NVMe over Fabric Architecture & Functional Model

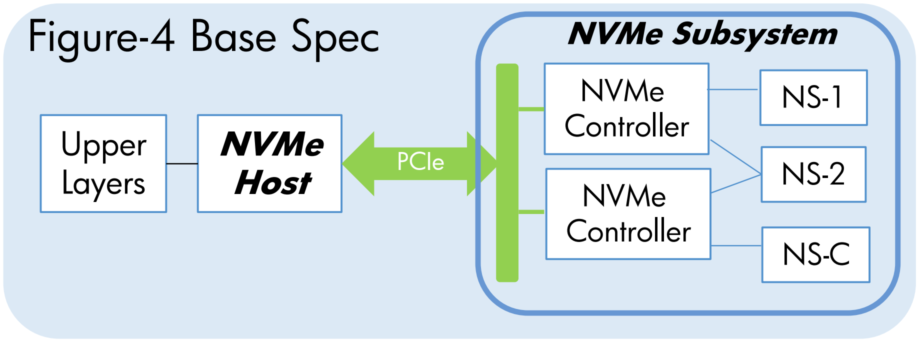
15-327v1

Siamack Ayandeh
Chief Architect
HP Networking
September, 2015
Siamack@HP.Com

NVMe over Fabric/FC -- Why

- SSD having generated significant TCO reductions as local storage has found its way into storage arrays
- Storage arrays add benefits of software intelligence including:
  - Deduplication/real-time Compression/thin provisioning,
  - High availability of RAID
- Flash is also about performance:
  - Offers advantage for comparable economy vs. HDD
  - Measurements for local flash using NVMe/PCI-e report reducing latency by half vs. SCSI/SAS
- NVMe standard is expected to have a 10+ year life span, during which time memory developments will reduce NVM access latencies to sub microseconds
  - Therefore fabric latency needs to be kept in check
  - Standards should avoid the temptation of TTM if it cripples the technology long term
- In order to reduce latency for NVM workloads fabrics need to:
  - Take advantage of higher port speed interconnects, &
  - Exploit parallelism offered by fabrics by definition

NVMe Local over PCI-e vs. over Fabric



- Figure shows NVMe Subsystem connected with one or more controllers via PCI-e ports – defined in NVMe Base Spec_1_2.x (figure-4)
  - PCI-e port, function, or VF are supported
  - Multi tenant extensions have been added
- NVMe Controller Register set is accessed by the ***NVMe Host***
  - A separate register-map is being defined for the capsule interface for fabrics
- For NVMe over Fabrics, each controller offers multiple software abstractions each dedicated to a host (abstraction, or instance, or virtual-controller are terms used)
- ***In the rest of this presentation I shorten NVMe over Fabrics to NVMe as has become customary; however NVMe over Fabric has differences with NVMe local over PCIe***

NVMe is developing an architecture model that is generic and supports multiple fabric I/O technologies

Scope of the first generation NVMe

- Continue to have a 1:1 temporal mapping of a host to a controller abstraction
  - A host can have sessions with a 2nd controller within the same subsystem or in another subsystem
- ***Discovery*** operation returns NVMe-subsystems names and addresses and not those of controllers
  - ***Hence all controller abstractions have access to the same set of namespaces (this statement needs to change to "all controllers allocated to a particular host have access …")***
  - Discovery returns a single record at a time with a single address
  - Multiple records can be discovered
- There is a 1:1 mapping of NVMe Submission to Completion Queues
- Queues shall be sized to support the maximum number of outstanding commands
  - No allowance for flow control at the NVM Express layer currently
- The host may have multiple outstanding ***NVMe commands*** for IO
  - The host shall have a maximum of one **capsule-command** outstanding (e.g. discover, connect, etc.)

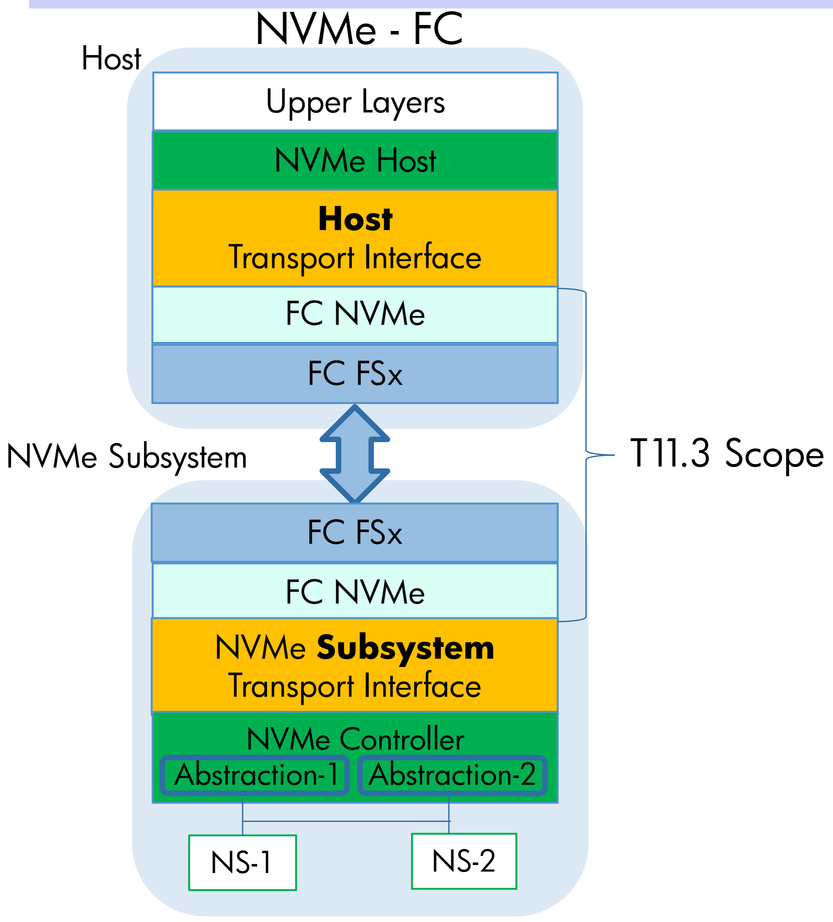
Capsule Ordering Requirement

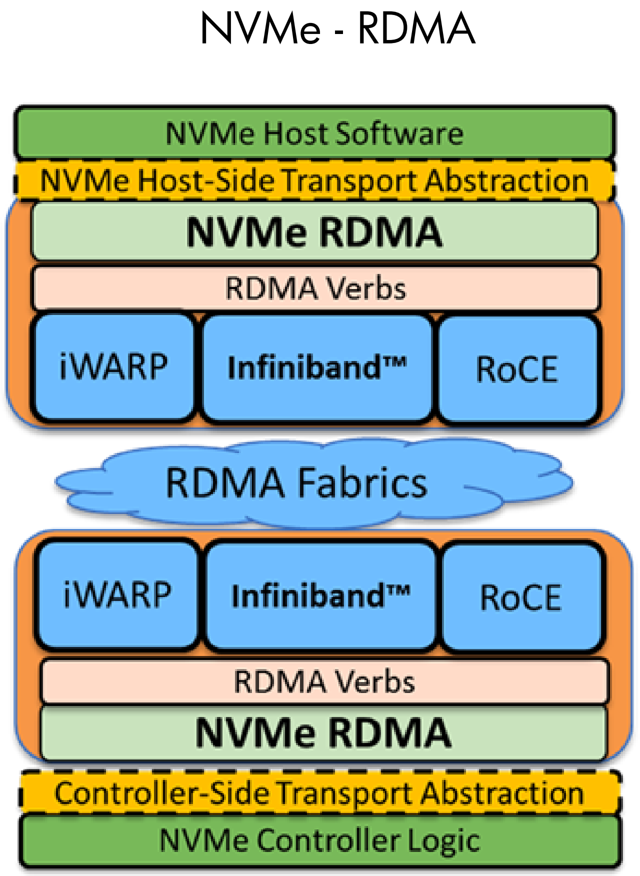
From fabric core specification:

“The transport shall provide *reliable in-order delivery* of capsules between a host and NVM subsystem over a *particular connection*. As one specific example, Submission Queue Entries (and Completion Queue Entries) shall be delivered in the order sent from the host to the NVM subsystem (or NVM subsystem to the host).”

- FC fabrics are connectionless on the wire! So there is no “particular connection”!
- There is no use case or documentation on why in-order delivery is required while a controller is allowed to execute commands on a submission queue out of order!
  - Upper layers usually don’t care
  - Current FC fabric granularity of concurrency is an exchange
  - The only additional requirement on a fabric should be to deliver capsules within a window of time (after which a capsule is discarded)
- *The FC-NVMe can attempt to deliver capsules in order between the transport layer interfaces on a per host session basis (i.e. per controller abstraction)*
  - *This may create some correlation between otherwise independent queues*
  - *Attempting to ensure order on a per controller queue basis is taxing on FC-NVMe*
- To enable ordered delivery a capsule command sequence number is defined as part of the FC Command IU which can be used in conjunction with the Host Session ID
- The scope of term reliable should also be clarified:
  - While FC offers reliable delivery the upper layer should be able to recover from loss of commands

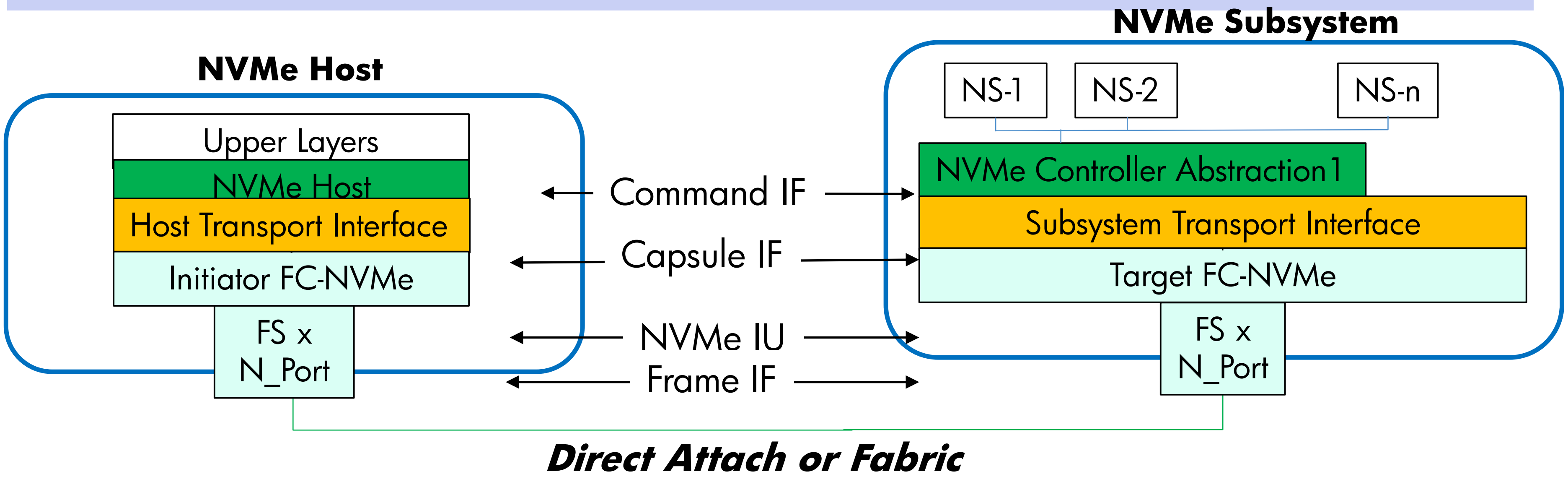
NVMe over Fabric (NVMe) – Protocol Layers and Abstractions





- Each subsystem (and host) terminates one or more fabric ports, but how?
- RDMA is connection oriented while FC is connectionless on the wire
- Fabric Interface layer in FC is equivalent to Transport Abstraction in RDMA
- FC-NVMf is the transport layer offering similar functions to FCP-4 for SCSI while Framing-Signaling provides the interconnect

NVMe– Functional Model – Host description



- NVMe Host Transport Interface description:
  - Is identified by an NVM-Qualified-Name (NQN)
  - For FC this qualified name is associated with the Initiator FC node WWN, and associated port WWNs and addresses
  - Discovers the subsystem and requests access to a controller using a *Connect-capsule*, can configure a controller, and enable its operation using *Get/Set* capsules
  - May have to authenticate with a subsystem on a per *connect* basis
- A distinguished host ***session-ID*** associates a host with a sub-system/controller and I_T_Nexus used for communications (see next slide)
- FC-NVMe maps capsules to NVMe Information Units (FCP4 equivalent)

Host Session-ID & ITNID Proposal

Define the following two parameters:

- Host Session-ID (**HSID**) [2 bytes]: Set by a host as part of "Connect-Capsule" is a handle for {Subsystem NQN, Controller-ID} (one HSID per controller)
- An Initiator_Target_Nexus-ID (**ITNID**) [x bytes]: Is a host-subsystem unique identifier for a given FC Initiator Target Nexus and is *discovered* via NVMe discovery service
  - e.g. the "Initiator port WWN + Target port WWN" can form the ITNID (please see next slide)

Operations:

- The term Distinguished Host Session-ID : (HSID, ITNID) refers to a local handle for host-subsystem-controller communications over an I_T_Nexus
- Queue-ID used in FC-NVMe IU and also SQE/CQE identifies a unique queue for a controller (identified by CNTLID in FC-NVMe IUs)
- ***Discover*** Capsule for FC fabrics will return all the ITNID(s) (instead of a port transport address) for a subsystem
- ***Connect*** Capsule will use the Host Session-ID as defined above
  - If a time-stamped session identifier (RFC 4122) is also required for purpose of collecting logs etc. it will be a different field
- A service request/indication or response/confirmation will provide the (HSID, ITNID) as parameters
- FC-NVMe Information Units (IU) will carry the HSID (proposal to T11.3) as well as the QID
- ITNID(s) can be added or deleted as status of ports change within the fabric –dynamics of discovery

Two Options considered for the Initiator-Target-Nexus Identifier

Option-1 (recommended)

ITNID = {host port WWN, subsystem port WWN} [16 bytes]

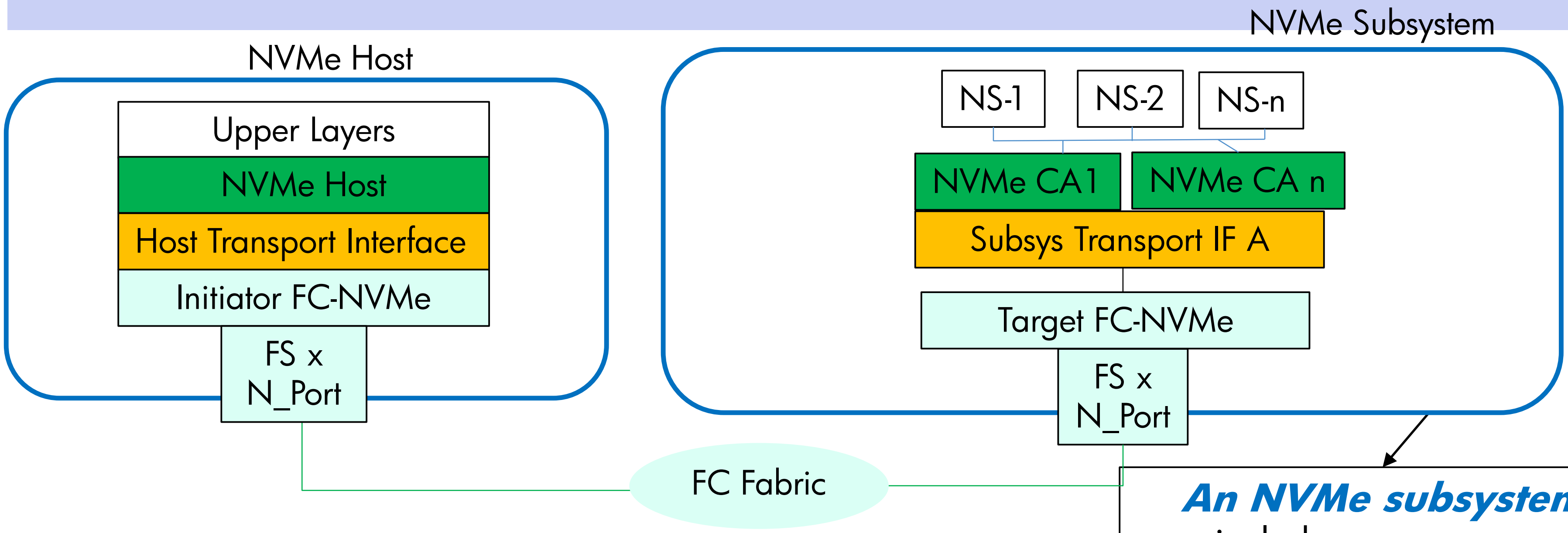
- Are known a priori for each device vs. FCID which is learnt after a FLOGI; so can be used to populate the NVMe directory ahead of time
- Remain stable while port FCIDs can change. This masks fabric events from upper layers
- Relatively easy to work with and map to FCIDs
- PWWNs are also used in fabric zoning, so this information is already configured somewhere and can easily be pushed/pulled into the NVMe directory service and associated with a SUBNQN
- By assigning an alias can be associated with a readable symbolic name

Option-2 (not recommended)

ITNID = {host port FCID, Subsystem port FCID} [6 bytes]

- The FC fabric assigns FCID(s), so they are usually NOT known a priori. Need to setup FC connectivity, find what addresses are assigned , and enter them into the NVMe Directory Server
- A port FCID may change after a port up/down (FLOGI) event or switch reboot and the change may have to be propagated up the layers past the FC-NVMe layer

NVMe – Subsystem description



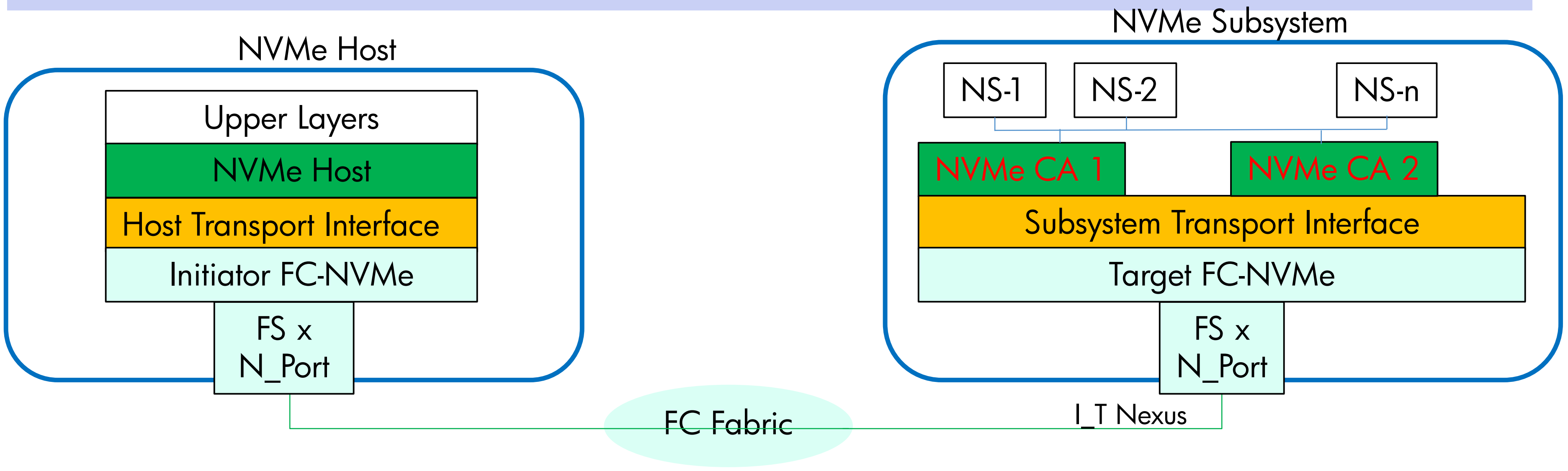
- ***NVMe subsystem Transport interface is identified by a NVM-Qualified-Name*** (SUBNQN)
  - The NQN is mapped to the target FC WWN, and associated port WWNs and addresses
  - Optionally authenticates a host,
  - Responds to a "***connect***" capsule and returns a controller-ID to a given host with access to the host's name space
  - ***Also need a means of communicating fabric events to the transport interface and NVMe layers***
  - A host and subsystem can exchange capsules over FC N_Ports zoned for this purpose

An NVMe subsystem includes one or more controllers each offering multiple controller abstractions with one or more namespaces, one or more fabric ports, a non-volatile memory storage medium, and an interface between the controller(s) and non-volatile memory storage medium which is proprietary

NVMe – Host subsystem - Discovery

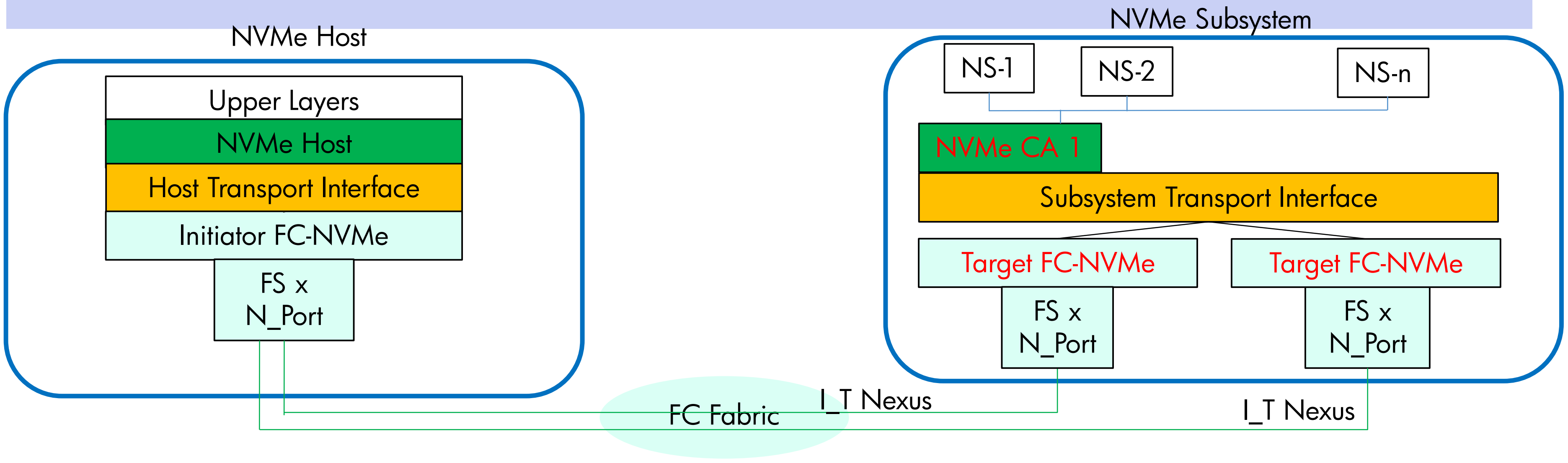
- Options for NVMe discovery over FC fabrics are under discussion within T11.3 – along side changes made to discovery in NVMe WG
- Purpose of NVMe discovery is to associate a SUBNQN and means of communicating to it with a host-NQN
- The fabric is zoned a priori and initiator and target ports on NVMe host and subsystem establish port/process LOGINs (independent of NVMe discovery)
- Therefore FC-NVMe currently makes no use of addresses supplied by discovery as the plumbing has already happened at FC-FSx layers
- The FCIDs are also not known until this plumbing is up and running, so the NVMe directory cannot be populated a priori unless we use port WWNs
- ***Therefore it is recommended that the NVMe discovery should provide the associated subsystem qualified names and*** ***I_T_Nexus-ID(s)*** in ***discovery response records***
- Discovery service should be dynamic enough to report any change to I_T_Nexus identifiers due to port up/down events
- The discovered subsystem qualified name is used by the host to establish a session between the host and a subsystem-controller using the Connect-Capsule
  - The discovered ITNID will be used as a local handle to steer the capsule along the selected I_T_Nexus
- Note that it is the FC fabric zoning policy that determines which subsystem ports are allowed for use by a given host port

NVMe – Why Use Host Session-ID in NVMe IUs



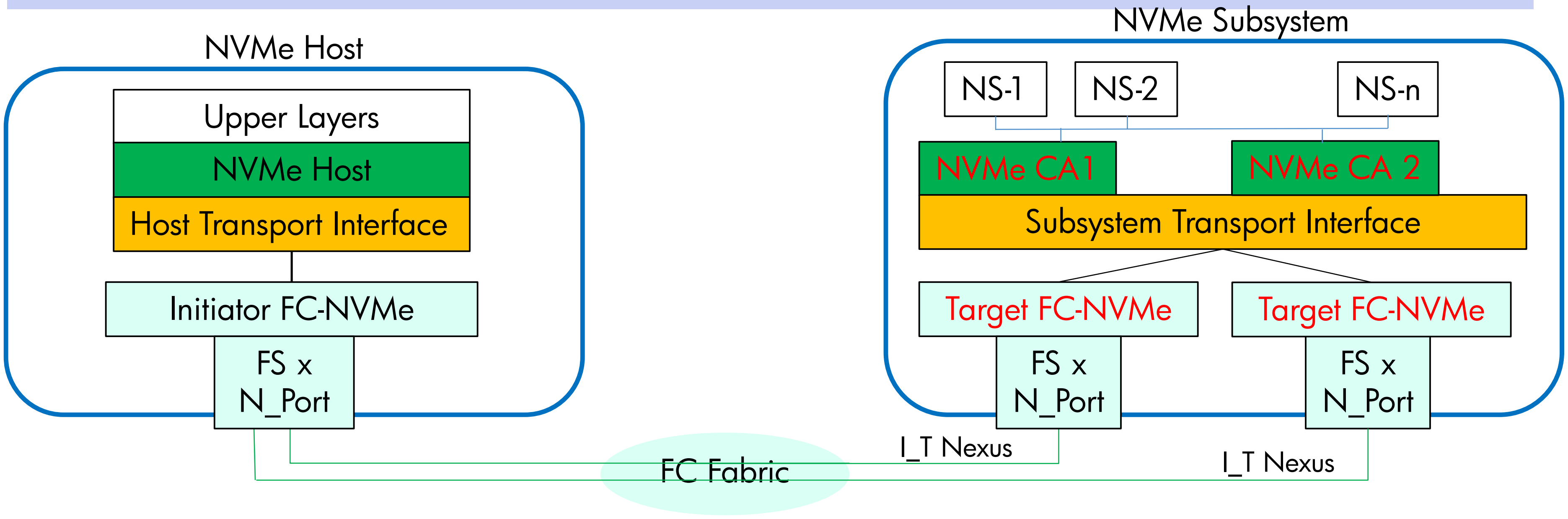
- NVMe IUs carry the CNTLID & QID
- ***We recommend to carry the host session-ID in NVMe IUs as well, here is how it is used:***
- The NVMe IU is sent as one or more frames each carrying a FQXID – ***no concept of a connection on the wire in FC (as opposed to RDMA)***
- At the subsystem the target FC-NVMe de-capsulates the NVMe IU and passes the NVMe capsule to the Transport interface (using the "HSID, ITNID" as the handle)
- Subsystem Transport-IF will in turn use the HSID, CNTLID to map to a given controller
- The controller will use the QID to place the command on a given queue
- A response capsule carries the completion QID and uses the (HSID, ITNID) as handle to return via the same I_T_Nexus
- The host having reassembled the IU, uses the host session-ID to forward to the correct Completion-Queue

NVMe – Dual Subsystem Ports



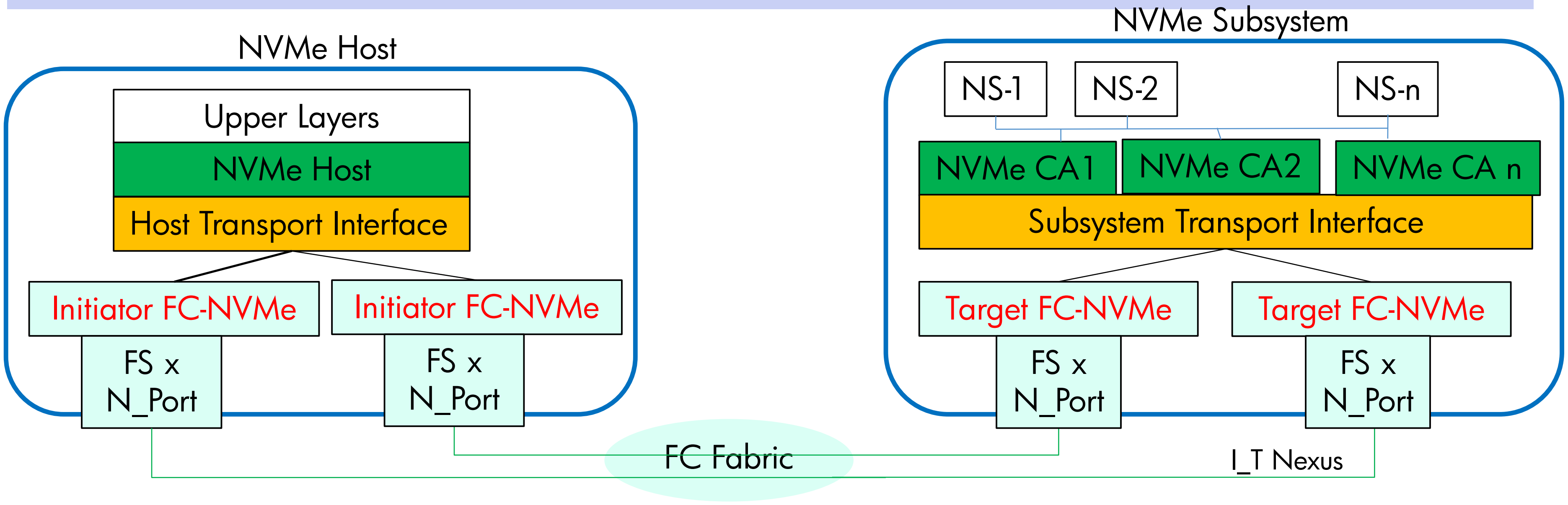
- NVMe sub-system with two ports both zoned for access by a host through port/process LOGINs; hence x2 I_T nexus are established by FC
- The target FC-NVMe will be detected twice, once for each I_T nexus
  - ***And registered/presented by FC FSx via two target devices***
- Hence the host Transport interface should ***discover*** the subsystem with two I_T nexus (proposal to NVMe WG)
- ***Since FC-NVMe does not pick a path, all capsule service calls should include the distinguished HSID (HSID, ITNID)***
- Capsules for a given controller queue can be transported over all available I_T nexus:
  - ***If one subsystem port is down the host controller session (and admin/SCQ pairs) continue over the other I_T_Nexus***
  - ***For RDMA type transports to achieve similar capability would require sticky sessions***
- By allowing capsule & IU transport to occur over both I_T nexus FC-NVMe can offer a robust transport service

NVMe – Single Host -- Dual Controllers



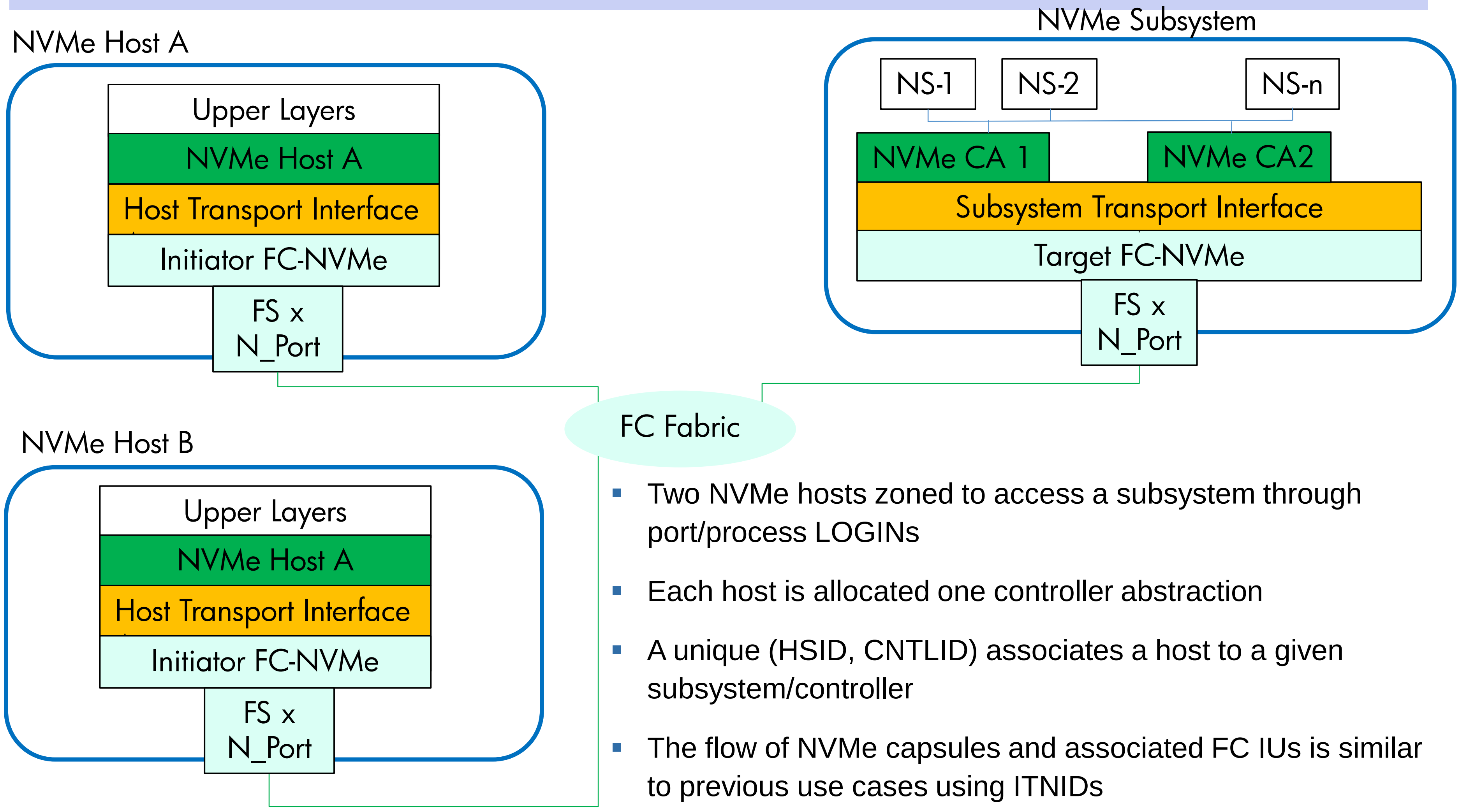
- In NVMe-local model, a host has register access to multiple controllers (once discovered through PCIe)
- In NVMe over Fabric, the response to a connect-capsule from subsystem allocates a single controller-abstraction to a host; subsequent connects use the allocated controller-ID
- For a host to access a 2nd controller on the same subsystem a connect with Controller-ID=FFFFh needs to be used (as well as a new session)
- ***We recommend that it be a requirement on NVMe to return a redundant controller for a 2nd session (e.g. on physically diverse controllers)***
- The host can communicate with both controller abstractions over both I_T Nexus using the (HSID, ITNID) as a handle to ensure responses are returned over the same path as corresponding command

NVMe – Dual Host to Dual Subsystem Ports



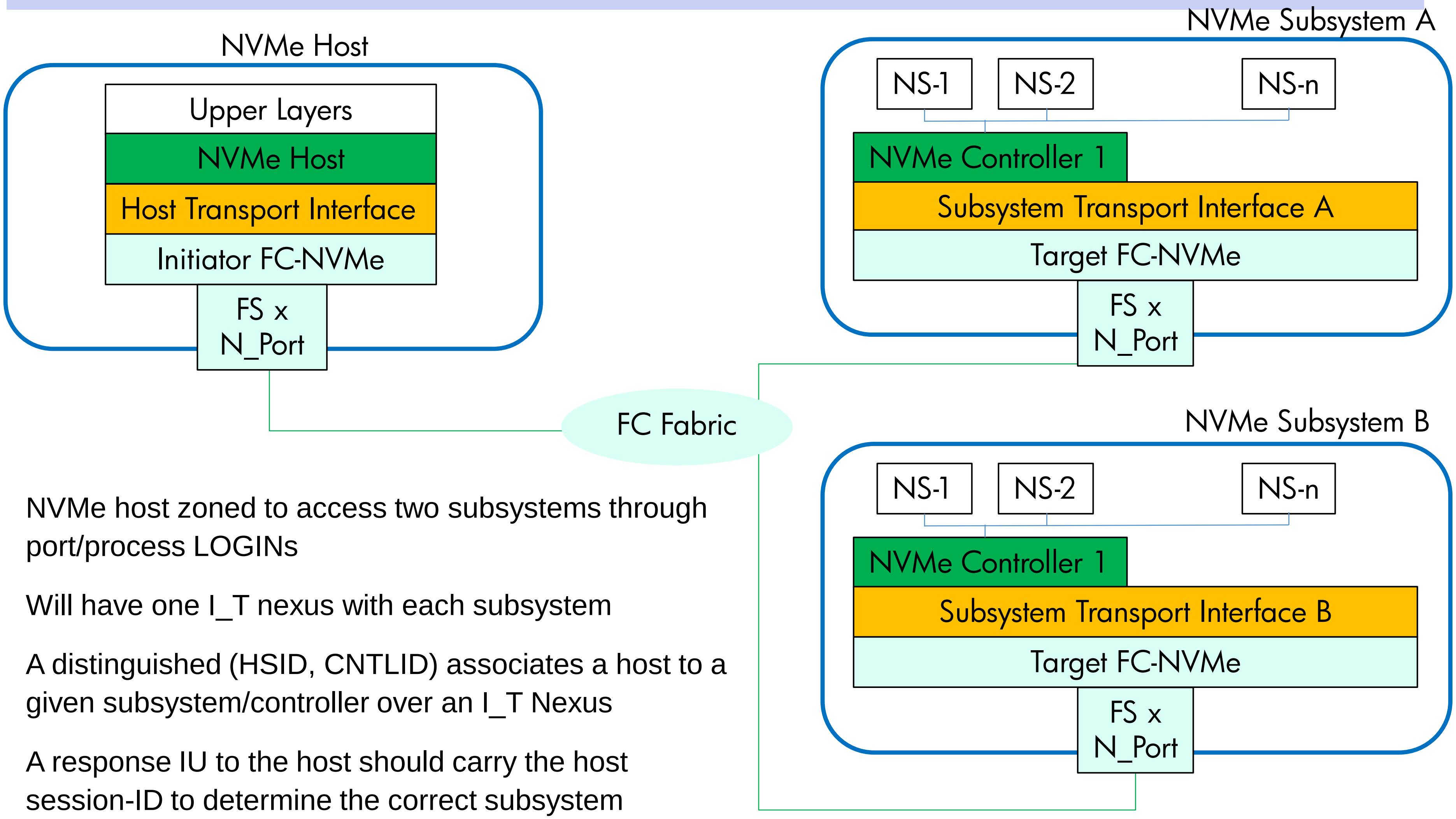
- NVMe sub-system with two ports both zoned for access by two host N_Ports through port/process LOGINs (x4 I_T Nexus e.g.)
- Each allocated NVMe controller/queue can receive exchanges/FCPIUs/Capsules from either port
- The flow of NVMe capsules and associated NVMe IUs is similar to previous slides
- ***The mechanisms supporting multipath while supported by NVMe and FC-NVMe constructs are beyond the scope of this specification***
- Note that multipath is defined as having multiple physical path to a name space via the same controller Abstraction
  - Path via a secondary controller to the same name space is often used for HA and needs to be redundant

NVMe – Two Hosts Single Subsystem



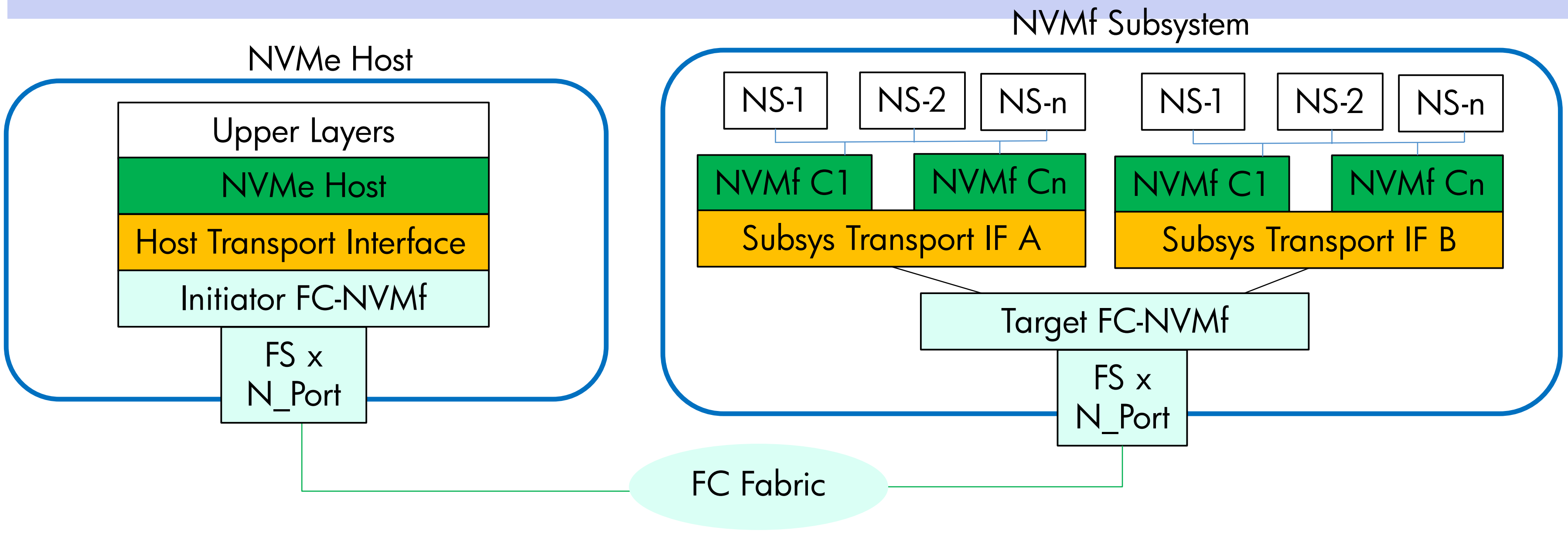
- Two NVMe hosts zoned to access a subsystem through port/process LOGINs
- Each host is allocated one controller abstraction
- A unique (HSID, CNTLID) associates a host to a given subsystem/controller
- The flow of NVMe capsules and associated FC IUs is similar to previous use cases using ITNIDs
- The HSID, ITNID should be used in various service calls to ensure delivery to the right host

NVMe – Single Host Two Subsystems



- NVMe host zoned to access two subsystems through port/process LOGINs
- Will have one I_T nexus with each subsystem
- A distinguished (HSID, CNTLID) associates a host to a given subsystem/controller over an I_T Nexus
- A response IU to the host should carry the host session-ID to determine the correct subsystem

NVMe – Subsystem with two NQNs!? – Is this allowed?



- ***Does the NVMe specification allow a subsystem with two or more NQNs?***
- NVMe host zoned to access a subsystem through port/process LOGINs over a single I_T Nexus
- Discovery will return two subsystem NQNs each with the same ITNID
  - One I_T nexus for both subsystem transport interface A & B each with a unique NQN
- The host needs to maintain unique host session-IDs across both subsystems
  - Host Session-ID => {SUBNQN, CNTLID}
- ***Does the NVMe standard allow this use case?***

Recommendations

- Defined a Functional Model with component descriptions, proposed for use in NVMe core fabric TP
- Add distinguished host session-ID to NVMe which is the tuple:
  - Host-Session-ID (2 bytes)
  - I_T_Nexus-ID (16 bytes) is a pair of port WWNs
- Proposed modifications to Discovery to return per subsystem ITNID(s)
- Proposed modifications to Connect capsules to use the new formatted HSID to identify a session for a given host's communications
- Proposed modifications to NVMe IUs to carry the host session-ID, in addition to CNTLID/queue-ID
- We recommend that it be a requirement on NVMe to return a redundant controller for a 2nd session (e.g. on physically diverse controllers)
- Need to revisit the order of capsule delivery
  - Capsule in order delivery offered by FC-NVMe on a per controller basis maybe a stretch, (need to establish feasibility if order is required on a per queue basis)
- Together with NVMe WG we need to clarify the scope of reliable transport
- NVMe WG needs to clarify if multiple NQNs per subsystem are allowed?

Summary

- Showed the application of changes in the following use cases:

Single Host Port	Multiple Host Ports	Single Subsystem Port	Multiple Subsystem Ports	Single Controller	Multiple Controllers
√		√		√	
√			√	√	
√		√			√
√			√		√
	√		√		√
Two Hosts	Single Subsystem				
Single Host	Two Subsystems				

Thank You