DSCI 519: Foundations and Policy for Information Security

Integrity and Hybrid Policy Models

Tatyana Ryutov

Outline

- Reviw
- Biba Integrity Model
- RM Implementation Details (Multics)
  – Hardware protection rings
- Lipner integrity Model
- Clark-Wilson Model

Presentation7

USC Viterbi
School of Engineering

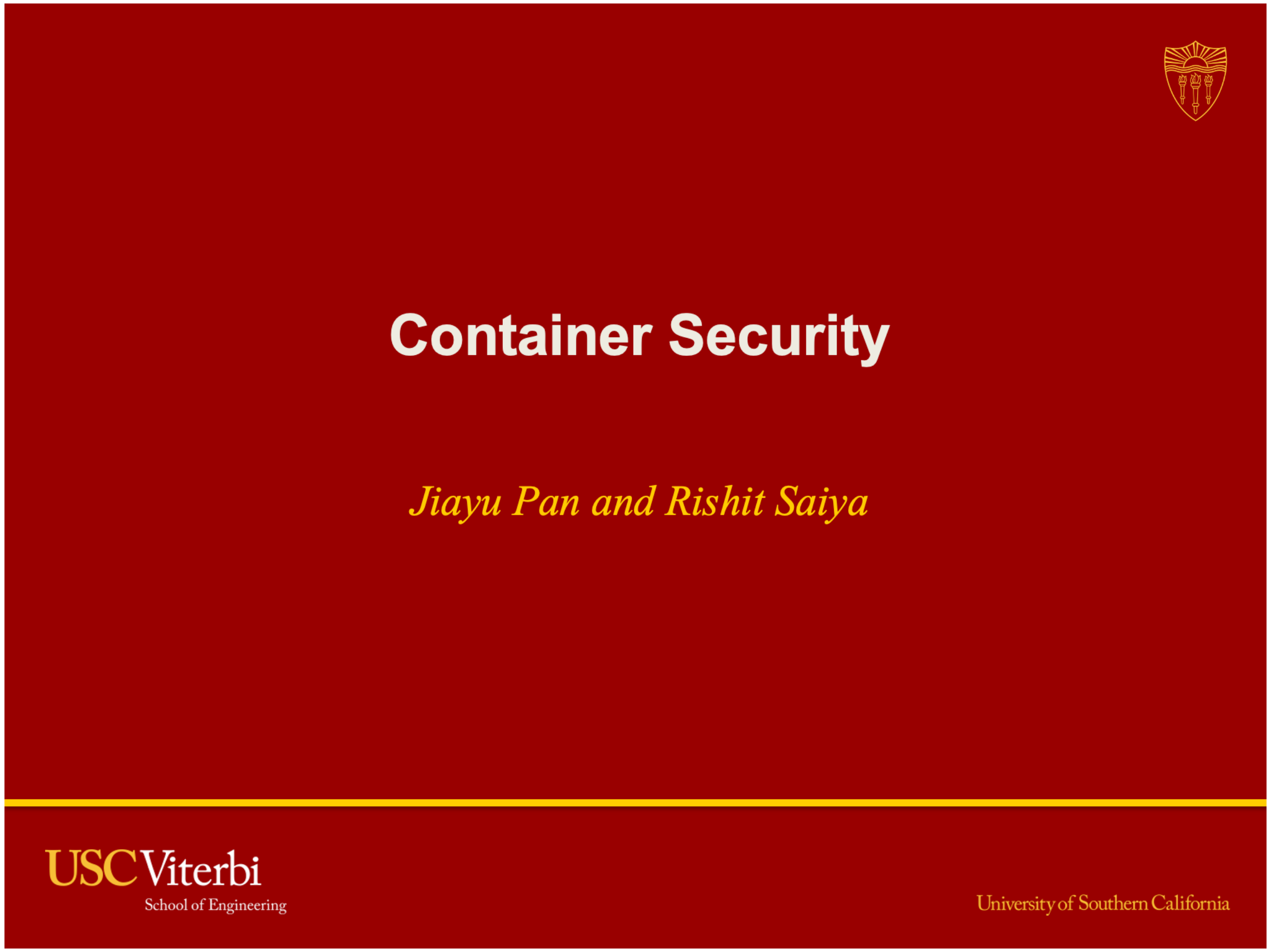
SOFTWARE GUARD EXTENSIONS

(SGX)

BY ARSHIA SHAIK & KEVIN SHAH

12th October, 2022

Presentation 8



Container Security

Jiayu Pan and Rishit Saiya

USC Viterbi School of Engineering

University of Southern California

USC Viterbi School of Engineering

L7.Q6

!

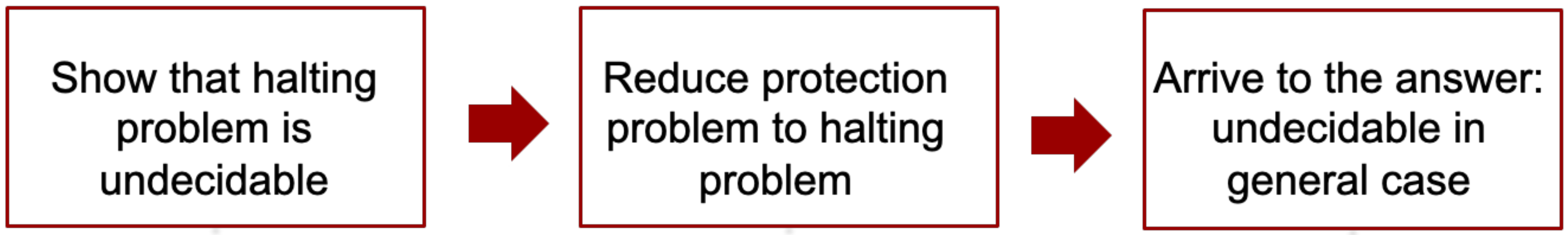
- Think about all the material covered today
- Indicate two specific topics/questions for review next lecture
  - focused questions/topics I can give you an answer to
- If everything is clear, it’s OK to say: “none”

Your Muddy Points

- Can we please go over how MLS and MAC are actually related?
- The concept of the safety problem and why it seems that proving that a system is secure isn't sufficient to guarantee safety
- I would like to know if unsafe means that the system is undecidable
- TM and HRU, I got a little confused during the decidable vs. undecidable discussion
- Reducing HRU to halting problem
- I do not understand the application of HRU

Review: Safety Question

- **Leaking**: adding a generic right r where none existed
- **Safe**: if a system cannot leak right r, it is safe with respect to right r
- **Safety Question** (a.k.a. Protection Problem): does an algorithm exist to determine whether a protection system is safe with respect to generic right r?
  - In general undecidable (Harrison, Ruzzo, and Ullman)
  - For the rules of the take-grant model can be computed in a time directly proportional to the size of the graph (linear)


Show that halting problem is undecidable
Reduce protection problem to halting problem
Arrive to the answer: undecidable in general case

Review: Mapping TM to HRU Protection System

- Mapping of TM to HRU:
  - The set of generic rights represent states and tape symbols
  - The moves of TM are represented as HRU commands

Turing Machine is a 6-tuple (Q, Σ, Γ, δ, q_0, q_f)

Generic HRU rights $\rightarrow$ Q is the set of states

$\rightarrow$ Σ is the input alphabet, does not include blank symbol

$\rightarrow$ Γ is the tape alphabet

HRU commands $\rightarrow$ δ is the transition function

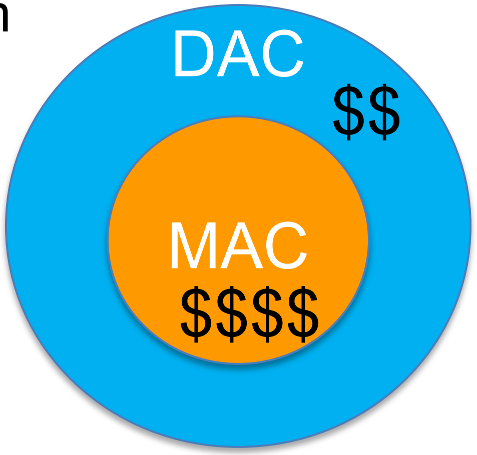
Initial HRU matrix $\rightarrow$ $q_0 \in Q$ is the starting state

Right is leaked $\rightarrow$ $q_f \in Q$ is the final, or halting state

- We showed that TM can be encoded as HRU protection system, so deciding whether TM enters final state q_f becomes equivalent to deciding that the HRU protection system leaks arbitrary right *r*
- Halting Problem: given an initial tape, a function δ and a final state q_f, will TM ever reach q_f?
  - We know this problem is undecidable, therefore the HRU safety question is undecidable

Review: Theoretical Limits on System Security

- Harrison, Ruzzo and Ullman (HRU) defined "safety" problem for protection systems
  - Safety refers to some **abstract** model
  - Security refers to **actual** implementation
- System can only be secure if it implements a policy based on a safe model
  - But a safe model does NOT ensure a secure system
- DAC has fundamental flow control limitation
  - It is generally unsafe model
  - Consistent with analysis of Trojan horse threat
- MAC can be safe
  - Imposes sufficient restrictions on right propagation
- How can we use this in practice?
  - Balanced assurance



Outline

- Review
- Biba Integrity Model
- RM Implementation Details (Multics)
  - Hardware protection rings
- Lipner integrity Model
- Clark-Wilson Model

Recall: Biba Integrity Problem

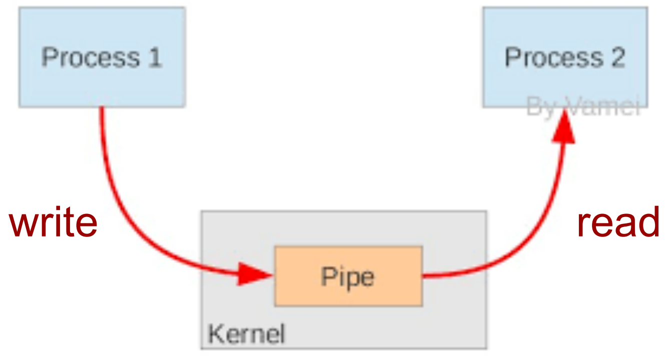
- Formulation of access control policies and mechanisms necessary for protection from **subversion**
  - Recall: Subversion—the intentional insertion of an artifice at some point during Software Development Life Cycle (SDLC)
    - Subversion favors a carefully hidden mechanism with a high likelihood of persistence subversion best meets the goals and objectives of the professional attacker
- Integrity is the guarantee that a subsystem will perform as intended by creator
  - Assume initially determined to perform properly
  - Must ensure that subsystem cannot be corrupted
- Does NOT imply guarantee of absolute behavior
- Does imply behavior consistent with a standard
  - Makes no statement about the quality of the standard

Biba Model Elements

- Much like BLP for confidentiality
  - Subjects access objects
  - Policy is defined as set of relations on subjects and objects
  - Decision rule determines access
- Mandatory policy has levels and categories
  - User max and “min necessary” clearances
  - Object classification based on possible damage due to **information sabotage**
  - Data at a higher level is more accurate, reliable, trustworthy than data at a lower level

Access Modes for Integrity Policy

- Observation (o)
  - allows a subject to read an object, synonyms with the read command of most other models
- Modification (m)
  - allows a subject to write to an object, similar to the write mode in other models
- Invocation (i)
  - allows a subject to communicate with another subject



USC Viterbi School of Engineering

Two Classes of Integrity Policy

- Discretionary integrity access control policy
  - Access authorizations may be <u>dynamically defined</u>
  - Based on individual identity
  - Includes “modify” access, but may also include software “update” privilege
- Mandatory integrity access control policy
  - <u>Static</u> access authorizations for the life of objects
  - Once defined for an object, is unchangeable
  - Must be satisfied for all states of the system

Abstraction for Integrity MAC Policy

- Define *clearance* of S and *classification* of O
  - Integrity “level” also called integrity “access class”
  - Each access class I describes a kind of information
- An integrity access class I has two components
  - An **integrity level** (L)
  - An **integrity category** set (C)
  - We write integrity access class as I(L, C)
- Integrity levels (L) form a total ordering
  - Can compare any two members , e.g., <, ≤, =, etc.
- Members of category set (C) are non-comparable

Integrity levels ≠ security (confidentiality) levels

Abstraction for Integrity MAC Policy

- Integrity MAC has set of rules comparing labels
- I(L, C) is *less than or equal (leq)* I(L', C')
  - If and only if (IFF) $L \leq L'$ and $C \subseteq C'$
  - Notation: access class (L', C') *leq* access class (L, C)
- *leq* induces lattice on set of access classes I
- Integrity access class of information is *global*
  - Has same integrity regardless of where it is
- Integrity access class of information is *persistent*
  - Always has same integrity, i.e., labels are tranquil

Abstraction for Integrity MAC Policy

- *leq* relation satisfies 3 standard conditions:
  - reflexivity
  - antisymmetry
  - transitivity
- For a set of labels x, y & z, and the relation *leq*
  - x *leq* x,
  - x *leq* y and y *leq* x implies x = y, and
  - x *leq* y and y *leq* z implies x *leq* z
- Labels that do not meet these conditions are not suitable for integrity MAC

Example of MAC Integrity Labels

- Consider integrity levels for software vendor
  released > beta > demo
- Consider set of integrity categories
  {internal, partner, customer}
- Example 1:
  For object $o \subseteq O$, $I(o) = (\text{released}, \{\text{partner}\})$
  For subject $s \subseteq S$, $I(s) = (\text{beta}, \{\emptyset\})$
  Then: I(s) *leq* I(o)
- Example 2:
  For object $o \subseteq O$, $I(o) = (\text{released}, \{\text{partner}\})$
  For subject $s \subseteq S$, $I(s) = (\text{beta}, \{\text{partner}, \text{customer}\})$
  Then: I(s) and I(o) are non-comparable

Biba: Mandatory Integrity Policies 1

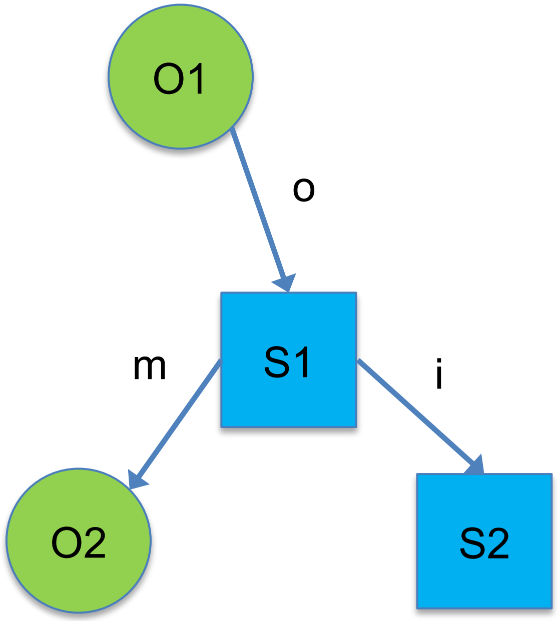
- Idea: subject can read down, but once it does, its integrity level drops (so it cannot corrupt higher integrity objects)
- **Low-water mark** policy
  1. Observation *allowed for any integrity level*
     - Integrity level of subject changes to the lowest integrity level of object observed
     - assume that the subject will rely on the data with lower integrity level, therefore its integrity level should be lowered
  2. For modify, I(o) leq I(s)
     - prevent writing to higher level, prevents passing of incorrect or false data
  3. For invoke, I($s_{invoked}$) leq I($s_{invoker}$)
     - prevent a less trusted invoker to control the execution of more trusted subjects

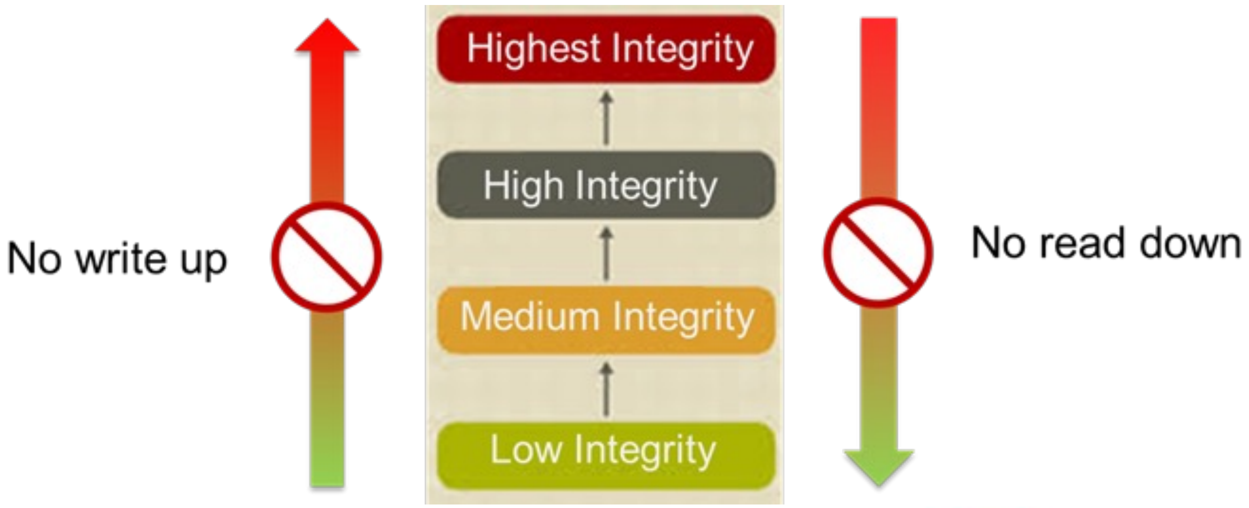
Biba: Mandatory Integrity Policies 2

- **Ring** policy:
  1. Observation *allowed for any integrity level*
  2. For modify, I(o) leq I(s)
  3. For invoke, I($s_{invoked}$) leq I($s_{invoker}$)
  - Does not address indirect modification
    - A subject can read a less trusted object, then the subject can modify data at its own integrity level
  - Subjects must provide internal validation controls

Biba: Mandatory Integrity Policies 3

- **Strict Integrity Policy**, most similar to BLP
  - Simple Integrity Condition:
    - For observe, I(s) leq I(o) ("no read down")
  - Integrity Star Property:
    - For modify, I(o) leq I(s) ("no write up")
  - Invocation Property:
    - For invoke, $I(s_{invoked})$ leq $I(s_{invoker})$



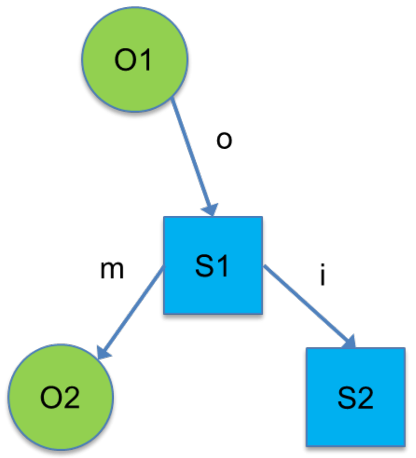


Biba Strict Integrity Policy Interpretation

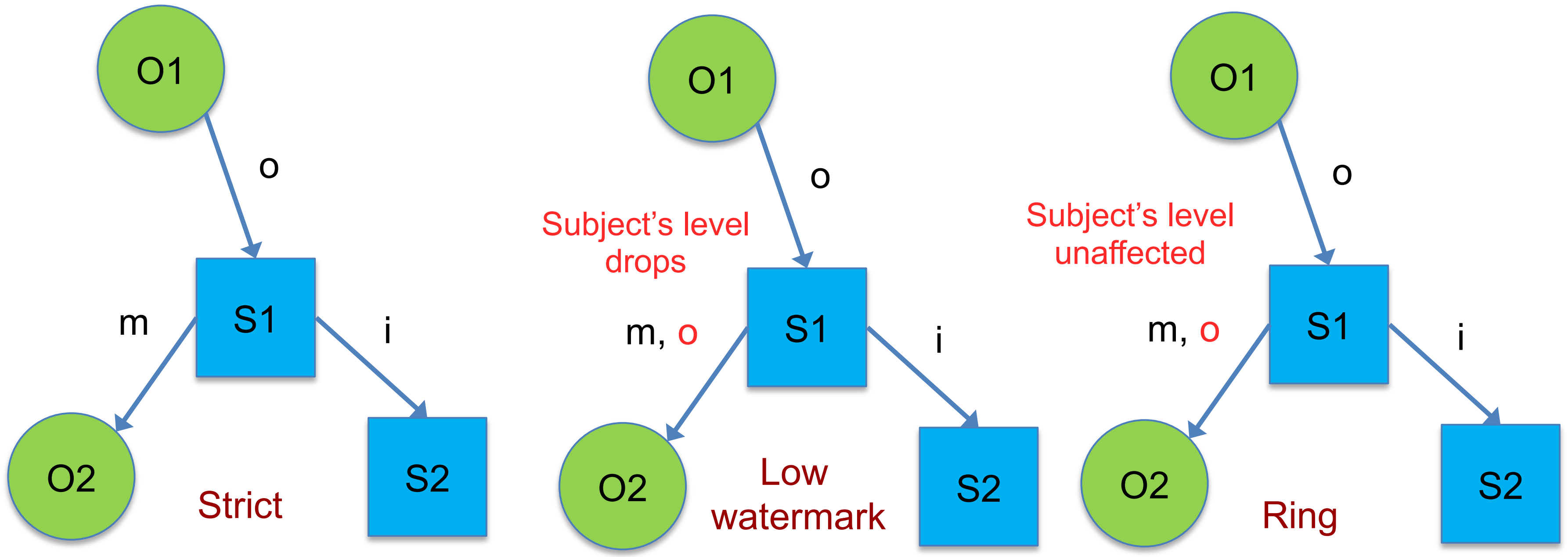
- Define dual of BLP *-Property:
  - BLP: S can write O IFF O dom S (“no write down”)
  - Biba: S can write O IFF i(O) *leq* i(S) (“no write up”)
- No unauthorized direct modification of objects
  - Limit write access to subjects of sufficient privilege
  - Integrity access class reflects damage from sabotage
  - Limits the amount of damage that can be done by a Trojan horse in the system

Strict Integrity Policy Examples

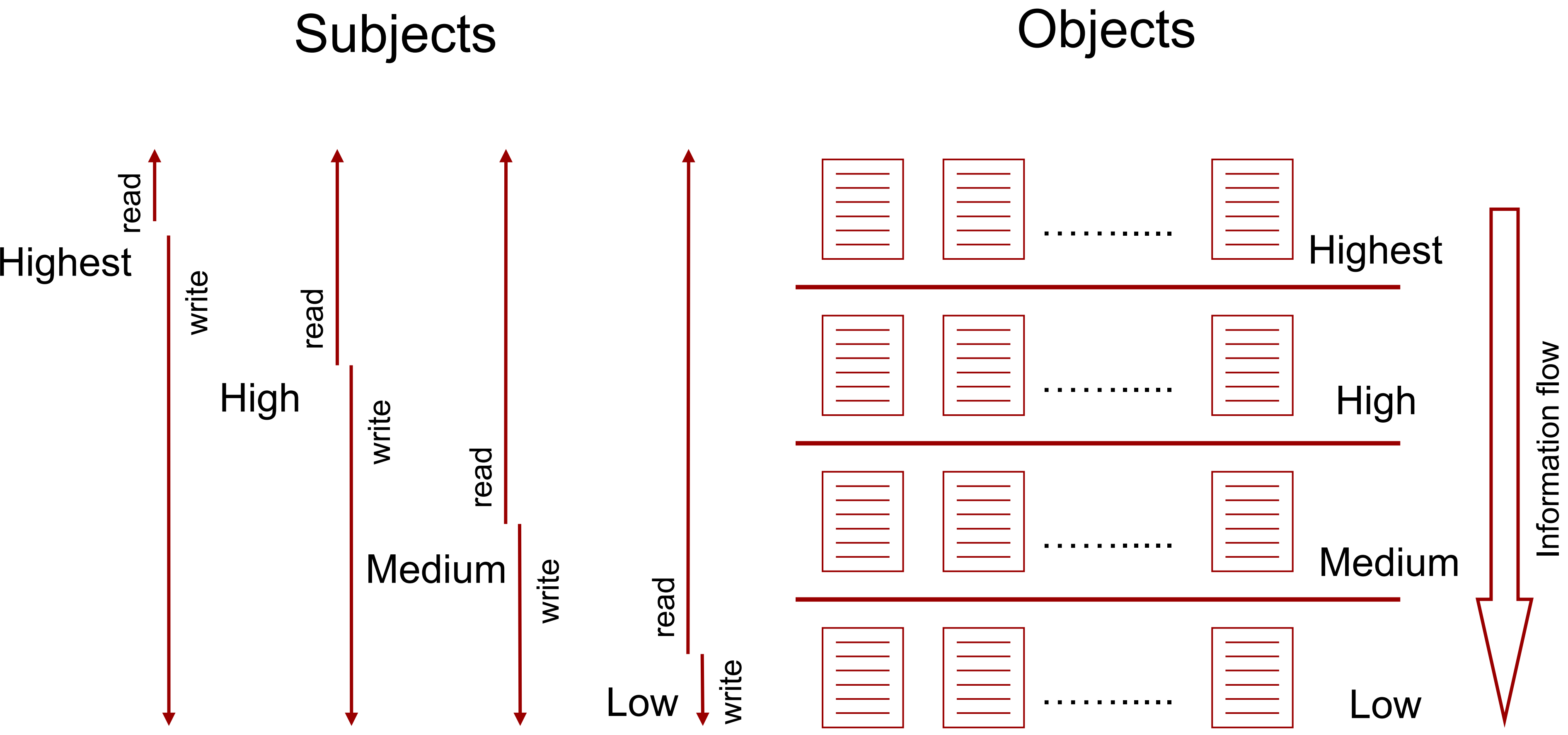
- Consider SW vendor integrity access classes
  - Levels: released > beta > demo
  - Categories: {internal, partner, customer}
- Consider **object** with integrity class (beta, {internal, partner})
- Example: what observe/modify access for these **subjects**:
  (beta, {internal, partner})
  (released, {internal, partner})
  (demo, {internal, partner})
  (beta, {internal})
  (beta, {internal, customer})



Biba Models

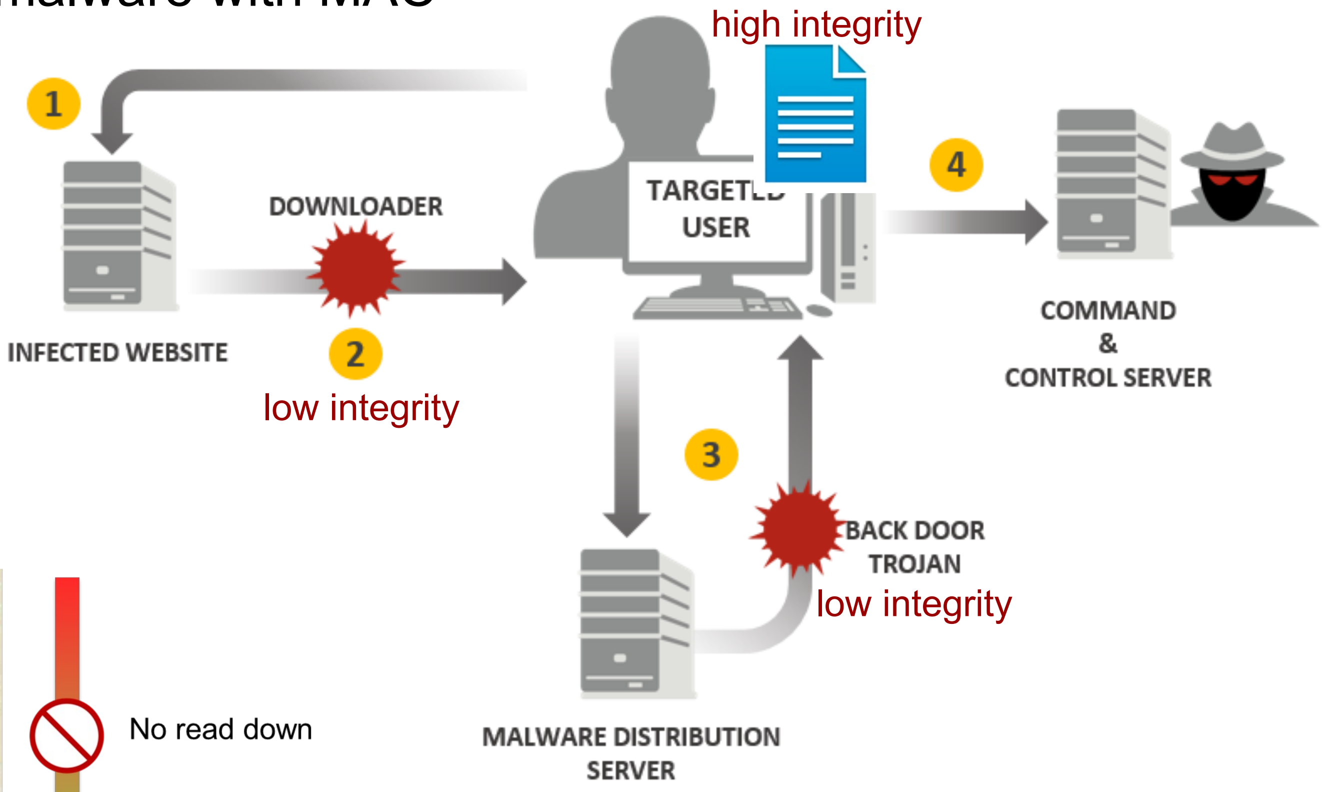

O1
o
m
S1
i
O2
Strict
S2
O1
o
Subject's level drops
m, o
S1
i
O2
Low watermark
S2
O1
o
Subject's level unaffected
m, o
S1
i
O2
Ring
S2

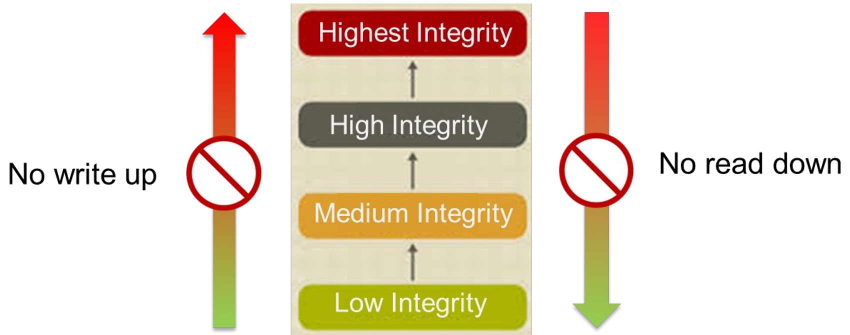
Biba Information Flow



How can we use Biba Strict Integrity model in practice?

- Infection injects malicious code into the system by modifying code/data
  - Example: VPNfilter 2018 attack
    - Infection stage adds code to the device's `crontab` (the list of tasks run at regular intervals by the `cron` scheduler)
- Solution: contain malware with MAC





Example: PitBull Mandatory Integrity Control (MIC)

- PitBull utilizes Mandatory Integrity Control (MIC)
  - Integrity Label (**`xattr`**) applied to files
    - Running at Kernel LSM provides adherence to the Biba model
    - Orthogonal and independent to enforcement of MAC Policy
  - PitBull provides the **`chkintegrity`** tool for checking alteration of data
    - 16 byte (128 bit) Hashed Check Block (HCB)
      - Note: CRC 16 is a simple integrity check, it does not check for alteration of data and is not FIPS 140 2 compliant
    - Run during system startup or by ISSO user
- Both labels (MAC) and integrity (MIC) are checked Kernel LSM
  - SELinux relies on an external tool, such as AIDE or Tripwire, to provide integrity checks
  - PitBull can utilize AIDE or Tripwire in addition to its native MIC and **`chkintegrity`** tool

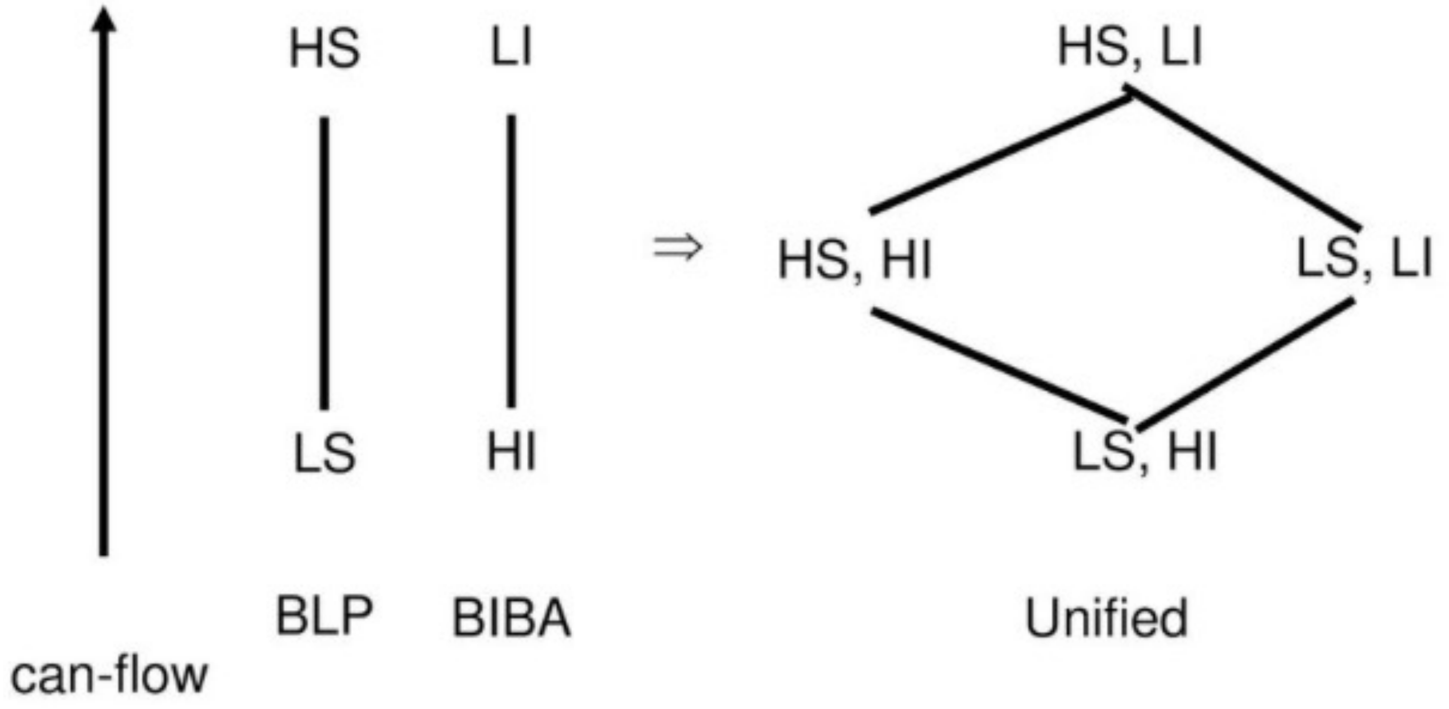
Integrity vs. Confidentiality

Confidentiality	**Integrity**
Control reading preserved if confidential info is not read	Control writing preserved if important object is not changed
For subjects who need to **read**, control writing after reading is sufficient, no need to trust them	For subjects who need to **write**, have to trust them, control reading before writing is not sufficient

Integrity requires trust in subjects!

BLP-Biba Unified Lattice

- Lattice-base control is a mechanism for enforcing one-way flow which can be applied to confidentiality or integrity goals



One directional flow (no need for opposite directions for confidentiality and integrity)

Combined BLP-Biba Access Matrix Example

Confidentiality labels: $\{S_H, S_L\}$, $S_L > S_H$
Integrity labels: $\{I_H, I_L\}$, $I_H > I_L$

Read rules: S_S *dom* S_O (BLP) and I_S *leq* I_O (Biba)
Write rules: S_O *dom* S_S (BLP) and I_O *leq* I_S (Biba)

objects

subjects

	S_L, I_L	S_L, I_H	S_H, I_L	S_H, I_H
S_L, I_L	rw	r	w	-
S_L, I_H	w	rw	w	w
S_H, I_L	r	r	rw	r
S_H, I_H	-	r	w	rw

Biba Integrity Model Summary

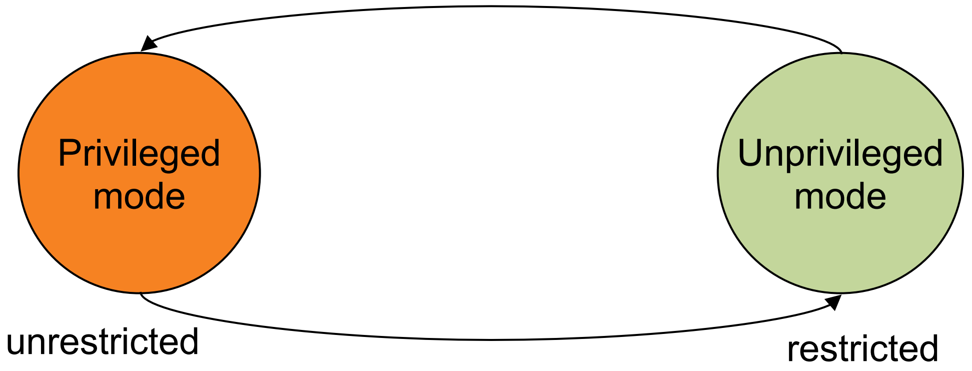
- The Biba model is a family of different models that can be selected
- The model should be combined with another model to address confidentiality
  - The Lipner model combines both the Bell-LaPadula and Biba models
- Biba model is currently used is in FreeBSD 5.0
  - https://www.freebsd.org/cgi/man.cgi?mac_biba
- Weaknesses:
  - Flow controls may be too restrictive
  - Integrity problem is more than restricting information flow
    - For example: concurrency control, integrity constraints (restrictions on values)

Outline

- Review
- Biba integrity model
- RM Implementation Details (Multics)
  - Hardware protection rings
- Lipner integrity Model
- Clark-Wilson Model

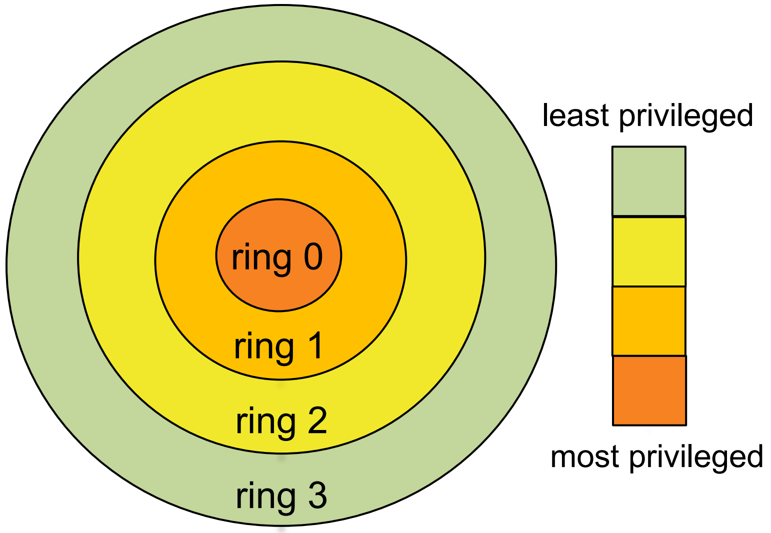
Dual Mode of Operation

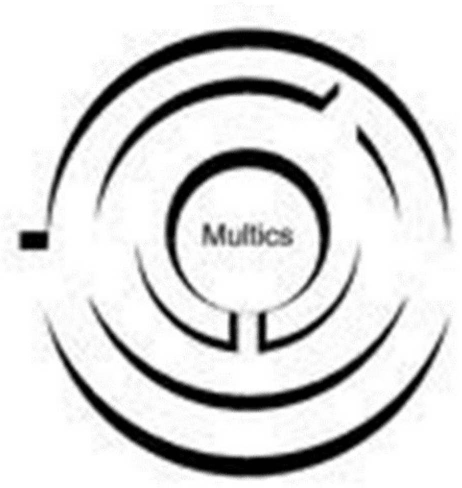
- We must distinguish between the execution of OS code and user code
- Most processors support dual mode of operation
  - Privileged (kernel) mode
    - Special mode of the processor for executing trusted OS code
      - Execution with the full privileges of the HW
    - No memory access limits
  - Unprivileged (user) mode
    - Limited privileges
      - Only those granted by the kernel
    - Limits on memory accesses



Multics Protection Rings

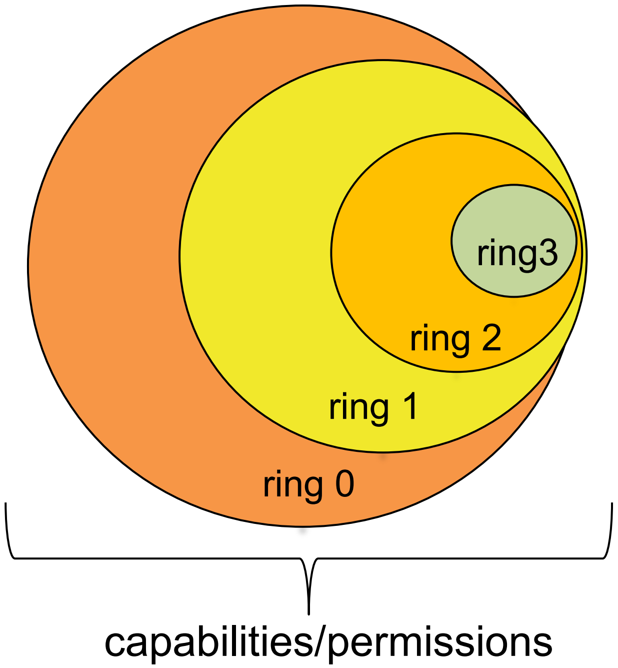
- Multics supported an extension of the privileged/unprivileged mode idea via an access control mechanism based on a *ring* structure (64 rings in theory, 8 rings in practice)
- Each mode was called a **ring**, and was associated with a set of access rights defined by ring policy
- The rings were hierarchically arranged: the innermost ring had the most power and the outermost ring the least

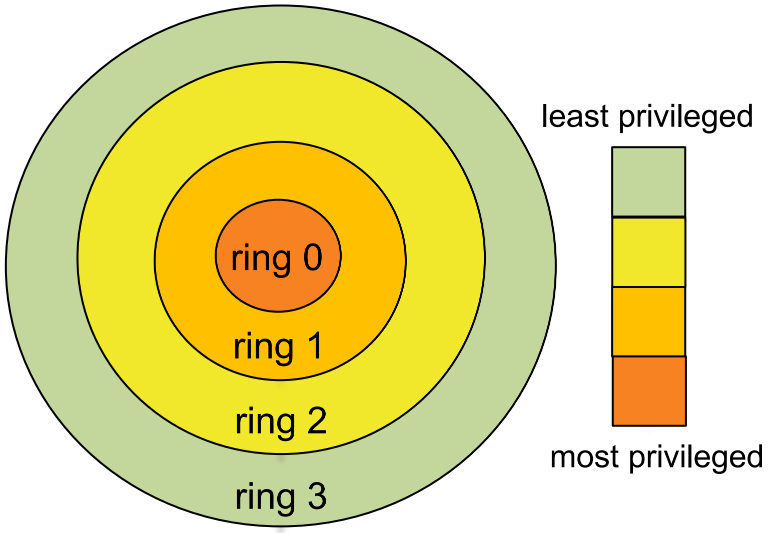


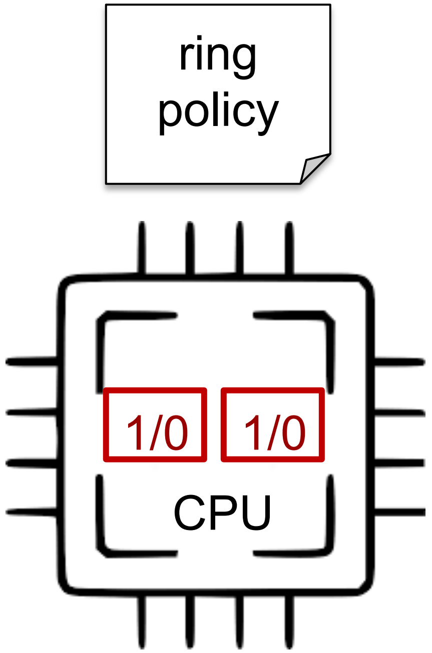


HW Protection Rings

- **HW protection rings** - generalization of the privileged/unprivileged modes in most processors (pioneered by Multics)
- Define a fixed number N of hierarchical domains (called protection rings)
  - These N rings are named by the integers 0 through N – 1
  - The access capabilities included in ring m are constrained to be a subset of those in ring n whenever m > n
- Privilege separation requires HW support

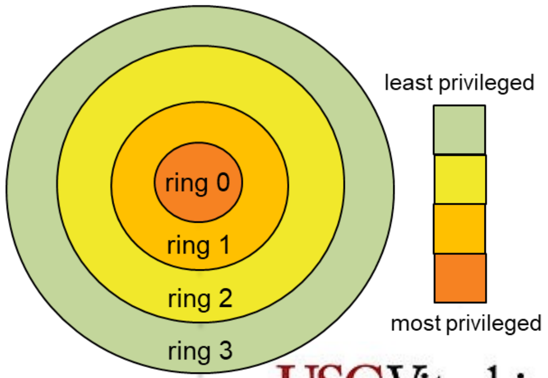






Ring-Based Access Control

- Security kernel (RM implementation) resides in ring 0
- User processes execute in one ring at a time
  - With each process a current ring number counter was associated
  - Access is based on the current ring as defined by the ring policy
  - Subject" is (process, ring) pair
- State of a process is defined by
  - Execution point: program (instruction) counter
  - Current ring number
    - Changing rings can cause HW trap to the kernel
  - Address space
- Segments can only be accessed within a range of rings (called ring bracket)
  - E.g., kernel segments only accessible in ring 0
- Current process ring number is compared to segment ring access range to enforce ring policy
  - Gatekeeper mechanism constrains ring crossings
  - Gates used to transfer between rings



USC Viterbi School of Engineering

Motivation for Ring Brackets

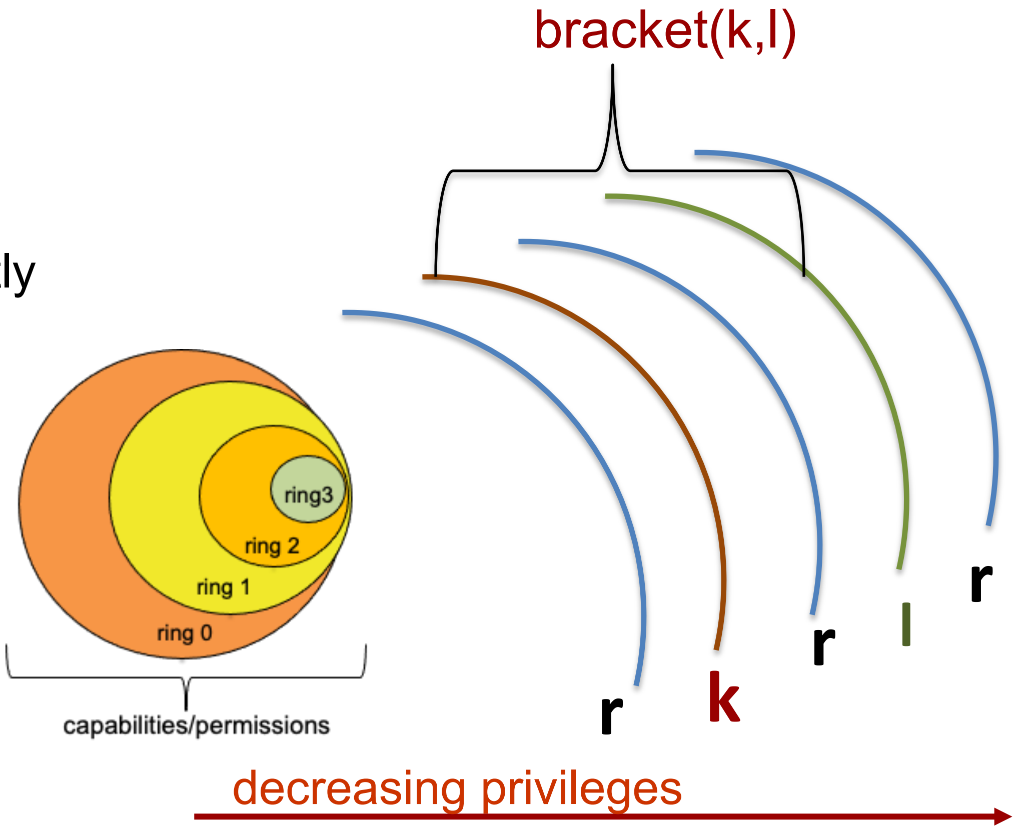
- In principle, process or segment can be assigned to one ring
- But this would require an OS call <u>every time</u> a process from one rings needs access to a segment in a different ring
- Need <u>range of rings </u>(a ring bracket) for access
- Access brackets minimize OS overhead
- Ring bracket are assigned to a segment when the segment is created

Segments in Multics

- Segments on disk and in memory are treated the same from the protection point of view
  - There is no conceptual difference between a segment of memory and a segment on a disk
- Access rights:
  - r (read)
  - w (write)
  - e (execute)
  - a (append)
- Segment types*:*
  - **Data** – not executable (will cause fault if try)
  - **Procedure** – executable, but also “read” access

Access Bracket for Segments

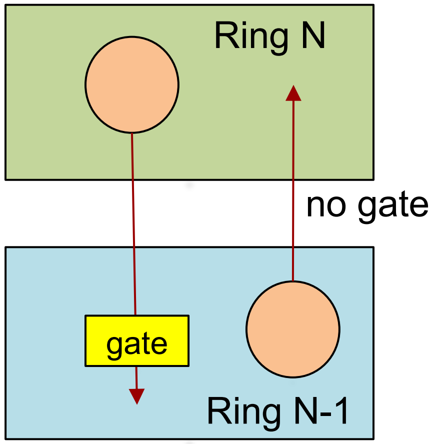
- Describes read and write access to data and procedure segments
- Each data and procedure segment has an **access bracket *(k, l)***
  - Pair of ring numbers with $k \leq l$
  - For subject in ring r :
    - $r \leq k$: r and w access permitted
    - $k < r \leq l$: r permitted; w denied
    - $l < r$: all access denied
  - Such policy ensures that lower (more privileged) rings have strictly greater access to segments than higher (less privileged) rings



Assume that a procedure executing in ring r wants to access a data segment with access bracket (k,l)

Invocation of **Procedure** Segments

- Switching the ring of execution to a lower number makes additional access capabilities available to a process
  - Switching the ring to a higher ring reduces the available access capabilities
- Thus downward ring switching capability must be controlled!
  - User process cannot call code of higher privilege directly
  - Gate extension is defined
- Gate is a special memory address where a lower privilege code can call higher privileged code
  - Gates are specified by associating a (possibly empty) list of entry points at which the segments may be called (executed)



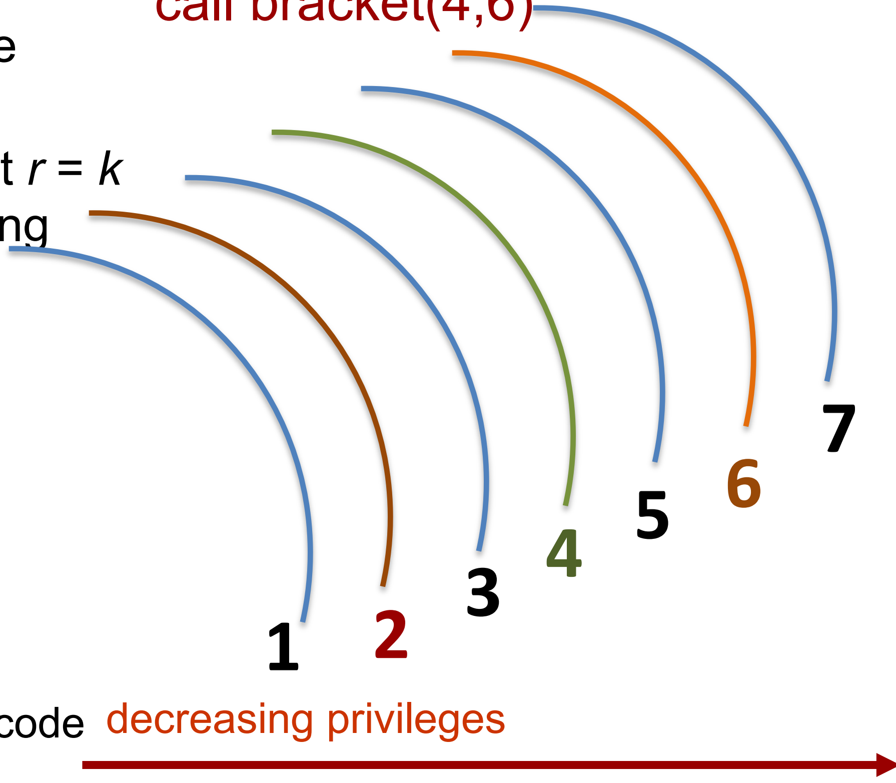
Call Bracket for **Procedure** Segments

- To provide control of **downward** ring switching capability, a gate extension to the execute bracket of a segment is defined
- Supports “transfer to a gate and change ring" capability for the segment
- Each procedure segment has a third ring number *m,* where $l \leq m$
- ($l + 1$, m) is called the “call bracket”
  - Sometimes called “gate extension”
  - $l = m$ means no call bracket; only access bracket
- Call bracket limits invocation of subsystems
- Call bracket is used (instead of access bracket) when “execute” is requested

Complete Bracket for **Procedure** Segments

- Complete bracket is (*k, l, m*):
  - (k,l) – access bracket
  - (l,m) – call bracket
- Subject in ring *r* tries to invoke a procedure segment with complete bracket (*k, l, m*):
  - *r* < *k*: call permitted, ring fault occurs; set *r* = *k*
  - *k* ≤ *r* ≤ *l* : call permitted, no fault, same ring number r
  - *l* < *r* ≤ *m* : call permitted through **gate** & transition to more privileged ring
  - *m* < *r* : call is denied
  - Requires **gate**
    - restricted start location (entry point) in procedure segment
    - a special memory address where lower-privileged code can call more privileged code

bracket(2,4,6)
access bracket (2,4)
call bracket(4,6)



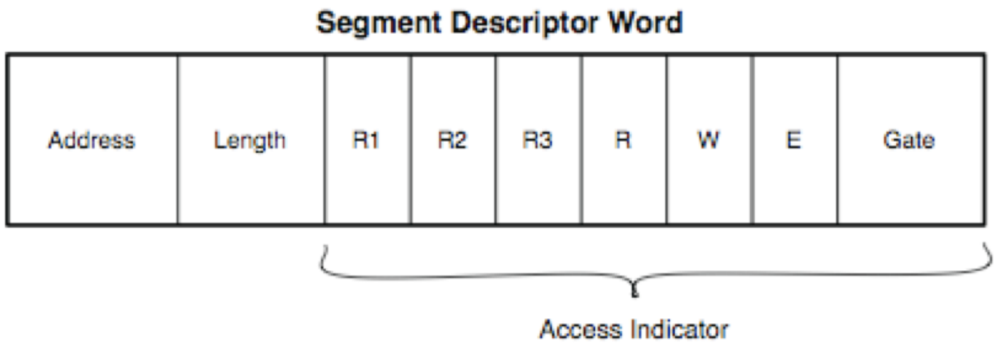
Do not want to execute lower integrity subject in more privileged ring, so current ring number for the process is set to k (less integrity) like Biba low watermark policy

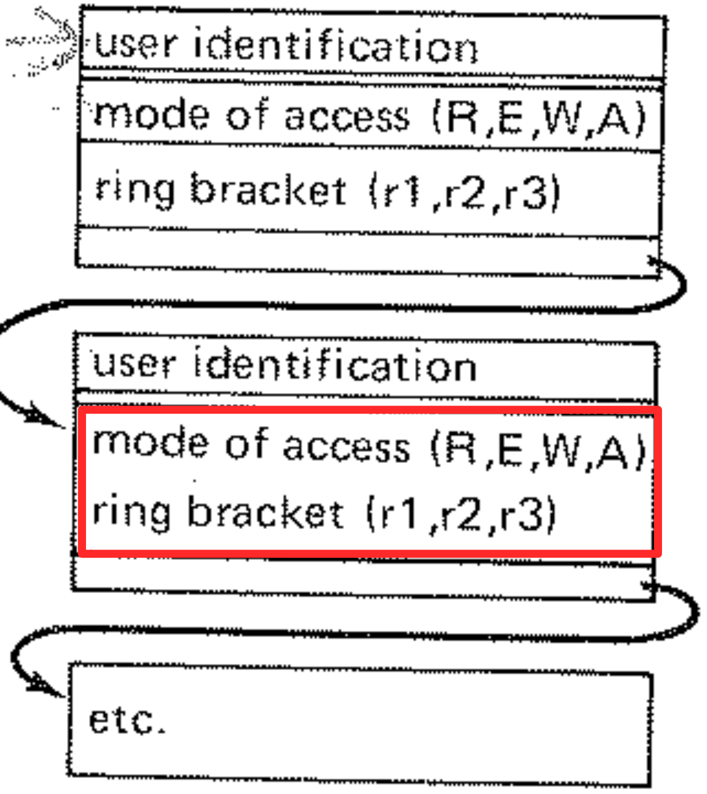
Question

- Why is there a difference between access bracket and call bracket?

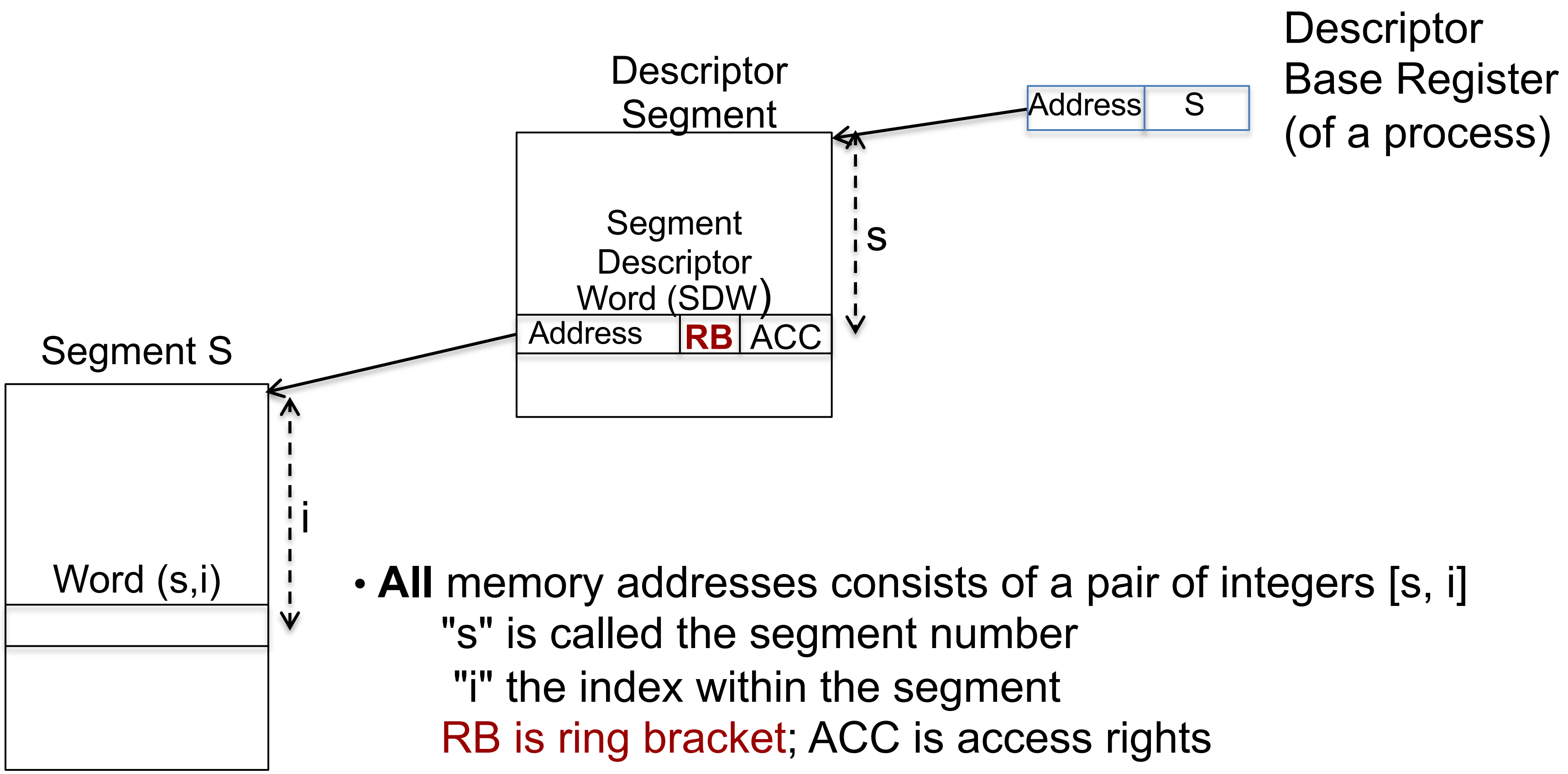
Protection Rings in Multics

- System-wide segment name is a path name
  - Directory stores ACL and ring bracket for a segment
  - Access bits and ring bracket info are copied into SDW
  - Each user can have different rings
  - Discretionary controls: owner can determine the rings, and can change them





Protection Rings in Multics 2


Descriptor Base Register (of a process)
Address
S
Descriptor Segment
s
Segment Descriptor Word (SDW)
Address
RB
ACC
Segment S
i
Word (s,i)

- **All** memory addresses consists of a pair of integers [s, i]
  - "s" is called the segment number
  - "i" the index within the segment
  - RB is ring bracket; ACC is access rights
- Key issue is making all checks *at hardware speed*

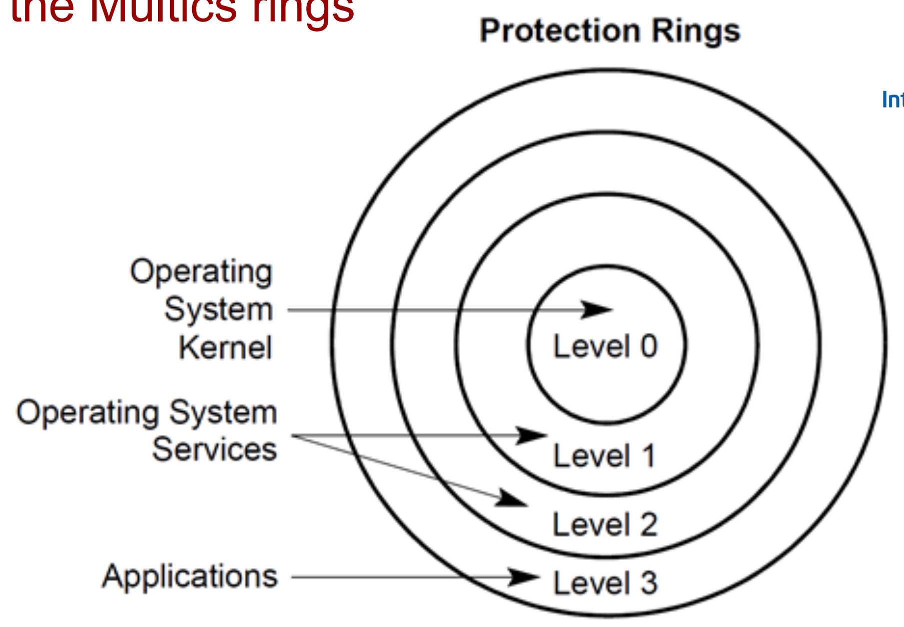
Multics Protection Summary

- When a segment is requested, all three policies must authorize the request for it to be allowed
- **Read request**:
  - MLS policy is checked to verify simple security-property
  - ACL is checked to determine if the user has access
  - Access bracket is checked to determine whether the process has read access to the object's segment
- **Write request**:
  - MLS policy is checked to verify *-property
  - ACL is checked for write access
  - Access bracket must permit the current ring write access
- **Execute request**:
  - Handled similarly to read request, except the call bracket is used instead of the access bracket
- Request may result in a protection ring transition (higher or lower)

Intel Protection Rings

- A very similar call gate mechanism for ring transition as was used in Multics is available in all Intel x86 systems supporting "protected mode", from 80286 on

Influenced by the Multics rings



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3A:
System Programming Guide, Part 1

Figure 5-3. Protection Rings

Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 3A:
System Programming Guide, Part 1
See section 5.5 Privilege Levels
https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.pdf

Multics Security Evaluation

- Multics RM implementation
  - Isolation:
    - 👍 RM implemented in kernel, runs in ring 0 protected by HW
    - 👎 But: ring brackets are discretionary, can be modified by any process with the appropriate privileges
  - Completeness:
    - 👍 HW support, checks on each access → non bypassable
      - SDW is checked on every reference
    - 👎 Unless one can temper with the kernel code (e.g., add code to skip/disable access control checks)
      - Possible due to the discretionary nature of ring management
  - Verifiability:
    - 👎 Trusted Computing Base (TCB) was large (54 LoC), and complex (DAC and MAC in the kernel, loops in dependencies)
    - Cannot formally verify the system
      - B2 rating in mid 1980s

MULTICS System

- Multiplexed Information and Computing Service (MULTICS)
- Confidentiality: BLP
- Integrity: HW rings of protection
- Multics meets the requirements of Class B2
  - https://multicians.org/multics-fer.html
- Resulting system considered a high point in trusted system design

Outline

- Review
- Biba integrity model
- RM Implementation Details (Multics)
  - Hardware protection rings
- Lipner integrity Model
- Clark-Wilson Model

USC Viterbi
School of Engineering

Requirements of Commercial Integrity Policies (Lipner 1982)

1. Users will not write their own programs, but use existing production software
2. Programmers develop and test applications on a non-production system using contrived data
3. Moving applications from development to production requires a special process
4. This process must be controlled and audited
5. Managers and auditors must have access to system state and system logs

Principals of Operation

- Separation of duty:
  - If two or more steps are required to perform a critical function, at least two people should perform the steps
  - E.g., a different person installs program on a production system
- Separation of function:
  - Do not use production system for development
  - Do not process production data on development system
- Auditing:
  - Commercial systems emphasize recovery and accountability
  - Ability to analyze who did what

Lipner’s Lattice (BLP + Biba)

- A realistic example showing that BLP and Biba can be combined to meet commercial requirements
- How does it combine BLP and Biba?
  - Uses disjoint sets of security levels and integrity levels
  - BLP is used first, and add Biba only when necessary

USC Viterbi
School of Engineering

Lipner's Lattice (BLP + Biba)

- BLP component for confidentiality:
  - 2 security clearances/classifications
    - AM (Audit Manager): system audit, management functions
    - SL (System Low): everything else, any process can read at this level
  - 3 Security categories
    - SP (Production): production code, data
    - SD (Development): production programs under development and testing, but not yet in production use
    - SSD (System Development): system programs under development
  - Security level = (classification, category)
- Biba component for integrity:
  - 3 integrity classifications
    - ISP (System Program): for system programs
    - IO (Operational): production programs, development software
    - ISL (System Low): users get this on log in
  - 2 integrity categories
    - ID (Development): development entities
    - IP (Production): production entities
  - Integrity level = (classification, category)

Subjects' Levels (Clearance)

Subjects	Security Level	Integrity Level
Ordinary users	(SL, { SP })	(ISL, { IP })
Application developers	(SL, { SD })	(ISL, { ID })
System programmers	(SL, { SSD })	(ISL, { ID })
System managers and auditors	(AM, { SP, SD, SSD })	(ISL, { IP, ID})
System controllers	(SL, { SP, SD }) and downgrade privilege	(ISP, { IP, ID})
Repair	(SL, { SP })	(ISL, { IP })

Exempt from *-property: allowed to write down (with respect to confidentiality) and write up (with respect to integrity)

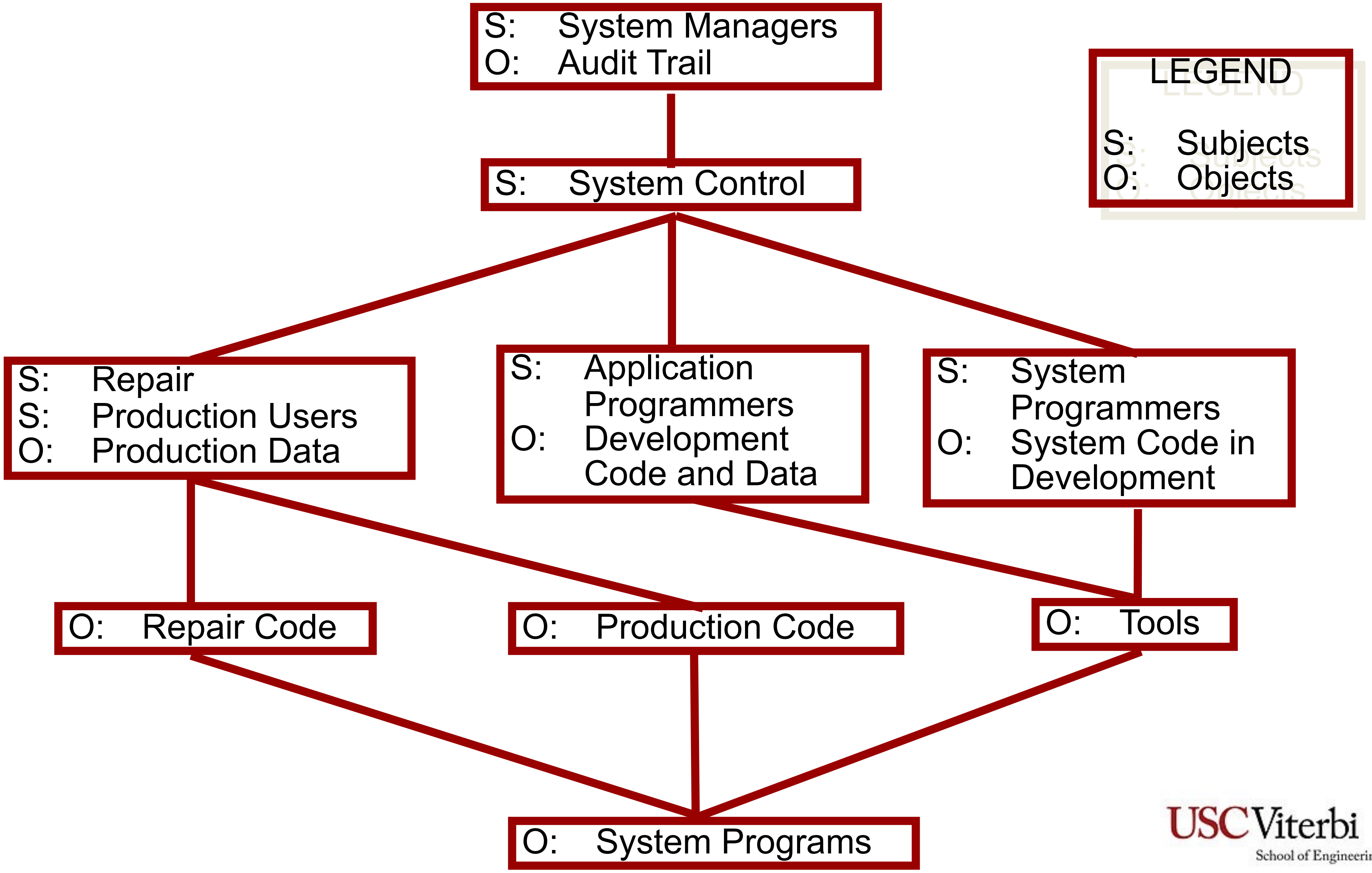
Objects' Levels (Classification)

Objects	Security Level	Integrity Level
Development code/test data	(SL, { SD })	(ISL, { IP})
Production code	(SL, { SP })	(IO, { IP })
Production data	(SL, { SP })	(ISL, { IP })
Software tools	(SL, ∅)	(IO, { ID })
System programs	(SL, ∅)	(ISP, { IP, ID })
System programs in modification	(SL, { SSD })	(ISL, { ID })
System and application logs	(AM, { *appropriate* })	(ISL, ∅)
Repair	(SL, {SP})	(ISL, { IP })

The position of system logs at lowest integrity demonstrates the limitation of info flow policy for integrity

USC Viterbi School of Engineering

The Lipner Lattice



What does the Lipner Lattice achieve?

- Ordinary users can execute (read) production code but cannot alter it
- Ordinary users can alter and read production data
- System managers need access to all logs but cannot change levels of objects
- System controllers need to install code (hence downgrade capability)
- Subjects need to have append-only access to logs
- These meet the stated requirements

Example: What can an ordinary user do?

- Ordinary users **(SL, { SP }) (ISL, { IP })** can:
  - Read and write production data **(SL, { SP }) (ISL, { IP })**
    - same security integrity levels
  - Read production code **(SL, { SP }) (IO, { IP })**
    - same classification and (ISL, {IP}) *leq* (IO, { IP })
  - Read system program **(SL, ∅) (ISP, { IP, ID })**
    - (SL, {SP}) *dom* (SL, ∅) and (ISL, {IP}) *leq* (ISP, {IP,ID})
  - Repair objects **(SL, { SP }) (ISL, { IP })**
    - same levels
  - Write (not read) system and application log **(AM, { SL }) (ISL, ∅)**
    - (AM, {SP}) *dom* (SL, {SP}) and (ISL, ∅) *leq* (ISL, {IP})

Outline

- Review
- Biba integrity model
- RM Implementation Details (Multics)
  - Hardware protection rings
- Lipner integrity Model
- Clark-Wilson Model

Clark-Wilson Integrity Model,1987

- Time-proven accounting practices generalized
- Goal: prevent illegal data modification due to fraud and error
- Validation of integrity is done to ensure that:
  - The data is being modified is valid
  - The results of the modification are valid
- Integrity policy is given as high-level rules: a set of constraints
  - Data in a consistent state when it satisfies the constrains
- Example: Bank
  - Objective: today's deposits - today's withdrawals + yesterday's balance = today's balance

Clark-Wilson (CW): Two mechanisms for enforcing Integrity

1. Well-formed transaction
   - Operations that move system from one consistent state to another
     - State before transaction consistent $\Rightarrow$ state after transaction consistent
   - Can manipulate data only through trusted code!
   - Sufficient for ensuring internal consistency, but insufficient for ensuring consistency with physical world
2. Separation of duty
   - Ensure external consistency: data objects correspond to the real world objects
   - Separating all operations into several subparts and requiring that each subpart be executed by a different person
   - E.g., the two-man rule

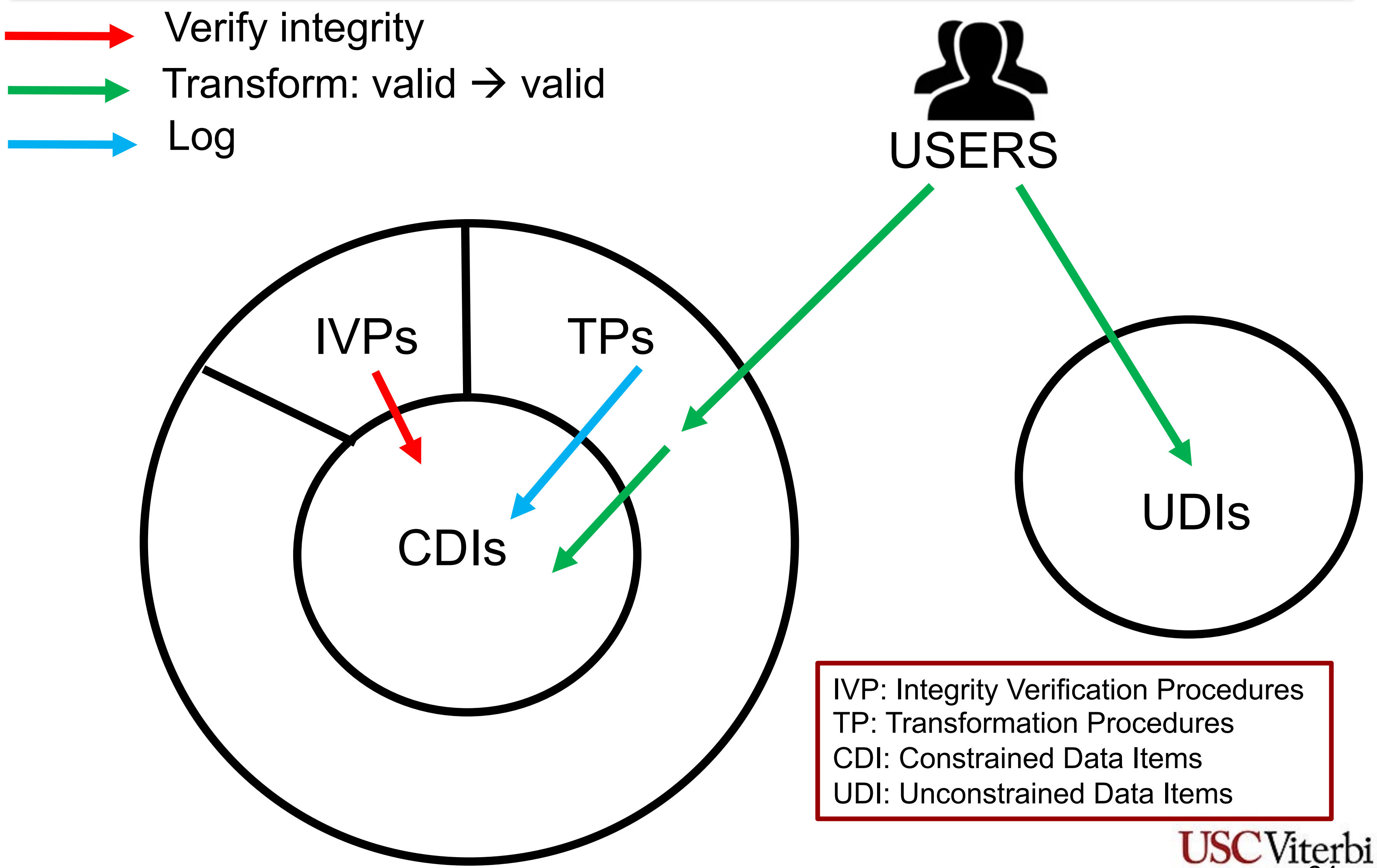
Implementing the Two High-level Mechanisms

- Mechanisms are needed to ensure
  1. controlled access to data: a data item can be manipulated only by a specific set of programs
  2. program certification: programs must be inspected for proper construction, controls must be provided on the ability to install and modify these programs
  3. controlled access to programs: each user must be permitted to use only certain sets of programs
  4. controlled administration: assignment of people to programs must be controlled and inspected

CW Model Elements

- Users – active agents
- **T**ransformation **P**rocedures (**TP**)
  - abstract operations, e.g., debit, credit
  - implement well-formed transactions
- **C**onstrained **D**ata **I**tems (**CDI**): data subject to integrity control
  - Can only be manipulated by Transformation Procedures
- **U**nconstrained **D**ata **I**tems (**UDI**): data not subject to integrity controls
  - Can be manipulated by via primitive read and write operations
- **I**ntegrity **V**erification **P**rocedures (**IVP**)
  - Test CDIs' conformance to integrity constraints at the time IVPs are run
    - E.g., checking account balances

CW Elements Interactions



IVP: Integrity Verification Procedures
TP: Transformation Procedures
CDI: Constrained Data Items
UDI: Unconstrained Data Items

Rules at a Glance

Certification Rules

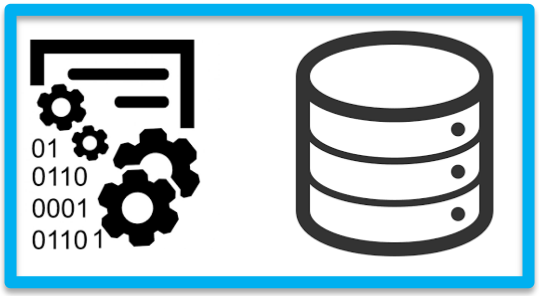
C1	IVPs verify CDI integrity
C2	TPs preserve CDI integrity
C3	Separation of duties for ER2
C4	TPs write to log
C5	TPs upgrade UDIs to CDIs

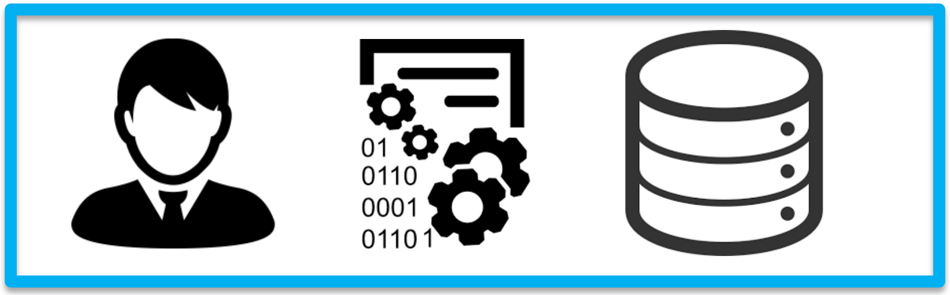
Enforcement Rules

E1	CDIs changed only by authorized TP
E2	TP run only by authorized users
E3	Users are authenticated
E4	Authorizations changed only by certifiers

CW: Certification/Enforcement Rules

- **C1**: When any IVP is run, it must ensure all CDIs are in valid state
- **C2**: A TP must transform a set of CDIs from a valid state to another valid state
- **E1**: System must maintain certified relations
  - TP/CDI mapping enforced
  - TP must not be used on CDIs it is not certified for
- **E2**: System must control users
  - user/TP/CDI mappings enforced
  - the model must account for the person performing the TP





IVP: Integrity Verification Procedures
TP: Transformation Procedures
CDI: Constrained Data Items
UDI: Unconstrained Data Items

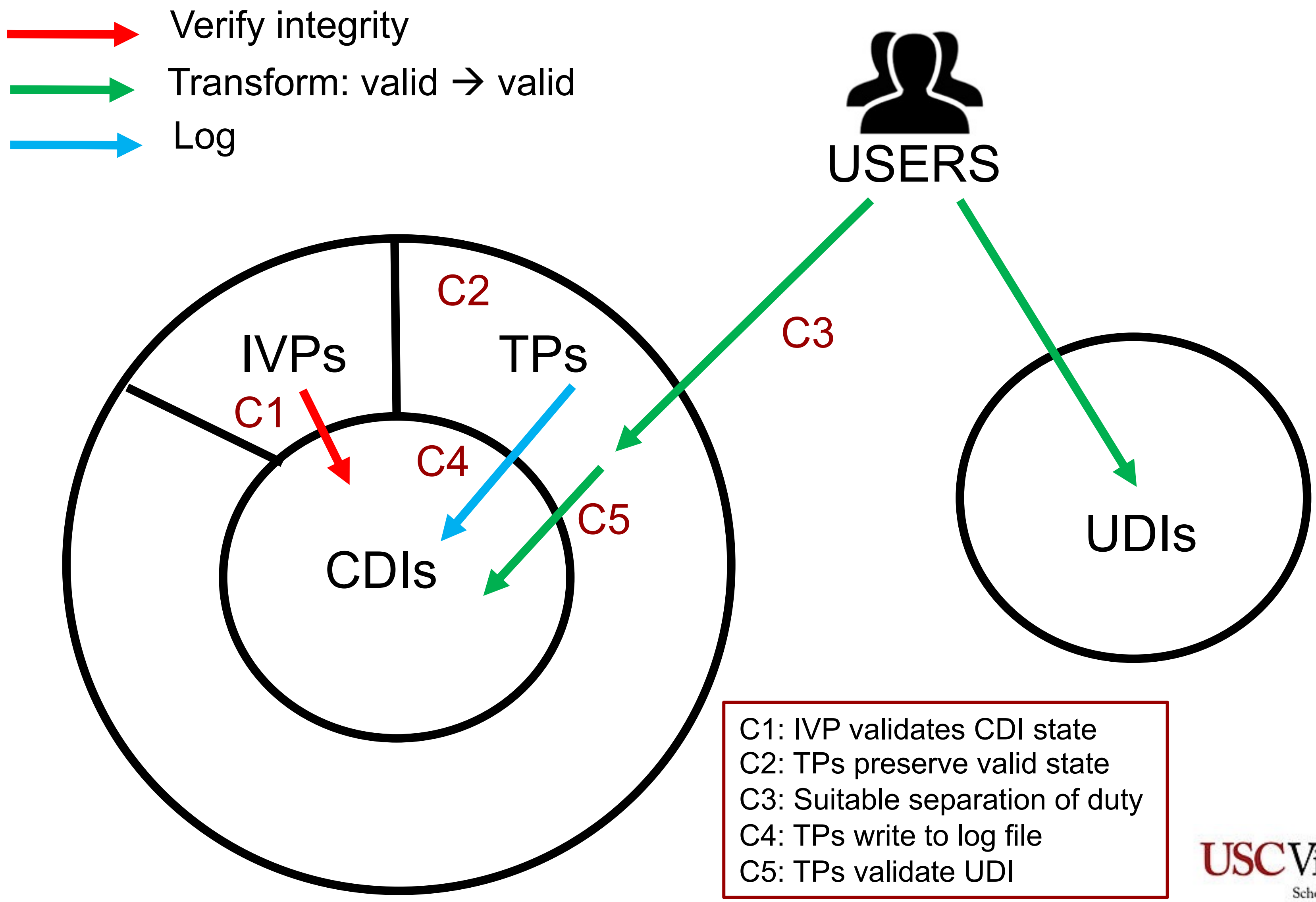
USC Viterbi School of Engineering

CW: Certification/Enforcement Rules

- **C3**: Relations between (user, TP, {CDI}) must support separation of duty
- **E3**: Users must be authenticated to execute TP
- **C4**: All TPs must log undo information to append-only CDI (to reconstruct an operation for review)
  - Log is CDI
- **C5**: A TP taking a UDI as input must either reject it or transform it to a CDI
  - information entering a system need not be trusted
- **E4**: Only certifier of a TP may change the list of entities associated with that TP
  - No certifier of a TP or CDI associated with that TP, may have execute permission on the TP/CDI
  - enforces the separation of duty

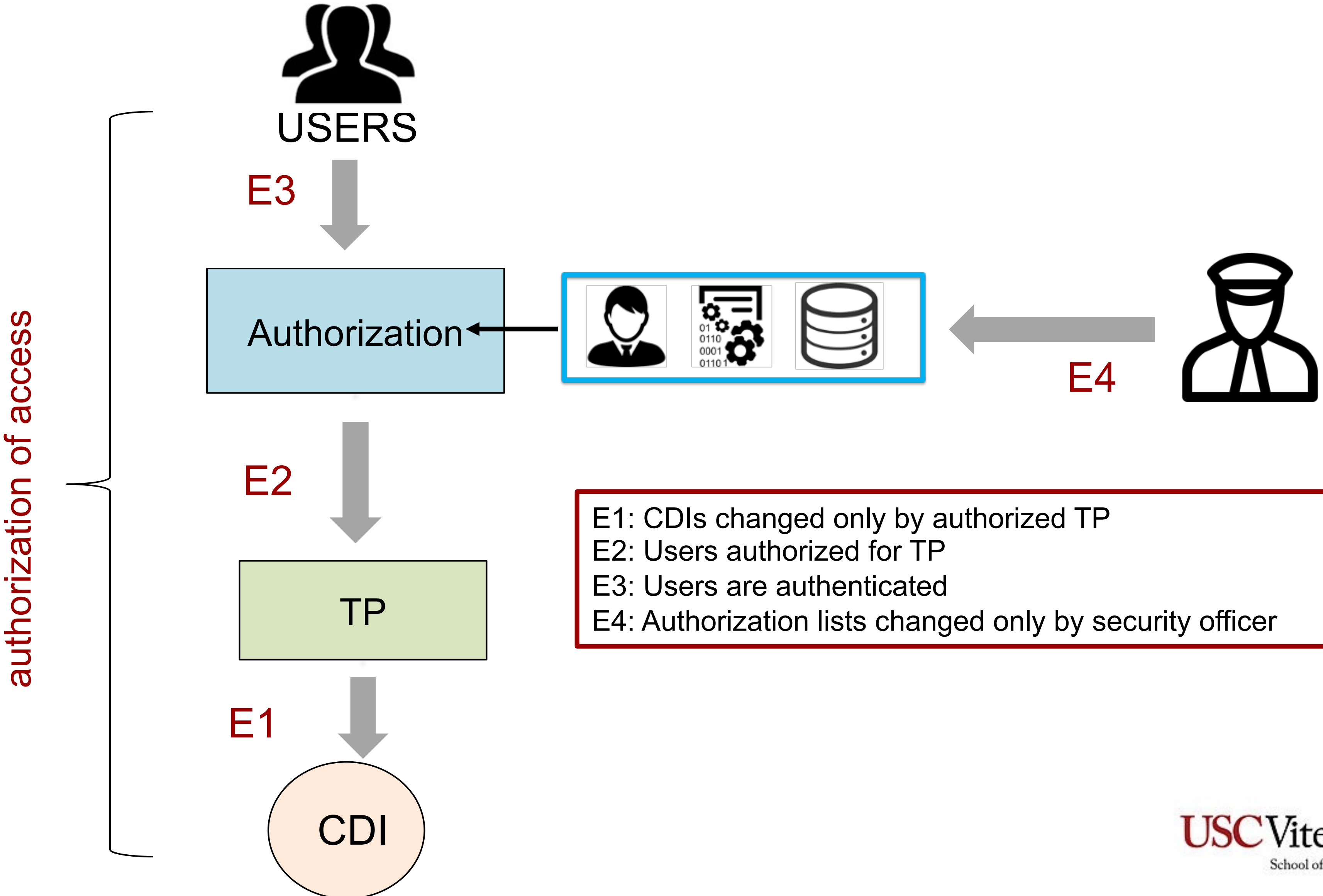
IVP: Integrity Verification Procedures
TP: Transformation Procedures
CDI: Constrained Data Items
UDI: Unconstrained Data Items

CW Elements and Certification Rules



C1: IVP validates CDI state
C2: TPs preserve valid state
C3: Suitable separation of duty
C4: TPs write to log file
C5: TPs validate UDI

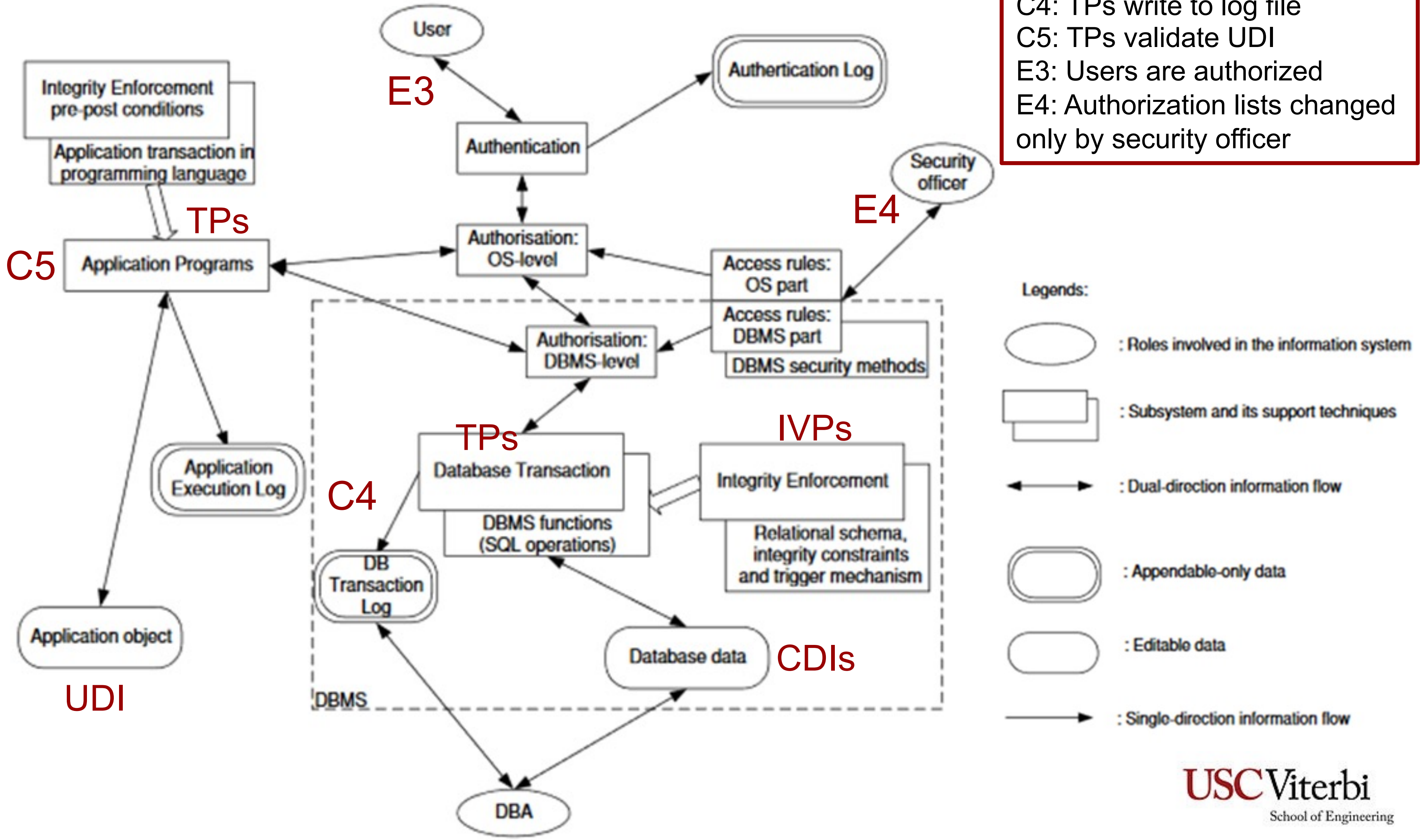
CW Elements and Enforcement Rules



E1: CDIs changed only by authorized TP
E2: Users authorized for TP
E3: Users are authenticated
E4: Authorization lists changed only by security officer

Example: Applying CW to a DBMS

C4: TPs write to log file
C5: TPs validate UDI
E3: Users are authorized
E4: Authorization lists changed only by security officer


User
E3
Authentication Log
Integrity Enforcement pre-post conditions
Application transaction in programming language
Authentication
Security officer
E4
TPs
C5
Application Programs
Authorisation: OS-level
Access rules: OS part
Access rules: DBMS part
Authorisation: DBMS-level
DBMS security methods
Legends:
: Roles involved in the information system
: Subsystem and its support techniques
TPs
IVPs
Application Execution Log
C4
Database Transaction
Integrity Enforcement
: Dual-direction information flow
DBMS functions (SQL operations)
Relational schema, integrity constraints and trigger mechanism
DB Transaction Log
: Appendable-only data
Application object
Database data
CDIs
: Editable data
UDI
DBMS
: Single-direction information flow
DBA

USC Viterbi School of Engineering

TP in practice: Transaction Concept

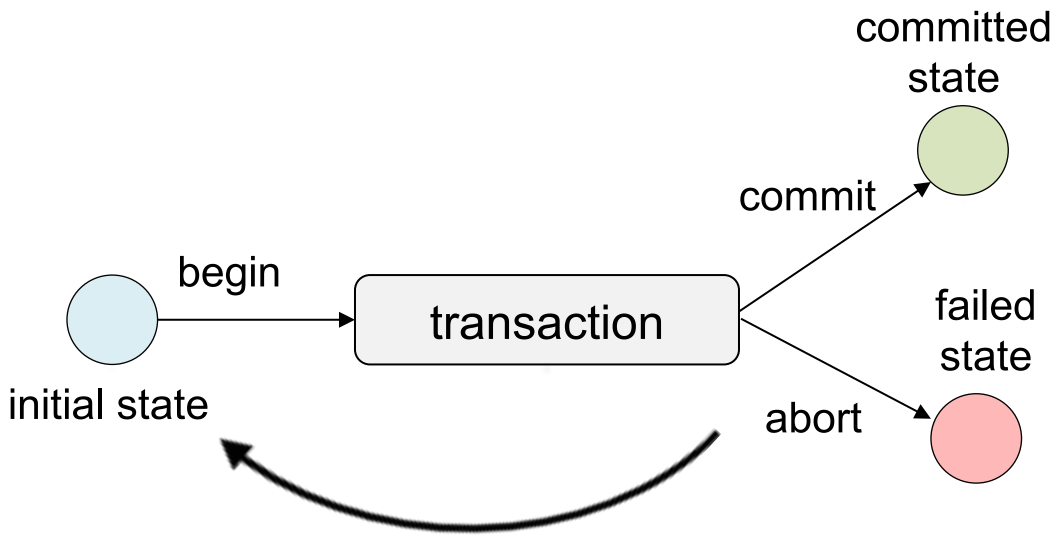
- A transaction is a set of atomic updates to the state of one or more objects
- Transaction can finish in one of two states:
  - **Committed**: If a transaction commits (succeeds) then the new state of the objects will be observed
    - i.e. all updates occur
  - **Rollback** (or **abort)**: If a transaction rolls back (fails) then the object will remain in its original state (as if no updates to any part of the state were made)
    - i.e. no updates occur

```
void threadTask(void* arg)
{
/* Do local computation */
/* checkpoints/saves state */

begin_transaction(val1,val2) {

/* Do some computation/updates */
val1 -= amount;
val2 += amount;
} // end_transaction

abort {
// restore/re-read val1, val2
// release locks
// restart
  }
}
```



ACID Properties

- Transactions help achieve the ACID properties:
  - Atomicity: update appears as indivisible (all or nothing), no partial updates are visible (at logical level)
    - At physical level, only single disk/flash write is atomic
  - Consistency: old state and new, updated state meet certain necessary invariants
    - E.g., no orphaned blocks, etc.
  - Isolation: other transactions do not see results of earlier transactions until they are committed
    - Serializability (transaction T1 appears to execute entirely before T2 or vice versa)
  - Durability: committed updates are persistent

Integrity Policy Models: Key Points

- Commercial world needs integrity
- Biba and Lipner models are based on multilevel integrity
  - Biba model
    - Dual of BLP (or BLP-upside-down)
      - Integrity levels distinct from security levels
      - Information flows differently
    - Can be combined with BLP
  - Lipner's lattice combines the two to meet commercial requirements
    - Pros & cons
  - Integrity problem is more than just restricting information flow
    - For example: concurrency control, integrity constraints, etc.
- Clark-Wilson model
  - Introduces new ideas
    - Enforces integrity by using well-formed transactions (through access triple), separation of user duties, and auditing