

Serial Attached SCSI Link layer – part 2



by Rob Elliott

HP Industry Standard Servers

Server Storage Advanced Technology

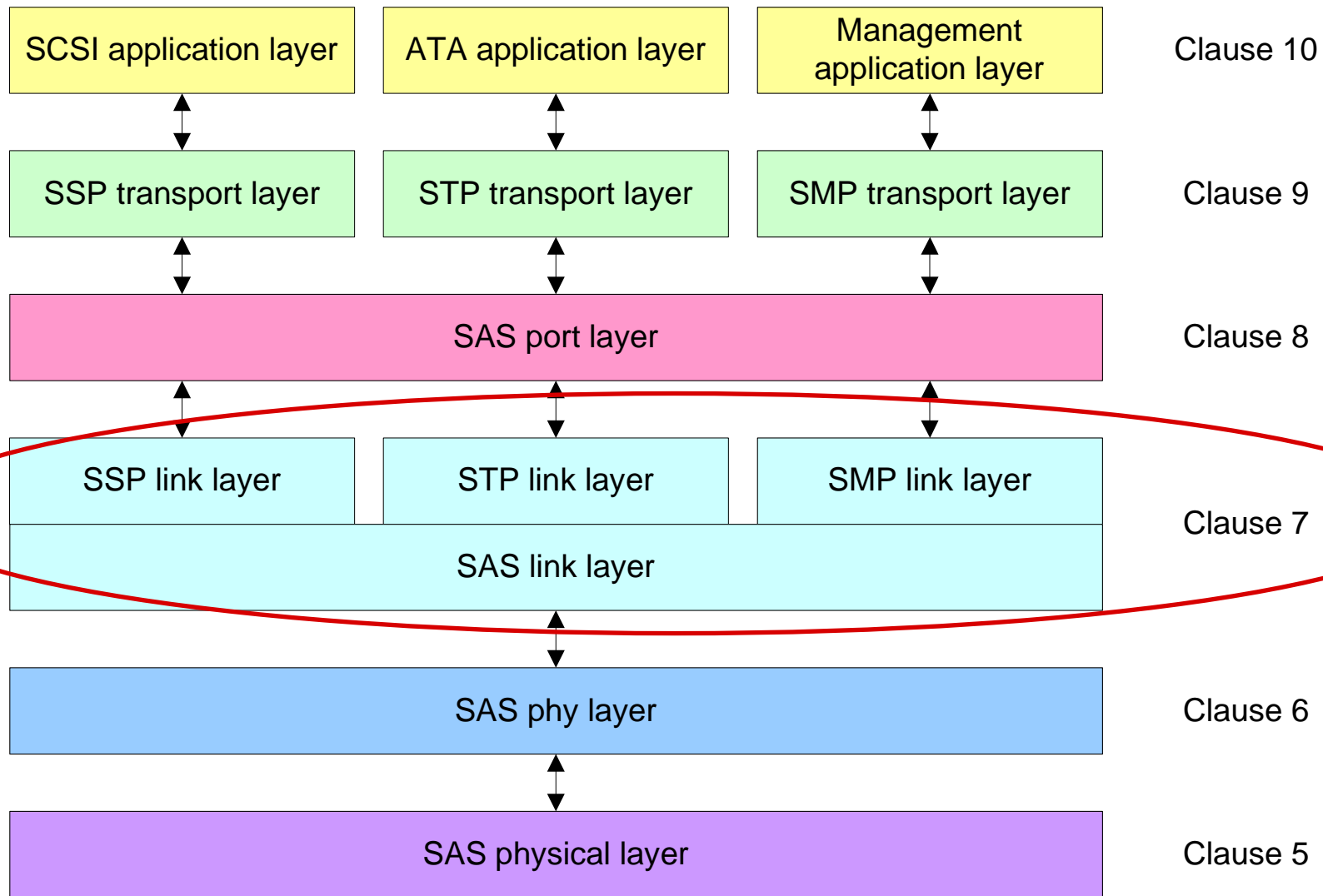
elliott@hp.com <http://www.hp.com>

30 September 2003

- These slides are freely distributed by HP through the SCSI Trade Association (<http://www.scsita.org>)
- STA members are welcome to borrow any number of the slides (in whole or in part) for other presentations, provided credit is given to the SCSI Trade Association and HP
- This compilation is © 2003 Hewlett-Packard Corporation



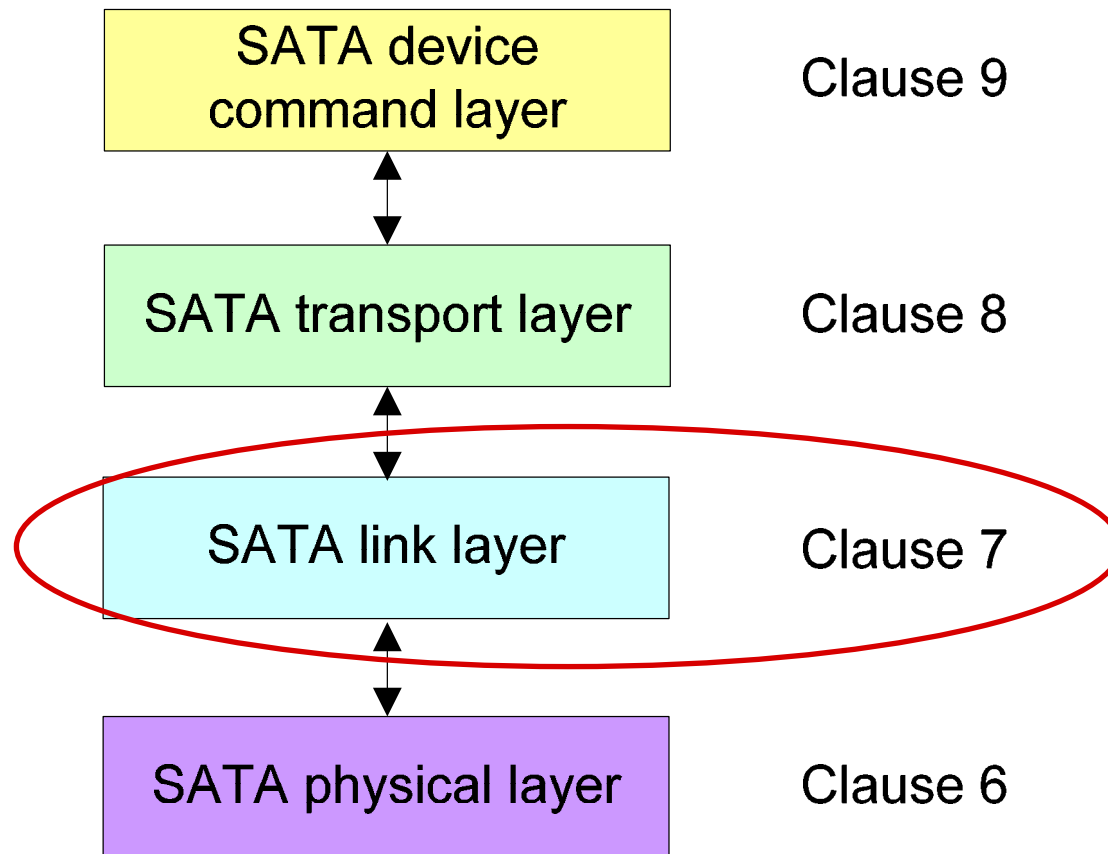
SAS standard layering



SATA 1.0a standard layering



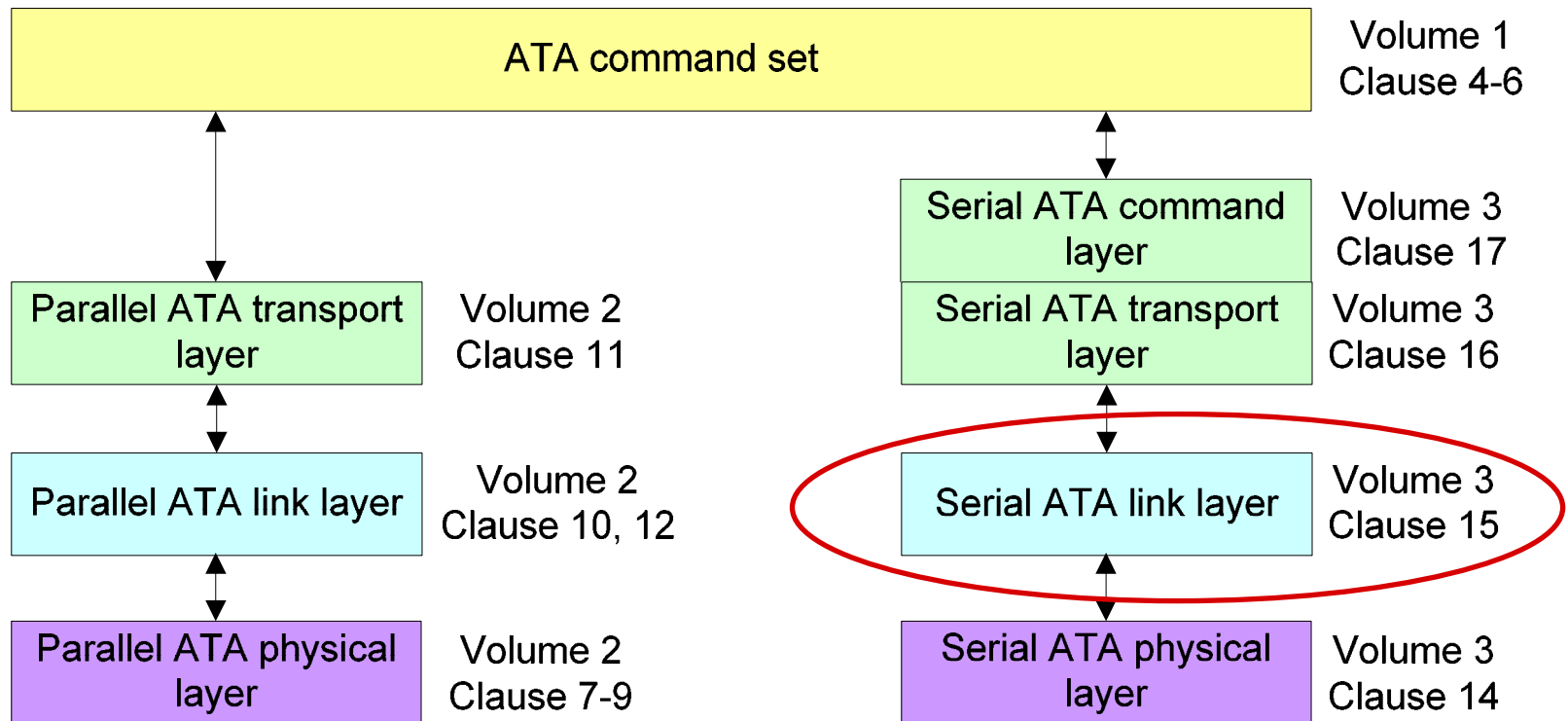
- For SATA 1.0a from the private Serial ATA working group



ATA/ATAPI-7 standard layering



- For the public standard ATA/ATAPI-7
- Subject to change by T13 standards committee



SAS clause 6 – Link layer (part 2)

- Arbitration fairness
- Deadlocks and livelocks
- Rate matching
- SSP (Serial SCSI Protocol)
- SMP (Serial Management Protocol)
- STP (Serial ATA Tunneling Protocol) and Serial ATA
- Wrap up

Link layer – Arbitration fairness

Arbitration schemes



- When multiple processes contend for access to a shared resource, how to choose which one gets access?
- Important attributes: Fairness, bounded latency, good bus utilization, scalability
- Fair schemes guarantee every process eventually gets access and maintains forward progress
 - **Fair schemes**
 - Random
 - Least recently used (LRU)
 - Least frequently used
 - Round-robin
 - First-in, first-out
 - **Unfair schemes**
 - Fixed priority
 - Shortest job next
 - Most recently used
 - Most frequently used

Arbitration fairness



- SAS implements least recently used (LRU) arbitration fairness
- Each source port has roughly equal ability to try to establish a connection (to any destination port)
- 16-bit **Arbitration Wait Time (AWT)** field in OPEN address frame
 - Indicates how long the request has been aging

Value	Meaning	Notes
0000h	0 μ s	Microseconds
0001h	1 μ s	
...	
7FFFh	32,767 μ s	
8000h	0 ms + 32,768 μ s	Milliseconds (up to 32 seconds!)
8001h	1 ms + 32,768 μ s	
...	...	
FFFFh	32,767 ms + 32,768 μ s	

Initial values for AWT

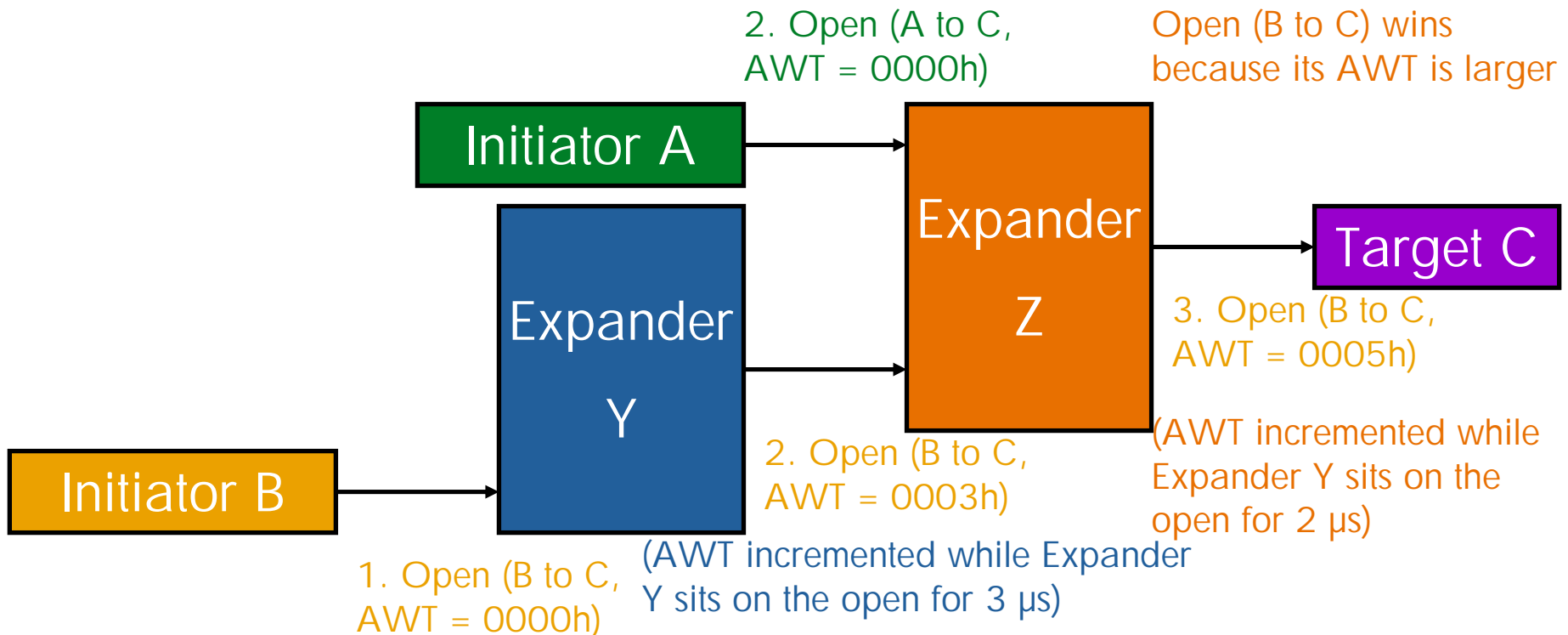


- When an outbound connection request is retried for select reasons, the source SAS port reissues it with a larger AWT
 - Retries allowed to increment AWT
 - higher-priority OPEN address frame – incoming traffic to this port prevented access
 - OPEN_REJECT (PATHWAY BLOCKED) – traffic between other ports prevented access
 - Retries **not** allowed to increment AWT (go back to 0000h)
 - OPEN_REJECT (RETRY) – source got access to the destination
 - OPEN_REJECT (NO DESTINATION) – the expanders declared the destination unknown
 - Open Timeout Timer expires

AWT incremented by expanders



- Expanders increment AWT from time they receive the OPEN address frame until they transmit it
- Ensures requests from sources further away get fair access



Arbitration fairness - cheating



- Requests should start out with AWT of 0000h
- Allowed to cheat and use up to 7FFFh
- Based on parallel SCSI arbitration fairness scheme
 - Targets had to honor arbitration fairness
 - Initiators did not; always had highest priority (e.g. SCSI ID 7)
 - Letting initiators sneak in to drop off commands helped overall performance
 - Command delivery was short compared to data transfer
 - Presumed that initiators didn't get caught in loops resending commands because they received BUSY or TASK SET FULL status
 - Targets controlled both read and write data delivery
 - SAS is different – initiator can arbitrate to deliver write data frames; target just controls XFER_RDY timing

Arbitration fairness inside an expander



- OPENs competing inside an expander
 - Largest AWT wins
 - If more than one AWTs is the largest, compare source SAS addresses
 - If more than one source SAS address is the largest, compare connection rates
 - Possible when a wide SAS port transmits OPENs on different phys simultaneously
 - If everything matches, it doesn't matter
 - Pick any request – they're equal with regard to fairness

MSB		LSB
AWT	Source SAS address	Connection rate

Arbitration fairness for crossing OPENs



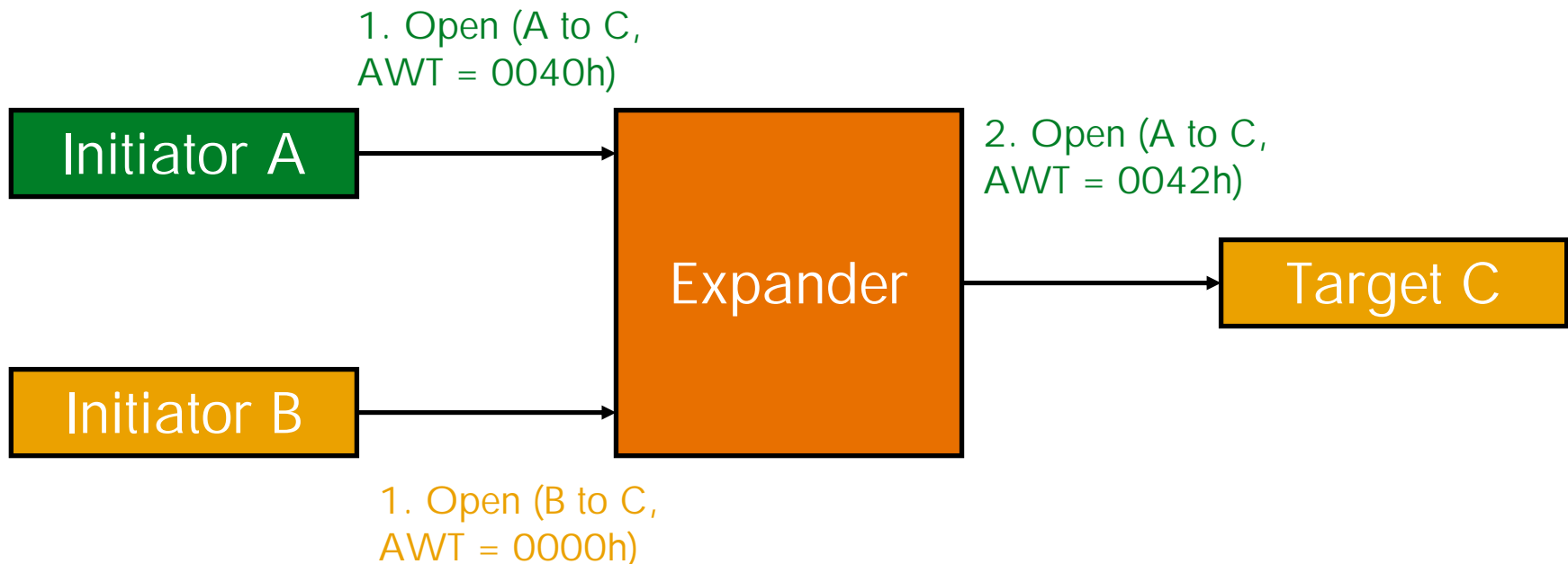
- OPENs crossing on a physical link before AIP
 - Critical that both phys choose the same result
 - Larger AWT wins
 - If AWTs match, compare source SAS addresses
 - Must be different or the devices were manufactured incorrectly

MSB	LSB
AWT	Source SAS address

Arbitration fairness – OPENs inside an expander



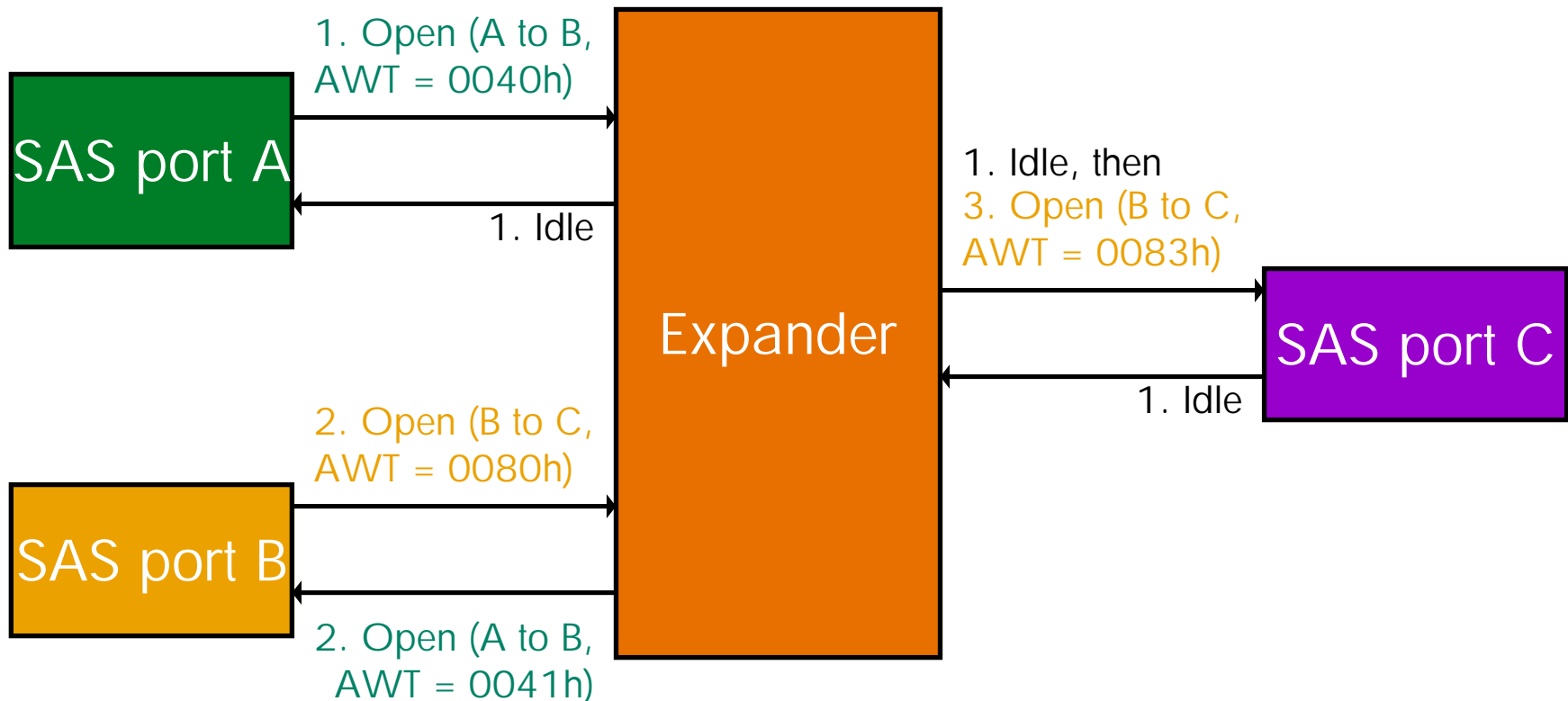
- Assume the initiators' OPENs are transmitted at the same time
- Although source address $B > A$, A's OPEN wins because its AWT is larger
- Expander increments AWT from the time the OPEN arrives until the time it forwards it



Arbitration fairness – OPENs crossing before AIP



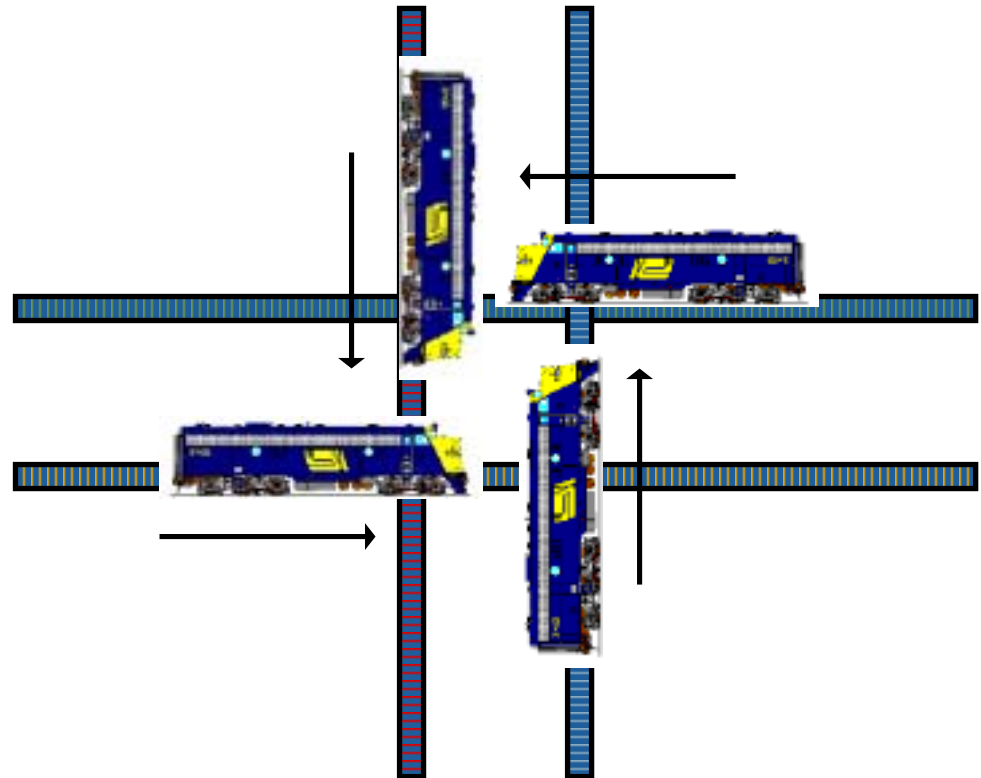
- Open (A to B) and Open (B to C) cross on the physical link
- Open (B to C) wins because its AWT is larger
- Expander forwards Open (B to C)



Link layer – Deadlocks and livelocks

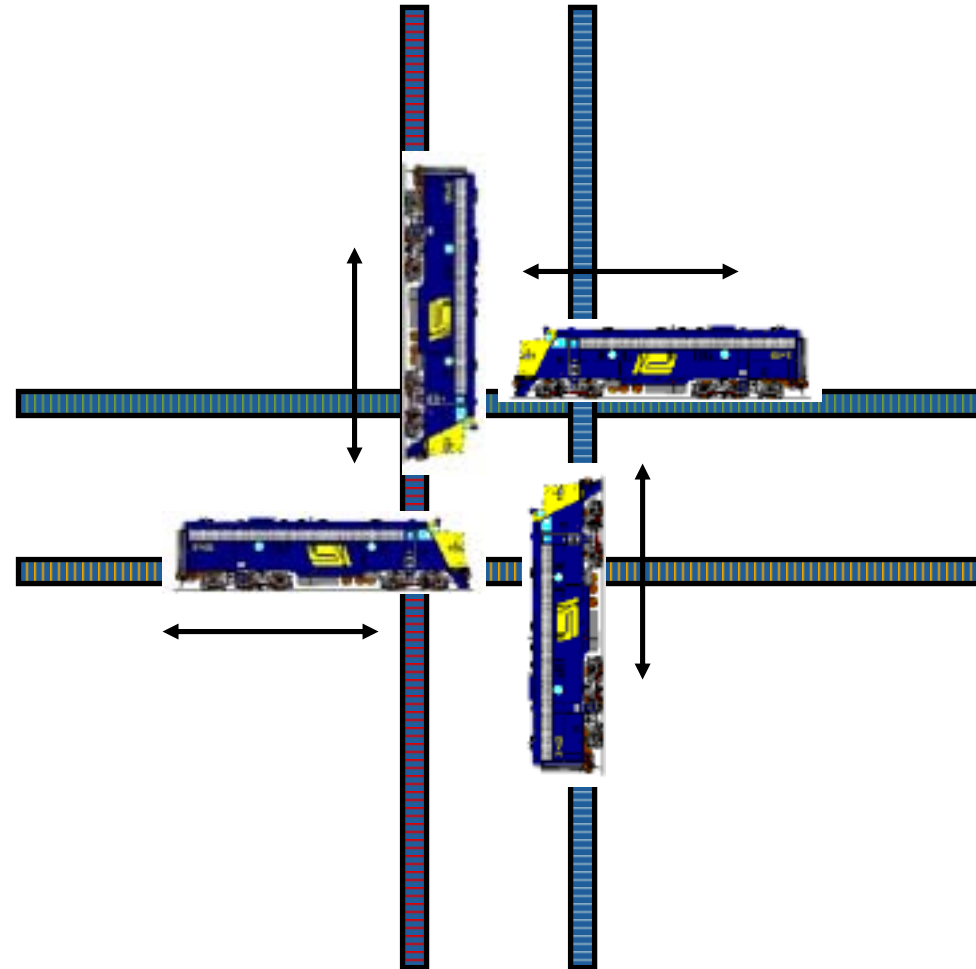
Deadlock

- Two or more processes are waiting on the other to complete, resulting in none completing
- Assume the trains can only go forward
- Assume trains are longer than an intersection
- For SAS, the trains represent connection requests and resulting connections occupying physical links in a cloud of expanders



Livelock

- Two or more processes continually change state in response to other changes in other processes, resulting in none completing
- Assume:
 - the trains detect the problem at the same time
 - Back up to the same distance
 - Go forward again
 - Always travel at the same speed
- No more progress than deadlock, although they're all busy
- Solutions to deadlock that just lead to livelock are incomplete



Conditions for deadlock



- 4 conditions must be simultaneously true for deadlock to exist
 - **Mutual exclusion**
 - there is a resource that is not shareable by two requesters
 - SAS: a physical link
 - **Hold and wait**
 - a requester holds a resource while requesting another
 - SAS: an initiator-to-expander physical link is occupied while the expander requests an expander-to-target physical link
 - **No preemption**
 - no way to override another requester
 - SAS: this is possible in a circular wait situation
 - **Circular wait**
 - there is a set of requesters [p1, p2, ... pN] such that p1 is waiting on p2, p2 is waiting on p3, etc. and pN is waiting on p1
 - SAS: this is possible with multiple expanders (multiple physical links)

Approaches to solve deadlock



- Two approaches to solving deadlock issues
 - Prevent and avoid
 - Implement rules that ensure that the 4 conditions can never be true at the same time
 - Detect and recover
 - Detect that all 4 conditions have occurred and do something to 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 approaches



- Avoid one of the 4 conditions and deadlocks are not a problem
 - Mutual exclusion
 - Allow sharing of the resources
 - Other fabrics like InfiniBand solve deadlocks by providing virtual lanes
 - 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 approaches part 2



- Preemption

- Allow resources to be taken over
- Problem: SAS connections and connection requests could be overridden; to always do this could create starvation
- Problem: cannot take over the proper resources when they are several expanders removed (circular wait)

- Circular wait

- Impose a global ordering and make sure all requesters request resources in order
- SAS addresses, being worldwide unique, can provide such ordering
- Problem: This can end up starving ports with lower priority SAS addresses. Counter to arbitration fairness.

Deadlock detection approach



- 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 suspect connection requests and force them to back off

Deadlock detection approach



- States of a connection request in an expander
 - **Waiting on full connection** – wait forever
 - Connections are assumed to be short-lived
 - If a SAS port leaves a connection open forever, it will cause problems
 - **Waiting on a partial pathway** – run the timer
 - Partial pathways waiting on partial pathways is the deadlock scenario

AIP primitives



- AIPs communicate arbitration state from expander to expander
 - AIP (NORMAL)
 - Accepted the OPEN address frame
 - Start to arbitrate internally for an output phy
 - AIP (WAITING ON DEVICE)
 - Forwarded the OPEN address frame; waiting for a reply
 - AIP (WAITING ON CONNECTION)
 - Waiting on an output phy currently involved in a 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 has not yet received AIP
 - Waiting on a phy which has passed through AIP (WAITING ON PARTIAL) [a Blocked Partial Pathway]

Partial Pathway Timeout timer

- Programmable initial value per expander phy
 - reported via SMP DISCOVER function
 - set via SMP PHY CONTROL function
 - Range: 0 to 15 μ s
 - Recommended value: 7 μ s
 - Want to provide enough time for the winning connection request to establish a connection
- Started if a connection request cannot get to an output phy, and one of the output phys is in a partial pathway
- When timer expires, reject the request with OPEN_REJECT (PATHWAY BLOCKED) if it is the lowest priority of those waiting on or using that output phy
 - Called **pathway recovery**

Pathway Recovery priority



- Competing requests are compared:

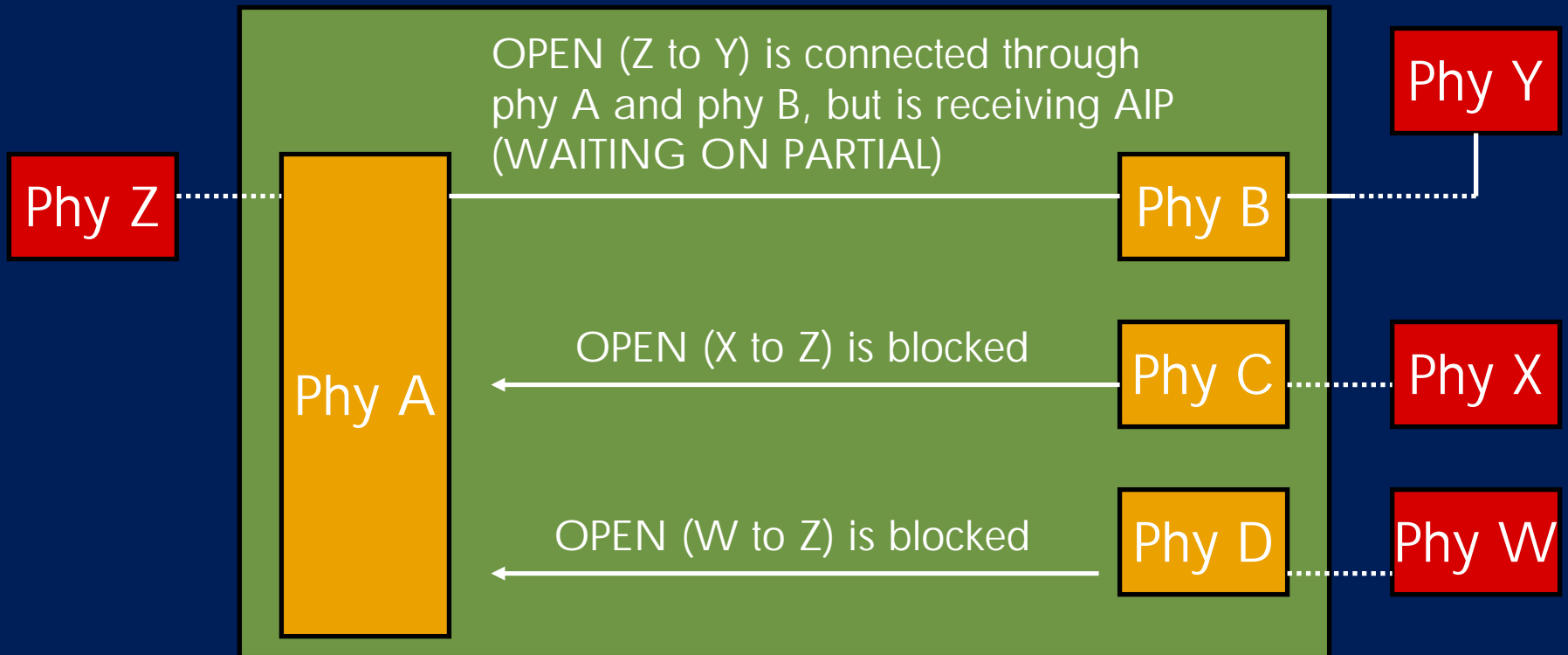
MSB		LSB
Pathway Blocked Count	Source SAS address	Connection rate

- Only the lowest priority request is rejected
 - The highest priority and all the middle priority requests remain
 - This breaks the deadlock
- By rejecting the lowest priority only, whatever request was waiting on it now proceeds, which then frees up the rest
 - Slower resolution if first reject doesn't clear the blockage
 - Rejecting all but the highest priority request also works
 - Results in extra retries of requests not contributing to the deadlock

Pathway Recovery Priority example



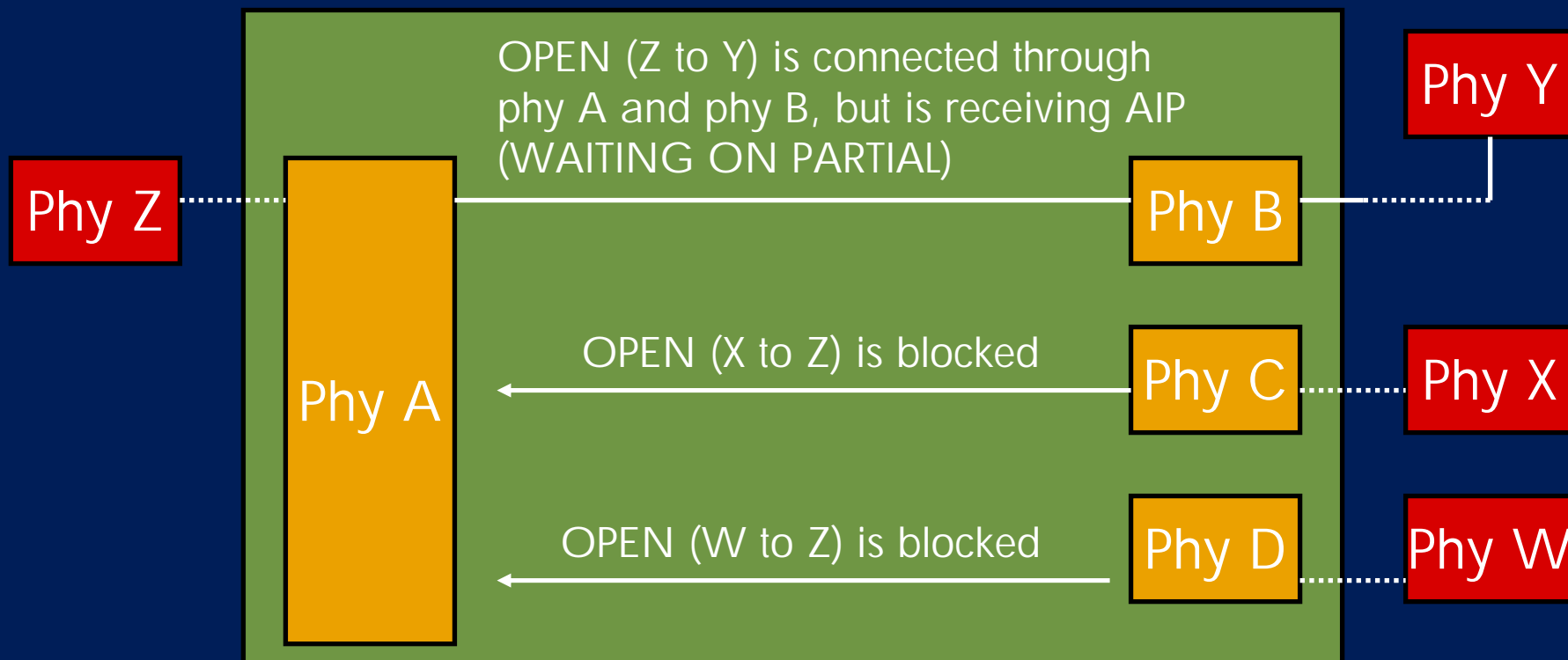
- Phy C and Phy D run their Partial Pathway Timeout timers
- Assuming $W < X < Z$, ECM instructs Phy D to reject its request with OPEN_REJECT (PATHWAY BLOCKED) when its timer expires
- If that caused the deadlock, clears the way for Z to Y
- If not, Phy C will be rejected next



Pathway Recovery Priority example 2



- Phy C and Phy D run their Partial Pathway Timeout timers
- Assuming $Z > X > Y$, ECM does nothing
- Whatever expander is blocking Z to Y should timeout and return OPEN_REJECT (PATHWAY BLOCKED), clearing the partial pathway between phy A and B
- No way for this expander to cancel the Z to Y connection



Pathway Blocked Count

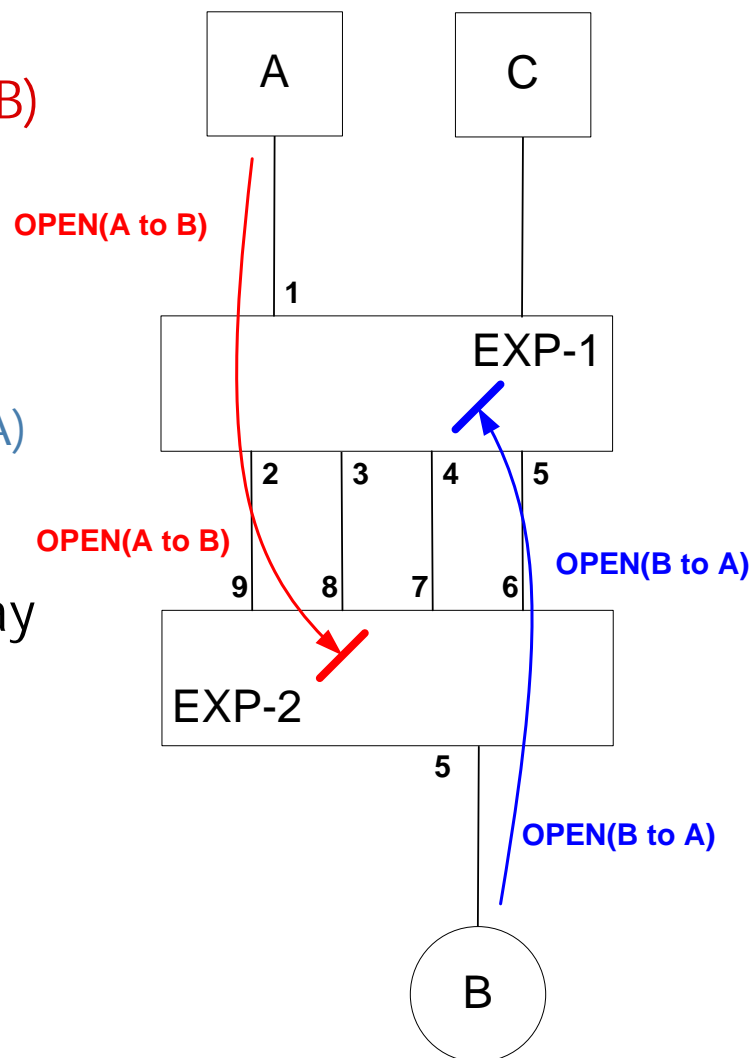


- Once a request is backed off for deadlock avoidance, don't want it to lose its place in line
- 8-bit **Pathway Blocked Count** field in OPEN address frame used to give it higher priority when retried

Simple deadlock example



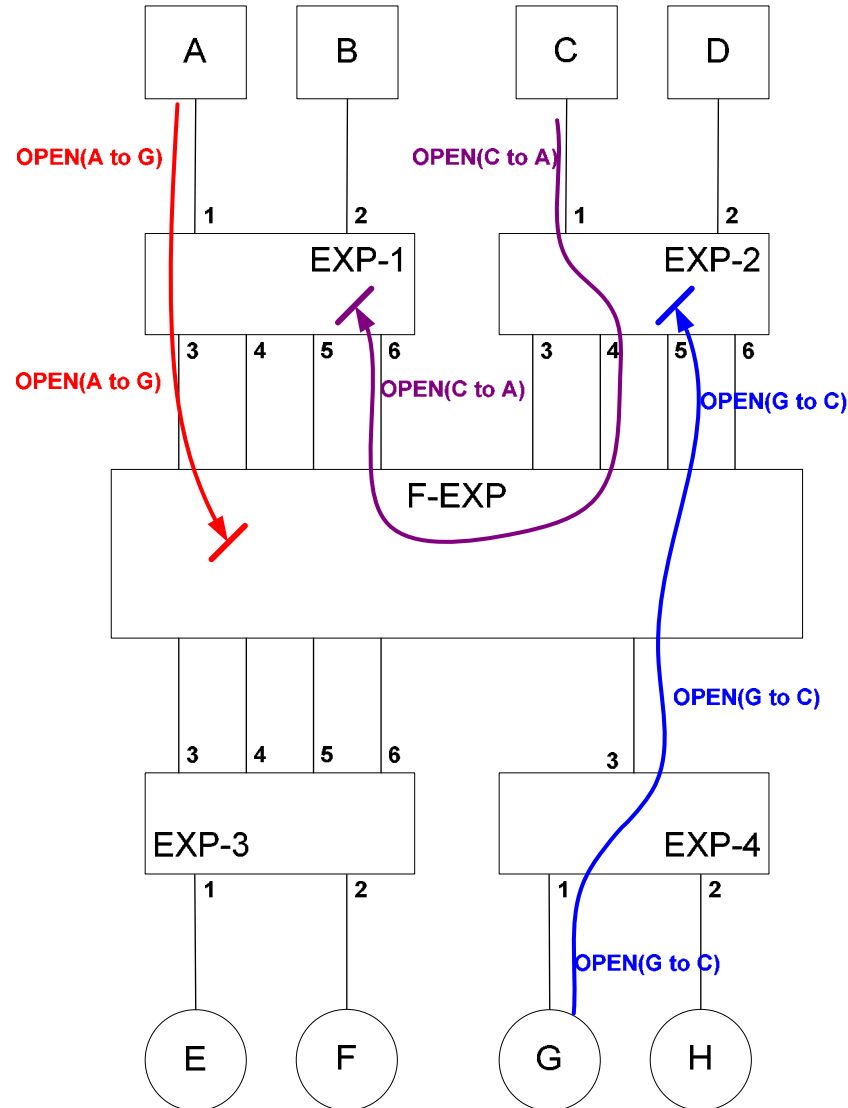
- In expander 1
 - OPEN (B to A) is waiting on OPEN (A to B)
 - OPEN (A to B) is a partial pathway, receiving AIP (WAITING ON PARTIAL) from Expander 2
- In expander 2
 - OPEN (A to B) is waiting on for OPEN (B to A)
 - OPEN (B to A) is a partial pathway receiving AIP (WAITING ON PARTIAL) from Expander 1
- Exp-1 Phy 5 and Exp-2 Phy 9 Partial Pathway Timeout timers expire
- Exp-1 Phy 5 sees that source B is higher priority than source A, so it waits
- 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



- **OPEN (A to G)** waiting on **OPEN (G to C)** waiting on **OPEN (C to A)**
- **OPEN (A to G)** will be backed off, letting **OPEN (C to A)** through, eventually letting **OPEN (G to C)** through
 - F-EXP sends OPEN_REJECT (PATHWAY BLOCKED) to A
 - Then **OPEN (C to A)** goes from EXP-1 to A
 - When connection closes, **OPEN (G to C)** goes to C
- When **OPEN (A to G)** is retried, it has a larger Pathway Blocked Count, giving it higher priority next time

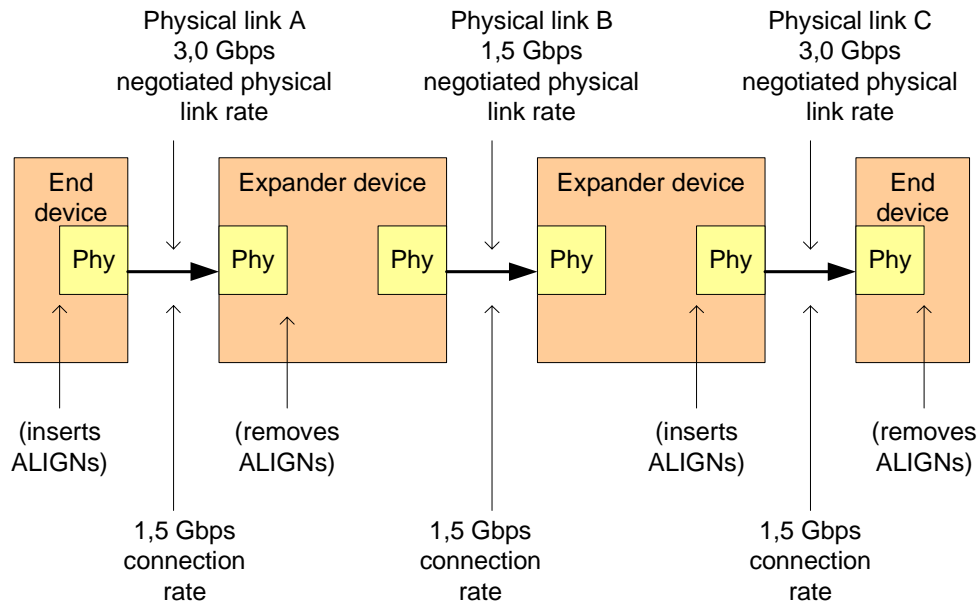


Link layer – Rate matching

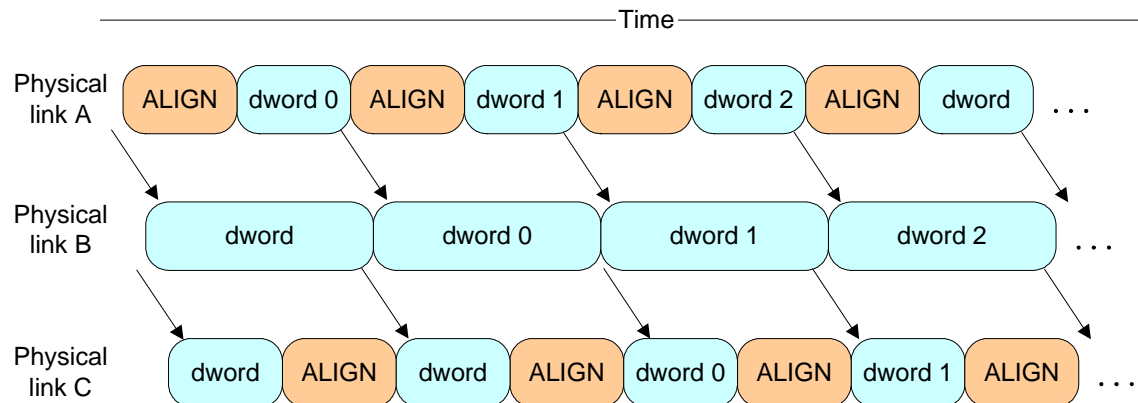
Rate matching



- Lets 1.5 Gbps and 3.0 Gbps SAS ports communicate
 - Especially 3.0 Gbps initiators and 1.5 Gbps SATA drives
- On faster physical links, insert ALIGN primitives to slow down the effective data rate to match the slowest physical link in the connection



Sample dwords on physical links (from left to right) during a 1,5 Gbps connection:



Connection Rate field



- Connection Rate field in OPEN address frame

Value	Rate
0h – 7h	Reserved
8h	1.5 Gbps
9h	3.0 Gbps
Ah	Reserved for 6.0 Gbps
Bh – Fh	Reserved for faster rates

- Connection rate must be less than or equal to the physical link rate
 - Cannot run 3 Gbps data stream over a 1.5 Gbps physical link

Connection rate rules



- 1.5 Gbps connection rate mandatory for all phys
 - Regardless of physical link rate
 - Expected to remain the rule forever
- Expanders
 - reject a connection request with OPEN_REJECT (CONNECTION RATE NOT SUPPORTED) if the connection rate is greater than the physical link rate of all potential output phys
 - If at least one phy supports the rate, expander will wait on it rather than reject the request
- Initiators
 - should use SMP discovery and pick a connection rate that will work
 - Can also just try the fastest and keep falling back
- Targets
 - don't do discovery; use whatever rate the initiator used to send the command (or previously open them)

Link layer – SSP (Serial SCSI Protocol)

- Inside an SSP connection...
- Phys exchange **SSP frames**
 - SSP frame = SOF primitive, data dwords, EOF primitive
 - Each frame results in an ACK or NAK primitive
- Credit-based flow control
 - Permission to send a frame must be granted with RRDY primitives
- Full duplex
 - SSP frames can be sent in both directions simultaneously
 - Independent credit for each direction
- ACK, NAK, and RRDY primitives may be interjected among frame dwords
 - (so can ALIGNs and NOTIFYs)

SSP frame definition



- SSP frame contents
 - 24 byte SSP frame header (defined in Transport layer)
 - 0 to 1024 data bytes, always multiple of 4 (defined in Transport layer)
 - 4 byte CRC
 - Minimum: 28 bytes (7 data dwords) (header + CRC)
 - Maximum: 1052 bytes (263 data dwords)
- No “frame size” field
 - Last dword before the EOF must be the CRC
 - Size of data implied by location of the EOF
- Transport layer defines frame contents; link layer just enforces rudimentary frame sizes and checks the CRC



SSP flow control



- A phy sends RRDY to grant credit to send a frame
 - Effect on the frame transmitter
 - Transmit frame = decrement credit
 - Receive RRDY = increment credit
 - Only allowed to send a frame when credit ≥ 1
- Maximum credit of 255
 - Since SAS is limited distance/low latency, not important to have large credits like in Fibre Channel (2 may be sufficient)
- No credit implied at start of connection
- No withdrawing credit once given
- Credit lost at end of connection
- Phy that grants credit must be prepared to accept a full 1 KB frame for each RRDY

SSP credit blocked



- A phy should send CREDIT_BLOCKED primitive if no RRDY is going to be available for 1 ms or longer
 - When this phy finishes transmitting frames, it will want to close the connection
 - The other phy might try to keep it open to transmit its frames
 - A 1 ms timeout will occur before it gives up
 - CREDIT_BLOCKED tells the other phy to start closing the connection rather than waiting for credit

SSP frame acknowledgement

- Every frame is acknowledged
 - ACK = positive acknowledgement
 - Frame received successfully (per the link layer)
 - NAK = negative acknowledgement
 - Used to report a CRC error
 - Bad frame sizes are just ignored – no ACK or NAK
- ACK or NAK must arrive within 1 ms or frame transmitter assumes something is wrong



- **Interlocked frames**

- COMMAND, TASK, XFER_RDY, RESPONSE frames (see Transport layer)
- Must receive ACK (or NAK) for previous frame before sending
- Forces a round-trip

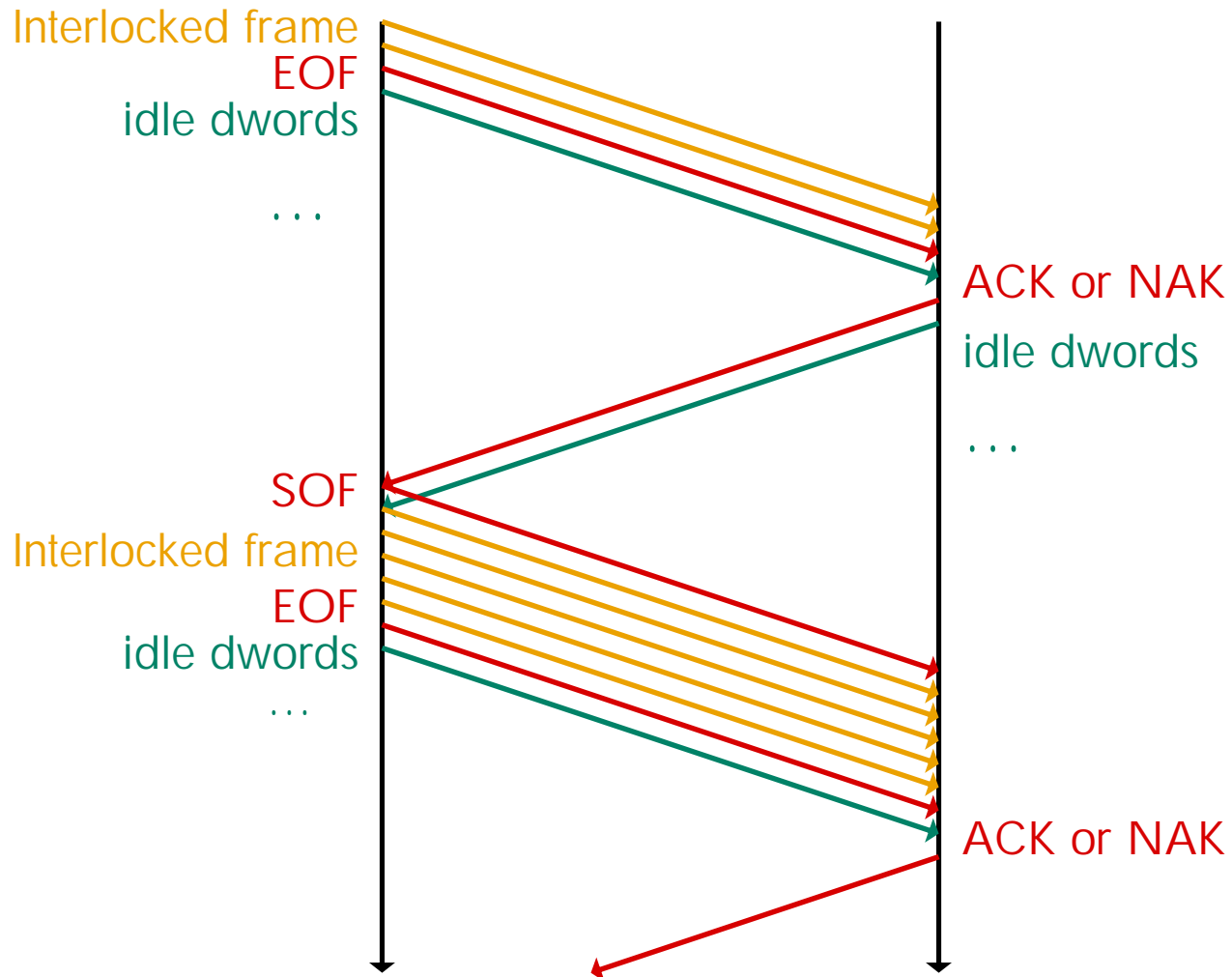
- **Non-interlocked frames**

- DATA frames (see Transport layer)
- If certain conditions are met, may send before ACKs (or NAKs) for previous frame(s) have been received
- Allows pipelining/streaming

SSP interlocked frames example



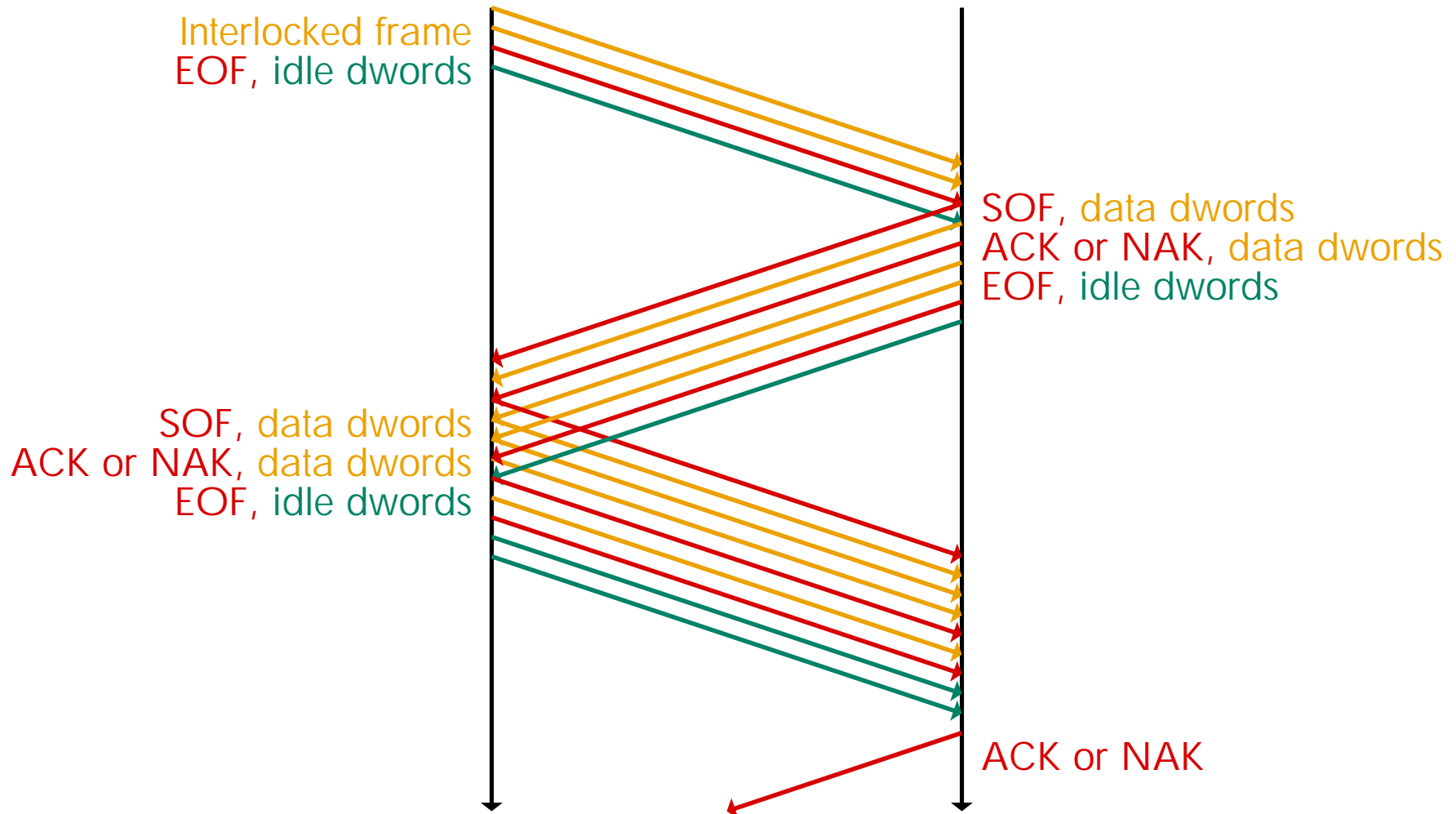
- Showing frames only in one direction



SSP interlocked frames example 2



- Showing frames in both directions



SSP non-interlocked frames

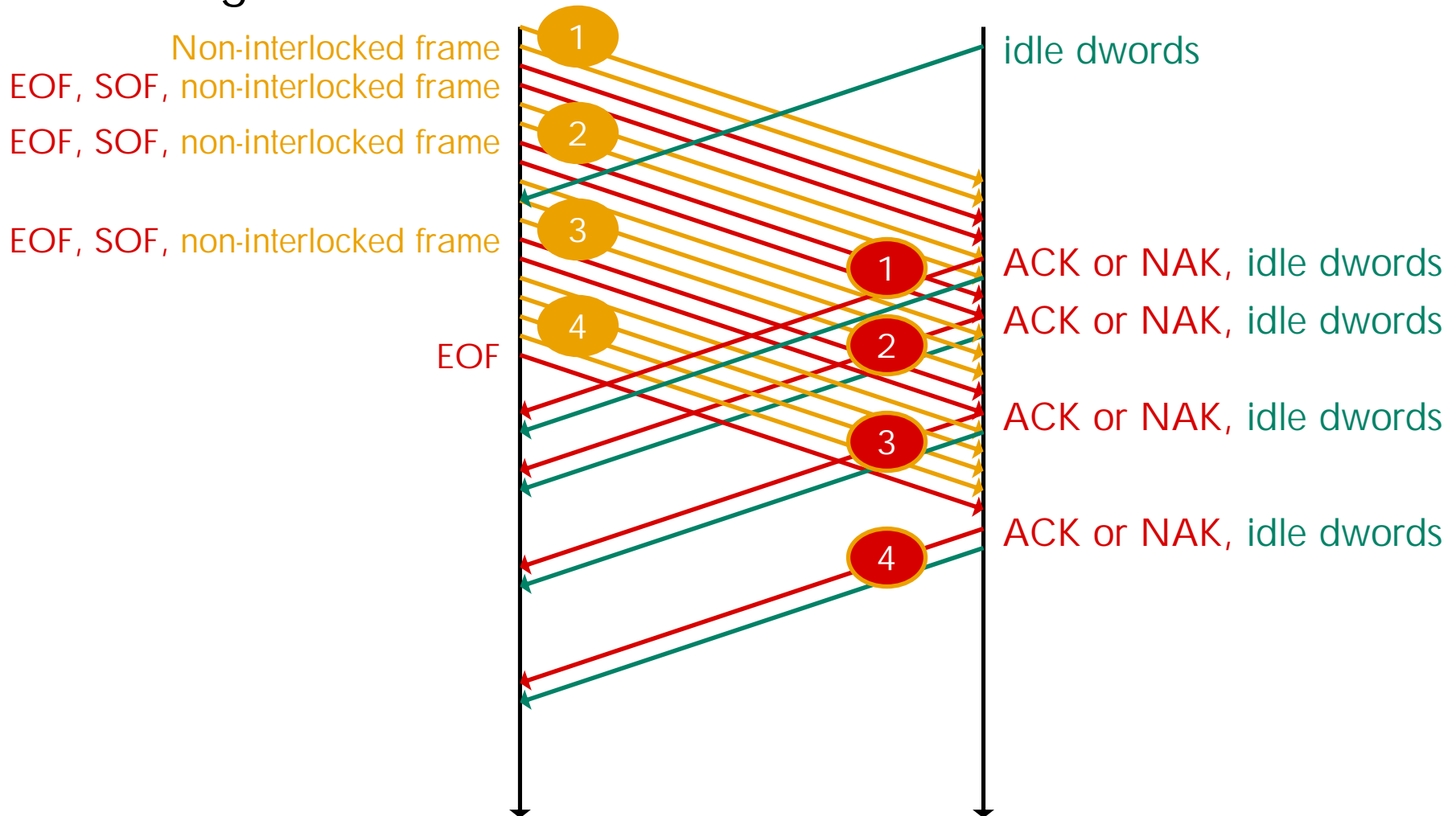


- Previous frame must also be a DATA frame with the same tag (see Transport layer)
 - hardware need not switch contexts during a 300 MB/sec data stream
- Credit must be available

SSP non-interlocked frames example



- Showing frames in one direction



SSP deadlock avoidance



- A phy that accepts an OPEN must provide at least one credit
 - If credit is not available, reply with OPEN_REJECT (RETRY)
- If a SAS phy transmits an OPEN, it must transmit at least one frame on that connection
 - Avoids livelocks with wide ports
- An initiator phy shall not refuse to provide credit just because it needs to transmit a frame itself
 - A target can do so if needed
 - Otherwise, initiator and target could wait on each other forever
 - Temporarily refusing to provide credit is fine (e.g. PCI bus congestion in an HBA)

Closing an SSP connection



- When done transmitting frames, transmit a DONE primitive
 - Traffic in other direction (to receive frames) unaffected
 - Must still transmit RRDY, ACK, NAK, and CREDIT_BLOCKED primitives to continue receiving frames
 - Other phy transmits a DONE when it is also finished
 - Like Fibre Channel's "Dynamic Half Duplex" mode
- When a phy has both transmitted and received DONE, it transmits CLOSE
- The CLOSEs will tend to overlap

DONE primitives



- Several reasons for DONE
 - DONE (NORMAL)
 - No more frames to transmit
 - DONE (ACK/NAK TIMEOUT)
 - Did not receive an ACK or NAK within 1 ms of transmitting a frame
 - DONE (CREDIT TIMEOUT)
 - No credit (no RRDY) for 1 ms after needing to transmit a frame
 - Received a CREDIT_BLOCKED primitive, which indicates credit will not be available within 1 ms

SSP state machines



- Lots of “state machines”
- Same set for SSP initiator phys and SSP target phys
 - SSP_TIM – transmit interlocked frame monitor
 - SSP_TCP – transmit frame credit monitor
 - SSP_D – DONE control
 - SSP_TF – transmit frame control
 - The only true state machine of the lot
 - Everything else has just one state
 - SSP_RF – receive frame control
 - SSP_RCM – receive frame credit monitor
 - SSP_RIM – receive interlocked frame monitor
 - SSP_TC – transmit credit control
 - SSP_TAN – transmit ACK/NAK control

- Timers

- 1 ms **ACK/NAK Timeout** – expect an ACK or NAK within 1 ms of transmitting a frame
- 1 ms **DONE Timeout** – expect frame transmission or DONE within 1 ms of transmitting DONE
 - If the connection is not being used, it will be forced closed
- 1 ms **Credit Timeout** – expect a new RRDY within 1 ms of having no credit



Link layer – SMP (Serial Management Protocol)

SMP overview



- Only an initiator can open an SMP connection
 - Target not allowed to open an initiator
- Inside an SMP connection...
- Two SMP frames are transferred
 - SMP frame = SOF primitive, data dwords, EOF primitive
 - 1. Initiator transmits one SMP_REQUEST frame to target
 - 2. Target transmits one SMP_RESPONSE frame to initiator

SMP simplifications



- Half duplex
 - Frames are never in flight in both directions at the same time
- No ACK or NAK primitives
 - Target sending a frame at all serves as positive acknowledgement
 - If CRC is bad, frame is ignored
- No flow control
 - no RRDY or CREDIT_BLOCKED primitives
 - Initiator has implicit “credit” after opening a connection
 - Target receiving a frame implicitly grants it “credit”

SMP frame definition



- Frame contents
 - 4 byte SMP frame header (defined in Transport layer)
 - 0 to 1024 data bytes, always multiple of 4 (defined in Transport and Application layers)
 - 4 byte CRC
 - Minimum: 8 bytes (2 data dwords) (header + CRC)
 - Maximum: 1032 bytes (258 data dwords)
- No “frame size” field
 - Last dword before the EOF must be the CRC
 - Size of data implied by location of the EOF
- Transport layer and application layer define frame contents; link layer just enforces rudimentary frame sizes and checks the CRC

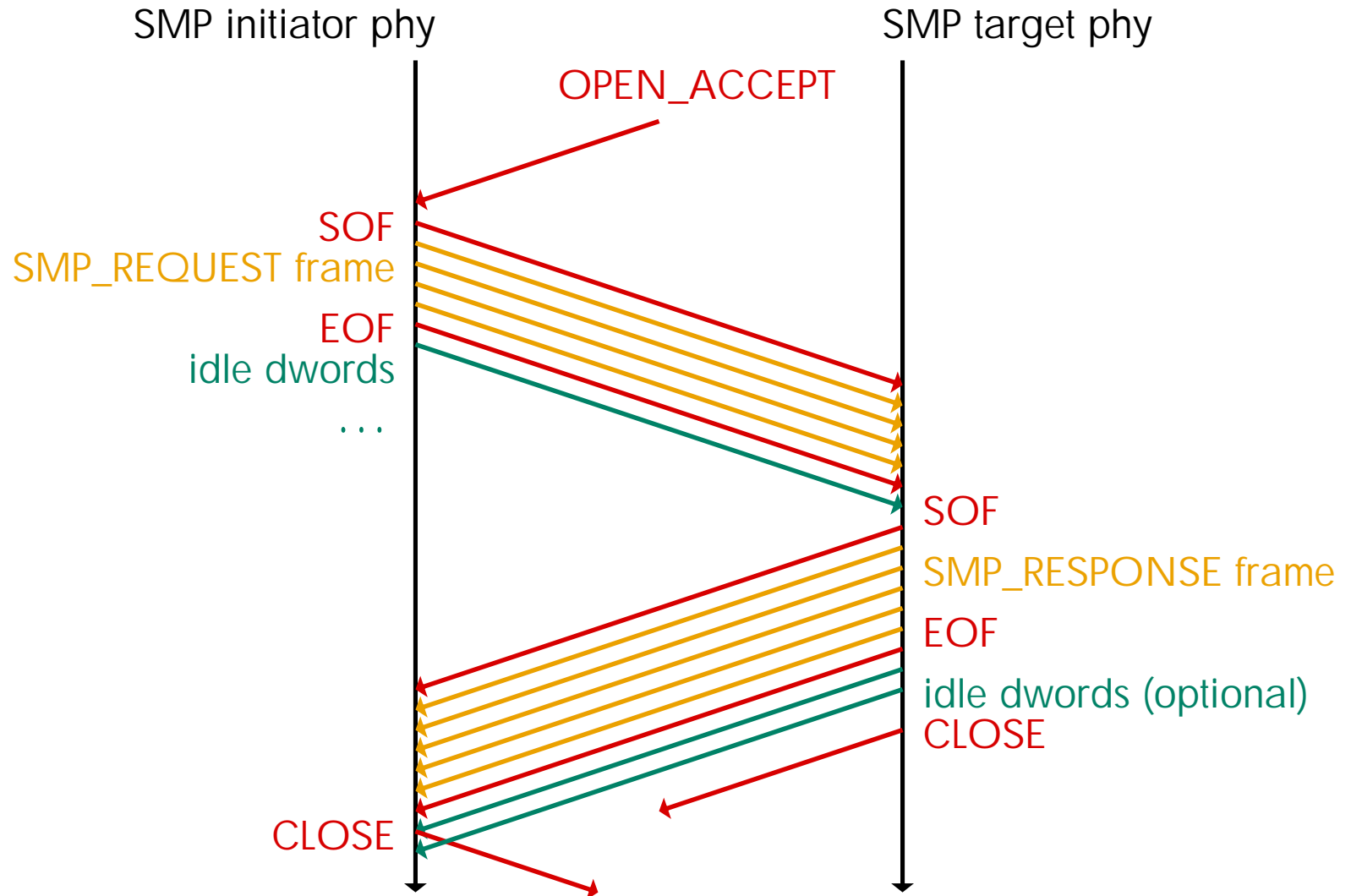


Closing an SMP connection



- No DONE primitives
- After target sends its frame, send CLOSE
- After initiator receives the frame, send CLOSE
- Closes will tend to overlap

SMP example



SMP state machines



- Different state machines for SMP initiator phys and SMP target phys
 - SMP_IP – Link layer for SMP initiator phys
 1. Transmit a frame
 2. Wait to receive a frame
 3. Finished
 - SMP_TP – Link layer for SMP target phys
 1. Wait to receive a frame
 2. Transmit a frame
 3. Finished

Link layer – STP (Serial ATA Tunneling Protocol) and Serial ATA

STP and SATA overview



- In SATA, SATA host and SATA device just communicate directly – no connections
- In SAS, once an STP connection is open, STP initiator and STP target communicate as if they were SATA host and SATA device directly attached on a physical link
 - Extra latency introduced as dwords flow through expanders
- Half duplex
 - SATA never transmit frames in both directions at one time
 - Usually the frame goes one way and R_IP primitives go the other way
- Note: SAS prefixes the SATA primitive names with “SATA_”
 - SATA_SOF (used in STP) is different than SOF (used in SSP)
 - In this presentation, prefix not always used

SATA repeated primitives



- Most primitives are repeated
 - The number that appear doesn't matter
 - Primitive is repeatedly transmitted until something else happens
 - Not repeated: SOF, EOF
 - Repeated: X_RDRY, R_RDY, R_IP, R_OK, R_ERR, HOLD, HOLDA, WTRM, SYNC
 - During repeats, SATA_CONT is used to enter scrambled data mode
 - Prevents EMI problems from repeated patterns
 - Mandatory in SAS STP, optional in SATA

SATA frame definition



- Frame contents
 - 4 byte SATA frame header (defined in Transport layer)
 - 0 to 8192 bytes, always multiple of 4 (defined in Transport layer)
 - Called the FIS – Frame Information Sequence
 - 4 byte CRC
 - Minimum: 8 bytes (2 data dwords) (header + CRC)
 - Maximum: 8204 bytes (1026 data dwords)
- No “frame size” field
 - Last dword before the EOF must be the CRC
 - Size of data implied by location of the EOF
- Transport layer and application layer define frame contents; link layer just enforces rudimentary frame sizes and checks the CRC

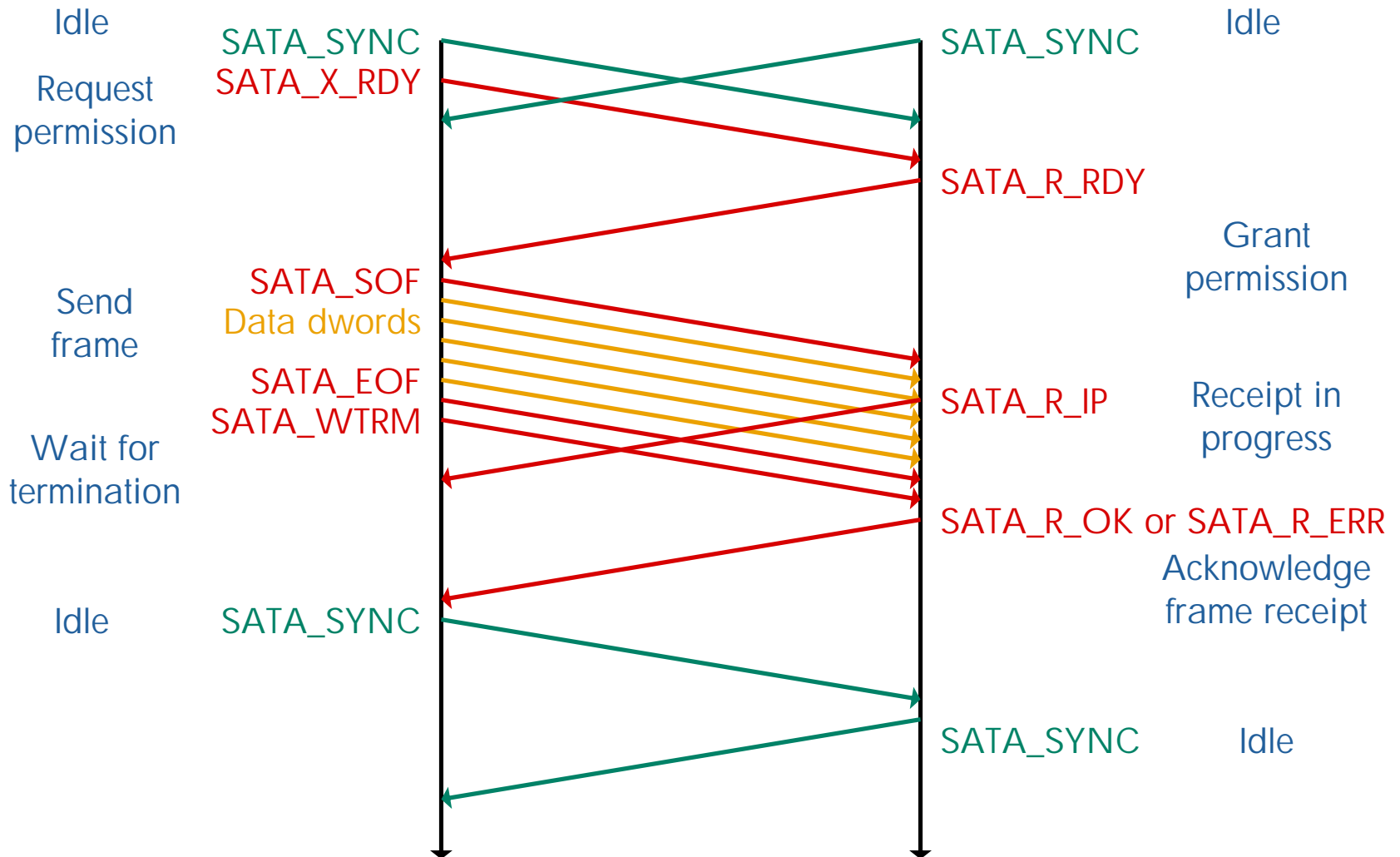


SATA frame transmission



- To transmit a frame...
 1. Start from idle, transmitting SYNC primitives
 2. Request permission by transmitting X_RDY primitives
 3. Receive permission by receiving R_RDY primitives
 4. Transmit frame (SATA_SOF, data dwords, SATA_EOF)
 5. Transmit WTRM primitives while waiting for a response
 6. Receive R_OK or R_ERR primitives indicating positive or negative acknowledgement
 7. Return to idle and transmit SYNC primitives

SATA basic frame transmission



SATA frame transmission notes



- SATA_X_RDY primitives may cross on the physical link
 - Host (initiator) always defers to device (target)
 - Host replies with SATA_R_RDY primitives
 - (not mentioned in the text, but the SATA state machines work this way)

- Receiver can pause an incoming frame
 - Receiver transmits HOLD primitives while receiving a frame
 - HOLD primitives travel in opposite direction of frame
 - Transmit when 20 dwords of buffer are still available
 - Within 20 dwords, frame transmission must stop
 - During the pause, frame transmitter transmits HOLDA primitives
 - When receiver is ready to resume, stop transmitting HOLD primitives
 - Frame transmitter stops transmitting HOLDA primitives and resumes transmitting data dwords for the frame
 - Only supposed to be used for DATA frames

SATA flow control budget



- SATA budgets zero transmission time on the wire
 - Frame transmitter can send 20 data dwords after receiving HOLD
 - Frame receiver can expect only 20 data dwords after transmitting HOLD
- This does not add up
 - One side has to do better (send less or expect more)
 - In practice, the frame transmitter stops as soon as possible, and does not reach the 20 data dword limit

SAS STP flow control budget



- SAS supports a 10 m external cable with significant wire delay
 - One-way propagation delay: $10 \text{ m} * 5 \text{ ns/m} = 50 \text{ ns}$
 - Round-trip propagation time: $50 \text{ ns} * 2 = 100 \text{ ns}$
 - Time to send a dword at 1.5 Gbps = $0.667 \text{ ns/bit} * 40 \text{ bits} = 26.667 \text{ ns}$
 - Number of dwords on the wire during round-trip propagation time = $100 \text{ ns} / 26.667 \text{ ns} = 3.75$

SAS STP flow control rules for SAS physical links

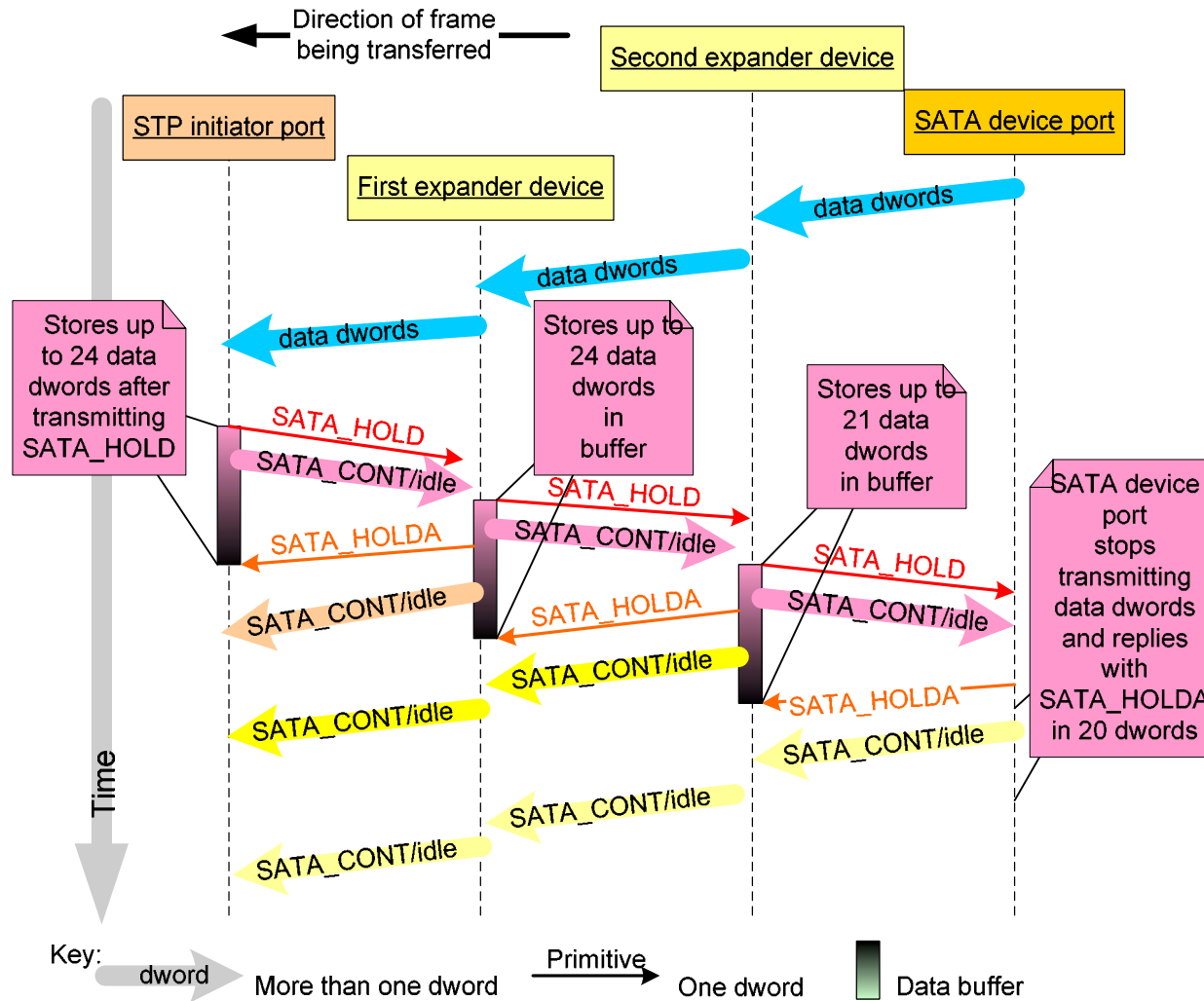


- On SAS physical links with STP connections

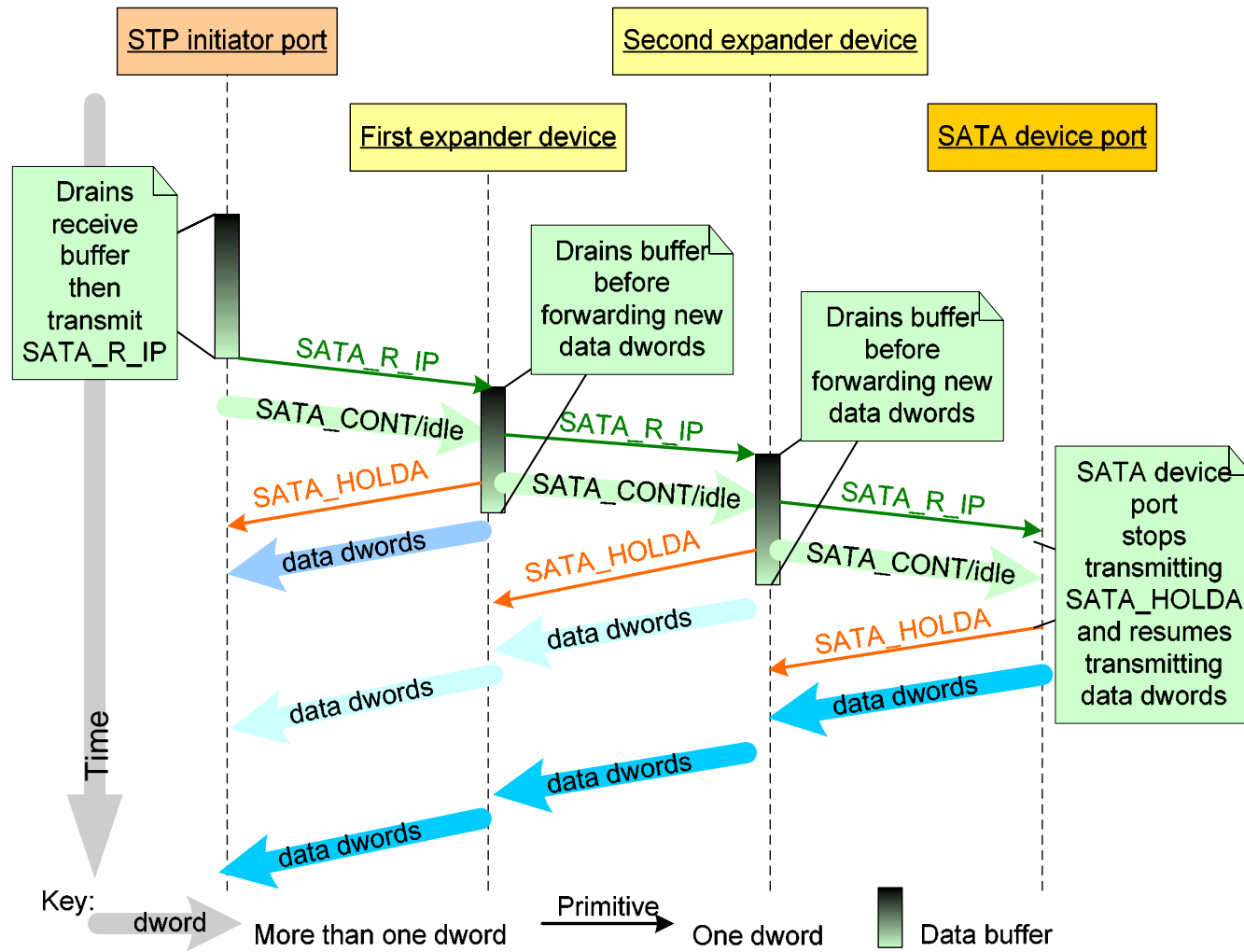
Connection rate	Frame receiver must allow ... after transmitting SATA_HOLD	Frame transmitter must transmit no more than ... after receiving SATA_HOLD
1.5 Gbps	24 data dwords	20 data dwords
3.0 Gbps	28 data dwords	20 data dwords

- Larger buffer sizes may be implemented to support longer cables
 - Vendor-specific value-add
 - Transmitters can wait until near the limit of 20 to try to avoid invoking SATA_HOLDS
- Expanders must guarantee these rules on each physical link
 - Requires buffering of up to 28 dwords in each expander

SAS STP flow control example part 1



SAS STP flow control example part 2



Expander flow control rules for SATA physical links



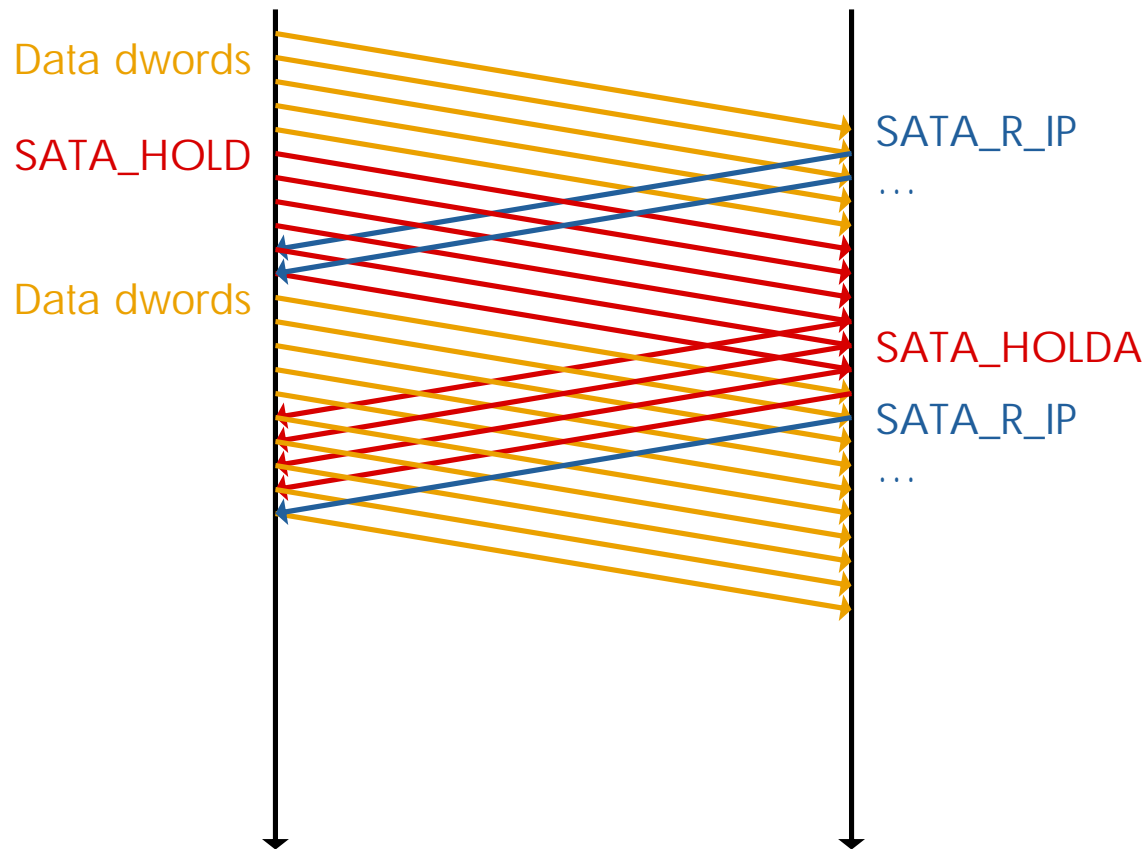
- For expanders bridging to SATA devices
 - Assume the SATA physical link follows SATA rules
 - 1 m maximum cable length
 - Not zero propagation time, but less than 1 dword worth
 - Expander as frame transmitter, after receiving SATA_HOLD
 - Transmit no more than 19 data dwords
 - Assumes SATA device buffers only 20 dwords after transmitting SATA_HOLD
 - Expander as frame receiver, after transmitting SATA_HOLD
 - Allows up to 21 data dwords
 - Assumes SATA device transmits 20 dwords after receiving SATA_HOLD

SATA outbound flow control



- While transmitting a frame, the source may temporarily run out of data
 - Not always desirable or allowed to end the frame at that point
- Solution – stall
 - Insert SATA_HOLD primitive while waiting for data
 - Frame receiver replies with SATA_HOLDA rather than R_IP while waiting
 - Frame transmitter does not wait for SATA_HOLDA; can return to data dwords at any time
- Only supposed to be used for DATA frames
- Could also insert ALIGNs to stall
 - In SATA, only one ALIGN encoding; EMI issues
 - SATA_HOLD can be followed by SATA_CONT and scrambled data (repeated primitive)

SATA transmitter flow control



- In SATA, time on wire is negligible; insignificant delay from transmitter to receiver

Opening an STP connection



- Open an STP connection whenever a phy needs to transmit a frame
- After OPEN_ACCEPT, usually follow with SATA_X_RDY requesting permission to transmit
- Might receive an SATA_X_RDY after the OPEN_ACCEPT
 - Follow SATA rules (initiator/host always defers to the target/device and has to change to an SATA_R_RDY)

Closing an STP connection

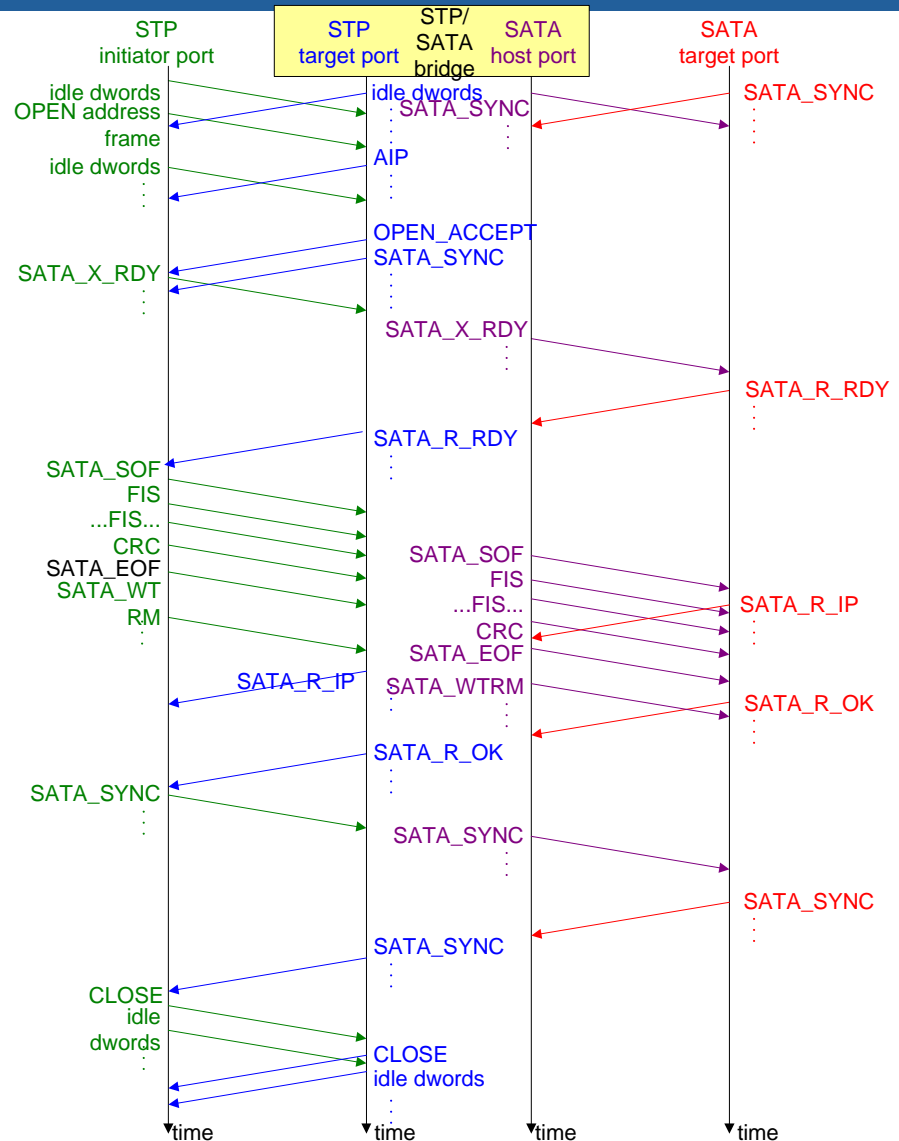


- Either side can close the connection when there are no frames in flight
- After last frame transfer, must exchange SATA_SYNCs before CLOSE is allowed
- After transmitting SATA_X_RDY, not allowed to transmit CLOSE
- After receiving SATA_X_RDY, may transmit CLOSE
- Expanders (STP/SATA bridges) should close "often" to avoid congestion
- Initiator policies are more vendor-specific

STP initiator sends frame to SATA device



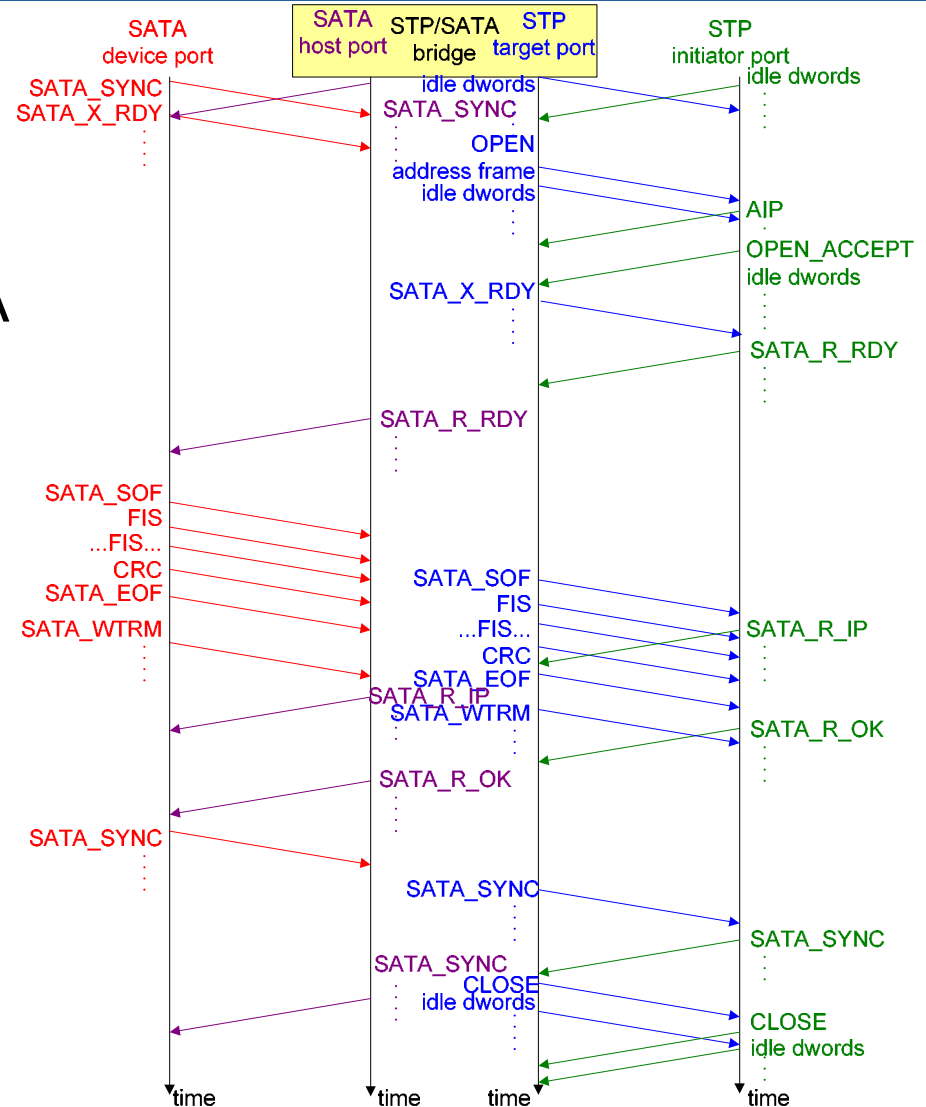
- STP initiator opens a connection
- Transmits SATA_X_RDY, receives SATA_R_RDY
- Transmits a frame
- After receiving SATA_SYNC, closes the connection



SATA device sends frame to STP initiator



- SATA device requests permission to transmit a frame with SATA_X_RDY
- STP target in the STP/SATA bridge opens a connection
- SATA_X_RDY and SATA_R_RDY forwarded
- SATA device transmits a frame
- SATA device returns to idle
- STP target chooses to close the connection



STP affiliations



- True SATA devices don't know about multiple hosts
 - SATA is supposed to be point-to-point, host-to-device
- STP target ports can understand multiple hosts
- STP/SATA bridge to a true SATA device may choose to limit access to one STP initiator at a time – called **affiliation**
- Affiliation is granted to first STP initiator to access the STP target port when it has no affiliation

STP affiliations 2



- Affiliation is held until:
 - Power on
 - Connection closed with CLOSE (CLEAR AFFILIATION) primitive
 - SMP PHY CONTROL function used to reset or clear the expander phy attached to the SATA device
 - SATA physical link undergoes a phy reset sequence
- Unaffiliated initiators receive OPEN_REJECT (STP RESOURCES BUSY) when trying to open the STP target port

SATA interface power management



- Put the phys to sleep to save power

State	Phys transmit	Exit within	Description
Active/ Phy Ready	1.5 Gbps	n/a	Fully operational
Partial	D.C. idle	10 μ s	Less than Active, more than Slumber
Slumber	D.C. idle	10 ms	Lowest power consumption

- This is just SATA interface power management
- Higher-level ATA commands control ATA IDLE and SLEEP modes independent of this (see Application layer)

SATA power management primitives



- Either phy may request entry with SATA_PMREQ_P or SATA_PMREQ_S
 - A mobile disk drive might aggressively try to put the interface to sleep, wake up when its seek completes
- Reply with SATA_PMACK or SATA_PMNAK
 - Unfortunately, SATA_PMNAK can mean either “not supported” or “try again”
- Wake up with **COMWAKE** OOB signal
 - Followed by SATA speed negotiation
- Not supported on SAS physical links
 - STP initiators always reply with SATA_PMNAK

Other SATA link layer notes



- DMAT primitive requests that a DMA be terminated early
 - Requests the frame transmitter stop sending the data frame (send the CRC dword and SATA_EOF as soon as possible)
 - Not required to be honored immediately (or ever)
 - Not recommended for use anymore
- 8192 byte frame size limit is not absolute
 - Parallel ATA to Serial ATA bridges allowed to violate
- Not required to check CRC on data frames before committing to storage
 - Control frames should be checked, however
- SATA_SYNC always sends the link layer state machines back to idle

SATA link layer state machine



- One link layer state machine
- Same for SATA host and SATA device
- Four sets of states
 - L and LS: Miscellaneous states
 - L1:L_IDLE – Transmits SYNC
 - LT: Link frame transmit states
 - Transmits X_RDY, SOF, data dword (including CRC), HOLDA, HOLD, EOF, or WTRM
 - LR: Link frame receive states
 - Transmits R_RDY, SYNC, R_IP, DMAT, HOLD, HOLDA, R_OK, or R_ERR
 - LPM: Link power management states
 - Transmits PMREQ_P, PMREQ_S, PMACK, or PMNAK

STP link layer state machine



- Not described by the SAS standard
- Implementation must add SAS connection management to the SATA link layer state machine
 - Open connection before sending SATA_X_RDY
 - Close connection when idle
 - Interact with port layer and support wide STP ports
 - Implement affiliations



Wrap up

Serial Attached SCSI tutorials



- General overview (~2 hours)
- Detailed multi-part tutorial (~3 days to present):
 - Architecture
 - Physical layer
 - Phy layer
 - Link layer
 - Part 1) Primitives, address frames, connections
 - Part 2) Arbitration fairness, deadlocks and livelocks, rate matching, SSP, STP, and SMP frame transmission
 - Upper layers
 - Part 1) SCSI application and SSP transport layers
 - Part 2) ATA application and STP/SATA transport layers
 - Part 3) Management application and SMP transport layers, plus port layer
 - SAS SSP comparison with Fibre Channel FCP

Key SCSI standards



- Working drafts of **SCSI** standards are available on <http://www.t10.org>
- Published through <http://www.incits.org>
 - Serial Attached SCSI
 - SCSI Architecture Model – 3 (SAM-3)
 - SCSI Primary Commands – 3 (SPC-3)
 - SCSI Block Commands – 2 (SBC-2)
 - SCSI Stream Commands – 2 (SSC-2)
 - SCSI Enclosure Services – 2 (SES-2)
- **SAS connector** specifications are available on <http://www.sffcommittee.org>
 - SFF 8482 (internal backplane/drive)
 - SFF 8470 (external 4-wide)
 - SFF 8223, 8224, 8225 (2.5", 3.5", 5.25" form factors)
 - SFF 8484 (internal 4-wide)

Key ATA standards



- Working drafts of **ATA** standards are available on <http://www.t13.org>
 - Serial ATA 1.0a (output of private WG)
 - ATA/ATAPI-7 Volume 1 (architecture and commands)
 - ATA/ATAPI-7 Volume 3 (Serial ATA standard)
- **Serial ATA II** specifications are available on <http://www.t10.org> and <http://www.serialata.org>
 - Serial ATA II: Extensions to Serial ATA 1.0
 - Serial ATA II: Port Multiplier
 - Serial ATA II: Port Selector
 - Serial ATA II: Cables and Connectors Volume 1

For more information



- International Committee for Information Technology Standards
 - <http://www.incits.org>
- T10 (SCSI standards)
 - <http://www.t10.org>
 - Latest SAS working draft
 - T10 reflector for developers
- T13 (ATA standards)
 - <http://www.t13.org>
 - T13 reflector for developers
- T11 (Fibre Channel standards)
 - <http://www.t11.org>
- SFF (connectors)
 - <http://www.sffcommittee.org>
- SCSI Trade Association
 - <http://www.scsita.org>
- Serial ATA Working Group
 - <http://www.serialata.org>
- SNIA (Storage Networking Industry Association)
 - <http://www.snia.org>
- Industry news
 - <http://www.infostor.com>
 - <http://www.byteandswitch.com>
 - <http://www.wwpi.com>
 - <http://searchstorage.com>
- Training
 - <http://www.knowledgetek.com>



i n v e n t