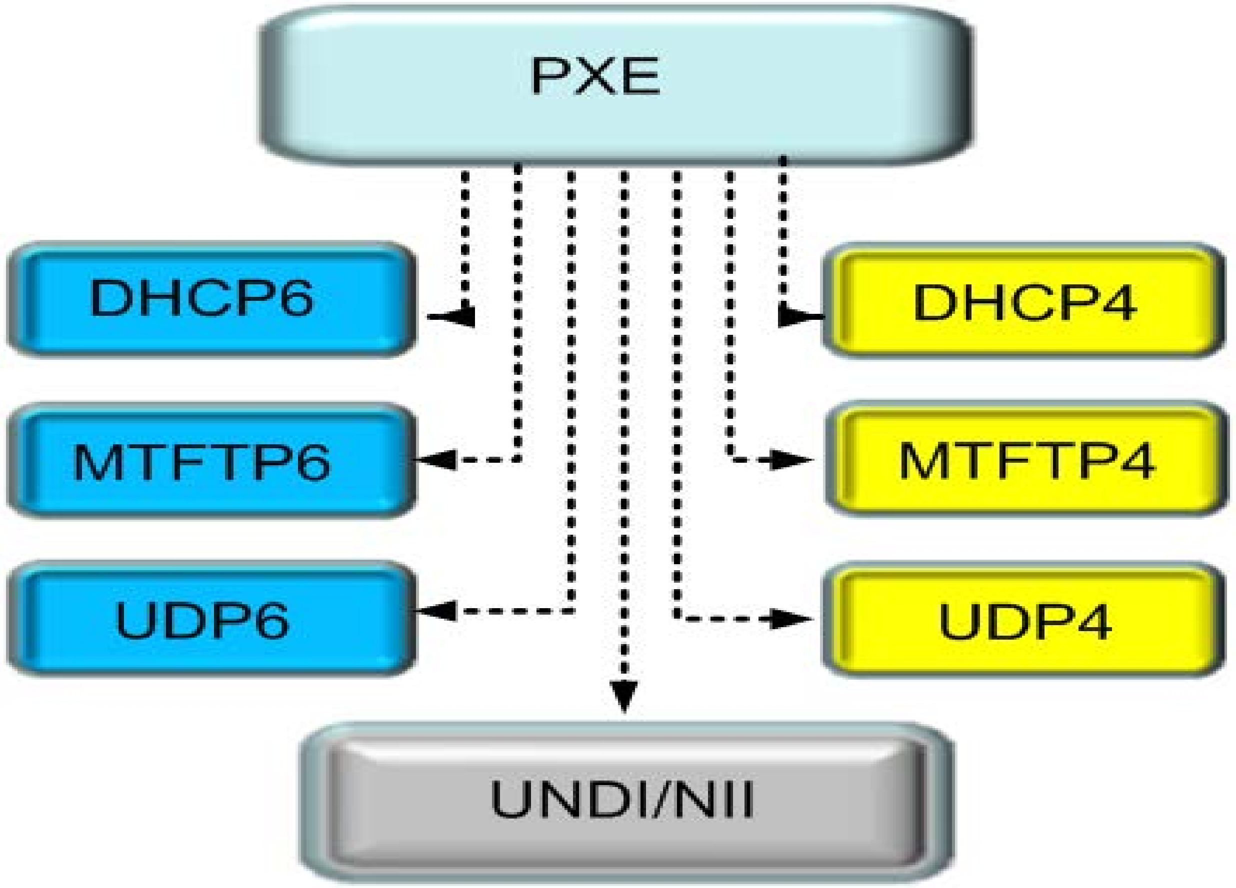


IPsec support: shared



UEFI PXE Solutions

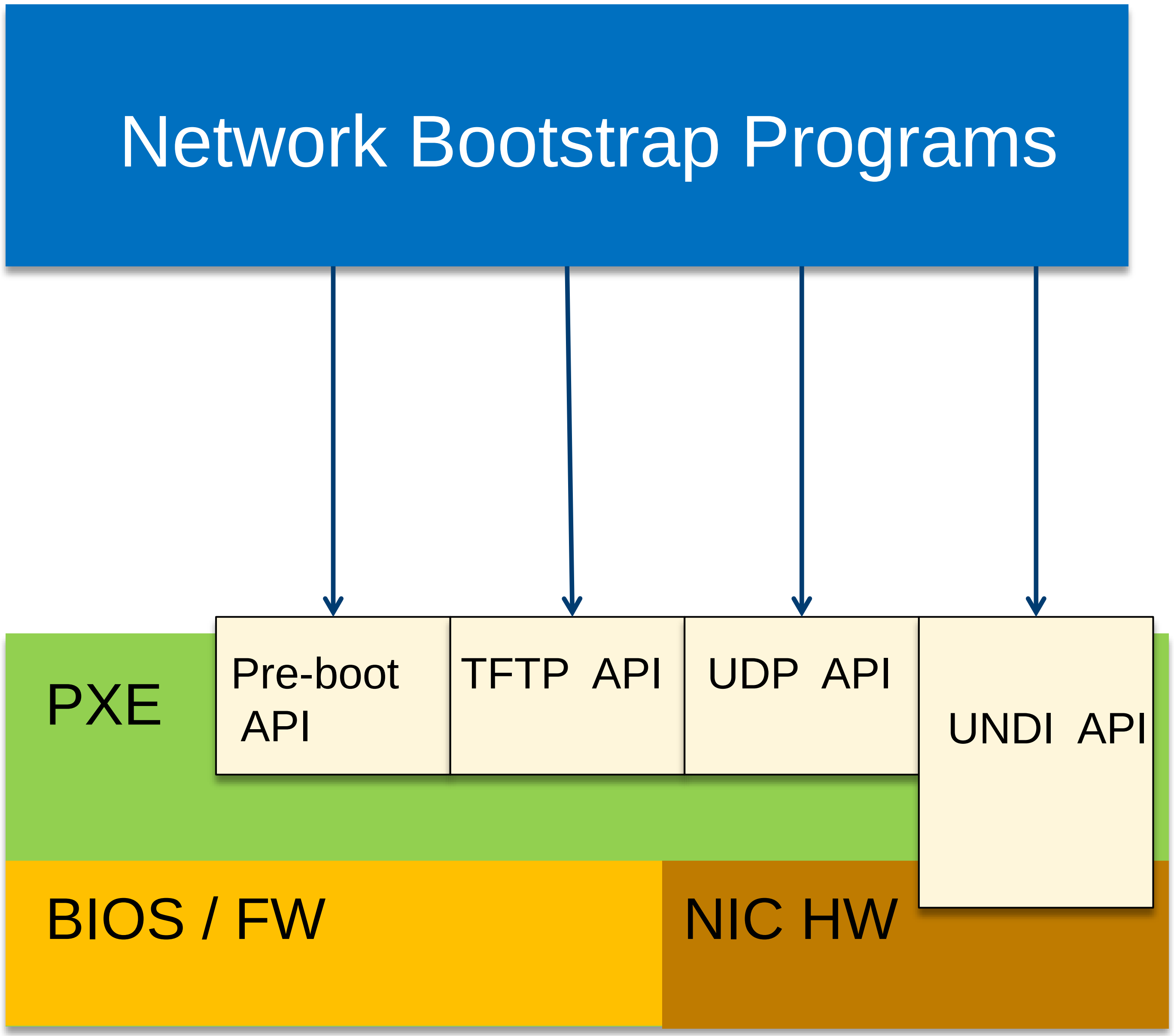
- Preboot eXecution Environment
  - General network booting
    - Independent of data storage device
  - IPv4 based PXE is defined in PXE 2.1
  - IPv6 based PXE is defined in UEFI 2.3
- Technology includes
  - Dual network stack support
    - Evolution of network boot to IPv6 defined
      - IETF RFC 5970
  - DHCP-UUID support
    - Use SMBIOS system GUID as UUID



BUT PXE is not keeping up with modern data center needs

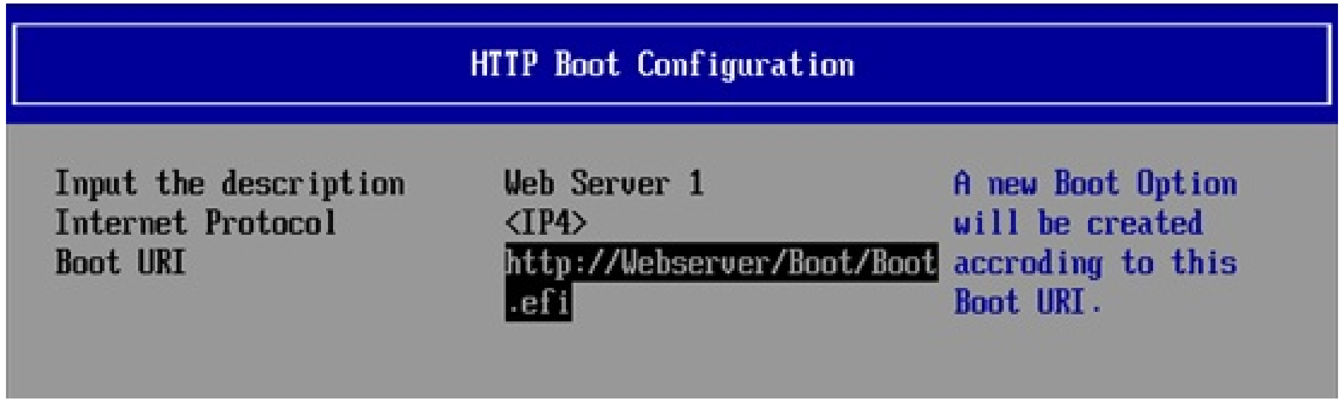
PXE Boot Challenges

- Security Issues
  - Only physical. No encryption or authentication.
  - Rouge DHCP servers, man-in-the-middle attacks
- Scaling issues
  - Circa 1998
  - TFTP timeouts / UDP packet loss
  - Download time = deployment time = $$$
  - Aggravated in density-optimized data centers
- OEMs and users workarounds “*duct-tape*”
  - Chain-load 3rd party boot loaders (iPXE, mini-OS)
  - Alternative Net Booting (SAN, iSCSI, etc.)
- Open source PXE iPXE issues (http://ipxe.org) – pre UEFI 2.5

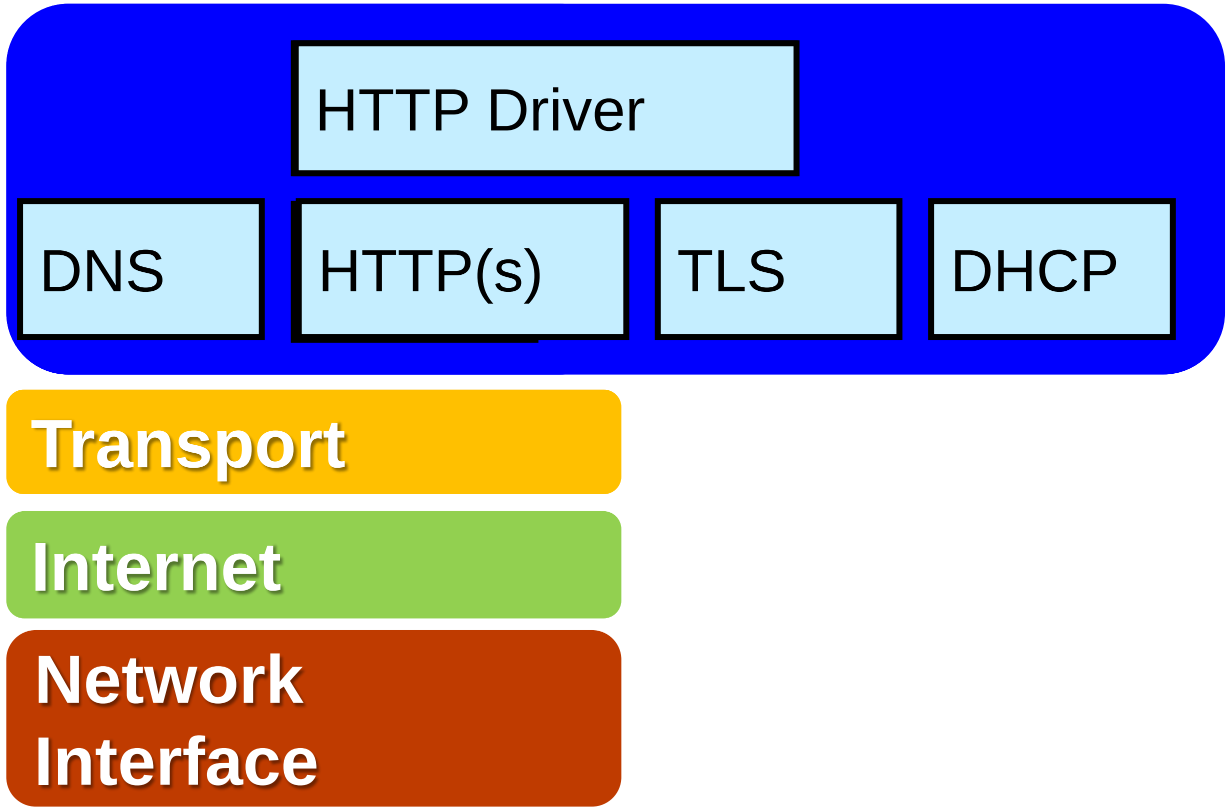


Why not solve PXE Boot challenges natively with standards using UEFI

HTTP(s) Boot Solutions



Add HTTP(s) to Network Stack



UEFI 2.5 defined RAM Disk device path nodes

- Standard access to a RAM Disk in UEFI and Virtual CD (ISO image)

ACPI 6.0 NVDIMM Firmware Interface Table (NFIT)

- Describe the RAM Disks to the OS
- Runtime access of the ISO boot image in memory

HTTP UEFI 2.5 2015

What UEFI Protocols Make Network Work

A deeper dive into the Network implementation in EDK II

Problem Statement

The original UEFI Driver Model is not complete

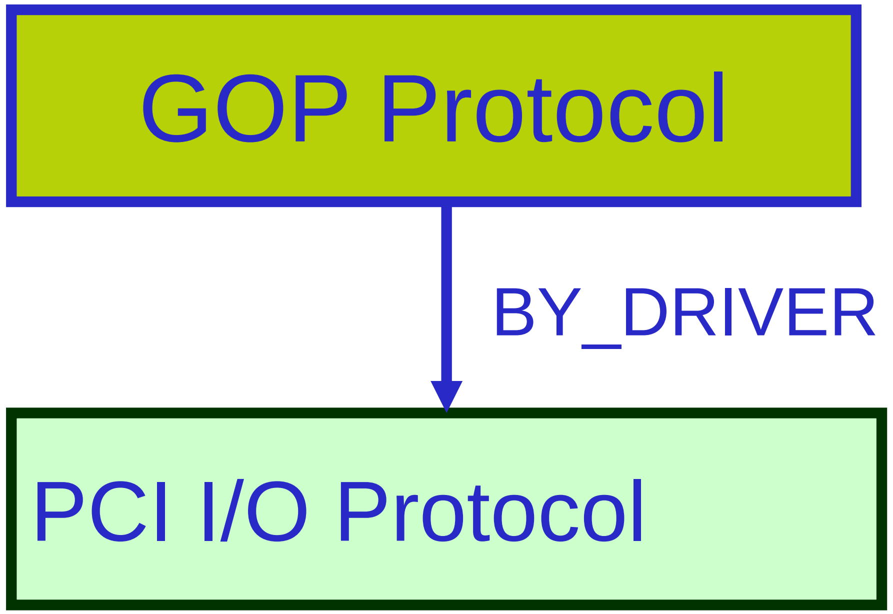
Good support for HW Device Driver

Good support for HW Bus/Hybrid Driver

Good support for Simple SW Layering driver

Poor support for complex SW Layering driver

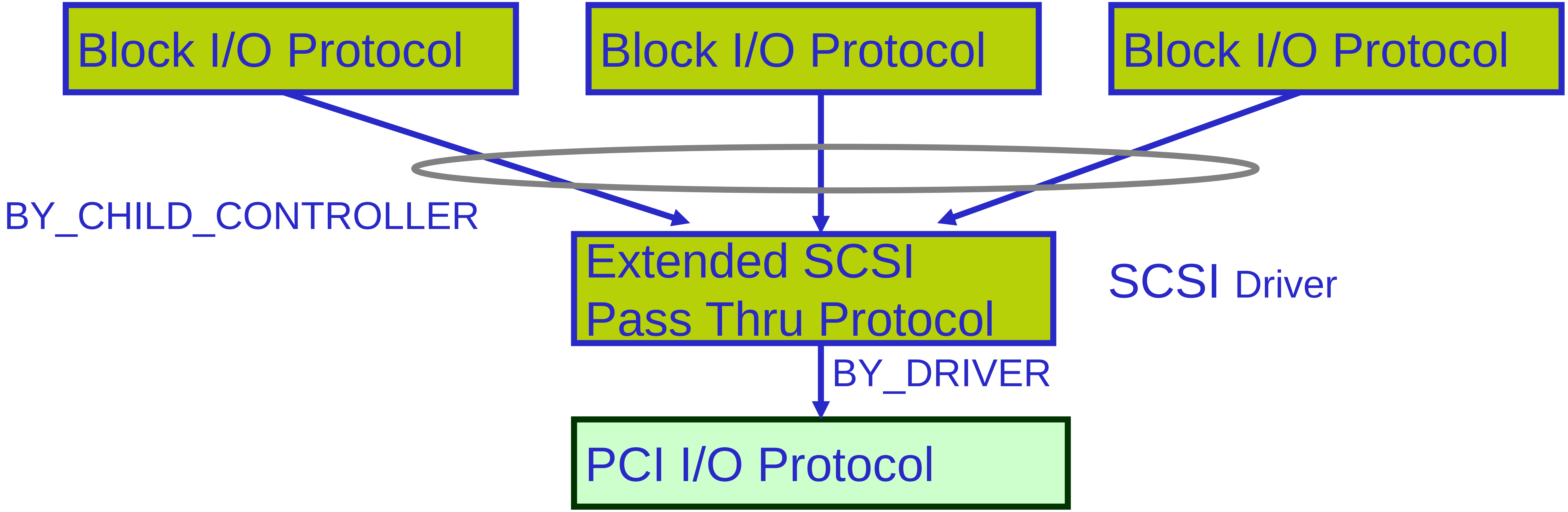
UEFI Hardware Device Driver



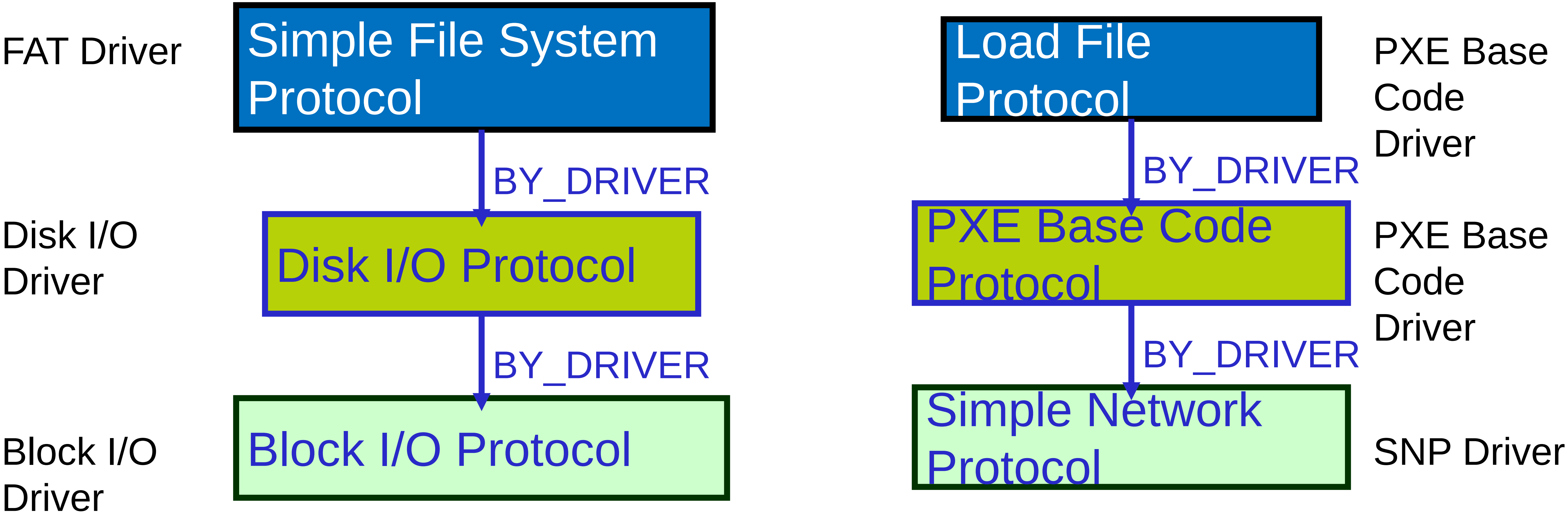
Graphics

PCI Device Driver

UEFI Hardware Bus/Hybrid Driver

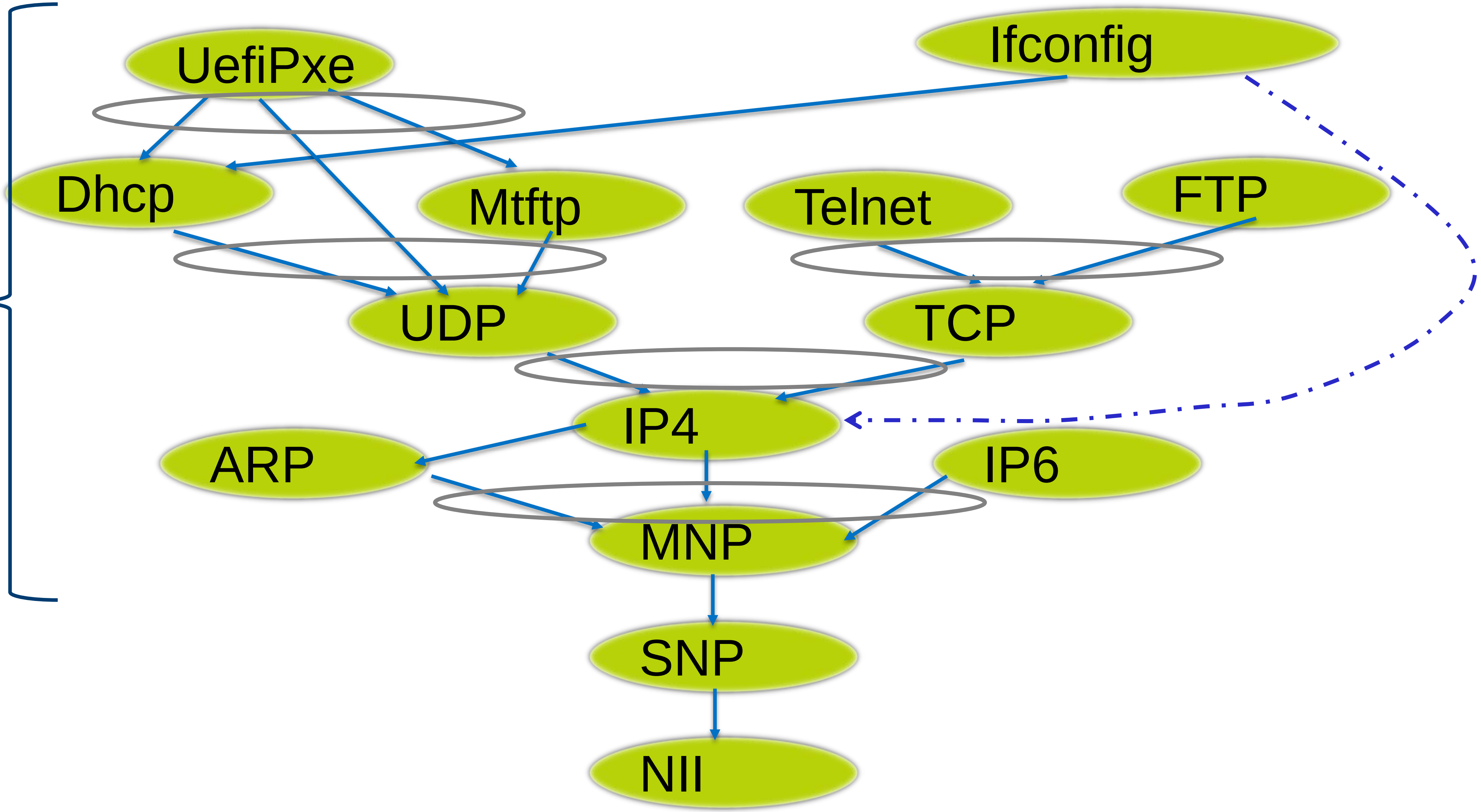


UEFI Simple Software Driver



Complex Software Drivers

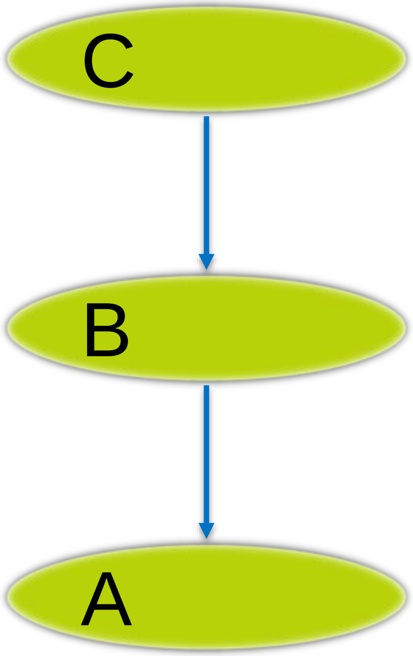
Multiple Consumers



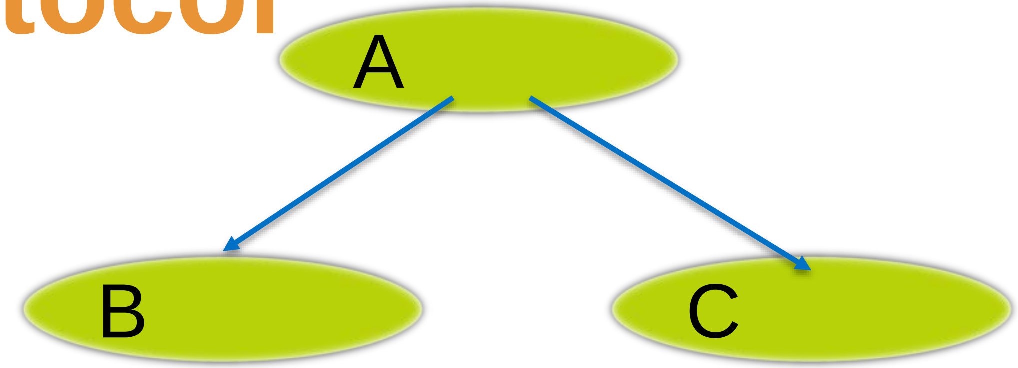
Complex Software Drivers

- OpenProtocol() BY_DRIVER does not support sharing of protocol interfaces
- Number of Children is not fixed
  - Different than enumerable Hardware busses (PCI, ISA)
  - Similar to hot plug Hardware buses (USB)
- Must support Load/Unload at the Network Service Level
- Must support Connect/Disconnect at the Network Service Level

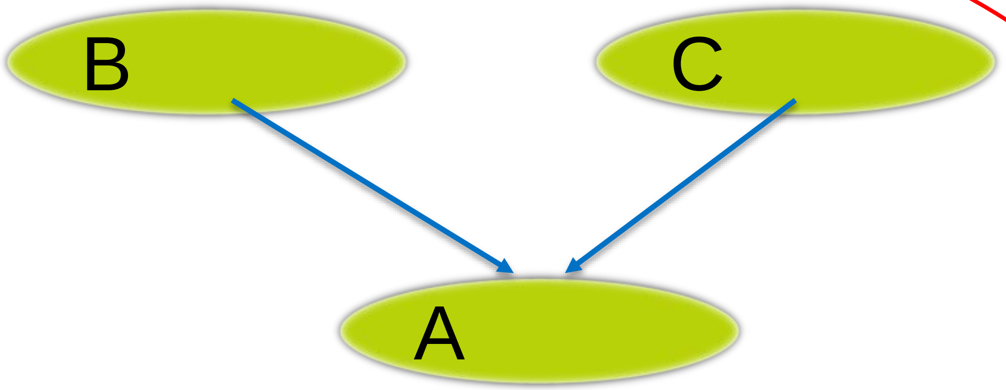
UEFI Service Binding Protocol



Case #1: Linear Stack



Case #2: Multiple Dependencies



Case #3: Multiple Consumers

UEFI Service Binding Protocol
– Complex Software Service Drivers

Multiple Consumers	This Protocol is only produced by drivers that know they will have multiple consumers
No changes	to the existing UEFI Driver Model
Supports	Load/Unload and Connect/Disconnect
Required	Additional protocol is only required for complex software service drivers
Hot-Plug	Software Service Drivers are Hot-Plug Hybrid Drivers Dynamically creates child handles based on number of Consumers

UEFI Service Binding Protocol

```
typedef struct _EFI_SERVICE_BINDING_PROTOCOL {
  EFI_SERVICE_BINDING_CREATE_CHILD   CreateChild;
  EFI_SERVICE_BINDING_DESTROY_CHILD DestroyChild;
} EFI_SERVICE_BINDING_PROTOCOL;
EFI_STATUS
```

```
EFI_STATUS
(EFIAPI *EFI_SERVICE_BINDING_CREATE_CHILD) (
  IN EFI_SERVICE_BINDING_PROTOCOL*This,
  IN OUT EFI_HANDLE*ChildHandle
);
EFI_STATUS
(EFIAPI *EFI_SERVICE_BINDING_DESTROY_CHILD) (
  IN EFI_SERVICE_BINDING_PROTOCOL*This,
  IN EFI_HANDLEChildHandle
);
```

UEFI Service Binding Protocol

- CreateChild()
  - Synchronous hot-plug add event
- DestroyChild()
  - Synchronous hot-plug remove event
- Consumers
  - Test for UEFI Service Binding Protocols for the software services the consumer depends upon
  - Call CreateChild() from Start() for each dependent software service
  - Creates a child handle with one or more software services
  - Call DestroyChild() from Stop() for each dependent software service

EDK II Only Supports Polling (NO Interrupts)

- The IO engine of the whole s POLLING, either timer driven or invoked in applications.
- **Asynchronous** – da two part, similar with the asynchronous IO in **request**; 2. wait for the result of the IO requ
- Most drivers suppor transmission.
- Separated IP, Route s based on IP, UDP, TCP, MTFTP and DHCP (B
- Instances are bound to a sharing mechanism in layers spanning over multiple NICs.

Deferred Procedure Call Protocol (DPC)

Method	• Used by UEFI Drivers to **queue** a deferred procedure call at a **lower TPL**
TPL levels	• Used by UEFI Drivers with event notification functions that execute at high TPL levels, and require the use of services that must be executed at **lower TPL levels** (i.e. Call Back)
Implementation	• EDK II specific implementation defined in *MdeModulePkg/Include/Protocol/Dpc.h*.

	TPL Levels
0	Application
1	Call Back
2	Notify
3	High

EDK II DPC Protocol

Queue DPC

```
typtypedef
EFI_STATUS
(EFIAPI *EFI_DPC_QUEUE_DPC)(
  IN EFI_DPC_PROTOCOL    *This,
  IN EFI_TPL             DpcTpl,
  IN EFI_DPC_PROCEDURE   DpcProcedure,
  IN VOID                *DpcContext    OPTIONAL
  );
```

Dispatch DPC

```
typedef
EFI_STATUS
(EFIAPI *EFI_DPC_DISPATCH_DPC)(
  IN EFI_DPC_PROTOCOL  *This
  );
```

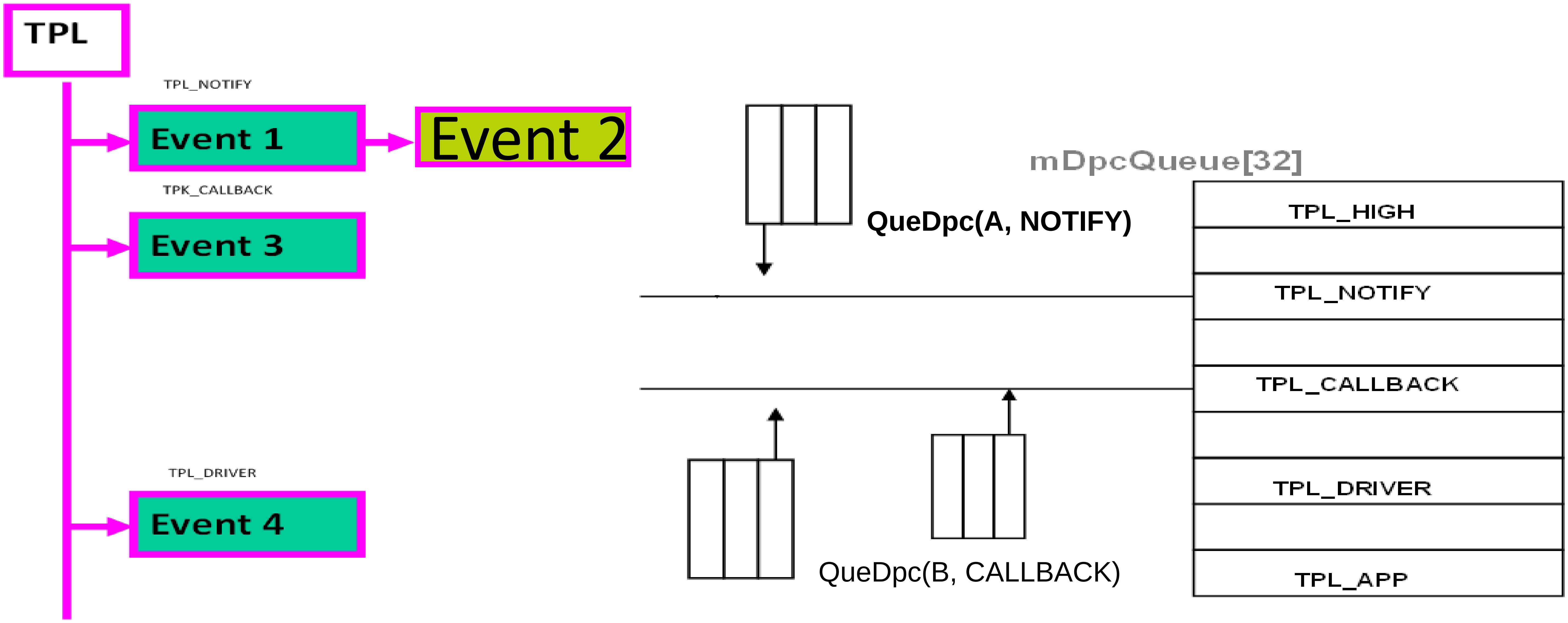
WHY DPC

iSCSI example

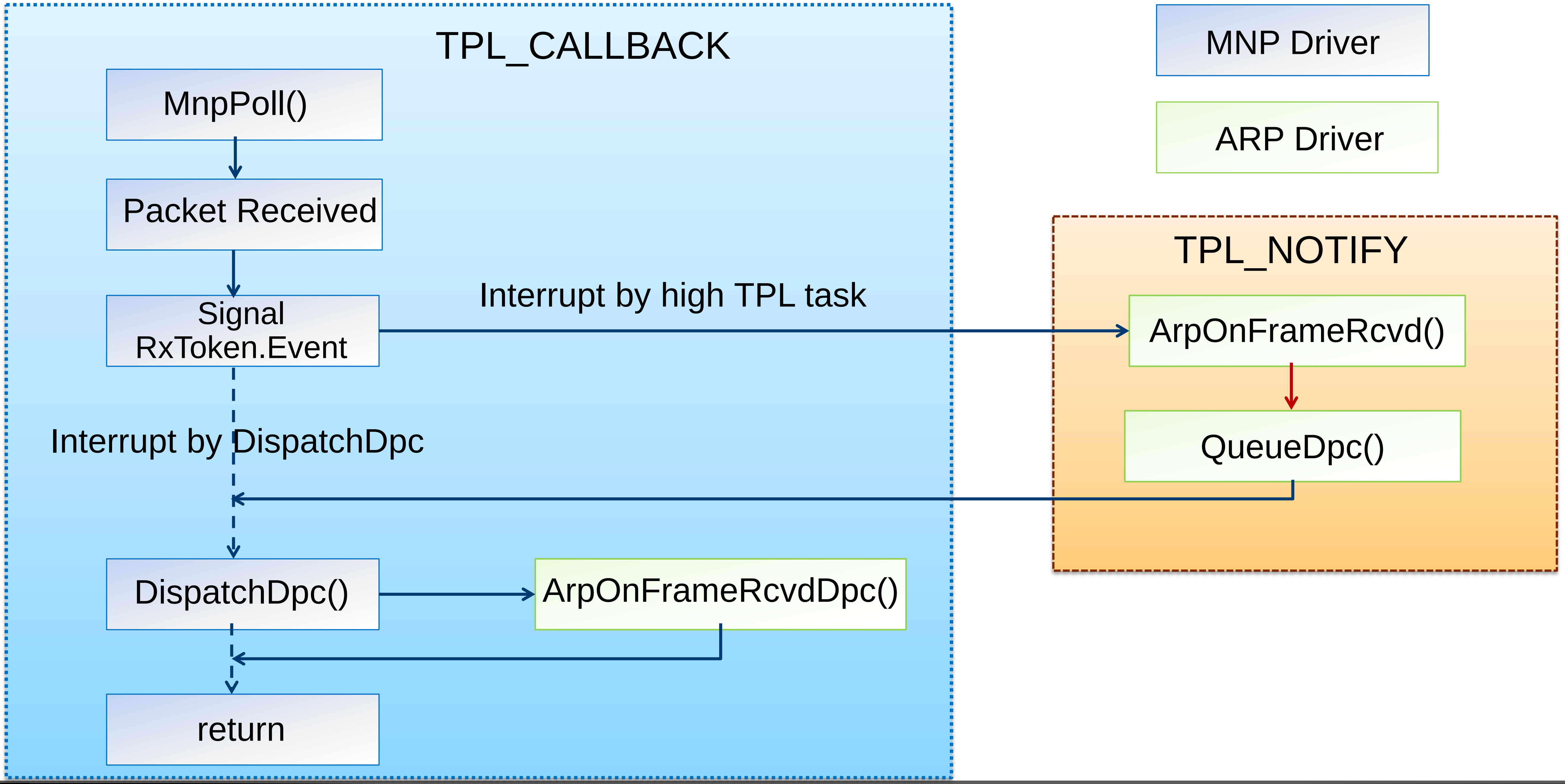
UEFI File System Driver	Synchronous
UEFI Block IO Driver	
UEFI SCSI Disk I/O Driver	
UEFI SCSI Bus Driver	
UEFI iSCSI	Synchronous
TCPv(4|6)	Asynchronous
Ipv(4|6)	
MNP (Managed Network Protocol)	
SNP (Simple Network Protocol)	
UNDI (OEM Supplied Network Driver)	

DPC Solves the TPL deadlock issue in UEFI network stack

How DPC Solves the Problem

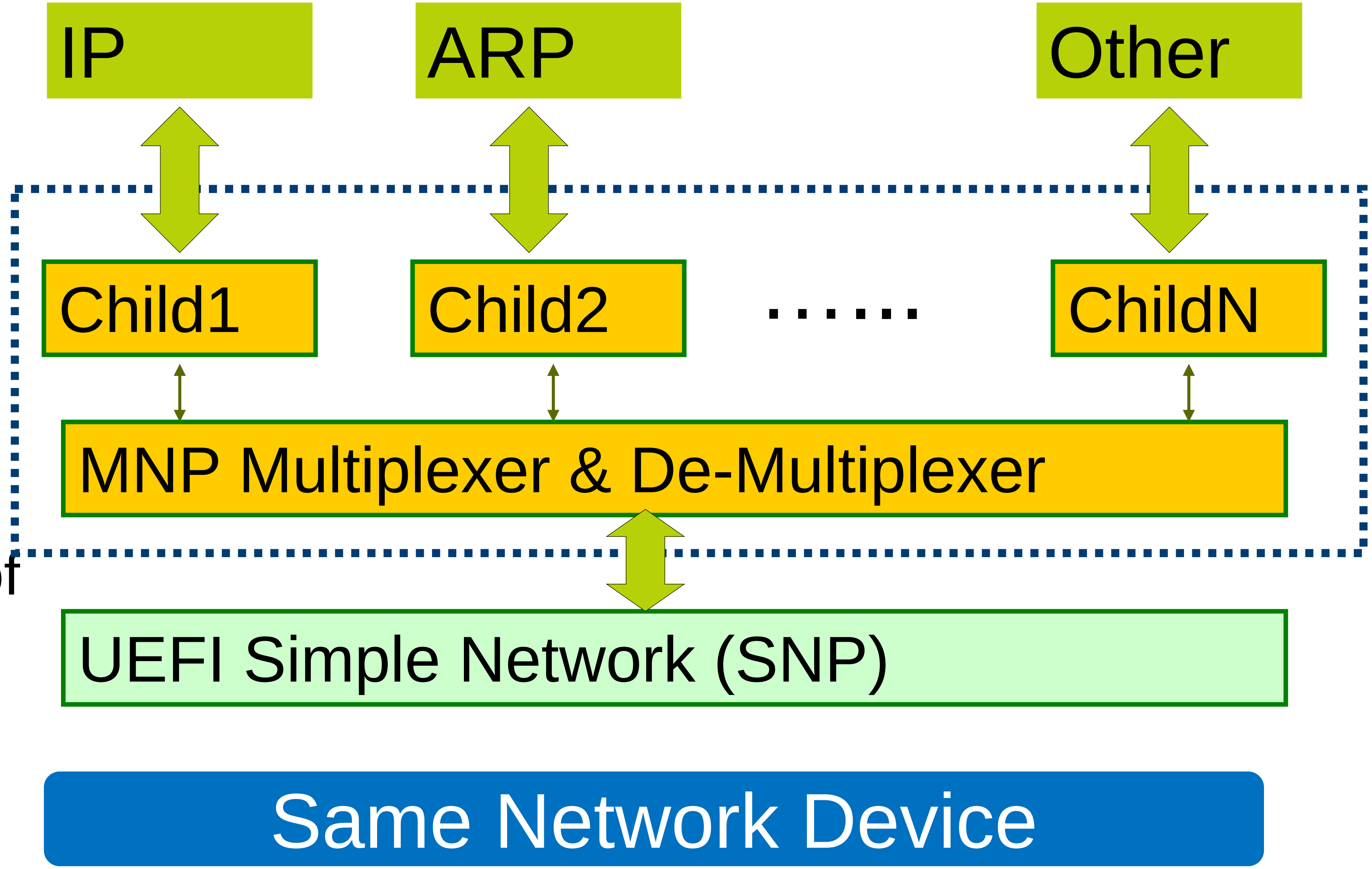


How DPC Solves the Problem -ARP Example



UEFI Managed Network Protocol (MNP)

- Provides raw (unformatted) asynchronous network packet I/O services and Managed Network Service Binding Protocol
- Used to locate communication devices that are supported by an MNP driver.
- Create and destroy instances of the MNP child protocol driver that can use the underlying communications devices.



UEFI MNP

Transmission:

- Transmitting packets is simple in the MNP driver - if MNP just appends the media header to the packets and send it via SNP regardless of the return status of the SNP sending function.

Receiving:

- Fetch - a buffer unit from the buffer pool, call SNP.Receive() to receive packets from SNP.
- Delivery -Try to deliver the packet to the appropriate receivers.
- Iterate - the MNP children, and check the receiving filters of the children against the packet. If matched and there is a receive token submitted by the upper layer protocol, wrap the received packet, queue it into the delivered queue, fill the receive token and signal the event in the token to notify the upper layer protocol.
- Post - Post receiving: recycle the buffer when the recycle event is signaled by the upper layer protocol.

System Polling:

- Use a timer event to periodically poll the SNP driver
- Guarantee in-sequence packets delivery.
- Intelligent Poll

Buffer management

- The MNP driver needs pre-allocation of enough buffer units for receiving to reduce the overhead.

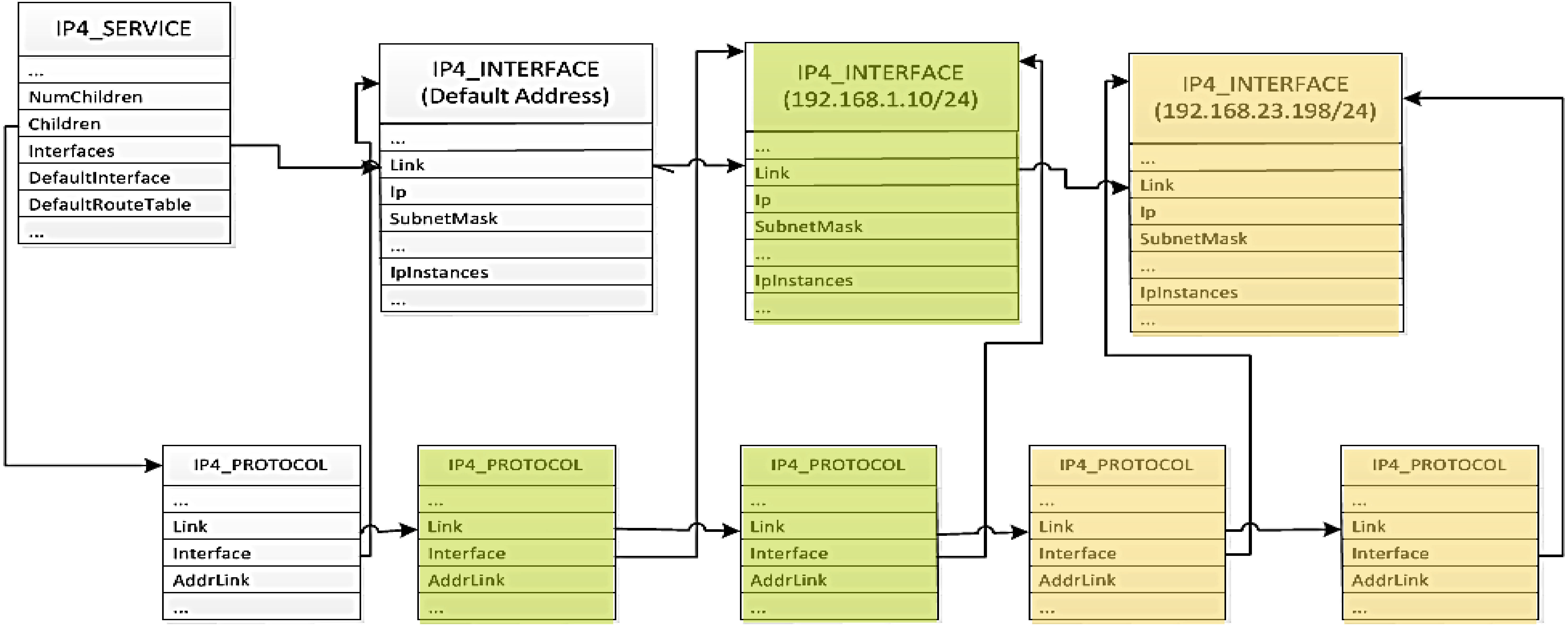
IPv4 Driver – in Detail

Ip4Common.c	Common helper routines for the whole driver
Ip4Driver.c	DriverBinding and ServiceBinding routines
Ip4Icmp.c	Internet Control Message Protocol ICMP related routines
Ip4If.c	IP pseudo interface related routines
Ip4Igmp.c	Internet Group Management Protocol IGMP related routines
Ip4Impl.c	Codes for the APIs defined and exposed by EFI_IP4_PROTOCOL
Ip4Input.c	IP packets input (receive) procedure
Ip4Output.c	IP packets output (transmit) procedure
Ip4Route.c	Route table related routines
ComponentName.c	Component name…

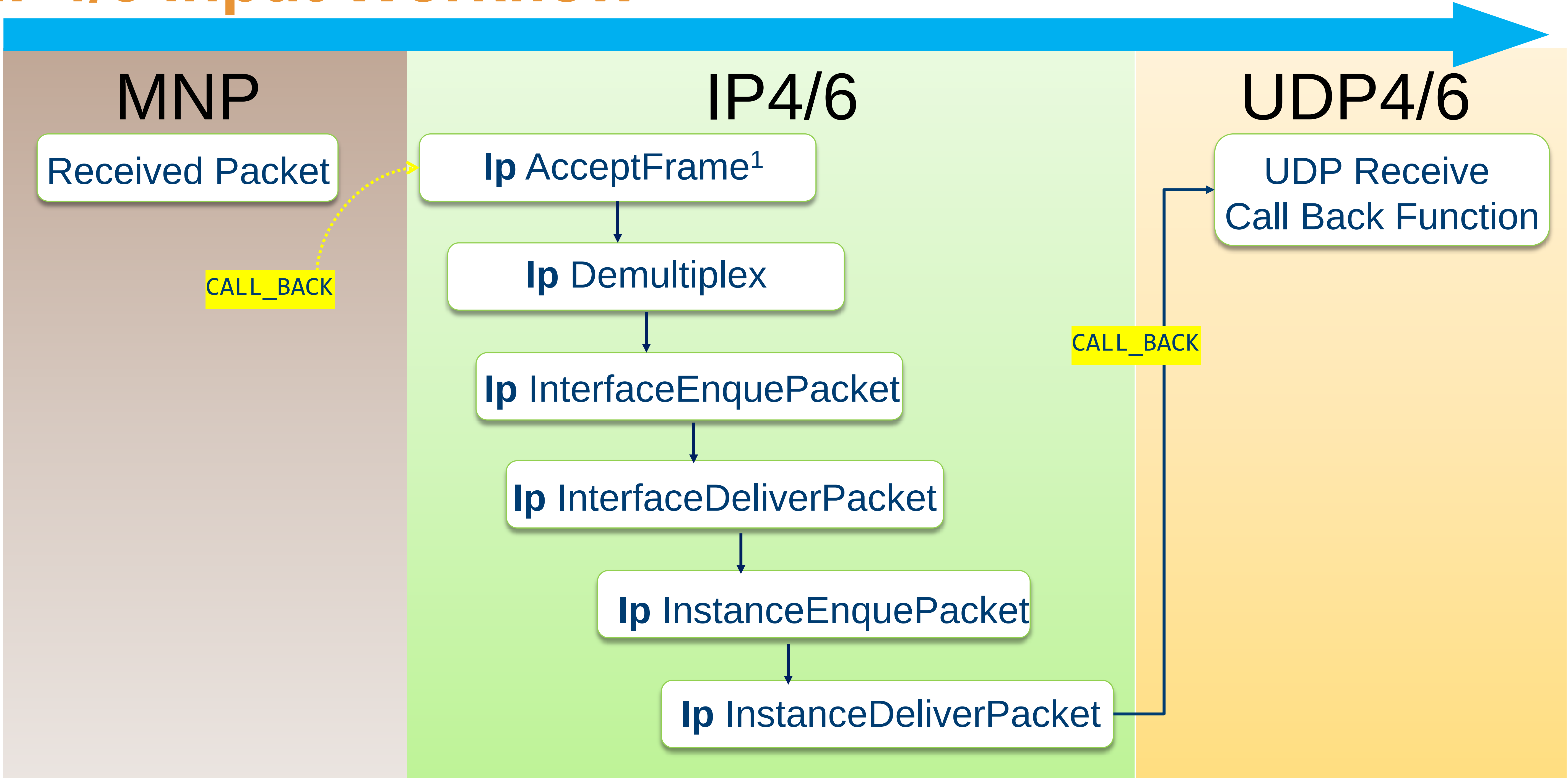
IPv6 Driver – In Detail

Ip6Common.c	Common helper routines for the whole driver
Ip6Driver.c	DriverBinding and ServiceBinding routines
Ip6Icmp.c	Internet Control Message Protocol ICMP related routines
Ip6If.c	IP pseudo interface related routines
Ip6Nd.c	Neighbor Discovery ND related routines
Ip6Mld.c	Multicast Listener Discovery MLD related routines
Ip6Impl.c	Codes for the APIs defined and exposed by EFI_IP6_PROTOCOL
Ip6Input.c	IP packets input (receive) procedure
Ip6Output.c	IP packets output (transmit) procedure
Ip6Route.c	Route table related routines
ComponentName.c	Component name…

IP Internal Structure – Example IPv4



IP4/6 Input Workflow

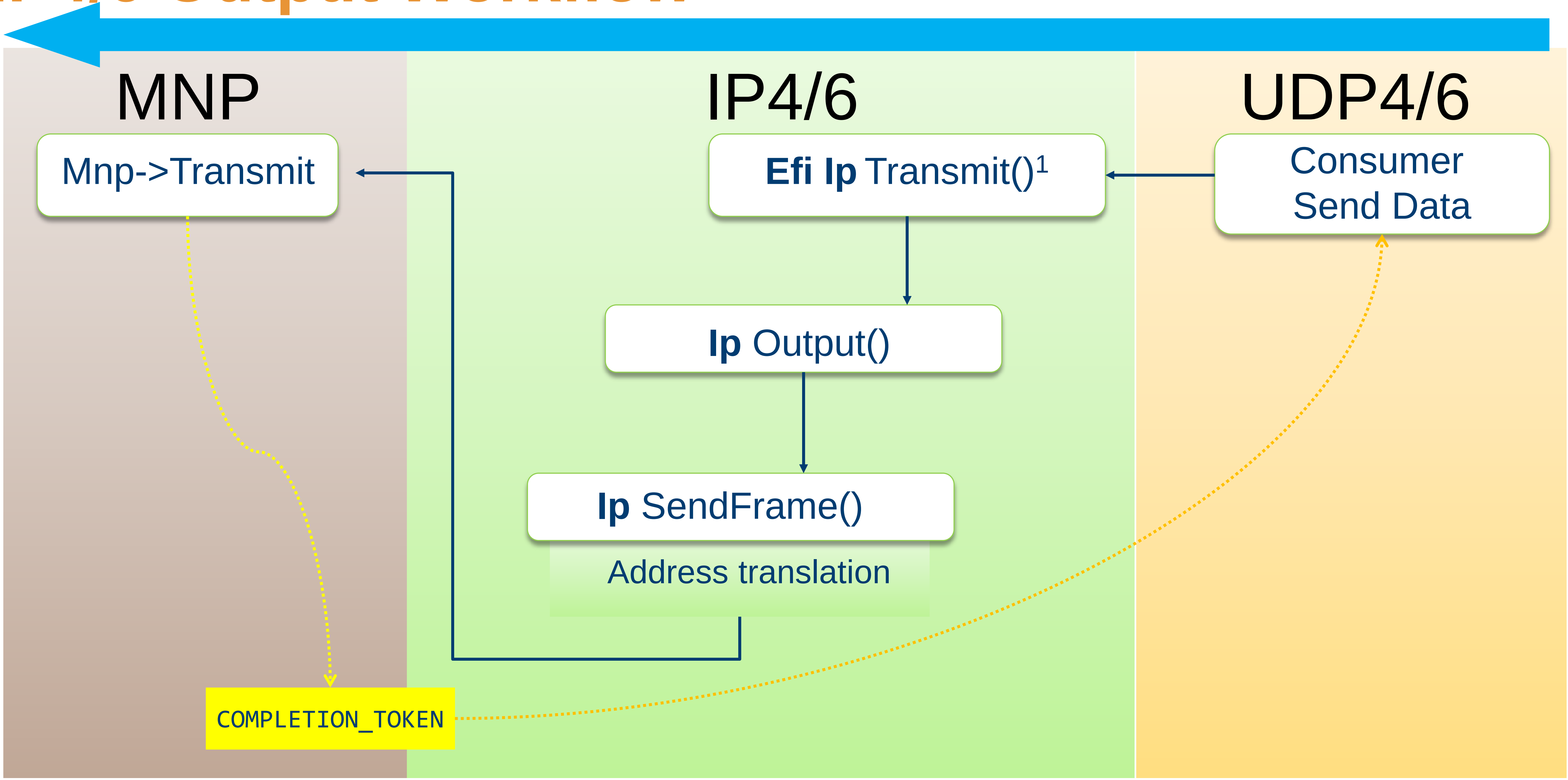


[1] IPv4 – `Ip4AccpetFrame()`
IPv6 – `Ip6AcceptFrame()`

IP4/6 Input Workflow

- Ip4ReceiveFrame ()
  - Request to receive the packet from the interface. Wrap receive procedure related information into an IP4_LINK_RX_TOKEN. On completion of the receive, Ip4OnFrameReceived () and Ip4OnFrameReceivedDpc (). After some simple check is done, the CallBack function provided will be called with a packet, or some error information. Normally, the CallBack function is Ip4AcceptFrame ().
- Ip4AcceptFrame ()
  - Most work of processing a received IP packet is done here.
    - Sanity check: version number, checksum, size, options, etc.
    - Reassemble fragments of an big IP packet.
    - Invoke IGMP or ICMP processing if it’s an IGMP or ICMP packet respectively, else invoke Ip4Demultiplex () to demultiplex the packet to any IP4 instance who is interested in it.
  - the receive procedure by calling Ip4ReceiveFrame ().
- Ip4Demultiplex ()
  - Iterate all the Ip4 Interfaces, and calling Ip4InterfaceEnquePacket () to let every Ip4 Interface check whether this IP4 packet matches their receive filter. If matches, enqueue it; else, do nothing.
  - If any IP4 interface is willing to receive this packet, iterate all the IP4 interfaces and call Ip4InterfaceDeliverPacket ().
- Ip4InterfaceEnquePacket ()
  - Match cast type first, such as, Multicast, Broadcast or Promiscuous.
  - Iterate all the Ip4 instances belonging to this Interface by calling Ip4InstanceEnquePacket () to check whether there is instance interested in this packet.
- Ip4InterfaceDeliverPacket ()
  - Iterate all the IP4 instances and call Ip4InstanceDeliverPacket () to deliver any queued packets of the instances
- Ip4InstanceEnquePacket ()
  - Check whether this instance is willing to receive the current IP packet. If all match rules are passed, clone this packet and put it into the Received list.
- Ip4InstanceDeliverPacket ()
  - Deliver all the queued IP packets to the consumer of this IP4 protocol by every time putting an IP packet into a consumer provided EFI_IP4_COMPETION_TOKEN, and finally signal the event in the token. The end condition is either there is no IP packet or no consumer provided token.
  - The consumer provided tokens for receive purpose come from the callings to EFI_IP4_PROTOCOL.Receive ().

IP4/6 Output Workflow



¹EFI_IP4_PROTOCOL.Transmit() or EFI_IP6_PROTOCOL.Transmit()

IP4/6 Output Workflow

- EfiIp4/6Transmit () – EFI_IP4/6_PROTOCOL.Transmit ()
  - The initial source of most IP4/6 output packets.
  - Sanity check.
  - Append IP4/6 head to the buffer passed in.
  - Record the Token and the corresponding Wrap (IP4/6_TXTOKEN_WRAP) to the TxTokens NET_MAP.
  - Call Ip4/6Output to output this packet.
- Ip4/6Output ()
  - Decide the IP address of the next hop for this packet, aka, route selection.
  - Fragment it if needed.
  - Call Ip4/6SendFrame () to send this packet.
- Ip4/6SendFrame ()
  - Do layer-3 to layer-2 address translation, in most cases need the help from the Arp instance belonging to the IP4 interface selected.
    - Immediately send out the packet by Mnp->Transmit if the translation can be done directly from the arp cache.
    - Or create an ArpQue which assembles all the IP4 packets with the same next hop. Once the translation is done, all these packets are transmitted out in the event notify function.
- Upon completion of the transmit, either successful or with some failure, the status will be returned to the consumer provided EFI_IP4/6_COMPLETION_TOKEN.Status with the signaling of the event, and at this time, the control of the token is returned to the consumer.

UEFI HTTP(S) BOOT OVERVIEW

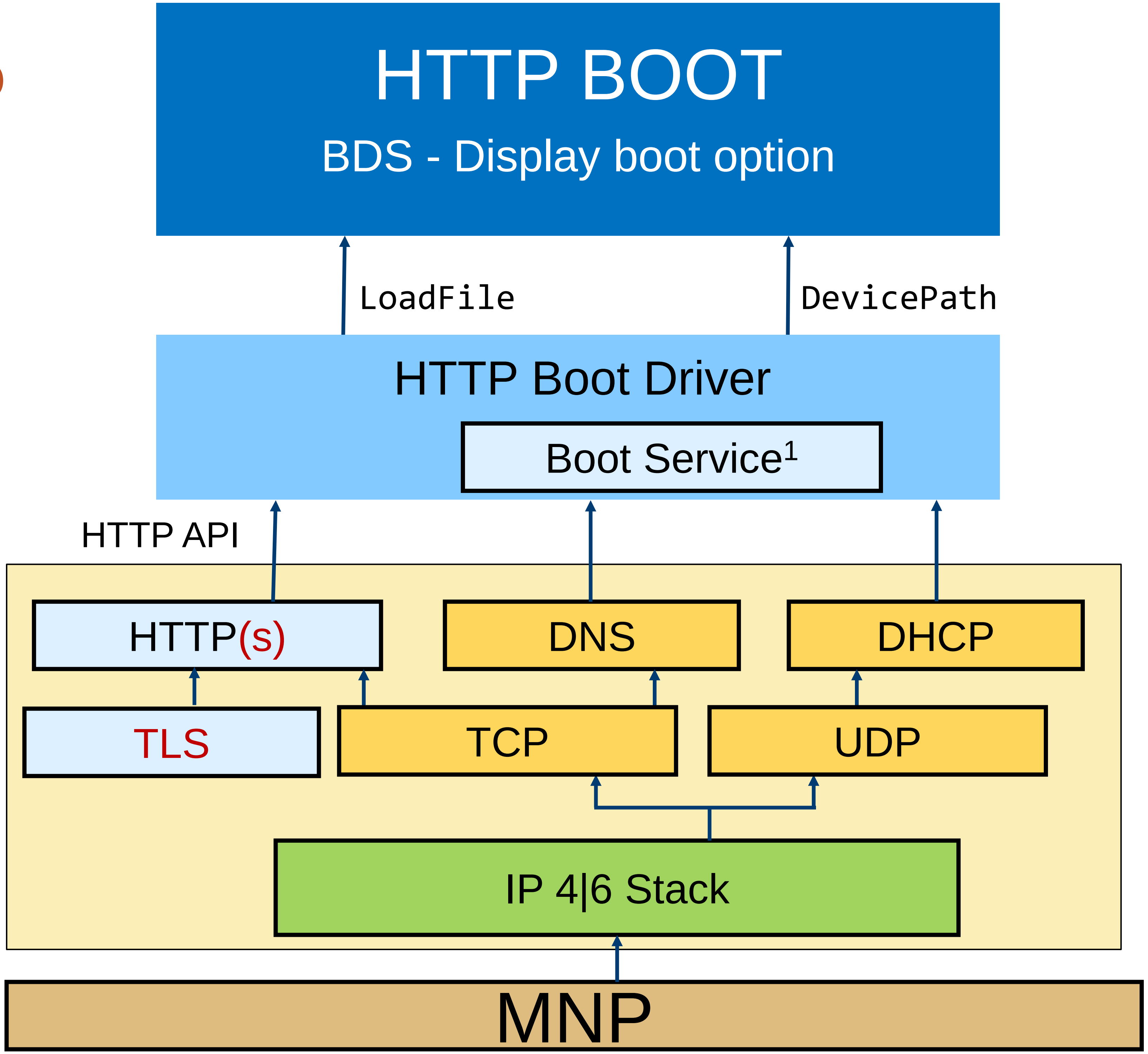
UEFI HTTP Boot Overview
-HTTP protocol for network booting

- HTTP can handle much larger files than TFTP, and scale to much larger distances. You can easily download multi-megabyte files, such as a Linux kernel and a root file system, and you can download from servers that are not on your local area network
- Booting from HTTP is as simple as replacing the DHCP filename field with an http:// URL
  - Specify URI[1]-based pointers to the "Network Boot Program (NBP)", the binary image to download and run, which can be used using HTTP instead of TFTP
- DHCP Servers will need to support HTTP Boot

[1]Uniform Resource Identifiers (URI)
[1]Uniform Resource Locator (URL)

HTTP(s) Boot UEFI 2.5
- Network Stack

- DNS IPv4 / IPv6
- HTTP IPv4 / IPv6
- TLS for HTTPs
- HTTP Boot Driver

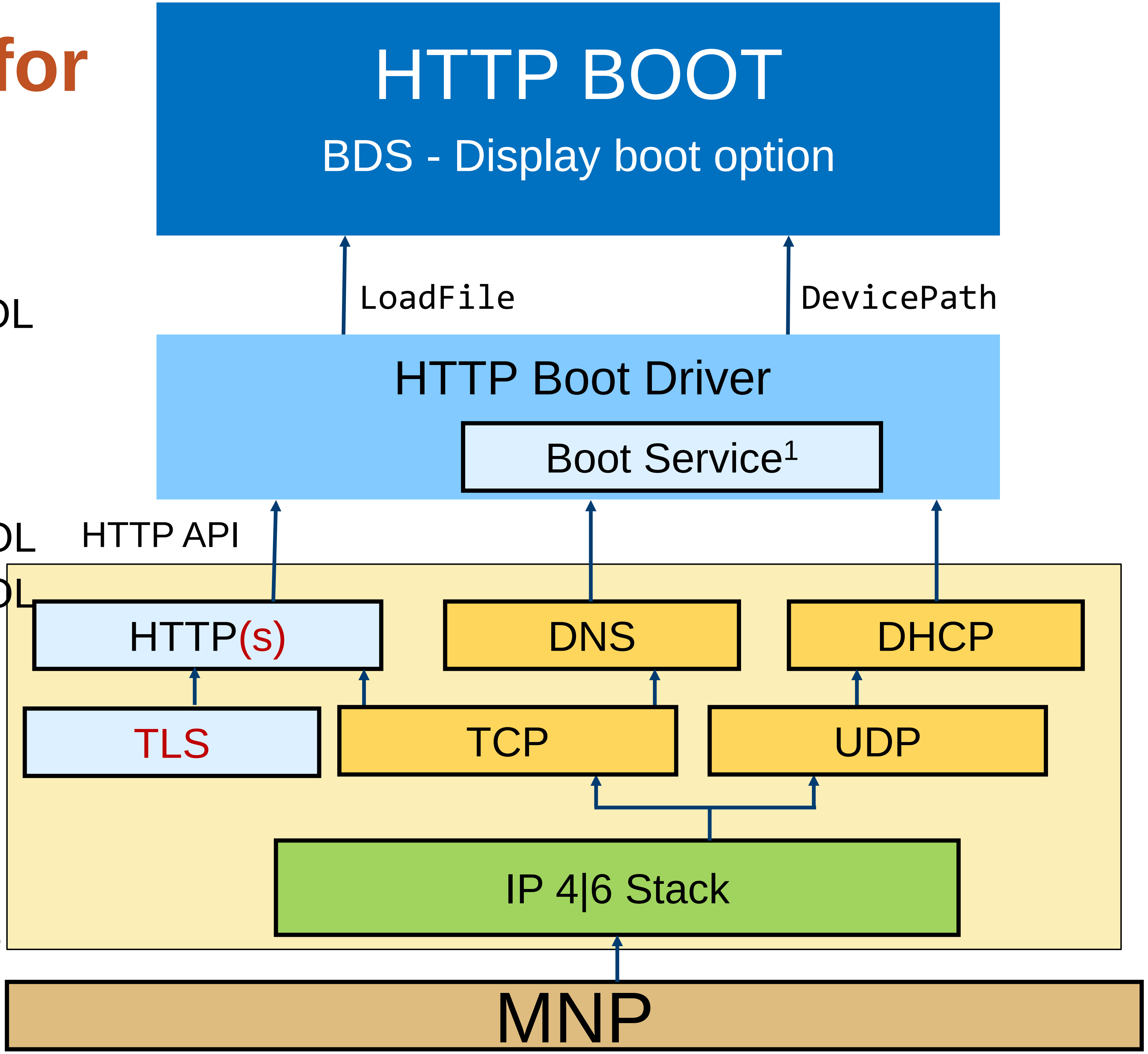


UEFI 2.5
2015

[1]Boot Service Discovery / Configuration

UEFI & EDK II Protocols for HTTP(s) Network Stack

- HTTP support
  - EFI_HTTP_SERVICE_BINDING_PROTOCOL
  - EFI_HTTP_PROTOCOL
  - EFI_HTTP_UTILITIES_PROTOCOL
- DNS Support
  - EFI_DNS4_SERVICE_BINDING_PROTOCOL
  - EFI_DNS6_SERVICE_BINDING_PROTOCOL
  - EFI_DNS4_PROTOCOL
  - EFI_DNS6_PROTOCOL
  - EFI_IP4_CONFIG2_PROTOCOL
  - EFI_IP6_CONFIG_PROTOCOL
- TLS support
  - EFI_TLS_SERVICE_BINDING_PROTOCOL
  - EFI_TLS_PROTOCOL
  - EFI_TLS_CONFIGURATION_PROTOCOL



[1]Boot Service Discovery / Configuration

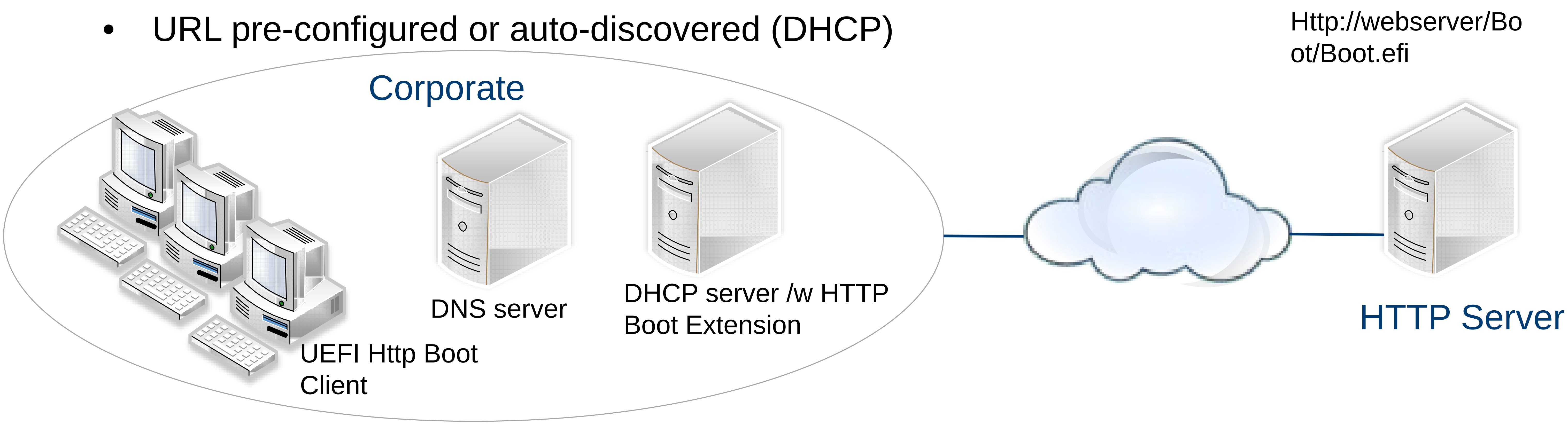
UEFI Native HTTP Boot – Corporate Environment

HTTP Protocols

- Boot from a configured URL
- Target can be:
  - UEFI Network Boot Program (NBP)
  - Shrink-wrapped ISO image
- URL pre-configured or auto-discovered (DHCP)

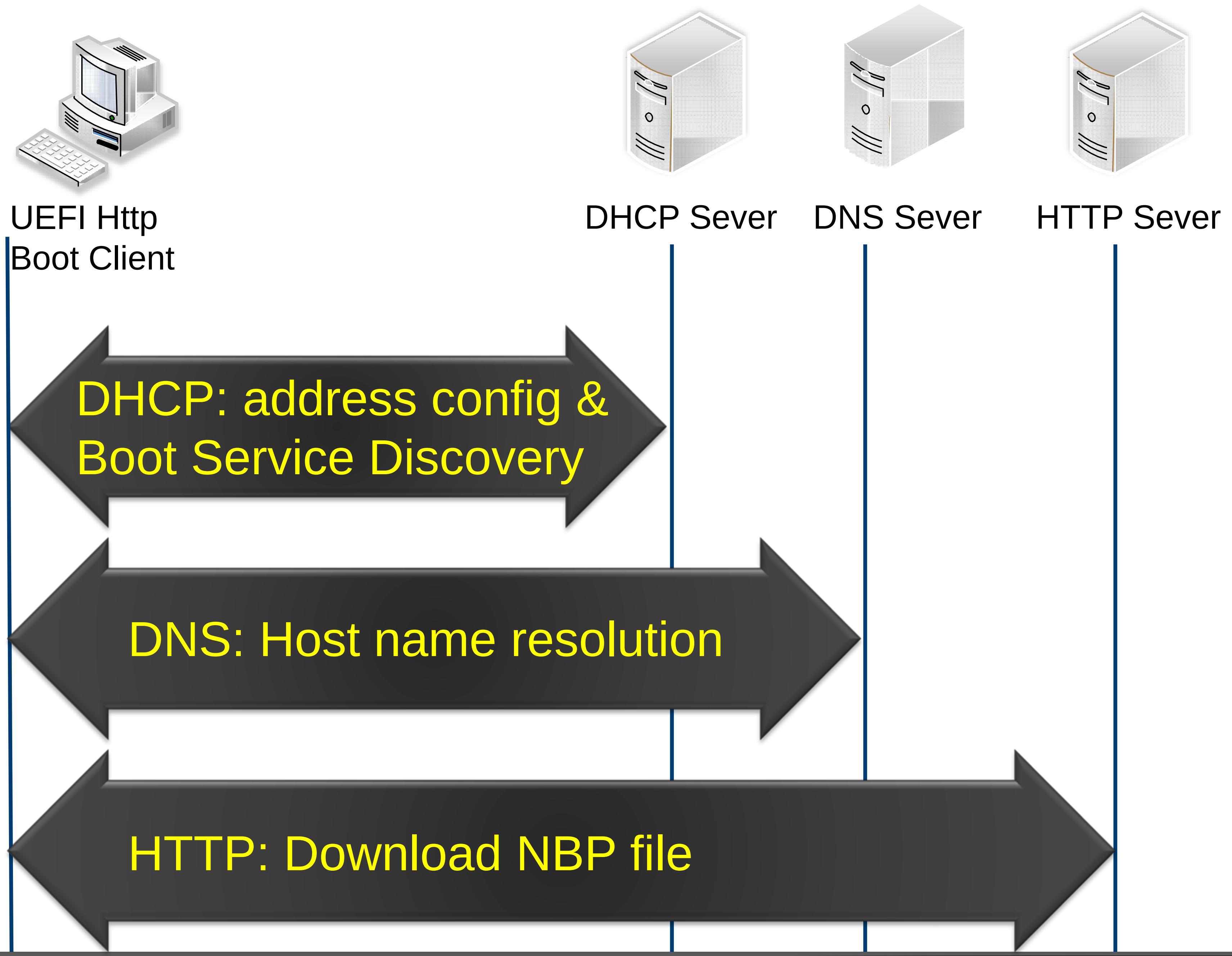
Addresses PXE issues

- HTTP(s) addresses security
- TCP reliability
- HTTP load balancing



HTTP Boot DHCP Discovery

- New HTTP Boot “Architectural Types” to distinguish from PXE
- Client sends DHCP Discover **request**
- DHCP Server responds with **offer** that includes the boot file URI
- Clients resolves URI server name from **DNS**
- Client downloads boot image from HTTP server using HTTP(s)



HTTP(s) Boot Discovery - Architectural Types

- DHCP - http://www.iana.org/assignments/dhcpv6-parameters/dhcpv6-parameters.xml
- IPv4/IPv6 DHCP Discover request
  - DHCP Option 93: Client system Architecture
  - DHCPv6 Option 61: Client system Architecture
    - 0x10 = x64 UEFI boot from HTTP
    - 0x0F = x86 UEFI boot from HTTP
- Server responds with DHCPOFFER that includes the boot file HTTP URI for the requested processor architecture

Processor Architecture Types

Registration Procedure(s)
Expert Review
Expert(s)
Vincent Zimmer, Bernie Volz, Tomek Mrugalski
Reference
[RFC5970]
Available Formats
CSV

Type	Architecture Name	Reference
0x00 0x00	x86 BIOS	[RFC5970][RFC4578]
0x00 0x01	NEC/PC98 (DEPRECATED)	[RFC5970][RFC4578]
0x00 0x02	Itanium	[RFC5970][RFC4578]
0x00 0x03	DEC Alpha (DEPRECATED)	[RFC5970][RFC4578]
0x00 0x04	Arc x86 (DEPRECATED)	[RFC5970][RFC4578]
0x00 0x05	Intel Lean Client (DEPRECATED)	[RFC5970][RFC4578]
0x00 0x06	x86 UEFI	[RFC5970][RFC4578]
0x00 0x07	x64 UEFI	[RFC5970][RFC4578]
0x00 0x08	EFI Xscale (DEPRECATED)	[RFC5970][RFC4578]
0x00 0x09	EBC	[RFC5970][RFC4578]
0x00 0x0a	ARM 32-bit UEFI	[RFC5970]
0x00 0x0b	ARM 64-bit UEFI	[RFC5970]
0x00 0x0c	PowerPC Open Firmware	[Thomas_Huth]
0x00 0x0d	PowerPC ePAPR	[Thomas_Huth]
0x00 0x0e	POWER OPAL v3	[Jeremy_Kerr]
0x00 0x0f	x86 uefi boot from http	[Samer_El-Haj-Mahmoud]
0x00 0x10	x64 uefi boot from http	[Samer_El-Haj-Mahmoud]
0x00 0x11	ebc boot from http	[Samer_El-Haj-Mahmoud]
0x00 0x12	arm uefi 32 boot from http	[Samer_El-Haj-Mahmoud]
0x00 0x13	arm uefi 64 boot from http	[Samer_El-Haj-Mahmoud]
0x00 0x14	pc/at bios boot from http	[Samer_El-Haj-Mahmoud]
0x00 0x15	arm 32 uboot	[Joseph_Shifflett]
0x00 0x16	arm 64 uboot	[Joseph_Shifflett]

iPXE – UEFI HTTP Chainloading

UEFI HTTP Boot client to chainload iPXE from an HTTP server (HTTP boot to iPXE then run iPXE to HTTP download)

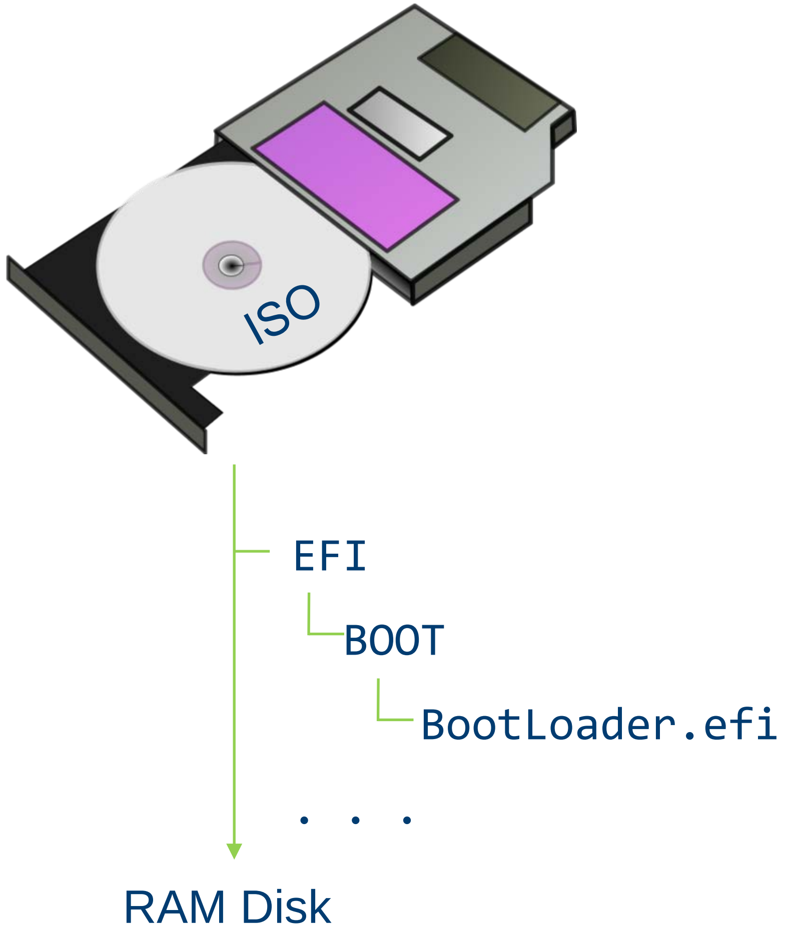
- Eliminates need for separate TFTP server
- UEFI HTTP Boot client will download and boot iPXE
- iPxe offers advanced features to download and boot OS

- Application note: http://ipxe.org/appnote/uefihttp

2 Options to address the PXE challenges: Native UEFI HTTP Boot and iPXE using UEFI HTTP

RAM Disk Boot from HTTP

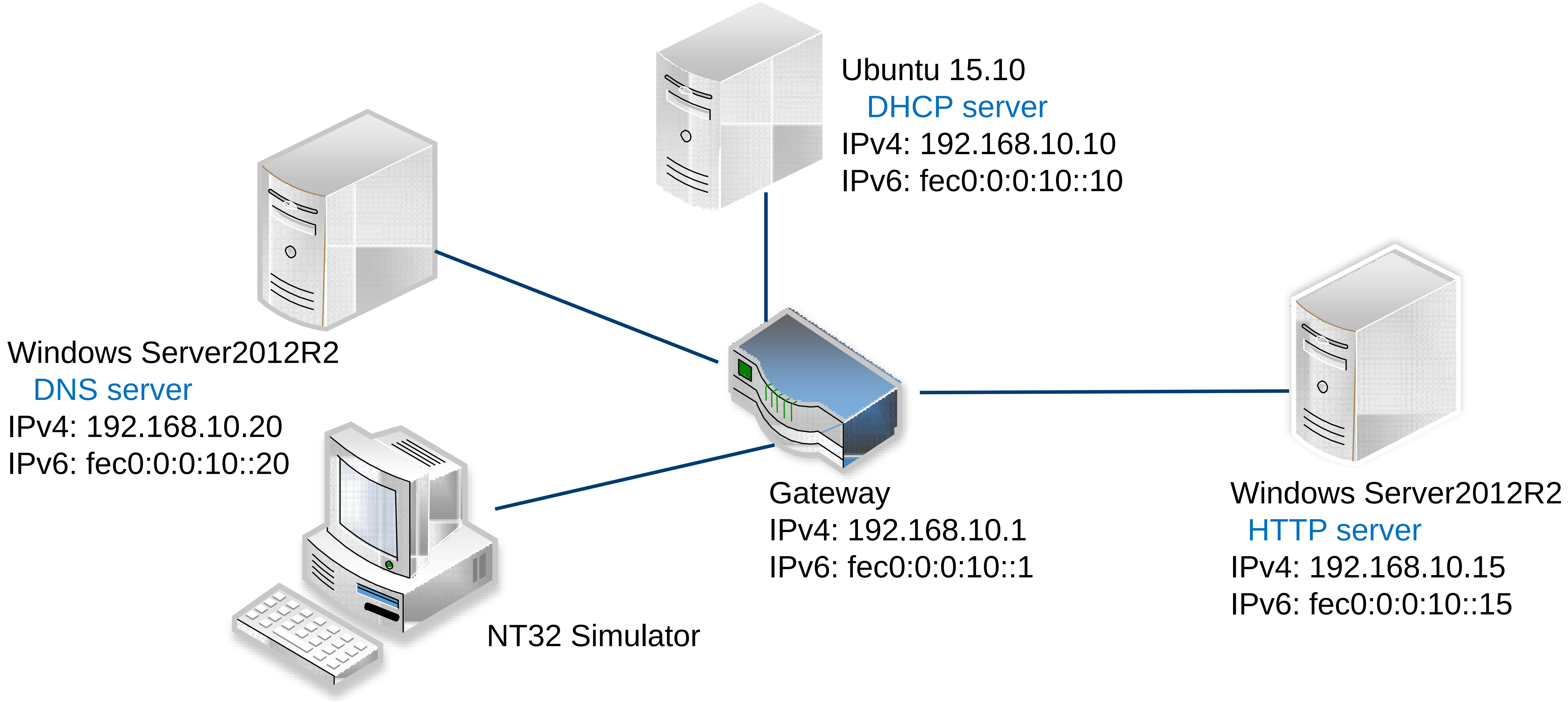
- UEFI 2.5 defined RAM Disk device path nodes
  - Standard access to a RAM Disk in UEFI
  - Supports Virtual Disk and Virtual CD (ISO image) in persistent or volatile memory
  - Device Path: Type:4 Subtype: 9
- ACPI 6.0 NVDIMM Firmware Interface Table (NFIT)
  - Describe the RAM Disks to the OS
  - Runtime access of the ISO boot image in memory
- Supported Image Types
  - *.ISO Virtual CD Image
  - *.img Virtual Disk Image
  - *.efi UEFI Executable Image



Feature Enabling:

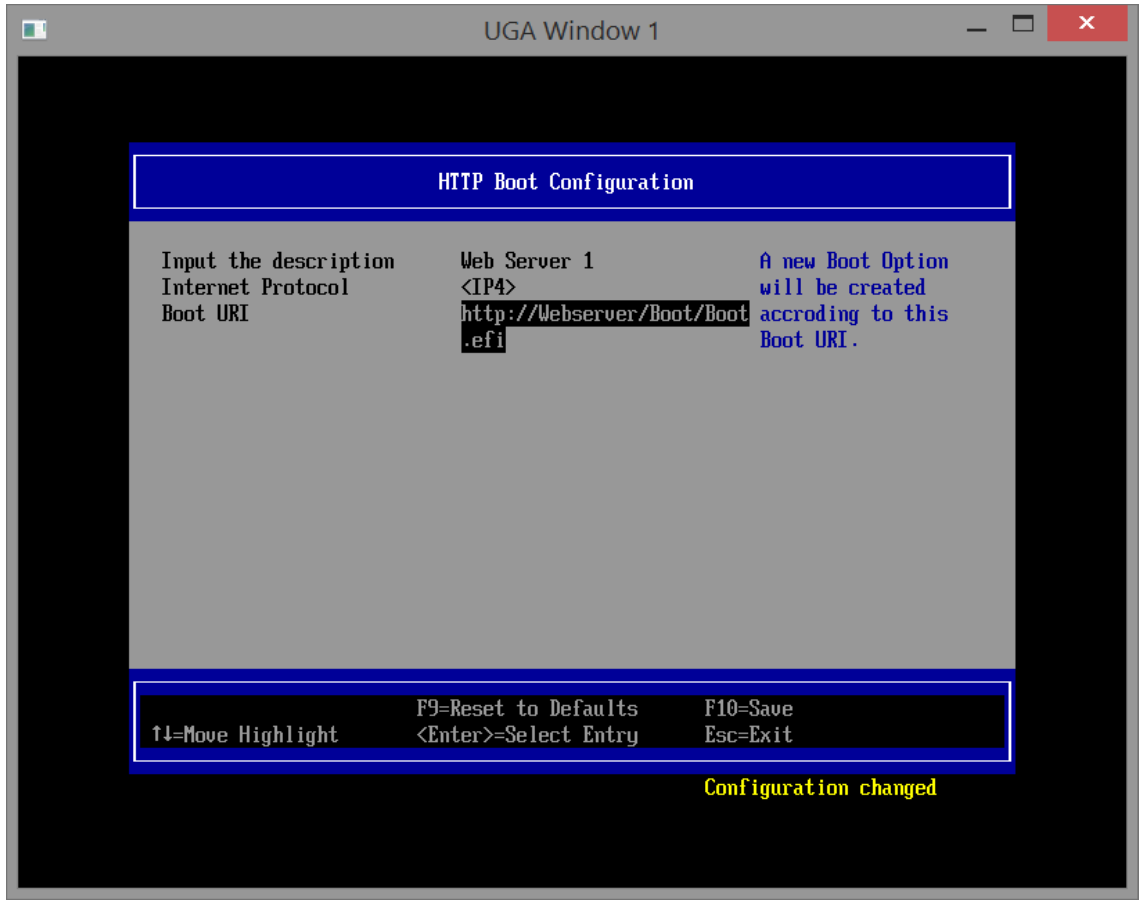
Add Edk2 `RamDiskDxe.inf` to Platform .DSC

UEFI Http Boot Example



Public paper EDK II HTTP Boot Getting Started Guide for a step-by-step guide of the HTTP Boot enabling and server configuration in **corporate environment**.

EDK II HTTP Boot Configuration

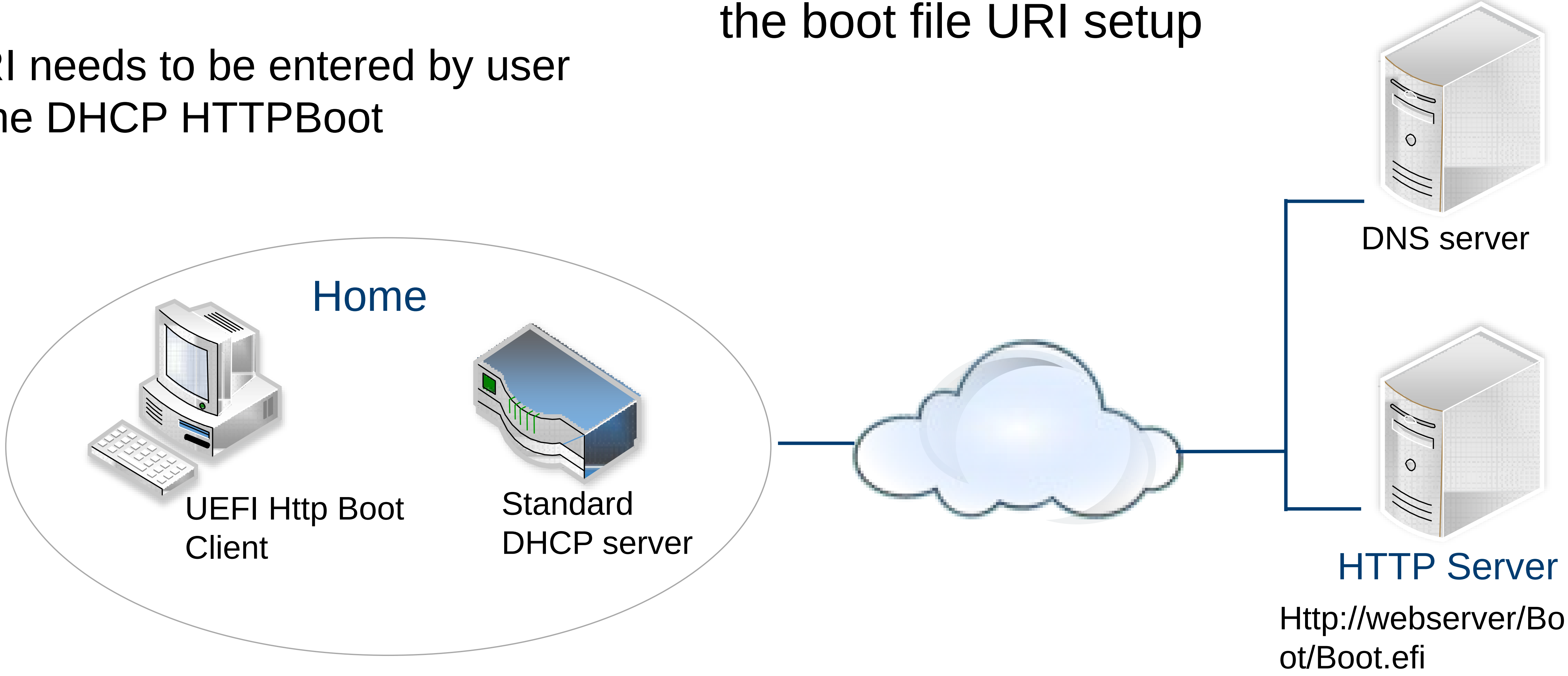


In the main page of Boot Manager Menu, enter [Device Manager] -> [Network Device List] -> Select a NIC device -> [HTTP Boot Configuration], set the HTTP boot parameters such as the boot option title, IP start version and the URI address

Save the configuration and back to the main page, enter [Boot Manager] menu and select the new created boot option to start the HTTP Boot

UEFI Native HTTP Boot – Home Environment

- Only a Standard DHCP server is available for host IP configuration assignment
- Boot file URI needs to be entered by user instead of the DHCP HTTPBoot extensions.
- The EDK II HTTP Boot Driver provides a configuration pages for the boot file URI setup



Getting Started Guides

HTTP:

Wiki Page https://github.com/tianocore/tianocore.github.io/wiki/HTTP-Boot

PDF HttpBootGettingStartedGuide_0_9.pdf

HTTPS:

Wiki Page https://github.com/tianocore/tianocore.github.io/wiki/HTTPs-Boot

PDF Getting Started with UEFI HTTPS Boot on EDK II .pdf

tianocore

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

www.tianocore.org

tianocore

ACKNOWLEDGEMENTS

Network Stack IPv4 and/or IPv6