A New Tape Driver Architecture

Curtis Anderson

canderson@TurboLinux.com

Voice: 650-244-7777

FAX: 650-244-7766

What Do Tape Drivers Do?

- Important characteristics of a tape driver
  - Provide access to a drive by an application
  - Insulate app from some quirks of the drive
  - Homogenize operational semantics across drive types
  - Define failure domains and failure modes

What Do Tape Drivers Look Like?

- Traditional tape driver architecture
  - Inside the kernel so it can access the hardware
  - Event driven state machine handling interrupts
  - Normal code threads plus interrupt threads
  - MP locking in normal and interrupt threads
  - Process scheduler interference
  - Delicate interfaces and bad debugging tools

*What **Should** a Tape Driver Do?*

- Requirements for a driver architecture
  - Failures must be contained
  - Same operational semantics from all drive types
  - Portable to multiple O/S platforms for consistency
  - Distributed and parallel development efforts
  - High performance
  - Isolate support of a device from other devices
  - Differing levels of investment for each drive type

What About a Traditional Approach?

- Monolithic source file drivers
  - Support for all devices is mingled in a single driver
  - Common semantics across devices is easy
  - Kernel code: no failure containment, not portable, skilled implementers required
  - Difficulty separating code for one device from another
  - Regression test all devices after any change is painful

What About a Traditional Approach?

- Separate source file drivers
  - Each device is segregated into a separate kernel driver
  - Good at separating code for one device from another
  - Development/maintenance for devices done in parallel
  - Common semantics across devices is much harder
  - Kernel code: no failure containment, not portable, skilled implementers required

Need Standard Tape Access Semantics!

- Document tape operating semantics
  - Driver writers need to know what to make the drive do
  - Apps need to know what to expect out of the drive
  - Tests to check for correct semantics/regressions
- IEEE P1563: Tape Driver Semantics Std.
  - Standard tape access semantics for apps running on UNIX and Linux systems

*What **Should** a Tape Driver Look Like?*

- Platform-dependent: Tape Support Driver
  - A data pump that is O/S specific and not drive specific
  - Extremely minimal error processing code
- Device-dependent: “Personality” Daemons
  - Drive specific error handling and recovery
  - Make native drive behave like the defined semantics
- IEEE P1563: Tape Driver Recom. Practice

What is a “Tape Support Driver”?

- Data pump, only does read/write operations
  - All other ops/exceptions sent to Personality daemon
  - Some “errors” need to be handled inline
- Platform and O/S specific
  - Written once per platform, common to all drive types
  - Uses all platform and O/S performance features
  - Written for I/O performance, errors handled elsewhere

What Interfaces Does a TSD Need?

- /dev/tape device node interface for apps
  - App must not be required to use a “new” API
  - Platform independent interface (IEEE 1563) will make semantics and operations common to all platforms
- Interface to a running Personality Daemon
  - Platform independent user-level interface (see paper for full details)

What is a "Personality Daemon"

- Defines semantics, not on the data path
  - Handles all exceptions and all ops except read/write
- Drive type specific
  - Written once per drive, common to all platforms
  - Uses all drive management and error recovery features
  - Written for error handling and conformance to desired semantics, I/O performance handled elsewhere

What Does a Personality Do?

- Processes control commands from the App
  - Intercepts and implements all non-read/write ops
  - Builds its own SCSI commands to control the drive
- Processes device exceptions
  - Unsuccessful device ops get passed to the Personality
  - Interacts with drive to diagnose/recover/log the error
  - Decides what error code (if any) to give the application

Is a Personality More Capable?

- Much better fault isolation than kernel code
  - Only a single drive impacted, not the entire system
  - Personality Daemons are restartable user processes
  - Assumes that there are no hard performance requirements on control/error handling code paths
- Individual admin. of each physical drive
  - Dynamically add/enable/disable support per drive

How Do You Program a Personality?

- One running daemon per physical drive
  - Simple event-reaction loop inside the Personality
  - No multi-threading or locking to worry about
  - Much simpler development and testing environment
  - Portable, user-level code
- Multiple Personalities developed in parallel
  - Best done by the drive vendor, but anyone can do it

How is a Device Exception Processed?

- App is blocked while Personality working
- Personality interacts with the drive
  - Does error characterization and recovery actions
  - Runs device-dependent diagnostic procedures
  - TapeALERT information is handled and logged
  - Log errors, status, diagnostics, and recovery actions
  - Interact with IEEE 1244 MMS, SYSLOG, etc

How Does a Personality Talk to a TSD?

- Fully synchronous calls into the kernel
  - See the paper for full details
- ioctl() function calls provide access to
  - App control requests (eg: rewind, set file-mark)
  - Device exceptions (eg: HBA status info, sense codes)
  - Device statistical info for management apps

How Does a Personality Talk to a TSD?

- ioctl() function calls provide control of
  - Direct SCSI command blocks sent to the drive
  - Device statistical info for management apps
  - Error codes to be returned to the app (if any)

How Do You Test a Personality?

- Tape Support Driver able to inject errors
  - Error recovery code in Personalities needs testing
  - Software layer at the bottom of the TSD can return an error for a command instead of success
  - Not required for initial development of the architecture

What is the Executive Summary?

- Personalities and Tape Support Drivers
  - Failures are contained to a restartable user process
  - Common semantics across drive depends on testing
  - Personalities are portable, and TSD's are common
  - Development/maintenance for devices done in parallel
  - I/O performance is very good
  - Good at separating code for one device from another
  - Differing levels of investment for each drive type

Structure at System Boot

User Mode

UNIX/Linux

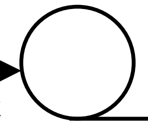
S/W

H/W

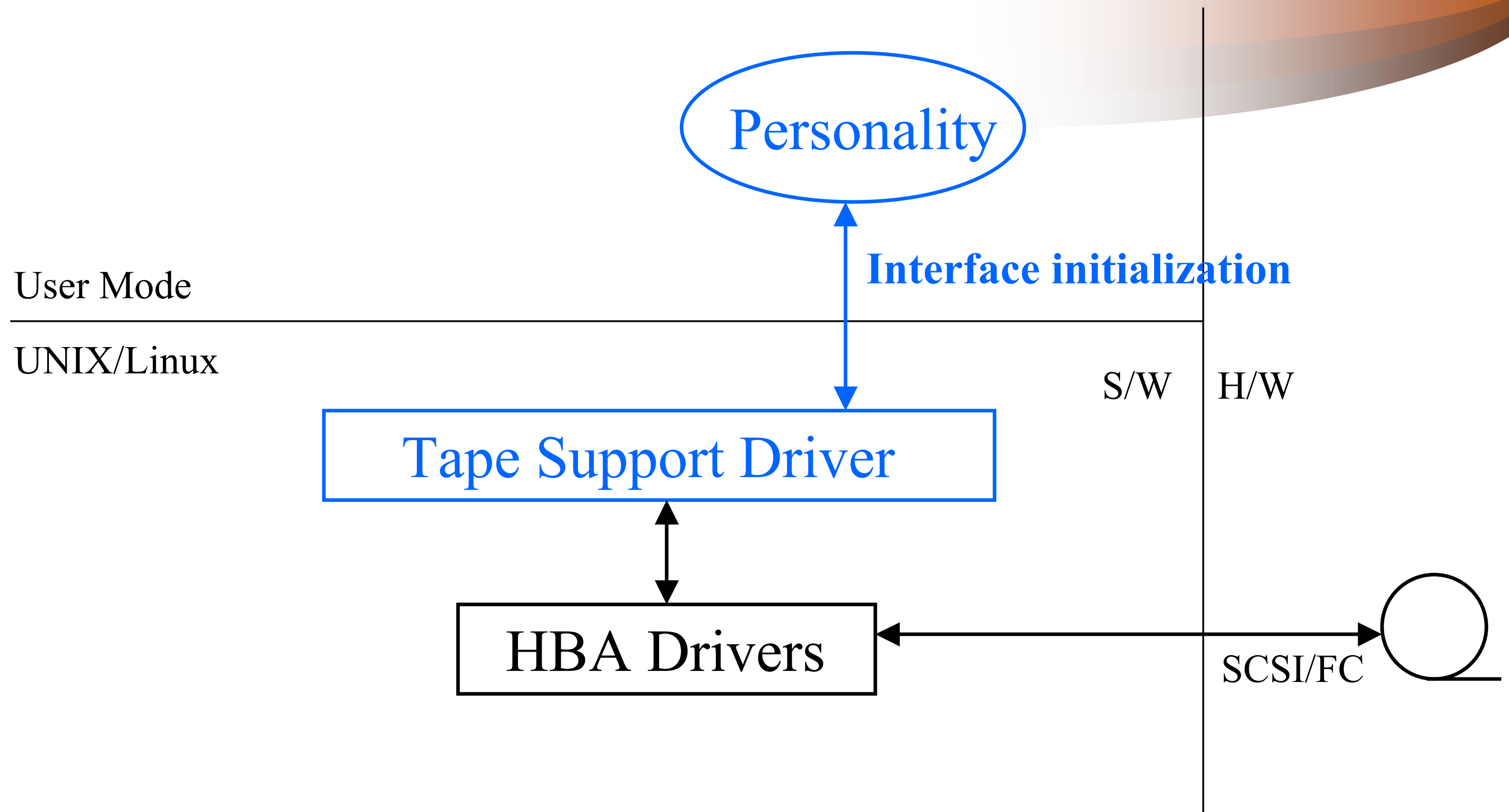
Tape Support Driver

HBA Drivers

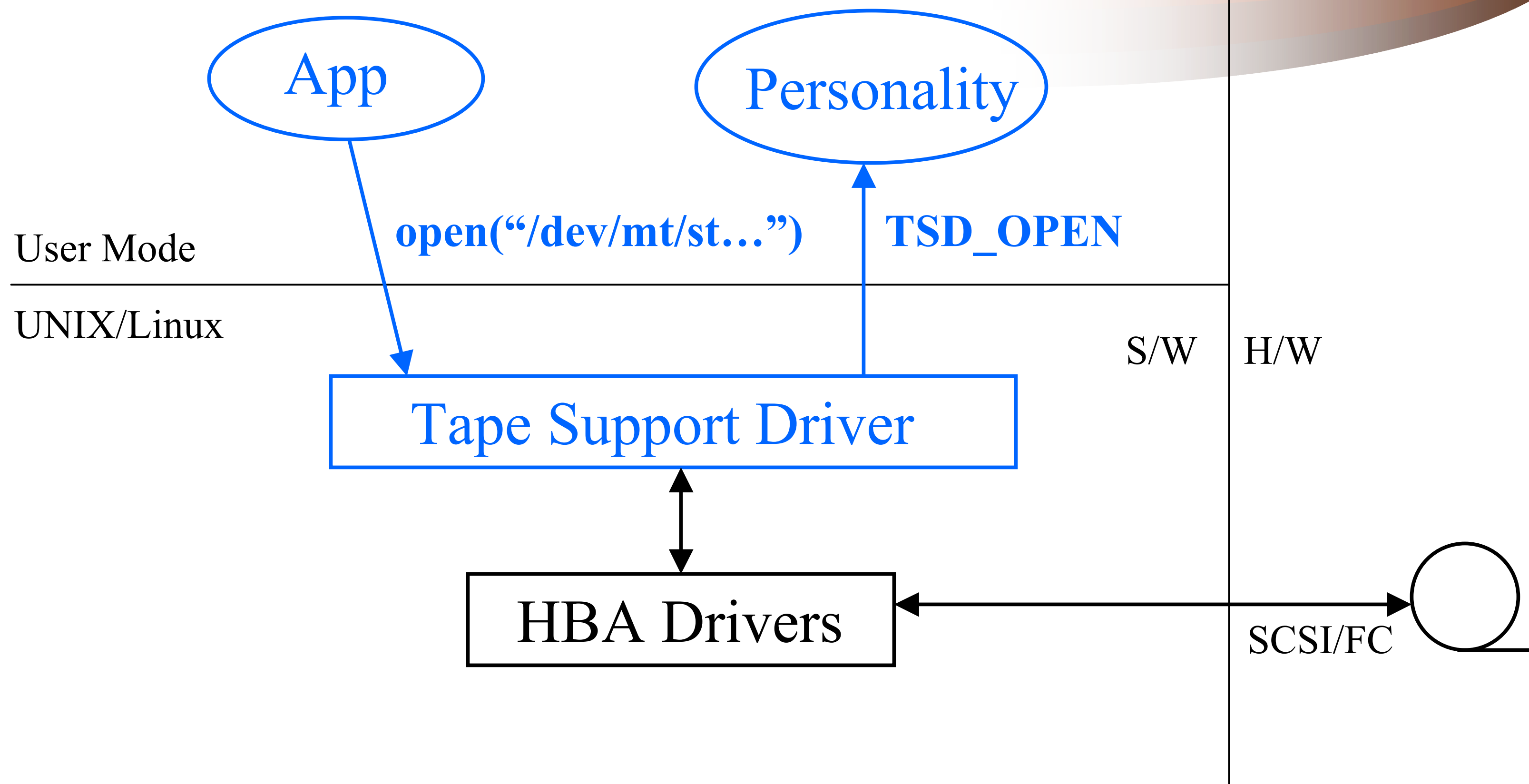
SCSI/FC



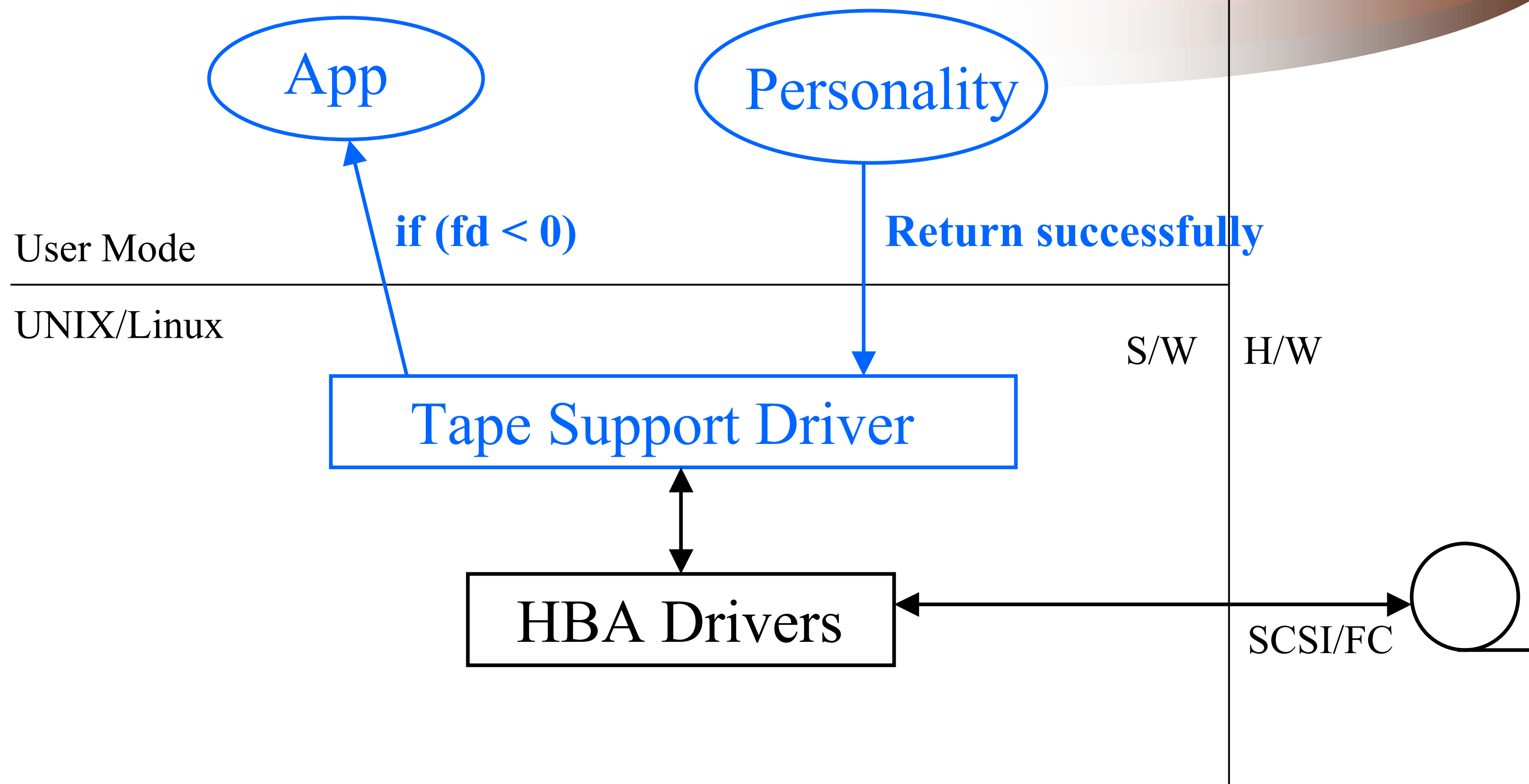
Personality Started by "init"



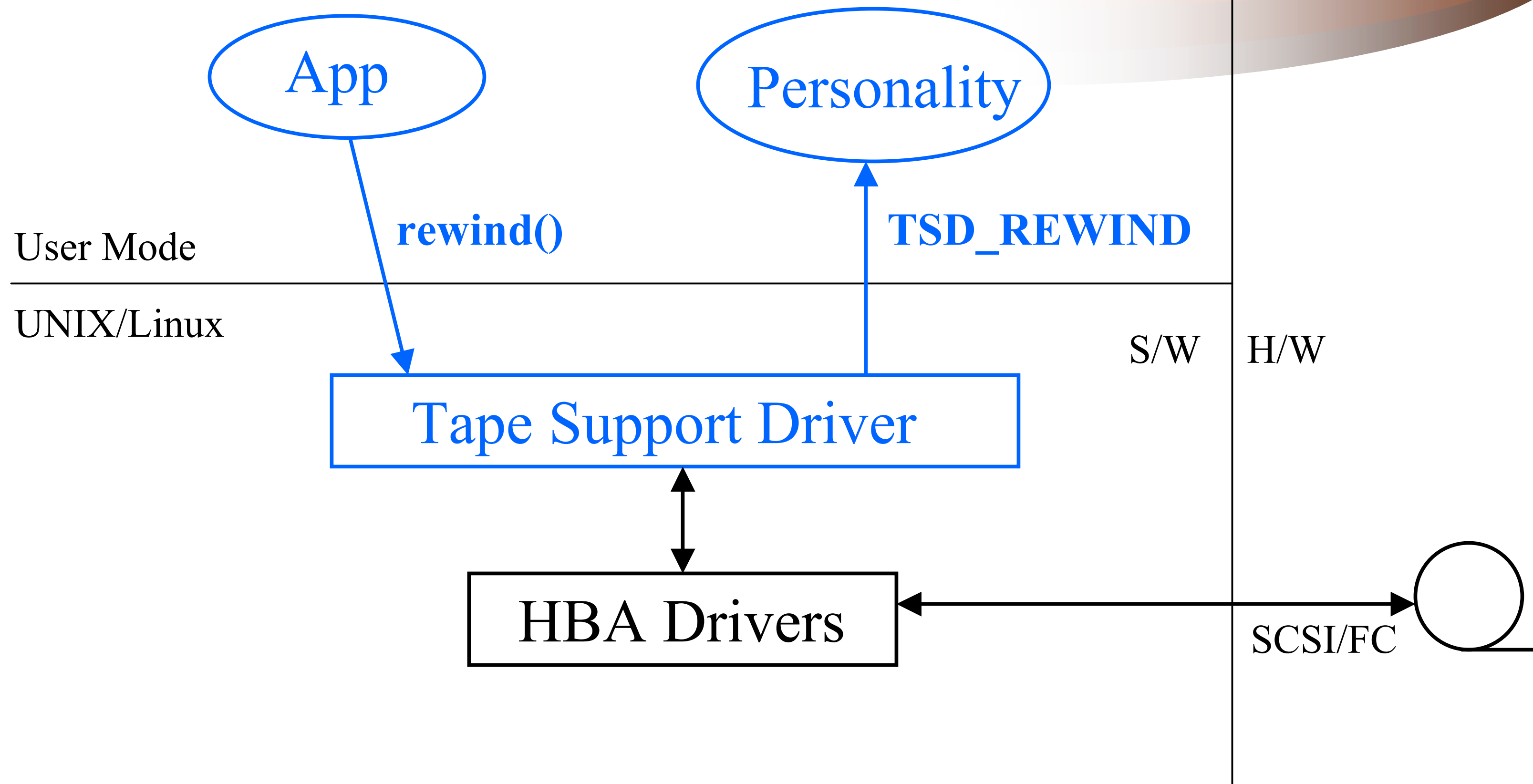
App Starts and Opens Drive



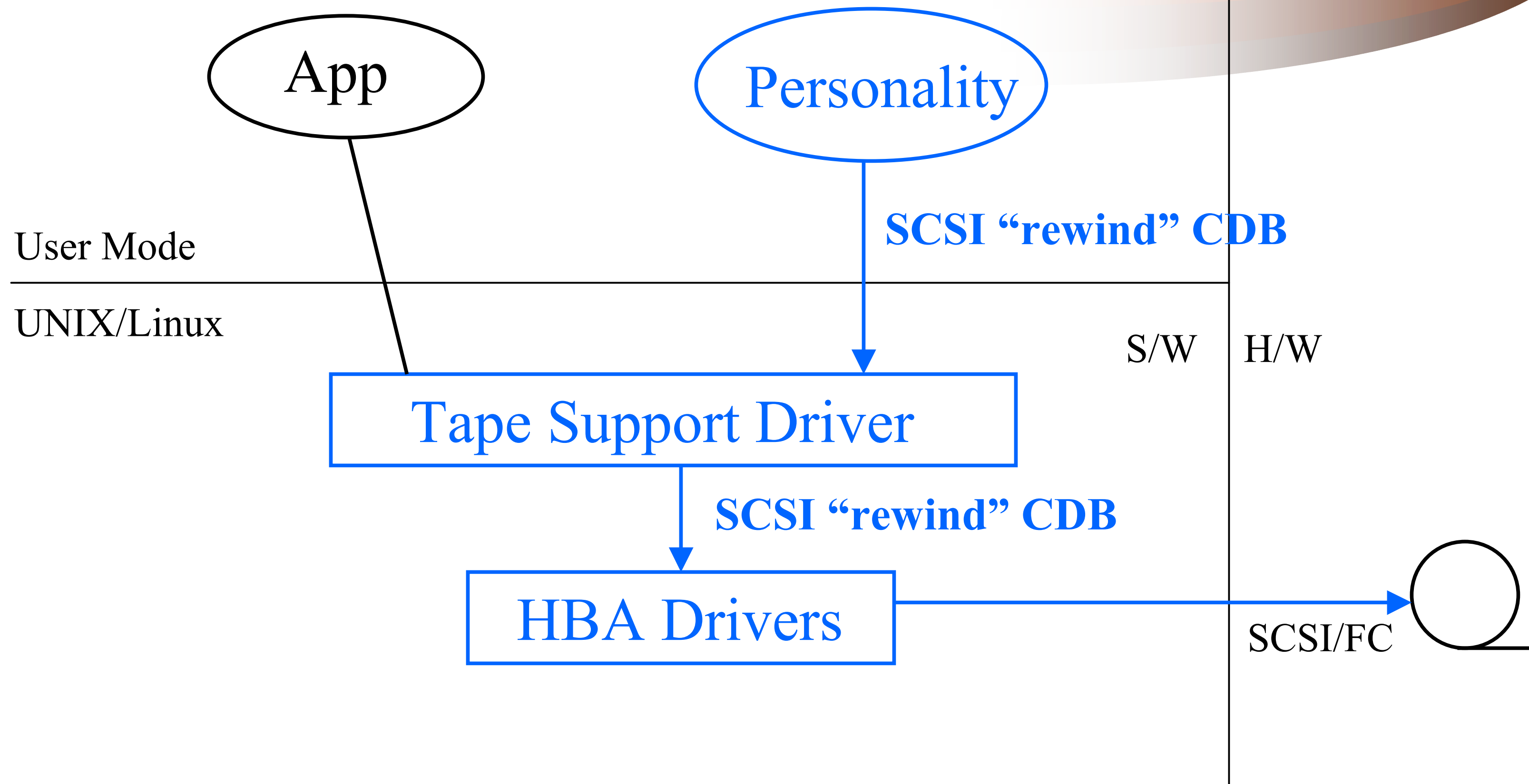
Personality Approves of the Open



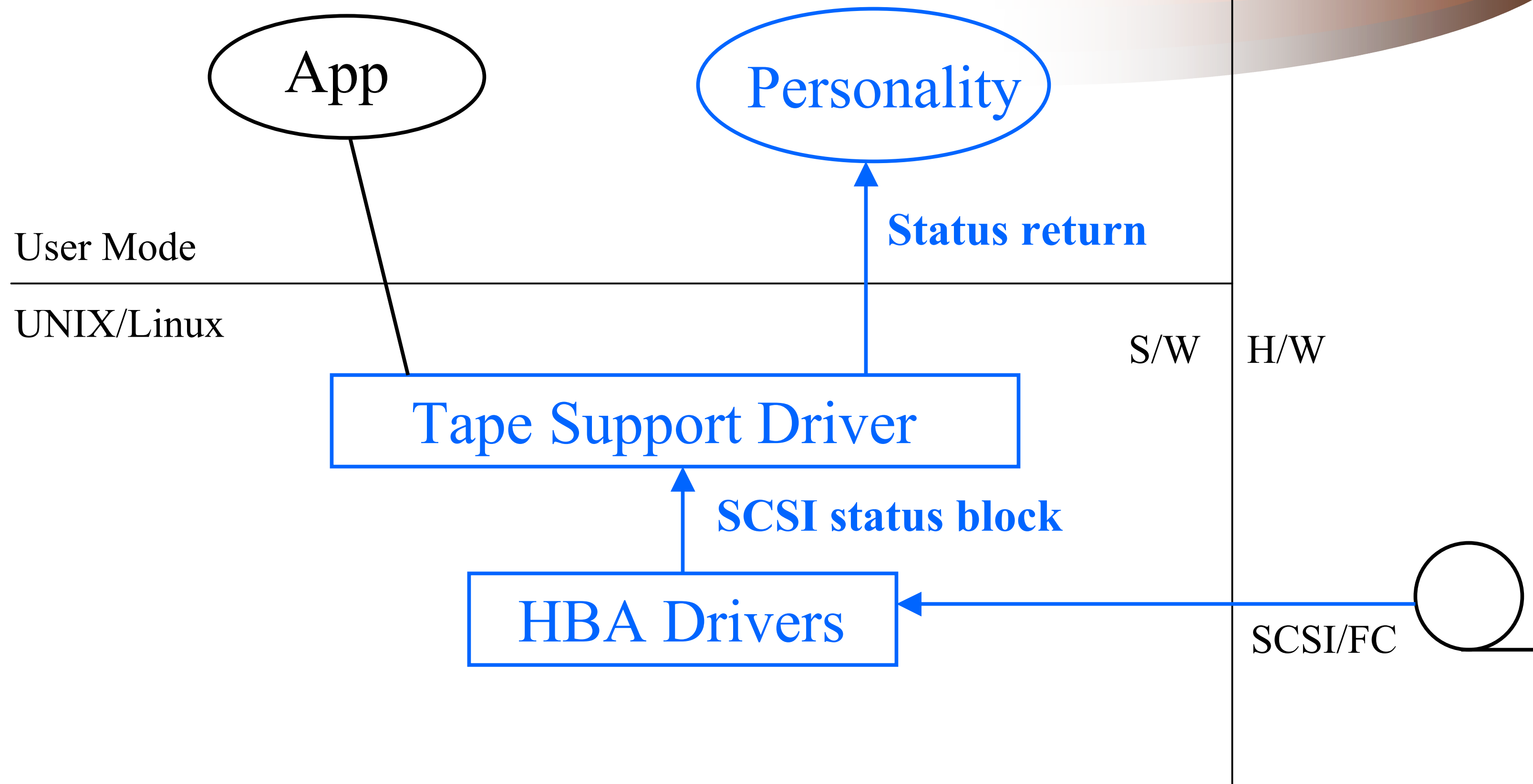
App Asks For Rewind()



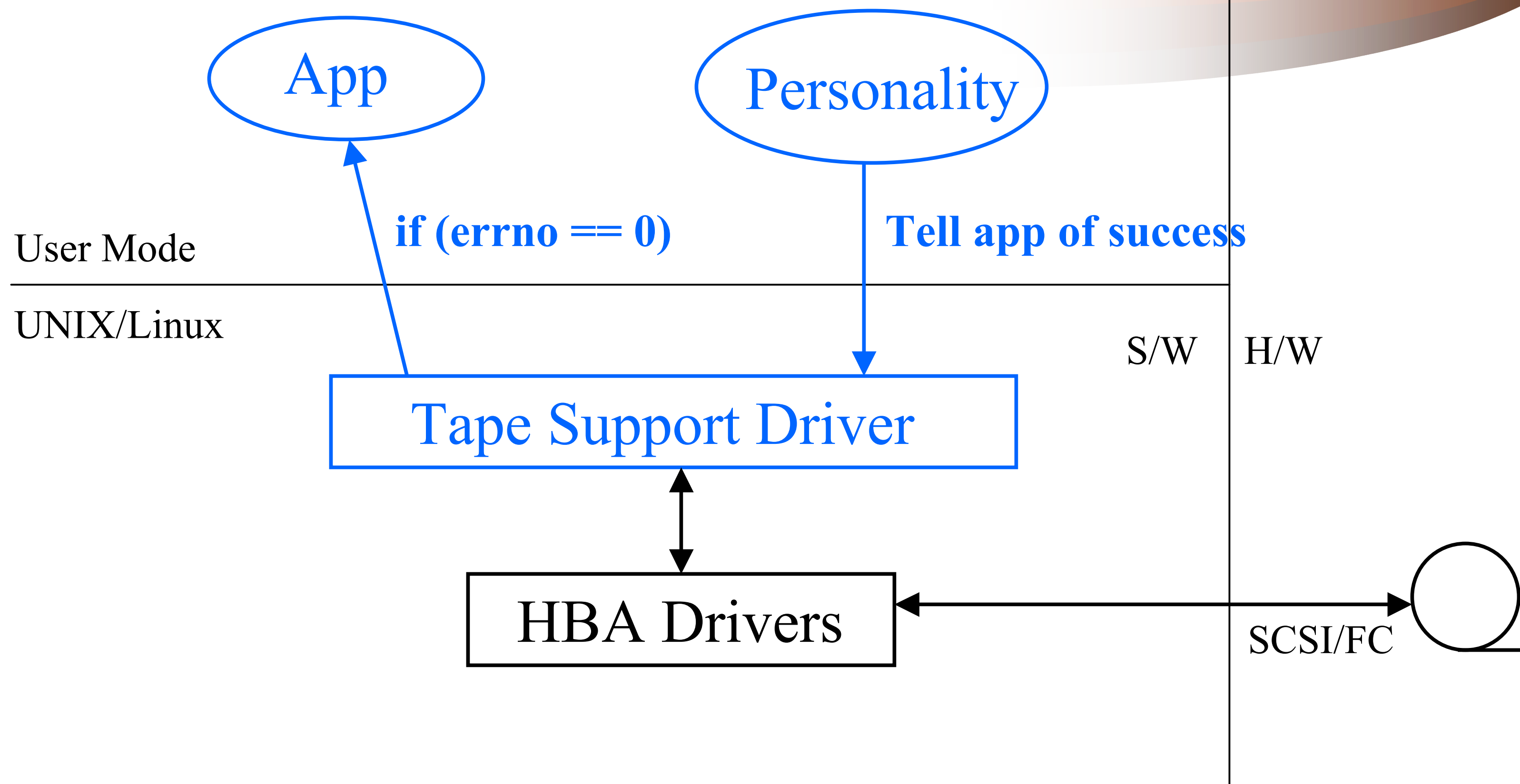
Personality Tells Drive to Rewind



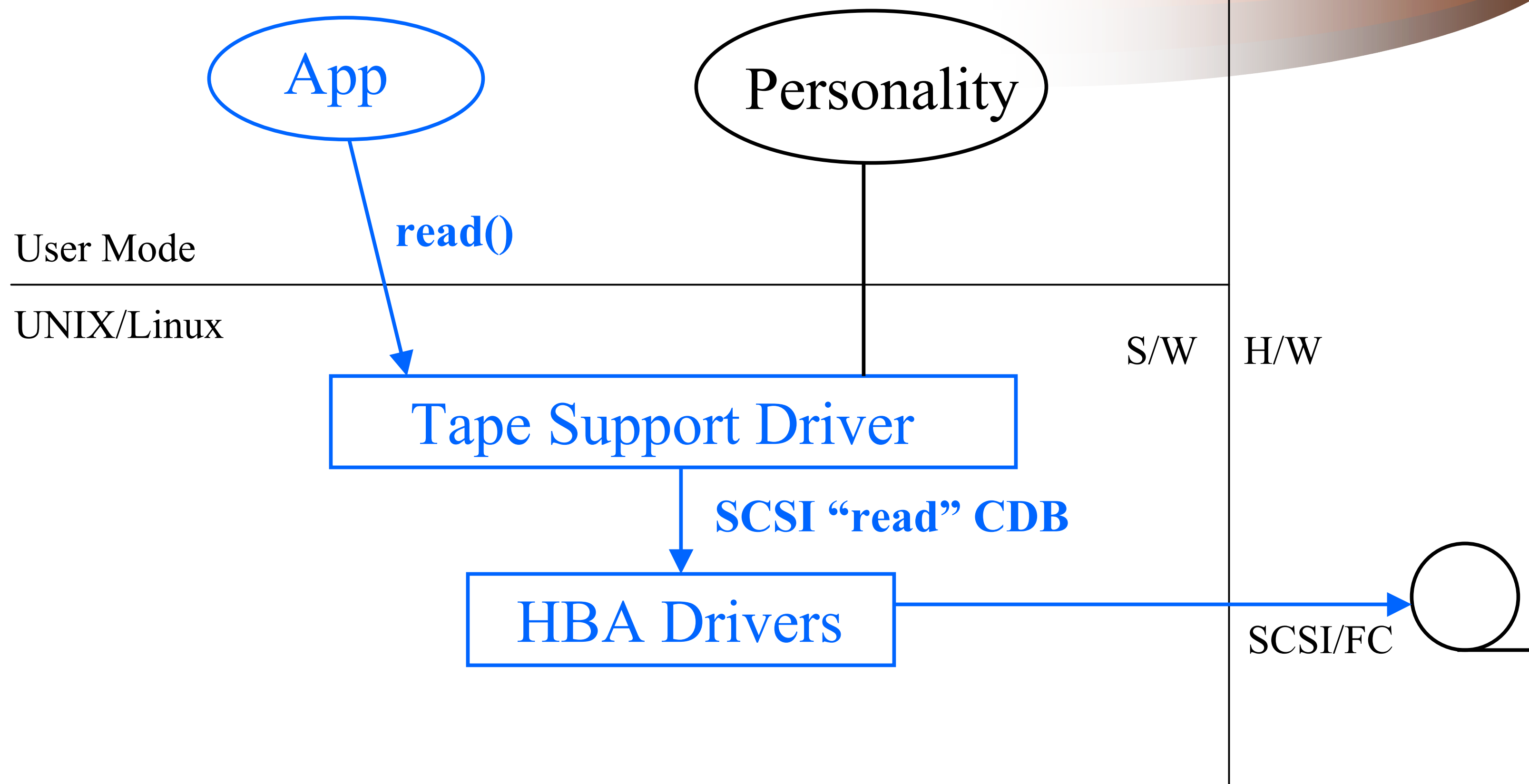
Drive Returns Status to Personality



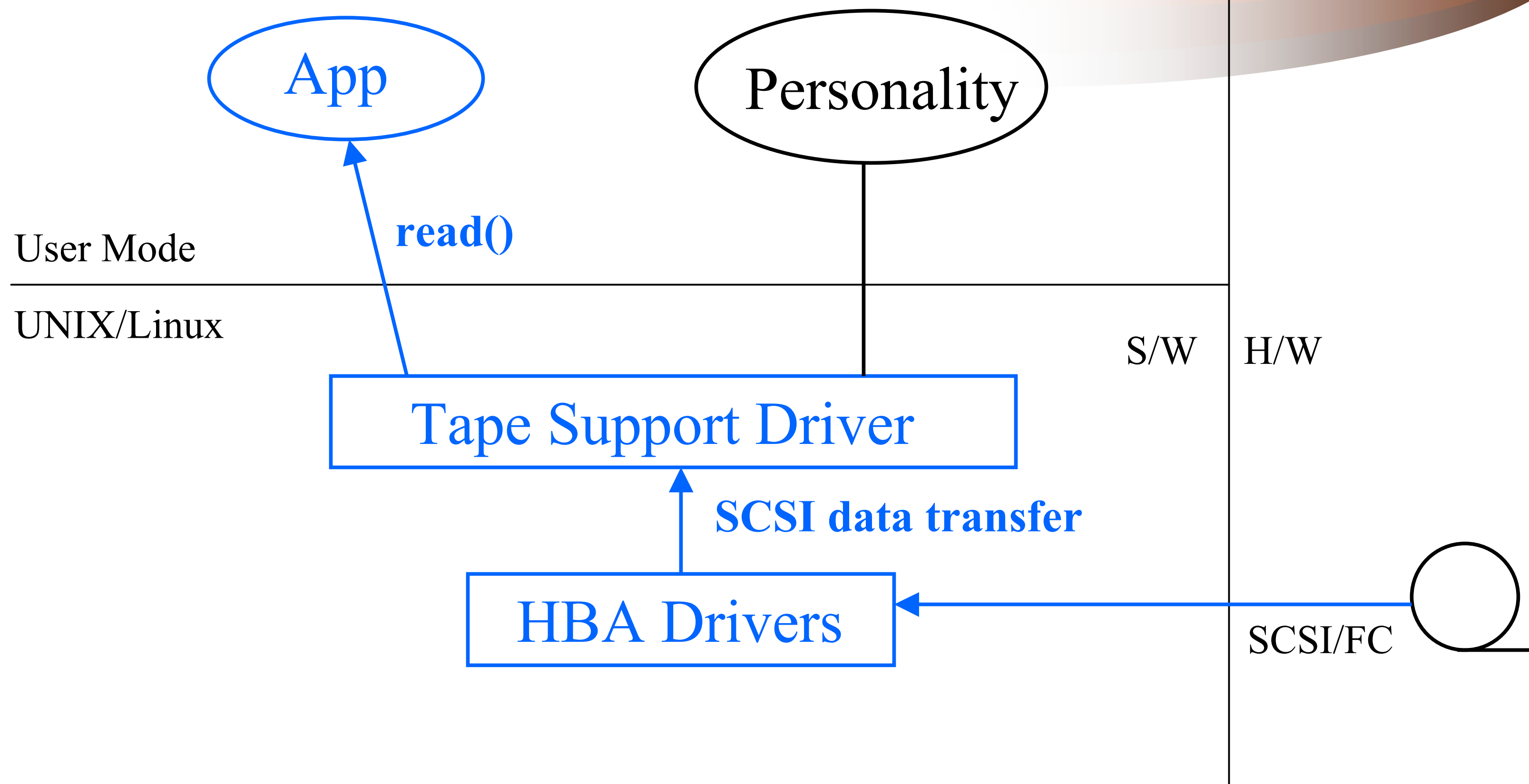
Personality Tells App Operation Done



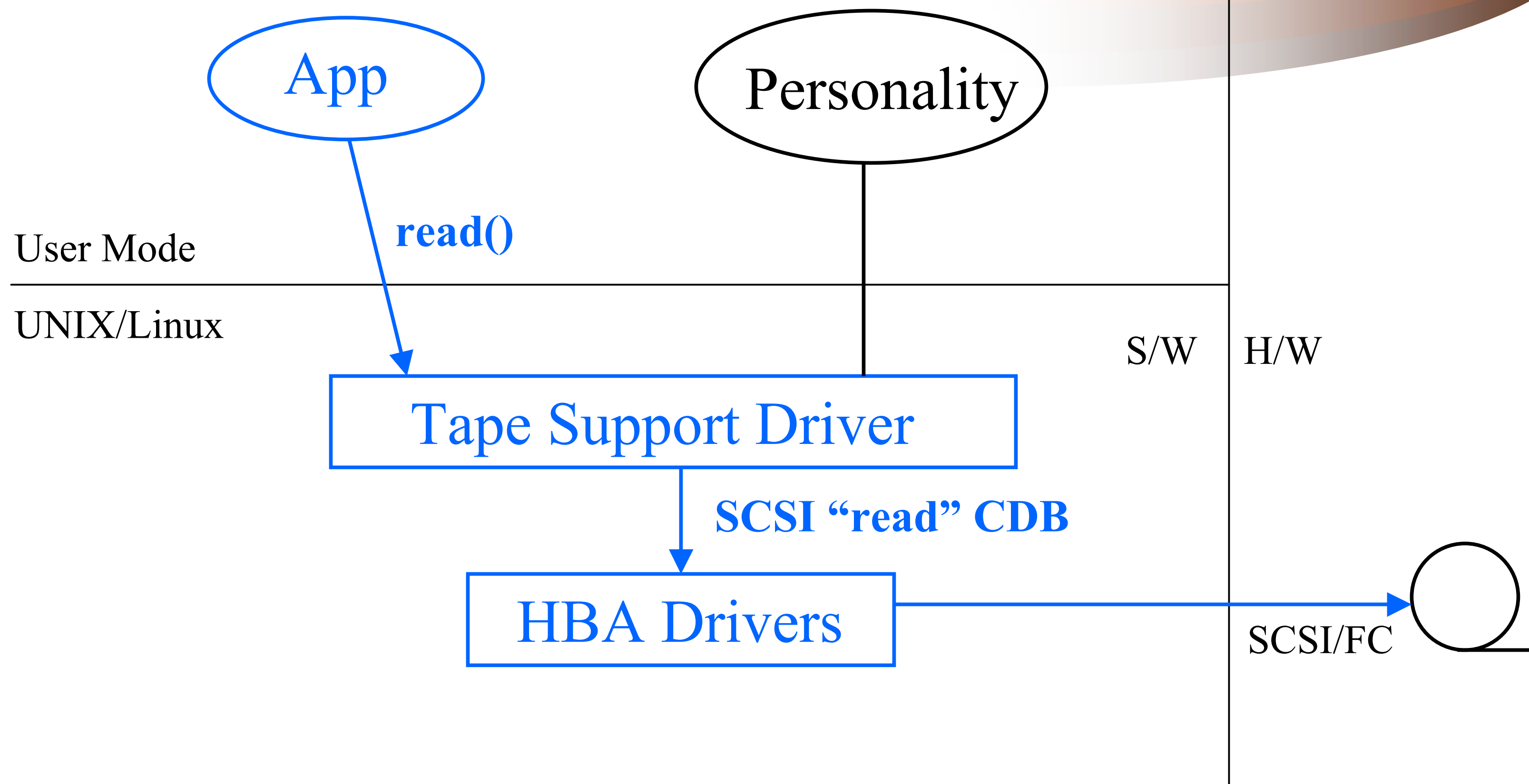
App Does a Read



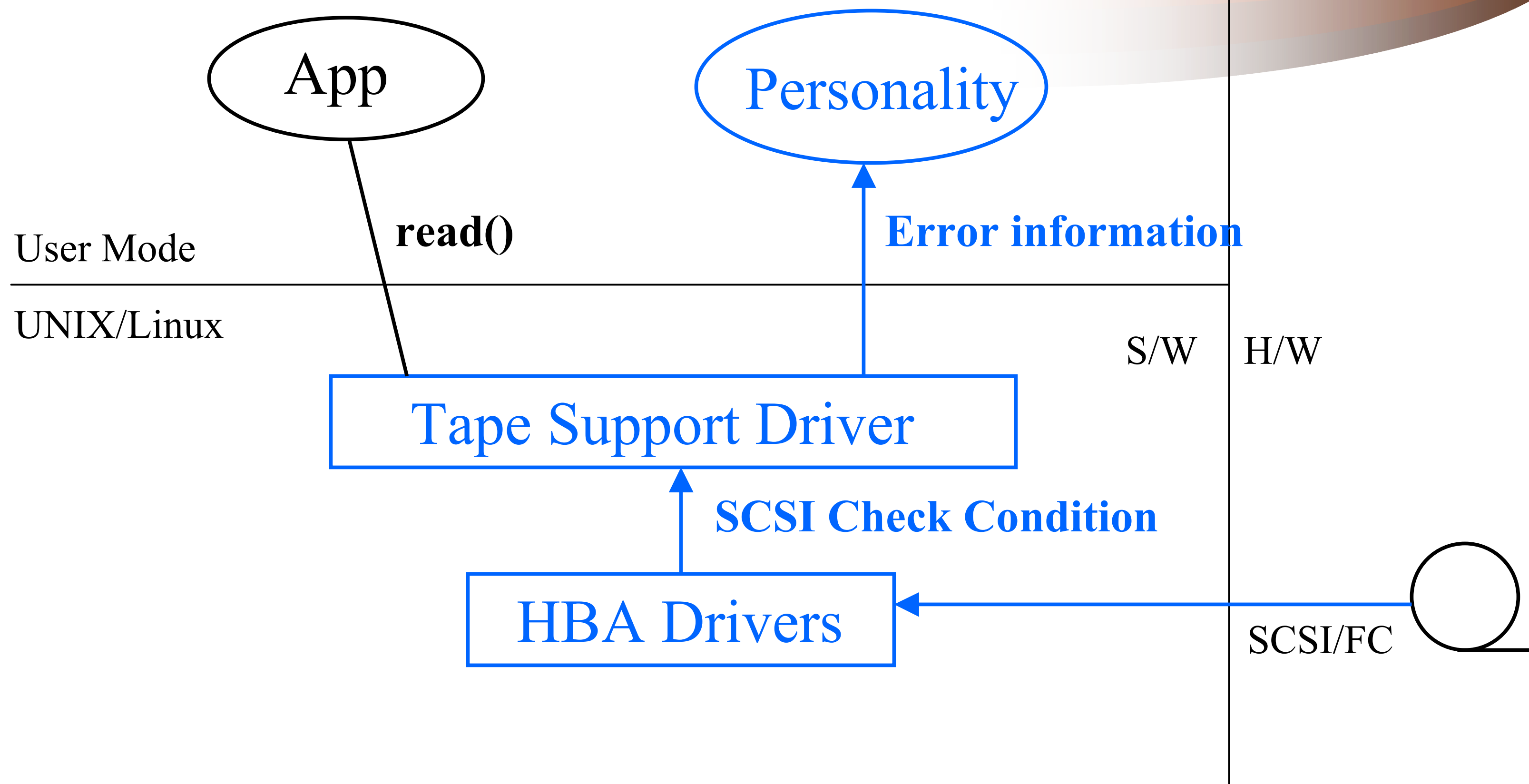
Device Completes Read Successfully



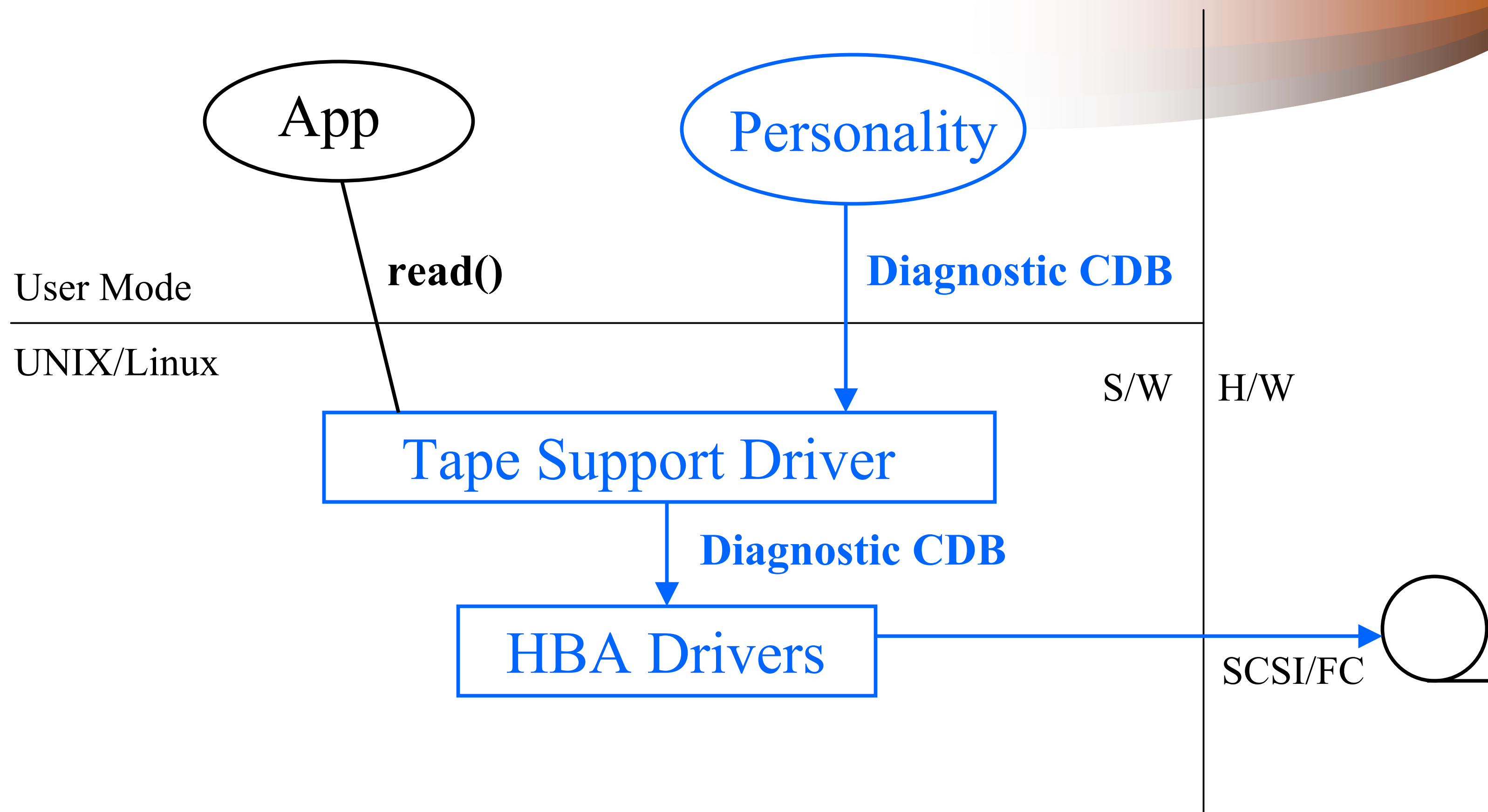
App Does Another Read



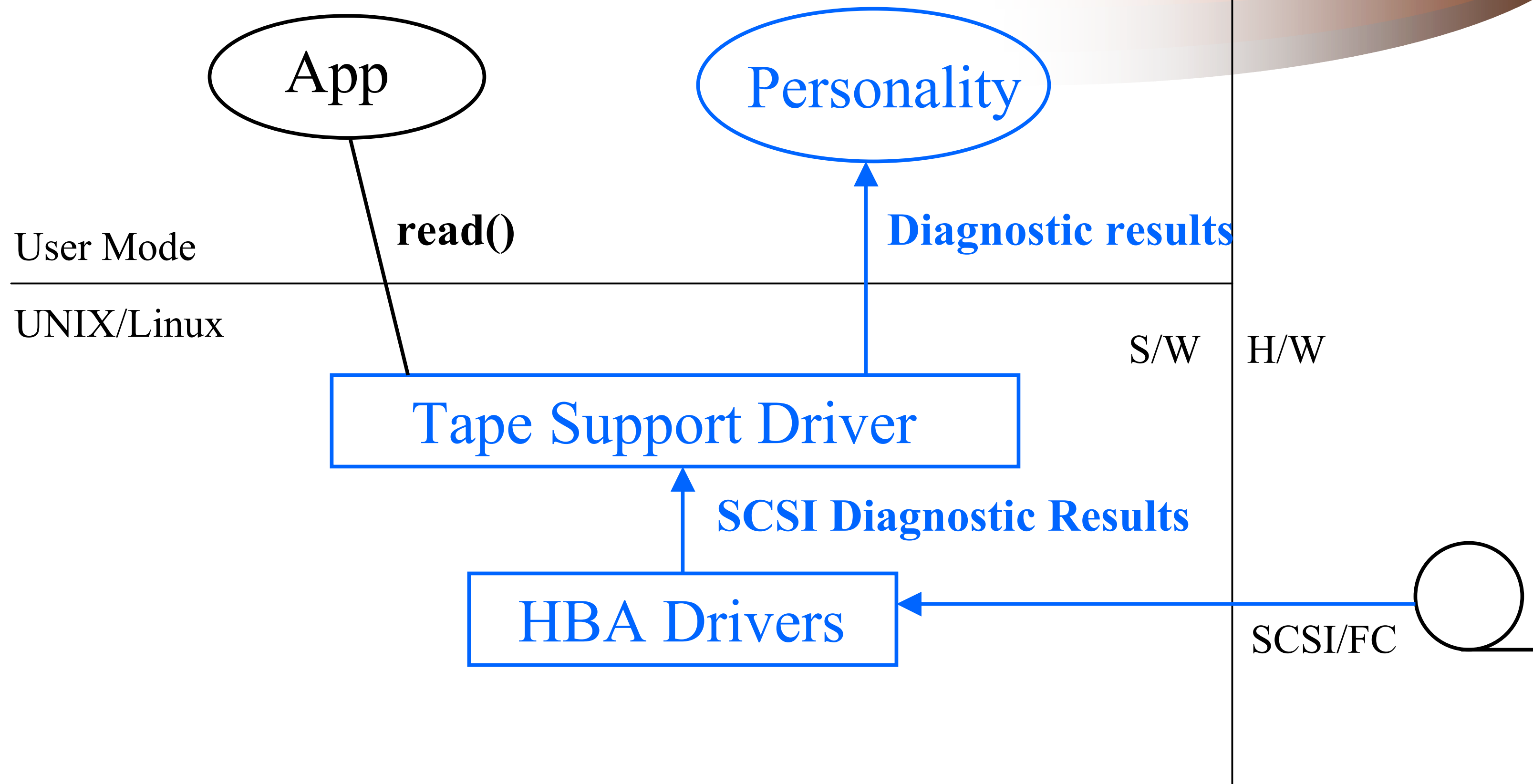
Device Takes an Exception



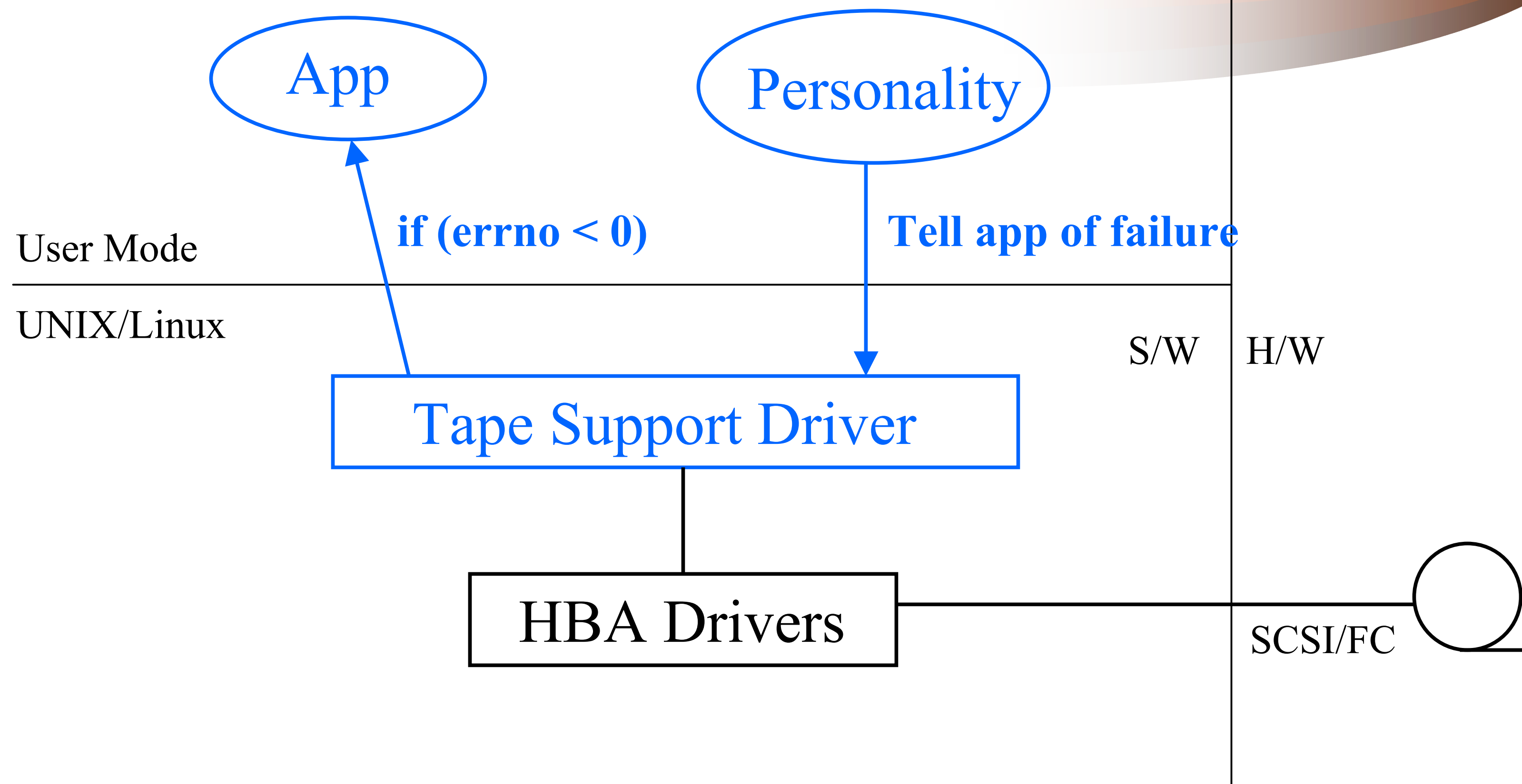
Personality Runs Diagnostics



Device Returns Diagnostic Results



Personality Tells App Operation Failed



For More Information

- IEEE P1563 Tape Driver Semantics
  - Standard plus Recommended Practice on driver design
  - Curtis Anderson <canderson@TurboLinux.com>
  - Neil Bannister <nb@sgi.com> - Open Source code
- IEEE P1244 Media Management System
  - www.SSSWG.org - standards working group
  - www.OpenVault.org - Open Source implementation