CS 453/698: Software and Systems Security

Module: Operating Systems Security

Lecture: Access Control Policies & Architectures

Adam Caulfield

University of Waterloo

Spring 2025

Reminders & Recap

Reminders:

- A3 is released

Recap – last time we covered:

- Secure boot
  - HW & SW roots of trust
- Inter process isolation
- Virtualization methods
- Compartmentalization
  - seccomp

Today

Access Control

Policies & Modeling

- Access Matrix (HRU Model)

PoC Architectures

- ACES

Access Control

System security mechanisms often implement some form of access control

Definition: *Access Control is the action of deciding whether a subject should be granted or denied access to an object; the act of accessing may mean consuming, setting, or using.*

Terms:

- **Subject:** entity that is requesting access of some resources
- **Object:** the resource itself

Implemented across systems at different levels & granularities:

- OS-based memory management
- Compartmentalization

Access Control

Components required for an access control system

Security Policy

- Defines the high-level rules according to which access control must be regulated

Security Model

- Provides a formal representation of the access control security policy
- Allows for proof of properties

Security Mechanism

- The low-level functions that implement the controls imposed by the policy stated by the formal model

Access Control Policies

Discretionary access

- Access is identity- & authorization-based
- Identify of the subject is considered for defining policy and enforcement

Mandatory access

- Central authority assigns security level of objects
- Subjects are assigned access levels

Role-based access

- Depend on a subject’s roles within a system
- Define roles, access for each role, then assign roles

Access Control Policies

Discretionary access: access is assigned per-subject

Example: file system permissions

Subjects: set of users

Objects: set of policies

Specify read, write, or execute permission for files by identify

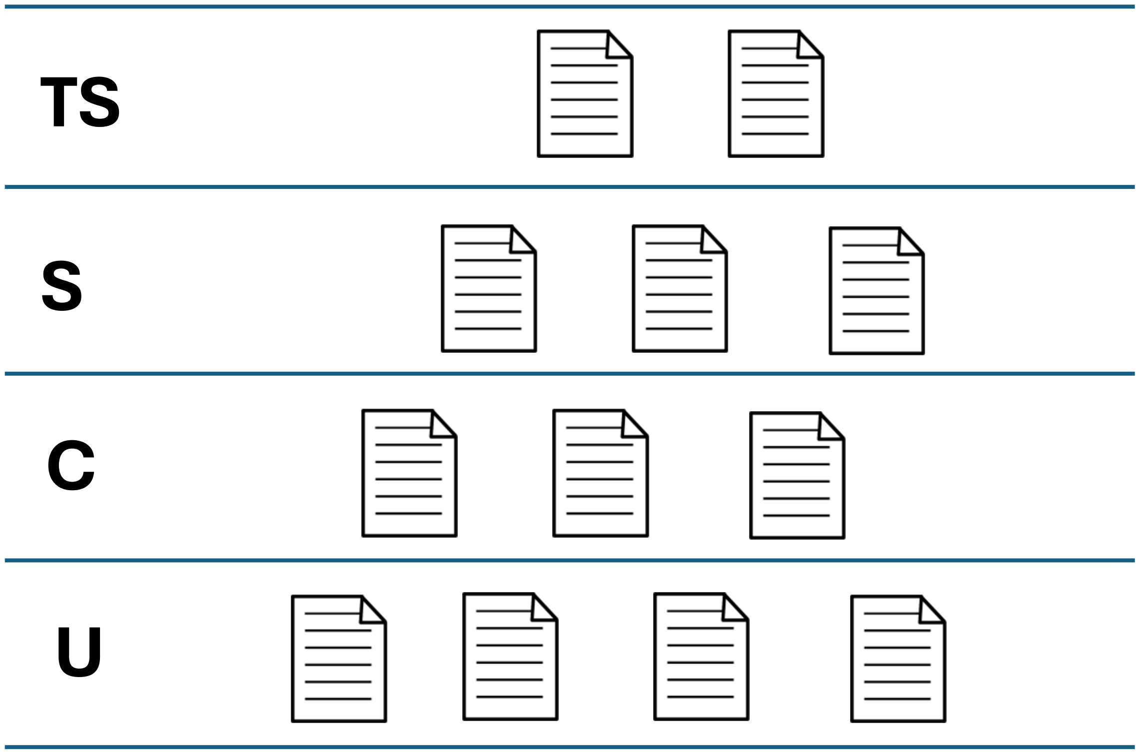
	file1	file2	file3
Alice	- - -	r – x	r - -
Bob	r - -	r w x	- - -
Carol	- - -	r – x	- - -

Access Control Policies

Mandatory access: a security level is assigned to each object based on its sensitivity in the system. Then subjects are assigned access level or *clearance*

Example: Government/military clearance:

- Top secret (TS), Secret (S), Confidential (C), Unclassified (U)



Access Control Policies

Mandatory access: a security level is assigned to each object based on its sensitivity in the system. Then subjects are assigned access level or *clearance*

Example: Government/military clearance:

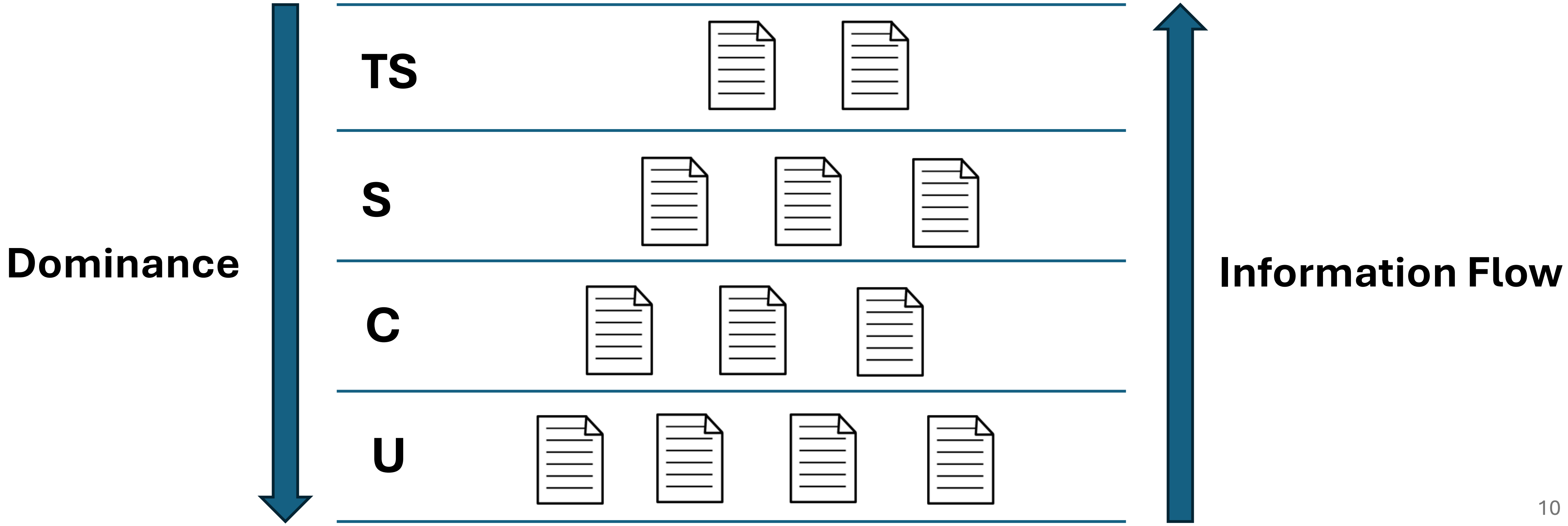
- Top secret (TS), Secret (S), Confidential (C), Unclassified (U)

Access Control Policies

Mandatory access: a security level is assigned to each object based on its sensitivity in the system. Then subjects are assigned access level or *clearance*

Example: Government/military clearance:

- Top secret (TS), Secret (S), Confidential (C), Unclassified (U)



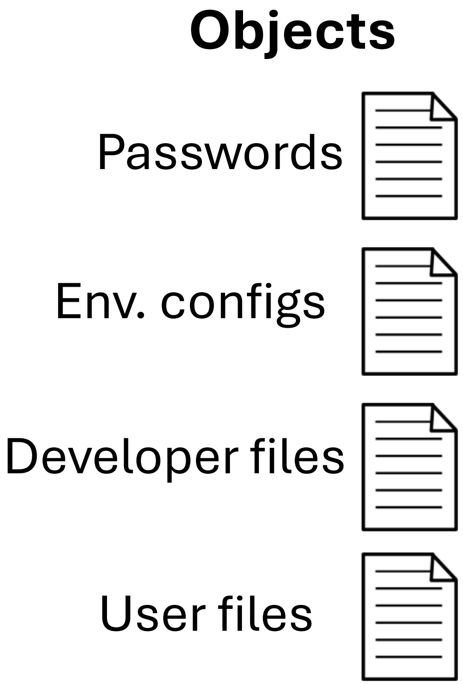
Access Control Policies

Role-based access: defined based on role in a system

- Tailored towards commercial applications
- Grouping privileges

Example: Named-protection domain (NPD) privilege graph

- Domains have unique names,



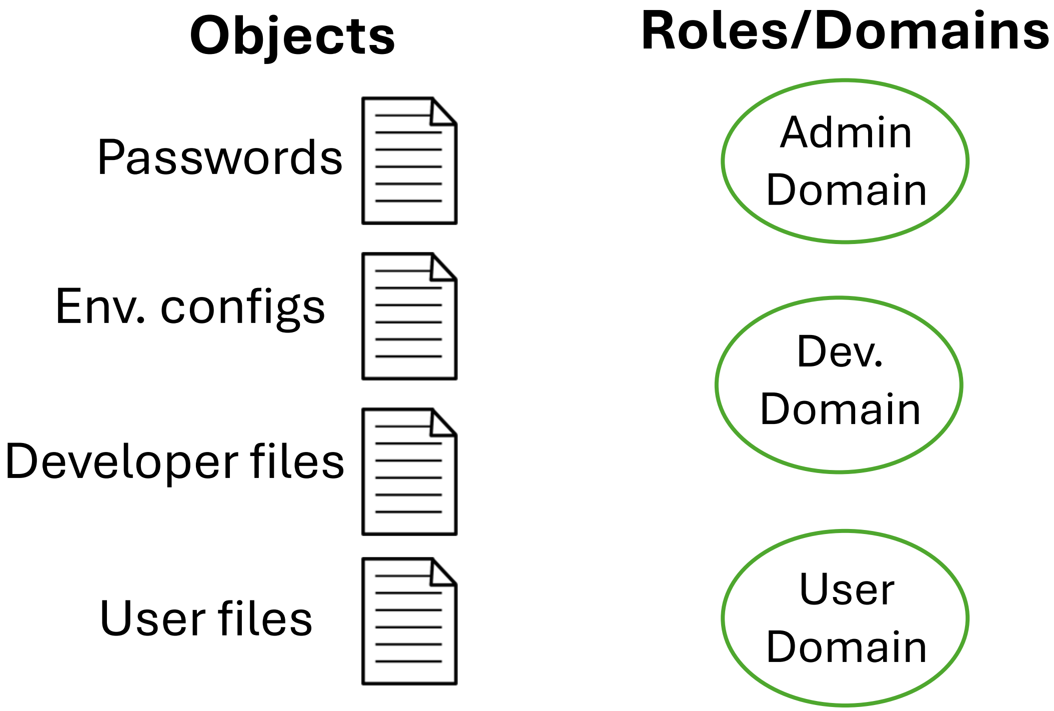
Access Control Policies

Role-based access: defined based on role in a system

- Tailored towards commercial applications
- Grouping privileges

Example: Named-protection domain (NPD) privilege graph

- Domains have unique names,



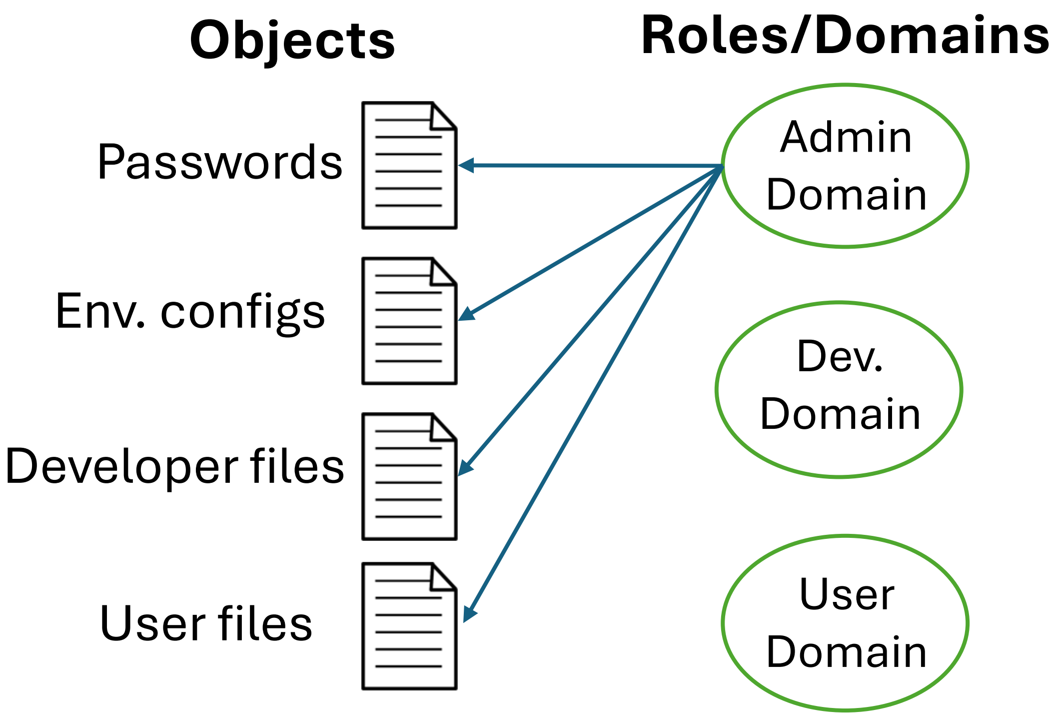
Access Control Policies

Role-based access: defined based on role in a system

- Tailored towards commercial applications
- Grouping privileges

Example: Named-protection domain (NPD) privilege graph

- Domains have unique names,



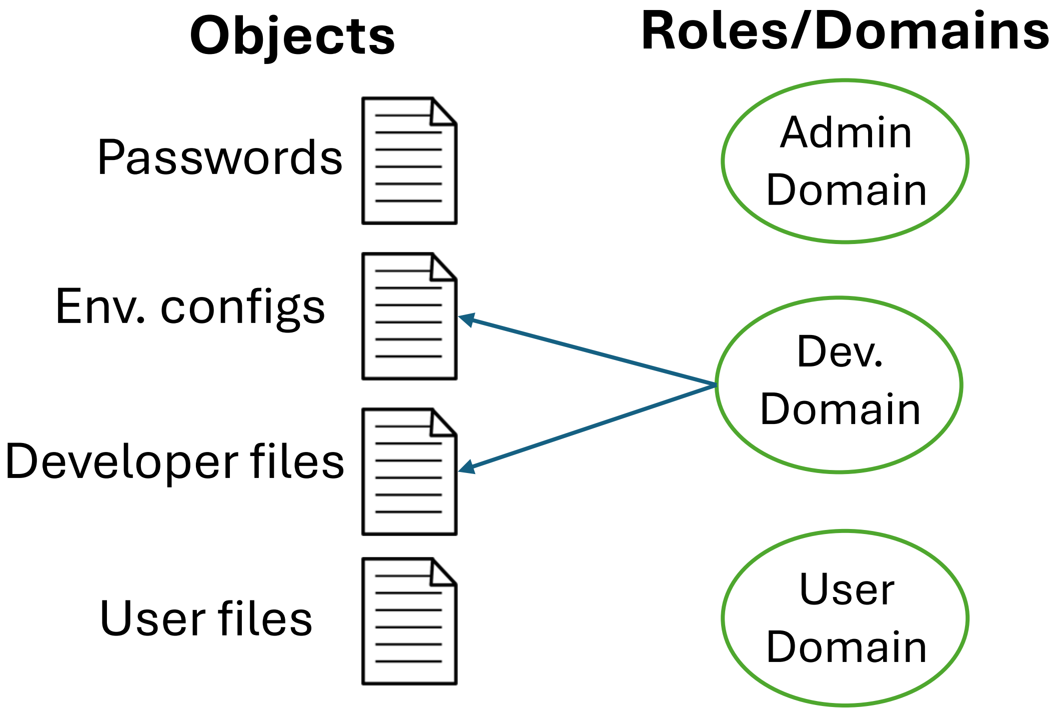
Access Control Policies

Role-based access: defined based on role in a system

- Tailored towards commercial applications
- Grouping privileges

Example: Named-protection domain (NPD) privilege graph

- Domains have unique names,



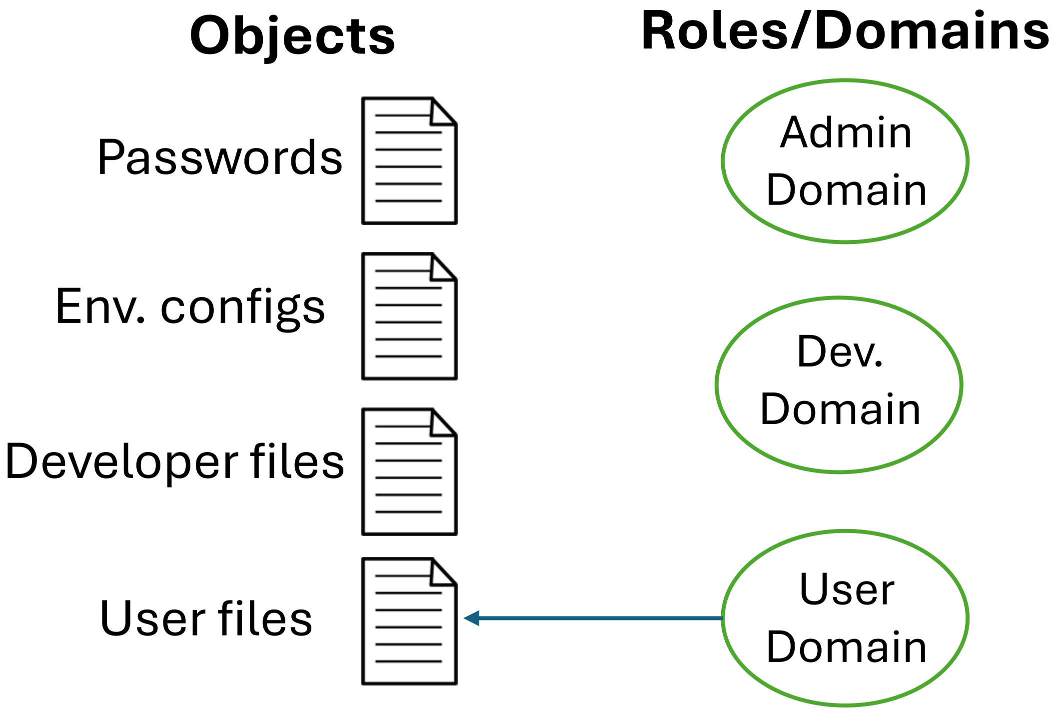
Access Control Policies

Role-based access: defined based on role in a system

- Tailored towards commercial applications
- Grouping privileges

Example: Named-protection domain (NPD) privilege graph

- Domains have unique names,



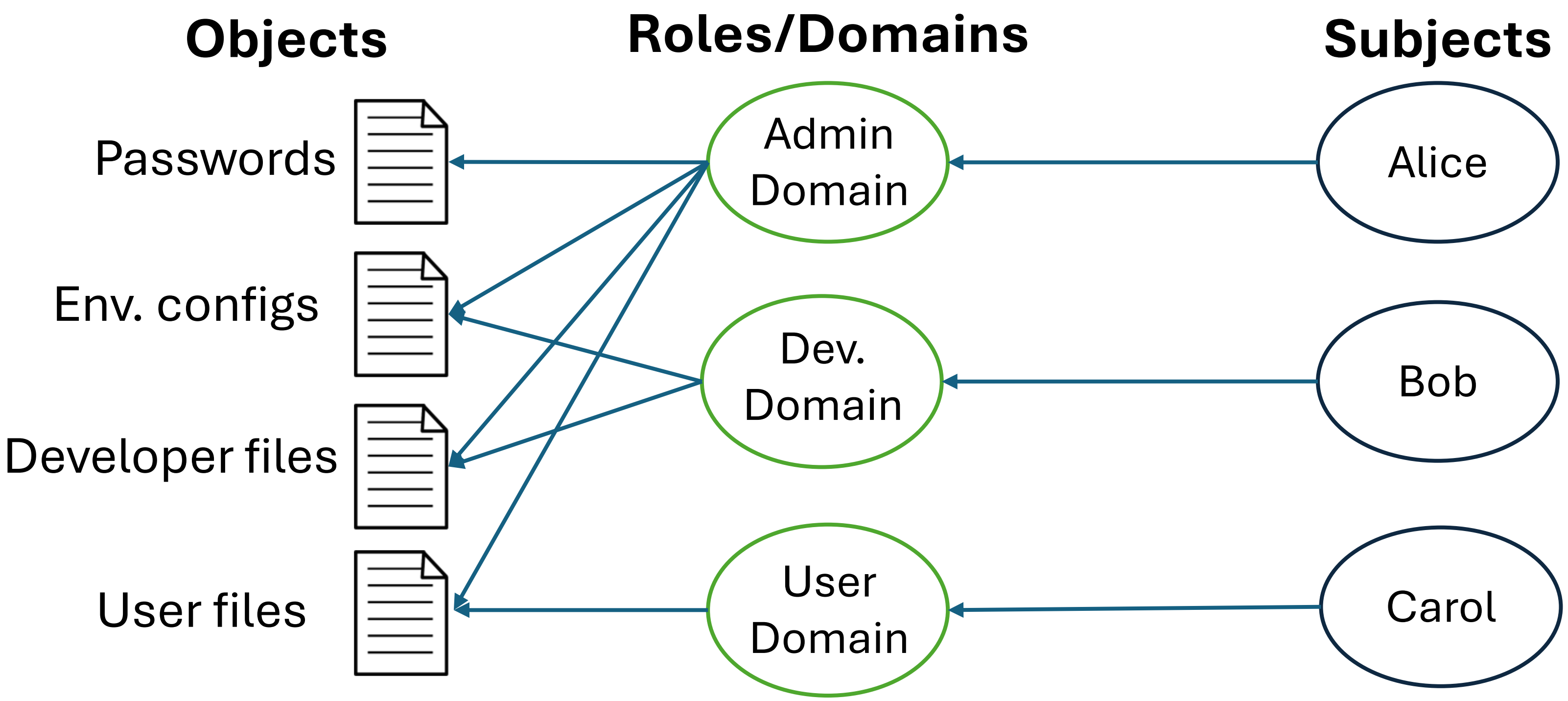
Access Control Policies

Role-based access: defined based on role in a system

- Tailored towards commercial applications
- Grouping privileges

Example: Named-protection domain (NPD) privilege graph

- Domains have unique names,



Access Control Models

Access Matrix: Harrison, Ruzzo, and Ullmann (HRU) Model

Let's revisit this simple file permissions table

	file1	file2	file3
Alice	- - -	r – x	r - -
Bob	r - -	r w x	- - -
Carol	- - -	r – x	- - -

Questions:

- Using this matrix model, how can we define the state of the system?
- How can matrix operations be formalized?

Access Control Models

Access Matrix: Defining state

Definitions:

- Set of subjects (*S*) are entities that request access of a resource
  - Rows in the matrix
  - Subjects can be objects
- Set of objects (*O*) are entries available for access (in adherence to the policy)
  - Columns in the matrix
- Access matrix (*A*) defines the access policy between *S*-*O*
- *A[s,o]* defines *actions* in *A* for subject *s* on object *o*
  - Example: A[Alice,file1] = r+x

System State: (*S*, *O*, *A*)

- Changes to state are carried out through primitive operations

Access Control Models

Primitive operations in HRU model

- Enter action into A[s,o]
- Delete action from A[s,o]
- Create subject s’
- Create object o’
- Destroy subject s’
- Destroy object o’

Each operation has:

- A condition that is required for its execution
- Outputs a new state
  - S’, O’, A’

Access Control Models

OPERATION (op)	CONDITIONS	NEW STATE ($Q \vdash_{op} Q'$)
enter r into $A[s,o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s,o] = A[s,o] \cup \{r\}$ $A'[s_i,o_j] = A[s_i,o_j] \quad \forall (s_i,o_j) \neq (s,o)$
delete r from $A[s,o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s,o] = A[s,o] \setminus \{r\}$ $A'[s_i,o_j] = A[s_i,o_j] \quad \forall (s_i,o_j) \neq (s,o)$
create subject s'	$s' \notin S$	$S' = S \cup \{s'\}$ $O' = O \cup \{s'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S, o \in O$ $A'[s',o] = \emptyset \quad \forall o \in O'$ $A'[s,s'] = \emptyset \quad \forall s \in S'$
create object o'	$o' \notin O$	$S' = S$ $O' = O \cup \{o'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S, o \in O$ $A'[s,o'] = \emptyset \quad \forall s \in S'$
destroy subject s'	$s' \in S$	$S' = S \setminus \{s'\}$ $O' = O \setminus \{s'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S', o \in O'$
destroy object o'	$o' \in O$ $o' \notin S$	$S' = S$ $O' = O \setminus \{o'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S', o \in O'$

Primitive Operations of the HRU model

Access Control Models

ENTER action into A[s,o]

OPERATION (op)	CONDITIONS	NEW STATE ($Q \vdash_{op} Q'$)
enter r into $A[s,o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s,o] = A[s,o] \cup \{r\}$ $A'[s_i,o_j] = A[s_i,o_j] \quad \forall (s_i,o_j) \neq (s,o)$

<u>Condition</u>

- The specified subject and object are in the matrix

<u>New state</u>

- Set of subjects *S* is unmodified
- Set of objects *O* is unmodified
- Access matrix *A* changes only at *A[s,o]* (adding action *r*)

Access Control Models

DELETE action into A[s,o]

OPERATION (op)	CONDITIONS	NEW STATE $(Q \vdash_{op} Q')$
delete r from $A[s,o]$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $A'[s,o] = A[s,o] \setminus \{r\}$ $A'[s_i,o_j] = A[s_i,o_j] \quad \forall (s_i,o_j) \neq (s,o)$

<u>Condition</u>

- The specified subject and object are in the matrix

<u>New state</u>

- Set of subjects *S* is unmodified
- Set of objects *O* is unmodified
- Access matrix *A* changes only at *A[s,o]* (removing action *r*)

CREATE subject *s'*

OPERATION (op)	CONDITIONS	NEW STATE ($Q \vdash_{op} Q'$)
create subject s'	$s' \notin S$	$S' = S \cup \{s'\}$ $O' = O \cup \{s'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S, o \in O$ $A'[s',o] = \emptyset \quad \forall o \in O'$ $A'[s,s'] = \emptyset \quad \forall s \in S'$

<u>Condition</u>

- The specified subject is not already in *S*

<u>New state</u>

- Add *s'* into set of subjects and objects
- All entries in *A* that are not *s'* remain the same
- Add *s'* as a subject into *A* with no actions on any object
- Add *s'* as an object into *A* with no actions by any subject

Access Control Models

CREATE object *o'*

OPERATION (op)	CONDITIONS	NEW STATE ($Q \vdash_{op} Q'$)
create object o'	$o' \notin O$	$S' = S$ $O' = O \cup \{o'\}$ $A'[s, o] = A[s, o] \quad \forall s \in S, o \in O$ $A'[s, o'] = \emptyset \quad \forall s \in S'$

<u>Condition</u>

- The specified object is not already in *O*

<u>New state</u>

- Subjects remain unchanged
- Add *o'* into set of objects
- All entries in *A* that are not *o'* remain the same
- Add *o'* as an object into *A* with no actions by any subject

Access Control Models

DESTROY subject *s'*

OPERATION (op)	CONDITIONS	NEW STATE $(Q \vdash_{op} Q')$
destroy subject s'	$s' \in S$	$S' = S \setminus \{s'\}$ $O' = O \setminus \{s'\}$ $A'[s,o] = A[s,o] \quad \forall s \in S', o \in O'$

Condition

- The specified subject is in *S*

New state

- Remove *s'* from *S* to make S'
- Remove s' from *O* to make S'
- Define *A'* as all A[s,o] in A such that
  - Each s is in S'
  - Each o is in O'

Access Control Models

DESTROY object *o'*

OPERATION (op)	CONDITIONS	NEW STATE ($Q \vdash_{op} Q'$)
destroy object o'	$o' \in O$ $o' \notin S$	$S' = S$ $O' = O \setminus \{o'\}$ $A'[s, o] = A[s, o] \quad \forall s \in S', o \in O'$

<u>Condition</u>

- The specified object is in *S*
- The specified object is not in *S*

<u>New state</u>

- *S* remains unchanged
- Remove s' from *O* to make S'
- Define *A'* as all A[s,o] in A such that
    - Each s is in S'
    - Each o is in O'

Access Control Matrix

Access Control Models

Others:

- Bell-LaPadula Model
- Biba model
- Composition models
- Certificate based

Access Control Mechanisms

Typical Requirements of Access Control Mechanisms:

- Tamper proof
  - Should not be possible to alter
  - Alterations should not go undetected
- Non-bypassable
  - It must mediate all access to the system and its resources
- Confinement
  - Within a limited part of the system
  - Scattering functions over the system requires multiple levels of verification
- Limited / well-defined
  - Designed with specific purpose
  - Have the ability to easily test and verify

ACES

Proof of Concept Architecture:

ACES – Automatic Compartmentalization for Embedded Systems

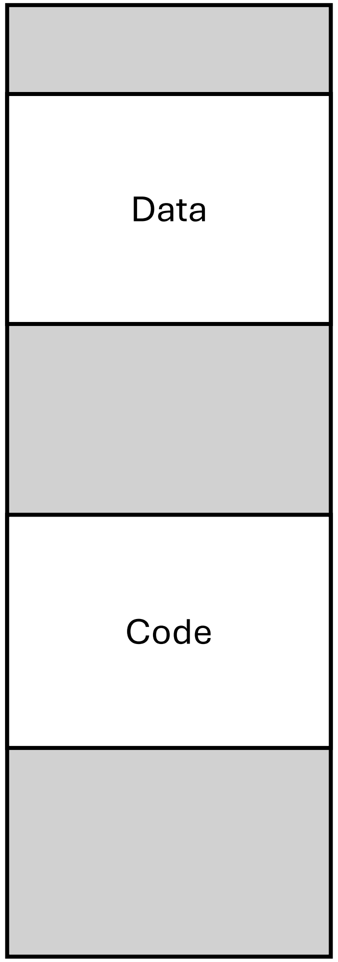
High-level idea:

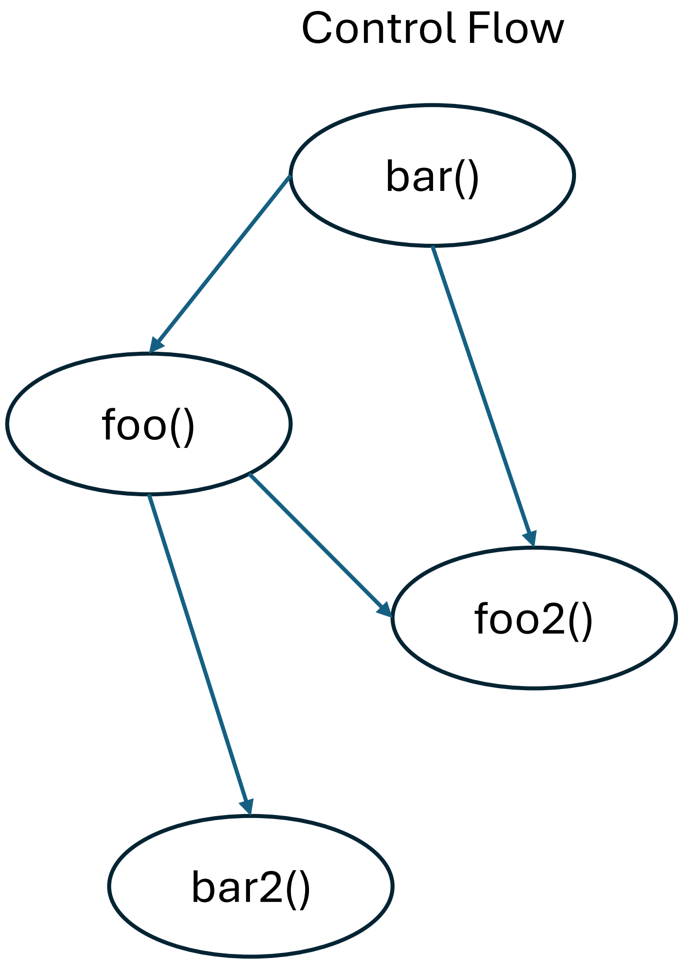
- Provide write and control flow integrity between regions of the same program
- If the application is attacked, it is contained within a compartment
- Compartments:
  - Isolated code, its accessible data, and allowed control flow transfers
  - Each instruction belongs to exactly one compartment
- Build compartments in an automated way

ACES

Simple model: data and code with a certain control flow

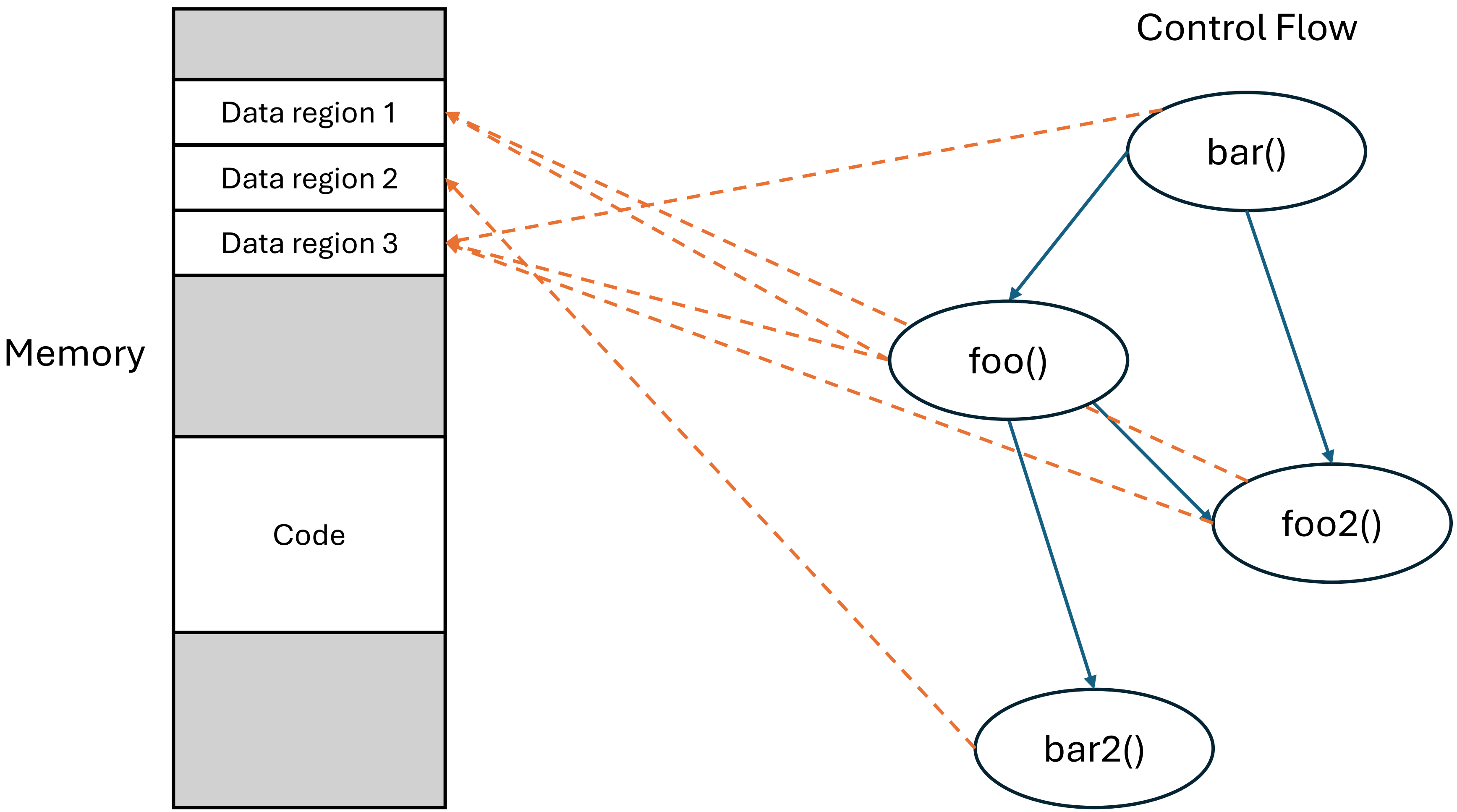
Memory





ACES

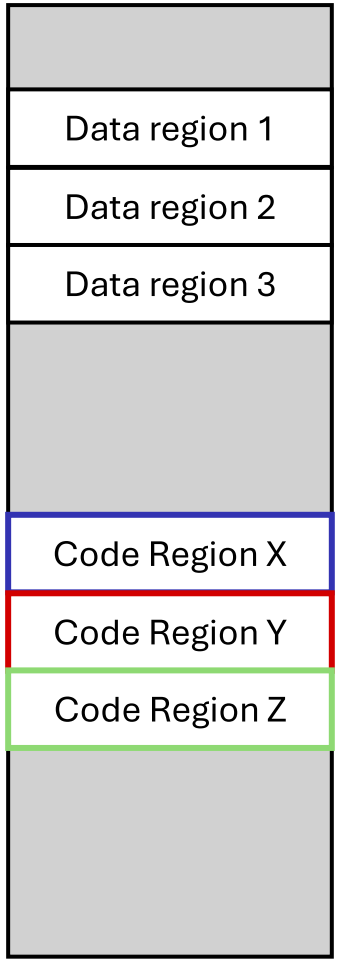
Task 1: Determine dependencies between data and code

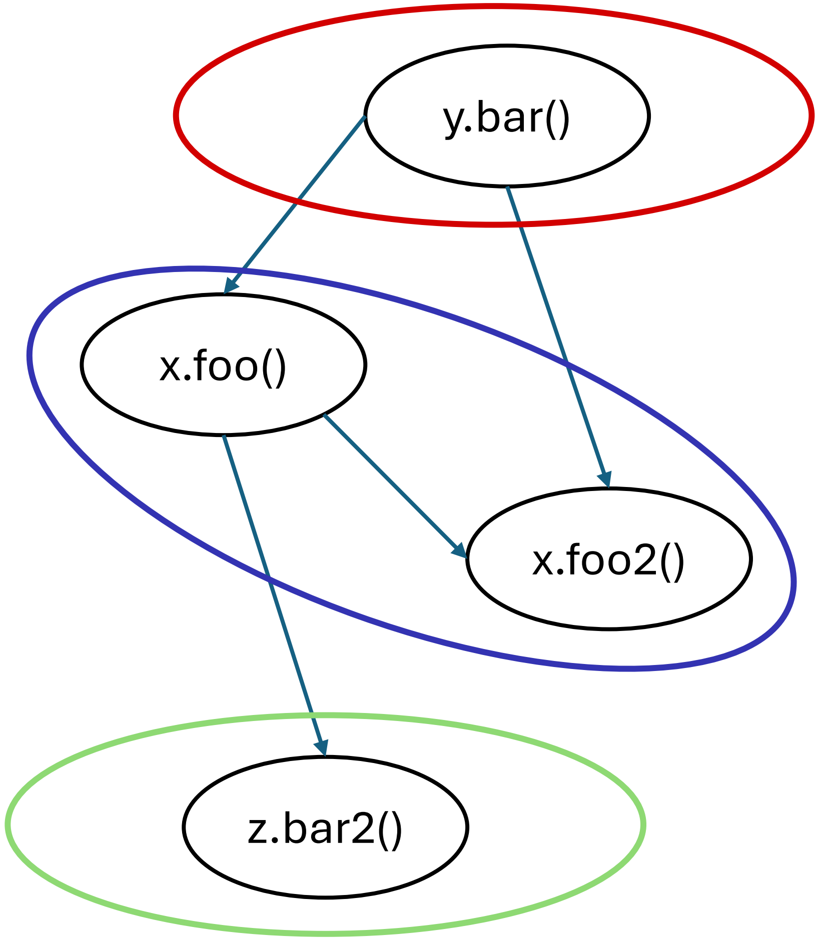


ACES

Task 2: Determine separation of code based on dependencies and flow

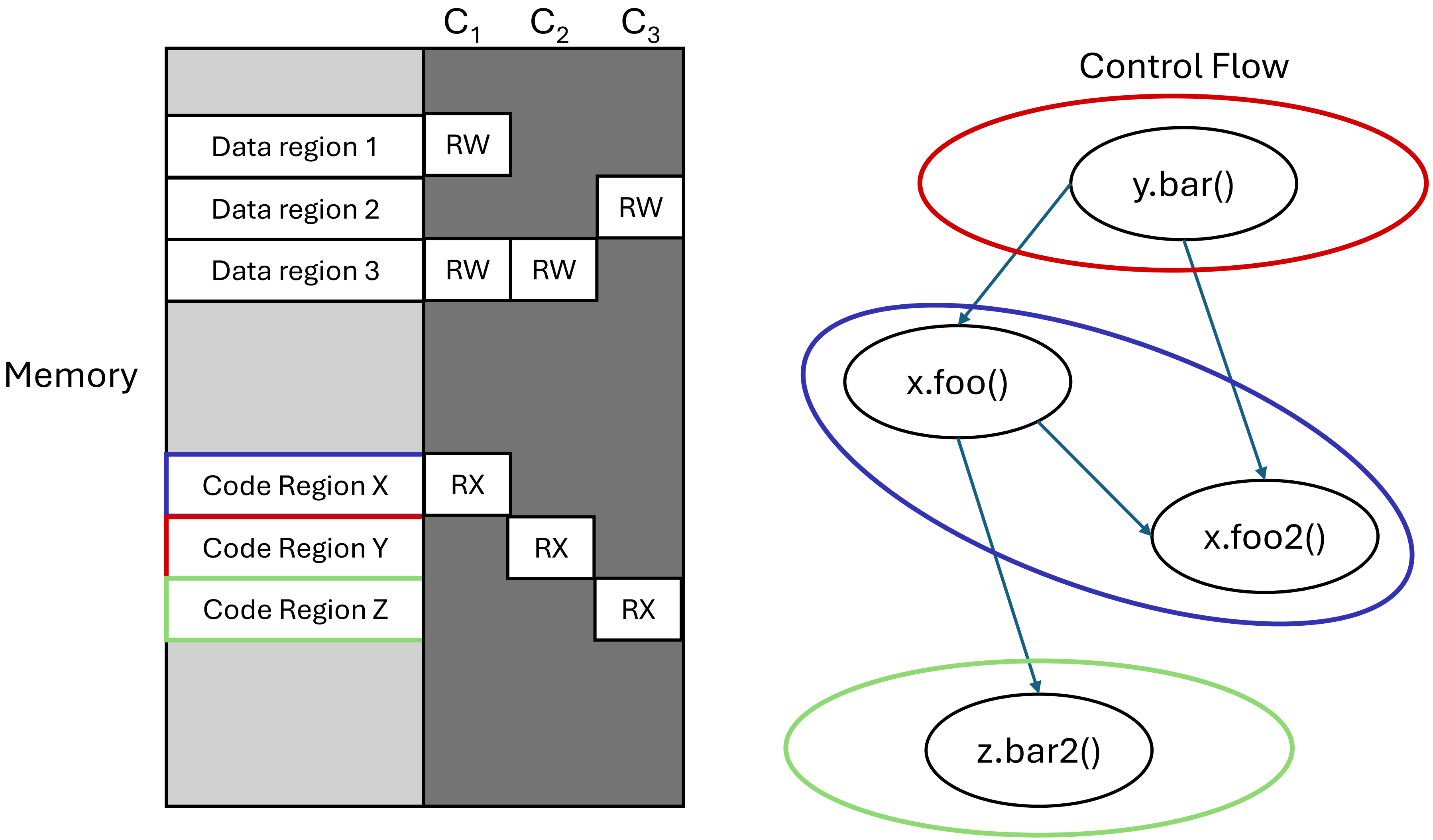
Memory





ACES

Task 3: Define compartments, set access permissions, enforce isolation



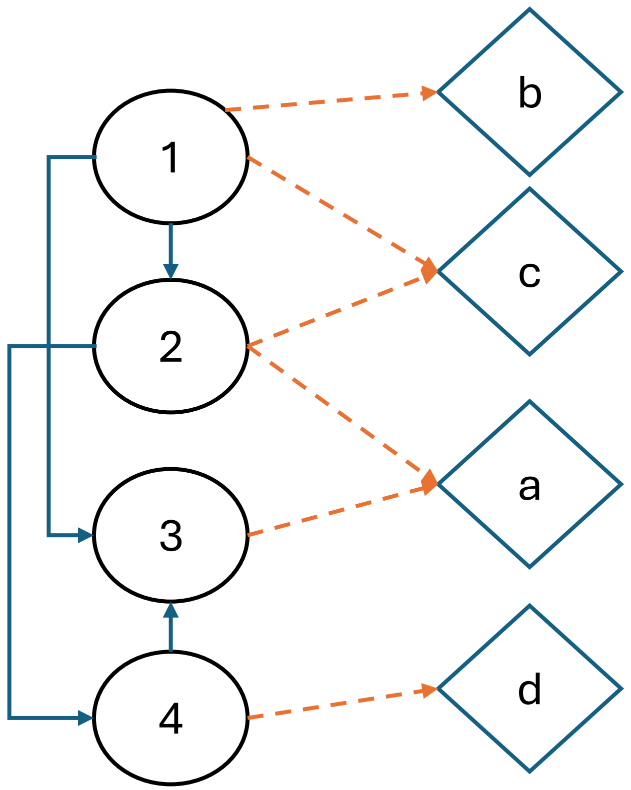
ACES

Approach:

Step 1: Program dependence graph (PDG)

- Mapping between code blocks and all dependencies
- Captures all control-flow of the application
- Dependencies between global data

Program Dependency Graph



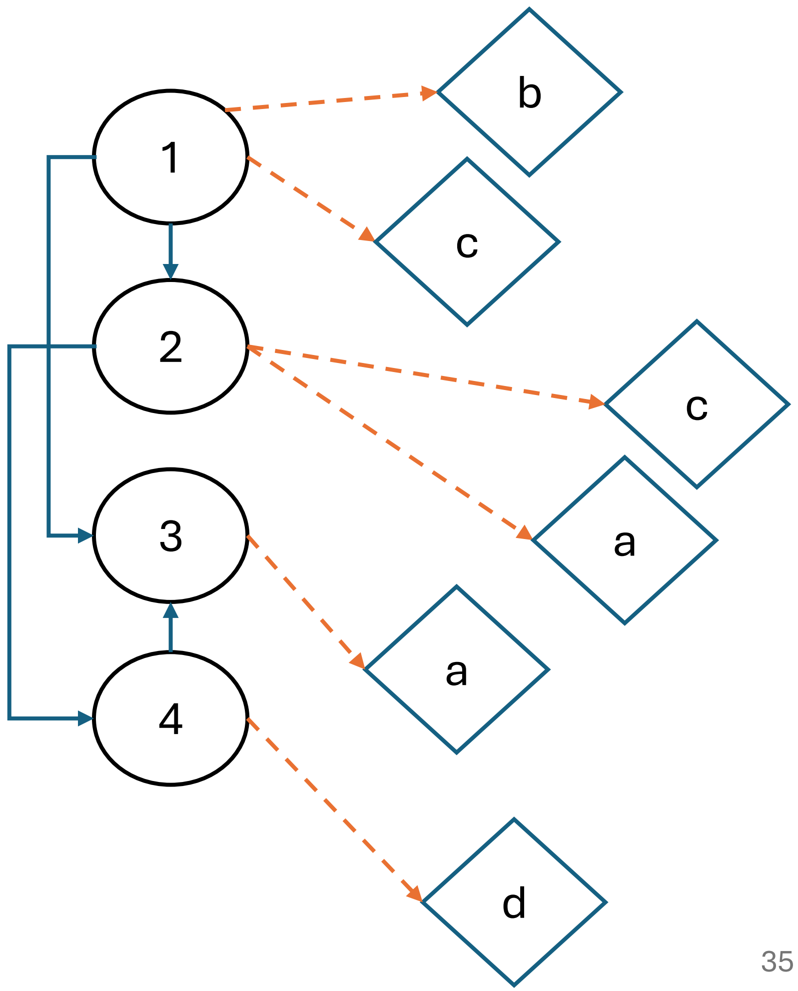
ACES

Approach:

Step 2: Create initial region graph

- Captures groupings of functions, global data
- Each vertex has a type based on what it contains
- Duplicates data vertices to separate “regions”
- Edges indicate a function in code vertex reads or writes to data a data vertex

Program Dependency Graph



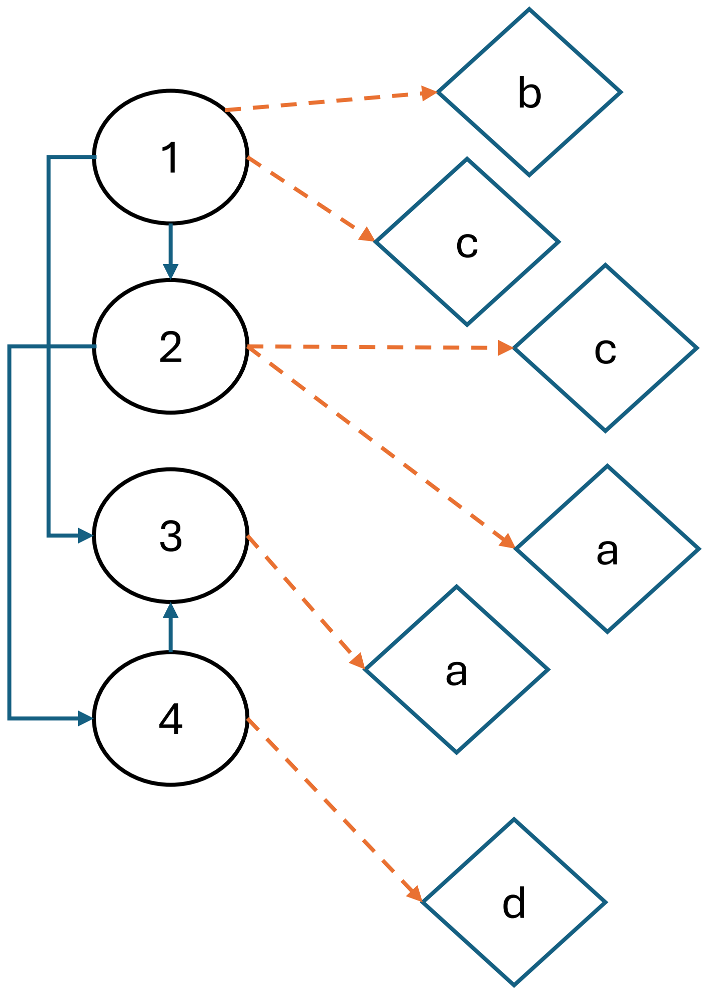
ACES

Approach:

Step 3: Defining regions

- Initial region graph may define many regions
- Perform a merging step to reduce the number or regions
- Based on compartmentalization policy
- Merged by:
  - Taking the union of their contained functions and associated edges

Program Dependency Graph



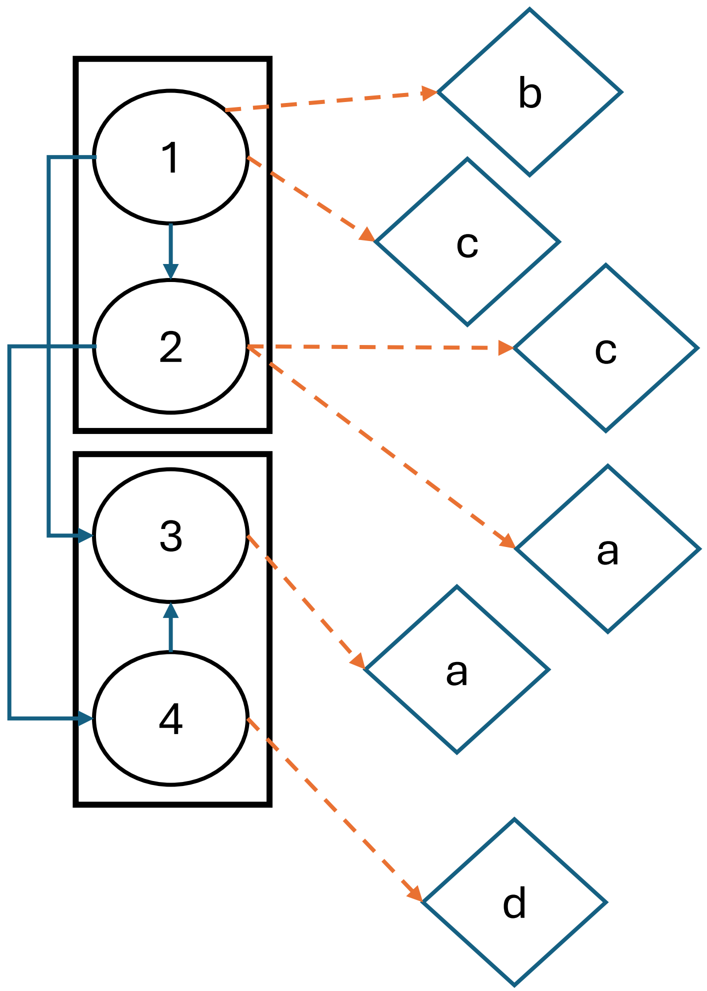
ACES

Approach:

Step 3: Defining regions

- Initial region graph may define many regions
- Perform a merging step to reduce the number or regions
- Based on compartmentalization policy
- Merged by:
  - Taking the union of their contained functions and associated edges

Program Dependency Graph



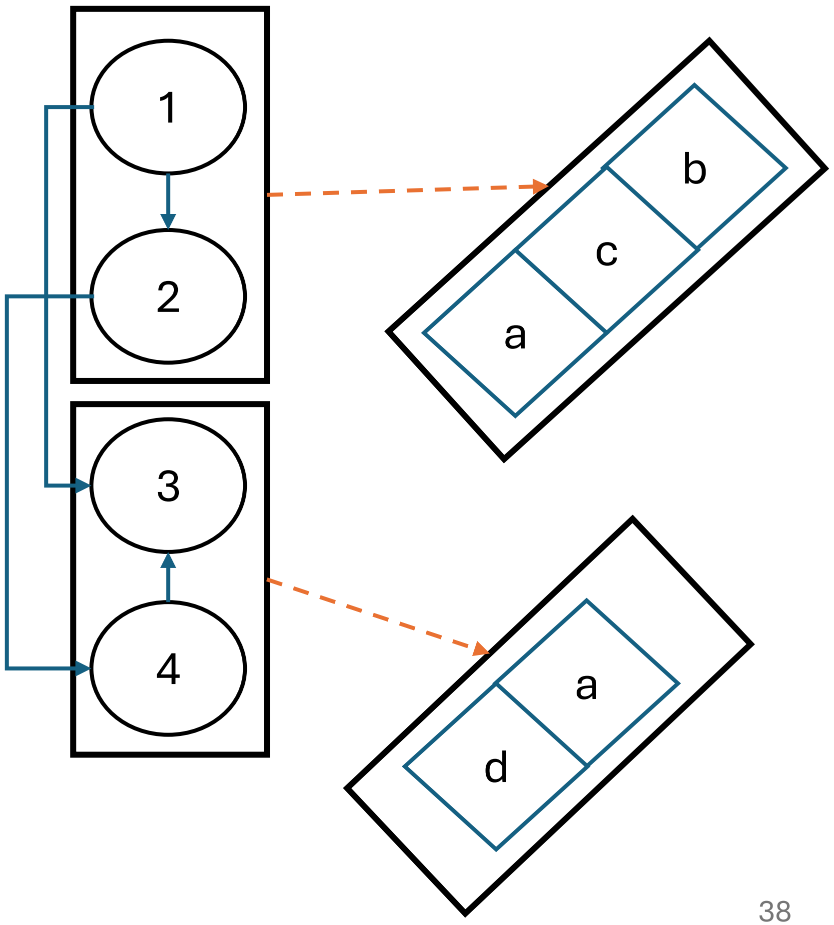
ACES

Approach:

Step 3: Defining regions

- Initial region graph may define many regions
- Perform a merging step to reduce the number or regions
- Merged by:
  - Taking the union of their contained functions and associated edges
- Based on compartmentalization policy

Program Dependency Graph

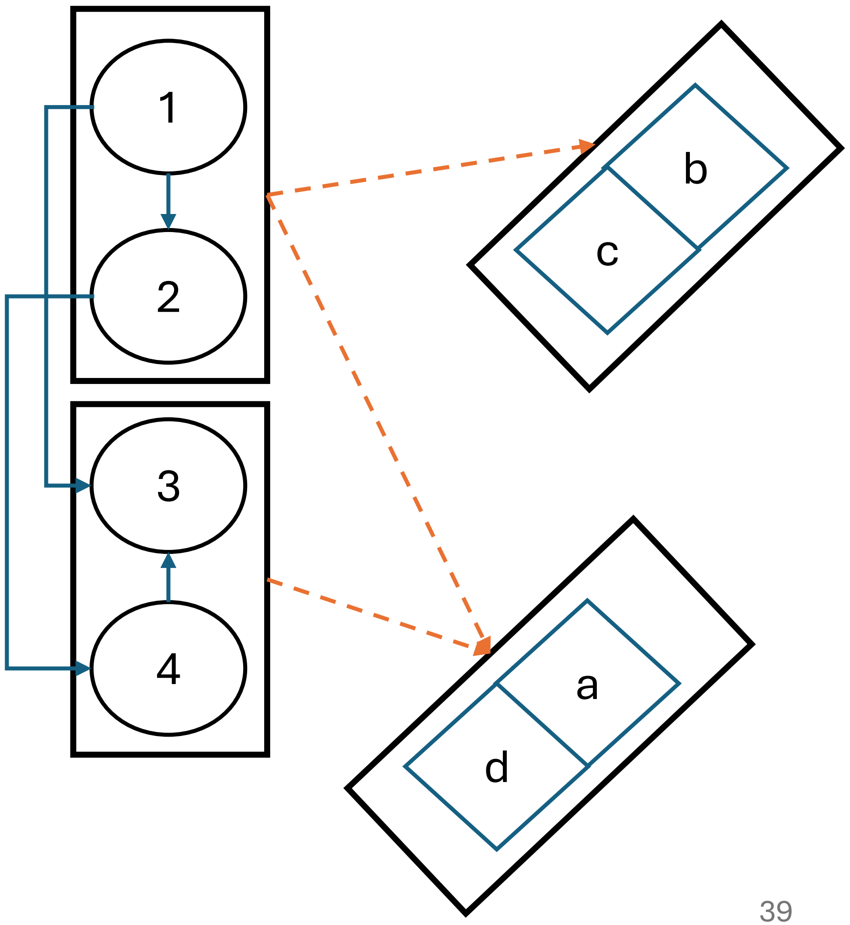


Approach:

Step 3: Defining regions

- Initial region graph may define many regions
- Perform a merging step to reduce the number or regions
- Merged by:
  - Taking the union of their contained functions and associated edges
- Based on compartmentalization policy
- When overlap, policy should specify
  - Which code has priority

Program Dependency Graph



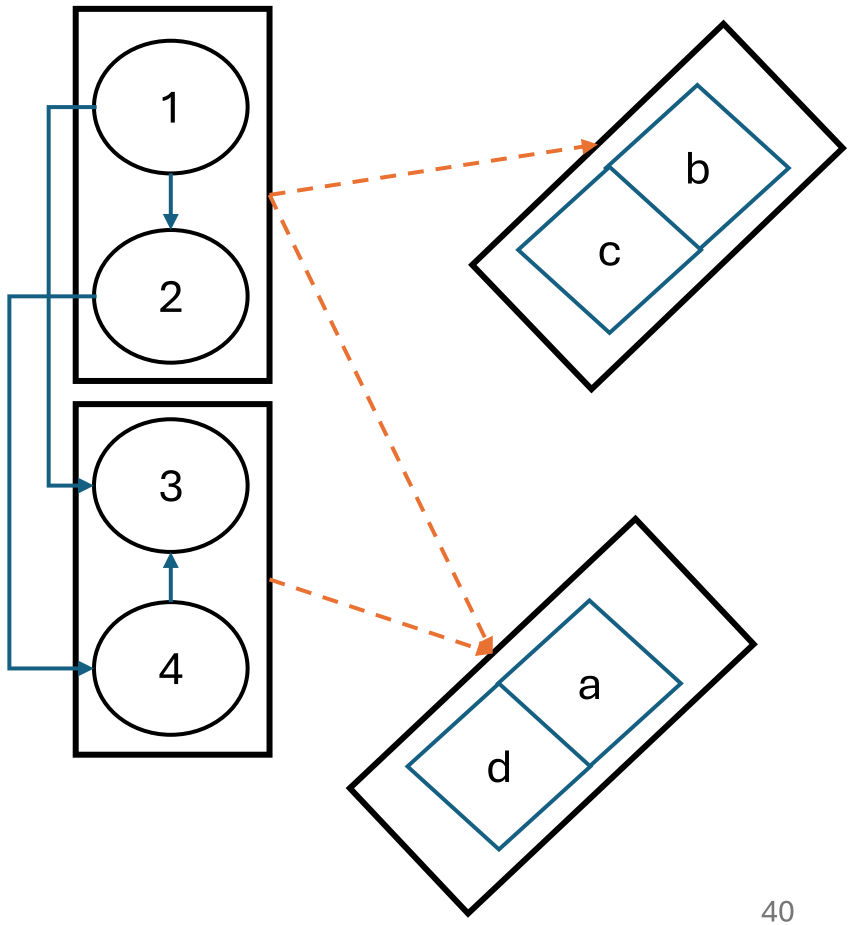
ACES

Approach:

Step 4: Lowering

- Additional merging
- Made applicable to lower end systems with limited hardware support
- This example:
  - 4 regions – typical possible for low-end MPUs

Program Dependency Graph

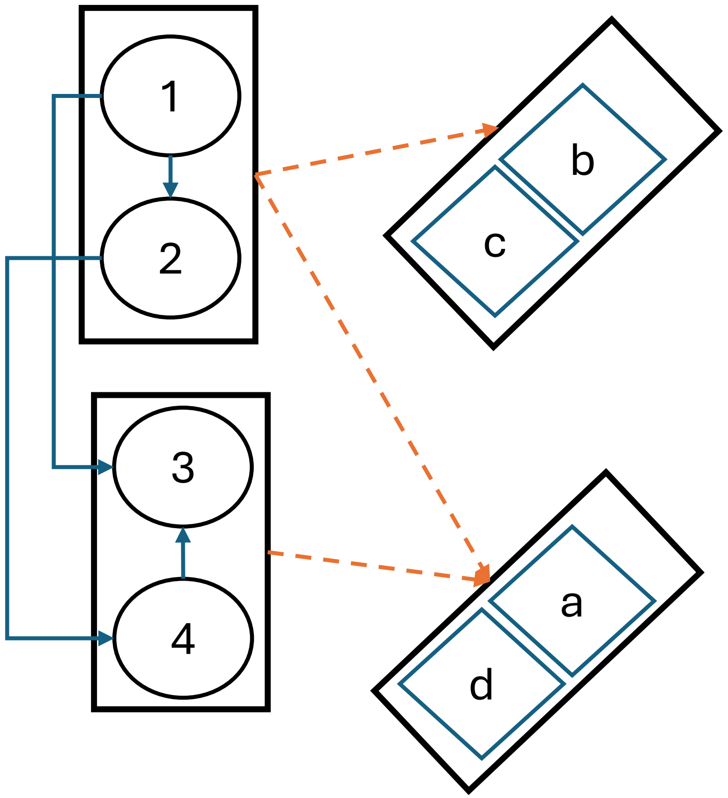


ACES

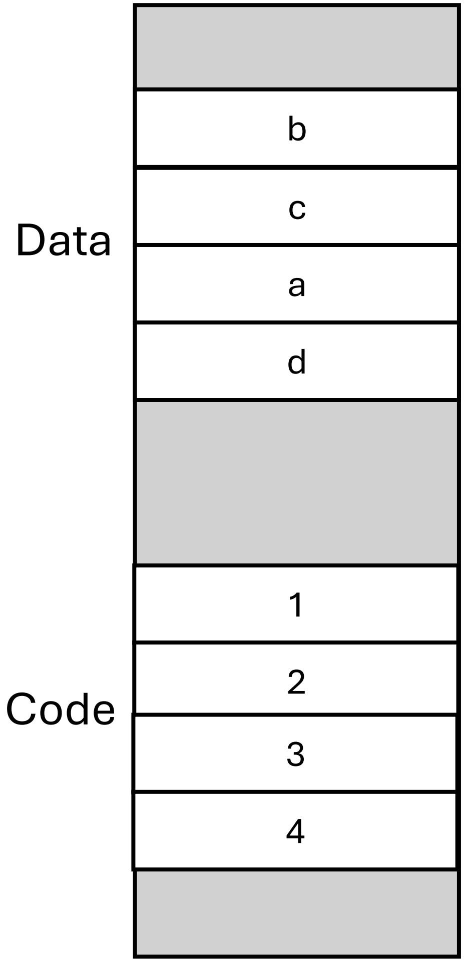
Approach:

Step 5: Configure hardware

- Use the final region graph to setup the hardware



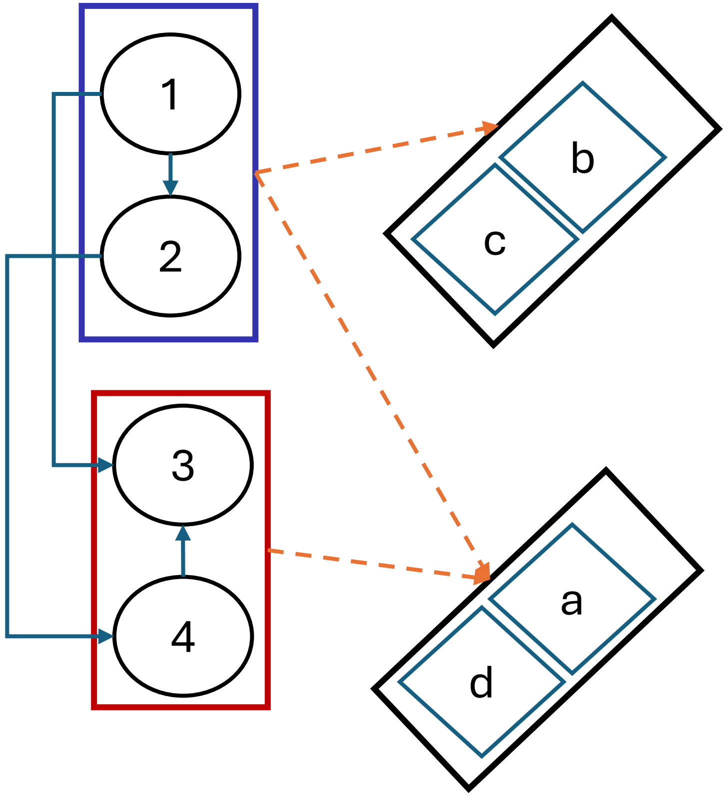
Memory Layout



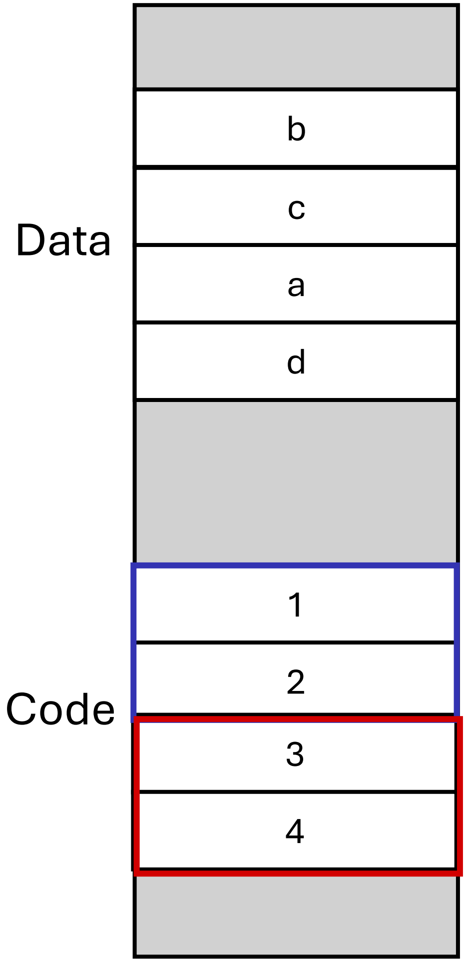
Approach:

Step 5: Configure hardware

- Use the final region graph to setup the hardware



Memory Layout

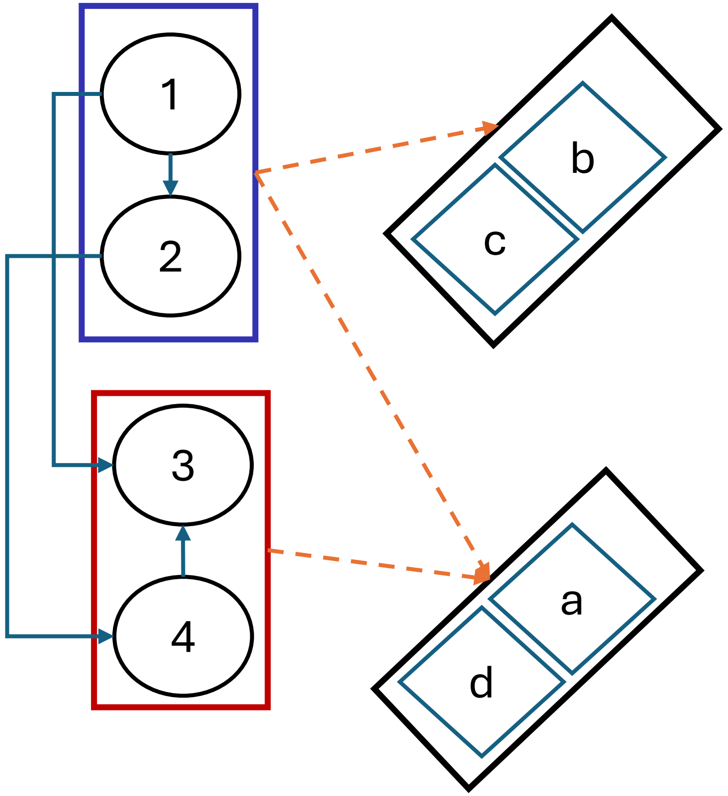


ACES

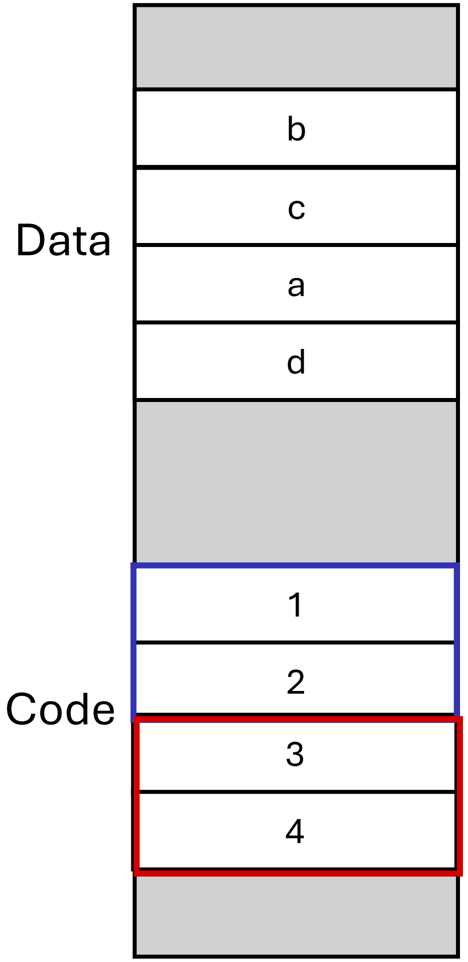
Approach:

Step 5: Configure hardware

- Use the final region graph to setup the hardware



Memory Layout

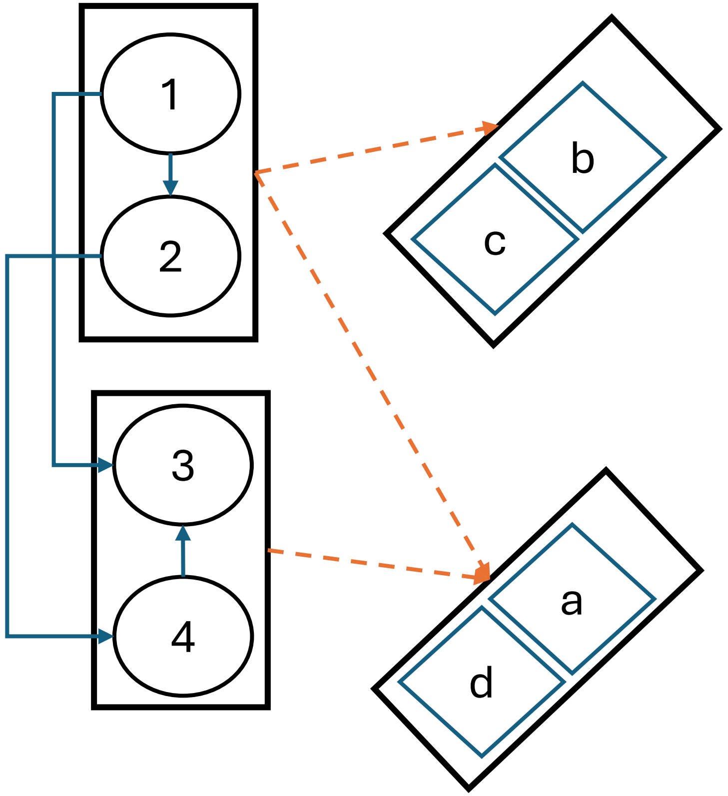


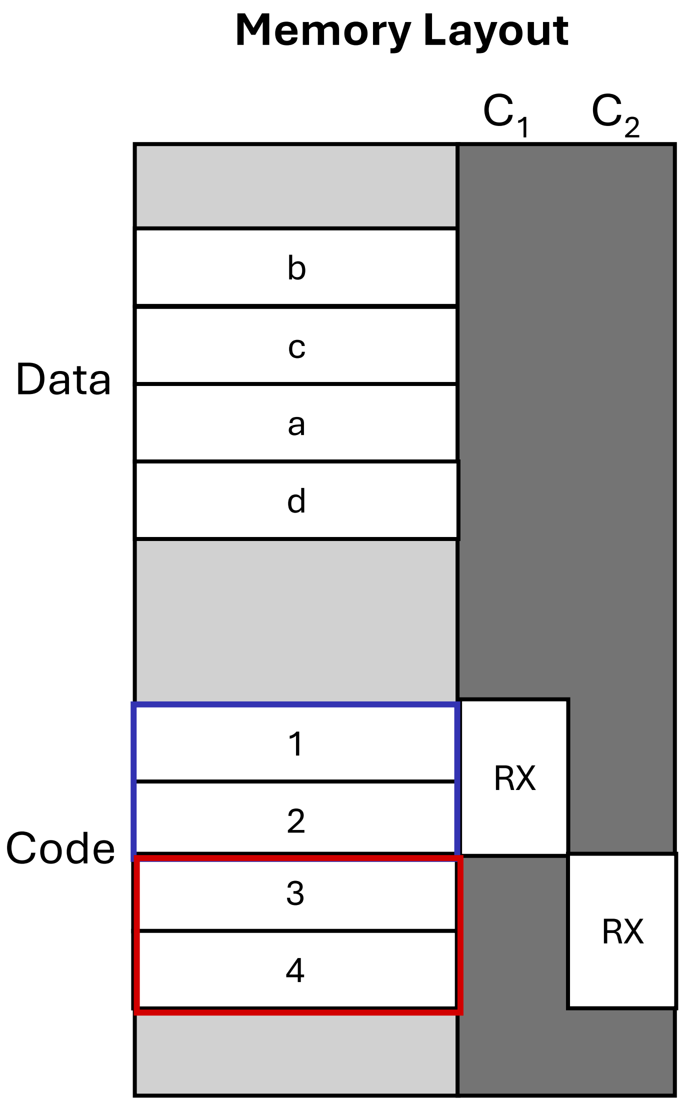
ACES

Approach:

Step 5: Configure hardware

- Use the final region graph to setup the hardware



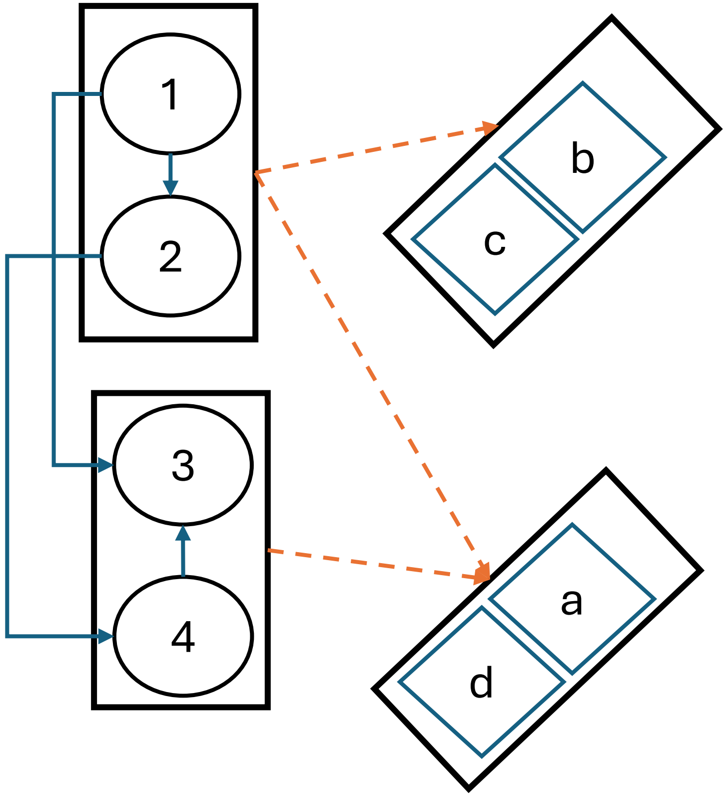


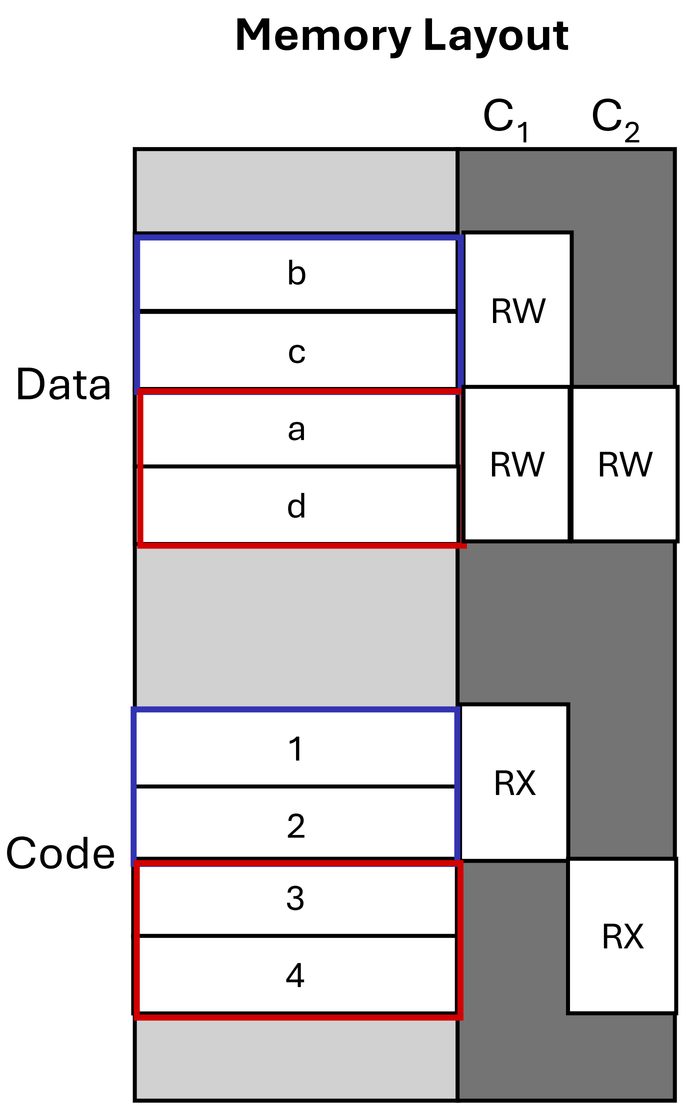
ACES

Approach:

Step 5: Configure hardware

- Use the final region graph to setup the hardware



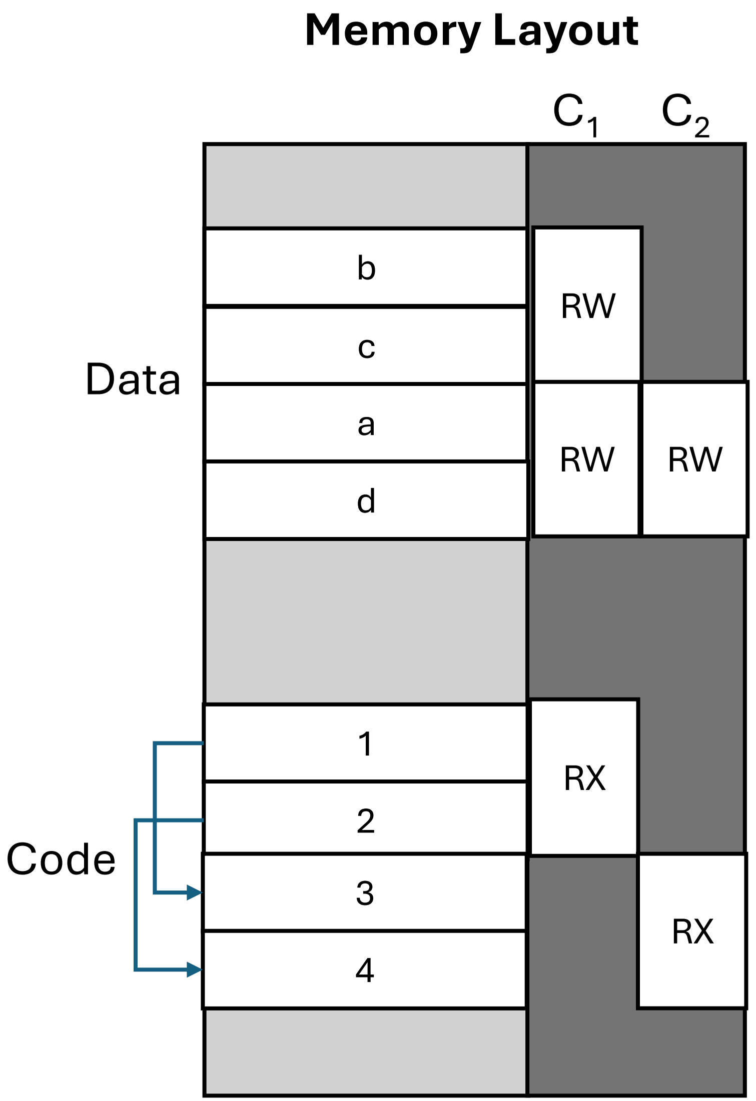


ACES

Approach:

Step 6: Instrumentation

- Controlled transitions between compartments



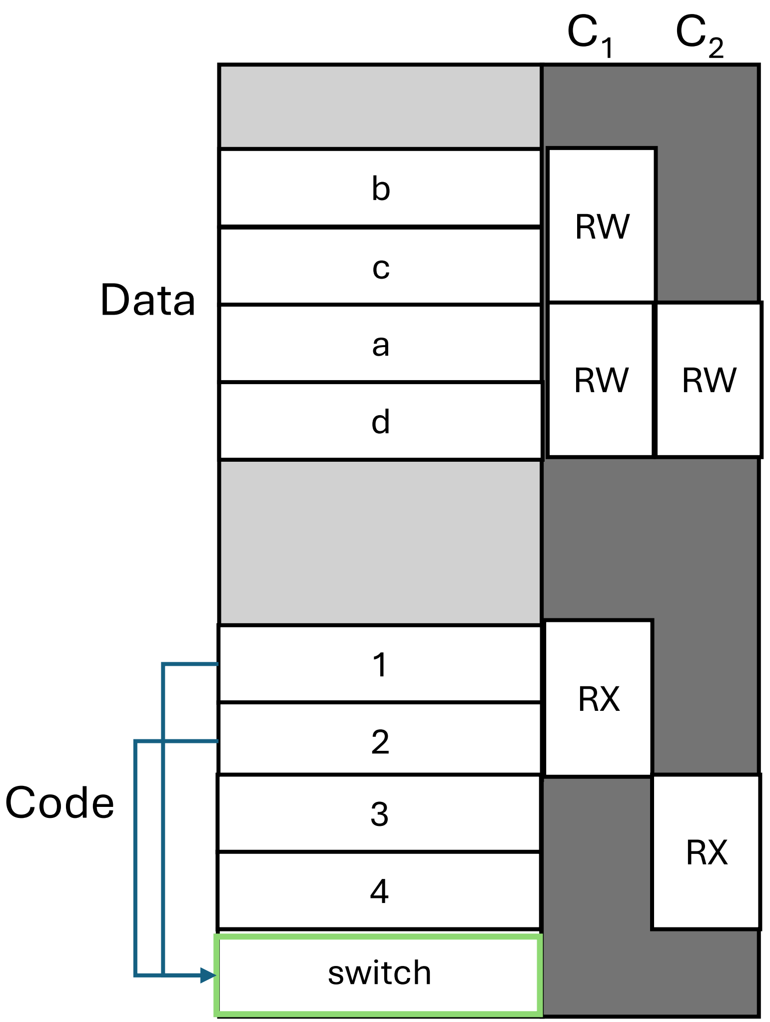
ACES

Approach:

<u>Step 6:</u> Instrumentation

- Controlled transitions between compartments
- Instrumentation modifies each function call between compartments
  - Returns invoke a compartment switch routine
  - Each switch has a list of valid targets for the transition

Memory Layout

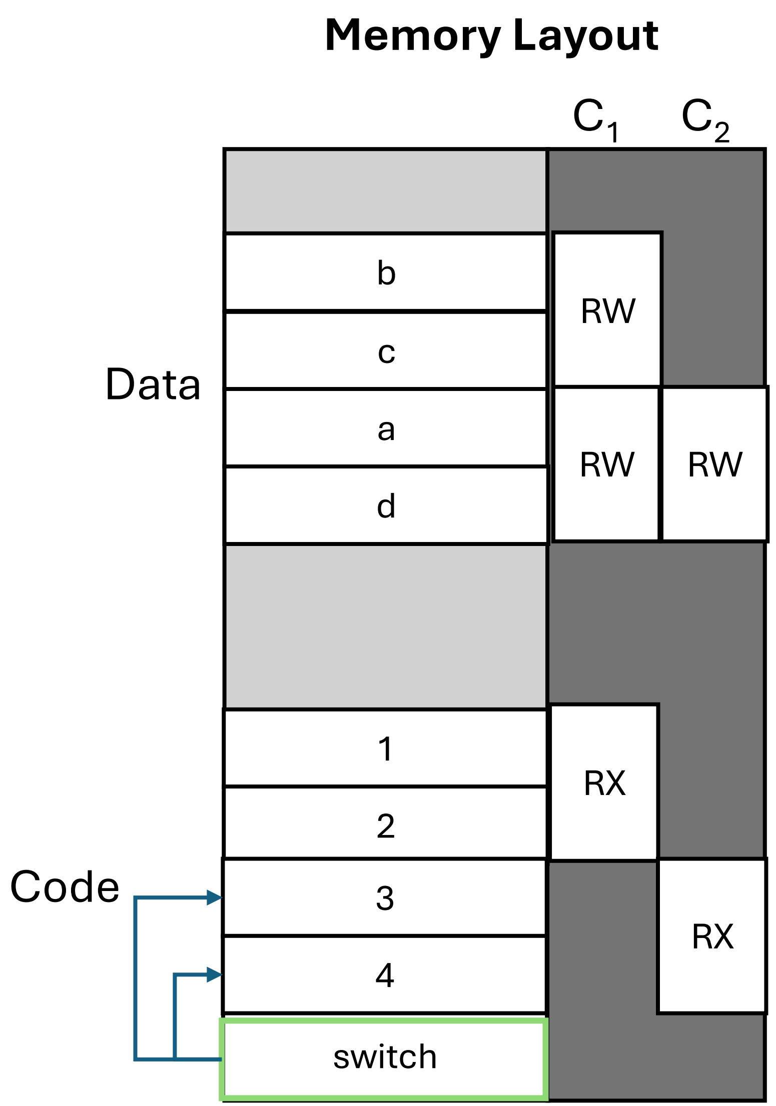


ACES

Approach:

Step 6: Instrumentation

- Controlled transitions between compartments
- Instrumentation modifies each function call between compartments
  - Returns invoke a compartment switch routine
  - Each switch has a list of valid targets for the transition
- If valid transition, performs a context switch
  - Reconfigures the MPU
  - Saves stack context



ACES

Implementation:

- Implemented in LLVM
  - Program analysis and instrumentation
- Applied to ARM devices

Limitations:

- Heavy overhead due to instrumentation
- Device-specific automation based on available MPU configurations
- Read more!

ACES

Questions?

ACES

Questions?

What is in the TCB for access control enforcement?

Does ACES follow a sandbox, safebox, or mutual-distrust compartmentalization model?

Other PoC Architectures

ACES achieves enforcement through:

- Static analysis + instrumentation
- MPU for hardware enforcement
- Automatic mutual-distrusting user-space (bare-metal) compartments
  - Can be made into sandbox or safebox based on user specified policy

Others:

- Privtrans: Safebox of user-space applications, OS-based control
- ERIM: Safebox for user-space applications, using Intel Memory Protection Keys
- CompartOS: Sandbox for user+kernel code, using CHERI

That's all for today!

Next time…

- Authentication & Attestation

Reminders:

- A3 is released