DSCI 519: Foundations and Policy for Information Security

Take-grant protection model, Biba Integrity Model

Tatyana Ryutov

Outline

- Midterm review
- General undecidability of security
  - Harrison-Ruzzo-Ullman result
  - Take-Grant Protection Model
- Biba Integrity Model

©Tatyana Ryutov

Reminder

- Midterm exam on October 5th
  - Exam time: 11-1pm
  - Time limit is enforced: 2hrs + 15minues
  - No lecture on this day
  - Closed notes/books
  - One handwritten cheat sheet (both sides)
  - Exam will be accessible via D2L in Quiz section
  - I'll be available to answer your exam-related questions via Piazza during the exam time

©Tatyana Ryutov

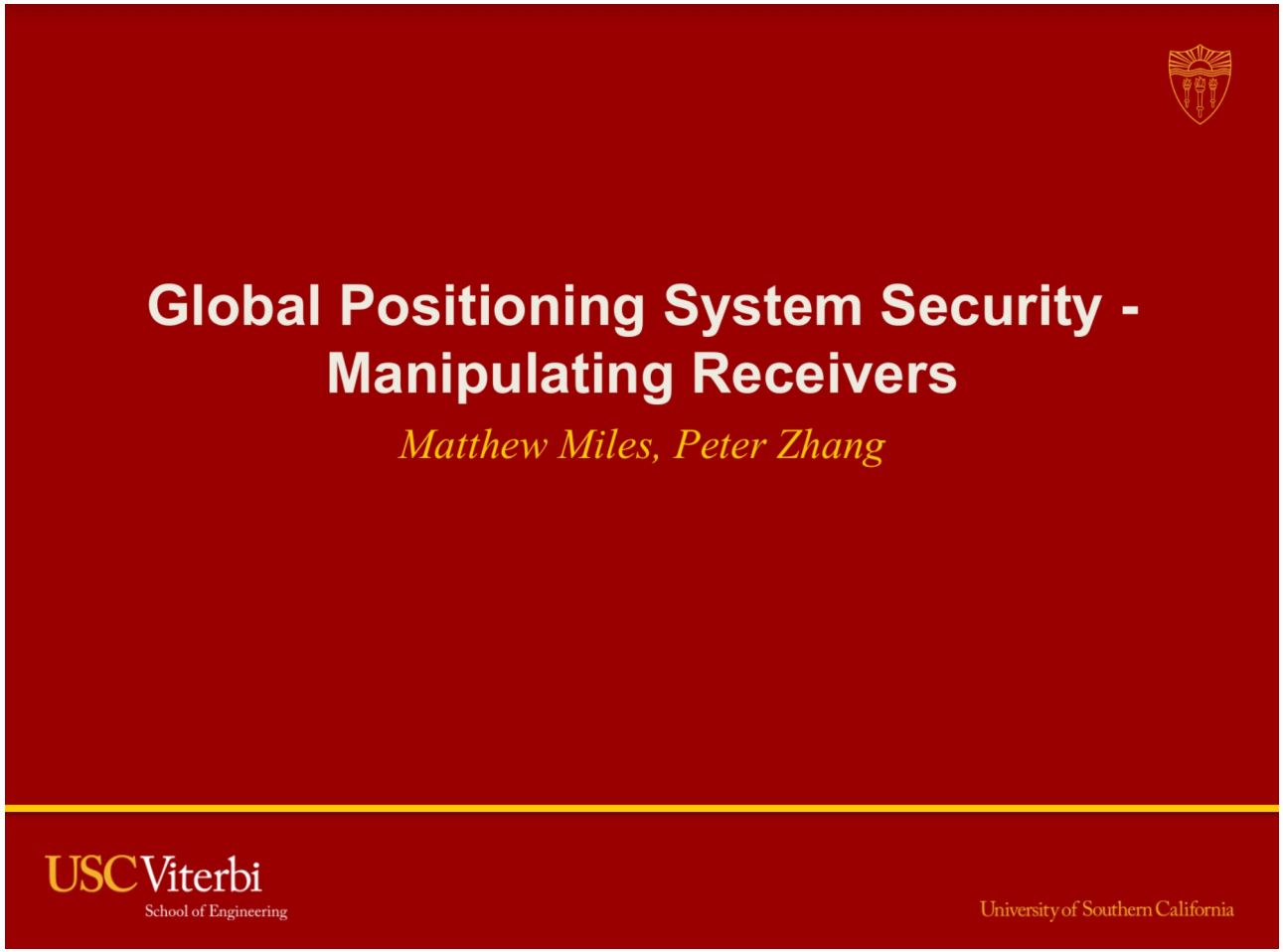
L6.Q6

- Complete the following sentences:

1. "I was surprised to learn … "
2. “My muddiest point is …”

©Tatyana Ryutov

Presentation 5



©Tatyana Ryutov

USC Viterbi
School of Engineering

Presentation 6

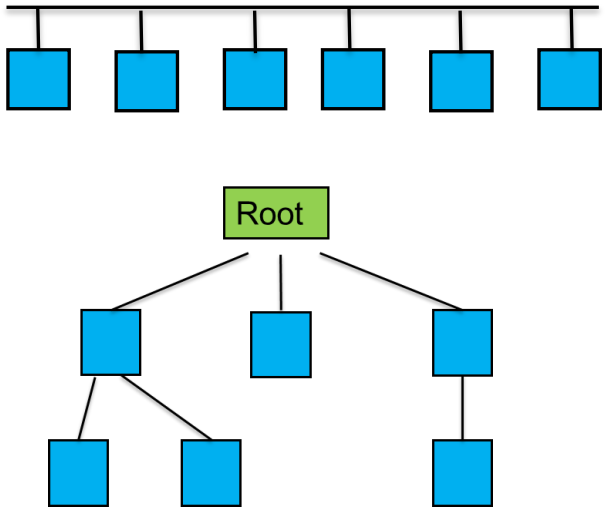
Hypervisor Security

Ashwini Prashanth Bhaskar
Arturo Paz

©Tatyana Ryutov

Your Questions

- What is the format of the test? Is it similar to the quizzes?
- Please go over why the flat structure isn't a good idea for implementing access class structure
- The difference between MAC and DAC
- What is MAC without MLS? is it just compartments?
- Can you review some answers to the homework problems briefly?



- MAC Policy
  - Subjects and Objects Labeled
- Access Matrix Policy
  - Processes with subject label
  - Can access object of object label
  - If operations in matrix cell allow
- Focus: Least Privilege
  - Just permissions necessary

SELinux Type Enforcement

	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

©Tatyana Ryutov

USC Viterbi School of Engineering

HW1 Keys: Problem1

- Threats
  - Insider obtaining confidential data
    - Malicious insider modifying data
    - Network-based theft of information
    - Customers obtaining elevated authority to modify payment information
    - Competitors stealing pricing data
    - DDoS
- Mitigation techniques
  - Red-team testing
  - Firewalls and Intrusion detection
  - Access control
  - FW and IDS
  - Non-disclosure agreements

Other meaningful solutions are acceptable

©Tatyana Ryutov

HW1 Keys: Problem 2

	Alice's files	Bob's files	Files that Alice shared with Bob	Files that Bob Shared with Alice	Files that Alice shared with Admin	Files that Bob shared with Admin	Admin's files
Alice	rwxo		rwxo	r	rwxo		
Bob		rwxo	r	rwxo		rwxo	
Admin					r	r	rwxo

Other meaningful solutions are acceptable

©Tatyana Ryutov

USC Viterbi School of Engineering

HW1 Keys: Problem 3

- Levels: board restricted > management restricted > insider restricted > corporate > confidential > public
  - "sensitive client data" should be a level and one of the high ones (e.g., "sensitive client data" > "management restricted") since required by law, therefore serious consequences are possible
- Categories: "HR data", "accounts receivable data"
  - HR data is a category: DAC (a number of people need access– whole HR department, everyone knows their own HR data, cannot be a level)
- Subjects:
  - Employees clearance "corporate"
  - People with NDAs, clearance "confidential"
  - Certain employees and managers, and designated outsiders who sign NDAs: clearance "insider restricted"
  - High-level managers clearance: "management restricted"
  - C-level managers and Board members: clearance "board restricted"
  - Certain employees and managers with a need-to-know: category "HR data" is added to their access level
  - Only certain employees and managers with a need-to-know: category "accounts receivable data" is added to their access level
  - Only certain employees and managers: clearance "sensitive client data"

Other meaningful solutions are acceptable

©Tatyana Ryutov

Midterm Topics

- Topics that will be tested on the midterm:
  - Challenge of security policy breaches
  - Characteristics of policy
  - Reference monitor and security policy models
  - U.S. classified information policy
  - Bell-LaPadula model and Multics interpretation
  - Theoretical limits on system security
    - NO proofs

©Tatyana Ryutov

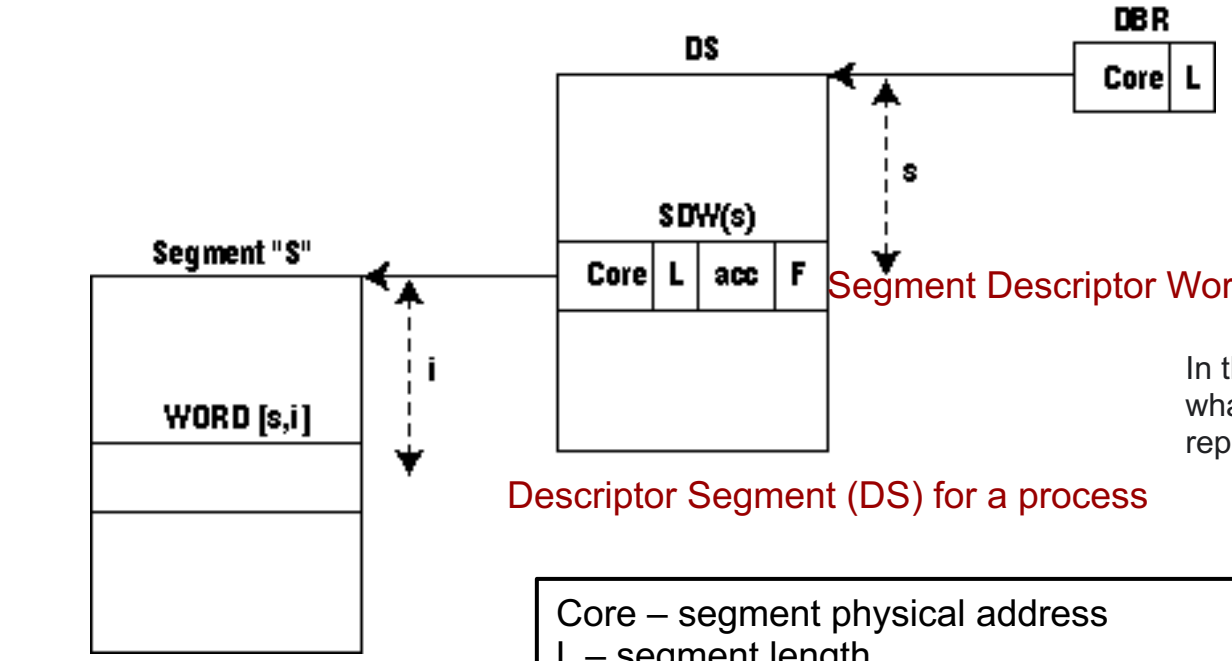
Your Questions

- HW Memory referencing is difficult. Also BLP interpretation is a little confusing
- Basic security theorem is still muddy for me

©Tatyana Ryutov

Multics Hardware Memory Referencing

- All code, data, I/O devices, etc. that may be accessed by a process are stored as segments
- A segment virtual address consists of a pair of integers [s, i]
  - "s" is called the segment number (descriptor)
  - "i" the index within the segment (offset)
- Descriptor Segment (DS) stores Segment Descriptor Words (SDW) that reference each of the process's active segments

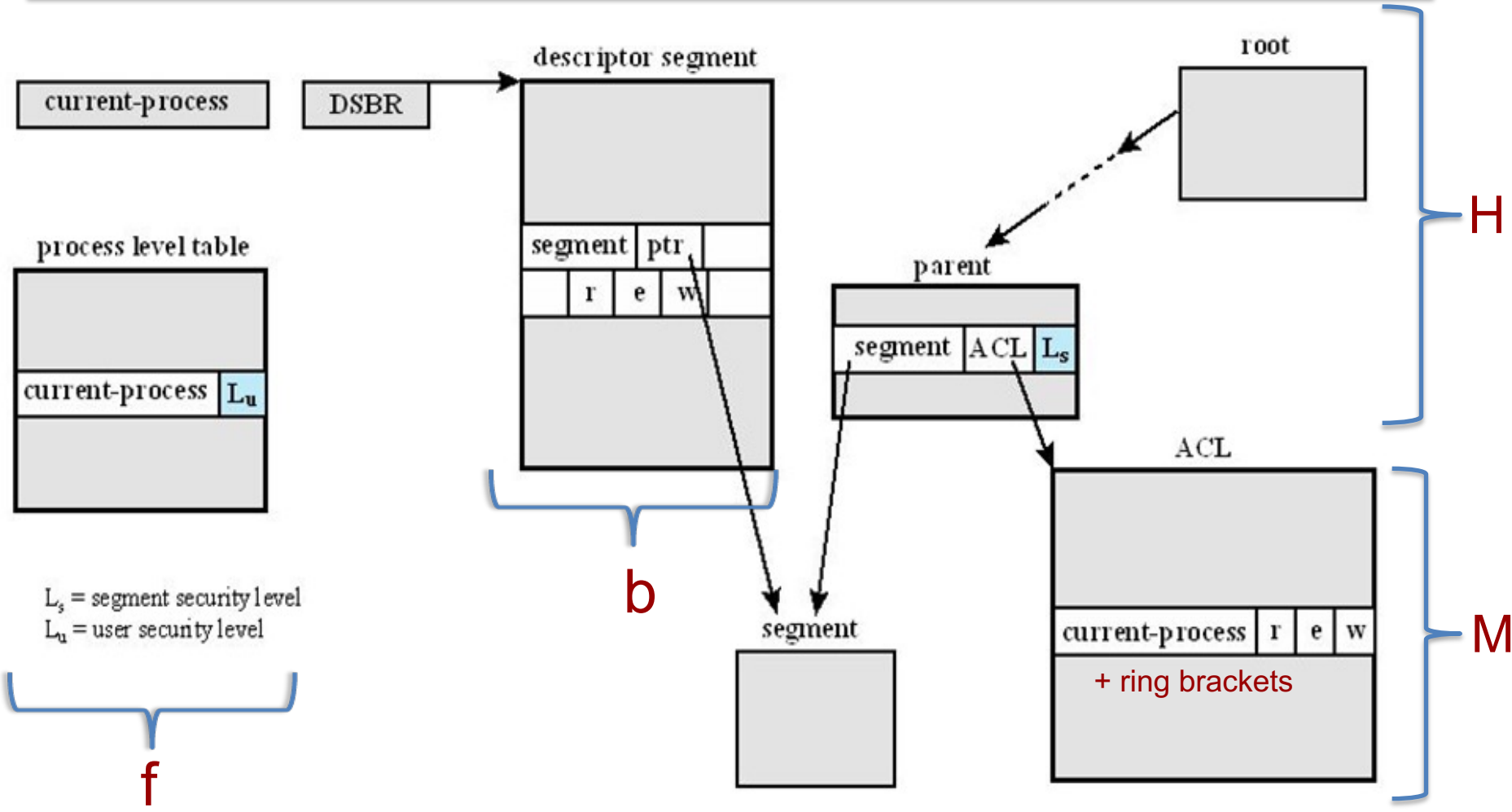


In the Multics system, I'm a bit confused about what 'i' is while defining the SDW. Does 'i' represent the object?

Core – segment physical address
L – segment length
F – flags
acc – segment access permissions (read, write, execute)

©Tatyana Ryutov

Multics Data Structures



©Tatyana Ryutov

USC Viterbi
School of Engineering

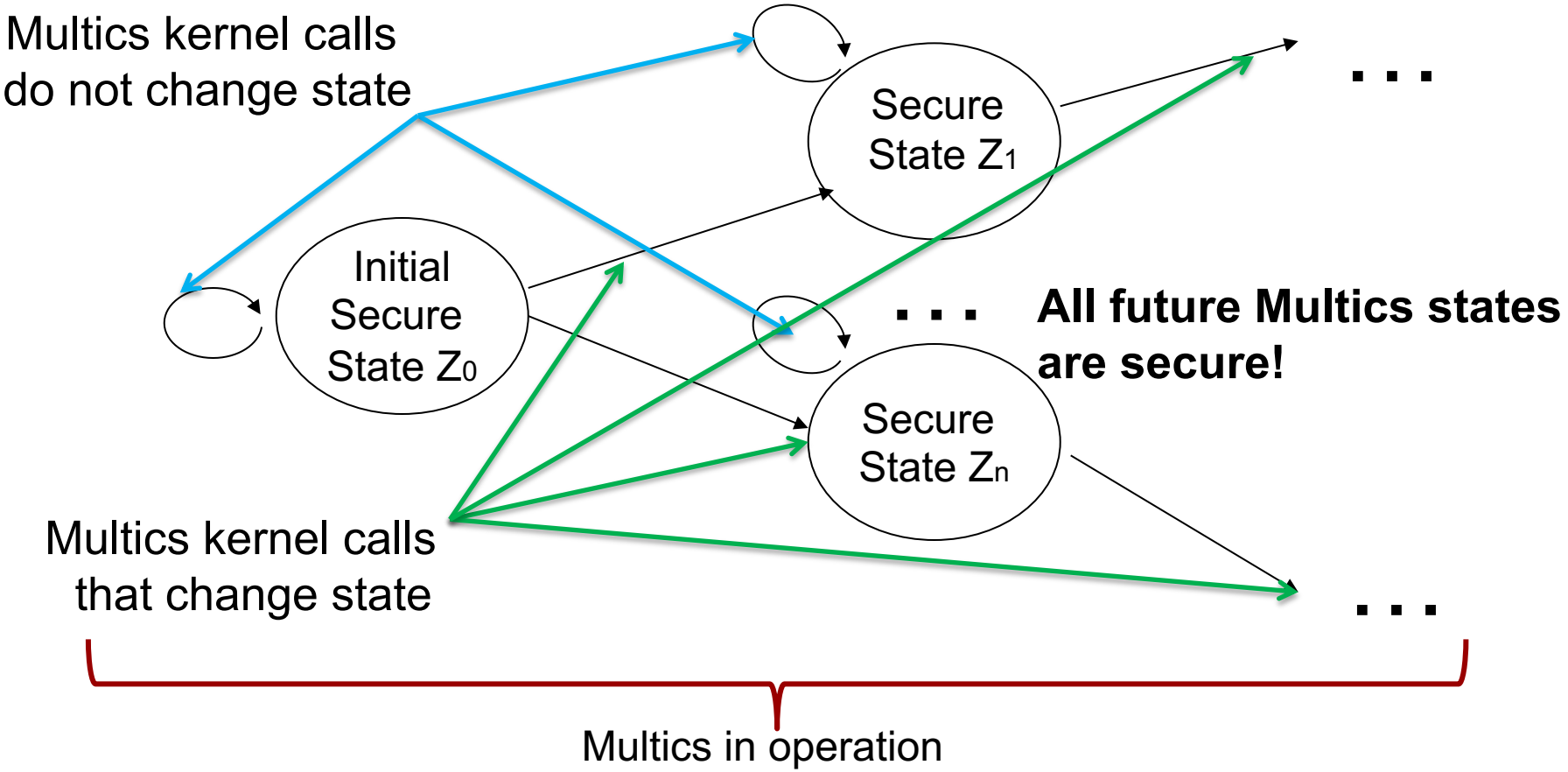
Multics Interpretation of BLP System Σ

- Interpret BLP for Multics *authorization* functions
  - Subjects *change* the state in authorization database
- In formal terms, system $\Sigma(R, D, W, z_0)$ where:
  - R denotes the set of requests for access
  - D denotes the set of outcomes
  - W is the set of actions of the system
  - z_0 is the initial state of the system
- Rules for transition from one state to another
  - Functions to change each element of state (b, M, f, H)
- Basic Security Theorem – Multics (Σ) is secure
  - If initial state secure, all states from rules are secure

©Tatyana Ryutov

USC Viterbi School of Engineering

Multics as a State Machine



- Defined **all possible** transitions (Multics kernel calls), each transition satisfies the 3 properties

©Tatyana Ryutov

USC Viterbi
School of Engineering

Outline

- Midterm review
- General undecidability of security
  - Harrison-Ruzzo-Ullman result
  - Take-Grant Protection Model
- Biba Integrity Model

©Tatyana Ryutov

What are we trying to do?

- Can we determine if a computer system is “secure” (i.e., implements our policy with high assurance)?
- Our goal is to answer the key question: is this protection problem decidable?

©Tatyana Ryutov

Objectives of the HRU Work

- Provide a model that is sufficiently powerful to encode several access control approaches, and precise enough so that security properties can be analyzed
- Introduce the “safety problem”
  - Accurately and concisely expresses the essence of the protection problem
- Show that the safety problem
  - is undecidable in general
  - is undecidable in monotonic case
    - A monotonic system is one in which rights cannot be deleted, and subjects and objects cannot be destroyed
  - is decidable in certain cases

©Tatyana Ryutov

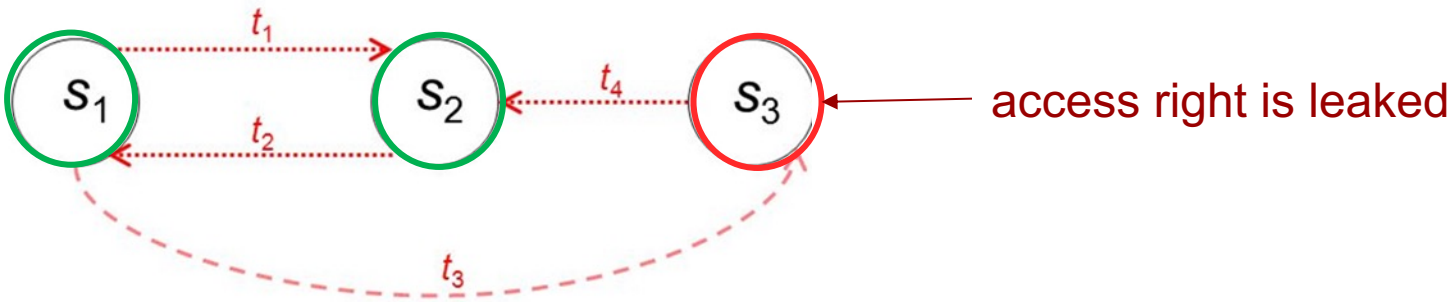
Overview of the HRU Model

- What is “Secure”?
  - Adding a generic right r where there was not one is “leaking”
  - If a system S, beginning in initial state s_0, cannot leak right r, it is **safe** with respect to the right r
- Safety Question
  - Does there exist an algorithm for determining whether a protection system S with initial state s_0 is safe with respect to a generic right r?
  - Here, “safe” = “secure” for an abstract model
- Answer for a general case is “no”
  - Reduce halting problem to safety problem
  - Show that a Turing machine can be “modelled” by a protection system with the “states" of the machine mapped to the “rights" of the protection system

©Tatyana Ryutov

Notion of a "Leaked" Right

- Rights are the entries in access control matrix
  - Each subject has (or does not have) rights to an object
- Define **leaked**
  - Generic right added to element of access matrix when elements do not already contain the right
- Policy defines the authorized set of states
  - No command can leak a right *r*
- Define **safe** state with respect to a right *r*
  - System can never leak the right *r*
  - System is *unsafe* if it can enter unauthorized state



S_1 and S_2 - safe states, S_3 – unsafe state

©Tatyana Ryutov

USC Viterbi School of Engineering

Recall: L5.Q5

- Can we answer the safety question in this example?

- Adding a generic right r where there was not one is “leaking”
- Leaked = granted
- Scenario:
  - Bob works on Project 1, needs to read and write Object1
  - Alice works on Project 2, creates an Object 2 (has full privileges)
  - The current protection state of the system:

	Object1	Object2
Bob	r w	r
Alice		r w own

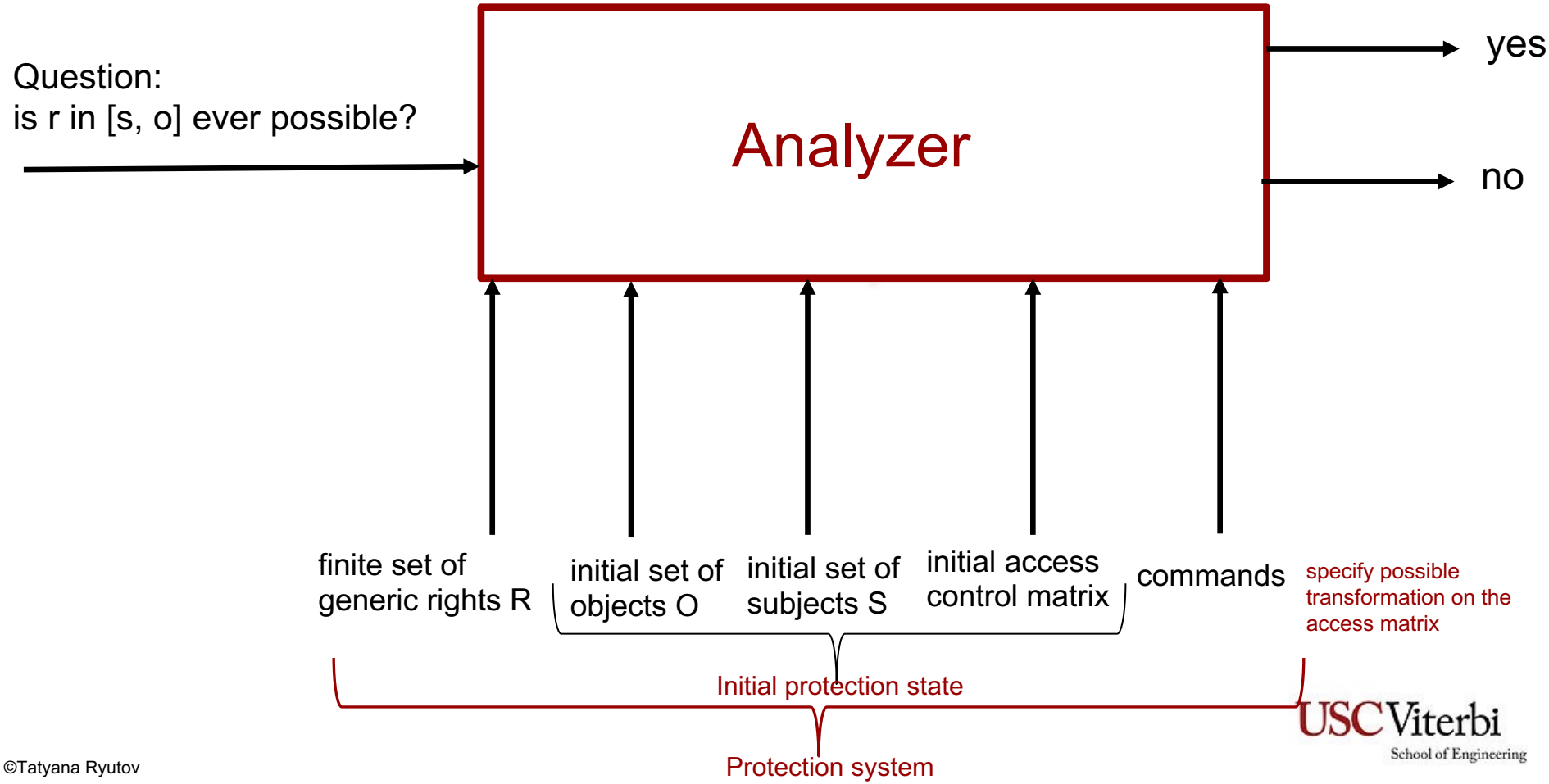
  - Alice wants Bob to help her with Project2 and grants him r access to Object 2
    - Access right is “leaked”

©Tatyana Ryutov

USC Viterbi School of Engineering

HRU: General Idea

- Can we build an analyzer to answer the safety question?
  - Assume that the sole purpose of a system is to change access privileges to objects, ignoring other computations that might be occurring



©Tatyana Ryutov

USC Viterbi School of Engineering

HRU Protection System

- A protection system is a state-transition system
- A model for **protection** of computer system consists of:
  1. A finite set of generic access rights R
  2. Initial protection state (initial set of objects O, initial set of subjects S, and initial access matrix P)
  3. A finite set of **commands** C of the form:

parameters

command $\alpha(X_1, X_2, \ldots, X_k)$

  if r_1 in (Xs_1, Xo_1) **and** r_2 in (Xs_2, Xo_2) **and** r_i in (Xs_i, Xo_i)

…

then

  $op_1; op_2; \ldots op_n$

end

conditions (test presence of certain rights in certain positions in access matrix)

command body is straight line code, no conditionals, no function invocation

- These commands are interpreted as a sequence of primitive **operations**

USC Viterbi School of Engineering

©Tatyana Ryutov

HRU Primitive Operations

- Primitive operations affect the state of access matrix:
  1. enter r into (Xs, Xo)
     - Condition: Xs ∈ S and Xo ∈ O
     - r may already exist in (Xs, Xo)
  2. delete r from (Xs, Xo)
     - Condition: Xs ∈ S and Xo ∈ O
     - r does not need to exist in (Xs, Xo)
  3. create subject Xs
     - Condition: Xs ∉ O
  4. create object Xo
     - Condition: Xo ∉ O
  5. destroy subject Xs
     - Condition: Xs ∈ S
  6. destroy object Xo
     - Condition: Xo ∈ O and Xo ∉ S

©Tatyana Ryutov

The State of A Protection System

- HRU define the “configuration” of a system
- Instantaneous description of protection system is a triple (S, O, P):
  - S is the set of current subjects,
  - O is the set of current objects, $S \subseteq O$ (subjects can be objects)
  - P is an access control matrix
    - one row for each subject
    - one column for each object
    - each cell contains a set of rights from R

©Tatyana Ryutov

How does state transition work?

- Given a protection system (R, C), state q_1 can reach state q_2 IFF there is an instance of a command in C so that all conditions are true at state q_1 and executing the primitive operations one by one results in state q_2
  - R is a finite set of generic rights
  - C is a finite set of commands

  } these sets do not change
- A command is executed as a whole (similar to a transaction), if one step fails, then nothing changes

©Tatyana Ryutov

Example 1

[Unix] process p creates file f with owner *read* and *write* (r, w) will be represented by the following:

Command $create\_file(p, f)$
  Create object f ← create a column in the access matrix
  Enter own into $a[p,f]$
  Enter r into $a[p,f]$
  Enter w into $a[p,f]$
End

©Tatyana Ryutov

Example 2

- Process p creates a new process q

  Command *spawn_process*(*p*, *q*)

  Create subject *q;* ← create a row in the access matrix

  Enter *own* into *a*[*p*,*q*]

  Enter *r* into *a*[*p*,*q*]

  Enter *w* into *a*[*p*,*q*]

  Enter *r* into *a*[*q*,*p*] ← parent and child can signal each other

  Enter *w* into *a*[*q*,*p*]

  End

©Tatyana Ryutov

HRU: The Safety Problem

- Given a protection system and generic right r, we say that the initial configuration Q_0 is unsafe for r (or leaks r) if there is a configuration Q and a command α such that:
  - Q is reachable from Q_0
  - α leaks r from Q
- We say that a command $\alpha(x_1,\ldots,x_k)$ leaks generic right r from Q if α, when run on Q, can execute a primitive operation which enters r into a cell of the access matrix which did not previously contain r

©Tatyana Ryutov

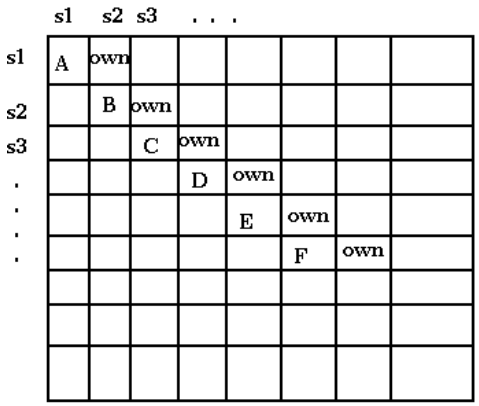
Relationship between TM and HRU

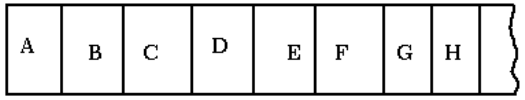
- It is undecidable (no generic algorithm) to determine whether an arbitrary TM halts or not
  - Or enters any arbitrary state q_f
- Idea: reduce protection problem (HRU) to TM
  - If TM enters state q_f, then the protection system can leak generic right r, otherwise, it is safe for r
  - Generic right *r* is arbitrary and hence yielding state q_f is also arbitrary
  - Since it is undecidable whether the TM enters arbitrary state q_f, it must be undecidable whether the protection system is safe for r
- Next question: how to map HRU to TM?

©Tatyana Ryutov

Mapping a Tape to an Access Matrix

- Create the protection matrix from the TM's tape
  - Encode the contents of the TM's tape on the diagonal of the protection matrix: element [s, s] will contain the s^{th} tape square
  - How can we represent **sequential** tape?
    - Subject s_i represents cell i^{th} cell on the tape
    - Each subject s_i "owns" subject s_{i+1}
    - Sequential ownership relation represents sequential tape
  - Any cell (in TM) holding a symbol indicates subject s_i gave that right to itself
  - Last subject s_k has right *end*, indicating that subject s_{k+1} (which s_k owns), has not yet been created
  - The head is at the i^{th} cell and the current state is q ⇒ q ∈(s_i , s_i)





©Tatyana Ryutov

USC Viterbi School of Engineering

Mapping a Tape to an Access Matrix

- We also need to encode the state of the TM
  - The set of generic rights represent states and tape symbols
  - Two special rights: *own* and *end*
    - *end* is the last cell before blanks
  - For example, rights g and c are elements of $[s_3, s_3]$ when the TM is in state q, the read/write head is on square 3, and symbol c is in $[s_3, s_3]$
- Turing Machine instructions are mapped to commands of the HRU protection system

©Tatyana Ryutov

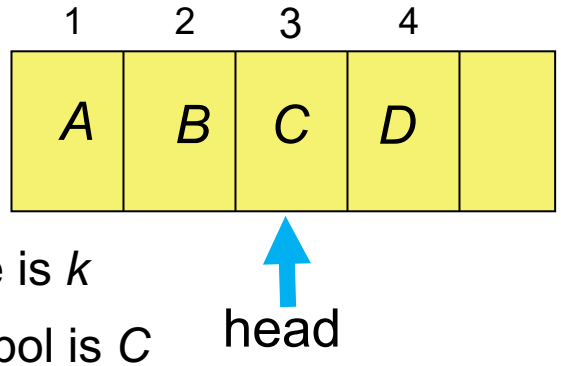
Representing Head Moves

- The moves of TM are represented as HRU commands
  - Changing the state of the TM is equivalent to commands that delete and add rights, objects, and subjects
- Choose HRU commands to represent TM moves
  - Right: $\delta(k, C) = (k_1, X, R)$
    - E.g., if cell k is the current position, command (k_1,C/X,R) substitutes access right C for access right X in the cell, and k_1 is the cell to the immediate right of k
    - When move right till end (blanks), need to create a new subject
  - Left: $\delta(k, C) = (k_1, X, L)$
    - E.g., if cell k is the current position, command (k_1,C/X,L) substitutes access right C for access right X in the cell, and k_1 is the cell to the immediate left of k
- For any possible TM transition, can have corresponding HRU command

©Tatyana Ryutov

Mapping Depiction

TM



Current state is *k*

Current symbol is *C*

HRU Matrix

	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			C *k*	*own*	
s_4				D *end*	

Symbols, States ⇒ rights

Tape cell ⇒ subject

Cell s_i has *A* ⇒ s_i has *A* right on itself

Cell s_k ⇒ s_k has *end* right on itself

State *p*, head at s_i ⇒ s_i has *p* right on itself

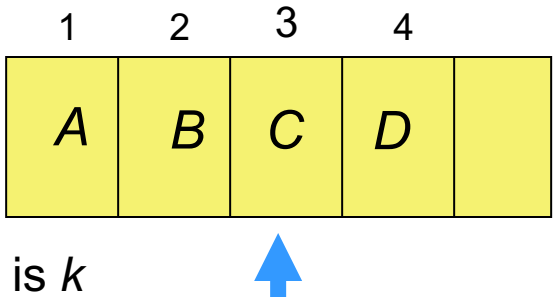
Distinguished right *own*: s_i *owns* s_i*+1* for $1 \le i < k$

©Tatyana Ryutov

USC Viterbi School of Engineering

Right Move (Left is Symmetrical)

```
command c_{k,C}(s_3, s_4)
if own in A[s_3, s_4] and k in A[s_3, s_3]
      and C in A[s_3, s_3]
then
   delete k from A[s_3, s_3];
   delete C from A[s_3, s_3];
   enter X into A[s_3, s_3];
   enter k_1 into A[s_4, s_4];
end
```



Current state is *k*

Current symbol is *C*

head

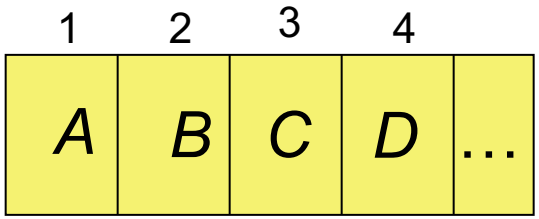
	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			C *k*	*own*	
s_4				D *end*	

$\delta(k, C) = (k_1, X, R)$

©Tatyana Ryutov

USC Viterbi School of Engineering

After One Right Move



Current state is k_1

Current symbol is C

head

$$\delta(k, \mathrm{C}) = (k_1, \mathrm{X}, \mathrm{R})$$

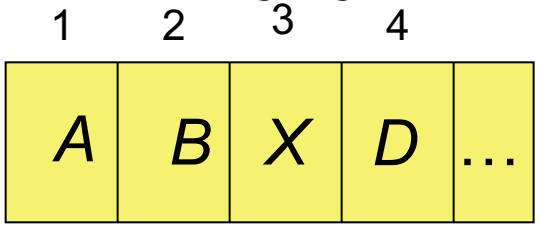
	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				D k_1 end	

©Tatyana Ryutov

USC Viterbi School of Engineering

Right Move at End

```
command crightmost_{k,c}(s4, s5)
if end in A[s4, s4] and k1 in A[s4, s4]
       and D in A[s4, s4]
then
   delete end from A[s4, s4];
   create subject s5;
   enter own into A[s4, s5];
   enter end into A[s5, s5];
   delete k1 from A[s4, s4];
   delete D from A[s4, s4];
   enter Y into A[s4, s4];
   enter k2 into A[s5, s5];
end
```



Current state is k_1

Current symbol is D

head

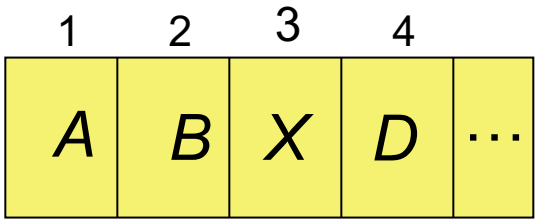
$\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$

	s_1	s_2	s_3	s_4	
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				D k_1 end	

©Tatyana Ryutov

USC Viterbi School of Engineering

After Right Move at End



Current state is k_1

Current symbol is D

head

	s_1	s_2	s_3	s_4	s_5
s_1	A	*own*			
s_2		B	*own*		
s_3			X	*own*	
s_4				Y	*own*
s_5					k_2 end

$$\delta(k_1, \mathrm{D}) = (k_2, \mathrm{Y}, \mathrm{R})$$

©Tatyana Ryutov

USC Viterbi School of Engineering

Rest of Proof

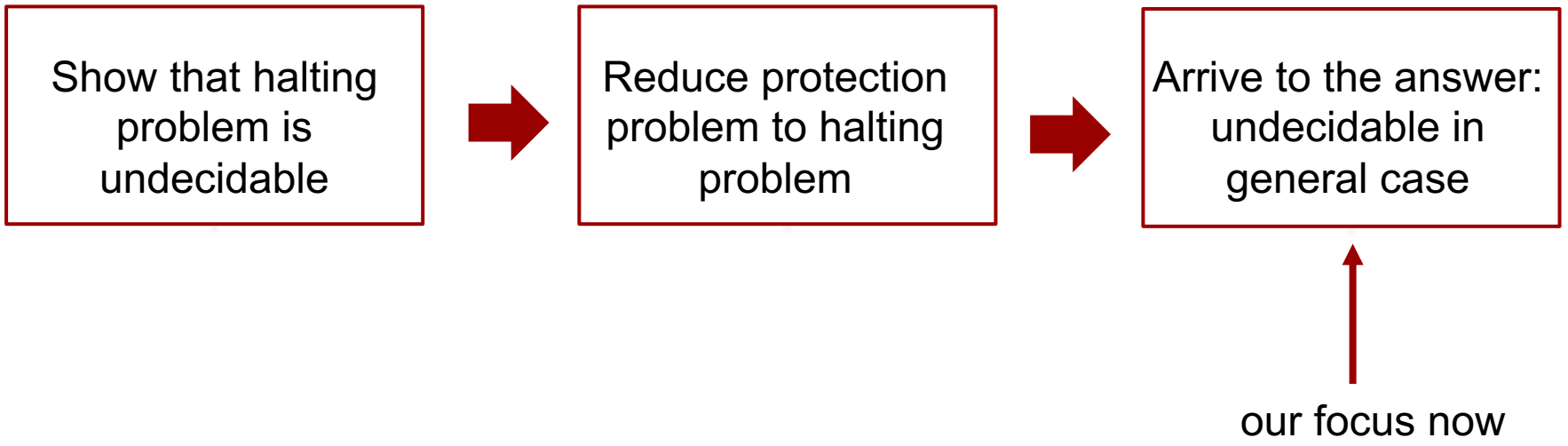
- Protection system exactly simulates actions of TM
  - Exactly one *end* right – blanks after that
  - Exactly one access right corresponds to a TM state
    - Thus, at most one applicable command
- If TM enters state q_f, then right is leaked
- Generic right *r* is **arbitrary** and hence yielding state q_f is also arbitrary
- If safety question is decidable, then we can represent TM as discussed and determine if q_f leaks
  - Implies halting problem is decidable. A contradiction!
- Conclusion: safety problem undecidable

©Tatyana Ryutov

USC Viterbi School of Engineering

What are we trying to do?

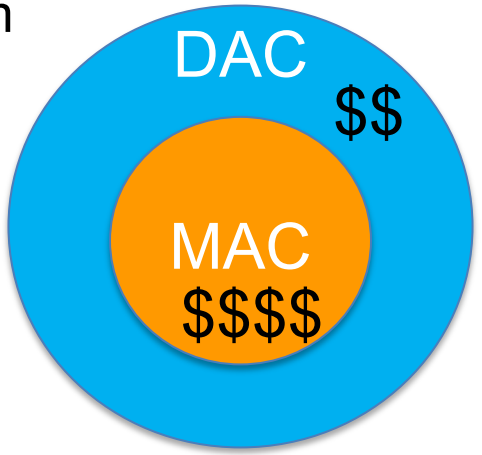
- Can we determine if a computer system is “secure” (i.e., implements our policy with high assurance)?
- Our goal is to answer the key question: is this protection problem decidable?



©Tatyana Ryutov

Theoretical Limits on System Security Summary

- 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



©Tatyana Ryutov

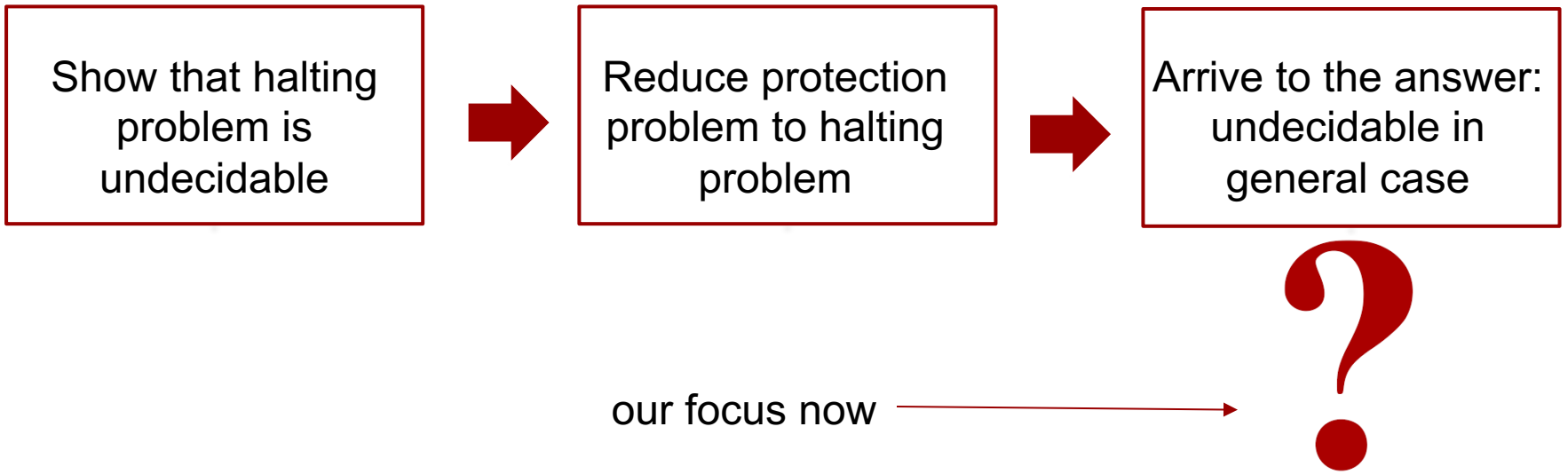
Outline

- Midterm review
- General undecidability of security
  - Harrison-Ruzzo-Ullman result
  - Take-Grant Protection Model
- Biba Integrity Model

©Tatyana Ryutov

What are we trying to do?

- Can we determine if a computer system is “secure” (i.e., implements our policy with high assurance)?
- Our goal is to answer the key question: is this protection problem decidable?



©Tatyana Ryutov

Recall: The "Safety Question"

- Does there exist an algorithm for determining whether a given protection system with initial state s_0 is safe with respect to a generic right r?
- Foundational result: it is undecidable
- Proof leverages TM halting problem
  - Undecidable whether it enters a particular state
- Must show that HRU protection system can simulate behavior of arbitrary TM
- Must map problem (access matrix) to cells/states in TM
- Map a final (halted) state to the open question (safety)
- Since we know that whether TM halts is undecidable
  - So whether right r is leaked is therefore undecidable

©Tatyana Ryutov

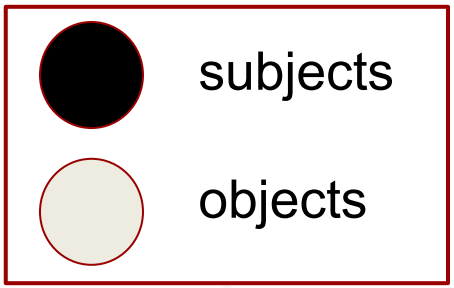
Take-Grant

- A **specific** (not generic) system
- Set of rules for state transitions
- Take-grant model is less expressive than HRU (special case of HRU)
- Safety decidable in linear time with respect to graph size
- Goal: find conditions under which rights can be transferred from one entity to another in the system

©Tatyana Ryutov

Take-Grant Model

- Uses directed graphs to model access control
  - More efficient than (sparsely populated) access matrix
  - Represent the same information found in an access matrix
- Graph nodes: subjects and objects
  - An edge from node **x** to node **y** indicates that subject **x** has an access right to object **y**: the edge is tagged with the corresponding access rights
- Possible access rights:
  - **read**, **write** with obvious meaning
  - **take** and **grant** - special access rights for propagating access rights to other nodes



USC Viterbi School of Engineering

©Tatyana Ryutov

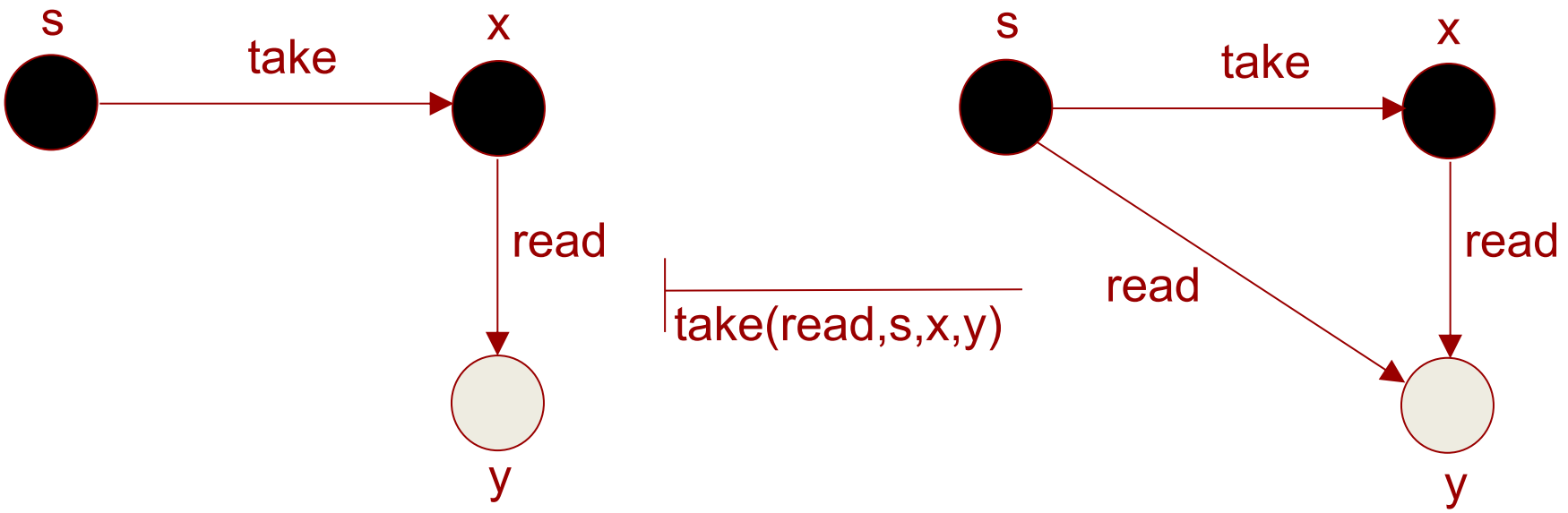
Take-Grant Model Contd.

- State and state transitions:
  - Protection state of the system is represented by a directed graph
  - System changes state when the directed graph changes
- Operations that change the directed graph:
  - **Create**: a new node is added to the graph
    - add a vertex and an edge from the subject to the new vertex
  - **Remove**: a node deletes some of its access rights to another node
    - remove an edge originating at the subject
  - **Take**: implies that a node can *take* another node's access rights and pass them to any other node
    - add an edge originating at the subject
  - **Grant**: if node **x** has access right *grant* to node **y**, then node **y** can be granted any of the access rights that **x** has
    - add an edge terminating at the subject

©Tatyana Ryutov

Take-Grant Model Example1

- Operations and transfer of privileges:
  - **take(read, s, x, y)**: subject **s** takes right **read** on object **y** from subject **x**
  - this requires that **s** holds the **take** privilege upon **x** and **x** must hold **read** privilege on **y**

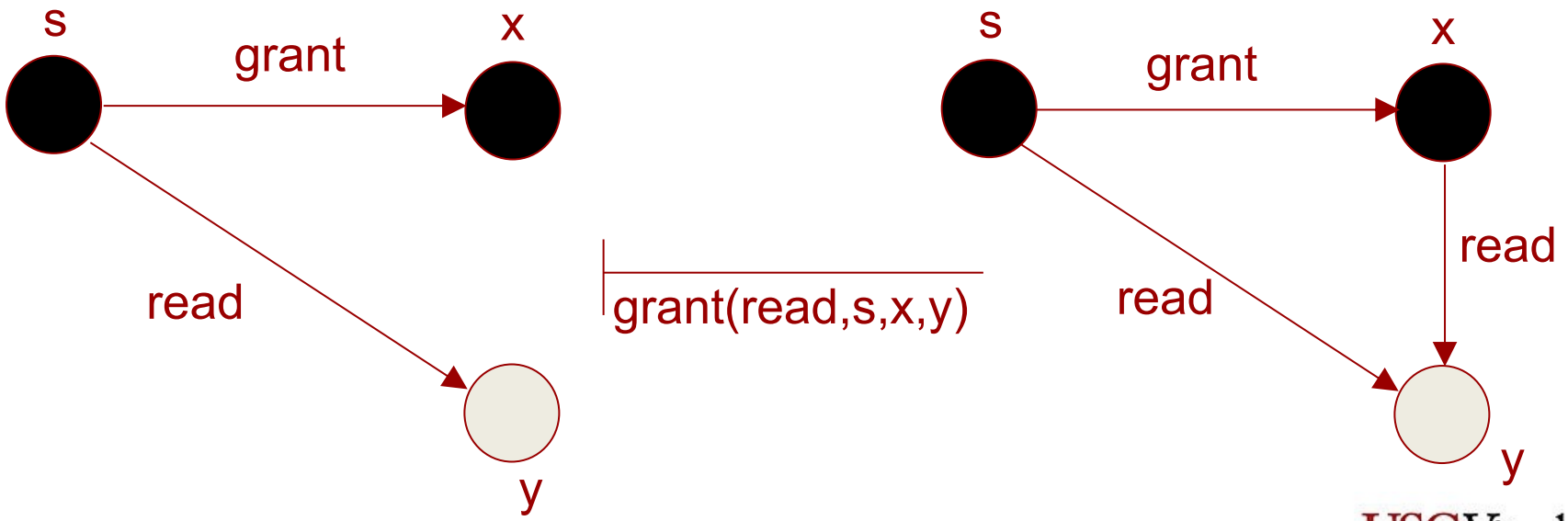


©Tatyana Ryutov

USC Viterbi
School of Engineering

Take-Grant Model Example2

- Operations and transfer of privileges:
  - **grant(read, s, x, y)**: subject **s** grants right **read** on object **y** to subject **x**
    - this requires that **s** holds the **grant** privilege on **x** and **x** must hold **read** privilege on **y**



©Tatyana Ryutov

USC Viterbi
School of Engineering

Take-Grant Model: Safety Question

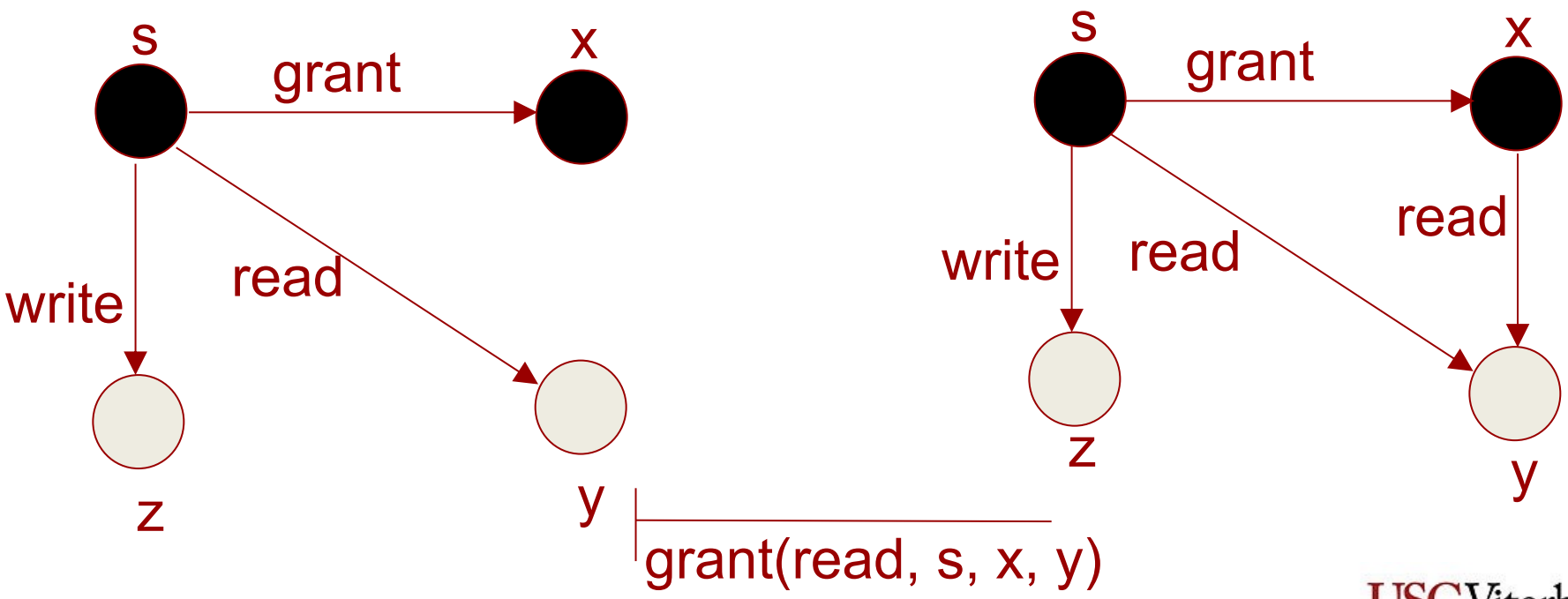
- **Leaking**: adding a generic right **r** where none existed
- **Safe**: if a system cannot leak right **r**, it is safe with respect to right **r**
  - In other words: given a policy, can we answer the question "will this specific subject X gain a specific type of access r to object Y?"
- **Safety Question**: 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)

©Tatyana Ryutov

USC Viterbi School of Engineering

Take-Grant Safety Question Example

1. Is this system safe with respect to **read**, object **z,** and subject **x**?
2. Is this system safe with respect to **read**, object **y,** and subject **x**?



©Tatyana Ryutov

USC Viterbi
School of Engineering

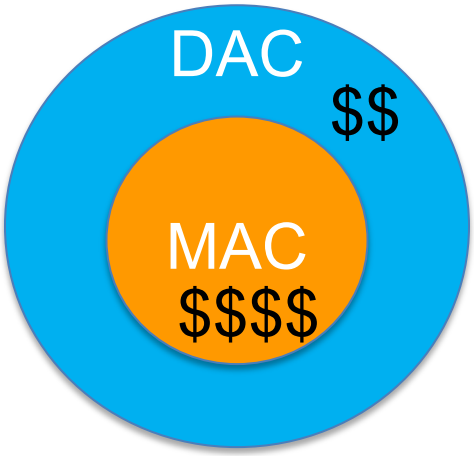
Key Question

- Characterize class of models for which safety is decidable
  - Existence: Take-Grant protection model is a member of such a class
  - Universality: In general, question undecidable, so for some models it is not decidable

©Tatyana Ryutov

Recall: Theoretical Limits on System Security Summary

- 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
  - Consistent with analysis of Trojan horse threat
- MAC can be safe
  - Imposes sufficient restrictions on commands
- How can we use this in practice?
  - Balanced assurance



©Tatyana Ryutov

Outline

- Midterm review
- General undecidability of security
  - Harrison-Ruzzo-Ullman result
  - Take-Grant Protection Model
- Biba Integrity Model

©Tatyana Ryutov

Integrity Policies

- BLP focuses on confidentiality
  - In most systems, integrity is equally, if not more, important
- Integrity refers to the trustworthiness of data or resources
  - Usually defined in terms of preventing improper or authorized change to data
- Data integrity vs. system integrity
- What is integrity in systems?
  - Critical data do not change
  - Critical data changed only as intended by authorized users
  - Critical data changed only in “correct ways”
  - Critical data changed only through certain “trusted programs”
- Integrity policy models
  - MLS: Biba and Lipner models
  - Clark-Wilson model

©Tatyana Ryutov

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

USC Viterbi School of Engineering

©Tatyana Ryutov

Integrity Threats (Biba)

- Threat sources
  - from **external** subsystems
    - E.g., one subsystem provides false data to another
  - from **internal** subsystems
    - Subsystem is itself malicious
- Threat Types
  - **direct** (overt)
    - Write into protected object
  - **indirect** (covert)
    - Results from use of malicious data or procedures (e.g., Trojan horse)
    - Confused deputy problem
- Biba model focuses on **external** threats

©Tatyana Ryutov

Biba Model Elements

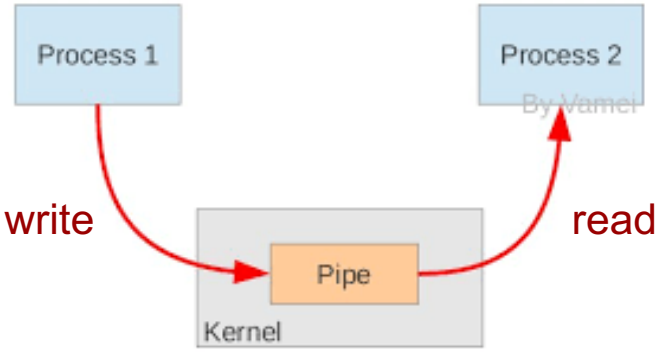
- Much like BLP for confidentiality
  - Subjects access objects
  - Policy is defined as set of relations on subjects and objects
  - Decision rule determines access
- Mandatory policy has levels and categories
  - User max and "min necessary" clearances
  - Object classification based on possible damage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

USC Viterbi School of Engineering

©Tatyana Ryutov

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



©Tatyana Ryutov

USC Viterbi
School of Engineering

Two Classes of Integrity Policy

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

USC Viterbi School of Engineering

©Tatyana Ryutov

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

©Tatyana Ryutov

USC Viterbi School of Engineering

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

©Tatyana Ryutov

USC Viterbi School of Engineering

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

©Tatyana Ryutov

USC Viterbi School of Engineering

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) = (released, {partner})
  For subject s $\subseteq$ S, I(s) = (beta, {$\varnothing$})
  Then: I(s) *leq* I(o)
- Example 2:
  For object o $\subseteq$ O, I(o) = (released, {partner})
  For subject s $\subseteq$ S, I(s) = (beta, {partner, customer})
  Then: I(s) and I(o) are non-comparable

©Tatyana Ryutov

USC Viterbi School of Engineering

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

©Tatyana Ryutov

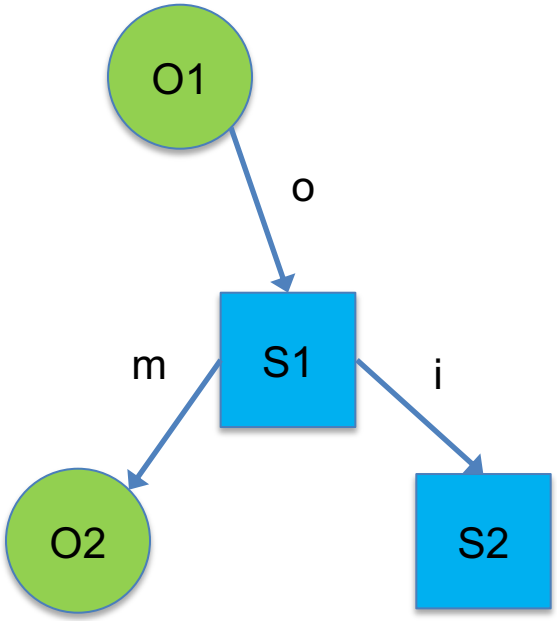
Biba: Mandatory Integrity Policies 2

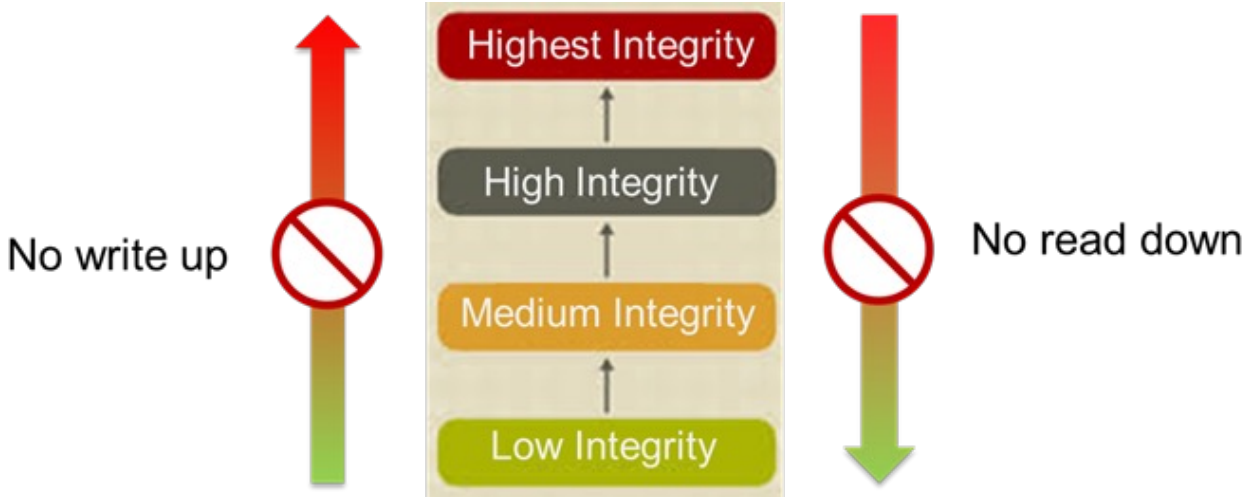
- **Ring** policy:
  1. Observation *allowed for any integrity level*
  2. For modify, I(o) leq I(s)
  3. For invoke, $I(s_{invoked})$ leq $I(s_{invoker})$
  - Does not address indirect modification
    - A subject can read a less trusted object, then the subject can modify data at its own integrity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

©Tatyana Ryutov

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 $I(s_{invoker})$





©Tatyana Ryutov

USC Viterbi
School of Engineering

Biba Strict Integrity Policy Interpretation

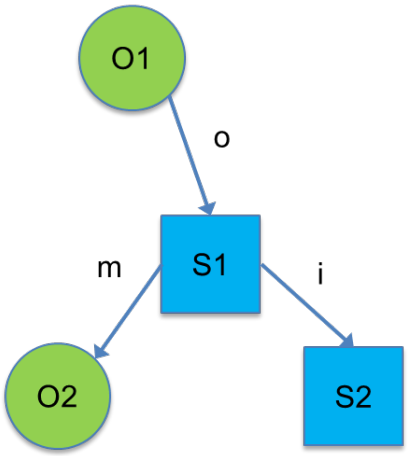
- Define dual of BLP *-Property:
  - BLP: S can write O IFF O dom S (“no write down”)
  - Biba: S can write O IFF i(O) *leq* i(S) (“no write up”)
- No unauthorized direct modification of objects
  - Limit write access to subjects of sufficient privilege
  - Integrity access class reflects damage from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

©Tatyana Ryutov

USC Viterbi School of Engineering

Strict Integrity Policy Examples

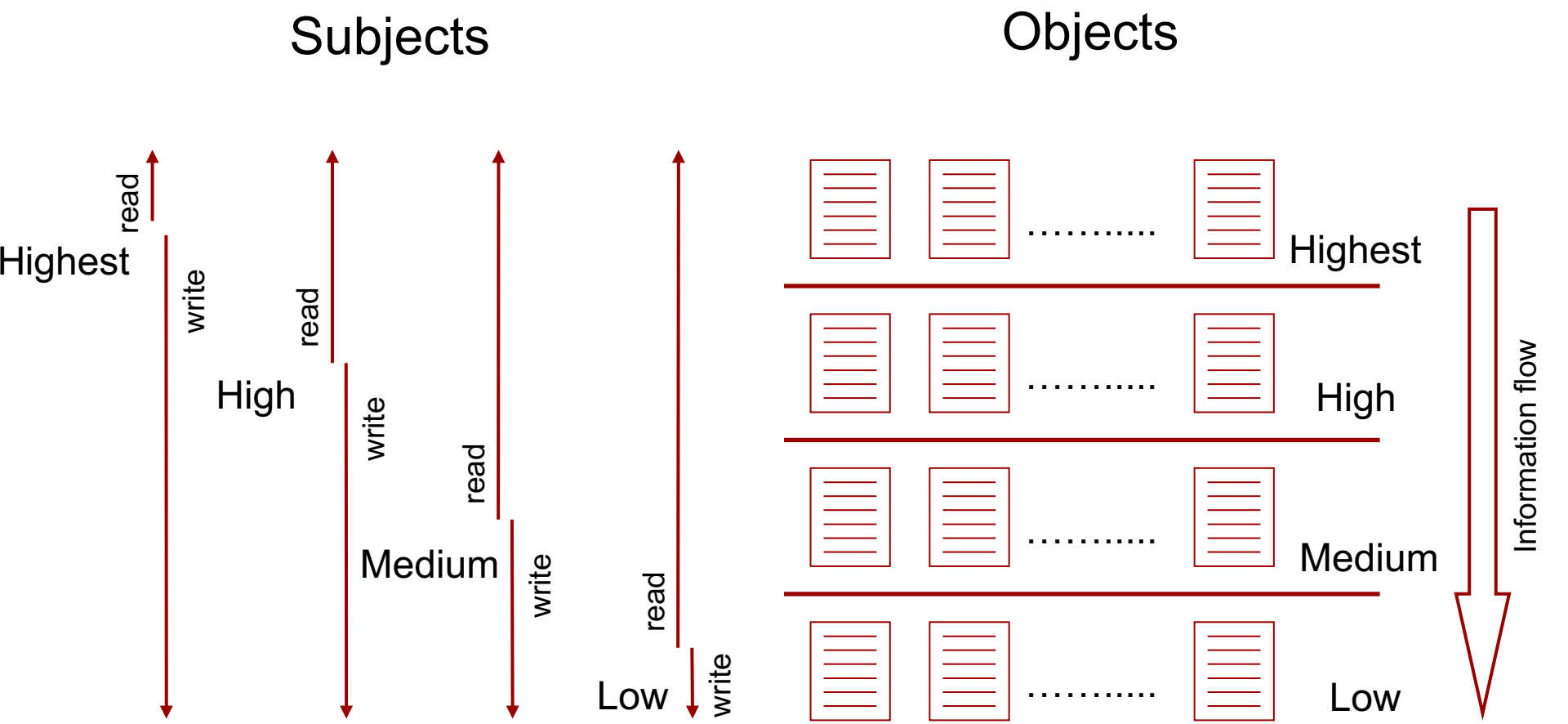
- Consider SW vendor integrity access classes
  - Levels: released > beta > demo
  - Categories: {internal, partner, customer}
- Consider **object** with integrity class (beta, {internal, partner})
- Example: what observe/modify access for these **subjects**:
  (beta, {internal, partner})
  (released,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



USC Viterbi School of Engineering

©Tatyana Ryutov

Biba Information Flow

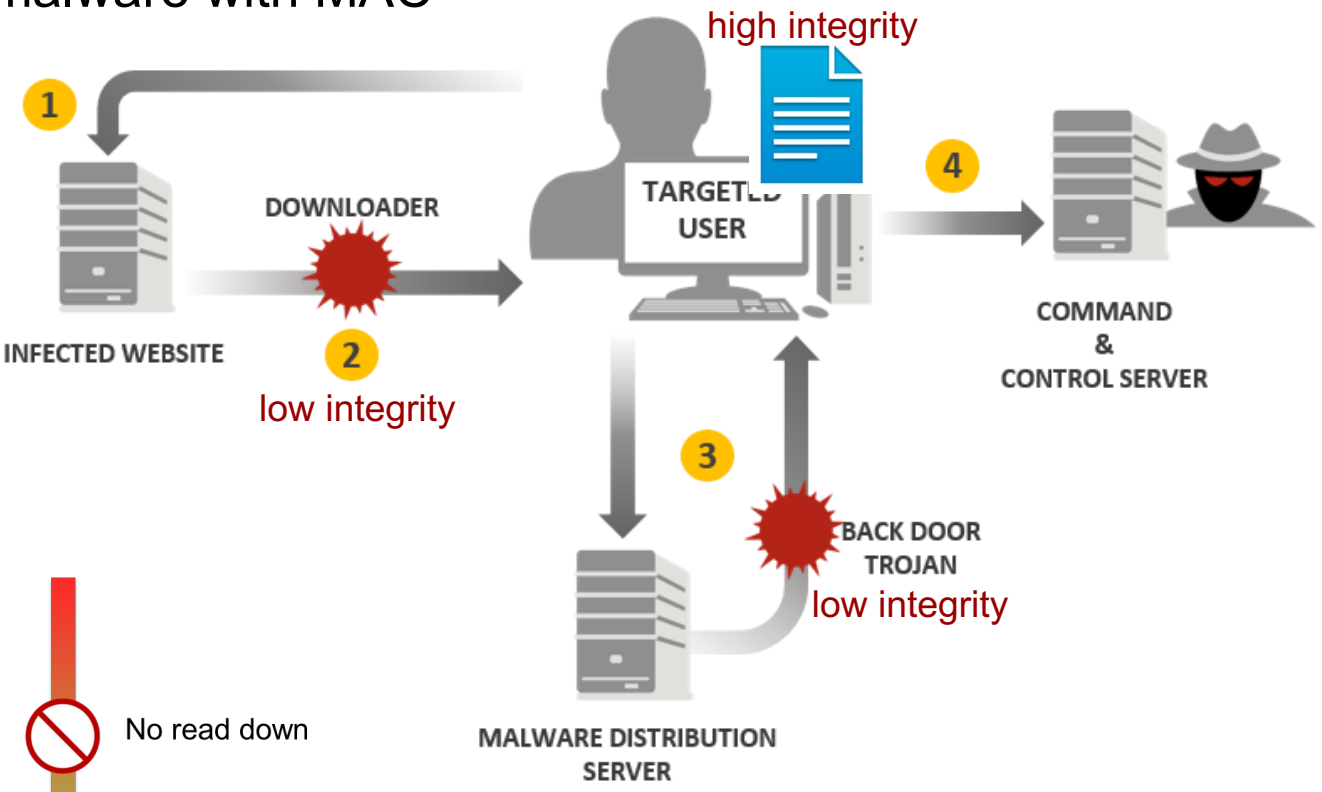


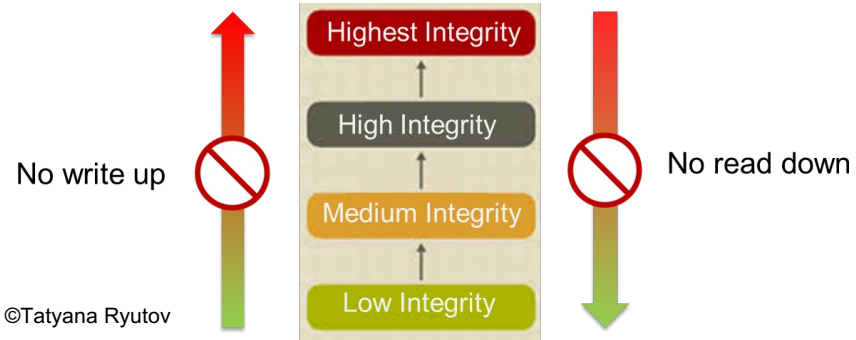
©Tatyana Ryutov

USC Viterbi
School of Engineering

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





©Tatyana Ryutov