Introduction to Computer Security

August 28, 2003

Course Objective

- The objective of the course is to cover the fundamental issues of information system security and assurance.

Course Material

- Textbook
  - Computer Security: Art and Science, Matt Bishop, Addison- Wesley, 2003
    - Will follow the book mostly
    - Will be supplemented by other material (references and papers)
    - Errata URL: http://nob.cs.ucdavis.edu/~bishop/
- Other References
  - Security in Computing, 2nd Edition, Charles P. Pfleeger, Prentice Hall
  - Security Engineering: A Guide to Building Dependable Distributed Systems, Ross Anderson, Wiley, John & Sons, Incorporated, 2001
  - Building Secure Software: How to avoid the Security Problems the Right Way, John Viega, Gary McGraw, Addison-Wesley, 2002
- Papers
  - List will be provided as supplemental readings and review assignments

Prerequisites

- Assumes the following background
  - Good programming experience
  - Working knowledge of
    - Operating systems, algorithms and data structures, database systems, and networks
  - Mathematics
    - Undergraduate mathematics
    - Some knowledge of mathematical logic
- Not sure? SEE ME

Course Outline

- Security Basics (1-8)
  - General overview and definitions
  - Security models and policy issues
- Basic Cryptography and Network security (9-12, 26)
  - Introduction to cryptography and classical cryptosystem
  - Authentication protocols and Key Management
- Systems Design Issues and Information assurance (13-21, 24, ??)
  - Design principles
  - Security Mechanisms
  - Auditing Systems
  - Risk analysis
  - System verification and evaluation
- Intrusion Detection and Response (23, 25, ??)
  - Attack Classification and Vulnerability Analysis
  - Detection, Containment and Response/Recovery
- Miscellaneous Issues (22, ??)
  - Malicious code, Mobile code
  - Digital Rights Management, Forensics
  - Emerging issues: E/M-commerce security, Multidomain Security Issues etc.

Grading

- Lab + Homework/Quiz/Paper review 30%
- Midterm 20%
- Paper/Project 15%
  - List of suggested topics will be posted;
  - Encouraged to think of a project/topic of your interest
- Comprehensive Final 35%

Contact

- James Joshi
- 721, IS Building
- Phone: 412-624-9982
- E-mail: jjoshi@mail.sis.pitt.edu
- Web: www2.sis.pitt.edu/~jjoshi/INFSCI2935
- Office Hours:
  - Fridays: 2.00 – 4.00 p.m.
  - By appointments
- GSA: will be announced later

Course Policies

- Your work MUST be your own
  - No copying from web or other books without understanding the material
  - Zero tolerance for cheating
  - You get an F for the course if you cheat in anything however small – NO DISCUSSION
- Homework
  - There will be penalty for late assignments (15% each day)
  - Ensure clarity in your answers – no credit will be given for vague answers
  - Homework is primarily the GSA's responsibility
  - Solutions will be posted in the library
- Check webpage for everything!
  - You are responsible for checking the webpage for updates

Security Assured Information Systems Track (SAIS)

- INFSCI 2935 will likely be TEL2810
- INFSCI 2935 is the foundation course for the SAIS track
- SAIS Courses
  - Prof. Krishnamurthy TELCOM 2820 – Cryptography
  - TELCOM 2821 – Network Security(??)
- Several interesting electives (??)
  - TELCOM 2825: Information System and Infrastructure Protection
    - Dr. Tipper – Fall 2003

SAIS Track Core (12 credits)	SAIS Track Electives (3 credits)
TEL-2810 Intro To Security	TEL-2825 Infrs. Protection
TEL-2820 Cryptography	IS-2771 Security in E-Commerce
TEL-2821 Network Security	TEL-2813 Security Management
TEL-2830 Capstone Course in Security	TEL-2829 Adv. Cryptography

Introduction to Security

Overview of Computer Security

Information Systems Security

- Deals with
  - Security of (end) systems
    - Examples: Operating system, files in a host, records, databases, accounting information, logs, etc.
  - Security of information in transit over a network
    - Examples: e-commerce transactions, online banking, confidential e-mails, file transfers, record transfers, authorization messages, etc.

"Using encryption on the internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench" –

Gene Spafford

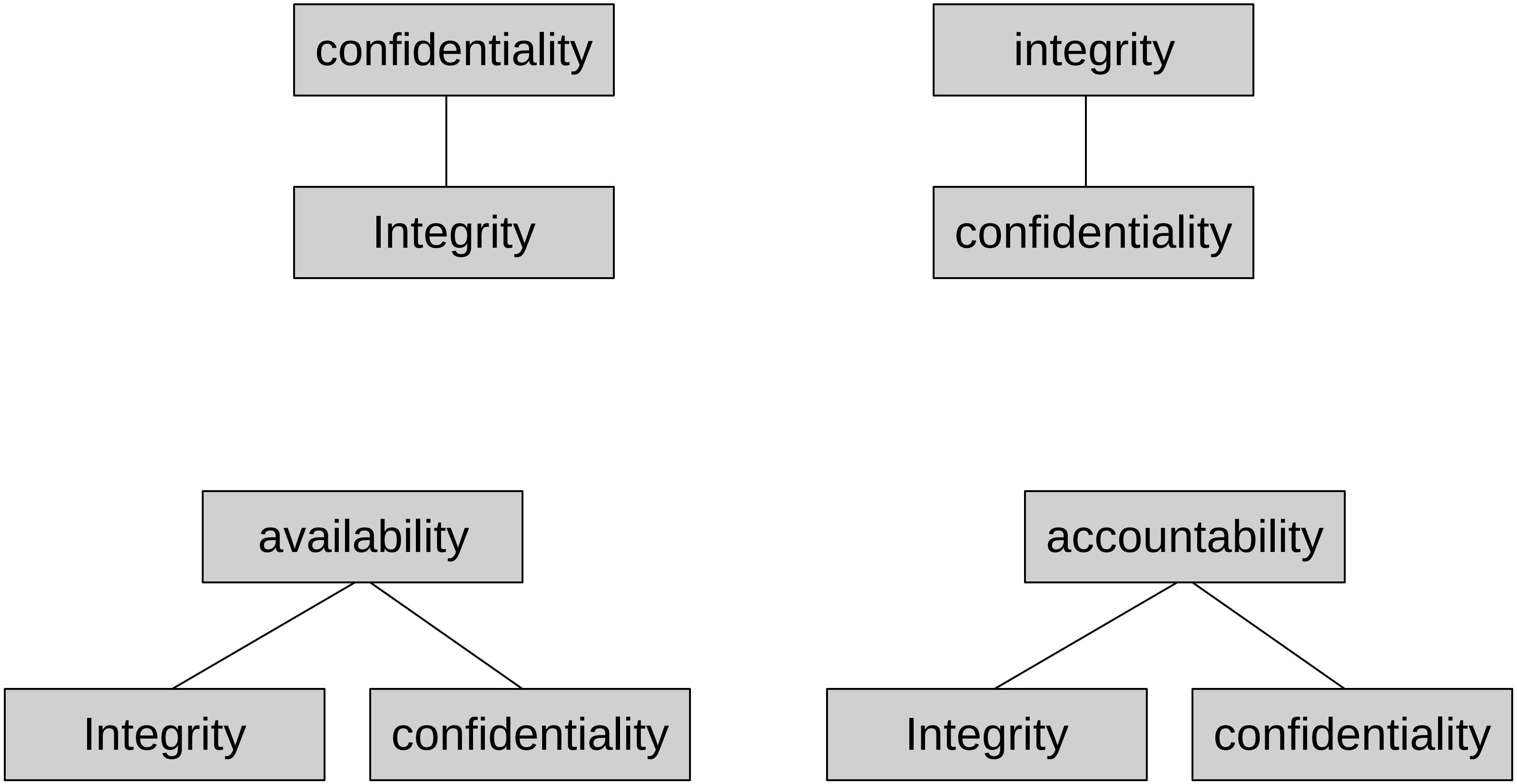
Basic Components of Security

- Confidentiality
  - Keeping data and resources secret or hidden
- Integrity
  - Ensuring authorized modifications;
  - Includes correctness and trustworthiness
  - May refer to
    - Data integrity
    - Origin integrity
- Availability
  - Ensuring authorized access to data and resources when desired

(Additional from NIST)

- Accountability
  - Ensuring that an entity's action is traceable uniquely to that entity
- Security assurance
  - Assurance that all four objectives are met

Interdependencies



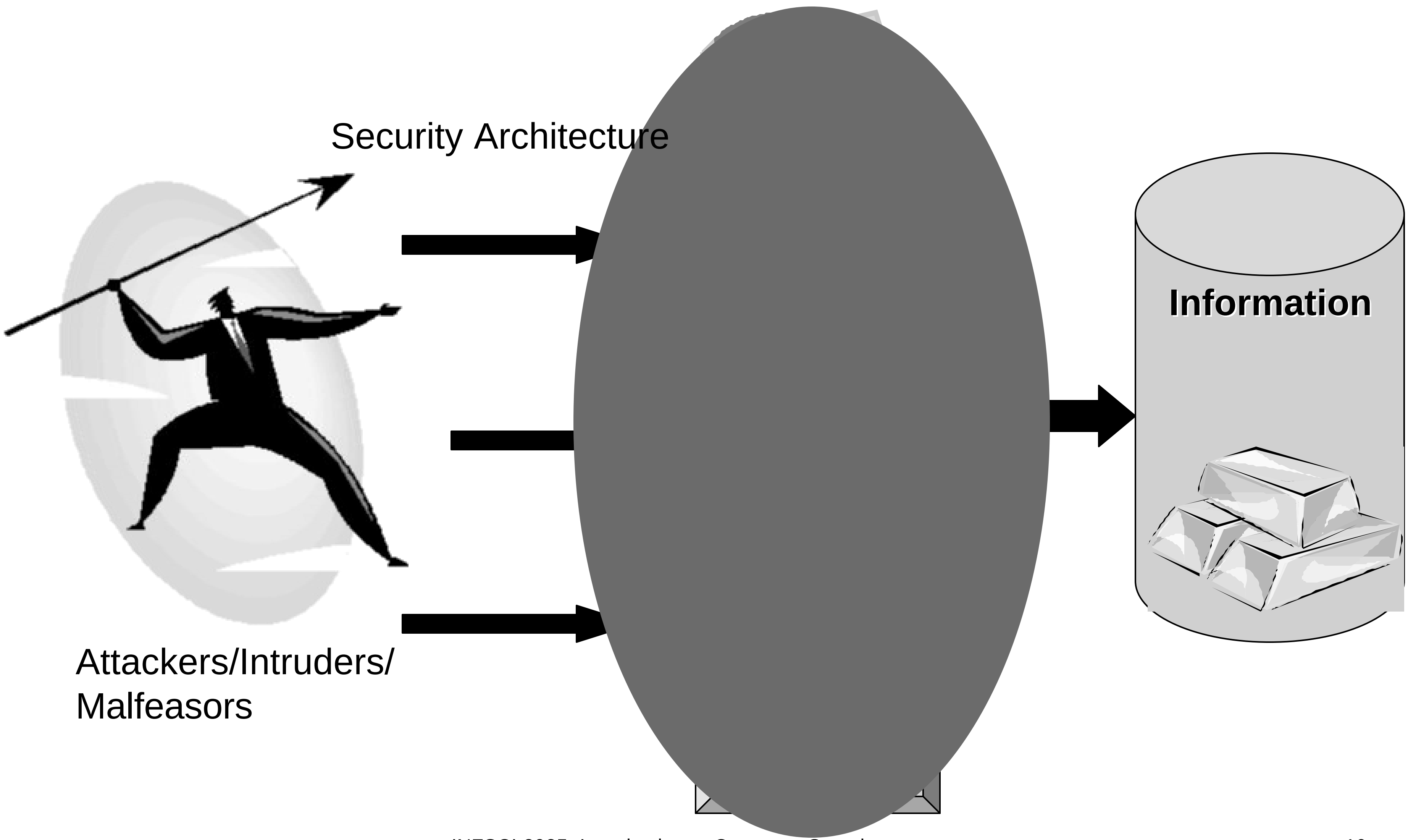
Information Security 20 years back

- Physical security
  - Information was primarily on paper
  - Lock and key
  - Safe transmission
- Administrative security
  - Control access to materials
  - Personnel screening
  - Auditing

Information security today

- Emergence of the Internet and distributed systems
  - Increasing system complexity
- Digital information needs to be kept secure
  - Competitive advantage
  - Protection of assets
  - Liability and responsibility
- Financial losses
  - The FBI estimates that an insider attack results in an average loss of $2.8 million
  - There are reports that the annual financial loss due to information security breaches is between 5 and 45 billion dollars
- National defense
  - Protection of critical infrastructures:
    - Power Grid;
    - Air transportation
  - Interlinked government agencies
    - Grade F for most of the agencies
    - Severe concerns regarding security management and access control measures (GAO report 2003)

Terminology



Attack Vs Threat

- A threat is a “potential” violation of security
  - The violation need not actually occur
  - The fact that the violation *might* occur makes it a threat
  - It is important to guard against threats and be prepared for the actual violation
- The actual violation of security is called an attack

Common security attacks

- Interruption, delay, denial of receipt or denial of service
  - System assets or information become unavailable or are rendered unavailable
- Interception or snooping
  - Unauthorized party gains access to information by browsing through files or reading communications
- Modification or alteration
  - Unauthorized party changes information in transit or information stored for subsequent access
- Fabrication, masquerade, or spoofing
  - Spurious information is inserted into the system or network by making it appear as if it is from a legitimate entity
  - Not to be confused with delegation
- Repudiation of origin
  - False denial that an entity created something

Classes of Threats

- Disclosure: *unauthorized access to information*
  - Snooping
- Deception: *acceptance of false data*
  - Modification, masquerading/spoofing, repudiation of origin, denial of receipt
- Disruption: *interruption/prevention of correct operation*
  - Modification
- Usurpation: *unauthorized control of a system component*
  - Modification, masquerading/spoofing, delay, denial of service

Goals of Security

- Prevention
  - To prevent someone from violating a security policy
- Detection
  - To detect activities in violation of a security policy
  - Verify the efficacy of the prevention mechanism
- Recovery
  - Stop policy violations (attacks)
  - Assess and repair damage
  - Ensure availability in presence of an ongoing attack
  - Fix vulnerabilities for preventing future attack
  - Retaliation against the attacker

Policies and Mechanisms

- A security policy states 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

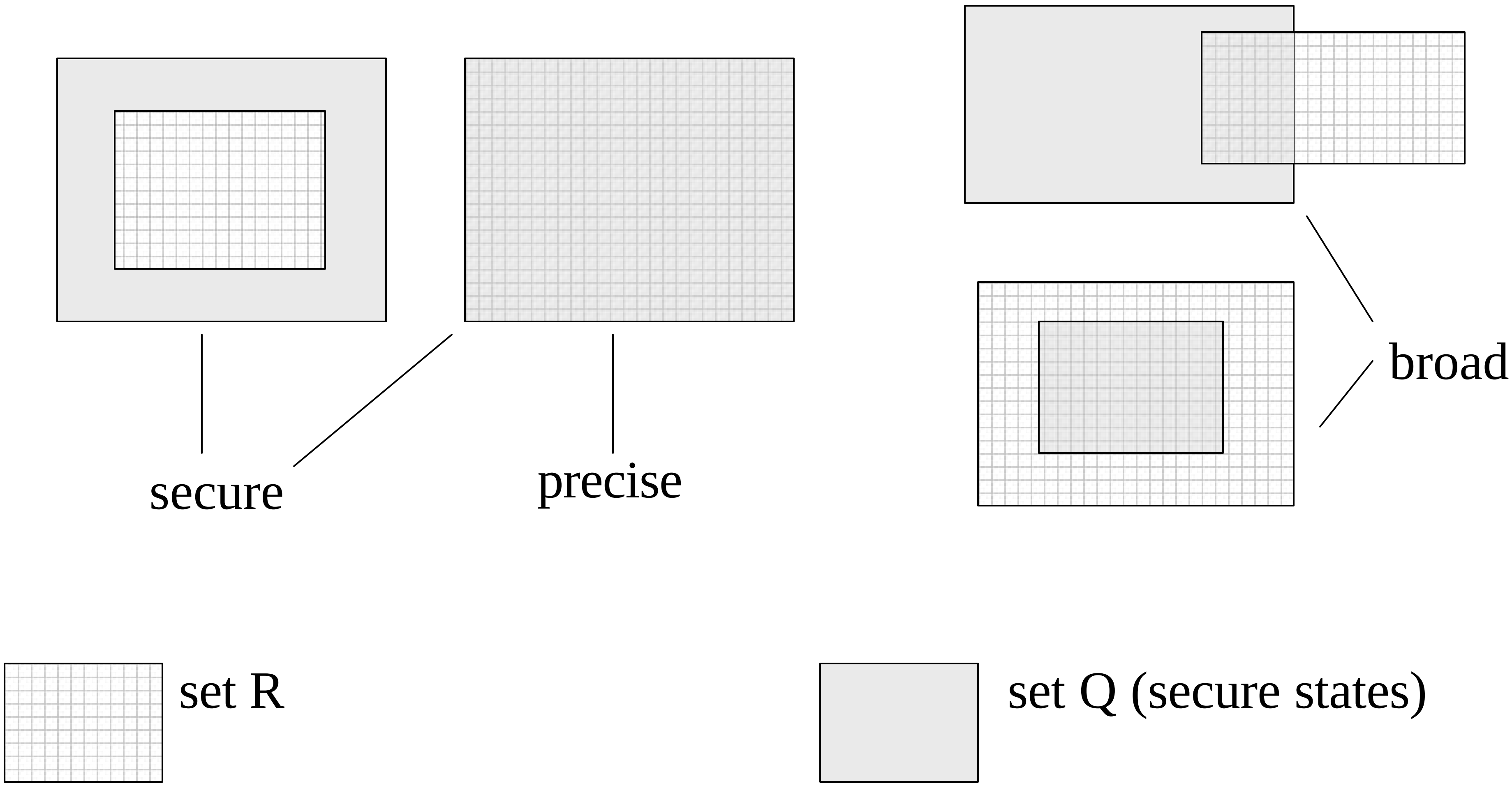
Assumptions and Trust

- Policies and mechanisms have implicit assumptions
- Assumptions regarding policies
    - Unambiguously partition system states into “secure” and “nonsecure” states
    - Correctly capture security requirements
- Mechanisms
    - Assumed to enforce policy; i.e., ensure that the system does not enter “nonsecure” state
    - Support mechanisms work correctly

Types of Mechanisms

- Let P be the set of all the reachable states
- Let Q be a set of secure states identified by a policy: $Q \subseteq P$
- Let the set of states that an enforcement mechanism restricts a system to be R
- The enforcement mechanism is
  - Secure if $R \subseteq Q$
  - Precise if $R = Q$
  - Broad if R-Q is non-empty

Types of Mechanisms



Information Assurance

- *Information Assurance Advisory Council (IAAC)*:

  "Operations undertaken to protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality and non-repudiation"

- National Institute of Standards Technology

  "Assurance is the basis for confidence that the security measures, both technical and operational, work as intended to protect the system and the information it processes"

Assurance

- Assurance is to indicate “how much” to trust a system and is achieved by ensuring that
  - The required functionality is present and correctly implemented
  - There is sufficient protection against unintentional errors
  - There is sufficient resistance to intentional penetration or by-pass
- Basis for determining this aspect of trust
  - Specification
    - Requirements analysis
    - Statement of desired functionality
  - Design
    - Translate specification into components that satisfy the specification
  - Implementation
    - Programs/systems that satisfy a design

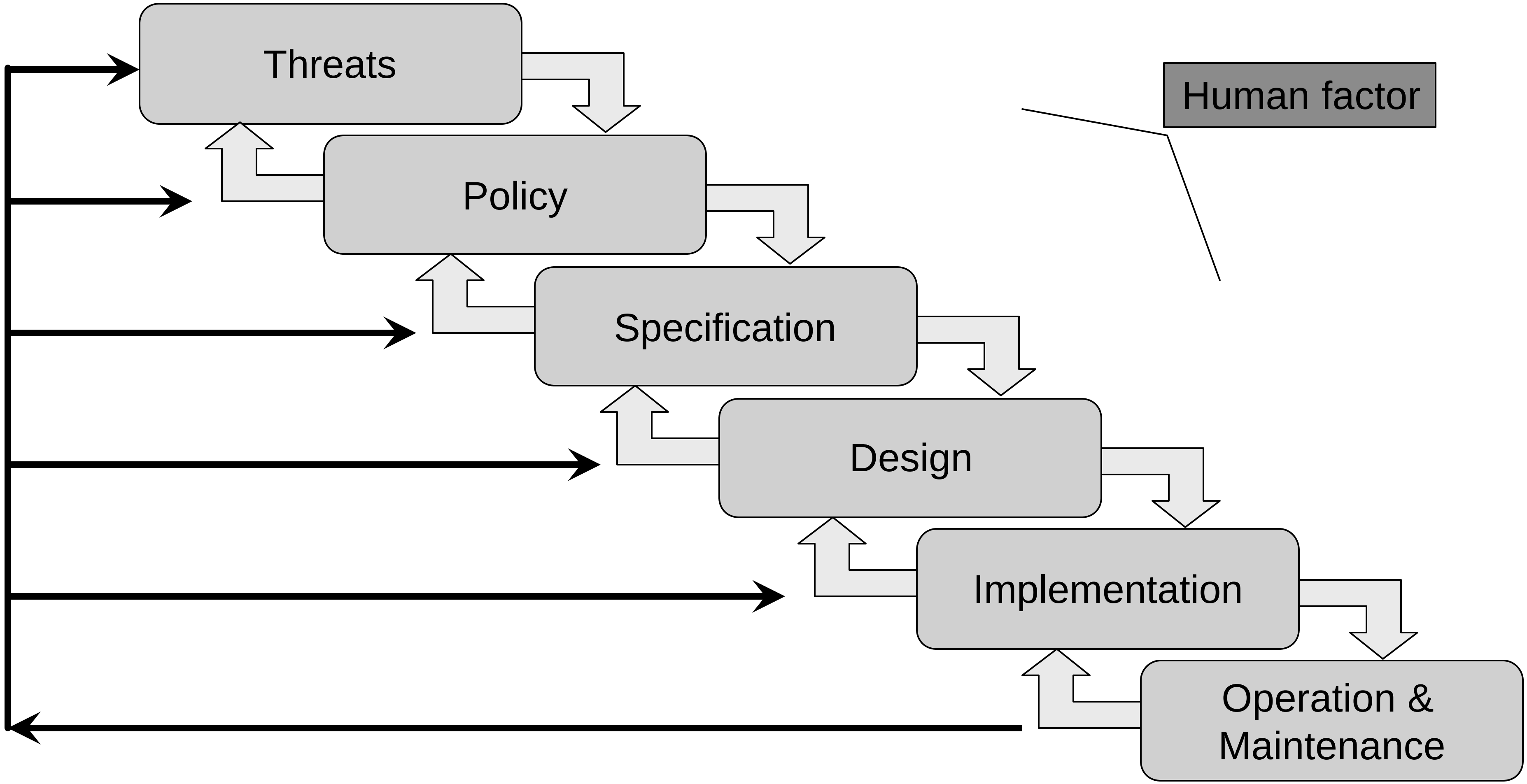
Operational Issues

- Cost-Benefit Analysis
  - Benefits vs. total cost
  - Is it cheaper to prevent or recover?
- Risk Analysis
  - Should we protect something?
  - How much should we protect this thing?
  - Risk depends on environment and change with time
- Laws and Customs
  - Are desired security measures illegal?
  - Will people do them?
  - Affects availability and use of technology

Human Issues

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

Tying all together: The Life Cycle



Protection System

- State of a system
  - Current values of
    - memory locations, registers, secondary storage, etc.
    - other system components
- Protection state (P)
  - A system state that is considered secure
- A protection system
  - Describes the conditions under which a system is secure (in a protection state)
  - Consists of two parts:
    - A set of generic rights
    - A set of commands
- State transition
  - Occurs when an operation (command) is carried out

Protection System

- Subject (S: set of all subjects)
  - Active entities that carry out an action/operation on other entities; Eg.: users, processes, agents, etc.
- Object (O: set of all objects)
  - Eg.:Processes, files, devices
- Right
  - An action/operation that a subject is allowed/disallowed on objects

Access Control Matrix Model

- Access control matrix
  - Describes the protection state of a system.
  - Characterizes the rights of each subject
  - Elements indicate the access rights that subjects have on objects
- ACM is an abstract model
  - Rights may vary depending on the object involved
- ACM is implemented primarily in two ways
  - Capabilities (rows)
  - Access control lists (columns)

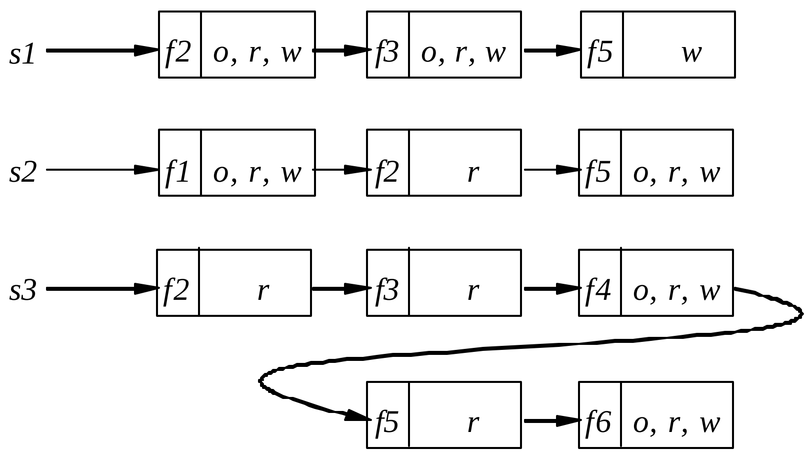
Access Control Matrix

o: own
r: read
w:write

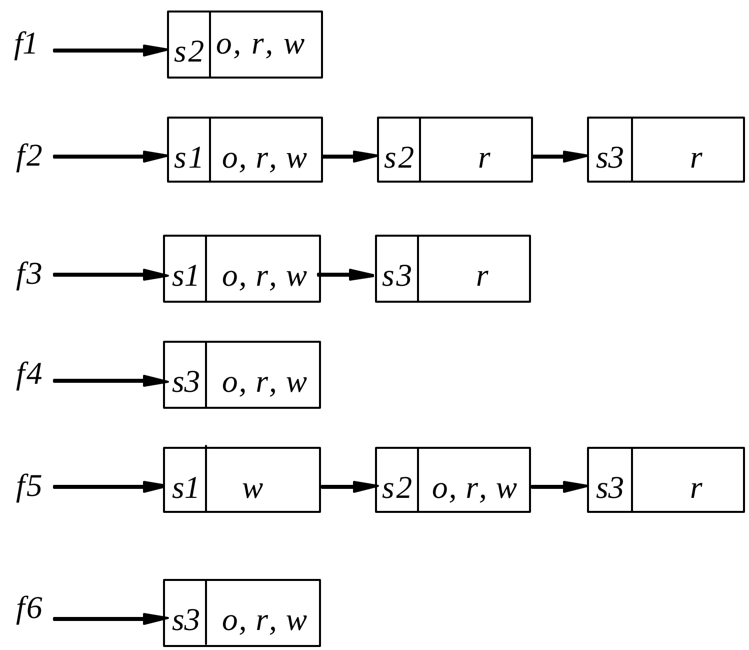
	f1	f2	f3	f4	f5	f6
s1		o, r, w	o, r, w		w	
s2	o, r, w	r			o, r, w	
s3		r	r	o, r, w	r	o, r, w

Access Matrix

Capabilities



Access Control List



Access Control Matrix

Hostnames	*Telegraph*	*Nob*	*Toadflax*
Telegraph	*own*	*ftp*	*ftp*
Nob		*ftp, nsf, mail, own*	*ftp, nfs, mail*
Toadflax		*ftp, mail*	*ftp, nsf, mail, own*

	Counter	*Inc_ctr*	*Dcr_ctr*	*Manager*
Inc_ctr	+			
Dcr_ctr	-			
manager		*Call*	*Call*	*Call*

Access Controlled by History

- Statistical databases need to
  - answer queries on groups
  - prevent revelation of individual records
- Query-set-overlap control
  - Prevent an attacker to obtain individual piece of information using a set of queries C
  - A parameter *r* is used to determine if a query should be answered

Name	Position	Age	Salary
Celia	Teacher	45	40K
Heidi	Aide	20	20K
Holly	Principal	37	60K
Leonard	Teacher	50	50K
Matt	Teacher	33	50K

Access Controlled by History

- Query 1:
  - sum_salary(position = teacher)
  - Answer: 140K
- Query 2:
  - sum_salary(age > 40 & position = teacher)
  - Should not be answered as Matt's salary can be deduced
- Can be represented as an ACM

Name	Position	Age	Salary
Celia	Teacher	45	40K
Leonard	Teacher	50	50K
Matt	Teacher	33	50K

Name	Position	Age	Salary
Celia	Teacher	45	40K
Leonard	Teacher	50	50K

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

- Query valid if intersection of query coverage and each previous query < *r*
- 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 $\forall_{q \in O_s(i-1)} |q \cap o| < r$

ACM of Database Queries

1. O_1 = {Celia, Leonard, Matt} so
2. A[asker, Celia] = Celia) = {read}
3. A[asker, Leonard] = Leonard) = {read}
4. A[asker, Matt] = *f*(Matt) = {read}
5. and query can be answered

But Query 2

1. O_2 = {Celia, Leonard} but $| O_2 \cap O_1 | = 2$ so
2. A[asker, Celia] = f(Celia) = $\varnothing$
3. A[asker, Leonard] = f(Leonard) = $\varnothing$
4. and query cannot be answered

State Transitions

- Let initial state $X_0 = (S_0, O_0, A_0)$
- Notation
  - $X_i + \tau_{i+1}\ X_{i+1}$: upon transition τ_{i+1}, the system moves from state X_i to X_{i+1}
  - $X +^* Y$: the system moves from state X to Y after a set of transitions
  - $X_i +\ c_{i+1}\ (p_{i+1,1}, p_{i+1,2}, \ldots, p_{i+1,\mathrm{m}})\ X_{i+1}$: state transition upon a command
- For every command there is a sequence of state transition operations

Primitive commands (HRU)

Create subject s	Creates new row, column in ACM;
Create object o	Creates new column in ACM
Enter r into $a[s, o]$	Adds *r* right for subject *s* over object *o*
Delete r from $a[s, o]$	Removes *r* right from subject *s* over object *o*
Destroy subject s	Deletes row, column from ACM;
Destroy object o	Deletes column from ACM

Create Subject

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

Create Object

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

Add Right

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

Delete Right

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

Destroy Subject

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

Destroy Object

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

System commands using primitive operations

- 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*
  Enter *own* into *a*[*p*,*f*]
  Enter *r* into *a*[*p*,*f*]
  Enter *w* into *a*[*p*,*f*]
  End

- Defined commands can be used to update ACM

  Command *make_owner*(*p*, *f*)
  Enter *own* into *a*[*p*,*f*]
  End

- Mono-operational: the command invokes only one primitive

Conditional Commands

- Mono-operational + mono-conditional

  Command *grant_read_file*(*p, f, q*)
  If *own* in *a*[*p,f*]
  Then
  Enter *r* into *a*[*q,f*]
  End

- Mono-operational + biconditional

  Command *grant_read_file*(*p, f, q*)
  If *r* in *a*[*p,f*] and *c* in *a*[*p,f*]
  Then
  Enter *r* into *a*[*q,f*]
  End

- Why not “OR”??

Attenuation of privilege

- Principle of attenuation
  - A subject may not give rights that it does not posses to others
- Copy
  - Augments existing rights
  - 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 Also called the *grant* right
- Own
  - Allows adding or deleting rights, and granting rights to others
  - Creator has the *own* right
  - Subjects may be granted *own* right
  - Owner may give rights that he does not have to others on the objects he owns (chown command)
    - Example: John owns file *f* but does not have *read* permission over it. John can grant *read* right on *f* to Matt.

Fundamental questions

- How can we determine that a system is secure?
  - Need to define what we mean by a system being “secure”
- Is there a generic algorithm that allows us to determine whether a computer system is secure?

What is a secure system?

- A simple definition
  - A secure system doesn't allow violations of a security policy
- Alternative view: based on distribution of rights to the subjects
  - Leakage of rights: (unsafe with res
    - Assume that *A* representing a secure state does not contain a right r in any element of A.
    - A right r is said to be leaked, if a sequence of operations/commands adds *r* to an element of *A,* which not containing *r*
- Safety of a system with initial protection state X_o
  - Safe with respect to r: System is *safe with respect to r* if *r* can never be leaked
  - Else it is called unsafe with respect to right r.

Safety Problem: *formally*

- Given
  - initial state $X_0 = (S_0, O_0, A_0)$
  - Set of primitive commands c
  - r is not in $A_0[s, o]$
- Can we reach a state X_n where
  - $\exists s,o$ 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?

Decidability Results *(Harrison, Ruzzo, Ullman)*

- Given a system where each command consists of a single *primitive* command (mono-operational), there exists an algorithm that will determine if a protection system with initial state X_0 is safe with respect to right *r*.
- It is undecidable if a given state of a given protection system is safe for a given generic right
- For proof – need to know Turing machines and halting problem

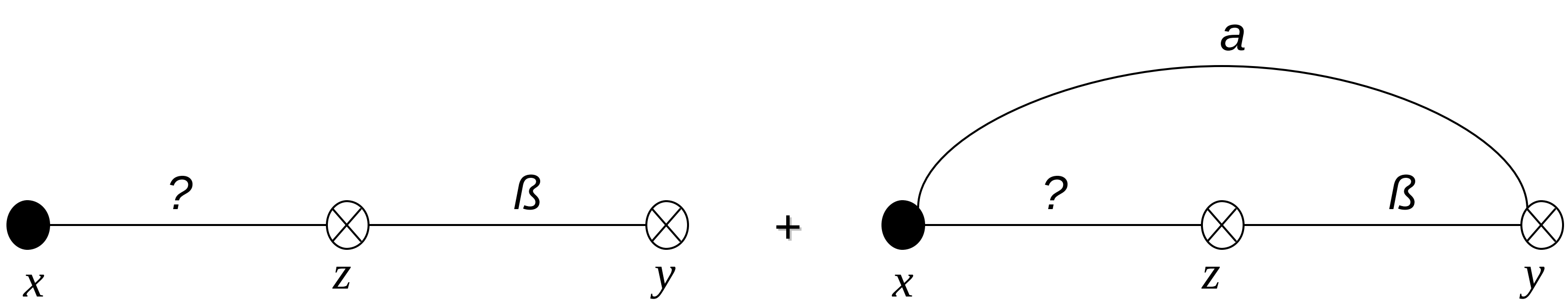
What is the implication?

- 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

Take-Grant Protection Model

- System is represented as a directed graph
  - Subject: ● Either: ⊗
  - Object: ○
  - Labeled edge indicate the rights that the source object has on the destination object
- Four graph rewriting rules ("de jure", "by law", "by rights")
  - The graph changes as the protection state changes according to

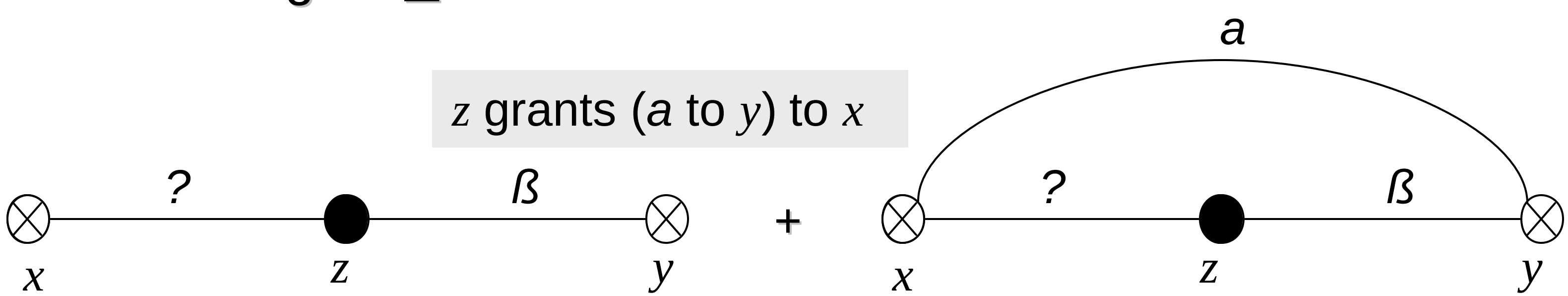
1. Take rule: if $t \in ?$, the take rule produces another graph with a transitive edge $a \subseteq ß$ added.



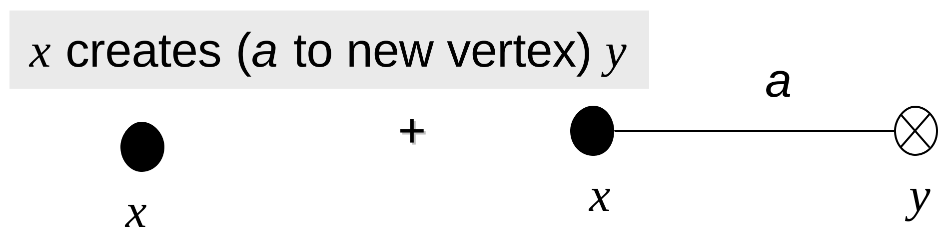
x takes (*a* to *y*) from *z*

Take-Grant Protection Model

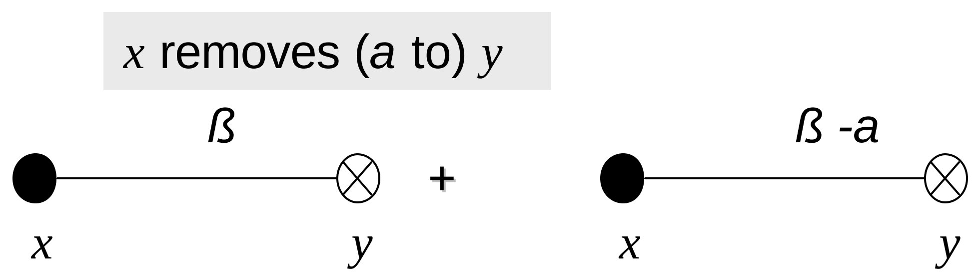
2. Grant rule: if *g* ∈ *?*, the take rule produces another graph with a transitive edge *a* ⊆ *ß added.*



3. Create rule:



4. Remove rule:

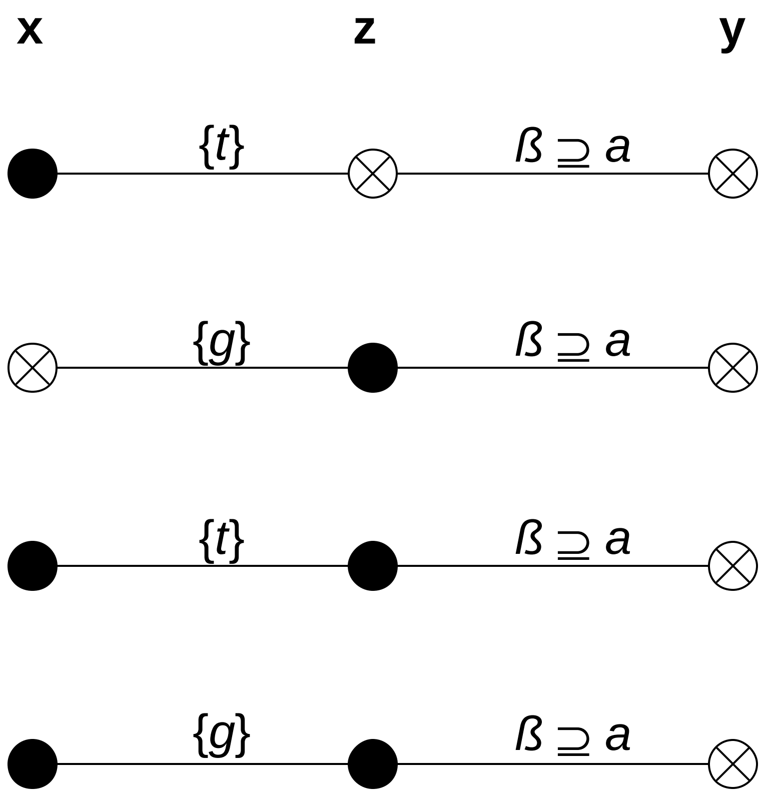


Take-Grant Protection Model: Sharing

- Given G_0, can vertex **x** obtain a rights over **y**?
  - Can_share(a,x, y,G_0) is true iff
    - $G_0 \vdash^* G_n$ using the four rules, &
    - There is an a edge from x to y in G_n
- *tg-path*: $\mathbf{v}_0,\ldots,\mathbf{v}_n$ with t or g edge between any pair of vertices $\mathbf{v}_i$, $\mathbf{v}_{i+1}$
  - Vertices *tg-connected* if *tg-path* between them
- Theorem: Any two subjects with *tg-path* of length 1 can share rights

Any two subjects with *tg-path* of length 1 can share rights

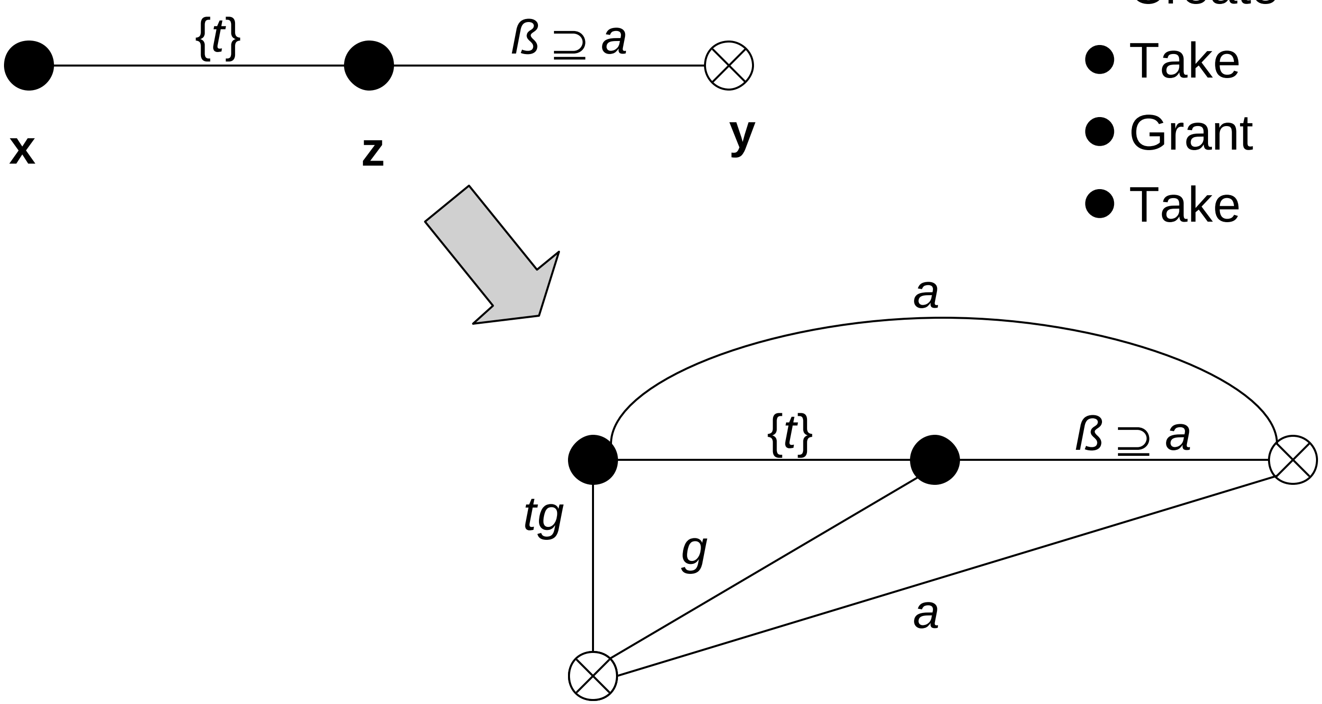
Can_share(a, ***x***, ***y***, G_0)



- Four possible length 1 *tg*-paths
  1. Take rule
  2. Grant rule
  3. Lemma 3.1
  4. Lemma 3.2

Any two subjects with *tg-path* of length 1 can share rights

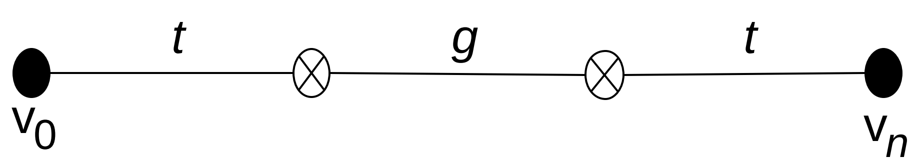
Can_share(a, ***x***, ***y***,G_0)



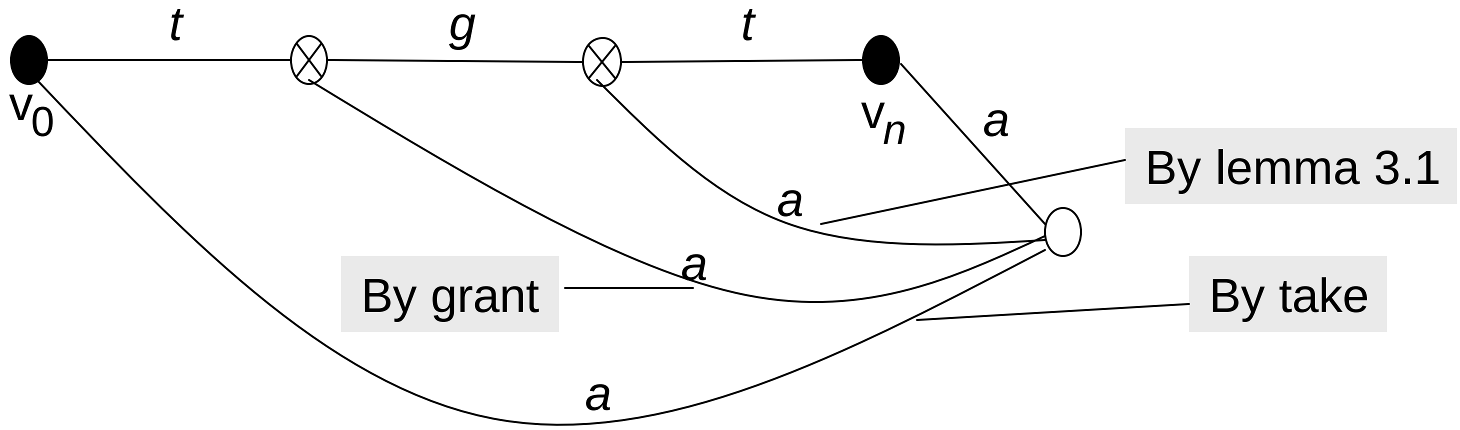
- Lemma 3.1
  - Sequence:
    - Create
    - Take
    - Grant
    - Take

Other definitions

- Island: Maximal *tg*-connected subject-only subgraph
  - Can_share all rights in island
  - Proof: Induction from previous theorem
- Bridge: *tg*-path between subjects v_0 and v_n with edges of the following form:
  - $t_?\ *,\ t_?\ *$
  - $t_?\ *,\ g_?\ ,\ t_?\ *$
  - $t_?\ *,\ g_?\ ,\ t_?\ *$



Bridge

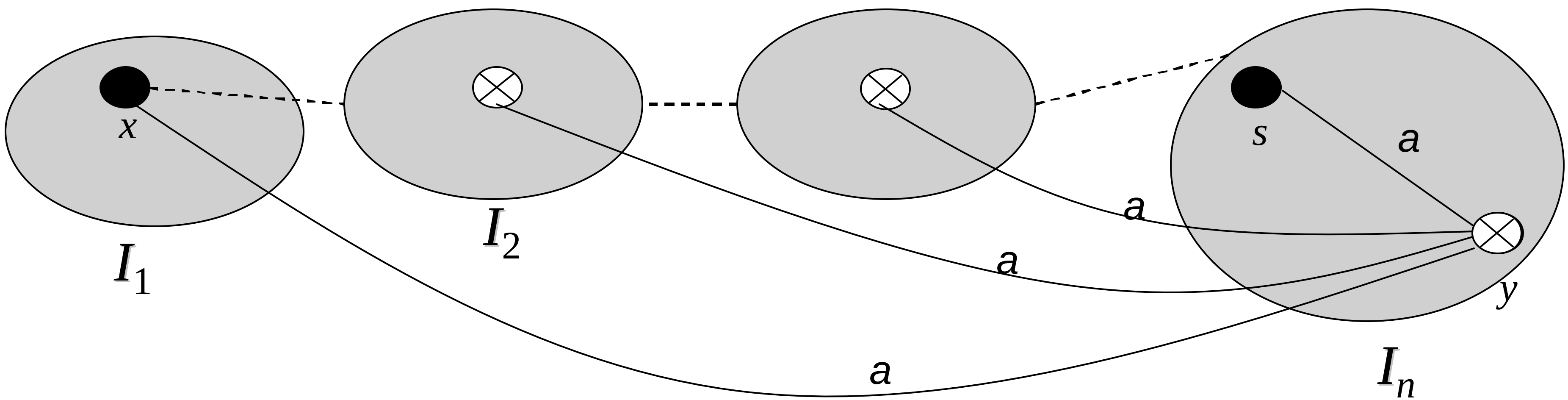


Theorem: Can_share(a,**x**,**y**,G_0) (for subjects)

- Subject_can_share(a, ***x***, ***y***,G_0) is true iff if ***x*** and ***y*** are subjects and
  - there is an a edge from x to y in G_0

  OR if:

  - ∃ a subject s ∈ G_0 with an ***s***-to-***y*** a edge, and
  - ∃ islands $I_1, \ldots, I_n$ such that ***x*** ∈ I_1, **s** ∈ I_n, and there is a bridge from I_j to I_{j+1}



What about objects? Initial, terminal spans

- ***x*** *initially spans* to ***y*** if ***x*** is a subject and there is a *tg*-path between them with *t* edges ending in a *g* edge (i.e., $t_? * g_?$)
  - ***x*** can grant a right to ***y***
- ***x*** *terminally spans* to ***y*** if ***x*** is a subject and there is a *tg*-path between them with *t* edges (i.e., $t_? *$)
  - ***x*** can take a right from ***y***

Theorem: Can_share(a,**x**,**y**,G_0)

- Can_share(a,x, y,G_0) iff there is an a edge from x to y in G_0 or if:
  - ∃ a vertex $s \in G_0$ with an s to y a edge,
  - ∃ a subject x' such that $x'=x$ or x' *initially spans* to x,
  - ∃ a subject s' such that $s'=s$ or s' *terminally spans* to s, and
  - ∃ islands $I_1, \ldots, I_n$ such that $x' \in I_1$, $s' \in I_n$, and there is a bridge from I_j to I_{j+1}