TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science, Institute of Systems Architecture, Operating Systems Group

Distributed Operating Systems Lecture

Security: Foundations, Security Policies, Capabilities

2014

Marcus Völp / Hermann Härtig

Can you trust your system?

... to protect your privacy / credentials / valuable data?

... to grant only trusted programs access to your data?

... to grant access to your data if / when and only if / when a trusted program needs it?

Can you trust your system?

... to protect your privacy / credentials / valuable data?

... to grant only trusted programs access to your data?

... to grant access to your data if / when and only if / when a trusted program needs it?

- How you can trust your system.
- How you can assure that your system is trustworthy.

Assurance

- trust developer / company
  - reputation
  - “I know the company so I can sue them if things go wrong”
- quality assuring processes
  - e.g., independent test and development team, documentation, ...
- certification
  - trust them because some experts said they are trustworthy
  - experts ensure that the company did their testing, ...
  - Examples:
    - ISO 9000
    - Common Criteria Security Evaluation
    - Arinc / DO 178b
- (formal verification)
  - mathematical proof of correctness
  - required as part of Common Criteria for EAL 7 (in parts), old BSI GISA

Security Evaluations

- Common Criteria (EAL 7)
  - Formal top level specification
  - Informal (through tests) correspondence of
  - source code to abstract specification
- GISA IT Security Evaluation Criteria (Q7)
  (old proposal for CC-EAL 7 from 1989)

  "The machine language of the processor used shall to a great extent be formally defined."

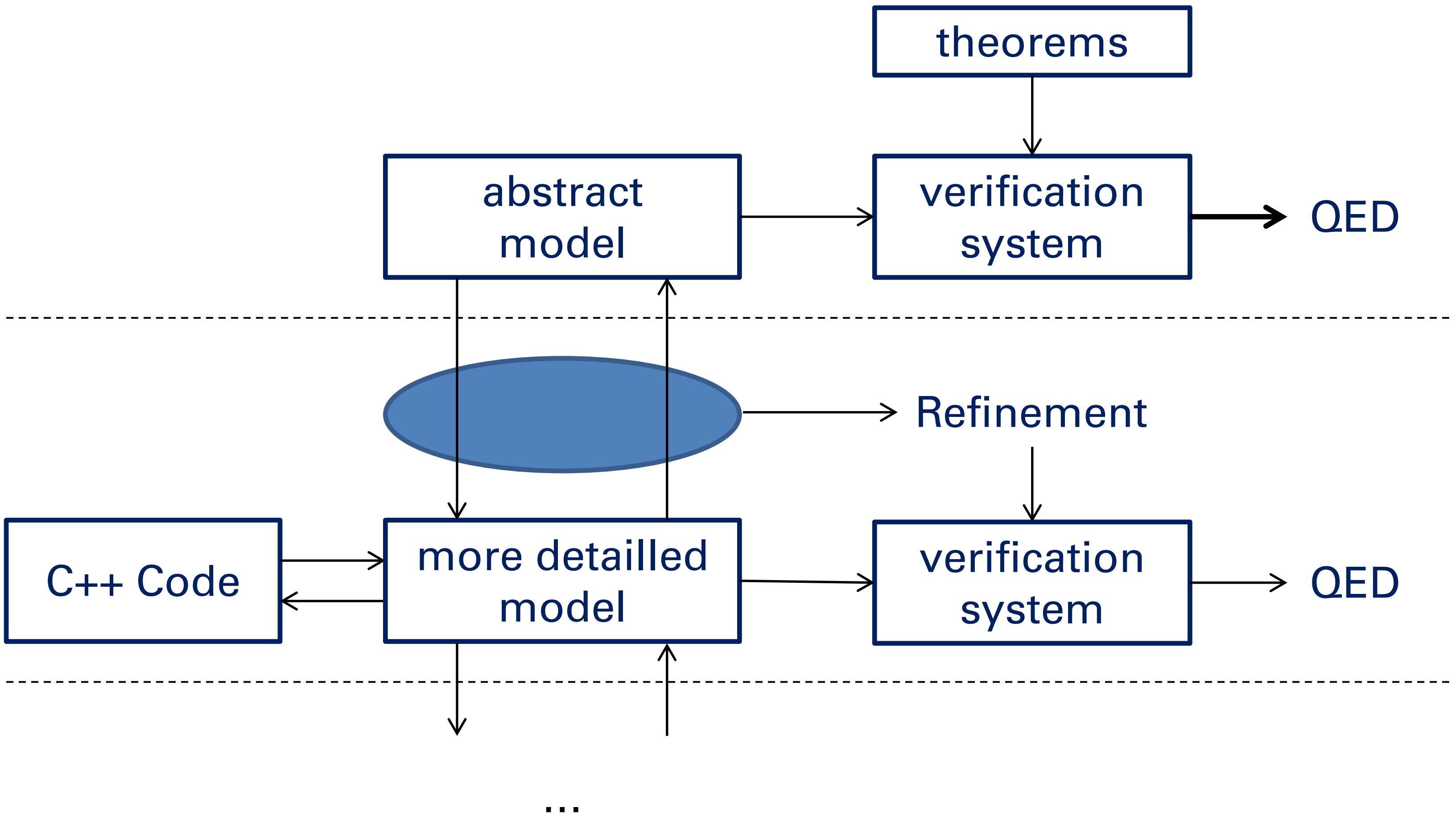
  "The consistency between the lowest specification level and the source code shall be formally verified."

  "The source code will be examined for the existence of covert channels, applying formal methods. It will be checked that all covert channels detected which cannot be eliminated are documented. [...]"

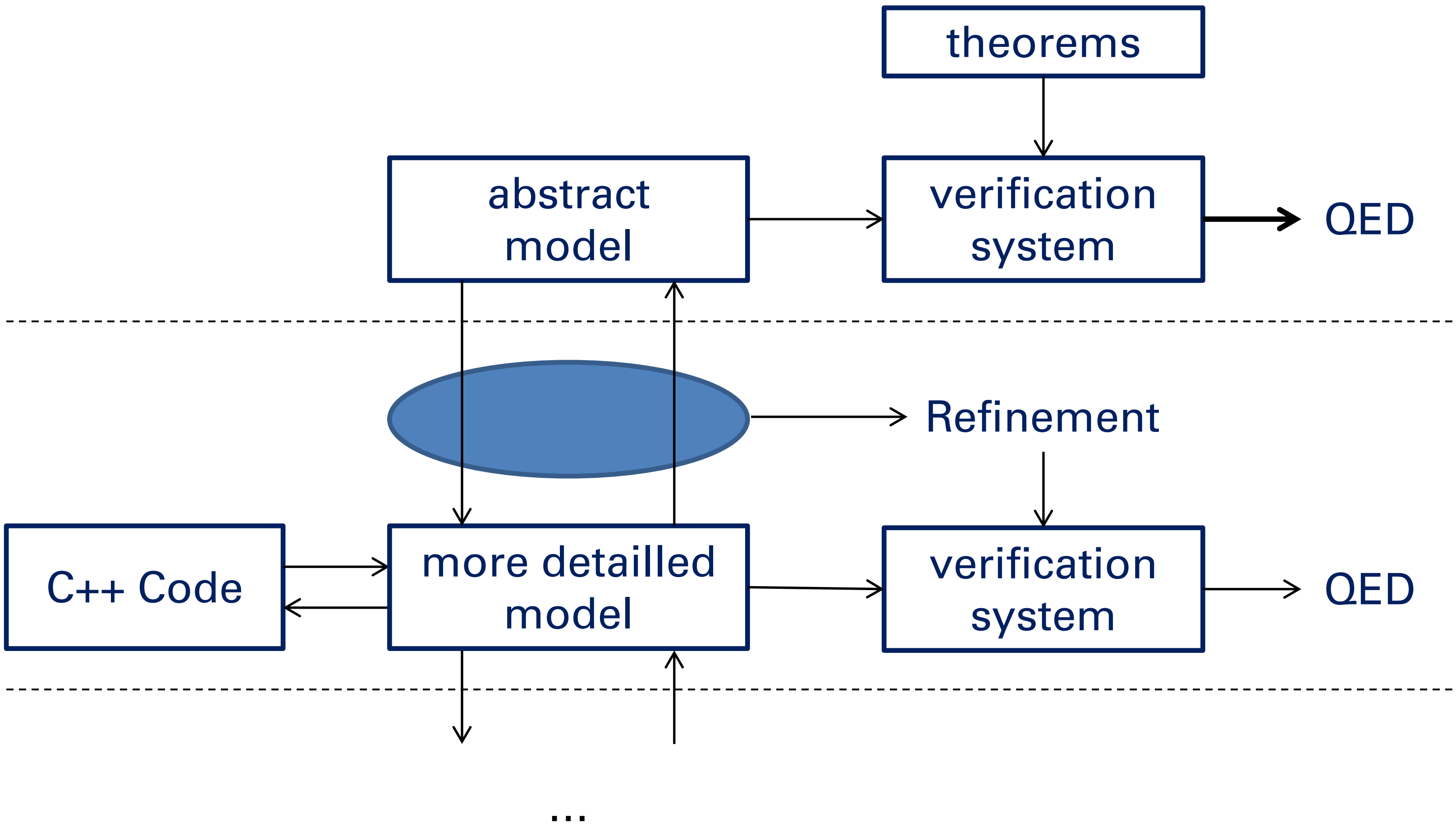
Outline

- Introduction
- Example Proof
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model

Formal Verification

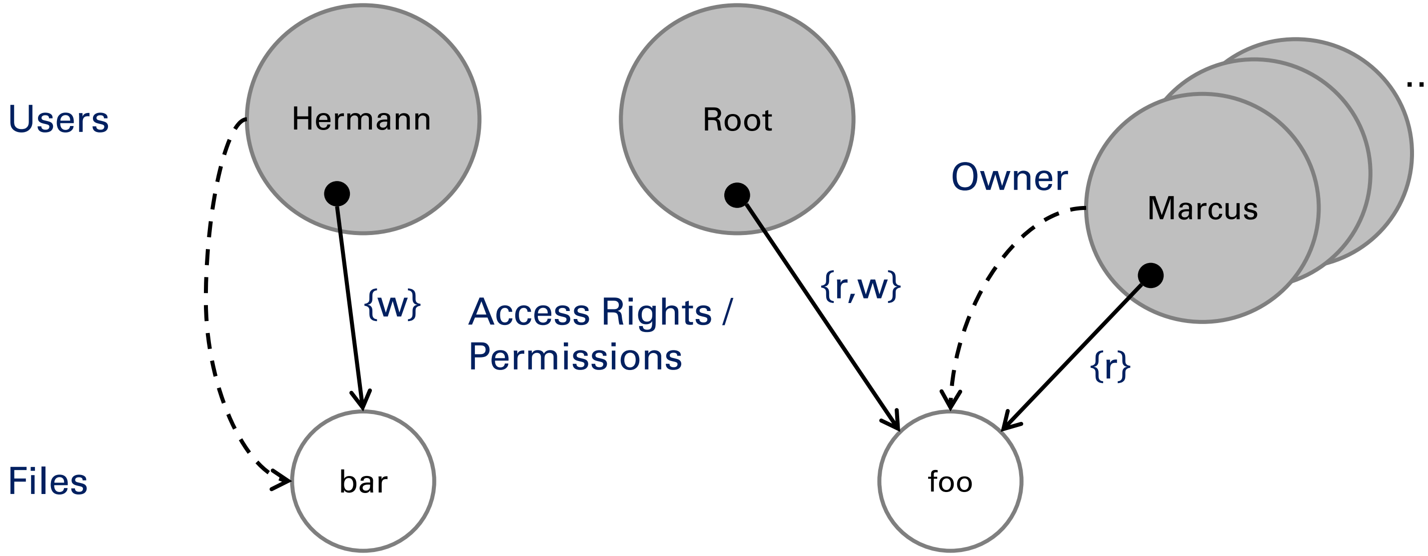


Formal Verification



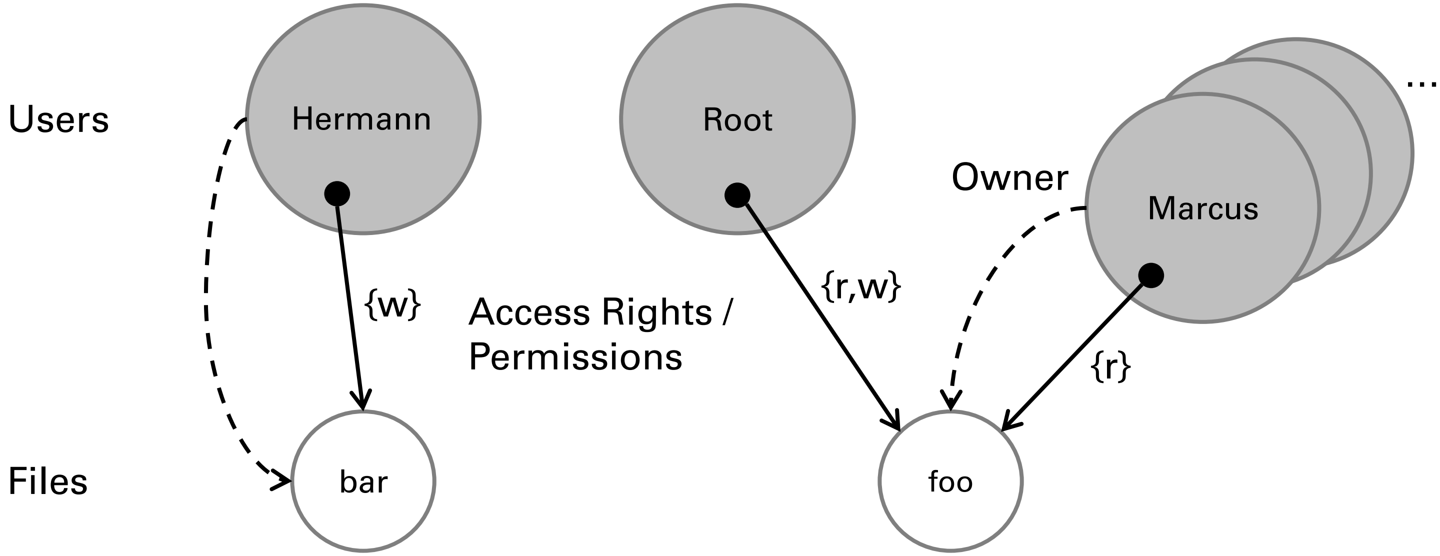
11 PY to verify a 10KLOC microkernel (seL4)

Proving Security – an Example



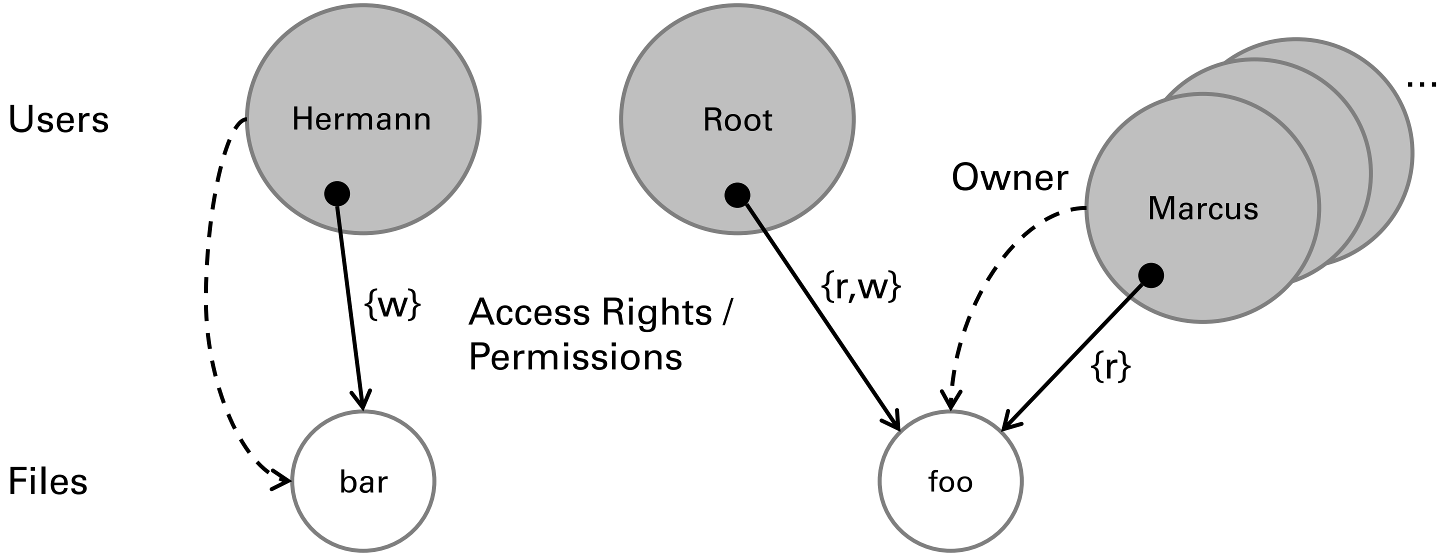
Operations: read, write, create / delete file, create / delete user, chmod

Proving Security – an Example



„Only the owner of a file or root shall obtain write permissions to a file.“

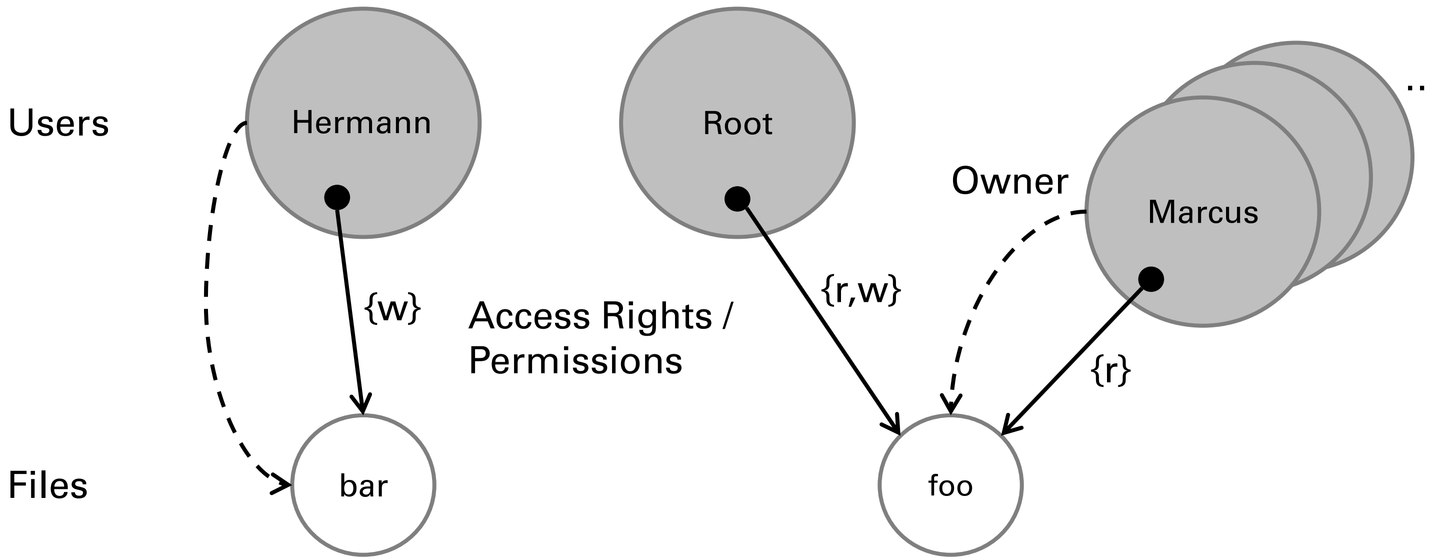
Proving Security – an Example



1st ingredient: abstract system model

- captures the details that are relevant for the theorem
- abstracts away all other details
- often characterized as states + state transitions

Proving Security – an Example



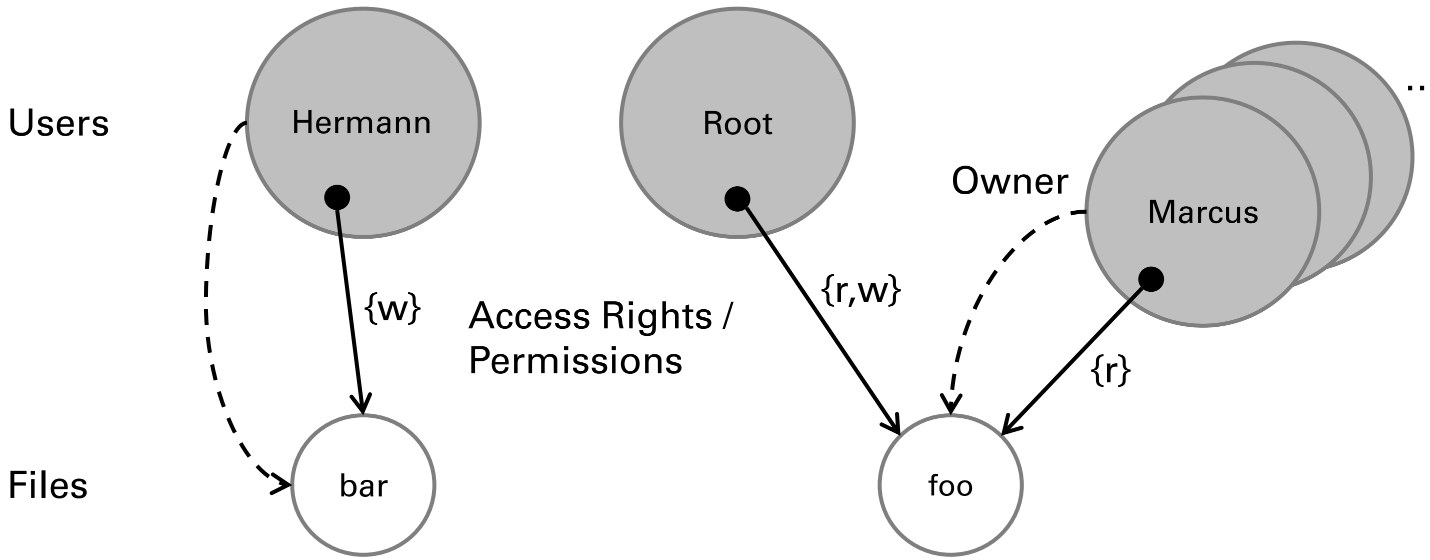
1[st] ingredient: abstract system model

- states:

  $\Sigma := \{ (U_{life}, F_{life}, \text{owner}, \text{rights}) \}$

$\sigma \in \Sigma :=$ ({root, hermann, marcus}, {foo, bar},
{(bar, hermann), (foo,marcus)}, // $F_{life} \rightarrow U_{life}$
{(hermann, bar, {w}), (root, foo, {r,w}), (marcus, foo, {r})}) // $F_{life} \times U_{life} \rightarrow 2^R$

Proving Security – an Example



1st ingredient: abstract system model

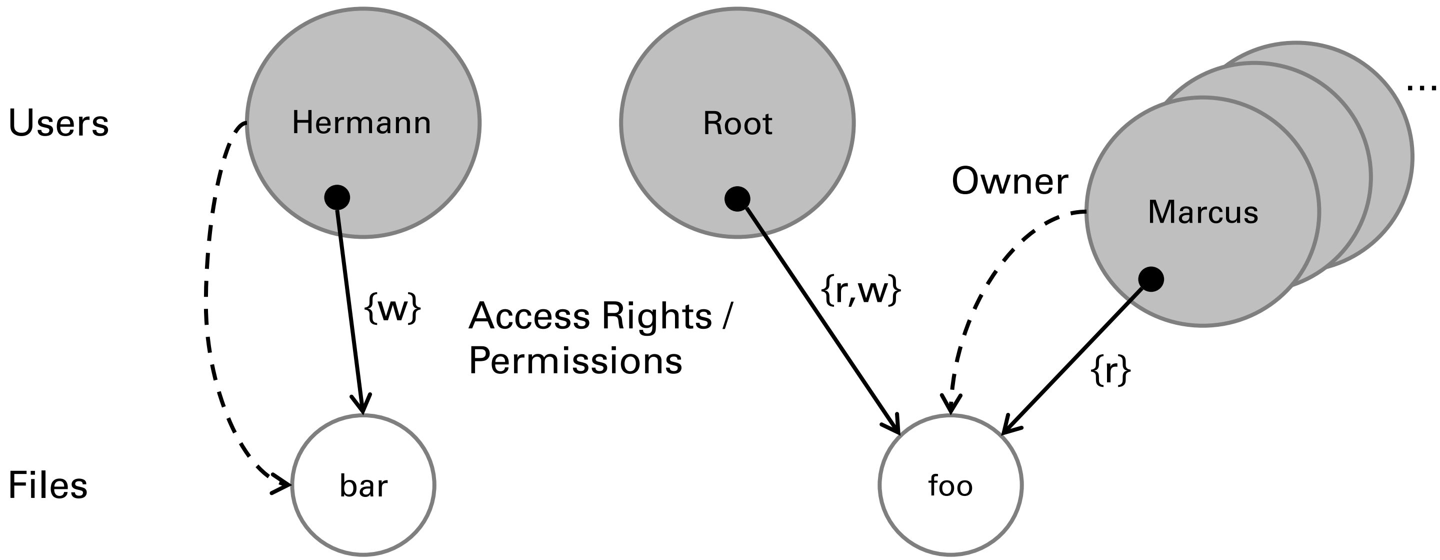
- state transitions:

  $C := \Sigma \rightarrow \Sigma$

read: $\sigma \rightarrow \sigma$

delete(bar) : $\sigma \rightarrow$ ({root, hermann, marcus}, {foo, ~~bar~~}, {~~(bar, hermann)~~, (foo,marcus)},
{~~(hermann, bar, {w})~~, (root, foo, {r,w}), (marcus, foo, {r})})

Proving Security – an Example



1st ingredient: abstract system model

– state transitions:

$C := \Sigma \to \Sigma$

read: $\sigma \to \sigma$

u.delete(bar) : $\sigma \to$ if u = root v u = σ .owner(bar) then
({root, hermann, marcus}, {foo, ~~bar~~}, {~~(bar, hermann),~~ (foo,marcus)},
{~~(hermann, bar, {w}),~~ (root, foo, {r,w}), (marcus, foo, {r})})
else σ endif

Proving Security – an Example

2nd ingredient: theorem

„Only the owner of a file or root shall obtain write permissions to a file.“

vs.

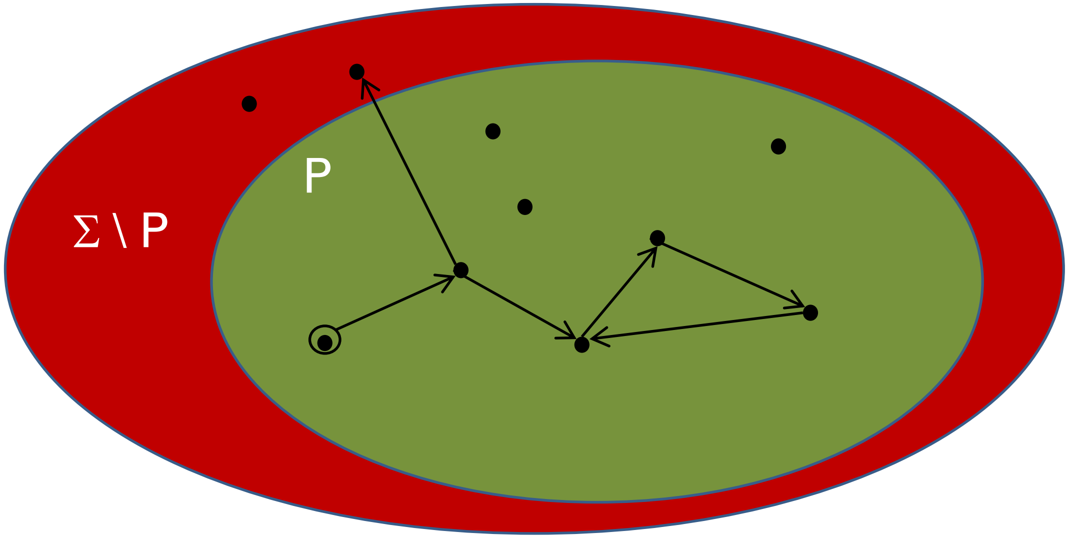
„Information in a file shall origin only from the owner of a file or from root.“

Proving Security – an Example

2nd ingredient: theorem

„Only the owner of a file or root shall obtain write permissions to a file.“

$$P : \Sigma \rightarrow \{true, false\}$$



secure wrt. P if $\sigma_0 \in P$ and $\Sigma_{reachable} \subseteq P$

2nd ingredient: theorem

„Only the owner of a file or root shall obtain write permissions to a file.“

$$P : \Sigma \rightarrow \{true, false\}$$

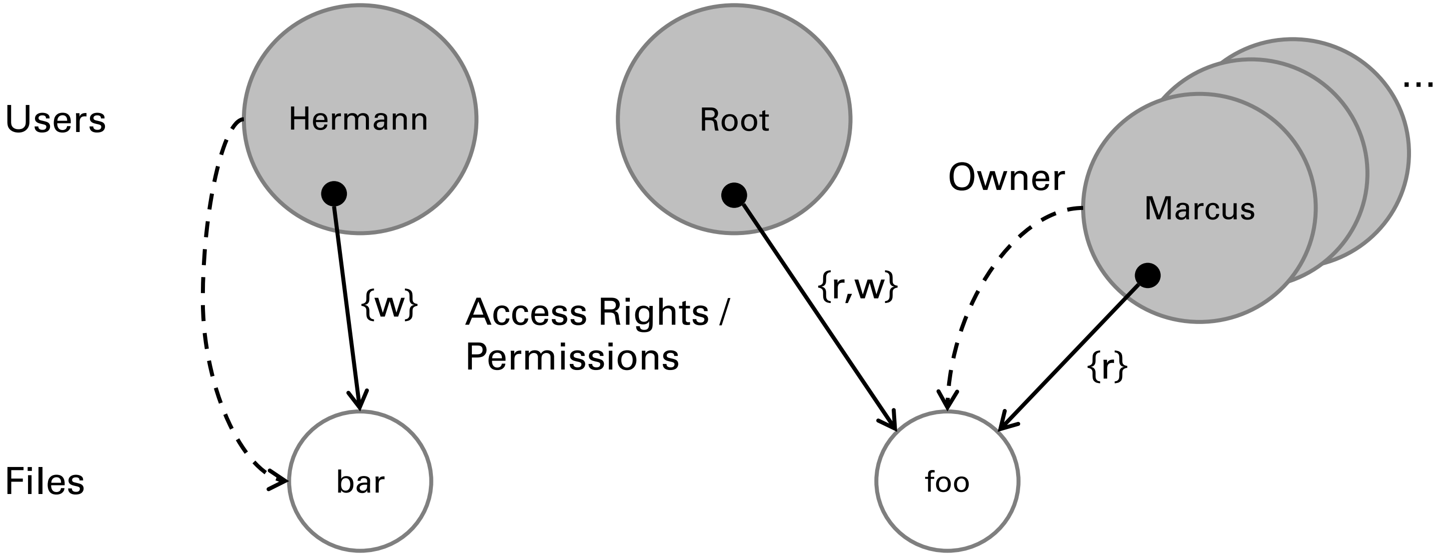
$$P(\sigma) := \forall f \in F_{life}, u \in U_{life}.\ w \in \sigma.rights(u,f) \Rightarrow u = root \vee u = \sigma.owner(f)$$

Proving Security – an Example

3rd ingredient: proof

Theorem: $\Sigma_{reachable} \subseteq P$

$P(\sigma) := \forall f \in F_{life}, u \in U_{life}. w \in \sigma.rights(u,f) \Rightarrow u = root \vee u = \sigma.owner(f)$



Operations: read, write, create / delete file, create / delete user, chmod

Proving Security – an Example

3rd ingredient: proof

Theorem: $\Sigma_{reachable} \subseteq P$

$$P(\sigma) := \forall f \in F_{life}, u \in U_{life}.\ w \in \sigma.rights(u,f) \Rightarrow u = root \vee u = \sigma.owner(f)$$

Proof:
by induction over all traces

$$\sigma_0 \xrightarrow{u.c} \sigma \xrightarrow{u'.c'} \sigma' \xrightarrow{u''.c''} \sigma'' \xrightarrow{u'''.c'''} \sigma''' \ \ldots$$

Operations: read, write, create / delete file, create / delete user, chmod

Proving Security – an Example

3rd ingredient: proof

Theorem: $\Sigma_{reachable} \subseteq P$

$$P(\sigma) := \forall f \in F_{life}, u \in U_{life}. w \in \sigma.rights(u,f) \Rightarrow u = root \vee u = \sigma.owner(f)$$

Proof:
by induction over all traces

$$\sigma_0 \xrightarrow{u.c} \sigma \xrightarrow{u'.c'} \sigma' \xrightarrow{u''.c''} \sigma'' \xrightarrow{u'''.c'''} \sigma''' \ldots$$

Operations: read, write, create / delete file, create / delete user, chmod

induction step succeeds for read, ..., delete user

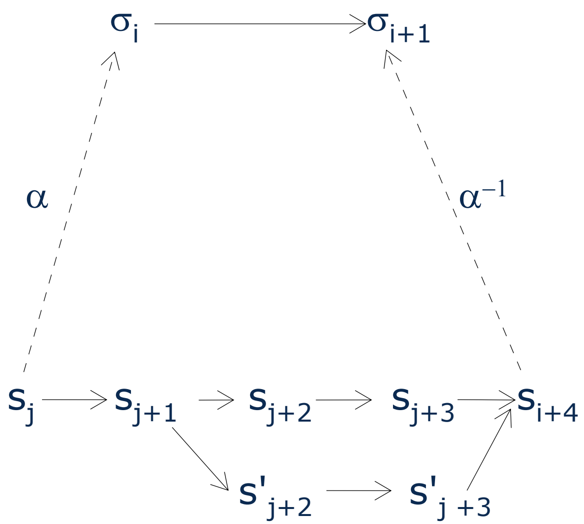
but
chmod(Marcus, bar, {w})

Proving Security – an Example

4th ingredient: refinement

```
chmod(u, f, R)(s) :=
        if u = root v owner(f, u) then
                s with rights (u, f) := R
        else
                s
        endif

sys_chmod:
  parse_parameters();
        owner = file.owner;
        if (current_thread->user == root ||
     current_thread->user == owner)
 {
         file->set_acl(user, rights);
 }
```



Outline

- Introduction
- Example Proof
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model

Security Policies - Definition

[Bishop: Computer Security Art and Science]

- **Security Policy**

  A *security policy* P is a statement that partitions the states S of a system into a set of authorized (or secure) states (e.g., $\Sigma_{sec} := \{ \sigma \in \Sigma \mid P(\sigma) \}$) and a set of unauthorized (or non-secure) states.

- **Secure System**

  A secure system is a system that starts in an authorized state and that cannot enter an unauthorized state

  (i.e., $\Sigma_{reachable} \subseteq \Sigma_{sec}$)

Confidentiality, Integrity, Availability

Confidentiality

prevent unauthorized disclosure of sensitive information (prevent information leakage).

Definition:

Information or data I is *confidential* with respect to a set of entities X if no member of X can obtain information about I.

Example: the PIN of my EC-Card is XXXX

Confidentiality, Integrity, Availability

Integrity

correctness of information or data

Definition 1:

Information I is *integer* if it is current, correct and complete

Confidentiality, Integrity, Availability

Integrity

correctness of information or data

Definition 1:

Information I is *integer* if it is current, correct and complete

Definition 2: (crypto)

Either information is current, correct, and complete (Def 1) or it is possible to **detect** that these properties do not hold.

Confidentiality, Integrity, Availability

Integrity

correctness of information or data

Definition 1:

Information I is *integer* if it is current, correct and complete

Definition 2: (crypto)

Either information is current, correct, and complete (Def 1) or it is possible to **detect** that these properties do not hold.

Recoverability

Eventually damaged information can be recovered.

Confidentiality, Integrity, Availability

Availability

accessibility of information, services and data

Definition:

A resource I is available with respect to X if all members of X can access I.

in practice, availability has also quantitative aspects:

- real-time systems:

  I is available within t milliseconds

- reliability:

  the probability that I is **not** available is less than 10^{-6}

TECHNISCHE UNIVERSITÄT DRESDEN

Security Policies - Classification

Concern

– confidentiality	e.g., Bell La Padula	(Document Mgmt)
– integrity	e.g., Biba	(Inventory System)
– availability		
– hybrid	e.g., Chinese Wall	(Clinical Information)

Level of Enforcement

- **discretionary**

  A user can allow or deny access to its objects

- **mandatory**

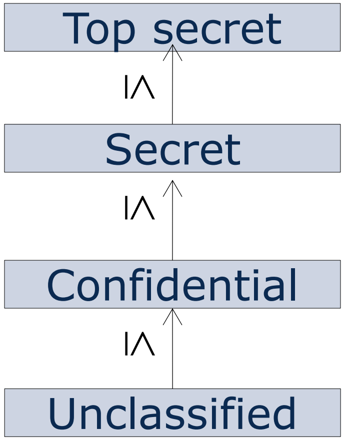
  System-wide rules control who may access an object

Bell-LaPadula Policy '73 (simple version)

Concern: confidentiality

set of secrecy levels: L

higher secrecy level indicates more sensitive information; greater need to keep this information confidential



total order: $\leq$

domain: Entity -> L

- each subject has a *security clearance:* dom(s) $\in$ L
- each object has a *security classification:* dom(o) $\in$ L

Bell-LaPadula Policy '73 (simple version)

Policy: (L, ≤, dom)

rules for reading / writing

simple security condition

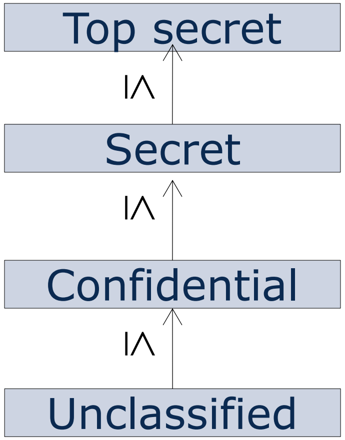
a subject s can read only lower or equally classified objects o

s can read o <=> dom(o) ≤ dom(s)

*** - property**

a subject s can write only higher or equally classified objects o

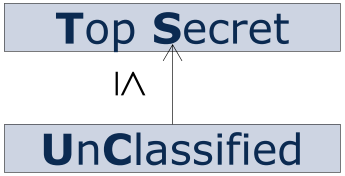
s can write o <=> dom(s) ≤ dom(o)



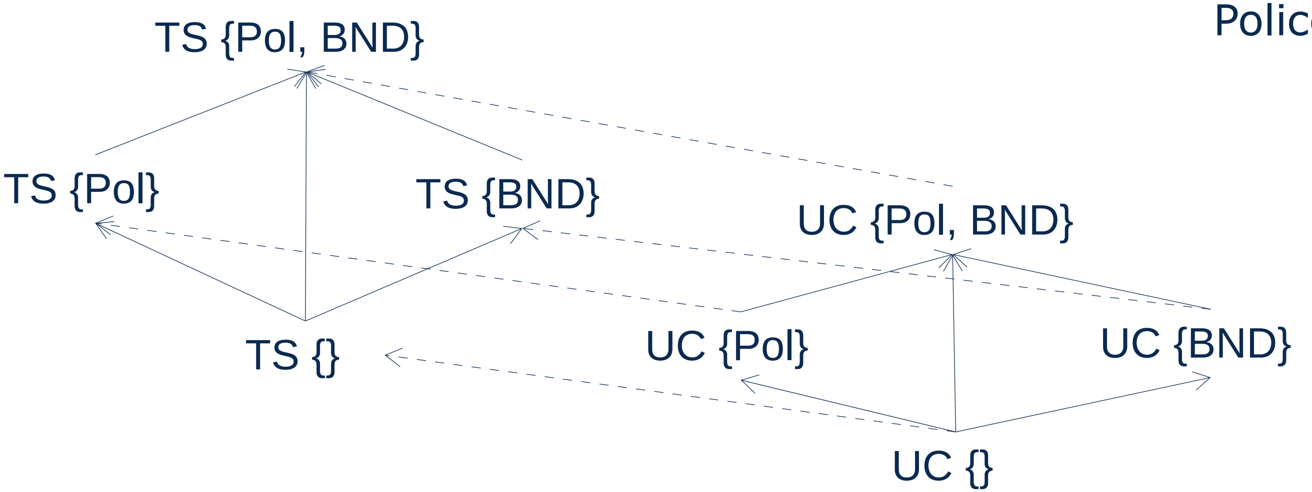
Bell-LaPadula: Multi-Level Security Policy

Policy: (L, ≤, dom)

≤ is a partial order, (L, ≤) form a lattice



Categories:
Police, BND

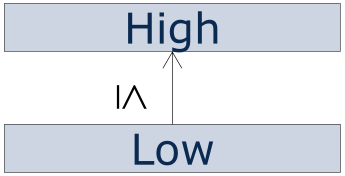


Bundesverfassungsschutzgesetz §17 - §26:

in general, no information exchange between the BND and the Police

Biba '77

Concern: Integrity (prevent damage)



(L, ≤, dom) dual to MLS

high integrity information must not be tainted with low integrity data.

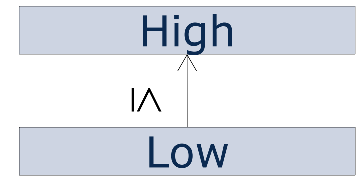
- s can read o <=> dom(s) ≤ dom(o)
- s can write o <=> dom(o) ≤ dom(s)

Biba '77 Low Water Mark

- **Concern:** Integrity (prevent damage)

  (L, ≤, dom) dual to MLS

  high integrity information must not be tainted with low integrity data.



  - ~~s can read o <=> dom(s) ≤ dom(o)~~
  - if s reads o then dom'(s) = min(dom(s), dom(o))

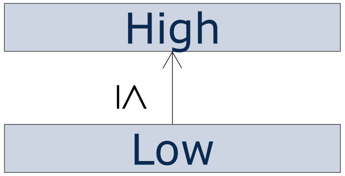
  - s can write o <=> dom(o) ≤ dom(s)

Biba '77 Low Water Mark

- **Concern:** Integrity (prevent damage)

  (L, ≤, dom) dual to MLS

  high integrity information must not be
  tainted with low integrity data.



  - ~~s can read o <=> dom(s) ≤ dom(o)~~
  - if s reads o then dom'(s) = min(dom(s), dom(o))
  - s can write o <=> dom(o) ≤ dom(s)

- **Problem:** label creep

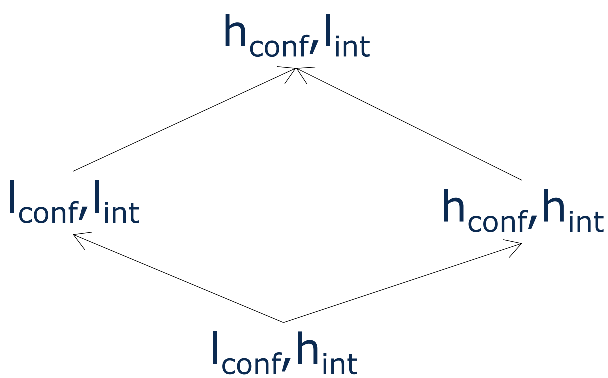
  subject clearances decrease over time
  no means to “clean” a tainted subject

Denning '76 + Sandhu '93

Confidentiality and integrity are dual and can be represented in the same lattice:

Confidentiality: $l_{conf} \leq h_{conf}$

Integrity: $h_{int} \leq l_{int}$



Brewer '89: Chinese Wall

Concern: Conflict of interest (integrity + confidentiality)

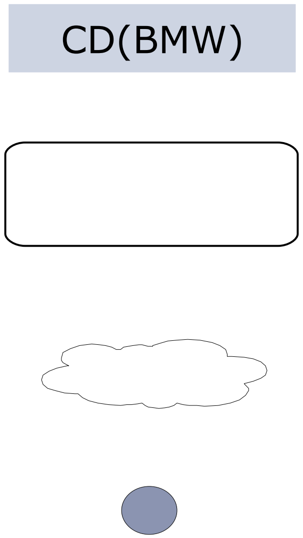
Example: British stock exchange
    a trader must not represent two competitors

Company Datasets (CD):
    set of objects (files) related to a company

Conflict of Interest Class (COI):
    CDs of companies in competition

Sanitized Objects:
    cleared to the public

Subjects (e.g., the trader)



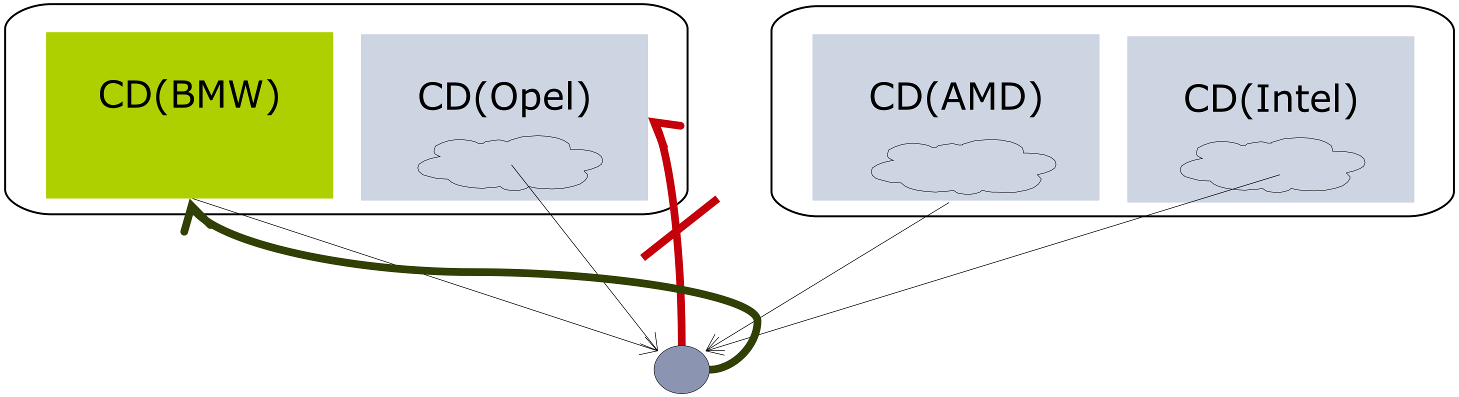
Brewer '89: Chinese Wall

*** property**

s can write o <=>

s can read o
and
if s can read an unsanitized object o' then o' must belong to the same company as o

i.e., $\forall$ o'. s can read o' => CD(o') = CD(o)



Outline

- Introduction
- Example Proof
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model

Access Control Matrix

Subjects S
Objects O
Entities $E = S \cup O$
Rights R

Matrix: S x E x R

	o_1	o_2	s_1	s_2
s_1	r, w	r	r, w	r
s_2	r, w	-	w	r, w

Operations:

- read / write entity
- create subject / object
- destroy subject / object
- **enter / delete R into cell (s,o)**

Access Control List

Subjects	S
Objects	O
Entities	E = S ∪ O
Rights	R

list of S x R tuples stored with every Entity

	o_1	o_2	s_1	s_2
s_1	r, w	r	r, w	r
s_2	r, w	-	w	r, w

abbreviations:

- owner / group e.g., Unix [user; group; all]
- wildcards e.g., sysadmin_*

conflicts:

- e.g., u – r; g + r resolved by order of occurrence / rules

Capabilities

Subjects S
Objects O
Entities $E = S \cup O$
Rights R

list of E x R tuples stored with every subject

	o_1	o_2	s_1	s_2
s_1	r, w	r	r, w	r
s_2	r, w	-	w	r, w

more in a few minutes

Principle of Attenuation

German: Abschwächung / Verminderung

A subject s must not be able to give away rights that it does not possess

	o_1	o_2	s_1	s_2
s_1	r, w	r	r, w	r
s_2	r, w	-	w	r, w

Problem: ACMs cannot enforce the principle of attenuation

e.g., s_1.enter w into (s_2, o_2)

Solution:

replace "enter r into (s,o)" with:

s'.grant R into (s,o) :=

if R $\subseteq$ (s',o) **then** enter R into (s,o)

Capabilities

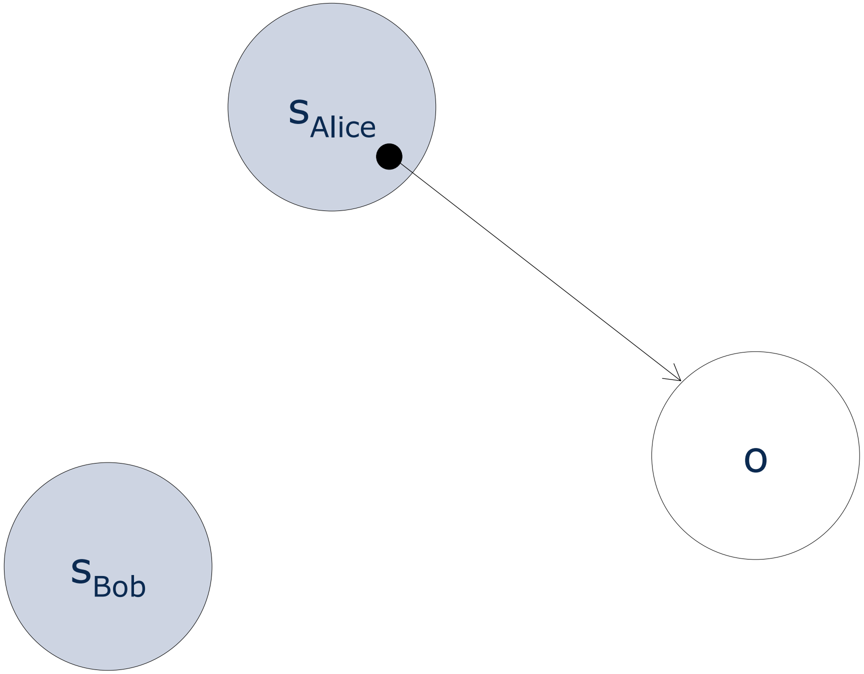
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - take / grant
  - diminish / remove



Capabilities

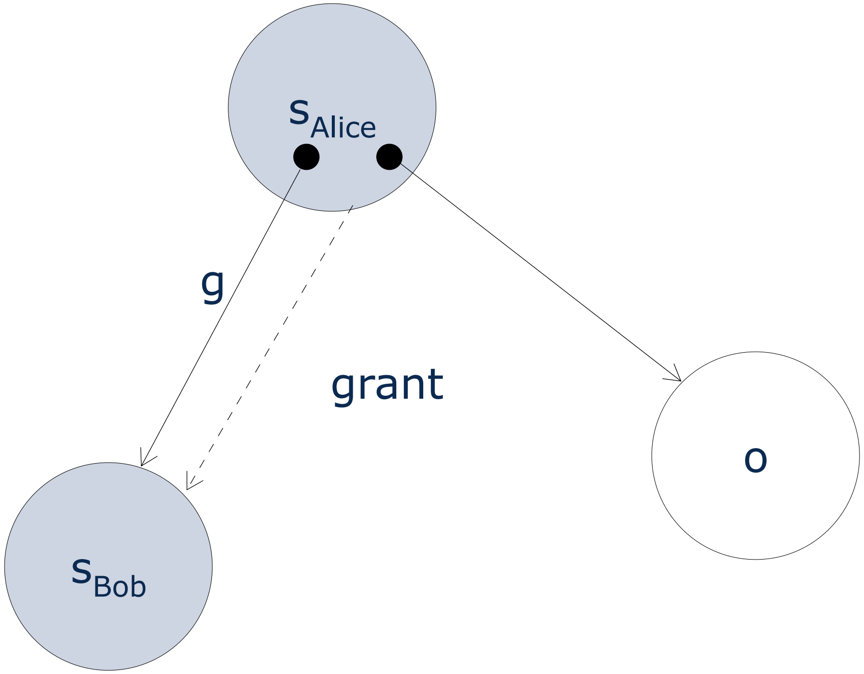
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - take / **grant**
  - diminish / remove



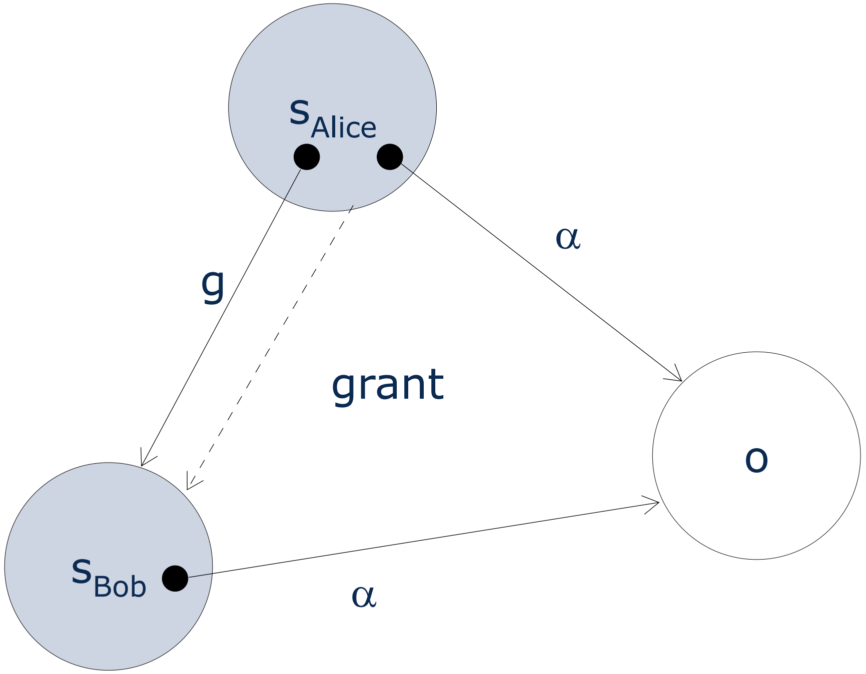
Capabilities

Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / remove



Capabilities

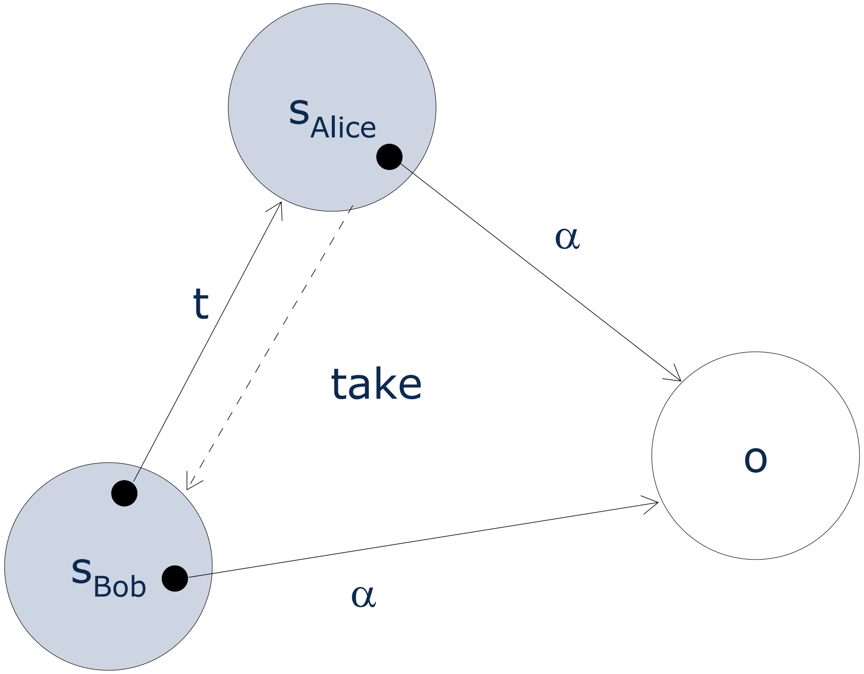
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - **take** / grant
  - diminish / remove



Capabilities

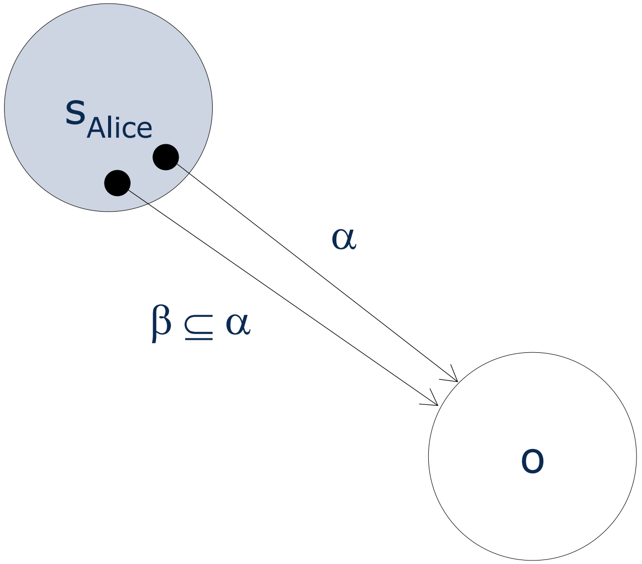
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - take / grant
  - **diminish** / remove



Capabilities

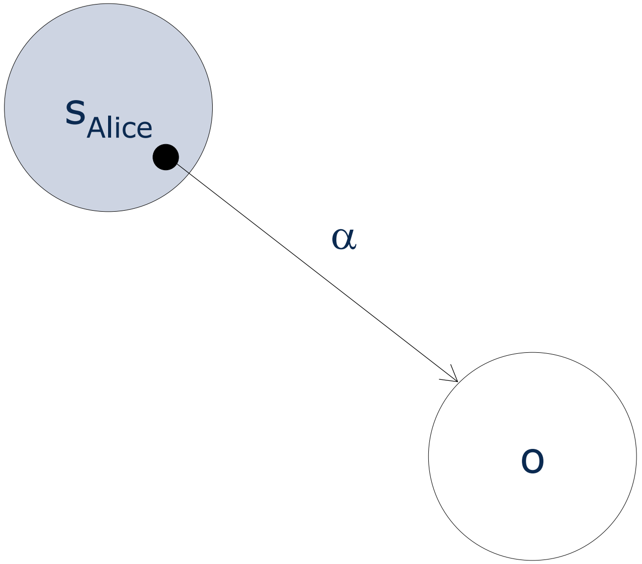
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - take / grant
  - diminish / **remove**



Capabilities

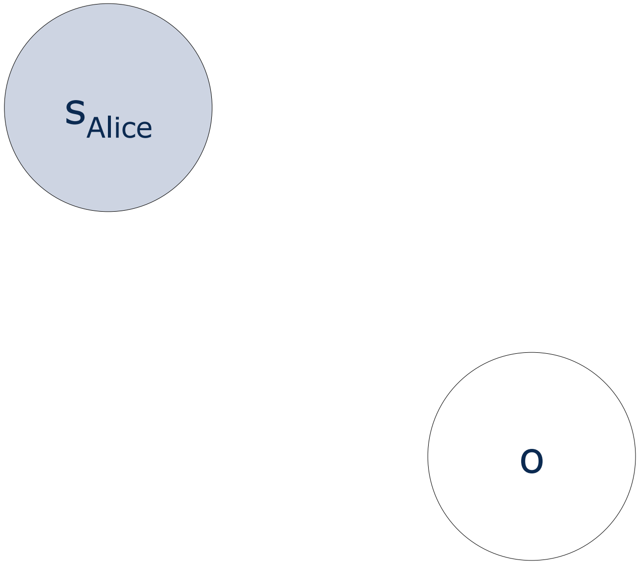
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - take / grant
  - diminish / **remove**



Capabilities

Implementation:

Software:	OS protected segment / memory page
Hardware:	Cambridge CAP / TLB
Cryptography:	Amoeba

Problems:

- How to control the propagation of capabilities?
- How to revoke capabilities?

Propagation of Capabilities

Problem is dual to controlling ACM / ACL modifications

Permissions on channel capabilities:

take permission (t); grant permission (g)

Permission on the capability:

copy permission

Right-diminishing channels:

extension to the take-grant model by J. Shapiro

Propagation of Capabilities

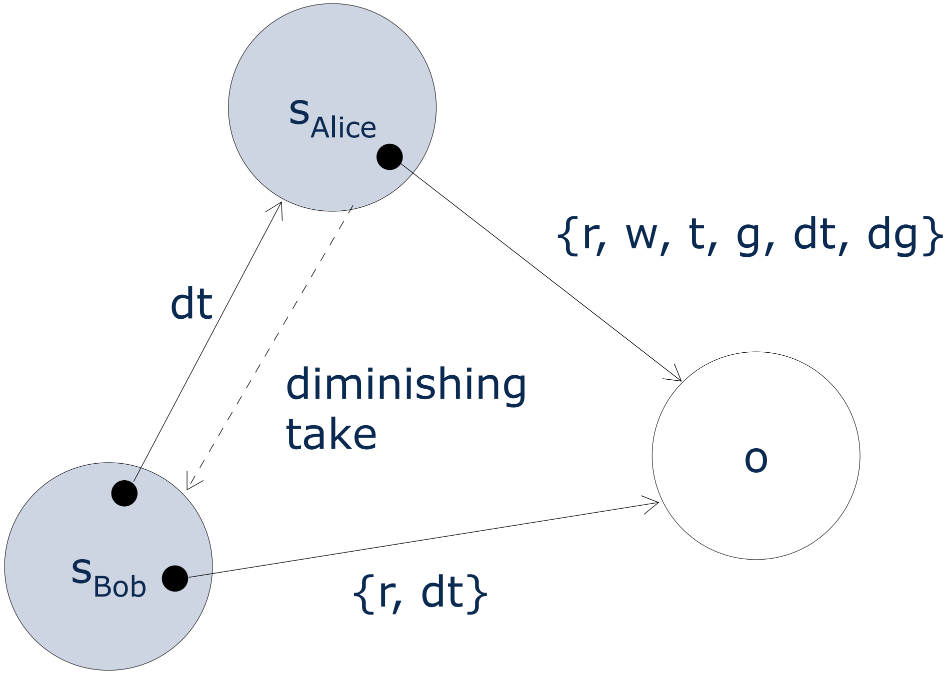
Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

- on objects
  - read / write
  - create / destroy

- on capabilities
  - take / grant
  - diminish / remove
  - **diminishing take**
  - diminishing grant



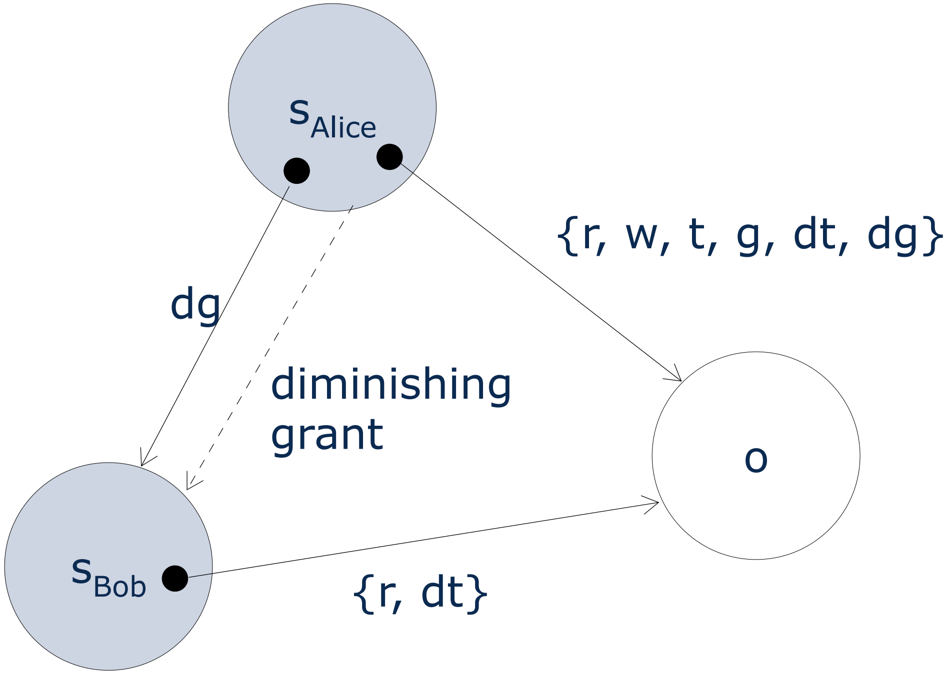
Propagation of Capabilities

Definition: unforgeable token E x R

possession of a capability is necessary and sufficient to access the referenced entity

Operations:

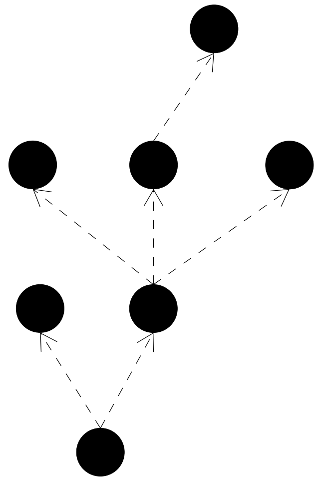
- on objects
  - read / write
  - create / destroy
- on capabilities
  - take / grant
  - diminish / remove
  - diminishing take
  - **diminishing grant**



Capability Revocation

Amoeba: leases – invalid after a certain amount of time

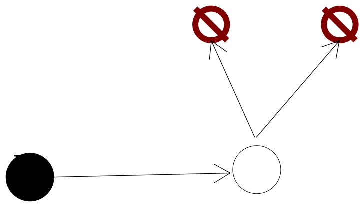
L4: find and invalidate all direct and indirect copies



Eros: indirection objects

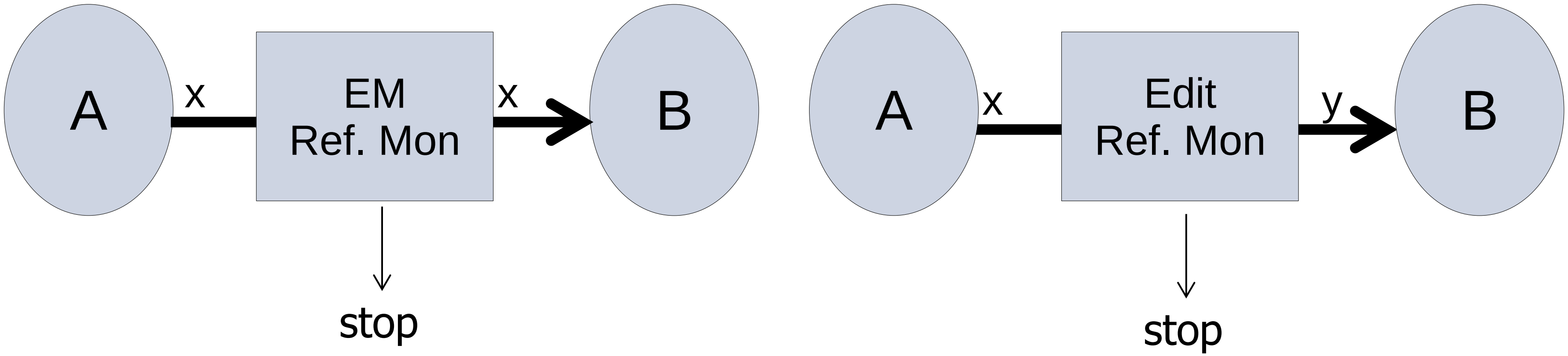
use stored capabilities
but no take / grant

revoke by destruction



Reference Monitors

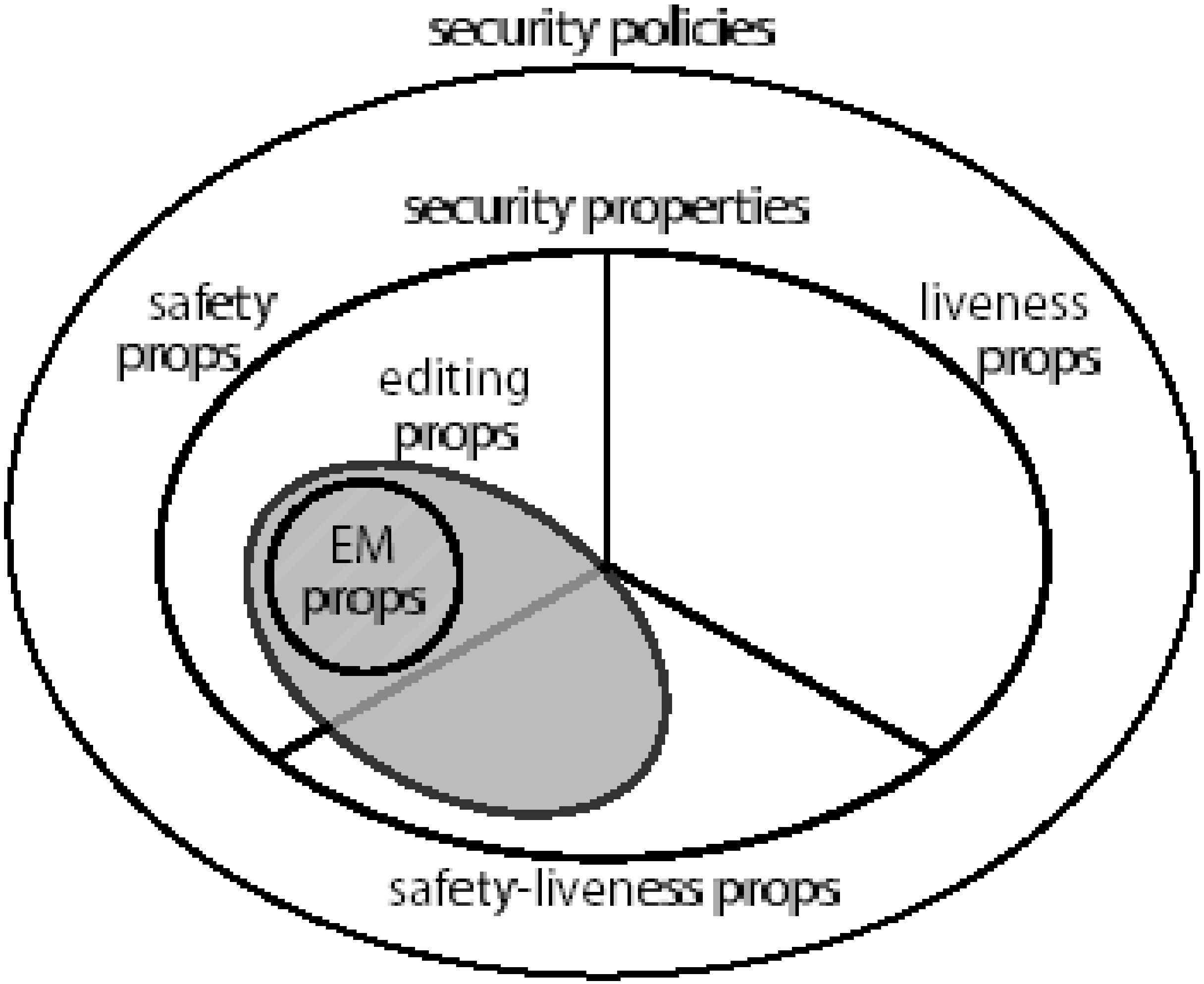
EM: suppress or pass
Edit: modify message



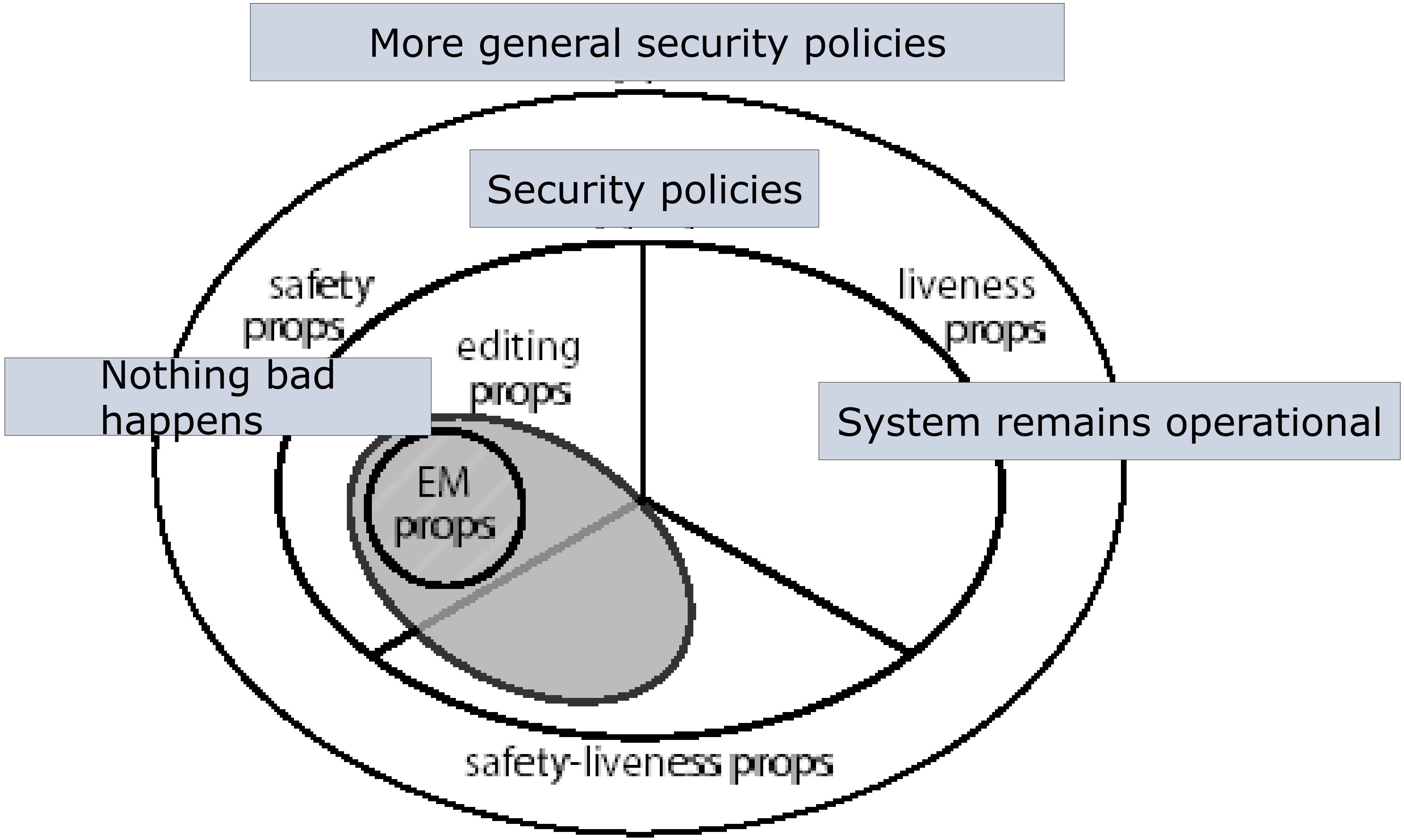
Schneider '98 / Bauer '02:
Theoretical results on the set of security policies that are enforceable with EM / Edit automata

!!! Results are in part based on a different system model !!!

Reference Monitors



Reference Monitors



Outline

- Introduction
- Example Proof
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- Take-Grant Protection Model

Leakage (of Access Rights)

Given a system S and a security policy P, decide whether S can enter a state in which s can access o with right r (i.e., whether access right r is leaked into (s,o)).

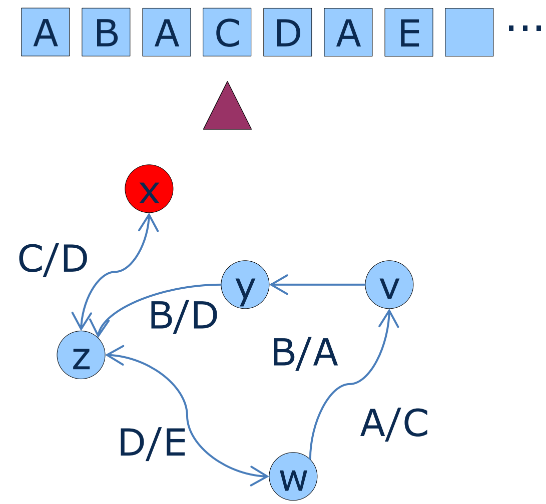
Theorem:
For a system S with a generic ACM it is in general undecidable whether S leaks r into (s, o).

Proof:
by reduction to the halting problem

Turing Machine

infinite tape

tape symbols	M: A, B, C, ...
state automaton	K: x, y, z, ...
head	



Operations:

- read symbol at head
- perform a transition step of the automaton based on this symbol
- write a new symbol to the tape
- move head one step to the left or to the right

$$\delta: K \times M \rightarrow K \times M \times \{L, R\}$$

Halting Problem

Given a turing machine TM and a program P, find a program of the TM that decides whether P will terminate (halt)

TM ≅ universal TM ≅ while

Theorem: the halting problem is undecidable

Halting Problem

TM ≅ universal TM ≅ <u>while</u>

Theorem: the halting problem is undecidable

Proof: by contradiction

assume such a program P exists; write two programs:

does_P_terminate_on_input_E (P, E) :=
if P(E) terminates { return true } else { return false }

test (P) := while (does_P_terminate_on_input_E(P, P))

now, if does_P_terminate_on_input_E(test, test) returns true, test(test) must terminate [*if condition*]

but then the condition of the while loop is true, which means test(test) will not terminate ↯

=> there cannot be a program that decides for all P, E whether P terminates on E

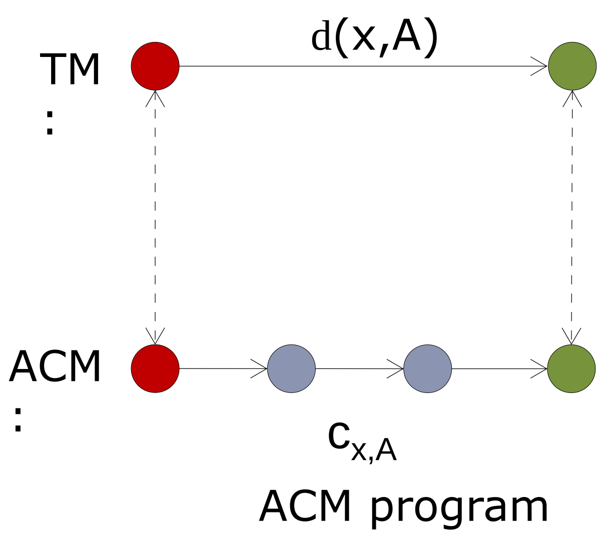
Leakage is Undecidable

Proof: by reduction to the halting problem

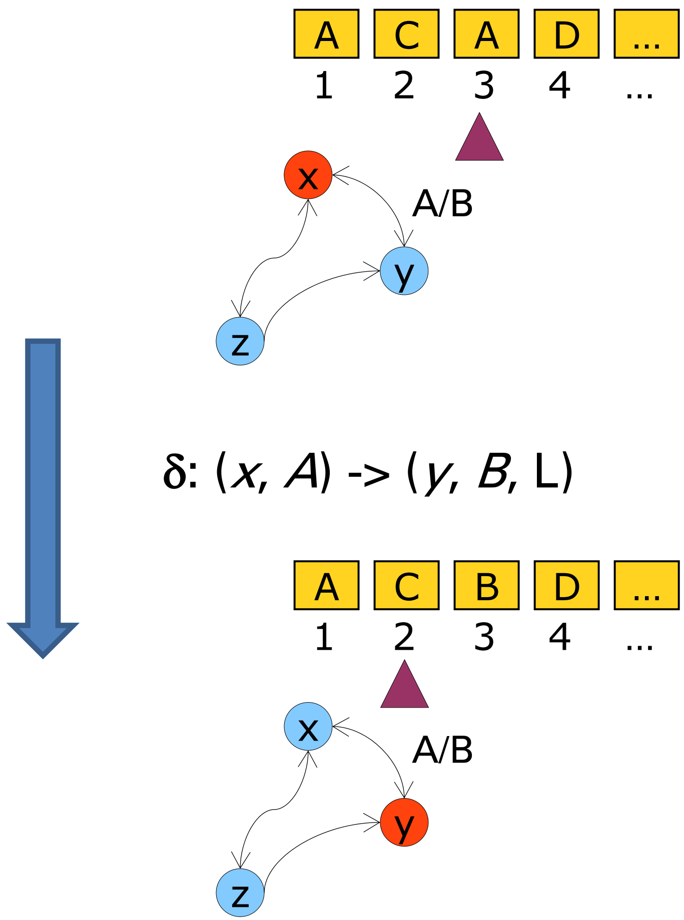
1. Simulate a TM with the ACM
2. Define a correspondence relation such that
   r is leaked to (s,o) <=> TM halts

=> leakage in the ACM could be used to solve the halting problem, which is known to be undecidable

=> leakage is undecidable



Simulating a TM with an ACM

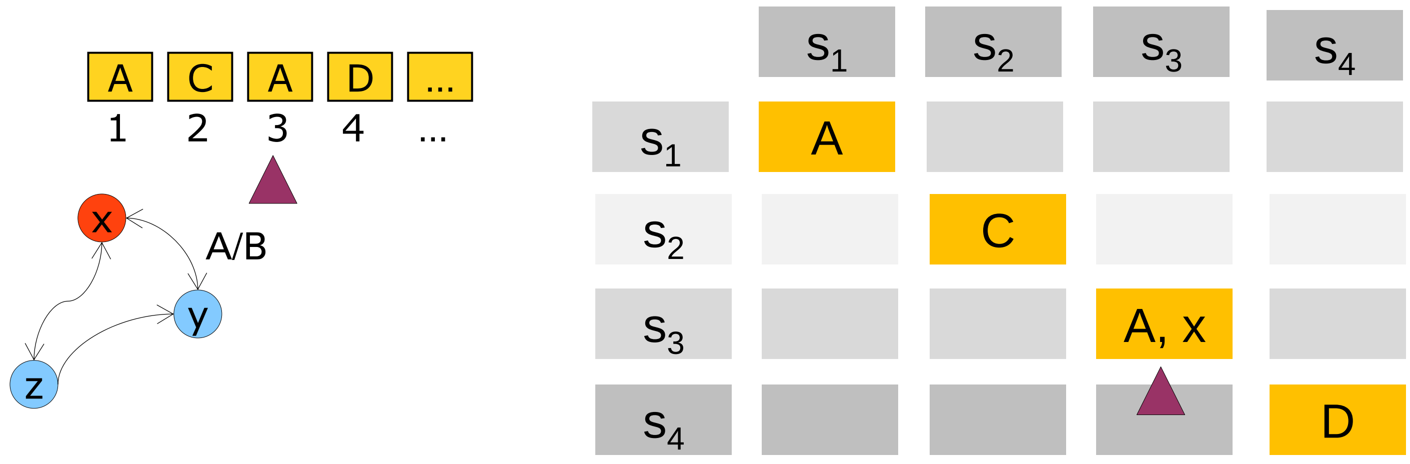


	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			A, x	
s_4				D

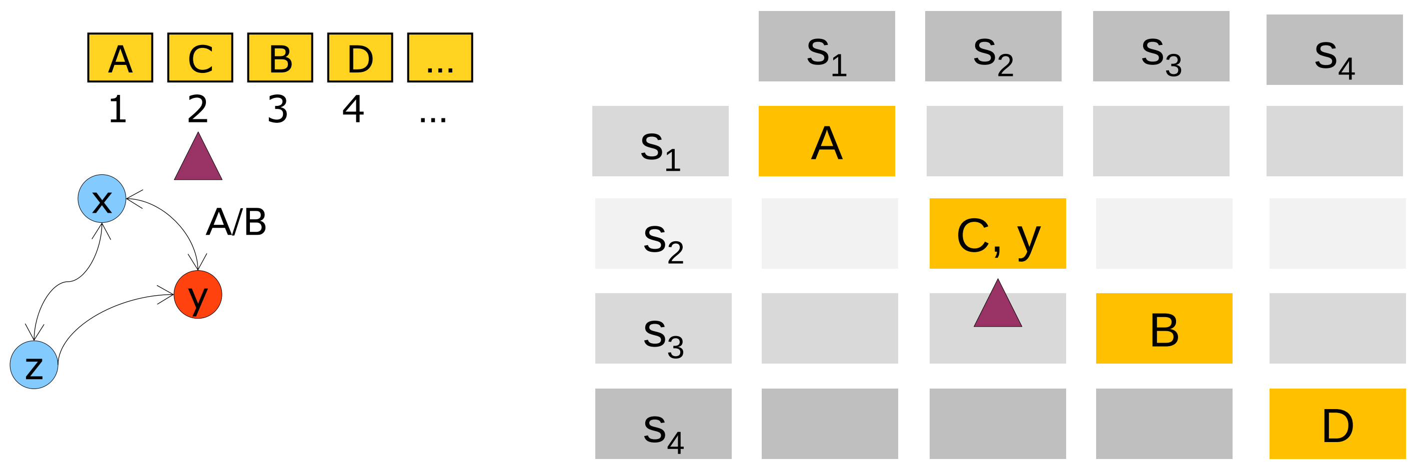
ACM Operations:

- create subject s
- create object o
- destroy subject s
- destroy object o
- enter r into (s, o)
- delete r from (s, o)

Simulating a TM with an ACM



δ: (*x*, *A*) -> (*y*, *B*, L)

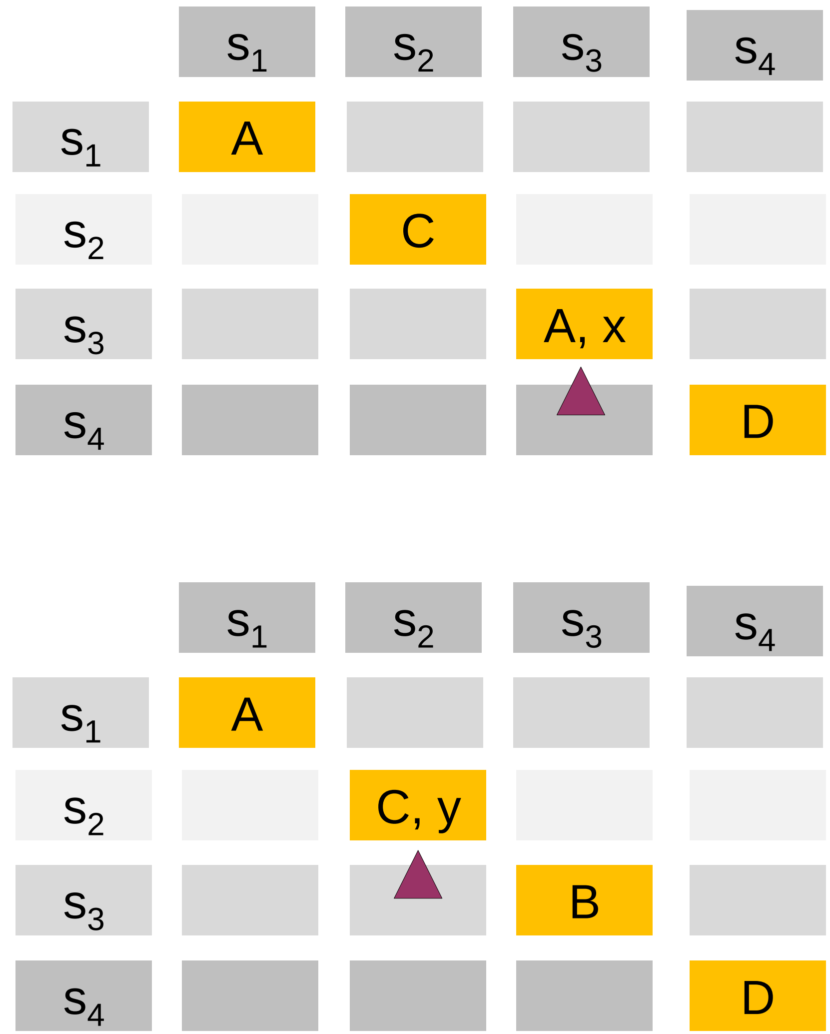


Simulating a TM with an ACM

$\delta: (x, A) \text{ -> } (y, B, L)$

$c_{x,A}(s_{head}, s_{left}) :=$

if $x \in (s_{head}, s_{head}) \wedge$
$A \in (s_{head}, s_{head})$
then
...



Simulating a TM with an ACM

δ: (x, A) -> (y, B, L)

$c_{x,A}(s_{head}, s_{left})$:=

if $x \in (s_{head}, s_{head}) \wedge$
$A \in (s_{head}, s_{head})$
then
delete x,A from (s_{head}, s_{head})
...

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3				
s_4				D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C, y		
s_3			B	
s_4				D

Simulating a TM with an ACM

δ: (*x*, *A*) -> (*y*, *B*, L)

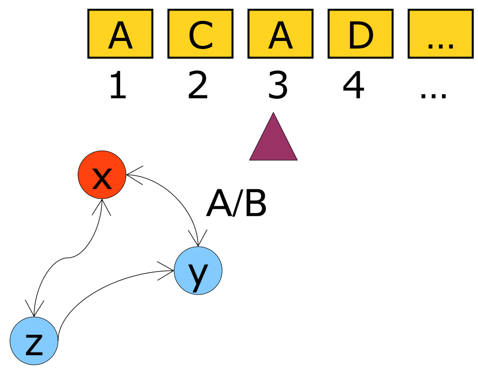
$c_{x,A}$ (s_{head}, s_{left}) :=

if x ∈ (s_{head}, s_{head}) ∧
  A ∈ (s_{head}, s_{head})
then
  delete x,A from (s_{head}, s_{head})
  enter B into (s_{head}, s_{head})
  enter y into (s_{left}, s_{left})
  ...

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C,y		
s_3			B	
s_4				D

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C, y		
s_3			B	
s_4				D

Simulating a TM with an ACM



	s_1	s_2	s_3	s_4
s_1	A			
s_2		C		
s_3			A, x	
s_4				D

δ: (*x*, *A*) -> (*y*, *B*, L)

$c_{x,A}\ (s_{head},\ s_{left})\ :=$

...

	s_1	s_2	s_3	s_4
s_1	A			
s_2		C, y		
s_3			B	
s_4				D

x is leaked into (si,si)

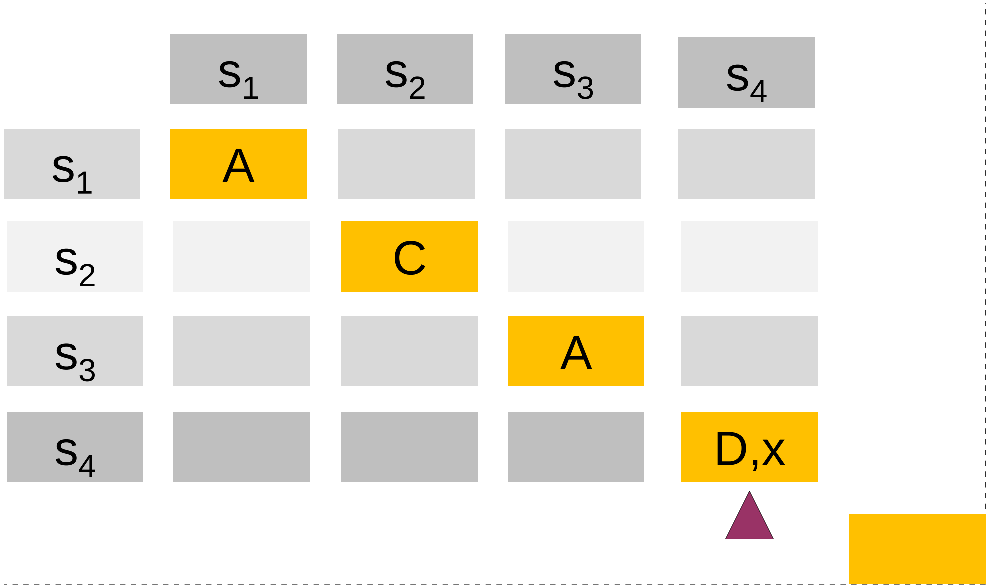
⇔

TM halts in x

Simulating a TM with an ACM

Problem 1:

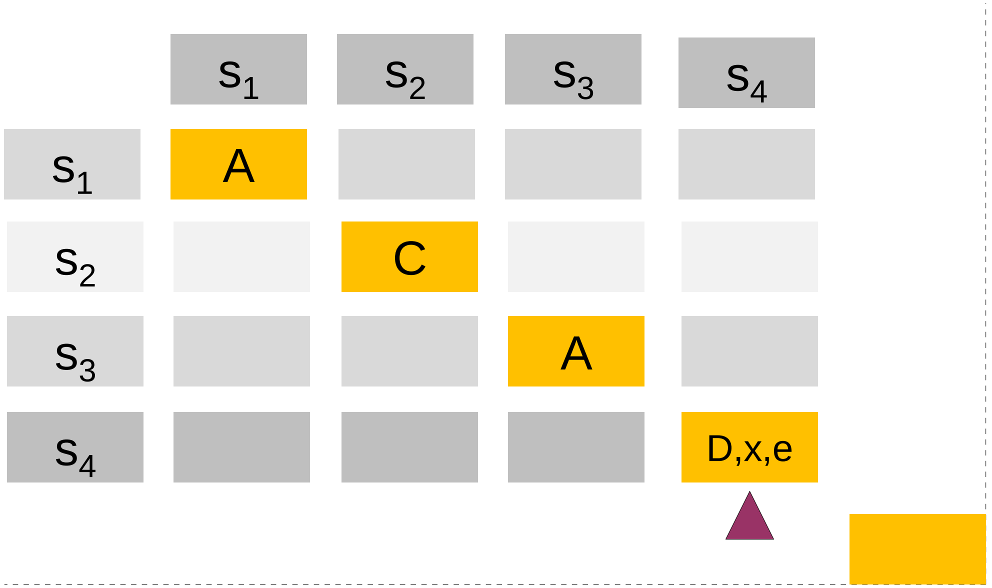
How to detect if we are at the last cell?



Simulating a TM with an ACM

Problem 1:

How to detect if we are at the last cell?



Simulating a TM with an ACM

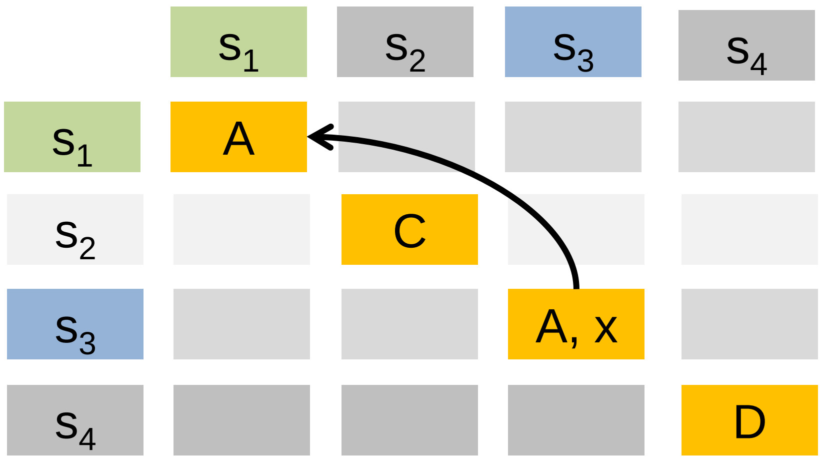
Problem 2:

How do we restrict the ACM to only execute the TM program?

$c_{x,A}$ (s, s') :=

...

applies to all s, s' pairs; not only neighboring



Simulating a TM with an ACM

Problem 2:

How do we restrict the ACM to only execute the TM program?

$c_{x,A}$ (s, s') :=

...

applies to all s, s' pairs; not only neighboring

	s_1	s_2	s_3	s_4
s_1	A	I		
s_2		C	I	
s_3			A, x	I
s_4				D

Outline

- Introduction
- Example Proof
- Security Policies
- Policy Enforcement Mechanisms
- Undecidability of Leakage
- **Take-Grant Protection Model**

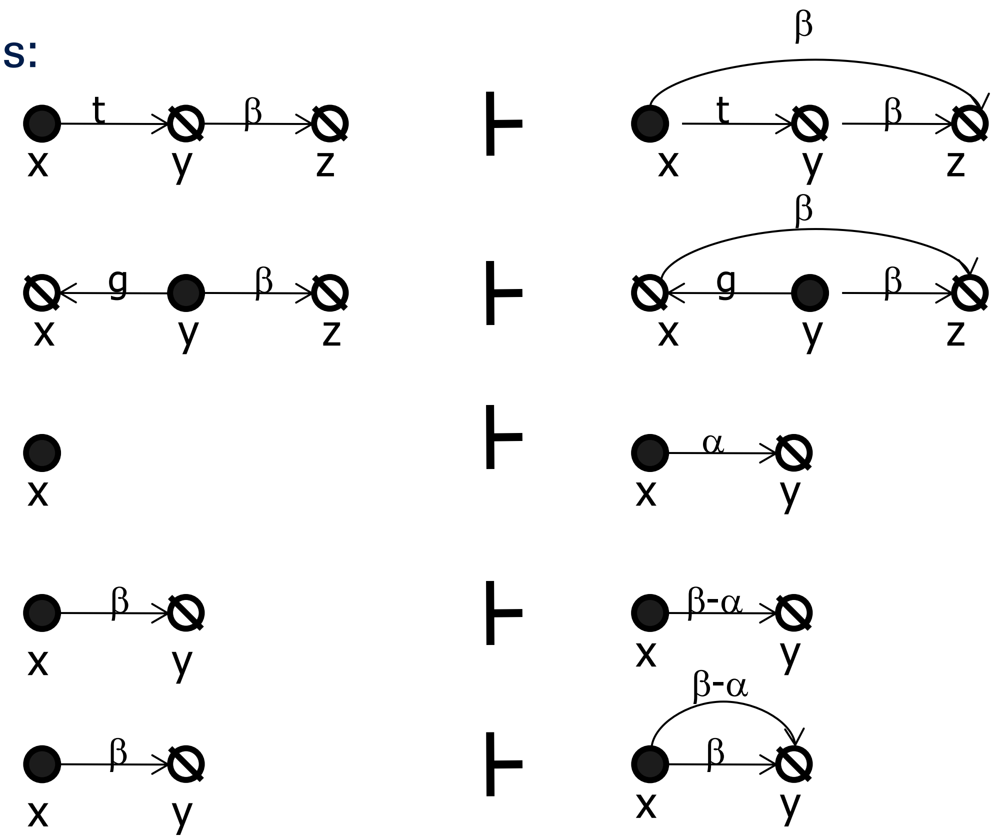
Take Grant Protection Model

Vertices: ○ object, ● subject (⊘ either object or subject)

Edges: ● —r→ ○ subject has capability with r right on object

Transition Rules:

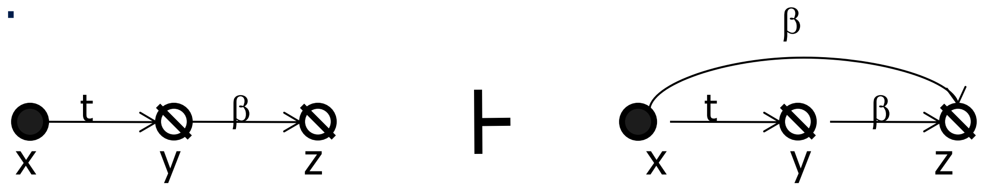
- Take
- Grant
- Create
- Remove
- Diminish



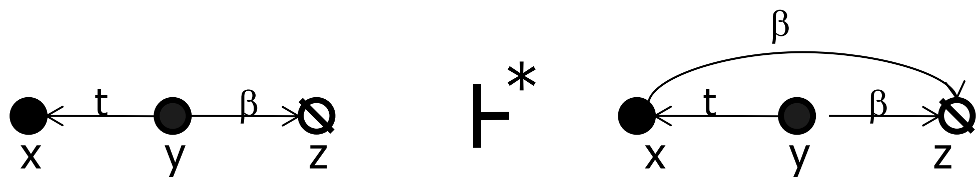
Take Grant Protection Model

A few Lemmas:

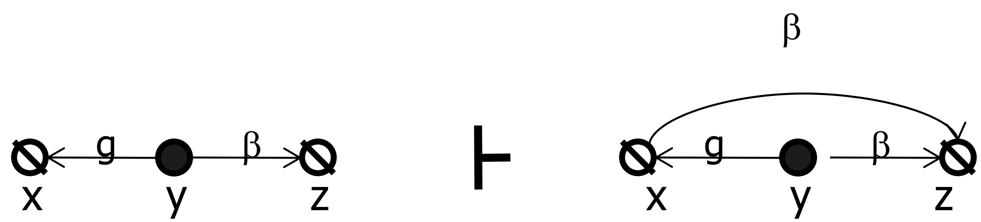
- Take



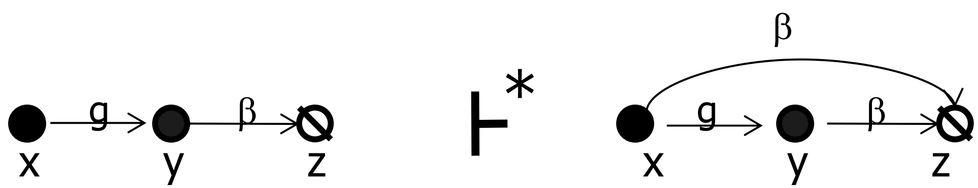
- Lemma 1:



- Grant

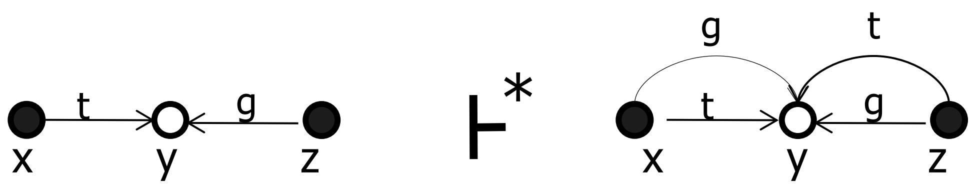


- Lemma 2:



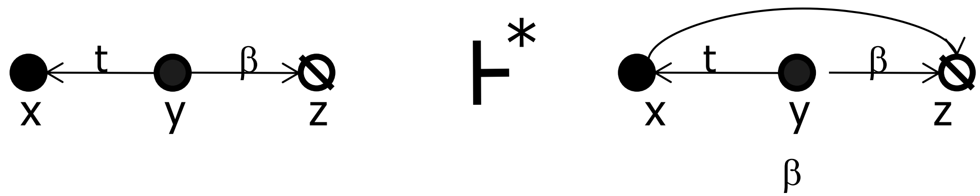
Take Grant Protection Model

A few Lemmas:

- Lemma 3: 

Take Grant Protection Model

Proof of Lemma 1



x.create	v (tg)
y.take	g on v
y.grant	β on z to v
x.take	β on z from v

Lemmas 2 and 3 are left for the exercises

Take Grant Protection Model

Theorem:

Leakage in the Take-Grant Protection Model is decidable (in linear time)

Proof Sketch:

construct potential access graph G
apply take + grant + 3 lemmas until G does not change anymore

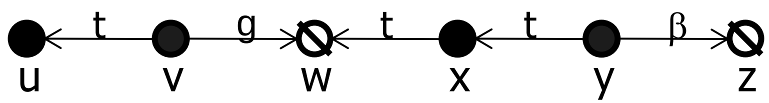
r is leaked to (s,o) if s holds (o, r) in the potential G

Note:
- delete / diminish / remove only reduce access
  => they can be omitted for the construction of G

- create introduces new entities which cannot get more privileged than their creators

Take Grant Protection Model

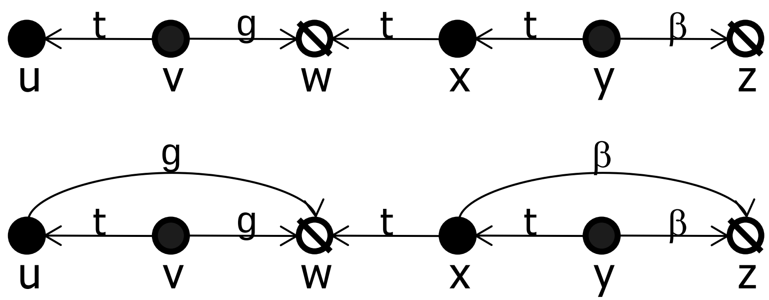
Example:



$\vdash^*$ by Lemma 1

Take Grant Protection Model

Example:

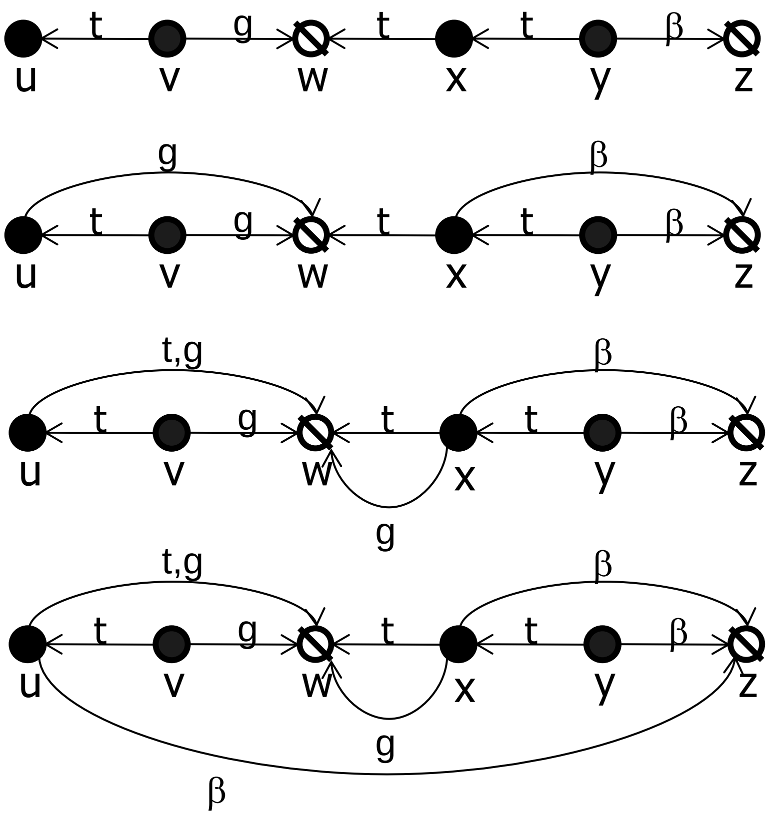


$\vdash^*$ by Lemma 1

$\vdash^*$ by Lemma 3

Take Grant Protection Model

Example:



$\vdash^*$ by Lemma 1

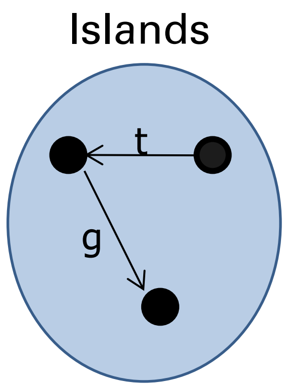
$\vdash^*$ by Lemma 3

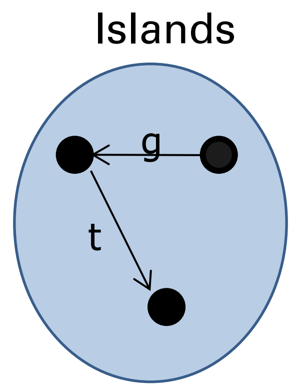
$\vdash^*$ x.grant β on z to w
u.take β on z from w

Take Grant Protection Model

Islands and bridges: leakage in TG is decidable in linear time

- need to consider only t,g edges for building the graph
- Lemmas 1, 2 => t v g edge between subjects => full rights exchange

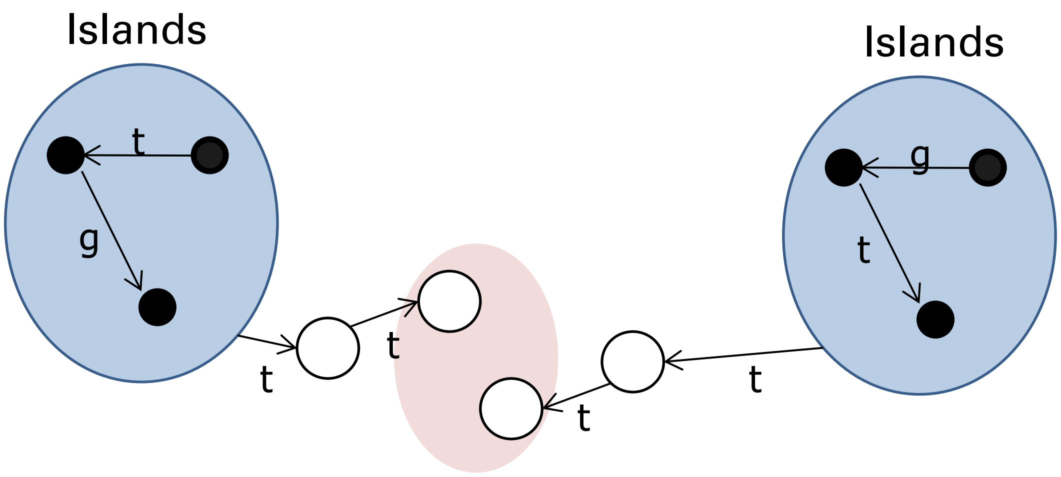




Take Grant Protection Model

Islands and bridges: towards deciding leakage in linear time

- need to consider only t,g edges for building the graph
- Lemmas 1, 2 => t v g edge between subjects => full rights exchange

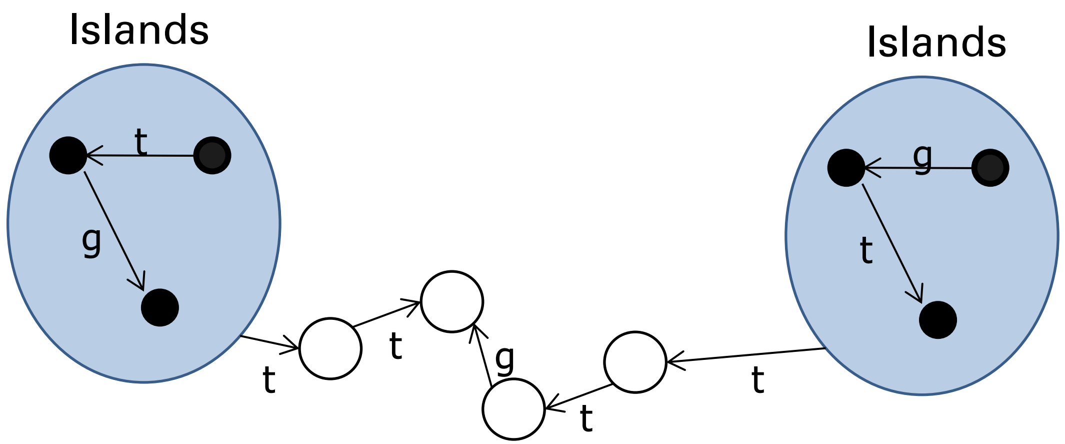


TECHNISCHE UNIVERSITÄT DRESDEN

Take Grant Protection Model

Islands and bridges: towards deciding leakage in linear time

- need to consider only t,g edges for building the graph
- Lemmas 1, 2 => t v g edge between subjects => full rights exchange



Bridge: $\overrightarrow{t}^{\,n}\ \overleftarrow{\overrightarrow{g}}\ \overleftarrow{t}^{\,m}$

Summary

- Certification
  - Assuring system security
- Verification Example
- Security Policies
  - Confidentiality (MLS), Integrity (Biba), mixed (Chinese Wall)
- Policy Enforcement Mechanisms
  - ACLs, Capabilities, Monitors
- Undecidability of Leakage
  - ACM implements turing machine
- Take-Grant Protection Model
  - Leakage is decidable in linear time

References

- B. Lampson: A note on the confinement problem
- **Matt Bishop – Text Book: Computer Security – Art and Science**
- P. Gallagher: A Guide to Understanding the Covert Channel Analysis of Trusted Systems [TCSEC – CC Guide]
- Proctor, Neumann: Architectural Implications of Covert Channels
- Sabelfeld, Myers: Language-based information-flow security
- Karger, Wray: Storage Channels in Disk Arm Optimizations
- Alpern, Schneider 87: Recognizing safety and lifeness
- Alves, Schneider: Enforceable security policies
- Walker, Bauer, Ligatti: More enforcable security policies
- Osvik, Shamir, Tromer: Cache Attacks and Countermeasures: the Case of AES
- Denning 67: A Lattice Model of Secure Information Flow
- Denning: Certification of programs for secure information flow.
- Hunt, Sands: On flow-sensitive security types
- Volpano, Irvine, Smith: A sound type system for secure inform. flow analysis
- Warnier: Statically checking confidentiality via dynamic labels
- Zheng, Myers: End-to-End Availability Policies and Noninterference
- Shapiro, Smith, Farber: EROS: A Fast Capability System
- Klein, Heiser +: seL4: Verifying an Operating System Kernel