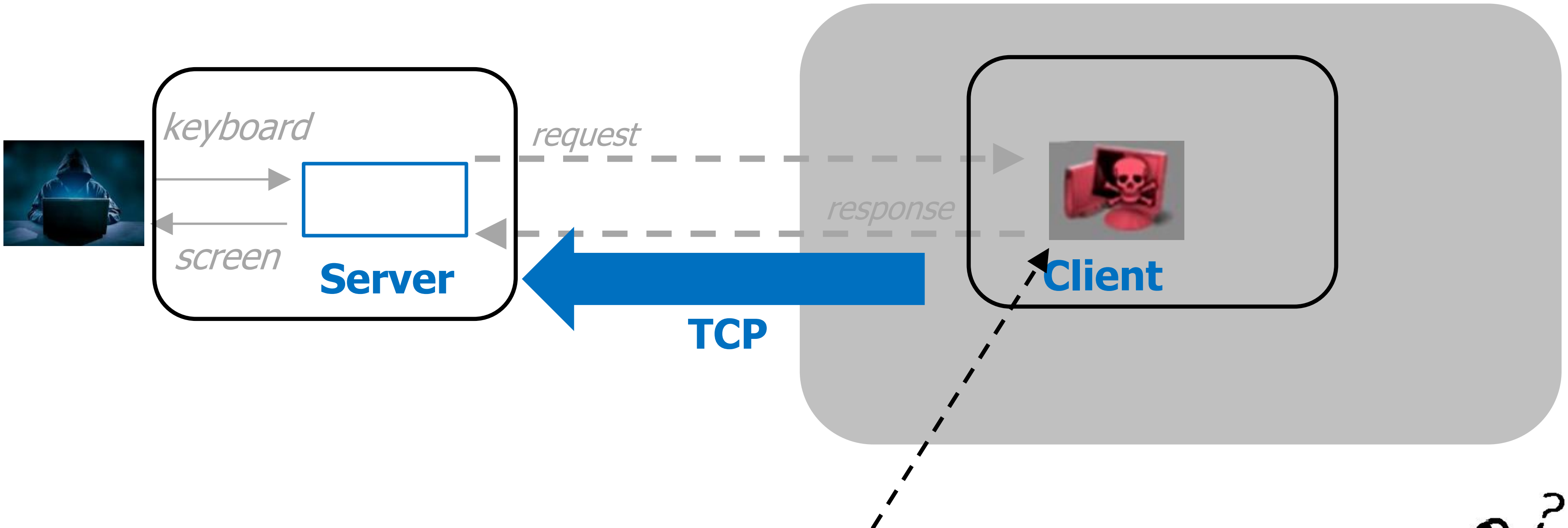
Security Policy

REVERSE Shell (REMIND)



- **What can it do, exactly?**
- Can it kill other processes?
- Can it read the memory of other processes?
- Can it modify the autostart/bootstrap configuration?

Important question (I)

- PC
- Dropbox app
- Chrome browser

- Can the Dropbox app **read**:
  - ...authentication cookies?
  - ...passwords stored in the browser?
  - ...encryption keys in the browser memory?

Important question (II)

- PC
- Macro in Excel, downloaded as an email attachment
- Chrome browser

- Can the Excel Macro app **read**:
  - ...authentication cookies?
  - ...passwords stored in the browser?
  - ...encryption keys in the browser memory?

Important question (III)

- Smartphone
- Gaming app
- Banking app

- Can the Gaming app **read**:
  - ...authentication token of Banking app?

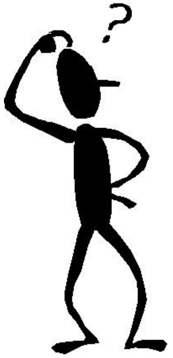
Security Policy

- **Set of rules** that determine "**who can do what**"
- **Every system** has one, **explicit** or **implicit**
  - Usually implicit

- We need to **understand** how these rules are **structured** in practice

Important question (IV)

- PC
- User U executes GUI / Shell
- How can you make sure that the GUI / Shell can **only** execute operations **allowed to U**?

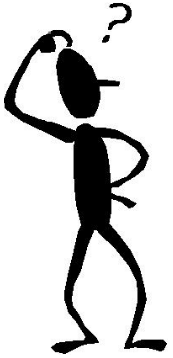


Important question (V)

- PC
- User U executes some program P
- How can you make sure that P cannot **modify** the internal code/data **of the o.s.**?

Important question (VI)

- Web server
- User U logged on a **webapp** (e.g., Banking)
- How can you make sure that U can **only** access **"his/her" data**?



Access Control (I)

- We need to **understand** how these rules are structured in practice
- And how they are **enforced**

Access Control (II)

- We need to understand how these rules are **structured** in practice
- And how they are **enforced**

- **Fundamental** problem at **every abstraction level**

- Application
  - Client→ Server resources (web documents, mailboxes, ...)
- Operating system
  - Process→ O.S. resources (files, network, ...)
- Hardware
  - CPU → Memory addresses

Roadmap

1. Access control in operating systems
2. How **enforced**
3. How **described**, in an **idealized** way
4. How **described**, in a **more realistic** way

5. **Fundamental lessons**
6. Access control "**in general**"

- Very simplified (many details omitted)

Access Control: Preliminaries

Account and Resources

- **Account**: Every **identity** in the system
  - **Username** (string)

- **Resource**: Every **"IT object"** in the system
  - File / Socket / ...
  - Server configuration / ...
  - Account attributes / Account configuration / ...
  - ...

- Accounts are often called "Users"...which may be **misleading**: certain accounts are **not** meant to be owned by a human operator

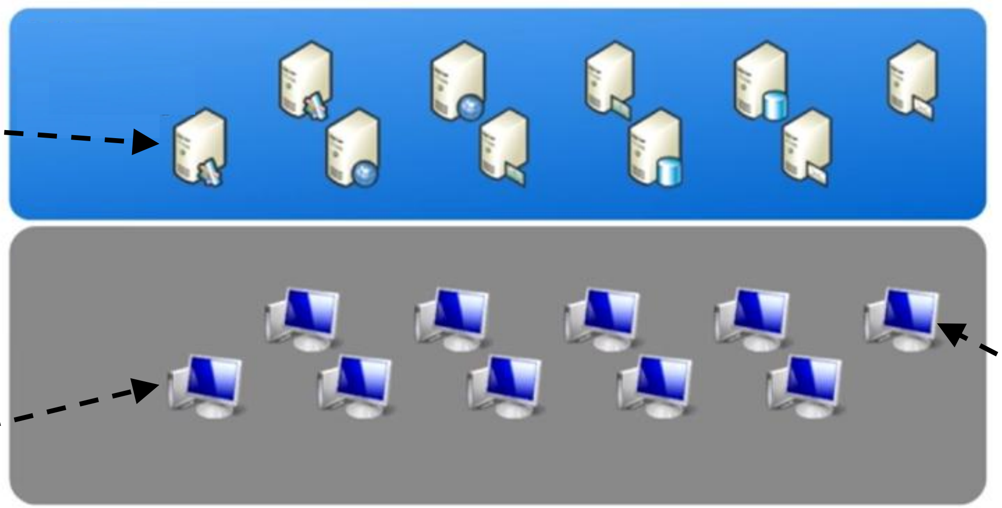
Security Policy

- **Account**: Every **identity** in the system
- **Resource**: Every **"IT object"** in the system

- For each <Account, Resource> which **Operations** are allowed

Example Scenario (I)

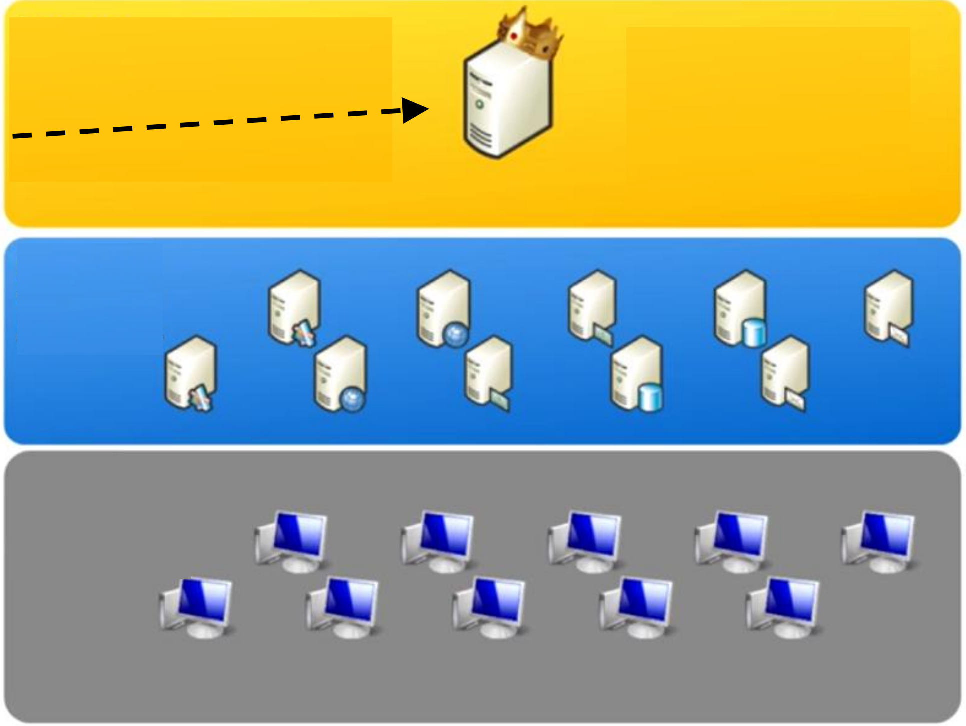
- **Defined** and **enforced** by **each o.s.**
- Each machine **independent** of each other



Example Scenario (II)

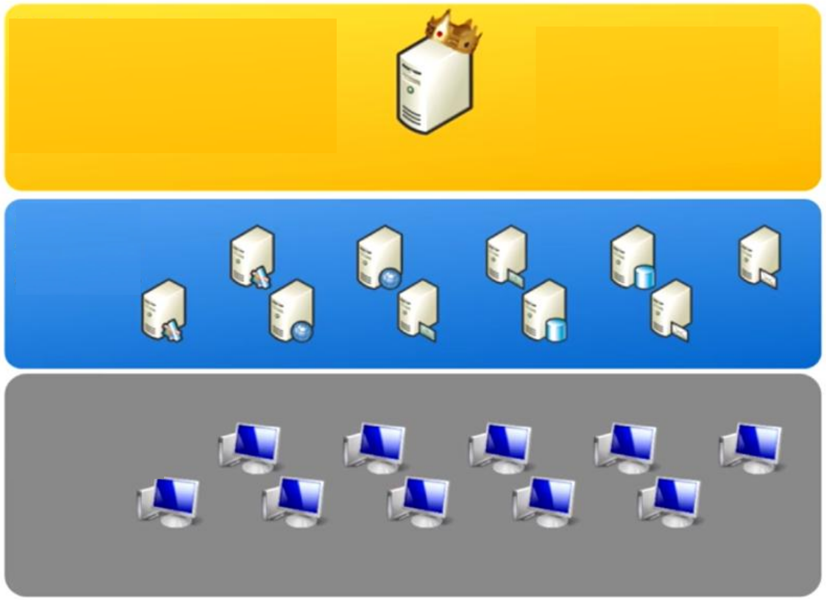
- Defined in **one** single place
- Enforced **everywhere**

Accounts
Resources
Policy



Example Scenario (III)

- Defined and enforced in the cloud
- **Independent** of those of the organization



Accounts
Resources
Policy

Access Control: Specification?

- For each <Account, Resource> which **Operations** are allowed

- Account A can read/write every file that it owns
- Accounts of interns can only read files in directories D1, D2
- Accounts in group G1 can modify the composition of group G2
- ...but only if the account is not an intern

*Which rules can be **defined**?*

Access Control: Enforcement?

- For each <Account, Resource> which **Operations** are allowed

- Account A can read/write every file that it owns
- Accounts of interns can only read files in directories D1, D2
- Accounts in group G1 can modify the composition of group G2
- ...but only if the account is not an intern

*..and how are they **enforced**?*

Access Control: Specification

- MANY models with different **expressiveness**

- Every concrete scenario:
  - **Hybrid** of several models
  - Many complex details

  - Windows / Linux / Android / ...
  - AWS / Azure / GCP / ...
  - Tomcat / Postfix / MySQL / ...

Access Control: Enforcement

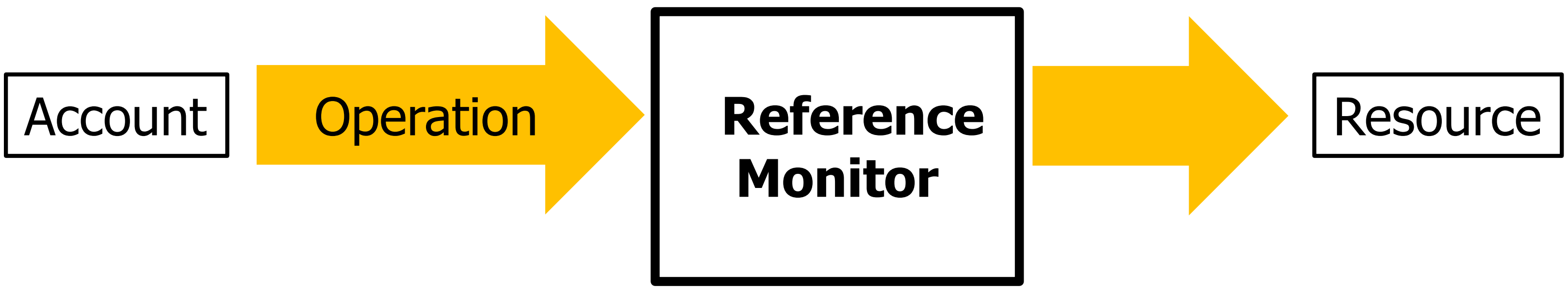
- Strongly dependent on the **operational scenario**
  - One machine
  - Many machines in a single organization
  - Many machines in many organizations
  - Web apps
  - Web apps with delegated authentication / authorization
  - Cloud services
  - ...

Our approach

- Operational scenario:
  - **One machine**
  - Later: Many machines in a single organization

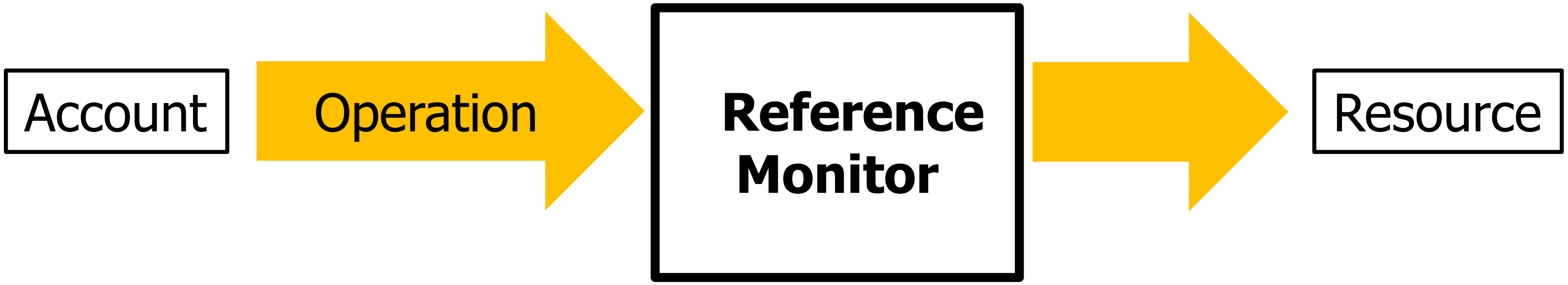
- Concrete implementation (specification and enforcement):
  - Linux, Windows
  - Just an **outline**

Access Control Model (preliminary)



- **Every** access to **resources** is mediated **(guarded)** by a "Reference Monitor"
- ...that knows the Security Policy

Access Control **Model**



- Think in terms of this **model**
- **Not** of how it is implemented

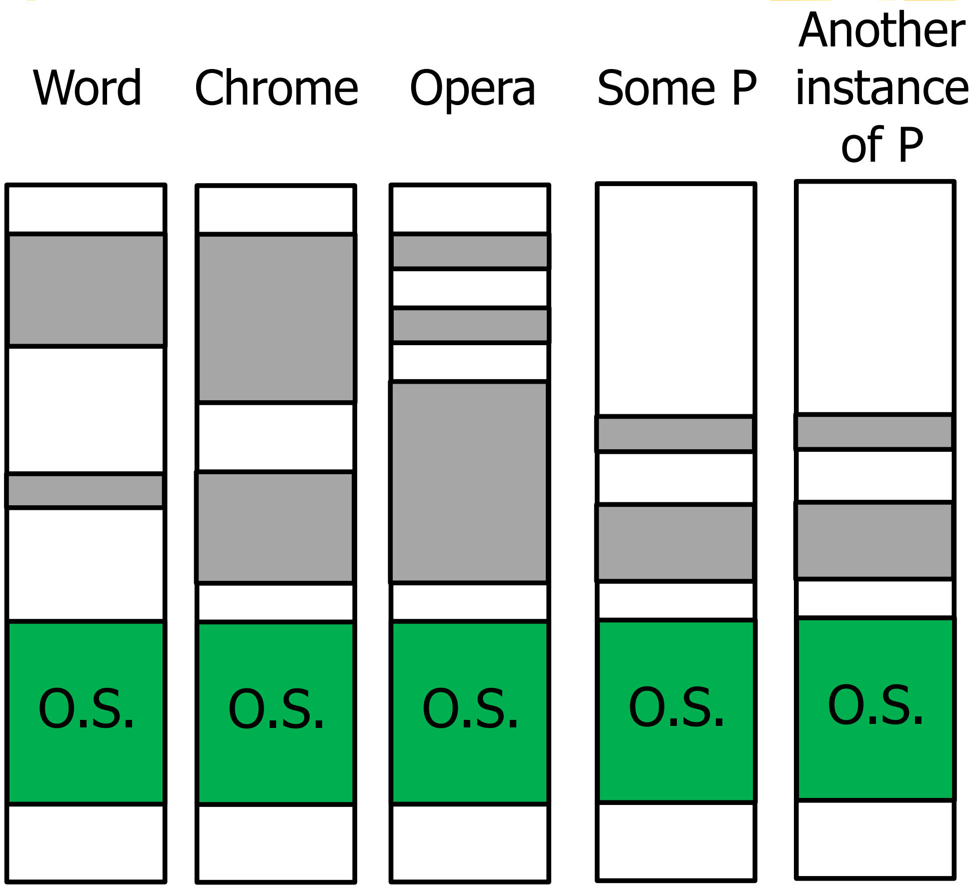
Example

Access Control: O.S.

Computer Architecture in a nutshell

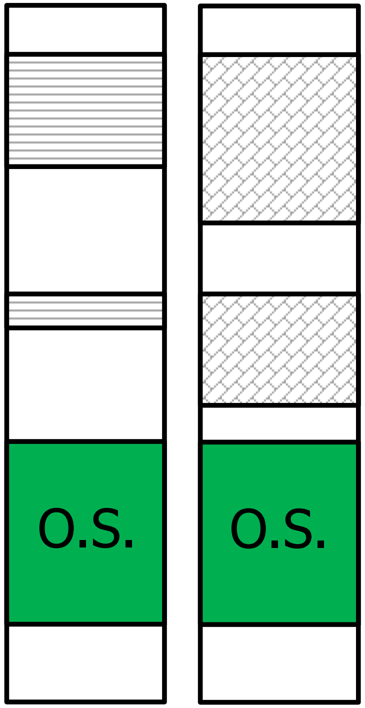
Process Address Space (I)

- "The executed program" (**user-level** code)
- Operating System (**system-level** code)
- Loaded at bootstrap



Process Address Space (II)

- Every process has **its own** address space
- Address spaces are **isolated** from each other
  - CPU executes process **P** and issues `addr-x`
  - CPU executes process **Q** and issues `addr-x`
  - The referenced cell is **different**
    (it might contain the same value)
- Isolation implemented by **hardware + O.S.**
  - The O.S. places itself at the **same** address,
    in **every** address space



Virtual Memory vs Physical Memory

- CPU executes process P and issues `addr-x`
- CPU executes process Q and issues `addr-x`
  - **Virtual** memory
- The referenced cell is **different**
  (it might contain the same value)
  - **Physical** memory
- Isolation implemented by **hardware + O.S.**
  - CPU emits (process-id, v-address)
  - Hardware with o.s. data maps to (p-address)

- Process address space: **virtual** memory
- Machine address space: **physical** memory

Address Space Size: Virtual vs Physical

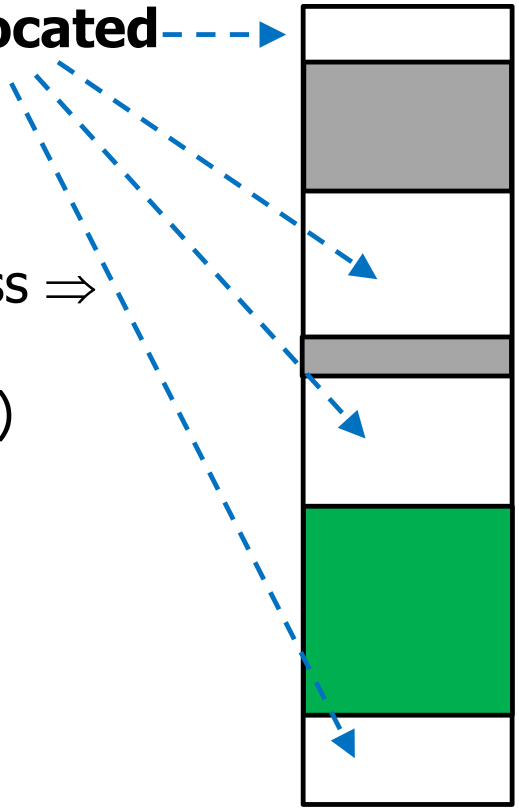
- **Virtual** address space size
  - Memory of **each** process: 2^64 addresses
    ⇒ 2^44 * 2^20
    ⇒ 2^44 M
    ⇒ 2^32 * 2^12 M
    ⇒ **4 * 10^9 * 1024** M

- **Physical** address space size
  - How much memory does your PC have? Maybe **16** GB?

- **A lot of** virtual mem. mapped **to much smaller** physical mem.

(Virtual) Address Space Allocation

- Every address space has parts that are **unallocated** (≈ not usable)



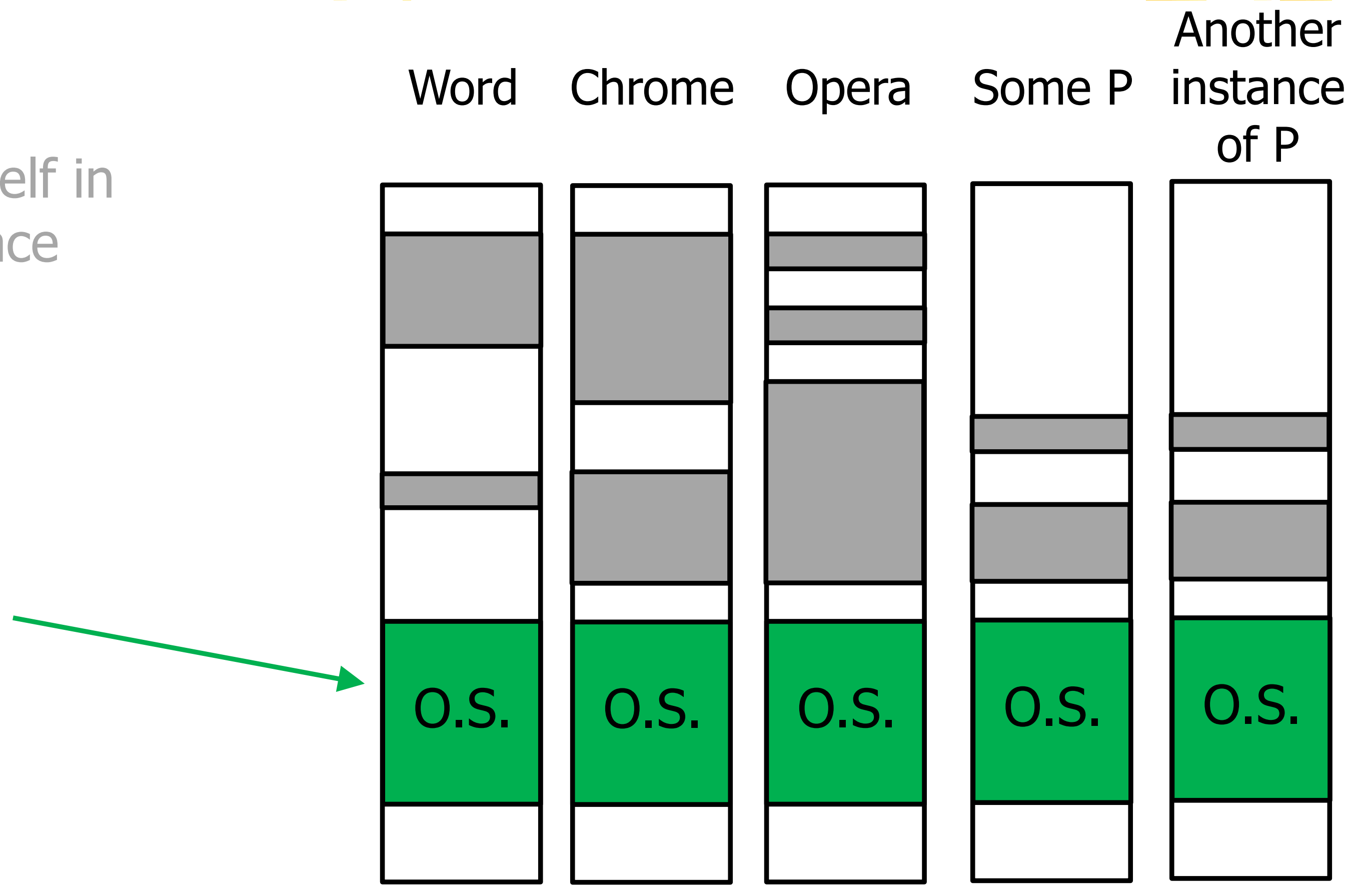
- CPU attempts to access an unallocated address ⇒
  1. Hardware error
     ((process-id, v-address) → memory fault)
  2. O.S. procedure called automatically
     (memory fault handler)

- I am neglecting swapping on secondary storage for simplicity...

Operating System

- The O.S. places itself in **every** address space

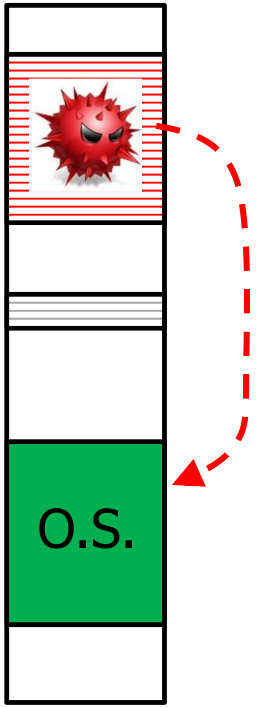
- Code
- Variables
  - Open sockets
  - Open files
  - "Permissions"
  - ...



O.S. Integrity?

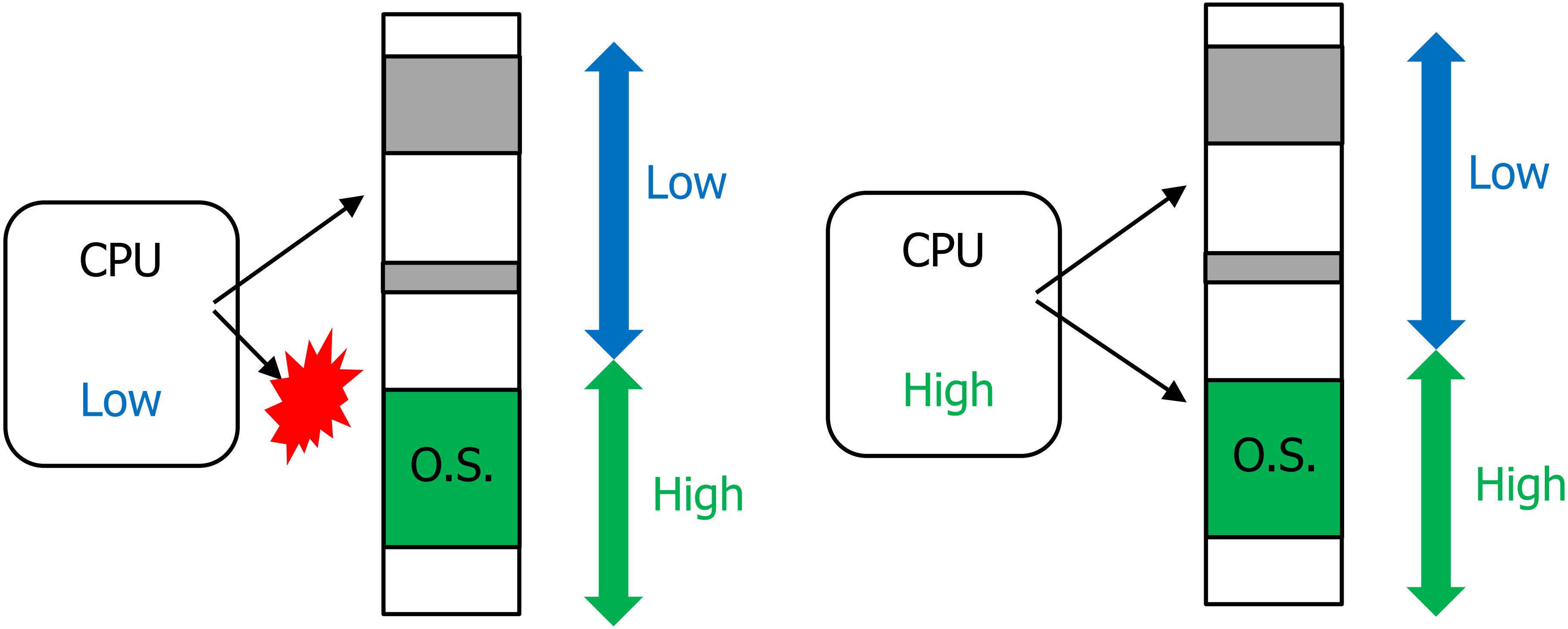
- A malicious process could attempt to:
  - Read o.s. variables
  - Write o.s. variables
  - Jump to arbitrary o.s. addresses

  - Read sensitive information (crypto keys / passwords / ...)
  - Modify "access rights" (access files that should not be accessed)
  - Skip permission checks



CPU Privilege Level: Memory Access Rights

- Every CPU has (at least) two privilege levels: High and Low
  - High ⇒ CPU can access **every** address
  - Low ⇒ CPU can access only **some** addresses



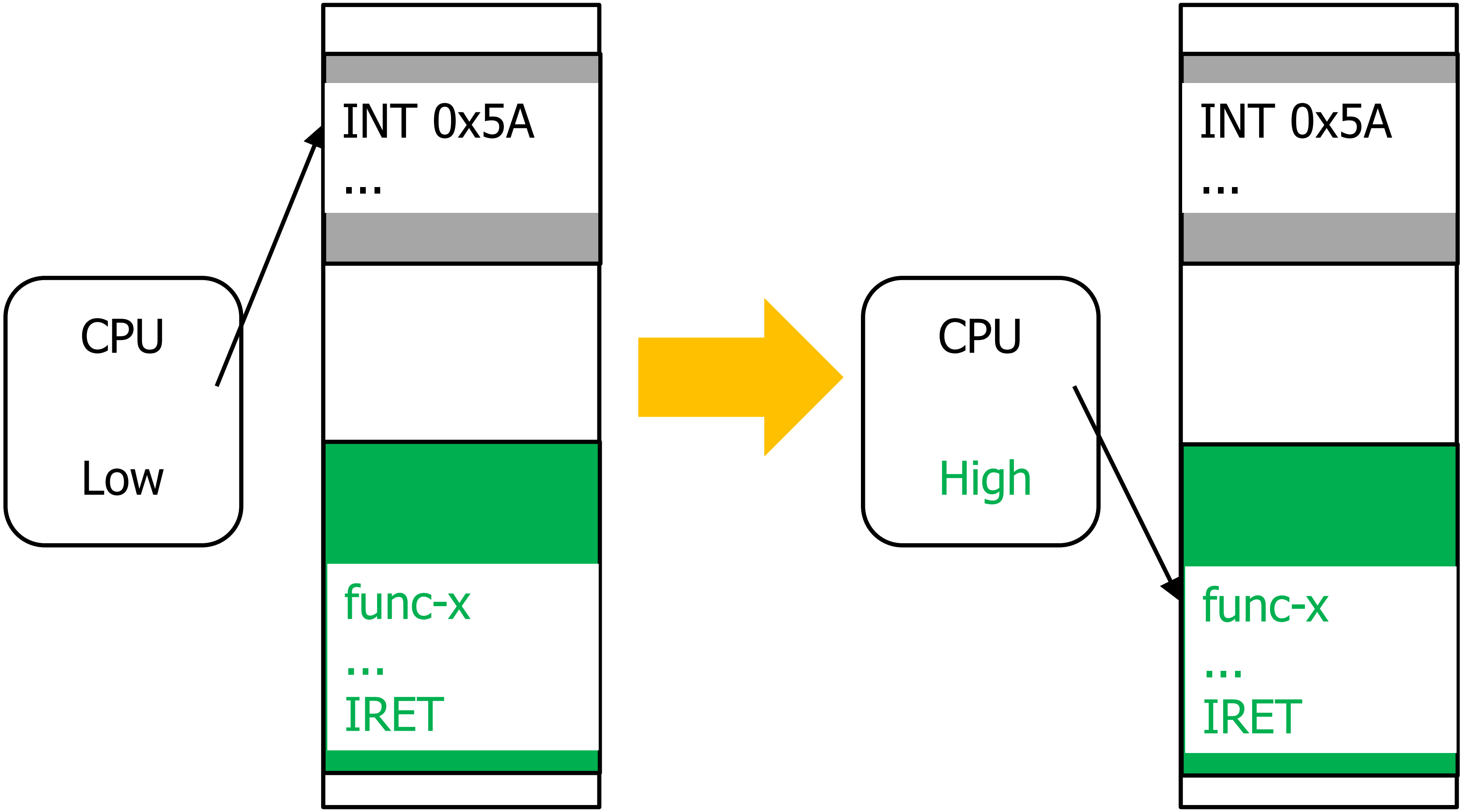
CPU Privilege Level: Privilege Switch

- Privilege level switch occurs in **hardware**

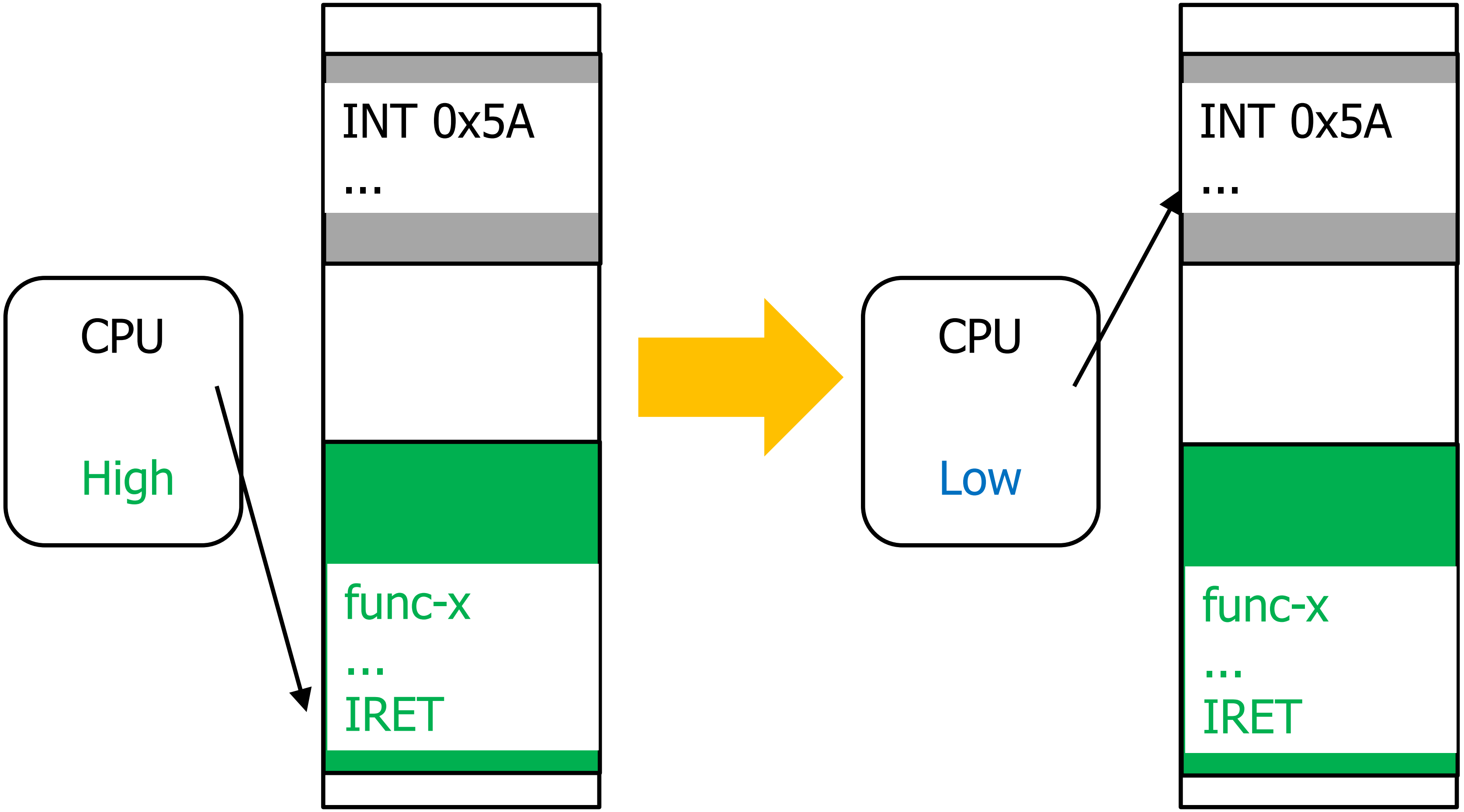
- **Low → High**
  - `INT operand` Calls a function in the o.s.
  - Mapping `operand` values → functions **predetermined** by the o.s.

- **High → Low**
  - `IRET` Return to caller user code

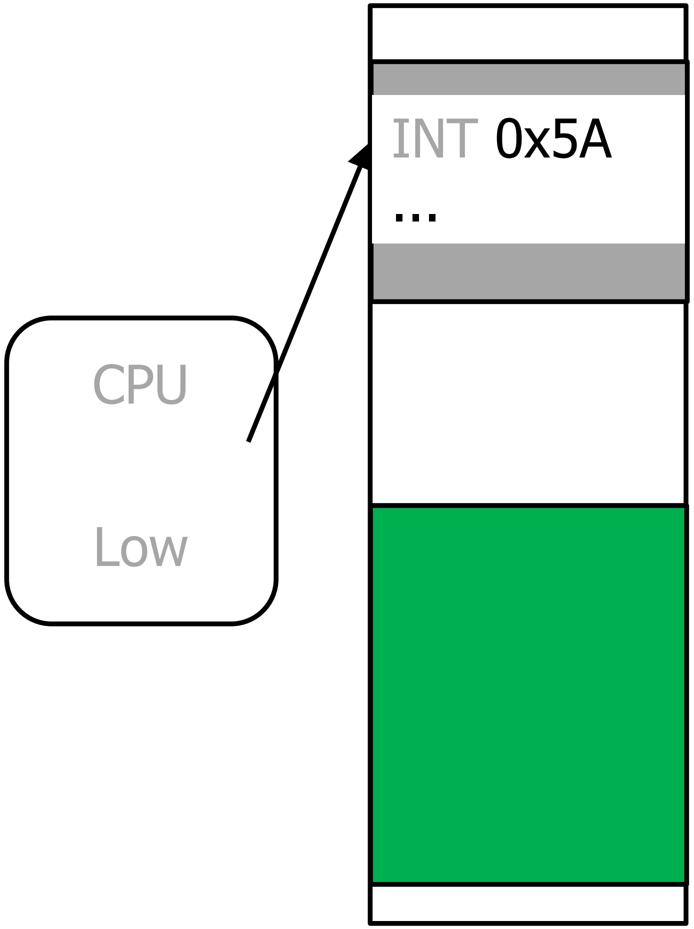
System Call Invocation



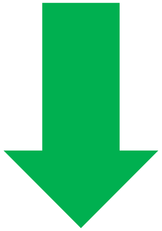
System Call Return



Remark



- CPU does **not** use `operand` as an address
- CPU uses `operand` as an **offset** in an o.s. table (that contains addresses)

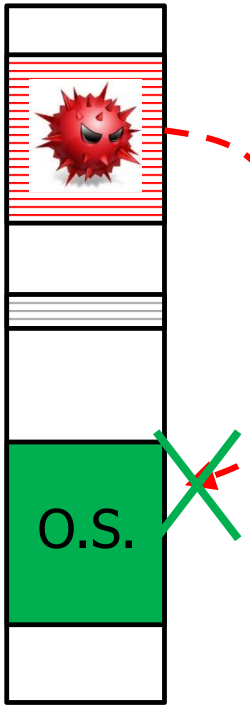


- Untrusted code can **only** call **predefined** addresses

O.S. Integrity

- A malicious process could attempt to:
  - Read o.s. variables
  - Write o.s. variables
  - Jump to arbitrary o.s. addresses

- **Not possible**:
  - Read / Write o.s. variables
    (it executes with Low privilege)
  - Jump to arbitrary o.s. addresses
    (it can only call predefined addresses)



Keep in mind

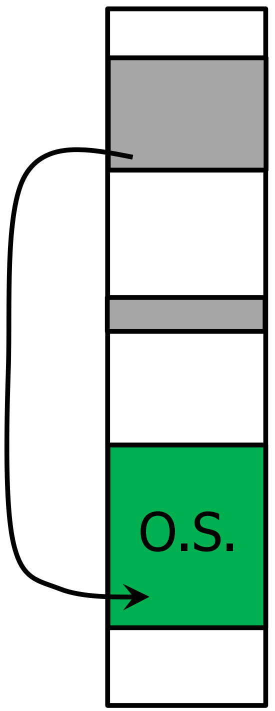
- User-level program executes with Low privilege
- O.S. executes with High privilege

↓

- User-level program:
  - Cannot access O.S. data
  - Can enter O.S. only at predefined points (by invoking a system call)

Resource Access (I)

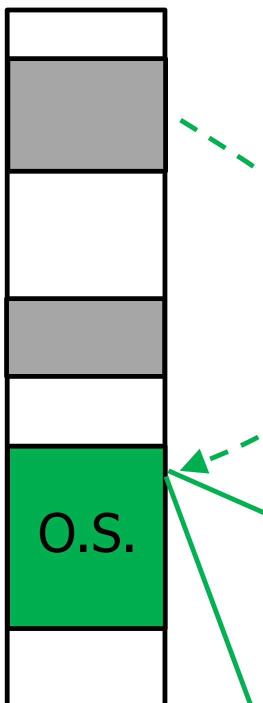
- Every **resource** is implemented by the o.s.
  - File
  - Socket
  - Screen
  - ...



- Every **operation** on a **resource** occurs by invoking a **system call**

- The o.s. decides whether to **grant** or **deny** the operation
  - We will see based on which criteria

Resource Access (II)




```
operation_A(resource_R,...)
```

```
IF   process.account can execute A on R
THEN execute A on R
ELSE return error_code
```

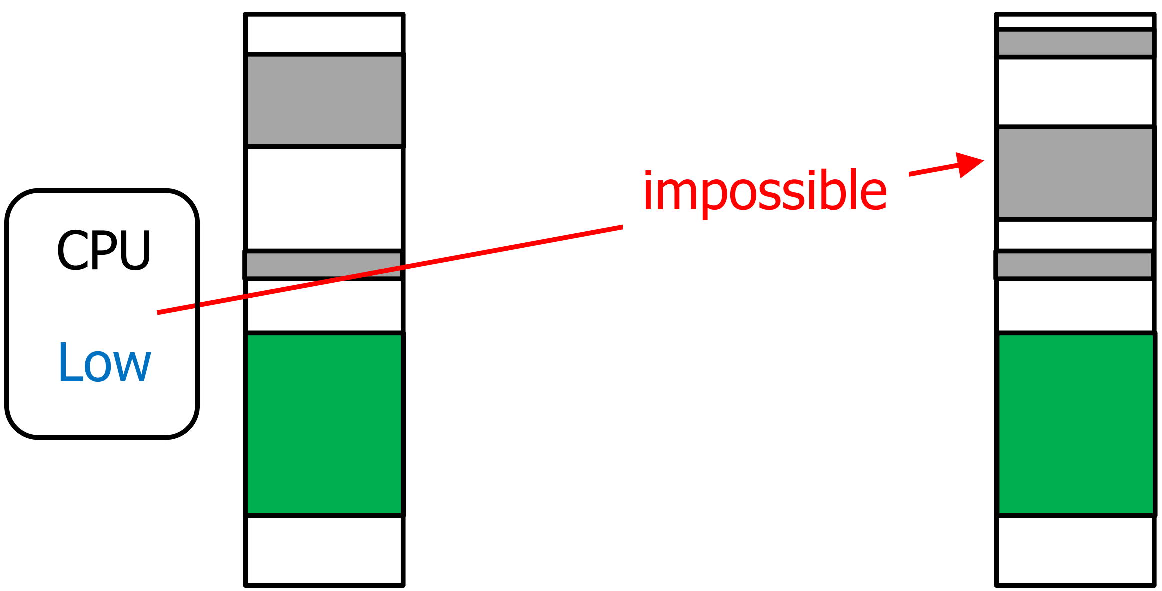
Resource Access (III)

Every access to **resources**
is mediated (**guarded)** by the O.S.

Resources can only be accessed
through **system calls**

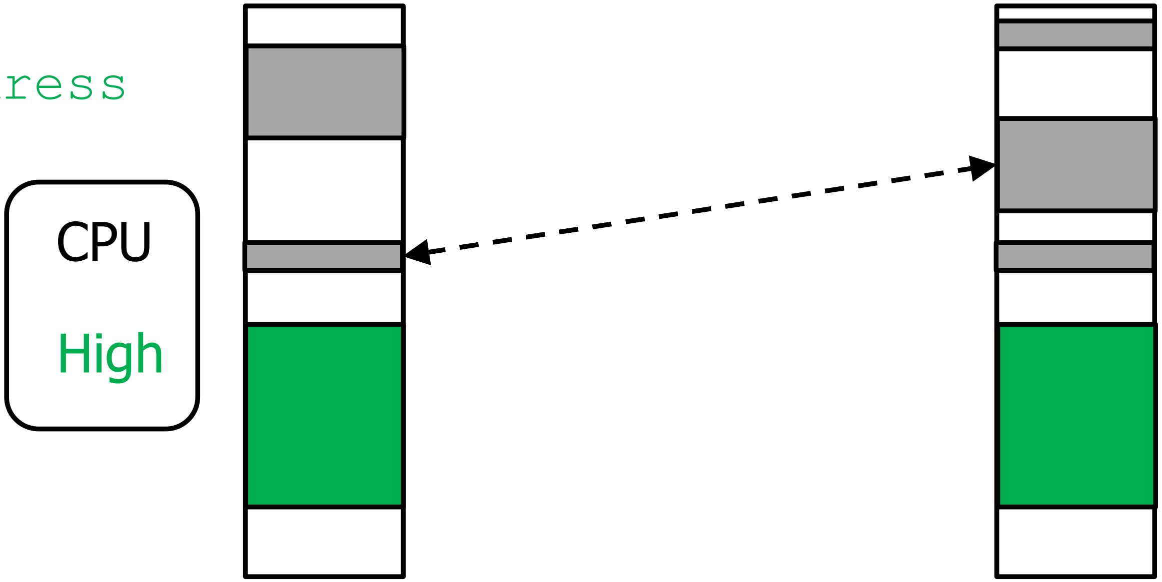
Isolation (I)

- A process **cannot** access the memory of another process **directly**
  - (P,v-address) and (Q, v-address) always map to **different** physical memory regions
  - ...except for v-address of the o.s.

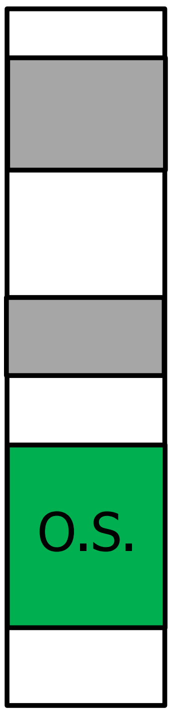


Isolation (II)

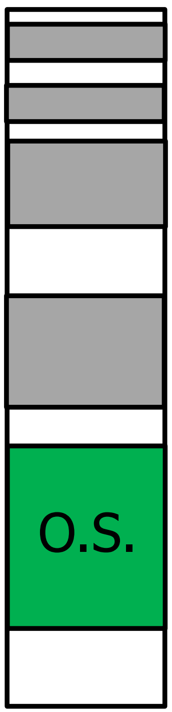
- A process can invoke a **system call** for reading/writing the memory of **another** process
- Typical input parameters
  - `other-proc-id`
  - `other-proc-address`
  - `how-many`
  - `this-proc-address`



Windows (Basic idea) (I)

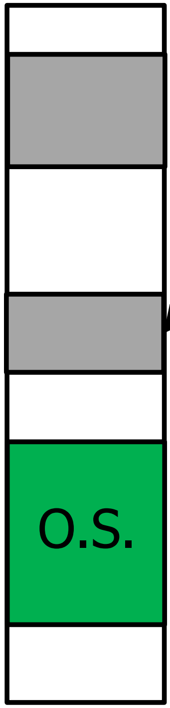


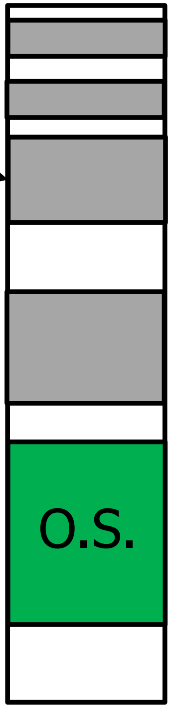

```
HANDLE OpenProcess(
  [in] DWORD dwDesiredAccess,
  [in] BOOL  bInheritHandle,
  [in] DWORD dwProcessId
);
```



Windows (Basic idea) (II)

```
BOOL ReadProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPCVOID lpBaseAddress,
  [out] LPVOID  lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesRead
);
```





Access Control Lists

Process ↔ Account

- Every **Process** is owned by an **Account**
  - A field in the process descriptor within the o.s.

- Basic ideas (more details later)
  - Bootstrap: Root/System account
  - Server Process: Account specified in o.s. configuration
  - GUI / Shell Process: Account that has provided credentials

Resource ↔ Account

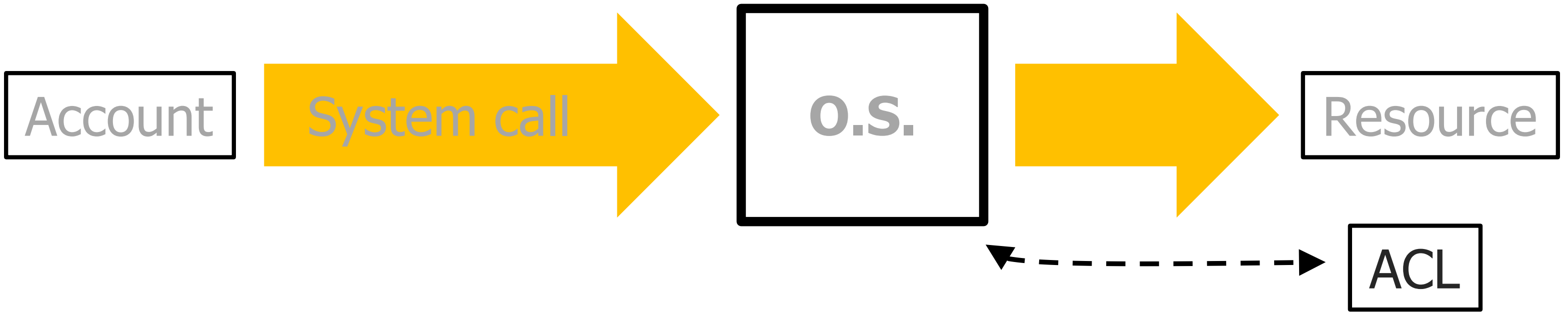
- Every Resource is **owned** by an Account
  - Usually it is the Account that **created** the Resource

- ≈ Resource.owner can do whatever it wants on the Resources that it owns

Hhmmm...

How does the o.s. ***decide*** *whether to grant or deny?*

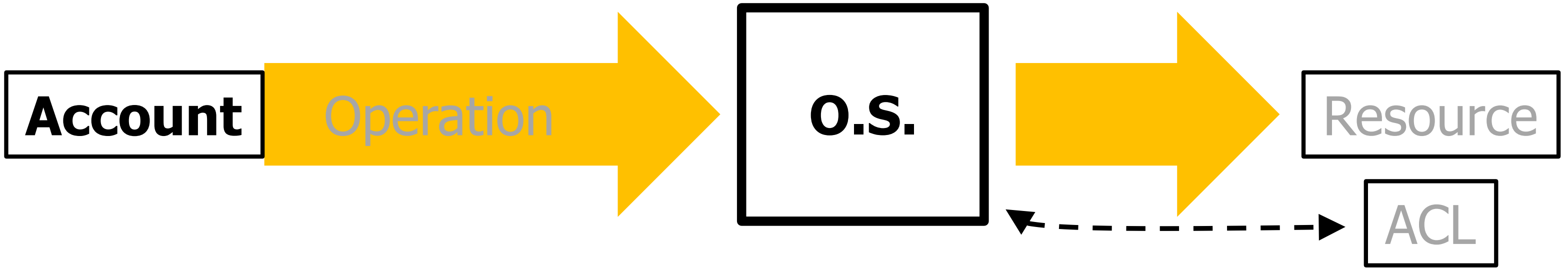
Resource ↔ ACL



- Every Resource has an ACL (**Access Control List**):
  - For each Account,
    Operations that it can execute on the Resource

- **Resource.Owner controls Resource.ACL**
  - R.Owner can execute Operations that modify R.ACL

Preliminary model: Keep in mind



- Decisions taken **only** based on the requesting **Account**
- All **Processes** of the same account can execute the **same** operations ("have the same **access rights**")

Groups and Privileges

Account Groups

- Every Resource has an ACL (**Access Control List**):
  - For each Account,
    Operations that it can execute on the Resource

- Describing it **separately for each Account** may be too complex

- Accounts may be **grouped**
- ACL may be specified in terms of **groups**

Resource Groups

- Every Resource has an ACL (**Access Control List**):
  - For each Account / Group,
    Operations that it can execute on the Resource

- Describing it **separately for each Resource** may be too complex

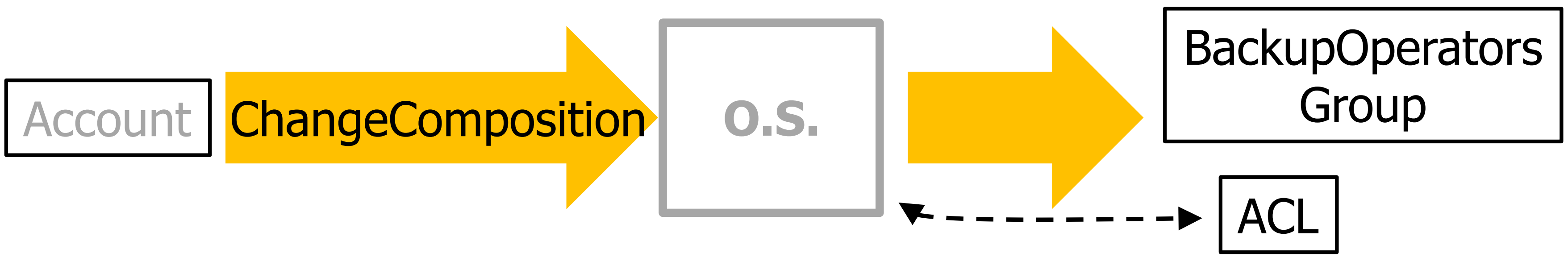
- Resources may be **grouped**
- ACL may be associated with **resource groups**

Hhmmm...

- Process P owned by account U

- Could it add U to **account group** BackupOperators?
- Could it add resource R to a **resource group** that U can access?

Controlling Group Composition



- Account/Resource groups are **resources**
- ...they have their own **ACL**
- ...that **must be structured "correctly"**

Hhmmm...

- Every Resource has an ACL (**Access Control List**):
  - For each Account/Group,
    Operations that it can execute on the Resource

- Certain accounts **must** be able to execute certain operations on certain resources
  - Example: accounts in charge of executing backups must be able to read everything on a filesystem
  - Example: accounts in charge of managing the system must be able to terminate any running process

- Do we need to insert a suitable ACL in **every resource**?
- What if some resource.Owner **does not agree**?

Privileges

- Every Resource has an ACL (**Access Control List**):
    - For each Account/Group,
      Operations that it can execute on the Resource
- The o.s. defines a set of **privileges**
- Each privilege allows executing a **predefined** set of operations on **every resource**
- ...irrespective of resource.ACL

- An account may have one or more privileges

Hhmmm...

- Process P associated with account U

- Could it add **privileges** to U?

"High Privilege" Account

- Each o.s. has one or more **predefined** accounts with **"high privilege"**
  - Linux `root`
  - Windows members of `Administrators` group
  - Windows `SYSTEM` (**not** associated with any user)

- Set of privileges that allows executing **every** operation on **every** resource
- ...irrespective of Resource.ACL

High Privilege Account: What it means

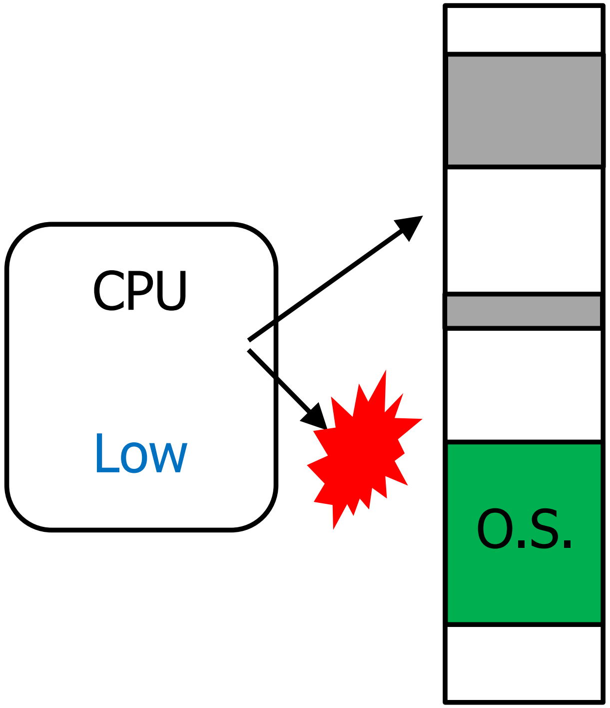
- They can execute **every** operation on **every** resource

- ≈ Every system call invocation by a process of a High Privilege account will succeed

- Examples:
  - "Read memory page M of process P in my buffer B"
  - "Write my buffer B in memory page M of process P"
  - "Delete file F"

High Privilege Account: What it does NOT mean

- ~~Can access every memory address~~



- It is an **o.s.** concept: not an **hardware** concept

Think about this (I)

- Process P owned with account U
- P **creates** resource R
  - File
  - Network connection
  - Child process
  - ...

- Could it **change the owner** of R?

Think about this (II)

- Process P associated with account U1

- Could it **change its account** to U2?

Think about this (III)

- Process P1 associated with account U

- Could it **read/write** the memory of a different process P2?

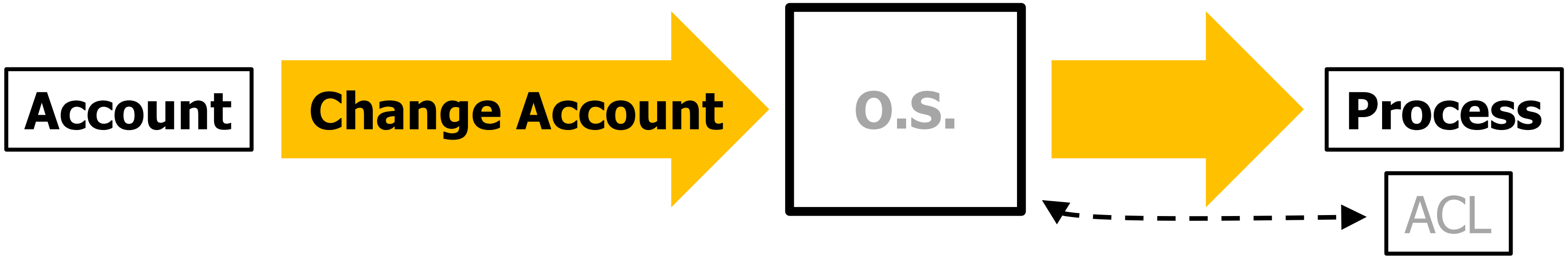
Understanding Process $\leftrightarrow$ Account

Process ↔ Account (REMIND)

- Every **Process** is associated with an **Account**
  - A field in the process descriptor within the o.s.

- Basic ideas (more details later)
  - Bootstrap: Root/System account
  - Server Process: Account specified in o.s. configuration
  - GUI / Shell Process: Account that has provided credentials

Changing Account



- Allowed only to **high privilege** accounts

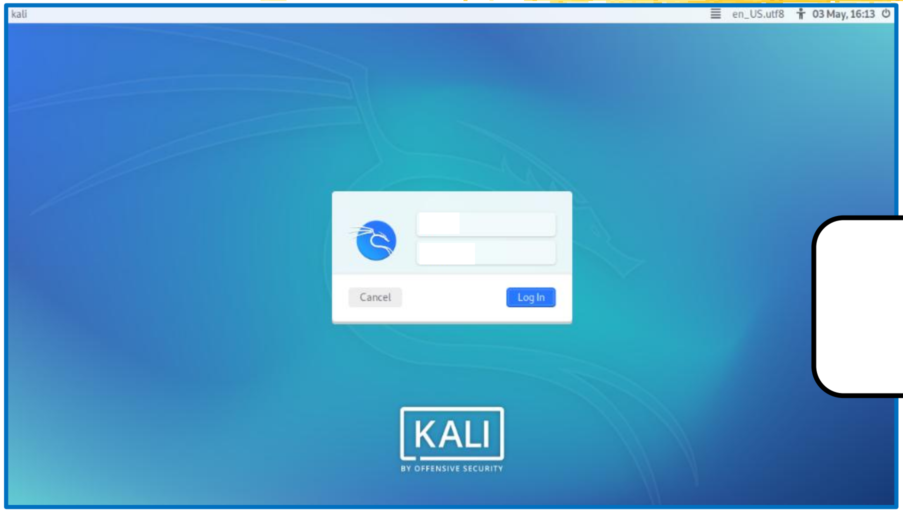
- Linux `setuid()`
- Windows `ImpersonateLoggedOnUser`

Bootstrap

- **Configuration** file describes set of (service, account) to spawn

- **First** process:
  - Associated with an **account** with **high privilege**
  - Read configuration and spawns many **child processes**
  - Child processes can change account **at their will** (because they start associated with high privilege account)

Interactive Logon (I)



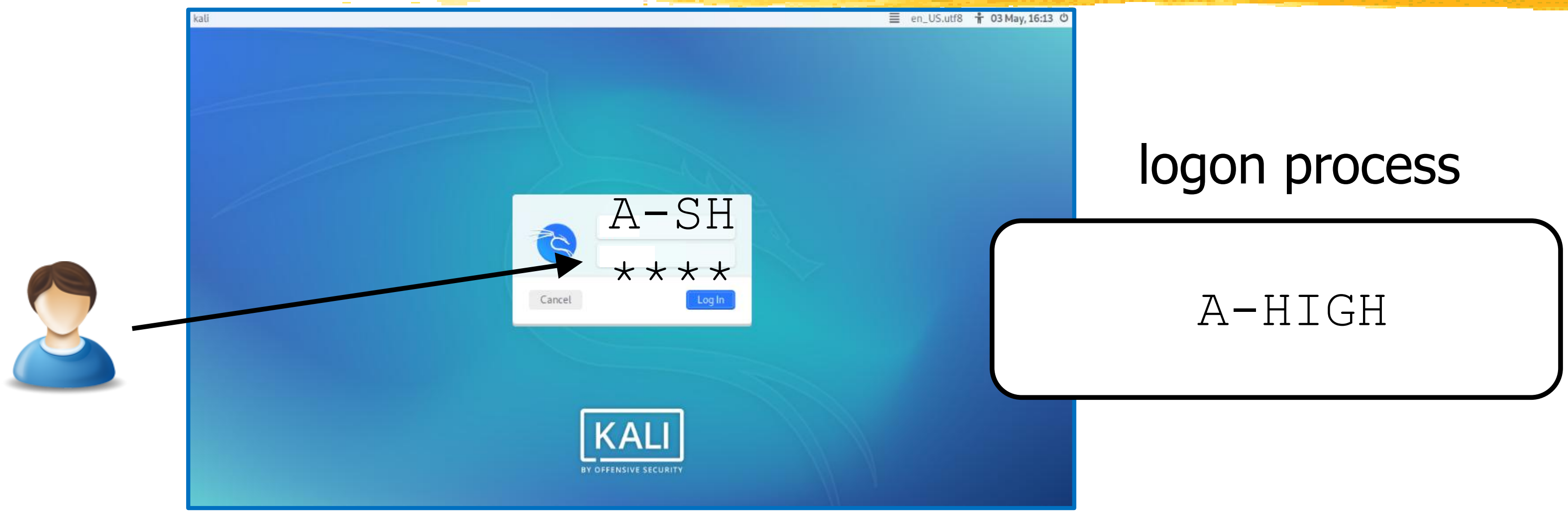
logon process

A-HIGH

Spawned during bootstrap

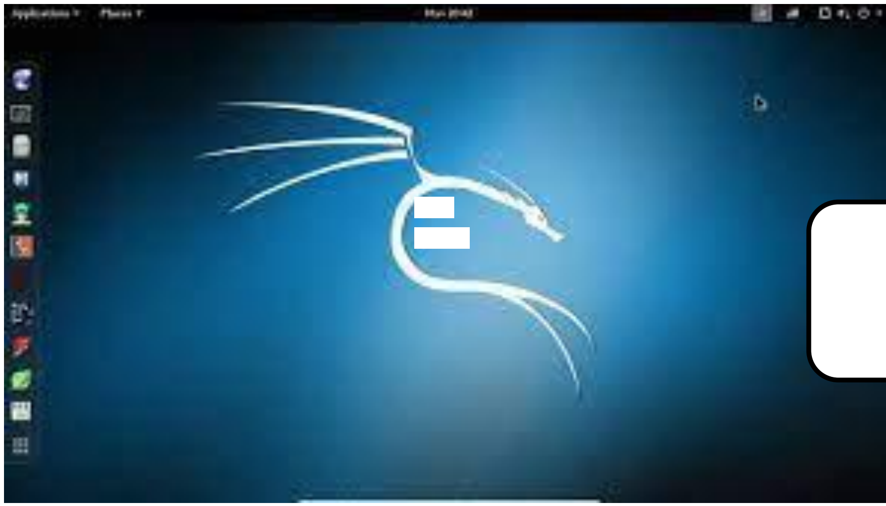
1. Wait for credentials
2. ...
3. ...

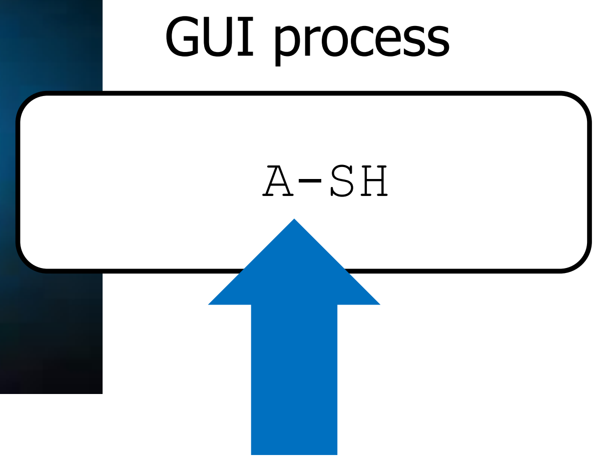
Interactive Logon (II)



1. Wait for credentials
2. Validate inserted credentials (account `A-SH`)
3. ...

Interactive Logon (III)





1. Wait for credentials
2. Validate credentials (authenticate account `A-SH`)
3. Spawn GUI process that **changes** account to `A-SH`

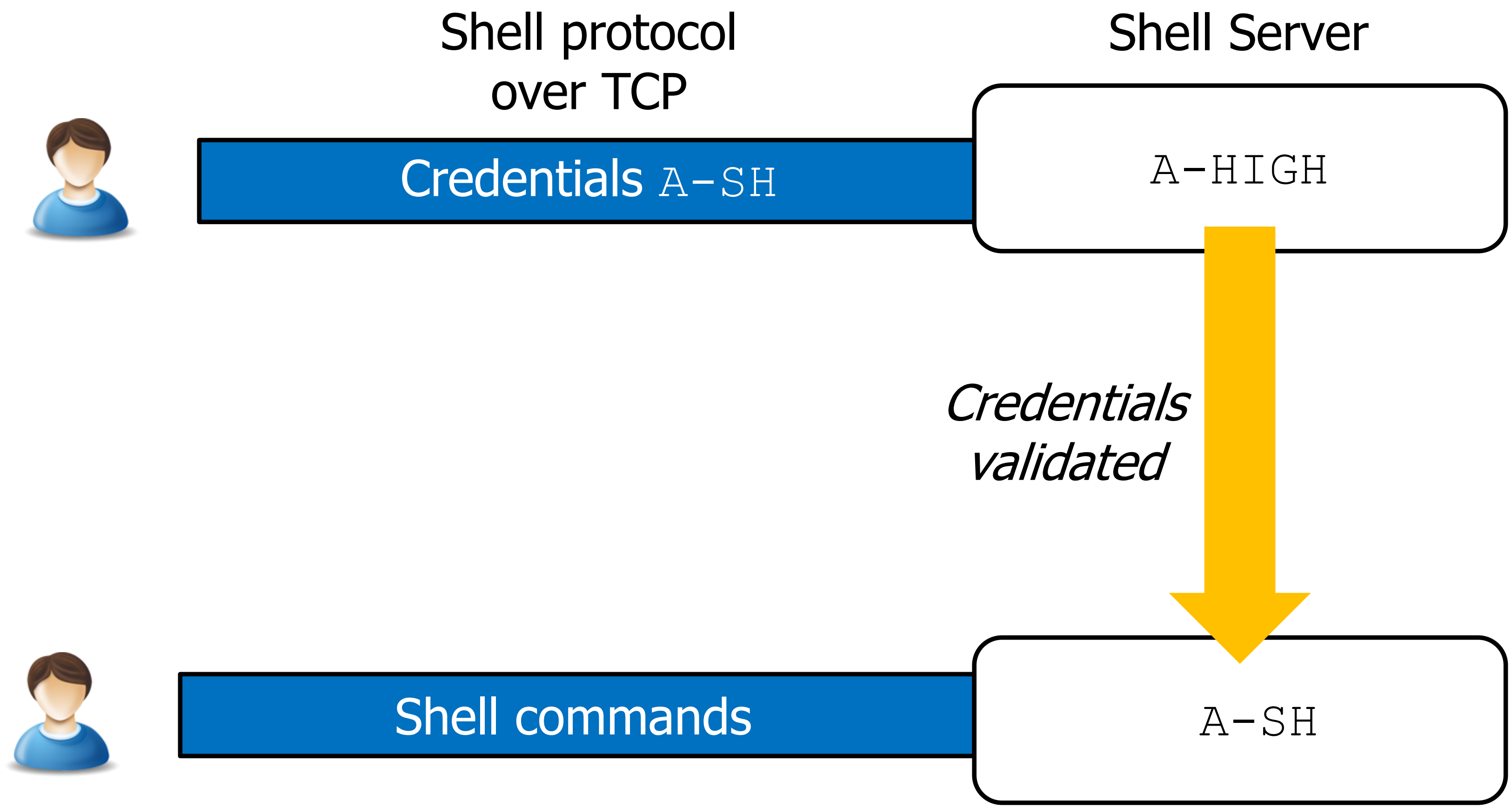
Remote Shell (I)

Shell Server

listen

A-HIGH

Remote Shell (II)



Shell session (I)

- Shell **process** owned by `A-SH`

1. Spawns a child process P
   - Which is the **owner** account of P?
2. P executes file F owned by `A-F`
   - Which **access rights** on F must have `A-SH`?
3. P creates file F1
   - Which is the **owner** account of F1?

Shell session (II)

- Shell **process** owned by `A-SH`

1. Spawns a child process P
2. P executes file F owned by `A-F`
3. P creates file F1

- "**Shell identity propagated everywhere**"
- The owner of the executable files is **irrelevant**

Back to the questions

Important questions (I-III) (REMIND)

- Process P owned by account U

- ...can it access the **memory** of another process owned by **U**?
- ...can it access **files** owned by **U**?

Answer in a nutshell

- Dropbox app and Chrome browser are Processes owned by **the same Account**
- All **Processes** of the same account can execute the **same** operations ("have the same **access rights**")

- Dropbox can read/modify any resource that Chrome can read/modify

Remark

- Dropbox can read/modify any resource that Chrome can read/modify

- We are considering **resources of the operating system**

- The "dropbox account" / "google account" are identities used across the network, on certain **remote servers**
- They have **nothing** to do with **local accounts**

Hhmmm...

- Process P owned by account U
- ...can it access the **memory** of another process owned by **U**?
- ...can it access **files** owned by **U**?

- As far as we know so far: Yes

- Do you really want your **Candy Crush / Pokémon GO apps** to be able to access your **banking tokens**?
- Do you really want an **email attachment** to be able to wipe all **your files**?

Further issues (I)

- Process P owned by account `A-X` requests to operate on a certain resource
- P executes:
  1. Google Chrome / Mozilla Firefox
  2. Excel Macro in an email attachment
  3. Application developed by some student

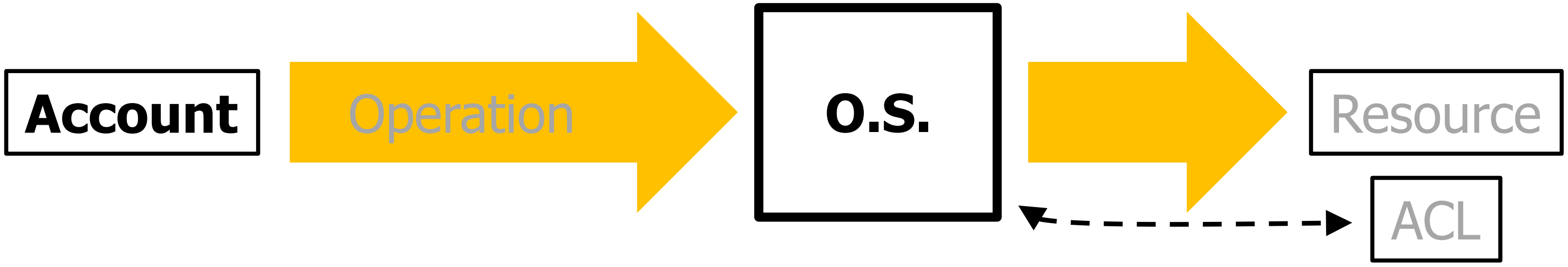
- The preliminary model decides **only** based on the **account**
- It may make sense to decide based **also** on the "**trust**" in the **process**

Further issues (II)

- Process P owned by account `A-X` requests to operate on a certain resource
- P has been created after an authentication that occurred:
  1. Locally
  2. Over a local network
  3. From a remote network location

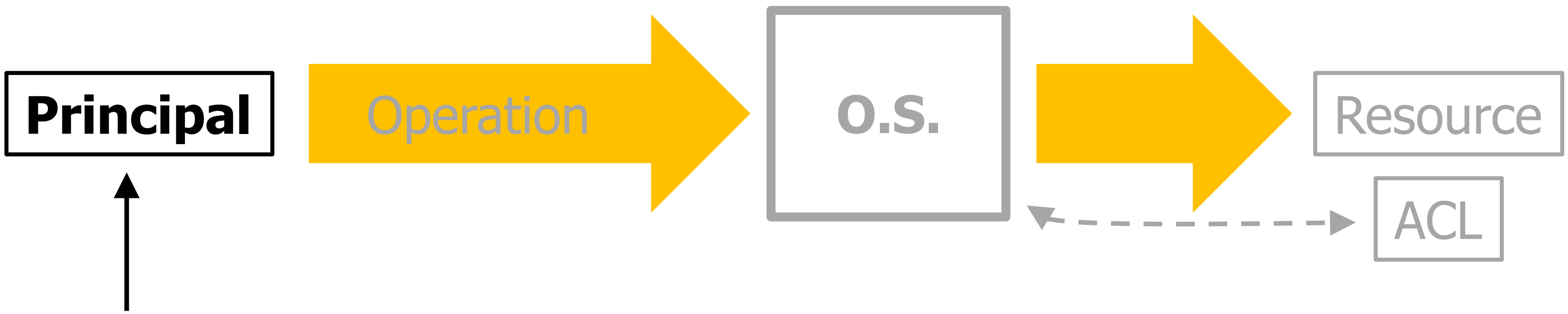
- The preliminary model decides **only** based on the **account**
- It may make sense to decide based **also** on the "**trust**" in the **process**

The Account alone is NOT enough



- Decisions taken **only** based on the requesting **Account**
- All **Processes** of the same account can execute the **same** operations ("have the same **access rights**")

Access Control: In practice

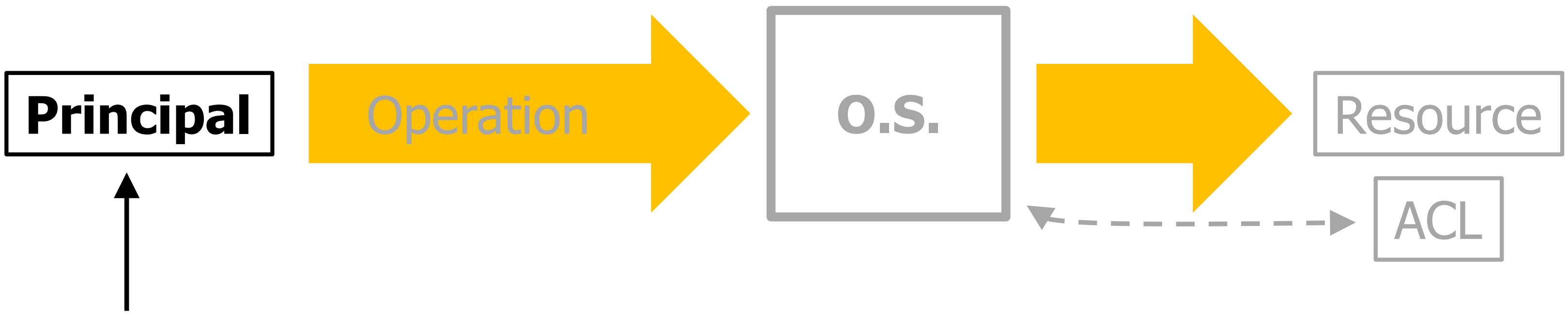


- Account
- Which executable
- How it was authenticated
- Local / Network
- ...

The O.S. can take **different** decisions for the **same** (Account, Operation, Resource)

*We will **not** discuss the details of these extensions*

Example (outline) (I)

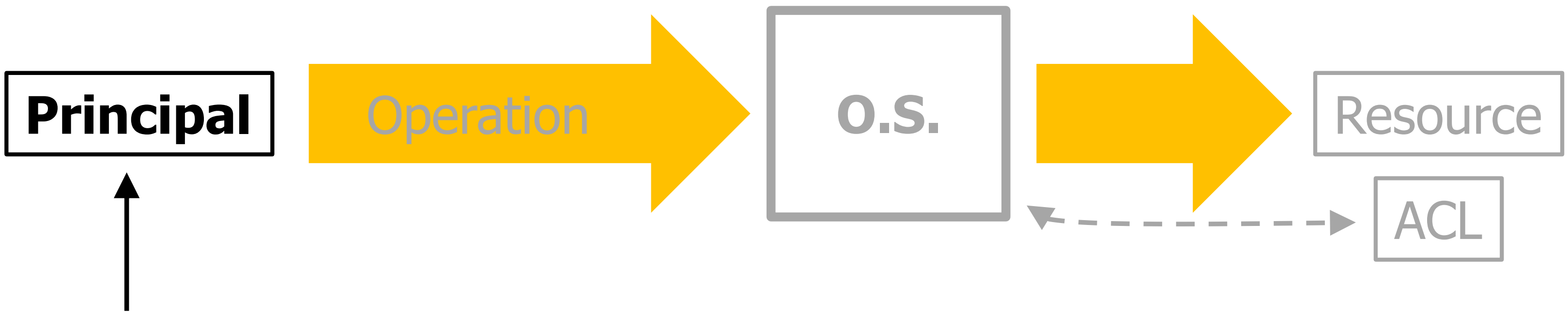


- Alberto
- **Candy Crush app**
- ...

- Alberto
- **PosteID app**
- ...

The O.S. can take **different** decisions for the **same** (Account, Operation, Resource)

Example (outline) (II)



- `Alberto`
- **`Local`**
- `...`

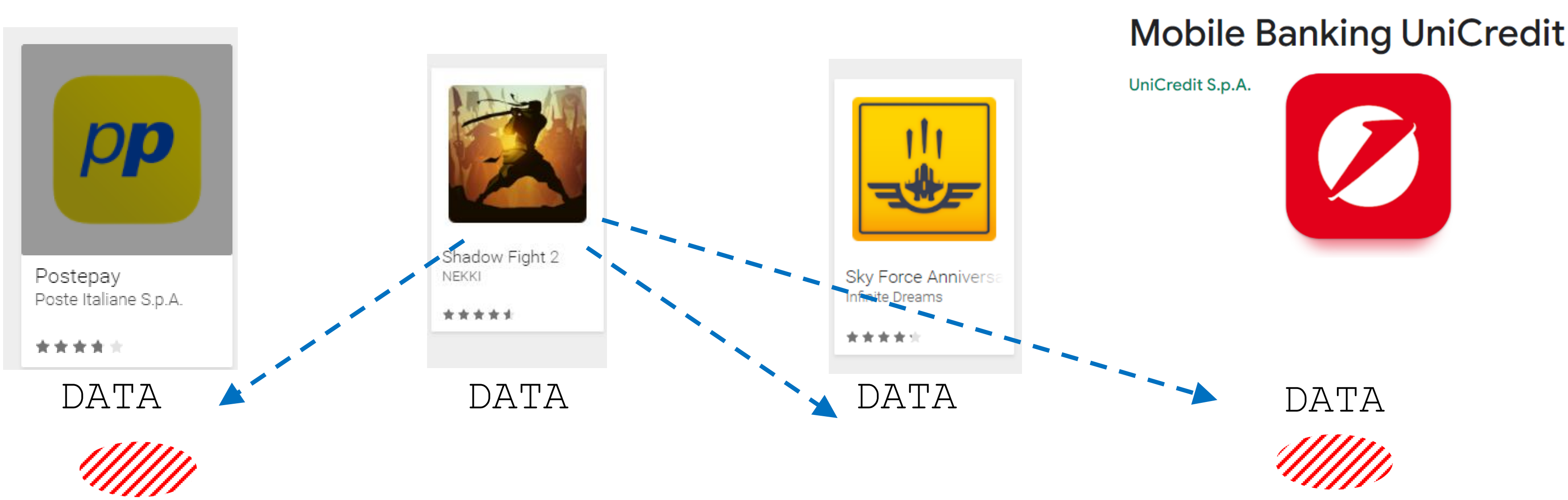
- `Alberto`
- **`Network`**
- `...`

The O.S. can take **different** decisions for the **same** (Account, Operation, Resource)

Smartphone Access Control (in a nutshell)

ACL based ONLY on Accounts

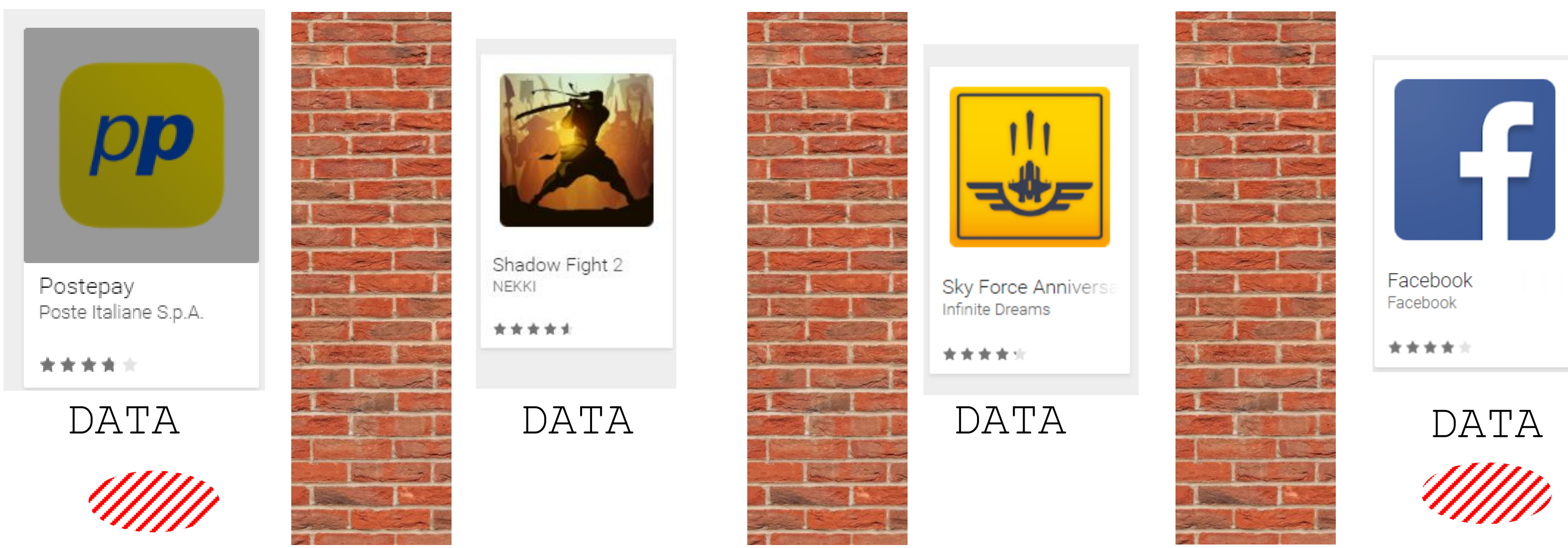
- All **Processes** of the same account can execute the **same** operations ("have the same **access rights**")
- **Any app** of an o.s. account could access **all data of any other app** of that o.s. account
- No, no, no, ...



ACL in Smartphone O.S. (I)

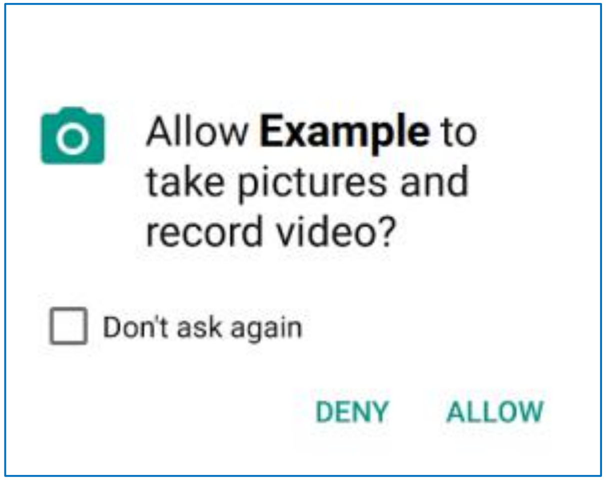
- Each installed app has an app-identifier
- Principal = [Account, app-identifier]

- Data of an app can be **isolated** from other apps of **the same** o.s. account



ACL in Smartphone O.S. (II)

- Access Rights of an app on "critical" resources are granted **by the Human Operator** when **installing** the app



- Resource = Camera
- Resource.ACL = (Account, ExampleApp), (Operation1, Operation2, ...)

ACL Examples: Linux, Windows

Resource ↔ ACL (REMIND)

- Resource.owner decides who can do what on the Resource
- Every Resource has an ACL (**Access Control List**):
  - For each Account,
    Operations that it can execute on the Resource

- Resource.Owner controls Resource.ACL
  - R.Owner can execute Operations that modify R.ACL

ACL Linux (in a nutshell)

Operations vs Access Rights

- Possible **Operations**:
  - Depend on the resource **type**
- Possible **Access Rights:**
  - Read, Write, Execute
  - The **same** for each resource type

- Each operation requires **one or more** access rights
  - Executing a file: R, X
  - Modifying the content of a directory: W, X
  - Set a directory as current directory: X
  - ...
  - Mapping is relatively intuitive

Linux ACL

- Every Resource has an ACL (**Access Control List**):
  - For each Account,
    Operations that it can execute on the Resource
- The set of all accounts is **partitioned**:
  1. Resource.Owner
  2. Accounts in Resource.Owner.Group
  3. All the other accounts

- Each partition has **the same** access rights (thus can execute the same set of operations)
  - Resource.Owner decides which ones

	R	W	X
Owner	x	x	x
Group	x		x
Other	x		

`rwx r-x r--`

chmod

- Modify the ACL of a resource
- Can be executed only by Resource.Owner ("user") and by root

- `chmod` **`u`**`=rw,`**`g`**`=rw,`**`o`**`=r file`
- `chmod` **`go`**`-r file`
- `chmod` **`o`**`+w file`

ACL Windows (in a nutshell)

Windows Security Architecture

EXTREMELY COMPLEX

Operations vs Access Rights

- Possible **Operations**:
  - Depend on the resource **type** **(≈70-80)**
- Possible **Access Rights:**
  - Type-independent set (Delete, WriteOwner, ...)
  - **Type-dependent** set

- Each operation requires one or more access rights
  - Mapping is extremely complex

- "Impossible to remember":
  - Types
  - Operations, Access rights
  - Operations→Access rights

Accounts, Groups (I)

- ~~The set of all accounts is **partitioned**:~~
  1. ~~Resource.Owner~~
  2. ~~Accounts in Resource.Owner.Group~~
  3. ~~All the other accounts~~

- **Many predefined** groups
- Each account belongs to **many** groups

Accounts, Groups (II)

```
PS C:\Users\alberto> whoami /groups

GROUP INFORMATION
-----------------


Group Name
=============================================================
Mandatory Label\Medium Mandatory Level
Everyone
NT AUTHORITY\Local account and member of Administrators group
DESKTOP-H4GP16B\docker-users
BUILTIN\Administrators
BUILTIN\Users
NT AUTHORITY\INTERACTIVE
CONSOLE LOGON
NT AUTHORITY\Authenticated Users
NT AUTHORITY\This Organization
MicrosoftAccount\bartoli.alberto@gmail.com
NT AUTHORITY\Local account
LOCAL
NT AUTHORITY\Cloud Account Authentication
```

Windows ACL

- Every Resource has an ACL (**Access Control List**)
- Sequence of ACE (Access Control **Entries**)
- Each ACE:
  - **Grant** access rights to account or **group**

Show file ACL from shell

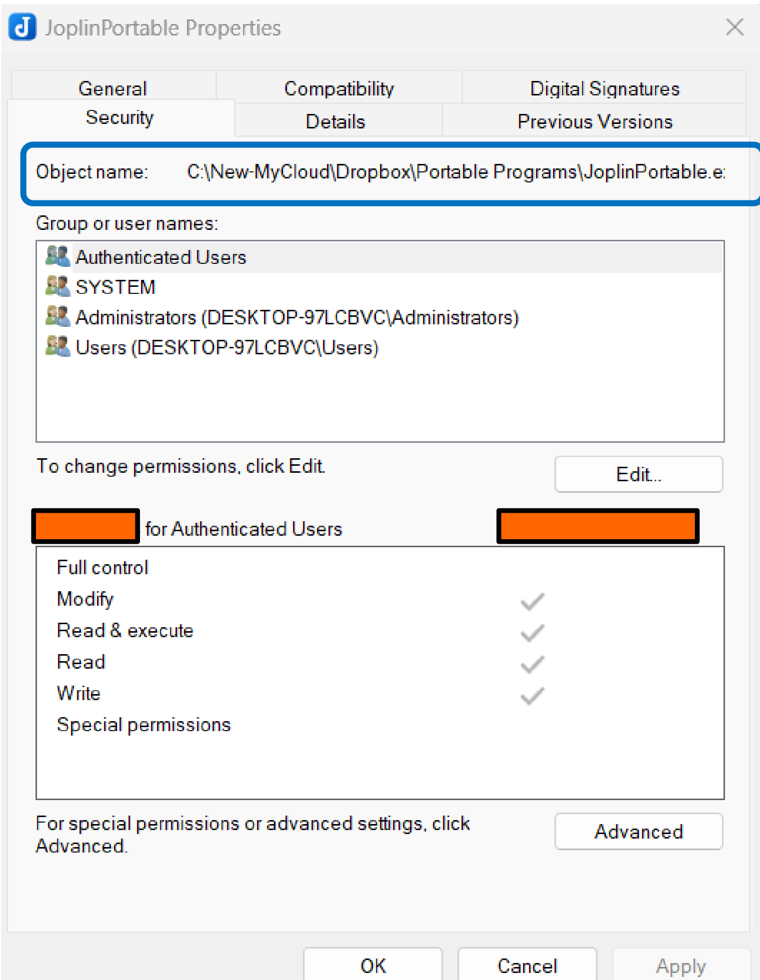
- Linux
  - **`ls -l`** *`filename`*
- Windows
  - **`icacls`** *`filename`*

```
C:\New-MyCloud\Dropbox\Portable Programs>icacls JoplinPortable.exe
JoplinPortable.exe BUILTIN\Administrators:(I)(F)
                   NT AUTHORITY\SYSTEM:(I)(F)
                   BUILTIN\Users:(I)(RX)
                   NT AUTHORITY\Authenticated Users:(I)(M)
```

ACEs

  - Ask ChatGPT to explain output (please see next slides first)
  - Can be used also for **modifying** the ACL

Show file ACL from GUI



users/groups that appear in ACEs

Summary of what the selected user/group can do on this object

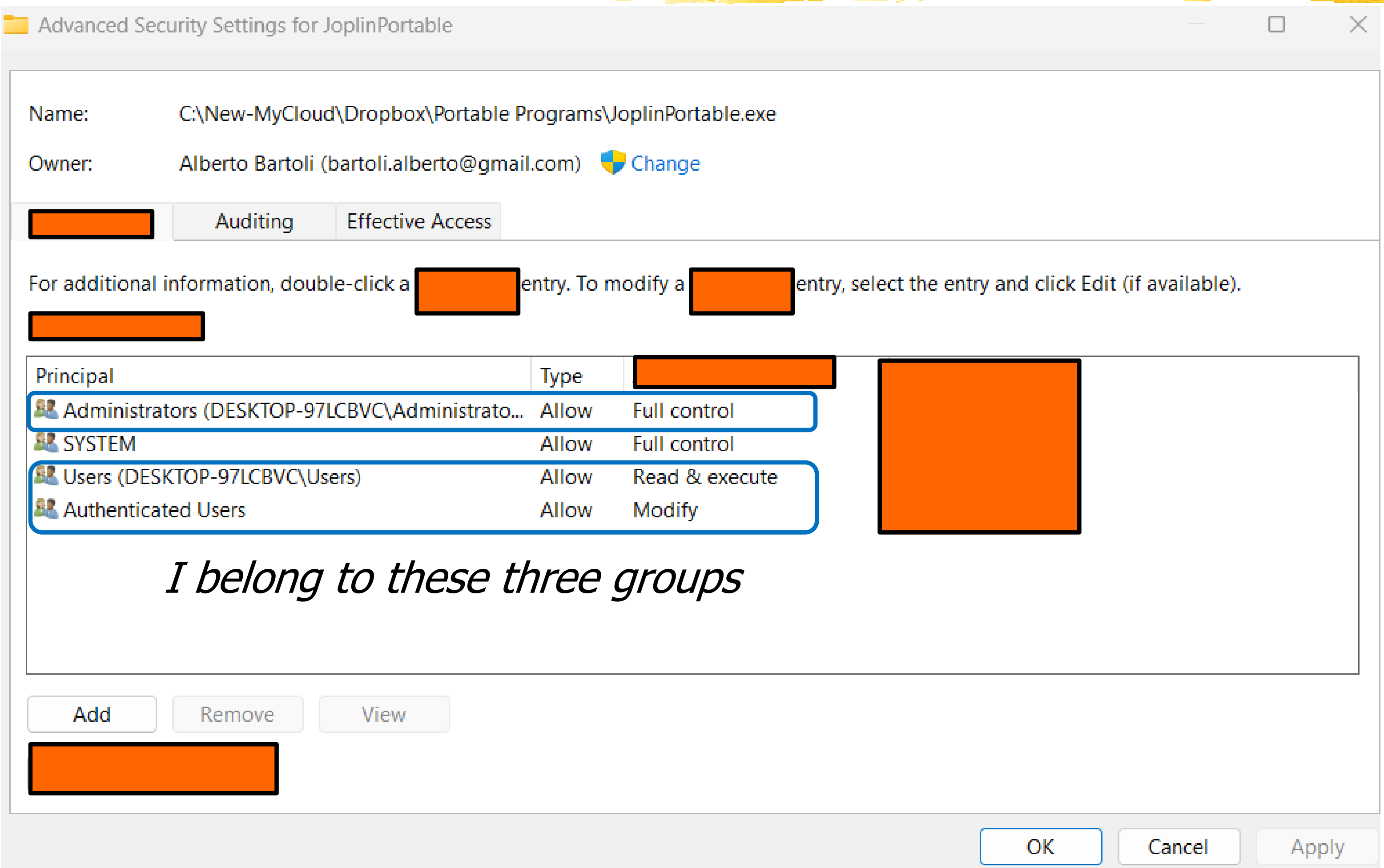
Windows ACL: Complication 1

- Every Resource has an ACL (**Access Control List**)
- Sequence of ACE (Access Control **Entries**)
- Each ACE:
  - **Grant** access rights to account or group

- An account may belong to **multiple** groups
- A **group** may belong to **multiple** groups

- Access rights of an account "**accumulate**" over **multiple** ACEs

Example


Advanced Security Settings for JoplinPortable
Name: C:\New-MyCloud\Dropbox\Portable Programs\JoplinPortable.exe
Owner: Alberto Bartoli (bartoli.alberto@gmail.com) Change
Auditing
Effective Access
For additional information, double-click a entry. To modify a entry, select the entry and click Edit (if available).
Principal
Type
Administrators (DESKTOP-97LCBVC\Administrato... Allow Full control
SYSTEM Allow Full control
Users (DESKTOP-97LCBVC\Users) Allow Read & execute
Authenticated Users Allow Modify
I belong to these three groups
Add
Remove
View
OK
Cancel
Apply

Windows ACL: Complication 2

- Every Resource has an ACL (**Access Control List**)
- Sequence of ACE (Access Control **Entries**)
- Each ACE:
  - Grants access rights to account or group
  - **Deny** access rights to account or group
- Access rights of an account "accumulate" over **multiple** ACEs
- **Complex** rules for composing sequences of ACEs
  - What if an ACE grants and another ACE denies?

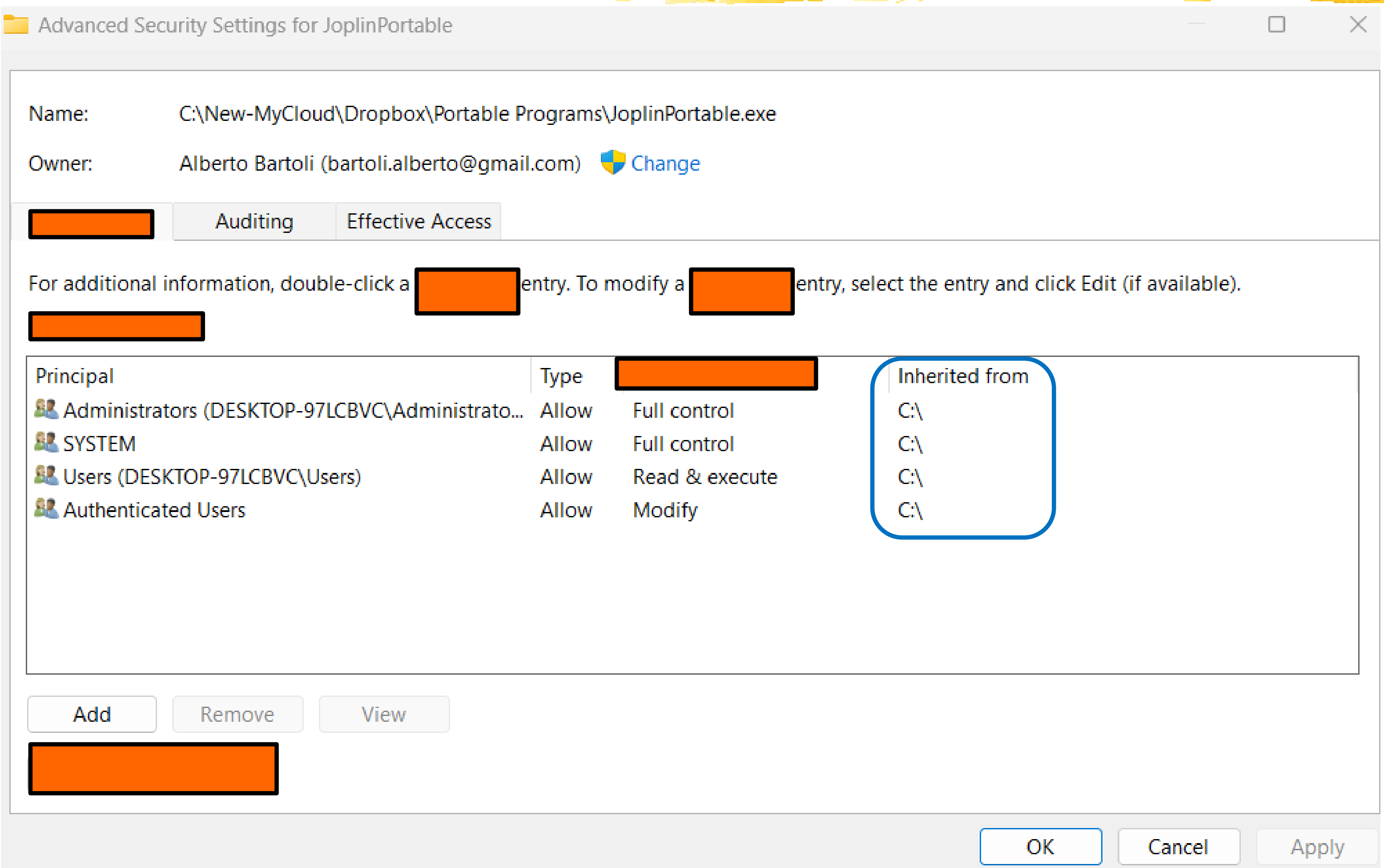
Windows ACL: Complication 3

- A resource may be **contained** in another resource
  - A file is contained in a directory
  - A registry key is contained in its parent registry key
- An ACE may be **inherited** by all the contained resources

- Access rights of an account "accumulate" over **multiple** ACEs possibly **inherited** from **other** resources

- Complex rules for composing sequences of ACEs

Example

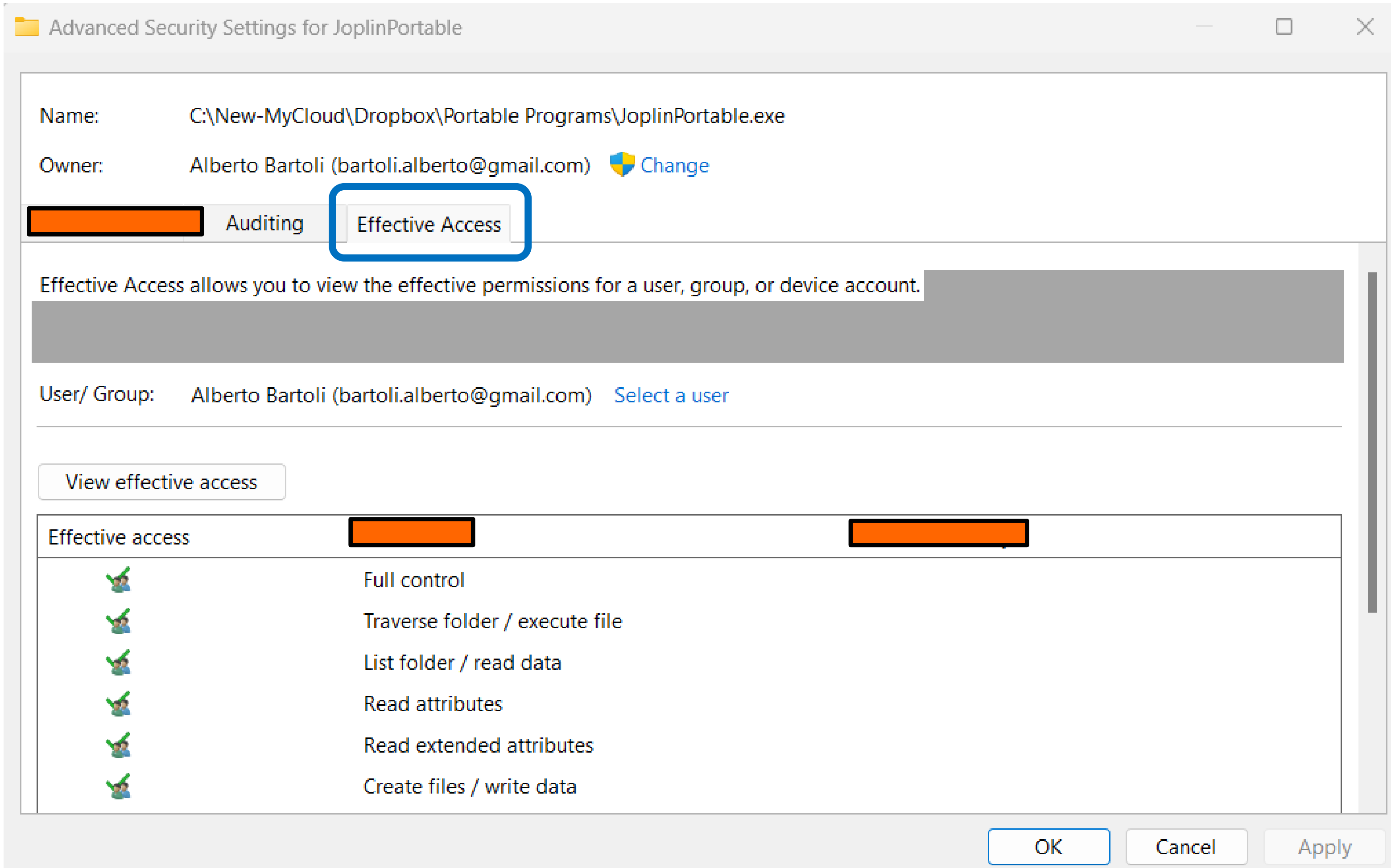

Advanced Security Settings for JoplinPortable
Name: C:\New-MyCloud\Dropbox\Portable Programs\JoplinPortable.exe
Owner: Alberto Bartoli (bartoli.alberto@gmail.com) Change
Auditing
Effective Access
For additional information, double-click a entry. To modify a entry, select the entry and click Edit (if available).
Principal
Type
Inherited from
Administrators (DESKTOP-97LCBVC\Administrato... Allow Full control C:\
SYSTEM Allow Full control C:\
Users (DESKTOP-97LCBVC\Users) Allow Read & execute C:\
Authenticated Users Allow Modify C:\
Add
Remove
View
OK
Cancel
Apply

Key fact

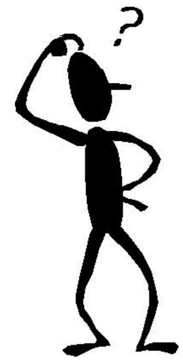
- Understanding who can do what on each resource is **extremely complex**

- The **actual** security policy resulting from ACLs may **not** be the **intended** one
- "Someone can do something they should **not** be able to do"

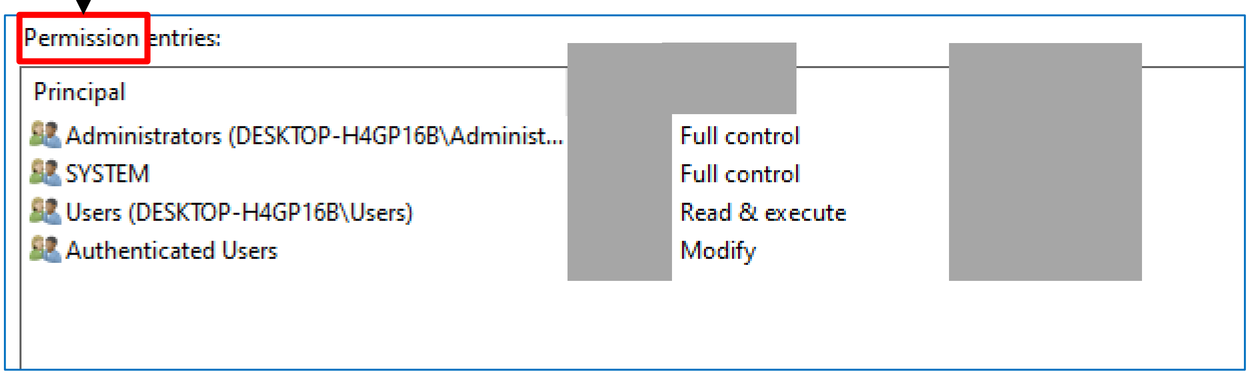
Imagine to do that FOR EACH account/resource...



Nightmare Terminology



*Shouldn't it be an **Access Control** entry?*



more doubts omitted

Access Control: Application Servers

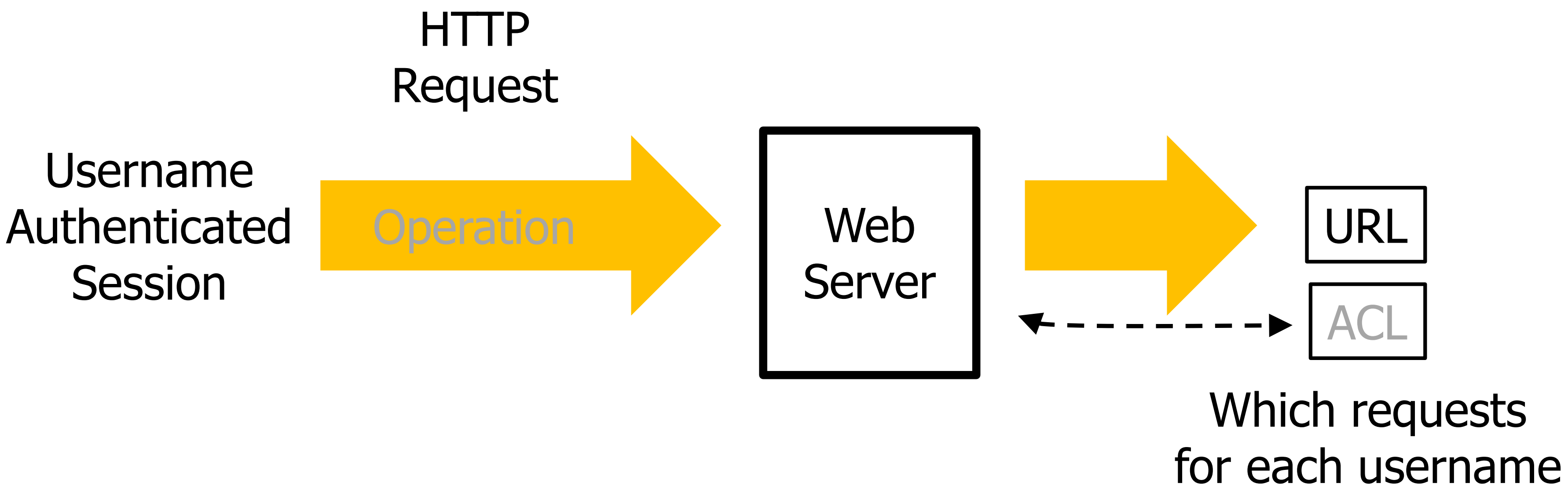
Application Resources?

- **Mail server** manages **mailboxes**
- Mailbox operations are **not** defined in the o.s.
- Access decisions must be taken **by the mail server** (**not** the o.s.)

- **Web server** manages **URLs**
- URL operations are **not** defined in the o.s.
- Access decisions must be taken **by the web server** (**not** the o.s.)

How does access control work for servers?

Access Control – Web Server



Access control must be implemented
in the **application**

Programmed and/or Configured

Example (I)

```xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
</tomcat-users>
```

- Tomcat web server
- Identities and groups (roles)
- Nothing to do with those of the local o.s.

Example (II)

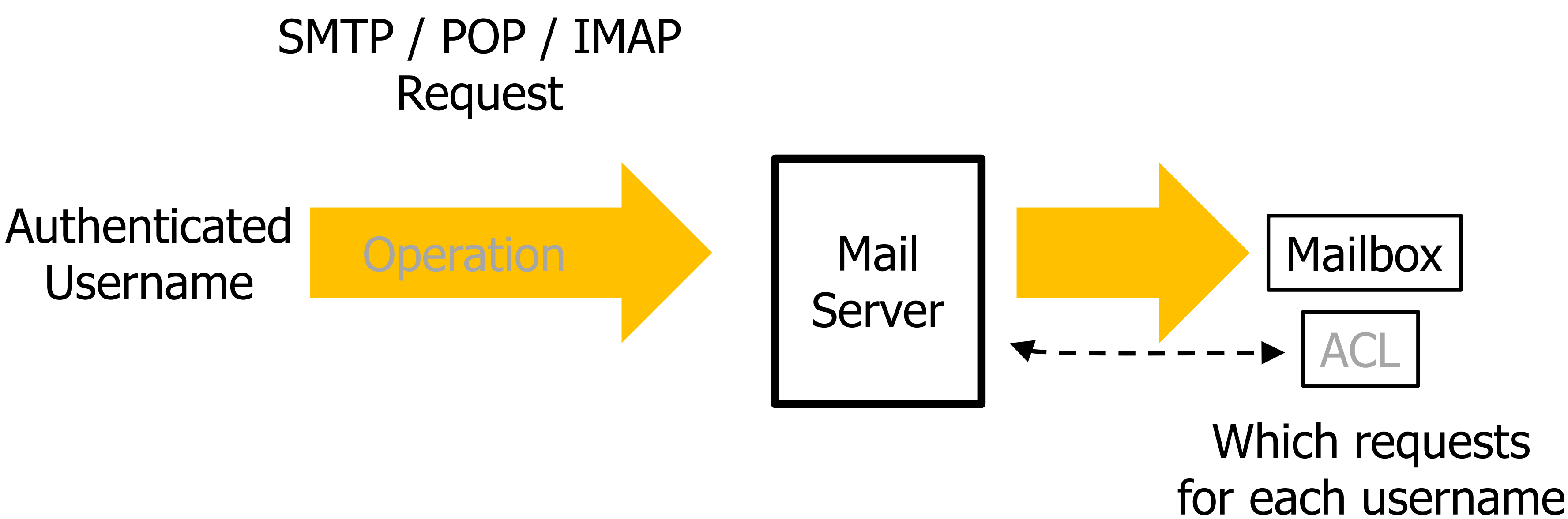
```
<security-constraint>
        <web-resource-collection>
                <url-pattern>/dir2/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
                <role-name>tomcat</role-name>
        </auth-constraint>

</security-constraint>
```

- Tomcat web server

- Resources
- Nothing to do with those of the local o.s.

Access Control – Mail Server



Access control must be implemented
in the **application**

Programmed and/or Configured

Access Control: O.S. vs Applications

- Operating system
  - Resources and Identities
  - **Mediates every resource access**
- Application server
  - Resources and Identities
  - **Mediates every resource access**

- **Independent of each other**

- Identities / Resources of the application server may have **nothing** to do with Identities / Resources of the o.s.