OpenVMS – 30 Years of Reliable and Secure Computing

Andy Goldstein
OpenVMS Engineering

30 Years of Evolution

VMS in 1977

- Digital's 32 bit supermini
- Terminals & timesharing
- Fortran: engineering, simulation
- 1.5M lines of code in base system

30 Years of Evolution

VMS Now

- $3K workstations to $25M multi-site clusters
- VAX, Alpha and, Itanium cpu
- Factory floor, interbank EFT, business transactions, traffic control, database, internet server
- > 25M lines of code
- More reliable than ever
- Most V1 apps still work

Major Evolutions in VMS

- DECwindows
- Clusters
- Symmetric Multiprocessing
- Alpha Port
- Memory Management Redesign
- Itanium Port

Overview of VMS Organization

- Process: address space + thread(s) of execution
- Demand paged, virtual memory
- Partitioned address space:
  - P0: application code & data
  - P1: stacks and process-specific OS context
  - P2: 64 bit extended process space
  - S0: OS code & global data
  - S2: 64 bit extended system space

Overview of VMS Organization

- Process access modes
  - User
  - Supervisor - DCL
  - Exec - higher OS layers
  - Kernel
  - Stack per access mode

Overview of VMS Organization

- Interrupt (fork) context
  - Kernel mode, hardware IPL
  - System space only
  - Limited stack
  - “Lightweight thread” model
- AST: asynchronous procedure call
  - Process equivalent of interrupt context

I/O Subsystem

- $QIO service in process context
- Loadable device drivers
- Driver pre-processing in user context
- Driver fork level
- I/O interrupt → driver fork
- Process context completion AST
- ACP - ancillary control process

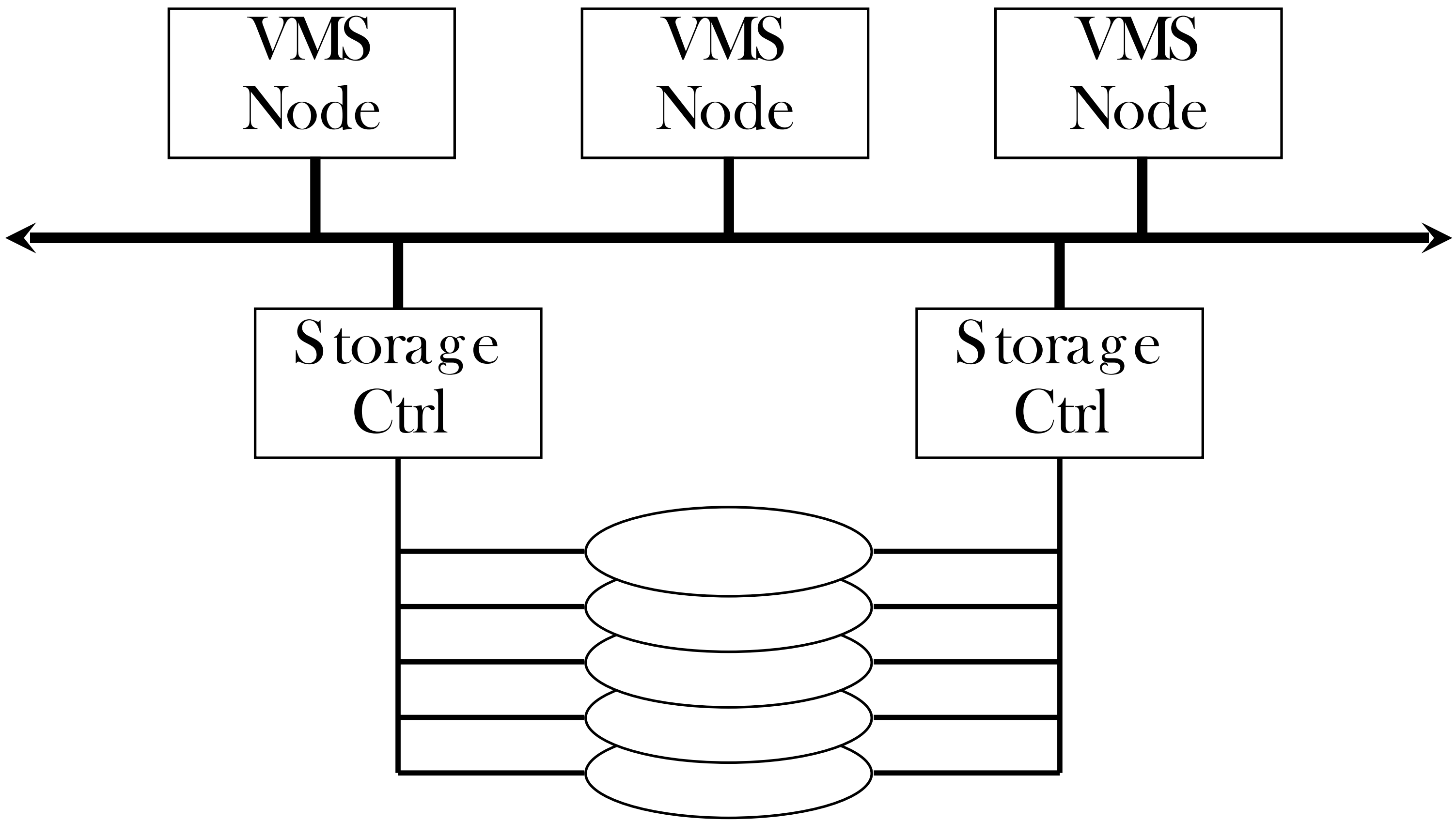
DECwindows

VMS becomes a workstation

- Graphics device drivers
- Port of X-11 and OSF Motif
- Session manager menu items:
  - DCL shell script
- Existing character cell apps:
  - Partition into character cell UI and callable application logic
  - Add new windows UI

Clusters

VMS Becomes a Distributed Operating System



Clusters: The Lock Manager

- Abstract named resources
- Lock modes to represent typical data access:
  - EX
  - PW
  - PR
  - CW
  - CR
  - NL

RMS and the Lock Manager

RMS Features

- Record-oriented I/O package
  - Sequential, direct, indexed
- Coherent shared write access with record locking
- Process local buffers with coherent cache management

Private locking implementation replaced with cluster lock manager

Before Clusters: File ACP

Server process intercepts complex file operations

- Open file context in system pool
- File metadata cache in process context
- Single thread operation provided implicit synchronization

Clusters: the File XQP

- Cluster implementation choices
  - Single server with failover
  - Multiple coordinated ACPs
- Server process converted to run in client process context
  - Cache moved to system pool
  - Simple threading package layered on AST mechanism
  - Explicit synchronization with lock manager

Symmetric Multiprocessing

Original kernel synchronization designed for uniprocessor:

- IPL 24-31: clock, cpu errors
- IPL 16-23: I/O interrupts
- IPL 8-11: device driver threads
- IPL 8: scheduling, memory management, kernel-level messages, etc.
- IPL 4: I/O completion processing
- IPL 3: process rescheduling
- IPL 2: AST delivery
- IPL 0: process execution

Symmetric Multiprocessing

Implicit IPL synchronization replaced with explicit spinlocks

- Each IPL becomes a spinlock
- IPL 8 broken into functional areas
  - Memory Management
  - Scheduling
  - Cluster communications
  - File system
  - etc.
- Continuing to refine locking

SMP Conversion

Brute force effort

- Entire kernel inspected for synchronization
- Aided by existing macros (DSBINT, ENBINT, SETIPL)
- Counters converted to interlocked instructions
- Spinlock rank design detects design deadlocks
- Debug and production locking macros

Port to Alpha

VMS and VAX were made for each other

- Privileged architecture (memory management, access modes, IPLs, etc.)
- Variable length CISC instructions, 32 bit architecture
- Most of VMS kernel in macro

Port to Alpha

Alpha is

- 64 bit architecture
- Fixed length RISC instruction

But...

- VAX-like privileged architecture
- Compatible datatypes

Port to Alpha

Rewrite:

- CPU support
- Boot code
- Some drivers
- Low level memory management
- Exception handling
- Math RTL

Port to Alpha

Compile everything else:

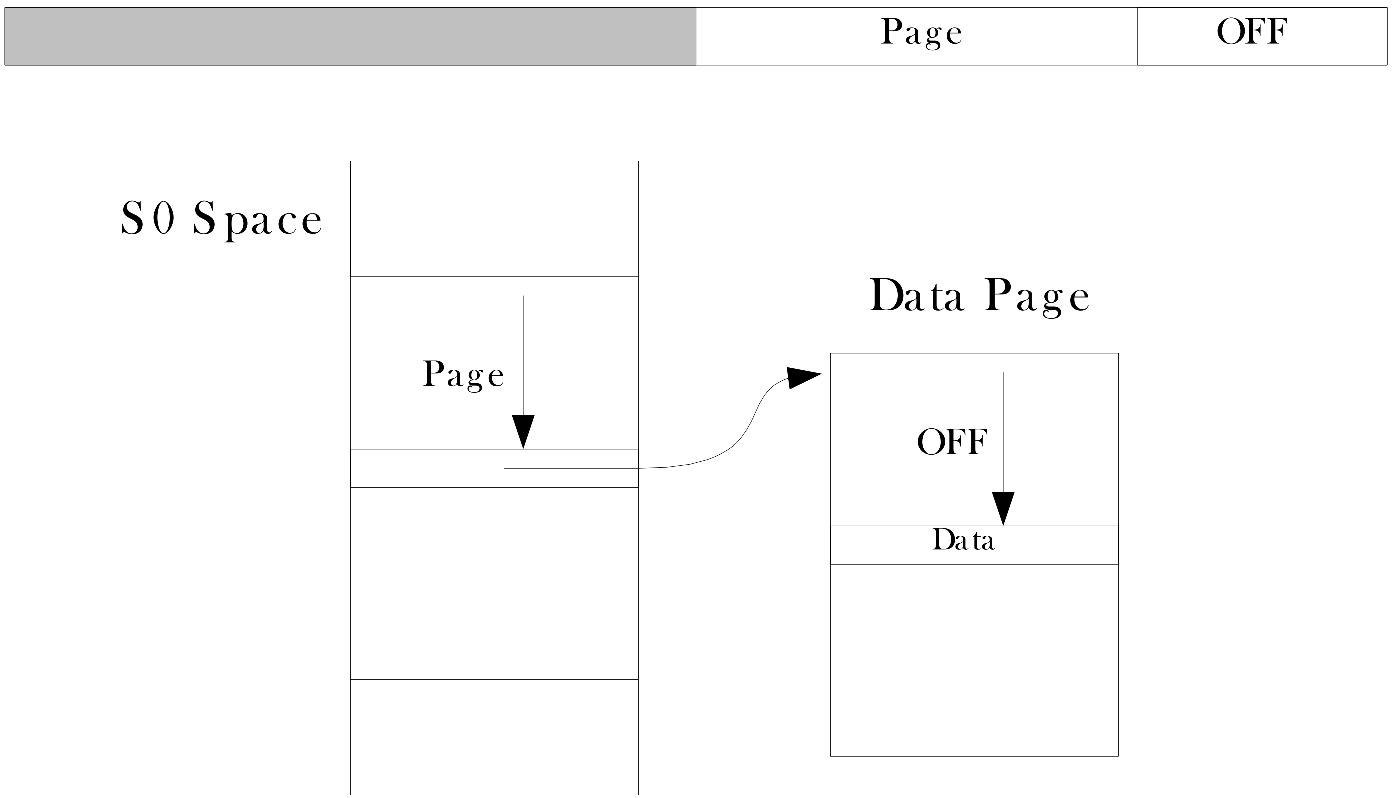
- Bliss & C
- Macro!
  - 32 bit vs 64 bit
  - Compilable macro
  - Atomicity issues
- Executable images!!

Result:

"It's really VMS. It even has the same bugs."
 - early Alpha user

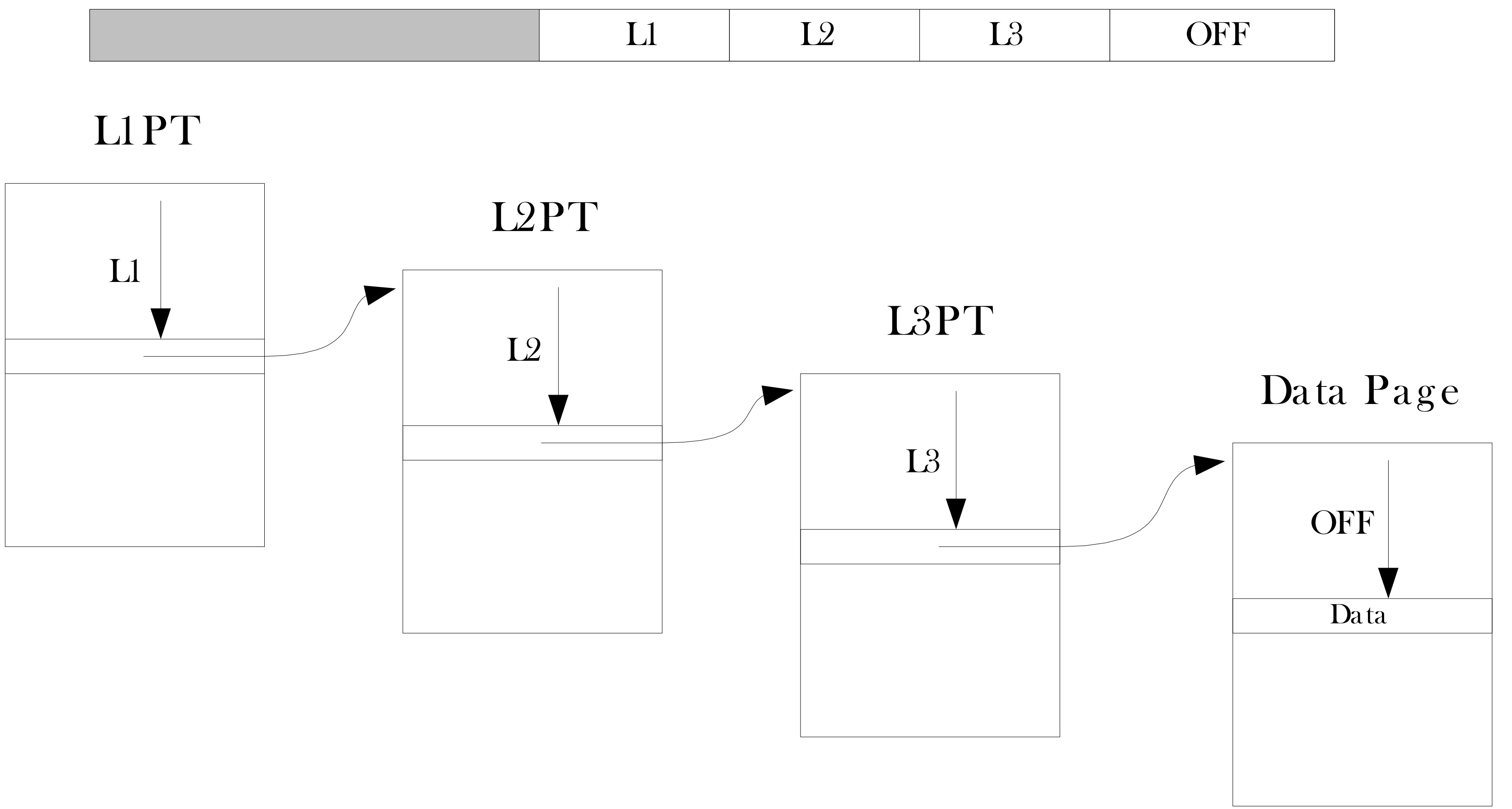
64 Bit Virtual Memory

- Original page table design



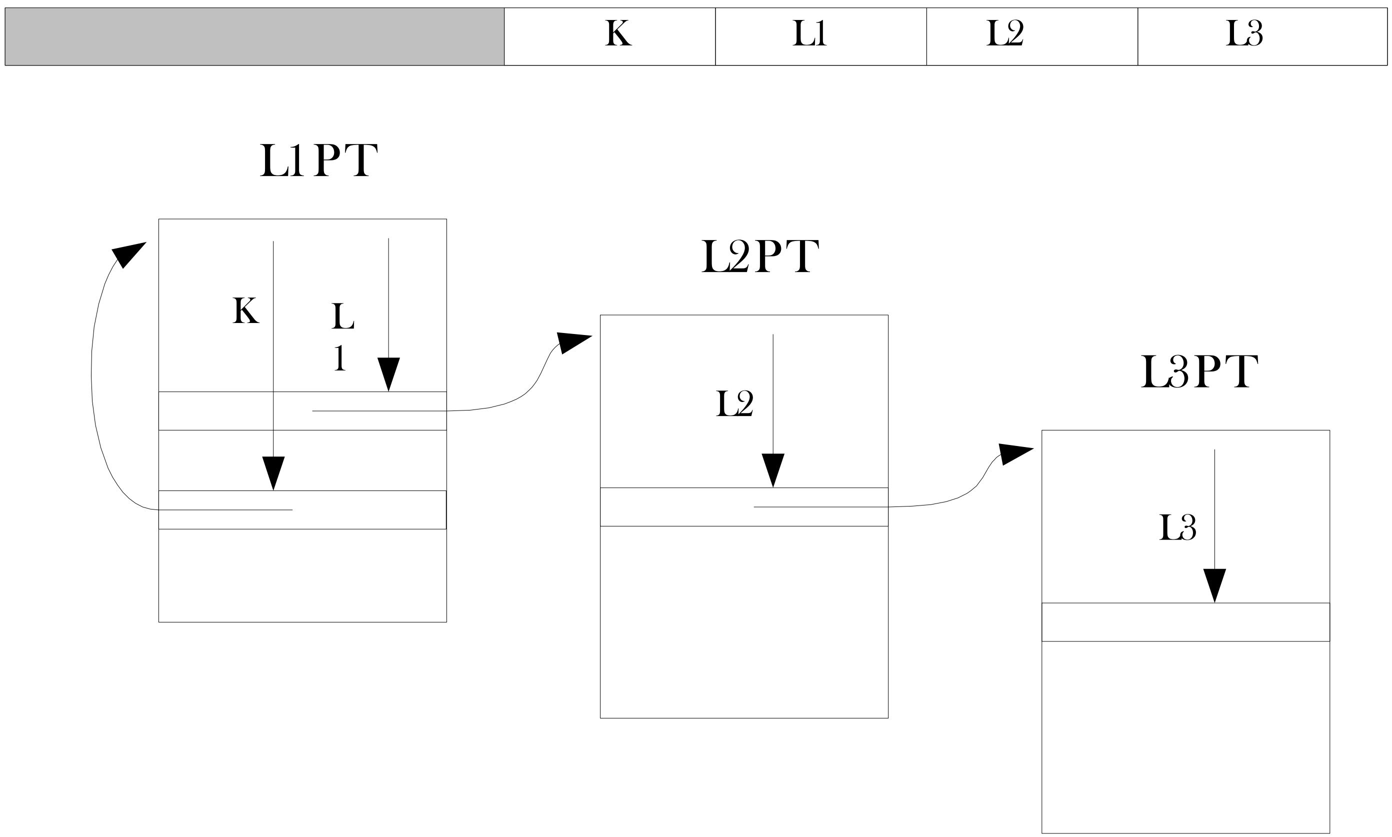
64 Bit Virtual Memory

- Extended virtual addressing



64 Bit Virtual Memory

- Page table reference



Port to Itanium

- Another 64-bit architecture, but…
- Different register conventions
- Intel calling standard
- Different privileged architecture
  - No PALcode
  - Different console / boot procedure
  - Different interrupt architecture
  - Different synchronization primitives

Port to Itanium

- Fortunately…
- 4 access modes
- Compatible memory protection features
- Memory atomicity no worse than Alpha

Port to Itanium

- Rewrite
  - CPU support
  - Boot code
- New
  - Interrupt & exception delivery in software
  - Emulation of interlocked instructions (queues, etc.)
  - EFI partition on system disk
- Redesign
  - Calling standard and condition handling
  - Object and executable file format

Port to Itanium

- Recompile
- 95% of base OS code recompiled without change
- Binary translator also available

Part 2

Building a Secure Operating System – the VMS Approach

What is Security?

Users' data handled according to a security policy

- Policy must meet users' needs
- Policy must always be correctly handled

so...

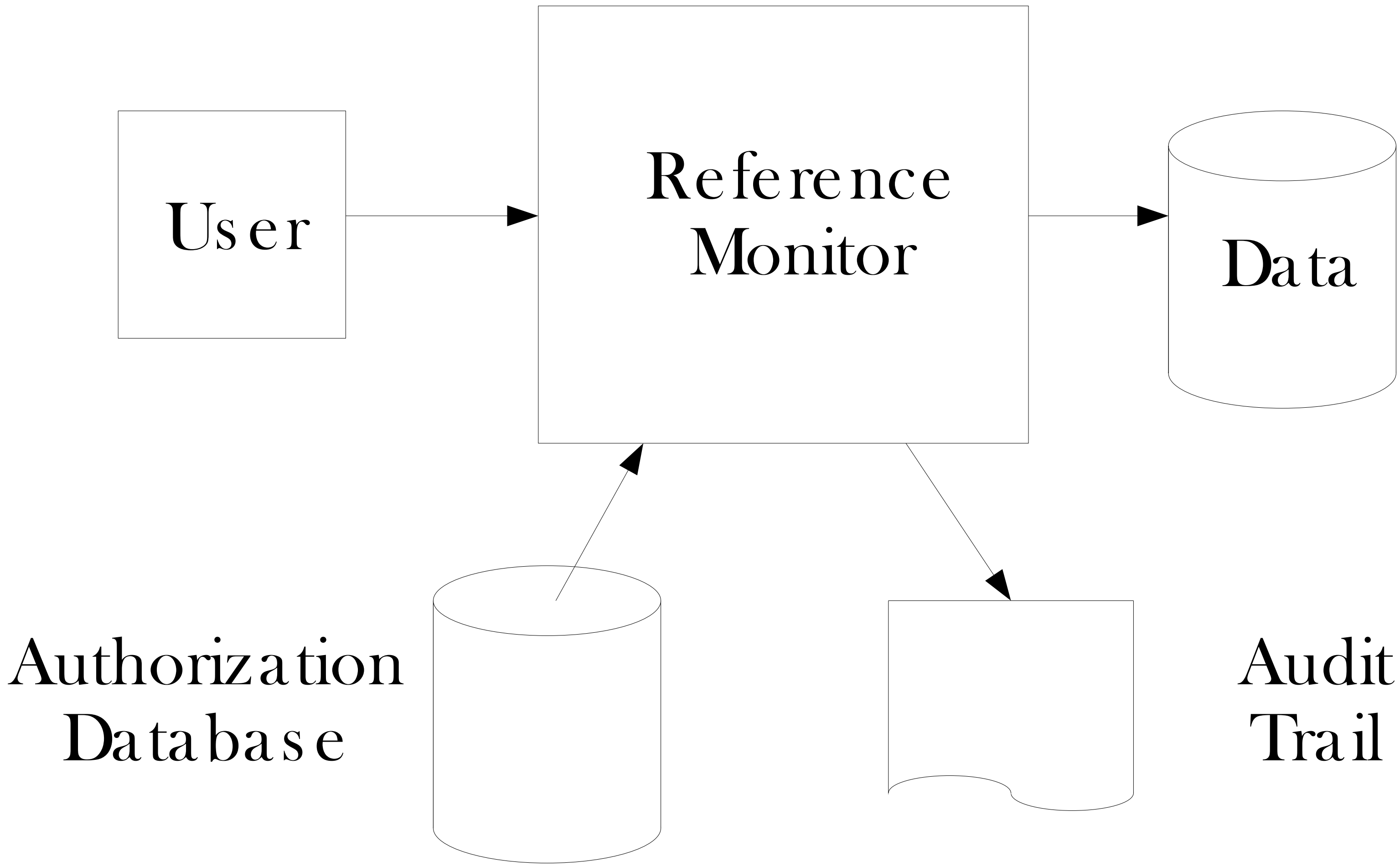
- Conceptually, security is very simple
- In practice, security is fractal (Butler Lampson)

General Concepts

- Reference monitor
  - Textbook model
  - Two examples
- System layering – the textbook and reality
  - Textbook model
  - The reality of VMS
- Typical privileged subsystem

Reference Monitor

The textbook model



Reference Monitor

- All object accesses are controlled by the reference monitor
- The authorization database determines the access control policy
- Object accesses may be recorded in the audit log
- The reference monitor, authorization database, and audit log are tamperproof

therefore…

- The reference monitor implementation must be ***correct***

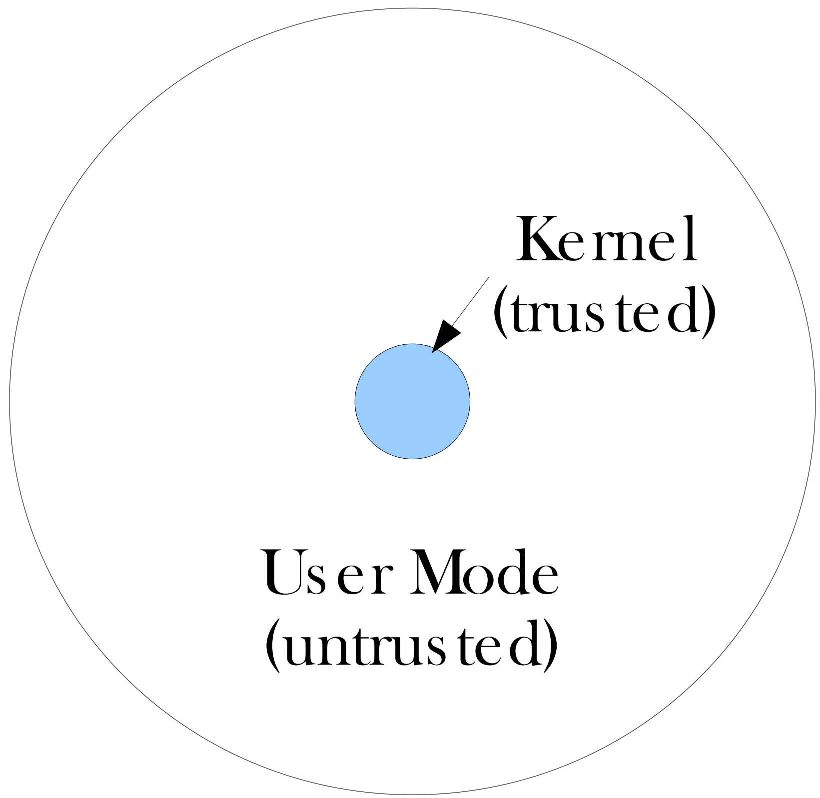
Reference Monitor

Two Examples

- An access to a file
  - The file represents the data object
  - The rights database and the file's ACL contain the authorization information
  - The access may be audited
- The $SETSWM service
  - A single bit of process state represents the data object
  - The authorization database is the PSWAPM privilege
  - Audit capability exists

System Layering

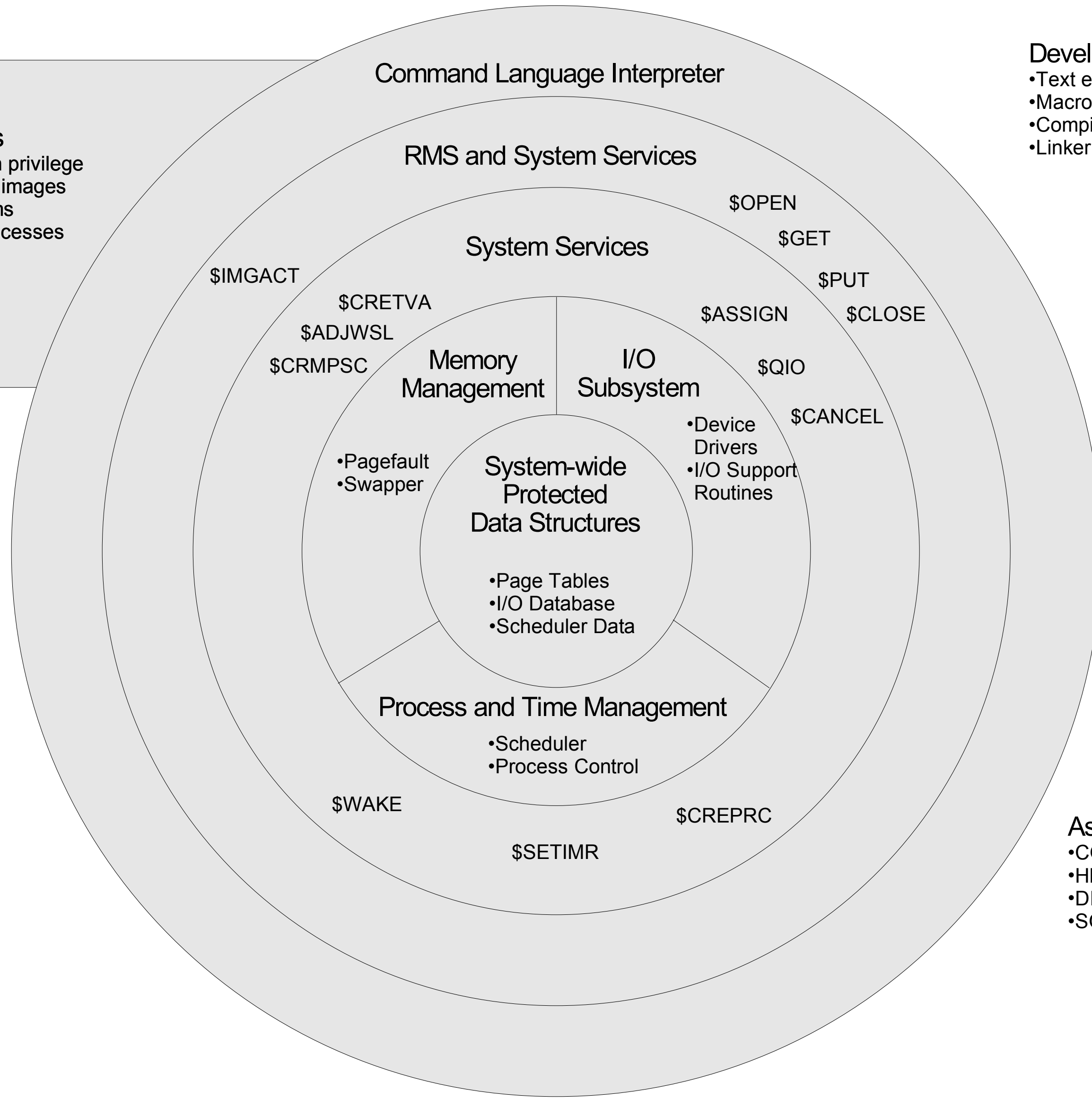
The Textbook Model



System Layering

Privileged Images
- Images installed with privilege
- Protected shareable images
- Protected subsystems
- Privileged server processes
- Informational utilities

Development Tools
- Text editors
- Macro
- Compilers
- Linker

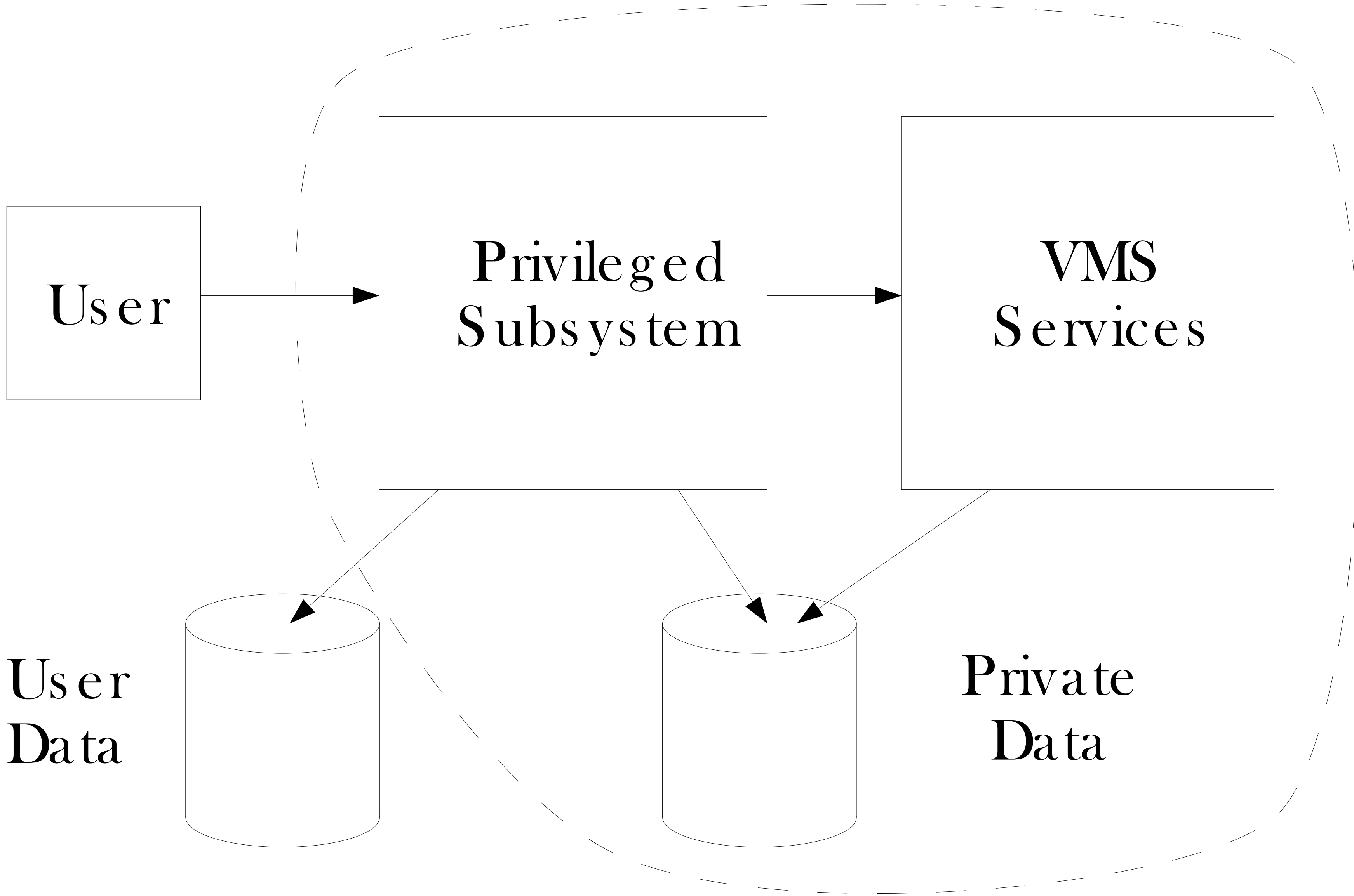


Run Time Library (General)
- Math library
- String handling
- Screen management
- Misc LIB functions

Run Time Library (Language-specific)
- CRTL
- FORTRAN
- PASCAL
- BASIC

Assorted Utilities
- COPY
- HELP
- DIRECTORY
- SORT

Typical Privileged Subsystem



Principles

- Who is performing the operation?
  - A privileged subsystem performs some operations on its own behalf, with its rights and privileges
  - It performs other operations on behalf of the user. The subsystem must take care not to take any actions the user could not have done by themselves.

Principles

- Keep track of the environment
  - How can the user affect your environment? (Logical names are a classic example.)
  - How have you altered the environment? Any alterations must be undone before the user can regain control in any way.

Principles

- Whose data is this anyway?
  - Private data must be protected from modification and/or reading by the user.
  - User data must be accessed with the access rights of the user.
  - Data under the user's control must be viewed as untrustworthy. Its contents and accessibility may change over time and must be fully validated on every reference.

Confidentiality and Integrity

- Untrusted code must not be allowed to see data that is considered private
  - Data belonging to other users
  - Data whose secrecy is critical to enforcing security policy (e.g., passwords)
- However… there is no need to hide data that is not confidential
  - Entire VMS kernel code is user readable
  - Most P1 context is user readable

Confidentiality and Integrity

- Untrusted code must not be allowed to modify data belonging to other users
- Untrusted code must not be allowed to modify data critical to the operation of the operating system
- However… there is no need to protect data that only affects the user.
  - AST active/enable, FP register use flags

Access Modes

- Hierarchy of access modes: user, supervisor, exec, kernel
- Inward transitions:
  - System service calls
  - Exceptions
  - Interrupts
  - ASTs
- Outware transitions:
  - REI

User Mode

User mode is the one mode accessible to unprivileged user software. Therefore, anything coming from user mode must be regarded with extreme suspicion.

- Logical names (including supervisor mode logicals)
- Address space. When user mode can execute, the presence, mapping, and protection of user-owned address space can change. The contents of user-owned address space can change ***at any time***.
- ASTs. User mode execution may be initiated by timers, I/O completion, etc., anytime no inner mode execution is in progress.

With a few exceptions, all user mode state is eliminated by image rundown.

Supervisor Mode

- Belongs to the command language interpretor, which is not accustomed to coexisting with other subsystems
- Is privileged because it has control over the integrity and execution of privileged images.

Exec Mode

- Outer layer of the "system kernel"
- Used for RMS, image activator, security services, etc.
- Read-only access to all system internal data structures
- Several implicit privileges
  - SYSLCK
  - Exec mode logical names
  - CMKRNL
  - SETPRV

Kernel Mode

- Access to all system internal data structures, I/O space, privileged instructions, etc.
- Execution at elevated IPL
- Access to internal synchronization

Why Have Four Modes?

Supervisor and exec modes are trusted but firewalled

- Supervisor
  - Protects in-process CLI from image
  - Only trusted to not actively interfere with privileged images
- Exec
  - Read access to all kernel data
  - Various implied privileges
  - But cannot directly crash the system

Privileges

- 64 bits of process state allowing access to security sensitive operations
- 4 privilege masks per process
  - Process authorized: maximum privileges permitted for the life of the process
  - Process current: process privileges as reduced by the user or application code
  - Image enhanced: additional privileges made available to the currently executing image by INSTALL
  - Image current: available privileges as reduced by the application

Privileges

- Why so many privileges?
- Many privileges allow complete control of the system
- Separate privileges protect against accidents
- Always apply “principle of least privilege”

Privileged Subsystems

Privileged Processes

- Own process context
- Own address space
- Typically many or all privileges set
- Typically well insulated from user attack

- but -

- Must validate all user inputs
- Operations “on behalf of the user” are problematical
  - Use impersonation services
- Examples: job controller, symbionts, security server

Privileged Subsystems

Images Installed With Privilege

- "Main programs" only
- Operate in user's process context
- Privileges enabled by image activator
- Privileges extend to all code called by image
- Examples: SET & SHOW

Privileged Subsystems

Protection of Privileged Images

- Privileges removed on image rundown
- DCL EXAMINE, DEPOSIT, DEBUG, etc., disabled; SPAWN drops privileges
- Shareable images must be installed and are activated with exec mode logicals only
- Debug and traceback hooks are disallowed

- but -

- User mode logicals apply to file operations unless disabled with RMS's FAB$V_LNM_MODE

Privileged Subsystems

Protected Subsystems (executable images with a Subsystem ACE)

- Analogous to images installed with privileges
- Rights list augmented by identifiers from subsystem ACE
- Identifiers may be enabled/disabled with $SUBSYSTEM service call
- Capabilities (and risk) determined by access conferred by subsystem identifiers
- Generally safer than privileged images

Privileged Subsystems

System Space System Service Code

- Executes in process context
- Protected by inner mode
- Have rights and privileges of the inner mode
- Callable from user mode via system service entry
- Examples: most VMS system services

Privileged Subsystems

Process Space System Service Code

- Known as
  - User-written system services
  - Protected shareable images
  - Privileged shareable images
- Loaded by image activator during image activation
- Share all other characteristics of system service code
- Examples: $MOUNT, $GETUAI, etc.

Privileged Subsystems

- Protection of Protected Shareable Images
- Address space
  - Owned by exec mode
  - Writable pages set to user read / exec write
- Cannot be overmapped
- Must not call other shareable images

Conclusions

Specific Techniques

- Privileges
- Parameter Validation
- Parameter Accessibility
- Process Deletion
- Environment
- Asynchronous Operation
- Operating on Behalf of the User
- Creating Protected Shareable Images

Privileges

- Keep them off
- Turn them on only when necessary
- Make sure privileges are off on ***all*** exit paths
- If user inputs can affect an operation executed with privilege, they must be carefully validated
- ***NEVER*** pass user parameters directly to an operation executed with privilege
- Remember the privileges implicit in inner modes

Parameter Validation

Nothing passed by a user program to a privileged procedure should be trusted

- Assume arbitrary values
  - Using addresses as "handles" is usually a bad idea
- Assume arbitrary combinations of parameters
- Validate parameter combinations for consistency
- Contents of user address space can change. If you need to use the value of a parameter multiple times, make an internal copy.

Parameter Accessibility

Inner mode services must check page protection of all parameters to ensure

- That the argument is accessible to the service
- That the argument was accessible to the user

PROBER/PROBEW operations check against previous mode in the PS. Note that in an interrupt or AST previous mode is your mode, not the user's!

Probing Arguments

- All arguments must be probed
- Argument list (except resident VMS services)
  - Alpha/IA64: memory resident arguments only
- Arguments passed by descriptor
  - Probe the descriptor
  - Copy descriptor into local storage
  - Probe the buffer
  - User the local copy from here on!

Process Deletion

A process can be deleted anytime it is executing below IPL 2 of kernel mode

- Run at IPL 2 to prevent deletion while holding unrecorded system-wide state
- Subsystem state in system storage (i.e., nonpaged or paged pool) must be cleaned up
- Use the rundown handler facility to guarantee execution at process rundown
- Exit handler execution is not guaranteed!

Environment

Inner mode subsystems do not have the entire VMS programming environment at their disposal

- Subroutine libraries: OTS$, SYS$, EXE$, IOC$, etc., are freely callable in inner mode
- Stateless RTL routines may be called
  - LIB$SIGNAL is OK
  - LIB$GET_VM, malloc(), etc. are not (work planned post V8.3)
  - CRTL components in resident exec are OK
- Do not call shareable images – link /NOSYSSHR
  - Link with SYS$BASE_IMAGE.EXE to use resident exec OTS routines
- RMS may be called from exec mode but not kernel

Logical Names

- Logical names are an implicit user input to many system services
- $OPEN/$CREATE, first and foremost
- Only exec and kernel mode logicals are "trusted"
- Image activator uses trusted logicals when activating a privileged image
- All other operations must use FAB$V_LNM_MODE

Memory Guarantees

- During inner mode execution the shape and accessibility of the address space do not change
  – Unless you call a service that changes the address space!
- The contents of memory can change from cycle to cycle
  – Direct I/O
  – Modification by other threads or processes
- Do not re-fetch arguments!
- Cross-cpu memory updates must be synchronized
- Only system space may be referenced from interrupt context

Asynchronous Operation

Kernel Threads

- Allow true concurrent execution in a single process
- Inner mode execution is protected by the inner mode semaphore
  - Thread-safe service coexists with all services
  - Thread-tolerant service coexists with other tolerant services
  - Thread-intolerant service coexists only with thread-safe services
- Almost all services are currently thread-intolerant
- Future opportunities for thread-tolerant services

Asynchronous Operation

An Inner mode subsystem may wait for an external event in the mode of the caller:

- Issue operation causing the event with a completion AST
- Clean up and return to user
- AST resumes execution (ASTPRM may pass context)

- but -

- Your environment may have changed
- All user inputs must be revalidated
- Remember previous mode = current mode

Operating on Behalf of the User

- In the same process:
  - Drop enhanced privileges
  - Disable subsystem identifiers
  - Beware of privileges implicit in inner mode
- In a server process: use impersonation services
  - $PERSONA_CREATE
  - $PERSONA_ASSUME
  - $PERSONA_RESERVE
  - $PERSONA_DELEGATE

Threads and Personas

- A process thread shares the persona of its parent
- $SETPRV and similar services modify the current persona
- Modifying a shared persona affects all threads!
- To modify the persona for just the current thread
  - $PERSONA_CLONE
  - $PERSONA_ASSUME
  - $SETPRV (or whatever)
- System manages personas for both kernel threads and pthreads

Part 3

Conclusions

How to Build a Secure and Evolvable System

It begins at the beginning

- Start with a team of grownups
- Design with care
- Keep the team small
  - Initial VMS architecture came from 3 people
  - Entire VMS V1 team was 24 people
- Keep the pressure up
  - The first known "fact" about VMS was the schedule
  - Beware of creeping elegance

How to Build a Secure and Evolvable System

- Modularity
- Modularity
- Modularity

Modularity in VMS

- Dynamically loaded modules for all configuration dependent components
- Huge number of system models and devices supported over the life of the system
- Any VMS system disk will boot on any configuration of a particular architecture
- New hardware is supported with minimal effect on the rest of the system

Modularity - Construction

- VMS kernel is organized around globally accessible data structures
  - Centralized definition
  - Synchronization rules
  - Conventions for shared vs private data
- Object-oriented is better but more expensive

Modularity - Construction

- VMS uses partial object design
  - Self-identifying data structures
  - Complex interpretation & manipulation are encapsulated
  - Direct access for simple reference
  - Partial layering of subsystems
- Modularity costs
  - Lightweight subroutine calls
  - Mode transitions (service calls) are more expensive
  - Context switch to server process is most expensive

Modularity - Interfaces

Well-defined interfaces are the core of an extensible and reliable system

- Specify behavior fully
- Specify what is unspecified
- Reject invalid inputs
- How to detect dependence on unspecified behavior?

Interface Compatibility

- Do not change documented behavior
  - New behavior may result from new inputs
- Design interfaces to be extensible
  - Variable length argument lists
  - Item lists (TLV encoding)
  - Option flags
- If all else fails, create a new interface for new behavior

Modularity - Interfaces

Do not permit use of unspecified interfaces

- To be successful, specified interfaces must be functionally complete
- VMS's nemesis is $CMKRNL

Given the opportunity, users will break the rules

Security is Built In

- Security is mostly about correctness and reliability
- Use a modular approach to minimize the impact of errors
  - Firewall functions and subsystems to confine failures
  - Apply principle of least privilege to make firewalls effective

Maintain Competence

Causes of "software rot"

- Lack of design understanding
- Quick and dirty changes
- Changes that compromise the original design
- Functional extension without extending the original design
- Duplication of function
- Runaway complexity

Maintain Competence

- Write design specifications!
- Retain engineering expertise (written designs are never good enough)
  - Major evolutions in VMS required widespread changes regardless of modularity
- "Why" is even more important than "how"
- Clean house occasionally
  - Many VMS components have been rewritten over time
- Organizational commitment to quality is critical

The End
and
Thank You

invent