

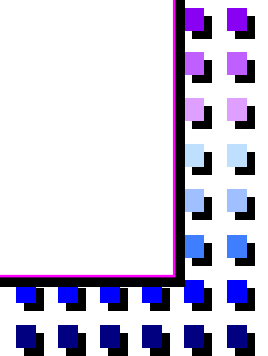


# OpenVault

## Media Library Management for Open Systems



**SiliconGraphics**  
Computer Systems





# Presentation Agenda

- Architecture/Overview — Geoff Peck
- Programming Interfaces & Design — Curtis Anderson and Mark Epstein
- Business Model — Mike Hardy
- Summary
- Discussion





# Programming Interfaces & Design

- Language Overview
- Hello Protocol
- Client API
- Abstract Library Interface
- Abstract Drive Interface
- Administrative API
- Data Model / Database Content





# Tertiary Storage Management

- *Direct Tape Access*: user reads and writes tapes directly
- *Backup*: frequent, automatic writes; infrequent, user-requested reads; automatic aging / recycling of tapes
- *Archive*: manual, user-requested writes and reads; long-term retention of tapes
- *HSM (Hierarchical Storage Management)*: automatic, “transparent” movement of data between tape and disk

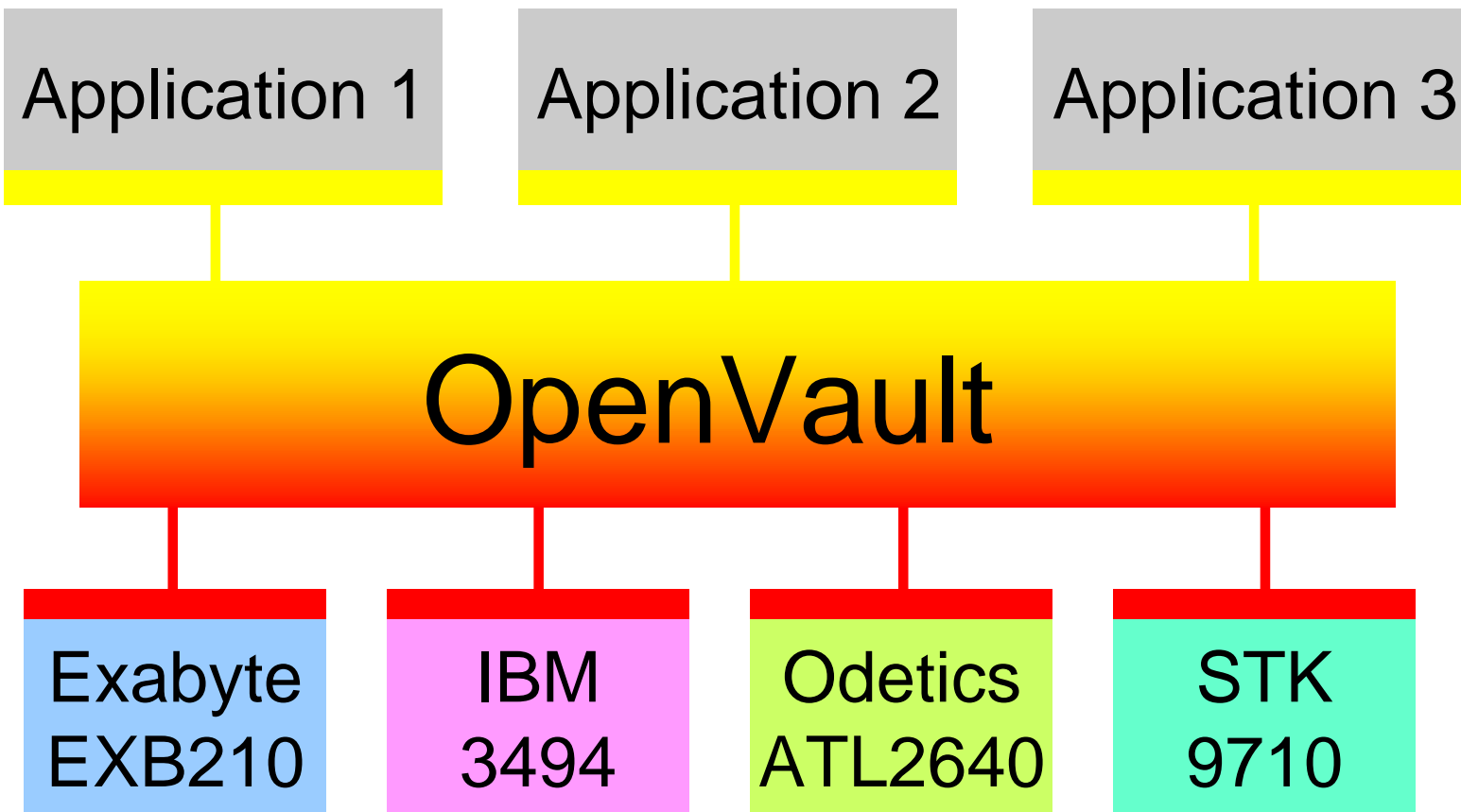


# Robots: Traditional Approach

Application			
Exabyte EXB210	IBM 3494	Odetics ATL2640	STK 9710

- Each application implements own robot control
- New robot → new application release
- No sharing of a robot between applications

# OpenVault Approach





# Open Architecture Goals

- OpenVault clients are *applications*, not end users
- Open interfaces for
  - ◆ applications
  - ◆ robotic library control
  - ◆ drive control
  - ◆ administration
- Portable to UNIX, NT, NetWare, MVS, ...
- Intended to be offered as an open standard





# OpenVault Advantages

- Ease of library access for applications
- Multiple applications, multiple host systems can share one library and one set of drives
- Support manual tape library operations
- Easy to add additional robots or drives without re-releasing core or application s/w
- All components qualified and released independently





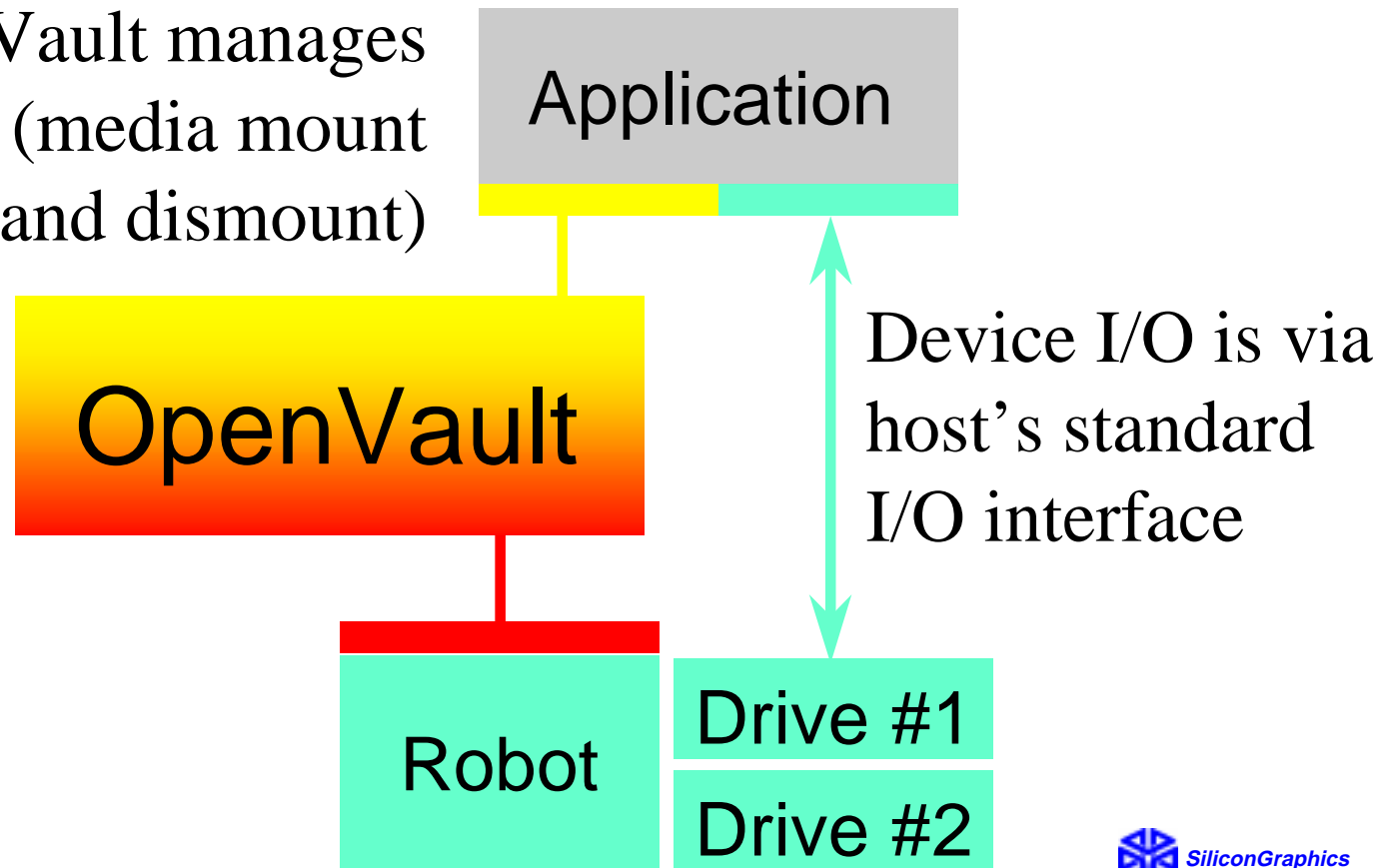
# Non-Intrusive Media Management

- *Don't* impose any format requirements on media content
  - ◆ Standard UNIX tapes don't have labels — data format is application-dependent (*tar*, *dump*, etc.)
  - ◆ Non-computer-readable media such as videotape
- *Don't* get in the I/O path
  - ◆ Allow existing programs to run without change using external mount service and OpenVault



# OpenVault and Device I/O

OpenVault manages the robot (media mount and dismount)



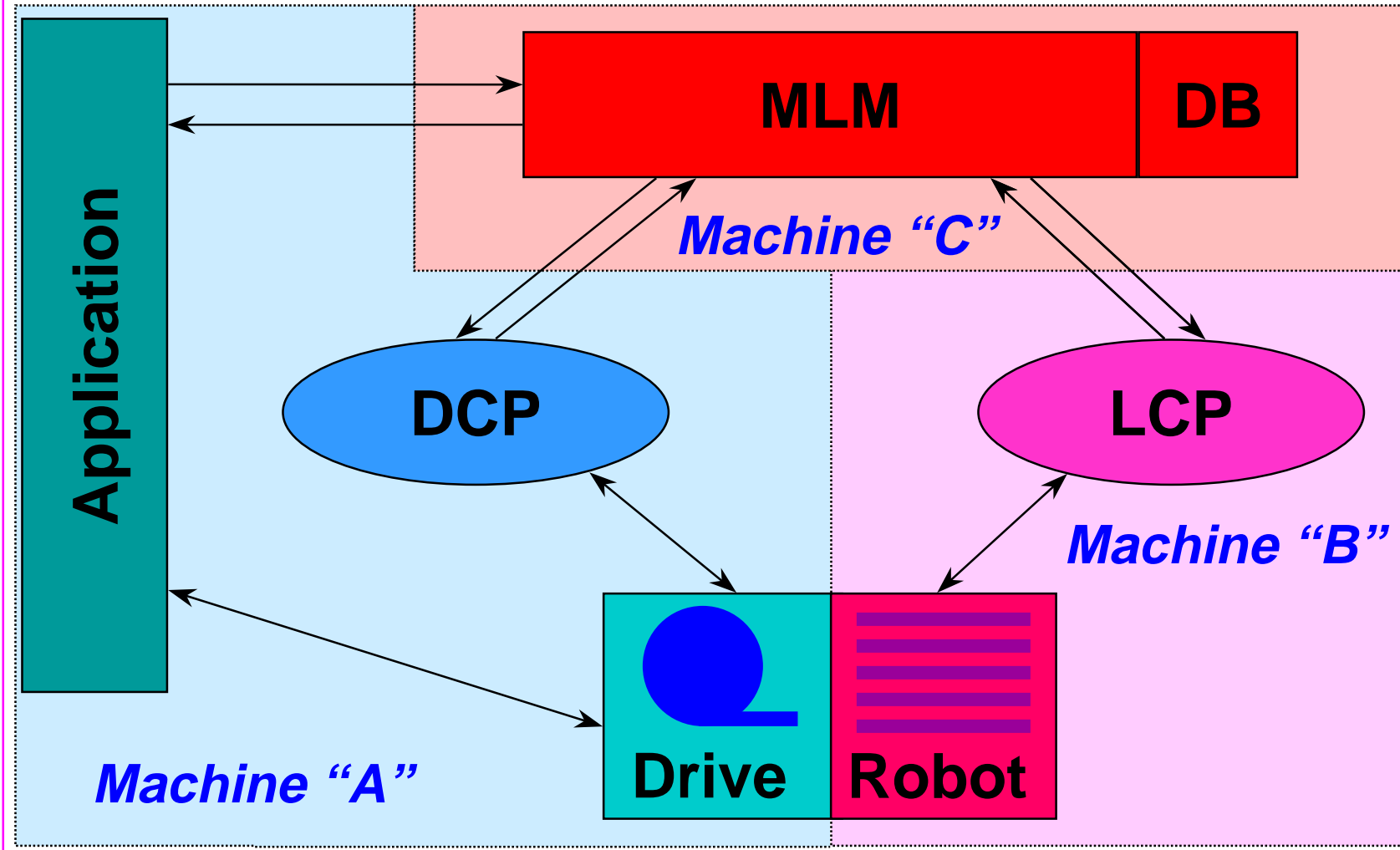


# Site and System Scalability

- Easy enough to use for a single-user desktop system (robotic or manual operation)
- Rich and scalable enough for a major corporate data center or an entire campus
- Implications:
  - ◆ Setup and administrative simplicity
  - ◆ Reduced operational complexity
  - ◆ Very easy to use “by hand” and with applications
  - ◆ Multi-platform
  - ◆ Pricing — *free* at the low end



# Distributed Architecture





# Distributed System Goals

- Protected against network attacks
  - ◆ Utilize digital signatures on all communications
  - ◆ Private keys between components
- Portable to wide variety of systems
  - ◆ Rely on TCP connections only — no RPC, XDR, DCE, etc. required
- Good behavior in the event of failures
  - ◆ Failure of application host or MLM, LCP, or DCP software or host is detected
  - ◆ Integrated with IRIS FailSafe™, etc.





# OpenVault Components

- **MLM** — Media Library Manager
  - ◆ Central resource management daemon
- **LCP** — Library Control Program
  - ◆ Controls specific make/model of library
  - ◆ Robotic or Manual
- **DCP** — Drive Control Program
  - ◆ Controls specific make/model of drive
  - ◆ Tape, magneto-optical, CD-ROM, etc.



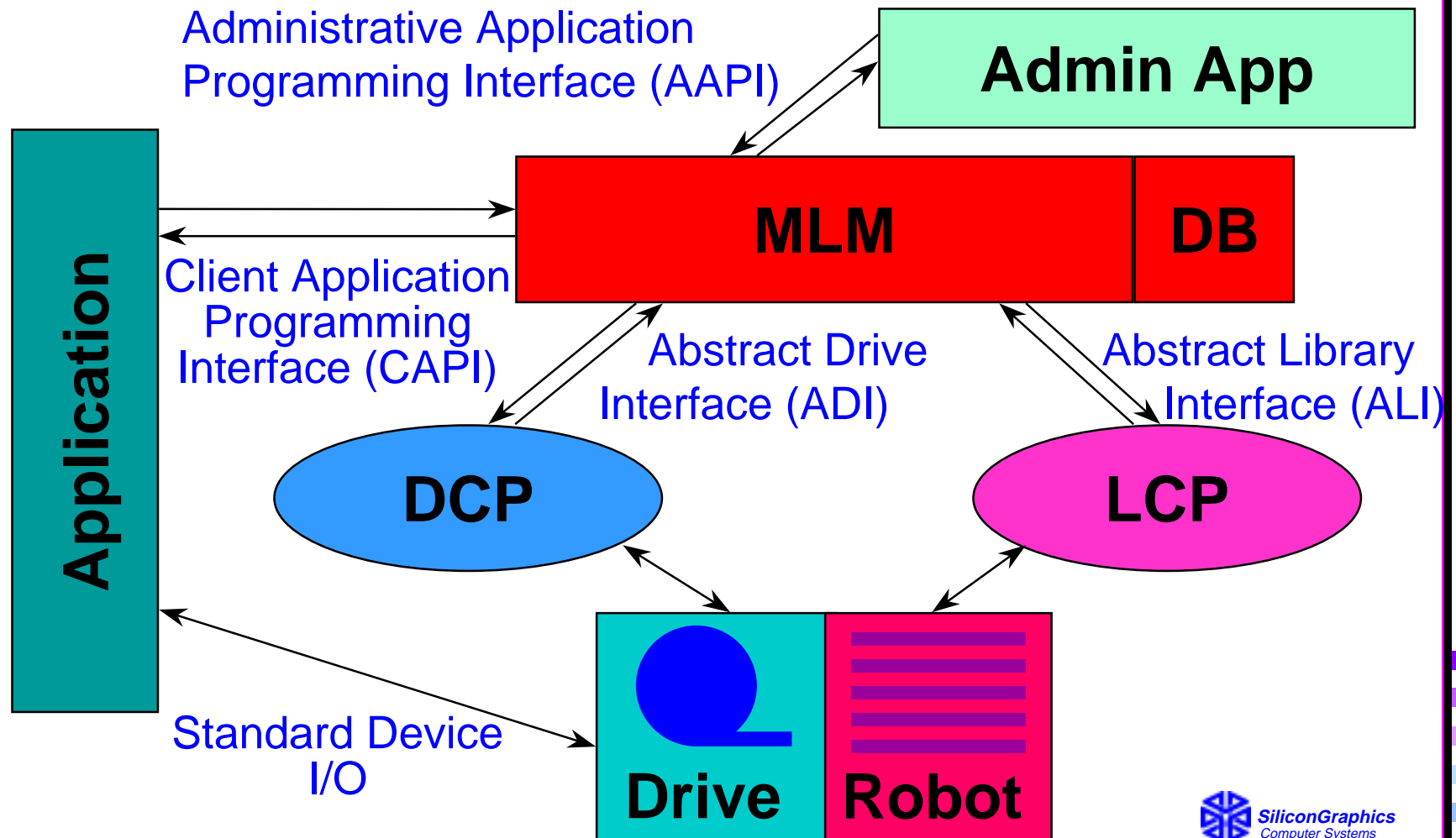


# OpenVault Interfaces

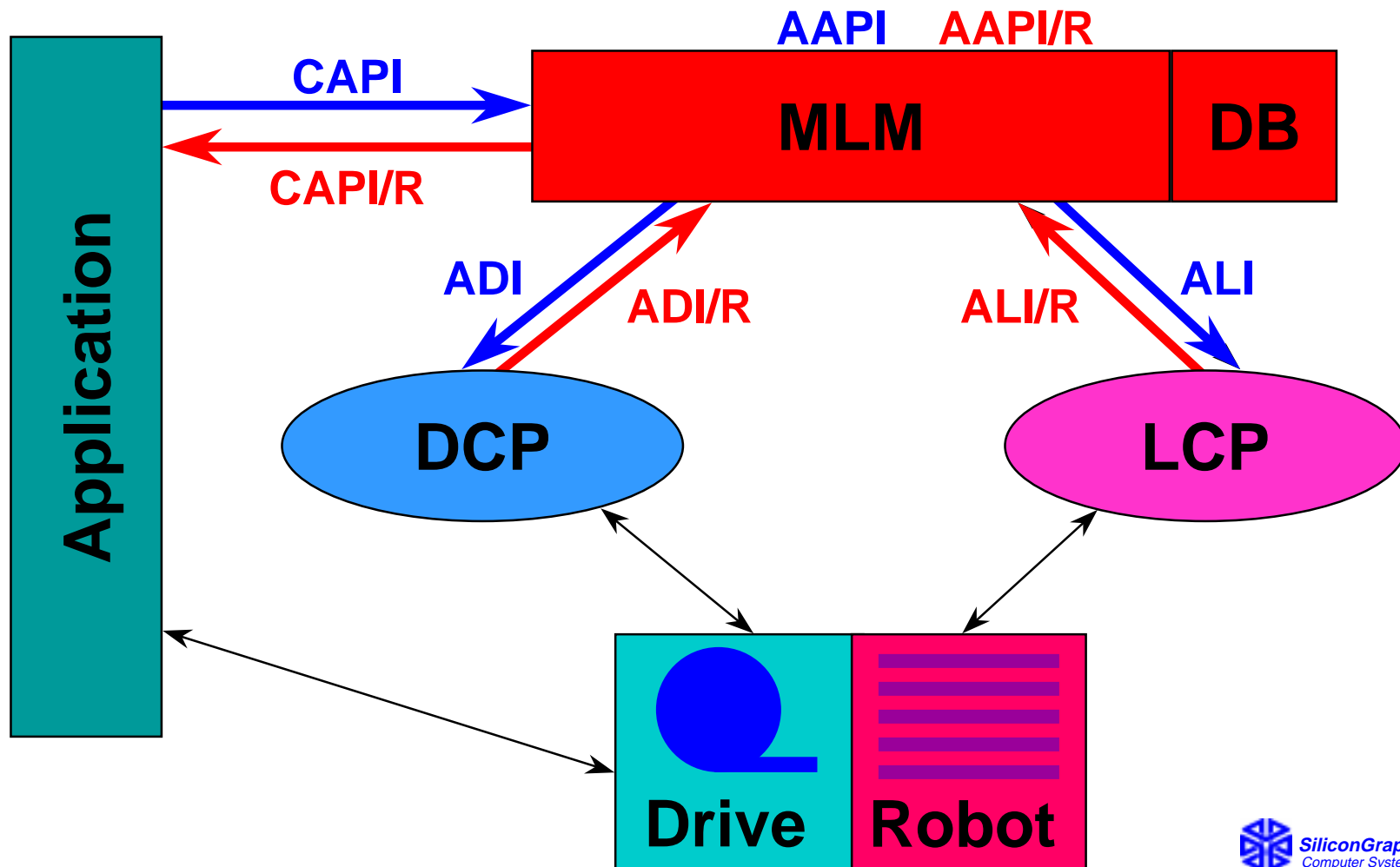
- **CAPI** — Client Application Programming Interface
- **AAPI** — Administrative Application Programming Interface
- **ALI** — Abstract Library Interface
- **ADI** — Abstract Drive Interface



# Components & Interfaces



# Request/Response Interfaces





# OpenVault Interface Design

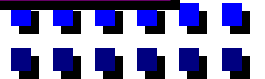
- Interfaces are simple ASCII, not RPC
- Platform neutrality is key
- Command/response and asynchronous
- Digital signatures for over-the-wire security
- Interface specifications will be made public
- Versioning permits future enhancement with full compatibility



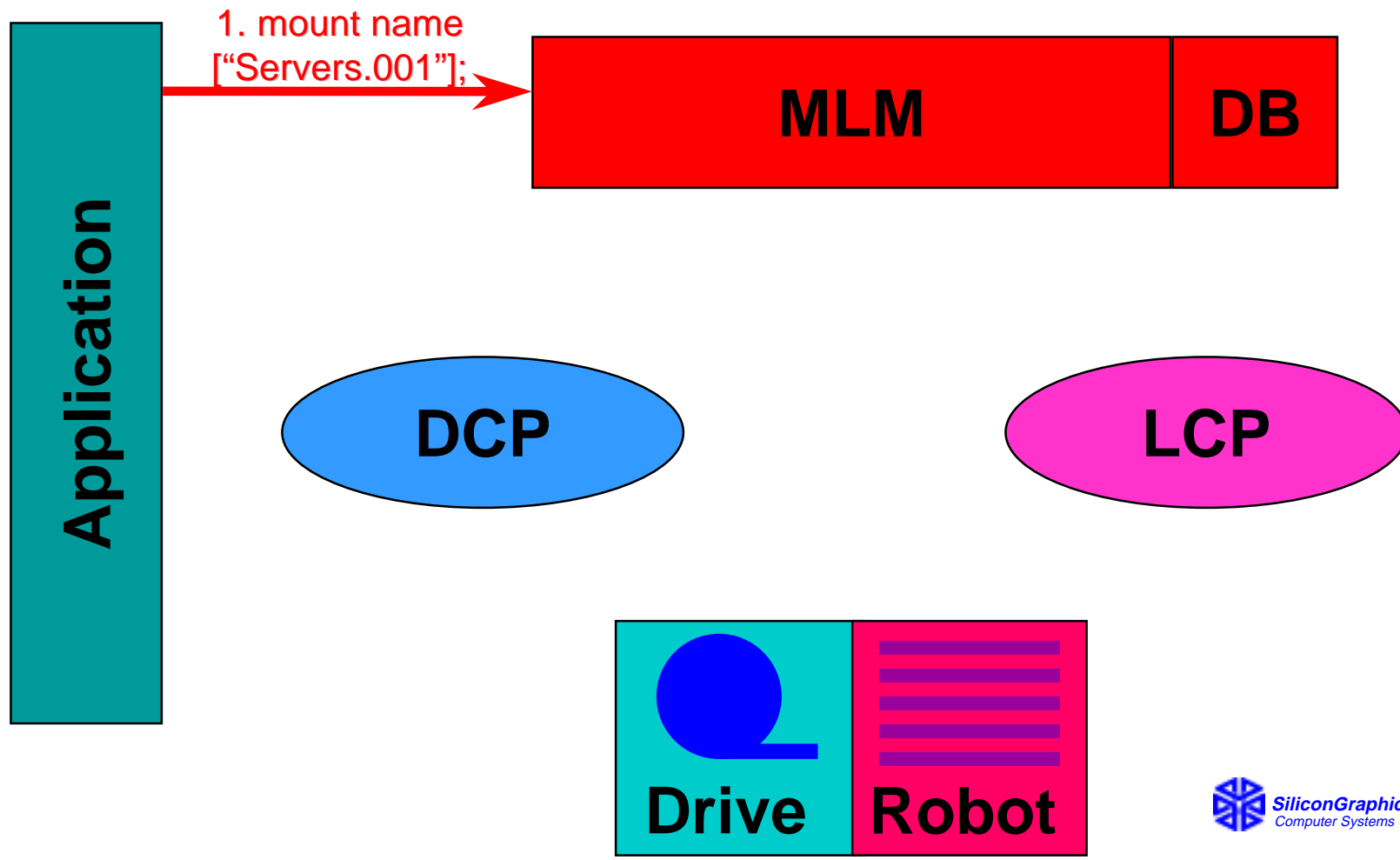


# Why ASCII Interfaces?

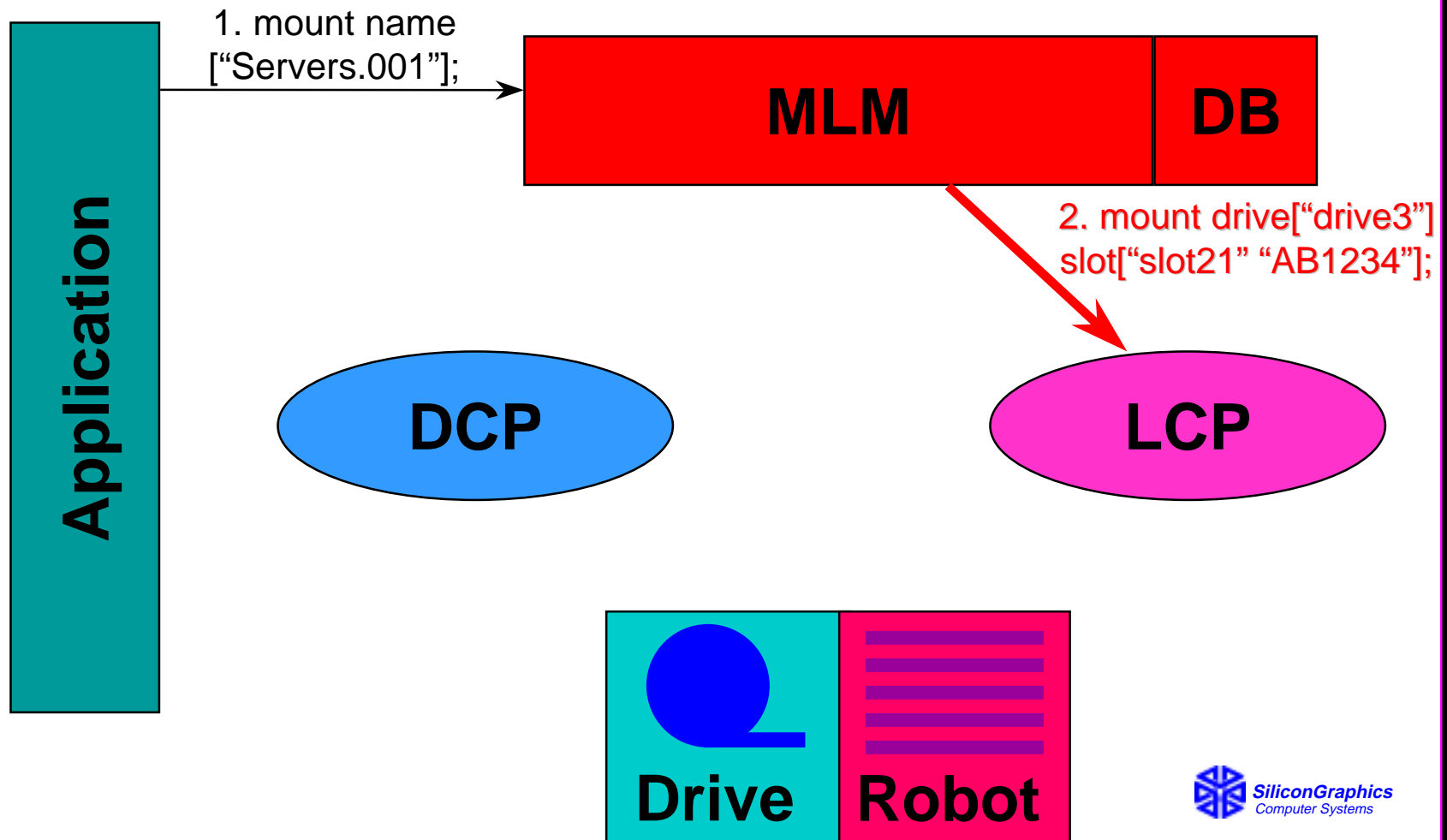
- Minimize dependence on RPC software
  - ◆ No licensing costs
  - ◆ No package requirement when porting to a new platform
- Easy to debug
- Follows successful Internet paradigm
  - ◆ *ftp, telnet, smtp, nntp, http, etc.*
- Easy to interface with scripting languages (*perl, etc.*)
- Easy to extend
- Relatively low transaction rates — efficiency not paramount



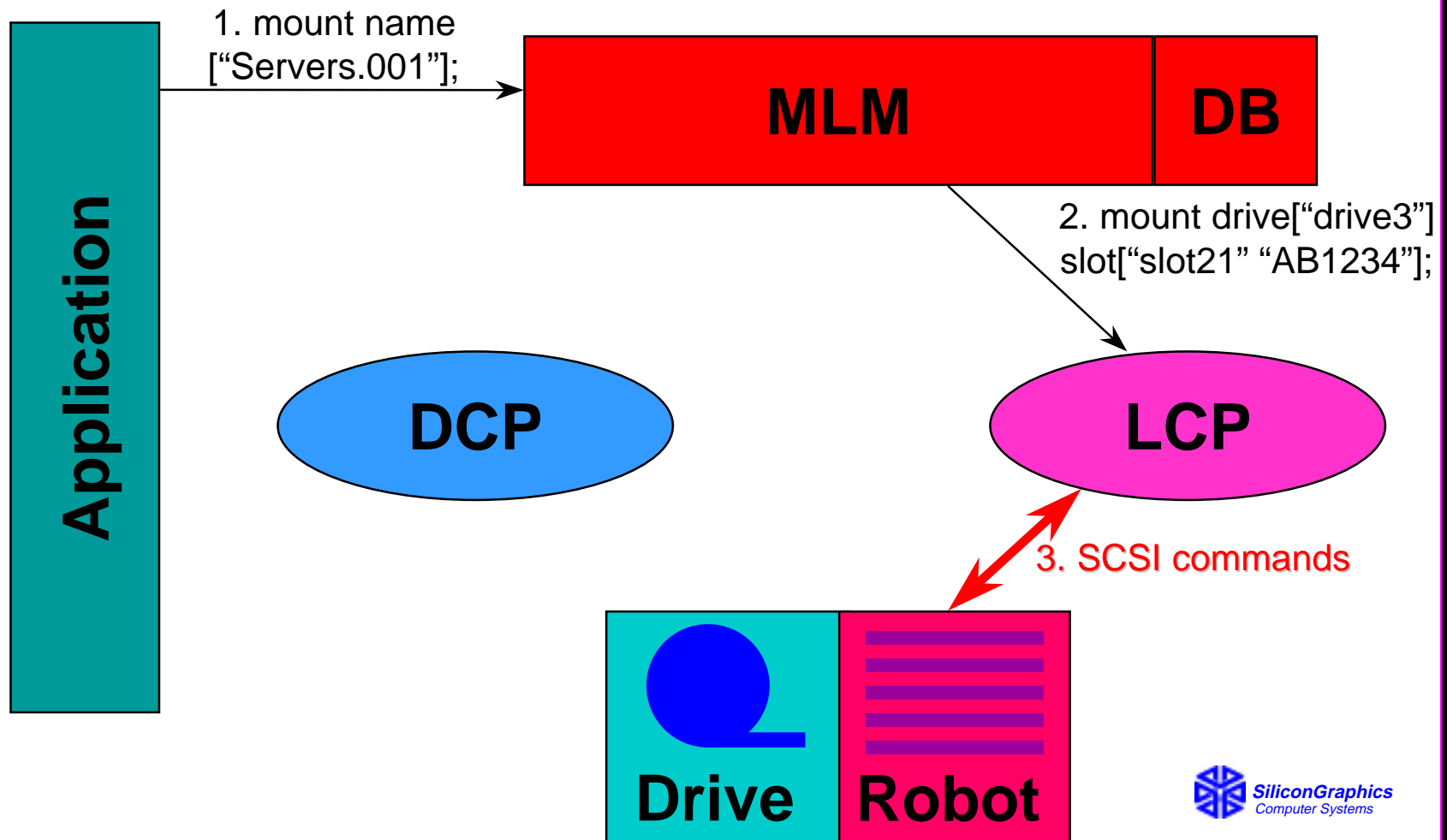
# OpenVault Operation (1)



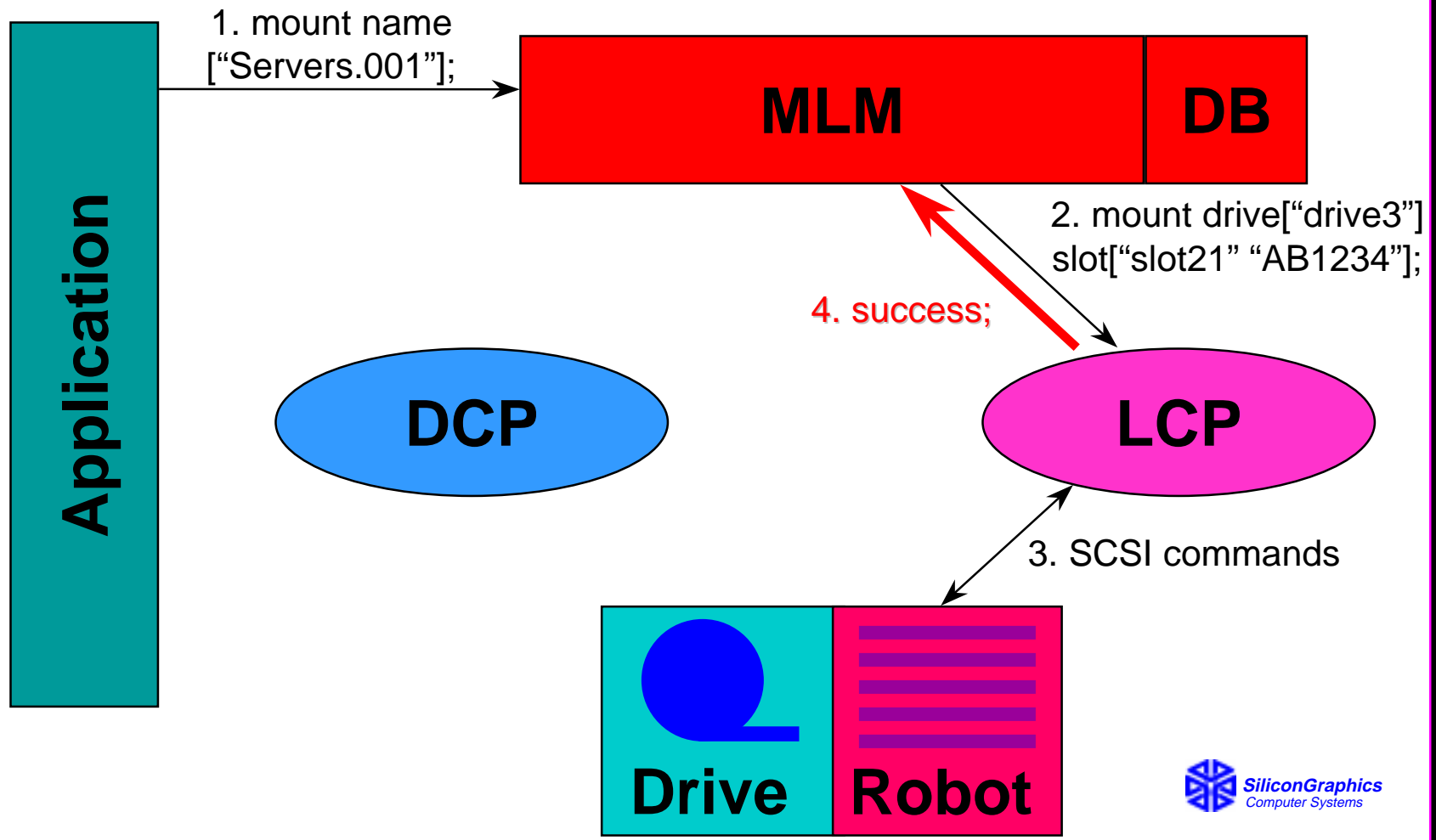
# OpenVault Operation (2)



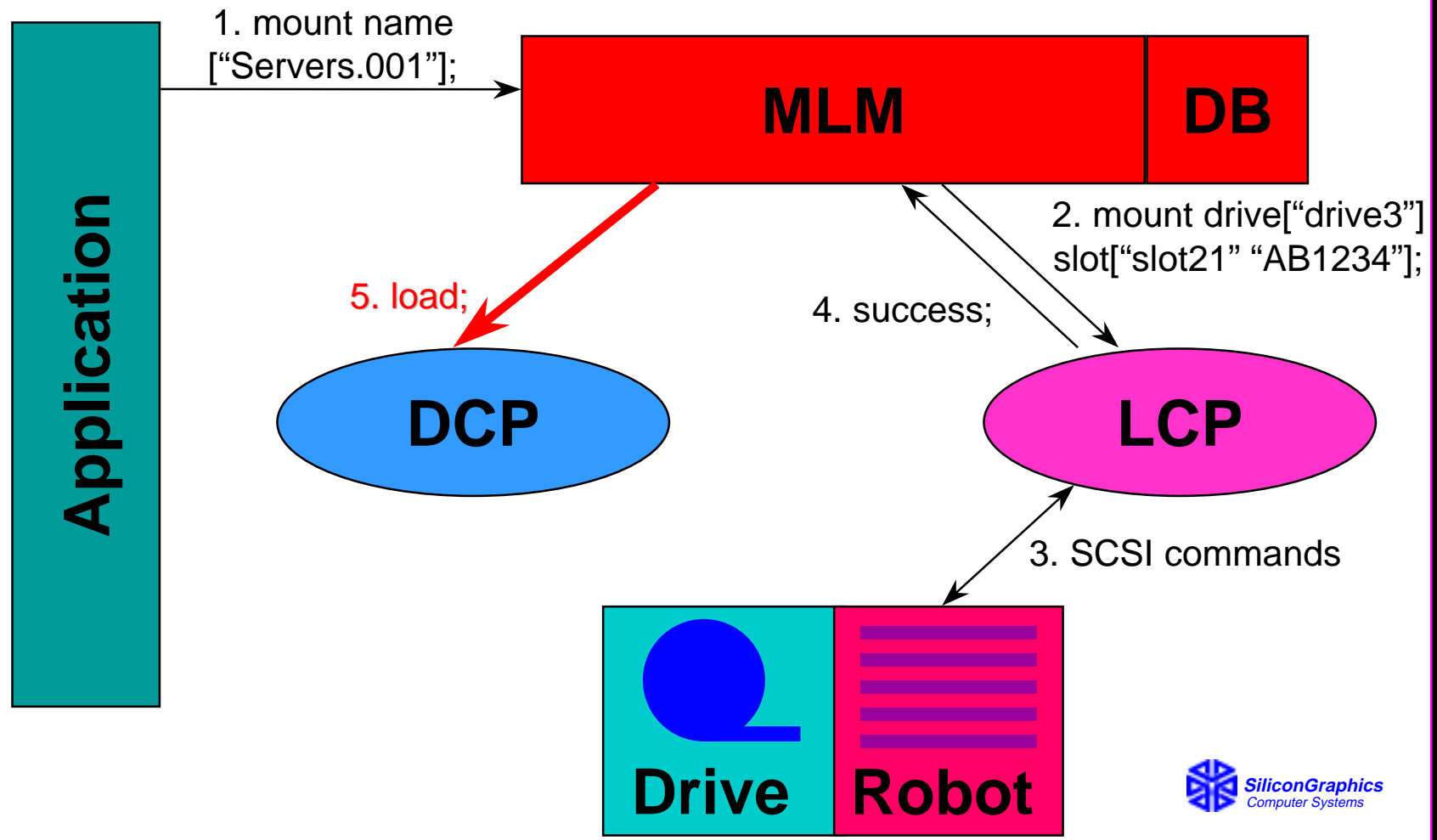
# OpenVault Operation (3)



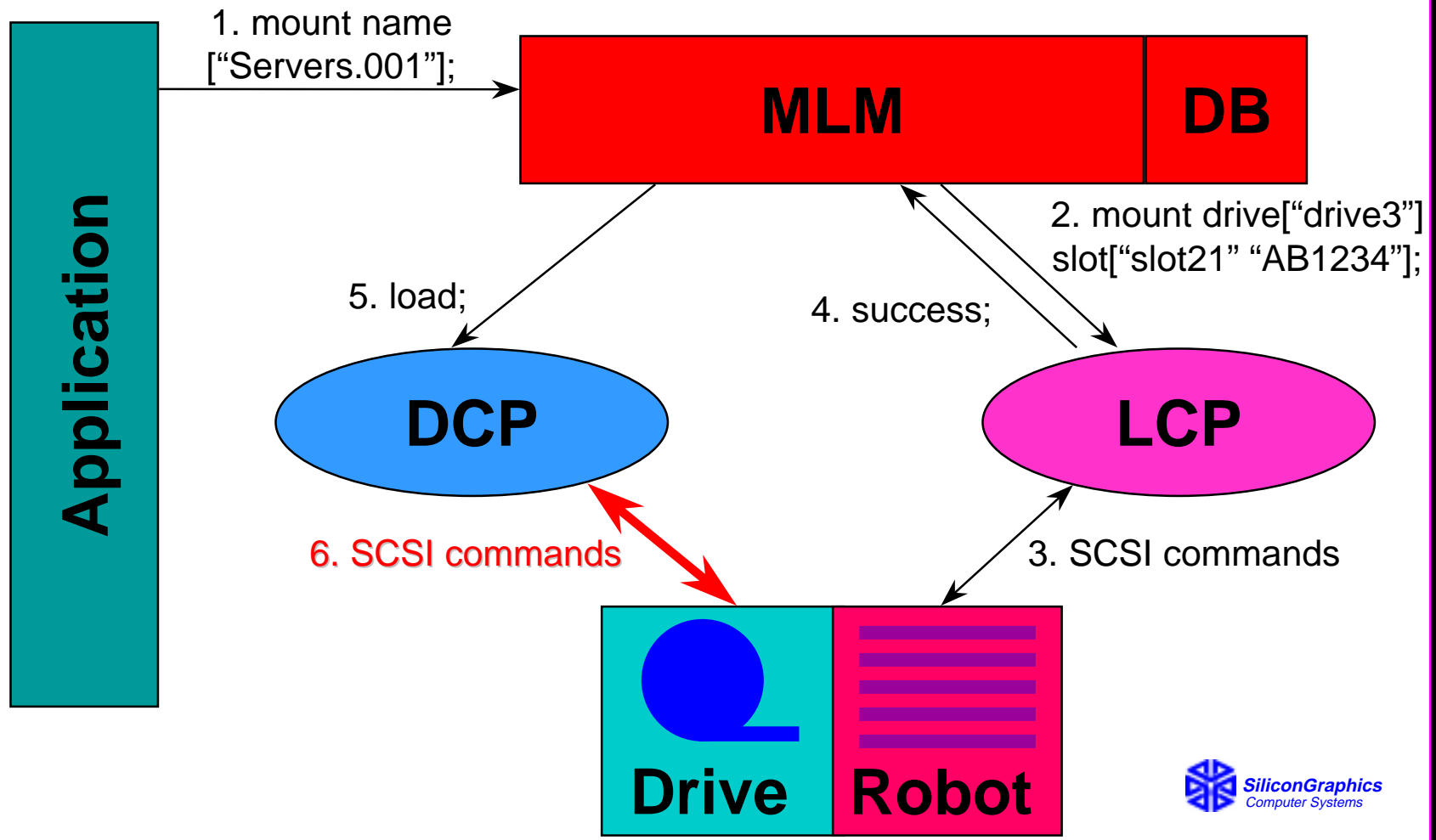
# OpenVault Operation (4)



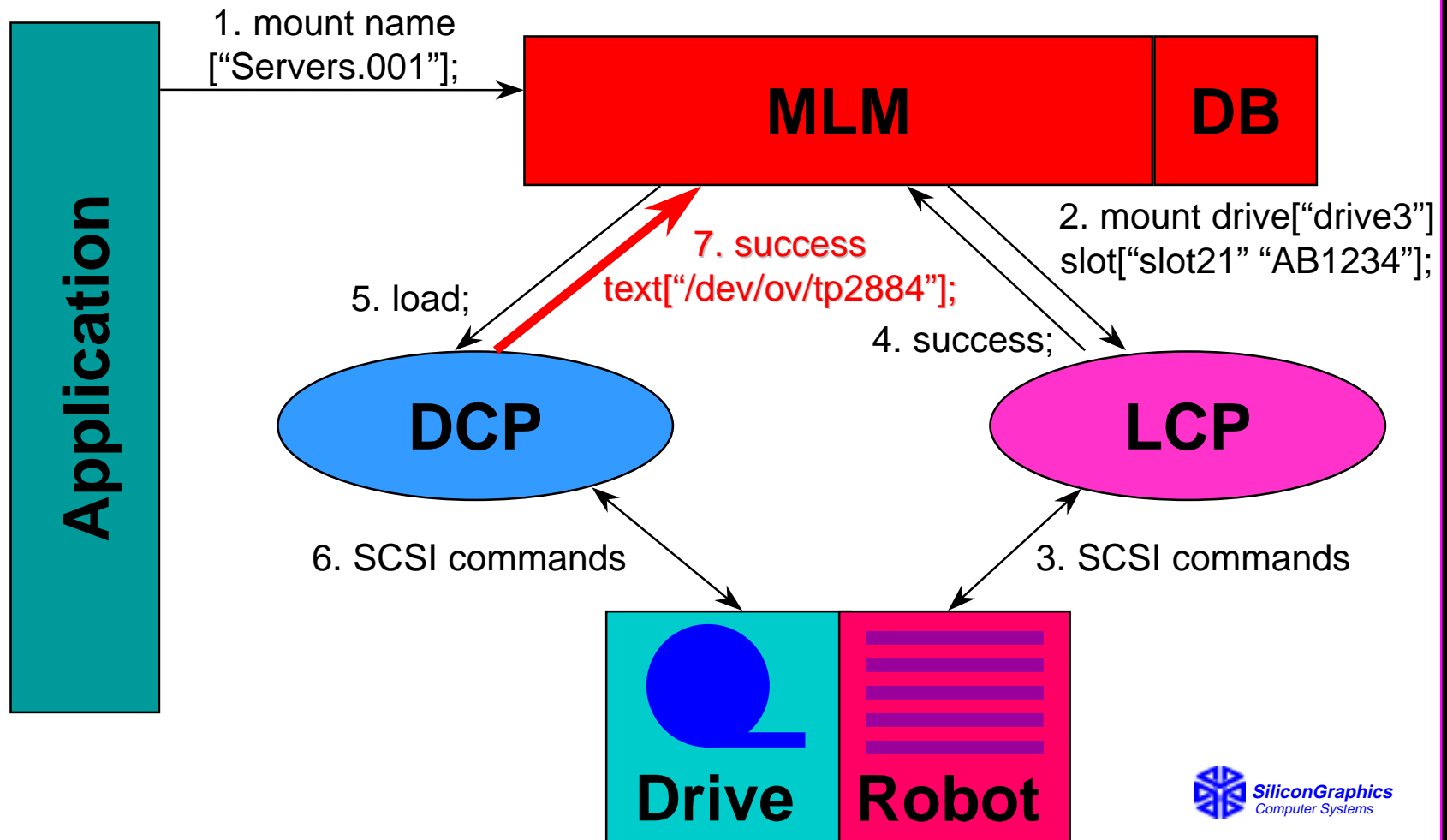
# OpenVault Operation (5)



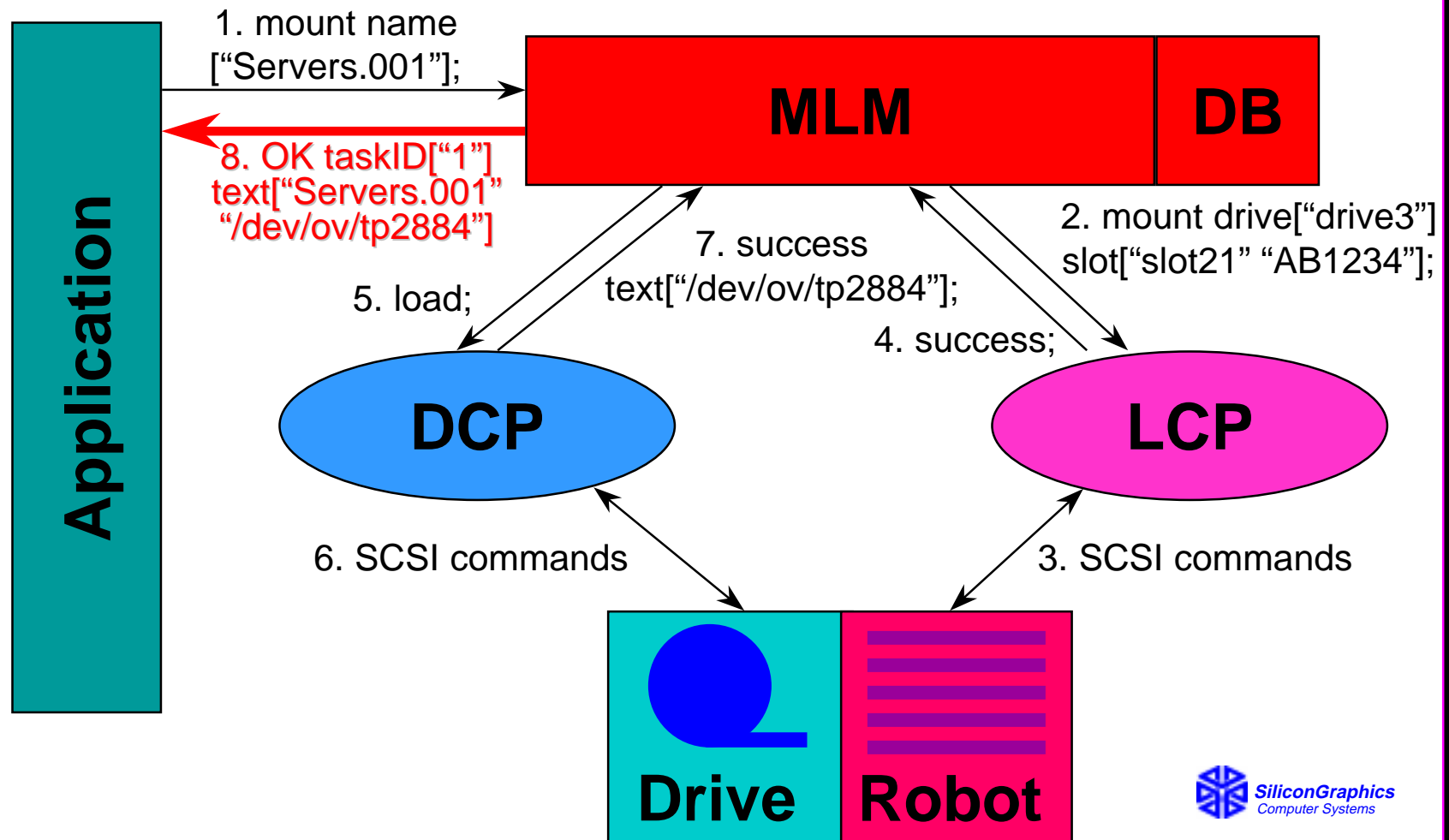
# OpenVault Operation (6)



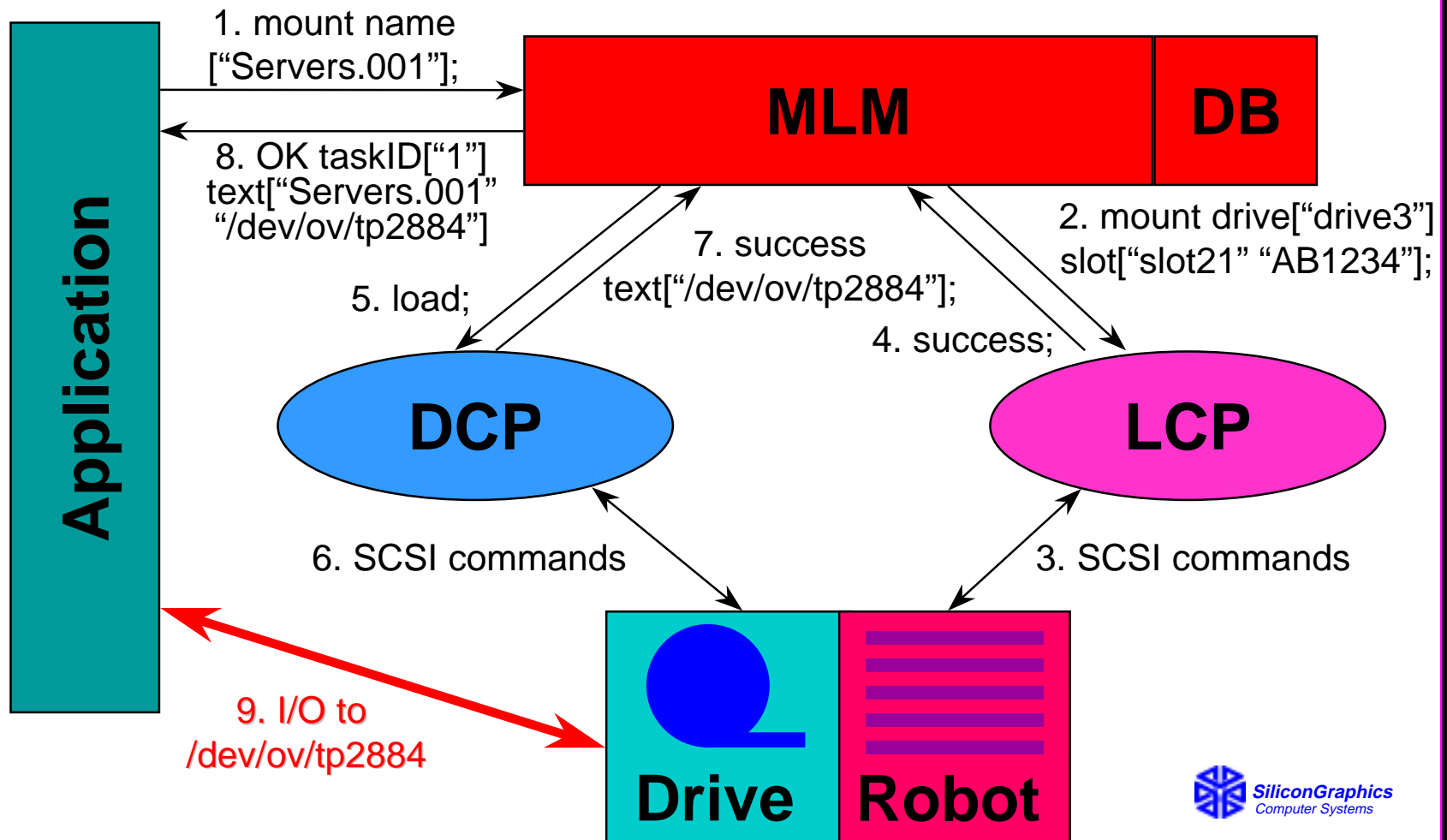
# OpenVault Operation (7)



# OpenVault Operation (8)



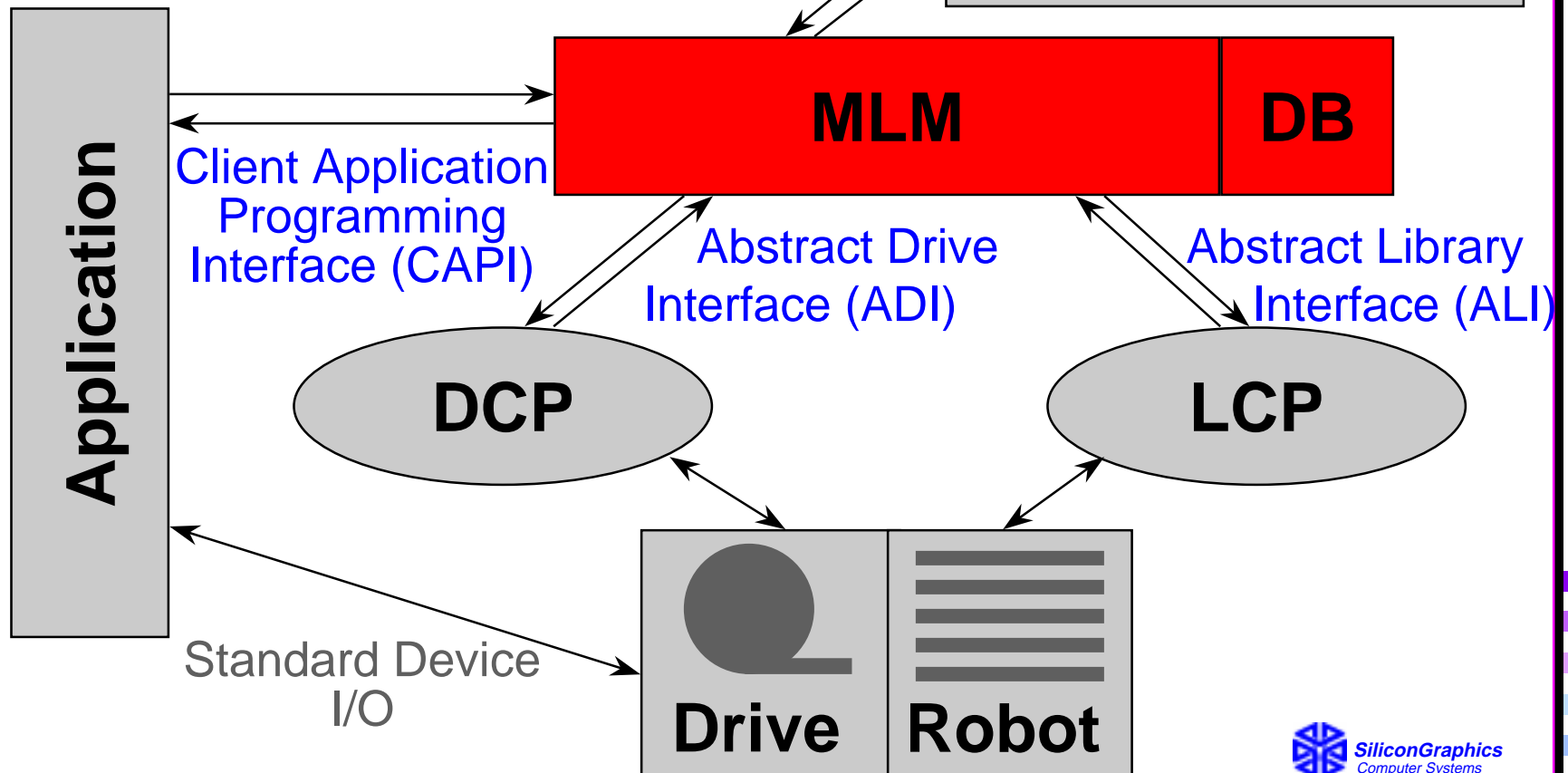
# OpenVault Operation (9)



# Media Library Manager

Administrative Application  
Programming Interface (AAPI)

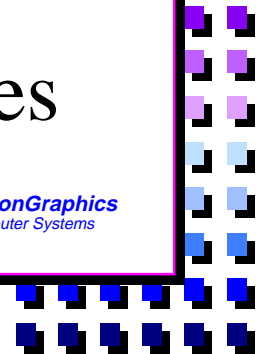
Admin App





# Media Library Manager (MLM)

- Maps volume to a cartridge
  - ◆ (detail: to a partition on a side of a cartridge)
- Selects and schedules resources
- Persistently stores *all* state
- Implements atomic transactions
- Has internal, procedural database interface
- Speaks CAPI, ALI, ADI, AAPI languages





# Interface “Handshakes”

- `hello language [“CAPI”]  
version [“1.0” “2.0”]  
client [“myprogram”]  
instance [“voyager”];`
- `welcome version [“2.0”];`





# CAPI: Logical Access

- `allocate name [ "myVolume-003" ]  
match [strEq  
 (CARTRIDGE."MediaType"  
 "dlt2000") ];`
- `deallocate name  
[ "myVolume-007" ];`



# CAPI: Physical Access

- `mount mountMode ["readWrite"] name ["myVolume-003"];`
- `mount mountMode ["readWrite"] number ["1"] match [and (strEq (CARTRIDGE."MediaType" "dlt2000") numLe (VOLUME."pctFull" "60")))];`
- `unmount name ["myVolume-003"];`



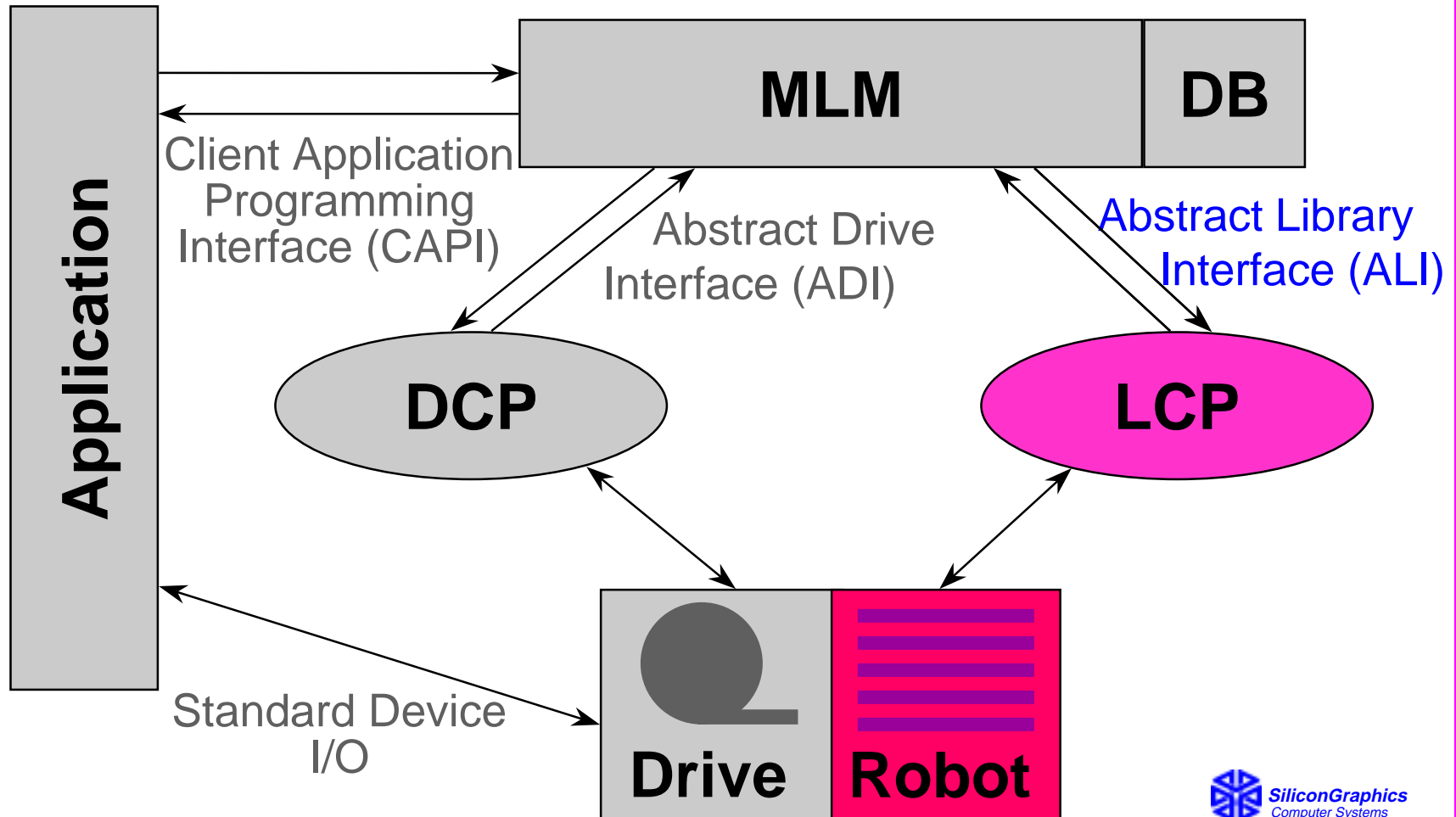


# CAPI: Database

- `attribute name ["myVolume"] set [nameValue ["subGroup" "server backups"]];`
- `attribute match [and ( numEq (VOLUME."pctFull" "0") strEq (VOLUME."Pool" "server backups")) ] set [nameValue ["Pool" "free"]];`



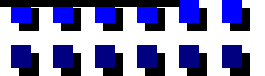
# Library Control





# Library Control Program (LCP)

- Provides abstraction of library for MLM
- Translates Abstract Library Interface (ALI) into library device-specific control language
- Unique binary for each library make/model
  - ◆ May group similar devices into one LCP
  - ◆ Different binaries for different host platforms
- One LCP instantiation per library
- One LCP for multiple “joined” robots (bays)



# ALI: Mount

- mount

```
slot["bay 2, slot 12" "AB1234"]  
drive["dlt1"];
```

- mount

```
slot["bay 2, slot 12" "AB1234"]  
slot["bay 1, slot 29" "XY9876"]  
drive["dlt1"];
```

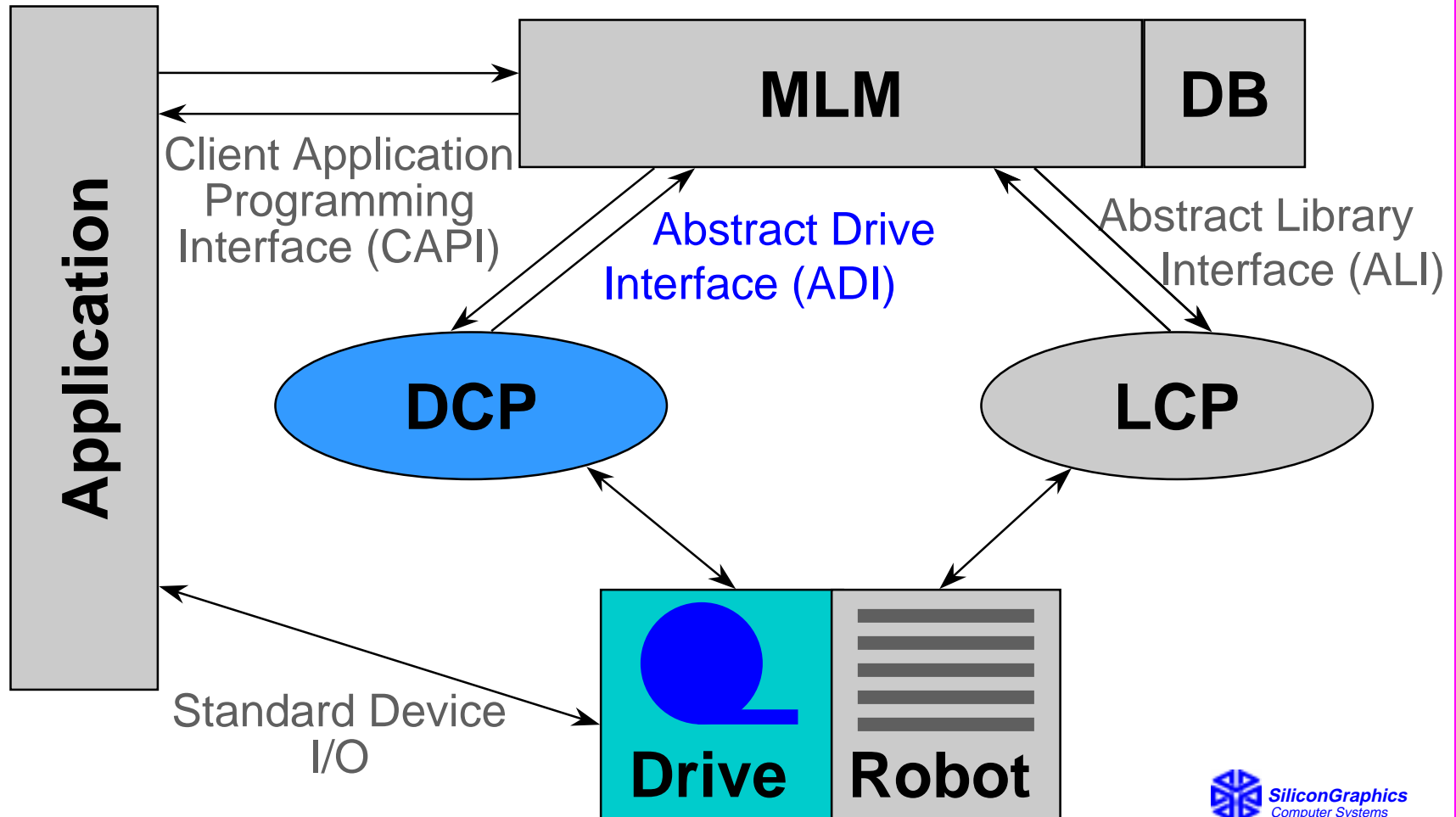


# ALI: Unmount, Move

- `unmount drive["DLT1"]  
slot["bay 1, slot 43"  
"XY9876"];`
- `move from["bay 4, slot 74"]  
to ["bay 2, slot 93"];`



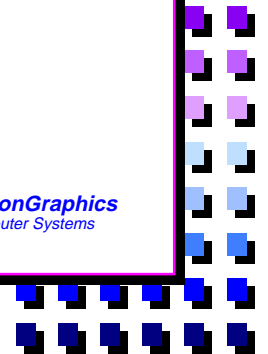
# Drive Control





# Drive Control Program (DCP)

- Provides abstraction of drive control for MLM
- Translates Abstract Drive Interface (ADI) into drive-specific control commands
  - ◆ Media/drive information, e.g. access, errors, statistics, etc.
  - ◆ Configure drive settings, e.g. compression, auto-rewind, etc.
  - ◆ Control drive load, unload, position operations
- Unique binary for each drive make/model
- One DCP instantiation per drive
- Mount verification
- Optionally create device nodes (protection)



# ADI: Start Up

- `attach modeName[ "base" ] ;`
- `success`  
`text [ "/dev/mlm/tpsc38a7b6" ] ;`
- `attach modeName[ "nrv" ]`  
`driveHandle [ "/udev/myapp/tp3" ] ;`
- `success text [ "/udev/myapp/tp3" ] ;`
- `load ;`

# ADI: Finish Up

- `unload;`
- `detach driveHandle`  
`[ "/dev/mlm/tpsc38a7b6" ];`



# OpenVault Developer Kits

- *Application* — CAPI & AAPI specifications, sample source, documentation, man pages, and license agreement
- *Infrastructure* — ALI/ADI specifications, sample LCP and DCP source, documentation, test suites, man pages, and license agreement
- *Server OEM* — MLM source and license agreement





# OpenVault Application

- Native client “speaks” and understands OpenVault CAPI
- Full OpenVault feature set available
- Application has flexible resource view as provided by OpenVault
- Existing applications with media databases:
  - ◆ use OpenVault to replace app media database
  - ◆ have app & OV databases work in parallel





# OpenVault Release Plan

- Specifications available now — in handout
- Initial (alpha) developer kits available Q1 '97:
  - ◆ Infrastructure, Application, and Server kits
  - ◆ Partners can begin development
  - ◆ *SGI needs feedback on language, interfaces, etc.*
- SGI end-user (binary) release: Q2 CY97





# Associated SGI Products

- Release with OpenVault in Q2 CY97:
  - ◆ User mount application
  - ◆ xfsdump
- Release to follow OpenVault:
  - \$ IRIX NetWorker™ release 4.2.6/OV
  - \$ Cray DMF™ (Data Migration Facility) rel. 2.6
  - \$ UNICOS™ (Cray) tape subsystem





# SGI Planned Library Support

- Ampex DST-410 Stacker, 810\* Robot w/DST-310 drive
- ✓ Breece Hill Q2/15, Q7, Q47
- + EMASS Grau family: AML/S, AML/J\*, AML/E, ...
- ❖ Exabyte family: 10e†, 10h†, 10i†, 210, 480, ...
- HP C1553A†
- + IBM 3494 with 3590 (MagStar) drives
- ✓ Odetics family: 4/52\*, 2640 series, ...
- ❖ Quantum† DLT2500XT, DLT2700, DLT4500, DLT4700
- SpectraLogic SL-4000, SL-9000
- ✓ STK 9711, 9704
- + STK 9710 /9714 (DLT), Wolfcreek (Timberline, Redwood), Powderhorn\* (Timberline, Redwood drives)

\* = not in SGI lab; †=non-barcode, may not be supported in 1.0





# OpenVault Benefits

- Integrating Applications with OpenVault
  - ◆ More robotic coverage
  - ◆ Separate releases + parallel engineering effort
  - ◆ OEM flexibility
    - ▼ Sell or include OpenVault and/or specific components (DCP, LCP) with OEM's package
  - ◆ Enables sharing of devices between applications





# Programming Interfaces & Design

- Language overview
- Hello protocol
- Client API
- Abstract Library Interface
- Abstract Drive Interface
- Administrative API
- Data Model / Database Content





# Language Style

- Common style and syntax elements in all languages
- Languages designed for easy parsing, not for optimal human readability
- Simple 7-bit ASCII only
- Non-positional, except that command verb must be first
- Quoted, uninterpreted strings are the only data type



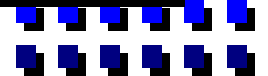
# TaskIDs

- Sender of a command assigns a unique *taskID* for each command
- All responses and queries related to that command must reference that *taskID*
- *TaskIDs* permit connecting each response with the command to which it relates
- *TaskIDs* are common to all languages



# Command Phasing

- Exchanges between any sender and recipient in all languages always happen in 3 phases
  - ◆ *Command*: command (with taskID) is sent
  - ◆ *Ack*: receiver acknowledges receipt of task
  - ◆ *Data*: receiver sends response and references the taskID
- Between *command* and *ack*, sender cannot send another command; simple flow control





# Resynchronization After

## LCP / DCP/ Application Failover

- MLM remembers taskIDs from each client for recent commands that have completed
- TaskIDs are generated by sender, rather than by the receiver, so sender knows them all
- After failover, sender resends all unfinished commands
- MLM avoids re-execution of DB modification commands it has already completed





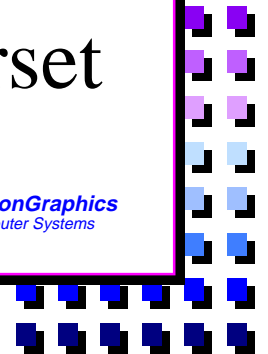
# HELLO Protocol

- Client session initiation with MLM
- Identifies language and language version to be used
- Identifies client application and instance
- Handles client authentication



# HELLO Rationale

- Single security model: clients always initiate a session
- Single entry point to the OpenVault System
- Ability to change languages or add new ones
- Allows changing all but the HELLO language, even character set used (UNICODE)
- Versions are exact match: no subset/superset is implied, eg: 2.0 and 1.0 not related





# HELLO Operation

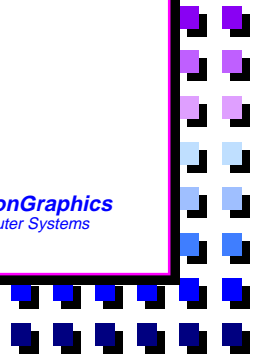
- Client contacts MLM on well-known TCP socket
- Sends *hello* string declaring who it is and what language it is willing to speak
- MLM verifies authentication information
- MLM starts handler which returns *welcome* and language version to be used
- Client and handler now send/receive in the language version agreed upon





# HELLO Errors

- Authentication failure results in immediate dropped connection
- Other errors return *unwelcome* with an error message, then connection is dropped
  - ◆ Errors include unknown language, unsupported language versions, missing handler, etc.





# HELLO Examples

```
hello language["CAPI"] versions["1" "2"]
  client["MyApp"] instance["abel"];
welcome version["2"];
hello language["ALI"] versions["2" "3"]
  client["alexandria"]
  instance["bigserver"];
welcome version["3"];
hello language["COBOL"];
unwelcome error["EBADLANG"]
  text["Unrecognized language name"];
```





# HELLO Language Pseudo-BNF

```
"hello" helloArgs
```

```
helloArgs:
```

```
"language"  "[ " STRING "]" |  
"versions"  "[ " STRING ... "]" |  
"client"    "[ " STRING "]" |  
"instance"  "[ " STRING "]" |  
/* NULL */
```





# HELLO/R Language Pseudo-BNF

```
"welcome" "version" "[" STRING "]"
```

```
"unwelcome" unwelcomeArgs
```

```
unwelcomeArgs:
```

```
  "error" "[" STRING "]" |
```

```
  "text"  "[" STRING "]" |
```

```
  /* NULL */
```



# HELLO Client Name

- For CAPI/AAPI clients, shows the name of the application connecting
- For ALI/ADI clients, shows the name of the device being controlled
- Determines the *parent* private attribute space to connect to



# HELLO Instance Name

- For all clients, shows which copy of the application/LCP/DCP is connecting
- Supports dual-ported drives, multi-homed applications, etc.
- Determines the *self* private attribute space to connect to



# CAPI Object Model

- Object directly managed by CAPI is *volume*
- *volume* is mapping to a *partition* on a *side* of a *cartridge*
- *volumes* are named by the application at *allocate* time



# CAPI: Basic Use

- CAPI provides a *volume* mounting service
- The *allocate* command moves a *partition* from available state to used state. A *volume* is created, named, and made to point to *partition*
- The *mount* command mounts a *volume* is mounted on an appropriate device, and returns the name of the *volume* mounted and a *drive* access handle to it
- *unmount* and *deallocate* are left as an exercise for reader





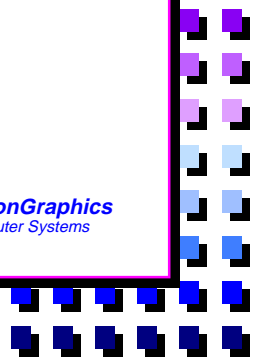
# CAPI Commands (1)

- *allocate* — bind a *volume* to a *partition*, make *volume* usable by *application*
- *deallocate* — destroy binding of *volume* to *partition* , destroy *volume*
- *mount* — bind *volume* to *drive*, making media accessible to *application*



# CAPI Commands (2)

- *unmount* — unbind *volume* from *drive*, making media inaccessible
- *attribute* — bind new attribute/value pair to *volume* meta-data store
- *show* — query entire meta-data store for match, display results
- *rename* — change the name of a *volume*





## CAPI commands (3)

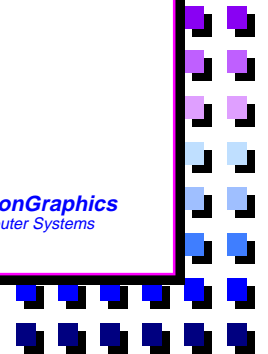
- *reject* — tell OpenVault that *volume* given by mount is wrong *volume*
- *cancel* — cancel outstanding CAPI command
- *goodbye* — end communication, and end logical CAPI session
- *attach* — re-attach to old (detached) CAPI session (not in R1)
- *detach* — end communication for now, leave logical CAPI session intact





# CAPI/R commands

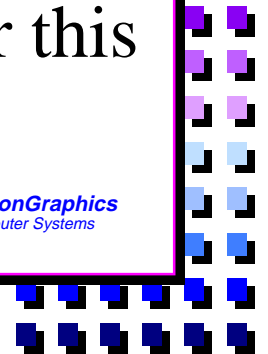
- In general, CAPI clients must wait for an **ACCEPTED** or **UNACCEPTABLE** response before sending another command; they need not wait for the final disposition of the command
- **ACCEPTED** — command has been accepted by OpenVault
- **UNACCEPTABLE** — command has not been accepted by OpenVault; explanation included
- **OK** — task completed successfully; return value(s) enclosed
- **ERROR** — task completed unsuccessfully; explanation enclosed





# CAPI Complex Command Example 1

- Mount in read/write mode one *volume*
  - ◆ which is on a *cartridge* of media type DLT2000,
  - ◆ which resides in *library* named Widener,
  - ◆ onto a drive of type DLT4000 or DLT7000.
- Report the name of the *volume*,
  - ◆ the bar-code of the *cartridge*,
  - ◆ and the name, type and handle of the *drive* for this mount





# CAPI Complex Command Example 1

```
mount match[and(  
  strEQ(CARTRIDGE."CartridgeMediaType" "DLT2000")  
  strEQ(LIBRARY."LibraryName" "Wiedener")  
  regex(DRIVE."DriveModel" "DLT[47]000"))]  
report[VOLUME."VolumeName" CARTRIDGE."CartridgePCL"  
  DRIVE."DriveName" DRIVE."DriveModel"  
  MOUNT."DriveHandle"];  
ACCEPTED taskID["21"];  
OK taskID["21"] text["Servers.001" "AB1234" "Fred"  
  "DLT7000" "/dev/ov/asdf21"];
```





# CAPI Complex Command Example 2

- Show names of all *library/slot* combinations used by this *application's volumes* in Exabyte model 440 or 480 *libraries*



# CAPI Complex Command Example 2

```
show
  taskID["showslottask"]
match [and(
  isAttr(VOLUME."VolumeName")
  or(strEQ(LIBRARY."LibraryModel" "EXB-440")
  strEQ(LIBRARY."LibraryModel" "EXB-480")))]
report [LIBRARY."LibraryName" SLOT."SlotName"];
ACCEPTED taskID["showslottask"];
OK taskID["showslottask"] text["Wiedener" "slot1"]
text["Wiedener" "slot12"] text["LittleLibrary"
  "fixedslot"];
```



# CAPI C-Language Example

```
main(){
  capi_handle_t *pH;
  capi_result_t *pR;

  pH = capi_open ("myProgram", "onlyInstance", "ovhost",
                 0, "1.0", 0);
  if (pH->status == CAPI_OK) {
    pR = capi_send_receive (pH,
                           "mount name['Servers.001'] report[DRIVE.'DriveHandle']");
    if (pR->status == CAPI_OK) {
      printf("Volume Servers.001 mounted: drivehandle=%s\n",
            capi_result_field(capi_result_row(pR, 0), 0));
    }
    capi_close (pH);
  }
}
```



```

main(){
    capi_handle_t *pH;
    capi_result_t *pR1, *pR2;

    pH = capi_open ("myProgram", "onlyInstance", "ovhost", 0, "1.0", 0);
    if (pH->status == CAPI_OK) {
        pR1 = capi_send (pH,
            "mount name['Servers.001'] report[DRIVE.'DriveHandle']");
        if (pR1->status == CAPI_PENDING) {
            pR2 = capi_send (pH, "mount name['Servers.002'] "
                "report[DRIVE.'DriveHandle']");
            if (pR2->status == CAPI_PENDING) {
                while (pR1->status == CAPI_PENDING ||
                    pR2->status == CAPI_PENDING) {
                    if (pR1->status == CAPI_PENDING &&
                        capi_receive (pH, pR1) == CAPI_OK)
                        printf ("Volume Servers.001 mounted: drivehandle=%s\n",
                            capi_result_field(capi_result_row(pR1, 0), 0));
                    if (pR2->status == CAPI_PENDING &&
                        capi_receive (pH, pR2) == CAPI_OK)
                        printf ("Volume Servers.002 mounted: drivehandle=%s\n",
                            capi_result_field(capi_result_row(pR2, 0), 0));
                }
            }
        }
        capi_close (pH);
    }
}

```





# Abstract Library Interface (ALI)

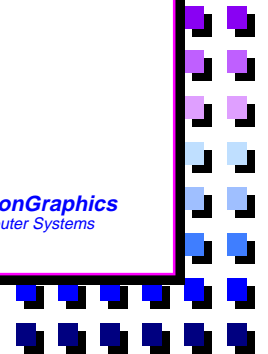
- Manages the configuration of a library
- Abstracts library control tasks associated with mount, unmount, and cartridge movement requests





# ALI Goals

- Independence from library make/model for application and MLM
- MLM not required to know topology of library
- Permit hardware-centric administration
- Allow LCP/library to optimize operations
- Let LCP be authoritative source for all information about library





# Library Control Program (LCP)

- Process that implements ALI semantics
- Translates between ALI and real library control interface
- Handles any idiosyncrasies of managing a specific library on a specific platform
- No local persistent storage; persistent storage in MLM if required by LCP



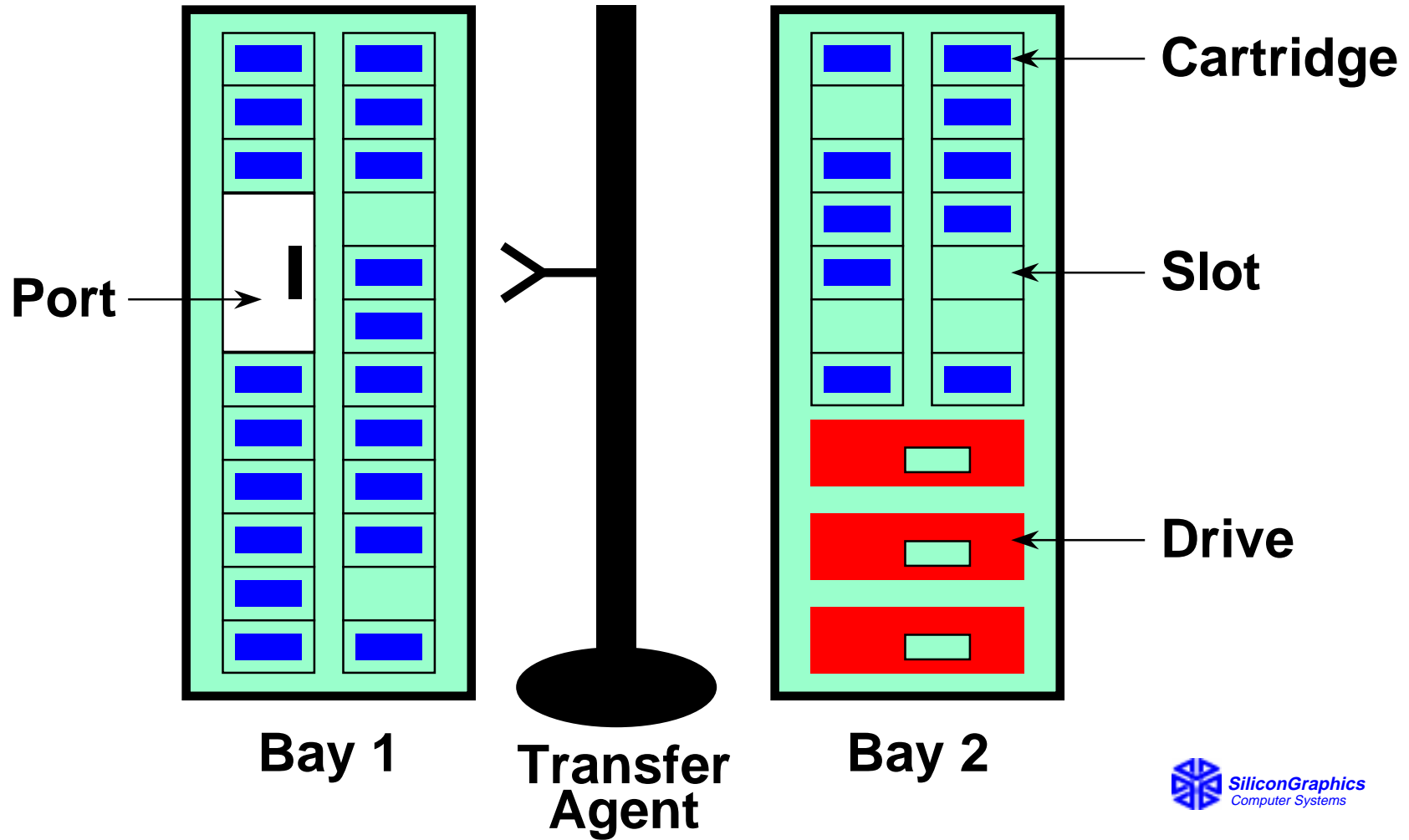


# Abstract Library Goals (1)

- ALI defines an abstract library and its controls
- Makes few/no assumptions about capabilities of the library hardware
- Underlying physical library may be automated or human-operated



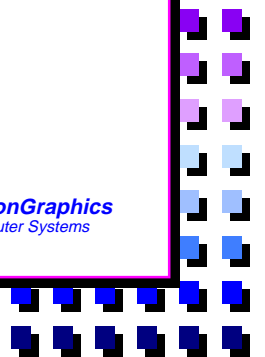
# Abstract Library





# Library Components (1)

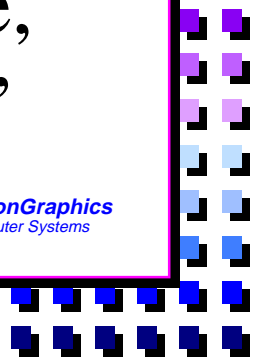
- *Cartridge* — houses media, lives in a slot, has a unique name
- *Transfer Agent* — mechanical or protein robot, implicitly used
- *Drive* — provides read/write access to media contained in a library
- *Port* — used to inject/eject cartridges, implicit support only





## Library Components (2)

- *slot* — stores a cartridge, has a unique name called a *slotID*, has a shape
- *bay* — group of slots/drives with some temporal locality, unique *bayID* for each
- *slotID* — human readable, LCP parseable, hardware-centric if possible, e.g., “slot 12”
- *bayID* — human readable, LCP parseable, hardware-centric if possible, e.g., “bay 2”





## Abstract Library Goals (2)

- Library topology details below the bay level are hidden from MLM
- *SlotIDs* and *BayIDs* are “magic cookies” — MLM only checks for equality.
- *SlotIDs* and *BayIDs* are intended to have physical meaning when read by a human
- *SlotIDs* and *BayIDs* are used by LCP on mount/unmount/move/eject





# Abstract Library Goals (3)

- PCLs (physical cartridge labels, barcodes) must be unique for a given cartridge form factor across an OpenVault installation
- *SlotID* may be a name (e.g., a PCL) rather than a location
  - ◆ used for libraries which hide physical locations, for example STK ACSLS-based libraries and IBM 3494





# LCP — MLM Relationship (1)

- Bi-directional communications using ALI and ALI/R language pair
- Multiple LCPs with the same client name define host/library connection topology — allows a library to be controlled by more than one host (not simultaneously)



# LCP — MLM Relationship (2)

- LCP is authoritative source for all library info
- MLM needs locally cached copy of configuration information to make decisions
- LCP is responsible for updating MLM's cache using *ALI/R config* cmd
- LCP knows when to update MLM based on hardware state changes

# LCP — MLM Relationship (3)

- LCP “advertises” library contents
- LCP provides *slotIDs* and *bayIDs*:  
human readable physical-based addressing
- *SlotID* and *bayID* are generated by LCP, and stored in MLM
- MLM only tests a *slotID* or *bayID* for equality with another *slotID* or *bayID*
- *SlotID* and *bayID* are recognized by LCP for move/mount operations



# ALI Commands (1)

- *mount* — mount one of a list of carts into a given drive
- *unmount* — unmount the cart in a given drive
- *move* — move a cart from one slot to another within a library



# ALI Commands (2)

- *eject* — push a cartridge out of the library or mark it to be pushed
- *openport* — either flush pending ejects or prepare to accept injects
- *scan* — resynchronize the LCP with the hardware, e.g., do a barcode scan



# ALI Commands (3)

- *barrier* — force all prior commands to complete before accepting new commands
- *activate* — start/stop talking to device — useful for dual ported devices
- *reset* — do what is required to force device through diagnostics, if possible.
- *exit* —LCP should clean up and exit



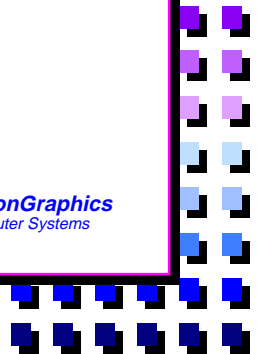
# ALI Commands (4)

- *attribute* — set/unset control parameters in the LCP
- *show* — return the value of the specified control parameters
- *cancel* — prevent execution of a command queued in the LCP, if possible



# ALI/R Commands (1)

- *config* — send new/changed library state/slotmap/etc. info to MLM
- *message* — send an error/status message to the log and/or operator
- *request* — ask the operator a question and expect to get a response
- *ready* — tell MLM the LCP is ready for commands, not-ready, or broken





# ALI/R Commands (2)

- *attribute* — set/unset persistent values in LCP private space in MLM
- *show* — return persistent values in LCP private space in MLM
- *cancel* — prevent execution of a command queued in MLM if possible

# Cartridge

- Houses physical media
- Main entity on which a library operates
- Many may reside within a library
- May be accepted into, removed from, and relocated within the library
- Has a unique name: *Physical Cartridge Label*
- PCL is human readable, may be machine readable



# Transfer Agent

- Means by which library moves cartridges
- A library may have one or more
- May be either mechanical like a robotic arm, or a human being
- Implicit in any ALI command that involves movement of a cartridge



# Drive

- Where carts may be mounted for media access
- A library may contain zero or more drives
- Each drive is in a library
- An ALI drive name is the same as the ADI drive name (required for rendezvous)
- Drive attributes include compatible cartridge form factor, bay, accessibility, and occupancy

# Slot (1)

- Location where a cartridge may be stowed inside of the library
- A library may contain many slots
- ALI exposes slots
- Slots are named, names determined by LCP
- Slot attributes include acceptable cartridge form factor, bay, accessibility, and occupancy



# Slot (2)

- Usually a slot is an addressable location within a library — the slot name corresponds to this address
- Not all libraries expose this level of topology info (e.g., STK ACSLS, IBM 3494), so PCL may be used as slot name



# *SlotID*

- Unique identifier for each slot in library
- Human-readable and LCP-recognizable
- Familiar: administrative interface can present hardware-centric view
- General: MLM does not need to know topology
- May also be a “name” (e.g., PCL) rather than a physical location (for example, ACSLS)





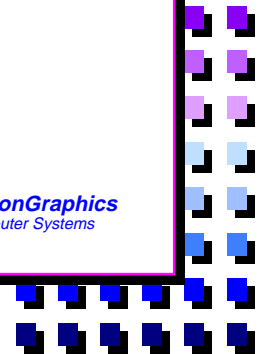
# Bay

- The housing for a physical grouping of slots and drives
- A bay is part of a library if any cartridge within that bay has an path to all other bays in the library



# *BayID*

- Unique identifier for each bay in the library
- Only reasonable measure of locality available, drive locality with respect to cartridges
- Human-readable and LCP-recognizable
- Familiarity: administrative interface can present hardware-centric view
- Generality: MLM does not need to know topology





# Port

- Physical door where cartridges may be introduced or removed (inject / eject)
- All libraries have a single, implicit port
- No name or explicit attributes
- LCP maps this notion onto the capabilities of the physical library



# Attributes

- Any other relevant info LCP wants to advertise
- Examples:
  - ◆ “nominal\_load” time may be used by MLM to pick a library
  - ◆ slot form factor “external\_form” tells admin/MLM which carts will fit in which slots



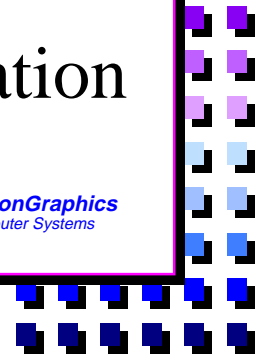
# Abstract Drive Interface (ADI)

- Manages the configuration of a drive
- Handles drive control tasks associated with mount and unmount requests



# ADI Goals

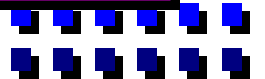
- Independence from OS and drive make/model for MLM
- Removable media drives on any platform
- Drives must be shareable between applications and between multiple host systems
- Drives may be connected
  - ◆ to multiple hosts
  - ◆ to hosts other than the MLM host
- DCP is authoritative source for all drive information
- Separate drive control and data





# Drive Control Program (DCP)

- Process that implements the ADI semantics
- Translates between ADI and real drive control interface
- Handles any idiosyncrasies of managing that drive on that platform
- No local persistent storage, possible local persistent storage in MLM





# Drive Abstraction (1)

- Defines an abstract drive and its “controls”
- Makes few/no assumptions about capabilities of the drive hardware
- Advertises the variability of a drive’s modes of access to the media



# Drive Abstraction (2)

- Only requires access to drive *control* path; access to the data path is not *required* (for example: a video tape drive)
- Uniform access to drive error info
- Knows “mount point” from “media access point”
- Supports sided and partitioned media



# Drive Abstraction (3)

- Sets up the access path that the application uses to read/write data
- Typically utilizes host operating system's unmodified I/O mechanism
- Currently does not support data access over general purpose networks



# DCP — MLM Relationship (1)

- Bi-directional communication using ADI and ADI/R language pair
- Multiple DCPs with the same client “name” define host/drive connection topology — multi-hosted drive support





# DCP — MLM Relationship (2)

- DCP is authoritative source for all drive info
- MLM utilizes a locally cached copy of configuration information to make decisions
- DCP is responsible for update of MLM's cache using *ADI/R config* command
- DCP knows when to update MLM based on hardware state changes



# DCP — MLM Relationship (3)

- DCP advertises drive capabilities and services
- DCP provides named groups of well-known capability strings
- Capability strings are tokens corresponding to drive features



# DCP — MLM Relationship (4)

- MLM compares application requested capability set(s) with DCP offered capability sets; matching sets are candidates for use
- DCP uses capability set name to condition drive and application access path (*/dev* node) on host into requested state



# ADI Commands (1)

- *attach* — create rendezvous point for application to get access to media
- *detach* — delete rendezvous point so application cannot continue access to media
- *load* — make media ready to be accessed
- *unload* — allow cartridge to be put away
- *position* — media to beginning of partition



# ADI Commands (2)

- *barrier* — force all prior commands to complete before accepting new commands
- *activate* — start or stop talking to device — useful for dual ported devices
- *reset* — force device through diagnostics, if possible
- *exit* — DCP cleans up and exits



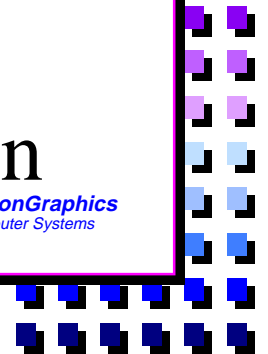
# ADI Commands (3)

- *attribute* — set/unset control parameters in the DCP
- *show* — return the value of the specified control parameters in the DCP
- *cancel* — if possible, prevent execution of a command queued in the DCP



# ADI/R Commands (1)

- *config* — send new/changed drive state/capabilities/etc. information to MLM
- *message* — send an error/status message to the log and/or operator
- *request* — ask the operator a question and expect to get a response
- *ready* — tell MLM the DCP is:  
ready for commands, not ready, or broken





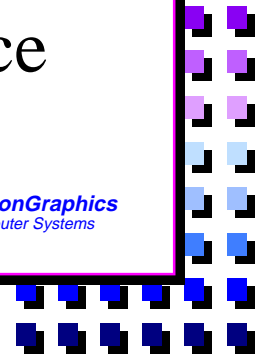
# ADI/R Commands (2)

- *attribute* — set/unset persistent values in DCP private space in MLM's metadata store
- *show* — return persistent values in DCP private space in MLM's metadata store
- *cancel* — if possible, prevent execution of a command queued in MLM



# Known Limitations/Problems

- One DCP per host-connection per drive won't work with direct network attached drives — too many idle DCPs
- Remote client access to media data not directly supported
  - ◆ Client application must be on same host as DCP
  - ◆ Note: UNIX *rmt* can be used for remote device access (with DCP, *rmtd* on same machine)





# Tag Name

- A unique-to-a-DCP name for a given set of capabilities and attributes
- Passed from DCP to MLM in *ADI config* command
- Passed back from MLM to DCP to identify which drive configuration to set up



# Capability String

- Agreed-upon names for particular drive behaviors
- Advertised by DCP, asked for by the application, stored by MLM
- MLM compares set of application-requested capabilities against DCP-offered sets of capabilities for a match



# External\_form

- External cartridge shape
- Eg: 8mm, DAT, DLT



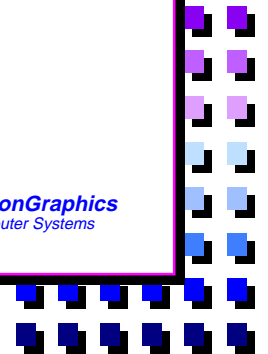
# Bit\_format

- Bit recording format(s) on the media the drive supports
- Eg: 3480, DLT2000, Exa8500-compression



# A-API: object model

- All objects known to OpenVault are visible to A-API
- Configuration, tasks, device enable / disable use, etc. are managed through A-API
- A-API client has full power to help or harm (i.e. root-equivalent privilege)





# CAPI versus AAPI views

- CAPI view of OpenVault table space is restricted:
  - ◆ CAPI clients can only see CARTRIDGEs onto which they might allocate VOLUMEs, or onto which they have already done so
  - ◆ CAPI clients can only see LIBRARYs in which there are DRIVEs which they might use
- AAPI client's view is *not* restricted — all AAPI clients see the same “full” view

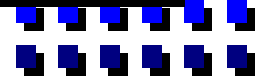




# AAPI commands (1)

AAPI has same command set as CAPI, with some additions:

- *create* — create a table entry — must declare all key fields
- *delete* — delete a table entry — must not be linked to other entries
- *export* — destroy all of OpenVault's knowledge of CARTRIDGEs — shorthand for a series of delete operations





# AAPI commands (2)

- *move* — move a CARTRIDGE from one SLOT to another in a LIBRARY
- *eject* — eject CARTRIDGEs from a LIBRARY
- *inject* — inject CARTRIDGEs into a LIBRARY



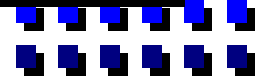
# AAPI commands (3)

- *accept* — take responsibility for a REQUEST
- *unaccept* — relinquish responsibility for a REQUEST
- *respond* — make a response to a REQUEST which you previously accepted



# AAPI/R commands

- AAPI clients must wait for an ACCEPTED or UNACCEPTABLE response before sending their next command
- Clients need not wait for final disposition of the command before proceeding
  - ◆ ACCEPTED — Command has been accepted by OpenVault
  - ◆ UNACCEPTABLE — Command has not been accepted by OpenVault; explanation included
  - ◆ OK — Task completed successfully; return value(s) enclosed
  - ◆ ERROR — Task completed unsuccessfully; explanation enclosed



# AAPI command example 1

- Create a new CARTRIDGEGROUP, and add an APPLICATION to it

```
create taskID["t5st419"] type[CARTRIDGEGROUP]
  set[nameValue["CartridgeGroupName" "Physics"]
    nameValue["CGPreference" "3612"]];
ACCEPTED taskID["t5st419"];
OK taskID["t5st419"];
create taskID["t5st4221"]
  type[CARTRIDGEGROUPELEMENT]
  set[nameValue["CartridgeGroupName" "Physics"]
    nameValue["ApplicationName" "App12"]
    nameValue["CartridgeGroupElementPriority"
      "100"]];
ACCEPTED taskID["t5st4221"];
OK taskID["t5st4221"];
```



## AAPI command example 2

- Show the names of all APPLICATIONs which have CARTRIDGEs in the LIBRARY named Wiedener

```
show taskID["14"]
  report[CARTRIDGE."CartridgeOwner"]
  reportMode["value"]
  match[and(
    strEQ(LIBRARY."LibraryName" "Wiedener")
    strNE(CARTRIDGE."CartridgeOwner"
          "OpenVault"))];
ACCEPTED taskID["14"];
OK taskID["14"] text["App1"]
  text["NewApp"];
```





# Data Model

- OpenVault data is stored in tables
- 22 externally-viewable tables
- Internal structure implementation dependent
- CAPI/AAPI queries produce implicit merges
- Merges are Least-Restrictive



# MLM Tables

- CARTRIDGE
- SIDE
- PARTITION
- VOLUME
- LIBRARY
- DRIVE
- BAY
- BAYFF
- SLOT
- APPLICATION
- LCP
- DCP
- SESSION
- CARTRIDGEGROUP
- CARTRIDGEGROUPELEMENT
- DRIVEGROUP
- DRIVEGROUPELEMENT
- REQUEST
- DRIVETAG
- DRIVETAGATTR
- DRIVETAGSTRING
- MOUNT



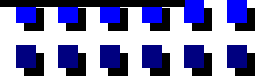
# Example AAPI query

- Show PCLs and current locations of all cartridges which could be placed into LIBRARY named Wiedener:

**QUERY: show**

```
match[ strEQ( LIBRARY. "LibraryName"  
"Wiedener" ) ]
```

```
report[ CARTRIDGE. "CartridgePCL"  
CARTRIDGE. "LibraryName" ] ;
```



# Example LIBRARY Table

- (Same) QUERY: `show match[streQ(LIBRARY."LibraryName" "Wiedener")] report[CARTRIDGE."CartridgePCL" CARTRIDGE."LibraryName"];`

- True LIBRARY Table:

<u>LIBRARYNAME</u>	<u>LibraryLCP</u>	<u>LibraryOnline</u>	<u>LibraryStatus</u>
Robbie	5	Yes	Ready
====> Wiedener	4	Yes	Ready
Snarf	9	Yes	Ready

- Resultant LIBRARY Table:

<u>LIBRARYNAME</u>	<u>LibraryLCP</u>	<u>LibraryOnline</u>	<u>LibraryStatus</u>
Wiedener	4	Yes	Ready



# Example CARTRIDGE Table

- (Same) QUERY: `show match[ strEQ( LIBRARY. "LibraryName" "Wiedener" ) ] report[ CARTRIDGE. "CartridgePCL" CARTRIDGE. "LibraryName" ] ;`

- True CARTRIDGE table (column names shortened)

<u>CARTID</u>	<u>CartPCL</u>	<u>ExtForm</u>	<u>CGName</u>	<u>CartOwn</u>	<u>LibraryName</u>	...
12-1...	AB1234	DLT	Finan	App12	Robbie	...
1-2A...	XY1211	DLT	XZGRP	NW	Wiedener	...
1-3A...	XZ1112	DLT	XZGRP	NW	Wiedener	...
142-12.	NM2511	8MM	XZGRP	NW	R2D2	...



# Doing CARTRIDGE x LIBRARY

- (Same) QUERY: show  
match[streQ(LIBRARY."LibraryName" "Wiedener")]  
report[CARTRIDGE."CartridgePCL"  
CARTRIDGE."LibraryName"];
- CARTRIDGE x LIBRARY  
naturaljoin LIBRARY with BAYFF (LibraryName)  
thetajoin PREVIOUS with CARTRIDGE where  
streQ(BAYFF.ExtForm CARTRIDGE.ExtForm)
- True BAYFF table (column names shortened)

<u>LIBRARYNAME</u>	<u>BAYNAME</u>	<u>ExtForm</u>	<u>NumSlts</u>	<u>NumFreeSlts</u>
Robbie	Bay1	DLT	28	10
Wiedener	Only	DLT	412	82
R2D2	Bin	8MM	20	2



# Intermediate Result

■ (Same) QUERY: `show`  
`match[streQ(LIBRARY."LibraryName" "Wiedener")]`  
`report[CARTRIDGE."CartridgePCL"`  
`CARTRIDGE."LibraryName"];`

■ LIBRARY

<u>LIBRARYNAME</u>	<u>LibraryLCP</u>	<u>LibraryOnline</u>	<u>LibraryStatus</u>
Wiedener	4	Yes	Ready

■ CART

<u>CARTID</u>	<u>CartPCL</u>	<u>ExtForm</u>	<u>CGName</u>	<u>CartOwn</u>	<u>LibraryName</u>	...
12-1...	AB1234	DLT	Finan	App12	Robbie	...
1-2A...	XY1211	DLT	XZGRP	NW	Wiedener	...
1-3A...	XZ1112	DLT	XZGRP	NW	Wiedener	...

■ BAYFF

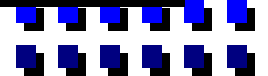
<u>LIBRARYNAME</u>	<u>BAYNAME</u>	<u>ExtForm</u>	<u>NumSlts</u>	<u>NumFreeSlts</u>
Wiedener	Only	DLT	412	82





# Final Result

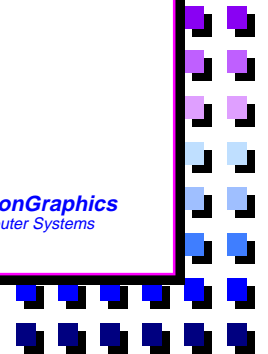
- (Same) QUERY: show  
match[ strEQ( LIBRARY. "LibraryName"  
"Wiedener" ) ]  
report[ CARTRIDGE. "CartridgePCL"  
CARTRIDGE. "LibraryName" ] ;
- OK text[ "AB1234" "Robbie" ]  
text[ "XY1211" "Wiedener" ]  
text[ "XZ1112" "Wiedener" ]

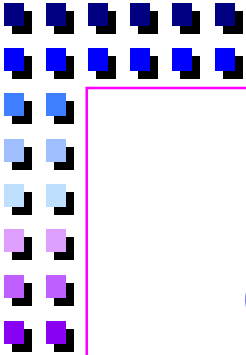




# Example CAPI Query

- CAPI same as AAPI for the above, except some restrictions:
  - ◆ CAPI view of LIBRARY only has LIBs w/application-accessible DRIVES
  - ◆ CAPI view of CARTRIDGE only has CARTs owned by or possibly-owned-by application
  - ◆ CAPI view of BAYFF is restricted to those LIBRARYs application can see





# Power of System: Complex AAPI Command

- Eject from all EXB-210 LIBRARYs all CARTRIDGEs which are not owned by APPLICATIONs, and report the affected LIBRARY names and CARTRIDGE PCLs:

**eject**

```
match[and(strEQ(CARTRIDGE."CartridgeOwner"  
              "OpenVault")  
         strEQ(LIBRARY."LibraryModel" "EXB-210")  
         strEQ(LIBRARY."LibraryName"  
               CARTRIDGE."LibraryName"))]  
report [LIBRARY."LibraryName"  
        CARTRIDGE."CartridgePCL"];
```





# CARTRIDGE

- CARTRIDGEID
- CartridgePCL
- ExternalForm
- MediaType
- MediaSize
- NumberSides
- CartridgeGroupName
- CartridgeOwner
- LibraryName
- NumberMounts
- TimeMounted
- LastMountTime
- CreationTime
- <attrs>

# SIDE

- CARTRIDGEID
- SIDENUMBER
- SideNumberParts
- <attrs>



# PARTITION

- CARTRIDGEID
- SIDENUMBER
- PARTITIONNUMBER
- PartHash
- <attrs>

# VOLUME

- CARTRIDGEID
- SIDENUMBER
- PARTITIONNUMBER
- APPLICATIONNAME
- VOLUMENAME
- <attrs>



# LIBRARY

- LIBRARYNAME
- LibraryLCP
- LibraryOnline
- LibraryStatus
- <attrs>

# DRIVE

- DRIVENAME
- LibraryName
- BayName
- DriveDCP
- DriveGroupName
- DrivePCL
- DriveOccupied
- DriveEnabled
- DriveOnline
- DriveState
- <attrs>

# BAY

- LIBRARYNAME
- BAYNAME
- BayAccessible
- <attrs>



# BAYFF

- LIBRARYNAME
- BAYNAME
- ExternalForm
- NumSlots
- NumFreeSlots
- <attrs>

# SLOT

- LIBRARYNAME
- SLOTNAME
- BayName
- CartridgePCL
- CartridgeID
- ExternalForm
- SlotOccupied
- SlotAccessable
- <attrs>



# APPLICATION

- APPLICATIONNAME
- ApplicationKey
- <attrs>

# LCP

- LIBRARYNAME
- LCPHOST
- LCPEntity
- LCPState
- <attrs>

# DCP

- DRIVENAME
- DCPHOST
- DCPEntity
- DCPState
- <attrs>

# SESSION

- SESSIONID
- ClientType
- ClientName
- ClientHost
- <attrs>



# CARTRIDGEGROUP

- CARTRIDGEGROUPNAME
- CGPreference
- <attrs>



# CARTRIDGEGROUPELEMENT

- CARTRIDGEGROUPNAME
- APPLICATIONNAME
- CGEUsePriority
- <attrs>



# DRIVEGROUP

- DRIVEGROUPNAME
- DefaultUnloadTime
- <attrs>



# DRIVEGROUPELEMENT

- DRIVEGROUPNAME
- APPLICATIONNAME
- DGEUsePriority
- DGEUnloadTime
- <attrs>



# REQUEST

- REQUESTID
- RequestorName
- RequestorType
- RequestorEntity
- Request
- RequestState
- RequestType
- ResponderName
- ResponderType
- ResponderEntity
- Response
- EntryTime
- AcceptTime
- CloseTime
- AcceptanceList
- <attrs>



# DRIVETAG

- DRIVENAME
- DCPHOST
- DRIVETAGNAME
- <attrs>



# DRIVETAGATTR

- DRIVENAME
- DCPHOST
- DRIVETAGNAME
- DRIVETAGCAPNAME
- DRIVETAGCAPVALUE
- <attrs>



# DRIVETAGSTRING

- DRIVENAME
- DCPHOST
- DRIVETAGNAME
- DRIVETAGSTRING
- <attrs>



# MOUNT

- DRIVENAME
- DRIVEHOST
- DRIVEHANDLE
- MountTag
- MountTime



# OpenVault Business Plan (1)

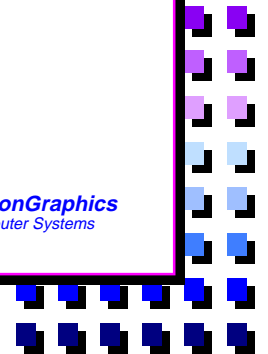
- Specifications available under NDA now
- Initial (alpha) developer kits:
  - ◆ Q1CY97
  - ◆ Partners can begin development
  - ◆ SGI needs feedback on language, interfaces, etc.
- Beta software available Q1CY97
- Release planned for Q2CY97





# OpenVault Business Plan (2)

- NetWorker 4.2.7/OV mid-1997
- DMF 2.6 FCS mid-1997
- Bundle MLM in future IRIX releases
- Bundle LCP & DCP with OEM hardware
- Charge for LCP & DCP for Gold Seal 3rd party hardware
- Others are free to write LCPs and DCPs





# OpenVault Business Plan (3)

- Source license — *no* recurring revenue
  - ◆ includes design documentation
  - ◆ includes limited amount of consultation
  - ◆ all components — \$50,000
  - ◆ application (CAPI, AAPI) — \$5,000
  - ◆ infrastructure (ADI, ALI) — \$10,000
- APIs will be published





# Contact Information

- Mike Hardy <mhardy@engr.sgi.com>  
(415) 933-7333
- Curtis Anderson <curtis@engr.sgi.com>  
(415) 933-1193
- Mark Epstein <mee@engr.sgi.com>  
(415) 933-2776
- Geoff Peck <geoff@sgi.com>  
(415) 933-3091

