Access Control in Practice

CS461/ECE422

Fall 2010

Reading

- Computer Security – Chapter 2
- Computer Security – Chapter 15

Outline

- Evolution of OS
- Object Access Control
  - Access control lists
  - Capabilities

In the Beginning...

- The program owned the machine
  - Access all power of the hardware
  - Could really mess things up
- Executives emerged
  - Gather common functionality
- Multi-user systems required greater separation
  - Multics, the source of much early OS development

Types of Separation

- Physical
  - Use separate physical resources, e.g. Printers, disk drives
- Temporal
  - Time slice different users
- Logical
  - Create virtual environment to make it seem that programs are running independently
- Cryptographic
  - Hide data and computation from others

Protecting objects

- Desire to protect logical entities
  - Memory
  - Files or data sets
  - Executing program
  - File directory
  - A particular data structure like a stack
  - Operating system control structures
  - Privileged instructions

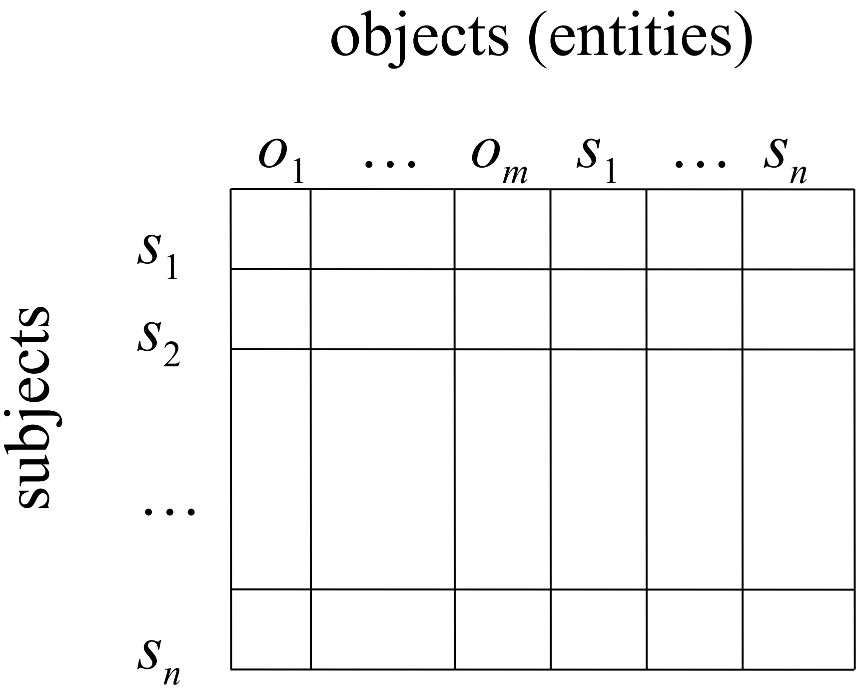
Access Control Matrix

- Access Control Matrix (ACM) and related concepts provides very basic abstraction
  - Map different systems to a common form for comparison
  - Enables standard proof techniques
  - Not directly used in implementation

Definitions

- Protection state of system
  - Describes current settings, values of system relevant to protection
- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix

Description



- Subjects $S = \{ s_1,\ldots,s_n \}$
- Objects $O = \{ o_1,\ldots,o_m \}$
- Rights $R = \{ r_1,\ldots,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject s_i has rights $r_x, \ldots, r_y$ over object o_j

Example 1

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

Example 2

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights +, –, *call*

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

State Transitions

- Change the protection state of system
- $\vdash$ represents transition
  - $X_i \vdash_{\tau} X_{i+1}$: command τ moves system from state X_i to X_{i+1}
  - $X_i \vdash^{*} X_{i+1}$: a sequence of commands moves system from state X_i to X_{i+1}
- Commands often called *transformation procedures*

Example Transitions

	Sam	Joe
Sam	∅	∅
Joe	∅	∅

⊢

	Sam	Joe	Code
Sam	∅	∅	{own}
Joe	∅	∅	∅

⊢

	Sam	Joe	Code	Data
Sam	∅	∅	{own}	{own}
Joe	∅	∅	∅	∅

⊢

	Sam	Joe	Code	Data
Sam	∅	∅	{own}	{own}
Joe	∅	∅	{execute}	∅

⊢

	Sam	Joe	Code	Data
Sam	∅	∅	{own}	{own}
Joe	∅	∅	{execute}	{read}

Example Composite Transition

	Sam	Joe
Sam	∅	∅
Joe	∅	∅

$\vdash^*$

	Sam	Joe	Code	Data
Sam	∅	∅	{own}	{own}
Joe	∅	∅	{execute}	{read}

HRU Model

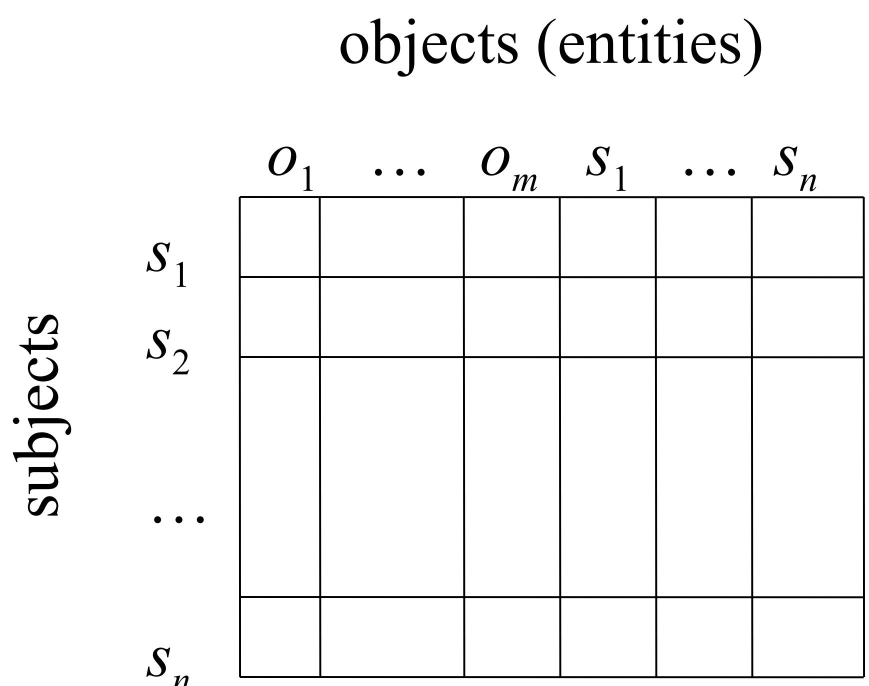
- Harrison, Ruzzo, and Ullman proved key safety results in 1976
- Talked about systems
  - With initial protection state expressed in ACM
  - State transition commands built from a set of primitive operations
  - Applied conditionally.

HRU Commands and Operations

- **command *α(X1, X2 , . . ., Xk)***
  if rl in A[*Xs1, Xo1*] **and** r2 in A[Xs2, *Xo2*] **and ...** rk in A[*Xsk, Xok*]
  then
  op1; op2; … opn
  end
- **6 Primitive Operations**
  - **enter r into A[***Xs, Xo***]**
  - **delete r from A[***Xs, Xo***]**
  - **create subject** *Xs*
  - **create object** *Xo*
  - **destroy subject** *Xs*
  - **destroy object** *Xo*

Practical object access control

- Can slice the logical ACM two ways
  - By row: Store with subject
  - By column: Store with object



Access Control List

- Slice by Object
  - Used by Multics and most modern OS's
- Let S be set of subjects and R set of rights in system
  - Access Control List (ACL) *l* is set of pairs

    $$l = (s, r) : s \in S, r \subseteq R$$

  - $acl(o) = \{ (s_i, r_i) : 1 \leq i \leq n \}$ means any s_i can access *o* using r_i

Example 1

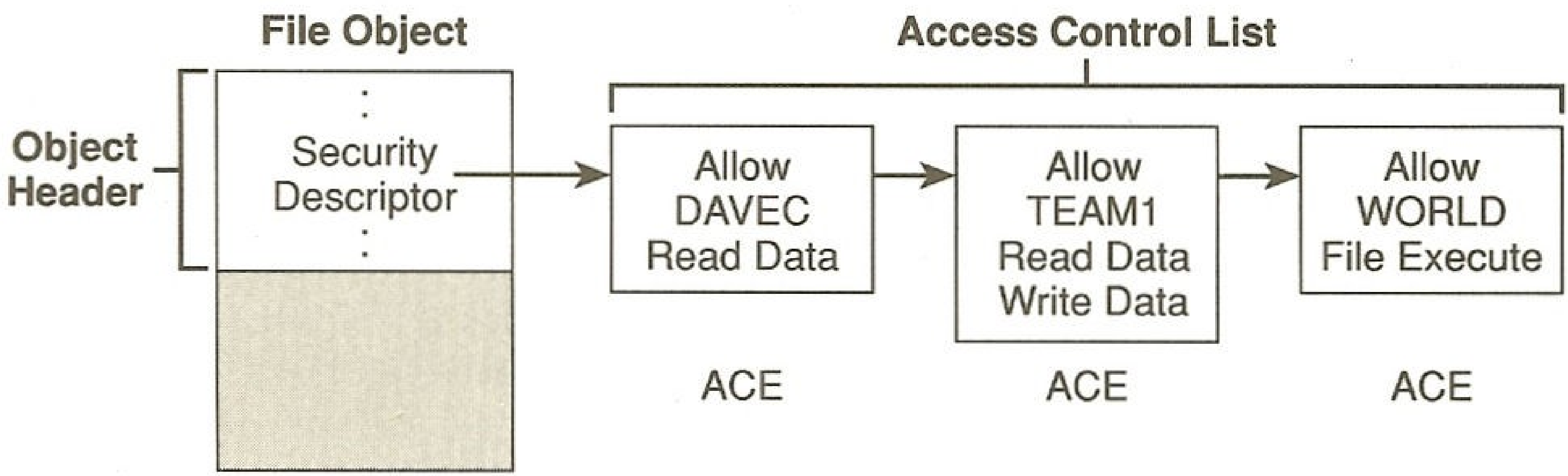
- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

Unix Access Control

- Three permission octets associated with each file and directory
  - Owner, group, and other
  - Read, write, execute
- For each file/directory
  - Can specify RWX permissions for one owner, one group, and one other

Windows ACL


File Object
Access Control List
Object Header
Security Descriptor
Allow DAVEC Read Data
Allow TEAM1 Read Data Write Data
Allow WORLD File Execute
ACE
ACE
ACE

Windows ACL

- Actually two ACL's per file
  - System ACL (SACL) – controls auditing and now integrity controls
  - Discretionary ACL (DACL) – controls object access
- Windows ACLs apply to all named objects
  - Files
  - Pipes
  - Events

ACL Distinctions

- What subjects can modify an object's ACL?
- If there is a privileged user, do the ACLs apply to that user?
- Does the ACL support groups or wildcards?
- How are contradictory access control permissions handled?
- If a default permission is allowed, do the ACL permissions modify it, or is the default only used when the subject is not mentioned in the ACL?

Revoking rights with ACLs

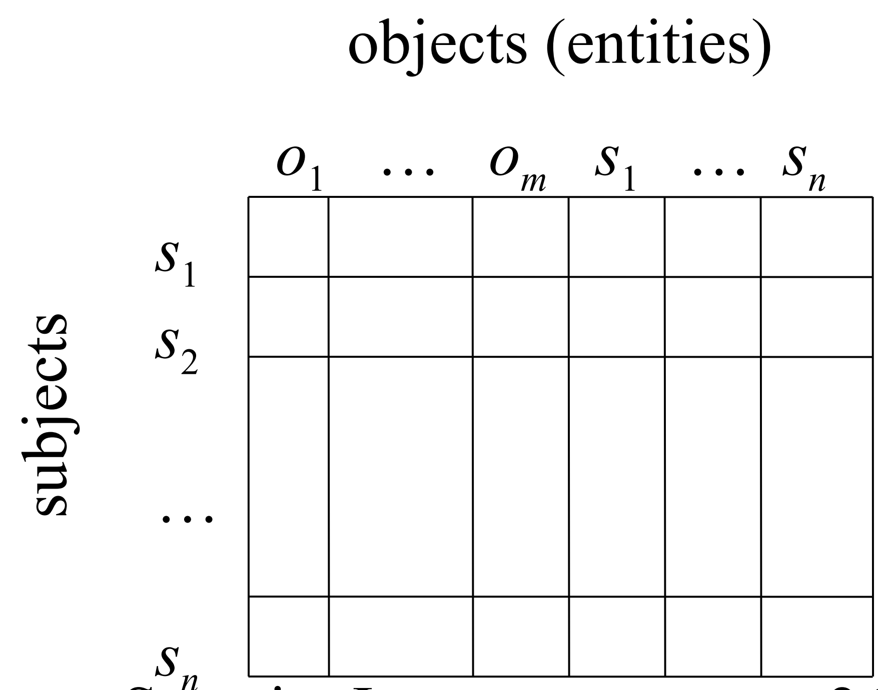
- Revoking rights for subject *s* to a particular object *o* straightforward
  - Remove *s* from ACL(*o*)
  - Make sure *s* has a negative entry in the ACL(*o*)
- Example: Alice removes all of Bob's rights to f
  - What if Bob had given Carol read rights to f?
  - Should Carol still have those rights?

ACL Scaling

- Groups of users
- Role Base Access Control
  - Users can take on role at a time
- Directory inheritance
- Negative rights

Practical object access control

- Can slice the logical ACM two ways
  - By row: Store with subject
  - By column: Store with object



Capability List

- Slice by Subject
  - Experimented with in the 80's. Often with object-oriented systems.
- Let O be set of objects and R set of rights in system
  - Capability list (C-List) c is a set of pairs
    - $c = (o, r) : o \in O, r \subseteq R$
  - $cap(s) = \{ (o_i, r_i) : 1 \leq i \leq n \}$ means s can access o_i using r_i

Example 1

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a*, *o*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

Capability Integrity

- Subject presents capability to access object
  - Capability encapsulates object ID with allowed rights.
- Unlike ACLs, capabilities are not completely contained by the OS
- Capability integrity is a big concern
  - Tagged memory
  - Segmented memory
  - Cryptographic hashs

Capabilities and propagation

- Copy rights
  - Separate version of the base right, e.g read-copy
  - Some systems had explicit copy bit
- Right amplification
  - May need to temporarily amplify rights to object
  - Perhaps just within particular method or module
  - Combine abstract class rights with object rights
  - Counter module example
    - In generally user only has right to invoke counter module on variable of counter type
    - In counter code, process must perform additional operations.

Revoking capabilities

- Easy to revoke all rights to a given subject
- What about revoking everyone's rights to a particular object?

Capabilities HW

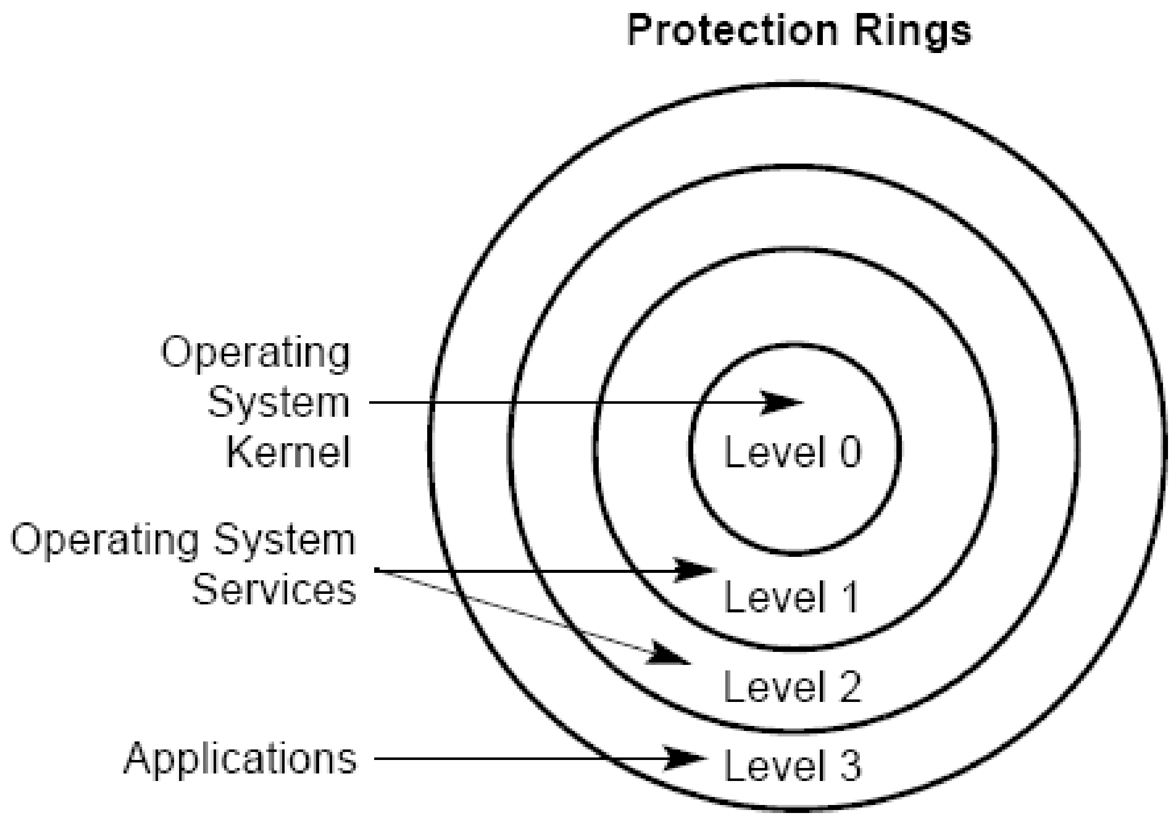
- Intel iAPX 432 (mid ’70s)
  - Tried to put even more security enforcement in hardware
  - Capabilities and object-oriented
  - Implementation too complex and compiler technology not sufficiently smart
  - http://en.wikipedia.org/wiki/Intel_iAPX_432
- IBM System/38
  - From about the same time period
  - Also had hardware capabilities support
- *Capability-Based Computer Systems* by Henry N. Levy
  - http://www.cs.washington.edu/homes/levy/capabook/

Protection Rings

- CS 15.4 – describes Multics implementation
- Intel Pentium II Software Developer's Manual: Volume 3. Sections 4.5 through 4.8
  - http://developer.intel.com/design/processor/manuals/2

Memory Protection Rings

- Originally in Multics
- In Intel arch since x386

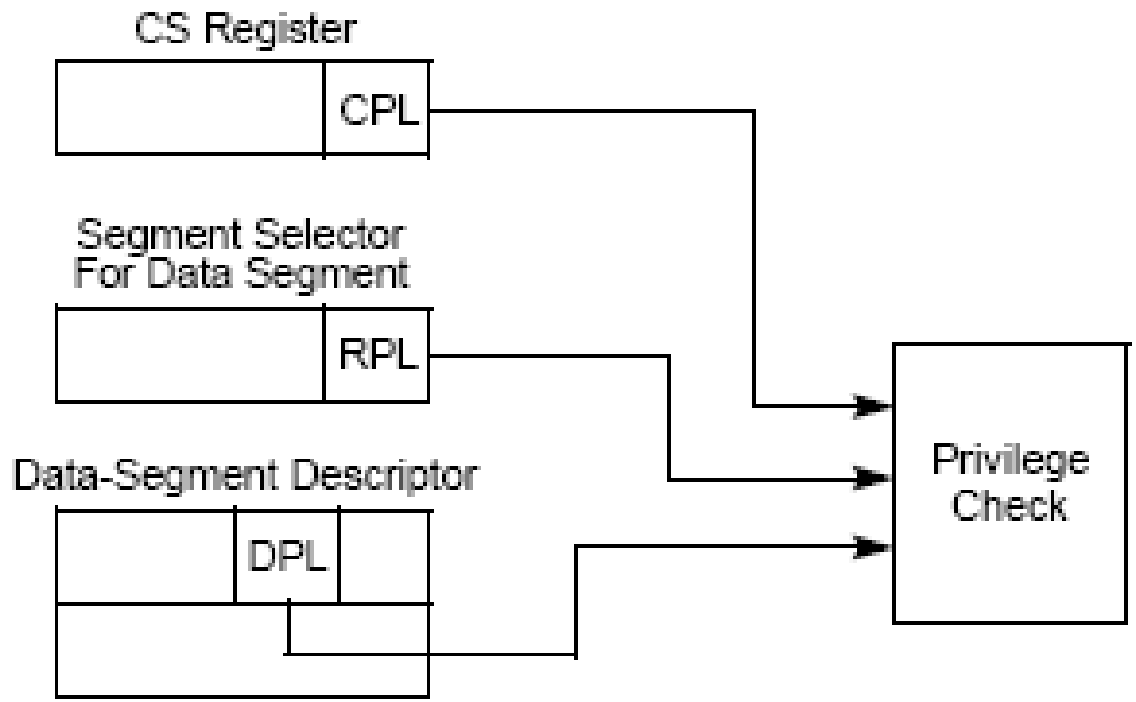


Privilege Levels

- CPU enforces constraints on memory access and changes of control between different privilege levels
- Similar in spirit to Bell-LaPadula access control restrictions
- Hardware enforcement of division between user mode and kernel mode in operating systems
  - Simple malicious code cannot jump into kernel space

Data Access Rules

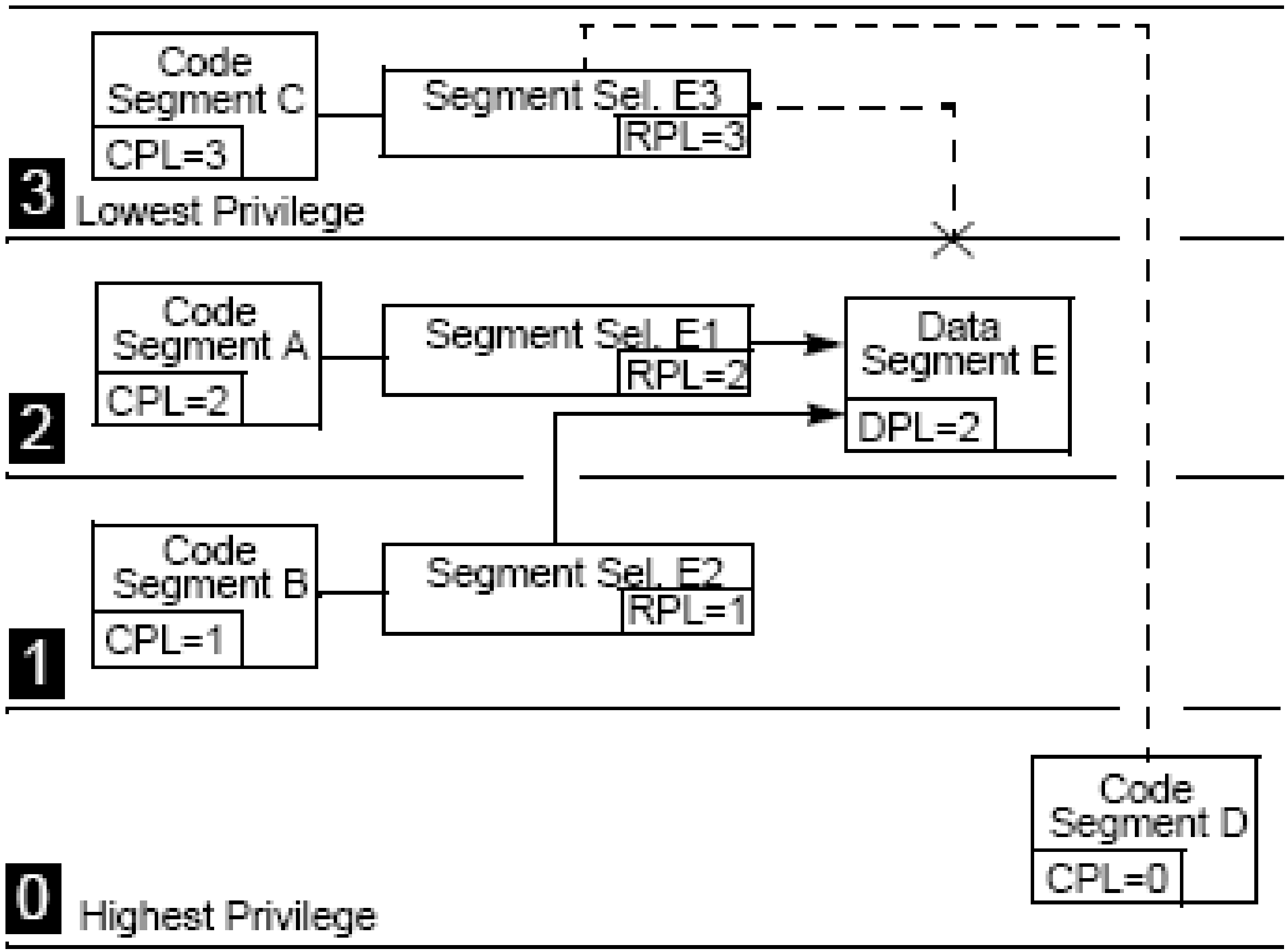
- Access allowed if
  - CPL <= DPL and RPL <= DPL



Data Access Rules

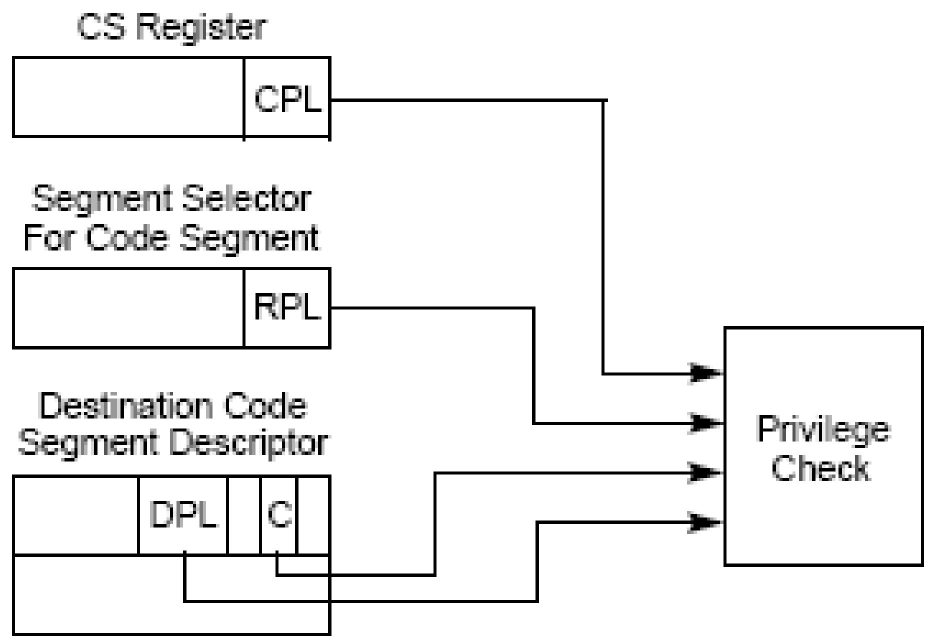
- Three players
  - Code segment has a current privilege level CPL
  - Operand segment selector has a requested privilege level RPL
  - Data Segment Descriptor for each memory includes a data privilege level DPL
- Segment is loaded if CPL <= DPL and RPL <= DPL
  - i.e. both CPL and RPL are from more privileged rings

Data Access Examples

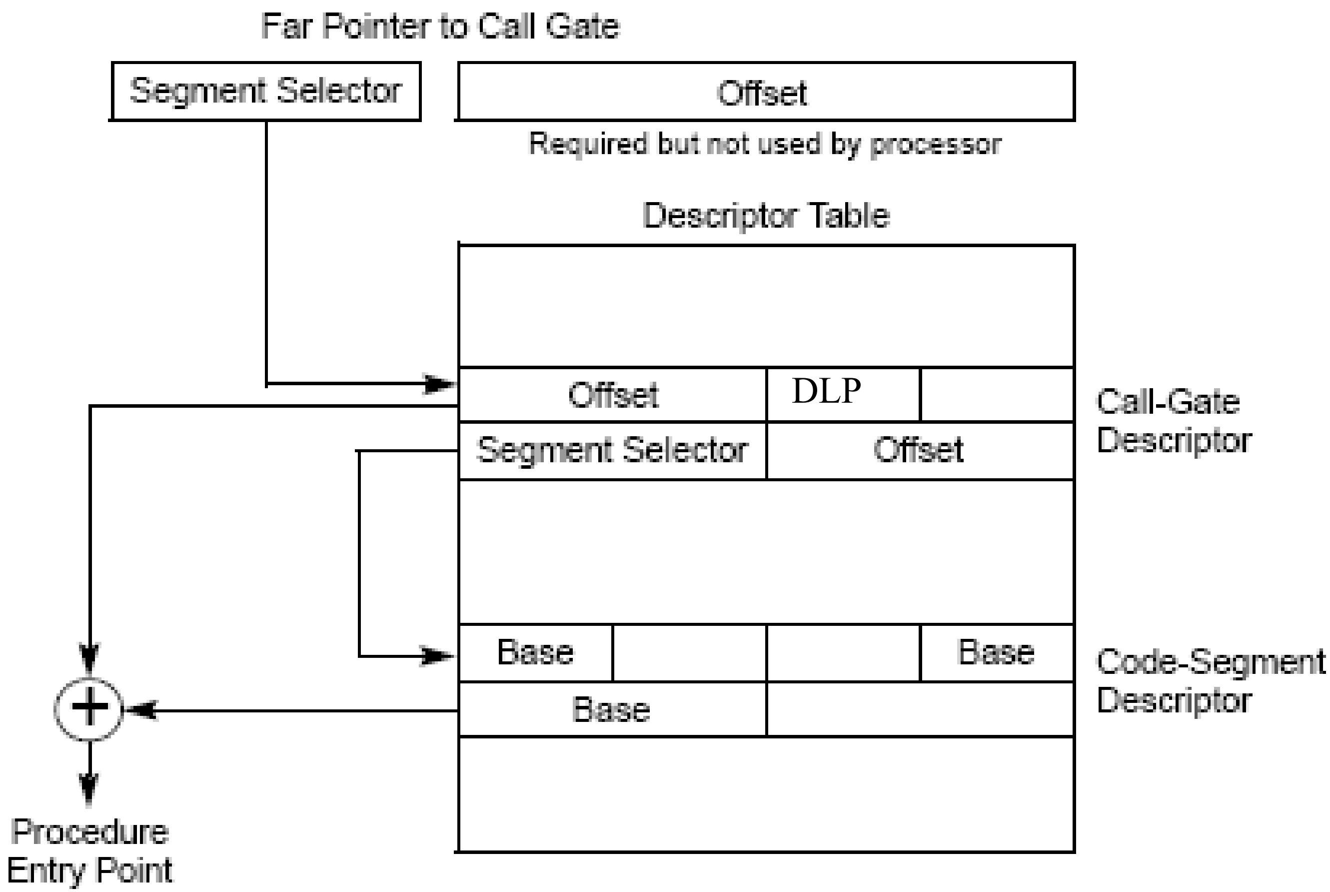


Direct Control Transfers

- For non-conforming code (the common case)
  - RPL <= CPL && CPL == DPL
  - Can only directly jump to code at same privilege level

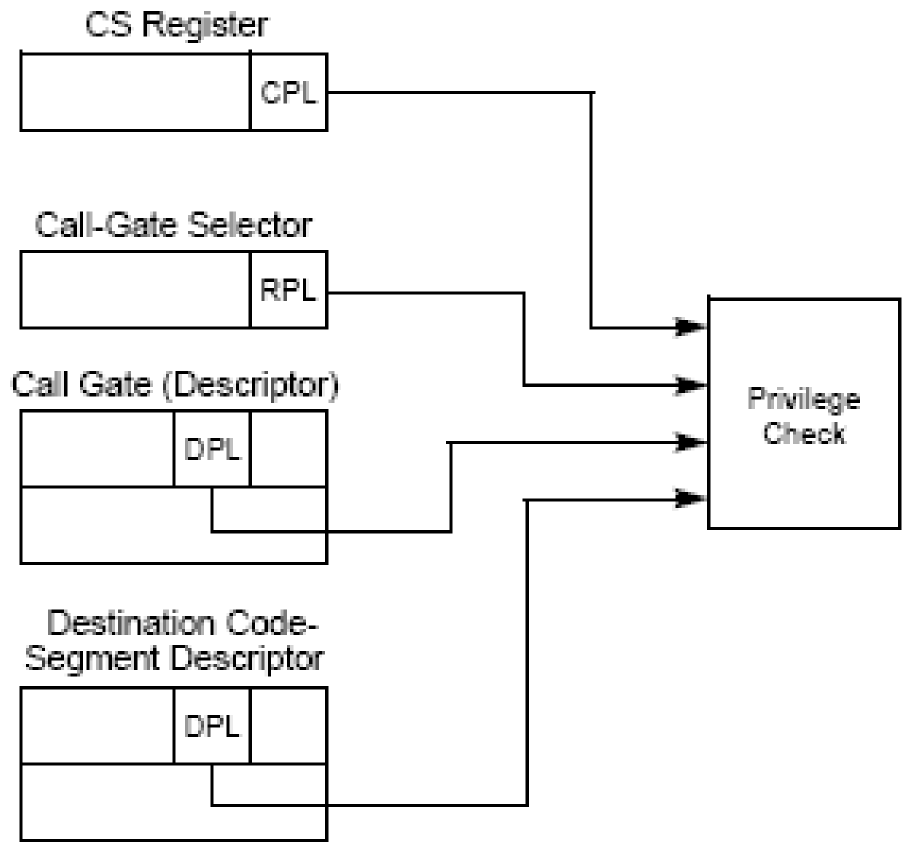


Calling Through Gates

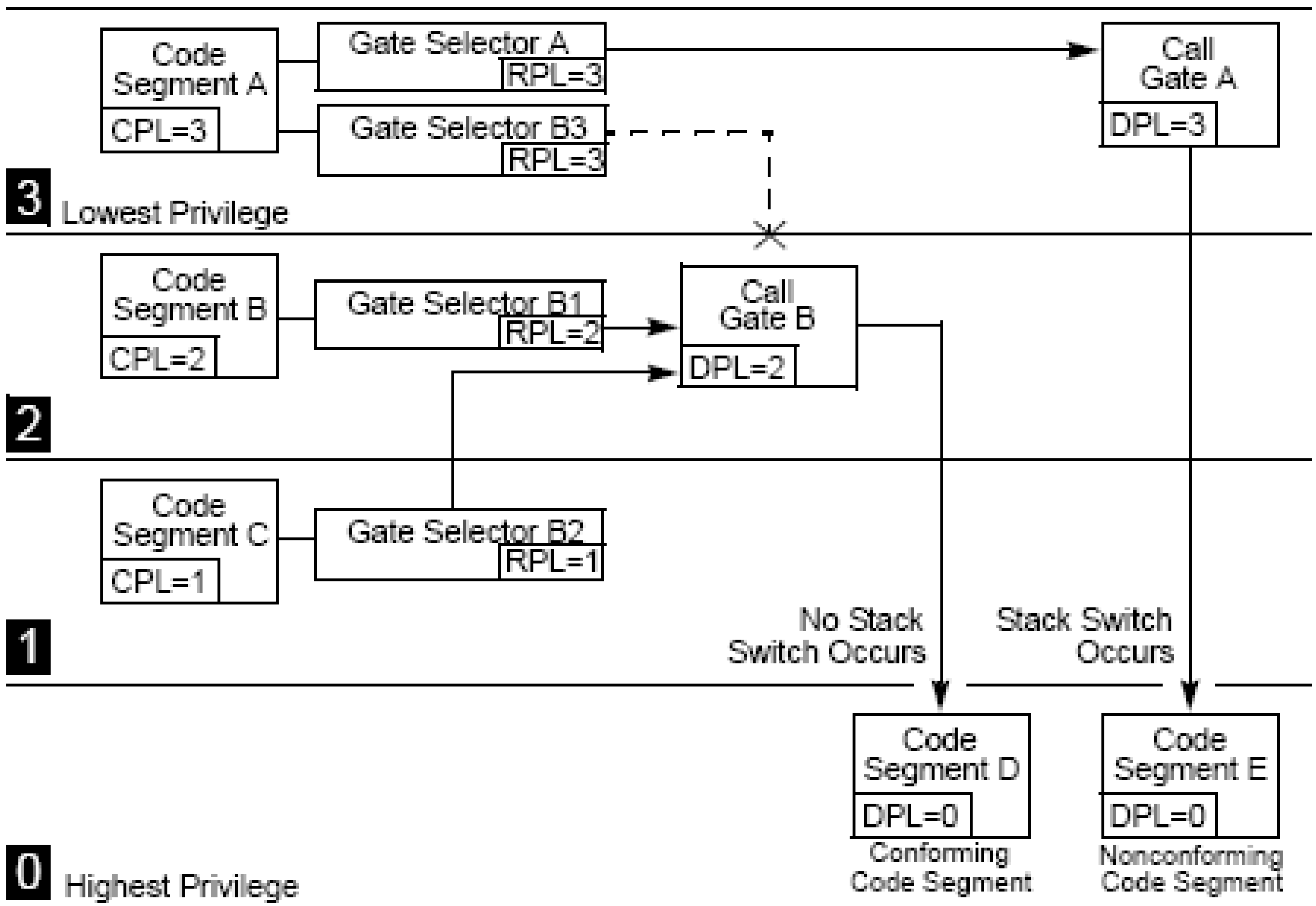


Call Gate Access Rules

- For Call
  - CPL <= CG DPL
  - RPL <= CG DPL
  - Dst CS DPL <= CPL
- Same for JMP but
  - Dst CS DPL == CPL



Call Gate Examples



Stack Switching

- Automatically performed when calling more privileged code
  - Prevents less privileged code from passing in short stack and crashing more privileged code
  - Each task has a stack defined for each privilege level

Hardware Rings

- Only most basic features generally used
  - 2 rings
  - Installed base
- Time to adoption
  - Must wait for widespread system code, e.g. Windows NT

Key Points

- Separation elements evolved in OS for safety as much as security
- Memory protections
  - Segments and pages and rings
  - HW support
- Object access control
  - File ACLs
  - Capabilities