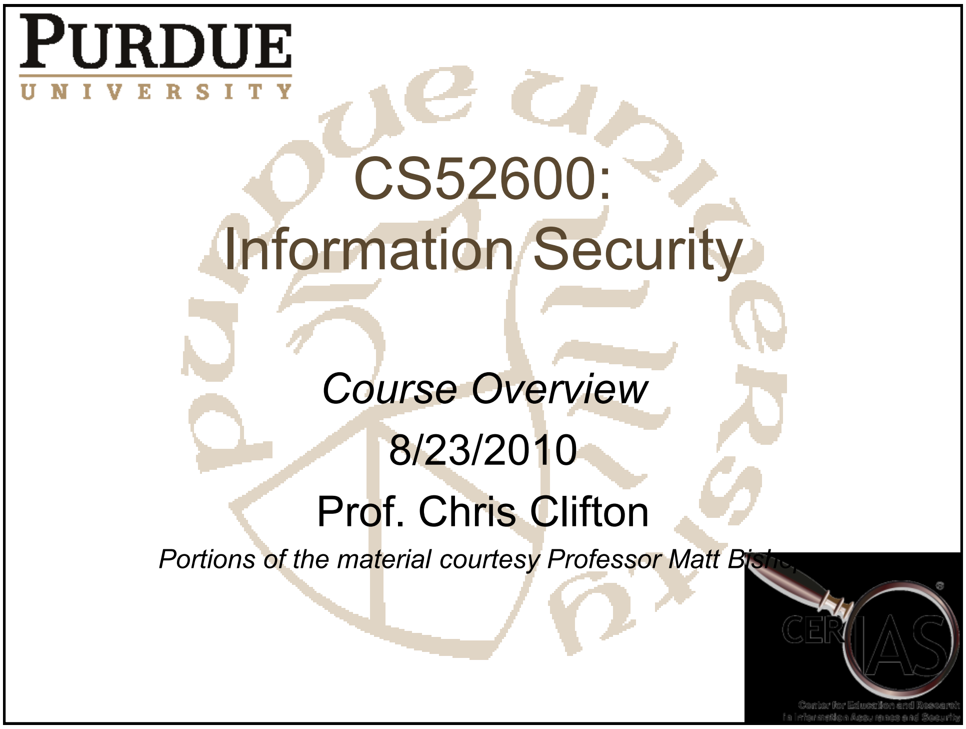


PURDUE UNIVERSITY

CS52600: Information Security

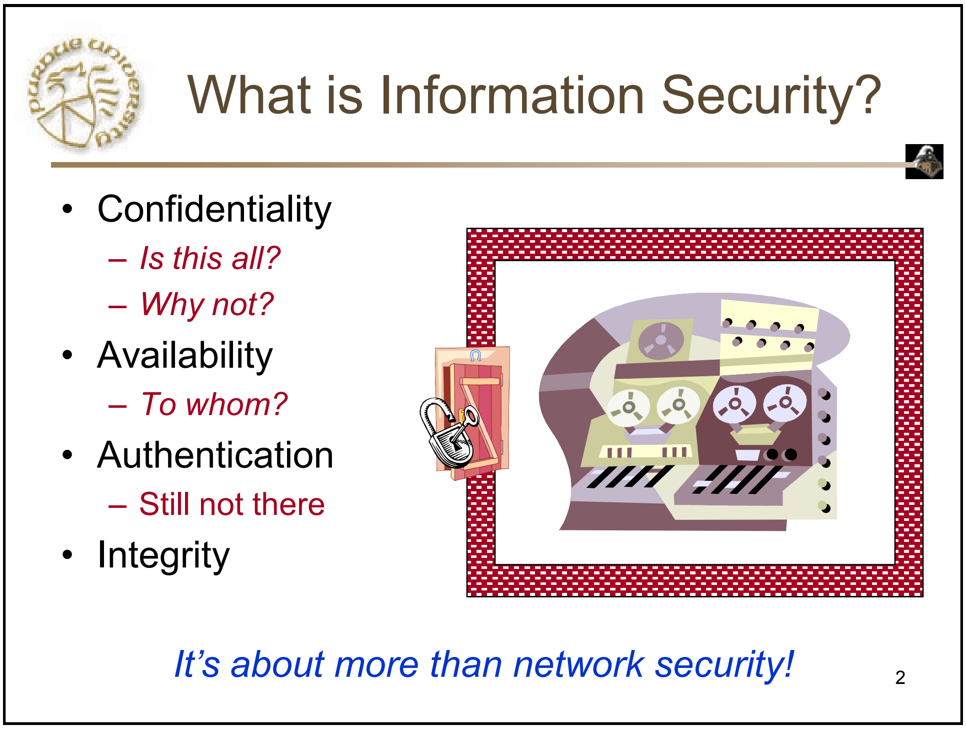
Course Overview

8/23/2010

Prof. Chris Clifton

Portions of the material courtesy Professor Matt Bishop

CERIAS
Center for Education and Research in Information Assurance and Security

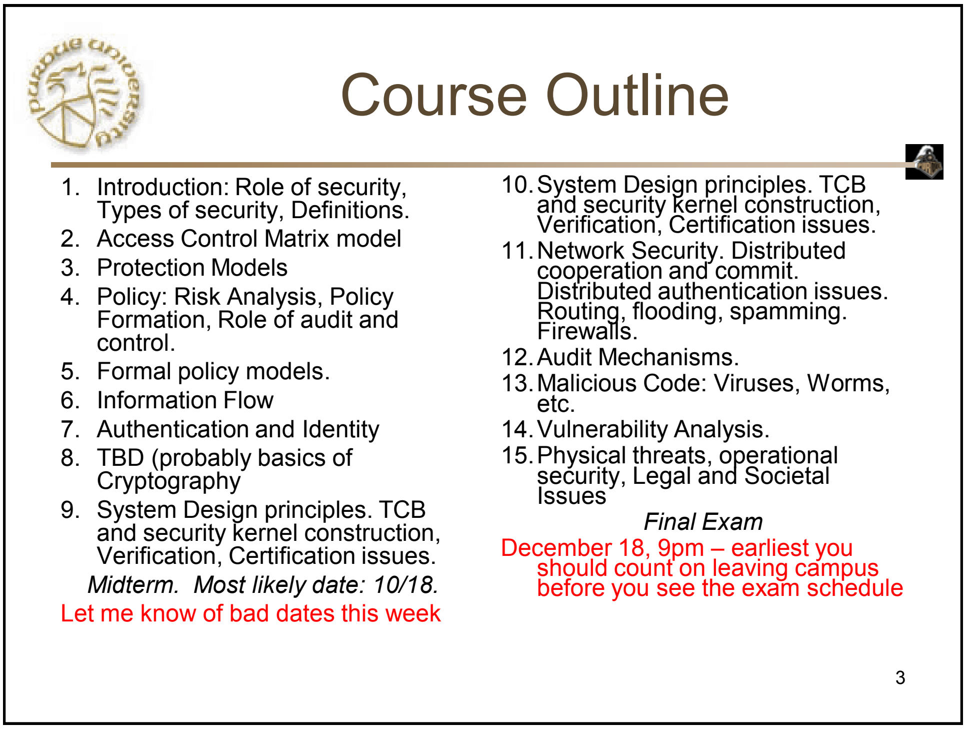


What is Information Security?

- Confidentiality
  - *Is this all?*
  - *Why not?*
- Availability
  - *To whom?*
- Authentication
  - Still not there
- Integrity

It's about more than network security!

2



Course Outline

1. Introduction: Role of security, Types of security, Definitions.
2. Access Control Matrix model
3. Protection Models
4. Policy: Risk Analysis, Policy Formation, Role of audit and control.
5. Formal policy models.
6. Information Flow
7. Authentication and Identity
8. TBD (probably basics of Cryptography
9. System Design principles. TCB and security kernel construction, Verification, Certification issues.

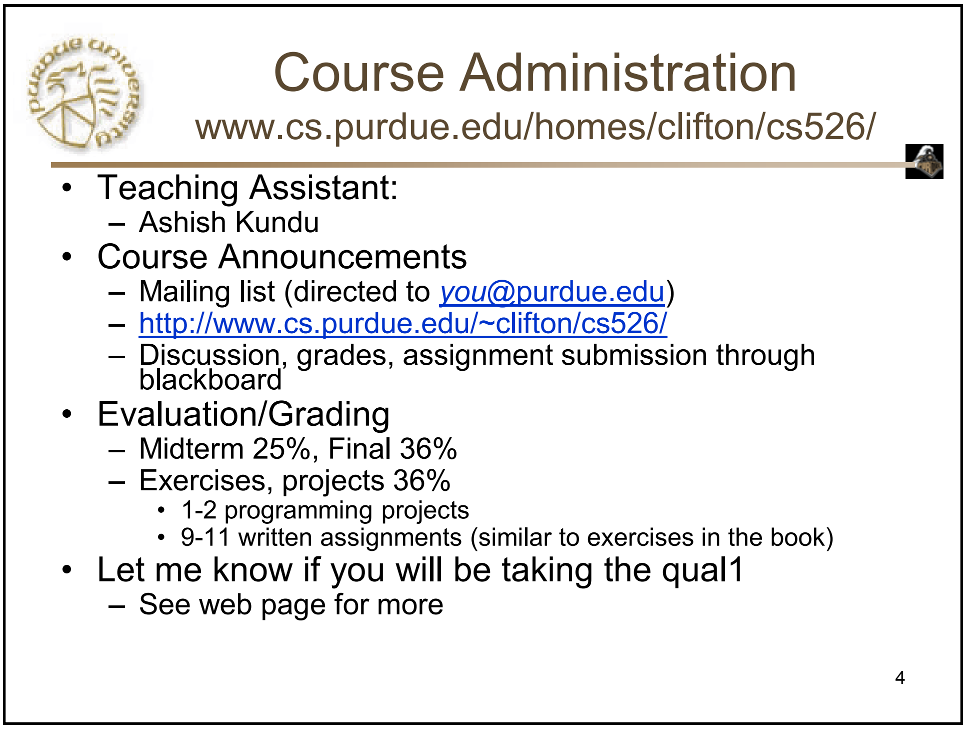
Midterm. Most likely date: 10/18.

Let me know of bad dates this week

10. System Design principles. TCB and security kernel construction, Verification, Certification issues.
11. Network Security. Distributed cooperation and commit. Distributed authentication issues. Routing, flooding, spamming. Firewalls.
12. Audit Mechanisms.
13. Malicious Code: Viruses, Worms, etc.
14. Vulnerability Analysis.
15. Physical threats, operational security, Legal and Societal Issues

Final Exam

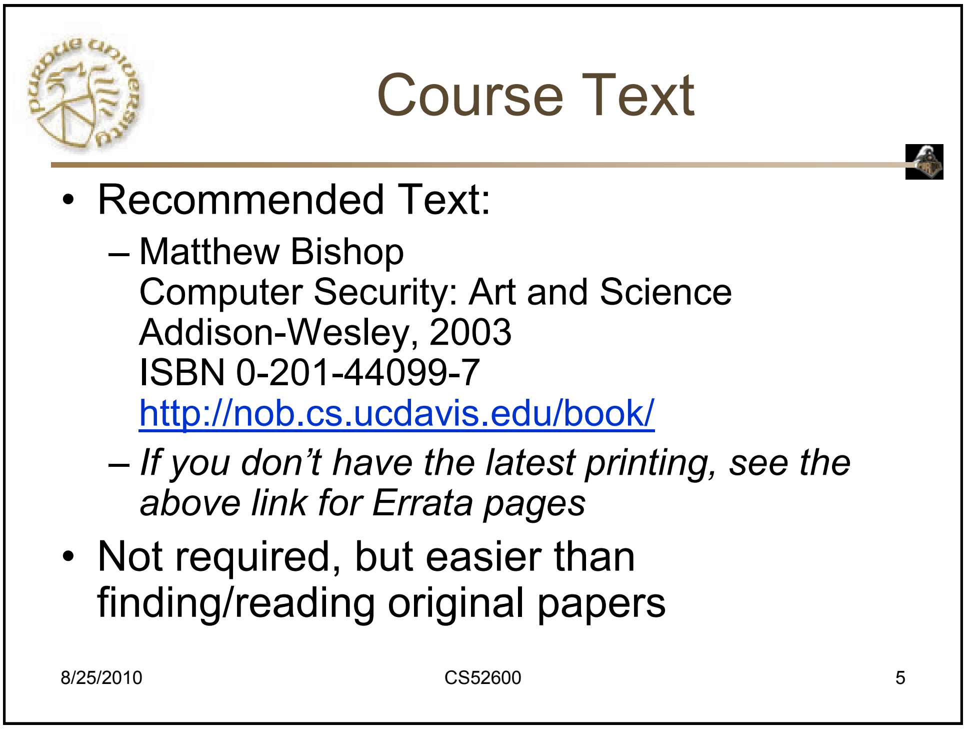
December 18, 9pm – earliest you should count on leaving campus before you see the exam schedule



Course Administration

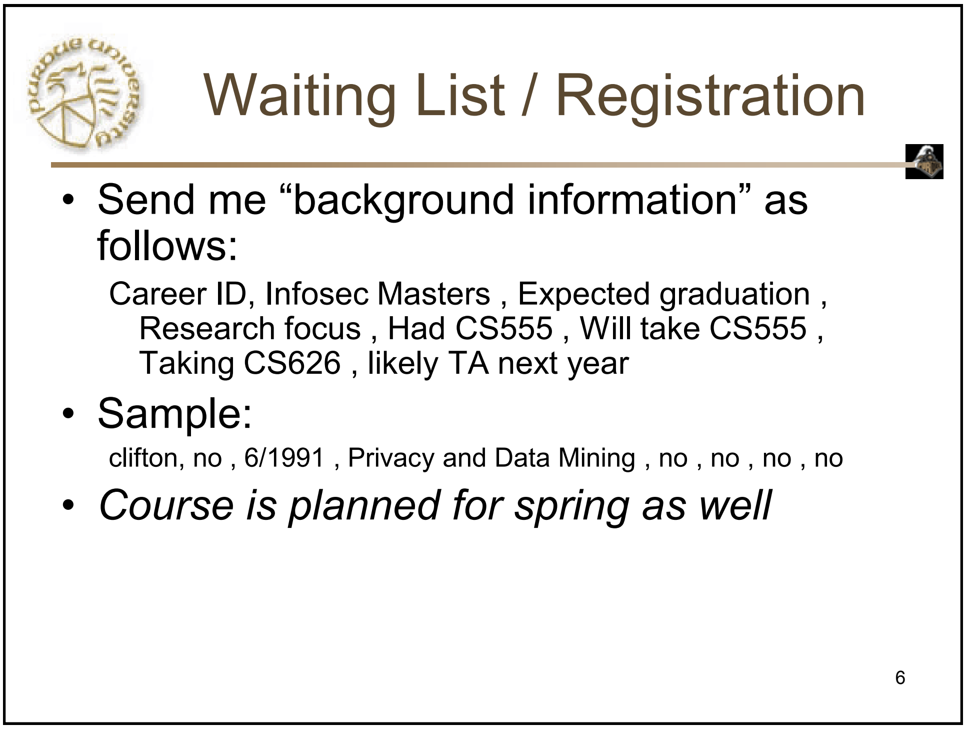
www.cs.purdue.edu/homes/clifton/cs526/

- Teaching Assistant:
  - Ashish Kundu
- Course Announcements
  - Mailing list (directed to *you*@purdue.edu)
  - http://www.cs.purdue.edu/~clifton/cs526/
  - Discussion, grades, assignment submission through blackboard
- Evaluation/Grading
  - Midterm 25%, Final 36%
  - Exercises, projects 36%
    - 1-2 programming projects
    - 9-11 written assignments (similar to exercises in the book)
- Let me know if you will be taking the qual1
  - See web page for more



Course Text

- Recommended Text:
  - Matthew Bishop
    Computer Security: Art and Science
    Addison-Wesley, 2003
    ISBN 0-201-44099-7
    http://nob.cs.ucdavis.edu/book/
  - *If you don't have the latest printing, see the above link for Errata pages*
- Not required, but easier than finding/reading original papers



Waiting List / Registration

- Send me "background information" as follows:

  Career ID, Infosec Masters , Expected graduation , Research focus , Had CS555 , Will take CS555 , Taking CS626 , likely TA next year
- Sample:

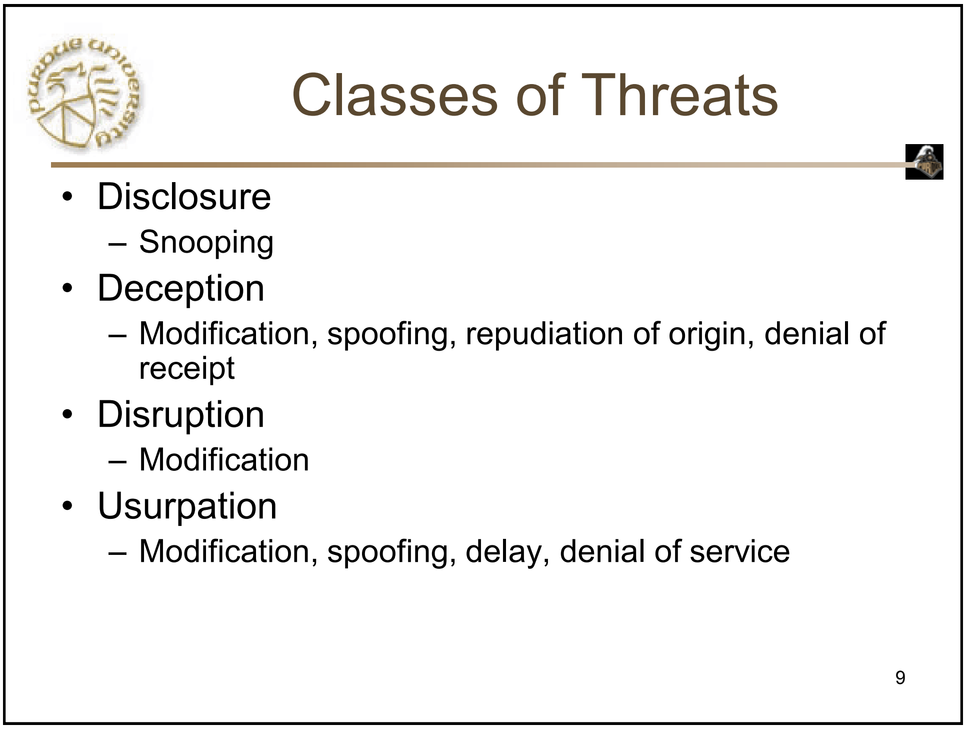
  clifton, no , 6/1991 , Privacy and Data Mining , no , no , no , no
- *Course is planned for spring as well*

Introduction

- Components of computer security
- Threats
- Policies and mechanisms
- The role of trust
- Assurance
- Operational Issues
- Human Issues

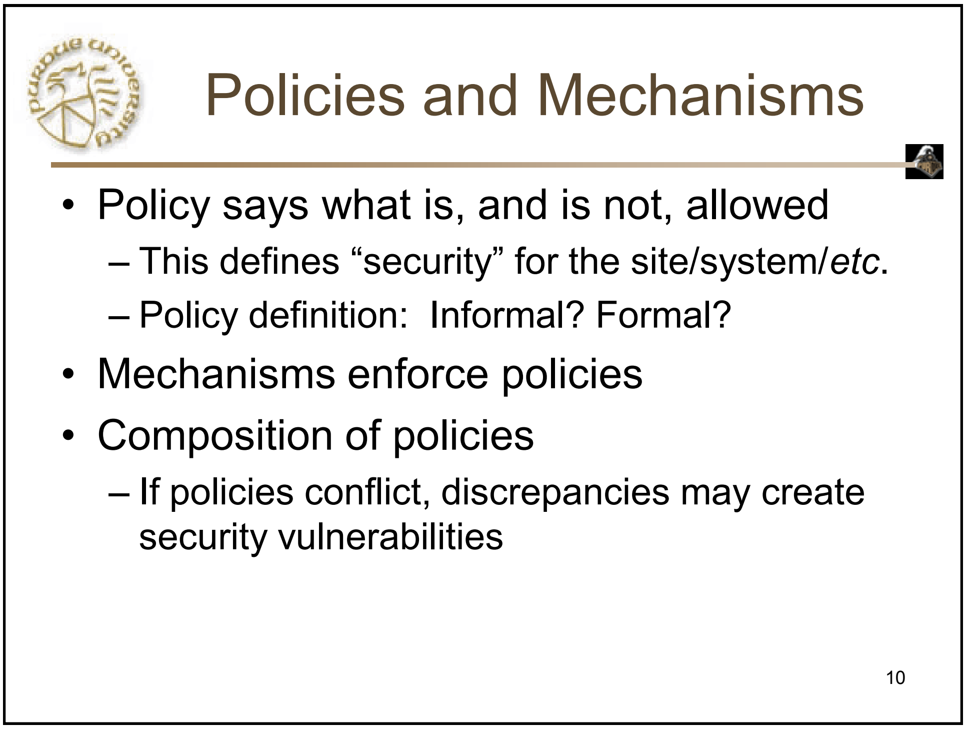
Basic Components

- Confidentiality
  - Keeping data and resources hidden
- Integrity
  - Data integrity (integrity)
  - Origin integrity (authentication)
- Availability
  - Enabling access to data and resources



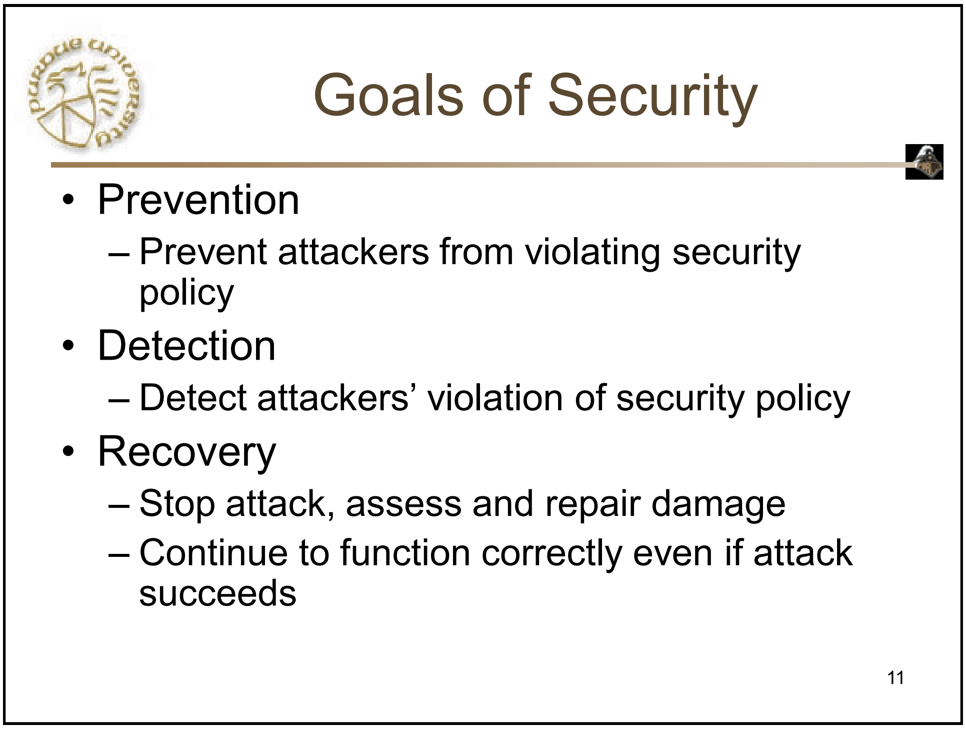
Classes of Threats

- Disclosure
  - Snooping
- Deception
  - Modification, spoofing, repudiation of origin, denial of receipt
- Disruption
  - Modification
- Usurpation
  - Modification, spoofing, delay, denial of service



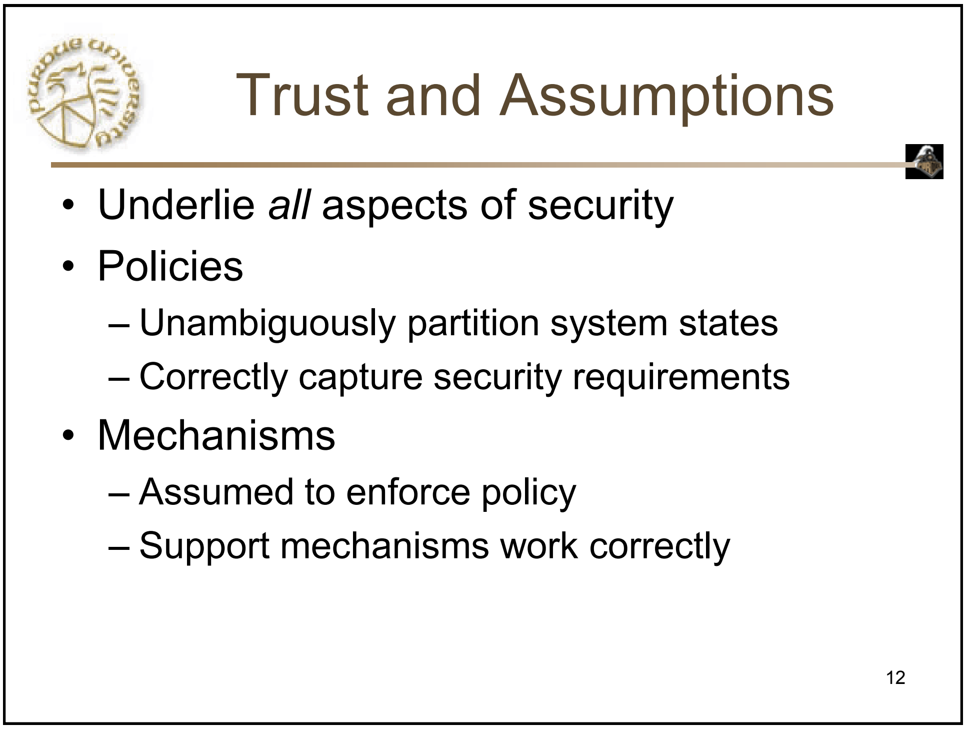
Policies and Mechanisms

- Policy says what is, and is not, allowed
  - This defines “security” for the site/system/*etc*.
  - Policy definition: Informal? Formal?
- Mechanisms enforce policies
- Composition of policies
  - If policies conflict, discrepancies may create security vulnerabilities



Goals of Security

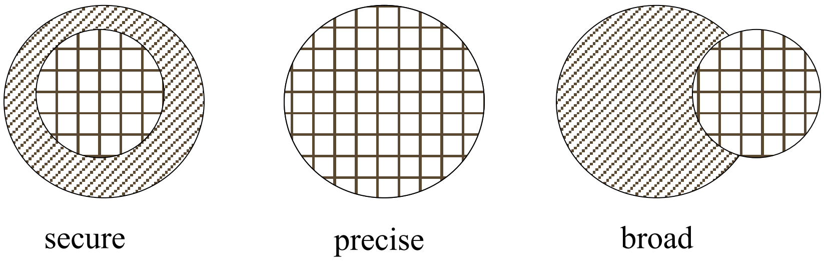
- Prevention
  - Prevent attackers from violating security policy
- Detection
  - Detect attackers’ violation of security policy
- Recovery
  - Stop attack, assess and repair damage
  - Continue to function correctly even if attack succeeds



Trust and Assumptions

- Underlie *all* aspects of security
- Policies
  - Unambiguously partition system states
  - Correctly capture security requirements
- Mechanisms
  - Assumed to enforce policy
  - Support mechanisms work correctly

Types of Mechanisms



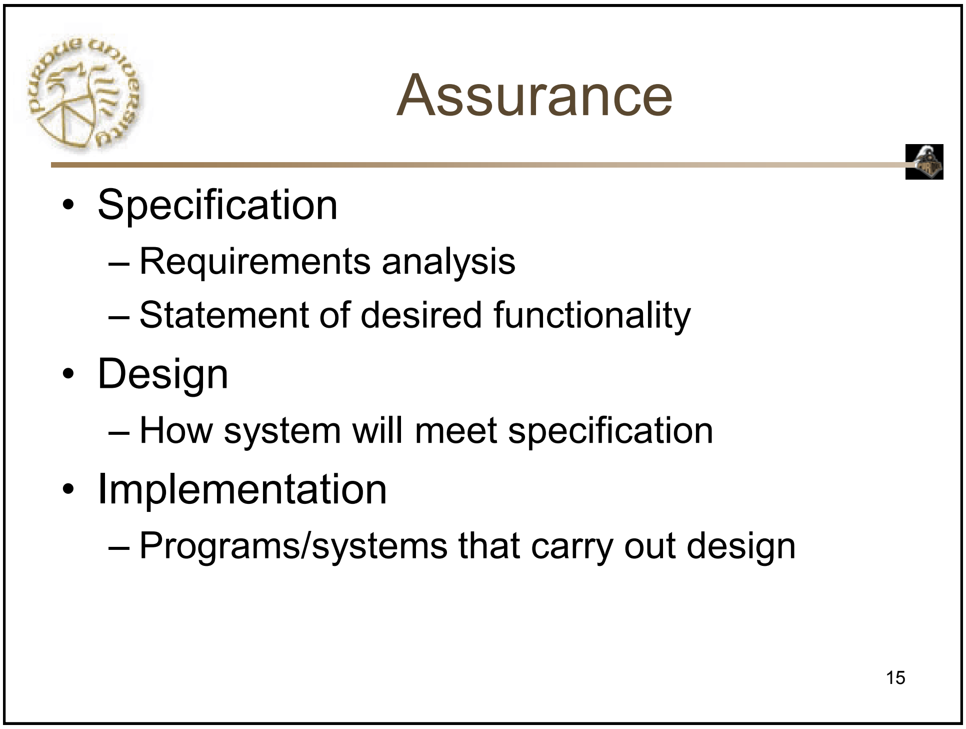
▦ set of reachable states ▨ set of secure states

13

PURDUE UNIVERSITY

CS526: Information Security Access Control Matrices

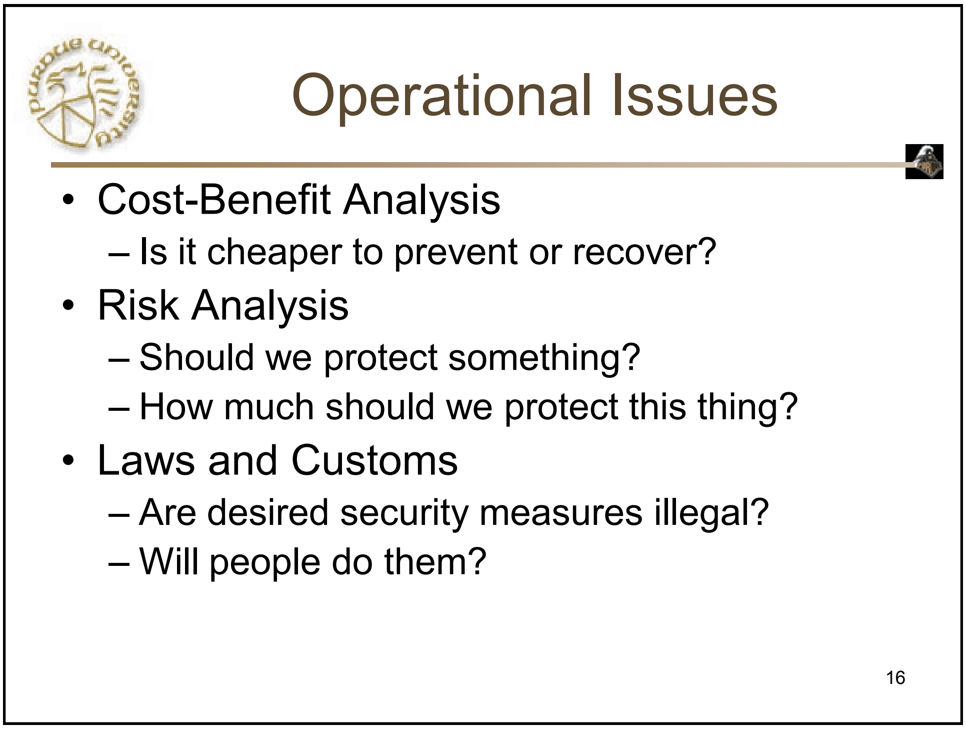
Prof. Chris Clifton
August 25, 2010



Assurance

- Specification
  - Requirements analysis
  - Statement of desired functionality
- Design
  - How system will meet specification
- Implementation
  - Programs/systems that carry out design

15



Operational Issues

- Cost-Benefit Analysis
  - Is it cheaper to prevent or recover?
- Risk Analysis
  - Should we protect something?
  - How much should we protect this thing?
- Laws and Customs
  - Are desired security measures illegal?
  - Will people do them?

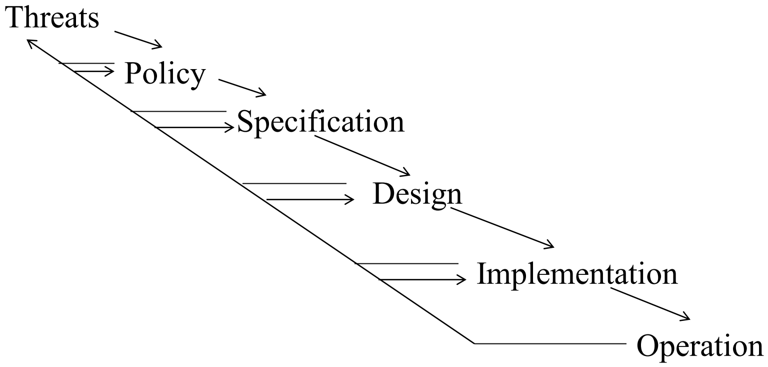
16

Human Issues

- Organizational Problems
  - Power and responsibility
  - Financial benefits
- People problems
  - Outsiders and insiders
    - *Which do you think is the bigger problem?*
  - Social engineering

17

Tying the Definitions Together

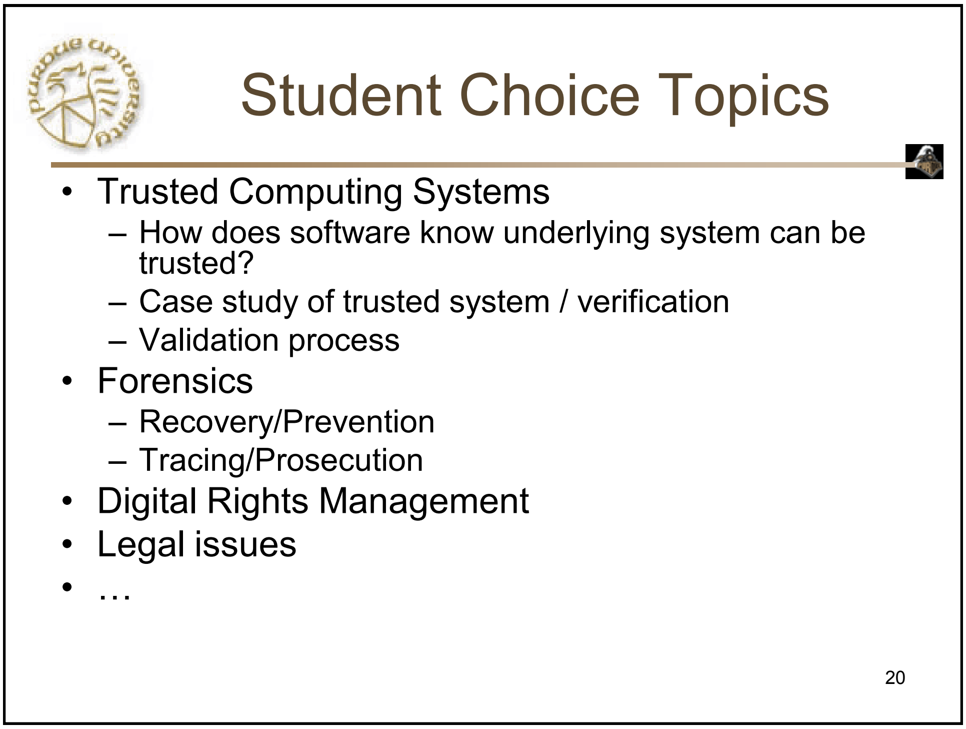


18

Key Points

- Policy defines security, and mechanisms enforce security
  - Confidentiality
  - Integrity
  - Availability
- Trust and knowing assumptions
- Importance of assurance
- The human factor

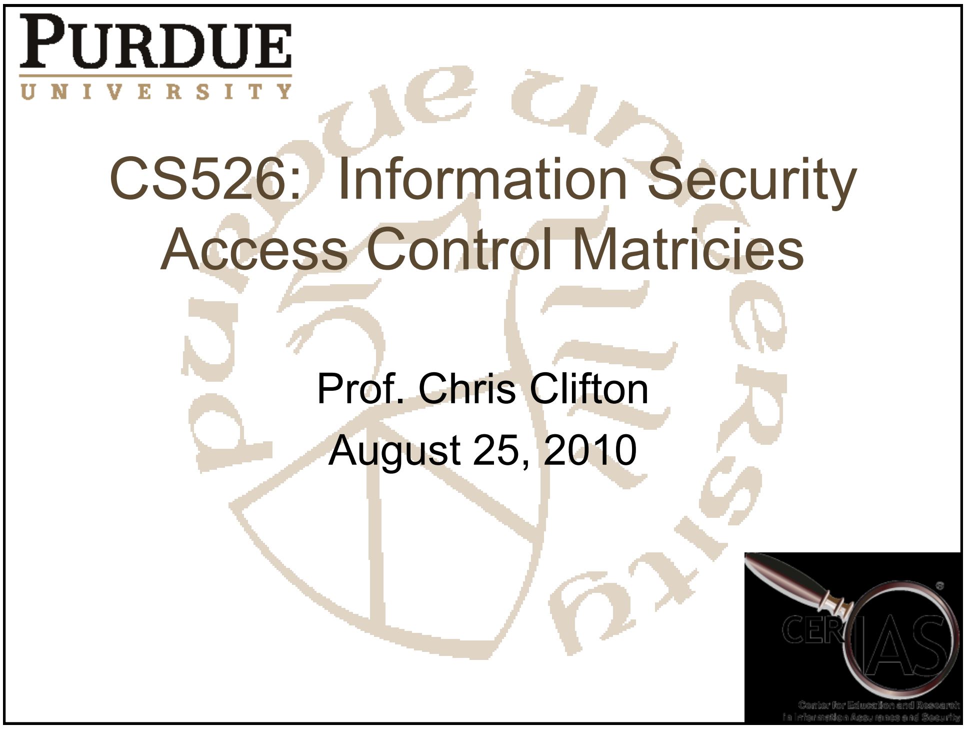
19



Student Choice Topics

- Trusted Computing Systems
  - How does software know underlying system can be trusted?
  - Case study of trusted system / verification
  - Validation process
- Forensics
  - Recovery/Prevention
  - Tracing/Prosecution
- Digital Rights Management
- Legal issues
- …

20

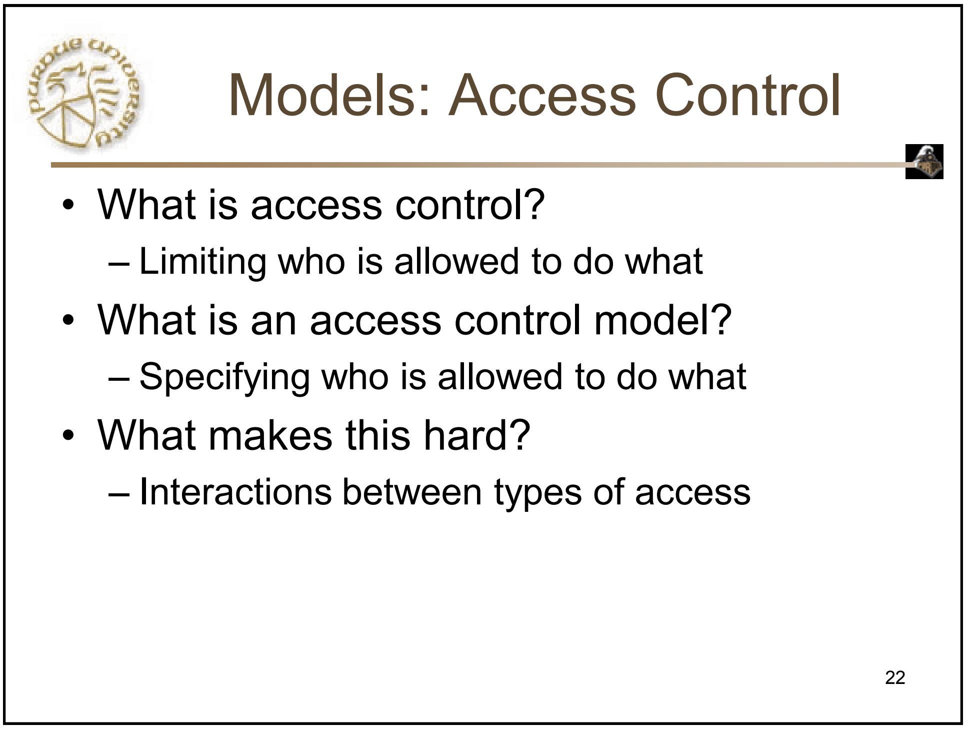


PURDUE UNIVERSITY

CS526: Information Security
Access Control Matricies

Prof. Chris Clifton

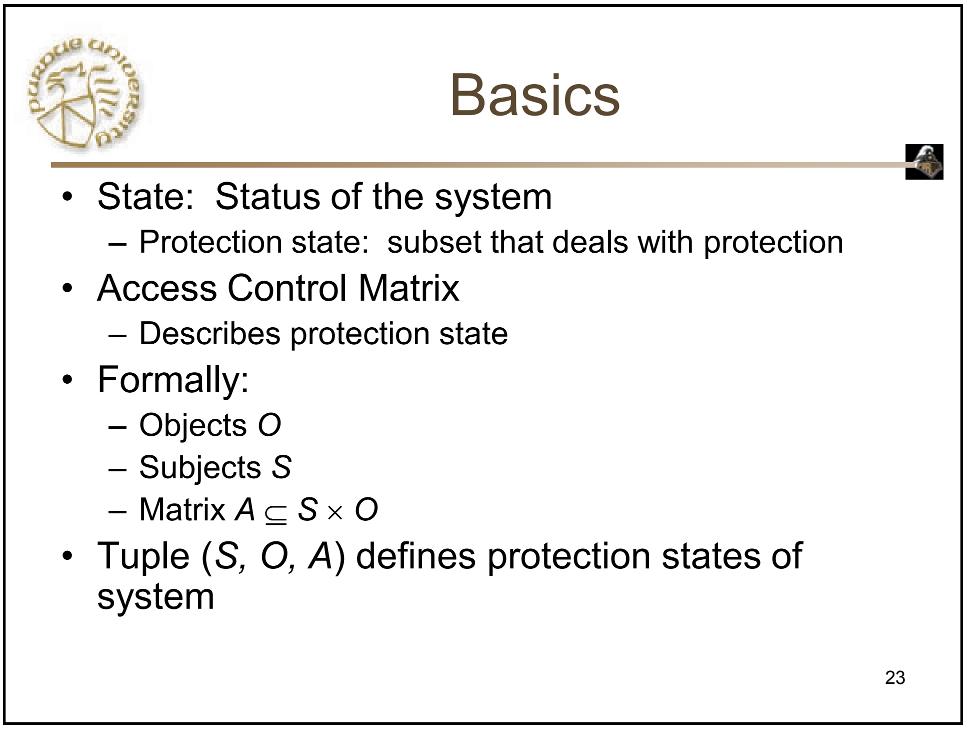
August 25, 2010



Models: Access Control

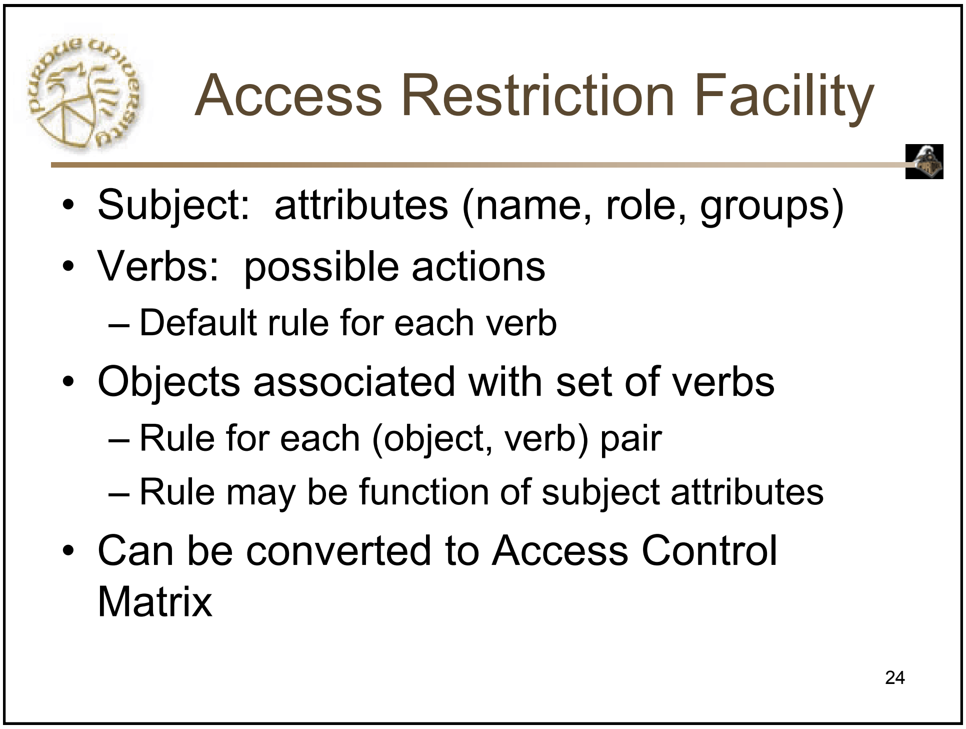
- What is access control?
  - – Limiting who is allowed to do what
- What is an access control model?
  - – Specifying who is allowed to do what
- What makes this hard?
  - – Interactions between types of access

22



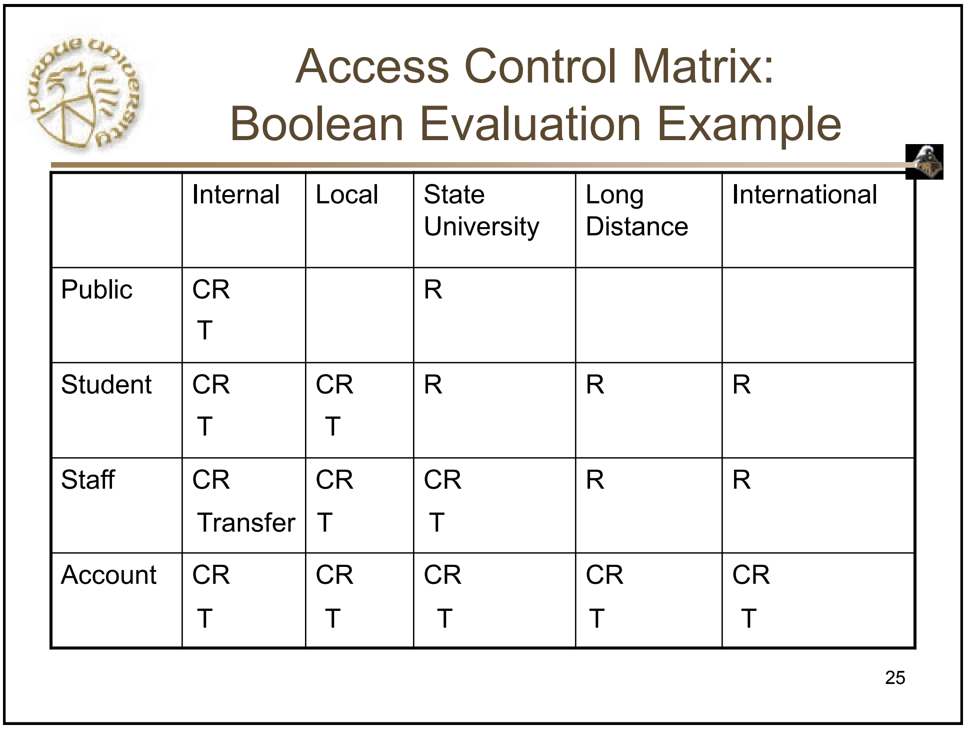
Basics

- State: Status of the system
  - Protection state: subset that deals with protection
- Access Control Matrix
  - Describes protection state
- Formally:
  - Objects *O*
  - Subjects *S*
  - Matrix $A \subseteq S \times O$
- Tuple (S, O, A) defines protection states of system



Access Restriction Facility

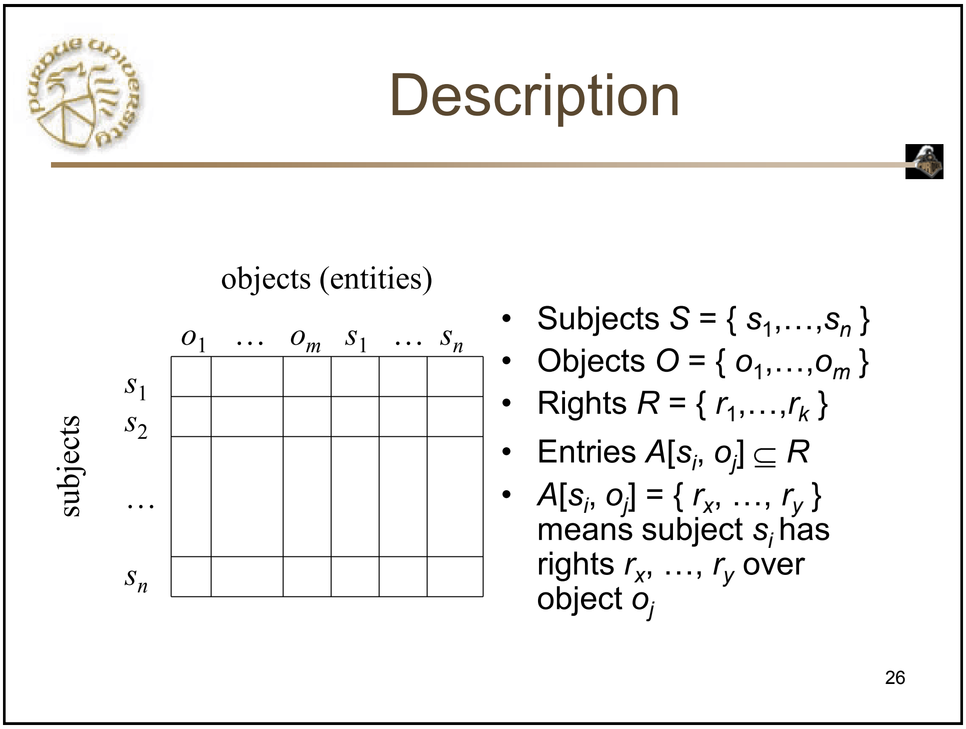
- Subject: attributes (name, role, groups)
- Verbs: possible actions
  - Default rule for each verb
- Objects associated with set of verbs
  - Rule for each (object, verb) pair
  - Rule may be function of subject attributes
- Can be converted to Access Control Matrix



Access Control Matrix: Boolean Evaluation Example

	Internal	Local	State University	Long Distance	International
Public	CR T		R		
Student	CR T	CR T	R	R	R
Staff	CR Transfer	CR T	CR T	R	R
Account	CR T	CR T	CR T	CR T	CR T

25



Description

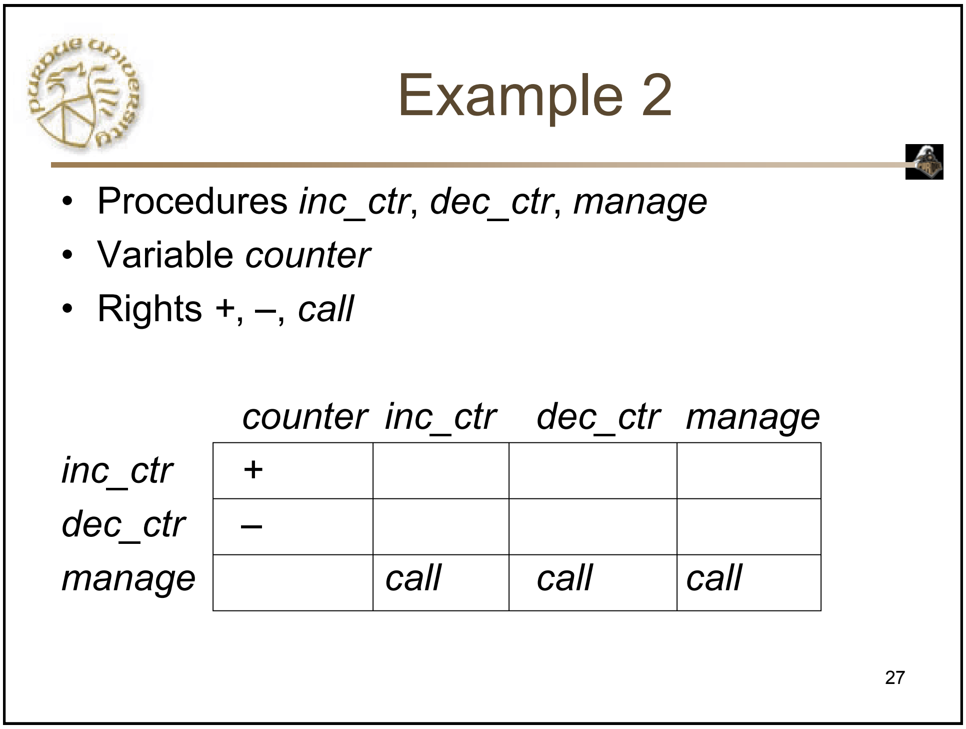
objects (entities)

subjects

$$\begin{array}{c|c|c|c|c|c|c|}
 & o_1 & \ldots & o_m & s_1 & \ldots & s_n \\ \hline
s_1 & & & & & & \\ \hline
s_2 & & & & & & \\ \hline
\ldots & & & & & & \\ \hline
s_n & & & & & & \\ \hline
\end{array}$$

- 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

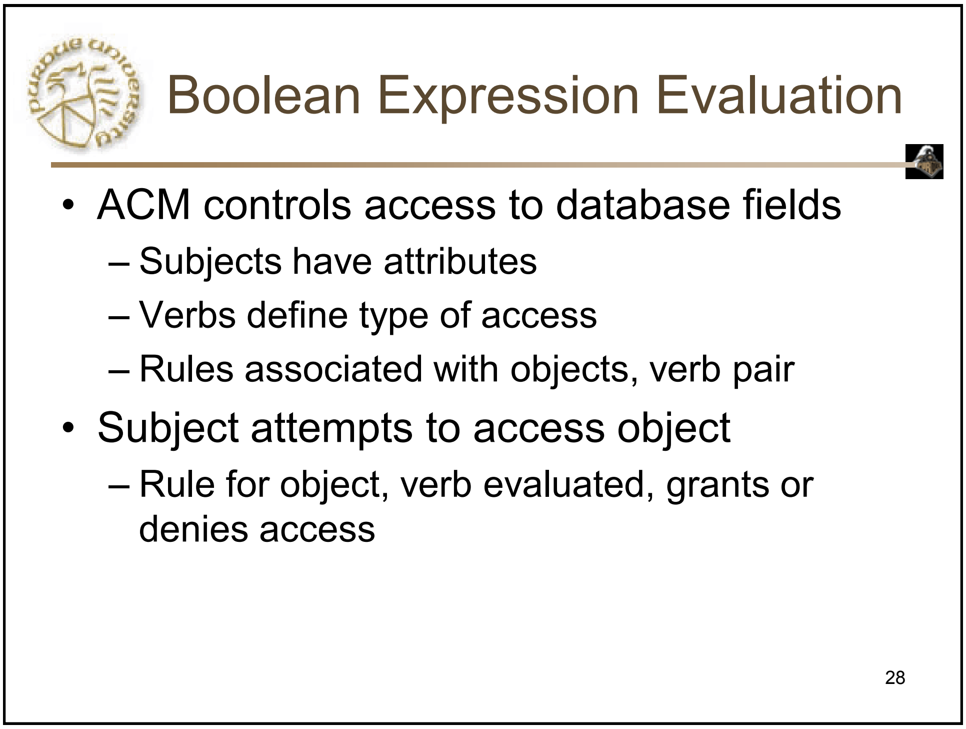
26



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*



Boolean Expression Evaluation

- ACM controls access to database fields
  - – Subjects have attributes
  - – Verbs define type of access
  - – Rules associated with objects, verb pair
- Subject attempts to access object
  - – Rule for object, verb evaluated, grants or denies access

Example

- Subject annie
  - Attributes role (artist), groups (creative)
- Verb paint
  - Default 0 (deny unless explicitly granted)
- Object picture
  - Rule:
    paint: ‘artist’ in subject.role and
    ‘creative’ in subject.groups and
    time.hour >= 0 and time.hour < 5

29

Protection State Transitions

- State $X_i = (S_i, O_i, A_i)$
- Transitions τ_i
  - Single transition $X_i \vdash_{\tau_{i+1}} X_{i+1}$
  - Series of transitions $X \vdash^* Y$
- Access control matrix may change
  - Change command c associated with transition
  - $X_i \vdash_{c_{i+1}(p_{i+1},\ldots,p_{i+1})} X_{i+1}$
- Commands often called *transformation procedures*

31

Special Privileges: Copy, Ownership

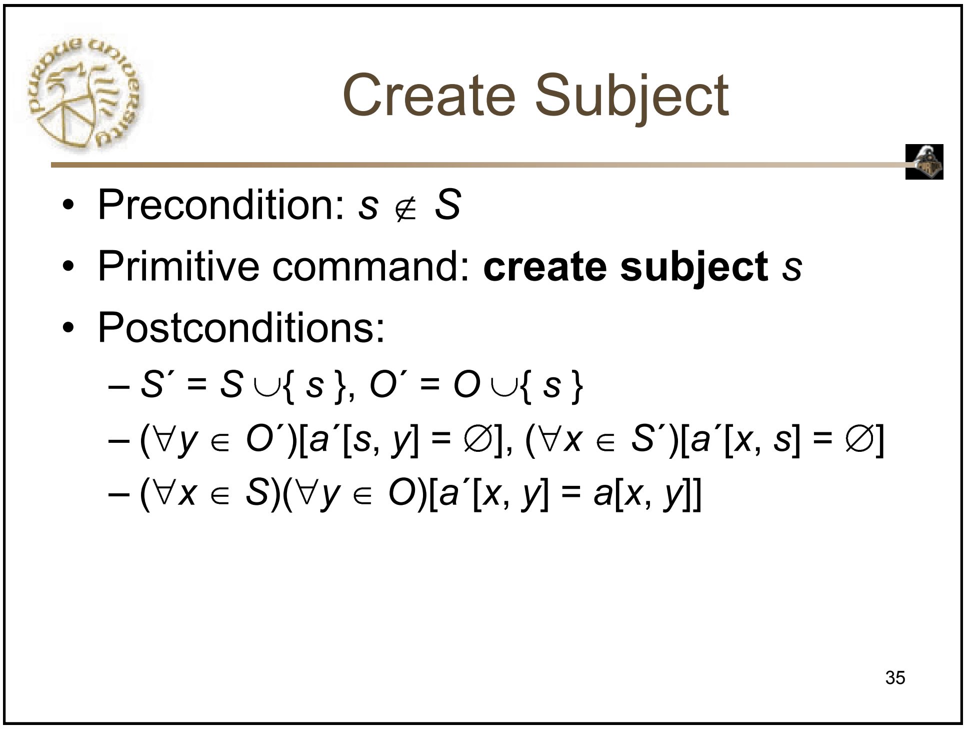
- Copy (or grant)
  - Possessor can extend privileges to another
- Own right
  - Possessor can change their own privileges
- Principle of Attenuation of Privilege
  - A subject may not give rights it does not possess

32

Primitive Commands
(Harrison, Ruzzo, Ullman CACM'76)

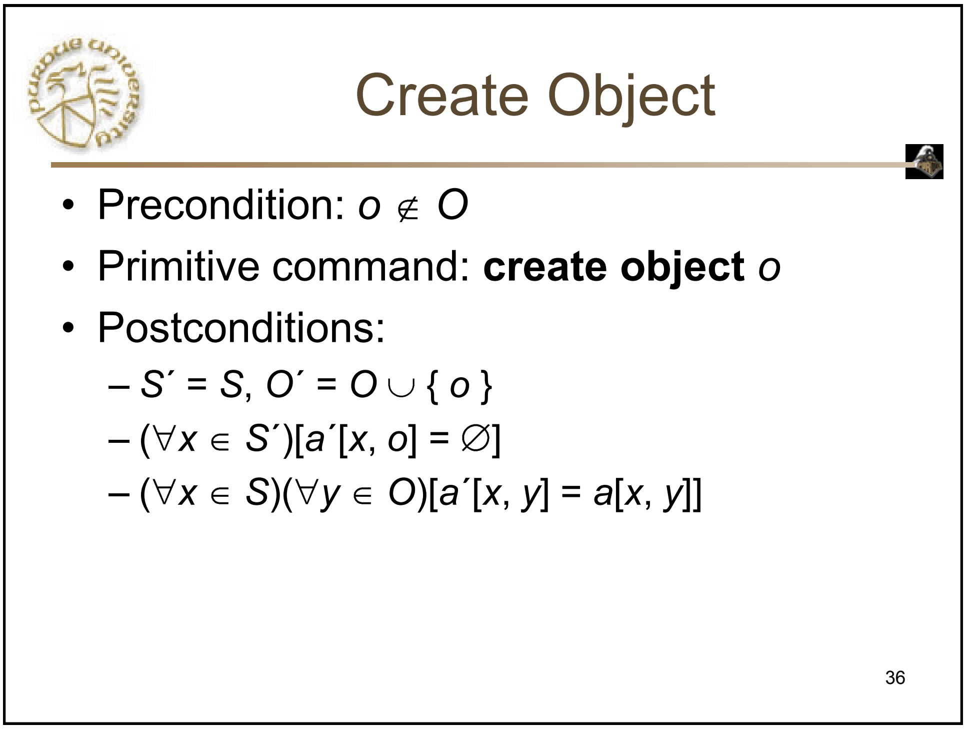
- Create Object *o*
  - Adds *o* to objects with no access
  - $S' = S$, $O' = O \cup \{o\}$, $(\forall x \in S')[a'[x,o] = \emptyset]$, $(\forall x \in S')(\forall y \in O)[a'[x,y] = a[x,y]]$
- Create Subject *s*
  - Adds *s* to objects, subjects, sets relevant access control to ∅
- Enter *r* into *a*[*s*,*o*]
- Delete *r* from *a*[*s*,*o*]
- Destroy subject *s*, destroy object *o*

33



Create Subject

- Precondition: $s \notin S$
- Primitive command: **create subject** *s*
- Postconditions:
  - $S' = S \cup \{ s \}$, $O' = O \cup \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$, $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$



Create Object

- Precondition: $o \notin O$
- Primitive command: **create object** *o*
- Postconditions:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$

Add Right

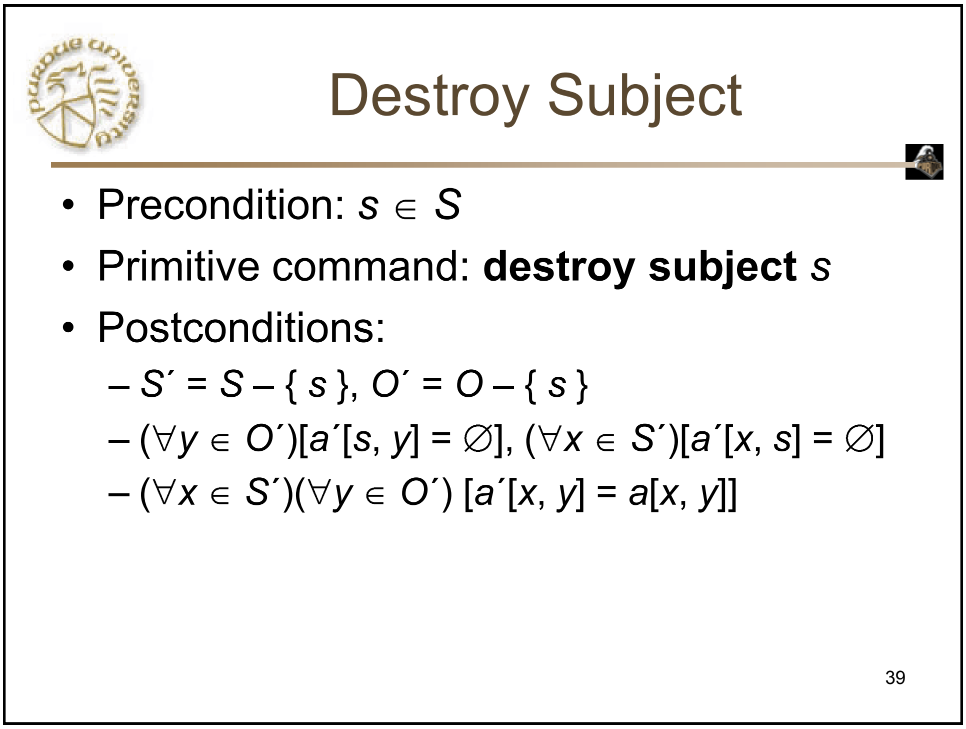
- Precondition: $s \in S$, $o \in O$
- Primitive command: enter r into $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] \cup \{ r \}$
  - $(\forall x,y \in S \text{x} O - \{ s,o \})$ $[a'[x, y] = a[x, y]]$

37

Delete Right

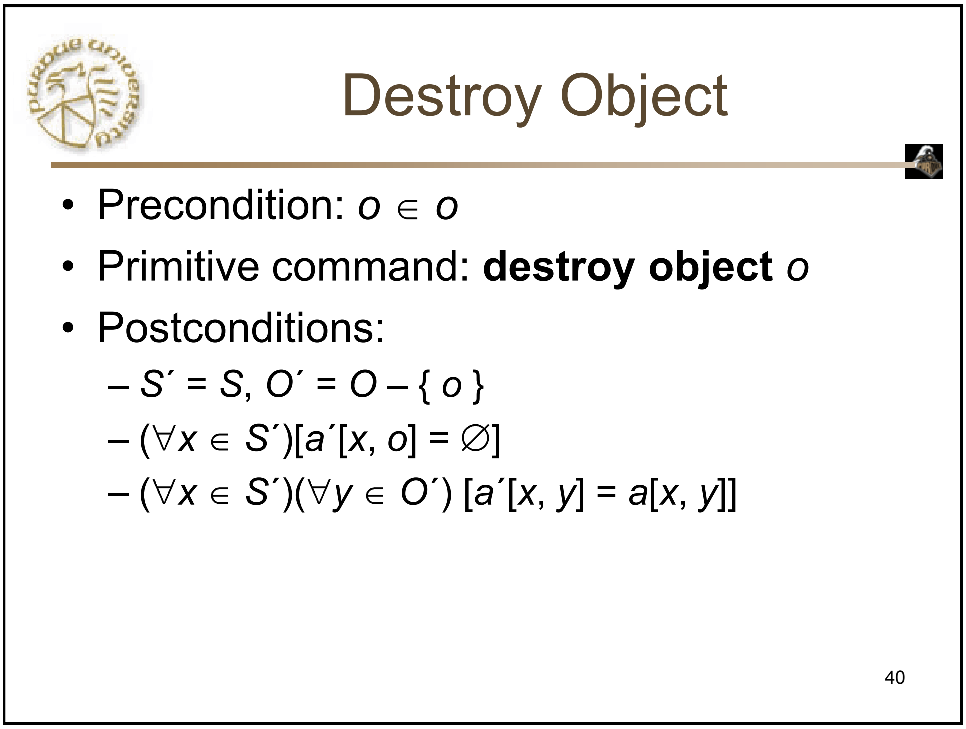
- Precondition: $s \in S$, $o \in O$
- Primitive command: **delete** r **from** $a[s, o]$
- Postconditions:
  - $S' = S$, $O' = O$
  - $a'[s, o] = a[s, o] - \{ r \}$
  - $(\forall x,y \in S \text{x} O - \{ s,o \})$ $[a'[x, y] = a[x, y]]$

38



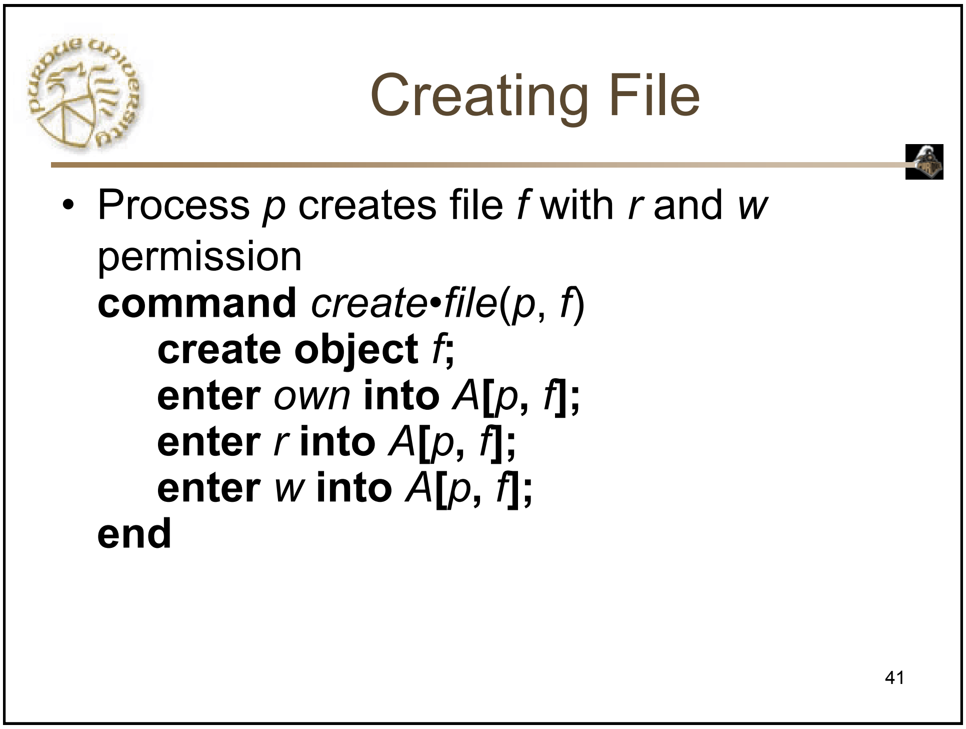
Destroy Subject

- Precondition: $s \in S$
- Primitive command: **destroy subject** *s*
- Postconditions:
  - $S' = S - \{ s \}$, $O' = O - \{ s \}$
  - $(\forall y \in O')[a'[s, y] = \varnothing]$, $(\forall x \in S')[a'[x, s] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$



Destroy Object

- Precondition: $o \in o$
- Primitive command: **destroy object** *o*
- Postconditions:
  - $S' = S$, $O' = O - \{ o \}$
  - $(\forall x \in S')[a'[x, o] = \varnothing]$
  - $(\forall x \in S')(\forall y \in O')\ [a'[x, y] = a[x, y]]$

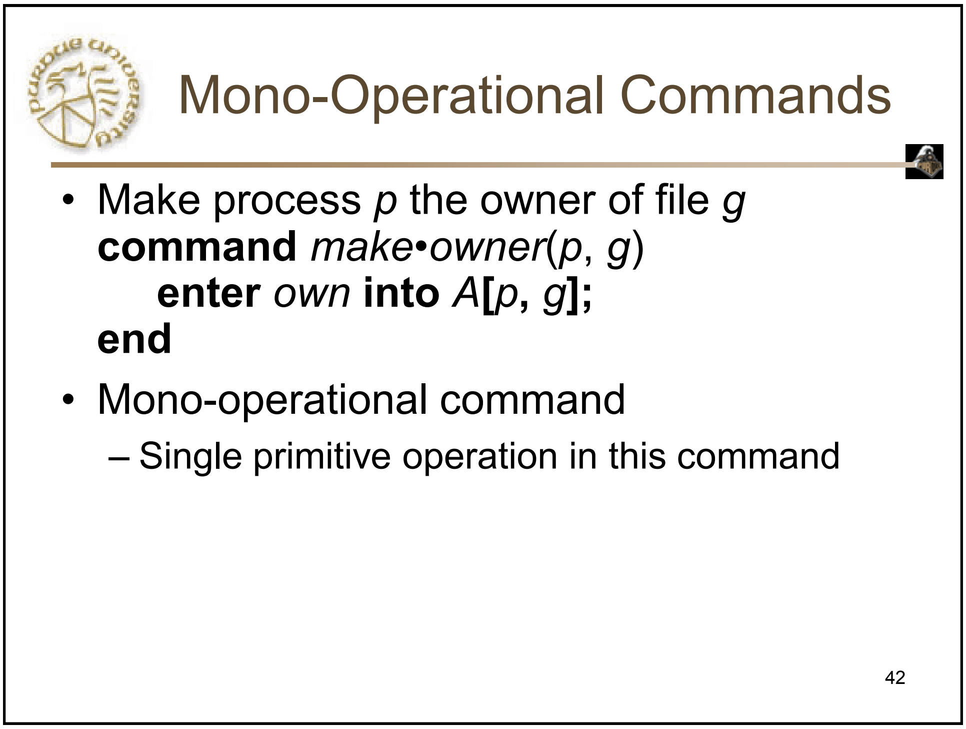


Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create•file(p, f)
    create object f;
    enter own into A[p, f];
    enter r into A[p, f];
    enter w into A[p, f];
end
```

41



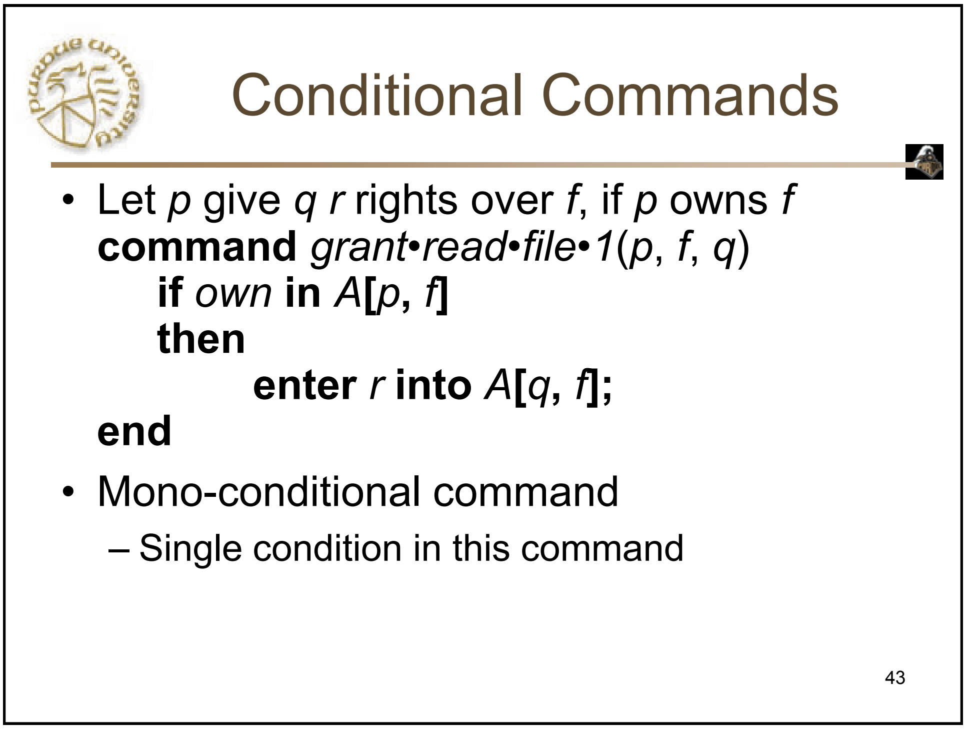
Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
    enter own into A[p, g];
end
```

- Mono-operational command
  - Single primitive operation in this command

42



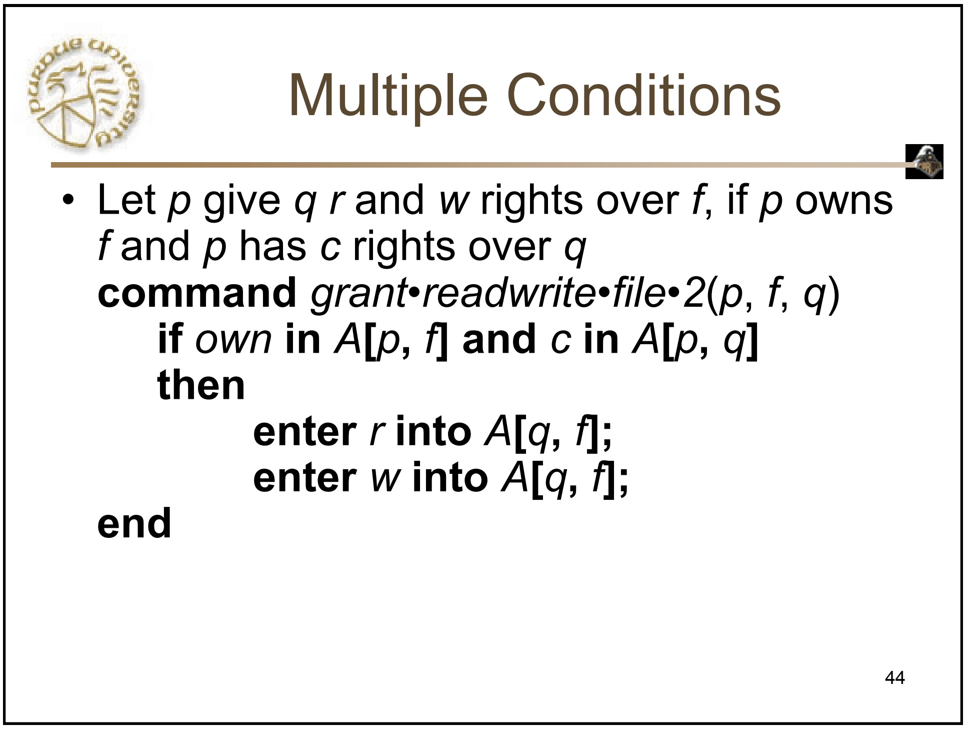
Conditional Commands

- Let *p* give *q r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
    if own in A[p, f]
    then
        enter r into A[q, f];
end
```

- Mono-conditional command
  - Single condition in this command

43



Multiple Conditions

- Let *p* give *q r* and *w* rights over *f*, if *p* owns *f* and *p* has *c* rights over *q*

```
command grant•readwrite•file•2(p, f, q)
    if own in A[p, f] and c in A[p, q]
    then
        enter r into A[q, f];
        enter w into A[q, f];
end
```

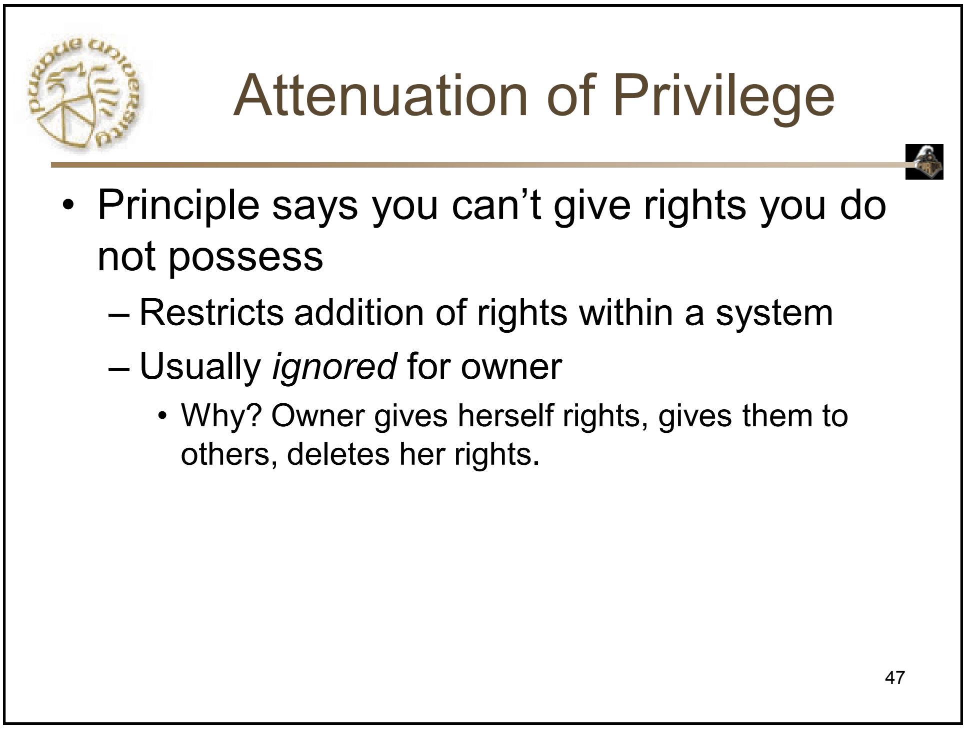
44

Copy Right

- Allows possessor to give rights to another
- Often attached to a right, so only applies to that right
  - *r* is read right that cannot be copied
  - *rc* is read right that can be copied
- Is copy flag copied when giving *r* rights?
  - Depends on model, instantiation of model

Own Right

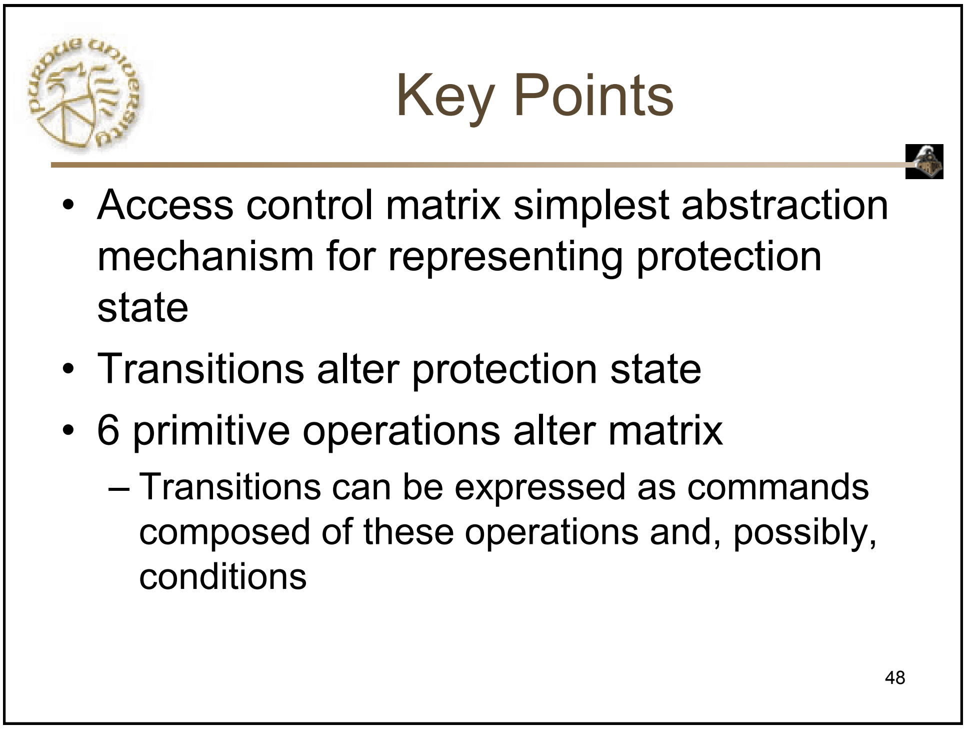
- Usually allows possessor to change entries in ACM column
  - So owner of object can add, delete rights for others
  - May depend on what system allows
    - Can’t give rights to specific (set of) users
    - Can’t pass copy flag to specific (set of) users



Attenuation of Privilege

- Principle says you can’t give rights you do not possess
  - Restricts addition of rights within a system
  - Usually *ignored* for owner
    - Why? Owner gives herself rights, gives them to others, deletes her rights.

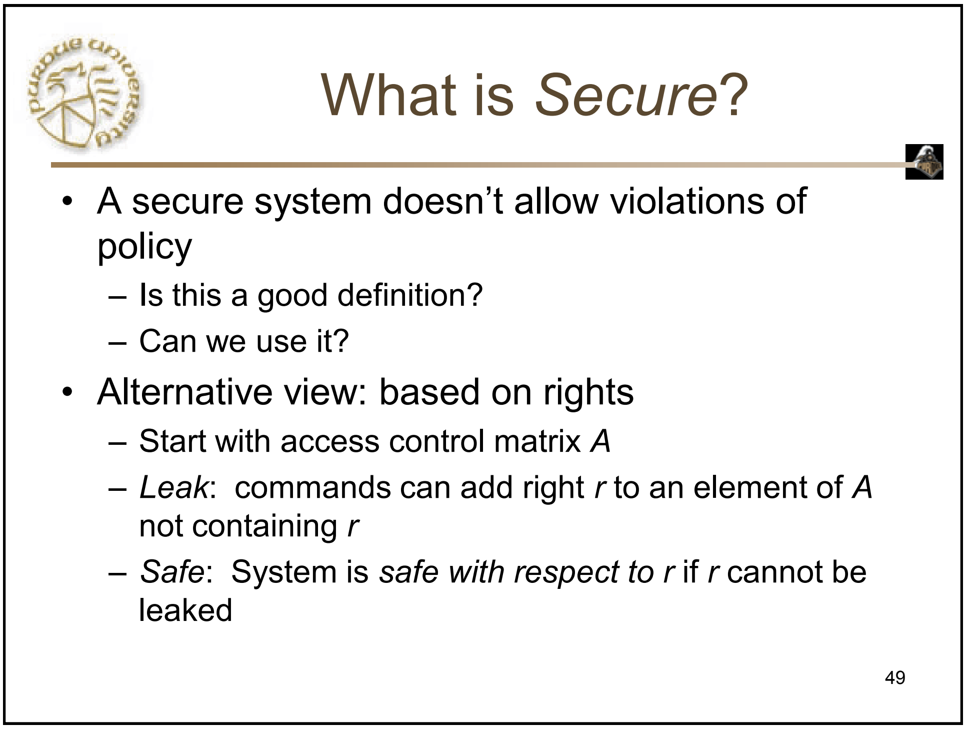
47



Key Points

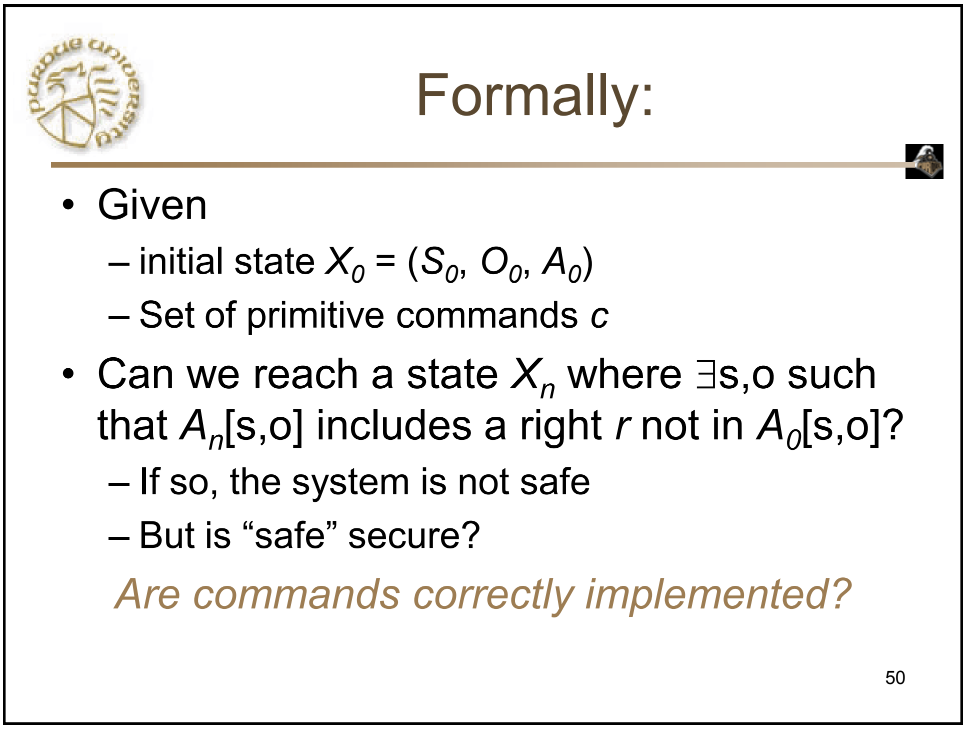
- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions

48



What is *Secure*?

- A secure system doesn't allow violations of policy
  - Is this a good definition?
  - Can we use it?
- Alternative view: based on rights
  - Start with access control matrix *A*
  - *Leak*: commands can add right *r* to an element of *A* not containing *r*
  - *Safe*: System is *safe with respect to r* if *r* cannot be leaked



Formally:

- Given
  - initial state $X_0 = (S_0, O_0, A_0)$
  - Set of primitive commands *c*
- Can we reach a state X_n where ∃s,o such that A_n[s,o] includes a right *r* not in A_0[s,o]?
  - If so, the system is not safe
  - But is "safe" secure?

Are commands correctly implemented?

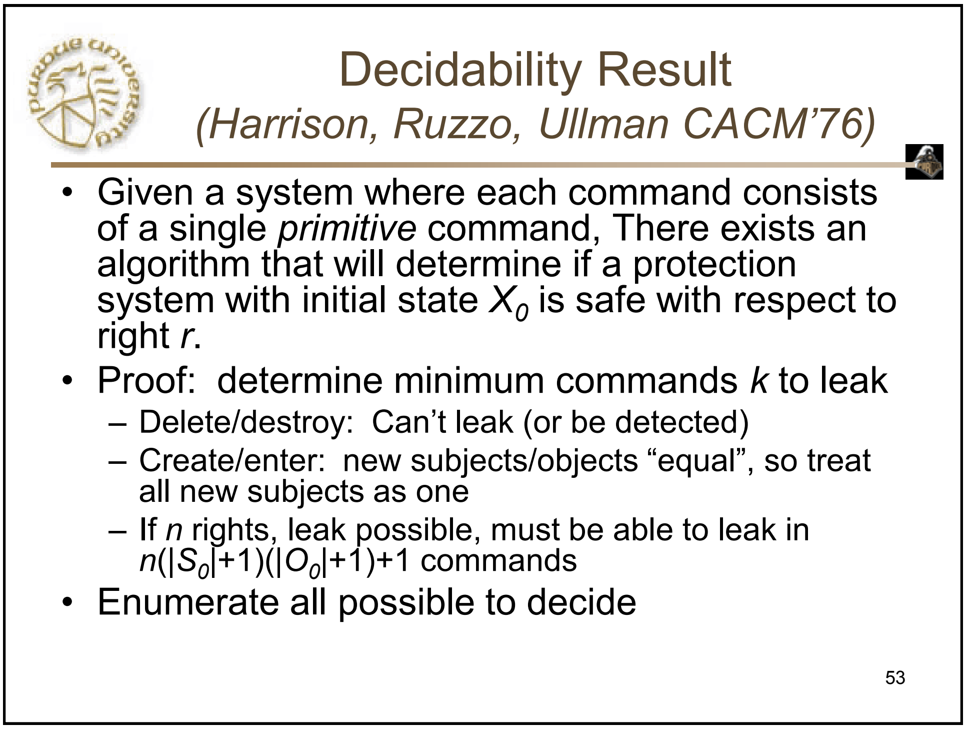
Example: Unix File System

- Access Control Matrix
  - Root has access to all files
  - Owner has access to their own files
- Safe with respect to file access right?
  - No chmod/chown command
  - Only “root” can get root privileges
  - Only user can authenticate as themselves

Is “Safe” definition useful?

Solution: Trust

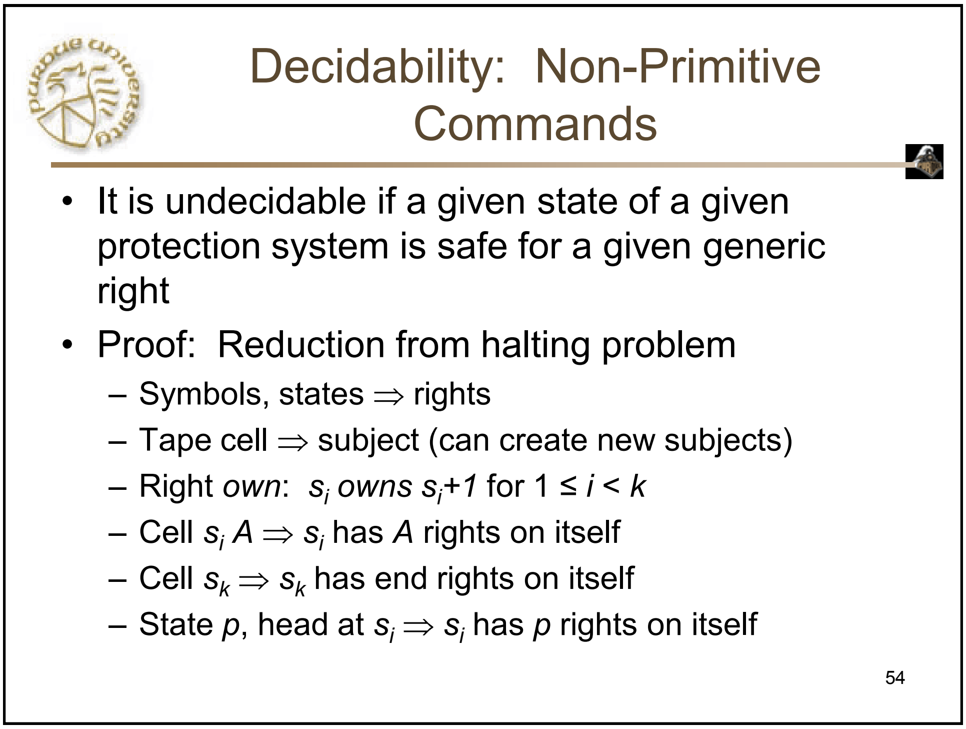
- Safety doesn’t distinguish leak from authorized transfer of rights
- Subjects authorized to receive transfer of rights deemed “trusted”
  - Eliminate trusted subjects from matrix



Decidability Result
(Harrison, Ruzzo, Ullman CACM'76)

- Given a system where each command consists of a single *primitive* command, There exists an algorithm that will determine if a protection system with initial state X_0 is safe with respect to right *r*.
- Proof: determine minimum commands *k* to leak
  - Delete/destroy: Can't leak (or be detected)
  - Create/enter: new subjects/objects "equal", so treat all new subjects as one
  - If *n* rights, leak possible, must be able to leak in $n(|S_0|+1)(|O_0|+1)+1$ commands
- Enumerate all possible to decide

53



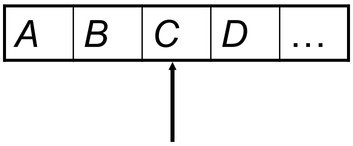
Decidability: Non-Primitive Commands

- It is undecidable if a given state of a given protection system is safe for a given generic right
- Proof: Reduction from halting problem
  - Symbols, states ⇒ rights
  - Tape cell ⇒ subject (can create new subjects)
  - Right *own*: s_i *owns* s_i+1 for $1 \le i < k$
  - Cell s_i A ⇒ s_i has A rights on itself
  - Cell s_k ⇒ s_k has end rights on itself
  - State p, head at s_i ⇒ s_i has p rights on itself

54

Example:

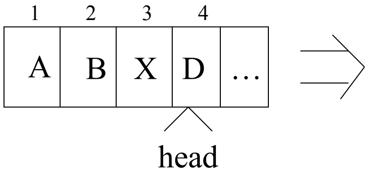
Turing Machine



Matrix

	s_1	s_2	s_3	s_4
s_1	*A*	*own*		
s_2		*B*	*own*	
s_3			*C, p*	*own*
s_4				*D, end*

Mapping



	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	

After $\delta(k, C) = (k_1, X, R)$ where k is the current state and k_1 the next state

Command Mapping

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

```
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
```

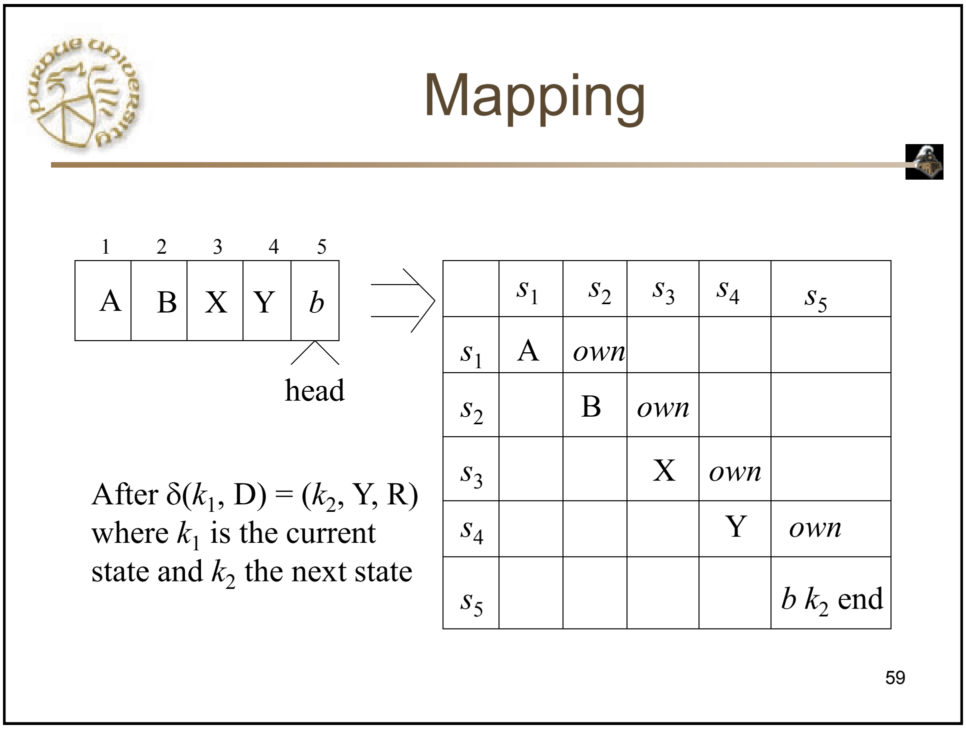
57

Commands:

- Halting problem Turing Machine: Symbols *A*, *B*; states *p, q*
- $C_{p,A}(s_i, s_{i-1})$ (move left)
  - if $own \in a[s_{i-1}, s_i]$ and $p \in a[s_i, s_i]$ and $A \in a[s_i, s_i]$
    - Delete p from $a[s_i, s_i]$, A from $a[s_i, s_i]$
    - Enter B into $a[s_i, s_i]$, q into $a[s_{i-1}, s_{i-1}]$
- Similar commands for move right, move right at end of tape
- Simulates Turing machine
  - Leaks halting state ⇒ halting state in the matrix ⇒ Halting state reached

This is undecidable!

58



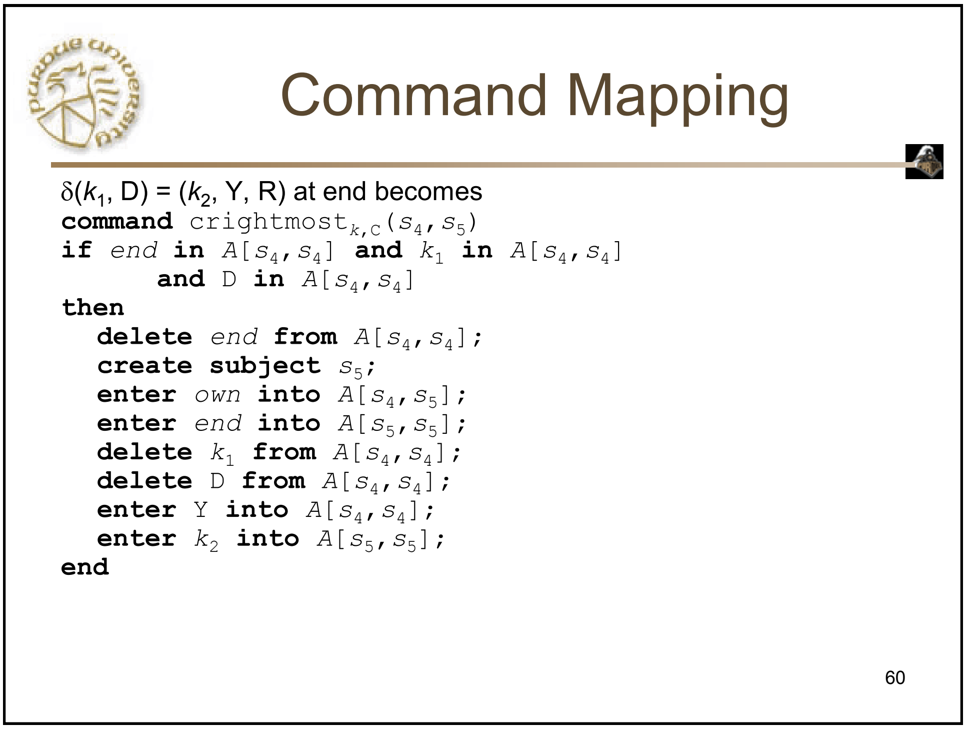
Mapping

1	2	3	4	5
A	B	X	Y	b
				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					b k_2 end

After $\delta(k_1, \text{D}) = (k_2, \text{Y}, \text{R})$ where k_1 is the current state and k_2 the next state

59



Command Mapping

$\delta(k_1, \text{D}) = (k_2, \text{Y}, \text{R})$ at end becomes

```
command crightmost$_{k,C}$($s_4$, $s_5$)
if end in A[$s_4$, $s_4$] and $k_1$ in A[$s_4$, $s_4$]
      and D in A[$s_4$, $s_4$]
then
  delete end from A[$s_4$, $s_4$];
  create subject $s_5$;
  enter own into A[$s_4$, $s_5$];
  enter end into A[$s_5$, $s_5$];
  delete $k_1$ from A[$s_4$, $s_4$];
  delete D from A[$s_4$, $s_4$];
  enter Y into A[$s_4$, $s_4$];
  enter $k_2$ into A[$s_5$, $s_5$];
end
```

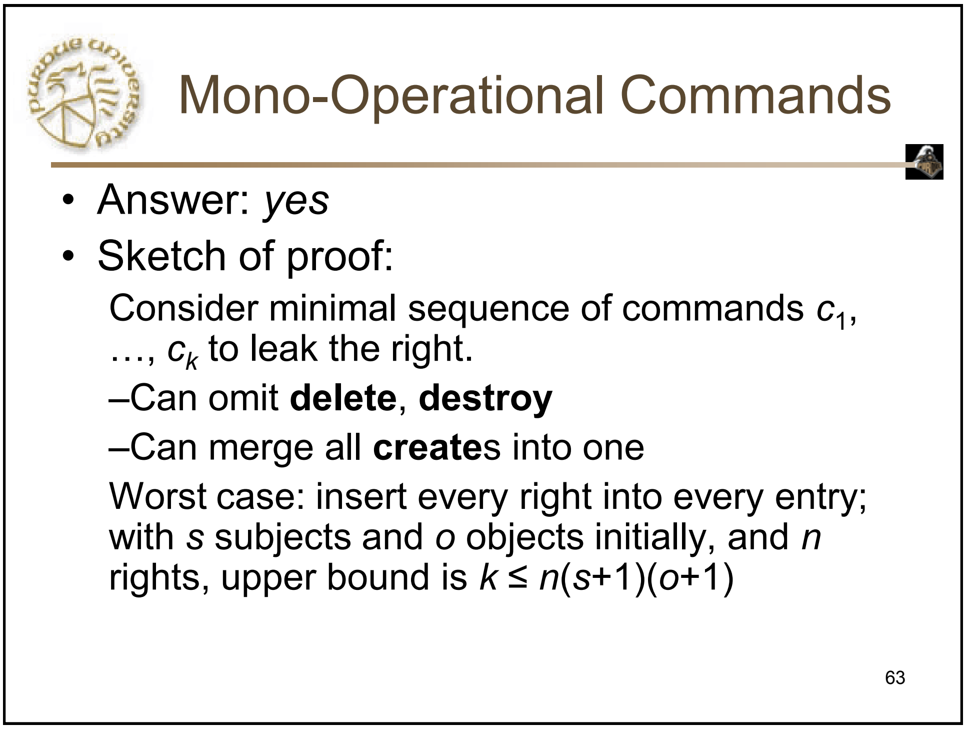
60

Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 *end* right in ACM
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command
- If TM enters state q_f, then right has leaked
- If safety question decidable, then represent TM as above and determine if q_f leaks
  - Implies halting problem decidable
- Conclusion: safety question undecidable

Other Results *(most from the same authors)*

- Set of unsafe systems recursively enumerable
- Without create primitive, safety in P-SPACE
  - Like halting problem reduction, but no unlimited tape
- Without delete/destroy, still undecidable
  - Decidable if at most one condition allowed per command
  - Still holds if delete allowed



Mono-Operational Commands

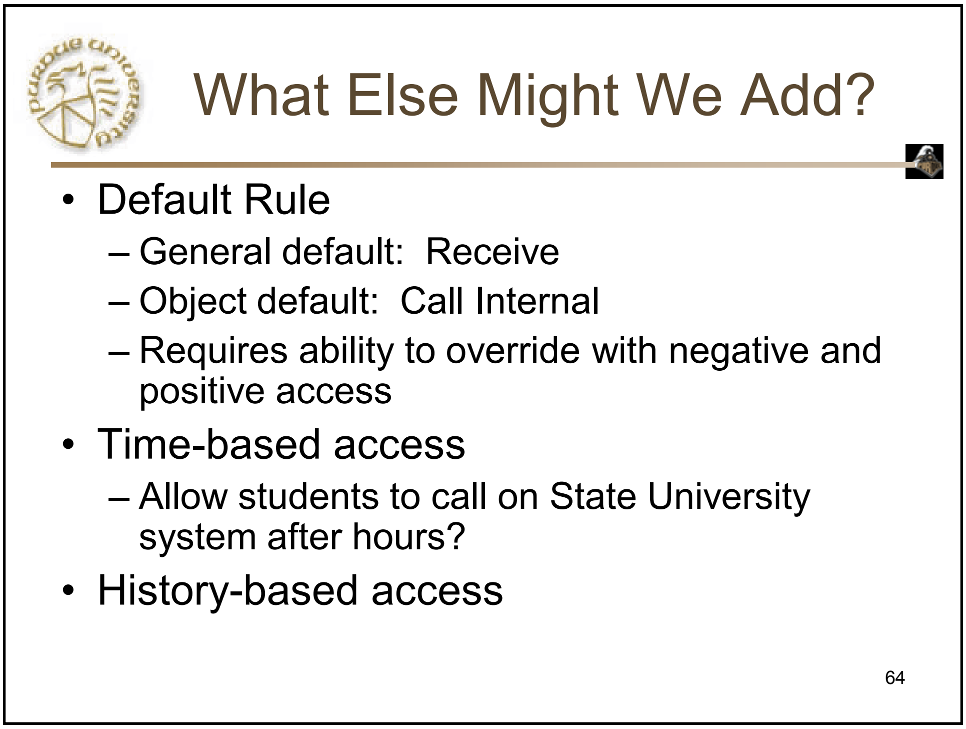
- Answer: *yes*
- Sketch of proof:

  Consider minimal sequence of commands c_1, …, c_k to leak the right.

  –Can omit **delete**, **destroy**

  –Can merge all **create**s into one

  Worst case: insert every right into every entry; with *s* subjects and *o* objects initially, and *n* rights, upper bound is $k \le n(s+1)(o+1)$



What Else Might We Add?

- Default Rule
  - General default: Receive
  - Object default: Call Internal
  - Requires ability to override with negative and positive access
- Time-based access
  - Allow students to call on State University system after hours?
- History-based access

Access Control by History

- Example: Statistical Database
  - Allows queries for general statistics
  - But not individual values
- Valid queries: Statistics on 20+ individuals
  - Total salary of all Deans
  - Salary of Computer Science Professors
- See a problem coming?
  - Salary of CS Professors who aren't Deans

Solution: Query Set Overlap Control (Dobkin, Jones & Lipton '79)

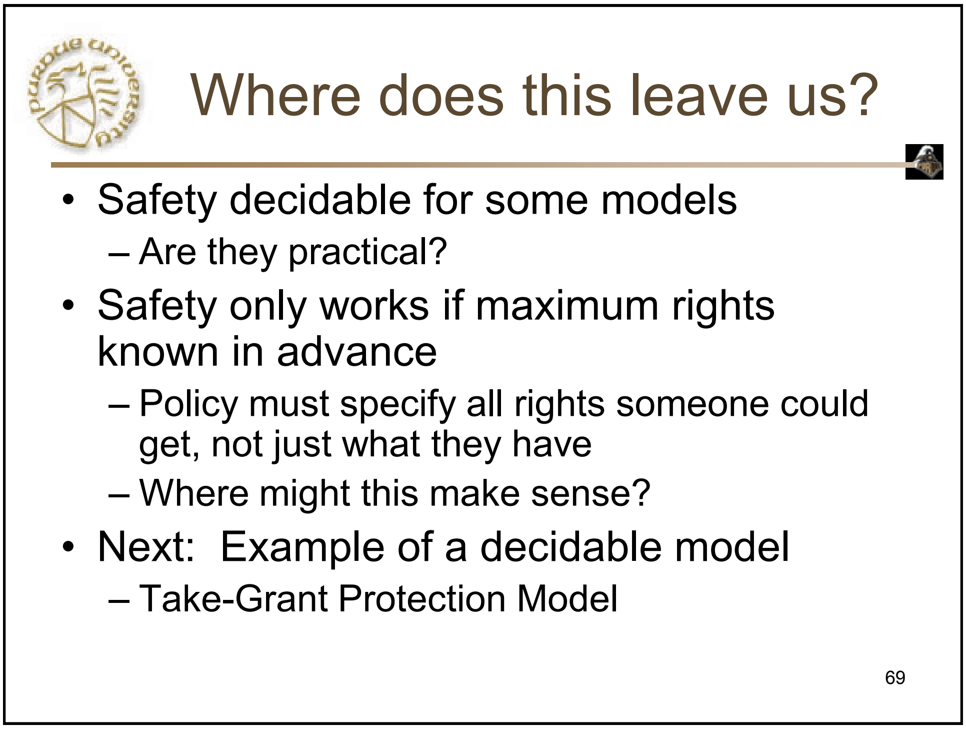
- Query valid if intersection of query coverage and each previous query < *r*
- Given *K* minimum query size, r overlap:
  - Need *1 + (K-1)/r* queries to compromise
- Can represent as access control matrix
  - Subjects: entities issuing queries
  - Objects: *Powerset* of records
  - $O_s(i)$: objects referenced by *s* in queries *1..i*
  - *A*[*s,o*] = read iff $\forall_{q \in \cup_s -} \quad \cap \quad <$

Next

- Optional reading: Dobkin, Jones, and Lipton (TODS 4(1), see course web site)
- Basic theorems on protection states
  - Decidability of safety of a state with respect to a right
- More Protection Models

Protection Study: Your Homework

- What does it take to make sure your homework is secure?
  - Let’s assume a Unix system (mentor.ics)
  - Issues?
- *Participation Expected!*



Where does this leave us?

- Safety decidable for some models
  - Are they practical?
- Safety only works if maximum rights known in advance
  - Policy must specify all rights someone could get, not just what they have
  - Where might this make sense?
- Next: Example of a decidable model
  - Take-Grant Protection Model

69