# Overview of advanced storage technologies and storage virtualization

Pablo Pérez Trabado

Dept. of Computer Architecture

University of Malaga (Spain)

# Disclaimers

- Some of the figures in the slides are taken from SNIA Education tutorials. To comply with the SNIA conditions of use for these documents, any SNIA tutorial used in the elaboration of these slides has been also provided as a handout

- Many (but not all) figures in the slides have been taken from Internet-available resources, including Wikipedia. Whenever possible by copyright restrictions, the original source has been also provided as a handout along with the slides. Also, whenever possible the original source is quoted. In any case, no ownership claims are made over images in the slides not drawn by the author himself.

# SCSI Interface

# (SCSI-3 specification)

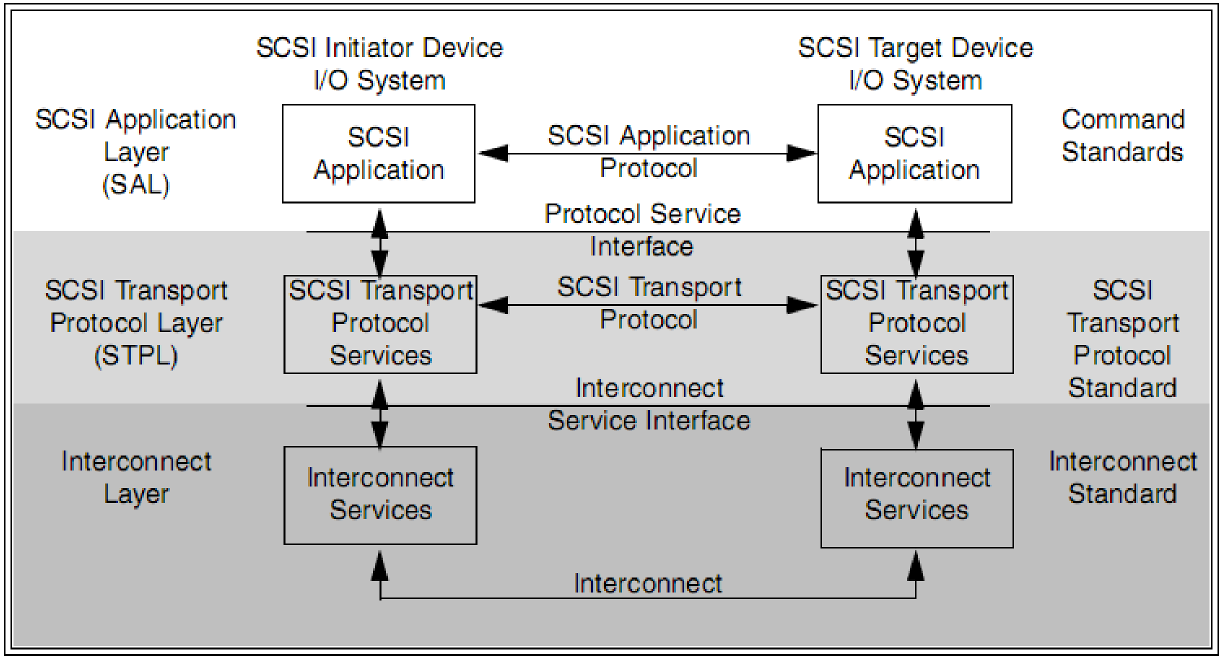
# What will we learn?

- SCSI interface structure and basic operation
- How the computer can, through its SCSI HBA (card), request a disk to perform a read or write operation
- How the SCSI interface allows the disk controller to hide what's going under the hood
- The concept of target, logic unit and LUN
- A detailed description of how a SCSI command is performed at the interface level
- How the SCSI interface is the *lingua franca* of all storage systems

# SCSI-3 History

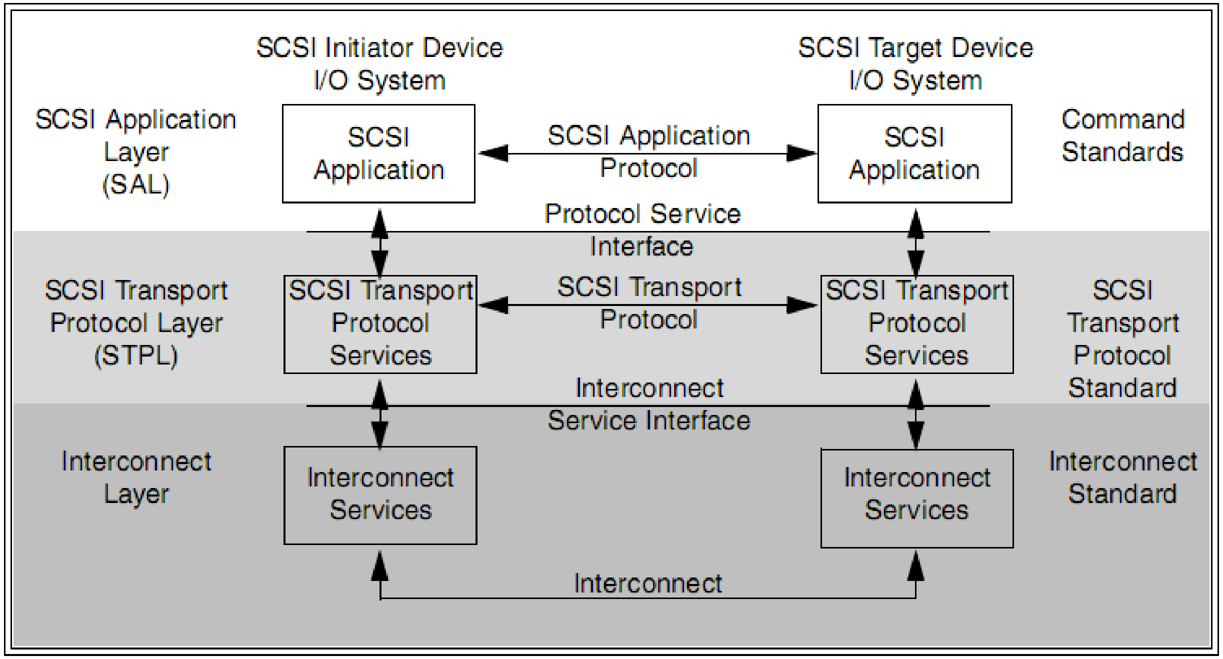
- SCSI interface created in 1978 by Shugart Associates, first standarized 1981
  - **SCSI** = *Small Computers System Interface*
- Developed as logical interface to drive high-performance disks over 8-bit SCSI parallel bus
- Very successful and widely accepted
  - Extended to control all kinds of storage peripherals
- Around 1995 begins development of SCSI-3 standard, published in 2003
- SCSI-3 is no longer a single standard, but a “standards family” (set of interrelated standards)
  - Suffix “-3” has been dropped, so nowadays “SCSI” means “SCSI-3”

# SCSI layered protocol stack



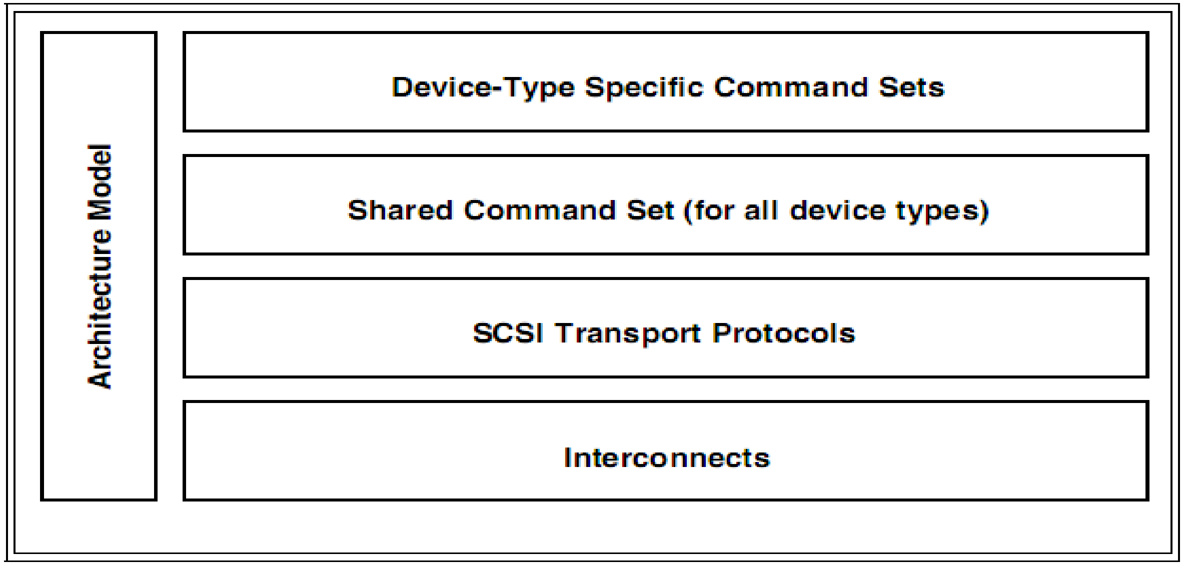

- SCSI-3 models SCSI interface as objects, protocol layers and service interfaces
  - Concept very similar to OSI computer networks reference model
- SCSI objects (protocol entities) exchange frames between peers
  - Frames get encapsulated when travelling downwards through protocol stack and de-encapsulated when travelling upwards

# SCSI layered protocol stack



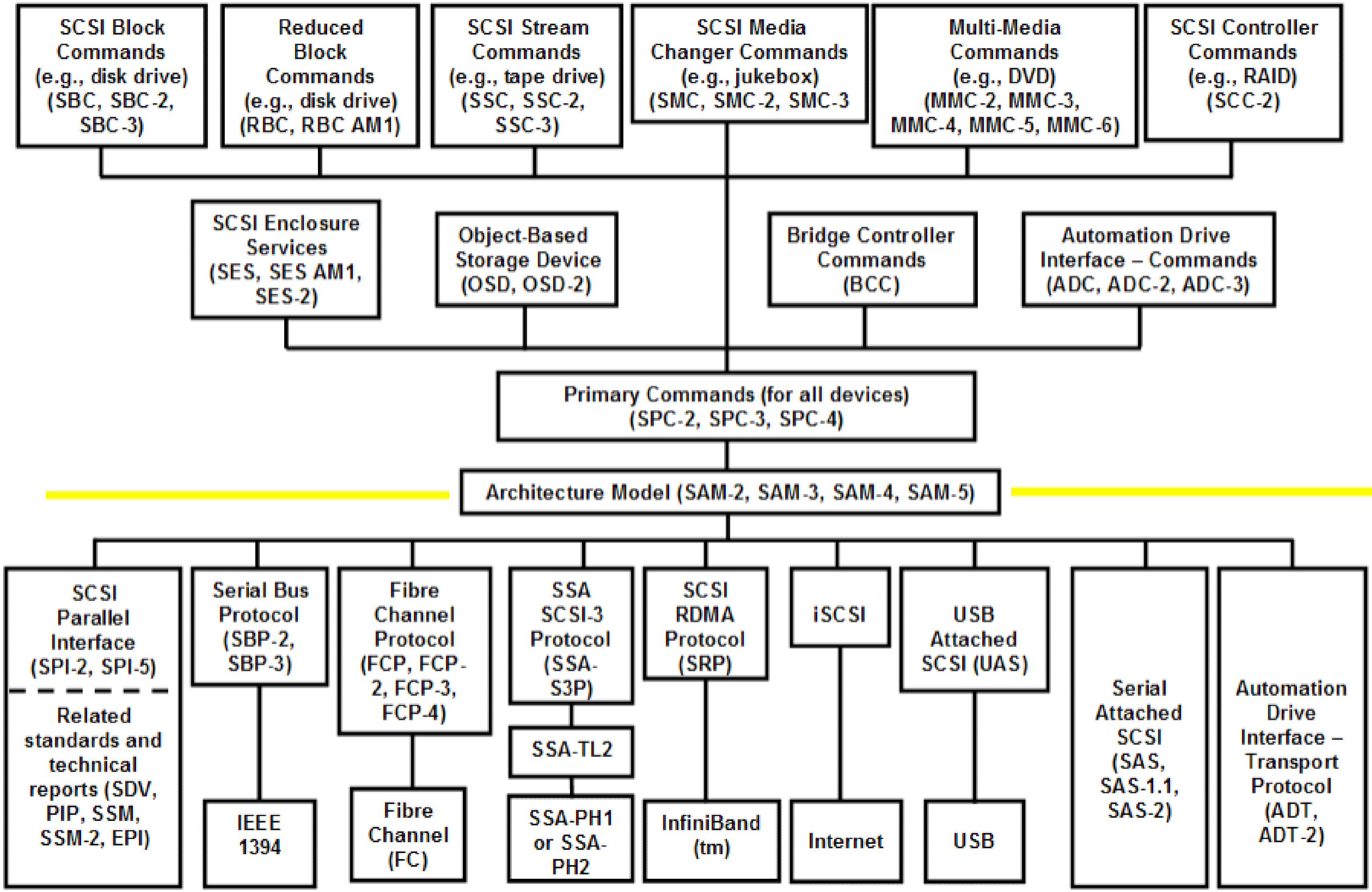

- Model has three layers:
  - Application layer: Commands sent from initiator to target, answers from initiator with data and status info
  - Transport layer: Protocols to wrap SCSI payload within link-level frames
  - Interconnect layer: Essentially, equivalent to link and physical layers of OSI model

# SCSI standards family

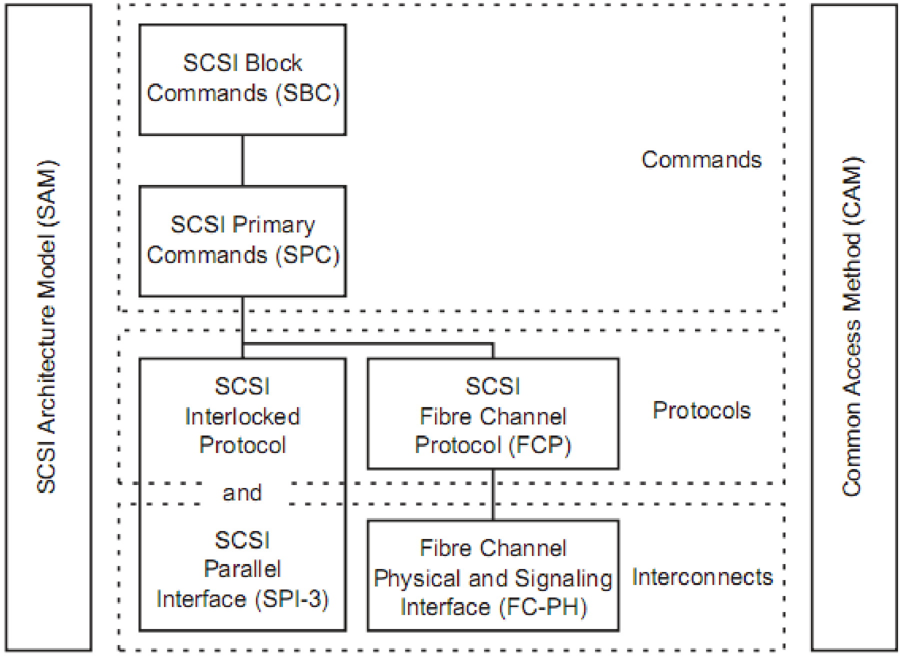



- SCSI-3 standards family defines:
  - A common architecture model, standardizing concepts and behaviors in all layers
  - Set of commands for the application layer
- Commands organized in two different subsets:
  - Primary commands: MUST be implemented by any SCSI device
    - Basic device identification and management
  - Device-specific sets of commands

# SCSI standards family

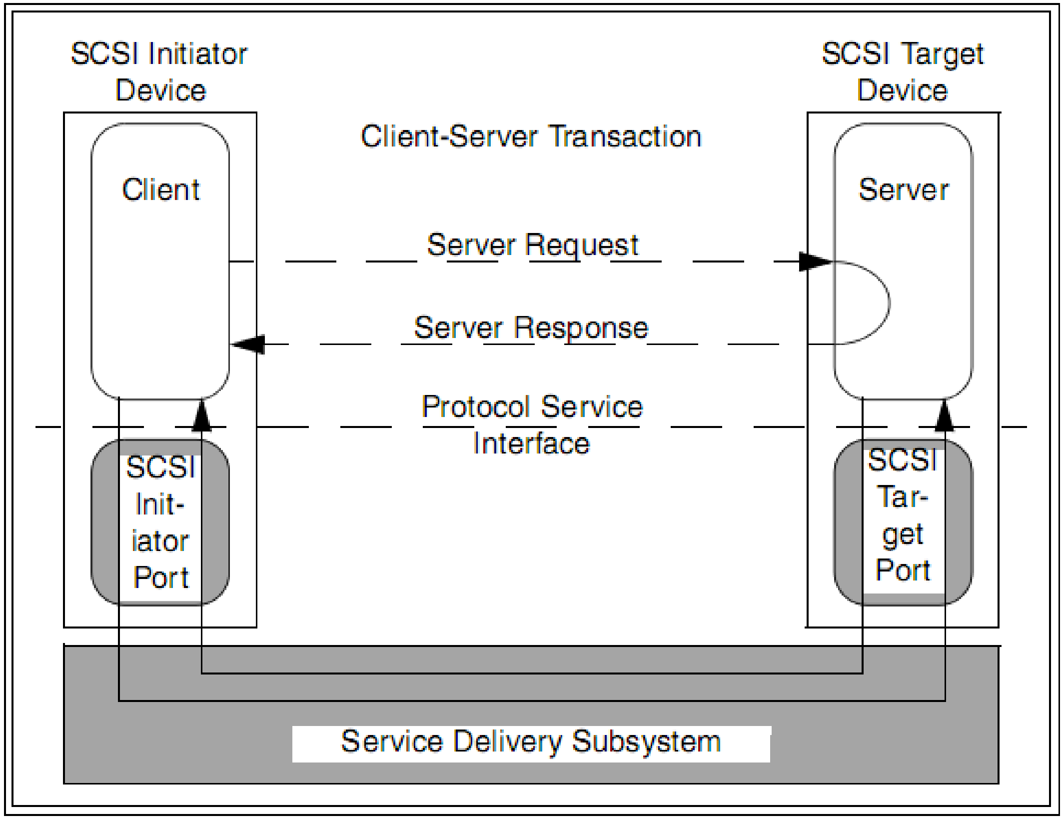


# SCSI standards family



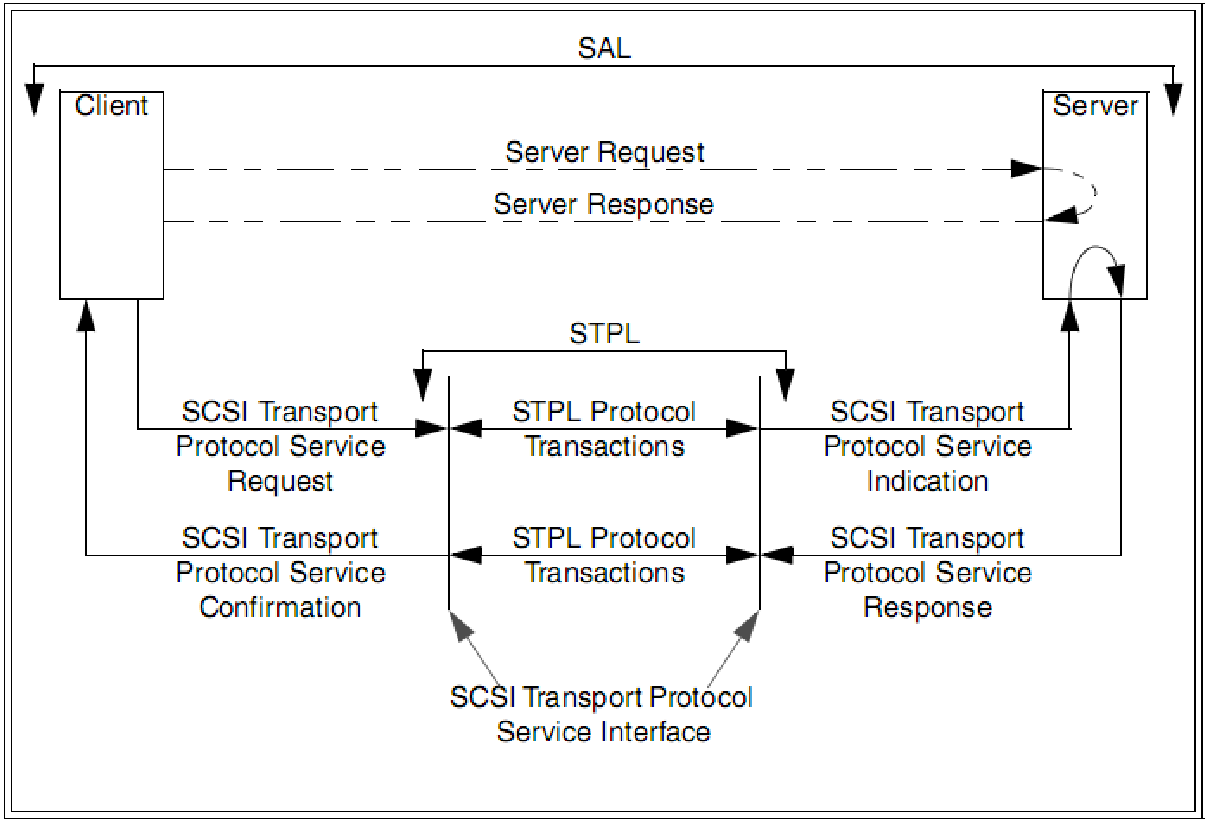

- SCSI-3 protocol stack model allows sending exactly the same commands over different buses or storage networking interconnects
  - Just transport-level wrapping of commands/answers, and link-level behavior, need to be changed
  - Allows also bidirectional payload transfer through two or more different transports between initiator and target

# SCSI client-server model



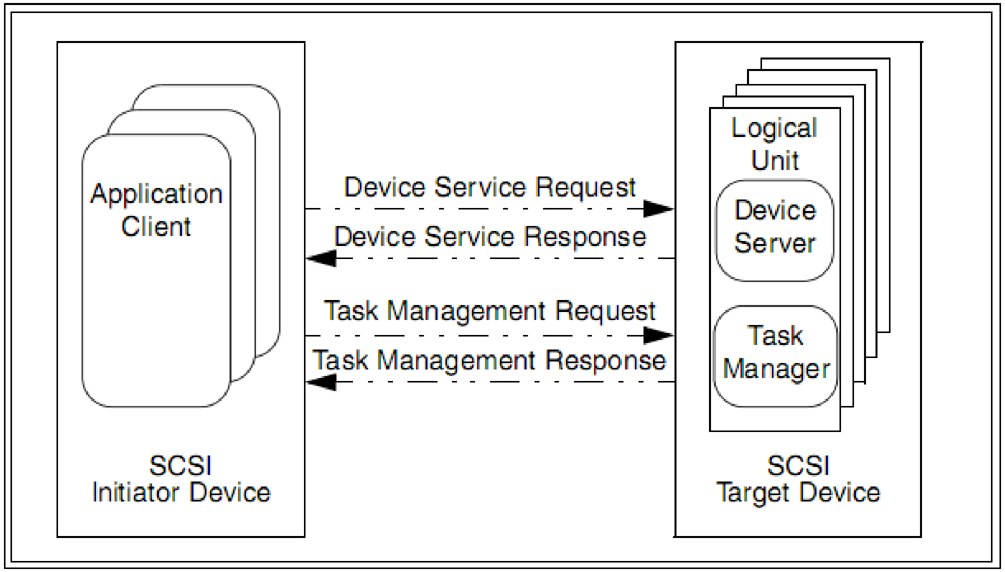

- SCSI I/O transaction between initiator and target is modelled as a client-server transaction
  - Server (= target = storage device) offers service to client (= initiator = HBA)
  - Service = access to write or retrieve data, get status or manage device

# SCSI client-server model




- SCSI command is request from service sent from client (initiator) to server (target)
- Transport and interconnect layer perform physical transport of request and response

# LUNs and tasks




- SCSI target (server) exposes services (functionality) through Logical Units
  - Logical Unit = object that processes commands (ex: read from media)
- Application client (driver in initiator) issues request (command) to Logical Unit
  - Command defines unit of work to be performed by Logical Unit
  - This unit of work is named *task* in SCSI parlance
  - Thus, initiator issues tasks (= commands) to Logical Unit, which processes them

# Command Descriptor Block (CDB)

**Typical CDB for 6-byte commands**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | OPERATION CODE | | | | | | | |
| 1 | Miscellaneous CDB information | | | (MSB) | | | | |
| 2 | LOGICAL BLOCK ADDRESS (if required) | | | | | | | |
| 3 | | | | | | | | (LSB) |
| 4 | TRANSFER LENGTH (if required)<br>PARAMETER LIST LENGTH (if required)<br>ALLOCATION LENGTH (if required) | | | | | | | |
| 5 | CONTROL | | | | | | | |

- Initiator sends command to Logical Unit issuing a CDB (*Command Descriptor Block*)
  - Structured set of bytes, typically of 6, 10, 12 or 16 bytes
  - Interpreted by target as request to execute a given command, and parameters to it

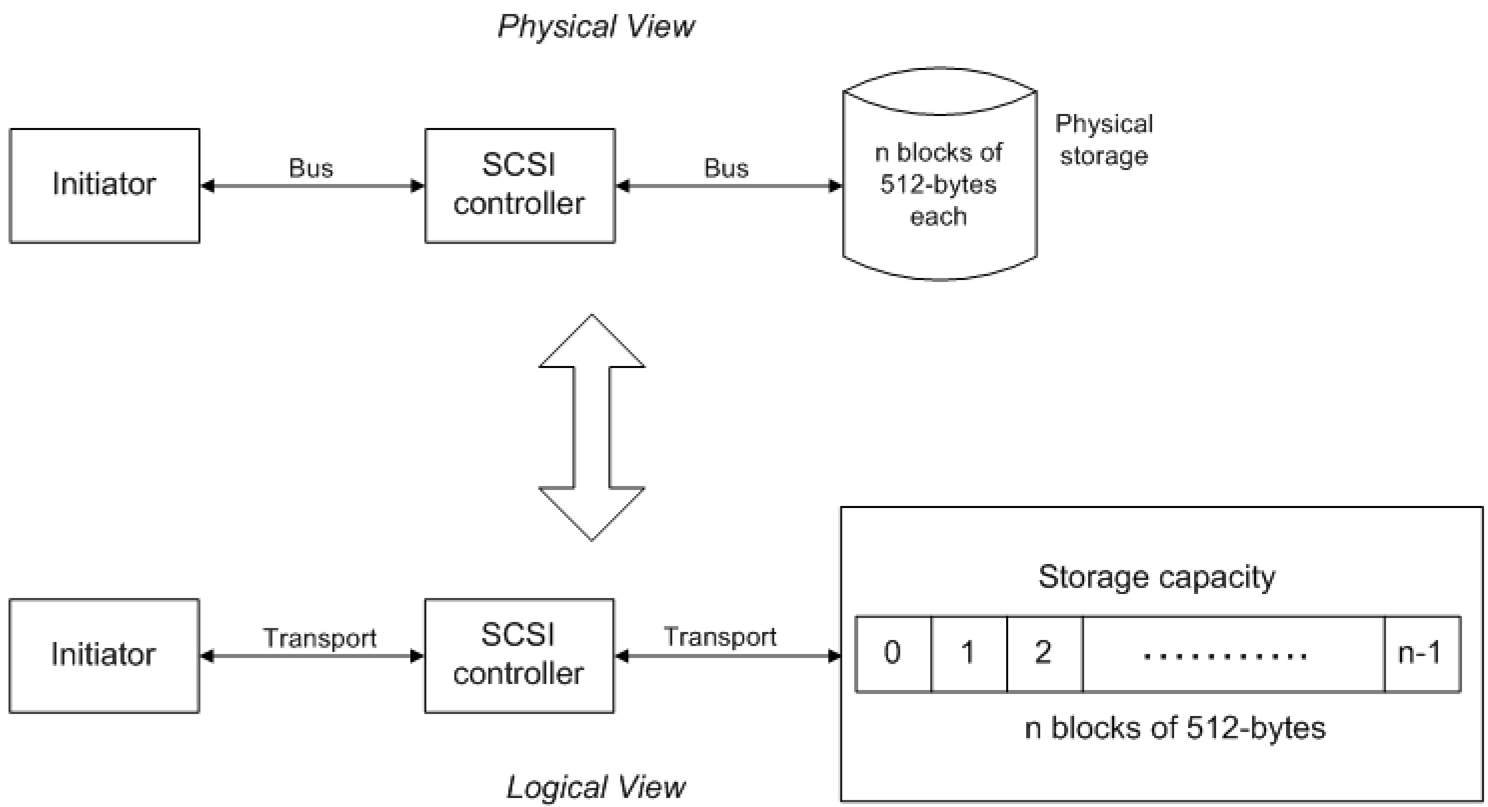
# Command Descriptor Block (CDB)

**Typical CDB for 10-byte commands**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | OPERATION CODE | | | | | | | |
| 1 | Miscellaneous CDB information | | | | SERVICE ACTION (if required) | | | |
| 2 | (MSB) | LOGICAL BLOCK ADDRESS (if required) | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | Miscellaneous CDB information | | | | | | | |
| 7 | (MSB) | TRANSFER LENGTH (if required)<br>PARAMETER LIST LENGTH (if required)<br>ALLOCATION LENGTH (if required) | | | | | | |
| 8 | | | | | | | | (LSB) |
| 9 | CONTROL | | | | | | | |

- Different size CDBs for same command differ in length of parameters
  - Larger address allows addressing larger storage volumes

# Logical Block Addressing




- SCSI controller hides physical details of storage organization
  - Storage capacity is shown just as a linear array of individually-addressable blocks

# Logical Block Addressing

**Typical CDB for 6-byte commands**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | OPERATION CODE | | | | | | | |
| 1 | Miscellaneous CDB information | | | (MSB) | | | | |
| 2 | LOGICAL BLOCK ADDRESS (if required) | | | | | | | |
| 3 | | | | | | | | (LSB) |
| 4 | TRANSFER LENGTH (if required)<br>PARAMETER LIST LENGTH (if required)<br>ALLOCATION LENGTH (if required) | | | | | | | |
| 5 | CONTROL | | | | | | | |

- Blocks to be accessed are addressed through 2 parameters in CDB:
  - Address of first logical block of transfer
  - Length, in blocks, of transfer
    - Typical block size = 512 bytes

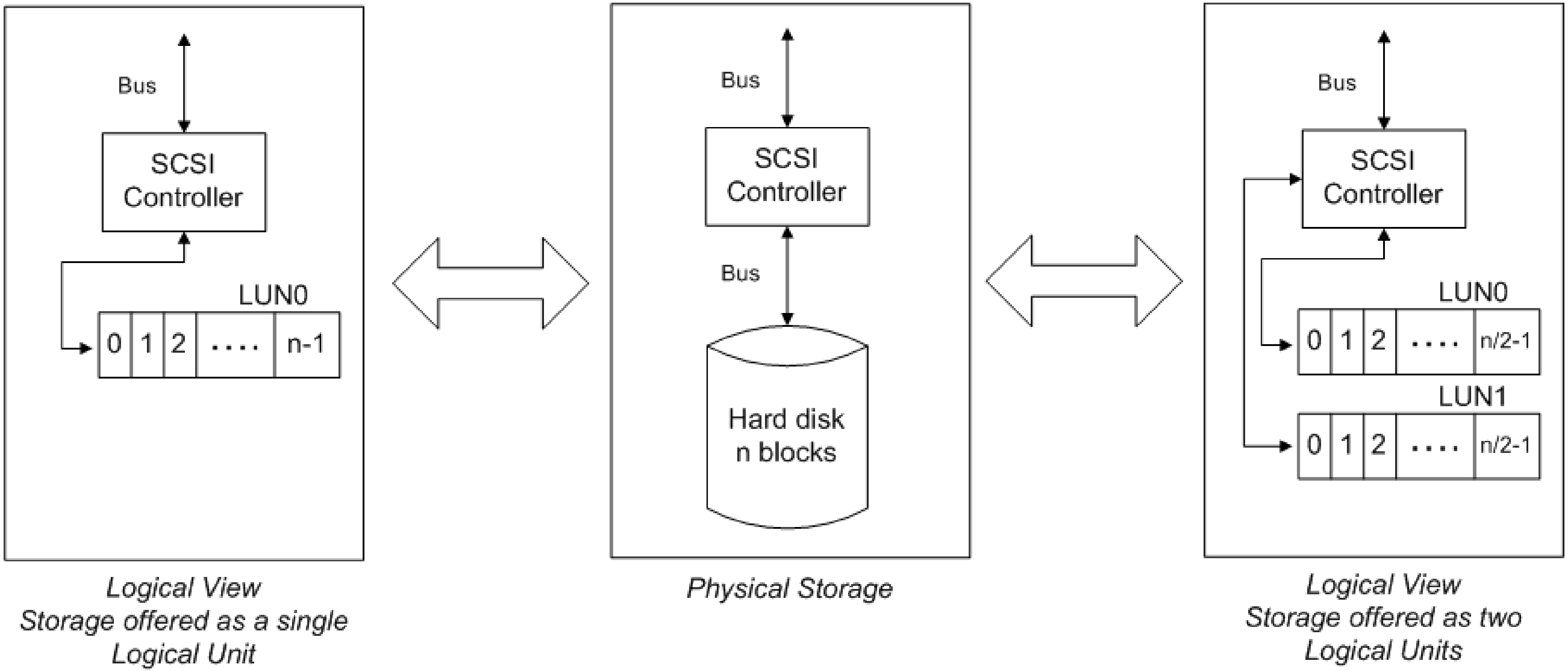
# Logical Block Addressing

Typical CDB for long LBA 16-byte commands

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | OPERATION CODE | | | | | | | |
| 1 | Miscellaneous CDB information | | | | | | | |
| 2 | (MSB) | LOGICAL BLOCK ADDRESS | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | | | | | | | | (LSB) |
| 10 | (MSB) | TRANSFER LENGTH (If required) PARAMETER LIST LENGTH (if required) ALLOCATION LENGTH (if required) | | | | | | |
| 11 | | | | | | | | |
| 12 | | | | | | | | |
| 13 | | | | | | | | (LSB) |
| 14 | Miscellaneous CDB information | | | | | | | |
| 15 | Control | | | | | | | |

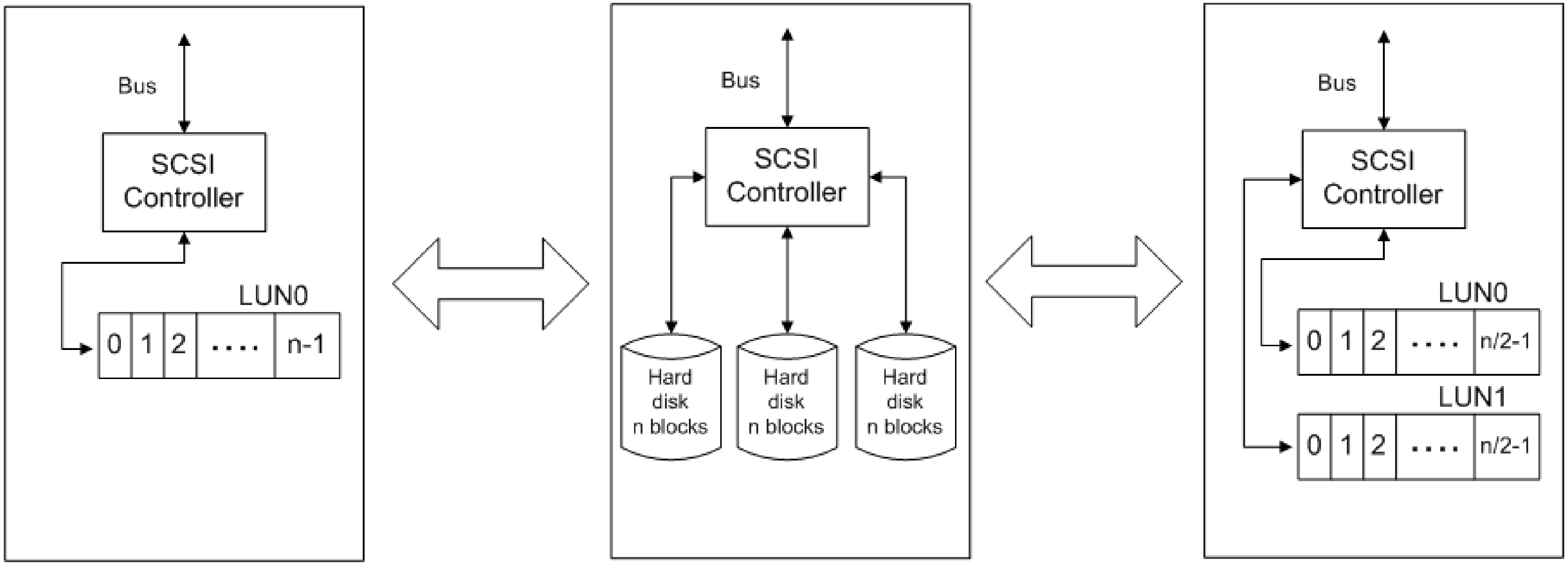
- Allowing larger address fields, SCSI interface copes with increasingly larger storage volumes:
  - CDB-6: $2^{21}$ blocks = 1 GByte
  - CDB-10: $2^{32}$ blocks = 2 TBytes
    - Typical LUN size limit for many SCSI controllers and virtualization software
  - CDB-16: $2^{24}$ blocks = 9 x $10^{18}$ bytes ~ 9 ExaBytes

# LUNs and LBA give flexibility




- Logical Units allow greater flexibility to organize storage
  - Controller just needs to map physical blocks to logical addresses
  - Required unique numerical identifier for each Logical Unit
    - **LUN** = *Logic Unit Number*

# LUNs and LBA hide complexity



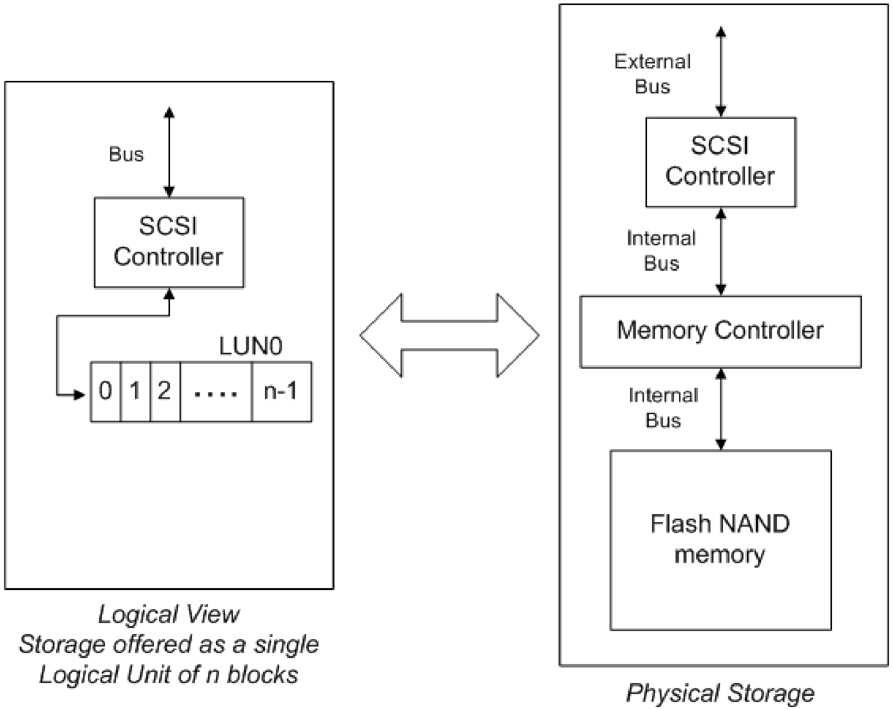

*Logical View*
*Storage offered as a single Logical Unit*

*Physical Storage*

*Logical View*
*Storage offered as two Logical Units*

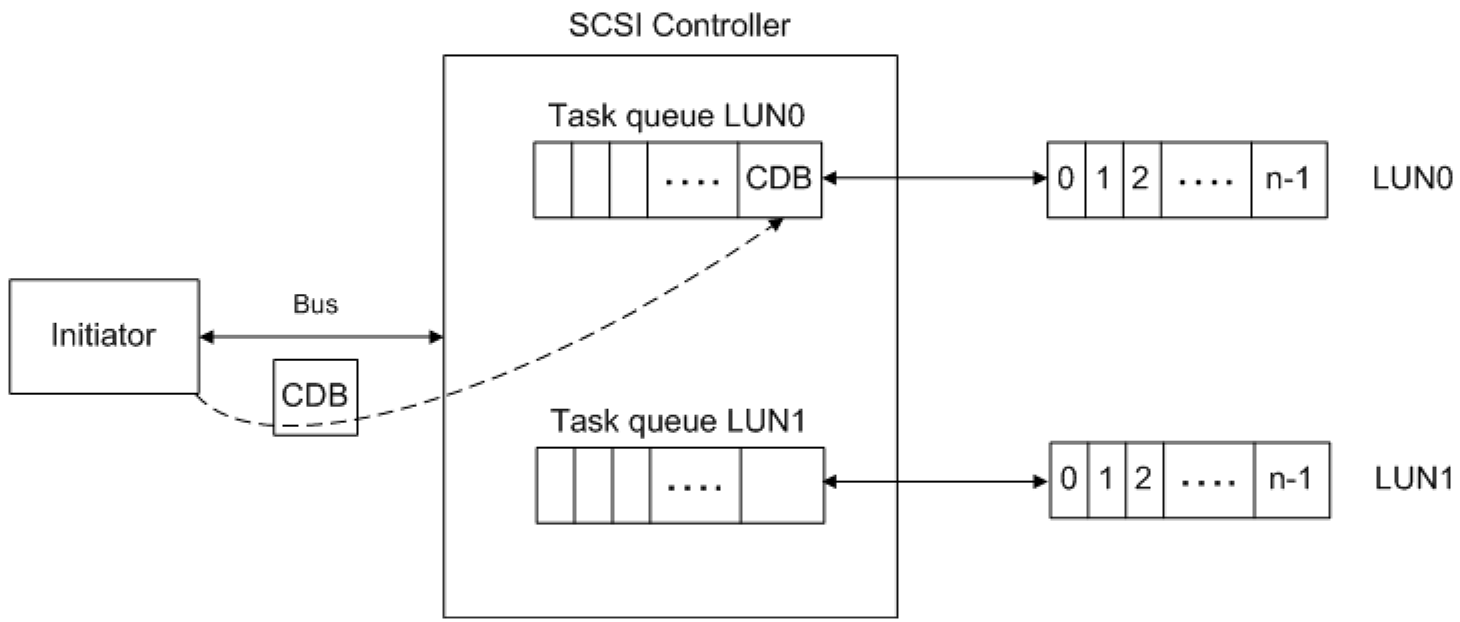
- Logical Units hide complexity of hardware setup
  - Left: controller shows several physical disks as single, larger disk
    - LBA hides physical placement of data
    - Controller free to transparently optimize placement
  - Right: controller sums total storage as split in several LUNs
    - Allow transparency of complex placement strategies (RAID, replication)

# LUNs and LBA hide complexity



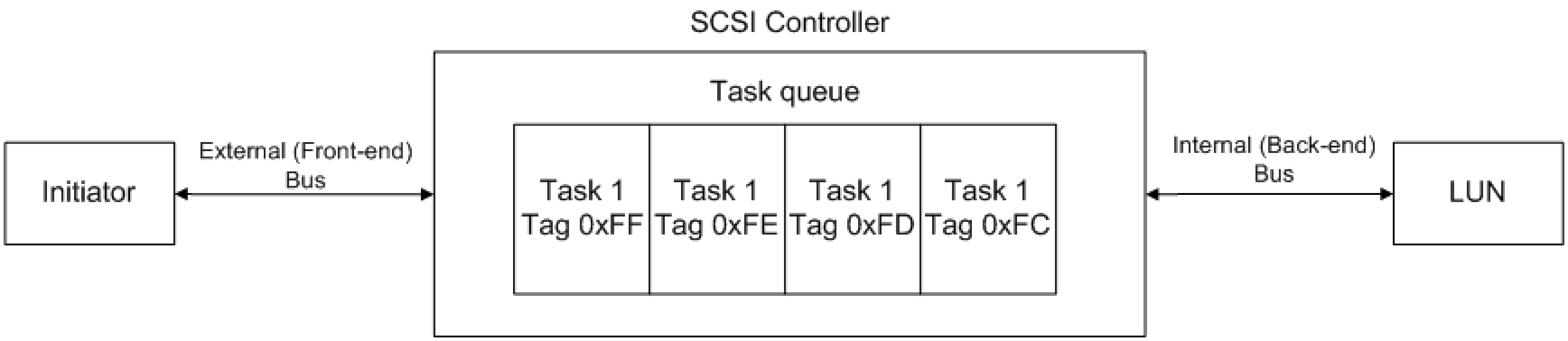

- Logical Units also hide nature of storage
  - Example: Flash storage (SSD, USB drives) has a complex internal architecture
    - Blocks are dynamically migrated for wear-leveling
    - Simple external view as a single LUN of n blocks
    - Memory controller manages mapping between logical LBA address and physical memory page

# LUNs and task queues



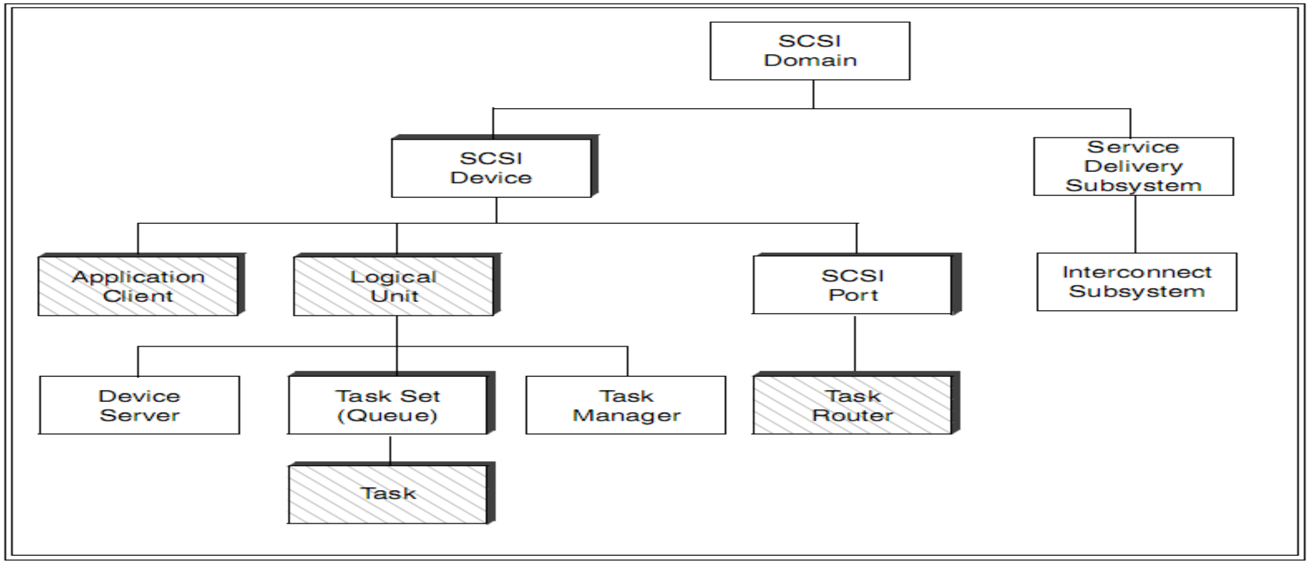

- SCSI controller can offer for each LUN none, one or several task queues to manage pending commands (*tasks*)
  - CDB inserted into queue, and can be executed asynchronously
  - If several commands in queue, can be reordered before execution, if hardware supports it
- If no queue, commands must execute using synchronous I/O (only a single command pending at any time)

# LUNs and task queues




- Use of task queue requires identifying each individual task
  - Each task is assigned numerical identifier, unique within queue
  - Tag provides context reference for task when executing
- Tags are not required in synchronous I/O ("*untagged commands*")

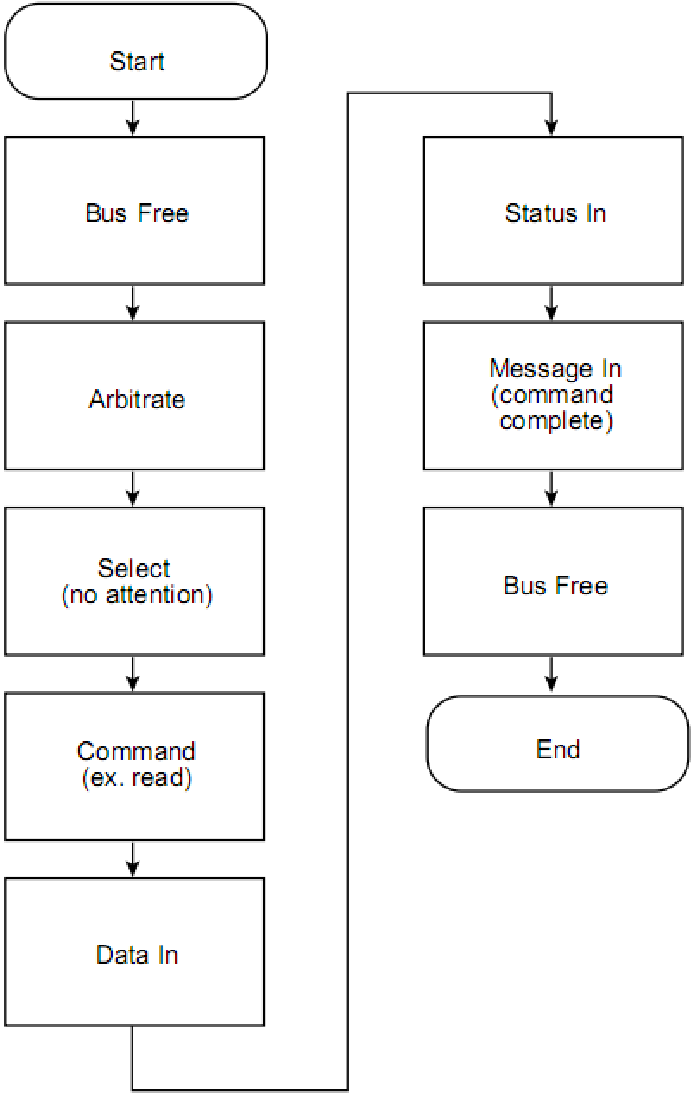
# SCSI task addressing




- Issuing or answering SCSI task requires use of several numerical identifiers:
  - Initiator ID and Target ID = Identifiers of physical devices involved
  - Logical Unit Number (LUN) = Identifier or logical unit, of target device, which must process this command
  - Task queue = Identifier of queue, within LU, in which command is inserted
  - Task tag = unique identifier for this task within queue

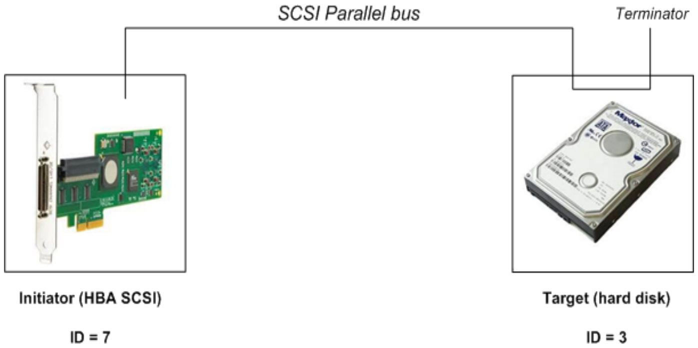
# Example: simple WRITE command on SPI

- Simplest way to understand operation of SCSI interface
- SCSI interface executes command as an ordered sequence of phases
  - Task will end with sending of status info on outcome of command



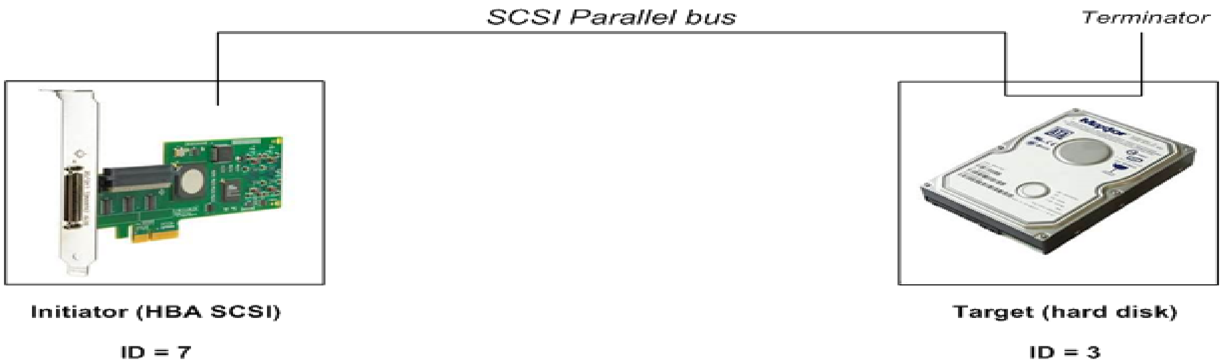
# Example: simple WRITE command on SPI

- Figure contains decoding of real traffic capture (with logic analyzer) for SCSI WRITE(10) command between initiator and target shown above
  - Write: data goes from initiator to disk




| Time | Time | STAT | DATA-8 |
|---|---|---|---|
| 4554214772 ns | 151.61 us | ARBITRATION | 80 |
| 4554215356 ns | 0.584 us | SELECT | 88 |
| 4554218924 ns | 3.568 us | MSG OUT | C0 |
| 4554220922 ns | 1.998 us | MSG OUT | 20 |
| 4554223122 ns | 2.2 us | MSG OUT | FE |
| 4554228472 ns | 5.35 us | COMMAND | 2A |
| 4554228896 ns | 0.424 us | COMMAND | 00 |
| 4554229172 ns | 0.276 us | COMMAND | 00 |
| 4554229448 ns | 0.276 us | COMMAND | 03 |
| 4554229722 ns | 0.274 us | COMMAND | 22 |
| 4554229998 ns | 0.276 us | COMMAND | FF |
| 4554230272 ns | 0.274 us | COMMAND | 00 |
| 4554230548 ns | 0.276 us | COMMAND | 00 |
| 4554230822 ns | 0.274 us | COMMAND | 80 |
| 4554231096 ns | 0.274 us | COMMAND | 00 |
| 4554236148 ns | 5.052 us | DATA OUT | 00 |
| 4554236248 ns | 0.1 us | DATA OUT | 00 |
| 4554236350 ns | 0.102 us | DATA OUT | 20 |
| 4560918946 ns | 6682.596 us | STATUS | 00 |
| 4560920940 ns | 1.994 us | MSG IN | 00 |

# Example: simple WRITE command on SPI




| | | | | |
|---|---|---|---|---|
| 4554214772 ns | 151.61 us | ARBITRATION | 80 | = 1000 0000 |
| 4554215356 ns | 0.584 us | SELECT | 88 | = 1000 1000 |

- SCSI transaction starts addressing physical devices involved
  - For parallel SCSI, addressing done in ARBITRATION stage
  - Parallel SCSI is multi-drop, so bus-master and bus-slave must be chosen
  - HBA (ID = 7) wins arbitration and becomes bus master
  - HBA selects then hard disk (ID = 3) as bus slave (peer for this transfer)

# Example: simple WRITE command on SPI

**IDENTIFY message format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | IDENTIFY | DISCPRIV | LUN | | | | | |

**Task attribute message codes**

| Code | Support | | | | Message name |
|---|---|---|---|---|---|
| | IU Transfers Disabled | | IU Transfers Enabled | | |
| | Iniator | Target | Initiator | Target | |
| 24h | O | O | N/A | N/A | ACA |
| 21h | Q | Q | N/A | N/A | Head of Queue |
| 22h | Q | Q | N/A | N/A | Ordered |
| 20h | Q | Q | N/A | N/A | Simple |

| | | | | |
|---|---|---|---|---|
| 4554218924 ns | 3.568 us | MSG OUT | C0 | LUN = 0 |
| 4554220922 ns | 1.998 us | MSG OUT | 20 | Simple queue |
| 4554223122 ns | 2.2 us | MSG OUT | FE | TAG |

- SCSI transaction needs now address LU, queue and command tag
  - Initiator addresses LUN0 of target device (hard disk)
  - Command is inserted into “Simple” queue
  - Command is uniquely identified within simple queue by tag 0xFE

# Example: simple WRITE command on SPI

**WRITE (10) command**

| Byte \ Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | OPERATION CODE (2Ah) | | | | | | | |
| 1 | WRPROTECT | | | DPO | FUA | Reserved | FUA_NV | Obsolete |
| 2 | (MSB) | LOGICAL BLOCK ADDRESS | | | | | | |
| 5 | | | | | | | | (LSB) |
| 6 | Reserved | | | GROUP NUMBER | | | | |
| 7 | (MSB) | TRANSFER LENGTH | | | | | | |
| 8 | | | | | | | | (LSB) |
| 9 | CONTROL | | | | | | | |

| | | | | |
|---|---|---|---|---|
| 4554228472 ns | 5.35 us | COMMAND | 2A | WRITE(10) codop |
| 4554228896 ns | 0.424 us | COMMAND | 00 | |
| 4554229172 ns | 0.276 us | COMMAND | 00 | LBA (MSB) |
| 4554229448 ns | 0.276 us | COMMAND | 03 | |
| 4554229722 ns | 0.274 us | COMMAND | 22 | |
| 4554229998 ns | 0.276 us | COMMAND | FF | LBA (LSB) |
| 4554230272 ns | 0.274 us | COMMAND | 00 | |
| 4554230548 ns | 0.276 us | COMMAND | 00 | Transfer Length (MSB) |
| 4554230822 ns | 0.274 us | COMMAND | 80 | Transfer Length (LSB) |
| 4554231096 ns | 0.274 us | COMMAND | 00 | |

- Initiator sends now CDB to target
  - Target interprets CDB and gets ready to write 0x80 blocks (64 KB) data in media, starting from specified LBA

# Example: simple WRITE command on SPI

**Status byte code bit values**

| Status byte | Status represented | Task Ended |
|---|---|---|
| **00h** | Good | Yes |

**Message format**

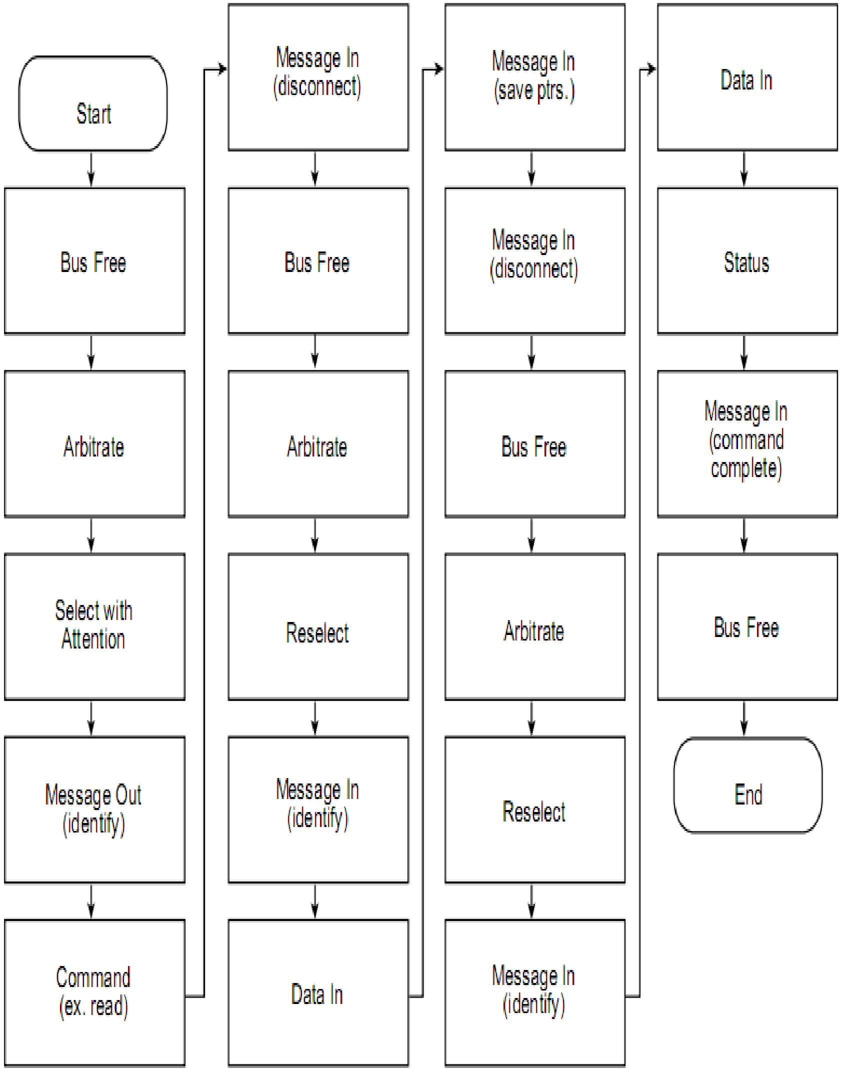
| Message code | Message format |
|---|---|
| **00h** | One-byte message (TASK COMPLETE) |

| | | | | |
|---|---|---|---|---|
| 4554236148 ns | 5.052 us | DATA OUT | 00 | Start of data transfer |
| 4554236248 ns | 0.1 us | DATA OUT | 00 | |
| 4554236350 ns | 0.102 us | DATA OUT | 20 | End of data transfer |
| 4560918946 ns | 6682.596 us | STATUS | 00 | Command ended OK |
| 4560920940 ns | 1.994 us | MSG IN | 00 | Task completed |

- Initiator sends data to target (disk)
- Target ends transaction by:
  - Reporting command executed OK (*Status Good)*
  - Declaring command as finished (*Task Complete)*

# Example: WRITE with split transaction

- SCSI interface allows split transactions
  - Target can disconnect after command, and reconnect to start data transfer
  - Data transfer itself can be interrupted and resumed, if needed
- Coupled with tagged queuing, provides high flexibility for optimizing command execution
  - Allows command reordering
  - Avoids wasting time idling the bus



# Example: WRITE with split transaction

| Time | Time | STAT | DATA-8 |
|---|---|---|---|
| 4547006302 ns | 0.584 us | SELECT | 88 |
| 4547009868 ns | 3.566 us | MSG OUT | C0 |
| 4547011968 ns | 2.1 us | MSG OUT | 20 |
| 4547014268 ns | 2.3 us | MSG OUT | FE |
| 4547019518 ns | 5.25 us | COMMAND | 2A |
| 4547019918 ns | 0.4 us | COMMAND | 00 |
| 4547020194 ns | 0.276 us | COMMAND | 00 |
| 4547020468 ns | 0.274 us | COMMAND | 03 |
| 4547020794 ns | 0.326 us | COMMAND | 22 |
| 4547021118 ns | 0.324 us | COMMAND | 7F |
| 4547021418 ns | 0.3 us | COMMAND | 00 |
| 4547021718 ns | 0.3 us | COMMAND | 00 |
| 4547021994 ns | 0.276 us | COMMAND | 80 |
| 4547022268 ns | 0.274 us | COMMAND | 00 |
| 4547024850 ns | 2.582 us | MSG IN | 04 |
| 4547339692 ns | 314.842 us | ARBITRATION | 08 |
| 4547340602 ns | 0.91 us | RESELECT | 88 |
| 4547342134 ns | 1.532 us | MSG IN | 80 |
| 4547356896 ns | 14.762 us | MSG IN | 20 |
| 4547358492 ns | 1.596 us | MSG IN | FE |
| 4547378766 ns | 20.274 us | DATA OUT | 00 |
| 4547378868 ns | 0.102 us | DATA OUT | 00 |
| 4547378968 ns | 0.1 us | DATA OUT | 00 |
| 4554061068 ns | 6682.1 us | STATUS | 00 |
| 4554063162 ns | 2.094 us | MSG IN | 00 |

**IDENTIFY message format**

| Bit / Byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 0 | IDENTIFY | DISCPRIV | LUN | | | | | |

- Figure shows single SCSI task (command), split in two I/O transactions
- Note that LUN addressing from initiator (MSG OUT C0) allows also DISCONNECT privilege
  - Essentially, tells target that initiator can store in RAM CDB, pointers and status info to resume this command, if needed

# Example: WRITE with split transaction

Link Control message codes

| Code | Support | | | | Message name |
|---|---|---|---|---|---|
| | IU transfers disabled | | IU transfers enabled | | |
| | Init | Targ | Init | Targ | |
| 04h | O | O | O | O | DISCONNECT |

| | | | | |
|---|---|---|---|---|
| 4547024850 ns | 2.582 us | MSG IN | 04 | |
| 4547339692 ns | 314.842 us | ARBITRATION | 08 | = 0000 1000 |
| 4547340602 ns | 0.91 us | RESELECT | 88 | = 1000 1000 |
| 4547342134 ns | 1.532 us | MSG IN | 80 | LUN = 0 |
| 4547356896 ns | 14.762 us | MSG IN | 20 | Simple queue |
| 4547358492 ns | 1.596 us | MSG IN | FE | TAG |
| 4547378766 ns | 20.274 us | DATA OUT | 00 | |

- After last byte of CDB, target disconnects from initiator
  - Task still active and pending
- When ready for transfer, target reconnects
  - Wins arbitration and becomes bus-master (!)
  - Re-selects initiator, which becomes bus-slave (!).
    - SPI functional equivalent to raising interrupt
  - Note that only <u>bus role</u> is reversed, NOT interface role

# Example: WRITE with split transaction

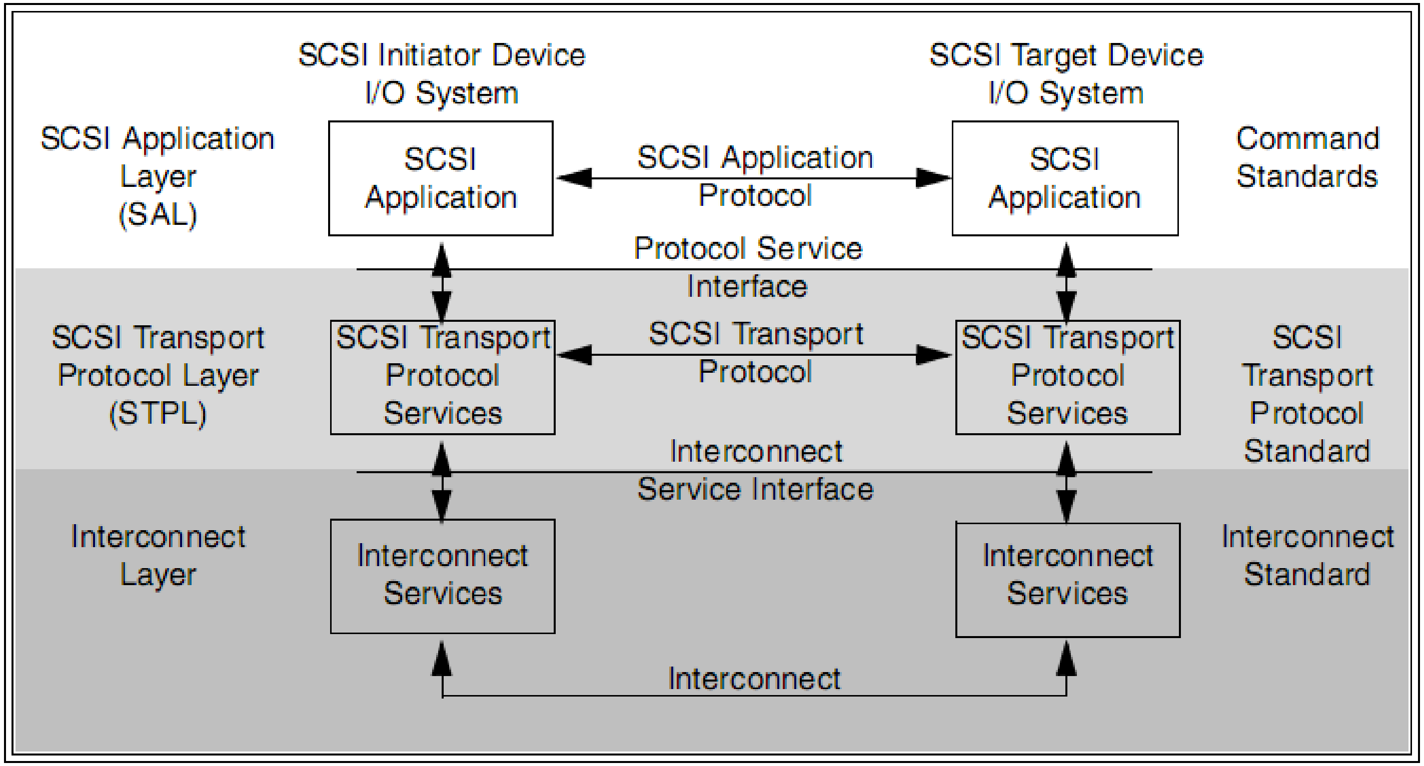
**Link Control message codes**

| Code | Support | | | | Message name |
|---|---|---|---|---|---|
| | IU transfers disabled | | IU transfers enabled | | |
| | Init | Targ | Init | Targ | |
| 04h | O | O | O | O | DISCONNECT |

| | | | | |
|---|---|---|---|---|
| 4547024850 ns | 2.582 us | MSG IN | 04 | |
| 4547339692 ns | 314.842 us | ARBITRATION | 08 | = 0000 1000 |
| 4547340602 ns | 0.91 us | RESELECT | 88 | = 1000 1000 |
| 4547342134 ns | 1.532 us | MSG IN | 80 | LUN = 0 |
| 4547356896 ns | 14.762 us | MSG IN | 20 | Simple queue |
| 4547358492 ns | 1.596 us | MSG IN | FE | TAG |
| 4547378766 ns | 20.274 us | DATA OUT | 00 | |

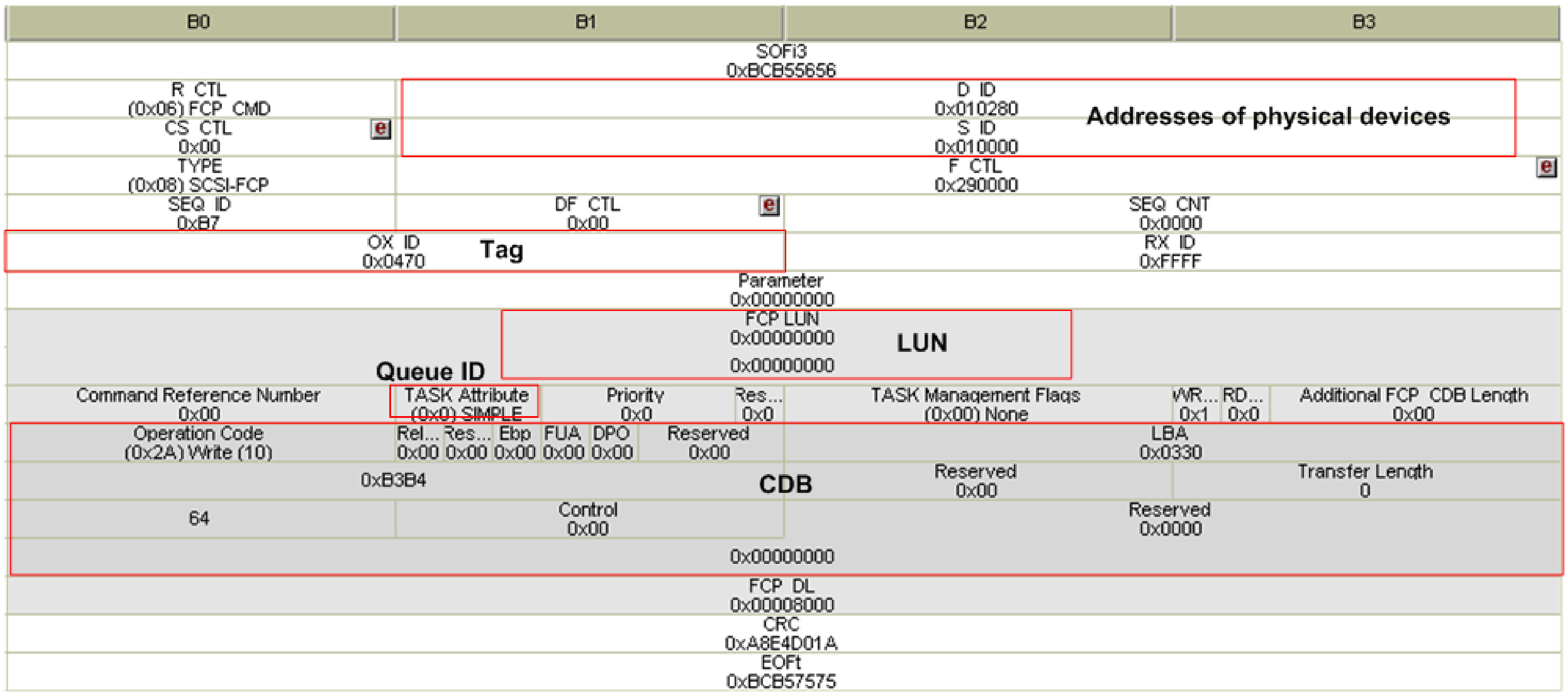
- Target sends now messages to identify task being resumed
  - Addresses LUN, queue and tag
- Disk accepts now data from initiator
- Command ends with Status Good (see end of complete transaction)

# Example: WRITE over other transports



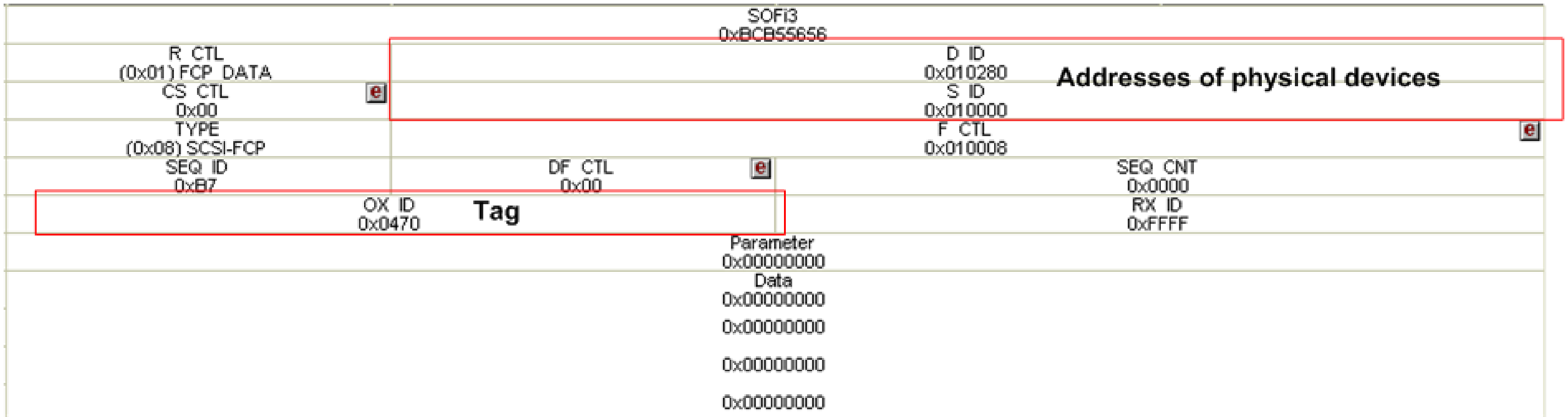

- Layered stack model means that SCSI protocol works the same way over any transport
  - Once understood over parallel SCSI (SPI), it is easy to understand its operation over any other bus

# Example: WRITE over Fibre Channel



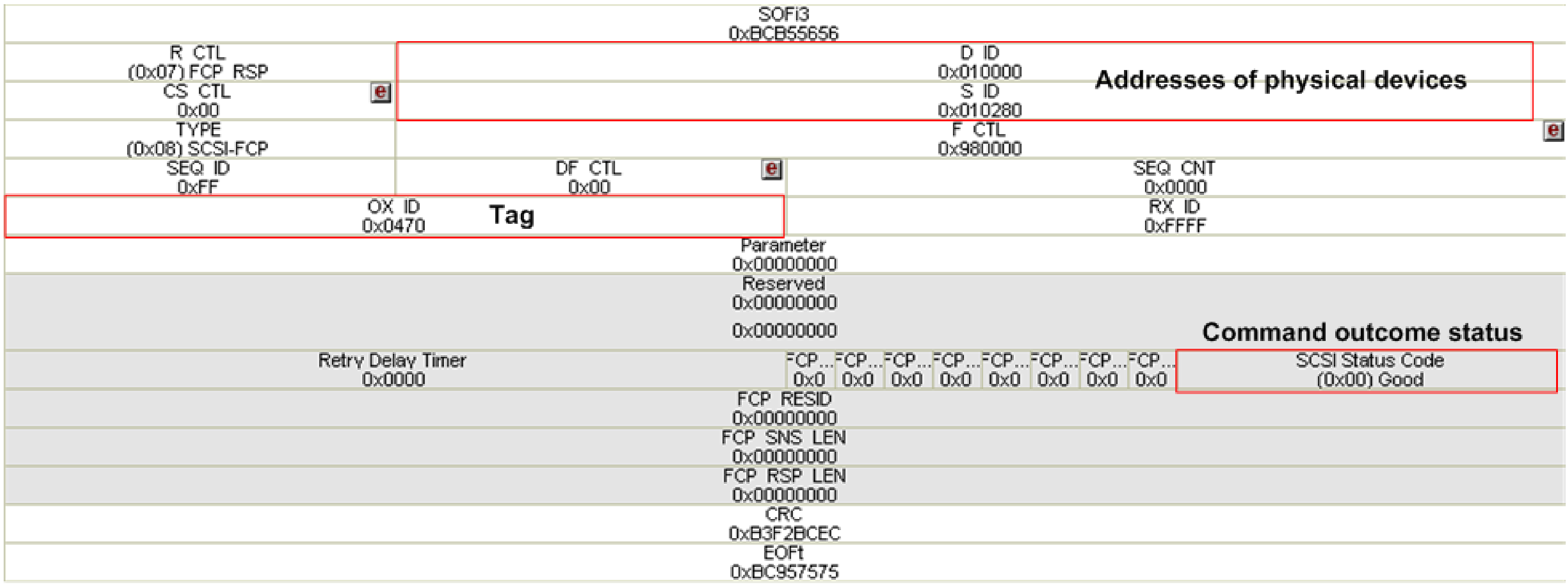

- Figure shows SCSI WRITE(10) command encapsulated within a Fibre Channel frame
  - SCSI protocol payload is straightforward to identify and interpret

# Example: WRITE over Fibre Channel



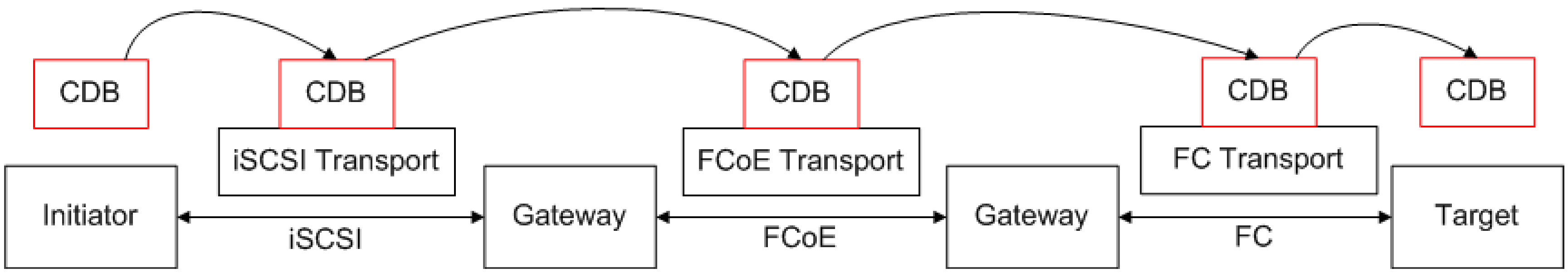

- Figure shows data transfer belonging to that same split transaction
  - Physical identifiers plus tag are enough to recover context of current task

# Example: WRITE over Fibre Channel




- Figure shows frame that completes SCSI protocol for WRITE command
    - Again, physical identifiers plus tag allow to recover context of current task
    - Status code is same as in parallel SCSI

# Summary: SCSI gives great flexibility




- SCSI protocol payload can be hopped from transport to transport with no changes
  - Encapsulation/De-encapsulation is straightforward and reasonably fast
- SCSI allows transaction between initiator and target while traversing several different buses/storage networking infrastructures
  - Ideal for accessing real storage from virtual machines, or for virtualizing storage

# What’s next?

- We’ve seen that most I/O transactions can today be described as the exchange of SCSI payload over various transport layers

- Now we will see how servers and storage devices can be interconnected in storage-specific networks, over which these SCSI exchanges are performed, and how these networks allow for redundant and/or high-performance storage configurations