Block Devices and Transport Classes: Where are we Going?

James Bottomley

SteelEye Technology

21 July 2005

Introduction: The Motivation

- Promises made at the Kernel Summit (2002):
  - Slim Down The SCSI Layer
  - Move Functionality to the Block Layer
  - Residue should be small helper libraries for things that are SCSI specific and not appropriate to be moved up to block.

Initial Efforts to Fulfil Promises

- Tag Command Queueing moved up to the block layer
  - Unfortunately, very few drivers using it.
- Queueing being pulled out of drivers in favour of the block layer queues.
- All separate queueing moved from the SCSI mid-layer.
- However, this has been a long, slow business: three years later still not finished.
- The helper library died an almost instantaneous death.

Some History of SCSI

- Subsystem used to undergo a complete rewrite for every stable kernel version (2.0→2.2→2.4)
- Stopped this (in spite of tremendous external pressure) for 2.6
- However instead began a program of gradual modifications.
- Significant problem though is the number of users of the SCSI layer
  - 2.0 and 2.2 were essentially parallel SCSI only (still shows in the error handler).
  - 2.4 began to acquire things like USB and Fibre Channel.

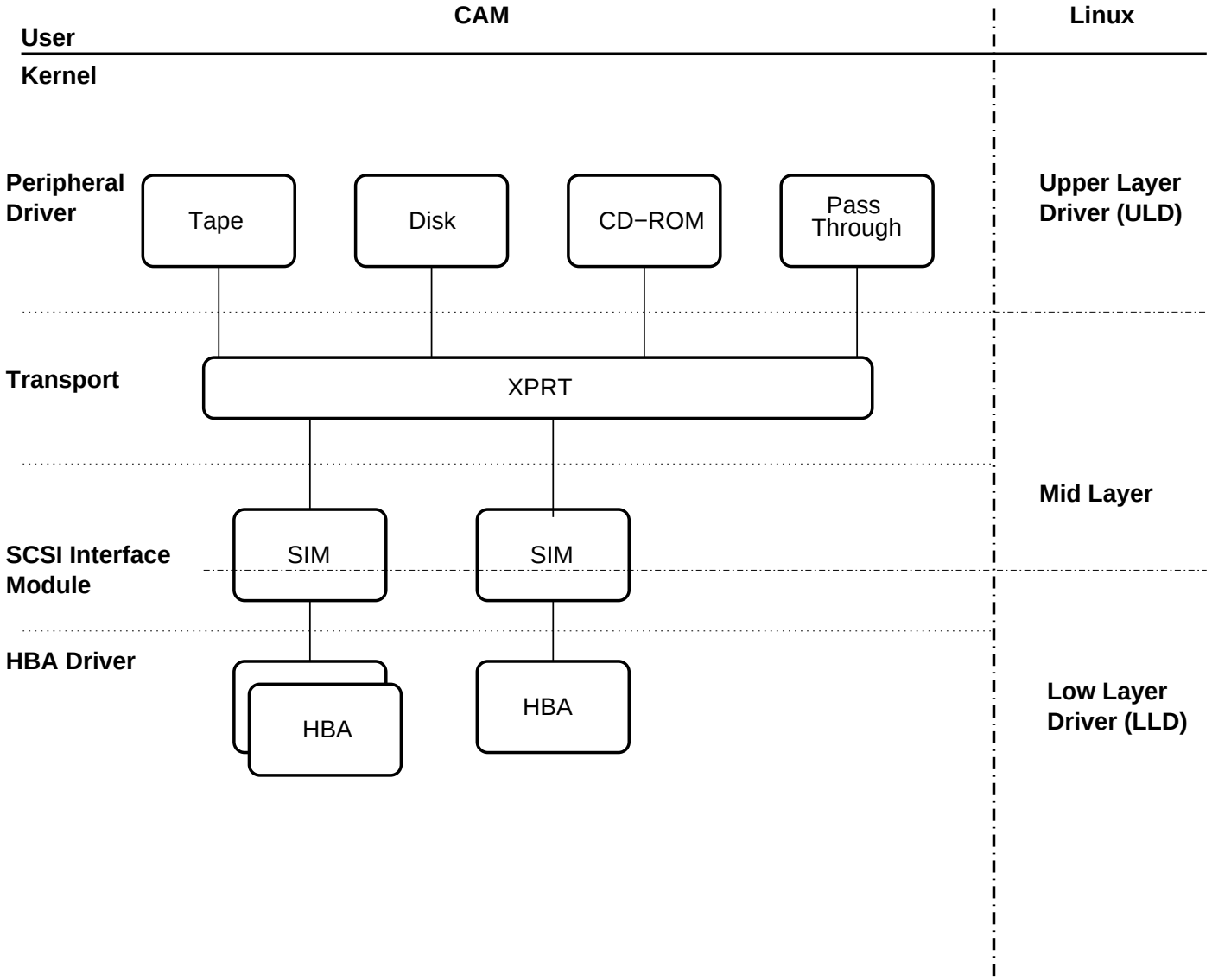
Where SCSI is Officially Going

- SCSI is under the control of the T10 SCSI standards Committee.
- SCSI-2 was a completely monolithic standard designed for parallel SCSI.
- SCSI-3 was a complete rewrite in terms of an architecture model that tries to get away from this.
  - Central architecture model
  - Device specific command models.
  - Transport specific standards for feeding commands and messages to devices.

What about CAM?

- CAM (Common Access Model) designed to provide uniform access to all SCSI devices
- Originally a T10 standard for SCSI-2.
- Attempt for SCSI-3 (CAM-2) was withdrawn.
- Adopted by BSD for the underpinning of its SCSI devices
- Some agreement at the Miami Storage Conference (2001) to adopt this for Linux too.
- So would this be a good idea?
  - It does fit with the helper library idea
  - but how well does it work out in practise?

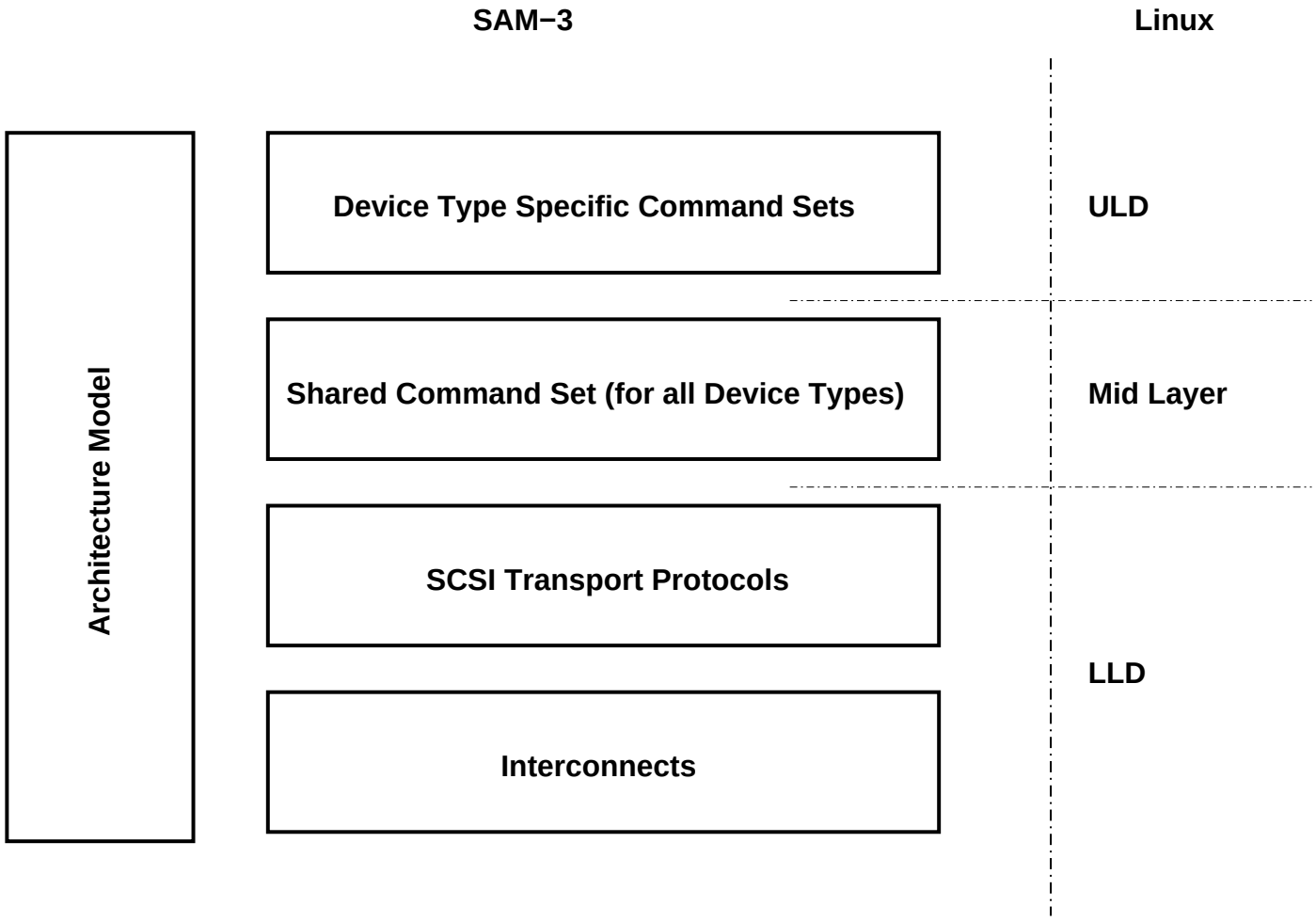
A View of CAM


CAM
Linux
User
Kernel
Peripheral Driver
Tape
Disk
CD−ROM
Pass Through
Upper Layer Driver (ULD)
Transport
XPRT
Mid Layer
SCSI Interface Module
SIM
SIM
HBA Driver
HBA
HBA
Low Layer Driver (LLD)

The Problems Begin

- CAM is a four layer model.
  - The linux kernel is three.
- The Middle layers of CAM map to the Linux Mid layer (and effectively the block layer).
  - All the transport specific pieces of CAM are in this middle layer.
  - The transports are the most variable aspect of SCSI and what cause the most difficulty from a block point of view.

The SAM Model



SAM is better, but not Perfect

- It maps nicely to the current SCSI stack
- But, the transport is buried in the LLD.
  - drivers do this already,
  - but it's **wrong**!
  - Transport code should be held in common.

Enter sysfs

- SCSI embraced sysfs early on (actually as a consequence of embracing the device model).
- Placed `struct device` inside `struct scsi_device` and began exporting parameters.
- Now have 13+ parameters.
- Also expanded the model to allow drivers to add extra parameters (both per device and per HBA).

sysfs and classes

- For full details, see Greg Kroah-Hartman
- A class represents and interface to a device
- you can have many classes per single device (i.e. anything that contains a `struct device`).
- each class can have parameters exported via sysfs

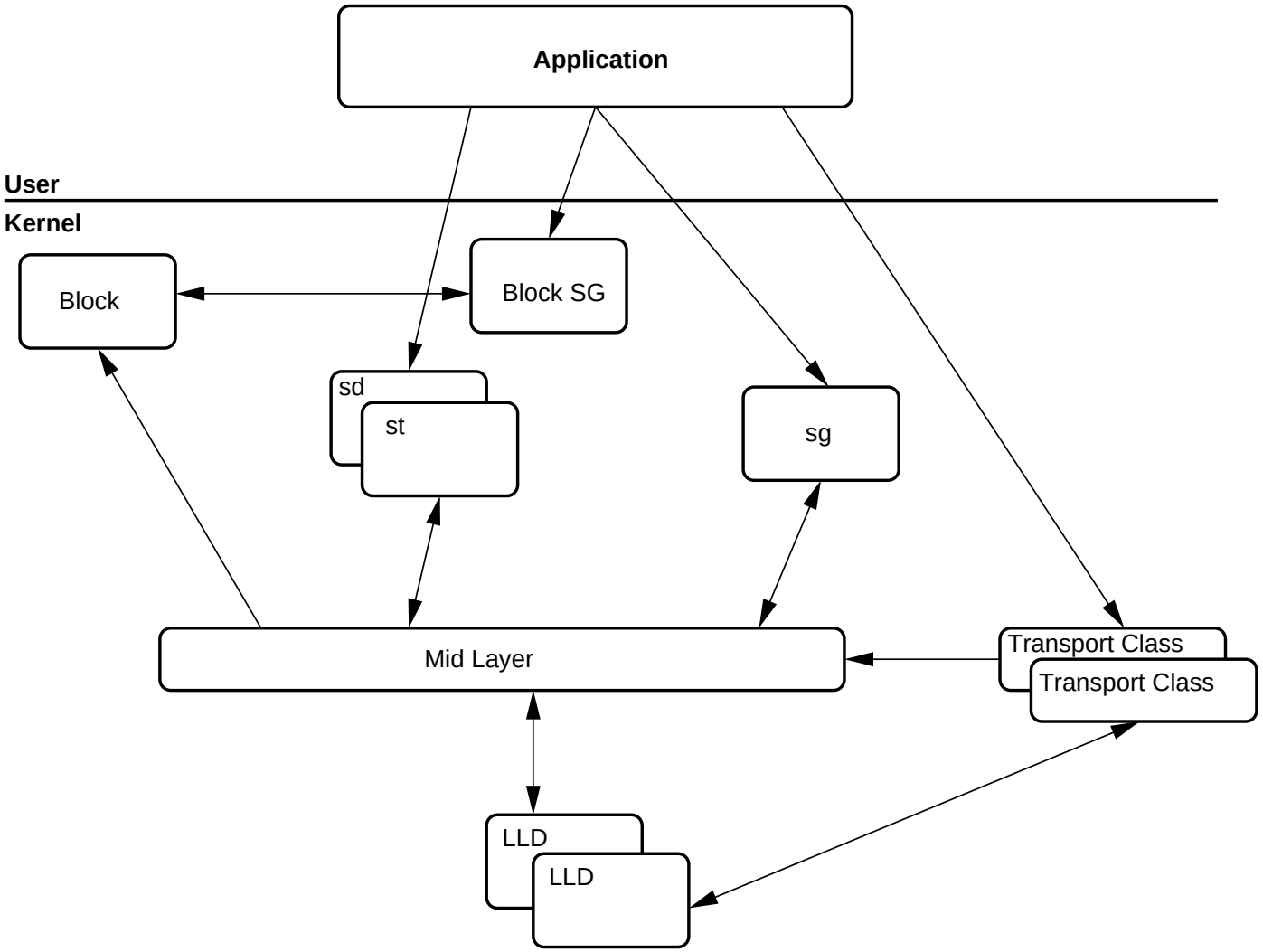
What do these "Libraries" look like?

- The original idea was to have a library (like a function call) that provided services to the card
- This is good for some things (like message display, or performing a common function).
- However, lots of things (e.g. queue depth, bus speed) that the user wants to influence
- This implies that we need mechanism for setting and reading values.
- Best thing about a library is that it centralizes common code, leading to a reduction in duplication (and in number of bugs).

Enter the SCSI Transport Class

- The helper library, married to a sysfs class becomes a SCSI transport class.
- Now you have the library functions you're looking for with a uniform mechanism for getting and setting values from user space.
- since sysfs is all files, the values can be simply manipulated without any special tools at all.
- This is perfect: it means the SCSI API, as it interacts with userspace, is now infinitely extensible.
- Parameters are exported per HBA, per target and per LU.

Anatomy of SCSI with Transport Classes



Implementing SCSI Transport Classes

- Began life as simple two parameter exports for SCSI and FC.
- First out of the gate with more complete functionality was the SCSI Parallel Interface (SPI).
- Major annoyance is how to get and set bus speed parameters (like u160 or u320).
  - How fast is my SCSI bus is a major issue for a lot of users
- On the kernel side, Domain Validation (DV) is a requirement (theoretically) and every driver implements its own.
- Putting DV in the transport class and switching the Adaptec driver to use it saved 2,000 lines of code.

The Fibre Channel Transport Class

- Developed primarily by Emulex
  - Effectively as a condition for acceptance of their `lpfc` driver.
- Began like the SPI transport class, exporting parameters and simple services.
- ended up (through rports) altering fundamentally the layout of the SCSI tree for fibre devices.
- This is good: it gives both flexibility and control in code sharing.
- Qlogic estimates that switching to the FC transport class reduced their driver by 30%

If it's Successful Use it Everywhere

- Originally the Transport Classes were extremely SCSI specific
  - As expected since they're designed to minimize the SCSI layer
  - However, every consumer of the block layer could be eligible to make use of this feature.
- Additionally, a single scsi device (target or host) could only have one transport class.
- So look at abstracting this.

Container Abstractions

- Begin by examining the real problem
  - Each class device needs an entry in some overall structure to which it belongs
  - this limits the number of classes to be the number of entries you have.
  - This is intuitively obvious, but in the SCSI case, we might not know this a priori.
- Need a way of hanging a class interface off a device without allocating space in the structure for it.
- This became the generic attribute container.

Generic Attribute Containers

- Introduce a wrapper (container) around a `struct class_device`
  - Allocate one wrapper per classdevice
  - So classdevice is now allocated when it comes into use rather than having to be present already.
  - Wrapper also contains back pointer to the attribute container and a list head.
- attribute containers consist of the class, a pointer to a set of attributes and a match callback to identify devices you're interested in.
- trigger points must be coded into the functions using the container.

Generic Transport Class

- Built directly on the attribute container.
- Really, identical, except instead of arbitrary triggers it defines five:
  1. **setup**: device created.
  2. **add**: device made visible
  3. **configure**: device ready to begin operating at full capacity (cf scsi `slave_configure()`).
  4. **remove**: device made invisible.
  5. **destroy**: last put called, free all resources.

OK, so Where Are We Going?

- The generic transport class is effectively freed from SCSI, so it should pick up several non-scsi uses
  - SAS/SATA: these share a PHY interface, so we could get a singly PHY class plus either a SAS or SATA class.
  - IDE (if they want).
  - Hardware RAID
  - ...
- However, doesn't stop there.

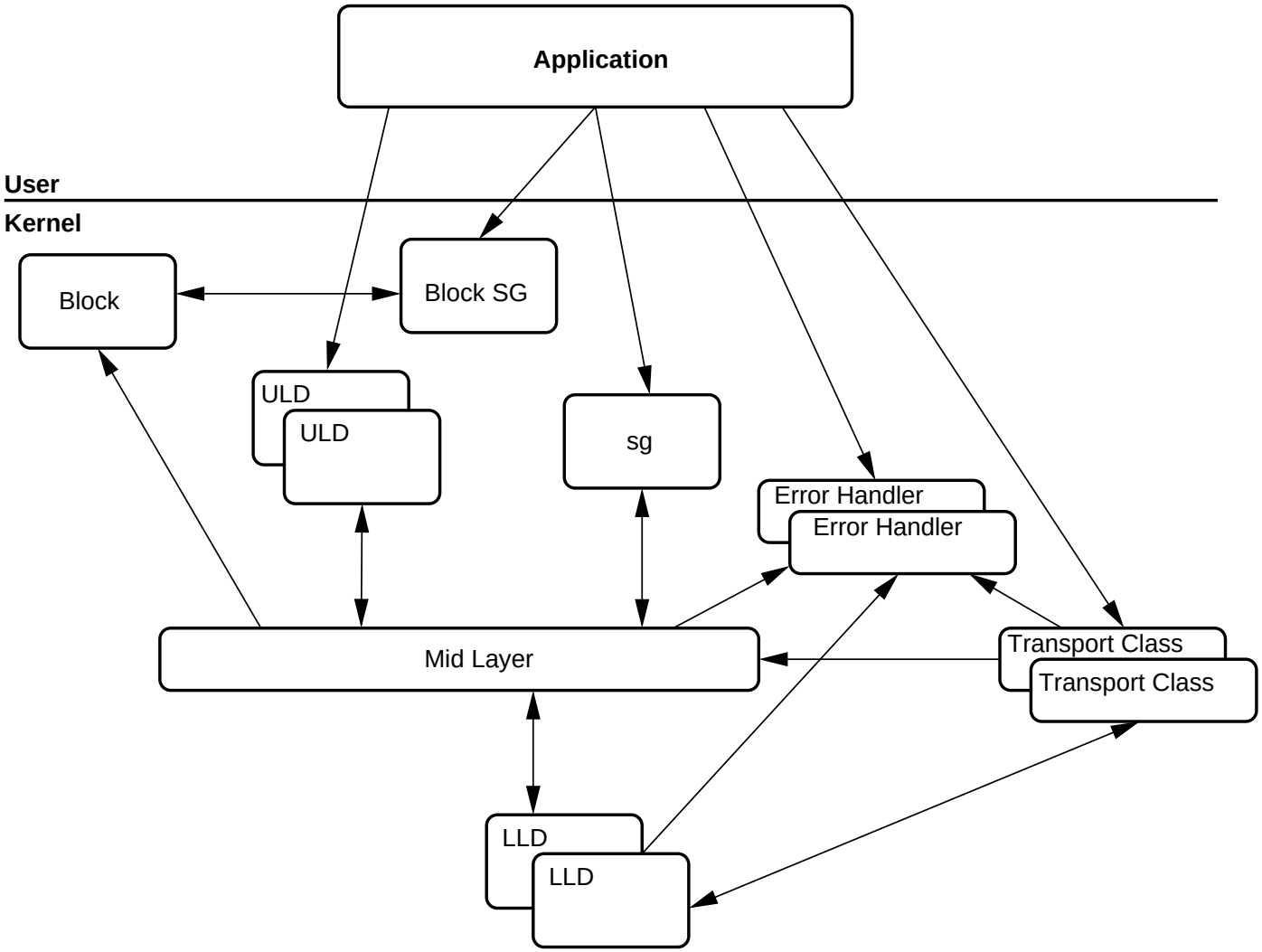
Other Transport Class Uses

- A transport class is essentially an interface.
- It could be regarded as a driver in its own right.
- It can attach to absolutely anything that has a generic device.
- It has its own overrideable match function to determine device attachment.
- Thus it could be used even in non transport cases
- Interesting one from this morning is PCI-E.

What Should We Be Doing in SCSI?

- Most obvious component crying out for reform is Error Handling.
- Single biggest complaint everyone still has with SCSI.
- Current error handler is still largely rooted in SCSI-2 (Parallel SCSI).
- Almost completely inappropriate to modern transports
- Thing most prone to error is the transport
- So move error handling into the transport classes.

Error Handler in Transport Classes



Conclusions

- Transport classes take us in the right direction
  - Smaller drivers
  - Common features in Common code
  - hopefully less bug prone
- As an abstraction (driver backed interface code) they may be far more generically useful around the kernel.