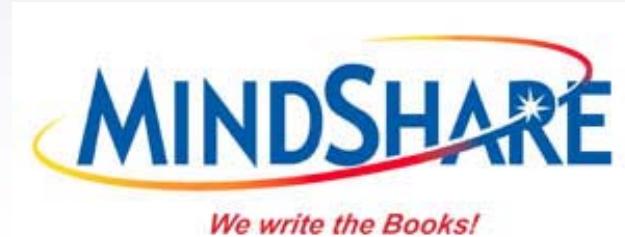




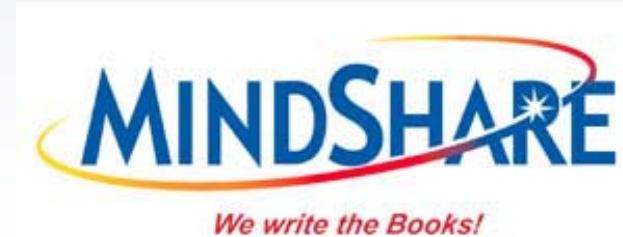
# *Introduction to SAS*

*Mike Jackson*

*June 2008*

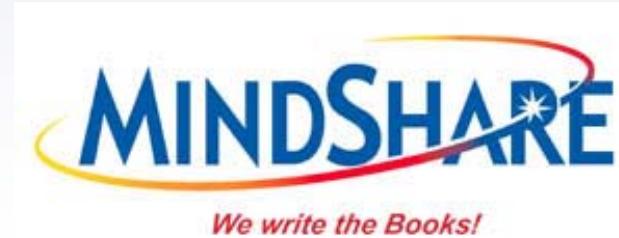


# *Part 1: Introduction and Background*



# *Chapter 1*

## *The Motivation for SAS*



# Serial SCSI (12)

## Shortcomings of existing models

### 1. Fibre Channel

- High-end, copper or fiber optics, capable of sophisticated networks of 16 million devices spread over several miles, but **Expensive** and complex

### 2. Serial Storage Architecture (SSA)

- Lower cost and lower performance, **Limited** to IBM, few others use it

### 3. IEEE 1394a

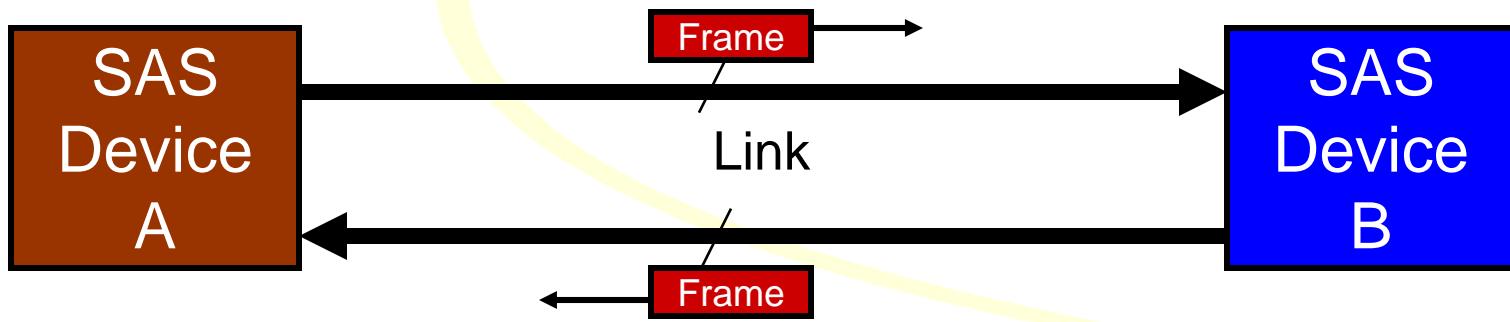
- Simple, supports isochronous traffic, but doesn't scale up well, considered too expensive and **too slow** for disk drives (~50MB/s)

# SAS Design Goals (13)

- Improve SCSI bandwidth
  - Wide ports permit several simultaneous connections, multiplying transmission bandwidth
- Low cost
  - Small, inexpensive connectors and cables, simple networks
- Maintain SCSI software stack
- Fail-over capability
  - Dual-ported drives

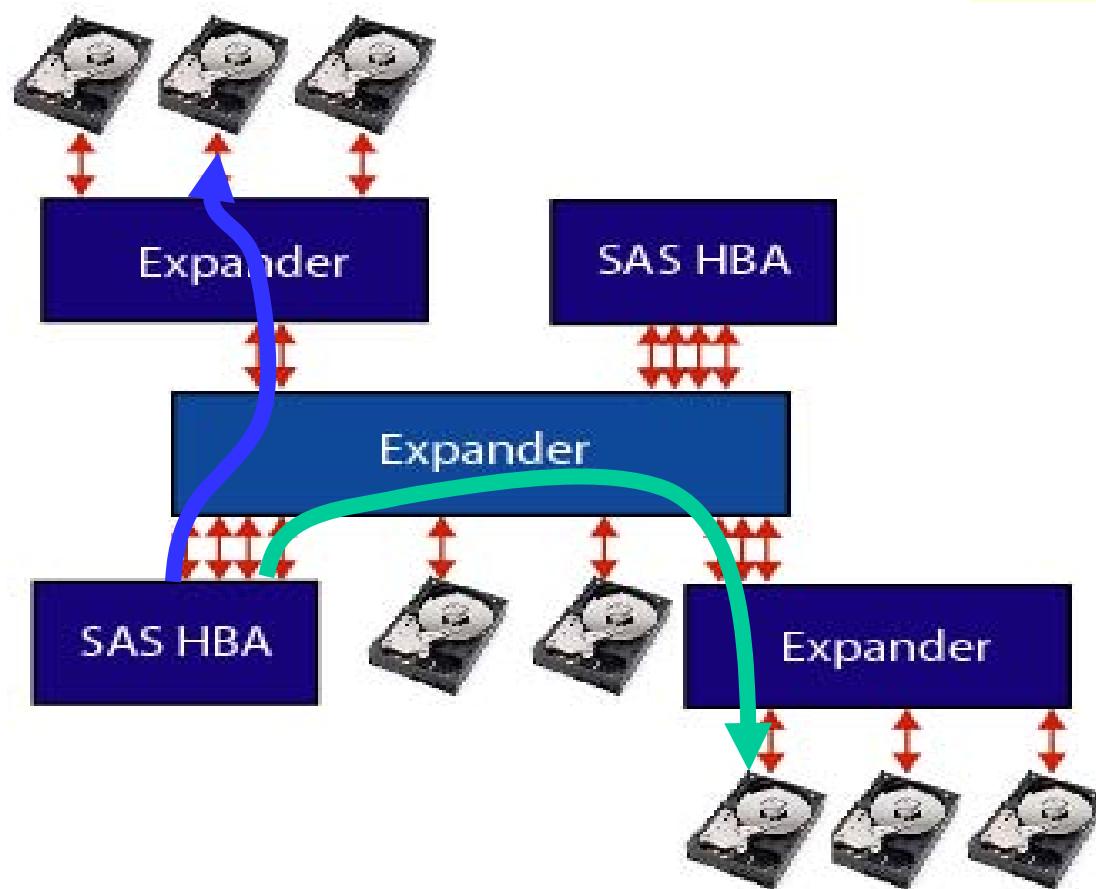
# SAS Solution (15)

- Point-to-point connection
- Dual-Simplex (full-duplex) connection permits bi-directional traffic
  - 1.5 Gbits/s (1<sup>st</sup> generation, or G1)
  - 3.0 Gbits/s (G2)
  - 6.0 Gbits/s (G3)
- Frame-based transaction protocol



# SAS Topology

- A wide port can access more than one device at a time



# SAS Bandwidth

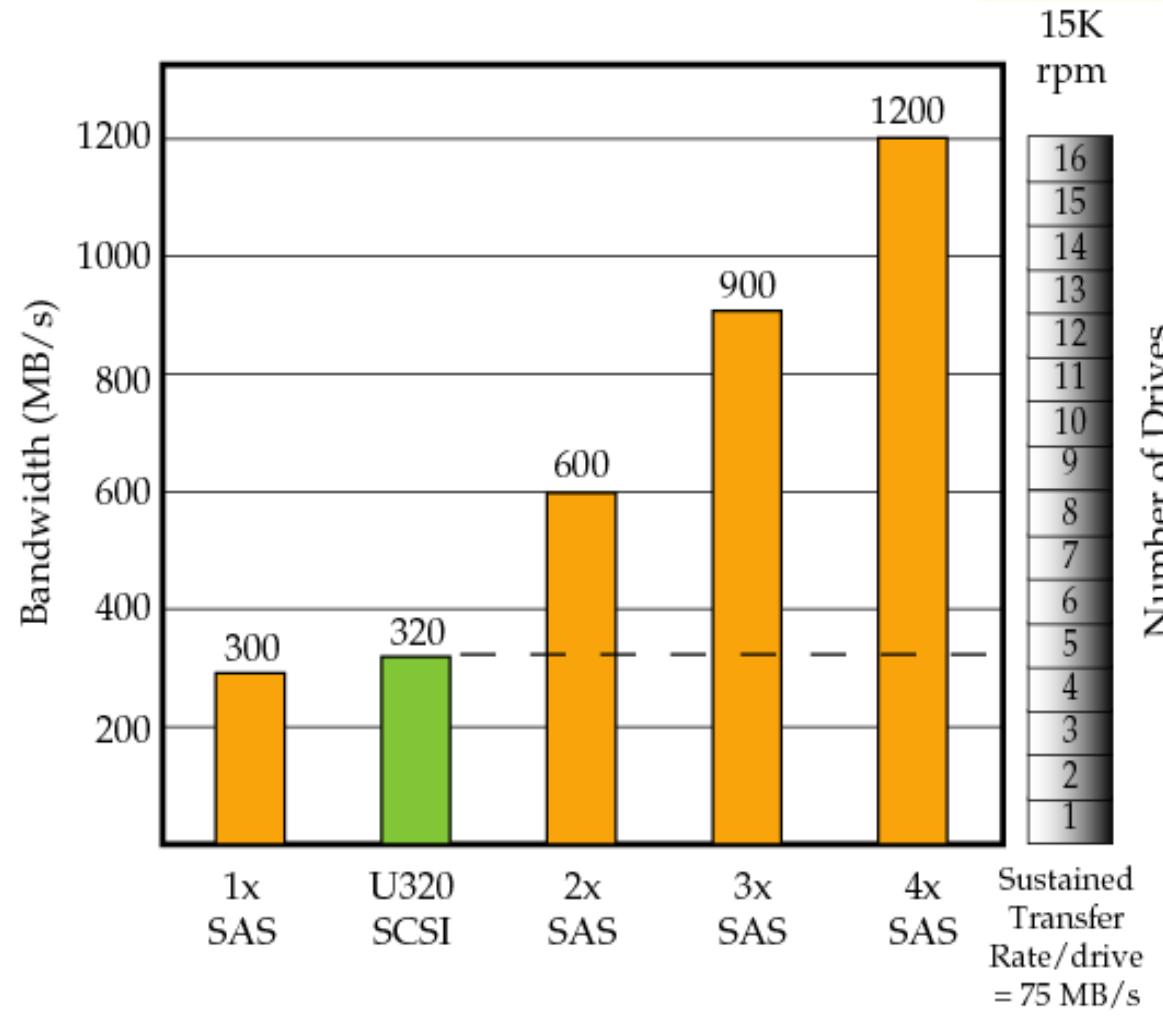
<b>Link Rate</b>	<b>3.0Gbps</b>
<b>Bidirectional BW (MB/s)</b>	300
<b>Aggregate BW (MB/s)</b>	600

- Bandwidth based on overhead of 8b/10b encoding: 10 bits sent per byte
- Aggregate BW further assumes simultaneous traffic in both directions

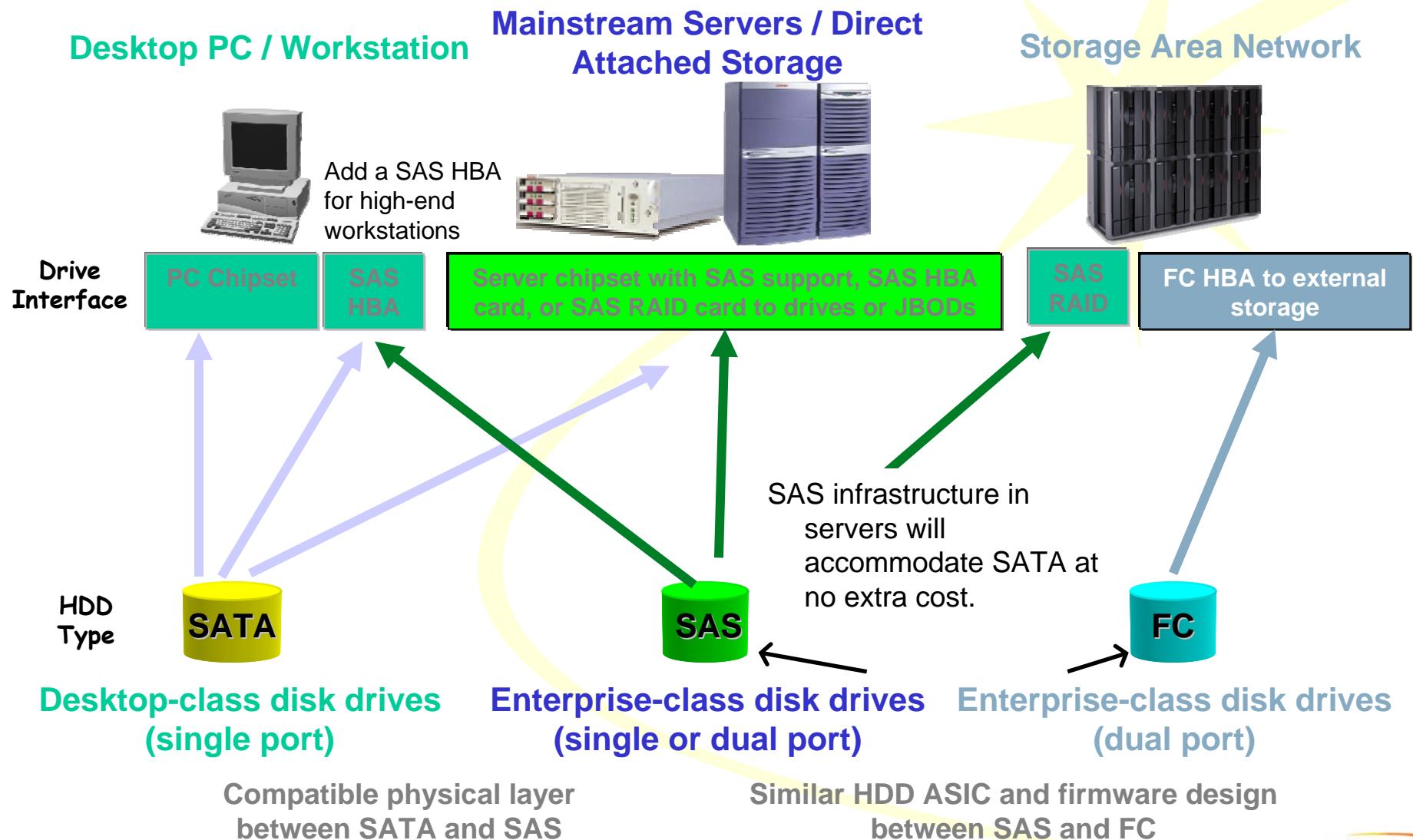
# Bandwidth Comparison (17)

Note that links **must** use independent transactions (only one link can be used for the same connection at the same time).

This is different from the way other architectures, like PCIe and IB work.



# Positioning SATA, SAS, and FC (18)



# Comparison of Drive Types (20)

Characteristics	Mobile	Desktop	Enterprise
<b>rpm</b>	3600, 4200, 5400	5400, 7200	10K, 15K
<b>Seek time</b>	12 – 14 ms	8.9 – 9.5 ms	3.2 – 7.4 ms
<b>Performance as file server*</b>	N/A	79 – 136	146 - 372
<b>Write cache</b>	2 MB	2 – 8 MB	2 – 8 MB
<b>Capacity</b>	10 – 500 GB	40 – 750 GB	18 – 300 GB
<b>Reliability</b>	300 K hr MTBF	500 K hr MTBF, 8hr/day	1.4 M hr MTBF, 24hr/day
<b>Power</b>	2.5 W	10 W	15 W
<b>Cost</b>	\$73 – \$160	\$75 – \$240	\$160 – \$1400
<b>Interfaces</b>	ATA/66, ATA/100	SATA-150/300	Ultra 320 SCSI, SAS, FC

\*Refer to [www.storagereview.com](http://www.storagereview.com) for recent HDD performance comparisons. These values are from 2002 testing.

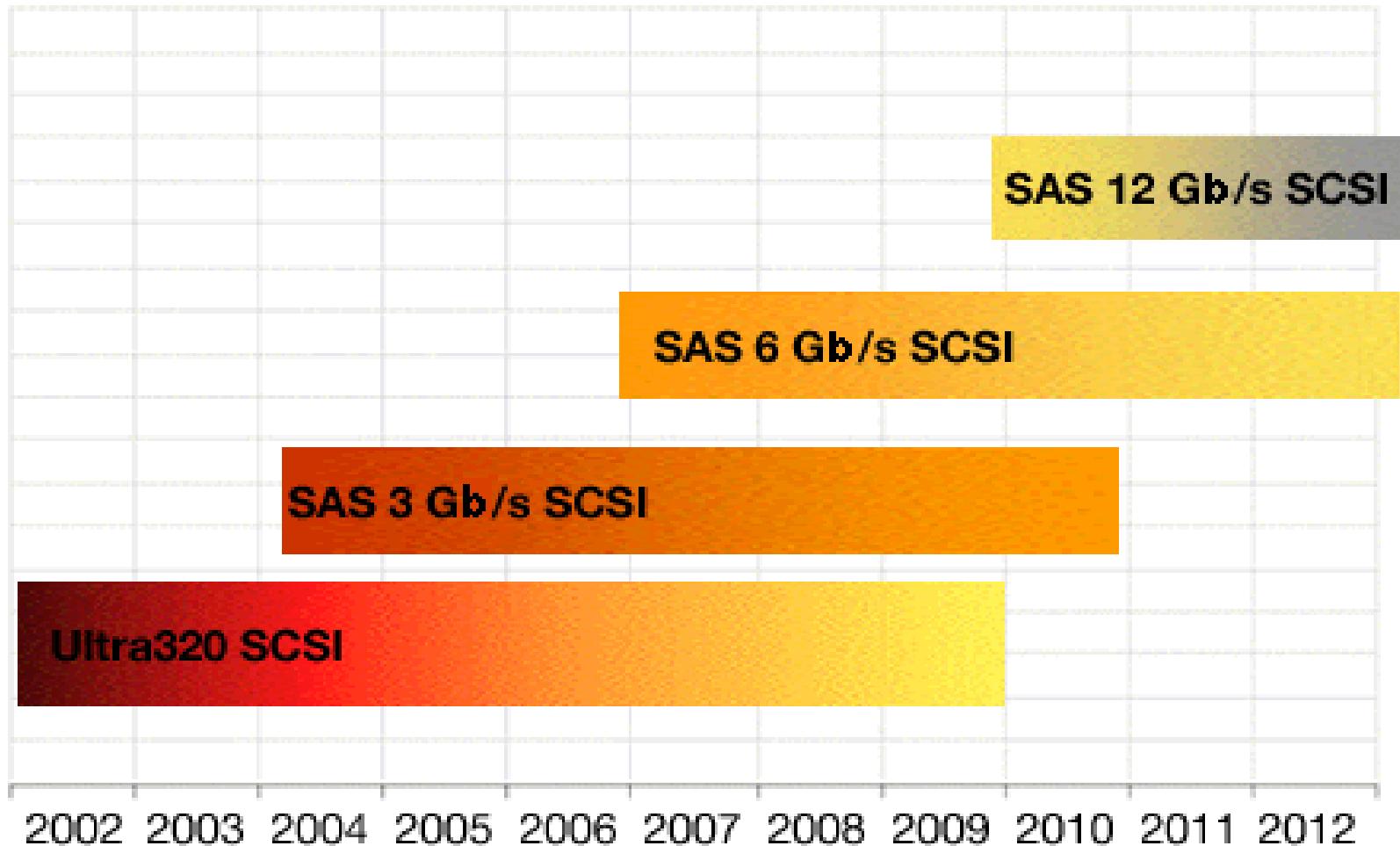
# Interface Comparison (22)

	SATA	SAS	FC
Cost	Low	Low	High
Complexity of Topology	Lowest; Port Multipliers permit limited expansion	Medium; up to 16K devices with Expanders	Highest; fabrics/loops, up to 16M devices
	no peer to peer	peer to peer	peer to peer
	single host, Port Selector provides fail-over	multi-initiator	multi-initiator
Cable Length	0.5 - 1m	~6m	10Km
Performance	1.5 - 3.0 Gb/s, Half Duplex	3.0 – 6.0 Gb/s, Full Duplex	2.0 – 4.0 Gb/s, Full Duplex
Reliability, Availability	Low - RAID improves it, but rebuild times are a factor	High	High
Dual-Port HDDs	With Port Selector	Yes	Yes
Drive Capacity	Highest	Low	Low
Other	Stricter consumer product FCC standard requires SSC	SSC supported only for SATA compatibility	No SSC requirement

# Usage Model Comparison

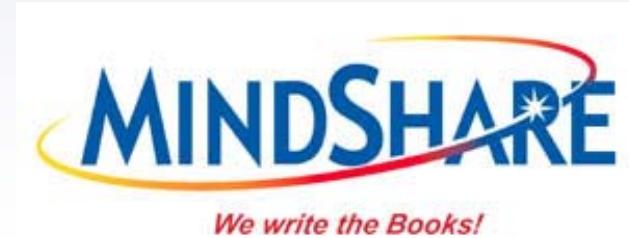
- SAS used where performance precludes SATA
  - Replacing existing SCSI installations
  - Servers, SANs (inside the box)
- SATA replaces older near-line or backup storage with RAID systems
  - Replacing some tape backup systems
  - Near-line disk backup
- Fibre Channel used for long-distance connections
  - SAN (outside the box)

# STA Roadmap (26)



# *Chapter 2*

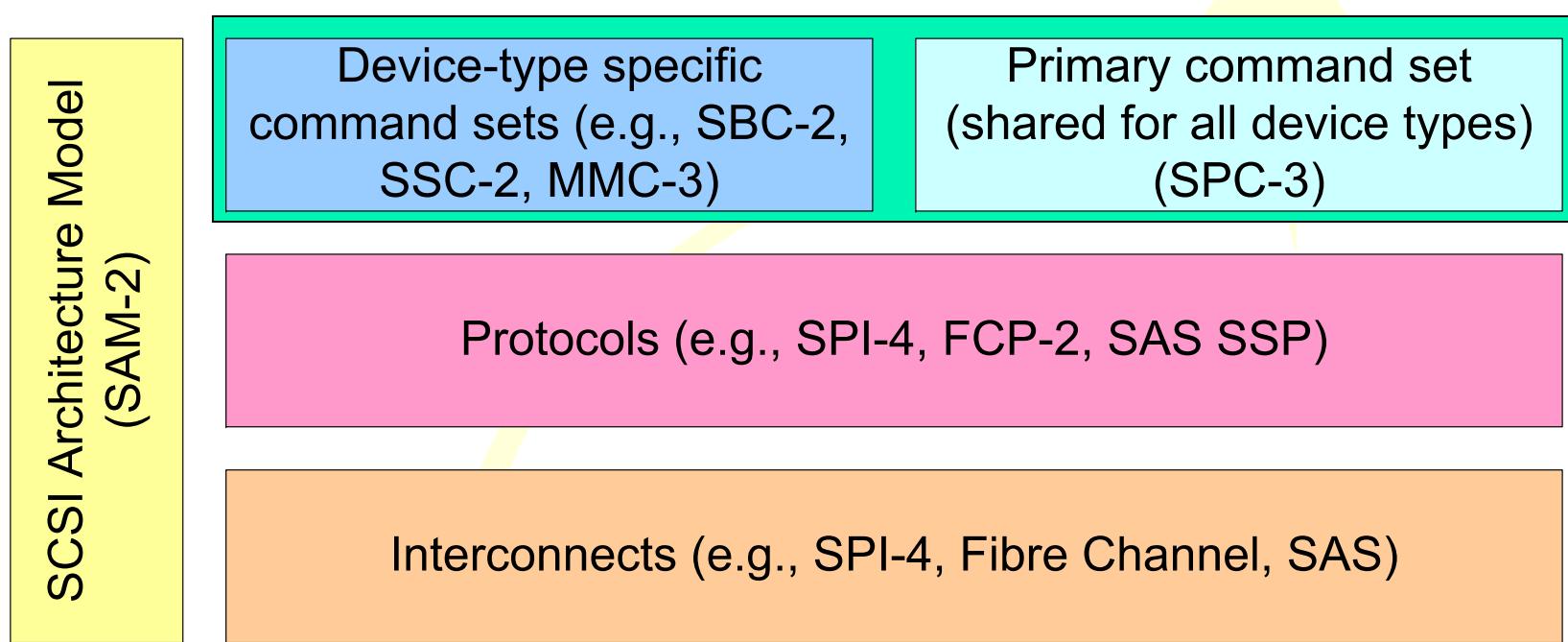
## *Origins of SAS and Background*



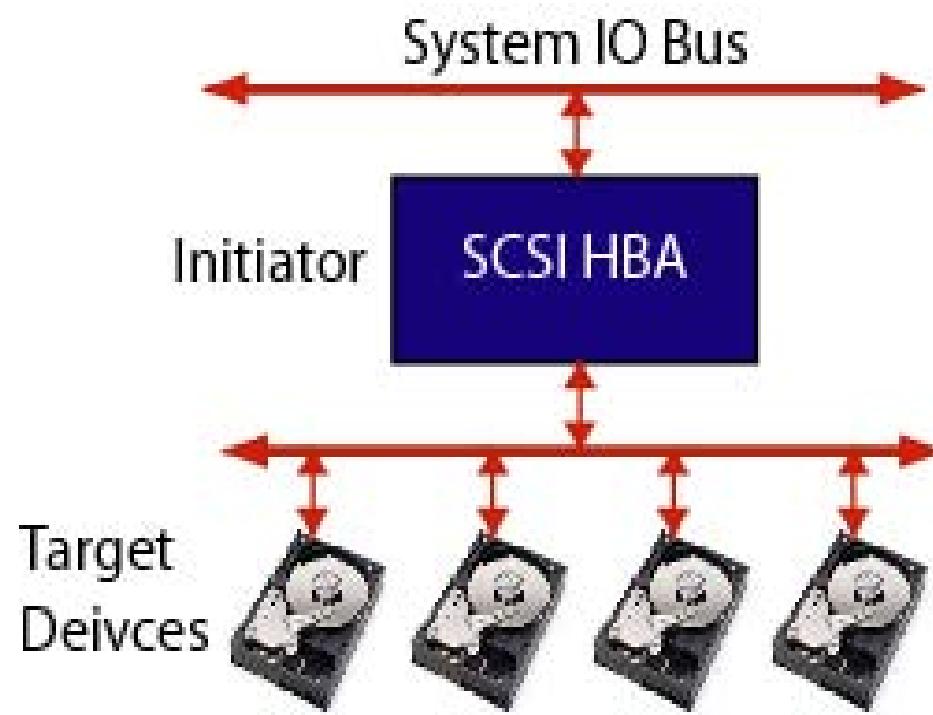
# The SCSI Approach (28)

- Prior to SCSI, hard drive interfaces were defined for device-level communications
  - Software needed to know the low-level details to access the devices
  - Number of devices supported was usually limited to 1 or 2.
- SCSI allowed:
  - More devices
    - 8 on a narrow bus, 16 on a wide (16-bit) bus
  - Different types of devices
  - Commands that didn't need details of the drive
  - Communication independent of main system bus

# SCSI Standards (29)

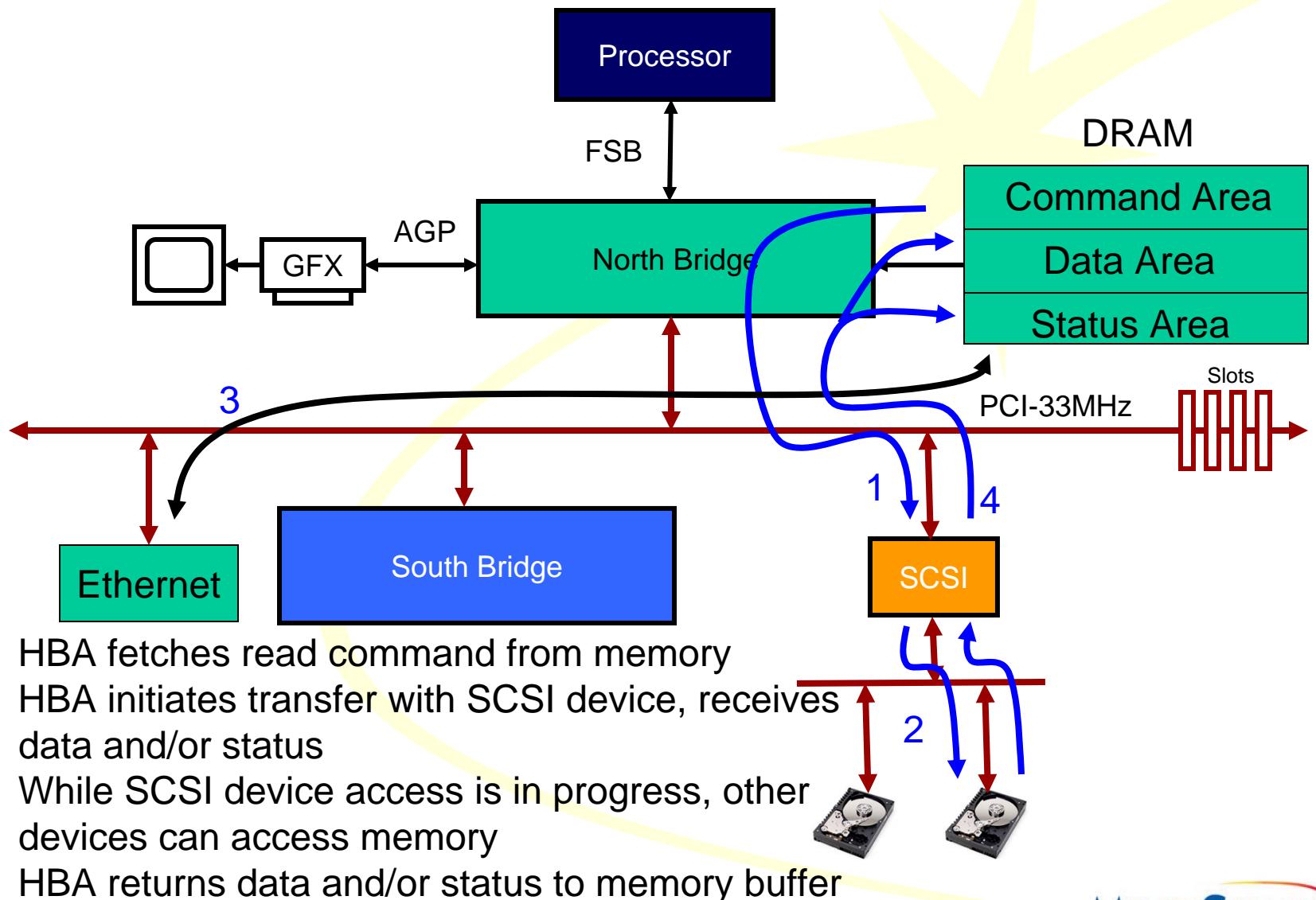


# SCSI Bus



- Host Bus Adapter (HBA) with direct access to main DRAM memory initiates most SCSI transactions.
- SCSI devices include an intelligent controller that manages the IO operation based on high-level commands.
- Devices can have multiple logical units, referred to by LUN (logical unit number), though most implement just LUN 0.

# SCSI Improves Performance (30)



# SCSI Definitions (31)

- **Connect:** Initiator successfully accesses a target and the nexus is stored.
- **Nexus:** relationship
  - I\_T nexus: initiator and target
  - I\_T\_L nexus: initiator, target, and logical unit
  - I\_T\_L\_Q nexus: initiator, target, logical unit, and tag for queued commands
- **Disconnect:** temporary termination of the transaction while waiting, allowing other devices to use SCSI bus during that time.
- **Reconnect:** target re-selects initiator to continue a previously-disconnected transfer

# SCSI Definitions (32)

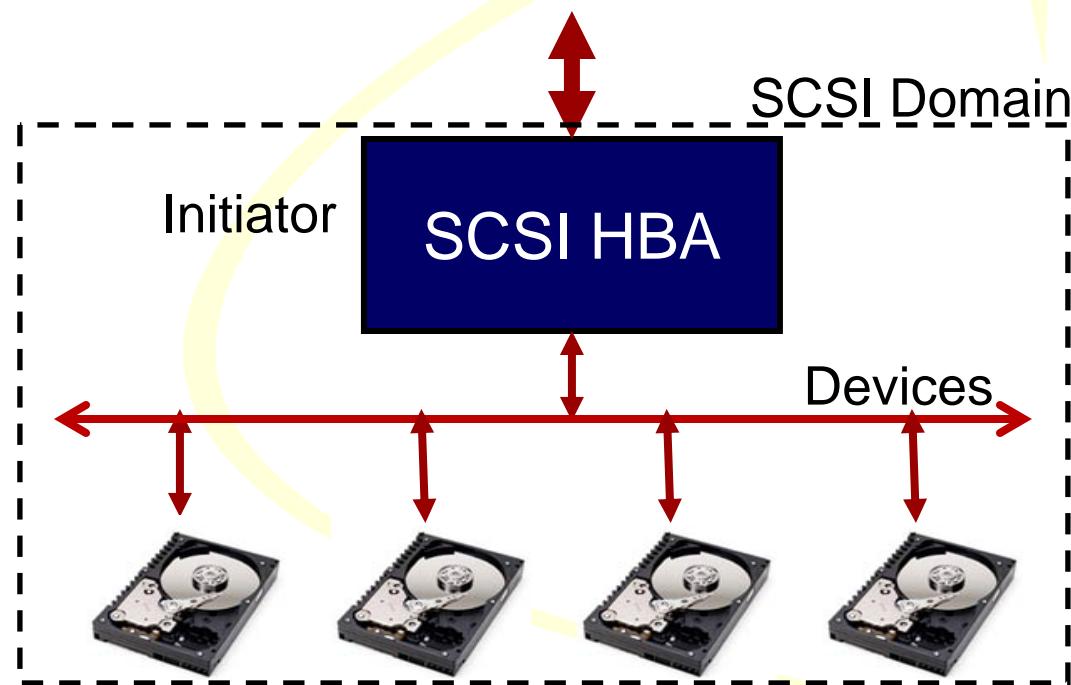
- Task is usually a command
- Task Management commands are defined in SAM3 to control tasks or logical units within a device
- Each command has a unique tag
  - For SAS, tag is 16 bits, unique within the I\_T nexus
  - Permits Tagged Queuing, which reduces access time by allowing drives to rearrange the order of transactions to improve performance
- Each task management function has a unique association
  - For SAS, 16 bits, unique within the I\_T nexus
- Commands are sent in Command Descriptor Block (CDB)
  - Operation Code field distinguishes commands
  - CDBs can be 6, 10, 12, or 16 bytes long; longer CDBs also allowed
  - CDB formats defined in SPC-3

# SCSI Definitions (32)

- **Service Delivery Subsystem:** bus that conforms to the SAM (SCSI Arch. Model). For more details, see SAM-3 at <http://www.incits.org/>
- **Initiator:** relatively simple device that initiates commands, then yields control to the target (RAID initiators can be more complex)
- **Target:** more complex device that controls the response to commands

# SCSI Domain (33)

- SCSI Domain consists of Initiators and Target devices that can communicate over a service delivery subsystem.

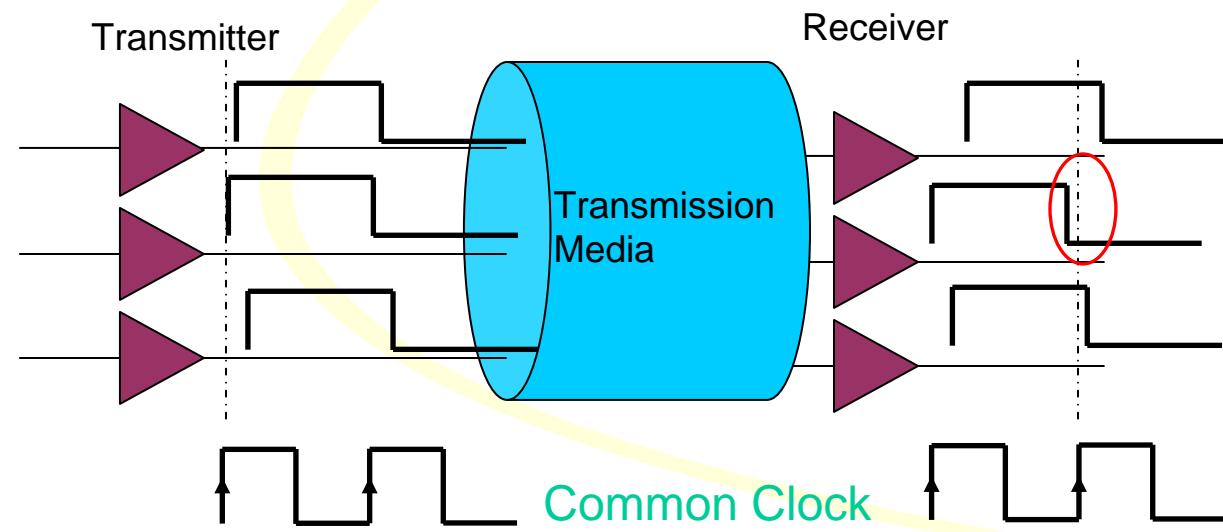


# Limitations of Parallel SCSI (33)

- Bandwidth of 320MB/s shared by 15 devices ( $320/15 = 21.3\text{MB/s}$  per device)
  - Typical enterprise drive can use about 80MB/s (up to 95MB/s) when reading from the media
  - 640 speed was considered, but proved too difficult to implement
- Limited number of devices on a bus – example: 15 possible, but reliable operation limits number to about 5
  - Shared bus with uncertain loading creates relatively noisy signaling environment

# Limits of any Parallel Interface (35)

- Power required to switch many buffers
- Cross-talk between signals
- Problems of clocking multiple signals at once:
  - Signal skew, flight time, clock skew, clock period



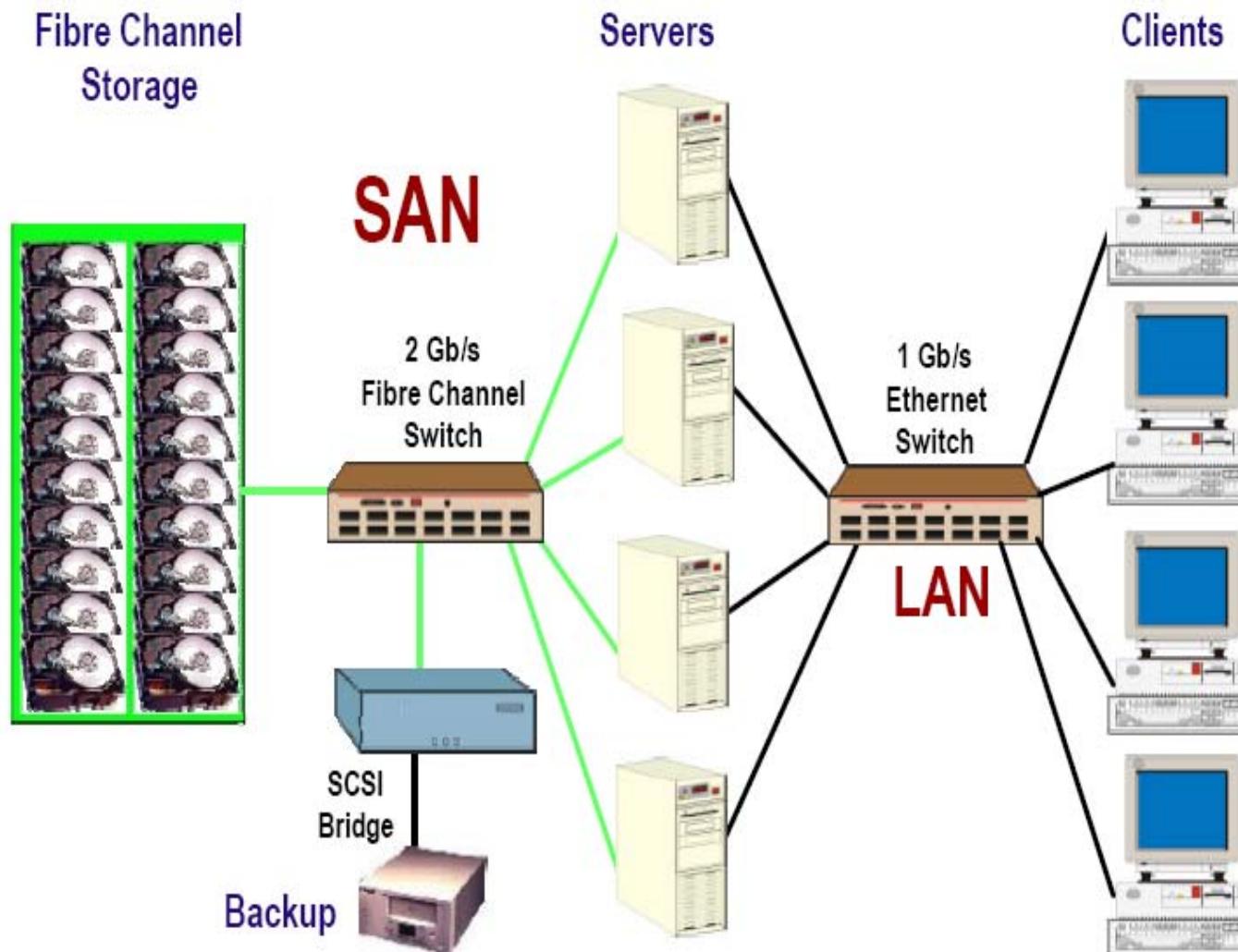
# Moving to a Serial Solution (36)

- Serial transmission embeds the clock into the data stream, no common clock required
  - Well-understood solution adopted from earlier designs like XAUI, SATA, Fibre Channel, InfiniBand, PCIe
- Serial transport solves limitations of parallel bus, enabling higher future speeds

# Fibre Channel Origins (36)

- Developed in 1988, supports fiber-optic cables to connect storage devices.
- Approved as ANSI standard in 1994
- Developed into a sophisticated technology and primary transport used in storage-area-networks (SANs).

# Example SAN Topology (37)

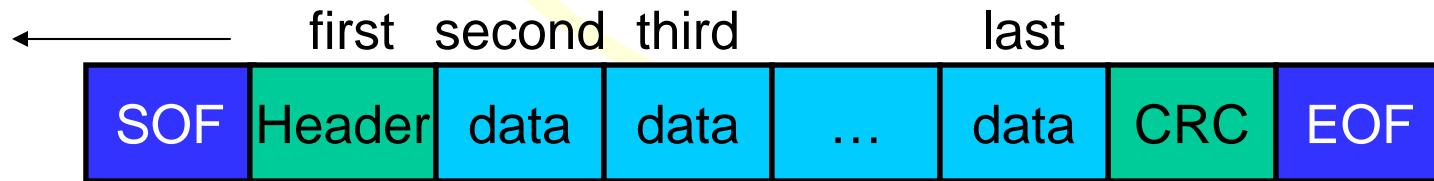


# Minimizing Design Effort

- SAS adopts elements of Fibre Channel to leverage industry experience and minimize repeated effort
- Examples: (see next slides)
  - Format of frame headers
  - Use of connections
  - Use of “switches”

# Frames (38)

- Like other serial transports, FC and SAS use a packet-based protocol. Packets, called Frames, are created within the device and sent out serially as a unit
- A frame is a structured sequence of dwords like the one shown below.



SAS drops some fields (and F\_CTL bits), but leaves their places in the header<sup>31</sup>

## Frame header (38)

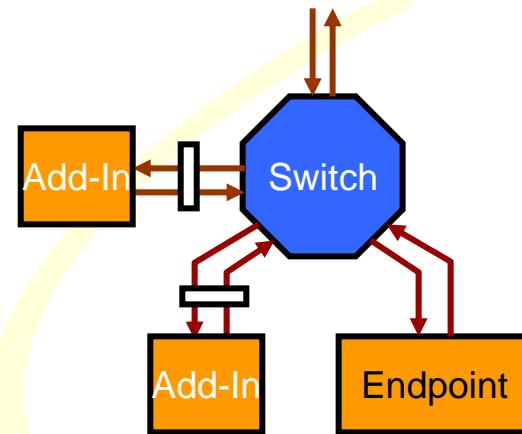
	Fields (bits)															
	7	6	5	4	3	2	1	0								
Byte 0	Frame Type (FC: R_CTL)															
1 to 3	Hashed Destination SAS Address (FC: Destination_ID)															
4	Reserved (FC: CS_CTL)															
5 to 7	Hashed Source SAS Address (FC: Source_ID)															
8	Reserved (FC: Type)															
9	Reserved (FC: F_CTL)															
10	Reserved (FC: F_CTL)			Retransmit		Rsvd										
11	Reserved (FC: F_CTL)			Number of Fill Bytes												
12 to 15	Reserved (FC: Sequence ID, DF_CTL, Sequence Count)															
16 to 17	Tag (FC: OX_ID)															
18 to 19	Target Port Transfer Tag (FC: RX_ID)															
20 to 23	Data Offset (FC: Parameter)															
24 to m	Information Unit															
m to (n-3)	Fill bytes, if needed															
(n-3) to n	CRC															

# Connections (40)

- FC supports several communication methods. SAS uses only one of those, similar to Fibre Channel class 1, called a “connection-based” method, that dedicates the resources between devices.
  - Advantages: Speed and reliability
  - Disadvantage: Resources are tied up for longer periods of time
  - Arbitration between competing requests may be needed when establishing the connection.
- Other protocols use a “connection-less” protocol, sending packets that each contain a destination address for routing.

# Switches (40)

- Serial transports like FC and PCIe implement switches to add devices
- Switches act as bridges to fan out the buses



- In SAS, **Expanders** do a similar job

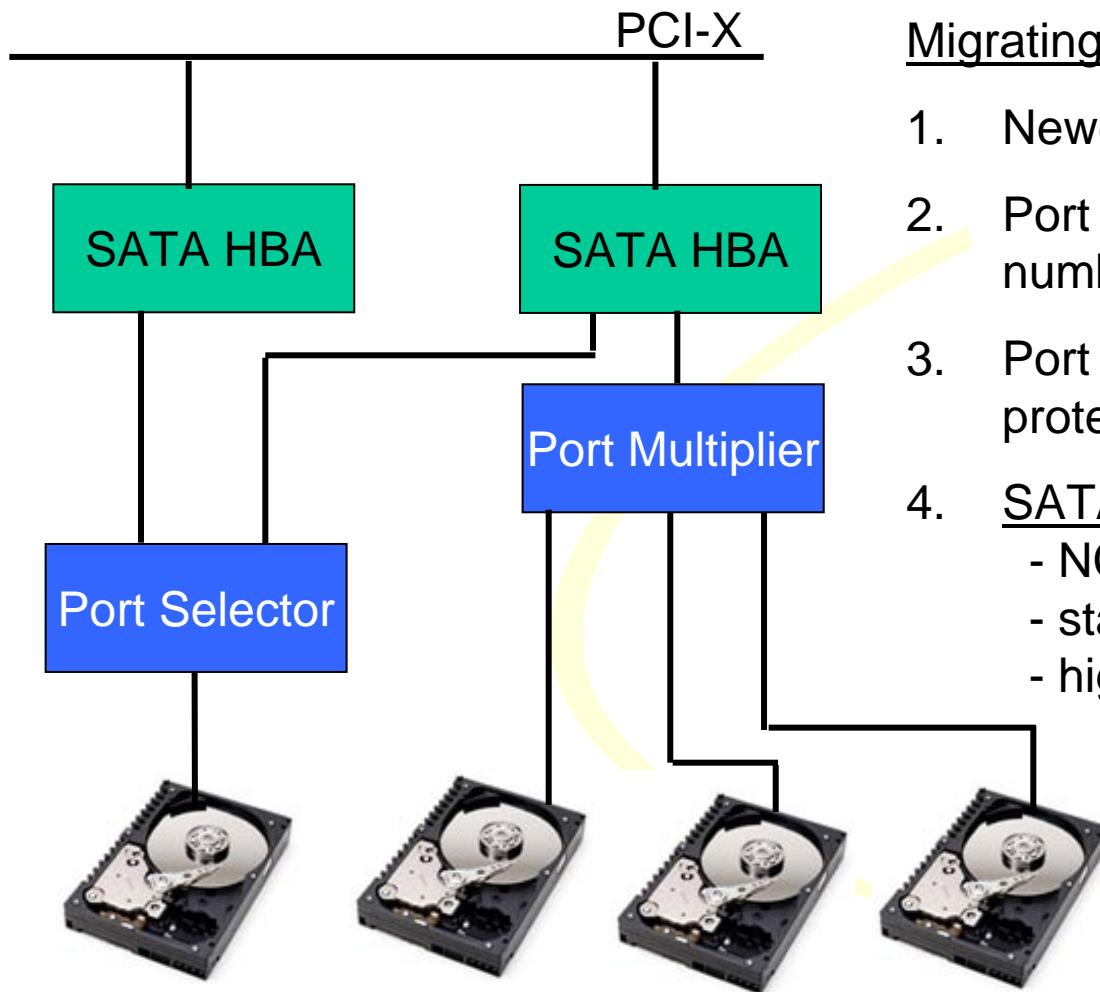
# Topology Comparison (42)

FC	SAS
<p><u>Simple or complex fabrics:</u></p> <ul style="list-style-type: none"> <li>• Cross-connected switches</li> <li>• <math>2^{24} = 16</math> M devices in a fabric</li> <li>• Dual fabrics for no-single-points-of-failure</li> </ul>	<p><u>Simple fabrics:</u></p> <ul style="list-style-type: none"> <li>• Tree of expanders</li> <li>• No loops allowed (they complicate routing)</li> <li>• Initiators perform discovery and program routing tables</li> <li>• Expander sets have maximum of 128 phys</li> <li>• 16 K max addresses in a SAS fabric</li> <li>• Dual fabrics for no-single-points-of-failure</li> </ul>
<p><u>FC supports arbitrated loops</u></p> <ul style="list-style-type: none"> <li>• Often used for disk drive connections</li> <li>• 127 devices in a loop</li> <li>• Dual loops for no-single-points-of-failure</li> </ul>	<p><u>No loops</u></p>
Both are point-to-point serial interfaces, not multidrop busses	

# History of Serial ATA (45)

Generation	Standard	Year	Speed	Key features
Serial ATA	ATA/ATAPI-7	2002	150 MB/sec	
Serial ATA II	ATA/ATAPI-8	2005	300 MB/sec	Native Command Queuing
Serial ATA III	ATA/ATAPI-9?	?	600 MB/sec	

# SATA Topology (46)



Migrating SATA toward the high end:

1. Newer HBAs have more ports,
2. Port Multipliers increase the number of accessible drives.
3. Port Selectors provide failover protection
4. SATA II
  - NCQ reduces latency
  - staggered spin-up
  - higher bandwidth: 300MB/s

# High-End SATA – Transport (47)

- Advantages: simple, inexpensive
  - Due to high-volume consumer market
- Disadvantages: slower, limited SCSI support
  - Half-duplex bus reduces performance (use of RAID can improve this)
  - Inherits ATAPI's limited support for SCSI commands and responses

# High-End SATA - Drives

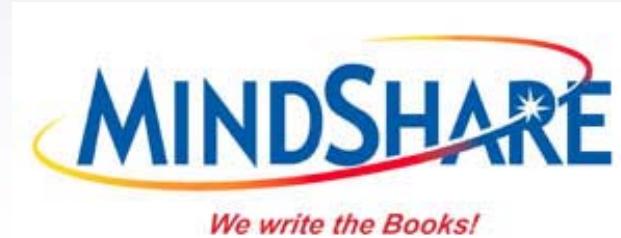
- Advantage: high capacity at low cost
  - Large, low-rpm drives have smaller, less-expensive motors
  - Example of cost savings: using small buffers and simply writing data to media without waiting to receive the whole frame and verify CRC.
- Disadvantages: not dual-ported, lower performance, lower MTBF
  - Use of RAID can mitigate performance and reliability problems. Creates viable near-line storage option for SATA.

# Combining SATA and SAS (48)

- SATA not a good replacement for SCSI
  - Protocol limits enterprise applications; example - ATA provides one 8-bit register for failure information, while a similar SCSI response can be 18 bytes long
- SATA Compatibility with SAS provides:
  - Lower cost of entry by re-using existing technology
  - Migration path for mixed storage requirements

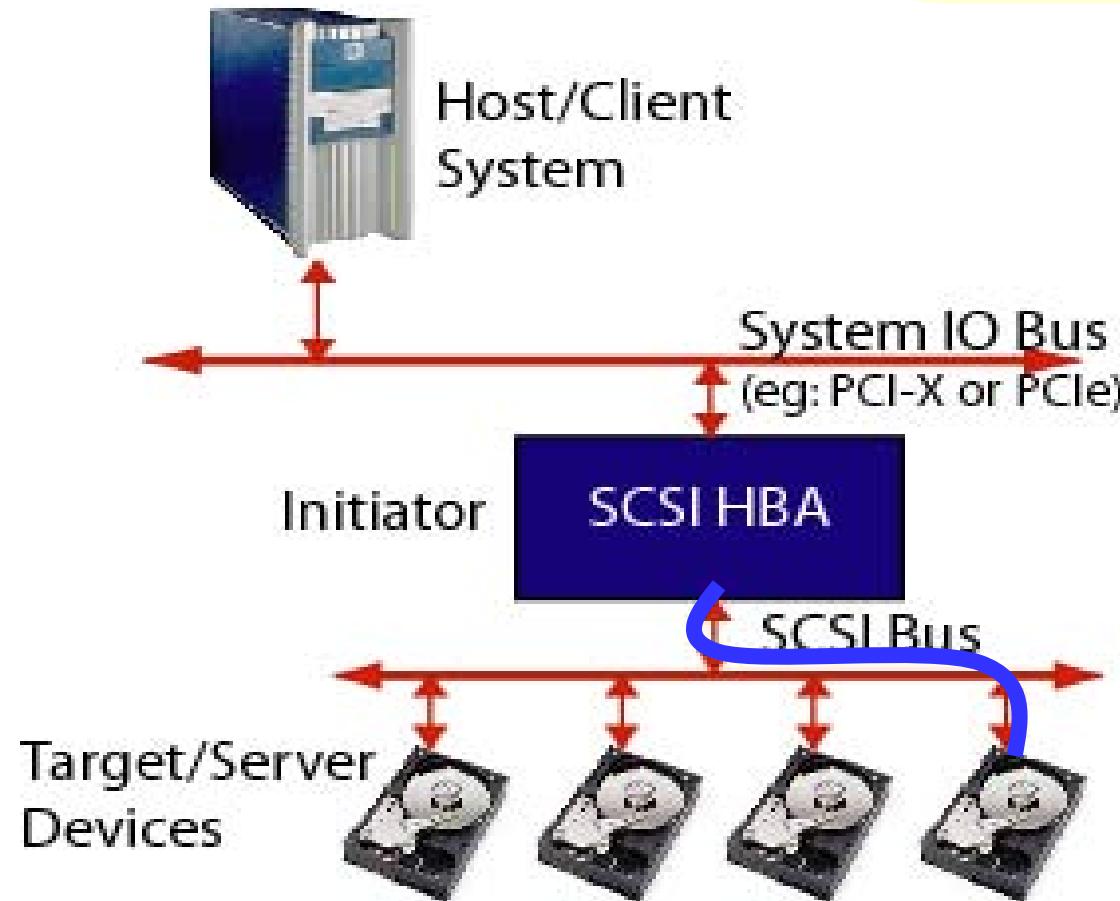
# *Chapter 3*

## *Overview of SAS Layers and Definitions*



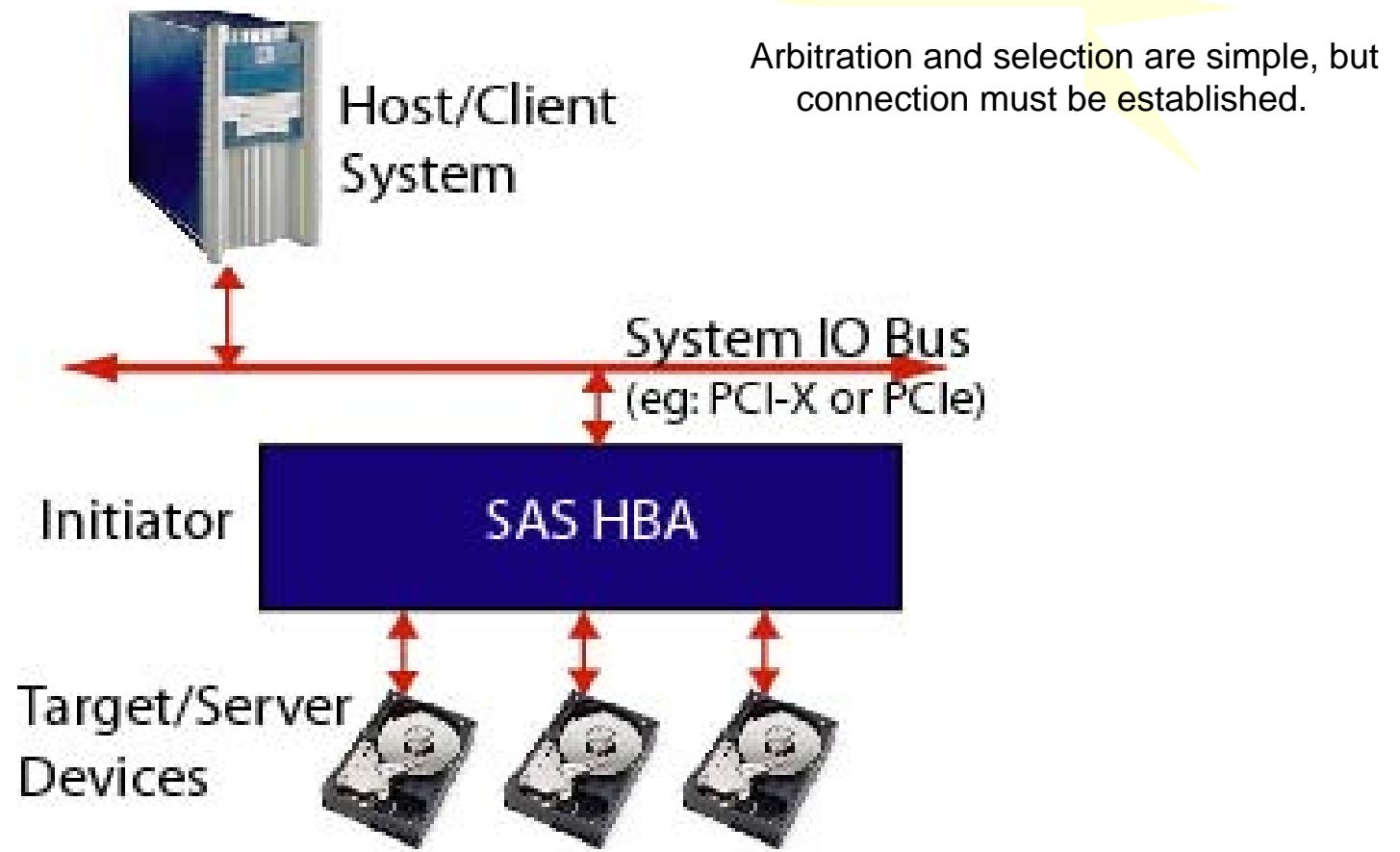
# SCSI Topology (50)

- Shared bus, connection paths limited



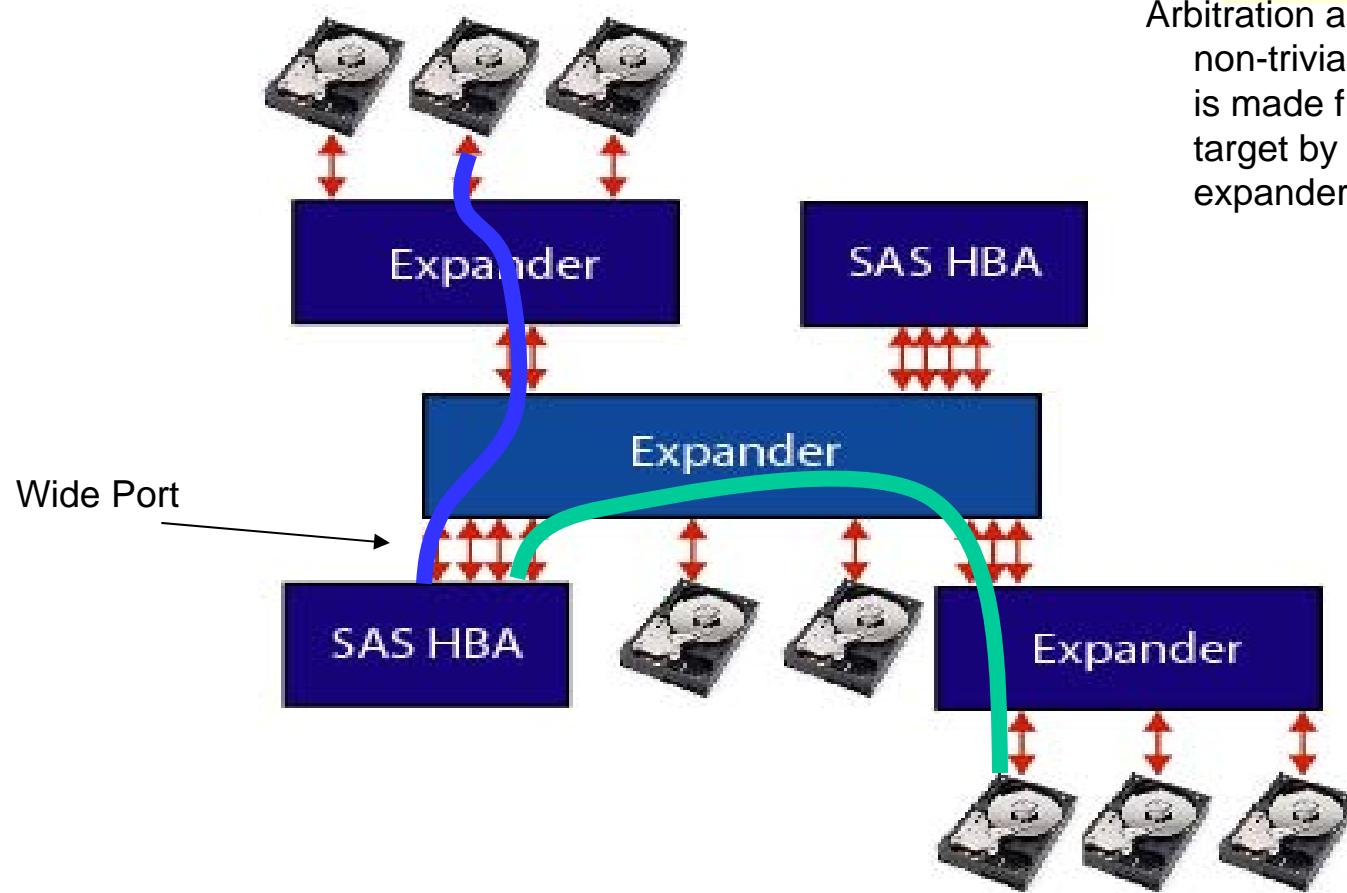
# Simple SAS Topology (51)

- Point to point, more connection path options



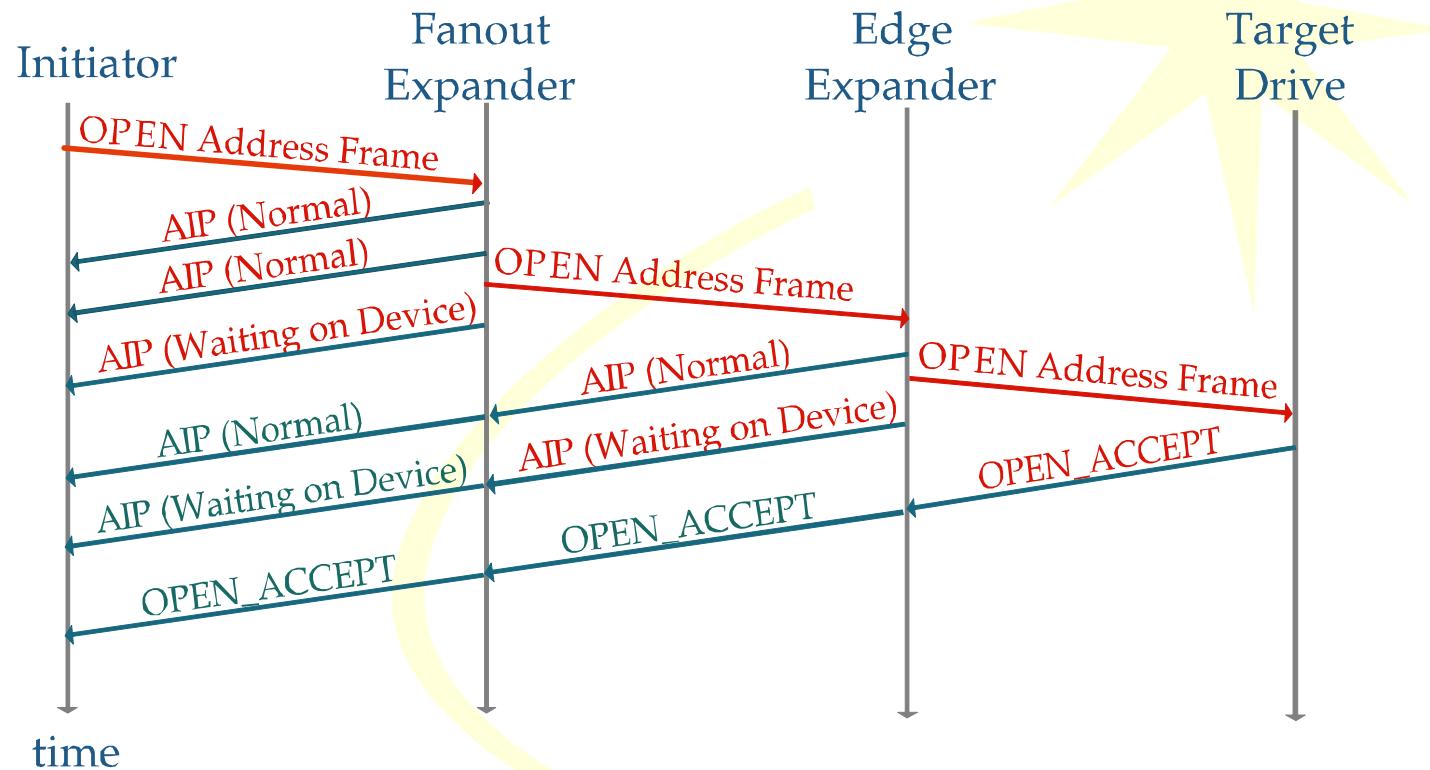
# Complex SAS Topology (55)

- Benefit of wide port: access to more than one device at a time



Arbitration and selection are non-trivial now. Connection is made from an initiator to a target by routing through expanders.

# Connection Request Sequence (59)



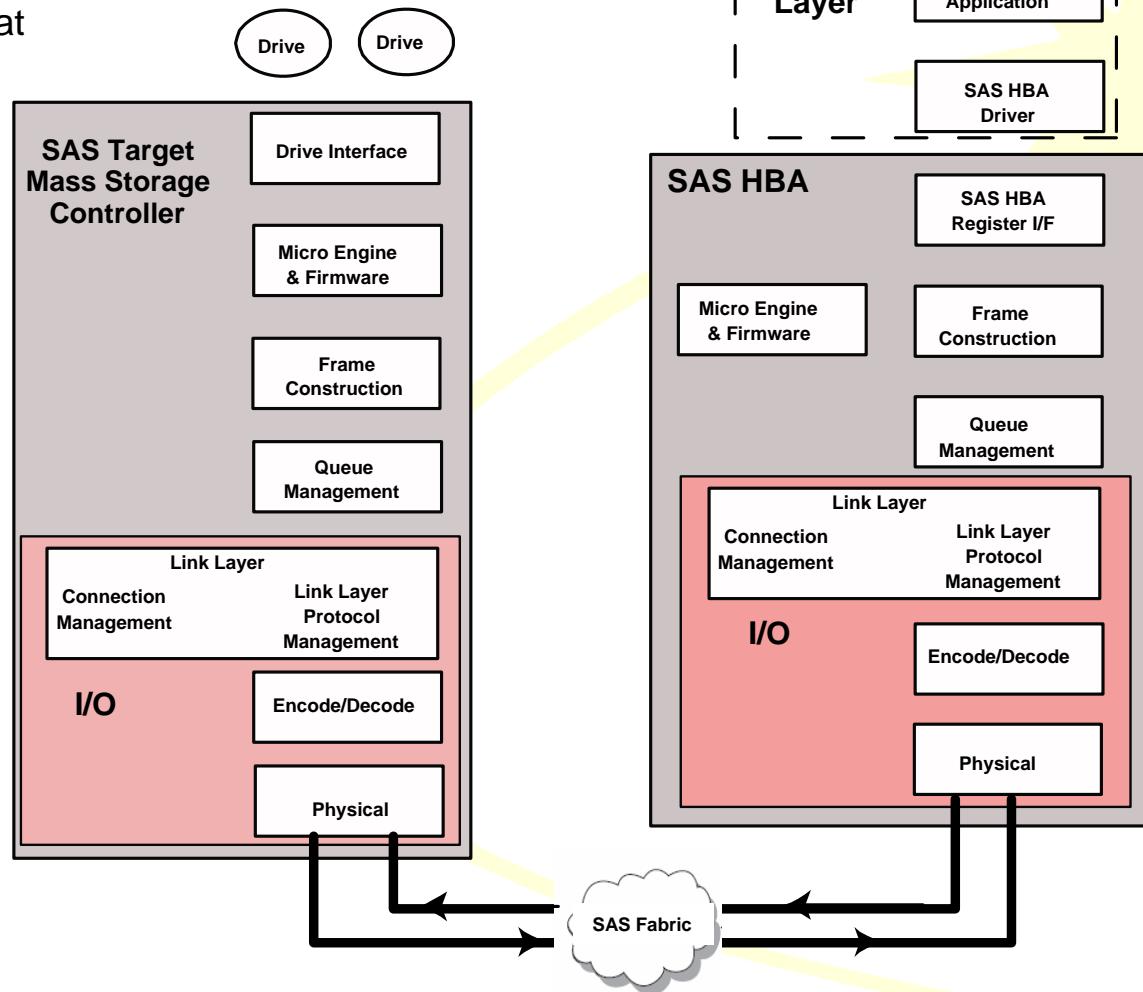
# Dwords and Primitives (63)

- Frames are composed of dwords (4 bytes)
- Primitives are special dwords that are used for control

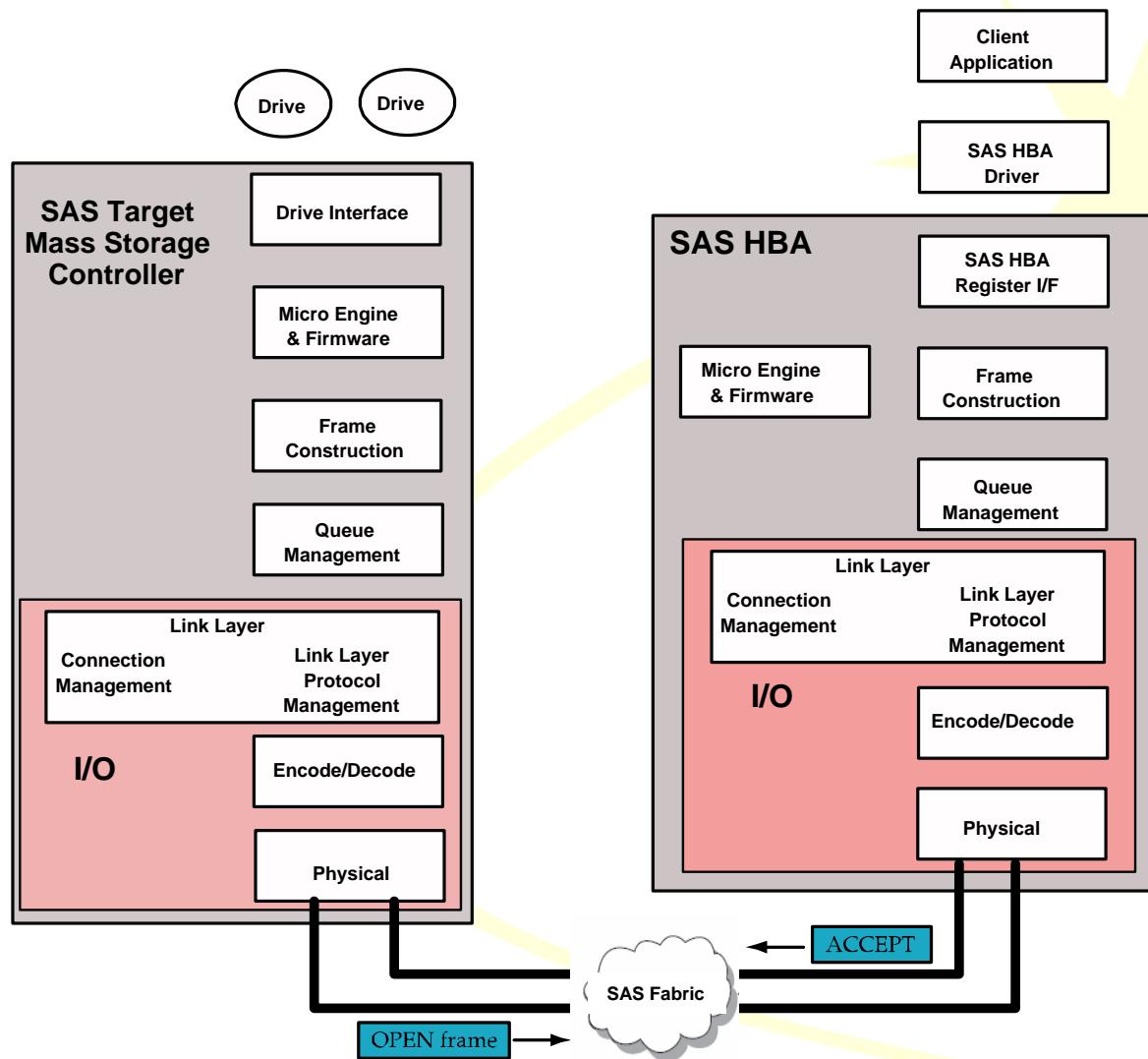
first	second	third	fourth	
K28.5	Dxx.y	Dxx.y	Dxx.y	SAS Primitive
K28.3	Dxx.y	Dxx.y	Dxx.y	SATA Primitive
K28.6	Dxx.y	Dxx.y	Dxx.y	SATA Error Primitive

# Introduction to Operation (60)

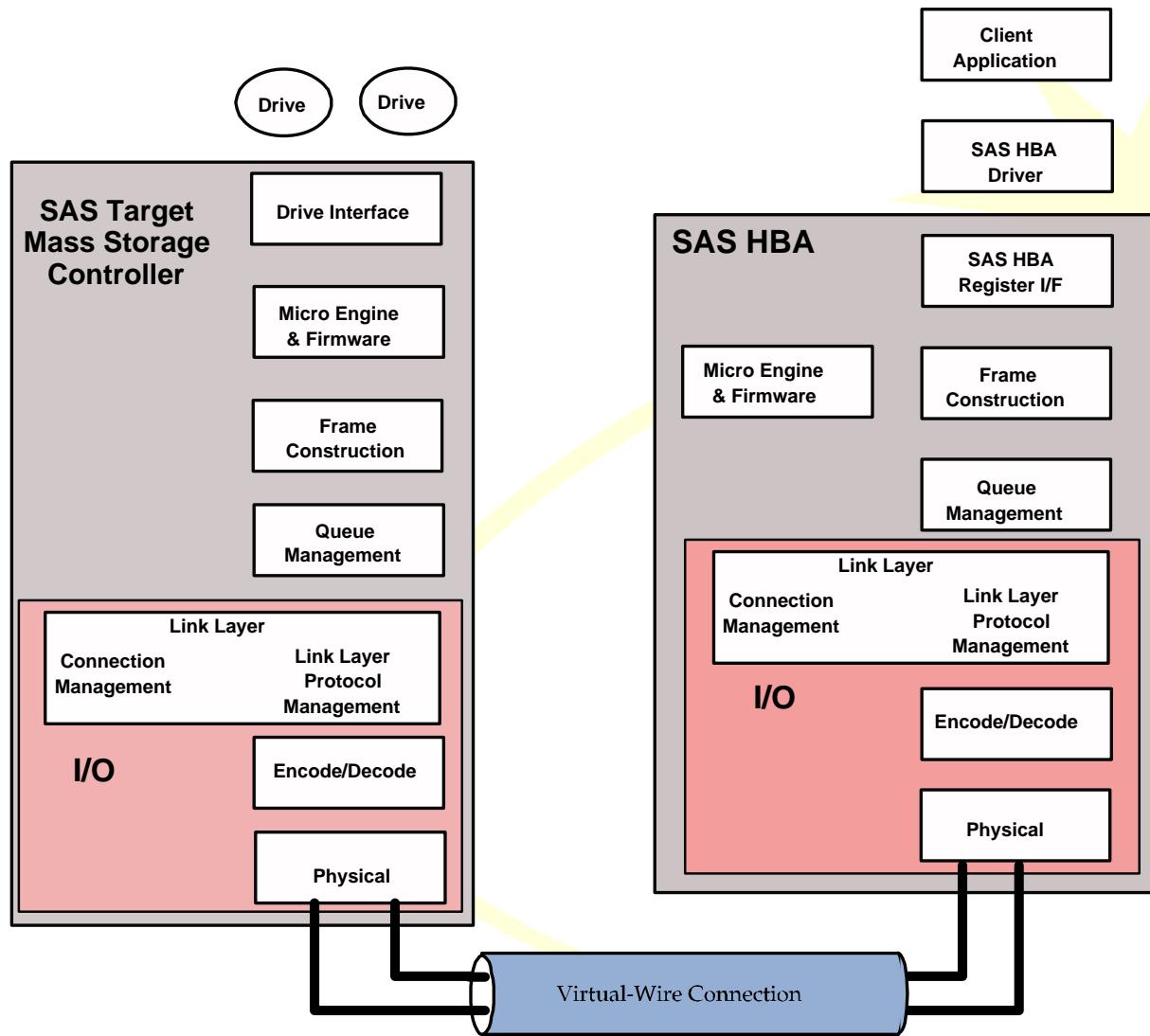
Here, the layers are given  
“user-friendly” names  
that describe what  
they do.



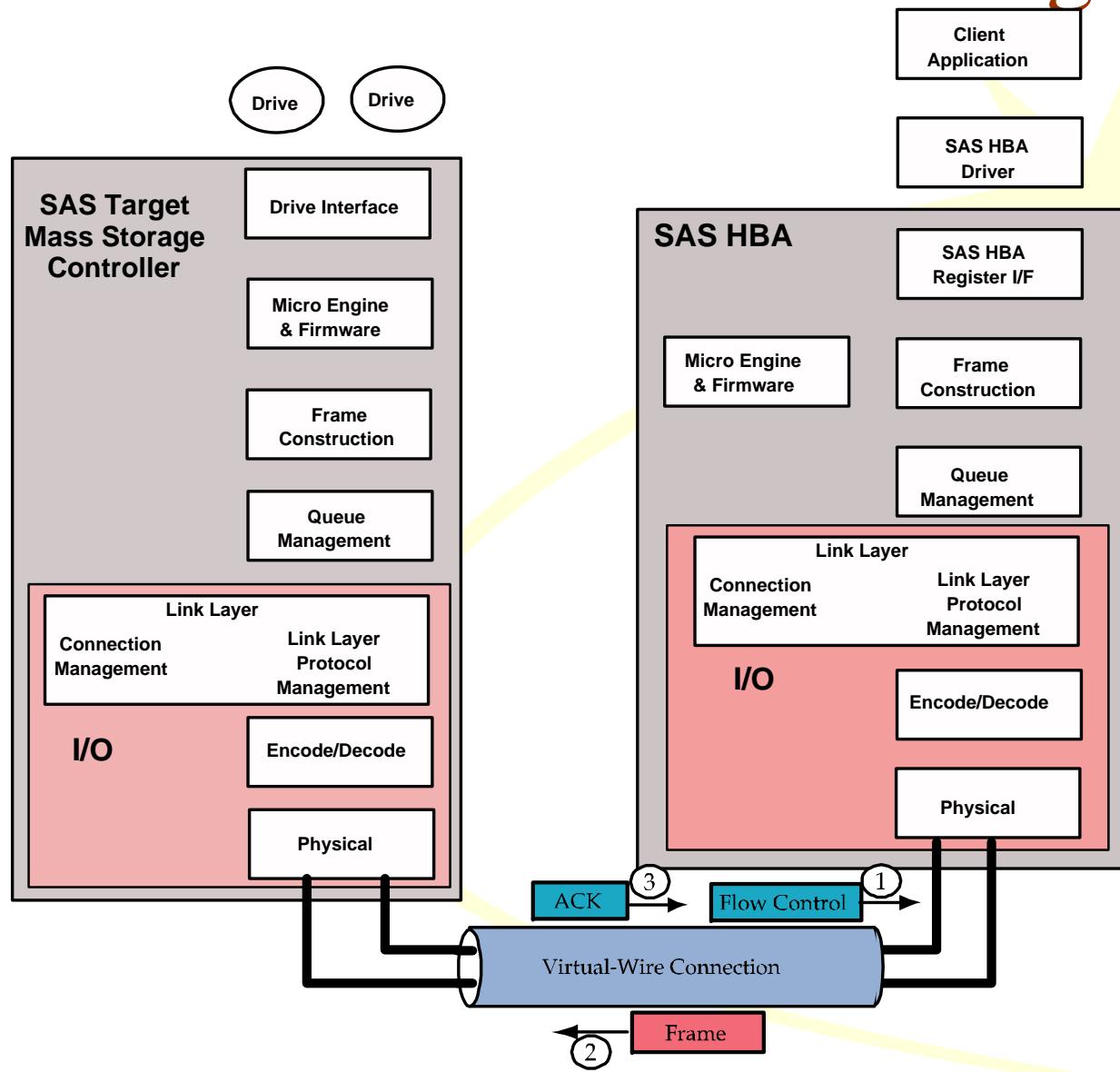
# Connection Request (65)



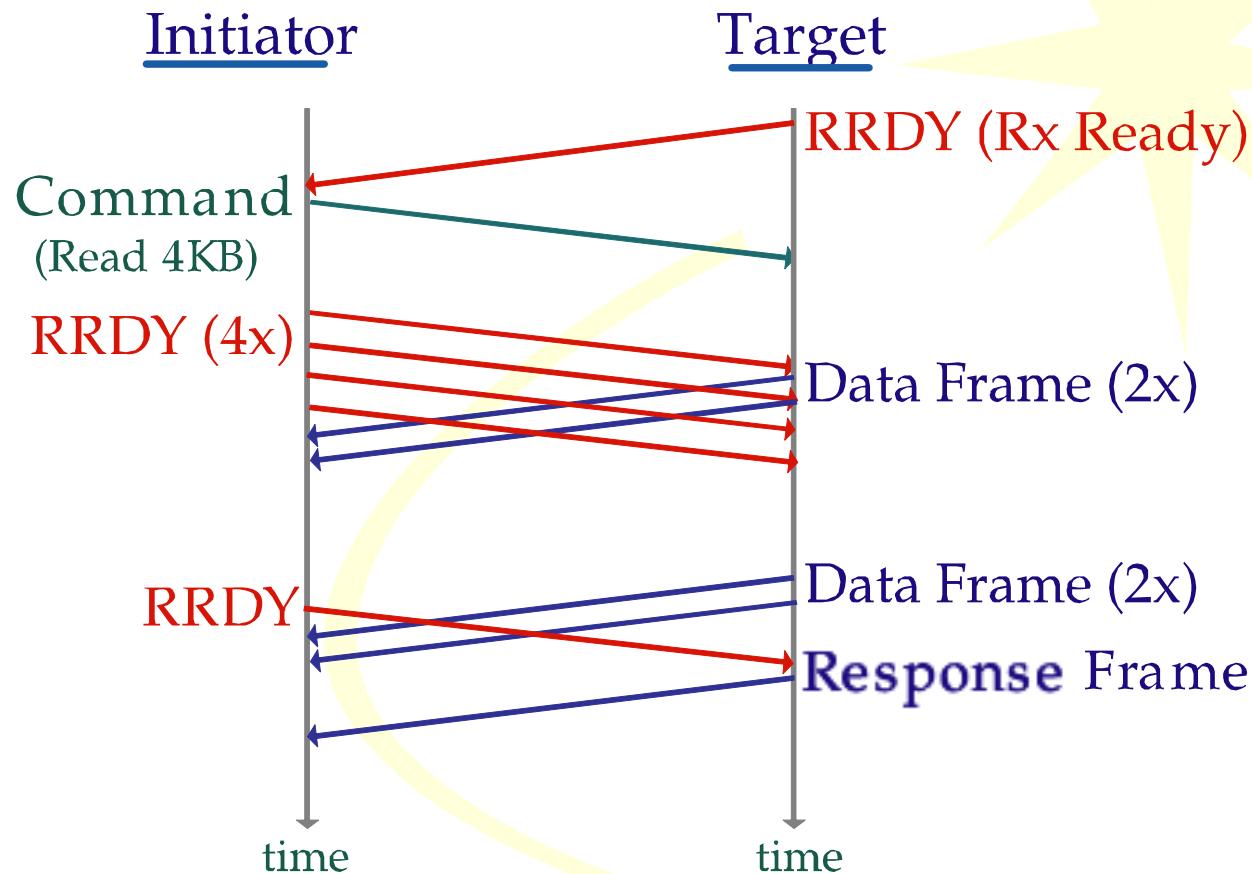
# Connection Established (67)



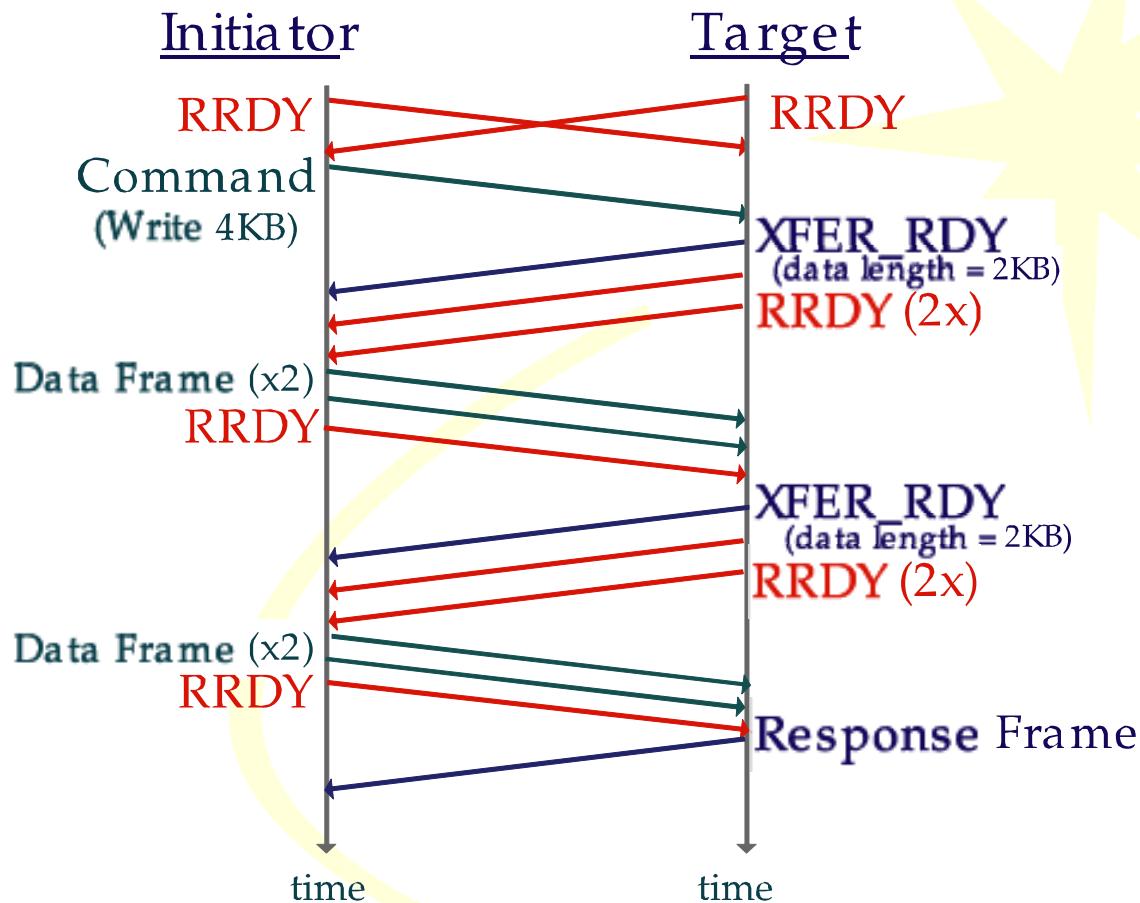
# Command Frame Sent to Target (69)



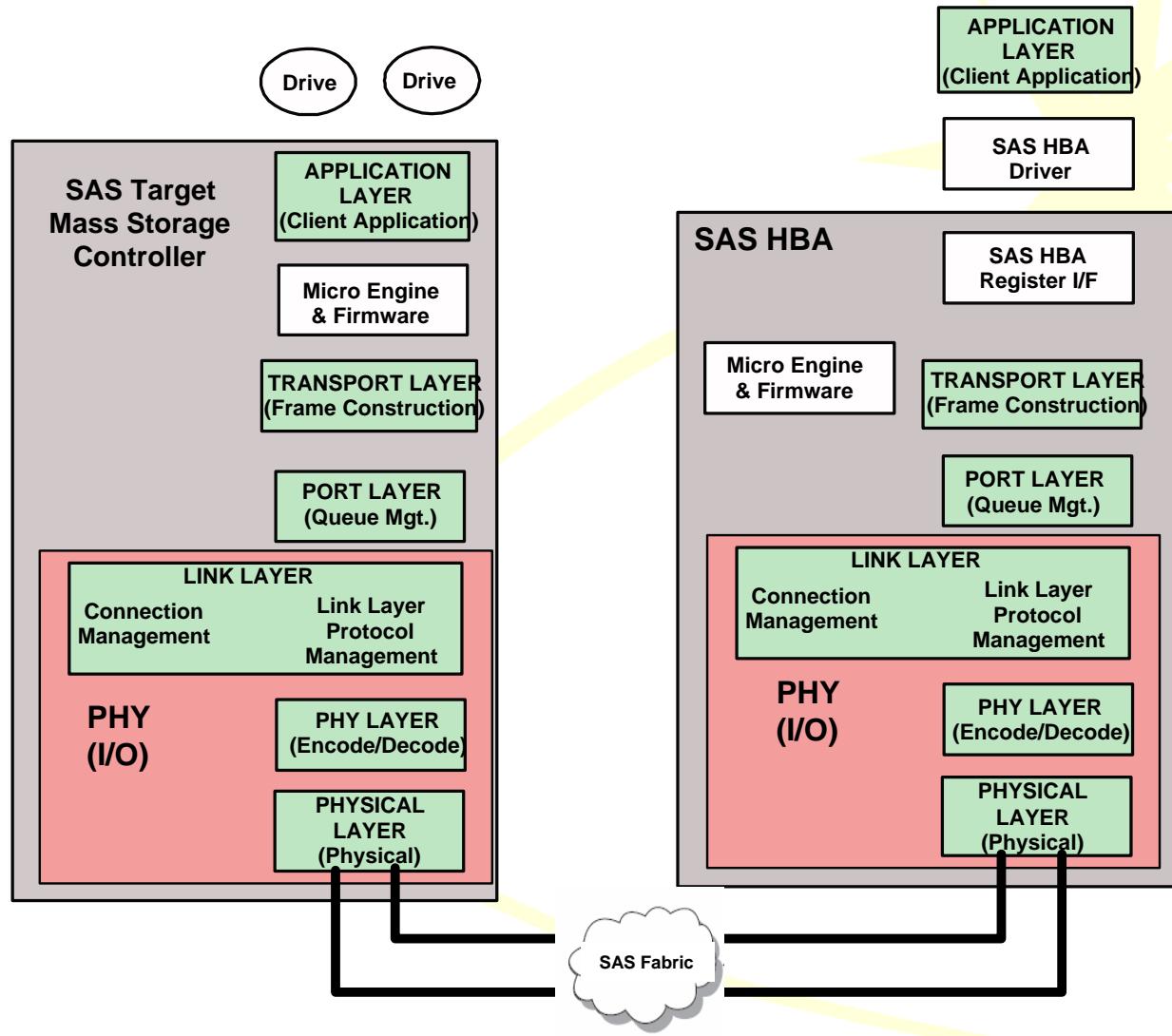
# Read Example (73)



# Write Example (74)

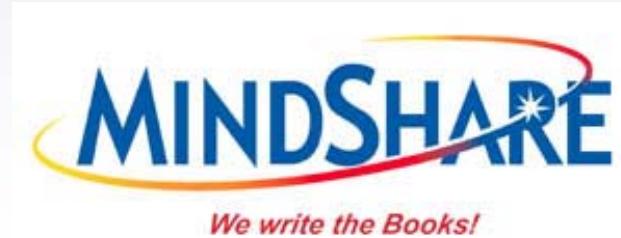


# Actual Layer Names (75)



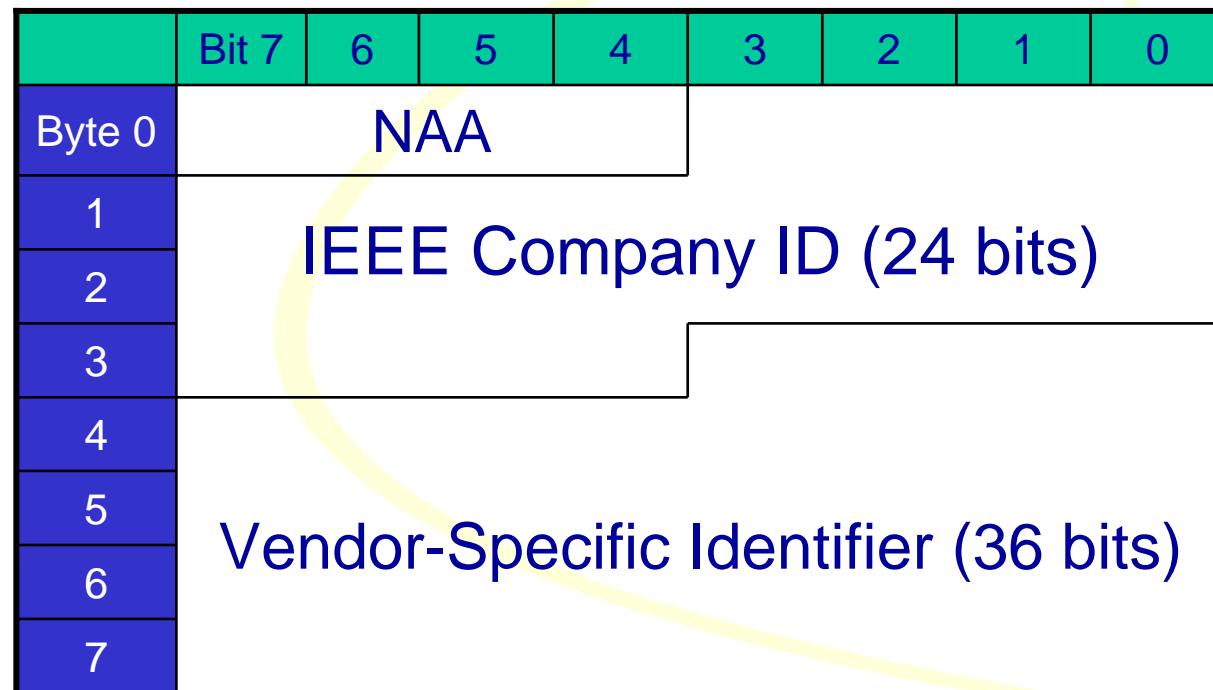
# *Chapter 4*

## *Device Types and Topologies*



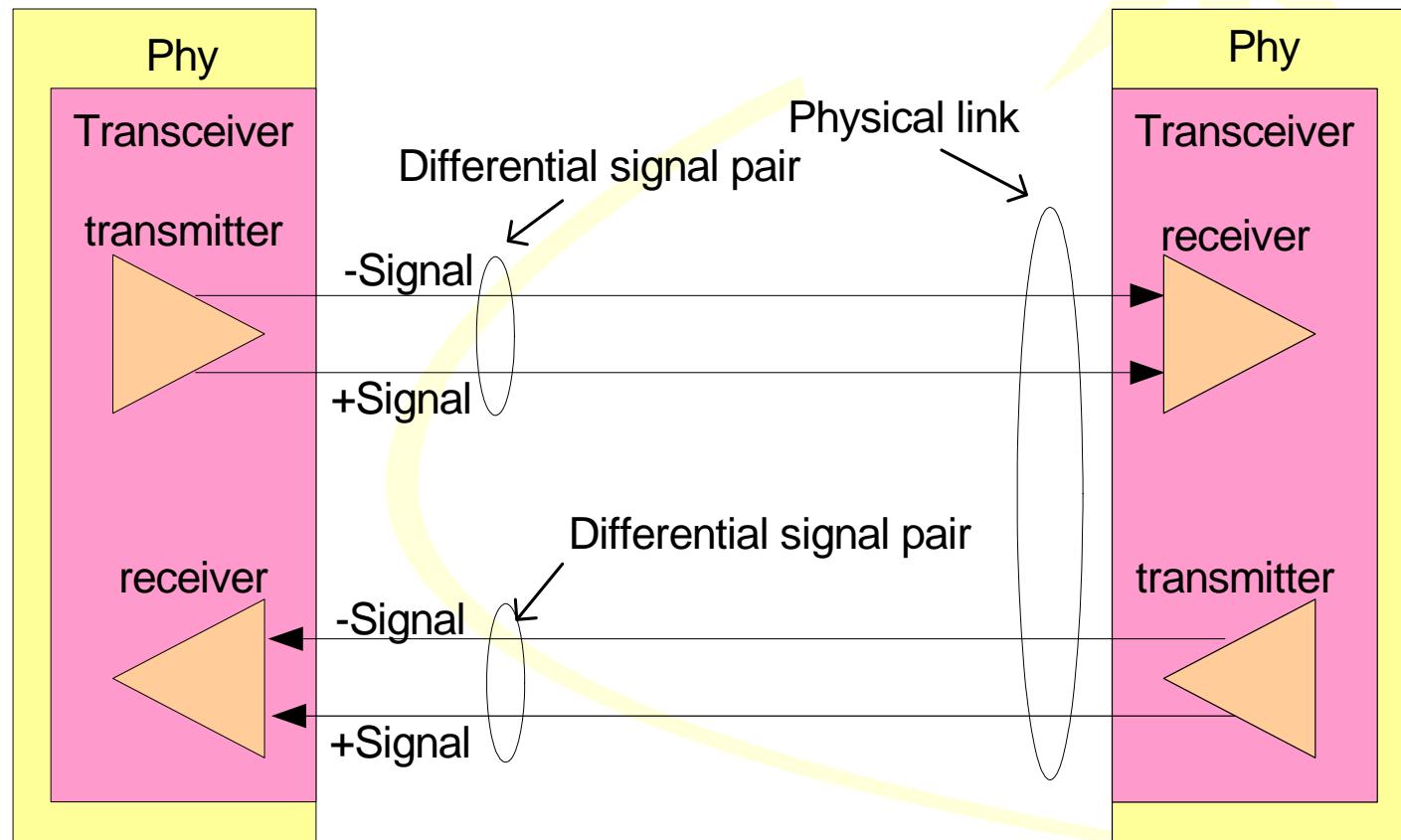
# SAS address

- Each SAS port and expander device has a worldwide-unique, 64-bit SAS address
- Uses the same namespace as Fibre Channel Port Name; NAA value = 5h (NAA is the Naming Address Authority)



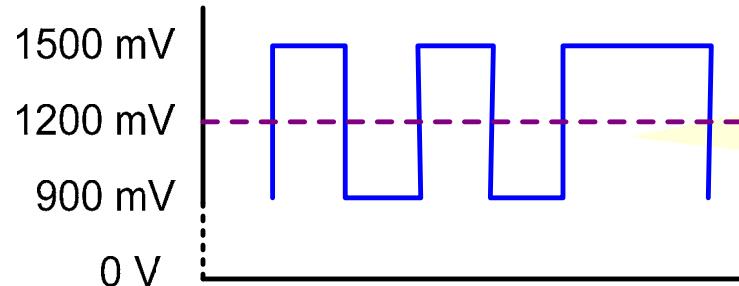
# Physical links and phys

- A phy contains one transceiver
- A physical link attaches two phys together



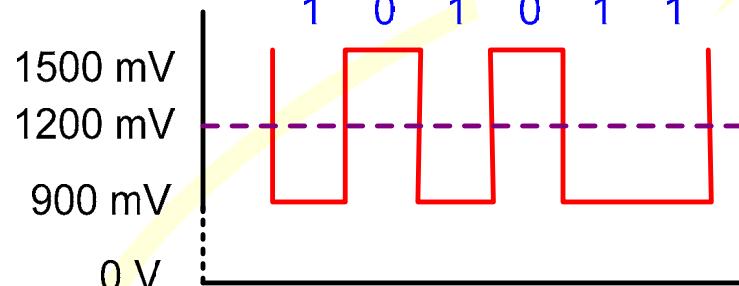
# Differential signaling

Positive signal  
(single-ended)  
(non-inverted)



Common mode  
voltage (the  
level is not very  
important)

Negative signal  
(single-ended)  
(inverted)



Common mode  
voltage

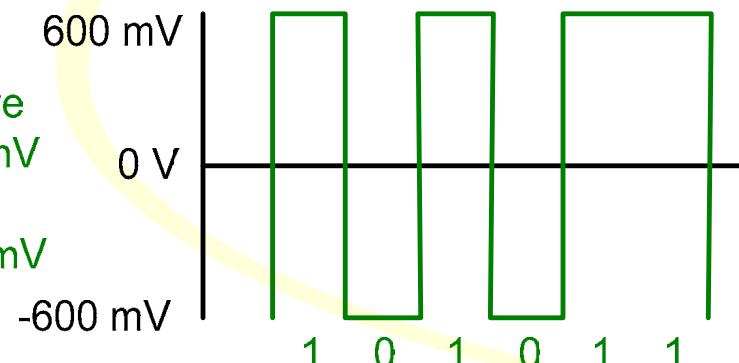
Differential signal

positive - negative

$$1500 - 900 = 600 \text{ mV}$$

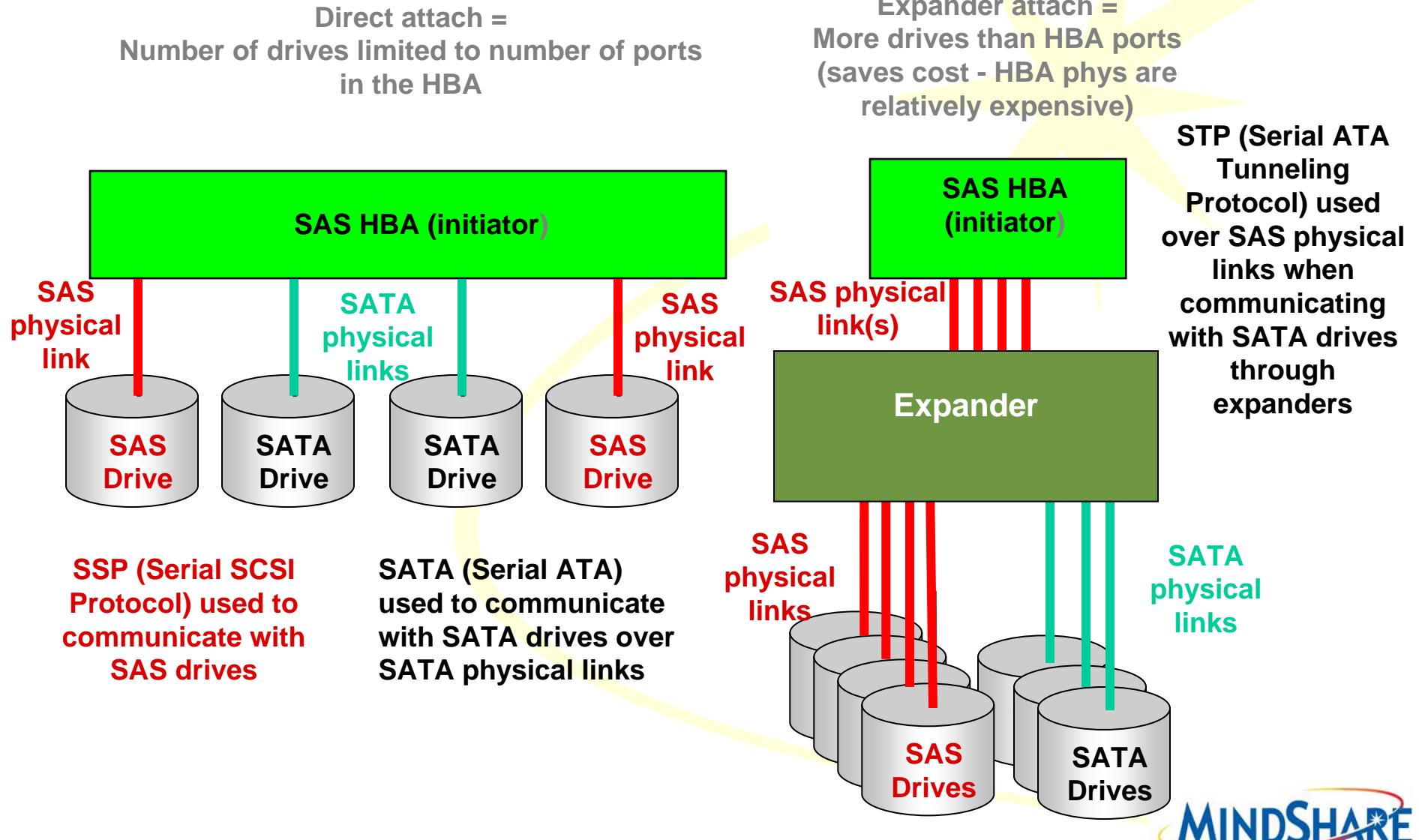
or

$$900 - 1500 = -600 \text{ mV}$$



Differential  
signal is  
immune to  
noise common  
to both single-  
ended signals

# Direct attach and expander attach (82)



# End device

- Two device types: End device and Expanders
- Two types of End device: Initiator and Target
- End device is any SAS device that is not an expander device. Examples:
  - HBA – 8 phys gives the following possibilities:
    - One SAS address for all 8 phys, potentially one wide port
    - One SAS address for 4 phys, different address for other 4
      - Guarantees at least two ports
      - Good match for 4-wide connectors
    - Eight different SAS addresses
  - Disk drive - 2 phys
    - Separate SAS address for each phy
      - Guarantees two ports
      - Never a wide port

# Port Description

- Port – a virtual construct in SAS devices
  - Port is associated with a SAS address and contains one or more phys
- Definition: Phys with the same SAS address attached to another set of phys that also all share a unique address.
  - Ports change with topology; determined during initialization (see following slides)

# Wide Ports Improve Data Rate

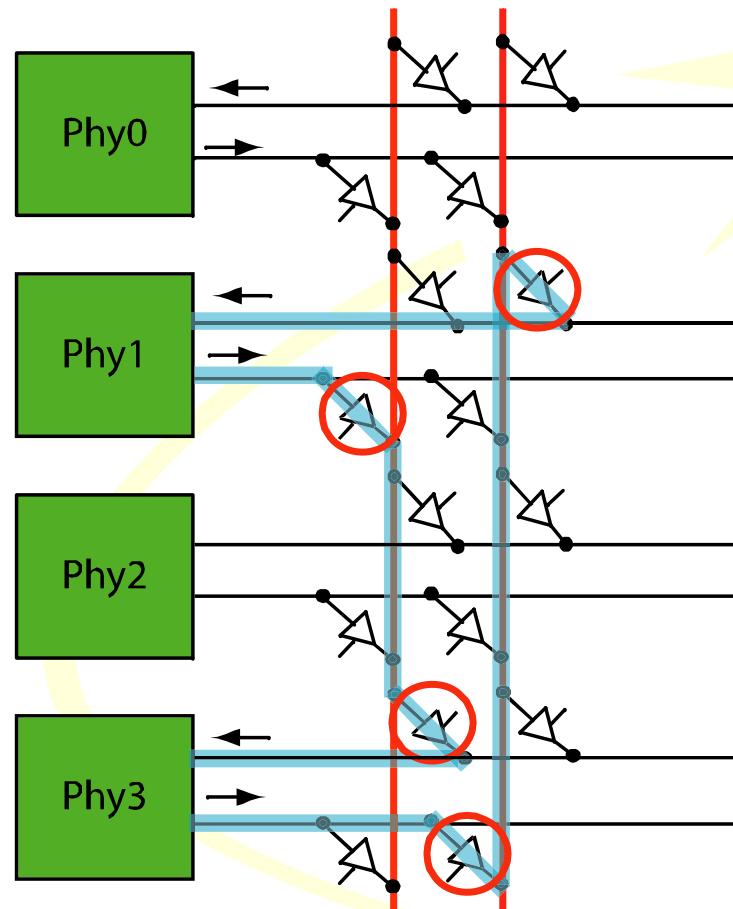
- More phys increase available bandwidth
- Example: peak bandwidth of an HBA with 8 phys running at 3.0 Gbps is 2400 MB/s using 8 separate connections in progress. If traffic is flowing in both directions, the overall BW would be twice that, or 4800 MB/s.

# Expander Types – Edge vs. Fanout

- Edge expander device
  - Always part of an “edge expander device set”
  - May perform subtractive routing (using a default path when destination address cannot be resolved)
- Fanout expander device
  - Never does subtractive routing; tracks and resolves all addresses
  - Supports larger tables for table routing

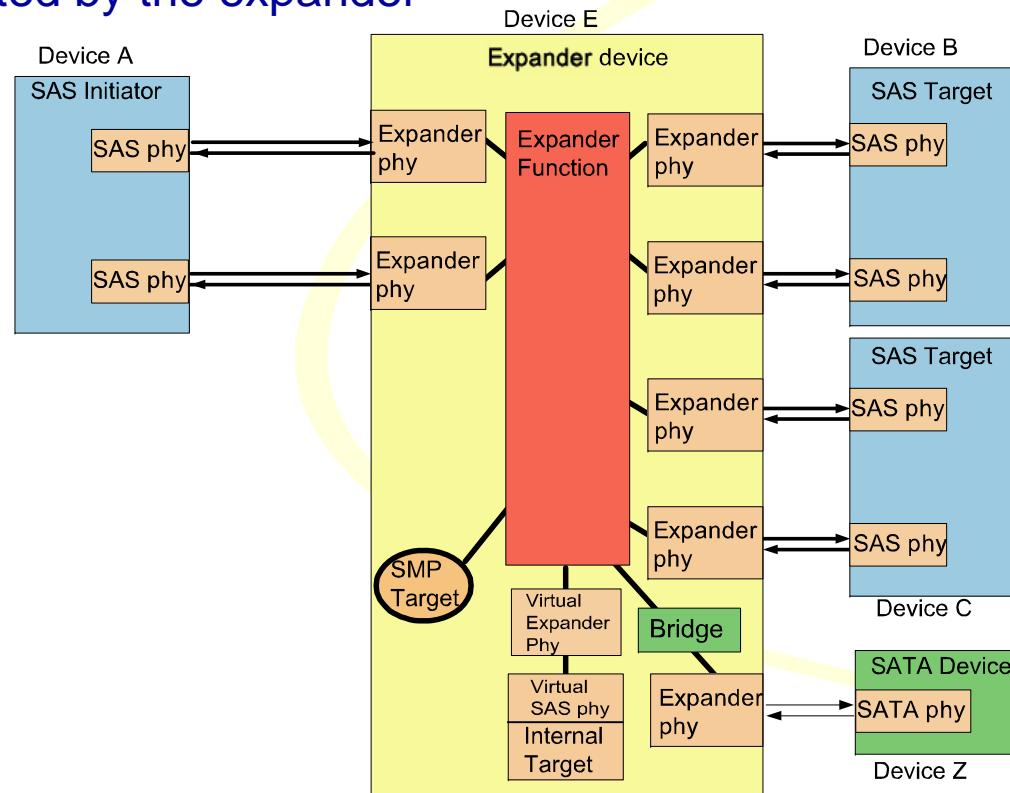
# Expander Crossbar Connection Path (84)

Main function of expanders is to make connections between its phys. Internally, a crossbar switch accomplishes this purpose.



# Expanders (85)

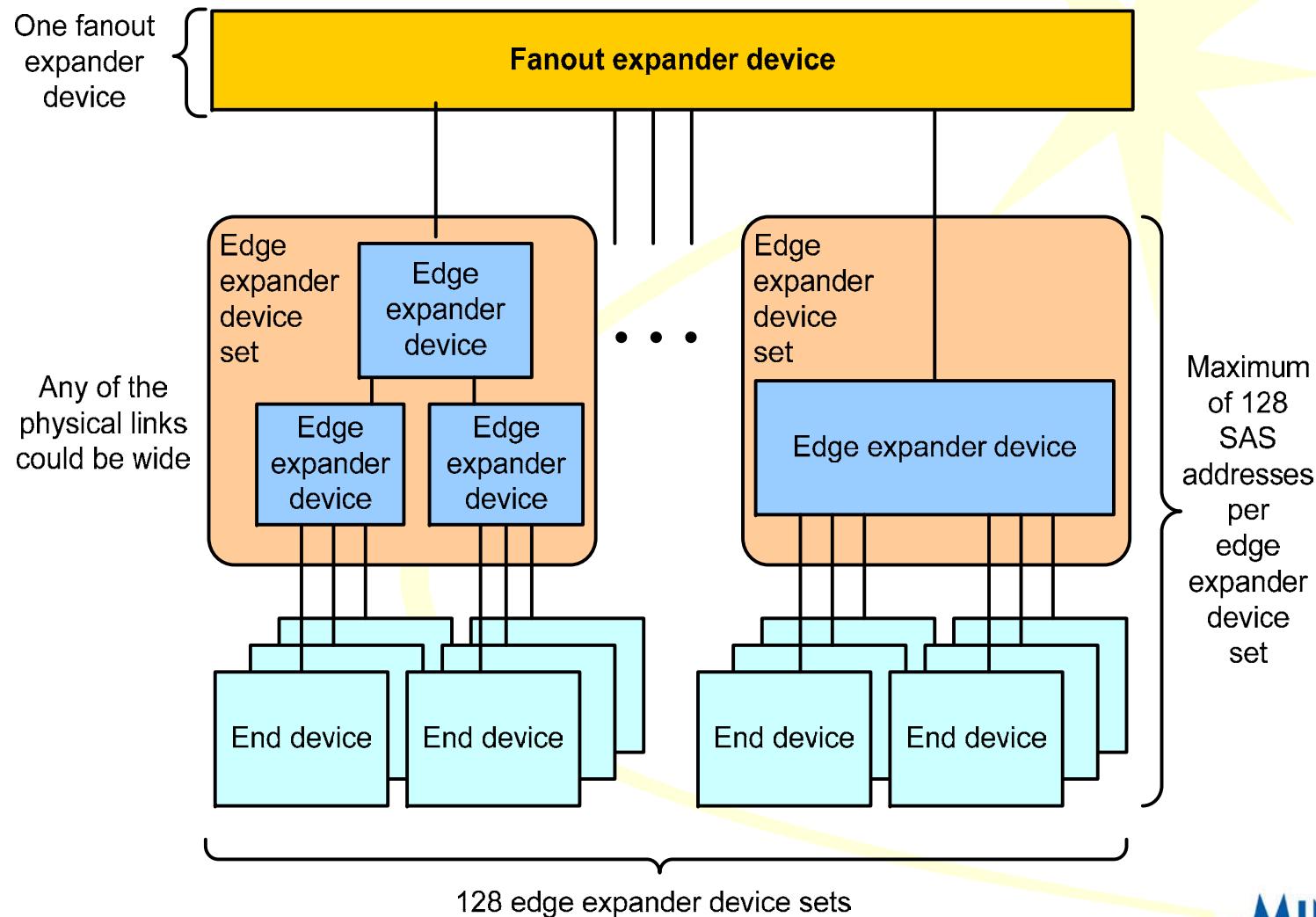
- Contain expander ports and a crossbar switch
- Must contain SMP port for configuration, may contain internal virtual SAS phys (e.g.: for enclosure management functionality)
- Each expander phy has same SAS address, but a unique phy identifier (SMP port doesn't have phy address)
- Virtual internal phy and SATA bridge are optional – their addresses are created by the expander



# Edge expander device set (86)

- Group of edge expander devices
- 128 SAS addresses max for the set
- Typically bounded by a subtractive port (to a fanout expander device, or to another edge expander device set)
- Edge expander devices uses table routing, direct routing and subtractive routing
- Wide links between expanders are allowed
- No loops are permitted

# Edge Expander Sets and Fanout Expander Device (86)



# Fanout Expanders (88)

- There can be at most only one fanout expander in a SAS domain
- If no fanout expander, there can be at most only two edge expander device sets (attached to each other via subtractive decode ports)
- End devices may be attached at any level
- Wide links possible between any two devices
- No loops or multiple paths permitted

# SAS Domain (89)

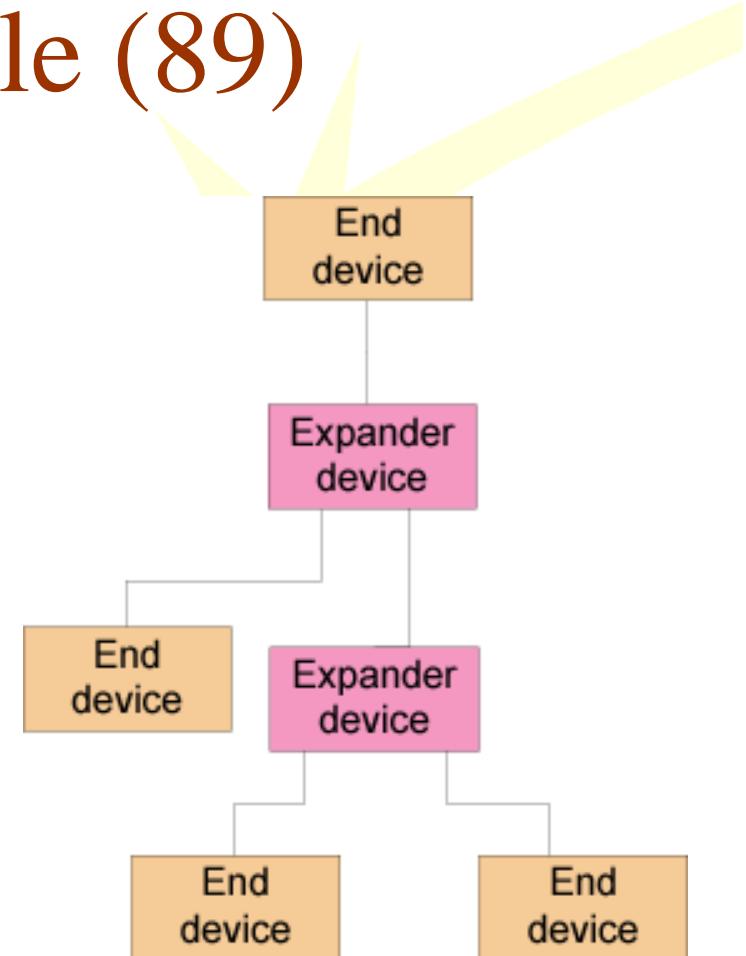
- SAS Domain consists of two basic parts
  - End Devices:
    - Initiators – Clients, devices that request service
    - Targets – Servers, devices that respond to requests (i.e.: drives)
  - Service Delivery Subsystem: everything that connects the end devices to each other and allows them to communicate
    - Cables, traces
    - Expander devices (similar to network switches)

# Domain Concepts

- Loops not allowed
- SAS Domain: Contains SAS devices and Expander devices
- SATA Domain: Only contains a SATA host and a SATA device

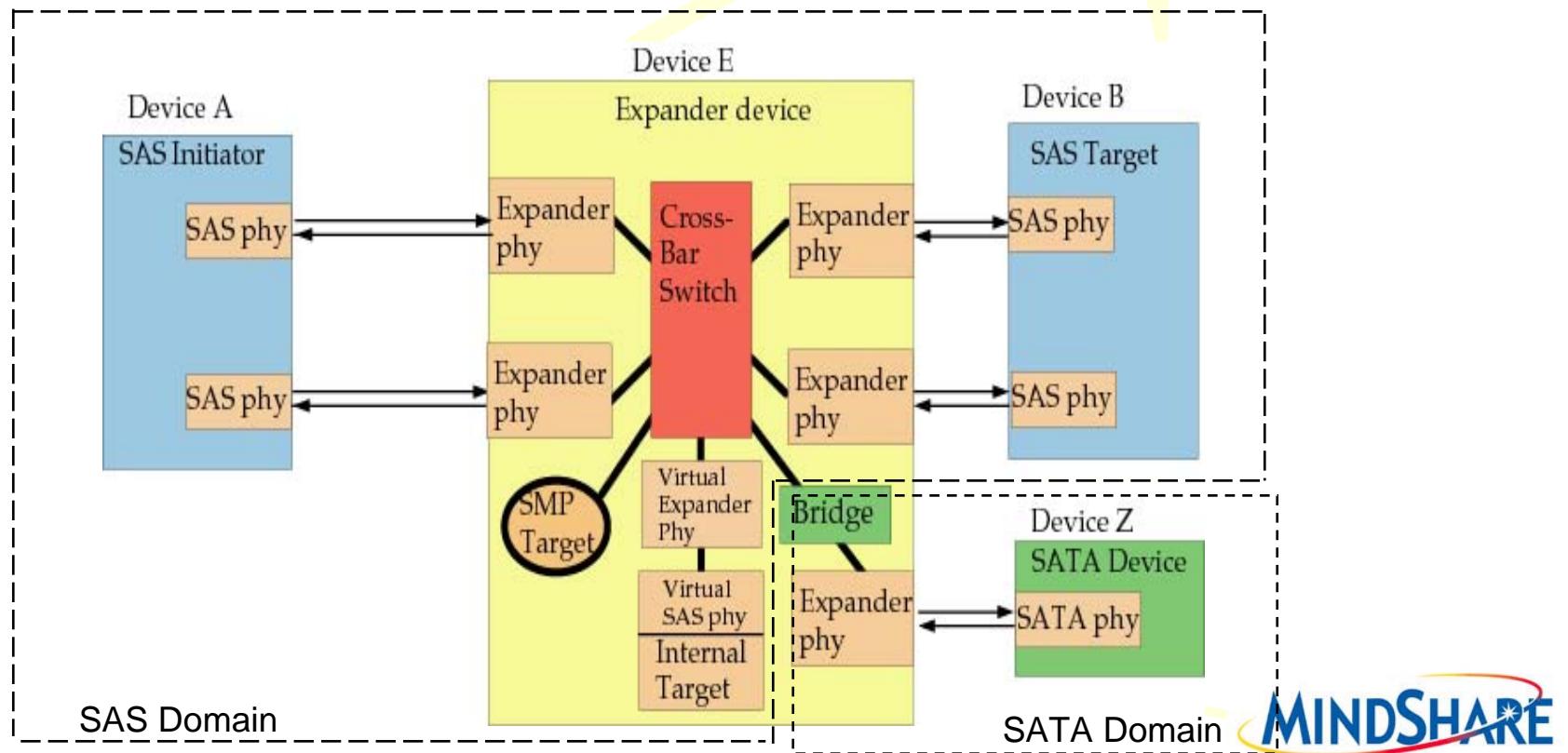
# Domain Example (89)

- SAS Domain contains devices that can communicate with each other
  - Note that all devices linked to an expander are in the same domain; expander can only reside in one domain



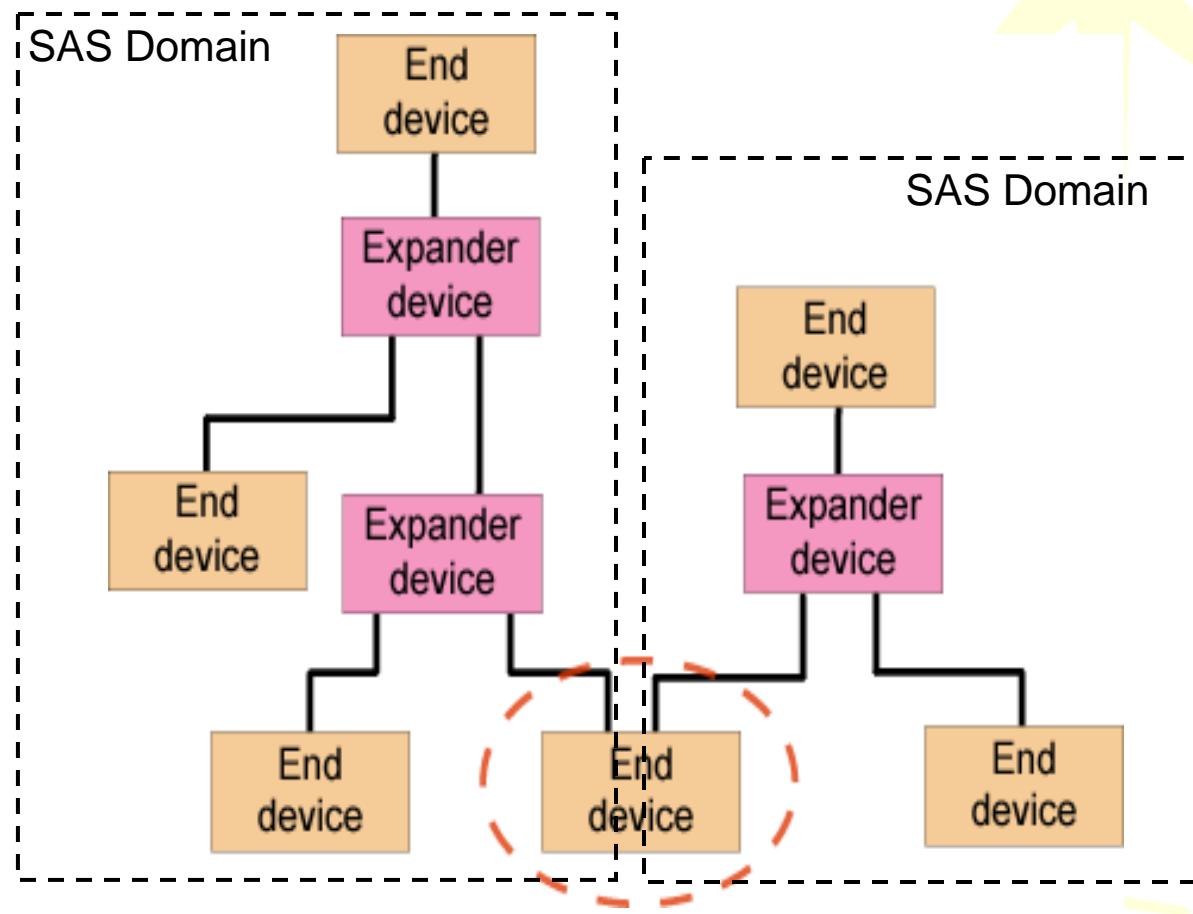
# Domain Example (90)

- SAS Domain bridged to SATA Domain
  - SATA Domain contains SATA host and device
  - This is referred to as the SCSI-to-ATA Translation layer (SATL)



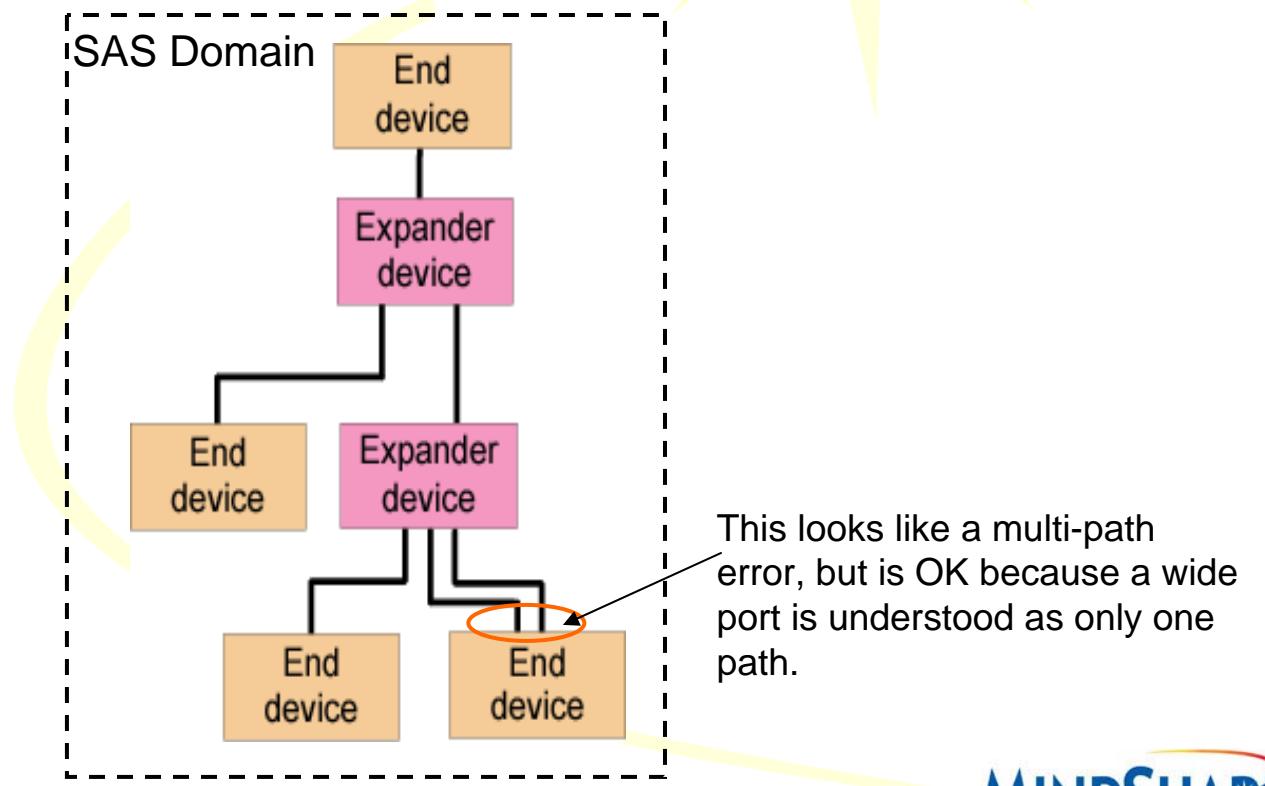
# Domain Example (91)

- SAS device can reside in multiple SAS domains if it has multiple ports



# Domain Example (92)

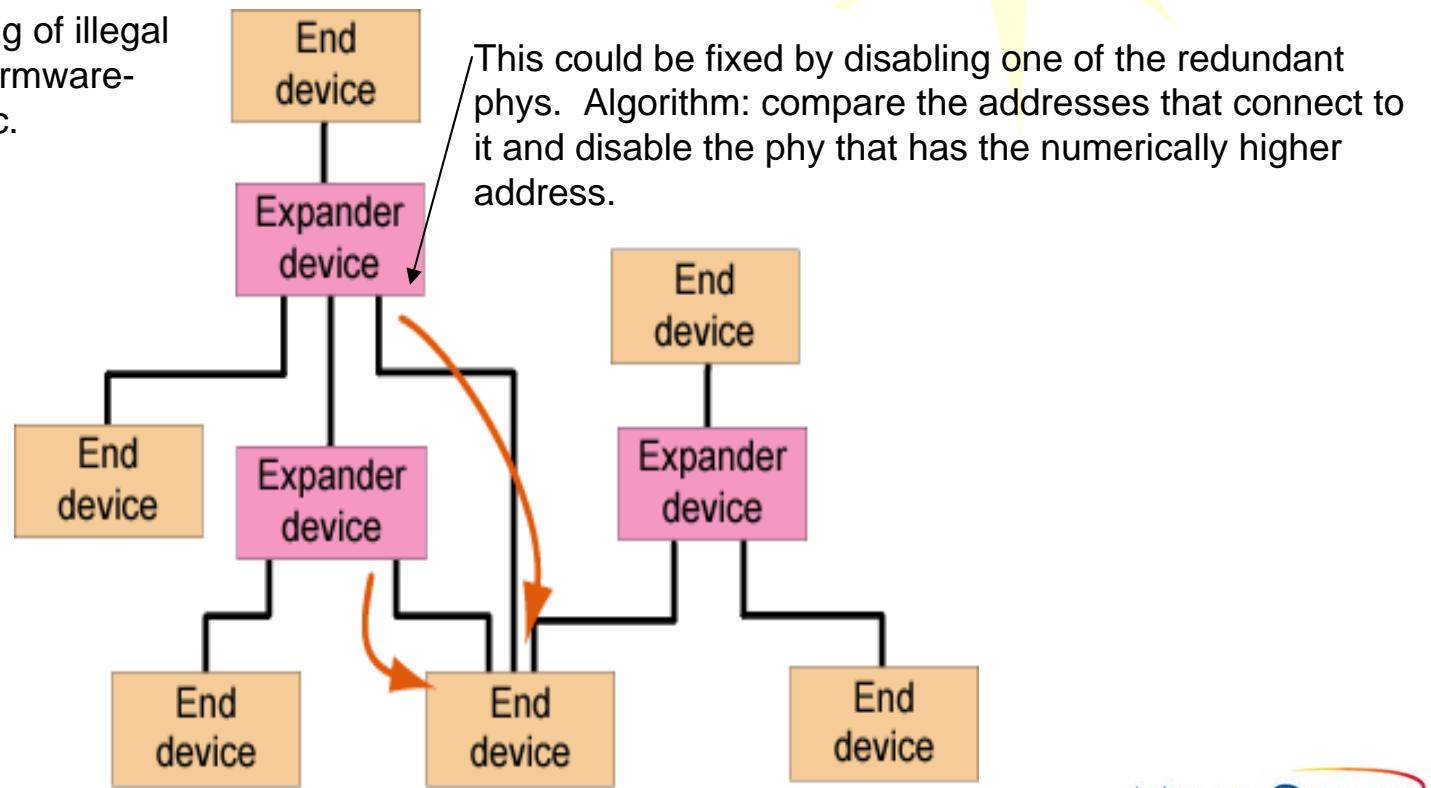
- Wide-port device can also reside in the same SAS domain, where the extra port provides redundant path or better bandwidth



# Domain with Multi-Path Error (92)

- Expander is not allowed to see more than one path to the same device if one includes another expander – illegal topology

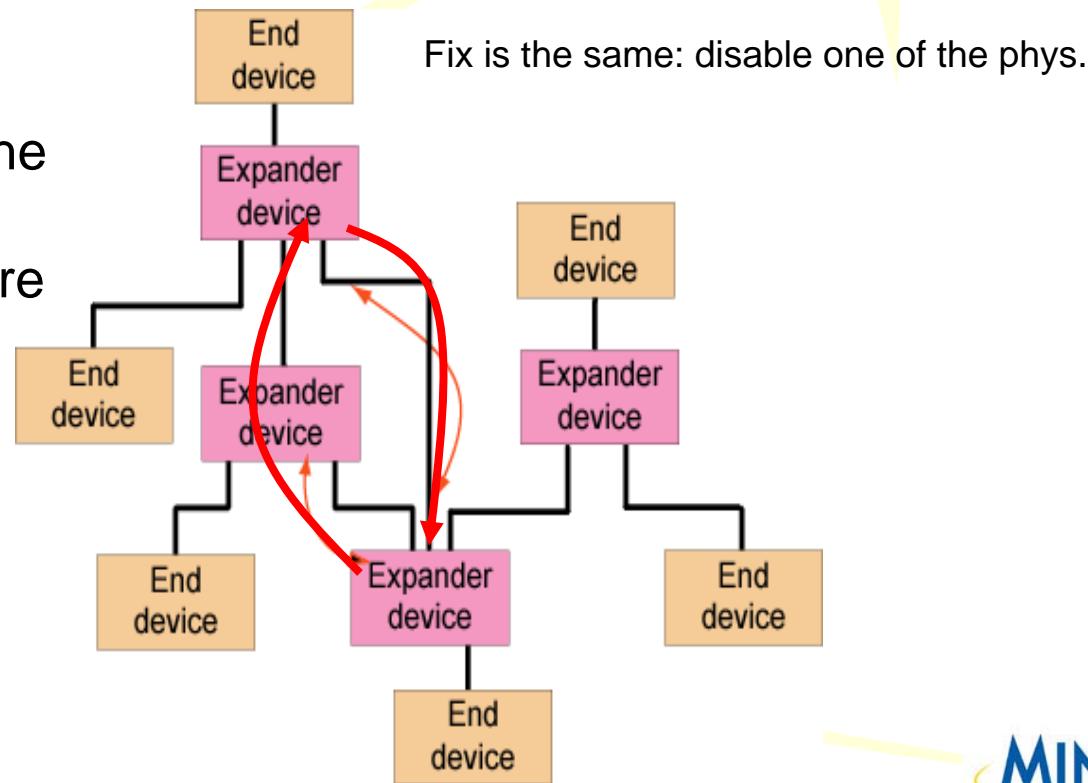
Reporting and handling of illegal topologies will be firmware- and vendor-specific.



# Domain with Loop Error (93)

- Expanders connected in a loop create an infinite loop when changes occur and one informs the other of a change and then sees another change notification coming back in response

This has to do with the way notification of domain changes are handled.



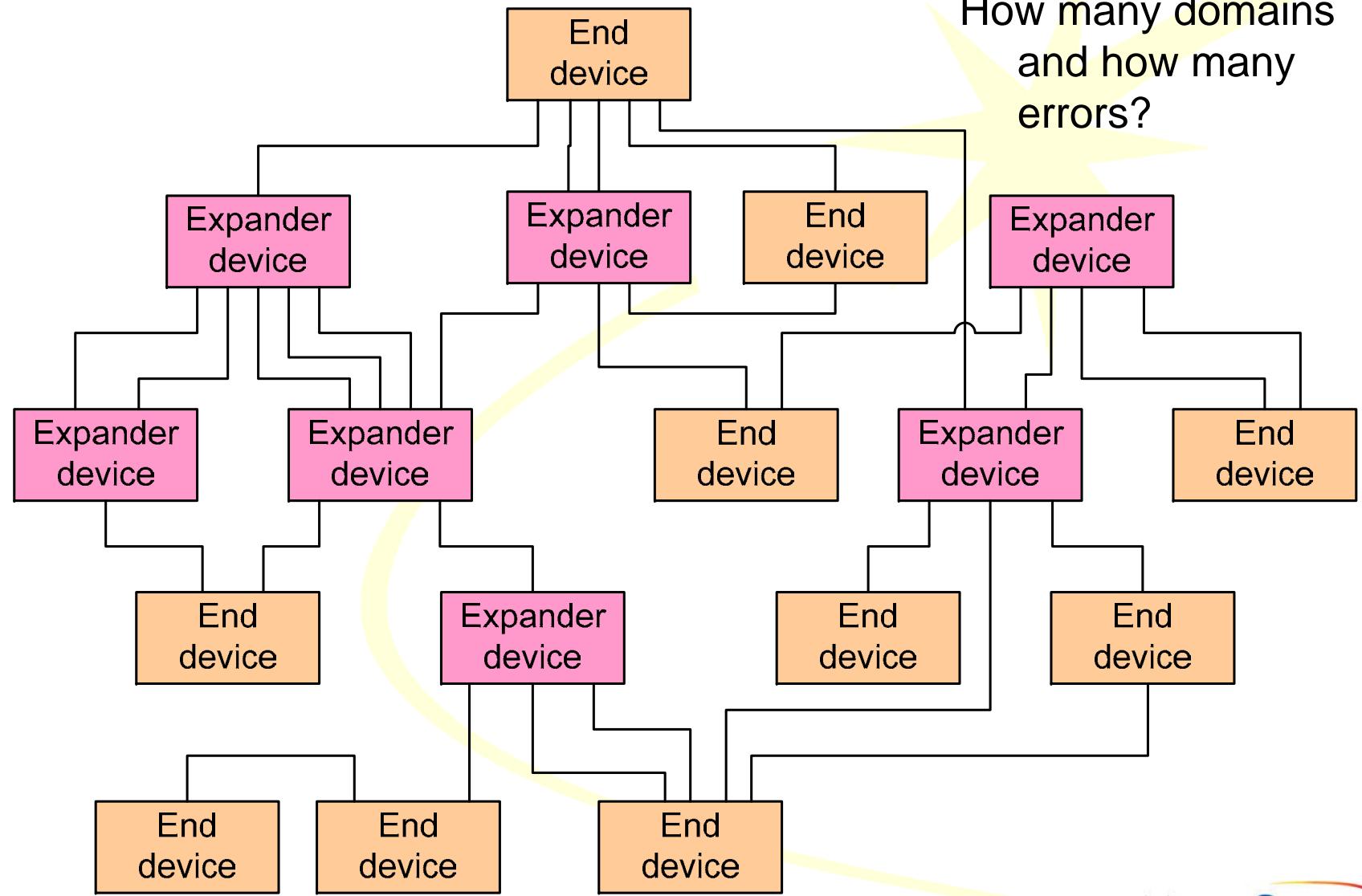
# Other Domain Rules

- Any endpoint-to-endpoint connection is always a separate domain
- Acceptable for an end device to see more than one path to another end device, but not for expanders to have a choice about paths

# Domain Quiz (94)

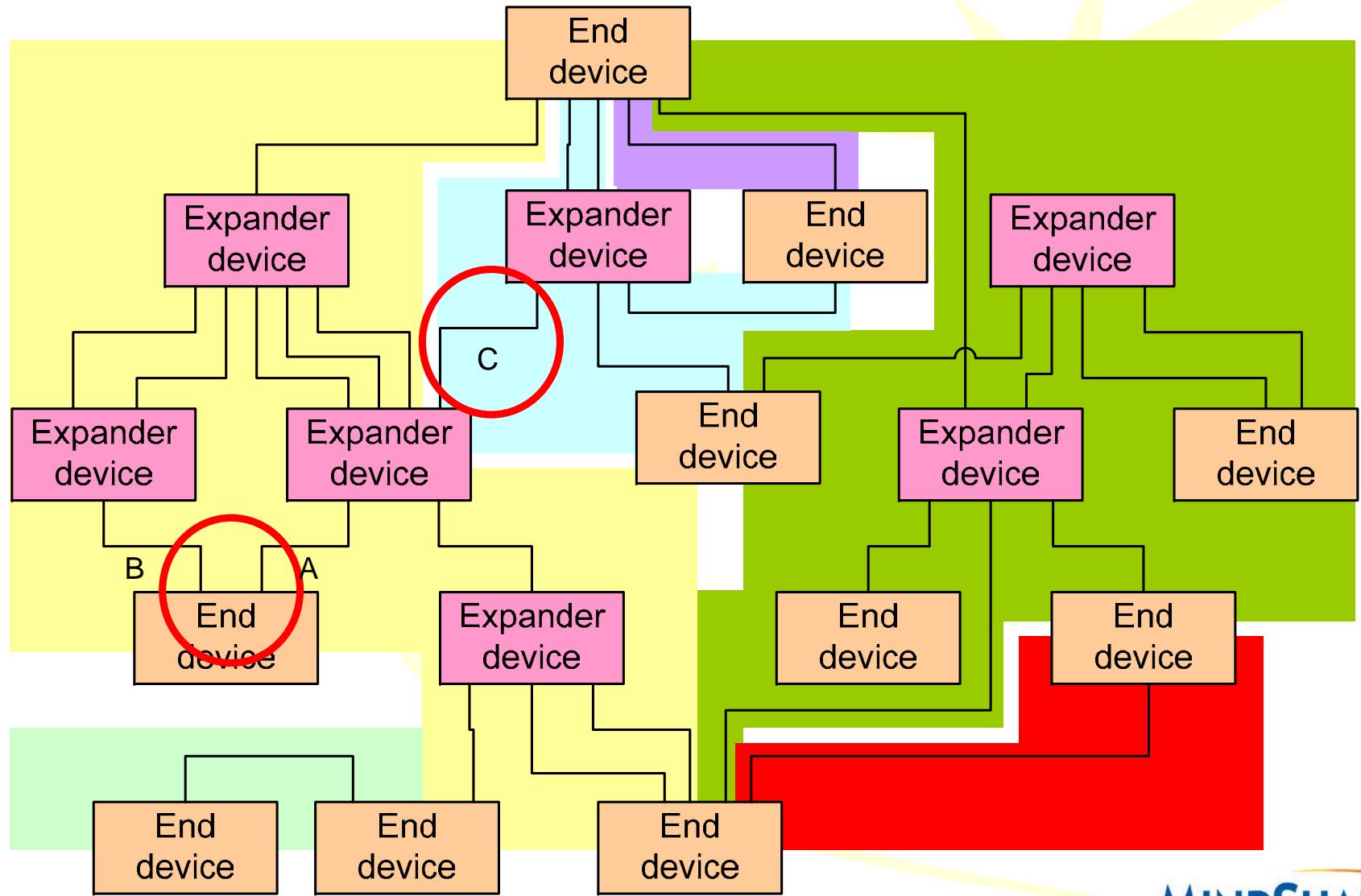
Assume each device uses only one address on all phys

How many domains and how many errors?



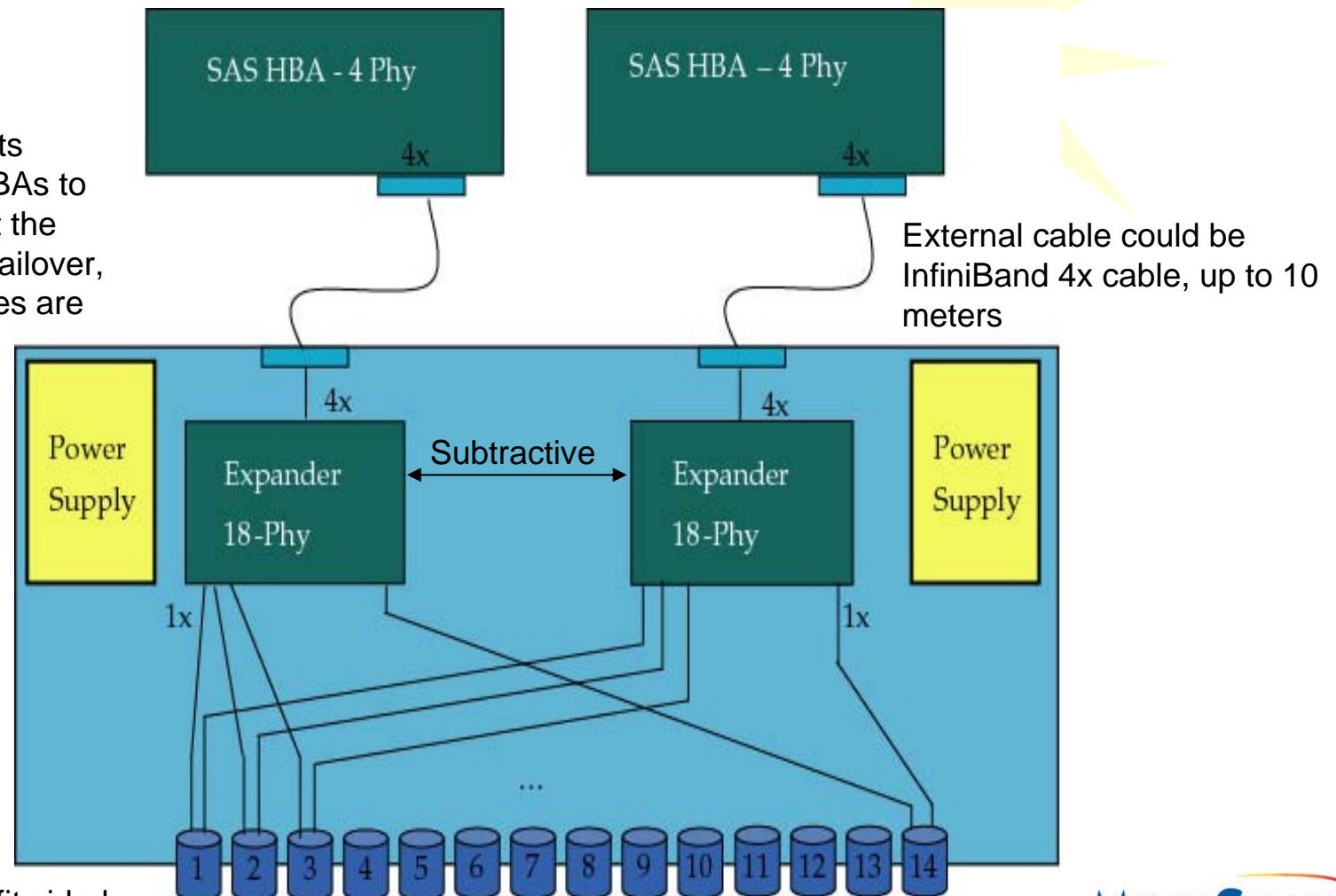
# Quiz Answers (95)

6 domains  
Multi-path error



# Realistic Domain Example (96)

Tying subtractive ports together allows HBAs to keep track of what the other is doing for failover, but means all drives are in one domain.

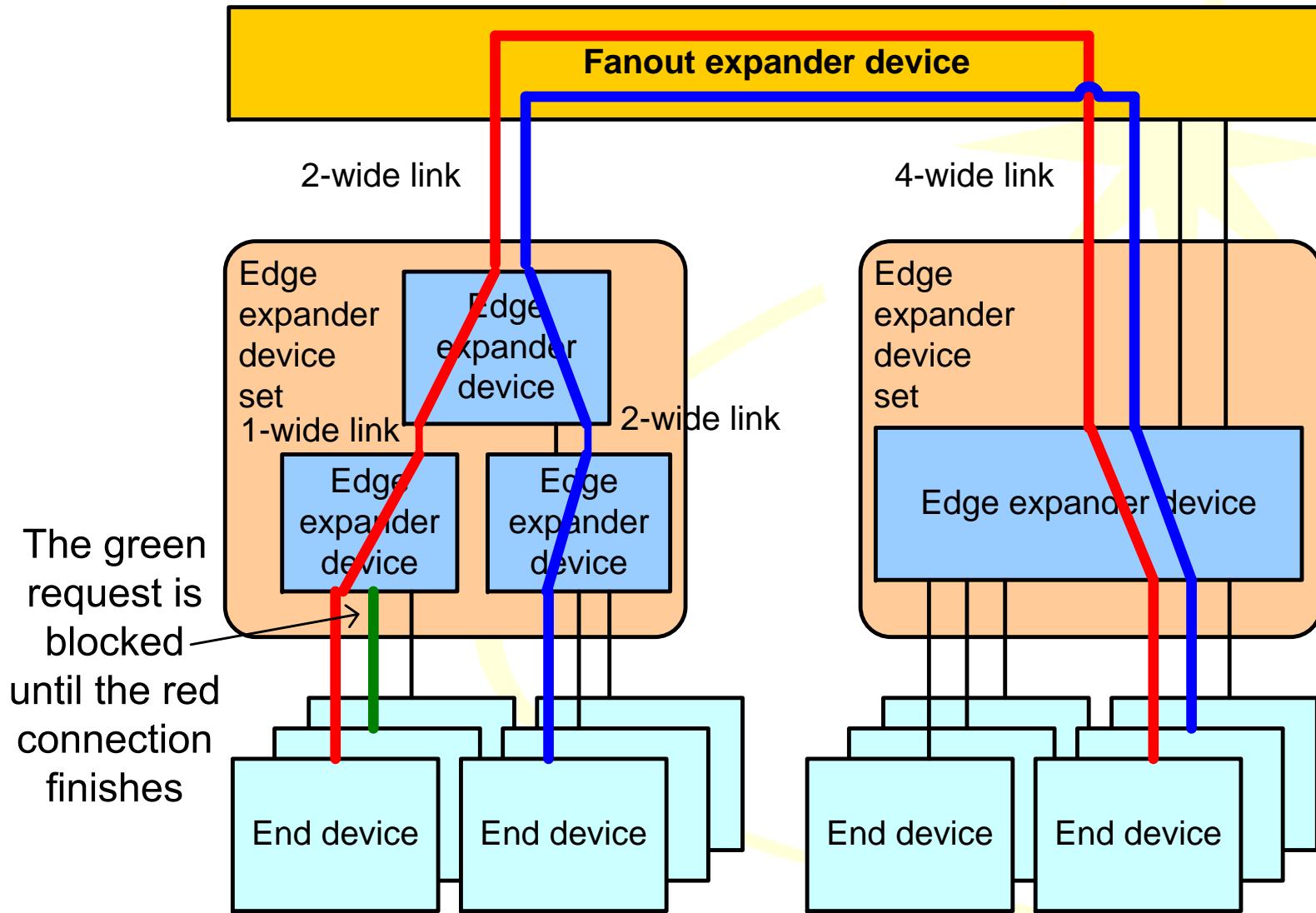


Fourteen 3.5" drives fit side-by-side in a cabinet drawer.

# Connections

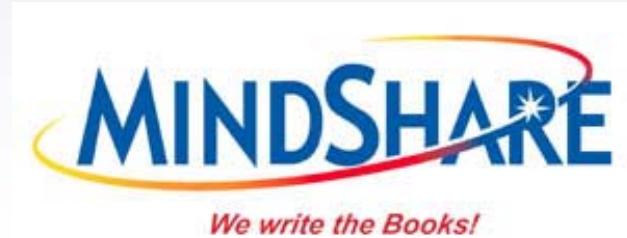
- To establish a connection:
  - Source phy transmits an OPEN address frame containing a destination address
  - Expanders connect an internal path to route the OPEN to the matching destination port
  - Destination phy replies with an OPEN\_ACCEPT primitive
  - Expanders connect an internal return path as they route the ACCEPT back to the initiator, and the Connection is established
  - To close the connection, both sides exchange DONE and CLOSE primitives

# Connection example (97)

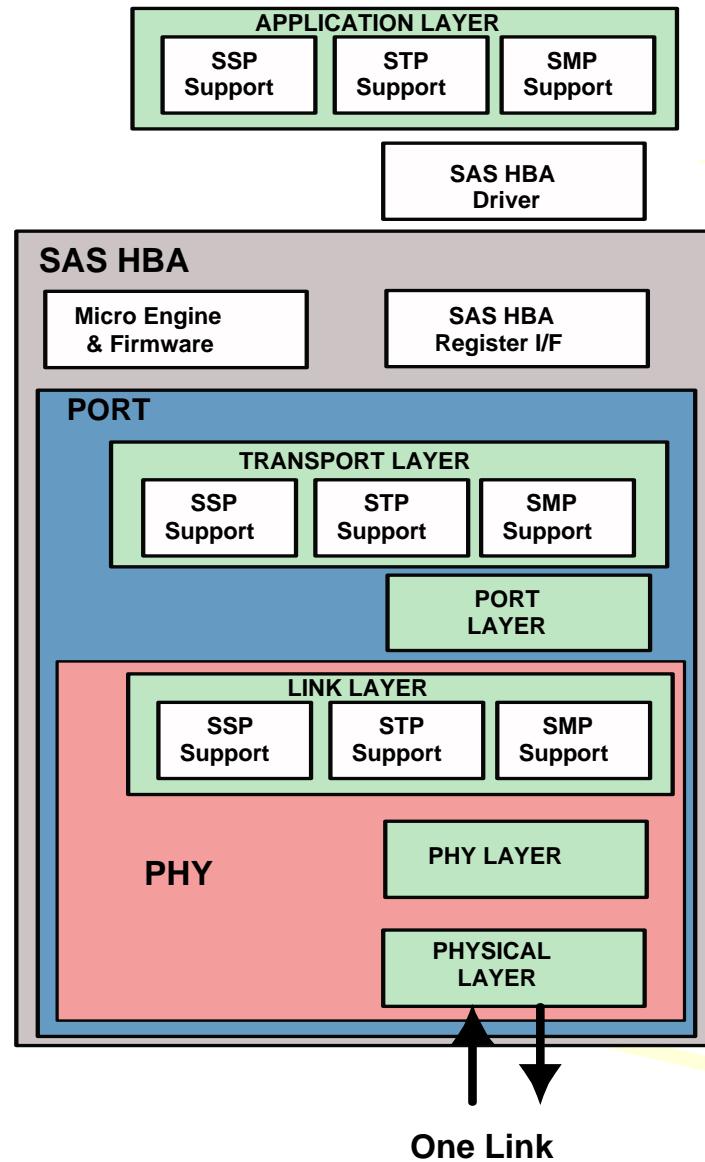


# *Chapter 5*

## *The Layered Device Architecture*



# Layers in Narrow Port Device (101)



# Layer Design

- Layers send messages and requests, receive confirmations and status
- Permits modular design, re-use of design code
- Note: Standard does not define layer responsibilities tightly, giving designers freedom to add value with unique implementations

# Application Layer

- Responsibilities:
  - Send Procedure Calls to Transport Layer based on software instructions. Example: **Send SCSI Command** (including arguments for CDB, address, tag, etc.)
  - Handle Errors - choosing whether to reissue a command, for example

# Application Layer

- Several SCSI mode, log, and diagnostic pages are customized for SAS, giving the application layer the ability to tune performance, report parameters, or log errors
- Use of the NOTIFY (ENABLE SPINUP) primitive allows application layer to control timing of drive spin-ups

# Transport Layer

- Frame Construction: Transport layer builds frames according to command and protocol being used

Command	Information Unit field size	Direction	Description
<b>COMMAND</b>	28 to 284	I to T	Send a command
<b>TASK</b>	28	I to T	Send a task management function
<b>XFER_RDY</b>	12	T to I	Request write data
<b>DATA</b>	1 to 1024	Both	Write data (I to T) or read data (T to I)
<b>RESPONSE</b>	24 to 1024	T to I	Send SCSI status (for commands) or task management response (for task management functions)

# SSP Frame



- Common SSP frame format
  - Frame header: first 24 bytes
  - Information Unit: 0 to 1024 bytes
  - Fill bytes: up to 3 bytes
  - CRC: 4 bytes

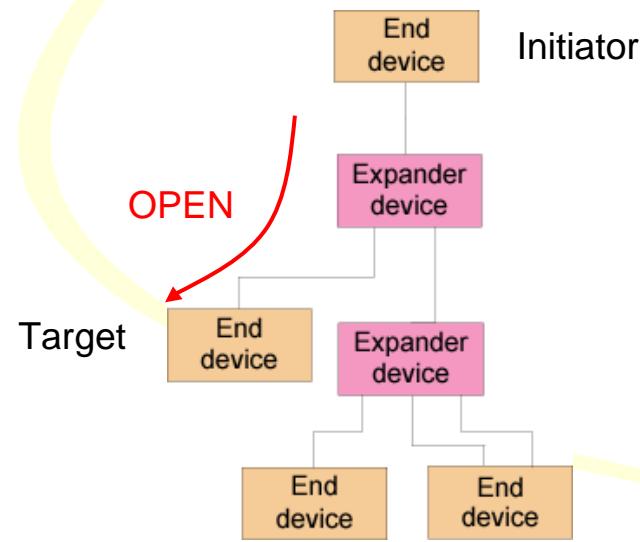
Byte	Field(s)				
0	Frame Type				
1 to 3	Hashed Destination SAS address				
4	Reserved				
5 to 7	Hashed Source SAS address				
8 to 9	Reserved				
10	Reserved	Retransmit	Rsvd		
11	Reserved	Number of Fill Bytes			
12 to 15	Reserved				
16 to 17	Tag				
18 to 19	Target Port Transfer Tag				
20 to 23	Data Offset				
24 to m	<b>Information Unit</b>				
m to (n-3)	Fill bytes, if needed				
(n-3) to n	CRC				

# Port Layer

- Creates queue of frames sent from Transport Layer
- Checks available phys to use for sending frames
  - If a connection to that address is already open, use it, otherwise, create a pending Open request to begin the process
  - Interact with Link Layer to open connection, forward the frames, and close the connection

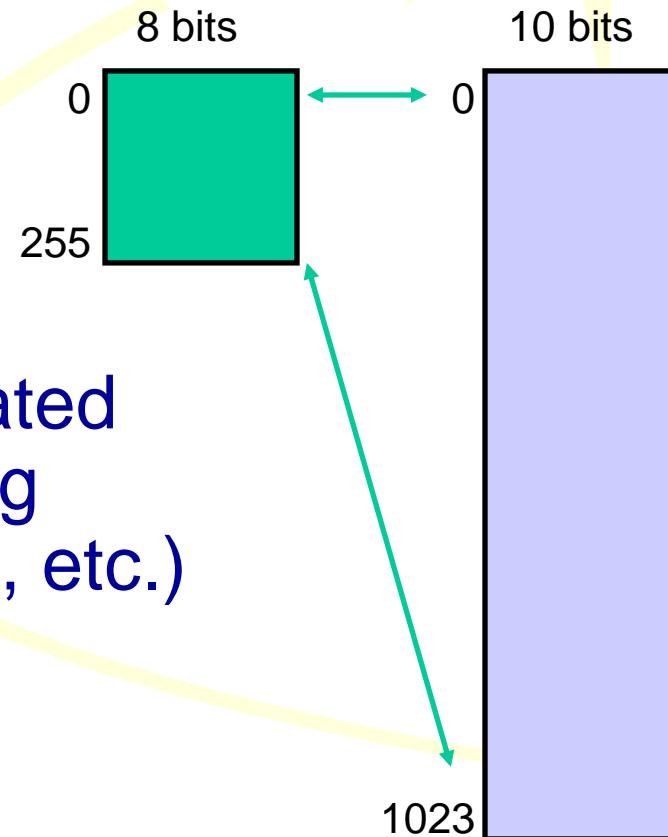
# Link Layer

- Opens connections based on Port Layer requests
- Review of connection process:
  - Initiator sends OPEN request
  - Expanders pass it to proper phy based on target address, return AIP to initiator until target responds
  - Target responds by accepting or rejecting request



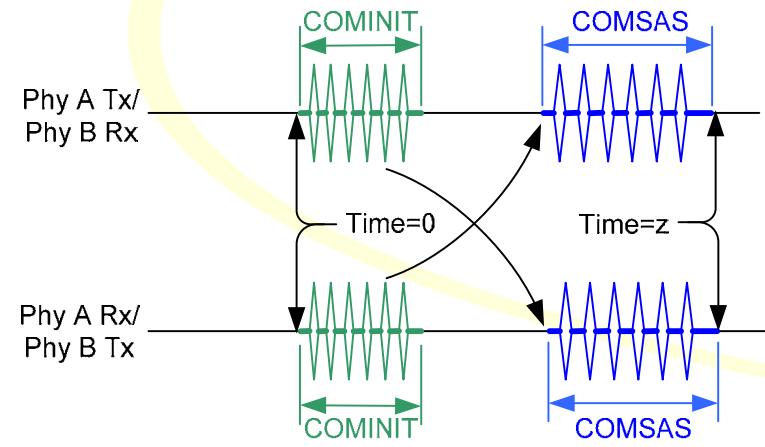
# Phy Layer: 8b/10b Encoding

- Encoding process invented by IBM in 1983 and used by many serial transports
- Converts 8-bit bytes into 10-bit data characters for transmission
- Goals of 8b/10b:
  - Clock recovery
  - DC balance
  - Error detection
- Note - this is not related to any drive encoding (MFM, RLL, EPRML, etc.)



# Phy Layer: OOB signals

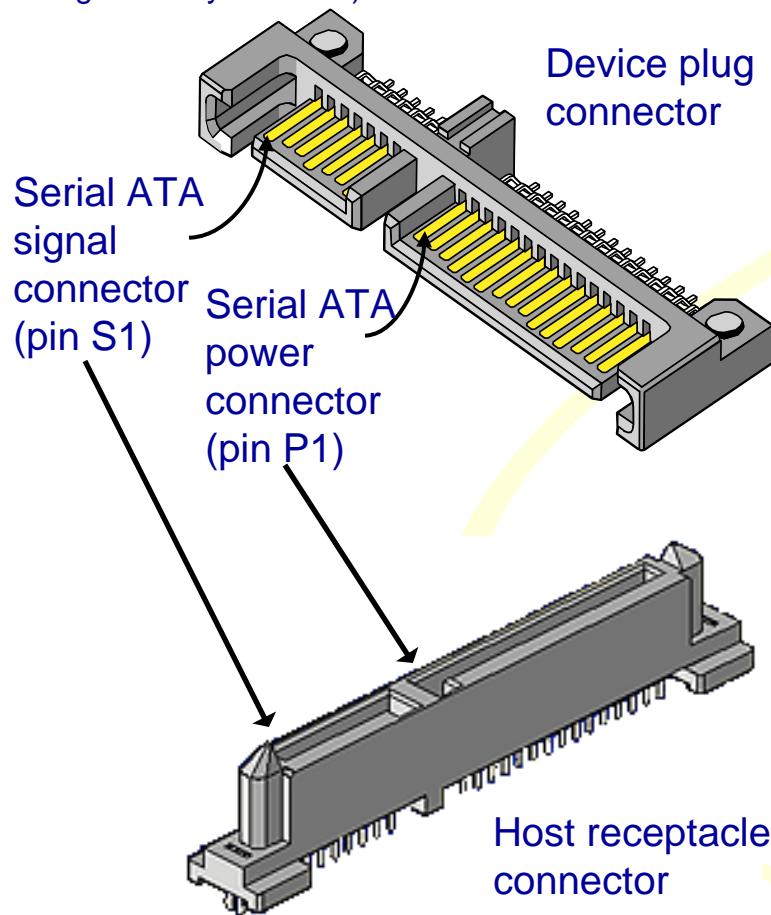
- OOB (Out Of Band) signaling is a pattern of idles and bursts used to initialize a link
  - Idle time (and negation time) - differential 0v  
(Positive signal = negative signal)
  - Burst time - burst of ALIGN(0) primitives
- Length of idle time distinguishes OOB signals
  - COMINIT, COMWAKE, and COMSAS



# Physical Layer: connectors

## Appearance of Serial ATA Connectors

(Drawing courtesy of Molex)

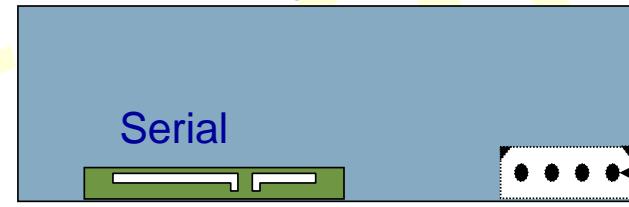


## Device connector sizes and locations

### Serial



power signal



Serial

power signal

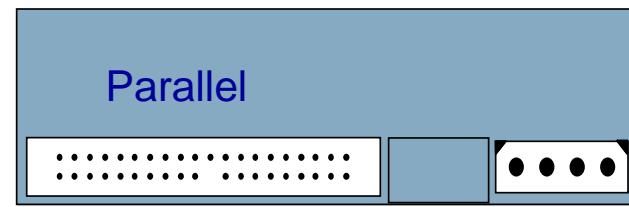
2.5"

3.5"

Legacy Power if desired  
(vendor specific)

(5.25" form factor also defined for  
devices like tape drives and DVDs)  
in comparison...

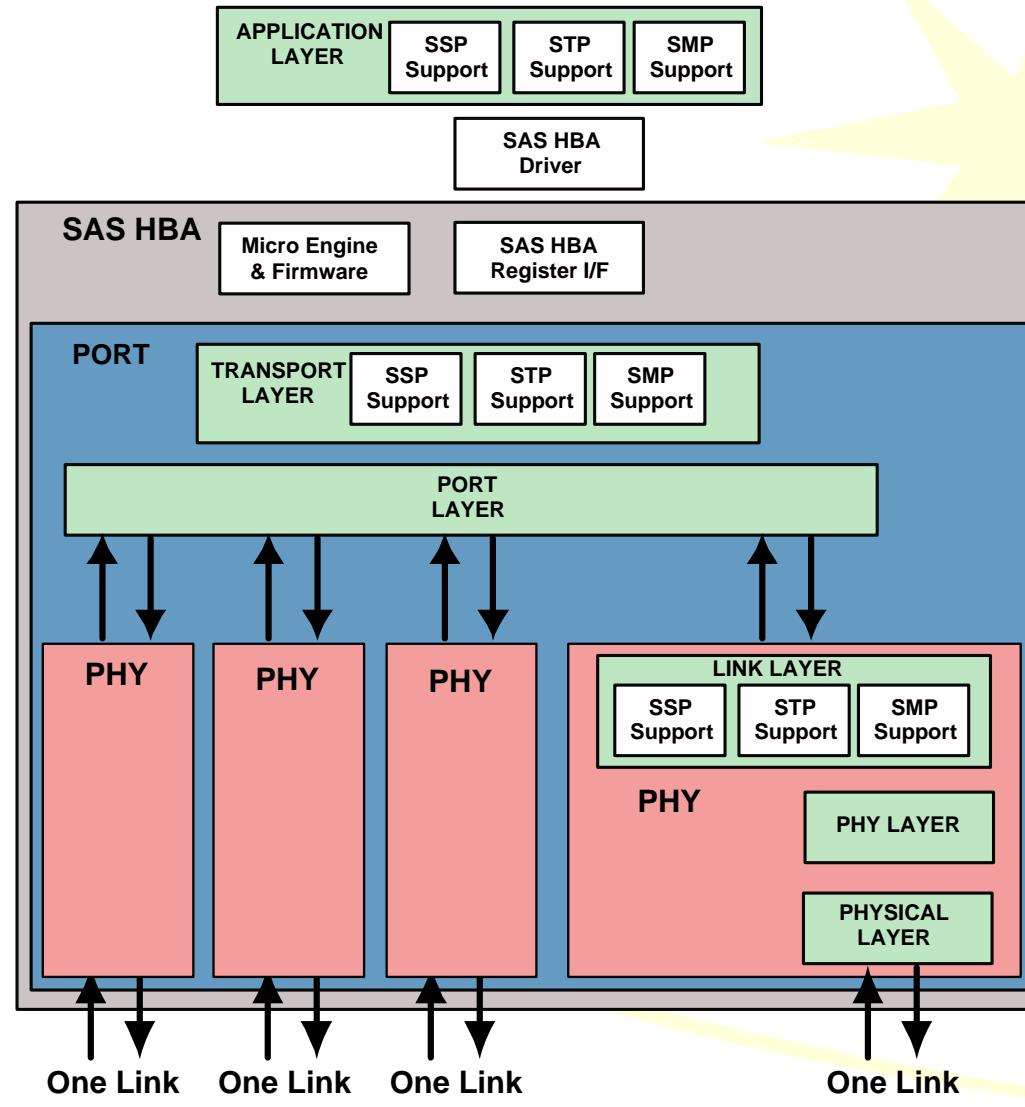
### Parallel



parallel ATA signals

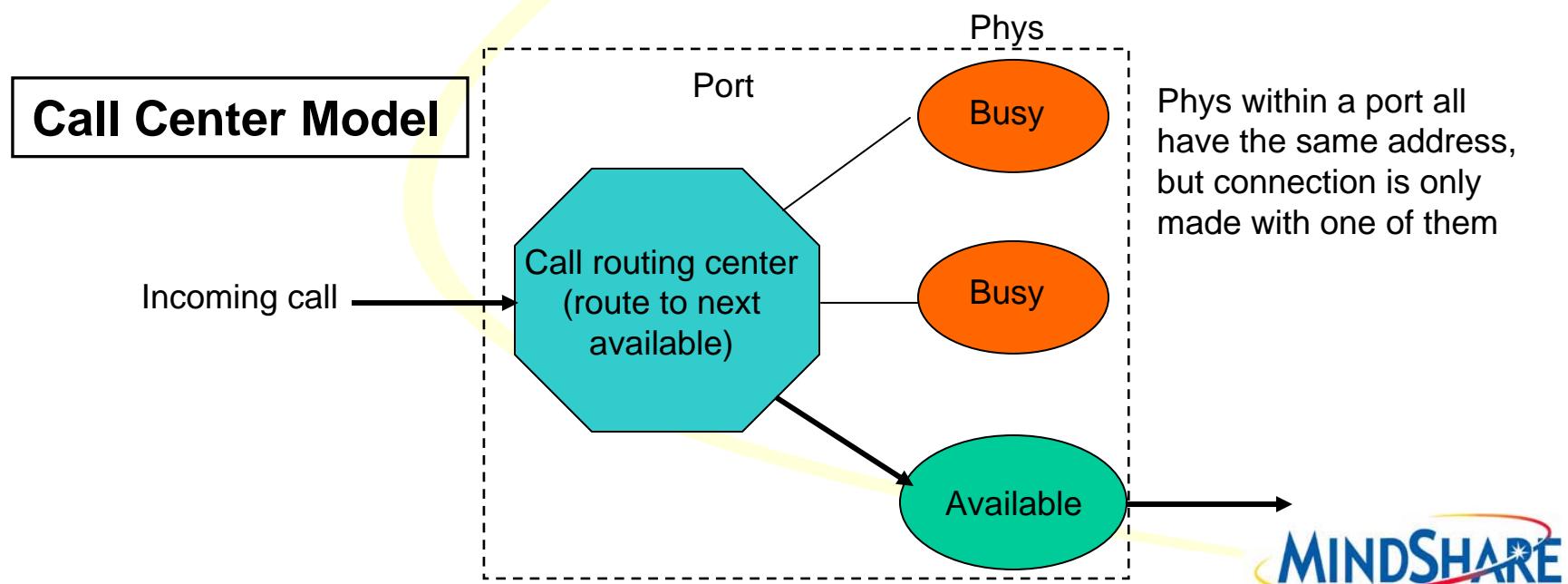
4-pin power

# Device with One Wide Port (112)



# Ports – Call Center Model (113)

- Commands are addressed to a SAS port, which is assigned the SAS address
- A port may have more than one phy to use (wide link), and chooses one that is available much like a telephone call center

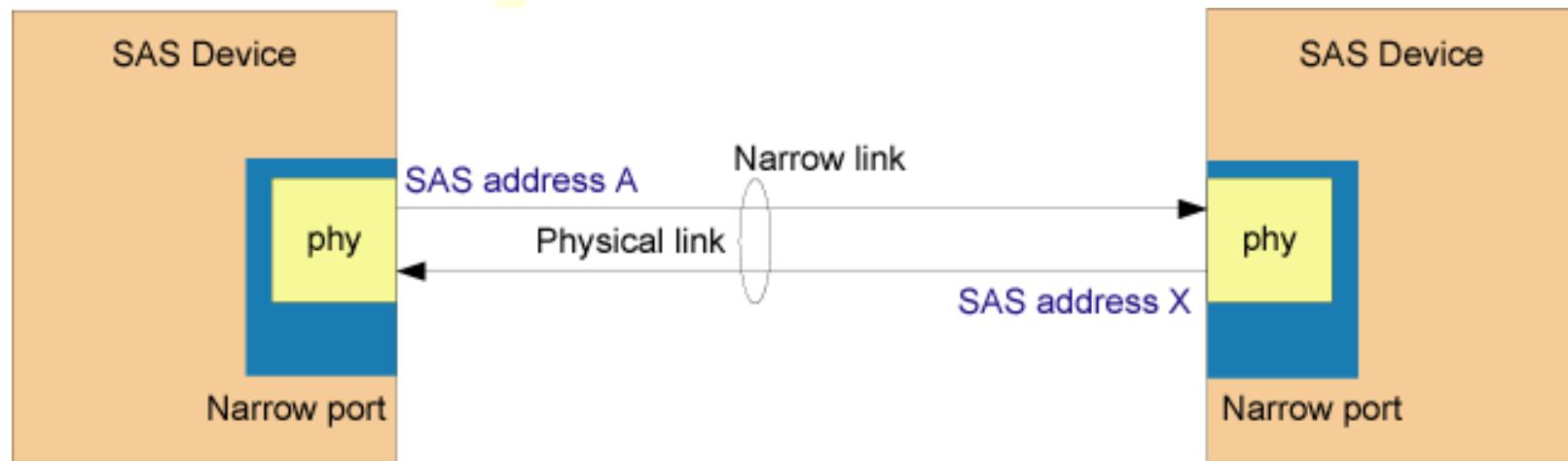


# Connection rules

- Connections are addressed to SAS ports and are established between phys
- Wide ports may establish multiple connections at a time (one per phy) to different destinations or the same destination if target has wide port
  - SAS disk drives only offer two narrow ports
  - HBAs and RAID controllers may offer wide ports

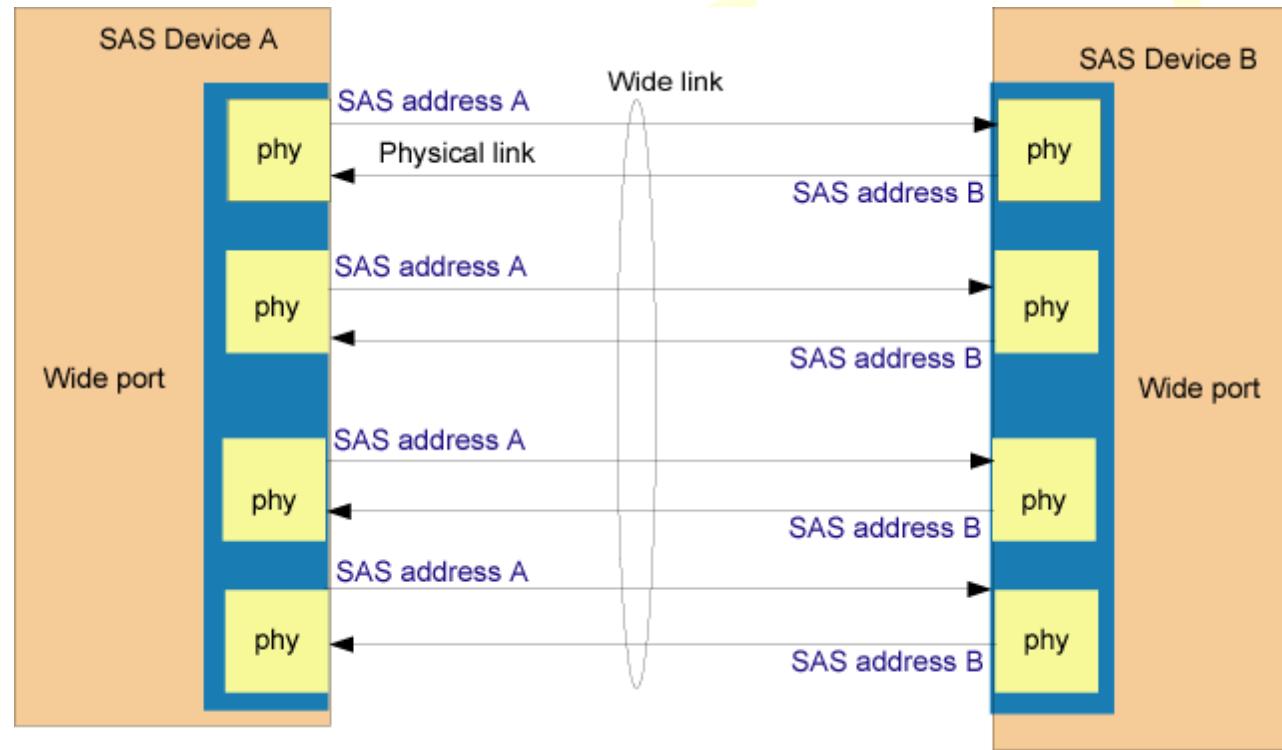
# Single-Phy Port Example

- Single-phy port (narrow port)
  - One transmit differential pair and one receive differential pair
    - Both directions use the same link rate
    - SAS links always symmetrical



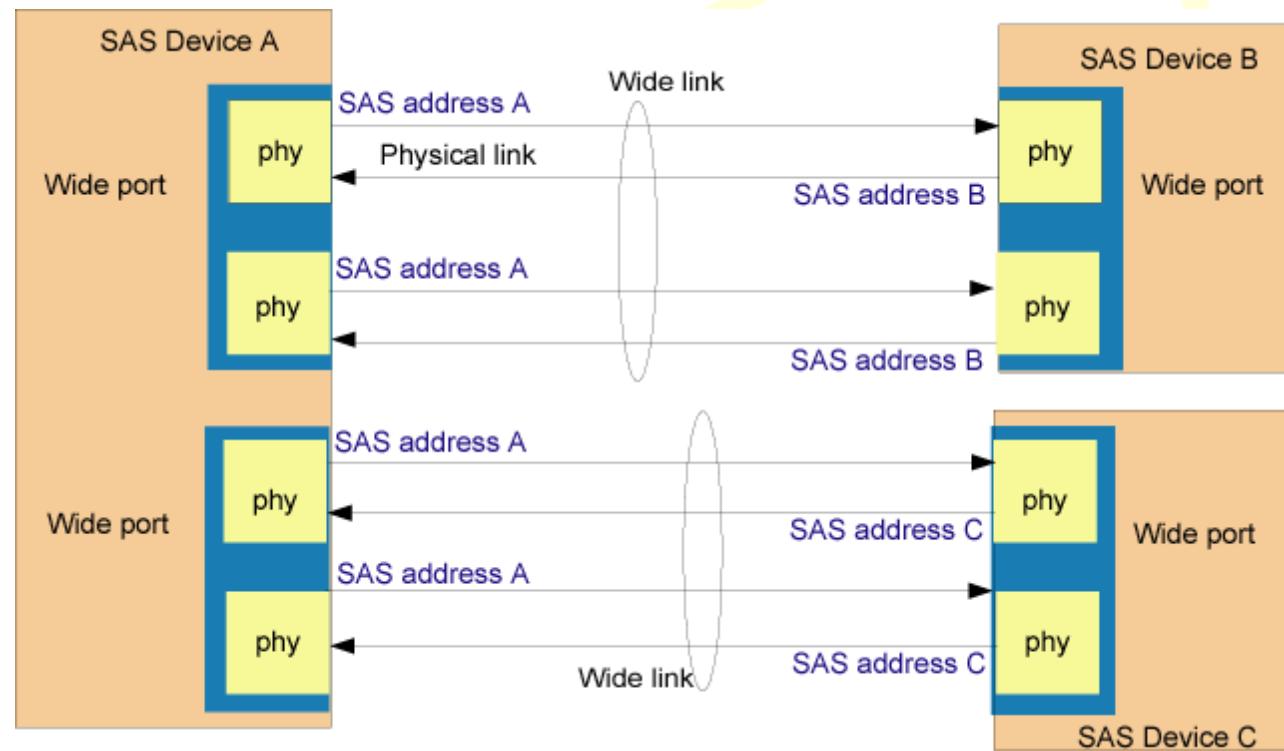
# Wide Port Example

- All attached SAS addresses visible to Device A are the same, so these 4 phys are grouped into a single, wide port



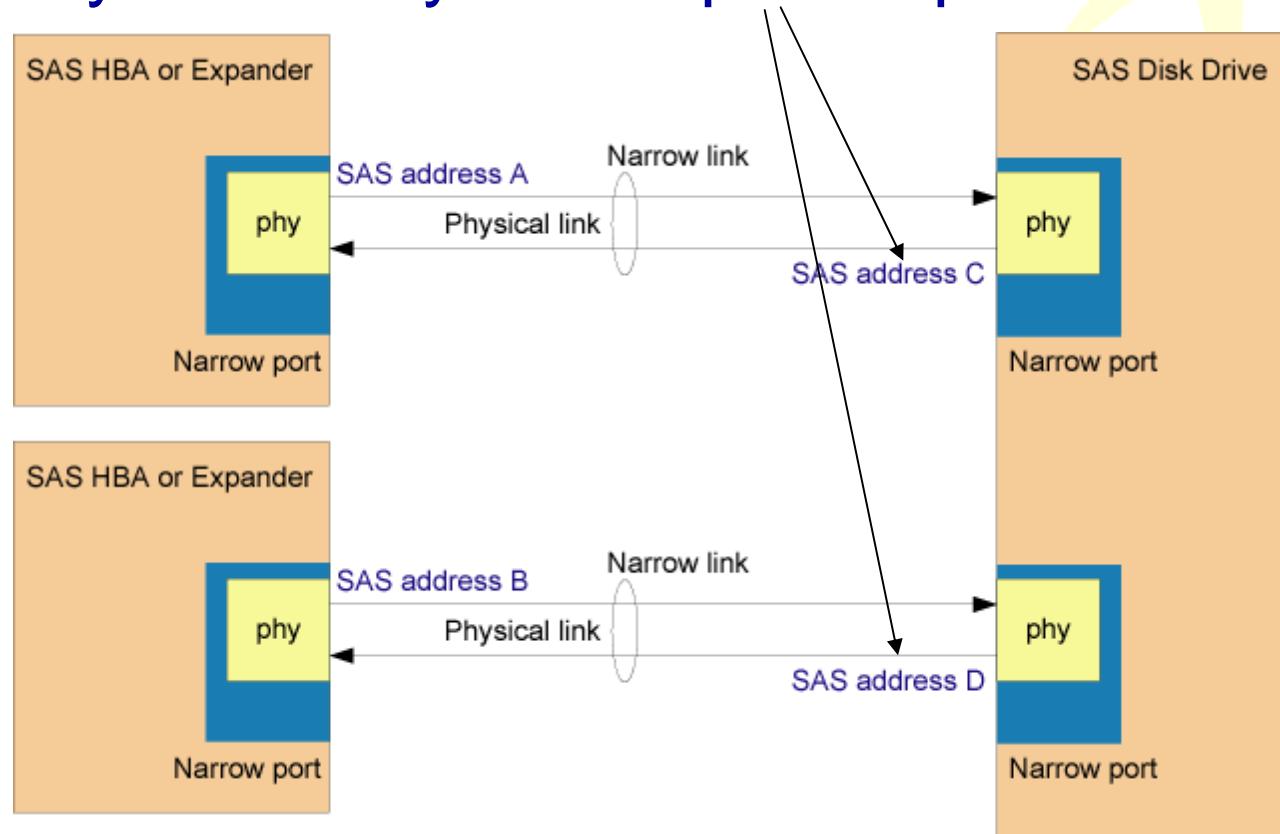
# Multi-phy Port Example

- Device A phys detect two different SAS addresses, so after identification sequence they are grouped into 2 Ports.



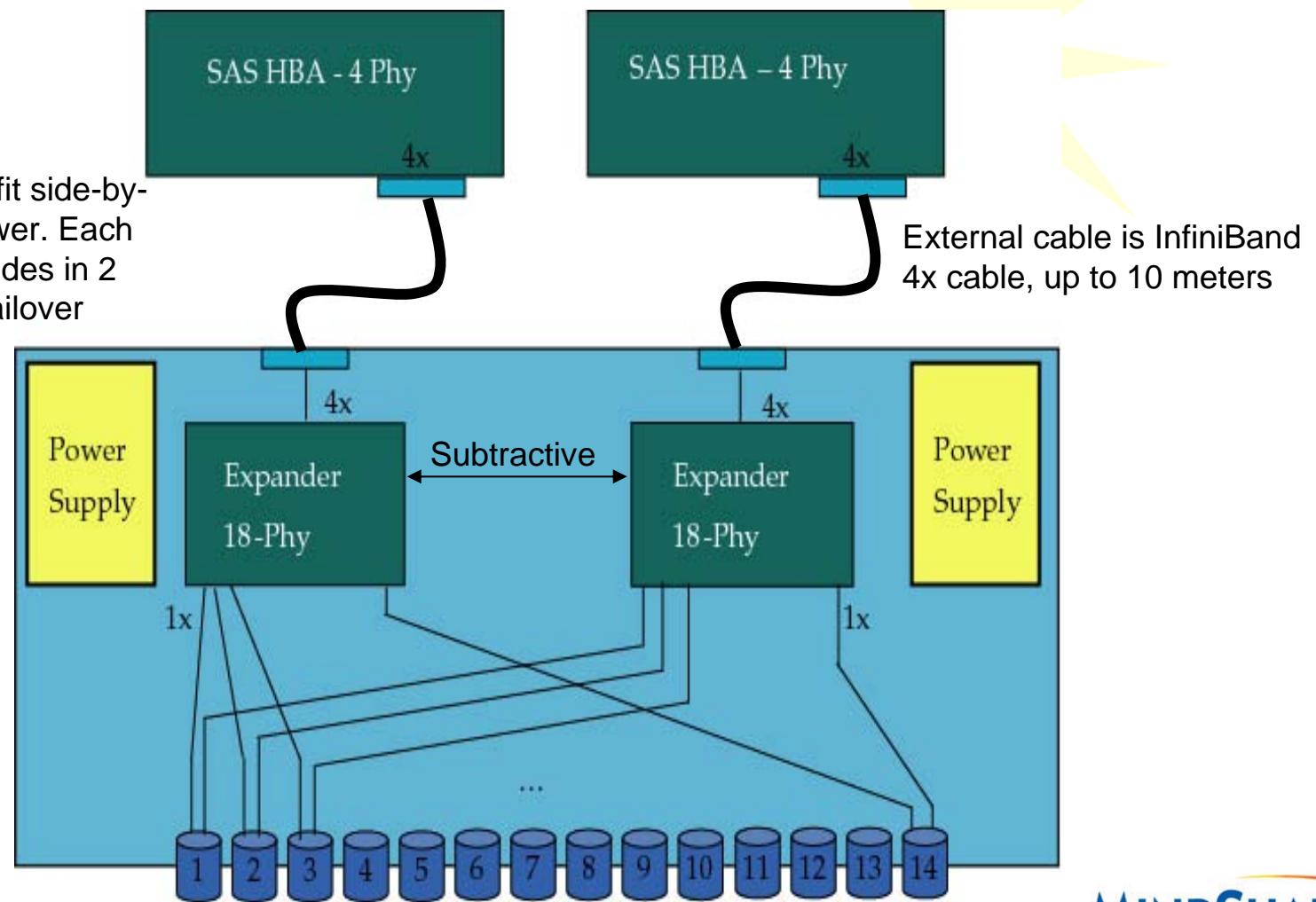
# Dual-Port Example

- Typical dual-ported disk drive implementation; different addresses means they will always be separate ports

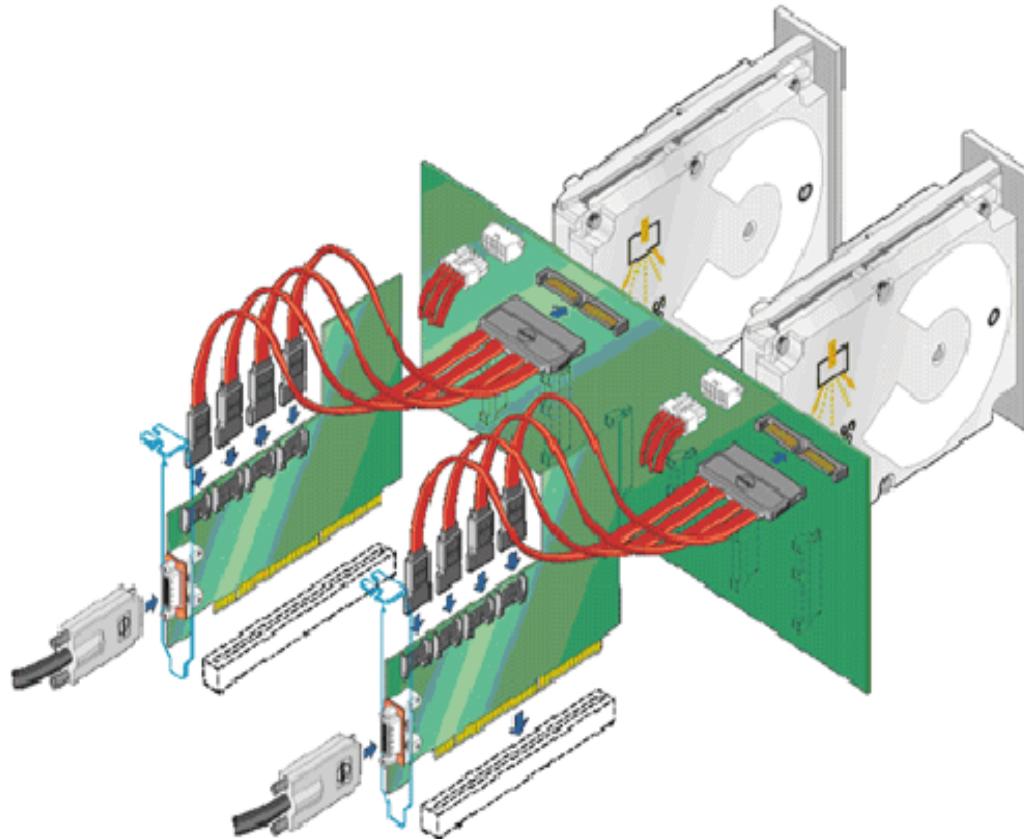


# Realistic Domain Example (96)

Fourteen 3.5" drives fit side-by-side in a cabinet drawer. Each dual-ported drive resides in 2 domains, providing failover capability.



# Realistic Domain Example

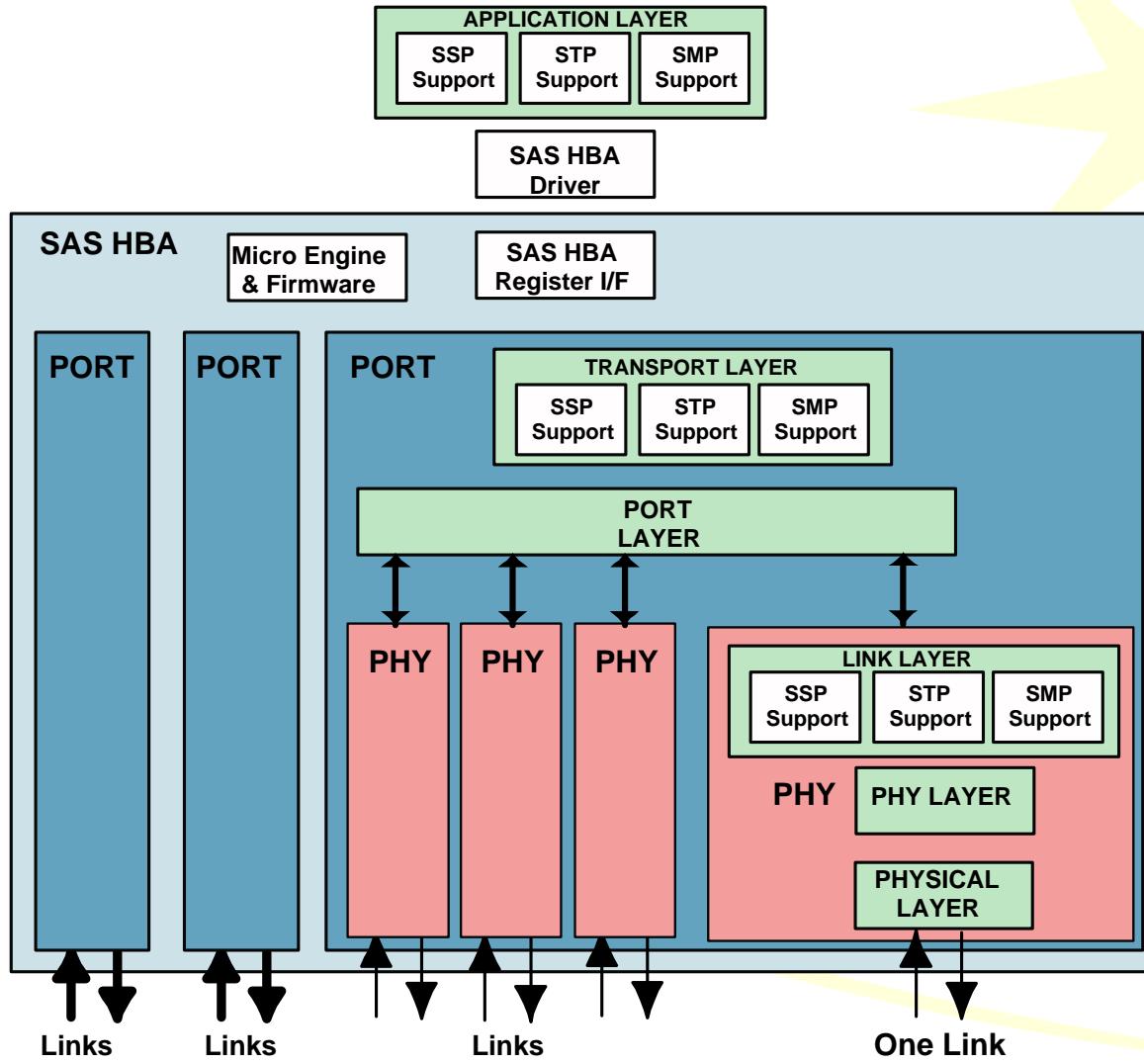


Drawing courtesy of Molex

# State Machines

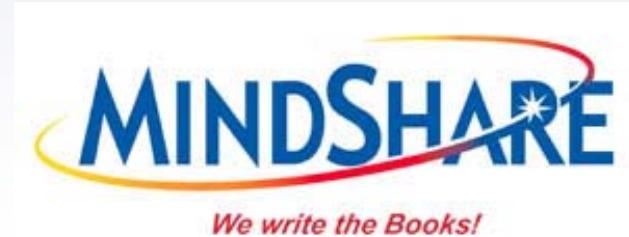
- Most layer behavior described with state machine definitions
  - Not state machines in the usual sense (some state machines have only one state, for example), but models intended to give more precise behavioral description than text alone
  - They generally don't provide hardware definition but may facilitate HDL coding for hardware design

# Device Containing Several Ports (120)



# *Chapter 6*

## *SAS Initialization*



# OOB Used for Initializing

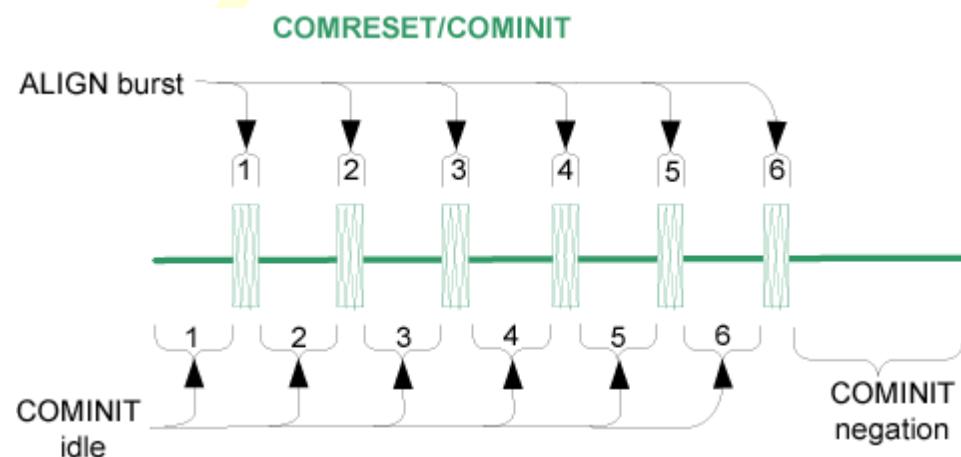
- OOB: Low-speed signal patterns detected by the phy that do not appear in normal data streams
- Pattern distinguished by length of time between idles
  - Idle time (and negation time)
    - Differential voltage = 0 V
    - No transitions (DC idle)
  - Burst time
    - Transmitted as a burst of ALIGN(0) primitives
    - Received as activity (characters are irrelevant)
- Designed to be detectable by analog squelch detection logic

# OOB Voltage

- Because the signaling levels are different for SAS and SATA, it's considered best to start with the lower SATA voltage (700mv) and look for a response first, before trying the higher voltage SAS (up to 1600mv).
- This will protect against possible damage if a lower-voltage SATA device is attached

# OOB – Transmitter (129)

- Transmit 6 idle/burst pairs, then a negation time
- Idle times define the signals
- Burst time is defined as 160 OOB1 (see next slide)



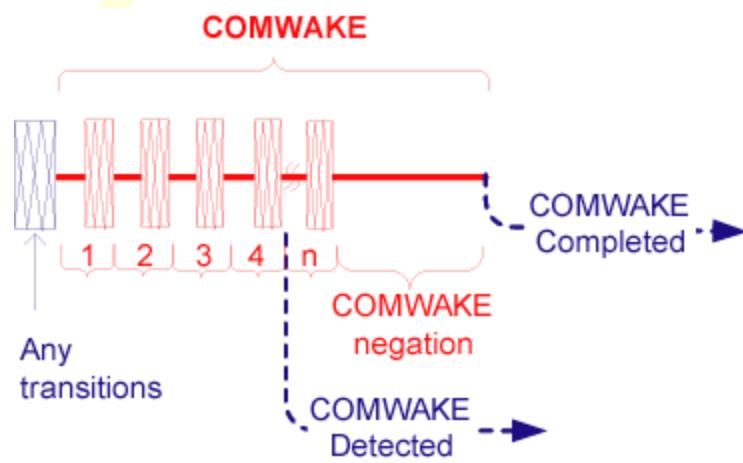
# OOB Transmit Timing (130)

- COMINIT, COMWAKE and COMSAS burst times all look the same; only idle times are different
- Idle times are defined in units called OOBI (Out-Of-Band Interval, 666.6ps to 666.734ps)

OOB Signal	Burst time	Idle time	Negation time
COMWAKE	160 OOBI (106.7 ns)	160 OOBI (106.7 ns)	280 OOBI (186.7 ns)
COMINIT/ COMRESET	160 OOBI (106.7 ns)	480 OOBI (320.0 ns)	800 OOBI (533.3 ns)
COMSAS	160 OOBI (106.7 ns)	1440 OOBI (960.0 ns)	2400 OOBI (1.6 µs)

# OOB – Receiver (130)

- Looking for 4 idle/burst pairs
- Won't detect the same signal again until negation time is recognized
  - Example: Receive COMWAKE



# OOB Receiver Timing (131)

- Idle Times

OOB Signal	May detect	<u>Shall detect</u>	Shall not detect
COMWAKE	55 to 175 ns	101.3 to 112 ns	< 55 or > 175 ns
COMINIT/ COMRESET	175 to 525 ns	304 to 336 ns	< 175 or > 525 ns
COMSAS	525 to 1575 ns	911.7 to 1008 ns	< 525 or > 1575 ns

- Negation Times

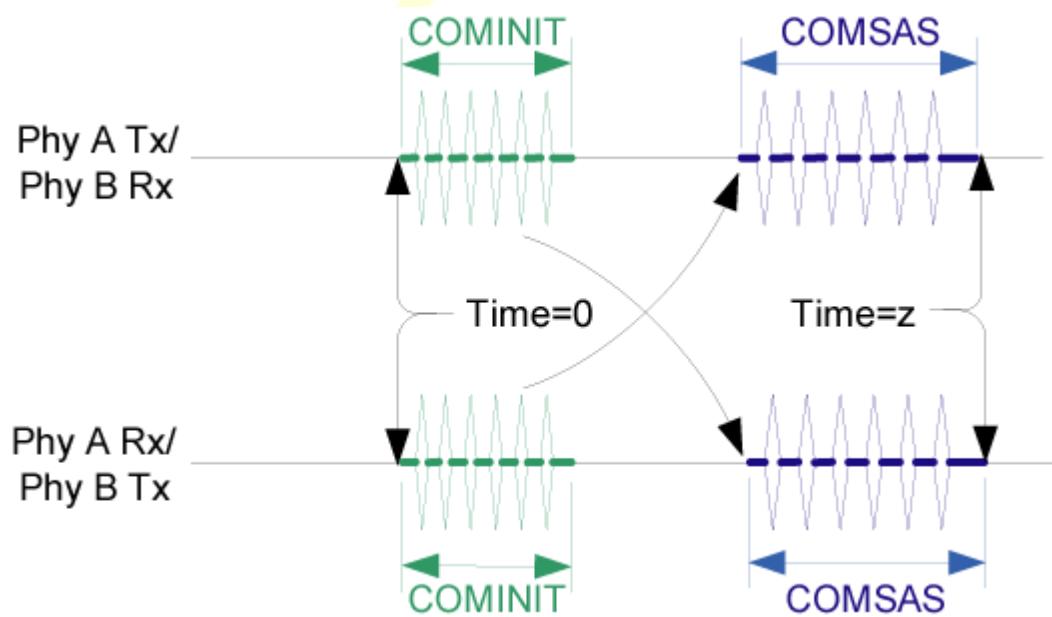
OOB Signal	Shall detect
COMWAKE	> 175 ns
COMINIT/ COMRESET	> 525 ns
COMSAS	> 1575 ns

# PHY Reset Sequences

- For SAS
  - Phy reset sequence includes
    - SAS OOB sequence
    - SAS Speed Negotiation sequence
  - Link reset sequence includes phy reset and the Identification sequence
- For SATA
  - Link and phy reset are the same, as is SATA power-on sequence:
    - SATA OOB
    - SATA Speed Negotiation

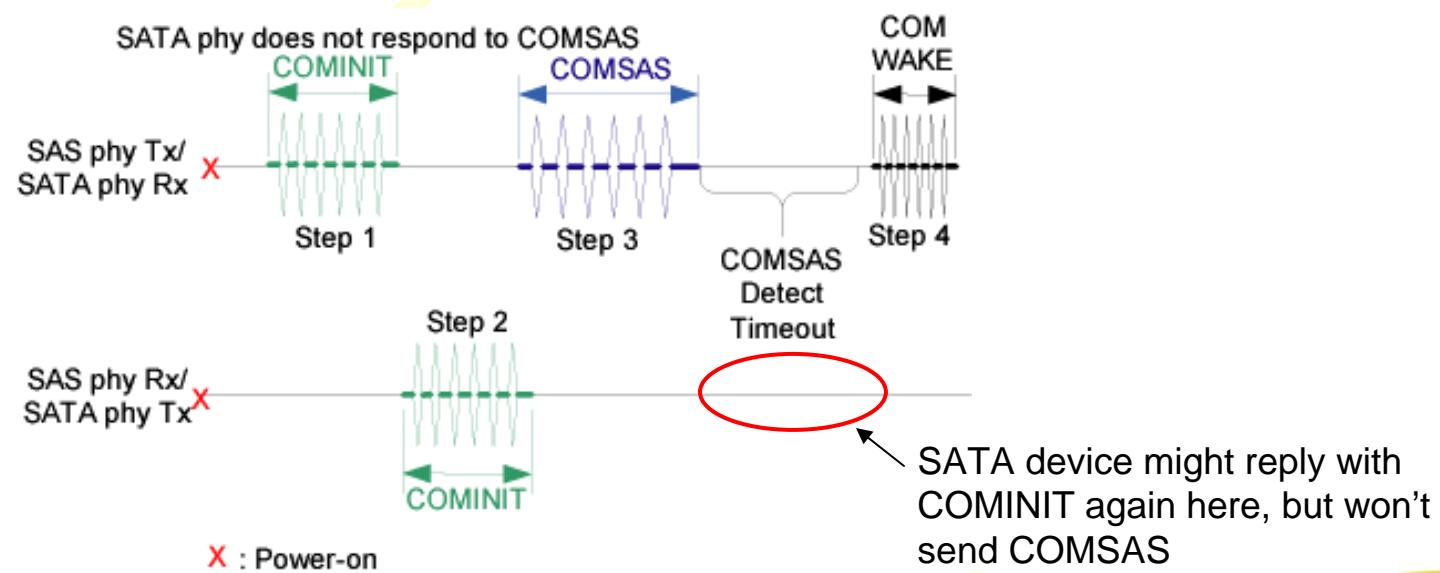
# SAS OOB (132)

- Phy A sends COMINIT and COMSAS, neighbor should do the same
  - If COMSAS is received, link is SAS to SAS
  - Otherwise, link must be SAS to SATA



# SAS to SATA OOB (133)

- If COMSAS not received, link is understood to be SATA and OOB changes to SATA OOB
- SATA device must be a target and not a host, since a SAS device drives COMWAKE and looks like a host

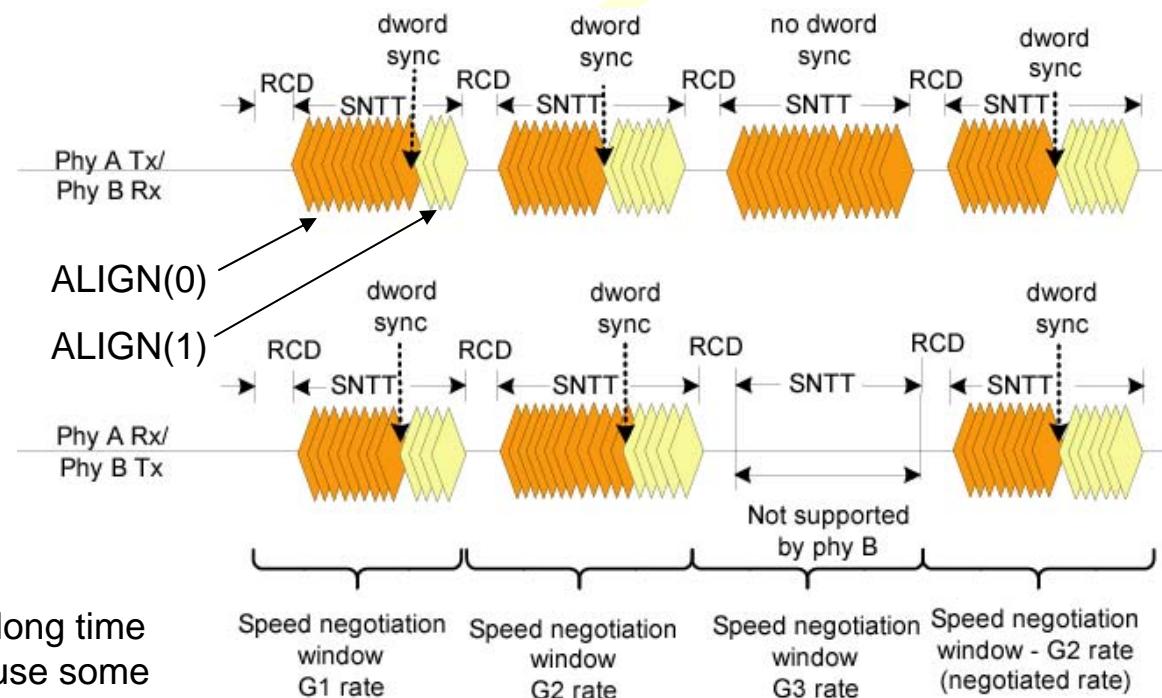


# SAS Speed Negotiation (133)

- Slow to fast progression
  - Both devices start at slowest rate, send ALIGN(0)s
  - If ALIGN(0)s received, send ALIGN(1)s
  - If ALIGN(1)s received, rate window succeeds, otherwise fails

# SAS Speed Negotiation (135)

- Try lowest rate first. If successful, increase to next speed and repeat the process
- Continue until both a supported and unsupported rate are found



RCD is a very long time at 500us because some vendors wanted a long settling time for a PLL.

# SAS Speed Negotiation

- Much slower than SATA
  - SATA starts at higher speed and finishes when lock is achieved
  - SAS must try every rate window. One reason for this is that it informs devices about all the possible rates in case it becomes necessary to change the rate later.
- Negotiation Time: ~2.4ms at 3.0Gbps

# OOB Timeout

- If no reply to COMINIT:
  - SAS targets only try once, to announce their presence, then wait for initiator
  - Expanders try again within 500ms
    - Initiators depend on them to detect new devices
  - Initiators try again, at not less than 10ms between attempts

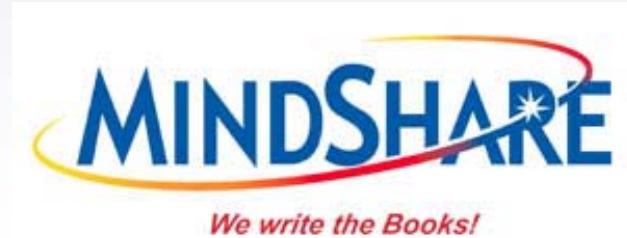
# Phy State Machines

- State Machine (SP) has 3 sets of states; one each for:
  - OOB sequence
  - SAS speed negotiation
  - SATA speed negotiation
- Works with SP\_DWS state machine to achieve dword synchronization

# SAS State Machines - SP\_DWS

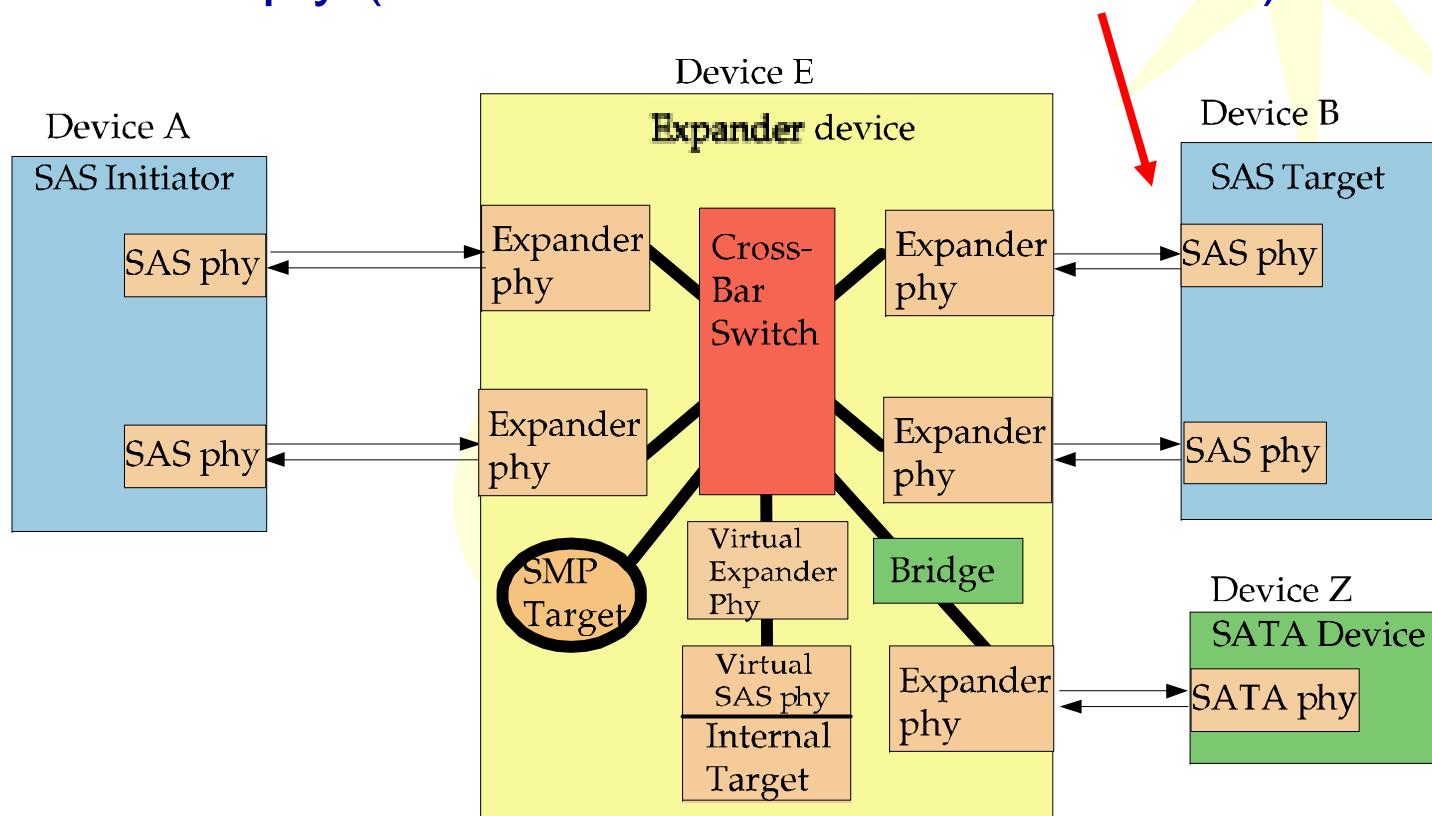
- Searches for character and dword boundaries  
(more on this in Phy Layer, p409 of book)
  - Search for K28.5. When found, this defines the character and dword boundaries
  - If next 3 characters create a valid primitive, dword alignment was successful
- Reports status of the search
  - When 3 good primitives received after the first valid primitive, dword synch has been achieved

# *Link Initialization Example*



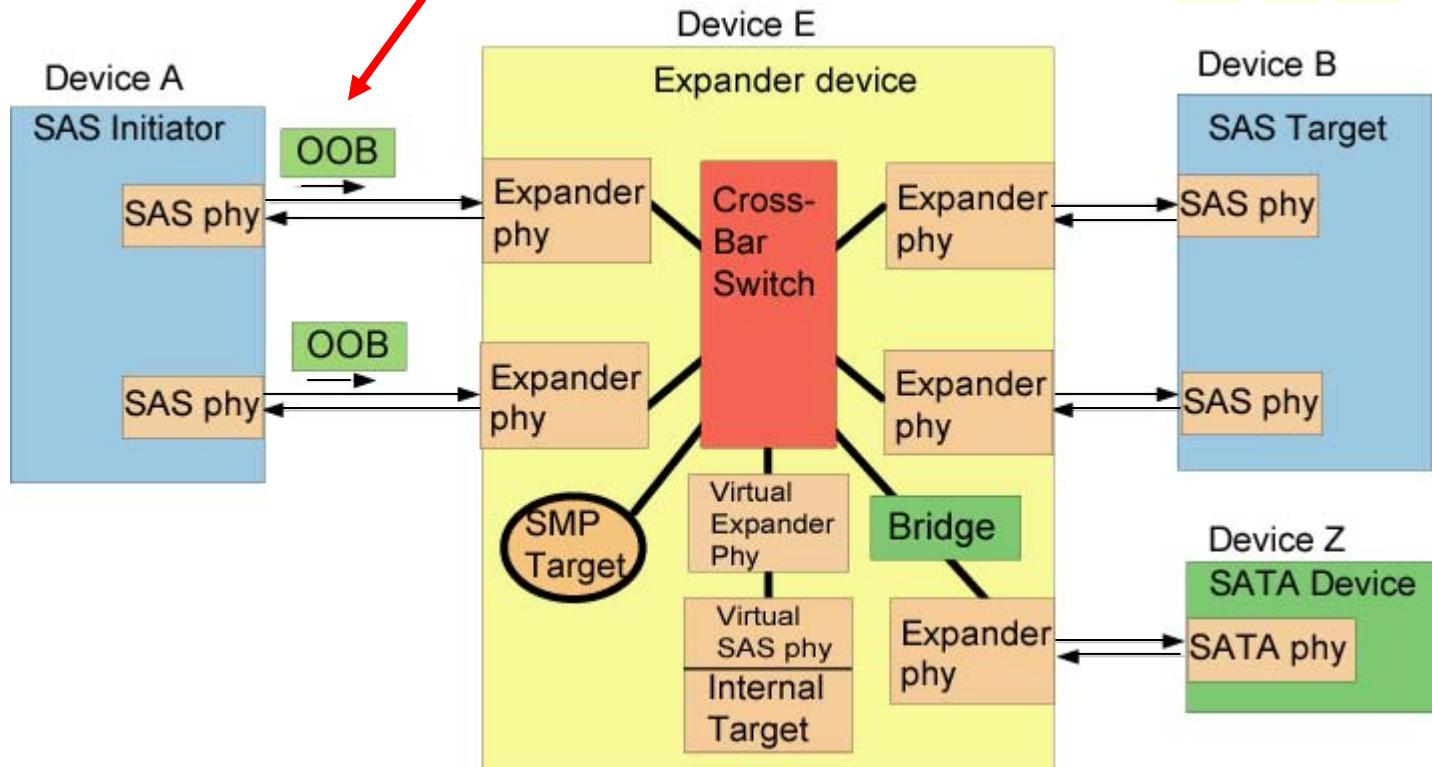
# Initialization Example (137)

- Assume Device B powers up first. It sends OOB signals on all phys and then, as a target, simply waits for a reply (which could be some time later).



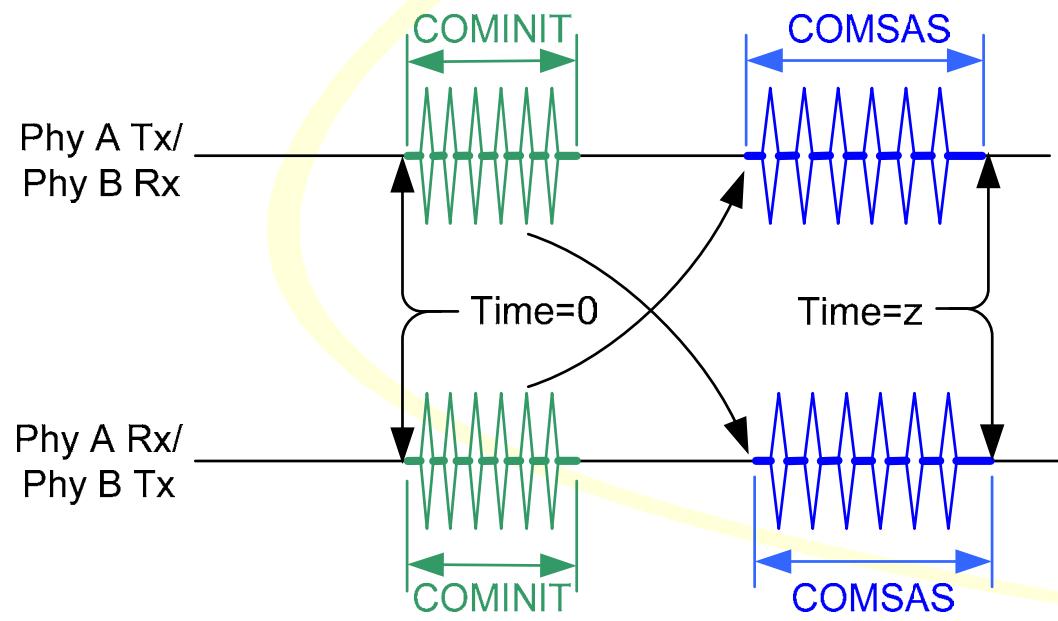
# Initialization Example

- Next, A powers up - it also sends OOB and gets no reply. As an initiator, it keeps retrying at a vendor-specific rate.



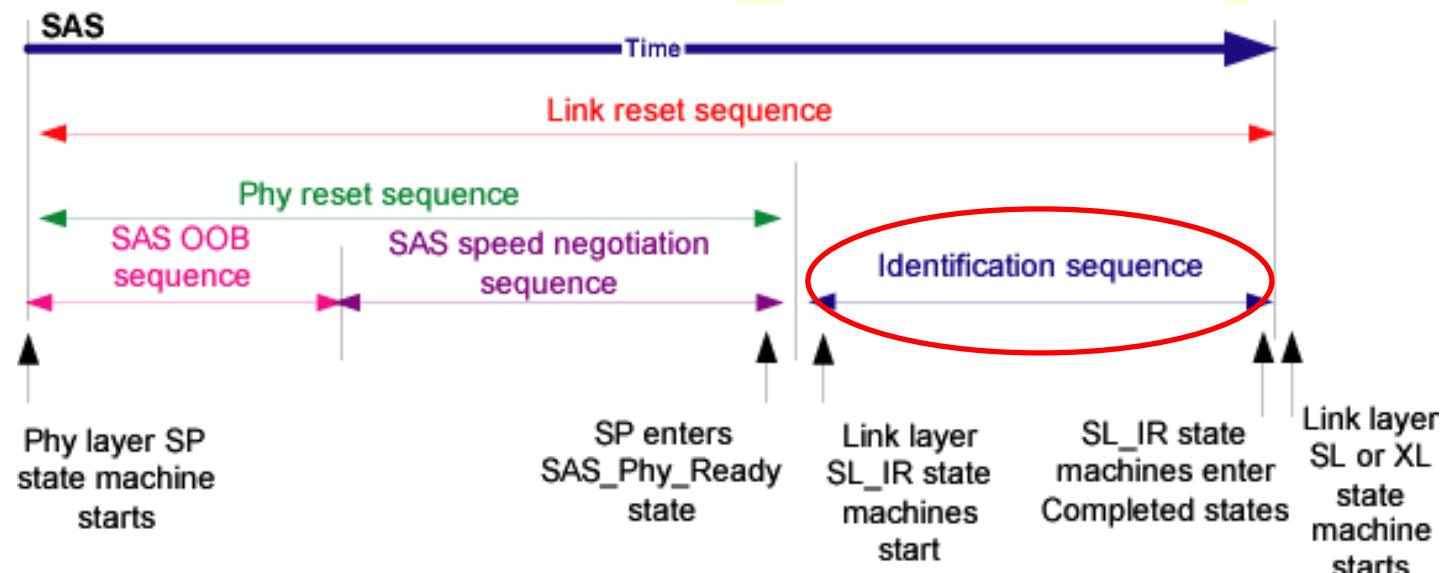
# Initialization Example

- When expander comes up, it sends COMINIT on all of its phys (probably all at once).
- Device A responds with COMINIT, then both exchange COMSAS and start speed negotiation, which resolves to 3.0 Gbps. That completes the phy reset sequence.



# Initialization - Device A

- Next, the Link Layer sends an Identify address frame and expects to receive one to be informed of the attached address.



# Identify Address Frame

	Bit 7	6	5	4	3	2	1	0
Byte 0	Restricted	Device Type						
1								Address Frame Type = 0
2				SSP Init	STP Init	SMP Init	Restricted	
3				SSP Target	STP Target	SMP Target	Restricted	
4 - 11								Restricted
12-19								SAS Address
20								Phy Identifier
21-27								Reserved
28-31								CRC

Annotations:

- Device Type: End device, edge expander, or fanout expander
- Initiator or target and protocol type: SSP, STP, SMP
- Address of the SAS port: SAS Address
- The phy number inside the attached device: Phy Identifier

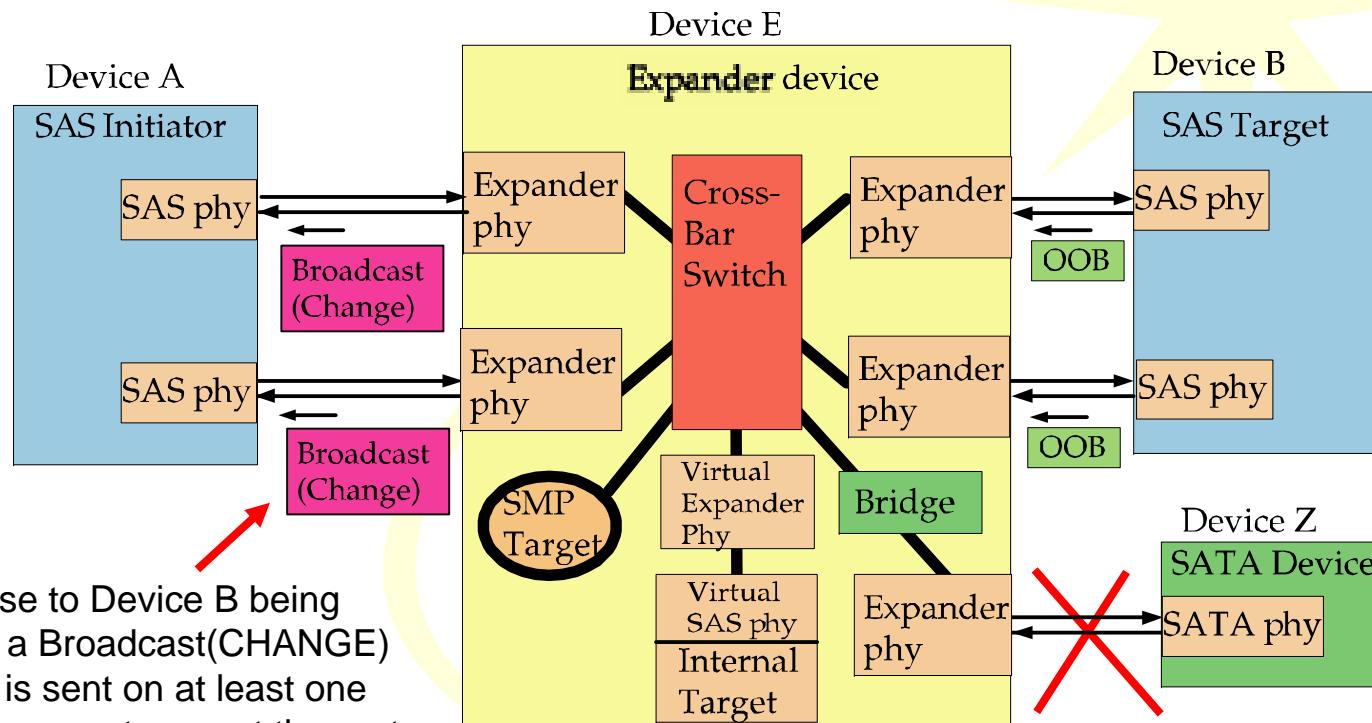
# Initialization - Device A

- At this point, A knows it is attached to a SAS expander device, the negotiated link rate, and the address of the attached device
- Device B goes through the same sequence and learns the same information

# Init Example

- Expander sees the devices become available at different times. When anything changes on its phys, it reports that by sending a Broadcast Change message on every phy
- In our example, Device Z is still not ready, so expander's phy keeps trying to initialize that link every 500ms.

# Broadcast Change Primitives (138)



In response to Device B being detected, a Broadcast(CHANGE) message is sent on at least one phy for every port, except the port that caused it.

If no device or SATA device is detected, can't send broadcast to it.

# Init Example

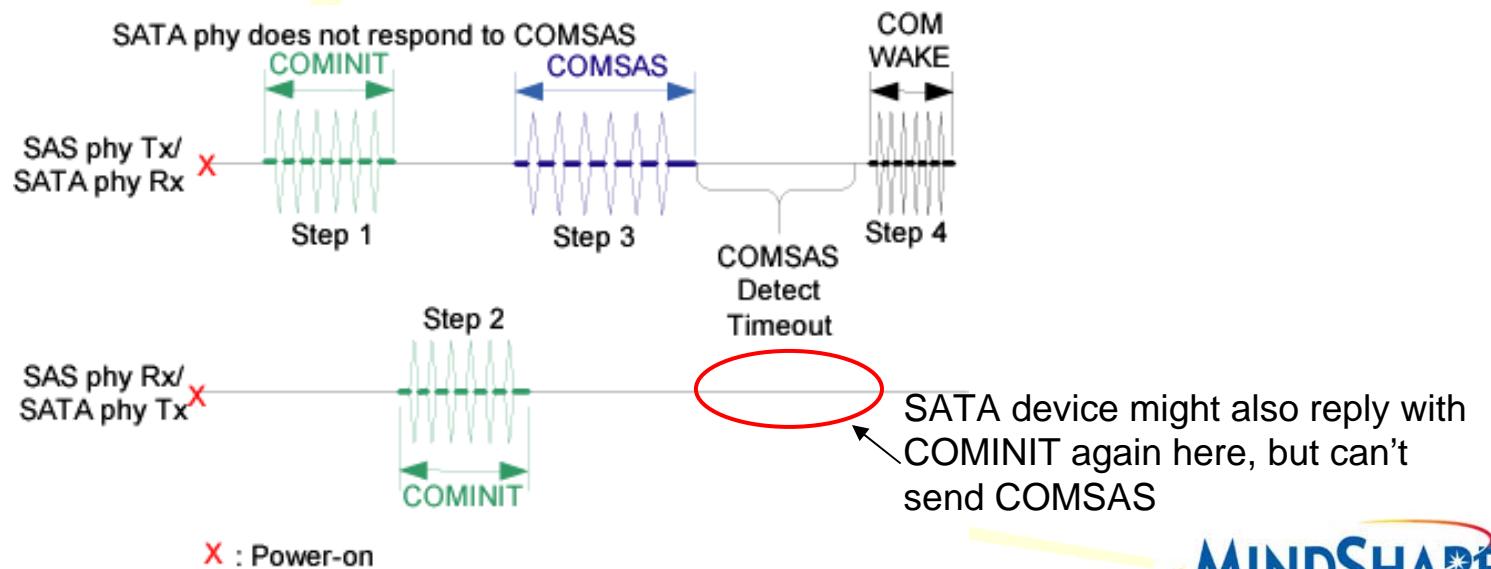
- As an initiator, A queries the expander SMP target to learn the number of phys in the expander.
- Next, SMP DISCOVER is sent for each phy in the expander starting with phy0 and incrementing forward until all of them have been explored.
  - In that process it learns that end device B is present, has 2 phys attached to the expander at a particular rate, and supports SSP protocol.

# Initialization Example

- Device A has now mapped all the addresses it can find. It might need to write table entries into the expander but in this case that's unnecessary because all the expander connections are direct.
- Device A could now send an OPEN to access device B.

# Initialization Example

- Now assume the SATA drive (Z) powers up and exchanges COMINIT with the expander.
- Expander sends COMSAS, which Z may ignore or may interpret as a COMRESET, causing it to reset and send COMINIT again
- Expander detects Z as a SATA device (or a broken SAS device), and sends COMWAKE, which appears to Z like a SATA host. Next step is speed negotiation.

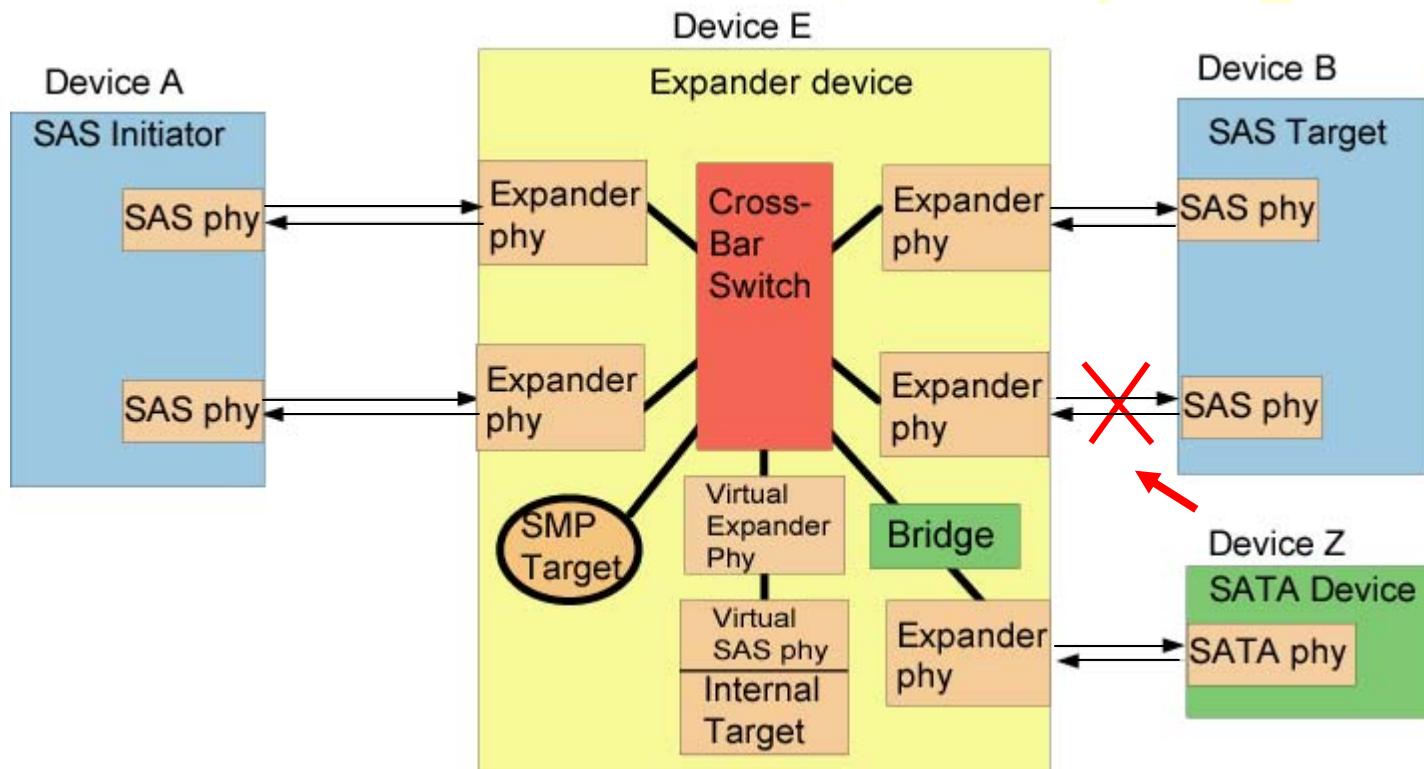


# Initialization Example

- At this point, expander has to send a broadcast change primitive to report that device Z was found. The broadcast only indicates that a change has occurred (keeping expanders simple).
  - Device B is a target only and ignores this
  - Device A will have to repeat the discovery process to find out what has changed.
  - Device A learns that there is a new SATA device (SATA devices don't have addresses, though, so Expander creates a SAS address for it based on the phy number and a base address of the expander)

# Working Phy Fails (140)

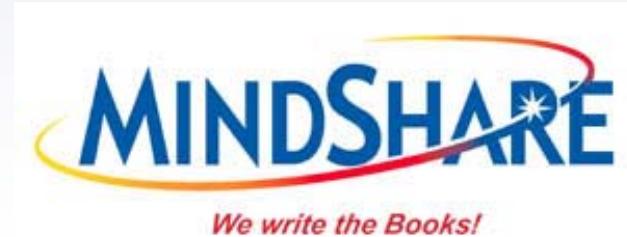
- Now assume a phy on B fails. Expander will again send a broadcast change message, forcing A to repeat the discovery process.



# Initialization Example

- The internal SAS target port, typically used for enclosure services, will have a unique SAS address.
- For enclosure services, the logic associated with the internal target could directly access the services, or could convert the commands into another protocol such as I<sup>2</sup>C
  - But I<sup>2</sup>C is slow (1-bit wide at 100KHz), so it's better to implement it as a full target device.
  - Better to talk to the services with SSP or SMP? Spec only documents SSP because SMP might not give enough payloads (limited to 1k frames per function).

# *Software Initialization*



# Mode Pages (141)

- The SBC-3 (SCSI Block Commands) standard defines several optional mode pages that software can access to configure devices. Optional Mode Sense and Mode Select commands are used to access these pages.
- A few existing pages are modified for SAS protocol:
  - Disconnect-Reconnect page
  - Protocol-Specific Port page (short and long formats)
  - Protocol-Specific Log page
- A new one is added: SAS Phy Mode Descriptor

# Disconnect-Reconnect Mode Page (143)

- Every transport modifies this for its own purposes, SSP shown here

Byte	7	6	5:0	
0	PS	SPF (0b)	Page Code (02h)	Page Savable – when read, indicates whether target can save page contents in non-volatile memory. If so, initiator can set the SP bit in Mode Select command to write the contents to the saved parameters.
1			Page Length (0Eh)	Max time a target can keep connection open without sending a frame (units = 100µs). If expires, target must issue DONE.
2 to 3			Reserved	
4 to 5			Bus Inactivity Time Limit	Max time a target can keep connection open overall (units = 100µs). If expires, target must issue DONE.
6 to 7			Reserved	
8 to 9			Maximum Connect Time Limit	Should not be used – initiators cannot be designed assuming they can turn this on.
10 to 11			Maximum Burst Size	Initiator can send this much data without waiting on XFER_RDY from target (acts like an implicit XFER_RDY). Intended for high-latency environments like iSCSI, <u>should not be used for SAS</u> .
12 to 13			Reserved	
14 to 15			First Burst Size	

# Protocol-Specific Port Control Mode Page – Short Format (145)

	7	6	5	4	3	2	1	0
0	PS	SPF =0						Page Code = 19h
1								Page Length = 6h
2	Reserved		Ready LED Meaning		Protocol ID = 6h			
3					Reserved			
4-5				I_T Nexus Loss Time				
6-7					Initiator Response Timeout			

- Allows application layer to control features of the transport

SAS SSP Protocol

If target already sent XFER\_RDY but initiator is taking too long to send data, resources are needlessly tied up. If no response after this time, abort the command (units in ms, zero means wait forever). **Note:** this is timing the initiator, which could still be doing useful work on another I\_T\_L\_Q nexus, so use with caution.

If target tries to OPEN initiator to return previously connected data but can't succeed, allow this time before giving up (units in ms). Could be that dword synchronization was lost and link is running the link reset sequence. Recommended setting: 2ms. Setting of zero implies vendor-specific, all F's means keep trying forever.

# Protocol-Specific Port Control

## Mode Page – Long Format (148)

- Provides a means of getting DISCOVER information from an end device that did not implement an SMP port (to save cost).
- Information is provided for all the phys of this device.

Sub-Page in this case is the Long page format

Byte	Field(s)	
0	PS	SPF (1h)
1	Subpage Code (01h)	
2 to 3	Page Length	
4 to 6	Reserved	
7	Number of Phys	
8 to n	First SAS phy mode descriptor (see next slide)	
n to m	Next SAS phy mode descriptor	
m to last	Last SAS phy mode descriptor	

# SAS Phy Mode Descriptor (149)

	Bit 7	6	5	4	3	2	1	0
Byte 0					Reserved			
1					Phy Identifier			
2 to 3					Reserved			
4	Rsvd	Attached Device Type				Reserved		
5		Reserved			Negotiated Physical Link Rate			
6		Reserved		Attached SSP Init Port	Attached STP Init Port	Attached SMP Init Port	Attached SATA Host	
7		Reserved		Attached SSP Target Port	Attached STP Target Port	Attached SMP Target Port	Attached SATA Device	
8 to 15				SAS Address				
16 to 23				Attached SAS Address				
24				Attached Phy Identifier				
25 to 31				Reserved				
32	Programmed Min Physical Link Rate				Hardware Min Physical Link Rate			
33	Programmed Max Physical Link Rate				Hardware Max Physical Link Rate			
34 to 41				Reserved				
42 to 43				Vendor Specific				
44 to 47				Reserved				

# Phy Mode Descriptor Fields

- Several fields refer to the “attached” phy
- They return information received from the IDENTIFY address frame (by the phy being queried) during the link reset sequence

DISCOVER querying this phy gets information about the attached phy



# Device Type (150)

- Attached Device Type field
  - If an end device is indicated, initiator can start opening SSP, STP, or SMP connections (as indicated by the Attached Initiator and Target bits)
  - If an expander is indicated, initiator should open SMP connections to perform further discovery of its phys

Value	Device type
00b	End device
01b	Edge expander
10b	Fanout expander

# Negotiated Link Rate (151)

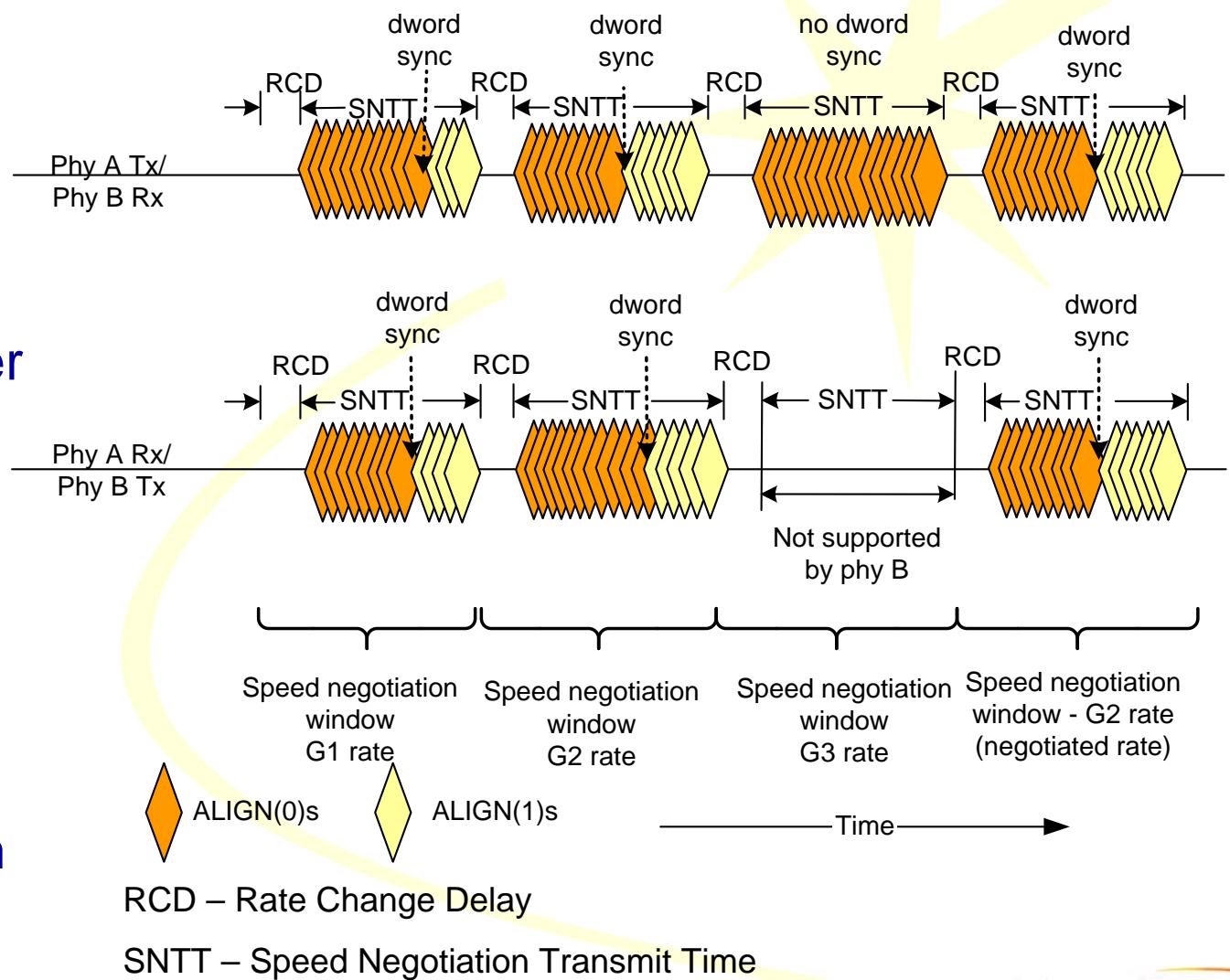
- Negotiated Physical Link Rate field (4 bits)
  - The current physical link rate in use by the phy
  - Also indicates phy state when it is not running at speed yet

Value	Negotiated physical link rate
0h	Unknown – place holder for application client data structure
1h	Phy is disabled (via PHY CONTROL)
2h	<b>Phy reset problem</b> (see next slide)
3h	SATA in spin-up hold state
4h	SATA Port Selector, rate not yet negotiated
8h	1.5 Gbps
9h	3.0 Gbps

} These two cases  
are only  
interesting for  
Expanders

# Phy Reset Problem (152)

- The last SAS speed negotiation window is expected to work (more on this in phy layer initialization)
- If it fails, this is called a phy reset problem
- Solution: Phys must rerun the phy reset sequence from the beginning



# Reporting Phy Reset Problem

- Reported in the DISCOVER function's **Negotiated Physical Link Rate** field
- Counted in the REPORT PHY ERROR LOG function's **Phy Reset Problem Count** field
- Resolution
  - Phy reruns the phy reset sequence and hopes for better results.
  - If it keeps failing, an initiator accessing the expander via another phy can use the PHY CONTROL function's Programmed Min/Max Physical Link Rate fields to disable the rate that is failing

# SATA Spin-up Hold

- Serial ATA drives spin up automatically after finishing speed negotiation
  - They don't wait for a NOTIFY (ENABLE SPINUP) primitive like SAS drives
  - In an enclosure they could all spin up at once, requiring a larger power supply
- To avoid this - don't finish speed negotiation
  - When expander sees a SATA drive rather than SAS in the OOB sequence, it stops the phy reset sequence
  - Reports in the SMP DISCOVER function's Negotiated Physical Link Rate field
  - Waits for software to use the SMP PHY CONTROL function's Phy Operation field to request a Link Reset before proceeding

# Protocol-Specific Log Page (154)

- SCSI defines page 18 for LUN and page 19 for port level information. SAS defines page 18 as shown on next slide.
- Log pages are accessed by the commands Log Sense (for reads) and Log Select (for writes)

# Log Pages (155)

- Allows getting REPORT PHY ERROR LOG information from device without SMP port.
- One log parameter per target port
  - Numbered by relative target port identifier
- One phy log descriptor for every phy in the port (next slide)

Byte	Field(s)
0	Page Code (18h)
1	Reserved
2 to 3	Page Length
4 to n	Protocol-Specific log parameters

Byte	Field(s)	
0 to 1	Parameter Code (relative target port identifier)	
2	Parameter Control Bits	
3	Parameter Length	
4	Reserved	Protocol Identifier (6h)
5 to 6	Reserved	
7	Number of Phys	
8 to n	Phy log descriptor	

# SAS Phy Log Descriptor (156)

	Bit 7	6	5	4	3	2	1	0				
Byte 0	Reserved											
1	Phy Identifier											
2 to 3	Reserved											
4	Rsvd	Attached Device Type		Reserved								
5	Reserved			Negotiated Physical Link Rate								
6	Reserved			Attached SSP Init Port	Attached STP Init Port	Attached SMP Init Port	Reserved					
7	Reserved			Attached SSP Target Port	Attached STP Target Port	Attached SMP Target Port	Reserved					
8 to 15	SAS Address											
16 to 23	Attached SAS Address											
24	Attached Phy Identifier											
25 to 31	Reserved											
32 to 35	Invalid Dword Count											
36 to 39	Running Disparity Error Count											
40 to 43	Loss of Dword Synchronization											
44 to 47	Phy Reset Problem											

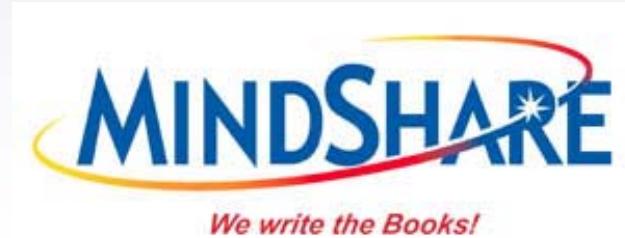
# Log Page Terms (157)

Fields are the same as Phy Mode Descriptor described earlier except for the last 4 items:

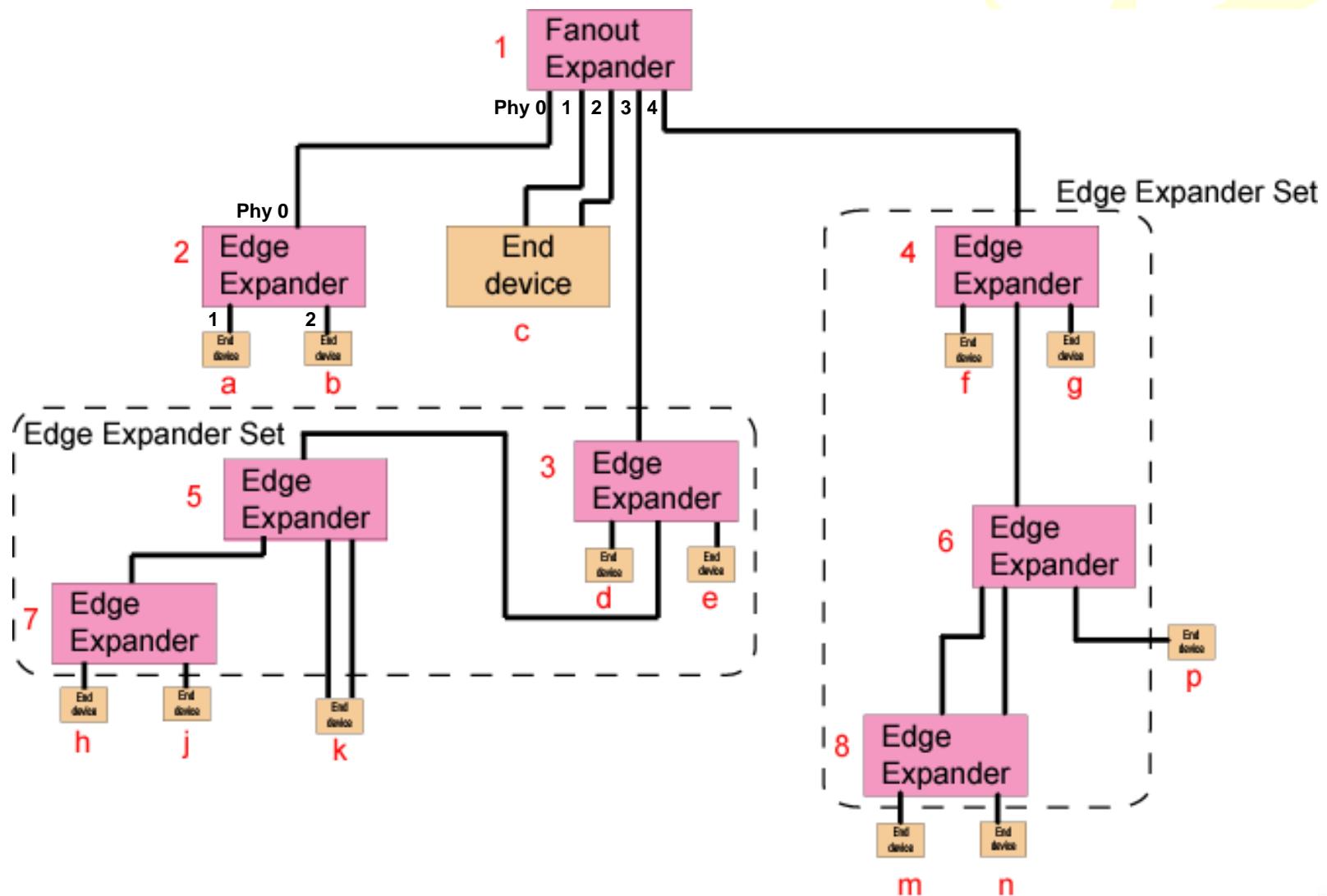
- Invalid dword count
  - Number of dwords received with errors, such as invalid characters
- Running disparity error count
  - Number of dwords received with disparity errors
- Loss of dword synch
  - Number of times dword synch was lost
- Phy reset problem
  - Number of times phy reset sequence failed

# *Chapter 7*

## *Discovery Process*

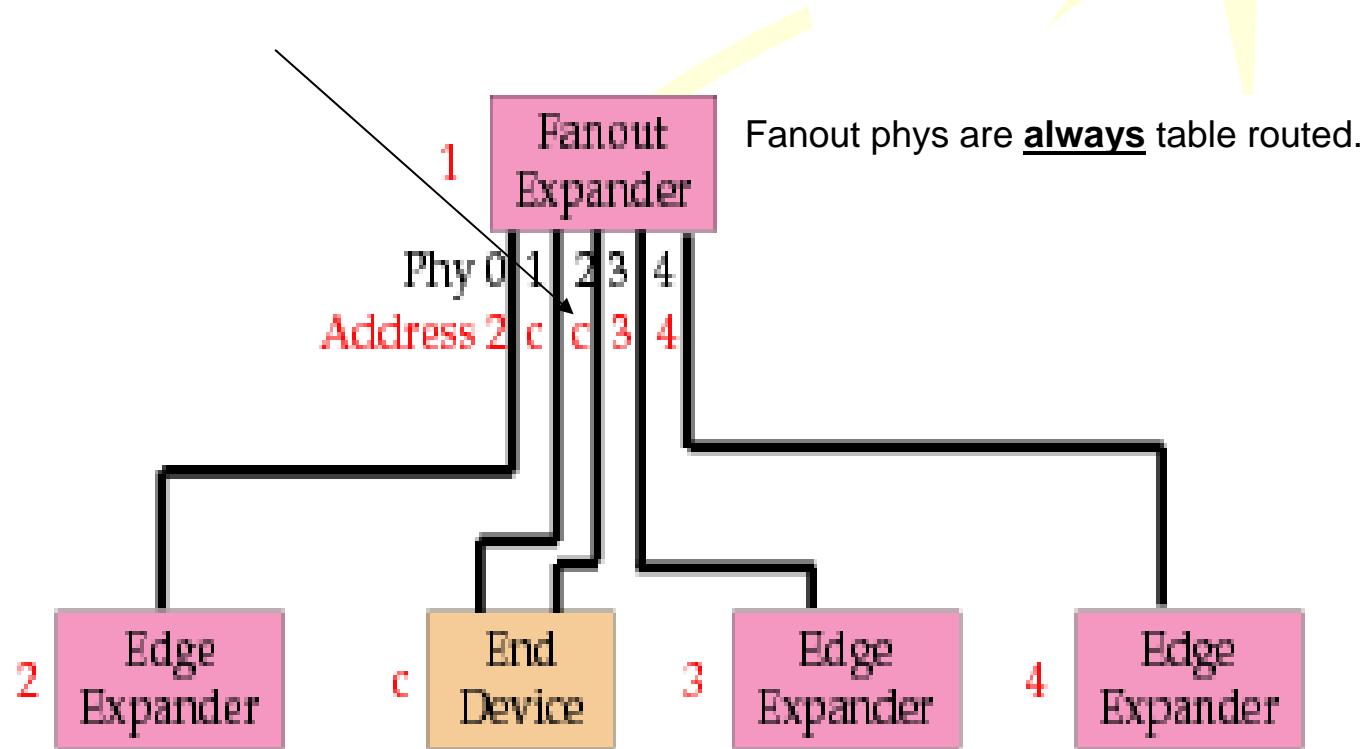


# Example Topology (160)



# Discovery Process (161)

- After phys exchange Identify frames, they know each others SAS address and device type
- Here, two phys see End Devices (and won't need table entries), while the other 3 see expanders



# Discover Process (162)

- Next step is to learn the other addresses in the system, using SMP requests
- If attached expanders are self-configuring, an initiator could get list of available addresses by reading their routing tables.
  - Self-configuring capability is indicated in the response to REPORT GENERAL request

SMP REPORT  
GENERAL  
request.

Byte	Field(s)
0	SMP Frame Type (40h)
1	Function (00h)
2 to 3	Reserved

# Report General Response (163)

Byte	Field(s)		
0	SMP Frame Type (41h)		
1	Function (00h)		
2	Function Result		
3	Reserved		
4 to 5	Expander Change Count		
6 to 7	Expander Route Indexes		
8	Reserved		
9	Number of Phys		
10	Reserved	Configuring	Configurable
11 to 27	Reserved		

Other details of this response  
are covered later

Indicates table can be  
configured (meaning it's  
not self-configuring)

Indicates self-configuring  
expander is still in the process  
of configuration

# Read Routing Table

- To read the routing table, send **REPORT ROUTE INFORMATION** request for each index of each phy.
  - Even with this information, though, initiators will still have to scan the domain to identify target devices, since device type is not included with table entries.

# Report Route Info – Request (164)

One request  
needed for each  
entry in the table.

Byte	Field(s)
0	SMP Frame Type (40h)
1	Function (13h)
2 to 5	Reserved
6 to 7	Expander Route Index
8	Reserved
9	Phy Identifier
10 to 11	Reserved

Routing Table

Expander Route Index	Phy Identifier			
	0	1	...	n
0	Entry	Entry		Entry
1	Entry	Entry		Entry
...				
M	Entry	Entry	...	Entry

# Report Route Info - Response

Byte	Field(s)	
0	SMP Frame Type (41h)	
1	Function (12h)	
2	Function Result	
3 to 5	Reserved	
6 to 7	Expander Route Index	
8	Reserved	
9	Phy Identifier	
10	Reserved	
11	Disabled	Reserved
12 to 15	Reserved	
16 to 23	Routed SAS Address	
24 to 39	Reserved	

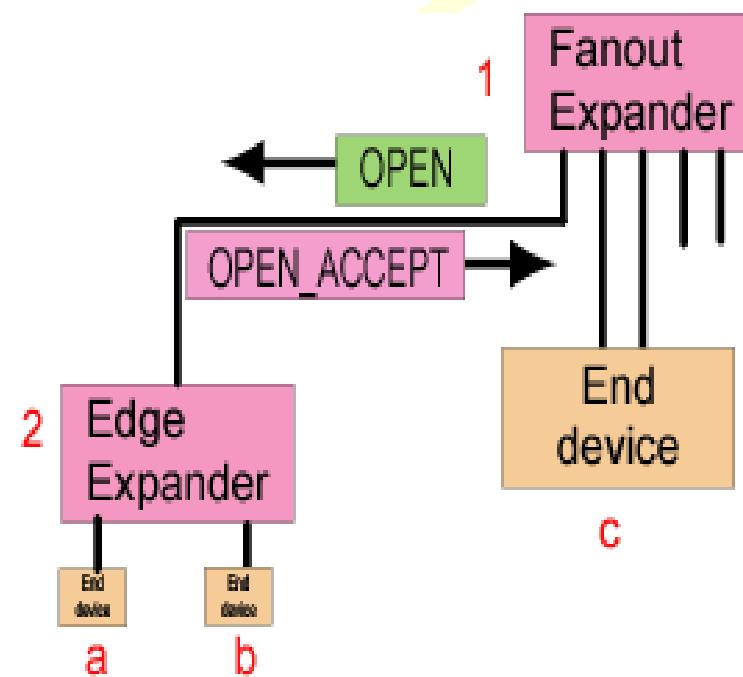
List of possible results  
on next slide

# Function Result (165)

Value	Function Result	Description
00h	SMP Function Accepted	Good result
01h	Unknown SMP Function	Target doesn't support the request
02h	SMP Function Failed	Request failed for some reason
03h	Invalid Request Frame Length	Request frame length was invalid
10h	Phy Does Not Exist	For functions including a <b>Phy Identifier</b> field, the <b>Phy Identifier</b> was out of range
11h	Index Does Not Exist	For functions including an <b>Expander Route Index</b> field, the <b>Expander Route Index</b> was out of range (or the specified phy doesn't have a routing table at all)
12h	Phy Does Not Support SATA	REPORT PHY SATA requested, but no SATA device attached.
13h	Unknown Phy Operation	Unknown PHY CONTROL <b>Phy Operation</b> requested.

# Discover Process

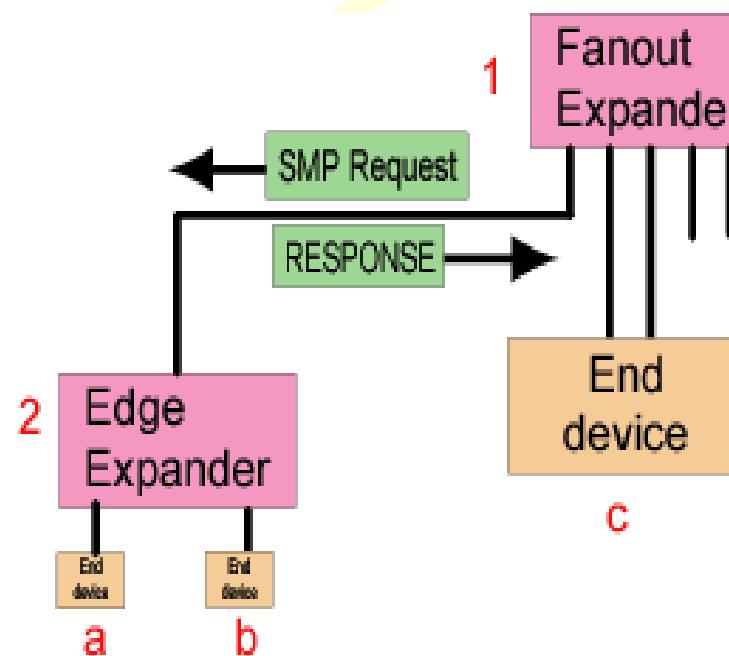
- If attached expanders are not self-configuring, discover which addresses reside beyond it
  - First, open a connection



# Report General Request

- Second, Send **REPORT GENERAL** request to that device.

Byte	Field(s)
0	SMP Frame Type (40h)
1	Function (00h)
2 to 3	Reserved



# Report General Response (167)

Byte	Field(s)		
0	SMP Frame Type (41h)		
1	Function (00h)		
2	Function Result = 0		
3	Reserved		
4 to 5	Expander Change Count		
6 to 7	Expander Route Indexes		
8	Reserved		
9	Number of Phys = 3		
10	Reserved	Configuring	Configurable
11 to 27	Reserved		

Confirm request

Status of the response:  
0 = good result

Used to reduce activity in response to Broadcast (Changes)

Max number of route indexes/phy.

Number of phys in this device, including any virtual phys

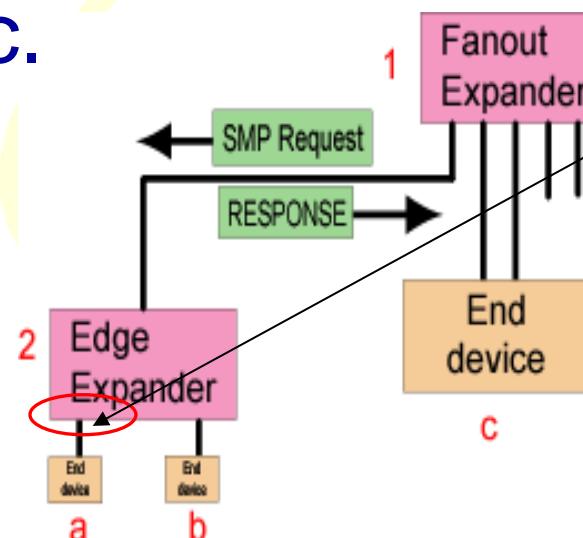
Indicates whether expander is self-configuring

Indicates self-configuration in progress (if supported)

# Discover Request (169)

- Next, send SMP DISCOVER request for each phy, starting with phy 0, then 1, then 2, etc.

Byte	Field(s)
0	SMP Frame Type (40h)
1	Function (10h)
2 to 8	Reserved
9	Phy Identifier = 1
10 to 11	Reserved



# Discover Response – part 1

Byte	Field(s)		
0	SMP Frame Type (41h)		
1	Function (10h)		
2	Function Result		
3 to 8	Reserved		
9	Phy Identifier = 1		
10 to 11	Reserved		
12	Attached Device Type = 001	Reserved	
13	Reserved	Negotiated Physical Link Rate	Data rate negotiated between neighbors
14	Reserved	Attached Initiator Port bits	Initiator/Target and Protocol (see next slide)
15	Reserved	Attached Target Port bits	
16 to 55	...see parts 2 and 3...		

Confirm request parameters

Attached Device

- 000 – No device attached
- 001 – End device
- 010 – Edge Expander
- 011 – Fanout Expander

MINDSHARE  
We write the Books!

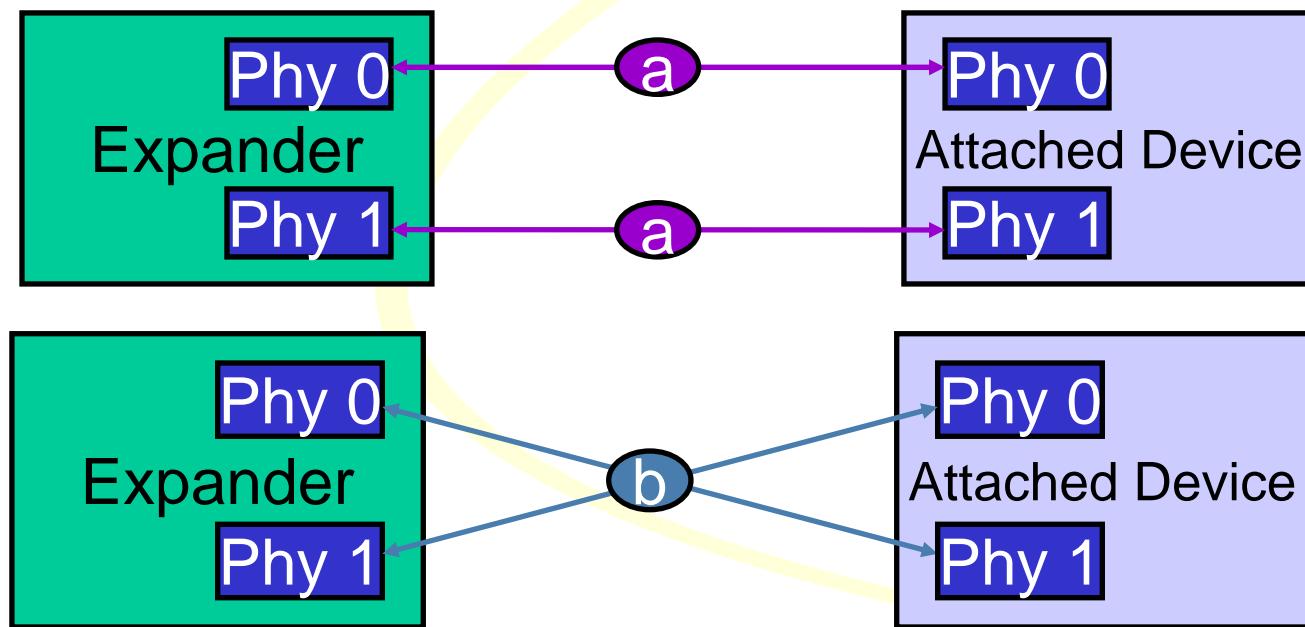
# Initiator and Target Fields

- Attached SSP/STP/SMP Initiator/Target bits
  - Any number of SSP, STP, and/or SMP initiator and/or target bits may be set based on the incoming IDENTIFY address frame
- Attached SATA host bit is optional
  - Attached SATA host not supported, but expander might report it
- Combination of SATA Target and SATA Port Selector shown on next slide

7	6	5	4	3	2	1	0
		Reserved		Attached SSP Initiator	Attached STP Initiator	Attached SMP Initiator	Attached SATA Host
Attached SATA Port Selector		Reserved		Attached SSP Target	Attached STP Target	Attached SMP Target	Attached SATA Target

# Phy Identifier (171)

- Knowing the internal phy number is necessary to see the whole topology. In this example of a wide link, either “a” or “b” topology could exist; phy identifier clarifies it.



# SATA Target Fields (172)

<b>Attached SATA Port Selector</b>	<b>Attached SATA Target</b>	<b>Description</b>
0	0	No port selector or device detected.
0	1	SATA device phy detected.
1	0	Attached phy is Port selector host phy and either: - attached phy is the inactive host phy or - Attached phy is the active host phy but the SATA device is not present or not ready.
1	1	Attached phy is Port selector host phy and SATA device is detected on the other side of it.

# Discover Response – part 2 (173)

Byte	Field(s)		
0 to 15	...see part 1...		
16 to 23	SAS Address = 2		
24 to 31	Attached SAS Address = a		
32	Attached Phy Identifier = 0		
33 to 39	Reserved		
40	Programmed Min Rate	Hardware Min Rate	These fields indicate what the h/w is capable of doing, but the s/w can program the rate to be equal to or less than that.
41	Programmed Max Rate	Hardware Max Rate	
42	Phy Change Count		
43	Virtual Phy	Reserved	Number of BROADCAST (CHANGES) initiated by this phy. It does wrap around, though, so value might appear aliased.
44	Reserved	Routing Attribute = 0h	Routing 0h – Direct 8h – Table 9h – Subtractive
45	Rsvd	Connector Type	
46	Connector Element Index		
47	Connector Physical Link		
48-49	Reserved		
50 to 51	Vendor-specific		

# Discover Response – part 3 (176)

Byte	Field(s)				
0 to 15	...see part 1...				
16 to 23	SAS Address = 2				
24 to 31	Attached SAS Address = a				
32	Attached Phy Identifier = 0				
33 to 39	Reserved				
40	Programmed Min Rate	Hardware Min Rate			
41	Programmed Max Rate	Hardware Max Rate			
42	Phy Change Count				
43	Virtual Phy	Reserved	Partial Pathway Timeout Value		
44	Reserved		Routing Attribute = 0h		
45	Rsvd	Connector Type			
46	Connector Element Index				
47	Connector Physical Link				
48-49	Reserved				
50 to 51	Vendor-specific				



Indicates type of connector used to access this phy, as reported by enclosure services process.

Of all the elements tracked by enclosure services process, this is the index of the element for this connector.

Reports the physical link in the connector that connects to this phy.

# Attached Expander

- If the attached device is another expander, send a request to that address by repeating the process
  1. Open a connection to that address
  2. Send REPORT GENERAL request
  3. Send DISCOVER request to phy 0, 1, etc.
  4. Addresses found this time are the first level beyond the direct connection to the original expander and will need to be written into the address table

# Updating the Routing Table

- For an expander with a routing table that is not self-configuring, initiators need to update the table.
  1. Initiators create a map of the topology as they discover it, later go back to update tables as needed.
  2. Order of discovery starts with phy0 and increments through all expander phys before moving on to the next device

## Updating (cont.)

3. Update table entries with CONFIGURE ROUTE INFORMATION request to the expander address, specifying the phy and index

Note that this discovery must be “breadth first” and, since all initiators will do it, it’s important that they all do it the same way

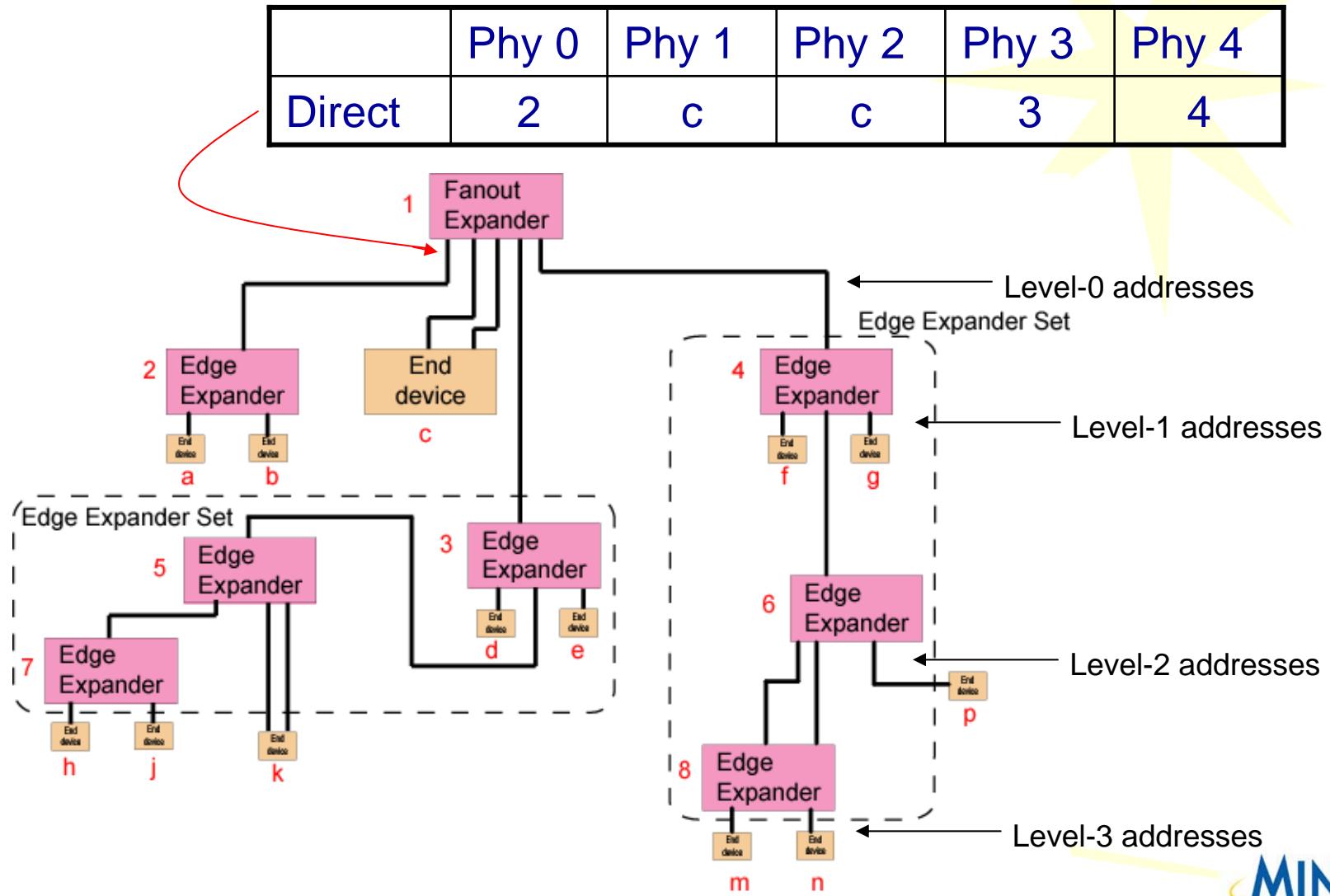
# Configure Route Info Request (177)

Byte	Field(s)
0	SMP Frame Type (40h)
1	Function (90h)
2 to 5	Reserved
6 to 7	Expander Route Index
8	Reserved
9	Phy Identifier
10 to 11	Reserved
12	Disable
13 to 15	Reserved
16 to 23	Routed SAS Address
24 to 39	Reserved

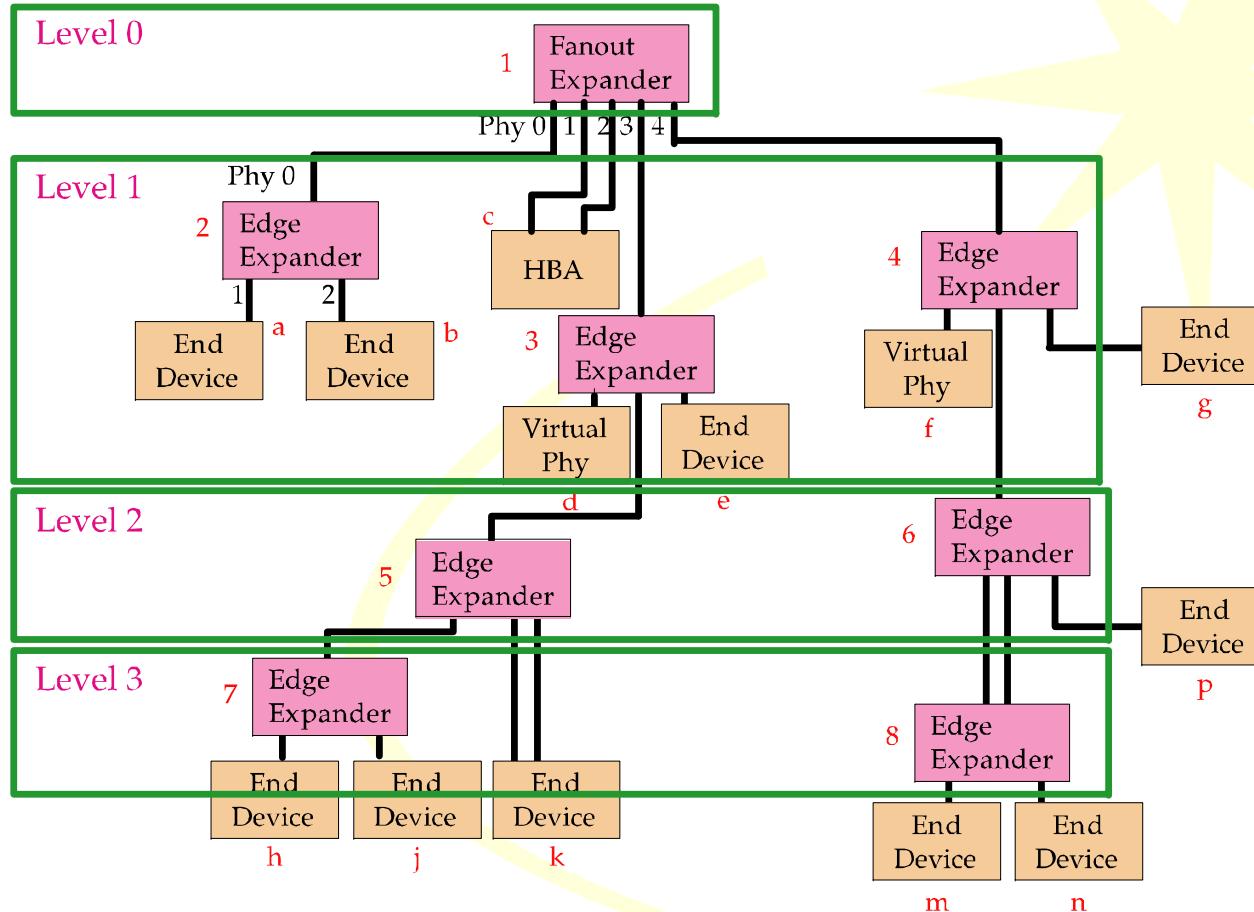
Used to write the SAS Address into the table entry specified by Index and Phy

# Level 0 Entries (180)

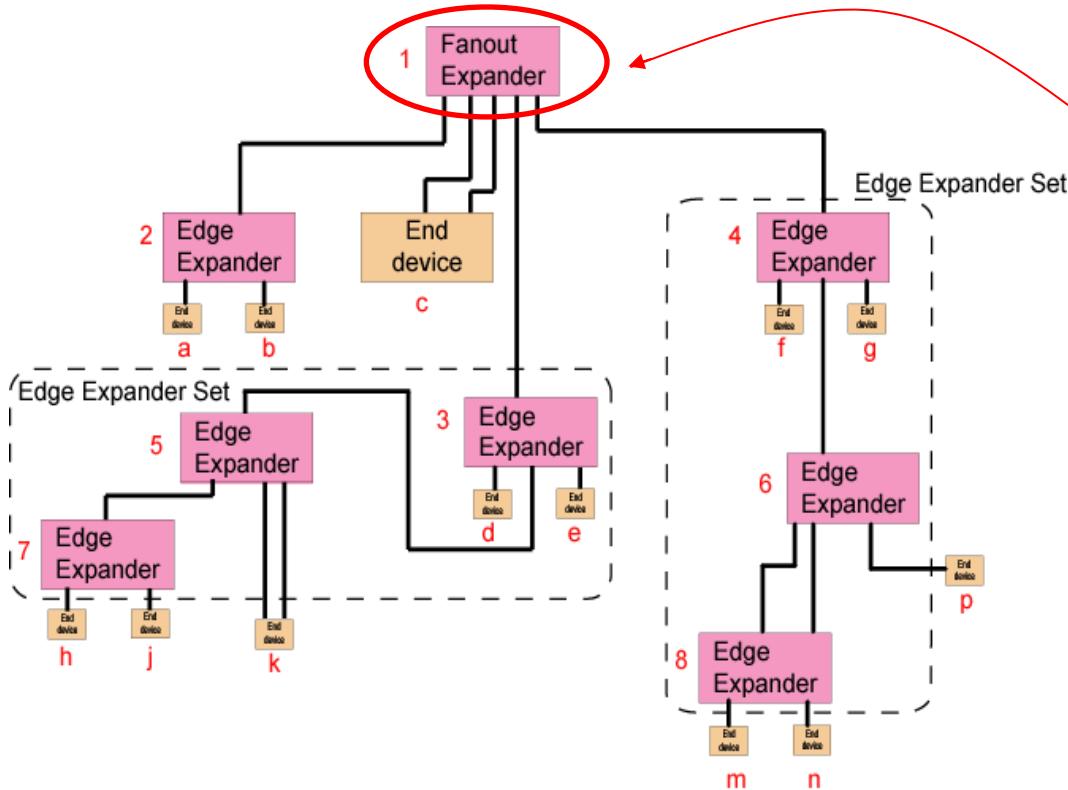
Automatically setup by h/w, directly connected addresses are not included in routing tables



# Topology Levels (182)



# Fanout Routing Table Example

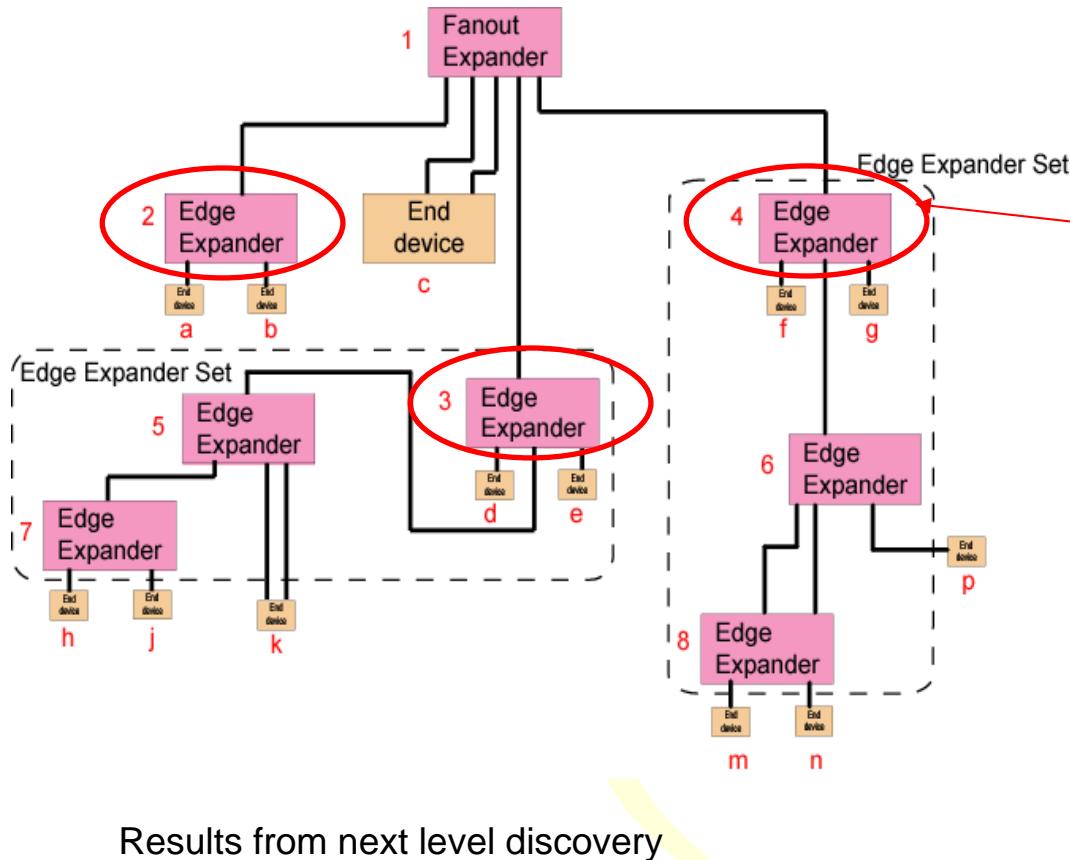


Note: Index 0 corresponds to level 1.

	Phy 0	Phy 3	Phy 4
Index 0			
Index 1			
Index 2			
Index 3			
Index 4			
Index 5			
Index 6			
Index 7			
Index 8			
Index 9			
Index 10			
Index 11			

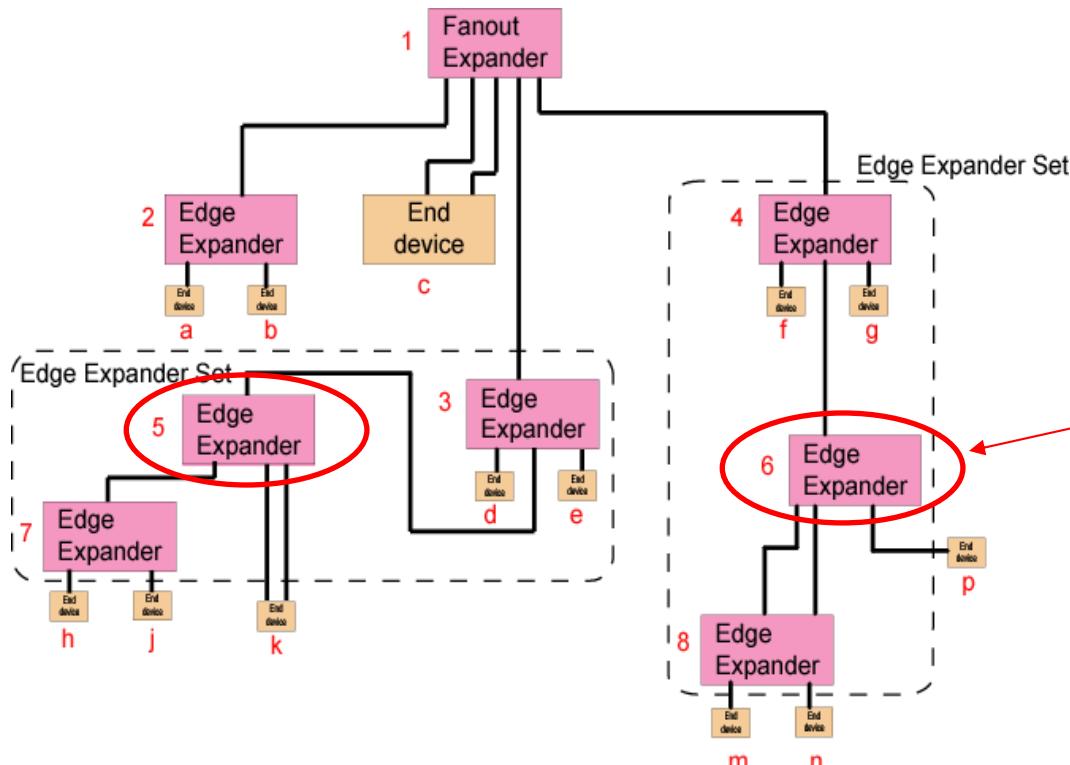
We don't show entries for phys 1 and 2, although they do appear in the table because they have direct connects and those entries would all be empty. How will the table be filled in?

# Fanout Routing – Level 1 (181)



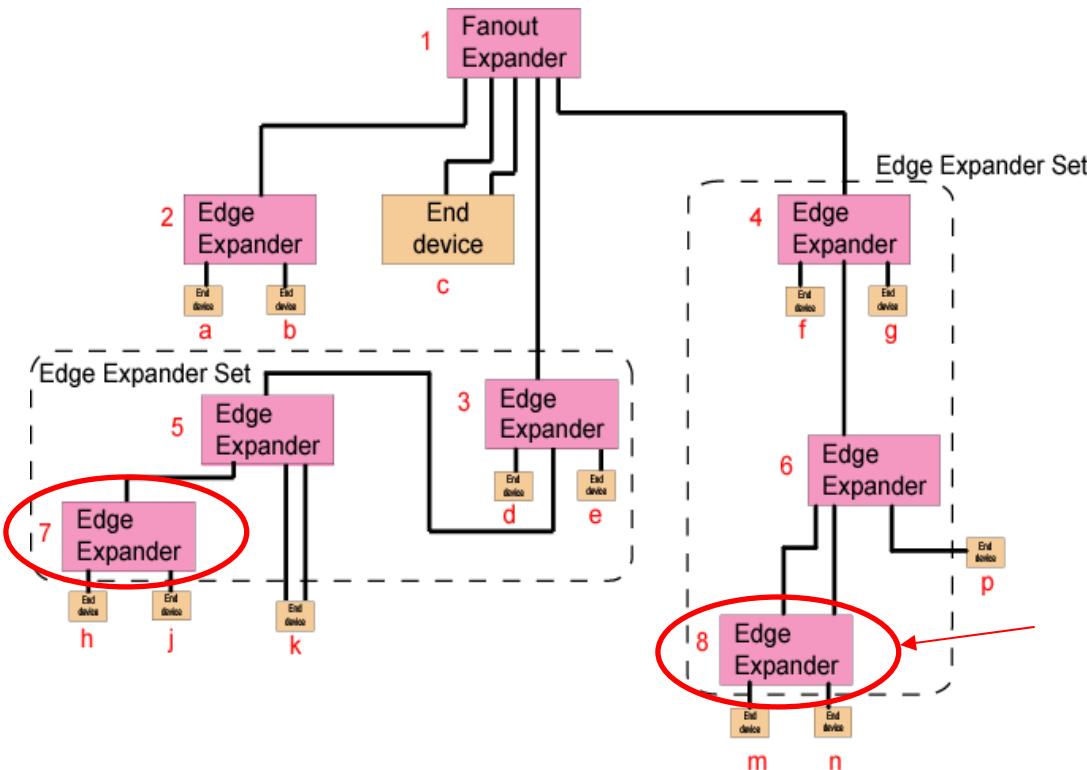
	Phy 0	Phy 3	Phy 4
Index 0	1	1	1
Index 1	a	d	f
Index 2	b	5	6
Index 3		e	g
Index 4			
Index 5			
Index 6			
Index 7			
Index 8			
Index 9			
Index 10			
Index 11			

# Fanout Routing – Level 2 (183)



	Phy 0	Phy 3	Phy 4
Index 0	1	1	1
Index 1	a	d	f
Index 2	b	5	6
Index 3		e	g
Index 4		3	4
Index 5		7	8
Index 6		k	8
Index 7		k	p
Index 8			
Index 9			
Index 10			
Index 11			

# Fanout Routing – Level 3 (184)



	Phy 0	Phy 3	Phy 4
Index 0	1	1	1
Index 1	a	d	f
Index 2	b	5	6
Index 3		e	g
Index 4		3	4
Index 5		7	8
Index 6		k	8
Index 7		k	p
Index 8		5	6
Index 9		h	6
Index 10		j	m
Index 11			n

# Non-Optimized Routing Table

	Phy 0	Phy 1	Phy 2	Phy 3	Phy 4	<u>Optimization Opportunities</u>
Index 0	1			1	1	“Self-Reference” entries for the device being programmed
Index 1	a			d	f	
Index 2	b			5	6	
Index 3				e	g	
Index 4				3	4	Addresses which are directly attached
Index 5				7	8	
Index 6				k	8	Duplicate entries
Index 7				k	p	
Index 8				5	6	
Index 9				h	6	
Index 10				j	m	
Index 11					n	

# Optimized Routing Table

	Phy 0	Phy 1	Phy 2	Phy 3	Phy 4
Index 0	a			d	f
Index 1	b			5	6
Index 2				e	g
Index 3				7	8
Index 4				k	p
Index 5				h	m
Index 6				j	n

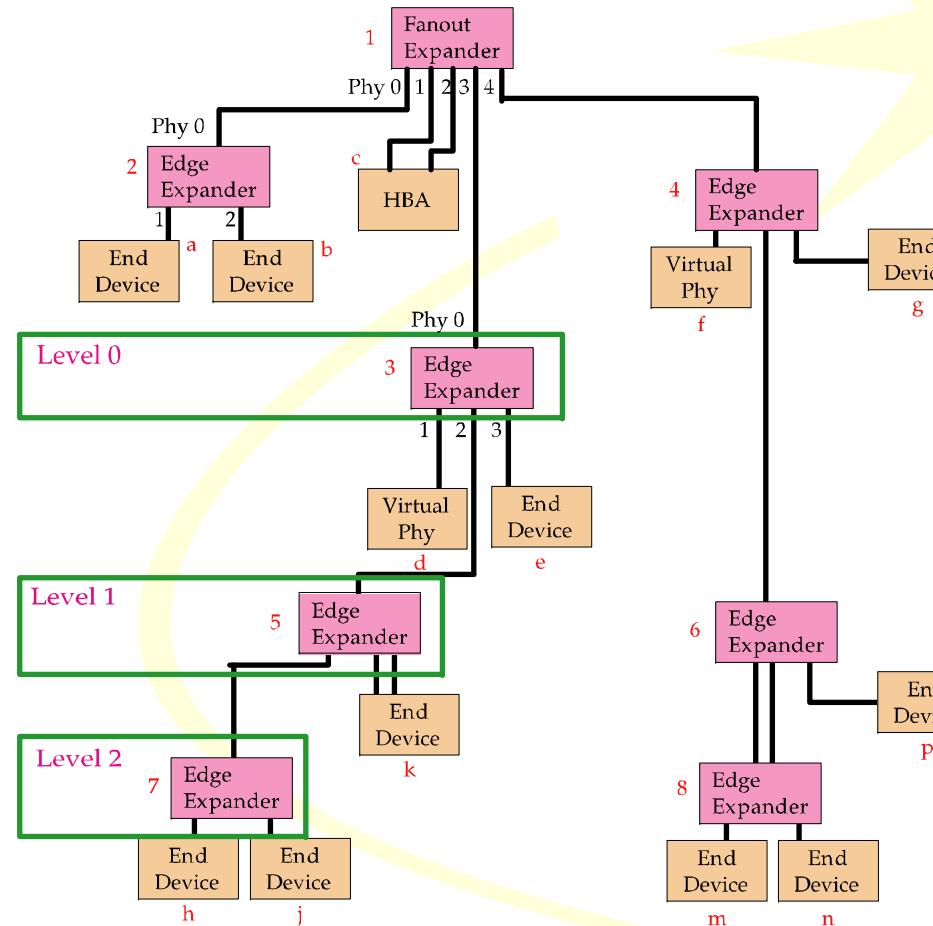
Loopback and Redundant entries removed

# Possible Table Implementations

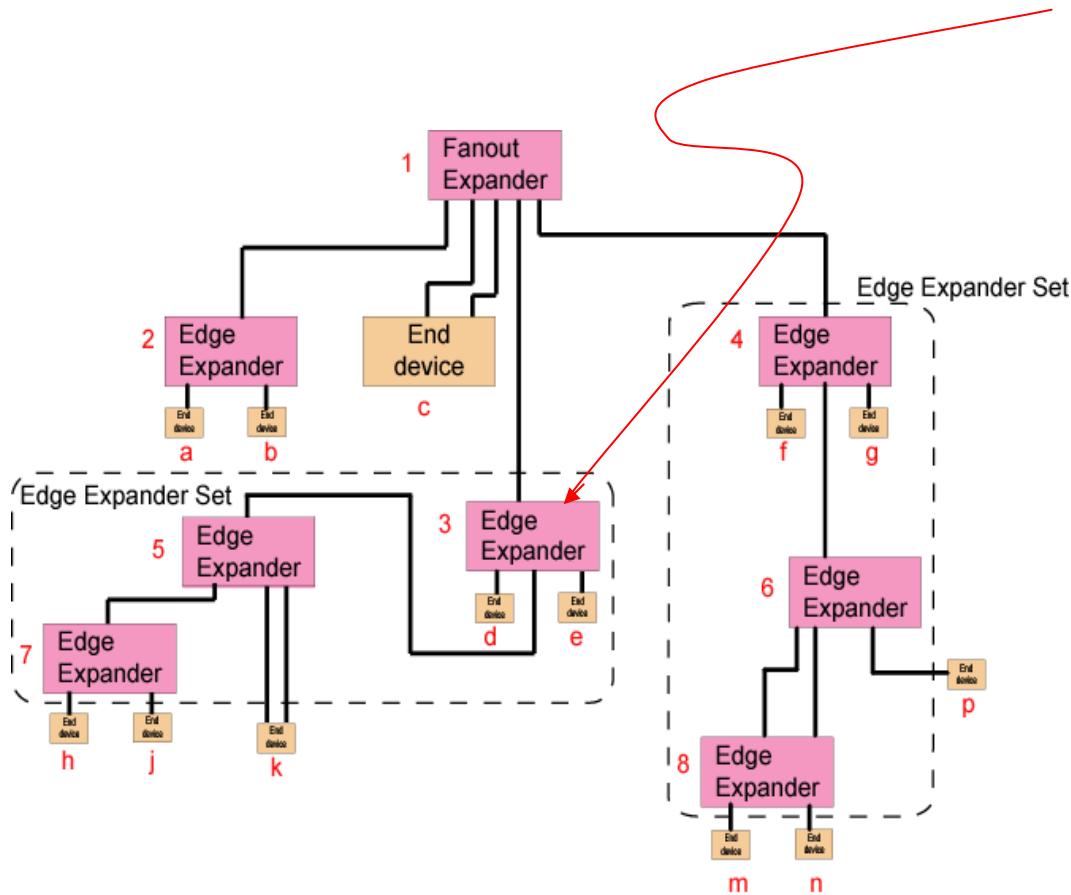
Phy, Index	Attached Address
0,0	a
0,1	b
3,0	d
3,1	5
3,2	e
3,3	7
3,4	k
3,5	h
3,6	j
4,0	f
4,1	6
4,2	g
4,3	8
4,4	p
4,5	m
4,6	n

Address	Phy
a	0
b	0
d	3
5	3
e	3
7	3
k	3
h	3
j	3
f	4
6	4
g	4
8	4
p	4
m	4
n	4

# Discovery for Expander 3 (188)

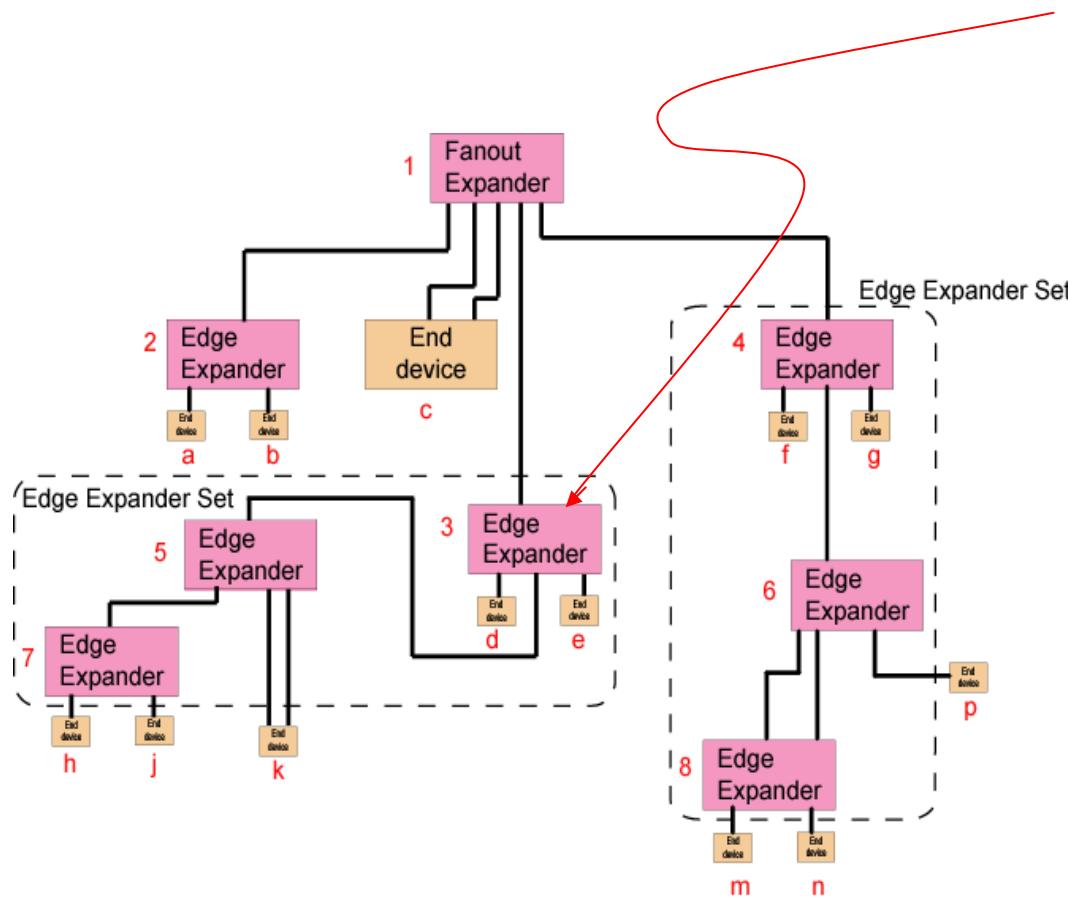


# Edge Expander Routing



	Phy 1	Phy 2	Phy 3
Index 0			
Index 1			
Index 2			
Index 3			

# Edge Expander Routing Result (189)



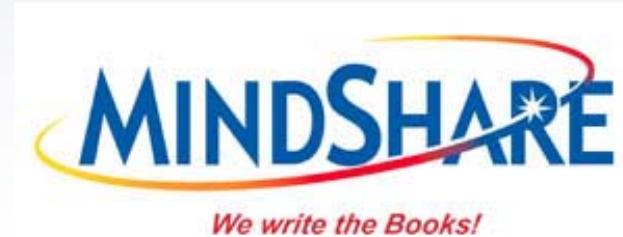
	Phy 1	Phy 2	Phy 3
Index 0	3		
Index 1	7		
Index 2	k		
Index 3	k		
Index 4	5		
Index 5	h		
Index 5	j		

# Summary

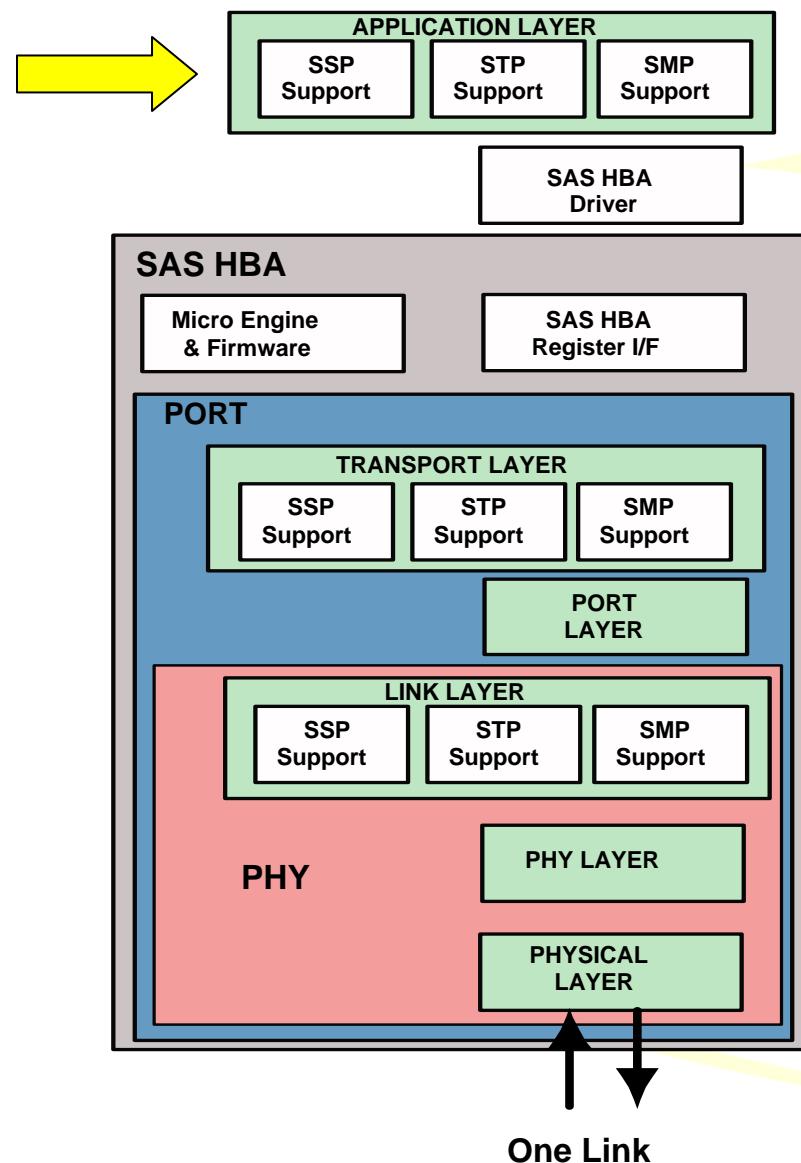
- REPORT GENERAL response gives max depth of route table and number of phys
  - Total table entries will never exceed Route Index times Number of Phys
- Read table - REPORT ROUTE INFO
- Write table - CONFIGURE ROUTE INFO

# *Chapter 8*

# *Application Layer*



# Layer Context



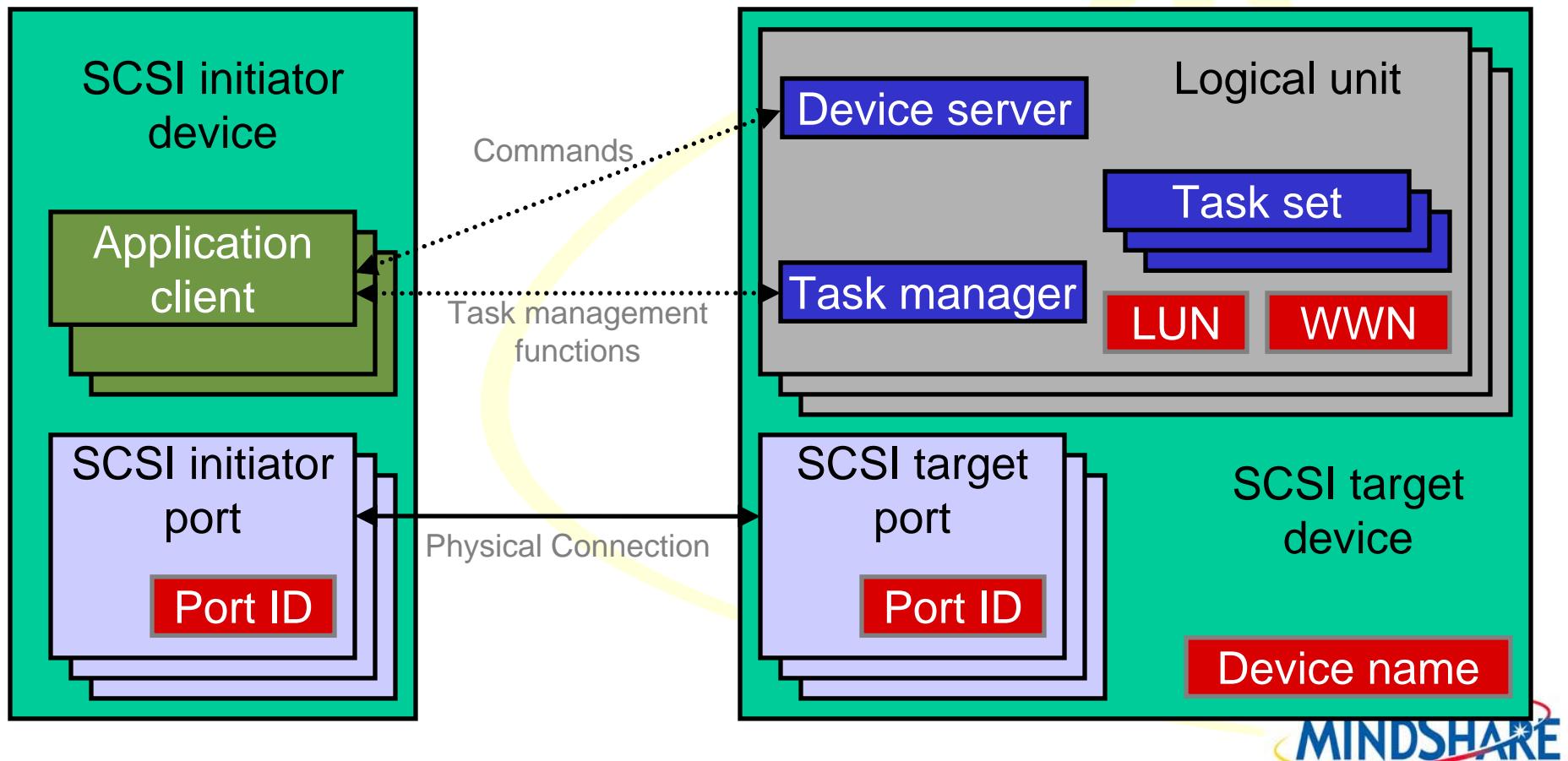
One Link

# Responsibilities

- Receive commands from Application Management Client (device driver)
- Pass requests to Transport layer using appropriate protocol. Examples:
  - “Send SCSI Command” becomes request for SSP COMMAND frame
  - “Discover” becomes request for SMP DISCOVER frame
- To better understand the types of requests the application layer will see, a review of SCSI application and transport layer functions follows -

# SCSI objects (195)

- Application clients communicate with device servers and task managers using SCSI initiator ports and SCSI target ports

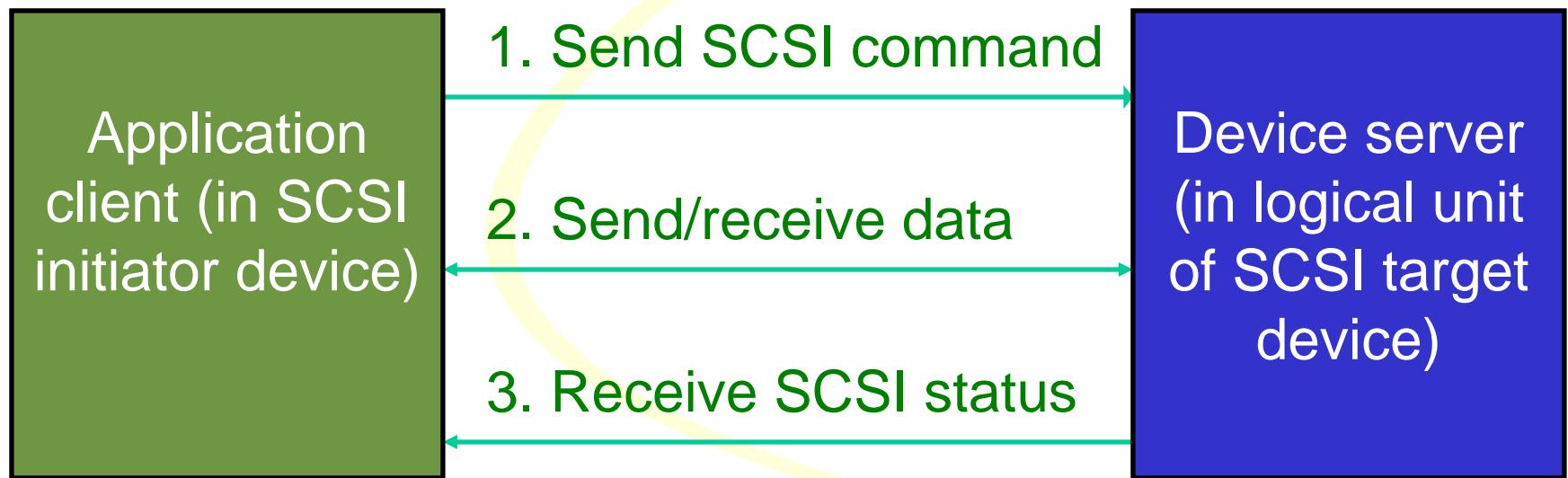


# SCSI Nomenclature

- Each SCSI initiator port and SCSI target port has a **SCSI port ID** that is unique to the domain
  - In SAS, this is the 64-bit SAS address.
- Each logical unit is assigned a **logical unit number (LUN)** within the SCSI target device
  - 64 bits represent this, subdivided into 4 “levels” of 16 bits each. Only LUN 0 is required

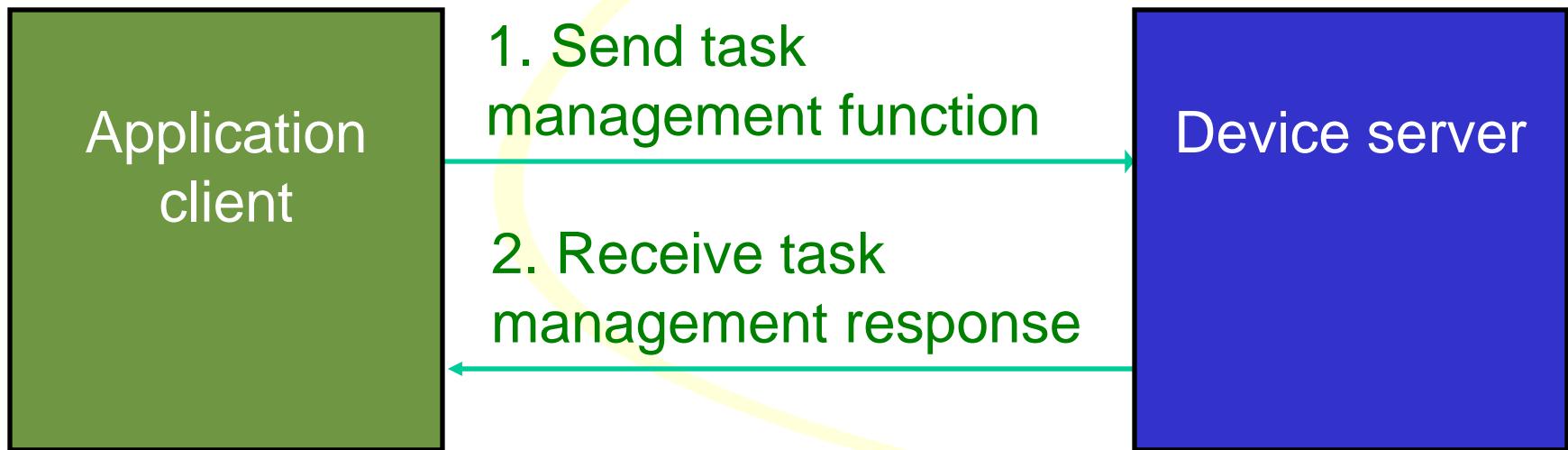
# SCSI command processing (196)

- Application client sends a SCSI command (task) to device server
- Device server transfers data, then returns SCSI status



# SCSI task management (197)

- Application client sends task management functions to device server
- Device server returns task management response
- No data transfer involved



# SCSI command sets (197)

- INQUIRY command returns the command sets a logical unit supports (Peripheral Device Type and some other fields)
- Commands are defined in command set standards

Standard	Name	Types of logical units that implement
SPC-3	Primary	All
SBC-3	Block	Disk drives
SSC-2	Streaming	Tape drives
SES-2	Enclosure Services	Enclosures (JBODs, external RAID)
MMC-5	Multimedia	CDs and DVDs
SMC-2	Media Changer	Tape libraries
SCC-2	Controller	RAID controllers (no known implementations follow this, but many do use its Peripheral Device Type)
OSD	Object	Object-based storage devices

# Required SCSI Commands (198)

- Four commands are required for all logical units:

Command	Type	Description
INQUIRY	Read	Returns information about the logical unit, such as Device Type (disk, tape, etc.), Vendor ID, Product ID, Serial Number, etc. Also returns <b>vital product data (VPD) pages</b> which contain Device Identifiers (worldwide names).
REPORT LUNS	Read	Returns a list of the logical unit numbers present in the SCSI target device (LUN 0 required)
REQUEST SENSE	Read	Returns logical unit's current sense data (This is actually obsolete now, since modern devices return the condition with the error indication.)
TEST UNIT READY	Non-data	Returns GOOD if the logical unit is ready to accept media-access commands

Four possible command types: Read, Write, Bi-directional, and Non-data

# SCSI Enclosure Services (198)

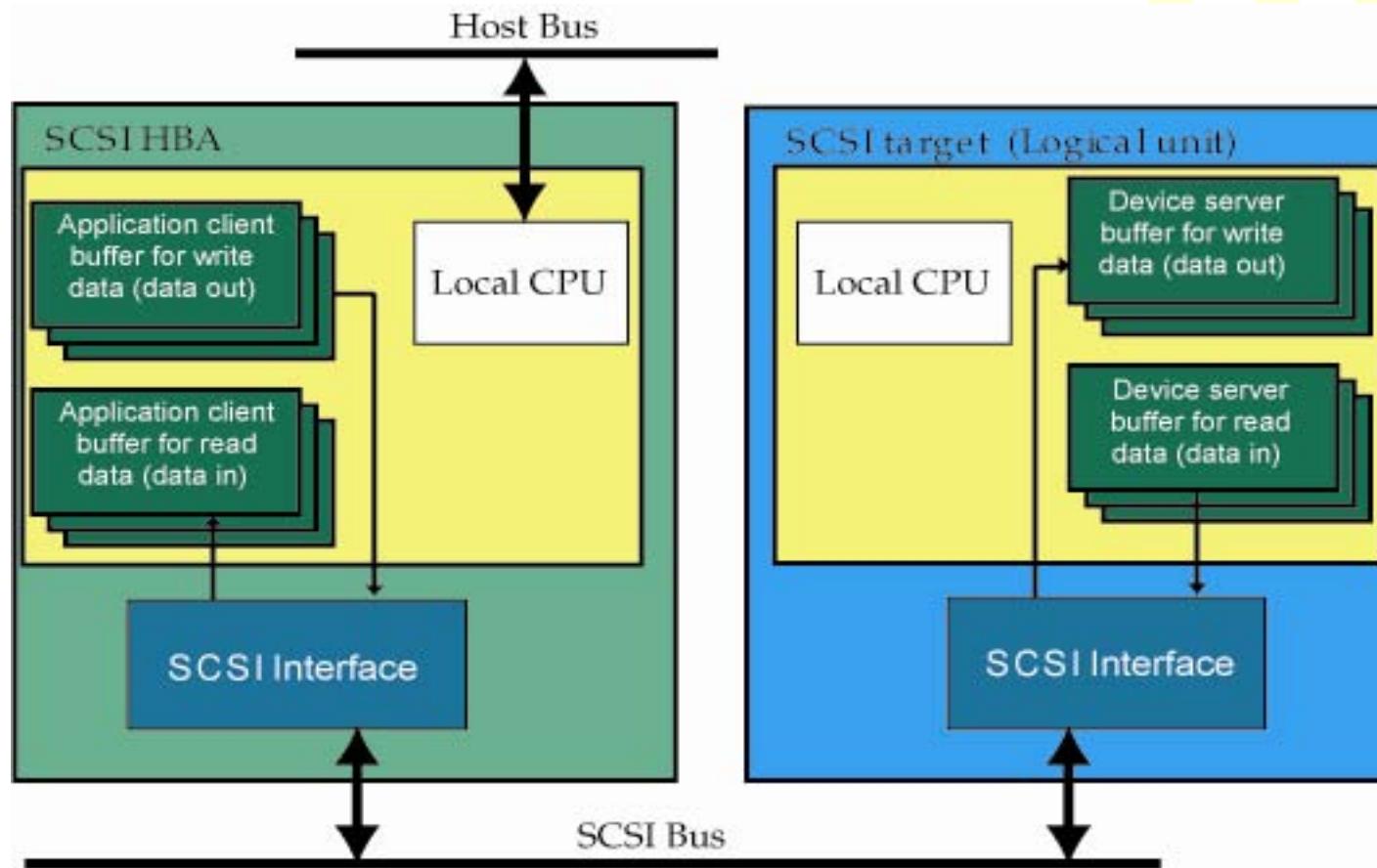
- Enclosures may include service processors to manage fans, temperature sensors, LEDs, etc.
- SES defines a series of diagnostic pages to communicate with the enclosure services processor over SCSI
  - SEND DIAGNOSTIC and RECEIVE DIAGNOSTIC RESULTS commands used to transmit the pages
  - Diagnostic page codes 00h-1Fh are defined by SES, regardless of the Peripheral Device Type reported in INQUIRY
- More discussion of SES and its predecessor, SAF-TE, in Appendix D of the textbook

# SCSI data transfers (198)

- Logical unit controls when data transfers occur
  - When application client sends a write command, it must be prepared to provide all the data at any time, but logical unit can wait as long as it wants to fetch write data or deliver read data
- Initiator does not “push” data to the target
  - Reduces target buffer requirements; no need to reserve space for unsolicited data
  - Exception: “first burst” data sometimes supported
    - More important for long-latency protocols like iSCSI than low-latency environments like SAS

# SCSI Data Transfer Buffers (200)

- Application client's buffers for write data and read data are separate from each other and from other commands



# SCSI status codes (201)

- When command is done, logical unit returns a Status byte
- Most common Status codes:

Value	Status	Description
00h	GOOD	The command completed successfully
02h	CHECK CONDITION	The command failed. Sense Data (a data structure of more than 8 bytes) is returned along with the Status indicating why.
08h	BUSY	Command refused - the logical unit is temporarily busy (unknown reason). Try again.
28h	TASK SET FULL	Command refused - the logical unit is busy handling other commands. Try again when a command completes (if that time can be determined).
18h	RESERVATION CONFLICT	Command refused - the logical unit is reserved by some other initiator port.

# SCSI sense keys (201)

- If CHECK CONDITION status is returned, **sense data** is sent explaining the reason
  - Includes a Sense Key field (4 bits) explaining the basic reason
  - Complete list in SPC-3, but some common Sense Keys are:

Value	Sense key	Description
0h	NO SENSE	No additional information
1h	RECOVERED ERROR	Command succeeded, but the logical unit recovered from an error to do so (e.g.: an abnormally high number of retries when reading from disk)
2h	NOT READY	Logical unit not ready for the command (e.g. not spinning when a READ was requested)
3h	MEDIUM ERROR	Non-recovered error (e.g. unrecoverable media area)
4h	HARDWARE ERROR	Non-recovered error (e.g. bad controller)
5h	ILLEGAL REQUEST	Problem with the requested command (e.g. bad field in CDB)
6h	UNIT ATTENTION	Command ignored; some other event happened that needs to be reported (e.g. a reset occurred)
Bh	ABORTED COMMAND	Command aborted for some reason

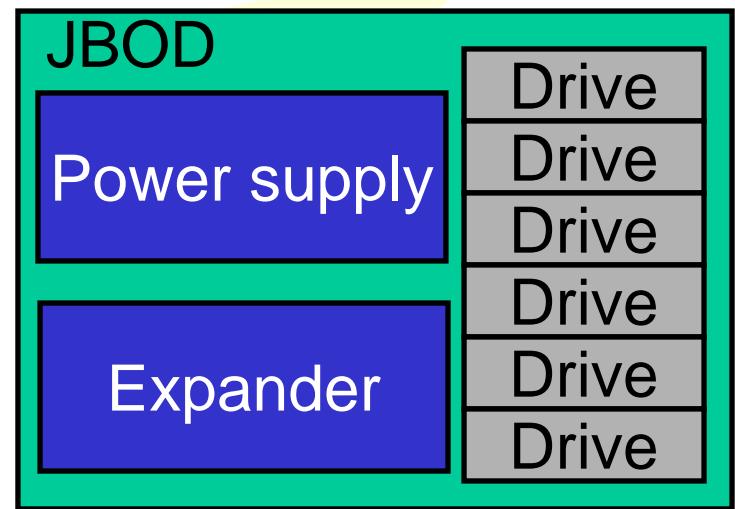
# SCSI Task Management (202)

- Task management functions are defined in SAM-3
  - Behavior defined, but how to request them is controlled by each transport protocol

Command	Scope	Description
QUERY TASK	I_T_L_Q	Check whether a specified command is still being processed. Used for NAK recovery, to see whether the command was received.
ABORT TASK	I_T_L_Q	Abort a specific command
ABORT TASK SET	I_T_L	Abort all commands from an initiator
CLEAR TASK SET	I_T_L	Abort all commands from all initiators
CLEAR ACA	I_T_L	Clear an Auto Contingent Allegiance condition
LOGICAL UNIT RESET	I_T_L	Reset the logical unit
I_T NEXUS RESET	I_T	Causes I_T Nexus loss in target. Logic behaves as if unable to OPEN initiator before nexus loss timer expired.

# SCSI Power Conditions

- Disk drives consume much more power when spinning up than while running
  - e.g. 20W while active, 50W peak during spinup
- If drives are allowed to spin up simultaneously, a larger power supply would be needed.



# Power Conditions and Spinup

- SATA drives may spin up automatically after phy reset
  - SATA II adds a solution for this – spin up is controlled by connector pin 11 (Ready Indicator)
    - Grounded = immediate spin up OK
    - High = postpone spin up (until system-specified event like reset sequence)
- SAS drives do not spin up automatically
  - They wait for a NOTIFY (ENABLE SPINUP) primitive

# Power Conditions

State	Description
Active	Fully operational – media is spinning
Idle	Operational - media is spinning. Longer command latency. Automatically transitions to Active as needed to process a command.
Standby	Media is stopped. Automatically transitions to Active or Idle as needed to process a command.
Stopped	Media is stopped. START STOP UNIT command required to restart.
Active_Wait	<b>New for SAS.</b> Waiting for a NOTIFY (ENABLE SPINUP) to enter Active state
Idle_Wait	<b>New for SAS.</b> Waiting for a NOTIFY (ENABLE SPINUP) to enter Idle state

# SMP Application Layer

- Management Protocol Services
  - Send SMP Function, Receive SMP Response
- Example SMP functions (List is found in SAS - 10.4.3)
  - Report General: return general information about the device: mode page info, phy info and addresses, etc.
  - Report Manufacturer Information: return vendor and product ID  
(Along this line, note that VPD data can be read from a target by using an INQUIRY command to read Device ID and VPD page 83h).

# SMP Functions

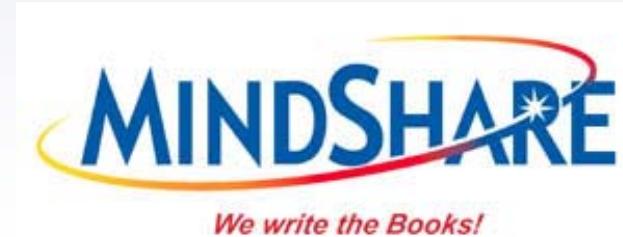
- SMP functions (continued)
  - Discover: return info about a specific phy – eg: address and device type for attached device
  - Report Phy Error Log
  - Report Phy SATA: returns info about SATA state of phy - affiliations, etc.
  - Report Route Information: read route table info
  - Configure Route Information: write table info
  - Phy Control: reset, update partial pathway timeout, program link rates

# Discovery Process

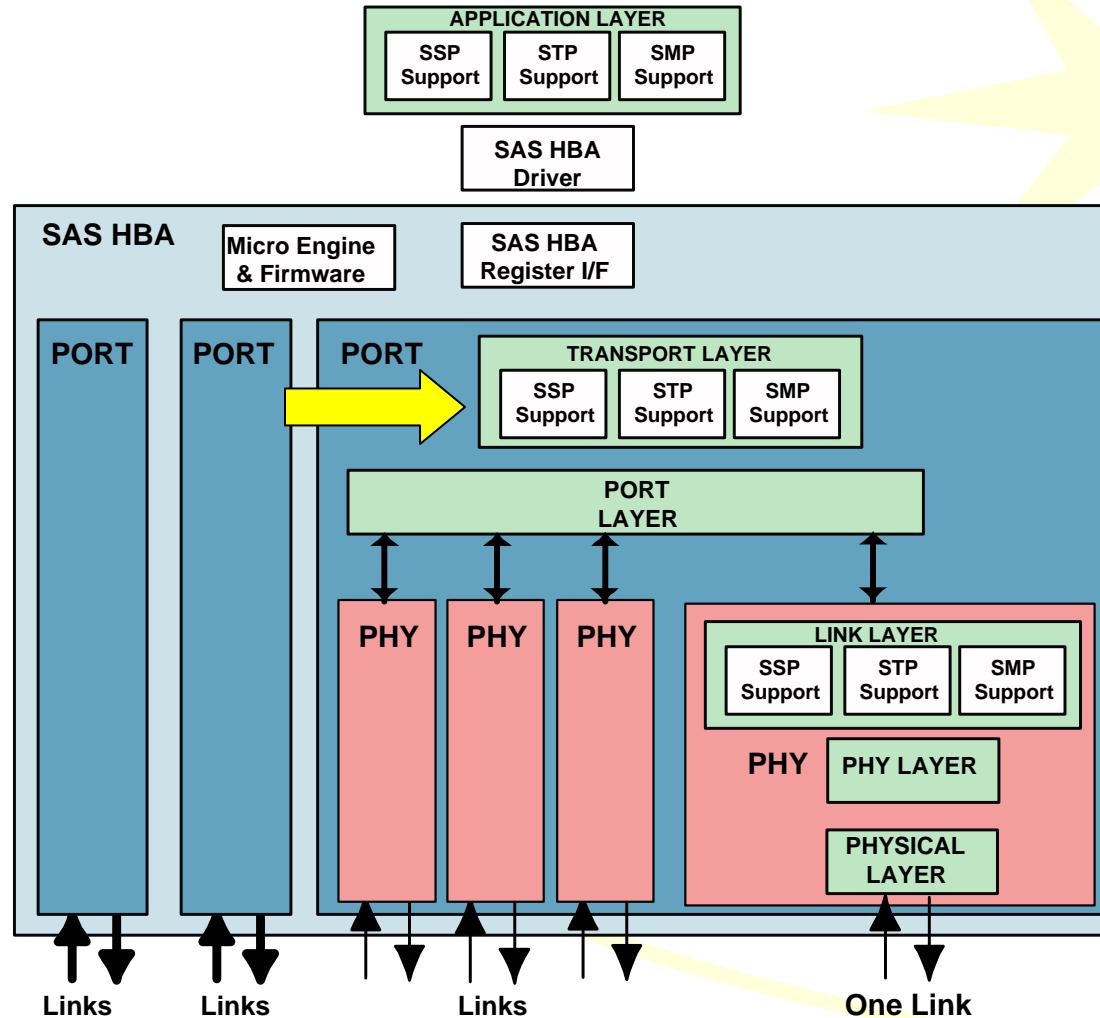
- SMP functions are used in the discovery process, giving an initiator visibility to the topology and allowing it to discover all the SAS addresses in its domain.
- The Discovery Process is discussed in detail in chapter 7 of the textbook

# *Chapter 9*

# *Transport Layer*



# Block Diagram of Layers (210)

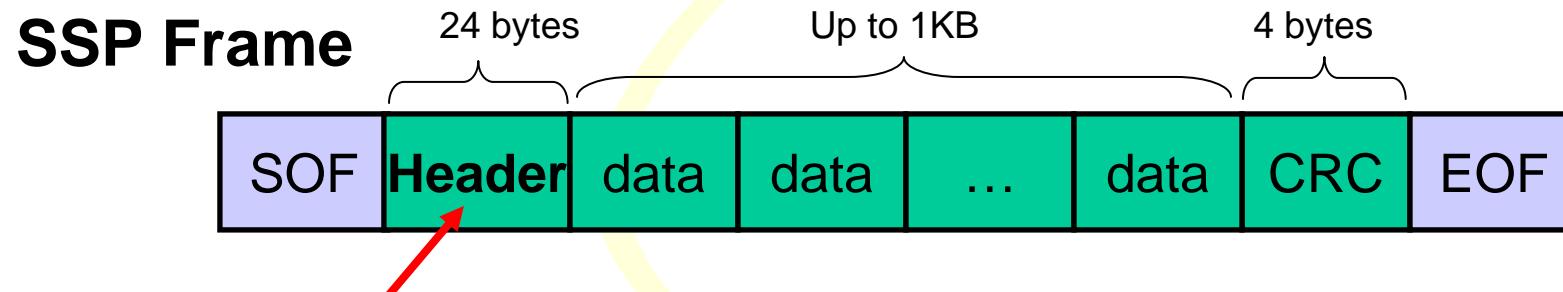


# Transport Layer Responsibilities

- Define and construct frames
- Interact with Application Layer and Port Layer – sending requests and getting confirmations
- Handle Link Layer errors

# SSP Support (211)

- Functions of transport layer are protocol specific.
- SSP frame header fields are described on the following slides



# SSP Frame Header – Type (213)

Byte	Field(s)				
0	Frame Type				
1 to 3	Hashed Destination SAS address				
4	Reserved				
5 to 7	Hashed Source SAS address				
8 to 9	Reserved				
10	Reserved	Retransmit	Rsvd		
11	Reserved	Number of Fill Bytes			
12 to 15	Reserved				
16 to 17	Tag				
18 to 19	Target Port Transfer Tag				
20 to 23	Data Offset				
24 to m	Information Unit (Corresponds to Frame Type)				
m to (n-3)	Fill bytes, if needed				
(n-3) to n	CRC				

## Frame Type Field

- 01h – DATA
- 05h – XFER\_RDY
- 06h – COMMAND
- 07h – RESPONSE
- 16h – TASK management

# Hashed Addresses

Byte	Field(s)				
0	Frame Type				
1 to 3	Hashed Destination SAS address				
4	Reserved				
5 to 7	Hashed Source SAS address				
8 to 9	Reserved				
10	Reserved	Retransmit	Rsvd		
11	Reserved	Number of Fill Bytes			
12 to 15	Reserved				
16 to 17	Tag				
18 to 19	Target Port Transfer Tag				
20 to 23	Data Offset				
24 to m	Information Unit (Corresponds to Frame Type)				
m to (n-3)	Fill bytes, if needed				
(n-3) to n	CRC				

Hashed Addresses can be used to double-check transmission accuracy, although some addresses will hash to the same value, making this imprecise.

Checking these fields is optional, and few devices do.

# Frame Attributes

Byte	Field(s)				
0	Frame Type				
1 to 3	Hashed Destination SAS address				
4	Reserved				
5 to 7	Hashed Source SAS address				
8 to 9	Reserved				
10	Reserved	Retransmit	Rsvd		
11	Reserved	Number of Fill Bytes			
12 to 15	Reserved				
16 to 17	Tag				
18 to 19	Target Port Transfer Tag				
20 to 23	Data Offset				
24 to m	Information Unit (Corresponds to Frame Type)				
m to (n-3)	Fill bytes, if needed				
(n-3) to n	CRC				

Indicates this frame is being retransmitted for some reason

Indicates whether bytes were added to make frame size evenly divisible by 4. Only DATA frames and RESPONSE frames will ever need this, the rest are automatically a multiple of four.

# Tag Information

Byte	Field(s)				
0	Frame Type				
1 to 3	Hashed Destination SAS address				
4	Reserved				
5 to 7	Hashed Source SAS address				
8 to 9	Reserved				
10	Reserved	Retransmit	Rsvd		
11	Reserved	Number of Fill Bytes			
12 to 15	Reserved				
16 to 17	Tag				
18 to 19	Target Port Transfer Tag				
20 to 23	Data Offset				
24 to m	Information Unit (Corresponds to Frame Type)				
m to (n-3)	Fill bytes, if needed				
(n-3) to n	CRC				



Maintains Initiator context for commands and task management functions.

Optional method for write data context that gives performance shortcut for target, with easier reference to buffer space for this transaction. (see following slides)

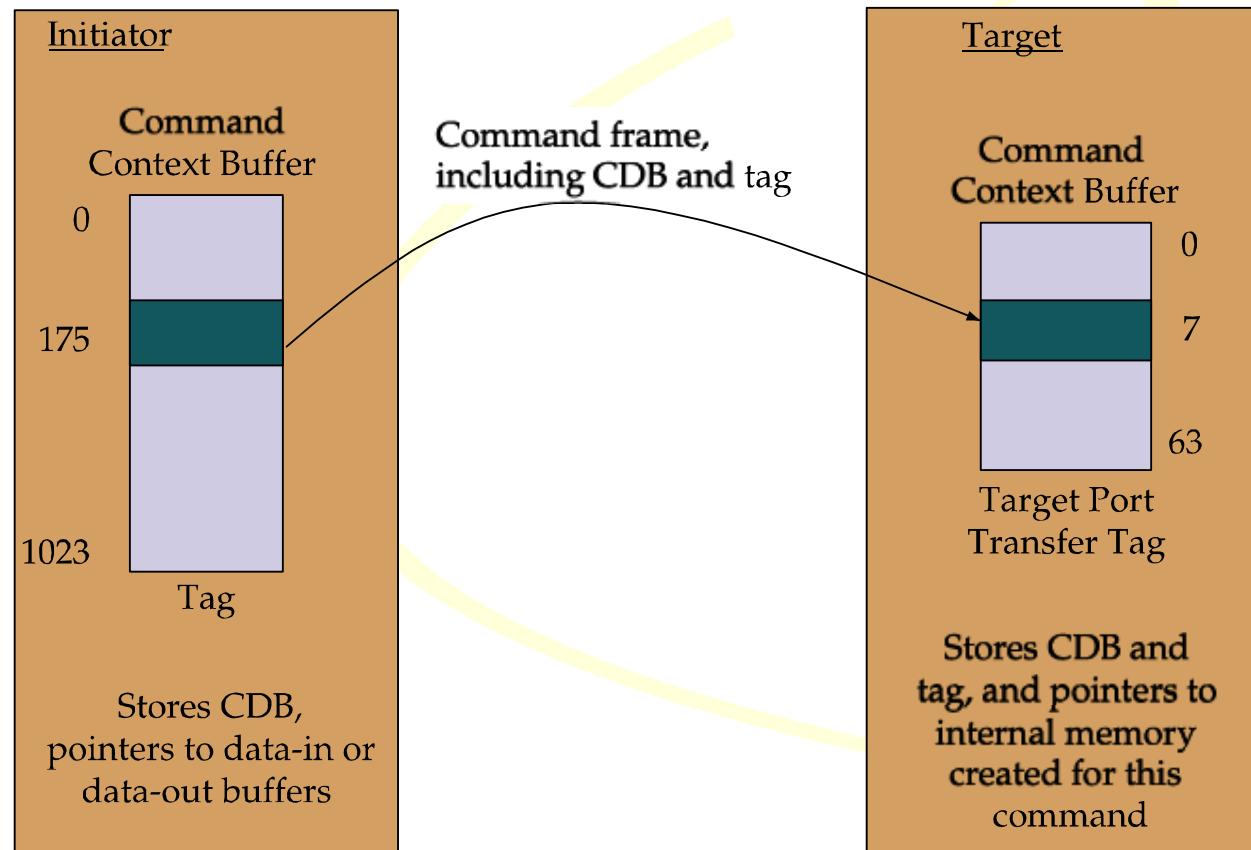
Application client buffer offset – set to zero for initial read or write data frame, incremented by transfer length in subsequent frames

# Target Port Transfer Tag

- Inherited from Fibre Channel
- Target may have several write commands in progress, each of which has an XFER\_RDY outstanding.
  - For each XFER\_RDY, target may have a DMA context ready to receive that data for storage on the disk.
  - Target must associate this context very quickly when the data begins to arrive, and it's much easier to do if there are only a small number of possible contexts. If target had to associate the Initiator's tag with its local context, it would take longer to look it up.

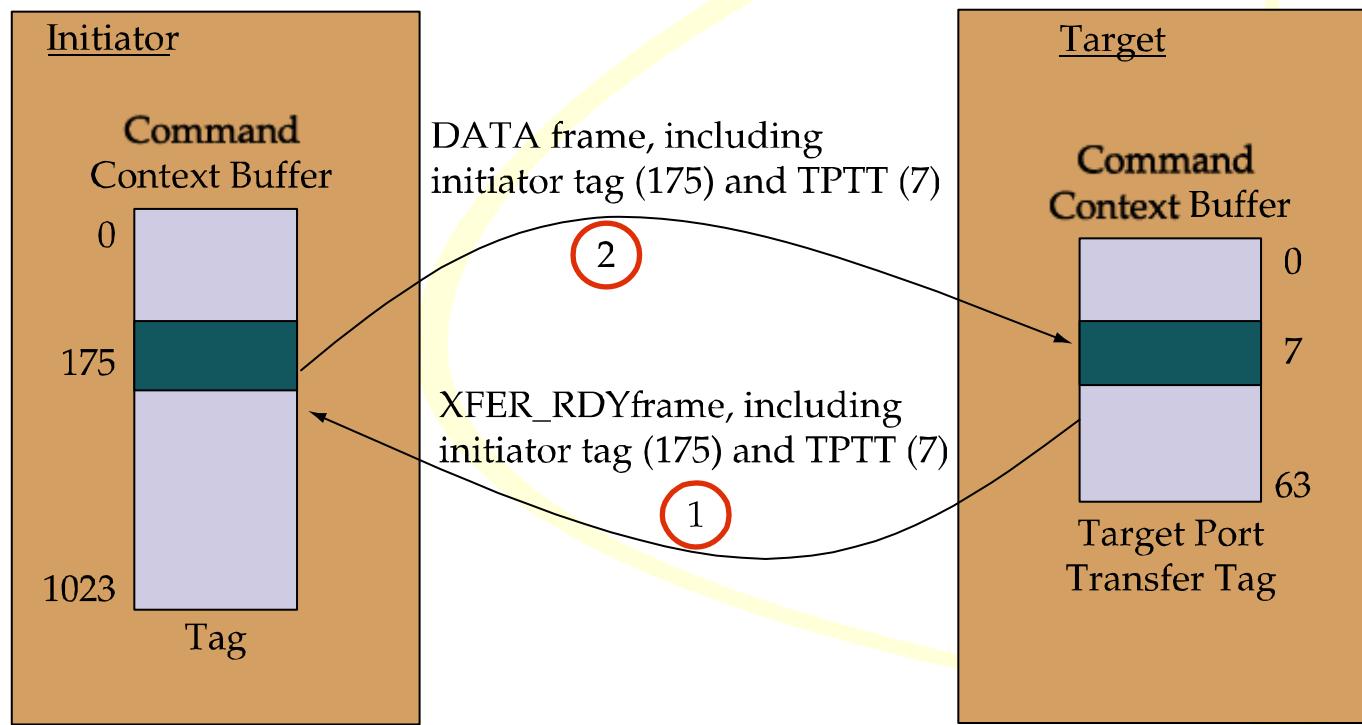
# Target Tag Example (217)

- Initiator stores command and pointers to the data that will be written with it in a Command Context buffer, then sends command
- Target sets up its own corresponding context registers



# Target Tag Example (218)

- When internal pointers are ready, target sends XFER\_RDY (for write command) to pull data, including initiator's original tag and its own TPTT.
- Initiator uses the tag to locate internal pointers and fetches the data to send as data frames.

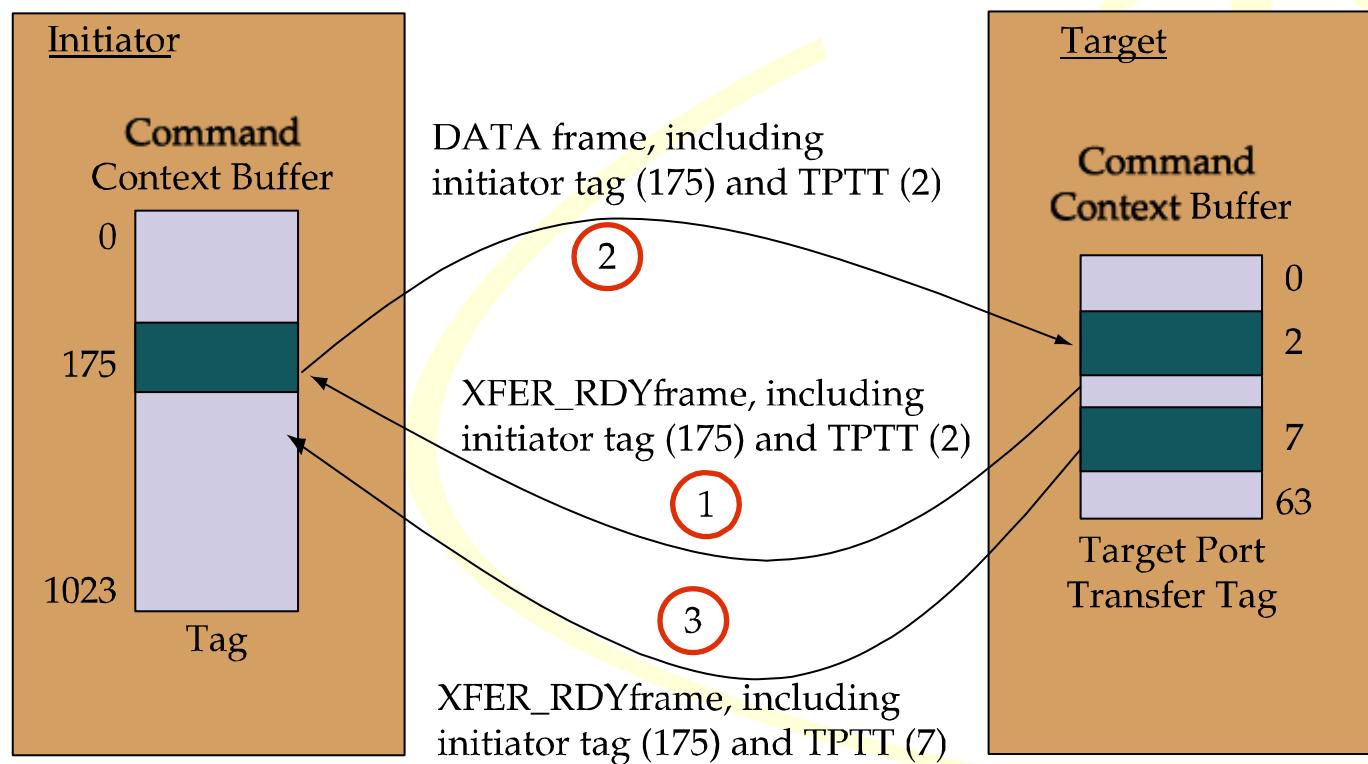


# Target Tag Use

- Target receives data frames and decides where to put them using the TPTT to find its pointers.
- Use of TPTT is optional, but when initiator sends the write data it must echo back whatever TPTT value the target sent.
- The TPTT can change for every XFER\_RDY, so initiator will need to memorize it each time (see next slides).

# Multiple Transfer Tags (219)

- Target can cycle through several buffer areas, resulting in a different TPTT with each XFER\_RDY



# Example of Changing Transfer Tag

- If multiple write commands are queued up, the first one might use TPTT 2 and finish the frame but not the transfer.
- A different write command starts, and target reuses TPTT 2 for that data because the space has become free.
- Target wants to resume first transfer, but now TPTT 2 is busy, so target selects TPTT 7 instead and sends that with XFER\_RDY. Result: same write command ends up using more than one TPTT.
- Conclusion: Initiator needs to store TPTT each time it receives an XFER\_RDY.

# Information Unit (220)

Byte	Field(s)				
0	Frame Type				
1 to 3	Hashed Destination SAS address				
4	Reserved				
5 to 7	Hashed Source SAS address				
8 to 9	Reserved				
10	Reserved	Retransmit	Rsvd		
11	Reserved	Number of Fill Bytes			
12 to 15	Reserved				
16 to 17	Tag				
18 to 19	Target Port Transfer Tag				
20 to 23	Data Offset				
24 to m	Information Unit (Corresponds to Frame Type)				
m to (n-3)	Fill bytes, if needed				
(n-3) to n	CRC				

**Information Unit** (corresponds to type)

For Data – data bytes

For COMMAND – LUN, Task attributes, CDB

For RESPONSE – status, sense data, response code

For XFER\_RDY – offset and data length

For TASK management – LUN, function, tag of task to be managed

# SCSI DATA IU (220)

- SAS allows from 1 to 1024 data bytes
- Fill bytes used to ensure size is always a multiple of Dwords

Byte	Field(s)
0 to n	Data
(0 to 3 bytes)	Fill bytes, if needed

# Command IU

	Bit 7	6	5	4	3	2	1	0						
Byte 0 to 7	Logical Unit Number													
8	Reserved													
9	Enable First Burst	Reserved		Task Attribute										
10	Reserved													
11	Additional CDB Length in dwords				Reserved									
12 to 27	CDB													
28 +	Additional CDB Bytes													

## Task Attribute

000b Simple Queue – target determines order

001b Head of Queue – put this command at top of queue

010b Ordered – initiator determines order

100b Auto Contingent Allegiance – not needed for SAS

# Response IU (222)

	Bit 7	6	5	4	3	2	1	0
Byte 0 to 9		Reserved						
10		Reserved		DataPres				
11		Status						
12 to 15		Reserved						
16 to 19		Sense Data Length (bytes)						
20 to 23		Response Data Length (bytes)						
24 +		Response Data or Sense Data						

DataPres  
 00 – No sense or response data  
 01 – Response data included  
 10 – Sense data included  
 11 – Reserved: can't do both  
 (See next slide)  
 Only one or the other can be used at a time

# Status Codes (223)

List defined in SAM-3

Status Code	Status
00h	Good
02h	Check Condition
04h	Condition met
08h	Busy
10h	Intermediate
14h	Intermediate, Condition met
18h	Reservation Conflict
22h	Obsolete
28h	Task Set Full
30h	ACA Active
40h	Task Aborted

# Sense Data

- Sense Data Length and Sense Data fields
  - Only present when Status is CHECK CONDITION
  - Format defined in SPC-3
  - Most common fixed-length format: 18 bytes
  - Includes Sense Key and Additional Sense Code fields
    - Example: attempt to send linked commands to logical unit that doesn't support them.  
Result – Status: check condition; Sense Key: Illegal request; Add'l Sense Code: Invalid field in CDB

# Response Codes (224)

- When DataPres field indicates Response data, it is encoded as shown:

Status Code	Status
00h	Task Management Function Complete
02h	Invalid Frame
04h	Task Management Function Not Supported
05h	Task Management Function Failed
08h	Task Management Function Succeeded
09h	Invalid Logical Unit Number

Response to Query Task if task does not exist.

# XFER\_RDY IU

Byte	Field(s)
0 to 3	Requested Offset
4 to 7	Write Data Length
8 to 11	Reserved

- Offset – distance from the beginning of the buffer from which data is to be fetched. Multiple of 4, starting at zero for first XFER\_RDY. (If First Burst was used, the initial offset might not be one. Use of First Burst is discouraged in SAS.)
- Length – bytes, in multiples of 4, which the target is prepared to receive; initiator must not send more. (The last XFER\_RDY for a command can be a non-dword length.)

# TASK Management IU (225)

Byte	Field(s)
(24 bytes)	SSP frame header
0 to 7	Logical Unit Number
8 to 9	Reserved
10	Task Management Function
11	Reserved
12 to 13	Tag of Task to be Managed
14 to 27	Reserved
(0 bytes)	Fill bytes NOT needed
(4 bytes)	CRC

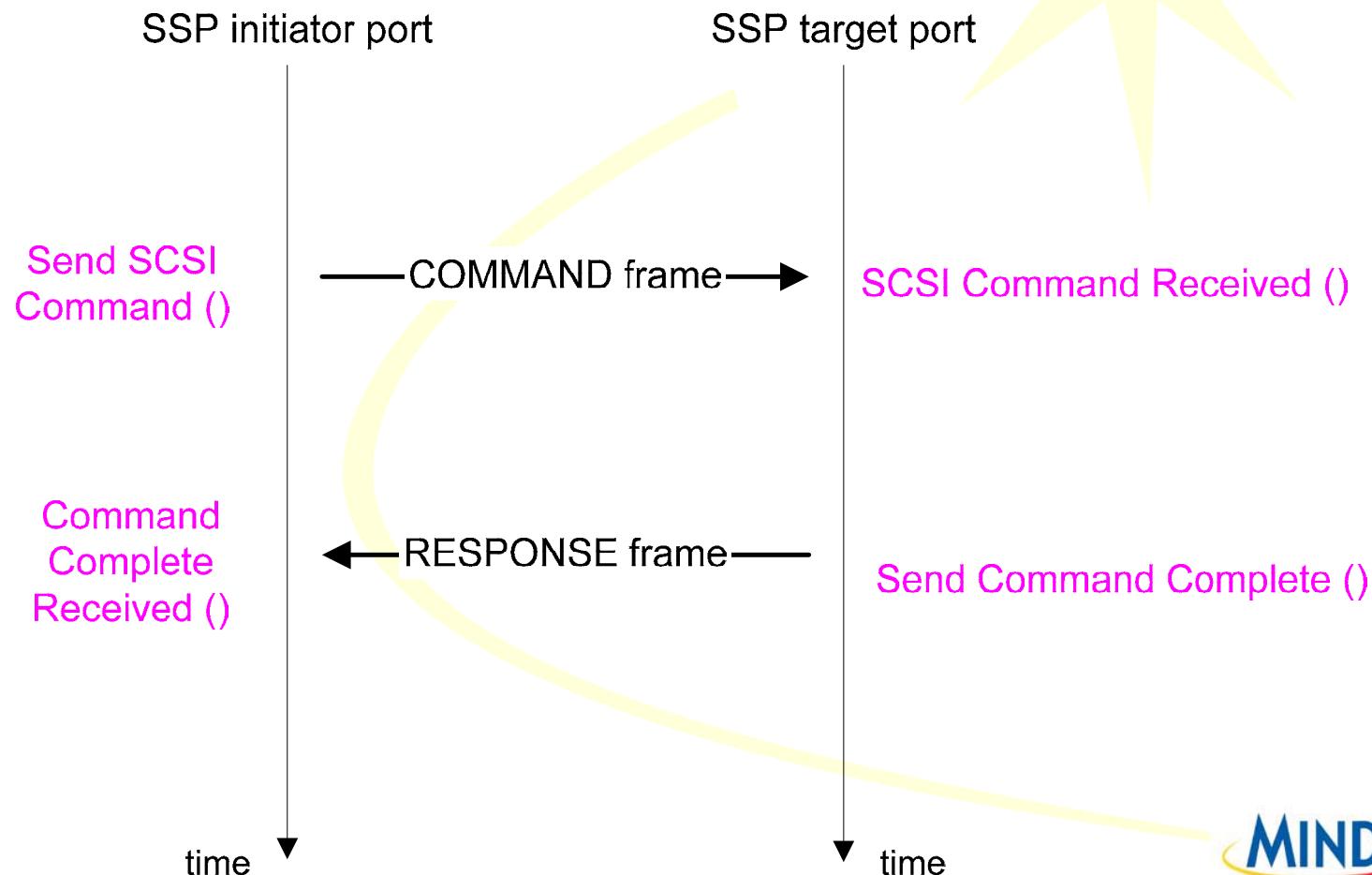
	Task Management Function
01h	Abort Task
02h	Abort Task Set
04h	Clear Task Set
08h	Logical Unit Reset
40h	Clear ACA
80h	Query Task

- Used to send task management functions from initiator to target
- Fixed frame length:  $24+28+4=56$  bytes
- Tag field only used for Abort Task and Query Task

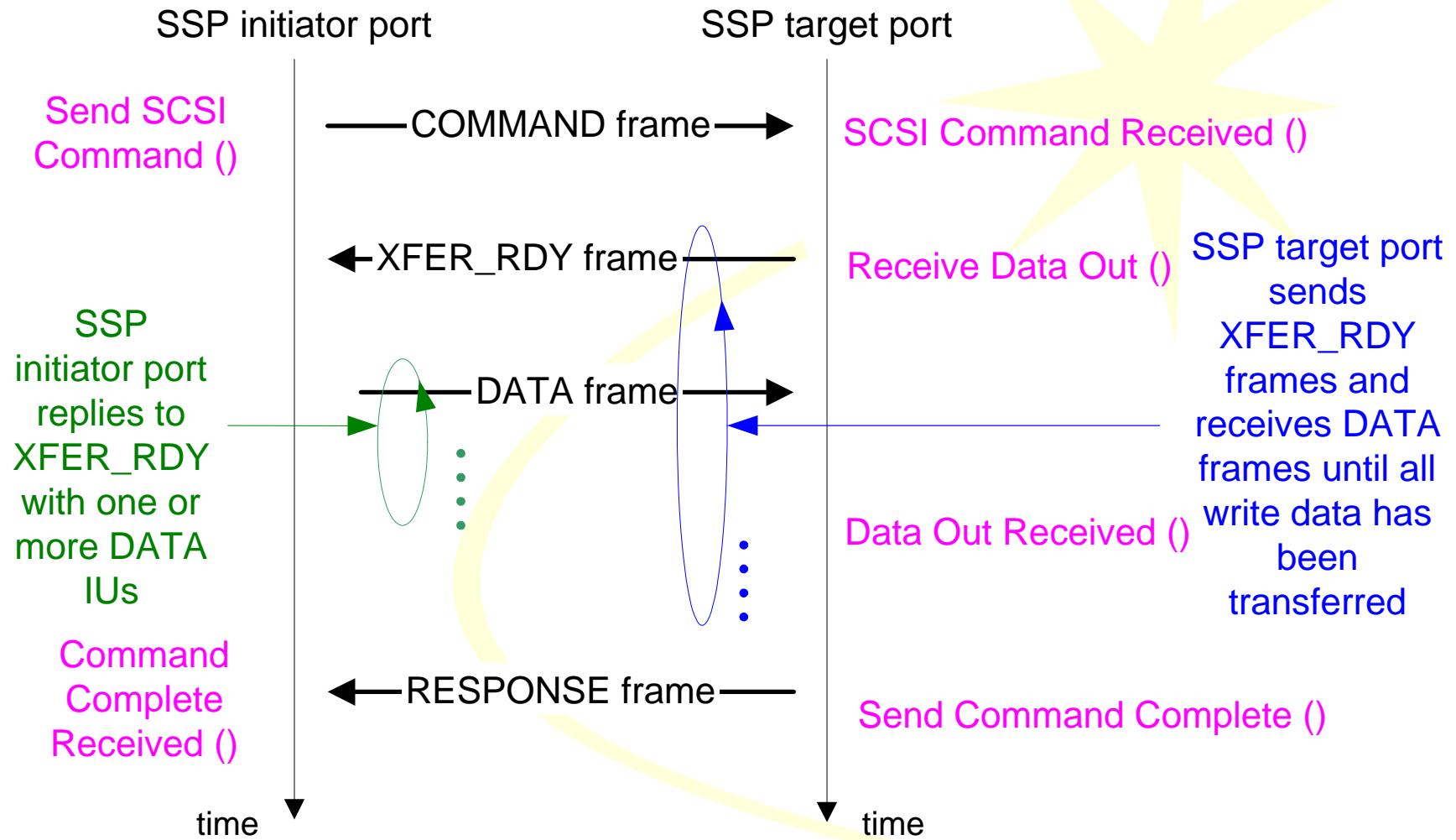
# Transport Examples: (227)

## SSP non-data command sequence

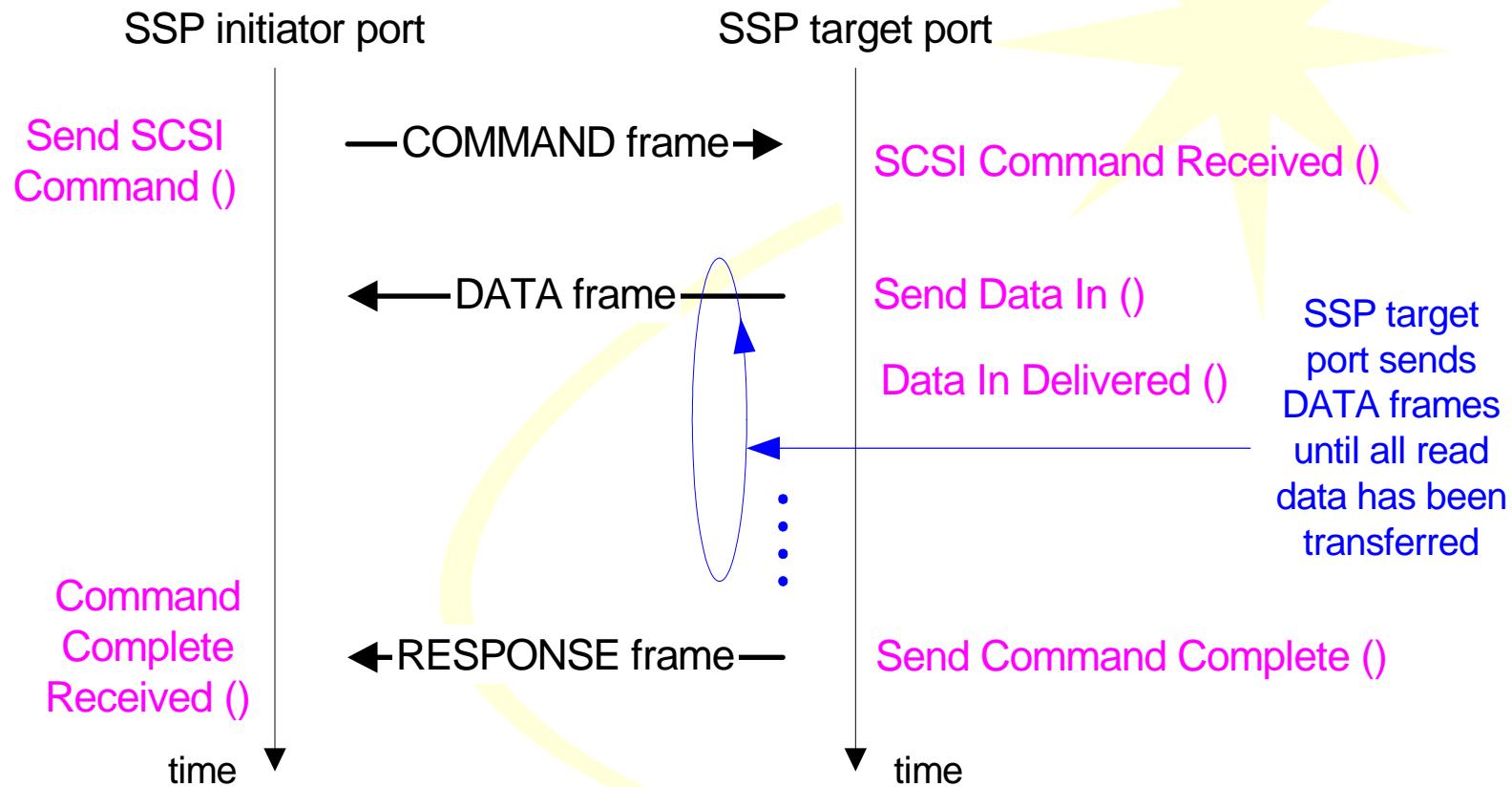
Examples do not include ACK/NAK or other Link layer activity



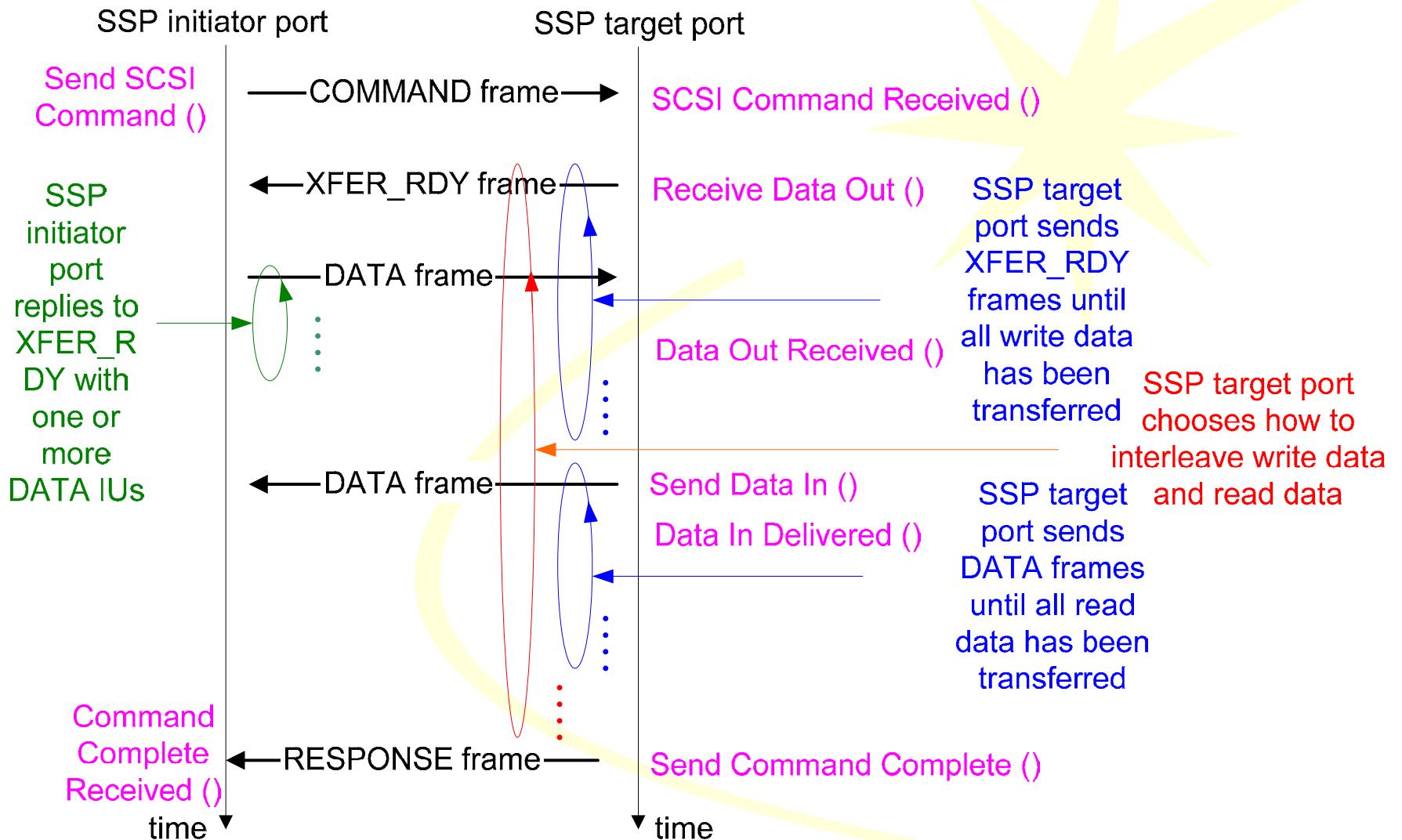
# SSP write command sequence (229)



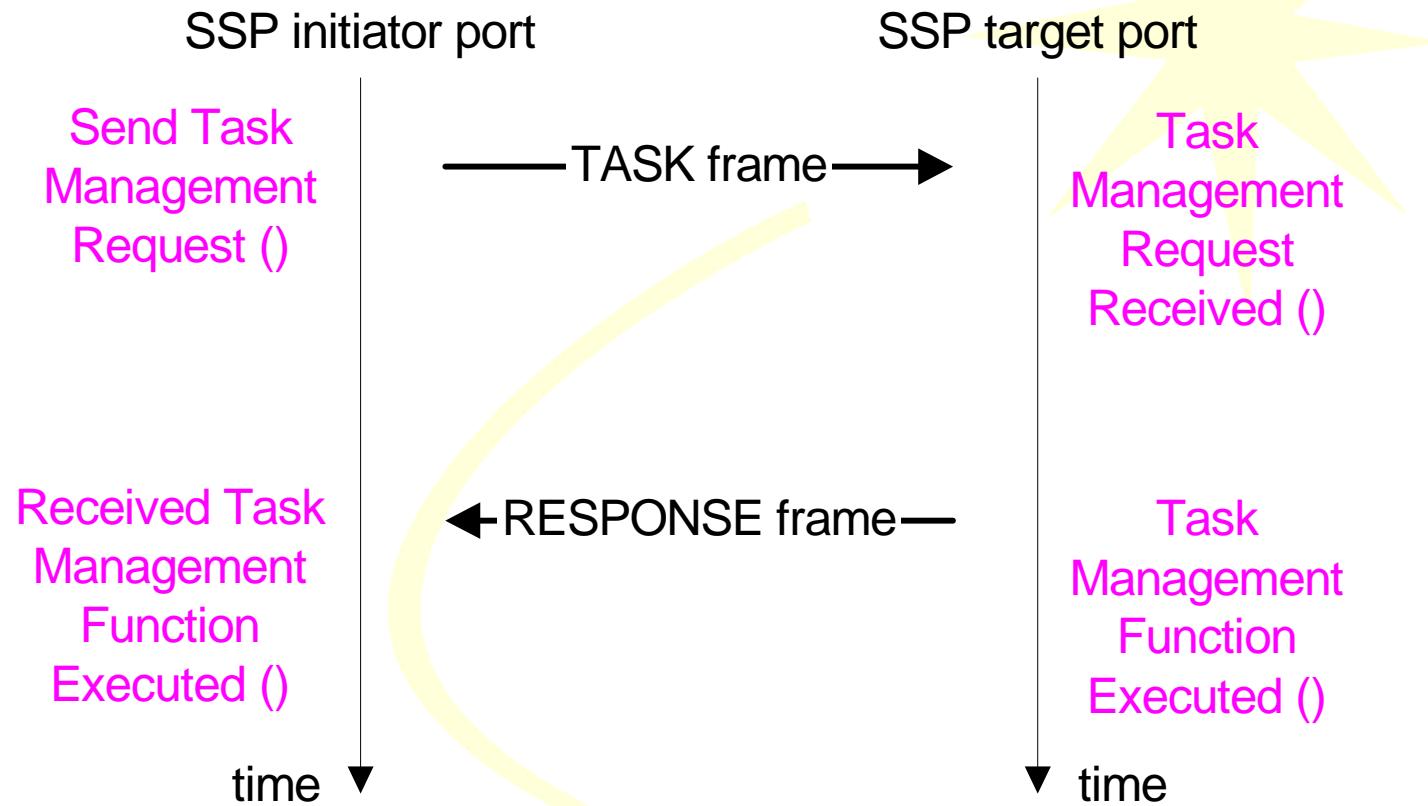
# SSP read example (231)



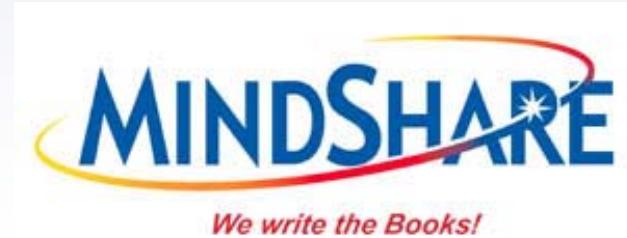
# SSP bidirectional sequence (232)



# SSP task management sequence (233)



# *SSP Error Handling*

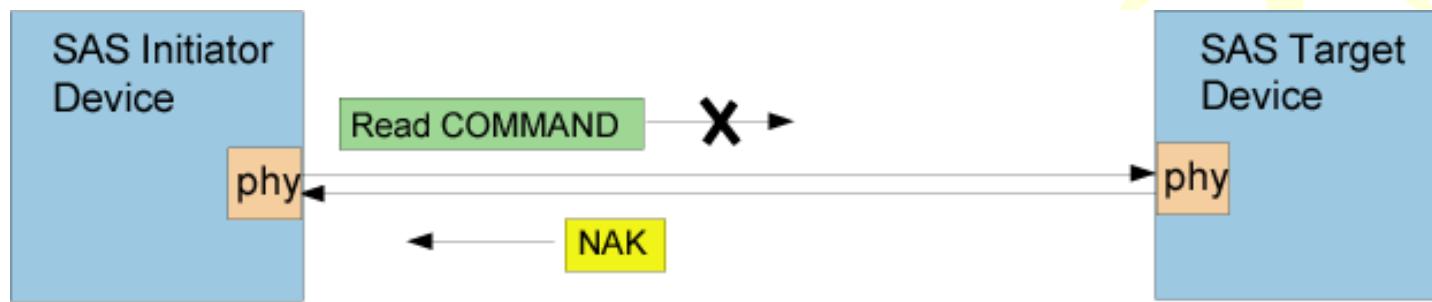


# SSP Frame Types

- Error handling is defined for each of the 5 basic transactions:
  - Command: initiating a transaction
  - Response: providing target feedback
  - Task Management
  - XFER\_RDY: reporting available buffer space
  - Data: Read or Write

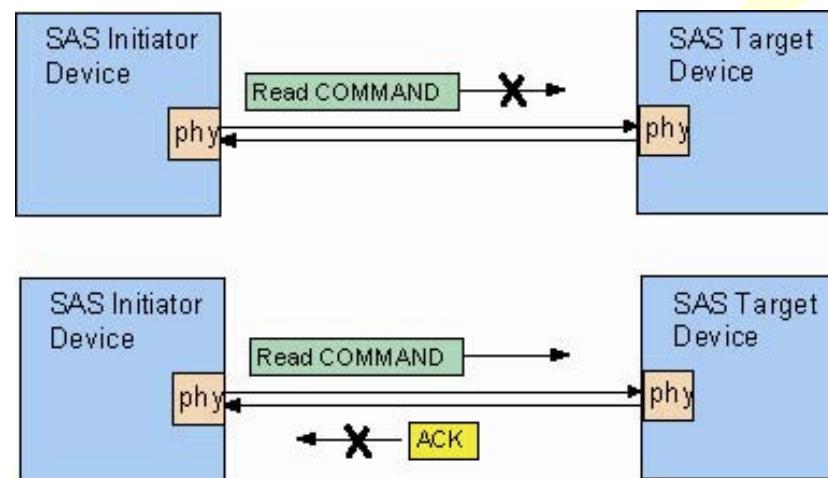
## Link Errors – Command Frame (235)

- If Initiator sees NAK, simply repeat command



# Link Errors – Command Frame

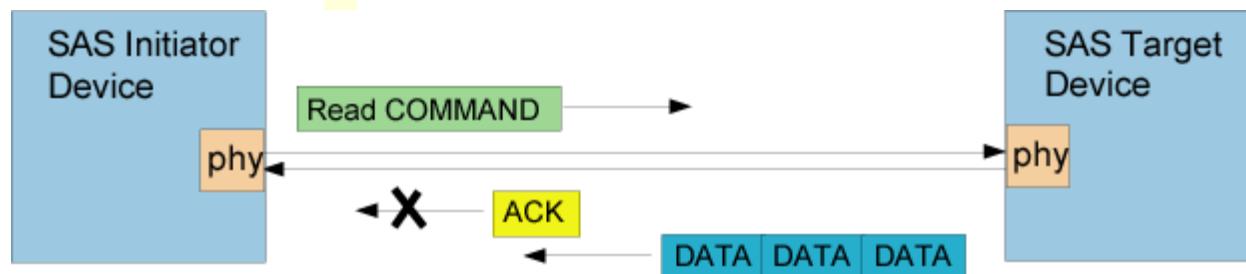
- If Initiator sees ACK/NAK timeout (1ms), it's unclear whether or not target received the command



- Action:
  - Close connection with **DONE (ACK/NAK TIMEOUT)**  
Note: this timer will not be reinitialized by incoming frames - error needs to be handled before accepting new frames. DONE timeout will cause BREAK.
  - Send **QUERY TASK** to see if command was received

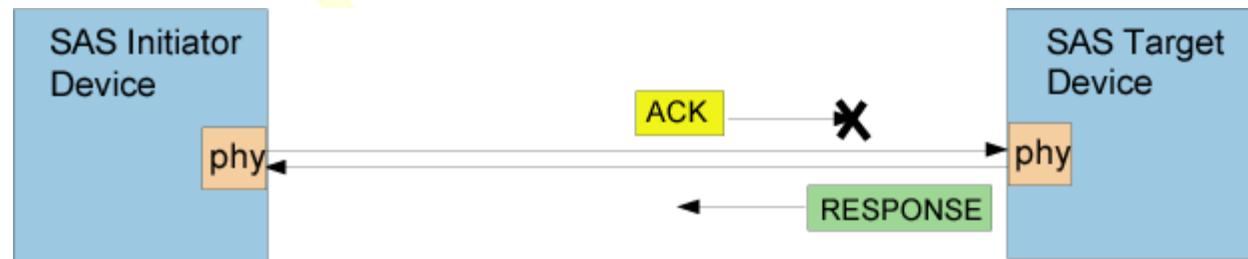
# Recovery from missing ACK

- If no ACK, but valid response arrives before timeout anyway, then command can be inferred to have been received OK.
  - Examples:
    - XFER\_RDY returned for next part of write command
    - DATA returned for read command
    - RESPONSE frame returned for a Query Task sent after the command that failed to see ACK/NAK



# Link Errors – Response Frame (237)

- If Target port sends Response frame and gets ACK/NAK timeout:
  - Close the connection with DONE (ACK/NAK TIMEOUT)
  - Open a new connection, send RESPONSE frame using same tag, but with RETRANSMIT bit set
- If NAK received
  - Simply resend the response at least once without setting RETRANSMIT bit

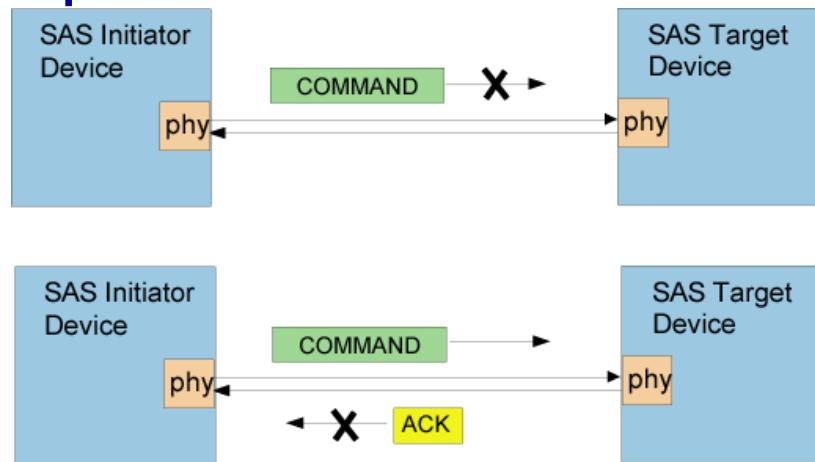


# Link Errors – Response Frame

- Initiator port receives a Response frame with the RETRANSMIT bit set
  - If already received a Response for this I\_T\_L\_Q nexus, the new one is discarded
  - If no previous Response received, accept this one as the valid Response

# Link Errors – Task Frame (238)

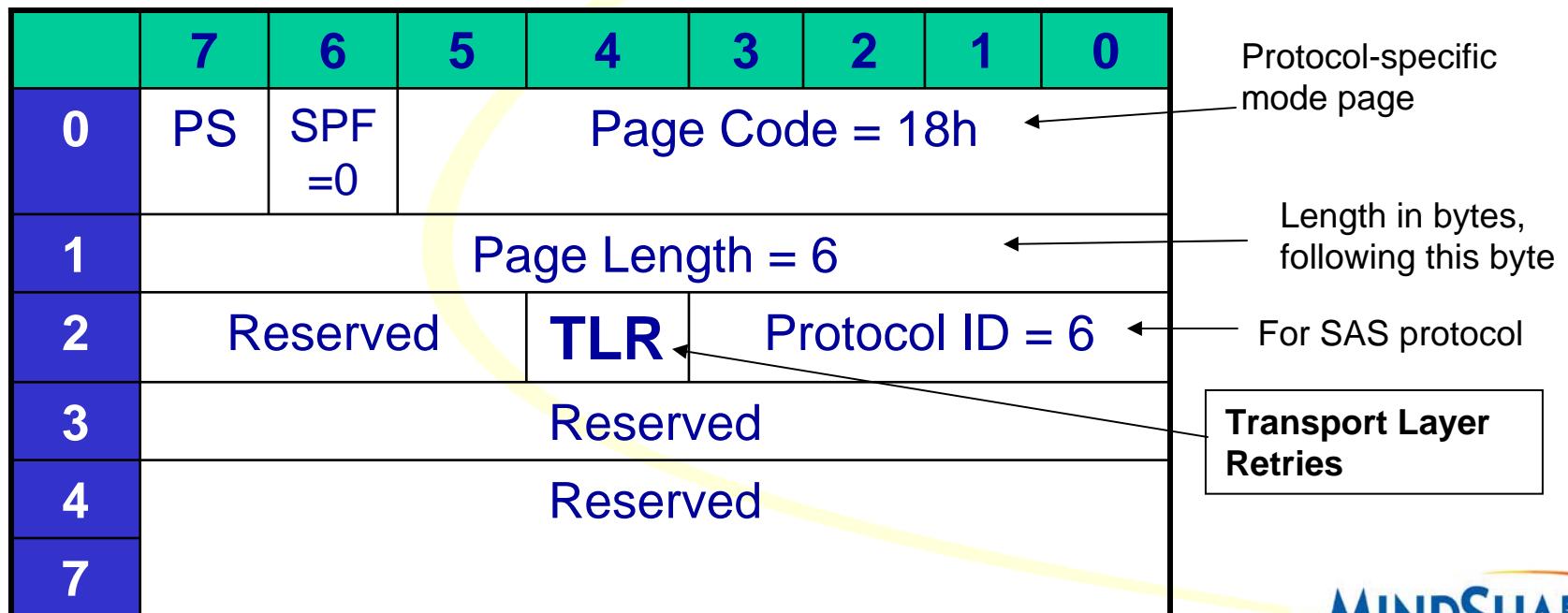
- Initiator port sees ACK/NAK timeout:



- Close the connection with **DONE (ACK/NAK TIMEOUT)**
- Retransmit, in a new connection, the **TASK** frame using same tag and with RETRANSMIT bit set
- If no ACK, but RESPONSE frame seen before timeout, task was completed

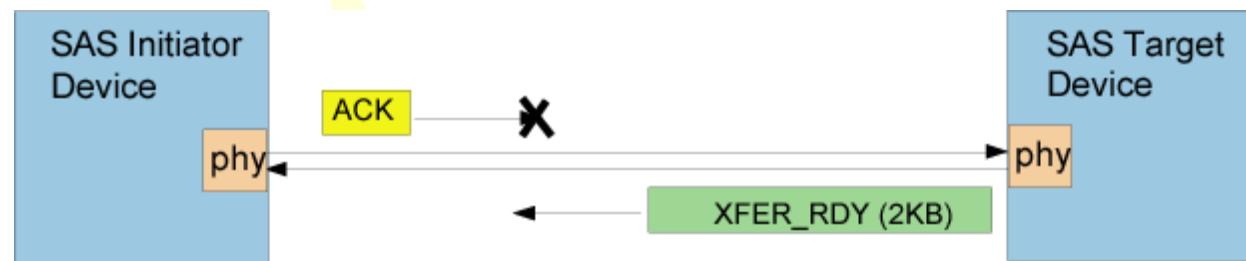
# Frame Retry Policy (239)

- Retry policy, for frames using XFER\_RDY or DATA, is controlled by the setting of the TLR bit in the Protocol-Specific Logical Unit Mode page.
- Retries support streaming devices, like tape drives, that would have to abort a backup job or do a tape rewind if an error required software attention to resolve. Hard drives don't implement this because they don't have that time constraint on resolving errors.



## Error - XFER\_RDY with Retries (240)

- If Target port sees ACK/NAK timeout:
  - Close the connection with DONE (ACK/NAK TIMEOUT),
  - Retransmit, in a new connection, the XFER\_RDY frame using a different Target Port Transfer Tag and with RETRANSMIT bit set



# Error - XFER\_RDY with Retries

- If NAK received:
  - Simply retransmit the XFER\_RDY frame using a different Target Port Transfer Tag and with RETRANSMIT bit set

## Error - XFER\_RDY with Retries

- If Initiator sees a duplicate XFER\_RDY with the RETRANSMIT bit set while servicing the previous XFER\_RDY
  - Stop sending write data frames for the previous one and start servicing the new one, since Target must have had trouble with first operation and is probably discarding all that data as it arrives anyway.

# Error - XFER\_RDY w/o Retries

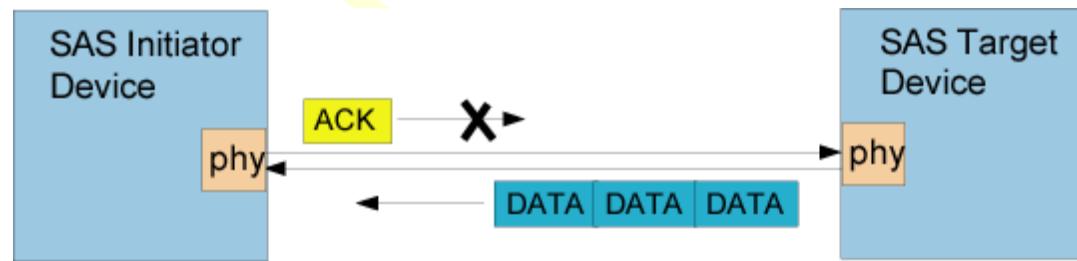
- If Target port sees ACK/NAK timeout:
  - Close the connection with DONE (ACK/NAK TIMEOUT)
  - Transmit, in a new connection, a RESPONSE frame returning CHECK CONDITION, sense key of ABORTED COMMAND, additional sense code ACK/NAK TIMEOUT
- If NAK received
  - Transmit a RESPONSE frame returning CHECK CONDITION, sense key of ABORTED COMMAND, additional sense code ACK/NAK TIMEOUT

# DATA Frames

- All SAS transactions are required to wait for an ACK or NAK before proceeding, except DATA frames, which can be queued up
  - Data frames can be sent without waiting for ACK if receiver has indicated room to take them (XFER\_RDY) and given permission for several frames (RRDYs)
  - ACK/NAKs do not contain transaction information, so sender must count them to verify that all frames were received
  - Count increments when frames are sent and decrements when ACKs received. Count is balanced when number sent and number ACKed are the same – referred to as ACK/NAK Balance.

## Error – Read DATA with Retries (242)

- If Target port sees ACK/NAK timeout for a Read data frame:
  - Close the connection with DONE (ACK/NAK TIMEOUT)
  - Retransmit, in a new connection, all the Data frames since the last time ACK/NAK Balance occurred
- If NAK received:
  - Retransmit all the Data frames since the last time ACK/NAK balance occurred



# Error – Read DATA w/o Retries

- If Target port sees ACK/NAK timeout for a Read data frame:
  - Close the connection with DONE (ACK/NAK TIMEOUT)
  - Transmit, in a new connection, a RESPONSE frame returning CHECK CONDITION, sense key of ABORTED COMMAND, additional sense code ACK/NAK TIMEOUT
- If NAK received:
  - Transmit a RESPONSE frame returning CHECK CONDITION, sense key of ABORTED COMMAND, additional sense code ACK/NAK TIMEOUT

# Error – Write DATA with Retries

- If Initiator port sees ACK/NAK timeout for a Write data frame:
  - Close the connection with DONE (ACK/NAK TIMEOUT)
  - Retransmit, in a new connection, all the Data frames for the previous XFER\_RDY
- If a new XFER\_RDY or RESPONSE for the command arrives, stop retransmission and service the new frame
- If NAK received:
  - Retransmit all the Data frames for the previous XFER\_RDY

# Error – Write DATA w/o Retries

- If Initiator port sees ACK/NAK timeout for a Write data frame:
  - Close the connection with DONE (ACK/NAK TIMEOUT)
  - Abort the command with Abort task frame
- If NAK received:
  - Abort the command with Abort task frame, possibly in the same connection

# Rules on Data Retries (243)

- For both reads and writes, the first retransmitted Data frame has its **CHANGING DATA POINTER** bit set, and subsequent frames have it cleared.
  - Note that changing the data pointer means target will need to generate an interrupt to get software to reset the scatter/gather list.
- If retries are enabled, each unacknowledged DATA frame gets retried at least once, and vendors can choose to do more than that.

# Transport Layer Errors

- If Target Port receives bad frames, such as:
  - XFER\_RDY or unknown frame type
    - target should not receive these, they only go from target to initiator - discard frame
  - COMMAND
    - If frame contains no LUN or CDB, or Additional Length field indicates wrong length - send Response with Invalid Frame code
  - TASK frame too short – send Response with Invalid Frame code
  - Command frame with Tag that's already in use – send Check Condition status with “Overlapped Commands Detected”

# Transport Layer Errors

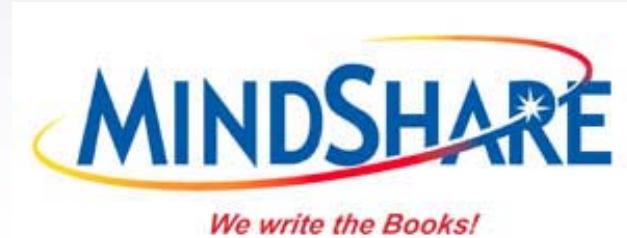
- If Target Port receives:
  - DATA frame with unknown Tag - discard
  - DATA frame without First Burst and no XFER\_RDY outstanding – discard
- Note that some devices can function as both target and initiator (eg: HBA), but they can only do one or the other within a connection – not both at the same time.

# Transport Layer Errors

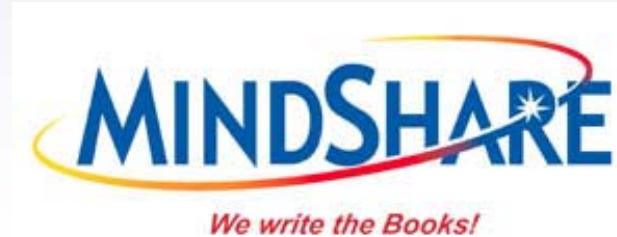
- If Initiator port receives:
  - Command or Task or unsupported frame type – discard the frame
  - Data with an unknown Tag (eg: a command that hasn't yet been ACKed) – discard the frame and possibly abort the command with that tag
    - Target is sending data to wrong initiator or based on obsolete commands. Send Task frame with ABORT TASK specifying the bad tag, or ABORT TASK SET, CLEAR TASK SET, or LOGICAL UNIT RESET (increasing levels of severity).
  - XFER\_RDY that isn't 12 bytes long – discard frame, abort the command

# *Transport Layer – STP*

*(See SATA Support chapter)*



# *Transport Layer - SMP*



# Construct or Parse SMP frames

- SMP Request frame - initiator to target

Byte	Field(s)
0	SMP Frame Type (40h)
1 to n	Request bytes
m to (n-3)	Fill bytes, if needed
(n-3) to n	CRC

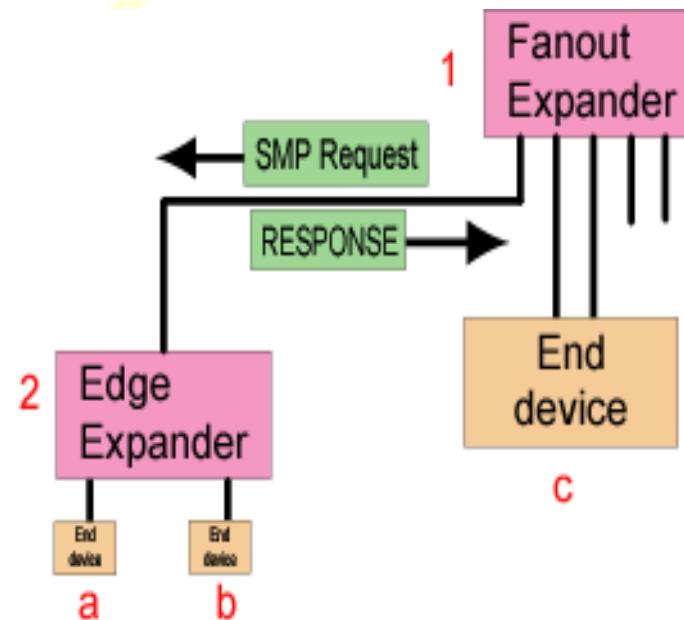
- SMP Response frame - target to initiator

Byte	Field(s)
0	SMP Frame Type (41h)
1 to n	Response bytes
m to (n-3)	Fill bytes, if needed
(n-3) to n	CRC

- Like SSP, Link layer checks CRC. If incorrect:
  - Target: BREAK the connection
  - Initiator: CLOSE the connection

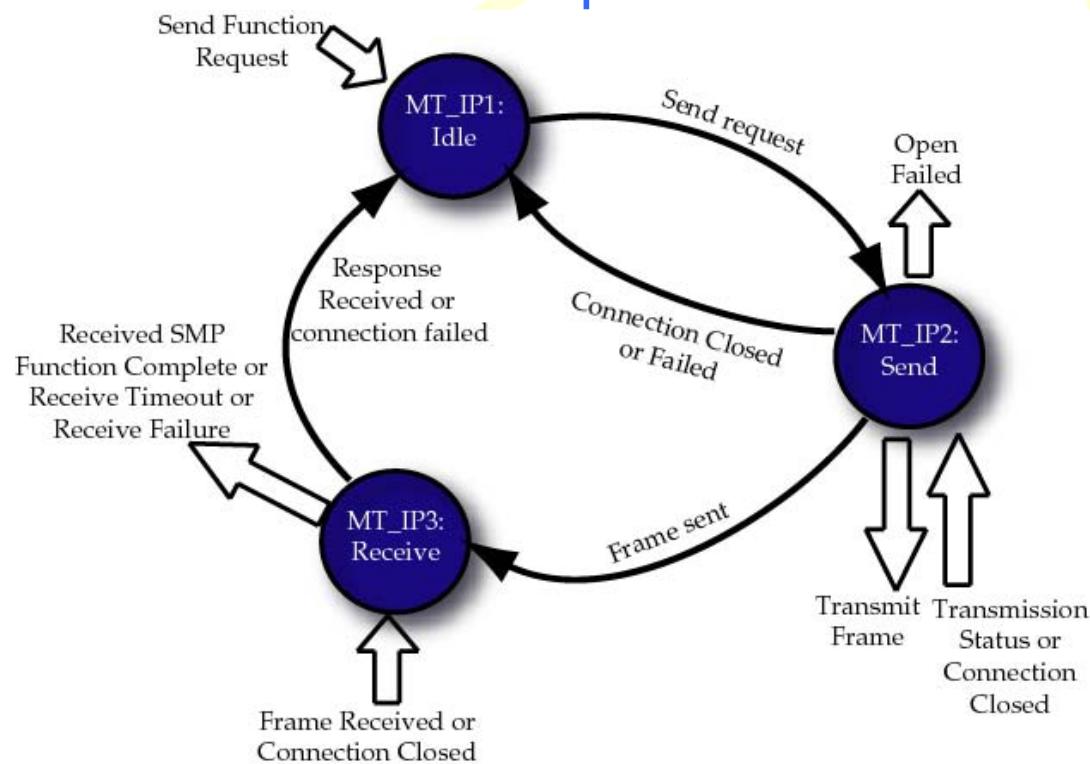
# SMP Frame Sequence (245)

- Initiator must open the connection
- Only one frame transferred in each direction (no RRDY or XFER\_RDY sent or checked)
- Both sides close the connection after SMP\_RESPONSE is transferred



# SMP Transport layer State Machines (256)

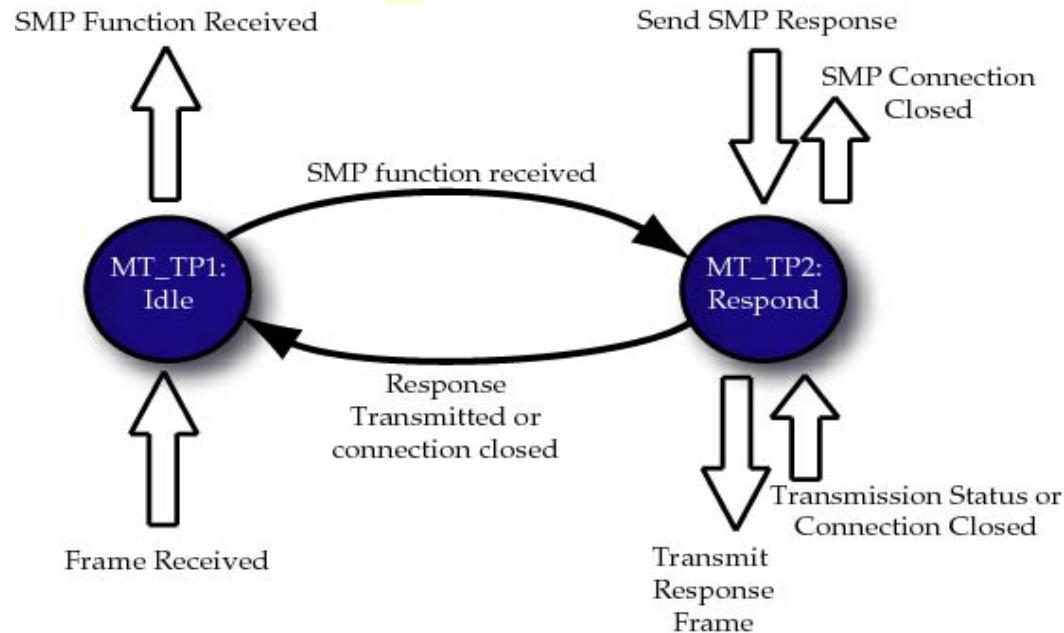
- MT\_IP – SMP transport layer initiator port
  - Upon request from SMP application layer
    1. Transmit an SMP Request frame
    2. Receive an SMP Response frame



# SMP Transport layer

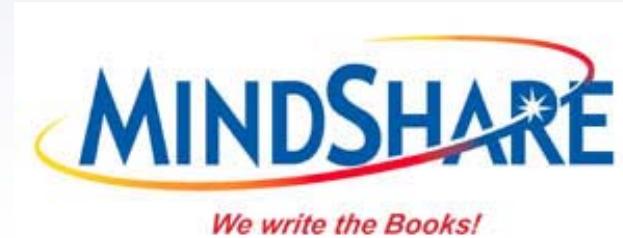
## State Machines (257)

- MT\_TP – SMP transport layer target port
  - Upon arrival of a frame
    - Receive an SMP Request frame
    - Transmit an SMP Response frame

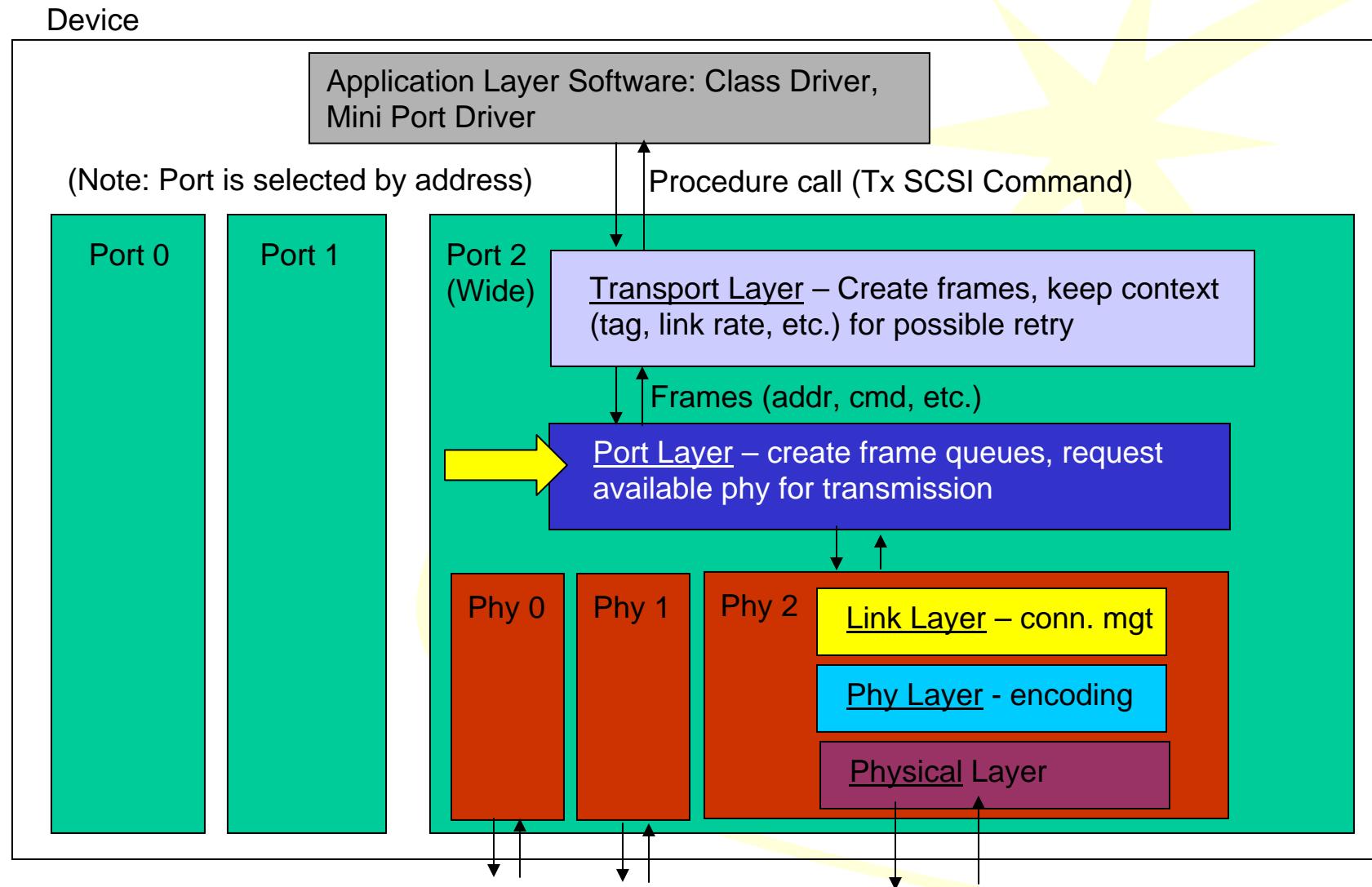


# *Chapter 10*

## *Port Layer*



# Layer Responsibilities



# Port Layer

Most interesting for transmitters with wide ports  
because it selects which phy to use for a connection.

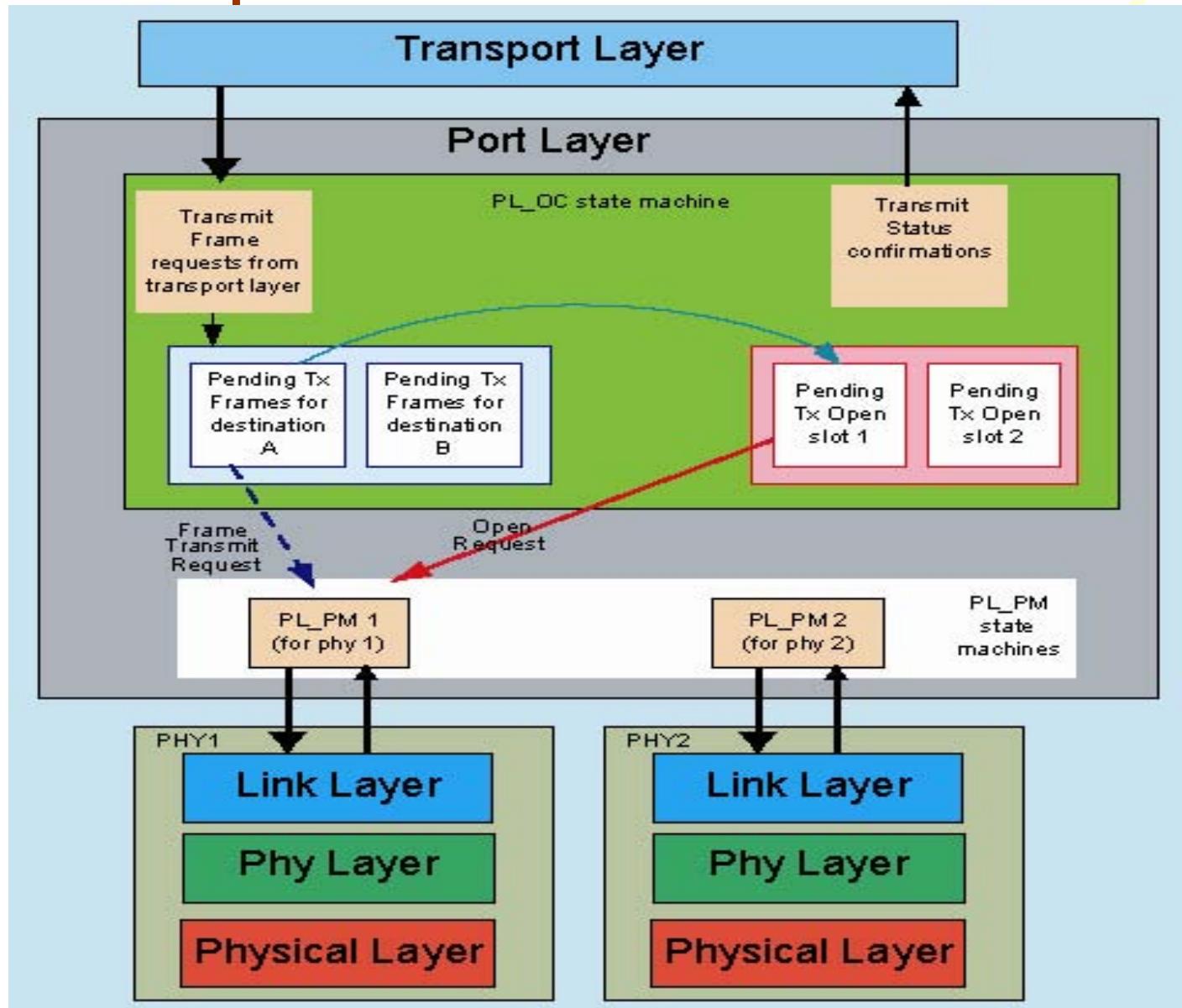
## Notes:

1. Expanders don't have a port layer. Instead, the expander function sorts out which phy to use.
2. SATA devices don't have a port layer either, because they can't implement wide ports.

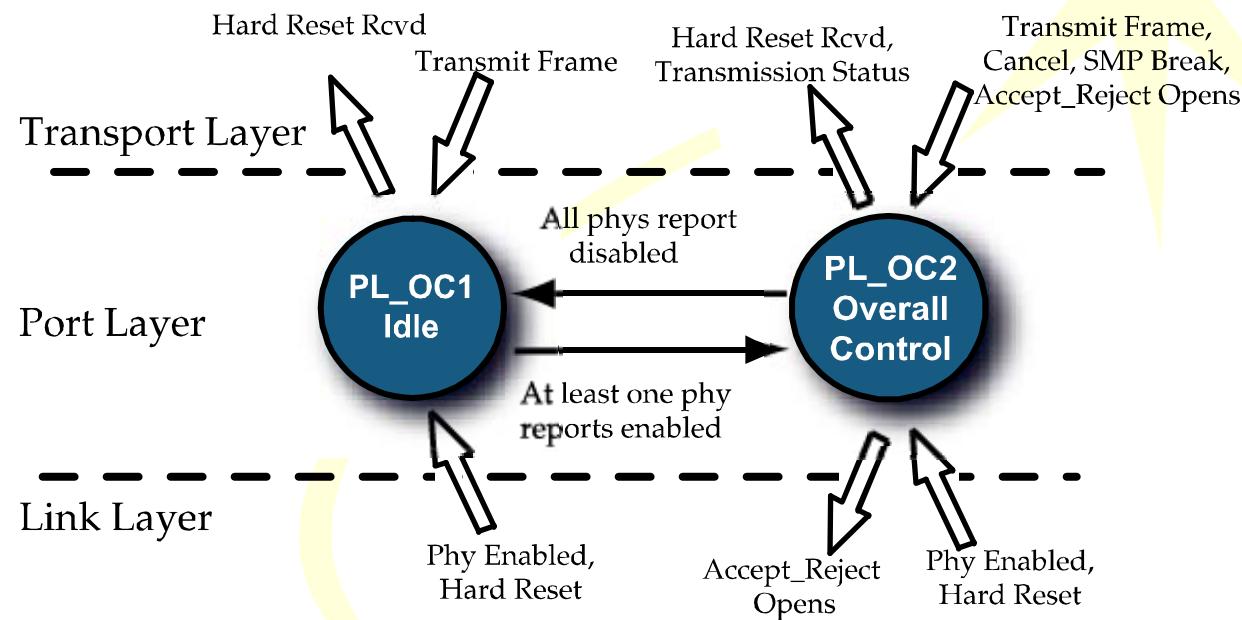
# Port Layer

- Interfaces with Transport layer :
  - Receives frames and queues up requests to send them
  - Returns transaction status and notice of incoming connections
- Interfaces with Phys in the port:
  - Selects phy for outgoing frames
  - Sends requests to open or close connections
  - Forwards frames to phy with open connection
  - Receives transaction status and notice of incoming connections
  - Receives status of connection progress

# Example: Wide Port with Two Phys



# Port Layer Overall Control S/M (263)



# Port Layer State Machines

PL\_OC (Overall Control) – one per port

- Basic functions:
  - Receive enabled or disabled message from each phy
  - Receive transmission requests from Transport Layer
    - Creates queue of pending frames for each destination address (I\_T nexus)
    - Forwards frames to PL\_PM when connection is available

# PL\_OC Interactions

- If no connection is available to send frame to the destination address, create a pending OPEN
  - There can be as many pending Opens as there are phys in the port
- If OPEN is pending, send requests to PL\_PM to open connections

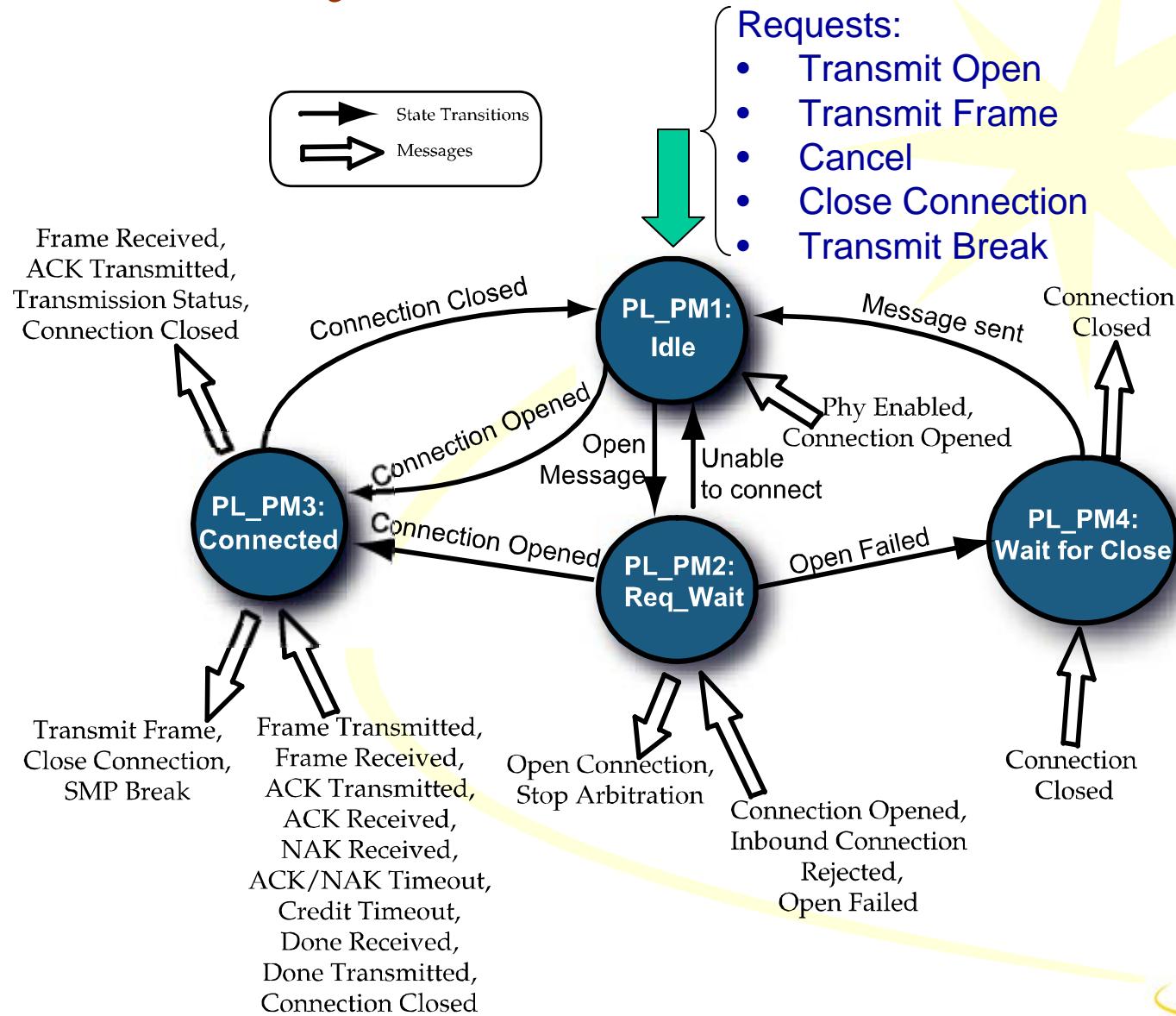
# Transport Layer Interaction (263)

- Requests from Transport Layer
  - Transmit frame
    - Includes information needed for OPEN request, such as destination address, protocol, and connection rate
  - Cancel
    - Stop the progress of the specified request
  - SMP Transmit Break
    - Terminate a connection without waiting for DONE process
  - Accept\_Reject OPENs
    - Upper layers can decide not to accept incoming requests temporarily, default is to accept them

# Transport Layer Interaction

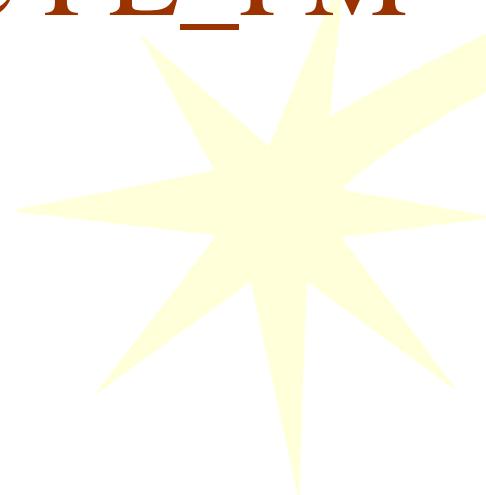
- Responses to Transport Layer
  - Transmission Status – reports problems with transmission of a frame, such as
    - I\_T Nexus Loss
    - Bad Destination
    - Rate not supported
    - Protocol not supported
  - Hard Reset Received

# Port Layer PL\_PM S/M (266)



# Messages from the PL\_PM

- Connection Open
- Retry Open
- Unable to Connect
- Transmission Status
- Retry Frame
- Disable Tx Frame
- Connection Closed



# Examples of Link Layer Interaction

- Requests to Link Layer
  - Open Connection
  - Stop Arbitration (open was cancelled)
- Confirmations from Link Layer
  - Open Failed (Rate not supported, Protocol not supported, etc.)
  - Inbound Connection Rejected
  - Connection Opened (Indicates protocol and whether opened by source or destination)
  - Connection Closed (Normal or timeout)

# Port Layer Timers

## 1. I\_T Nexus Loss

- If target is unable to reestablish connection with initiator, timeout notifies upper layers
  - Timeout value set by the mode page
  - Initiator can optionally do this as well

## 2. Arbitration Wait Time

- If connection is rejected for certain reasons (eg: incoming OPEN overrides the outgoing OPEN), AWT is updated and reported in the next Open Address frame
  - Unlike the others, this one is not related to any mode page value

# Port Layer Timers

## 3. Bus Inactivity Time Limit

- Measures time between sending frames in a connection
  - Timeout value set by mode page

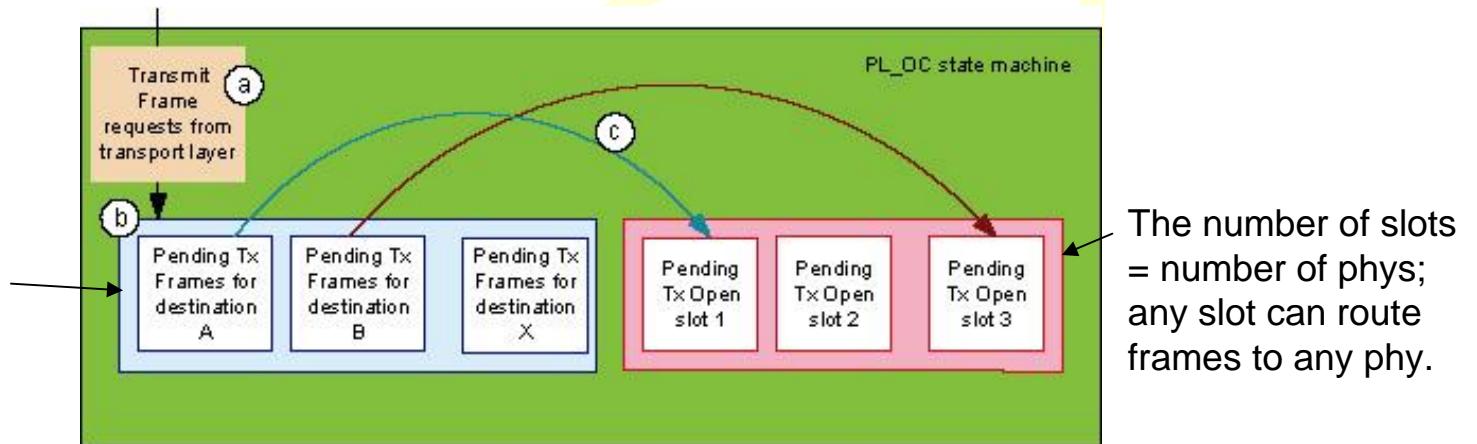
## 4. Max Connect Time Limit

- Max time a connection can stay open, regardless of activity
  - Timeout value set by mode page

# Port Layer Example (267)

- Receive requests from Transport Layer and store Pending Frames
- PL\_OC constantly tries to send OPEN for each pending frame to an available phy

These are queues for each destination address. The number is design-dependent.



PL\_PM  
state  
machines

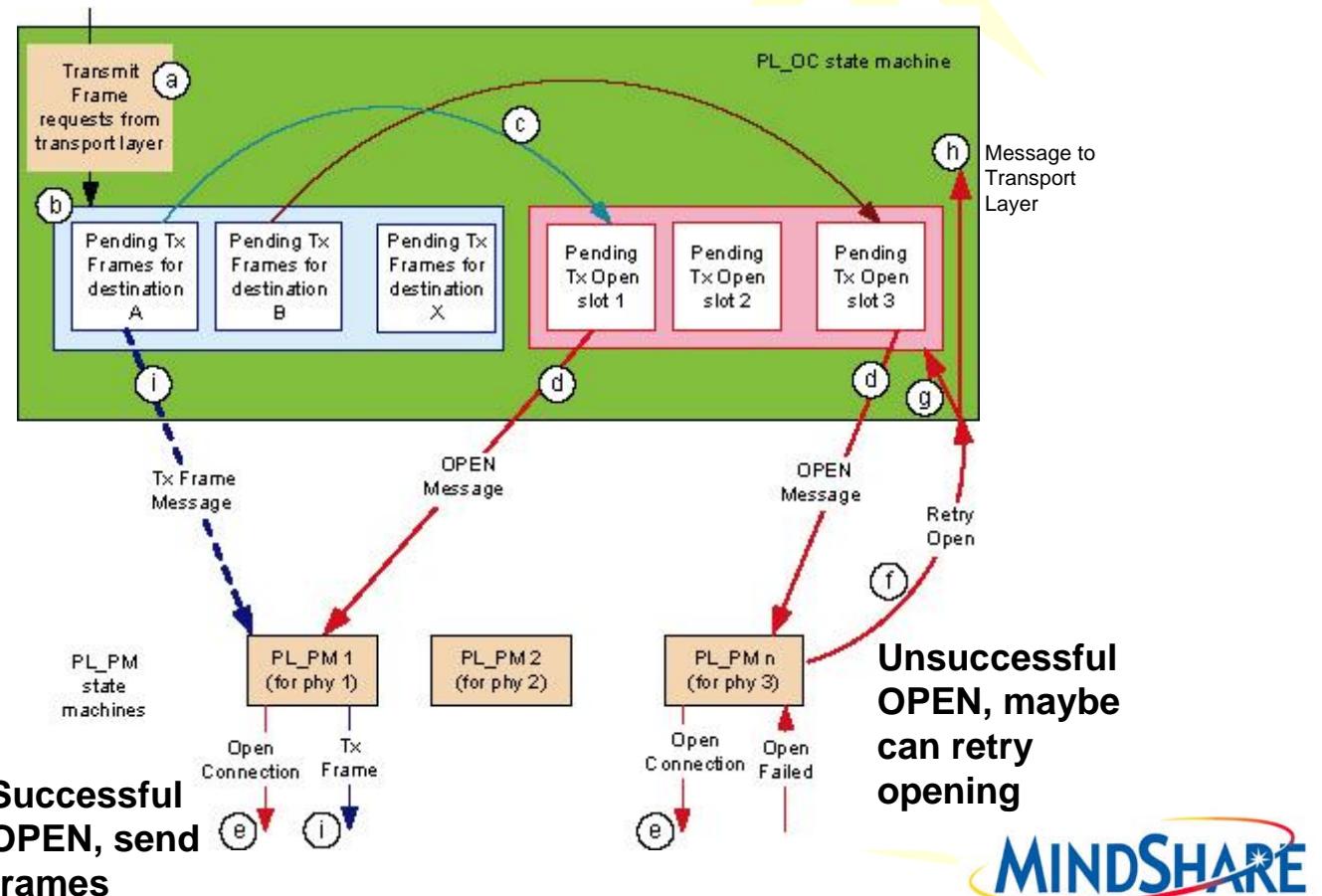
PL\_PM 1  
(for phy 1)

PL\_PM 2  
(for phy 2)

PL\_PM n  
(for phy 3)

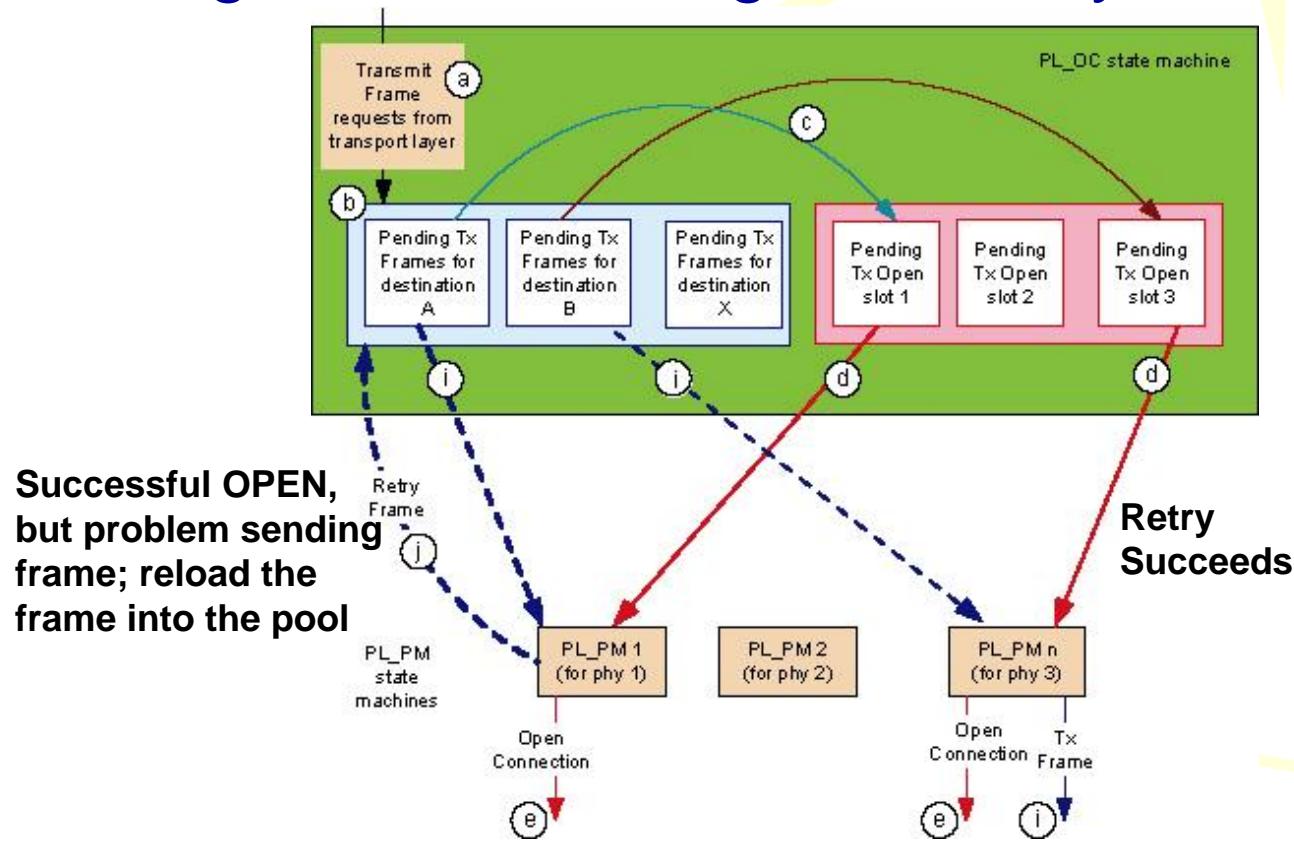
# Port Layer Example (269)

- Send OPEN message to attempt connection
  - If successful, forward the pending Frame Message
  - If unsuccessful, may be able to try Open again



# Port Layer Example

- If retried OPEN succeeds, send Frame message
- If OPEN succeeds, but Frame can't proceed to link layer (eg: credit timeout), create another pending Frame message and retry

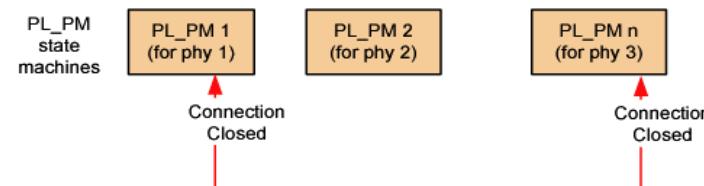
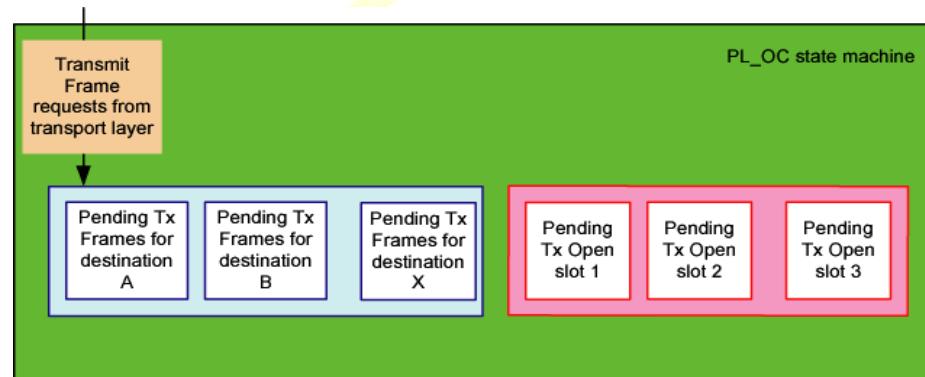


Once a phy is opened, all the transactions for that address are drained out.

Transactions to different address do not have any ordering relationship

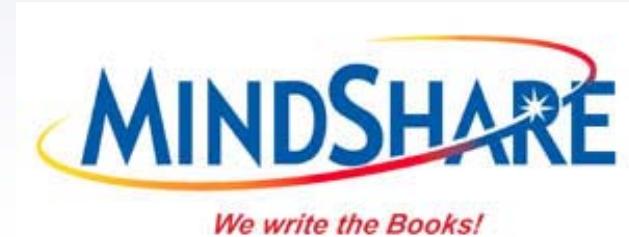
# Port Layer Example

- When all frames have been sent, close the connection, or leave it open for future frames until inactivity timeout or max connect timeout. Decision is design-specific but affects arbitration, which depends on complexity of the system topology.

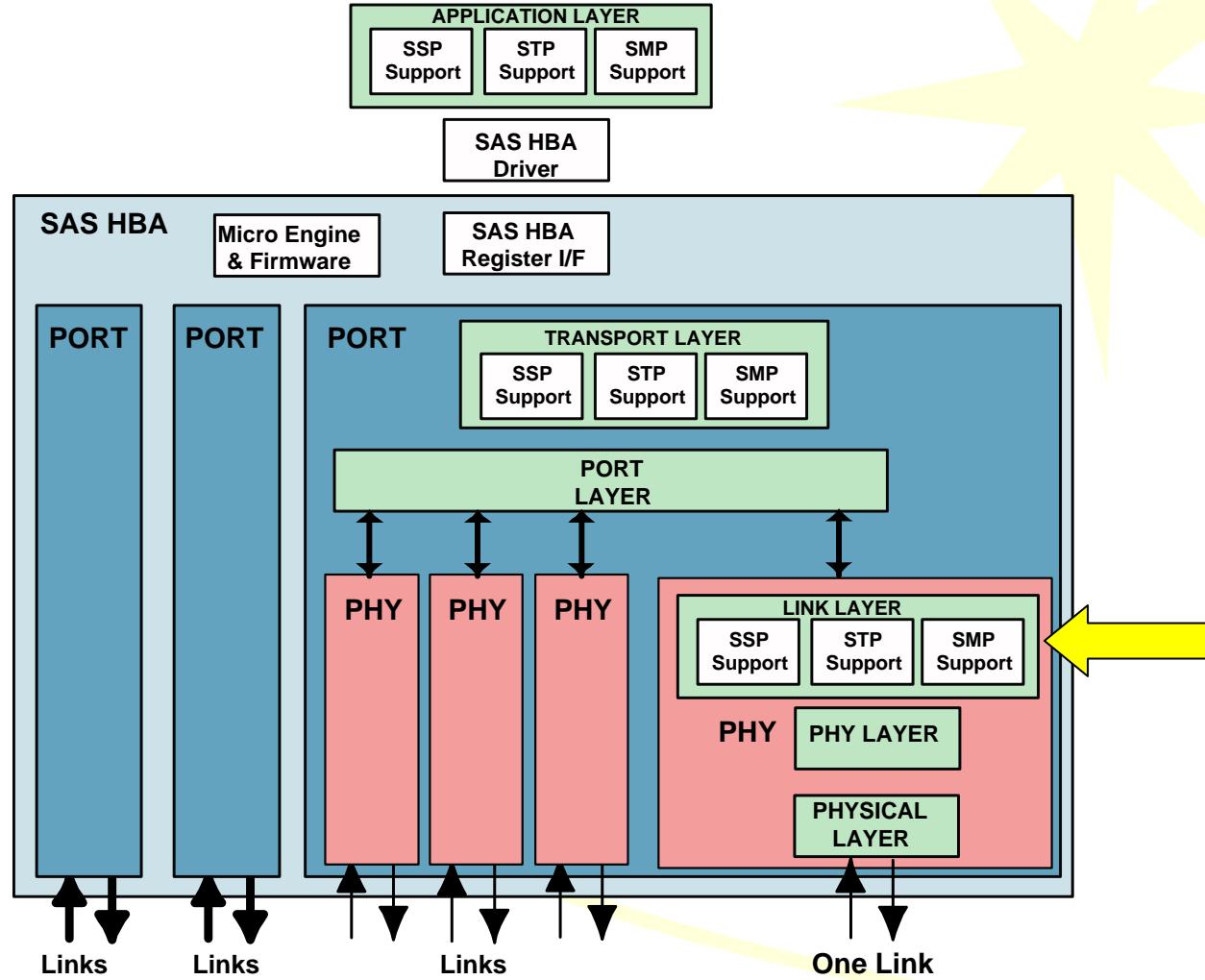


# *Chapter 11*

## *Link Layer Overview*



# Block Diagram of Layers (210)



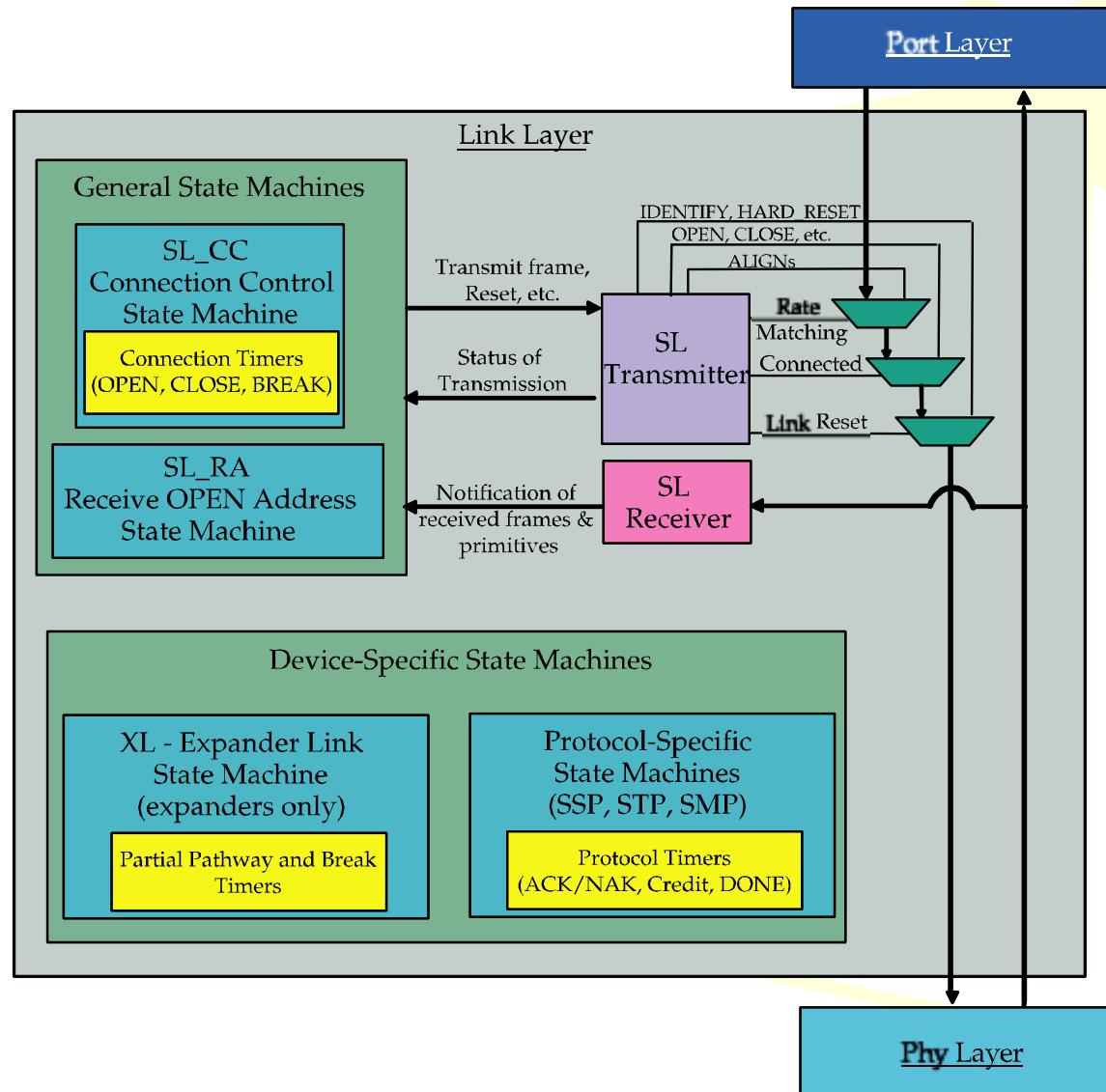
# Link Layer Responsibilities - 1

- Overview (this chapter)
  - Sending and receiving Address Frames
  - Introduction to Primitives
  - Identification Sequence
- Serial Support
  - Clock Skew Compensation (crossing clock boundaries)
  - CRC generation and checking
  - Scrambling
  - Connection Management
  - Generation of Idle cycles
  - Byte ordering

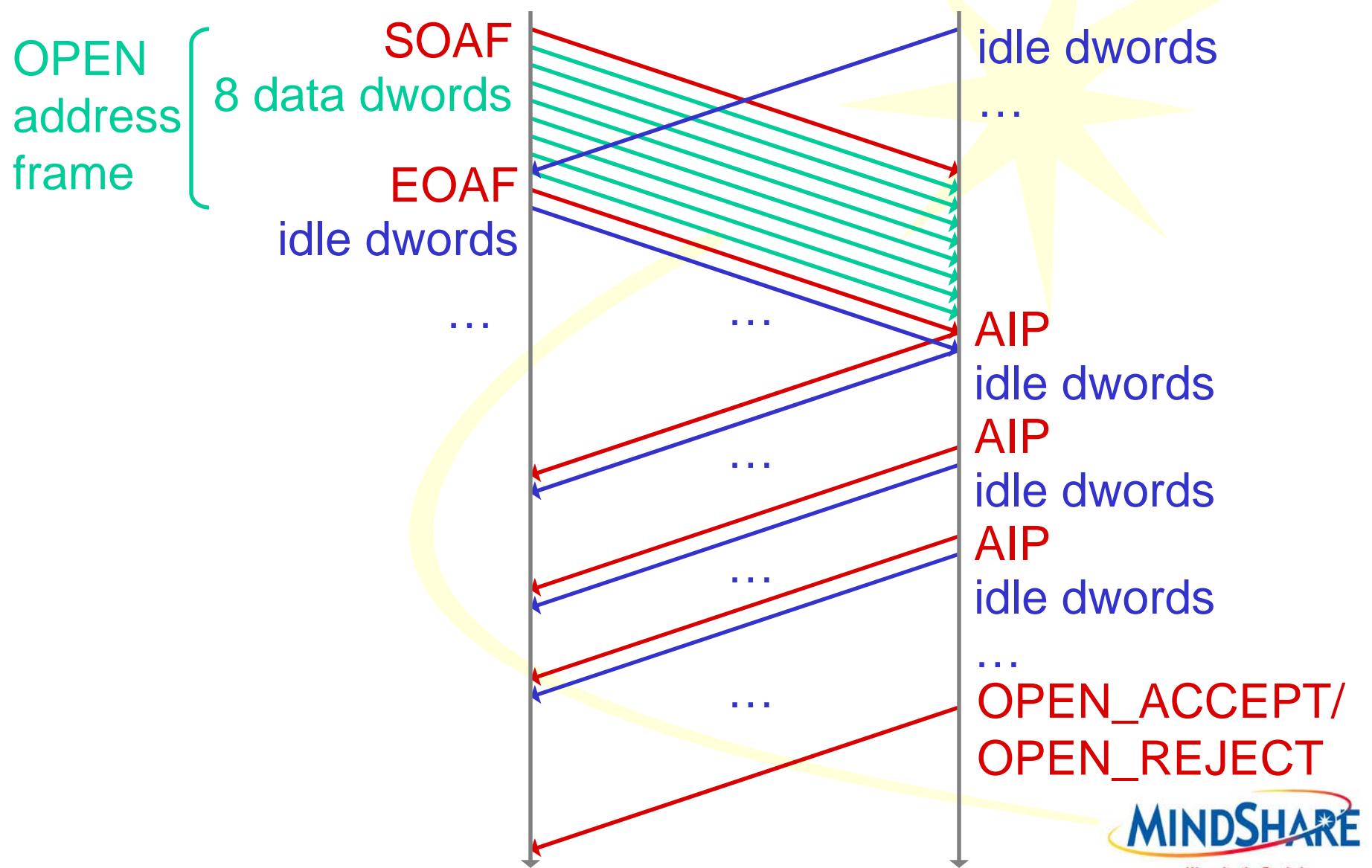
# Link Layer Responsibilities - 2

- Connection management
- Arbitration Fairness
  - Deadlock and Livelock Avoidance
- Protocol Differences
  - Opening and closing a connection
  - Frame Transmission
  - SSP (Serial SCSI Protocol)
    - Flow control, Interlocked frames
  - STP (Serial ATA Tunneling Protocol)
    - Throttling & Flow control, Affiliations
  - SMP (Serial Management Protocol)

# Link Layer Block Diagram (276)



# Opening a Connection (277)



Copyright MindShare, Inc. 2006

# OPEN Address Frame (278)

- OPEN Address Frame
  - Start Of Address Frame (SOAF) primitive,
  - Header contains open request and target address
  - CRC for detection of transmission errors
  - End Of Address Frame (EOAF) primitive

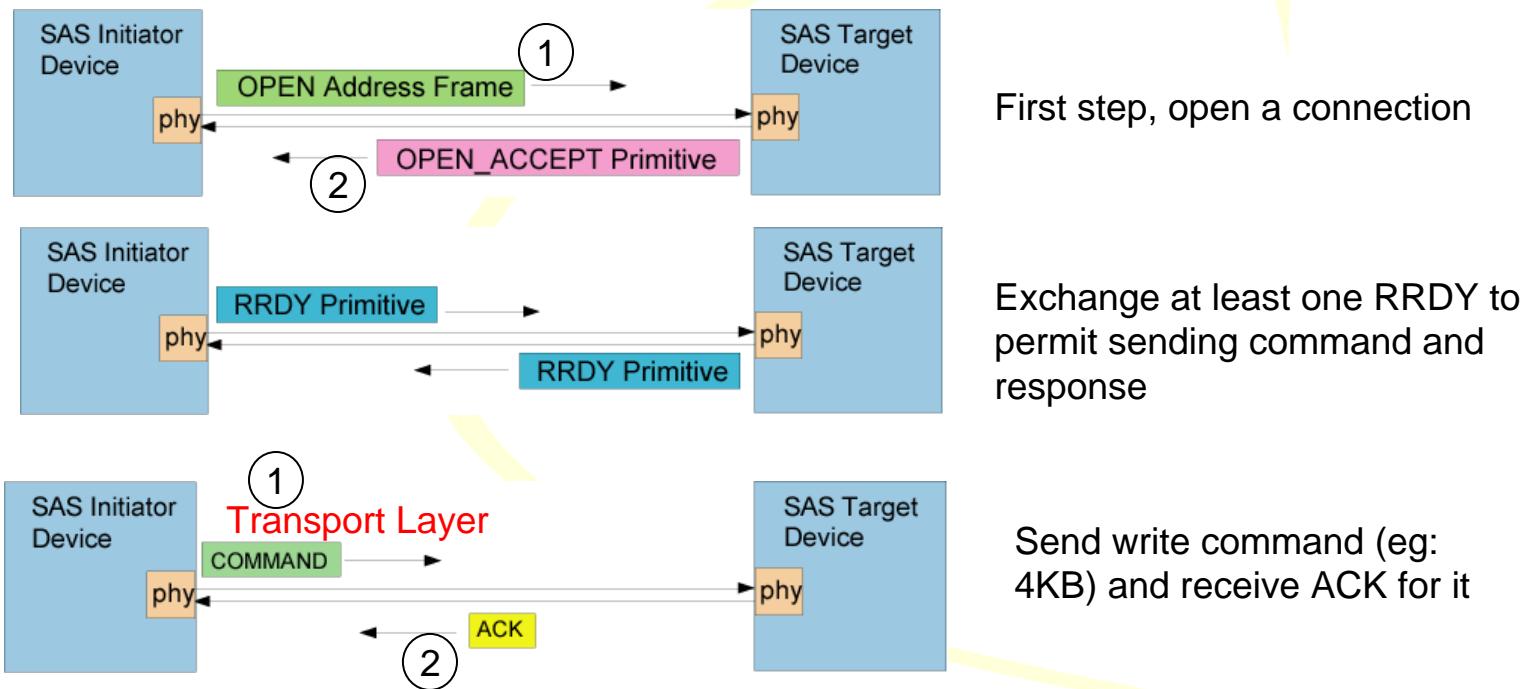


# OPEN Address Frame Header

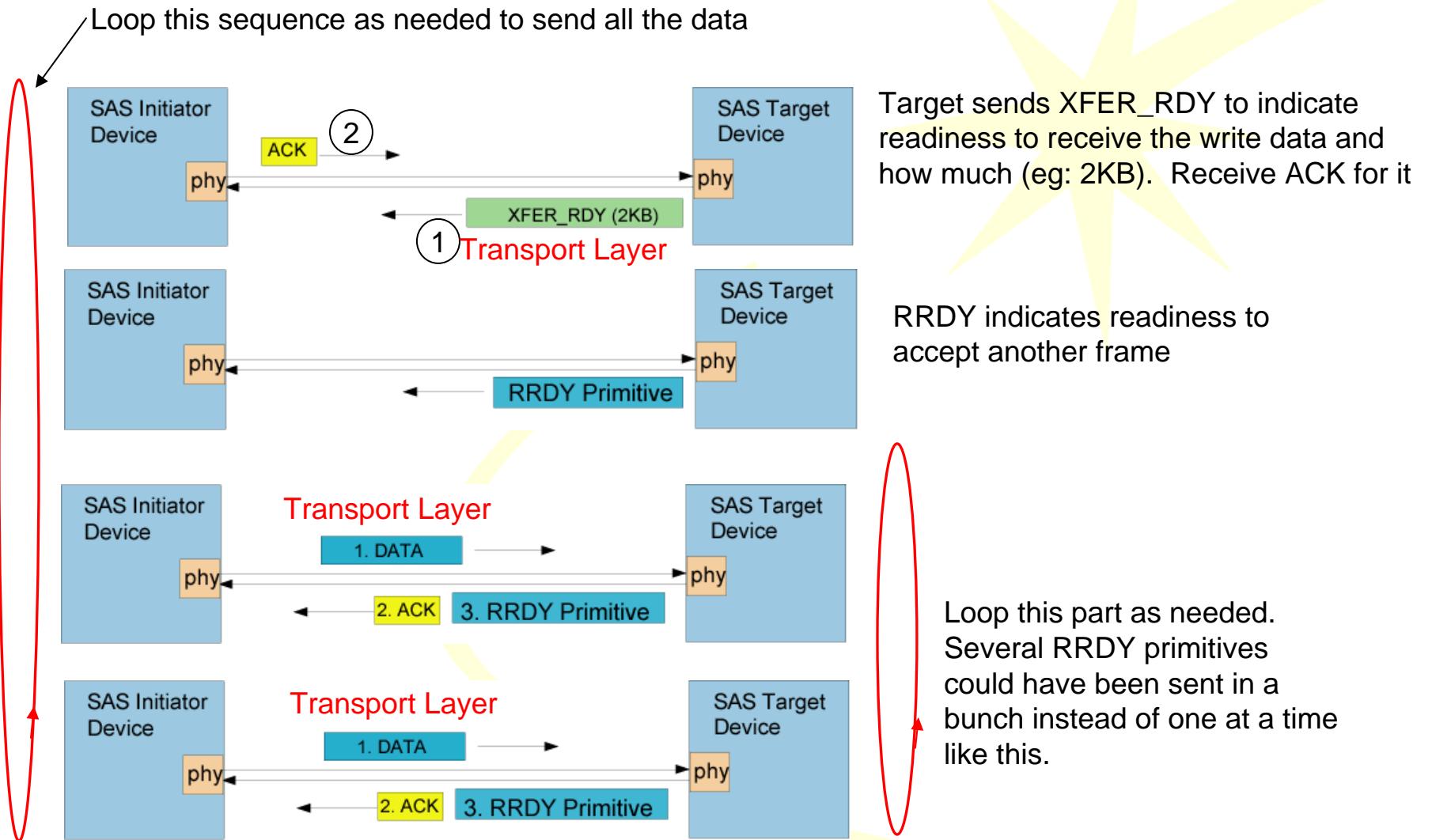
Byte	Fields			
0	Initiator Port	Protocol	Address Frame Type = 1	Indicates whether the source port is acting as initiator or target for this transfer SSP, STP, or SMP
1	Features ( <u>= 0 for now</u> )		Connection Rate	1.5 or 3.0 Gbps
2 to 3			Initiator Connection Tag	Target can use tag when opening connection in reply to command, and initiator will be able to easily look up context. Initiator will see command tag later
4 to 11	<b>Destination SAS Address</b>			
12 to 19	<b>Source SAS Address</b>			Where to route the frame Originator's address for later replies
20	Compatible Features ( <u>= 0 for now</u> )			
21	Pathway Blocked Count			Number of times this access has been blocked, used for arbitration fairness
22 to 23	Arbitration Wait Time			
24 to 27	More Compatible Features ( <u>= 0 for now</u> )			Length of time this access has been waiting, used for arbitration fairness
28 to 31	CRC			

# Link Layer Example

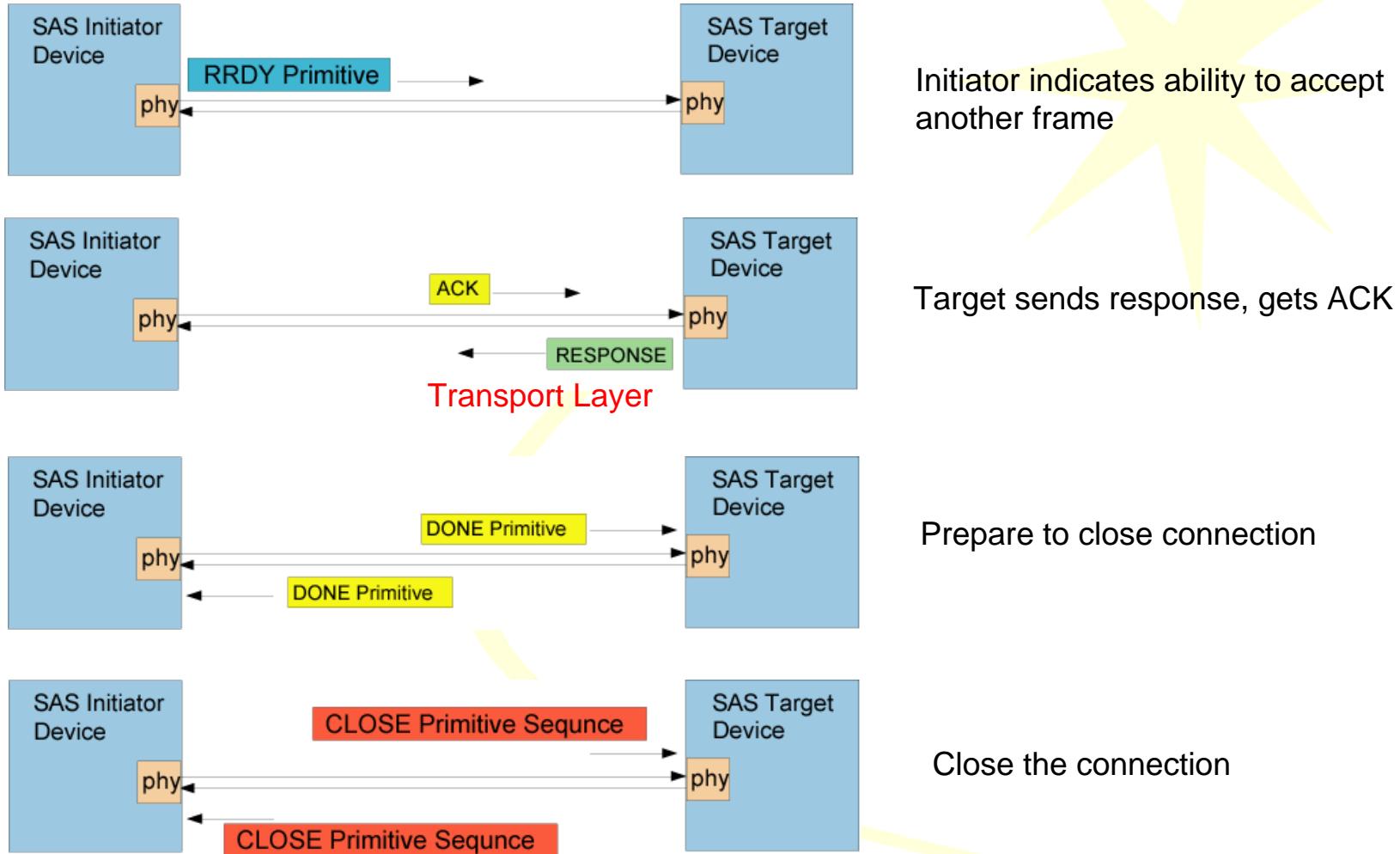
- SSP Write Example: some commands are labeled as Transport layer responsibility; all others are Link Layer responsibility



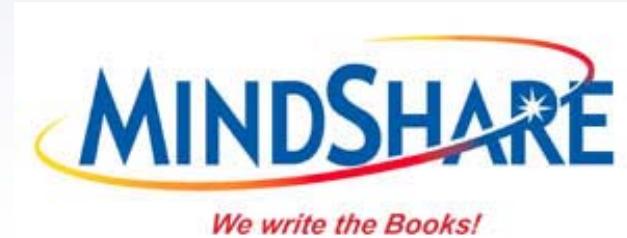
# Example – part 2



# Example – part 3



# *Primitives*



Christy Choi(christy.choi@ sandisk.com)  
Do Not Distribute

# Introduction to Primitives

- Primitives are special dwords that are recognized because they start with a special identifying character (K28.5 for SAS, K28.3 for SATA, and K28.6 for SATA errors).
  - Description of this nomenclature in phy layer chapter
- Definitions:
  - Dword = 4 characters, either data or primitive
  - Byte = 8 bits, encodes to a 10-bit character
- Primitives are used for things like managing
  - connections
  - clock-compensation
  - flow control

first	second	third	fourth
K28.5	Dxx.y	Dxx.y	Dxx.y

# Note on Endianness

- SAS is byte-wise big endian, while SATA is byte-wise little endian
- But primitives are not affected by Endianness; both SAS and SATA send the control character first in the serial stream.

	first	second	third	fourth	
K28.5	Dxx.y	Dxx.y	Dxx.y	Dxx.y	SAS Primitive
K28.3	Dxx.y	Dxx.y	Dxx.y	Dxx.y	SATA Primitive
K28.6	Dxx.y	Dxx.y	Dxx.y	Dxx.y	SATA Error Primitive

# Primitives (287)

- Three Functional categories of primitives
  1. Those not tied to specific types of connections
    - Some are used outside connections only, others inside or outside connections
  2. Those used inside SSP and SMP connections
  3. Those used inside STP connections
    - Encodings and functionality defined by SATA
- Several primitives have multiple versions
  - To communicate functional differences (e.g. different reasons for an OPEN\_REJECT)
  - To reduce EMI (electromagnetic interference) for example, by rotating through 4 different ALIGNs

# Some Primitives are not tied to specific connections

- Connection management
  - AIP – arbitration in progress
  - BREAK – break a connection (emergency close)
  - CLOSE – close a connection
  - OPEN\_ACCEPT – accept a connection request
  - OPEN\_REJECT – reject a connection request
  - SOAF, EOAF – start/end of address frame
- General
  - ALIGN – clock skew management, rate matching
  - NOTIFY – version of ALIGN that communicates information
  - BROADCAST – expander broadcasts to all initiators
  - ERROR – for forwarding invalid dwords
  - HARD\_RESET – force a low level reset

# Primitives for SSP and SMP connections (288)

- Used in SSP connections
  - RRDY – permission to send a frame
  - CREDIT\_BLOCKED – warning that an RRDY is not coming soon
  - SOF, EOF – start and end of frame
  - ACK, NAK – positive and negative frame acknowledgement
  - DONE – prepare to close connection
- SMP uses SOF and EOF

# Primitive sequences

- To reduce problems from errors, some primitives are sent multiple times. For example:
  - Missing an SOF only corrupts that frame and connection; single instance OK
  - Missing a CLOSE is more serious; can confuse devices; receipt of triple instance required

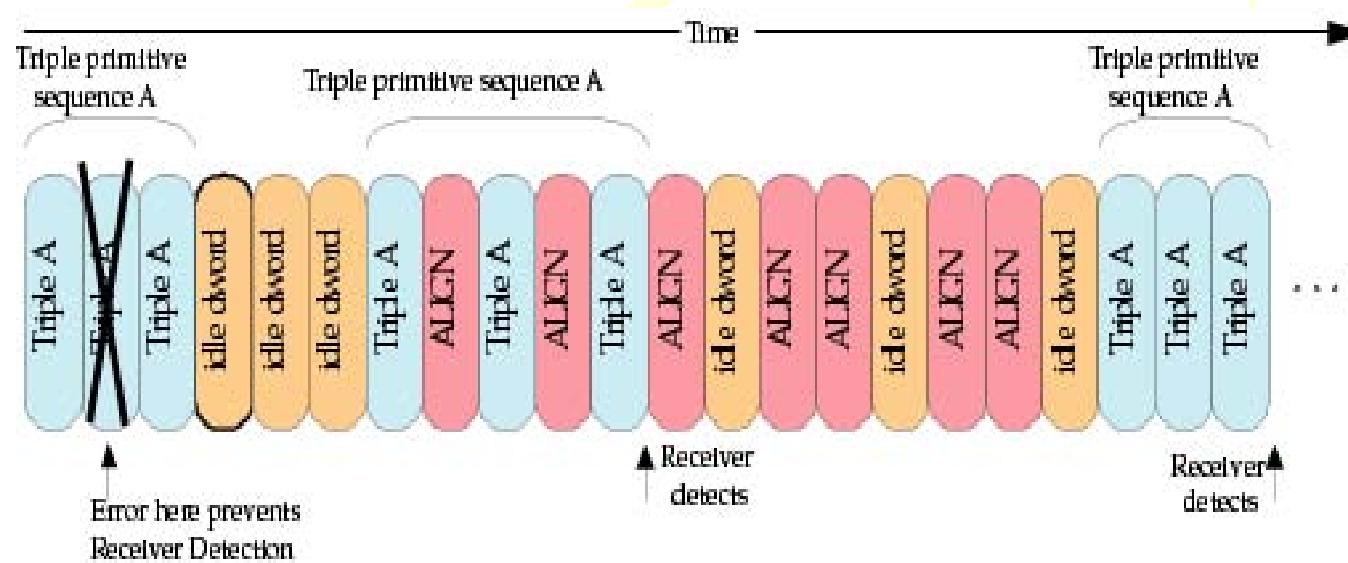
Type	Transmit	Receive
Single	1	1
Repeated	2	1
Triple	3	3
Redundant	6	3

# Single primitive sequence

- Just one instance of the primitive is transmitted
- Subject to single-bit errors
- Most primitives independent of connections are single primitive sequences
  - AIP, ALIGN, NOTIFY, OPEN\_ACCEPT, OPEN\_REJECT, SOAF, EOAF, ERROR
- All primitives used only inside SSP and SMP connections are single primitive sequences
- SATA\_SOF, SATA\_EOF, and SATA\_ERROR inside STP connections are single primitive sequences

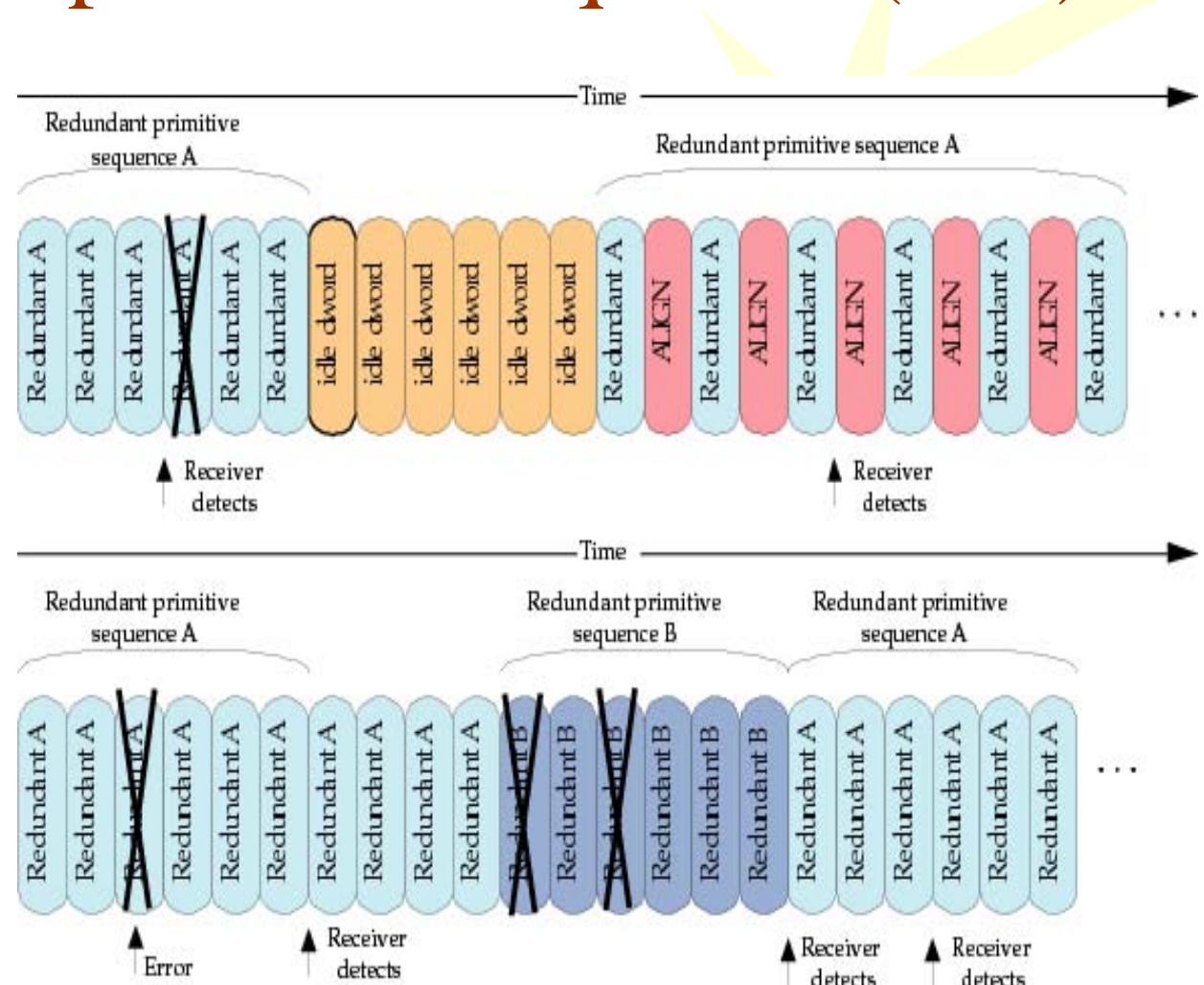
# Triple primitive example (291)

- Send-3, receive-3
- Error causes primitive to be missed
- Don't detect again until 3 other dwords arrive
- **CLOSE** is the only triple primitive sequence; guards against accidental closure



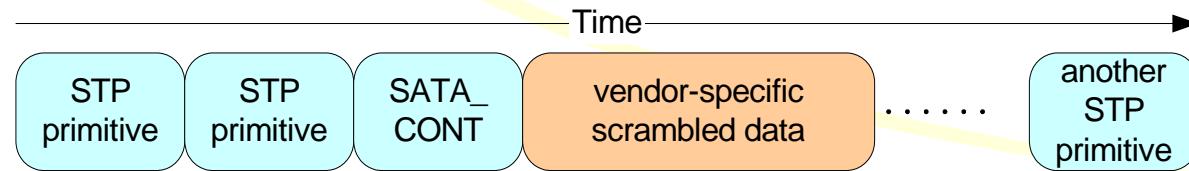
# Redundant primitive sequence (292)

- Send 6, must receive 3
- Allows tolerating some errors
- Don't detect again until 6 other dwords arrive
- BREAK, HARD\_RESET, BROADCAST are the only redundant primitive sequences

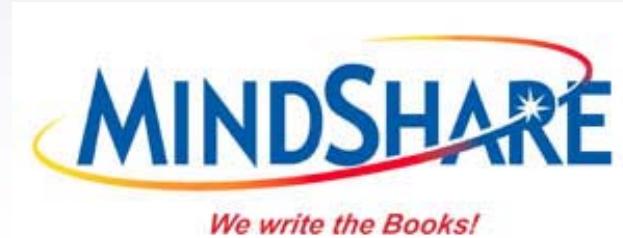


# Repeated primitive sequence

- SATA sometimes sends long strings of repeated primitives while waiting for a response from the other device, but this repetition can cause unacceptable EMI
- To handle this case in STP connections:
  - Send the primitive at least twice
  - Send the special SATA\_CONT primitive once
  - Send vendor-specific scrambled data dwords that are ignored by the receiver while the previous primitive is understood to continue repeating
  - Sending another primitive exits this mode
- Most SATA primitives can be repeated



# *Identification Sequence*



# Identification

- SAS devices need to know the addresses in the system to which they can open connections
- After power on, attached devices inform each other of their addresses by sending an Identification frame that contains their address and device type
- Using this as a starting point, initiators perform discovery to find all the devices they can access

# Identification

- Although this sequence is a complete frame, it's still a link layer responsibility and doesn't require transport layer support



# Identify Address Frame Format

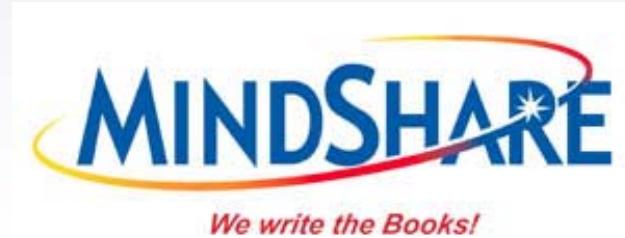
	Bit 7	6	5	4	3	2	1	0
Byte 0	Restricted	Device Type						
1								Address Frame Type = 0
2				SSP Init	STP Init	SMP Init	Restricted	
3				SSP Target	STP Target	SMP Target	Restricted	
4 - 11								Restricted
12-19								SAS Address
20								Phy Identifier
21-27								Reserved
28-31								CRC

Annotations:

- Device Type: End device, edge expander, fanout expander
- Initiator or target and protocol type: SSP, STP, SMP
- Address of the SAS port: SAS Address
- The phy number inside the attached device: Phy Identifier

# *Chapter 12*

## *Link Layer - Serial Support*



# Serial Support

- Link layer manages several aspects of serial communication, including:
  - Clock Skew Compensation
  - Error checking using CRC
  - Data scrambling

# Clock Compensation (300)

- Embedded-clock designs must recover the clock at the receiver and re-synchronize with local clock (a process referred to as crossing clock domains)
- SAS clocks are specified accurate to within a tolerance of +/- 100ppm (parts/million)
  - $100\text{ppm} = 100/1\text{million} = 0.0001 = 0.01\%$
  - Neighboring devices could each have 100ppm variation, for a total difference between them of 200ppm
- SATA clocks are specified to +/- 350ppm and support Spread-Spectrum Clocking, giving a range from +350ppm to -5350ppm around the center frequency

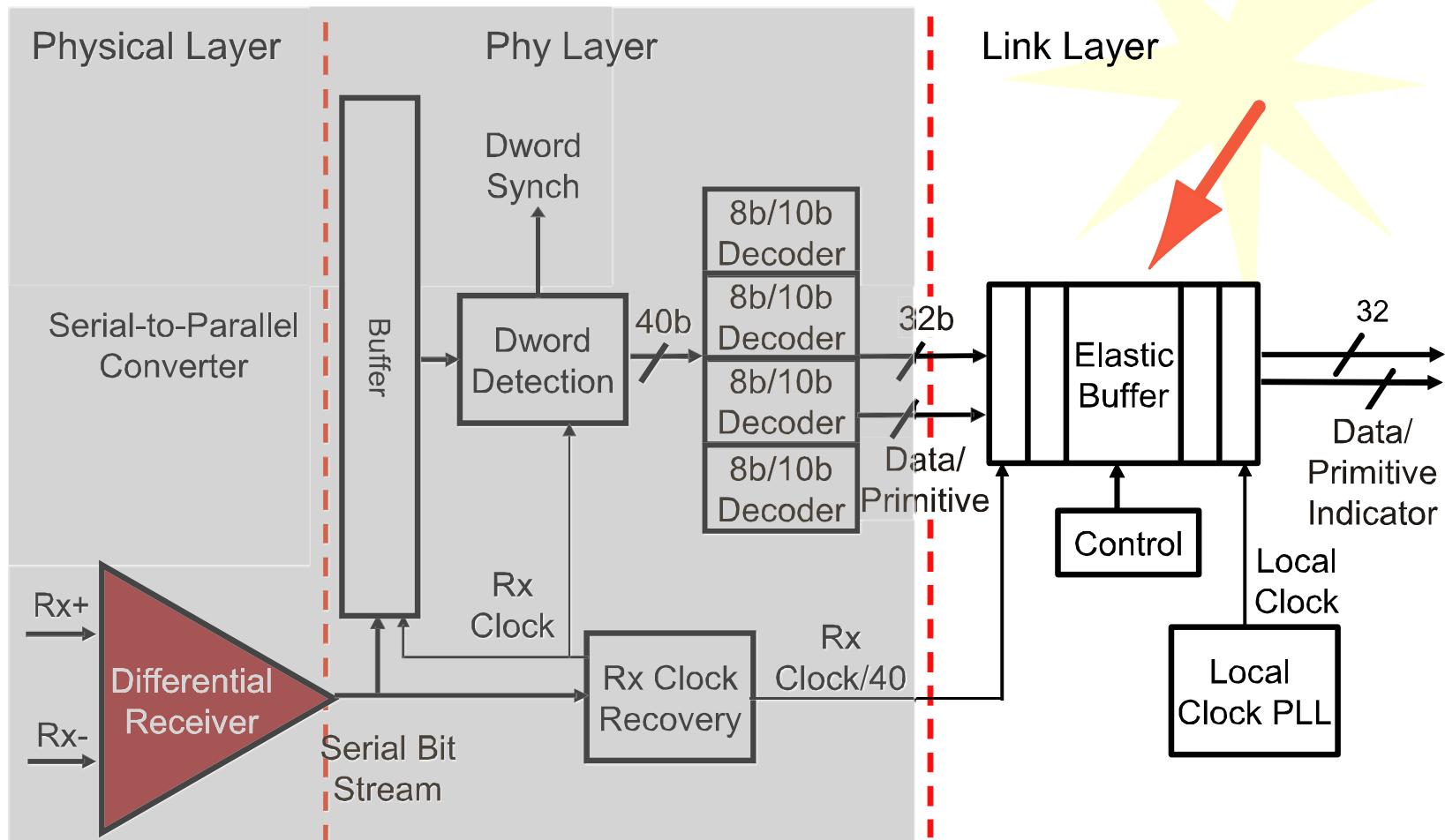
# Clock Management

- Needed for most serial designs. SAS Goals in this regard: simplicity and low cost
- Max skew is 200 ppm for SSP and SMP
  - Receive buffer could gain or lose one dword every 5000 dwords
- Max skew is 5700 ppm for STP (when SATA device is using SSC)
  - Receive buffer could gain or lose one dword every 175 dwords
- In both cases, elasticity buffer needs to be managed to avoid overrun or underrun

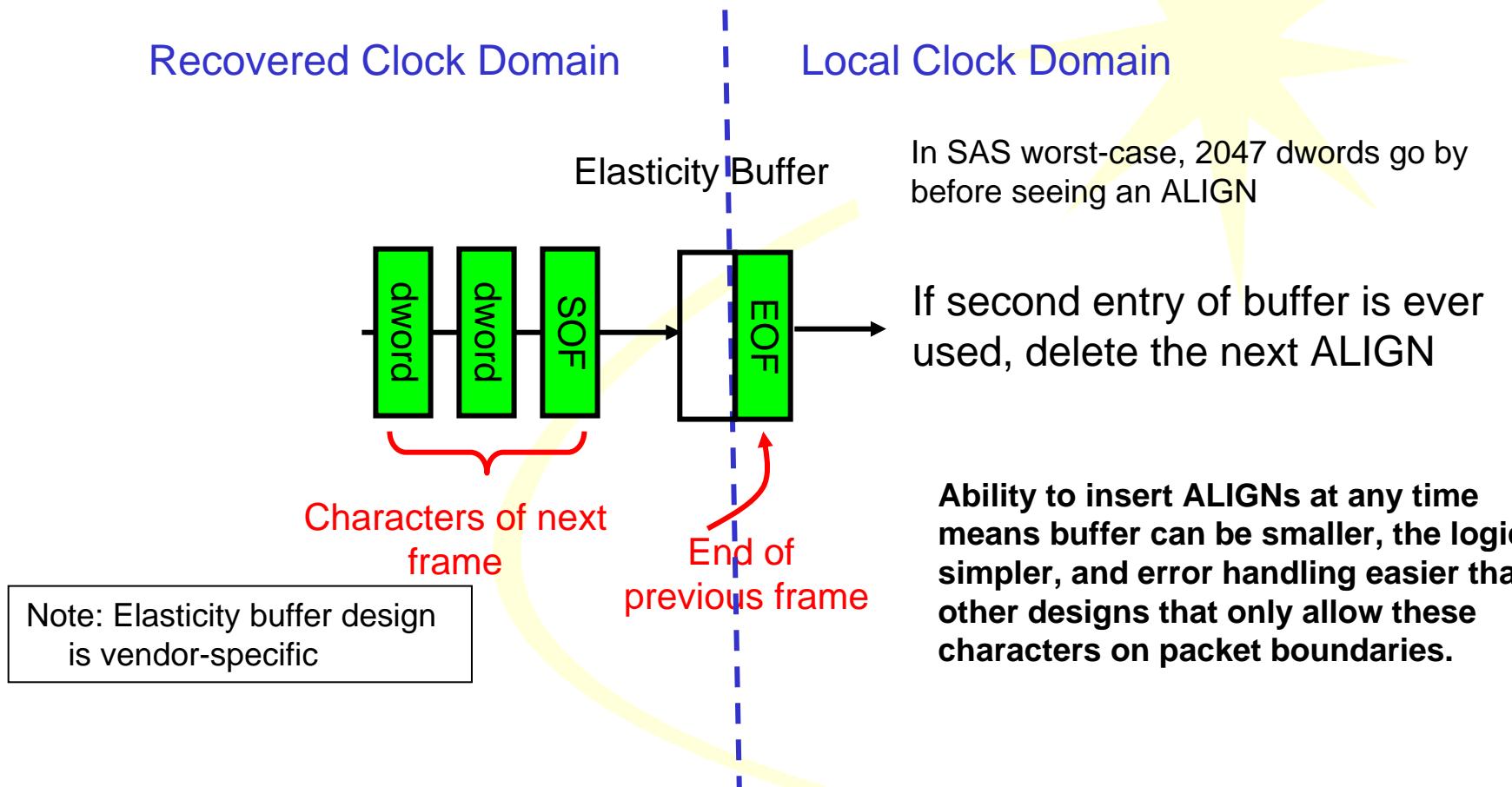
# Clock Skew Compensation

- Phy layer transmitter inserts two ALIGN dwords for every 4096 dwords in 3.0GHz SAS, and 2 for every 256 dwords for SATA
  - NOTE: These can be inserted at any time and don't have to wait for a frame boundary, simplifying the process.
- Receiver
  - When approaching an overrun condition, delete incoming ALIGNs
  - When approaching an underrun, add more ALIGNs to the internal stream (they'll be discarded by internal logic later)

# Elasticity Buffer

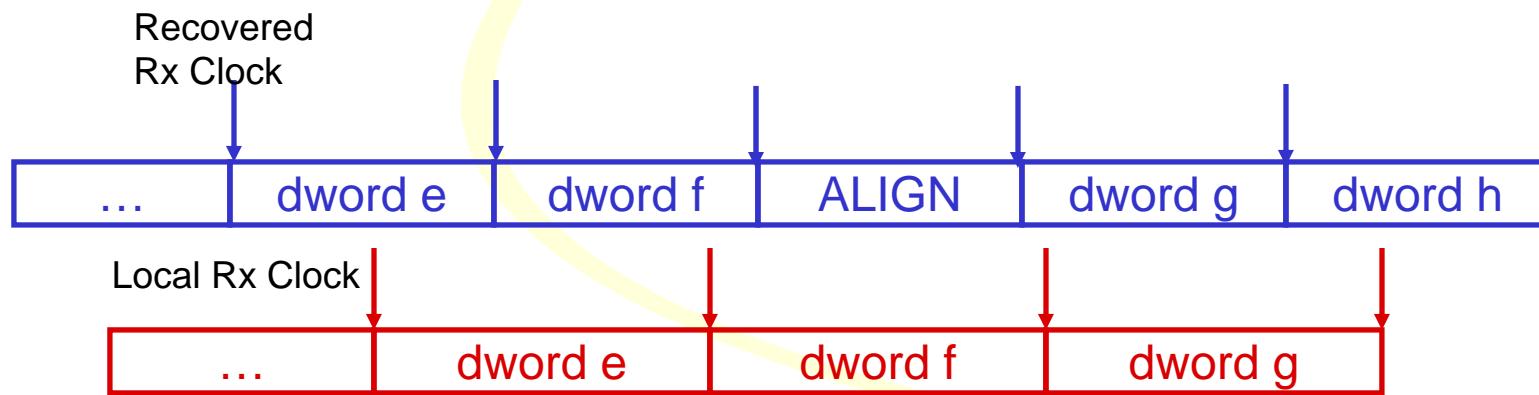


# Elasticity Buffer Example (304)



# Clock Skew Example (304)

- Transmitter clock is slightly faster
  - Data is latched into Elasticity buffer with Recovered clock (upper arrows)
  - Data is latched out with Local Rx clock (lower arrows), difference between clock rates causes buffer to begin to fill.
  - Elasticity buffer recognizes this and discards the ALIGN, so dword g is ready for the next clock



# Metastability

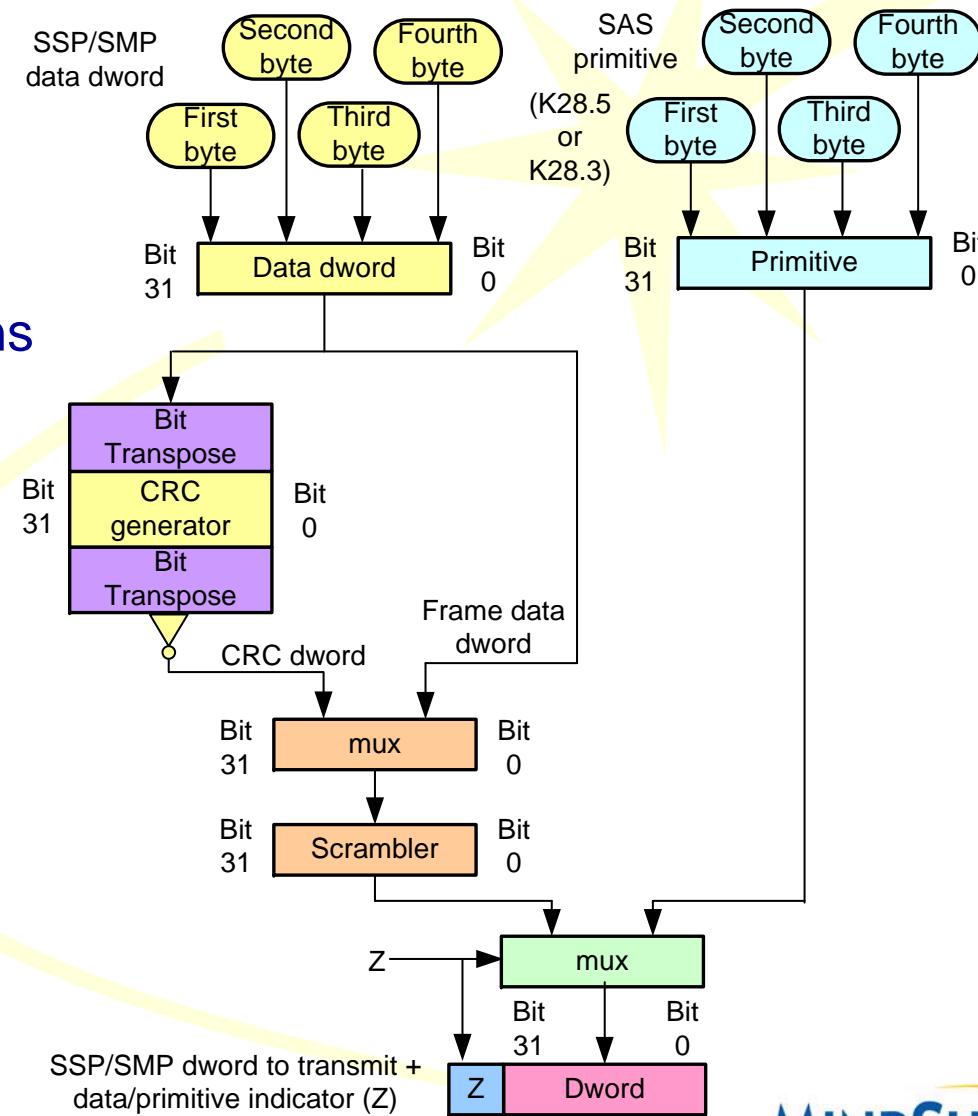
- Another consideration when crossing clock domains is avoiding metastability issues
  - Metastability (undefined state) for flip-flop output can occur when setup or hold timing is violated. Since data is not synchronized to the local clock this could happen.
  - Impossible to avoid completely, but likelihood is greatly reduced by including more register stages.
- In practice, elasticity buffers are typically designed to be at least 4-deep, partly for this reason.

# CRC Generation

- Common approach for error detection in serial buses
- Every frame protected by a 32-bit CRC (Cyclic Redundancy Code)
  - Mathematical model creates many bit changes in response to small changes in the payload dword, providing robust error detection
- Transmitter calculates CRC based on outgoing frame, appends it to frame
- Primitives do not include a CRC and so bypass this calculation stage

# SSP & SMP CRC Generation (307)

- Address frame, SSP frame, and SMP frame data dwords are big-endian
- Primitives outside connections and inside SSP and SMP connections can be interpreted as big-endian to match
- The CRC covers the unscrambled data
- Note that primitives are not involved in CRC calculations and are not scrambled



# CRC Checking

- Two Possible Receiver Approaches:
  1. Calculate expected CRC from received frame and compare result with the CRC attached to the frame
  2. Calculate CRC over the entire frame, (including the attached CRC) and verify that the output matches an expected constant value.

# CRC Coverage

- All bursts of up to 32-bit inversions are detected (important for serial transfers)
- All odd-number bit inversions detected, though some even numbers of bit inversions can go undetected
- Hamming distance (number of bits that have to change to convert one valid value into another valid value) 4 for up to 91,607 bits
  - Detects all 2-bit and 3-bit errors
  - 4-bit errors could convert one valid frame into another
  - Slightly more than 1 out of  $2^{32}$  possible 4-bit errors are undetectable

# CRC Coverage (309)

- Smaller frames have greater Hamming distance
- Biggest SAS frame, 1KB (8K bits), has worst-case Hamming distance of 4

– “32-Bit Cyclic Redundancy Codes for Internet Applications” by Philip Koopman  
[http://www.ece.cmu.edu/~koopman/networks/dsn02/dsn02\\_koopman.pdf](http://www.ece.cmu.edu/~koopman/networks/dsn02/dsn02_koopman.pdf)

Hamming distance	Number of bits in the frame
15	8 to 10
14	-
13	-
12	11 to 12
11	13 to 21
10	22 to 34
9	35 to 57
8	58 to 91
7	92 to 171
6	172 to 268
5	269 to 2974
4	2975 to 91,607
3	91,608 to more than 131,072

# Scrambling

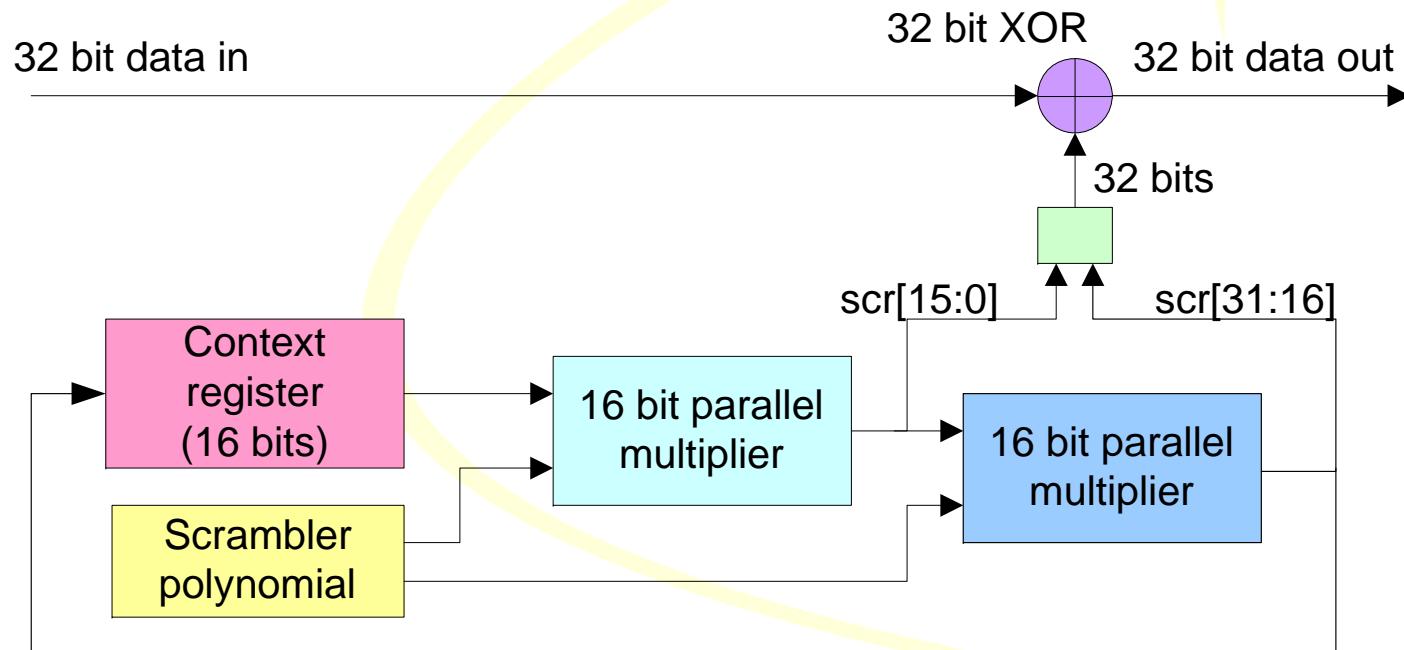
- Purpose: reduce EMI problems resulting from repeated data patterns
- All Data dwords for both SAS and SATA are scrambled by transmitter, and must be unscrambled by receiver
- Scrambling imposes a pseudo-random pattern on transmitted dwords
- Note: Primitives are not scrambled, so care must be taken not to let them repeat for too long or that will defeat the purpose
  - SAS automatically avoids repeating primitives
  - SATA uses a “continue” primitive (SATA\_CONT) to avoid repeating primitives

# Scrambler Implementation

- XOR dwords with output of an LFSR (linear feedback shift register) that implements the following 16-bit polynomial:
  - $G(x) = X^{16} + X^{15} + X^{13} + X^4 + 1$
- SAS SSP uses one LFSR
- SAS STP (SATA) scrambling requires two LFSRs
  - One for STP frame
  - Another for repeated STP primitives when they're sent inside an STP frame (it can happen that not all devices in a path would see repeated primitives if intermediate device discards them, causing scramblers to get out of sync)

# Scrambler Operation

- Annex E of the SAS standard gives the C source code for the following implementation. Two 16-bit multipliers are used since the width of the polynomial is only 16 bits.



# Scrambler Operation (cont)

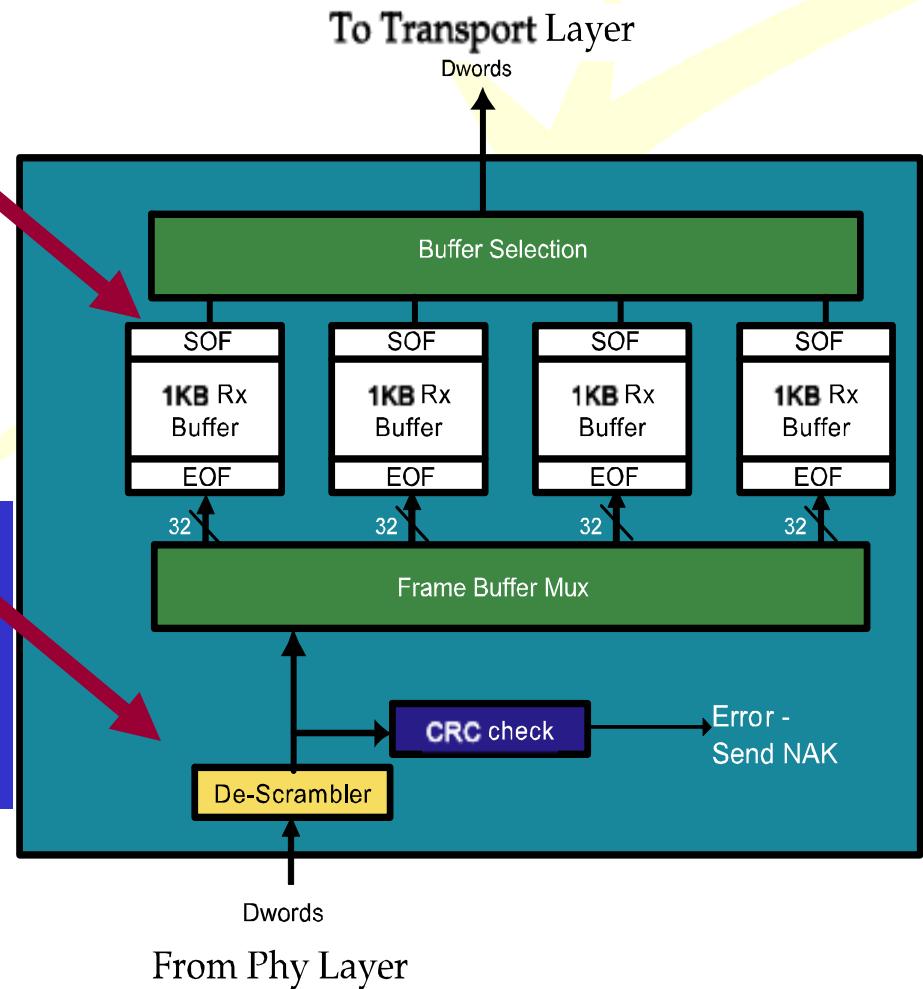
- Scrambler re-initializes to FFFFh when SOF or SOAF are seen (these primitives are not scrambled)
- LFSR has no feedback from the output, so its pattern is predictable

# Receiver De-Scrambles Input Data

- There could be several Rx Buffers holding incoming frames
- Link Layer informs Transport layer when frames have been received
- Transport layer fetches contents of indicated frame buffer

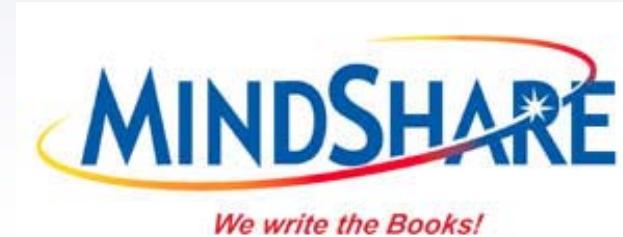
- De-scrambler uses the same scrambling procedure as the transmitter. XORing the data with the same pattern a 2<sup>nd</sup> time recovers the original data.

**Standard does not tightly define layer boundaries or responsibilities, giving designers freedom in implementation specifics**



# *Chapter 13*

## *Link Layer - Connection Management*



# Introduction

- SSP, SMP, and STP Frames are constructed by the Transport Layer
- SSP frames expect to see ACK/NAK in response (interlocked)
- In contrast, address frames
  - Constructed and sent by Link Layer
  - Do not get ACK/NAK

# Address Frames

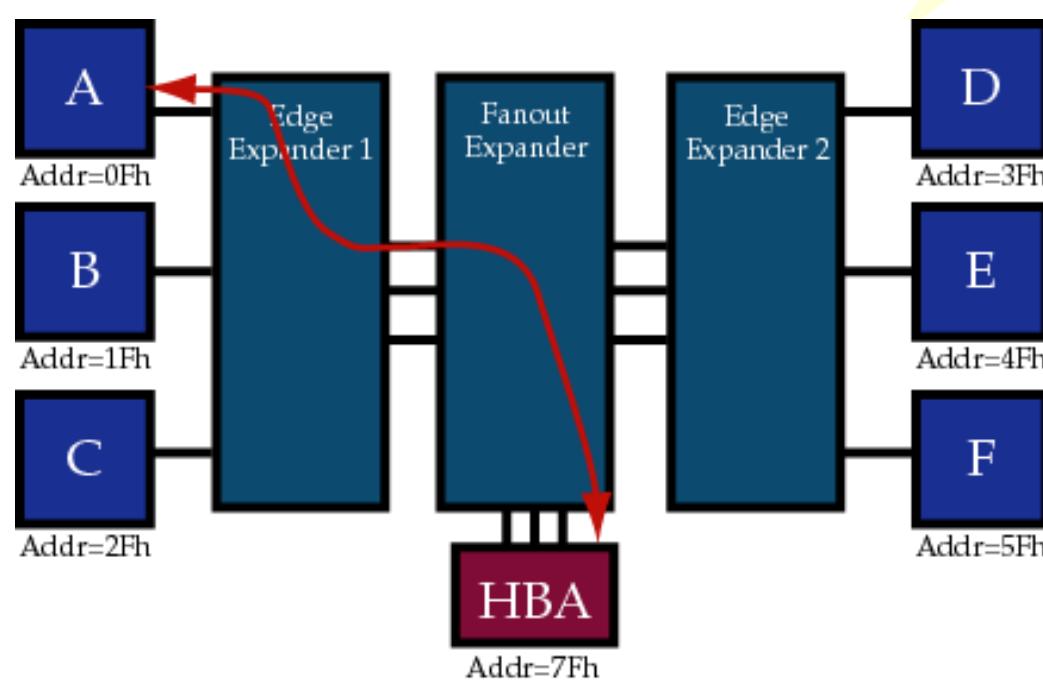
- Enclosed by SOAF and EOAF primitives
- Always 32 bytes long, including CRC
- Only two types:
  1. IDENTIFY frame
    - Sent by SAS phys after reset to exchange addresses with each other
  2. OPEN frame
    - Used to establish a new connection
    - Only sent when the phy is idle
    - Specifies source and destination SAS addresses for the desired path

# Connection

- A connection is a temporary association between two SAS phys
  - Request specifies the protocol to be used in this connection, and protocol cannot change during the connection
  - Transactions are addressed to SAS ports based on address, but connections are established between two phys
  - Wide ports can establish multiple connections at the same time
    - Normally, only HBAs and RAID controllers have wide ports; disk drives usually don't because they can't sustain the high data rates that would make them useful. And using two narrow ports removes a single-point-of-failure case, improving reliability.

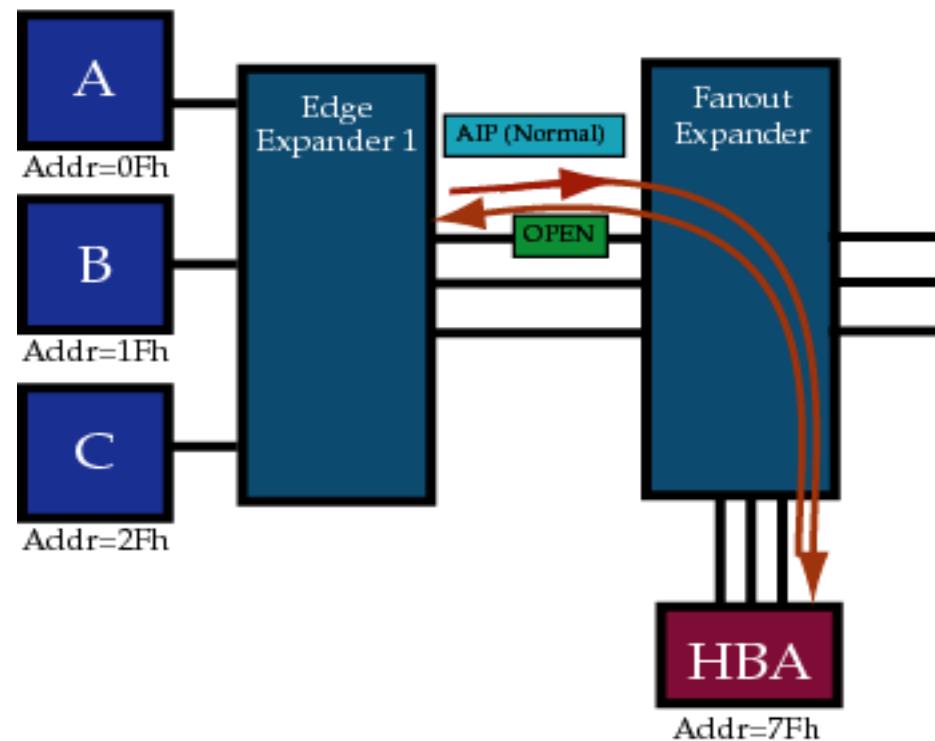
# SSP Connection Example (317)

- Goal: Establish a connection between the HBA and device A



# SSP Example (321)

- HBA begins by sending OPEN
  - Both Initiators and targets can open connections
  - Idle dwords follow while originator waits for response. It also starts the 1ms Open Timeout counter. A timeout will cause the requester to send BREAK and abort the request.



# OPEN Address Frame Format (316)

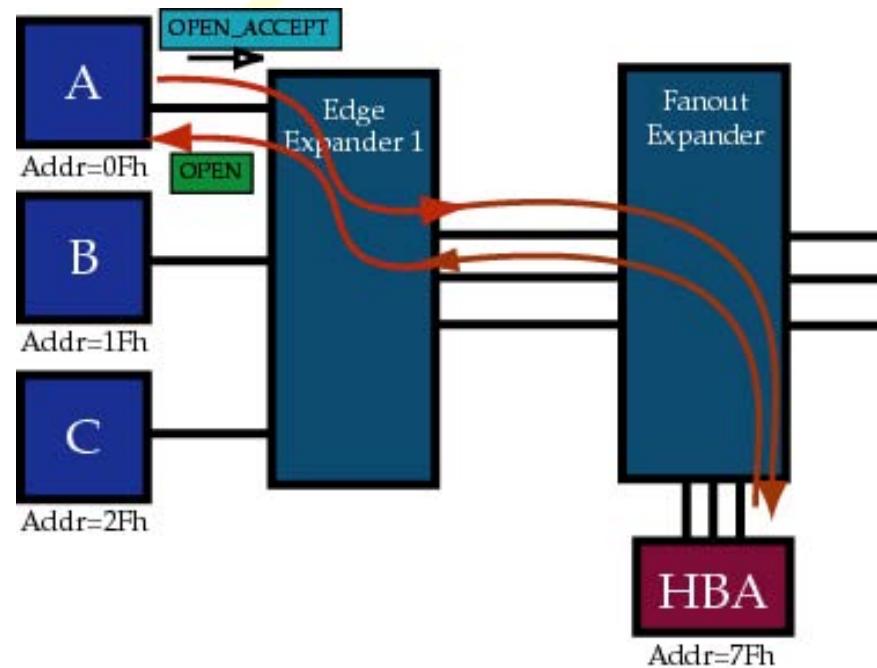
Byte	Fields			
0	Initiator Port	Protocol	Address Frame Type = 1	Indicates whether the source port is acting as initiator or target for this transfer
1	Features ( <u>= 0 for now</u> )		Connection Rate	SSP, STP, or SMP 1.5 or 3.0 Gbps
2 to 3			Initiator Connection Tag	Target can use tag when opening connection in reply to command, and initiator will be able to easily look up context. Init will see command tag later
4 to 11	<b>Destination SAS Address</b>			Where to route the frame
12 to 19	<b>Source SAS Address</b>			Originator's address for later replies
20	Compatible Features ( <u>= 0 for now</u> )			
21	Pathway Blocked Count			Number of times this access has been blocked, used for arbitration fairness
22 to 23	Arbitration Wait Time			
24 to 27	More Compatible Features ( <u>= 0 for now</u> )			Length of time this access has been waiting, used for arbitration fairness
28 to 31	CRC			

# Expander Responses (359)

- AIP (Normal) followed by AIP (Waiting on Device) returned by Expanders until request reaches the target.
  - Requester resets Open Timeout counter with receipt of each AIP
- OPEN\_REJECT – problem with the routing
  - Example: Bad destination – request would have to go back out the same port it came in on.

# SSP Connection (322)

- Target responds
  - If target accepts the request, a dedicated circuit is now open between the devices.
  - If target rejects the connection – two categories:
    - Abandon – give up the request. Caused by unsupported protocol or rate, bad destination, etc.
    - Retry – caused by temporary blockage in the domain



# Reject (Abandon – No Retry)

- Permanent Rejection Reasons:
  - Connection rate not supported
    - Rate is faster than physical link supports
  - Protocol not supported
    - Device doesn't support requested protocol or requested role (initiator/target), or initiator connection tag
  - STP resources busy
    - STP port already has an active affiliation with another initiator
  - Wrong destination
    - Destination address doesn't match the phy in an end device that receives the request

# Reject – Retry (323)

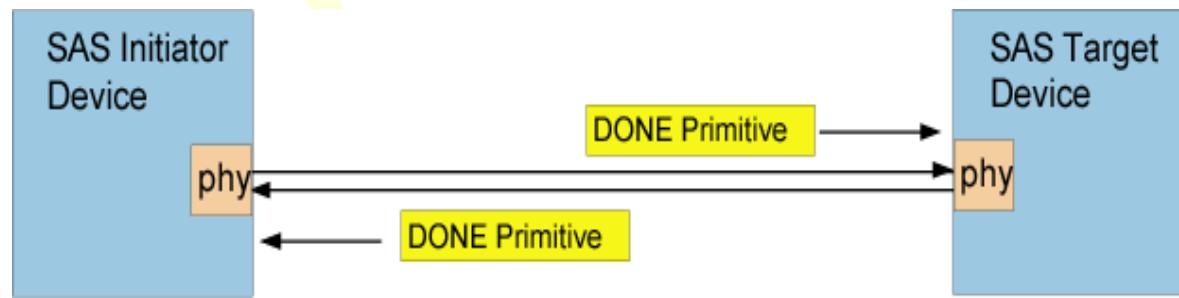
- Temporary Rejection Reasons:
  - No destination
    - No such destination address in the domain
    - Request arrives through a subtractive routing phy, but there is no hit in the device
    - Targets an STP/SATA bridge but SATA device has not yet sent initial register FIS
  - Pathway blocked
    - Partial pathway timer expired in an expander
  - Retry
    - Destination is busy, try again later. Destination must eventually accept the request.

# SSP Connection

- Once connection is established, devices indicates readiness to receive frames by sending RRDY. Each RRDY represents credit for one frame (up to 1KB data).
- Usually both requester and responder will send RRDY, to avoid throttling packets from the other device.

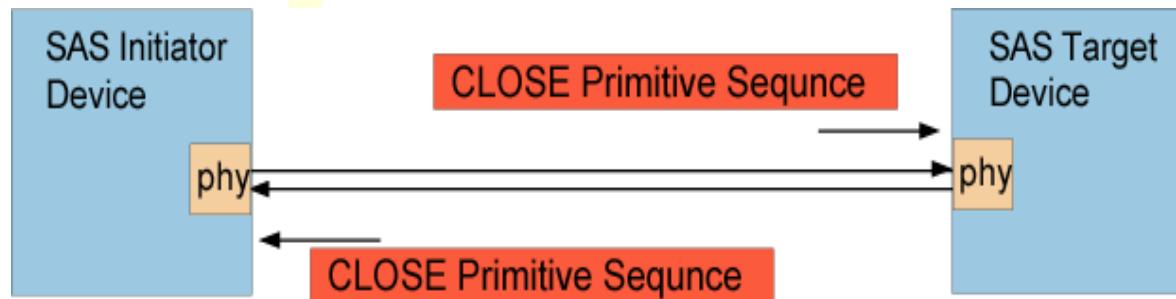
# Closing Connection

- Device indicates readiness to close connection by sending DONE when it has no more frames to send
- DONE is a commitment - no more frames from this device.
- Traffic in the other direction is unaffected
- Although device cannot send more frames, it must still respond with ACK, NAK, etc., until neighbor responds with DONE



# Closing (cont)

- When a phy has sent and received DONE, it sends the CLOSE primitive sequence (send 3, receive 3), waits for CLOSE in response
- Timeout – after sending DONE, if the other device wants to send frames but can't for 1ms, it has to return DONE. It has to actively use the connection or close it.



# SMP Connection

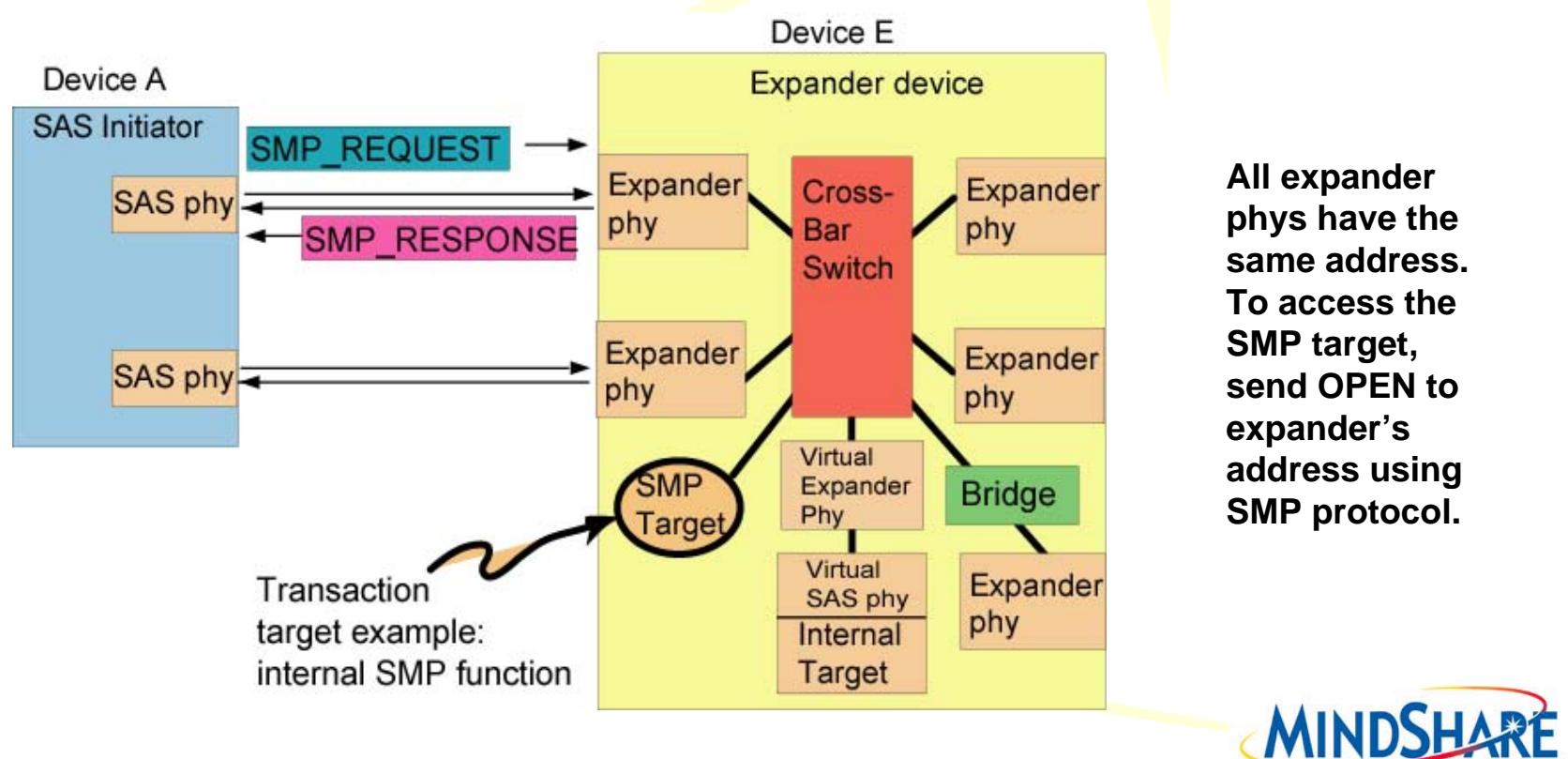
- Purpose: Access control and status registers of a device
- Only initiators can open SMP connection
- Expanders must have an SMP target, but it's optional for other devices to implement because it adds cost
- Access is half-duplex (not queued)

# SMP is a Simple Protocol

- No ACK or NAK
- No flow control (no RRDY) – target accepting SMP connection implicitly grants “credit” for the one SMP request
- Initiator always sends one request, gets one response, and closes the connection

# SMP Connection

- After OPEN accept, only two frames are sent: request from initiator and response from target, followed by CLOSE
- SMP target can be used to query every Phy in the expander and learn what addresses are attached to them (Discovery process).

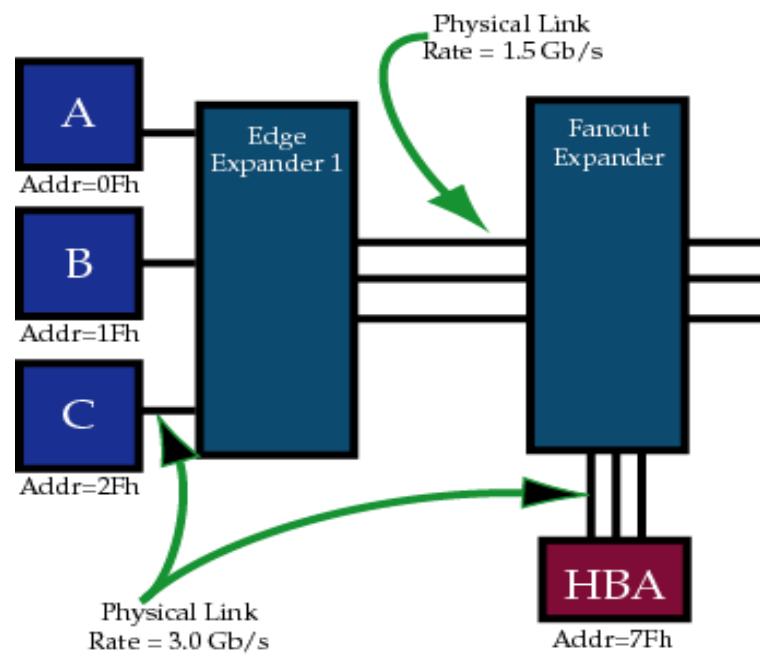


# Rate Matching

- Successful connection contains the connection rate of the pathway.
- If a Phy's rate is faster than the connection rate established by the connection, it must inject ALIGN or NOTIFY primitives to accommodate the slower links.
- Example would be a phy operating at 3.0 Gb/s in a connection running at 1.5 Gb/s.
  - Insert 1 ALIGN or NOTIFY within every 2 dwords in addition to any clock management primitives.

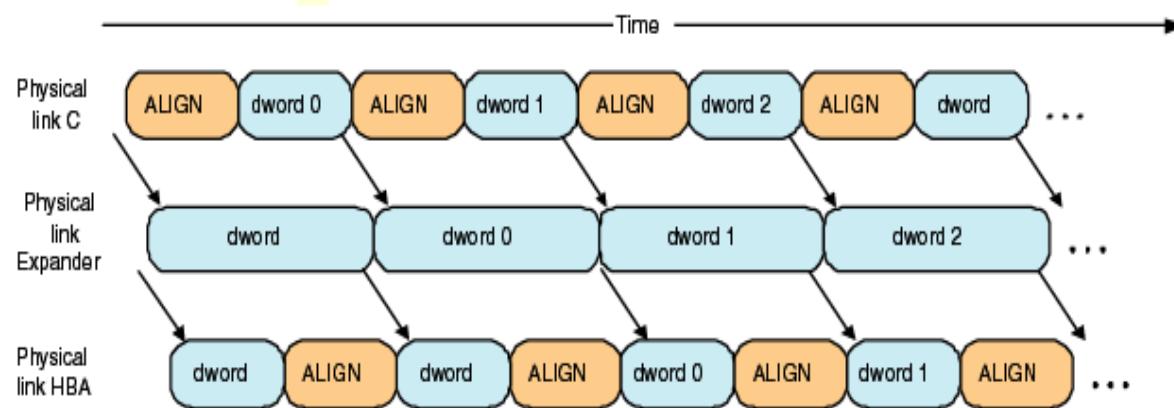
# Rate Matching Example

- Connection from HBA to C has rate of 1.5 Gb/s because intermediate link cannot support faster rate.
- Faster links will have to match the rate



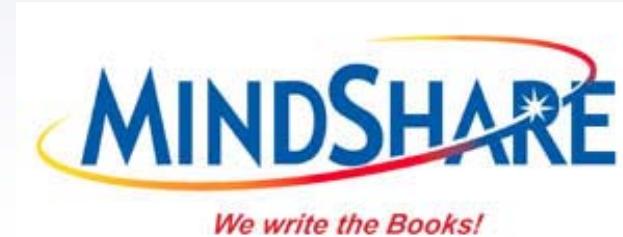
# Injecting Additional Primitives

- Slower link can only see every other dword sent by faster links, so fast ones must inject throw-away primitives.
- Faster transmitters add them, slower ones remove the extras
- Target receiver ignores any extra primitives.



# *Chapter 14*

## *Link Layer – Arbitration*

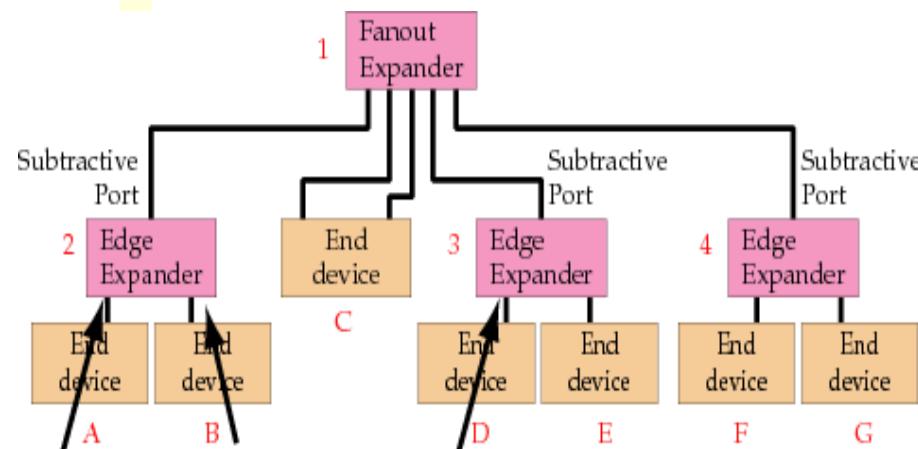


# Arbitration Fairness (334)

- Purpose: prevent starvation of devices
  - Select among packets competing for access to the same resources
  - Fairness factors:
    - How long the transaction has already been waiting to get access
    - How many times the transaction has previously been blocked from accessing the destination

# Arbitration Example (335)

- Assume devices A and B want to access D, while D wants to access A.
- All 3 send OPEN frames. All receive AIP(Normal) from expanders.
- Expander 2 doesn't find either destination address in list of direct connections, and routes both to its subtractive port.
- Only one link is available, so arbitration is necessary.  
Normal order of evaluation:  
AWT, PBC, Source Address, connection rate

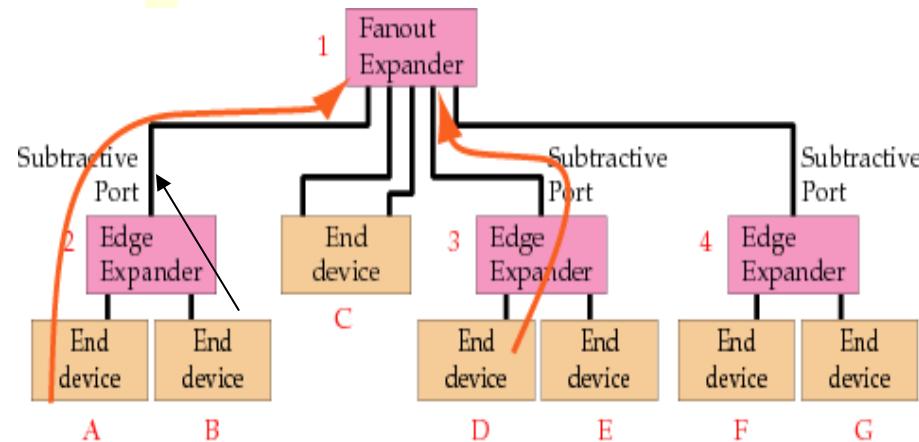


# AWT

- Arbitration Wait Time, the total time an OPEN has been waiting to get to its destination, is normally set to 0 by initiator. Each expander updates this field as it is forwarded by adding the time it took the frame to get through the expander.
- Initiators are allowed to cheat on this by setting the initial AWT value as high as 7Fh. This permits a system administrator to assign arbitration priority to some domains or devices.

# Pathway Blocked

- Assume ECM (Expander Connection Mgr) of expander 2 determines Device A to be the winner. As long as request A is making progress, then all is well and request B simply waits forever.
- If expander 2 should see AIP(WAITING ON PARTIAL) returned, then the partial timeout counter would come into play. Timeout would cause expander to send B an OPEN REJECT (PATHWAY BLOCKED).
- In that case, B would increment PBC (Pathway Blocked Count) field for next time and try again later.



# Partial Pathway Timeout timer (335)

- Programmable initial value per expander phy
  - reported via SMP DISCOVER function
  - set via SMP PHY CONTROL function
  - Range: 0 to 15  $\mu$ s
  - Recommended value: **7  $\mu$ s**
  - Want to provide enough time for the winning connection request to establish a connection
- Timer starts if connection request cannot get to any output phy and one of those phys is in a partial pathway state

# Pathway Recovery priority

- When timer expires, reject all but the highest request waiting to access that output phy, using OPEN\_REJECT (PATHWAY BLOCKED)
  - This process is called **pathway recovery**
- To determine priority, competing requests are compared:

MSB		LSB
Pathway Blocked Count	Source SAS address	Connection rate

- Pathway Blocked Count
  - Higher value gives blocked request higher priority for the next try
- Reject all but the highest priority request (v1.1)
  - That frees up the rest, since request that was waiting on it can now proceed

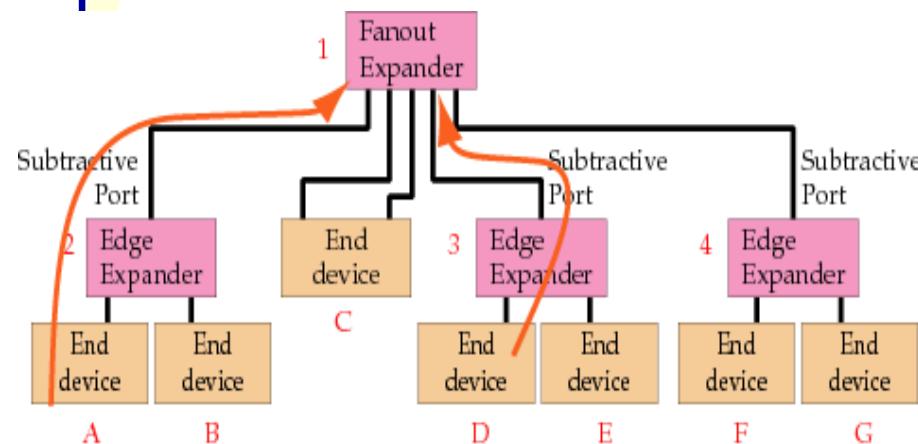
# Continue Arbitration Example

- Continuing the illustration, let's consider three possible cases :
  - Backoff-Reverse Path
  - Crossing Requests
  - Backoff-Retry

# Backoff-Reverse Path (336)

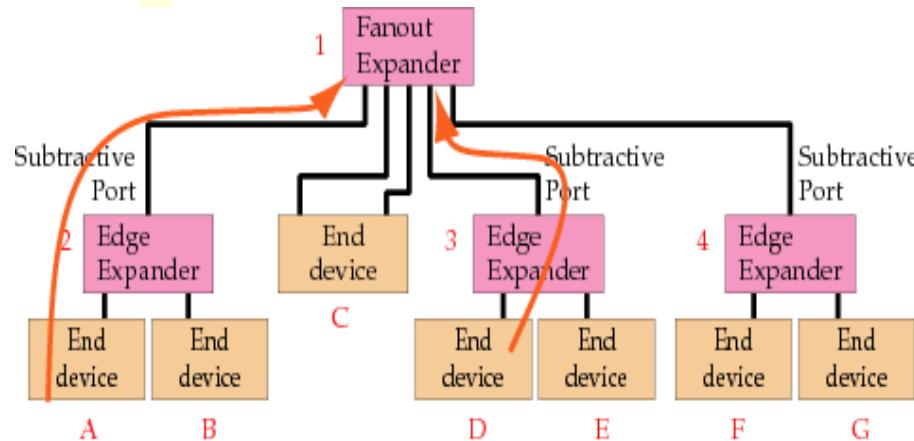
- Requests from A and D arrive at expander 1 at the same time.
- In this example, the requests are attempting to pass each other in the expander. ECM evaluates them based on AWT and source address:
- Assume request A wins

Bits 79-64	Bits 63-0
AWT	Source Address



# Backoff-Reverse Path

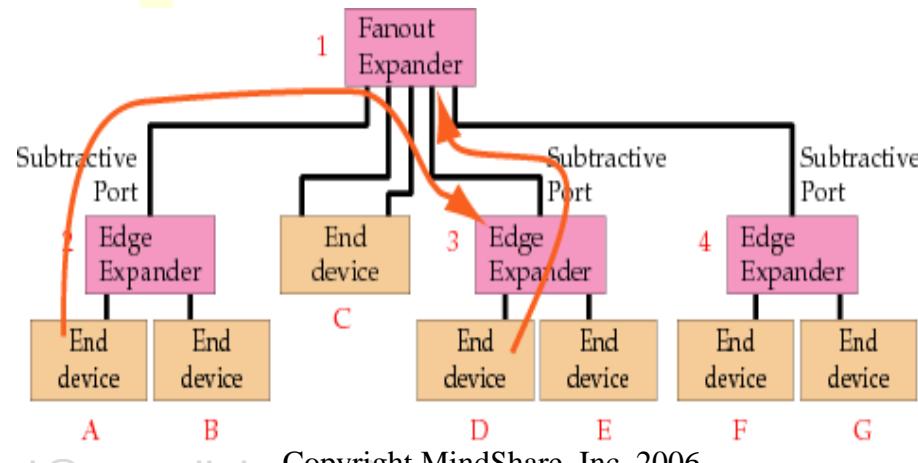
- Fanout expander forwards request A to expander 3, which recognizes that request from D must have lost the arbitration, because OPEN is received after AIP.
- For this case, the OPEN from A is now guaranteed to win the arbitration to its destination, because the request from that phy had already won arbitration previously.
- Next, D recognizes that it lost the arbitration for the same reason (OPEN received after AIP already received).
- Finally, the OPEN from A reaches D, which responds with OPEN\_ACCEPT or OPEN\_REJECT



# Crossing Requests (337)

- Request from A and request from D cross on the wire between expander 1 and 3.
- Both expanders (1 & 3) compare the contents of the OPENs based on the same fields:
- They will reach the same conclusion – let's assume the open from A wins
- Expander 3 forwards incoming OPEN from A, and D recognizes that it lost the arbitration.

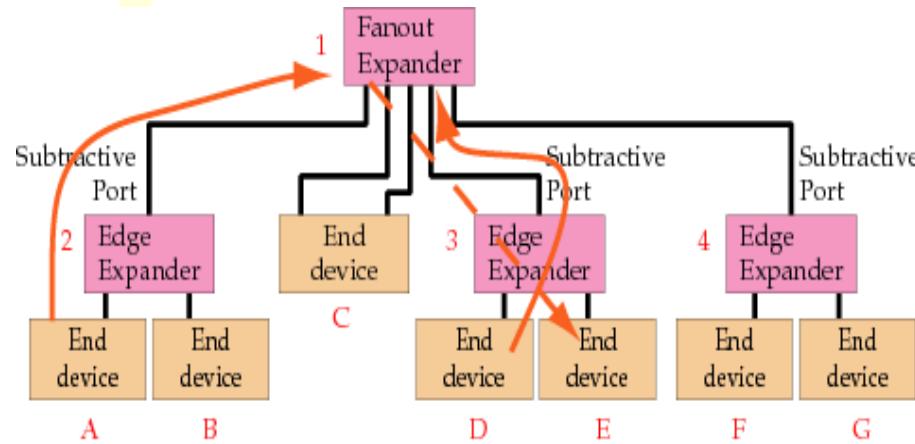
Bits 79-64	Bits 63-0
AWT	Source Address



Copyright MindShare, Inc. 2006

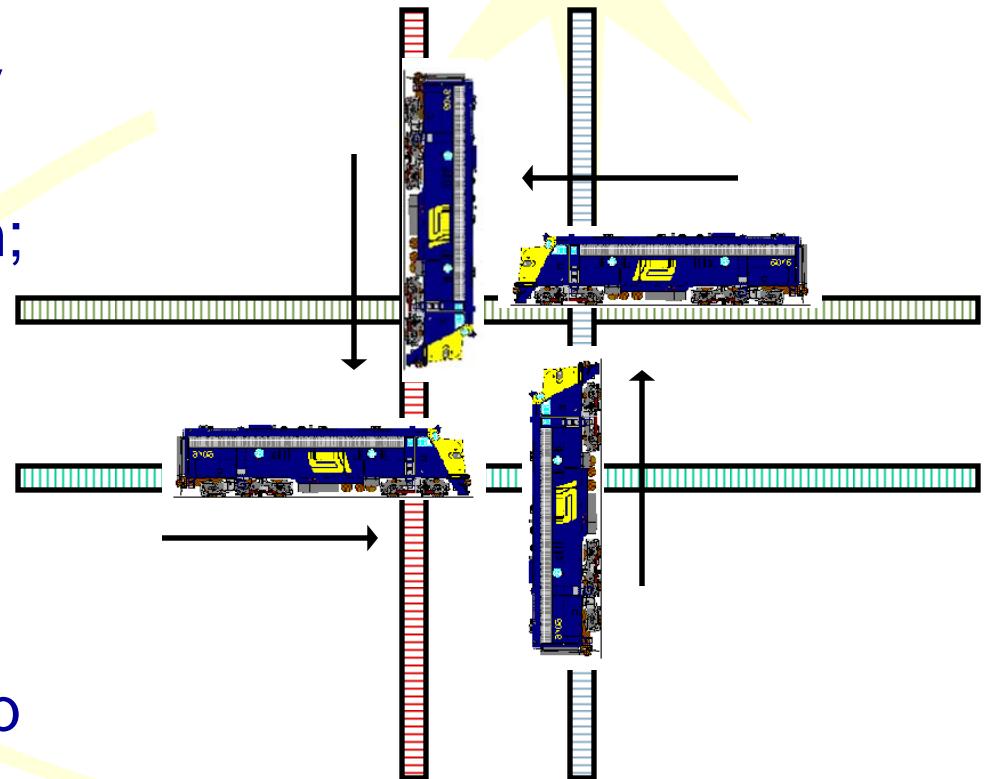
# Backoff-Retry (338)

- Request from A, targeting E this time, and request from D, again targeting A arrive at expander 1 together. Assume A wins again.
- This time, though, there's no guarantee the open from A will win arbitration in expander 3 because a new request from E could have a higher priority.
- Ping-pong between expanders is possible (AIP followed by OPEN one way, then AIP followed by OPEN the other way, then the first way again, etc.)



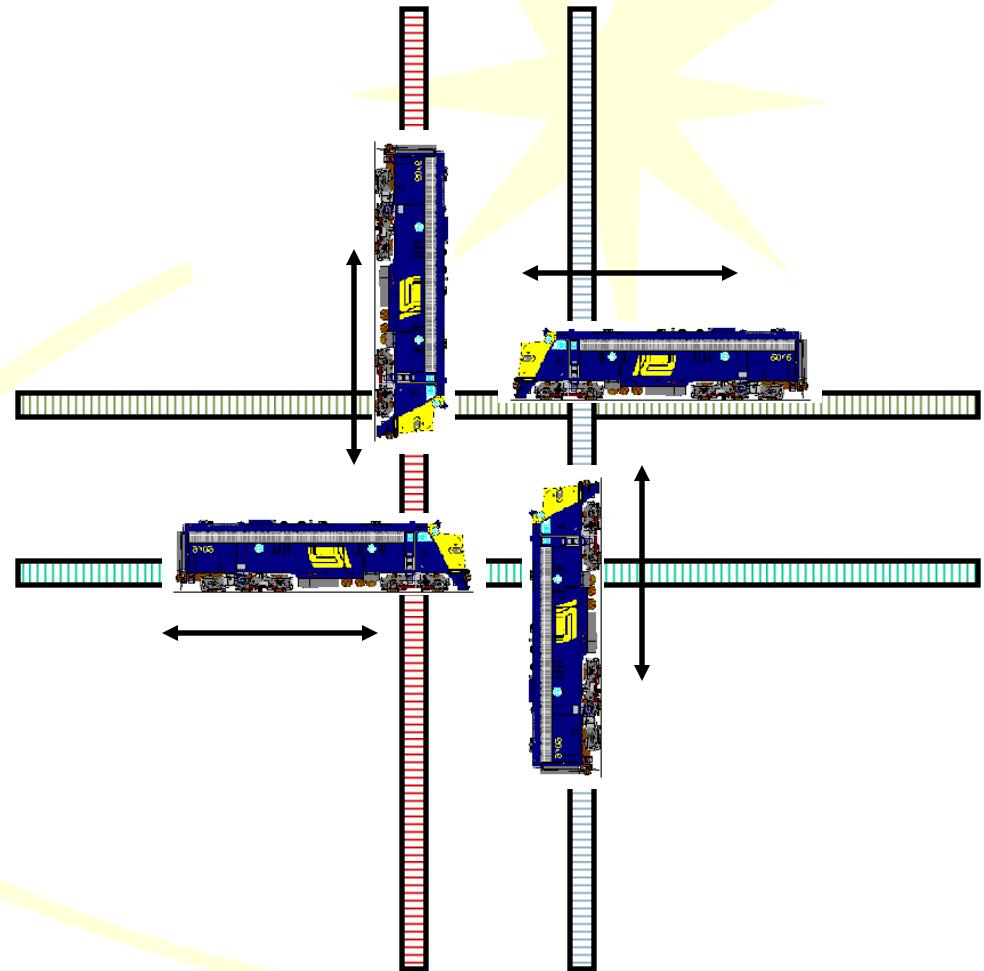
# Deadlock Illustration (339)

- Two or more processes waiting on each other to complete
- Assume trains can only go forward and are longer than intersection; result is deadlock
- For SAS, trains represent connection requests and resulting connections occupying physical links in a group of expanders



# Livelock Illustration (339)

- Two or more processes changing state in response to changes in other processes
- Illustration: trains detect problem at the same time, back up same distance, go forward again at same speed
- Links are busy, but no progress is made



# Conditions for deadlock (339)

Four conditions must be simultaneously true for deadlock to exist

## 1. Mutual exclusion

- resource not shareable by two requesters
- SAS: a physical link

## 2. Hold and wait

- requester holds one resource while requesting another
- SAS: initiator-to-expander physical link is occupied while the expander requests an expander-to-target physical link

# Conditions for Deadlock

## 3. No preemption

- no way to override another requester
- SAS: possible in a circular wait situation

## 4. Circular wait

- One requester waiting on a second device, which is waiting on a third device, which in turn is waiting on the first one.
- SAS: possible with multiple expanders (multiple physical links)

# Resolving Deadlock (339)

- Two approaches to solving deadlock issues
  - Prevent and avoid
    - Ensure the 4 conditions can never be true at the same time
  - Detect and recover
    - Detect that all 4 conditions have occurred and fix it
- Deadlock resources
  - Operating System Concepts textbook by Silberschatz, Peterson, and Galvin
  - <http://www.cs.cornell.edu/courses/cs414/2001SP/lectures/9-deadlock.pdf>  
(class notes using that textbook)

# Deadlock prevention

- Prevention: Avoid one of the 4 conditions
  - Mutual exclusion
    - Allow sharing of the resources
    - Problem: SAS physical links only allow one connection at a time
  - Hold and wait
    - Ensure that requesters request/release all resources at once
    - Problem: SAS arbitrates for one physical link at a time; waiting for all physical links to be available would hurt performance and be difficult to implement

# Deadlock prevention - part 2

## – Preemption

- Allow resources to be taken over
- Problem: SAS connections and connection requests might be overridden; could create starvation
- Problem: cannot take over the proper resources when they are in other expanders (circular wait)

## – Circular wait

- Impose a global ordering and ensure all requesters request resources in order
- Problem: This can end up starving ports with lower priority SAS addresses. Counter to arbitration fairness.

# Deadlock Recovery (340)

- SAS implements deadlock detection and recovery
- Ideal approach
  - Examine every possible allocation sequence for incomplete requests and determine if they can complete
  - Either abort all requests or abort one at a time until the deadlock cycle is eliminated
  - Impractical in a distributed arbitration environment
- SAS approach
  - Expanders implement Partial Pathway Timeout timers to detect suspicious connection requests and force them to back off

# Detection in Expander

- States of a connection request in expander
  - Waiting on full connection – no problem, keep waiting
    - Connections are assumed to be short-lived so this shouldn't be a problem
    - SAS port can't leave connection open forever or it will cause problems
  - Waiting on blocked partial pathway – potential problem: run the timer
    - Partial pathways waiting on partial pathways is the deadlock scenario

# AIP primitives (341)

- AIPs communicate arbitration state from expanders
  - **AIP (NORMAL)**
    - Accepted the OPEN address frame
    - Started internal arbitration for an output phy
  - **AIP (WAITING ON DEVICE)**
    - Forwarded the OPEN address frame; waiting for a reply

# AIP Primitives (341)

## – AIP (WAITING ON CONNECTION)

- Waiting on an output phy currently involved in a connection
- That connection must eventually close

## – AIP (WAITING ON PARTIAL)

- Waiting on a partial pathway
  - Waiting on a phy during internal arbitration [a Partial Pathway]
  - Waiting on a phy that hasn't yet received AIP
  - Waiting on a phy which has passed through AIP (WAITING ON PARTIAL) [a Blocked Partial Pathway]

# OPEN Frame Arbitration Fields

Byte	Fields		
0	Initiator Port	Protocol	Address Frame Type = 1
1	Features (= 0 for now)		Connection Rate
2 to 3	Initiator Connection Tag		
4 to 11	<b>Destination SAS Address</b>		
12 to 19	<b>Source SAS Address</b>		
20	Compatible Features (= 0 for now)		
21	Pathway Blocked Count		
22 to 23	Arbitration Wait Time		
24 to 27	More Compatible Features (= 0 for now)		
28 to 31	CRC		

**Fields used for arbitration**  
The least significant part in arbitration priority

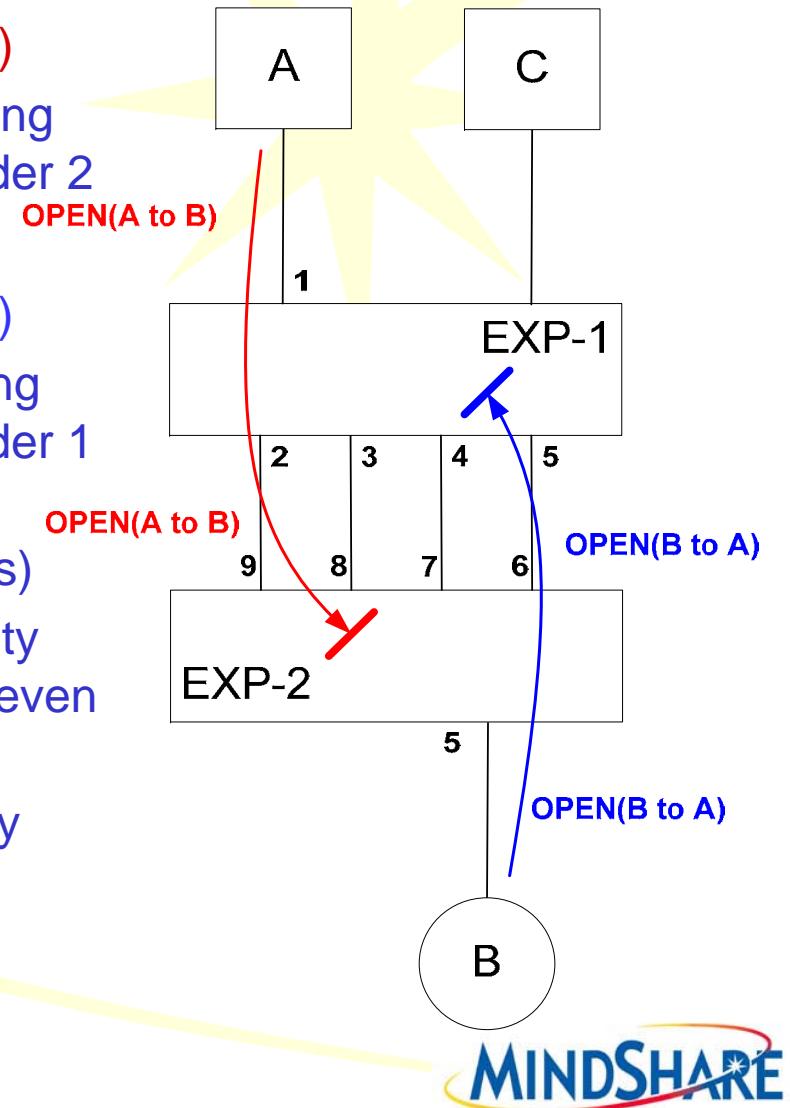
Source address is next most significant

Next highest significance - Number of times this access has been blocked

Most significant: Length of time this access has been waiting

# Deadlock example (342)

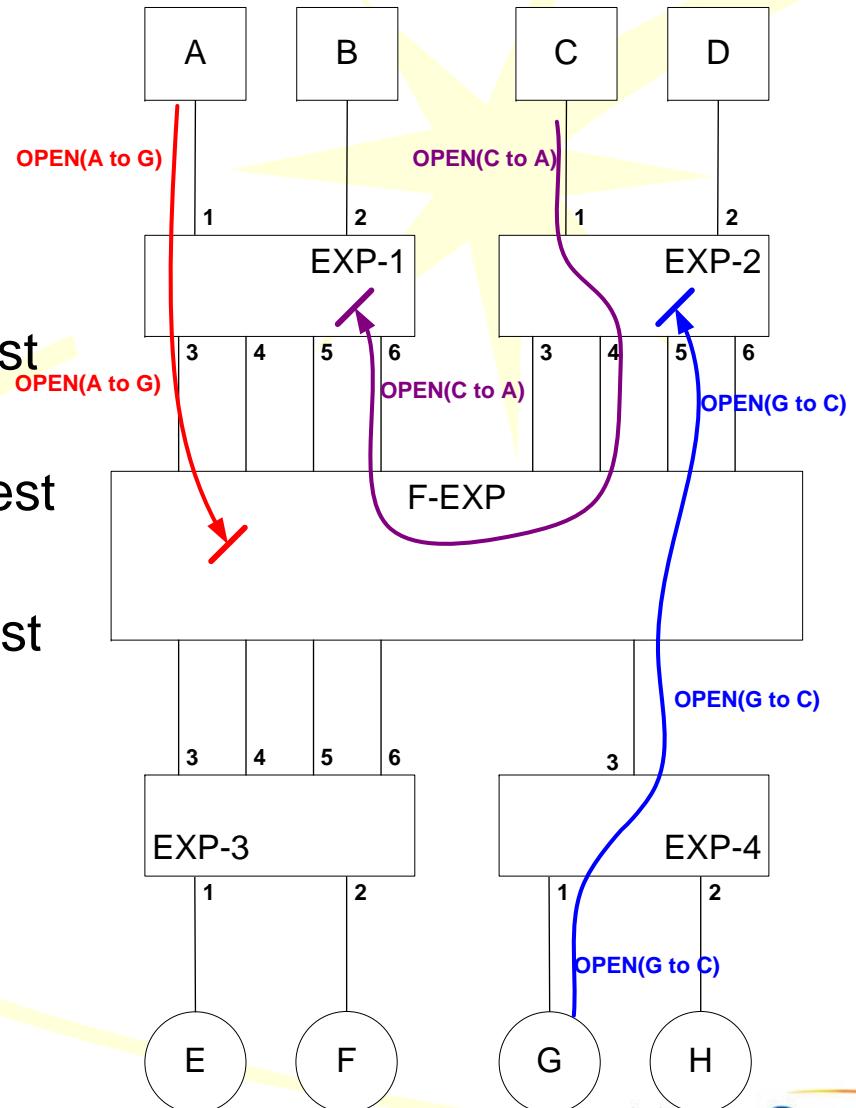
- In expander 1
  - OPEN (B to A) is waiting on OPEN (A to B)
  - OPEN (A to B) is a partial pathway, receiving AIP (WAITING ON PARTIAL) from Expander 2
- In expander 2
  - OPEN (A to B) is waiting on OPEN (B to A)
  - OPEN (B to A) is a partial pathway receiving AIP (WAITING ON PARTIAL) from Expander 1
- Exp-1 Phy 5 and Exp-2 Phy 9 Partial Pathway  
Timeout timers expire (maybe at different times)
- Exp-1 Phy 5 sees that source B is higher priority than source A, so it waits, and is not rejected, even though timeout occurred
- Exp-2 Phy 9 sees that source A is lower priority than source B, so it returns OPEN\_REJECT (PATHWAY BLOCKED)
- Then OPEN (B to A) gets through



# Circular wait deadlock example

## Example 1

- A: PBC = 3, address = 5
- G: PBC = 3, address = 3
- C: PBC = 0, address = 1
- F-exp: A vs. G, A wins so A's request is not rejected
- Exp-2: G vs. C, G wins so G's request is not rejected
- Exp-1: C vs. A, A wins so C's request is rejected



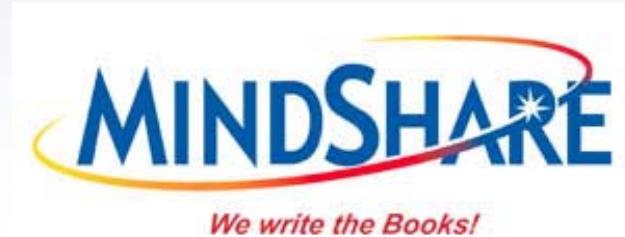
## Example 2

- A: PBC=3, address = 5
- G: PBC=4, address = 3
- C: PBC=0, address = 1
- F-exp: A vs. G, G wins so reject A

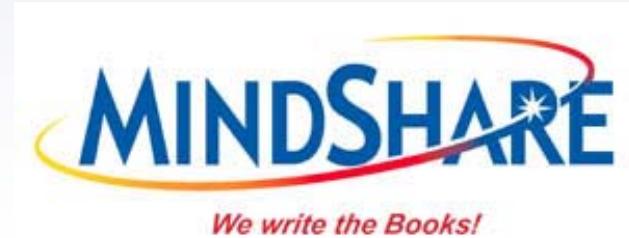
# *Chapter 15*

## *Link Layer –*

## *Protocol Differences*

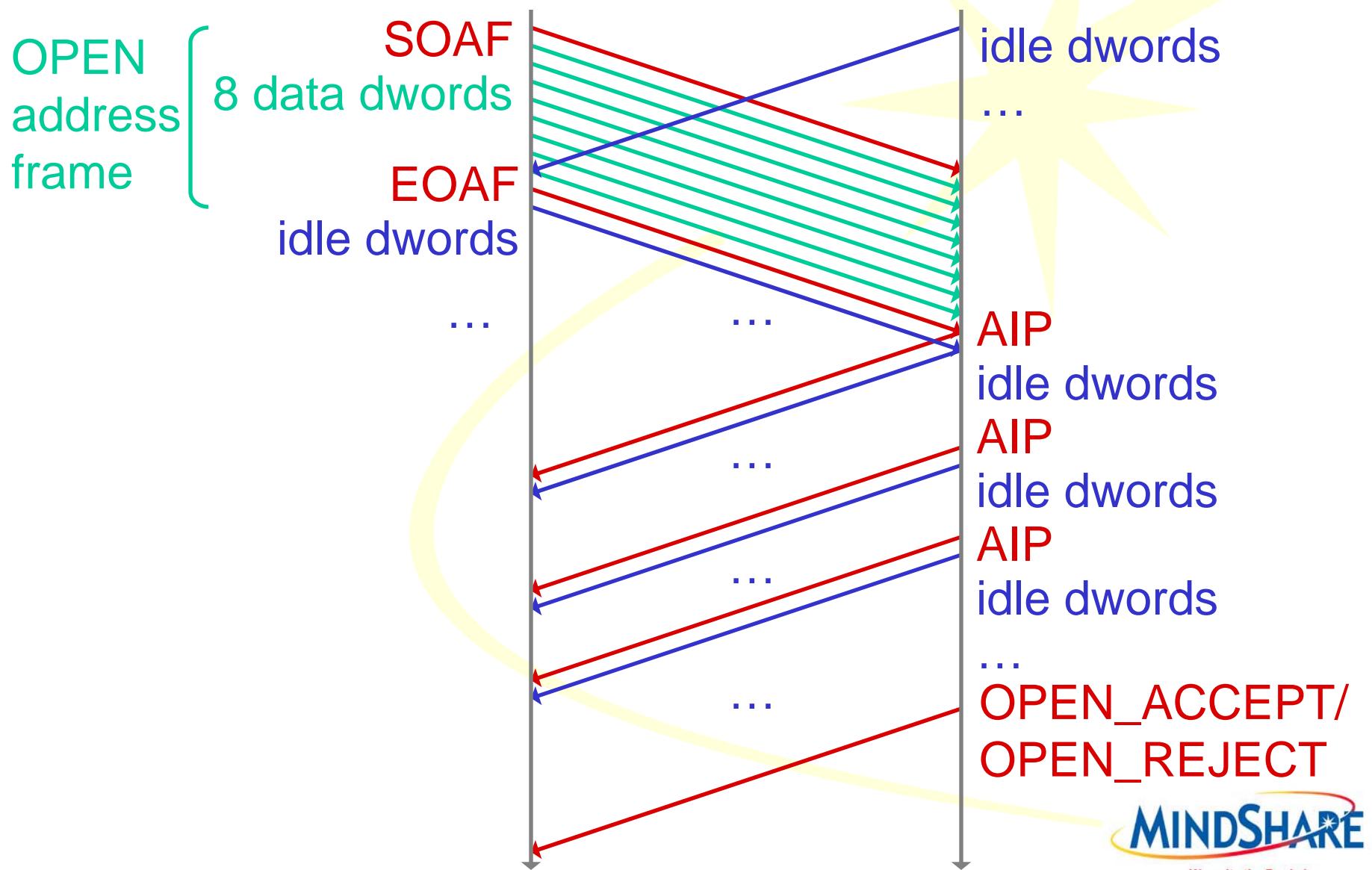


# *SSP Connections*



Christy Choi(christy.choi@ sandisk.com)  
Do Not Distribute

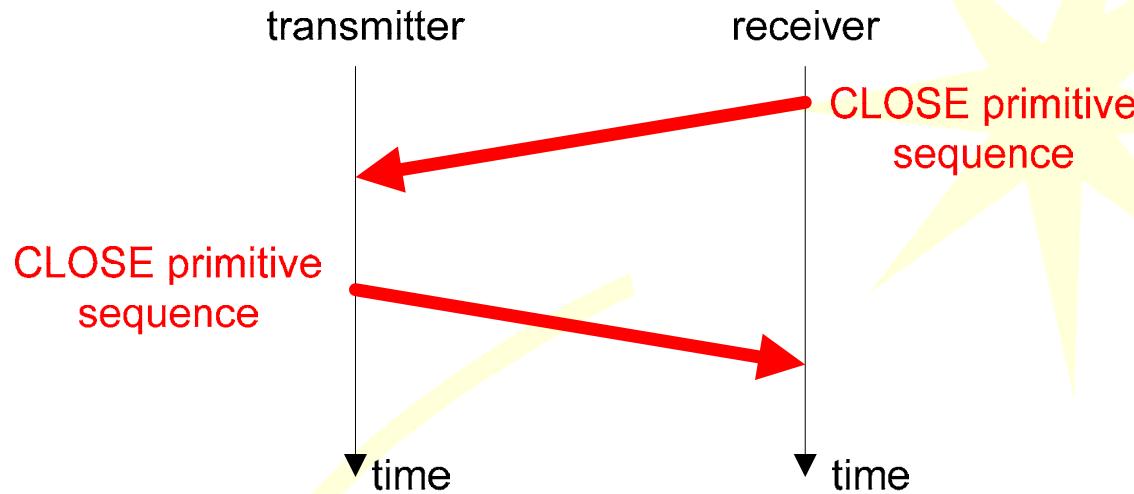
# Opening a Connection



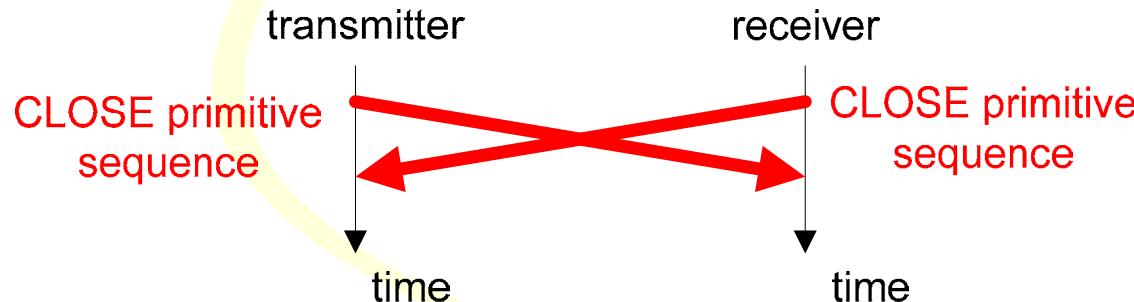
Copyright MindShare, Inc. 2008

# Closing a Connection

Example 1: CLOSEs sent one at a time



Example 2: CLOSEs sent simultaneously



- For SSP protocol, DONE must be sent before CLOSE

# Within an SSP Connection

- Phys exchange SSP frames



- Link Layer Responsibilities:
  - Flow control – RRDY grants permission to send one frame
  - Acknowledging received frames with Ack/Nak
  - Observing incoming Ack/Naks in response to frames sent

# SSP Flow Control (346)

- Credit-based flow control
  - Once a connection is open, *permission to send frames* is granted by sending RRDY primitives
    - RRDY guarantees that device has at least 1KB buffer space, since every frame might contain that much data
    - RRDYs can be queued up, and transmitter can send as many frames as the number of RRDYs received
  - Readiness to *receive write data* is indicated with XFER\_RDY frame, which specifies available buffer size but doesn't give permission to send

# Flow Control Considerations

- Rule: A phy that accepts an OPEN must provide at least one credit. If unable to do so, it must reject the OPEN request.
- RRDY will be sent the other way, too, in case frames going the other direction will also be handled in this connection.
  - Example: initiator can choose not to send RRDY after a request if it expects the target to open again later when ready with data. But it could happen that the target might have data ready from a previous command, or even for this request if the request turns out to be a cache hit in the drive, so sending at least one RRDY is a good practice.

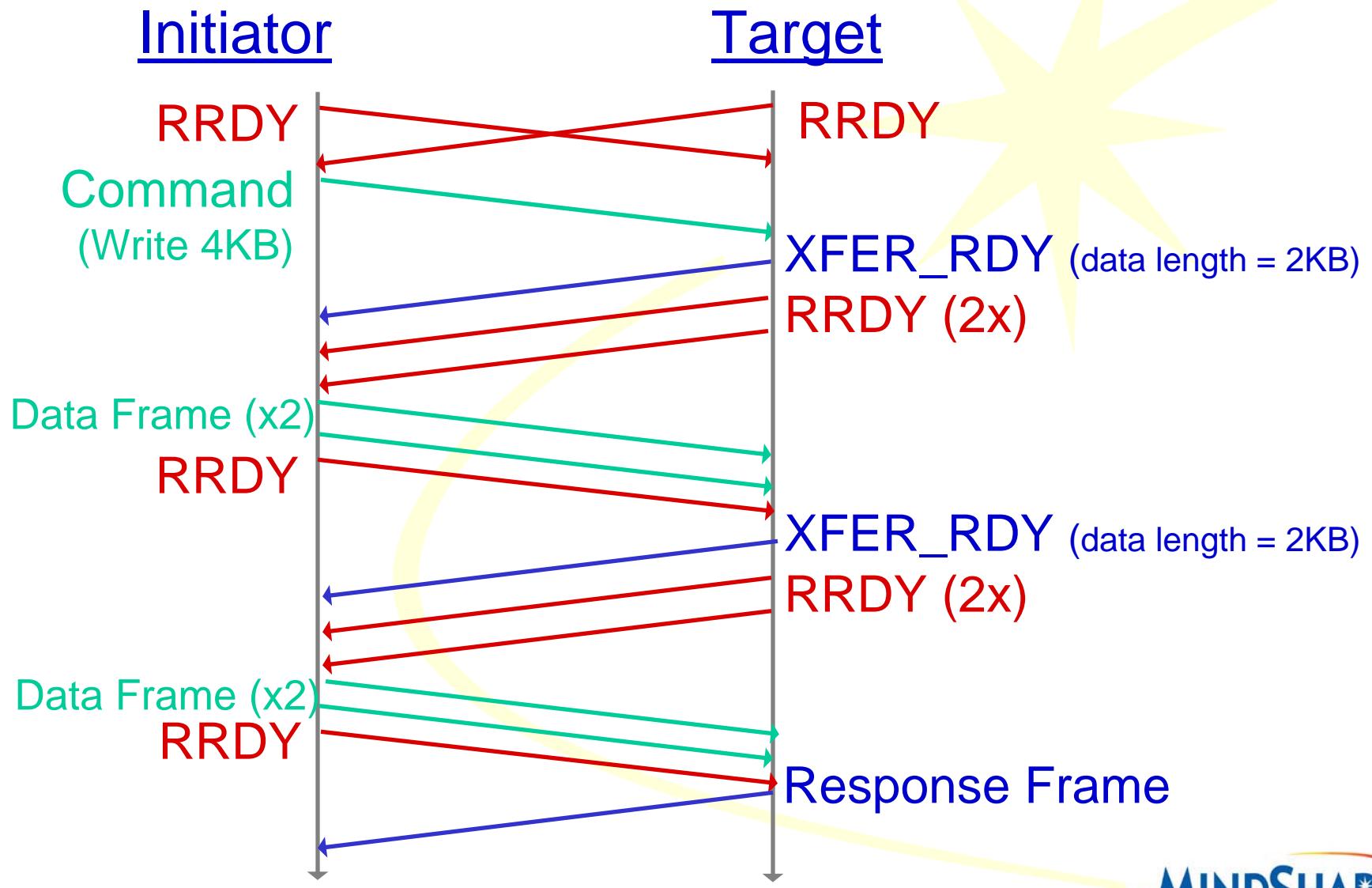
# Credits

- At start of connection, credit is zero.
- Each RRDY primitive received increments credit by one (up to 255 max) and indicates buffer space for one frame (up to 1KB of data + header).
- Each frame sent decrements available credit by one.

# Write Example

- Initiator wishes to send 4KB data, target can accept 2KB at a time
- After connection opened, devices exchange RRDY to indicate readiness to receive a packet
- Initiator sends command, waits for XFER\_RDY to tell it how much buffer space target has available, and RRDYs to indicate readiness for more frames

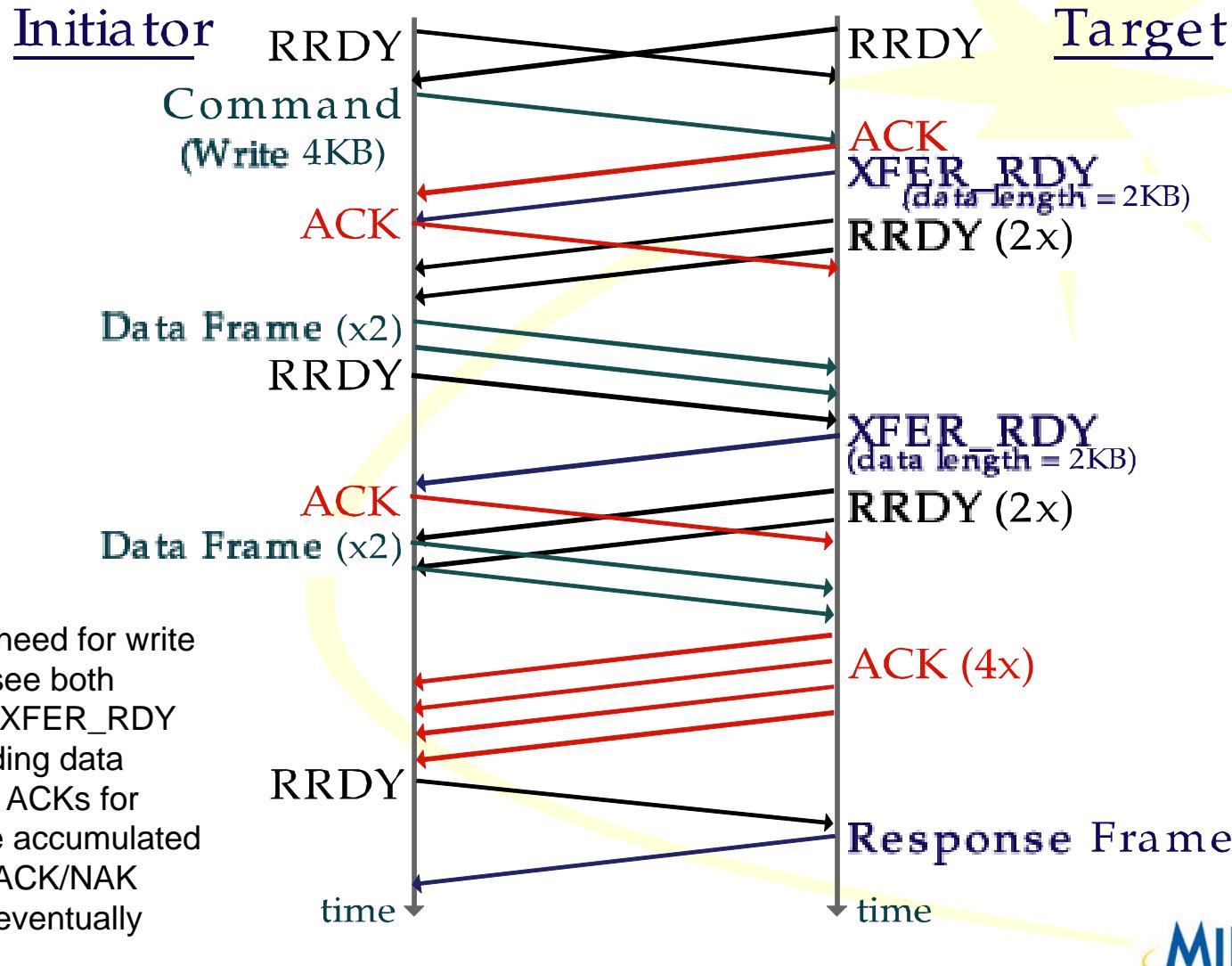
# Flow Control Example (348)



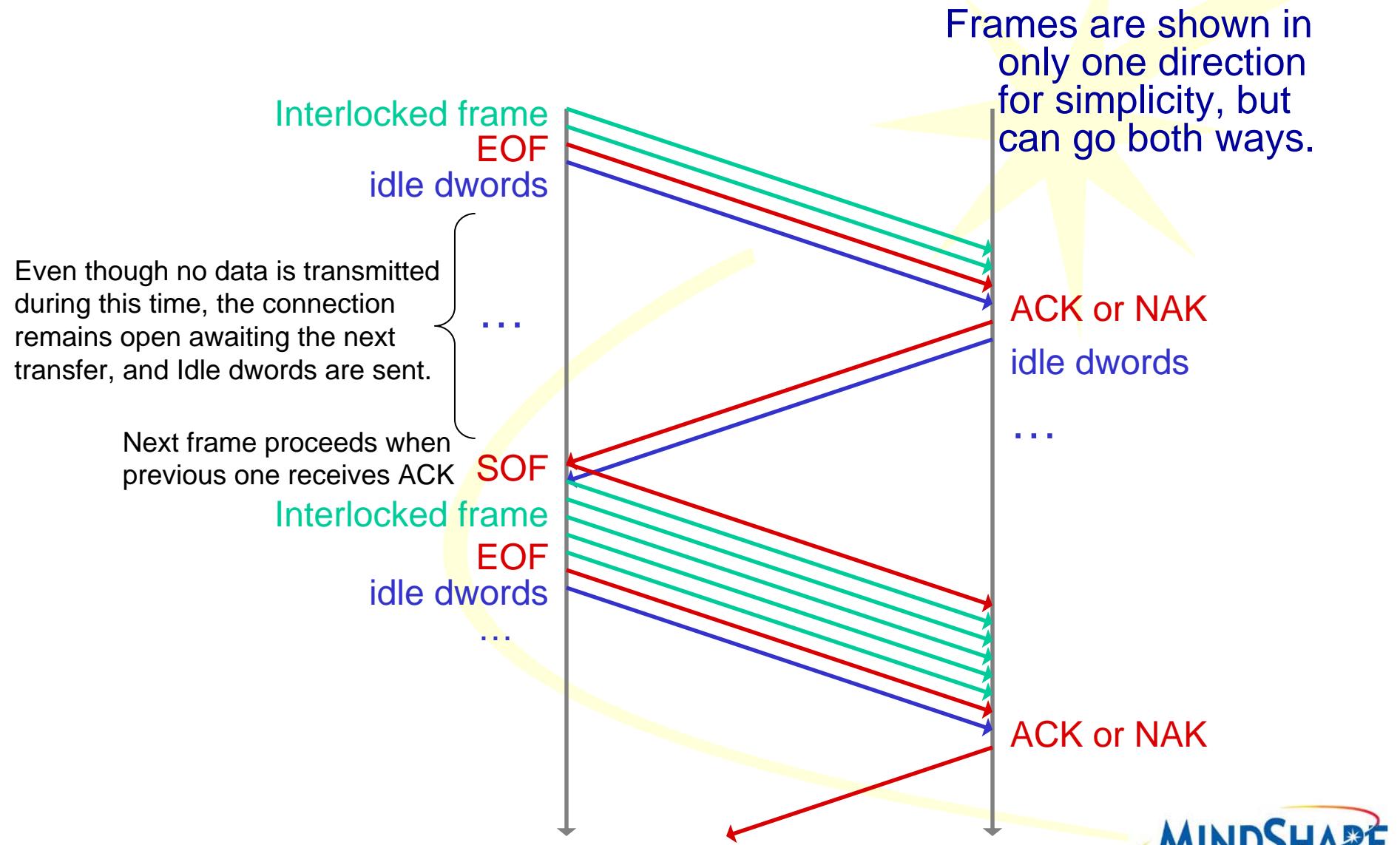
# Ack/Nak Considerations

- After sending most frames the transmitter must wait for the Ack/Nak handshake before sending another frame. These are called “interlocked” frames.
- The exception to this is Data frames, which can be non-interlocked and don’t have to wait for Ack/Nak before sending the next one.
  - Instead, Ack/Nak responses are counted
  - Total Acks must eventually match the total frames sent, referred to as “Ack/Nak Balance”
  - Error handling may consist of going back to the last time Ack/Nak balance was attained and retransmitting data frames from that point with the “retransmit” bit set.

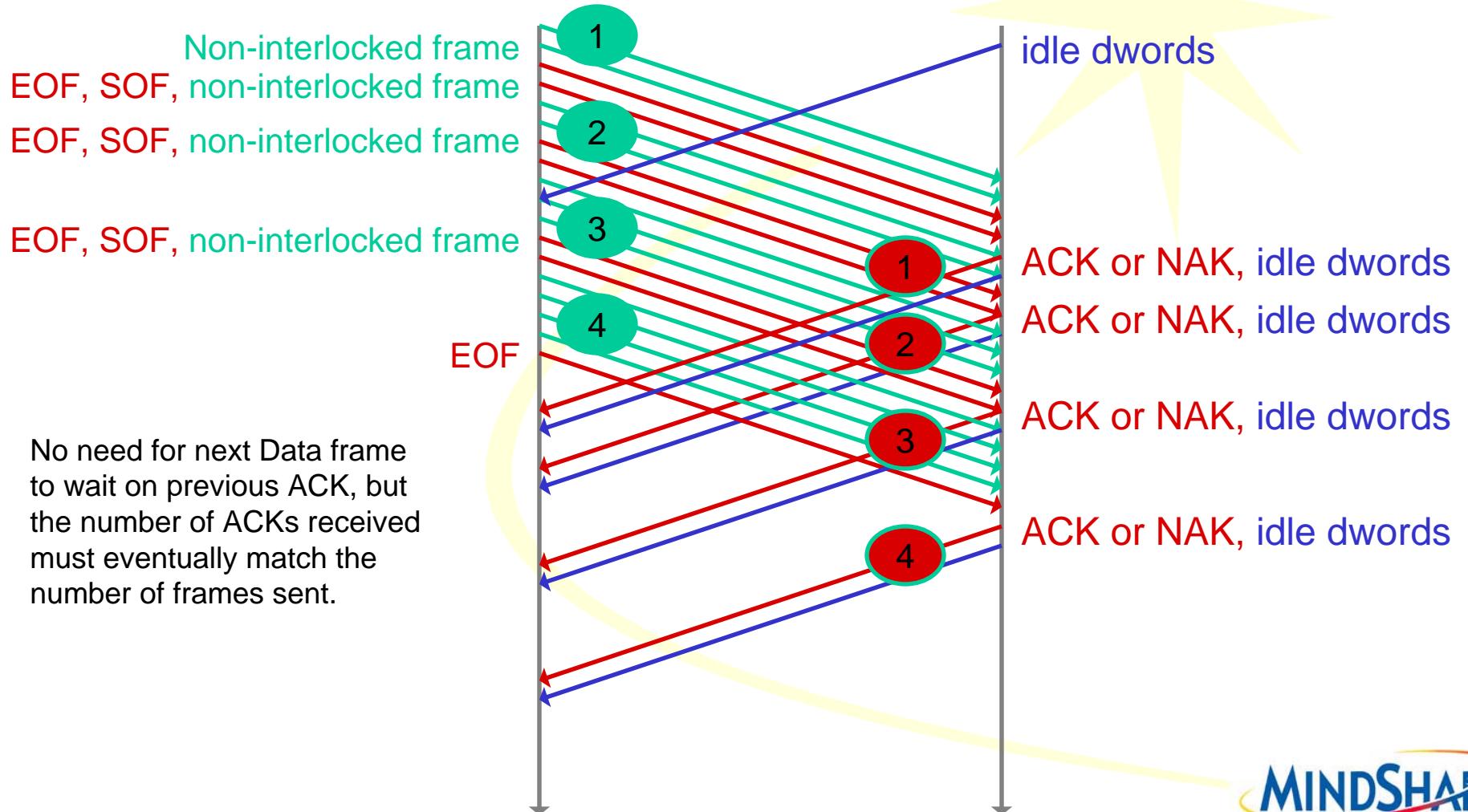
# ACK/NAK Example (352)



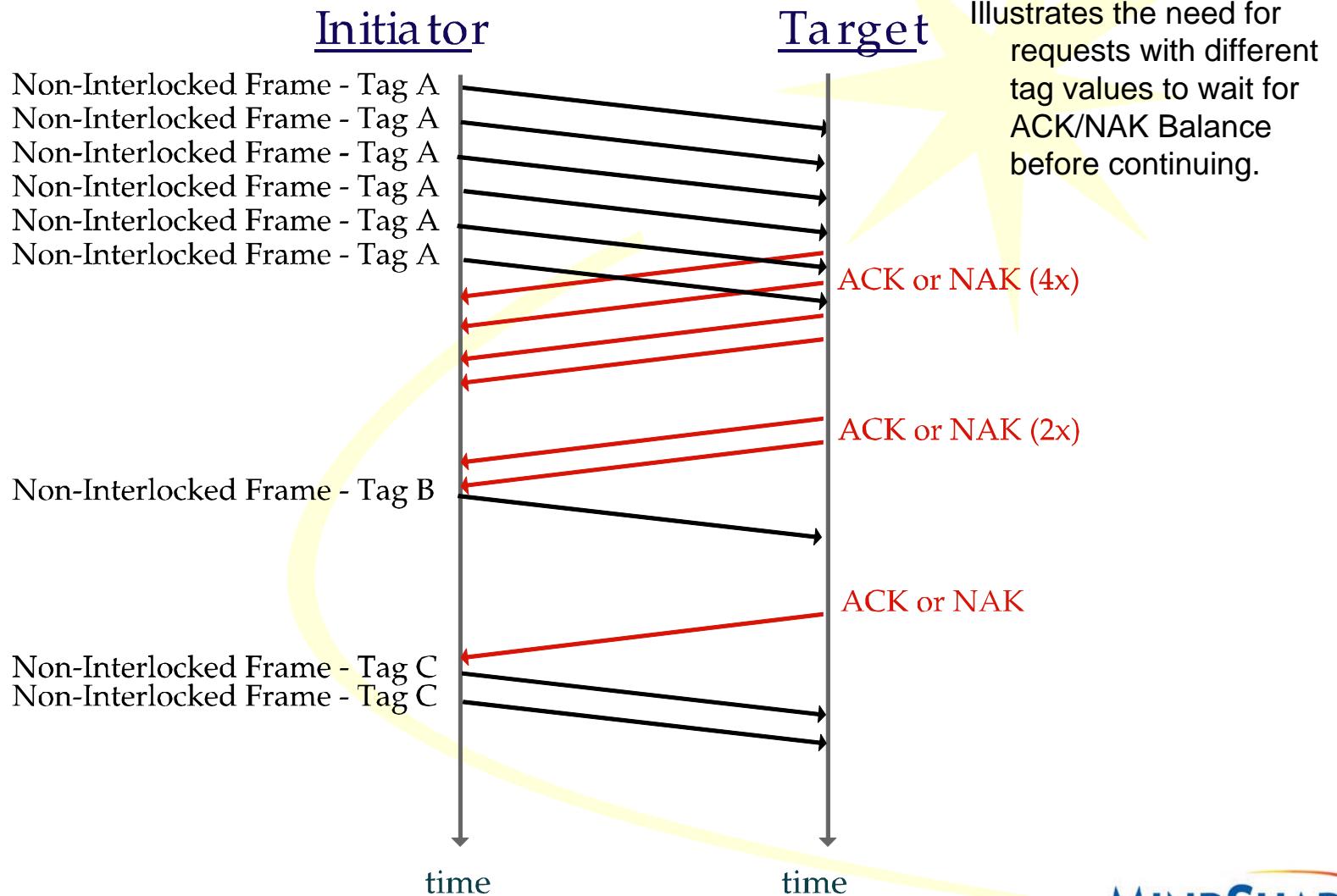
# Interlocked Frames



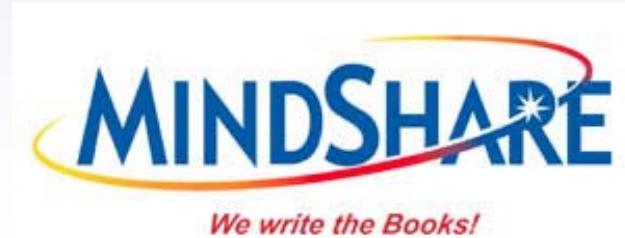
# Non-interlocked frames



# ACK/NAK Example 2 (354)



# *SMP Connections*



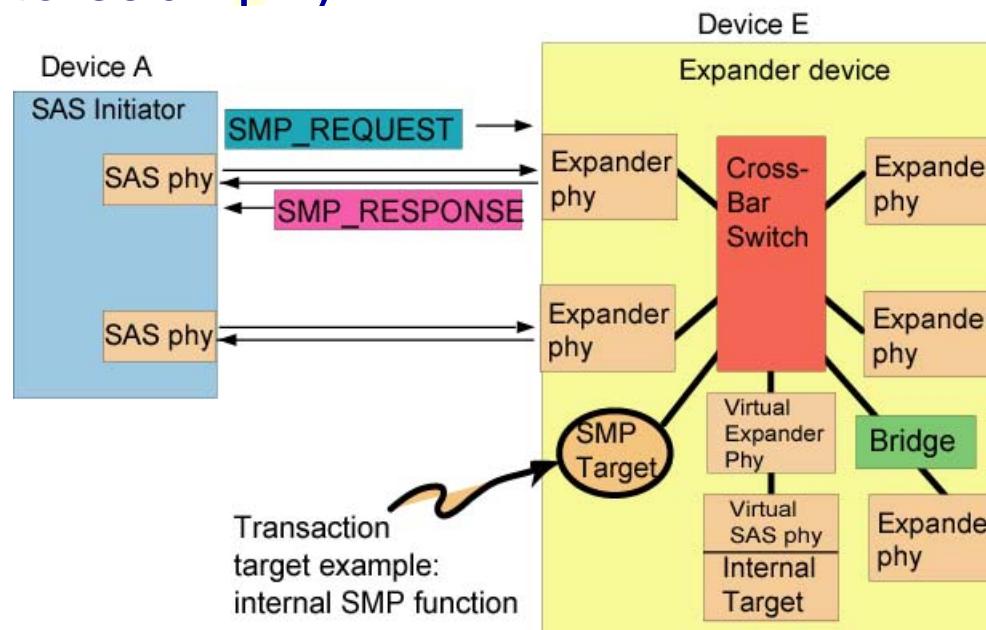
Christy Choi([christy.choi@sandisk.com](mailto:christy.choi@sandisk.com))  
Do Not Distribute

# SMP - SAS Management Protocol (361)

- Purpose:
  - Support SCSI system management tasks
  - Access the control and status registers of a device
- Only initiators can open SMP connection
- Half-duplex (not queued)
- No ACK or NAK
- No flow control
  - Target receiving frame implicitly grants “credit”
- Initiator always sends one frame, gets one response, and closes the connection

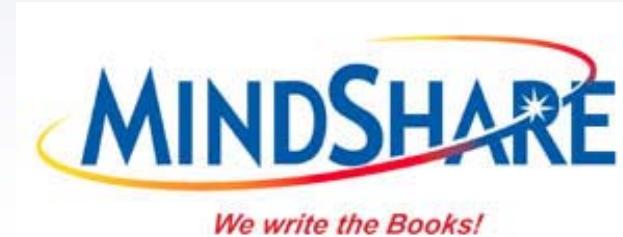
# SMP Connection

- After OPEN accept with SMP protocol, two frames are sent: request from initiator and response from target, followed by CLOSE in both directions.
- The SMP target in an expander is important because it gives initiators a simple way to query every Phy within it and learn the address and device type attached to each phy.

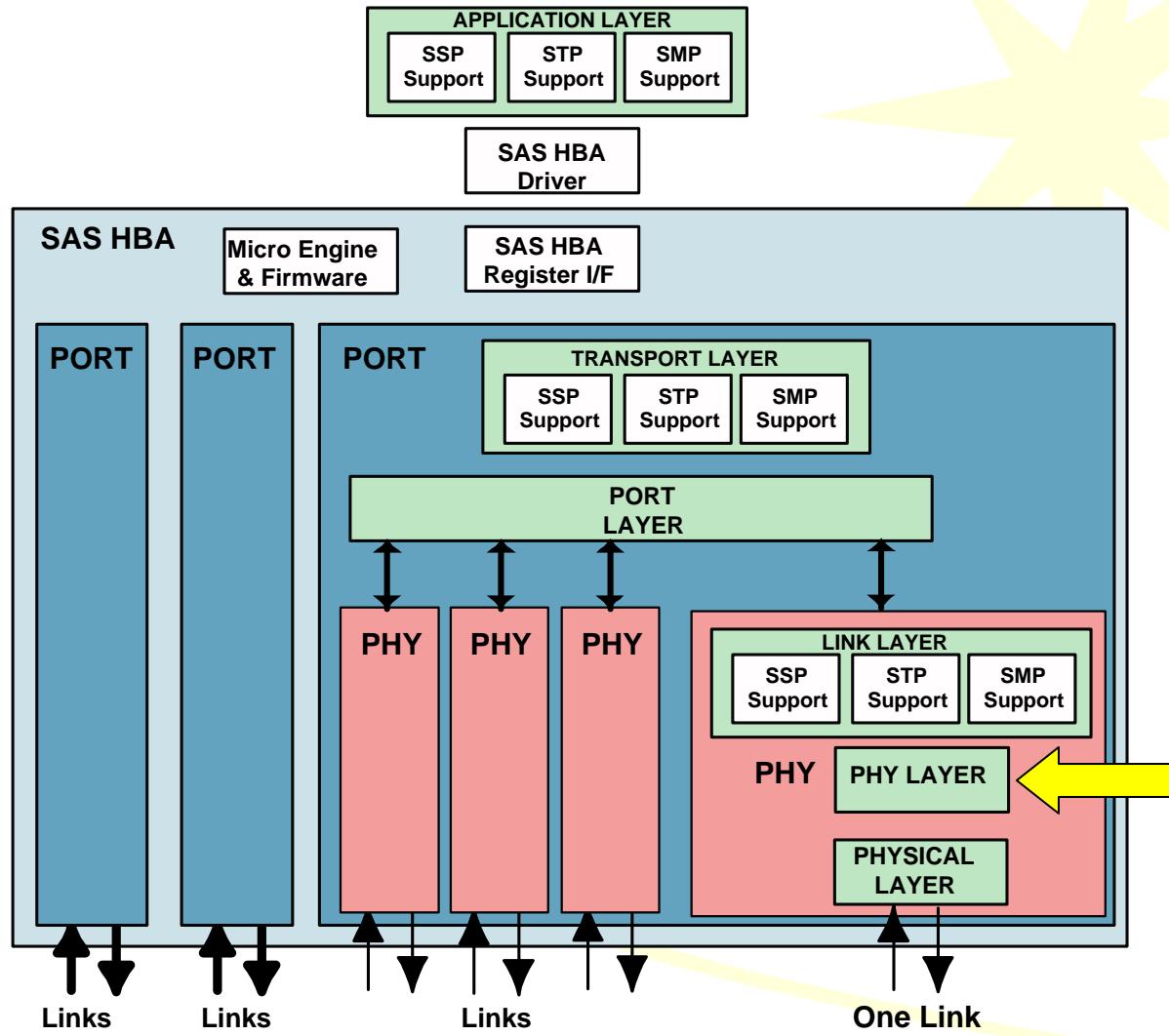


# *Chapter 16*

## *Phy Layer*



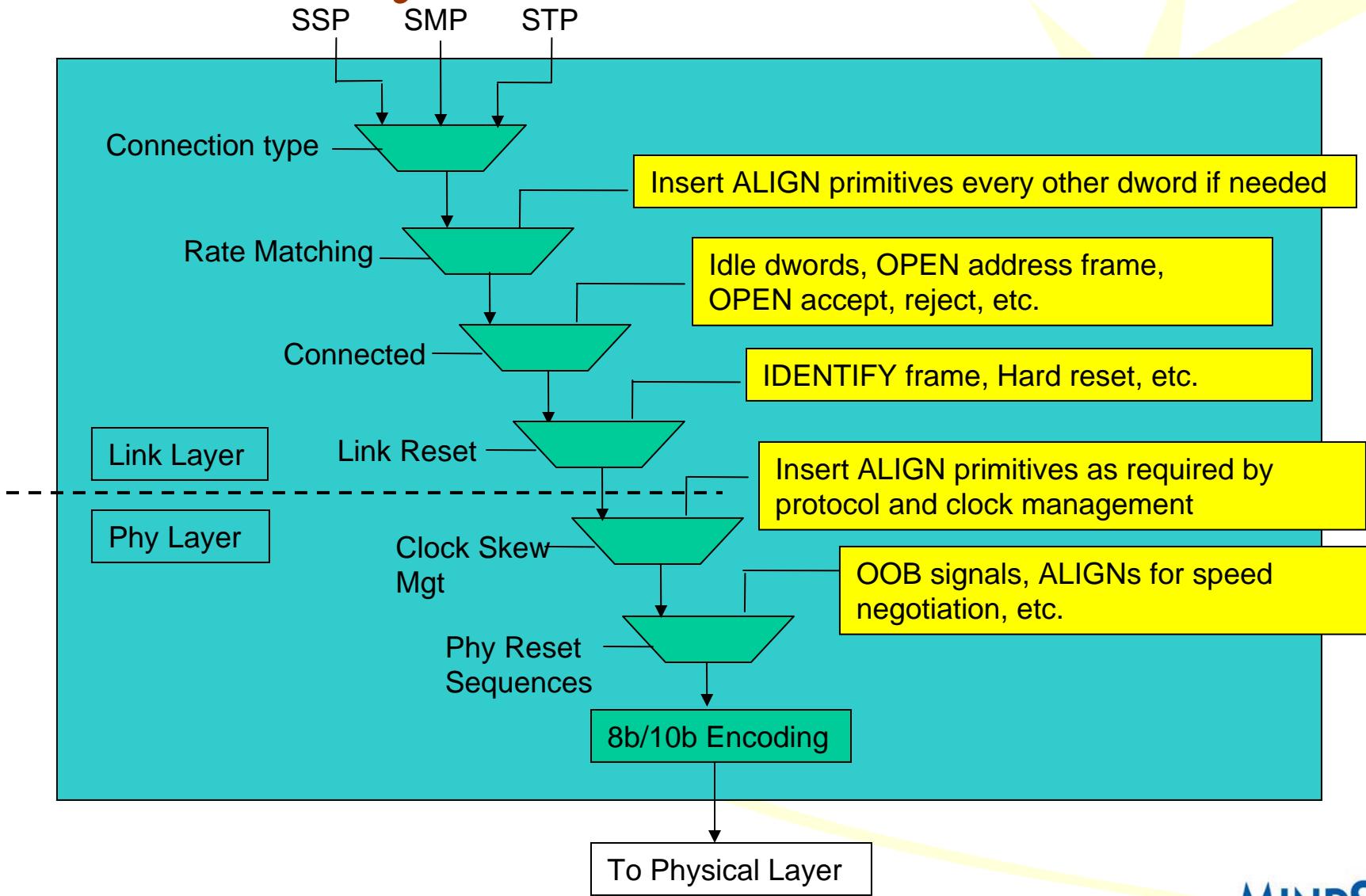
# SAS Layer Review (366)



# Phy Layer Responsibilities

- Inject primitives for Link training and Phy reset sequences
  - OOB (Out of Band) Signaling
- 8b/10b Encoding/Decoding
- Inject ALIGN primitives for Clock Skew Management
- Clock Recovery at receiver
- Dword synchronization

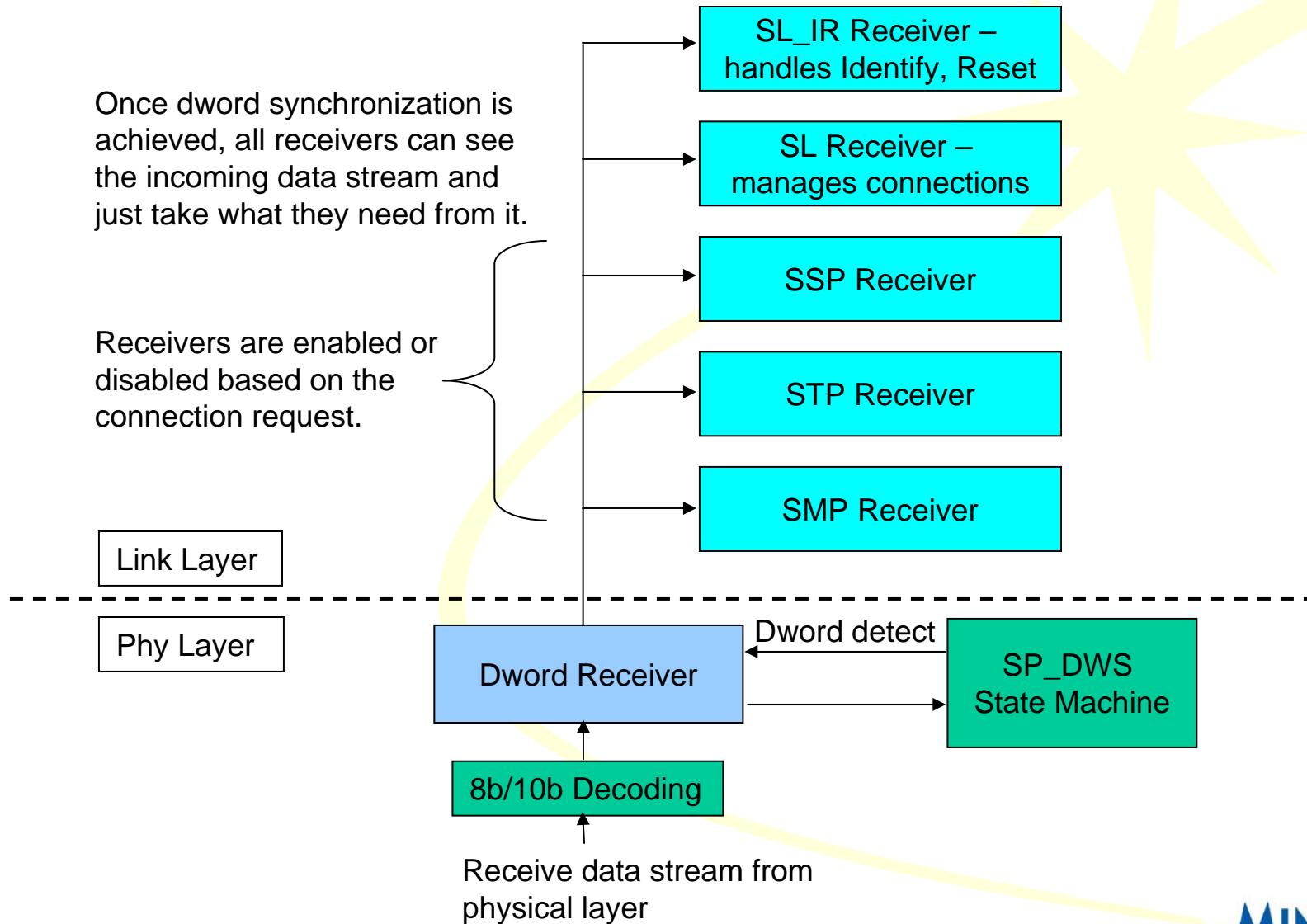
# Phy Tx Data Path (367)



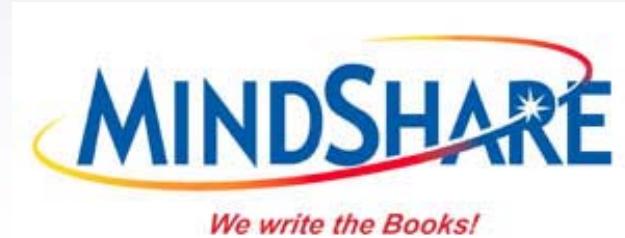
# Phy Rx Data Path (369)

Once dword synchronization is achieved, all receivers can see the incoming data stream and just take what they need from it.

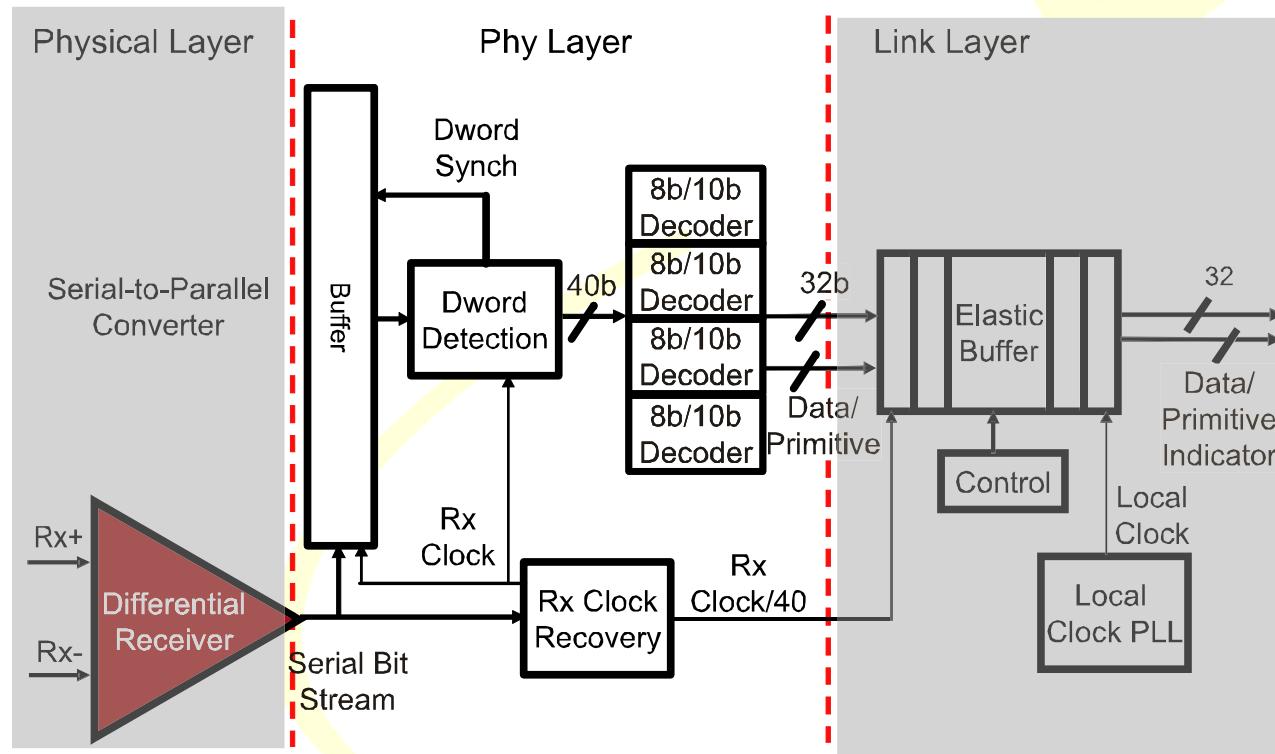
Receivers are enabled or disabled based on the connection request.



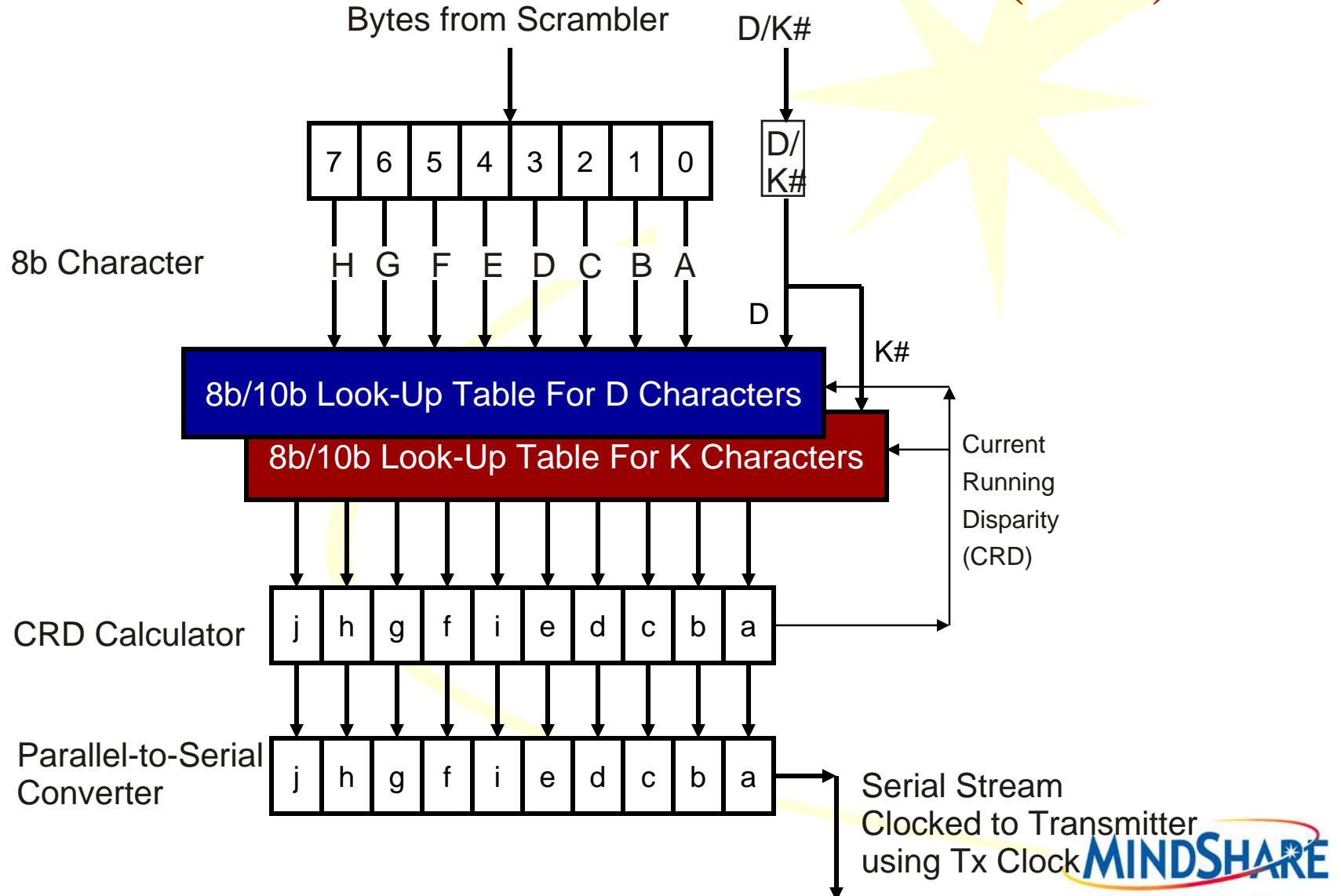
# *Encoding and Dwords*



# Phy Layer Rx Block Diagram (370)



# 8b/10b Encoder and Parallel-to-Serial Converter (371)

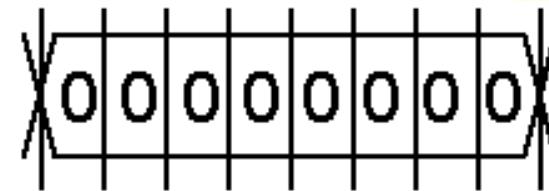


# 8b/10b Tutorial (372)

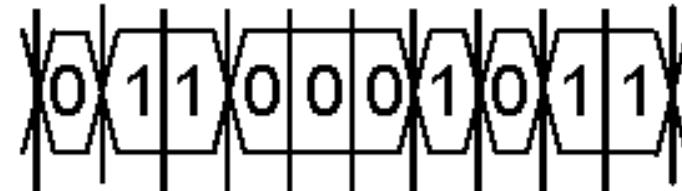
- Standard encoding method invented by IBM and used in several serial standards:
  - Ethernet, Fibre Channel, ServerNet, FICON, IBA
- Goals of 8b/10b encoding
  - Create sufficient transition density and limit “Run Length”
  - Balance number of 1’s and 0’s to maintain “DC Balance”
  - Encode special control (K) characters
  - Facilitate detection of transmission errors
- Cost: Performance loss of 20% at receiver

# 8b/10b Encoding Example (372)

8b Value  
Data 00h



10b Encoded  
Value

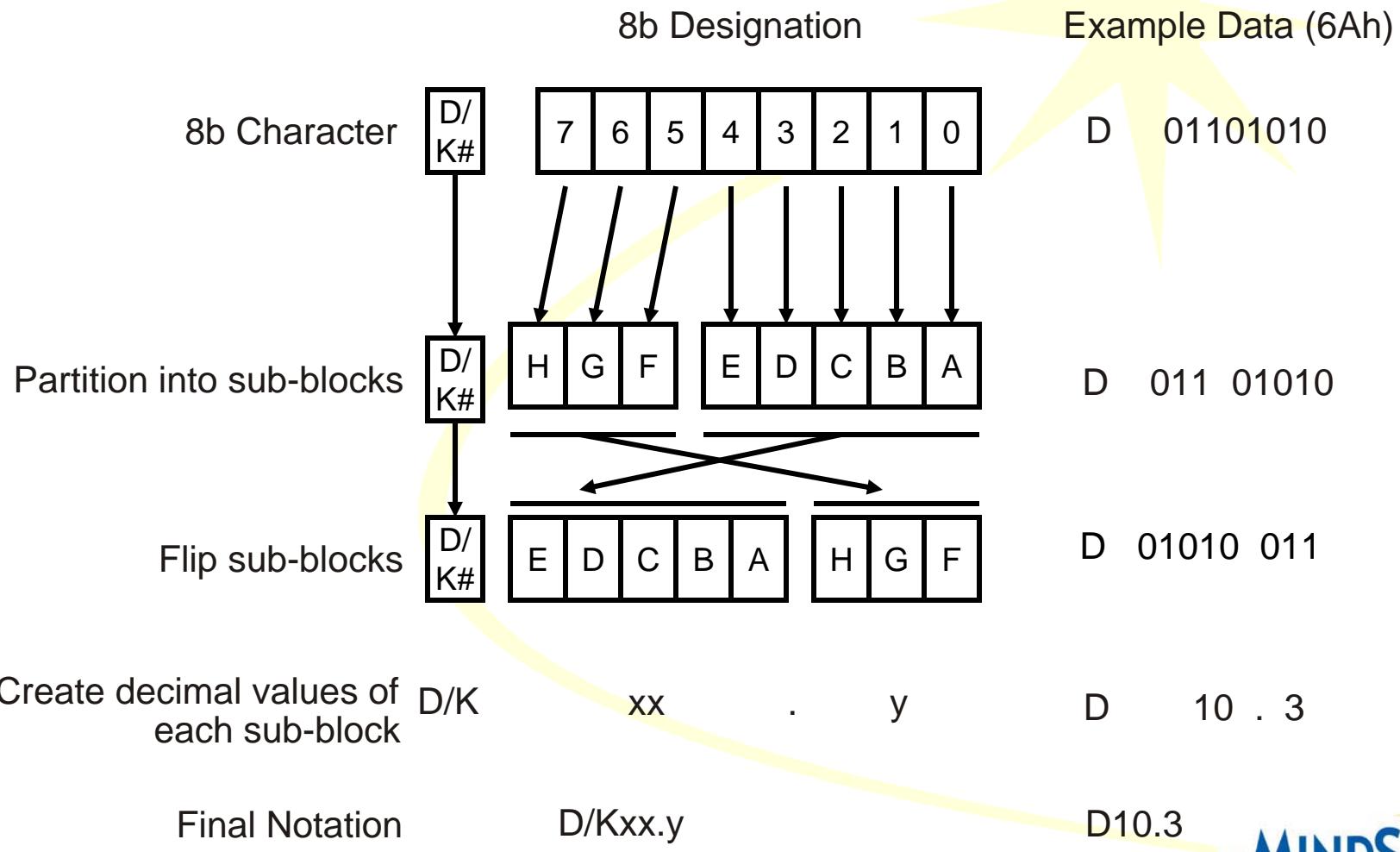


# DC Balance in 8b/10b (372)

- Balance is maintained by monitoring the Disparity or difference in the number of 1's and 0's of a character
- Character with:
  - Four 0's and six 1's has positive disparity
  - Six 0's and four 1's has negative disparity
  - Five 0's and five 1's has neutral disparity
- Most bytes encode to 1 of 2 possible characters based on the “Current Running Disparity” (CRD).
  - One encoding contains four 0's and six 1's while the second encoding contains six 0's and four 1's, or
  - Both encodings contain five 0's and five 1's (neutral disparity)

# 8b/10b Nomenclature (375)

- Notation in 8b/10b Tables



# Properties of 10b symbols (390)

- Generally equal number of 1's and 0's over 2 consecutive characters
- Maximum of five continuous 1's or 0's
- Each character has 4/6 or 6/4 or 5/5 zeros/ones
- 6-bit sub-block of 10b character contains no more than four 1's or four 0's
- 4-bit sub-block of 10b character contains no more than three 1's or three 0's

Any character without these properties is considered invalid

# Some Example Encodings (373)

Encode to this if CRD is positive

Encode to this if CRD is negative

This is 8-bit character

D or K Character	Binary Bits HGF EDCBA	Byte Name	CRD – abcdei fghj	CRD + abcdei fghj
Data (D)	011 01010	D10.3	010101 1100	010101 0011
Data (D)	000 11011	D27.0	110110 0100	001001 1011
Data (D)	111 10111	D23.7	111010 0001	000101 1110
Control (K)	111 10111	K23.7	111010 1000	000101 0111
Control (K)	101 11100	K28.5	001111 1010	110000 0101

If character encode yields neutral disparity, then CRD remains unchanged, else it flips

# Example Transmission (373)

Use these two characters in the example below:

D/K#	Hex Byte	Binary Bits HGF EDCBA	Byte Name	CRD – abcdei fghj	CRD + abcdei fghj
Control (K)	BC	101 11100	K28.5	001111 1010	110000 0101
Data (D)	6A	011 01010	D10.3	010101 1100	010101 0011

## Example Transmission

	CRD	Character	CRD	Character	CRD	Character	CRD
Character to be transmitted	-	K28.5 (BCh)	+	K28.5 (BCh)	-	D10.3 (6Ah)	-
Bit stream transmitted		Yields 001111 1010 Disparity is +		Yields 110000 0101 Disparity is -		Yields 010101 1100 Disparity is neutral	

Initialized value of CRD is don't care. Receiver can determine from incoming bit stream

# Control Characters (389)

- 10-bit output means we're adding 2 bits for each byte, giving 4 possible encodings for each.
  - We only need 2 encodings to maintain DC balance, though, and some data bytes map to the same character regardless of CRD, so even limiting the number of encodings to only half still leaves 12 encodings for special “control characters”
- Only 3 control characters are currently used in SAS spec, and only K28.5 contains the comma pattern (see next slide)

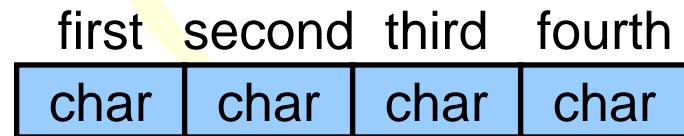
Special Code Name	10b (CRD-)	10b (CRD+)	Usage in SAS	Usage in SATA
K28.3	001111 0010	000101 0111	Primitives used in STP connections	All primitives except ALIGN
K28.5	<u>001111 1010</u>	<u>110000 0101</u>	ALIGN and most primitives used in SAS	ALIGN
K28.6	001111 0100	110000 1011	SATA_ERROR (used on SATA physical links)	Not used

# Comma Description (388)

- Comma (K28.5) is the first symbol in a primitive (see following slides)
- Encoding of K28.5 contains 2 bits of one polarity followed by 5 bits of the opposite polarity (**001111 1010** or **110000 0101**)
  - This property does not legally appear in any other case, making this pattern easily detectable
  - One SATA primitive starts with K28.5 – the ALIGN primitive

# Bits to Dwords

- Byte – 8 bits
- Character – 10 bits
- Dword – 4 bytes or 4 characters
  - Represents 32 bits of data
  - Uses 40 bits on the link
  - Dword can be data or a primitive



# Primitives

- Primitive is a dword with one control character followed by 3 data characters
  - First character is K28.5 (for SAS primitives), K28.3 (for SATA primitives), or K28.6 (special SATA error primitive)
  - Unlike other dwords, “endianness” is not a factor for primitives, because both SAS and SATA always send the control character first

	first	second	third	fourth
K28.5	Dxx.y	Dxx.y	Dxx.y	
K28.3	Dxx.y	Dxx.y	Dxx.y	
K28.6	Dxx.y	Dxx.y	Dxx.y	

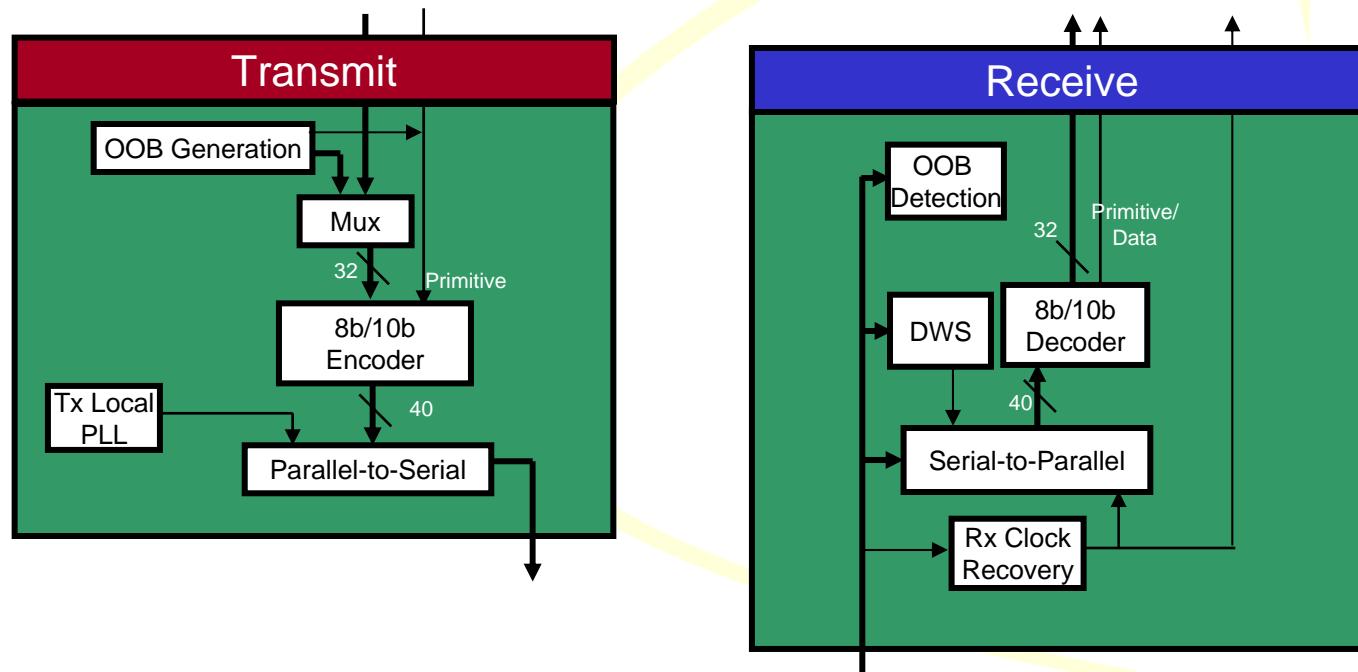
# Primitive Characteristics

- Primitives, defined by link layer, have no CRC and so must have error protection built in
  - Data characters were chosen for best Hamming distance, requiring 8 bit-errors to morph one valid primitive into another.
- Phy layer
  - D10.2 characters during SATA speed negotiation
  - ALIGN(0) primitive during OOB signaling and speed negotiation
  - ALIGN(1) primitive during SAS speed negotiation
- Must start with K28.5 or K28.3 to be valid, and must not have control character in any but first position

	first	second	third	fourth
ALIGN(0)	K28.5	D10.2	D10.2	D27.3
ALIGN(1)	K28.5	D07.0	D07.0	D07.0

# Detailed Physical Layer

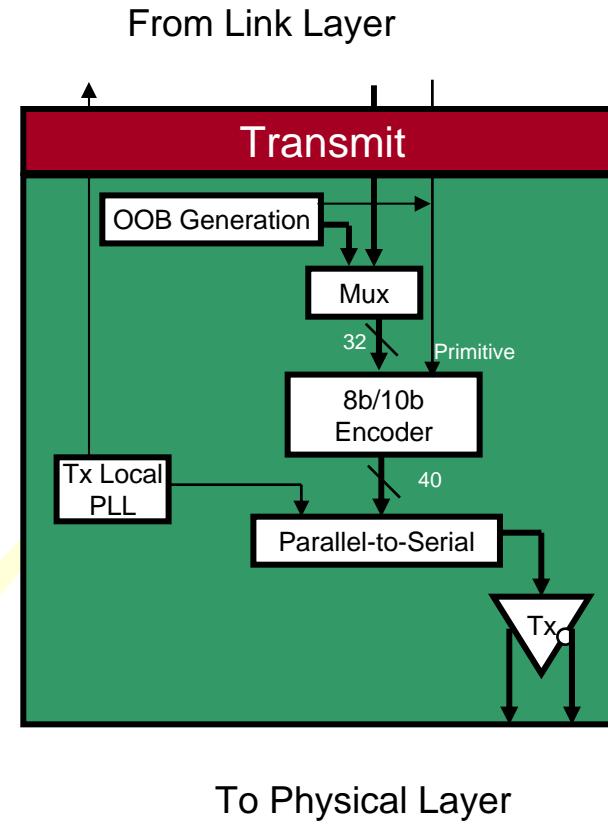
- Responsibilities:
  - Differential signaling
  - SER/DES
  - 8b/10b
  - OOB
  - Rx Clock Recovery



# Transmit Physical Layer (392)

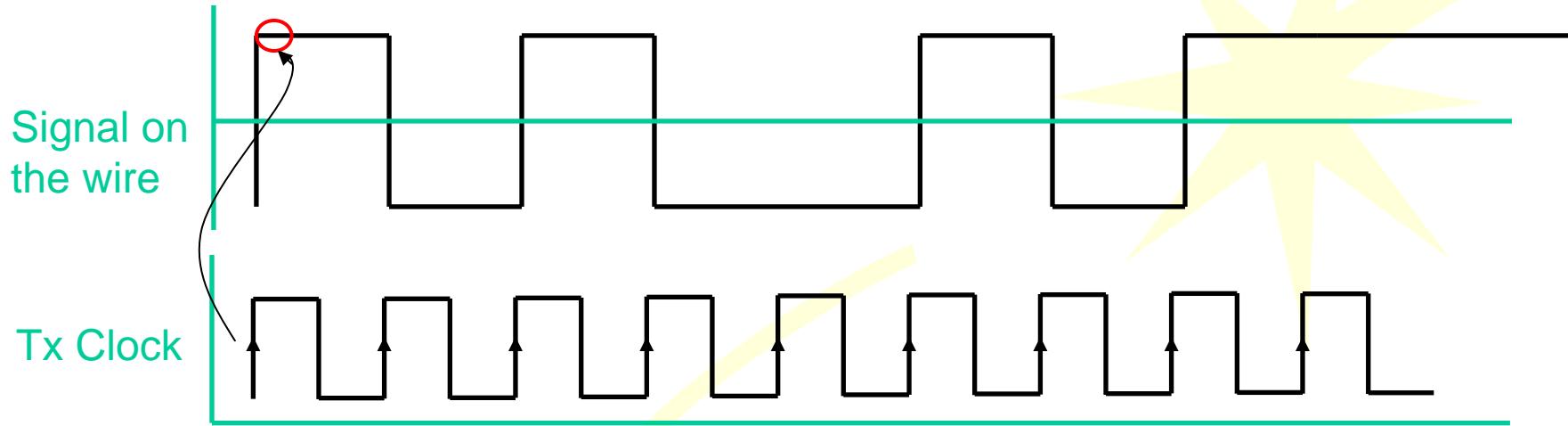
**SAS devices will usually implement a dword-wide path, resulting in an internal frequency of 75MHz for a 3Gbps link.**

**Note that the highest frequency actually seen on the wire is half of the transmission clock (see next slide)**



To Physical Layer

# Transmission Frequency (394)



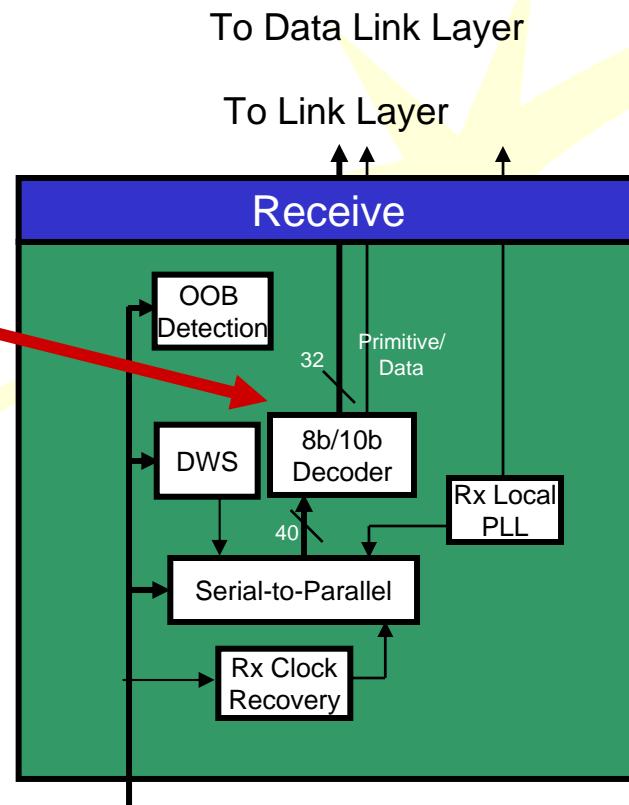
Signal frequency cannot be greater than half of transmitter clock.

# Tx Clock (391)

- The serial output is clocked out to the differential driver using Tx Clock of 1.5 or 3.0 Gbps
- Accuracy:
  - +/- 100ppm (or 200ppm) for SAS.
    - This can result in gaining or losing 1 clock every 5000 clocks
  - +350/-5350ppm for SATA. SSC (Spread Spectrum Clocking) is permitted, resulting in large apparent jitter
    - SAS devices don't use SSC, but if they support SATA, they must support the wider frequency range.
- Note that Tx Clock is different from the Rx local clock even though both run at same rate - they are in different clock domains.

# Receive Block (395)

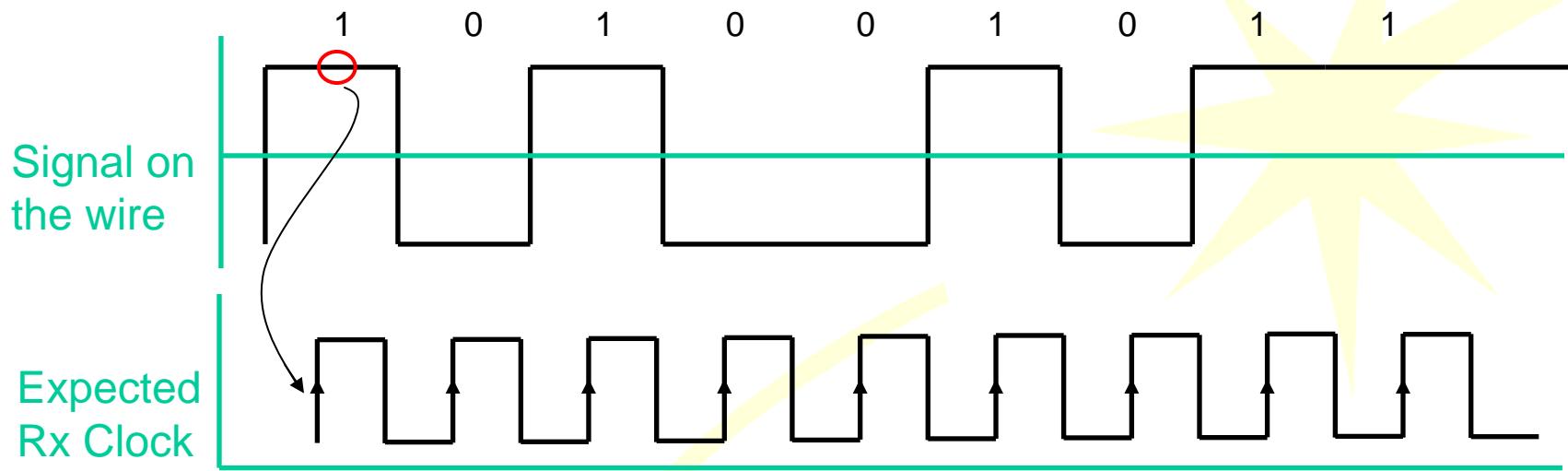
- 8b/10b decodes the 10b character stream into:
  - four 8b data characters or
  - one 8b control character followed by 3 data characters
  - the indication of primitive/data



# Rx Clock (396)

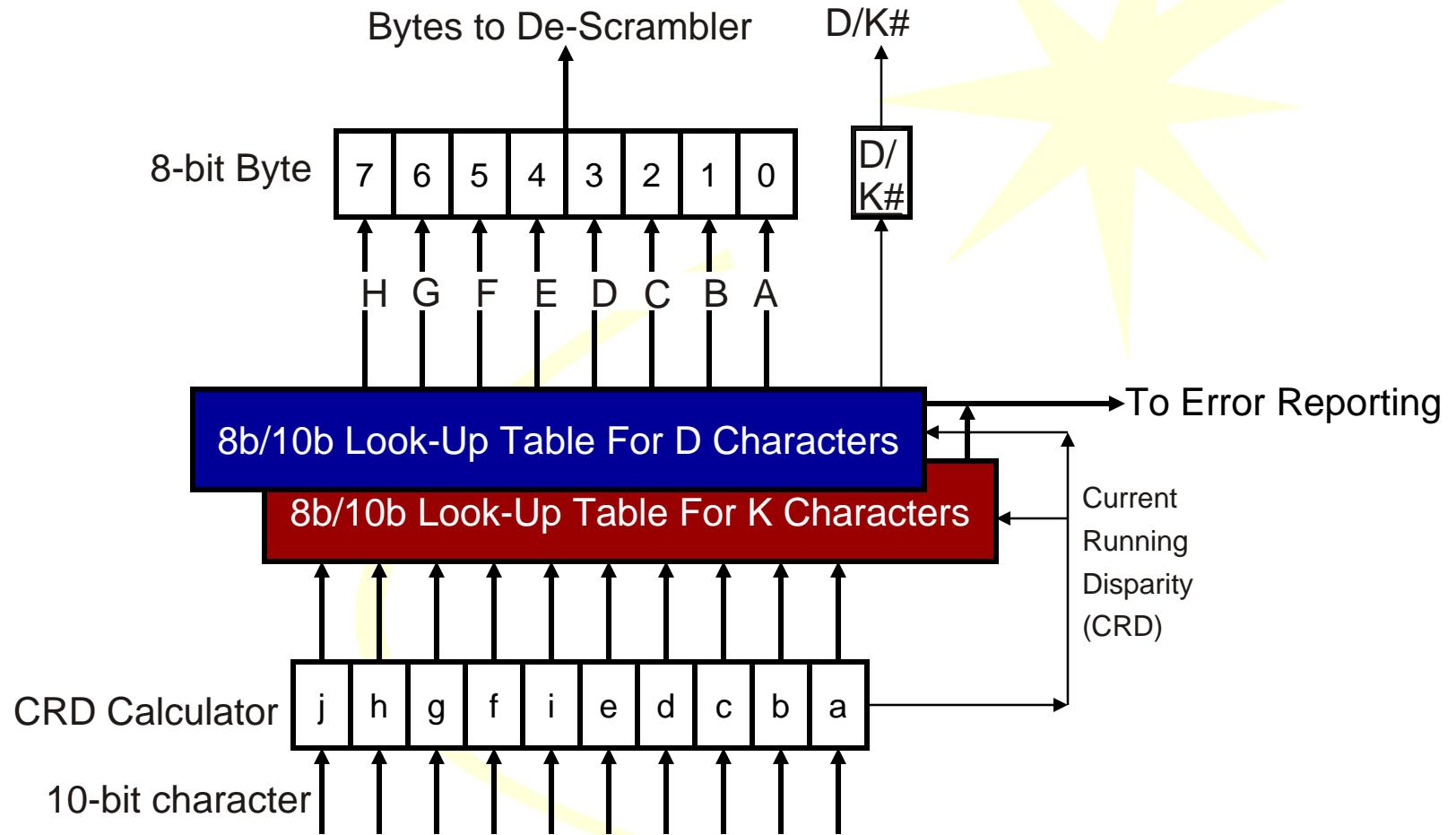
- Using PLL or other means, receiver circuit generates a Rx Clock from transitions in the input data
- Rx Clock has same frequency as the Tx Clock at the transmitter
- 1.5 GHz or 3.0 GHz clock recovered
- Rx Clock latches incoming packets into register and elastic buffer
- “Rx Clock” is different from “Rx Local Clock” used to clock data out of the elastic buffer  
+/- 100ppm difference

# Clock Recovery (396)

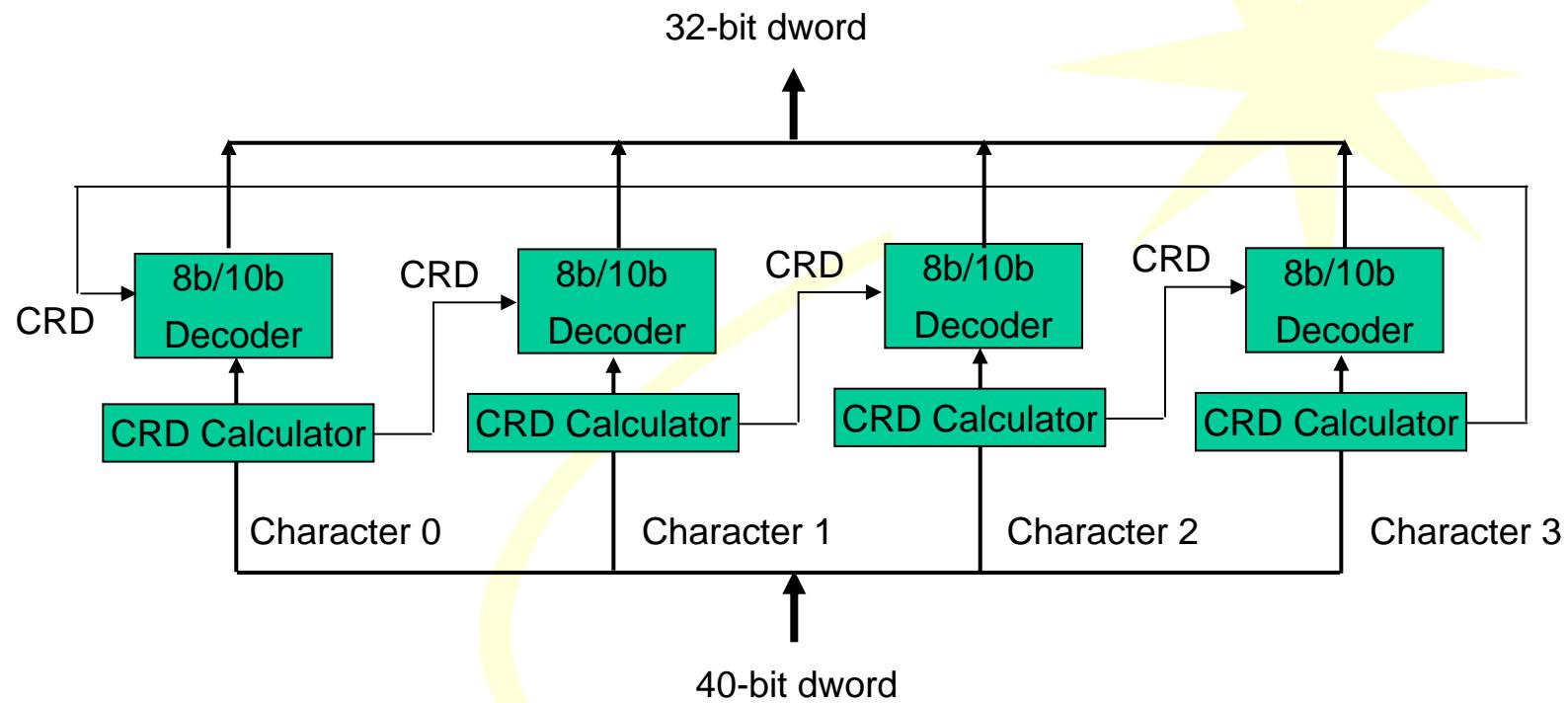


Receiver PLL knows what the negotiated frequency is, and only needs to align clock rising edges away from signal transitions to reliably latch incoming bits.

# 8b/10b Decoding (397)



# 40-bit Version of 8b/10b Decode (398)



# Disparity Error Example (399)

	CRD	Character	CRD	Character	CRD	Character	CRD
Transmitted Character Stream	-	D21.1	-	D10.2	-	D23.5	+
Transmitted Bit Stream	-	101010 1001	-	010101 0101	-	111010 1010	+
Bit Stream After Error	-	101010 10 <u>11</u>	+	010101 0101	+	111010 1010	+
Decoded Character Stream	-	D21.0	+	D10.2	+	Invalid	+

Error occurs here

Error detected here  
This dword labeled as invalid

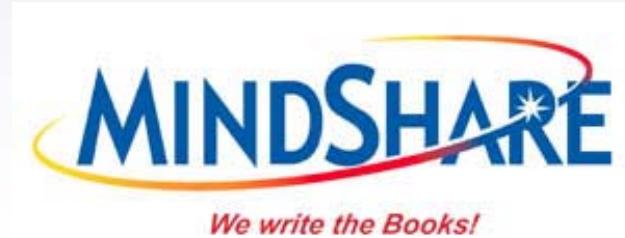
# Physical Layer Error Handling

- Physical Layer Detectable Errors
  - 8b/10b disparity error
  - 8b/10b code violation
  - Elastic Buffer overrun or underrun error
  - Packet not consistent with packet format rules
- Response: Send NAK

# No Power Management (399)

- Since it targets the Enterprise market, SAS does not include power management features.
- Important Enterprise market concerns:
  - Fast access time
    - Recovery time from low power states is undesirable
  - Reliability
    - If a device is put into a low-power state, there is some risk that it won't come back up properly
    - Enterprise hard drives are designed for fewer power cycles - integrity could be reduced if they were powered down too often.

# *Resets*

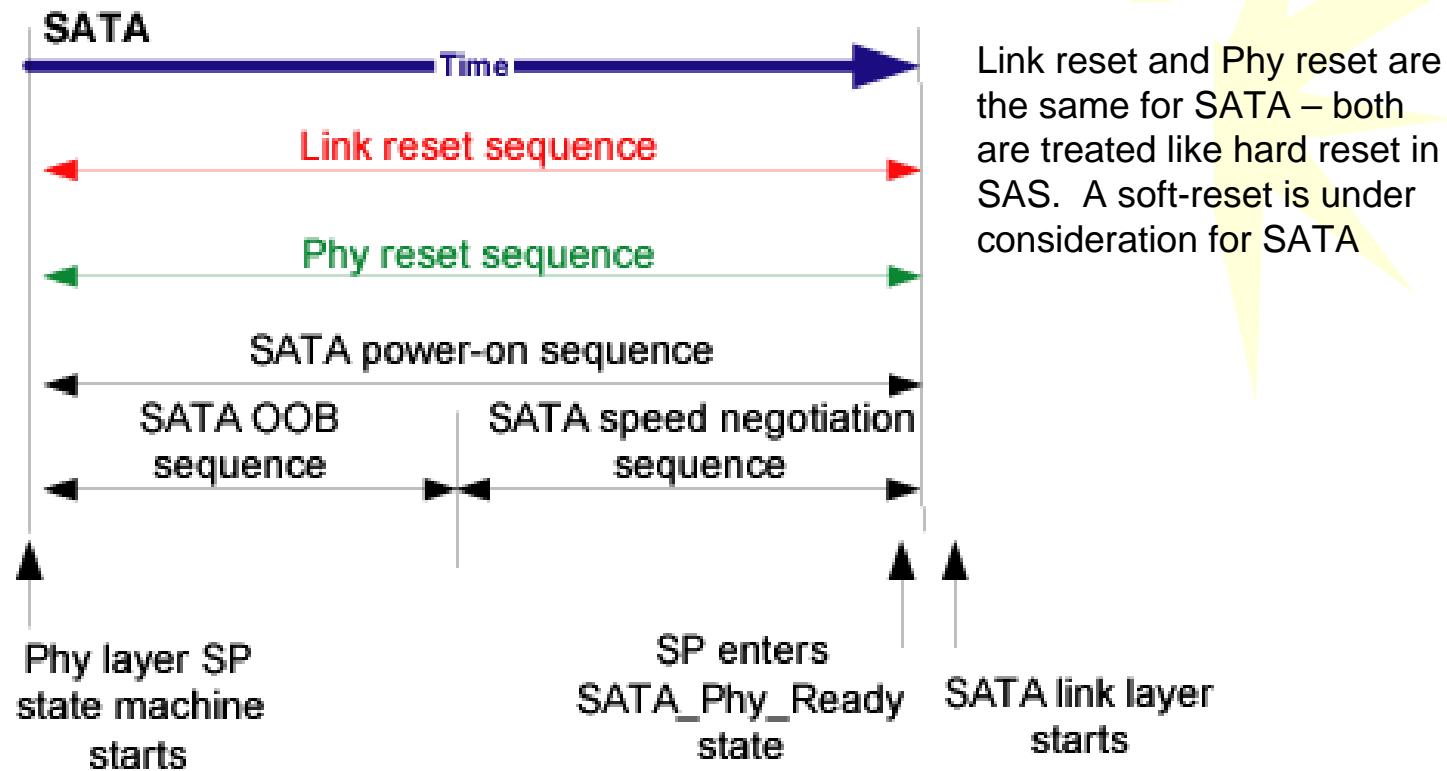


Christy Choi(christy.choi@ sandisk.com)  
Do Not Distribute

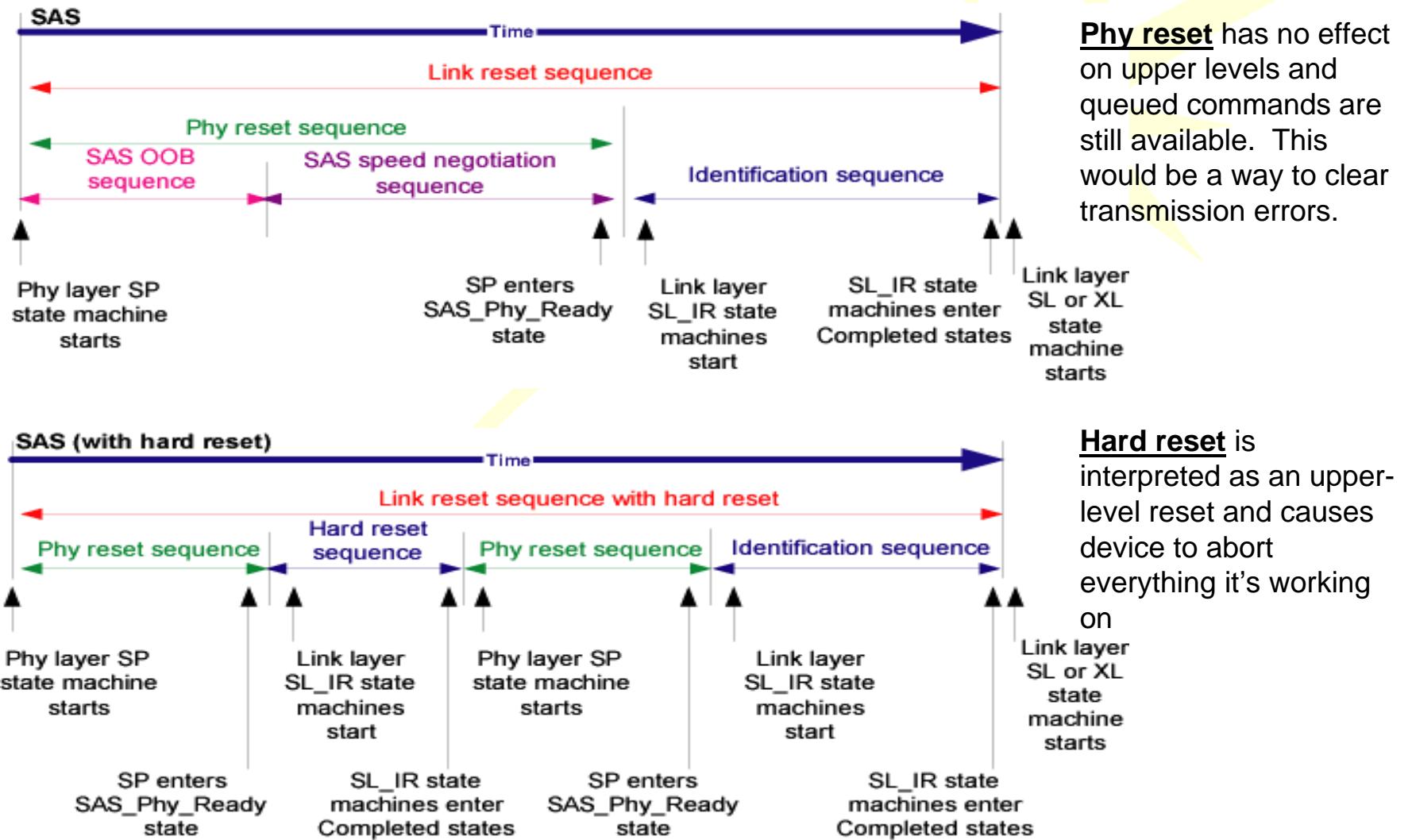
# Resets (400)

- Reset sequences:
  - SATA Reset
  - SAS Phy Reset, or Link Reset, or Link Reset followed by HARD\_RESET redundant primitive sequence (send 6, must see 3)
- After reset
  - Phys exchange OOB to establish communications
  - Phys go through speed negotiation
  - Each SAS phy sends its address to its neighbor with Identify address frame (SATA devices don't use addressing and don't do this)
  - Initiators begin the process of discovery by sending SMP requests to neighboring expanders

# SATA Reset Sequence (402)



# SAS Reset Sequences (403)

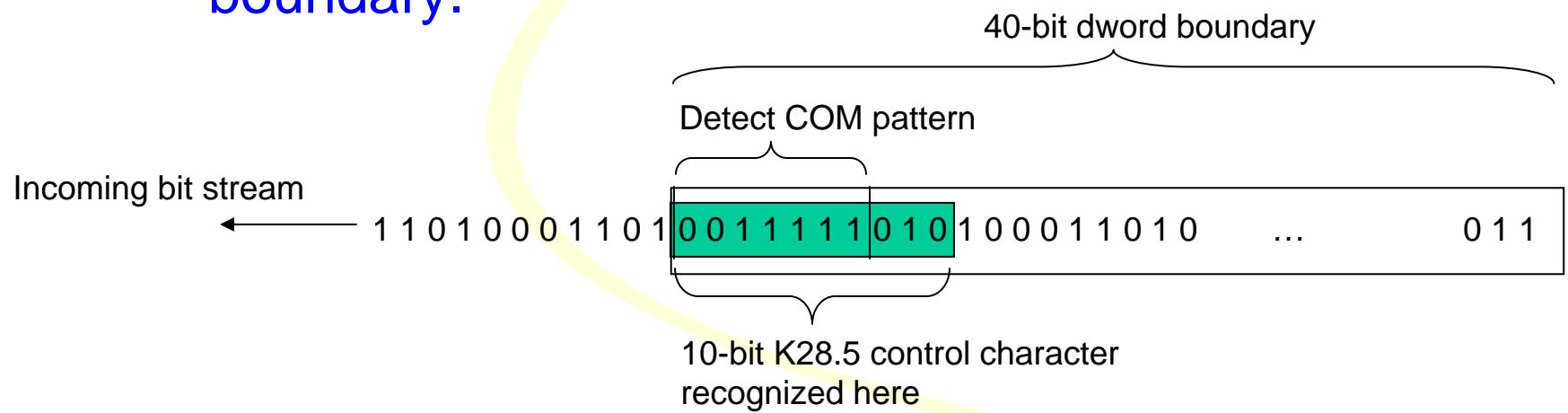


# Expander Reset

- Expanders do not forward OOB or Link Reset, and they do not reset (don't clear routing tables for example)
- Hard reset sent to one phy of a wide port will cause all the phys of that port to reset, but not the other ports of that device.
- Phy Reset Timeout
  - If IDENTIFY frame not received within 1ms of phy reset completion, restart the phy reset sequence

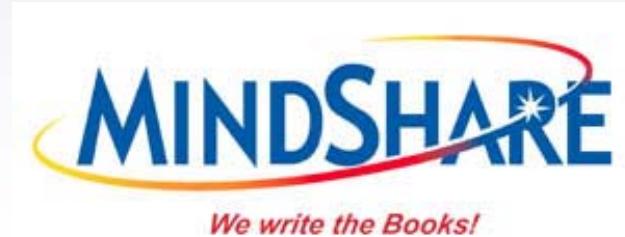
# Dword Detection (409)

- Once receiver clock is recovered, detect Dword boundaries
  - Search incoming bits for the 7-bit COM pattern
  - Once found, the next 3 bits will finish the K28.5 character, so character boundary is known. By definition, the 30 bits that follow it will complete a 40-bit Dword primitive and define the dword boundary.

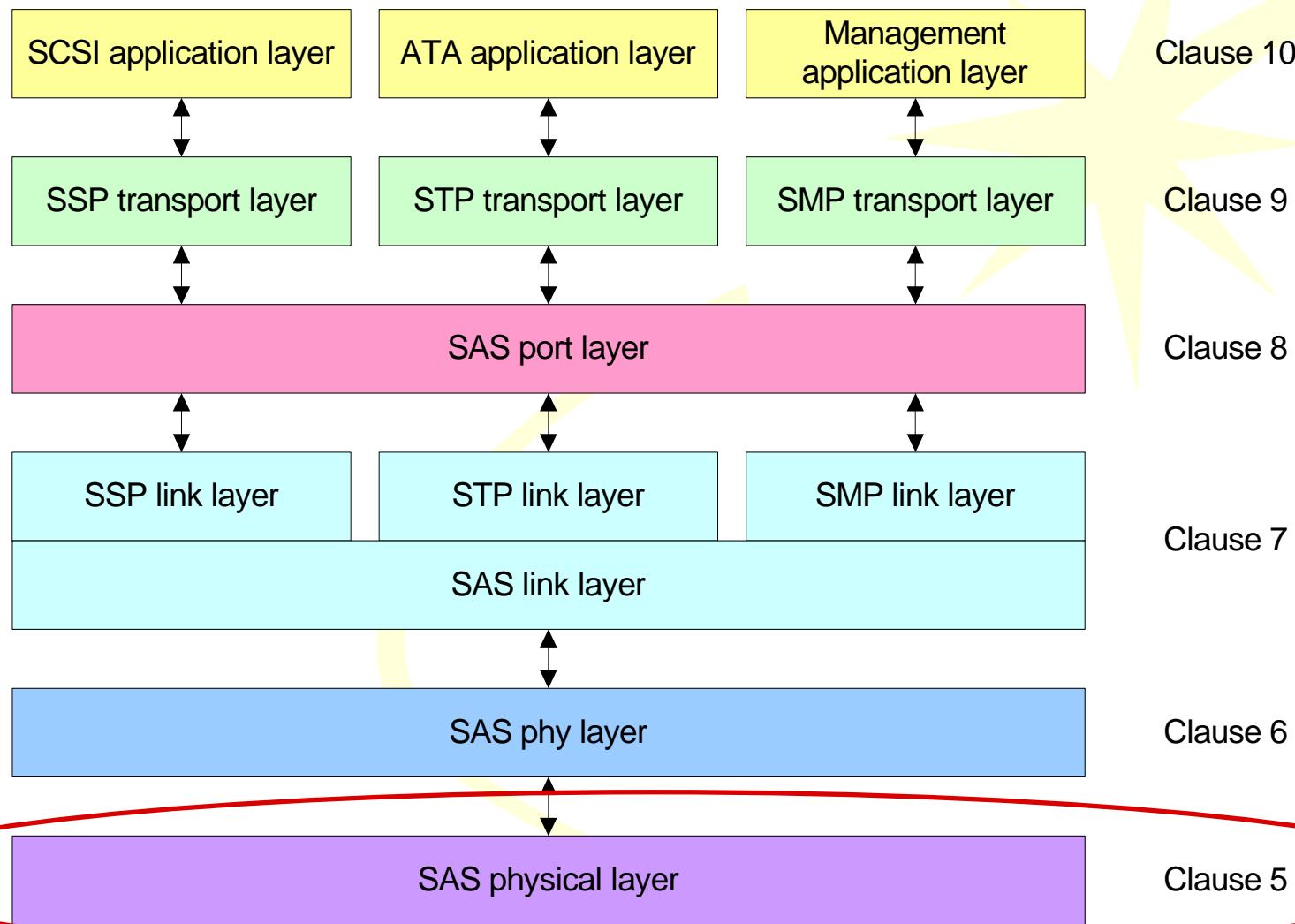


# *Chapter 17*

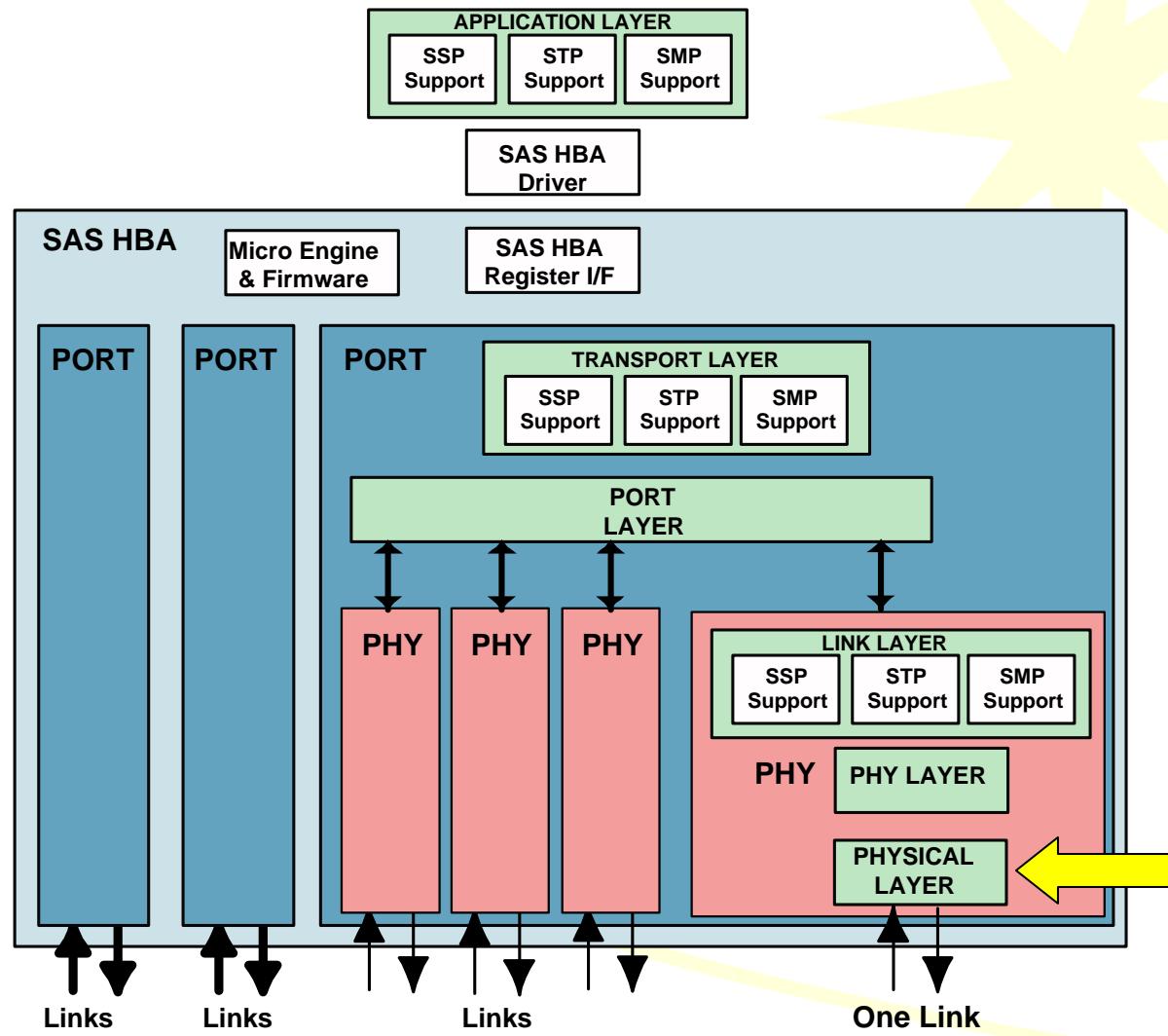
# *Physical Layer*



# SAS standard layering



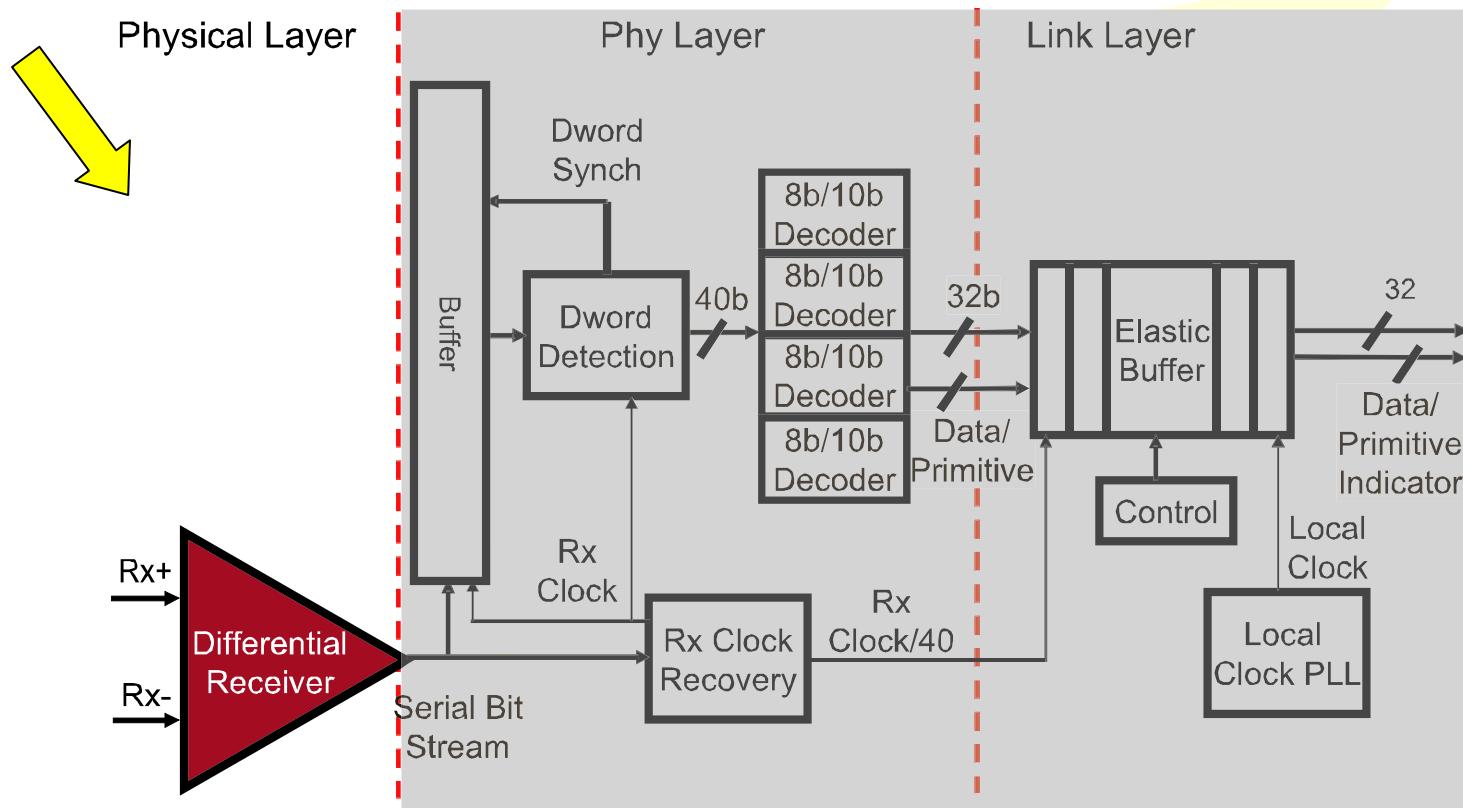
# SAS Layer Review (366)



# Physical layer

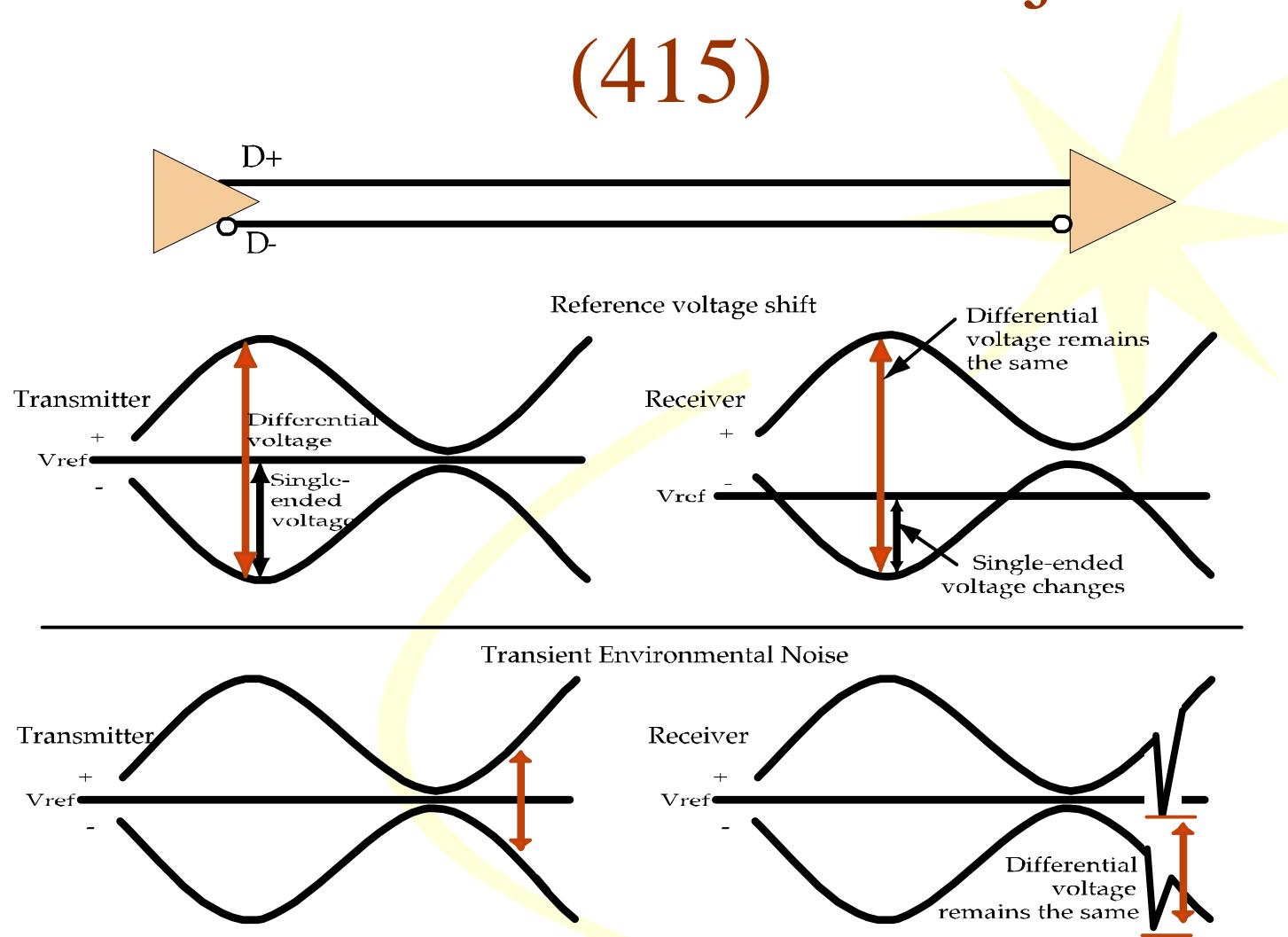
- Differential signaling
- Passive interconnect
  - Internal cables/connectors
  - External cables/connectors
- Electrical interface
  - SATA
  - SAS
  - READY LED signal
  - Pre-emphasis and equalization

# Physical Layers (413)



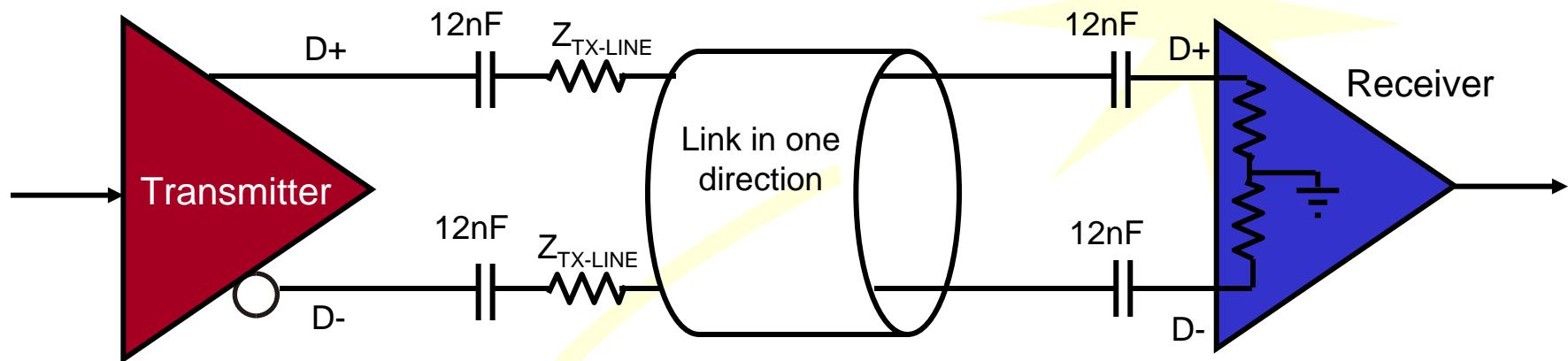
# Common-Mode Noise Rejection

## (415)



# Differential Buffers (417)

Caps on board near connector for AC coupling. SAS Rx must include it, Tx may also include it. SATA may or may not have them on either end.

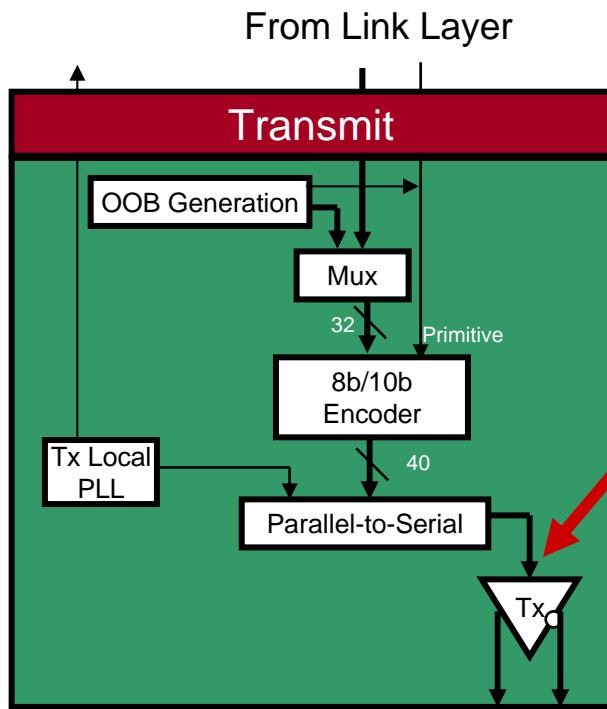


The DC common mode impedance is typically 50 ohms, differential impedance is 100 ohms.

The termination of 100 ohms differential is typically implemented internal to the receiver.

The coupling capacitor is between 10 and 12 nF.

# Differential Transmitter

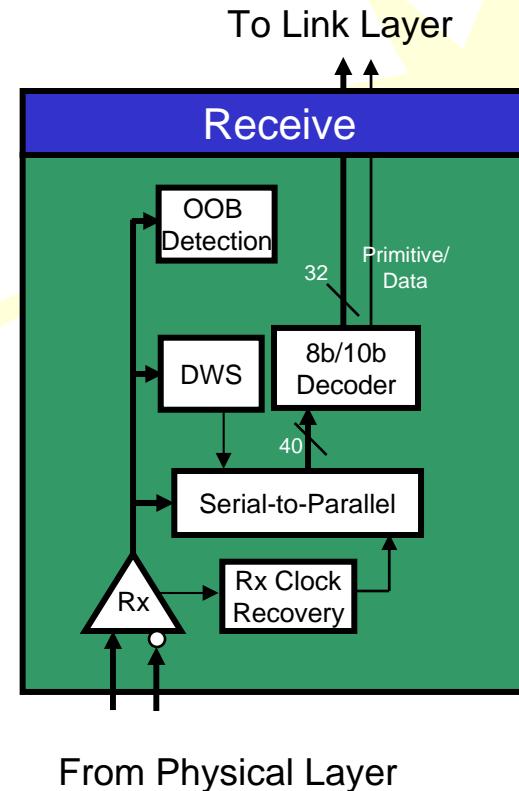


- Serial data pattern is transmitted differentially
- NRZ encoding
- Transmission at 1.5 or 3.0 Gbps
- Differential peak-to-peak Voltage  
SATA: 400-600mV  
SAS: 325-1600mV

Wide voltage range is needed to accommodate longer transmission lines where greater loss is expected. Choice of voltage levels is design-specific.

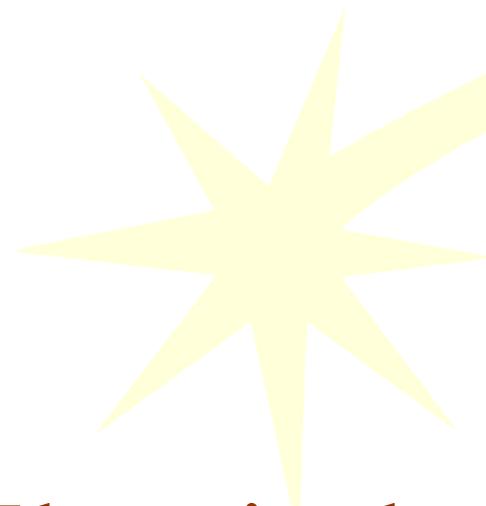
# Differential Receiver

- Differential receiver senses a voltage difference
  - + difference = logical 1
  - difference = logical 0
- OOB detection might be analog squelch detect



# High Speed Signaling

- Clock requirements
  - Spread Spectrum Clocking not supported for SAS Tx
    - Enterprise designs are considered “inside the box” and don’t have the same emission requirements of consumer designs, so they don’t need SSC
  - But any Phy that supports SATA must tolerate SSC. PLL designers must allow for wider tolerance.
- Transmission line characteristics defined by TCTF (Tx Compliance Transfer Function) to give a worst-case mathematical model of the medium



# Physical layer - Electrical characteristics

# SAS Electrical characteristics

- Transmitter, Receiver tightly specified
  - Eye masks, 75 ps skew between Rx– and Rx+
- A.C. coupling only – receivers must include the capacitor, transmitters may include it
- SSC not supported

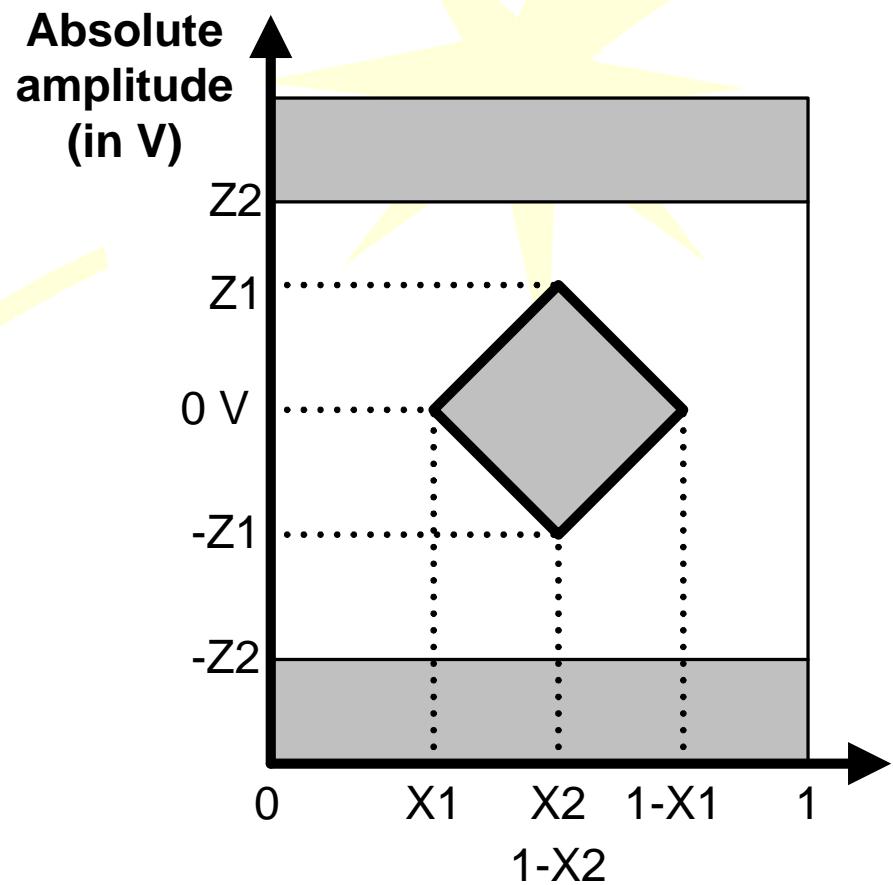
# SATA Electrical characteristics

- Transmitters and Receivers are specified
- Cables are indirectly (or imprecisely) specified
- Either A.C. or D.C. coupling allowed
- SSC is optional

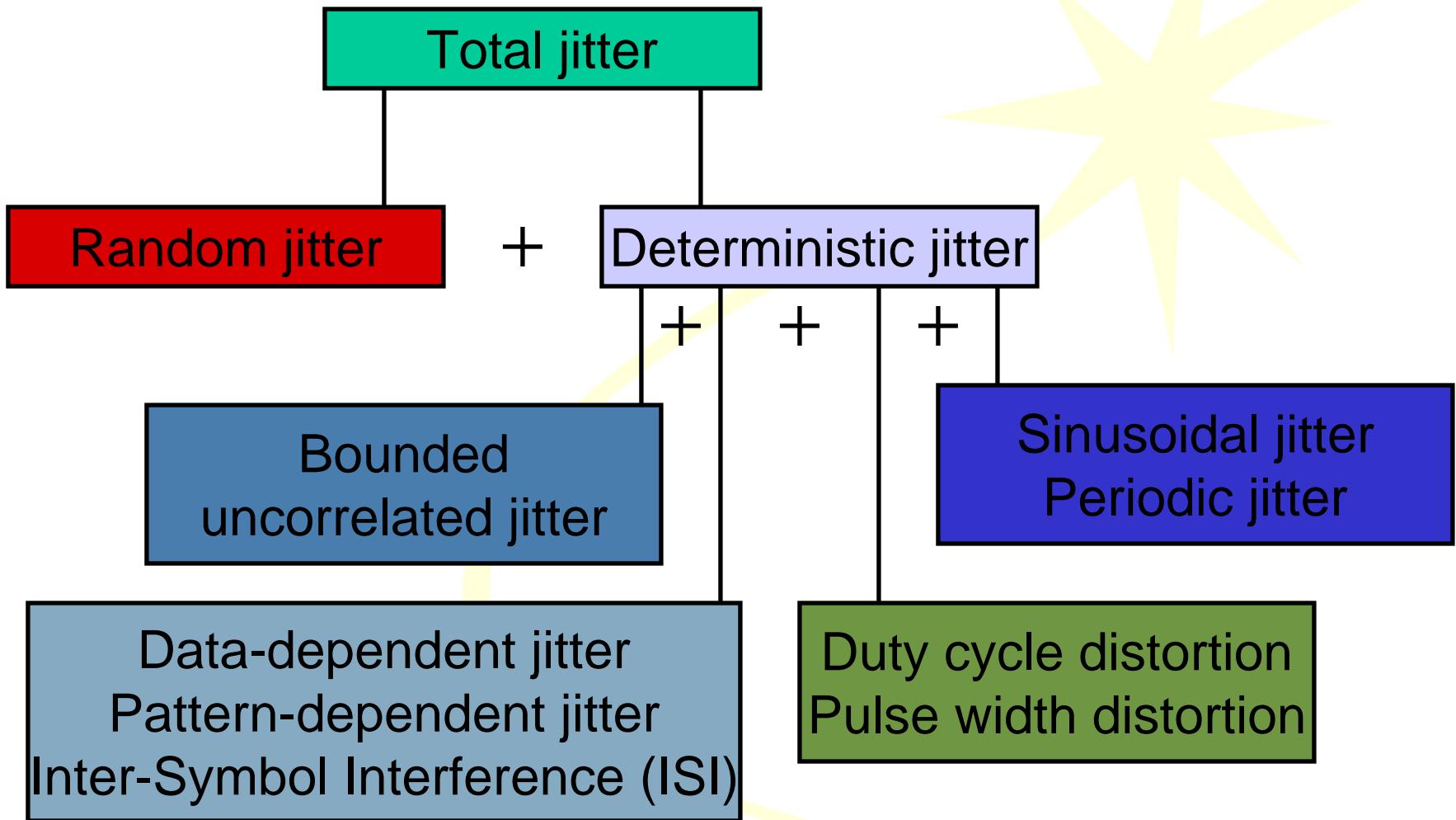
# Tx, Rx eye mask (423)

Characteristic	3 Gbps at IR, CR, XR
2 x Z2	1600 mV (P-P)
2 x Z1	275 mV (P-P)
X1	0.275 UI
X2	0.50 UI

Characteristic	SATA 1.5 Gbps
2 x Z2	600 mV (P-P)
2 x Z1	325 mV (P-P)



# Jitter types



# Jitter definitions

Type	Definition
<b>Total jitter</b>	Deviation of an event from its ideal time. Total jitter = Deterministic jitter + random jitter.
<b>Deterministic jitter</b>	Jitter component bounded in amplitude. Includes the following types:
<b>Duty-cycle distortion</b> <b>Pulse-width distortion</b>	Difference between the duty cycle (bit period) of a 1 bit and a 0 bit.
<b>Data-dependent jitter</b> <b>Pattern-dependent jitter</b> <u>Inter-symbol interference (ISI)</u>	Jitter that occurs on non-clocklike waveforms when the data path's bandwidth limitations distort the waveform edge locations.
<b>(Applied) Sinusoidal jitter</b> <b>Periodic jitter</b>	Jitter caused by crosstalk of signals in the system related to the data pattern. Applied sinusoidal jitter is a test applied for jitter tolerance testing in the receiver.
<b>Bounded uncorrelated jitter</b>	Jitter that does not correlate to the data pattern but is bounded. (Bounded because causes are known and controlled, such as data patterns. Unbounded would have a statistical chance of infinite jitter.)
<b>Random jitter</b>	Jitter that is unbounded in amplitude and follows a Gaussian probability distribution. Mean is 0; standard deviation expressed in rms; infinite tails.

# Jitter requirements

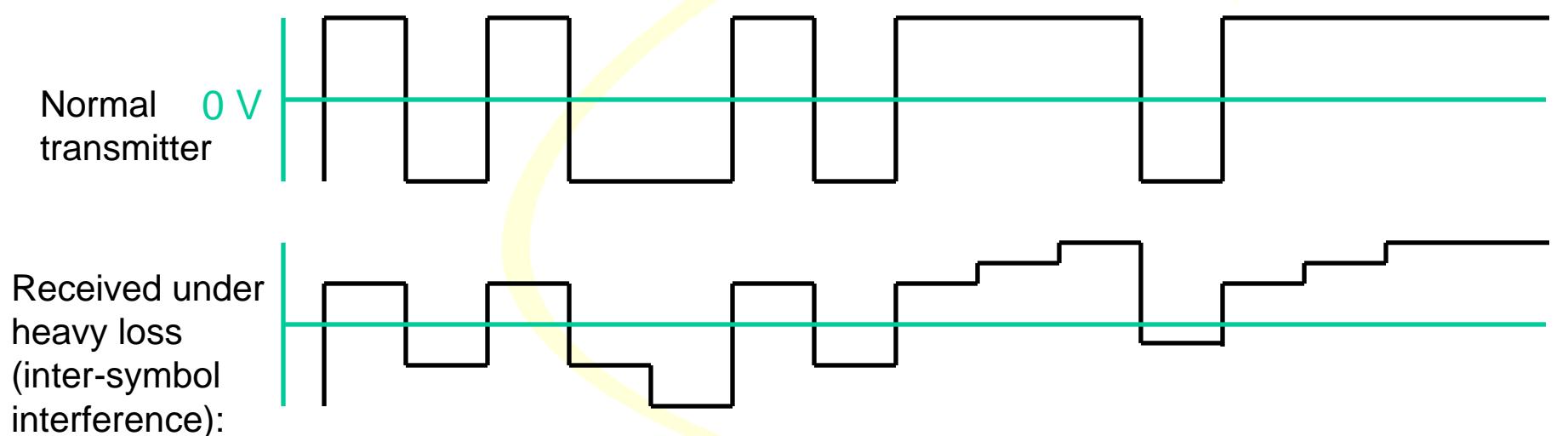
- Receivers must tolerate specified amounts of jitter using CJTPAT (Compliance Jitter Test PATtern)

Characteristic	Value at 3 Gbps	Notes
Sinusoidal	0.10 UI	Caused by crosstalk of related signals
Deterministic *	0.35 UI	Duty-cycle distortion, data-dependent jitter (inter-symbol interference), sinusoidal jitter, and uncorrelated jitter (unrelated signals)
Total allowed at receiver compliance point *	0.55 UI	Deterministic + random
Total that receivers shall tolerate *	0.65 UI	Deterministic + random

\*Considered after applying a high-pass function that attenuates low frequency jitter

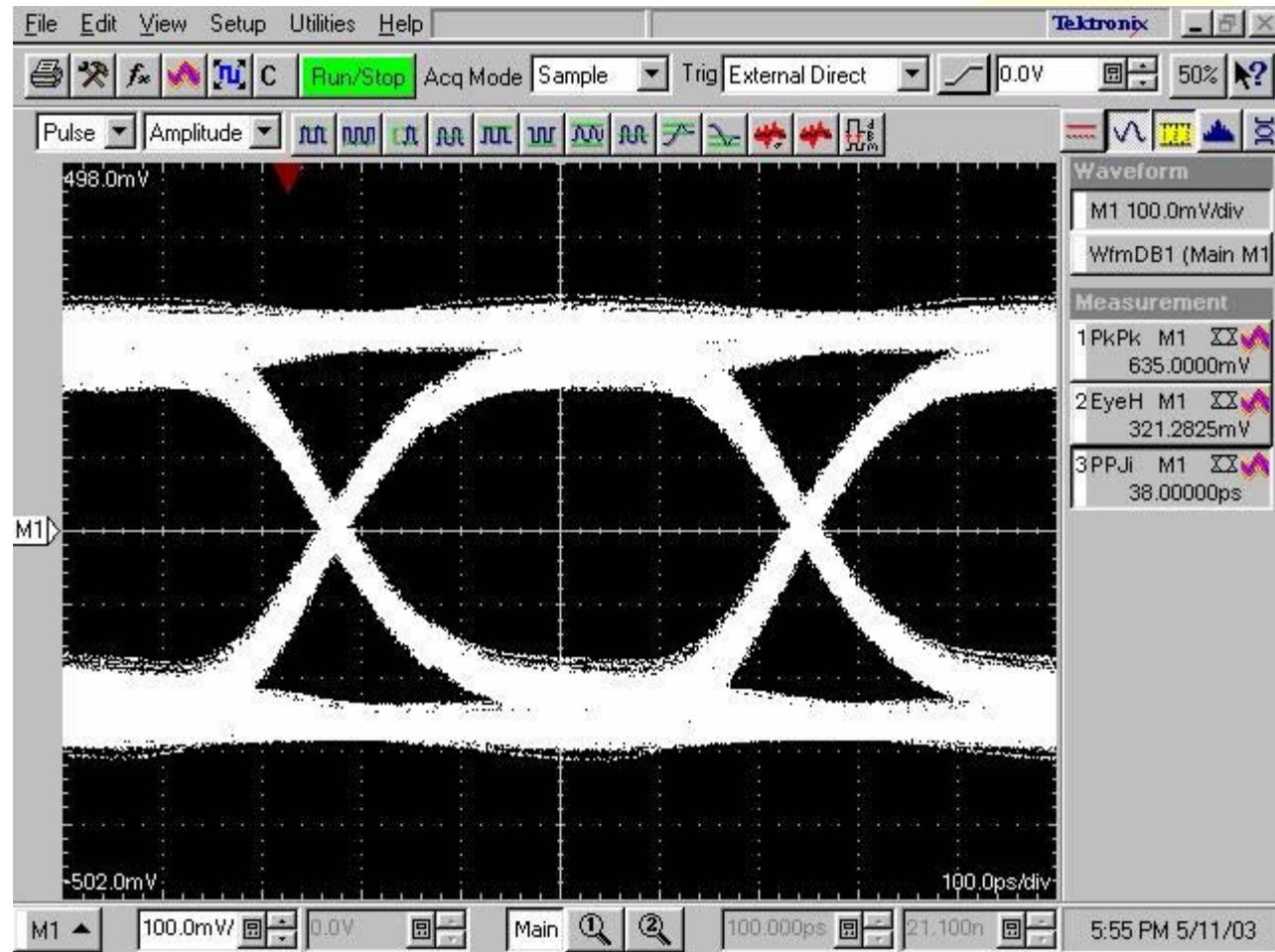
# Effects of interconnect on signals

- The interconnect introduces more loss at high frequencies than low frequencies
  - View as if the wire takes time to fully “charge up”
- Conductor and dielectric losses
- Attenuation and phase shift



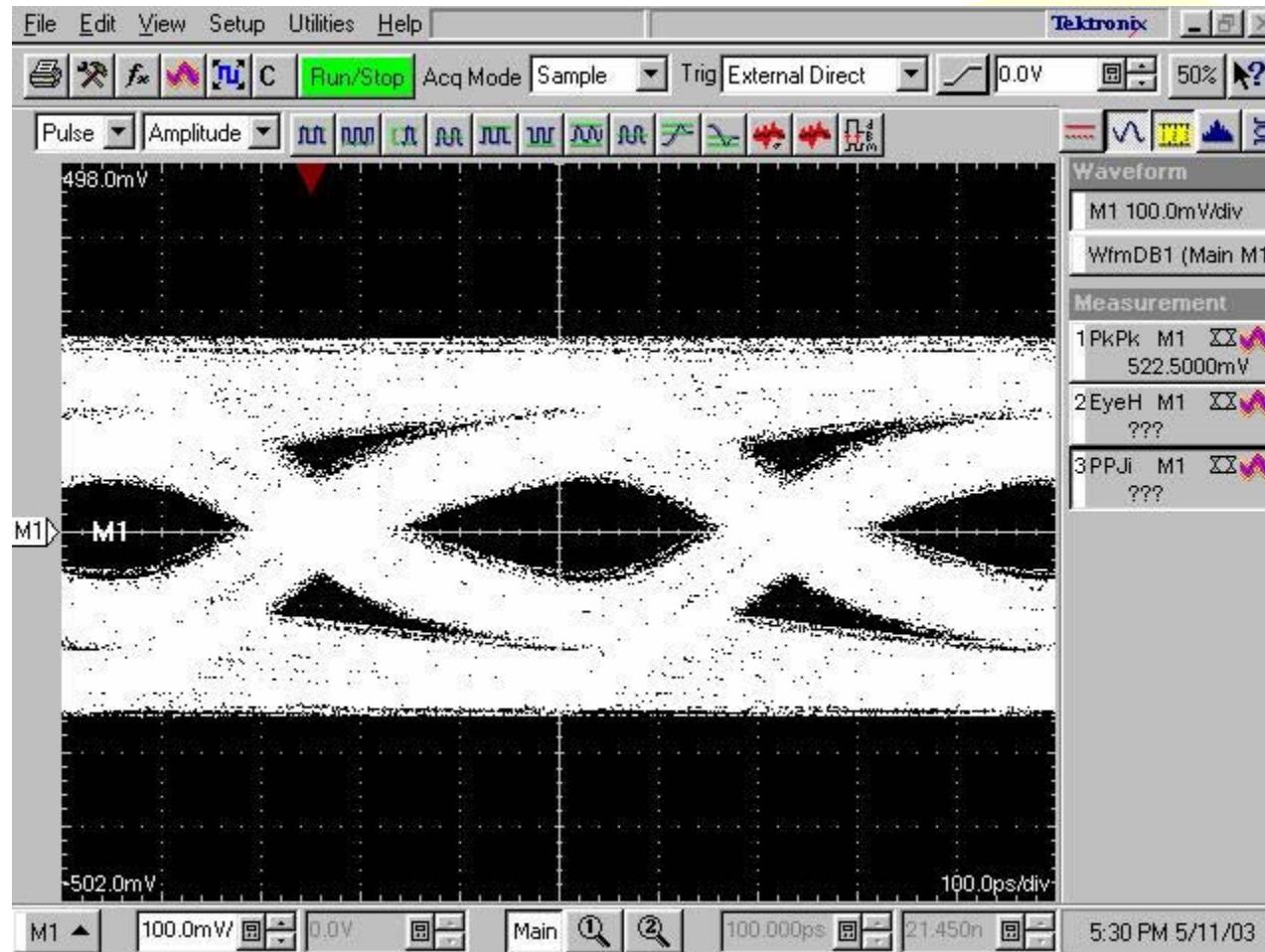
# Interconnect loss – short cable (424)

- After going through a 1 m cable



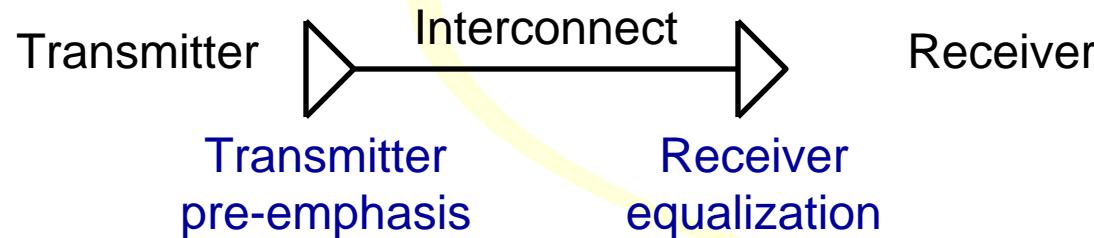
# Interconnect loss example – long cable

- After going through a 10 m cable



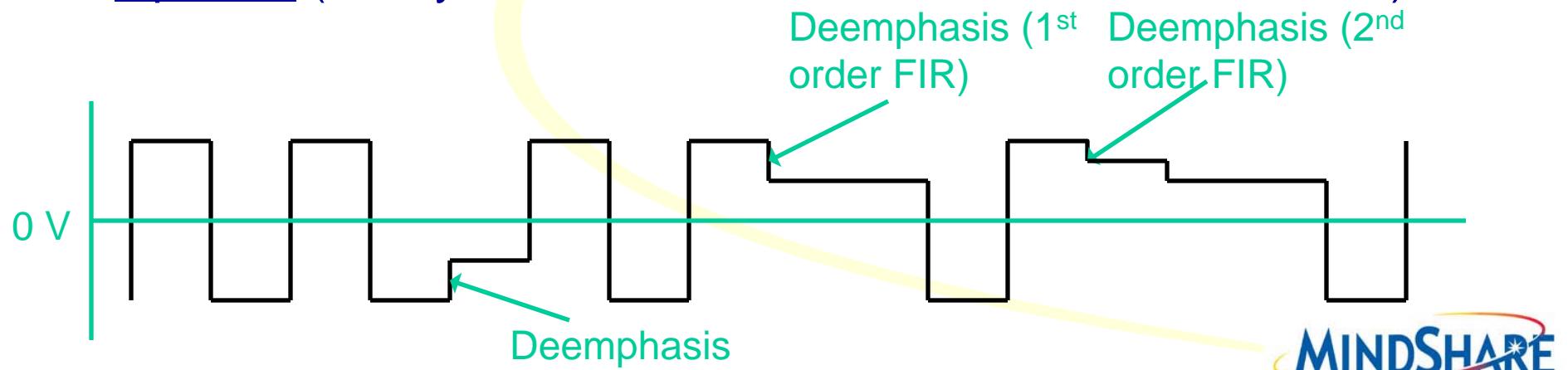
# Signal Compensation (425)

- Transmitter pre-emphasis (or de-emphasis)
  - Distort the signal going into the interconnect to counter expected loss effects
- Receiver equalization
  - Change receiver interpretation of signal to compensate for loss on the wire

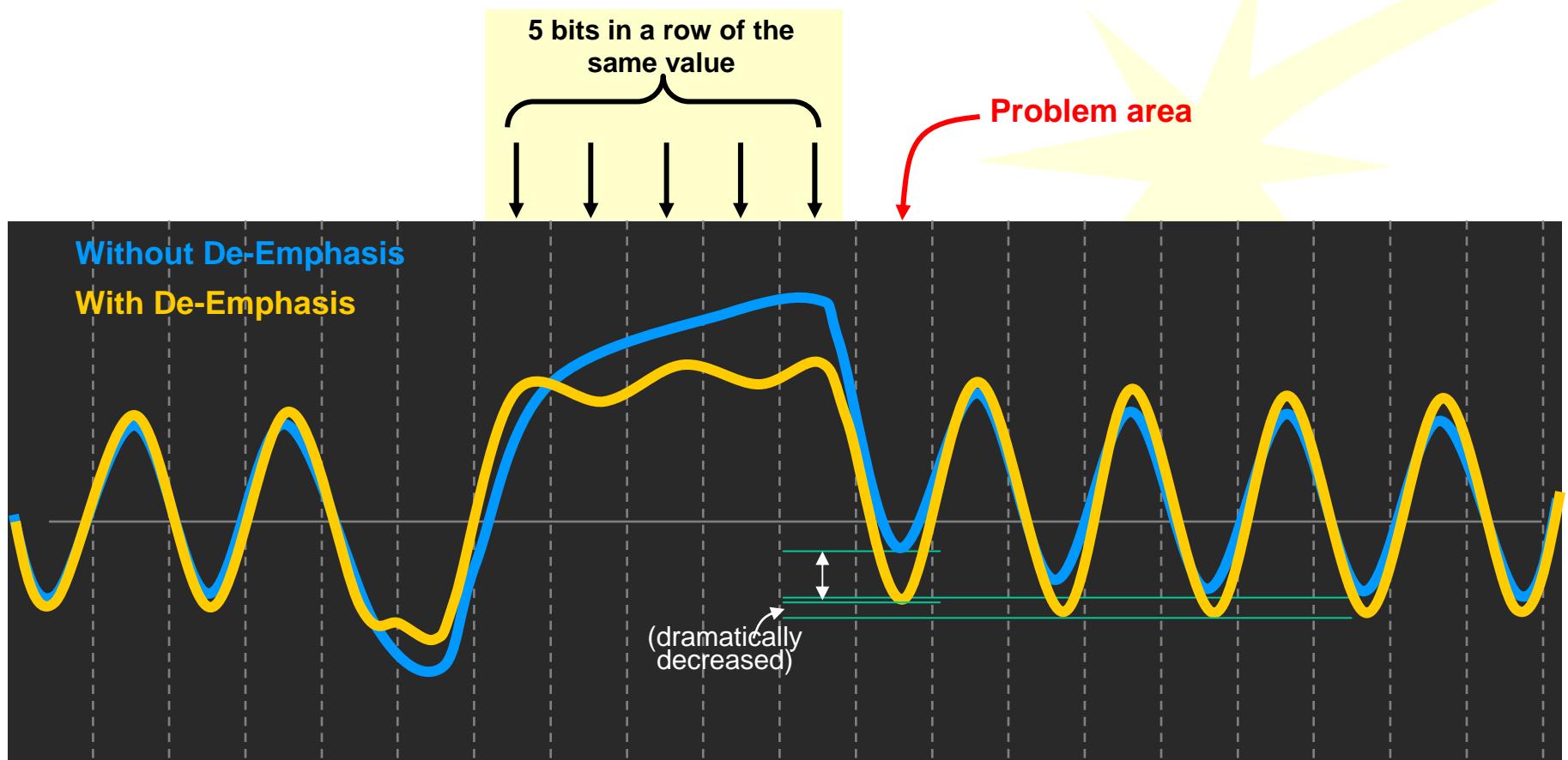


# Transmitter De-emphasis

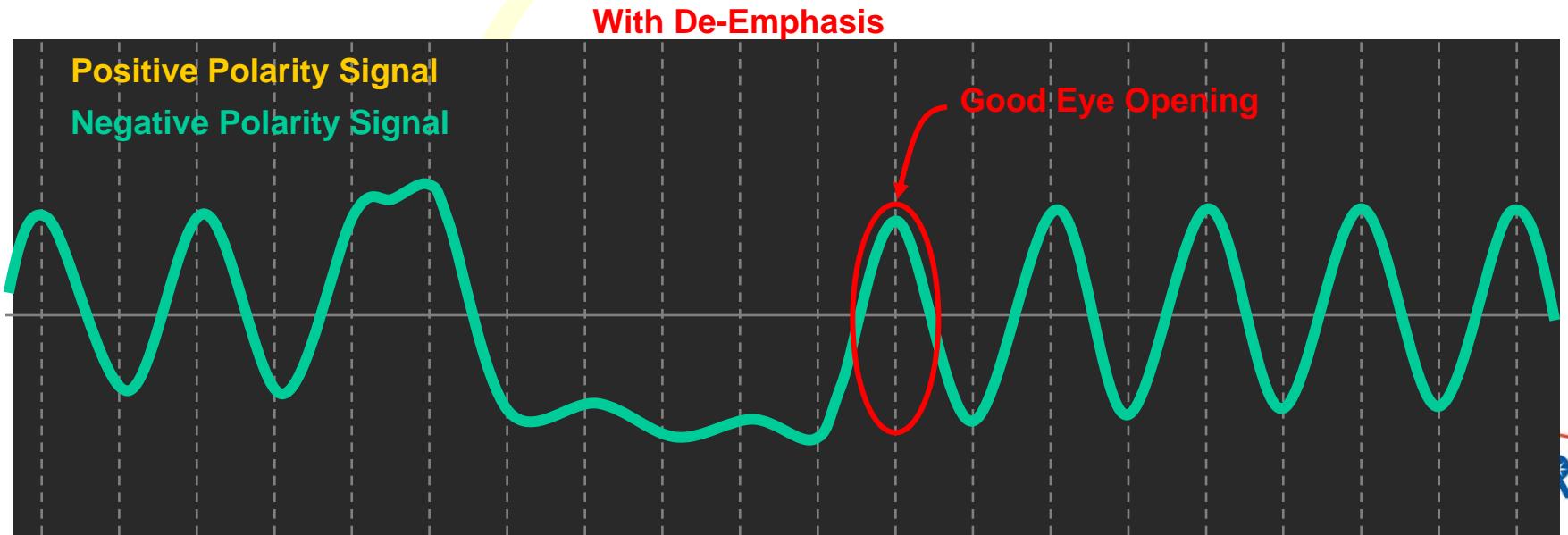
- Attenuate lower frequencies to counter cable loss
- Drive signals full strength after transition
- Drive signals more weakly (down by 40%) when consecutive bits stay the same (also called de-emphasis)
- Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters
  - FIR = based on historical input
  - IIR = based on historical input and output
  - More complex filters better counter the cable loss effects
- Optional (barely mentioned in SAS, not mentioned in SATA)



# ISI (Inter-Symbol Interference)

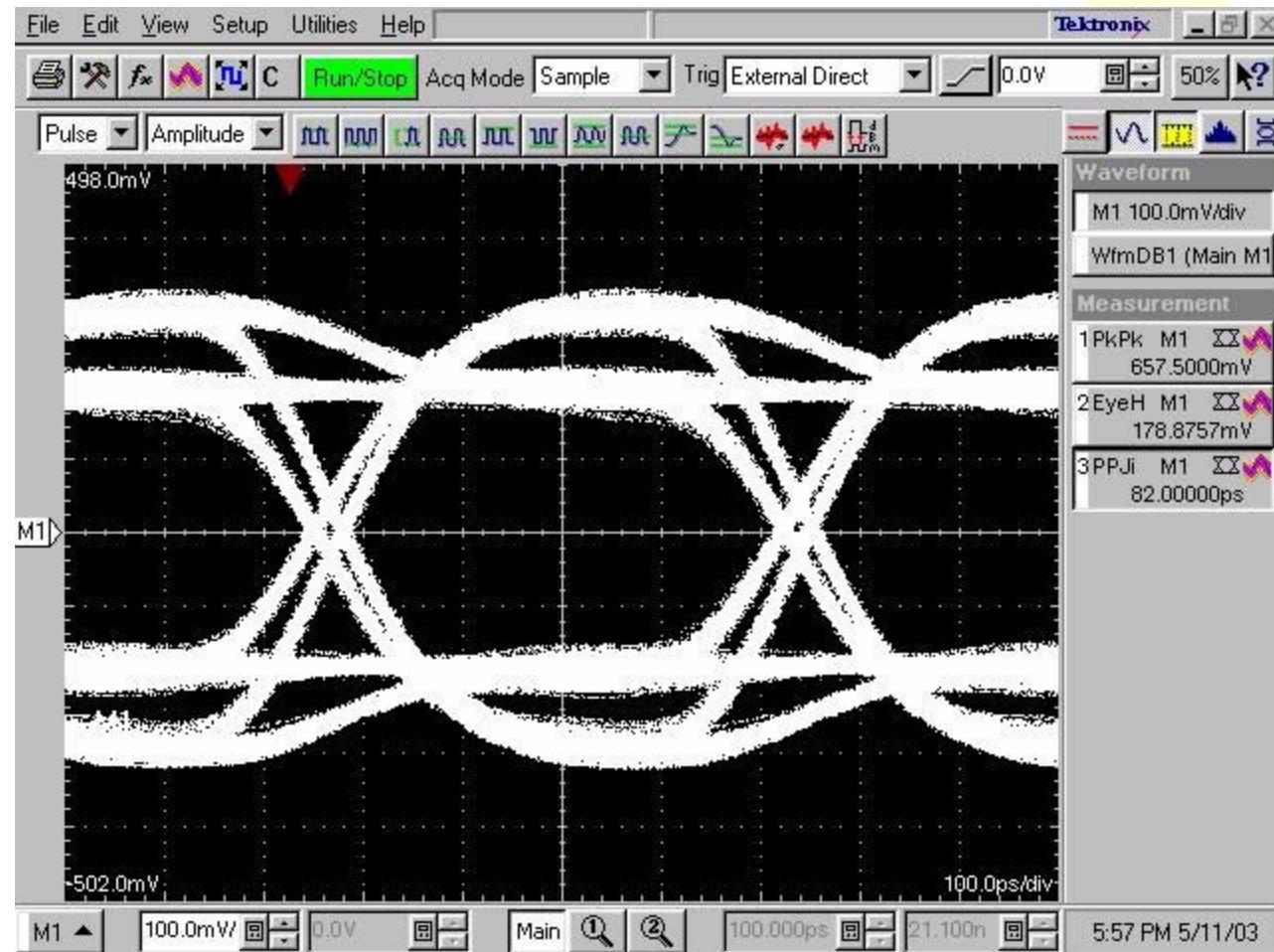


# Effects of ISI



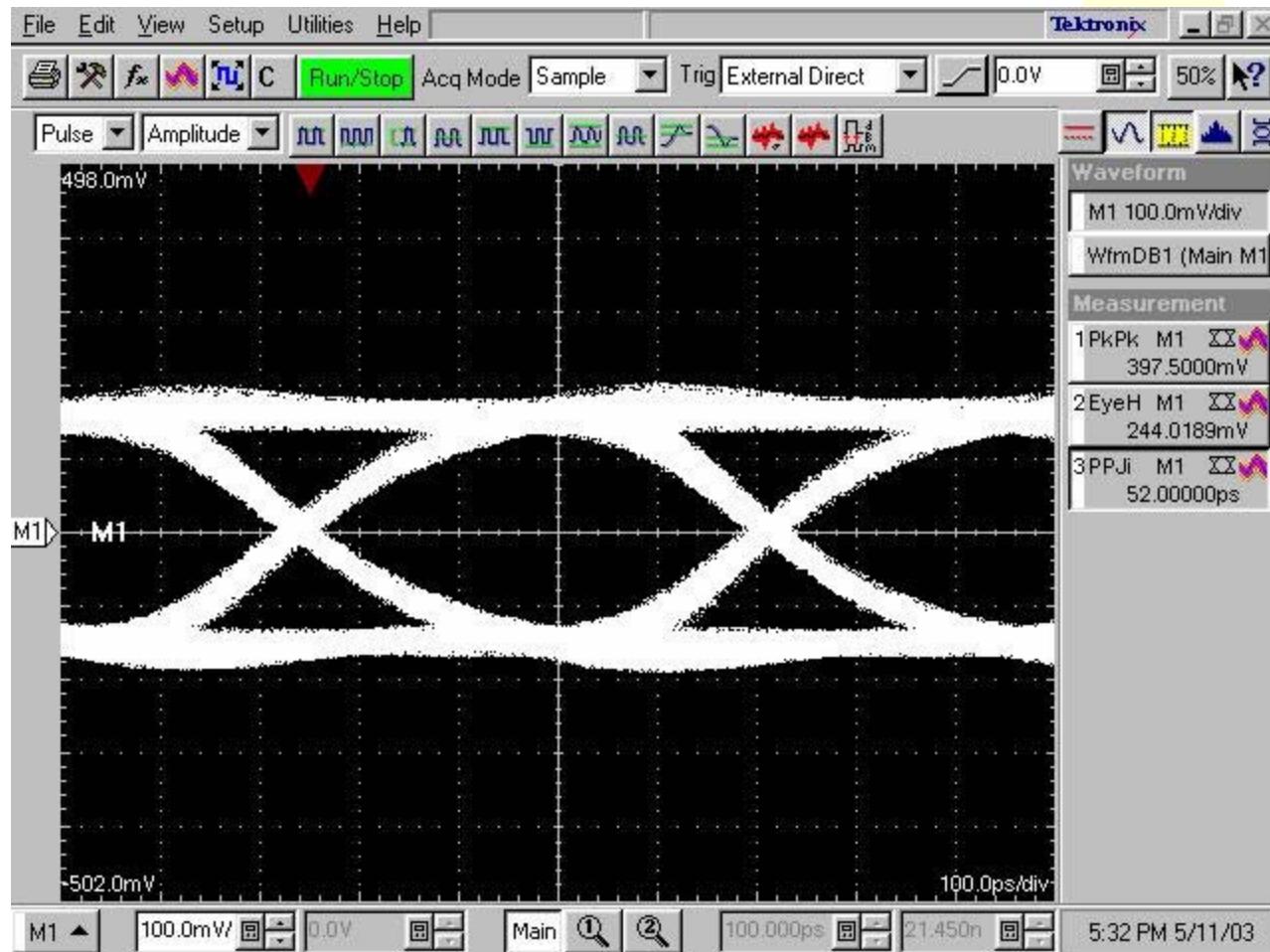
# De-emphasis example – short cable

- Using 1 m cable, injected distortion actually results in worse signal at receiver



# De-emphasis example – long cable

- In a 10 m cable, transmission line loss is canceled by the de-emphasis



# Programmable Compensation (427)

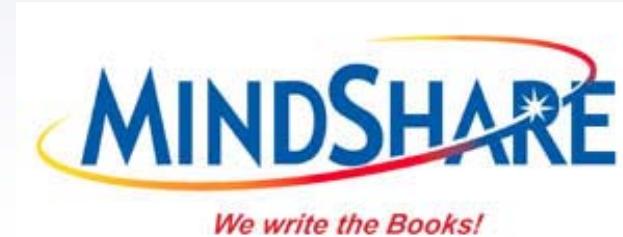
- Many designs have registers that can be programmed to select which pre-conditioning values to use, for high, or medium, or low-loss environments.
- HBA example:
  - external connection can use de-emphasis of 30%
  - internal connection might use only 10%
  - very short controlled path would not want any de-emphasis
- Another option: maintain BER (bit error ratio) counters that could indicate whether the compensation setting needs to be changed

# Receiver equalization (428)

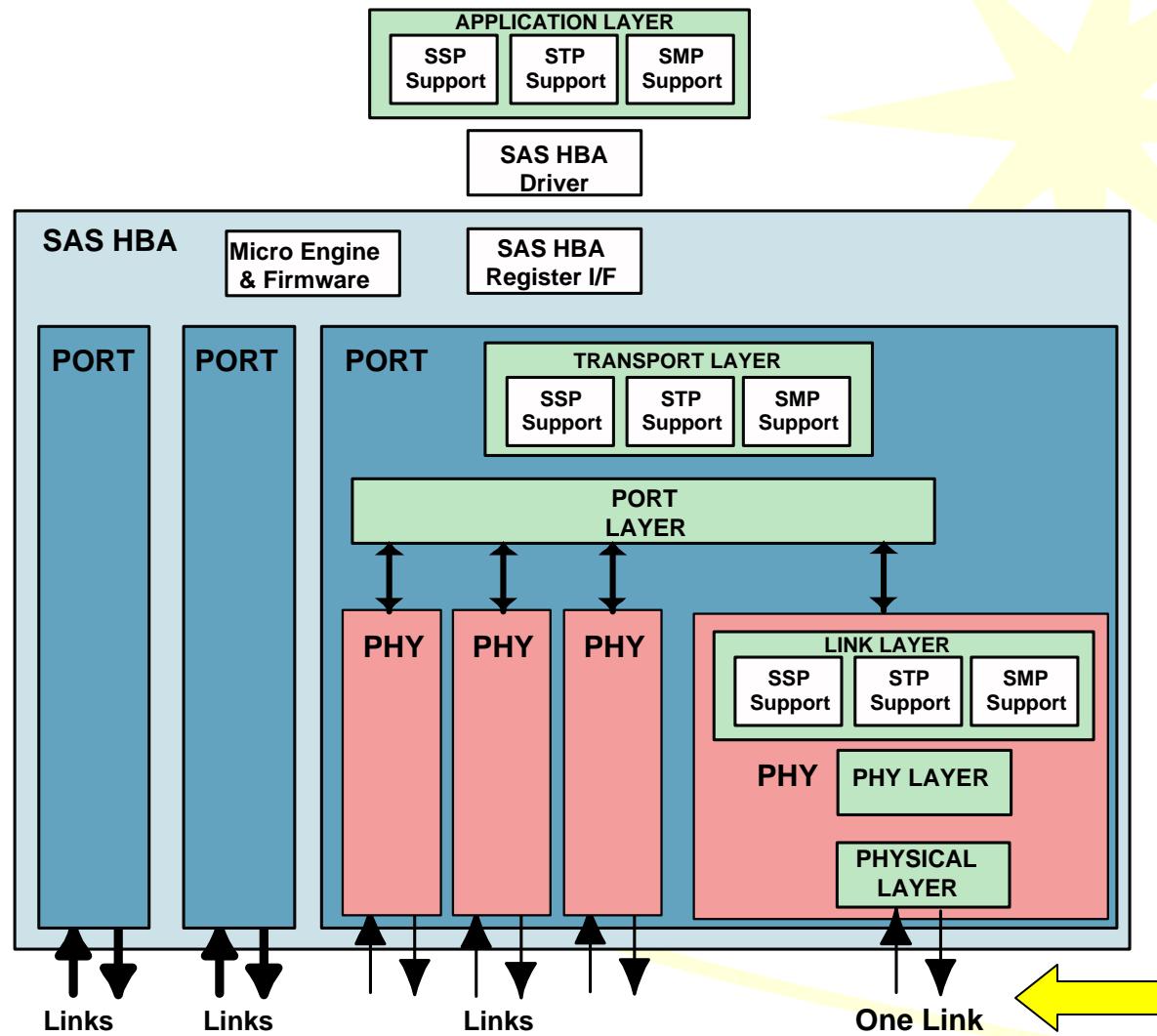
- Amplify (boost) higher frequencies near 1.5 or 3.0 Ghz to counter cable loss
- Filter out lower frequencies
  - Pro – can adapt to environment, even recover signals when no eye is present at the receiver
  - Con – amplifies noise, cannot observe receiver behavior
- Optional (barely mentioned in SAS standard)

# *Chapter 18*

## *Cables and Connectors*

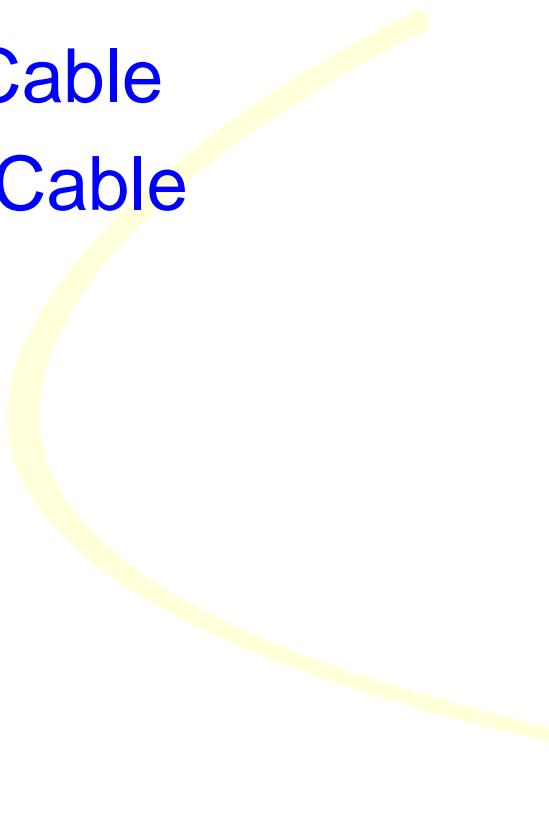
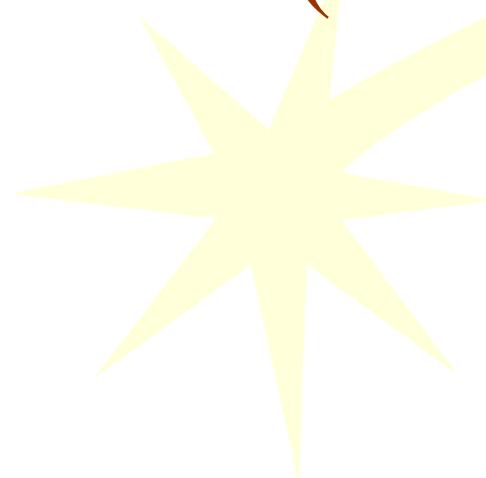


# SAS Layer Review (366)

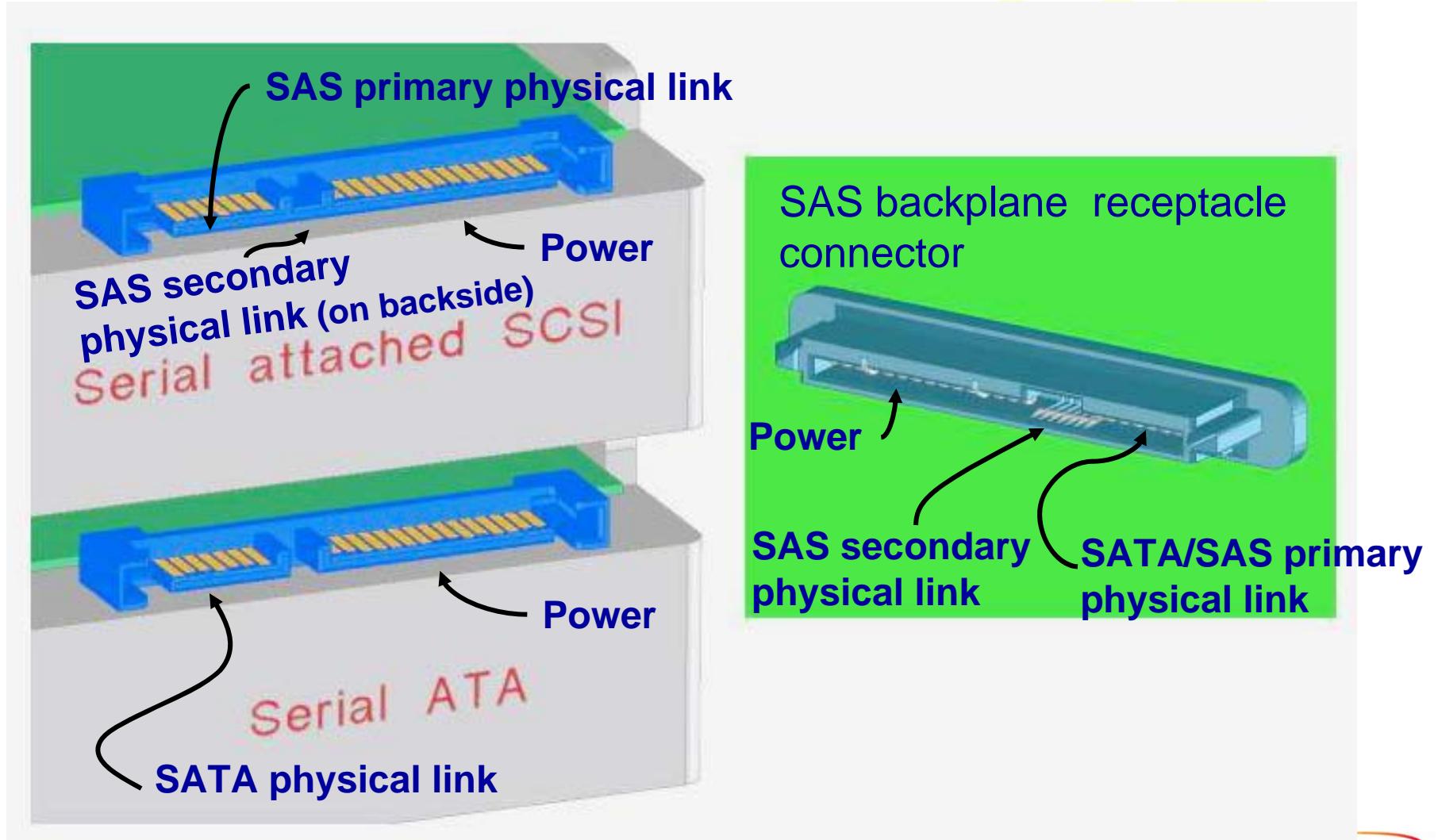


# Interconnect Environments (434)

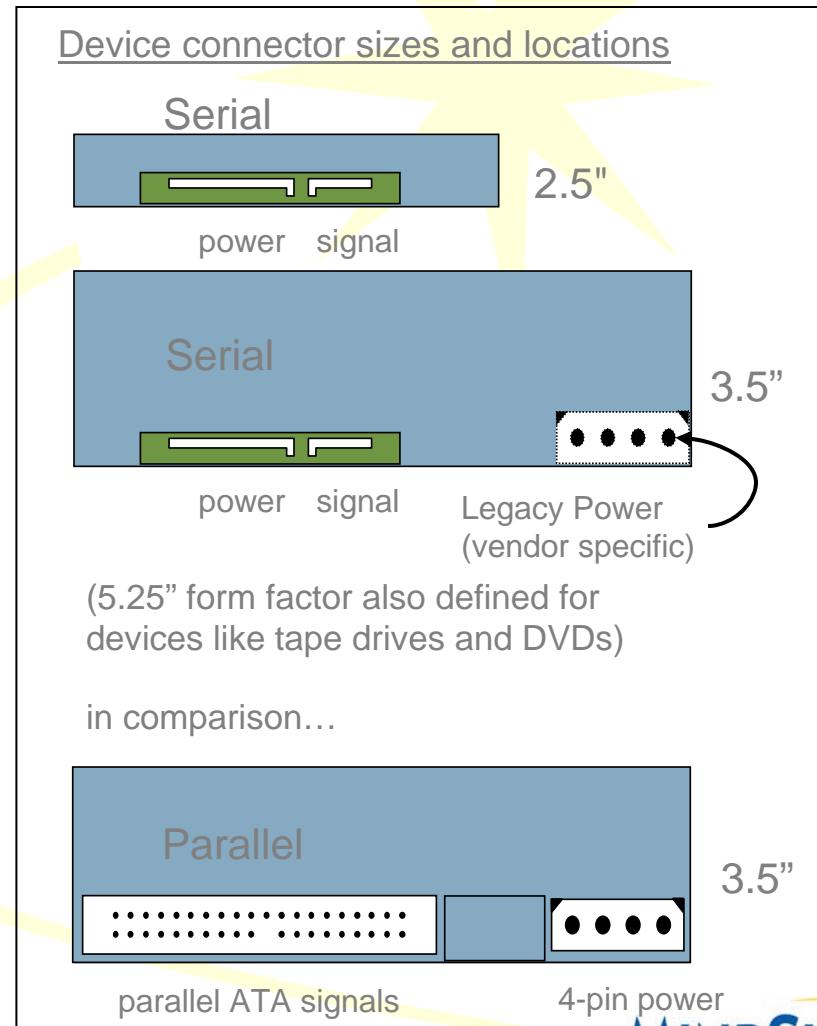
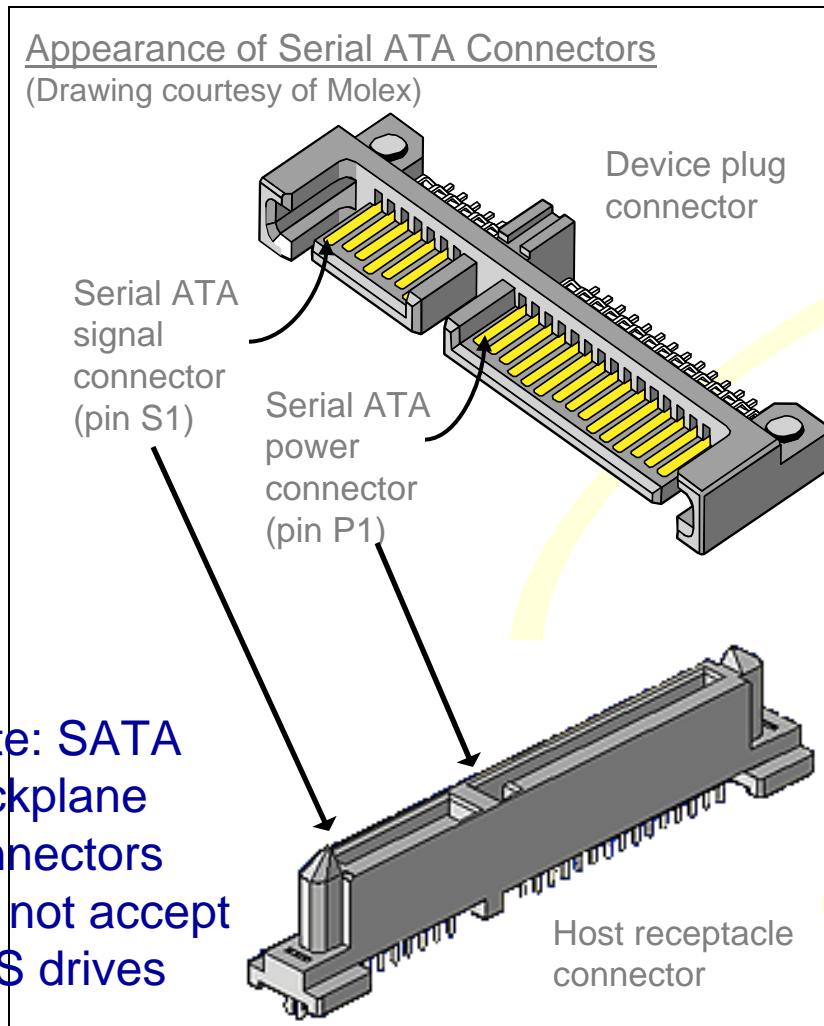
- Four defined for SAS
  - Drive Backplane
  - Drive Cable
  - Internal Cable
  - External Cable



# SAS Drive Plug and Backplane Connectors



# SATA Drive Plug and Backplane Connector (436)

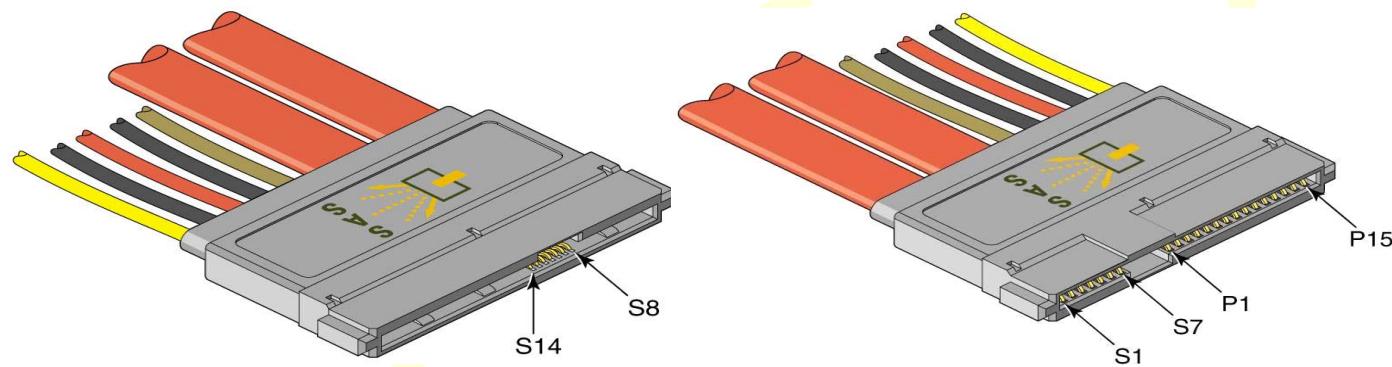


# Connector Finger Lengths

- The connector fingers have two different lengths, allowing for three connect times.
  - First mate (long to long) for grounds and “precharge” power pins
  - Second mate (long/short) for power pins
  - Third mate (short/short) for signals
- Goal is to guarantee certain microseconds settling time before signals make connection.

# Drive Cable Connector (438)

- Dual-ported cable design



# Internal connector signals (439)

- Secondary physical link is optional
- The cable or backplane connects Tx on one side to Rx on the other side

Device connector				
Pin	Backplane Receptacle	Drive Plug	Notes	
S1	GROUND		Primary phy	
S2	TP+	RP+		
S3	TP-	RP-		
S4	GROUND			
S5	RP-	TP-		
S6	RP+	TP+		
S7	GROUND			
S8	GROUND			
S9	TS+	RS+	Secondary phy (no-connects on SAS single- phy drives, SATA drives, and narrow cables)	
S10	TS-	RS-		
S11	GROUND			
S12	RS-	TS-		
S13	RS+	TS+		
S14	GROUND			

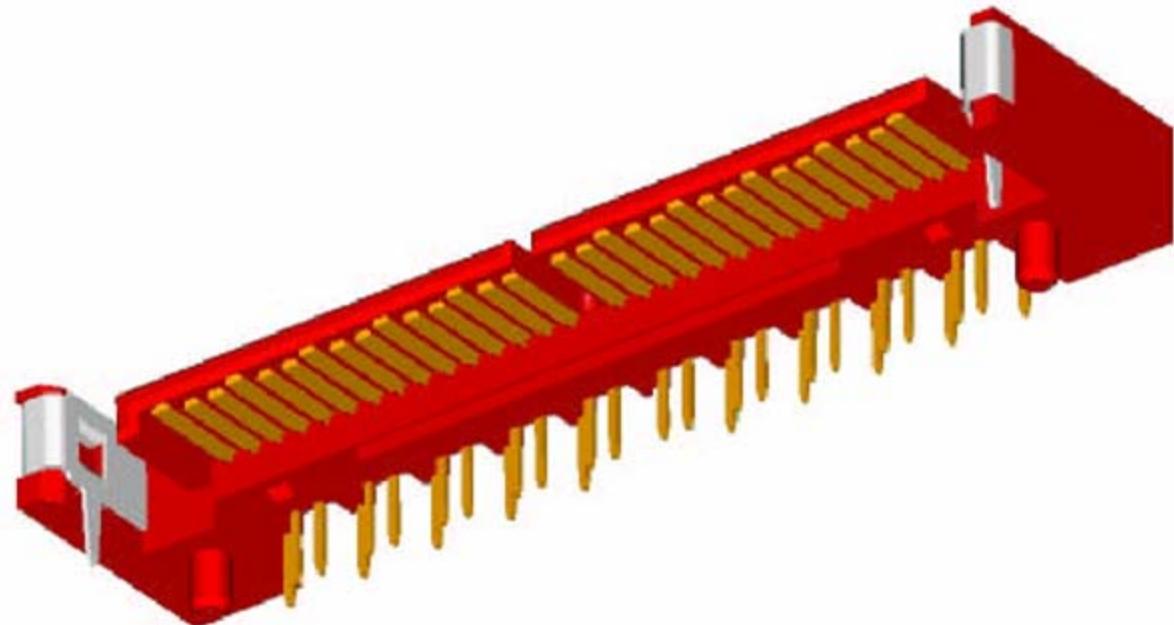
# Internal device connector power assignments

- Three voltages are provided
  - 3.3 V (new for serial interfaces)
  - 5 V
  - 12 V (for drive motors)
- All the pins of one voltage are tied together by the target device
- Precharge pins are longer
- Signal cables do not carry power
  - Separate power cable
  - Converts cables from old large power connector to the SATA hot plug capable power connector
- SFF discussing 12 V only drives

Pin	Signal
P1	V3.3
P2	V3.3
P3	V3.3, precharge
P4	GROUND
P5	GROUND
P6	GROUND
P7	V5, precharge
P8	V5
P9	V5
P10	GROUND
P11	READY LED
P12	GROUND
P13	V12, precharge
P14	V12
P15	V12

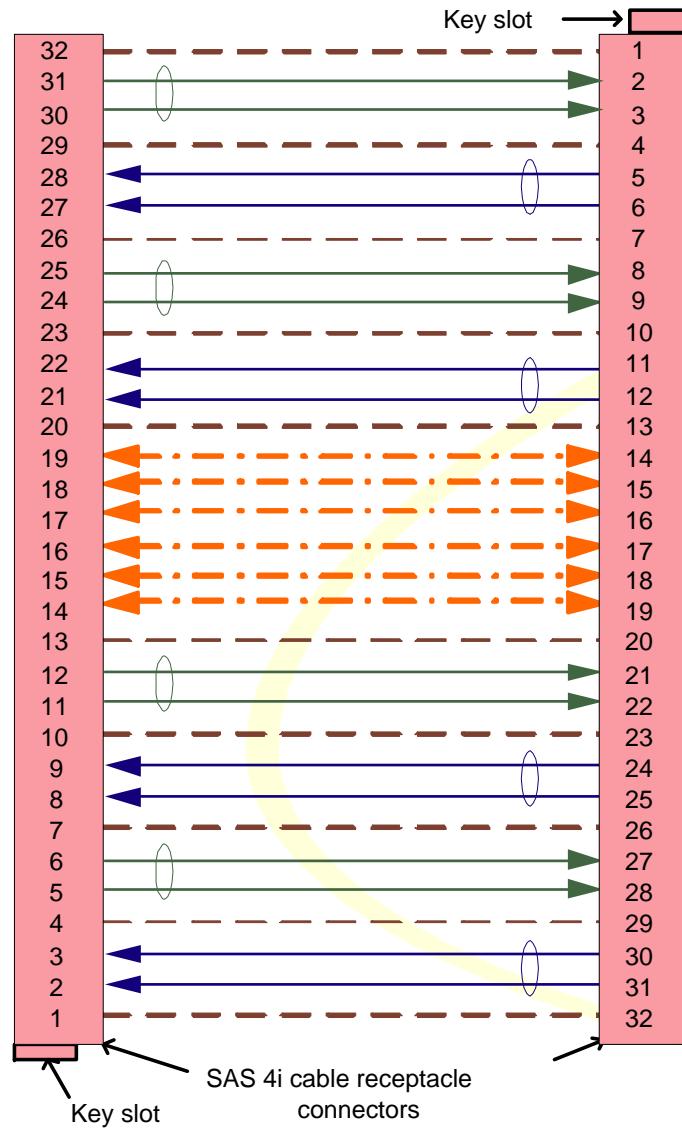
# SAS 4i Connector (441)

- Essentially 5 SATA connectors side by side
- Middle pins available for sideband signals (e.g., I<sup>2</sup>C or vendor-unique signals)
- Supports round or ribbon cable
- No power pins; grounds between pairs



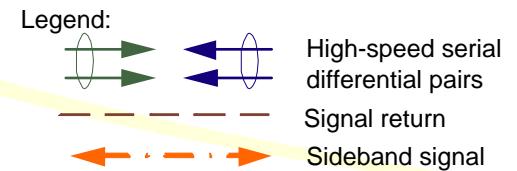
Pin	Signal
S1	Tx 0+
S2	Tx 0-
S3	Rx 0-
S4	Rx 0+
S5	Tx 1+
S6	Tx 1-
S7	Rx 1-
S8	Rx 1+
SB0	Sideband
SB1	Sideband
SB2	Sideband
SB3	Sideband
S9	Tx 2+
S10	Tx 2-
S11	Rx 2-
S12	Rx 2+
S13	Tx 3+
S14	Tx 3-
S15	Rx 3-
S16	Rx 3+

# SAS 4i Internal Symmetric Cable (442)

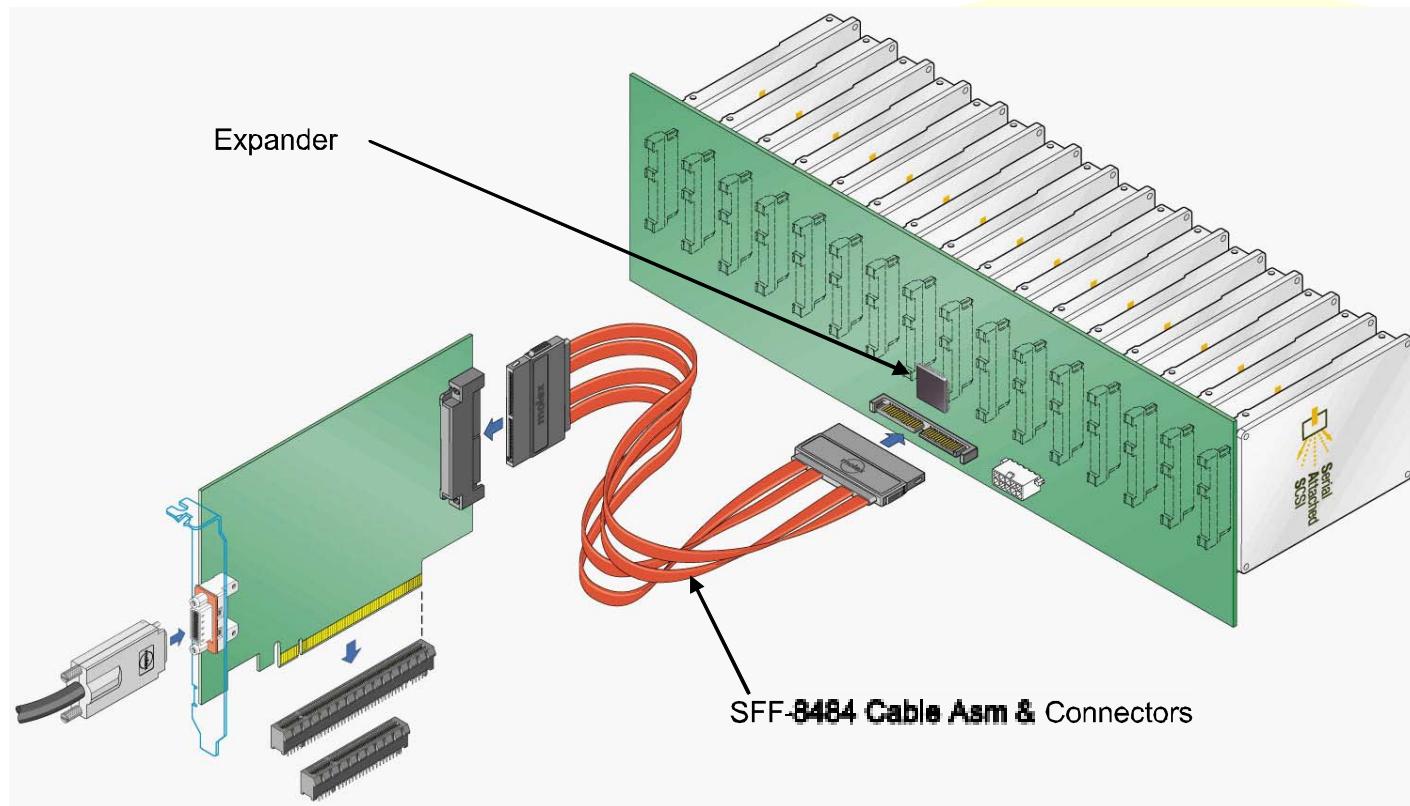


Note that signals are flipped from one side to the other, expecting target to connect to a controller.

Two controllers can still connect as long as they use more than 2 of the links. If each only used links 0 and 1, they wouldn't be able to see any device connected.

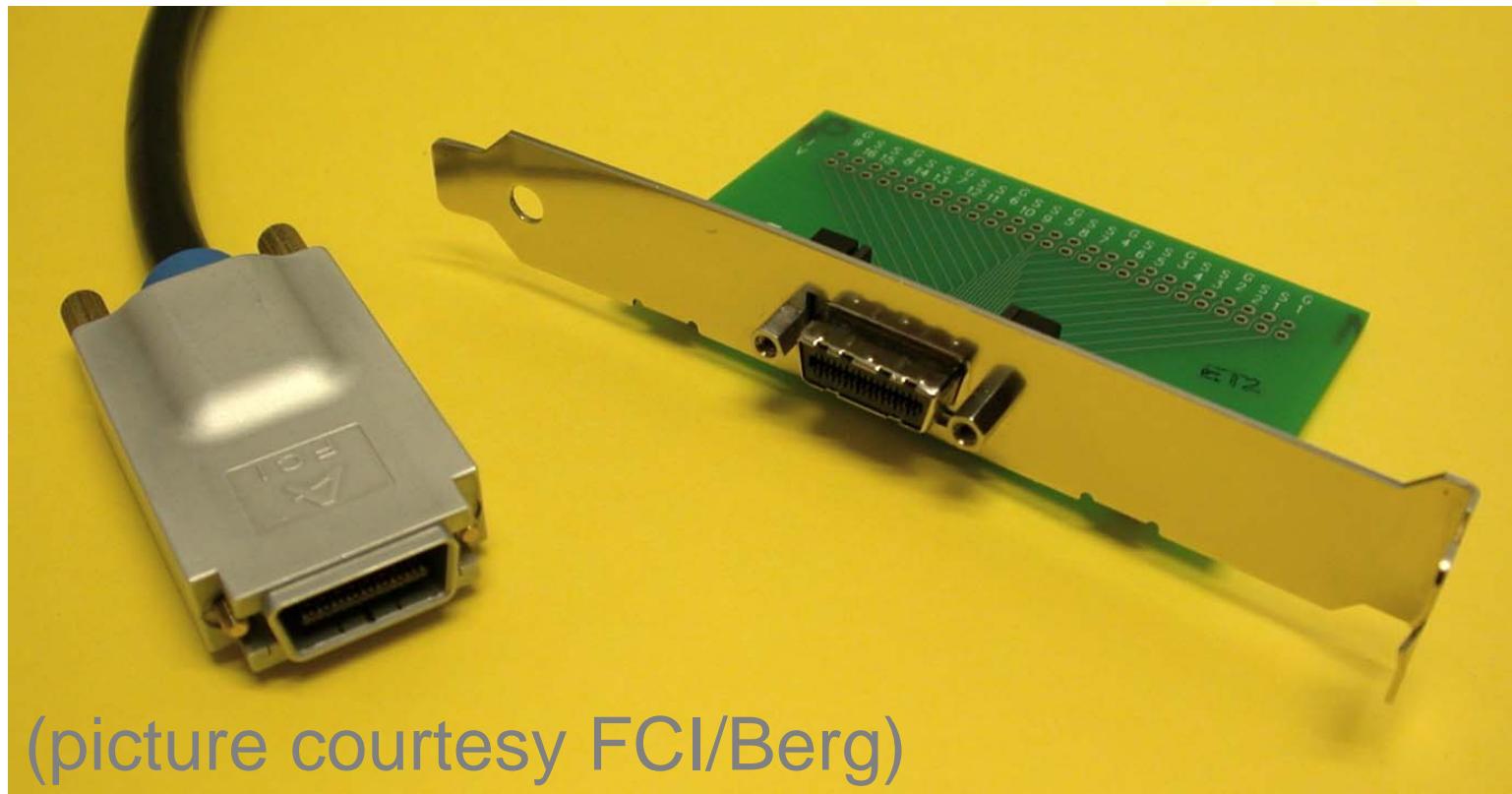


# SAS 4i Internal Symmetric Cable Example (443)



# SAS External Connectors – 4x

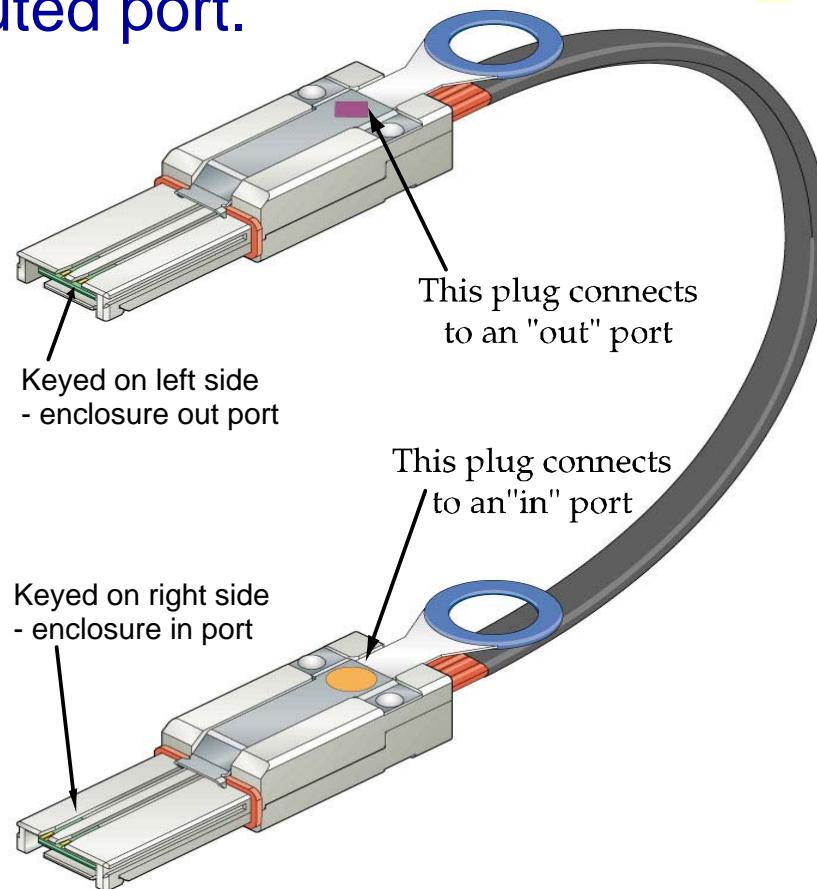
- SAS defines two external (box-to-box) cables. The SAS 4x connector uses the old InfiniBand 4x connectors and cables



(picture courtesy FCI/Berg)

# SAS Mini 4x Connector (449)

- Mini connector includes keying to indicate which end should be plugged into a connector. The “out” plug goes to a table-routed port, while the “in” plug goes to a subtractive-routed port.



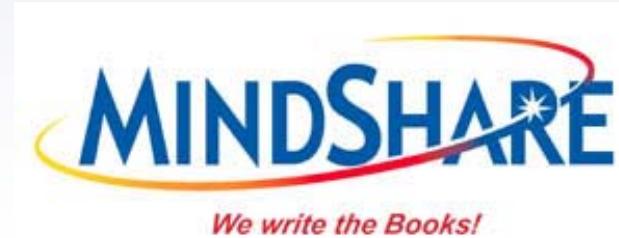
# Connector standards

- Connector pinouts and dimensions are defined for various drives by the Small Form Factor committee (SFF)
- Some relevant definitions are shown here; see [www.sffcommittee.org](http://www.sffcommittee.org) for the entire list

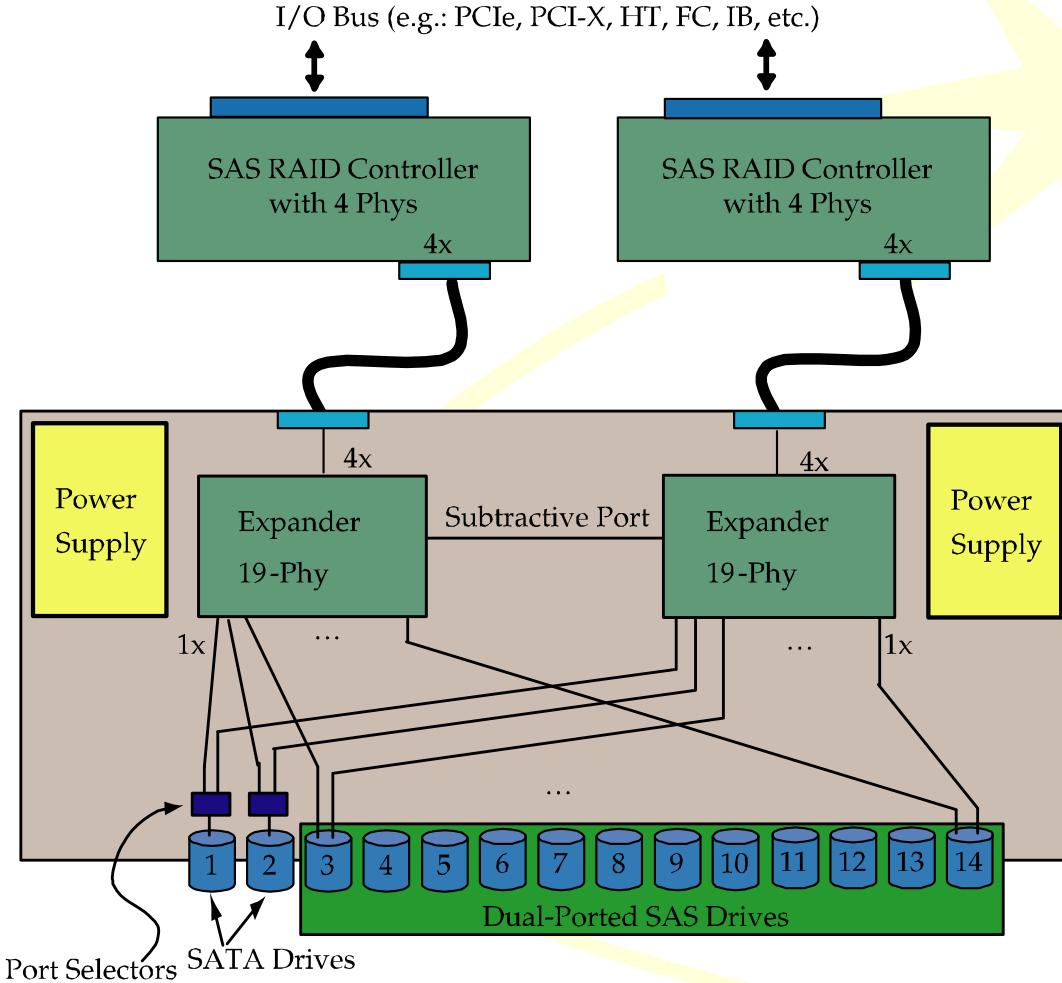
Standard	Description
SATA	SATA device plug and SATA signal host plug SATA host receptacle and SATA signal cable receptacle
SFF-8482	SAS plug (2x), SAS internal cable receptacle (1x or 2x), SAS backplane receptacle (2x)
SFF-8470	SAS external cable plug (4x),SAS external receptacle (4x)
SFF-8484	SAS internal wide plug and receptacle (4x)
SFF-8460	High-speed signaling backplane design guidelines
SFF-8223	2.5" form factor (connector location)
SFF-8323	3.5" form factor (connector location)
SFF-8523	5.25" form factor (connector location)

# *Chapter 19*

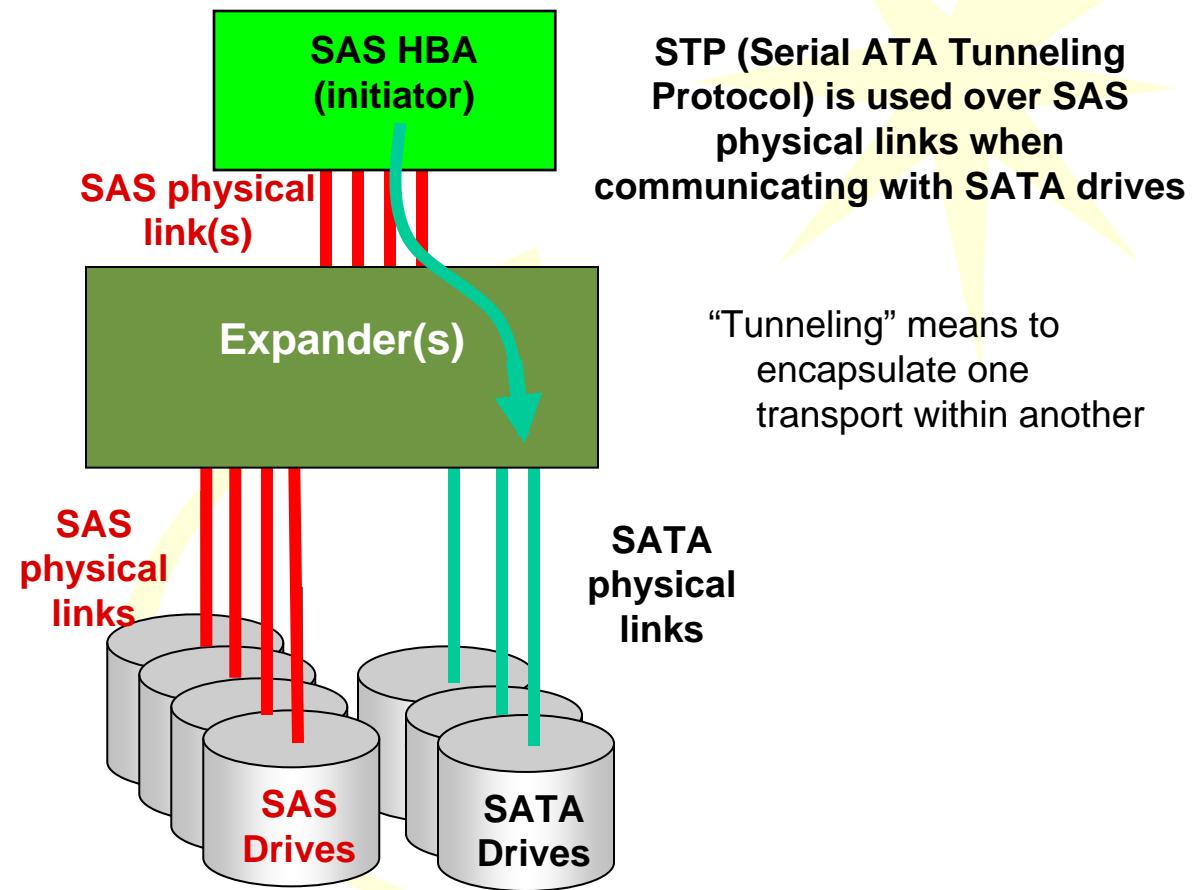
## *SATA Support*



# SATA Drives in SAS Fabric (459)

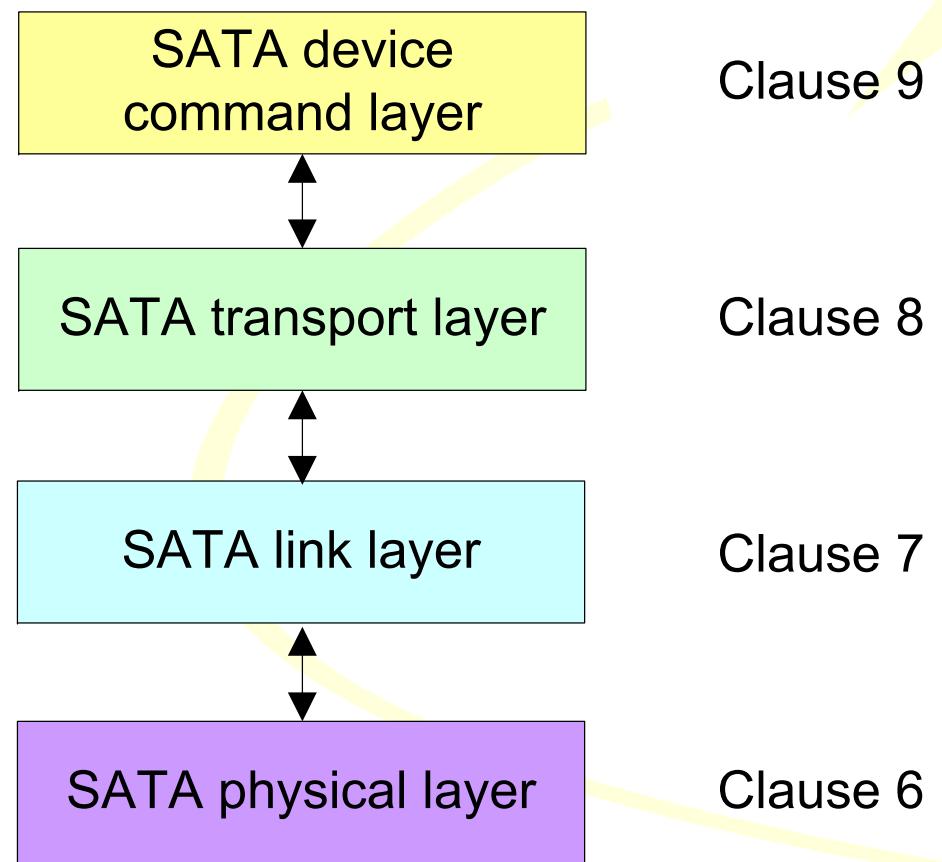


# Serial ATA Tunneling Protocol



# SATA Layers (458)

- For SATA 1.0a from the Serial ATA working group

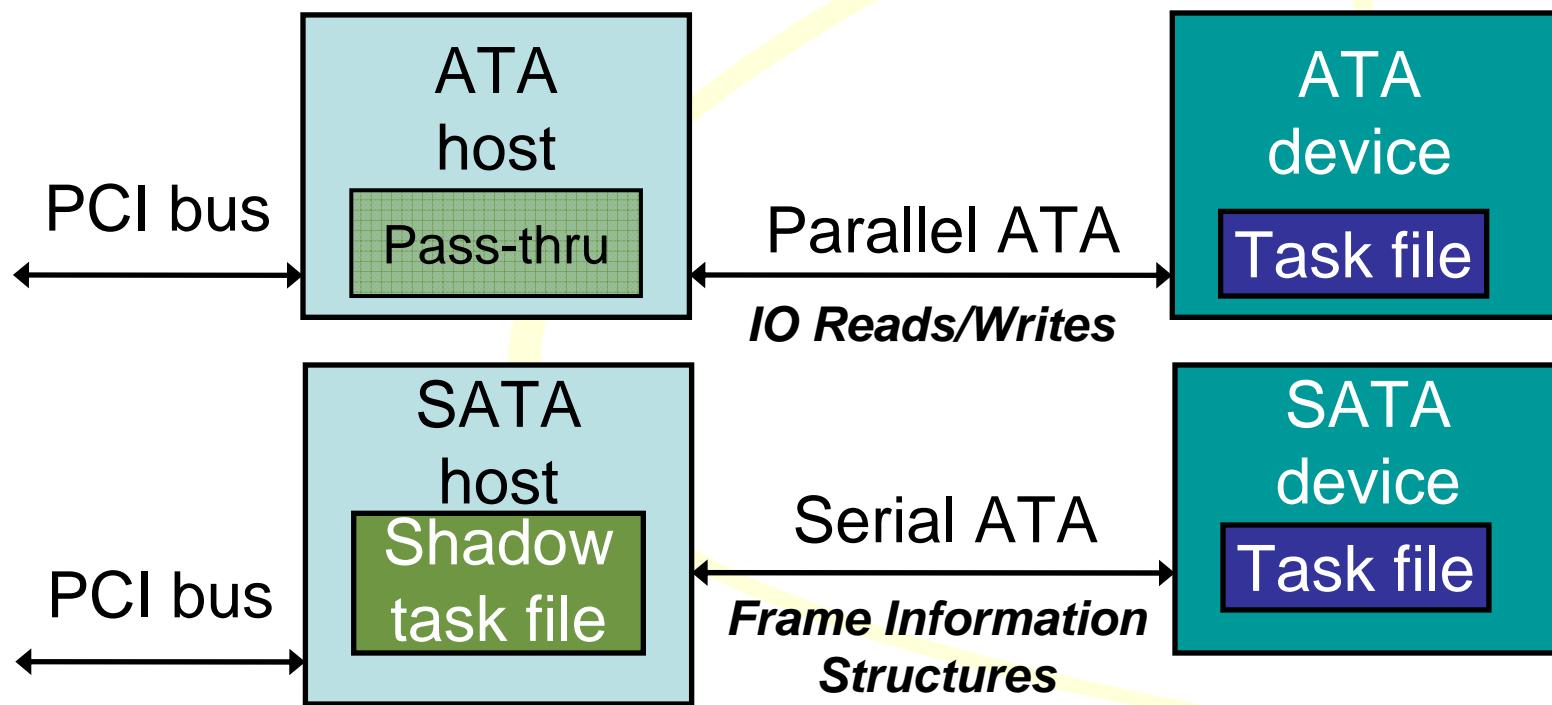


# STP Application Layer (462)

- The task file registers reside in this layer. In the host, this is the shadow task file, while in the target it's the drive registers.
- SATA supports two access methods
  - Legacy method
  - AHCI (Advanced Host Controller Interface) method

# Task File Location (457)

- The task file (set of registers) in the ATA device is read or written by the ATA host.
  - In parallel ATA, every access to these registers results in bus traffic
  - In serial ATA, a Shadow Task file register set within the host mirrors the ATA device's task file. This allows these register accesses to be grouped into a single, more efficient serial transfer.



# SATA Programming Interfaces

- Legacy Register Set
  - Software performs I/O writes to Control and Command Registers
  - DMA registers programmed to move data
  - Interrupt generated to call software to move data to or from memory, or to notify software that transfer has completed and status is available
- Advanced Host Controller Interface (AHCI)
  - Software creates command table in memory containing the SATA FIS's to be sent
  - Data structures also contain scatter/gather lists that specify the target memory locations

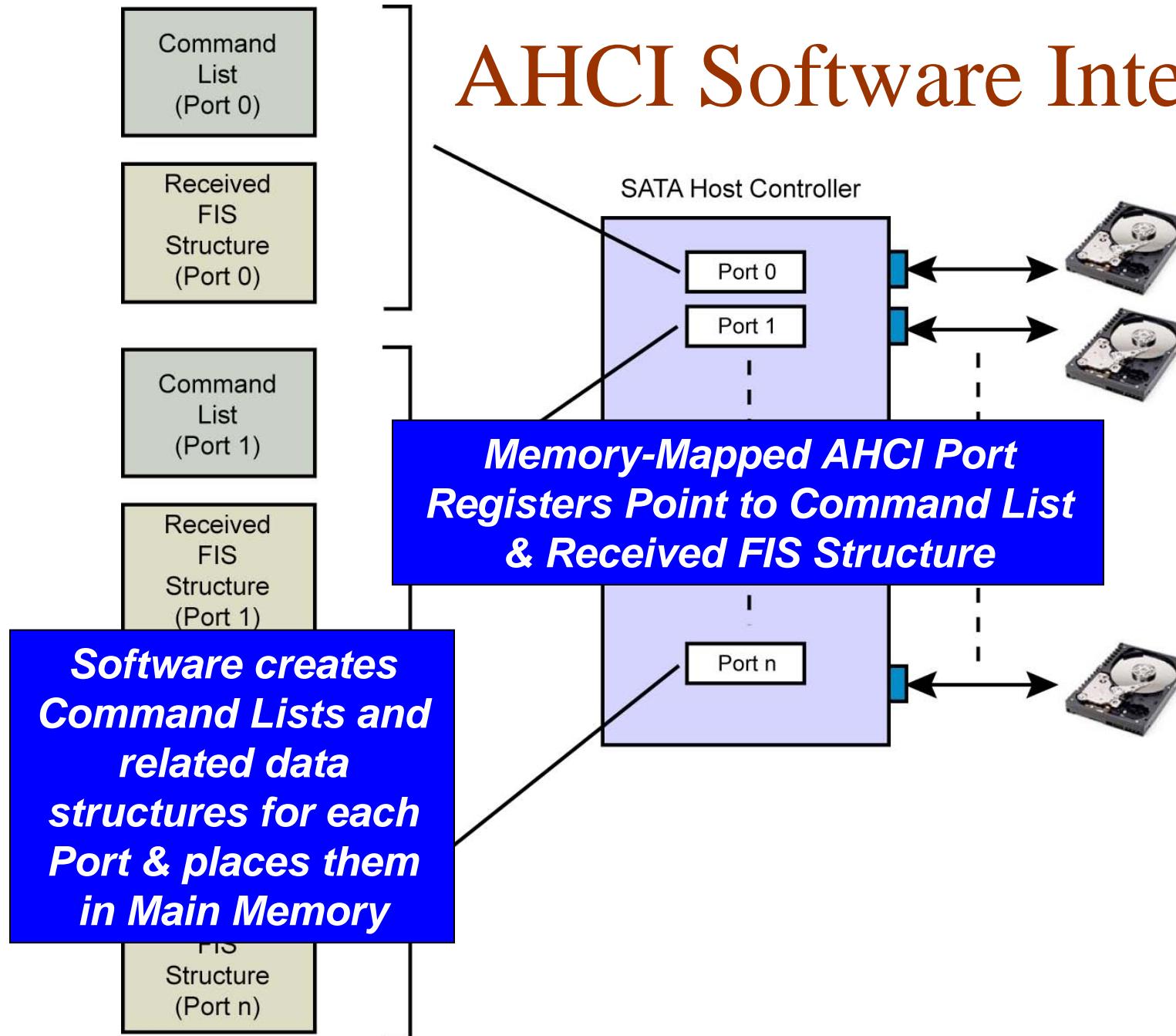
# Legacy Programming Interface

## [ATA Task File Registers]

- Processor IO registers defined for use with ATA - task file register interface
- ATA/ATAPI-7 Volume 1 defines the task file registers

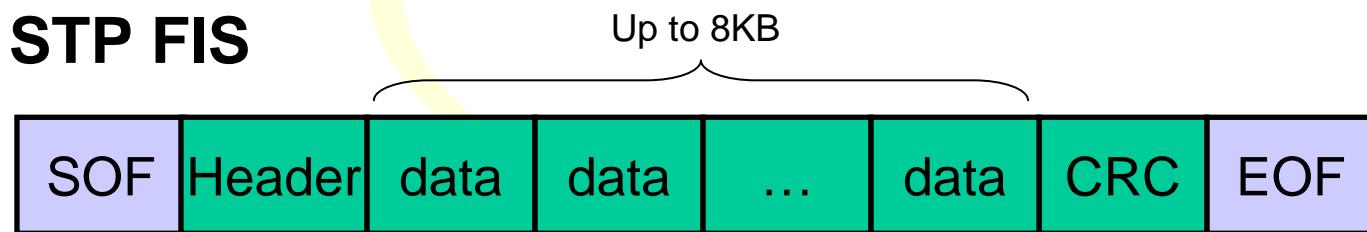
Offset	Reads	Writes	Notes
Cmd+0		Data	Data register – only shows 8 bits, but 16-bit access possible, starting at CMD+0
Cmd+1	Error	Feature	
Cmd+2		Sector Count	Logical Block Address (replaces older-style sector, cylinder, head info)
Cmd+3		LBA Low	Formerly Sector Number
Cmd+4		LBA Mid	Formerly Cylinder Low
Cmd+5		LBA High	Formerly Cylinder High
Cmd+6		Device	Device select bit=0 in SATA; no master/slave function
Cmd+7	Status	Command	Host sends command code, which starts execution of the command
Ctrl+2	Alternate Status	Device Control	Status and Alt. Status both contain same info, but reading Status will clear a pending interrupt

# AHCI Software Interface



# Transport Layer (463)

- In legacy operation, transport layer receives a request from the application layer and creates a FIS (*Frame Information Sequence*).
- For AHCI, software creates the frames directly in memory, so this task doesn't need to be done in hardware.



# SATA Frame Format (463)

- FIS Type field indicates the type of frame
- PM Port field – specifies which device behind a port multiplier is targeted for transfer
- CRC protects the entire frame
- These fields are scrambled and sent between SOF and EOF

Byte	Field(s)	
0	FIS Type	
1	FIS-specific	PM Port
1 to (n-4)	FIS-specific (see next slide)	
(n-3) to n	CRC	

# Sample Frame Header

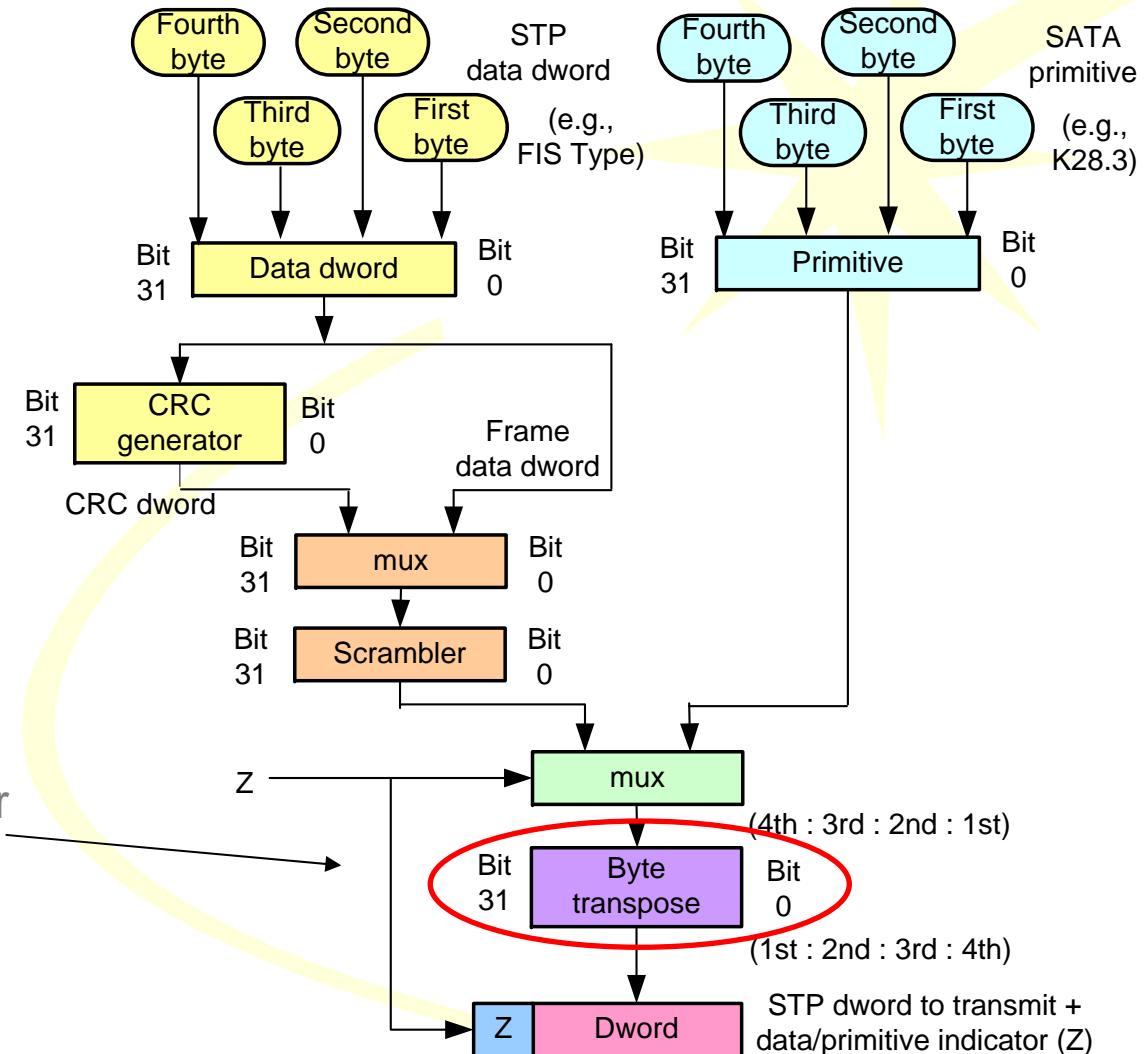
Byte	Field(s)				
0	FIS Type (27h)				
1	C	R	R	R	PM Port
2	Command				
3	Features				
4	LBA Low (Sector Number)				
5	LBA Mid (Cylinder Low)				
6	LBA High (Cylinder High)				
7	Device				
8	LBA Low (Sector Number) exp				
9	LBA Mid (Cylinder Low) exp				
10	LBA High (Cylinder High) exp				
11	Features exp				
12	Sector Count				
13	Sector Count (exp)				
14	Reserved				
15	Control				
16 to 19	Reserved				
20 to 23	CRC				

Value	FIS Name
27h	Reg – Host to Dev
34h	Reg – Dev to Host
39h	DMA Activate
41h	DMA Setup
46h	Data
58h	BIST Activate
5Fh	PIO Setup
A1h	Set Device Bits

# SAS Frame vs. SATA FIS

- Register host-to-device - similar to command frame.
- Register device-to-host - like response frame, sent back to host to give results of a command.
- Data frame = data frame
- DMA setup and PIO setup - similar to XFER\_RDY.
  - Indicate readiness to send or receive data. ATA sends one to request write data and another before sending read data, telling the host which DMA context to load and prepare. DMA setup FIS, used for native command queuing, is like XFER\_RDY and contains tag and a data offset and transfer length.
- For reads, DMA setup is sent with tag, data offset, and transfer length because the 1-dword data frame header doesn't have room for that information.

# SAS STP transmit bit order



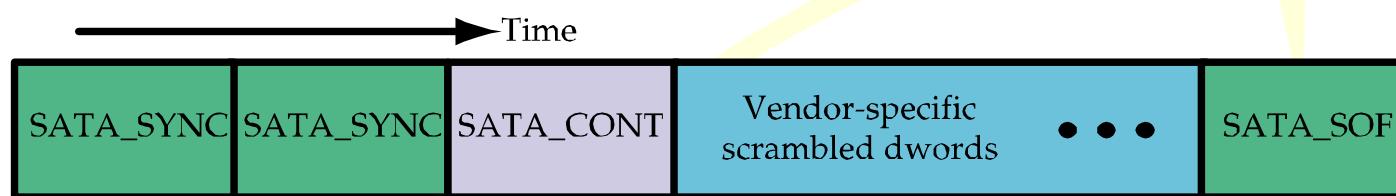
- SATA uses little-endian bytes, so SAS transmitter must reorder them for STP connections because SAS internally works with big-endian bytes

# Link Layer (465)

- Generation of primitives
  - Note: SAS adds prefixes to the SATA primitive names to distinguish them from SAS primitives
    - For example, SATA\_SOF used in STP is different from SOF used in SSP
- CRC generation and checking
- Scrambling
  - Two scramblers needed to cover the case of continued primitives (see next slide)

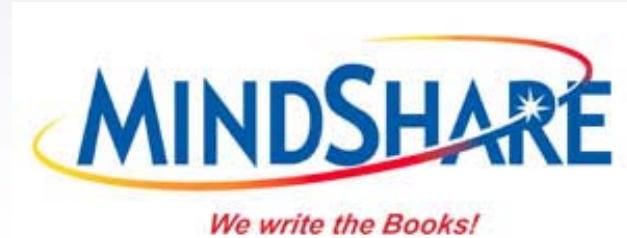
# “Continued” Primitives (470)

- Scrambled dwords could be discarded by expanders, causing scrambler to get out of step with receiver's descrambler.



- A second scrambler is used for the “trash data” that is discarded at the receiver to ensure that the real data scramble/descramble doesn't get out of sync.

# *Link Layer – Connection Management*

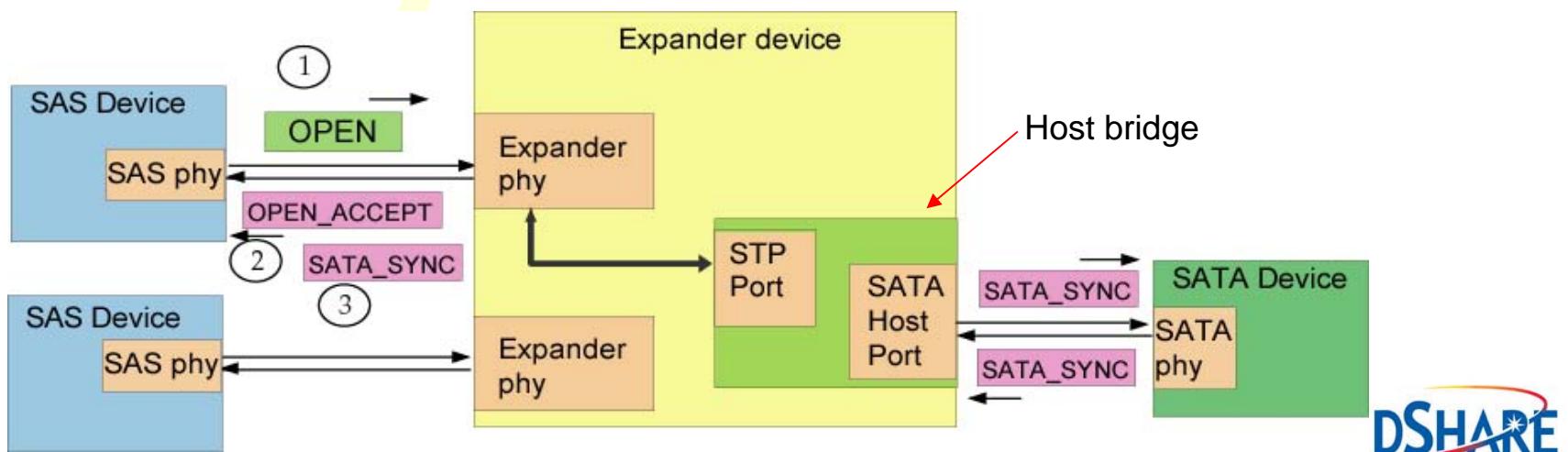


# Link Layer Connection Mgt.

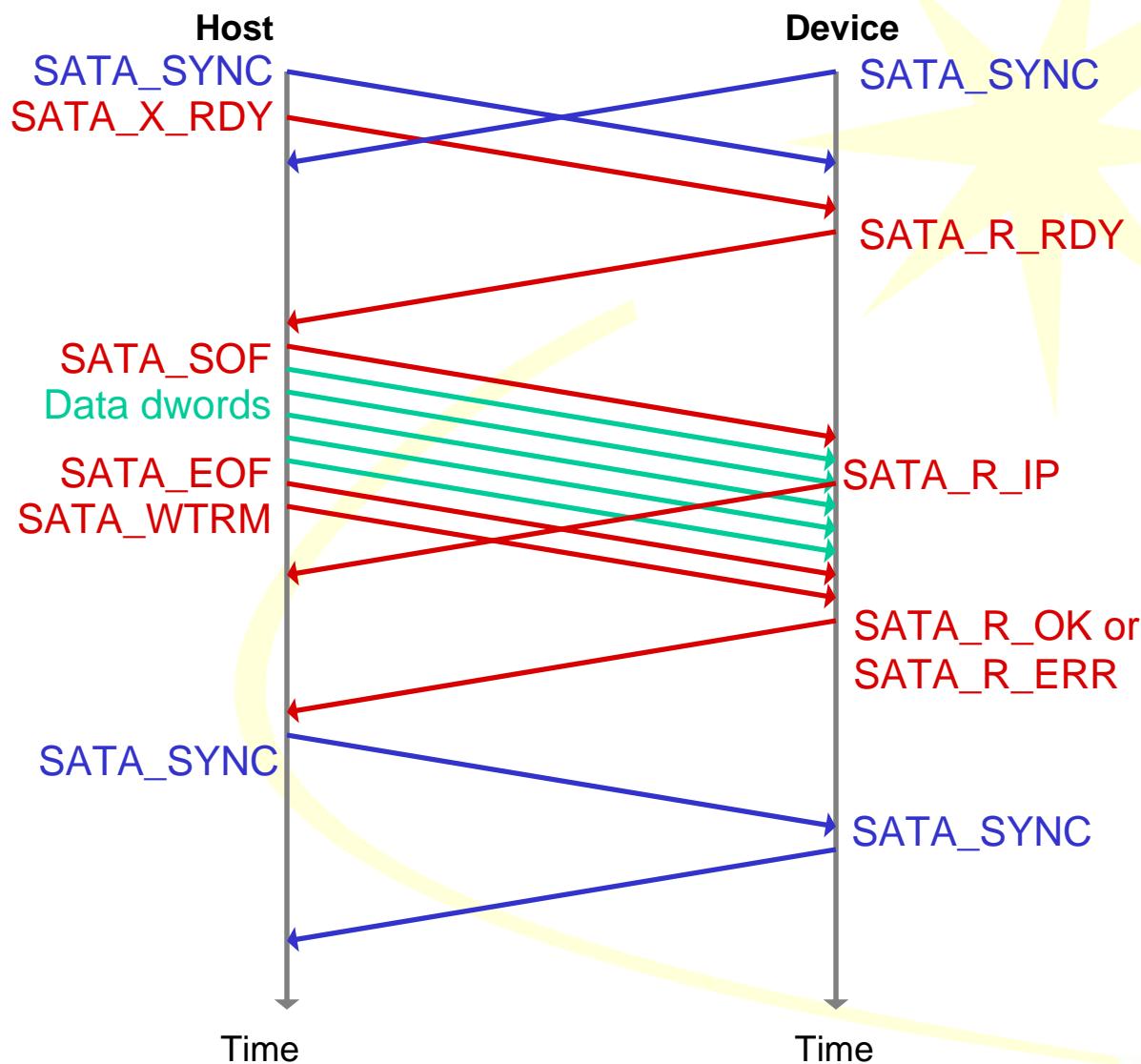
- In a pure SATA system, host and device always communicate directly – no connections
- For SAS infrastructure, initiator must first make connection with SAS/SATA bridge. Bridge and target communicate as if they were SATA host and device
- SATA is Half duplex only and never sends frames in both directions at same time

# STP Open Example (473)

- Initiator attempts to open connection, STP to SATA host bridge accepts connection, responds to initiator
- Once connection is established, initiator begins to see whatever is on the SATA link - probably SATA\_SYNC, or maybe SATA\_X\_RDY if target was trying to send data for an earlier request.

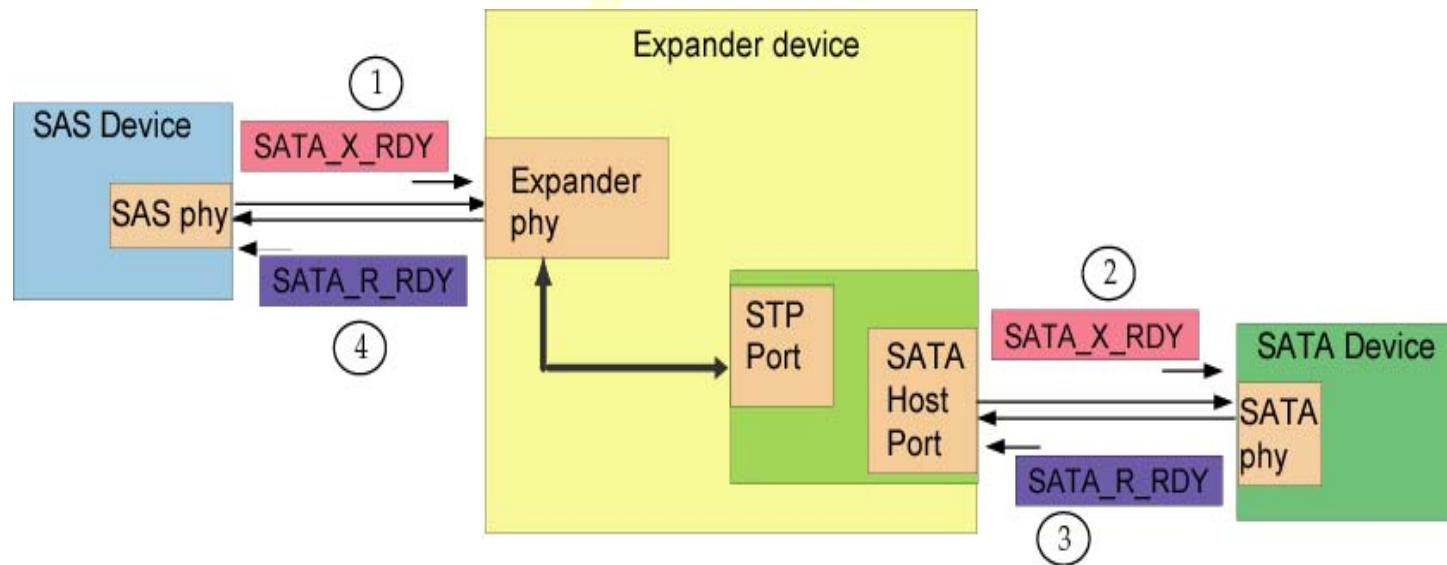


# SATA basic frame transmission



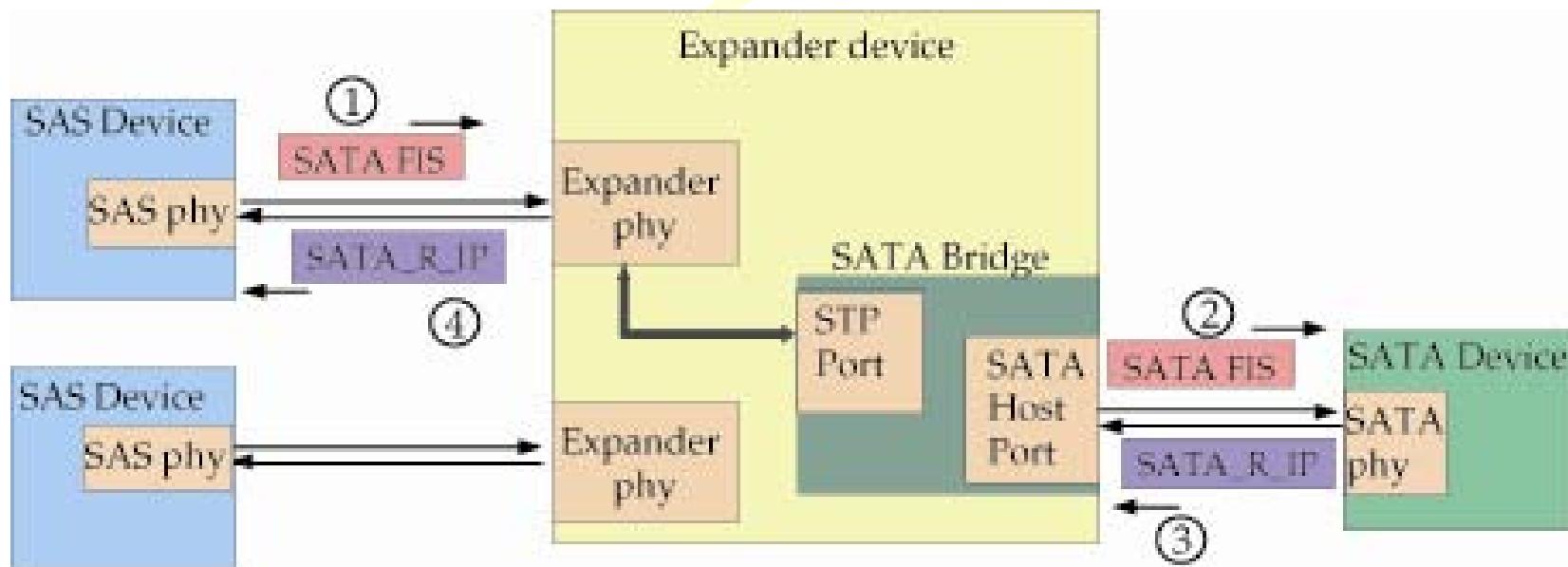
# Data in STP Connection (474)

- Initiator sends SATA\_X\_RDY repeatedly – transmit ready
- SATA device responds with SATA\_R\_RDY repeatedly to indicate readiness for 1 frame (data or register).
  - This is only treated as a single event (not queued up or counted like SAS RRDY)



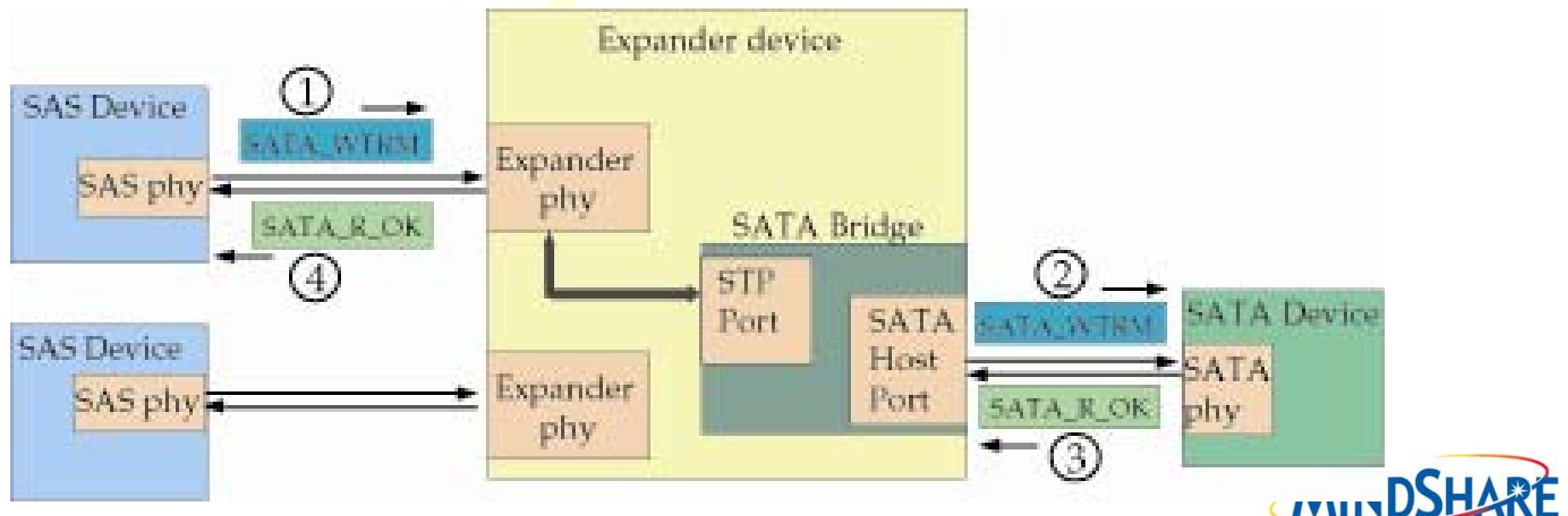
# Data in STP Connection (474)

- Now initiator can send one FIS – Frame Information Structure – which can contain up to 8KB. To send more, the sequence would be repeated as needed.
- While receiving the frame, target responds continuously with Receive in Progress (SATA\_R\_IP)



# Finishing STP Connection (475)

- When frame completes with SATA\_EOF, initiator sends SATA\_WTRM (wait for termination)
- Target responds to SATA\_EOF with SATA\_R\_OK (similar to ACK) or SATA\_R\_ERR (similar to NAK) and then SATA\_SYNC.

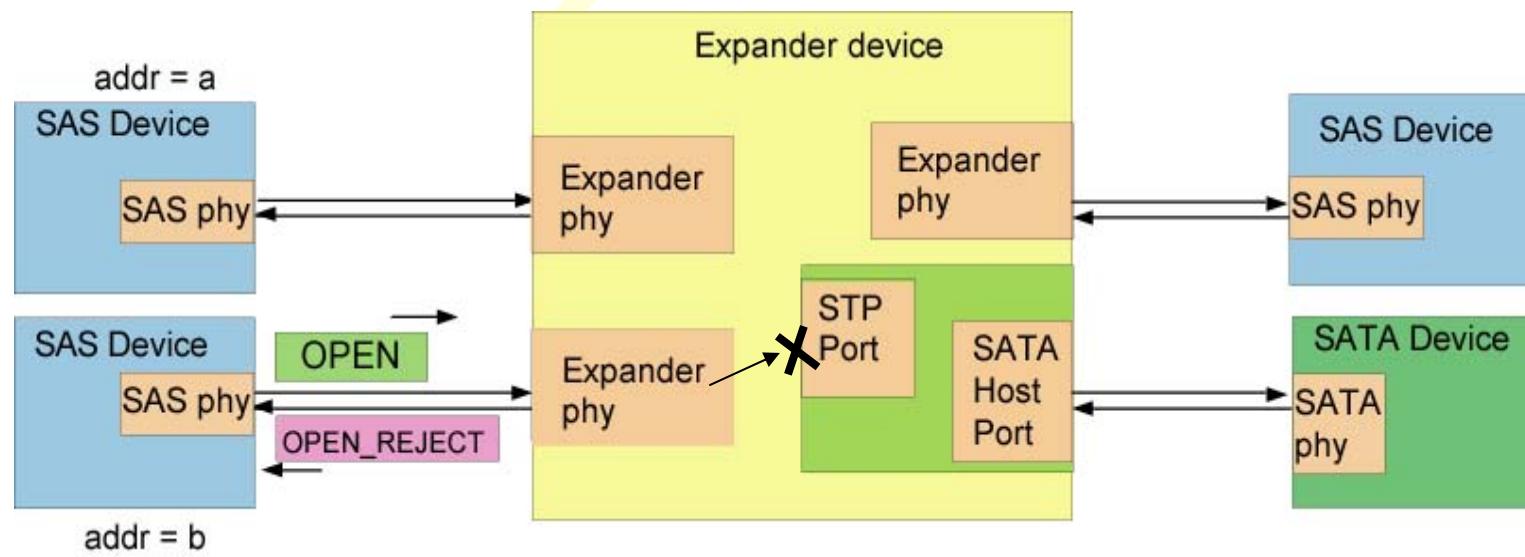


# Affiliation (476)

- Affiliation provides a simple way for bridge to remember address of initiator that made connection.
  - Allows it to provide SAS address for connection request when target wants to return data
  - Rejects requests from other initiators until affiliation is cleared.
  - Bridge must either support affiliation or keep a memory of the equivalent ITLQ nexus information. Standard does not describe how this would be implemented.
- Affiliation remains until cleared by:
  - Power cycle, SMP request, or CLOSE (Clear Affiliation)

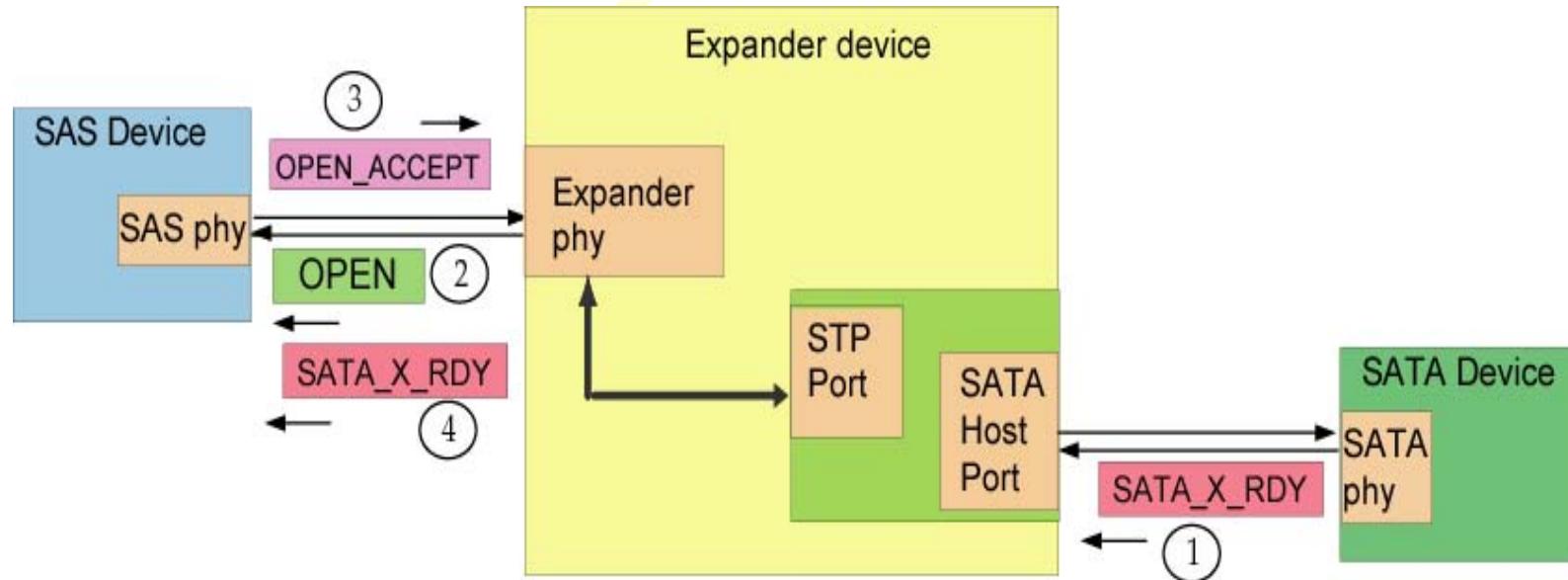
# Affiliation (477)

- Until affiliation is cleared, OPEN requests from other initiators will be rejected with the reason “STP Resources Busy”.
- Requests from other initiators will be allowed after affiliation has been cleared.



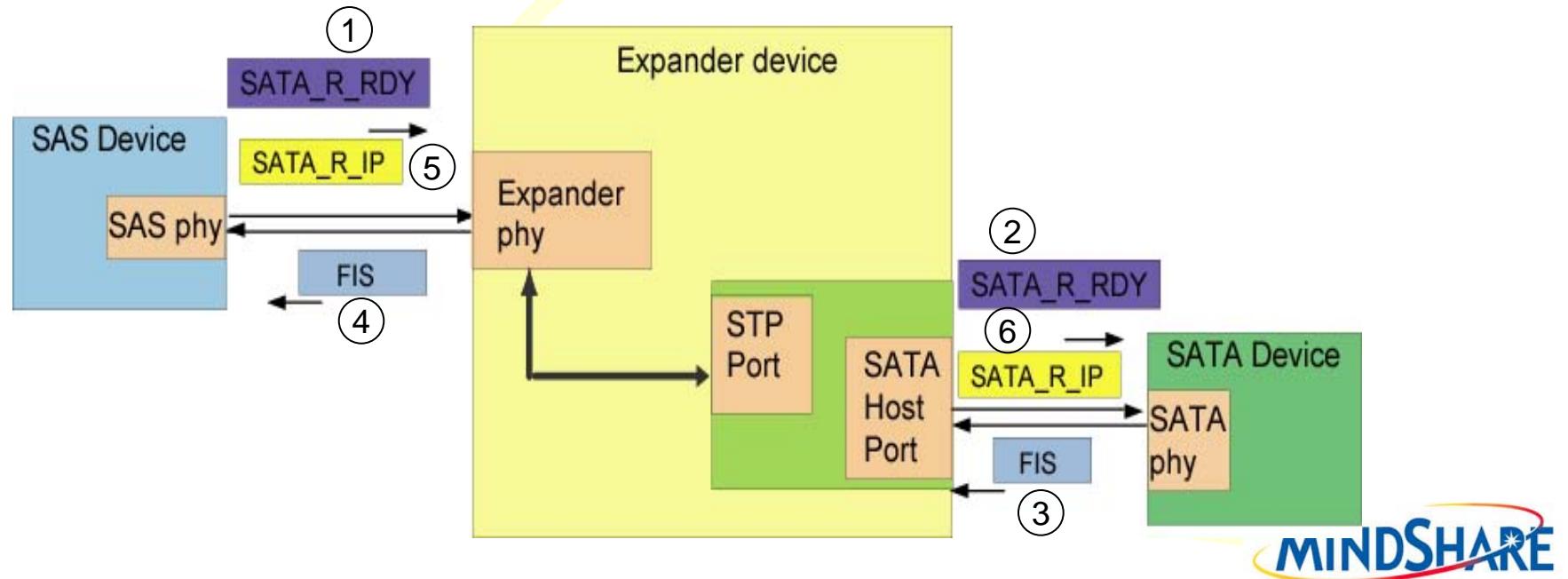
# SATA Target Initiates (478)

- Process is a little different from SATA device
  - SATA device indicates desire to transmit with `SATA_X_RDY`. No address information provided since SATA only understands one host in the system.
  - Bridge/Expander opens connection using the address of the affiliated SAS port.
  - When connection opened, `SATA_X_RDYs` flow through



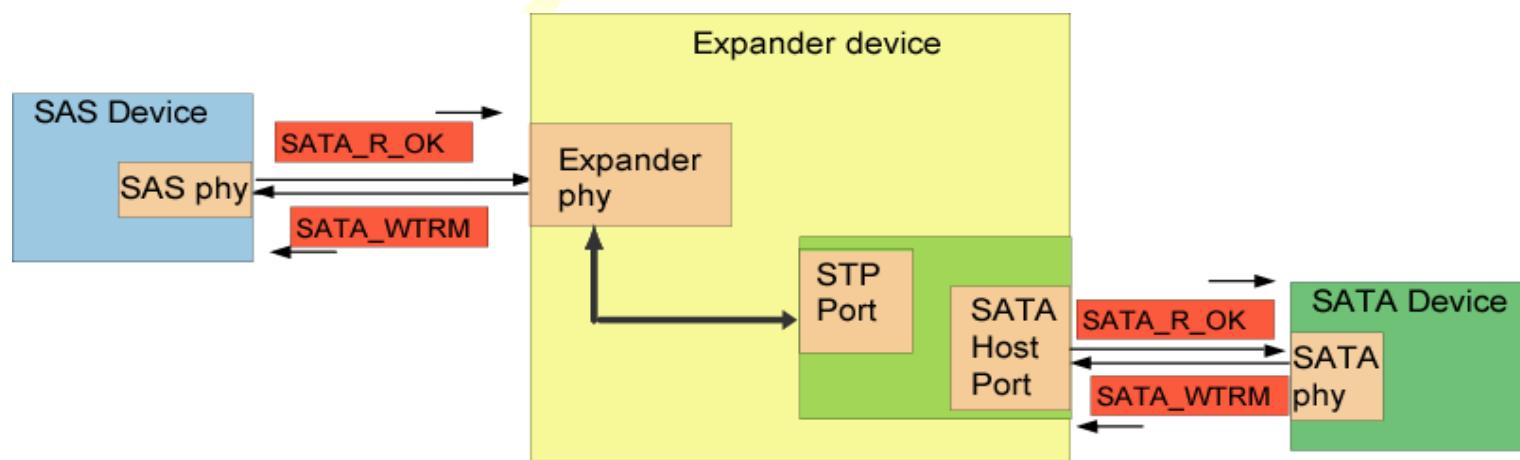
# STP Connection (479)

- SAS device responds to SATA\_X\_RDY with repeated SATA\_R\_RDY, indicating readiness for 1 frame
- Target sends FIS to provide data
- Initiator responds with SATA\_R\_IP



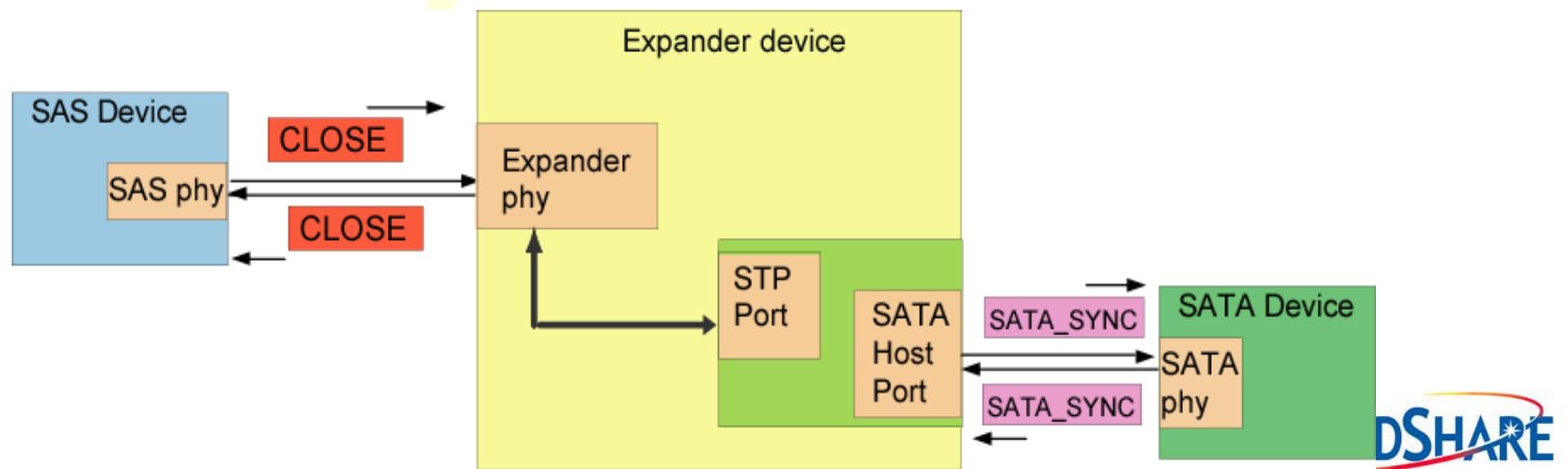
# Target Finishes

- Target sends SATA\_WTRM
- Initiator responds with SATA\_R\_OK
- Both finish by sending SATA\_SYNC



# Bridge Closes Connection

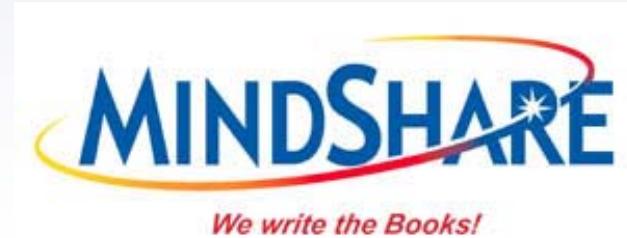
- Bridge recognizes transaction is done, sends CLOSE primitive sequence
  - To clear affiliations, send CLOSE (CLEAR AFFILIATION) or use SMP functions
  - Host shall not clear affiliations while any command is outstanding. Note that connections may be opened or closed many times during a command.
- Initiator responds with CLOSE sequence and connection is broken down



# Note Regarding Affiliations

- If more than one initiator will access the same SATA device they'll need to coordinate
  - First initiator to gain access will establish affiliation, others will be unable to open a connection with that target until the affiliation is cleared.
  - Software must take appropriate precautions to allow sharing, such as clearing affiliations in a timely manner.

# *SATA Flow Control*



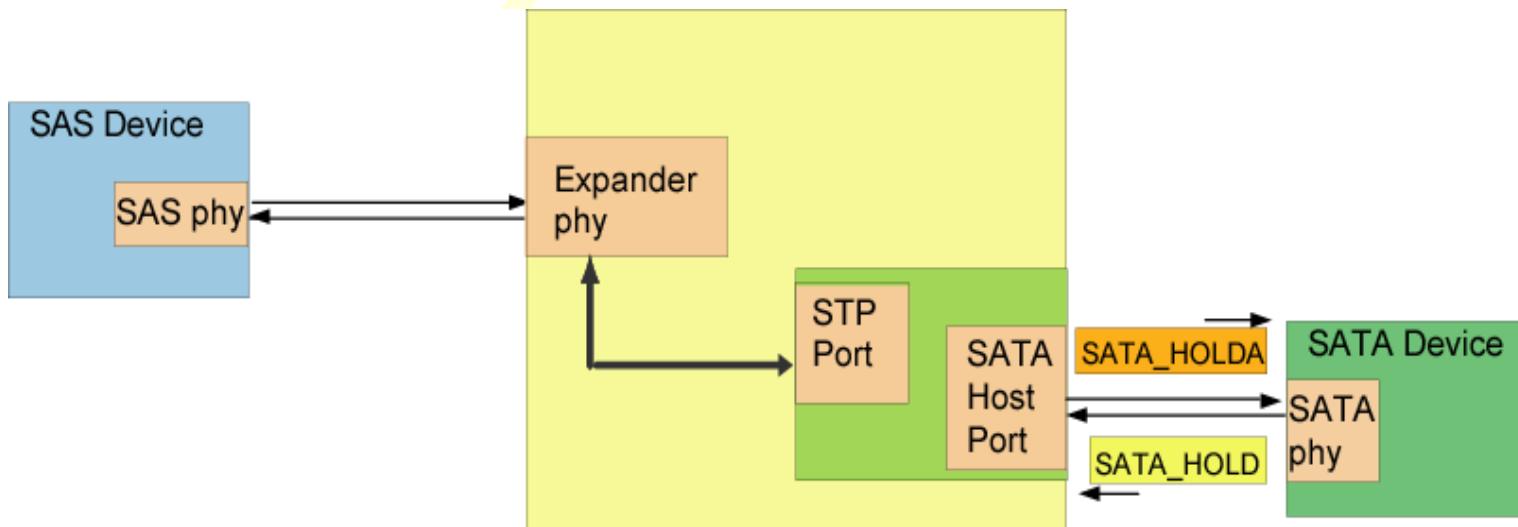
Christy Choi([christy.choi@sandisk.com](mailto:christy.choi@sandisk.com))  
Do Not Distribute

# SATA Flow Control (480)

- Credits are not granted in advance. Instead, flow control is designed to allow designers to use receive buffers that are much smaller than the max frame size
  - Facilitates a low-cost, almost “buffer-less” design
  - If the small buffer approaches an underflow or overflow condition, pause transmission

# STP Rx Flow Control

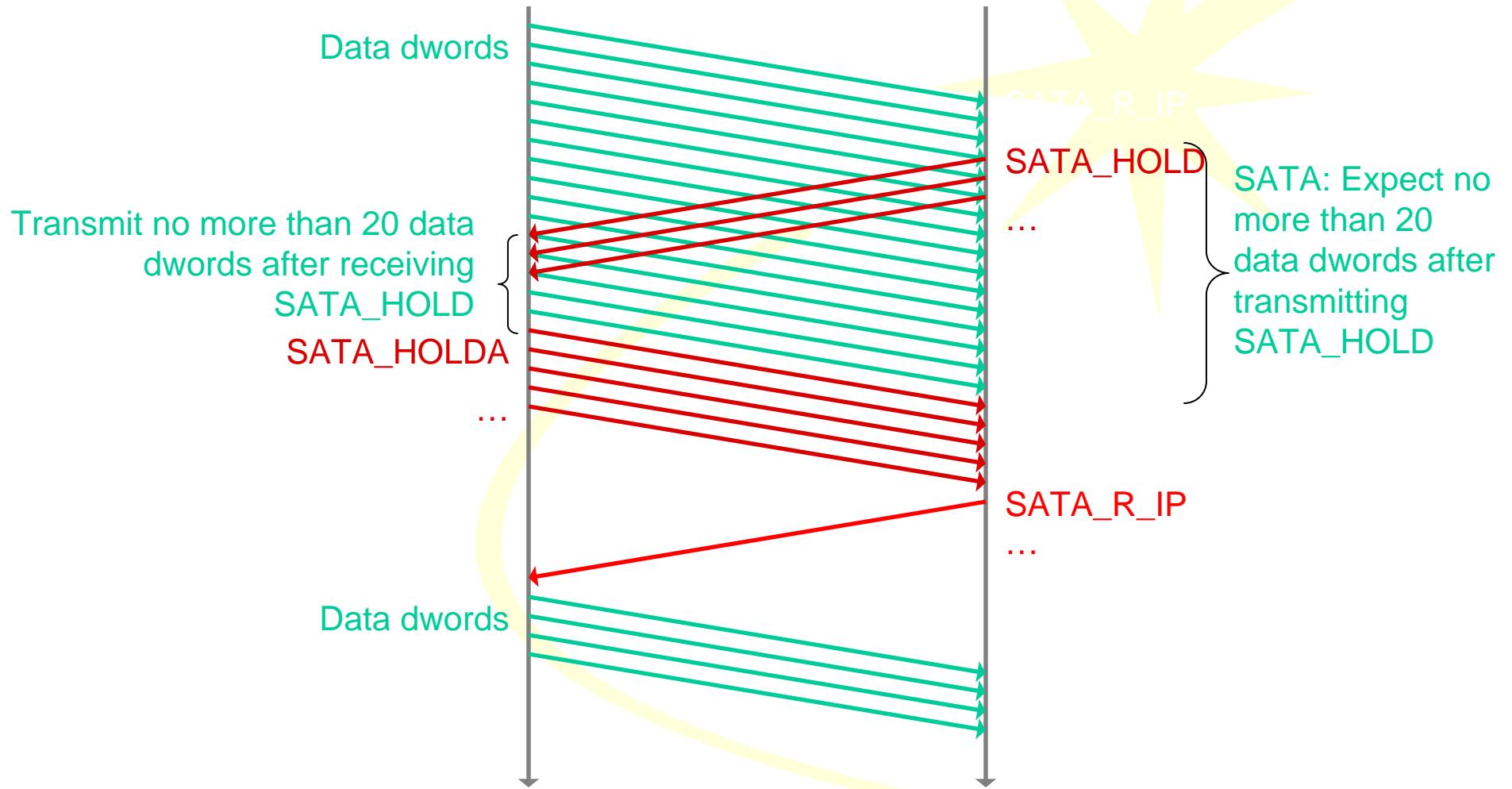
- If SATA receiver approaches buffer full condition it stops sending SATA\_R\_IP (Receipt In Progress) and begins to send SATA\_HOLDs instead.
- When SATA transmitter sees SATA\_HOLD it must stop sending data and respond with SATA\_HOLDA within 20 dwords



# Buffer Size

- The prop delay from SATA transmitter to receiver is small but nonzero, so SAS requires bridges to:
  - Send no more than 19 dwords when `SATA_HOLD` received
  - Implement buffer size to accept 21 dwords when sending `SATA_HOLD`
- The path through expanders might take even longer because a long (10m) cable might be in the path, so expander buffers must be bigger still:
  - 24 dwords at 1.5 Gbps
  - 28 dwords at 3.0 Gbps

# Rx Flow Control Example

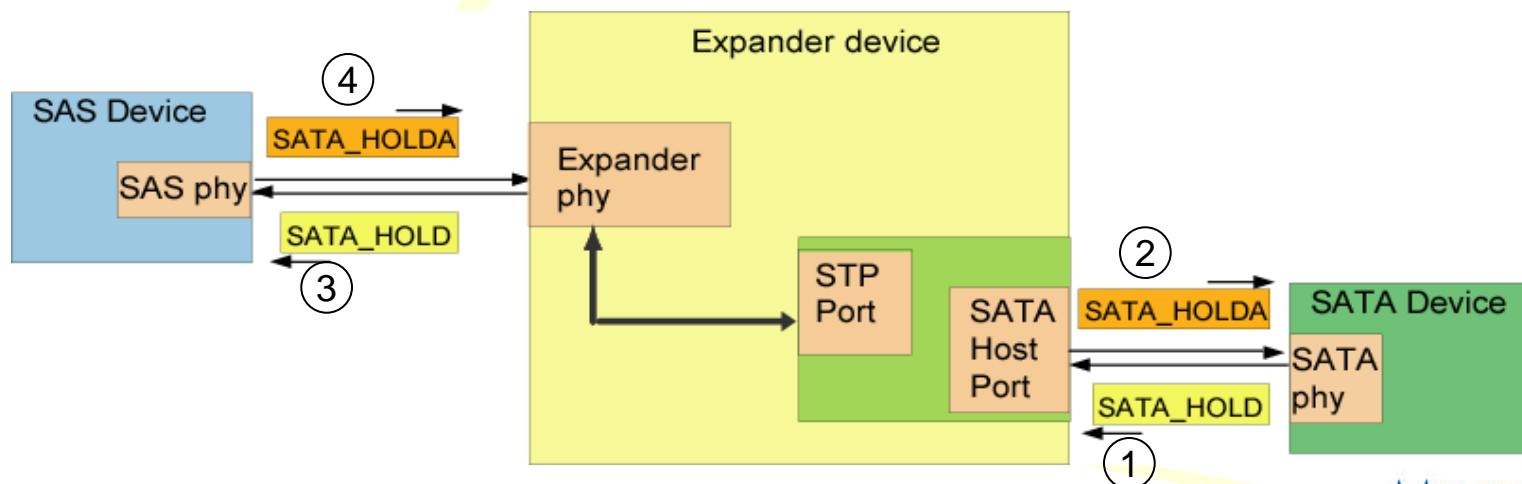


# SATA flow

- Flow control only takes place during data transfer (longer frames)
  - Not allowed for other types like Register FIS, but those are so short that it wouldn't work anyway
- Half-duplex
  - everything waits for response, no credit established ahead of time

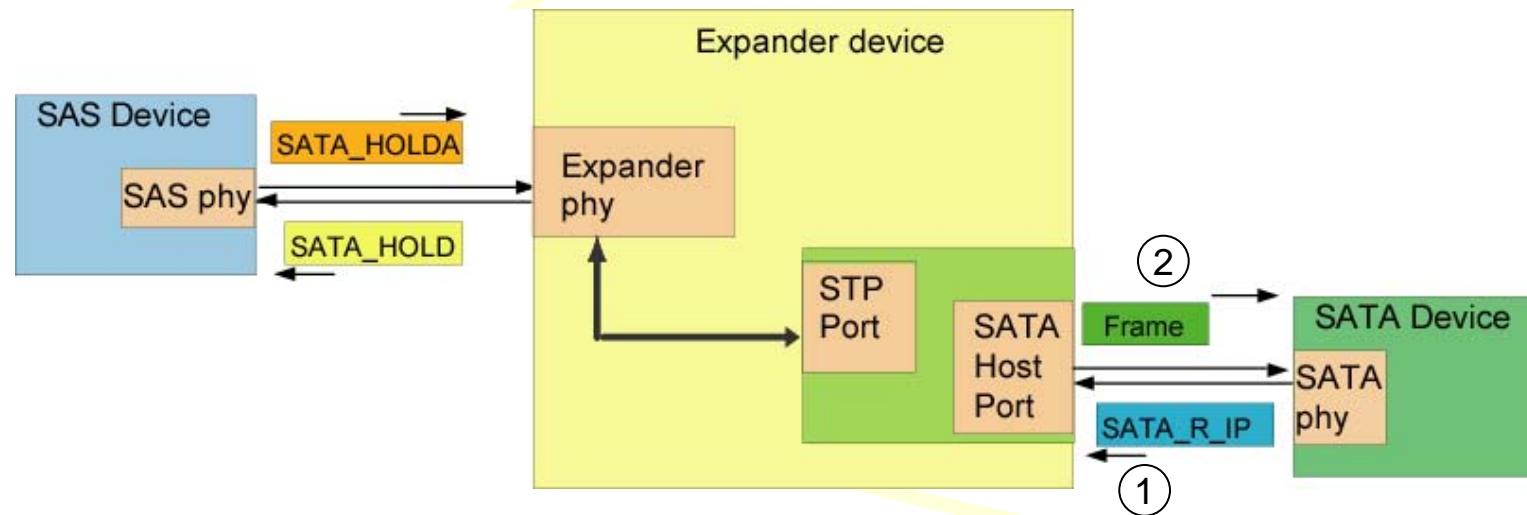
# STP Hold Propagation

- SATA\_HOLD must propagate back to sending device to stop the flow of data before intermediate buffers overflow
- As before, when SATA\_HOLD received, transmitter must stop sending packets and start sending SATA\_HOLDA within short time – no more than 20 dwords



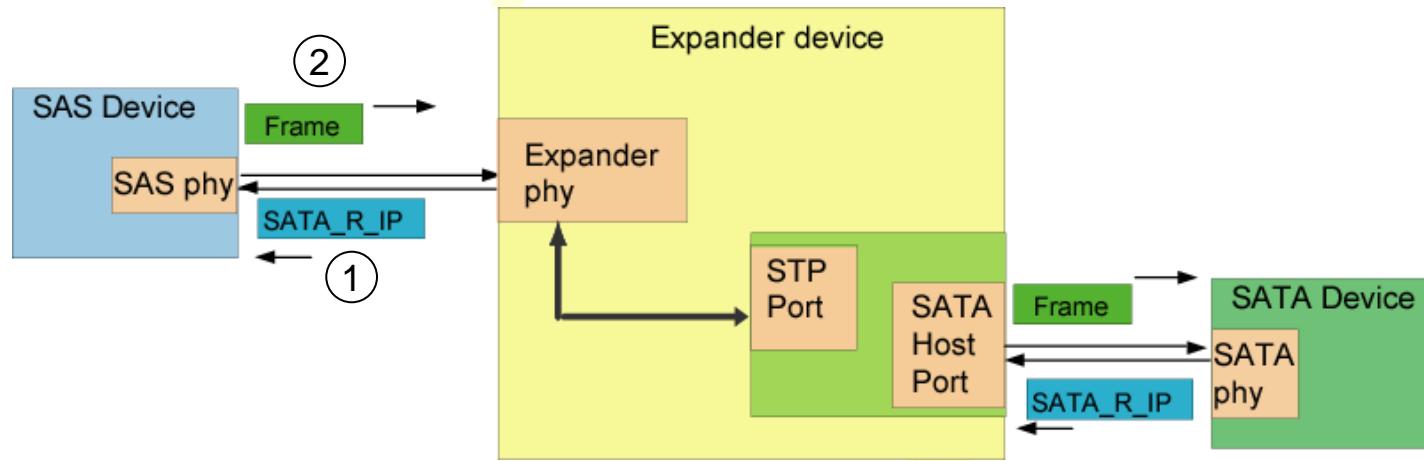
# STP Rx Restart

- When receiver is ready again, it stops sending HOLD and resumes sending SATA\_R\_IP
- Expander responds by resuming data flow, but doesn't stop sending SATA\_HOLD to SAS initiator yet



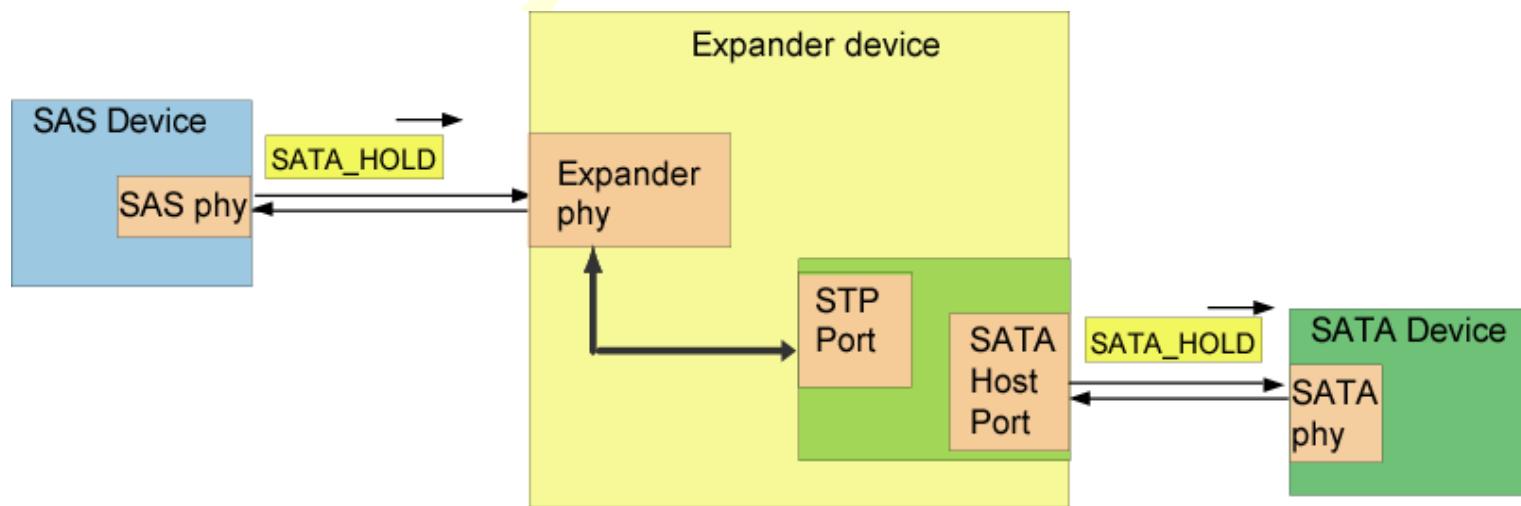
# STP Rx Restart Propagation

- When the expander buffer has been drained, SATA\_R\_IP is propagated back to sending device
- In response, initiator resumes data flow



# STP Tx Flow Control

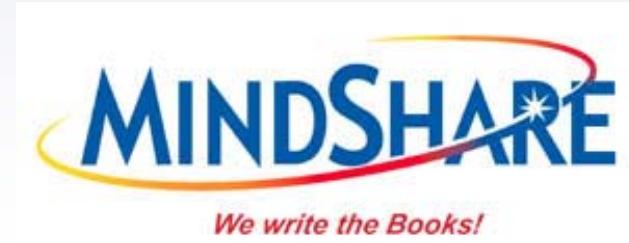
- If Tx approaches buffer-dry but isn't at a place to end the frame, it begins sending SATA\_HOLD.
  - Tx does not wait for HOLDA response to stop transmission and does not look for it



# STP Tx Target Response

- SATA target responds by sending **SATA\_HOLDA**, which propagates back to sending device, but this time there's no concern about buffer space since transmitter already stopped
- When Tx is ready again, it simply resumes sending frames
- When Rx sees frames resumed, it stops sending HOLDA and replies with **SATA\_R\_IP** again

# *SATA II*



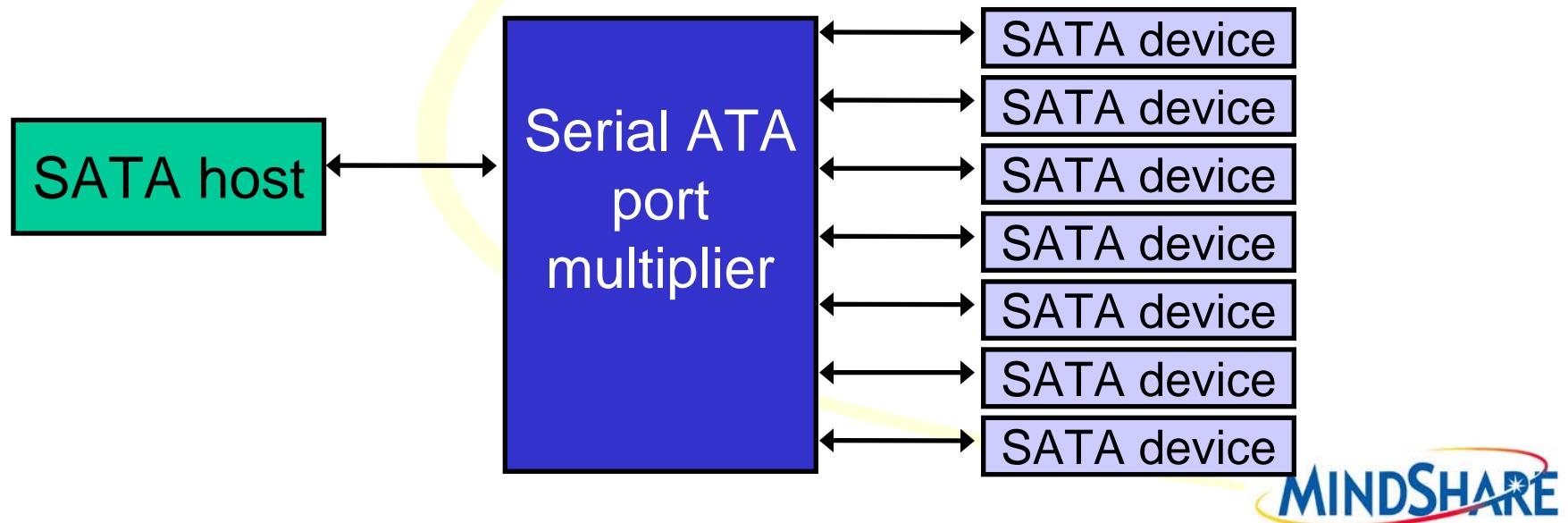
Christy Choi([christy.choi@sandisk.com](mailto:christy.choi@sandisk.com))  
Do Not Distribute

## SATA II (484)

- Includes extensions to migrate SATA toward the higher end:
  - Port multiplier – allows more devices/host
  - Port selector – allows two hosts/device
  - SEMB – enclosure management
  - Native command queuing
  - Higher speed: 3.0 Gbps
  - New connectors and cables

# Port Multiplier (485)

- Port multipliers attach up to 15 devices to a host
  - Store and forward frames from host based on a new PM\_PORT field in the FIS header (see next slide). Port Multiplier strips off this field and zeros it.
  - Inserts the PM\_PORT field for frames going back to host
  - Can't be cascaded
  - Example: Silicon Image Sil3726 1-to-5 PM



# Register – Host to Device FIS

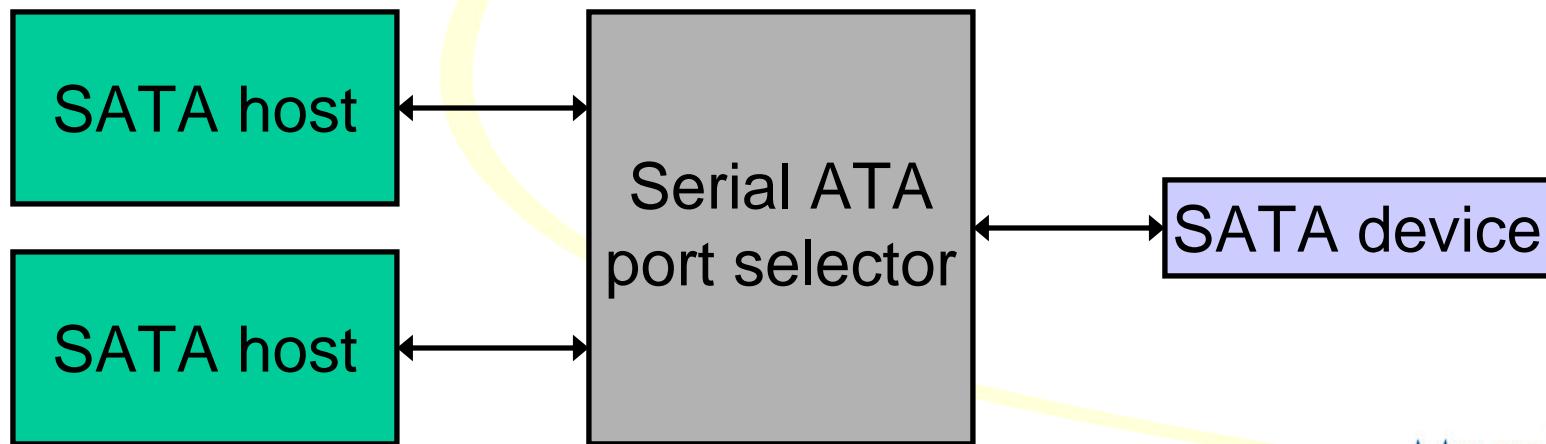
## Includes PM Info

Byte	Field(s)					
0	FIS Type (27h)					
1	C	R	R	R	PM Port	
2	Command					
3	Features					
4	LBA Low (Sector Number)					
5	LBA Mid (Cylinder Low)					
6	LBA High (Cylinder High)					
7	Device					
8	LBA Low (Sector Number) exp					
9	LBA Mid (Cylinder Low) exp					
10	LBA High (Cylinder High) exp					
11	Features exp					
12	Sector Count					
13	Sector Count (exp)					
14	Reserved					
15	Control					
16 to 19	Reserved					
20 to 23	CRC					

For Port Multiplier

# Port Selector (485)

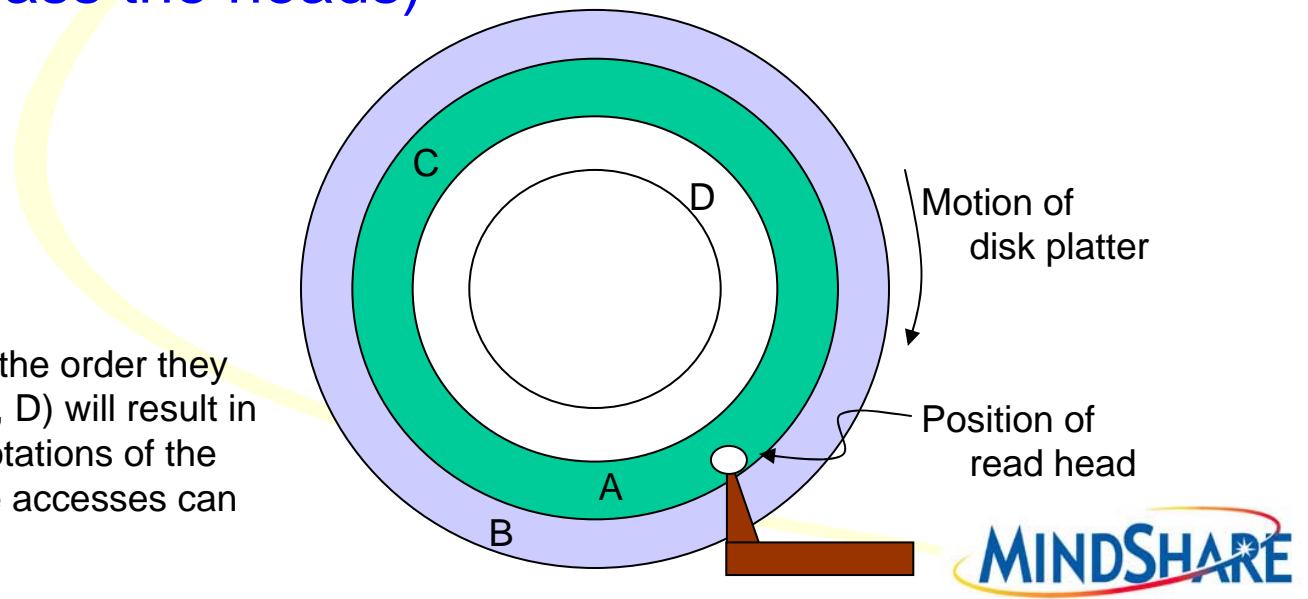
- Allows a device to be accessed by two hosts
  - Provides fail-over capability
  - First host to run OOB is active, latecomer is inactive
  - Inactive host can take control by sending a particular sequence of COMINIT OOB signals, or a side-band signal could be used
  - One port is always inactive and failover takes some time, so this isn't as useful as a true dual-ported design, but it does eliminate a single point of failure.



# Command Queuing (486)

- Command queuing allows re-ordering a group of commands to achieve better performance and improved drive endurance
- Improvements comes from reducing:
  - Search time (moving the read/write head)
  - Rotational latency (waiting for the sector on the media to pass the heads)

Accessing locations in the order they were received (A, B, C, D) will result in waiting for additional rotations of the platter. Reordering the accesses can save time (D, C, B, A).



# Command Queuing

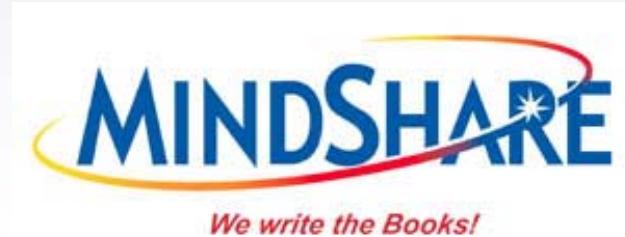
- Improves performance in systems that generate random accesses
  - SCSI implemented it years ago to improve server drive performance, since many types of server drive access are random
  - ATA didn't need it; single-user desktop accesses were more linear and the gain would not have been significant.
  - New multi-core and multi-threaded processors can benefit from this
  - Desire to move SATA into server market also drives queuing for SATA

# Native Command Queuing

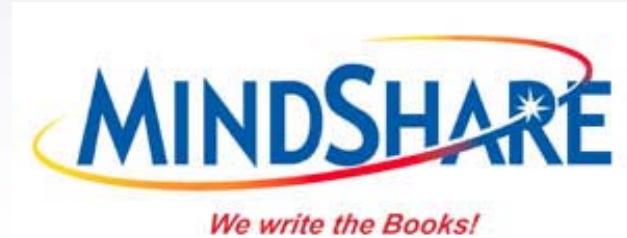
- New SATA drives implement NCQ, equivalent to SAS simple queuing.
- Up to 32 commands can be queued
  - Less than the 256-deep queue that SAS offers, but real-life server queues reportedly are less than 64 anyway, and many are only 32.
- Use of NCQ requires a controller that supports it

# *Chapter 20*

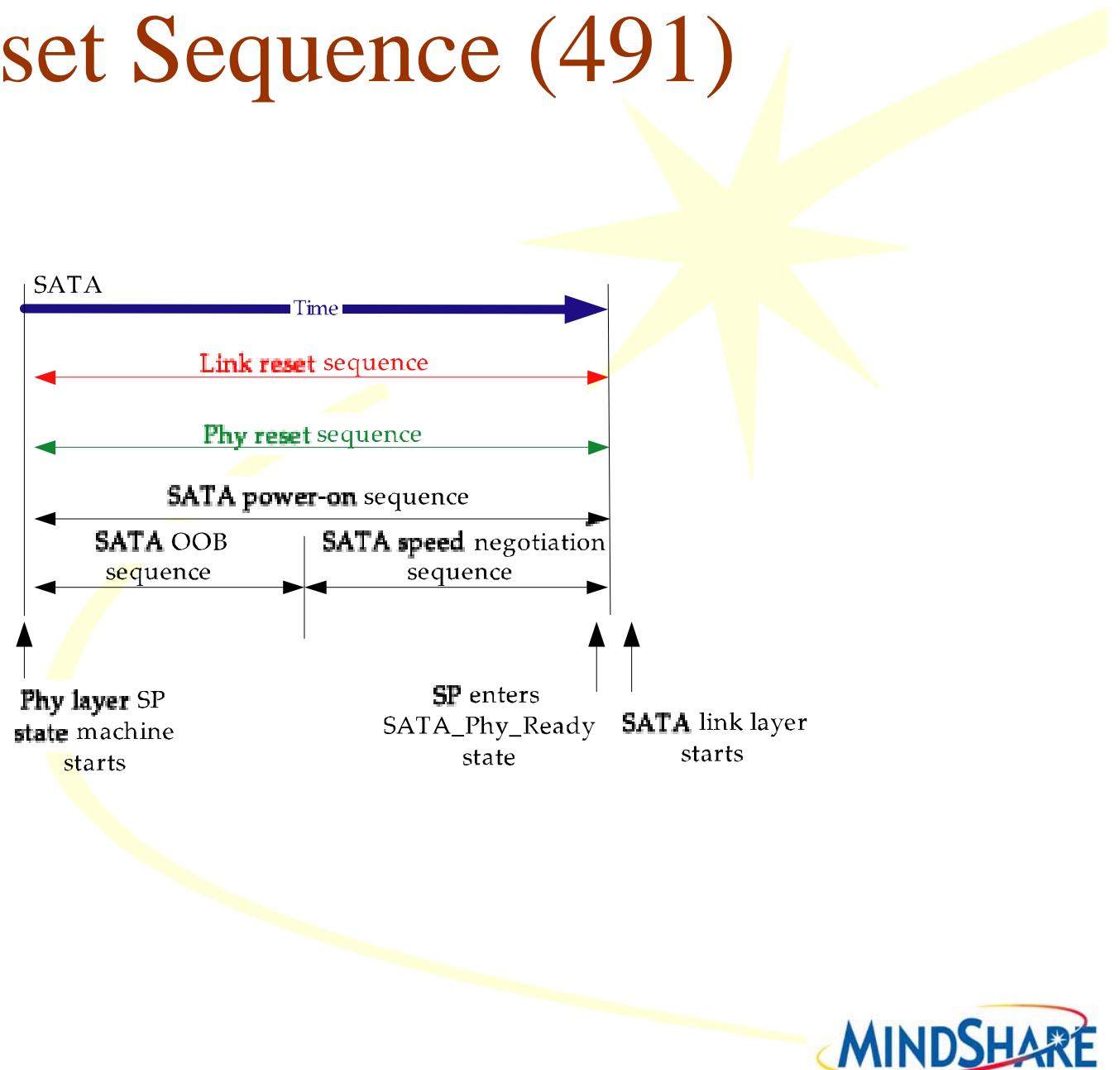
## *SATA Initialization*



# *Hardware Initialization*

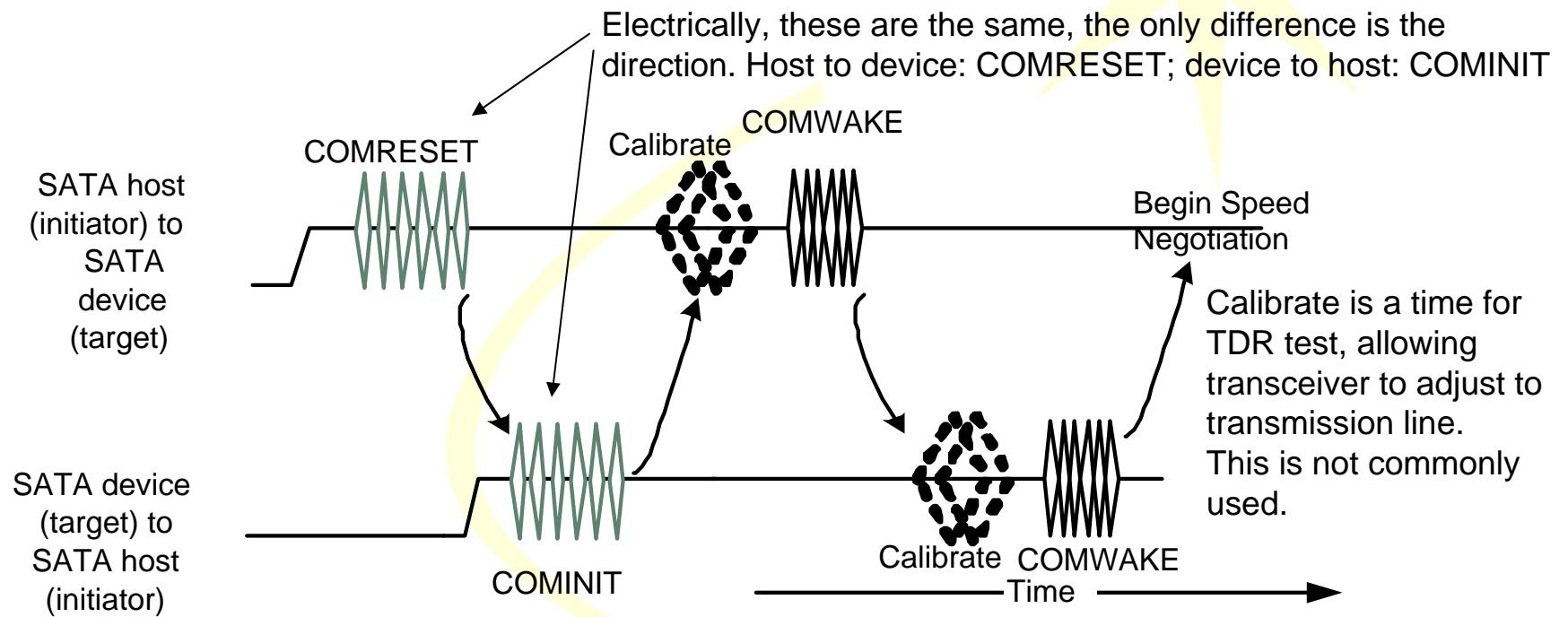


# Reset Sequence (491)



# OOB Sequence

- Send COMRESET and receive COMINIT
- Host then sends COMWAKE



After reset, it's the target that will actively continue trying to get OOB working by sending COMINIT; initiator is passive.

# Phy State Machines - Initialization

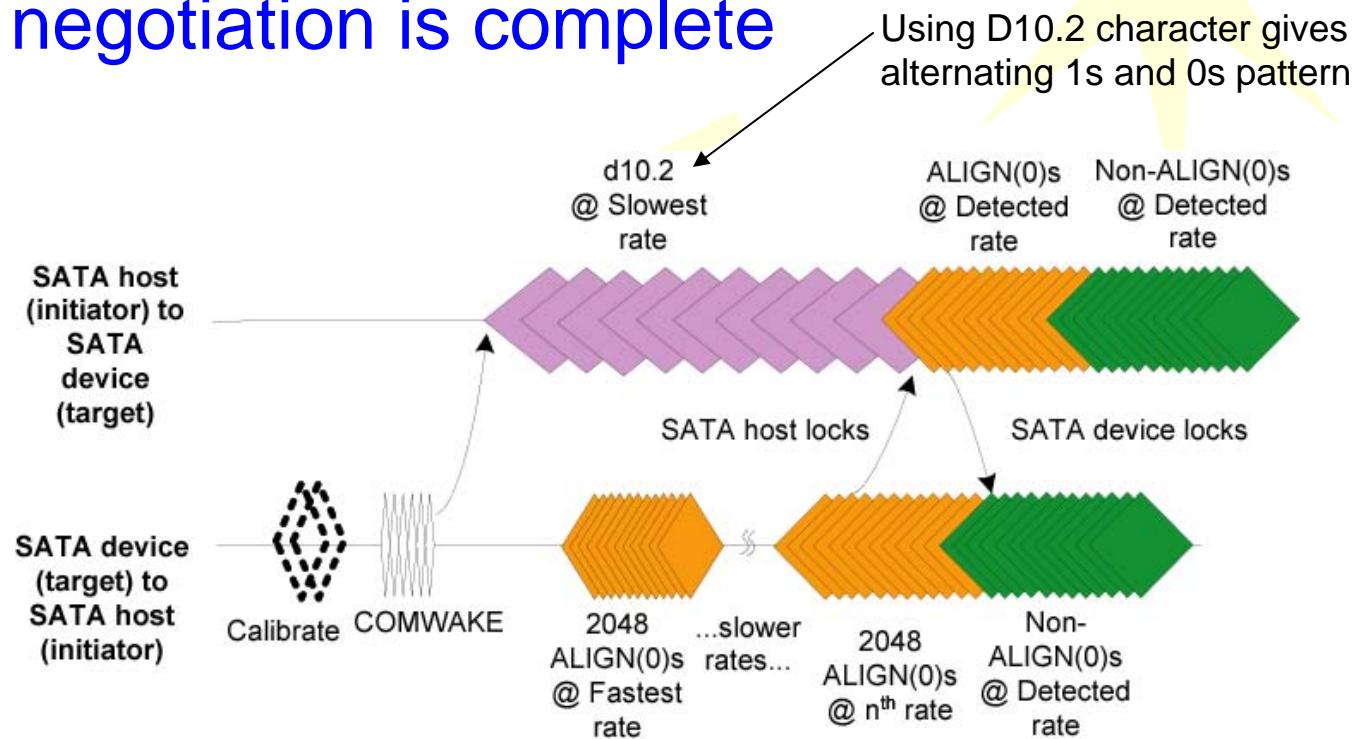
- Separate state machines for hosts and devices
- Each chooses which OOB signals to send (COMRESET, COMWAKE, ALIGN) and at what time

# SATA Speed Negotiation (493)

- Fast to slow progression
  - SATA target device sends ALIGN primitives at the fastest supported rate
  - Waits for host to reply with ALIGNs
  - If no reply after sending 2048, step down to next slower speed and try again

# Speed Negotiation (495)

- When host replies with ALIGN(0)s, it has locked at the current frequency and negotiation is complete

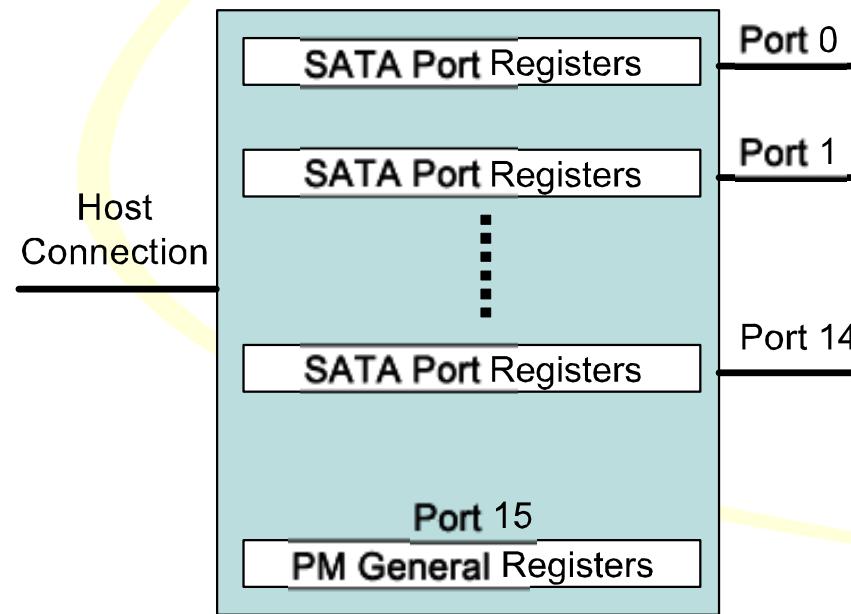


# SATA Software Initialization

- Accomplished by sending a series of commands to the drive
- Example:
- Write SET TRANSFER mode into ATA drive Features register
  - Write SET TRANSFER command into command register
  - Wait for interrupt indicating drive has accepted the SET FEATURES command

# Accessing a Port Multiplier (500)

- If a PM is present, the host will not be aware of it and will be communicating with PM port 0 by default. To learn whether a PM exists, software must set the port value to 15 and send a reset. The internal PM port will return a FIS with the power-on signature of the PM.



Copyright MindShare, Inc. 2006