School of Computer Science & Engineering

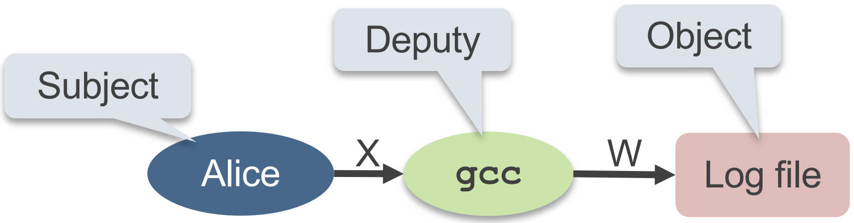
COMP9242 Advanced Operating Systems

2025 T3 Week 07 Part 2

Security Fundamentals

Gernot Heiser

Incorporating material from Toby Murray



Today's Lecture

- Security Intro
- Access-Control Principles
- ACLs vs Capabilities
- Mandatory Access-Control Policies

Computer Security

Protecting *my interests* (that are under computer control) from *threats*

- Inherently subjective
  - Different people have different interests
  - Different people face different threats
- Don't expect one-size-fits-all solutions
  - Grandma doesn't need an air gap
  - Windows insufficient for protecting TOP SECRET (TS) classified data on an Internet-connected machine

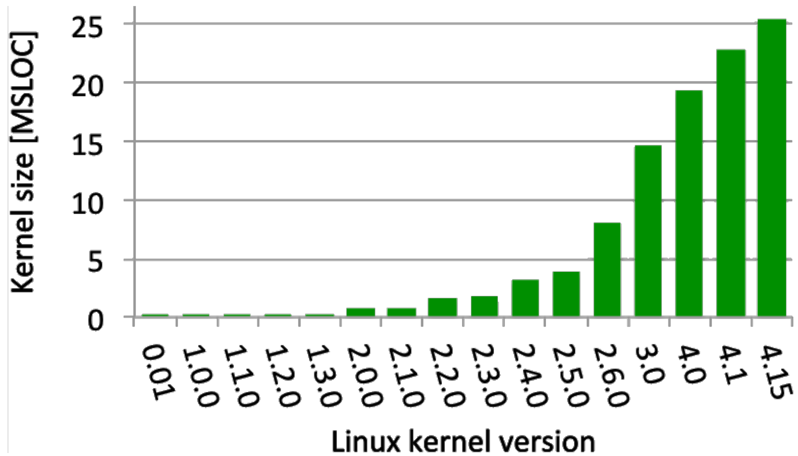
Security claims only make sense
- wrt *defined objectives*
- while *identifying threats*
- and *identifying secure states*

State of OS Security

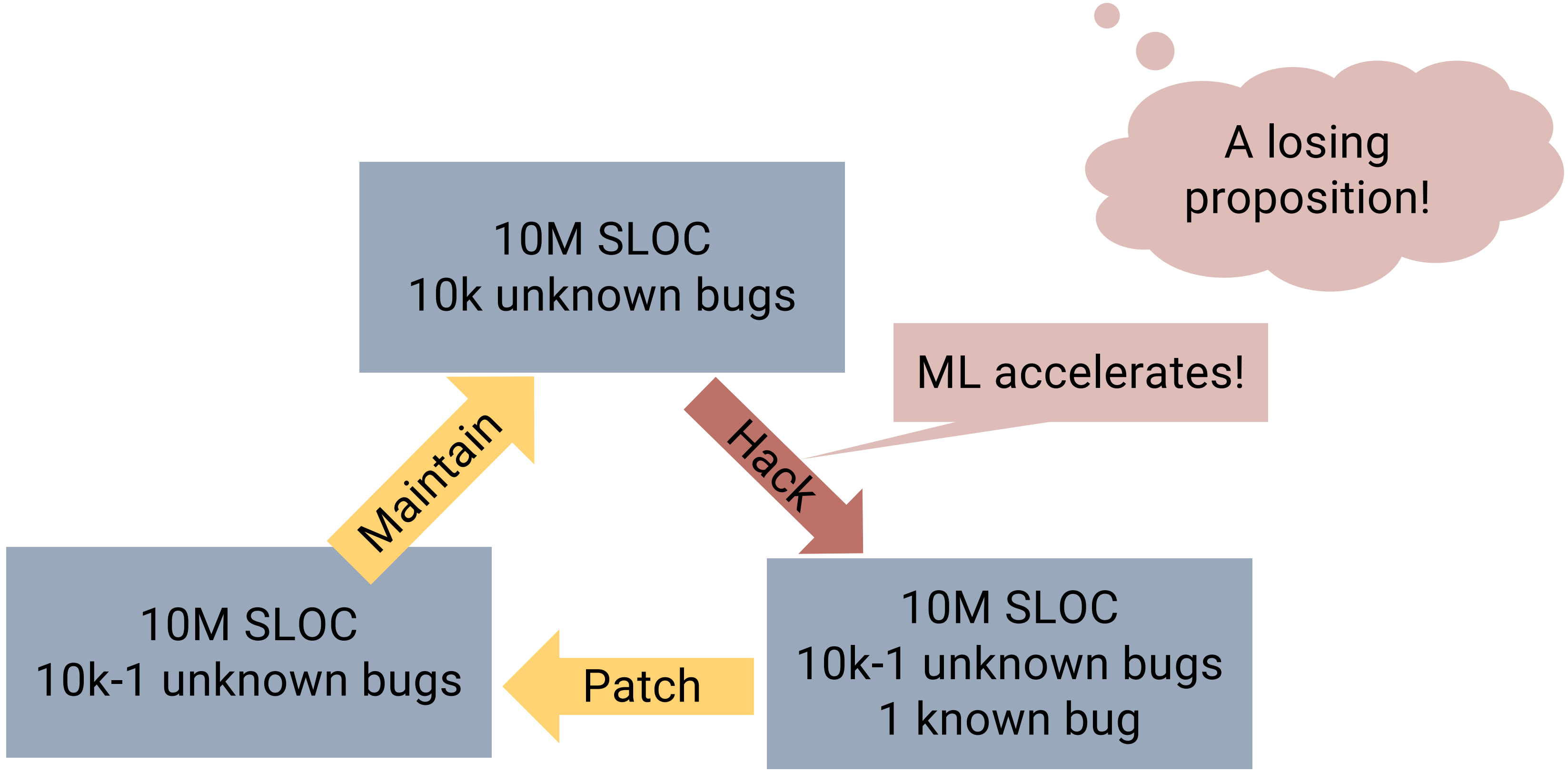
- Traditionally:
  - Has not kept pace with evolving user demographics
    - Focused on e.g. Defence and Enterprise
  - Has not kept pace with evolving threats
    - Much security work is reactive rather than proactive

Some things are getting better:
- more systematic hardening of OSes
- Better security models in smartphones compared to desktops

Other things are getting worse:
- OS kernel sizes keep growing
- Fast growth in attacker capabilities
- Slow growth in defensive capabilities



Standard Approach: Patch-and-Pray



Security Design Principles

Saltzer & Schroeder [SOSP ’73, CACM ’74]

- **Economy of mechanisms** – KISS
- **Fail-safe defaults** – as in any good engineering
- **Complete mediation** – check everything
- **Open design** – no security by obscurity
- **Separation of privilege** – defence in depth
- **Least privilege** – aka *principle of least authority* (POLA)
- **Least common mechanisms** – minimise sharing
- **Psychological acceptability** – if it’s hard to use it won’t be

Mainstream OSes violate most of them! Especially KISS, POLA

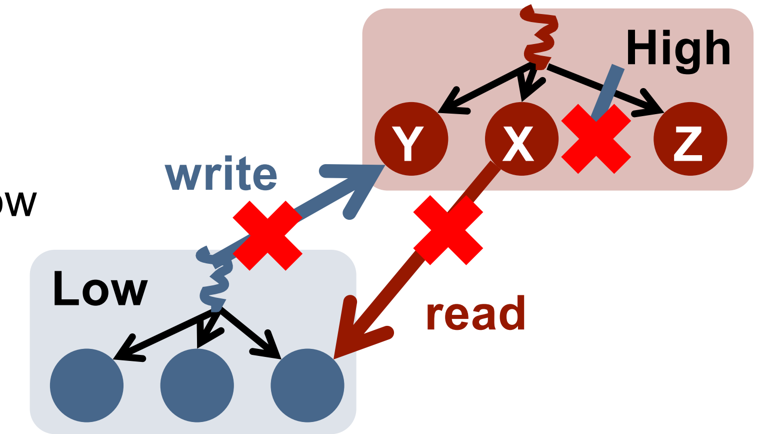
Common OS Security Mechanisms

- Access Control Systems
  - control what each process can access
- Authentication Systems
  - confirm the identity on whose behalf a process is running
- Logging
  - for audit, detection, forensics and recovery
- Filesystem Encryption
- Credential Management
- Automatic Updates

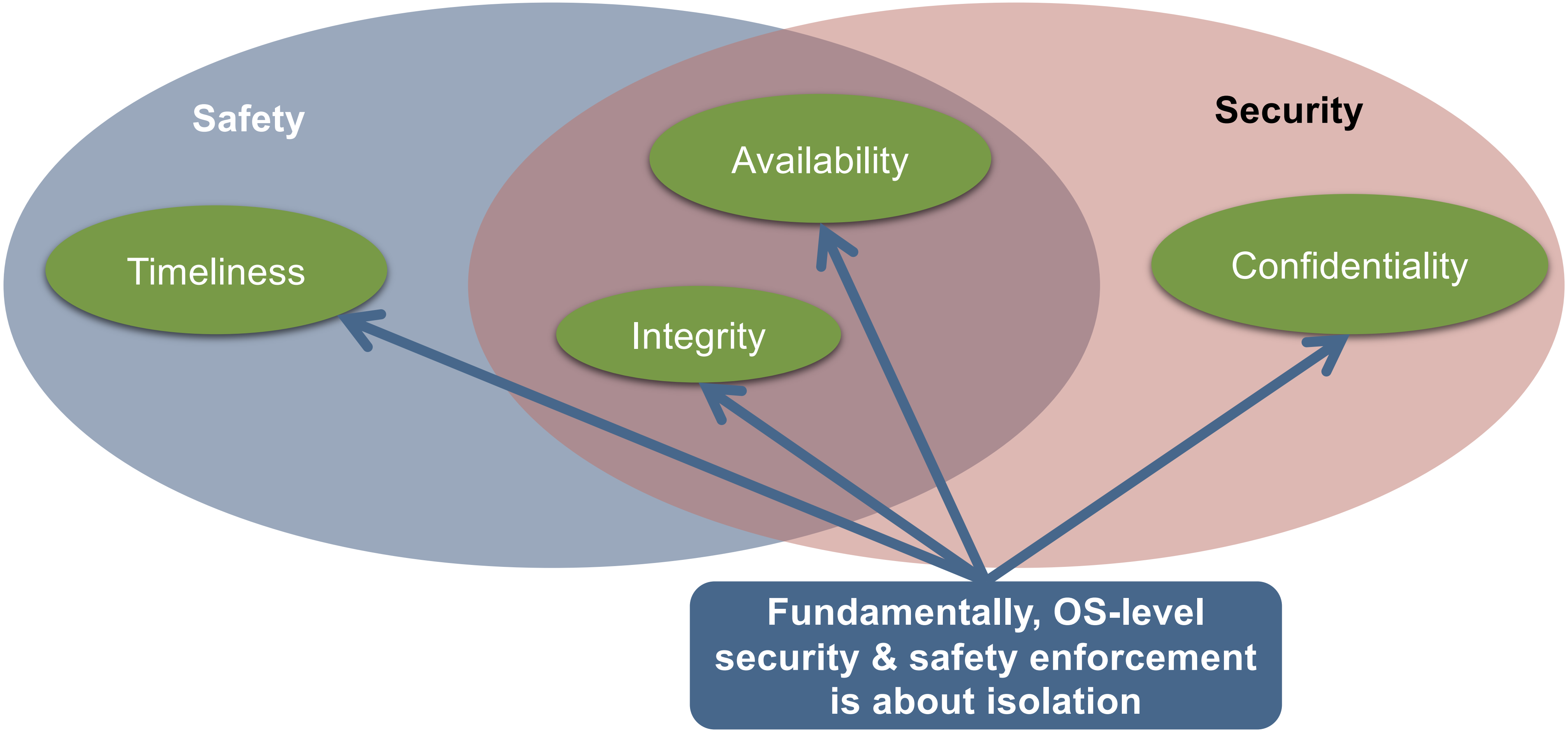
Fundamental mechanism

Security Policies

- Define what should be protected, and from whom
- Often in terms of common security goals (*"CIA properties"*):
  - **Confidentiality**
    - X should not be learnt by Low
  - **Integrity**
    - Y should not be tampered with by Low
  - **Availability**
    - Z should not be made unavailable to High by Low



Security vs Safety



Assumptions

- All policies and mechanisms operate under certain **assumptions**
  - **e.g.** TS-cleared users can be trusted not to write TS data into the UNCLASS window
  - some *trusted* entities behave as expected
- Problem: implicit or poorly understood assumption

Good assumptions are
- *clearly identified*
- *verifiable*!

Trust

- Systems always have **trusted** entites
  - whose misbehaviour can cause insecurity
  - hardware, OS, sysadmin ...

Trusted computing base (TCB):
The set of all trusted entities

- Secure systems require the TCB to be **trustworthy**
  - achieved through **assurance** and **verification**
  - shows that the TCB is unlikely to misbehave

Minimising the TCB is key for ensuring correct behaviour

Assurance and Formal Verification

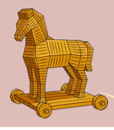
- **Assurance**:
  - systematic evaluation and testing
  - essentially an intensive and onerous form of quality assurance
- **Formal verification**:
  - mathematical proof
- **Certification**: independent examination
  - confirming that the assurance or verification was done right

Assurance and formal verification aim to establish correctness of
- mechanism design
- mechanism implementation

Covert Channels: Bypassing Access Control

User
Sensitive data

Browser tab
Java



Encryption
Server key

Network Driver

Leak

Confinement [Lampson '73]

Violated!

Browser tab
Helper

Encryption
Attacker key

Trojan sends bit:

```
create ("/tmp/covert<bit>");
```

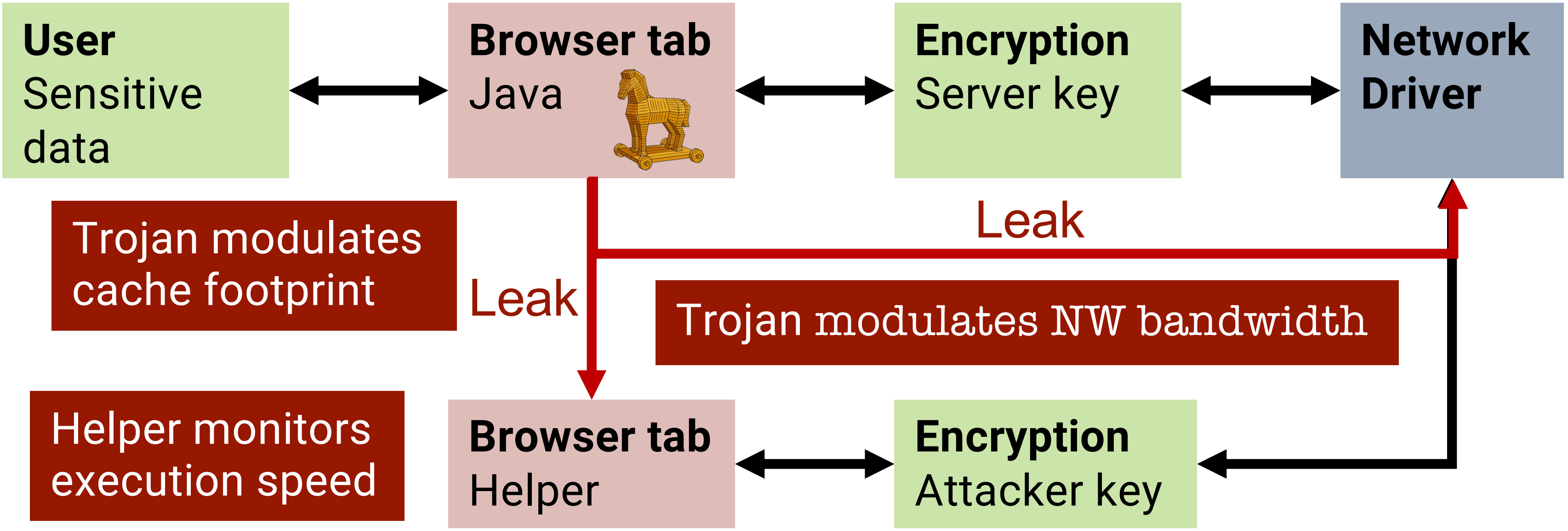
Helper reads bit:

```
if (create ("/tmp/covert0")) bit=0
else if (create ("/tmp/covert1")) bit=1
else retry;
```

Covert storage channel:

- Uses attribute of shared resource
- Controlable by access control

Covert Channels: Bypassing Access Control



Covert timing channel:
- Outside access control
- Very hard to control/analyse!

Covert Timing Channels

- Created by shared resource whose effect on timing can be monitored
  - Cache, network bandwidth, CPU load...
- Requires access to a time source
  - Anything that allows processes to synchronise
  - Generally any relative occurrence of two events

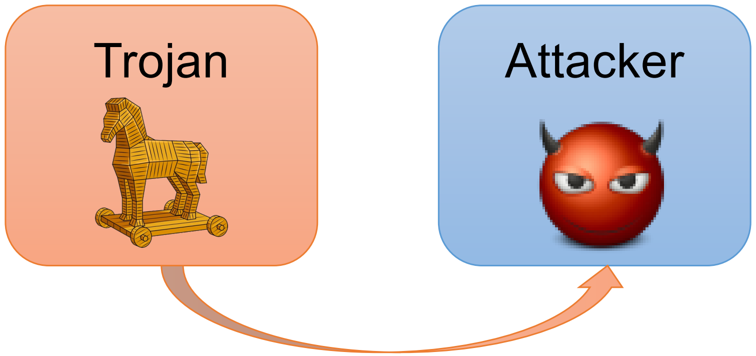
Usually based on some hidden state (e.g. caches)
⇒ timing/storage channel distinction is not deep!

Channel bandwidth may matter: is 1b/s ok?
- Leaking a video vs
- Leaking a server's SSL key

Other physical channels:
- Power
- Temperature (fan speed)
- Electromagnetic emanation
- Acoustic emanation…

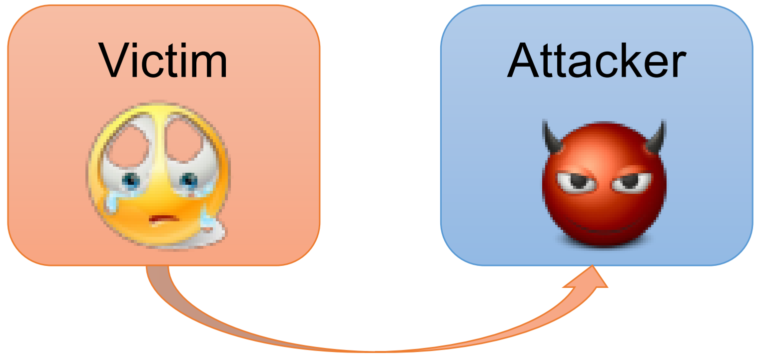
Covert Channels vs Side Channels

Covert Channel



- Trojan intentionally creates signal through targeted resource use
- Worst-case bandwidth

Side Channel



- Attacker uses signal created by victim's innocent operations
- Much lower bandwidth

Summary of Introduction

- Security is very subjective, needs well-defined objectives
- OS security:
  - provide good security **mechanisms**
  - that support users' **policies**
- Security depends on establishing **trustworthiness** of trusted entities
  - **TCB: set of all such entities**
    - **should be as small as possible**
  - Main approaches: assurance and verification

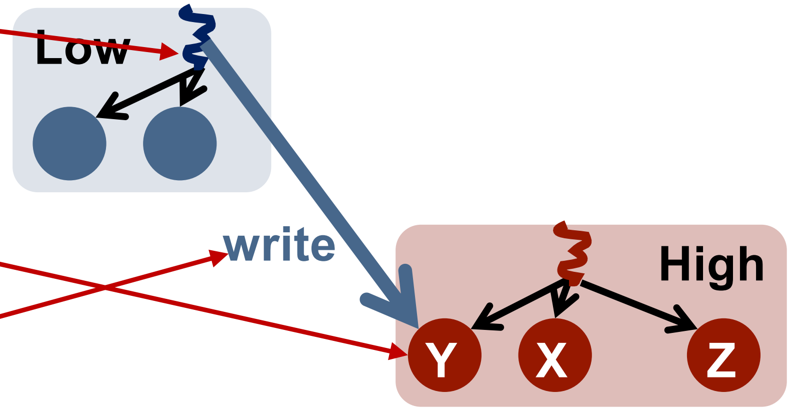
The OS is necessarily part of the TCB

Access-Control Principles

Access Control

Who can access **what** in which **ways**

- The "who" are called **subjects** (or **agents**)
  - e.g. users, processes etc.
- The "what" are called **objects**
  - e.g. individual files, sockets, processes etc.
  - includes all subjects
- The "ways" are called **permissions**
  - e.g. read, write, execute etc.
  - are usually specific to each kind of object
  - include those meta-permissions that allow modification of the protection state
    - e.g. own



Access Control Mechanisms & Policies

- Access Control **Policy**
  - Specifies allowed accesses
  - And how these can change over time
- Access Control **Mechanism**
  - Used to implement the policy

Certain mechanisms led the selves to some policies but not others

Some policies cannot be expressed with the OS's mechanisms!

Protection State: Access-Control Matrix

Defines system's protection state at a particular time instance [Lampson '71]

Subjects are also objects!

	Obj 1	Obj 2	Obj 3	Subj 2
Subj 1	R	RW		send
Subj 2		RX		control
Subj 3	RW		RWX own	recv

Representing Protection State

Storing full matrix is infeasible
- huge but sparse
- highly dynamic

Obj 1

```
Subj1: R
Subj3: RW
```

	Obj 1	**Obj 2**	**Obj 3**	**Subj 2**
Subj 1	R	RW		send
Subj 2		RX		control
Subj 3	RW		RWX own	recv

Columns are access-control lists (ACLs)

Access Control Lists (ACLs)

- Subjects usually aggregated into classes
  - e.g. UNIX: owner, group, everyone
  - more general lists in Windows, recent Linux
  - Can have negative rights
    eg. to overwrite group rights
- Meta-permissions (e.g. own)
  - control class membership
  - allow modifying the ACL

Obj 1

```
Subj1: R
Subj3: RW
```

Used by all mainstream OSes

Representing Protection State

How about rows?

	Obj 1	Obj 2	Obj 3	Subj 2
Subj 1	R	RW		send
Subj 2		RX		control
Subj 3	RW		RWX own	recv

Set of rights a subject has – the subject's *protection domain*

"Object capability"

Subj 3

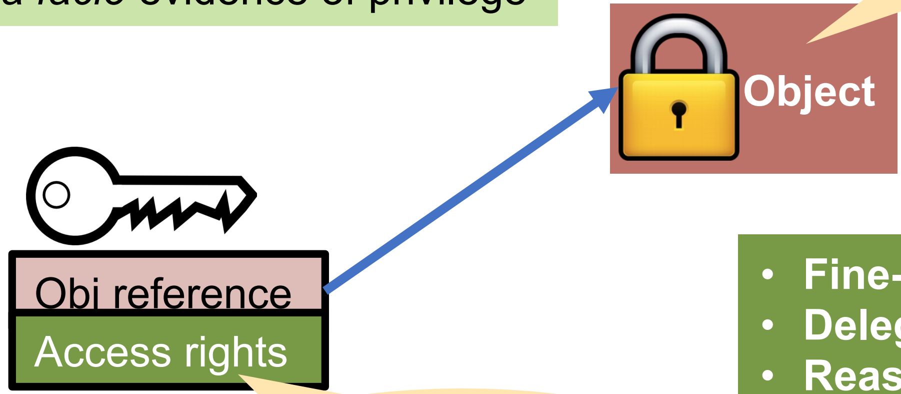
```
Obj1:   RW
Obj3:   RWX, own
Subj2:  recv
```

Represented as a *capability list* (Clist)

(Object) Capabilities (aka Ocaps, Caps)

Capability = Access Token
Prima facie evidence of privilege

E.g. thread,
address space



E.g. read, write,
send, execute…

Linux "capabilities" do not have these properties (not *object* capabilities)!

- **Fine-grained access control**
- **Delegation of rights**
- **Reasoning about information flow**

Any operation is invoking a capability:

```
err = cap.method( args );
```

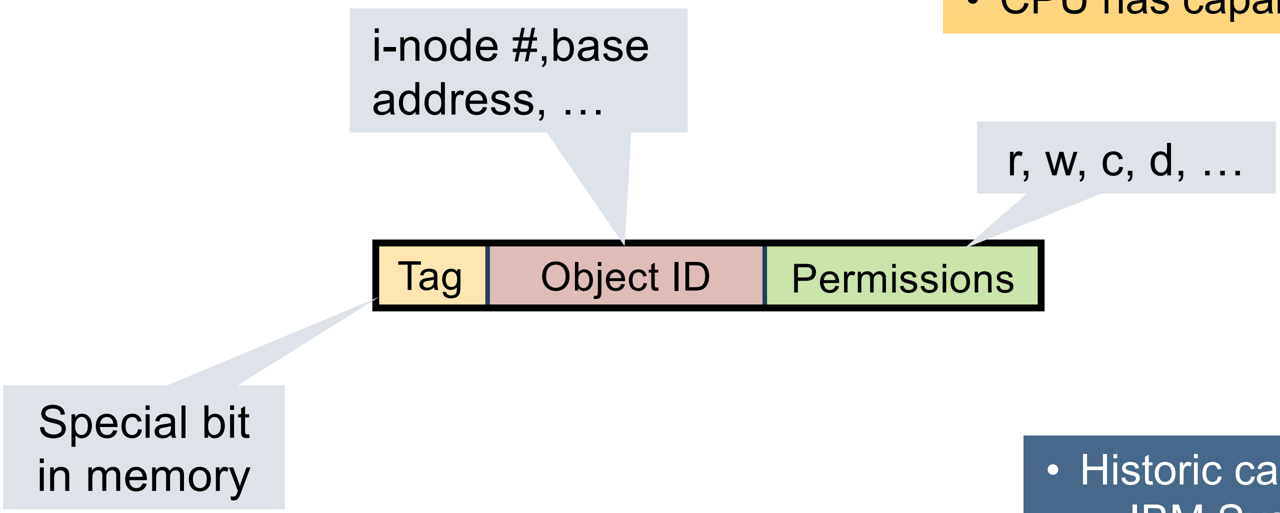
Used in very few commercial systems:

- IBM System/38→AS/400→i-Series
- KeyKOS [Bomberger et al, 1992]
- L4 microkernels, Google Fuchsia

Implementing Ocaps: Hardware

- Cap can be copied like data ⇒ “delegation”
- Tag is reset when modifying word
- CPU has capability registers

Revocation is hard!



- Historic capability machines:
  - IBM System/38 → AS/400 → i-Series
  - Hydra
- Revived with CHERI

Implementing Ocaps: Software-Usermode

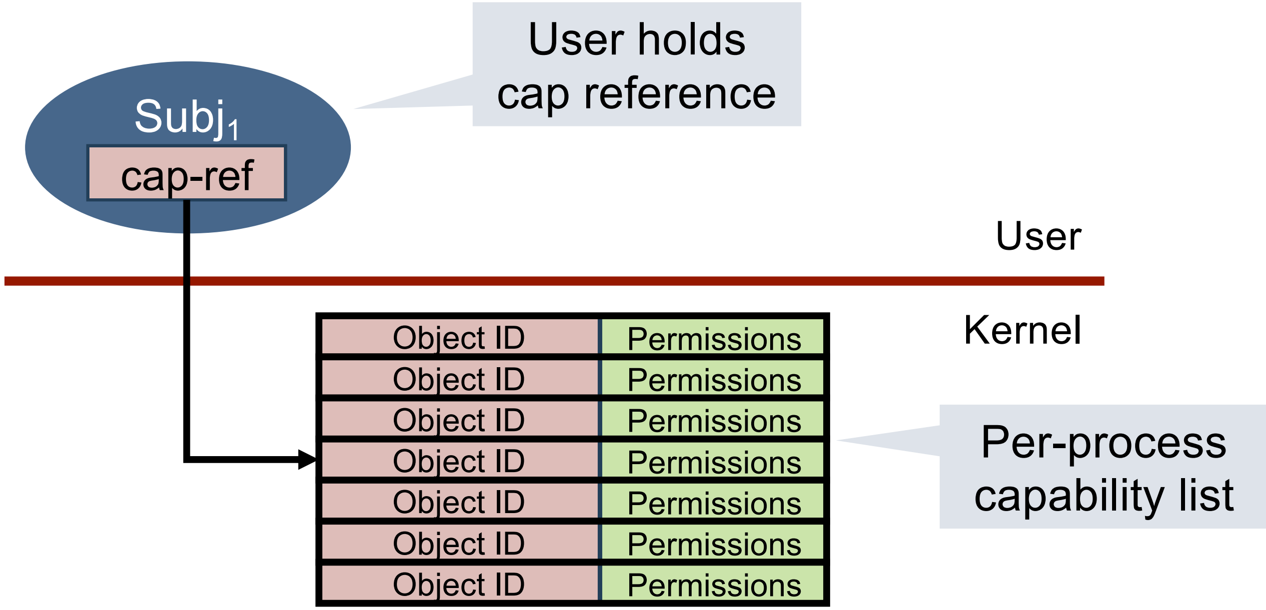
Object ID	Signature

Revocation is hard!

- "Sparse capabilities"
- Cap can be copied like data ⇒ delegation
- Signature mismatch when modifying cap
- OS has object table,
  holds signatures and permissions

- Amoeba [Tanenbaum '81]
- Monash *Password Capability System* [Anderson '86]
- Mungi [Heiser '98]

Implementing Ocaps: Software-Kernel



- "Partitioned" or "segregated" caps
- Delegation is system call
- Revocation is easy

- Mach [Accetta '86]
- EROS [Shapiro '99]
- Modern L4 kernels
- Unix file descriptors

ACLs vs Capabilities

ACLs & Object Capabilities – Duals?

- In theory dual representations of access control matrix
- Practical differences:
  - Naming and namespaces
    - Ambient authority
    - Deputies
  - Evolution of protection state
  - Process creation
  - Auditing of protection state

Duals: Naming and Name Spaces

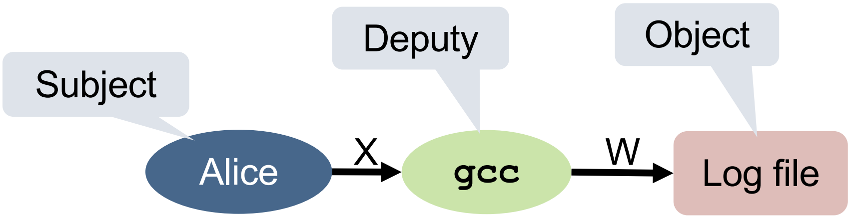
- ACLs:
  - objects referenced by **name**
    - requires separate (global) name space
    - e.g. open(“/etc/passwd”,O_RDONLY)
  - require a subject (class) namespace
    - e.g. UNIX users and groups
- Capabilities:
  - objects referenced by **capability**
  - no further namespace required –
    object discovery orthogonal to access control
  - cannot even *name* object without access

Covert storage channel?

Least common
mechanisms!

Duals: Confused Deputy

Unix:
- Log file is group **`admin`**
- Alice not member of **`admin`**
- gcc is set-GID **`admin`**




```
alice$ gcc -o Log_file source.c
```

```
static char* log = "/var/gcc/log";
int gcc (char *src, *dest) {
    int s = open (src, RDONLY );
    int l = open (log, APPEND);
    int d = open (dest, WRONLY);
     …
    write (dest, …);
}
```

Clobber log!

- ACLs separate naming and permissions
- Deputy depends on *ambient authority*: Uses own privileges for access

Confused deputy is inherent problem of ACLs!

Deputy With Ocaps

Object capability (Ocaps) system:
- gcc holds **`w`** cap for log file
- Alice holds **`r`** cap for source, **`w`** cap for destination
- Alice holds no cap for log file

```
alice$ gcc -o Log_file source.c
```

Invalid cap

```
static cap_t log = <cap>;
int gcc (cap_t src, dest) {
    fd_t s = open (src, RDONLY );
    fd_t l = open (log, APPEND);
    df_t d = open (dest, WRONLY);
    …
    write (d, …);
}
```

Open fails!

- Caps are both names and permissions
- Presentation is explicit, not ambient
- Can't name object if don't have access!

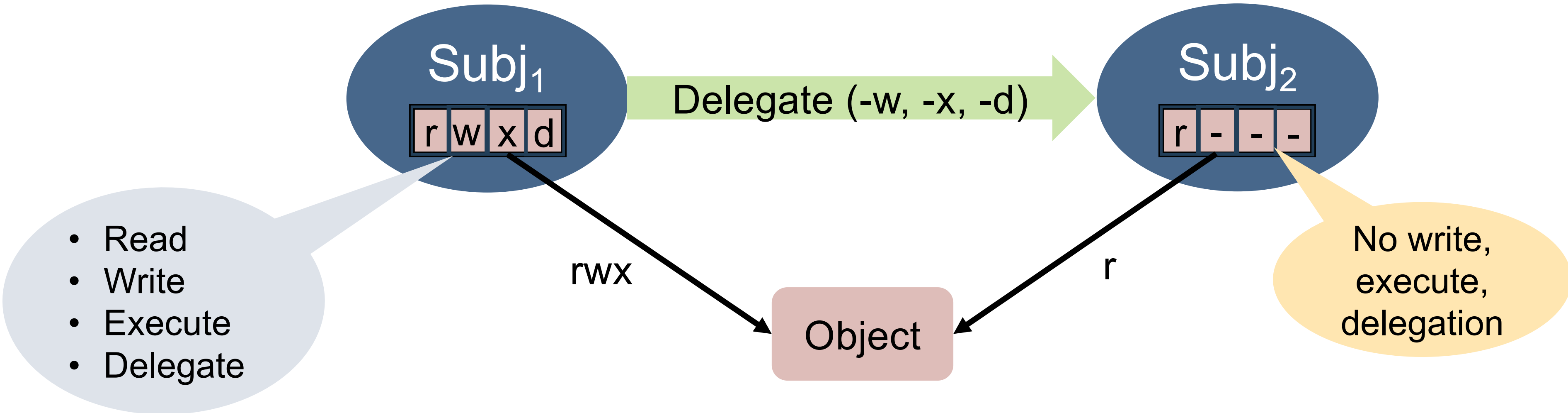
Linux "capabilities" don't help!

Duals: Evolution of Protection State

ACLs: Protection state changes by modifying ACLs

- Requires certain meta-permissions on the ACL

Capabilities: Protection state changes by delegating and revoking caps



Duals: Evolution of Protection State

ACLs: Protection state changes by modifying ACLs

- Requires certain meta-permissions on the ACL

Capabilities: Protection state changes by delegating and revoking caps

Caps support reasoning about information flow:
- A can send message to B only if A holds cap to B
- A can obtain access to C only if it receives message with cap to C

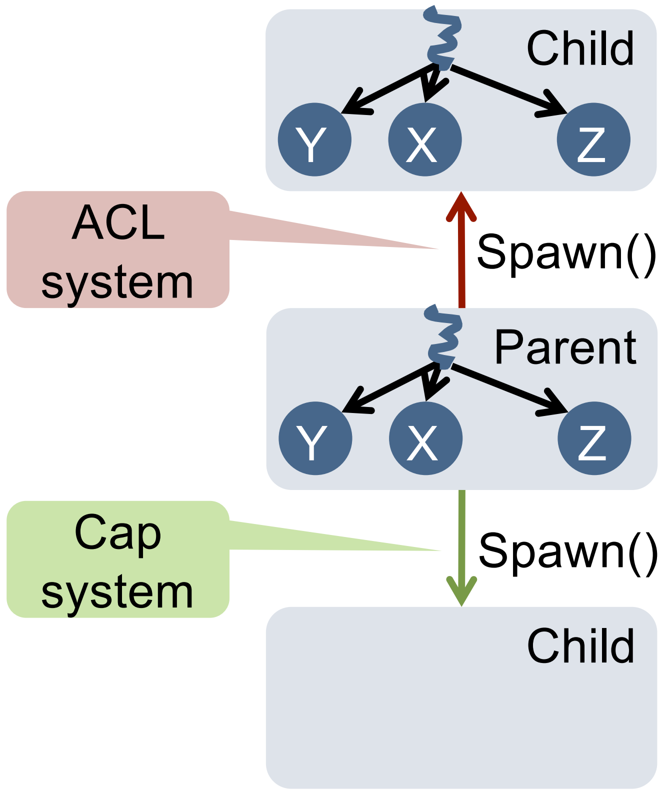
Duals: Process Creation – Child Permissions

ACLs:
- Permissions defined by child's subject
- Generally inherit all of parent's rights

Max privilege

Caps :
- Child is created with no rights
- Parent gets cap to child,
  can delegate as needed

Least privilege



Duals: Auditing of Protection State

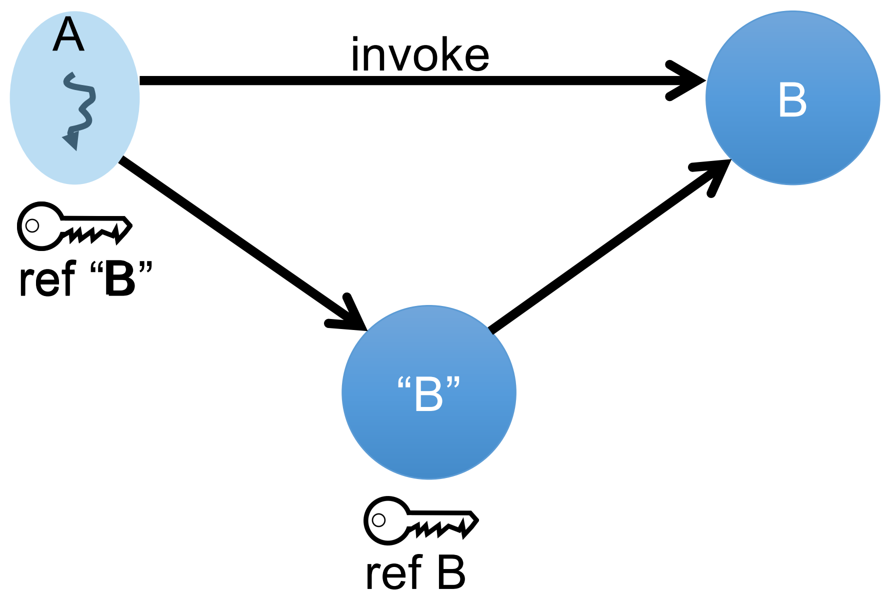
- Who has permission to access a particular object (right now)?
  - ACLs: Just look at the ACL
  - Caps:
    - sparse or tagged caps: hard/impossible without full memory scan
    - partitioned caps: doable (maintain derivation tree)
- What objects can a particular subject access (right now)?
  - Capabilities: Just look at its capabilities
  - ACLs: may be impossible to determine without full scan

"Who can access my stuff?"
vs
"How much damage can X do?"

Interposing Access

Caps are opaque object references (pure names)

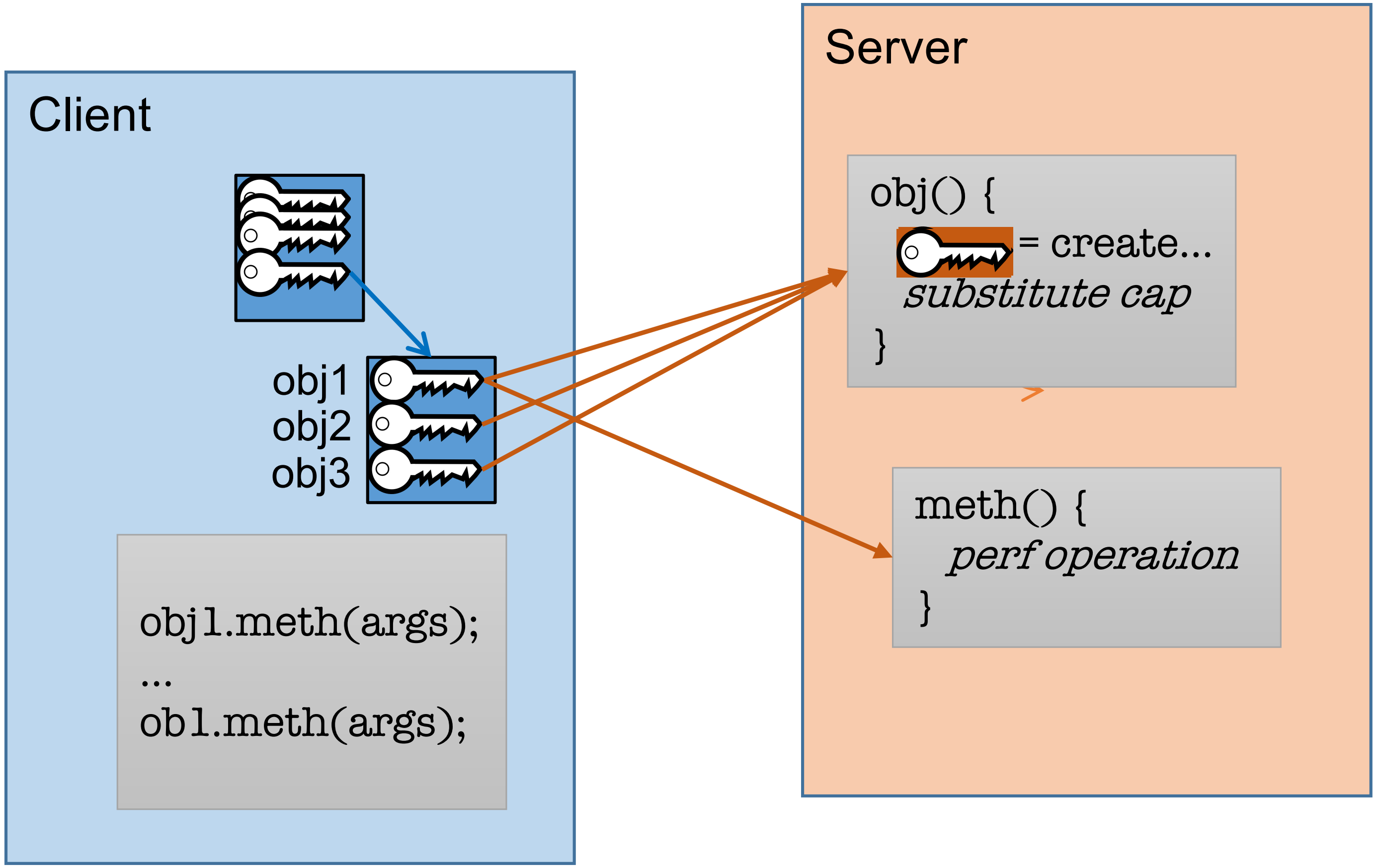
- Holder cannot tell which object a cap references nor the authority
- Supports transparent interposition (virtualisation)



Usage:

- API virtualisation
- Reference (security) monitor
  - Security policy enforcement
  - Info flow tracing
  - Packet filtering…
- Secure logging
- Debugging
- Lazy object creation

Example: Lazy Object Construction



Duals: Satzer & Schroeder Principles

Security Principle	ACLs	Capabilities
Economy of Mechanism	Dubious	Yes!
Fail-safe defaults	Generally not	Yes!
Complete mediation	Yes (if properly done)	Yes (if properly done)
Open design	Neutral	Neutral
Separation of privilege	No	Doable
Least privilege	No	Yes
Least common mechanism	No	Yes, but…
Psychological acceptability	Neutral	Neutral

Mandatory AC Policies

Mandatory vs Discretionary Access Control

Discretionary Access Control (DAC):

- Users can make access control decisions
  - Delegate their access to other users etc.

Can I stop my browser leaking secrets?

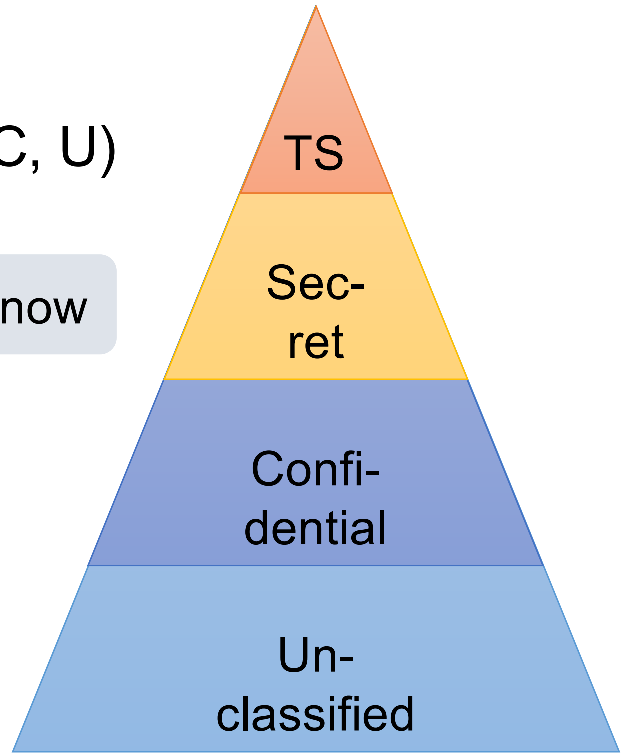
Mandatory Access Control (MAC):

- System enforces administrator-defined policy
- Users can only make access control decisions subject to mandatory policy
- Can limit damage caused by untrusted applications
- Traditionally used in national security environments

MAC: Bell & LaPadula (BLP) Model [1966]

- MAC Policy/Mechanism
  - Formalises national security classifications
- Every object assigned a **classification** (eg TS, S, C, U)
  - orthogonal security **compartments**
- Classifications ordered in a **lattice**
  - e.g. TS > S > C > U
- Every subject assigned a **clearance**
  - Highest classification it's allowed to learn

Need-to-know



Labelled security:
- Subjects and objects are *labelled*
- Permitted accesses: relation over labels
  `allow(subject.label, object.label, operation)`

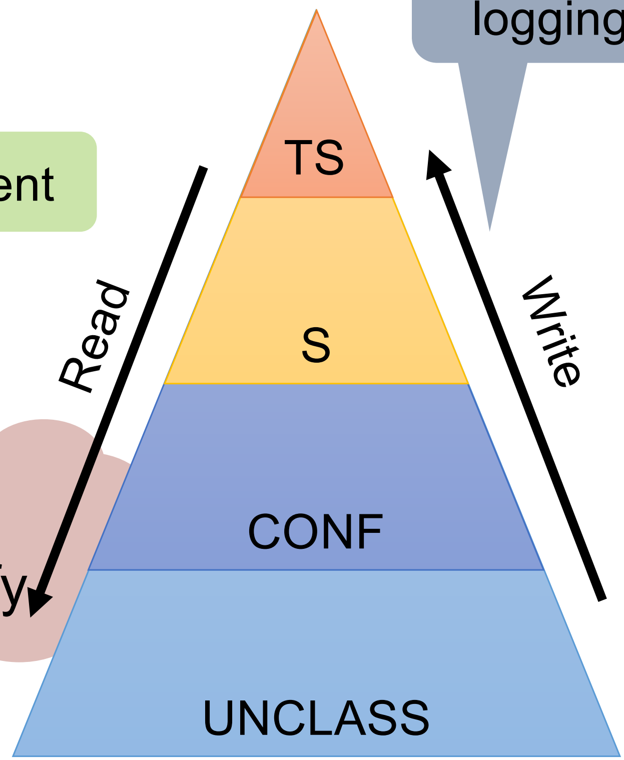
BLP: Rules

- **Simple Security Property** (“no read up”):
  - **s** can read **o** iff `clearance(s) >= class(o)`
  - S-cleared subject can read U,C,S but not TS
  - standard confidentiality
- **★-Property** (“no write down”):
  - **s** can write **o** iff `clearance(s) <= class(o)`
  - S-cleared subject can write TS,S, but not C,U
  - to prevent accidental or malicious leakage of data to lower levels
- In practice need exceptions
  - allow *trusted* entity to write down
  - *“de-classify/downgrade”* – e.g. encryption

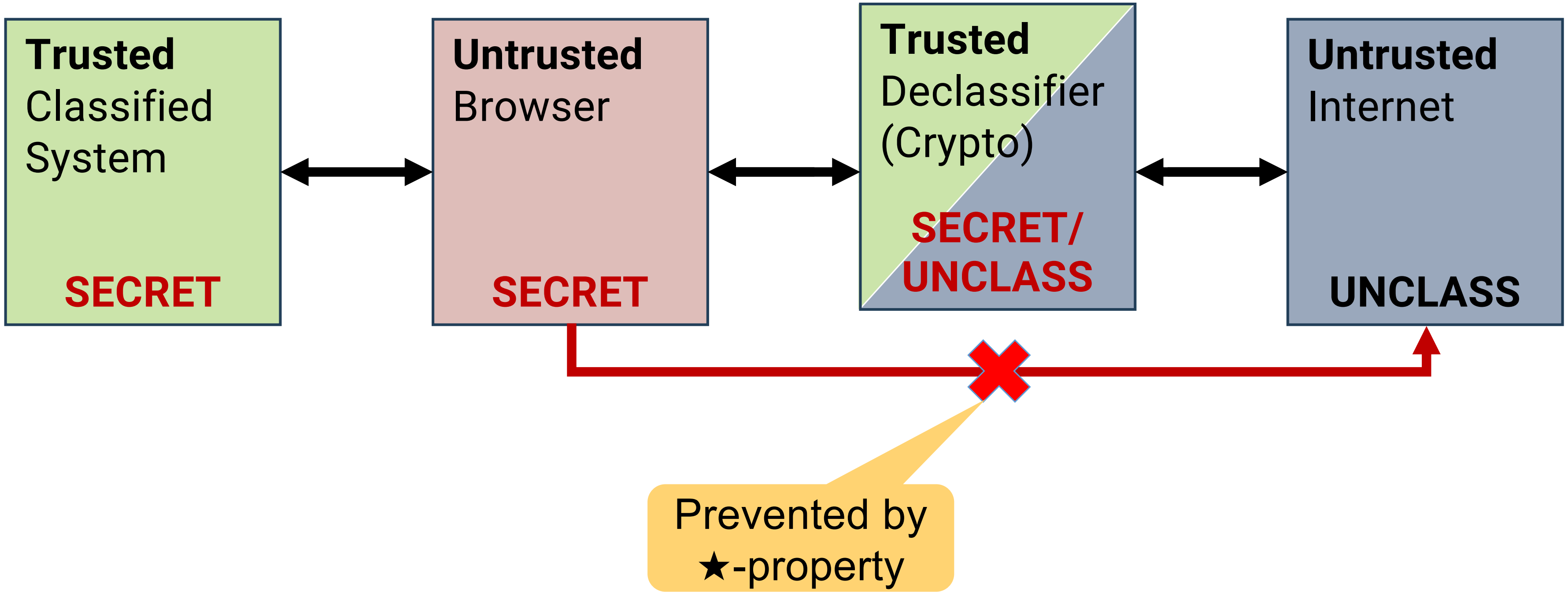
Confinement

Trend to over-classify

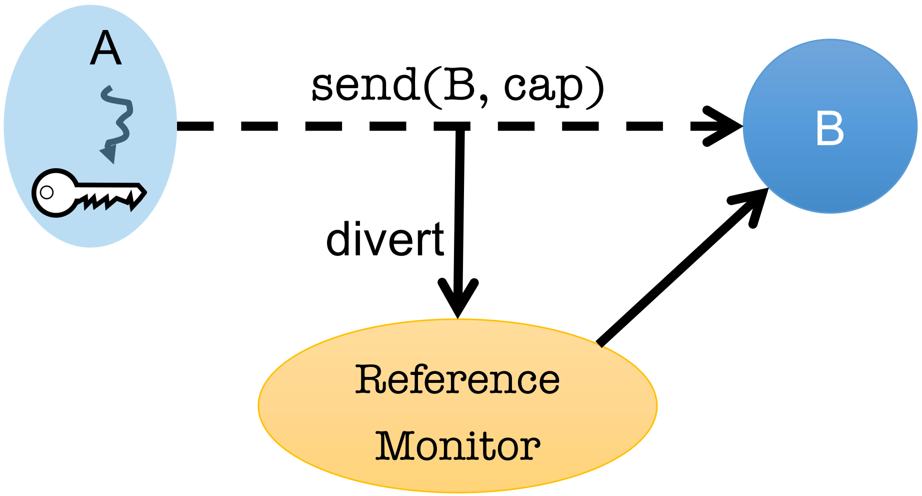
E.g. logging



Web Browser Example



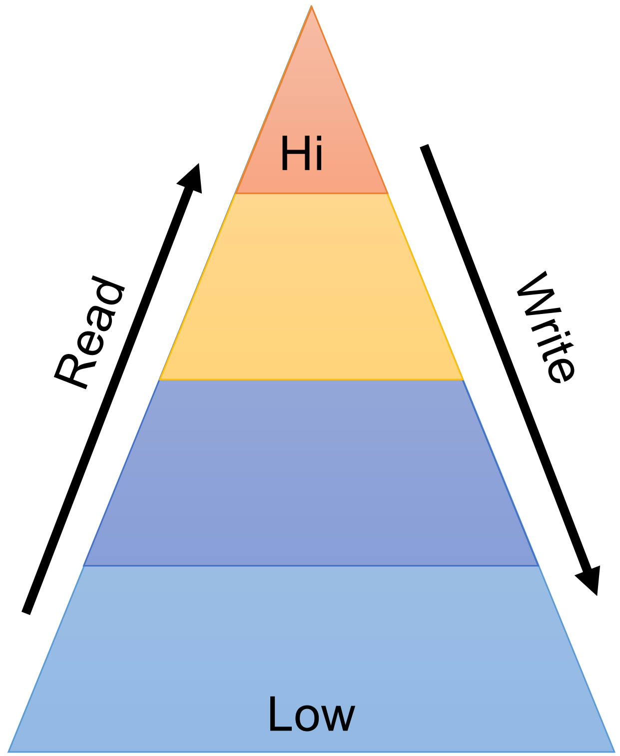
MAC With Caps: Reference Monitor




```
interpose_transfer(cap) {
  if (A.clear > B.clear) {
    c = mint(cap, -r);
    send(B,c);
  } else if (A.clear < B.clear) {
    c = mint(cap, -w);
    send(B,c);
  } else {
    send(B,cap);
  }
}
```

MAC: Biba Integrity Model

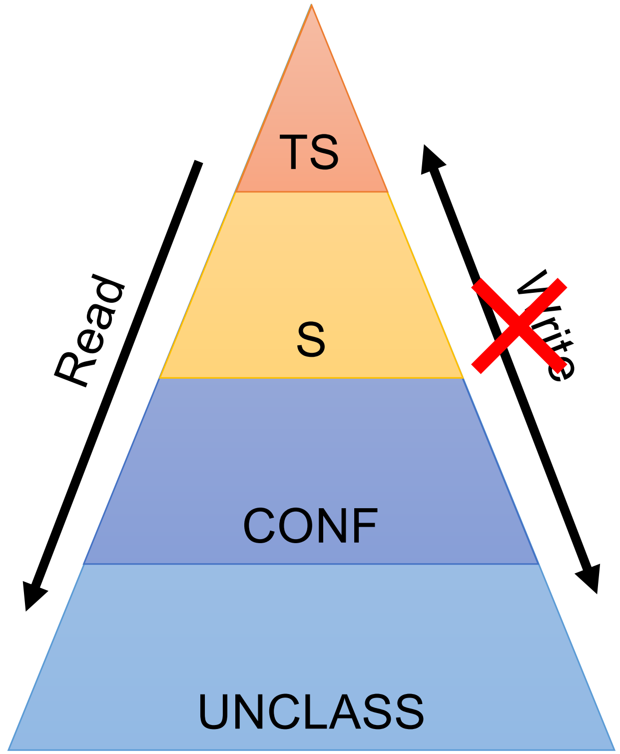
- Bell-LaPadula enforces **confidentiality**
- **Biba:** Its dual, enforces **integrity**
- Objects now carry **integrity** classification
- Subjects labelled by **lowest** level of data each subject is allowed to learn
- BLP order is inverted:
  - **`s`** can read **`o`** iff `clearance(s) <= class(o)`
  - **`s`** can write **`o`** iff `clearance(s) >= class(o)`



UNSW SYDNEY

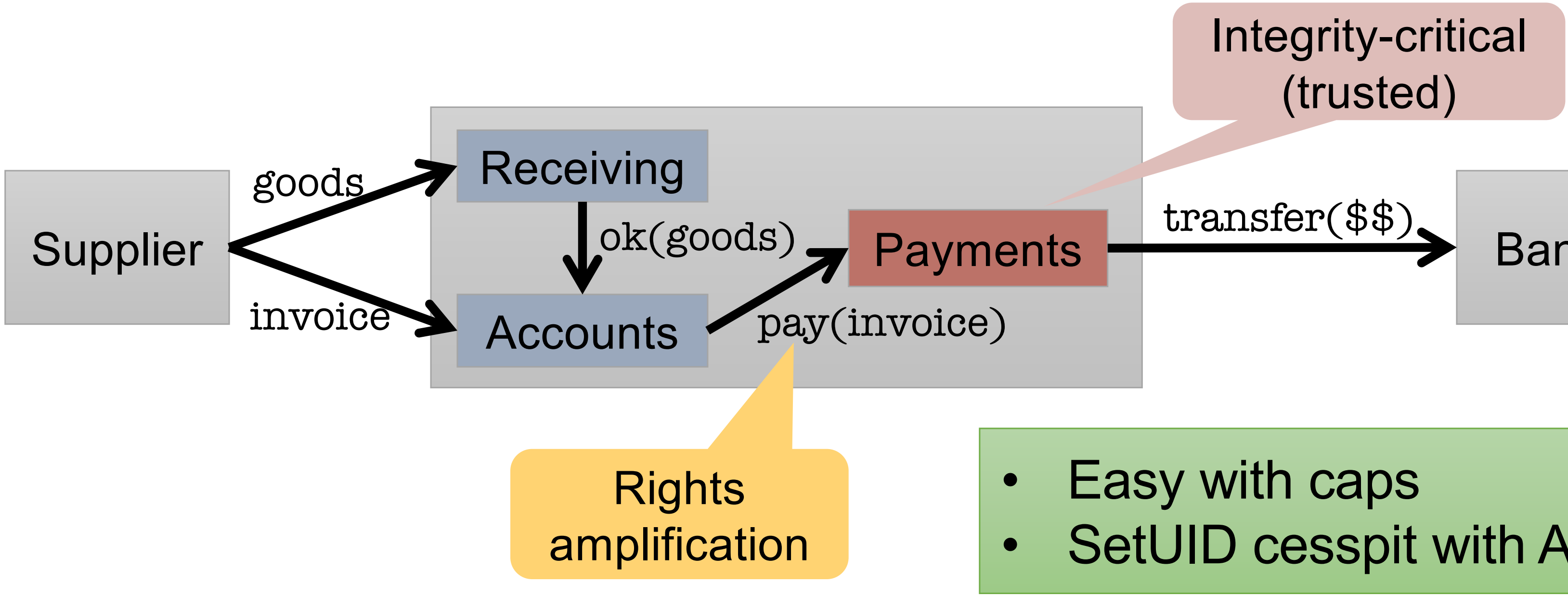
Confidentiality + Integrity

- BLP+Biba allows no information flow across classes
- Practicality requires weakening
  - Assume high-classified subject to treat low-integrity info responsibly
  - Allow read-down
  - Independent labelling for confidentiality and integrity
- **Strong *-Property** ("matching writes only"):
  - **s** can write **o** iff `clearance(s) = class(o)`
  - Eg for logging, high reads low data and logs



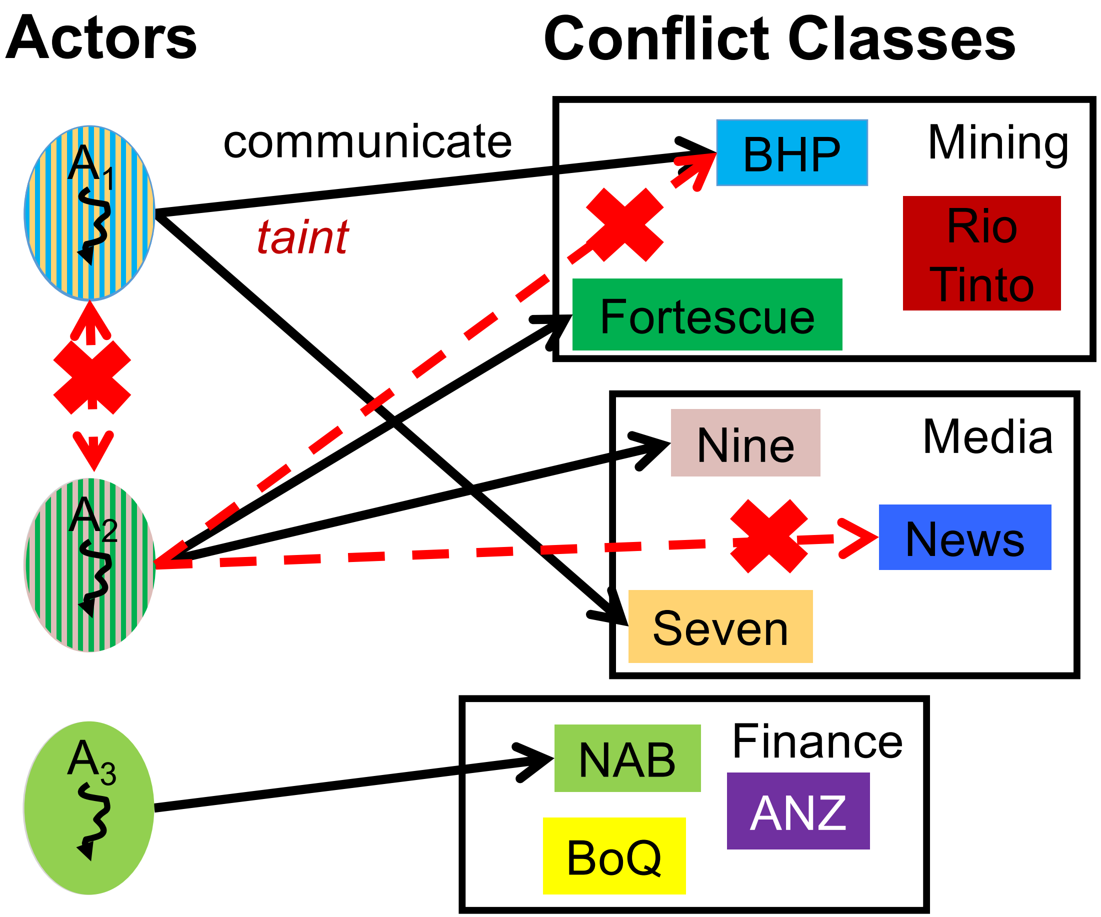
Clark & Wilson Model

- In commercial settings integrity is more important than confidentiality
- Restrict possible operations to *well-formed transactions*
  - eg payment issued only after goods and invoice received



- Easy with caps
- SetUID cesspit with ACLs

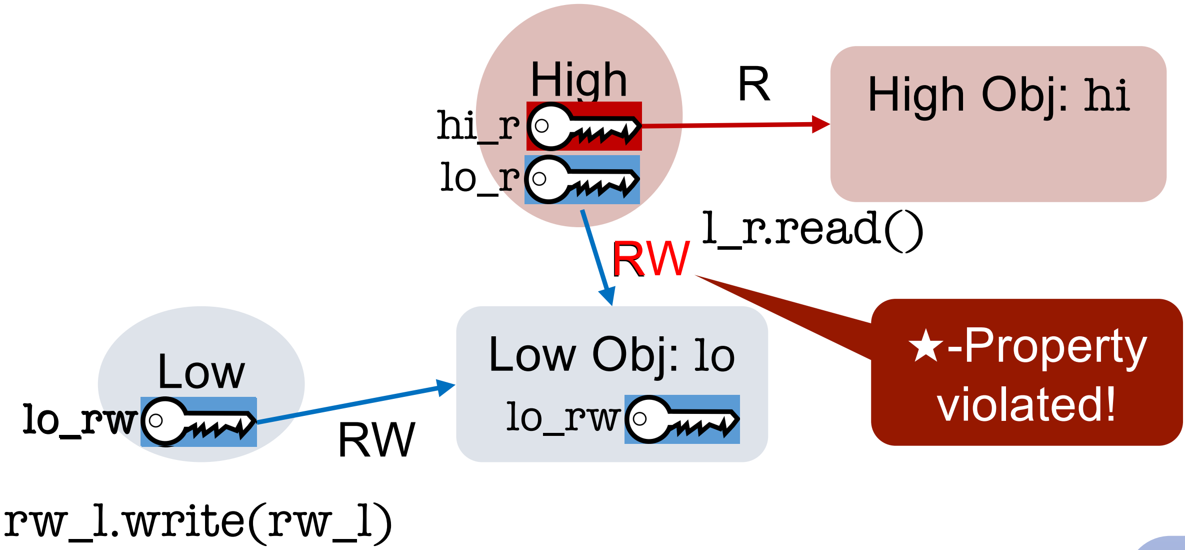
Chinese Wall (aka Brewer & Nash) Model



Dynamic policy:
- Communication *taints* actor
- tainted actor cannot communicate with other entity in same conflict class

Conflict-of-interest prevention
- Law practices
- Consultancies

Boebert's Attack on Capability Machines



Takeaway:

- Need mechanism to limit cap propagation: *take-grant model*
- seL4 *grant* right on endpoints

"On the inability of an unmodified capability machine to enforce the ★-property" [Boebert'84]

Works where caps are indistinguishable from data (HW & sparse caps)

Decidability

Safety: Given initial *safe state s*, system will never reach *unsafe state s'*

Decidability: AC system is decidable if safety can aways be computationally determined

Equivalent to halting problem [Harrison, Ruzzo, Ullman '75]

- Most capability systems are decidable
- Unclear for many common ACL systems

Summary: AC Principles

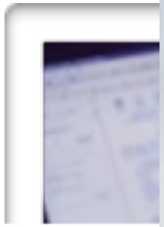
- ACLs and Capabilities:
  - Capabilities tend to better support least privilege
  - But ACLs can be better for auditing
- MAC good for global security requirements
- Not all mechanisms can enforce all policies
  - e.g. ★-property with sparse or HW capabilities
- AC systems should be decidable so we can reason about security

Reminders

- Honours theses at Trustworthy Systems
  https://trustworthy.systems/students/theses
- **John Lions CS Honours Award for thesis in OS**
  https://www.scholarships.unsw.edu.au/scholarships/id/1757
  Deadline: 25 November for T1/25!

What is Security?

Different things to different people:



By Laura Poitras / Praxis Films, CC BY 3.0

On June 8, as the investigation into the initial intrusion p… response team shared with relevant agencies that there was a high degree of confidence that OPM systems containing information related to the background investigations of current, former, and prospective Federal government employees, and those for whom a Federal background investigation was conducted, may have been compromised.

Good Security Mechanisms

- Are widely applicable
- Support general security principles
- Are easy to use correctly and securely
- Do not hinder non-security priorities (e.g. productivity, generativity)
  - Principle of “do not pay for what you don’t need”

Good mechanisms lend themselves to correct implementation and *verification*!

OS Security

- What is the role of the OS for security?
- Minimum:
  - provide **mechanisms** to allow the construction of secure systems
  - that are capable of securely implementing the intended users'/administrators' **policies**
  - while ensuring these mechanisms cannot be subverted

Policy vs Mechanism

- Policies accompany mechanisms:
  - **access control** policy
    - who can access what?
  - **authentication** policy
    - is password sufficient to authenticate TS access?
- Policy often restricts the applicable mechanisms
- One person's policy is another's mechanism

Risk Management

- Comes down to **risk management**
  - There is no absolute security, what risks we are willing to tolerate?
  - Cost & likelihood of violation vs. cost of prevention
  - Gain vs cost for attacker
- Actions:
  - mitigate – using security mechanisms
  - transfer – e.g. by buying insurance

Good security policy will identify appropriate action, based on risk assessment