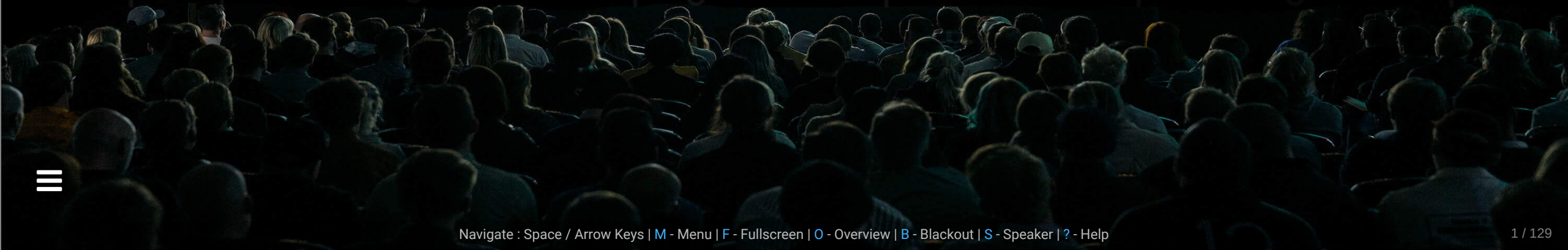
tianocore

UEFI & EDK II TRAINING

UEFI NEWTWORK IN EDK II

tianocore.org



tianocore

LESSON OBJECTIVE

- UEFI Network Stack Layers
  host and target basic configuration and components
- EDK II Network Features Overview
- What UEFI Protocols Make Network Work in EDK II
- UEFI HTTP(s) Boot Overview

tianocore

THE NETWORK STACK

Industry Standard on Networking

tianocore

Industry Standard – Network Stack Layers

tianocore

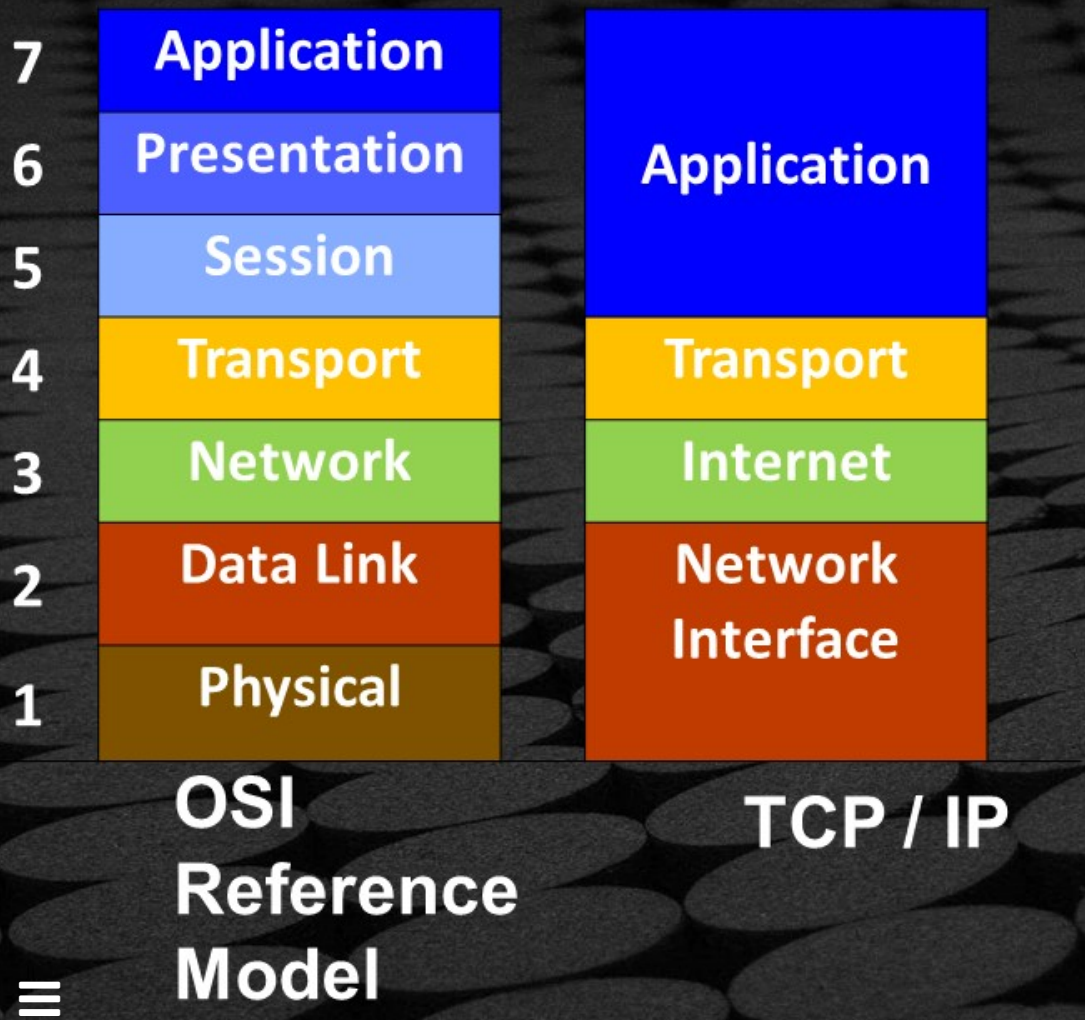
Industry Standard – Network Stack Layers

	OSI Reference Model	TCP / IP
7	Application	Application
6	Presentation	
5	Session	
4	Transport	Transport
3	Network	Internet
2	Data Link	Network Interface
1	Physical	

≡

tianocore

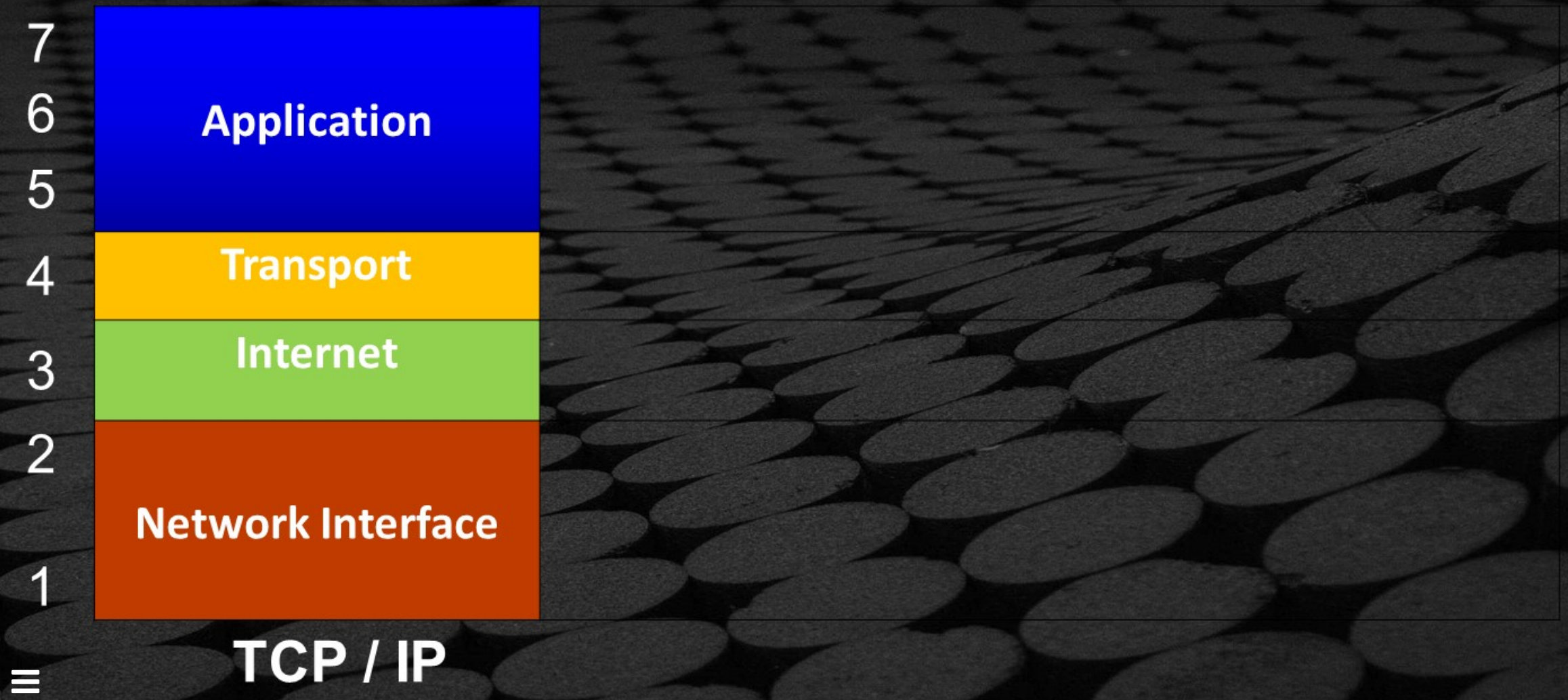
Industry Standard – Network Stack Layers



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.

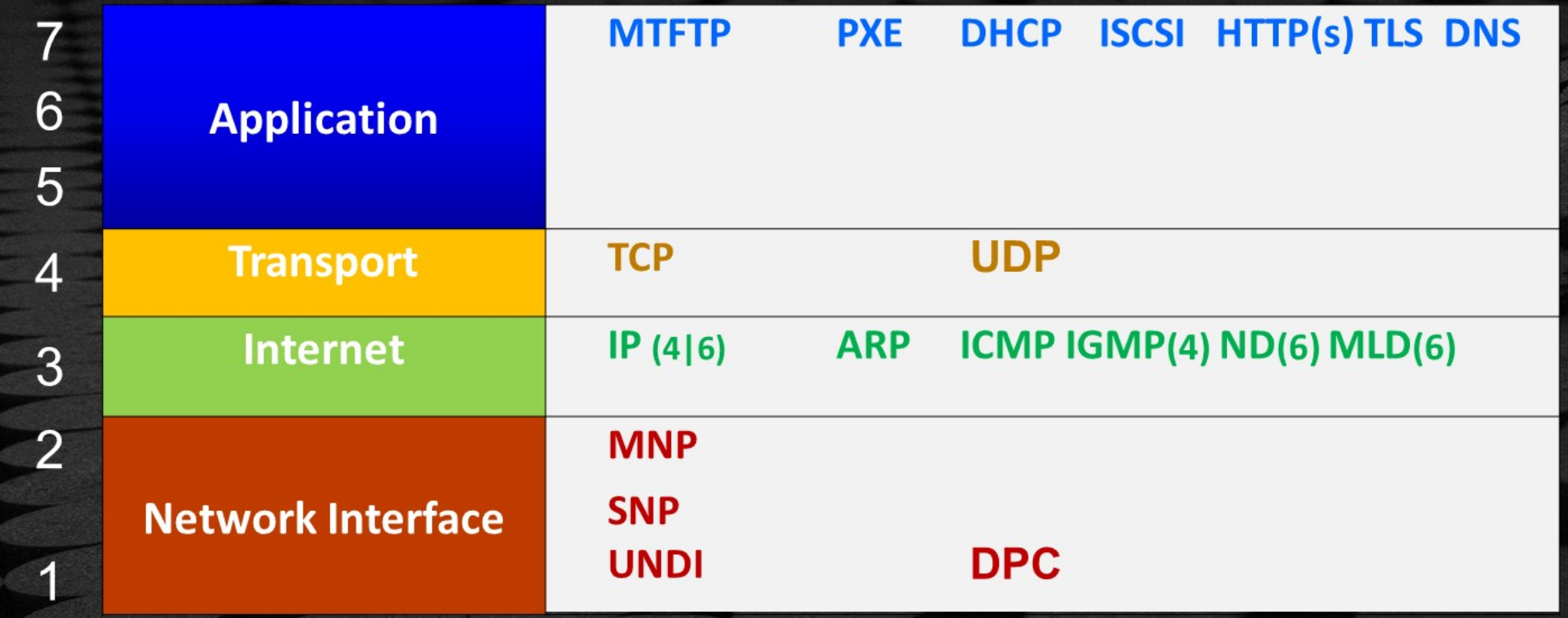
tianocore

TCP / IP Network Stack for UEFI



tianocore

TCP / IP Network Stack for UEFI



tianocore

Network Acronyms

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

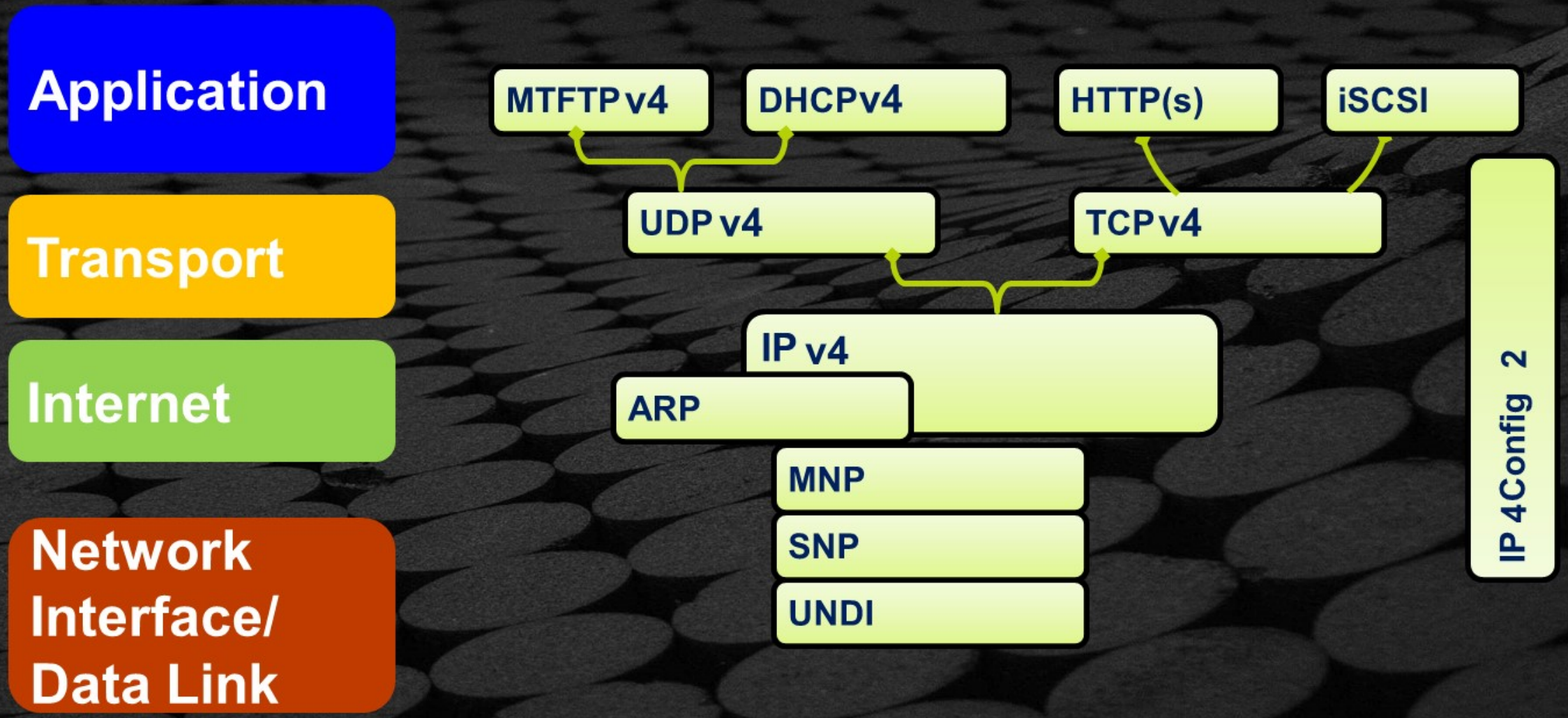
Internet

UDP - User Datagram Protocol
IP - Internet Protocol

Network Interface/ Data Link

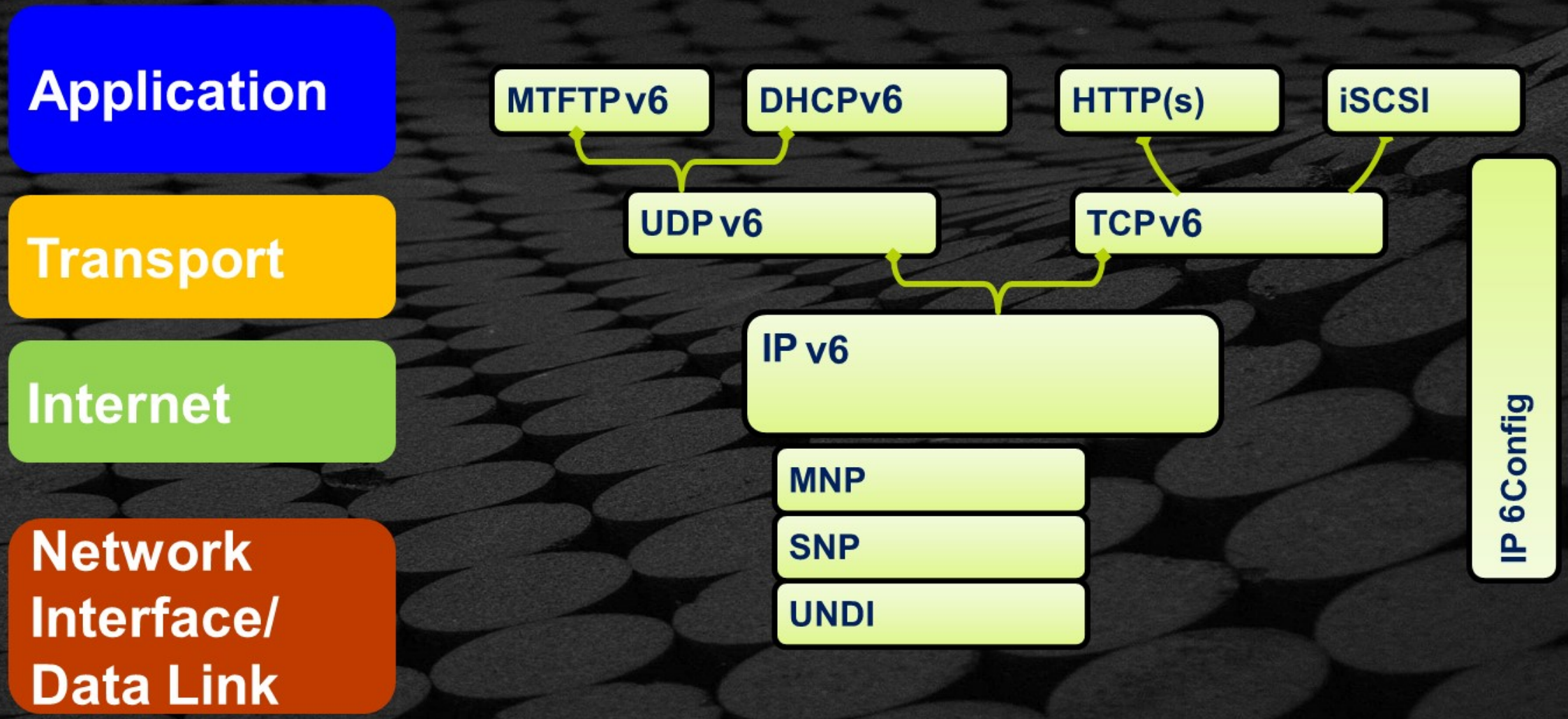
ARP - Address Resolution Protocol
MNP – Managed Network Protocol
SNP – Simple Network Protocol
UNDI - Universal Network Device Interface
DPC - Deferred Procedure Call

Network Stack IPv4



tianocore

Network Stack IPv6



Network Stack IPv4 or IPv6

MNP

SNP

UNDI

tianocore

Network Stack IPv4 or IPv6

MNP

SNP

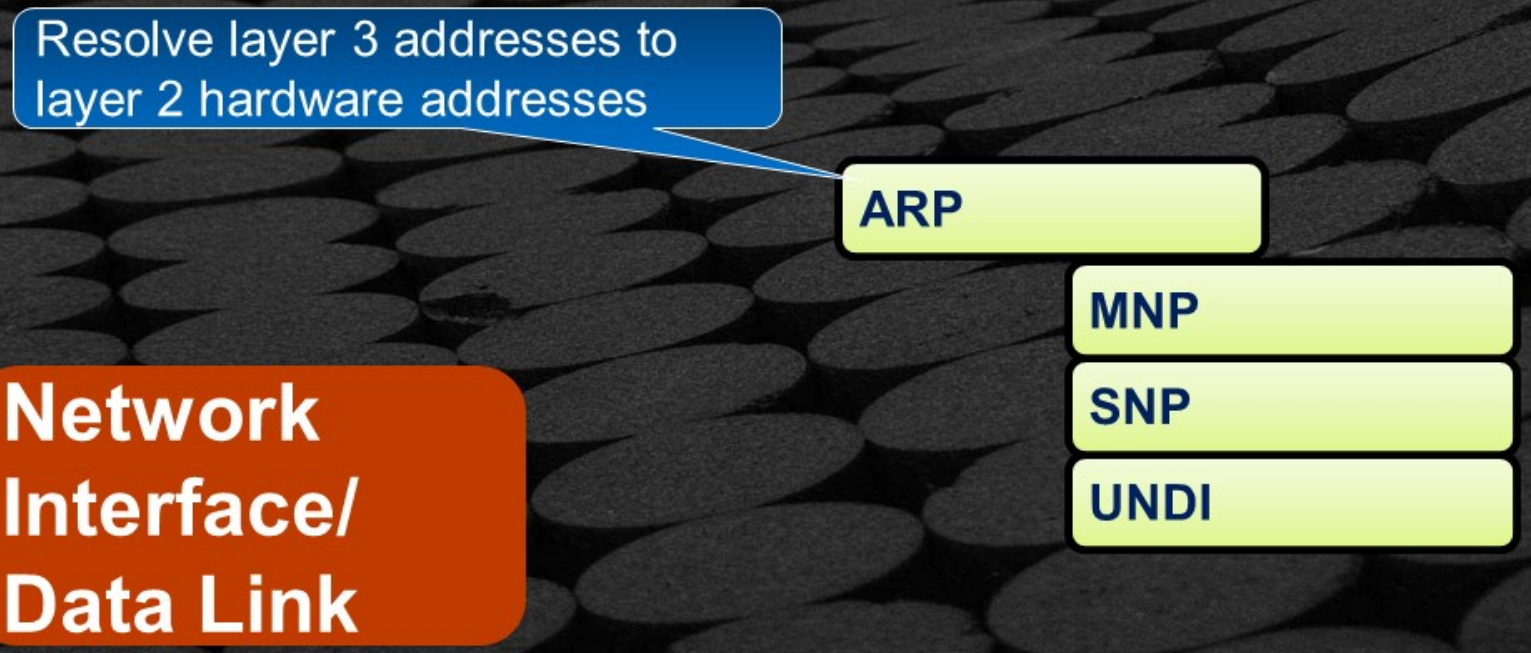
UNDI

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

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

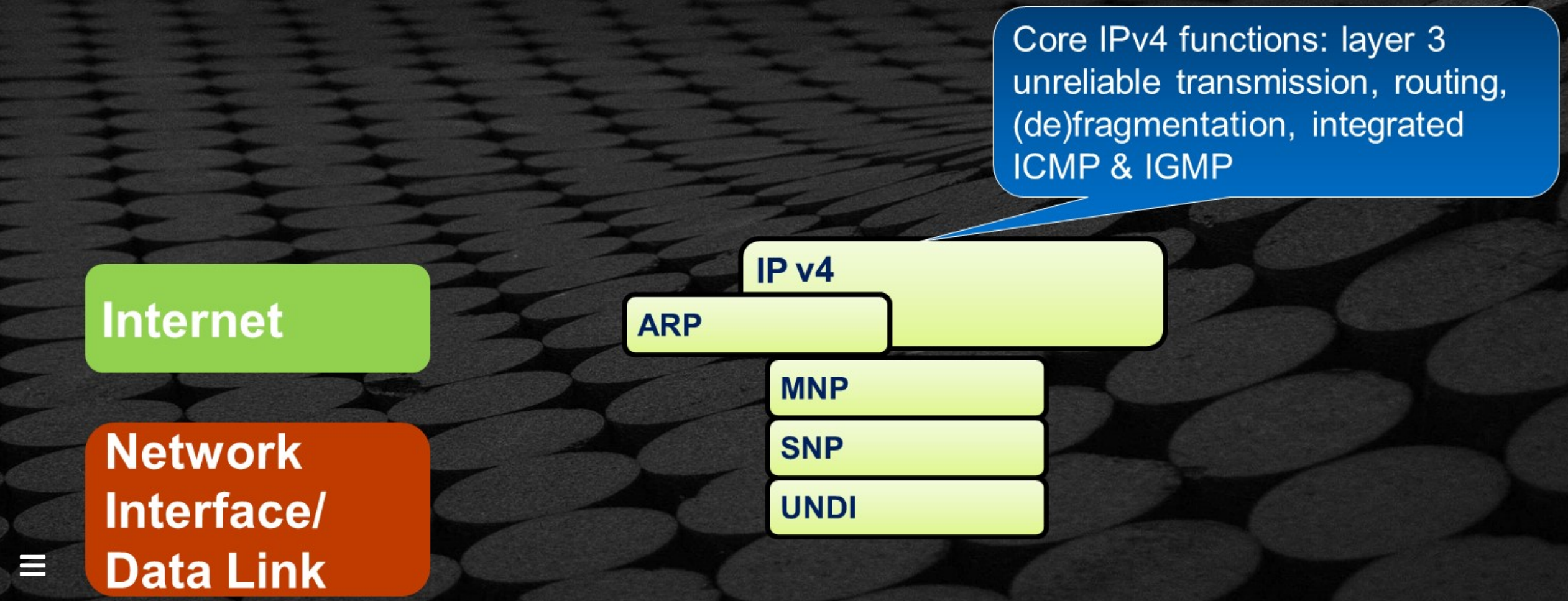
Network Interface Card Device Driver – Provide interfaces to operate the NIC.

Network Stack IPv4 or IPv6



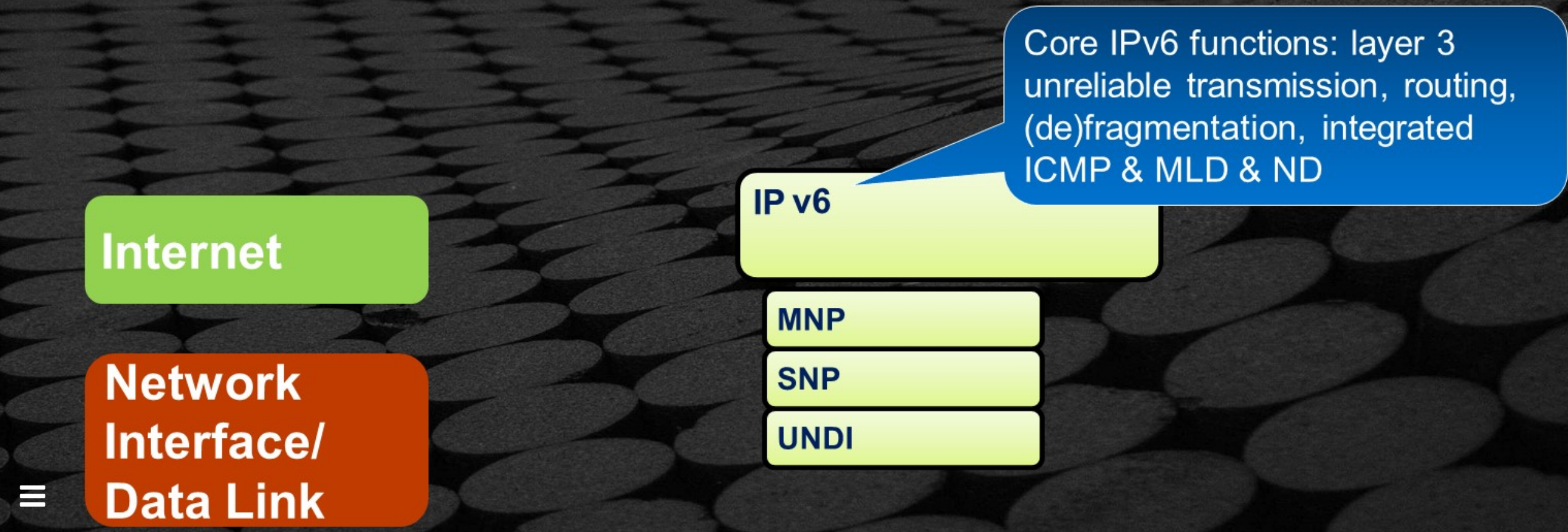
tianocore

Network Stack IPv4 or IPv6



tianocore

Network Stack IPv4 or IPv6



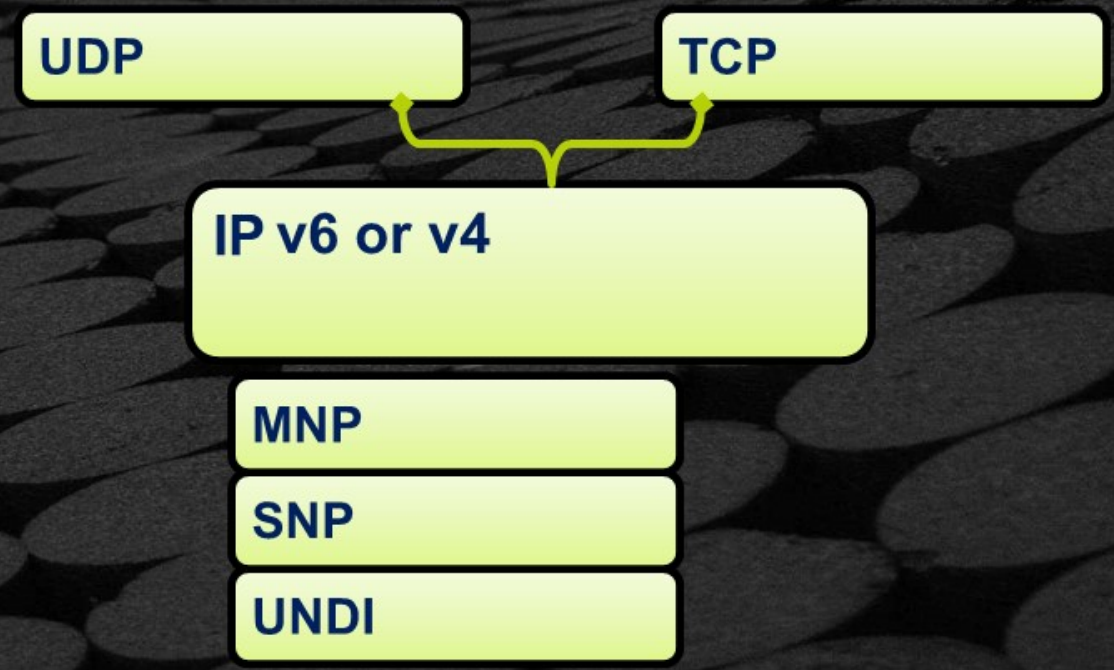
tianocore

Network Stack IPv4 or IPv6

Transport

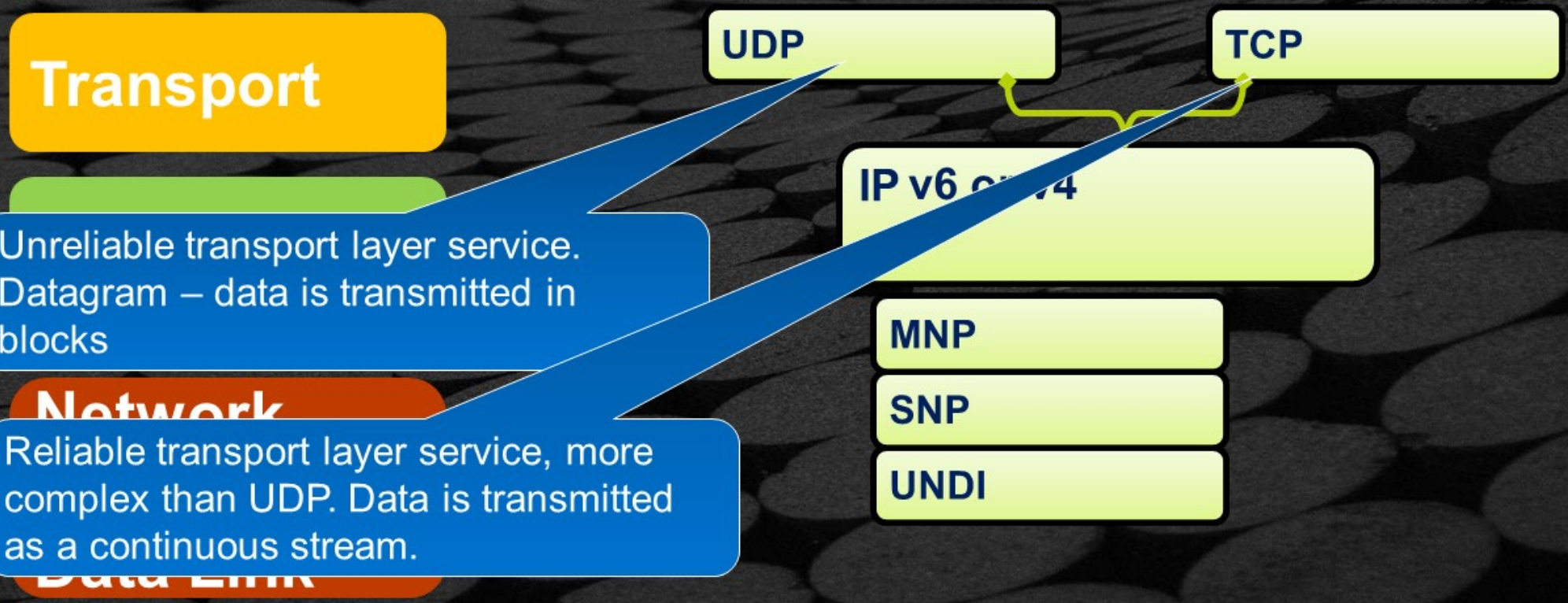
Internet

Network Interface/ Data Link



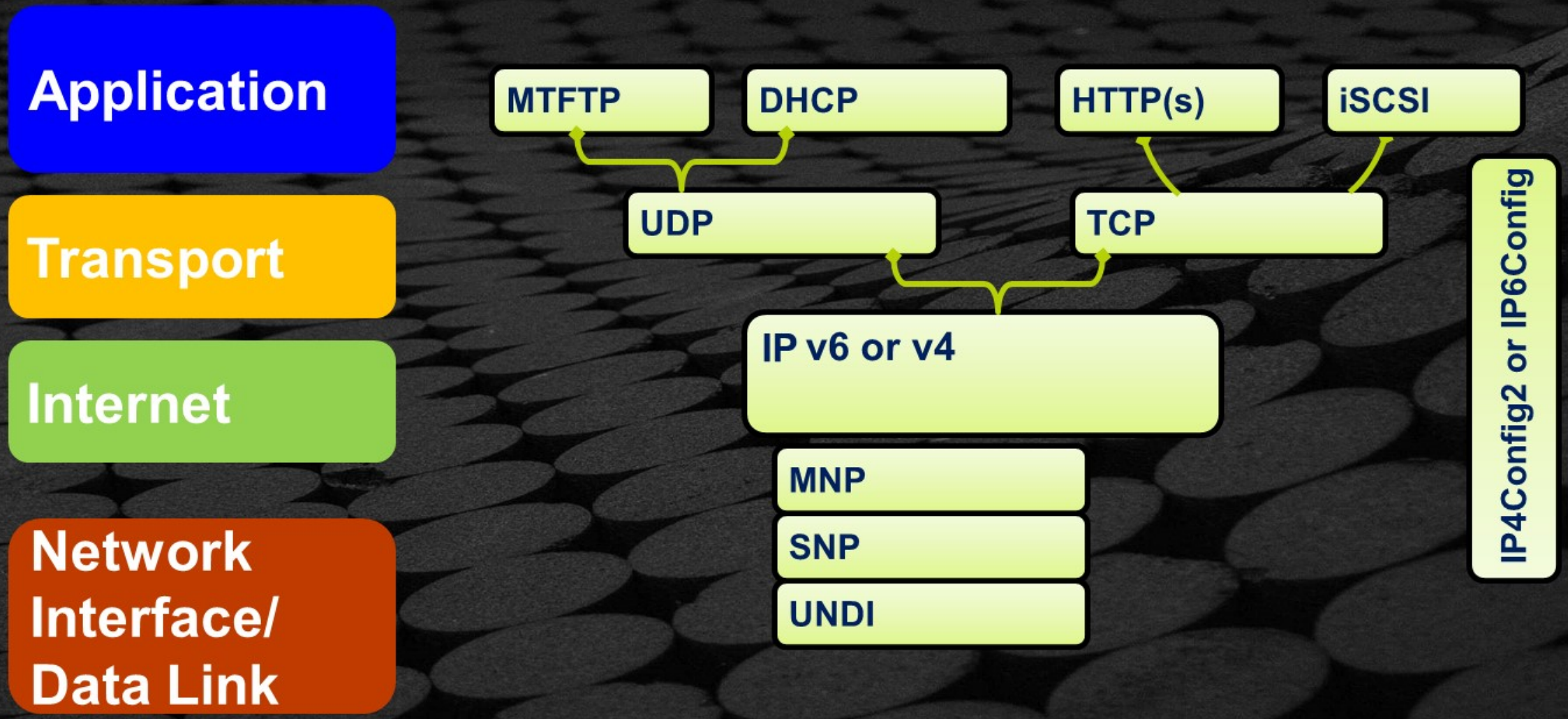
tianocore

Network Stack IPv4 or IPv6

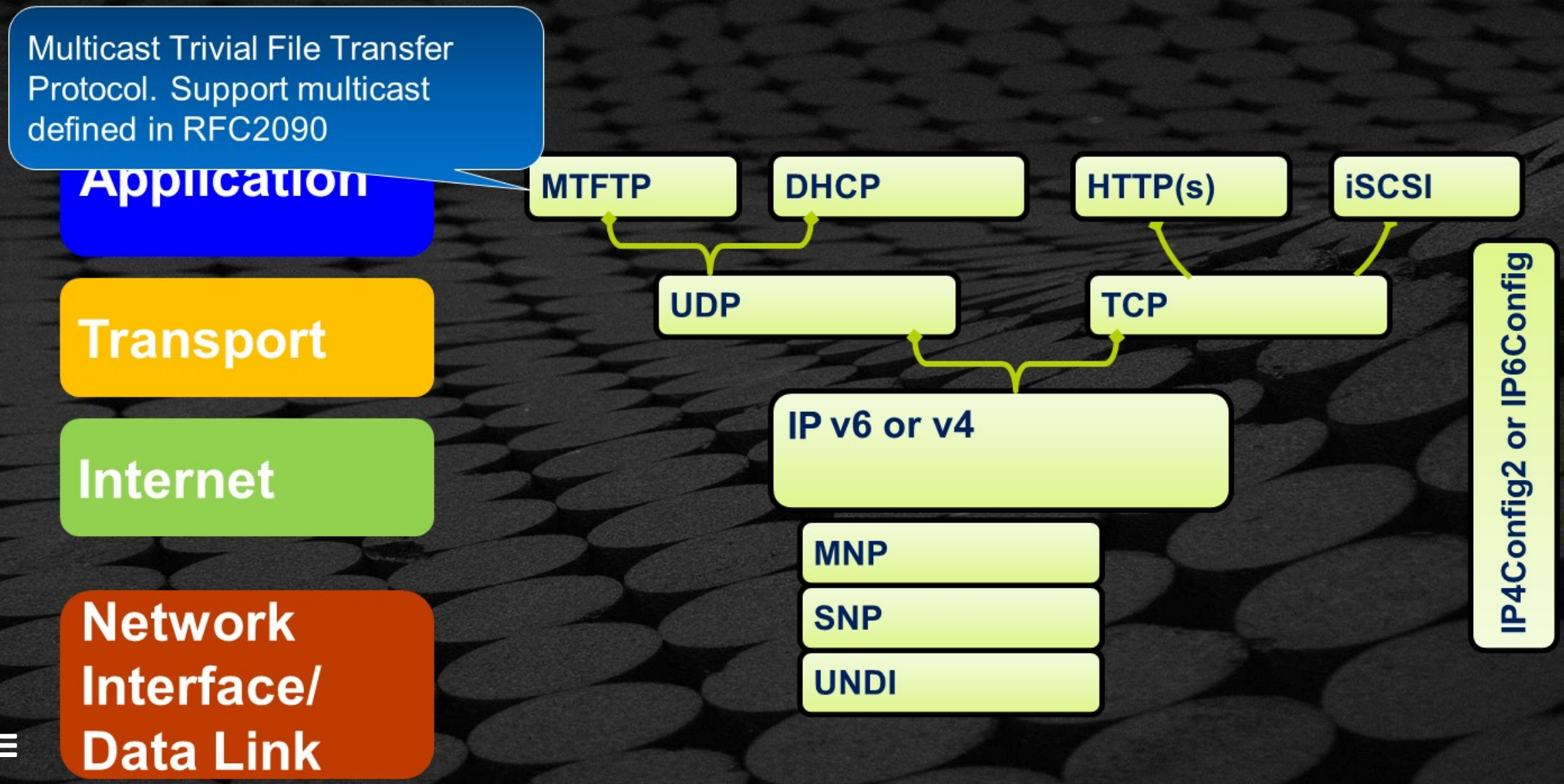


tianocore

Network Stack IPv4 or IPv6

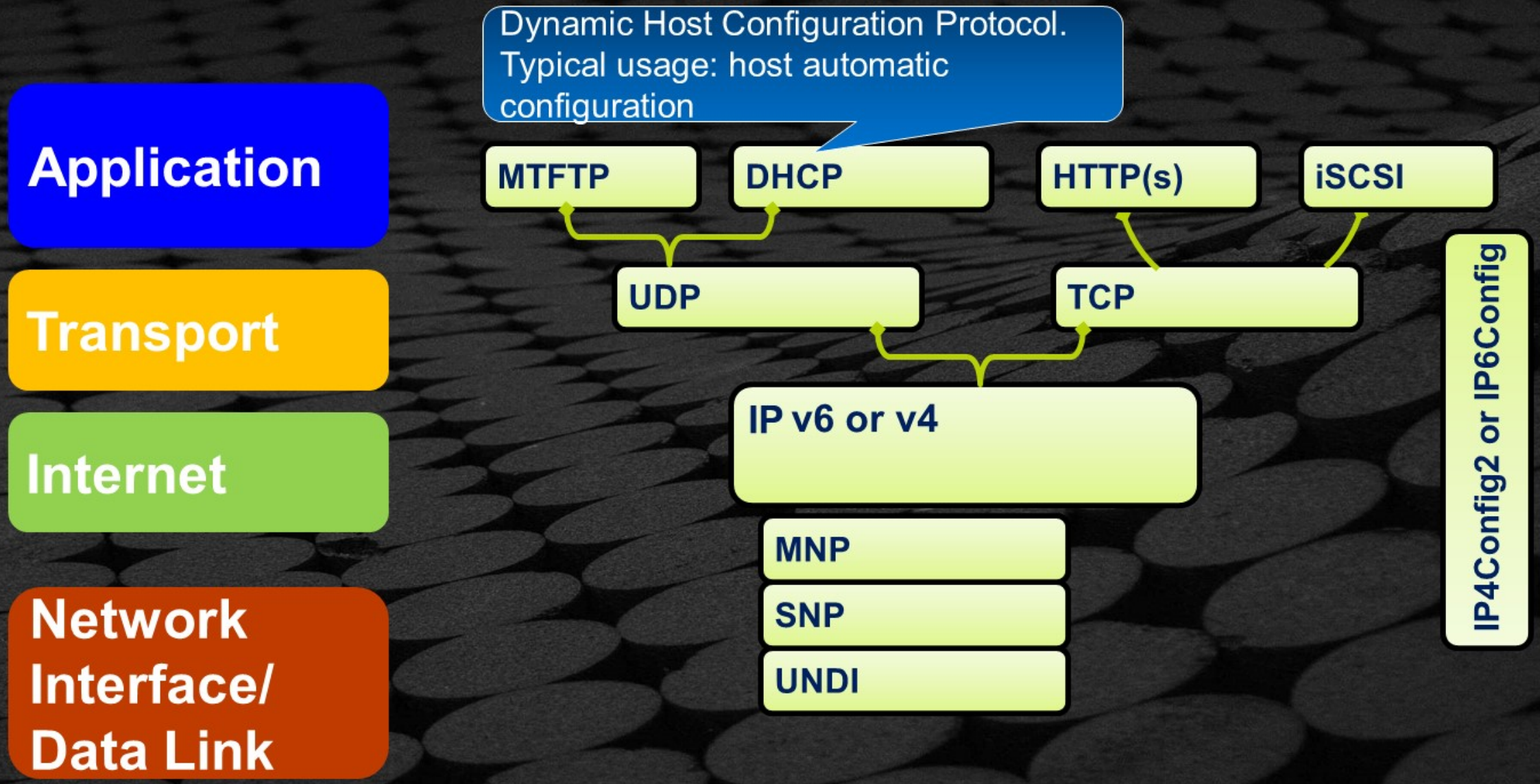


Network Stack IPv4 or IPv6



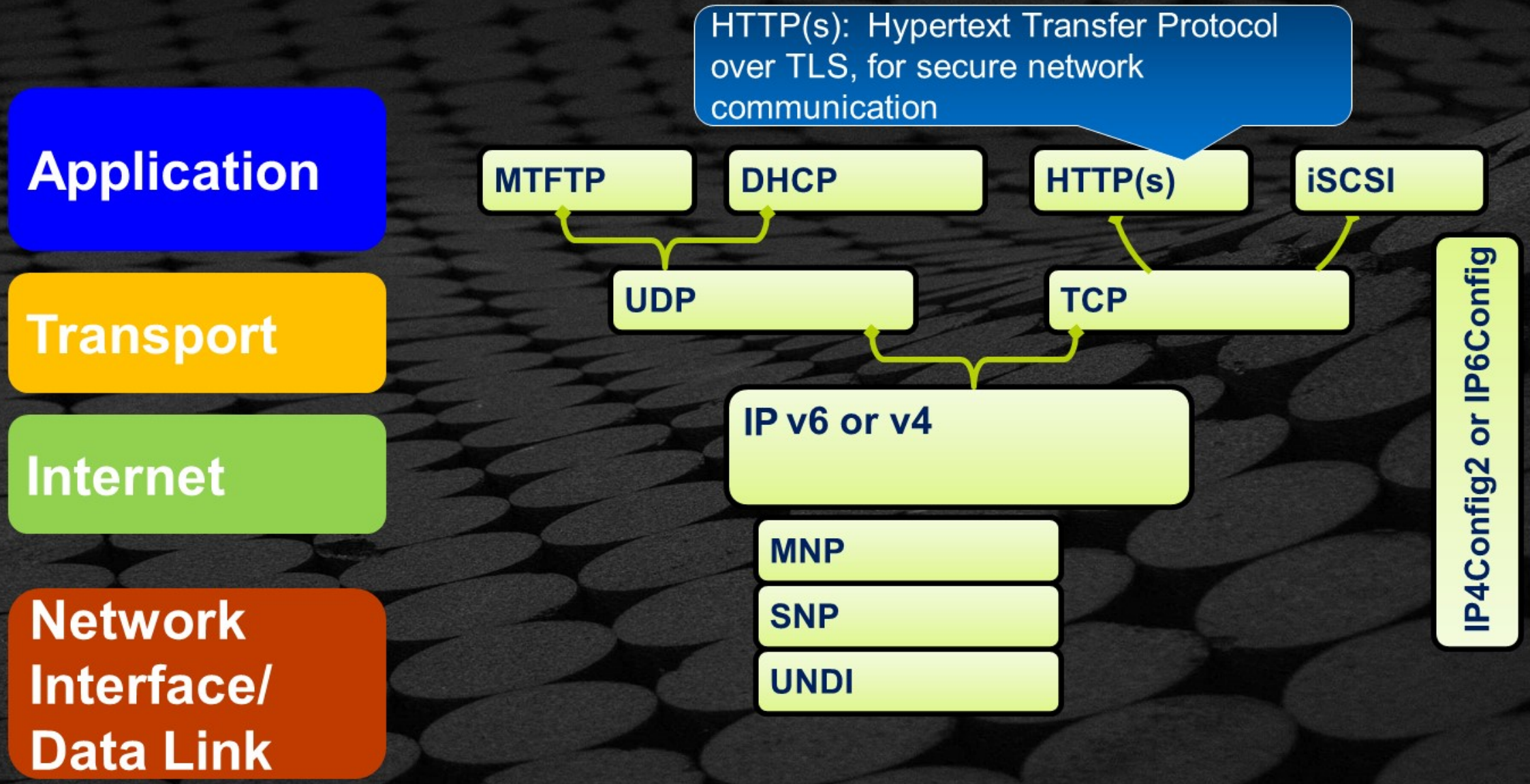
tianocore

Network Stack IPv4 or IPv6



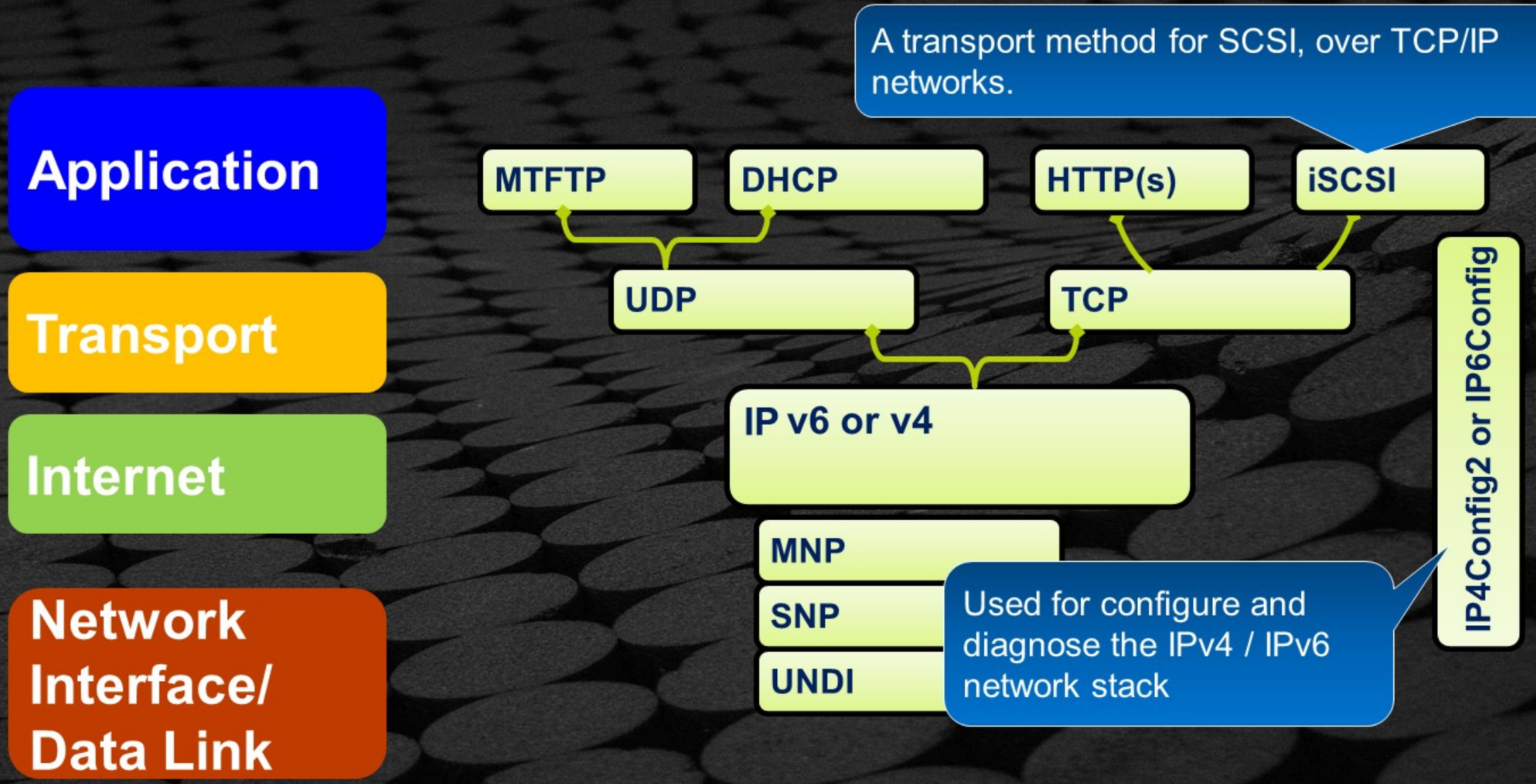
tianocore

Network Stack IPv4 or IPv6



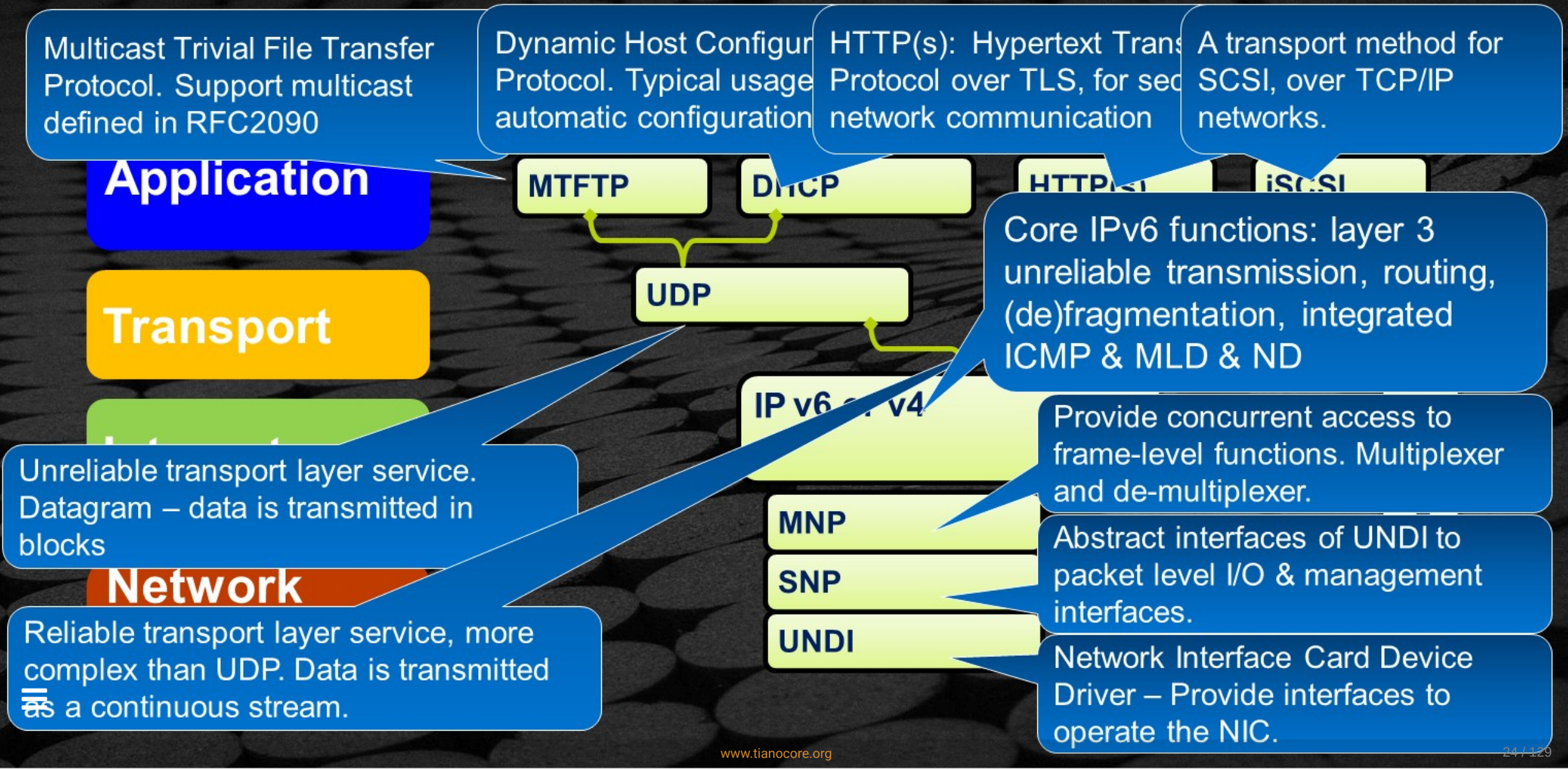
tianocore

Network Stack IPv4 or IPv6



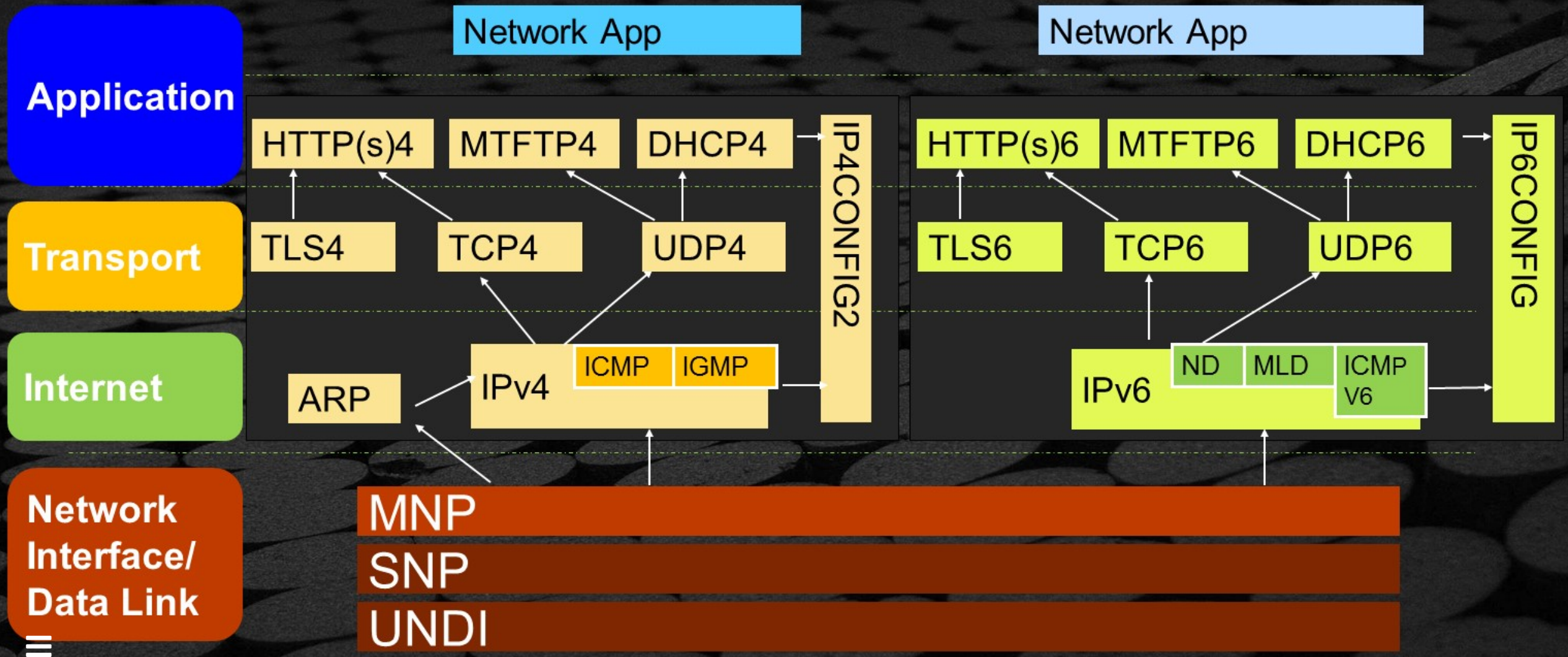
tianocore

Network Stack IPv4 or IPv6



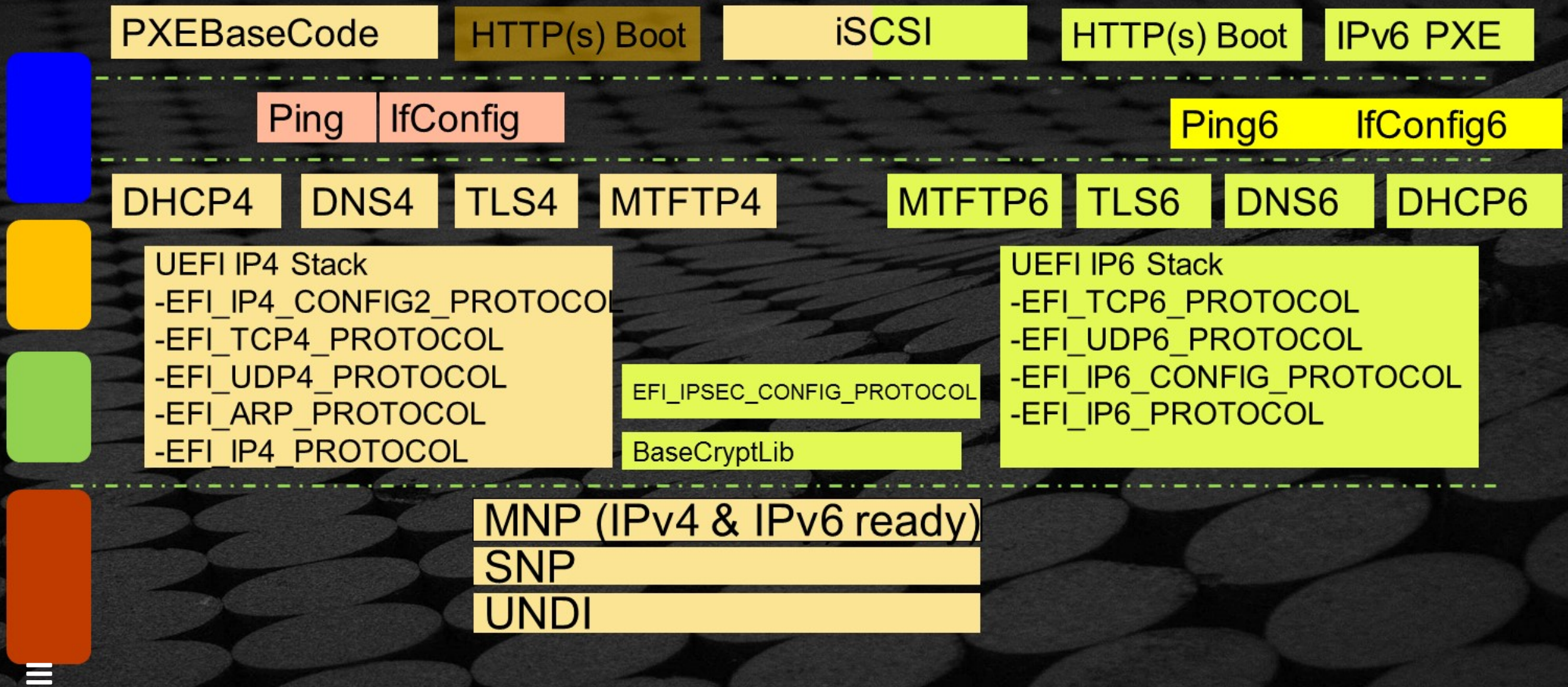
tianocore

UEFI Network Stack: IPv4 Vs. IPv6



tianocore

UEFI Network Stack with Protocols



UEFI Network Stack with Protocols

iSCSI

HTTP(s) Boot

IPv6 PXE

Ping6

IfConfig6

MTFTP6

TLS6

DNS6

DHCP6

UEFI IP6 Stack
-EFI_TCP6_PROTOCOL
-EFI_UDP6_PROTOCOL
-EFI_IP6_CONFIG_PROTOCOL
-EFI_IP6_PROTOCOL

EFI_IPSEC_CONFIG_PROTOCOL

BaseCryptLib

MNP (IPv6 ready)

SNP

UNDI

tianocore

UEFI Network Stack with Protocols

iSCSI

HTTP(s) Boot

IPv6 PXE

Ping6

IfConfig6

MTFTP6

TLS6

DNS6

DHCP6

UEFI IP6 Stack
-EFI_TCP6_PROTOCOL
-EFI_UDP6_PROTOCOL
-EFI_IP6_CONFIG_PROTOCOL
-EFI_IP6_PROTOCOL

EFI_IPSEC_CONFIG_PROTOCOL

BaseCryptLib

MNP (IPv6 ready)

SNP

UNDI

UEFI 2.5 added HTTP(s) Boot

tianocore

UEFI Network Stack Live Picture

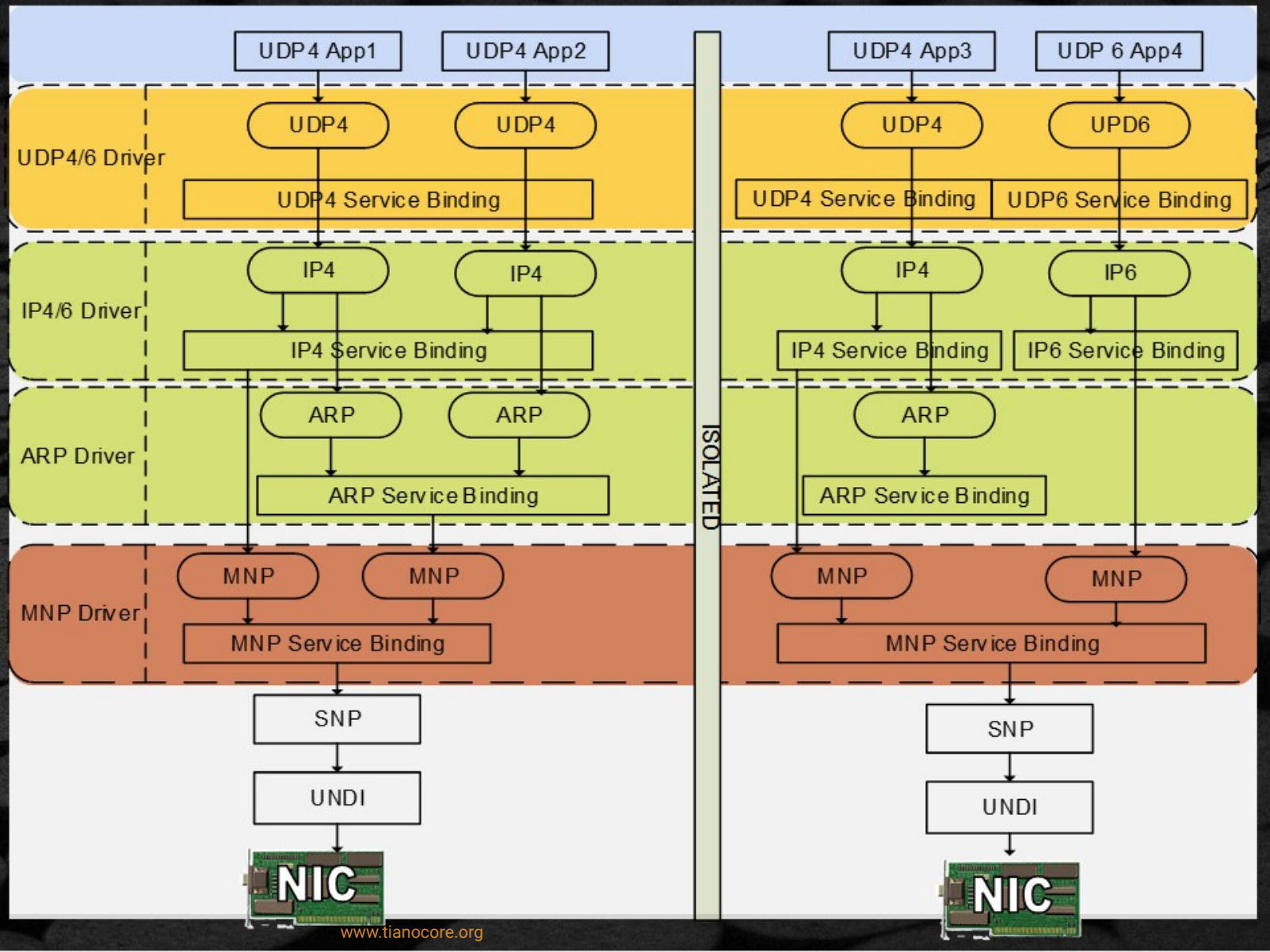
Application

Transport

Internet

Network Interface/ Data Link

≡



tianocore

EDK II NETWORK

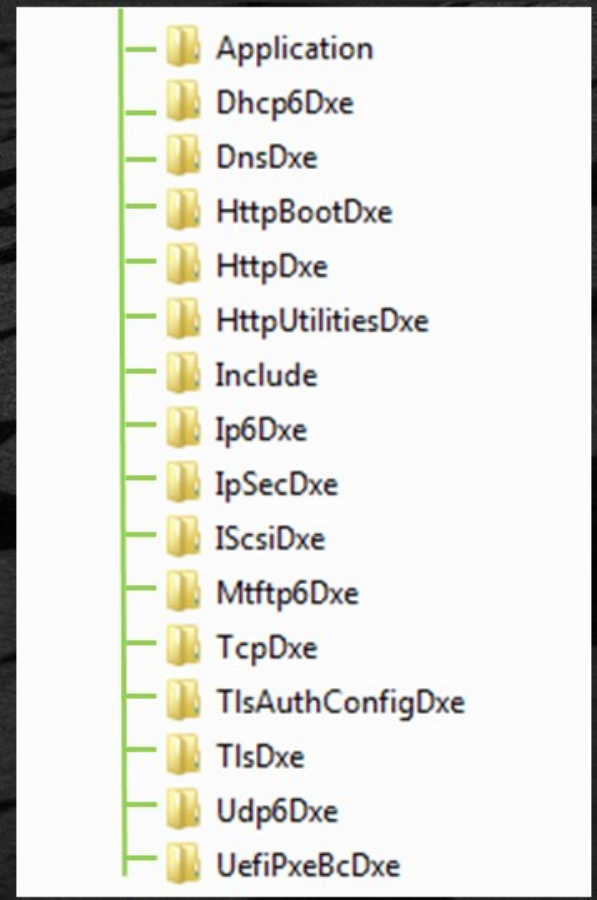
EDK II Network Features Overview

Where is the EDK II Network Stack located?

github.com/tianocore/edk2/

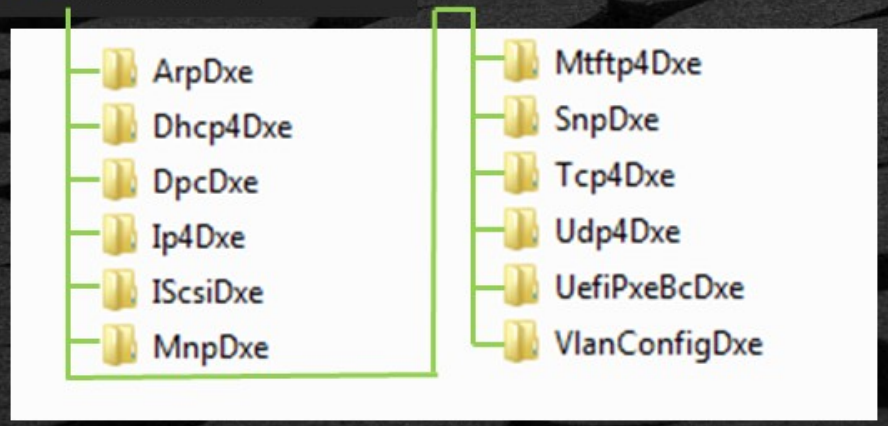
IP6:

NetworkPkg



IP4:

MdeModulePkg
└ Universal
└ Network



Network Related Libraries

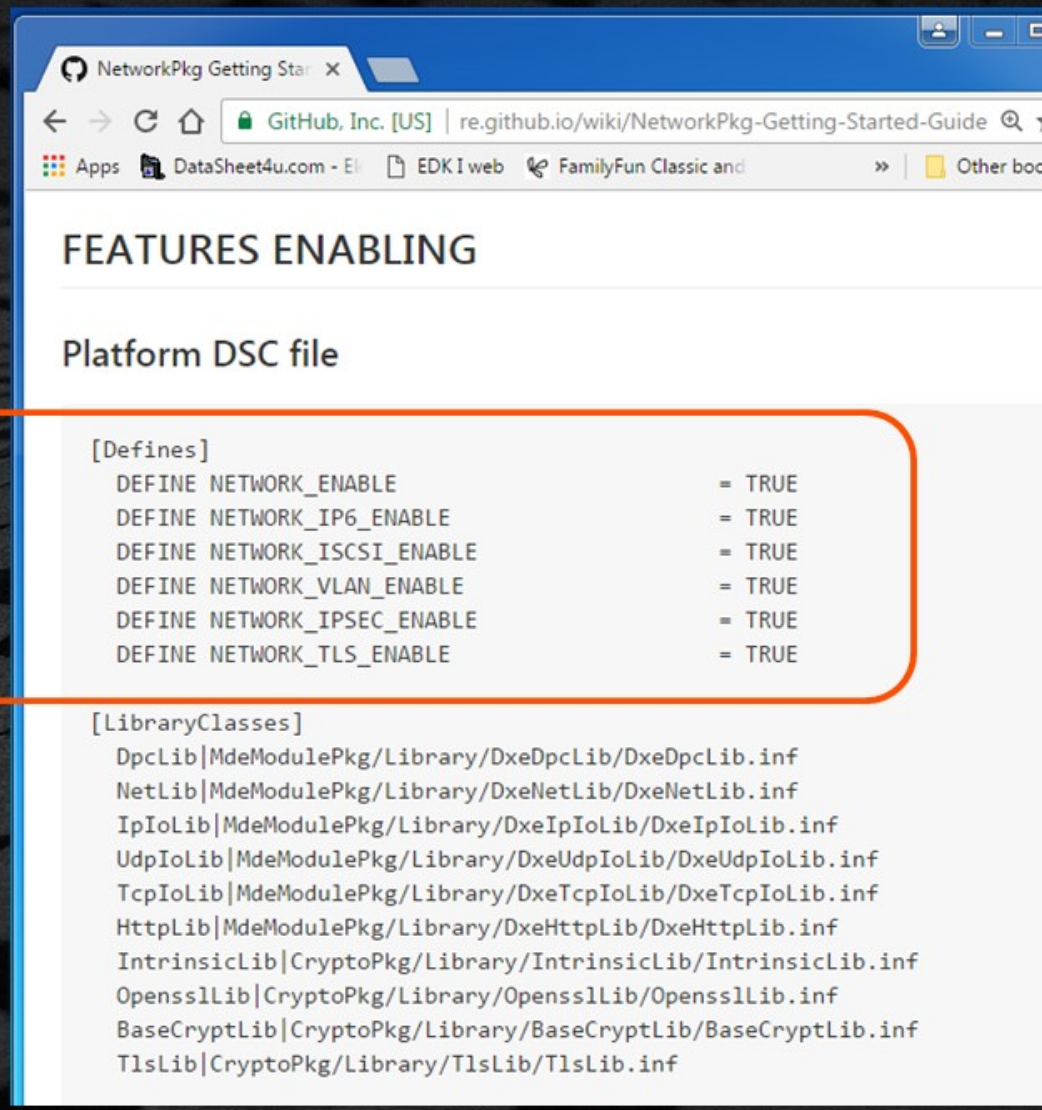
MdeModulePkg
└Library

How to Enable the EDK II Network Stack

Update the Platform DSC and FDF files:

Wiki link: Featues enabling.

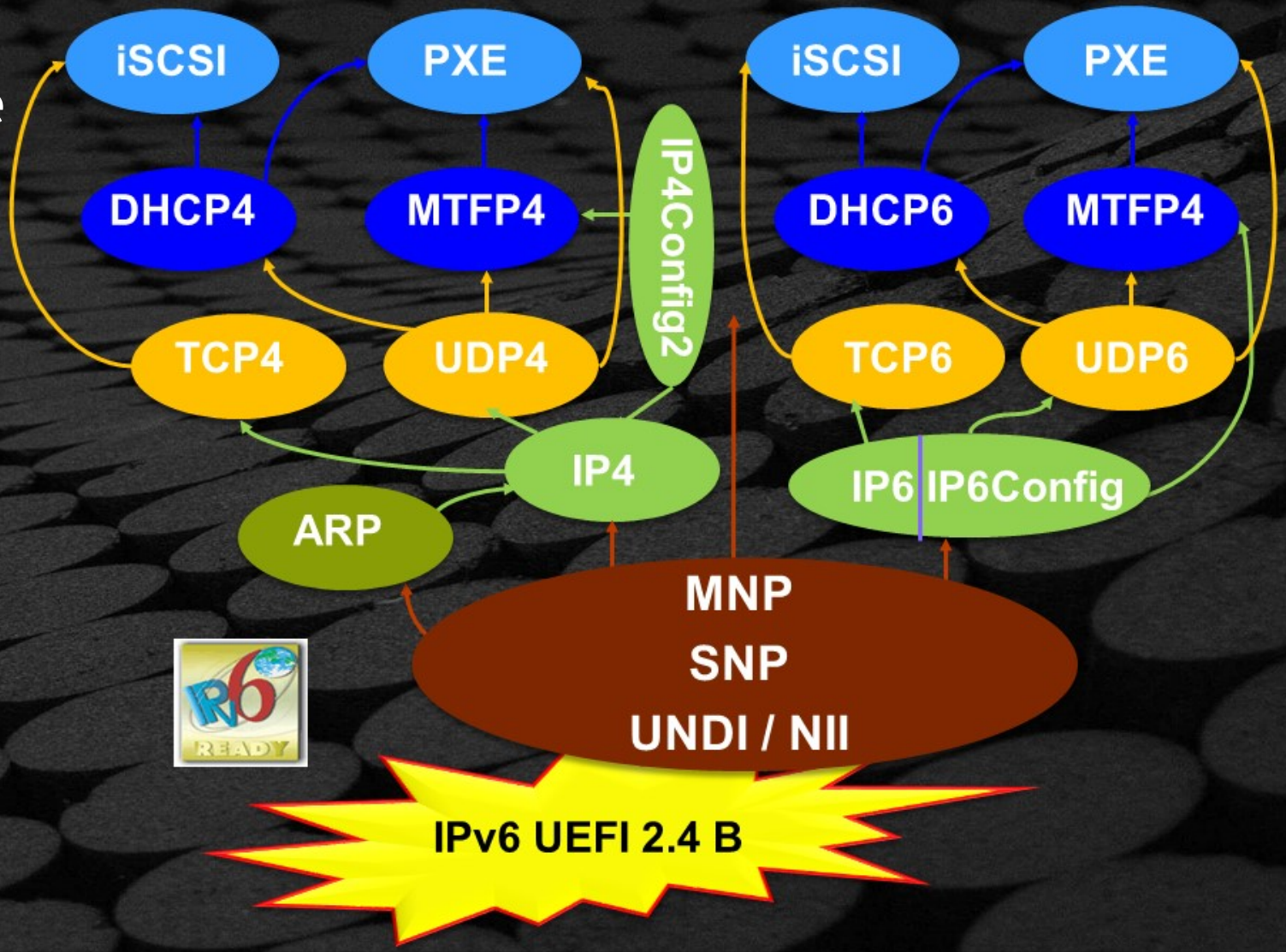
```
DEFINE NETWORK_ENABLE   = TRUE
DEFINE NETWORK_IP6_ENABLE = TRUE
// . . .
```



IP6 Networking - Vendors

IPv6 protocols compliance

- IPv6 ready logo approved ipv6ready.org
- Requirements for IPv6 transition usgv6-v1.pdf
- Vendor Testing: faq-c.html#vendors



tianocore

Virtual LAN - VLAN

Logical groups of Stations at the data link layer (Tagging)

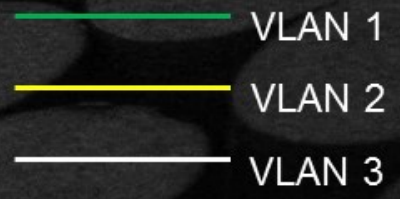
- VLAN's allow a network manager to logically segment a LAN into different broadcast domains Link

Why VLAN?

- Performance
- Security
- Formation of Virtual Workgroups
- Simplified Administration
- Cost

VLAN Standard: IEEE 802.1Q

UEFI Host

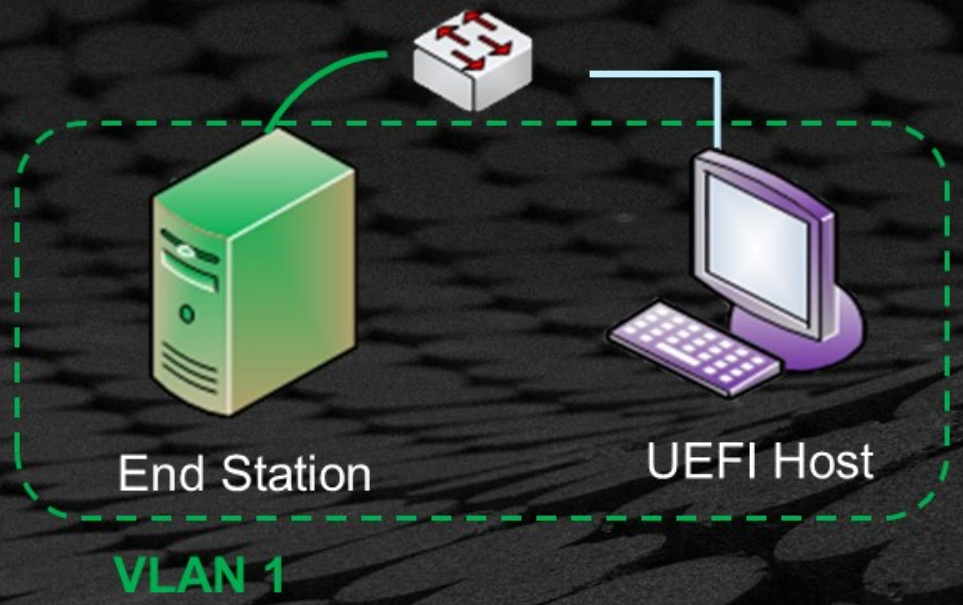


tianocore

Virtual LAN - VLAN

Logical groups of Stations at the data link layer (Tagging)

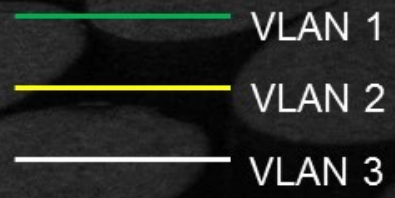
- VLAN's allow a network manager to logically segment a LAN into different broadcast domains Link



Why VLAN?

- Performance
- Security
- Formation of Virtual Workgroups
- Simplified Administration
- Cost

VLAN Standard: IEEE 802.1Q



tianocore

Virtual LAN - VLAN

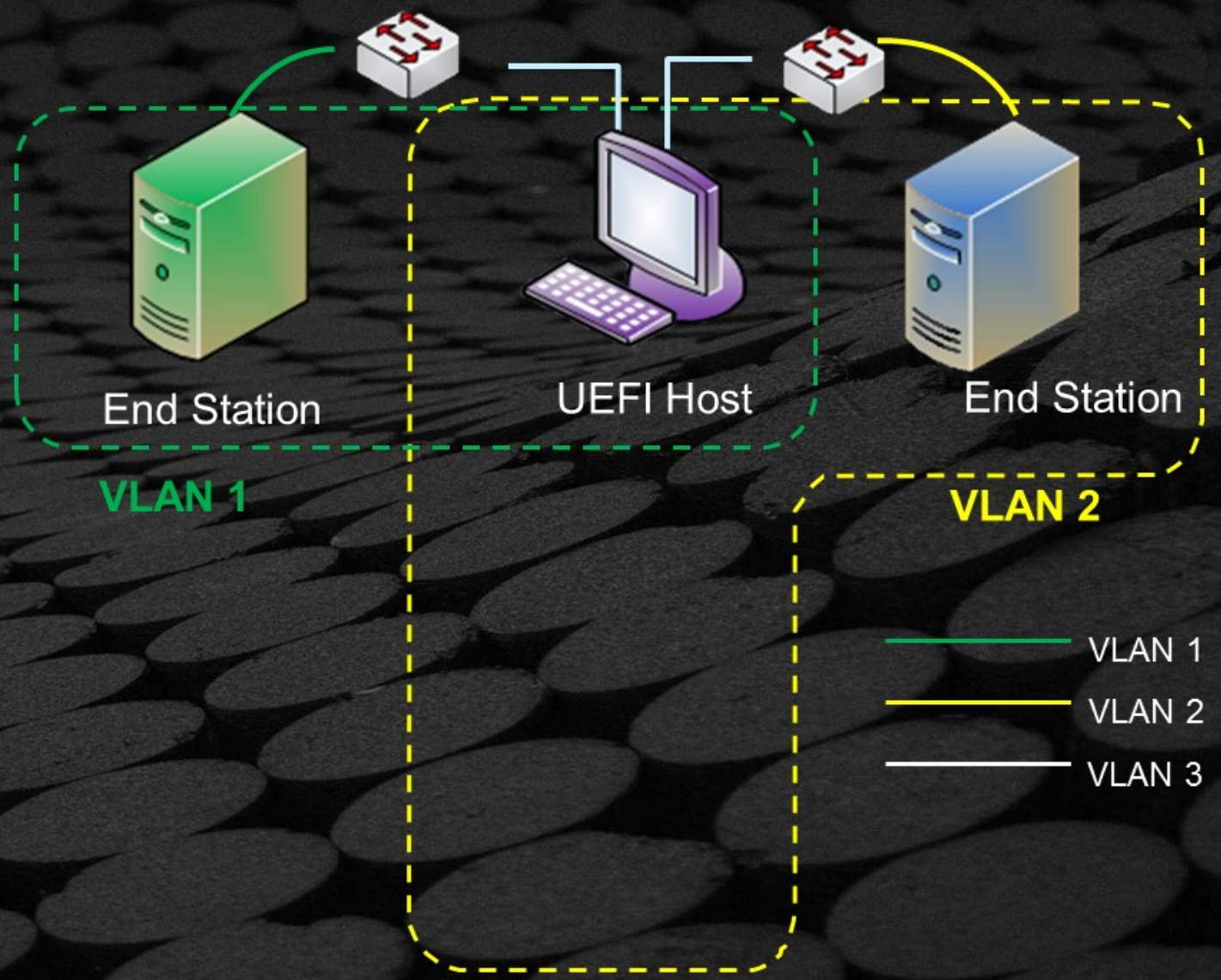
Logical groups of Stations at the data link layer (Tagging)

- 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



tianocore

Virtual LAN - VLAN

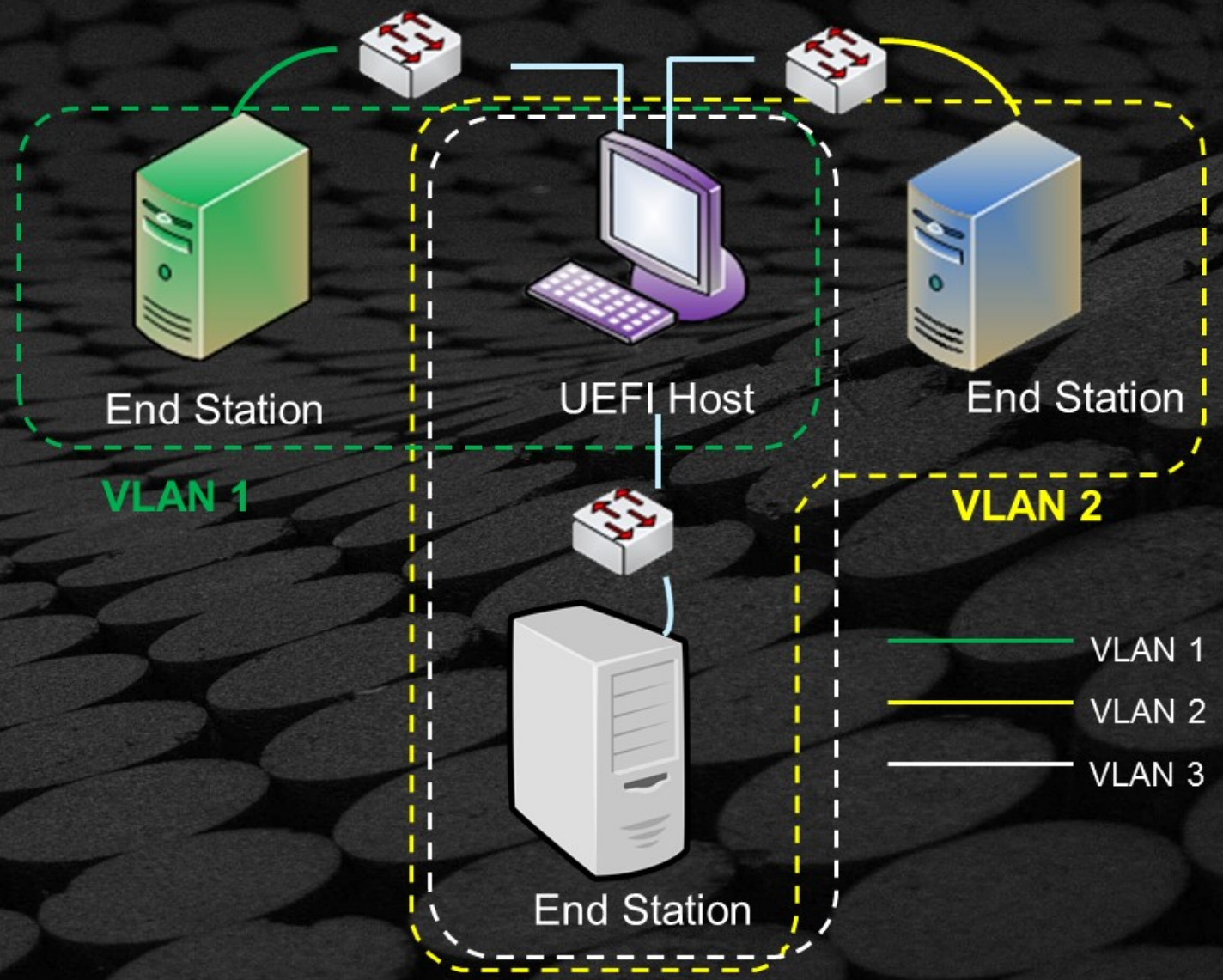
Logical groups of Stations at the data link layer (Tagging)

- 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



Virtual LAN - VLAN

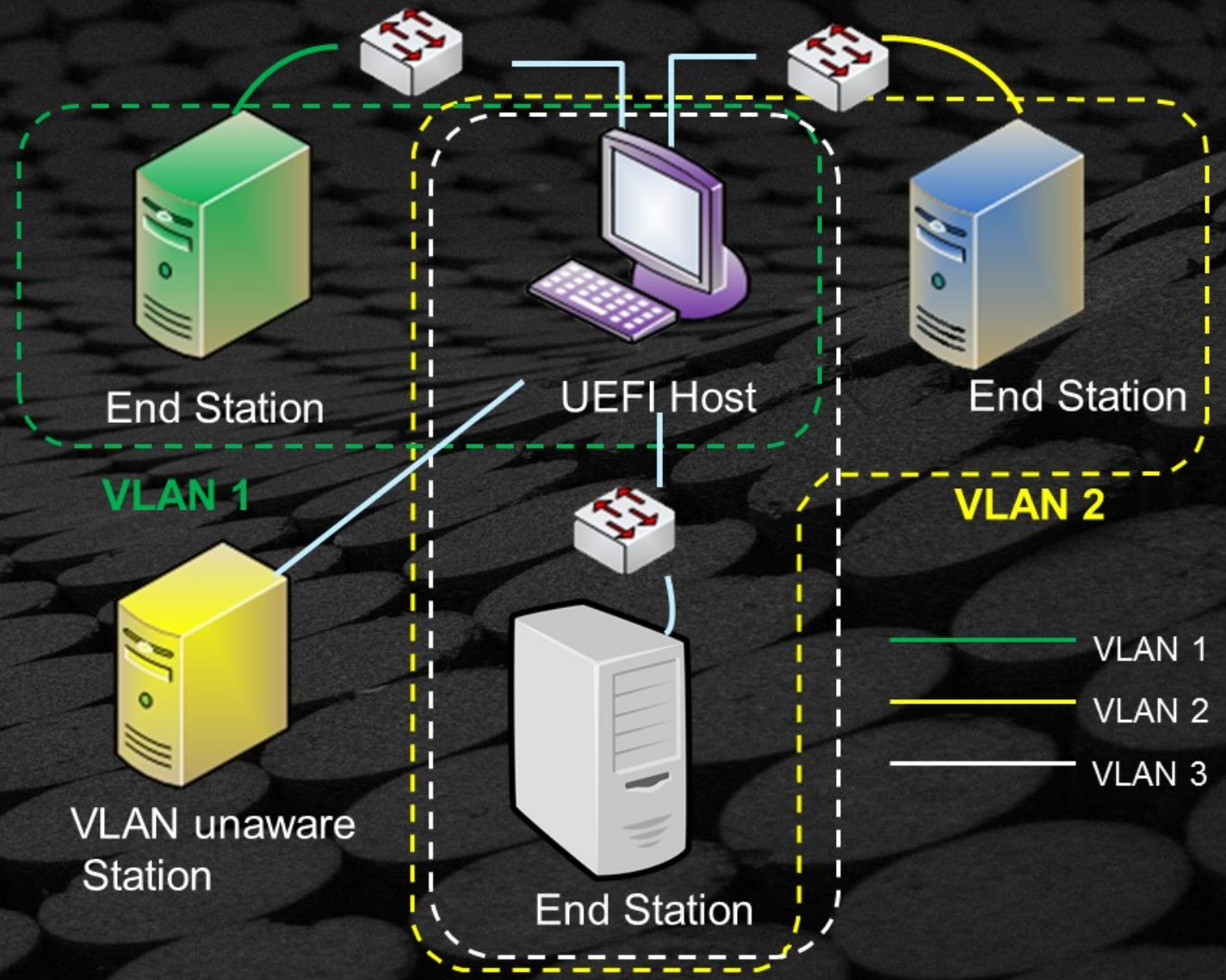
Logical groups of Stations at the data link layer (Tagging)

- 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

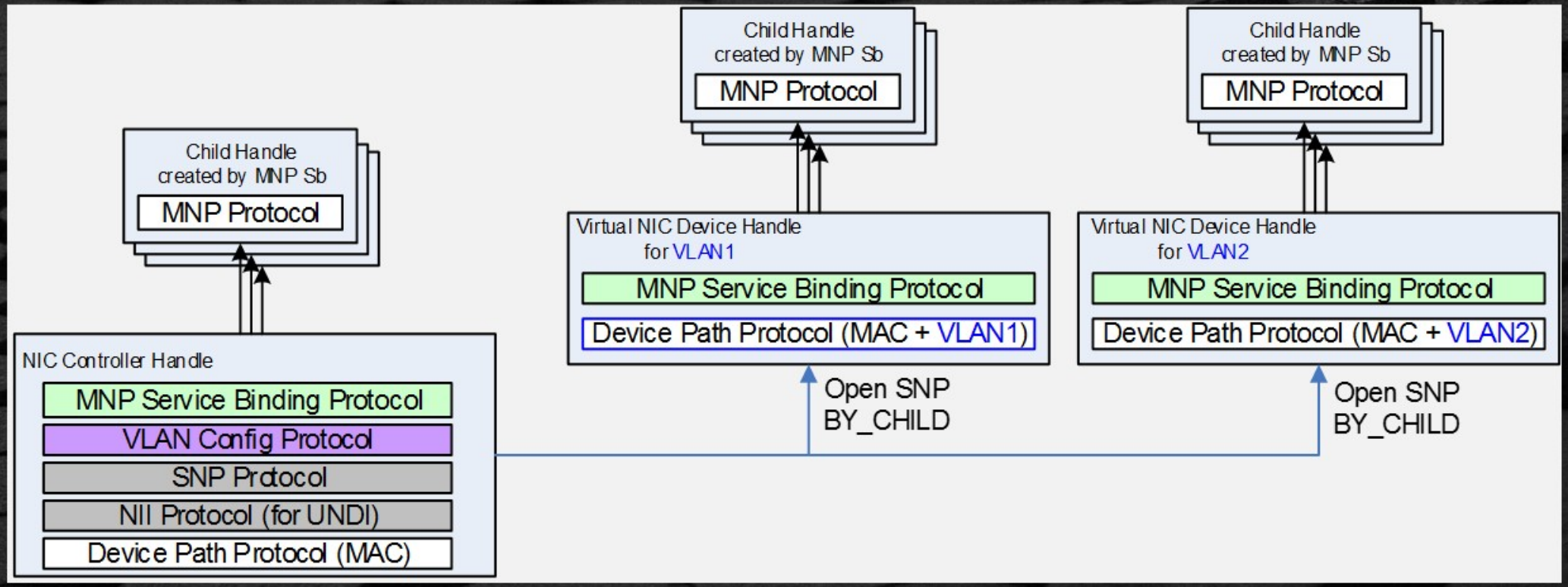


Hybrid link containing both VLAN-aware and VLAN-unaware devices.

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`

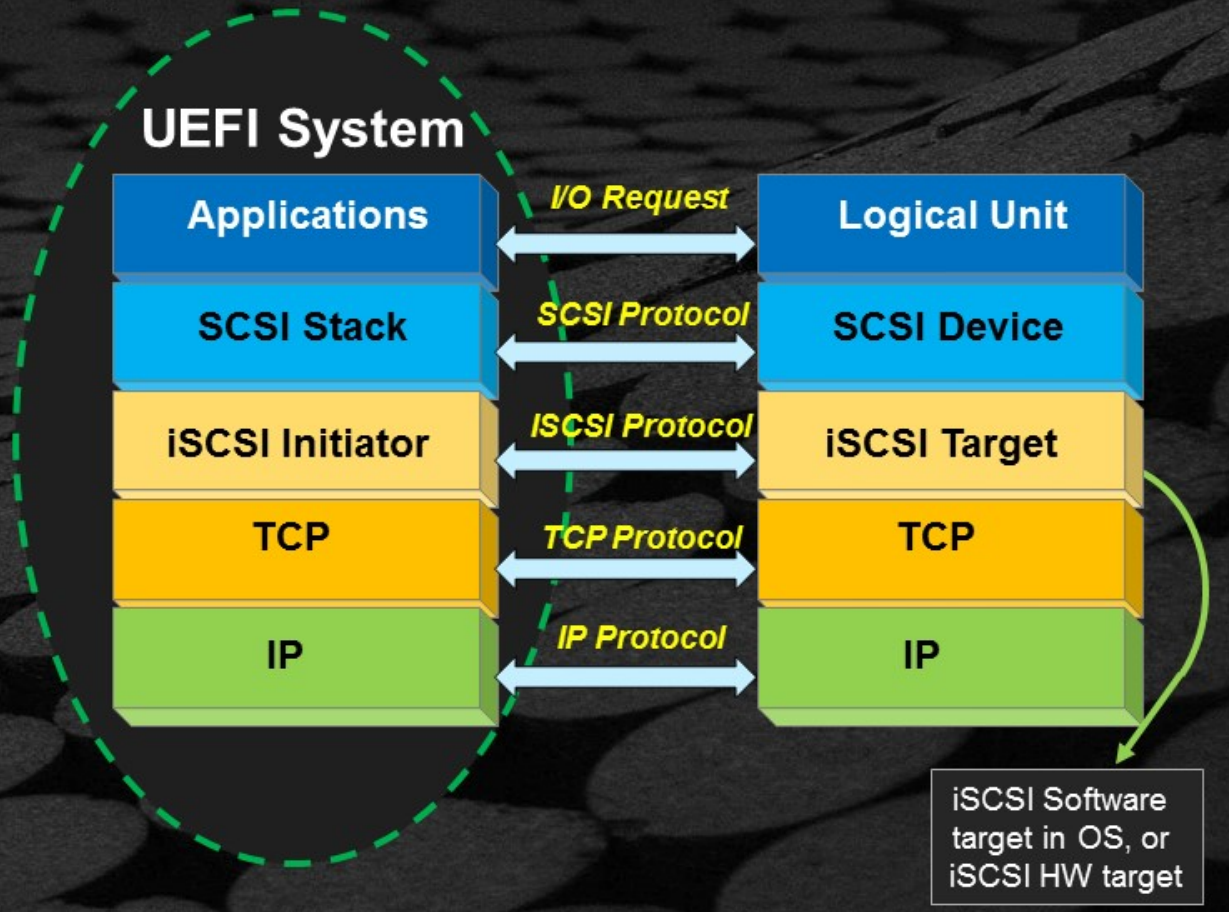


tianocore

UEFI iSCSI Solutions

Uses

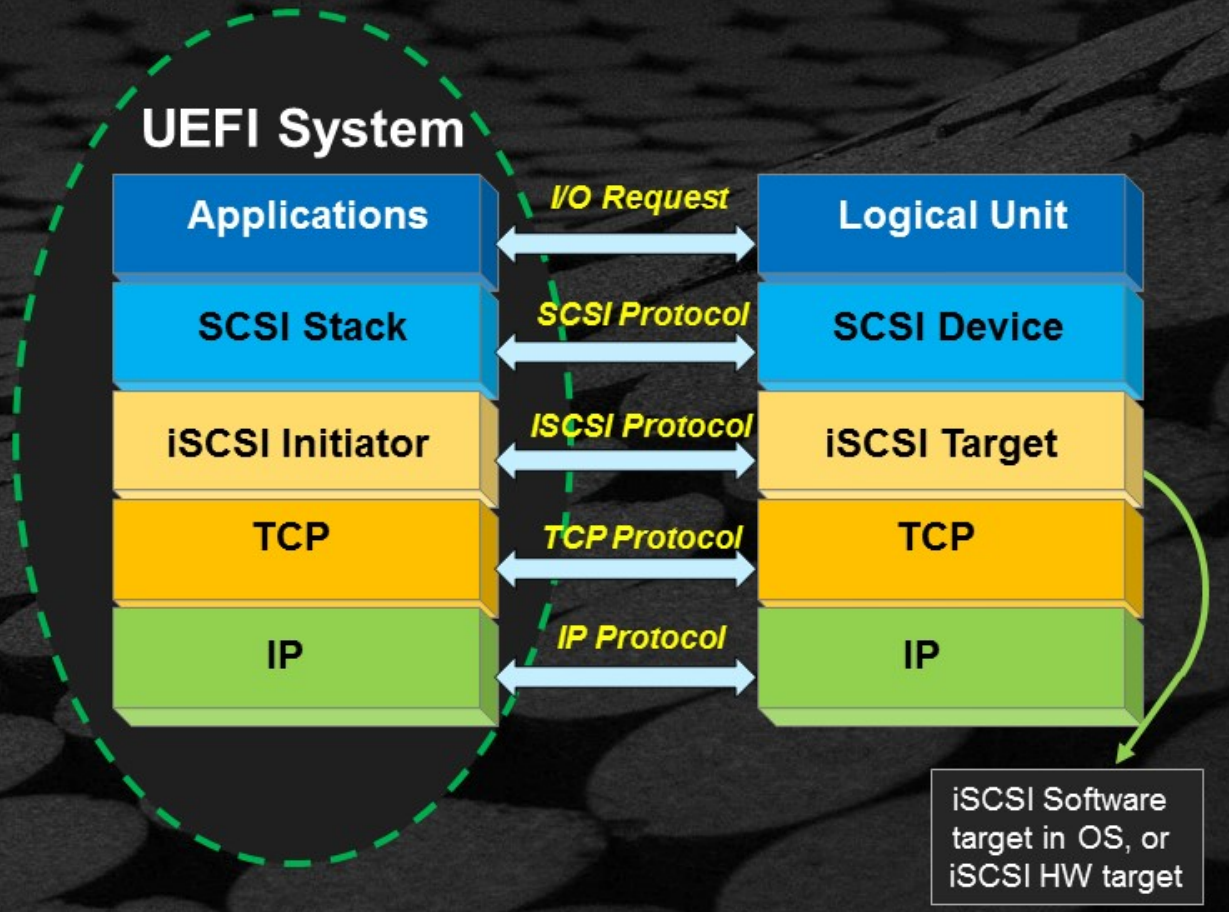
- Storage consolidation
- Build a low cost SAN
- Cluster Shared Volumes
- Diskless Booting



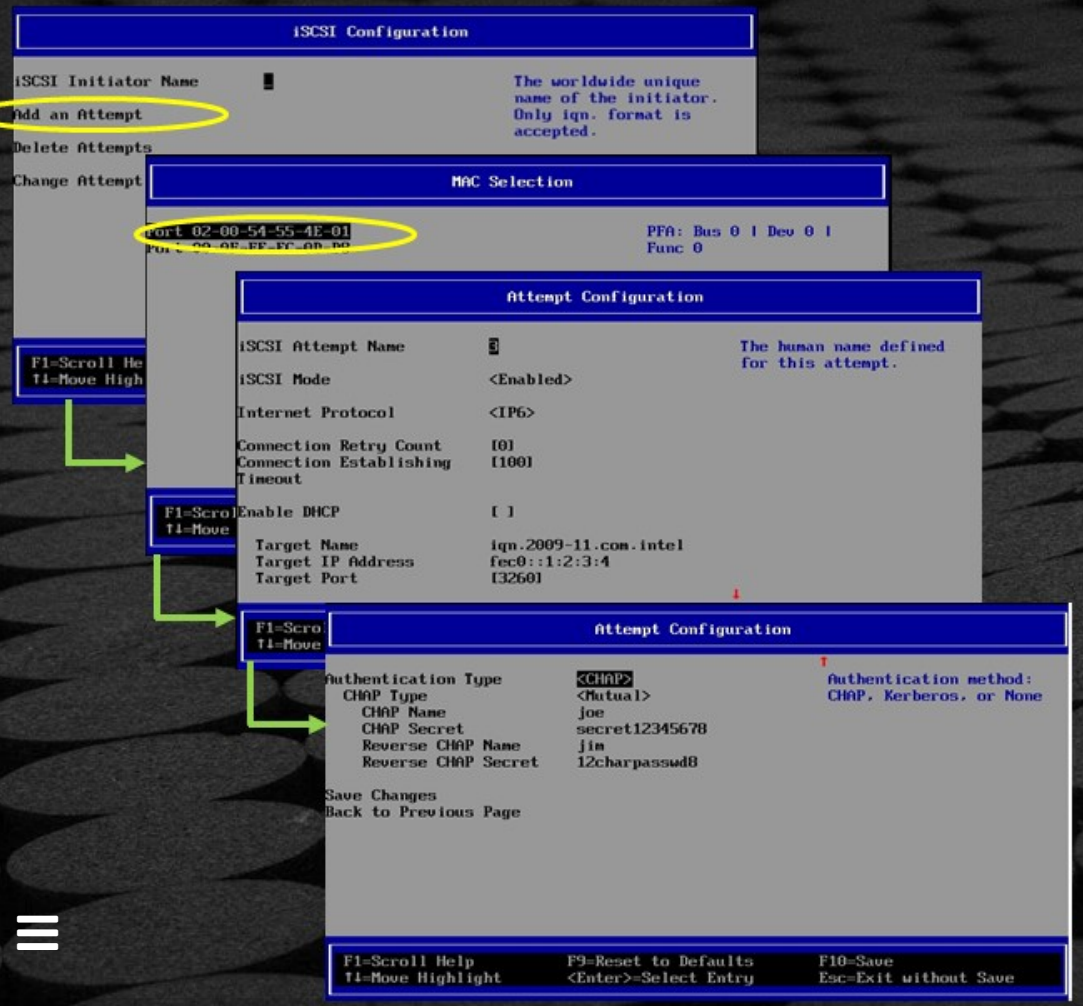
UEFI iSCSI Solutions

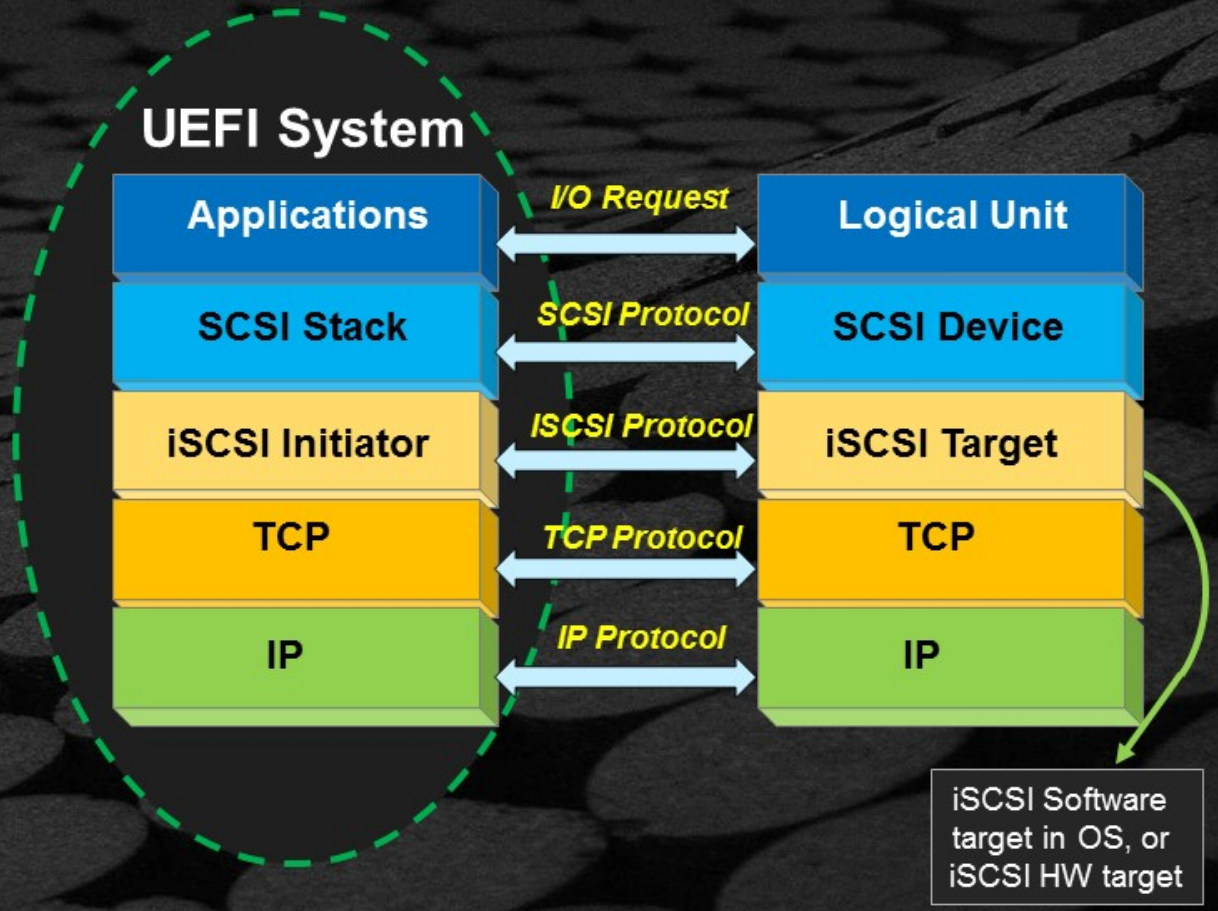
SAN/Data center boot over iSCSI

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

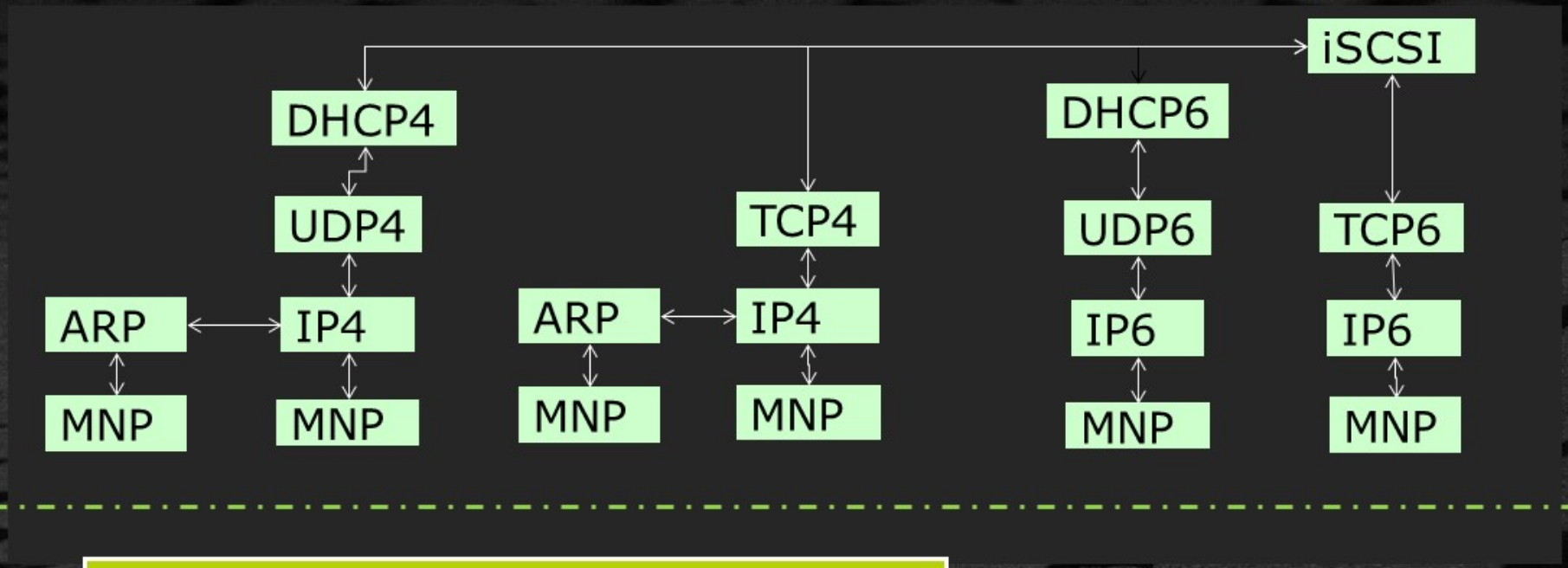


UEFI iSCSI Solutions





Dual-Stack Heritage – iSCSI usage model EDK II



IPsec – Network Security

Secure Internet Protocol Communication

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

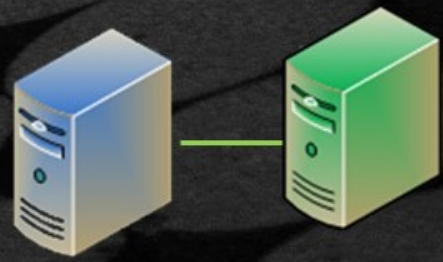
Features include

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

iSCSI initiator 1
10.239.10.96/24

iSCSI initiator 2
Home Addr: fec0::1:2:3:4/64
Virtual Addr: 2011::1:2:3:4/64

iSCSI target 1
10.239.10.204/24



IPsec Gateway
fec0::5:6:7:8/64

iSCSI target 2
2011::a:b:c:d/64

IPsec – Network Security

Secure Internet Protocol Communication

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

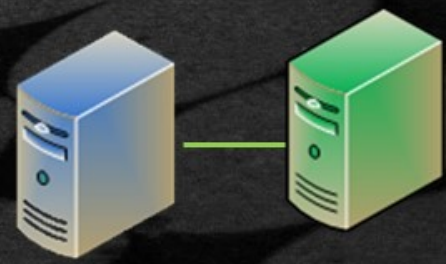
Features include

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

iSCSI initiator 1
10.239.10.96/24

iSCSI initiator 2
Home Addr: fec0::1:2:3:4/64
Virtual Addr: 2011::1:2:3:4/64

iSCSI target 1
10.239.10.204/24



IPsec Gateway
fec0::5:6:7:8/64

iSCSI target 2
2011::a:b:c:d/64

IPsec – Network Security

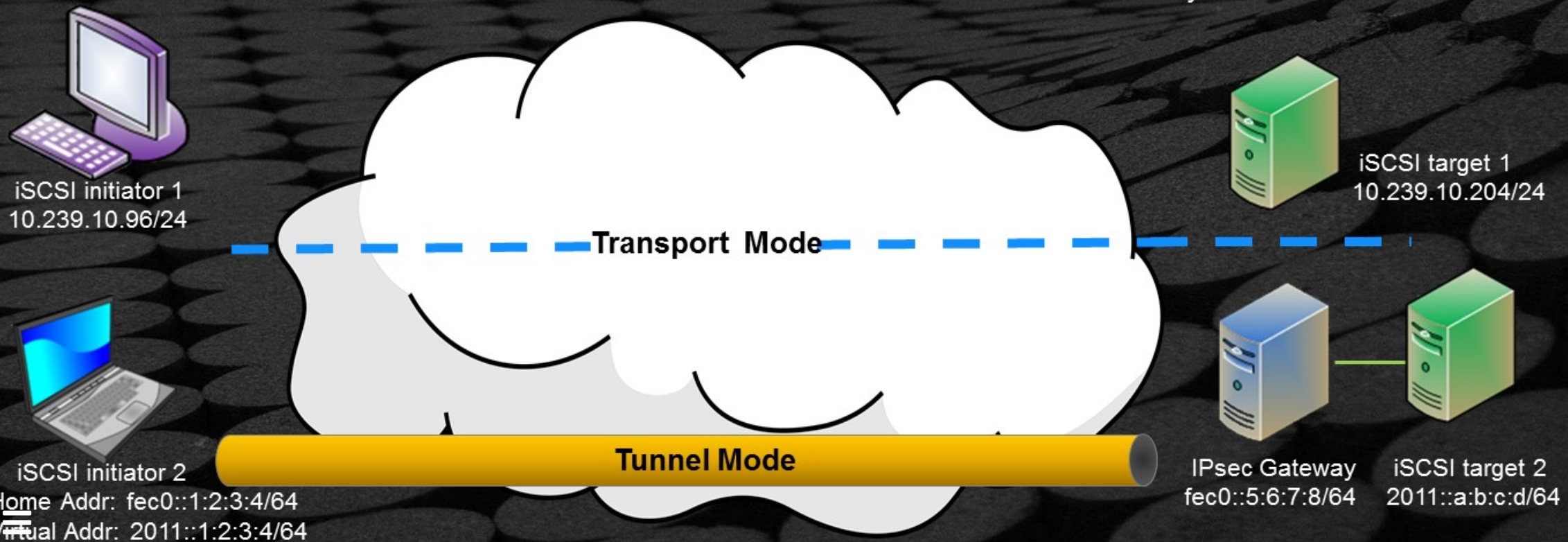
Secure Internet Protocol Communication

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

2 Modes

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



IPsec – Network Security

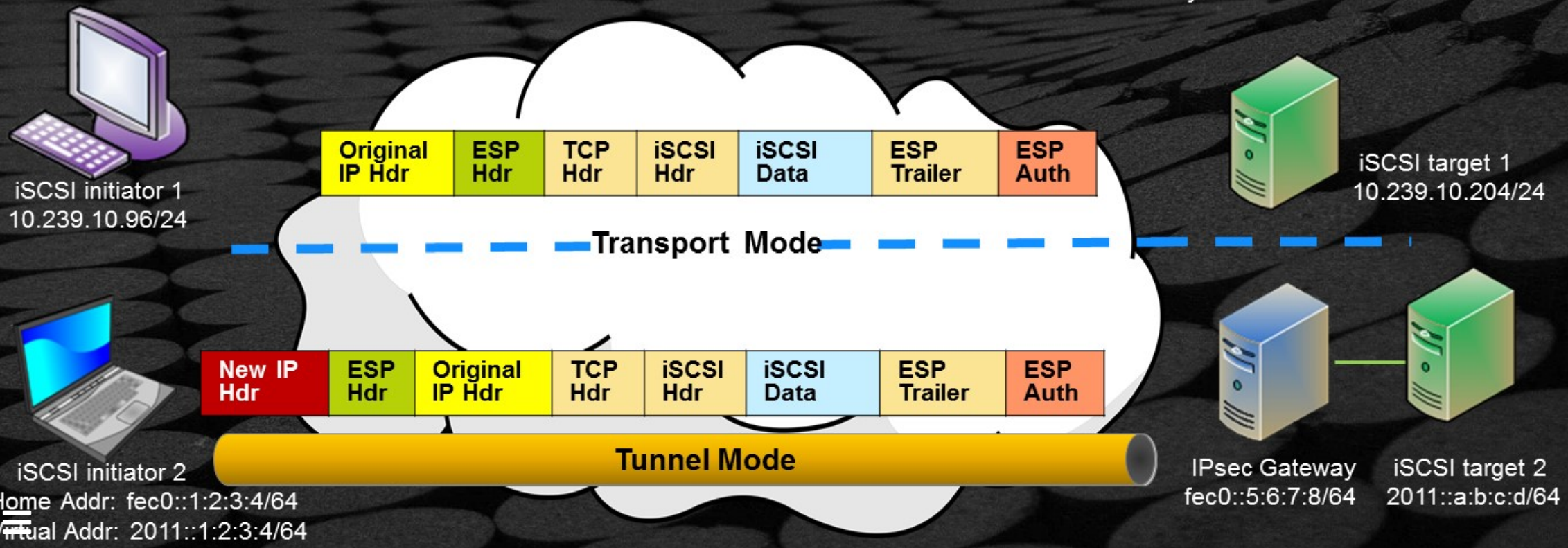
Secure Internet Protocol Communication

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

2 Modes

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



tianocore

IPsec – Network Security

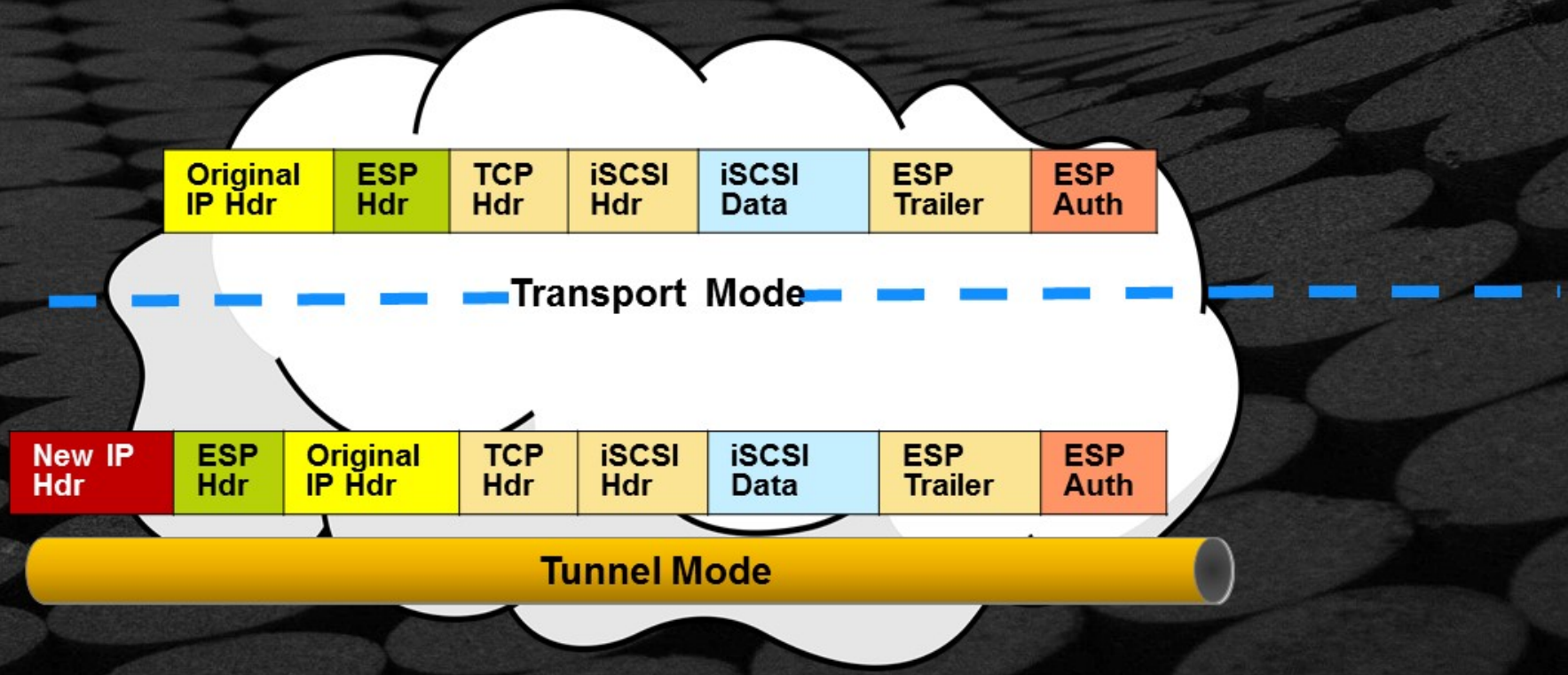
Secure Internet Protocol Communication

- 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 – Network Security

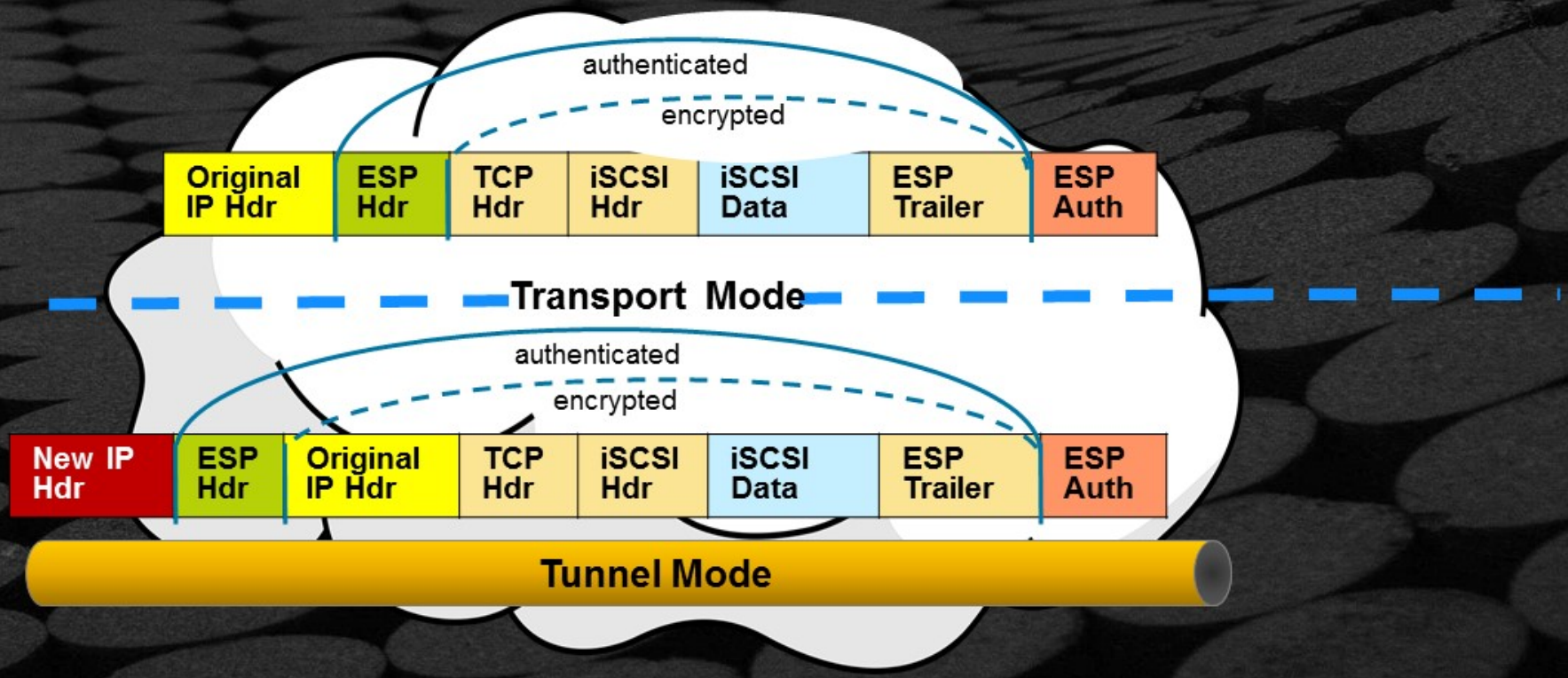
Secure Internet Protocol Communication

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

2 Modes

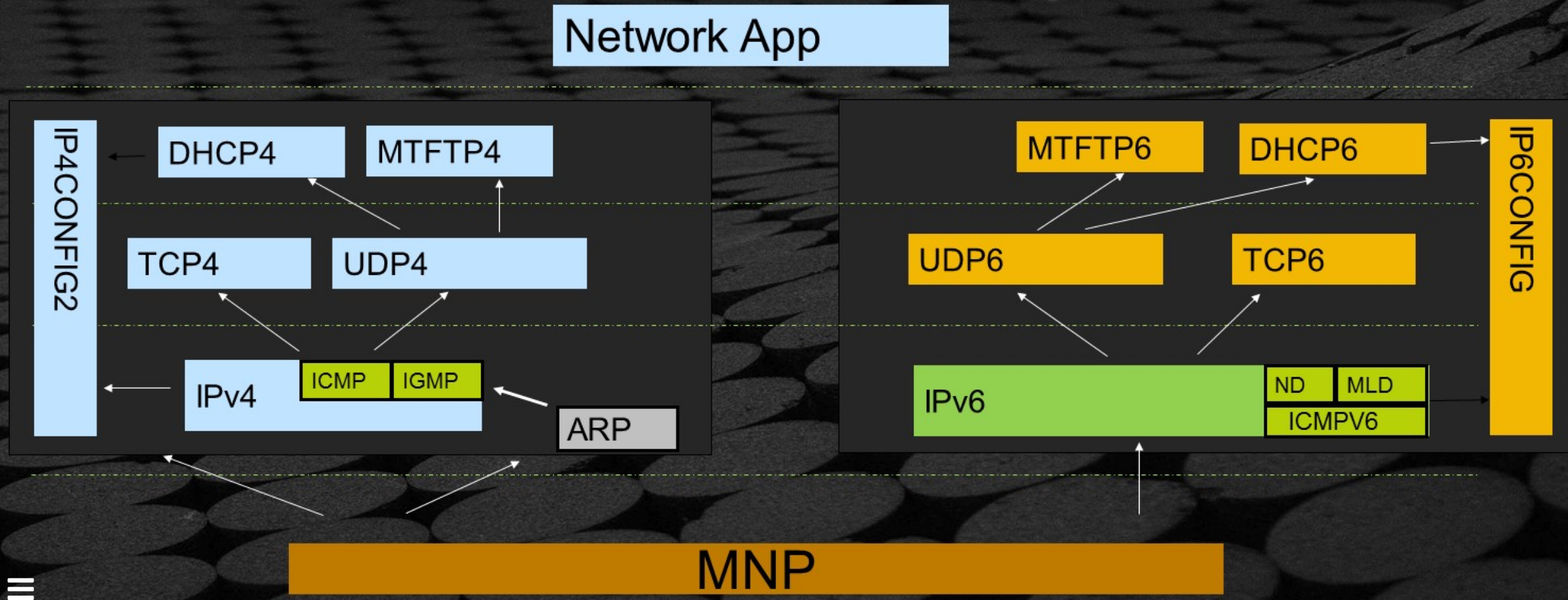
Features include

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



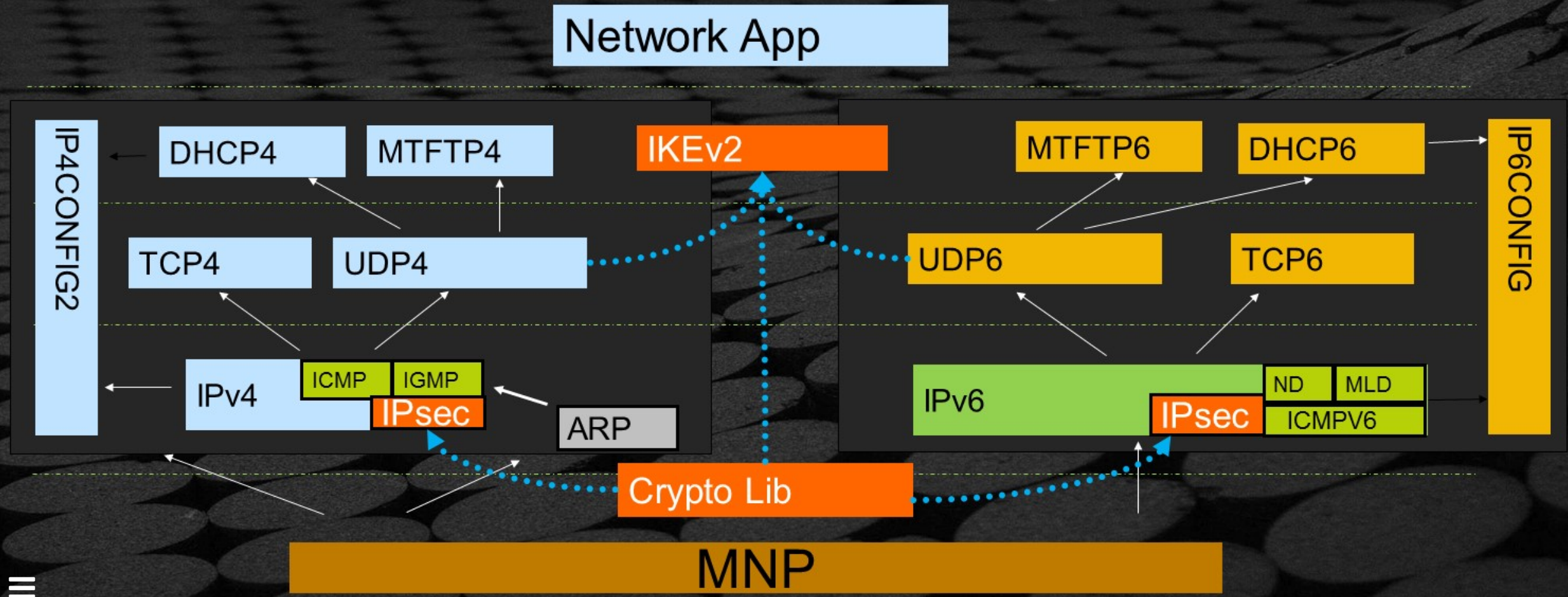
tianocore

IPsec support: shared



tianocore

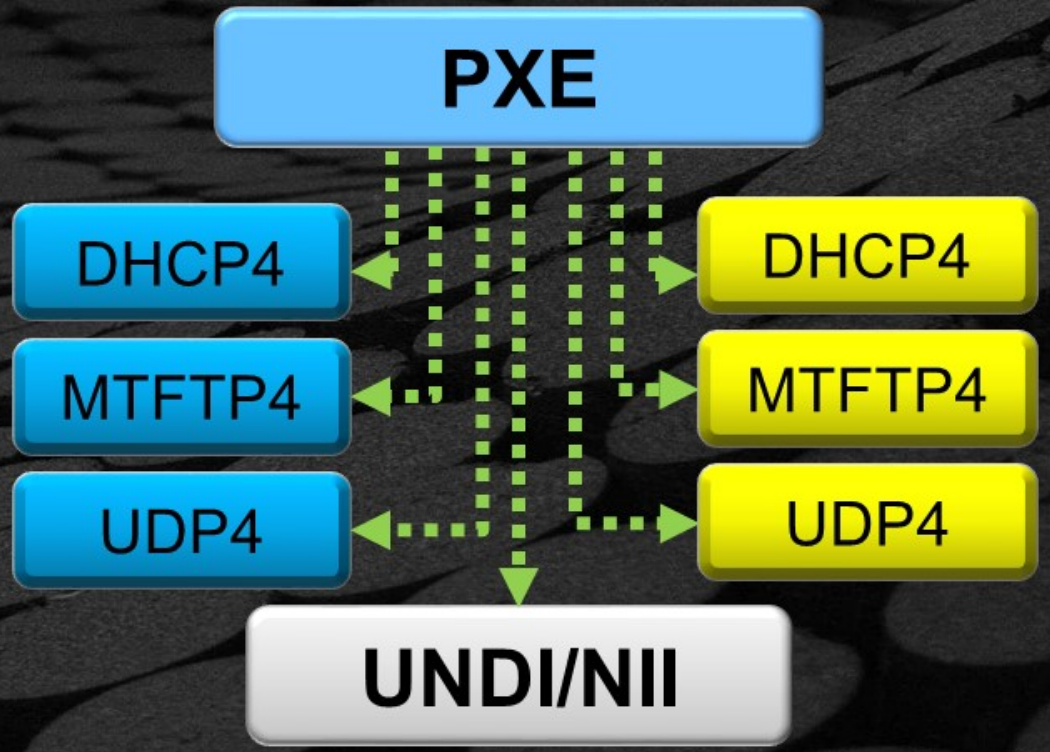
IPsec support: shared



tianocore

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 in - IETF RFC 5970
  - DUID-UUID support
    - Use SMBIOS system GUID as UUID

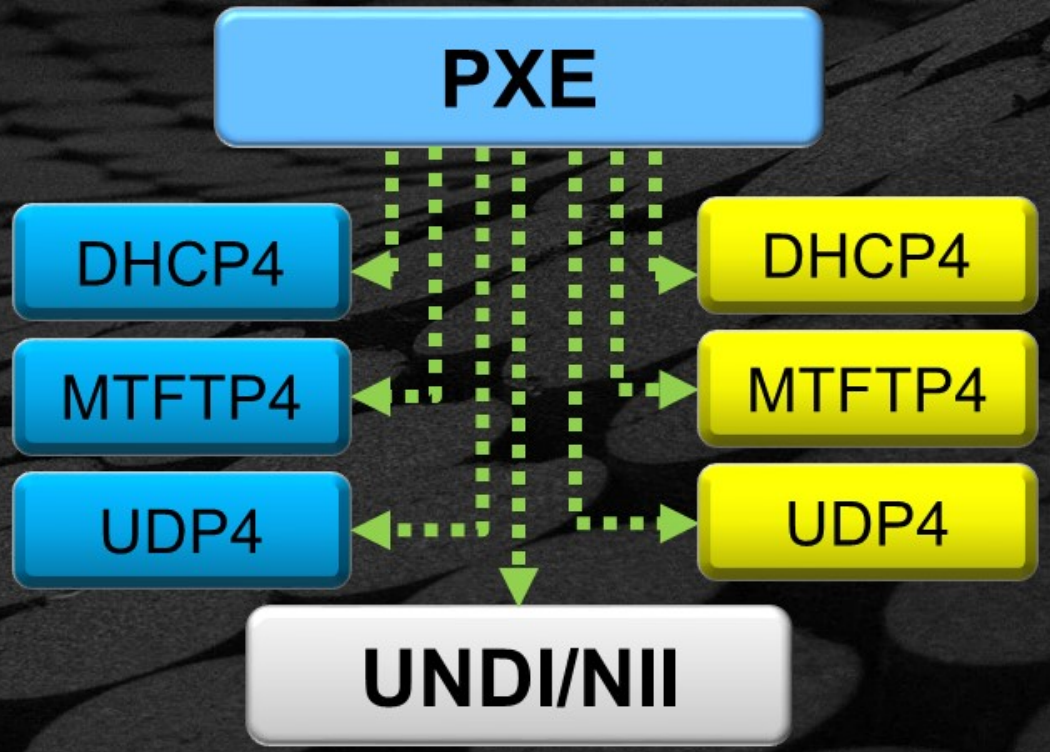


tianocore

UEFI PXE Solutions

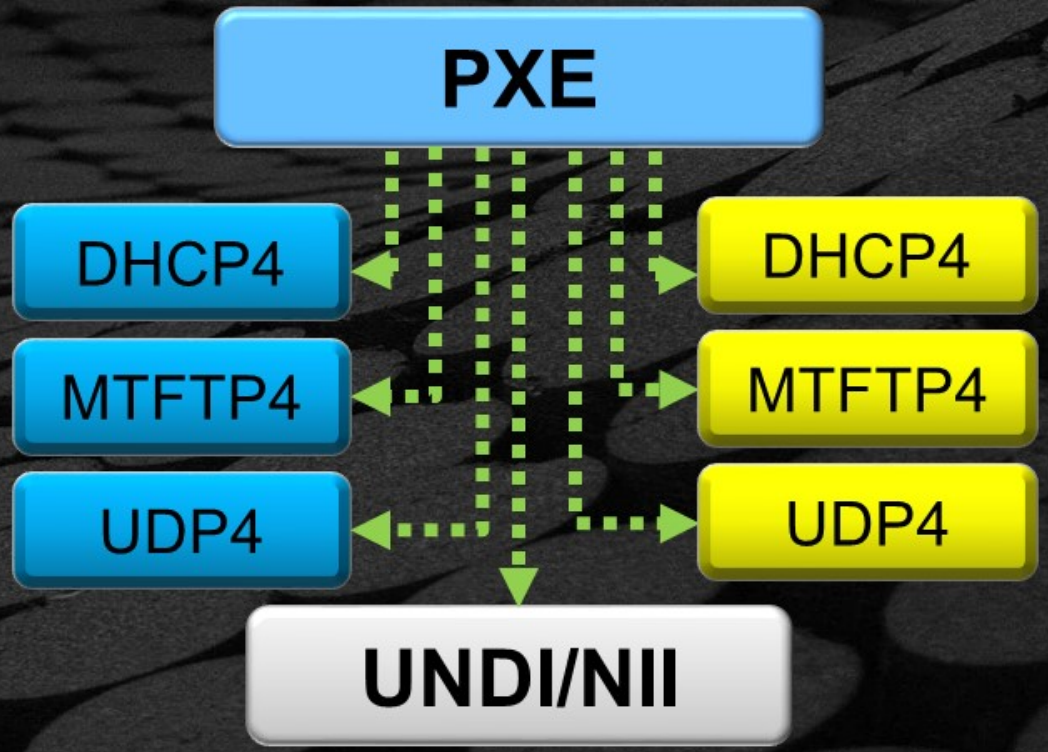
- Preboot eXecution Environment
  - General network booting
    - Independent of data storage device
  - IPv4 based PXE is defined in PXE 2.1
  - IPv6 based PXE is defined in UEFI 2.3
- Technology includes
  - Dual network stack support
  - Evolution of network boot to IPv6 defined in IETF RFC 5970
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

BUT



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

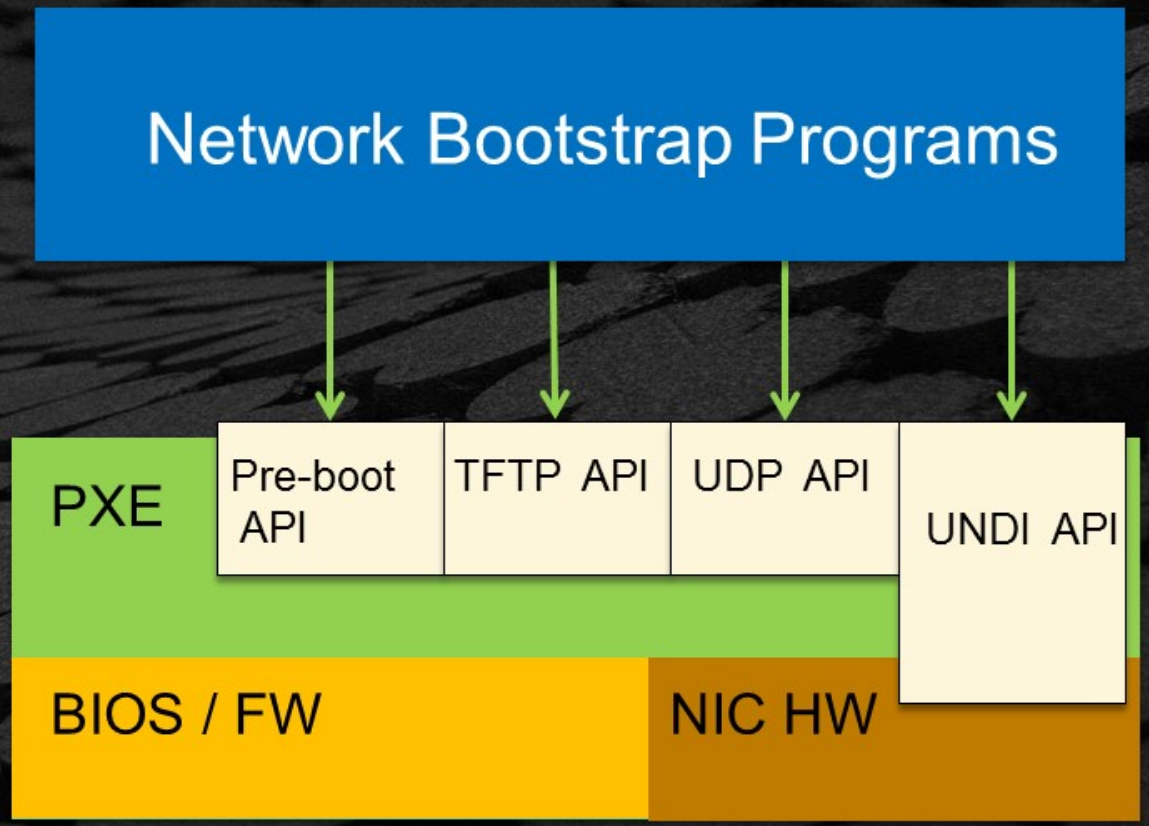


BUT PXE is not keeping up with modern data center needs

tianocore

PXE Boot Challenges

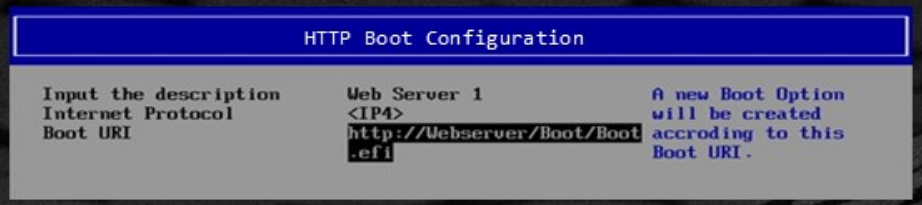
- Security Issues
  - Only physical. No encryption or authentication.
  - Rouge DHCP servers, man-in-the-middle attacks
- Scaling issues
  - Circa 1998
  - TFTP timeouts / UDP packet loss
  - Download time = deployment time = $$$
  - Aggravated in density-optimized data centers
- OEMs and users workarounds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

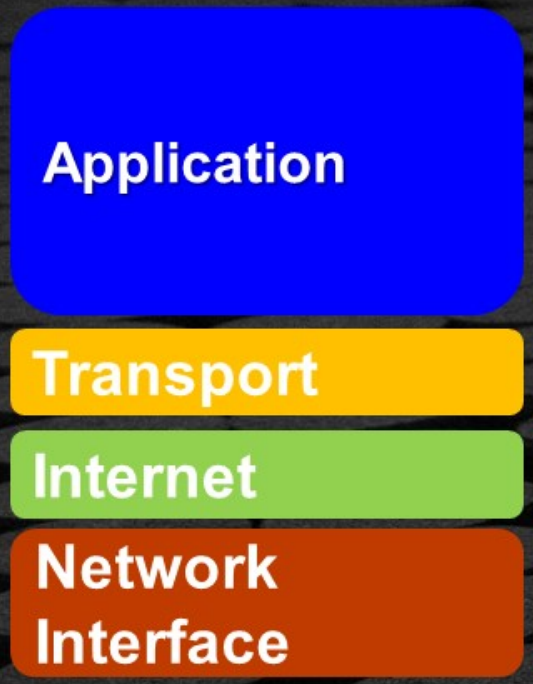


Why not solve PXE Boot challenges natively with standards using UEFI

HTTP(s) Boot Solutions

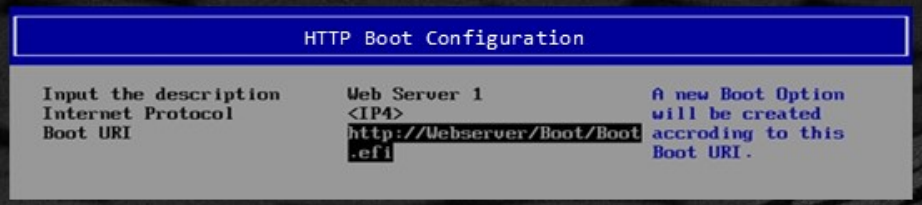
Add HTTP(s) to Network Stack

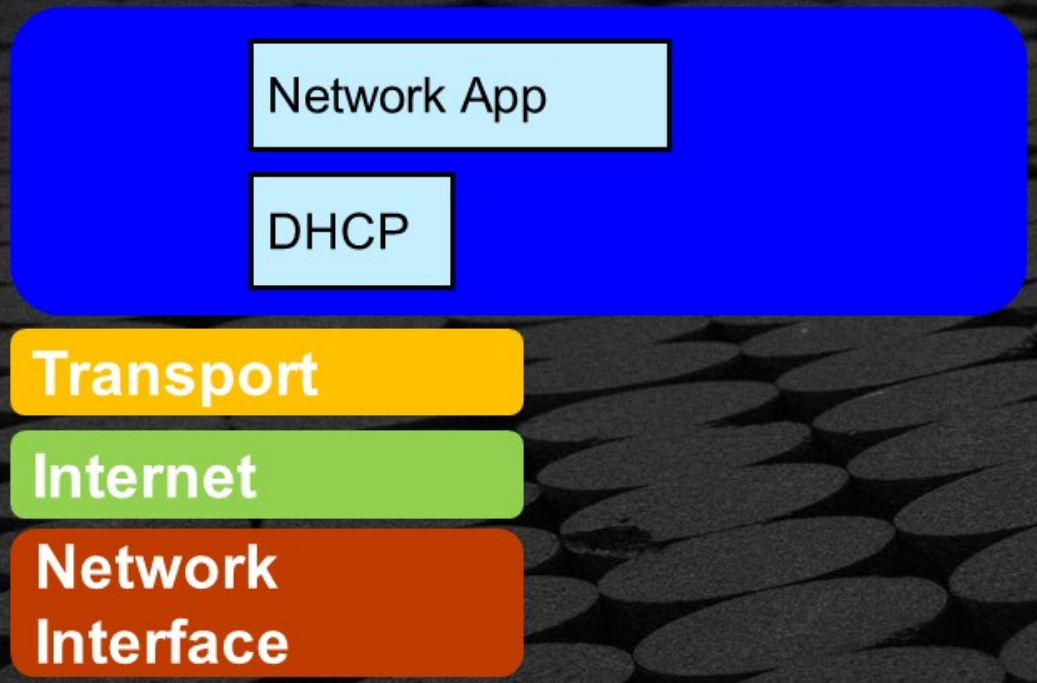




HTTP(s) Boot Solutions

Add HTTP(s) to Network Stack

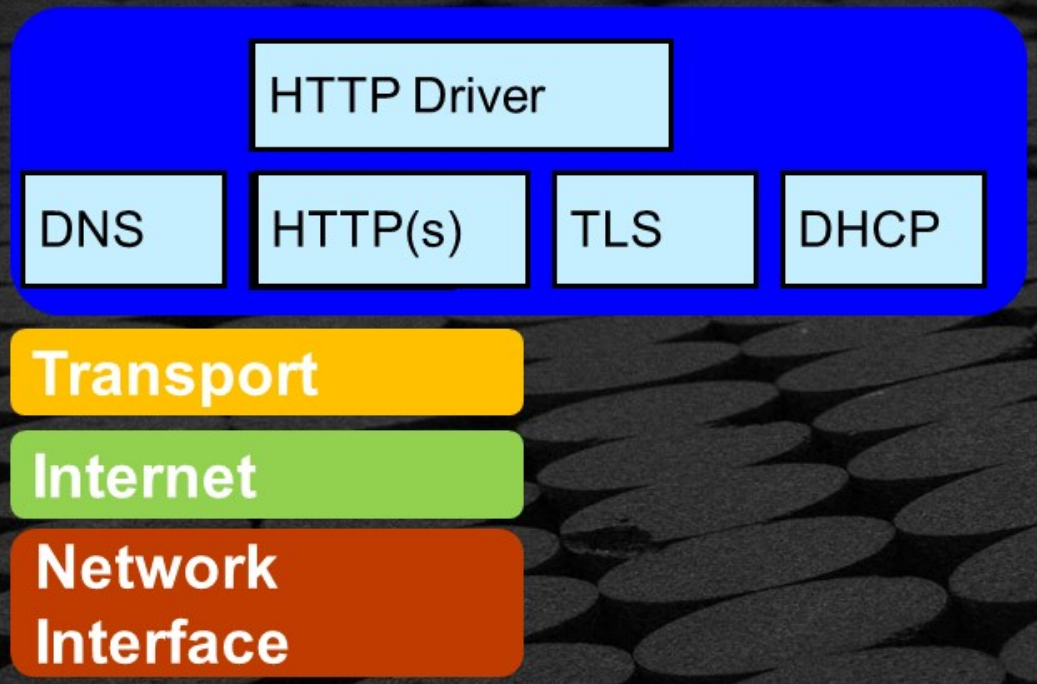


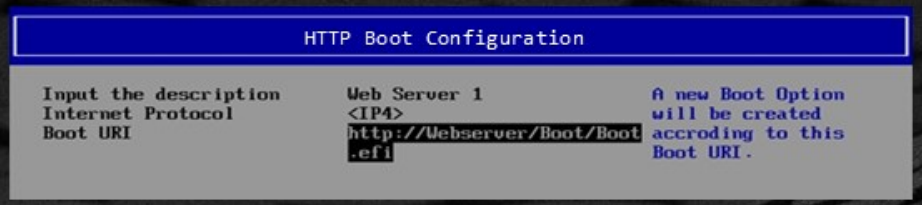


tianocore

HTTP(s) Boot Solutions

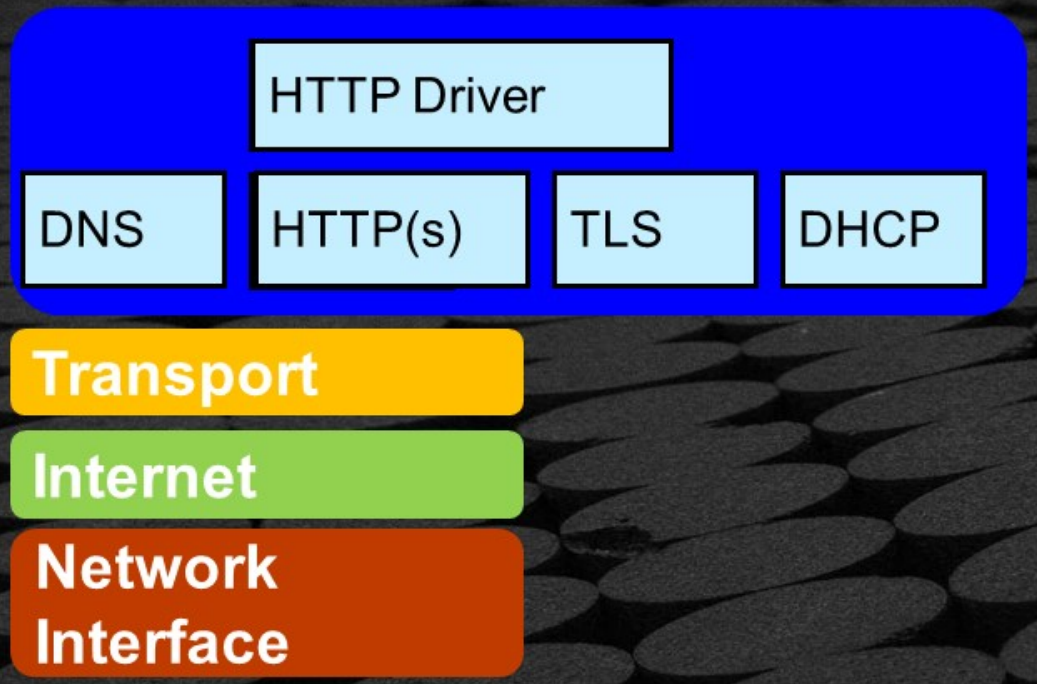
Add HTTP(s) to Network Stack

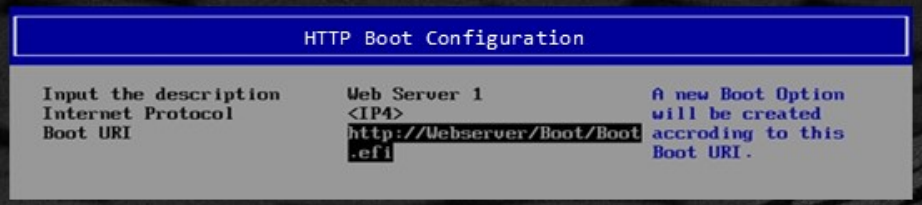




HTTP(s) Boot Solutions

Add HTTP(s) to Network Stack



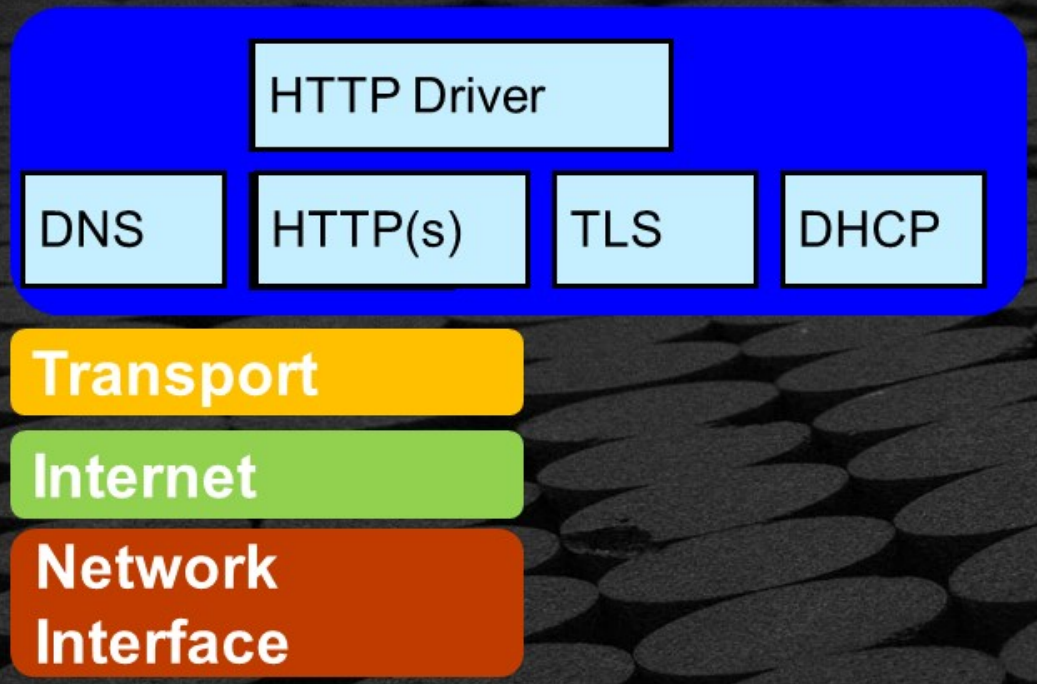


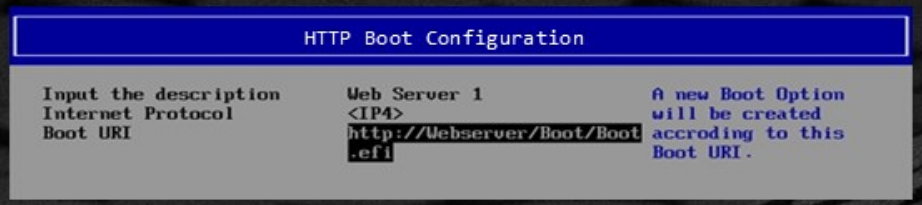
UEFI 2.5 defined RAM Disk device path nodes

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

HTTP(s) Boot Solutions

Add HTTP(s) to Network Stack





UEFI 2.5 defined RAM Disk device path nodes

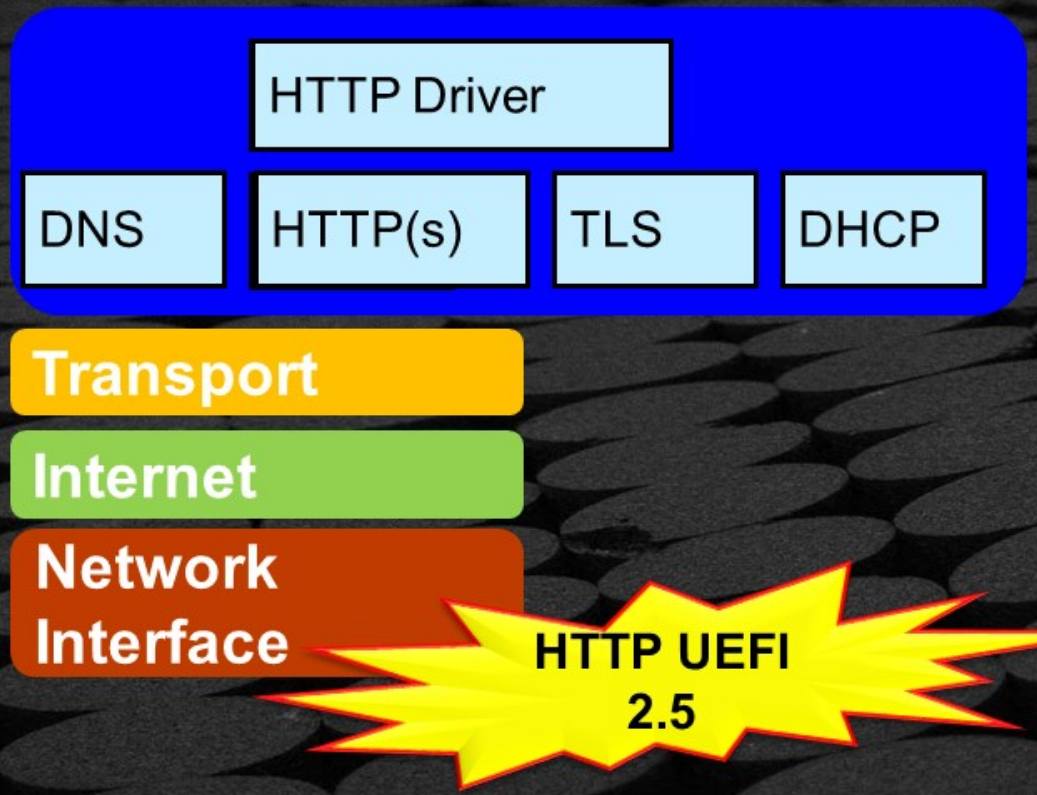
- Standard access to a RAM Disk in UEFI and Virtual CD (ISO image)

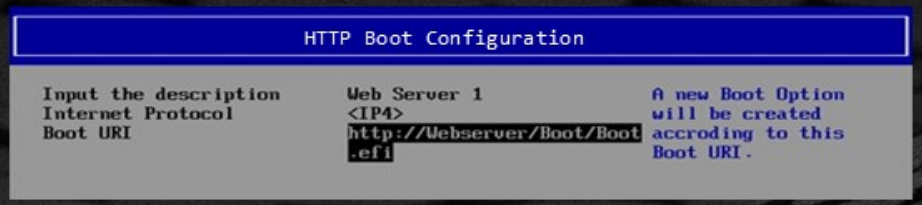
ACPI 6.0 NVDIMM Firmware Interface Table (NFIT)

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

HTTP(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

tianocore

EDK II UEFI PROTOCOLS

What UEFI Protocols Make Network Work

- A deeper dive into the Network implementation in EDK II

tianocore

UEFI Driver - Problem Statement

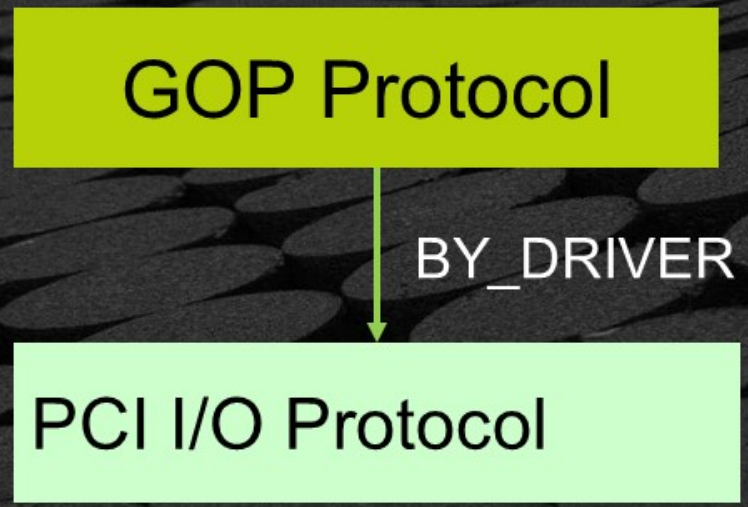
- The original UEFI Driver Model is not complete
- Good support for HW Device Driver
- Good support for HW Bus/Hybrid Driver
- Good support for Simple SW Layering driver
- Poor support for complex SW Layering driver

tianocore

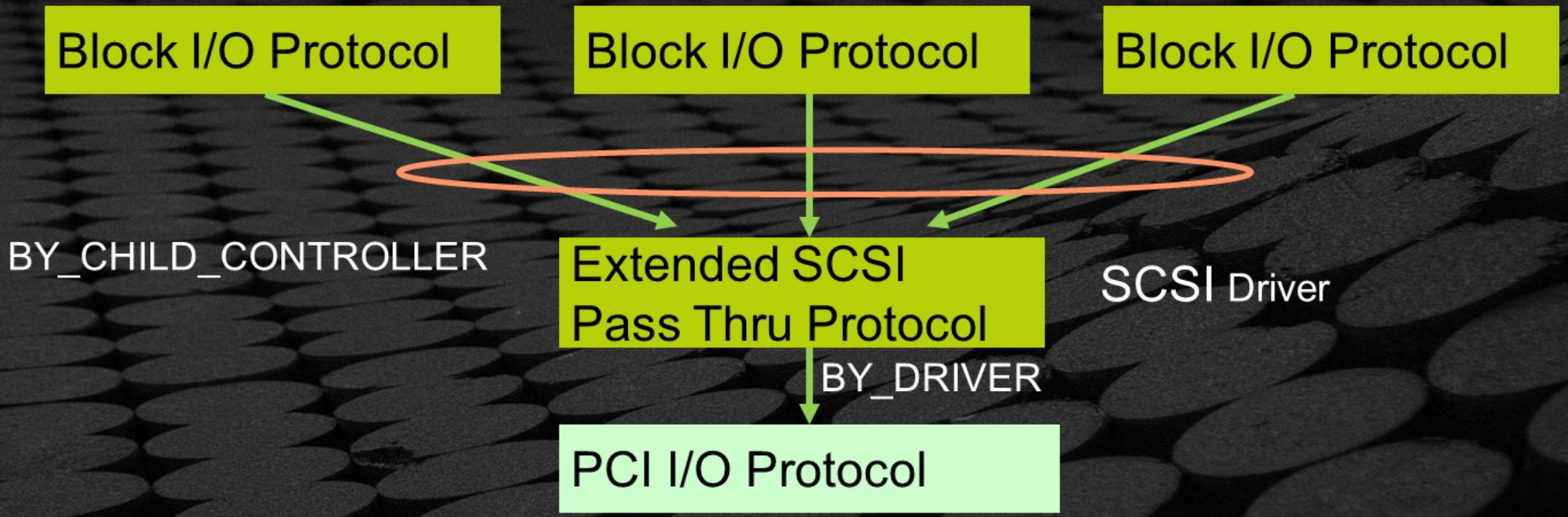
UEFI Hardware Device Driver

Graphics

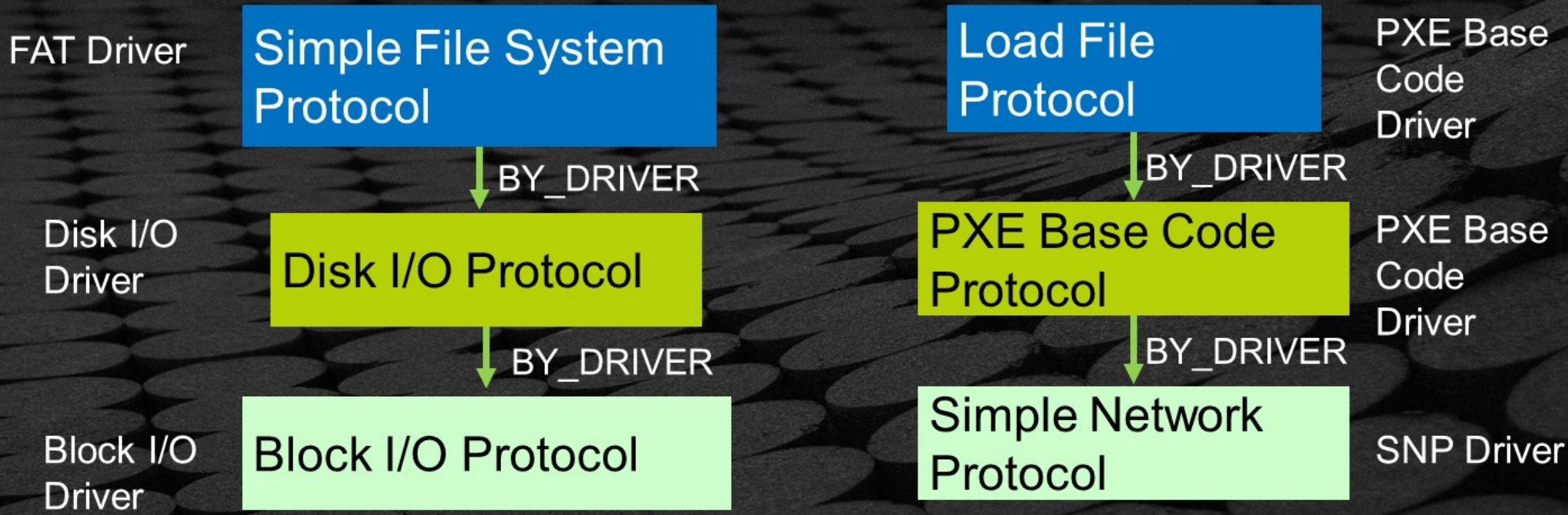
PCI Device Driver



UEFI Hardware Bus/Hybrid Driver

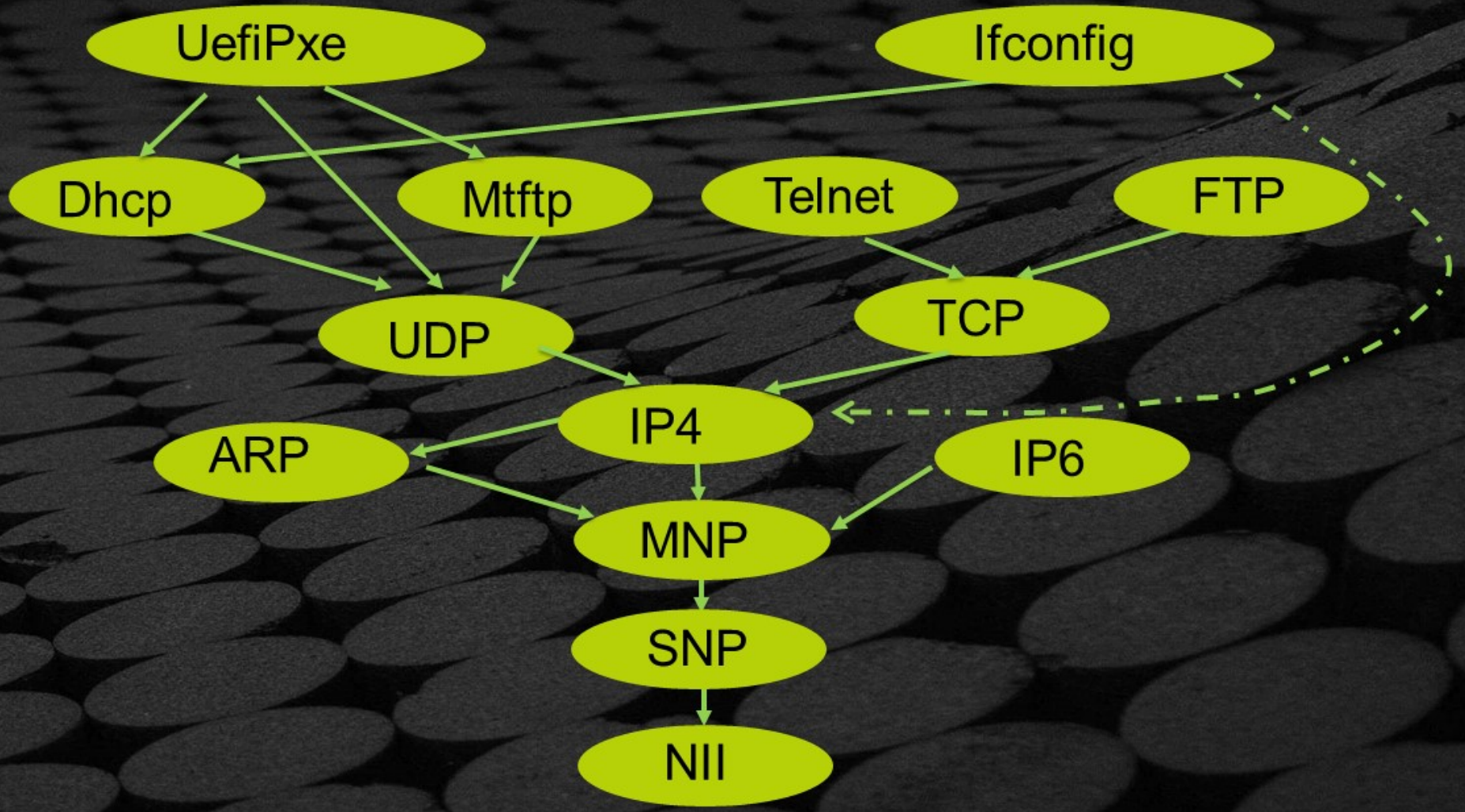


UEFI Simple Software Driver



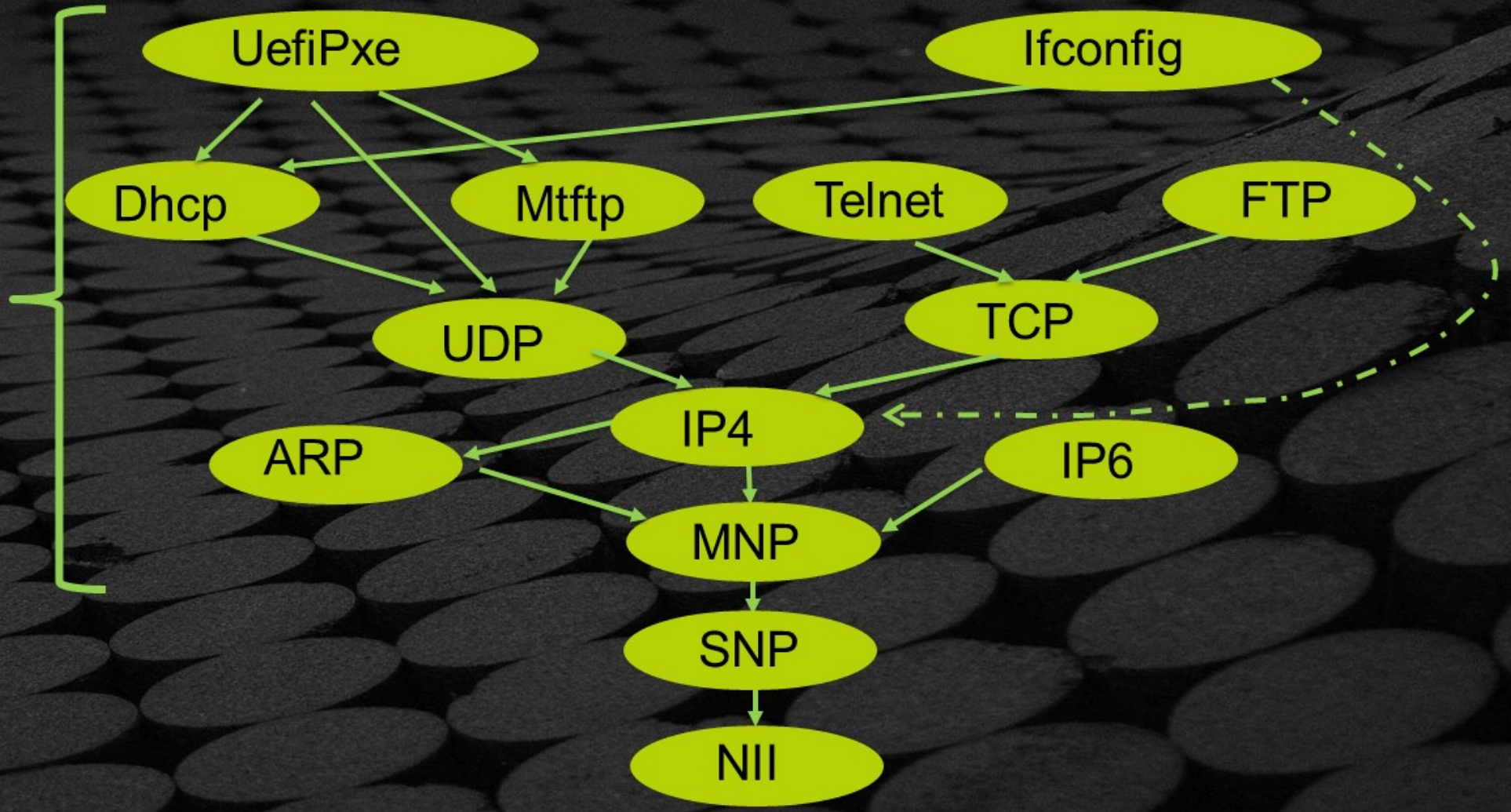
tianocore

Complex Software Drivers



Complex Software Drivers

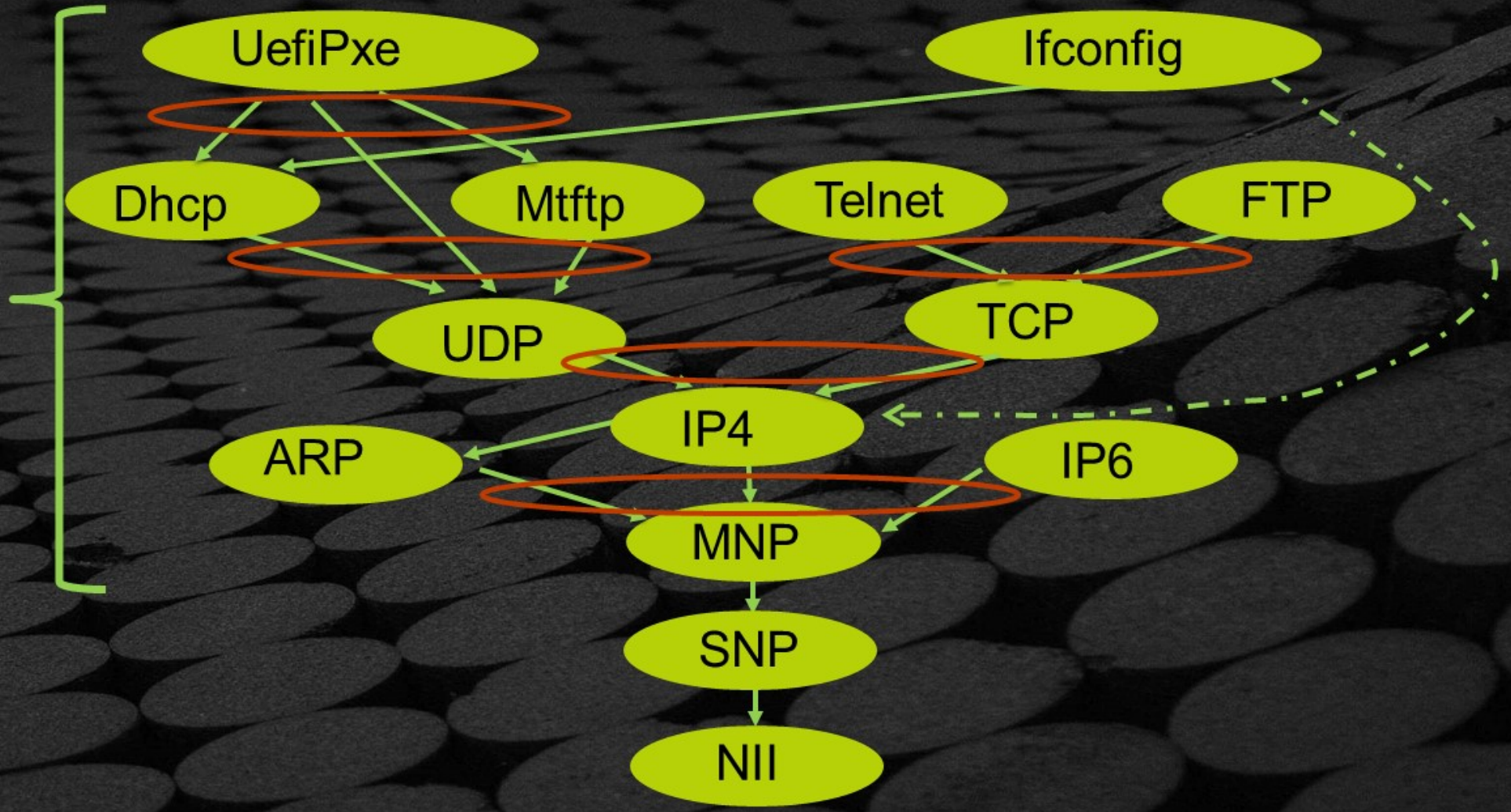
Multiple Consumers



tianocore

Complex Software Drivers

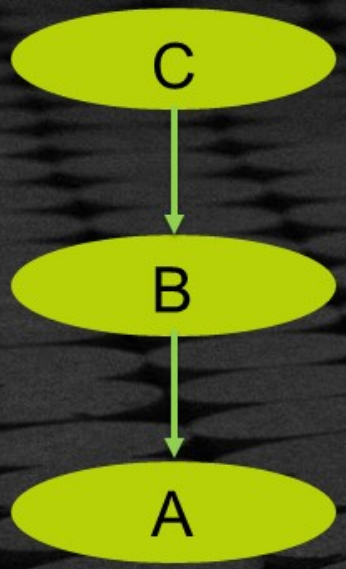
Multiple Consumers



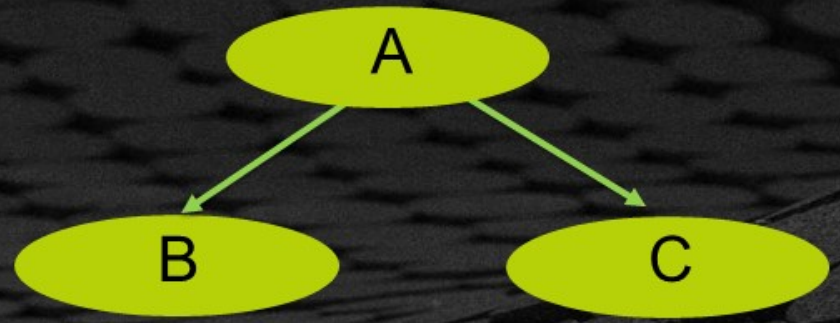
Complex Software Drivers

- 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

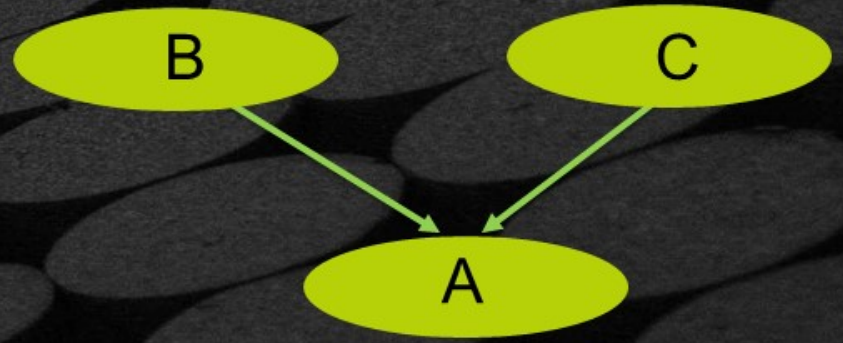
UEFI Service Binding Protocol



Case #1: Linear Stack

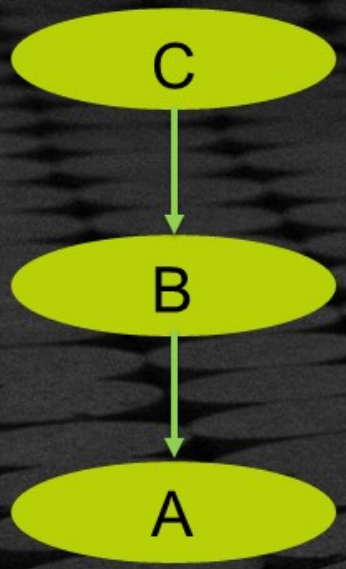


Case #2: Multiple Dependencies

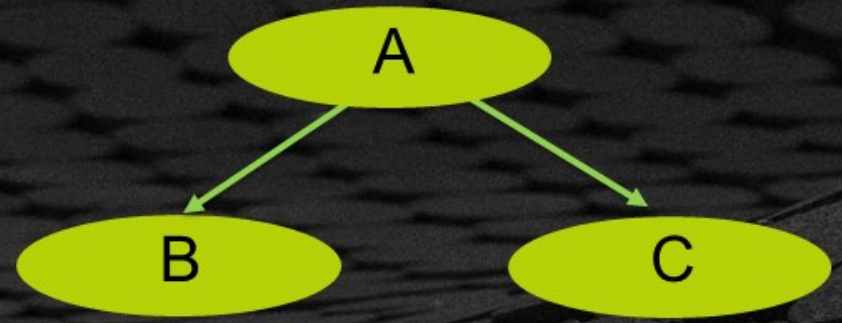


Case #3: Multiple Consumers

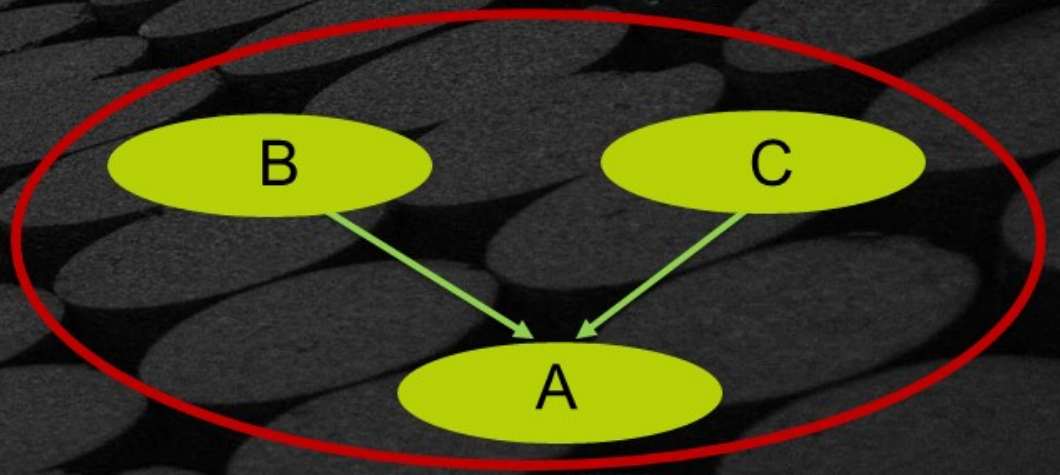
UEFI Service Binding Protocol



Case #1: Linear Stack



Case #2: Multiple Dependencies



Case #3: Multiple Consumers

tianocore

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

Service Binding Protocol - Similar to Object Factory Method

tianocore

UEFI Service Binding Protocol

Code Example

```
typedef struct _EFI_SERVICE_BINDING_PRC
  EFI_SERVICE_BINDING_CREATE_CHILD   Cr
  EFI_SERVICE_BINDING_DESTROY_CHILD Des
} EFI_SERVICE_BINDING_PROTOCOL;
EFI_STATUS
```

```
EFI_STATUS
(EFIAPI *EFI_SERVICE_BINDING_CREATE_CHI
  IN EFI_SERVICE_BINDING_PROTOCOL*This,
  IN OUT EFI_HANDLE*ChildHandle
);
EFI_STATUS
(EFIAPI *EFI_SERVICE_BINDING_DESTROY_CH
  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)

EDK II Only Supports Polling (NO Interrupts)

- The IO engine of the whole software stack – POLLING, either timer driven or invoked in applications.
- **Asynchronous** – data transmission is divided into two part, similar with the asynchronous IO in OS scope: 1. initiate the IO via a **request**; 2. wait for the result of the IO request through **event** notification.
- Most drivers support data block scatter/gather during transmission.
- Separated IP, Route configurations for each instances based on IP, UDP, TCP, MTFTP and DHCP (Both IPv4 and IPv6)
- Instances are bound to a NIC on creation. No 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

Task Priority Level (TPL)

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

Task Priority Level (TPL)

EDK II DPC Protocol

Code Example

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
  );
```

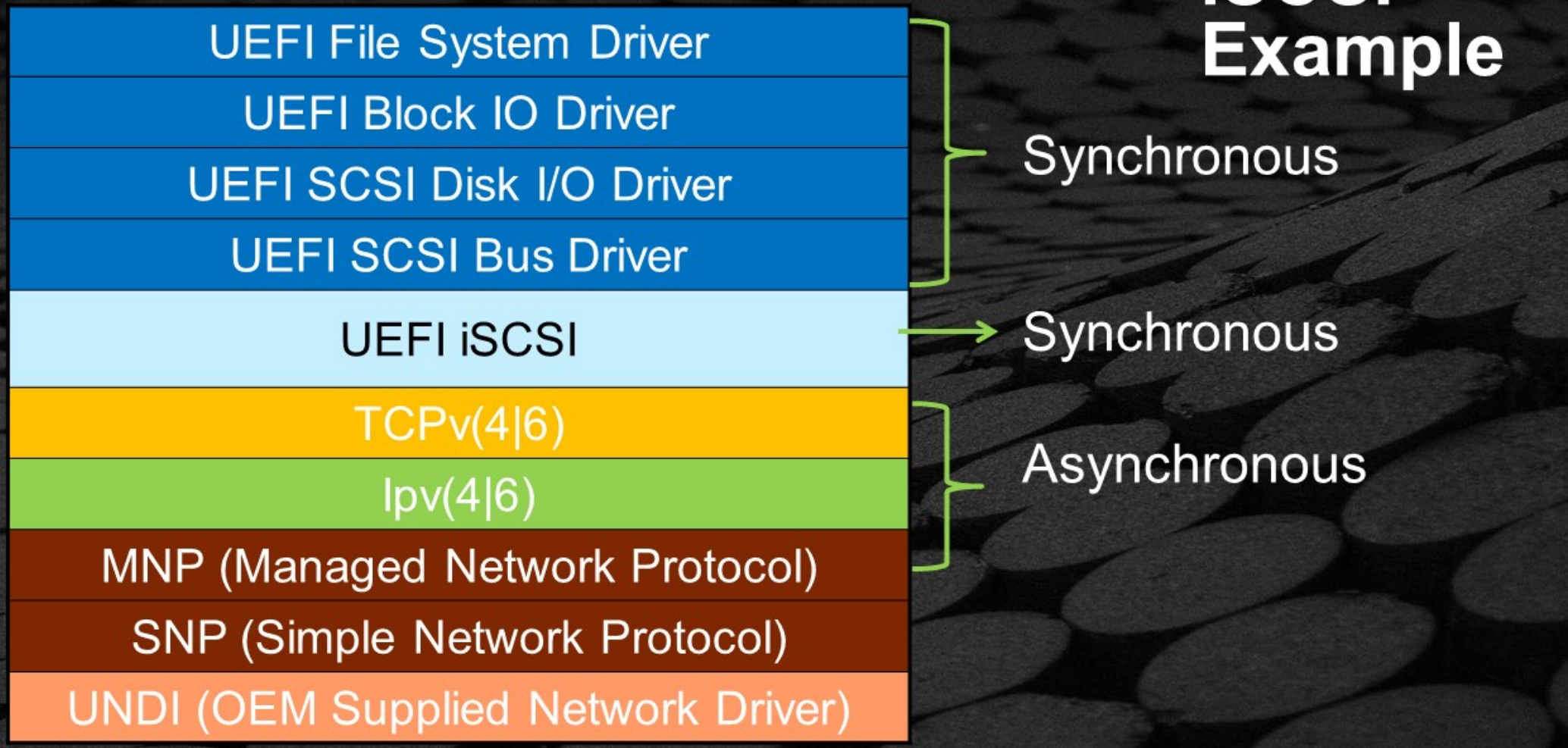
Dispatch DPC

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

tianocore

WHY DPC?

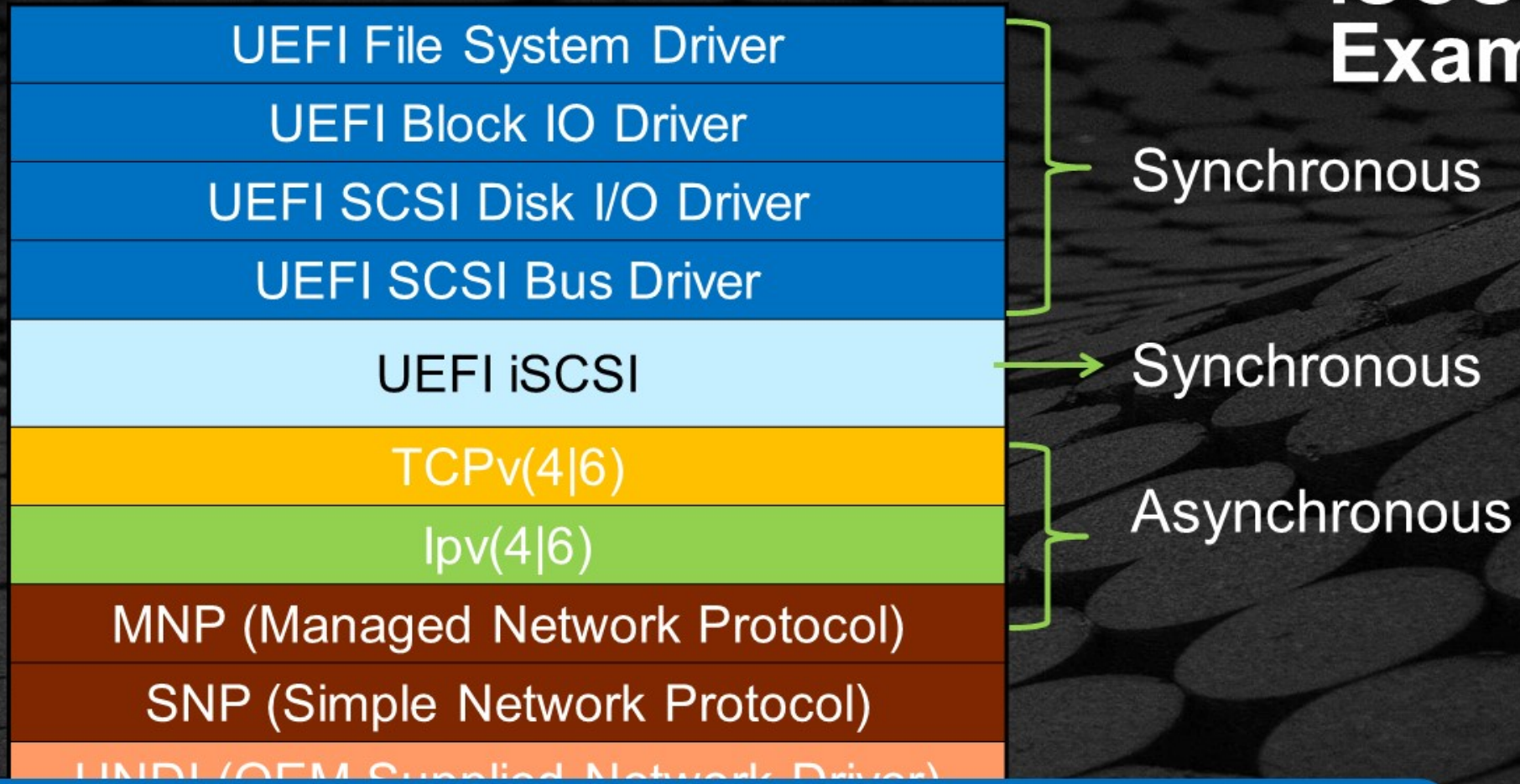
iSCSI Example



tianocore

WHY DPC?

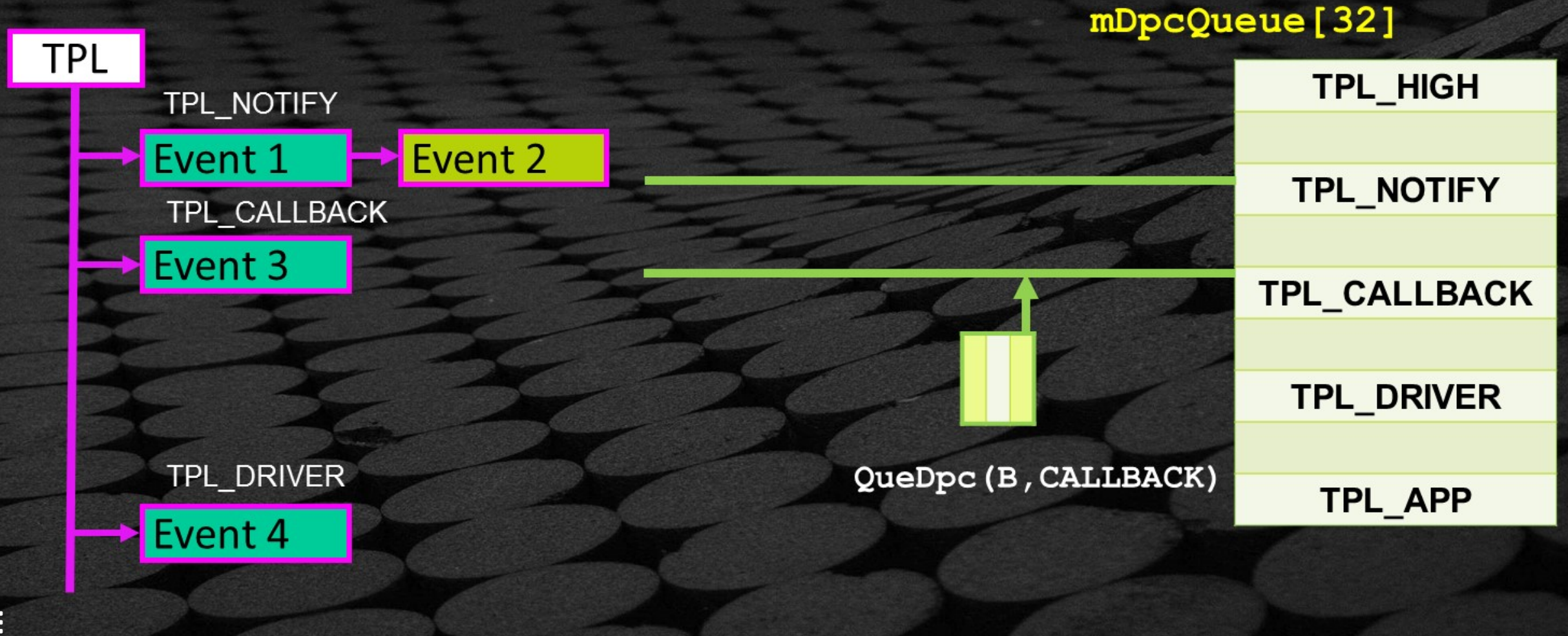
iSCSI Example



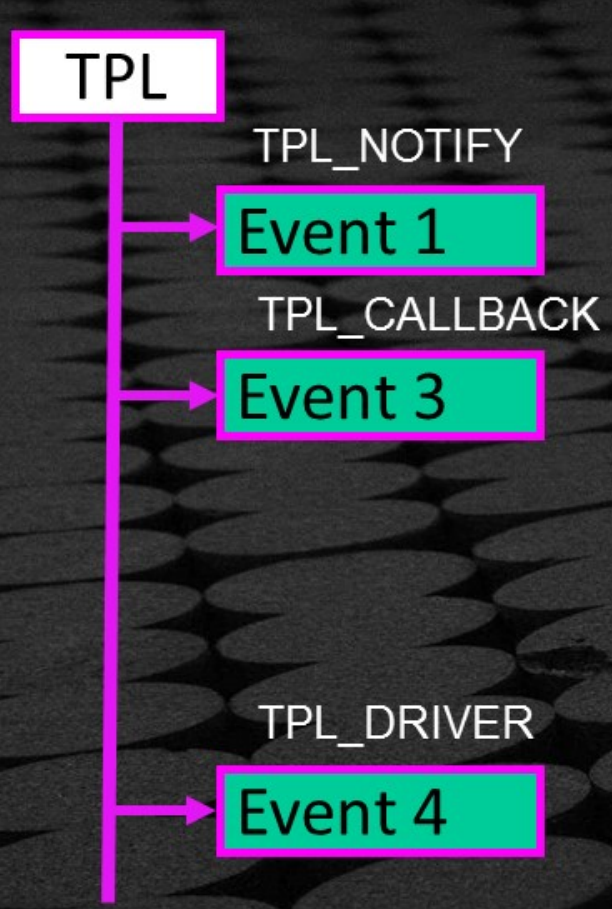
DPC Solves the TPL deadlock issue in UEFI network stack

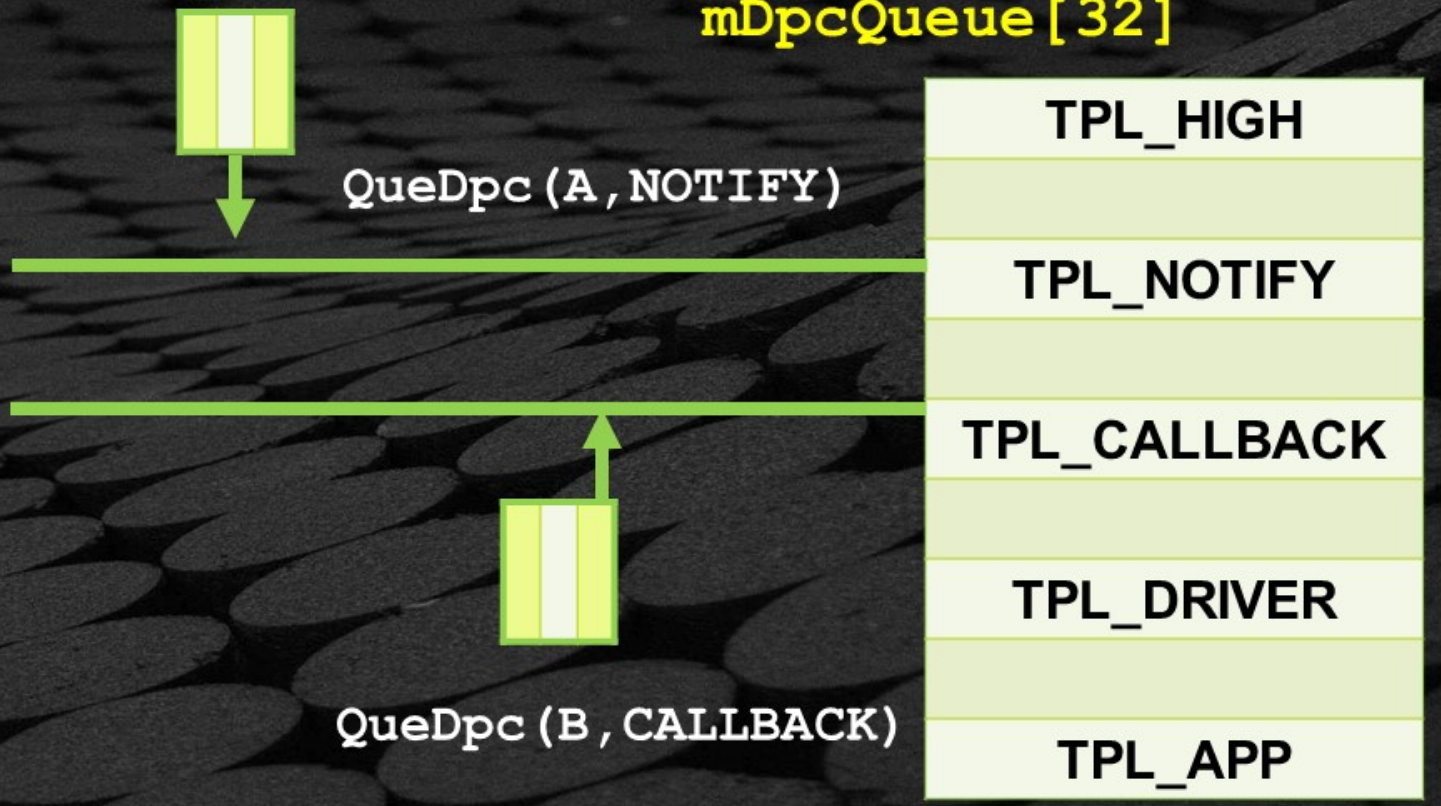
Where no packet could be delivered to the upper layer drivers

How DPC Solves the Problem

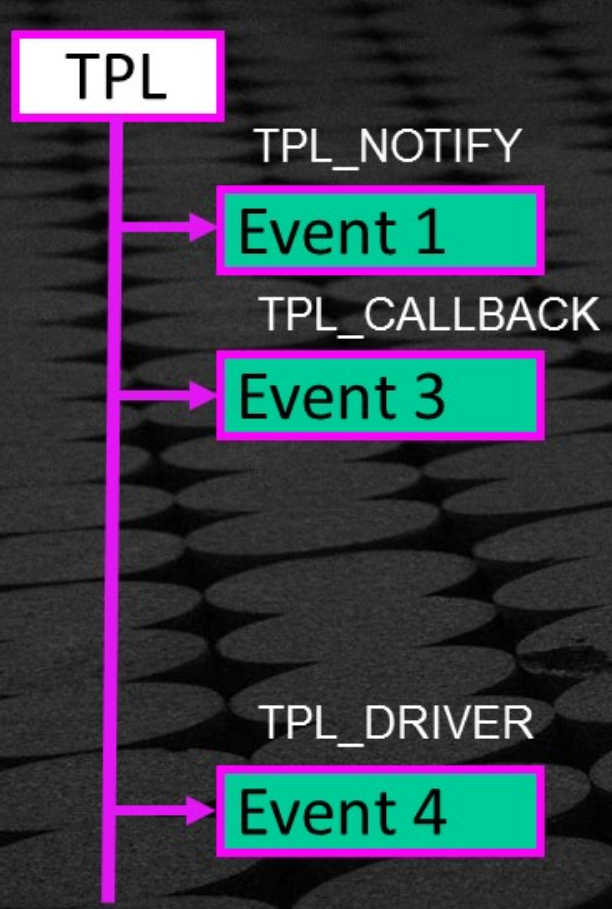


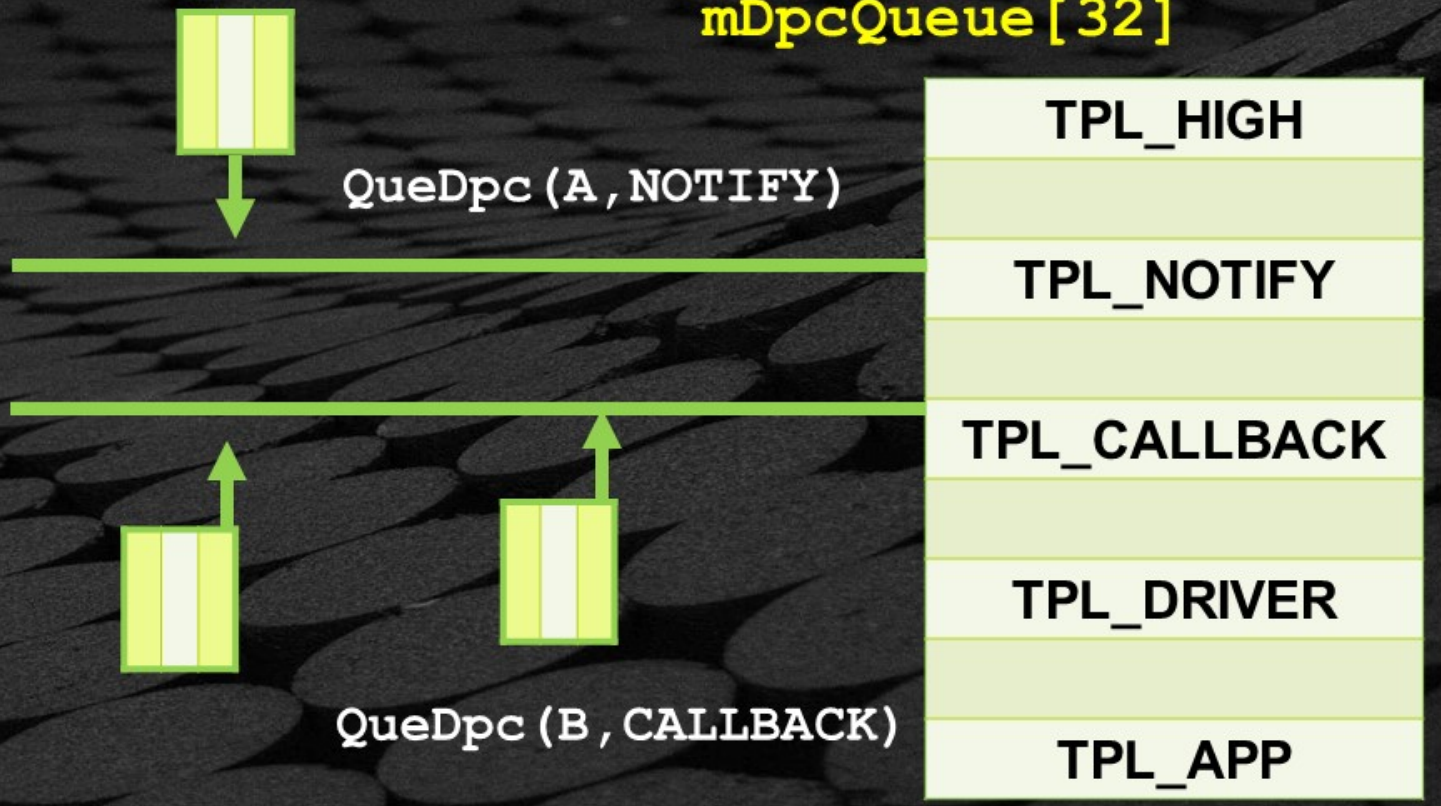
How DPC Solves the Problem



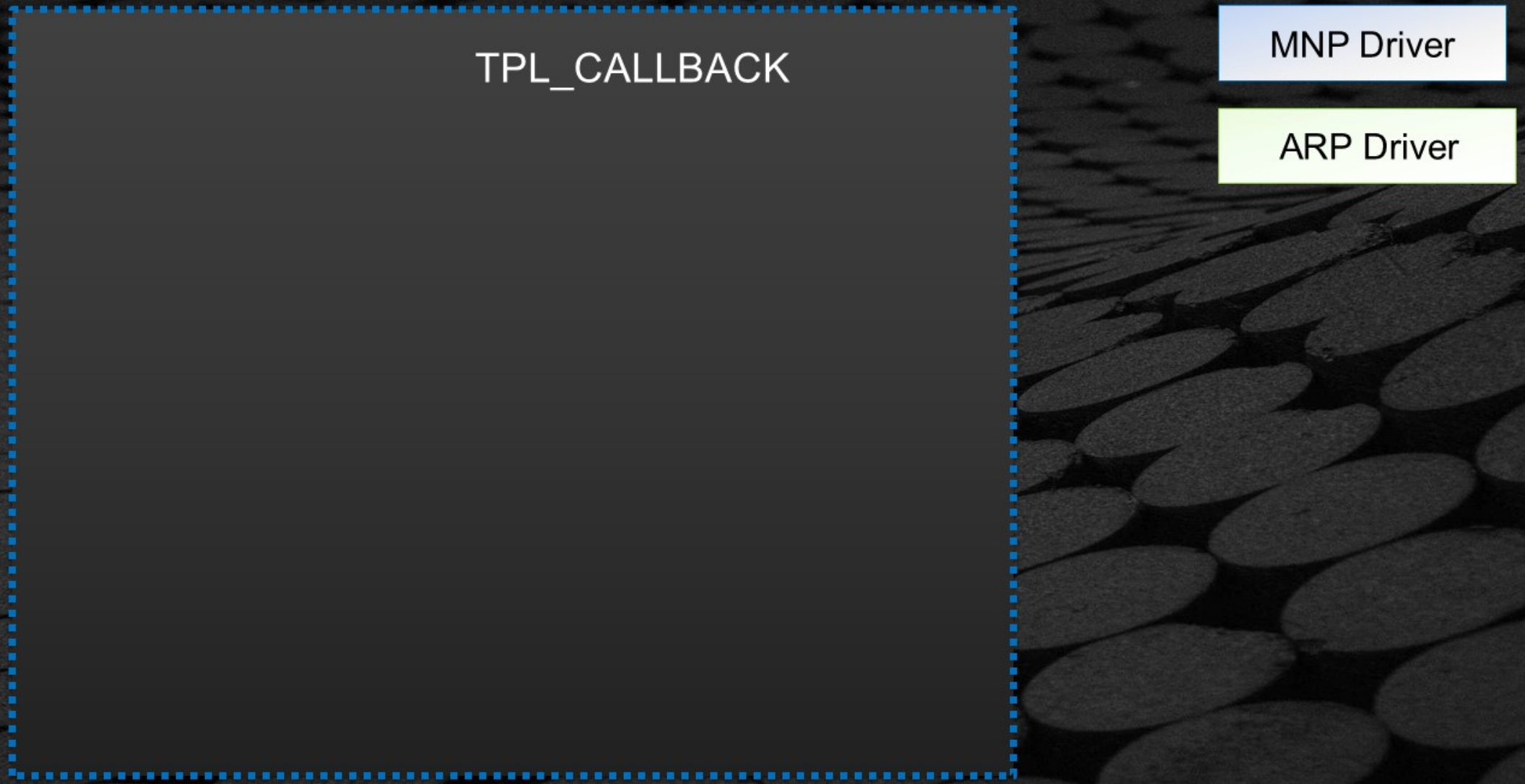


How DPC Solves the Problem

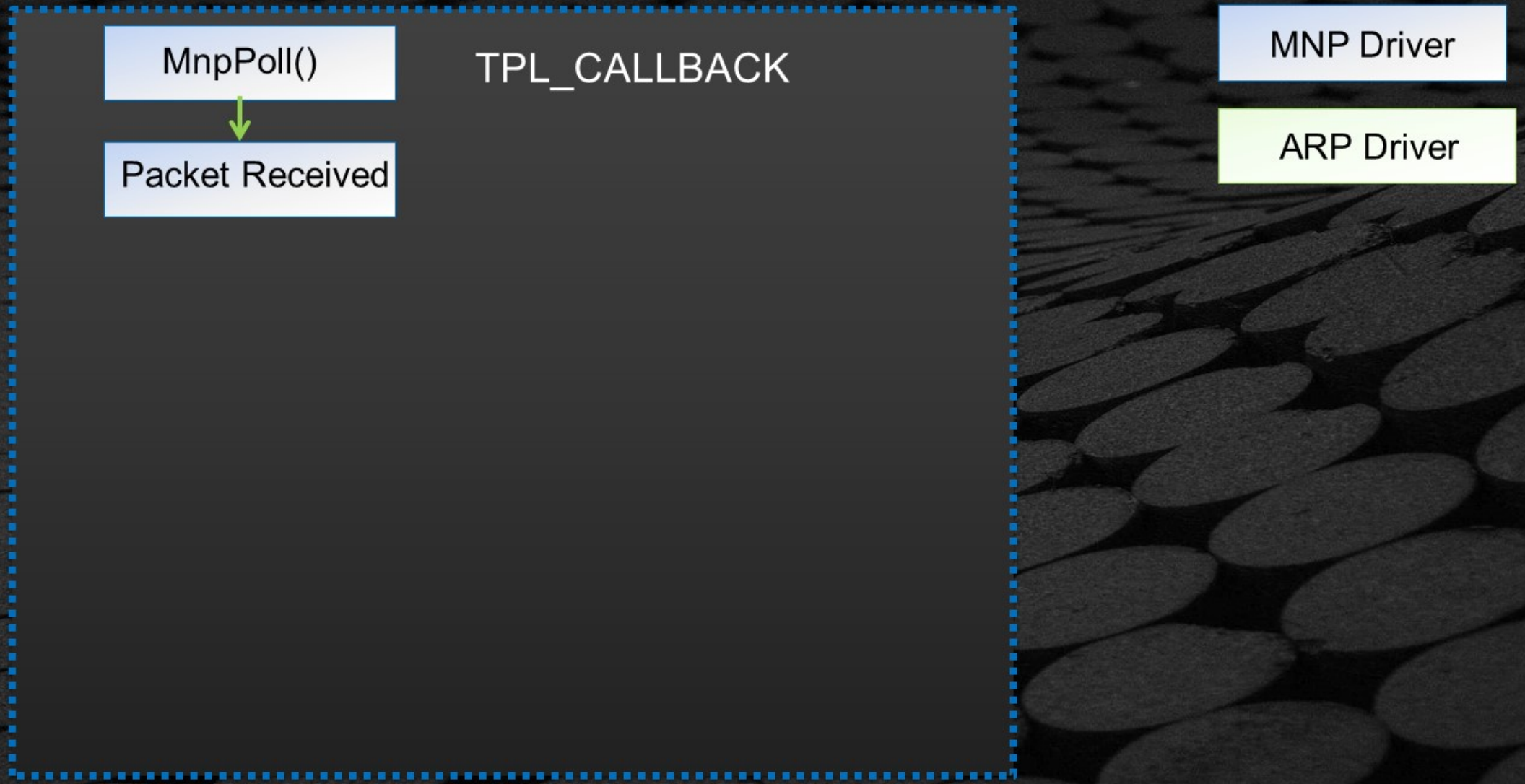




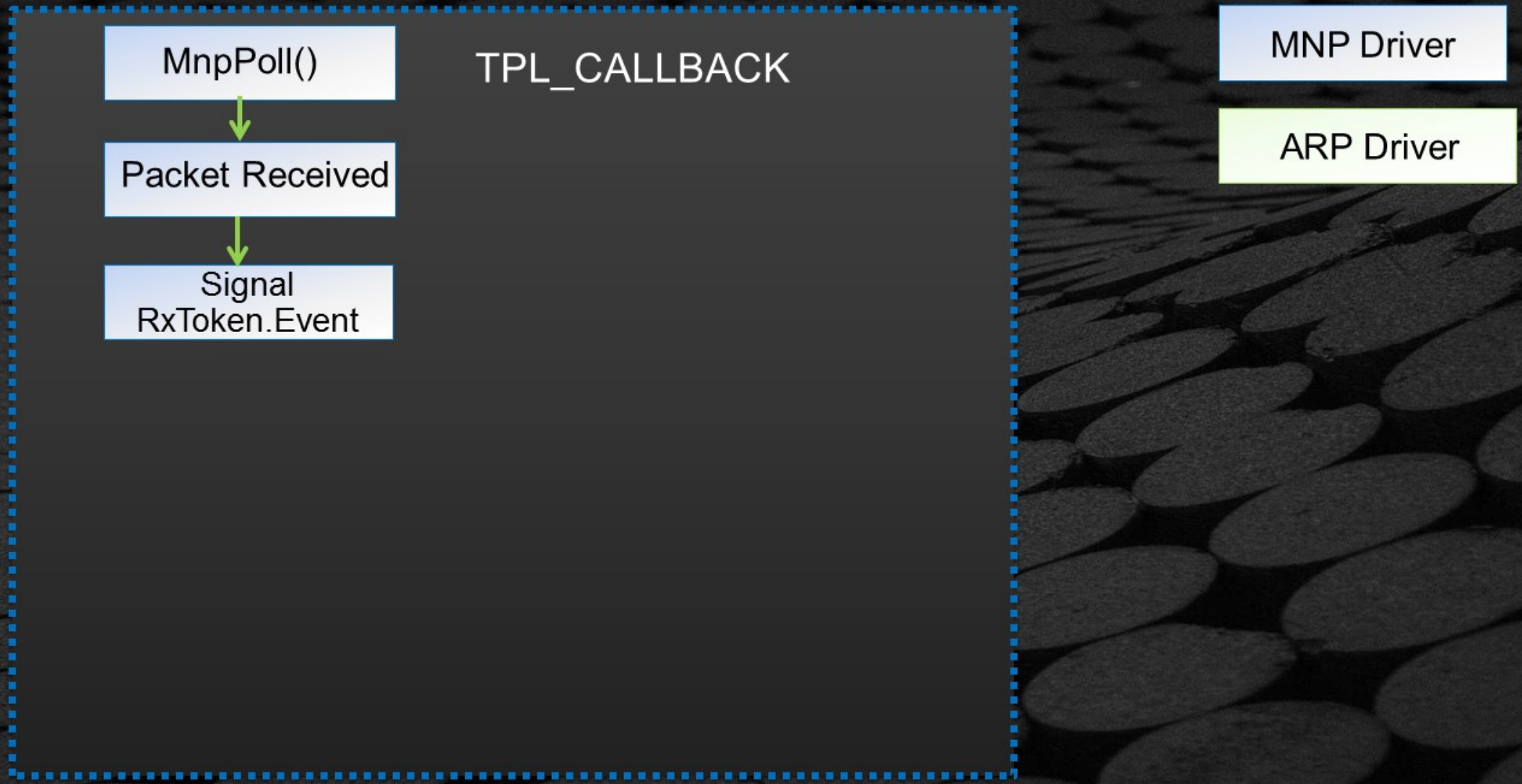
How DPC Solves the Problem - ARP



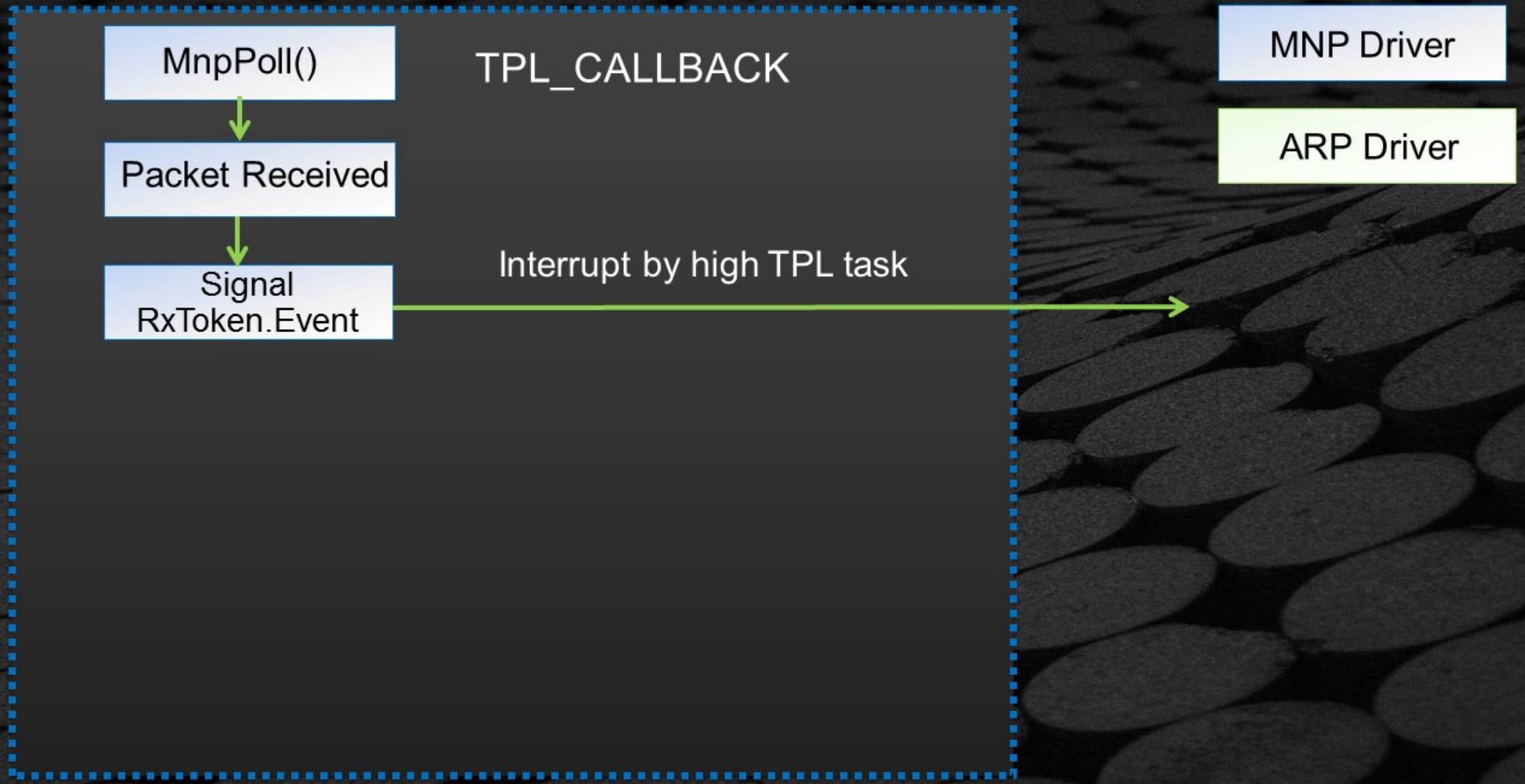
How DPC Solves the Problem - ARP



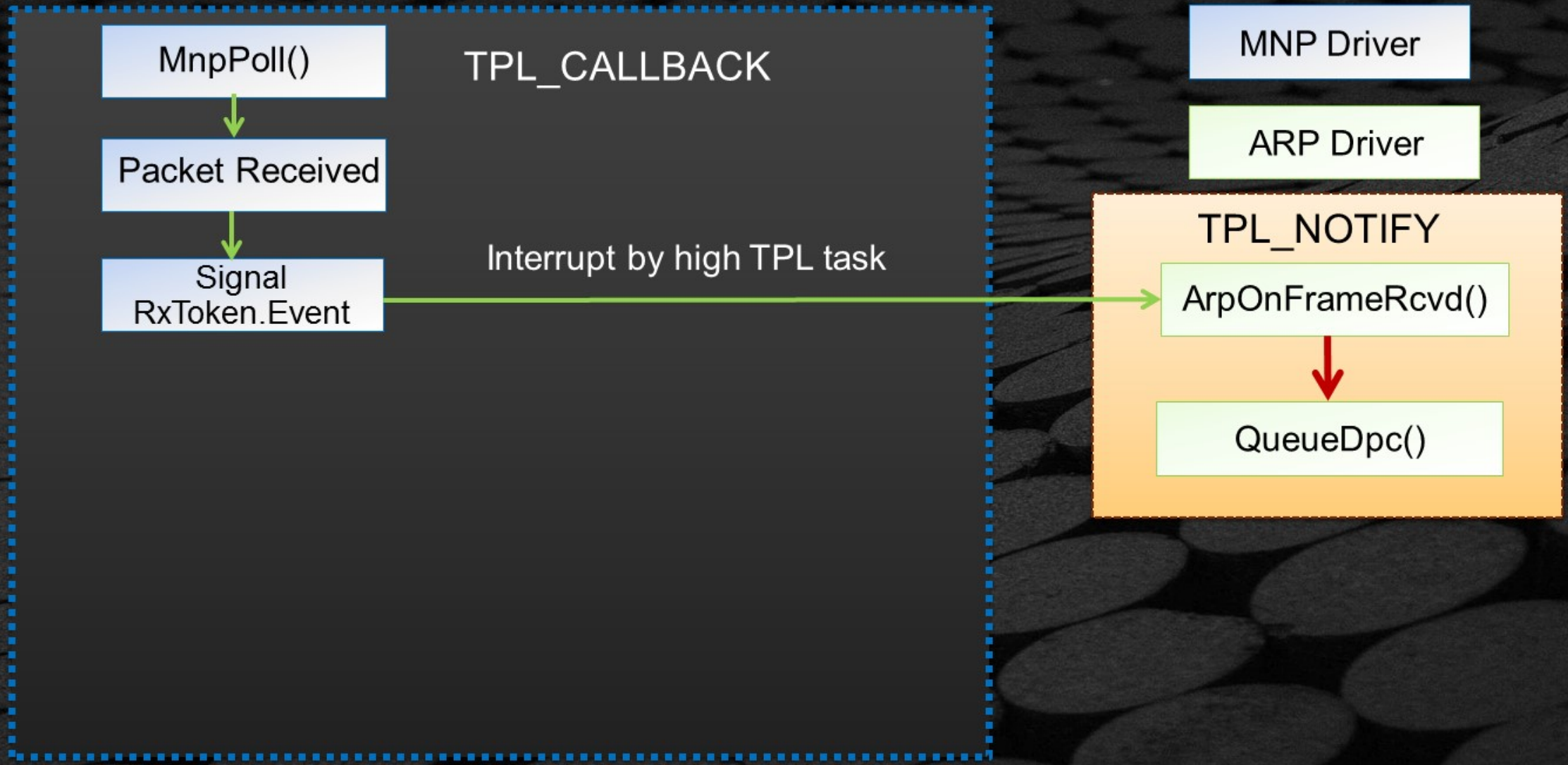
How DPC Solves the Problem - ARP



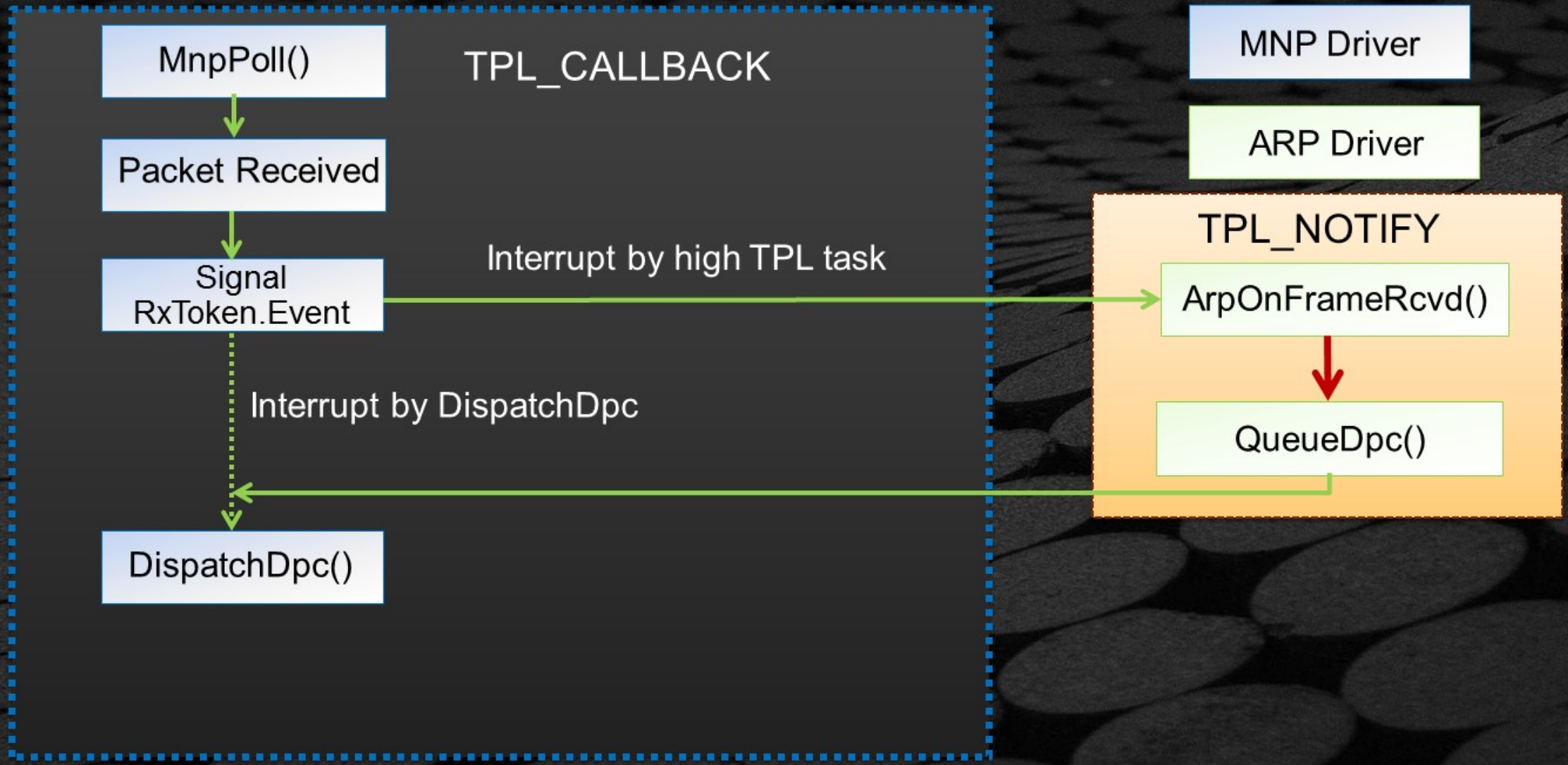
How DPC Solves the Problem - ARP



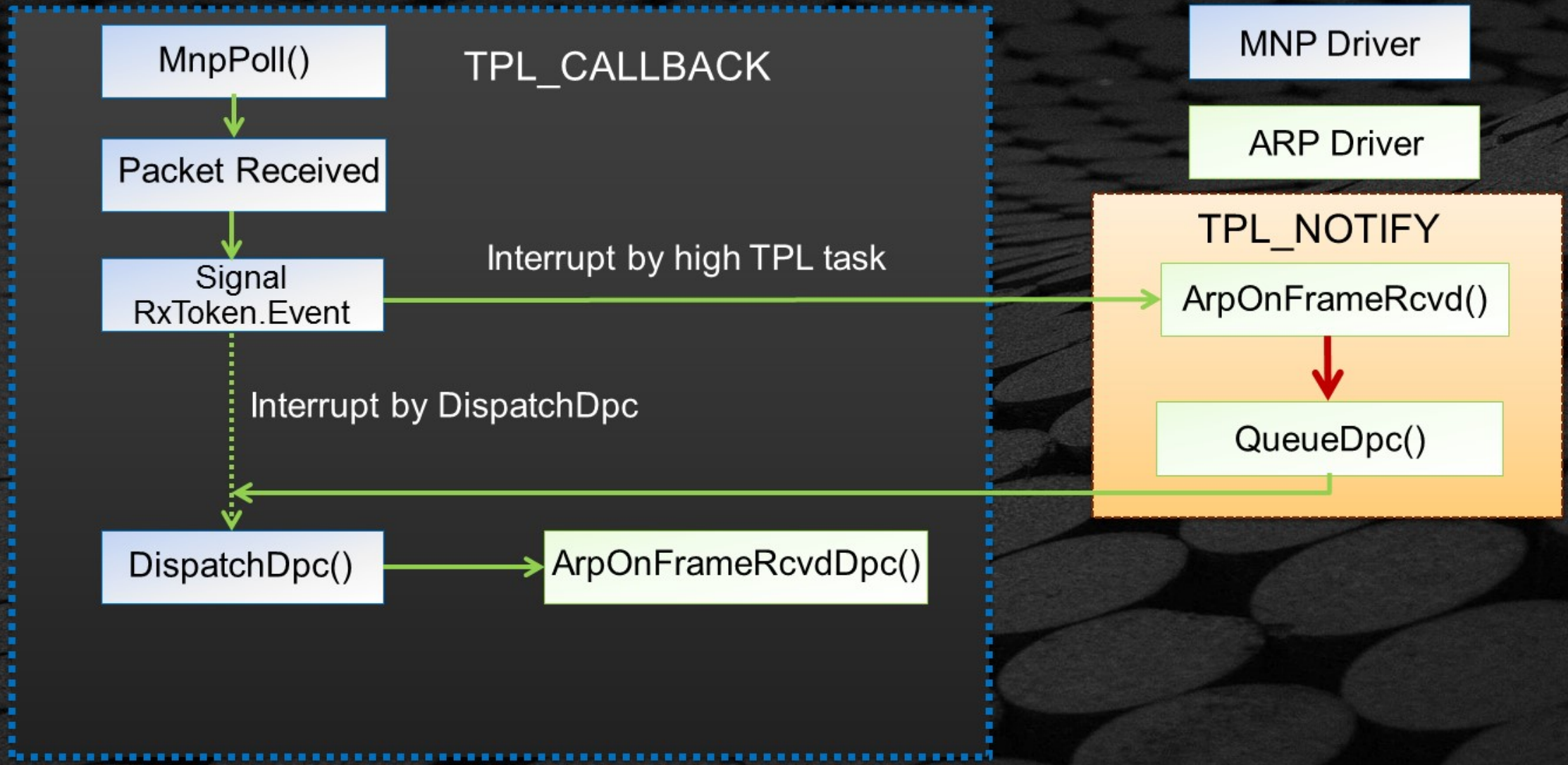
How DPC Solves the Problem - ARP



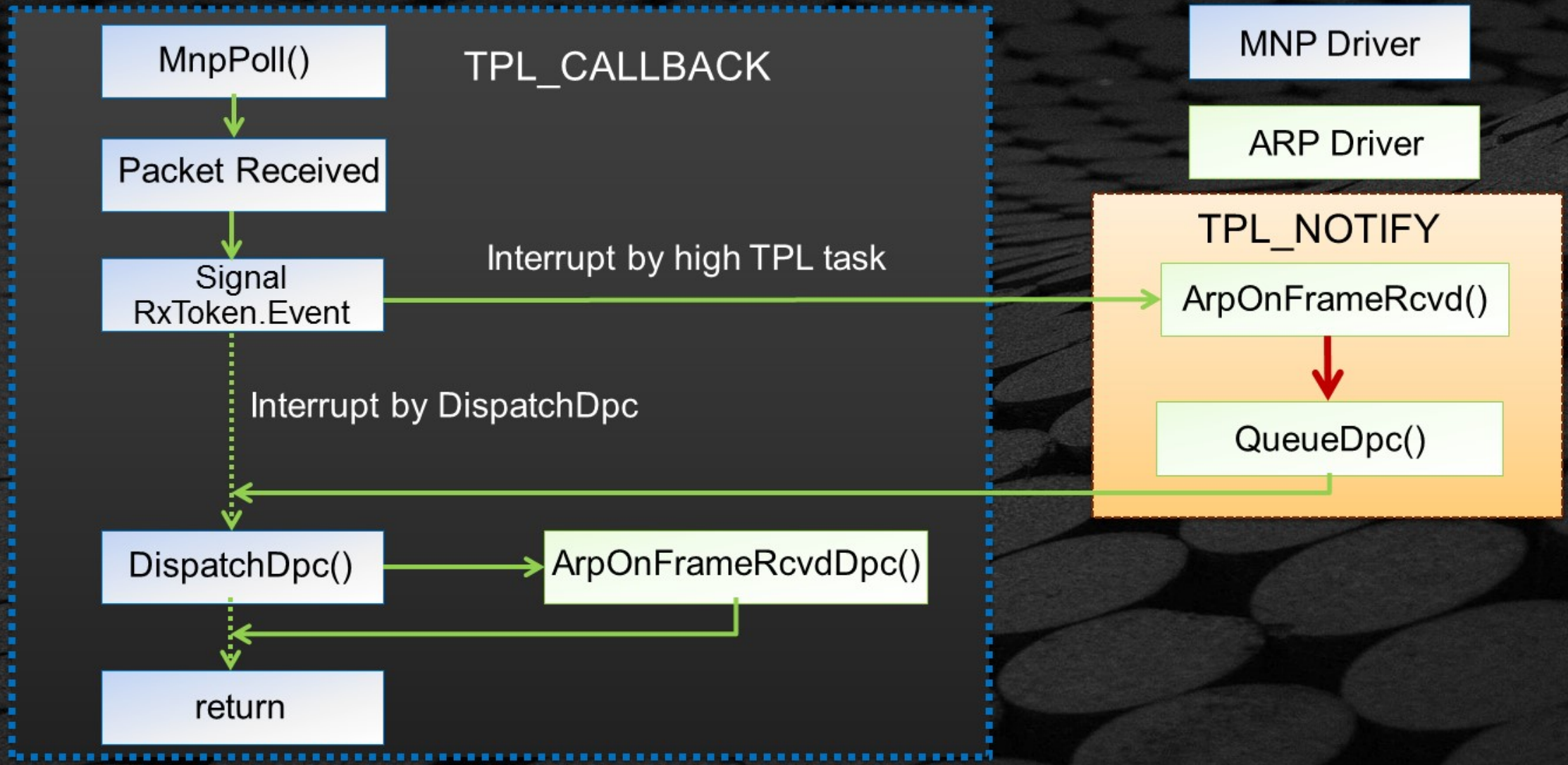
How DPC Solves the Problem - ARP



How DPC Solves the Problem - ARP

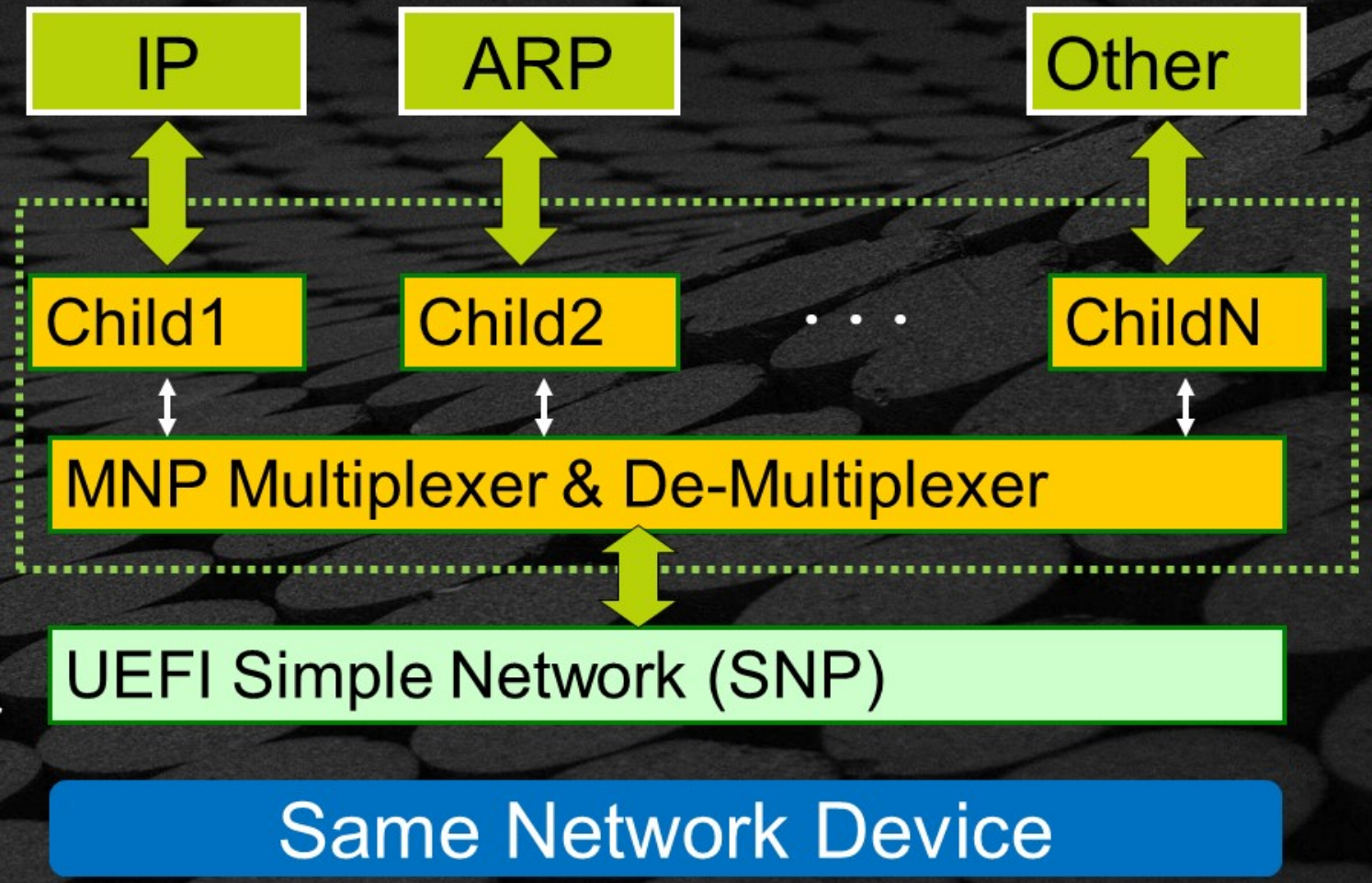


How DPC Solves the Problem - ARP



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 Managed Network Protocol (MNP)

- Transmission:
- Receiving:
- System Polling:
- Buffer management

UEFI Managed Network Protocol (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:

System Polling:

Buffer management

UEFI Managed Network Protocol (MNP)

Transmission:

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.

UEFI Managed Network Protocol (MNP)

Transmission:

Receiving:

System Polling:

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

UEFI Managed Network Protocol (MNP)

Transmission:

Receiving:

System Polling:

Buffer management

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

UEFI Managed Network Protocol (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.

tianocore

IPv4 Driver – in Detail

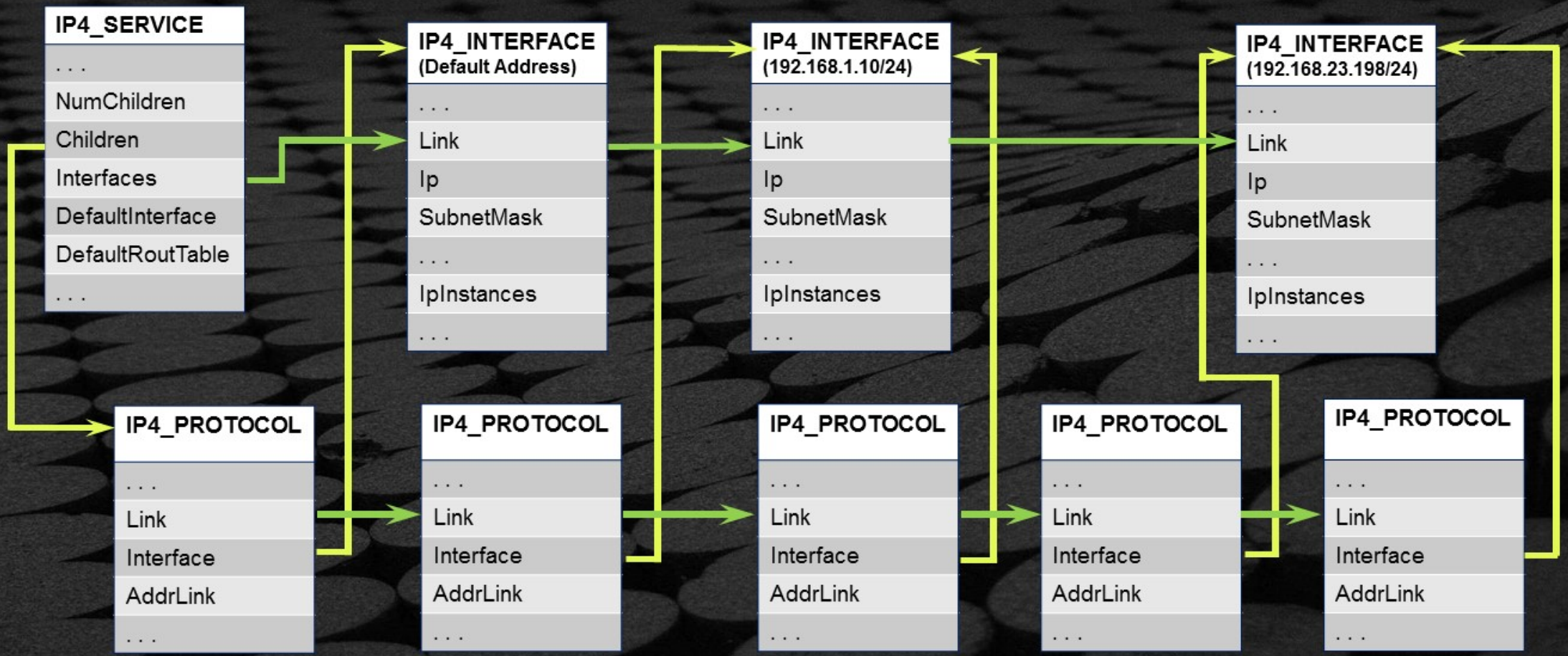
Source	Description
Ip4Common.c	Common helper routines for the whole driver
Ip4Driver.c	DriverBinding and ServiceBinding routines
Ip4Icmp.c	Internet Control Message Protocol ICMP related routines
Ip4If.c	IP pseudo interface related routines
Ip4Igmp.c	Internet Group Management Protocol IGMP related routines
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

IPv6 Driver – in Detail

Source	Description

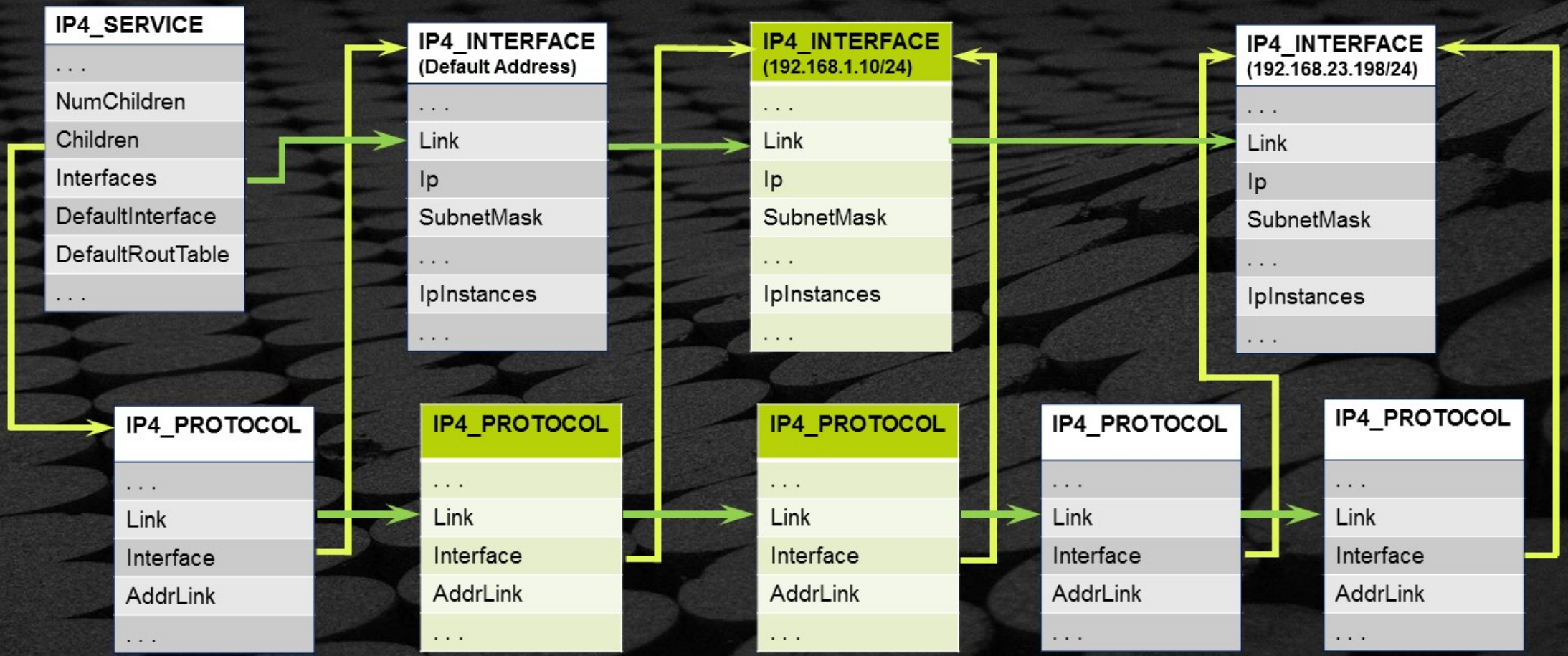
```
Ip6Common.c     Common helper routines for the whole driver
Ip6Driver.c     DriverBinding and ServiceBinding routines
Ip6Icmp.c       Internet Control Message Protocol ICMP related routines
Ip6If.c         IP pseudo interface related routines
//   Different from IPv4
Ip6Nd.c         Neighbor Discovery 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
```

IP Internal Structure – Example IPv4

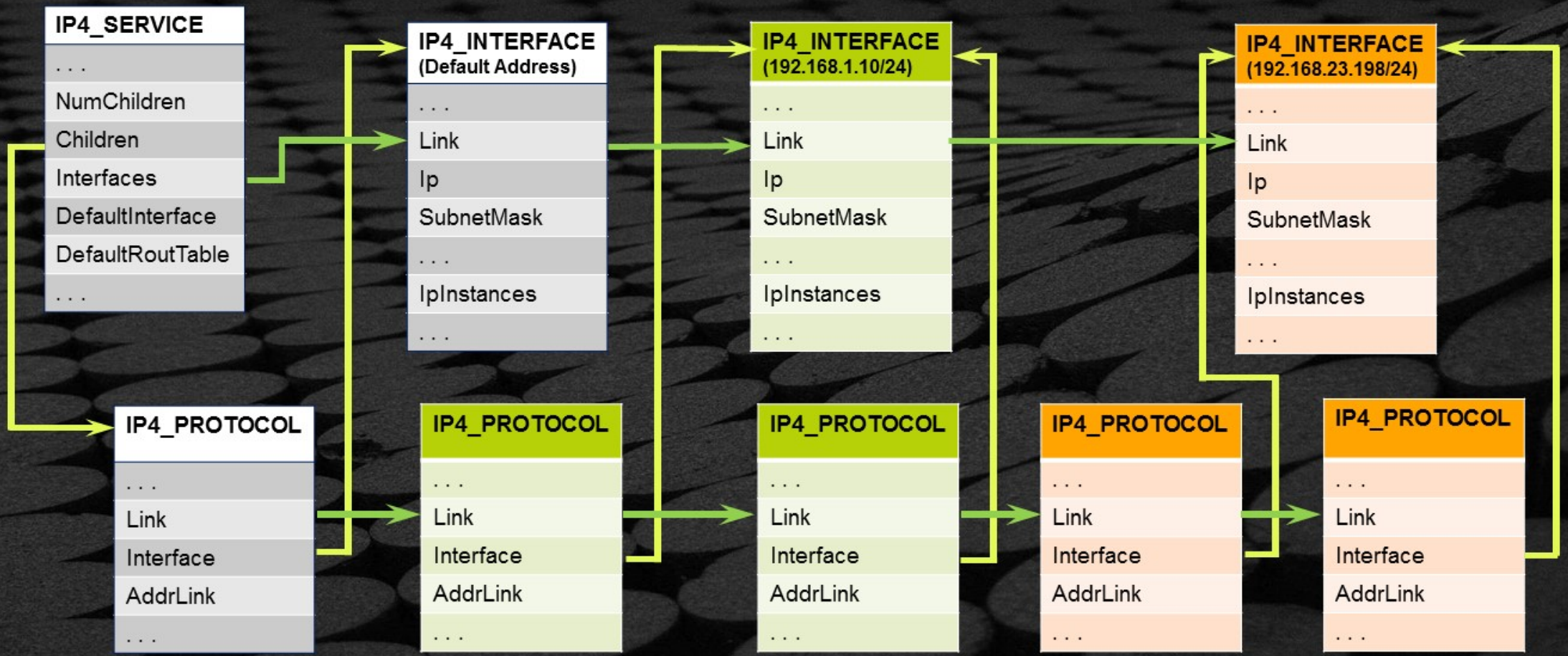


tianocore

IP Internal Structure – Example IPv4

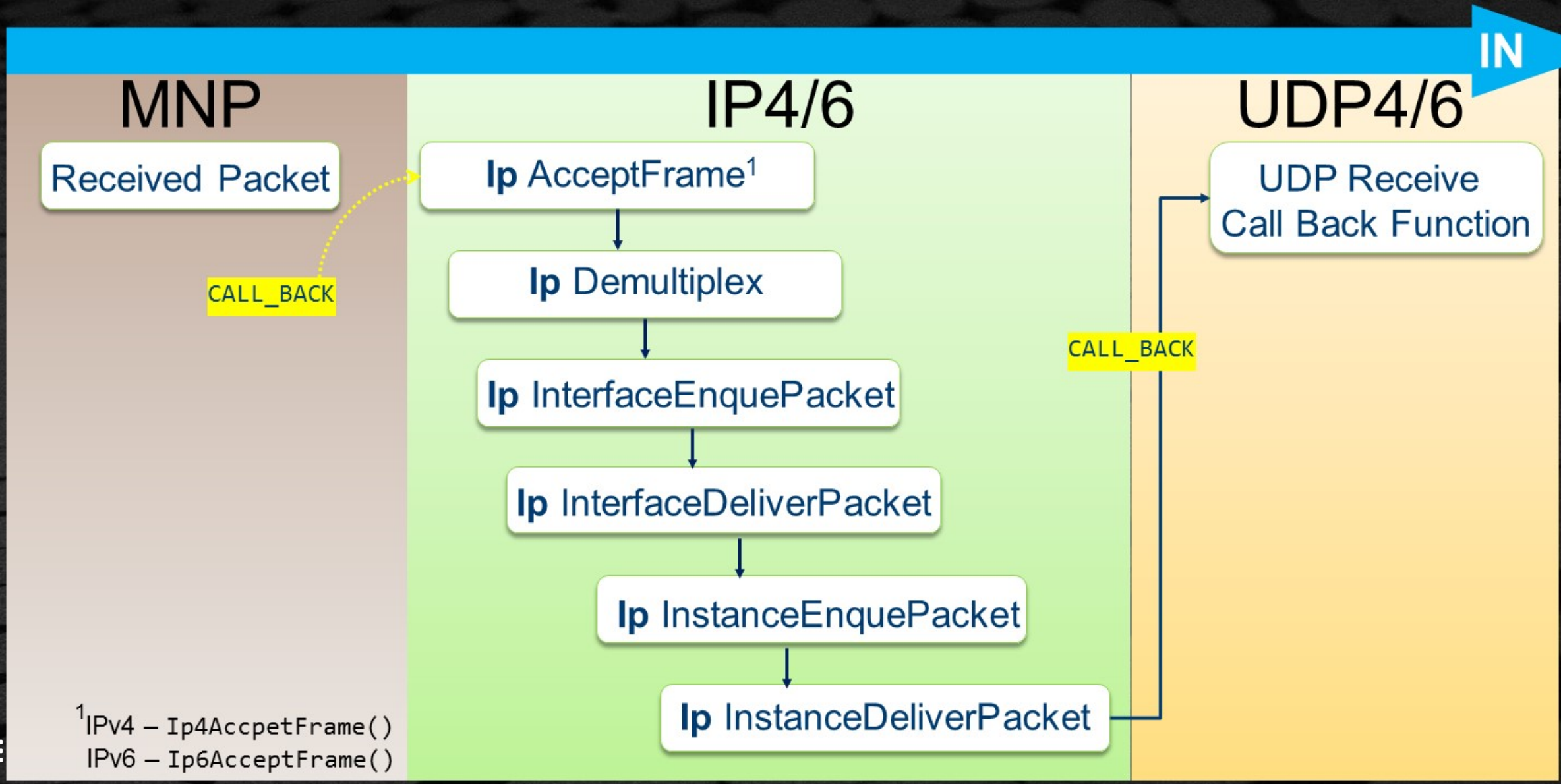


IP Internal Structure – Example IPv4



tianocore

IP4/6 Input Workflow



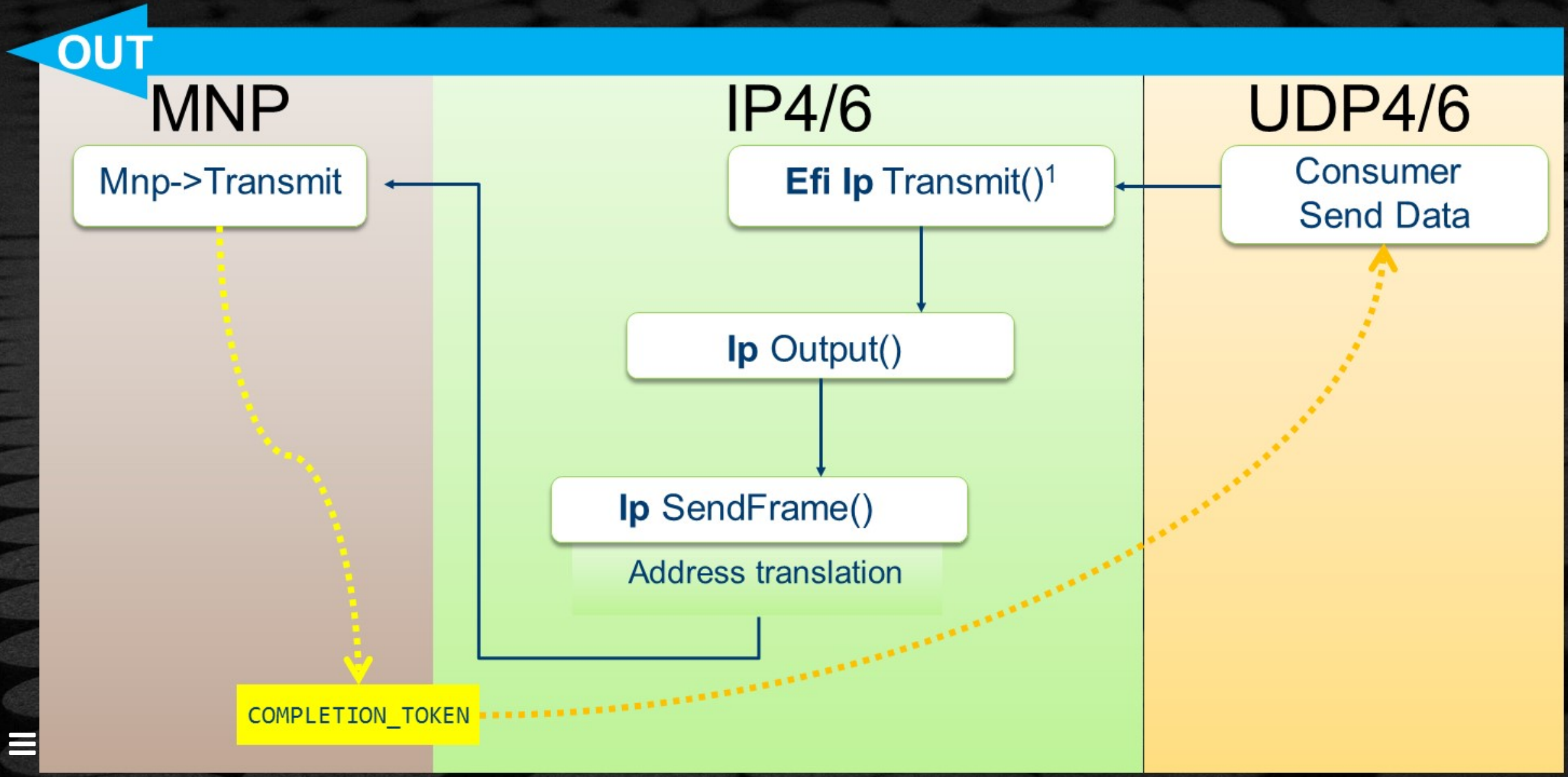
tianocore

IP4/6 Input Workflow - Details

```
Ip4ReceiveFrame ()
  - Request to receive the packet from
    procedure related information into
    completion of the receive, Ip4OnFra
    Ip4OnFrameReceivedDpc (). After som
    CallBack function provided will be
    information. Normally, the CallBack
Ip4AcceptFrame ()
  - Most work of processing a received
   - Sanity check: version number, chec
   - Reassemble fragments of an big IP
   - Invoke IGMP or ICMP processing if
     respectively, else invoke Ip4Demul
     packet to any IP4 instance who is
 - the receive procedure by calling Ip4
Ip4Demultiplex ()
  - Iterate all the Ip4 Interfaces, and
    to let every Ip4 Interface check wh
    this IP4 packet matches their recei
    else, do nothing.
  - If any IP4 interface is willing to
    the IP4 interfaces and call Ip4Inte
```

```
Ip4InterfaceEnquePacket ()
- Match cast type first, such as, Multi
- Iterate all the Ip4 instances belongi
  Ip4InstanceEnquePacket () to check wh
  interested in this packet.
Ip4InterfaceDeliverPacket ()
- Iterate all the IP4 instances and cal
  to deliver any queued packets of the
Ip4InstanceEnquePacket ()
- Check whether this instance is willir
  IP packet. If all match rules are pas
  and put it into the Received list.
Ip4InstanceDeliverPacket ()
- Deliver all the queued IP packets to
  protocol by every time putting an IP
  EFI_IP4_COMPETION_TOKEN, and finally
  The end condition is either there is
  provided token.
- The consumer provided tokens for rece
  callings to EFI_IP4_PROTOCOL.Receive
```

IP4/6 Output Workflow



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

IP4/6 Output Workflow - Details

```
EfiIp4/6Transmit () – EFI_IP4/6_PROTOCC
  - The initial source of most IP4/6 ou
  - Sanity check.
  - Append IP4/6 head to the buffer pas
  - Record the Token and the correspond
    (IP4/6_TXTOKEN_WRAP)  to the TxToke
  - Call Ip4/6Output to output this pac
Ip4/6Output ()
  - Decide the IP address of the next h
    packet, aka, route selection.
  - Fragment it if needed.
  - Call Ip4/6SendFrame () to send this
```

```
Ip4/6SendFrame ()
  - Do layer-3 to layer-2 address trans
    need the help from the Arp instance
    interface selected.
     - Immediately send out the packet
       translation can be done directly
     - Or create an ArpQue which assemb
       the same next hop. Once the trar
       packets are transmitted out in t
Upon completion of the transmit, either
failure, the status will be returned to
EFI_IP4/6_COMPLETION_TOKEN.Status with
and at this time, the control of the to
```

tianocore

UEFI HTTP(S) BOOT OVERVIEW

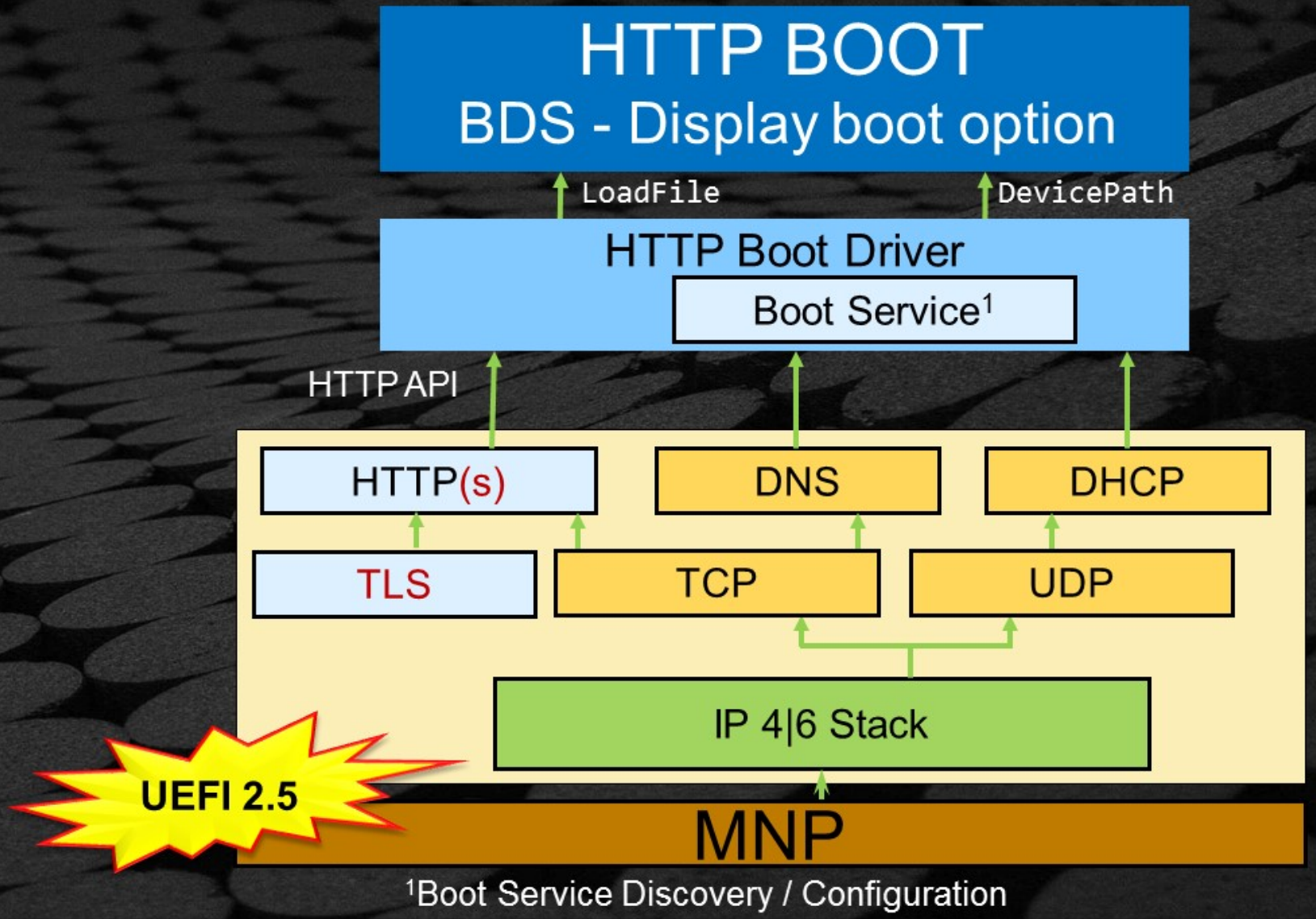
UEFI HTTP(s) Boot Overview

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

HTTP(s) Boot UEFI Network Stack

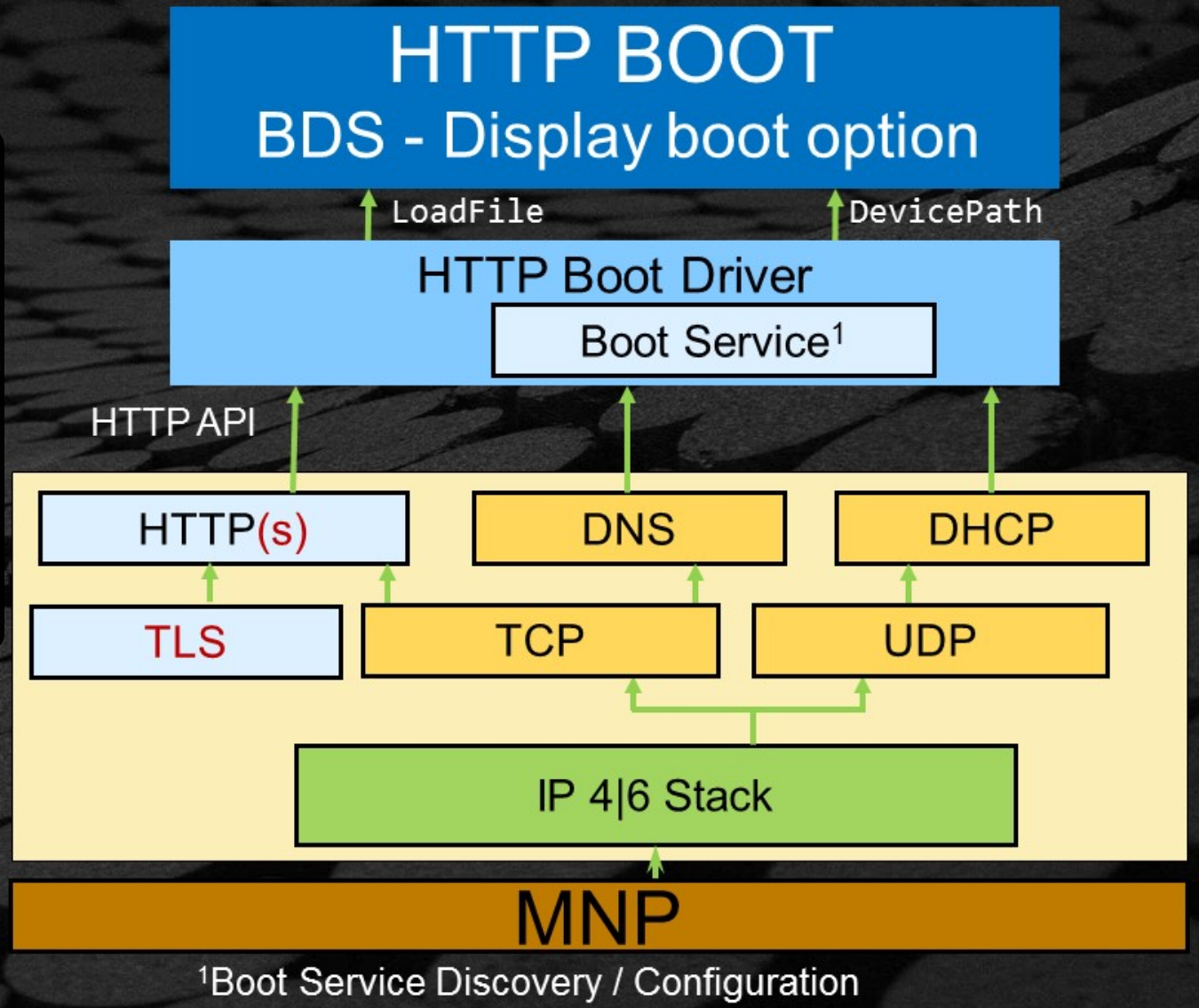
- DNS IPv4 / IPv6
- HTTP IPv4 / IPv6
- TLS for HTTPs
- HTTP Boot Driver



HTTP(s) Boot UEFI Network Stack

- UEFI & EDK II Protocols

```
- 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
```



tianocore

UEFI Native HTTP Boot

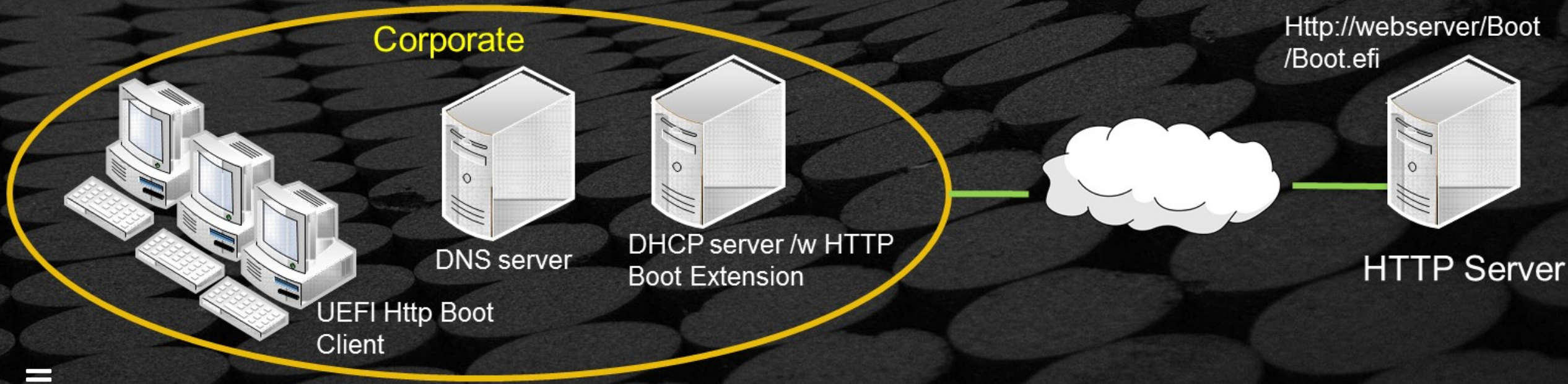
- Corporate Environment

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)

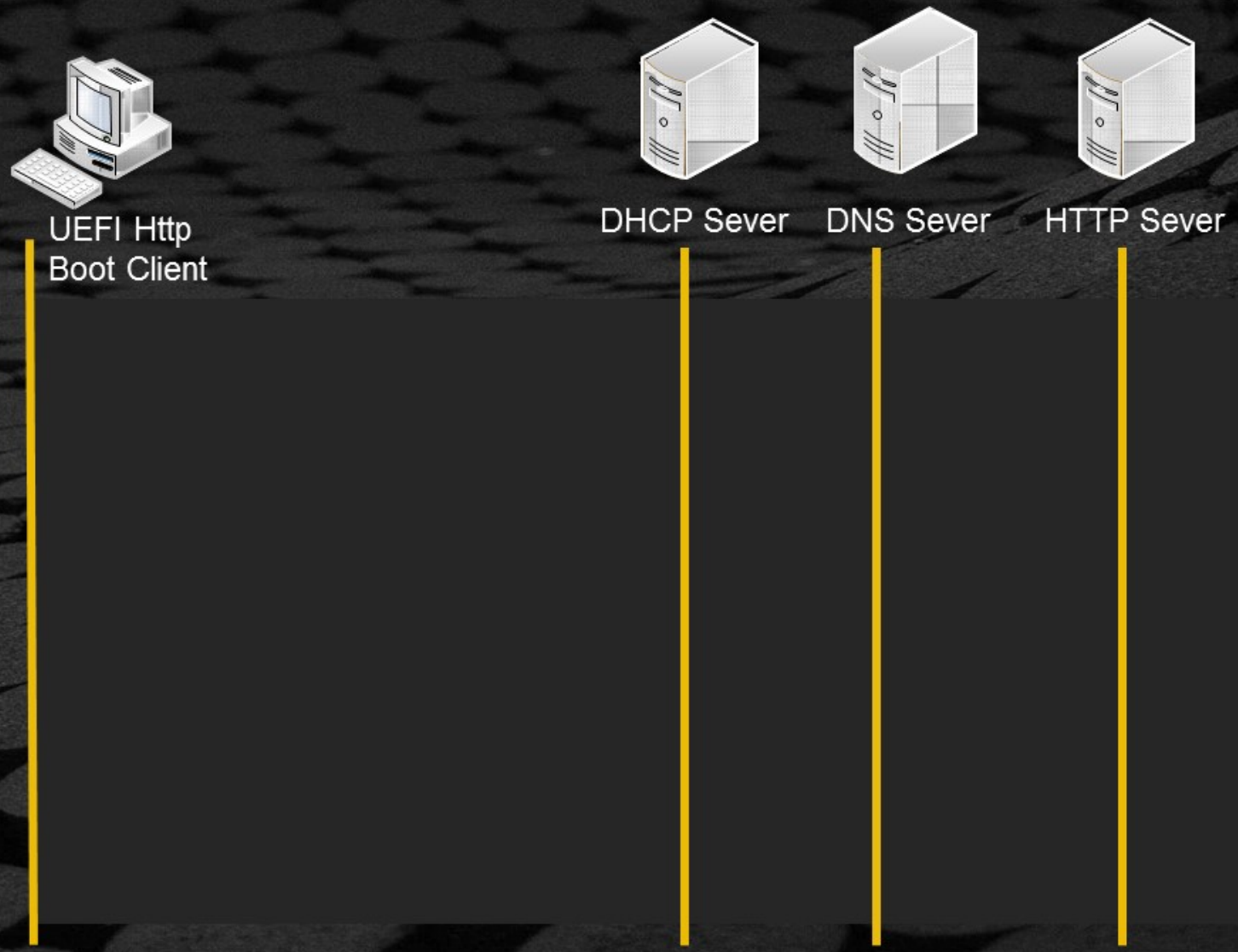
Addresses PXE issues

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



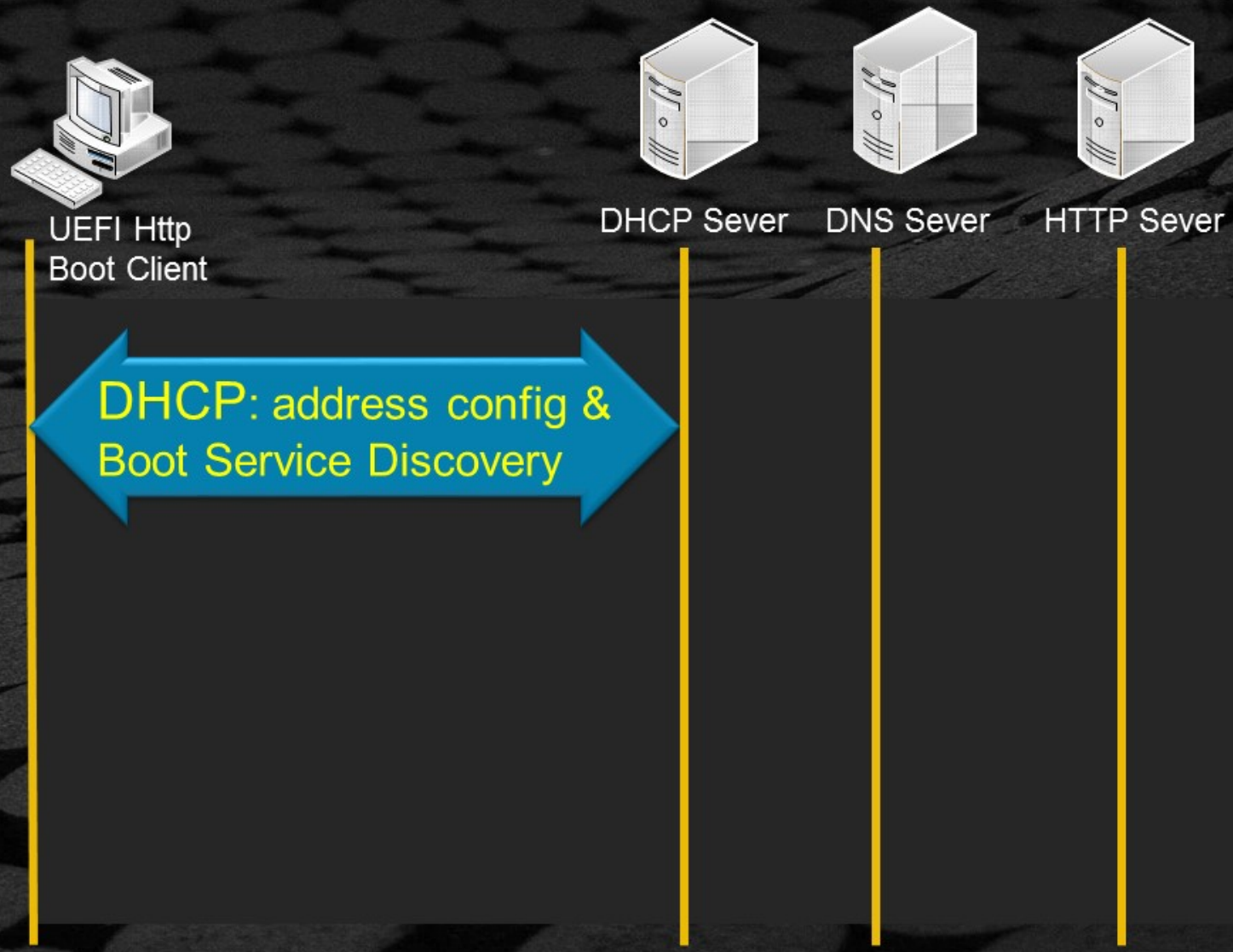
HTTP Boot DHCP Discovery

- 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)



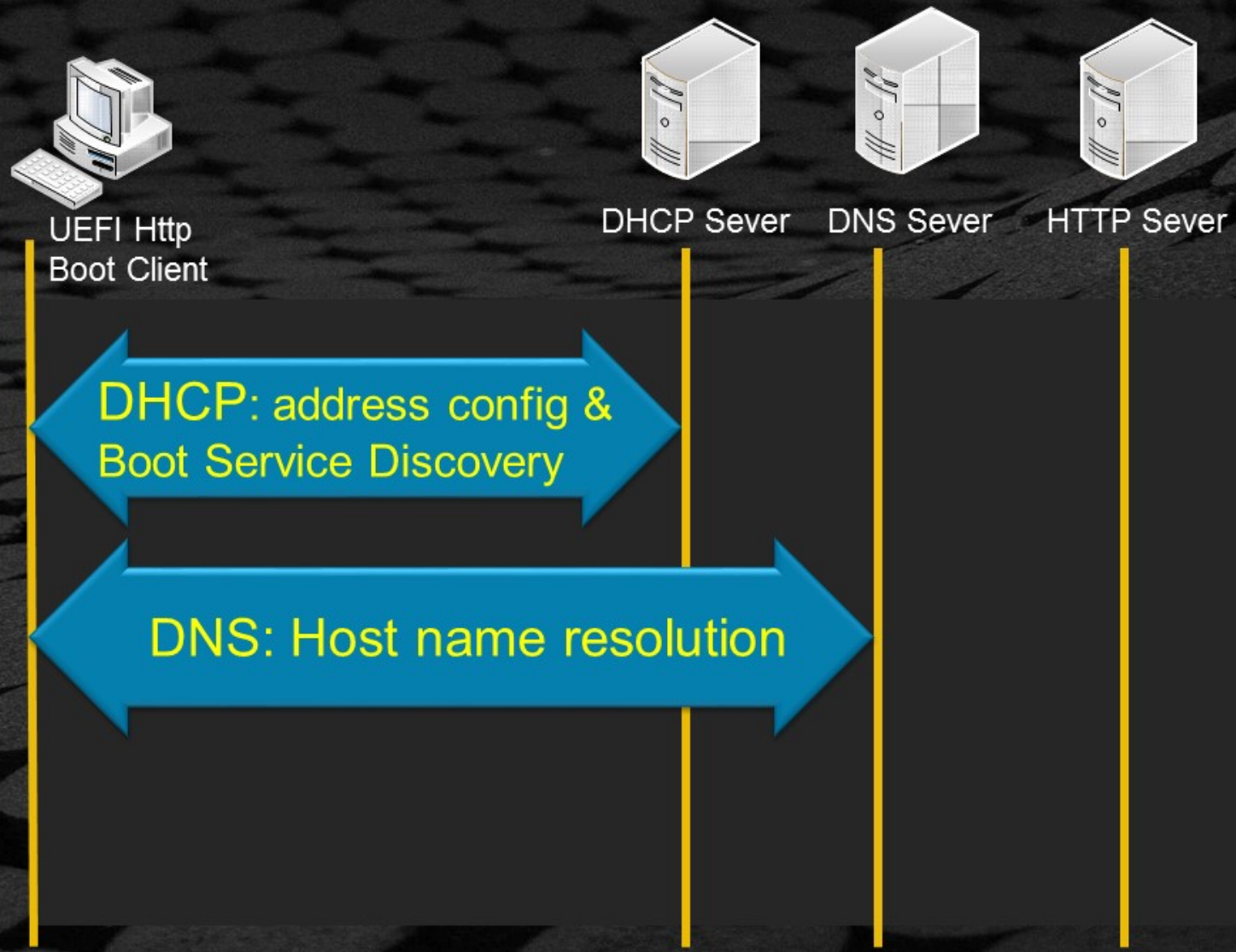
HTTP Boot DHCP Discovery

- 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)



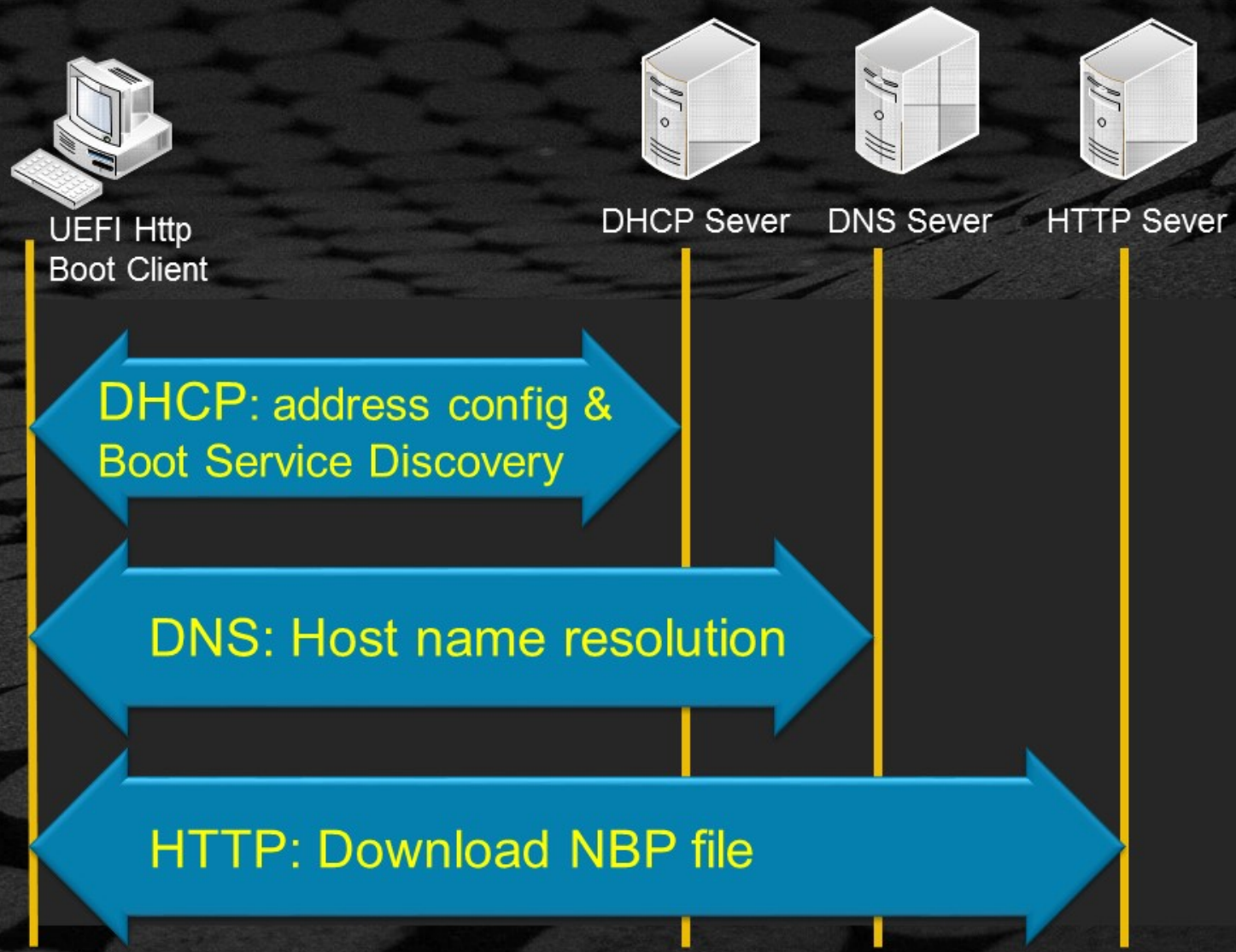
HTTP Boot DHCP Discovery

- 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 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 - 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_Shiffett]
0x00 0x16	arm 64 uboot	[Joseph_Shiffett]

tianocore

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: ipxe.org/Uefihttp

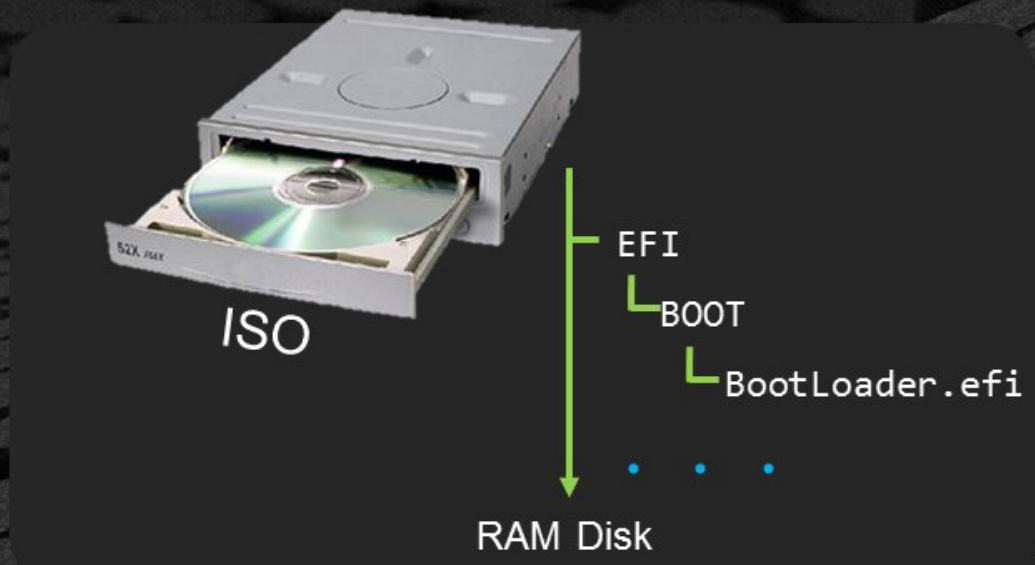
2 Options to address the PXE challenges:

Native UEFI HTTP Boot and iPXE using UEFI HTTP

tianocore

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

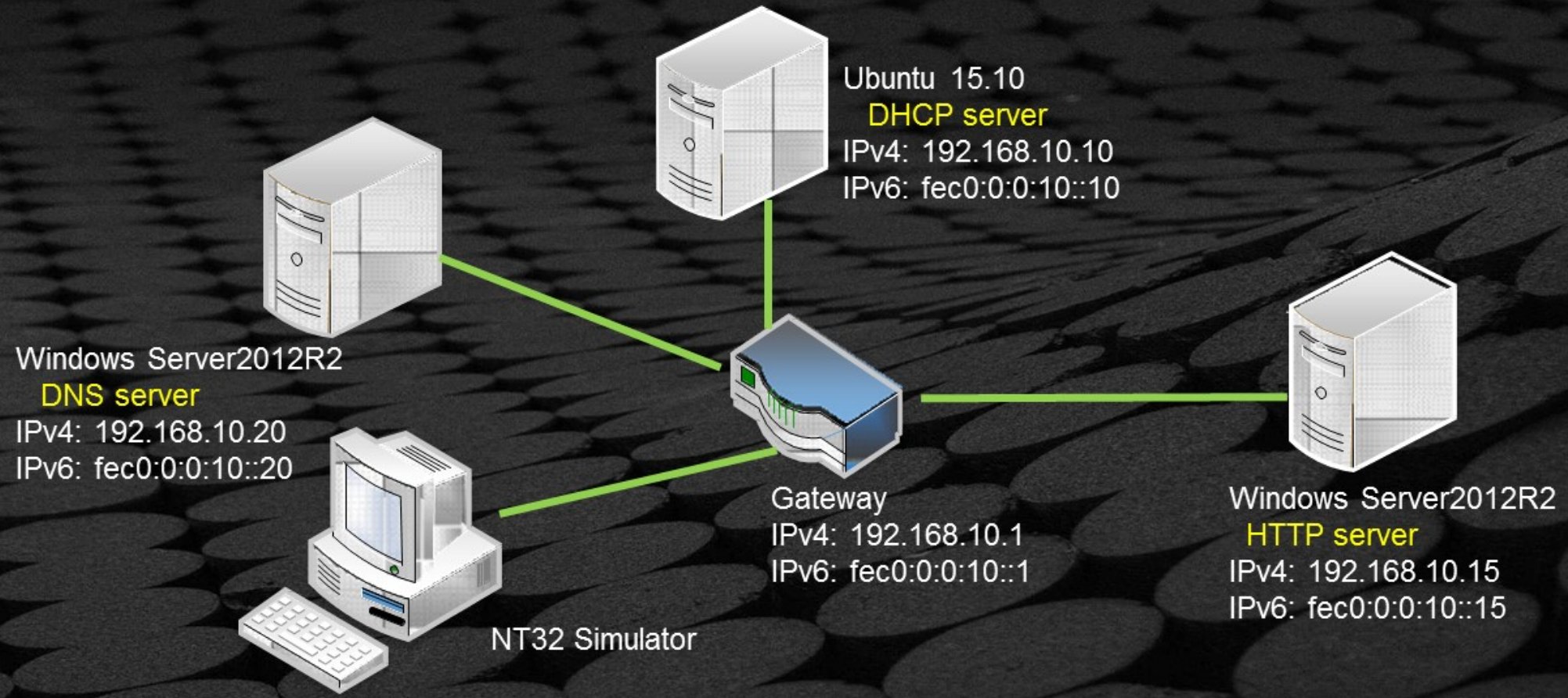


Feature Enabling:

Add Edk2 `RamDiskDxe.inf` to Platform .DSC

tianocore

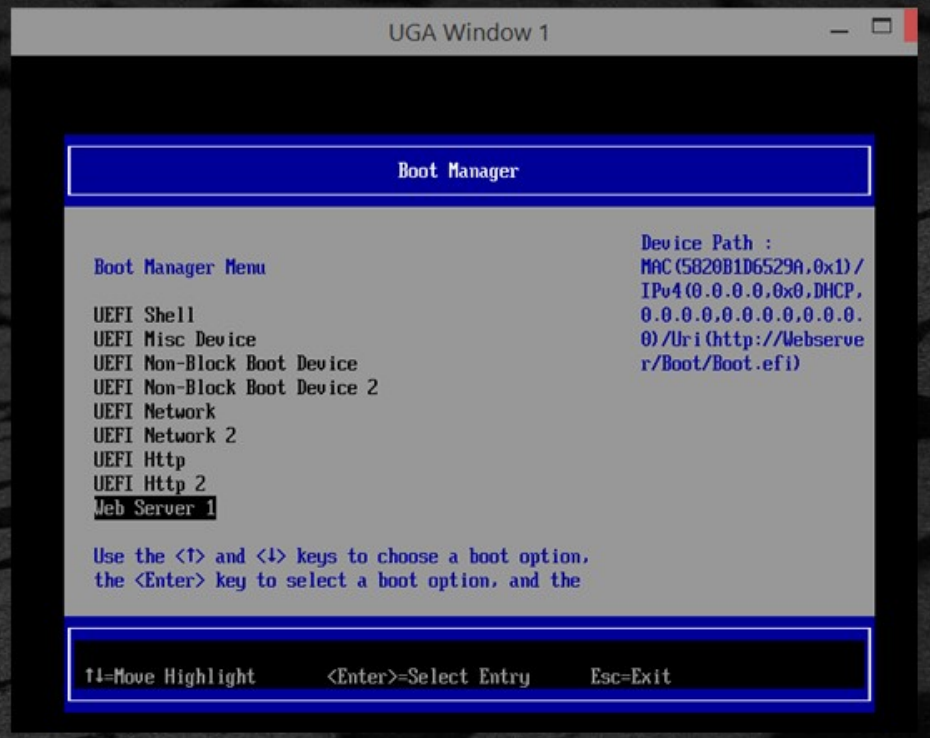
UEFI Http Boot Example



White paper EDK II HTTP Boot Getting Started Guide for a step by step guide

EDK II HTTP Boot Configuration

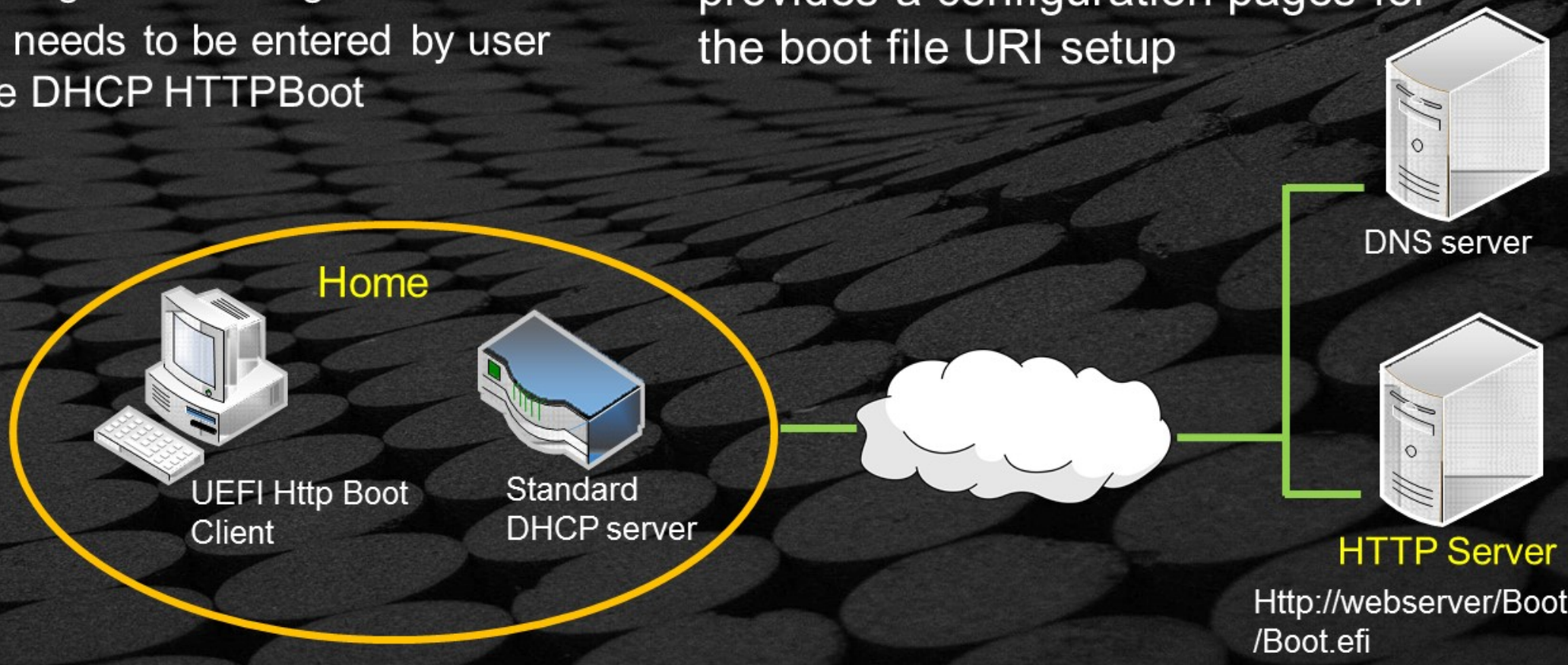
In the main page of Boot Manager Menu, enter [Device Manager] → [Network Device List] → Select a NIC device → [HTTP Boot Configuration], set the HTTP boot parameters such as the boot option title, IP start version and the URI address



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



tianocore

Getting Started Guides

HTTP:

Wiki Page

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

PDF EDKIIHttpBootGettingStartedGuide_0_8.pdf

HTTPS:

Wiki Page

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

PDF UEFI HTTPS Boot on EDK II .pdf

tianocore

SUMMARY

- UEFI Network Stack Layers
  host and target basic configuration and components
- EDK II Network Features Overview
- What UEFI Protocols Make Network Work in EDK II
- UEFI HTTP(s) Boot Overview


Questions?

tianocore

ACKNOWLEDGEMENTS