Next Generation ESX Storage:

A Pluggable Core Storage Architecture

Tom Phelan

Staff Engineer

VMWORLD 2006

Motivation

- The current ESX storage stack is highly functional, but limited in flexibility and extensibility.

- The Pluggable Core Storage Architecture addresses these limitations.
  - Implements clear APIs between the various components of the storage stack.
  - Allows third party software vendors to port their proprietary multipathing and load balancing code to the ESX platform.
  - Allows SAN array hardware vendors to write device specific code for ESX that takes advantage of advanced array features without compromising proprietary information.

Audience

- This presentation is focused at software developers and development managers.
- The aim is to describe the current state of the ESX Pluggable Core Storage Architecture and ellicit feedback from our partners and customers about their needs in this area.

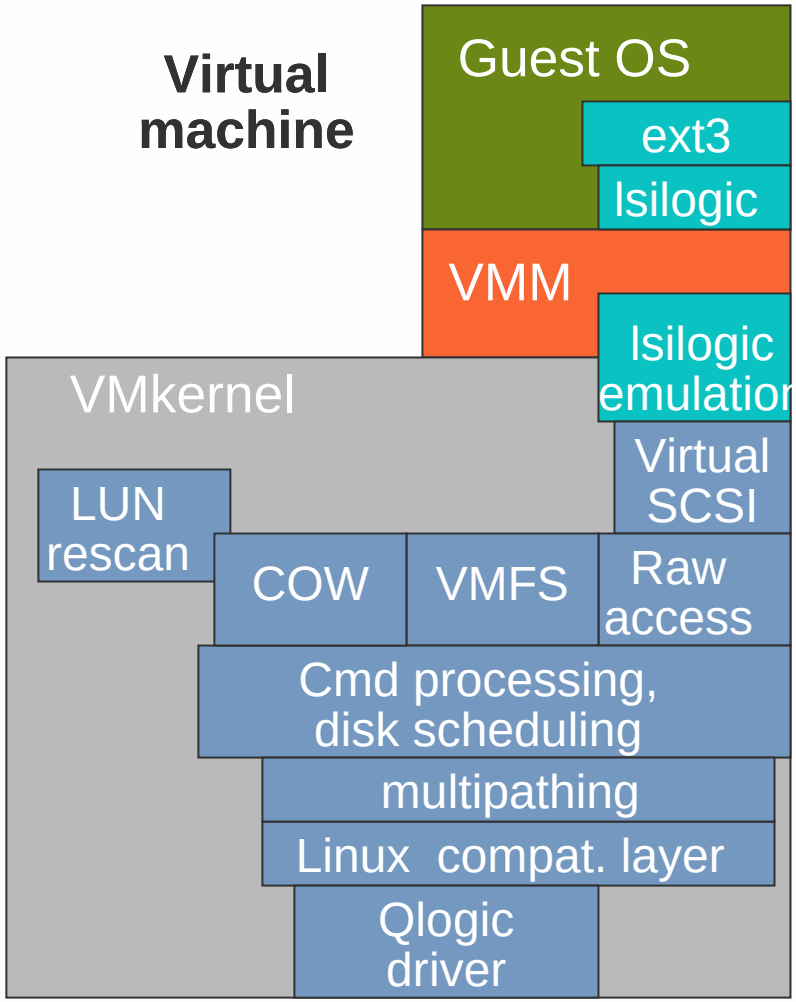
Outline

- Core Storage Architecture Definition
- Motivation For Change
- ESX Storage Architecture
  - Existing Storage Architecture
  - New Storage Architecture
- ESX Storage Plugin
  - Storage Plugin Framework Functionality
  - Storage Plugin Functionality
- Native Multipathing Implementation of an ESX Storage Plugin

Definition

- ESX Core Storage Architecture
  - The part of the storage stack that resides in the ESX operating system (vmkernel)
    - “below” the Virtual SCSI implementation,
    - “above” the Linux compatibility and physical disk driver layers

Core Storage Architecture



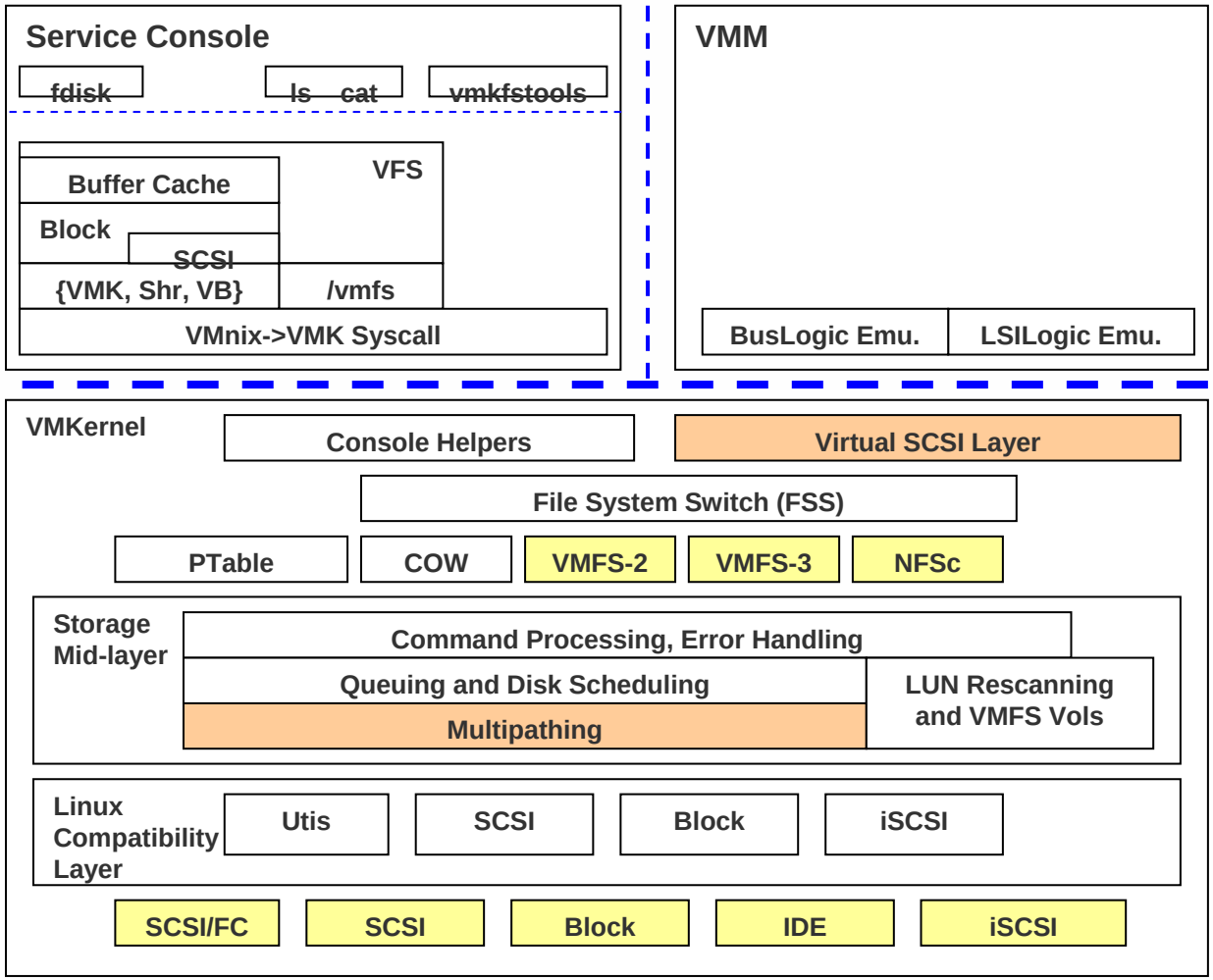
- Common illustration of vmkernel architecture
- Core Storage Architecture Includes
  - Part of command processing
  - Disk scheduling
  - Path scanning
  - Multipathing

Motivation for Changing the Core Storage Architecture

- Enhance the ESX Server infrastructure for storage partner integration
  - Empower Software partners to port their proprietary multipathing products to the ESX platform
  - Allow Storage partners to take full advantage of their device specific functionality within ESX
    - Protect array specific code from regression (recertification)
  - Speed the time to market of the support for new storage arrays
    - Enable asynchronous support for new arrays
  - Promote the proliferation of enhanced load balancing algorithms

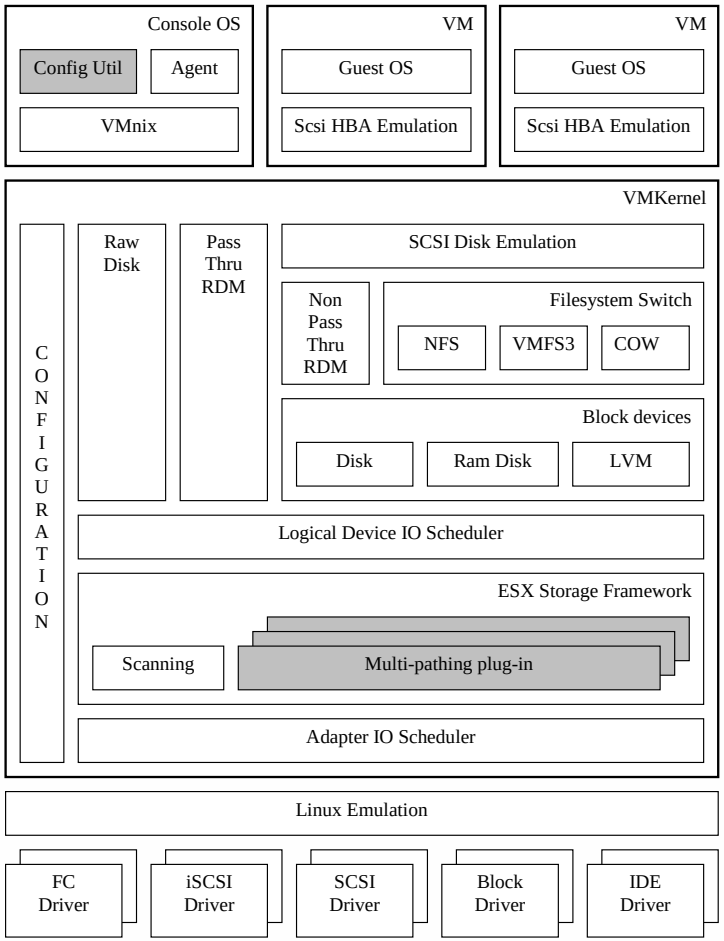
- Pack ESX with more functionality than VMware can provide all by itself!

Existing ESX Storage Stack Architecture



- Monolithic design.
- No support for isolating 3rd party proprietary code.

New ESX Storage Stack Architecture



- Well defined functional layers
- Isolate SCSI emulation code.
- Support multiple file system types
- Support multiple block device types
- Improved path scanning functionality
- Improved storage management
- Provide APIs for
  - Multipathing implementations
  - Device Drivers

Plan To Get To The New Architecture

- Redesign the Storage Stack
  - Introduce functional layers
  - Implement the new ESX Core Storage framework
  - Develop an example of an ESX Storage plugin
- Port ESX native multipathing to the Core Storage framework
  - Modularise the code: core, array plugins, path policy plugins
  - Port the failover engine to the Storage framework
  - Port all array specific plug-ins to the new native multipathing APIs.
  - Develop example array specific and path policy plugins.

Benefits of Using VMkernel Infrastructure APIs

- VMkernel development kit
  - Build environment: compiler, header files, makefiles
  - Code samples and Documentation
  - Debugging environment
- Buildable on standard ESX installation
- Pluggable modules can be shipped from 3rd parties
  - Binary Compatible
  - Versionable
- Headers and sample code available to non-Community Source partners

Components of VMkernel Infrastructure API

- The VMkernel API
  - Library of routines used by both internal and external developers
- The VMkernel SCSI API
  - Library of routines specific to SCSI path/device/I/O manipulation
  - Used to implement a new multipathing algorithm or support for a new SAN device.
- Mechanisms to allow 3rd party software to be shipped separately from an ESX release
  - Init time & configuration integration

VMkernel API

- Functionality – basic OS services
  - Memory: heaps, copy to/from virtual address & physical address
  - Locking: spinlock class, semaphore class
  - Worlds: sleep, wakeup, delay
  - Timers and time conversion
  - Character devices
- Benefits
  - Built on existing code currently used within ESX
  - Self-contained header files
  - Opaque data structures hide ESX specifics

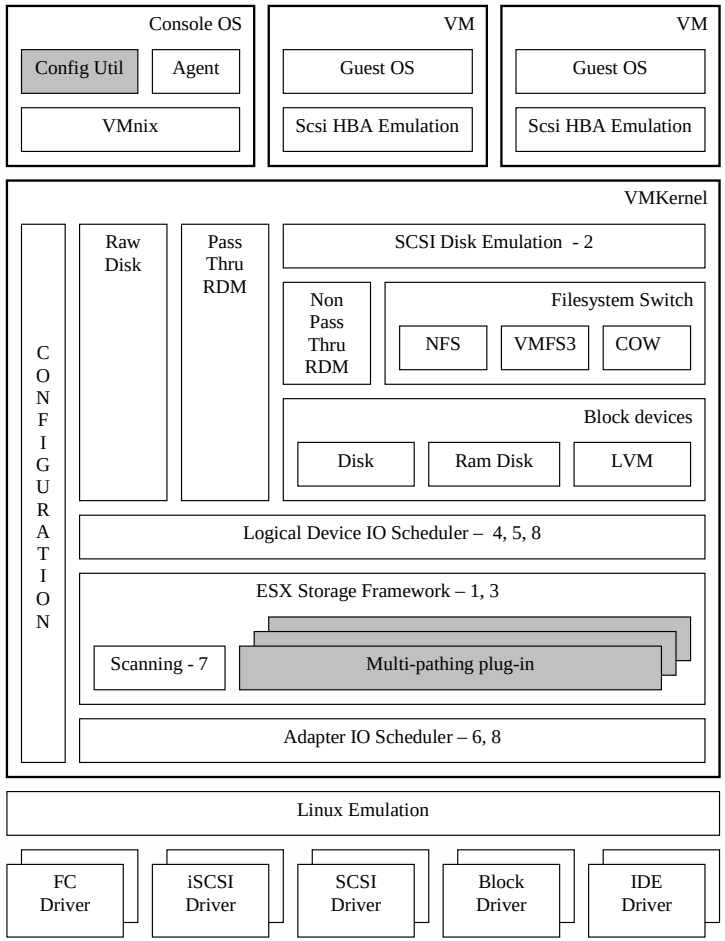
VMkernel SCSI API

- Functionality
  - Plugins: create, register, unregister
  - Device: create, remove, open, close, issue I/O, abort I/O
  - Paths: claim, set state, issue I/O, complete I/O
  - Commands: create, issue sync, issue async
- Benefits
  - Sharing of common code across storage plugin implementations
  - Provide support for Virtual Machine I/O without having to be aware of the Virtual Machines themselves.

Stages of the ESX Boot Process

- Load RAM disk
- Load COS kernel image
- Load Vmkernel
- Load modules
  - vmklinux
  - vmknmpmod
  - Other mp plugins
- Load Device Drivers
- Scan For Physical Devices
  - Scan paths
  - Claim paths
  - Locate boot device
- Boot COS and Start VMs

Core Storage Architecture Functions



- (1) Provide mechanisms to load/manage/unload multiple storage plugins
- (2) Hide Virtual Machine specific activities from the storage plugins
- (3) Route I/O requests for a specific logical device to the appropriate storage plugin
- (4) Handle I/O queuing to the logical devices
- (5) Implement logical device bandwidth sharing between Virtual Machines
- (6) Handle I/O queuing to the physical storage HBAs
- (7) Handle physical path discovery and removal process
- (8) Provide logical device and physical path I/O statistics

Core Storage Plugin Functions

- Manage physical path claiming and unclaiming
- Manage the open and closing of logical devices
- Process I/O requests to logical devices
  - Select an optimal physical path for the request
  - Handle failures and request retries if necessary
    - Specific actions may be required for storage devices.
- Support management tasks such as abort or reset of logical devices

Core Storage Plugin States

- Enabled
  - Logical devices are operational
- Path Claiming
  - In the process of claiming physical paths and exporting logical devices
    - May be different behaviour for active/active and active/passive arrays.
- Disabling
  - In the process of quiescing logical devices
- Disabled
  - Logical devices are not operational

Path Discovery Process

- Path scanning
  - ESX Storage Framework scans the device.
  - Breakout scan process for new path discovery from scan process to remove missing paths.
    - Create/Keep/Destroy
  - The physical paths end up in the pool of unclaimed paths

Storage Plugin Path Claiming Process

- The Storage Framework sends a “start of path claiming” notification to all plugins
- Next the Framework presents the paths to the plugins based on inquiry data and configuration file rules
  - A plugin can claim or reject the path
  - Only a single plugin can claim any given path
- Finally the framework sends an “end of path claiming” notification to all plugins.
  - Now that a plugin has all the paths, it may create a logical device and expose it the Framework.

Path Claiming Rules

- Sample vmkpath.conf
  - Model String “Disk Type A ” Plugin “mpmgmt#1”
  - Model String “Disk Type B ” Plugin “mpmgmt#2”
  - Model String “Disk Type C ” Plugin “mpmgmt#1,mpmgmt#2”
  - Model String “” Plugin “vmware-nmp”
- Default vmkpath.conf:
- Model String “” Plugin “vmware-nmp”
- Reality will very likely be a more complex set of rules

Logical Device Identifiers

- Support multiple Identifiers per logical device
- Legacy unique identifier (UID)
  - Format is ESX proprietary.
  - Needed for backward compatibility/upgrade
    - RDMs, VMFS File system signature
- Standards compliant UID
  - UTF-8 string version:
    - naa.1F92A3E84B4D56C7
    - eui.9A18F7B27E6C54D3
    - iqn.1987-05.com.cisco:01.7fef329e15c8
  - Same UID across all storage plugins
- Storage plugin specific identifiers

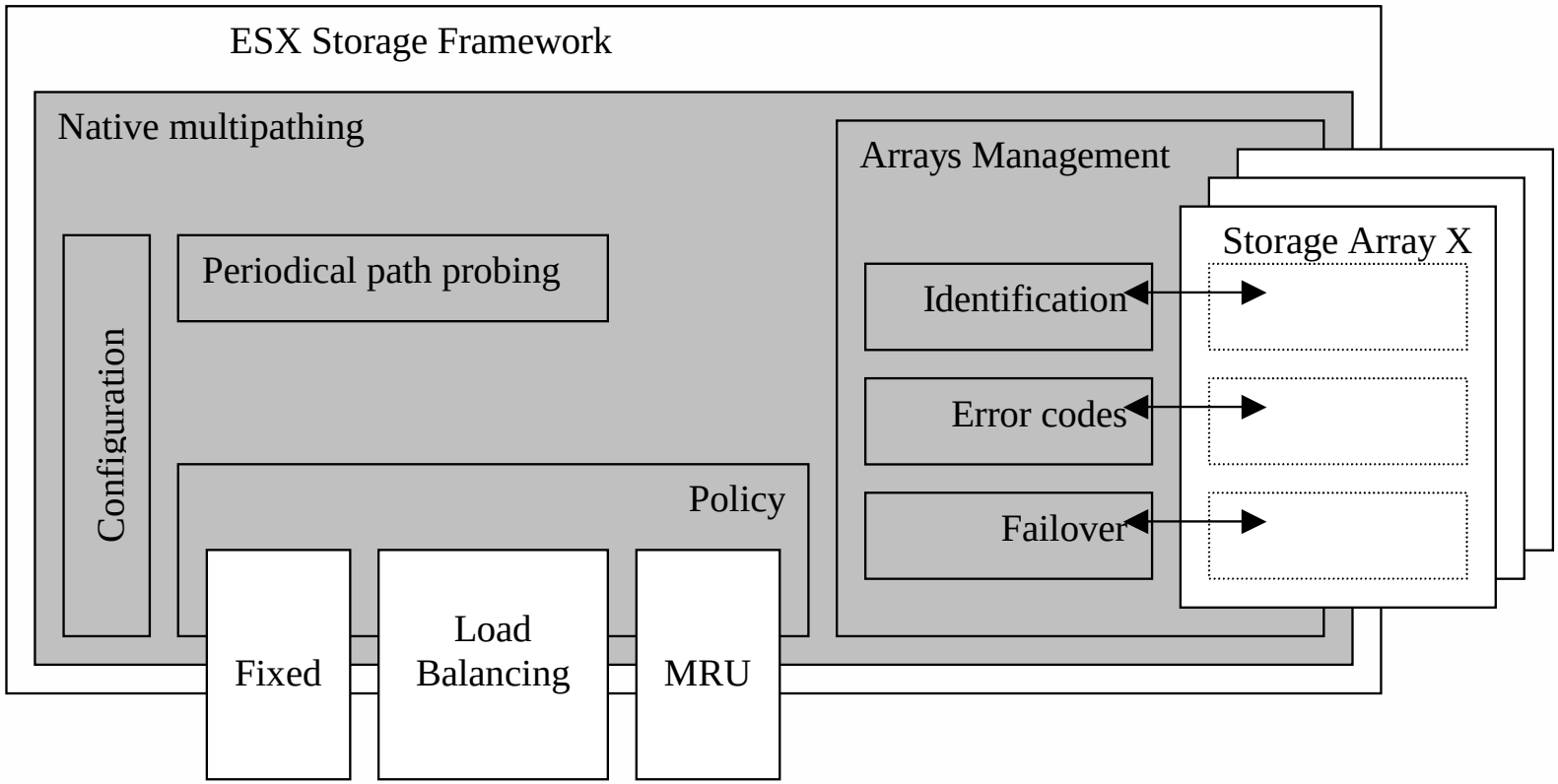
ESX Native Multipathing Plugin Specific Goals

- Replace existing ESX multipathing code
  - Isolate Storage Array Type specific implementations (SAT)
    - active/active
    - active/passive
  - Isolate Path Selection implementations (PS)
    - Fixed path selection
    - Most Recently Used path selection
    - Load Balanced path selection
- Provide public APIs for SAT and PS plugins
- Simultaneously co-exist with other storage plugins
- Support true path load balancing
- Decrease the time and effort involved in supporting new SAN devices.

NMP Initial Scope

- Replace the existing ESX multipathing code
  - Implement as a plugin using the new Core Storage Framework
  - Use only the VMkernel API and VMkernel SCSI APIs.
- Path Selection Specific Plugins (PSP)
  - Fixed
  - MRU
  - Load balanced
- SAN/Storage Array Type Specific Plugins (SATP)
  - All the arrays currently supported by ESX 3.0
- SAT and PS plugins can be shipped separately from an ESX release.

ESX Native Multipathing (NMP)



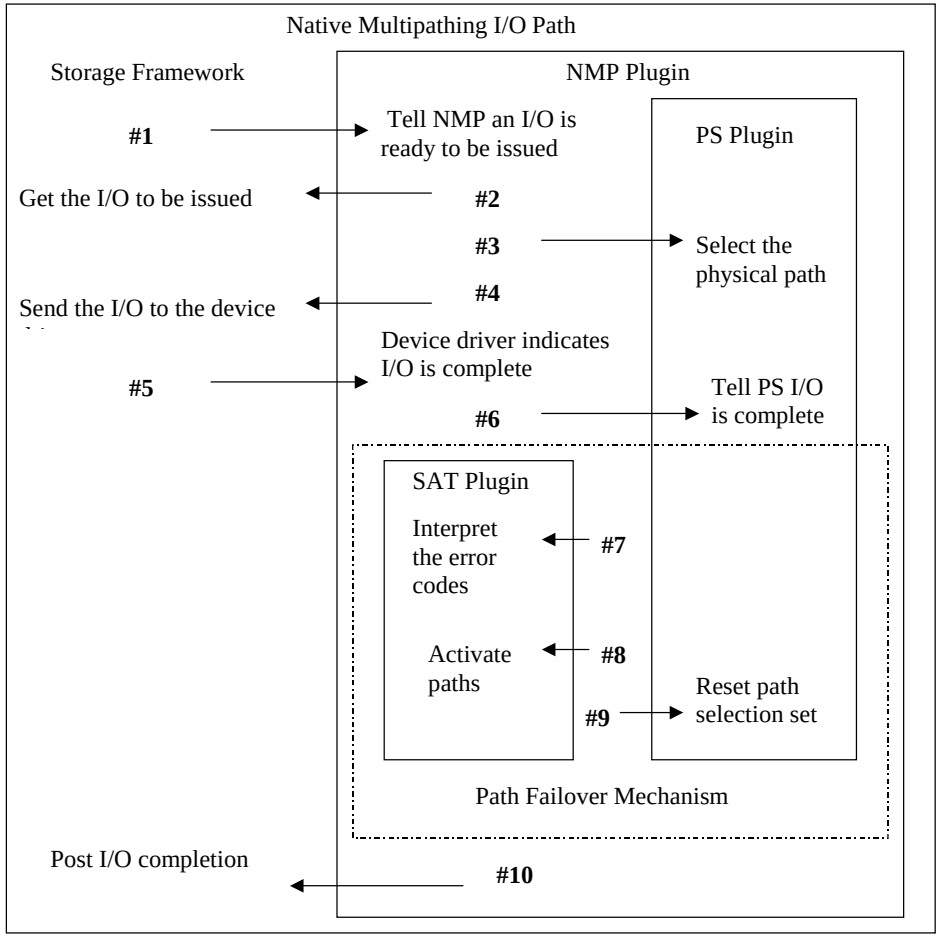
NMP SAT Plugin

- Settable per physical path
- Handle array specific actions
  - For Active/Passive Devices
    - Detect SP failures
    - Activate passive paths
- Called for each I/O request that fails to complete successfully
- Monitor health of each physical path.
- Periodically queried to report the state of each physical path to the NMP plugin
- SATP_UNKN by default

NMP PS Plugin

- Settable per logical device
- Called on each I/O request to a logical device
- Determine which of the physical paths should be used when issuing an I/O request to a logical device.
- PSP_UNKN by default

NMP Flow of I/O



Status

- Work on the new Storage Framework and the NMP plugin is underway
- Storage partners have expressed interest in developing software for this framework

Many Ways to Add Functionality to ESX

- Device Driver Toolkit (DDK)
  - When adding a device driver
- Native Multipath Plugin
  - When adding support for a specific SAN array or path policy
- Storage Stack Plugin
  - When adding support for a complete multipathing mechanism
- Community Source
  - When modifications beyond the scope of the existing APIs is required

Questions?

Rochan Dhand – rdhand@vmware.com

Presentation Download

Please remember to complete your
session evaluation form
and return it to the room monitors
as you exit the session

The presentation for this session can be downloaded at
http://www.vmware.com/vmtn/vmworld/sessions/

Enter the following to download (case-sensitive):

Username: cbv_rep
Password: cbvfor9v9r

Some or all of the features in this document may be representative of feature areas under development. Feature commitments must not be included in contracts, purchase orders, or sales agreements of any kind. Technical feasibility and market demand will affect final delivery.

VMWORLD 2006

The remaining slides are backup information only

- These slides describe how the data structures in the new framework may be defined.
- These slides show how the VMkernel SCSI API routines may be called.
- These slides are only illustrations to describe concepts. The API will evolve before its initial release with ESX.

Data Structures – Reference Counting

- Getting a reference
  - Lookup by name / identifier
    - SCSILookup{Adapter|Path|Device|Plugin}()
  - Iteration over all instances
    - SCSI{Adapter|Path|Device|Plugin}IteratorInit()
      - Returns the first object: takes a reference
    - SCSI{Adapter|Path|Device|Plugin}IteratorNext()
      - Return the next object: takes a reference on the new instance and drops it on the previous instance
    - SCSI{Adapter|Path|Device|Plugin}IteratorDone()
      - Stop iterating: drops the ref on the current object if any
  - Duplicate a reference
    - SCSIRef{Adapter|Path|Device|Plugin}

Data Structures Reference Counting

- Dropping a reference
  - SCSIUnref{Adapter|Path|Device|Plugin}
- Destroying the object
  - Prevent new lookups
  - Sleep until the last reference is dropped
    - Avoids dance with global locks in IO path
  - Returns

Plugin Module Reference Counting

- Always pin a module before accessing it
  - Slow path: acceptable
  - Performance path: not acceptable
    - vmk_ScsiDevice: protected by open/close
      - Device open atomically looks up the device and pins its module
        - *Fails if the module is being unloaded*
    - vmk_ScsiPath: protected by claim/unclaim events
    - vmk_ScsiAdapter: protected by claimed paths

Data Structures – Physical Adapter

- vmk_ScsiAdapter
  - driver name
  - adapter name
  - moduleId
  - capabilities: maxIoSize, maxSgLen, paeCapable
  - entry points: discover, command, taskMgmt, dump
  - dump info: interrupt handler & vector
- ScsiAdapter
  - lock
  - flags
  - useCount
  - Stats

Data Structures – Physical Path

vmk_ScsiPath

- plugin private data

ScsiPath

- Lock
- name
- flags
- useCount
- stats
- inquiry data
- plugin
- Legacy UID

Data Structures – Logical Device

- vmk_ScsiDevice
  - Plugin private data
- ScsiDevice
  - Lock
  - Flags
  - useCount
  - stats
  - plugin
  - device UIDs

Data Structures - Plugin

- vmk_ScsiPlugin
  - Name, moduleId
  - version number, SCSI API revision number
  - entry points:
    - openDevice, closeDevice
    - startCommand, taskMgmt
    - device ioctl, path ioctl, plugin ioctl
- ScsiPlugin
  - lock
  - flags
  - useCount
  - Stats

Data Structures – I/O Command

- vmk_ScsiCommand
  - Issuing Path
    - cdb, cdbLen, sgArray
    - direction, requiredDataLen
    - senseData, senseDataLen
    - done(), doneData
    - worldId
    - commandId: serialNumber, originator
    - timeoutMs
  - Completion Path
    - host status, SCSI status
    - xferredDataLen

Data Structures – Task Management

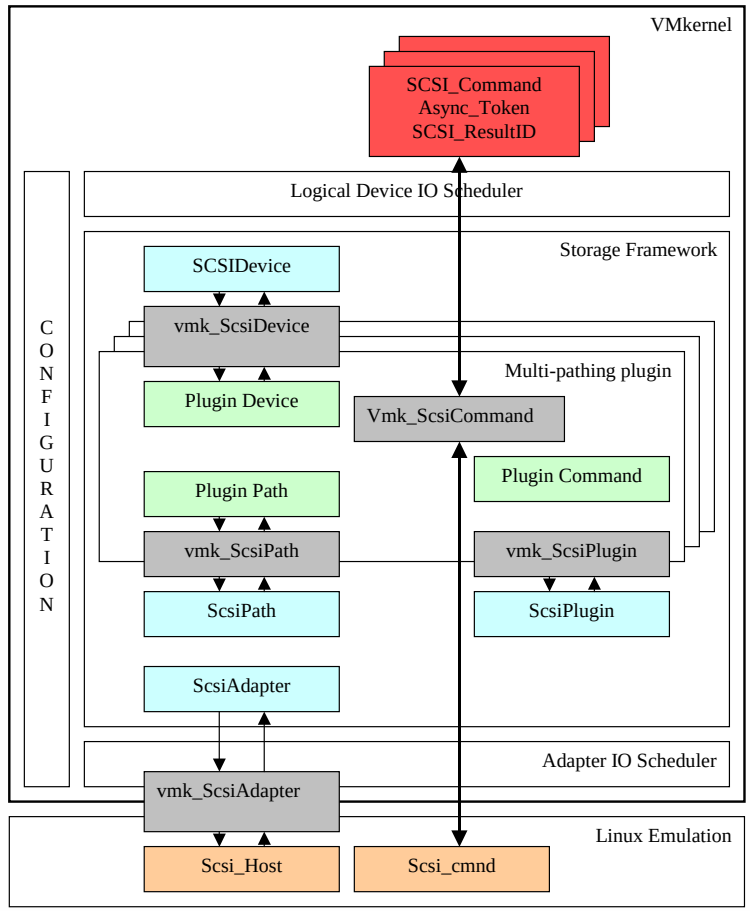
- vmk_ScsiTaskMgmt
  - Task Mgmt Type
  - Completion Status
  - Initiator Id (handle)
  - Serial number

Data Structures

Legacy Structures	VMKAPI	VMKAPI_SCSI
SCSI_Command Async_Token SCSI_ResultID	vmk_ScsiCommand vmk_TaskMgmt	N/A N/A
SCSI_Target	vmk_ScsiDevice	ScsiDevice
N/A	vmk_ScsiPlugin	ScsiPlugin
SCSI_Path	vmk_ScsiPath	ScsiPath
SCSI_Adapter	vmk_ScsiAdapter	ScsiAdapter

I/O Flow through the Data Structures

- SCSI_Command from GOS is converted to a vmk_ScsiCommand
- vmk_ScsiCommand converted to Scsi_Command sent to physical device driver.



I/O Flow Through the Framework