

CS 489 / 698: Software and Systems Security

Module 3: Operating System Security
access control and capabilities

Meng Xu (*University of Waterloo*)

Spring 2023

Outline

- 1 Introduction to access control
- 2 Implementing the access control matrix
- 3 Models for security policies
- 4 Case study: seL4 microkernel

Why this topic?

Q: Recap: what does an operating system do?

A: Resource sharing — An operating system (OS) allows different “entities” to access different resources in a **shared** way.

- OS makes resources available to entities **if** required by them and **when** permitted by some policy (and availability).
 - What is a resource?
 - What is an entity?
 - How does an entity request for a resource?
 - How does a policy get specified?
 - How is the policy enforced?

Why this topic?

Q: Recap: what does an operating system do?

A: Resource sharing — An operating system (OS) allows different “entities” to access different resources in a **shared** way.

- OS makes resources available to entities **if** required by them and **when** permitted by some policy (and availability).
 - What is a resource?
 - What is an entity?
 - How does an entity request for a resource?
 - How does a policy get specified?
 - How is the policy enforced?

All based on the requirement that:

- an entity can correctly **identify** itself **AND**,
- the OS can correctly **authenticate** the entity.

Goals of access control

In general, access control has three goals:

- **Check on every access:** else the operating system might fail to notice that access rights have been revoked
- **Enforce least privilege:** grant user/program access only to **smallest** number of objects required to perform a task
- **Verify acceptable use:** limit types of activity that can be performed on an object

Access control matrix

- Set of protected objects: O
 - E.g., files or hardware devices
- Set of subjects: S
 - E.g., users, processes acting on behalf of users
- Set of rights: R
 - E.g., read, write, execute, own

Access control matrix

- Set of protected objects: O
 - E.g., files or hardware devices
- Set of subjects: S
 - E.g., users, processes acting on behalf of users
- Set of rights: R
 - E.g., read, write, execute, own
- Access control matrix consists of entries $a[s, o]$, where
 - $s \in S$
 - $o \in O$, and
 - $a[s, o] \subseteq R$

Example access control matrix

	File 1	File 2	File 3
Alice	orw	rx	o
Bob	r	orx	
Carol		rx	

Implementing access control matrix

In practice, access control matrix is rarely implemented as a matrix.

Q: Why?

Implementing access control matrix

In practice, access control matrix is rarely implemented as a matrix.

Q: Why?

A: Too fine-grained, hard to manage (e.g., adding a new subject or object requires the addition of an entire role or column respectively), too sparse \implies waste of space.

Implementing access control matrix

In practice, access control matrix is rarely implemented as a matrix.

Q: Why?

A: Too fine-grained, hard to manage (e.g., adding a new subject or object requires the addition of an entire role or column respectively), too sparse \implies waste of space.

Instead, an access control matrix is typically implemented as

- a set of **access control lists**
 - column-wise representation
- a set of **privilege lists**
 - row-wise representation
- a set of **capabilities**
 - cell-wise representation that encapsulates authentication as well
- or a combination

Outline

- 1 Introduction to access control
- 2 Implementing the access control matrix**
- 3 Models for security policies
- 4 Case study: seL4 microkernel

Access control lists (ACLs)

Each object has a list of subjects and their access rights

Example:

- File 1: {Alice:orw, Bob:r}
- File 2: {Alice:rx, Bob:orx, Carol:rx}
- File 3: {Alice:o}

Access control lists (ACLs)

Each object has a list of subjects and their access rights

Example:

- File 1: {Alice:orw, Bob:r}
- File 2: {Alice:rx, Bob:orx, Carol:rx}
- File 3: {Alice:o}

Implementation on real-world operating systems:

- ACLs are implemented in Windows file system (NTFS), user entry can denote entire user group (e.g., “Students”)
- Classic UNIX file system has a simpler model of ACLs.
 - Each file lists its owner, a group, and a third entry representing all other users.
 - For each class, there is a separate set of rights.
 - Groups are system-wide defined in /etc/group, use `chmod/chown/chgrp` for setting access rights to your files

Access control lists (ACLs)

Q: Which of the following can we do quickly for ACLs?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

Access control lists (ACLs)

Q: Which of the following can we do quickly for ACLs?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

A: Easy, Hard, Easy, Hard, Easy

Privilege lists

Each subject has a list of objects it can access with associated rights

Example:

- Alice: {File 1:orw, File 2:rx, File 3:o}
- Bob: {File 1:r, File 2:orx}
- Carol: {File 2:rx}

Privilege lists

Each subject has a list of objects it can access with associated rights

Example:

- Alice: {File 1:orw, File 2:rx, File 3:o}
- Bob: {File 1:r, File 2:orx}
- Carol: {File 2:rx}

Implementation on real-world operating systems:

- Android / iOS permission framework
- [POSIX capabilities](#) (despite its name...)

Privilege lists

Q: Which of the following can we do quickly for privilege lists?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

Privilege lists

Q: Which of the following can we do quickly for privilege lists?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

A: Hard, Easy, Easy, Easy, Hard

Capabilities

A capability is an **unforgeable token** that gives its owner some access rights to an object.

Example:

- C1: {File 1:w}, C2: {File 2:r}, C3: {File 3: o}, C4: {File 2: x}
- Alice: {C1, C2, C3, C4}, Bob: {C2, C4}, Carol: {C4}

Capabilities

A capability is an **unforgeable token** that gives its owner some access rights to an object.

Example:

- C1: {File 1:w}, C2: {File 2:r}, C3: {File 3: o}, C4: {File 2: x}
- Alice: {C1, C2, C3, C4}, Bob: {C2, C4}, Carol: {C4}

Some properties about capabilities-based system:

- Unforgeability enforced by either
 - a component running at a higher privilege level (e.g., kernel)
 - cryptographic mechanisms (e.g., digital signatures)
- Tokens might be transferable (or non-transferable)
- Tokens might be copyable (or non-copyable)
- Tokens serve both authentication and access control

Capabilities

A capability is an **unforgeable token** that gives its owner some access rights to an object.

Example:

- C1: {File 1:w}, C2: {File 2:r}, C3: {File 3: o}, C4: {File 2: x}
- Alice: {C1, C2, C3, C4}, Bob: {C2, C4}, Carol: {C4}

Some properties about capabilities-based system:

- Unforgeability enforced by either
 - a component running at a higher privilege level (e.g., kernel)
 - cryptographic mechanisms (e.g., digital signatures)
- Tokens might be transferable (or non-transferable)
- Tokens might be copyable (or non-copyable)
- Tokens serve both authentication and access control

Some research/experimental OSs (e.g., Fuchsia, seL4) have fine-grained support for tokens.

Capabilities

Q: Which of the following can we do quickly for capabilities?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

Capabilities

Q: Which of the following can we do quickly for capabilities?

- Determine set of allowed users per object
- Determine set of objects that a user can access
- Revoke a user's access right to an object
- Revoke a user's access right to all objects
- Revoke all users' access rights to an object

A: Hard, Easy, Easy, Easy, Easy

Outline

- 1 Introduction to access control
- 2 Implementing the access control matrix
- 3 Models for security policies**
- 4 Case study: seL4 microkernel

Why do we need security models?

Why do we need security models?

Q: You have implemented the access control matrix (e.g., as ACLs, privilege lists, or capabilities), how can you be certain that the matrix is secure?

Security policies

- Many security policies have their roots in military scenarios
- Each object/subject has a sensitivity/clearance level
 - “Top Secret” $>_c$ “Secret” $>_c$ “Confidential” $>_c$ “Unclassified”
where “ $>_c$ ” means “more sensitive”
- Each object/subject might also be assigned to one or more compartments
 - E.g., “Soviet Union”, “East Germany”
 - **Need-to-know rule**

Security policies

- Many security policies have their roots in military scenarios
- Each object/subject has a sensitivity/clearance level
 - “Top Secret” $>_C$ “Secret” $>_C$ “Confidential” $>_C$ “Unclassified”
where “ $>_C$ ” means “more sensitive”
- Each object/subject might also be assigned to one or more compartments
 - E.g., “Soviet Union”, “East Germany”
 - **Need-to-know rule**
- Subject s can access object o iff $\text{level}(s) \geq \text{level}(o)$ **AND** $\text{compartments}(s) \supseteq \text{compartments}(o)$
 - **s dominates o** , short “ $s \geq_{dom} o$ ”

Example

Q: Secret agent James Bond has clearance “Top Secret” and is assigned to compartment “East Germany”.

Can he read a document with sensitivity level “Secret” and compartments “East Germany” and “Soviet Union”?

Example

Q: Secret agent James Bond has clearance “Top Secret” and is assigned to compartment “East Germany”.

Can he read a document with sensitivity level “Secret” and compartments “East Germany” and “Soviet Union”?

A: No

Lattices

Dominance relationship \geq_{dom} defined in the security model is **transitive** and **antisymmetric**. It defines a **partial** order (neither $a \geq_{dom} b$ nor $b \geq_{dom} a$ might hold for two levels a and b).

Lattices

Dominance relationship \geq_{dom} defined in the security model is **transitive** and **antisymmetric**. It defines a **partial** order (neither $a \geq_{dom} b$ nor $b \geq_{dom} a$ might hold for two levels a and b).

This forms a **lattice**, i.e., for every a and b , there exists a

- **unique lowest upper bound** u for which $u \geq_{dom} a \wedge u \geq_{dom} b$
- **unique greatest lower bound** l for which $a \geq_{dom} l \wedge b \geq_{dom} l$

Transitively, there are also two elements U and L that dominates/is dominated by all levels:

- $U = (\text{"Top Secret"}, \{\text{"Soviet Union"}, \text{"East Germany"}\})$
- $L = (\text{"Unclassified"}, \emptyset)$

Example lattice

Sensitivity levels:

TS = Top Secret

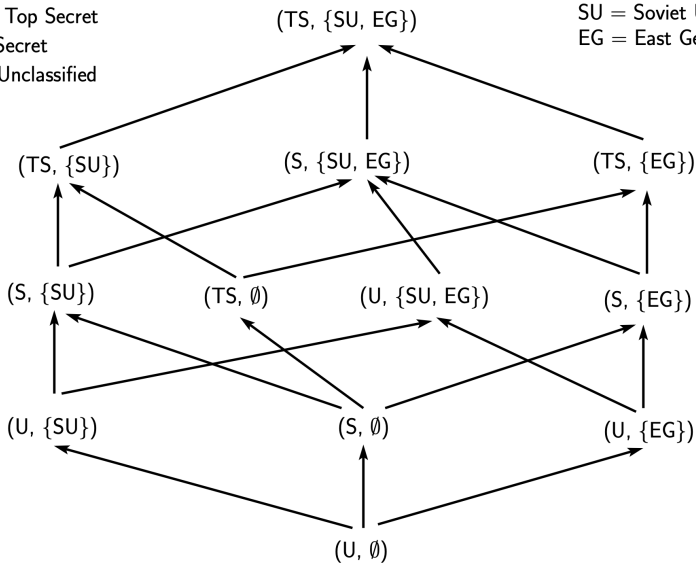
S = Secret

U = Unclassified

Compartments:

SU = Soviet Union

EG = East Germany



The Bell-LaPadula model

Security goal: ensures that information does not flow to those not cleared for that level.

The Bell-LaPadula model

Security goal: ensures that information does not flow to those not cleared for that level.

- The *ss*-property: s can read o iff $C(s) \geq_{dom} C(o)$
- The ***-property: s can write o iff $C(o) \geq_{dom} C(s)$

The Bell-LaPadula model

Security goal: ensures that information does not flow to those not cleared for that level.

- The ss-property: s can read o iff $C(s) \geq_{dom} C(o)$
 - no read-up
- The *-property: s can write o iff $C(o) \geq_{dom} C(s)$
 - no write-down

Q: Why having the “no write-down” policy?

The Bell-LaPadula model

Security goal: ensures that information does not flow to those not cleared for that level.

- The ss-property: s can read o iff $C(s) \geq_{dom} C(o)$
 - no read-up
- The *-property: s can write o iff $C(o) \geq_{dom} C(s)$
 - no write-down

Q: Why having the “no write-down” policy?

A: To prevent someone reading secret document and then summarizing it in an unclassified document

The Bell-LaPadula model

Security goal: ensures that information does not flow to those not cleared for that level.

- The ss-property: s can read o iff $C(s) \geq_{dom} C(o)$
 - no read-up
- The *-property: s can write o iff $C(o) \geq_{dom} C(s)$
 - no write-down

Q: Why having the “no write-down” policy?

A: To prevent someone reading secret document and then summarizing it in an unclassified document

Q: How to transfer information from a high-sensitivity document to a lower-sensitivity document (i.e., declassification)?

The Bell-LaPadula model

Security goal: ensures that information does not flow to those not cleared for that level.

- The ss-property: s can read o iff $C(s) \geq_{dom} C(o)$
 - no read-up
- The *-property: s can write o iff $C(o) \geq_{dom} C(s)$
 - no write-down

Q: Why having the “no write-down” policy?

A: To prevent someone reading secret document and then summarizing it in an unclassified document

Q: How to transfer information from a high-sensitivity document to a lower-sensitivity document (i.e., declassification)?

A: via trusted subjects

Biba integrity model

Security goal: ensures that information cannot be modified by those not cleared for that level.

- Dual of Bell-La Padula model
- Subjects and objects are ordered by an integrity classification scheme, $I(s)$ and $I(o)$
- Should subject s have access to object o ?

Biba integrity model

Security goal: ensures that information cannot be modified by those not cleared for that level.

- Dual of Bell-La Padula model
- Subjects and objects are ordered by an integrity classification scheme, $I(s)$ and $I(o)$
- Should subject s have access to object o ?

- The ss-property: s can read o only iff $I(o) \geq_{dom} I(s)$
 - Unreliable information cannot “contaminate” subject

- The *-property: s can modify o only iff $I(s) \geq_{dom} I(o)$
 - Unreliable subject cannot modify data with high integrity information

Biba integrity model

Security goal: ensures that information cannot be modified by those not cleared for that level.

- Dual of Bell-La Padula model
- Subjects and objects are ordered by an integrity classification scheme, $I(s)$ and $I(o)$
- Should subject s have access to object o ?

- The ss-property: s can read o only iff $I(o) \geq_{dom} I(s)$
 - Unreliable information cannot “contaminate” subject
 - **no read-down**
- The *-property: s can modify o only iff $I(s) \geq_{dom} I(o)$
 - Unreliable subject cannot modify data with high integrity information
 - **no write-up**

Low Watermark Property

- Biba's access rules are very restrictive, a subject cannot ever read lower integrity object
- Can use dynamic integrity levels instead
 - **Subject Low Watermark Property:**
If subject s reads object o , then $I(s) = glb(I(s), I(o))$, where $glb()$ = greatest lower bound
 - **Object Low Watermark Property:**
If subject s modifies object o , then $I(o) = glb(I(s), I(o))$
- Integrity of subject/object can only go down, information flows down

Review of Bell-La Padula & Biba

- Very simple, which makes it possible to even prove correctness properties about them
 - E.g., can prove that if a system starts in a secure state, the system will remain in a secure state

Review of Bell-La Padula & Biba

- Very simple, which makes it possible to even prove correctness properties about them
 - E.g., can prove that if a system starts in a secure state, the system will remain in a secure state
- Probably too simple for great practical benefit
 - Need declassification
 - Need both confidentiality and integrity, not just one
 - What about object creation?

Review of Bell-La Padula & Biba

- Very simple, which makes it possible to even prove correctness properties about them
 - E.g., can prove that if a system starts in a secure state, the system will remain in a secure state
- Probably too simple for great practical benefit
 - Need declassification
 - Need both confidentiality and integrity, not just one
 - What about object creation?
- Information leaks might still be possible through covert channels in an implementation of the model

Chinese Wall security policy

Security goal: dealing with **conflicts of interests** — Once you've decided for a side of the wall, there is no easy way to get to the other side.

Chinese Wall security policy

Security goal: dealing with **conflicts of interests** — Once you've decided for a side of the wall, there is no easy way to get to the other side.

Once you have been able to access information about a particular kind of company, you will no longer be able to access information about other companies of the same kind.

- Useful for consulting, legal, or accounting firms
- Need history of accessed objects
- Access rights change over time
- **ss-property:** Subject s can access object o iff each object previously accessed by s either belongs to the same company as o or belongs to a different kind of company than o does
- ***-property:** For a write access to o by s , we also need to ensure that all objects readable by s either belong to the same company as o or have been sanitized

Example

- Fast Food Companies = {McDonalds, Wendy's}
- Book Stores = {Chapters, Amazon}
- Alice has accessed information about McDonalds
- Bob has accessed information about Wendy's

- ss-property prevents Alice from accessing information about Wendy's, but not about Chapters or Amazon
 - Similar for Bob
- Suppose Alice could write information about McDonalds to Chapters and Bob could read this information from Chapters
 - Indirect information flow violates Chinese Wall Policy
 - *-property forbids this kind of write

Outline

- 1 Introduction to access control
- 2 Implementing the access control matrix
- 3 Models for security policies
- 4 Case study: seL4 microkernel**

What is seL4?

Overview: seL4 is an open source, high-assurance, high-performance operating system **microkernel**.

What is seL4?

Overview: seL4 is an open source, high-assurance, high-performance operating system **microkernel**.

- Available on [GitHub](#) under GPLv2 license
- Contains a comprehensive set of mathematical proofs for correctness and security
- Arguably the fastest microkernel in the world
- Aims to be a piece of software that runs at the heart of any system and controls all accesses to resources

Monolithic kernel vs microkernel

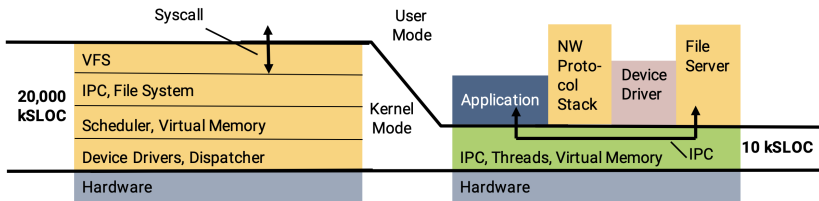
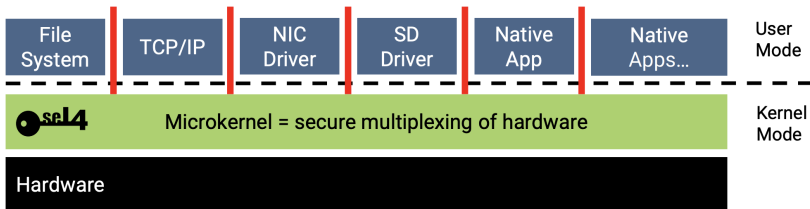


Figure illustrating the difference between

- monolithic kernel (e.g., the Linux kernel) on the left and
- microkernel (e.g., seL4) (on the right)

Adapted from [seL4 Whitepaper](#).

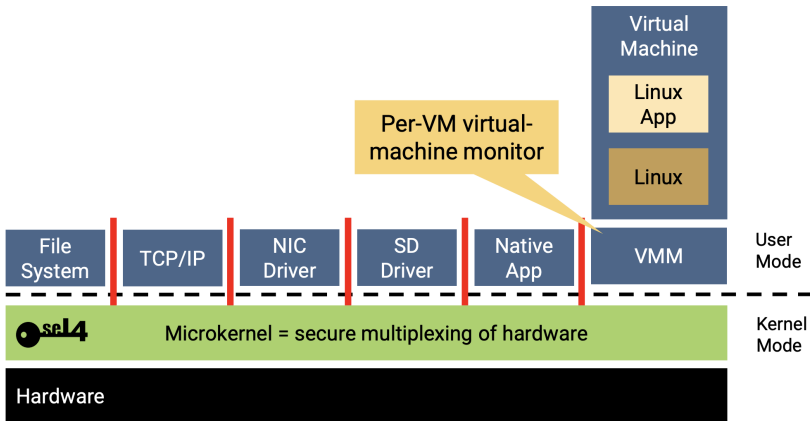
Microkernel



All operating-system services are user-level processes:

- file systems
- device drivers
- network stack
- power management
- ...

Microkernel as hypervisor



Adapted from [seL4 Overview Slides on seL4 Summit 2022](#)

seL4 capability system

General principle: anything goes through seL4 needs a **capability!**

seL4 capability system

General principle: anything goes through seL4 needs a **capability!**



seL4 capability system

General principle: anything goes through seL4 needs a **capability**!



A capability is an object reference that conveys specific rights to a particular object

- Capability = Access Token: **prima-facie evidence of privilege**
- Access rights include read, write, send, reply, execute, . . .
- Kernel object is one of ten object types

seL4 capability system

General principle: anything goes through seL4 needs a **capability**!



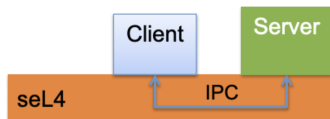
A capability is an object reference that conveys specific rights to a particular object

- Capability = Access Token: **prima-facie evidence of privilege**
- Access rights include read, write, send, reply, execute, . . .
- Kernel object is one of ten object types

Any system call is invoking a capability: `r = cap.method(args);`

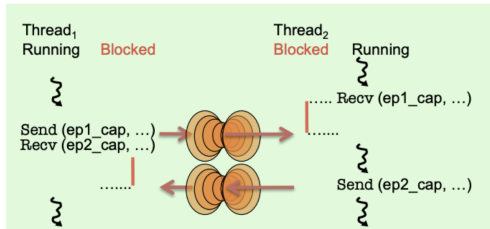
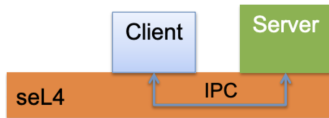
seL4 protected procedure calls (IPC)

Protected procedure call
(IPC for historical reasons)
is a fundamental operation
in seL4.



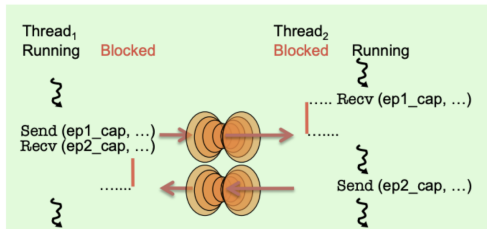
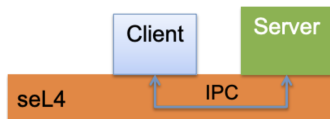
seL4 protected procedure calls (IPC)

Protected procedure call
(IPC for historical reasons)
is a fundamental operation
in seL4.



seL4 protected procedure calls (IPC)

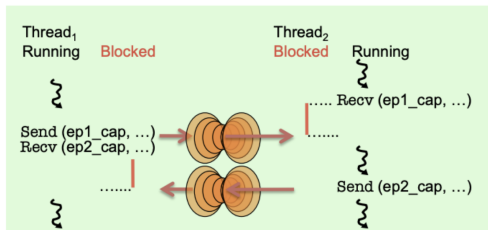
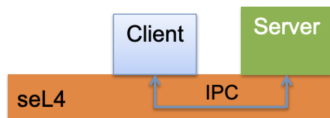
Protected procedure call
(IPC for historical reasons)
is a fundamental operation
in seL4.



Q: How would a normal open syscall be like in seL4?

seL4 protected procedure calls (IPC)

Protected procedure call
(IPC for historical reasons)
is a fundamental operation
in seL4.



Q: How would a normal open syscall be like in seL4?

A: `Call(ext4fs_endpoint_cap, OPEN_FILE, <extra-args>)`

- `Mint reply_cap`
- `Send(ext4fs_endpoint_cap, reply_cap, ...)`
- `Recv(reply_cap, ...)`

seL4 kernel objects

- **Endpoints** are used to perform protected function calls
- **Reply Objects** represent a return path from a protected procedure call
- **Address Spaces** provide the sandboxes around components (thin wrappers abstracting hardware page tables)
- **Cnodes** store capabilities representing a component's access rights
- **Thread Control Blocks** represent threads of execution
- **Scheduling Contexts** represent the right to access a certain fraction of execution time on a core
- **Notifications** are synchronisation objects (similar to semaphores)
- **Frames** represent physical memory that can be mapped into address spaces
- **Interrupt Objects** provide access to interrupt handling
- **Untyped** unused (free) physical memory that can be converted ("retyped") into any of the other types.

⟨ End ⟩