GENERALLY ACCESSIBLE

Building Secure Systems

Plattform Security – Isolation, Privileges and Server Security

Walter Kriha, Computer
Science and Media Faculty

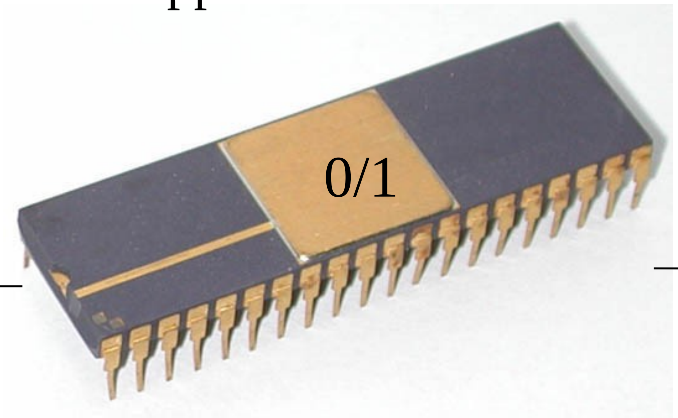
OS-Architecture and Security Mechanisms

CPU Protection Levels

State of protected mode bit:

1 = protected/kernel mode

0 = application/user mode



Sensing operations (I/O)

Control operations (halt, memory mgmt.)

Regular compute operations (add, mul)

Most CPUs offer a simple protetion scheme. Dangerous operations (sensing, control) are only allowed when the CPU has been put in kernel mode (protection bit is set). Applications can NOT change the state of the CPU arbitrarily. They MUST use certain controlled gates (software interrupts) to change the mode. From then on, operating system code runs!

Plattform-level Methods for Secure Systems

Different privilege levels

Isolation of components (virtual memory, micro kernels)

Interception with Wrappers

Virtual Machines

Inversion of Control, Dependency Injection Architectures

Capabilities

Name Spaces, Services and Components

Trusted Computing Bases

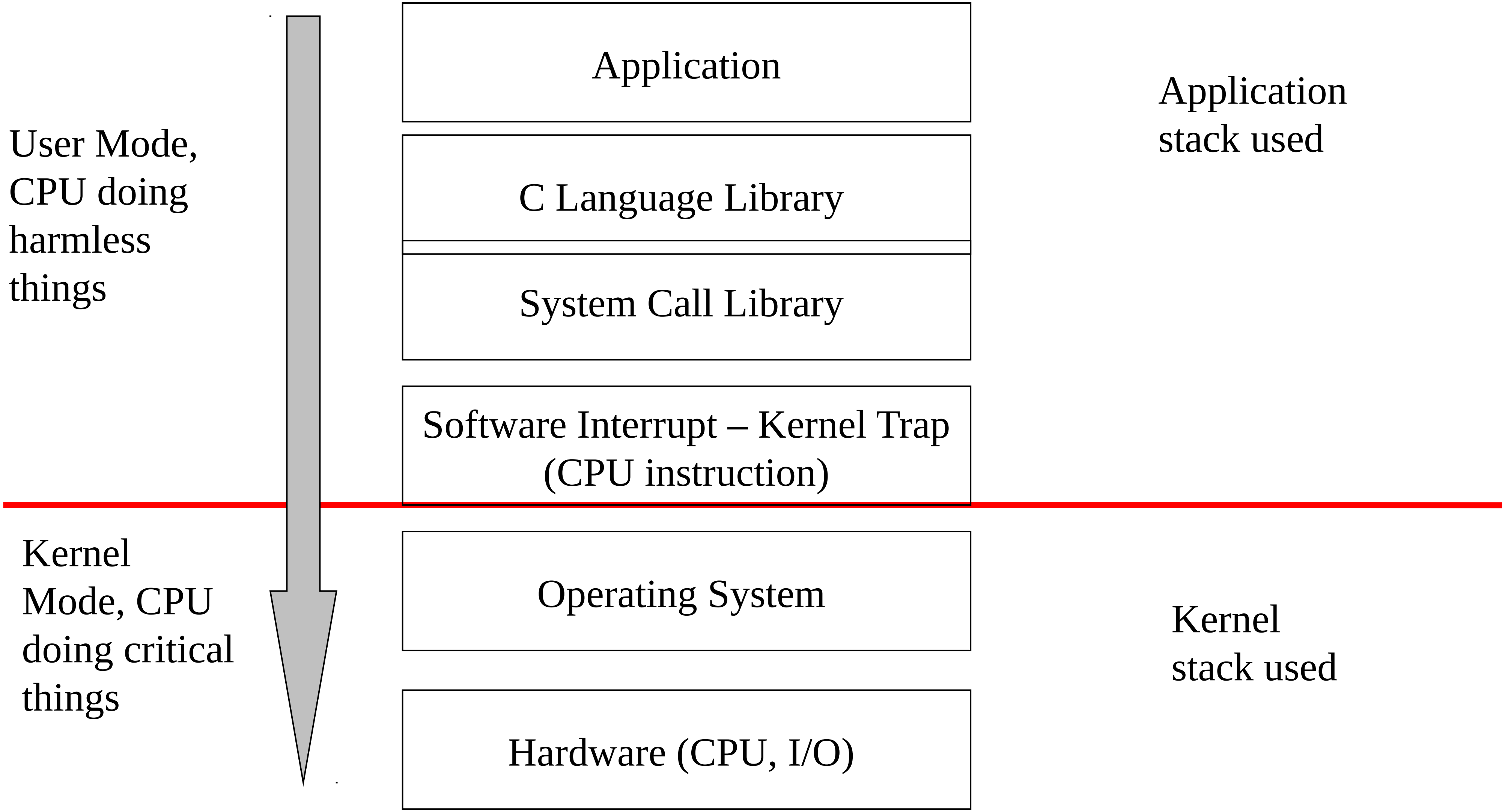
Sandboxes

Jails

Funnels

Code Verification

Switching to Kernel Mode



User Mode, CPU doing harmless things

Application

C Language Library

System Call Library

Software Interrupt – Kernel Trap (CPU instruction)

Application stack used

Kernel Mode, CPU doing critical things

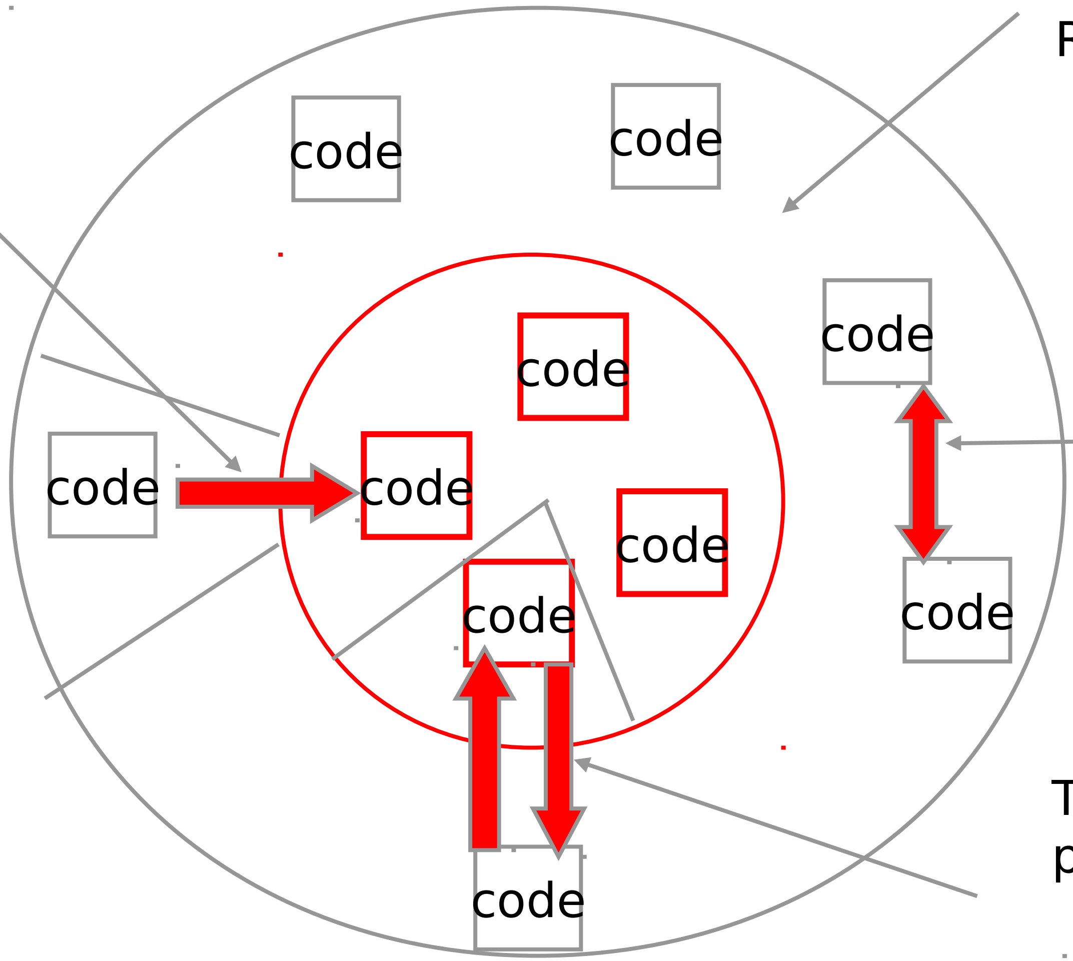
Operating System

Hardware (CPU, I/O)

Kernel stack used

Only in kernel mode will the CPU allow critical instructions. The application will be terminated if it tries to execute critical instructions without changing through kernel traps into protected mode.

Security Properties of Privilege Modes



Trick code into extending callers privileges

Rights/Authority

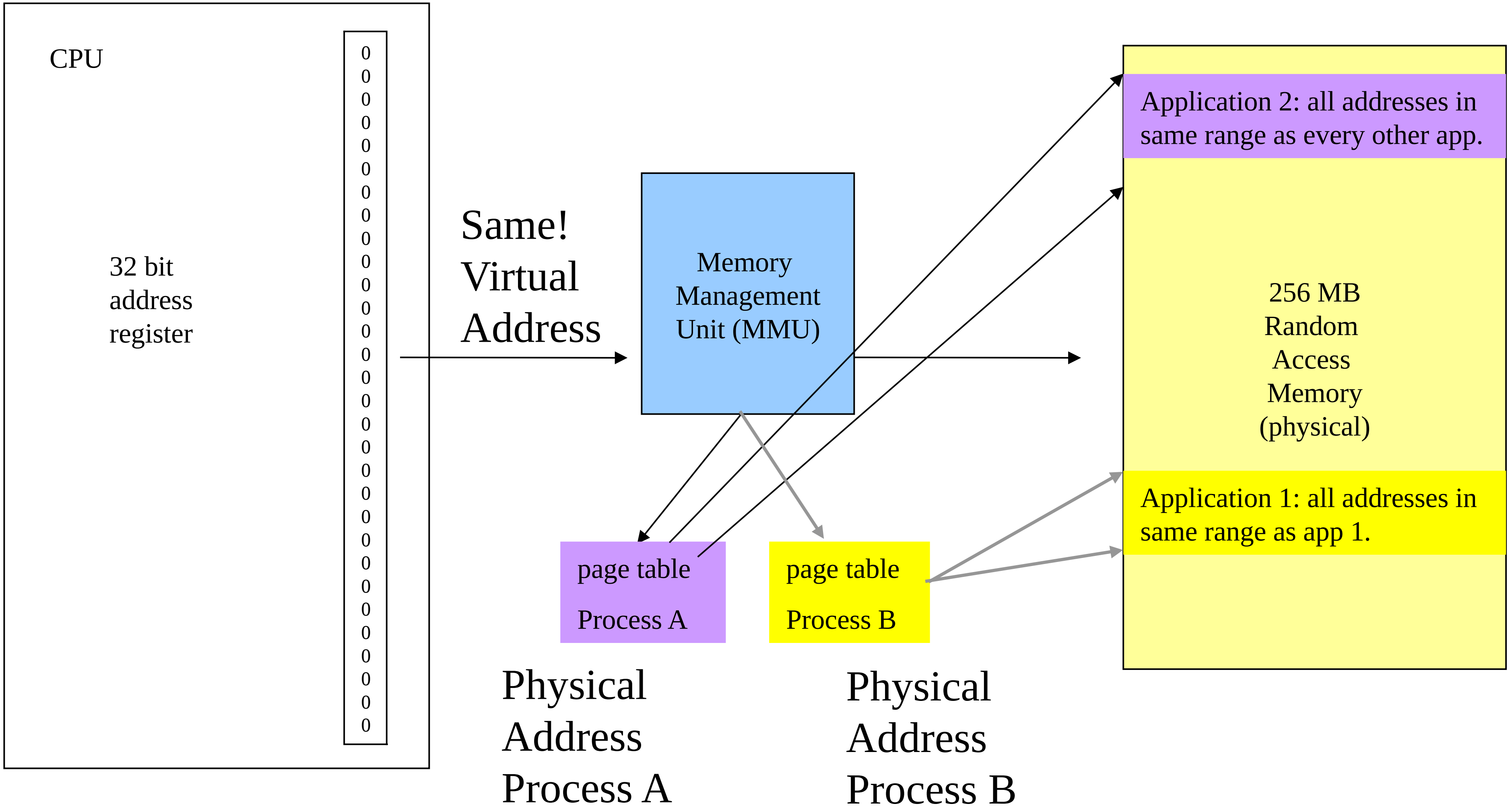
Attack code on same level

Turn off privileged mode

Security Properties of Privilege Modes

1. Rights organized as privilege levels typically lead to ambient authority for code pieces because no sectors are possible within the levels. Most code pieces would need a different segmentation of rights.
2. On the same level each piece of code is a threat to all others (same privileges, no sep.)
3. Every mode change needs to be undone later. Software needs to make sure that on the way back the privilege mode gets reversed to the old value. There cannot be a bypass of this piece of code
4. The mode change needs to be authorized
5. The privileged code needs to interpret arguments carefully to avoid extending the callers privileges in an unwanted way (confused deputy)

Address Isolation



CPU

32 bit address register

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Same! Virtual Address

Memory Management Unit (MMU)

page table Process A

page table Process B

Physical Address Process A

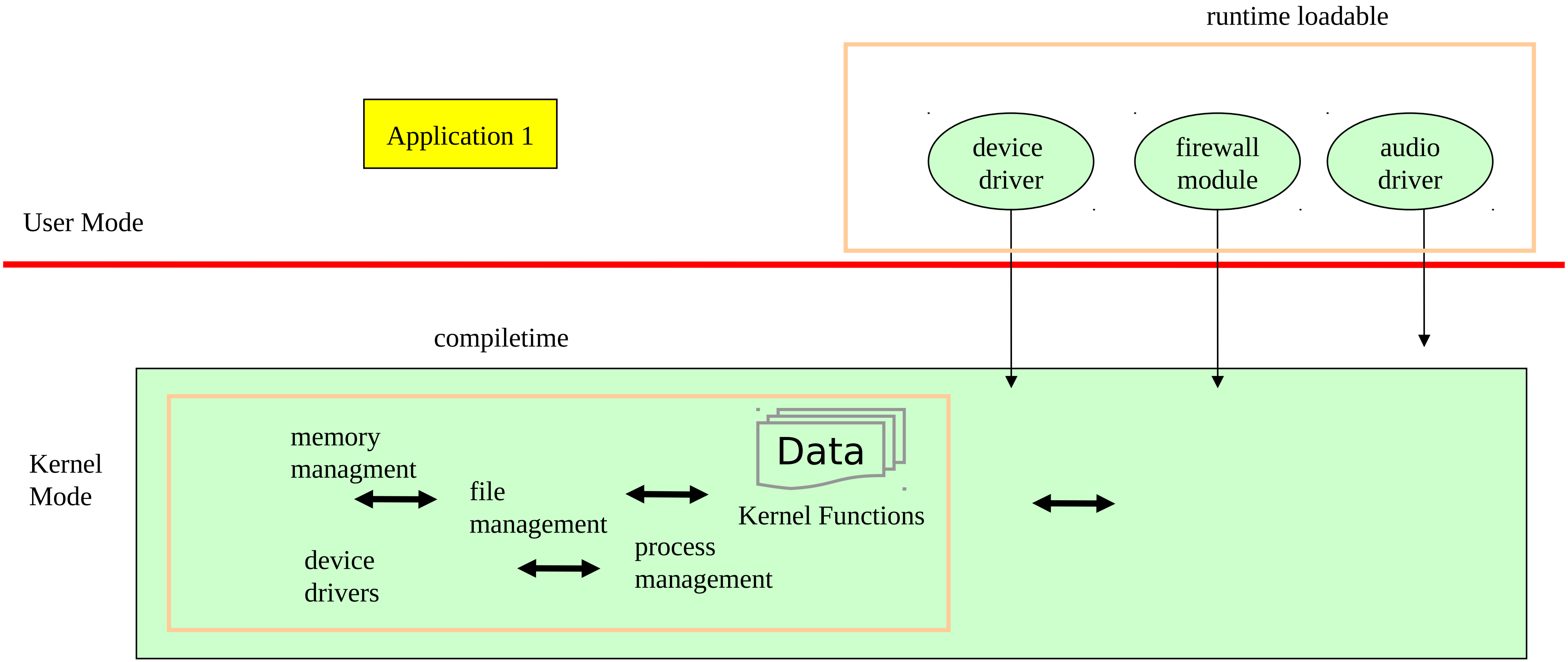
Physical Address Process B

Application 2: all addresses in same range as every other app.

256 MB Random Access Memory (physical)

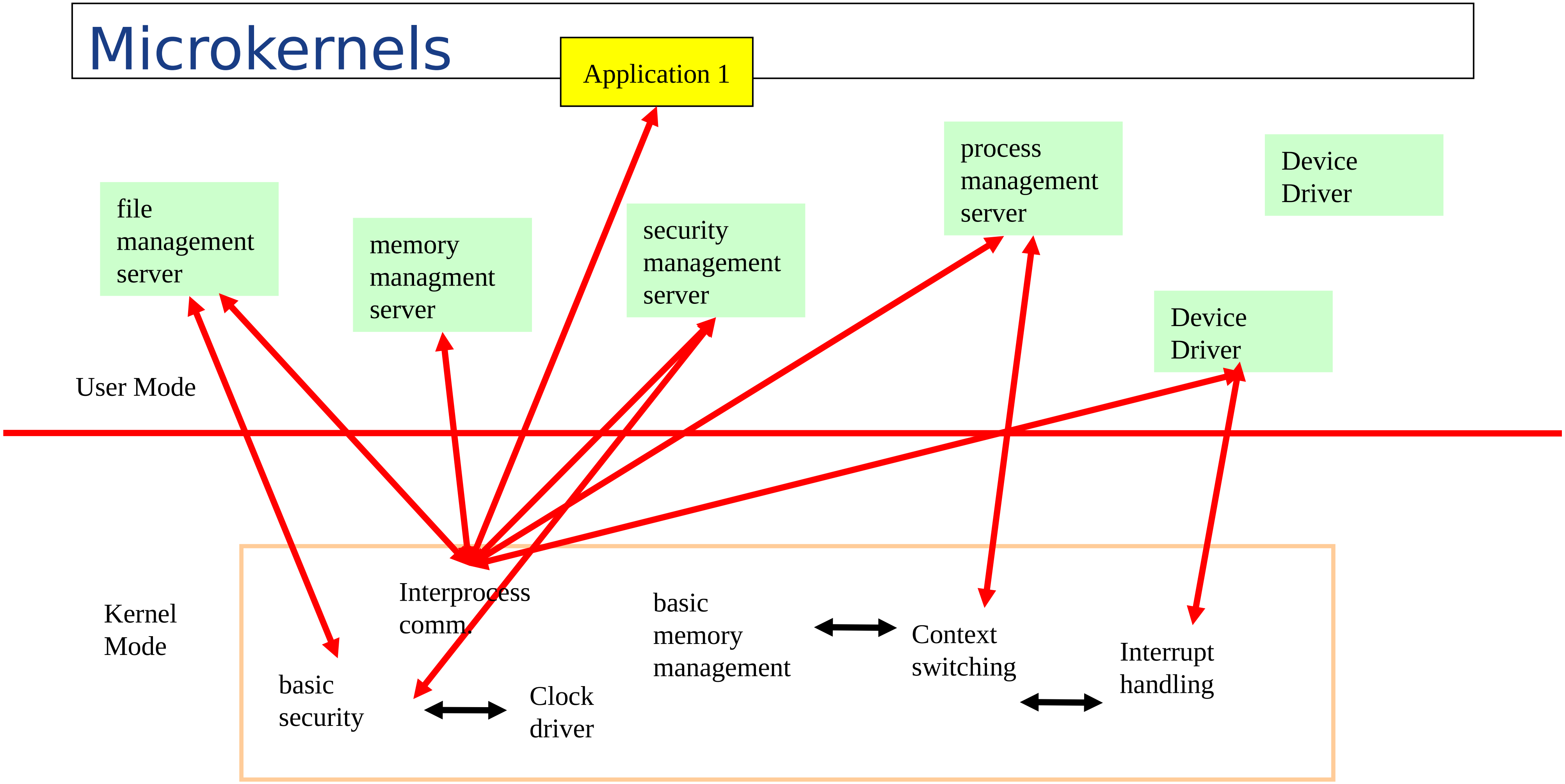
Application 1: all addresses in same range as app 1.

Monolithic Kernels



Fast access to functions and shared kernel data

Microkernels



Application 1

file management server

memory managment server

security management server

process management server

Device Driver

Device Driver

User Mode

Kernel Mode

Interprocess comm.

basic security

Clock driver

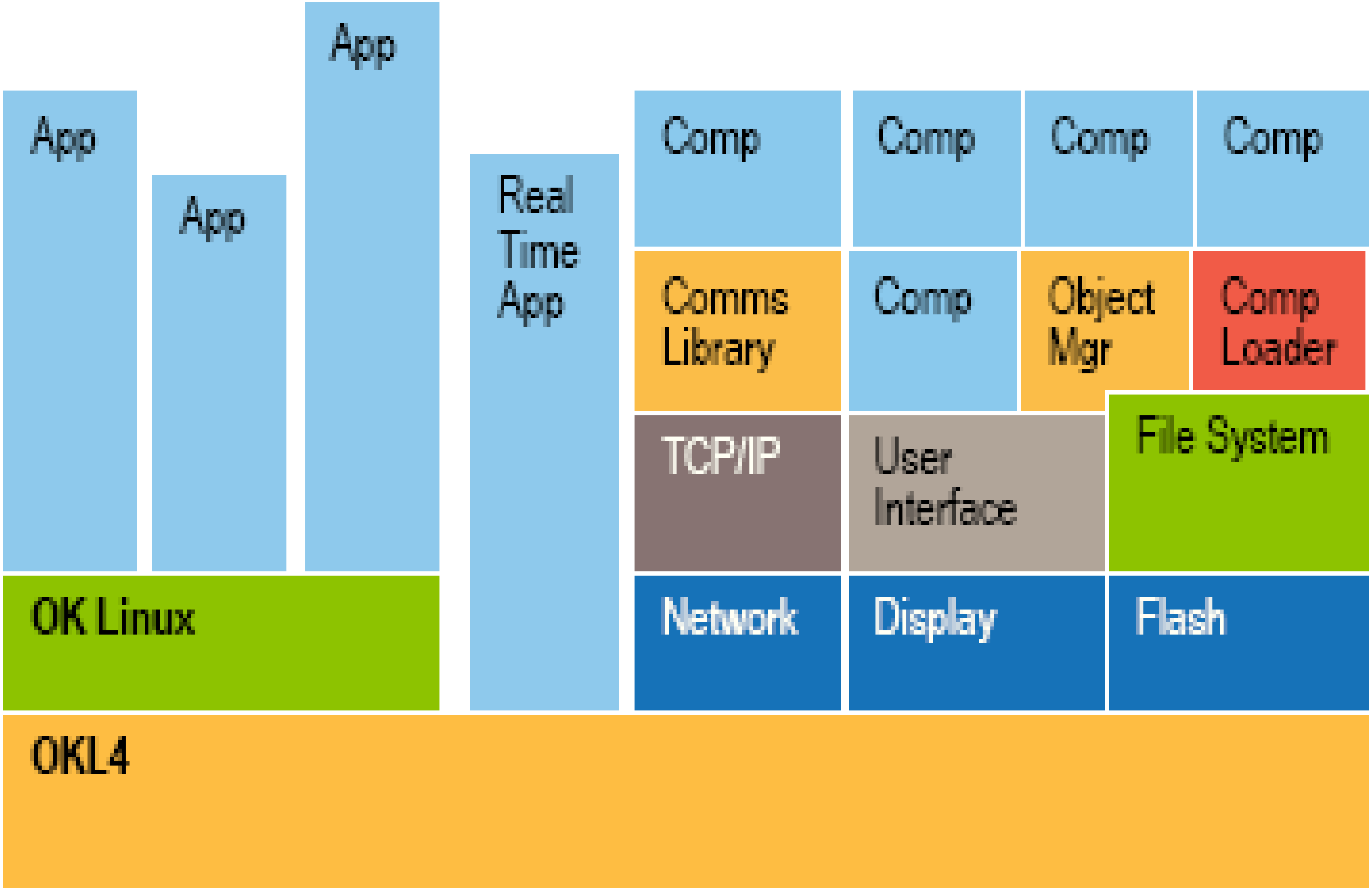
basic memory management

Context switching

Interrupt handling

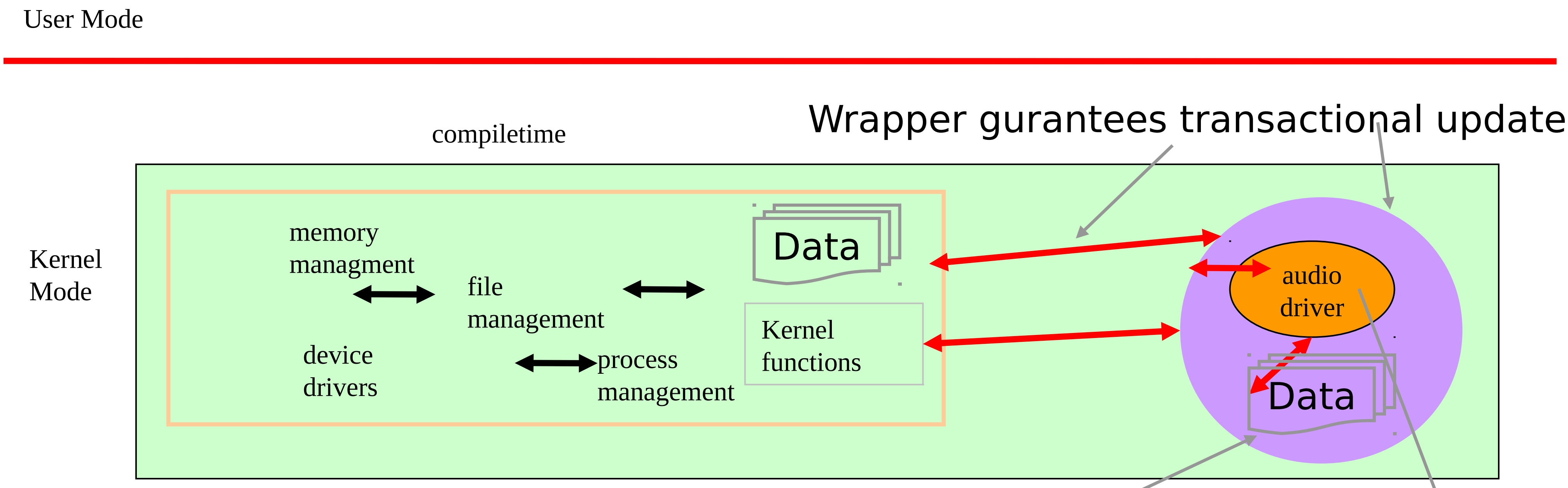
↔ Fast access to functions and shared kernel data

↔ Interprocess Communication or system calls



From: G.Heiser, Virtualization of Embedded Systems

Armored Monolithic Kernels



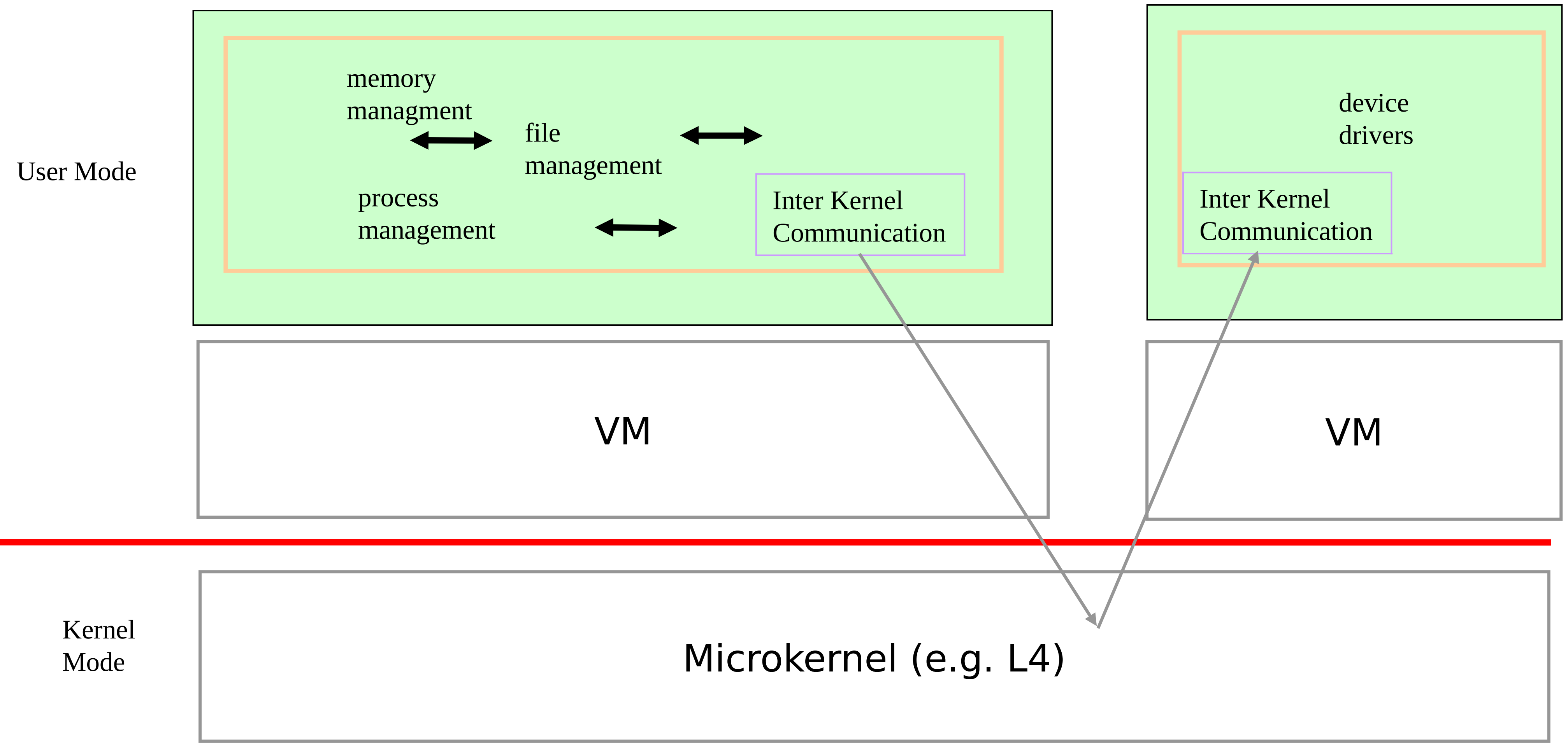
Wrapper gurantees transactional updates

Intercepted and controlled calls to and from kernel

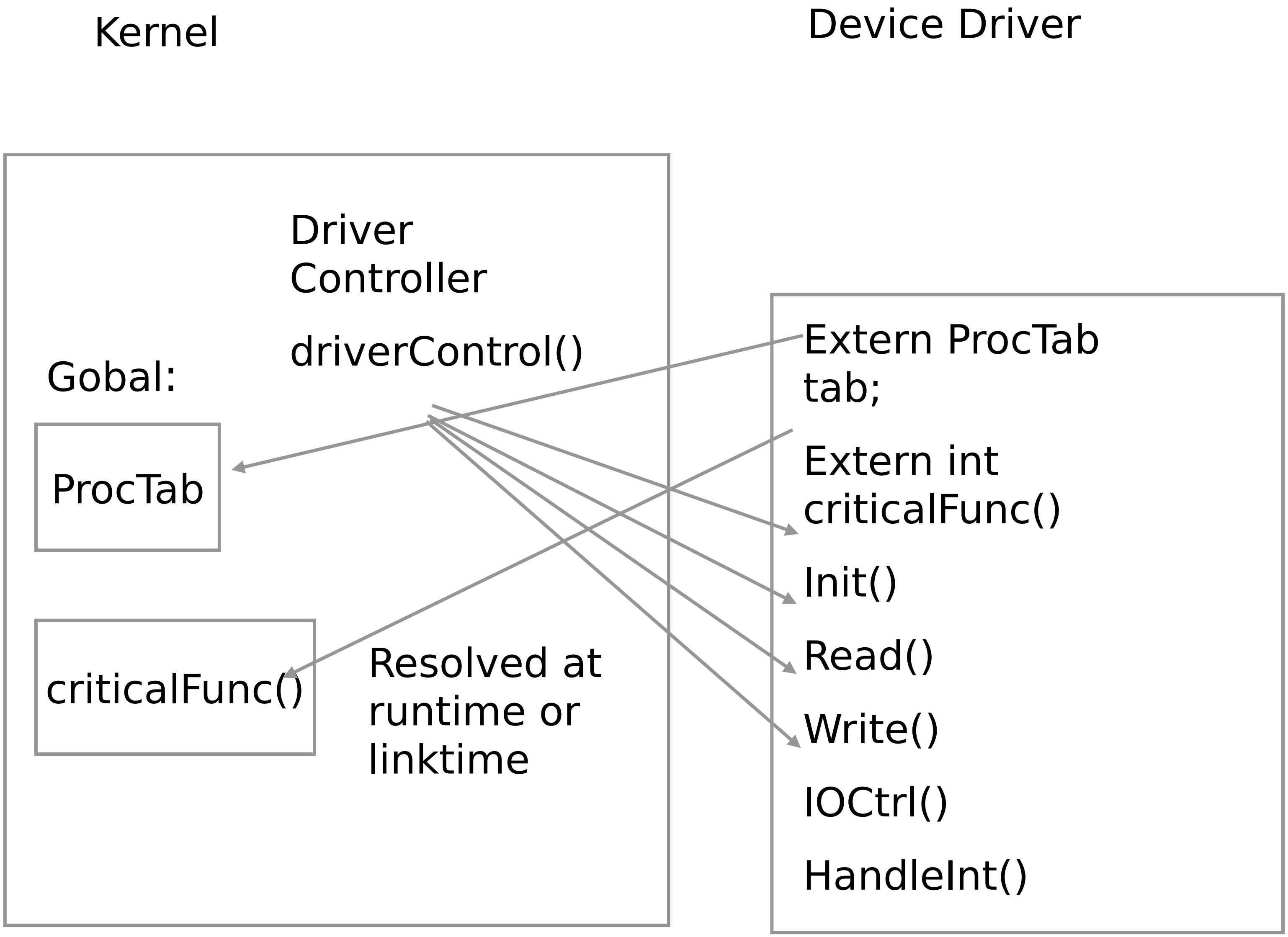
Temporary Copy of kernel data

Modified page table to restrict driver to certain kernel addresses

VM Armored Micro Kernels



The Secure Extension Problem



Kernel

Device Driver

Driver Controller

driverControl()

Gobal:

ProcTab

criticalFunc()

Resolved at runtime or linktime

Extern ProcTab tab;

Extern int criticalFunc()

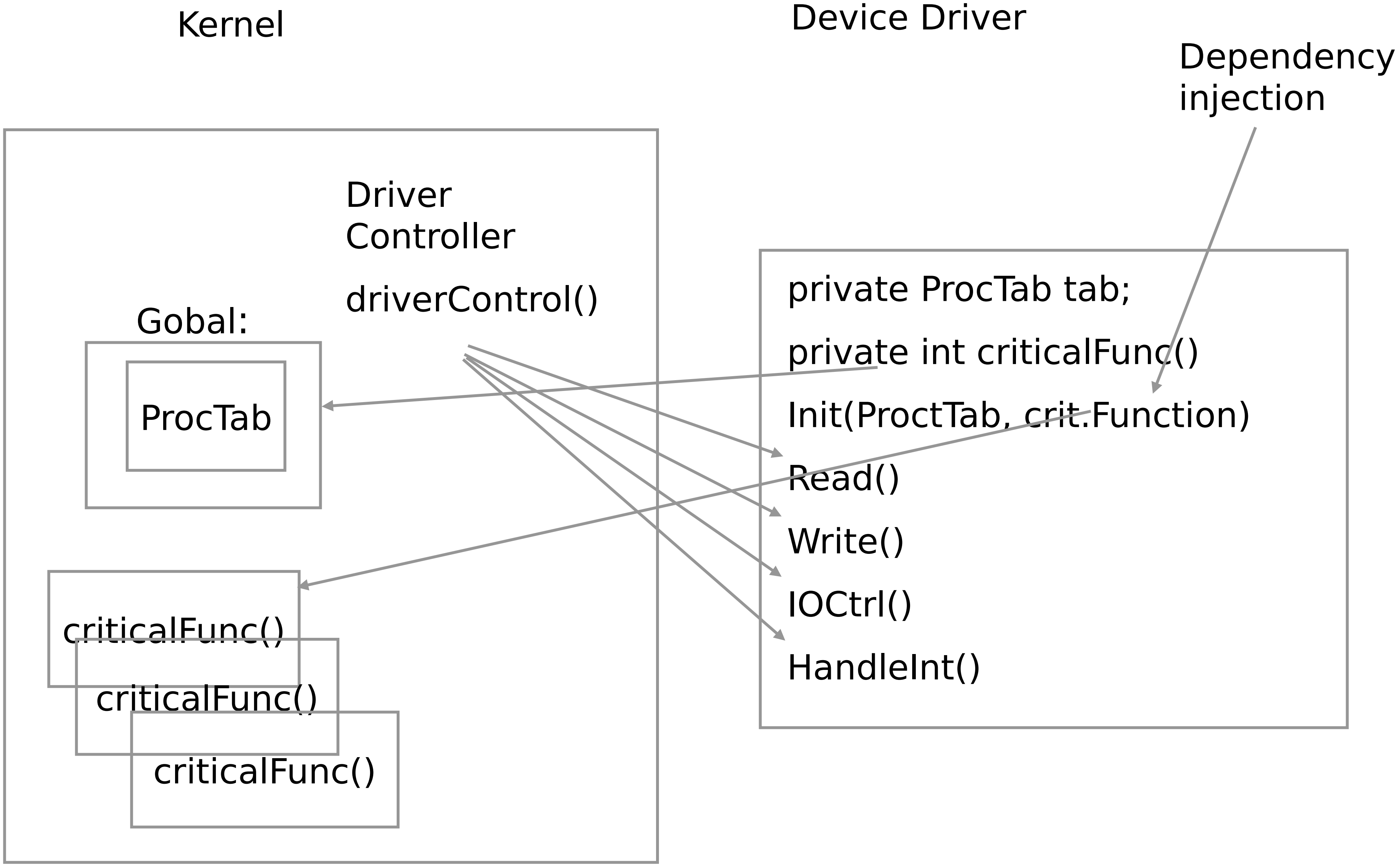
Init()

Read()

Write()

IOCtrl()

HandleInt()



Kernel

Device Driver

Dependency injection

Driver Controller

driverControl()

Gobal:

ProcTab

criticalFunc()

criticalFunc()

criticalFunc()

private ProcTab tab;

private int criticalFunc()

Init(ProctTab, crit.Function)

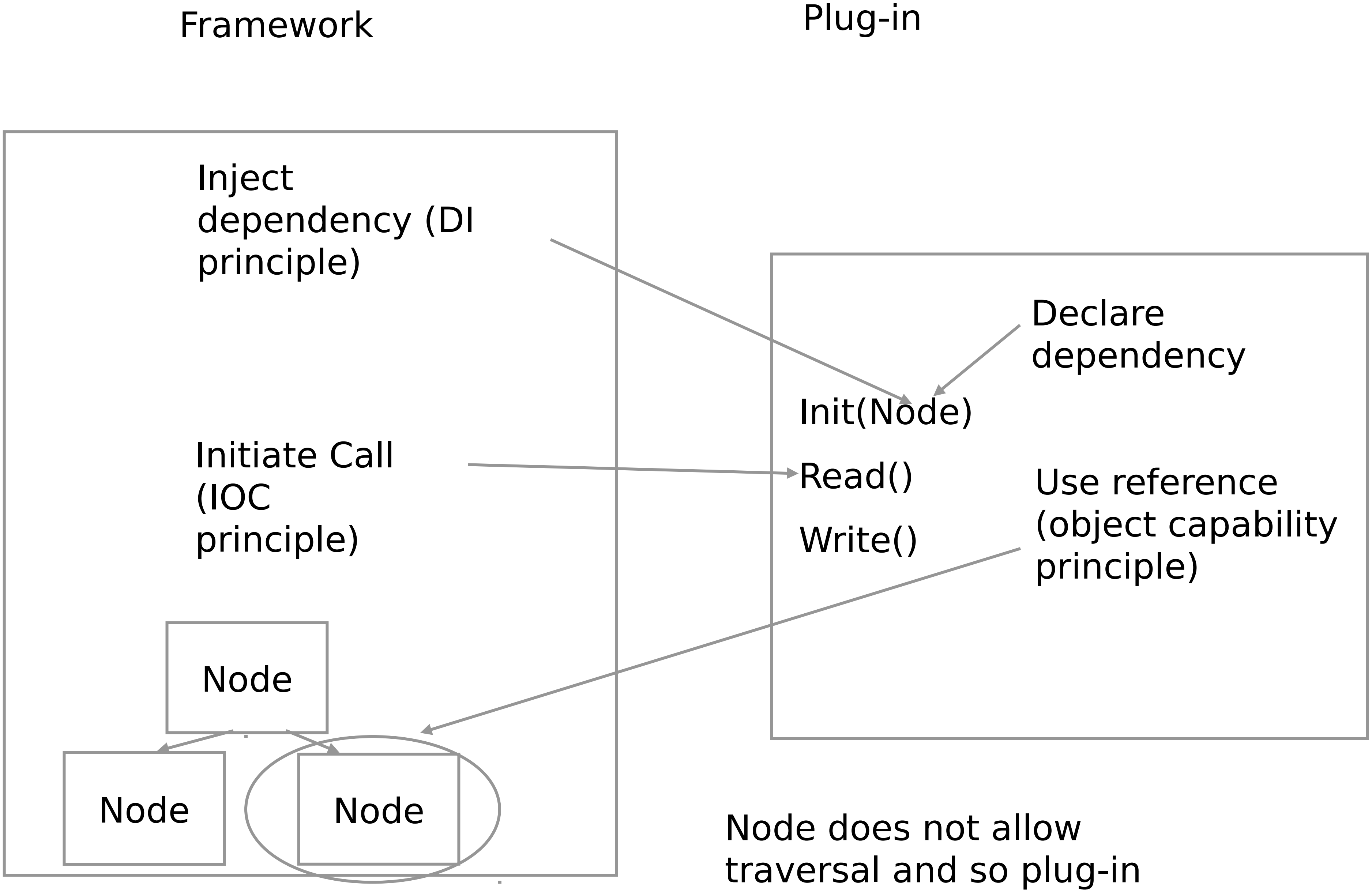
Read()

Write()

IOCtrl()

HandleInt()

- Der Driver kann ausschließlich über die Argumente und deren Methoden auf Kernel-Daten oder Funktionen zugreifen. Es gibt keine andere Möglichkeit, auch nicht durch die Navigation von globalen Verzeichnissen oder durch sog. Composite Objects (Design Pattern), die über ihre Tree-Navigation Zugriffe auf viele andere Objekte gestatten.
- Der Treiber macht eigene Abhängigkeiten und Bedürfnisse im Interface der init-Methode sichtbar: Dort steht, was der Treiber vom Kernel benötigt, um zu funktionieren.
- Der Kernel ist in der Lage, Proxy Objekte oder speziell für den Treiber zugeschnittene Closures (das sind initialisierte higher-order Functions oder deren objektorientiertes Äquivalent bzw. deren Simulationen durch Command Patterns) herzustellen und an den Treiber zu übergeben. Damit kontrolliert der Kernel komplett die Ausführungsumgebung des Drivers.



Framework

Plug-in

Inject dependency (DI principle)

Initiate Call (IOC principle)

Node

Node

Node

Declare dependency

Init(Node)

Read()

Write()

Use reference (object capability principle)

Node does not allow traversal and so plug-in cannot access parent node. The plug-in declares ist dependencies in its interface

Modes and Privileges

Same runtime environment for all applications

A common, navigable filesystem with ambient authority

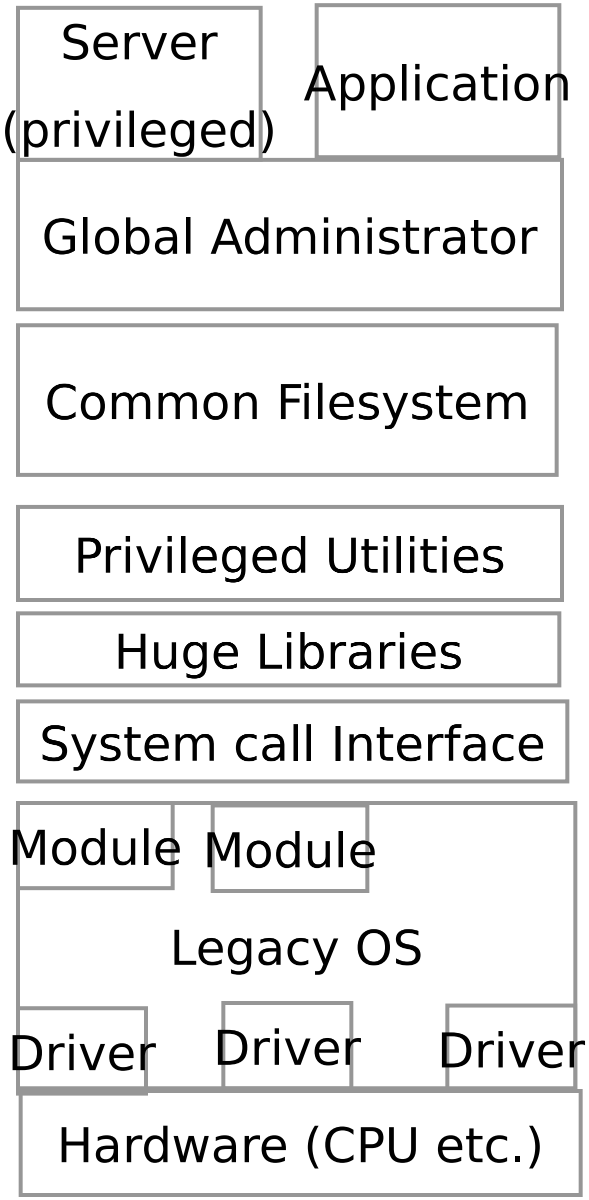
Tons of unsafe but privileged scripts and utilities (setUid)

>600 complex system calls

Countless dynamically loadable modules

>100.000 drivers for windows

Network scans (IP seq.)



Server

(privileged)

Application

Global Administrator

Common Filesystem

Privileged Utilities

Huge Libraries

System call Interface

Module

Module

Legacy OS

Driver

Driver

Driver

Hardware (CPU etc.)

Huge TCB, 2 modes only

Cycle stealing applications create a problem for near-realtime multimedia applications

Dangerous accounts, single audit and log features

Lots of unverified system libraries with memory leaks etc.

Incomplete quota administration

Attacks on random number generation

Unsafe kernel languages (memory)

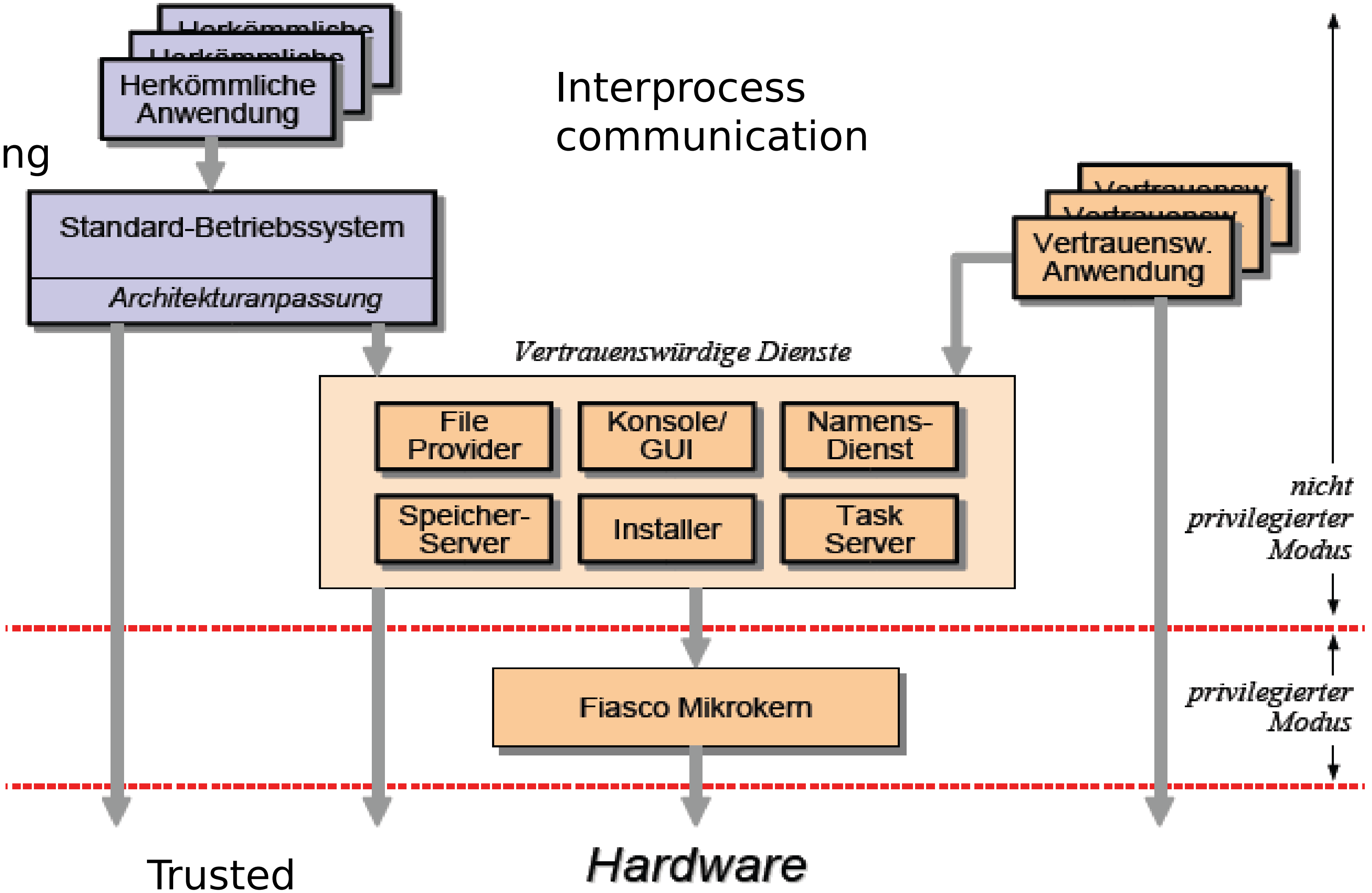
Covered channels (cache, bios, CPU)

Driver.c:

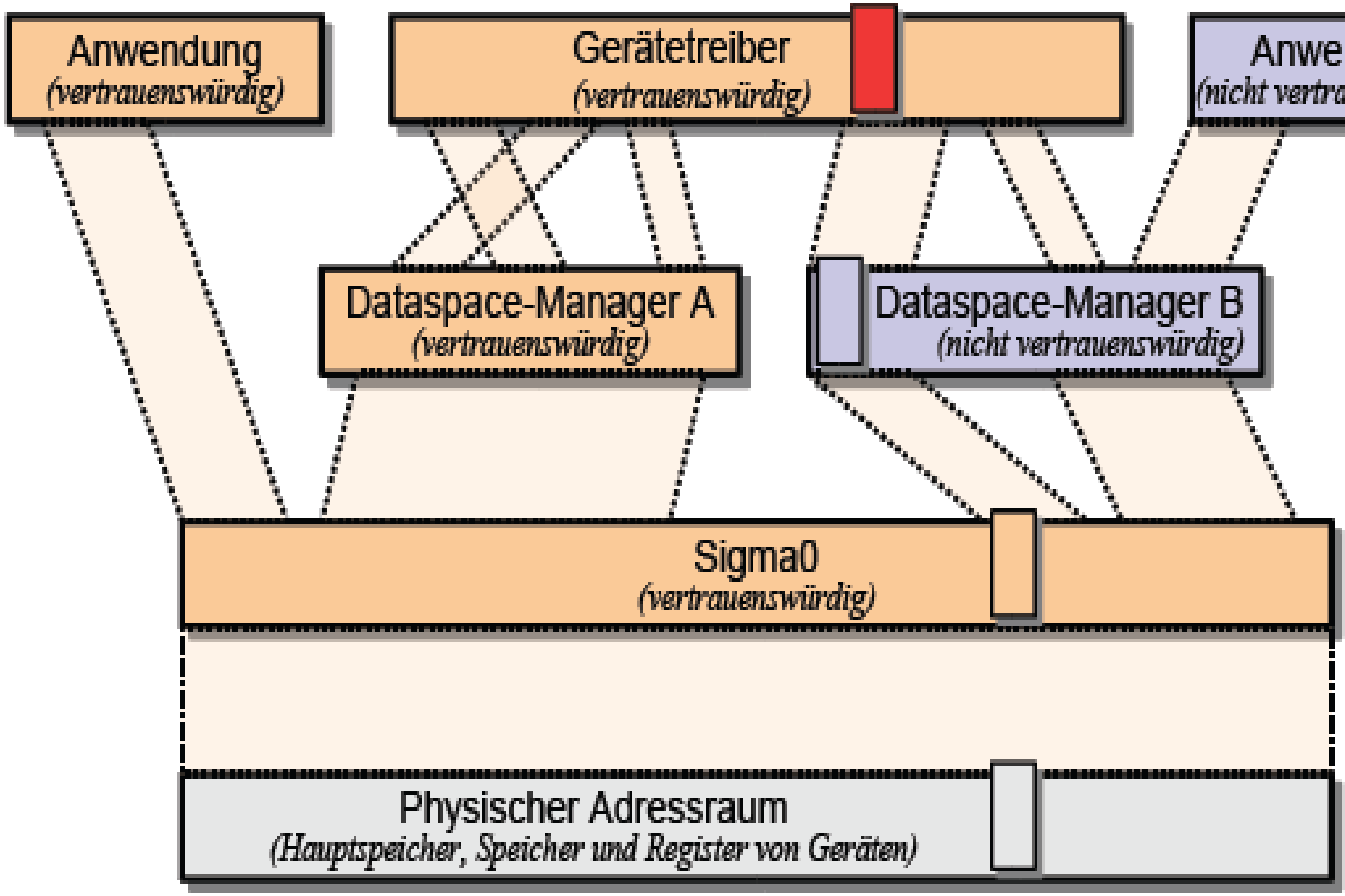
```
If (uid = 0) {
// do something harmless
}
// run as root now!
```

Para-virtualisierung

Interprocess communication



Trusted Computing Base


Anwendung
(vertrauenswürdig)
Gerätetreiber
(vertrauenswürdig)
Anwendung
(nicht vertrauenswürdig)
Dataspace-Manager A
(vertrauenswürdig)
Dataspace-Manager B
(nicht vertrauenswürdig)
Sigma0
(vertrauenswürdig)
Physischer Adressraum
(Hauptspeicher, Speicher und Register von Geräten)

Rules for Virtualization/Guest Operating Systems

VMs usually work on some privileged account should be treated like other services which can be compromised (chroot, systrace etc.)

VMs like guest operating systems should be configured and installed as small as possible to keep the attack surface small. Unused drivers etc. need to be removed.

The Integrity of guest systems needs to be secured like the hosts themselves (updates, control)

Systems which support different levels of privileges (XEN) are better. Security levels from platforms (e.g. Berkeley Security Levels) should be used. [Orm]

Ressources: Ormandy

Software-based Isolation: Singularity

Singularity Features

Die Kapselung beruht auf leichtgewichtigen Prozessen

Kernel, Applikationen, Treiber und System-Server sind allesamt Prozesse

Ein Prozess ist ein geschlossener Object-Space, der keine Referenzen in andere Prozesse hinein besitzt

IPC ist durch ein Kanalkonzept realisiert, das sich durch streng typisierte Kommunikation, definiert durch State-Machines, auszeichnet

Objekte können an andere Prozesse übertragen werden, wechseln dann jedoch den Besitzer, so dass keine cross-process synchronization Probleme oder zentrale Garbage Collection nötig sind.

Ein kleiner Teil Code ist unverified und trusted, der große Rest von Microkernel und Applikationen etc. ist verifiably trusted, d.h. er wurde vom sicheren Compiler erzeugt.

Es sind keine dynamischen Erweiterungen von Prozessen (Code laden) erlaubt. Jede Erweiterung muss ein eigener Prozess sein.

Jede Softwarekomponente ist durch Meta-Daten umfangreich beschrieben und wird ohne eigenen Installer installiert.

Small, light-weight processes

Safe interprocess communication (separation)

Individual runtimes (GC, libraries)

No cross object space sharing

One virtual address space

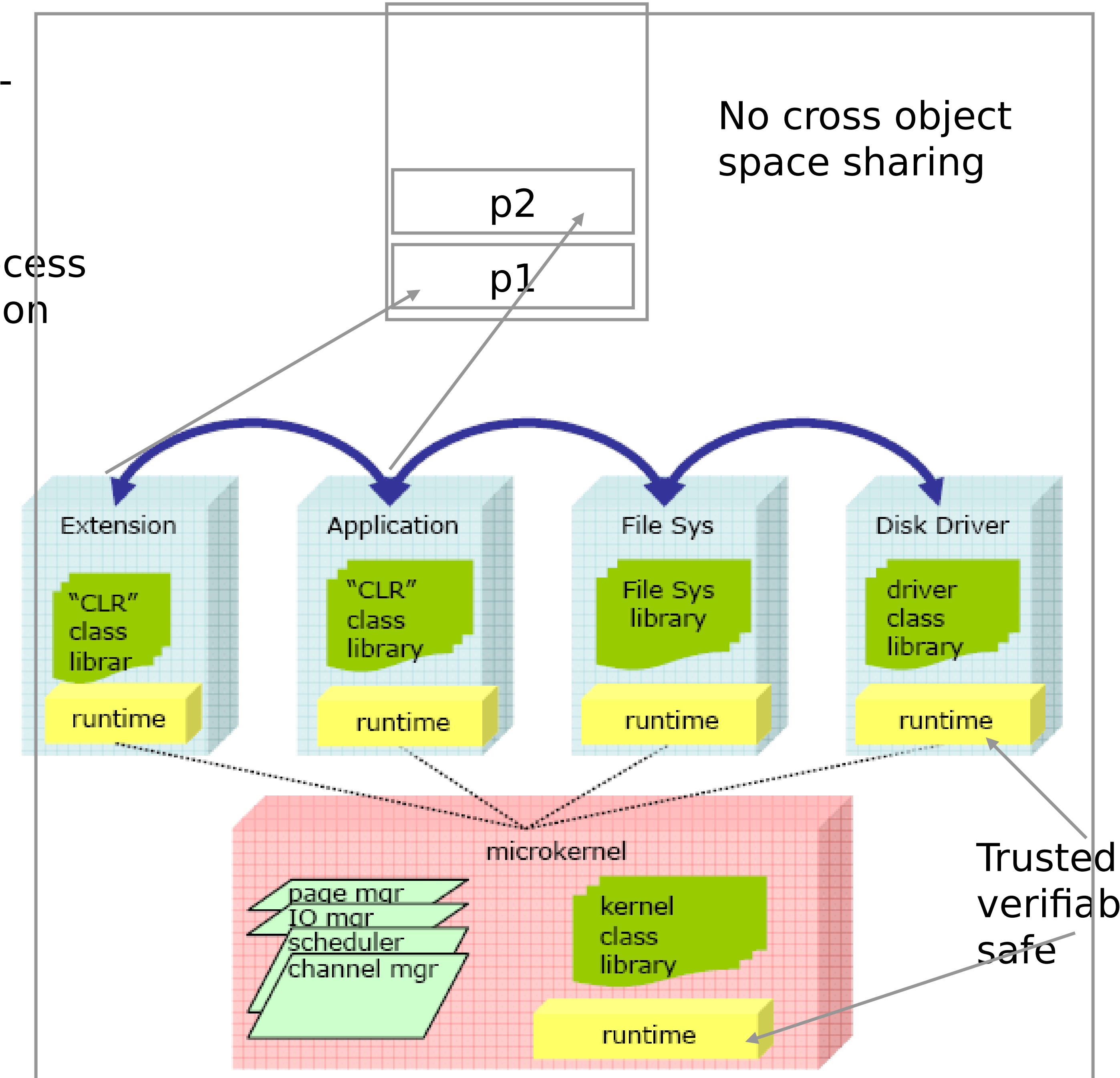
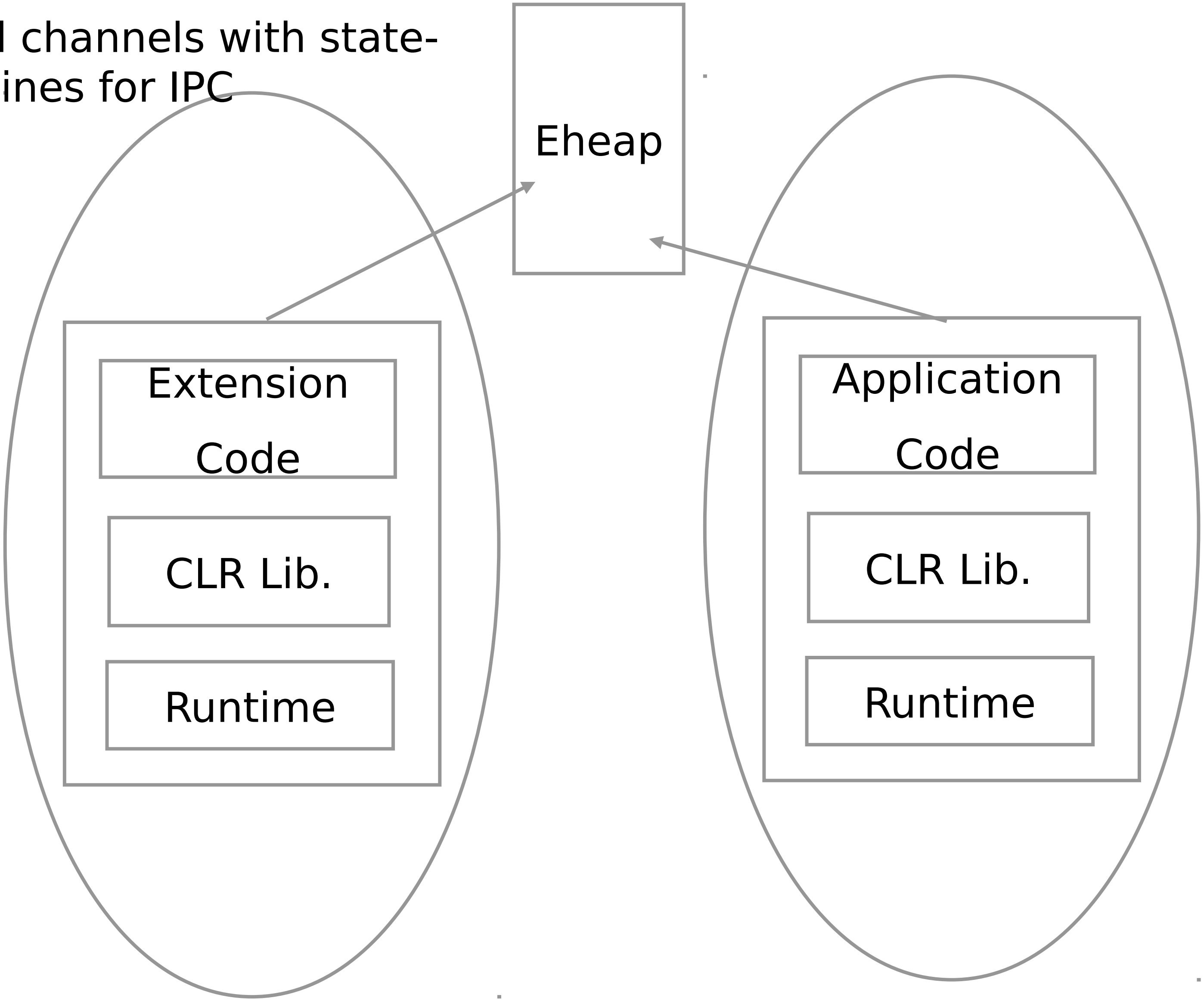


Figure 1 Singularity architecture.

Trusted, not verifiably safe

Typed channels with state-machines for IPC



No common, shared variables between both object spaces

```
public contract NamespaceContract : ServiceContract {
in message Bind(char[] in path, ServiceContract.Exp:Start exp);
out message AckBind();
out message NakBind(ServiceContract.Exp:Start exp);
in message Notify(char[] in pathSpec, NotifyContract.Imp:Start imp);
out message AckNotify();
out message NakNotify(NotifyContract.Imp:Start imp);
in message Find(char[] in pathSpec);
out message AckFind(FindResponse[] in results);
out message NakFind();
out message Success();
override state Start: one {
Success! -> Ready;  }
state Ready: one {
Bind? -> ( AckBind! or NakBind! ) -> Ready;
Find? -> ( AckFind! or NakFind! ) -> Ready;
Notify? -> ( AckNotify! or NakNotify! ) -> Ready; }}
```

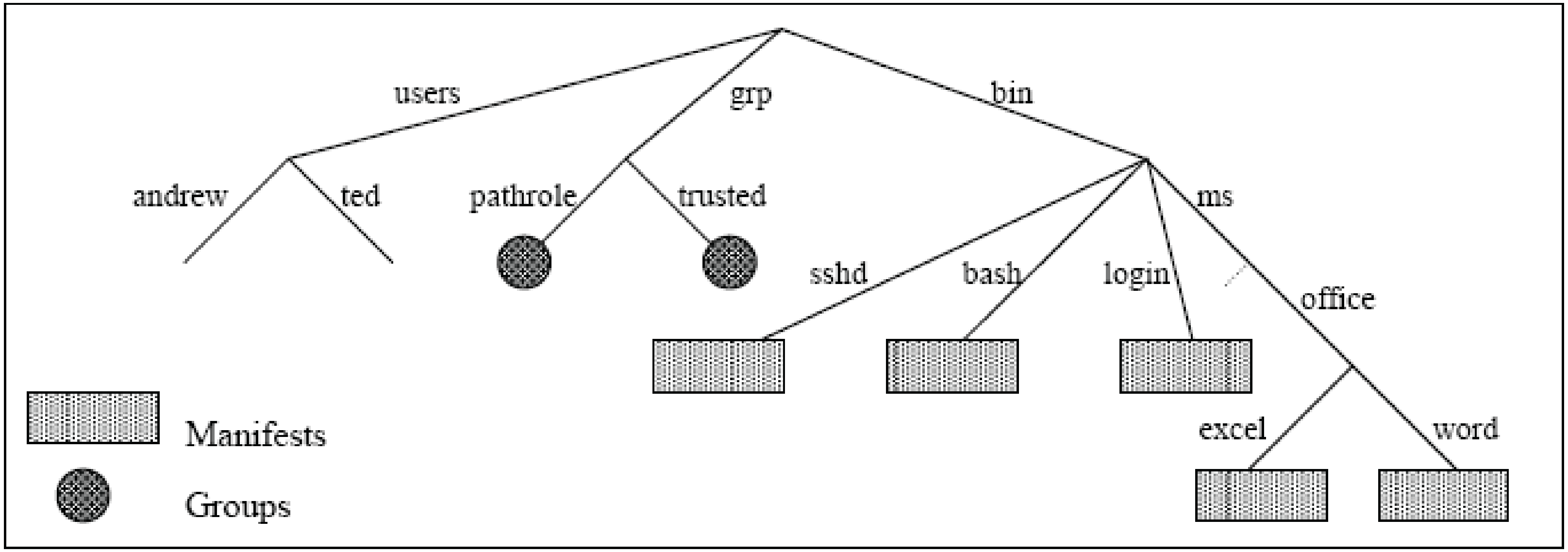
Als eingebettete Klassen eines Kanals werden jeweils ein *exporter* bzw. *importer* definiert, die die entsprechenden Messages des Kontrakts zugeordnet bekommen. Will ein Prozess einen Kanal eröffnen, dann erzeugt er Instanzen von exporter und importer und schickt die exporter-Instanz über einen existierenden Kanal an den Empfänger. Danach können beide miteinander kommunizieren.

```
<manifest>

<application identity="S3Trio64" />

<assemblies>

<assembly filename="S3Trio64.exe" />

<assembly filename="Namespace.Contracts.dll" version="1.0.0.2299„ />

<assembly filename="Io.Contracts.dll" version="1.0.0.2299" /> …..

<assembly filename="ILHelpers.dll" version="1.0.0.2299" />

<assembly filename="Singularity.V1.ill" version="1.0.0.2299" />

</assemblies>

<driverCategory>

<device signature="/pci/03/00/5333/8811" />

<ioMemoryRange baseAddress="0xb8000" rangeLength="0x8000„ fixed="True" />

<ioPortRange baseAddress="0x4ae8" rangeLength="0x2" fixed="True" /> ….

<extension startStateId="3" contractName="Microsoft.Singularity-.Extending.ExtensionContract"
endpointEnd="Exp„ assembly="Namespace.Contracts" />

<serviceProvider startStateId="3" contractName="Microsoft-.Singularity.Io.VideoDeviceContract"
endpointEnd="Exp„ assembly="Io.Contracts" /> </driverCategory> </manifest>
```

A manifest file describes the physical structure of an application and its installation needs. Control is never passed to the application for installation purposes (no setup.exe)

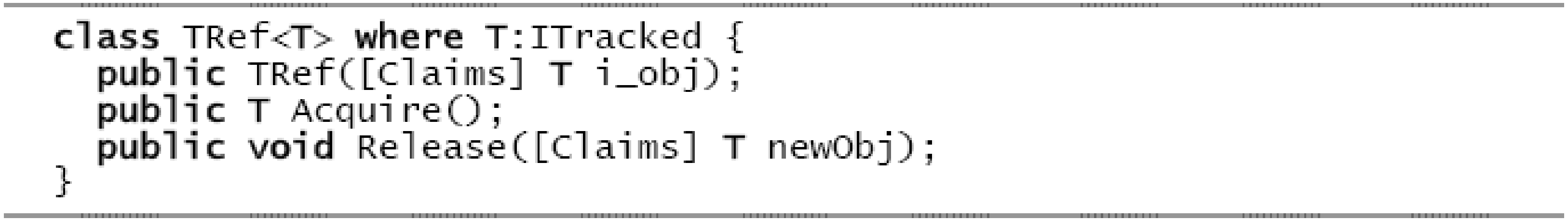
/bin/sshd @ /users/ted (+ {/grp/pathrole})* + /bin/ms/office/word

Compound Principal



An example naming tree

Tracked data structures with ownership transfer



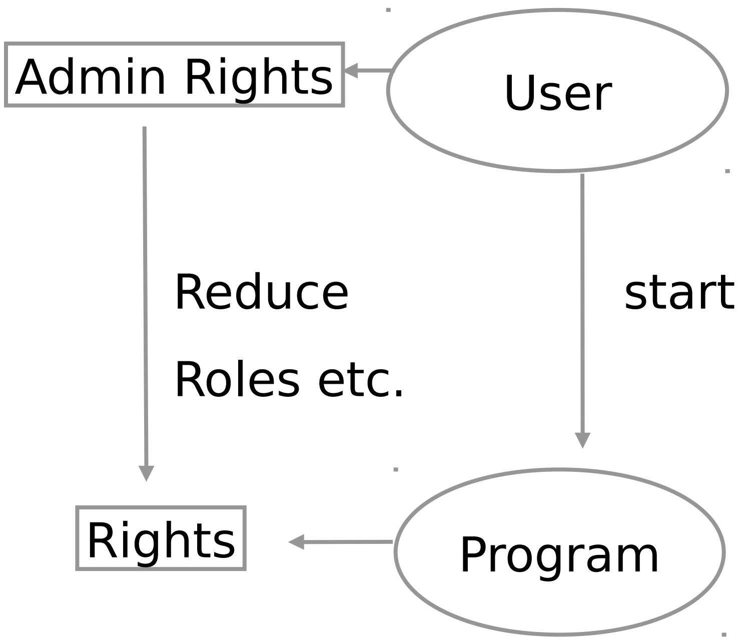
```
class TRef<T> where T:ITracked {
  public TRef([Claims] T i_obj);
  public T Acquire();
  public void Release([Claims] T newObj);
}
```

When creating a TRef<T>, the constructor requires an object of type T as an argument. The caller must have ownership of the object at the construction site. After the construction, ownership has been passed to the newly allocated TRef. The Acquire method is used to obtain the contents of a TRef. If the TRef is full, it returns its contents and transfers ownership to the caller of Acquire. Afterwards, the TRef is said to be empty. Release transfers ownership of a T object from the caller to the TRef. Afterwards, the TRef is full. TRefs are thread-safe and Acquire operations block until the TRef is full.

Privilege Reduction vs. Capabilities

Privilege Restriction and Separation with Identities or ACLs

DecreaseMyRights



Admin Rights

User

Reduce

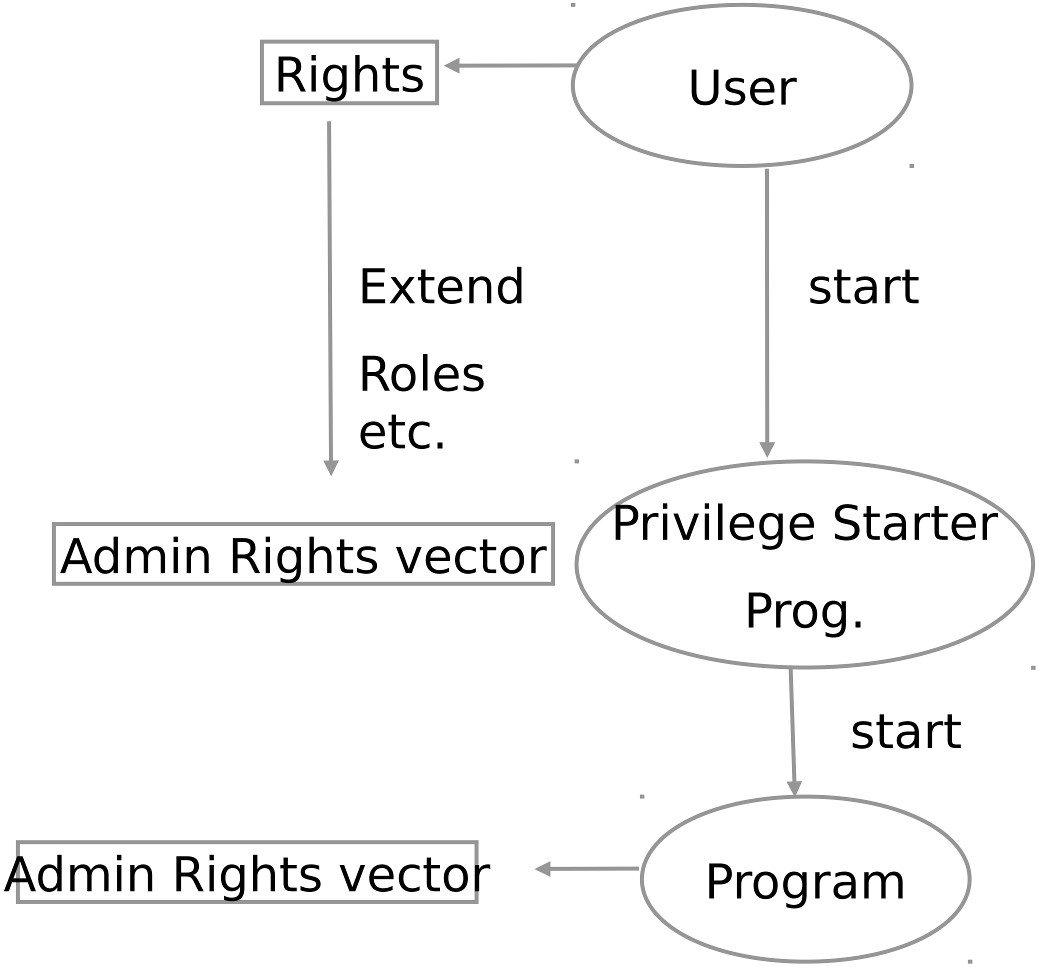
Roles etc.

start

Rights

Program

MakeMeAdmin



Rights

User

Extend

Roles etc.

start

Admin Rights vector

Privilege Starter Prog.

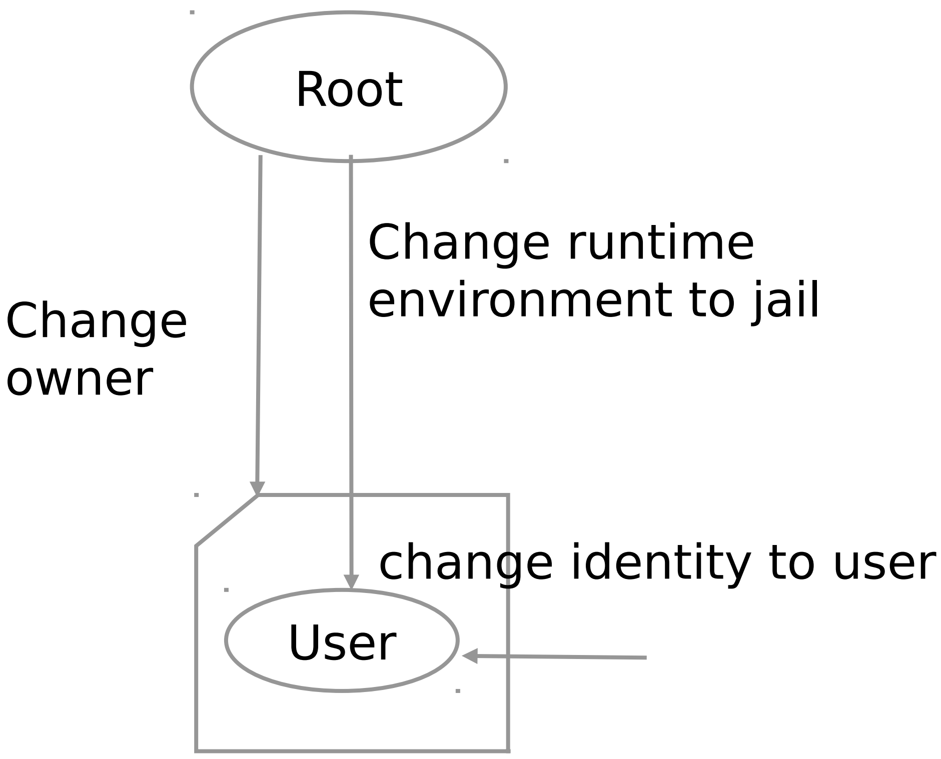
start

Admin Rights vector

Program

Privilege Restriction and Separation with Rights Reduction vs Capabilities

Subtractive Model



Root

Change owner

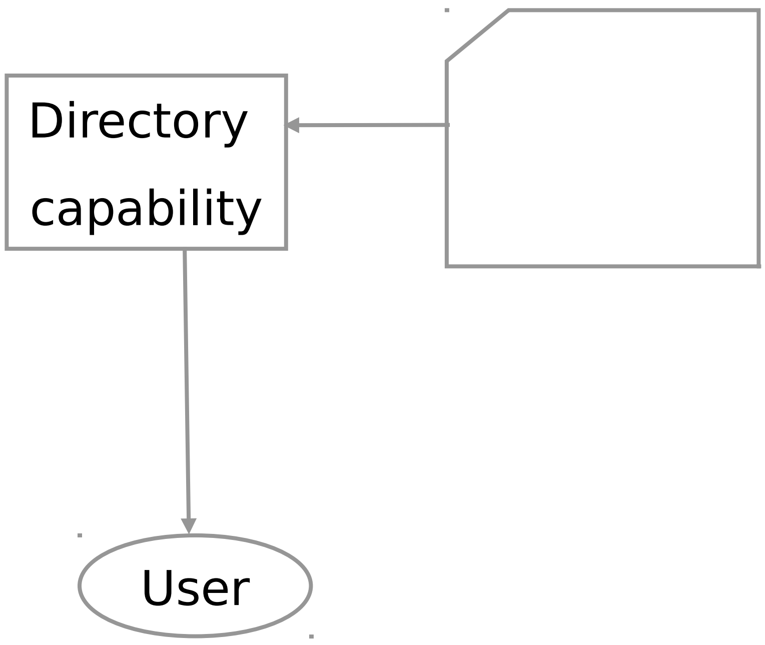
Change runtime environment to jail

change identity to user

User

Jails strips off other rights

additive Model (default is deny)

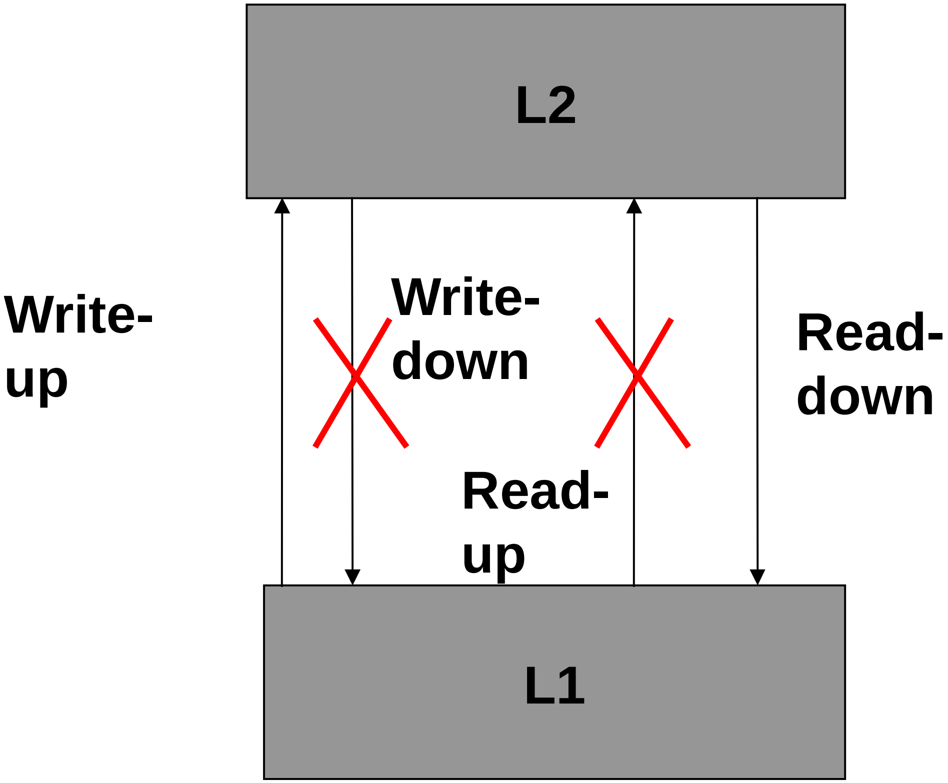


Directory capability

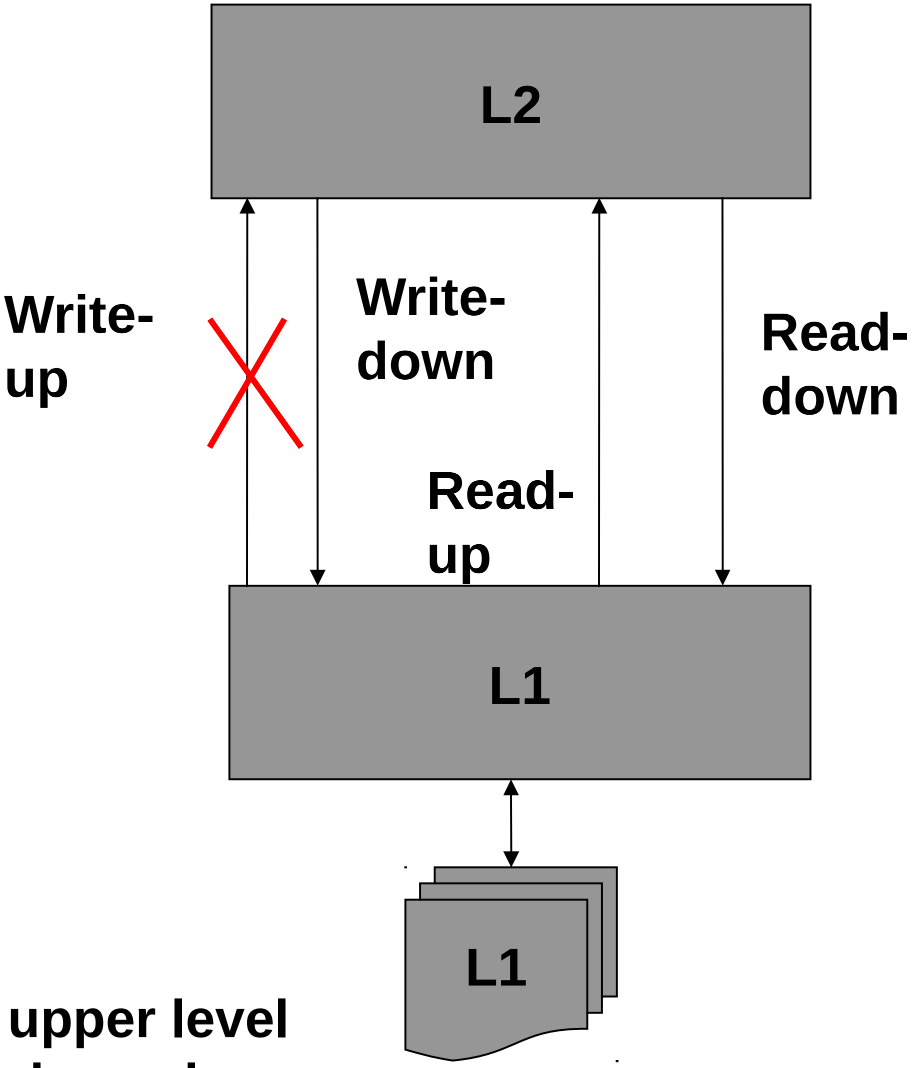
User

Hand directory capability to unprivileged user process. Without capabilities user process cannot access resources

Classic *-properties

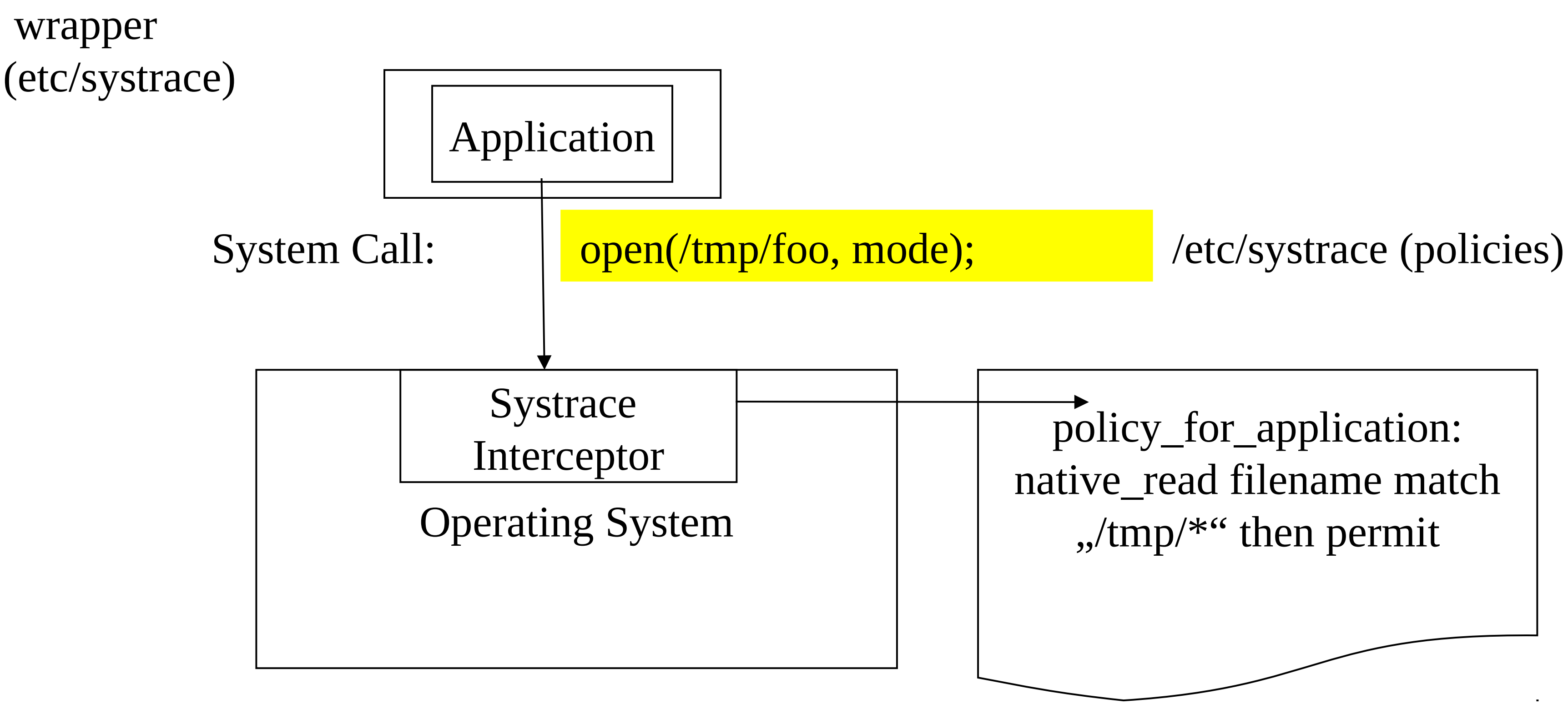


Vista Integrity Levels



Vista: Lower level unprotected against upper level attacks. Upper level open to luring attacks and data exposure. Ambient authority through queries and messages

Systrace in OpenBSD



wrapper
(etc/systrace)

Application

System Call:

open(/tmp/foo, mode);

/etc/systrace (policies)

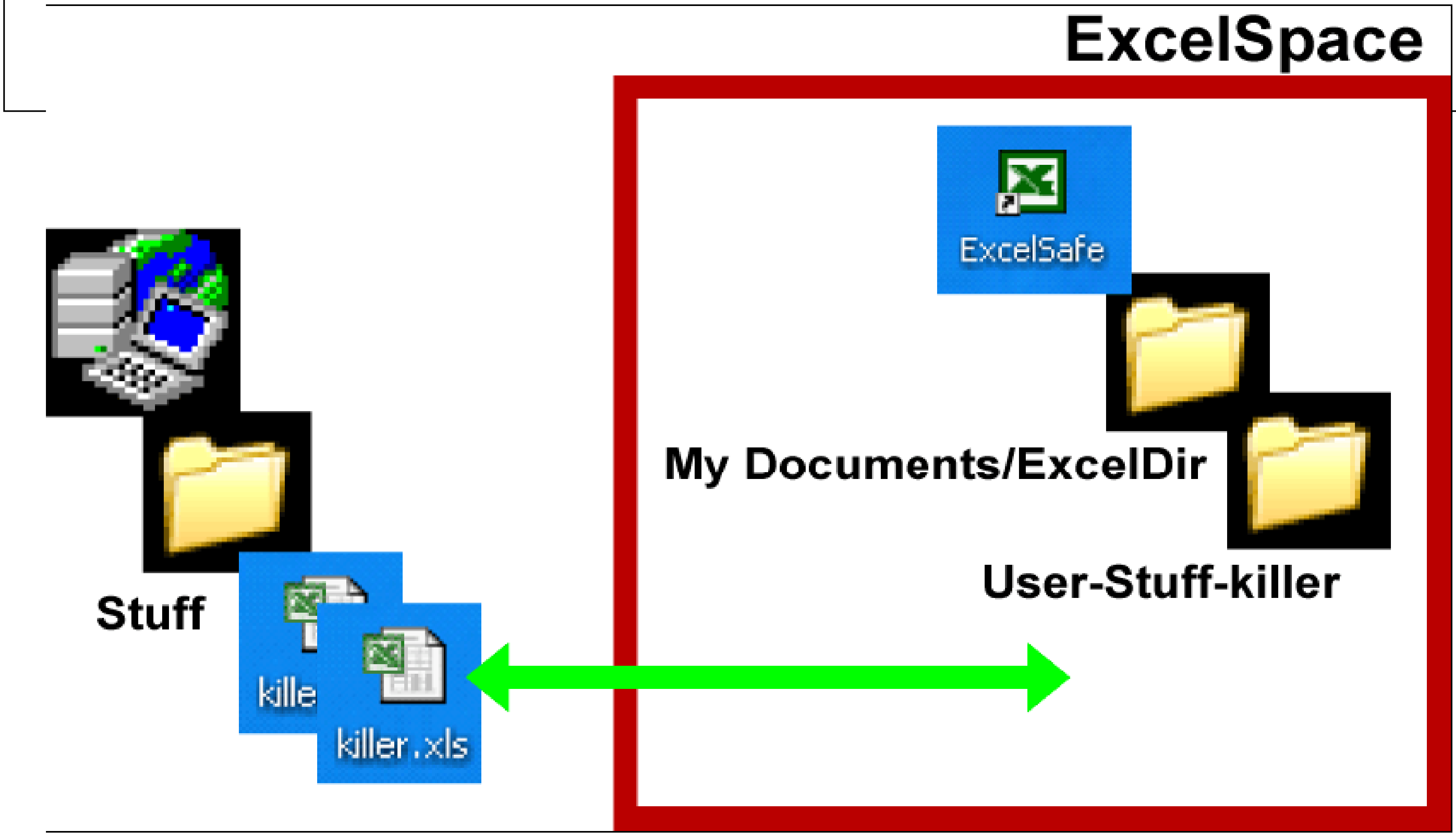
Systrace
Interceptor

Operating System

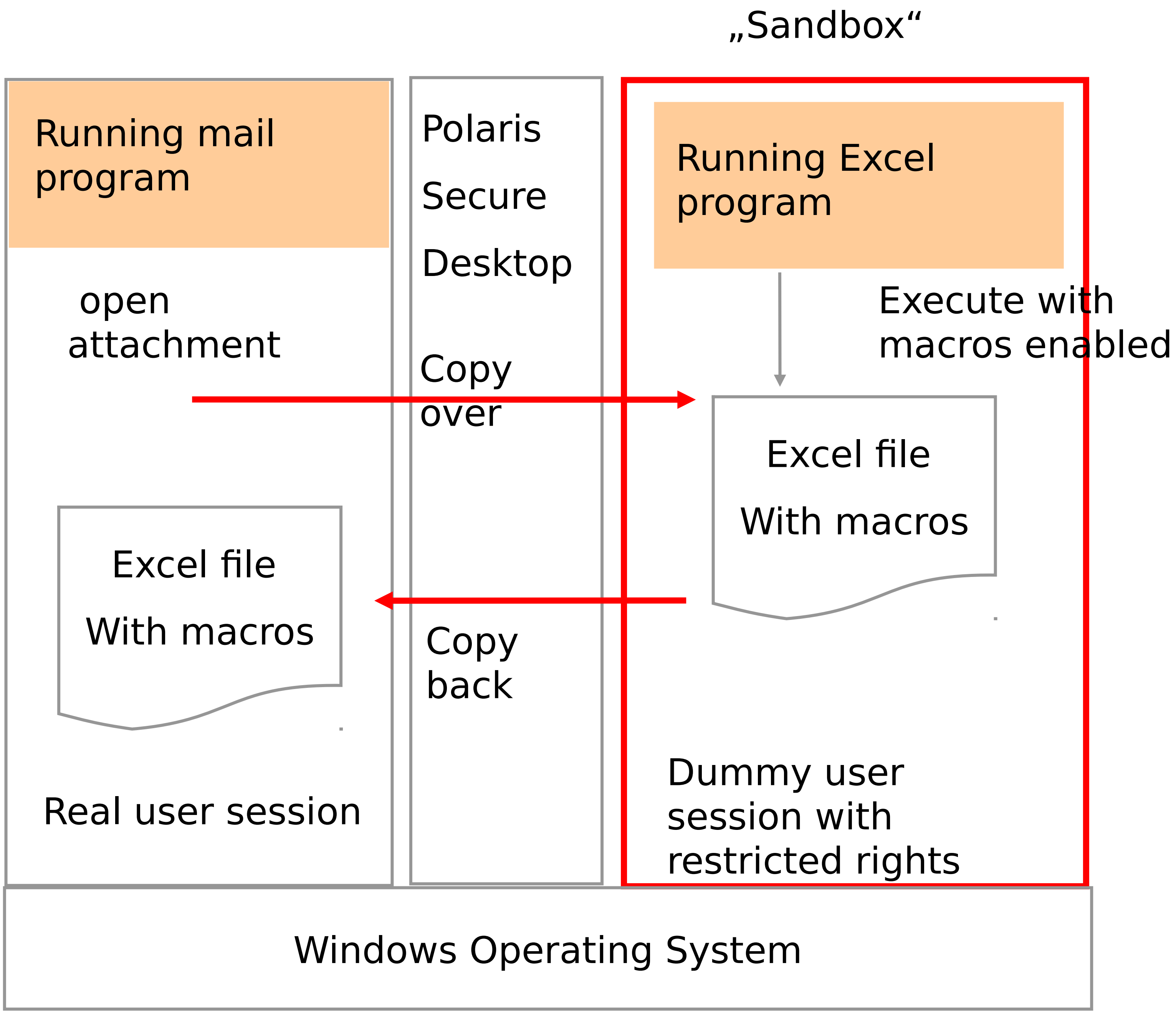
policy_for_application:
native_read filename match
„/tmp/*“ then permit

Systrace intercepts system calls and evaluates them according to specified policies. By wrapping user shells, applications and daemons most code can be easily sandboxed.

Polaris



The file is edited in the context of a powerless dummy user and then copied back. The application gets the file via the so called „powerbox“ – a GUI component that distributes capabilities upon user request.



„Sandbox“

Running mail program

open attachment

Excel file

With macros

Real user session

Polaris

Secure

Desktop

Copy over

Copy back

Running Excel program

Execute with macros enabled

Excel file

With macros

Dummy user session with restricted rights

Windows Operating System

Secure Server Design and Reduction of Exposure by Software-Architecture

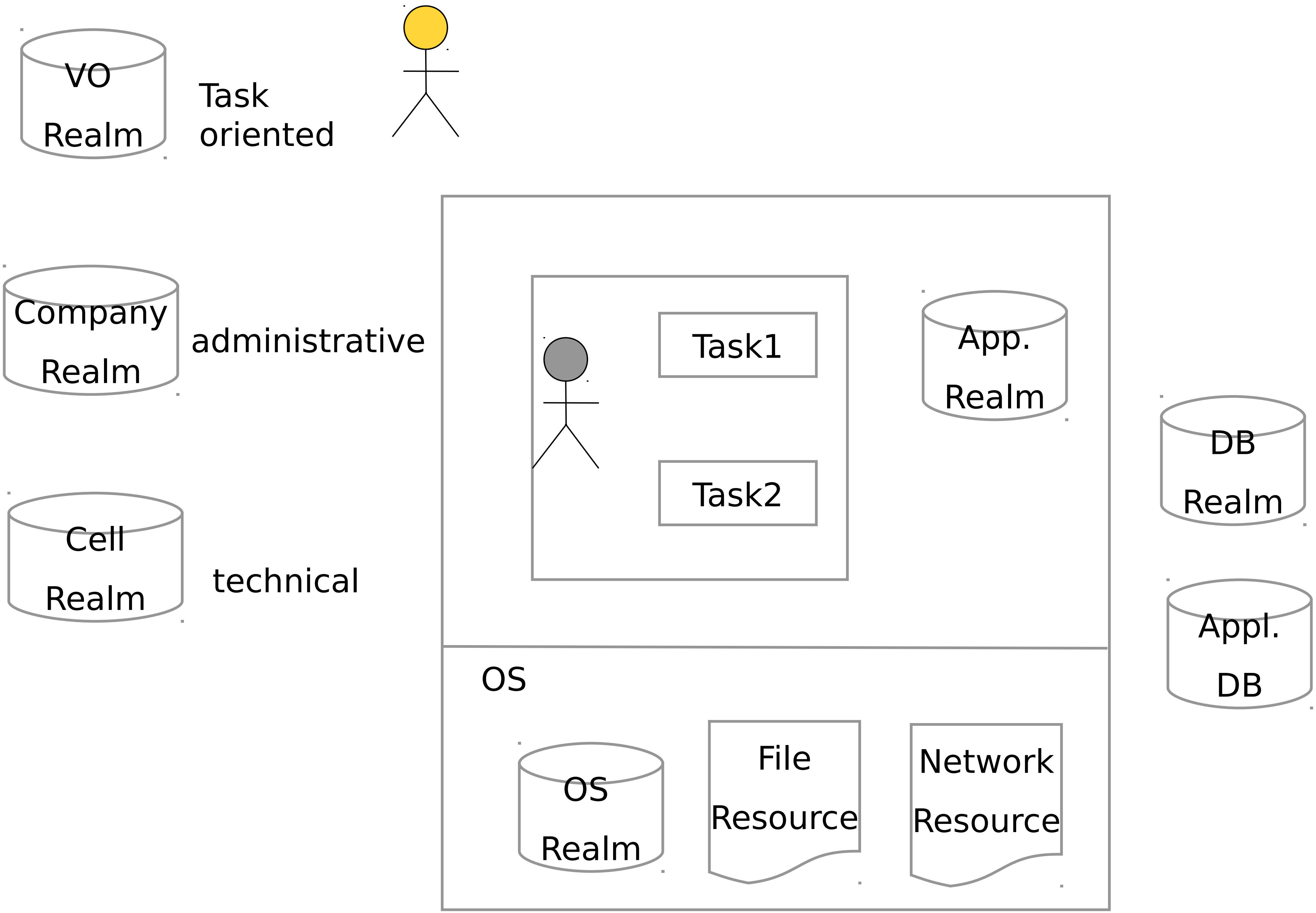
Security principles for server design

- Isolation of server processes with chroot or jails
- Server processes run as unprivileged user
- Database access with minimal rights
- Different functionalities should run on different servers

Nach: Maxwell Krohn, Building Secure High-Performance Web Services with OKWS

```
      ServerSocket ss = new ServerSocket(port);
      for(;;) {
          Socket client = ss.accept();
          BufferedReader in = new BufferedReader(
             new InputStreamReader(client.getInputStream()));
          PrintWriter out = new PrintWriter(client.getOutputStream());
          while((line = in.readLine()) != null) {
             // read command (or filename) from network
 // execute command
}
          out.print(result);
       }
```

David Flanagan, Java Examples in a Nutshell (shortened)



VO Realm

Task oriented

Company Realm

administrative

Cell Realm

technical

Task1

Task2

App. Realm

DB Realm

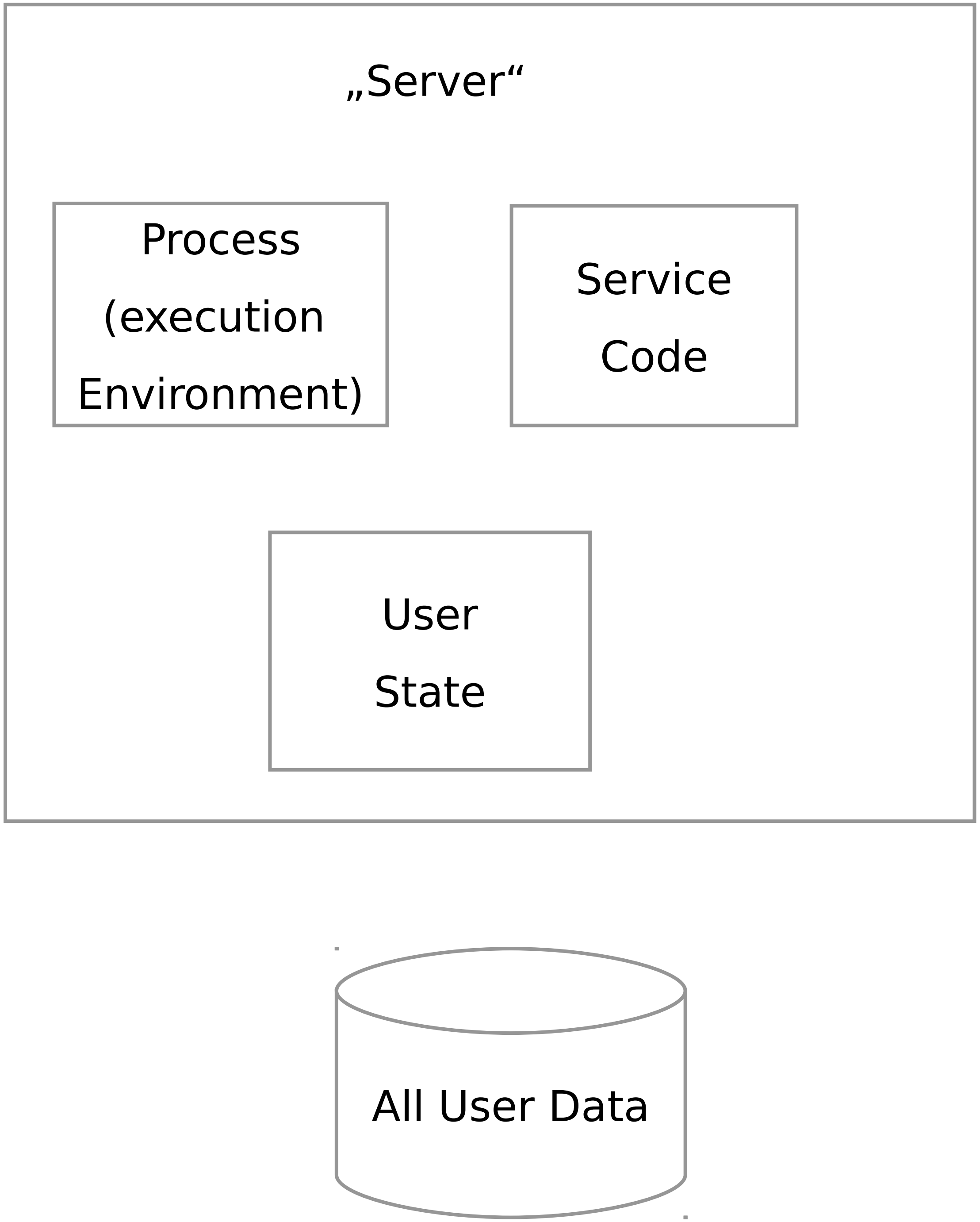
Appl. DB

OS

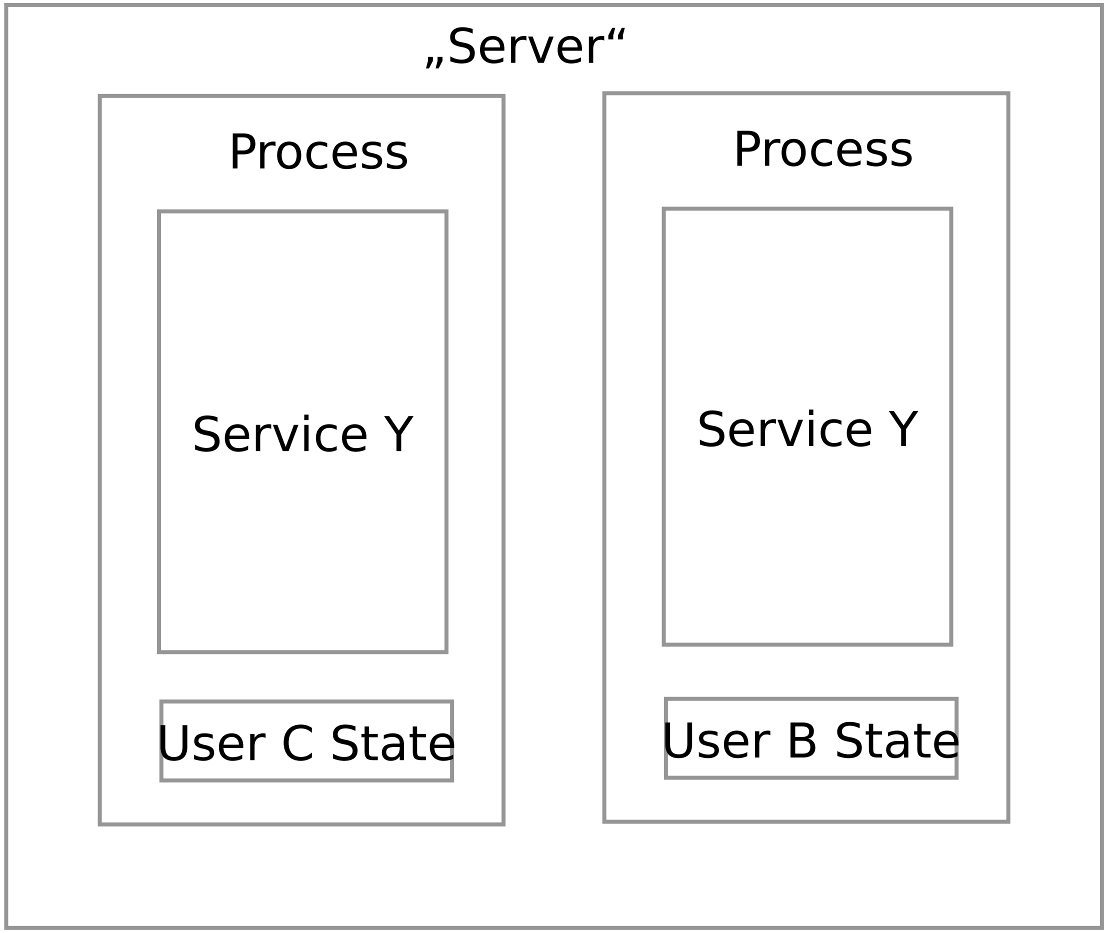
OS Realm

File Resource

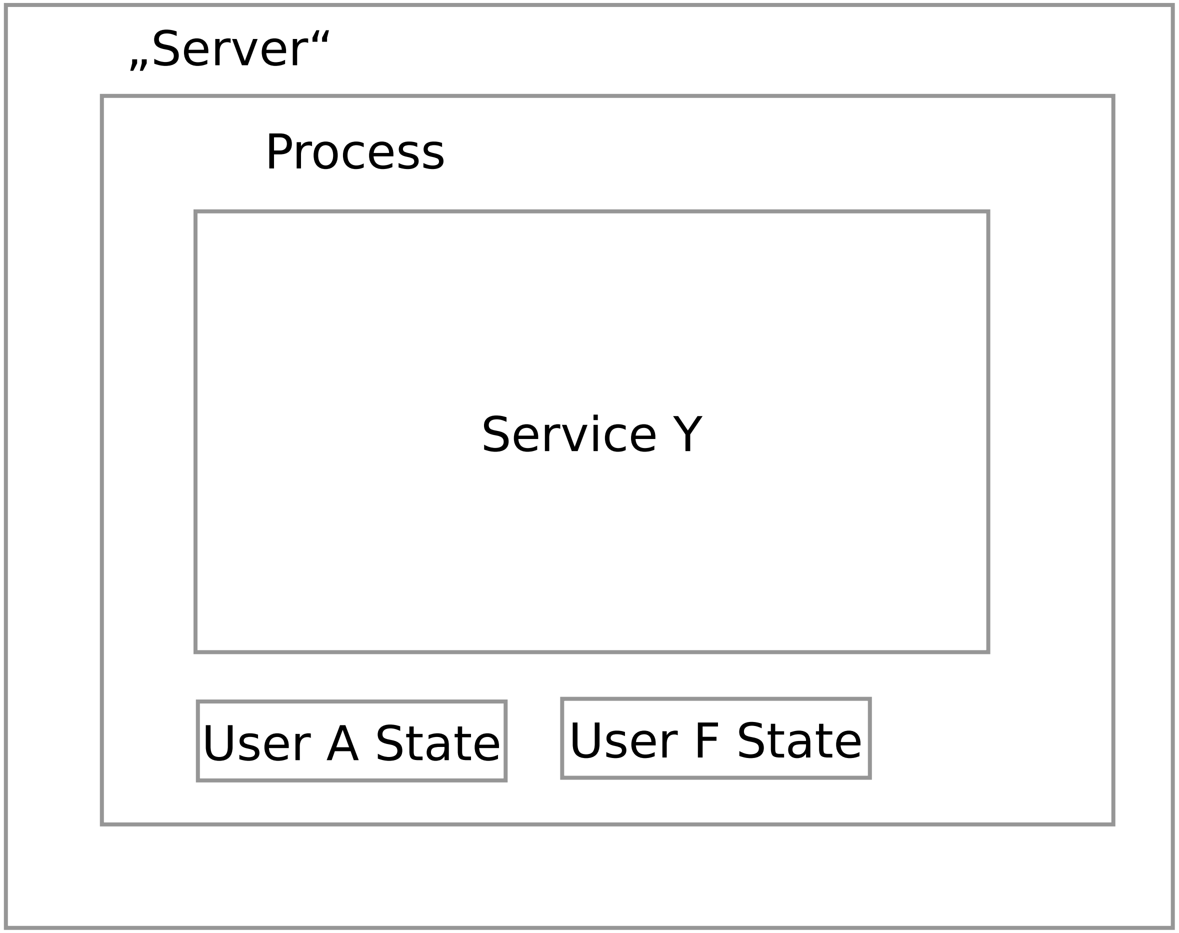
Network Resource



Server components

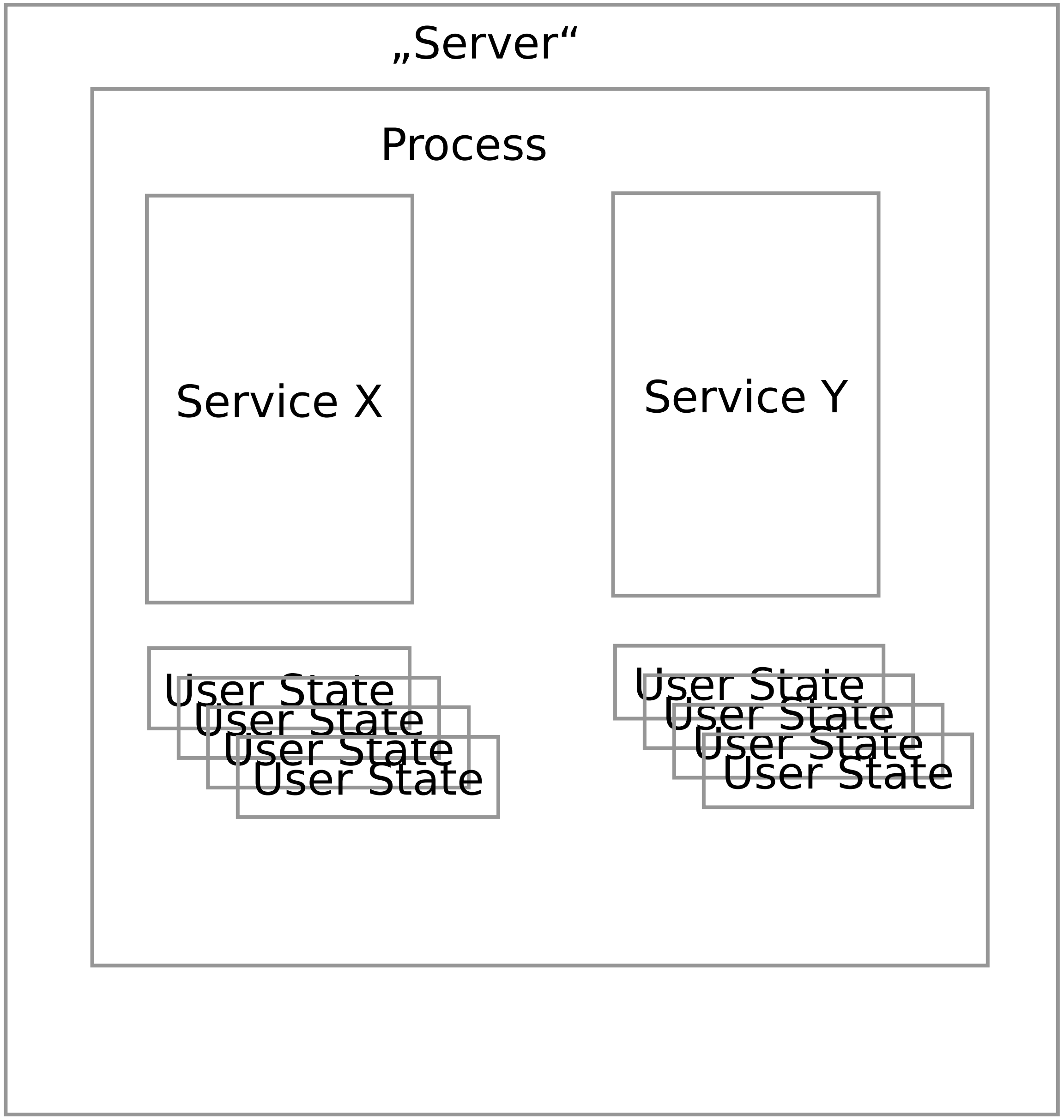


One Process per User- too expensive? (idle users?)



Process is either single-threaded, event-driven or multithreaded.

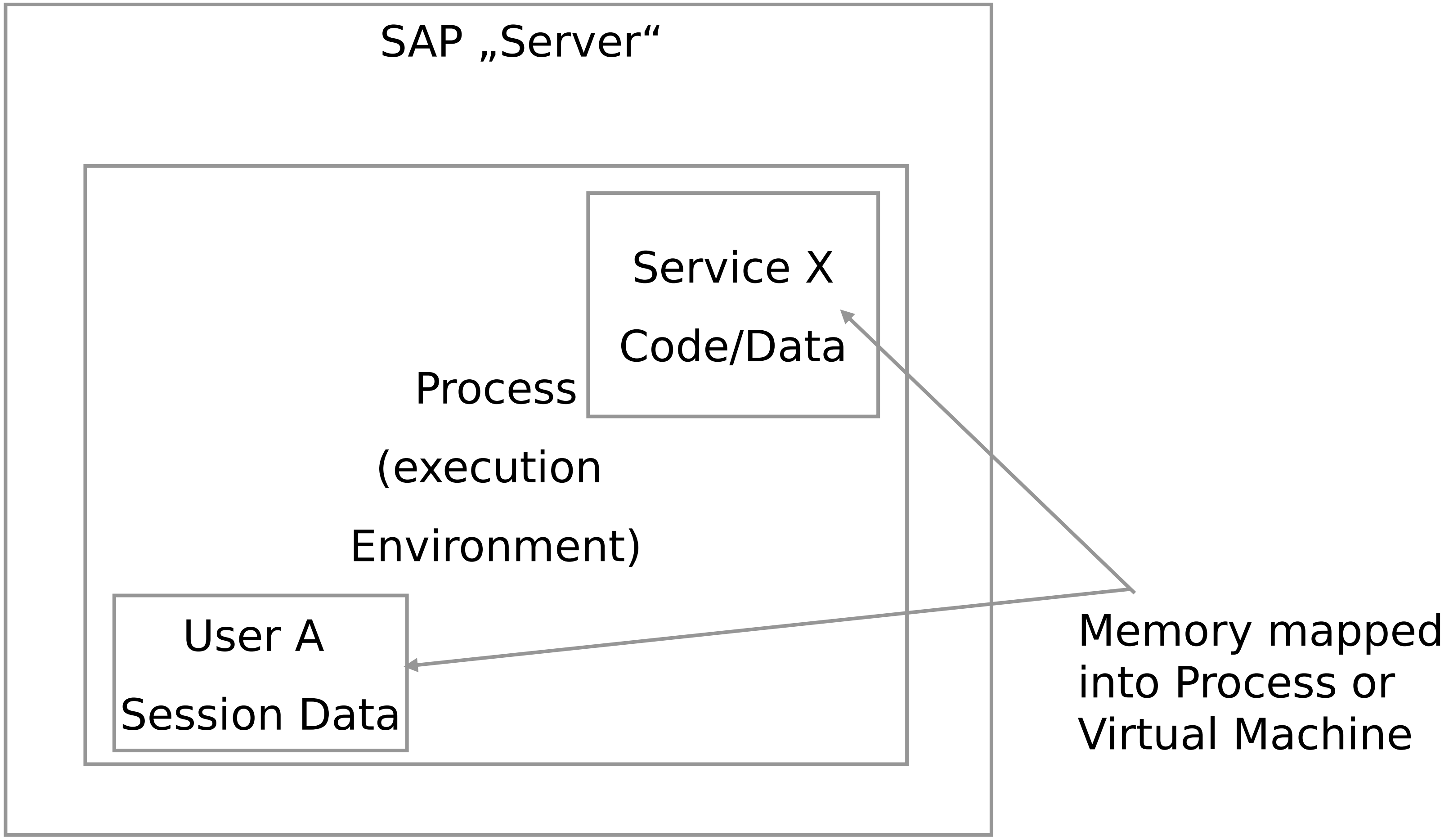
One Process per Service.



One Process

Many Services

Many User
(States)



SAP „Server“

Service X

Code/Data

Process

(execution

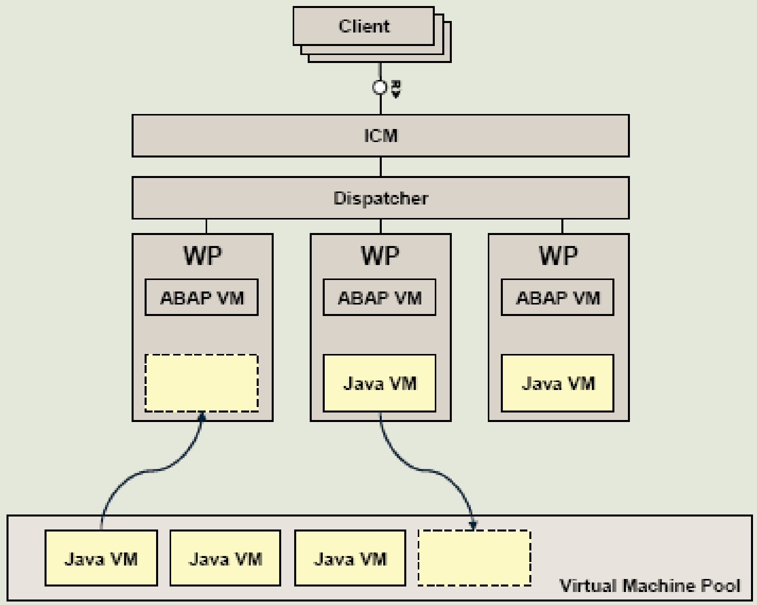
Environment)

User A

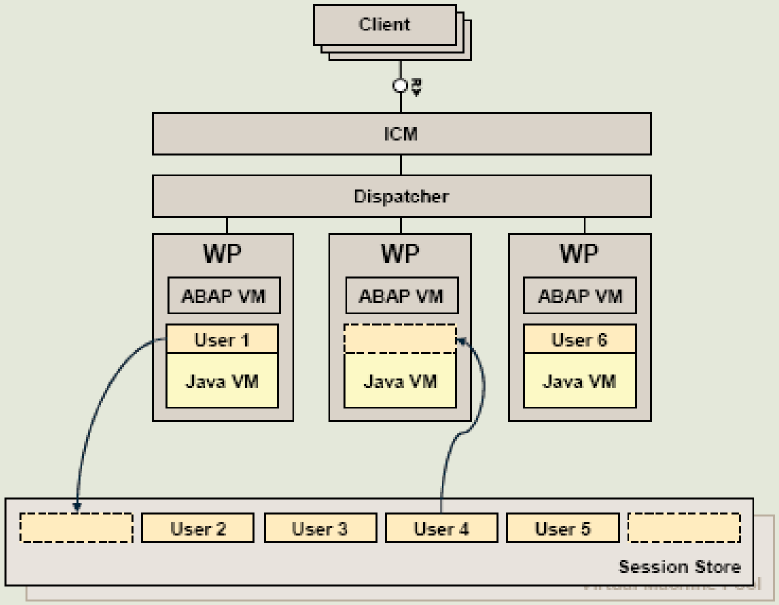
Session Data

Memory mapped
into Process or
Virtual Machine

SAP VM

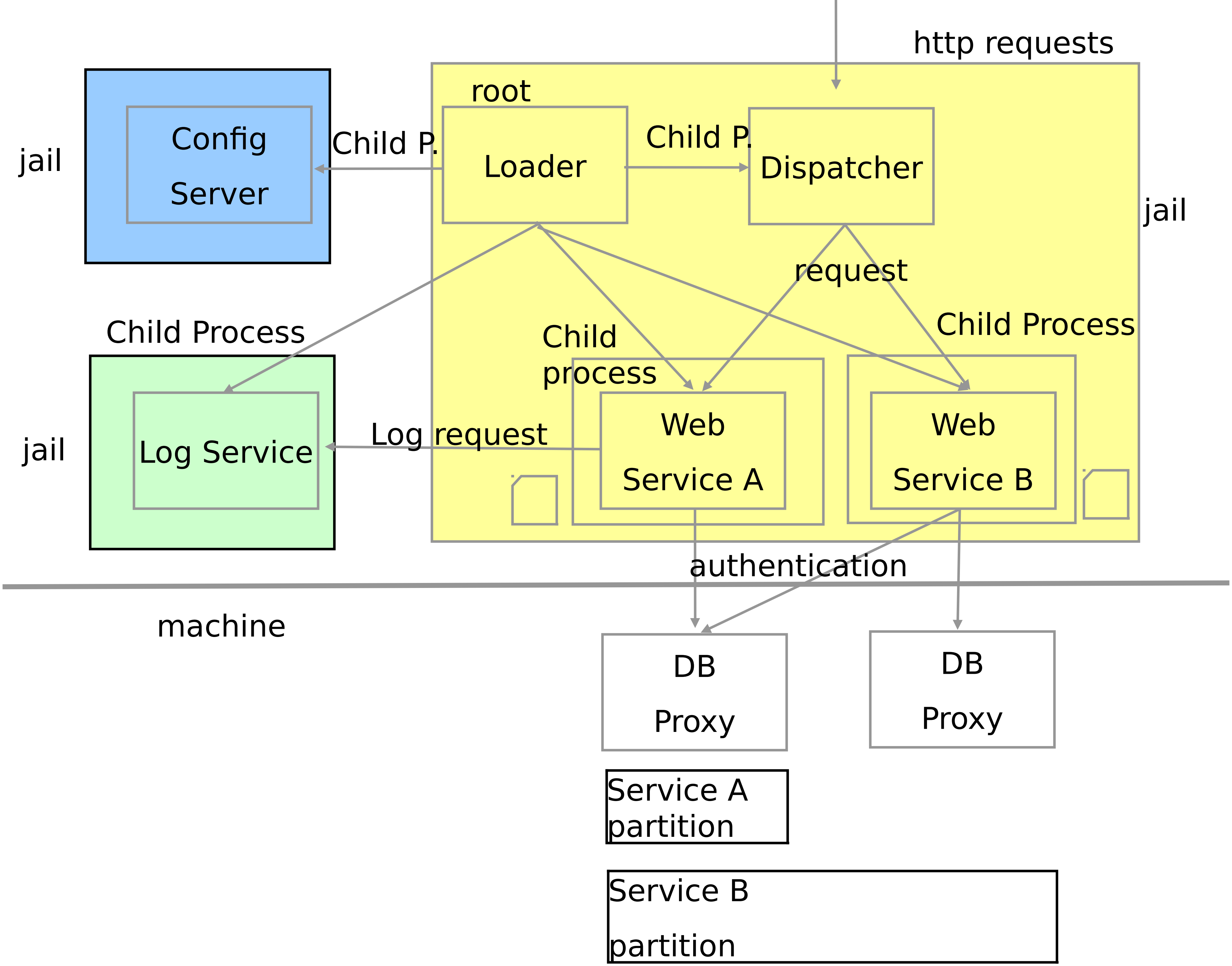


The virtual machine pool contains process attachable VMs that can be assigned to work processes on demand.

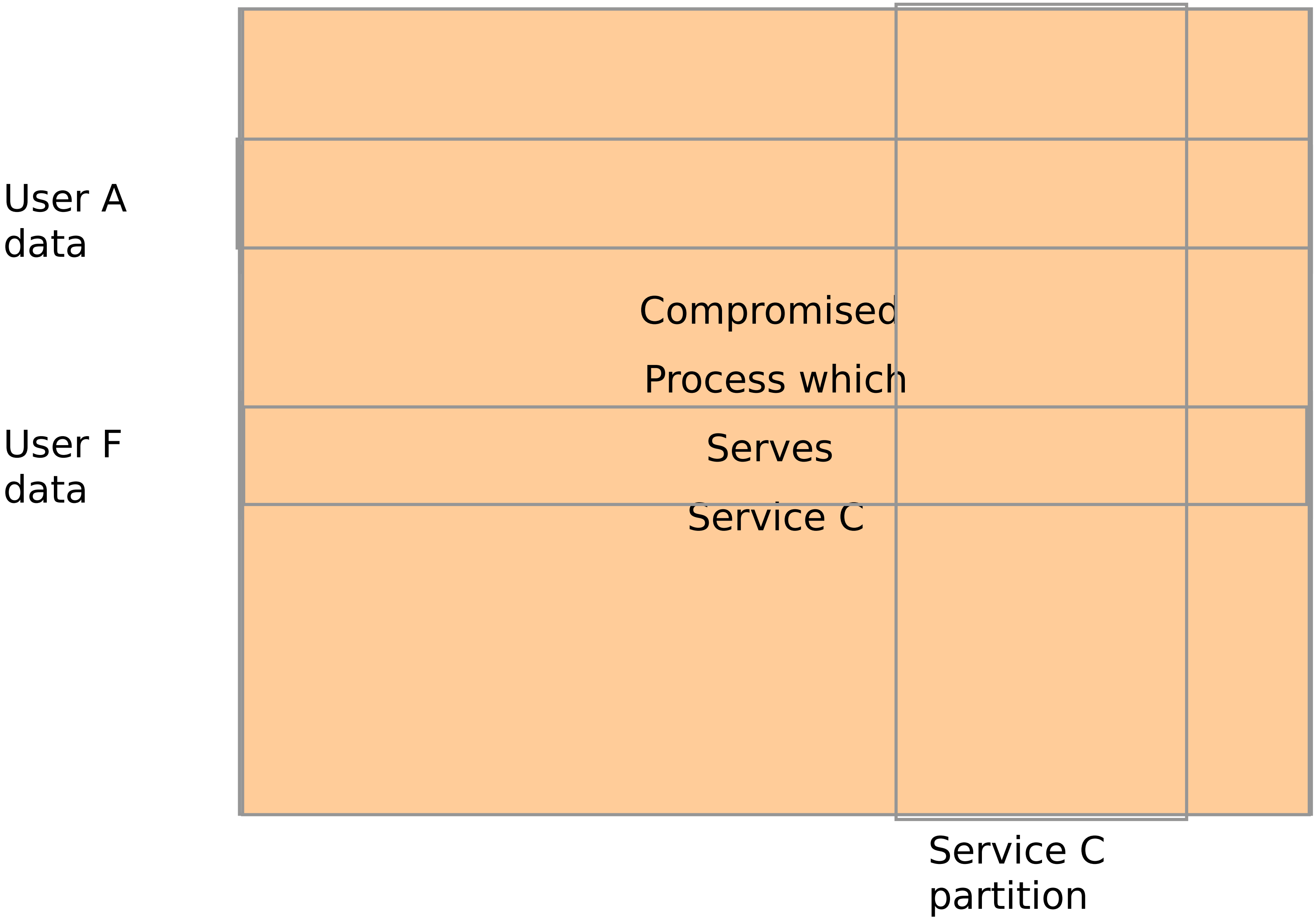


User sessions are kept in shared memory and copied to the virtual machine before the request is processed and copied back after the request was finished..

Authority Restriction with Secure Software Entities


http requests
root
Child P.
Loader
Child P.
Dispatcher
jail
Config
Server
jail
request
Child Process
Child Process
Child
process
Web
Service A
Web
Service B
jail
Log Service
Log request
authentication
machine
DB
Proxy
DB
Proxy
Service A
partition
Service B
partition

Data in Database



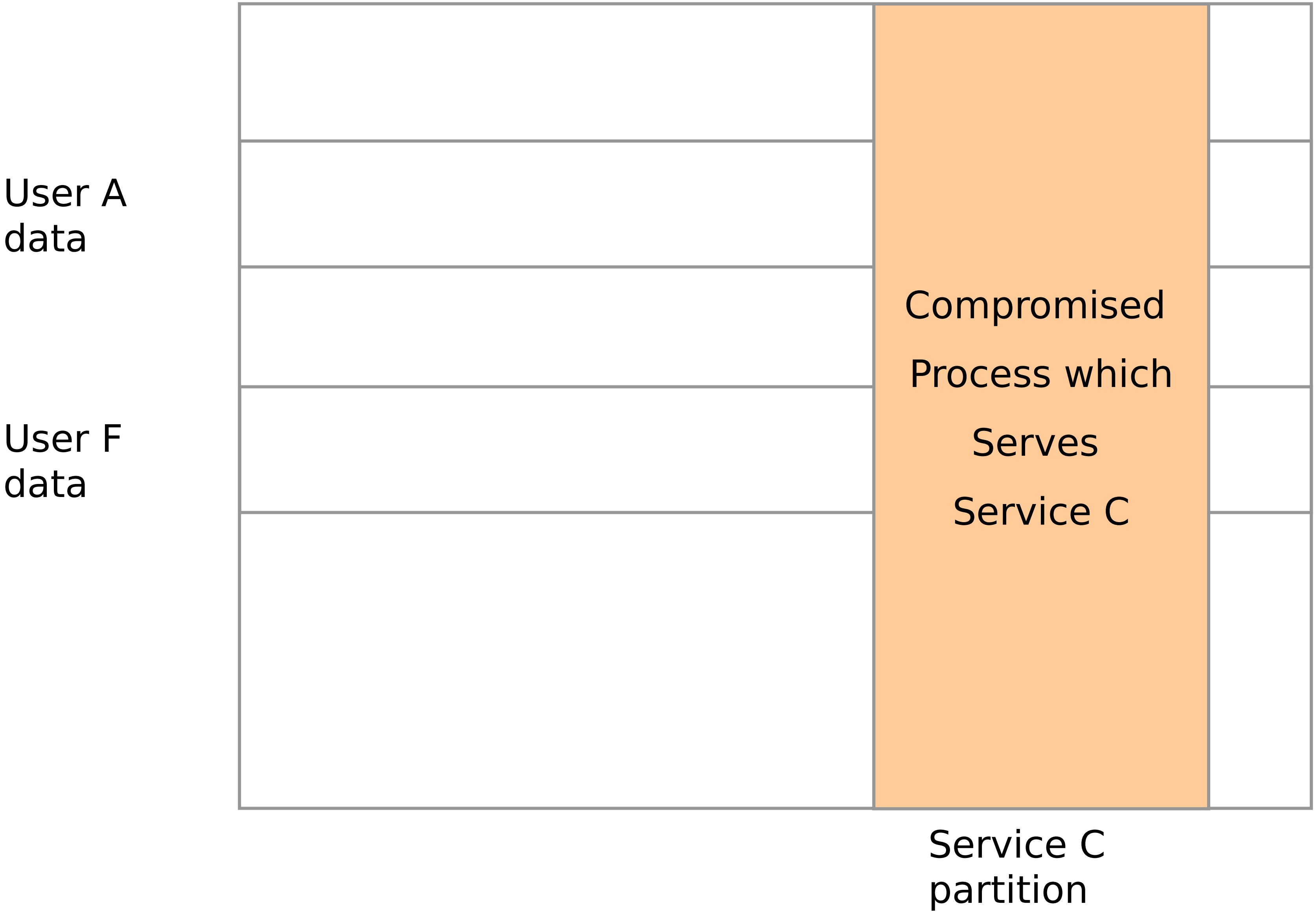
User A
data

User F
data

Compromised
Process which
Serves
Service C

Service C
partition

Data in Database



User A data

User F data

Compromised

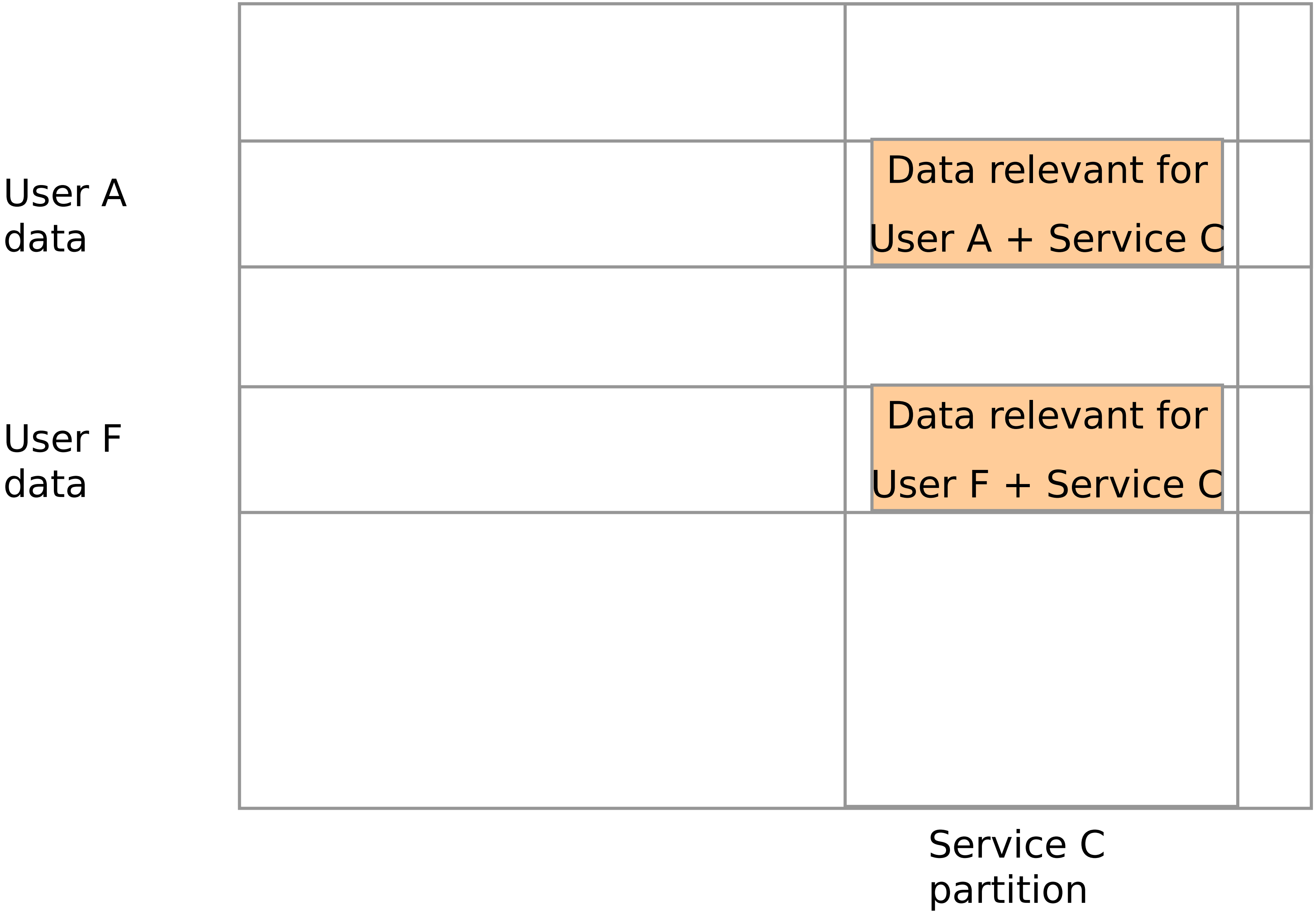
Process which

Serves

Service C

Service C partition

Data in Database



User A data

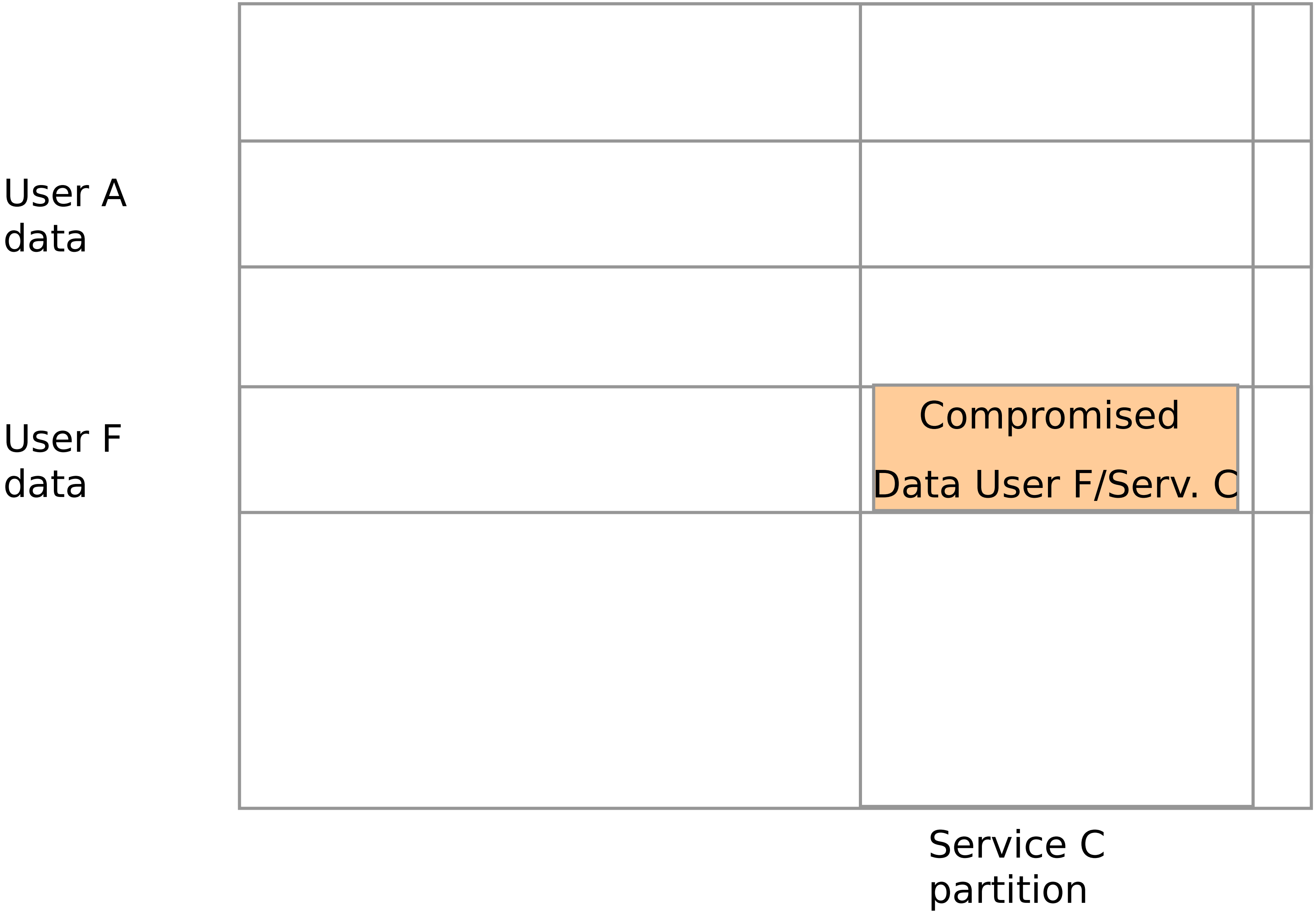
Data relevant for User A + Service C

User F data

Data relevant for User F + Service C

Service C partition

Data in Database



User A data

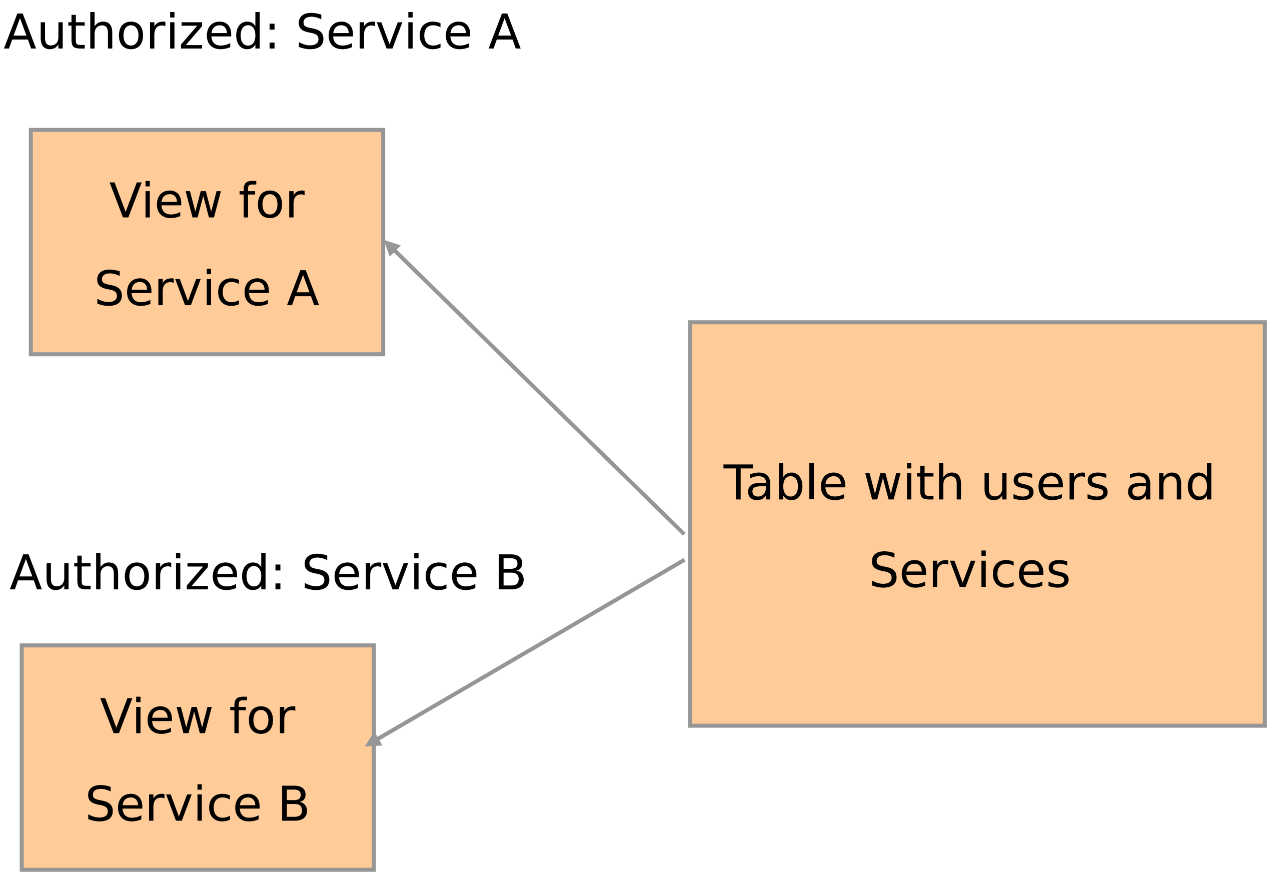
User F data

Compromised

Data User F/Serv. C

Service C partition

Database Views to reduce exposure and use backend access control



Authorized: Service A

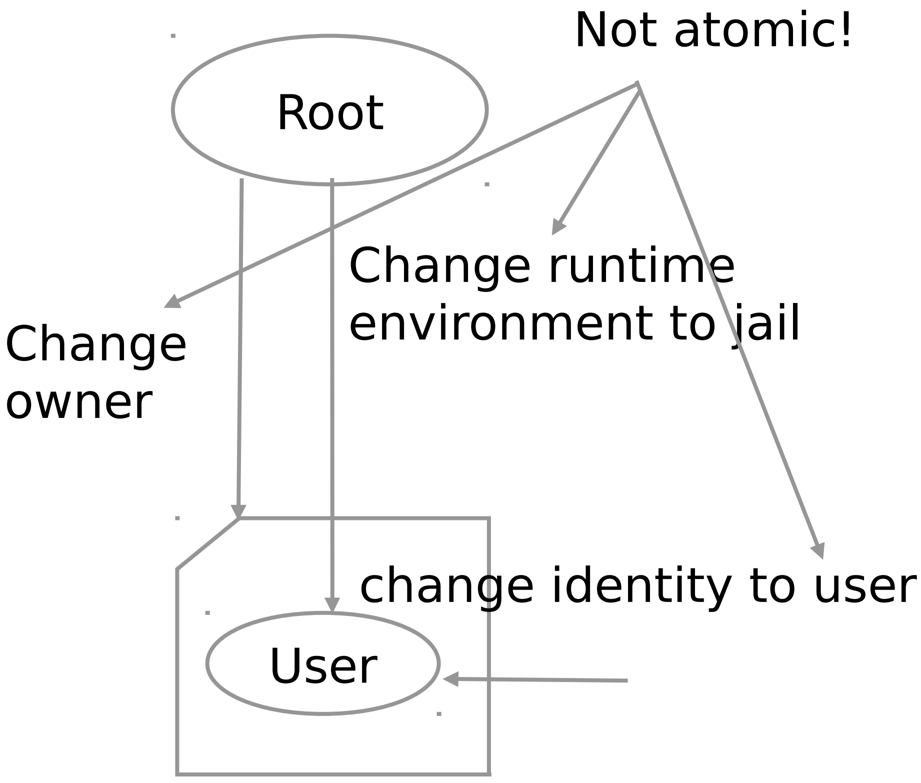
View for Service A

Table with users and Services

Authorized: Service B

View for Service B

Administration and Race Conditions



Not atomic!

Root

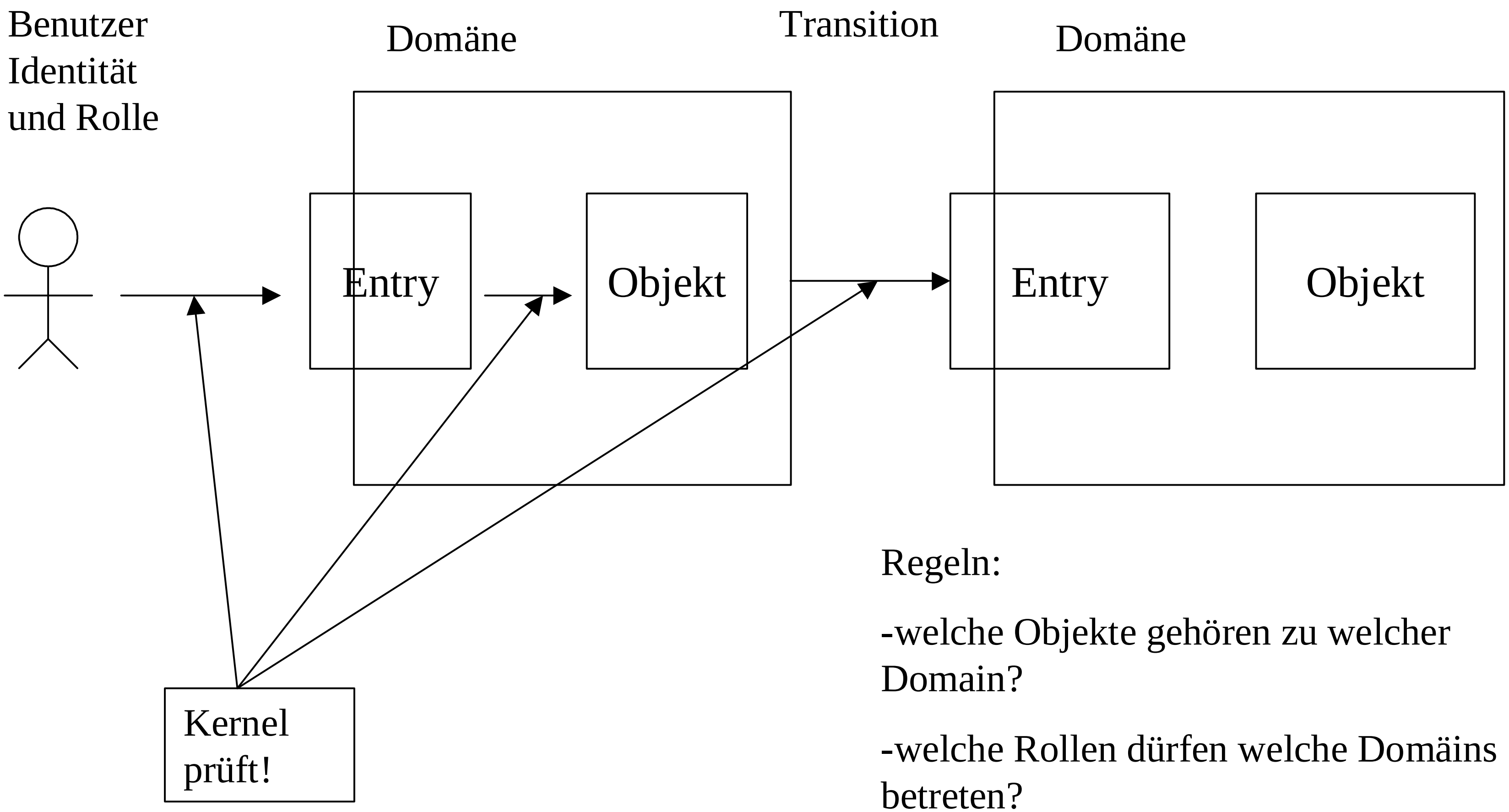
Change runtime environment to jail

Change owner

change identity to user

User

Jails strips off other rights



Benutzer Identität und Rolle

Domäne

Entry

Objekt

Transition

Domäne

Entry

Objekt

Kernel prüft!

Regeln:

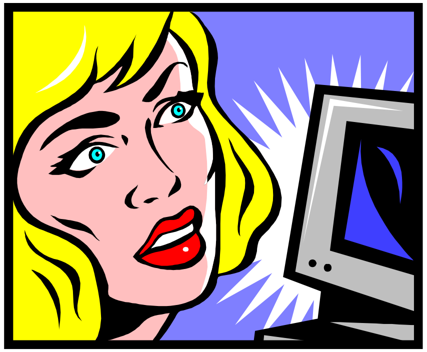
-welche Objekte gehören zu welcher Domain?

-welche Rollen dürfen welche Domäins betreten?

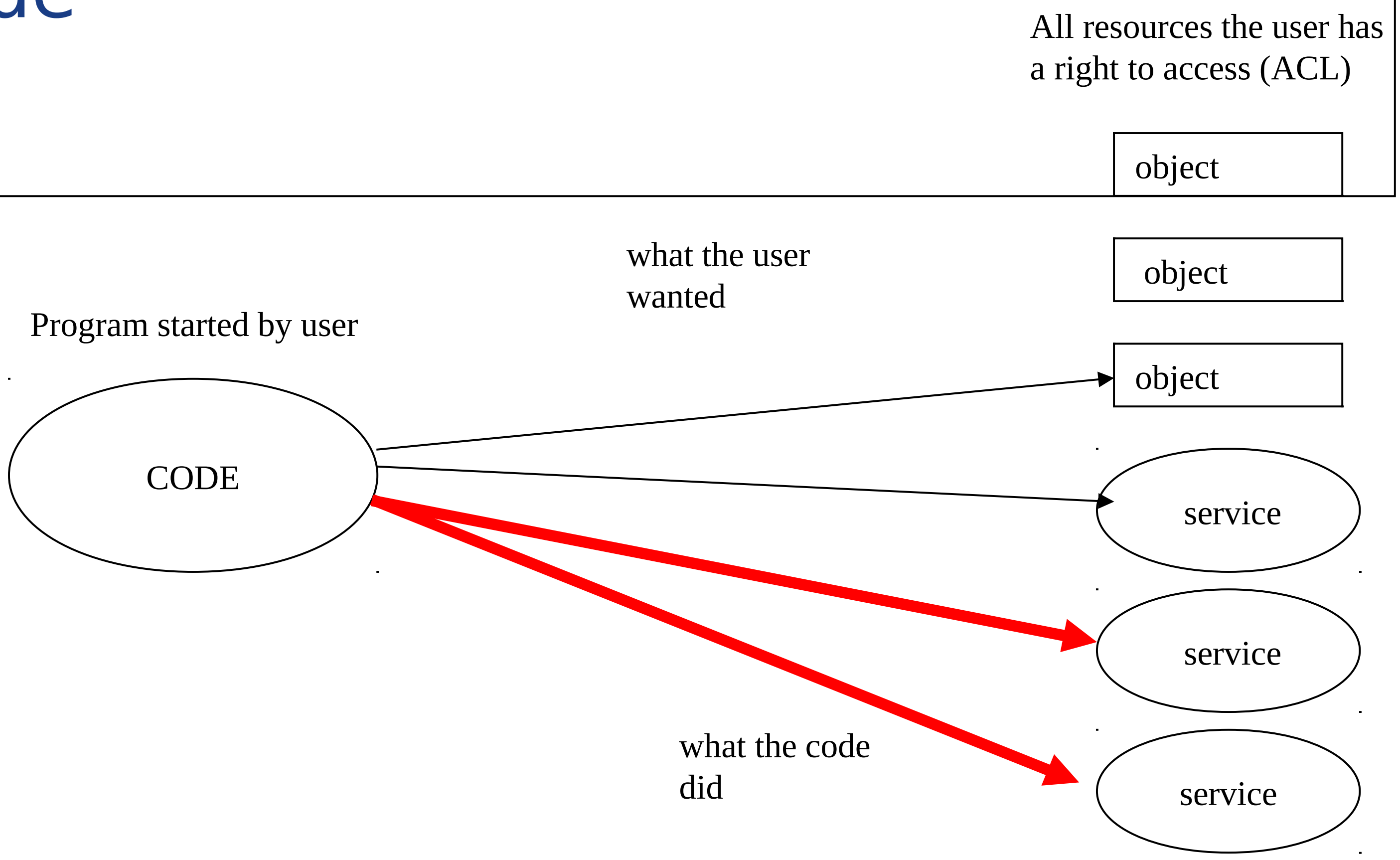
-Zwischen welchen Domains kann ein Übergang (transition) stattfinden? Zwischen welchen Rollen?

-Welchen domains erlauben objekte welche operationen?

Malicious Code



User A



All resources the user has a right to access (ACL)

Program started by user

CODE

what the user wanted

what the code did

object

object

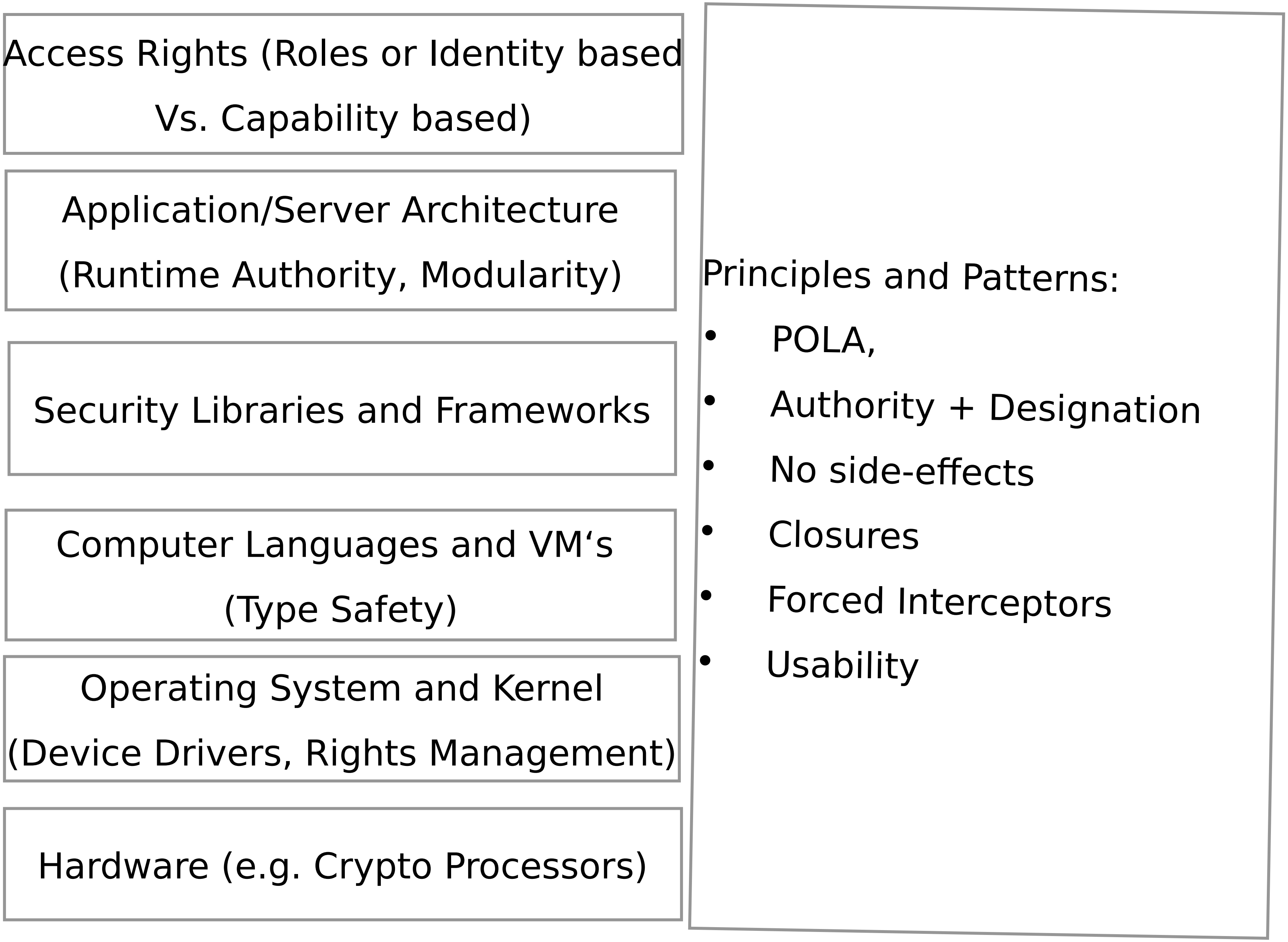
object

service

service

service

All the resources have an attached ACL which says that user A may access the object. This allows malicious code once started by a user to access and abuse all resources the user has access to.



Access Rights (Roles or Identity based Vs. Capability based)

Application/Server Architecture (Runtime Authority, Modularity)

Security Libraries and Frameworks

Computer Languages and VM‘s (Type Safety)

Operating System and Kernel (Device Drivers, Rights Management)

Hardware (e.g. Crypto Processors)

Principles and Patterns:

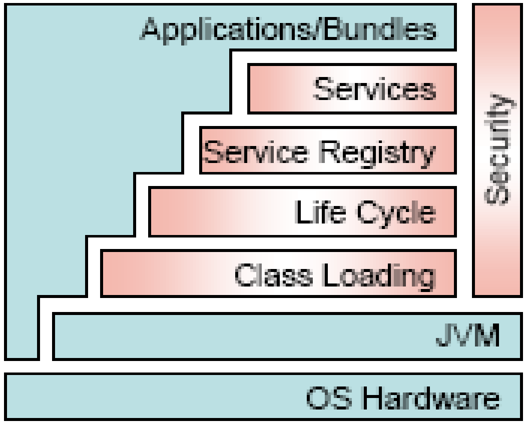
- POLA,
- Authority + Designation
- No side-effects
- Closures
- Forced Interceptors
- Usability

Threats:

- Buffer Overflow Attack on Server Application (external)
- Administrator rights abuse
- Kernel Network stack vulnerability
- Weak keys in application (e.g. SSO)
- Race condition attack on admin scripts
- shatter attack (windows)
- root-kit placement from CD or DVD
- resource hogging application (cycle stealer)
- Input validation problem in application
- Bad device drivers (stability, malicious code)
- Cache coloring

OSGI

Mission Critical Security

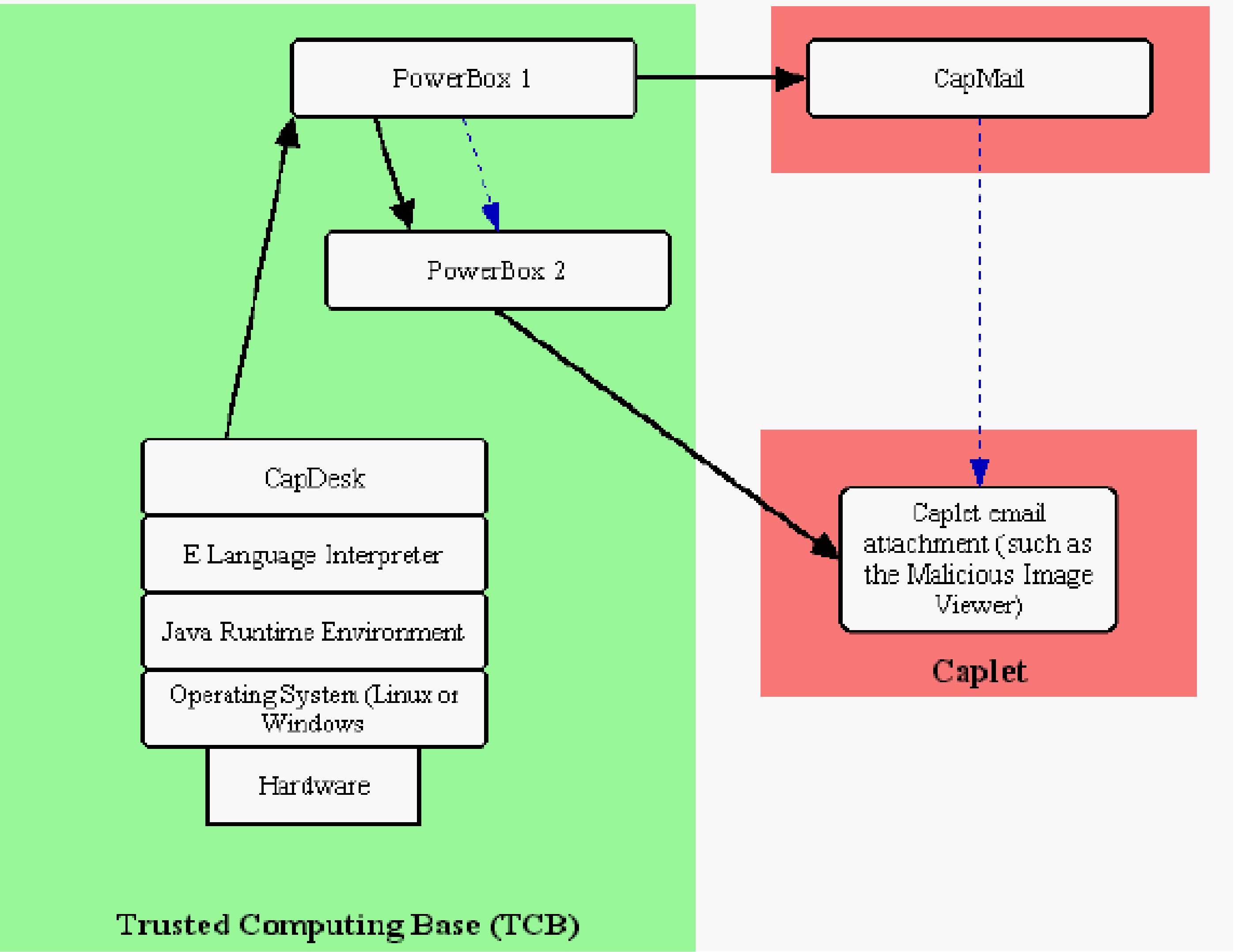


- **Java Code Security (Protection domain = Codebase, Signature, Principal)**
- **Added permissions for administration and service management**
- **Service registry and service interfaces to control export/import and use of services and packages with the help of the class loader**

OSGI is a first step towards a reliable, type-safe platform for mission critical applications in homes, cars and industries. But it is not perfect: class loaders are a terrible way to achieve isolation. Service management is coarse grained and not expressive enough (e.g. if there are two similiar services available I'd like number A). The difficulty is to allow the addition or removal of services ANY TIME.

Objcect Capabilities

Fine-Grained Authority Reduction


PowerBox 1
CapMail
PowerBox 2
CapDesk
E Language Interpreter
Java Runtime Environment
Operating System (Linux or Windows
Hardware
Caplet email attachment (such as the Malicious Image Viewer)
Caplet
Trusted Computing Base (TCB)

Each application or module gets only as much authority as it needs for the job. This allows safe plug-ins like the renderer for a mail type. From the DARPA browser study (www.combex.org)