Security Policy

Important questions (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 questions (II)

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

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

Important questions (III)

- Smartphone
- Banking app
- Gaming app

- Can the Gaming app read the authentication token of Banking app?

Security Policy (I)

- **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

Even more important questions (I)

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

- You execute "your code" P on a PC
- How can you make sure that P cannot modify the internal code/data of the o.s.?

Even more important questions (II)

- esse3 webapp
- Student S1 logged in

- How can you make sure that S1 cannot see data of other students?
- ...modify grades?

Security Policy (II)

- **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
- And how they are **enforced**

Roadmap

1. How described, in an idealized way
2. How enforced
3. How described, in a more realistic way

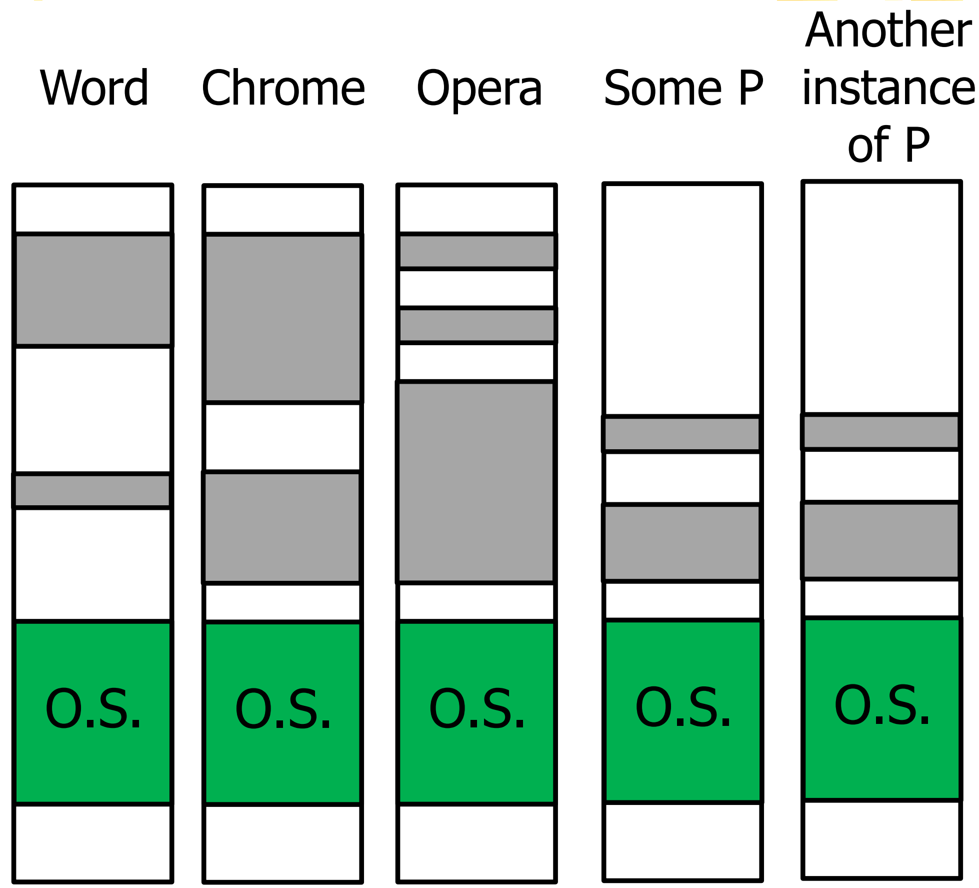
- Several important / fundamental observations

- Very simplified (many details omitted)

O.S. Protection (in a nutshell)

Process Address Space (I)

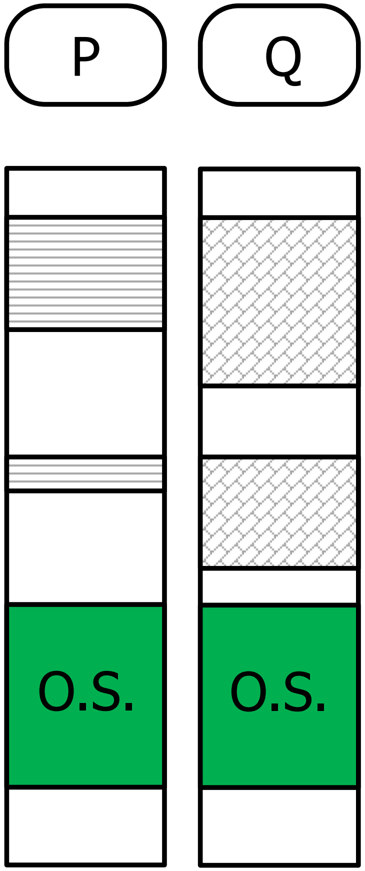
- "The executed program" (**user-level** code)
- Operating System (**system-level** code)
- Loaded at bootstrap



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 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 G
    ⇒ 2^32 * 2^12 G
    ⇒ **4 * 10^9 * 1024** G

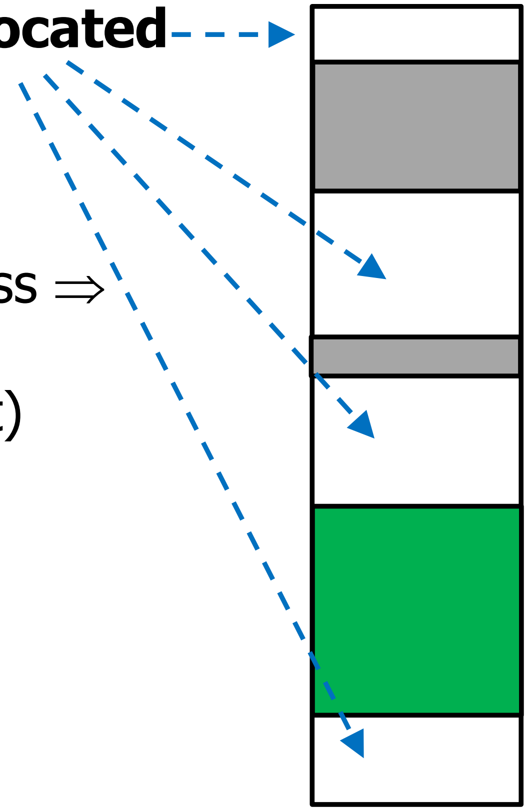
- **Physical** address space size
  - How much memory does your PC have? Maybe **16** GB?

- **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)

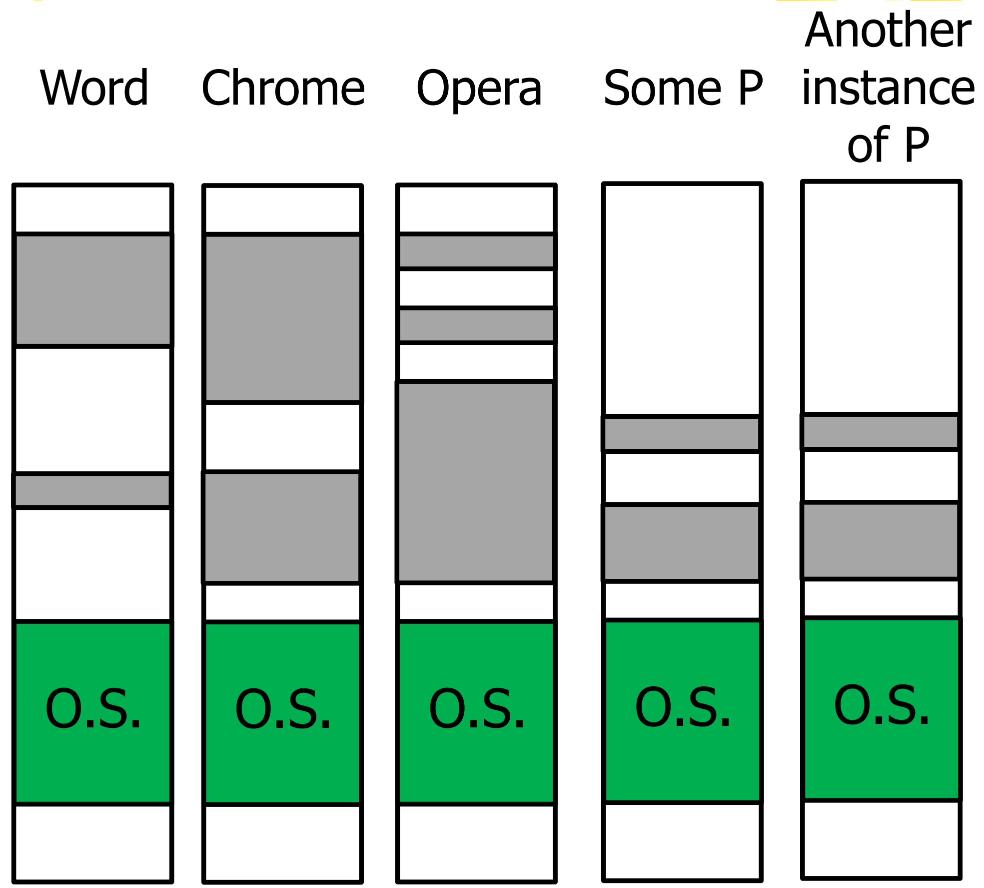


- I am neglecting swapping on secondary storage for simplicity...

Operating System

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

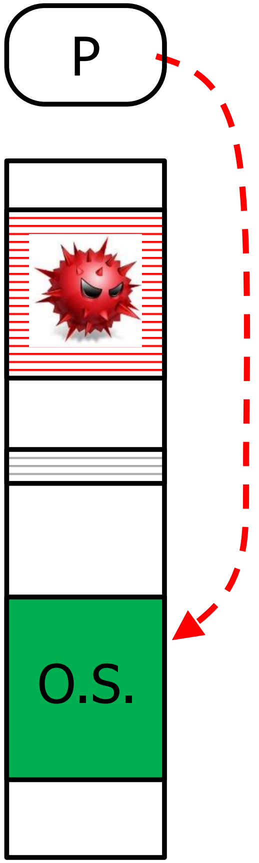
- Code
- Variables
  - Open sockets
  - Open files
  - "Permissions"
  - ...



Hhmmm...

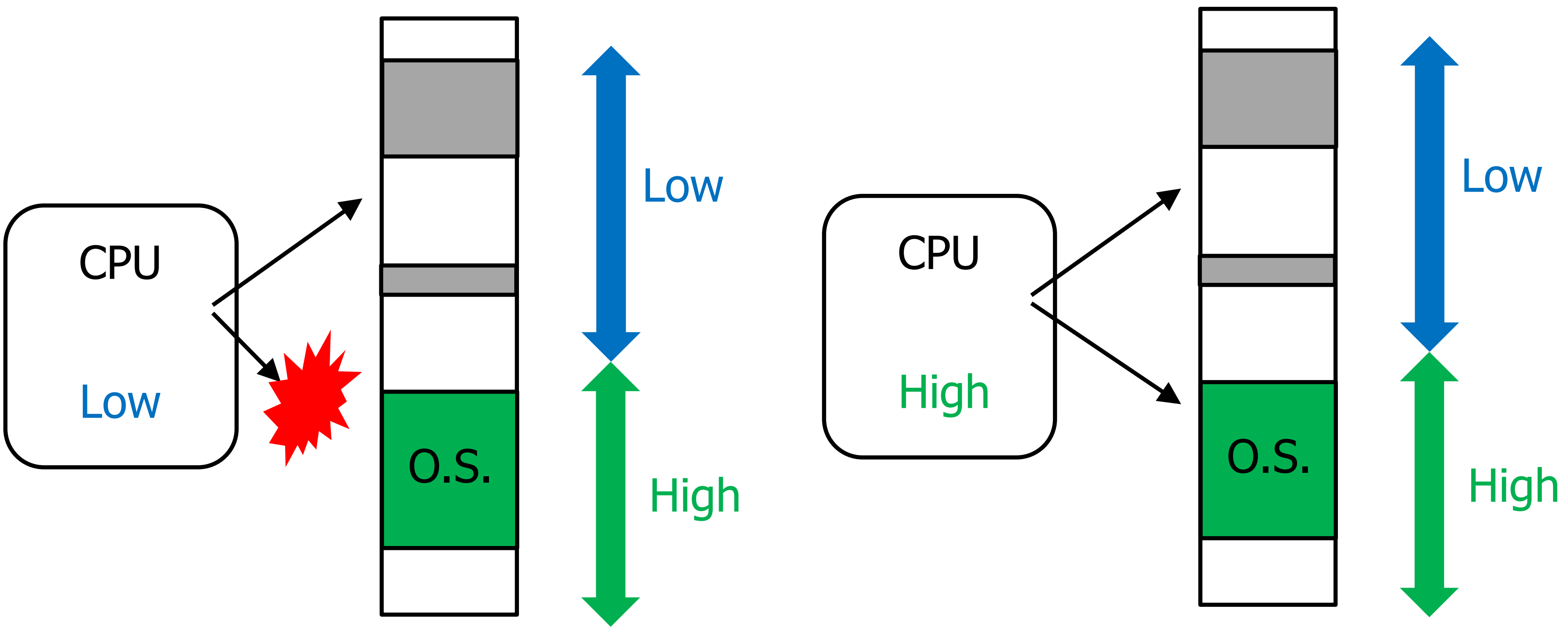
- A malicious process could attempt to:
  - Read o.s. variables
  - Write o.s. variables
  - Jump to arbitrary o.s. addresses

  - 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



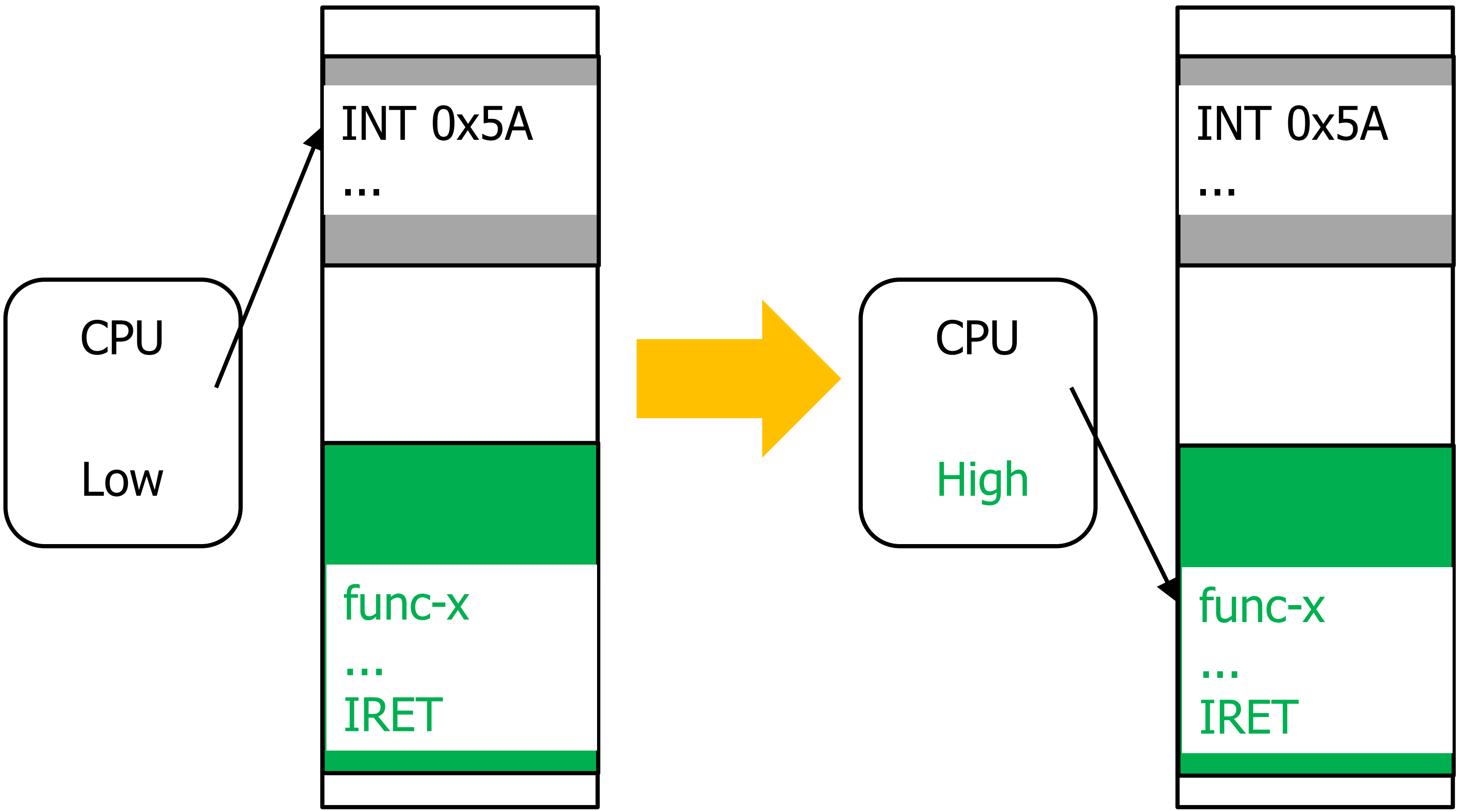
CPU Privilege Level: Privilege Switch

- Privilege level switch occurs in **hardware**

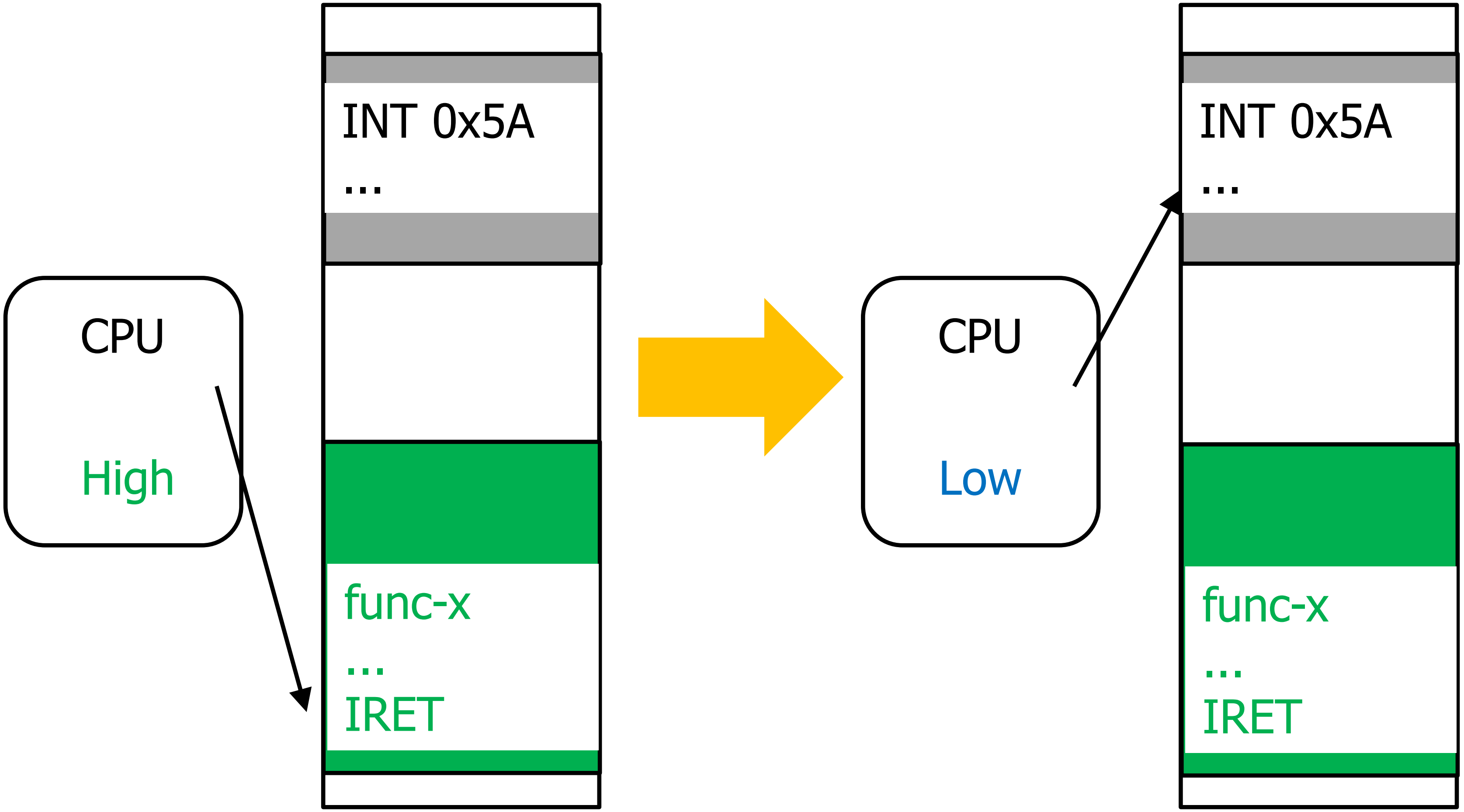
- **Low → High**
  - `INT operand` Calls a function in the o.s.
  - Mapping `operand` values → functions **predetermined** by the o.s.

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

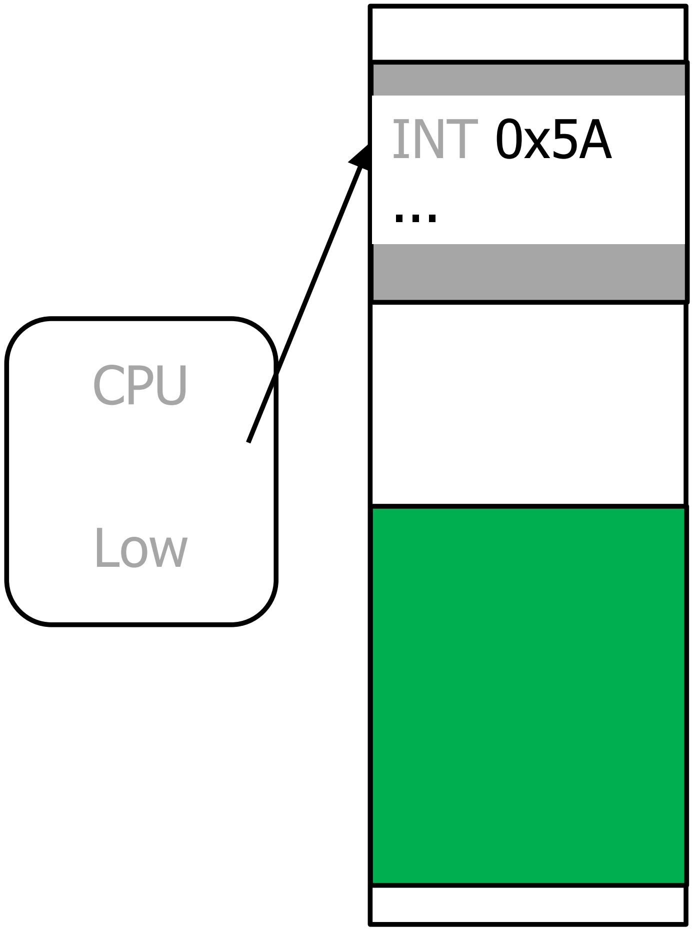
System Call Invocation



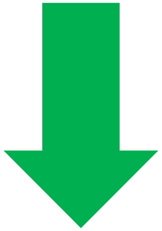
System Call Return



Remark



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

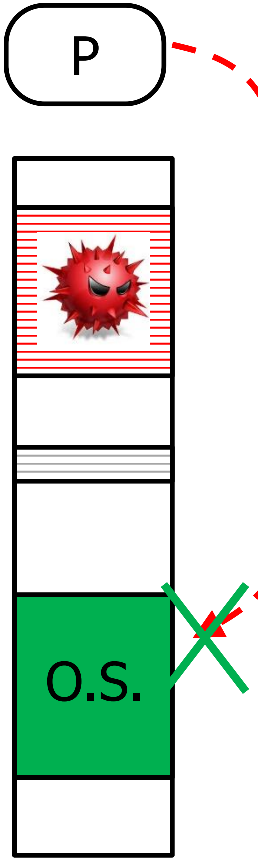


- Untrusted code can **only** call **predefined** addresses

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)



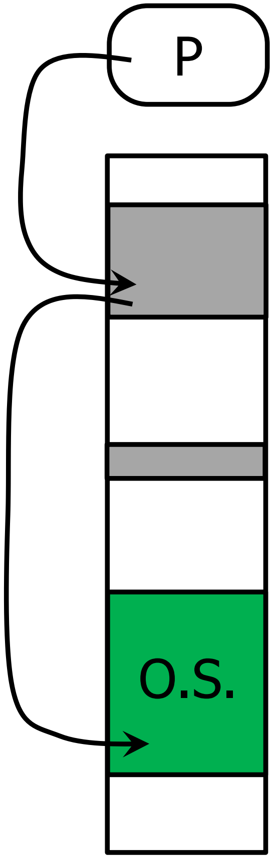
Keep in mind

- 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

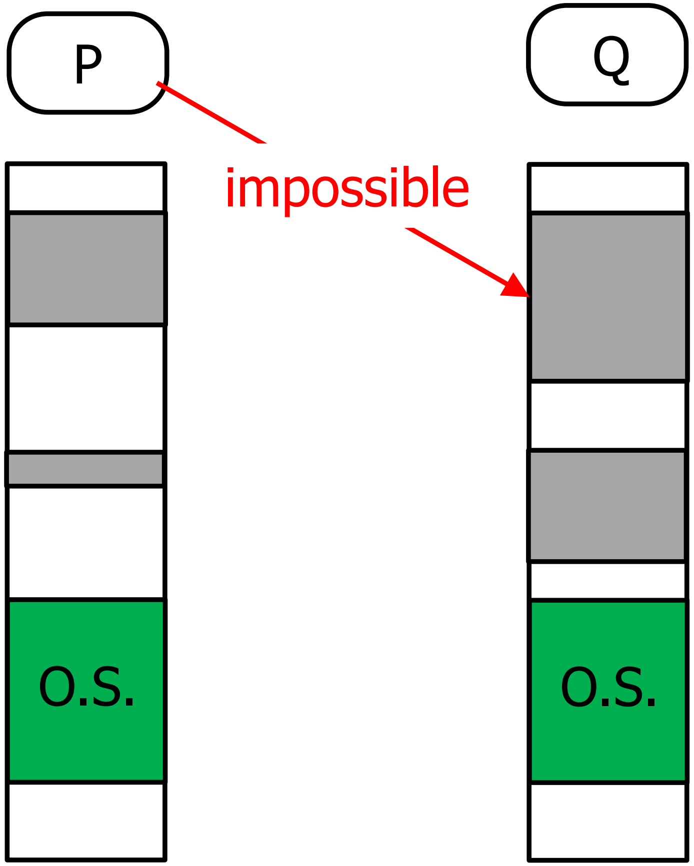
- Every **resource** is implemented by the o.s.
  - File
  - Socket
  - Screen
  - Process management
  - Access rights
  - ...



- 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

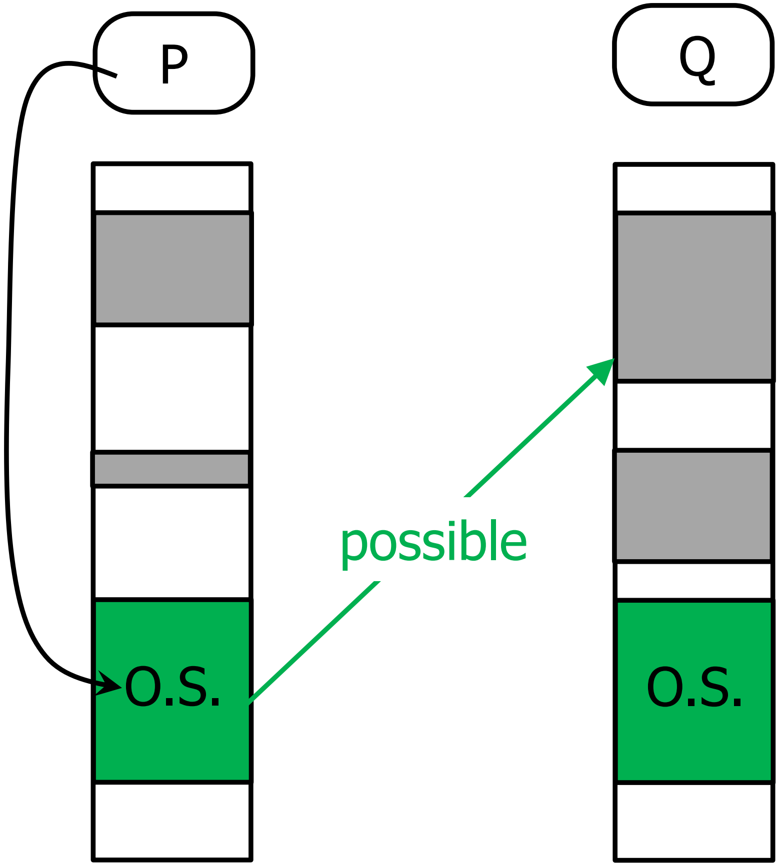
Isolation (I)

- 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
  - p-address
  - how-many
  - Q
  - q-address
- The o.s. decides whether to **grant** or **deny** the operation



Accounts and Resources

Account ("User")

- **Account**: Every **identity** in the system
  - **Username** (string)
  - **Credentials** for the initial authentication
  - **Internal identifier** used by the o.s.

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

Process ↔ Account

- 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

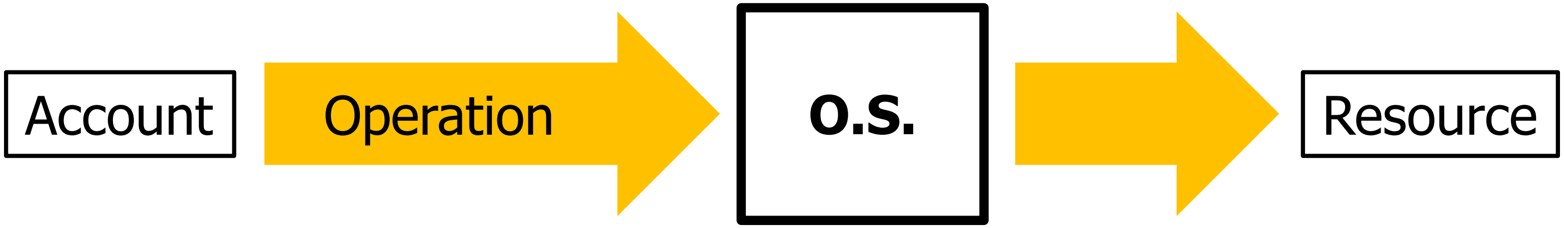
  - Child Process: **same** Account as the Parent process
  - Special case:
    Process of Root/System can choose **any** Account for its children

Resource

- **Resource**: Every **"object"** in the system
  - File
  - Socket
  - Process
  - I/O device
  - ...
- Every **resource access** occurs through a **System Call**
  - Process invokes a system call
  - Parameters specify which operation on which resource

Access Control "Model" (preliminary)

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



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

Resource ↔ Account

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

- The owner of a resource decides who can do what on the resource

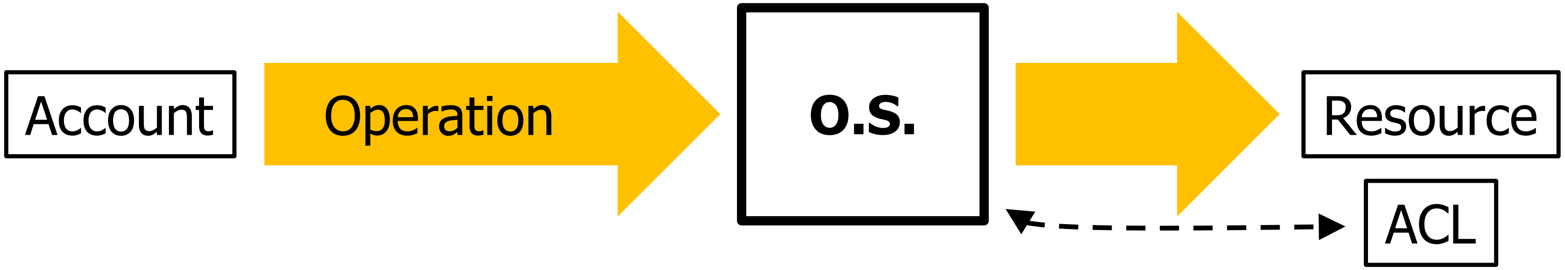
Resource $\leftrightarrow$ ACL

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

- System Call execution decides whether to **grant** or **deny**:
  - Input: Account, Operation, Resource
  - O.S. data: Resource.ACL

- Resource.Owner controls Resource.ACL
  - Operations that modify R.ACL are granted to R.Owner
  - R.owner might decide to grant other accounts the rights to modify R.ACL ("with constraints")

Access Control "Model"



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

- Think in terms of this model
- Not of how it is implemented
  - Process invokes System call
  - Low / High CPU privilege
  - . . .

"High Privilege" Account

- Each o.s. has one or more **predefined** accounts with **"high privilege"**
  - Linux `root` (internal id `0`)
  - Windows `NT Authority/SYSTEM` (internal id "complex")
  - Windows `Administrator` (internal id "complex")

- ≈ They can execute **every** operation on **every** resource
  - Linux: operation requests issued by `root` are granted irrespective of the content of the ACL
  - Windows: every ACL grants full control to `SYSTEM` and `Administrator`

Windows: Security Identifier (SID)

- **Process identifier** for **access control** decisions
- **String** whose structure has a certain semantics

- High privilege SID:
  - `Administrator`
    `S-1-5``-21-1559272821-92556266-1055285598-`**`500`**
  - `NT AUTHORITY/SYSTEM`
    `S-1-15-18`

- Groups also have a SID

High Privilege Account: What it means

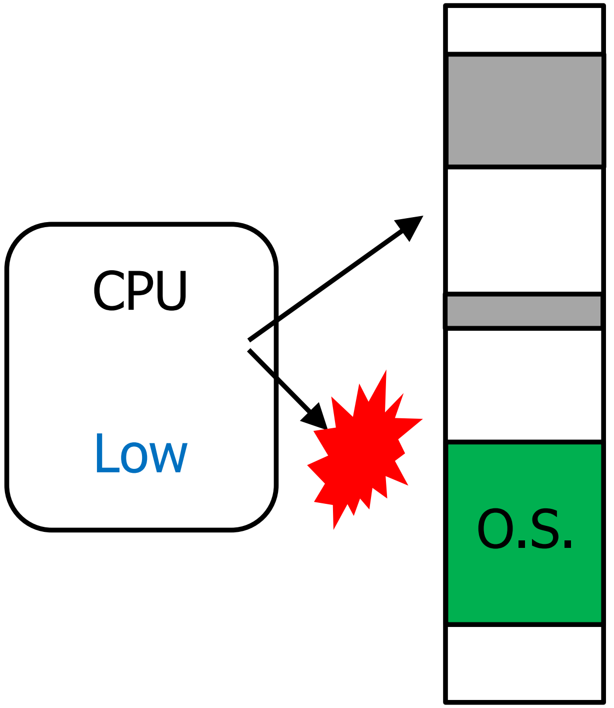
- ≈ They can execute **every** operation on **every** resource

- ≈ 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"

High Privilege Account: What it does NOT mean

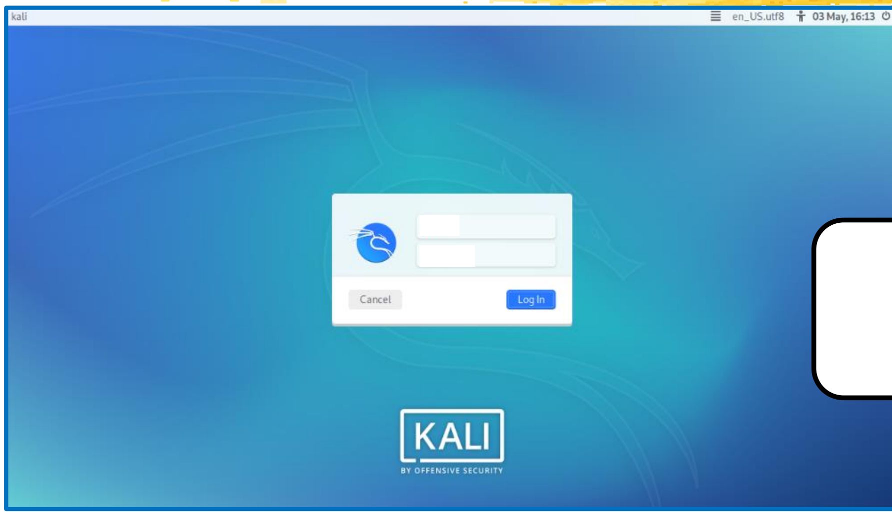
- ~~Can access every memory address~~



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

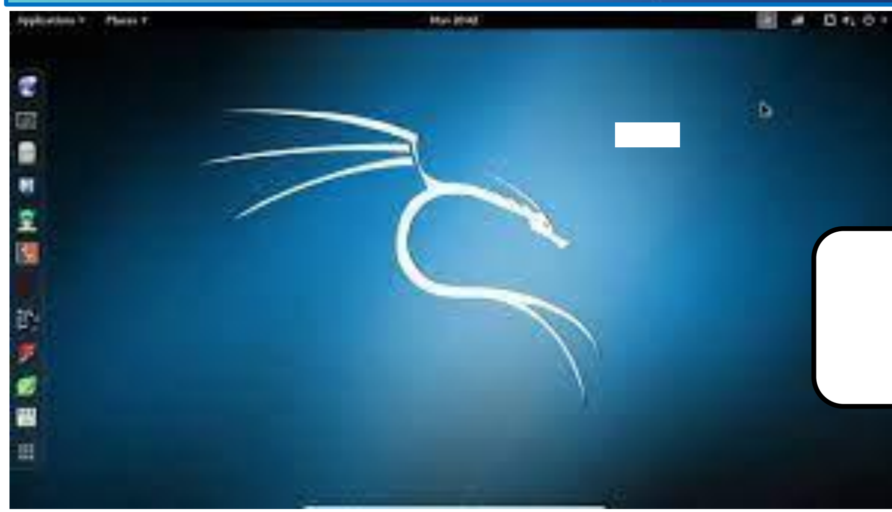
Understanding Account $\leftrightarrow$ Process

Account ↔ Process: Interactive Logon



logon process

account?



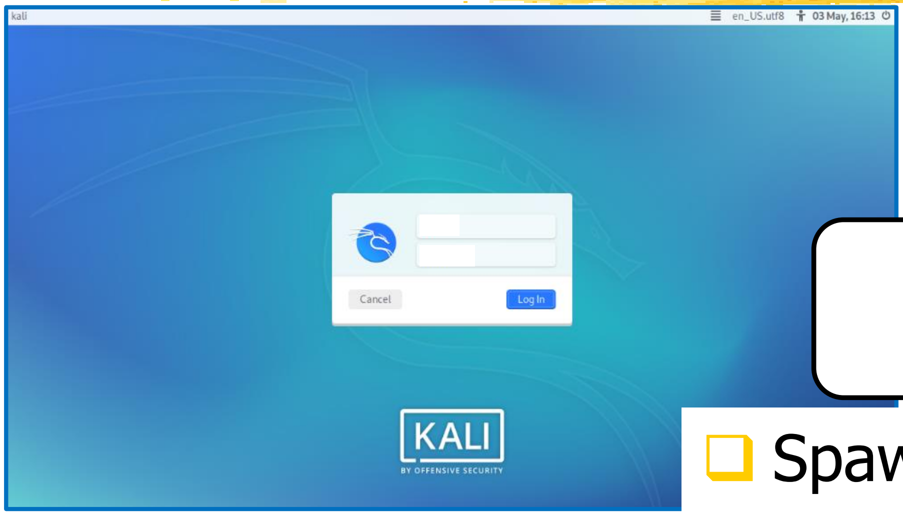
GUI

account?

Bootstrap

- **First** process:
  - Associated with an account with **high privilege**
  - Spawns many **child processes** (usually servers)
  - Child processes can change account **at their will** (because they start with high privilege)
    - Usually accounts of **lower** privilege
  - Configuration information describes which servers and which accounts

Interactive Logon (I)



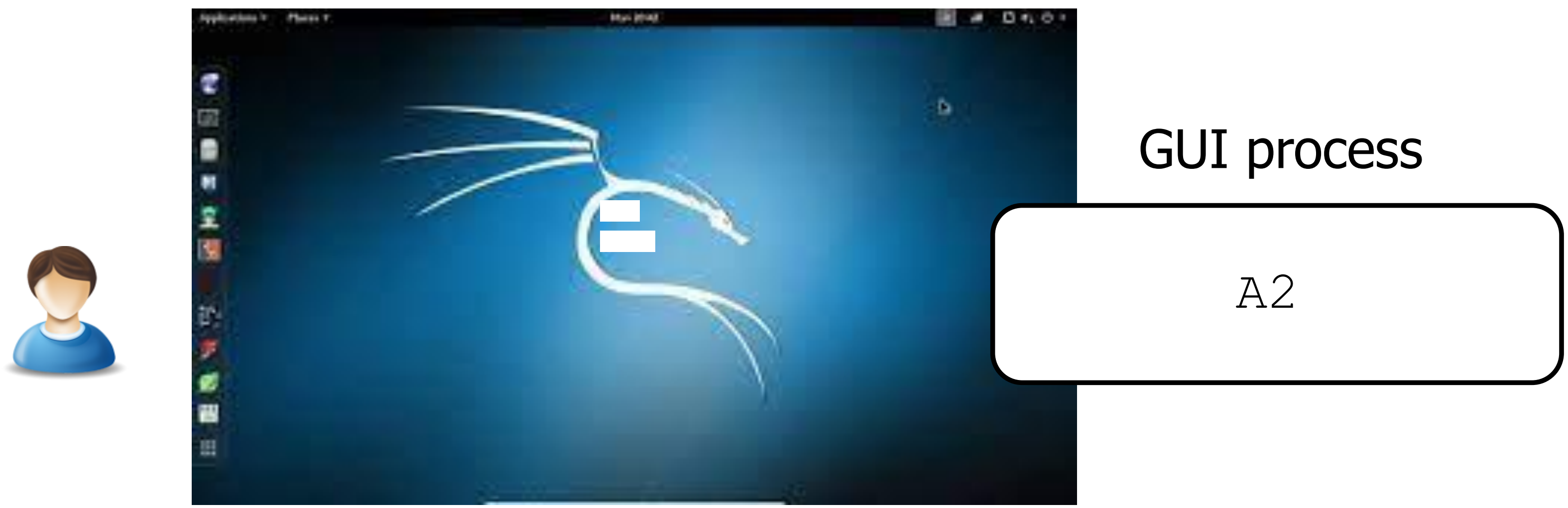
logon process

root / SYSTEM

- Spawned during bootstrap
- Account with high privilege

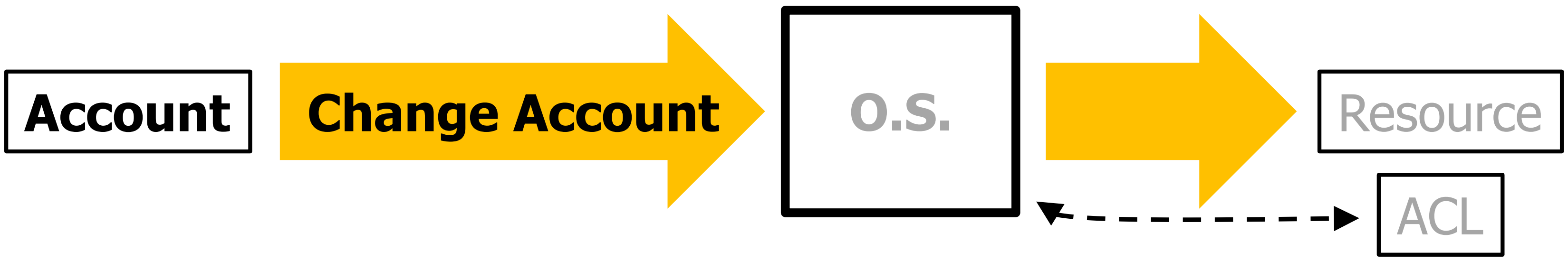
1. Wait for credentials
2. ...
3. ...

Interactive Logon (II)



1. Wait for credentials
2. Validate credentials (authenticate account A2)
3. Spawn GUI process that changes account to A2

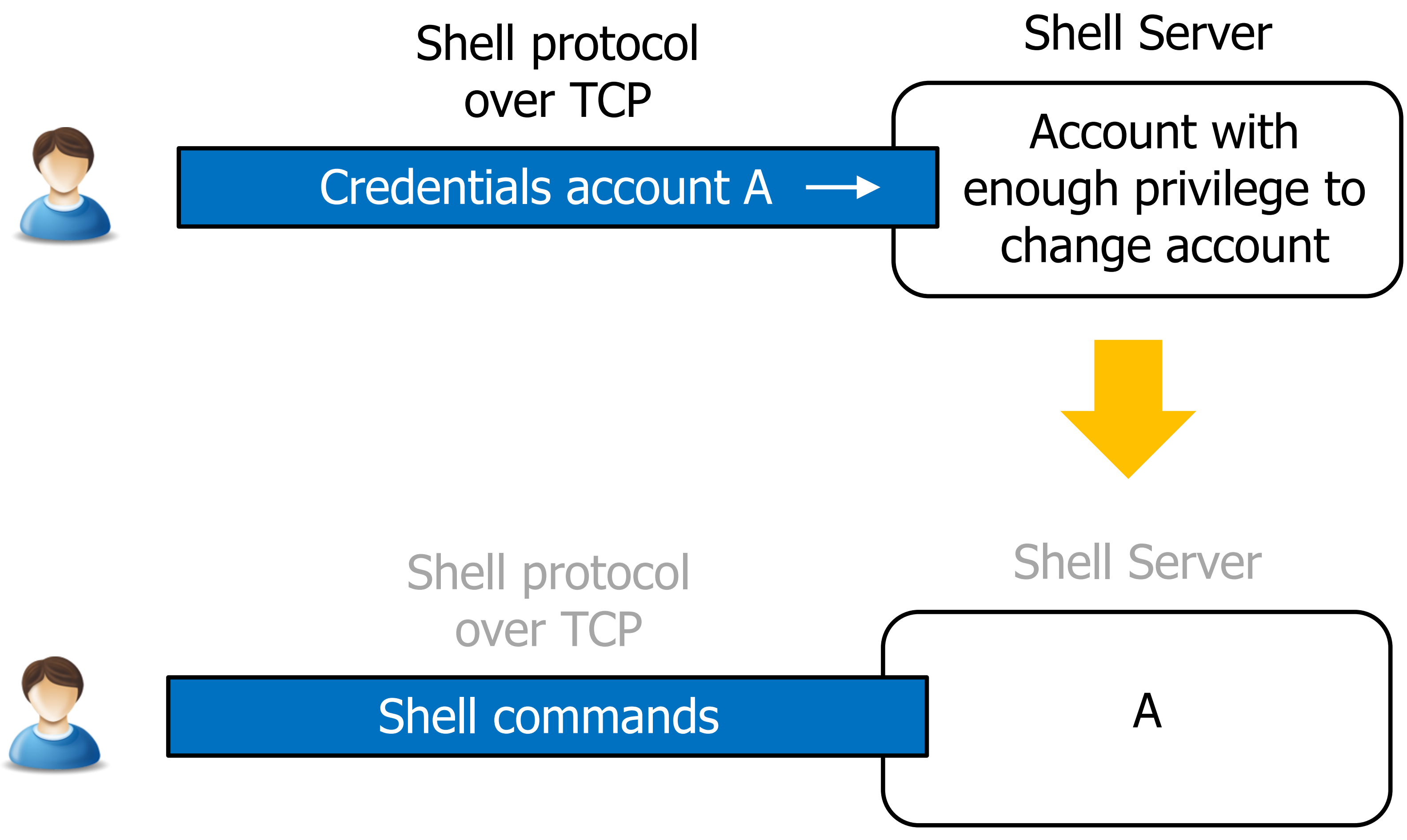
Changing Account



- Allowed only to **high privilege** accounts

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

Account ↔ Process: Remote Shell



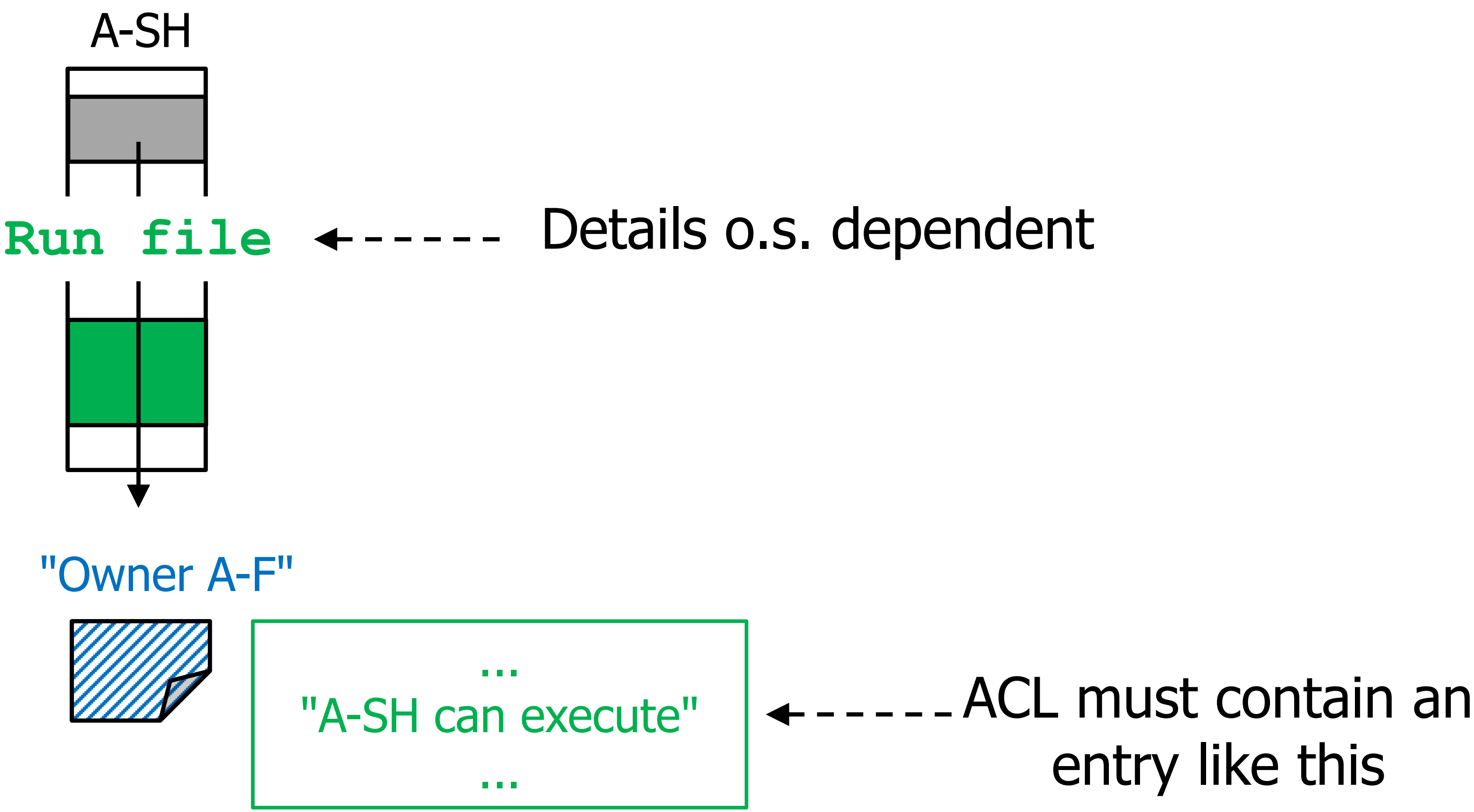
Crucial Scenario: Command Execution

- Shell / GUI associated with A-SH

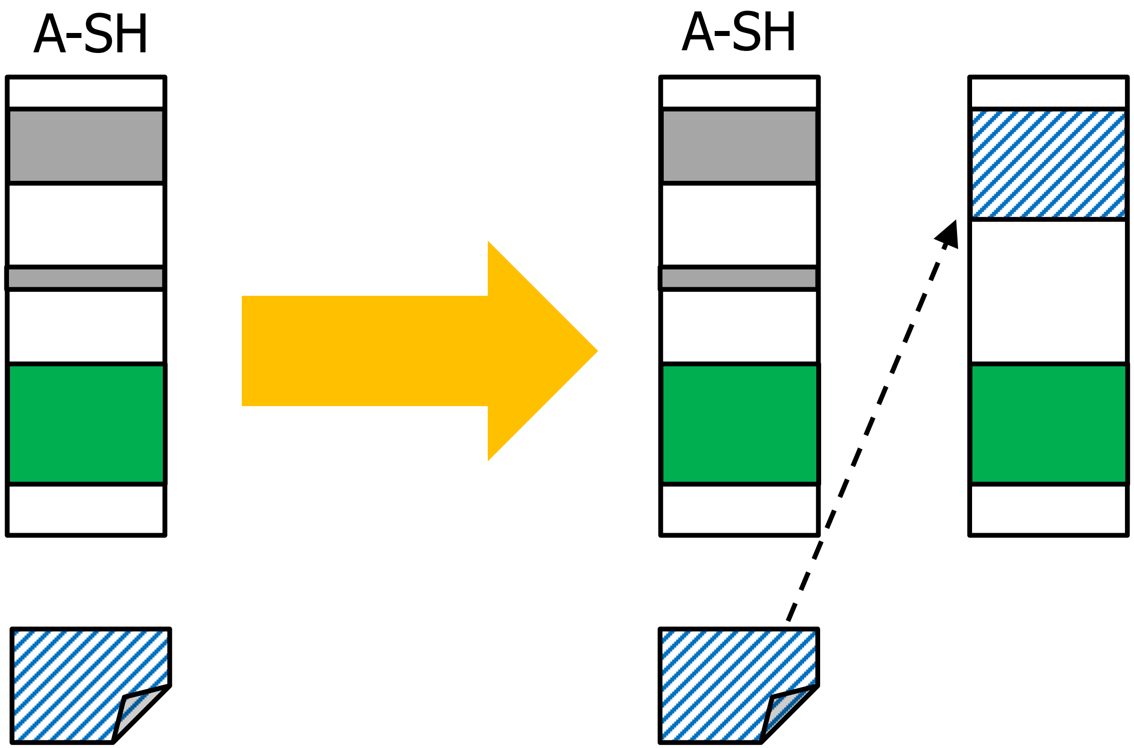
1. Executes command/program in file F owned by A-F
2. ...that creates a file R

- What happens in terms of processes and accounts?

1: Shell / GUI executes F (I)

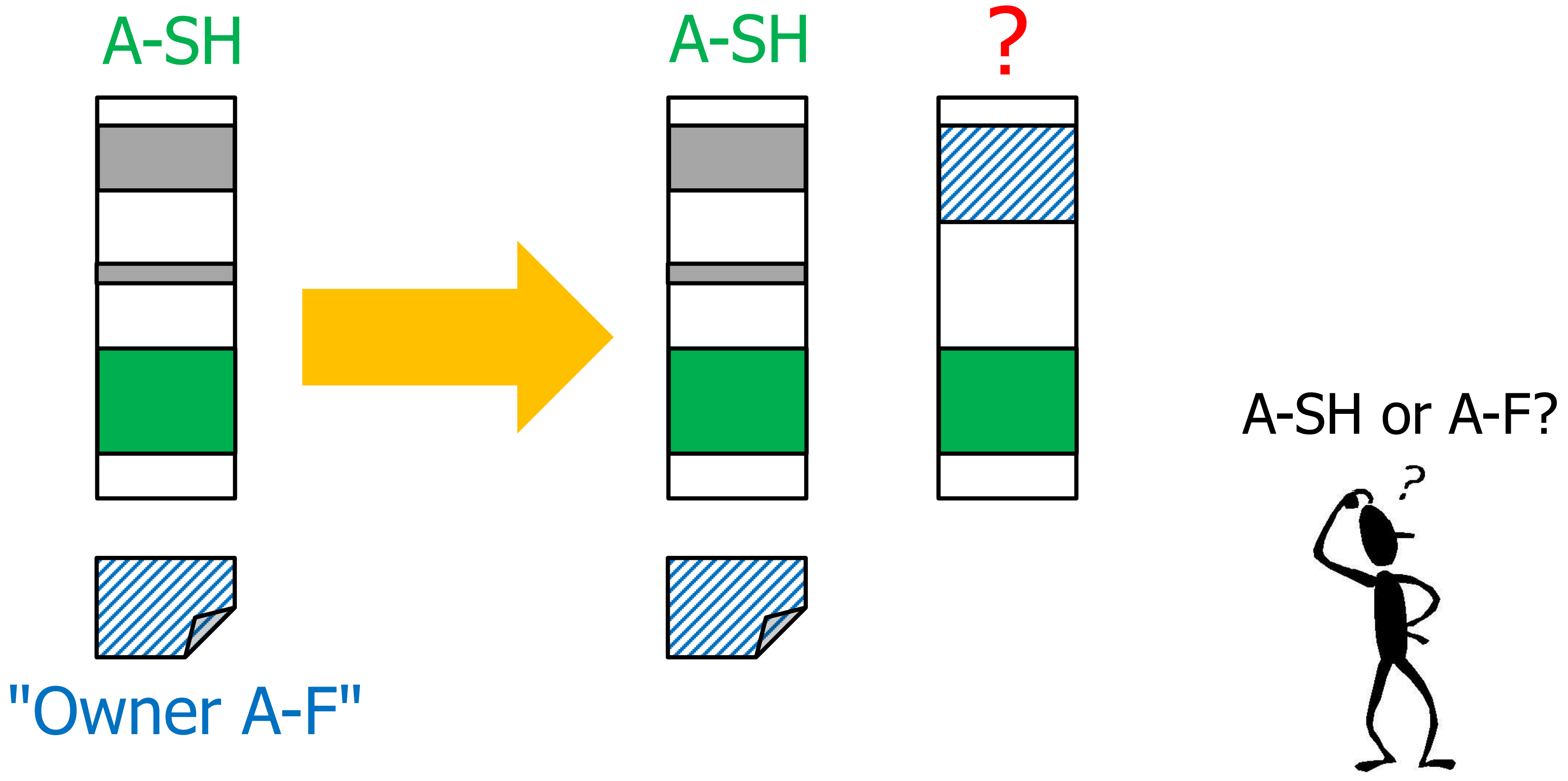


1: Shell / GUI executes F (II)

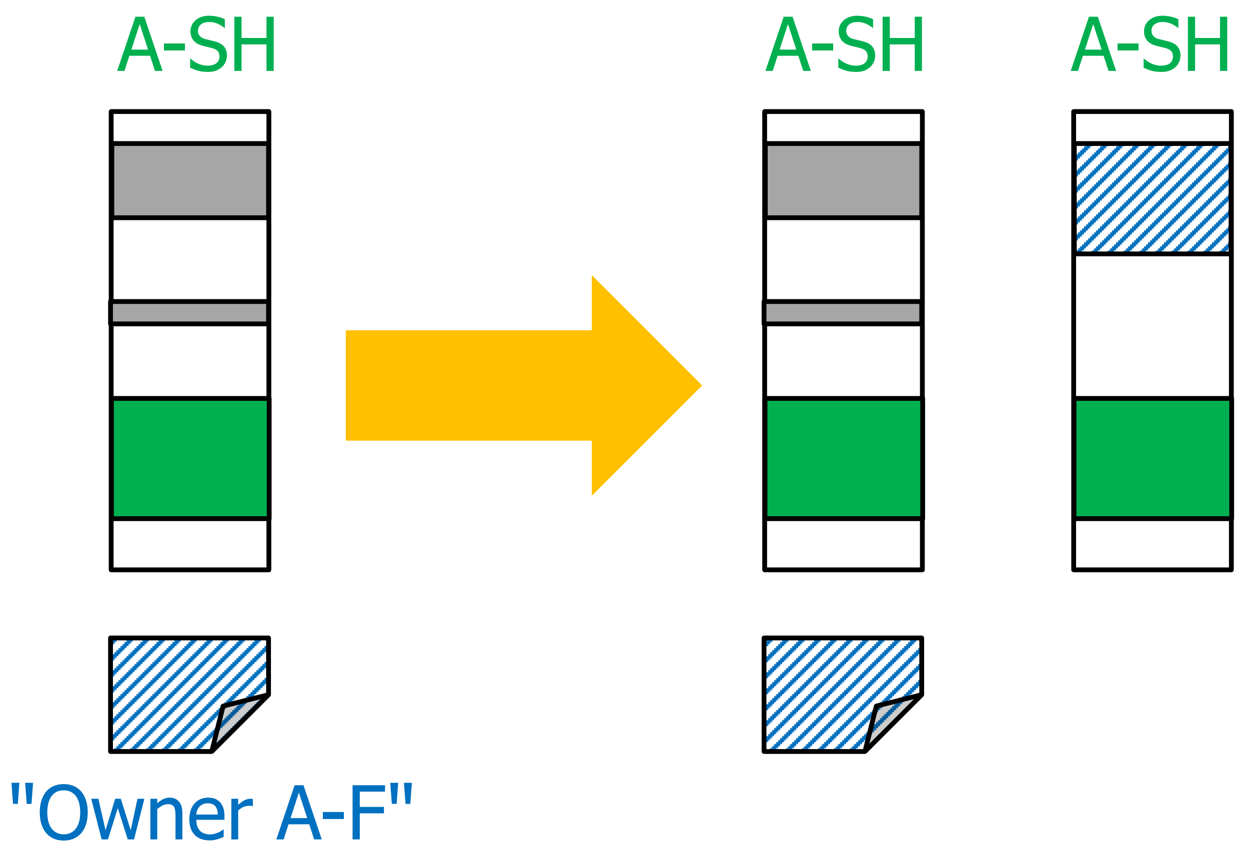


- **New** process
- VM contains file content (≈)

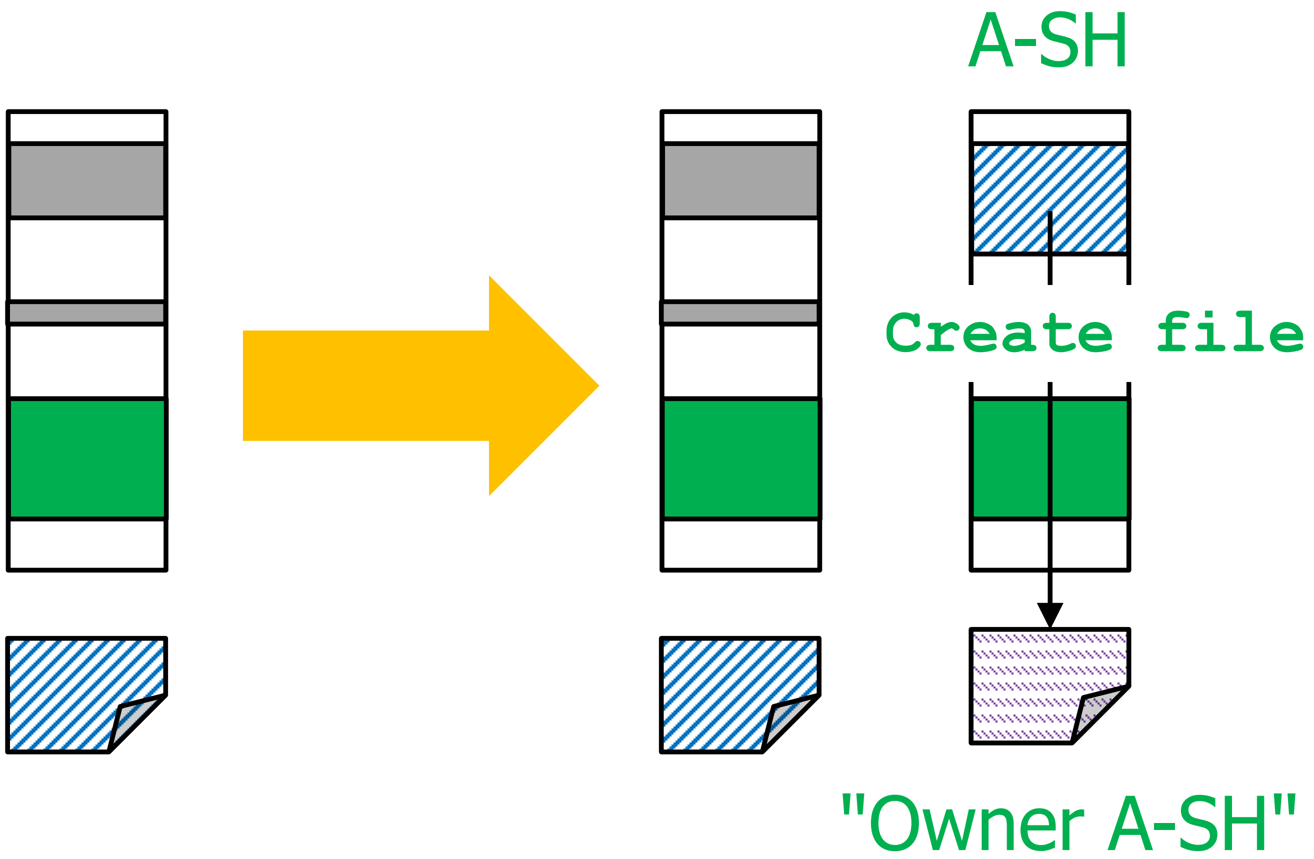
Account?



Child = Parent



2: Child creates resource



Important Remark

- Shell / GUI associated with A-SH

1. Executes command/program in file F owned by A-F
2. ...that creates a file R

- One process for each command
- "**Shell identity everywhere**" (processes, created resources)
- The owner of the executable files is **irrelevant**

- Except for specific cases...

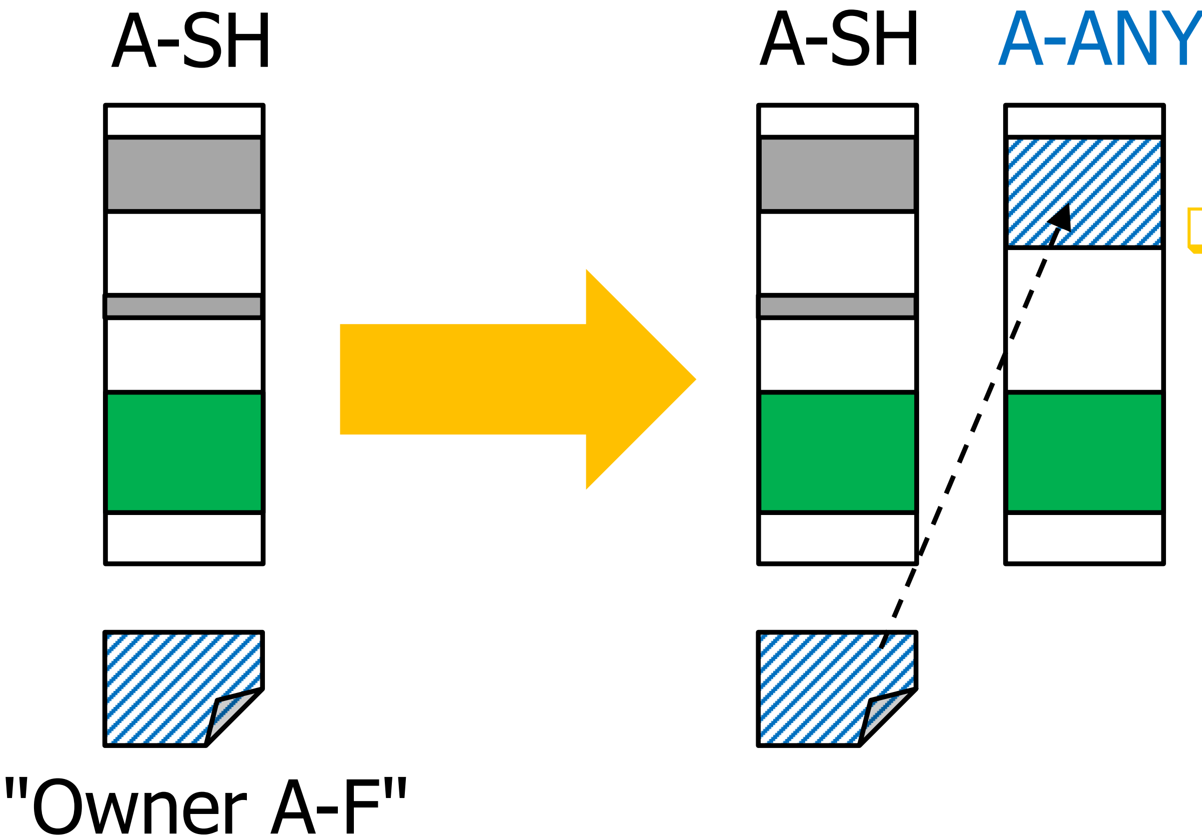
Linux `suid`

Command Execution: Specific need (I)

- Shell or GUI process associated with A-SH
- Execute **one command** with a **different** account
  - **Password** of the destination account **required**

- Temporary **impersonation**

Command Execution: Specific need (II)



- Knowledge of A-ANY password required

Solution (in a nutshell)

- Shell or GUI process associated with A-SH
- Execute **one command** with a **different** account
  - **Password** of the destination account **required**
- Temporary **impersonation**

- Linux `sudo`
- Windows `Run as Administrator`

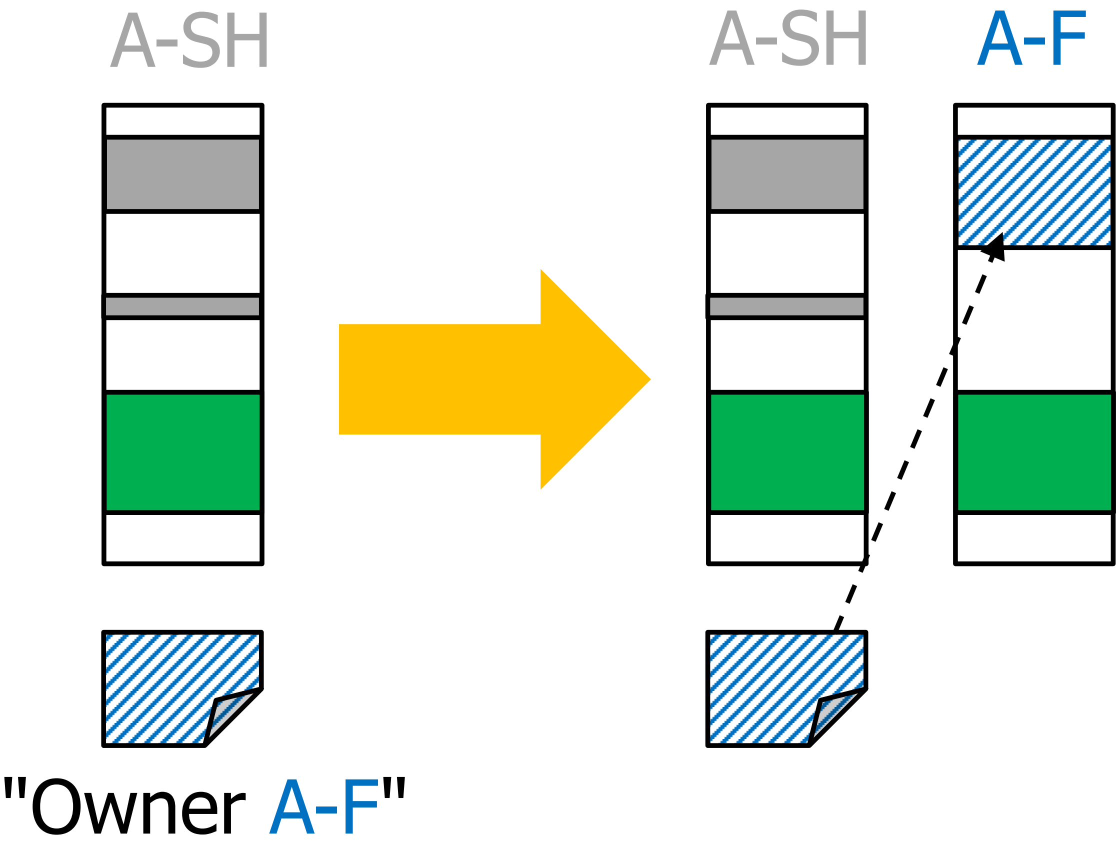
- Various configurations / constraints possible
  (e.g., multiple commands)

Command Execution: More Specific need (I)

- Shell or GUI process associated with A-SH
- Execute **one command** with a **different** account
- Temporary **impersonation**
  - Account of the **owner** of the command file
  - **No password required**

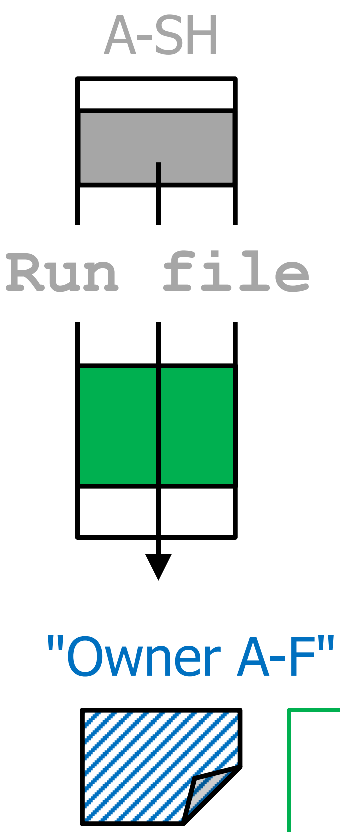
- Different point of view:
  - A-X encodes certain actions in a program
  - Everyone can execute those actions **as A-X (without A-X password)**

Command Execution: More Specific need (II)

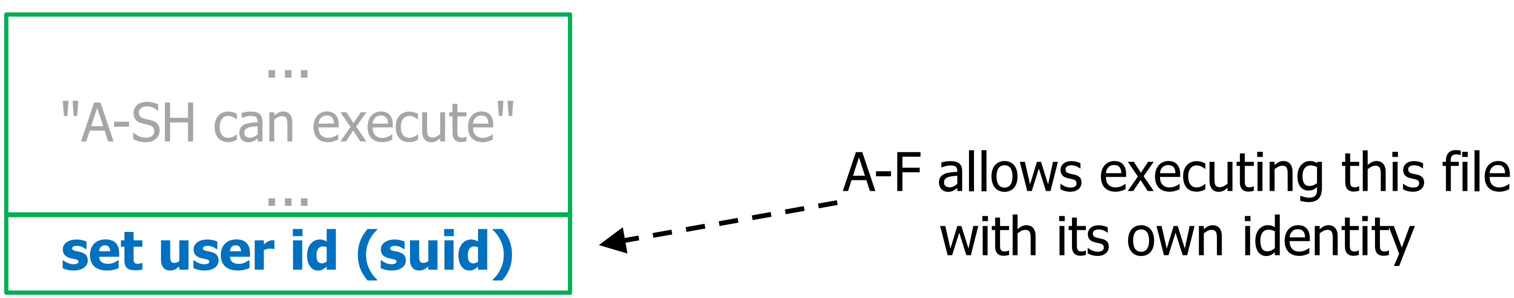


- Knowledge of A-F password **not** required

Linux `suid`



- Executable file F with `suid` bit set in ACL ("**set user id**"):
  - Executed with the account of its **owner**
  - **Without** providing its credentials



Common Use Case

- Different point of view:
  - A-X encodes certain actions in a program
  - Everyone can execute those actions **as A-X (without A-X password)**
  - A-X is **high privilege**

- Example commands:
  - Mounting a disk
  - Changing the password of the shell user
  - ...

Interesting Question

- Shell A-SH
  - Its children are A-SH
  - Command `sudo` is a child

- How can `sudo` take a **different** identity?

How `sudo` works (outline) (I-a)

```
┌──(kali㉿kali)-[~]
└─$ which sudo
/usr/bin/sudo

┌──(kali㉿kali)-[~]
└─$ ls -l /usr/bin/sudo
-rws            root root 261080 Oct 10  2022 /usr/bin/sudo
```

Executable file
with "set user id"

Owned by the
`root` account

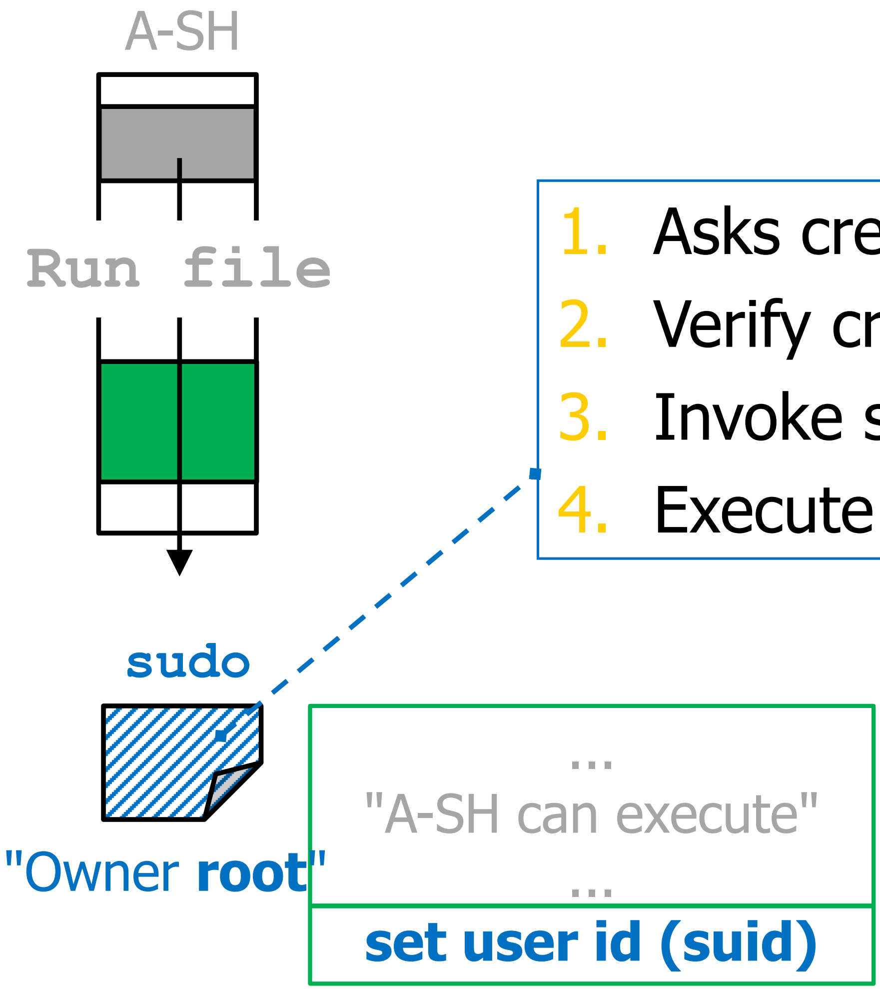
How `sudo` works (outline) (I-b)

```
┌──(kali㉿kali)-[~]
└─$ which sudo
/usr/bin/sudo

┌──(kali㉿kali)-[~]
└─$ ls -l /usr/bin/sudo
-rwsr-xr-x 1 root root 261080 Oct 10  2022 /usr/bin/sudo
```

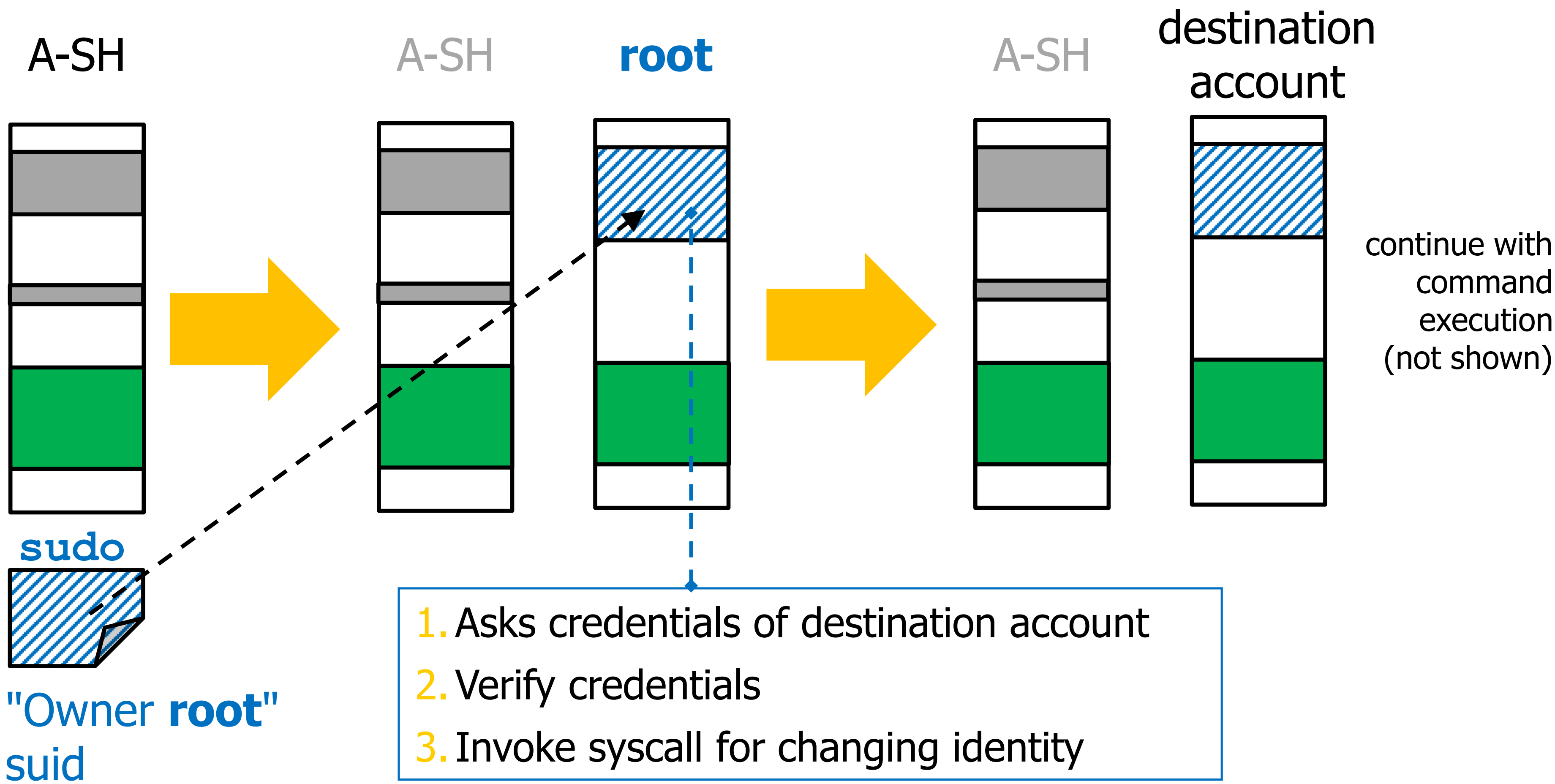
Can be read and executed
(but **not modified**)
by any account

How `sudo` works (outline) (II)



1. Asks credentials of destination account
2. Verify credentials
3. Invoke syscall for changing identity
4. Execute command

How `sudo` works (outline) (III)



1. Asks credentials of destination account
2. Verify credentials
3. Invoke syscall for changing identity

Linux `suid` summary

- **Temporary privilege elevation without credentials**
  - It works for any owner...typical usage is for high privilege

- Example application: sudo

- Risk: behavior might not be as intended
  - Mistakes
  - Vulnerabilities

Back to the Important questions

Important question (I) (REMIND)

- PC
- Dropbox app
- Chrome browser

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

Answer in a nutshell

- Dropbox app and Chrome browser are Processes associated **with the same Account**
- **Any** operation allowed for **one** Process is **also** allowed for the **other** Process
  - ACL: (**Account**, Operation)
- Dropbox can read/modify anything that Chrome can read/modify

Important question (II) (REMIND)

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

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

Answer in a nutshell

- Process that opens the email attachment and Chrome are Processes associated **with the same Account**
- Same reasoning as before
- Each process can perform the **same** operations as the other

Important question (III) (REMIND) + Answer

- Smartphone
- Banking app
- Gaming app

- Can the Gaming app read the authentication token of Banking app?

- As far as we know so far: Yes

Keep in mind 1

- ACLs have the form (**Account**, Operation)

↓

- ACLs do not distinguish between **different commands** with the same **same account**

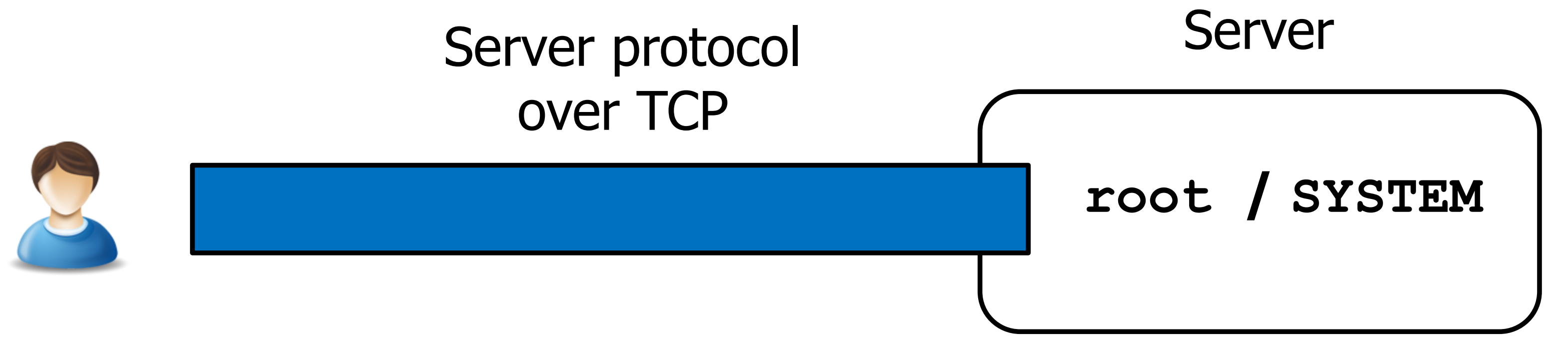
- All processes with the same account can do the same things
- Irrespective of who developed their code

Keep in mind 2

- **Account A takes a malware M**
- **M can perform anything that A can perform**
- M may be more or less sophisticated
- ...but in principle it can perform anything:
  A is (potentially) fully disrupted

Principle of Least Privilege

Common Server Config. (up to a few years ago)



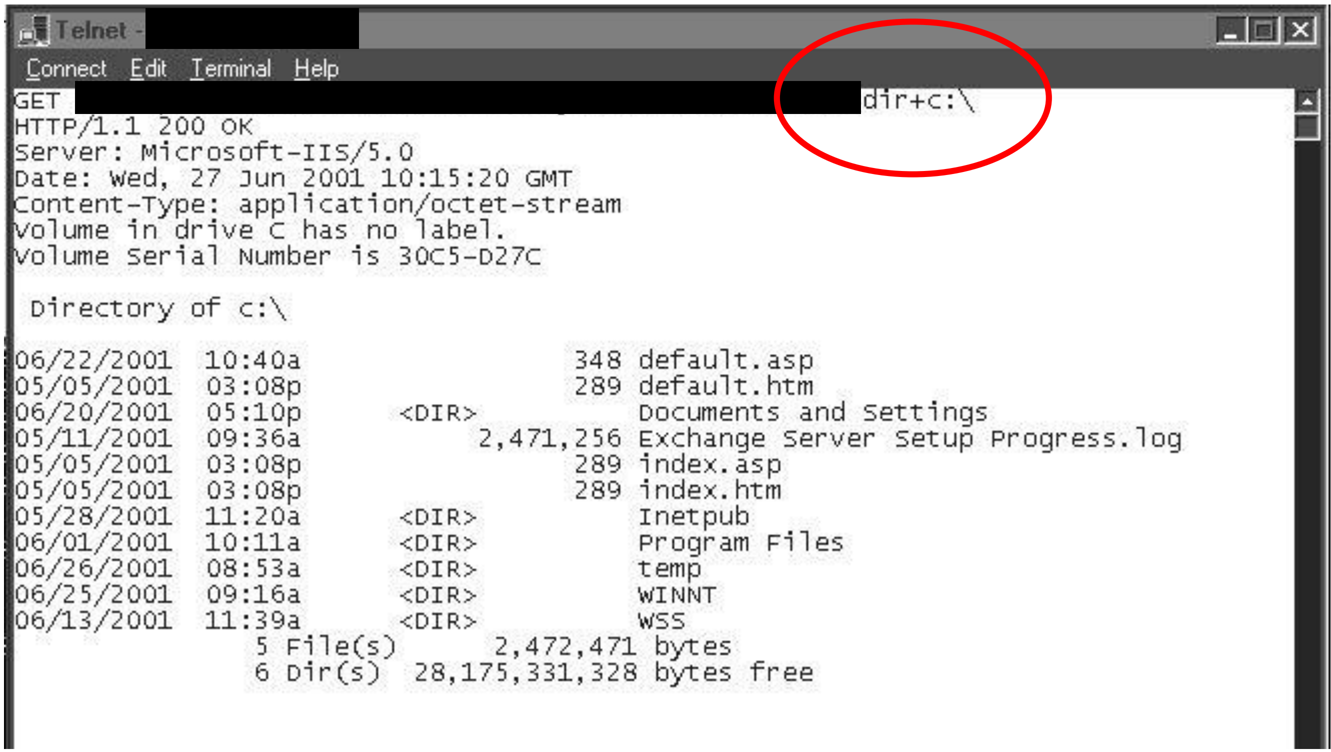
Remote Shell
Web Server
File Server
Mail Server
...

Example
(Old but interesting) (I)

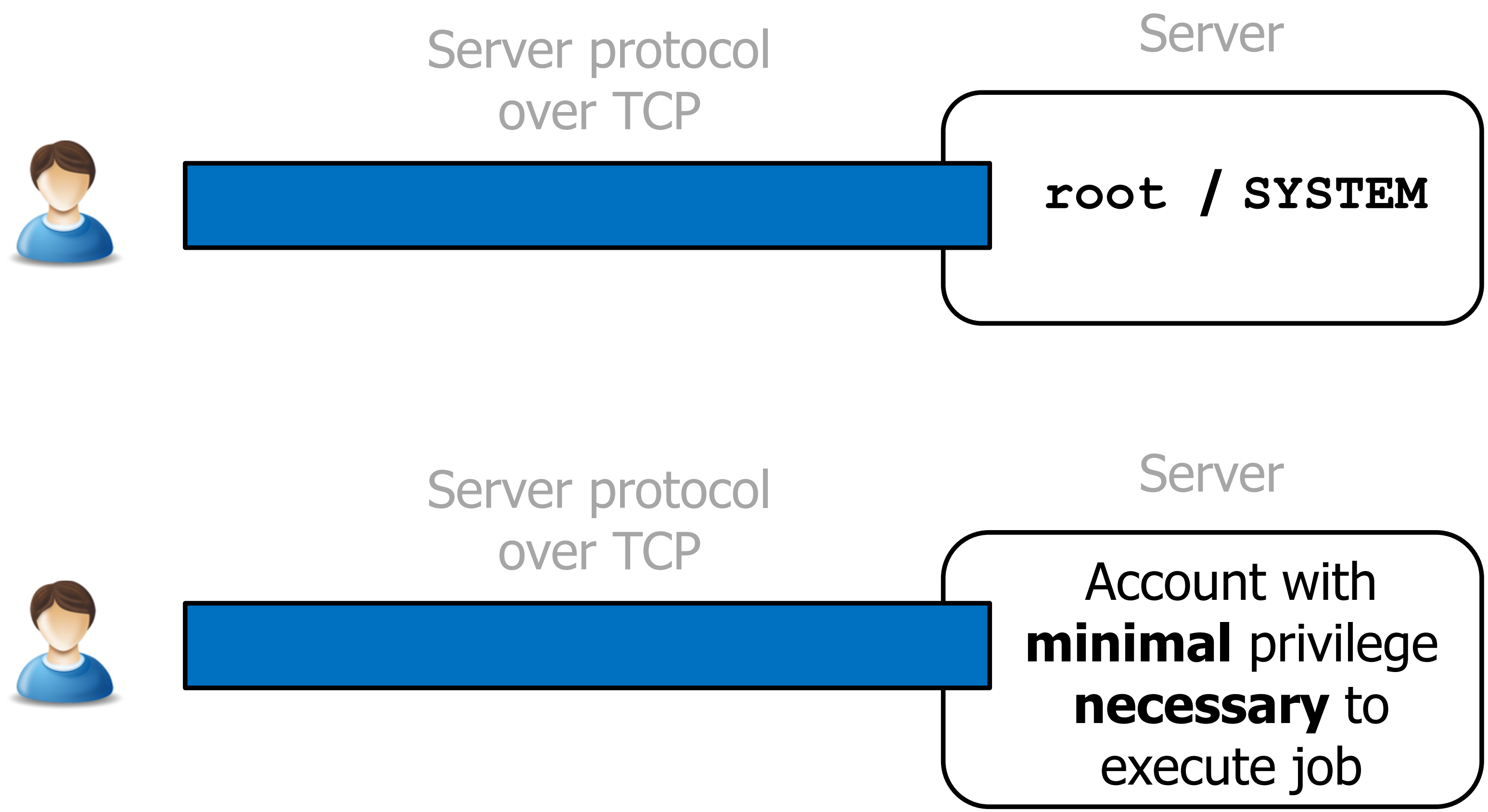
HTTP Request
with "long and wrong URL"
ending with command

Execute command

Example
(Old but interesting) (II)



Which approach is wiser?



Principle of Least Privilege

- **Every** program and every user of the system should operate using the **least** set of privileges **necessary** to complete the job...
- It also reduces the number of potential interactions among privileged programs to **the minimum for correct operation**, so that **unintentional**, **unwanted**, or **improper** uses of privilege are **less likely** to occur...

- *Saltzer and Schroeder* ***1974 (!)***

- Please take a moment to reflect and admire its depth and generality
- We will find more examples of its relevance

Microsoft Exchange (March 2021): Ouch!

- Mail Server used by **a myriad of organizations**
- **Necessarily exposed to the Internet**
- "Exchange is, **by default**, installed with **some of the most powerful privileges** in Active Directory" (`SYSTEM`)
- Several vulnerabilities. Their chaining leads to:
  - An **unauthenticated** attacker can **execute arbitrary commands** on Microsoft Exchange Server ("ProxyLogon")

EMERGENCY DIRECTIVES

ED 21-02: Mitigate Microsoft Exchange On-Premises Product Vulnerabilities

CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY

Cybersecurity & Economics

Hhmmm...

- *Principle of Least Privilege: **1974***
- *Why in many practical scenarios it is still **not** enforced, **50 years later?***

Security is NEVER the ONLY objective (I)

- **Every** choice must be a tradeoff among:
  1. Security
  2. Cost
  3. Functionality
- Design, Development, Deployment, Usage, Maintenance

- In many practical cases, Security is sacrificed

Security is NEVER the ONLY objective (II)

- In many practical cases, Security is sacrificed

- The chosen tradeoff might be wrong (perhaps retrospectively)

- ...but it often is **economically rational**
  - More Security $\Rightarrow$ More short term costs
  - Long term savings uncertain
  - Market forces could penalize short term costs

Think in Economical Terms

- To understand cybersecurity **never** think only in **technical** terms
  - Or, worse, in "moral" terms
- **Always** think in **economical** terms

- What is the cost?
  - Attack, Defense, Incident
- Who pays?

- **Money is what drives the world**
  - It may sound cynical...but thinking in these terms is very helpful

Key Practical Scenario: Administrators

Key Practical Scenario: Administrators

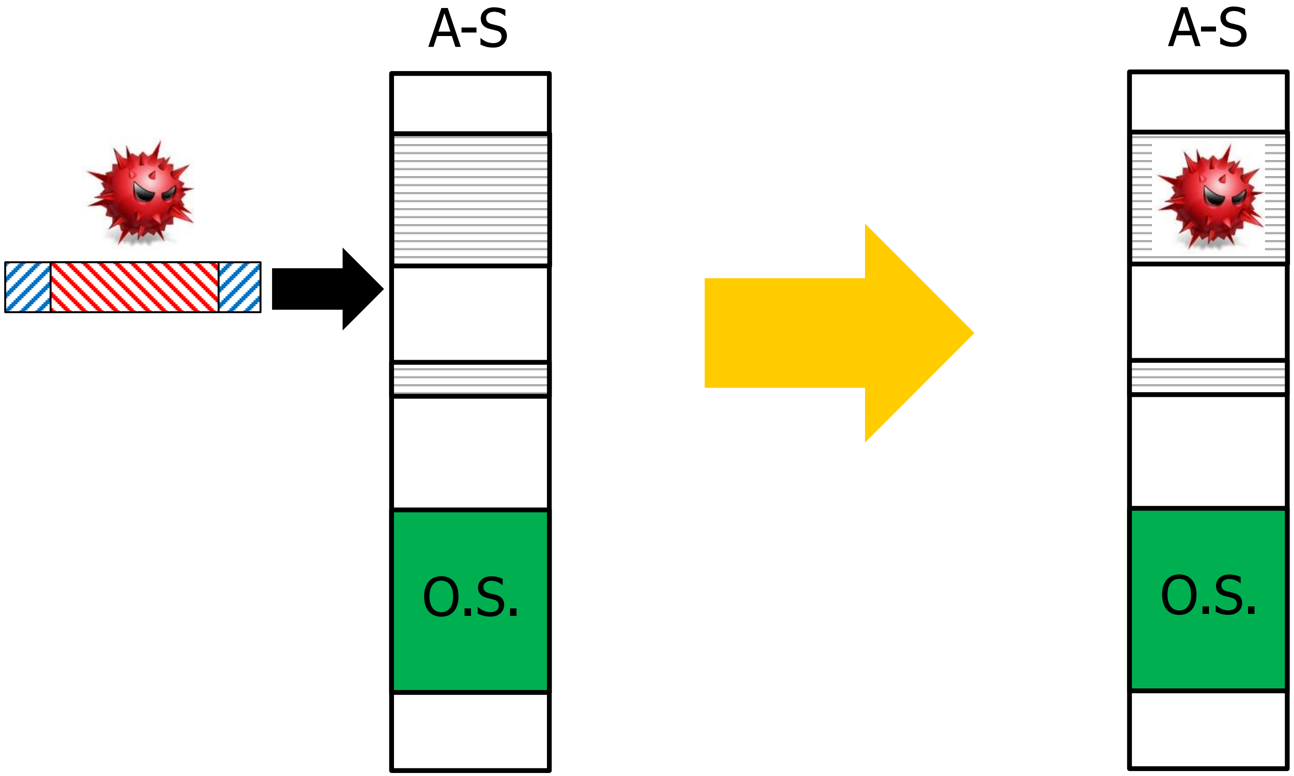
- Human operator H has to perform:
  1. **Daily** "normal" activities
     - Email, web browsing, programming, ...
  2. **Occasionally** "administration" activities
     - Server configuration,
       Account / Access Rights management,
       Program installation/removal, ...

- Which account(s) should H use?

Roadmap

- Common approach
- What should be done and why
- Better approach: Linux
- Better approach: Windows

Remark: RCE vulnerability



Malware has the privilege level of the vulnerable process

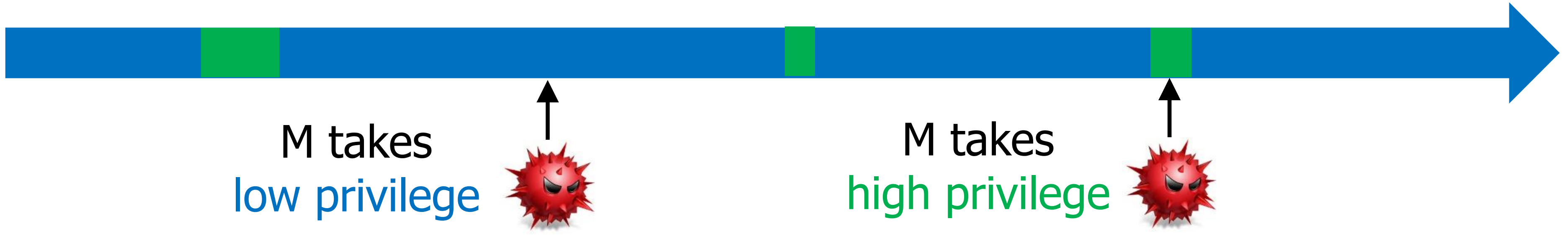
Common Approach

- Human operator H has to perform:
  1. Daily "normal" activities
  2. Occasionally "administration" activities
- H is given **one** account A with **high** privilege

- Is it wise?
- Why?

What should be done (and why)

- H is given **two** accounts: A-H, A-L
  - Use A-L for Daily / "normal"
  - Use A-H **only** for Occasional / "technical administration"



- Most of the time low privilege
- Much less opportunities for taking malware high privilege

Once again...Least privilege!

- **Every** program and every user of the system should operate using the **least** set of privileges **necessary** to complete the job...
- It also reduces the number of potential interactions among privileged programs to **the minimum for correct operation**, so that **unintentional**, **unwanted**, or **improper** uses of privilege are **less likely** to occur...

Much easier said than done

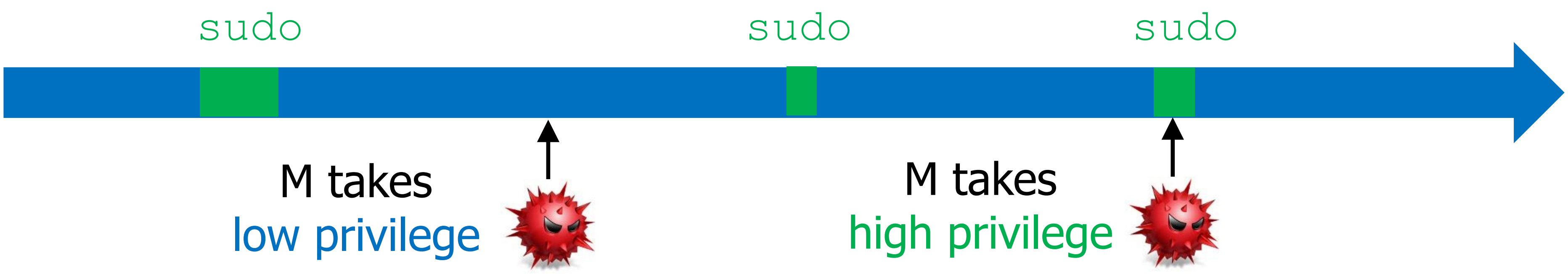
- H is given **two** accounts: A-H, A-L
  - Use A-L for Daily / "normal"
  - Use A-H **only** for Occasional / "technical administration"

- Require **strong** and **systematic** personal **discipline**
- "Why bother?!"

- How many accounts do you have on your Windows PC?
- Do they belong to the `Administrators` group?

Linux approach: `sudo`

- H is given **one** account A-L with **low privilege**
- H always executes shell with A-L...
- ...and may **temporarily** acquire **high privilege:** `sudo cmd`



- **Much more practical** than double account

`sudo`: details ("curiosity")

- To acquire high privilege with `sudo`:
  - A-L must belong to `sudoers` group (membership controlled by the `root` account)
  - A-L password must be provided again

- Normal users: **not** inserted in `sudoers`
- Administrators: inserted in `sudoers`

Windows approach: `UAC` / `run as administrator`

- H is given **one** account A-L with **low privilege**
- H always executes shell with A-L...

- When launching a program C that we want to execute with high privilege:
  - C is launched with '`run as Administrator`' (which asks `Administrator` credentials)

or

  - C must be have been configured to ask for administrator credentials (`UAC`)

`sudo` **vs** `UAC` / `run as administrator`

- Roughly equivalent (**if used properly**)

- In practice, in Windows, usage of a **single** account **with High privilege** is **quite common**
- Default configuration and standard practice encourage this approach

- ...which makes `UAC` / `Run as admin` less effective than `sudo`

Keep in mind

- Human operator H has to perform:
  1. Daily "normal" activities
  2. Occasionally "administration" activities
- H is given **one** account A with **high** privilege

- Very **common** (in Windows)
- Very **dangerous**

O.S. Access Control Essentials

User Groups (Account Groups)

- **Account** belongs to **one** or more **Groups** (one is the **Primary** Group)

- Every resource has:
  - Owner Account
  - ACL with (Account / Group, ...) specified by Owner

ACL in theory

- Every Resource has:
  - **Owner** Account
  - (Account / Group, Operations) specified by Owner

	O1	O2	O3	...
U1	**x**	**x**		
U2	**x**		**x**	
U3	**x**		**x**	
...				

U = Account / Group

ACL in practice

- **MUCH MORE COMPLEX (and O.S.-dependent)**
- Typical (simplified) scenario in next slides

Linux Access Control (in a nutshell)

Linux Example: Files

- **Accounts** described as:
  - Owner
  - Primary Group of the owner
  - Other
- Accounts have **Access Rights** R, W, X

ACL

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

- **Operations** require one or more Access Rights
  - Read → R
  - Write → W
  - Execute → X

Linux Example: Directories

- **Accounts** described as: ACL
  - Owner
  - Primary Group of the owner
  - Other
- Accounts have **Access Rights** R, W, X

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

- **Operations** require one or more Access Rights
  - Listing content → R
  - Modifying content → W,X
  - Listing content and ACLs, Use as current directory, ... → X

ACL in practice

- ~~ACL = (Accounts/Groups, Operations)~~
- ACL = (Accounts/Groups, **Access Rights**)
  - Managed by the Resource Owner

- Mapping **Operation→Access Rights** needed
  - Defined by the O.S. once and for all

Access Rights ≡ Permissions

- More or less synonyms

- Linux tends to use Access Rights
- Windows tends to use Permissions

- But you can find **both** terms in **both** environments

ACL in Linux

- ACL = (Accounts/Groups, **Access Rights**)
- Mapping **Operation→Access Rights** needed

- **Every** resource:
  - **3** Access Rights (R, W, X)
  - **3** entries for describing **all** the accounts
- Mapping Operation → Access Rights "≈intuitive"

Linux ACL: Representation

Access Rights

Accounts

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

"Standard"
representation

`rwx r-x r--`

Remark 1

- **Account** belongs to one or more **Groups** (one is the **Primary** Group)
- **Resource:**
  - **Owned** by an Account
    Can be owned by **multiple** users
    (thus **multiple** primary groups)
  - ACL ≡ 3 x 3 matrix
    More info needed
    (more flexibility)

- Details omitted for simplicity

Remark 2

- `root` processes have **all** access rights on **all** resources

- Implemented with **capabilities**
  - Process with a certain capability $\Rightarrow$
    Process bypasses access control checks for certain operations
  - A `root` process has all capabilities

- A process may be given a **subset** of the capabilities
- Granular control of high privilege

Windows Access Control (in a nutshell)

Windows Access Control

- EXTREMELY COMPLEX

- TERMINOLOGY VERY CONFUSING
  - Sometimes even incoherent

ACL in Windows (I)

- **Every** resource:
  - **3** Access Rights (R, W, X)
  - **3** entries for describing **all** the accounts
- Mapping Operation → Access Rights "≈intuitive"

- **MANY** Access Rights, usually **Resource-specific**
- Mapping Operation → Access Rights "**extremely complex**"

- ACL:
  - **LOTS of entries**
  - **VERY COMPLEX rules for combining them**

ACL in Windows (II)

Windows:

- **MANY** Access Rights, usually **Resource-specific**
- Mapping Operation → Access Rights "**extremely complex**"

- Example in the next two slides

Windows Example: Access Rights (I)

- **Operation** *"Execute file F"*

- Required **access rights** on F:
  - "`GenericExecute`"
  - "`FileReadAttributes`"
  - "`Synchronize`"
- Required **access rights** on D that contains F:
  - "`FileTraverse`"

Windows Example: Access Rights

- Registry:
  - Database of <name, value> (**keys**)
  - Keys are organized as a **hierarchy** based on their name (separator /)
  - Describes the o.s. configuration

- **Operation** *"Create registry key"*

- Required **access rights** on parent key:
  - "`KeyWrite`"
  - "`KeyCreateSubKey`"

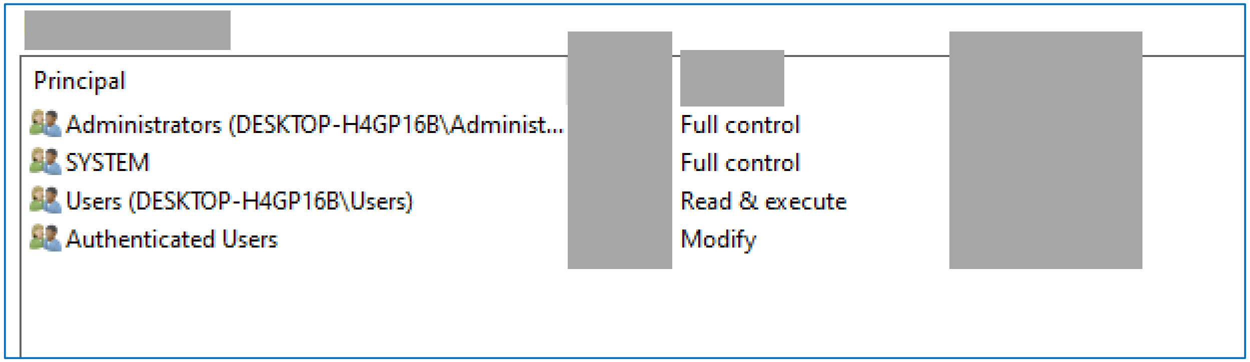
ACL in Windows (III)

- ACL:
  - **LOTS of entries**
  - **VERY COMPLEX rules for combining them**

- Example in the next slides

Windows Example: File (I)

- ACL ≡ List of **Access Control Entries**



- There can be many entries (granularity single account)

⇒ multiple entries for a given principal

⇒ complex rules for choosing the entry

Windows Example: File (II)

- ACL ≡ List of **Access Control Entries**
- Allow or **Deny**
- Can be **inherited** from "parent resource"

Principal	Type		Inherited from
Administrators (DESKTOP-H4GP16B\Administ...	Allow	Full control	C:\
SYSTEM	Allow	Full control	C:\
Users (DESKTOP-H4GP16B\Users)	Allow	Read & execute	C:\
Authenticated Users	Allow	Modify	C:\

- Complex rules for resolving **conflicts**

Nightmare Terminology (I)

"Permission" is usually a synonym of "Access Right"

So is it an Access Right entry?
Shouldn't it be an ACL entry?

Permission entries:

Principal		
Administrators (DESKTOP-H4GP16B\Administ...	Full control	
SYSTEM	Full control	
Users (DESKTOP-H4GP16B\Users)	Read & execute	
Authenticated Users	Modify	

Nightmare Terminology (II)

What it means "access", exactly?

What is the difference w.r.t. "operation"?

Permission entries:

Principal	Access
Administrators (DESKTOP-H4GP16B\Administ...	Full control
SYSTEM	Full control
Users (DESKTOP-H4GP16B\Users)	Read & execute
Authenticated Users	Modify

Remark

```
$ ls -l

drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

- Linux: You see/manage Access Rights

Permission entries:

Principal	Type	Access	Inherited from
Administrators (DESKTOP-H4GP16B\Administ...	Allow	Full control	C:\
SYSTEM	Allow	Full control	C:\
Users (DESKTOP-H4GP16B\Users)	Allow	Read & execute	C:\
Authenticated Users	Allow	Modify	C:\

- Windows: You see/manage "Access" (whatever it means): **not** Access Rights
- Access Rights are hidden behind the user interface

Show ACL from shell

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

- Ask ChatGPT to explain output

BA **You**

can you explain this Windows command execution?

```
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)

Successfully processed 1 files; Failed processing 0 files
```

Smartphone Access Control (in a nutshell)

Keep in mind 1 (REMIND)

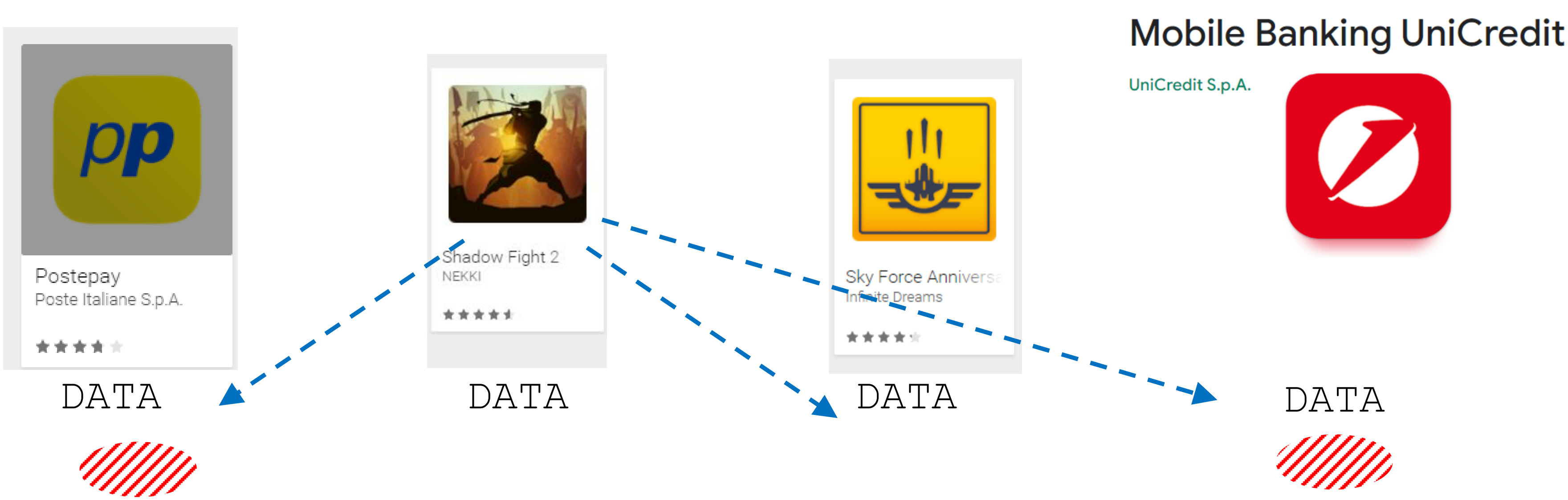
- ACLs have the form (Account, Operation)

- ACLs do not distinguish between **different commands** with the **same account**

- All processes with the same account can do the same things
- Irrespective of who developed their code

Different Point of View

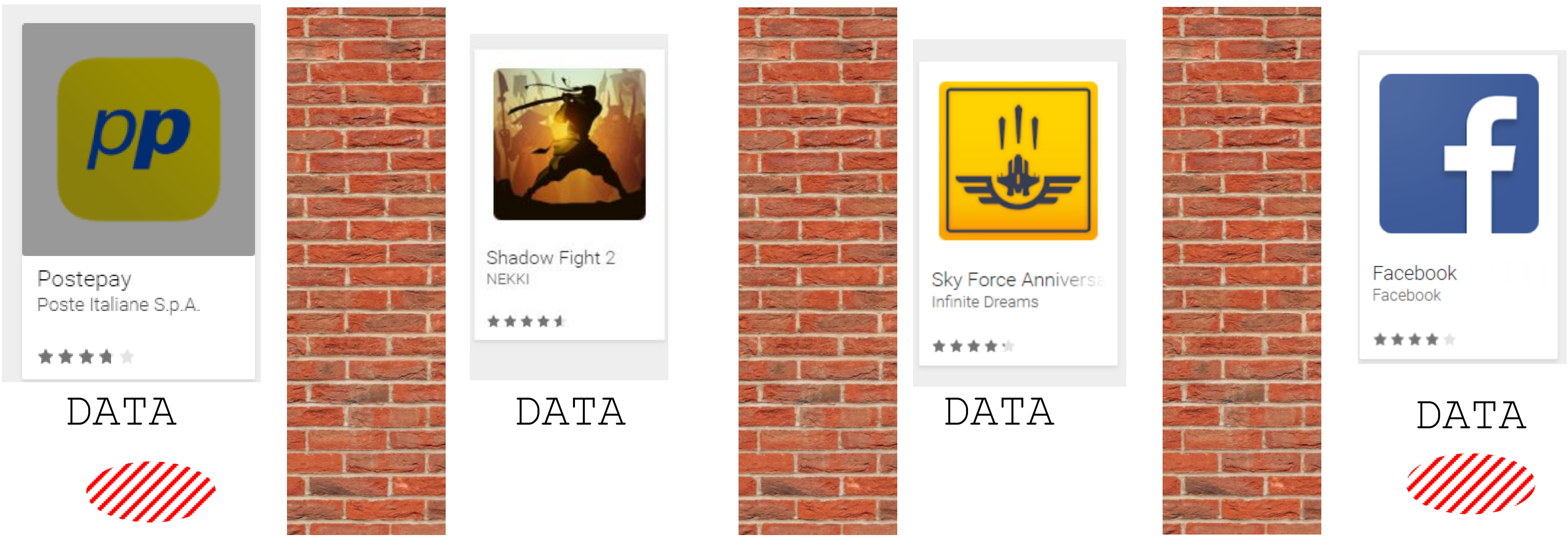
- ACLs have the form (Account, Operation)
- Any app of an user can access **all data of any other app** of that user



Smartphone Access Control (I)

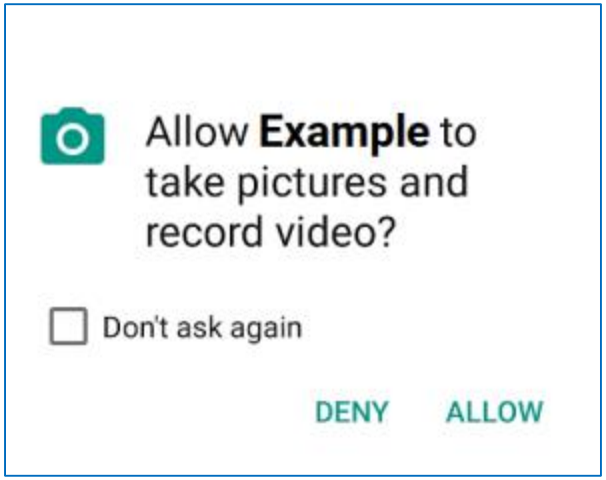
- Each installed app has an identifier
- ACLs are expressed in terms of ([Account, app-identifier], Operation)

- Data of an app can be **isolated** from other apps of **the same** user



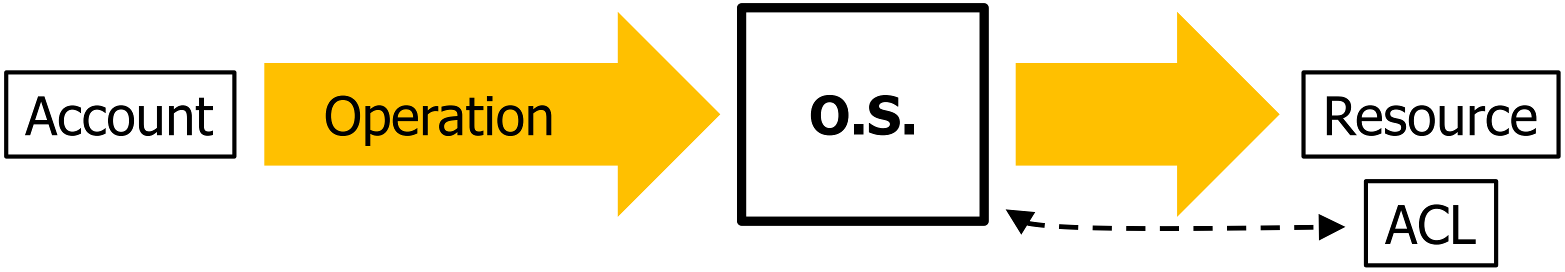
Smartphone Access Control (II)

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



Understanding Access Control

REMIND
Access Control - O.S. Level

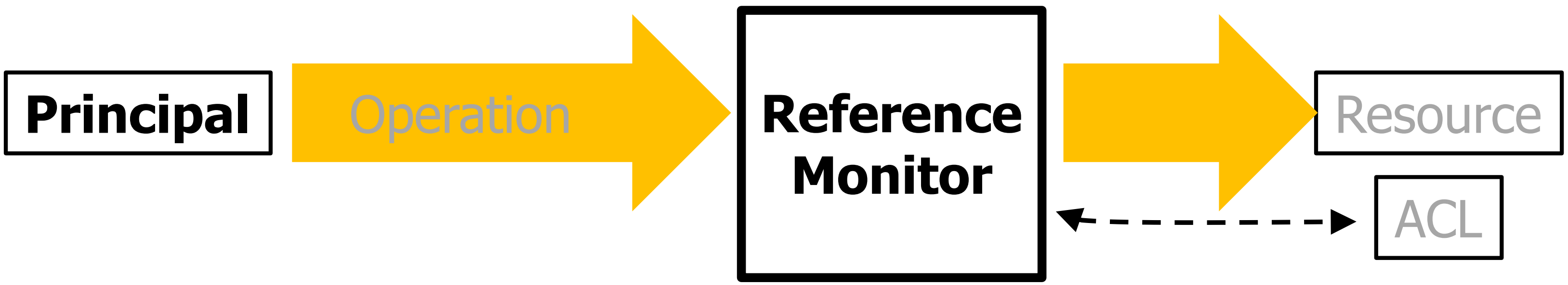


- Every access to **resources** is mediated **(guarded)** by the O.S.
- Every resource has an **ACL**
- O.S. decides whether to execute the operation:
  - Account, Operation, Resource.ACL

Access Control = Authorization (≠ Authentication)

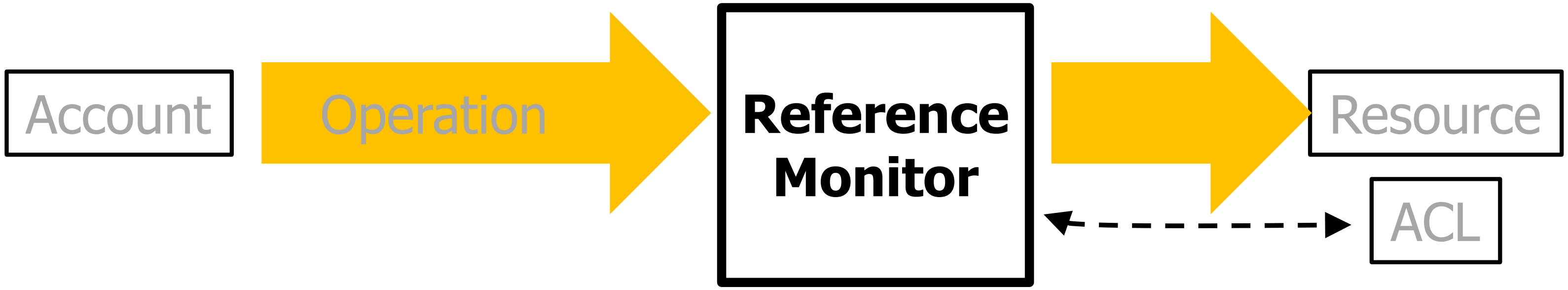
- Account is an **input** data (it is "certain"):
  it is determined **prior** to issuing the OpRequest
- How it is determined is a **different** problem
  - **Authentication** is usually required

Access Control: Terminology



- Every access to **resources** is mediated **(guarded)** by the Reference Monitor
- Every resource has an **ACL**
- Reference Monitor decides whether to execute the operation:
  - Principal, Operation, Resource.ACL

Everything is perfect (I)



- Reference Monitor:
  - No way of **bypassing** it
  - No **mistakes**

Everything is perfect (II)

- Principal:
  - No way of impersonating a **different** Principal

Everything is perfect (III)

- Principals are **not** able to **modify**:
  - Reference Monitor
  - ACLs
    (unless through authorized operations)

Why Cybersecurity is an issue? (I)

- Actual Security Policy **different from the intended one** (ACLs allow operations that should not be allowed)
- Something is **not** perfect:
  - Entity that should not be able to control Principal-A may control Principal-A
  - ...

- See "Midnight Blizzard attack to Microsoft" on the companion website:
  - Test application $\rightarrow$ Senior leadership Cybersec people email and docs
- Incident in Trieste (27K ransom paid)
  - Secretary receives pdf invoice with malware from (unsuspecting) commercial partner
  - Malware encrypts all files in all folders of the company filesystem

Why Cybersecurity is an issue? (II)

- Actual Security Policy **different from the intended one** (ACLs allow operations that should not be allowed)
- Something is **not** perfect:
  - Entity that should not be able to control Principal-A may control Principal-A
  - Reference Monitor has **mistakes**
  - Reference Monitor may be **bypassed**
  - Principal-A may emit (OpReq, **Principal-B**)

- Do NOT consider these cases! (for the time being...)

Access Control: FUNDAMENTAL Mechanism

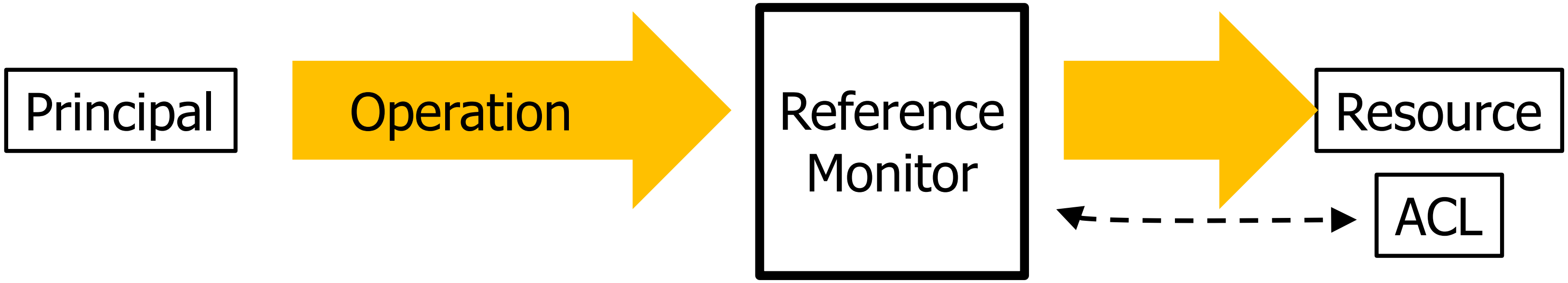
Application Resource ≠ O.S. Resource

- **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?

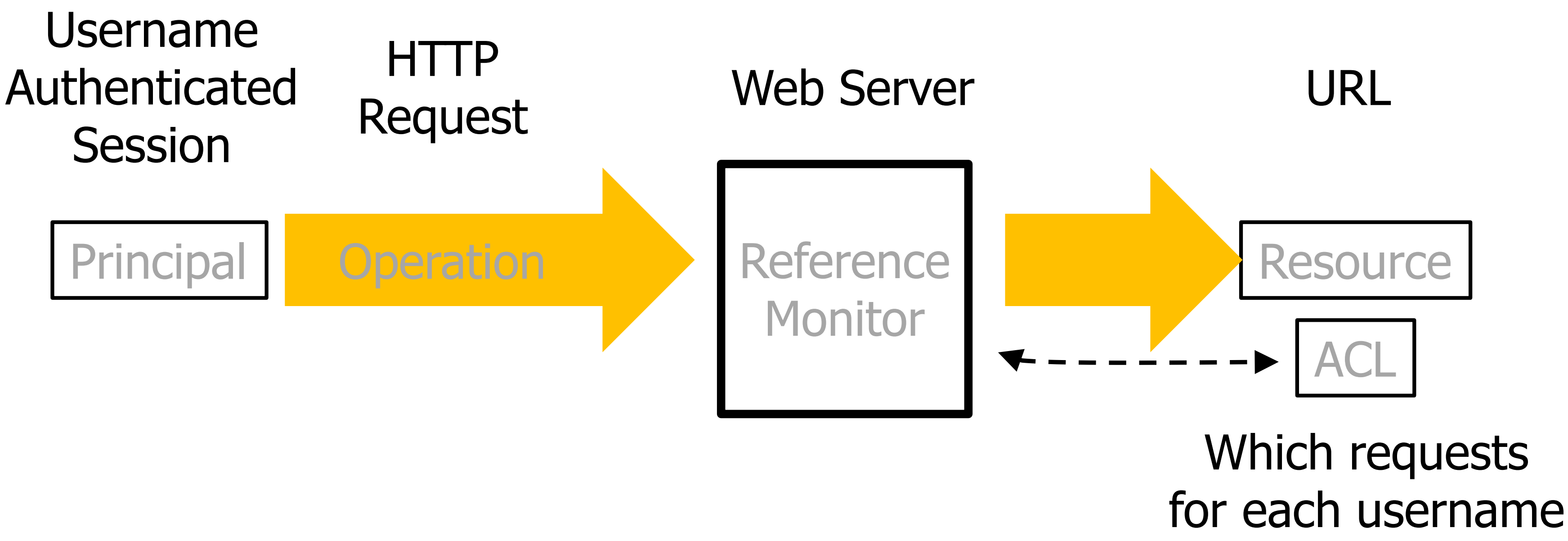
What we need



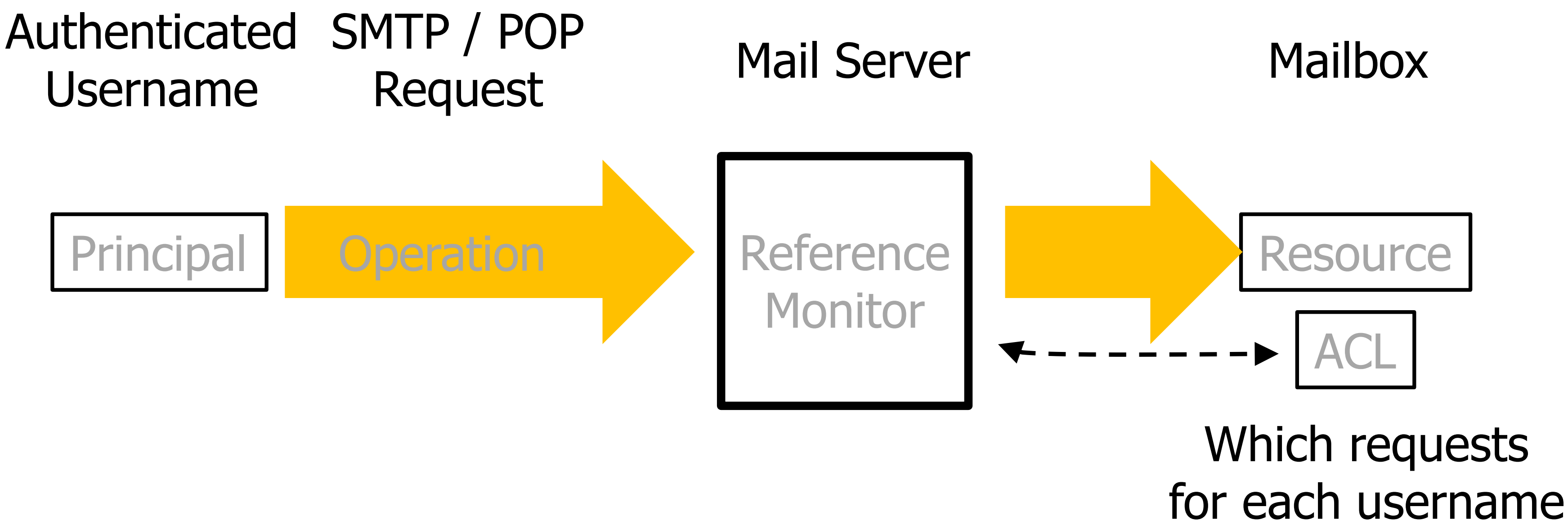
- Resource access must be mediated:
  - Operating system level
  - Application level

- Mechanisms **independent of each other**

Access Control – Web Server



Access Control – Mail Server

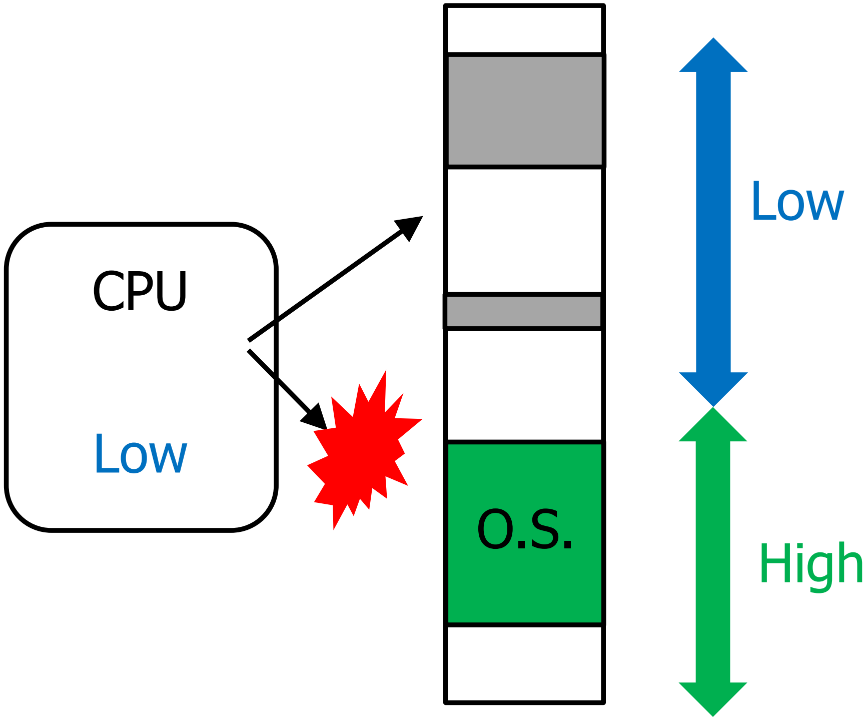


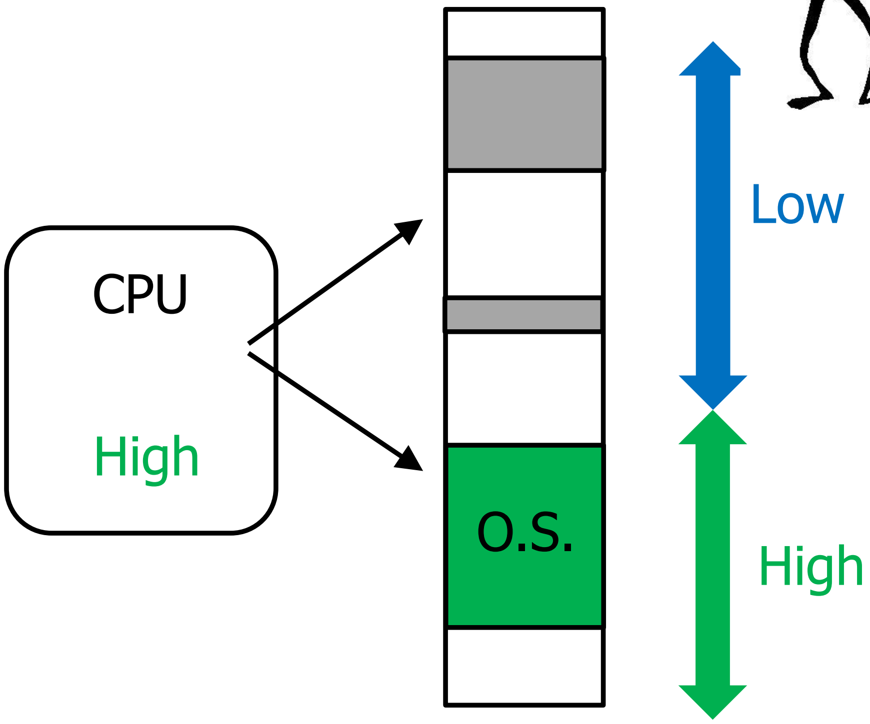
Access Control: Abstract (=GENERAL) Model

Username Authenticated Session	HTTP Request	Web Server	URL
Authenticated Username	SMTP / POP Request	Mail Server	Mailbox
Account	System call	O.S.	O.S. Resource

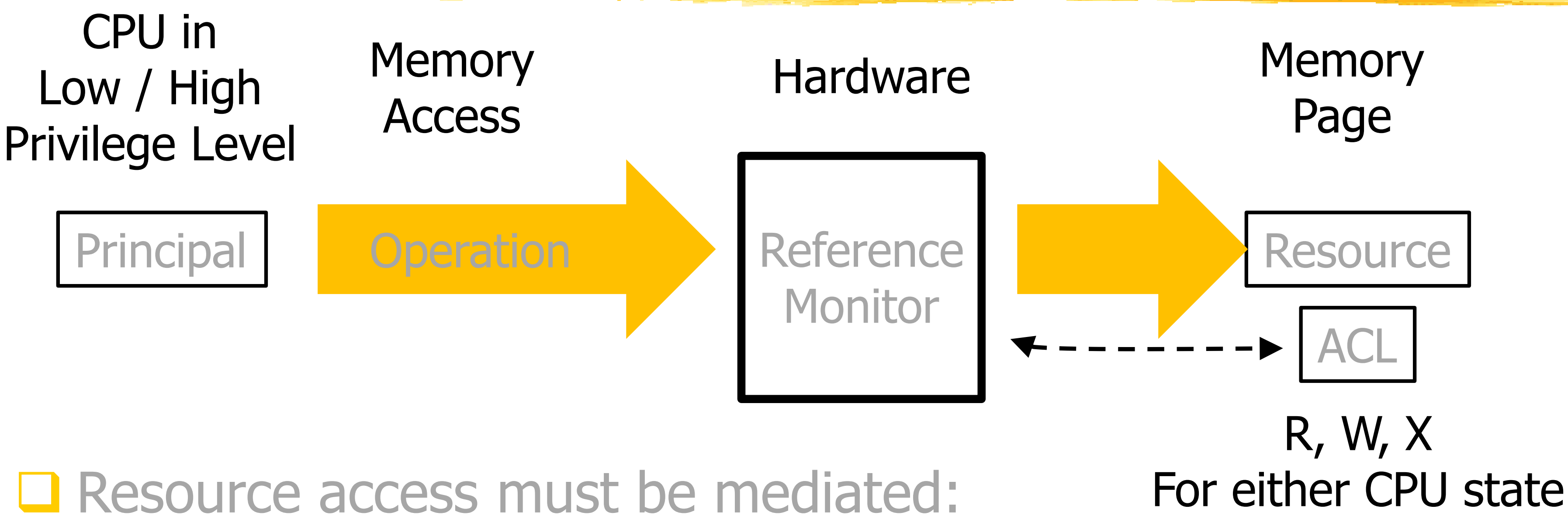
Hhmmm...

- How can **the memory** know which privilege level can access it?
- How is this security policy actually **enforced**?





A truly GENERAL model



- Resource access must be mediated:
  - Operating system level
  - Application level
  - Hardware level
- Mechanisms **independent of each other**

Access Control

- **Fundamental** feature of computer systems
- **Enforces** the **security policy**: "who can do what"

- Occurs at **multiple** and **different** levels:
  - Application
  - Operating system
  - Hardware

- Each level:
  - Is **independent** of the other levels
  - Has **its own** mechanisms

Saltzer and Schroeder (1974)

- **Complete mediation: Every access to every object must be checked for authority.**

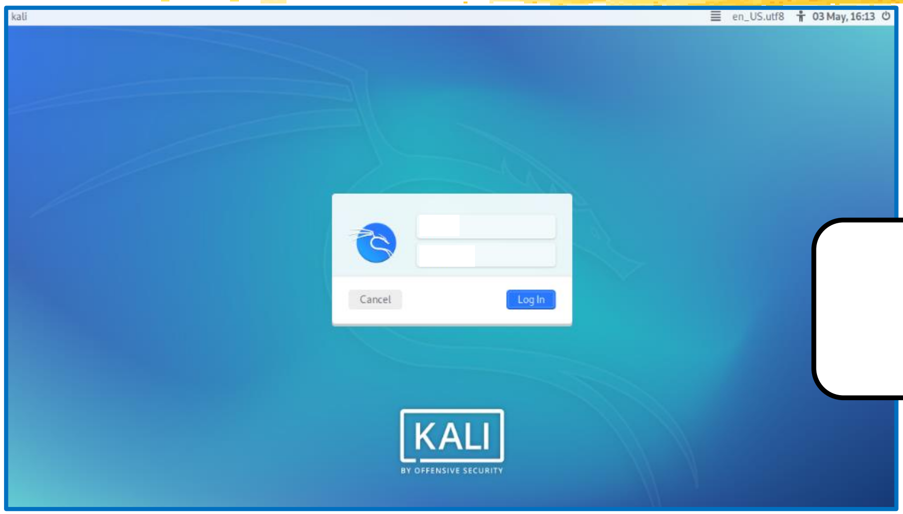
- This principle, when systematically applied, **is the primary underpinning** of the protection system...
- It implies that **a foolproof method of identifying the source of every request** must be devised.

- Please take a moment to reflect and admire its depth and generality
- We will find more examples of its relevance

Access Control in Large Organizations

Authentication

Where are Accounts defined?



logon process

`root / SYSTEM`

1. Wait for credentials
2. Validate credentials (authenticate account A2)
3. Spawn GUI process that changes account to A2

Authentication DB: Local

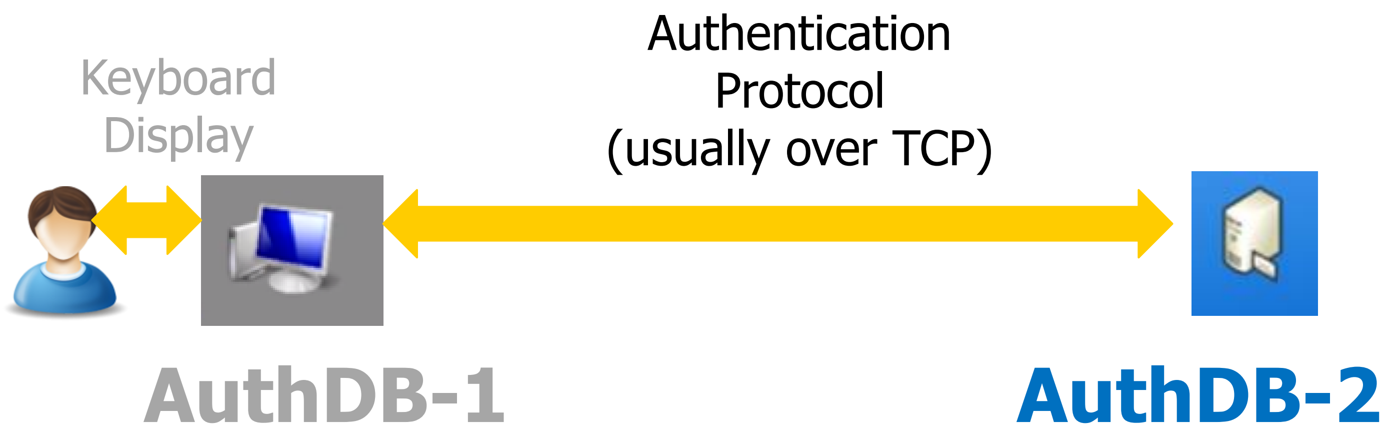
Account	Password
…	
Paolo	pwd-Paolo
…	

One row for each Account

AuthDB

- Impersonating an account requires proving knowledge of a certain **secret** (password)
- AuthDB usually managed by the **operating system** (a certain file, at a certain location)

Authentication DB: Network (I)



- Web site
- Mail server
- File server
- Printer
- API server
- ...

- Either the same or different organizations
- Sets of accounts and passwords completely independent of each other

Authentication DB: Network (II)

Authentication
Protocol
(usually over TCP)

AuthDB

- Web site
- Mail server
- File server
- Printer
- API server
- …

- Depending on the server, AuthDB may be either:
    1. AuthDB of the local operating system
    2. **Another** AuthDB managed by the server (usually stored in a **database table**)

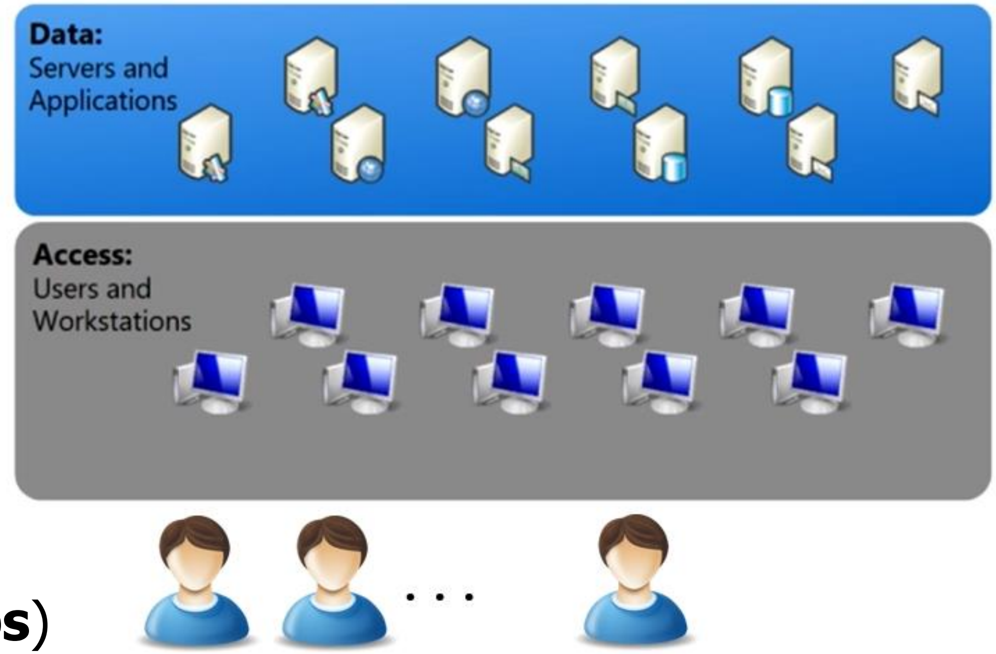
Large Organizations

Large Organizations (I-a)

Tens/Hundreds of **Servers**
(storing **Files**, **Databases**)

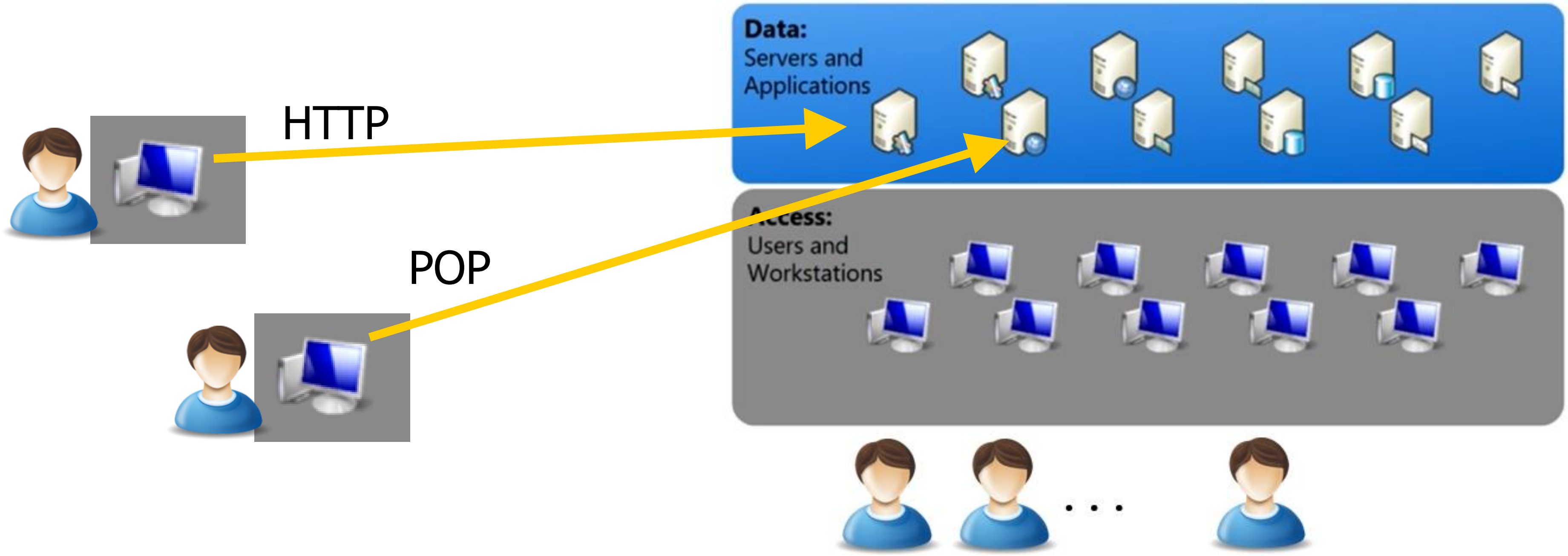
Thousands of
Workstations / Notebooks
(either private or shared)

Thousands of **Accounts**
(**tens** of partially overlapping **Groups**)



Large Organizations (I-b)

Some Servers may be accessed
from the **outside**



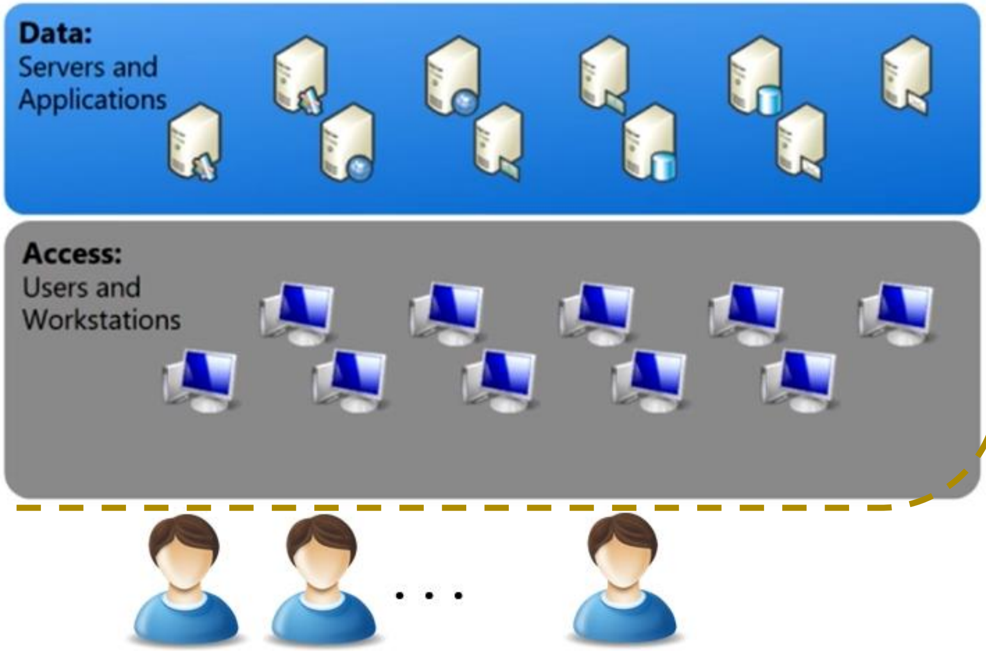
Large Organizations (II)

Resources

Routers, Firewalls, Switches, Networks,...

Servers
(storing **Files**, **Databases**)

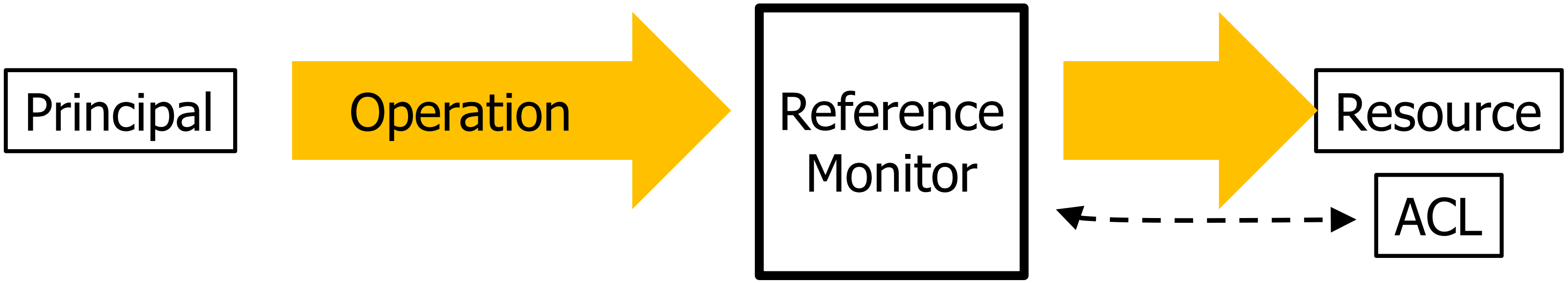
Workstations / Notebooks
(either private or shared)



Accounts
(partially overlapping **Groups**)

Identities

Access Control



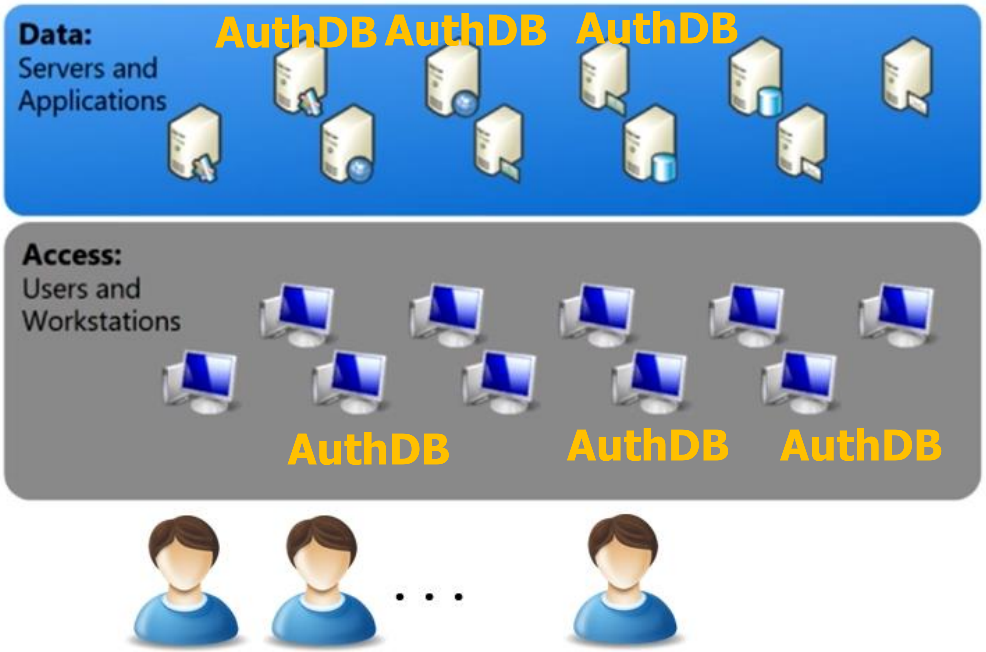
- **Every resource access must follow this framework**
  - Application level (e.g., access to a remote server)
  - O.S. level (e.g., shell / GUI)

- Pre-requisite: Authentication

Authentication: Key practical requirement

We do **not** want a **separate** AuthDB on **each** Reference Monitor

(identity management would be a nightmare)



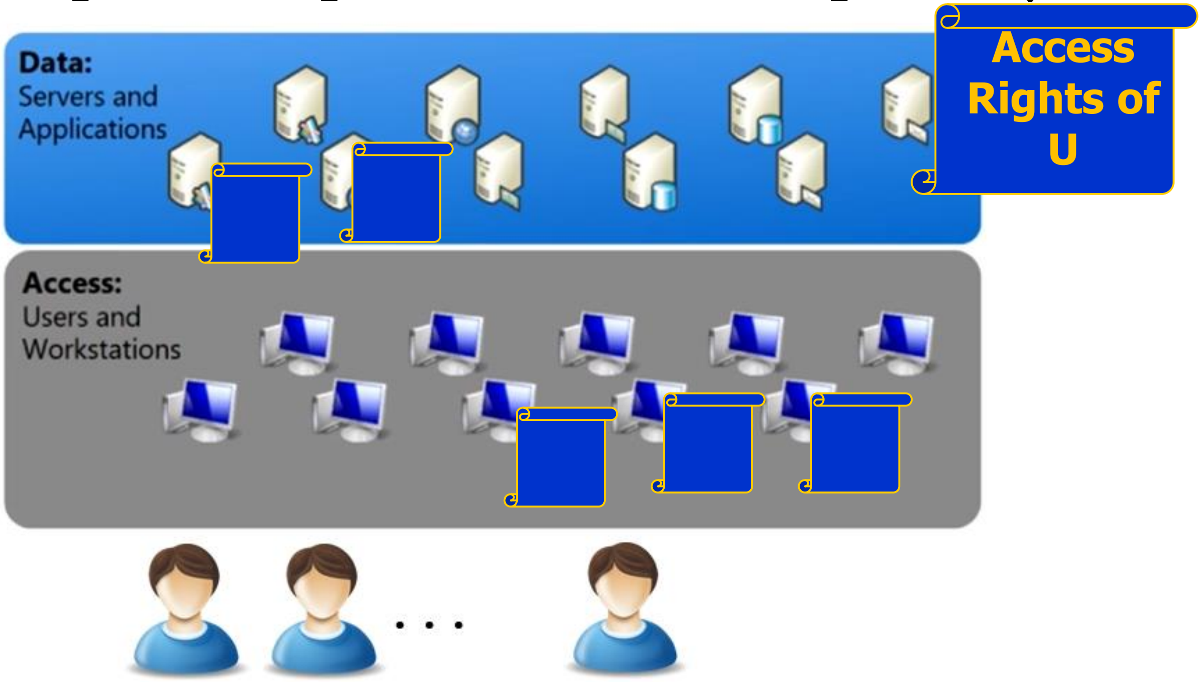
Key Practical Problems

- Can account U modify file F?
- Can account U read database D?
- Can account U logon on computer C?
- ...
- Can account U at computer C access server S?
- ...
- Can computer C connect to network N?
- Can computer C access server S?
- ...

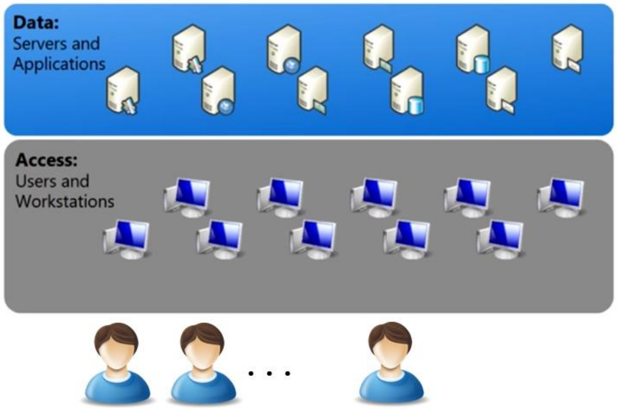
Authorization: Key practical requirement

We do **not** want to specify ACLs **separately** on **each** resource

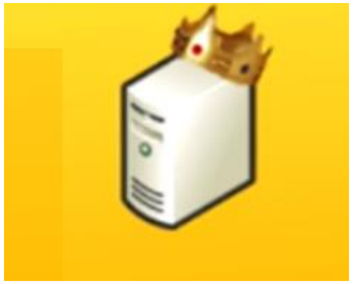
(access rights management would be a nightmare)



Directory Service



DIRECTORY SERVICE



- **Centralized** repository (**Directory Service**) describes:
  - All **identities** (including their credentials)
  - All **resources**
  - All **access rights** of identities to resources (ACLs)

Example: myself@UniTS

- Every account is described in our Directory Service
- My description consists of **>60 attributes**

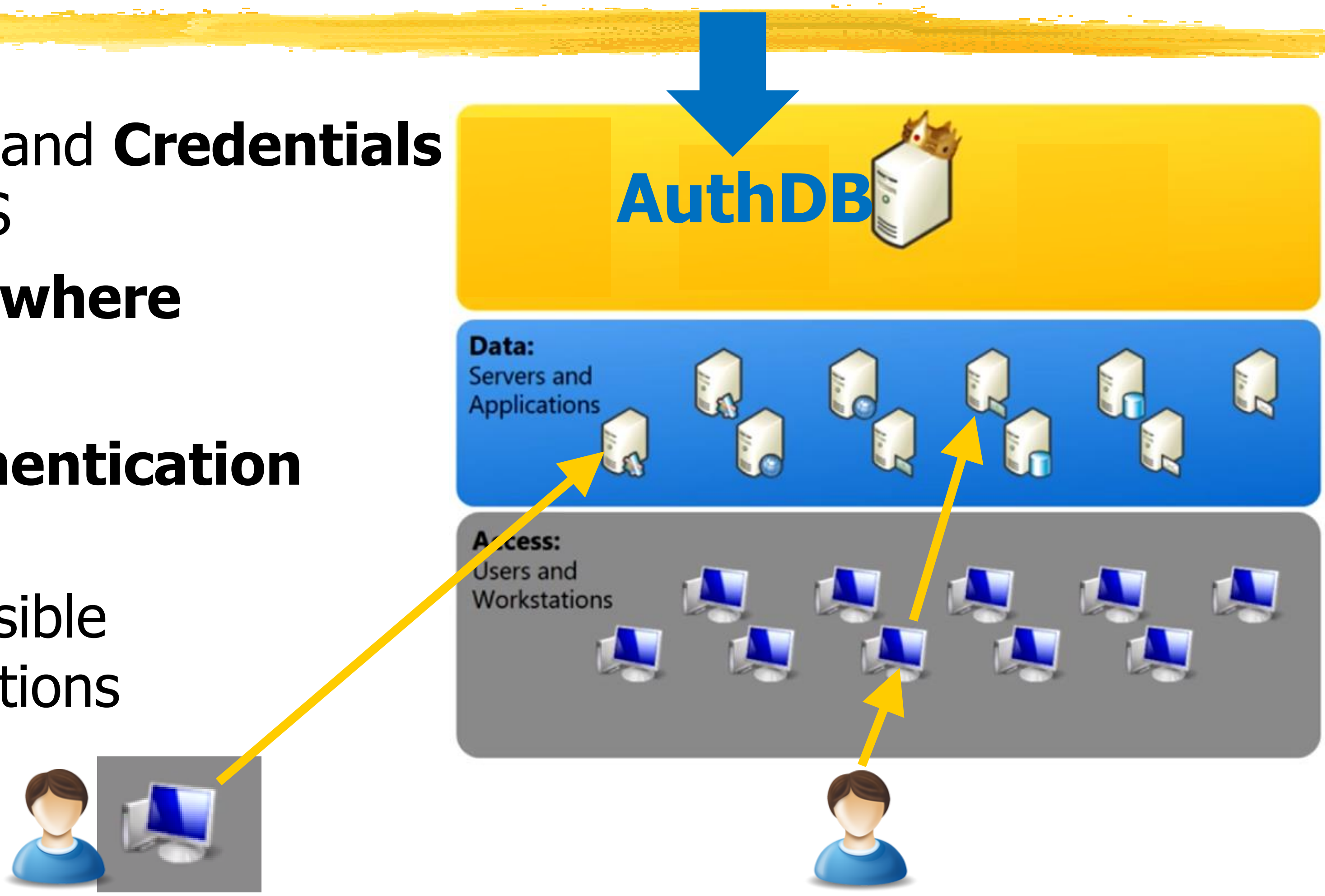
accountExpires	Integer8	1 0x0
cn	DirectoryString	1 BARTOLI ALBERTO [5943]
lastLogonTimestamp	Integer8	1 2/10/2023 13:22
mail	DirectoryString	1 bartoli.alberto@units.it
mAPIRecipient	Boolean	1 FALSE
name	DirectoryString	1 BARTOLI ALBERTO [5943]

…

Single Sign On (SSO)

- **Identities** and **Credentials** stored in DS
- Valid **everywhere**

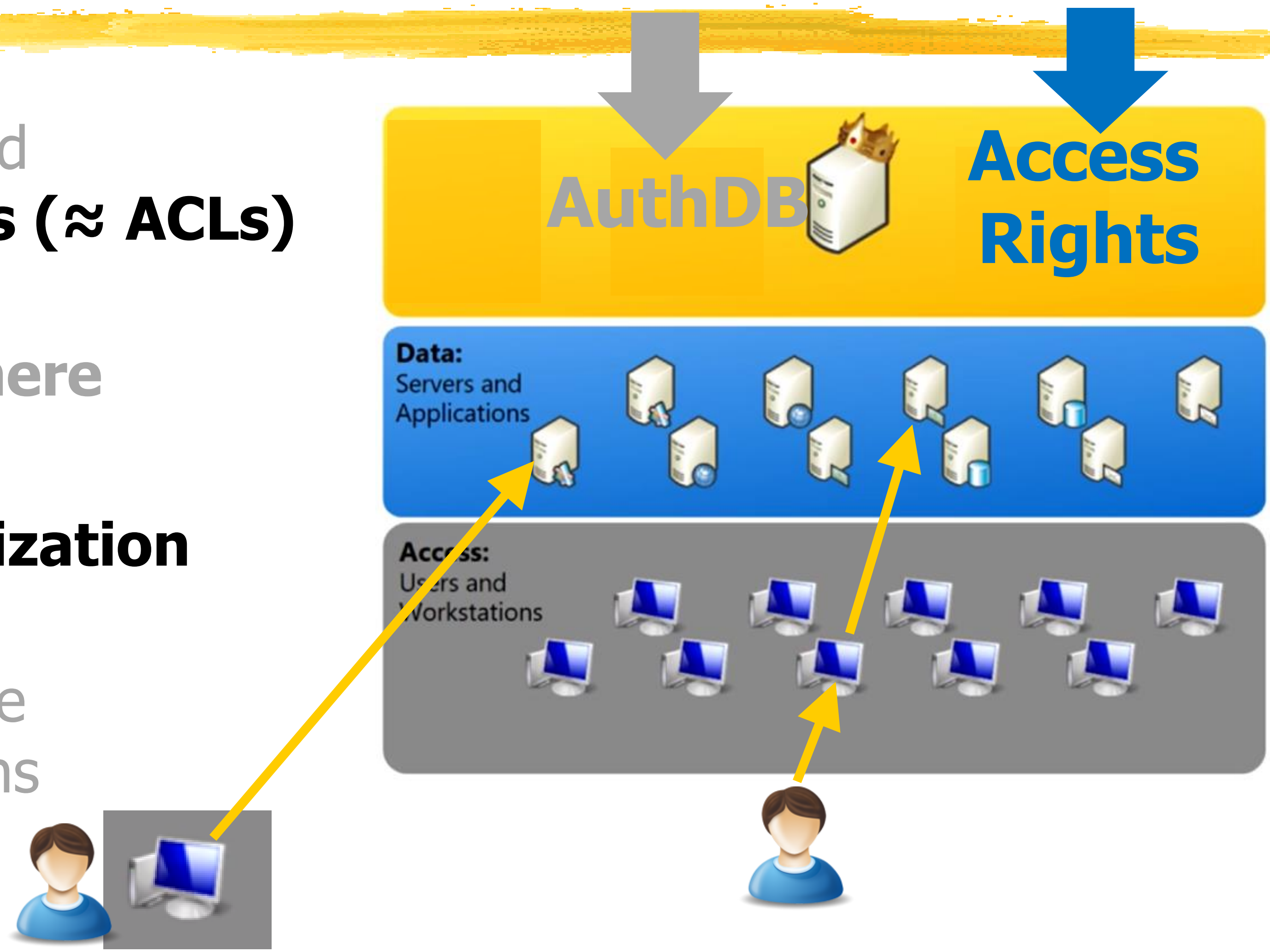
- **Every authentication** involves DS
- Several possible implementations



SSO + Centralized Authorization

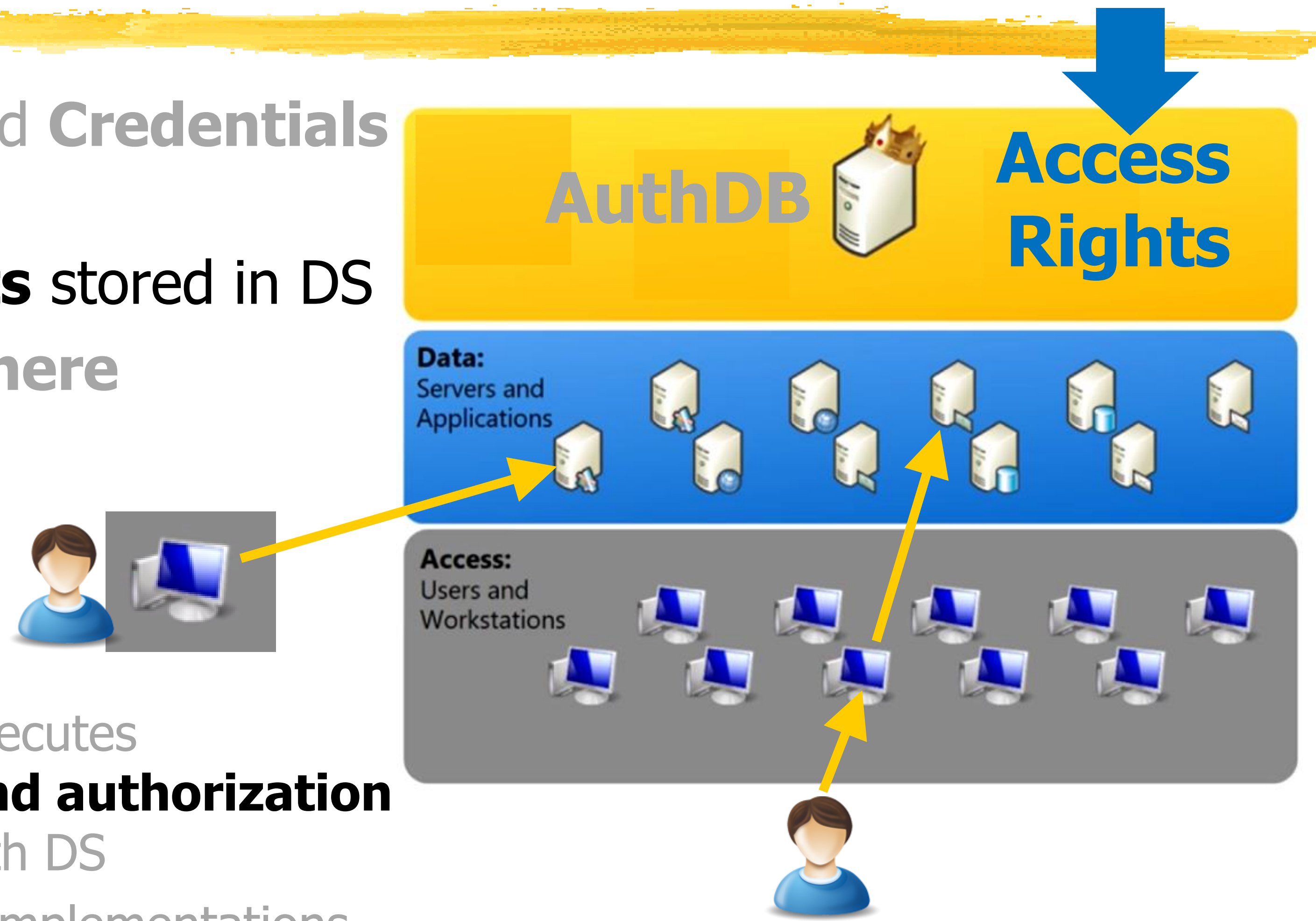
- **Resources** and **Access Rights (≈ ACLs)** stored in DS
- Valid **everywhere**

- **Every authorization** involves DS
- Several possible implementations



SSO + Centralized Authorization

- **Identities** and **Credentials** stored in DS
- **Access Rights** stored in DS
- Valid **everywhere**



- Each resource executes authentication **and authorization** by interacting with DS
- Several possible implementations

Identity and Access Management (IAM)

- **Procedures** and **technologies** for management of individual **identities**, their **authentication**, **authorization**, and **access rights**
- **within** or **across** **enterprise** boundaries

Our focus

- Our focus is **within** enterprise boundaries
  - Account and resource in the same organization
- Widely prevalent technology:
  - **Windows Active Directory**
  - **Domain** ≈ All IT entities in an organization
  - **Domain Controller** ≈ Directory Service

- Technologies **across** enterprise boundaries
  - OAuth, SAML (SPID)
  - Kerberos realms

Our learning path

- Every authentication and every authorization involves DS
- Several possible implementations

1. LDAP SSO (outline)
2. ...
3. Passwords and MFA
4. NTLM
5. Kerberos

Real Usage (in Windows Active Directory)

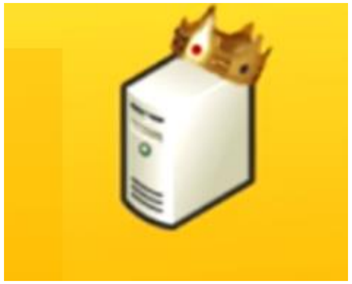
- **Kerberos**
  - **Default** for Windows software
- **NTLM**
  - Supported for **compatibility** in Windows software
  - **"It should be disabled for security reasons" (Microsoft 2010)**
  - It is still with us
- **LDAP SSO**
  - Used **only** by software hard to integrate in Windows AD
  - Example: Web applications on Linux (e.g., `esse3`)
  - Example: Enterprise Wi-Fi authentication server (e.g. `eduroam`)

LDAP SSO

LDAP: Double Meaning

- **L**ightweight **D**irectory **Access Protocol**

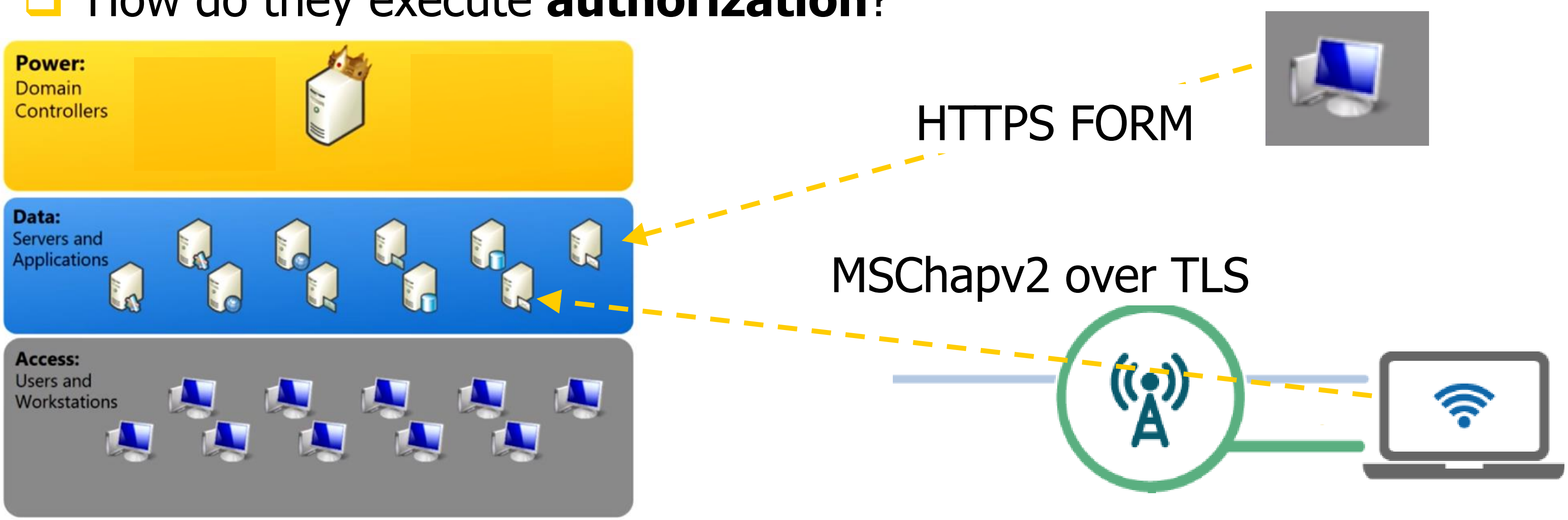
DIRECTORY SERVICE



- A **standard** for **describing** IT entities:
  - Identities
  - Credentials
  - Resources
  - Access Rights
- A **protocol** for interacting with a Directory Service (server that stores those descriptions)

Practical Problem

- Not every software can act as a client for Windows Active Directory
  - Example: Web applications on Linux (e.g., `esse3`)
  - Example: Enterprise Wi-Fi authentication server (e.g. `eduroam`)
- How do they execute **authentication**?
- How do they execute **authorization**?



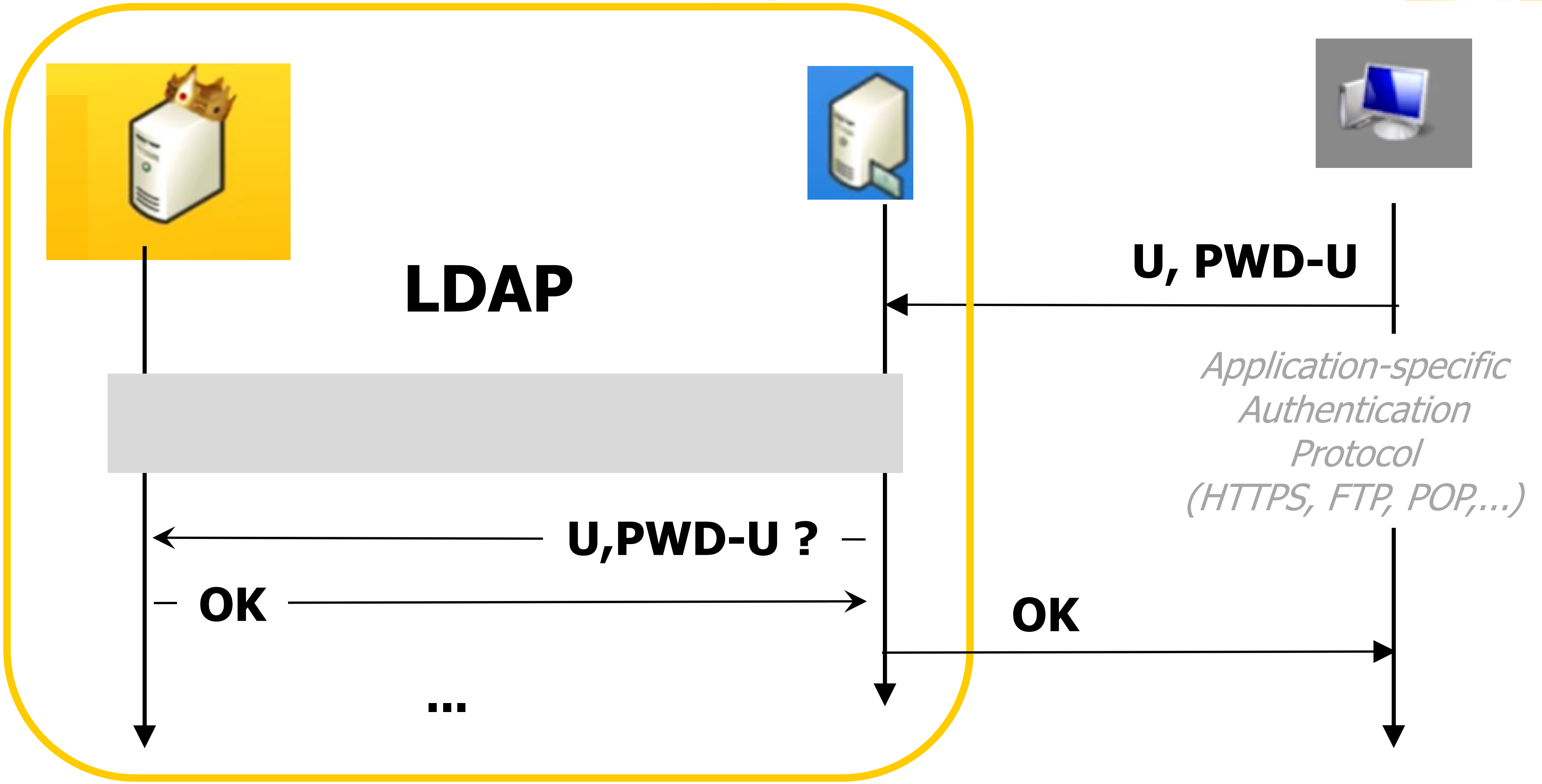
Common Solution (outline)

Server **asks Directory Service**

1. Credentials valid?
2. What are its Access rights?

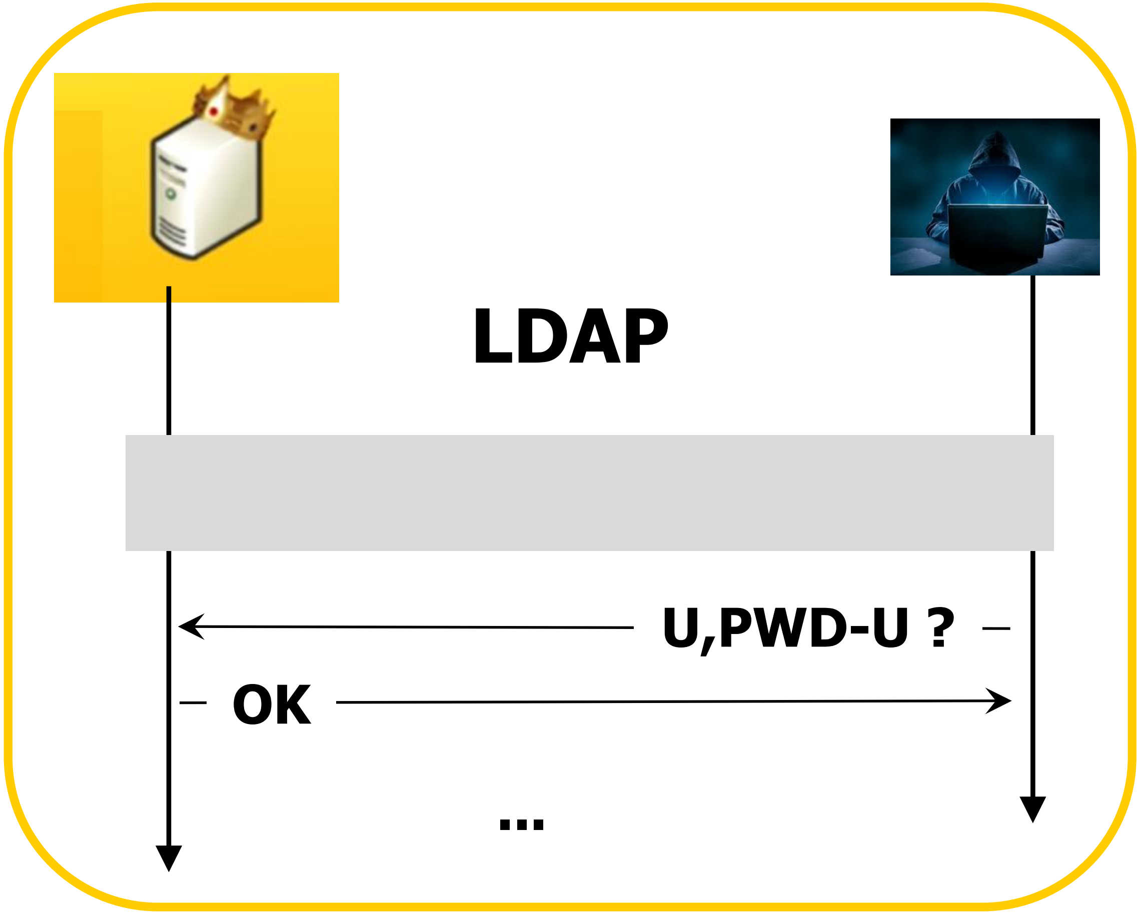
- Dedicated application protocol: **LDAP**

LDAP SSO (I)

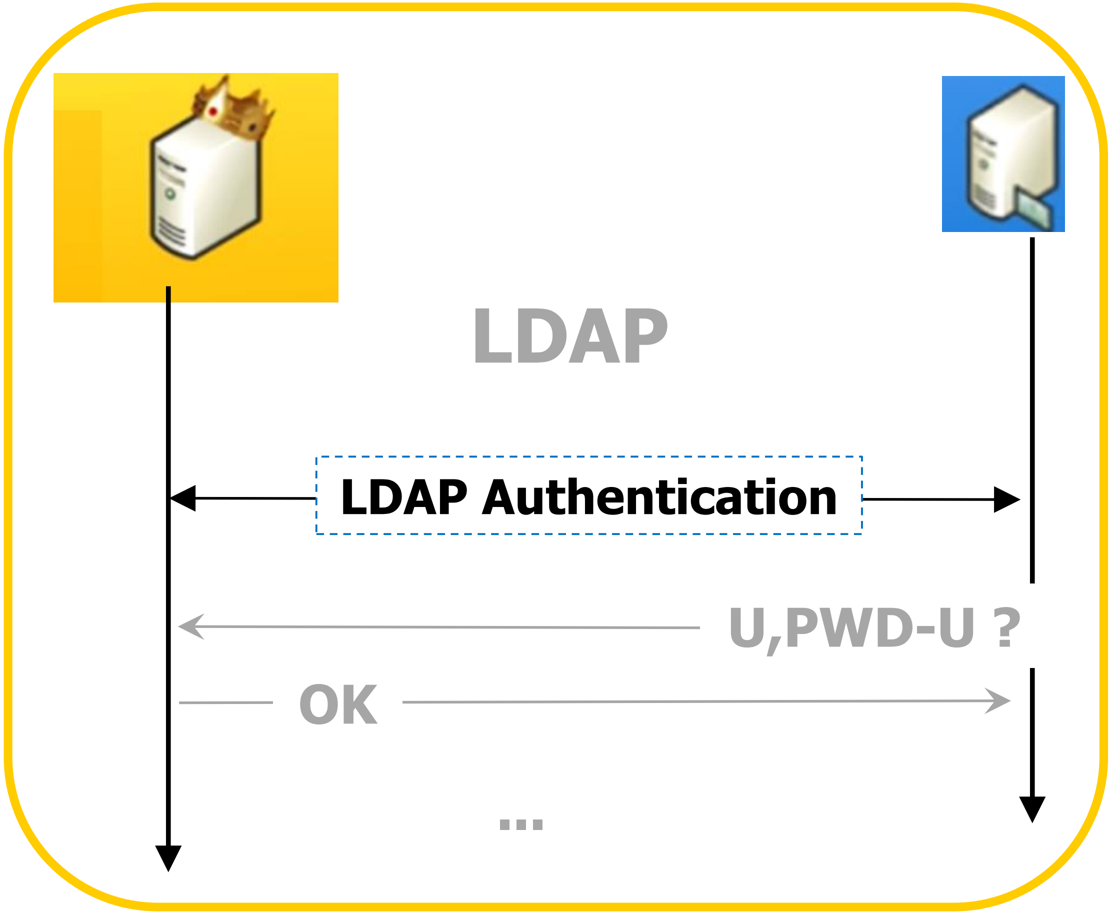


LDAP
U, PWD-U
Application-specific
Authentication
Protocol
(HTTPS, FTP, POP,...)
U,PWD-U ?
OK
OK
...

Hhmmm…



LDAP SSO (II)



- Certain LDAP operations require **LDAP authentication** of the client **application**
- Several options in LDAP
  - TLS+username/password (**LDAPS**)
  - ...